SCHOOL OF OPERATIONS RESEARCH
AND INDUSTRIAL ENGINEERING
COLLEGE OF ENGINEERING
CORNELL UNIVERSITY
ITHACA, NEW YORK 14853

TECHNICAL REPORT NO. 845

June 1989

AN EFFICIENTLY COMPUTABLE METRIC FOR COMPARING POLYGONAL SHAPES

by

Esther M. Arkin*
L. Paul Chew**
Daniel P. Huttenlocher
Klara Kedem**
Joseph S.B. Mitchell***

*This research was partially supported by NSF grants DMC 8451984 and ECSE 8857642.

**This research is supported by DARPA under ONR contract N0014–86–K–0591, NSF grant DMC–86–17355 and ONR contract N00014–86–K–0281.

***This research was partially supported by NSF grants IRI–8710858 and ECSE 8857642, and by a grant from Hughes Research Laboratories.

An Efficiently Computable Metric for Comparing Polygonal Shapes

Esther M. Arkin[*], L. Paul Chew[†], Daniel P. Huttenlocher
Klara Kedem[†] and Joseph S. B. Mitchell[‡]

Cornell University
Ithaca, NY 14853

Abstract

Model-based recognition is concerned with comparing a shape A, which is stored as a *model* for some particular object, with a shape B, which is found to exist in an image. If A and B are close to being the same shape, then a vision system should report a *match* and return a measure of how good that match is. To be useful this measure should satisfy a number of properties, including: (1) it should be a metric, (2) it should be invariant under translation, rotation, and change-of-scale, (3) it should be reasonably easy to compute, and (4) it should match our intuition (i.e., answers should be similar to those that a person might give). We develop a method for comparing polygons that has these properties. The method works for both convex and nonconvex polygons and runs in time $O(mn \log mn)$ where m is the number of vertices in one polygon and n is the number of vertices in the other. We also present some examples to show that the method produces answers that are intuitively reasonable.

[*]Partially supported by NSF grants DMC 8451984 and ECSE 8857642.

[†]Supported by DARPA under ONR contract N0014-86-K-0591, NSF grant DMC-86-17355 and ONR contract N00014-86-K-0281.

[‡]Partially supported by NSF grants IRI-8710858 and ECSE 8857642, and by a grant from Hughes Research Laboratories.

1 Introduction

A problem of both theoretical and practical importance in computer vision is that of comparing two shapes. To what extent is shape A similar to shape B? Model-based recognition is concerned with comparing a shape A, which is stored as a *model* for some particular object, with a shape B, which is found to exist in an image. If A and B are close to being of the same shape, then a vision system should report a *match* and return a measure of how good that match is. Hence, we are interested in defining and computing a *cost* function $d(A, B)$ associated with two shapes A and B that measures their similarity.

The long-term goal of this research is to develop methods of comparing arbitrary shapes in two or three dimensions. Here, we restrict our attention to polygonal shapes in the plane, with an extension to the case in which a boundary may contain circular arcs in addition to straight line segments. Our technique is designed to work with objects for which the entire boundaries are known.

Before suggesting a similarity measure to be used for comparing polygons, we examine several properties that such a measure $d(\cdot, \cdot)$ should have.

- It should be a metric.

  $d(A, B) \geq 0$ for all A and B.

  $d(A, B) = 0$ if and only if $A = B$. We expect a shape to resemble itself.

  $d(A, B) = d(B, A)$ for all A and B (Symmetry). The order of comparison should not matter.

  $d(A, B) + d(B, C) \geq d(A, C)$ for all A, B, and C (Triangle Inequality).

  The triangle inequality is necessary since without it we can have a case in which $d(A, B)$ and $d(B, C)$ are both very small, but $d(A, C)$ is very large. This is undesirable for pattern matching and visual recognition applications. If A is very similar to B and B is very similar to C, then A and C should not be too dissimilar.

- It should be invariant under translation, rotation, and change-of-scale. In other words, we want to measure *shape* alone.

- It should be reaonably easy to compute. This must hold for the measure to be of practical use.

- Most important of all, it should match our intuitive notions of shape resemblance. In other words, answers should be similar to those that a human might give. In particular, the measure should be insensitive to small perturbations (or small errors) in the data. For example, moving a vertex by a small amount or breaking a single edge into two edges should not have a large effect.

Representation of Polygons

A standard method of representing a simple polygon A is to describe its boundary by giving a (circular) list of vertices, expressing each vertex as a coordinate pair. An alternative representation of the boundary of a simple polygon A is to give the *turning function* $\Theta_A(s)$. The function $\Theta_A(s)$ measures the angle of the counterclockwise tangent as a function of the arc-length s, measured from some reference point O on A's boundary. Thus $\Theta_A(0)$ is the angle v that the tangent at the reference point O makes with some reference orientation associated with the polygon (such as the x-axis). $\Theta_A(s)$ keeps track of the turning that takes place, increasing with left-hand turns and decreasing with right-hand turns (see Figure 1). Formally, if $\kappa(s)$ is the curvature function for a curve, then $\kappa(s) = \Theta'(s)$. The curvature function $\kappa(s)$ is frequently used as a shape signature ([HT], [HW], [SS], [Wo1], [Wo2]).

