Matching Shapes With Self-Intersections: Application to Leaf Classification

Farzin Mokhtarian and Sadegh Abbasi

***Abstract*—We address the problem of two-dimensional (2-D) shape representation and matching in presence of self-intersection for large image databases. This may occur when part of an object is hidden behind another part and results in a darker section in the gray level image of the object. The boundary contour of the object must include the boundary of this part which is entirely inside the outline of the object.**

The Curvature Scale Space (CSS) image of a shape is a multiscale organization of its inflection points as it is smoothed. The CSS-based shape representation method has been selected for MPEG-7 standardization. We study the effects of contour self-intersection on the Curvature Scale Space image. When there is no self-intersection, the CSS image contains several arch shape contours, each related to a concavity or a convexity of the shape. Self intersections create contours with minima as well as maxima in the CSS image. An efficient shape representation method has been introduced in this paper which describes a shape using the maxima as well as the minima of its CSS contours. This is a natural generalization of the conventional method which only includes the maxima of the CSS image contours. The conventional matching algorithm has also been modified to accommodate the new information about the minima. The method has been successfully used in a real world application to find, for an unknown leaf, similar classes from a database of classified leaf images representing different varieties of chrysanthemum. For many classes of leaves, self-intersection is inevitable during the scanning of the image.

Therefore the original contributions of this paper is the generalization of the Curvature Scale Space representation to the class of 2-D contours with self-intersection, and its application to the classification of Chrysanthemum leaves.

***Index Terms*—Curvature scale space, leaf classification, multiscale organization, self-intersections, shape matching.**

I. Introduction

SHAPE representation/description and matching is a central and challenging problem in Image Processing and Computer Vision which arises in many applications since shape is an inherent property of most objects. A large number of shape representation methods have been introduced in the literature [4], [16], [21], [23], [25], [26]. These include techniques based on the Fourier descriptors [5], [22], [30]. A number of shape representations have been proposed to recognize shapes even under affine transformation [3], [8], [10], [31]. Affine invariant scale space [24] and affine curvature [6], [17] have also been explored. A number of shape representation techniques are based on level-set methods [13]–[15] and volumetric diffusion [12]. These representations suffer from inefficiency and lack of robustness with respect to occlusion. Other techniques based on curve evolution [27], [28] are more suitable for applications other than shape representations.

The problem of self-intersection, however, has not been addressed properly. This may occur as a result of self-occlusion, when a part of an object is hidden behind another part. The resulting section of the image can be darker than its neighborhood. If segmented properly, the boundary contour of the object intersects itself and this must be considered in the related shape representation method.

The Curvature Scale Space image [19] of a shape is a multiscale organization of its inflection points as it is smoothed. For nonintersected shapes, the CSS image contains several arch shape contours, each related to a concavity or a convexity of the shape. The maxima of these contours have already been used for shape representation in shape similarity retrieval [18]. For self-intersected parts of a shape, the CSS contours are different. They include a minimum as well as a maximum which convey information about the size and location of the relevant intersected segment. Therefore, the CSS representation may still be used to represent a self-intersected contour. While a convexity or a concavity of the shape is represented by the maximum of its related arch shape contour in the CSS image, a self-intersected segment is represented by the locations of the maximum as well as minimum of the relevant contour of the CSS image.

The segmentation problem is always associated with the contour-based approach to shape representation. In the case of self-intersected shapes, segmentation is even more difficult [11] and may require user interaction.

We have used this representation in a real world application to find, for an unknown leaf, similar classes from a database of classified leaf images representing different varieties of chrysanthemum. For many classes of leaves, self intersection is inevitable during the scanning of the image. The task is to determine whether the unknown leaf belongs to one of the existing varieties or it represents a new variety. The system finds the most similar varieties to the input and allows the user to make the final decision. We have tested our method on a prototype database of 120 leaf images from 12 different varieties. The results have indicated a promising performance of the system.

The following is the organization of the remaining sections of this paper. Section II explains the real world problem of leaf classification which sparked this research in the first place. Section III explains the problem of self-intersection through an example. The segmentation of images to recover the object boundary is explained in Section IV. The method of computing

Manuscript received February 22, 2001; revised November 5, 2003. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Jean-Luc Dugelay.

The authors are with the Centre for Vision Speech and Signal Processing, University of Surrey, Guildford, Surrey GU2 7XH, U.K. (e-mail: F.Mokhtarian@surrey.ac.uk).

Digital Object Identifier 10.1109/TIP.2004.826126

curvature scale space image, and the algorithm for finding maxima and minima of CSS image are described in Section V. The matching algorithm which is used to measure the similarity between the two CSS images is explained in Section VI. A small number of global parameters are used in conjunction with CSS representation which are explained in Section VII. The experimental results are presented in Section VIII, and concluding remarks in Section IX.

II. Problem of Leaf Classification

In Britain, plant breeders who develop a new variety of plant are granted exclusive right to sell that variety for a period of time. One of the requirements imposed by current Plant Breeders Rights legislation is the distinctness of the new varieties. They should be different in at least one character from all existing varieties. The distinctness test is carried out by National Institute of Agricultural Botany (NIAB). There are 3000 registered varieties, each represented by ten leaf images, and NIAB receives about 300 new applications to be tested each year. The distinctness tests and leaf classification are currently carried out by NIAB experts based on a number of heuristic features. These features have not been well defined yet.

The main aim of our work has been to ease the process of test and classification by finding the most similar varieties to the input leaf image. We randomly selected a subset of the classified NIAB leaf images to create our prototype database. It consists of 120 leaf images from 12 different varieties of chrysanthemum. Every image contains one object on a simple background. The 8-bit grey-scale images have been scanned at 400 dpi.

