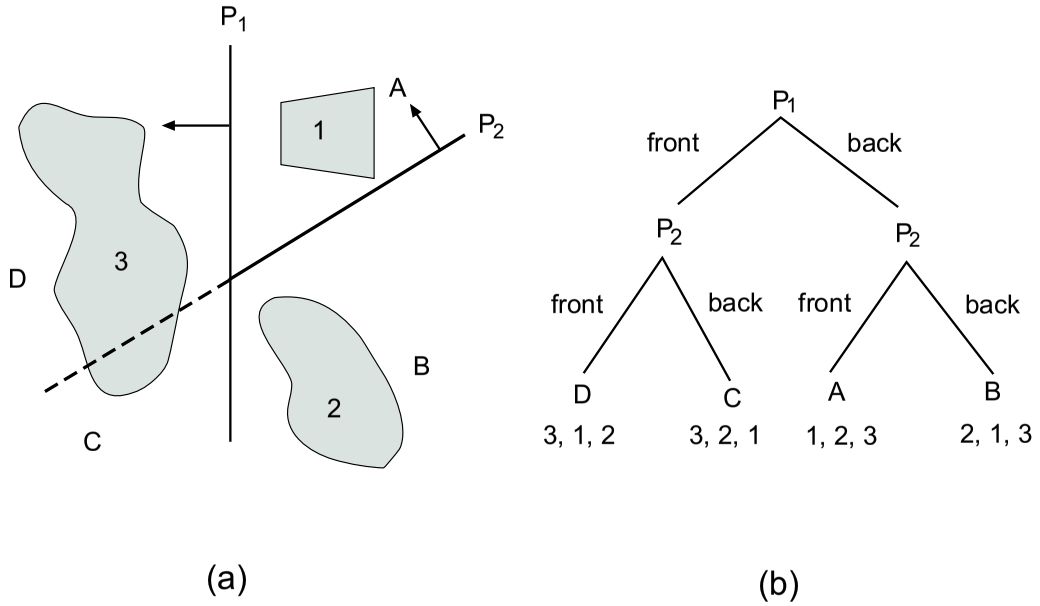
# Binary Space Partitioning Trees

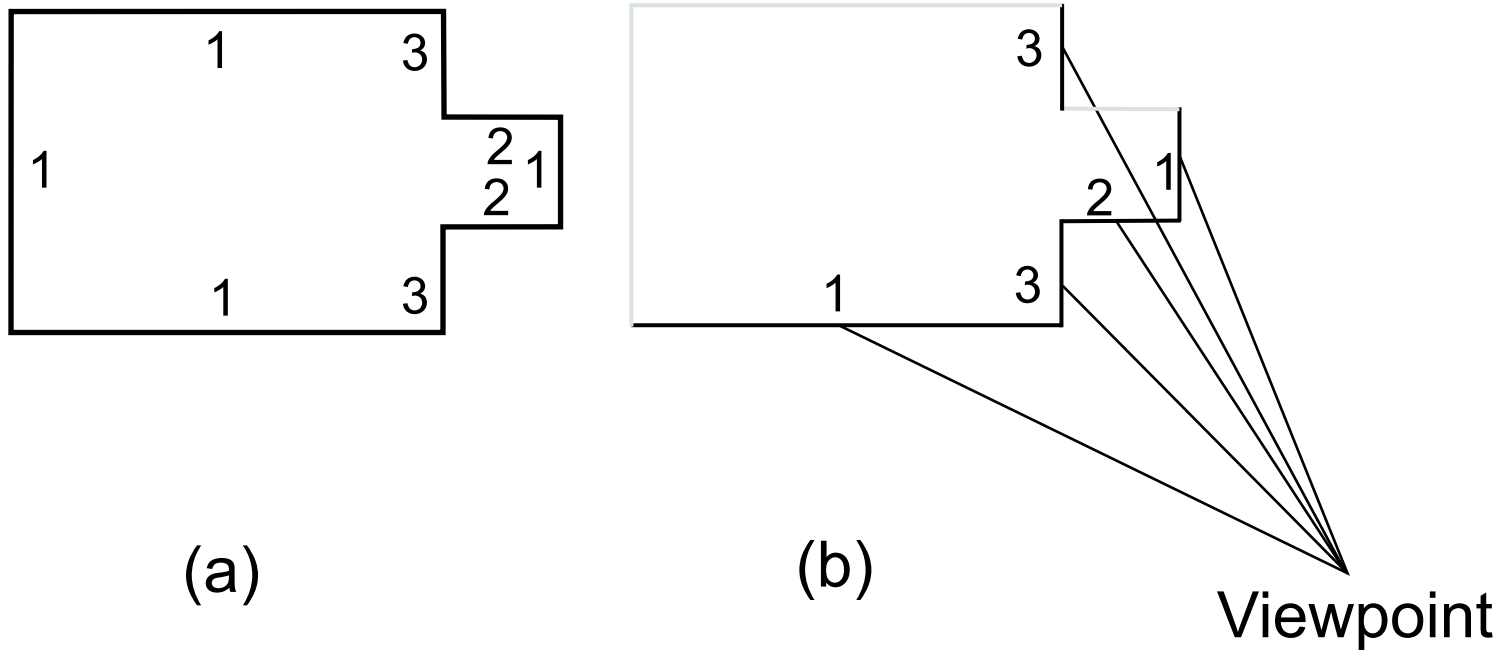
*Cluster priority*




(a) Clusters 1 through 3 are divided by partitioning planes $P_1$ and $P_2$, determing regions $A$ through $D$ in which the eyepoint may be