Other authors have used a slightly different definition of the turning function in which $\Theta_A(0)$ is defined to be 0. Our definition, in which $\Theta_A(0)$ is the angle of the tangent line at the reference point, leads to a simple correspondence between a shift of $\Theta_A(s)$ in the θ direction and a rotation of A. This correspondence is less clear for the alternate definition.

Without loss of generality, we assume that each polygon is rescaled so that the total perimeter length is 1; hence, Θ_A is a function from $[0, 1]$ to $\Re$. For a convex polygon A, $\Theta_A(s)$ is a monotone function, starting at some value v and increasing to $v + 2\pi$. For a nonconvex polygon, $\Theta_A(s)$ may become arbitrarily large, since it accumulates the *total* amount of turn, which can grow as a polygon "spirals" inward. Although $\Theta_A(s)$ may become very large over the interval $s \in [0, 1]$, in

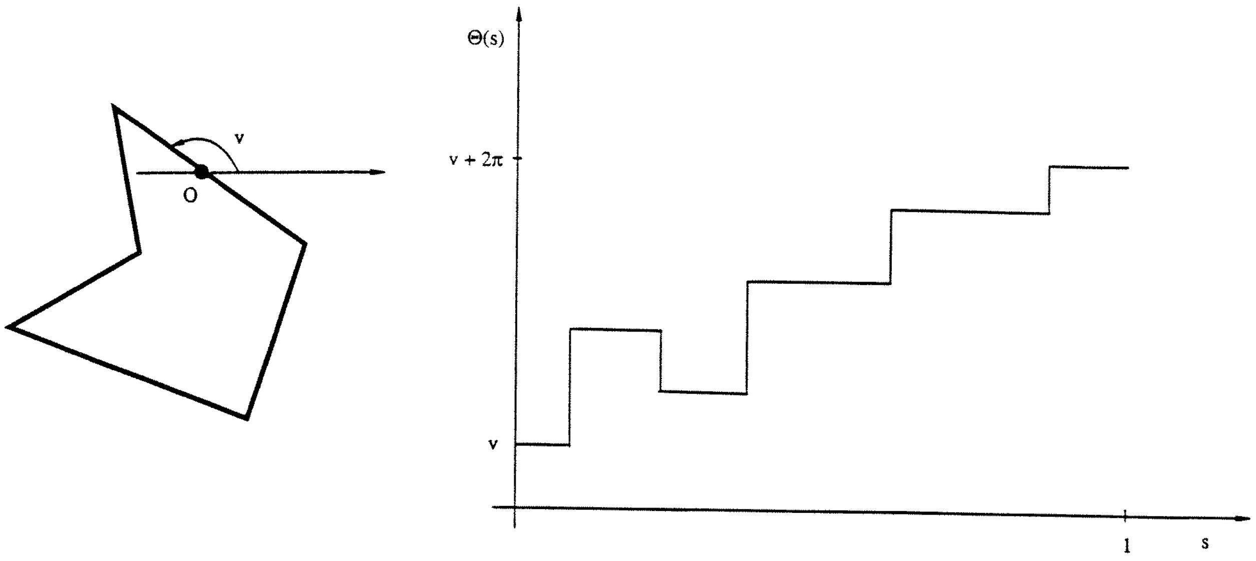

Figure 1: Defining the turn function $\Theta(s)$.

order for the function to represent a simple closed curve, we must have $\Theta_A(1) = \Theta_A(0) + 2\pi$ (assuming that the origin O is placed at a differentiable point along the curve).

The domain of $\Theta_A(s)$ can be extended to the entire real line in a natural way by allowing angles to continue to accumulate as we continue around the perimeter of the polygon A. Thus, for a simple closed polygon, the value of $\Theta_A(s+1)$ is $\Theta_A(s) + 2\pi$ for all s. Note that the function $\Theta_A(s)$ is well-defined even for arbitrary (not necessarily simple or closed or polygonal) paths A in the plane. When the path is polygonal, the turning function is piecewise-constant, with jump points corresponding to the vertices of A.

Representation of planar curves (and, in particular, polygons) in terms of some function of arc length has been used by a number of other researchers (e.g., [OW], [SS], etc.) in computational geometry and computer vision. We use this representation to compute a distance function for comparing two simple polygons (A and B) by looking at natural notions of distances between the turning functions $\Theta_A(s)$ and $\Theta_B(s)$.

The function $\Theta_A(s)$ has several properties which make it especially suitable for

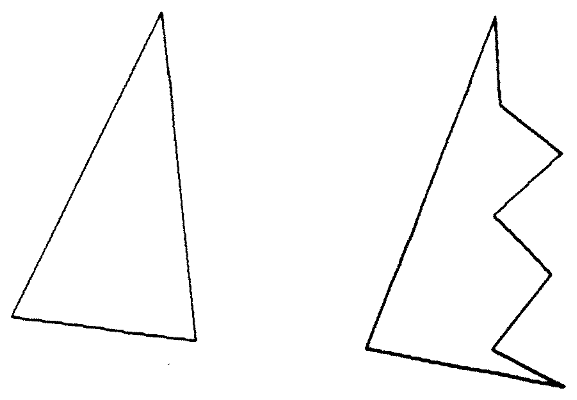

Figure 2: Non-uniform noise is problematic for this distance function.

our purposes. It is piecewise-constant for polygons (and polygonal paths), making computations particularly easy and fast. By definition, the function $\Theta_A(s)$ is invariant under translation and scaling of the polygon A. Rotation of A corresponds to a simple shift of $\Theta_A(s)$ in the θ direction. Note also that changing the location of the origin O by an amount $t \in [0,1]$ along the perimeter of polygon A corresponds to a horizontal shift of the function $\Theta_A(s)$ and is simple to compute (the new turning function is given by $\Theta_A(s+t)$).

