元智大學 最佳化設計實驗室
OPTIMAL DESIGN LABORATORY
Copyright@2010, YZU Optimal Design Laboratory. All rights reserved.

Last updated: Yeh-Liang Hsu (2010-12-10).

Note: This is the course material for "ME550 Geometric modeling and computer graphics," Yuan Ze University. Part of this material is adapted from *CAD/CAM Theory and Practice*, by Ibrahim Zeid, McGraw-Hill, 1991. This material is be used strictly for teaching and learning of this course.

Constructive solid geometry and sweep representation

1. Introduction to Constructive Solid Geometry (CSG)

A CSG model is based on the topological notion that a physical object can be divided into a set of primitives (basic elements or shapes) that can be combined in a certain order following a set of rules (Boolean operations) to form the object. Each primitive is bounded by a set of surfaces; usually closed and orientable. A CSG model is fundamentally and topologically different from a B-rep model in that the former does not store explicitly the faces, edges, and vertices. Instead, it evaluates them whenever they are needed by applications' algorithms, e.g., generation of line drawings. The concept of primitives offers a different conceptual way of thinking that may be extended to model engineering processes such as design and manufacturing. It also appears that CSG representations might be of considerable importance for manufacturing automation as in the study of process planning and rough machining operations.

The modeling domain of a CSG scheme depends on the half-spaces that underlie its bounded solid primitives, on the available rigid motion and on the available set operators. For example, if two schemes have the same rigid motion and set operations but one has just a block and a cylinder primitive and the other has these two plus a tetrahedron, the two schemes are considered to have the same domain. Each has only planar and cylindrical half-spaces, and the tetrahedron primitive the other system offers is just a convenience to the user and does not extend its modeling domain. Extending the solid modeling domain to cover sculptured surfaces requires representing a "sculptured" half-space and its supporting utilities.

Primitives themselves are considered valid "off-the-shelf" solids. In addition, some packages, especially those that support sweeping operations, permit users to utilize wireframe entities to create faces that are swept later to create solids.

There is a wide variety of primitives available commercially to users. However, the four most commonly used are the block, cylinder, cone and sphere. These are based on the four natural quadrics: planes, cylinders, cones, and spheres. These quadrics are considered natural because they represent the most commonly occurring surfaces in mechanical design which can be produced by rolling, turning, milling, cutting drilling, and other machining operations used in industry.

From a user-input point of view and regardless of a specific system syntax, a primitive requires a set of location data, a set of geometric data, and a set of orientation data to define it completely. Primitives are usually translated and/or rotated to position and orient them properly before applying Boolean operations. Following are descriptions of the most commonly used primitives:

1. Block. This is a box whose geometrical data is its width, height, and depth. Its local coordinate system $X_L\ Y_L\ Z_L$ is shown in Figure 1. Point **P** defines the origin of the $X_L\ Y_L\ Z_L$ system. The signs of *W*, *H* and *D* determine the position of the block relative to its coordinate system. For example, a block with a negative value of *W* is displayed as if the block shown in Figure 1 is mirrored about the $Y_L Z_L$ plane.
2. Cylinder. This primitive is a right circular cylinder whose geometry is defined by its radius (or diameter) *R* and length *H* The length *H* is usually taken along the direction of the Z_L axis. *H* can be positive or negative.
3. Cone. This is a right circular cone or a frustum of a right circular cone whose base radius *R* top radius (for truncated cone), and height *H* are user-defined.
4. Sphere. This is defined by its radius and is centered about the origin of its local coordinate system.
5. Wedge. This is a right-angled wedge whose height *H* width *W* and base depth *D* form its geometric data.
6. Torus. This primitive is generated by the revolution of a circle about an axis lying in its plane (Z_L axis in Figure 1). The torus geometry can be defined by the radius (or diameter) of its body R_1 and the radius (or diameter) of the centerline of the torus body R_2, or the geometry can be defined by the inner radius (or diameter) R_I and outer radius (or diameter) R_O.

