An Algorithm for Finding Three Dimensional Symmetry in Trees *

Seok-Hee Hong and Peter Eades

Basser Department of Computer Science, University of Sydney, Australia.
{shhong, peter}@cs.usyd.edu.au

Abstract. This paper presents a model for drawing trees symmetrically in three dimensions and a linear time algorithm for finding maximum number of three dimensional symmetries in trees.

1 Introduction

Symmetry is one of the most important aesthetic criteria for Graph Drawing. Drawings of graphs in Graph Theory textbooks are often symmetric, because the symmetry clearly reveals the structure of the graph.

However, previous work on symmetric graph drawing has only focused on two dimensions. The problem of determining whether a given graph can be drawn symmetrically is NP-complete in general [10]. Heuristics for symmetric drawings of general graphs have been suggested [4]. For restricted classes of graphs, there are polynomial time algorithms: Manning presents algorithms for constructing symmetric drawings of trees, outerplanar graphs and embedded planar graphs [8,9,10]; Hong gives algorithms for finding maximum number of symmetries in series-parallel digraphs and planar graphs [5,6].

In this paper, we extend symmetric graph drawing into three dimensions. Symmetry in three dimensions is much richer than that in two dimensions. For example, a maximal symmetric drawing of a tree in two dimensions is in Figure 1 (a), showing 12 symmetries. However, the maximal symmetric drawing of the same tree in three dimensions shows 48 symmetries, as in Figure 1 (b). In fact, using more complex examples, we can prove the following theorem.

Theorem 1. *For each integer $n \geq 10$ there is a tree T with n nodes such that T has no symmetric drawing in two dimensions, but has a drawing in three dimensions that displays $4\lfloor \frac{n-6}{8} \rfloor + 2$ symmetries.* [1]

This paper is organized as follows. In the next section, we give a model for drawing graphs symmetrically in three dimensions. The main results of the paper

* This research has been supported by a Postdoctoral Fellowship from the Korean Science and Engineering Foundation and a grant from the Australian Research Council. Animated drawings are available from S. Hong at http://www.cs.usyd.edu.au/@shhong/research3.htm.

[1] In this extended abstract, proofs are omitted.

J. Marks (Ed.): GD 2000, LNCS 1984, pp. 360–371, 2001.

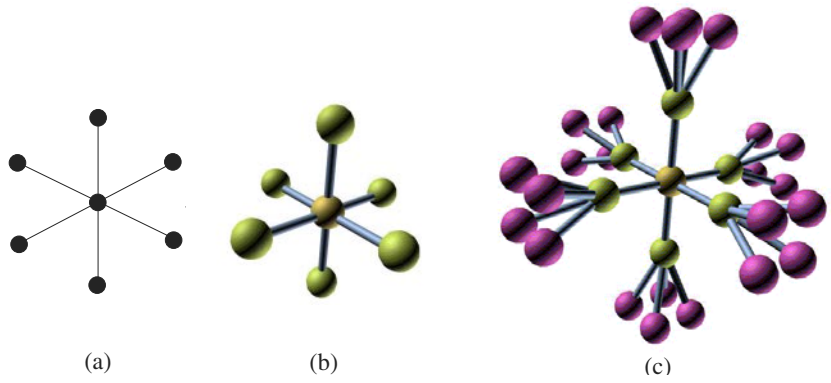

Fig. 1. *Symmetric drawings of trees in two and three dimensions.*

are in Section 3: here we present an algorithm for finding maximum number of symmetries of trees in three dimensions. A significant contribution of Section 3 is the introduction of a new data structure called the "Isomorphism Class Tree"; this structure is critical for the efficiency of the algorithm. A simple drawing algorithm is briefly described in Section 4. Section 5 concludes.

2 Symmetric Graph Drawing in Three Dimensions

In this section, we first review the model for symmetric graph drawing in two dimensions. To explain symmetry in three dimensions, we review some terminology from geometric symmetry and group theory [2,7,11]. Finally describe our new symmetry model in three dimensions.

2.1 Symmetric Graph Drawing in Two Dimensions

A symmetry of a two dimensional figure is an isometry of the plane that fixes the figure. There are two types of two dimensional symmetry, *rotational symmetry* and *reflectional (or axial) symmetry*. Rotational symmetry is a rotation about a *point* and reflectional symmetry is a reflection in an *axis*.

