The Visual Computer

Optimal algorithms for symmetry detection in two and three dimensions*

Jan D. Wolter, Tony C. Woo[2], and Richard A. Volz[1]

[1] Department of Electrical Engineering and Computer Science,
[2] Department of Industrial and Operations Engineering,
The University of Michigan Ann Arbor, MI 48109, USA

Exact algorithms for detecting all rotational and involutional symmetries in point sets, polygons and polyhedra are described. The time complexities of the algorithms are shown to be $\Theta(n)$ for polygons and $\Theta(n \log n)$ for two- and three-dimensional point sets. $\Theta(n \log n)$ time is also required for general polyhedra, but for polyhedra with connected, planar surface graphs $\Theta(n)$ time can be achieved. All algorithms are optimal in time complexity, within constants.

Key words: Symmetry – Similarity – Computational geometry – Pattern matching – Graph isomorphism

An object is symmetrical if its shape is unchanged under an affine transform. This paper presents optimal algorithms to find several types of symmetry for polygons, point sets, and polyhedra - point, line and plane symmetries.

The authors originally encountered the need for computing symmetry in a robotics application, in which a set of images of polyhedra were generated for the training of a vision system (Wolter et al. 1985). Knowledge of the symmetry of the object was necessary to eliminate redundant orientations. Because of its potential capability in data extraction and data compaction, symmetry is useful for solving problems in image analysis and computer graphics. Several algorithms for detecting symmetry in images have appeared in the literature. Davis (1977) described a method for finding lines of symmetry in images by clustering local symmetries. Parvi and Dutta Majumder (1983) detected approximate lines of symmetry in chain coded polygons. Friedberg and Brown (1984) used moments to find lines of skewed symmetry. Johansen et al. (1984) have presented algorithms based on the boundary representations of objects which may be used to detect symmetries. They extend an algorithm by Tanimoto (1981) to encode polygons or polyhedra into nondeterministic finite state automata. This requires $O(n^2)$ states for polyhedra, and $O(2^{\sqrt{n}})$ states for polyhedra, where n is the number of verties.

This paper presents a set of algorithms for solving the following class of problems. Given either a point set or a boundary representation of a polygon or polyhedron, all rotational and involutional symmetries are found. For polygons and polyhedra with connected, planar surface graphs, $\Theta(n)$ operations are used. For all other structures, $\Theta(n \log n)$ operations are required. All algorithms are shown to be optimal within a constant. These algorithms are based on the algorithm for linear time polygon similarity published by Manachar (Manachar 1976; Akl 1978; Bykat 1979) and on the algorithm for linear time graph isomorphism by Hopcroft and Wong (1974). The computational model used throughout this paper is an RAM-based algebraic decision tree (Lee and Preparta 1984).

* Work on this paper was partially supported by the Air Force Office of Scientific Research under contract number F4920-82-C-0089. Work by the first author was supported through an IBM Graduate Pre-Doctoral Fellowship

Definitions

In this paper a d-dimensional object Π is defined as a set of points $\{p_1, p_2, \ldots\}$ in d-dimensional space. The transform of an object $T(\Pi)$ is the object $\{T(p_1), T(p_2), \ldots\}$. Π is symmetrical under the transform T, if $T(\Pi)=\Pi$.

The transforms of interest in this paper fall in two classes: rotational transforms and involutional transforms. Let $R_{a,\theta}$ denote a rotation transform of θ degrees about the $(d-2)$-dimensional axis a. All possible rotational symmetry transforms can be written as $C_{a,k} \equiv R_{a, 360/k}$ where k is a natural number. If Π is symmetrical under $C_{a,k}$ then a is called a "k-fold point of rotational symmetry" in two dimensions, or a "k-fold line of rotational symmetry" in three dimensions. Note that the transform $C_{a,1}$ is the identity transform. A one-fold axis of symmetry is called a *trivial* axis, since every object Π has such a symmetry.

The second class of transforms are *involutional transforms*, denoted $Z_{b,k}$, where b is a $(d-1)$-dimensional axis, and k is a natural number. In two dimensions, only $Z_{b,1}$ is defined. This denotes a reflection through the line b. If a two-dimensional (2D) point set is symmetrical under $Z_{b,1}$, then b is called a "line of reflectional symmetry." In three dimensions, let b be a plane, and let $\bar{b}$ be a line perpendicular to b. Then the transform $Z_{b,k}$ is a rotation of $360/k$ degrees around the line $\bar{b}$, followed by a reflection through the plane b. If a Π is symmetrical under $Z_{b,k}$, then a line $\bar{b}$ is said to be a "k-fold line of involutional symmetry." $Z_{b,1}$ and $Z_{b,2}$ are of particular interest. $Z_{b,1}$ is pure reflection through the plane b, and if Π is symmetrical under that transform, b is said to be a "plane of reflective symmetry." Note that $Z_{b,1}$ is self-inverse. $Z_{b,2}$ is equivalent to inversion through the point where b intersects $\bar{b}$. We call such points "points of inversional symmetry."

Any transform $R_{a,k}$ or $Z_{b,k}$ leaves at least one point fixed in space. If an object is symmetrical under a transform, it can be shown that the centroid γ of the object must be a fixed point under that transform. Since the centroid can be calculated in linear time, it is a very convenient starting point from which to search for symmetries.

A rotational transform $R_{a,\theta}$ can be expressed as a composite of two reflectional transforms, $Z_{b,1} \circ Z_{c,1}$, such that b and c intersect at a to form an angle of $\theta/2$ degrees. Because of this, any object with more than one reflectional symmetry must also be rotationally symmetric.

The symmetries which may occur together form symmetry groups. All possible symmetry groups for two and three dimensions have been formally classified (Martin 1982; Lockwood and Macmillan 1978).