A. Nature of Images

Five members of three classes of leaf images are shown in Fig. 1. Considering these images, one can easily appreciate that the problem of automatic classification of leaf images is a difficult task.

- *Overlaps* between some adjacent parts of leaves are sometimes unavoidable. They create major differences between the boundary contours of similar leaves.
- The between-class similarity is considerable, while the within-class similarity is not adequate. Therefore, the misclassification can happen frequently.
- The texture of leaves is rather similar, and texture features e.g the parameters derived from the co-occurance matrix are not useful to classify the leaf images. A complex and more sophisticated texture analysis may help, but it will be much more time consuming in comparison with the present system.
- Even in our prototype database the number of classes is notably large, while the number of samples in each class is quite small.

It is almost impossible to classify these images automatically. However, as the results of our experiments show, it is possible to find the most similar classes to an input image and help the user make the final decision.

III. Problem of Self-Intersection

Fig. 2 shows how an intersection occurs. The actual boundary of the object of Fig. 2(a) has been partly hidden by some parts of the object. In order to extract the boundary of the object, one may ignore the hidden part and extract the outline of the object as presented in Fig. 2(b), where apparently some information is missing in a tradeoff which reduces the complexity of segmentation. A simple thresholding and a contour tracing algorithm extracts the boundary of the object. The actual boundary, as presented in Fig. 2(c) includes three points of self-intersection which need to be recovered interactively. During the process of contour tracing, the user should help the program to follow the contours inside the object rather than the boundary of it.

This contour must finally be represented by appropriate shape descriptors. The CSS image of the contour provides a good source of information which has been used to describe the shape. In the following section we briefly explain how the CSS image of a contour is constructed and how the useful information is extracted from this image.

IV. Image Segmentation

The aim of this stage is to recover the boundary of objects, taking into account the self-intersection parts. Using a gray level histogram, we find the best threshold and then separate the object from the background automatically. We then use a simple contour tracing method to extract the boundary of object (see Fig. 3).

If there were no overlaps, this method would extract the actual boundary of the object. To extract the boundary of an object in presence of an overlap, we employ an interactive method. The gray level image is first segmented into three regions, namely background, overlapped parts of the object, and other parts of the object. The contour tracing method starts from an arbitrary point of the boundary, indicated by the user. The system traces the boundary between the background and the object until it reaches an intersection point, indicated by the user. The system then changes the tracing direction and goes inside the object to trace the boundary between the darker and the lighter parts of the object. This process continues until the whole boundary of the object including the internal intersected part is extracted. The user help is needed at all stages as the intersected parts do not have a regular pattern. For example they may create one or two loops, and they may or may not have border with the background of the image.

V. Curvature Scale Space Representation

The curvature of a curve is defined as the derivative of the tangent vector to the curve and can be expressed as

$$\kappa(u) = \frac{\dot{x}(u)\ddot{y}(u) - \ddot{x}(u)\dot{y}(u)}{(\dot{x}^2(u) + \dot{y}^2(u))^{\frac{3}{2}}}. \tag{1}$$

There are several approaches in calculating the curvature of a digital curve [29]. We use the idea of *curve evolution* which basically studies shape properties while deforming in time. A certain kind of evolution can be achieved by Gaussian smoothing to compute curvature at varying levels of detail. If $g(u, \sigma)$ is a

