Assembling Jigsaw Puzzles

Aric Bartle Tianye Lu

Abstract

We propose an improved method for assembling real world jigsaw puzzles. Our approach uses both shape and color measures to determine piece compatibility. We give a novel approach based upon the z-buffer algorithm to give a shape measure. The shape measure is then used to prune away highly incompatible pieces. A color measure based upon Mahalanobis Gradient Compatibility is then used for any further comparisons. We assemble the border of a puzzle by formulating it as TSP and using a greedy approximation. We then consider greedy assembly of the interior of the puzzle. Our approach gives improved performance over other attempts at assembling real world jigsaw puzzles.

1 Introduction

Jigsaw puzzles have long been an entertaining and often challenging recreation. Starting from the mid 1700s when John Spilsbury, a London mapmaker and engraver, commercialized jigsaw puzzles, they have continued to grow in appeal and widespread use. Beyond just being a fun activity, the problem of solving puzzles is interesting from a computational standpoint as the problem is NP-hard and devising solutions that work well in polynomial time is a challenge. The problem is also of great importance in the closely related problems of reconstructing shredded documents and recovery of archaeological artifacts.

In creating an automated puzzle solver, there are many challenges obvious and not that need to be overcome. The first issue is simply that of creating a digital representation of the pieces, which in itself is not a trivial process, but the real challenge comes in actually fitting these "pieces" together.

While it may seem that this latter problem could be solved simply by finding the best match for two pieces and successively adding on more, this is not possible. Inherently, the metrics used for measurement do not give exact answers that absolutely determine what pieces fit together. There are numerous reasons for this. For one, the various measures of how closely to pieces fit normally just consider the one or two pixel borders of pieces which may not be enough in some cases. For instance, in the particularly problematic case of pieces of uniformly the same color (such as blue sky pieces), any measure that compares colors only will fail.

Regardless of the choice of measure, there needs to be a digital representation of the pieces to actually operate upon. Simply scanning in puzzle pieces leads to noisy borders and imprecise colors along them. Measures, then, will need to take into account these artifacts. However, for the most part, works do not consider actual scanned in puzzle pieces and instead side step the problem by using computer generated puzzles.

The final aspect in assembling jigsaw puzzles is that of using the measure in an algorithm to actually assemble the puzzle. Because of the inexactness of measures the overall problem of jigsaw assembly is NP-hard [1]. A variety of different strategies have been proposed ranging from greedy selection of puzzle pieces, edge identification and assembly, and Markov Random Fields [3].

For this work, we consider the lesser studied problem of jigsaw assembly from real world jigsaw pieces. Our approach follows most closely the work of [3, 8]. We first extract a set of edges for each puzzle piece (4 in this case since we only consider standard jigsaw puzzle pieces). These edges contain both position and color information of the pixels that make them. When considering sets of possibly matching edges, we first use the shape measure to eliminate all but a few that fall within a threshold of allowable error. After which,

the color measure is used to compare the pieces. We assemble the puzzle with a greedy approach. The border and interior are dealt with separately. We take advantage of the limited orientations of the border pieces to formulate it as a traveling salesman problem. Once we have got the border, we find the unoccupied positions with at least two edges fixed to boost our confidence in committing to fill in a piece. Our main contributions are using shape information as a filter, and a new method to do curve matching.

2 Related Work

The first work ever related to puzzle assembly, [5], considered a measure based upon shape matching alone. Their method tried to identify critical points along the edge that can be used for segmentation; a measure may then be calculated from this. The best result using shape matching alone was obtained in [4] in which they correctly assembled a 204 piece puzzle. Their approach uses a measure based upon shape along with a greedy assembly algorithm. The algorithm first assembles the border of the puzzle using an approximation of the TSP. It then uses a greedy approach to assemble the interior of the puzzle in which it fills in the locations with the best match first.

The first work to use both jigsaw piece shape and image information, [6], required that adjacent jigsaw pieces to have similar colors by computing color compatibility along the matching contour. Others followed suit [7, 8] pushing the state-of-the-art to 320 pieces in [8] for computer generated puzzles. The work also assembles a 54 piece puzzle based upon real world puzzle pieces.

More recent works have made various simplifications about the puzzle pieces. They for the most part assume that the pieces are square and computer generated [3, 9]. This strips away any uncertainty in the digital representation of the pieces and focuses the problem purely on a color measure and an assembly approach. At the present time the state of the art approach is by [3] in which they assemble a 9600 piece square jigsaw piece puzzle. In the work they propose the so called Mahalanobis Gradient Compatibility measure that measures how closely the color gradients of two edges match. For assembly they consider puzzle assembly as a graph problem and compute the minimal spanning tree. While this does not guarantee a correct assembly, it at least acts as a fairly good heuristic.