We formally define the distance function between two polygons A and B as the L_p distance between their two turning functions $\Theta_A(s)$ and $\Theta_B(s)$, minimized with respect to vertical and horizontal shifts of the turning functions (in other words, we minimize with respect to rotation and choice of reference points). For $p = 2$ we show that this distance function can be computed efficiently in $O(n^2 \log n)$ time for polygons with n vertices.

One possible drawback of our distance function is that it may be unstable under certain kinds of noise, in particular, *non-uniform* noise. For example, suppose we have a triangle with one very wiggly side so that the wiggly side accounts for most of the length of the boundary of the polygon. Comparing it with a triangle, we will get a very bad (in fact, arbitrarily bad) match. See Figure 2. Fortunately, in many computer vision applications it is reasonable to assume that the noise is roughly uniformly distributed over the sides of the polygon, in which case the similarity measure we define performs nicely. See Section 4 for examples.

Schwartz and Sharir [SS] have defined a notion of distance similar to ours.

However, they compute an approximation based on discretizing the turning functions of the two shapes into many equally spaced points; thus, the quality of the approximation depends on the number of points chosen. Our approach, on the other hand, is to examine the combinatorial complexity of computing the *exact* metric function between two polygon boundaries, using only the original vertices. Thus, our method runs in time $O(n^2 \log n)$ where n is the total number of polygon vertices, while their method computes an approximate distance in time $O(k \log k)$, where $k >> n$ is the total number of interpolation points used. Furthermore, [SS] suggest "convexifying" non-convex polygons in order to compare them, as their method does not apply to non-convex polygons. Our algorithm applies to both convex and non-convex simple polygons (and even to some types of non-simple polygons).

The remainder of this paper is organized as follows. In Section 2 we give a formal definition of the distance between two polygons based on their turning functions. We prove that this function is a metric and show some of its properties. These results are used in Section 3 to develop an $O(n^3)$ algorithm for computing the distance between two polygons, where n is the total number of vertices; we then refine this algorithm to obtain an $O(n^2 \log n)$ running time. Section 4 contains examples of the distance function computed for several polygons using an implementation of our method. Section 5 is a summary and discussion of extensions and further research.

2 A Polygon Distance Function

Consider two polygons A and B and their associated turning functions $\Theta_A(s)$ and $\Theta_B(s)$. The degree to which A and B are similar can be measured by taking the distance between the functions $\Theta_A(s)$ and $\Theta_B(s)$ according to our favorite metric on function spaces (e.g., L_p metrics) . Define the L_p distance between A and B as

$$\delta_p(A, B) = ||\Theta_A - \Theta_B||_p = \left(\int_0^1 |\Theta_A(s) - \Theta_B(s)|^p \, ds \right)^{\frac{1}{p}},$$

where $|| \cdot ||_p$ denotes the L_p norm.

δ_p has some undesirable properties: it is sensitive to both rotation of polygon A (or B) and choice of reference point on the boundary of A (or B). Since rotation and choice of reference point are arbitrary, it makes more sense to consider the distance to be the minimum over all such choices. If we shift the reference point O along A's boundary by an amount t, then the new turning function is given by $\Theta_A(s+t)$. If we rotate A by angle θ then the new function is given by $\Theta_A(s)+\theta$. Thus, we want to find the minimum over all such shifts t and rotations θ (i.e., over all horizontal and vertical shifts of $\Theta_A(s)$). In other words, we want to solve for

$$\begin{aligned} d_p(A,B) &= \left(\min_{\theta\in\Re\ t\in[0,1]} \int_0^1 |\Theta_A(s+t) - \Theta_B(s) + \theta|^p \, ds \right)^{\frac{1}{p}} \\ &= \left(\min_{\theta\in\Re\ t\in[0,1]} D_p^{A,B}(t,\theta) \right)^{\frac{1}{p}}, \end{aligned}$$

where

$$D_p^{A,B}(t,\theta) = \int_0^1 |\Theta_A(s+t) - \Theta_B(s) + \theta|^p \, ds$$

.

Lemma 1 *$d_p(A,B)$ is a metric for all $p > 0$.*

Proof. This follows from the fact that $||\cdot||_p$ is a metric for all $p > 0$. ∎

Lemma 2 *For any fixed value of t, and for any $p \geq 1$, $D_p^{A,B}(t,\theta)$ is a convex function of θ.*

Proof. For fixed s and fixed t, the function $F(\theta) = |\Theta_A(s+t) - \Theta_B(s) + \theta|^p$ is clearly a convex function by the convexity of $G(y) = |y|^p$ (for $p \geq 1$), and integrating a convex function maintains convexity. ∎

In particular, using the L_2 metric, $D_2^{A,B}(t,\theta)$ is a quadratic function of θ for any fixed value of t. This holds for any simple closed shapes, but it is especially easy to see for polygons.

