[7] S. Alliney and C. Morandi, "Digital image registration using projections," *IEEE Trans. Patt. Anal. Machine Intell.*, vol. PAMI-8, pp. 222–233, 1986.
[8] J. Weng, T. S. Huang, and N. Ahuja, "3-D motion estimation, understanding and prediction from noisy image sequences," *IEEE Trans. Patt. Anal. Machine Intell.*, vol. PAMI-9, pp. 370–388, 1987.
[9] A. Goshtasby, "Template-matching in rotated images," *IEEE Trans. Patt. Anal. Machine Intell.*, vol. PAMI-7, pp. 338–344, 1985.
[10] E. DeCastro and C. Morandi, "Registration of translated and rotated images using finite Fourier transforms," *IEEE Trans. Patt. Anal. Machine Intell.*, vol. PAMI-9, pp. 700–703, 1987.
[11] S. Alliney, "Linear operators and discrete transforms," in *Signal Processing II: Theories and Applications* (H. W. Schuessler, Ed.). Amsterdam: North-Holland, 1983, pp. 9–12.
[12] J. Wilkinson, *Rounding Errors in Algebraic Processes*. Englewood Cliffs, NJ: Prentice-Hall, 1963.
[13] I. S. Gradshteyn and I. M. Ryzhik, *Tables of Integrals, Series and Products*. New York: Academic, 1980.
[14] G. H. Golub and C. F. Van Loan, *Matrix Computations*. Baltimore, MD: Johns Hopkins University Press, 1989.
[15] A. Papoulis, *Systems and Transforms with Application in Optics*. New York: McGraw-Hill, 1968.

Computing 2-D Min, Median, and Max Filters

Joseph Gil and Michael Werman

***Abstract*— Fast algorithms to compute min, median, max, or any other order statistic filter transforms are described. The algorithms take constant time per pixel to compute min or max filters and polylog time per pixel, in the size of the filter, to compute the median filter. A logarithmic time per pixel lower bound for the computation of the median filter is shown.**

***Index Terms*—Algorithm, lower bound, max filter, median filter, min filter, order filter.**

I. Introduction

Order filters, such as the median filter, are widely used as an effective tool for reducing certain types of noise and periodic interference patterns in signals and images without severely degrading the signal [3]. One of the more useful properties of the median filter is that in many situations, it does not blur edges and monotone changes in the signal, but it cleans up sporadic noise. The extreme cases of the order filters (min and max filters) are the primitive operations for all of the so-called morphological filters [5], [6]. In this correspondence, we describe fast algorithms to compute min, median, max, or any other order statistic filter transforms and show lower bounds for their computation.

We consider filters with a square shape with side p. It will be convenient to assume that p is odd and to denote $q = (p-1)/2$. Let A be an $n \times n$ matrix, where $n > p$. Then, for all i, j satisfying $q < i \leq n - q$ and $q < j \leq n - q$, let $A_{i,j}$ be the $p \times p$ square submatrix of A centered in $A[i, j]$. We say that an $n \times n$ matrix B is a min, median, or max filter transform of A if $B[i, j]$ equals, respectively, the min, median, or max of the elements of $A_{i,j}$. (The boundary elements can be treated as a special case or totally ignored.)

The min, max, and median transforms could be computed by considering each of the $A_{i,j}$ matrices separately. This approach would result in $O(p^2)$ operations for each element. The observation that the size of the symmetric difference of the sets of elements in $A_{i,j}$ and $A_{i,j+1}$ is $2p$ can be used to devise a better algorithm. Suppose that the p^2 elements of $A_{i,j}$ are stored in a 2-3 tree. Then, computing any order statistic of $A_{i,j}$ can be done in $O(\log p)$ time. Initializing such a tree requires $O(p^2 \log p)$ time. However, a 2-3 tree for $A_{i,j+1}$ can be constructed from the 2-3 tree for $A_{i,j}$ by executing p inserts and p deletes in time $O(p \log p)$. Thus, producing one row of B can be done in $O(p^2 \log p + np \log p + n \log p) = O(np \log p)$ time, i.e., $O(p \log p)$ time per element.

