# Acquisition of 2-D Shape Models from Scenes with Overlapping Objects Using String Matching

Horst Bunke and Marcel Zumbühl
University of Bern, Institut für Informatik und angewandte Mathematik,
Neubrückstrasse 10, CH 3012 Bern, Switzerland
{bunke, zumbuehl}@iam.unibe.ch

**Abstract.** In this paper we describe a system that is able to acquire models of 2-D shapes from cluttered scenes. The input of the system is a sequence of images each of which shows an unknown number of overlapping unknown 2-D objects. The system identifies matching partial shapes across different images and combines them into complete 2-D shape models thus giving a complete interpretation of the input scenes. The identification of partial shapes is based on string matching, whereas a graph search procedure is used for shape model generation. The system has been fully implemented and tested on images containing parts of a jigsaw puzzle.

## 1 Introduction

Model based recognition of objects, particularly in the two-dimensional case, has reached a high level of maturity ([1], [2], [3], [4], [5], [6], [7], [8], [9]). But the problems of automatic model acquisition and learning are only partially solved. Typically, object models are still constructed by hand. This process is time consuming and error prone. Moreover, the recognition of instances of a model under noisy conditions in the input image may be difficult.

A general introduction to the area of machine learning is provided in [10], [11]. Machine learning in the field of computer vision is addressed in [12], [13]. In this paper we describe a method that is able to infer shape models of 2-D objects from cluttered scenes. Input for the system is a sequence of scenes, where each scene shows a collection of overlapping objects. The system attempts to identify similar partial shapes and integrates them to complete shape models. Shapes are formally represented by sequences, or strings, of local curvature values and the identification of similar partial shapes is based on string matching [14]. The representation is intrinsically invariant under translation and rotation. Moreover the learning and recognition procedure has the potential of invariance under scaling.

In the following, we consider an introductory example. Assume we are observing the image shown in Fig. 1(a). Can we draw any conclusions from this image about the number and shape of objects involved? Certainly not – at least not as

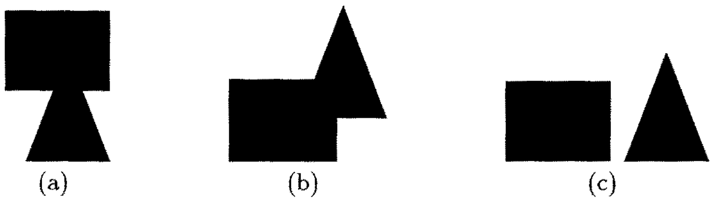


**Fig. 1.** An example: (a) first observed image; (b) second observed image; (c) potential shape models that may have generated the images in (a) and (b)

long as we don't make any assumptions about the underlying generic class of objects. However, if we additionally consider the image in Fig. 1(b), we can identify matching parts of the contours of the shapes in Figure 1(a) and Figure 1(b), and can conclude that Figures Fig. 1(a) and Fig. 1(b) may have been generated from the two shapes shown in Fig. 1(c) . There are, of course, infinitely many other objects that may have generated both Fig. 1(a) and Fig 1(b).

The general task considered in this paper can be described as follows. Given a sequence of images $I_1, I_2, \ldots, I_n$, where each image $I_j$ contains a shape $S_j$, infer a set of 2D-object models $M_1, \ldots, M_k$ such that each shape $S_j$ is potentially generated by overlapping all or some of the $M_i$'s. Throughout this paper we assume that our shape models may undergo any translation and rotation in the observed images.

## 2 Detecting local shape matches

The method for shape model generation consists of two phases: (1) detection of local shape matches, (2) synthesis of shape models from local matches. In this section, we describe the first phase. It is similar to the method presented in [14].

The curvature of the boundary of a shape is used as shape representation. It is measured at discrete points. Thus the curvature of a shape is given by a sequence of numbers $\underline{x} = (x_1 \ldots x_n)$ where each $x_i$ is an angle $-180° < x_i \leq +180°$. The $x_i$'s are quantized values. Hence every shape is represented by a string of symbols from a finite alphabet.

