A Global Approach to Automatic Solution of Jigsaw Puzzles

David Goldberg
Palo Alto Research Center
3333 Coyote Hill Rd.
Palo Alto, CA 94304
goldberg@parc.com

Christopher Malon
MIT Department of Mathematics
Cambridge, MA 02139
malon@math.mit.edu

Marshall Bern
Palo Alto Research Center
3333 Coyote Hill Rd.
Palo Alto, CA 94304
bern@parc.com

ABSTRACT

We present a new algorithm for automatically solving jigsaw puzzles by shape alone. The algorithm can solve more difficult puzzles than could be solved before, without the use of backtracking or branch-and-bound. The algorithm can handle puzzles in which pieces border more than four neighbors, and puzzles with as many as 200 pieces. Our overall strategy follows that of previous algorithms but applies a number of new ideas, such as robust fiducial points, "highest-confidence-first" search, and frequent global reoptimization of partial solutions.

Categories and Subject Descriptors

I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling; I.4.7 [**Image Processing and Computer Vision**]: Feature Measurement

General Terms

Algorithms

Keywords

Shape matching, fiducial points, mesh smoothing

1. INTRODUCTION

Automatic solution of jigsaw puzzles by shape alone goes back at least to 1964 [7]. Although numerous papers have been written on this subject since then, there are still no published algorithms that can solve large puzzles reliably and efficiently. In this paper we introduce a few new ideas that extend the reach of what can be done to a wider class of puzzles and to puzzles with more pieces.

Papers on this subject traditionally justify the work by citing related problems. Related problems include reconstructing archeological artifacts [8, 10, 11, 12, 13], mating surface patches of scanned objects [14], and even fitting a protein with known amino acid sequence to a 3D electron density map [19]. The real interest in jigsaw puzzle solving, however, is simply that it is a natural and challenging problem that catches people's imaginations.

The apictorial jigsaw problem has two main difficulties. One is combinatorial: there are a very large number of ways that pieces can be assembled. The other is geometric: it is difficult to detect if a pair of complementary pieces really match. When solved by hand, a person can usually feel a snap when making a true match, but scanned piece shapes (both ours and those of previous researchers) are not precise enough for such a determination. Evidently a robot can feel a snap too: Burdea and Wolfson [4] used this sort of force feedback "matching oracle" in robotic solution of jigsaw puzzles.

Standard toy-store jigsaw puzzles obey certain rules that make the problem more tractable that it would otherwise be. Standard rules include: (1) the puzzle has a rectangular outside border; (2) pieces form an overall rectangular grid so that each interior piece has four *primary neighbors* (left, right, above, and below); and (3) pieces interlock with their primary neighbors by tabs, consisting of an "indent" on one piece mating with an "outdent" on its neighbor. Another rule is optional: (4) each piece has no neighbors except its primary neighbors, that is, the cutting lines between pieces meet only at $+$-junctions rather than a mix of $+$-, T-, and Y-junctions. Our algorithm can solve reasonably big apictorial jigsaw puzzles (100 or more pieces), even if they do not obey rule (4).

We know of only one other automatic jigsaw puzzle solver that can handle large puzzles: the algorithm given by Wolfson et al. [21]. Our algorithm follows the same overall approach as that of Wolfson et al., that is, first solving the border and then filling in interior "pockets", but our algorithm differs in many substeps. We make more use of global geometry, for example, at all times maintaining a geometric embedding of the best partial solution; whereas Wolfson et al. [21] use only local geometry, the pairwise matching of sides of pieces.

Our algorithm appears to be more capable, solving a 204-piece puzzle, the largest puzzle solved automatically to date. Our algorithm also solves a 100-piece puzzle that grossly disobeys rule (4). Because Wolfson et al. rely on pieces having four well-defined sides, their algorithm cannot solve puzzles that significantly disobey rule (4). (Although our 204-piece puzzle obeys rule (4), our algorithm does not take advantage of this property. We did not realize the importance of rule (4) at the time we bought the puzzles!) We have not yet tried our program on an independent "test puzzle", not used in the development of the algorithm. This experiment would be a more rigorous criterion for success.