We describe an algorithm that takes constant time per element to compute the min or max filter and $O(\log^2 p)$ time per element to compute a median filter transform. We show an $O(\log p)$ lower bound for computing the median filter. Previous known algorithms for computing the 2-D median filter [3] took either $O(p^2)$ or $O(ph)$ time per element, where h is the number of grey levels in the image. Lower bounds for computing the median filter of $O(\log p)$ were shown in the 1-D case [2], [4]. As the proposed algorithm has a small constant, it is already superior for moderately sized filters.

II. 2-D Min and Max Filters

We present an algorithm for computing the filter transform for any semi-group operation (associative, binary), for example min, max, or +, which takes only constant time per element computed; this is obviously optimal.

Let $\diamond$ be any semi-group operation. In order to calculate the $p \times p$ filter, it suffices to compute row-by-row a 1-D p filter and, on the resulting filtered image, a column-by-column 1-D p filter. This follows from the associative law:

$$a_{1,1} \diamond a_{1,2} \cdots \diamond a_{1,p} \diamond a_{2,1} \cdots \diamond a_{p,p} =$$
$$(a_{1,1} \diamond a_{1,2} \cdots \diamond a_{1,p}) \diamond (a_{2,1} \diamond a_{2,2} \cdots \diamond a_{2,p})$$
$$\diamond \cdots \diamond (a_{p,1} \diamond a_{p,2} \cdots \diamond a_{p,p}).$$

Thus, it suffices to show how to compute a 1-D filter. In order to compute a 1-D p filter B for a $(2p-1) \times 1$ image $a_1, a_2 \cdots a_{2p-1}$, we compute the following values in $O(p)$ time:

$$\begin{aligned} R_i &= a_i \diamond a_{i+1} \diamond \cdots \diamond a_{p-1} = a_i \diamond R_{i+1} && 0 < i \leq p-1 \\ S_j &= a_p \diamond a_{p+1} \diamond \cdots \diamond a_{p+j} = S_{j-1} \diamond a_{p+j} && 0 \leq j \leq p-1 \end{aligned}$$

Then, $B[k]$ is $R_u \diamond S_v$, where $u = (2k - p + 1)/2$ and $v = (2k - p - 1)/2$ (the cases where the index is out of bounds are treated vacuously). Thus, the computation of the p values takes $O(p)$ time or $O(1)$ amortized time per element. Since + is a semi-group operation, we also have a constant time per element mean filter algorithm.

The 1-D algorithm can be used iteratively on all axes to compute a d-dimensional (where the filter shape is the p^d sized hypercube) semi-group filter in $O(d)$ time per element.

III. Lower Bound for the Median Filter Problem

Our lower bound is based on a reduction from sorting. Given an input of q^2 numbers to be sorted, we will see that the power of computing the median filter of size $p = 2q - 1$, combined with $O(q^2)$ preprocessing, is sufficient for sorting the inputs.

Manuscript received December 11, 1990; revised May 1, 1992. Recommended for acceptance by Associate Editor S. Tanimoto.

The authors are with the Department of Computer Science, The Hebrew University of Jerusalem, Jerusalem, Israel.

IEEE Log Number 9205813.

We arrange q^2 inputs for a sorting problem in a $q \times q$ matrix X. The matrix X is then placed in the middle of a $(3q-2) \times (3q-2)$ matrix A. The rest of A is filled by x^- and x^+, where x^- is a number less than all the inputs, and x^+ is a number greater than all the inputs. Clearly, x^- and x^+ can be found in $O(q^2)$ time.