located. Each region has a unique cluster priority.

(b) The binary-tree representation of (a).

*Face priority*




(a) Faces in a cluster and their priorities. A lower number indicates a higher priority.

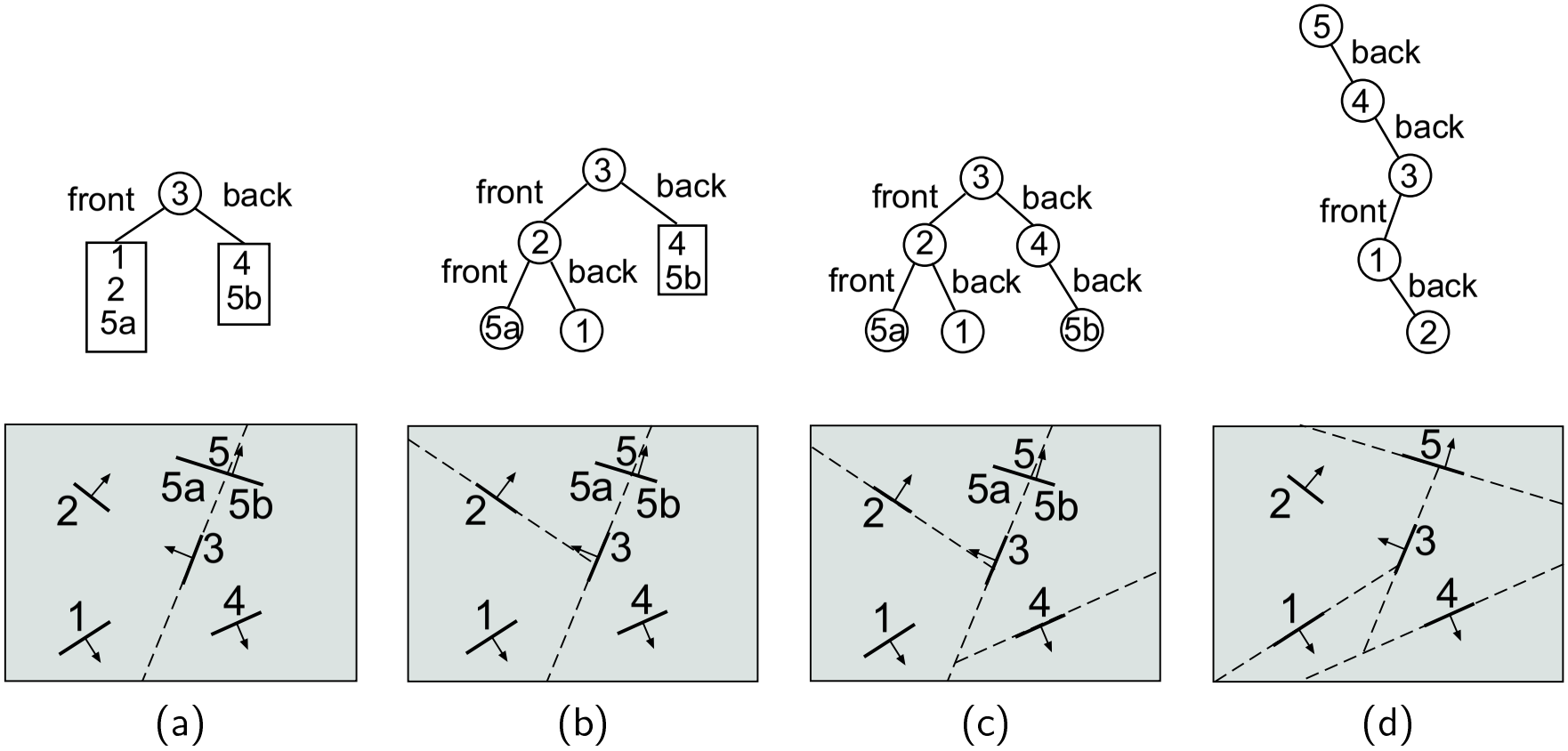
(b) Priorities of visible faces.

