Symmetries of Polyhedra: Detection and Applications

X. Y. Jiang, H. Bunke
Institut für Informatik und angewandte Mathematik
Universität Bern, Länggassstrasse 51, 3012 Bern, Switzerland

Abstract

This paper deals with the detection of symmetries of polyhedra and its applications. It consists of two parts. First, we review seven algorithms for symmetry detection of polyhedra. Since these algorithms supply symmetry information in quite different forms, we classify the three most common output forms of the symmetry detection algorithms and discuss their relationships and the transformation from one form into another. For each algorithm, the following five aspects are considered: the output form, the computational complexity, the polyhedra class the algorithm can handle, the implementation, and the suitability for solving the related polyhedral congruity problem. Then, we compare the seven symmetry detection algorithms and give some recommendations as to which algorithm to choose for particular applications. In the second part of this paper we discuss some applications of symmetry information in robotics, geometric problem-solving, and computer vision. The conclusions of this review are twofold. On the one hand, symmetries can be very useful to resolve ambiguities or increase computational efficiency in problem-solving processes involving geometric objects. On the other hand, simple and efficient symmetry detection algorithms are available now. This makes the symmetry exploration a practical issue for many applications.

CR Categories and Subject Descriptors: F.2.2 [Analysis of Algorithms and Problem Complexity]: Nonnumerical Algorithms and Problems; I.3.5 [Computer Graphics]: Computational Geometry and Object Modelling.
General Terms: Algorithms.
Additional Key Words: Polyhedral symmetry, graph theory, graph isomorphism, automaton theory, algorithm comparison, applications.

1 Introduction

Symmetry plays an important role in the world and human beings' perception of the world. In engineering, symmetry has been gaining increasing interest in the design of manufactured parts since it can decrease the cost and increase the utility of the parts. With the advances of computer technology, automatic methods for symmetry analysis have been developed in the last decades. Today algorithms are known for the analysis of rotational and reflectional symmetries of two-dimensional shapes in digital images [23, 26, 27] and three-dimensional objects in range (depth) images [37]. Computational geometry has brought out numerous algorithms for determining the symmetries of many geometric entities, including finite point sets [2, 14], finite sets of line segments [8], polygons [1, 6], and polyhedra. For a survey for finding symmetries of two-dimensional geometric objects, see [9].

In this paper we concentrate on the symmetries of polyhedra. Further we restrict our consideration to rotational symmetry and neglect mirror symmetry. As the latter cannot be realized by any physical movement, it has little relevance to application fields like artificial intelligence, robotics, computer vision, etc. In the present paper, we follow two goals, namely the detection and applications of symmetries of polyhedra. There are a few known algorithms for detecting polyhedral symmetry. However, they are quite scattered in the literature and it is not easy to get access to all of them. One goal of this paper is thus to present an overview of the algorithms for polyhedral symmetry detection. We will compare these algorithms with respect to a number of factors and give some recommendations on choosing algorithms for particular applications. The second part of this paper is devoted to the applications of symmetries. We discuss some examples from robotics, geometric problem-solving, and computer vision that demonstrate the usefulness of symmetries.

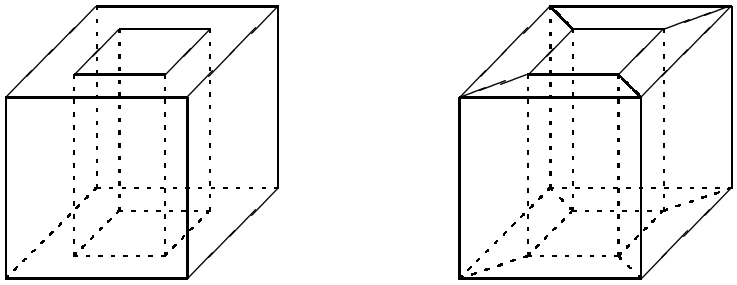
The rest of this paper is organized as follows. After the definition of the polyhedral symmetry detection problem in the next section we review seven algorithms for determining the symmetries of polyhedra in Section 3. These algorithms supply symmetry information in quite different forms. Their relationships and the transformation from one form into another will be discussed in Section 4, followed by some comparisons of the seven symmetry detection algorithms in Section 5. Potential applications of symmetry information in a number of different areas will be desbribed in Section 6. Finally, some conclusions are presented.

2 Polyhedral symmetry detection problem

In our study, the symmetries of polyhedra are defined mainly from an algorithmic point of view so that efficient algorithms can be designed to find them. For a mathematical treatment of polyhedral symmetry in terms of group theory, see for example [3, 25]. A polyhedron is symmetric if there exists a three-dimensional (3D) rotation that doesn't change its shape. The problem of polyhedral symmetry detection is not well defined in the sense that the output of a symmetry detection algorithm may be specified in a number of ways. As a matter of fact, the algorithms reported in the literature generate quite different outputs, depending on the nature of the algorithm and the potential application in mind. In this section we specify the three most common output forms of the symmetry detection algorithms found in the literature.

Before we can discuss the output form, however, we need a formal description of a polyhedron. A polyhedron consisting of n vertices, m edges and h faces can be defined as a graph $G = (V, E, F)$ embedded on the surface of a 3D solid object with

$$V = \{v_1, v_2, \ldots, v_n\},$$



(a) (b)

Figure 1: (a) A polyhedron with cavities in its faces. (b) A polyhedron with one hole.

$$E = \{e_k = (v_{k1}, v_{k2}) \mid k = 1, 2, \ldots, m\},$$
$$F = \{f_k = (v_{k1}, v_{k2}, \ldots, v_{kl_k}) \mid k = 1, 2, \ldots, h\}.$$

In this definition, V represents the set of vertices, E the set of undirected edges, and F the set of oriented faces where each face is represented by the closed chain of its vertices. A chain of vertices is oriented clockwise if we look at the corresponding face from outside of the polyhedron. In our notation a chain of vertices is always cyclic, meaning $(v_{ki}, v_{k,i+1}, \ldots, v_{kl_k}, v_{k1}, v_{k2}, \ldots, v_{k,i-1}) = (v_{k1}, v_{k2}, \ldots, v_{kl_k})$, for $i = 1, 2, \ldots, l_k$. Further we define the set of directed edges E^* by replacing each undirected edge with two directed edges of each direction, i.e.,

$$E^* \;=\; \{(v_{k1}, v_{k2}), (v_{k2}, v_{k1}) \mid (v_{k1}, v_{k2}) \in E\}.$$

Dependent on the actual context, we will use either E or E^* as the representation of the set of edges.

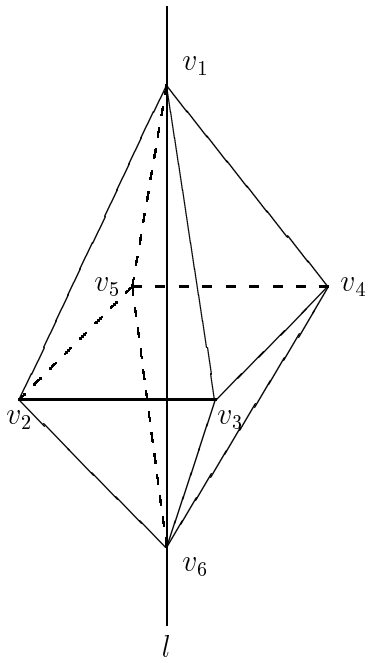
In this paper we consider the class of polyhedra whose graph is connected, thus excluding those polyhedra with cavities in their faces. An example of such a polyhedron is shown in Fig. 1a). Further we distinguish between two classes of polyhedra, P^1 and P^2. The class P^1 contains all polyhedra allowed in our study, i.e., all polyhedra the graph of which is connected, while P^2 consists of those without holes. A polyhedron with one hole is shown in Fig. 1b). It is in class P^1 but not in P^2. For the discussions later, it is worth to mention that the graph of a polyhedron from P^2 is planar while a polyhedron from $P^1 - P^2$, say that in Fig. 1b), has a nonplanar graph.

Given the formal description of a polyhedron, we specify three possible output forms of a symmetry detection algorithm.

Output form 1: A rotational symmetry $Sym(\theta_V, R)$ consists of a topologic and a geometric part. The topologic part is an automorphism of the embedded graph G, i.e., there exists a bijective mapping $\theta_V : V \to V$ such that the following conditions are fulfilled,

$$(v_i, v_j) \in E \Longrightarrow (\theta_V(v_i), \theta_V(v_j)) \;\in E,$$
$$(v_{k1}, v_{k2}, \ldots, v_{kl_k}) \in F \Longrightarrow (\theta_V(v_{k1}), \theta_V(v_{k2}), \ldots, \theta_V(v_{kl_k})) \in F.$$

An additional geometric constraint requires that there exists a spatial rotation, represented by



$$f_1 = (v_2, v_3, v_1)$$
$$f_2 = (v_3, v_4, v_1)$$
$$f_3 = (v_4, v_5, v_1)$$
$$f_4 = (v_5, v_2, v_1)$$
$$f_5 = (v_3, v_2, v_6)$$
$$f_6 = (v_4, v_3, v_6)$$
$$f_7 = (v_5, v_4, v_6)$$
$$f_8 = (v_2, v_5, v_6)$$

Figure 2: An octahedron and its single symmetry axis l.