Basic ideas

The algorithms in this paper will all follow the same general outline, which consists of three steps:

1. ORDER: sort the points of the object into cycles
2. ENCODE: encode each cycle into a string of symbols
3. CHECK: test the symmetry of the encoded string

Before describing the specific algorithms in detail, we will define the structures produced by the ORDER and ENCODE steps. In these definitions, **T** is the set of all symmetry transforms to be tested for.

The ORDER step takes the vertex set, $P \subset \Pi$, and forms it into a cycle $\Gamma = \langle c_0, c_1, \ldots, c_{n-1} \rangle$, where each c_i is one of the n elements of P. This ordering is a *cycle* when it has the property that if Π is symmetrical under any transform $T \in \mathbf{T}$ such that $T(c_i)=c_j$, then for all k, $T(c_{i+k}) = c_{j+k}$. (Note that in this paper, all additions and subtractions in subscripts are assumed to be done with the appropriate modulus, in this case, n.)

The ENCODE step converts a cycle Γ into a finite string S on an infinite alphabet. Each element c_i in the cycle will be encoded into an m-tuple of symbols s_i, such that for any transform $T \in \mathbf{T}$ under which P is symmetric, if $T(c_i) = c_j$, then $s_i = s_j$. Furthermore, the string should be such that two objects Π_1 and Π_2 have encodings which are cyclic permutations of each other if, and only if, for some $T \in \mathbf{T}$, $T(\Pi_1) = \Pi_2$. In other words, it must contain enough information about the original object Π_1 to allow the construction of an object Π_2 which is equivalent to Π_1 under some transform in **T**.

The CHECK step makes use of the properties of the encoded string to locate all transforms in **T** which are symmetries for the object. This includes several different tests for different kinds of symmetry, but all are variations of the rotational similarity test of Manachar (1976). His algorithm is as follows.

Algorithm 0: similarity of cycles

Problem 0: Given two encoded cycles, S and T, check if S is a cyclic permutation of T, i.e., if there is any k such that

$$\langle s_k, s_{k+1}, \ldots, s_{k+n-1}\rangle = \langle t_0, t_1, \ldots, t_{n-1}\rangle$$

To solve this problem, a substring pattern matching algorithm such as that of Knuth et al. (1977) is used. Given two strings of total length m on a possibly infinite alphabet, the Knuth algorithm finds the first occurrence of one in the other in $\Theta(m)$ time.
Algorithm 0 then consists of two main steps. First, we construct the following two strings, A and B, from the encoded cycles, S and T.

$$A = \langle s_0, s_1, \ldots, s_{n-1}\rangle$$
$$B = \langle t_0, t_1, \ldots, t_{n-1}, t_0, t_1, \ldots, t_{n-2}\rangle$$

Second, we use the string pattern matching algorithm to determine whether A is a substring of B. If it is, then S is a cyclic permutation of T.

Algorithm 1: symmetry of a polygon

Problem 1. Given a planar polygon, find all rotational and reflectional transforms under which it is symmetric.
A polygon is represented by a sequence of n points (vertices), $P = \langle p_0, p_1, \ldots, p_{n-1}\rangle$, and n line segments (edges), $E = \langle e_{0,1}, e_{1,2}, \ldots, e_{n-1,0}\rangle$, such that the edge $e_{i,i+1}$ has endpoints p_i and p_i+1. This representation is unique up to a cyclic permutation of E and P.

Polygon ORDER

Theorem 1.1. *cycle property of polygons. The vertex list P of a polygon is a cycle for rotation transforms.*

Proof. Since an edge connects p_i and p_{i+1}, and the polygon is symmetrical under the transform T, there must be an edge connecting $p_j = T(p_i)$ and $T(p_{i+1})$. Thus $T(p_{i+1})$ equals either p_{j+1} or p_{j-1}. The latter case can be excluded, because the vorticity of the triangle (γ, p_i, p_{i+1}) would be opposite that of $(T(\gamma), T(p_i), T(p_{i+1}))$, and this is impossible if T is a rotation transform. Thus, we have that $T(p_{i+1}) = p_{j+1}$, which by induction implies that $T(p_{i+k}) = p_{j+k}$, so P satisfies the cycle condition.
Due to Theorem 1.1, the ORDER step of the algorithm is unnecessary for a polygon, since the vertex list P already forms a valid cycle.

Polygon ENCODE

The ENCODE step generates a two-tuple of measures for each point which describes the location of that vertex. For a measure to be a candidate for inclusion in the encoding, it should be invariant under rotation. These are measures of the location of the point relative to the centroid or relative to adjacent points of the polygon. Possibilities include:
M1. Distances between adjacent vertices
M2. Distances of vertices from the centroid
M3. Angles formed by edges at each vertex
M4. Angles formed at the centroid by two adjacent vertices
In his polygon similarity algorithm, Bykat (1979) uses measures M1 and M2. However, these do not yield a unique encoding. Figure 1a shows a polygon with vertices $a_1, a_2, a_3, b_1, b_2, c_1, c_2$ and c_3, such that

$$\mathrm{centroid}(a_1, a_2, a_3) = \mathrm{centroid}(b_1, b_2)$$
$$= \mathrm{centroid}(c_1, c_2, c_3) = \gamma$$

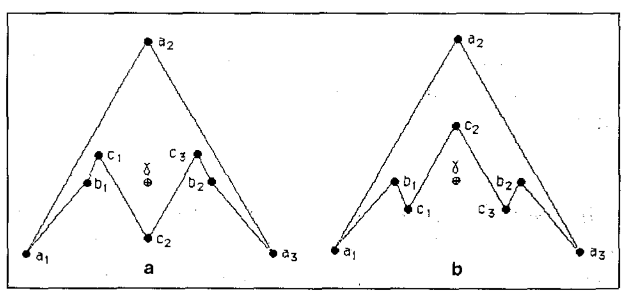

Fig. 1a, b. Non-uniqueness of length-radius encoding for polygons

Figure 1b is the same polygon but with the coordinates of points c_1, c_2 and c_3 reflected through the line (b_1, b_2). Both figures have the same centroid and edge lengths, and corresponding points are the same distance from the centroid, yet the polygons are not similar.