3 Compatibility Measures

3.1 Edge extraction

The first part of our approach is to extract the border from a piece represented by an image (Figure 1). This gives us a single curve which can be segmented into 4 individual ones corresponding to the 4 different edges. We find the start and end points of each edge by finding the points on the curve closest to the top left, bottom left, bottom right, and top right corners of the image. With the top level edges extracted, we extract more edges at additional insets from the border. The reason for doing this is that color information is quite poor particularly around the outside border of the piece. By insetting we get more representative samples of the "true" borders.

3.2 Shape Measure

Our approach to compute a shape measure is inspired by the z-buffer algorithm. We first consider an edge of a piece represented by a set of points $X = \{(x_1, y_1), (x_2, y_2), ...\}$(figure 1). We then define a line segment with end points at the first and last points. We want to find a distance function $f(t), 0 < t < L$ where L is the length of the line segment such that $f(t)$ gives the perpendicular distance to the farthest away point from the line segment at position t on it. To compute $f(t)$ we use the z-buffer algorithm. That is, we discretize the line segment into L/dt bins (we selected $dt = 1$ to give 1 pixel resolution). Then for each line segment X_i, X_{i+1} we compute which bins it belongs to and the distances to the line corresponding to those bins. If a distance is greater than a value in the bin, then the bin value is updated to the new greatest distance. We

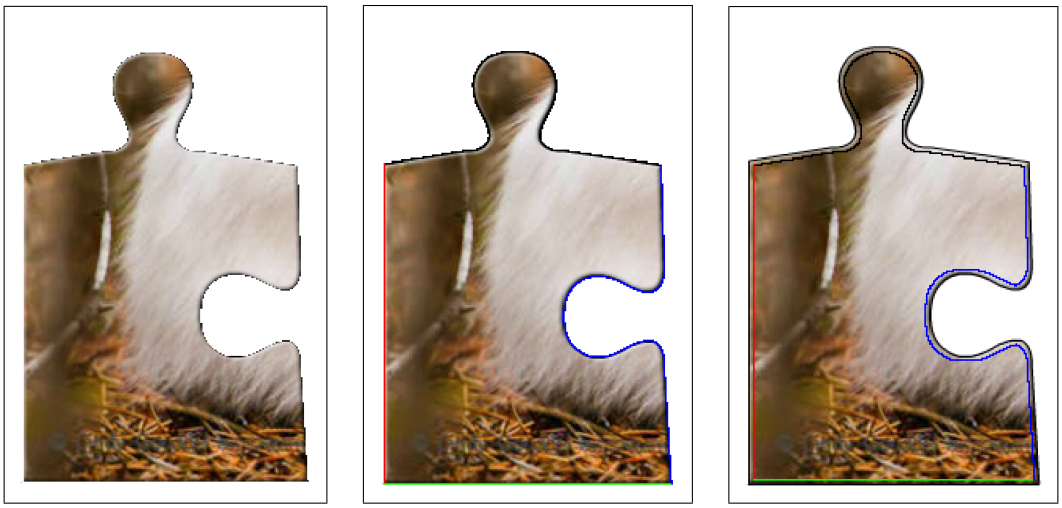


Figure 1: Left: the original image. Middle: the image with the extracted edges highlighted by red, blue, green, black. This is termed inset level 0. Right: the extracted edges at inset level 3.

note that we cannot operate just on the points because that could lead to bins that were not updated with some distance. We can view $f(t)$ as giving the upper envelope (Figure 2).

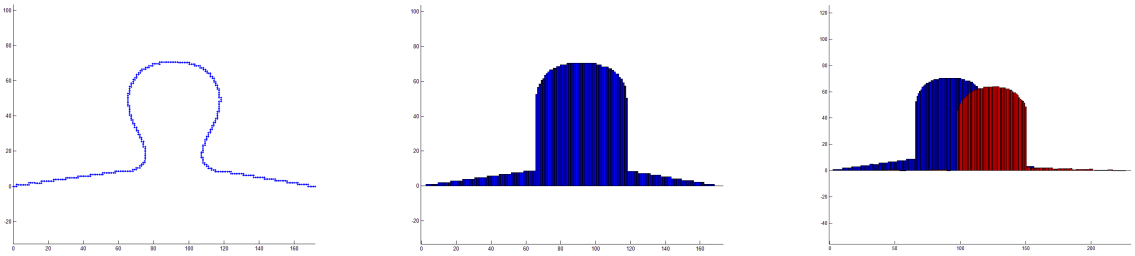


Figure 2: Left: Puzzle edge. Middle: $f(t)$, the upper envelope. Right: comparison of two edges represented by their upper envelopes.

