CSE251
Basics of Computer Graphics
Module: Visibility and Culling Module

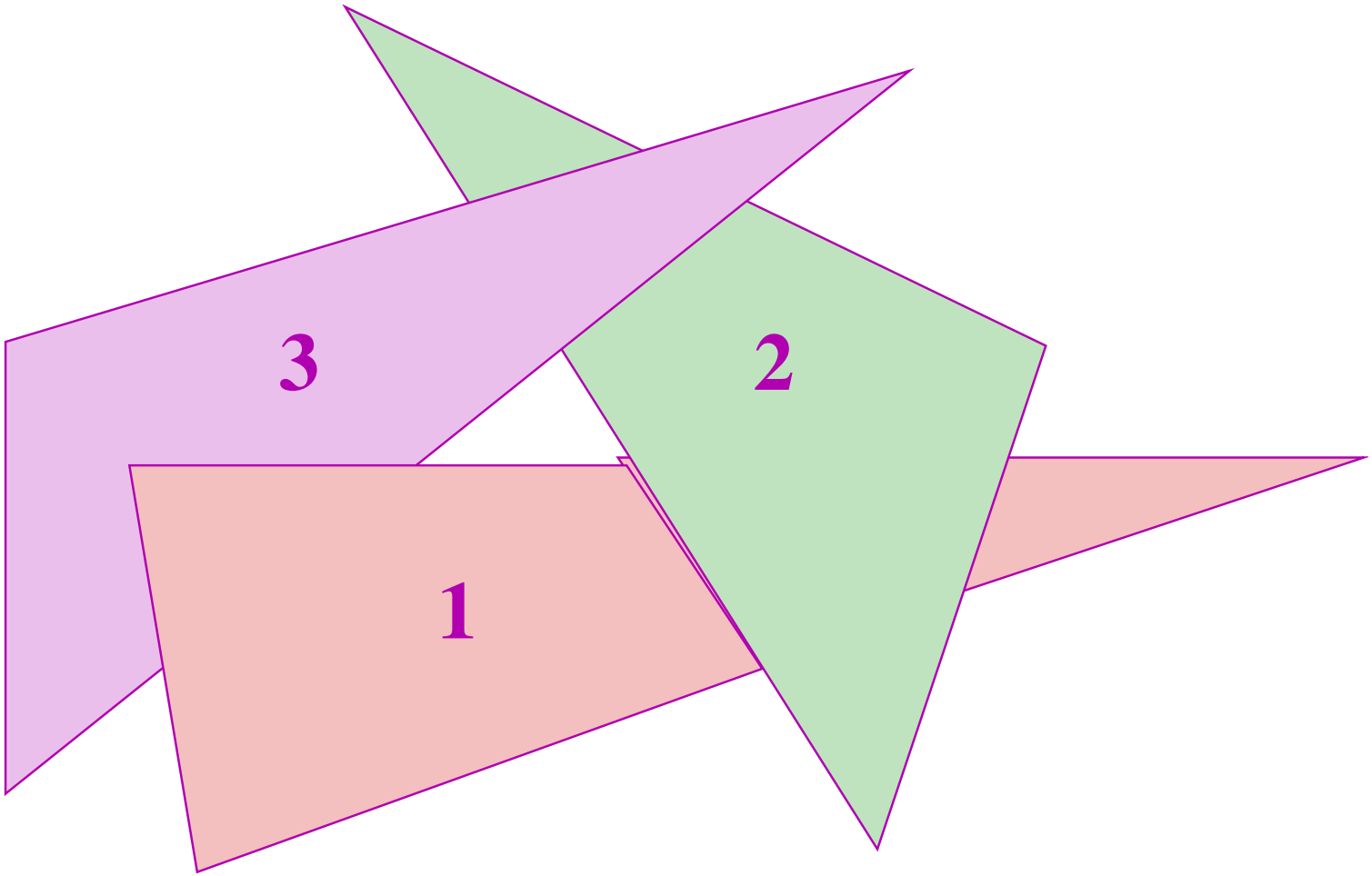
Avinash Sharma

Spring 2018

List Priority Algorithms

- Reorder objects such that the correct picture results if you draw them in that order. If objects do not overlap in z, draw them from back to front.
- Objects may need splitting if no unique ordering exists.
- **Needs expensive sorting/reordering every frame!!**
- Ordering and splitting polygons: Object-precision operation.
- Overwriting farther points while scan conversion: Image-precision operation.