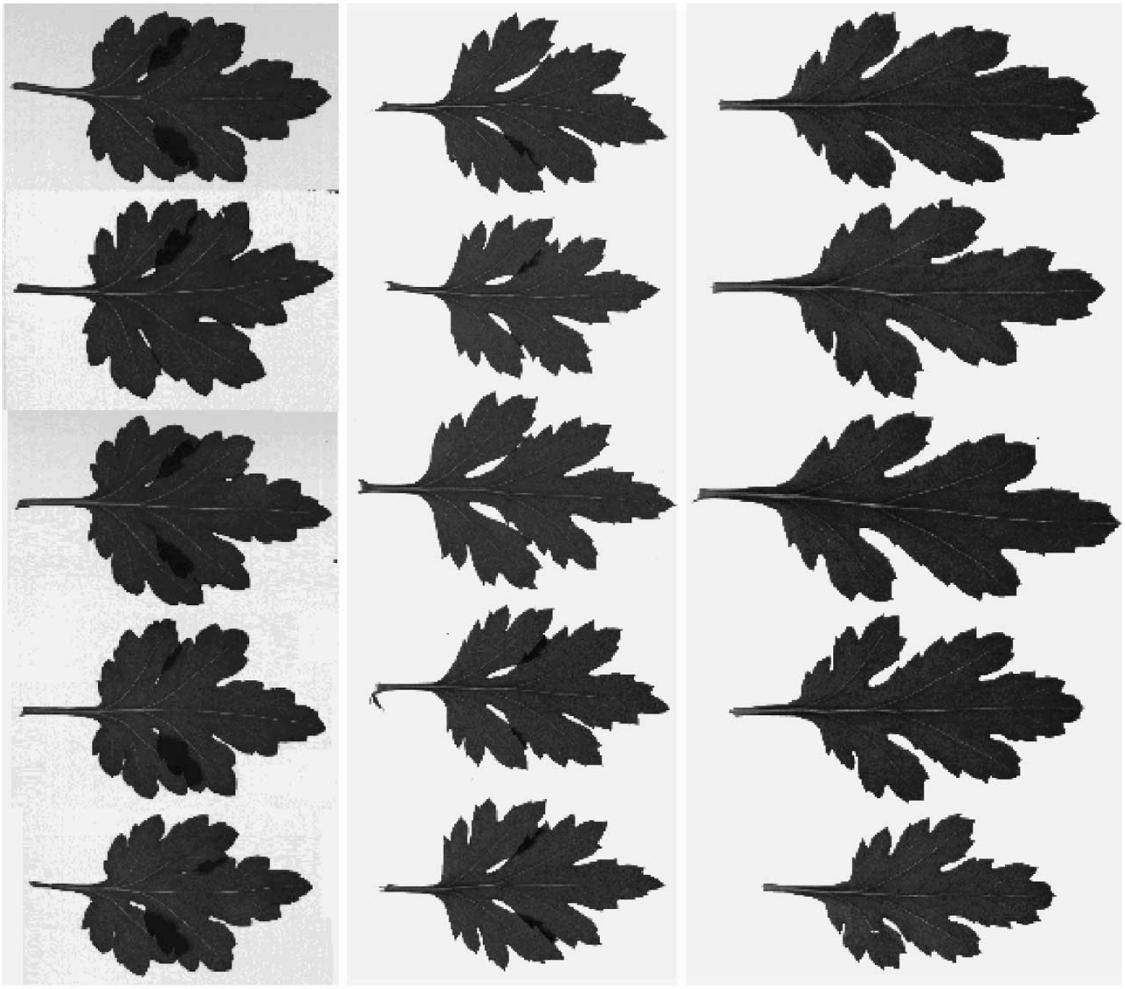


Fig. 1. Three classes of images. Due to intraclass similarity and interclass dissimilarity, misclassification is likely to happen.

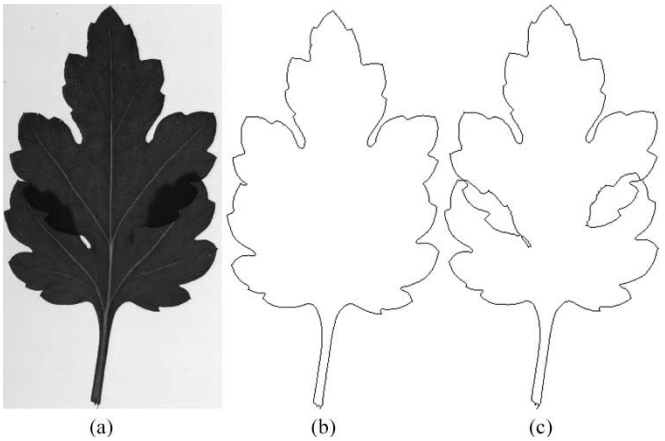


Fig. 2. An example of self-intersection. (a) Gray level image. (b) The boundary of object without considering self-intersection. (c) The actual boundary of the object.

1-D Gaussian kernel of width σ, then $X(u,\sigma)$ and $Y(u,\sigma)$ represent the components of *evolved* curve

$$X(u,\sigma) = x(u) * g(u,\sigma) \quad Y(u,\sigma) = y(u) * g(u,\sigma).$$

According to the properties of convolution, the derivatives of every component can be calculated easily

$$X_u(u,\sigma) = x(u) * g_u(u,\sigma) \quad X_{uu}(u,\sigma) = x(u) * g_{uu}(u,\sigma)$$

and we will have similar formulas for $Y_u(u,\sigma)$ and $Y_{uu}(u,\sigma)$. Since the exact forms of $g_u(u,\sigma)$ and $g_{uu}(u,\sigma)$ are known, the curvature of an evolved digital curve can be computed easily

$$\kappa(u,\sigma) = \frac{X_u(u,\sigma)Y_{uu}(u,\sigma) - X_{uu}(u,\sigma)Y_u(u,\sigma)}{(X_u(u,\sigma)^2 + Y_u(u,\sigma)^2)^{\frac{3}{2}}}. \quad (2)$$

As σ increases, the shape of Γ_σ changes. This process of generating ordered sequences of curves is referred to as the evolution of Γ.

Following the preprocessing stage, every object is represented by the x and y coordinates of its boundary points. To obtain a representation based on normalized arc length, we re-sample the boundary and represent it by 200 equally distant points. Considering the resampled curve as Γ, we can determine the locations of curvature zero crossings on Γ_σ, using the above mentioned formula. We start the process with $\sigma = 1$ and increase it by 0.1 at each level. As σ increases, Γ_σ shrinks and

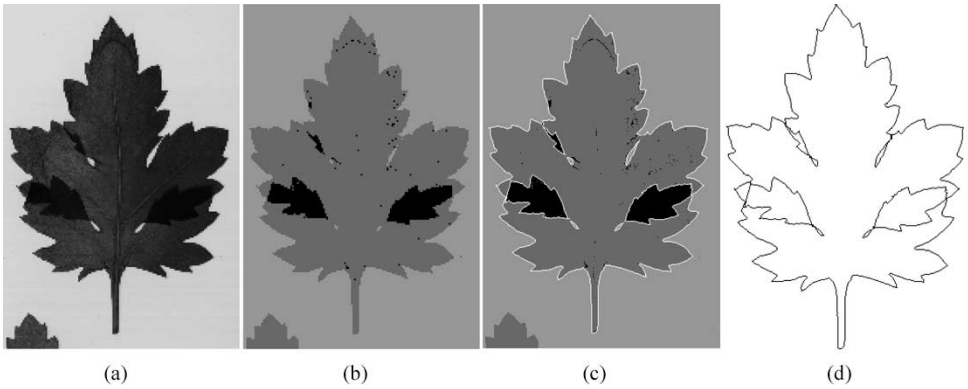


Fig. 3. Image segmentation: (a) original image, (b) multilevel thresholding, (c) interactive contour tracing, and (d) object boundary.