Symmetry in graph drawing is closely related to automorphisms of graphs: a symmetry of a graph drawing induces an automorphism of the graph. In this case, we say that the drawing *displays* the automorphism. If an automorphism is displayed as a symmetry in a drawing of the graph, then it is a *geometric* automorphism. The most critical part of the problem of drawing a graph symmetrically is to find a large group of geometric automorphisms. This formal model for symmetric drawing in two dimensions was introduced by a number of authors [3,5,6,8,9,10].

2.2 Symmetries in Three Dimensions

Symmetry in three dimensions is richer and more complex than symmetry in two dimensions. The types of symmetry in three dimensions can be roughly classified as *direct symmetry* and *indirect symmetry*. These are further refined as *rotation*, *reflection*, *inversion* and *rotary reflection (or rotary inversion)* [7, 11]. The difference from two dimensions is that a rotational symmetry in three dimensions is a rotation about an *axis* and a reflectional symmetry in three dimensions is a reflection in a *plane*. Inversion (or central inversion) is a reflection in a *point*. Rotary reflection (inversion) is a composition of a rotation and a reflection (inversion).

A *finite rotation group* in three dimensions is one of following three types. A *cyclic group* (C_n), a *dihedral group* (D_n) and the rotation group of one of the *Platonic solids* [2,11]. There are only five regular Platonic solids, the *tetrahedron*, the *cube*, the *octahedron*, the *dodecahedron* and the *icosahedron*.

There are many types of *full symmetry groups* of a finite object in three dimensions. The complete list of all possible symmetry groups in three dimensions can be found in [2,7,11]. However, all are variations on just three types: pyramids, prisms, and Platonic solids. It can be shown that for the case of *trees*, a *maximum* size three dimensional symmetry group is one of three types: a regular pyramid configuration, a regular prism configuration, and the Platonic solids configuration.

A *regular pyramid* is a pyramid with a regular k-gon as its base. There is only one *k-fold rotation axis*, passing through the apex and the center of its base. There are k rotational symmetries, each of which is a rotation of $2\pi i/k$, $i = 0, 1, \ldots, k-1$. Also there are k reflectional symmetries in *reflection planes*, each containing the rotation axis. In total, the regular pyramid has $2k$ symmetries.

A *regular prism* has a regular k-gon as its top and bottom face. There are $k+1$ rotation axes and they can be divided into two classes. The first one, called the *principal* axis, is a k-fold rotation axis which passes through the centers of the two k-gon faces. The second class, of *secondary* axes, consists of k 2-fold rotation axes which lie in a plane perpendicular to the principal axis. The number of rotational symmetries is $2k$. Also, there are k reflection planes, each containing the principal axis, and another reflection plane perpendicular to the principal axis. Further it has $k-1$ rotary reflections. If k is even, then they are the same as rotary inversions including the central inversion. In total, the regular prism has $4k$ symmetries.

The tetrahedron has four 3-fold rotation axes and three 2-fold rotation axes. It has 12 rotational symmetries and in total 24 symmetries. The octahedron has three 4-fold rotation axes, four 3-fold axes, and six 2-fold rotation axes. It has 24 rotational symmetries and a full symmetry group of size 48. The icosahedron has six 5-fold rotation axes, ten 3-fold rotation axes, and fifteen 2-fold rotation axes. It has 60 rotational symmetries and a full symmetry group of size 120. Note that the cube and the octahedron are dual solids, and the dodecahedron and the icosahedron are dual. For details, see [2,7,11].

2.3 Symmetric Graph Drawing in Three Dimensions

A symmetry of a three dimensional graph drawing induces an automorphism of the graph, and this automorphism is *displayed* by the symmetry. We say that the automorphism is a *three dimensional symmetry* of the graph.

To draw a graph symmetrically in three dimensions, there are two steps: first find the three dimensional symmetries, then construct a drawing which displays these symmetries. The first step is the more difficult; given the three dimensional symmetries, the drawing is easy to construct. This paper concentrates on the first step.