To compare two edges, we first align the edges so that they are centered and then compare the distance functions. We define the measure between them as

$$S = \frac{1}{T} \sum_{t=1}^{K} |f(t) - f'(t)|$$

The right image of Figure 2 shows how this measure penalizes two edges that misalign.

A possible criticism of this approach is that we compute an upper envelope, so some information is lost. However, for the most part, if pieces are different, then they will differ in the upper envelope, not just in the regions that were lost. We found that this simplification was reasonable. We were able to correctly assemble the rabbit puzzle (Figure 6) with just the shape measure. For more complex puzzles, we found that shape even if we had considered a more complex measure was inadequete because at times there could be multiple pieces that visually fit perfectly despite being wrong because their colors were inconsistent.

Figure 3: Pieces that don't match would have an edge detected on their border

3.3 Measurement based on image content

In this section, we discuss the pairwise compatibility measurement based on image content. For the convenience of discussion, we define the measurement on square pieces. They can be easily extended to other shapes. Suppose we are trying to compare two square pieces x_i and x_j, with edges of size P. x_i is placed to the left of x_j. We use $c \in \{1, 2, 3\}$ to denote RGB channels of a pixel.

3.3.1 Standard Color Compatibility (SCC)

As a baseline, We first tried a simple measurement based on the difference of colors on the boundary of two pieces. The intuition is that if we place two pieces side by side, for pieces which don't belong together, there would be an edge detected on the border they share, as shown in 3. This can be implemented based on an edge detector.Specifically we used the Sobel operator H. The edge detector takes a convolution between H and the two strips of pixels on the boundary of x_i and x_j. Using Sobel operator of size 3, this yields

$$V(p, c) = \sum_{r=1}^{3} (H_{r1} x_i(p + r - 2, P, c) + H_{r2} x_j(p + r - 2, 1, c)) \tag{1}$$

And a dissimilarity of measurement can be defined as

$$S_{LR}(x_i, x_j) = \sqrt{\sum_{c=1}^{3} (\frac{1}{P} \sum_{p=1}^{P} V^2(p, c))} \tag{2}$$

3.3.2 Mahalanobis Gradient Compatibility (MGC)

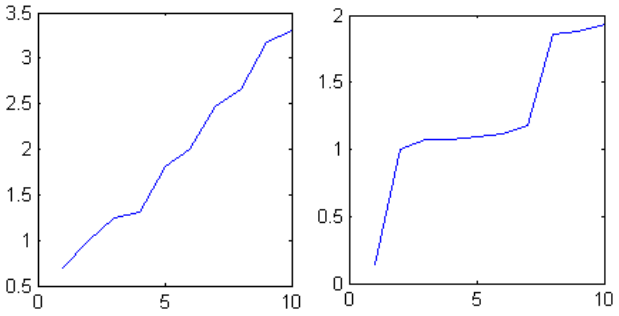
The measurement above can fail under certain circumstances. For example, suppose we have many puzzle pieces from an image of the sky. Chances are that there will be many pieces of similar color and it would be difficult to identify the one perfect match. Another example is that when there happens to be an edge occurred in the neighborhood of where the two puzzle pieces match, SCC would be expected to perform badly.

The observation from these examples is that instead of color intensity continuity, we could expect the intensity gradient to be continuous across the boundary. Using the rightmost two columns of pixels in x_i and the leftmost column of pixels in x_j, we first compute the gradient going from x_i to x_j. G_{iL} is the gradient on the boundary within x_i while G_{ijLR} is the gradient across the boundary.

$$G_{iL}(p, c) = x_i(p, P, c) - x_i(p, P - 1, c) \tag{3}$$

$$G_{ijLR}(p, c) = x_j(p, 1, c) - x_i(p, P, c) \tag{4}$$

Now that we have two gradient vectors, we take the Mahalanobis distance between the two vectors to see how different they are from each other. Mahalanobis distance differs from Euclidean distance in that it takes