R, a 3×3 rotation matrix[1], such that

$$v_k \in V \Longrightarrow R \cdot v_k = \theta_V(v_k).$$

I.e., the location of the rotated version $\theta_V(v_k)$ of any vertex v_k must be identical to the location obtained by applying the rotation R to v_k. The output form 1 of a symmetry detection algorithm consists of the set of all symmetries $Sym(\theta_V, R)$. In a similar way we may also define a bijective mapping $\theta_{E^*} : E^* \to E^*$ or $\theta_F : F \to F$, and find out all symmetries $Sym(\theta_{E^*}, R)$ or $Sym(\theta_F, R)$. In this and the next section we will always use the octahedron drawn in Fig. 2 to explain our ideas. It is constructed by stacking two different pyramids together. For the sake of the simplicity of description, the two pyramids are chosen in such a way that none of their triangular faces is equilateral. For this octahedron the set of symmetries $Sym(\theta_V, R)$ contains

$$Sym(\theta_V^1, R_1): \quad \theta_V^1 = \begin{pmatrix} v_1 & v_2 & v_3 & v_4 & v_5 & v_6 \\ v_1 & v_2 & v_3 & v_4 & v_5 & v_6 \end{pmatrix}, \quad R_1 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$Sym(\theta_V^2, R_2): \quad \theta_V^2 = \begin{pmatrix} v_1 & v_2 & v_3 & v_4 & v_5 & v_6 \\ v_1 & v_3 & v_4 & v_5 & v_2 & v_6 \end{pmatrix}, \quad R_2 = \begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$Sym(\theta_V^3, R_3): \quad \theta_V^3 = \begin{pmatrix} v_1 & v_2 & v_3 & v_4 & v_5 & v_6 \\ v_1 & v_4 & v_5 & v_2 & v_3 & v_6 \end{pmatrix}, \quad R_3 = \begin{pmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

[1] A spatial rotation can be represented by a 3×3 rotation matrix only when the rotation axis goes through the origin of the coordinate system in which the object is defined. In general this condition is not satisfied. For a symmetric polyhedron, however, the axis of any symmetry rotation goes through the centroid of the vertex set V. Accordingly, we first transform the coordinate system so that the new origin corresponds to the centroid of V. This transformation leads to a simple representation of symmetry rotations by a 3×3 rotation matrix.

$$Sym(\theta_V^4, R_4): \quad \theta_V^4 = \begin{pmatrix} v_1 & v_2 & v_3 & v_4 & v_5 & v_6 \\ v_1 & v_5 & v_2 & v_3 & v_4 & v_6 \end{pmatrix}, \quad R_4 = \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Here each bijective mapping θ_V^i is represented by a permutation, respectively. Note that $Sym(\theta_V^1, R_1)$ is the trivial identity symmetry.

Output form 2: We define a relation $e_V \subseteq V \times V$,

$$(v_i, v_j) \in e_V \Longleftrightarrow \exists Sym(\theta_V, R)(\theta_V(v_i) = v_j).$$

Clearly, e_V is an equivalence relation. We can thus build the equivalence classes of e_V. All vertices of an equivalence class are equivalent in the sense that some symmetry rotation transforms one into another. Two relations e_{E^*} and e_F can be defined for directed edges and faces in a similar way. The output form 2 of a symmetry detection algorithm consists of all equivalence classes of e_V, e_{E^*}, or e_F. This definition may be extended to higher order relations $e_V^k \subseteq V^k \times V^k$ by defining,

$$((v_{i1}, \ldots, v_{ik}), (v_{j1}, \ldots, v_{jk})) \in e_V^k \Longleftrightarrow \exists Sym(\theta_V, R)(\theta_V(v_{il}) = v_{jl}, l = 1, 2, \ldots, k).$$

Similarly, e_{E^*} and e_F are extended to $e_{E^*}^k$ and e_F^k. Then the output form 2 of a symmetry detection algorithm could be all equivalence classes of e_V^k, $e_{E^*}^k$, or e_F^k. It is even possible to define a mixed relation $e_{VE^*F}^{ijk}$ so that a pair of two tuples of i vertices, j directed edges, and k faces, respectively, are contained in the relation iff there exists a symmetry that transforms each entity of the first tuple into the corresponding entity of the second tuple. A symmetry detection algorithm could also generate all equivalence classes of $e_{VE^*F}^{ijk}$. As an example of this output form, there are three equivalence classes of e_V for the octahedron in Fig. 2,

$$\begin{aligned} &\text{equivalence class 1:} && \{v_1\}, \\ &\text{equivalence class 2:} && \{v_2, v_3, v_4, v_5\}, \\ &\text{equivalence class 3:} && \{v_6\}. \end{aligned}$$

Output form 3: The rotation axis of a rotational symmetry is said to be r-fold if r is the largest natural number such that a rotation of $360/r$ degrees around the axis brings the polyhedron back onto itself. A symmetry axis is always related to a partition of the set V so that each subset $\{v_{i0}, v_{i1}, \cdots, v_{i,l-1}\}$ of the partition has the property that a symmetry rotation of $360/r$ degrees maps v_{ik} to $v_{i,k+l/r}$ (modulus l). The output form 3 of a symmetry detection algorithm is the set of all symmetry axes together with their corresponding fold number r and the partition of V. For the octahedron in Fig. 2, the single symmetry axis l has the fold number four and the set V is partitioned into four disjoint subsets

$$\{v_1\}, \quad \{v_2, v_3, v_4, v_5\}, \quad \text{and } \{v_6\}.$$

With respect to usefulness in practice, the three output forms are quite different. However, they are not independent of each other. In some cases it is possible to derive one output form from another. The relationships and the transformations between the three output forms will be discussed in detail in Section 4.

3 Symmetry detection algorithms

In this section we review seven symmetry detection algorithms known from the literature. From the algorithmic point of view, they are based on graph-theoretic methods, the generate-and-test paradigm, or principles from automata theory. For each algorithm, besides the description of the algorithm itself, the following aspects will be discussed:

- the output form,
- the computational complexity,
- the polyhedra class the algorithm can handle,
- the implementation effort, and
- the suitability for determining the congruity of two polyehdra.

For actual use in practice, not only the output form, the computational complexity and the polyhedra class of a symmetry detection algorithm are crucial, but also the implementation effort is an important factor.

The congruity problem is to decide whether the polyhedra $G_1 = (V_1, E_1, F_1)$ and $G_2 = (V_2, E_2, F_2)$ are identical. Similarly to symmetry, a congruity consists of a topologic and a geometric part. The topologic part means an isomorphism between G_1 and G_2, i.e., a bijective mapping $\theta_V : V_1 \to V_2$ with the property

$$(v_i, v_j) \in E_1 \Longrightarrow (\theta_V(v_i), \theta_V(v_j)) \ \in E_2,$$
$$(v_{k1}, v_{k2}, \ldots, v_{kl_k}) \in F_1 \Longrightarrow (\theta_V(v_{k1}), \theta_V(v_{k2}), \ldots, \theta_V(v_{kl_k})) \in F_2.$$

The geometric constraint requires the existence of a spatial transformation R such that

$$v_k \in V_1 \Longrightarrow R \cdot v_k = \theta_V(v_k) \in V_2.$$

Since the congruity problem and the symmetry detection problem have many aspects in common, it is interesting to see whether a symmetry detection algorithm can be readily modified to solve the congruity problem.

3.1 Graph-theoretic methods

Since a symmetry consists of an automorphism of the graph G (isomorphism of G onto itself) and a three-dimensional rotation, it is not surprising that some of the symmetry detection algorithms are extensions of graph isomorphism algorithms. In its general form the graph isomorphism is a hard problem [13, 29]. No efficient (polynomial) algorithm is known. It has been conjectured that the graph isomorphism problem is NP-complete and thus no polynomial algorithm can exist. For planar graphs, especially triply connected planar graphs, however, low-order polynomial algorithms have been proposed. All symmetry detection algorithms based on graph-theoretic concepts make use of these efficient graph isomorphism algorithms.

3.1.1 The algorithm of Jiang & Bunke

In [19], Jiang and Bunke extended Weinberg's algorithm for determining isomorphisms of triply connected planar graphs [34]. Weinberg's algorithm is based on a fundamental result in graph theory which says that in a finite connected graph it is always possible to construct a cyclic directed path passing through each edge once and only once in each direction. Starting from some directed edge (v_i, v_j), such a path $P(v_i v_j)$ can be generated by the following rules:

1. When a new vertex is reached, take the right-most edge relative to the edge on which the vertex is reached.

2. When an old (already visited) vertex is reached on a new (not yet traversed) edge, go back in the opposite direction.

3. When an old vertex is reached on an old (already traversed) edge, leave the vertex on the right-most edge that has not previously been traversed in that direction.