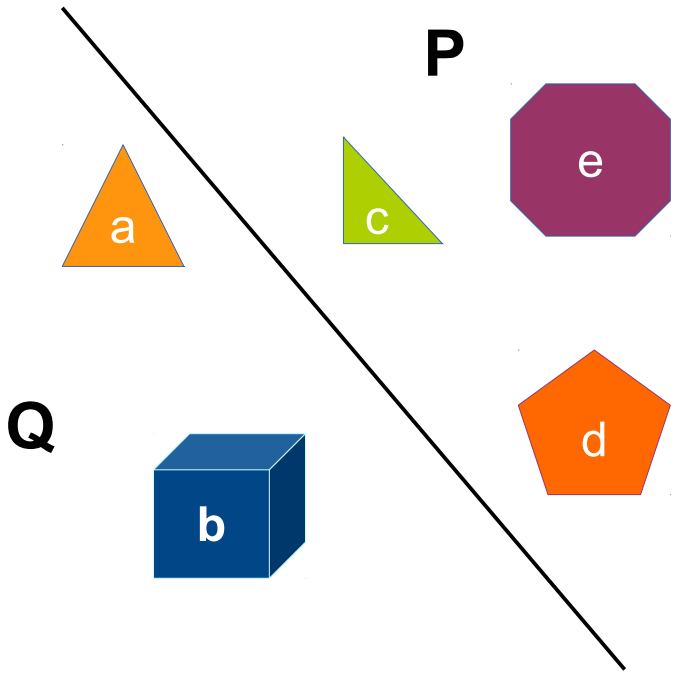
A Difficult Case for Ordering



Binary Space Partitioning Trees

- Can we have simple linear display algorithm, perhaps using expensive preprocessing?
- Consider a plane in space that divides scene into two halfs.
- Objects on the same side of the plane as the eye cannot be blocked by objects on the other side.

Binary Space Partitioning Trees

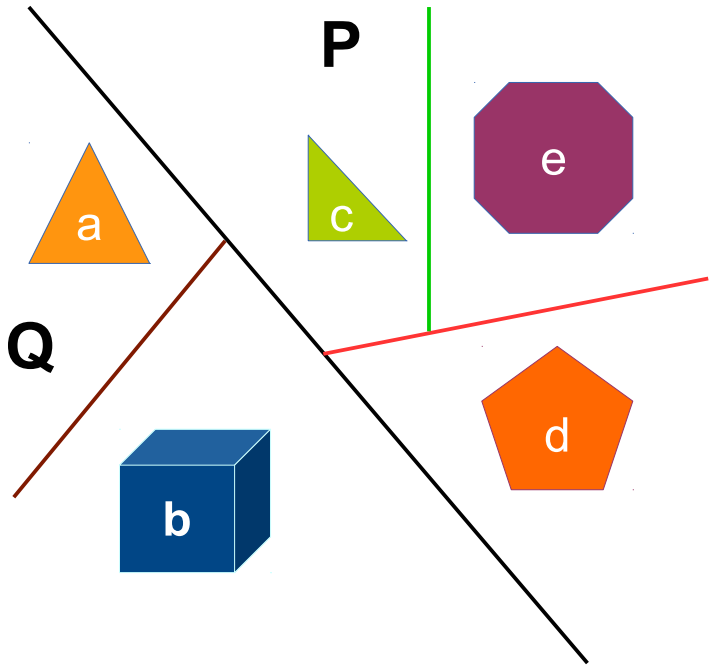


- Ordering of **b** and **c** from viewpoint **P**? From **Q**?

Binary Space Partitioning Trees

- Each side of the plane can further be divided using other planes till we reach a single object.
- If enviroments consist of clusters of objects, separate them using an appropriate plane.
- We end up with the **BSP Tree** representation of the scene.
- Internal nodes contain partitioning planes; leaf nodes are polygons.
- Some preprocessing to construct the tree, but simple algorithm to render using it from any viewpoint.

Binary Space Partitioning Trees



- Total ordering from viewpoint **P**? From **Q**?