(a) SCC (b) MGC

Figure 4: Top ten matches

into account the covariance between the color channels. We can think of each row in G_{iL} as a sample drawn from a distribution, and a small Mahalanobis distance from a new set of samples in G_{ijLR} to G_{iL} means that they have a similar distribution, which is exactly what we desire to achieve.The distance is defined as

$$D_{LR}(x_i, x_j) = (\sum_{p=1}^{P}(G_{ijLR}(p) - \mu_{iL})S_{iL}^{-1}(G_{ijLR}(p) - \mu_{iL})^T)^{\frac{1}{2}} \tag{5}$$

where μ_{iL} (1×3 vector) and S (3×3 matrix)is the mean and covariance matrix of the samples in G_{iL}

By symmetry, we can compute D_{RL} going from x_j to x_i. And finally we define a symmetric compatibility measure C_{LR}

$$C_{LR}(x_i, x_j) = D_{LR}(x_i, x_j) + D_{RL}(x_j, x_i) \tag{6}$$

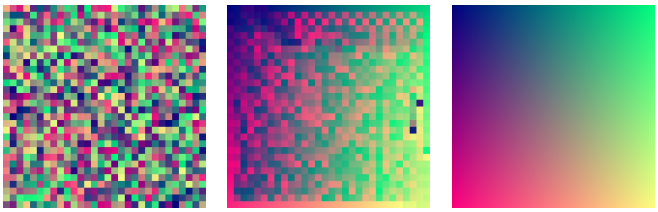
In practice we find that MGC almost always outperforms SCC. We have discussed that gradient based measurement captures the continuity of gradients and thus can handle more cases than SCC. Another advantage of MGC is that Mahalanobis distance treats the gradients to match as sets of sample points and measure the compatibility in a statistical sense. Therefore it's not as critical to have a perfect alignment of pixels sampled from two matching pieces, which makes MGC more suitable in the case of real puzzle pieces.

SCC and MGC both give a dissimilarity score for pairwise compatibility, meaning a small value indicates a high compatibility of the two pieces on the given edge. In order for a measurement to work well, we want it to not only find the correct match but also find the correct one with much "certainty". We can measure the "certainty" or "confidence" by taking the ratio between the minimum score and second minimum score (similar to the SIFT feature matching). The lower this ratio is, the more confident we are that we have made the correct match. Figure 4 shows the top 10 pieces found to match a single piece using both SCC and MGC, with the y-axis being the confidence ratio. MGC performs better in this case with a much lower ratio between the minimum and second minimum dissimilarity score.

More results on the performance of these two measurements can be found in [8] and [3].

4 Puzzle Assembly

With a pairwise compatibility measurement at hand, we use a greedy approach to assemble the puzzle pieces. The algorithm is simulating how humans usually solve a puzzle by putting together the border first and then filling in the interior pieces.



(a) Input (b) SCC (c) MGC

Figure 5: Test with 30×30 square pieces

4.1 Solving the border

Each border piece has at least one straight edge. If we rotate the border piece to have the straight edge on the bottom, then we will have a well defined left and right edge for this piece. For corner pieces, rotate the piece so that one straight edge is on the bottom and the other on the right, then define the left edge as left and the top edge as right. Among all the border pieces, we require that a left edge always connoted with a right edge, then to solve the border is to find a loop that goes through each border piece exactly once and have the minimum sum of compatibility scores on the edges they connect, which is equivalent to the traveling salesman problem (TSP). TSP is a well known NP problem and we try to find an approximate solution with a greedy heuristics.

Algorithm 1. Assembly the border

1. Construct a compatibility matrix $M(n \times n$, where n is the total number of border pieces). Element in row i column j in M gives the compatibility of piece i's left edge and piece j's right edge.

2.For each row of M, find the minimum two elements and take their ratio. Find the row with the minimum ratio and combines the two pieces (strips) corresponding to the minimum element in the row.

3.Update matrix M, each row and element corresponds to either a piece or a strip of pieces we have decided to put together (with a high confidence).

4.Repeat 2 and 3 until there are only a limited number of strips left. Test all the combinations of the strips to find one with correct dimension.

In experiments, we end in step 4 when we have reduced to no more than 9 strips. For every combination of these strips, we detect whether they have coherent dimensions by locating the corner pieces and see if their positions would guarantee a rectangular border.

4.2 Solving the interior

We again use a greedy approach to fill in the interior pieces. Since latter choices are made relying on former ones, we need to be very careful in every local optimal we commit in order to get a global optimal. We define a pocket ([4]) to be an unoccupied position with at least two adjacent pieces. By matching at least two edges every time, we would be more confident in our choice and improve the stability of our algorithm.

