On Solving 2D and 3D Puzzles Using Curve Matching

Weixin Kong Benjamin. B. Kimia
Brown University
Providence, RI 02912

Abstract

We approach the problem of 2-D and 3-D puzzle solving by matching the geometric features of puzzle pieces three at a time. First, we define an affinity measure for a pair of pieces in two stages, one based on a coarse-scale representation of curves and one based on a fine-scale elastic curve matching method. This re-examination of the top coarse-scale matches at the fine scale results in an optimal relative pose as well as a matching cost which is used as the affinity measure for a pair of pieces. Pairings with overlapping boundaries are impossible and are removed from further consideration, resulting in a set of top valid candidate pairs. Second, triples arising from generic junctions are formed from this rank-ordered list of pairs. The puzzle is solved by a recursive grouping of triples using a best-first search strategy, with backtracking in the case of overlapping pieces. We also generalize aspects of this approach to matching of 3-D pieces. Specifically, ridges of 3-D fragments scanned using a laser range finder are detected using a dynamic programming method. A pair of ridges are matched using a generalization of the 2-D curve matching approach to space curves by using an energy solution involving curvature and torsion, which are computed using a novel robust numerical method. The reconstruction of map fragments and broken tiles using this method is illustrated. [1]

1 Introduction

The aim of this paper is to provide an automatic method for 2-D and 3-D jigsaw puzzle solving. This is an interesting problem from both theoretical and practical points of view with applications in archaeology, art restoration, failure analysis and so on. In these applications, we may encounter a large number of irregular fragments resulting from one or several broken objects. The automatic reconstruction of the original objects would alleviate the manual effort, which is tedious and laborious.

[1]We gratefully acknowledge the support of NSF KDI grant BCS-9980091

The jigsaw puzzle problem is approached here in two stages: local shape matching followed by global search and reconstruction. First, local shape matching aims to find likely candidate pairs for adjacent fragments using only local shape information. Second, in finding a global solution, ambiguities resulting from local shape matching are resolved and the pieces are merged together. While in solving 2-D and 3-D puzzle, we use different shape matching methods, the search step is shared by both.

Previous works on solving jigsaw puzzle mainly emphasize the first step for 2-D problems. Puzzle pieces are often represented by their boundary curves and local shape matching is usually achieved by curve matching. Since matching between two pieces usually occurs over only a small fraction of their respective boundaries, *partial* curve matching is needed. Some approaches [3, 11, 4] split each boundary curve first at breakpoints and then match subcurves. However, the identification of breakpoints is not a trivial problem. More general partial curve matching algorithms were proposed in [17], where 2-D boundary curves are represented by shape feature strings which are obtained by a polygonal approximation. The matching stage finds the longest common sub-string and is solved by geometric hashing. The feature-based matching methods are fast, so most puzzle solving methods use this scheme. The drawback is that they can't give detailed matching of curves, which are required to determine a "tooth-in-tooth match", or firmly rule out overlapping regions. Thus, they break down when the number of puzzle pieces becomes large. Dense curve matching methods [2, 18, 13] can match curves at a fine scale. But due to their high computation cost, they have not yet been used in puzzle solving. Generalization of the 2-D problem to 3-D can be found in [6, 16]. However, a main problem of 3-D curve matching is that the computation of curvature and torsion involves up to third-order derivatives, which are not robustly computed by traditional numerical techniques.

Since local shape analysis produces ambiguous matches, search is required to dis-ambiguite the global picture. There is surprisingly little work on the search problem for puzzle solving, to the best of our knowledge. In [11], a new

0-7695-1272-0/01 $10.00 © 2001 IEEE

shape representation method is used to solve a puzzle of 4 pieces. When the number of puzzle pieces increases, the ambiguity multiplies exponentially, thus requiring a global search technique. This is an interesting combinatorial problem which is NP complete. In [4], authors reported an algorithm that can solve a large puzzle, but with strict restrictions on the shape of puzzle pieces and the *apriori* knowledge of the shape of the outer frame of the puzzle. However, this is not practical in many real applications. In [3, 16], all pairs of pieces are searched and the best matched pair is selected and merged to form a new piece. The process is repeated until there is only one piece left. Since invalid matching may occur in the merging process, backtracking is used. But typically there are not enough constraints to get good partial merging results at first, so backtracking occurs often and the process is not efficient. Alternatively, computers may be used only in the local shape analysis stage, while using user-interaction for the global search aspect [7].