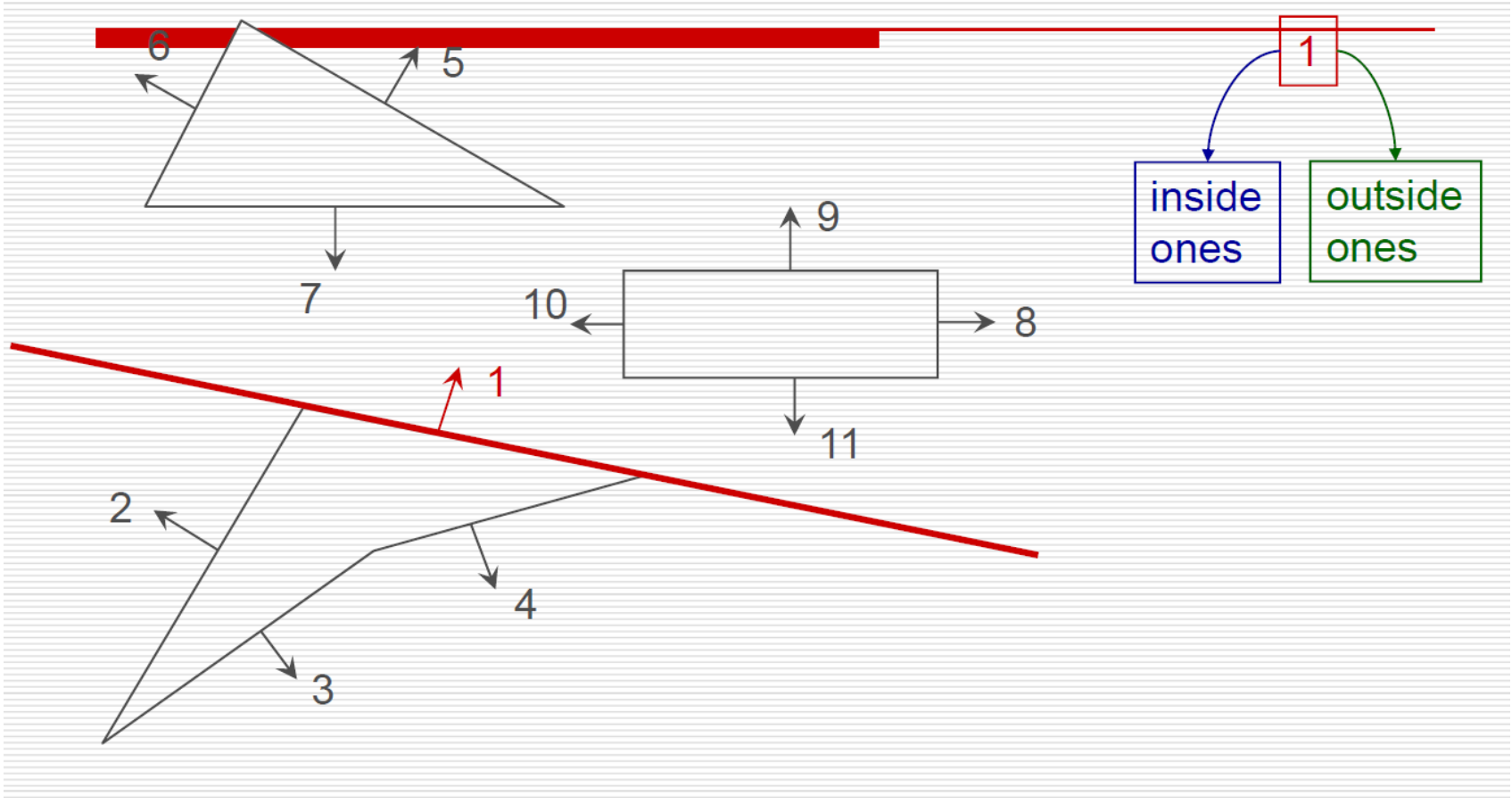
BSP Trees

- Use planes of polygons in the scene as partitioning planes.
- Normal direction indicates the “front” side of the plane.
- Each plane divides space into two sides.
- If a polygon lies on both sides of the plane, divide it into two parts.
- Continue this recursively till each side contains exactly one polygon.