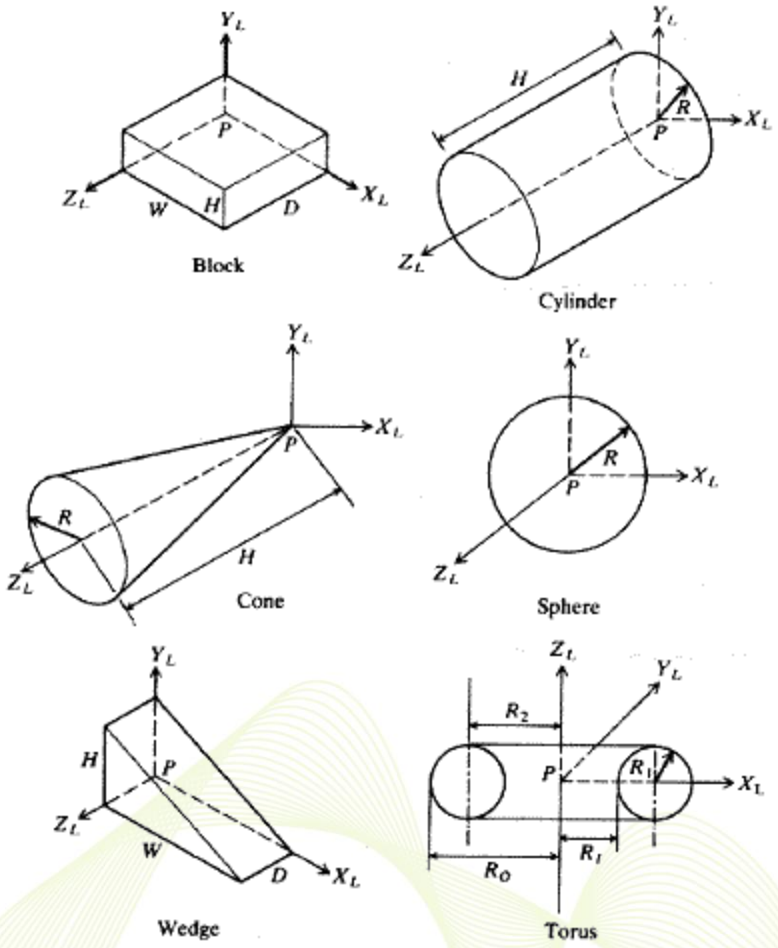
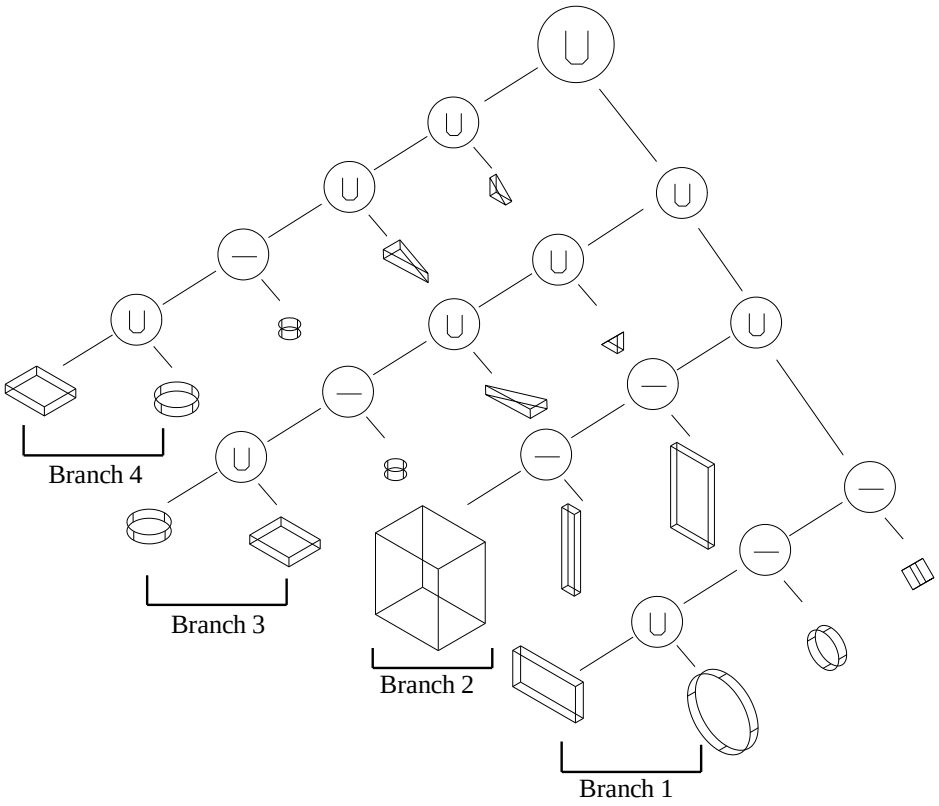


Figure 1. Most common primitives

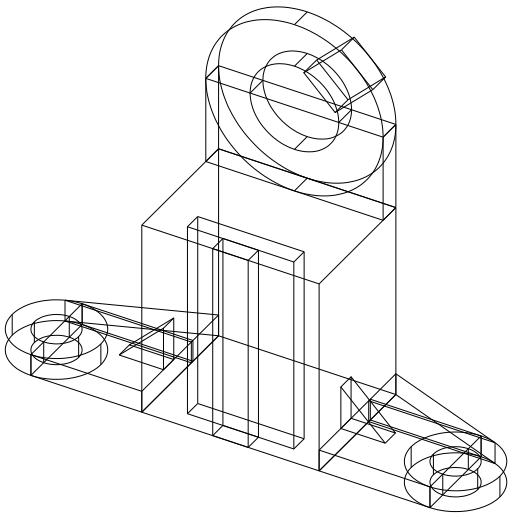
The available boolean operators are union (∪ or +), intersection (∩ or I), and difference (－). Figure 2 shows a solid model of a guide bracket constructed by primitives and boolean operators.