For the purpose of drawing graphs symmetrically, we require a graph drawing to satisfy three *non-degeneracy* conditions: no two vertices are located at the same point, no two edges overlap (they may intersect at a point), and no vertex lies on an edge with which it is not incident.

Now we are ready to find three dimensional symmetry in trees. In the next section, we present an algorithm for finding maximum number of symmetries of trees in three dimensions.

3 Symmetry Finding Algorithm

In this section we describe an algorithm for finding three dimensional symmetry in trees.

The *center* of a tree is a vertex c such that the maximum distance between c and any leaf is minimized. The algorithm treats the input tree as a rooted tree, rooted at a center. Every tree has either one center or a set of two adjacent centers; however, for this algorithm, we need a single center and if there are two centers, then we add a new vertex on the edge joining the two centers to make one center. The center of a tree T is fixed by every automorphism of T and thus every symmetry of a drawing of T fixes the location of the center.

The basic idea of the symmetry finding algorithm is to construct all possible symmetric configurations and then find the configuration which has the maximum number of symmetries. To construct the symmetric configurations, we place the center of the input tree at the apex of a pyramid, or at the centroid of a prism and the Platonic solids. Then we place each subtree attached to the center to form a symmetric configuration.

The algorithm computes an auxiliary tree called the *Isomorphism Class Tree* (ICT). This tree is a fundamental structure defining the isomorphisms between subtrees in the input tree. Once the ICT has been computed, we use separate subroutines to find the symmetries in the pyramid, prism, and the Platonic solids configurations. These subroutines use the data stored at the root and the first level of the ICT. Thus, the overall algorithm can be divided into four steps, as follows.

`Algorithm 3DSymmetry_Tree`
`Input:` A tree T.
`Output:` A maximum size group of three dimensional symmetries of T.

1. Find the center of T and root T at the center.
2. Construct the *Isomorphism Class Tree* (ICT) of T.
3. Find symmetries of each type.
   a) Construct a pyramid configuration.
   b) Construct a prism configuration.
   c) Construct Platonic solids configuration.
4. Output the group of the configuration which has maximum size.

In Section 3.1, we define the ICT and describe an algorithm to construct it. Steps 3 (a), (b) and (c) are described in Section 3.2, 3.3 and 3.4. The following theorem summarises the results.

Theorem 2. `Algorithm 3DSymmetry_Tree` *computes a maximum size three dimensional symmetry group of a tree in linear time.*

3.1 The Isomorphism Class Tree

Roughly speaking, the *Isomorphism Class Tree (ICT)* of a tree T represents the isomorphism classes of subtrees of T, some relationships between these classes, the size of each class, and the sizes of the rotational symmetry groups of the subtrees. Before giving the formal definition of the ICT, first we explain why it is needed. As an example, we consider the pyramid configuration, because it is simplest.

Let c be the center of T. As mentioned above, we root T at c. Deleting c and all its incident edges results in a collection of rooted disjoint subtrees $T_1, T_2, \ldots, T_m$. Each T_i is rooted at the vertex c_i that was adjacent to the center. Using a rooted tree isomorphism algorithm [1] we can partition the T_i into rooted isomorphism classes $I_1, I_2, \ldots, I_k$. That is, if T_i and T_j are isomorphic subtrees, then they belong to the same isomorphism class. Let $n_i = \mid I_i \mid$ and g be the *greatest common divisor* (*gcd*) of all n_i. Then we can construct a pyramid with a g-fold rotation axis by placing the center at the apex and distributing the subtrees in the reflection planes, each containing a side edge of the pyramid (See Figure 3 (a)). It is clear that for each divisor j of g, there is a pyramid drawing which displays j rotational symmetries.

However, other symmetries are possible. We can choose one subtree T_j from I_i and place it such that the edge $(c,\ c_j)$ is on the rotation axis as in Figure 3 (b). Note that in this case, the tree T_j must be fixed by a rotational symmetry. Thus the symmetry group of the pyramid is the intersection of the two symmetry groups: one of T_j, and the other which permutes the remaining subtrees $T_1, T_2, \ldots, T_{j-1}, T_{j+1}, \ldots, T_m$. The size of this group is a divisor of $gcd(e, n_1, n_2, \ldots, n_i - 1, \ldots, n_k)$ where e is the size of a rotational symmetry group of T_j.