Manachar (1976) encodes each vertex of the polygon cycle into a two-tuple containing measures M1 and M3. Thus

$$s_i = \langle \mathrm{dist}(p_i, p_{i+1}), \mathrm{angle}(p_{i-1}, p_i, p_{i+1}) \rangle$$

This plainly satisfies both the requirement that vertices which can be mapped into each other by a symmetry transform have the same encoding, and the requirement that the polygon be completely described by the encoded string. Constructing the string $S = \langle s_1, s_2, \ldots, s_{n-1} \rangle$ takes only linear time.

An encoding using M2 and M4 is very convenient if the polygon is specified in polar coordinates about the centroid. This case will arise as part of the point set symmetry algorithm.

Polygon CHECK

To check for the rotational symmetry of a polygon, we need only to make a slight modification to algorithm 0. Let S be the encoded cycle of the polygon. We search for

$$A = \langle s_0, s_1, \ldots, s_{n-1} \rangle$$

in the string

$$B' = \langle s_1, \ldots, s_{n-1}, s_0, s_1, \ldots, s_{n-1} \rangle$$

If A first occurs in B' at offset $k-1$ then the polygon must have n/k-fold rotational symmetry. At least a one-fold symmetry will be found for any polygon, since, if A is found nowhere else in B', it will be found at offset $n-1$.

Having found the rotational symmetries of the polygons, we now test for the reflectional symmetries.

Theorem 1.2. *reflection and rotation in polygons. Let P be a polygon with centroid γ, and b be an arbitrary line containing γ. Then there exists an angle of rotation θ such that $R_{\gamma,\theta} \circ Z_{b,1}(P) = P$ if, and only if, P has a some line of symmetry c.*

Proof. If the polygon is symmetrical under the reflection transform $Z_{c,1}$ then

$$P = Z_{c,1}(P)$$

Any arbitrary reflection transform $Z_{b,1}$ is self-inverse. So

$$P = Z_{c,1} \circ Z_{b,1} \circ Z_{b,1}(P)$$

Suppose b and c intersect at γ forming an angle $\theta/2$. Then, their composition is $R_{\gamma,\theta}$. Then

$$P = R_{\gamma,\theta} \circ Z_{b,1}(P)$$

On the other hand, if the polygon has no line of symmetry, the reversal of the above argument leads to a contradiction.

Using this theorem, we can test for lines of symmetry by reflecting one copy of the polygon about any line b containing the centroid and then using algorithm 0 to see if it can be rotated onto the original polygon.

The reflection of the polygon can be found simply by taking the vertices in the order opposite to that given in P. It would be possible to repeat the ENCODE step for the reversed polygon, but it is more efficient to construct it directly from the forward encoded cycle. For example, if the encoding is based on measures M1 and M3, then the reversed encoding R of S would have terms

$$r_i = \langle \mathrm{dist}(p_{n-i}, p_{n-i-1}),\ \mathrm{angle}(p_{n-i+1}, p_{n-i}, p_{n-i-1}) \rangle$$

Thus R can be found simply by rearranging the terms of the encoding S described above.

Since we know that the polygon has k-fold rotational symmetry, the CHECK algorithm for reflectional symmetry can be improved by looking at only k symbols in the string. The test is then to use algorithm 0 to find if string

$$S' = \langle s_0, s_1, \ldots, s_{k-1} \rangle$$

is cyclically similar to

$$R' = \langle r_0, r_1, \ldots, r_{k-1} \rangle$$

If a match between these strings is found index j, and $k-j$ is odd, then there is a line of symmetry bisecting the angle at $p_{(k-j-1)/2}$. If $k-j$ is even, it bisects the edge connecting $p_{(k-j-2)/2}$ and $p_{(k-j)/2}$. The $k-1$ other lines intersect the first line at the centroid, forming angles of $360/k$ degrees. Thus, reflectional symmetry can be found in $\Theta(k)$ operations after $\Theta(n)$ processing to find the rotational symmetry.

Algorithm 2a: symmetry of a 2D point set

Problem 2a. Given a finite 2D point set, find all rotations and reflections under which that point set is symmetrical.
A d-dimensional point set ($d>0$) is any set of n points $P=\{p_0, p_1, \ldots, p_{n-1}\}$ in d-dimensional space. No ordering of the points is assumed.

Theorem 2.1. *complexity of point set symmetry testing. To find the symmetries of a d-dimensional point set, $\Omega(n \log n)$ operations are required.*

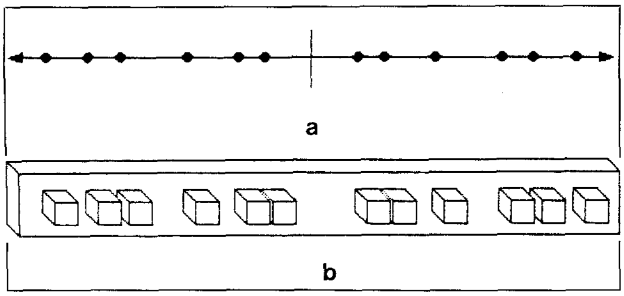

Fig. 2a, b. Complexity of symmetry detection

Proof. Consider a one-dimensional (1D) point set whose centroid lies at the origin (Fig. 2a). To test for reflectional symmetry through the origin, we must determine if the set of absolute values of the coordinates of points on the negative axis is equivalent to the set of coordinates of points on the positive axis. This is a set equivalence problem. Suppose there are $n/2$ points in each set. To verify that two sets are equivalent, it is necessary to find which of the $(n/2)!$ permutations of the first set forms the second. The decision tree for this problem is identical to that for comparison sorting (Aho et al. 1974); so that, as in comparison sorting, $O(n \log n)$ time is required. The 1D point set is a special case of all higher dimensional problems, so this lower bound applies in all dimensions.
We will now develop an algorithm to find the symmetries of a 2D point set in $\Theta(n \log n)$ time. It will be based on the same three steps used in the polygon algorithm.