Consider the matrices $A_{i,j}$ for all i, j with $q \le i, j < 2q$. The center of each of these matrices is a member of X, and all of them contain X as a submatrix. Let $X^+_{i,j}$ and $X^-_{i,j}$ be the number of x^+ and x^- in $A_{i,j}$. The layout of the x^- and x^+ in A is devised so that the following properties hold:

1. The p^2 elements of the top left submatrix $A_{q,q}$ consist of $(q-1)^2$ copies of x^+ (i.e., $X^+_{q,q} = (q-1)^2$), q^2 members of X, and $(q-1)^2+q^2-1$ copies of x^- (i.e., $X^-_{q,q} = (q-1)^2+q^2-1$). Thus, the median of $A_{q,q}$ is the minimum of X.
2. If both $A_{i,j}$ and $A_{i,j+1}$ exist, then
$$X^+_{i,j+1} = X^+_{i,j} + 1$$
$$X^-_{i,j+1} = X^-_{i,j} - 1.$$
In other words, in each columnwise step over the $A_{i,j}$ matrices, one of the x^- is replaced by a x^+.
3. If both $A_{i+1,j}$ and $A_{i,j}$ exist, then
$$X^+_{i+1,j} = X^+_{i,j} + q$$
$$X^-_{i+1,j} = X^-_{i,j} - q.$$
In other words, in each rowwise step over the $A_{i,j}$ matrices, exactly q of the x^- are replaced by x^+.

These properties guarantee that the median filter will sort the inputs. $B[q,q]$ is the minimum of X, and a raster scan over B generates the rest of X in ascending order.

We give the layout of A for the case $p = 9$, $q = 5$. For brevity, x^- and x^+ are marked as "$-$" and "$+$." In addition, since the exact placement of the input elements in X is irrelevant, they are collectively marked as x's.

$$\begin{matrix}
- & - & - & - & - & - & - & - & - & - & - & - & + \\
- & - & - & + & - & - & - & - & - & - & - & + & + \\
- & - & + & + & - & - & - & - & - & - & + & + & + \\
- & + & + & + & - & - & - & - & - & + & + & + & + \\
+ & + & + & + & x & x & x & x & x & + & + & + & + \\
+ & + & + & - & x & x & x & x & x & + & + & + & - \\
+ & + & - & - & x & x & x & x & x & + & + & - & - \\
+ & - & - & - & x & x & x & x & x & + & - & - & - \\
- & - & - & - & x & x & x & x & x & - & - & - & - \\
- & - & - & - & + & + & + & + & + & - & - & - & + \\
- & - & - & + & + & + & + & + & + & - & - & + & + \\
- & - & + & + & + & + & + & + & + & - & + & + & + \\
- & + & + & + & + & + & + & + & + & + & + & + & +
\end{matrix}$$

The reader can easily verify that the desired properties hold in this layout. This symmetrical construction is readily generalized for all values of q.

Theorem 1: Any algorithm for computing the p-sized median filter of an $n \times n$ input with $n \ge (3p-1)/2$ runs in $\Omega(\log p)$ amortized time per element.

Proof: If $n = (3p-1)/2$, then set $q = (p+1)/2$. The above construction shows that if a p-sized median filter can be computed in time T, then q^2 arbitrary inputs can be sorted in time $O(q^2 + T)$. It follows that $T = \Omega(q^2 \log q) = \Omega(p^2 \log p)$.

The generalization for $n > (3p-1)/2$ is done by selectively pruning the outputs of the median filter so that the remaining ones can be described as outputs of p-sized median filters applied to several $(3p-1)/2 \times (3p-1)/2$-sized independent matrices. It is not hard to see that no more than a constant fraction of the output values are pruned in this process. Then, each of these median filters can be used to solve an independent sorting problem. ■

IV. A 2-D Median Filter Algorithm

We present an algorithm for the case $n = 2p-1$. The algorithm produces the p^2 median filter transform values in $O(p^2 \log^2 p)$ time. The generalization for $n > 2p-1$ is straightforward, and the performance remains $O(\log^2 p)$ time per element.