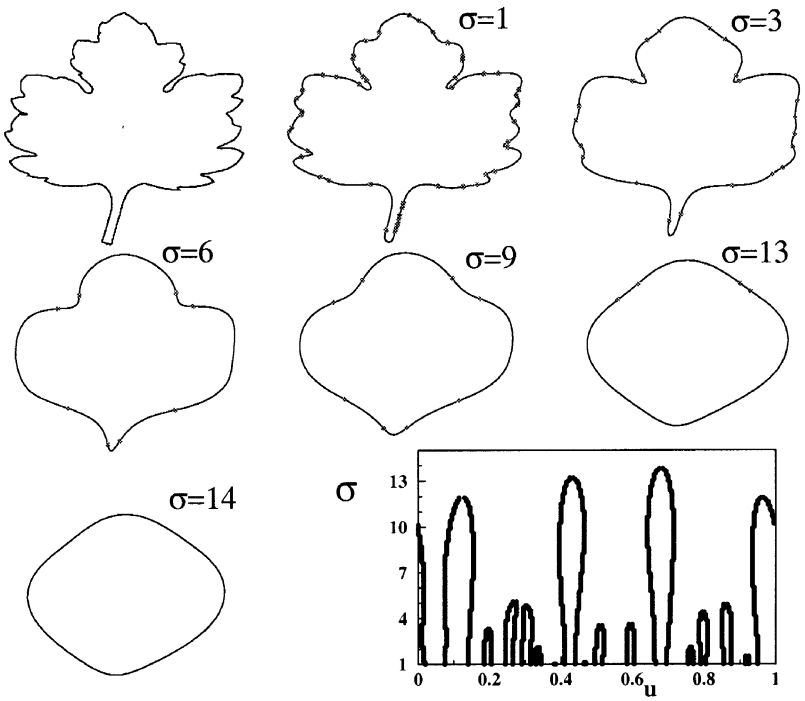


Fig. 4. Shrinkage and smoothing of the curve and decreasing of the number of curvature zero crossings during the evolution.

becomes smoother, and the number of curvature zero crossing points on it decreases. Finally, when σ is sufficiently high, Γ_σ will be a convex curve with no curvature zero crossings. The process has been shown in Fig. 4. The original curve is represented in top left and the evolution has been shown through several values of σ.

If we determine the locations of curvature zero crossings of every Γ_σ during the evolution, we can display the resulting points in (u, σ) plane, where u is the normalized arc length and σ is the width of the Gaussian kernel. The result of this process can be represented as a binary image called CSS image of the curve (see bottom right of Fig. 4). The intersection of every horizontal line with the contours in this image indicates the locations of curvature zero crossings on the corresponding evolved curve Γ_σ. For example, by drawing a horizontal line at $\sigma = 9.0$, it is observed that there should be 8 zero crossing points on Γ_9. This fact is confirmed by the smooth curve with $\sigma = 9.0$ in the same figure.

As shown in Fig. 4, the curvature zero crossings appear in pairs. Each pair is related to a concavity (or sometimes a convexity) on the boundary. As σ increases, the concavities are gradually filled and the related pair of zero crossings approach each other. The result on the CSS image will be two branches of a contour. Each branch conveys information on the locations of one of the zero crossings during the process. For a particular σ the concavity is totally filled and its pair of curvature zero crossings join each other. At this stage a contour of CSS image reaches its maximum. The $\sigma\_coordinate$ of the maximum is the relevant σ and the $u\_coordinate$ is the location of joined zero crossing pair.

If a local deformation occurs on the boundary, the shape of the contours on CSS image may change, but the location of the relevant maximum does not change dramatically. This is the main reason for selecting these points as our shape descriptors. Therefore the final CSS representation consists of the locations of the maxima of contours in the CSS image.

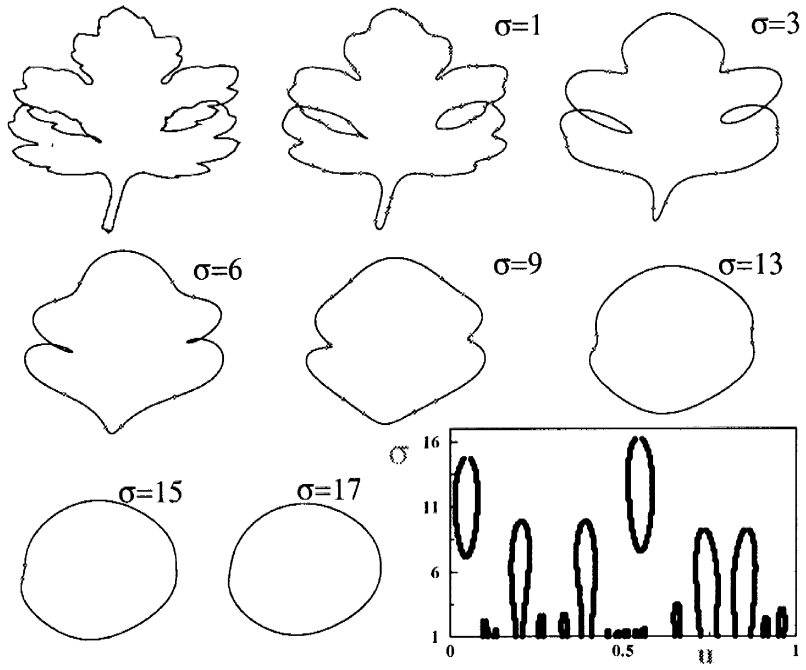


Fig. 5. Evolution for a shape with self-intersection.