This paper is organized as follows. Section 2 presents a two-stage coarse-to-fine strategy for finding potential pair matches by using curve matching. Potential pair matches with overlapping region are ruled out and the resulting rank-ordered list is used for searching for a solution. Section 3 describes a generalization to 3-D. The main difficulty in generalizing the curve matching process to space curves is that it requires the robust computation of curvature and torsion, involving up to second and third order derivatives, respectively. Finally, Section 4 describes the global search, which is a best-first search with backtracking as used in [16]. The novelty is the idea that generic breaks in puzzles only produce T and Y junctions, thus motivating us to merge three pieces at a time in this process. This significantly reduces the number of matches due to those ruled out by violating the overlapping constraint. We present results for solving 2-D puzzles and results for matching break curves on 3-D puzzles.

2 Pairing of 2-D puzzle pieces

In this section, we present a method for computing the affinity of two puzzle pieces using a partial curve matching technique and show results on real images. Our matching algorithm is based on [13] where the notion of an alignment curve is used to represent a correspondence between two curves. The space of alignment curves is searched for one optimizing an elastic energy.. To reduce the computational complexity, we first re-sample the curves using a polygonal approximation. We get coarse alignment using dynamic programming on the reduced version of the curves. Then, we go back to the original curves to get fine-scale alignment using dynamic programming again. The matched pairs of points are used to find an optimal 2-D Euclidean transformation (translation and rotation). The residual distances of corresponding points give the similarity measure of the matching of the two partial curves. Since each fine-scale alignment of a potential pair must satisfy the *overlap constraint*, namely, that two pieces can not occupy the same position in space, we rule out those pairings which violate it. We then present the rank-ordered results of valid matches to the search algorithm, which in practice is the top ten such valid pairings. The processing stages are illustrated in Figure 1 and discussed in further detail below.

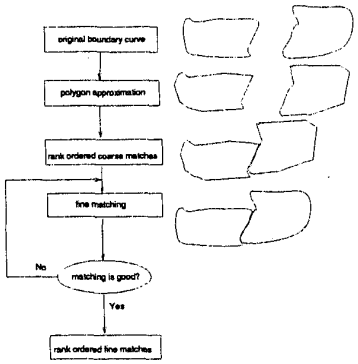

Figure 1. Steps for partial curve matching

Preprocessing and Contour Representation: We assume a binary representation of each piece is available, *e.g.*, using scanning and segmentation. We represent each piece by its boundary contours using a number of sample points. Consider the pairing of two pieces $A = (a_1, a_2, ..., a_M)$ and B=$(b_1, b_2, ..., b_N)$ represented by the cyclic list of M and N points of the two contours, respectively. If A and B can be fit together, they must have a common sub-boundary that occurs as sublist $(a_i, a_{i+1}, ..., a_{i+l})$ in A and also as sublist $(b_j, b_{j+1}, ..., b_{j+l'})$ in B. Because every point can be the starting or ending point of a match, the brute-force matching of all subsegments has complexity of $O(M^2N^2)$. To reduce this complexity, we first re-sample the curves using a polygonal approximation, as described in [12], resulting in a coarse-scale version of the original curve. The reduced number of points depends on the chosen scale and the shape of the contour, but typically it is less than 50 as compared to 200-300 points on the original curve. Another problem to note is that the common sub-boundaries of A and B traverse in opposite directions. So for each curve, in the process of curve matching, we assume one of the curves has been reversed. Figure 2 gives the images of several pieces taken from a map of the United States and the contours extracted from these pieces, where the red points are the vertices of polygon to approximate the original contour.

Dynamic Programming Matching of Open Curves: First we assume that the pairs of starting and ending points on the two contours to be matched are known. The problem is then to match the entire segments of two open curves. This can be solved by using the matching algorithm based

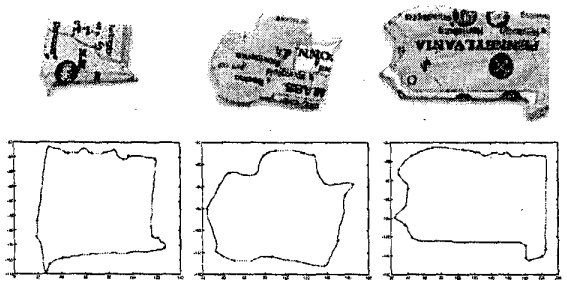

Figure 2. (top) Images of several puzzle pieces taken from a map (bottom) Curves extracted from the puzzle pieces and their coarse-scale representation.

on dynamic programming which is described in [13].