The general outline of the algorithm is as follows: All $(2p-1)^2$ elements of the input matrix are initially stored in a data structure. The algorithm works in *phases*; in each phase, elements from p consecutive columns are marked *active* in the data structure. In a phase, the algorithm finds the median of every p consecutive rows of the active elements. The data structure supports each median computation in $O(\log^2 p)$ time.

The essence of the algorithm is in the implementation of the data structure. We will first describe a data structure we call an *implicit offline balanced binary search tree* (IOBBST). We then show how IOBBST's are used in our data structure.

Implicit Offline Balanced Binary Search Trees

Suppose that an ordered set S is represented in a binary search tree T of height h. In addition, suppose that each node α of T stores the number of nodes in the subtree of T rooted at α. Then, the following two facts are basic properties of binary search trees.

Fact 1: Given $x_1, x_2 \in S$, the size of the set $\{y \in S : x_1 \le y \le x_2\}$ can be found in $O(h)$ operations.

Fact 2: Given k, the kth ranked element of S can be found in $O(h)$ operations.

The minimal value for h is achieved if T is an *essentially complete binary tree*, i.e., a tree in which all levels except (maybe) the last are completely full. In this case, $h = \lceil \log |S| - 1 \rceil$. We say that a binary search tree for a set S is *balanced* if its height h satisfies $h = O(\log |S|)$.

Binary search trees are most useful for representing dynamically changing sets. By using tree-restructuring operations, it is possible to implement insertion and deletion updates in logarithmic time while maintaining the balance property. (Numerous such restructuring schemes (AVL trees, weight-balanced trees, etc.) are described and analyzed in the literature.)

IOBBST's are suitable for an off-line version of the problem of maintaining a balanced binary search tree for a dynamically changing set S. Specifically, suppose that S is initially empty and that updates to S are given in a sequence $\mathrm{op}_1(x_1), \ldots, \mathrm{op}_m(x_m)$, where op_i is either an insert or a delete operation. Let $U = \{x_1, \ldots, x_m\}$, and let $u = |U|$. We then build an essentially complete binary search tree for U represented implicitly in an array $A[1 \ldots u]$ i.e., the root of the tree is stored in $A[1]$, and if a tree node is stored in $A[i]$, then the node's left (right) child is stored in $A[2i]$ ($A[2i+1]$).

A node $A[i]$ consists of the following fields:

1. $A[i].x$, which is the key associated with the node
2. $A[i].\mathit{active}$ of type boolean, which is true whenever $A[i].x \in S$ (in this case, we say that the node is active)
3. $A[i].\mathit{count}$ of type integer, which stores the number of active nodes in the subtree rooted at $A[i]$.

An insertion of an element x is implemented by activating its node and by incrementing the *count* field on the path nodes leading from the root to x. Deletion of x is done by deactivating its node and by decrementing the *count* field in all the nodes on the path leading from the root to x.

Clearly, the IOBBST structure requires $O(u)$ space and supports rank queries (as described in Facts 1 and 2) as well as updates in

$O(\log u)$ operations. The initial construction of an IOBBST can be done in $O(u \log u)$ operations.

B. The Data Structure

The $(2p-1)^2$ image elements are stored in a data structure. The data structure consists of a main IOBBST T, which stores all $(2p-1)^2$ input elements and uses *element value* as the search key. Each node α in the main IOBBST, in addition to its key, active, and count fields, consists of a secondary IOBBST R_α of all its descendants in which the elements *row value* (j for $A[j, \cdot]$) is the key.

From Fact 1, it follows that R_α may be used to compute the size of the set of active elements in the subtree T_α that come from any set of consecutive rows of the original matrix A in logarithmic time.

From Fact 2, it follows that computing the median of the active elements from a set of consecutive rows of A requires $O(\log p)$ visits to nodes of T. Extracting the size information of each node α calls for $O(\log p)$ operations in R_α. This amounts to a total of $O(\log^2 p)$ operations for each median computed.

