Measuring Point Set Similarity with the Hausdorff Distance: Theory and Applications

Scott D. Cohen *

1 Introduction

We consider the problem of measuring the similarity between two finite point sets using the Hausdorff distance $H(A, B)$. This distance is small when each point in A is close to some point in B, and each point in B is close to some point in A. The definition does not require that A and B have the same number of points. More importantly, $H(A, B)$ is a metric. A point set is identical only to itself, the order of comparison is irrelevant, and the triangle inequality holds. If two point sets are similar to a third, then they are similar to each other. This agrees with our intuition of shape similarity.

The Hausdorff distance is insensitive to small perturbations of the point sets. This stability is important because it allows for small positional errors in point feature sets. Also, the Hausdorff distance does *not* build one-to-one correspondences between the points in the two sets. Shape matchers which do so will require an unreasonable amount of time when the number of features detected becomes very large.

For cases in which we do not want symmetry, there is a directed version of the Hausdorff distance, denoted $h(A, B)$. This distance from A to B is small whenever each point of A is close to some point in B. Consider, for example, an illustration retrieval system which compares point feature sets of a database illustration and a given query. We may decide that a query is similar to a database figure if features present in the query are also present in the database figure, but not necessarily vice versa. In this case we would use the directed Hausdorff distance from the query set to the database figure set.

This paper is intended to be a summary and synthesis of four papers:

1. Geometric Pattern Matching under Euclidean Motion [2]
2. Geometric Pattern Matching in d-Dimensional Space [1]
3. Comparing Images Using the Hausdorff Distance [12]
4. Locating Objects Using the Hausdorff Distance [15].

Papers 1 and 2 are "theory" papers concerned with the exact computation of the Hausdorff distance. Both present algorithms for the optimization problem: find $\min_{g \in \mathcal{G}} H(A, g(B))$, where $\mathcal{G}$ is some transformation group. Paper 1 considers the case when A and B are point sets in $\mathbf{R}^2$ with $\mathcal{G} = \mathcal{E}_2$, the group of planar Euclidean motions. Paper 2 works in $\mathbf{R}^d$ for $d \geq 3$, with $\mathcal{G} = \mathcal{T}$, the group of translations. Papers 3 and 4 are "application" papers which show how to use the

*Submitted on November 7, 1995 for the paper synthesis part of the Physical Qual.

Hausdorff distance to find a binary model within a binary image. They differ in the allowable transformations of the model that can be located. In paper 3, a rotated, translated version of a given model can be recognized (i.e., $\mathcal{G} = \mathcal{E}_2$). In paper 4, an affine transformation of the model is allowed (i.e. $\mathcal{G} = \mathcal{A}_2$).

The rest of this paper is organized as follows. In Section 2 we give some basic definitons. In Section 3 we develop a common framework for papers 1 and 2. Details for the optimization problem with $d = 2$ and $\mathcal{G} = \mathcal{E}_2$ are presented in Section 4, and a brief discussion of the case $d \geq 3$ and $\mathcal{G} = \mathcal{T}$ is included in Section 5. In Section 6 we develop a common framework for papers 3 and 4. Details for model location with $\mathcal{G} = \mathcal{T}$ are presented in Section 7, and a brief discussion of the case $\mathcal{G} = \mathcal{A}_2$ is given in Section 8. Finally, Section 9 contains some concluding remarks.

2 Basic Definitions

The Hausdorff distance between two finite point sets $A = \{a_1, \ldots, a_m\}$ and $B = \{b_1, \ldots, b_n\}$ is defined as

$$H(A, B) = \max(h(A, B), h(B, A)),$$

where

$$h(A, B) = \max_{a \in A} \min_{b \in B} \rho(a, b)$$

is the directed Hausdorff distance from A to B. Here $\rho(a, b)$ is the distance between a and b. In this paper, ρ will be the L_2 or L_∞ metric. For a transformation group $\mathcal{G}$, we define the Hausdorff distance under $\mathcal{G}$ as

$$M_{\mathcal{G}}(A, B) = \min_{g \in \mathcal{G}} H(A, g(B)).$$

In words, we find the transformation $g \in \mathcal{G}$ which matches A and $g(B)$ as closely as possible. In this paper, $\mathcal{G}$ will be the group of translations $\mathcal{T}$, or planar Euclidean motions $\mathcal{E}_2$, or planar affine transformations $\mathcal{A}_2$. In many applications, however, we will not be satisfied with finding the exact minimum of $H(A, g(B))$. Rather, we will want to find all transformations g that make $H(A, g(B))$ smaller than some specified threshold ϵ.

3 A Common Theory Framework

The goal in papers 1 and 2 is to solve

The Optimization Problem. Compute $M_{\mathcal{G}}(A, B) = \min_{g \in \mathcal{G}} H(A, g(B))$.

A closely related problem is

The Decision Problem. Given $\epsilon > 0$, does there exist $g \in \mathcal{G}$ such that $H(A, g(B)) \leq \epsilon$?

The smallest $\epsilon = \epsilon^*$ for which the answer to the decision problem is *yes* is the answer to the optimization problem. The general strategy used in both theory papers is to develop an algorithm for the decision problem and then use this algorithm in some form of parametric search [13] to get an algorithm for the optimization problem.

A common framework is used in attacking the decision problem. This framework turns the decision problem into a geometric intersection problem. For the moment we assume $\mathcal{G} = \mathcal{T}$. Let C_ϵ denote the closed "ball" of radius ϵ centered at the origin:

$$C_\epsilon = \{x \in \mathbf{R}^d : \|x\| \leq \epsilon\}.$$

When the L_2 norm is used, C_ϵ is a true ball of radius ϵ. When the L_∞ norm is used, C_ϵ is a cube with side length 2ϵ. Now define

$$A^\epsilon = \bigcup_{i=1}^{m} (a_i \oplus C_\epsilon),$$

where $\oplus$ represents Minkowski sum. In the plane with the L_2 norm, A^ϵ is the union of m discs with centers at the a_i. The set of translations that cause some $b_j \in B$ to fall within A^ϵ is simply $A_j^\epsilon = A^\epsilon \oplus -b_j$, a translation of A^ϵ by $-b_j$. The set of translations which cause all points in B to fall within A^ϵ is

$$S(A, \epsilon, B) = \bigcap_{j=1}^{n} A_j^\epsilon.$$

The set $S(A, \epsilon, B)$ is an intersection of n translated copies of A^ϵ. Clearly, there exists $t \in \mathcal{T}$ such that $h(B \oplus t, A) \leq \epsilon$ iff $S(A, \epsilon, B) \neq \emptyset$. Using the fact that $h(A, B \oplus t) = h(A \ominus t, B)$, we can show that the answer to the decision problem is *yes* iff $S(A, \epsilon, B) \cap -S(B, \epsilon, A) \neq \emptyset$.