Algorithm 2. Assembly the interior

1.Find all the pockets

2.Among all the pockets find the most confident one to fill in

3.Recompute the pockets and repeat step 1-2

Again we evaluate the confidence of a pocket by taking the ratio of the best choice and the second best choice at the pocket. After step 2, there might be new pockets generated. Besides, since we have committed one puzzle piece in step 2, there are fewer available pieces to choose from for the rest of the pockets. Their best and second best choice can potentially be changed. Therefore we need to recompute them in step 3.

Figure5 shows the result of a test on the measurements and the whole framework with oriented square pieces.

Putting the shape and color measurement all together and using the framework above, we can extend

the algorithm to solve puzzles with classic pieces. To combine shape and color information, we use the shape measurement as a filter when computing compatibility between pieces. For pieces within certain threshold in shape measure, we compute their MGC as before and use the MGC for border/interior assembly. For the pieces that don't pass, we assign a large constant as their compatibility measure. Another challenge is that the pixel samples taken from two curved edges can be of different length and are hard to align. However, as discussed before, since we are using MGC for measurement, the sample alignment is not critical in the performance. In practice, we align pixel samples by their midpoints and take the length of the shorter between the two.

5 Results and Discussion

We tested our algorithm on multiple puzzles ranging from sizes 15 to 108 pieces. The assembled images are shown in Figure 6. The pieces for these puzzles were obtained from the site www.jigidi.com which is an online jigsaw puzzle website. The puzzle pieces from this site are notably not perfect: the pieces have noticable shadow artifacts on the edges of the pieces and have in some areas specular highlights. Despite these issues, our algorithm correctly assembles the puzzles ranging in size from 15 to 91 pieces in contrast to the work of [8] which only assembled a 54 piece puzzle that had been carefully digitized to avoid artifacts.

When we tested on puzzles higher than 91 pieces, we were unable to correctly assemble the border. The nature of the problem (TSP) makes our greedy approach inherently difficult to scale. Our approach now is to essentially make easy decisions first and postpone the hard ones. If we can get the size of problem down before we have to make any tough decisions, we can then avoid making those decisions by doing an exhaustive search in the reduced space. However, as the problem size grows, tough decisions are inevitably made before we can get down to a size small enough for exhaustive search. Besides, currently we are dynamically detecting the dimension of the puzzle pieces by shape and color measurement with only a rectangular constraint. If we assume the dimensions are known, we might be able to scale up to more pieces.

6 Conclusion

In summary, this work introduces a new approach to assembling puzzles made from real world pieces. We give an approach based upon using shape as a filter and then color as the piece compatibility measure. We use blah for assembly. We also give a novel method using a z-buffer approach to compute shape similarity for two edges. Our overall algoirthm gives improved performance with a correct reconstruction of a 91 piece puzzle.

In the future, we would like to continue refining our approach. In terms of puzzle assembly, as discussed in previous section, the performance of greedy algorithm depends largely on how difficult it is to make a decision (commit a local optimal) in each step. [9] and [3] provides some insight into how to push this idea further by first making easy decisions to generate groups of puzzle pieces and later consider some sort of global optimization to glue the groups together coherently. However, both of them assume square pieces and [9] further requires oriented pieces and known dimension, thus not trivial to extend to real jigsaw puzzles.

References

[1] E. Demaine and M. Demaine. Jigsaw puzzles, edge matching, and polyomino packing: Connections and complexity. *Graphs and Combinatorics*, 23, 2007.

[2] M. Frenkel and R. Basri. Curve Matching Using the Fast Marching Method. In *EMMCVPR*, 2003

[3] A. C. Gallagher Jigsaw Puzzles with Pieces of Unknown Orientation In *Proc. CVPR*, 2012

[4] D. Goldberg, C. Malon, and M. Bern. A global approach to automatic solution of jigsaw puzzles. In *Symposium on Computational Geometry*, 2002