Point set ORDER

In the polygon problem the cycle of points was given. For point sets it must be computed. Suppose the points are sorted by their polar coordinates around the centroid, taking the angle as the primary sort key and omitting any points at the centroid. This produces a star-shaped polygon in which the points are connected in clockwise order around the centroid. If the point set is rotated, this order will be preserved, since each point is rotated by the same angle. Thus a rotation which superimposes point i on point j is a symmetry transform only if it also superimposes point $i+k \bmod n$ on point $j+k \bmod n$, for all k. Therefore, this ordering of the points qualifies as a cycle. Since the algorithm requires sorting, its complexity is $\Theta(n \log n)$.
In practice, this algorithm may have a serious problem. Figure 3a shows a 2D point set with

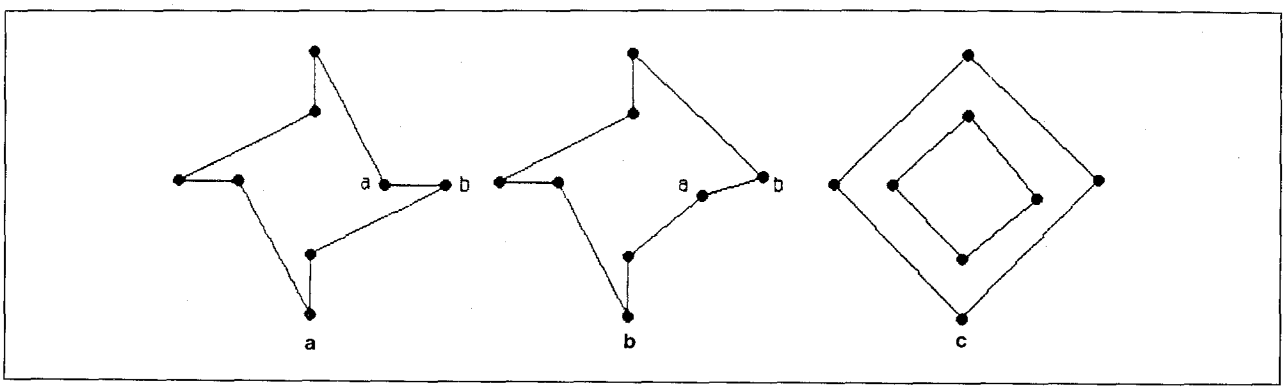

Fig. 3a–c. Effect of errors on point-set ORDER algorithm

the vertices sorted properly. But a very small error in the location of point a or point b may give rise to the cycle shown in Fig. 3b, which is not symmetrical. This behavior makes the algorithm very sensitive to round-off errors.
The problem of finding approximate symmetries in point sets is beyond the scope of this paper. However, we will describe a modification to the algorithm which makes it more robust in cases where the errors are much smaller than the distances between points, as for round-off errors. First, all points whose radii are equal within some ε are formed into cycles, and then each cycle is sorted by angle. This produces a set of cycles $\{\Gamma_1, \Gamma_2, \ldots, \Gamma_m\}$ instead of a single cycle (Fig. 3c). This modified algorithm is $\Theta(n \log n)$.

Point set ENCODE

The algorithm to encode the cycles is essentially the same as that in the polygon problem. Each point is represented by the difference between the polar angle coordinates of the point and its successor. The radii of the points need not be included, since they are constant within each cycle. Of course, other encodings could be used.

Point set CHECK

The tests to check a cycle for rotational and reflectional symmetry are exactly the same as those for polygons. We must, however, apply the tests to all cycles of the point set. Let cycle Γ_i have o_i-fold rotational symmetry. The degree of symmetry for the total point set is the greatest common divisor of the orders of the rings, $k = \mathrm{GCD}(o_1, o_2, \ldots, o_m)$.

Theorem 2.2. *complexity of GCD. Finding $GCD(o_1, o_2, \ldots, o_m)$ requires only linear time.*

Proof. To find $\mathrm{GCD}(a, b)$ requires $\Theta(\log \max(a, b))$ time (Aho et al. 1974; Knuth 1981), and $\mathrm{GCD}(a, b) \leqq \min(a, b)$. Thus the total complexity is of an order less than or equal to

$$\sum_{i=2}^{m} \log \max(o_i, \min(o_1, \ldots, o_{i-1}))$$

since $\log x \leqq x$, this is less than or equal to

$$\sum_{i=2}^{m} \max(o_i, \min(o_1, \ldots, o_{i-1}))$$

Let m_i be $\min(o_1, \ldots, o_i)$. Then $m_1 = o_1$ and

$$m_i = \begin{cases} o_i & \text{if } \max(o_i, m_{i-1}) = m_{i-1} \\ m_{i-1} & \text{if } \max(o_i, m_{i-1}) = o_i \end{cases}$$

Thus m_i for $i > 2$ is always the value not taken by $\max(o_i, m_{i-1})$. Therefore, every o_i appears in the sum exactly once except m_m, and the previous sum is equal to

$$\sum_{i=1}^{m} o_i - \min(o_1, \ldots, o_m)$$

This is less than or equal to n, since each o_i is less than or equal to the number of points in cycle Γ_i and each point is in only one cycle. Thus finding $\mathrm{GCD}(o_1, o_2, \ldots, o_m)$ requires $O(n)$ operations. From the case in which there is only one point in each cycle, it can be seen that this is, in fact, $\Theta(n)$. Thus the rotational symmetry can still be found in $\Theta(n)$ time, even when there are multiple cycles.
Having done this, we can check the reflectional symmetry of each cycle. If all have lines of symmetry which are colinear, then the point set has that line of symmetry. This can be done in linear time, given that we know the rotational symmetry of the point set and so need consider only one line per cycle.
Thus, in two dimensions, all symmetries of a point set can be found in $\Theta(n \log n)$ operations. (Only the ORDER step actually requires $\Theta(n \log n)$ operations. The other steps are linear.) This is optimal, by Theorem 2.1.