Let us rephrase the emptiness question for $S(A, \epsilon, B)$. Suppose each $b_j \in B$ has its own color which is used to color the m balls in the union $A_j^\epsilon = A^\epsilon \oplus -b_j$. We call A_j^ϵ a layer because it is composed of balls which are all the same color (the color assigned to b_j). Thus $S(A, \epsilon, B)$ is the intersection of n different colored layers. Define the depth of a point p to be the number of layers A_j^ϵ which contain p. (This is the same as the number of different colored balls which contain p.) Then $S(A, \epsilon, B) \neq \emptyset$ iff there exists a point p of depth n.

In the next section we discuss the directed decision problem (more precisely, the decision problem with $H(A, g(B))$ replaced by $h(g(B), A)$) for $\mathcal{G} = \mathcal{E}_2$. Thus we need to introduce a rotation parameter θ into our geometric intersection formulation:

$$\begin{aligned} A_j^\epsilon(\theta) &= A^\epsilon \oplus -R_\theta b_j \\ S(A, \epsilon, B, \theta) &= \bigcap_{j=1}^{n} A_j^\epsilon(\theta), \end{aligned}$$

where R_θ denotes the standard rotation matrix for $\mathbf{R}^2$. The centers of the discs in the union $A_j^\epsilon(\theta)$ are $a_i - R_\theta b_j$, $i = 1, \ldots, m$. As θ changes, the center $a_i - R_\theta b_j$ moves along a circular path with center a_i and radius $\|b_j\|$. Thus each set of centers $S_j(\theta) = A \oplus -(R_\theta b_j)$ (along with their corresponding discs) moves rigidly in the plane. The set $A_j^\epsilon(\theta)$ does not change shape or orientation as θ advances - it only changes position. In order to answer the directed decision problem for $\mathcal{G} = \mathcal{E}_2$, we answer question: Does there exist $\theta \in [0, 2\pi)$ such that $S(A, \epsilon, B, \theta) \neq \emptyset$? The answer to this question is *yes* iff the answer to the directed Hausdorff decision problem is *yes* .

4 The Optimization Problem: $d = 2$, $\mathcal{G} = \mathcal{E}_2$, $\rho = L_2$

In this section we shorten $S(A, \epsilon, B, \theta)$ to $S(\theta)$. Define $\mathcal{A}(\theta)$ to be the arrangement formed by overlaying the boundaries $\partial(A_j^\epsilon(\theta)), j = 1, \ldots, n$. Since $S(\theta)$ is the intersection of sets with boundaries $\partial(A_j^\epsilon(\theta))$, it consists of some (possibly zero) faces of the arrangement $\mathcal{A}(\theta)$. Clearly $S(\theta)$ is nonempty iff there exists a vertex of $\mathcal{A}(\theta)$ that has depth n. See Figure 1 to help get a visual understanding of $\mathcal{A}(\theta)$ and $S(\theta)$.

Our strategy is to "sweep" the transformation space from $\theta = 0$ to $\theta = 2\pi$ while maintaining the depth of the vertices in $\mathcal{A}(\theta)$. If at any time θ during the sweep, a vertex (x, y) of depth n is found, we stop the sweep and report that $g = (x, y, \theta)$ is a Euclidean motion of B that makes

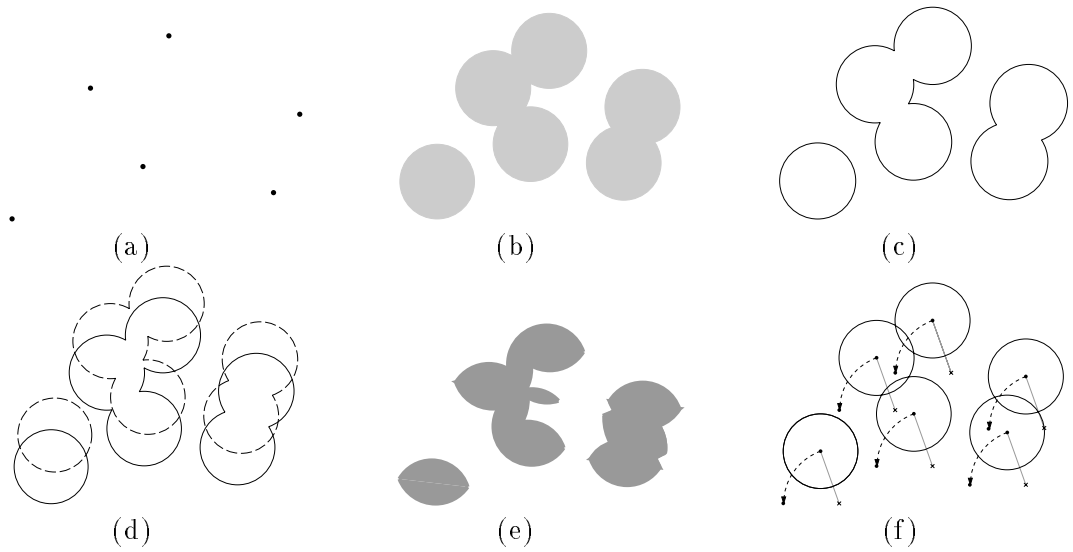

Figure 1: The sets in our geometric intersection formulation of the decision problem for $\mathcal{G} = \mathcal{E}_2$ and $\rho = L_2$. (a) A. (b) A^ϵ. Each $A_j^\epsilon(\theta)$ is a translated version of A^ϵ. (c) $\partial(A^\epsilon)$. (d) $\mathcal{A}(\theta)$, θ fixed, $n = 2$. This is the overlay arrangement of $\partial(A_1^\epsilon(\theta))$ (solid) and $\partial(A_2^\epsilon(\theta))$ (dashed). (e) $S(\theta)$ is composed of faces of $\mathcal{A}(\theta)$. (f) The evolution of a single $A_j^\epsilon(\theta)$. The x's mark the centers of rotation a_i. The discs in each $A_j^\epsilon(\theta)$ move rigidly.

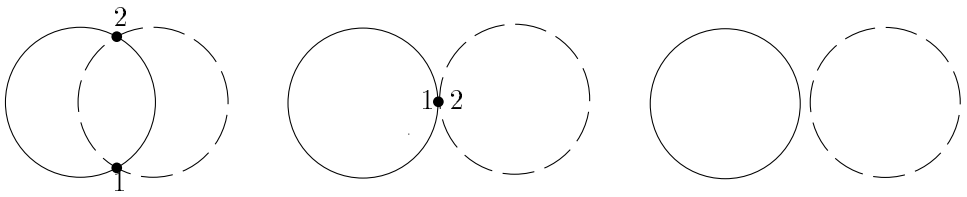

Figure 2: A Creation/Deletion Event. Vertices 1 and 2 are destroyed if we read the diagram from left to right, and created if we read from right to left.