Given two shapes described by their boundary strings $\underline{x} = (x_1 \ldots x_n)$ and $\underline{y} = (y_1 \ldots y_m)$, we can measure their similarity using a string matching algorithm. Assume for the moment that the two strings are in registration with each other, i.e., the first symbol $y_1$ of $\underline{y}$ represents the point on the contour that corresponds to the first symbol $x_1$ of $\underline{x}$. We can then use the standard dynamic programming algorithm for string matching [15]. This algorithm computes the string edit distance between $\underline{x}$ and $\underline{y}$, $d(\underline{x}, \underline{y})$, which is equal to the minimum cost sequence of edit operations that transform $\underline{x}$ into $\underline{y}$. Clearly, if $\underline{x}$ and $\underline{y}$ represent the same shape then $d(\underline{x}, \underline{y}) = 0$ in the ideal case. If there is noise in the data, $d(\underline{x}, \underline{y})$ will be greater than zero. But we can still expect $d(\underline{x}, \underline{y})$ to be smaller than $d(\underline{x}, \underline{z})$ if $\underline{z}$ is a different shape. In order to make the algorithm applicable in practice, appropriate costs have to be defined for each of the edit operations substitution insertion, and deletion. As the symbols in a string represent angles, it is natural to define the cost of a substitution $c(a \to b) = |a - b|$, where $a$ and $b$ are integers from the interval $[0, 360]$. This definition of substitution cost has actually been used in [14]. Moreover $c(a \to \epsilon)$ and $c(\epsilon \to a)$, i.e. the cost of deleting and inserting a symbol, respectively, have been set to the constant 120. (This constant was experimentally determined.)

Clearly, the assumption that $\underline{x}$ and $\underline{y}$ are in registration with each other is not realistic. In [14] a procedure to overcome this assumption was described that only has a time complexity of $O(n \cdot m)$. That is, it has the same time complexity as the basic string matching algorithm [15]. In the procedure described in [14] two identical copies of string $\underline{y}$, i.e., $\underline{y}^2 = \underline{yy}$ are considered, instead of $\underline{y}$. Furthermore the dynamic programming procedure is modified such that leading symbols in $\underline{y}^2$ can be skipped at no cost. Consequently, if one of the strings is a cyclically rotated version of the other, i.e., $\underline{y} = x_i \dots x_n x_1 \dots x_{i-1}$ then the edit distance between $\underline{x}$ and $\underline{y}$ is equal to zero. As the underlying boundary representation is invariant under rotation the resulting algorithm is able to recognize shape similarity in presence of translation and rotation.

The string matching algorithm described until now can be further extended to detecting partially matching shapes. Formally, we want to find out if there are substrings $x_i \dots x_{i+l}$ and $y_j \dots y_{j+k}$ of $\underline{x}$ and $\underline{y}$, respectively, that have a small edit distance to each other. If such substrings exist we can conclude that the corresponding partial shapes are similar.

For finding partial shape matches we compute the rotation invariant edit distance between $\underline{x}$ and $\underline{y}$ for each cyclically shifted version $\underline{x}^{(i)}$ of $\underline{x}$, with $\underline{x}^{(i)} = x_i \dots x_n x_1 \dots x_{i-1}$ and $i = 1, \dots, n$. The edit distance computation algorithm proceeds, analogously to the methods described in [15], by filling in the elements of a matrix. Each row of this matrix corresponds to one symbol of $\underline{x}$. After the matrix has been filled, we scan it from the last row towards the first row and search for an element representing a match that exceeds a certain length and has a small cost. Once an appropriate match has been found, we record it and stop scanning the matrix. Repeating this procedure for each cyclically shifted version of $\underline{x}$ we get a set that contains all potential partial matches of the two shapes. However, some of these partial matches may be in conflict with each other. In a postprocessing phase, the conflicting partial matches are filtered out. Let $\alpha$ and $\alpha'$ be substrings of $\underline{x}$, and $\beta$ and $\beta'$ substrings of $\underline{y}$. Furthermore, let $(\alpha, \beta)$ and $(\alpha', \beta')$ be partial matches that have been recorded. If $\alpha$ is a substring of $\alpha'$ or $\beta$ a substring of $\beta'$, then the pair $(\alpha, \beta)$ is removed. Moreover, if there is an overlap between $\alpha$ and $\alpha'$ or $\beta$ and $\beta'$, only the longer of the two partial matches is kept.