Suppose that L_i is the set of sizes of rotational symmetry groups of T_j, and G_i is the set of divisors of $gcd(n_1, n_2, \ldots, n_i - 1, \ldots, n_k)$. Using this notation, we can compute the set L of sizes of the rotational symmetry groups (in the pyramid configuration) for T as follows.

`Algorithm ComputeL`
`Input:` The sizes $n_1, n_2, \ldots, n_k$ of the isomorphism classes of the subtrees at node u.
`Output:` The set L of sizes of rotational symmetries of the subtree rooted at u.

1. L = the set of divisors of $gcd(n_1, n_2, \ldots, n_k)$.
2. for $i = 1$ to k do
   a) Compute G_i = the set of divisors of $gcd(n_1, n_2, \ldots, n_i - 1, \ldots, n_k)$.
   b) Add $L_i \cap G_i$ to L.

This algorithm requires the computation of L_i for $i = 1, 2, \ldots, k$; in principle, this can be computed recursively. In practice it is more efficient to use the *Isomorphism Class Tree* (ICT), defined as follows.

Each node in the ICT represents an isomorphism class. Suppose that c is the center of a tree T and $I_1, I_2, \ldots, I_k$ are the rooted isomorphism classes of the rooted subtrees $T_1, T_2, \ldots, T_m$ of c. The root node r of the ICT represents the whole tree T. The children $u_1, u_2, \ldots, u_k$ of r represent the isomorphism classes $I_1, I_2, \ldots, I_k$ respectively. Suppose that $T_j \in I_i$ and c_j is the root of T_j. We can recursively decompose T_j into subtrees $T_{j_1}, T_{j_2}, \ldots, T_{j_p}$ by deleting c_j and then divide them into isomorphism classes $I_{i_1}, I_{i_2}, \ldots, I_{i_q}$. Then u_i has $u_{i_1}, u_{i_2}, \ldots, u_{i_q}$ as its children in the ICT. The ICT can be recursively constructed in this way until a subtree becomes a node. Figure 2 shows an example of the ICT.

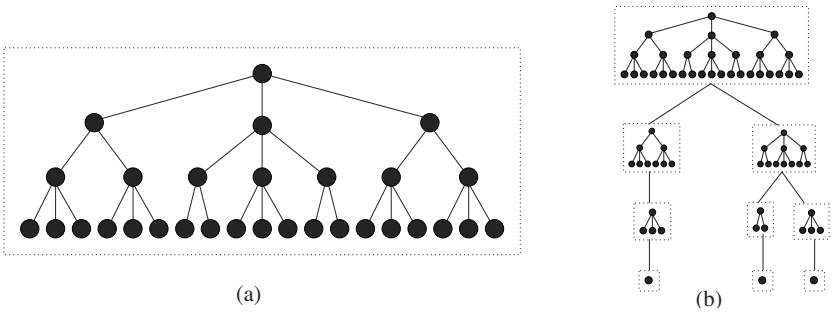

Fig. 2. *(a) a tree T (b) The Isomorphism Class Tree (ICT) of T.*

Suppose that v is a node in the ICT, representing an isomorphism class I_v and suppose that $T_j \in I_v$. We associate two values with v: the integer $n_v = |I_v|$ and the set L_v of sizes of the rotational symmetries of T_j. These two values are useful in finding the three dimensional symmetries of a tree.

Next we consider algorithms for constructing the ICT and its associated values. Using a rooted tree isomorphism algorithm [1], it is simple to construct the ICT and each n_v in linear time. However, we need an algorithm to compute L_v of each v. This can be computed by applying `ComputeL` in a bottom up

approach on the ICT. However, a direct implementation of `ComputeL` may be expensive; we now show how to do it in linear time. Suppose that u is a node of T and $I_1, I_2, \ldots, I_k$ are the rooted isomorphism classes of the rooted subtrees $T_1, T_2, \ldots, T_m$ of u. We want to implement `ComputeL` at the node v corresponding to u in time $O(m)$. This can be done as follows.