Weinberg added a numbering scheme to the path generation process described above to obtain a code $C(v_iv_j)$. Each time a new vertex is reached, it is labeled with the next unused natural number. For the octahedron in Fig. 2, some paths and their corresponding codes are

$$P(v_2v_1) = v_2v_1v_3v_2v_3v_6v_2v_6v_5v_2v_5v_1v_5v_4v_1v_4v_3v_4v_6v_4v_5v_6v_3v_1v_2$$
$$C(v_2v_1) = 123134145152526364654321$$

$$P(v_3v_1) = v_3v_1v_4v_3v_4v_6v_3v_6v_2v_3v_2v_1v_2v_5v_1v_5v_4v_5v_6v_5v_2v_6v_4v_1v_3$$
$$C(v_3v_1) = C(v_2v_1)$$

$$P(v_2v_6) = v_2v_6v_5v_2v_5v_1v_2v_1v_3v_2v_3v_6v_3v_4v_6v_4v_5v_4v_1v_4v_3v_1v_5v_6v_2$$
$$C(v_2v_6) = C(v_2v_1)$$

$$P(v_1v_2) = v_1v_2v_5v_1v_5v_4v_1v_4v_3v_1v_3v_2v_3v_6v_2v_6v_5v_6v_4v_6v_3v_4v_5v_2v_1$$
$$C(v_1v_2) = C(v_2v_1)$$

$$P(v_2v_3) = v_2v_3v_6v_2v_6v_5v_2v_5v_1v_2v_1v_3v_1v_4v_3v_4v_6v_4v_5v_4v_1v_5v_6v_3v_2$$
$$C(v_2v_3) = C(v_2v_1)$$

Let $G = (V, E, F)$ be the graph of a polyhedron and (v_i, v_j), (v'_i, v'_j) be two directed edges of G. Based on the results in [34] it was proved in [19] that there exists an automorphism of G (not necessarily being triply connected and planar) such that the vertices v_i and v_j are mapped to v'_i and v'_j, respectively, iff $C(v_iv_j) = C(v'_iv'_j)$. This leads to the following simple algorithm for finding all automorphisms of G:

1. Choose arbitrarily a directed edge (v_i, v_j) of G and compute $P(v_iv_j)$ and $C(v_iv_j)$.

2. For each directed edge (v'_i, v'_j) of G do step 2.1.

2.1. If $C(v_iv_j) = C(v'_iv'_j)$ then there exists an automorphism of G which maps each vertex in $P(v_iv_j)$ to the corresponding vertex in $P(v'_iv'_j)$.

If we assume in the above example (v_2, v_1) to be the reference edge (v_i, v_j), then it is easy to see that the octahedron has automorphisms that map (v_2, v_1) to (v_3, v_1), (v_2, v_6), (v_1, v_2), and (v_2, v_3), respectively. Totally, there exist 24 such automorphisms.

After finding all automorphisms it remains to check whether each automorphism satisfies the geometric condition. This is done by computing the rotation R by the first three corresponding vertices of $P(v_iv_j)$ and $P(v'_iv'_j)$ and verifying the geometric condition for all vertices. In [19] the verification has been incorporated into the path finding and coding process. After the first three vertices of $P(v'_iv'_j)$ have been found, the rotation R is computed. Later on, as soon as a new vertex is generated for $P(v'_iv'_j)$, the geometric condition is tested immediately. In case of failure, the path finding process is stopped and the algorithm considers the next edge (v'_i, v'_j). Among the 24 automorphisms of the octahedron, four satisfy the geometric condition, corresponding to the paths $P(v_2v_1)$, $P(v_3v_1)$, $P(v_4v_1)$, and $P(v_5v_1)$. Thus, there are totally four symmetries, as shown in Section 2.

The algorithm output is the set of all symmetries $Sym(\theta_V, R)$. The algorithm has a time complexity of $O(m^2)$ and requires $O(m)$ storage. It can handle polyhedra of class P^1. This algorithm can be easily implemented. As a matter of fact, the Pascal implementation reported in [19] consists of only about 120 lines of code.

The modification of this algorithm to solve the congruity problem is straightforward. Given two polyhedra G_1 and G_2, the following algorithm generates all isomorphisms between G_1 and G_2:

1. Choose arbitrarily a directed edge (v_i, v_j) of G_1 and compute $P(v_iv_j)$ and $C(v_iv_j)$.

2. For each directed edge (v'_i, v'_j) of G_2 do step 2.1.

2.1. If $C(v_iv_j) = C(v'_iv'_j)$ then there exists an isomorphism which maps each vertex in $P(v_iv_j)$ to the corresponding vertex in $P(v'_iv'_j)$.

The geometric condition can be tested in the same way as for symmetry detection. The isomorphisms surviving the geometric test build the set of all congruity mappings that cause G_1 to coincide with G_2.

3.1.2 The algorithm of Sugihara

The algorithm of Sugihara is an extension of the graph isomorphism algorithm of Hopcroft and Tarjan [15]. For the description of the algorithm we need some definitions. A path of directed edges $(e_1, e_2, \cdots, e_n)$ is said to be *primary* if e_{i+1} is either to the immediate right or to the immediate left of e_i, i.e., there must not be any edge between e_i and e_{i+1}. Two primary paths $(e_1, e_2, \cdots, e_n)$ and $(e'_1, e'_2, \cdots, e'_n)$ are said to be *corresponding* if e_ie_{i+1} and $e'_ie'_{i+1}$, $i = 1, 2, \ldots, n-1$, have the same turn direction. Each directed edge e gets associated a vector $\lambda(e)$ of four geometric features,

$$\lambda(e) = (l(e), \psi(e), \theta_L(e), \theta_R(e))$$

where $l(e)$ is the length of e, $\psi(e)$ is the angle between the two faces on the left and right side of e, and $\theta_L(e)$ ($\theta_R(e)$) represents the angle between e and the edge immediately left (right) of e. Two edges e_1 and e'_1 are said to be *indistinguishable* if any primary path $(e_1, e_2, \cdots, e_n)$ starting from e_1 and the corresponding path $(e'_1, e'_2, \cdots, e'_n)$ starting from e'_1 satisfy $\lambda(e_i) = \lambda(e'_i), 1 \leq i \leq n$. The algorithm for symmetry detection is based on a theorem which says that there exists a symmetry mapping that maps e to e' iff e and e' are indistinguishable. Hence, two edges e and e' are equivalent under symmetry iff they are indistinguishable.

The algorithm of Sugihara divides the set of directed edges E^* into equivalence classes of e_{E^*}. Clearly, a necessary but not sufficient condition for two edges e and e' being indistinguishable and thus belonging to the same equivalence class is $\lambda(e) = \lambda(e')$. Consequently, the symmetry detection algorithm begins with an initial partition of E^* into subsets satisfying the necessary condition and partitions the subsets further into sets of mutually indistinguishable edges. In more detail the algorithm can be described as follows:

1. Compute $\lambda(e)$ for all directed edges $e \in E^*$.

2. Divide E^* into subsets $B_1, B_2, \cdots, B_k$ in such a way that e and e' belong to the same subset iff $\lambda(e) = \lambda(e')$.

3. If there exists $i, 1 \leq i \leq k$, such that B_i is a singleton, then conclude that the polyhedron P is not symmetric and terminate the algorithm.

4. Partition each $B_1, B_2, \cdots, B_k$ further into sets consisting of mutually indistinguishable edges. Let the resulting sets be $B'_1, B'_2, \cdots, B'_l, l \geq k$.

5. If B'_1 is a singleton, then P is not symmetric. Otherwise P is symmetric and the sets B'_i correspond to the equivalence classes of e_{E^*}.

Steps 3 to 5 need some explanation. Because a nontrivial symmetry always maps a directed edge into another directed edge, an equivalence class of e_{E^*} can never be a singleton. Accordingly, if some set after the initial partition in step 2 is a singleton, the algorithm terminates immediately. For the details of step 4 that partitions $B_1, B_2, \cdots, B_k$ into sets of mutually indistinguishable edges, see [32]. Finally, the singleton test in step 5 is based on the same reasoning as for step 3.

For the octahedron in Fig. 2, the initial partition in step 2 produces the following six sets,

$$B_1 = \{(v_1, v_2), (v_1, v_3), (v_1, v_4), (v_1, v_5)\},$$

$$B_2 = \{(v_2, v_1), (v_3, v_1), (v_4, v_1), (v_5, v_1)\},$$

$$B_3 = \{(v_2, v_3), (v_3, v_4), (v_4, v_5), (v_5, v_2)\},$$

$$B_4 = \{(v_3, v_2), (v_4, v_3), (v_5, v_4), (v_2, v_5)\},$$

$$B_5 = \{(v_6, v_2), (v_6, v_3), (v_6, v_4), (v_6, v_5)\},$$

$$B_6 = \{(v_2, v_6), (v_3, v_6), (v_4, v_6), (v_5, v_6)\}.$$

The final partition in step 4 doesn't change anything. So the six sets above are the equivalence classes of e_{E^*}.

The algorithm of Sugihara generates the equivalence classes of e_{E^*}. The algorithm needs $O(mlogm)$ time and $O(m)$ storage. Similar to the algorithm of Jiang & Bunke, it can handle polyhedra of class P^1. The implementation of this algorithm is relatively easy.

Originally, Sugihara has developed his algorithm for determining the congruity of polyhedra. In fact, the algorithm above is a modification of the formulation in [32]. The original algorithm for congruity determination is as follows:

1. Compute $\lambda(e)$ for all directed edges $e \in E_1^* \cup E_2^*$.

2. Divide $E_1^* \cup E_2^*$ into subsets $B_1, B_2, \cdots, B_k$ in such a way that e and e' belong to the same subset iff $\lambda(e) = \lambda(e')$.

3. If there exists $i, 1 \leq i \leq k$, such that $B_i \subseteq E_1^*$ or $B_i \subseteq E_2^*$, then conclude that the polyhedra P_1 and P_2 are not congruent and terminate the algorithm.

4. Partition each $B_1, B_2, \cdots, B_k$ further into sets consisting of mutually indistinguishable edges. Let the resulting sets be $B'_1, B'_2, \cdots, B'_l, l \geq k$.