◇Assignment 1

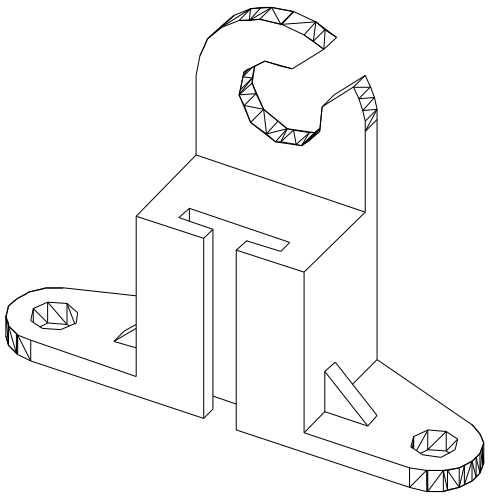
Construct a meaningful solid object by primitives and boolean operators using your CAD software. Record the procedure of constructing the object step by step. ◇



(a) Construction tree



(b) Primitives in their proper locations and orientations



(c) Final solid model

Figure 2. Solid model of the guide bracket

2. Regularized set operation

Surface/surface intersection is very crucial in geometric modeling in general and in solid modeling in particular. It is a decisive factor in determining and/or limiting the

modeling domain of a solid modeler. As a matter of fact, it is the only factor in slowing down the implementations of sculptured surfaces into solid modeling.

A solid model of an object is defined mathematically as a point set S in three-dimensional Euclidean space (E^3). If we denote the interior and boundary of the set by iS and bS respectively, we can write

$$S = \mathrm{i}S \cup \mathrm{b}S \tag{1}$$

And if we let the exterior be defined by cS (complement of S), then

$$W = \mathrm{i}S \cup \mathrm{b}S \cup \mathrm{c}S \tag{2}$$

where W is the universal set, which in the case of E^3 is all possible three-dimensional points.

Geometric closure implies that the interior of the solid is geometrically closed by its boundaries.

$$S = \mathrm{k}S \tag{3}$$

where kS is the closure of the solid or point set S and is given by the right-hand side of Eq. (1), that is, $\mathrm{k}S = \mathrm{i}S \cup \mathrm{b}S$.

It should be noted here that both wireframe and surface models lack geometric closure which is the main reason for their incompleteness and ambiguity.

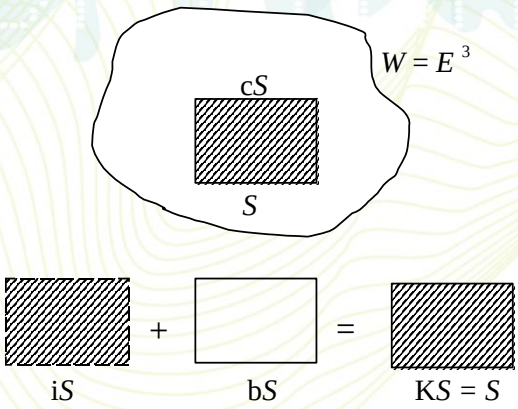


Figure 3. Solid and geometric closure definitions

A regular set is defined as a set that is geometrically closed. The notion of a regular set is introduced in geometric modeling to ensure the validity of objects they represent and therefore eliminate nonsense objects. Under geometric closure, a regular set has interior and boundary subsets.

A set S is regular if and only if

$$S = \mathrm{ki}S \tag{4}$$

This equation states that if the closure of the interior of a given set yields that same given set, then the set is regular. Figure 4(a) shows that set S is not regular because $S' = \mathrm{ki}S$ is not equal to S. A set S is regular open if and only if

$$S = \mathrm{ik}S \tag{5}$$

This equation states that a set is regular open if the interior of its closure is equal to the original set. Figure 4(b) shows that S is not regular open because $S' = \mathrm{ik}S$ is not equal to S.