Finally, for a given pair of shapes $\underline{x}$ and $\underline{y}$, we get a set of matching partial shapes $(\alpha_1, \beta_1), \dots, (\alpha_n, \beta_n)$ where each $\alpha_i$ is a substring of $\underline{x}$, each $\beta_i$ is a substring of $\underline{y}$, and all $\alpha_i$'s and $\beta_i$'s are disjoint. The complexity of this method is $O(n^2 \cdot m)$. For further details see [14].

An example of the detection of partial shape matches is given in Fig. 2. In Fig. 2(a) two input shapes are shown. In Fig. 2(b) the partial matches that were detected are highlighted.

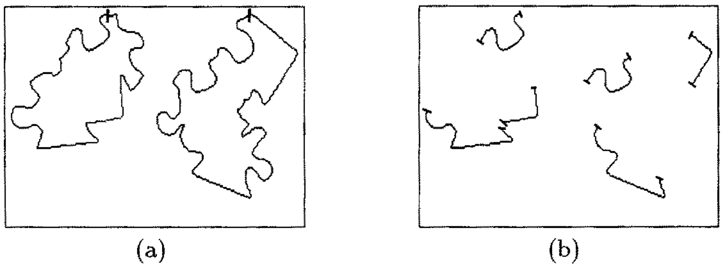


**Fig. 2.** Example of partial shape matching; (a) two given shapes; (b) partial matches detected by the algorithm

## 3 Synthesis of shape models

Given a sequence of shapes $\Sigma_1, \ldots, \Sigma_N$ each contained in an image, we match each pair of shapes, $\Sigma_i$ and $\Sigma_j$, $i \neq j$, and get, according to the procedure described in Section 2, for each pair of shapes a list of partial matches

$$M_{ij} = ((\alpha_1, \beta_1), ...(\alpha_t, \beta_t))^1 \tag{1}$$

If $\Sigma_i$ is represented by string $\underline{x} = x_1 \ldots x_n$ and $\Sigma_j$ by string $\underline{y} = y_1 \ldots y_m$, then each $\alpha$ is a substring of $\underline{x}$ and each $\beta$ a substring of $\underline{y}$. In other words,

$$\begin{aligned} \underline{x} &= x_1 \ldots x_n = \gamma_1 \alpha_1 \gamma_2 \alpha_2 \ldots \alpha_t \gamma_{t+1} \\ \underline{y} &= y_1 \ldots y_m = \delta_1 \beta_1 \delta_2 \beta_2 \ldots \beta_t \delta_{t+1}, t \geq 0 \end{aligned} \tag{2}$$

where the $\gamma$'s and $\delta$'s are the non-matching parts of $\Sigma_i$ and $\Sigma_j$, respectively, and $(\alpha_1, \beta_1), \ldots, (\alpha_t, \beta_t)$ are the matching partial shapes. Note that $M_{ij}$ may be empty, i.e., t=0.

Our procedure for shape model synthesis is based on the observation that partial shapes occuring more than once in different shapes $\Sigma_i$ are very likely to be part of a model. Thus the synthesis procedure attempts to collect matching partial shapes between different pairs of shapes $\Sigma_i$ and $\Sigma_j$ and integrate them by proper concatenation into complete models.

Starting point for the model synthesis procedure is the so-called *segmentation graph*, which is built from the partial matches $M_{ij}$; $i, j = 1, \ldots, N$. Intuitively, this graph represents the maximal segmentation of each shape taking all available information from all partial matches $M_{ij}$ into account. Formally, the segmentation graph consists of nodes and edges. Each node represents a substring $x_k \ldots x_l$ of any of the shapes $\Sigma_i$. The edges represent the concatenation operation. That is, if node $\nu$ represents substring $x_k \ldots x_{k+l}$ of shape $\Sigma_i$ and node $\mu$ represents substring $x_{k+l+1} \ldots x_s$ of the same shape, then there is an edge from $\nu$ to $\mu$ with label $\Sigma_i$. Strings are considered cyclically, i.e., if $\underline{x} = x_1 \ldots x_r x_{r+1} \ldots x_s x_{s+1} \ldots x_n$ and substrings $x_1 \ldots x_r$ and $x_{s+1} \ldots x_n$ both