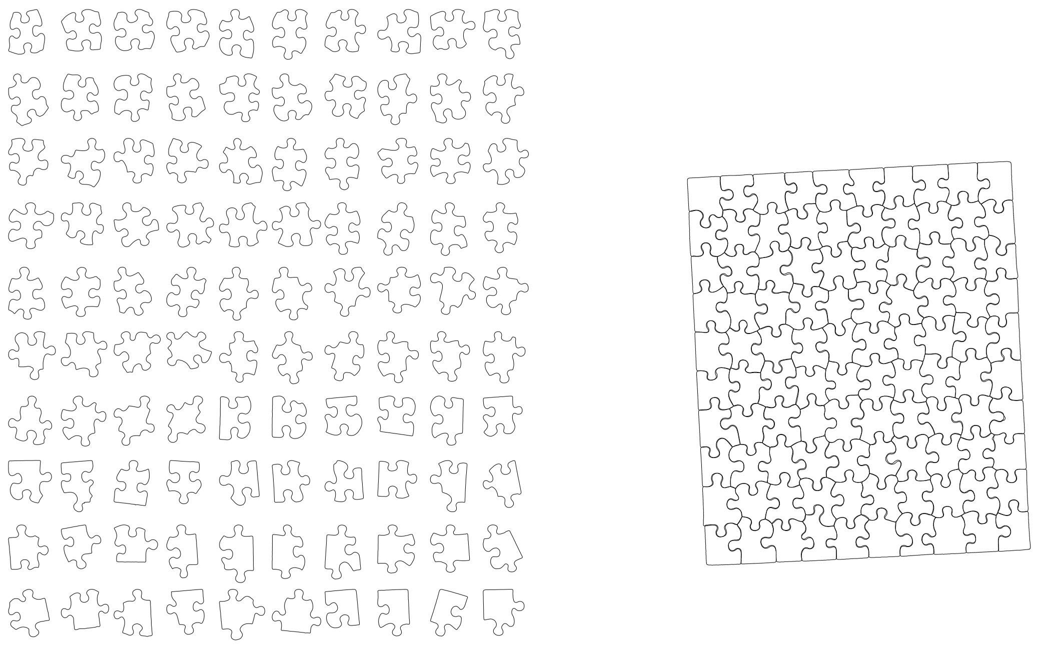

Figure 1: This 100-piece puzzle presents a difficulty for previous algorithms: pieces do not have four well-defined sides.

2. OVERVIEW

As in the work of Wolfson et al. [21], our algorithm first assembles the border pieces using a heuristic for the Traveling Salesman Problem. We depart from previous work in how we place the interior pieces. Because we do not assume that pieces have well-defined sides, we require a more global matching technique. At all times, we maintain an optimized planar embedding of the current partial solution. We fit a piece into a pocket not by independent pairwise fitting with top and side neighbors as in [21], but by fitting it into the embedded partial solution, thus allowing for any number of neighbors around the pocket. Wolfson et al. rejected global embedding because of the possibility of accumulated errors, but we found to the contrary that global embedding gave more accurate results than pairwise matching, enabling a greedy placement algorithm—without any backtracking or branch-and-bound—to solve the jigsaw puzzles. (For more complicated puzzles, we could easily add backtracking or branch-and-bound.)

We used *fiducial points* (specifically the centers of ellipses fit to the indents and outdents) to find the best translation and rotation of a piece to match a pocket. An alternative would be to use the Schwartz-Sharir curve-to-subcurve matching algorithm [16, 21], or Wolfson's subcurve-to-subcurve matching algorithm [22]. Yet more possibilities include a string matching approach [3] or a dynamic programming energy minimization approach [8, 17]. The fiducial points approach, however, worked quite well and is significantly faster, because it does not need to test all subcurve or substring starting points. Another advantage of fiducial points is that they are more robust to scanning noise than some of the other techniques. For example, the Schwartz-Sharir algorithm picks points p_i along the boundary of one piece and q_i along the boundary of another piece. Then it finds a rigid motion that carries the p_i's to p_i''s and minimizes $\sum(p_i' - q_i)^2$. This only works well if the p_i and q_i points can be brought into alignment. If the points are equally spaced by arc length, then scanner noise that introduces a bump into one of the pieces will throw the two sequences out of synchronization.