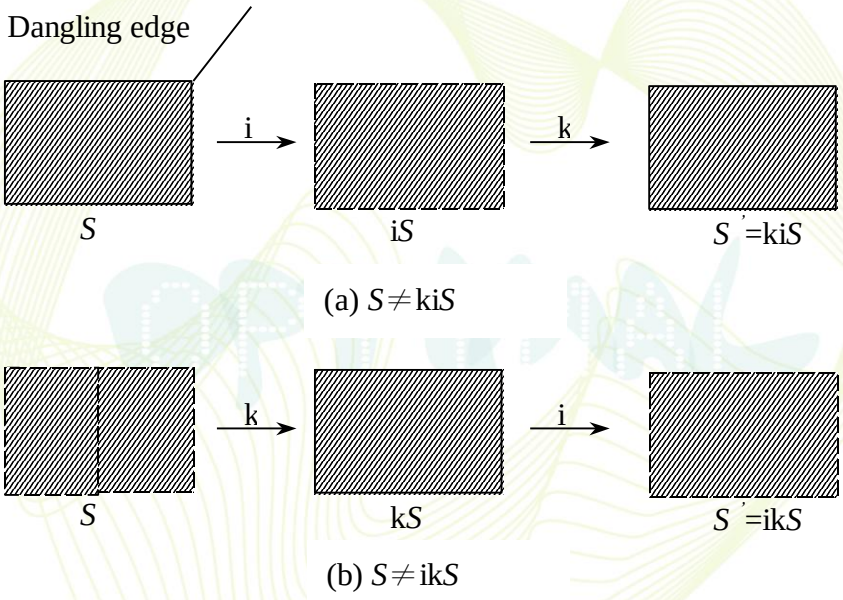


Figure 4. Set regularity

Set operations (known also as Boolean operators) must be regularized to ensure that their outcomes are always regular sets. For geometric modeling, this means that solid models built from well-defined primitives are always valid and represent valid (no-nonsense) objects. Regularized set operators preserve homogeneity and spatial

dimensionality. The former means that no dangling parts should result from using these operators and the latter means that if two three-dimensional objects are combined by one of the operators, the resulting object should not be of lower dimension (two or one dimension).

Based on the above description, regularized set operators can be defined as follows:

$$P \cup^* Q = \mathrm{ki}(P \cup Q) \tag{6}$$

$$P \cap^* Q = \mathrm{ki}(P \cap Q) \tag{7}$$

$$P -^* Q = \mathrm{ki}(P - Q) \tag{8}$$

$$\mathrm{c}^* P = \mathrm{ki}(\mathrm{c}P) \tag{9}$$

where the superscript "*" to the right of each operator denotes regularization. Figure 5 shows examples of using regularized set operations.

◇Assignment 2

Make up 3 examples similar to those in Figure 5 to show that invalid objects would occur if conventional boolean operators are used. Apply regularized set operators to these 3 examples and show that regularized set operators would ensure the validity of the objects. ◇