We use a bit array to represent G_i, that is, $G_i[p] = 1$ if and only if p is a divisor of each of $n_1, n_2, \ldots, n_i - 1, \ldots$, and n_k. Also we represent L_i as a bit array: if $T_j \in I_i$ has a drawing which displays p rotations, then $L_i[p] = 1$ and $L_i[p] = 0$ otherwise. Note that if $L_i[p] = 1$, then $L_i[q] = 1$ for all divisors q of p. The output L of `ComputeL` can be represented in the same way. Then to compute $G_i \cap L_i$ and add it to L, we take the bitwise AND of L_i and G_i, and then the bitwise OR with L; this can be done in time $O(\min(\max(G_i), \max(L_i)))$.

Since $gcd(n_1, n_2, \ldots, n_k) \leq \min(n_1, n_2, \ldots, n_k)$, Step 1 of `ComputeL` can be implemented in time $O(k \min(n_1, n_2, \ldots, n_k))$, which is $O(m)$.

For Step 2, we consider two cases.

- For all p, $n_p > 1$. In this case,

$$gcd(n_1, n_2, \ldots, n_i - 1, \ldots, n_k) \leq \min(n_1, n_2, \ldots, n_i - 1, \ldots, n_k). \quad (1)$$

  Thus $\max(G_i) \leq \min(n_1, n_2, \ldots, n_k)$. It can be deduced that both parts of step 2 can be implemented in time $O(k \min(n_1, n_2, \ldots, n_k))$, which is $O(m)$.
- For some p, $n_p = 1$. In this case, the inequality (1) does not hold. However, for $i \neq p$, $G_i = \{1\}$; thus we only need to execute steps 2(a) and 2(b) for the single case $i = p$. Both Step 2(a) and Step 2(b) take time $O(\max(G_p))$, which is $O(\max(n_1, n_2, \ldots, n_{p-1}, n_{p+1}, \ldots, n_k))$, which is $O(m)$.

It follows that `ComputeL` can be implemented in time $O(m)$, and thus the whole of the ICT, including each L_v, can be computed in linear time.

The following sections describe steps 3(a), 3(b), and 3(c) of `3DSymmetry_Tree` in turn. These algorithms use the ICT; in fact, they only consider the root r and its children u_i (together with n_i and L_i) in the ICT.

3.2 Pyramid Configuration

In this section, we give an algorithm for finding all possible rotational symmetries in the pyramid configuration. The basic idea is to construct a pyramid-type drawing of a tree by placing the center of the tree at the apex of a pyramid, some fixed subtrees about the rotation axis, and k isomorphic subtrees in the reflection planes that contain the side edges of the pyramid. The resulting drawing has the same symmetry group of the k-gon based pyramid.

The set L_r at the root of the ICT contains the sizes of rotational symmetry groups which fix at most one subtree. However, there is another way of constructing symmetric drawings of trees by placing another subtree on the rotation axis: we can place *two* possibly different subtrees on the rotation axis, if the symmetry groups of the fixed subtrees are appropriate.

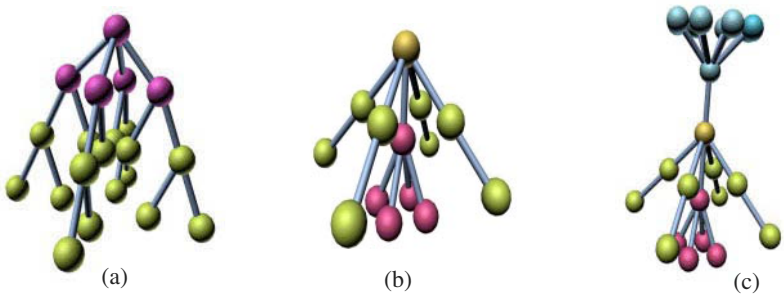

Fig. 3. *The pyramid configuration with different number of fixed subtrees.*

For example, Figure 3 (a) shows the pyramid with no fixed subtree and Figure 3 (b) shows the pyramid with one fixed subtree. Figure 3 (c) shows the pyramid with two fixed subtrees.

The pyramid algorithm mainly depends on the number of fixed subtrees on the rotation axis. This is at most two, as described in the following Lemma.