We assume from now on that A and B are polygons. Then, for a fixed t, the integral $\int_0^1 |\Theta_A(s+t) - \Theta_B(s) + \theta|^2 \, ds$ can be computed by adding up the value of the integral within each *strip* defined by a consecutive pair of discontinuities in $\Theta_A(s)$ and $\Theta_B(s)$ (see Figure 3). The integral within a strip is trivially computed

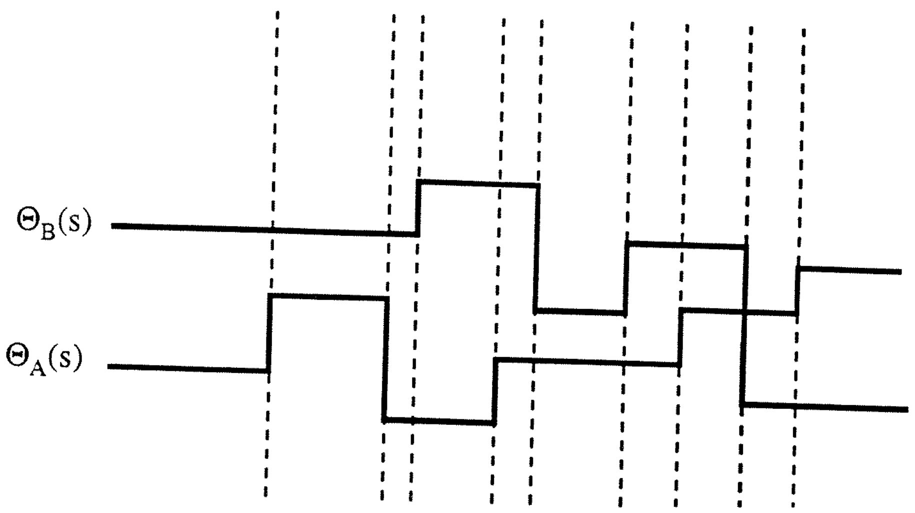

Figure 3: The rectangular strips formed by the functions $\Theta_A(s)$ and $\Theta_B(s)$.

as the width of the strip times the square of the difference $|\Theta_A(s+t) - \Theta_B(s)|$ (which is constant within each strip). Note that if m and n are the numbers of vertices in A and B, respectively, then there are $m+n$ strips and that as θ changes, the value of the integral for each strip is a quadratic function of θ.

In order to compute $d_2(A,B)$, we must minimize $D_2^{A,B}(t,\theta)$ over all t and θ. We begin by finding the optimal θ for any fixed value of t. To simplify notation in the following discussion, we use $f(s) = \Theta_A(s)$, $g(s) = \Theta_B(s)$, and $h(t,\theta) = D_2^{A,B}(t,\theta)$.

Lemma 3 *Let $h(t,\theta) = \int_0^1 (f(s+t) - g(s) + \theta)^2 \, ds$. Then, in order to minimize $h(t,\theta)$, the best value of θ is given by*

$$\begin{aligned} \theta^*(t) &= \int_0^1 (g(s) - f(s+t)) \, ds \\ &= \alpha - 2\pi t, \end{aligned}$$

where $\alpha = \int_0^1 g(s) \, ds - \int_0^1 f(s) \, ds$.

Proof.

$$\begin{aligned} \frac{\partial h(t,\theta)}{\partial \theta} &= \int_0^1 (2\theta + 2f(s+t) - 2g(s)) \, ds \\ &= 2\theta + 2\int_0^1 (f(s+t) - g(s)) \, ds. \end{aligned}$$

Lemma 2 assures us that the minimum occurs when we set this quantity equal to zero and solve for θ. Thus,

$$\theta^*(t) = \int_0^1 (g(s) - f(s+t))\, ds.$$

Now,

$$\begin{aligned}
\int_0^1 f(s+t)\, ds &= \int_t^{t+1} f(s)\, ds \\
&= \int_t^1 f(s)\, ds + \int_1^{t+1} [f(s-1) + 2\pi]\, ds \\
&= \int_t^1 f(s)\, ds + \int_0^t f(s)\, ds + 2\pi t \\
&= 2\pi t + \int_0^1 f(s)\, ds.
\end{aligned}$$

Thus,

$$\begin{aligned}
\theta^*(t) &= \int_0^1 g(s)\, ds - 2\pi t - \int_0^1 f(s)\, ds \\
&= \alpha - 2\pi t.
\end{aligned}$$

■

Substituting the expression for $\theta^*(t)$ in $d_2(A, B)$ we are left with a one-variable minimization problem:

$$\begin{aligned}
d_2(A, B) &= \min_{t \in [0,1]} h(t, \theta^*(t)) \\
&= \left\{ \min_{t \in [0,1]} \left[\int_0^1 [f(s+t) - g(s)]^2\, ds - [\theta^*(t)]^2 \right] \right\}^{\frac{1}{2}}.
\end{aligned}$$

3 Algorithmic Details

In this section we show that the distance function achieves its minimum at one of mn discrete points on $[0, 1]$, which we call *critical events*. Recall that in the process of finding $d_2(A,B)$ we have to shift the function $f(s)$ to $f(s+t)$ for $t \in [0,1]$. During this shifting operation the breakpoints of f collide with the breakpoints of g. We define a *critical event* as a value of t where a breakpoint of f collides with a breakpoint of g. Clearly there are mn such critical events for m breakpoints in f and n breakpoints in g.