[1] To keep the notation simple, the dependency of the parameter $t$ as well as of the $\alpha$'s and $\beta$'s on $i$ and $j$ is not explicitly shown

are represented by a node, then we insert an edge from the node representing $x_{s+1} \ldots x_n$ to the node representing $x_1 \ldots x_r$.

The segmentation graph is constructed as follows. Given a list $M_{ij}$ of partial matches for a pair of shapes $\Sigma_i$ and $\Sigma_j$ according to (1), we partition strings $\underline{x}$ and $\underline{y}$ according to (2). For each matching pair $(\alpha, \beta)$, each $\gamma$, and each $\delta$ we generate a node and subsequently connect the nodes with edges as described above. This procedure is repeated for all pairs of shapes. Identical nodes, i.e., nodes that represent the same substring, are generated only once. If for pairs $(\Sigma_i, \Sigma_j)$ and $(\Sigma_i, \Sigma_k)$, $j \neq k$, two nodes are obtained that overlap each other, we split them.

Splitting is accomplished as follows. Let $\Sigma_i$ be represented by string $\underline{x}$. If for pair $(\Sigma_i, \Sigma_j)$ node $\nu$ represents substring $x_r \ldots x_{s-1} x_s \ldots x_{t-1}$ of shape $\Sigma_i$ and for pair $(\Sigma_i, \Sigma_k)$ node $\mu$ represents substring $x_s \ldots x_{t-1} x_t \ldots x_{l-1}$ of the same shape, then node $\nu$ is split into $\nu_1$ representing $x_r \ldots x_{s-1}$ and $\nu_2$ representing $x_s \ldots x_{t-1}$. In a similar way, $\mu$ is split into $\mu_1$ representing $x_s \ldots x_{t-1}$ and $\mu_2$ representing $x_t \ldots x_{l-1}$. The nodes $\nu_2$ and $\mu_1$ are identified with each other as they represent the same partial shape. A similar splitting operation is applied if the string represented by one node is a proper substring of the string represented by another node.