Lemma 1. *Suppose that T is a tree rooted at the center c of T. Let $T_1, T_2, \ldots, T_m$ be the rooted subtrees of c. In a pyramid configuration, there are at most two fixed subtrees T_i and T_j on the rotation axis.*

From Lemma 1 one can derive the following theorem, which forms the basis of the algorithm.

Theorem 3. *Suppose that T is a tree, r is the root node of the ICT of T, and $u_1, u_2, \ldots, u_k$ are the children of r in the ICT. Then:*

1. *If g is the size of a rotation group in the pyramid configuration, then g is a divisor of $gcd(n_1, n_2, \ldots, n_k)$.*
2. *If there is one fixed subtree T_j, then the rotation group of the pyramid is the intersection of the rotation group of T_j and a rotation group of the remaining T_i $(i = 1, 2, \ldots, m, i \neq j)$. Suppose that u_p represents the isomorphism class I_p for which $T_j \in I_p$, and G_p is the set of divisors of $gcd(n_1, n_2, \ldots, n_p - 1, \ldots, n_k)$. Then the intersection is $G_p \cap L_p$.*
3. *If there are two isomorphic fixed subtrees T_i and T_j, then the rotation group of the pyramid is the intersection of the rotation group of T_i (or T_j) and the rotation group of the remaining T_ℓ $(\ell = 1, 2, \ldots, m, \ell \neq i, j)$. Suppose that u_p represent the isomorphism class I_p to which T_i and T_j belong, and G'_p is the set of divisors of $gcd(n_1, n_2, \ldots, n_p - 2, \ldots, n_k)$. Then the intersection is $G'_p \cap L_p$.*
4. *If there are two nonisomorphic fixed subtrees T_i and T_j, then the rotation group of the pyramid is the intersection of the rotation group of T_i, the rotation group of T_j and the rotation group of the remaining T_ℓ $(\ell = 1, 2, \ldots, m, \ell \neq i, j)$. Suppose that u_p (u_q) represents the isomorphism class*

I_p (I_q) to which T_i (T_j) belong, and G_{pq} is the set of divisors of $gcd(n_1, n_2, \ldots, n_p - 1, \ldots, n_q - 1, \ldots, n_k)$. Then the intersection is $G_{pq} \cap L_p \cap L_q$.

A direct implementation based on Theorem 3 may be computationally expensive. To reduce the time complexity, we use an approach based on trial division. We test candidate sizes g of the rotation group. First we compute the remainder r_i of the division of n_i by g. There are four cases, corresponding to the four parts of Theorem 3:

1. If all $r_i = 0$ for all i, then g is the size of a rotation group, with no fixed subtree.
2. Suppose that there is only one p such that $r_p = 1$ and all the others are 0; further suppose that $g \in L_p$. Then there is a rotation of size g, with one fixed subtree.
3. Suppose that there is only one p such that $r_p = 2$ and all the others are 0; further suppose that $g \in L_p$. Then there is a rotation of size g, with two isomorphic fixed subtrees.
4. Finally, suppose that there are only two r_p and r_q which are 1, and all other are 0; further suppose that $g \in L_p \cap L_q$. Then there is a rotation of size g, with two nonisomorphic fixed subtrees.

If none of the four cases above hold, then g is not the size of a rotation group.

Next we need to compute the intersection of the possible candidate g and the rotation group of the fixed subtrees (L_p or L_q). We can use the same bit representation which was used for L_v in the ICT as in the previous section. Each candidate g is represented by a bit array G in a similar fashion. Again, the intersections are computed by a bitwise AND, but in this case there may be more than two vectors involved (when there are two candidates, as in item 4 above).

Now we present an algorithm to find the symmetries in a pyramid configuration.

`Algorithm Pyramid`
(1) For each candidate g do
  (1.1) Compute the remainder r_i of n_i divided by g, for $1 \leq i \leq k$.
  (1.2) Compute the set G of divisors of g.
  (1.3) Compute the set L of sizes of all rotational symmetry groups:
    (1.3.1) If $r_i = 0$ for $1 \leq i \leq k$, then add G to L.
    (1.3.2) If $r_p = 1$, and $r_i = 0$ for $i \neq p$, then add $G \cap L_p$ to L.
    (1.3.3) If $r_p = 2$ and $r_i = 0$ for $i \neq p$, then add $G \cap L_p$ to L.
    (1.3.4) If $r_p = 1$, $r_q = 1$ and $r_i = 0$ for $i \neq p, q$, then add $G \cap L_p \cap L_q$ to L.
(2) Return the maximum element of L.