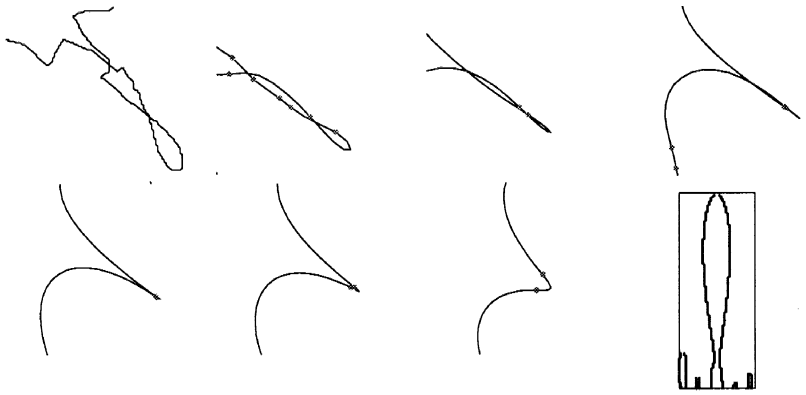


Fig. 6. Sometimes self-intersection does not create a minimum in the CSS image.

A. CSS Image of Self-Crossed Boundaries

An example of the evolution of such a shape is represented in Fig. 5. The original shape includes self-intersections and is seen in top left. For $\sigma = 1$, there are some inflection points inside the intersected loops due to small concavities in these areas which disappear in very early stages and before $\sigma = 3$. At this stage, there are no inflection points in these regions. However, the intersected loop gradually vanishes and a concavity appears in its place. This concavity, in turn, creates a contour in the CSS image which obviously does not start from $\sigma = 1$. The two branches of this contour are created from the moment that the intersected loop vanishes and the concavity is born. A minimum is then created in the CSS image at the relevant σ. It is obvious that the height of this minimum is proportional to the size of the intersected loop.

The location of a CSS minimum in the CSS image conveys information about the self-intersected loop of the shape. The horizontal coordinate of a minimum reflects the position of the loop on the shape, while the vertical coordinate shows the size of the loop.

We discovered that while we might expect a minimum to appear in the CSS image for every self-intersected region, this is not always the case in practice. In fact, if the size of the self-intersected loop is small, the minimum is expected to appear in early stages when the inflection points inside the loop have just disappeared. As a result, the maximum of the contour created by those small ripples joins the minimum of the contour created after the vanishing of the loop. This can clearly be seen in Fig. 6. The original segment of the shape is seen in top left. As σ increases, all inflection points in the segment disappear except for one pair. When the loop disappears, this pair disappears but a new pair of inflection points is born. The result has been shown in the corresponding contour of the CSS image in the lower part of the figure. The thin part of the contour is related to the moment when this event happens.

It should be noted that even if a minimum is created in such situations, due to its small height, its effect is not considerable.

B. Extracting Maxima and Minima of Curvature Scale Space Contours

We represent every image in the database with the locations of its CSS contour extrema. For example, in Fig. 5 there are six maxima and two minima. Therefore, the shape will be represented by 8 pairs of integer numbers. The locations of extrema are not readily available and must be extracted from the image. The CSS contours are usually connected everywhere except sometimes in a neighborhood of their maxima as seen in Fig. 7. We find the peaks of both branches of a contour in the CSS image and consider the middle point of the line segment joining the pair as a maximum of the CSS image. Starting from the top, each row of the CSS image is scanned for a black pixel. When found, the search continues in the same row to find another one in the neighborhood. After finding a maximum of a contour, both branches of the contour are marked and at the same time, search for a possible minimum of the contour begins.

An explanation of our algorithm follows.

1) Start with scanning the second row. If a zero-crossing (black) point is found examine its neighboring points. If there is no zero-crossing neighbor at the row just above and there is just one zero-crossing neighbor at the following row, go to step 3, otherwise go to step 2.
2) Scan the remaining points of the current row, and start scanning the next row if it is not the last one to be scanned. If a candidate is found go to step 3, and if this is the last row to be scanned, stop.
3) Scan the same row to find the same zero-crossing as described in step 1, in a reasonable distance. If the next candidate is not found, mark the first one and go to step 2. If it is found, do the following:
    - Consider the middle point of the line segment joining the pair as a maximum.
    - Mark (delete) all zero-crossings at both branches of the corresponding contour whose maximum has just been found.
    - When marking a branch, look for a minimum. A minimum is a point where there is no black point just below it.
    - If a minimum is found, go to step 2 immediately. Otherwise mark the contour down to the last row to be scanned and then go to step 2.

Note that usually the bottom few rows of CSS image represent some information about the existing noise on the actual image contour, so the last row to be scanned in step 2 has 1/6 the height of the CSS image. Also note that if a candidate in step 2 is in first few columns, the point paired with it may exist in the last few columns of the same row and vice versa. In this case the search for next candidate must include the relevant interval.

VI. Curvature Scale Space Matching

When there is no self-intersection in the boundary of the object, the CSS representation includes only the maxima of the CSS image contours. The algorithm used for comparing two sets of maxima, one from the input (also called *image*) and the other from one of the models, has been described in [1]. In this section, we first explain this algorithm briefly. Then we extend it to match the minima as well as the maxima.

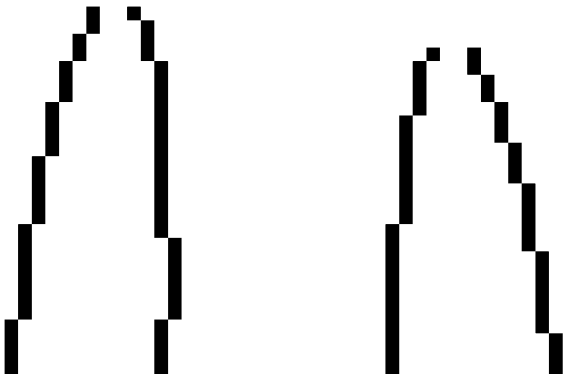


Fig. 7. To find a CSS maximum, we start from the top and scan each row of the CSS image, looking for a pair of black points with a small gap.

The matching algorithm first finds any possible changes in orientation of the underlying contour as well as changes in the starting point for contour parametrization which may have occurred in one of the two shapes. A circular shift is then applied to one of the image maxima to compensate the effects of change in orientation or starting point. The summation of the Euclidean distances between the relevant pairs of maxima is then defined to be the matching value between the two CSS images.

