Matching Concavity Trees

Ossama El Badawy and Mohamed Kamel

Pattern Analysis and Machine Intelligence Laboratory
Department of Systems Design Engineering
University of Waterloo, Waterloo, ON, N2L 3G1, Canada
{obadawy,mkamel}@pami.uwaterloo.ca

Abstract. Concavity trees are structures for 2-D shape representation. In this paper, we present a new recursive method for concavity tree matching that returns the distance between two attributed concavity trees. The matching is based both on the structure of the tree as well as on the attributes stored at each node. Moreover, the method can be implemented on parallel architectures, and it supports occluded and partial matching. To the best of our knowledge, this is the first work to detail a method for concavity tree matching. We test our method on 625 silhouettes in the context of shape-based nearest-neighbour retrieval.

1 Introduction

A new concavity-tree extraction algorithm and a method for shape-based image retrieval have been proposed in [1]. This paper complements that work by detailing a concavity-tree matching method. Together, the two papers constitute, to the best of our knowledge, the first work that reports on the applicability of concavity trees to shape-based image retrieval. The proposed matching method is well suited to labelled concavity trees, but is by no means confined to them; it can be applied to the matching of any labelled trees.

A *concavity tree* is a data structure used for describing non-convex two dimensional shapes. It was first introduced by Sklansky [2] and has since been further researched by others [3–6]. A concavity tree is a rooted tree where the root represents the whole object whose shape is to be analysed/represented. The next level of the tree contains nodes that represent concavities along the boundary of that object. The following level contains nodes, each representing one of the concavities of its parent, *i.e.*, its meta-concavities. If an object or a concavity is itself convex, then the node representing it does not have any children. Typically, each node in a concavity tree stores attributes, or features, describing the underlying object or concavity. Concavity trees are clearly a tool for structural pattern recognition in which a pattern (a 2-D shape in our case) is decomposed into its primitive components. Figure 1 shows an example of a shape (a), its convex hull, concavities, and meta-concavities (b), and its corresponding concavity tree (c). The shape has $five$ concavities as reflected in level *one* of the tree. The four leaf nodes in level *one* correspond to the highlighted triangular concavities shown in (d), whereas the non-leaf node corresponds to the (non-convex)

A. Fred et al. (Eds.): SSPR&SPR 2004, LNCS 3138, pp. 556–564, 2004.

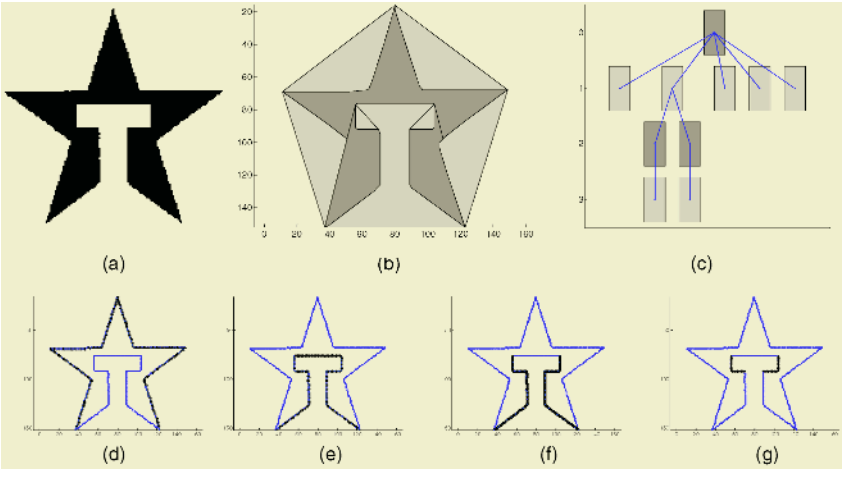

Fig. 1. An object (a), its convex hull and concavities (b), the corresponding concavity tree (c), and contour sections corresponding to concavities (d-g).

concavity shown in (e). Similarly, the nodes in levels *two* and *three* correspond to the meta-concavities highlighted in (f) and (g), respectively. The tree shown in (c) was extracted using the contour-based algorithm presented in [1]. That algorithm is able to deal with the many minor "noisy" concavities seen along the boundary of the shape in (a). In addition to the feature vector labelling the tree nodes, other "helper" features are computed by the tree extraction algorithm and stored at each node (as the tree is being constructed.) They include the level of the node, the height, number of nodes, number of leaves in the subtree rooted at the node, as well as the relative size of the concavity (represented by the node) with respect to its parent.