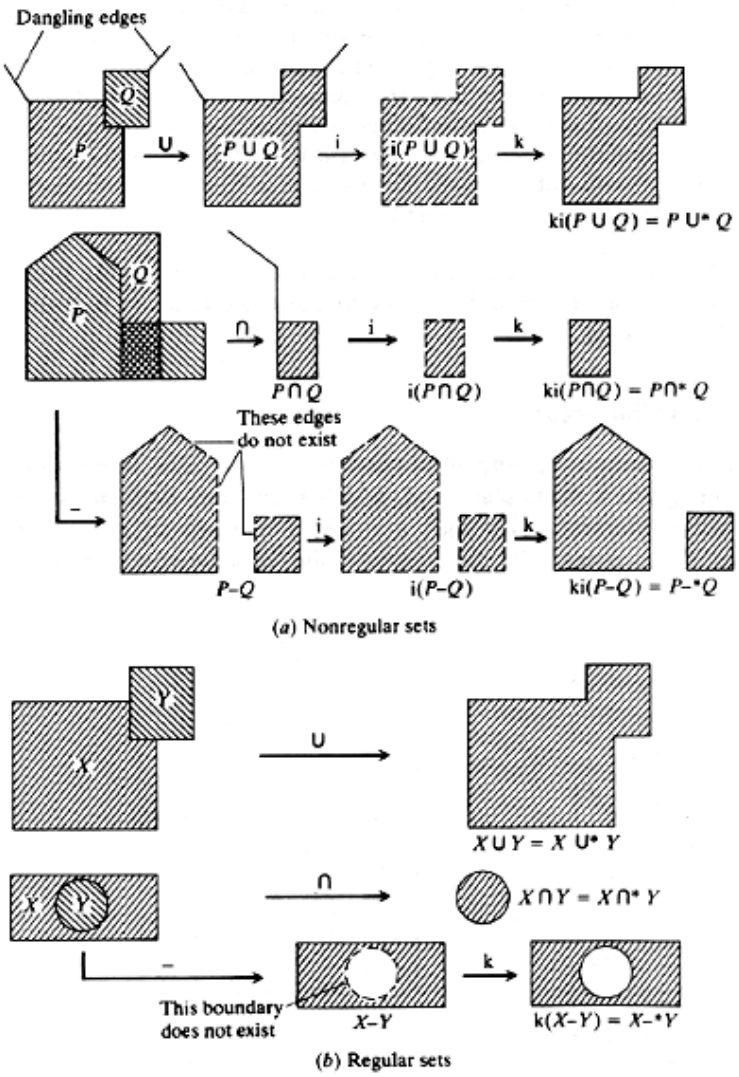


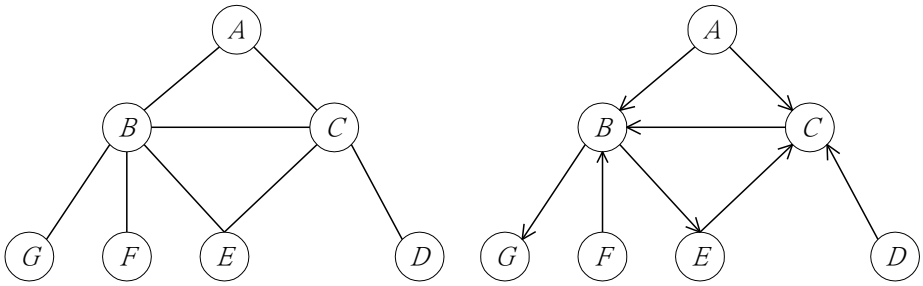
Figure 5. Regularized set operators

3. Graphs and Trees

The database of a CSG model, similar to B-rep, stores its topology and geometry. Topology is created via the regularized set (Boolean) operations that combine primitives. The geometry stored in the database of a CSG model includes configuration parameters of its primitives and rigid motion and transformation. Data structures of most CSG representations are based on the concept of graphs and trees.

A graph is defined as a set of nodes connected by a set of branches or lines. Each branch in a graph is specified by a pair of nodes. As shown in Figure 6, if the pairs of nodes that make up the branches are ordered pairs, the graph is said to be a directed graph or digraph. This means that branches have directions in a digraph and become in a sense

arrows going from one node to another. The tail of each arrow represents the first node in the pair and its head represents the second node.



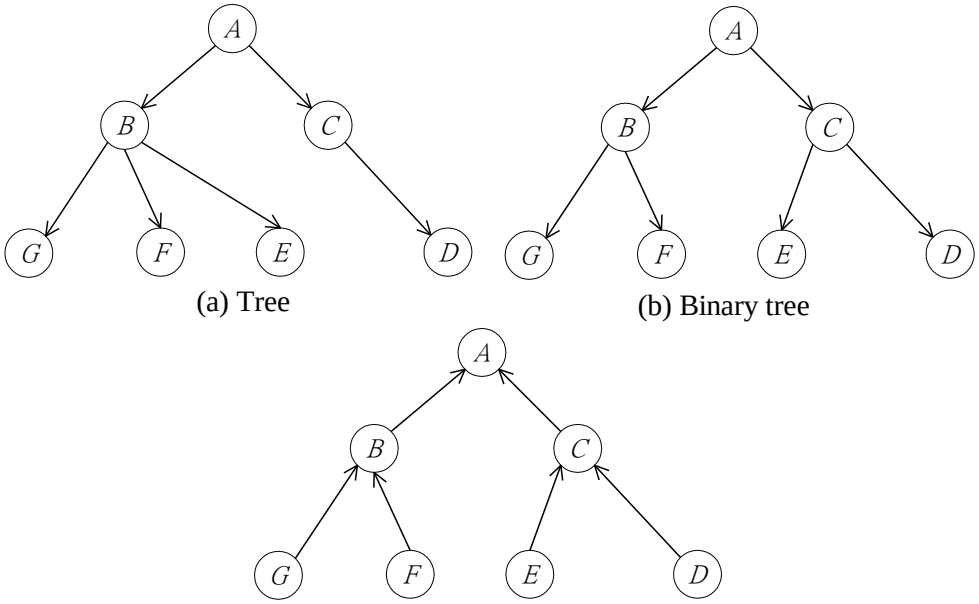
(*a*) Graph (*b*) Digraph

Figure 6. Graphs and digraphs

Each node in a digraph has an indegree and outdegree and has a path it belongs to. The indegree of a node is the number of arrow heads entering the node and its outdegree is the number of arrow tails leaving the node.

Each node in a digraph belongs to a path. A path from node *n* to node *m* is defined as a sequence of nodes $n_1, n_2, \cdots, n_k$ such that $n_1 = n$ and $n_k = m$, and any two subsequent nodes (n_i, n_{i+1}) in the sequence are adjacent to each other. If the start and end nodes of a path are the same, the path is a cycle. If a graph contains a cycle, it is cyclic; otherwise it is acyclic.

A tree is defined as an acyclic digraph in which only a single node, called the root, has a zero indegree and every other node has an indegree of 1. This implies that any node in the tree except the root has predecessors or ancestors. Based on this definition, a graph need not be a tree but a tree must be a graph. Moreover, when each node of an ordered tree has two descendants (left and right), the tree is called a binary tree. Finally, if the arrow directions in a binary tree are reversed such that every node, except the root, in the tree has an outdegree of 1 and the root has a zero outdegree, the tree is called an inverted binary tree. An inverted binary tree is very useful to understand the data structure of CSG models (sometimes called Boolean models). Any node in a tree that does not have descendants, that is, with an outdegree equal to zero, is called a leaf node and any node that does have descendants (outdegree greater than zero) is an interior node. Figure 7 shows the type of trees.