Algorithm 2b: axial symmetry of a 3D point set

Problem 2b. Given an axis and a 3D point set, find the rotational symmetry of the polyhedron about that axis, and find all planes of reflectional symmetry containing that axis.
Note that in this section, only the symmetries about a given axis are tested. The problem of proposing lines of symmetry will be considered in a later section.
All three steps for this algorithm are direct extensions of the 2D ones. In the ORDER step,

we first specify the points in a cylindrical coordinate system whose origin is at the centroid, and whose z-axis is parallel to the axis of rotation. We can then sort the points by their coordinates - first partitioning points whose radii and z-coordinates fall within some ε of each other into cycles, and then sorting each cycle by the angle. This requires $\Theta(n \log n)$ operations.
To ENCODE a cycle, each point can be represented by the difference between its cylindrical angle coordinate and that of the succeeding point in the cycle. The other two coordinates are already guaranteed to be equal within ε for all points in a cycle. This requires $\Theta(n)$ operations.
Finally, the CHECK step is exactly the same as that in the 2D case, so the total complexity of the algorithm to find all 2D symmetries of a 3D point set is $\Theta(n \log n)$.

Algorithm 3a: axial symmetry of a polyhedron

Problem 3a. Given an axis and a polyhedron, find the rotational symmetry of the polyhedron about that axis, and find all planes of reflectional symmetry containing that axis.
A general polyhedron is a set of polygon sets (faces) in 3D space such that an edge (p_i, p_j) occurs at most once among all the faces, and if it does occur, then (p_j, p_i) is also an edge of exactly one face. This definition forces the surface to be oriented and closed, but does not rule out self-intersections or disconnected surfaces.

Theorem 3.1. *complexity of polyhedron symmetry testing. For general polyhedra, Problem 3a requires at least $\Omega(n \log n)$ operations.*

Proof. Suppose Problem 3a could be solved in less than $O(n \log n)$ operations. Given a 1D point set (as in Fig. 2a), we could, in linear time, construct a polyhedron (Fig. 2b) with the same symmetries as the point set. Thus, if there were a solution for Problem 3a which took less than $O(n \log n)$ operations, Problem 2a could also be solved faster than $O(n \log n)$. This contradicts Theorem 2.1, and makes $\Omega(n \log n)$ the lower bound on Problem 3a.

The implication of Theorem 3.1 is that, for general polyhedra, no ORDER algorithm can be written that is better than the one described for 3D point sets. However, if we restrict our attention to polyhedra whose surface graphs are connected (Harary 1969), then the ORDER step can be performed in linear time.

Polyhedron ORDER

We begin with the observation that a nontrivial line of symmetry can intersect the surface of a polyhedron in only one of three ways. It may intersect a vertex, the midpoint of an edge, or the centroid of a face. In each case the points topologically adjacent to the point of intersection must be symmetrical about the axis. These vertices will be used to form a cycle Γ_1. If the intersection point is on a face or a vertex, the ordering of the vertices in the cycle can be taken from the clockwise list of adjacent vertices. If the intersection point is on an edge, then there are only two adjacent points, so either ordering will do.
If we define the vertices in Γ_1 to be at graphical distance one from the point of intersection, then all vertices $p_i \notin \Gamma_1$ which are connected by an edge to a vertex $p_j \in \Gamma_1$ are a distance two from the point of intersection, and will form the cycle Γ_2. Similarly, the set of points whose distance in the surface graph from the intersection point is k (i.e., those that are connected by edges to points at distance $k-1$ but not to points at distance less than $k-1$) must also be symmetrical about the axis and will be placed in cycle Γ_k.
The ordering of Γ_1 is known, and the ordering of each subsequent cycle can be deduced from the previous cycle. Each point in Γ_{k+1} is, by definition, edge-connected to some point in Γ_k. These edges define a many-to-many mapping between the points of the cycles. We use geometrical information to distinguish one of these edges for each point in Γ_{k+1}. To do this, we define a function $\Delta(p_j, p_i)$ whose value is the three-tuple of Cartesian coordinates of the point p_i in the coordinate system whose origin is at p_j, whose z-axis is directed parallel to the axis of rotation, and whose y-axis intersects the axis of rotation. This value is unique for all edges adjacent to p_j, and symmetrical points

have exactly the same set of values for their adjacent points. Thus, for each point p_j in Γ_{k+1}, we distinguish the adjacent point p_i in Γ_k, which has the lexicographical minimum value for $\Delta(p_j, p_i)$. This defines a mapping under which each point in Γ_{k+1} maps into exactly one point in Γ_k. The points in Γ_{k+1} are placed in the same order as the corresponding points on Γ_k, with those that map to the same point in Γ_k placed in their clockwise order about that point.

Let $P=\{p_1, p_2, \dots, p_{n-1}\}$ be the vertex set of the polyhedron. Suppose that *succ*(i, j), the index of the clockwise successor of point p_j around point p_i, and *pred*(i, j), the counterclockwise successor of point p_j around p_i, are computable in constant time. Then cycles of symmetrical points about the given axis can be constructed by the following algorithm. This algorithm constructs the m cycles $\Gamma_1, \Gamma_2, \dots, \Gamma_m$ in the correct order in $\Theta(n)$ time.