Generation of Initial Candidates for Closed Curves: In the problem of fragment re-assembly, every broken piece is represented by its closed contour. Since the pairs of start and end points of the matching segments are not known, the open curve matching method can not be used directly. Some authors propose to split the closed contour to get partial curves to be matched [3, 11]. Their assumption is based on the fact that the junction point is generally formed by three break lines or one break line and two border lines. The junction points are often the high curvature points on the contour. This assumption is valid only in the case of "Y" junction. But in practice, two of the three lines forming a junction often make an angle close to 180^o, *i.e.*, form a "T" junction. In this case, a high curvature point occurs only on two of its three incident piece contours. Also, contours of the pieces may have high curvature points that don't correspond to junctions. Thus, using high curvature points to split the curves to get partial curves is not suitable for our problem.

One solution is that we search all the possible combination of pairs of start and end points. Then the matching corresponding to the minimal cost is optimal. The problem of this method is: *I* computation complexity is high even using the polygonal approximation, and *ii* the dynamic programming cost increases without bound with curve length. Normalizing the cost with respect to curve length is not trivial [8].

Here we propose another method to find the possible pairs of start and end points. Our method is based on dynamic programming again. But we only need to search all pairs of start points. We begin with the corresponding staring points and extend the correspondence between the remaining un-matched points, preserving their cyclic order. We introduce a negative cost, such that the total cost of a match should be decreasing only if the extension is good. If the total cost begins to increase, it means that we should not extend the correspondence any further. Now, we give the detail of the algorithm.

Let $A = (a_1, a_2, ..., a_M)$ and B=$(b_1, b_2, ..., b_N)$ be the cyclic list of M and N points of the two contours respectively. The algorithm builds a DP table of costs of partial matches. The table has M rows and N columns at most. Denote the start point pairs by (a_i, b_j). The rows of a DP table are indexed by m, $1 \leq m \leq M$, and its columns are indexed by n, $1 \leq n \leq N$. The entry at the intersection of row m and column n is referred to as the $cell(m, n)$. A link between cells (m_{w-1}, n_{w-1}) and (m_w, n_w) denotes the matching of the merged sequences of points $a(m_{w-1}|m_w)$ and $b(n_{w-1}|n_w)$. Here, $a(m_{w-1}|m_w)$ denotes the sequence of points $(a(m_{w-1}), a(m_{w-1}+1), ..., a(m_w))$, and similarly for $b(n_{w-1}|n_w)$. A path is a linked sequence of cells $((m_0, n_0), (m_1, n_1), ..., (m_t, n_t))$, indicating a partial match, where $m_0 = i, n_0 = j, m_0 < m_1 < ... < m_t, n_0 < n_1 < ... < n_t$. Each entry $cell(m_w, n_w)$ contains the following values: $cost, u_w, v_w$, where em cost is the partially accumulated match cost up to that entry, u_w and v_w are the indices of the parent entry of $cell(m_w, n_w)$ (*i.e.*, $u_w = m_{w-1}, v_w = n_{w-1}$) and are used to trace back a path. The cost $D(A, B, i, j)$ of matching contour A and B with staring points a_i and b_j is denoted as

$$D(A, B, i, j) = \sum_{w=1}^{t} \psi(a(m_{w-1}|m_w), b(n_{w-1}|n_w)),$$

where the function $\psi(a(m_{w-1})|m_w), b(n_{w-1})|n_w))$ represents the cost of its two arguments and consists of two terms

$$\psi(a, b) = \eta(a, b) + R\gamma(a, b).$$

The first term is the stretching cost related to the length of curve segment. We use l_{Aw} and l_{Bw} to denote the length of segment $a(m_{w-1}|a(m_w)$ and $b(n_{w-1}|n_w)$, respectively. We define c_w as the ratio of l_{Aw} and l_{Bw}. The stretching cost is defined as

$$\eta(a(m_{w-1})|m_w), b(n_{w-1})|n_w)) = \begin{cases} -2.0/(c_w + 1/c_w), & c_1 < c_w < c_2; \\ (c_w + 1/c_w)/2, & otherwise, \end{cases}$$

where we have used $c_1 = 0.8, c_2 = 1.2$.

The second term is the bending cost related to the orientation of the curve segments. We use S_{Aw} and S_{Bw} to denote the angles of segments $a(m_{w-1})|m_w)$ and $b(n_{w-1}|n_w)$, respectively. To achieve rotation invariance, we compare the orientation change of the contour. We define α_w as the difference of orientation change, $\alpha_w = (S_{Aw} - S_{Aw-1}) - (S_{Bw} - S_{Bw-1})$. The bending cost is defined as

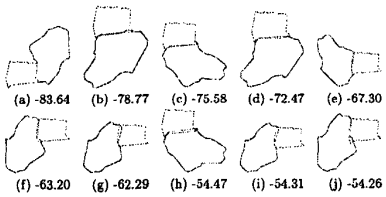

Figure 3. The top ten rank-ordered candidate pairs after the matching stage, for matching the puzzle pieces corresponding to "New Hampshire" and "Maine".