*Pseudocode for building a BSP tree*

```
type
   BSP_tree = record
      root : polygon;
      backChild, frontChild : ^ BSP_tree
   end;
function BSP_makeTree ( polyList : listOfPolygons ) : ^ BSP_tree;
var
   root : polygon;
   backList, frontList : listOfPolygons;
   p, backPart, frontPart : polygon {We assume each polygon is convex.}
begin
   if polyList is empty then
      BSP_makeTree := nil
   else
      begin
         root := BSP_selectAndRemovePoly ( polyList );
         backList := nil;
         frontList := nil;
```

```
    for each remaining polygon p in polyList
      begin
        if polygon p in front of root then
          BSP_addToList( p, frontList )
        else if polygon p in back of root then
          BSP_addToList ( p, backList )
        else { Polygon p must be split.}
          begin
            BSP_splitPoly ( p, root, frontPart, backPart );
            BSP_addToList ( frontPart, frontList );
            BSP_addToList ( backPart, backList );
          end
      end;
    BSP_makeTree := BSP_combineTree ( BSP_makeTree ( frontList ),
                                      root, BSP_makeTree( backList ))
  end
end;
```

*BSP Tree*


front
3
back
1
2
5a
4
5b
front
3
back
front
2
back
4
5b
5a
1
front
3
back
front
2
4
back
back
5a
1
5b
5
back
4
back
3
front
1
back
2
5
5a
5b
2
3
1
4
(a)
(b)
(c)
(d)

(a) Top view of scene with BSP tree before recursion with polygon 3 as root.

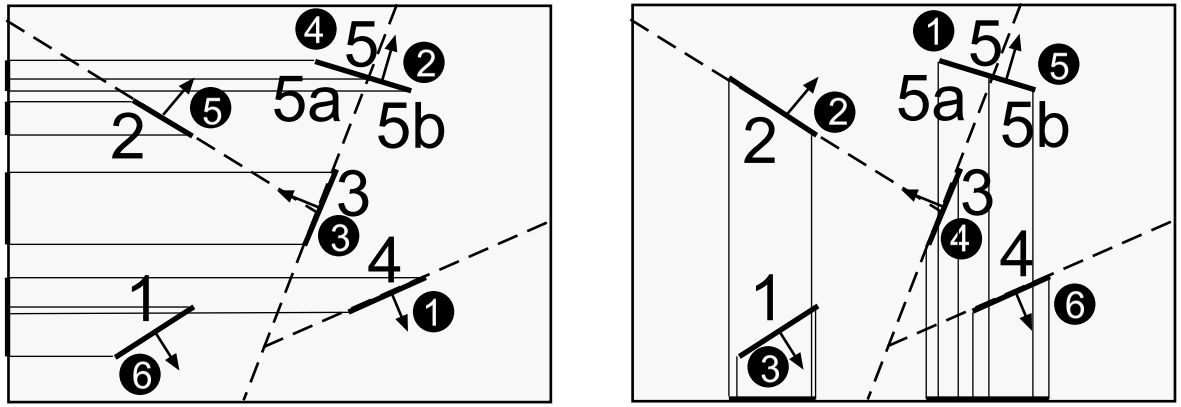
(b) After building left subtree.