We filled pockets in *highest-confidence-first* order. Call an empty position an *eligible pocket* if it has at least two primary neighbors that have already been placed. Initially, when only the border pieces have been placed, there are four eligible pockets; later there may be quite a few eligible pockets as shown in Figure 2. At each step we fill the eligible pocket that has the highest ratio of the score of best fitting piece to the second best fitting piece. This order turned out to be more reliable than best-first order, which has been used before [3].

After fitting a piece, we reoptimize the global embedding of all pieces. We do this by minimizing the squares of the distances between corresponding points on neighboring pieces, for all neighboring pieces at once. Global optimization distributes the matching inconsistencies throughout the partial solution, and in our experiments outperformed a smoothing procedure that moved one piece at a time.

3. DETAILED DESCRIPTION

We now describe the algorithm in more detail, starting from data acquisition. We used two jigsaw puzzles purchased at the local toy store: a 100-piece puzzle made by Milton-Bradley (Figure 1) and a 204-piece puzzle made by Ravensburger (Figure 5). The puzzle pieces are about a millimeter thick and cast shadows when scanned, making it difficult to accurately extract the boundaries of the pieces. We found that a color copier produced less shadow than a flatbed scanner, so we first copied the pieces—copying the blank, back sides of well-separated pieces against a red background—and then scanned the copy at 300 dpi.

To extract the pieces from the scans, we used a color histogram to determine the color range of the back sides of pieces, and then defined the background to be the pixels not falling within this color range, in order that shadows be considered background. The pieces are then the largest connected components of the foreground (complement of the background). We smoothed piece boundaries using morphological operations: we switched any foreground pixel to back-

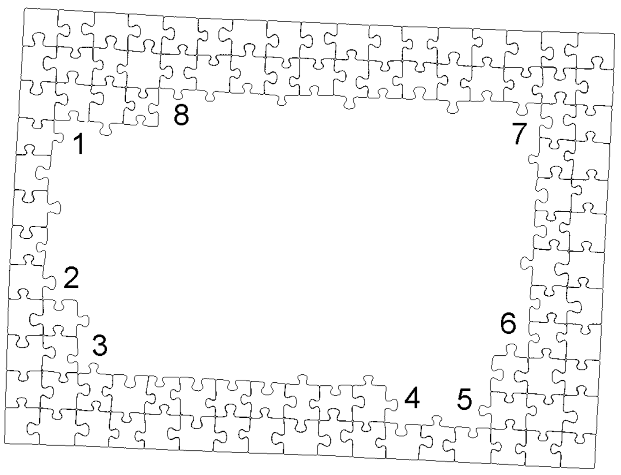

Figure 2: There are 8 eligible pockets at this step.

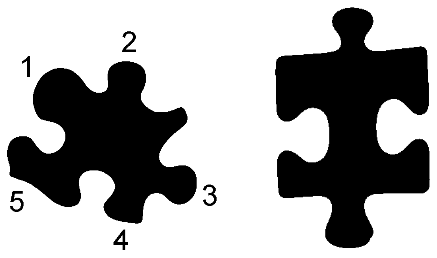

Figure 3: (a) The piece on the left illustrates the importance of identifying indents before outdents. Of the five possible outdents, only 2 and 3 are genuine. (b) The piece on the right has some long straight stretches that are not straight sides.

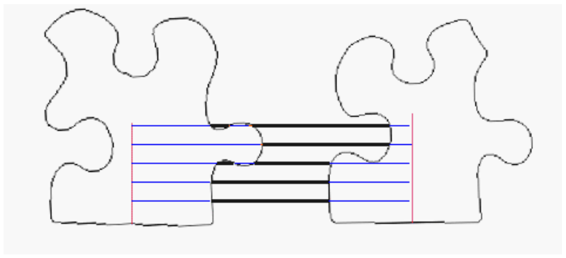

Figure 4: For pieces that fit perfectly, the lengths of the bold parts of horizontal lines should be equal.

ground if it had three or more background neighbors, or if it was in a row or column with two or fewer foreground pixels. We then took every other pixel around the boundary as a vertex, and performed some Gaussian smoothing on these points, in order to obtain a polygonal representation of the boundary. The polygons, each with about 600 vertices with floating-point coordinates, seemed to be accurate to about .5 mm tolerance. Sources of error included piece shadows, scanner noise, specks of color on the back sides of pieces, and most ominously "hanging chad". We manually numbered the polygons according to their positions in the solved puzzle, in order that we could check the computer's solution.