Using the fact that the minimum is obtained at a critical event, we present a basic algorithm for computing $d_2(A,B)$ that runs in $O(n^3)$ time for two n-vertex polygons (and time $O(mn(m+n))$ for an m vertex polygon and an n vertex polygon). We then describe how to modify the basic method to improve the runtime to $O(n^2 \log n)$ (or $O(mn \log mn)$ for unequal numbers of vertices).

Recall that $d_2(A,B) = \min_{t,\theta}(h(t,\theta))^{\frac{1}{2}}$, where $h(t,\theta) = D_2^{A,B}(t,\theta)$. We prove that $h(t,\theta)$ has properties that lead to efficient algorithms for computing our polygon metric.

Lemma 4 *If $f(\cdot)$ and $g(\cdot)$ are two piecewise-constant functions with m and n breakpoints respectively, then for constant θ,*

$$h(t,\theta) = \int_0^1 (f(s+t) - g(s) + \theta)^2 \, ds$$

is piecewise linear as a function of t, with mn breakpoints.

Proof. We give a geometric proof. First recall that for a given value of t the discontinuities in f and g define a set of $m+n$ rectangular strips (see Figure 3). The value of $h(t,\theta)$ is simply the sum over all these strips of the width of a strip times the square of its height. Except at critical events, as f is shifted the width of each strip changes, but the height remains constant. Each changing rectangle contributes to changes in $h(t,\theta)$. If t is the amount of shift, then for a *shrinking* rectangle, the change is $(-t)$ times the square of the height; for a *growing* rectangle the change is $(+t)$ times the square of the height. Since the heights are constant, the change in $h(t,\theta)$ is a sum of linear terms and is therefore linear. Breakpoints in $h(t,\theta)$ clearly occur at each of the mn critical events where a discontinuity of f is aligned with a discontinuity of g. ∎

This result leads to a straightforward algorithm for computing $d_2(A,B)$. Let (t^*,θ^*) be the location of the minimum value of $h(t,\theta)$. By the preceeding lemma, $h(t,\theta^*)$ is linear as a function of t; thus, t^* must be at one of the mn breakpoints of h. In other words, we can find the minimum by checking just the values at breakpoints. These observations lead to the following result.

Corollary 5 *The distance $d_2(A, B)$ between two polygons A and B (with m and n vertices) can be computed exactly in time $O(mn(m+n))$.*

Proof. We prove the theorem by describing the algorithm. Given values for both t and θ, $h(t, \theta)$ can be computed in $O(m+n)$ time by adding the contributions of the $m+n$ rectangular strips between f and g. Let $c_0, c_1, \ldots, c_{mn}$ be the critical events that occur as f is shifted by t. By the preceeding observations, the minimum occurs when t equals one of $c_0, c_1, \ldots, c_{mn}$. Since the best θ value for a given t can be found in constant time (Lemma 3), we simply compute $h(t, \theta^*(t))$ in $O(m+n)$ time for each of these critical events, find the minimum, and take its square root to get $d_2(A, B)$. ∎

Refinement of the Algorithm

Theorem 6 *The distance $d_2(A, B)$ between two polygons A and B (with m and n vertices) can be computed exactly in time $O(mn \log mn)$.*

Proof. We prove the theorem by describing the algorithm. The basic idea is the same as the previous algorithm: we compute $h(t, \theta^*(t))$ for each of the critical values of t. First, note that we only need to compute the critical values of $h(t, 0)$, since by Lemma 3 the effect due to $\theta^*(t)$ can be calculated separately. Second, we observe that $h(t, 0)$ changes in a very constrained fashion. As a matter of fact, by keeping track of a small set of values we can easily determine how the function $h(t, 0)$ changes at each critical event.

The values we keep track of are based on the rectangular strips that appear between the two functions $f(s)$ and $g(s)$. (To simplify notation, we use f and g in place of Θ_A and Θ_B, respectively.) Recall that $g(s)$ is fixed in place and that $f(s)$ is shifted by t. For a given value of t, the discontinuities in $f(s+t)$ and $g(s)$ define a set of rectangular strips, as was illustrated in Figure 3. Each rectangular strip has f at the top and g at the bottom or vice-versa. The sides of a strip are determined by discontinuities in f and g.

For the purposes of the algorithm, we separate the strips into four groups based on the discontinuities at the sides of the strips: R_{ff} for those with f on both sides; R_{gg} for those with g on both sides; R_{fg} for those with f on the left and g on the right; and R_{gf} for those with g on the left and f on the right. The sets R_{fg} and R_{gf} are particularly important, as these are the strips whose widths change as t changes (as f is shifted). Thus, these strips affect the slope of $h(t, 0)$.

We keep track of two quantities: H_{fg} and H_{gf}. H_{fg} is the sum of the squares of the heights of all the strips in R_{fg}, and H_{gf} is the sum of the squares of the heights of all the strips in R_{gf}.