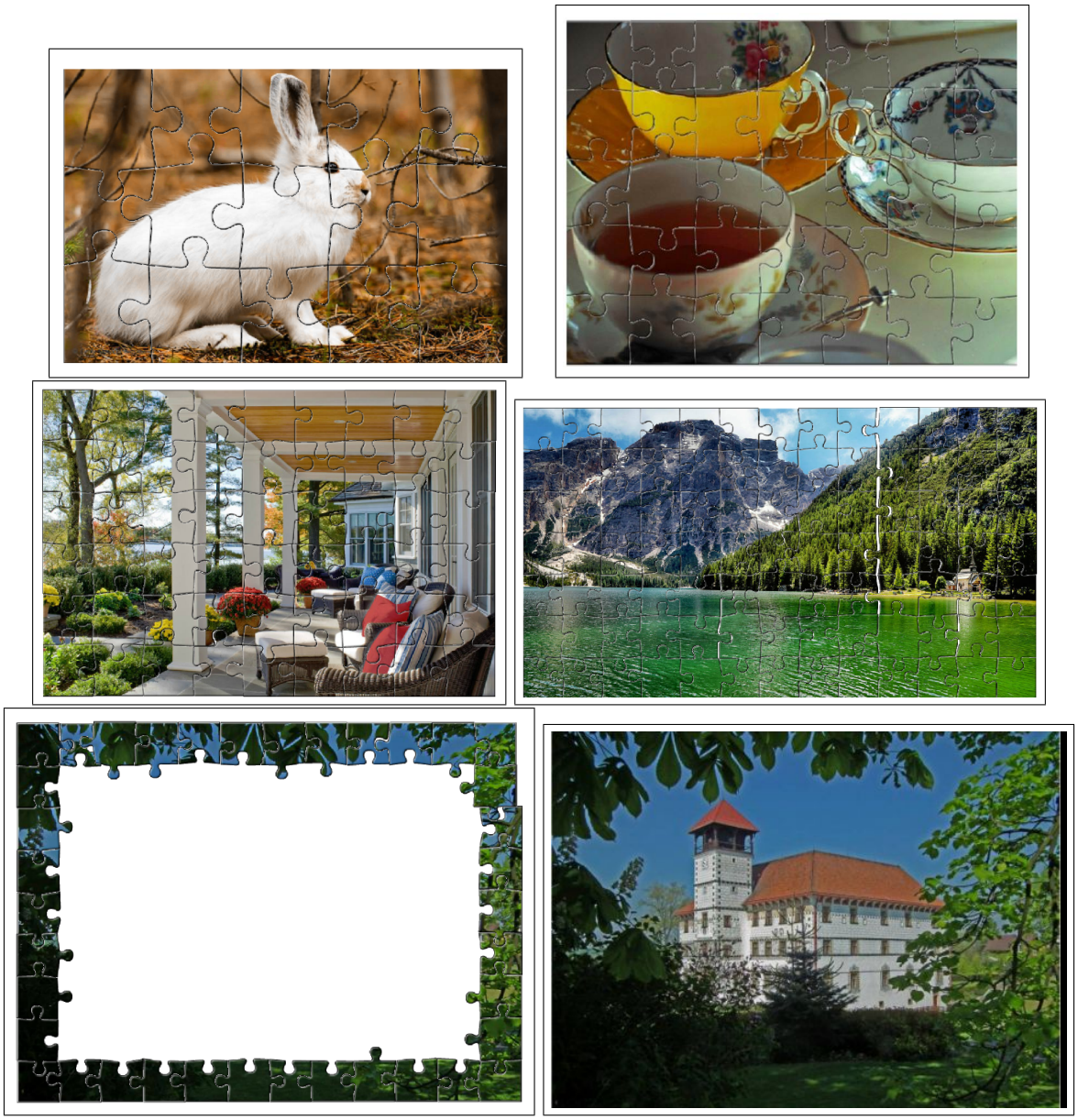


Figure 6

[5] H. Freeman and L. Garder. Apictorial jigsaw puzzles: the computer solution of a problem in pattern recognition. *IEEE TEC*, (13):118–127, 1964

[6] W. Kong and B. B. Kimia. On solving 2D and 3D puzzles using curve matching. In *IEEE CVPR*, 2001

[7] M. Makridis and N. Papamarkos. A new technique for solving a jigsaw puzzle. In *IEEE ICIP*, 2006.

[8] T. R. Nielsen, P. Drewsen and K. Hansen Solving jigsaw puzzles using image features. In *Pattern Recognition Letters 29* 2008

[9] D. Pomeranz, M. Shemesh, and O. Ben-Shahar. A fully automated greedy square jigsaw puzzle solver. In *Proc. CVPR*, 2011.

[10] S. Mistry, U. N. Niranjan and M. Gopi Puzzhull: Cavity and Protrusion Hierarchy to Fit Conformal Polygons In *Computer-Aided Design* 2014