Structural matching of trees is computationally expensive, especially if the trees are not rooted. Three (structural) tree matching classes exist [7]: perfect matching (isomorphism), partial matching, and similarity matching. The third class is more useful for classification and retrieval purposes. Inexact and heuristic methods for similarity matching of trees exist in the literature in many contexts including shock tree matching [8], remote sensing [9], and VLSI [10].

The decision of whether two objects have similar shapes or not is based on both the shape of the whole object as well as the shapes of the objects' primitives. Moreover, shape can be described using high- and low-level features. The main goal of the proposed matching method is to enhance image similarity retrieval by incorporating high-level structural shape information (trees) as well as low-level shape information (labels) of the whole object and its primitives into the matching process. The analysis and matching of single-object logo images is one viable application of the proposed method.

2 Concavity-Tree Matching

This section details the proposed concavity-tree matching method. Suppose we would like to find a measure of similarity between two general labelled trees T and S. We assume the following:

- Number of nodes in T (S) is n (m), where $n > 0$ ($m > 0$.)
- Number of subtrees rooted in level 1 in T (S) is N (M), where $N \geq 0$ ($M \geq 0$.)
- $M \geq N$. If this is not the case, we just switch the names of the two trees. (Note that $M \geq N$ does not necessarily imply that $m \geq n$.)
- $A = \{T^{(1)}, \cdots, T^{(N)}\}$ and $B = \{S^{(1)}, \cdots, S^{(M)}\}$ are the sets of subtrees rooted in level *one* of T and S, respectively.
- Nodes in T and S are labelled with l-dimensional feature vectors $\mathbf{x}_i = (x_{i_1}, \cdots, x_{i_l})$ and $\mathbf{y}_j = (y_{j_1}, \cdots, y_{j_l})$, where i and j denote the nodes of T and S, respectively, $0 \leq i < n$, and $0 \leq j < m$.
- $\mathbf{x}_0$ and $\mathbf{y}_0$ are the attributes of the root nodes of T and S, respectively.
- Each of the individual features, x_i (y_i) in vector $\mathbf{x}$ ($\mathbf{y}$) is bounded between *zero* and a maximum value max_i, $1 \leq i \leq l$.

The proposed method computes a distance $\mathcal{D}(T, S)$ between the two trees as a measure of dissimilarity. The closer the distance is to *zero*, the more the two shapes are similar and the closer it is to *one*, the more dissimilar they are.

In matching the two trees, we can identify four cases as follows:

1. $N = M = 0$.
2. $N = M, N > 0, M > 0$.
3. $N = 0, M > 0$.
4. $N > 0, M > 0, M > N$ (the most general case.)

2.1 Case 1, $N = M = 0$

This case is the simplest case (see Figure 2a); there is no structural matching and the distance between T and S is

$$\mathcal{D}(T, S) = a\, d(\mathbf{x}_0, \mathbf{y}_0) \tag{1}$$

where a is a constant between *zero* and *one* (discussed later.) Typically, $a = 1/3$. $d(\mathbf{x}, \mathbf{y})$ is defined as follows.

$$d(\mathbf{x}, \mathbf{y}) = \frac{1}{\sqrt{\sum_{i=1}^{l} max_i^2}} \sqrt{\sum_{i=1}^{l} (x_i - y_i)^2} \tag{2}$$

That is, $d(\mathbf{x}, \mathbf{y})$ is the Euclidean distance between $\mathbf{x}$ and $\mathbf{y}$ normalized by the maximum possible distance between $\mathbf{x}$ and $\mathbf{y}$. This will guarantee that $d(\mathbf{x}, \mathbf{y})$ is bounded between *zero* and *one*.