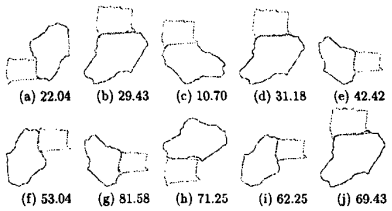

Figure 4. This figure illustrates the refinement of the results of the rank-ordering of Figure 3 based on a fine-scale alignment, which gives a *new* cost, as well as allowing for the removal of alignments which partially overlap. The ordering matches that of Figure 3. Observe that (g), (h), (j) in Figure 3 violate the overlap constraint and they are replaced by candidates further down the coarse-scale rank-ordered list. The optimal alignment is shown in (c).

$$\gamma(a(m_{w-1})|m_w), b(n_{w-1})|n_w)) = \begin{cases} -cos(\alpha_w), & c_3 < \alpha_w < c_4; \\ |\alpha_w|, & otherwise. \end{cases}$$

where we have used $c_3 = -\pi/6$, $c_4 = \pi/6$. The constant R represents the relative importance of the stretching and bending costs.

We can see that if two segments are sufficiently similar, their matching cost is negative. So the minimum of total cost in one row of DP table should be decreasing when we fill the DP table row by row. If the minimum of total cost in one row begins to increase, it means that we should not extend the correspondence any further and finalize the pairs of end points. This gives a sequence of matching segments for every pair of start points and the corresponding matching cost. We can rank-order and then select K sequences of segments with minimal matching cost as candidates for further matching. From our experiments, we find the correct match is included in the top 10 coarse matching results, *i.e.*, K=10. Then, the fine-scale curve matching in the original resolution uses the specified start and end corresponding points.

2-D curve matching results: We give the results of our partial curve matching algorithm on contours extracted from real broken piece images. Figure 3 gives the top 10 matches between polygonal approximations of two map piece contours using our algorithm. Figure 4 gives the finer matching results based on the original resolution. The cost here is from the residual distance of correspondence points after optimal transformation. Comparing Figure 3 and Figure 4, it is evident that fine-scale matching gives a more accurate alignment. Also, pairs (g,h.j) in Figure 3 are replaced because they violate the overlapping constraint after fine alignment. The top-ranking pairing in Figure 4 which has the minimal cost is the correct pairing in the original puzzle.

3 Space Curve Matching

This section discusses a generalization to the problem of matching two space curves. These space curves are typically the break curves of 3-D puzzle, or its ridges. We first give a distance metric based on speed, curvature, and torsion. Computing curvature and torsion, however, introduces coordinate derivatives up to the third-order, in comparison to up to second-order derivatives in 2D. The estimation of these quantities is very unstable in the presence of noise or when the curve is not uniformly sampled. Thus, we propose a method to improve the accuracy of curvature and torsion estimation based on the "ENO scheme". Furthermore, we give an implicit method to compute the distance metric instead of using curvature and torsion explicitly to avoid the third-order derivatives. At the end of this section, we give some experimental results on space curve matching using our methods.

Distance Metric for Space Curve: Since our curve matching is used to re-assemble different surface pieces, we need a coordinate-independent description of the ridge curve. From the local theory of curves in differential geometry [10], we know that two different curves parameterized by their arc length, have the same torsion and curvature function, $\kappa(s)$ and $\tau(s)$, if and only if they are the same curves (up to a rigid motion, *i.e.*, translation and rotation); Thus, the intrinsic representation we use is one of curvature and torsion.

The 3-D curve matching algorithm is based on a generalization of the 2-D curve matching algorithm using dynamic programming [2, 13], but modified to include torsion in the cost function. We define the cost of matching two infinites-

imal segments on the two curves as

$$\mu[g] = \int_C [|g'-1|+R_1|\bar{k}(\bar{s})g'(s) - \kappa(s)| + R_2|\bar{\tau}(\bar{s})g'(s)-\tau(s)|]ds,$$

where $g(s)$ is the function which maps the curve $C(s)$ to $\bar{C}(\bar{s})$, *i.e.*, $\bar{s} = g(s)$.