The following lemmas and theorems lead to a proof that the cycle construction algorithm is linear in complexity and correct.

Lemma 3.2. During the execution of algorithm ORDER 3a, no vertex ever appears in two cycles or more than once in a cycle.

Proof. In each place where a vertex is added to any Γ_i, it is first verified that the vertex was marked UNSEEN before the insertion, and afterwards the UNSEEN mark is removed. The exception to this is step 3b, which only moves the vertex to the end of the list. Thus each vertex is only inserted once.

Theorem 3.3. *complexity of ORDER algorithm. The complexity of algorithm ORDER 3a is linear in the total number of edges E and the total number of vertices V.*

Algorithm ORDER 3a: construction of cycles from connected polyhedron

initialize array cycle[0: $n-1$] := UNSEEN.
initialize array back[0: $n-1$].
initialize linked lists $\Gamma_{1:n}$ empty.

for each p_i which lies on the axis,
  cycle[i] := ONAXIS.

Locate any intersection between the axis and the polyhedral surface.
If the intersection is a vertex p_i,
  for each p_j adjacent to p_i in clockwise order,
    append p_j to Γ_1.
    cycle[j] := 1, back[j] := i.
If the intersection is the midpoint of an edge (p_i, p_j),
  append p_i to Γ_1.
  cycle[i] := 1, back[i] := j.
  append p_j to Γ_1.
  cycle[j] := 1, back[j] := i.
If the intersection is a face,
  for each edge (p_i, p_j) in clockwise order about the face,
    append p_j to Γ_1.
    cycle[j] := 1, back[j] := i.

```
           k := 1
loop 1:  while Γ_k is not empty,
loop 2:      for each point p_i in Γ_k,
                 j := succ(i, back[i]).
loop 3:          while j ≠ back[i]
step 3a:             if cycle[j] = UNSEEN,
                         append p_j to Γ_{k+1}.
                         cycle[j] := k+1. back[j] := i.
```

step 3b: else if cycle[j] $=k+1$ and $\Delta(p_j, p_{\mathrm{back}[j]}) > \Delta(p_j, p_i)$,
delete p_j from Γ_{k+1},
append p_j to Γ_{k+1}.
back[j] $:= i$.
$j := \mathrm{succ}(i, j)$.
$k := k+1$.
$m := k-1$.

Proof. Finding a point where the axis of rotation intersects the surface is accomplished by checking each face, edge, and vertex. The edge and vertex checks each require constant time. The face check is linear in the number of vertices bounding the face, which, when totaled over the polyhedron, add to $2E$. The initialization of Γ_1 requires fewer than V computations.
Loop 2 iterates at most once per vertex. If the algorithm ever iterates on a vertex in Γ_i, that vertex will still be in Γ_i when the algorithm terminates, because that iteration and all subsequent iterations operate only on Γ_j with $j>i$. Thus, if loop 2 iterated more than once on the same vertex, that vertex would appear more than once among the final Γ_k's. But Lemma 3.2 shows this to be impossible. Inner loop 3 iterates at most once per edge adjacent to each vertex, or, in other words, twice on each edge. Thus, the algorithm is linear on V and E.

Theorem 3.4. *correctness of ORDER algorithm. All Γ_i constructed by the polyhedron ORDER algorithm 3a are cycles.*

Proof. Let $\Gamma_k = \langle c_{k,0}, c_{k,1}, \ldots, c_{k,n_k}\rangle$ and let the point's respective back-pointers as given by back[ ] be $\langle b_{k,0}, b_{k,1}, \ldots, b_{k,n_k}\rangle$. It is easily shown that Γ_1 is a cycle in all three cases. It is also easily shown that, if the polyhedron is symmetrical under the rotational transform C and $C(c_{1,i}) = c_{1,j}$, then $C(b_{1,i}) = b_{1,j}$.
We need to show that if Γ_k is a cycle then the Γ_{k+1}, as constructed by the polyhedron ORDER algorithm, is also a cycle. We assume further that, if the polyhedron is symmetrical under C and $C(c_{k,i}) = c_{k,j}$, then $C(b_{k,i}) = b_{k,j}$. Let $K_{k,i}$ be the list of vertices adjacent to $c_{k,i}$ beginning with $b_{k,i}$. These are the points inspected by loop 3. From the above, we can conclude that if $C(c_{k,i}) = c_{k,j}$, then $C(K_{k,i}) = K_{k,j}$. Not all points in $K_{k,i}$ are actually inserted in Γ_{k+1} by loop 3. Points in some Γ_h where $h \leqq k$ are not included. Points which adjoin a previous point in Γ_{k+1} for which the Δ function is larger are not included. Points which adjoin a subsequent point in Γ_{k+1}, for which the Δ function is smaller, are removed. Let $L_{k,i}$ be the points of $K_{k,i}$ which are actually inserted, namely those points p_h at distance $k+1$ from the intersection point for which $\Delta(p_h, c_{k,i})$ is equal to $\min(\Delta(p_h, c_{k,l})$ for $0 \leqq l \leqq n_k$. Thus, the function Δ is defined to be invariant under C, if $C(K_{k,i}) = K_{k,j}$, then $C(L_{k,i}) = L_{k,j}$ and if $C(c_{k,i}) = c_{k,j}$, then $C(L_{k,i}) = L_{k,j}$, the concatination

$$\Gamma_{k+1} = L_{k,0} + L_{k,1} + \ldots + L_{k,m}$$

must satisfy the cycle condition if Γ_k does. Furthermore, the back-pointers for the points in $L_{k,i}$ point to $c_{k,i}$, so if $C(c_{k+1,i}) = c_{k+1,j}$, then $C(b_{k+1,i}) = b_{k+1,j}$.
This completes the induction step, and proves the theorem.