C. Algorithm Description

The algorithm runs in p phases. As before, let $q = (p-1)/2$. Then, at the beginning of phase $j = q+1, \ldots, p+q$, all $(2p-1)p$ elements of the p submatrices (windows) $A_{i,j}$, $q < i \leq q+p$ are active.

```
1. Initialize $T$ and the $R_\alpha$'s setting all elements from the first $p$
   columns of the matrix to be active.
2. for $j := q+1$ to $p+q$ do
   (a) for $i := q+1$ to $p+q$ compute the median of $A_{i,j}$.
       A median computation is performed by applying
       a selection process (Fact 2) on $T$, where the size
       of a subtree of $T$ rooted at node $\alpha$ is taken as
       the result of range query (Fact 2) with $i-q$ and
       $i+q$ as the range query limits applied to $R_\alpha$
   (b) Deactivate the elements in column $j-q$.
       The deactivation of an element is done by a
       binary search using the element value in $T$. For
       every node $\alpha$ encountered in the search, we deac-
       tivate the element in the secondary tree $R_\alpha$. Since
       $R_\alpha$ is an IOBBST, deactivation of an element
       in it is done by doing a binary search, using
       the element row as a key, while decrementing
       the count field of all nodes encountered in the
       search.
   (c) Activate the elements in column $j+q+1$.
       Activation is similar to the deactivation.
```

D. Analysis

1) Space Requirements: Each of the $(2p-1)^2$ elements is stored once in T. Let β be an image element. Then, β is stored in R_α for each α ancestor of β in T. This accounts for $O(p^2 \log p)$ space for the whole data structure.

2) Initialization Time: Initialization takes $O(p^2 \log p)$ time in the following way: Construct T in $O(p^2 \log p)$ time, and build the R_α's from the bottom up. For a node α in T with sons α_1, α_2, R_α is the merge of R_{α_1} and R_{α_2}. Since each element is involved in at most $O(\log p)$ merge operations, the total construction of the R_α's takes $O(p^2 \log p)$ time.

3) Computing a Median: Each median takes $O(\log^2 p)$ time to compute. The number of nodes of T visited is $O(\log p)$. In each node α visited, $O(\log(p))$ nodes of R_α are visited to count how many active elements (being in columns $j-q, \ldots, j+q$) are in the correct rows ($i-q, \ldots, i+q$) of subtree T_α.

4) Time for Activation/Deactivation: Basic update operations to the data structure require $O(\log^2 p)$ time. For an activation of an element β, one needs to update all the R_α's between the root and β such that α is an ancestor of β in T. For each R_α so updated, we need to mark β as active and update the counts of each subtree of which β is a member. Deactivation is carried out in a similar manner.

5) Time for Phase: In proceeding to the next column of A, $2p-1$ elements have to be deactivated, and $2p-1$ elements have to be activated. This phase maintenance takes $O(p \log^2 p)$ time, which can be charged to the p medians produced.

Theorem 2: The $p \times p$ median filter can be computed in time $O(\log^2 p)$ time per element processed, using $O(p^2 \log p)$ memory.

Note that no pointers are needed in the data structure as they use only simple implicit representation of complete binary trees. We also avoid the tedious complicated handling and the extra overhead needed for complete trees. Thus, the algorithm is not only theoretically interesting but also practical.

Using similar ideas a d-dimensional filter with window of size p^d can be computed in $O(\log(p^d))$ time.

V. Weighted Median

In many cases, one is interested in a filter with a large support but with a localized property. This is usually achieved by giving larger weights to elements closer to the center. The analog in the median filter is the weighted median filter [1] defined as follows. The weighted median of $a_1, a_2, \ldots, a_n$ with weight function $w(1), w(2), \ldots, w(n)$ is the maximal a for which

$$\sum_{a_i < a} w(i) \leq \frac{1}{2} \sum_{i=1}^{n} w(i).$$

If $w(i) \equiv 1$, then this definition reduces to the regular median.