Curvature and Torsion Estimation Using ENO: To use the above distance metric, we need to calculate the curvature and torsion at each discrete sample point on the curve. For a curve $C(s)$ parameterized by its arc length s, curvature and torsion are defined as

$$\kappa = |C''|, \tau = \frac{1}{\kappa^2}[C'\, C''\, C'''],$$

where prime denotes differentiation with respect to arc length s, and the square brackets denotes the determinant. Curvature involves coordinate derivatives up to the second order. Torsion involves coordinate derivatives up to the third order. If we use ordinary method to compute, these high order derivative estimates are very unstable when noise exists or the sample points are not uniformly separated. To illustrate it, consider the cylindrical spiral $(x(s), y(s), z(s))$=

$$(a\cos(\frac{s}{\sqrt{a^2+b^2}}), a\sin(\frac{s}{\sqrt{a^2+b^2}}), \frac{bs}{\sqrt{a^2+b^2}}),$$

where $a = 0.1$, $b = 0.2$. The curvature and torsion are constants, $\kappa = \frac{a}{a^2+b^2} = 2$ and $\tau = \frac{b}{a^2+b^2} = 4$. Now, we discretize the curve from $s = 0$ to $s = 2$ to 100 segments. The length of every segment is $0.02 \pm rand(0.01)$, where $rand(0.01)$ is a random number in the range $[0, 0.01]$. We can get 101 sample points and 101 tuples (x_i, y_i, z_i, s_i), $i = 0, 1, ..., 100$. We use differences to approximate differentials.

Similarly we can get high order derivatives and the curvature and torsion. We compute the average, minimal and maximal value of the curvature and torsion at the 101 points. The result is as the following:

$$\kappa_{avg} = 1.98,\ \kappa_{max} = 2.72,\ \kappa_{min} = 1.57,\ \kappa_{std} = 0.22;$$

$$\tau_{avg} = 4.07,\ \tau_{max} = 5.99,\ \tau_{min} = 2.59,\ \tau_{std} = 0.64.$$

We can see that the error is as large as 36% for curvature and 50% for torsion. The mean error is 11% and 16%, respectively. Figure 5(a) gives a plot of curvature and torsion function. It is clear that these estimates can not be directly used for curve matching.

We propose to improve the accuracy of curvature and torsion estimation using ENO, the Essentially Non-Oscillatory schemes, which were introduced by Harten *et al.* [5] [1], later made more efficient by Shu and Osher [14], and extended to shock-placing ENO in [15] It can be used to smoothly interpolate the discrete data while preserving the disconuity of the data at the same time.

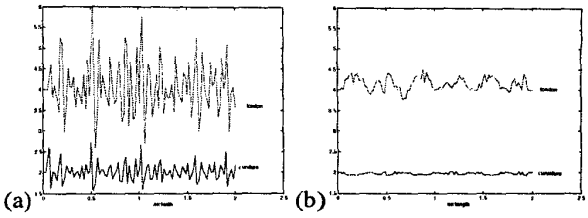

Figure 5. (a) Curvature and torsion versus arc-length using ordinary difference method. (b) Curvature and torsion versus arc-length using ENO method.

Specifically, to calculate curvature and torsion using the third-order ENO, consider the interpolation polynomial of degree 3, coefficients a_3, a_2, a_1 and a_0, and derivatives

$$x' = a_1 + 2a_2x + 3a_3x^2,\ x'' = 2a_2 + 6a_3x,\ x''' = 6a_3.$$

For a 3-D curve, we can get derivatives of three coordinate components, respectively, to compute curvature and torsion. The curvature and torsion results from the above example curve using ENO are

$$\kappa_{avg} = 1.99,\ \kappa_{max} = 2.03,\ \kappa_{min} = 1.95,\ \kappa_{std} = 0.02;$$

$$\tau_{avg} = 4.04,\ \tau_{max} = 4.40,\ \tau_{min} = 3.66, \tau_{std} = 0.15.$$

The maximum error is 2.5% for curvature and 10% for torsion and the mean error is 1.3% and 3.9% respectively, figure 5(b). This result is a significant improvement considering the third derivative derivatives required to calculate torsion.

An Implicit Method for Curve Matching: The use of an ENO scheme improves the accuracy of high-order derivative estimation considerably. However, we observe that we can reduce the order of the derivatives by one using an implicit method. The main idea is that instead of using κ directly in the energy function, we can use $\frac{\Delta\theta}{\Delta s}$ in the integral, and similarly for torsion.

Cost of matching ridges in the database of seven broken pieces of a pot: We have used this algorithm to match the ridges on several broken pieces of pot-shaped object. Ridges of 3-D fragments, which are scanned using a laser range finder, are detected using a dynamic programming method [9], Figure 6. Figure 7 gives the matching costs of 21 ridges from 7 pieces. The first digit indicates the piece number(1 to 7) and the second digit indicates the ridge number, *i.e.* 63 indicates ridge number 3 of piece number 6. All costs below 10 are shown in boldface (except the trivial case

of 0 cost). The correct matches are (11-43, 12-61, 13-21, 22-71, 23-31, 32-51, 42-62, 52-73). All, except one (52-73), are picked out by this method (The cost of matching 52-73 is 10.31, slightly above threshold). Three pairs are also below threshold (12-22, 12-71, and 22-61) although they are not correct matches. These must be dealt with in the course of global search. This table indicates that there is sufficient information in matching ridges allowing for the global stage to perform the correct reconstruction.