(a) Tree (b) Binary tree

(c) Inverted binary tree

Figure 7. Types of trees

4. CSG Tree

Figure 8 shows a typical solid and its building primitives. This solid can be built following the steps below:

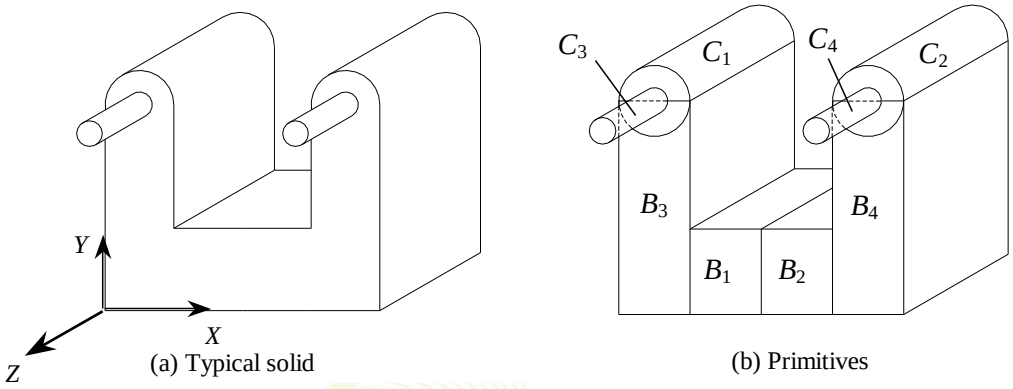
$$\left.\begin{array}{l} B_1 = block \quad positioned \quad properly \\ B_2 = block \quad positioned \quad properly \\ B_3 = block \\ B_4 = B_3 \quad moved \quad properly \quad in \quad the \quad X \quad direction \\ C_1 = cylinder \quad positioned \quad properly \\ C_2 = C_1 \quad moved \quad properly \quad in \quad the \quad X \quad direction \\ C_3 = cylinder \quad positioned \quad properly \\ C_4 = C_3 \quad moved \quad properly \quad in \quad the \quad X \quad direction \end{array}\right\} \text{Primitives' definitions}$$

$$\left.\begin{array}{l} S_1 = B_1 \quad \cup * \quad B_3 \\ S_2 = S_1 \quad \cup * \quad C_1 \\ S_3 = S_2 \quad \cup * \quad C_3 \end{array}\right\} Construct \quad left \quad half$$

$$\left.\begin{array}{l} S_4 = B_2 \quad \cup * \quad B_4 \\ S_5 = C_2 \quad \cup * \quad S_4 \\ S_6 = C_4 \quad \cup * \quad S_5 \end{array}\right\} Construct \quad right \quad half$$

$$S = S_3 \quad \cup * \quad S_6 \} Model$$

To save the above steps in a data structure, such a structure must preserve the sequential order of the steps as well as the order of the Boolean operations in any step; that is, the left and right operands of a given operator. The ideal solution is a digraph; call it a CSG graph. A CSG graph is a symbolic (unevaluated) representation and is intimately related to the modeling steps used by the user.



(a) Typical solid (b) Primitives

Figure 8. A typical solid and its building primitives

A CSG tree is defined as an inverted ordered binary tree whose leaf nodes are primitives and interior nodes are regularized set operations. The creation of a balanced, unbalanced, or a perfect CSG tree depends solely on the user and how he/she decomposes a solid into its primitives. The general rule to create balanced trees is to start to build the model from an almost central position and branch out in two opposite directions or vice versa. Another useful rule is that symmetric objects can lead to perfect trees if they are decomposed properly. Figure 9 shows a perfect CSG tree and Figure 10 shows an umbalance CSG tree.