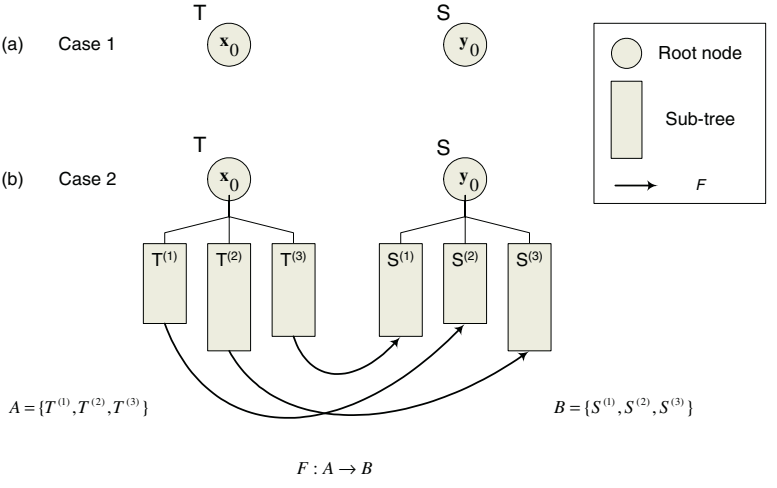

Fig. 2. Concavity-tree matching. Case 1 (a), case 2 (b).

2.2 Case 2, $N = M, N > 0, M > 0$

Here, the first step toward matching the two trees is to find a bijective mapping function $F : A \rightarrow B$ that relates each $T^{(k)}$ in A, $1 \leq k \leq N$, to its "closest" match in B. This mapping is based on the distance between the $\mathbf{x}_0$ and $\mathbf{y}_0$ as well as on other features such as the height and total number of nodes in the subtrees. The goal is to find the best possible match between the components of each pair in the mapping. This case is shown in Figure 2b where, as an example, $M = N = 3$. The distance between T and S is then defined as

$$\mathcal{D}(T, S) = a\, d(\mathbf{x}_0, \mathbf{y}_0) + b \left[\frac{1}{N} \sum_{k=1}^{N} \mathcal{D}(T^{(k)}, F(T^{(k)})) \right] \tag{3}$$

where a is the same constant as in case 1 and $0 < b < 1 - a$.

2.3 Case 3, $N = 0, M > 0$

In this case, T is a single (root) node and S is not. This is depicted in Figure 3a where S has, as an example, $M = 3$ subtrees. To match the two trees, we need to make the number of subtrees, N and M, equal. We therefore add, at level *one* of T, M dummy single-node subtrees $T^{(k)}$, $1 \leq k \leq M$, and we define

$$A' = \{T^{(1)}, \cdots, T^{(M)}\} \tag{4}$$

Each node in the added subtrees is labelled with a dummy feature vector $\mathbf{v}$. By definition, $d(\mathbf{v}, \mathbf{y}) = 1$.

A bijective function $G : A' \rightarrow B$ is then defined such that the concavity represented by $G(T^{(i)})$ has an area (relative to its parent) larger than or equal to the area of the one represented by $G(T^{(j)})$, for $i > j$.

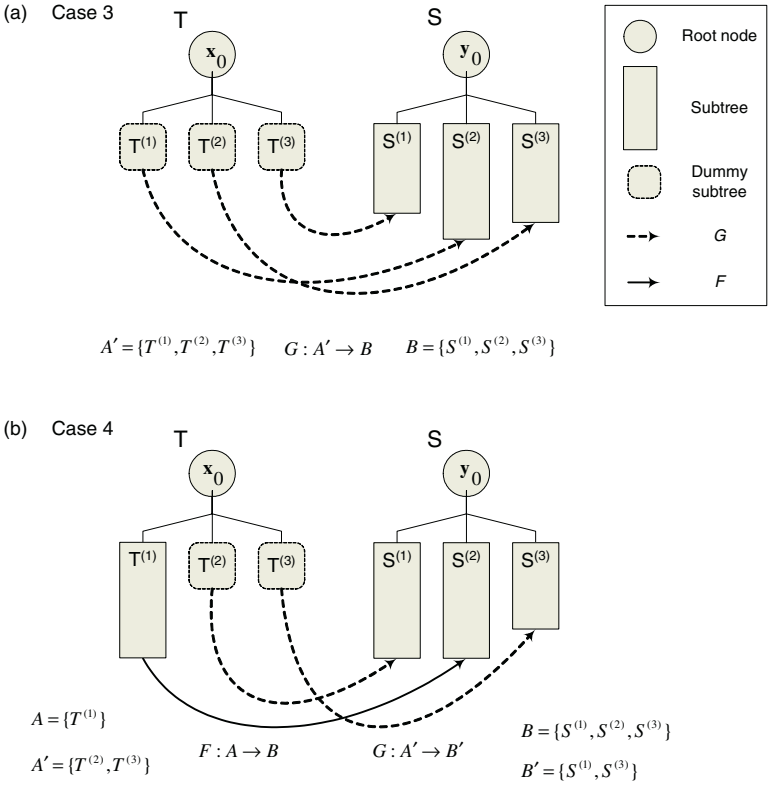

Fig. 3. Concavity-tree matching. Case 3 (a), case 4 (b).