3.1 Finding Indents, Outdents, and Flat Sides

Many previous algorithms, including that of Wolfson et al. [21], simply divided piece boundaries into four sides by finding sharp corners. In order to handle puzzles such as Figure 1, however, we need a somewhat more sophisticated classification of boundary "parts".

Our algorithm searches for *tabs*, which are either *indents* or *outdents*. Finding the tabs requires some care. For example, if we define an outdent as a region that touches the convex hull of the piece and has a "neck" that will interlock with a neighboring piece, then the piece in Figure 3(a) would be incorrectly classified as having five outdents. It turns out to be easier to find the indents first.

In order to find the indents, we first find *inflection points*, edges at which the turning of the polygon changes direction. Because straight parts of the boundary may have many spurious inflection points, we reject inflection points unless the turning after each end totals at least 10° before turning back. Figure 6(a) shows inflection points along an indent and an outdent. The precise locations of inflection points are not very robust, because piece boundaries are often quite straight near their inflection points; however, the mere existence of inflection points on either side of an indent is quite reliable. Indents are identified by finding points p along the boundary that locally maximize the distance to the convex hull of the piece, finding the inflection points on either side of p, and finally testing whether the tangent lines through the inflection points cross outside the piece. This last test embodies the assumption that tabs interlock.

Before finding outdents, our algorithm finds straight sides. Even this step is nontrivial. For example, the piece in Figure 3(b) has long straight stretches of the boundary that are not straight sides. To handle the situations shown in this figure, the algorithm finds straight sides by first finding straight stretches of boundary, but rejecting a stretch unless it turns "inward" at each end, and neither inward turn is part of a previously identified indent.

The algorithm finds outdents along stretches of the boundary that have not yet been assigned to indents or straight sides. More precisely, an indent claims the boundary on either side up until the boundary comes sufficiently close ($\leq$ 10 pixels) to the convex hull and then falls off the hull again. Symmetric to the procedure for identifying indents, outdents are identified by finding points p along the boundary that locally minimize the distance to the convex hull of the piece, finding the inflection points on either side of p, and finally testing whether the tangent lines through the inflection points cross inside the piece.

3.2 Ordering the Border

As in previous algorithms [3, 21], we begin by placing the border pieces. First we find the order of the border pieces, then we actually embed the border pieces in the plane. Each border piece has a right and left side, unambiguously defined by orienting the piece with its straight side down. In the case of a corner piece, we orient the two straight sides to be down and to the right. We define a score $s(A, B)$ measuring how well the right side of piece A fits the left side of piece B. Finding the best ordering is now an asymmetric traveling salesman problem with $s(A, B)$ serving as the distance from "city" A to city B. It is asymmetric in that $s(A, B) \neq s(B, A)$. We solve this NP-hard problem using the assignment problem heuristic.

If the number of border pieces is n, imagine n workers and n machines where the cost of assigning worker A to machine B is $s(A, B)$. In polynomial time we can find the best assignment of workers to machines. To convert this to a path, pick any piece P to start the path. If worker P is assigned to machine Q, we set the second element of the path to be Q. If worker Q is assigned to R, then set R to be the third element of the path, and so on. This may not give a traveling salesman tour because the path may return to P