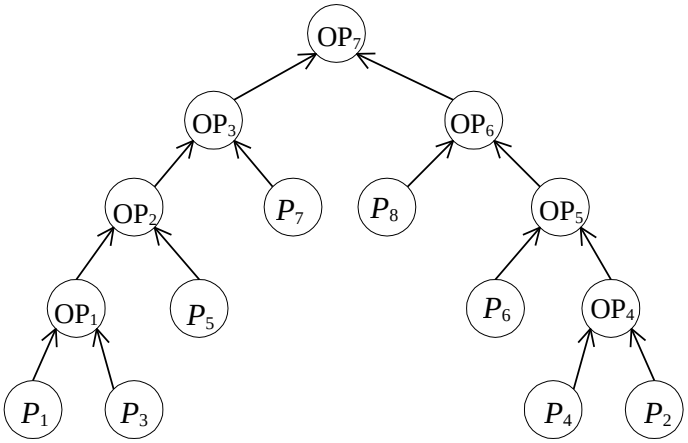


Figure 9. CSG tree of a typical solid

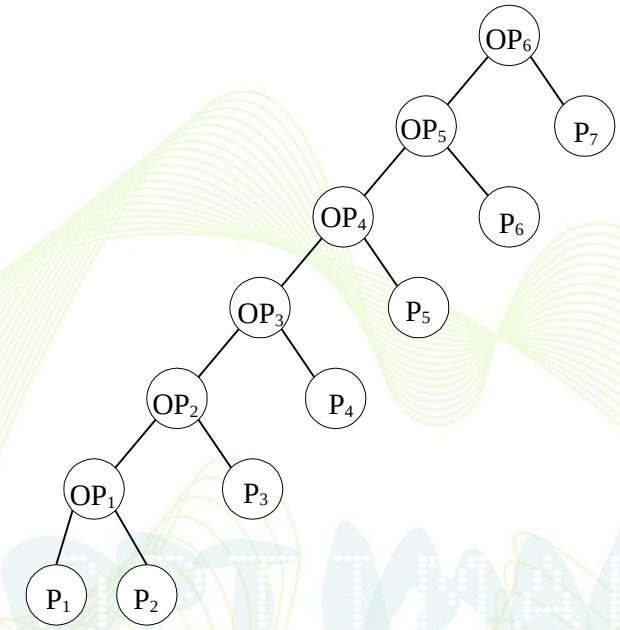


Figure 10. An unbalanced CSG tree

◇Assignment 3

Draw the CSG tree for the object you created in Assignment 1. ◇

Application algorithms must traverse a CSG tree, that is, pass through the tree and visit each of its nodes. Also traversing a tree in a certain order provides a way of storing a data structure. The order in which the nodes are visited in a traversal is clearly from the first node to the last one. However, there is no such natural linear order for the nodes of a tree. Thus different orderings are possible for different cases. There exist three main traversal methods. The three methods are preorder, inorder, and postorder

traversals. Sometimes, these methods are referred to as prefix, infix, and postfix traversals.

To traverse a tree in preorder (Figure 11), we perform the following three operations:

(1) Visit the root.
(2) Traverse the left subtree in preorder.
(3) Traverse the right subtree in preorder.

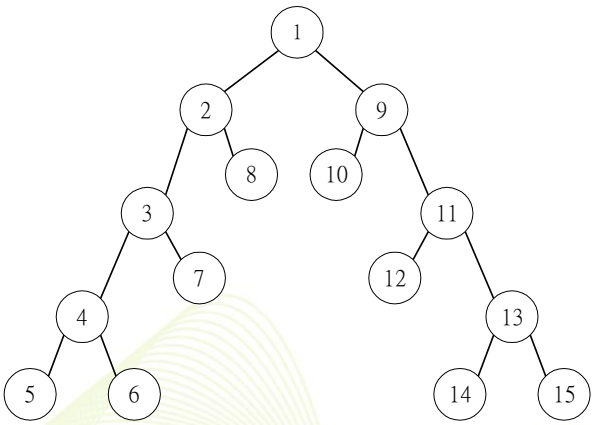


Figure 11. Preorder traversals of a tree.

To traverse a tree in inorder (Figure 12):

(1) Traverse the left subtree in inorder.
(2) Visit the root.
(3) Traverse the right subtree in inorder.

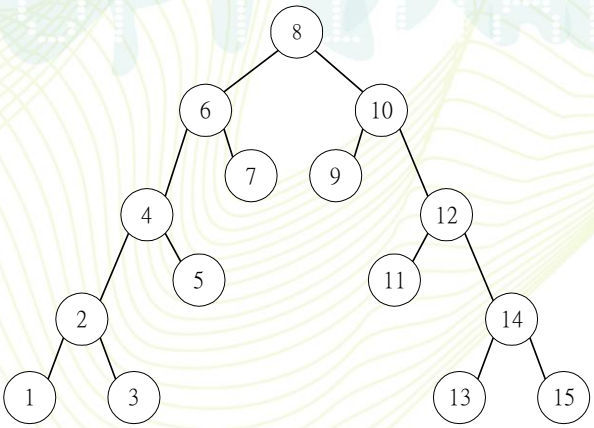


Figure 12. Inorder traversals of a tree.

To traverse a tree in postorder (Figure 13):

(1) Traverse the left subtree in postorder.
(2) Traverse the right subtree in postorder.
(3) Visit the root.

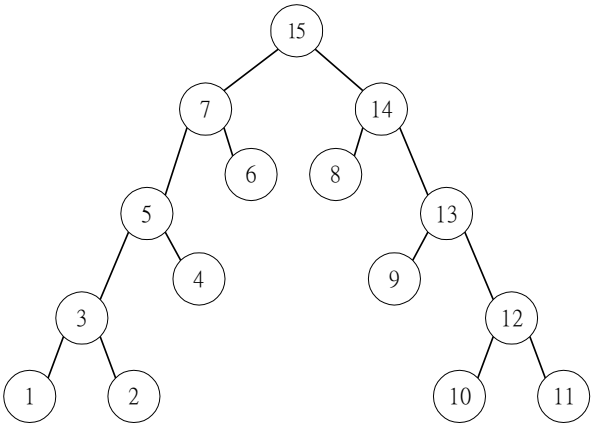


Figure 13. Postorder traversals of a tree.