(c) Complete tree.

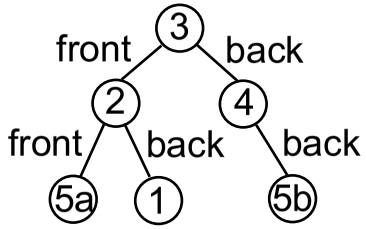
(d) Alternate tree with polygon 5 as root.

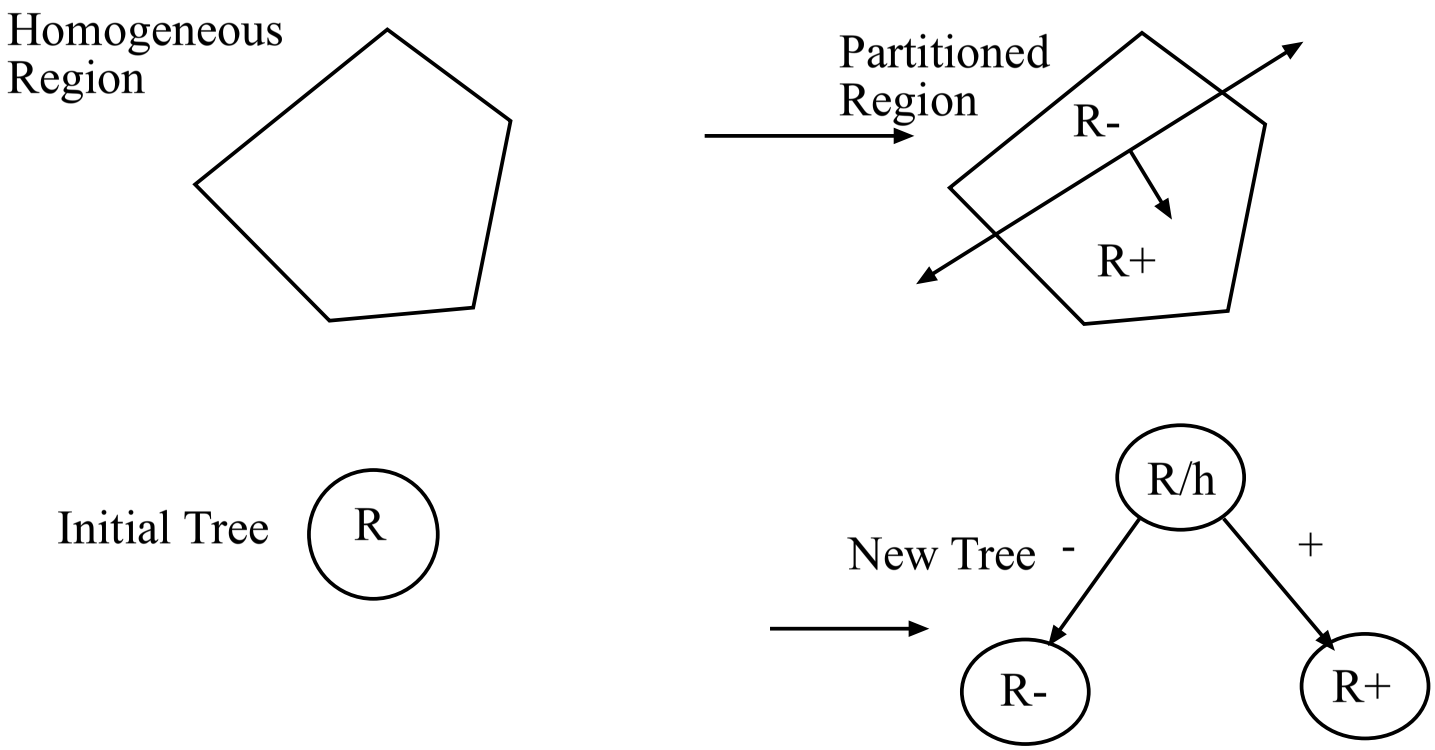
*Pseudocode for displaying a BSP tree*

```
procedure BSP_displayTree ( tree : ^ BSP_tree );
begin
  if tree is not empty then
    if viewer is in front of root then
      begin
        { Display back child, root, and front child. }
        BSP_displayTree ( tree ^ .backChild );
        displayPolygon ( tree ^ .root );
        BSP_displayTree ( tree ^ .frontChild )
      end
    else
      begin
        { Display front child, root, and back child. }
        BSP_displayTree ( tree ^ .frontChild );
        displayPolygon ( tree ^ .root );        { Only if back-face culling not desired}
        BSP_displayTree ( tree ^ .backChild )
      end
end;
```