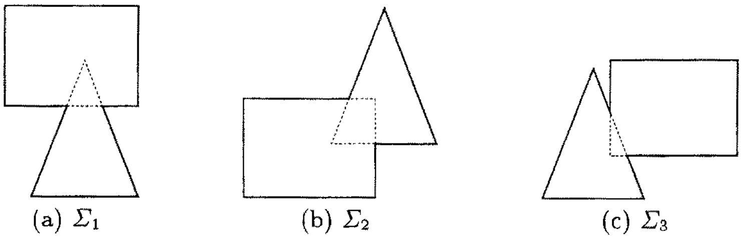


**Fig. 3.** Example shapes built up by a triangle and a rectangle (dotted lines)

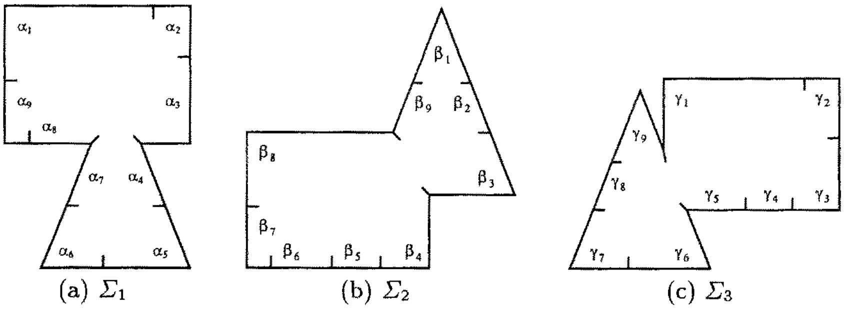


**Fig. 4.** Partitioned shapes after matching

$$
\begin{array}{ll}
\underline{x} = \alpha_1 \ldots \alpha_9 & M_{12} = ((\alpha_3, \beta_4), (\alpha_4\alpha_5, \beta_2\beta_3), (\alpha_7, \beta_9), (\alpha_8\alpha_9\alpha_1, \beta_6\beta_7\beta_8)) \\
\underline{y} = \beta_1 \ldots \beta_9 & M_{13} = ((\alpha_1\alpha_2\alpha_3, \gamma_1\gamma_2\gamma_3), (\alpha_5\alpha_6\alpha_7, \gamma_6\gamma_7\gamma_8), (\alpha_8, \gamma_5)) \\
\underline{z} = \gamma_1 \ldots \gamma_9 & M_{23} = ((\beta_3, \gamma_6), (\beta_4\beta_5\beta_6, \gamma_3\gamma_4\gamma_5), (\beta_8, \gamma_1), (\beta_9\beta_1, \gamma_8\gamma_9))
\end{array}
$$

**Fig. 5.** List of partial matches and string representations of the shapes

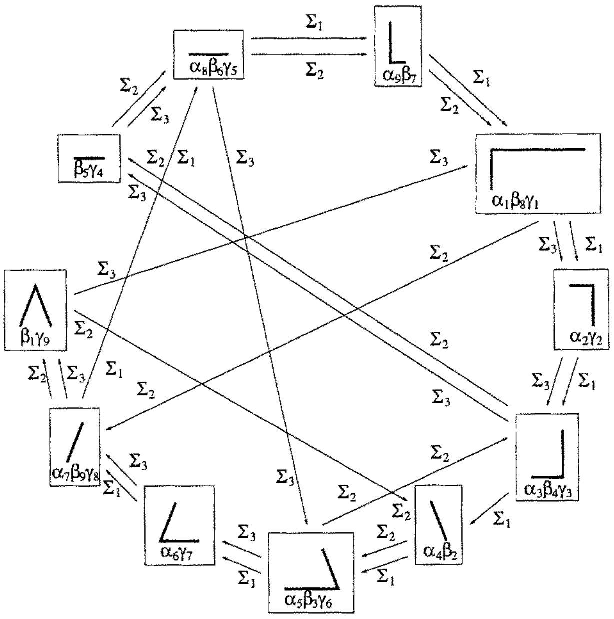


**Fig. 6.** Segmentation graph for $\Sigma_1$, $\Sigma_2$ and $\Sigma_3$. The nodes represent substrings of each $\Sigma_i$ to which the indicated shape segments belong (e.g. the node labelled $\alpha_9\beta_7$ represents a shape segment of both $\Sigma_1$ and $\Sigma_2$)

For example, let's assume that the shapes $\Sigma_1$, $\Sigma_2$ and $\Sigma_3$ shown in Fig. 3 are generated by a triangle and a rectangle. (For the purpose of simplicity, we furthermore assume that the shape models may be translated but not rotated.) The partitioning of the shapes after matching every shape to every other and the related partial matches are shown in Figs. 4 and 5, respectively. The corresponding segmentation graph is given in Fig. 6.

About Fig. 6 we can make a number of observations: First, each of the shapes $\Sigma_i, 1 \leq i \leq 3$, is represented by a cycle whose edges are labeled with $\Sigma_i$. Secondly, if a node represents a partial match $(\zeta, \xi)$ from $M_{ij}$, then it has incident edges labeled with $\Sigma_i$ and $\Sigma_j$. Moreover, we observe that the cycles leading through nodes containing $(\alpha_8, \alpha_9, \alpha_1, \alpha_2, \alpha_3, \beta_5)$ and $(\beta_1, \beta_2, \beta_3, \gamma_7, \beta_9)$ correspond to the shape models that generated the shapes $\Sigma_1$, $\Sigma_2$ and $\Sigma_3$. On the other hand, not every cycle corresponds to a possible shape model.