Given a window

$$\left\{ A[l, m] \middle| \begin{array}{l} i-q \leq l \leq i+q \\ j-q \leq m \leq j+q \end{array} \right\}$$

We consider weight functions $w_{i,j}(l, m)$ that are bi-variate polynomials of $(l-i)$ and $(m-j)$. (For example, $w_{i,j}(l, m) = q^2 - (l-i)^2 - (m-j)^2$.)

Theorem 3: If the weight function is a bivariate polynomial of degree d, then the weighted median filter can be computed in $O(d^2 \log^2 p)$ time per element and $O(d^2 p^2 \log p)$ space.

Proof: Observe that in the previous section algorithm, the median was found by answering range queries of the form *"What is the size of the intersection of a window and a subtree of T?"*. To find the weighted median, we need to answer queries of the form *"What is the weight of intersection of a window and a subtree of T?"* We show how to modify R_α to support weight queries.

In each node of R_α, we store the moments $\sum 1, \sum l, \sum m, \sum lm, \sum l^2, \sum m^2, \ldots$ of the active elements $\langle l, m \rangle$. Given a weight function $w_{i,j}(l, m)$, which is a bivariate polynomial of $(l-i)$ and $(m-j)$, it can be rewritten as a bivariate polynomial w' of l and m, with coefficients depending on i and j. The sum of this polynomial is computed by substituting the moments for the monomial in l and m ($lm \Leftrightarrow \sum lm$, etc.).

As there are $O(d^2)$ different monomials in w', the storage requirements of R_α increase by a factor of $O(d^2)$. Likewise, each of these moments need to be updated when elements activate and deactivate, increasing the run time by a factor of $O(d^2)$. ■

REFERENCES

[1] D. R. Brownrigg, "The weighted median filter," *Commun. Assoc. Comput. Mach.*, vol. 27, pp. 204–208, Aug. 1984.

[2] J. Gil, W. Steiger, and A. Wigderson, *Running Medians*, to be published.

[3] T.S. Huang, *Two-Dimensional Digital Signal Processing II: Transfoms and Median Filters.* Berlin: Springer-Verlag, 1981.

[4] J. Katajainen M. Juhola, and T. Raito, "Comparison of algorithma for standard median filtering," *IEEE Trans. Signal Procesing*, vol. 39, pp. 204–208, Jan. 1991.

[5] J. Serra, *Image Analysis and Mathematical Morphology.* New York: Academic, 1982.

[6] M. Werman and S. Peleg, "Min max operators in texture analysis," *IEEE Trans. Patt. Anal. Machine Intell.*, vol. PAMI-7, pp. 730–733, Nov. 1985.

Symmetry Identification of a 3-D Object Represented by Octree

Predrag Minovic, Seiji Ishikawa, and Kiyoshi Kato

***Abstract*—An algorithm for identifying symmetry of a 3-D object given by its octree is presented, and the symmetry degree (a measure of object symmetry) is proposed. The algorithm is based on traversals of the octree obtained by the principal axis transform of an input octree. An object can be in an arbitrary position and with arbitrary orientation within the octree space, and a wide range of symmetries represented by groups of proper and improper rotations can be identified. It is shown that the octree data structure supports these operations well, especially for objects whose symmetry types are simpler or equal in complexity with a four-fold rotational symmetry. The operation of the algorithm is illustrated using some synthetic test objects. The results, which are composed of identified symmetry types and the corresponding symmetry degrees, were satisfactory.**

***Index Terms*—Octree, principal axis transform, symmetry, symmetry identification, 3-D object recognition.**

I. INTRODUCTION

Symmetry is one of the most important features of objects. Identifying symmetry is an interesting problem in itself in computational geometry, but the notion of symmetry can be useful in other fields as well. Among possible applications, an obvious one is in data compression. A representation of a symmetric object is possible by means of a model of its symmetric portion, its symmetry type, and positions of its planes and/or axes of symmetry. In object recognition, symmetry can be used as a special object feature that would reduce the number of candidates for matching in the library. A more advanced application can be automatic correction of an "almost symmetric" object to a symmetric one. This may be useful in computer vision for object models obtained from camera or range images, in CAD systems for object models built on the basis of digitized line drawings, or in computer graphics for models of biomedical organs obtained by computed tomography. However, in spite of these potentialities, the symmetry identification problem has not been treated with adequate attention. In this contribution, in an attempt to alleviate the deficiency, we present an algorithm for symmetry identification of 3-D objects that can detect a wide range of symmetries.