$h(g(B), A) \leq \epsilon$. If no such vertex is detected for $\theta \in [0, 2\pi)$, then no such Euclidean motion exists. To perform the sweep, we need to process events in which a vertex is created or destroyed or when an existing vertex changes depth. The latter event corresponds to a vertex entering or leaving some $A_j^\epsilon(\theta)$.

The only time that a vertex of $\mathcal{A}(\theta)$ is created or destroyed is when two boundary arcs in $\mathcal{A}(\theta)$ become tangent. We call this a creation/deletion event. See Figure 2. Every creation/deletion event is a tangency of two discs in two different $A_j^\epsilon(\theta)$'s but not vice versa (an event requires that the point of tangency is on the boundary of the two $A_j^\epsilon(\theta)'s$). Since the discs in each $A_j^\epsilon(\theta)$ are executing circular motion, two such discs can have at most two times of tangency. Hence the total number of creation/deletion events is $O(m^2n^2)$. In fact, we can charge a creation/deletion event involving circles with centers $p \in S_{j_1}(\theta)$ and $q \in S_{j_2}(\theta)$ to the Voronoi edge between p and q in the dynamic Voronoi diagram of $S_{j_1}(\theta) \cup S_{j_2}(\theta)$. (The point of tangency between the two boundary arcs is a point on the edge determined by p and q.) Considering all $O(m^2n^2)$ creation/deletion events, no edge of a pairwise dynamic Voronoi diagram is charged more than twice.

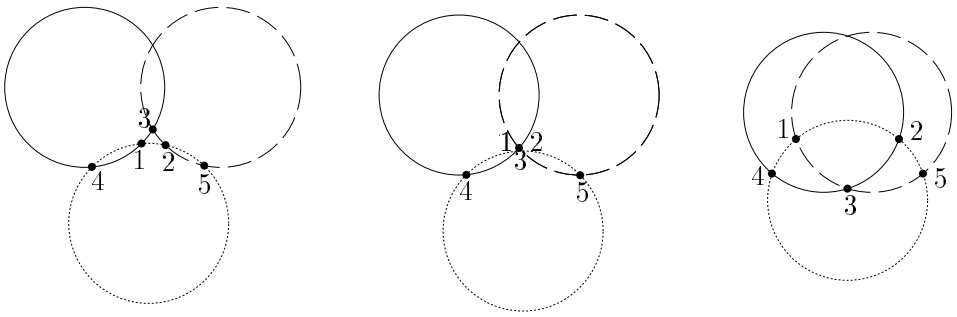

Figure 3: A Depth Change Event. Reading from left to right, the depths of vertices 1, 2, and 3 increase by one. Reading from right to left, the depths of vertices 1, 2, and 3 decrease by one.

The only time that an existing vertex of $\mathcal{A}(\theta)$ changes depth is when three boundary arcs from at least two different $A_j^\epsilon(\theta)$'s intersect at a point. We call this a depth change event. See Figure 3. A coincidence of three boundary arcs centered at $p \in S_{j_1}(\theta)$, $q \in S_{j_2}(\theta)$, and $r \in S_{j_3}(\theta)$ means that p, q, and r are equidistant from one another and that no other point in $S_{j_1}(\theta) \cup S_{j_2}(\theta) \cup S_{j_3}(\theta)$ is closer to the point of coincidence than p, q, and r. Thus p, q, and r determine a vertex in the pairwise (if only two j_l are distinct) or triplewise (if all three j_l are distinct) dynamic Voronoi diagram of $S_{j_1}(\theta) \cup S_{j_2}(\theta) \cup S_{j_3}(\theta)$. Using this fact, we charge depth change events to vertices in pairwise or triplewise dynamic Voronoi diagrams of $\{S_j(\theta)\}_{j=1}^n$. Considering all depth change events, no vertex of a pairwise or triplewise diagram is charged more than a constant number of times. A result from Huttenlocher et. al. [10] states that there are $O(m^2k^2\lambda_s(k))$ topological changes in the dynamic Voronoi diagram of $\cup_{i=1}^k T_i$ when each set of points T_i, $|T_i| = m$, moves rigidly (here s is a small constant). Therefore, the number of depth change events involving arcs from the boundaries of a fixed pair or triple of $A_j^\epsilon(\theta)$'s is $O(m^2)$. It follows that the total number of depth change events is $O(m^2n^3)$.

We can compute a superset of the creation/deletion and depth change events by computing all pairwise and triplewise dynamic Voronoi diagrams of $\{S_j(\theta)\}_{j=1}^n$. Using a result from Guibas et. al. [9], each pairwise or triplewise dynamic Voronoi diagram can be computed in time $O(m^2 \log m)$: $O(m)$ sources, $O(m \log m)$ to compute the initial Voronoi diagram at $\theta = 0$, and $O(\log m)$ per update for $O(m^2)$ topological events. For each of the $O(m^2)$ edges and vertices that arise, we consider the circles corresponding to the sources which define the edge or vertex. We want to determine all the creation/deletion which charged a given edge and all the depth change events which charged a given vertex (possibly none). In either case, the times of tangency or coincidence of the circles involved can be determined in constant time. We need to check at each time of tangency (coincidence) if the point of tangency (coincidence) is "in" $\partial(A^\epsilon)$. We preprocess the $O(m)$ arcs in $\partial(A^\epsilon)$ as follows. Compute an array bdy in which $bdy[i]$ contains a list of angle pairs $[\phi_{start}, \phi_{stop}]$ which define the arcs of the circle with center a_i that are in $\partial(A^\epsilon)$. Each arc list is sorted cyclically around its center. The bdy array can be computed in $O(m \log m)$ time. Given a point of tangency (coincidence) on a circle with center $a_i - R_\theta b_j$, we lookup its angle ϕ in $bdy[i]$ in time $O(\log m)$ to determine if the point is in $\partial(A^\epsilon)$. Since this is the same time bound as for the update to the dynamic Voronoi diagram, we can compute all creation/deletion and depth change events in time $O(m^2n^3 \log m)$.

We are now ready to fully describe and analyze the sweep algorithm to answer the directed

decision problem. We will store the vertices of the current $\mathcal{A}(\theta)$ in a dynamic dictionary V which is implemented as some balanced tree structure. A vertex v which is defined by the intersection of circle k and circle l is labelled with (k,l) (and stored twice in V - once as (k,l) and once as (l,k)). The vertices stored in V are ordered by their first index, and ties are broken by ordering the vertices $(k,*)$ by their cyclic ordering around circle k. There are two types of vertices in V: those that are vertices on the boundary of some $A_j^\epsilon(\theta)$ (type I), and those that are the intersection between two arcs from the boundaries of two different $A_j^\epsilon(\theta)$'s (type II). A type I vertex is defined by an intersection between two of m circles. Since two distinct circles intersect in at most two places, the sequence of boundary arcs labelled with circles is an $(m,2)$ Davenport-Schinzel sequence, and there are $O(m)$ type I vertices from a single $A_j^\epsilon(\theta)$. Therefore there are $O(mn)$ type I vertices in V at any time θ. By definition, a type II vertex is the intersection of one of the $O(m)$ arcs on $\partial(A_{j_1}^\epsilon(\theta))$ with one of the $O(m)$ arcs on $\partial(A_{j_2}^\epsilon(\theta))$, $j_1 \neq j_2$. Thus there are $O(m^2)$ type II vertices for any fixed pair (j_1, j_2) and a total of $O(m^2n^2)$ type II vertices present in V at any time θ. Since V is a balanced tree structure, we can perform lookups, insertions, and deletions in time $O(\log mn)$. The algorithm to solve the directed decision problem is as follows:

Directed Decision Problem Algorithm

- For $\theta = 0$, compute the minimum Hausdorff distance under translation from B to A. For $\rho(\cdot) = \|\cdot\|_2$, this can be done in time $O(mn^2 \log mn)$ [11]. If this distance is less than or equal to ϵ, then stop and answer *yes* .
- Compute the $O(m^2n^3)$ events in time $O(m^2n^3 \log m)$.
- Sort the events in time $O(m^2n^3 \log mn)$.
- Consider each event e in succession.
  - If e is a depth change event, then up to three vertices in V will change depth by 1. In constant time we compute whether each such vertex is entering or leaving some $A_j^\epsilon(\theta)$. If a vertex is entering, we increment its depth. If a vertex is leaving, we decrement its depth. Lookup of the vertices in V takes time $O(\log mn)$. Updating the depths takes constant time. We must also swap the positions of $O(1)$ vertices in V because some cyclic orderings have changed. The total time for a depth change event is $O(\log mn)$ and the total time for all depth change events is $O(m^2n^3 \log mn)$.
  - If e is a creation/deletion event, then two vertices of V are either being created or destroyed. If the vertices are being destroyed, then we just delete them from V in $O(\log mn)$ time. If the vertices are being created, we insert them into V in $O(\log mn)$ time. In addition, we need to compute the depths of the new vertices. We do this by examining the ≤ 2 cyclic neighbors of the new vertices. The time for one creation/deletion event is thus $O(\log mn)$ and the total time for all creation/deletion events is $O(m^2n^2 \log mn)$.
  - If e causes the discovery of a depth n vertex, then stop and answer *yes* .
- If all events are processed without discovering a depth n vertex, then answer *no* .

The total time for processing all events is $O(m^2n^3 \log mn)$. This is also the time to sort all the events. Since no other step takes longer, the total sweep time is $O(m^2n^3 \log mn)$. We have thus proven

Theorem 1 *Given ϵ and planar point sets A and B, $|A| = m$, $|B| = n$, the directed Hausdorff decision problem from B to A under Euclidean motion can be solved in time $O(m^2n^3 \log mn)$.*

In the (undirected) decision problem, we are searching for a Euclidean motion $e \in \mathcal{E}_2$ for which $h(e(B), A)$ and $h(A, e(B))$ are simultaneously both less than or equal to ϵ. This problem can be solved using the same setup for the directed decision problem from B to A by constructing an arrangement $\mathcal{B}(\theta)$ for $h(A, e(B))$ which is analogous to $\mathcal{A}(\theta)$ for $h(e(B), A)$, and overlaying the two arrangements. In this composite arrangement, we are looking for a vertex whose depth is $m + n$. The previous analysis can be modified to show

Theorem 2 *Given ϵ and planar point sets A and B as above, the Hausdorff decision problem under Euclidean motion can be solved in time $O(m^2n^2(m + n) \log mn)$.*

We now briefly describe how parametric search can be used to get an algorithm for the optimization problem using our algorithm for the decision problem. The idea is to run our decision problem algorithm A_S "generically", without specifying ϵ, but with the intent of simulating the execution of A_S on ϵ^*. During the generic run, A_S generates a list of event times that are functions of ϵ. To sort these event "times", we will use a sequential simulation of a parallel comparison-based sorting routine A_P. This sorting routine is parametrized by ϵ and uses P processors in $O(\log P)$ parallel steps. At each step, there are P independent comparisons to be made between two event times. Suppose we are comparing $\theta(\epsilon) \leq \hat{\theta}(\epsilon)$. The critical values of ϵ for this comparison are the values for which equality holds and these values define intervals over which the result of the comparison is constant. For our problem, each comparison has $O(1)$ critical values which can be computed in $O(1)$ time. The set of all critical values for all P independent comparisons for one step in the parallel sort can be computed and sorted in $O(P)$ time. Since we are trying to mimic the execution of A_S on ϵ^*, we want to resolve the comparisons at ϵ^*. The P comparisons can be resolved in $O(P)$ time once we locate ϵ^* within the sorted set of critical values. We can do this by using A_S to do comparisons during binary search. If we run A_S on some critical (concrete) value ϵ_c and it answers *yes* , then we know that $\epsilon^* \leq \epsilon_c$. If it answers *no* , then we know that $\epsilon^* > \epsilon_c$. Since a binary search for ϵ^* only needs to do comparisons against ϵ^*, and A_S can be used to resolve any such comparison, we can locate ϵ^* in the set of critical values without knowing the value of ϵ^*! In doing so, we have also narrowed down the location of ϵ^* to a concrete interval. This interval constraint is updated after each parallel step. One parallel step costs $O(P + T_S \log P)$, where T_S is the time for a concrete run of A_S (we resolve $O(\log P)$ comparisions during the binary search using A_S). Since we simulate $O(\log P)$ parallel steps, the total time for the algorithm is $O(P \log P + T_S \log^2 P)$. "The important property here is that ϵ^* is a critical value for both the sorting algorithm and our sequential algorithm A_S, ensuring that it will be discovered during the parametric search." [2] The value of ϵ^* is the left endpoint of the final interval constraint. Putting $P = O(mn)$ and $T_S = O(m^2n^2(m + n) \log(mn))$, we see that we can solve the optimization problem in time $O(m^2n^2(m + n) \log^3 mn)$. Using Cole's parallel sorting algorithm and sequential simulation [4], we can remove one $\log mn$ factor. In summary,

Theorem 3 *Given point sets A and B as above, the optimization problem for Euclidean motion may be solved in time $O(m^2n^2(m + n) \log^2 mn)$.*

5 The Optimization Problem: $d \geq 2$, $\mathcal{G} = \mathcal{T}$, $\rho = L_\infty$

The planar case for the L_∞ metric with $\mathcal{G} = \mathcal{T}$ is considered in [3]. We start with the result for the decision problem.