The algorithm is based on the observation that for values of t between two critical events the slope of $h(t, 0)$ is $H_{gf} - H_{fg}$. This follows from the fact that, as f is shifted by t, R_{fg} is the set of all strips that increase in width by t, and R_{gf} is the set of all strips that decrease in width by t. The widths of the R_{ff} and R_{gg} strips remain unchanged.

Consider what happens at one of the critical events, where the change is no longer simply linear. We claim that the quantities H_{fg} and H_{gf} can be easily updated at these points. To see this note that, at a critical event, an fg-type strip disappears (its width goes to zero) and a new gf-type strip appears (see Figure 3). At the same time, the right boundary of the adjacent strip to the left is converted from f to g, and the left boundary of the adjacent strip to the right is converted from g to f. To update H_{fg} and H_{gf} we need to know just the values of f and g around the critical event.

This gives us the following algorithm:

1. Initialize:
    - Given the piecewise constant functions f and g, determine the critical events: the shifts of f by t such that a discontinuity in f coincides with a discontinuity in g. Sort these critical events by how far f must be shifted for each event to occur. Let $c_0, c_1, \ldots, c_e$ be the ordered list of shifts for the critical events; $c_0 = 0$.
    - Calculate $h(0, 0)$. This involves summing the contributions of each of $m + n$ strips and takes linear time.
    - Determine initial values for H_{fg} and H_{gf}.

2. For $i = 1$ to e

   - Determine the value of $h(c_i, 0) = (H_{gf} - H_{gf})(c_i - c_{i-1}) + h(c_{i-1}, 0)$.
   - Update H_{fg} and H_{gf}.

The algorithm takes advantage of the fact that $h(t, 0)$ is piecewise linear as a function of t; thus, the entire function can be determined once we know an initial value and the slope for each piece. It is easy to see that the time for initialization is dominated by the time it takes to sort the critical events: $O(e \log e)$, where e is the number of critical events, or $O(mn \log mn)$ where m and n are the sizes of the two polygons. The updates required for the remainder of the algorithm take a total of $O(e)$, or $O(mn)$ time. ∎

In practice, it might be useful to recalculate $h(t, 0)$ periodically from scratch to avoid errors that could accumulate. If this is done every $O(\frac{m+n}{\log e})$ steps then the time bound for the entire algorithm remains $O(e \log e)$.

4 Examples

In this section we illustrate some of the qualitative aspects of the distance function $d_2(A, B)$ by comparing some simple polygons using the algorithm described in the previous section. In addition to providing a distance, $d_2(A, B)$, between two polygonal shapes, the method gives the relative orientation, θ^*, and the corresponding reference points of the two polygons for which this distance is attained.

The first example compares two simple polygons that are very similar in shape, but which are at different orientations (see Figure 4). The value of $d_2(A, B)$ is 5.06 which is attained at a rotation of 180 degrees and with the upper left vertex of the first polygon matched with the lower right vertex of the second one. (Distances less than about 5 seem to correspond to polygons that a person would rate as resembling each other; pairs of polygons that are very different can have arbitrarily high distances.)

To illustrate how the distance function can be used to compare a *model* with several different instances, we consider the case of matching a triangle to three different shapes: another triangle, a "cut-off" triangle, and a quadrilateral, as

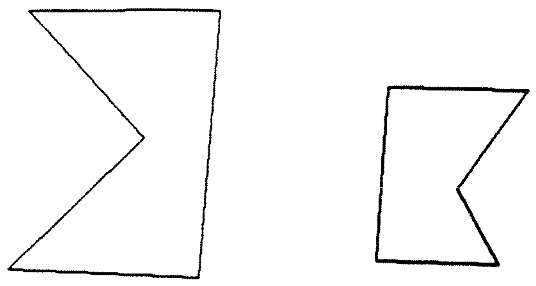

Figure 4: Comparing two simple polygons.

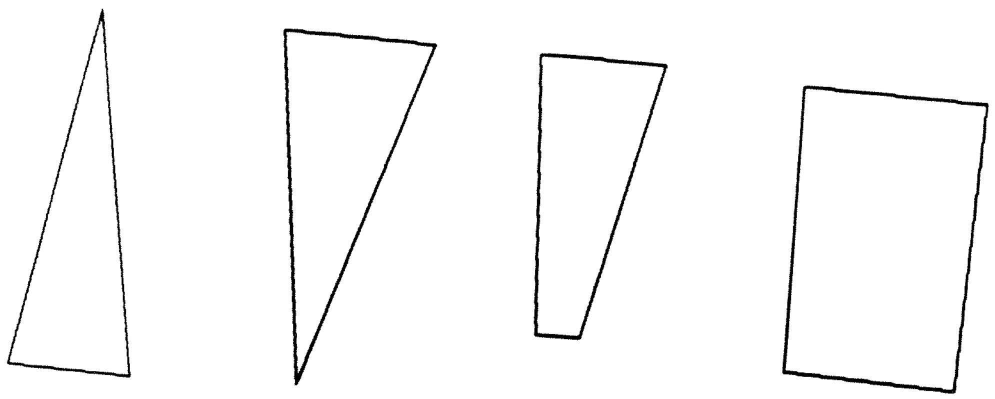

Figure 5: Comparing several polygons.