5. P_1 and P_2 are congruent iff $B'_1 \cap E_1^* \neq \emptyset$ and $B'_1 \cap E_2^* \neq \emptyset$.

In the final partition, $e_1, e_2 \in B'_i, e_1 \in E_1^*, e_2 \in E_2^*$ means the existence of a spatial transformation that maps e_1 of P_1 to e_2 of P_2.

3.2 Generate-and-test paradigm

One of the most popular paradigms in algorithm design is that of generate-and-test. For symmetry detection, this class of algorithms can be further divided into two categories, i.e., generation of symmetry axes and generation of spatial rotations. In this section we describe three algorithms of this class, two from the first and one from the second category.

3.2.1 The algorithm of Waltzman

The algorithm of Waltzman [33] is based on the observation that any symmetry axis of a polyhedron $G = (V, E, F)$ without holes passes through at least two elements of the set $V \cup E \cup F$. Here we want to emphasize that this is not true for polyhedra with holes, see Fig. 3 for an example. Totally, there exist six possible combinations. The symmetry axis may pass through:

1. two faces,
2. a face and a vertex,
3. a face and an edge,
4. two vertices,
5. a vertex and an edge, or
6. two edges.

Thus, hypotheses of symmetry axes can be generated by exhaustively enumerating all the six combinations. Essentially, Waltzman follows this approach. In his algorithm three procedures were given to separately consider three different cases: symmetry axes which pass through at least one face (combinations 1-3), symmetry axes which pass through at least one vertex (4-5), and symmetry axes which pass through two edges (6). In each procedure, constraints are tested first to filter out obviously impossible hypotheses. If a symmetry axis l passes through face f of P, for instance, then f, considered as a polygon, must be symmetric. In addition, l must be perpendicular to f and intersect f at its centroid. Another constraint for the existence of a symmetry axis passing through two edges is that the number of edges of P is divisible by 2. Each hypothesized symmetry axis surviving these tests is then verified in a second stage to see whether it is a true symmetry axis.

From each hypothesis it is possible to partition the set of vertices V into a number of disjoint subsets, called cycles, such that, for each cycle $\{v_0, v_1, \cdots, v_{l-1}\}$, if the hypothesis is a true symmetry and the symmetry maps v_i to v_j, then it also maps v_{i+k} to v_{j+k} (modulus l) for all k. Obviously, a necessary condition for the existence of a true symmetry is that each cycle is locally symmetric with respect to the hypothesized symmetry axis. This local symmetry is tested for each cycle and the overall symmetry of the polyhedron is determined by combining the local symmetries. For the octahedron, only the hypothesis l as shown in Fig. 2 will be successfully verified.

The algorithm provides the set of all symmetry axes and their corresponding fold number and partition. It has a time complexity of $O(eh^2)$ where e is the maximum number of edges of any of the faces of the polyhedron. The storage requirement is linear. Apparently, the algorithm can only handle polyhedra of class P^2 since the fundamental observation concerning the symmetry axes assumes polyhedra without holes. Although the idea behind the algorithm is simple, the algorithm itself is involved and lengthy. An actual implementation seems tedious.

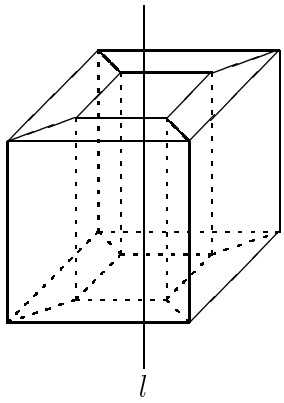


Figure 3: A symmetric polyhedron whose symmetry axis l doesn't go through any vertex, edge, or face.

The algorithm of Waltzman cannot be extended to solve the congruity problem. The reason is that the hypotheses generation makes use of a property of symmetric polyhedra. Since two congruent polyhedra may have no symmetry at all, this first stage will fail.

3.2.2 The optimal algorithms of Wolter et al.

Wolter et al. proved in [35] that the computational complexity of the symmetry detection problem has a lower bound of $O(n)$ for polyhedra of class P^2 and $O(nlogn)$ for polyhedra of class P^1. For each polyhedra class they also constructed an optimal algorithm which reaches the lower bound. Both algorithms generate in a first stage hypotheses of symmetry axes. Each hypothesis is then verified to see whether it is really a symmetry axis of the polyhedron.

For polyhedra whose graphs are planar, it is possible to find the symmetry group of the graph in linear time by making use of a technique from the graph isomorphism algorithm of Hopcroft and Wong [16]. It reduces a planar graph to either a ring, a skein (a graph consisting of two vertices u and v, and k edges each of which being incident at both u and v), or one of the graphs corresponding to the Platonic solids. It can be shown that these reductions never destroy any of the symmetries of the original graph, although they may create new symmetries. The symmetry group of the original graph can then be derived from the reduced graph. Since the possible symmetry groups of a graph are fairly restricted and each symmetry group has only a finite number of possible symmetry axes, a finite set of hypotheses of symmetry axes can be generated from the symmetry group found by the linear time graph isomorphism algorithm.

A polyhedron can have symmetry axes only where its graph does, but not all symmetries of the graph need be symmetries of the polyhedron. Therefore each symmetry axis hypothesis is verified in a similar way as in [33] to see whether it is a true symmetry axis of the polyhedron.

Since the linear time graph isomorphism algorithm of Hopcroft and Wong is useful only for polyhedra with planar graphs, the symmetry detection algorithm above must be modified in order to find the symmetries of general polyhedra. Wolter et al. [35] used the graph of the convex hull of the original nonplanar graph to hypothesize the symmetry axes. The verification step is identical to that described above.

The two optimal algorithms provide the set of all symmetry axes and their corresponding fold number and partition. The version for polyhedra of class P^2 needs $O(n)$ computation time and that for general polyhedra $O(nlogn)$ time. Because the linear time graph isomorphism algorithm is very complicated, the two optimal algorithms are very involved and an implementation seems

difficult. The authors stated "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 contribution of the work reported in [35] lies certainly in the establishment of the lower bound for the symmetry detection problem.

The two optimal algorithms cannot be extended to solve the congruity problem for similar reasons as the algorithm of Waltzman.

3.2.3 The algorithm of Flynn

The key idea behind the the algorithm of Flynn [12] is that given a symmetry, the symmetry rotation can be uniquely determined by two faces and their corresponding faces under the symmetry. Therefore, we can select a reference pair of faces and compare that pair against all other compatible pairs, each leading to a symmetry hypothesis with the related spatial rotation. Each hypothesis is then verified by checking whether for all other faces there exists a corresponding face under the hypothesized symmetry rotation. A pseudo-code description of this algorithm is as follows.

```
        Select a reference face pair (f_i, f_j);
Loop:   FOR each possible face pair (f_k, f_l) DO
          IF COMPATIBLE((f_i, f_j), (f_k, f_l)) THEN
            Compute rotation R that maps (f_i, f_j) to (f_k, f_l);
            Add θ_F(f_i)=f_k and θ_F(f_j)=f_l to a possible new symmetry Sym(θ_F, R);
            /* Verify the hypothesis ((f_i, f_j), (f_k, f_l)) with related R */
            FOR each f_a ∈ F − {f_i, f_j} DO
              IF ∄f_b(f_b ∈ F ∧ Rf_a = f_b) THEN
                Goto loop to generate next hypothesis
              ELSE
                Add θ_F(f_a) = f_b to Sym(θ_F, R)
              ENDIF
            ENDFOR
            /* The hypothesis has been successfully verified */
            Output the new symmetry Sym(θ_F, R);
          ENDIF
        ENDFOR
```

In this formulation a predicate COMPATIBLE has been used. It takes two face pairs (f_i, f_j) and (f_k, f_l) as arguments and returns TRUE if corresponding faces in the pairs, i.e., (f_i, f_k) and (f_j, f_l), can be exactly aligned to each other (identical in size and form).

For the octahedron in Fig. 2, if we take (f_1, f_5) as the reference pair, the algorithm will generate four hypotheses: $((f_1, f_5), (f_1, f_5))$, $((f_1, f_5), (f_2, f_6))$, $((f_1, f_5), (f_3, f_7))$, and $((f_1, f_5), (f_4, f_8))$. All the four hypotheses will be successfully verified. Thus, the algorithm will output four symmetries:

$$Sym(\theta_F^1, R_1): \quad \theta_F^1 = \begin{pmatrix} f_1 & f_2 & f_3 & f_4 & f_5 & f_6 & f_7 & f_8 \\ f_1 & f_2 & f_3 & f_4 & f_5 & f_6 & f_7 & f_8 \end{pmatrix}, \quad R_1 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$Sym(\theta_F^2, R_2): \quad \theta_F^2 = \begin{pmatrix} f_1 & f_2 & f_3 & f_4 & f_5 & f_6 & f_7 & f_8 \\ f_2 & f_3 & f_4 & f_1 & f_6 & f_7 & f_8 & f_5 \end{pmatrix}, \quad R_2 = \begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$Sym(\theta_F^3, R_3): \quad \theta_F^3 = \begin{pmatrix} f_1 & f_2 & f_3 & f_4 & f_5 & f_6 & f_7 & f_8 \\ f_3 & f_4 & f_1 & f_2 & f_7 & f_8 & f_5 & f_6 \end{pmatrix}, \quad R_3 = \begin{pmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$Sym(\theta_F^4, R_4): \quad \theta_F^4 = \begin{pmatrix} f_1 & f_2 & f_3 & f_4 & f_5 & f_6 & f_7 & f_8 \\ f_4 & f_1 & f_2 & f_3 & f_8 & f_5 & f_6 & f_7 \end{pmatrix}, \quad R_4 = \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

The output of the algorithm is the set of all symmetries $Sym(\theta_F, R)$. The algorithm has a computational complexity of $O(h^4)$. The storage requirement is linear. It can handle polyhedra of class P^1. An actual implementation of this algorithm is relatively easy.