Theorem 4 *Given $\epsilon > 0$ and two point sets A and B in the plane, and using the L_∞ distance as the underlying metric, we can determine whether the minimum Hausdorff distance under translation between A and B is less than or equal to ϵ in time $O(mn \log mn)$, where $m = |A|$ and $n = |B|$.*

Proof. (Sketch) We consider the directed problem from B to A. With the L_∞ metric, each layer A_j^ϵ is a union of m squares. As in the previous section, we are trying to determine if there is a point in translation space which is covered by all the A_j^ϵ. If we break up each A_j^ϵ into nonoverlapping rectangles, then the depth of a point becomes the number of rectangles which cover it. Since the complexity of the boundary of a union of m axis-aligned squares is $O(m)$, each A_j^ϵ can be divided into $O(m)$ nonoverlapping rectangles. Thus we have a total of $O(mn)$ rectangles. The maximum depth over all points in the union of $O(mn)$ rectangles can be computed in $O(mn \log mn)$ time by sweeping a segment tree [8] over the arrangement of rectangles. The events are the $O(mn)$ y coordinates of the horizontal sides of the rectangles, and the intervals inserted and deleted from the segment tree are the horizontal sides. Simple modifications to the above proof show that this is also the bound for the undirected problem. ■

A form of parametric search due to Frederickson and Johnson [5, 6] is used to obtain an algorithm for the optimization problem from the algorithm given above for the decision problem. The additional cost factor in passing from the decision problem to the optimization problem is $O(\log mn)$:

Theorem 5 *Given planar point sets A and B as above and using the L_∞ metric, the minimum Hausdorff distance under translation can be computed in time $O(mn \log^2 mn)$.*

The higher dimensional case for the L_∞ metric with $\mathcal{G} = \mathcal{T}$ is treated in paper 2. As in the plane, the strategy is to track depth information as a hyperplane is swept along x_d. A higher dimensional data structure to play the role of the segment tree is needed for this task. The authors use a modified version of the Orthogonal Partition Tree (OPT) first introduced by Overmars and Yap [14]. They prove the following result:

Theorem 6 *Given point sets A and B in d-space and using the L_∞ metric, the minimum Hausdorff distance under translation between A and B can be computed in time $O(n^3 \log^2 n)$ if $d = 3$ and $O(n^{(4d-2)/3} \log^2 n)$ if $d > 3$, where $n = \max(|A|, |B|)$.*

6 A Common Application Framework

Both application papers 3 and 4 use the Hausdorff distance to locate a binary model within a binary image. The point set A defines the image and the point set B defines the model. These are both nonnegative integer point sets:

$$\begin{aligned} A &\subseteq \{(i,j) \in \mathbf{Z}^2 : 0 \le i < h_I, 0 \le j < w_I\} \\ B &\subseteq \{(i,j) \in \mathbf{Z}^2 : 0 \le i < h_M, 0 \le j < w_M\}, \end{aligned}$$

where the image is $w_I \times h_I$ and the model is $w_M \times h_M$.

In the theory sections of this paper we were concerned with the optimization problem. In searching for a model within an image, however, we may wish to find all occurrences of the model within an image. That is, we would like to solve

The Reporting Problem. Given $\tau > 0$, compute $\{g \in \mathcal{G} : H(A, g(B)) \leq \tau\}$.

The Hausdorff distance $H(A, B)$ is too sensitive to outliers for use in applications. To compute $h(A, g(B))$, we compute the distance of each point in A to its nearest neighbor in $g(B)$, order these distances, and then take the maximum distance. To allow for noisy data, we could generalize the above procedure to take the L^{th} ranked distance, $0 \leq L \leq m = |A|$. Thus we define the partial directed Hausdorff distance from A to B as

$$h_L(A, B) = L^{th}_{a \in A} \min_{b \in B} \rho(a, b).$$

The partial (undirected) Hausdorff distance between A and B is then defined as

$$H_{LK}(A, B) = \max(h_L(A, B), h_K(B, A)).$$

The user specifies the fractions f_1 and f_2, $0 \leq f_1, f_2 \leq 1$, which determine $L = \lfloor f_2 m \rfloor$ and $K = \lfloor f_1 n \rfloor$. In this way, the user does not need to know the number of points in the image or model.

There are some important points to be made about the partial Hausdorff distance. First, the definition of $h_L(A, B)$ does not require the L points used in the computation to be specified beforehand – the "best" L points are used. Second, the partial Hausdorff distance technically does not obey the triangle inequality. If A and B are similar because A matches some piece of B, and B and C are similar because C matches some piece of B, then A and C need not be similar. This is perfectly reasonable when A and C match different parts of B. The triangle equality will indeed hold when A and C match the same part of B.

There is still a problem with using the value of $h_L(A, g(B))$ as the distance from the image to the transformed model. Ideally, when computing $h_L(A, g(B))$ we only want to consider image points which are close to the transformed model. Image points which are far away from the transformed model are likely to be points which came from other objects in the image. In practice, it suffices to consider only the image points which lie underneath the transformed model. Furthermore, the parameter L should still be present to handle occluded objects, but it should depend directly on the number of image points m' underneath the transformed model, not the number of image points m. Thus we redefine $L = \lfloor f_2 m' \rfloor$ and we define the partial "box" Hausdorff distances as

$$\begin{aligned} h^{box}_L(A, B) &= L^{th}_{a \in A'} \min_{b \in B} \rho(a, b) \\ H^{box}_{LK}(A, B) &= \max(h^{box}_L(A, B), h_K(B, A)), \end{aligned}$$

where A' denotes the points in the image which lie underneath B. When computing $h^{box}_L(A, g(B))$, A' denotes the points in the image which lie underneath $g(B)$. Both application papers 3 and 4 give algorithms for

The Practical Reporting Problem. Given τ,f_1,f_2, compute $\{g \in \mathcal{G} : H^{box}_{LK}(A, g(B)) \leq \tau\}$.

Only transformations in a discretized transformation space are reported. The key restriction in defining the discretization is that if g and $\hat{g}$ are neighbors in the discretized transformation space, then, for any $b \in B$, $g(b)$ and $\hat{g}(b)$ are neighbors on the image grid. In this way, moving one step in transformation space changes the transformed model by very little.

The final ingredients in the algorithm for the model location problem are the model and image distance transforms $d(x)$ and $d'(x)$, respectively:

$$\begin{aligned} d(x) &= \min_{b \in B} \rho(x, b) \\ d'(x) &= \min_{a \in A} \rho(a, x). \end{aligned}$$

The function $d(x)$ gives the distance from x to its nearest neighbor in B, while $d'(x)$ gives the distance from x to its nearest neighbor in A. Distance transforms can be computed very efficiently on an integer grid with the help of a rendering engine and z-buffer [12].