4 Reconstruction of Broken Pieces Using Best First Search

After rank-ordering pairs of broken pieces, we want to stitch together all the pieces to reconstruct the original object. In previous approaches to jigsaw puzzle solving, it was assumed that all the pieces of a puzzle are significantly different from each other such that local shape analysis sufficiently disambiguates potential local matches for finding a global solution. This assumption is violated as the number of pieces is increased to the range of one to two dozen pieces or higher. In our approach, we use triplets to pieces to narrow down the search space. Specifically, given N pieces of a broken object, we match each piece with all other pieces using the methods described in the last two sections. Each comparison may generate several possible matches between the same pair of contour of pieces. As some pieces are similar to each other, we may get for the same partial boundary of a piece several possible matches with partial boundaries of a number of other pieces. One of the fundamental requirements of our local shape analysis is not to miss any correct local match that is part of the global solution. As a consequence, many wrong matches that are not part of global solution are retained. Resolving this kind of ambiguity is the problem to be solved in constructing a global solution.

Best First Search with Backtracking: Table 1 gives a best-first procedure with backtracking that removes two pieces from the piece database every time. Initially, we call this algorithm with the complete list of contour of pieces. If there are more than two pieces in the list, we generate all local triple groups using local shape analysis algorithm. Then, all local triple groups are ordered such that the most likely local triple group will be checked first. Each local group will be tested to see if it can construct a global solution. If it can't, the algorithm backtracks.

Sorting of Possible Triple Groups: Our search algorithm is based on best-first search. So we need to sort all possible local triple groups according to some measure to find the correct local match earlier, resulting in significant computational savings. The measures we use include several characteristics of local match:

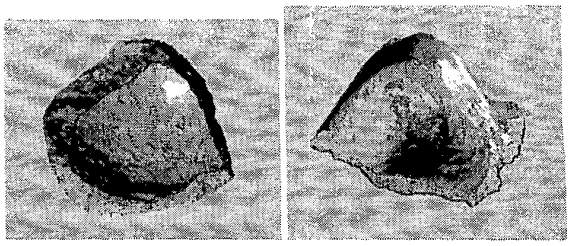

Figure 6. Ridges of 3-D broken pot pieces are computed using the technique described in [9] and highlighted in blue.

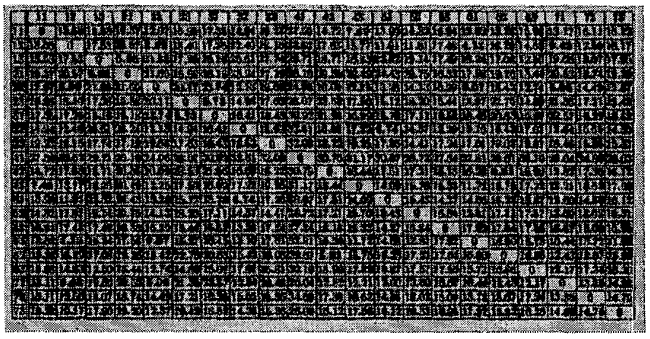

Figure 7. Space curve matching results.

1. Dissimilarity measure: The residual distance of corresponding points after the optimal transformation. The smaller this distance, the better the match.

2. Length measure: There may be many very short matching segments with small dissimilarity cost. But we prefer longer matching segments.

3. Diagnostic measure: This is to measure the confidence we can get from local shape analysis. If the two matched segments are almost straight, this match may be excellent in quality but does not convey as much confidence as two equally matching but jagged segments do. Similarly, if two segments change their orientation rapidly and still match with small distance, the match has more possibility to be a correct local match.

Experimental Results: We have experimented on several broken objects including a map puzzle and several ceramic fragments. The reassembled map is shown in Figure 10. The 7 pieces of a broken ceramic tile are shown in Figure 8 with the reassembled tile shown in Figure 9 Other ceramic tiles with 18 pieces and with 25 pieces are shown in Figure 11.

algorithm: puzzle-solving(piecelist)
input: N piece contours represented by their surrounding contours
output: FAIL if no solution exists; otherwise a joint-piece by merging three pieces in piecelist
procedure:

```
PUZZLE-SOLVING(piecelist)
 1
 2  if number(piecelist) == 1or2
 3    then if number(piecelist) == 1
 4           then output piecelist, return SUCCESS
 5           else merge the two pieces
 6    else
 7         do local shape analysis to find all local triple groups
 8         if no local match is found
 9           then return FAIL
10           else
11                sorting all local groups
12                s = FAIL
13                for every local group
14                do
15                    merge the three pieces of the local group
16                    remove the three pieces from piecelist
17                    insert the new merged piece into piecelist
18                    s = puzzle - solving(newpiecelist)
19                    if s == SUCCESS
20                      then exit for-loop
21
22                return s
```