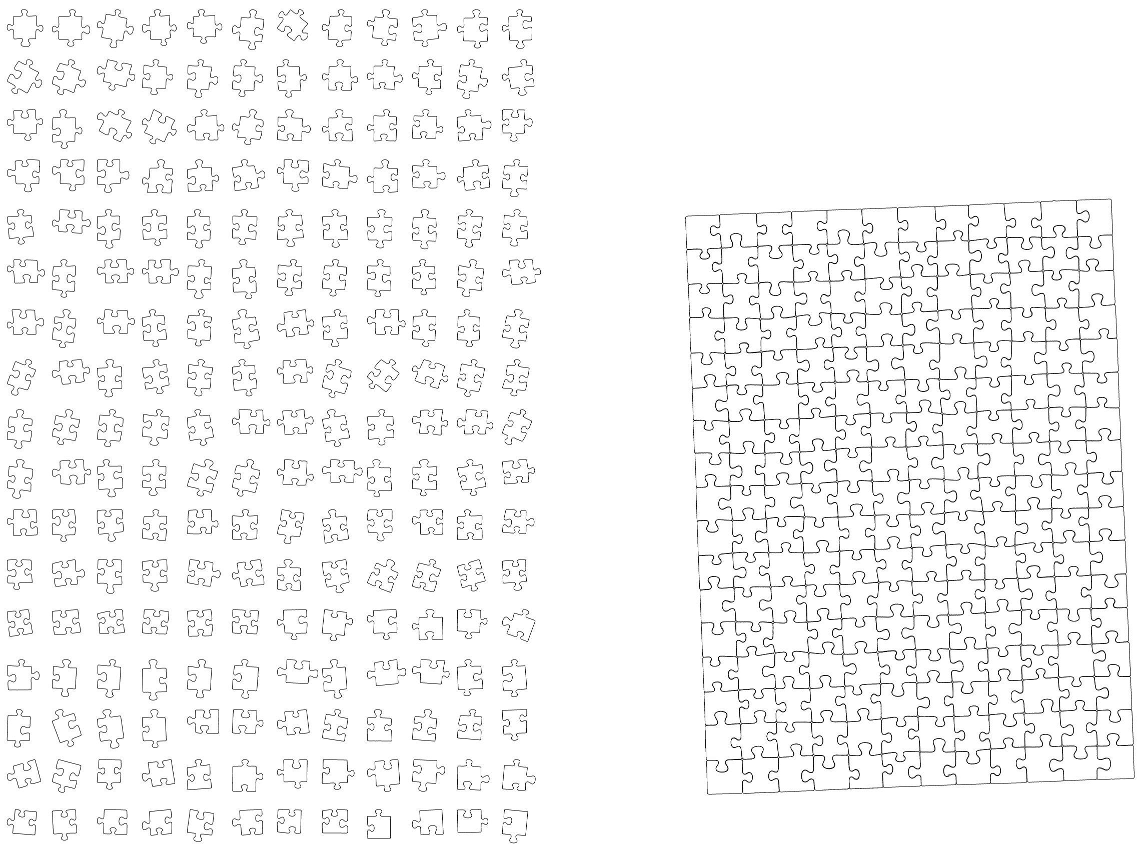

Figure 5: This 204-piece puzzle is the largest one solved automatically to date. Wolfson et al. solved two intermixed 104-piece puzzles, but the two-puzzle problem is somewhat easier because there is more border and near-border.

in less than n steps. For our test puzzles we obtained either a single n-long cycle or two cycles. In the latter case it requires only $O(n^2)$ steps to test all possible ways to stitch the two cycles together. Taking the optimal stitched cycle will find the correct border ordering provided that there were no more than two errors in the initial ordering. One of the advantages of starting with border pieces is that the four corner pieces provide a natural check on the solution, since they must be symmetrically placed. Thus if the assignment problem heuristic made more than two errors, we would detect this failure at this early stage, and go on to consider more complicated ways of stitching together cycles.

It is relatively easy to come up with a scoring function $s(A, B)$ for border pieces, since such pieces must be aligned at their straight sides. The score we use is computed by positioning the pieces with their straight sides along a common supporting line, and then examining lines parallel to the supporting line, as shown in Figure 4. For each such line, we compute the length “between” the two pieces (total length outside both pieces). If the pieces fit perfectly, the distribution of lengths should cluster tightly about the median, and hence we compute the score as the average difference to the median of the between lengths.

3.3 Planar Embedding

The next step is to embed the border pieces in the plane, that is, to give them x- and y-coordinates. First, we place one piece arbitrarily. From the solution to the traveling salesman problem, we can find two neighboring pieces. Although we could place the second piece by aligning its straight side to the straight side of the first piece, instead we use a more general method that we shall reuse for placing pieces in the interior of the puzzle.

We use *fiducial points*—canonical locations—to align adjacent pieces. We define one fiducial point per tab, both indents and outdent. One possible choice for a fiducial point is an inflection point near the neck of a tab, but this choice is rather poor, because the precise location of inflection points is quite sensitive to scanner noise.