Psuedocode: BSP Tree Construction

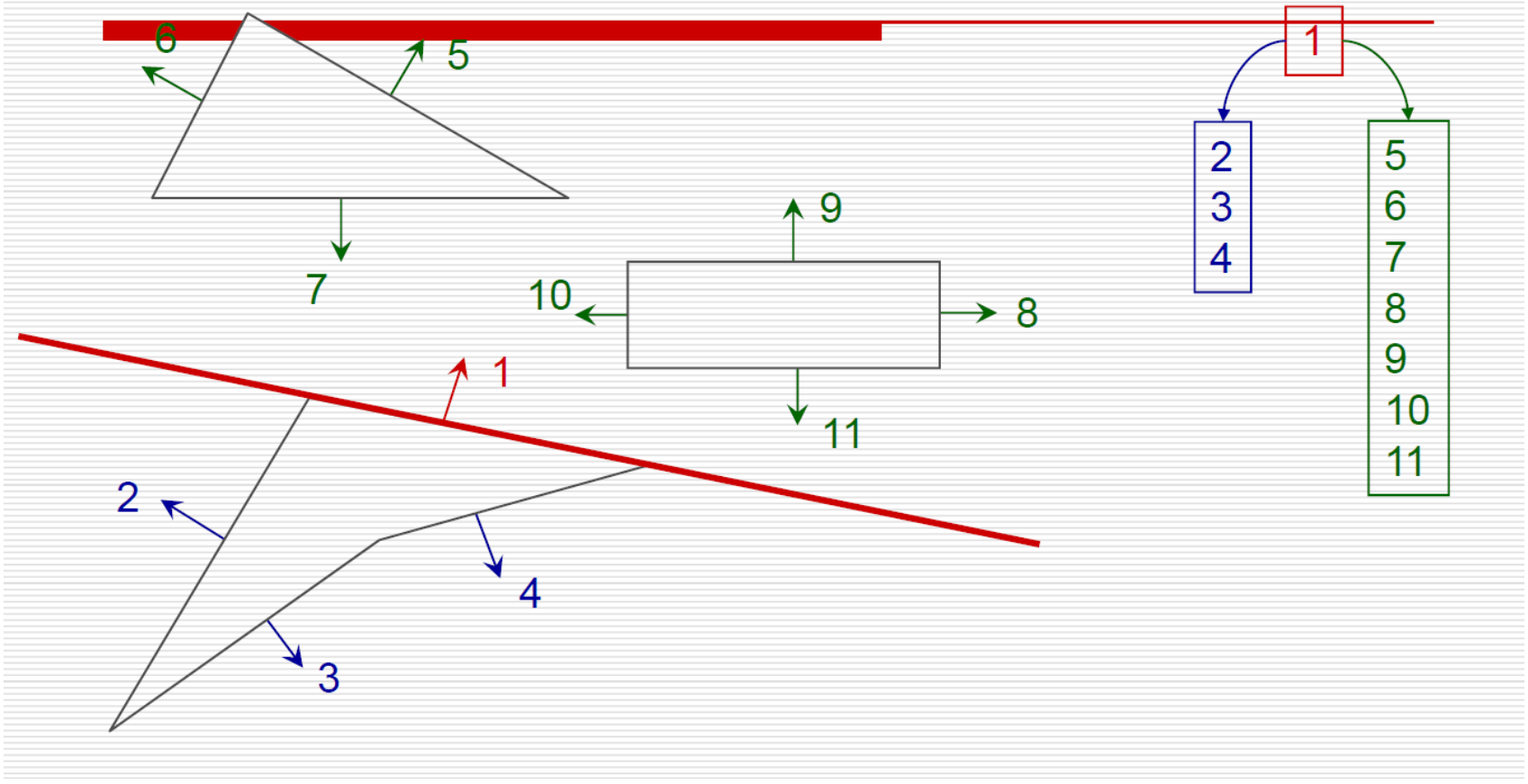
```
makeBSPTree(pList){
    if (pList is empty)
        return NULL
    end
    root ← selAndRemove(pList);
    bList, fList ← NULL
    for each polygon p in pList
        if (p is in front of root)
            addToList(p, fList)
        elsif (p is in back of root)
            addToList(p, bList)
        else
            splitPoly(p, fp, bp)
            addToList(fp, fList);
            addToList(bp, bList)
            return combineTree(makeBSPTree(fList), root, makeBSPTree(bList))
        end
    end
}
```