Table 1. Best-first search with backtracking

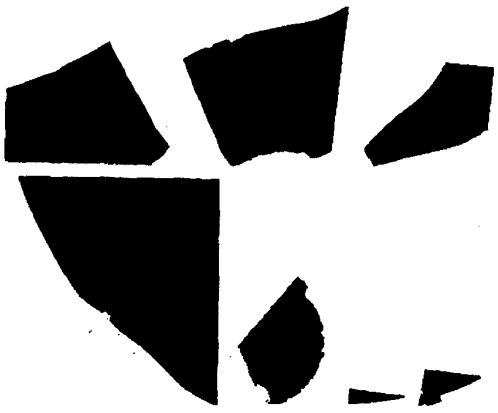

Figure 8. A ceramic tile is broken into seven pieces.

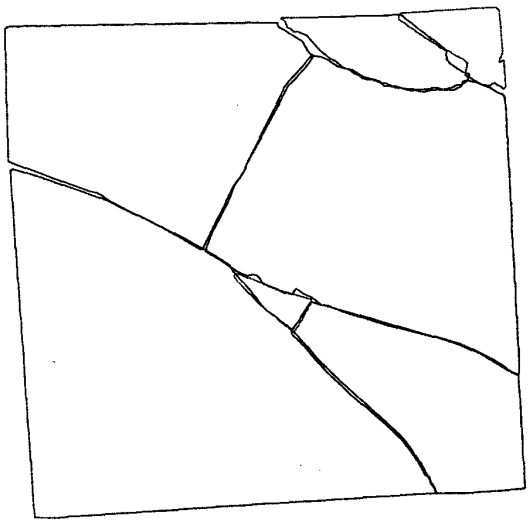

Figure 9. The result of re-assembling the tile pieces presented in Figure 8.

References

[1] A. Harten, S. Osher, B. Engquist and S. Chakravarthy. Uniformly high order accurate essentially non-oscillatory schemes, III. *Journal of Computational Physics*, 71:231–303, 1987.

[2] I. Cohen, N. Ayache, and P. Sulger. Tracking points on deformable objects using curvature information. *ECCV*, pages 458–466, 1992.

[3] H. Freeman and L. Garder. Apictorial jigsaw puzzles: the computer solution of a problem in pattern recognition. *IEEE Trans. Elec. Comp.*, 13:118–127, 1964.

[4] H. Wolfson, A. Kalvin, E. Schonberg and Y. Lambdan. Solving jigsaw puzzles by computer. *Annales of Operations Research*, 12:51–64, 1988.

[5] A. Harten. ENO schemes with subcell resolution. *J Comp. Phy.*, 83:148–184, 1989.

[6] E. Kishon and H. Wolfson. 3-d curve matching. In *AAAI workshop on spatial reasoning and multi-sensor fusion*, pages 250–261, 1987.

[7] H. C. G. Leitao and J. Stolfi. Automatic reassembly of irregular fragments. Technical Report IC-98-06, Institute of Computing, University of Campinas, 1998.

[8] A. Marzal and E. Vidal. Computation of normalized edit distance and applications. *PAMI*, 15:926–932, 1993.

[9] N. Khaneja, M. I. Miller and U. Grenander. Dynamic programming generation of curves on brain surfaces. *PAMI*, 20:1260–1265, 1998.

[10] B. O'Neill. *Elementary Differential Geometry*. Academic Press, Inc, 1966.

[11] G. M. Radack and N. I. Badler. Jigsaw puzzle matching using a boundary-centered polar encoding. *CGIP*, 19:1–17, 1982.

[12] P. L. Rosin and G. A. W. West. Non-parametric segmentation of curves into various representations. *PAMI*, 17:1140–1153, 1995.