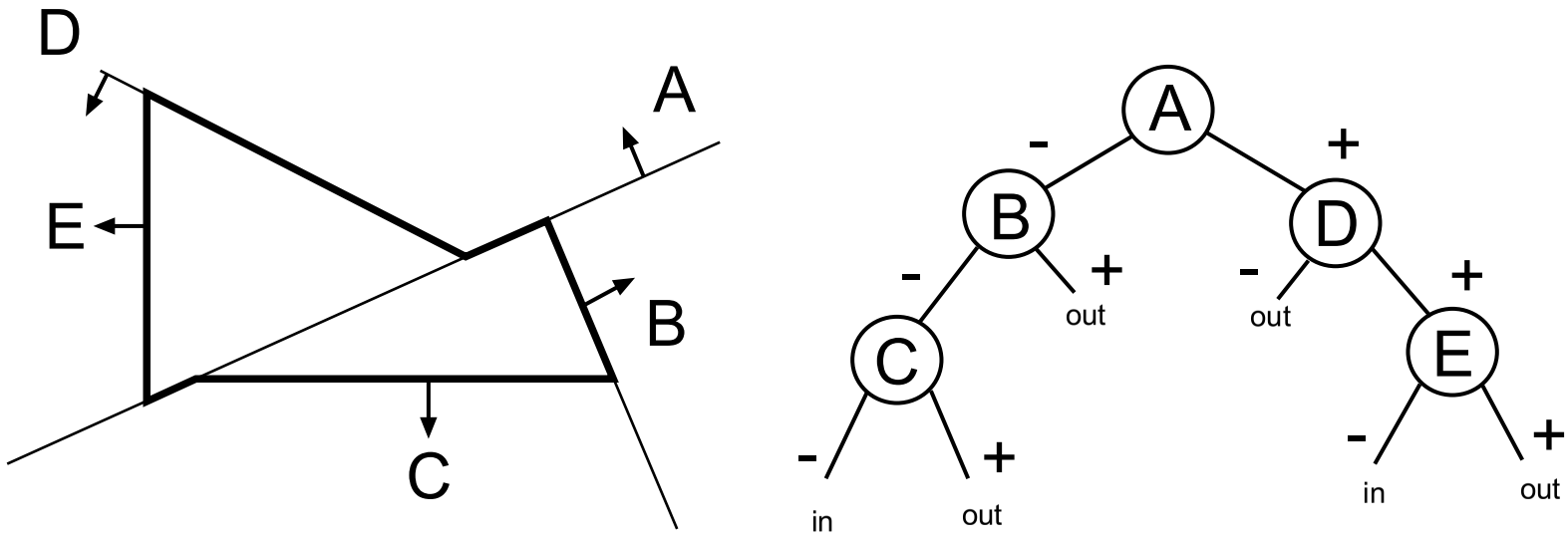



Two traversals of the BSP tree corresponding to two different projections. Projectors are shown as thin lines. White numbers indicate drawing order.






Elementary operation used to construct Partitioning Trees




Example intra-object Partitioning Tree