The modification of Flynn's algorithm to solve the congruity problem is straightforward. We simply select the reference face pair (f_i, f_j) from one of the two polyhedra and compare that pair against all face pairs (f_k, f_l) of the other polyhedron.

3.3 The polyhedral automaton approach

In their approach to recognizing polyhedra, Johansen et al. [22] represented a model polyhedron P by an automaton. For the recognition, a partial polyhedron P' observed in the scene is coded by a string. Then P' is identified as an instance of P iff this string is accepted by the automaton. As a byproduct of this approach, the symmetry of a polyhedron can be found by checking a simple property of its automaton.

The model polyhedron P is represented by a sequence of faces $S = f_{i1} f_{i2} \cdots f_{ik}$ where f_{ij} and $f_{i,j+1}$ are adjacent and each face of P occurs at least once in this sequence. For the coding of S, all possible reference polygons that appear as faces in P are identified and the edges of each reference polygon are uniquely numbered. The octahedron in Fig. 2, for example, has two reference polygons as shown in Fig. 4. In addition all possible transitions from one face to another are also identified and assigned a label. A transition (e_1, α, e_2, p) means that the traversal goes from the current face f_{ij} to a new face $f_{i,j+1}$ corresponding to the reference polygon p by crossing an edge that is numbered e_1 for f_{ij} and e_2 for $f_{i,j+1}$, respectively. Moreover, the two faces f_{ij} and $f_{i,j+1}$ include an angle α. For the octahedron there exist six transitions as shown in Fig. 4.

Given the coding alphabet consisting of the set of reference polygons A_p and the set of transitions A_t, the face sequence S is actually coded by a string

$$S \;=\; p t_1 t_2 \cdots t_{k-1}$$

where $p \in A_p$ is the reference polygon related to the first face f_{i1} and $t_i \in A_t, i = 1, 2, \cdots, k-1$, corresponds to the transition from f_{ij} to $f_{i,j+1}$. As an example, the sequence $S = f_1 f_2 f_6 f_5 f_1$ is one possible representation of the visible part of the octahedron. This sequence will be coded by $S = p_1 aedf$.

A non-deterministic automaton can be constructed to accept all strings from $A_p A_t \cdots A_t = A_p (A_t)^*$. The automaton for the octahedron is shown in Fig. 5 where s_0 is a special start node and each node $s_i, i \geq 1$, represents the fact that the current face corresponds to the face f_i of the model. Finally, the non-deterministic automaton can be converted into an equivalent deterministic polyhedral automaton by means of the standard technique, i.e., the subset construction. For the recognition, an unknown polyhedron P' is coded by a string $S \in A_p (A_t)^*$.

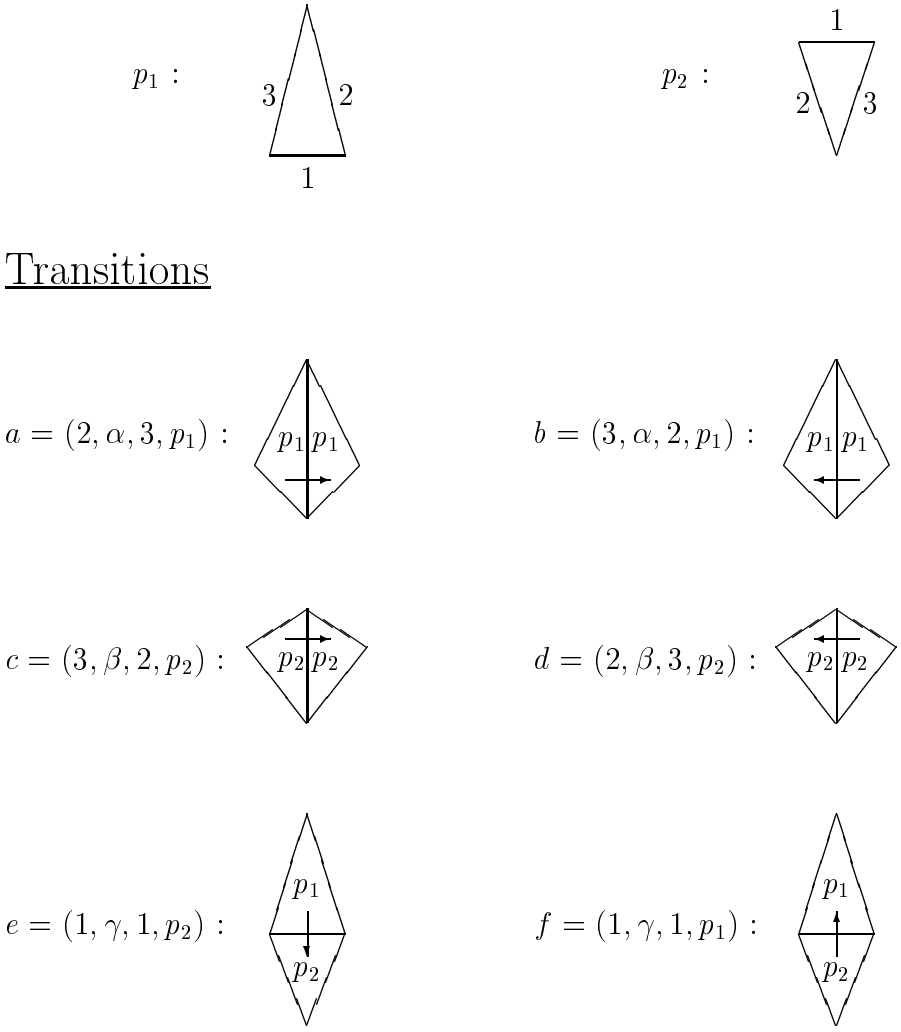


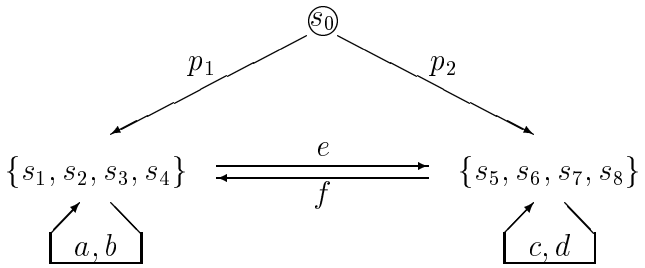
Figure 4: The coding alphabet for the octahedron.

P' is considered as an instance of a model polyhedron P iff S is accepted by the polyhedral automaton of P. It can be easily verified that $S = p_1aedf$ representing the visible part of the octahedron is accepted by the polyhedral automaton of the octahedron shown in Fig. 5.

It was proved in [22] that a polyhedron is symmetric iff its polyhedral automaton does not contain singleton states. For polyhedra with no symmetric faces, the states of the polyhedral automaton represent the equivalence classes of e_F. If there is some symmetric face, the polyhedral automaton will not immediately generate the equivalence classes of e_F. This kind of equivalence classes, however, can be easily derived from the automaton states. While the symmetry condition is very easy to test, the construction of the polyhedral automaton has unfortunately an exponential complexity. Thus, the number of its states cannot be limited by a polynomial function of the number of faces. The automaton method can handle polyhedra of class P^1 and has a medium level of implementation complexity.

The automaton method can be extended to solve the congruity problem. Given two polyhedra P and P' and their polyhedral automata A and A', respectively, we can consider A and A' as attributed directed graphs. P and P' are congruent iff they have the same coding alphabet

Deterministic polyhedral automaton



Non-deterministic automaton

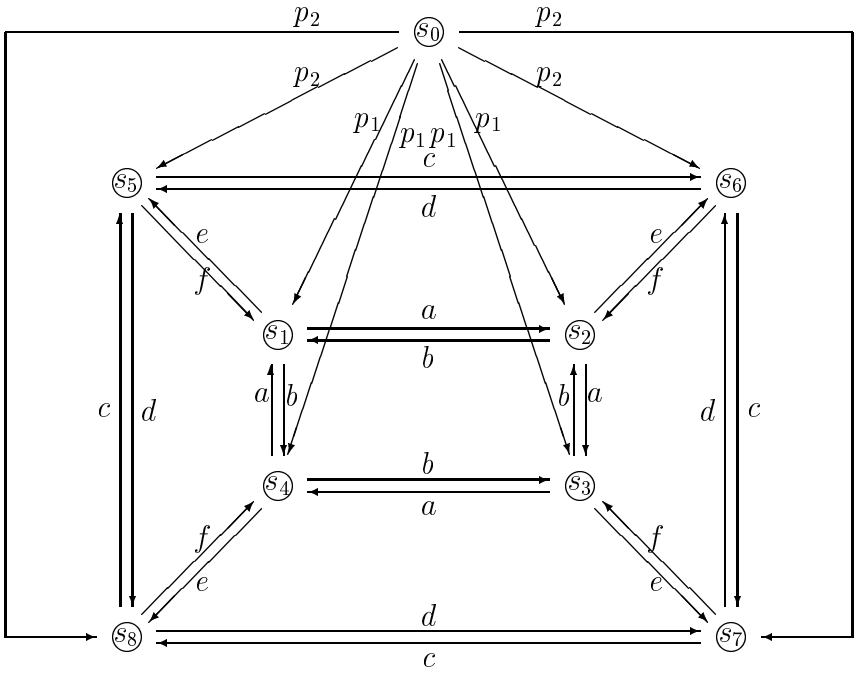
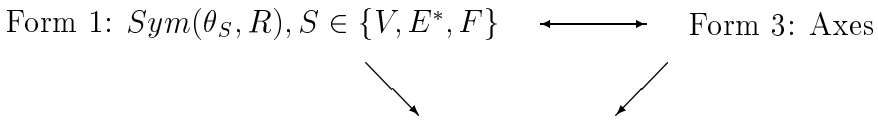


Figure 5: The non-deterministic and the deterministic polyhedral automaton of the octahedron.