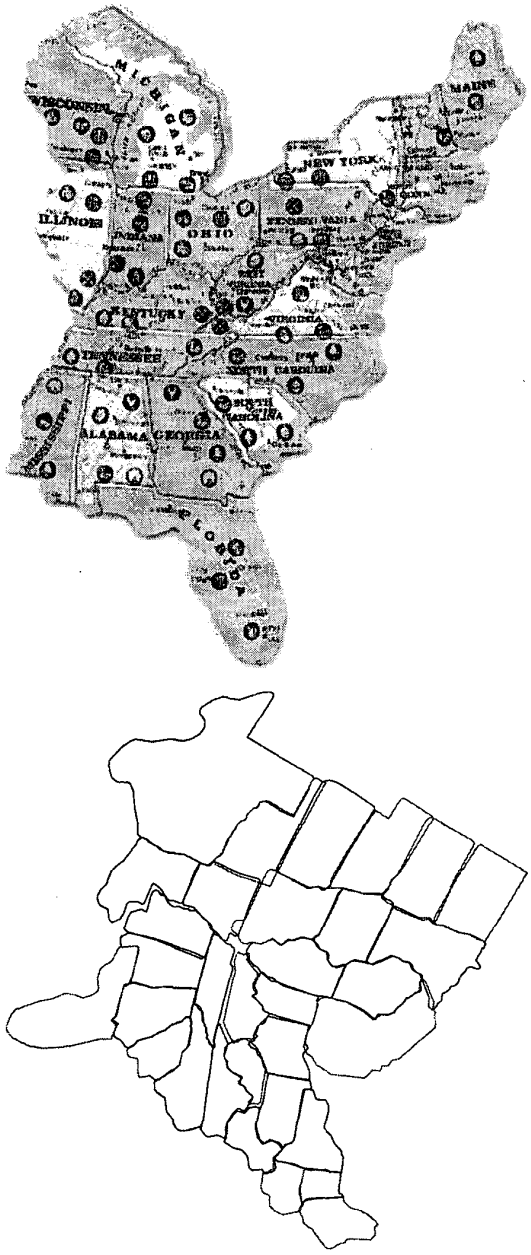

Figure 10. (Top) A portion of the U.S. map is scanned. (Bottom) The results of re-assembling the map using our approach.

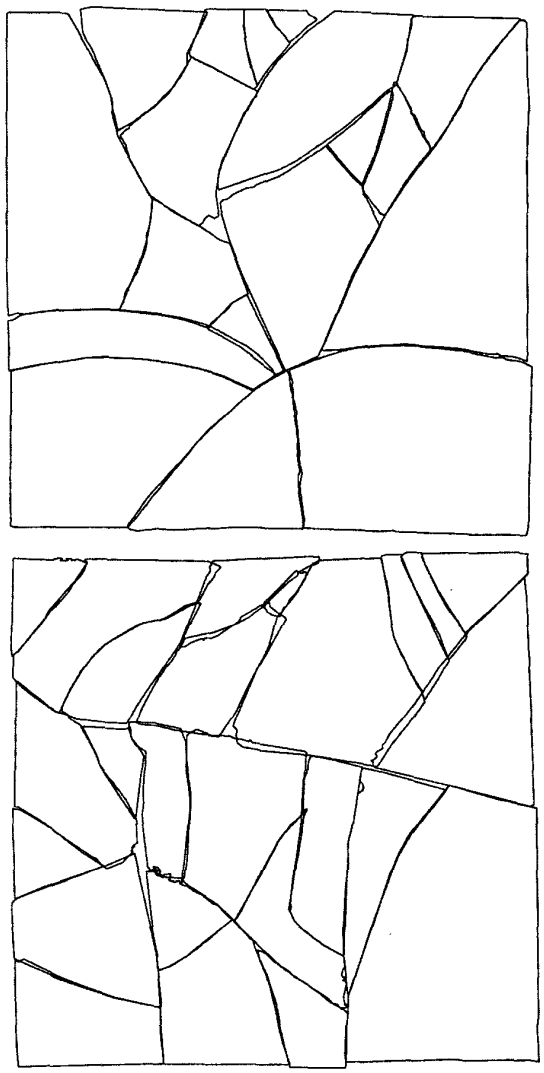

Figure 11. The reconstruction of a ceramic tile with 18 pieces (top) and another with 25 pieces (bottom).

[13] T. B. Sebastian, J. J. Crisco, P. N. Klein, and B. B. Kimia. Constructing 2D curve atlases. *MMMBIA*, pages 70–77, 2000.

[14] C. W. Shu and S. Osher. Efficient implementation of essentially non-oscillatory shock-capturing schemes. *J. Comp. Phy.*, 77:439–471, 1988.

[15] K. Siddiqi, B. B. Kimia, and C. Shu. Geometric shock-capturing ENO schemes for subpixel interpolation, computation and curve evolution. *GMIP*, 59(5):278–301, September 1997.

[16] G. Ucoluk and I. H. Toroslu. Automatic reconstruction of broken 3-d surface objects. *Computer Graphics*, 23:573–582, 1999.

[17] H. Wolfson. On curve matching. *PAMI*, 12:483–489, 1990.

[18] L. Younes. Computable elastic distance between shapes. *SIAM J. Appl. Math.*, 1996.