7 Locating a Model within an Image: $\mathcal{G} = \mathcal{T}$

We now consider the model location problem with $\mathcal{G} = \mathcal{T}$; i.e., the model is only allowed to translate with respect to the image. For $\mathcal{G} = \mathcal{T}$ we define

$$\begin{aligned} f_A(t) &= h_L^{box}(A, B \oplus t) = L^{th}_{a \in A'} d(a - t) \\ f_B(t) &= h_K(B \oplus t, A) = K^{th}_{b \in B} d'(b + t) \\ f(t) &= \max(f_A(t), f_B(t)). \end{aligned}$$

$f_A(t)$, $f_B(t)$, and $f(t)$ give the partial box distance from the image to the model, the partial distance from the model to the image, and the partial box distance between the image and model, respectively, as functions of the translation t. These functions are valid for any $t = (x, y) \in \mathbf{R}^2$. Here, however, we will only consider translations that keep the translated model on the integer grid and with some overlap of the image. Thus we search in the discretized, finite translation space: $\{t = (x, y) \in \mathbf{Z}^2 : -w_M < x < w_I, -h_M < y < h_I\}$. Rasterizing d, d', f_A, f_B, f, we get

$$\begin{aligned} D[x, y] &= \min_{b \in B} \rho((x, y), b) \\ D'[x, y] &= \min_{a \in A} \rho(a, (x, y)) \\ F_A[x, y] &= L^{th}_{a \in A'} D[a_x - x, a_y - y] \\ F_B[x, y] &= K^{th}_{b \in B} D'[b_x + x, b_y + y] \\ F[x, y] &= \max(F_A[x, y], F_B[x, y]). \end{aligned}$$

Note that these functions/arrays are just sampled versions of their continuous counterparts.

A simple, but inefficient algorithm for the reporting problem is to use the above definitions to compute $F[x, y]$ for each pair (x, y) in the valid translation range and report all translations (x, y) for which $F[x, y] \leq \tau$. A few clever pruning techniques can be used to make the basic algorithm much more efficient. At the heart of each of these techniques is that we only want to know whether $F[x, y] \leq \tau$, not the exact value of $F[x, y]$.

7.1 Ruling Out Circles

An important property of the function $F_B[x, y]$ is that it does not decrease more rapidly than linearly.

Claim 1 $|F_B[x_1, y_1] - F_B[x_2, y_2]| \leq \|(x_1, y_1) - (x_2, y_2)\|$. *This is true for all values of f_1 (the fraction of nonzero model pixels considered).*

Proof. Suppose (a_x^1, a_y^1) is the closest point in A to (x_1, y_1). Then $D'[x_1, y_1] = \|(x_1, y_1) - (a_x^1, a_y^1)\|$. By the triangle inequality, $D'[x_1, y_1] + \|(x_2, y_2) - (x_1, y_1)\| \geq \|(x_2, y_2) - (a_x^1, a_y^1)\|$. But $D'[x_2, y_2] \leq \|(x_2, y_2) - (a_x^1, a_y^1)\|$ because it is the distance from (x_2, y_2) to its closest point in A. Therefore

$$D'[x_2, y_2] \leq D'[x_1, y_1] + \|(x_2, y_2) - (x_1, y_1)\|. \tag{1}$$

Now let $v_1 = F_B[x_1, y_1]$ and $v_2 = F_B[x_2, y_2]$. WLOG, assume $v_1 \leq v_2$. By definition of $F_B[x_1, y_1]$, there exists at least $K = \lfloor f_1 n \rfloor$ model points (b_x^j, b_y^j), $j = 1, \ldots, K$, which, after translation by (x_1, y_1), are each within distance v_1 of its closest image point; i.e.,

$$D'[b_x^j + x_1, b_y^j + y_1] \leq v_1 \quad 1 \leq j \leq K. \tag{2}$$

Combining (1) and (2), we get

$$D'[b_x^j + x_2, b_y^j + y_2] \leq v_1 + \|(x_2, y_2) - (x_1, y_1)\| \quad 1 \leq j \leq K.$$

Since $F_B[x_2, y_2]$ is the K^{th} ranked value of $D'[b_x + x_2, b_y + y_2]$ over $b \in B$, it follows that

$$v_2 = F_B[x_2, y_2] \leq v_1 + \|(x_2, y_2) - (x_1, y_1)\|.$$

■

From the claim, if $F_B[x_1, y_1] = v > \tau$, then $F_B[x, y]$ cannot be less than or equal to τ inside a circle C of radius $v - \tau$ centered at (x_1, y_1) (the shape of the circle depends on the norm used). Thus we can eliminate any translation (x, y) inside C from further consideration.

The claim says that moving the model a little bit will not change the model to image distance by too much. This makes sense because the minimum distances from each model point to image points will not change a lot (so neither will their K^{th} ranked value). The claim does not hold, however, for $F_A[x, y]$ because the image points under the model may change when we move the model by even a small amount.

7.2 Early Scan Termination

The idea is simple: Stop the computation of $F_B[x, y]$ when we are sure that the value will be greater than τ. As we probe $D'[b_x + x, b_y + y]$ with the points in B, we keep a count of the number of times we get a value that exceeds τ. If this count ever becomes greater than $n - K$, then the K^{th} ranked value of the probes will be greater than τ and we can deduce $F_B[x, y] > \tau$. If we assume all zero values for the remaining probes, we can compute a lower bound for $F_B[x, y]$. This bound can be used to eliminate a circle about (x, y).

7.3 Skipping Forward

This technique is used to eliminate large sections of a row when we compute the values $F_B[x, y]$ in row order. Let $D'_{+x}[x, y]$ be the distance in the increasing x direction to the closest location (x', y) for which $D'[x', y] \leq \tau$ (and ∞ if no such location exists).

Claim 2 $D'_{+x}[x, y] \geq D'[x, y] - \tau$.

Proof. If $D'_{+x}[x, y] = \infty$, then the claim is obvious. Otherwise $D'_{+x}[x, y] = \Delta x$ is finite and $D'[x + \Delta x, y] \leq \tau$. Applying (1) with $(x_2, y_2) = (x, y)$ and $(x_1, y_1) = (x + \Delta x, y)$, we get

$D'[x,y] \leq D'[x+\Delta x, y] + D'_{+x}[x,y]$. Using the fact that $D'[x+\Delta x, y] \leq \tau$ gives the desired result. ■

Now Define

$$G_B[x,y] = K^{th}_{b\in B} D'_{+x}[b_x + x, b_y + y].$$

Claim 3 *If $G_B[x,y] = 0$, then $F_B[x,y] \leq \tau$.*

Proof. This follows from the definition of $G_B[x,y]$ and the previous claim:

$$\begin{aligned} G_B[x,y] &= K^{th}_{b\in B} D'_{+x}[b_x + x, b_y + y] \\ &\geq K^{th}_{b\in B}(D'[b_x + x, b_y + y] - \tau) \\ &= (K^{th}_{b\in B} D'[b_x + x, b_y + y]) - \tau \\ &= F_B[x,y] - \tau. \end{aligned}$$

Thus $G_B[x,y] \geq F_B[x,y] - \tau$, and $G_B[x,y] = 0$ implies $F_B[x,y] \leq \tau$. ■

A nonzero value for $G_B[x,y]$ provides even more useful information.