We instead compute fiducial points by fitting an ellipse to each tab (using least squares) and defining the center of the ellipse to be a fiducial point, as seen in Figure 6(a). We fit the ellipse to polygon points from one inflection point to another. Although the positions of the inflection points are not robust, small variations in their positions have little effect on the fitted ellipse.

We use the fiducial points in turn to place a sequence of points along the tab that should match well with corresponding points on the tab of the neighboring piece. Rather than taking points equally spaced by some measure of arc length as in previous work [3, 21], we found that synchronization was more robust with points that are equiangular as viewed from the ellipse center. In order to align two neighboring pieces, we then use a least-squares fit to find the rigid motion (rotation and translation) that minimizes the sum of squares of distances between corresponding points.

The method just described performs quite well locally, but is insufficient to compute a good global embedding. The last boundary piece will fit well with the next-to-last piece, but typically will be quite far from the first piece and hence will not form a closed frame.

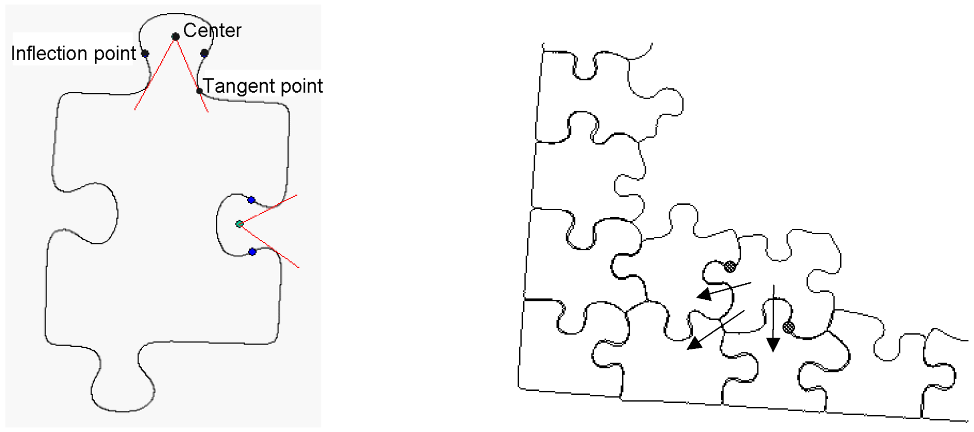

Figure 6: (a) The algorithm uses three feature points on a tab: inflection points, ellipse centers, and tangent points. (b) The matching score of a piece to a pocket is computed using the path between the two tangent points marked by dots.

We perform a *global relaxation* that distributes the total error evenly among all the border pieces. This relaxation will later be used for the interior pieces as well.

For the global relaxation, we pick k pairs ($k = 30$ worked well) of corresponding points along each common border between neighboring pieces. We then minimize the sum of the squares of all intra-pair distances in order to find new positions for all n pieces at once. The adjustment of piece P is a rigid motion given by

$$\begin{aligned} x' &= x\cos\theta_P - y\sin\theta_P + c_P \\ y' &= x\sin\theta_P + y\cos\theta_P + d_P. \end{aligned}$$

We want to find the values of the $3n$ variables $\{\theta_P, c_P, d_P\}$ that minimize the distance between pairs of corresponding points

$$\sum_{P,Q}\sum_{j=1}^{k} d(z'^{P}_{j}, z'^{Q}_{j})$$

where z'^{P}_{j}, z'^{Q}_{j} are the pairs of points along the adjusted common border of neighboring pieces P and Q. This minimization problem is not linear. But since the relaxation only moves pieces a small amount, θ_P is small and we can linearize using

$$\begin{aligned} x' &\approx x - y\theta_P + c_P \\ y' &\approx x\theta_P + y + d_P. \end{aligned}$$

A similar global relaxation has been proposed as a method for repositioning vertices in finite-element meshes in order to improve the shapes of elements [5], but the idea has not gained much popularity because one-at-a-time repositioning seems to be just as effective and much more efficient (see for example [2]). The superiority of global relaxation for jigsaw puzzles thus came as a bit of a surprise to us. We now expect that global relaxation would be better than one-at-a-time repositioning for the problem of distributing errors smoothly over a surface scanned in nearly-rigid patches, a problem that arises in the Digital Michelangelo project [14].