Example:BSP Tree Construction

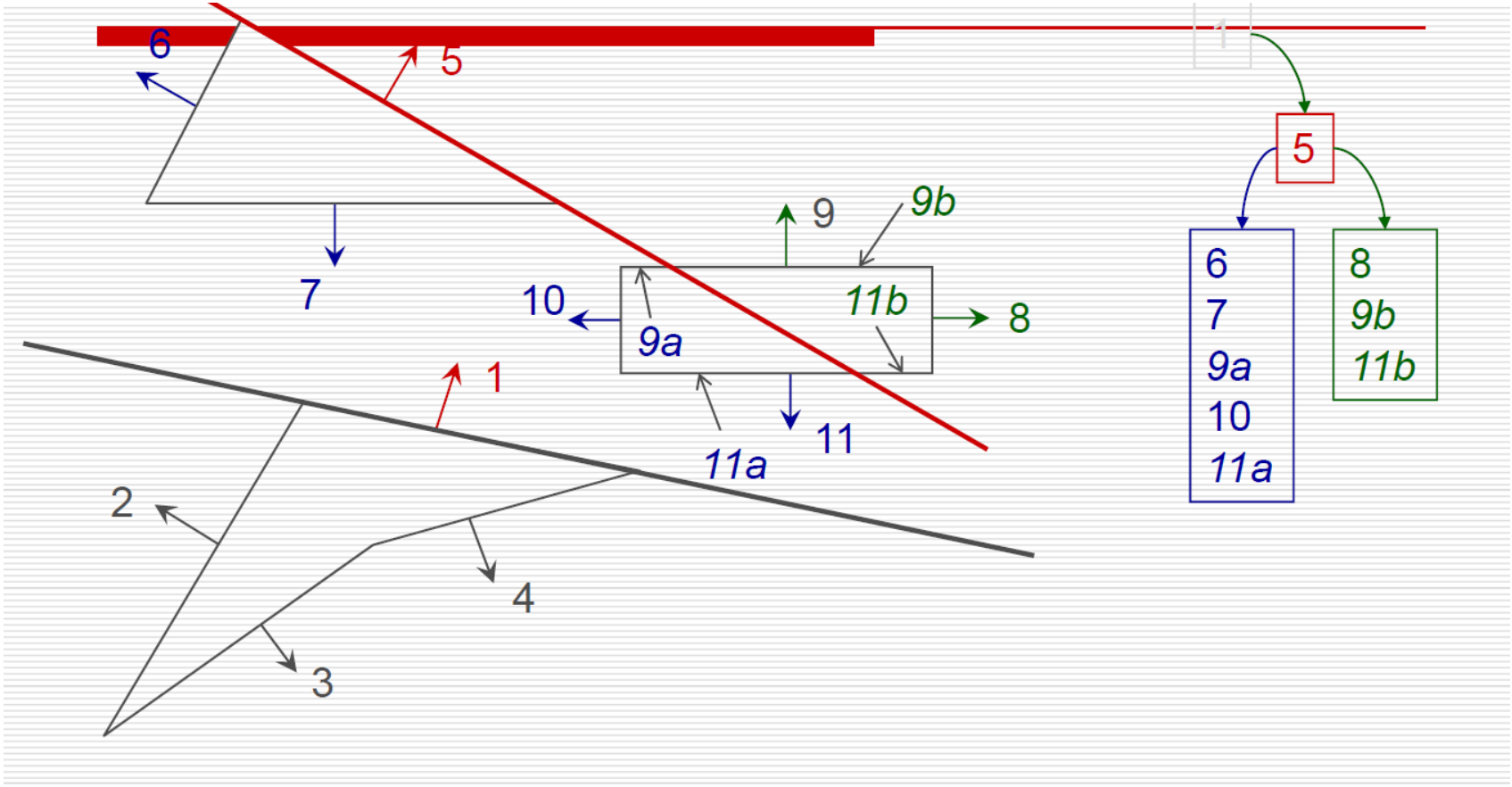


inside ones: backside (bList)
outside ones: front side (fList)