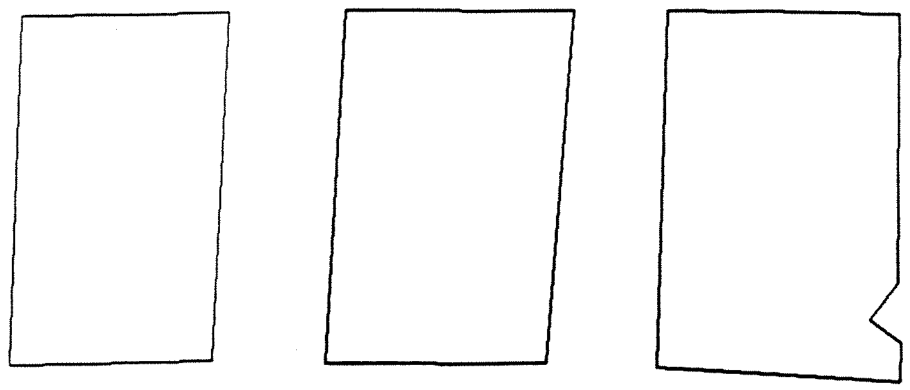

Figure 6: A rectangle with a notch removed.

shown in Figure 5. The distances for these three matches are 2.85, 4.43 and 7.27, respectively. The three corresponding rotations are 175, 176 and 184 degrees. Thus we see that the triangle is the best match, the cut-off triangle matches second best, and the trapezoid is the worst match. The match to the cut-off triangle suggests that the metric is useful for matching partially occluded objects, as long as the overall shape of the object does not change too radically.

Our metric also provides a qualitatively good estimate of a match when one polygon is an instance of another, but with some perturbation of its boundary. A simple example is given by the cut-off triangle in the previous figure, where the orientation estimate is still about 180 degrees even though the number of vertices has changed. Another example is given in Figure 6, where we compare a model rectangle against a second rectangle and against another rectangle with a notch removed. The distances are 2.21 and 3.48, respectively, with relative orientations of 2 and 1 degrees.

An extreme case of matching distorted polygons is shown in Figure 7, where a triangle is compared with a somewhat triangular shape. In this case the distance is 6.96, and the orientation difference is 2 degrees. Note however, that, as mentioned in the introduction (see Figure 2), such perturbations must occur relatively uniformly along the perimeter of the polygon for the match to be reasonable.

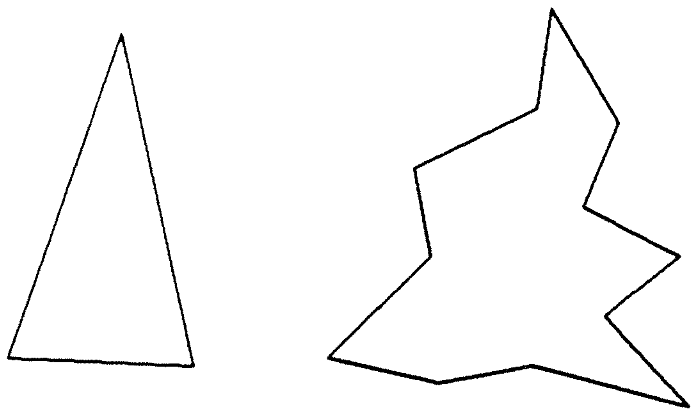

Figure 7: Matching a triangle to a highly noisy shape.

5 Summary and Discussion

We have suggested using the L_2 metric on the turning functions of polygons as a way to implement the intuitive notion of shape-resemblance. This method for comparing shapes has the following advantages:

- It is a metric on polygonal shapes.
- It compares *shape* alone; it is invariant under translation, rotation, and change-of-scale.
- It is reasonably easy to compute, taking time $O(mn \log mn)$ to compare an m vertex polygon against an n vertex polygon.
- Finally, it corresponds well to intuitive notions of shape resemblance.

In addition, this metric works for nonconvex as well as convex polygons, and even works for some polygonal shapes that are not simple and/or not closed.

Like the method developed by Schwartz and Sharir [SS], our method is actually based on a convolution. Recall that the major portion of our algorithm is devoted to minimizing $h(t,\theta) = \int_0^1 (f(s+t) - g(s) + \theta)^2 \, ds$. When this formula is multiplied out, all the terms depend on f alone or g alone, except for the convolution term $\int_0^1 f(s+t)g(s) \, ds$. If f and g are piecewise constant with m and n discontinuities, respectively, then each term can be calculated in either $O(m)$ or $O(n)$ time

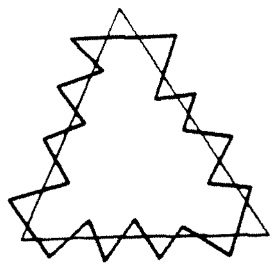

Figure 8: A match on which the L_1 metric does poorly but the L_2 does well.

except for the convolution term, which seems to require $O(mn \log mn)$ time. Of course, the Fast Fourier Transform (FFT) can be used to compute a convolution in $O(k \log k)$ time, but this requires k evenly spaced sample points for each of f and g. For our problem, the discontinuities are not necessarily evenly spaced, so the FFT cannot be used unless we are willing to approximate our functions f and g. A good approximation may require more than mn points. In any case, the development of a fast method for convolutions using unevenly spaced sample points would lead to improvements in the time bound for our technique.