In what follows, we shall briefly survey existing algorithms for symmetry identification in 2-D and 3-D cases. One of the first results for 2-D line drawings was presented by Evans [8]; his program tested some primitive figures for bilateral symmetry about the horizontal and the vertical axis. The next approach using the quadtree image representation was reported in [2]. It is also limited in scope because image symmetry can be detected along four directions only (horizontal, vertical, and the left and the right diagonal), and it is rather impractical since an image has to be aligned with the quadtree space. A group of algorithms for polygons, point sets, and collections of lines is surveyed in [7]. These algorithms usually work in $O(n \log n)$ time, where n is the number of corresponding elements, but they often impose some restrictions (e.g., convexity) on input figures. Furthermore, most of them are proposed for continuous coordinates; hence, their performance in a digital, noisy environment can be problematical. Other attempts include methods for identification of n-fold rotational symmetry of digitized, simple-connected closed curves [12], detection of both ordinary and skewed symmetry [9] of arbitrary images (its time complexity can be excessive, and it is not fully automatic), and analysis of "almost symmetric" images [14] using the so-called "coefficient of symmetry."

Symmetry is also the subject of active research in psychophysics and the theory of perception. There have been some attempts to quantitatively describe symmetry, and several symmetry measures were proposed [25], [26], but no attempts to locate symmetry axes in an analytical way have been reported. It is interesting to note that a special device called "the symmetrograph" was constructed to locate symmetry axes mechanically [26].

The number of reported algorithms for 3-D objects is considerably smaller. The first attempt in this direction was probably done by Gips [10]. He analyzed the relationship between two connected strings of cubes with the mirror symmetry being one of the possible solutions. Three recent and more general solutions are summarized in [7]. The algorithm reported by Sugihara [19] for congruity detection of polyhedra can be used for identifying proper rotational symmetries of a single polyhedron (as its congruent mappings onto itself). It is applicable to a wide class of polyhedra without holes, and it works in $O(n \log n)$ time (n is the number of edges). Wolter *et al.* [23] solved the problem for polyhedra whose surface graphs are planar and triply connected in linear time, but they noted that it may not be practical because of a large constant involved. Finally, Atkinson [3] reported a solution for determining congruity of two finite sets of points applicable for symmetry identification of a single set in $O(n \log n)$ time (n is the number of points). Although these three algorithms are important contributions for computational geometry, they do not seem applicable for a practical environment where errors are inevitable. Besides the restrictions on input objects mentioned before, they cannot successfully handle curvilinear objects either.

Recently, Bigun [4] addressed the problem of generalized symmetry in local neighborhoods of multidimensional gray-scale images. Linear, circular, parabolic, and other generalized symmetry types can be detected (e.g., linear symmetry is exhibited by an object with isogray parallel cross sections), but some aspects of "classical"

Manuscript received June 13, 1990; revised August 25, 1992. Recommended for acceptance by Associate Editor N. Ahuja.

P. Minovic was with the Department of Electric, Electronic and Computer Engineering, Faculty of Engineering, Kyushu Institute of Technology, Kitakyushu, Japan. He is now with SECOM Intelligent Systems Laboratory, Tokyo, Japan.

S. Ishikawa and K. Kato are with the Department of Electric, Electronic, and Computer Engineering, Faculty of Engineering, Kyushu Institute of Technology, Kitakyushu, Japan.

IEEE Log Number 9206563.