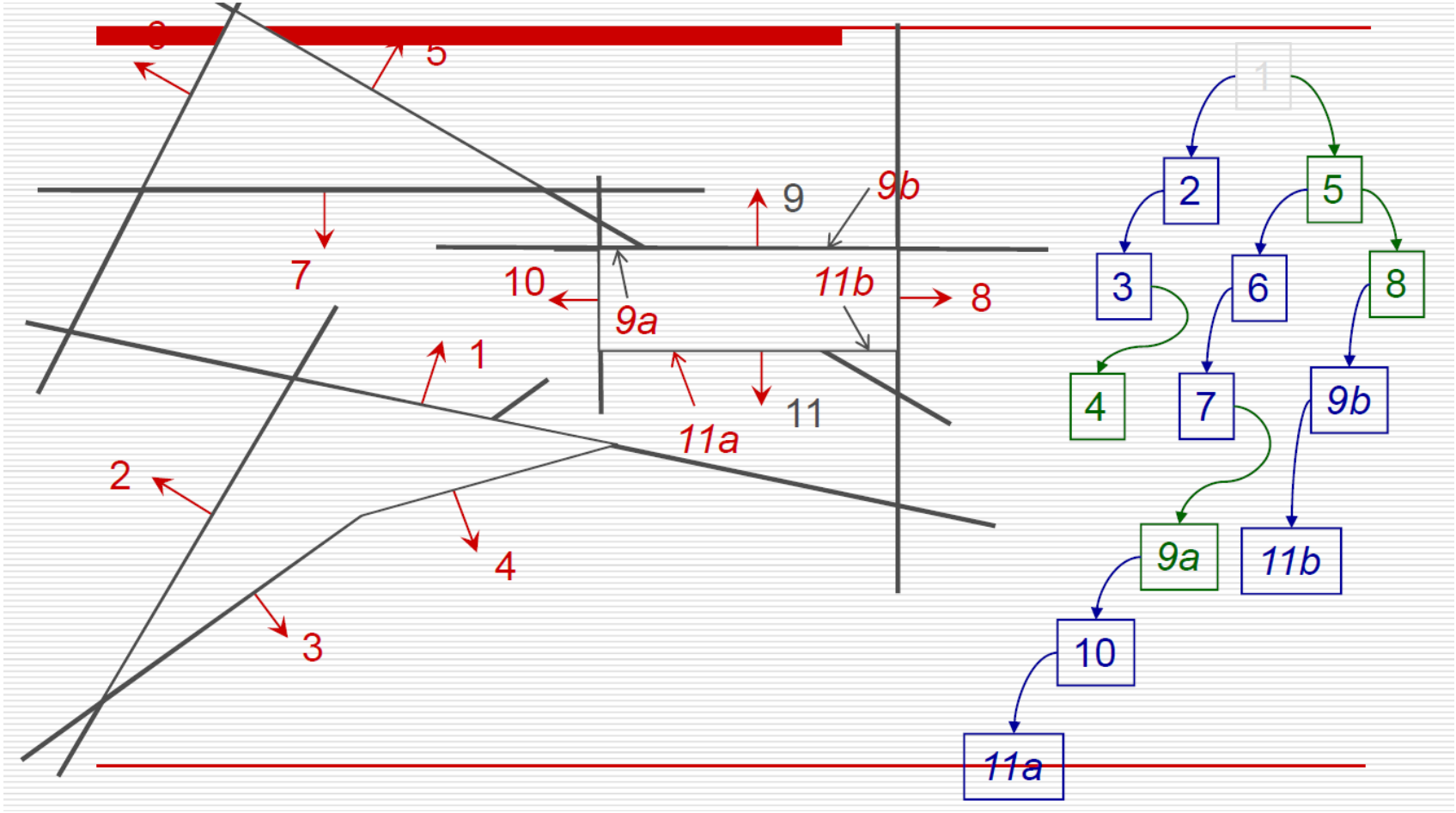
Example:BSP Tree Construction (cont.)