Based on these observations, the algorithm for shape model synthesis proceeds as follows. It sequentially considers each node in the segmentation graph as starting node of a cycle. For each starting node, it enumerates all cycles that pass through it. If a cycle represents a closed contour and contains edges labeled with two different shape identifiers $\Sigma_i$ and $\Sigma_j$, $i \neq j$, at least, it is kept and recorded as a model. Otherwise it is discarded. All nodes and edges that belong to a potential model are deleted from the segmentation graph. This process is repeated as long as new cycles are found.

If for example the shape model synthesis procedure receives the segmentation graph given in Fig. 6 as input, the algorithm yields the two cycles shown in Fig. 7. Theses two cycles represent the two model shapes in Fig. 8(a) and Fig. 8(b).

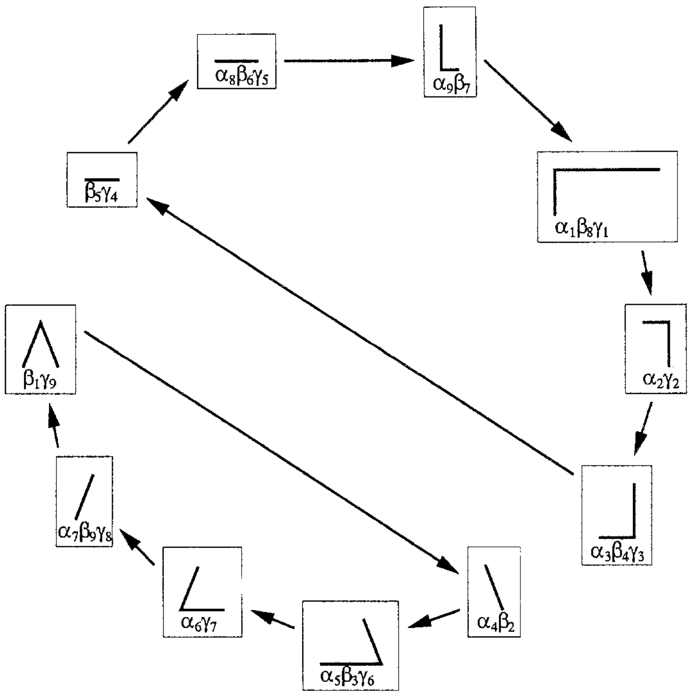

$\alpha_8\beta_6\gamma_5$
$\alpha_9\beta_7$
$\alpha_1\beta_8\gamma_1$
$\alpha_2\gamma_2$
$\alpha_3\beta_4\gamma_3$
$\alpha_4\beta_2$
$\alpha_5\beta_3\gamma_6$
$\alpha_6\gamma_7$
$\alpha_7\beta_9\gamma_8$
$\beta_1\gamma_9$
$\beta_5\gamma_4$

**Fig. 7.** Cycles within the segmentation graph

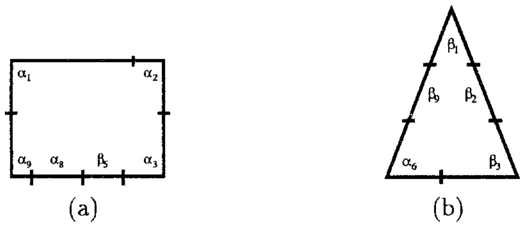


**Fig. 8.** Resulting model shapes. For each segment only one of the labels is given

## 4 Experimental results

The method described in Sections 2 and 3 has been implemented and tested on a number of image sequences. The models that were used for the generation of the scenes were jigsaw puzzle parts, similar to those used in [14]. The sequences presented to the system consisted of four to eight images with either two or three models involved. All computation times reported in this section were measured on a SUN Ultra 1 (Model 170E) station.

In Fig. 9 a sequence consisting of four images is shown. From this sequence the system has identified two shape models. These models are superimposed to the original images in Fig. 10(a) and (b). In this example the segmentation graph consisted of 85 nodes. The model synthesis as described in section 3 took about 24 seconds computation time.