An analysis similar to that used in the previous section shows that `Pyramid` can be implemented in linear time. Note that the output of `Pyramid` is the maximum number of *rotational* symmetries in the pyramid configuration; the maximum size of a symmetry group in the pyramid configuration is twice as big.

3.3 Prism Configuration

A tree drawn in a prism configuration has at most three types of rotation axes for fixing subtrees, and the number of fixed subtrees can be larger than with the pyramid configuration. This is described in the following Lemma.

Lemma 2. *Suppose that T is a tree rooted at the center c of T. Let $T_1, T_2, \ldots, T_m$ be rooted subtrees of c. In a regular k-gon prism configuration, there are zero or two fixed isomorphic subtrees on the k-fold rotation axis. Also, there are at most two types of k fixed isomorphic subtrees on the secondary axes.*

For example, Figure 4 (a) shows the prism with no fixed subtree and Figure 4 (b) shows the prism with fixed subtrees on the principal axis. Figure 4 (c) shows the prism with fixed subtrees on the principal and the secondary axes.

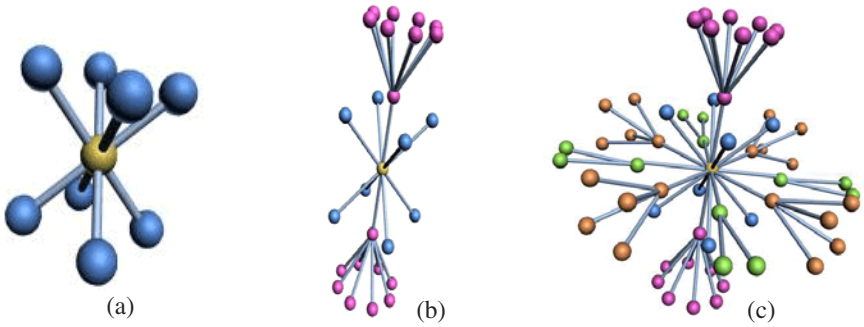

Fig. 4. *The prism configuration with different number of fixed subtrees.*

From Lemma 2 we can derive the following theorem, which forms the basis of the prism algorithm.

Theorem 4. *Suppose that T is a tree and the ICT is the Isomorphism Class Tree of T. Suppose that r is the root node of the ICT and $u_1, u_2, \ldots, u_k$ are the children of r in the ICT. Each u_i has n_i and L_i. Suppose that g is the size of the principal rotation axis of the prism configuration. Let r_i be the remainder of each n_i divided by g and m_i be the quotient of n_i divided by g. Then g satisfies both of the following conditions.*

1. *At most two m_i are odd and $2 \in L_i$.*
2. *At most one r_i is 2 and $g \in L_i$.*

Using theorem 4, one can construct a linear time algorithm, similar to `Pyramid`, to find the maximum number of rotational symmetries in a prism configuration. We omit this algorithm. Note that maximum number of symmetries in the prism configuration is four times as big as the maximum number of rotational symmetries in the prism configuration.

3.4 Platonic Solids Configuration

The Platonic solids have many rotation axes. However, the symmetry groups of the Platonic solids are fixed, and we only need to test whether we can construct a three dimensional drawing of a tree which has the same symmetry group as one of the Platonic solids. Using the similar method in the previous section, we can test this in a relatively simple way. For an example, we consider the cube. The number of the fixed subtrees on each axis is described in the following Lemma.

Lemma 3. *Suppose that T is a tree rooted at the center c of T. Let $T_1, T_2, \ldots, T_m$ be rooted subtrees of c. In the cube configuration, there are either zero or six fixed isomorphic subtrees on the 4-fold rotation axes. Also there are either zero or eight fixed isomorphic subtrees on the 3-fold rotation axes and either zero or twelve fixed isomorphic subtrees on the 2-fold rotation axes.*

For example, Figure 1 (c) shows the fixed subtree on the 4-fold axes of the cube configuration. From Lemma 3 we can derive the following theorem.