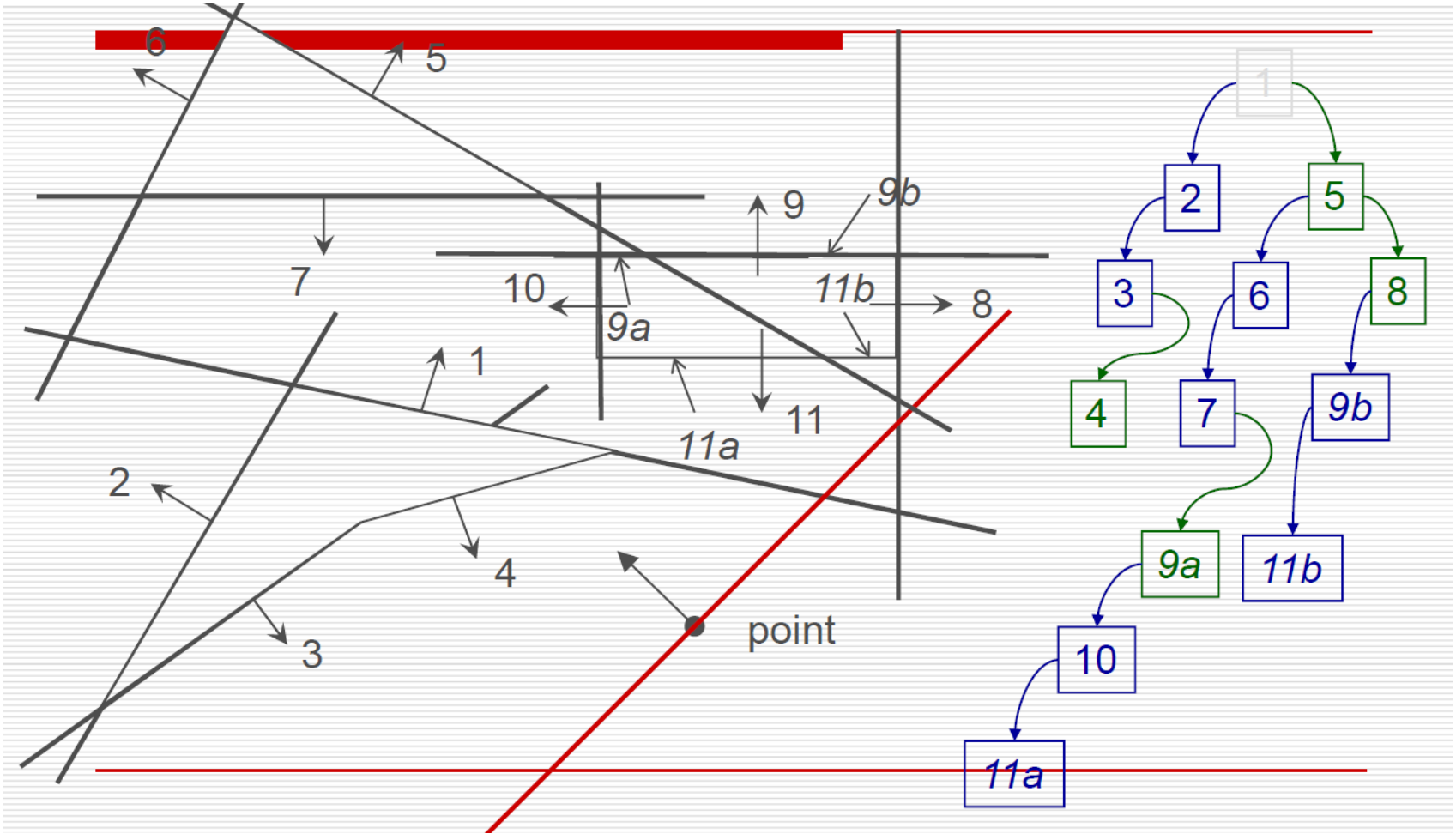
Example:BSP Tree Construction (cont.)