We used the L_2 metric, but similar techniques can be used to develop polygon-resemblance metrics that are based on different function-space metrics. Unfortunately, not all such metrics have L_2's advantages of being reasonbly easy to compute and matching our ituitive idea of shape resemblance. For instance, it is also possible to compute the L_1 metric on two $\Theta(s)$ functions using an algorithm similar to that in Section 3. In the case of the L_1 metric, however, the value of θ^* is not given directly for each value of s as it is for the L_2 metric. Thus for each of the mn critical events, the optimal value of θ must be computed explicitly. Using a data structure similar to that in Section 3, the overall computation can be done in time $O(n^3 \log n)$, as opposed to $O(n^2 \log n)$ for the L_2 metric.

The L_1 metric has an additional drawback: The optimal match will occur when one side of polygon A is at the same orientation as some side of polygon B. That is, the value of θ^* is such that in some strip the two functions $f(s+t)$ and $g(s)$ are coincident. In contrast, the L_2 metric finds the optimal orientation (in a least squares sense) without requiring any two edges to be identically oriented.

Examine Figure 8 to see why requiring identical orientations can be undesirable; for the L_1 metric the best match occurs at an orientation difference of 76 degrees, bringing two edges into alignment. This would rotate the two figures so that they approximately form a star, a bad match. In contrast, for the L_2 metric the best match is at an orientation difference of 7 degrees, which agrees quite well with our intuitive sense of the best match.

It may be possible to apply our methods to problems involving partially occluded objects, that is objects for which the entire model is known, but for which only a portion of the boundary appears in the image. Our technique as presented here has not been designed to work with such objects, although, as shown by some of our examples, it seems to give intuitively correct answers when objects are not severely occluded. The combination of occluded objects and our desire to make our metric independent of change-of-scale causes some difficulty. We were able to control change-of-scale problems by normalizing our polygons to make all perimeters have length one. If portions of a boundary are unknown then it is unclear how this normalization should be done. Of course, if the scale of the image is known, then partially occluded objects do not present any difficulties.

Our results can be generalized to include cases in which the shapes A and B have some or all of their boundary represented as circular arcs. If A includes some circular arcs on its boundary, then the turning function $\Theta_A(s)$ is piecewise linear instead of piecewise constant. As before, to compare shapes A and B we need to minimize

$$h(t,\theta) = \int_0^1 |\Theta_A(s+t) - \Theta_B(s) + \theta|^2 \, ds.$$

The derivation of $\theta^*(t)$, the best value of θ for given t, does not change at all; it is still a quadratic function of t. But if the shapes include circular arcs, then $h(t,0)$ can be piecewise cubic instead of simply piecewise linear. Thus, the minimum value of $h(t,\theta)$ does not necessarily occur at a critical event, and more information is needed in order to determine the behavior of h between critical events.

However, since $h(t,\theta^*(t))$ is piecewise cubic as a function of t, we can determine the behavior of h between critical events if we have enough data points for h. If we collect two additional $(t, h(t,\theta^*(t)))$ pairs between each adjacent pair of critical events then we can determine the coefficients of $h(t,\theta^*(t))$ $(= a_1t^3 + a_2t^2 + a_3t + a_4$ between critical events) and compute the minimum analytically. Thus, the basic approach of calculating $h(t,\theta)$ in $O(m+n)$ time for each of $O(mn)$ values is still valid and there is an algorithm that runs in time $O(mn(m+n))$.

The $O(mn \log mn)$ time algorithm can also be generalized to work with circular arcs, and the resulting algorithm has the same asymptotic time bound. Unfortunately, the number of updates needed and the resulting accumulation of round-off error may make this algorithm impractical.

Acknowledgement

We would like to thank Karel Zikan for suggesting the L_2 norm over the L_1 norm.

References

[HT] J. Hong and X. Tan, "The Similarity Between Shapes under Affine Transformation", Technical Report No. 336, Robotics Report No. 133, New York University, Courant Institute of Mathematical Sciences, December, 1987.

[HW] J. Hong and H.J. Wolfson, "An Improved Model-Based Matching Method Using Footprints", *Ninth International Conference on Pattern Recognition*, Rome, Italy, November 14-18, 1988.

[OW] J. O'Rourke and R. Washington, "Curve Similarity via Signatures", in *Computational Geometry*, G. Toussaint (ed.), North-Holland, 1985. pp. 295-318.

[SS] J.T. Schwartz and M. Sharir, "Some Remarks on Robot Vision", Technical Report No. 119, Robotics Report No. 25, New York University, Courant Institute of Mathematical Sciences, April, 1984.

[Wo1] H. Wolfson, "On Curve Matching", Technical Report No. 256, Robotics Report No. 86, New York University, Courant Institute of Mathematical Sciences, November, 1986.

[Wo2] H. Wolfson, "On Curve Matching", *Proc. of IEEE Workshop on Computer Vision*, Miami Beach, FL, November 30-December 2, 1987.