The distance between T and S is given by

$$\mathcal{D}(T,S) = a\, d(\mathbf{x}_0, \mathbf{y}_0) + c \left[h \sum_{k=1}^{M} (1-h)^{k-1}\, \mathcal{D}(T^{(k)}, G(T^{(k)})) \right] \tag{5}$$

where a is the same constant as in cases 1 and 2, $c = 1 - a$, and $0 < h < 1$ (typically 0.5, discussed later.)

2.4 Case 4, $N > 0, M > 0, M > N$

Case 4 (Figure 3b) is the most general one and it combines the previous cases. We first find a function $F : A \to B$ that relates each $T^{(k)}$ in A, $1 \le k \le N$, to its "closest" match in B. F in this case is an injective function. That is, $M - N$ elements in B are not in the range of F. Let

$$B' = \{S^{(k)} : S^{(k)} \notin \text{range of } F\} \tag{6}$$

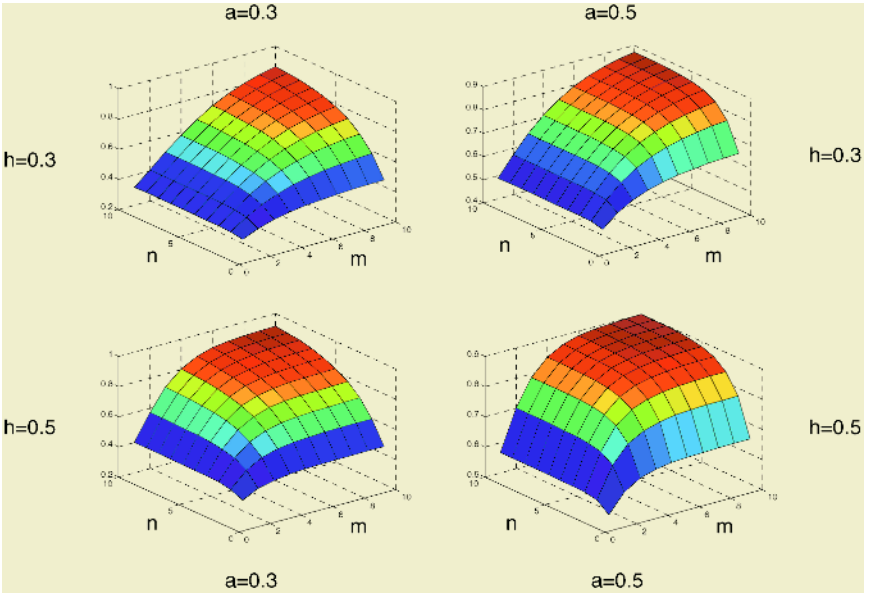

Fig. 4. The distance between a single-node tree and a tree with height n whose nodes (except leaves) have m children.

The next step is to add, at level *one* of T, $M-N$ dummy single-node subtrees $T^{(k)}$, $N < k \leq M$, and to define

$$A' = \{T^{(k)} : N < k \leq M\} \tag{7}$$

As in case 3, we define a bijective function $G : A' \rightarrow B'$ such that the concavity represented by $G(T^{(i)})$ has an area (relative to its parent) larger than or equal to the area of the one represented by $G(T^{(j)})$, for $i > j$. The distance between T and S is then given by

$$\mathcal{D}(T,S) = a\, d(\mathbf{x}_0, \mathbf{y}_0) + b \left[\frac{1}{N}\sum_{k=1}^{N} \mathcal{D}(T^{(k)}, F(T^{(k)}))\right] + c \left[h \sum_{k=N+1}^{M} (1-h)^{k-N-1}\, \mathcal{D}(T^{(k)}, G(T^{(k)}))\right] \tag{8}$$

where a is as defined in cases 1, 2, and 3, h is as defined in case 3, and b and c are real positive constants (weights) such that $a + b + c = 1$.