Form 2: Equivalence classes

Figure 6: The relationships between the three forms of symmetry information.

and there exists a bijective mapping from the state set of A to that of A' so that the two attributed graphs A and A' are isomorphic. Since graph isomorphism test is a difficult problem, this method for congruity determination is certainly less practical than the methods described in Sections 3.1.1 and 3.1.2.

4 Relationships between the output forms

In Section 2 we have classified three possible output forms of a symmetry detection algorithm. The seven algorithms described in the last section provide symmetry information in one of the three forms. With respect to the usefulness, the three output forms are fairly different. It seems that symmetry mappings $Sym(\theta_S, R)$ and equivalence classes of $e_S, S \in \{V, E^*, F\}$, are the most useful representation of symmetries for practical usage. In this section we investigate the relationships between the three forms of symmetry information. Such an investigation allows us to determine the usefulness of a symmetry detection algorithm for a given application. Assume that the algorithm provides symmetry information in form X and the application needs it in form Y. Then, for this particular application, the algorithm makes only sense if Y can be directly derived from X.

In this section we will show that the three forms of symmetry information are not independent. Their relationships are shown in Fig. 6, where $X \rightarrow Y$ means that X can be transformed into Y. These transformations will be discussed in the following subsections.

4.1 Equivalence of form 1 and form 3

Given all symmetry mappings $Sym(\theta_V, R)$, the symmetry axes as well as their corresponding fold number and partition can be easily computed. Consider one symmetry given by $\theta(v_i) = v_k, v_i \neq v_k$ and $\theta(v_j) = v_l, v_j \neq v_l$, its symmetry axis must lie in a plane that is perpendicular to the straight line segment $\overline{v_i v_k}$ and passes through the middle point of $\overline{v_i v_k}$. The same observation is true for the pair (f_j, f_l). The intersection of the two planes must be the symmetry axis.

The transformation from form 3 to form 1 is easy. Given a r-fold symmetry axis l and its corresponding partition, there are r symmetry mappings. For each vertex v_{ik} in a subset $\{v_{i0}, v_{i1}, \cdots, v_{i,l-1}\}$ of the partition, these symmetry mappings map it to $v_{i,k+j \cdot l/r}$ (modulus l), $1 \leq j \leq r$, respectively. Considering all symmetry axes gives the set of all symmetry mappings.

4.2 Transformation between form 2 and other forms

Due to the equivalence of form 1 and 3, we consider here only the transformation between form 1 and 2. Given the symmetry mappings $Sym(\theta_S, R), S \in \{V, E^*, F\}$, the equivalence relations

e_S^k are uniquely defined. Thus, their corresponding equivalence classes can also be inferred. An algorithm for actually doing the transformation from form 1 to form 2 will be given in Section 4.3.

The inverse transformation from form 2 into form 1 is not possible. That is, we cannot directly reconstruct the symmetry mappings from the equivalence classes. Certainly, we could apply some algorithm X that finds out the symmetry mappings from the initial representation of the polyhedron. In this case, however, X itself is a symmetry detection algorithm. If symmetry mappings are actually needed, we can use the algorithm X directly without involving the symmetry detection algorithm that generates the equivalence classes at all.

4.3 Construction of equivalence classes

In practice, equivalence classes are a very useful representation of symmetries. Thus, construction of equivalence classes from the other two output forms is an important task. Due to their equivalence, however, we will only consider the case with the symmetry mappings.

In [19] Jiang and Bunke proposed an algorithm for the construction of equivalence classes of e_V from the mapping $Sym(\theta_V, R)$. The modification for $e_{E^*}^k$ or e_F^k is straightforward.

The construction task is to generate all equivalence classes of e_V from the set L of all symmetry mappings $Sym(\theta_V, R)$. The construction algorithm in [19] scans L once and incrementally register e_V by each mapping in L. For each equivalence class

$$\{v_{j1}, v_{j2}, \cdots, v_{jk}\}, \quad j1 < j2 < \cdots < jk,$$

it records the equivalence relationships

$$(v_{j1}, v_{ji}) \in e_V, \ i = 2, 3, \cdots, k.$$

This proceeds as follows. We use an array $rec[1..n]$ which is initialized with zeros. The ith element of rec represents v_i. From each symmetry mapping θ_V we get a new set of pairs $(v_i, v_j) \in e_V, i < j$. For each of them, the array entry j is updated by

$$\textbf{IF } (rec[j] = 0) \textbf{ OR } (i < rec[j]) \textbf{ THEN } rec[j] := i$$

After all symmetry mappings have been processed, the array rec will look like

$$rec[l] = \begin{cases} 0, & \text{if } v_l = v_{j1} \\ j1, & \text{if } v_l = v_{ji}, \ i = 2, 3, \ldots, k \end{cases}$$

Thus, one scanning of rec will produce all equivalence classes of e_V.

The octahedron in Fig. 2 has four symmetry mappings as shown in Section 2. To illustrate how the construction algorithm works for this polyhedron, we show in the first table of Fig. 7 the content of array rec after the processing of each symmetry mapping. The final content of rec reveals four equivalence classes: $\{v_1\}$, $\{v_2, v_3, v_4, v_5\}$, and $\{v_6\}$. The modification of the construction algorithm above to handle e_{E^*} or e_F is straightforward and is thus not given here. As an example for these two relations, let's consider the four symmetry mappings $Sym(\theta_F, R)$ for the octahedron shown in Section 3.2.3. The behavior of the algorithm for this relation is shown in the second table of Fig. 7. Finally, two eqivalence classes $\{f_1, f_2, f_3, f_4\}$ and $\{f_5, f_6, f_7, f_8\}$ will be found.

The extension of the construction algorithm to the general case $e_V^k, k \geq 1$, is straightforward. For a full formulation of the general version, see [20].

Mapping	Set of new pairs $(v_i, v_j) \in e_V, i < j$	Array rec
initial		(0,0,0,0,0,0)
θ_V^1	$\emptyset$	(0,0,0,0,0,0)
θ_V^2	$\{(v_2, v_3), (v_3, v_4), (v_4, v_5)\}$	(0,0,2,3,4,0)
θ_V^3	$\{(v_2, v_4), (v_3, v_5)\}$	(0,0,2,2,3,0)
θ_V^4	$\{(v_2, v_5)\}$	(0,0,2,2,2,0)

Mapping	Set of new pairs $(f_i, f_j) \in e_F, i < j$	Array rec
initial		(0,0,0,0,0,0,0,0)
θ_F^1	$\emptyset$	(0,0,0,0,0,0,0,0)
θ_F^2	$\{(f_1, f_2), (f_2, f_3), (f_3, f_4), (f_5, f_6), (f_6, f_7), (f_7, f_8)\}$	(0,1,2,3,0,5,6,7)
θ_F^3	$\{(f_1, f_3), (f_2, f_4), (f_5, f_7), (f_6, f_8)\}$	(0,1,1,2,0,5,5,6)
θ_F^4	$\{(f_1, f_4), (f_5, f_8)\}$	(0,1,1,1,0,5,5,5)

Figure 7: The behavior of the equivalence class construction algorithm for the octahedron.

5 Comparison of symmetry detection algorithms

In the previous sections we have briefly described seven symmetry detection algorithms found in the literature[2]. For each algorithm the following five aspects have been considered: the output form, the computational complexity, the polyhedra class the algorithm can handle, the implementation complexity, and the suitability for solving the related congruity problem. Since the algorithms supply symmetry information in different forms we have classified three possible output forms of a symmetry detection algorithm and discussed their relationships. In this section we compare the seven symmetry detection algorithms and try to give some recommendations as to which algorithm to choose for a particular application.

The seven algorithms are summarized in Table 1. The first four features in the table, i.e., columns two to five, play an important role in selecting an appropriate algorithm for a particular application. A high execution speed is always desired. However, the O-notation for the computational complexity should be considered with caution because it describes merely the asymptotic behavior of an algorithm and neglects constant factors and additive constants. The algorithms WWV1 and WWV2, for instance, are both optimal in the sense that their asymptotic complexity reaches the lower bound of the symmetry detection problem. Since they have a very large constant, however, they are likely to be more time-consuming for relatively small n values than other algorithms with a higher asymptotic complexity. The output form of a symmetry detection algorithm is another important decision factor for algorithm selection. It makes only sense to use an algorithm if it supplies symmetry information in the desired form directly, or if the desired form can be easily derived. Among the three output forms, the symmetry mapping $Sym(\theta_S, R), S \in \{V, E^*, F\}$, and the equivalence classes are the most useful symmetry representations for a variety of applications. The symmetry mappings and the symmetry axes representations are equivalent. Equivalence classes of the relation e_S^k can be easily constructed from the other two representations. Which polyhedra class a symmetry detection algorithm can handle also determines its usefulness. In most cases the more general polyhedra class P^1 is desired. Finally, the implementation complexity of an algorithm should

[2]An earlier version of the algorithm of Jiang & Bunke and Flynn's algorithm, reported in [18] and [11], respectively, has not been discussed in this paper.