The following is a condensed version of the algorithm which includes the basic concepts.

- Apply a circular shift to all image maxima so that the $u\_coordinates$ of the largest maxima, one from the image and the other from the model become identical.
- Starting from the second largest maximum of the image, determine the nearest maximum of the model to each maximum of the image.
- Consider the cost of the match as the summation of the Euclidean distances between the corresponding maxima.
- If the number of the image and the model maxima differs, add the σ coordinates of the unmatched maxima to the matching cost. In other words, if a CSS image contains a maximum but the other CSS image does not contain the corresponding maximum, the σ coordinate of the unmatched maximum is added to the matching cost.

A. Matching Minima

A straightforward approach in taking into account the minima is to match the two sets of minima, one from the image and the other from the model exactly the same as the way we do with two sets of maxima. In this approach, the two sets of maxima are first matched and the corresponding matching value is found. Then the two sets of minima are matched and the resulting matching value is added to the latter to produce the final matching value between the two CSS image. In this approach, the shift parameter which is used to compensate the effect of change in orientation, may be different for the two parts of matching. In another approach, one may match the maxima and obtain the best shift parameter which is then used for matching the minima. The third approach is to match maxima and minima simultaneously.

In other words, when two maxima are matched, their possible corresponding minima are also matched and the Euclidean distance of the minima is added to the Euclidean distance of the maxima. We examined these approaches and found out that the first approach leads to the best results which are presented in the following section.

VII. Global Parameters

In order to increase the efficiency of the system, a number of global parameters are also computed and stored in a shape record in conjunction with the CSS extrema. These parameters are used for indexing and narrowing down the search space before applying the CSS matching. In this section we explain these parameters briefly which include eccentricity, circularity and aspect ratio of the CSS image. These parameters are scale and orientation invariant.

Eccentricity: Eccentricity has been widely used as a shape feature [7], [20]. It is a region based parameter and illustrates how the region points are scattered around the centre of the region, *centroid*.

The central moments of a region is defined as

$$\mu_{p,q} = \sum_x \sum_y (x - \overline{x})^p (y - \overline{y})^q$$

where x and y are coordinates of the region points and $(\overline{x}, \overline{y})$ is called the *centre of gravity* or *centroid*.

The *principal moments* of a region are eigenvalues of the matrix

$$\begin{bmatrix} \mu_{2,0} & \mu_{1,1} \\ \mu_{1,1} & \mu_{0,2} \end{bmatrix}$$

and the eccentricity of the region is

$$eccentricity = \sqrt{\frac{\lambda_{max}}{\lambda_{min}}}$$

where λ_{max} and λ_{min} are the eigenvalues of the matrix. This feature depends solely on the shape, not on size and orientation.

Aspect ratio of the CSS image: Every CSS contour corresponds to a concavity or a convexity in the image boundary. If the width of the CSS image, ie. the number of samples used to represent the image contour is the same, the longer and deeper a concavity, the taller the corresponding CSS contour. This means that if the image contour consists of long and/or deep concavities, the height of its CSS image is large.

The *aspect ratio* of the CSS image represents the height of the largest scale maximum of the CSS image which in turn reflects the size of the major concavity of the image boundary. This parameter also depends only on the shape, not on size and orientation.

Circularity: Circularity is the ratio of perimeter squared to the area. It can be used to distinguish between rippled boundaries and smoothed ones, when the overall shapes are rather the same.

VIII. Experiments and Results

In order to test our method, we used a database of classified leaf images representing different varieties of chrysanthemum. For many classes of leaves, self-intersection is inevitable during the scanning of the image.

We tested our method on a prototype database of 120 leaf images from 12 different varieties, both with and without considering the self-intersection. The task was to find out whether an unknown leaf belongs to one of the existing varieties or it represents a new variety. The system found the most similar varieties to the input and allows the user to make the final decision. The results indicated a promising performance of the new approach and its superiority over the conventional method.

To reject the dissimilar images based on the global parameters, we first calculate α_e, α_c and α_a as follows:

$$\alpha_e = \frac{|e_i - e_m|}{max(e_i, e_m)} \quad \alpha_c = \frac{|c_i - c_m|}{max(c_i, c_m)} \quad \alpha_a = \frac{|a_i - a_m|}{max(a_i, a_m)}$$

where e and c represent the eccentricity and circularity of the boundary and a represents the aspect ratio of the CSS image, while i and m stand for image and model respectively.

According to their definition, α_e, α_c and α_a are between zero and one. We need to choose a threshold for each of these parameters so that if one of them is above the relevant threshold, the corresponding model is rejected. For our system we chose these threshold values

$$\alpha_{et} = 0.125 \quad \alpha_{ct} = 0.25 \quad \alpha_{at} = 0.30.$$

Note that the performance of the system is not sensitive to small changes of these values [2].

The CSS matching is then applied to the surviving models to find the most similar objects to the input query and also make a decision about the class of the the input leaf.

An analysis of system computational complexity follows. Each of the values α_e, α_c and α_a is computed for each shape in the database. CSS matching is then applied to each surviving model in the database. Therefore if the size of the database is n, the computational complexity of the system is $O(n)$.

A. Results

We tested the proposed method on a database of 120 leaf images from 12 different chrysanthemum varieties, some including self-intersections. Each image consisted of just one object on a uniform background. The system software was developed using the C language under Unix operating system. The response rate of the system was much less than one second for each user query.