Claim 4 *If $G_B[x,y] = \Delta x > 0$, then $F_B[x,y] > \tau, F_B[x+1,y] > \tau, \ldots, F_B[x+\Delta x - 1, y] > \tau$.*

Proof. Suppose the ranked values of $D'_{+x}[b_x + x, b_y + y]$ are

$$D'_{+x}[b^1_x + x, b^1_y + y] \leq \ldots \leq D'_{+x}[b^n_x + x, b^n_y + y].$$

Then $G_B[x,y] = \Delta x > 0$ implies

$$D'_{+x}[b^j_x + x, b^j_y + y] \geq \Delta x > 0 \quad j = K, \ldots, n.$$

By definition of $D'_{+x}[b^j_x + x, b^j_y + y]$, we must have

$$D'[b^j_x + x + l, b^j_y + y] > \tau \quad l = 0, \ldots, \Delta x - 1 (\geq 0) \quad j = K, \ldots, n.$$

Thus for each of $l = 0, \ldots, \Delta x - 1$, there are $n - K + 1$ probes of $D'[b_x + (x + l), b_y + y]$ which are greater than τ. The result follows. ■

For a fixed (x,y), terminating a scan as soon as possible when computing $F_B[x,y]$ or $G_B[x,y]$ will tend to produce lower bounds which are close to τ and 0, respectively. This means that only very small circles or portions of a row can be eliminated. Thus we terminate the scan for $F_B[x,y]$ only when we know $F_B[x,y] > \tau + r$ and for $G_B[x,y]$ when $G_B[x,y] \geq r$. This allows the elimination of a circle of radius r and r elements within a row, respectively.

To solve the reporting problem, we first generate a list of transformations for which the model to image distance $F_B[x,y]$ is $\leq \tau$. This can be done efficiently using the previously described speedup techniques. We then check which of these transformations also makes the image to model distance $F_A[x,y] \leq \tau$. This verification phase is usually not a bottleneck.

7.4 An Example

Figure 4 shows an example in which we try to locate translated versions of a window model within a hotel image (using the efficient algorithm described in the previous subsections). This example is similar in complexity and timing results to the examples given in Section VI of [12]. The authors report achieving speedups of a factor of 1000 or more in some cases over the simple, brute-force algorithm.

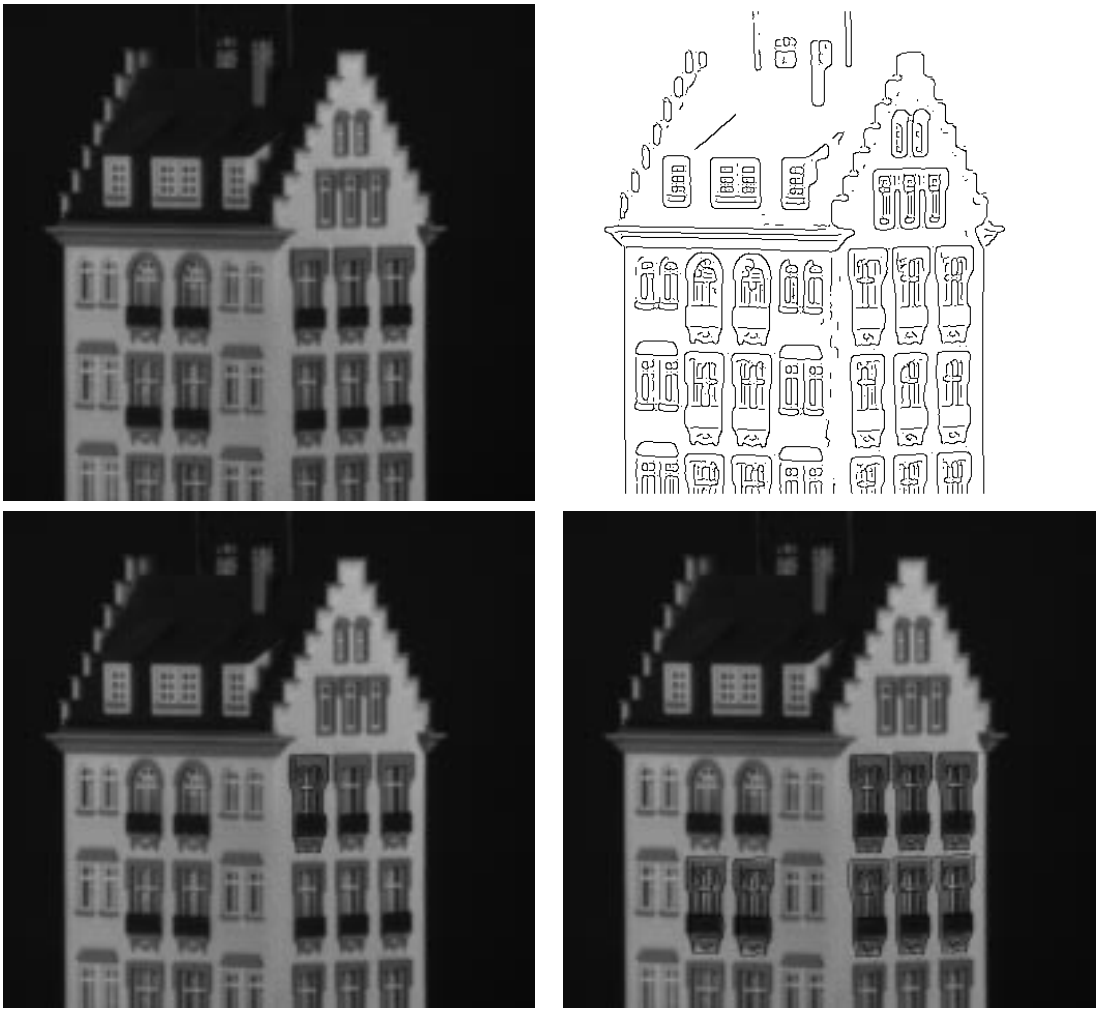

Figure 4: (top-left) The hotel image is 512×480. (top-right) Canny edge detection yields an image point set with 15330 points. (bottom-left) The outlined window model is 44×103 and contains 545 points. (bottom-right) Using $\tau = 3$ pixels and $f_1 = f_2 = 0.90$, eight translated instances of the window model were located within the hotel image in just 18.3 seconds (on an SGI Indigo workstation). Many similar translations were reported for each of the eight windows located. Here we show a representative translation of the model for each such window location. The code used to perform this experiment is available via anonymous ftp from cs.cornell.edu in the /pub/wjr directory.

8 Locating a Model within an Image: $\mathcal{G} = \mathcal{A}_2$

Weak perspective images of a shallow object (thickness small compared to camera distance) in two different poses are related approximately by an affine transformation. In paper 4, Rucklidge considers the problem of locating an affinely transformed model within an image (i.e., $\mathcal{G} = \mathcal{A}_2$). As in the previous section, we concentrate on the model to image distance. In the following discussion, t denotes a planar affine transformation in discretized transformation space.