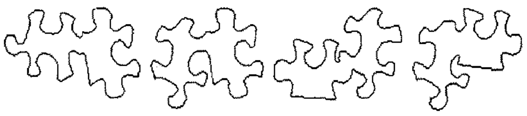
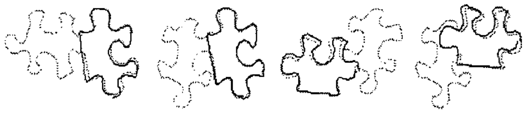
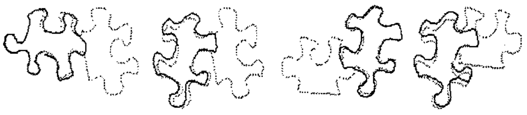


**Fig. 9.** A sequence of four images



(a) First model superimposed to Fig. 9



(b) Second model superimposed to Fig. 9

**Fig. 10.** The system has identified two different shape models from Fig. 9

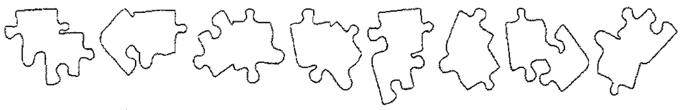
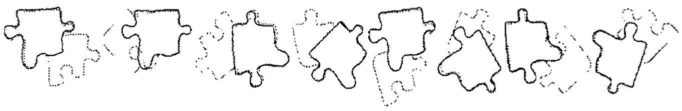
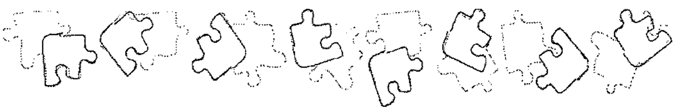


**Fig. 11.** Another sequence of eight images



(a) First model superimposed to Fig. 11



(b) Second model superimposed to Fig. 11

**Fig. 12.** The system has identified two different shape models from Fig. 11

Another sequence and the corresponding results are shown in Fig. 11 and Fig. 12, respectively. In this example, the segmentation graph consisted of 96 nodes. About 26 seconds computation time were needed for model synthesis.

A sequence of eight images involving three different shapes is shown in Fig. 13. The shape models that were identified are presented in Fig. 14. In this example, the segmentation graph consisted of 613 nodes. About 743 minutes computation time were needed for shape synthesis.

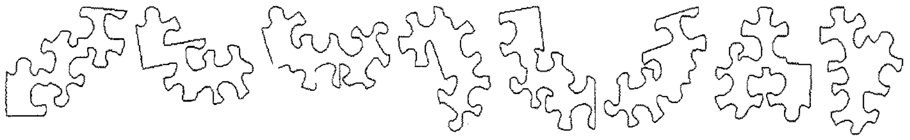
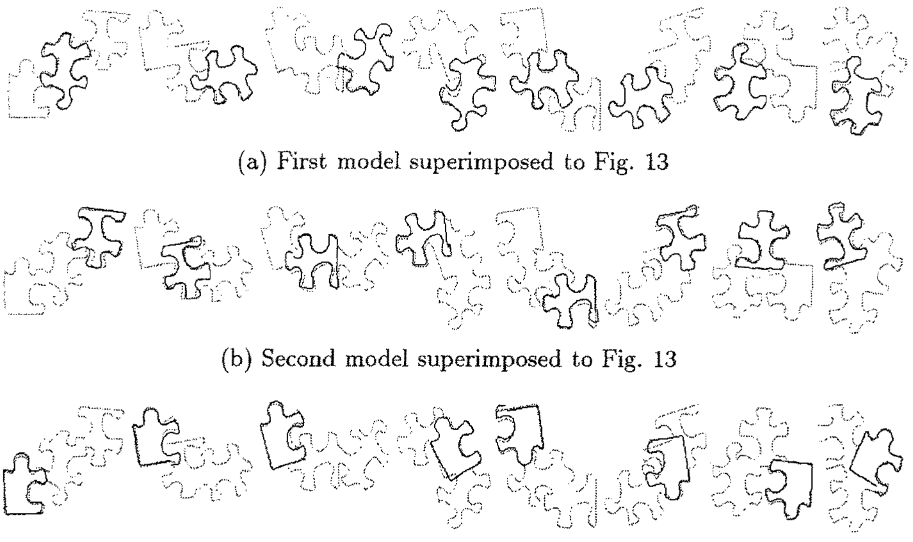