Algorithm	Complexity	Output	Class	Implementation	Congruity
Jiang & Bunke (JB)	$O(m^2)$	$Sym(\theta_V, R)$	P^1	easy	yes
Sugihara (SU)	$O(mlogm)$	EC of e_{E^*}	P^1	easy	yes
Waltzman (WA)	$O(eh^2)$	Axes	P^2	medium	no
Wolter et al. (WWV1)	$O(n)$	Axes	P^2	difficult	no
Wolter et al. (WWV2)	$O(nlogn)$	Axes	P^1	difficult	no
Flynn (FL)	$O(h^4)$	$Sym(\theta_F, R)$	P^1	easy	yes
Johansen et al. (JJC)	Exponential	EC of e_F	P^1	medium	yes

Table 1: Summary of the seven symmetry detection algorithms.

not be ignored.

According to the criteria formulated above, the algorithms WWV1, WWV2, and JJC are less suitable for practical applications. The two optimal algorithms of Wolter et al., as stated by the authors themselves, are of primarily theoretical interest. They illustrate that the lower bound of the symmetry detection problem, $O(n)$ for the polyhedra class P^2 and $O(nlogn)$ for P^1, can actually be reached. An extremly difficult implementation hinders the two algorithms from practical usage. The algorithm JJC has the drawback of exponential computational complexity.

Among the other four algorithms, the algorithms JB and SU seem to be superior. The algorithm SU has an optimal computational complexity and its actual implementation is relatively easy. A potential problem with this algorithm is that it provides the equivalence classes of the relation e_{E^*}. This excludes its usage, for instance, in those situations where the symmetry mappings are needed. The algorithm JB has the advantage of implementation ease and generality of its output. It has a quadratic complexity in the worst case. As stated in [19], however, the average case occurring in practical applications can be expected to be substantially better.

With respect to the computational complexity and the output form, the algorithm WA is comparable to JB. An actual implementation of WA, however, seems tedious. A more serious handicap of this algorithm is that it cannot find symmetries of polyhedra with holes. The algorithm FL is comparable with JB with respect to all criteria except that of computational complexity. In this sense the algorithm JB is more preferable.

Four of the seven symmetry detection algorithms can be modified to solve the related congruity problem. Also for this problem the algorithm JB and SU are the best choice for practical usage. Both have a low computational complexity and an easy implementation. The two other algorithms FL and JJC have a much higher computational complexity. In particular, JJC requires an isomorphism test for two attributed graphs. The difficulty of this problem is well-known.

To conclude this section, simple and efficient symmetry detection algorithms like JB and SU are now available, making the actual use of symmetry information in many applications possible. Also for the congruity problem practical algorithms are known.

6 Applications of symmetries

Symmetry means some similarity or redundancy in the structure of a polyhedron. This redundancy leads often to equivalent subtasks in a problem-solving process, and thus to inefficiency. In the last years an increasing interest in the use of symmetry information could be observed

aiming at removing this redundancy in a variety of problems. In this section we discuss some applications of symmetries in robotics, geometric problem-solving, and computer vision.

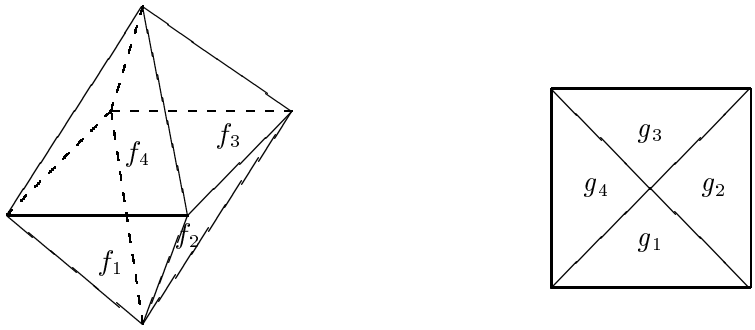
6.1 Robotics

The motivation behind the development of the optimal algorithms of Wolter et al. was the need for detecting symmetry in the context of a robotics application, where a set of images of polyhedra were synthesized for the training of a vision system [36]. If a polyhedron has symmetries, certain sets of viewpoints will produce identical-appearing images. They only incerase the computational burden for the training and don't make any essential contribution to the training. Here knowledge of the symmetries of the polyhedra can be used to eliminate redundant orientations.

High-level robotic assembly planning is concerned with how bodies fit together and how spatial relationships among bodies are established over time. In assembly planning one potential problem arises from the symmetries of the assembly components. A component of cubic shape, for example, contains eight equivalent vertices, twelve equivalent edges, and six equivalent faces. Without a proper treatment of symmetry a complete specification of any type of contact with such a component would be tedious. In [24, 28] symmetries have been exploited to overcome this problem. The assembly planning problem was formulated as a constraint satisfaction problem (CSP) over finite and infinite domains in [24]. It is known that finding a consistent solution of a general CSP is an NP-complete problem. The authors used clues deduced from the symmetries of the assembly components to reduce the combinatorics by minimizing the variable domains, thus making the assembly planning process computationally more tractable.

6.2 Geometric problem-solving

Waltzman [33] considered a geometric problem of the following type: We are given a set of three-dimensional polyhedral pieces and told that they all fit together to form a solid cube (with no empty space inside) of given dimensions. The problem is to determine how all the pieces fit together. The solution to this problem is thus the spatial location and orientation of each piece relative to the cubic container so that the container is completely filled by the pieces. In essential the problem is solved by a search. The basic problem state at any given time consists of the location and orientation of all the pieces placed so far, as well as the shape, location, and orientation of the containers remaining to be filled. (Note that there may be more than one container to be filled since the placement of any piece may divide a single container into two or more containers.) Each solution step places a piece at a particular location and orientation in space. If there are symmetries in the pieces or the containers, the symmetry information can be used to reduce the search space. The problem formulation above represents a general problem class and contains many special instances. The container, for example, may be a means of transportation (e.g., ship, plane, or truck) or of storage (e.g., shelve or bin). Related to this are problems of laying out office space or factory floor where the container is the total floor space that we want to work with and the objects are wall partitions, desks, office equipment, and so on. As test for his experimental system for solving this kind of problems, Waltzman considered two non-trivial three-dimensional jigsaw puzzles. In one puzzle, the problem-solving process would perform 27,648 state computations if no symmetry information was used. But by taking advantage of the symmetries of the pieces and the containers, the system was able to solve the puzzle in roughly 10 state computations. In another puzzle of much greater complexity, the search space contained about $3.3 \cdot 10^{29}$ states. By making use of symmetry information the



(a) (b)

Figure 8: (a) The model octahedron. In this perspective f_1, f_2, f_3 and f_4 are visible, and f_5, f_6, f_7 and f_8 are invisible. In our notation, the face $f_{i+4}, i = 1, 2, 3, 4$ is parallel to f_i, respectively. (a) A scene.

system solved the puzzle in roughly 200 state computations.

6.3 Computer vision

One of the central research topics in computer vision is that of model-based three-dimensional object recognition [4, 5, 7, 30, 31]. The goal of an object recognition system is to identify the objects present in an input scene and to determine the pose of each object, i.e., the transformation that brings the features of an object model in its own inherent coordinate system into agreement with the corresponding features in the world coordinate system of the scene.

Given a set of scene features $S = \{s_1, s_2, \cdots, s_n\}$ extracted from an input image and a set of model features $M = \{m_1, m_2, \cdots, m_l\}$ being, for instance, vertices, edges, or faces of the objects, the recognition task is to find a matching

$$\{(s_1, m_{i1}), (s_2, m_{i2}), \cdots, (s_n, m_{in})\}$$

that gives the corresponding model feature of each scene feature[3] and the related pose. Usually, topologic and geometric constraints are exploited to find consistent matchings.

Matching of a scene description to a symmetric model always leads to a number of equivalent solutions. As an illustrative example, let's consider the regular octahedron (in the following simply octahedron) in Fig. 8, which is one of the five Platonic solids. If we have detected four surface patches g_1, g_2, g_3, and g_4 in the scene and match them to the faces of the octahedron, 24 different matchings will result. Three of them are:

$$\{(g_1, f_1), (g_2, f_2), (g_3, f_3), (g_4, f_4)\},$$

$$\{(g_1, f_1), (g_2, f_4), (g_3, f_6), (g_4, f_7)\},$$

$$\{(g_1, f_5), (g_2, f_6), (g_3, f_4), (g_4, f_3)\}.$$

Obviously, all the 24 matchings are equivalent and finding any of them entirely solves the recognition task.

[3]Here it is assumed that all scene features stem from the same model. In real applications extraneous scene features resulting from an inperfect feature extraction and multiple objects must be taken into account.

An object recognition system should be aware of the symmetries of the models. Otherweise, it will find a number of equivalent matchings of equal goodness and resolution of this inherent ambiguity is not possible. Recently, researchers have begun to explicitly make use of the symmetries of models [11, 12, 17]. In [21] a framework of symmetry exploration for a number of popular object recognition paradigms, including interpretation tree search, hypothesize-and-test, invariant feature indexing of interpretation tables, pose clustering, and evidence based techniques, has been proposed. Generally, object recognition can profit from symmetry information in two ways. An object recognition system can use it to avoid the search for equivalent solutions, thus resulting in more efficiency. Alternatively, symmetry information can guide the ambiguity resolution process after the matchings have been found by finding out one representative and throwing away all other equivalent matchings.