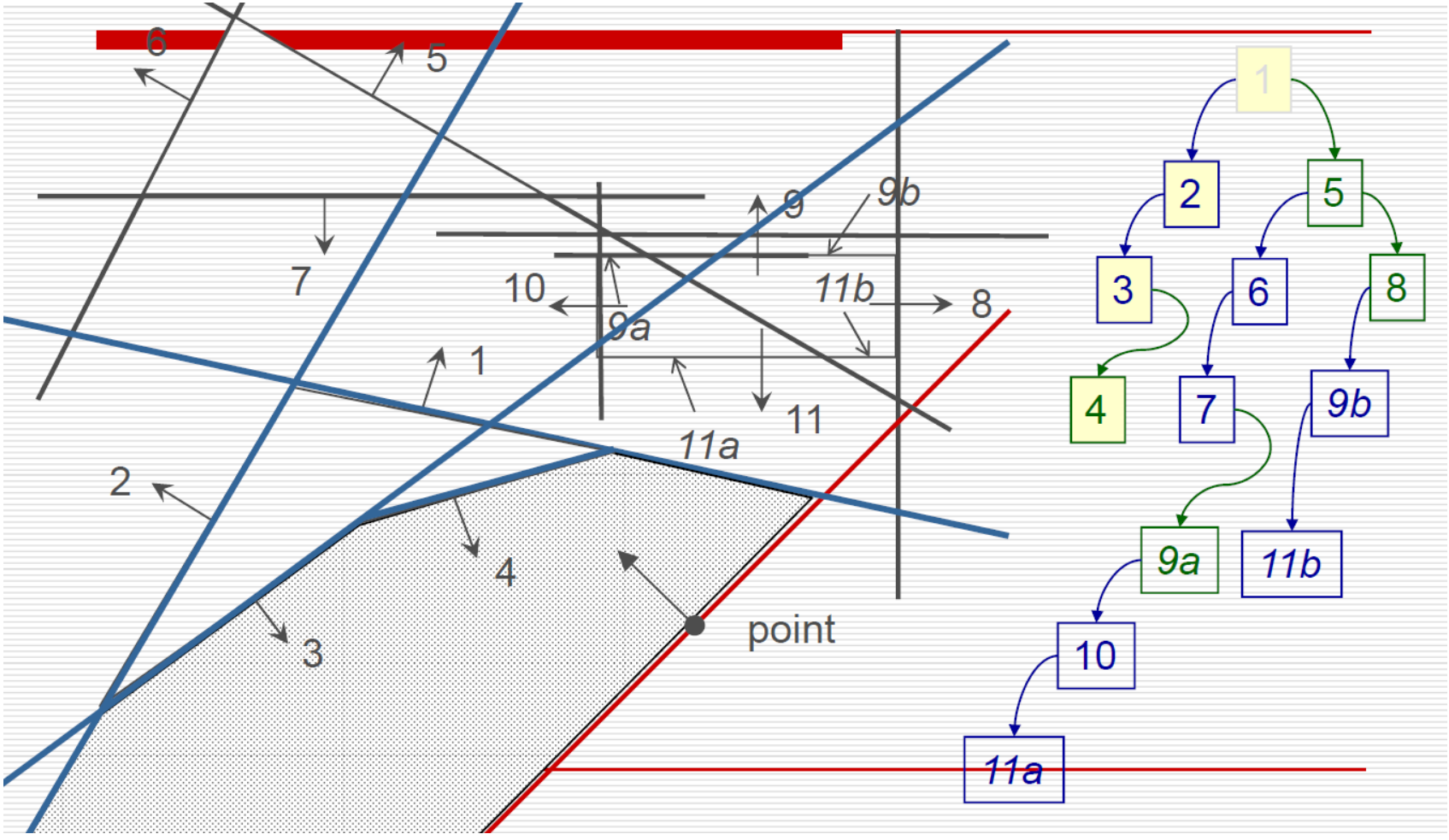
Example:BSP Tree Construction (cont.)



Example:BSP Tree Construction (cont.)



Example: BSP Tree Rendering



Psudeocode: Displaying a BSP Tree

```
displayBSPTree(bTree)
        if (empty(bTree)) return
        if (eye in front of root)
            displayBSPTree(bTree→bChild)
            displayPoly(root)
            displayBSPTree(bTree→fChild)
        else
            display(BSPTree(bTree→fChild)
            if (no back-face culling)
                displayPoly(root)
             display(BSPTree(bTree→bChild)
```

Performance Considerations

- Construction of the BSP tree is expensive.
- No splitting while rendering; everything is done during preprocessing.
- Straightforward display algorithm. Back-face culling woven into it.
- Strategy of selecting the root has a great impact.
- Select the polygon that splits least number of polygons.

Other Methods

- **Painter's Algorithm**: Reorder polygons back-to-front from the camera
  - Involves sorting the polygons for each view point
  - Sometimes, polygons need to be split as no unique ordering
- **Ray-Casting**: Examine each ray from the camera center
  - Expensive operation to *trace each ray from camera*
  - Can provide very high visual realism in addition to visibility

Depth-Sort Algorithms: Discussion

- Ensures back-to-front ordering for proper rendering.
- No aliasing effects introduced as objects are reordered/split.
- Reordering and splitting of polygons have to be done at run time.
- Redo the whole calculations if the view-point changes.
- Computationally expensive.

Z-buffering: Discussion

- Any shape with per pixel z can be handled correctly.
- Time is independent of number of primitives.
- Easy to implement; can do with a single scan-line Z-buffer.
- Z-buffer can be read back and saved.
- Needs extra memory, but memory is cheap.
- Can cause aliasing or **z-fighting** (*shimmering*).

VSD: Summary

- Sorting in the right order is key to all of them.
- If expensive lighting/shading is used, do not shade an image pixel more than once.
- For quick rendering, z-buffer algorithms are better.
- BSP Trees can be fast if environment is static.
- Ease of implementation and scope of hardware acceleration are also important.
- Z-buffering is popular due to memory being cheap.