3 Discussion and Examples

In this section, we examine the role of the method parameters and their effects on the matching process. Four parameters are involved in the tree distance measure,

namely, a, b, c, and h. The first three parameters determine the weight given in the matching process to the overall shape attributes (a), the shapes of the underlying primitives (concavities) (b), and the cost of inserting new nodes in the smaller tree (c). Typically, a is first chosen such that it reflects the extent to which the similarity is based on the attributes in the root node, then b and c are chosen such that $a + b = 1$ (case 2), $a + c = 1$ (case 3), or $a + b + c = 1$ (case 4). In case 4, one possible variation is to choose b and c such that their ratio is $N/(N + M)$. If a and c are chosen to be much smaller than b, this will give more weight in the distance to the common part of the objects and hence allow for occluded matching.

Prior to recursively calling itself M times, each call to the matching method requires $1 + MN$ feature vector comparisons, resulting in a complexity of $O(l\,M^2)$, including any sorting operation. Consequently, a worst-case time complexity for the whole matching would be $O(l\,M^{H+1})$, where H is the height of the "higher" tree, and l, as defined before, is the dimensionality of the feature vector labelling the nodes of the trees. M here is the maximum branching factor in the two trees, *i.e.*, the number of children of the node with the highest number of children in any of the two trees. Clearly, this is an upper bound that is rarely reached (in the context of concavity trees) unless every node in both trees has M children. Given that the number of nodes m in a tree is at most $(M^{H+1}-1)/(M-1)$, which is $O(M^H)$, then the matching complexity, which was seen to be $O(l\,M^{H+1})$, can also be expressed as $O(l\,m\,M)$. Moreover, because they are independent, recursive calls to $\mathcal{D}(T, S)$ can be done in parallel, which significantly improves the matching speed.

Figure 4 shows the distance between two trees, one is a single-node tree and the other has a height of n and its non-leaf nodes each has m children. The distance between the feature vectors in the root of the two trees is assumed to be 0.8. The plots show that as n and m increase, *i.e.*, as the second tree grows, the distance approaches one. The rate of distance increase as a function of an increase of n and/or m can be controlled by adjusting the values of a and h.

$\mathcal{D}(T, S)$ is bounded by 1. What makes this characteristic suitable for shape matching using concavity trees is the following. If T_1, T_2, and T_3 are three trees such that T_1 has say 10 nodes, T_2 has 100 nodes, and T_3 has 110 nodes, $\mathcal{D}(T_1, T_2)$ is approximately the same as $\mathcal{D}(T_1, T_3)$ (slightly smaller). Which is desirable, since both T_2 and T_3 are much different from T_1. Another important characteristic is that as we move down the tree, each level gets less and less weight in the overall distance. So all the level is treated in a similar way. This is different from other tree matching algorithm where the weight of each node decreases monotonically both horizontally and vertically.

Figure 5 shows the pair-wise distances between 5 shapes. We note that the distance increases in each row from left to right and decreases in each column downward (as would be desired). Moreover, the closest non-identical pairs are in order: (4,5), (3,4), (2,3), and (1,2), which is also desired.

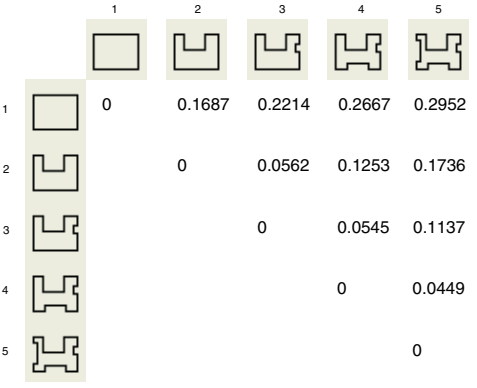

Fig. 5. The progression of the pair-wise distance between similar shapes.