To evaluate the method, we considered every image in the database as an input and in each case, asked the system to identify the variety of the input, based on the first k similar images. Obviously, the first output of the system is identical to the input, but the system does not consider it in classification. In fact, for each sample we first pull it out of the database and classify it based on the remaining classified samples. The best varieties are then selected based on the number of their samples in the best k similar samples. The output of the system is the name of

TABLE I
RESULTS OF EVALUATION FOR DIFFERENT VALUES OF k, INITIAL METHOD

	$k=5$	$k=6$	$k=7$	$k=8$	$k=9$	$k=10$
1	75.8%	77.5%	78.3%	75.0%	72.5%	70.0%
<= 2	90.0%	88.3%	86.6%	87.5%	88.3%	89.1%
<= 3	91.6%	94.1%	94%	94.1%	95.8%	93.2%

TABLE II
RESULTS OF EVALUATION FOR DIFFERENT VALUES OF k, IMPROVED METHOD

	$k=5$	$k=6$	$k=7$	$k=8$	$k=9$	$k=10$
1	81%	85%	81.7%	79.2%	80%	78.3%
<= 2	94.3%	94.2%	93.4%	92.5%	91.7%	92.5%
<= 3	97.6%	97.5%	98.4%	97.5%	95.8%	95.8%

the first 3 classes. Note that this procedure corresponds to measuring the *precision* of the system using terminology from the field of information retrieval [9].

We first used a simple automatic segmentation to recover the outline of the objects without considering the self-intersection (see Fig. 2). The resulting representations included only the maxima of the CSS images. We also used some global parameters to reject dissimilar candidates prior to the CSS matching [2]. The results for different values of k, the number of observed outputs for each query, has been presented in Table I. As mentioned before, in response to a query, the system returns the top 3 classes which are most similar to the input. The first row of Table I shows the success rate of the system to identify the correct class as its first choice is 75.8% if the judgement is based on the first 5 outputs of the system. This figure is 77.5% for $k=6$ and so on. The second row shows the success rate of the system to identify the correct class as its first or second choices, and so on for the third row.

As this table shows, for this particular database, one may obtain good results. However, as shown in Table II, even better results may be achieved by including the self-intersections in the process. It is interesting that the performance of the system is not sensitive to the value of k, specially when we consider the last row of this table. Overall, the superiority of the improved method over the conventional one is seen in these two tables. Note that the matching cost might increase for a given pair of CSS images when minima are added to the matching process. This would indicate that those CSS images are more different than indicated by matching maxima only.

IX. CONCLUSIONS

The problem of self-intersection was discussed in this paper. A shape representation method which can properly represent such shapes was introduced and explored. The Curvature Scale Space (CSS) image of a planar curve normally consists of several arch shape contours each related to concavity or a convexity of the curve. We observed that the shape of CSS contours change when the curve includes self-intersection. The arch shape of the contours is converted to a vertical ellipse. While the arch shape contours are represented by their maxima, the new contours of the CSS image include a minimum as well as the maximum.

In conventional form of CSS representation, a curve is represented by the locations of its CSS maxima. A new representation was introduced here which uses the minima as well of maxima of the CSS image. A method to extract extrema of the CSS image, as well as a matching algorithm to compare two sets of extrema and assign a matching value as the measure of similarity between the two curve were also introduced. The method was tested on a prototype database of 120 leaf images from 12 different varieties of chrysanthemum with promising results. The performance of the method was also compared to the performance of the conventional method.

REFERENCES

[1] S. Abbasi and F. Mokhtarian, "Affine-similar shape retrieval: application to multiview 3-D object recognition," *IEEE Trans. Image Processing*, vol. 10, pp. 131–139, Jan. 2001.

[2] S. Abbasi, F. Mokhtarian, and J. Kittler, "Reliable classification of chrysanthemum leaves through curvature scale space," in *Proc. Scale-Space'97 Conf.*, Utrecht, The Netherlands, July 1997, pp. 284–295.

[3] K. Arbter, W. E. Snyder, H. Burkhardt, and G. Hirzinger, "Applications of affine-invariant Fourier descriptors to recognition of 3-d objects," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 12, pp. 640–646, July 1990.

[4] A. Del Bimho, P. Pala, and S. Santini, "Image retrieval by elastic matching of shapes and image patterns," in *Proc. 1996 Int. Conf. Multimedia Computing and Systems*, Hiroshima, Japan, June 1996, pp. 215–218.

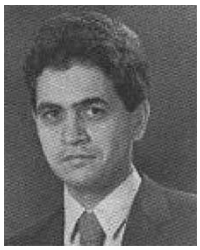
[5] T. D. Bui, G. Y. Chen, and L. Feng, "An orthonormal-shell-fourier descriptor for rapid matching of patterns in image database," *Int. J. Pattern Recognit. Artif. Intell.*, vol. 15, no. 8, pp. 1213–1229, 2001.

[6] D. Cyganski, T. A. Cott, J. A. Orr, and R. J. Dodson, "Development, implementation, testing, and application of an affine transform invariant curvature function," in *Proc. 1st Int. Conf. Computer Vision*, London, U.K., June 1987, pp. 496–500.

[7] P. Eggleston, "Constraint-based feature indexing and retrieval for image databases," *Proc. SPIE*, vol. 1819, pp. 27–39, 1992.

[8] J. Flusser and T. Suk, "Pattern recognition by affine moment invariants," *Pattern Recognit.*, vol. 26, no. 1, pp. 167–174, Jan. 1993.