Polyhedron ENCODE

The coordinates of the points can be encoded in a manner similar to that used for point sets. Each point is represented by a three-tuple composed of its cylindrical angle coordinate, the radius coordinate, and the z coordinate. In this case the second two coordinates must be included, because they may differ among points in the same cycle.
In addition, each tuple must contain a list of points which are connected to it by edges of the polyhedron. The adjacent points should be given strictly in clockwise order, so that the locations of the faces can be deduced. They are each represented by the three-tuple $\Delta(p_i, p_j)$. The lists of points are rotated so the point back[i] is given first in each list. This ensures that the list of points is the same for all similar vertices. This encoding can be done in $\Theta(n)$ operations.

Polyhedron CHECK

The encodings produced by the previous step use tuples of variable size to represent different points. To show that it is still possible to run the CHECK algorithm in linear time, we construct a new string from the original and show that it is linear in length. Let $v_{i,1}, v_{i,2}, \ldots, v_{i,n}$ be the elements of the ith tuple. Let M be a value different from any $v_{i,j}$. Consider the string

$$\langle M, v_{1,1}, \ldots, v_{1,n_1}, M, v_{2,1}, \ldots, v_{2,n_2}, \ldots, M, v_{n,1}, \ldots, v_{n,n}\rangle$$

This has the same symmetry as the original string. For each vertex, it contains one M and three point coordinates. For each edge, it contains two three-tuples, one associated with the vertex on each end. Thus the total length is $4V+2E$, which is $\Theta(n)$. We can conclude that the CHECK algorithm still operates in $\Theta(n)$ time.

Algorithm 3b: symmetry of a polyhedron

Problem 3b. Given a polyhedron, whose surface graph is connected and planar, find all involutional and rotational symmetries.

So far, we have considered only symmetry about a gives axis. In this section we will show that all symmetries can be found in linear time. First, the problem of finding all lines of rotational symmetry will be considered, followed by the problem of finding all planes of involutional symmetry.

The possible arrangements of nontrivial lines of symmetry in 3D space are fairly restricted. These are designated as follows:

(k) One k-fold line of symmetry, as in a regular k-sided regular cone

$(2, 2, k)$ One k-fold line of symmetry and k 2-fold lines of symmetry uniformly spaced in the plane perpendicular to the first line, as in a k-sidered regular prism

$(2, 3, 3)$ Four 3-fold lines and three 2-fold lines, arranged as in a regular tetrahedron

$(2, 3, 4)$ Three 4-fold lines, four 3-fold lines and six 2-fold lines, as in a regular octahedron or hexahedron

$(2, 3, 5)$ Six 5-fold lines, ten 3-fold lines and fifteen 2-fold lines, as in a regular dodecahedron or icosahedron

For proof that this list is complete, see Lockwood and Macmillan (1978) or Martin (1982).

For polyhedra whose surface graphs are planar, it is possible to find the symmetry group of the surface graph in linear time by making use of the graph isomorphism algorithm of Hopcroft and Wong (1974). This algorithm finds all isomorphisms between two planar graphs by reducing each graph to either a ring, a skein (the dual of a ring), or one of the graphs corresponding to the surface graph of a Platonic solid. It can be shown that these reductions never destroy a symmetry of the original graph, though (if labeling is ignored) they may create new symmetries. The symmetry group of the surface graph can be derived from the reduced graph, and the vertex, edge, or face intersected by a given line can be found by backtracking the reduction.

The polyhedron can have lines of symmetry only where its surface graph does, but not all symmetries of the surface graph need be symmetries of the polyhedron. In the following three cases, it is shown that once the symmetry group of the surface graph is known, all symmetries of the polyhedron can be found in linear time.

If the symmetry group of the surface graph is (k) for some $k>1$, there is at most one axis, and it must intersect the polyhedral surface in the same place it intersects the surface graph. We can then use the axial symmetry test (algorithm ORDER 3a) to check that axis.

If the symmetry group of the surface graph is one of $(2, 2, 2)$, $(2, 3, 3)$, $(2, 3, 4)$ or $(2, 3, 5)$, then the graph has a finite number of possible lines of symmetry. Therefore, applying the axial symmetry algorithm to each line costs only linear time.

This leaves only the case where the symmetry group of the graph is $(2, 2, k)$ for some $k>2$. Let a be the line corresponding to the k-fold line of symmetry for the graph. Let z_1 and z_2 be the first and last intersections between line a and the surface of the polyhedron (these must be vertices or centroids of edges or faces). Let $\bar{a}$ be the plane perpendicular to a at the midpoint of (z_1, z_2). The actual rotational symmetry of line

a can be tested in linear time with algorithm ORDER 3a. However, applying algorithm 3a to each of the other lines would require a total of $\Theta(n^2)$ operations, since there may be $\Theta(n)$ such lines.

We know from the surface graph that any other lines must be two-fold lines. Any symmetry transform around one of these lines must map the point z_1 into the point z_2, since we know from the surface graph that they cannot be similar to any other points on the polyhedron. All other lines must thus lie in the plane $\bar{a}$, even if the line a is only a one-fold line of symmetry.