A large-scale test of symmetry exploration in an object recognition system based on invariant feature indexing in interpretation tables [10] has been reported in [12]. In this experiment the model database consisted of 70 objects, ranging from the Platonic solids and variants over block-like polyhedra to chair-like objects. The 70 models yielded 48,146 possible face triples for the interpretation table, 37,681 of those being redundant. Hence, symmetry information discarded 78% of the entries in the interpretation table. The saving in recognition time was tested on five synthetic range images of each of the 70 models, i.e., totally 350 scenes. A total of 665,658 hypotheses for object recognition were retrieved from the interpretation table when they were constructed using symmetry information, while 3,454,122 hypotheses were retrieved without symmetry processing. Therefore, the use of symmetry information yielded an 81% reduction in the computational burden associated with hypothesis retrieval from the same model database. This experiment showed that the use of symmetry information not only drastically reduces the storage need for the model representations, but also significantly decreases the computation time.

7 Conclusion

Symmetries of objects have been found useful in fields like robotics, geometric problem-solving, and computer vision. The actual use of symmetries, however, is crucially dependent on the availability of simple and efficient symmetry detection algorithms. In this paper, we have reviewed seven algorithms for extracting symmetries of polyhedra and compared them with respect to the computational complexity, the implementation, the suitability of the algorithm output for practical use, and the polyhedra class that the algorithm can handle.

In our study we have restricted the class of polyhedra to those without cavities in their faces whose graphs are planar. This restriction is due to the fact that most of the symmetry detection algorithms assume a connected polyhedron graph. An exception is the algorithm of Flynn. Also, the earlier version of the algorithm of Jiang & Bunke in [18] that was not considered in this study can handle disconnected polyhedron graphs. A future research work is thus to extend the symmetry detection algorithms to the most general case. This would definitely enlarge the usefulness of these algorithms in many applications.

Polyhedra correspond to the lowest level in a hierarchy of object representations with increasing complexity. More complex geometric objects are, for instance, quadric-surfaced objects that are very popular in research on three-dimensional object recognition. Another extension of symmetry detection algorithms is to accommodate this kind of curved objects. Currently, this aspect has been almost entirely ignored. Only Flynn [11] claims that his algorithm can handle some quadrics.

References

[1] S. G. Akl and G. T. Toussaint, "An improved algorithm to check for polygon similarity", Information Processing Letters, Vol. 7, 127–128, 1978.

[2] H. Alt, K. Mehlhorn, H. Wagener, and E. Welzl, "Congruence, similarity, and symmetries of geometric objects", Discrete & Computational Geometry, Vol. 3, 237–256, 1988.

[3] M. A. Armstrong, Groups and Symmetry, Springer-Verlag, New York, 1988.

[4] P. J. Besl and R. C. Jain, "Three dimensional object recognition", ACM Computing Surveys, Vol. 17, No. 1, 75–145, 1985.

[5] J. P. Brady, N. Namdhakumar, and J. K. Aggarwal, "Recent progress in object recognition from range data", Image and Vision Computing, Vol. 7, No. 4, 295–307, 1989.

[6] A. Bykat, "On polygon similarity", Information Processing Letters, Vol. 9, 23–25, 1979.

[7] R. T. Chin and C. R. Dyer, "Model based recogntion in robot vision", ACM Computing Surveys, Vol. 18, No. 1, 67–108, 1986.

[8] P. Eades and H. C. Ng, "An algorithm for detecting symmetries in line drawings", Ars Combinatoria, Vol. 23A, 95–104, 1987.

[9] P. Eades, "Symmetry finding algorithms", In: Computational Morphology (G. T. Toussaint, Ed.), North-Holland, 41–51, 1988.

[10] P. J. Flynn and A. K. Jain, "3D object recognition using invariant feature indexing of interpretation tables", Computer Vision, Graphics, and Image Analysis: Image Understanding, Vol. 55, No. 2, 119–129, 1992.

[11] P. J. Flynn, "Saliencies and symmetries: Towards 3D object recognition from large model databases", Proc. of CVPR'92, 322–327, 1992.

[12] P. J. Flynn, "3-D object recognition with symmetric models: Symmetry extraction and encoding", IEEE Trans. on PAMI, Vol. 16, No. 8, 814–818, 1994.

[13] G. Gati, "Further annotated bibiography on the isomorphism disease", Journal of Graph Theory, Vol. 3, 95–109, 1979.

[14] P. T. Highnam, "Optimal algorithms for finding the symmetries of a planar points set", Information Processing Letters, Vol. 22, 219–222, 1986.

[15] J. E. Hopcroft and R. E. Tarjan, "A $VlogV$ algorithm for isomorphism of triconnected planar graphs", Journal of Computer and System Sciences, Vol. 7, 323–331, 1973.

[16] J. E. Hopcroft and J. K. Wong, "Linear time algorithm for isomorphism of planar graphs", Proc. of 6th Annual ACM Symposion on Theory of Computing, 172–184, 1974.

[17] X. Y. Jiang and H. Bunke, "Determination of the symmetries of polyhedra and an application to object recognition", In: Computational geometry – Methods, algorithms and applications (H. Bieri, H. Noltemeier, Eds.), Lecture Notes in Computer Science 553, Springer-Verlag, 113–121, 1991.

[18] X. Y. Jiang and H. Bunke, "Determining symmetry of polyhedra", In: Visual form: Analysis and recognition (C. Arcelli, L. P. Cordella, G. Sanniti di Baja, Eds.), Proc. of Int. Workshop on Visual Form, Capri, 1991, Plenum Press, New York, 303–312, 1992.

[19] X. Y. Jiang and H. Bunke, "A simple and efficient algorithm for determining the symmetries of polyhedra", Computer Vision, Graphics, and Image Analysis: Graphical Models and Image Processing, Vol. 54, No. 1, 91–95, 1992.

[20] X. Y. Jiang and H. Bunke, "Polyhedral symmetry: Detection algorithms and application to 3-D object recognition", Proc. of Swiss Vision'93, 169–177, 1993.

[21] X. Y. Jiang and H. Bunke, A framework of symmetry exploration in 3D object recognition, Proc. of IAPR Int. Workshop on Structural and Syntactic Pattern Recognition, Nahariya, Israel, 1994. (to appear)

[22] P. Johansen, N. Jones, and J. Clausen, "A method for detecting structure in polyhedra", Pattern Recognition Letters, Vol. 2, 217–225, 1984.

[23] J.-C. Lin, S.-L. Chou, and W.-H. Tsai, "Detection of rotationally symmetric shape orientations by fold-invariant shape-specific points", Pattern Recognition, Vol. 25, No. 5, 473–482, 1992.

[24] Y. Liu and R. J. Popplestone, "Symmetry constraint inference in assembly planning – Automatic assembly configuration specification", Proc. of AAAI-90, 1038–1044, 1990.

[25] G. E. Martin, Transform Geometry: An Introduction to Symmetry, Springer-Verlag, New York, 1982.

[26] T. Masuda, K. Yamamoto, and H. Yamada, "Detection of partial symmetry using correlation with rotated-reflected images", Pattern Recognition, Vol. 26, No. 8, 1245–1253, 1993.

[27] S.-C. Pei and C.-N. Lin, "Normalization of rotationally symmetric shapes for pattern recognition", Pattern Recognition, Vol. 25, No. 9, 913–920, 1992.

[28] R. J. Popplestone, Y. Liu, and R. Weiss, "A group theoretic approach to assembly planning", AI magazine, Vol. 11, No. 1, 82–97, 1990.

[29] R. C. Read and D. G. Corneil, "The graph isomorphism disease", Journal of Graph Theory, Vol. 1, 339–363, 1977.

[30] G. Stockman, "Object recognition", In: Analysis and Interpretation of Range Images (R. C. Jain, A. K. Jain, Eds.), Springer-Verlag, 225–253, 1990.

[31] P. Suetens, P. Fua, and A. J. Hanson, "Comptational strategies for object recognition", ACM Computing Surveys, Vol. 24, No. 1, 5–61, 1992.

[32] K. Sugihara, "An *nlogn* algorithm for detecting the congruity of polyhedra", Journal of Computer and System Sciences, Vol. 29, 36–47, 1984.

[33] R. Waltzman, Geometric problem solving by machine visualization, CS-TR-2291, University of Maryland, 1989.

[34] L. Weinberg, "A simple and efficient algorithm for determining isomorphism of planar triply connected graphs", IEEE Trans. on Circuit Theory, Vol. 13, No. 2, 142–148, 1966.

[35] J. D. Wolter, T. C. Woo, and R. A. Volz, "Optimal algorithms for symmetry detection in two and three dimensions", The Visual Computer, Vol. 1, 37–48, 1985.

[36] J. D. Wolter, R. A. Volz, and T. C. Woo, "Automatic generation of gripping positions", IEEE Trans. on Systems, Man, and Cybernetics, Vol. 15, No. 2, 204–213, 1985.

[37] H. Zabrodsky, S. Peleg, and D. Arnir, "A measure of symmetry based on shape similarity", Proc. of CVPR'92, 703–706, 1992.