Theorem 5. *Suppose that T is a tree and the ICT is the Isomorphism Class Tree of T. Suppose that r is the root node of the ICT and $u_1, u_2, \ldots, u_k$ are the children of r in the ICT. Each u_i has n_i and L_i. Let r_i be the remainder of each n_i divided by 24. Then, if each r_i and L_i satisfies one of the following conditions, T has the same symmetry group as the cube.*

1. *At most one r_i is 6 and $4 \in L_i$ and at most one r_j is 8 and $3 \in L_j$ and at most one r_k is 12 and $2 \in L_k$.*
2. *At most one r_i is 14 and $4, 3 \in L_i$ and at most one r_j is 12 and $2 \in L_j$.*
3. *At most one r_i is 18 and $4, 2 \in L_i$ and at most one r_j is 8 and $3 \in L_j$.*
4. *At most one r_i is 20 and $3, 2 \in L_i$ and at most one r_j is 6 and $4 \in L_j$.*
5. *At most one r_i is 2 and $4, 3, 2 \in L_i$.*

Theorem 5 can be used to construct a linear time algorithm to test whether a tree has the same symmetry group as the cube. Similar results can be used to construct algorithms to test whether a tree has the same symmetry group as the icosahedron and the tetrahedron. We omit these algorithms.

4 Drawing Algorithm

Given a tree T, and a group Γ of three dimensional symmetries of T, it is straightforward to construct a straight-line drawing of T which displays Γ.

We draw the center of the tree at the origin. Non-fixed subtrees of the center are drawn in reflection planes through the origin, in such a way that the drawings of isomorphic subtrees are congruent. Fixed subtrees are drawn recursively. By assigning disjoint areas of the reflection planes to different subtrees, one can ensure planarity of the drawing.

5 Conclusion

This paper presents a linear time algorithm to construct maximally symmetric drawings of trees in three dimensions. These symmetries are symmetries of the *whole* drawing. It is also possible to extend this work to display symmetries of part of the drawing by providing a model and algorithms for such "partial symmetries" in three dimensions.

As further work, we would like to draw planar graphs symmetrically in three dimensions. Heuristics for drawing general graphs symmetrically in three dimensions remains as a challenge.

References

1. A. Aho, J. Hopcroft and J. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, 1974.
2. M. A. Armstrong, *Groups and Symmetry,* Springer-Verlag, 1988.
3. P. Eades and X. Lin, Spring Algorithms and Symmetry, *Computing and Combinatorics,* Springer Lecture Notes in Computer Science 1276, (Ed. Jiang and Lee), pp. 202-211.
4. H. Fraysseix, An Heuristic for Graph Symmetry Detection, *Graph Drawing'99,* Lecture Notes in Computer Science 1731, (Ed. J. Kratochvil), pp. 276-285, Springer Verlag, 1999.
5. S. Hong, P. Eades, A. Quigley and S. Lee, Drawing Algorithms for Series-Parallel Digraphs in Two and Three Dimensions, In S. Whitesides, editor, Graph Drawing (Proc. GD'98), vol. 1547 of Lecture Notes in Computer Science, pp. 198-209, Springer Verlag, 1998.
6. S. Hong, P. Eades and S. Lee, An Algorithm for Finding Geometric Automorphisms in Planar Graphs, *Algorithms and Computation,* Lecture Notes in Computer Science 1533, (Ed. Chwa and Ibarra), pp. 277-286, Springer Verlag, 1998.
7. E. H. Lockwood and R. H. Macmillan, *Geometric Symmetry,* Cambridge University Press, 1978.
8. J. Manning and M. J. Atallah, Fast Detection and Display of Symmetry in Trees, *Congressus Numerantium,* 64, pp. 159-169, 1988.
9. J. Manning and M. J. Atallah, Fast Detection and Display of Symmetry in Outerplanar Graphs, *Discrete Applied Mathematics,* 39, pp. 13-35, 1992.
10. J. Manning, Geometric Symmetry in Graphs, *Ph.D. Thesis, Purdue Univ.,* 1990.
11. G. E. Martin, *Transformation Geometry, an Introduction to Symmetry,* Springer, New York, 1982.