Theorem 3.5. *reflection and rotation in polyhedra. If a is a line, and $\bar{a}$ is the plane perpendicular to the line, then for any line b in $\bar{a}$ which intersects a, there exists some angle θ such that $R_{a,\theta} \circ C_{b,2}(P) = P$ if and only if P has a two-fold line of symmetry c in plane $\bar{a}$.*

Proof. If c is a two-fold axis of symmetry for P then

$$P = C_{c,2}(P)$$

Let $\hat{c}$ be the plane containing c and perpendicular to $\bar{a}$. Since it forms a 90° angle with $\bar{a}$, we can write

$$P = Z_{\hat{c},1} \circ Z_{\bar{a},1}(P)$$

Let $\hat{b}$ be the plane containing b and perpendicular to $\bar{a}$. Since all reflection transforms are self-inverse,

$$P = Z_{\hat{c},1} \circ Z_{\hat{b},1} \circ Z_{\hat{b},1} \circ Z_{\bar{a},1}(P)$$

$\hat{c}$ and $\hat{b}$ must both contain a. If they intersect at an angle $\theta/2$, then

$$P = R_{a,\theta} \circ Z_{\hat{b},1} \circ Z_{\bar{a},1}(P)$$

$\hat{b}$ is perpendicular to $\bar{a}$ and both contain line b, so

$$P = R_{a,\theta} \circ C_{b,2}(P)$$

If we assume that this relation holds for some P with no line of symmetry c, the reversal of the argument leads to a contradiction.

Using this theorem, we can find all lines of rotation perpendicular to line a by rotating the polyhedron 180° about any line perpendicular to a and then using the cycle similarity algorithm to find if there are any rotations about a under which the rotated polyhedron is similar to the original. In this way, all k possible lines of symmetry perpendicular to the graph's k-fold line can be tested in linear time.

Once we have the lines of symmetry, it is not difficult to find all involutions under which the polyhedron is symmetrical. We can test for all involutions $Z_{b,k}$ through a plane b by reflecting the object through that plane, and using the cycle similarity algorithm to determine if any rotation about the line perpendicular to b aligns the reflected object with the original object. This test is linear.

Polyhedra with lines in classes (2, 2, 2), (2, 3, 3), (2, 3, 4), or (2, 3, 5) have at most a constant number of possible planes of symmetry, so all can be tested in linear time. Polyhedra with lines in classes (k) and (2, 2, k) may have two types of involutional symmetry. First, there may be involutions through the plane perpendicular to the k-fold line. This plane can be tested as above. Second, there may be k planes of reflectional symmetry which contain the k-fold line. These can be detected with the same algorithm that was used to find lines of symmetry for 2D polygons.

There remains only the case of polyhedra with no lines of rotational symmetry. These may have at most one plane of involutional symmetry. Its location may be guessed from the surface graph's symmetry group as noted in the previous paragraph, or, if the surface graph has rotational symmetry group (1), the location may be proposed by using the graph isomorphism algorithm to find isomorphisms between the surface graph and its reflection.

We find that it is possible to locate all symmetries for a polyhedron with a connected, planar surface graph in linear time. For general polyhedra and 3D point sets we have seen that the axial symmetry algorithm requires $O(n \log n)$ time. To find all symmetries for these objects, we use the surface graph of the convex hull to propose lines. The convex hull can be found in $O(n \log n)$ time (Preparata and Hong 1977). This leads to an $O(n \log n)$ algorithm for these objects.

Unfortunately, the graph isomorphism algorithm of Hopcroft and Wong (1974) is very complicated and has a rather large constant. Although that algorithm could be somewhat simplified for this application, its use may be impractical.

Conclusion

It has been shown that, for polygons and polyhedra with connected, planar surface graphs, all symmetries can be detected in linear time. For point sets and general polyhedra $\Theta(n \log n)$ time is required. The $\Theta(n \log n)$ algorithms can be quite easily extended to a wide variety of geometrical structures without increasing the complexity. All these algorithms have been shown to be optimal.

While the asymptotic behavior of the algorithms is good, the 3D cases share a rather large constant because they require a graph isomorphism test. Thus, the full 3D symmetry algorithms are of primarily theoretical interest. The axial symmetry tests, however, are both practical and useful.

Acknowledgement. The authors would like to thank Professor John H. Remmers for his assistance with one of the theorems in this paper.

References

Aho AV, Hopcroft JE, Ullman JD (1974) The design and analysis of computer algorithms. Addison-Wesley, Reading

Akl SG (1978) Comments on: G. Manacher. An application of pattern matching to a problem in geometrical complexity. Inf Process Lett 7:86

Bykat A (1979) On polygon similarity. Inf Process Lett 9:23–25

Davis LS (1977) Understanding shape: II symmetry. IEEE Systems Man Cybernet 7:204–212

Friedberg SA, Brown CM (1984) Finding axes of skewed symmetry. Proceedings of the IEEE Conference on Pattern Recognition, pp 322–325

Harary F (1969) Graph theory. Addison-Wesley, Reading

Hopcroft JE, Wong JK (1974) Linear time algorithm for isomorphism of planar graphs. Proceedings of the 6th Annual ACM Symposion on Theory of Computing, pp 172–184

Johansen R, Jones N, Clausen J (1984) A method for detecting structure in polyhedra. Pattern Recognition 2:217–225

Knuth DE, Morris JH, Pratt VR (1977) Fast pattern matching in strings. SIAM J Computing 6:323–350

Lee DT, Preparata FP (1984) Computational geometry – a survey. IEEE Trans Comput 33:1072–1101

Lockwood EH, Macmillan RH (1978) Geometric symmetry. Cambridge University Press, Cambridge

Manachar GK (1976) An application of pattern matching to a problem in geometrical complexity. Inf Process Lett 5:6–7

Martin GE (1982) Transform geometry: an introduction to symmetry. Springer, New York

Parvi SK, Dutta Majumder D (1983) Symmetry analysis by computer. Pattern Recognition 16:63–67

Preparata FP, Hong SJ (1977) Convex hulls of finite sets of points in two and three dimensions. Commun ACM 20:87–93

Tanimoto SL (1981) A method for detecting structure in polygons. Pattern Recognition 13:387–394

Wolter JD, Volz RA, Woo TC (1985) Automatic generation of gripping positions. IEEE Trans Systems Man Cybernet (in press)

Received February 25, 1985