[9] B. Furht, S. W. Smoliar, and H. Zhang, *Video and Image Processing in MultiMedia Systems*. Norwell, MA: Kluwer, 1996.
[10] Z. Huang and F. S. Cohen, "Affine-invariant B-spline moment for curve matching," in *Proc. IEEE Computer Society Conf. Computer Vision and Pattern Recognition*, 1994, pp. 490–495.
[11] N. Katzir, M. Lindenbaum, and M. Porat, "Curve segmentation under partial occlusion," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 16, pp. 513–519, May 1994.
[12] J. J. Koenderink, *Solid Shape*. Cambridge, MA: MIT Press, 1990.
[13] R. Malladi and J. A. Sethian, "Level set and fast marching methods in image processing and computer vision," in *Proc. IEEE Int. Conf. Computer Vision*, vol. 1, 1996, pp. 489–492.
[14] ——, "Unified approach to noise removal, image enhancement, and shape recovery," *IEEE Trans. Image Processing*, vol. 5, pp. 1554–1568, Nov. 1996.
[15] ——, "Real-time algorithm for medical shape recovery," in *Proc. IEEE Int. Conf. Computer Vision*, 1998, pp. 304–310.
[16] R. Mehrotra and J. E. Gary, "Feature-based retrieval of similar shapes," in *Proc. 9th Int. Conf. Data Engineering*, Vienna, Austria, Apr. 1993, pp. 108–115.
[17] F. Mokhtarian and S. Abbasi, "Affine curvature scale space with affine length parametrization," *Pattern Anal. Applicat.*, vol. I5, no. 2, pp. 143–158, 2001.
[18] ——, "Shape similarity retrieval under affine transforms," *Pattern Recognit. (Special Issue on Shape Representation and Similarity for Image Databases)*, vol. 35, no. 1, pp. 31–41, 2002.
[19] F. Mokhtarian and A. K. Mackworth, "A theory of multi-scale, curvature-based shape representation for planar curves," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 14, pp. 789–805, Aug. 1992.
[20] W. Niblack, R. Barber, W. Equitz, M. D. Flickner, E. H. Glasman, D. Petkovic, P. Yanker, C. Faloutsos, and G. Taubin, "The QBIC project: Querying images by content using color texture and shape," *Proc. SPIE*, vol. 1908, pp. 173–187, 1993.
[21] W. Niblack and J. Yin, "Pseudo-distance measure for 2d shapes based on turning angle," in *Proc. IEEE Int. Conf. Image Processing*, vol. 3, Washington, DC, USA, Oct. 1995, pp. 352–355.
[22] Z. Ping, Y. Sheng, S. Deschenes, and H. Arsenault, "Fourier-Mellin descriptor and interpolated feature space trajectories for 3-D object recognition," *Opt. Eng.*, vol. 39, no. 5, pp. 1260–1266, 2000.
[23] E. Saber and A. M. Tekalp, "Image query-by-example using region-based shape matching," *Proc. SPIE*, vol. 2666, pp. 200–211, 1996.
[24] G. Sapiro and A. Tannenbaum, "Affine invariant scale space," *Int. J. Comput. Vis.*, vol. 11, no. 1, pp. 25–44, 1993.
[25] S. Sclaroff, "Deformable prototypes for encoding shape categories in image databases," *Pattern Recognit.*, vol. 30, no. 4, pp. 627–641, Apr. 1997.
[26] S. Sclaroff and A. P. Pentland, "Modal matching for corresponding and recognition," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 17, pp. 545–561, June 1995.
[27] K. Siddiqi, B. B. Kimia, and C. W. Shu, "Geometric shock-capturing eno schemes for subpixel interpolation, computation and curve evolution," *Graph. Models Image Process.*, vol. 59, no. 5, pp. 278–301, Sept. 1997.
[28] K. Siddiqi, B. B. Kimia, A. Tannenbaum, and S. W. Zucker, "Shapes, shocks and wiggles," *Image Vis. Comput.*, vol. 17, no. 5, pp. 365–373, 1999.
[29] D. Tsai and M. Chen, "Curve fitting approach for tangent angle and curvature measurement," *Pattern Recognit.*, vol. 27, no. 5, pp. 699–711, 1994.
[30] A. Wimmer, G. S. Ruppert, O. Sidla, H. Konrad, and F. Gretzmacher, "Fft-descriptors for shape recognition of military vehicles," *Proc. SPIE*, vol. 4029, pp. 81–87, 2000.
[31] A. Zhao and J. Chen, "Affine curve moment invariants for shape recognition," *Pattern Recognit.*, vol. 30, no. 6, pp. 895–901, 1997.



Farzin Mokhtarian received the BES degree in math sciences from The Johns Hopkins University, Baltimore, MD, in 1982, and the M.Sc. and Ph.D. degrees in computer science from the University of British Columbia, Vancouver, BC, Canada, in 1984 and 1990, respectively.

From 1984 to 1985, he was with the Schlumberger-Doll Research Lab, Ridgefield, CT. From 1991 to 1993, he was a Research Scientist at NTT Basic Research Labs, Tokyo, Japan. In 1994, he joined the Center for Vision, Speech, and Signal Processing in the Department of Electronic and Electrical Engineering at the University of Surrey, Guildford, U.K., as a Lecturer. In 2000, he was promoted to a Senior Lecturer. His research interests include 2-D and 3-D multiscale shape representation and recognition through differential invariants, image database retrieval using shape features, and other vision tasks requiring shape analysis. His Curvature Scale Space shape descriptor has been selected for MPEG-7 standardization. He is the first author of the book *Curvature Scale Space Representation: Theory, Applications, and MPEG-7 Standardization* (Norwell, MA: Kluwer, 2003).

Sadegh Abbasi received the B.Sc. and M.Sc. degrees in telecommunications systems from the University of Tehran, Tehran, Iran. He received the Ph.D. degree in 1999 from the Centre for Vision, Speech and Signal Processing (CVSSP), University of Surrey, Guildford, U.K., working on multimedia similarity retrieval based on shape content.

He was with Iran Telecommunication Research Centre for 2 years. He then worked for 3 years at CVSSP as a Research Fellow carrying out research on shape similarity retrieval. He is now with Phillips R&D Lab, Southampton, U.K.