**Fig. 13.** A third sequence containing eight images



(a) First model superimposed to Fig. 13

(b) Second model superimposed to Fig. 13

(c) Third model superimposed to Fig. 13

**Fig. 14.** The system has identified three different shape models from Fig. 13

Our method is based on the assumption that each segment of the model shapes can be observed at least in two of the input scenes. If an input sequence does not meet this constraint the model synthesis procedure will eventually not provide a correct result. Moreover, if the models forming the scenes are heavily overlapping each other the partial shape matching algorithm will deliver only short partial matches. The shorter and more numerous these partial matches are, the more likely is the graph search procedure to fail to return correct results.

## 5 Summary and conclusions

In this paper, we have presented a novel approach to the learning of shape models from cluttered scenes. The method consists of two steps. First, matching partial shapes across different images are identified. This procedure is based on string matching. Secondly, partial matching shapes are assembled into complete shape models using a graph search procedure. The proposed method has been implemented and tested on images containing overlapping parts of jigsaw puzzles.

## References

[1] D.H. Ballard, *Generalizing the Hough Transform to Detect Arbitrary Shapes*, Pattern Recognition, Vol.13, 1981, pp. 111–122

[2] J.L. Turne, T.N. Mudge, R.A. Volz, *Recognizing Partially Occluded Parts*, IEEE Trans. Pattern Anal. Mach. Intell., Vol.7, No.4, 1985, pp. 410–421

[3] W.E.L. Grimson and T. Lonyano–Perez, *Localizing Overlapping Parts by Searching the Interpretation Tree*, IEEE Transc. Pattern Anal. Mach. Intell., Vol.9, No.4, 1987, pp. 469-482

[4] N. Ayache, O.D. Faugeras, *HYPER: a New Approach for the Recognition and Positioning of two-dimensional Objects*, IEEE Trans. Pattern Anal. Mach. Intell., Vol.8, No.1, 1986, pp. 44–54

[5] Y.-T. Tsay, W.-H. Tsai, *Model-guided Attributed String Matching by Split-and-merge for Shape Recognition*, Int. J. Pattern Recognition Artif. Intell. No.3, 1989, pp. 159–179

[6] H. Bunke, T. Glauser, *Viewpoint Independent Representation and Recognition of Polygonal Faces in 3-D*, IEEE Trans. Robotics Automat., Vol.9, No.4, 1993, pp. 457–462

[7] H.J. Wolfson, *Model-based Object Recognition by Geometric Hashing*, Proc. 1st Eur. Conf. Comput. Vision Antibes, 1990, pp. 526–536

[8] D. Huttenlocher, G. Klaudermann, W. Ruckblidge, *Comparing Images Using the Hausdorff Distance*, IEFF Trans. Pattern Anal. Mach. Intell., Vol.15, No.9, 1993, pp. 850–863

[9] K.Y. Kupeev, H.J. Wolfson, *A New Method of Estimating Shape Similarity*, Pattern Recognition Letters, No.17, 1996, pp. 873–887

[10] A. Hutchinson, *Algorithmic Learning*, Oxford University Press, 1994

[11] J.W. Shavlick, T.G. Dietterich (eds.), *Readings in Machine Learning*, Morgan Kaufman, San Mateo, 1990

[12] Y. Kodratoff, S. Moscatelli, *Machine Learning for Object Recognition and Scene Analysis*, Int. Journal of Pattern Recognition and Artificial Intelligence, Vol.8, No.1, 1994, pp. 259–305

[13] B. Bhanu, T. Poggio (guest eds.), *Special Section on Learning in Computer Vision*, IEEE Trans. Pattern Anal. Mach. Intell., Vol.16, No.9, 1994, pp. 865–919

[14] H. Bunke, U. Bühler, *Applications of Approximate String Matching to 2D Shape Recognition*, Pattern Recognition, Vol.26, No.12, 1993, pp. 1797–1812

[15] R.A. Wagner, M.J. Fischer, *The String-to-string Correction Problem*, Jour. ACM, Vol.21, 1994, pp. 168–173