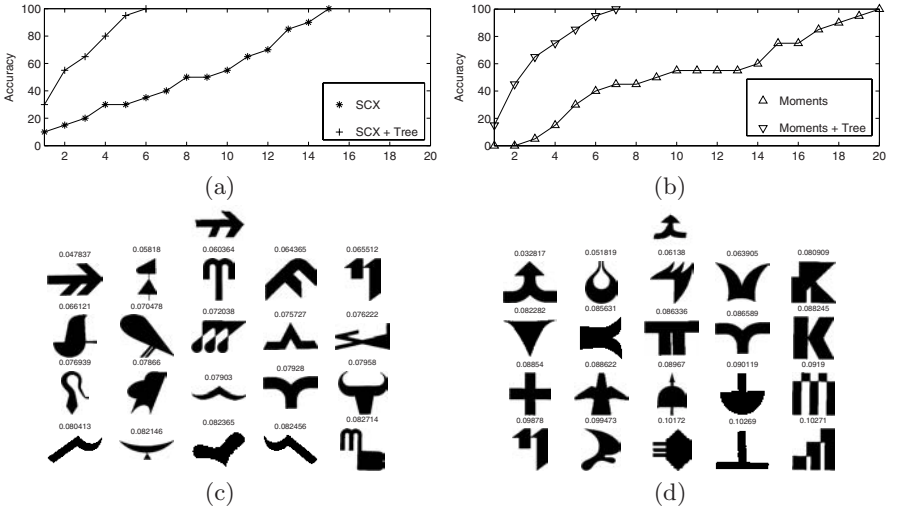

Fig. 6. Performance curves and query example.

4 Experimental Results

We use a database of 625 logo images to assess the new distance measure. We label the trees with two sets of attributes, namely SCX (Solidity, Eccentricity, and eXtent) [11] and moment invariants. We use 50 hand-sketched query images to test the retrieval performance with and without trees. In Figure 6 (a) and (b), the vertical axis is the percentage of the query images that were in the first k (horizontal axis) retrievals. In the plot for example, about 30% of the 50 query images returned the correct database image as the first hit (using SCX labelled trees). From the plots, it is clear that the performance of the SCX feature was improved by 20% when used in conjunction with concavity trees (the accuracy

actually tripled). The performance of moment invariants was similarly boosted by 15%. Figure 6 (c) and (d) show two sample queries that returned the target image in first position (using SCX labelled trees.) Using just SCX, the target images were originally in the 13^{th} and 21^{st} positions, respectively. The distance (to the query the image) is shown above each retrieved image.

5 Conclusion

This paper detailed a new recursive method for 2-D shape matching based on matching concavity trees. The method is suitable for concavity trees but can be applied to other attributed trees as well. Experiments using the proposed matching method show an increase of retrieval performance by at least 15%.

References

1. El Badawy, O., Kamel, M.: Shape retrieval using concavity trees. In: Proceedings of the International Conference on Pattern Recognition. (2004)
2. Sklansky, J.: Measuring concavity on a rectangular mosaic. IEEE Transactions on Computers **C-21** (1972) 1355–1364
3. Batchelor, B.: Hierarchical shape description based upon convex hulls of concavities. Journal of Cybernetics **10** (1980) 205–210
4. Borgefors, G., Sanniti di Baja, G.: Methods for hierarchical analysis of concavities. In: Proceedings of the International Conference on Pattern Recognition. Volume 3. (1992) 171–175
5. Borgefors, G., Sanniti di Baja, G.: Analyzing nonconvex 2D and 3D patterns. Computer Vision and Image Understanding **63** (1996) 145–157
6. Xu, J.: Hierarchical representation of 2-D shapes using convex polygons: A morphological approach. Pattern Recognition Letters **18** (1997) 1009–1017
7. Sanfeliu, A.: Matching tree structures. In Bunke, H., Sanfeliu, A., eds.: Syntactic and Structural Pattern Recognition, Theory and Applications. World Scientific (1990) 146–178
8. Torsello, A., Hancock, E.: Computing approximate tree edit distance using relaxation labeling. Pattern Recognition Letters **24** (2003) 1089–1097
9. Peura, M., Saltikoff, E., Syrjasuo, M.: Image analysis by means of attribute trees-remote sensing applications. In: Proceedings of the International Geoscience and Remote Sensing Symposium. (1999) 696–698
10. Sitaraman, K., Ranganathan, N., Ejnioui, A.: A vlsi architecture for object recognition using tree matching. In: Proceedings of the IEEE International Conference on Application-Specific Systems, Architectures and Processors. (2002) 325–334
11. El Badawy, O., Kamel, M.: Shape-based image retrieval applied to trademark images. International Journal of Image and Graphics **2** (2002) 375–393