3.4 Placing Interior Pieces

As we mentioned in Section 2, we use a simple greedy algorithm, without backtracking or branch-and-bound, to place interior pieces. There are three ingredients to the algorithm: a score measuring how well a piece fits into an eligible pocket, a strategy for the order in which we place pieces, and an optimization step to readjust the embedding after each new piece is placed.

Recall that an unfilled piece location is called an eligible pocket if it is adjacent to at least two existing pieces (Figure 2). We score how well a piece P fits into an eligible pocket in two steps. First we calculate the position P would have if it really belongs in that pocket, and then we compute a score using this position. The position for P is the one that minimizes the sum of squares of distances between corresponding fiducial points, those on P's tabs and those on the pocket's tabs. The score for P is computed by walking along the boundary of P, and for each vertex of P, finding the closest boundary point on any of the neighboring pieces defining the pocket. (Here we measure vertex-to-edge distances rather than the simpler, but less robust, vertex-to-vertex distances.) As can be seen in Figure 6(b), a piece bordered on two sides with already placed pieces can share a third border with a diagonally adjacent piece. The score is the average of the cubed distances between vertices of P and their closest pocket points. We do not really know why the cubes of distances worked better than the more usual squares of distances (for the 100-piece puzzle either one worked); evidently true matches are distinguished by close distances everywhere.

Because we do not assume that pieces have well-defined sides, we need to know how far to extend the walk along the boundary of P. We accomplish this using *tangent* points, points on the neck of the tab that are tangent to a radius emanating from the ellipse center, as shown in Figure 6(a). As shown in Figure 6(b), the operative part of the boundary of P lies between two tangent points, one on each of the tabs of the eligible pocket.

The part of the boundary not included between the two tangent points also contains valuable information on how well pieces fit together. This information was not needed for the 100-piece puzzle, but became important for the larger 204-piece puzzle. To incorporate this information into the scoring for a piece P, we use one step of lookahead. Placing P creates some new eligible pockets adjacent to P. For each of these pockets we find the best fitting piece. Then we recompute the score for P using the (temporarily) newly fitted neighbors. In other words, the score is recomputed using a path that includes much more of the boundary of P.

As mentioned in Section 2, we place pieces with highest confidence first, whether or not we are using one-step lookahead. The highest-confidence placement fills the eligible pocket with the largest ratio between the first and second-best scores. After placing each piece, we reoptimize the embedding using the global least-squares relaxation described in Section 3.3.

4. DISCUSSION

Our implementation includes a few simple speed-up tricks. For example, we cache the computation of scores, and we only perform the one-step lookahead mentioned above for pieces whose score is within a factor of two of the best scoring piece. Overall it took about 3 minutes to solve the 100-piece puzzle and 20 minutes to solve the 204-piece puzzle on a Sun Ultra-60 workstation. This includes the time needed to process each scan, as well as the actual placement of each piece.

Many parts of our algorithm do not use any hand-tuned parameters. Examples are the computation of fiducial points and the relaxation calculation. Other steps will probably have to be modified slightly to work properly on a wide range of test puzzles. For example the identification of straight sides involves some parameters that are hand-tuned. The most serious example of ad hoc tuning occurs in the scoring function used to place interior pieces. As mentioned above, we computed scores as a sum of cubes of distances (rather than the usual squares) in order to solve our 204-piece test case.

In summary, we were attracted to the jigsaw puzzle problem because it presents an interesting combination of combinatorial and geometric challenges, and because no completely satisfactory algorithms have yet been published. We believe our algorithm performs better than previous algorithms for this "toy problem". More speculatively, we think that some of our techniques could be applicable to other problems. First, we use a feature-based method, focusing on tabs, to match sides of pieces. We expect that analogous problem-specific features (color, texture, thickness, etc.) would be more powerful than generic curve-subcurve matching for many realistic assembly problems, such as those arising in archeology. Second, we use a global optimization step to readjust the positions of all the pieces after fitting each new piece. This step seems especially important when the boundary curves themselves hold little information, for example, in the case of broken glass with relatively straight fracture lines. Finally, we fit pieces in order of confidence. For assembling broken objects, the power of this idea could perhaps be magnified by a probabilistic model of the breaking process and use of log-likelihood for scoring matches.