Let $P_B[t]$ denote the fraction of all values probed (with no early scan termination) to compute $F_B[t]$ which are $\leq \tau$. Note that $P_B[t] \geq f_1$ iff $F_B[t] \leq \tau$. The discretized search space is six-dimensional, so developing efficient search techniques is crucial. The search space is limited to a finite (but usually very large) number of transformations by the user. A multi-level cell decomposition strategy is used to eliminate a large number of transformations without explicitly considering each one. The initial space of transformations to search is tiled with rectilinear cells of equal size. For each cell R, we determine an upper bound $P_B[R]$ for any $P_B[t]$ with $t \in R$. If this value is $< f_1$, then we eliminate all the transformations in R from further consideration. Otherwise, it is possible that there is a transformation $t \in R$ for which $F_B[t] \leq \tau$ and we mark the cell R "interesting". After considering all the cells at the current level, we subdivide each of the interesting ones and repeat the process. The end result is a set of interesting cells containing only one transformation. These transformations satisfy $F_B[t] \leq \tau$. This search process is a breadth first search in the tree representing the recursive cell decomposition.

Rucklidge also considers the case when only a single match is required: either any match, or the best distance match for a fixed fraction, or the best fraction match for a fixed distance. In these cases, the search can be done much more efficiently by doing a best-first search in which we investigate the most promising interesting cells first. Interesting cells are ranked by the values $P_B[R]$ which were used to label the cells interesting. The greater the value of $P_B[R]$, the more likely (we hope) R is to contain a transformation that satisfies $F_B[t] \leq \tau$. In the any match case, we stop when a match is found. During the search for the best distance match for a fixed fraction f_1, each time we find t such that $F_B[t] \leq \tau$, we decrease the value of τ to $F_B[t]$ (and continue the search). Decreasing the value of τ used in computing $P_B[R]$ strengthens the pruning process. Similarly, in the search for the best fraction match, we increase f_1 as the search progresses.

Examples in [15] show that the method is accurate - it correctly locates affinely transformed versions of a model within an image. The time taken is quite low (seconds) when the point set sizes are small or when the transformation range to be searched is strongly restricted. For large point sets and weak restrictions on the transformation space, times increase dramatically. In one example with 14834 image points and 1473 model points, the algorithm required 39 minutes 39 seconds to find the two affinely transformed instances of the model within the image. The author points out: "While this time is not as good as might be hoped for, it should be noted that the number of points in the image and model are many times larger than can be handled by other methods that search under affine transformation ...." [15]

9 Conclusion

We have considered three types of Hausdorff point set matching problems: the decision problem, the optimization problem, and the reporting problem. The decision problem questions the existence of a transformation that makes the Hausdorff distance less than or equal to some given threshold, while the optimization problem seeks the transformation which minimizes the Hausdorff distance. This minimum Hausdorff distance is the smallest threshold for which the answer

to the decision problem is *yes* . In a parametric search setting, we can obtain an optimization problem algorithm by using a decision problem algorithm to compare hypothesized thresholds with the minimum threshold that we seek. We showed how to rephrase the decision problem for the group of translations as a problem in determining whether an intersection of translates of a union of "spheres" is empty. The shape of the spheres depends upon the underlying point distance metric. Details for the optimization problem for Euclidean motion in the plane with the L_2 point distance were presented.

The reporting problem asks for a list of transformations that makes the Hausdorff distance less than or equal to some threshold. An example application is the problem of finding all occurrences of a binary model within a binary image. In this application, it was necessary to modify the Hausdorff distance definitions to handle noisy data sets, images containing many objects, and partially occluded objects. The main modification was to replace the max in the "theoretical" definitions with a percentile measure. The problem of locating translated versions of a model was closely examined. Pruning techniques such as ruling out circles and early scan termination were presented. We also briefly considered the problem of locating an affinely transformed model. The high dimensionality of the transformation space makes an efficient search strategy essential. The main tool discussed was a recursive cell decomposition strategy in which we question whether a cell in transformation space can possibly contain a transformation which we want to report. If not, we never need to consider the transformations inside the cell. If so, we subdivide the cell and repeat the process. The same framework can be used when we want to return only a single match, say any match or the best match by distance. In this case, the search will be much more efficient if we subdivide and search the most promising cells first.

References

[1] L. Paul Chew, Dorit Dor, Alon Efrat, and Klara Kedem. Geometric pattern matching in d-dimensional space. Unpublished.

[2] L. Paul Chew, Michael T. Goodrich, Daniel P. Huttenlocher, Klara Kedem, Jon M. Kleinberg, and Dina Kravets. Geometric pattern matching under euclidean motion. In *Proceedings of the Fifth Canadian Conference on Computational Geometry*, pages 151–156, 1993.

[3] L. Paul Chew and Klara Kedem. Improvements on geometric pattern matching problems. In O. Nurmi and E. Ukkonen, editors, *Algorithm Theory - SWAT '92*, pages 318–325. Springer-Verlag, July 1992.

[4] Richard Cole. Parallel merge sort. *SIAM Journal of Computing*, 17(4):770–785, August 1988.

[5] Greg N. Frederickson. Optimal algorithms for tree partitioning. In *Proceedings of the Second Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 168–177, 1991.

[6] Greg N. Frederickson and Donald B. Johnson. Finding kth paths and p-centers by generating and searching good data structures. *Journal of Algorithms*, 4:61–80, 1983.

[7] Leonidas J. Guibas. Parametric search. In *CS368: Geometric Algorithms*. Stanford University, 1994.

[8] Leonidas J. Guibas and Lyle Ramshaw. The wonders of segment trees. In *CS348a: Computer Graphics – Mathematical Foundations*. Stanford University, 1993.

[9] L.J. Guibas, J.S.B. Mitchell, and T. Roos. Voronoi diagrams of moving points in the plane. In *Seventeenth International Workshop on Graph-Theoretic Concepts in Computer Science*. Springer-Verlag, June 1991.

[10] Daniel P. Huttenlocher, Klara Kedem, and Jon M. Kleinberg. On dynamic voronoi diagrams and the minimum hausdorff distance for point sets under euclidean motion in the plane. In *Proceedings of the Eighth Annual ACM Symposium on Computational Geometry*, pages 110–119, 1992.

[11] Daniel P. Huttenlocher, Klara Kedem, and Micha Sharir. The upper envelope of voronoi surfaces and its applications. *Discrete and Computational Geometry*, 9(3):267–291.

[12] Daniel P. Huttenlocher, Gregory A. Klanderman, and William J. Rucklidge. Comparing images using the hausdorff distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(9):850–863, September 1993.

[13] Nimrod Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *Journal of the ACM*, 30(4):852–865, October 1983.

[14] Mark H. Overmars and Chee-Keng Yap. New upper bounds in klee's measure problem. *SIAM Journal of Computing*, 20(6):1034–1045, December 1991.

[15] William J. Rucklidge. Locating objects using the hausdorff distance. In *Fifth International Conference on Computer Vision*, pages 457–464, 1995.