Which of the traversal methods is more suitable to store a tree in a solid modeler? In arithmetic expressions, eg., $A+(B+C)\times D$, the order of operations in an infix expression might require cumbersome parentheses while a prefix form requires scanning the expression from right to left. Since most algebraic expressions are read from left to right, postfix is a more natural choice.

◇Assignment 4

Convert Figure 9 into the CSG tree of the object in Figure 8. Use preorder, inorder, and postorder to traverse the CSG tree to create expressions similar to $A+(B+C)\times D$. ◇

5. Sweep representation

Solid models can also be built up from **sweeps**. Schemes based on sweep representation are useful in creating solid models of two-and-a-half-dimensional objects. The class of two-and-a-half-dimensional objects includes both solids of uniform thickness in a given direction and axisymmetric solids. The former are known as extruded solids and are created via linear or translational sweep; the latter are solids of revolution which can be created via rotational sweep.

Sweeping is based on the notion of moving a point, curve, or a surface along a given path. There are three types of sweep: linear, nonlinear, and hybrid sweeps. In

linear sweep, the path is a linear or circular vector described by a linear, most often parametric, equation while in nonlinear sweep, the path is a curve described by a higher-order equation (quadratic, cubic, or higher). Hybrid sweep combines linear and/or nonlinear sweep via set operations and is, therefore, a means of increasing the modeling domain of sweep representations.

Linear sweep can be divided further into translational and rotational sweep. In translational sweep, a planar two-dimensional point set described by its boundary (or contour) can be moved a given distance in space in a perpendicular direction (called the directrix) to the plane of the set. Nonlinear sweep is similar to linear sweep but with the directrix being a curve instead of a vector. Hybrid sweep tends to utilize some form of set operations. Figure 14 shows various types of sweep.

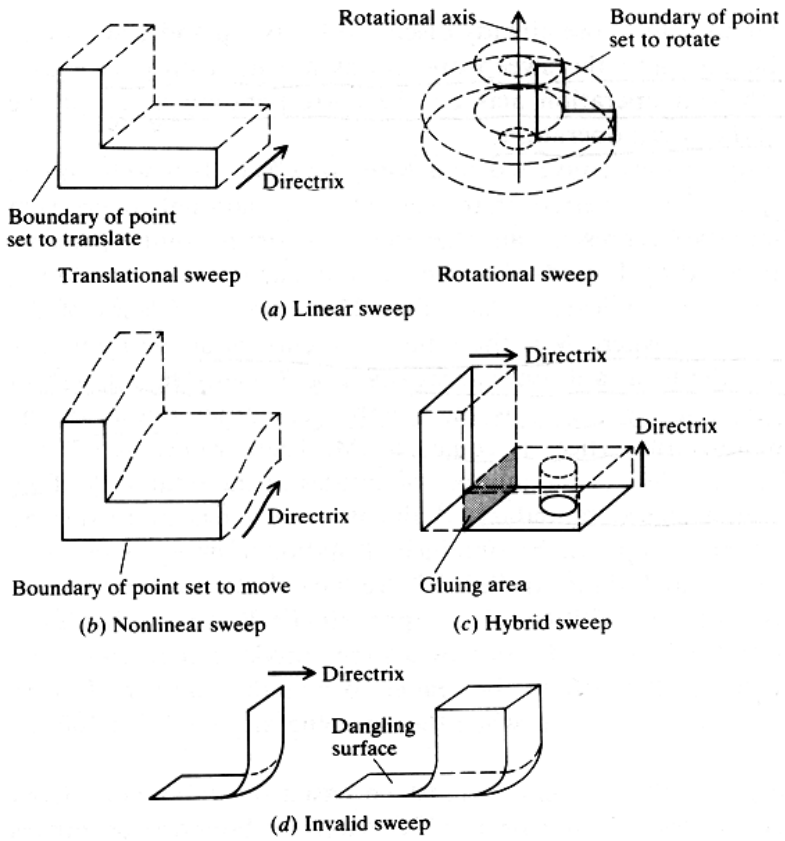


Figure 14. Types of sweep.

The building operations of linear and nonlinear sweep models are simple: generate the boundary and sweep it. Sweep representation is often called "**constraint-based solid modeling**" because the shape of the initial 2-D sketch provides the constraint to the added dimension. **This modeling technique is considered more intuitive than CSG and is**

popular in CAD software. If hybrid sweep is available, these operations extend to include Boolean operations.

◇Assignment 5

Use your CAD software to generate examples of translational sweep, rotational sweep, and, if possible, nonlinear sweep and hybrid sweep. ◇

◇Assignment 6

In your CAD software, can you perform boolean operations on objects generated by sweep? If so, display some examples. ◇