5. REFERENCES

[1] T. Altman. Solving the jigsaw puzzle problem in linear time. *Applied Artificial Intelligence*, **3** (1989) 453–462.

[2] M. Bern and P. Plassmann. Mesh generation. In *Handbook of Computational Geometry*, J.-R. Sack and J. Urrutia, eds., North-Holland, 2000, 291–332.

[3] H. Bunke and G. Kaufmann. Jigaw puzzle solving using approximate string matching and best-first search. in *Computer Analysis of Images and Patterns,* Chetverikov and Kropatchs, eds., LNCS, Springer, (1993) 299–308.

[4] G.C. Burdea and H.J. Wolfson. Solving Jigsaw Puzzles by a Robot. *IEEE Trans. on Robotics and Automation* **5** (1989), 752–764.

[5] S. Canann, M. Stephenson and T. Blacker. Optismoothing: an optimization-driven approach to mesh smoothing. *Finite Elements in Analysis and Design* 13 (1993) 185–190.

[6] M.G. Chung, M. Fleck and D.A. Forsyth: Jigsaw Puzzle Solver Using Shape and Color. *Proc. ICSP '98*, 1998, 877–880.

[7] H. Freeman and L. Gardner. Apictorial jigsaw puzzles: The computer solution of a problem in pattern recognition. *IEEE Trans. on Electronic Computers* **13** (1964) 118–127.

[8] W. Kong and B.B. Kimia. On solving 2D and 3D puzzles using curve matching. *Proc. IEEE Computer Vision and Pattern Recognition*, 2001.

[9] D.A. Kosiba, P.M. Devaux, S. Balasubramanian, T. Gandhi, R. Kasturi. An Automatic Jigsaw Puzzle Solver. *Proc. Int. Conf. Pattern Recognition*, Jerusalem, 1994, 616–618.

[10] H. C. da Gama Leitão and J. Stolfi. Automatic reassembly of irregular fragments. Tech. Report IC-98-06, Univ. of Campinas, 1998.

[11] H. C. da Gama Leitão and J. Stolfi. Information Contents of Fracture Lines. Tech. Report IC-99-24, Univ. of Campinas, 1999.

[12] K. Leutwyle. Solving a digital jigsaw puzzle. http://www.sciam.com/explorations/2001/062501fresco/

[13] M. Levoy. The digital *Forma Urbis Romae* project. http://www.graphics.stanford.edu/projects/forma-urbis/

[14] M. Levoy. The digital Michelangelo project. http://www.graphics.stanford.edu/projects/mich/

[15] G.M. Radack and N.I. Badler. Jigsaw puzzle matching using a boundary-centered polar encoding. *Computer Graphics and Image Processing* **19** (1982), 1–17

[16] J.T. Schwartz and M. Sharir. Identification of partially obscured objects in two and three dimensions by matching noisy characteristic curves. *Int. J. Robotics Research* **6** (1987), 29–44.

[17] T. Sebastian, J.J. Crisco, P. Klein, and B. Kimia. Constructing 2D curve atlases. *Proc. Mathematical Methods in Biomedical Image Analysis*, 2000, 70–77.

[18] G. Ücoluk and I.H. Toroslu. Automatic reconstruction of broken 3-D surface objects. *Computers and Graphics* **23** (1999), 573–582.

[19] C. Wang. Determining the Molecular Conformation from Distance or Density Data. PhD thesis, Department of Electrical Engineering and Computer Science, MIT, 2000.

[20] R.W. Webster, P.S. LaFollette and R.L. Stafford. Isthmus critical points for solving jigsaw puzzles in computer vision. *IEEE Trans. on Systems Man and Cybernetics* **21** (1991), 1271–1278.

[21] H. Wolfson, E. Schonberg, A. Kalvin and Y. Lamdan. Solving jigsaw puzzles by computer. *Annals of Operations Research* **12** (1988), 51–64.

[22] H. Wolfson. On curve matching. *IEEE Trans. on Pattern Analysis and Machine Intelligence* **12** (1990), 483–489.