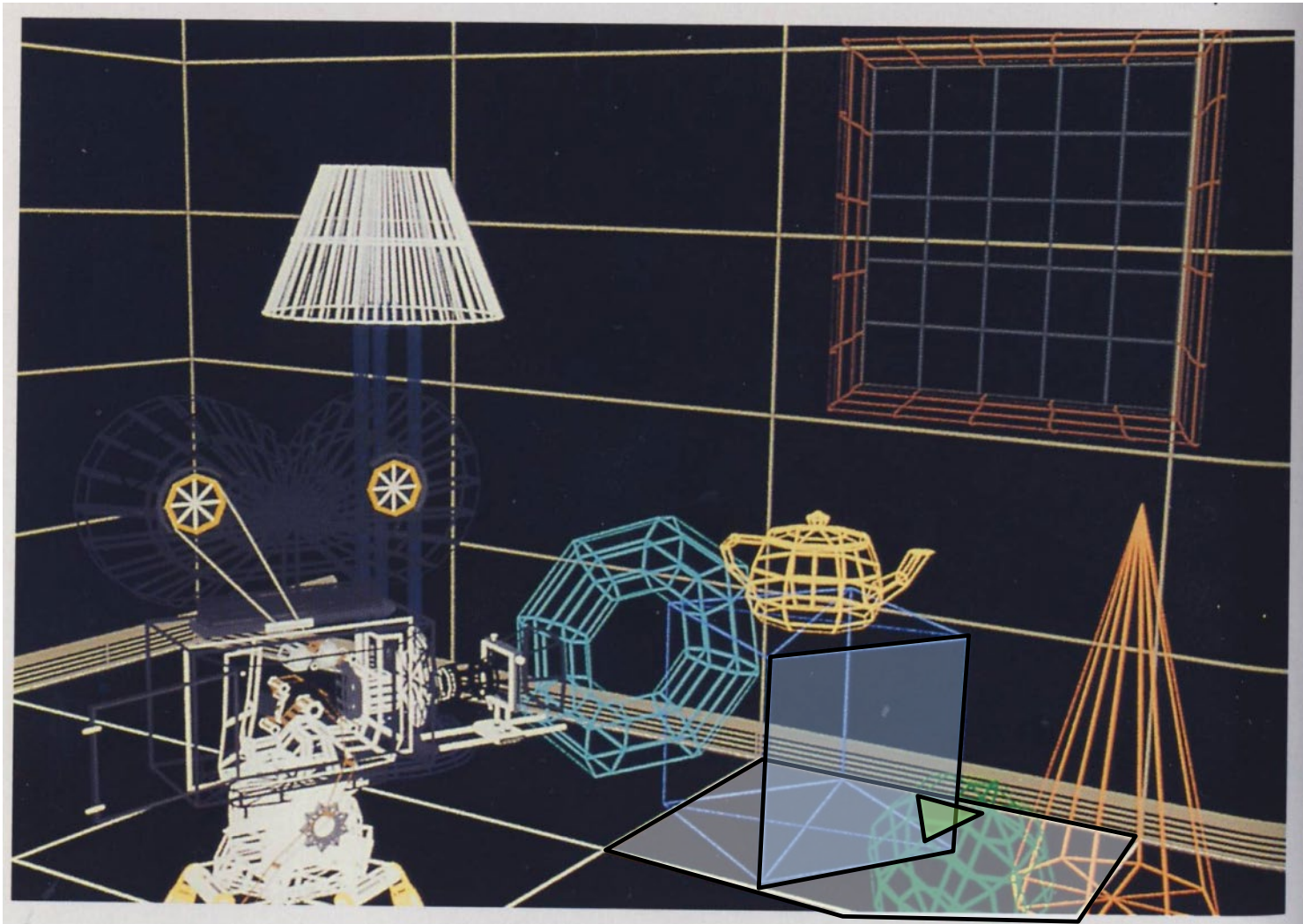
Computer Graphic

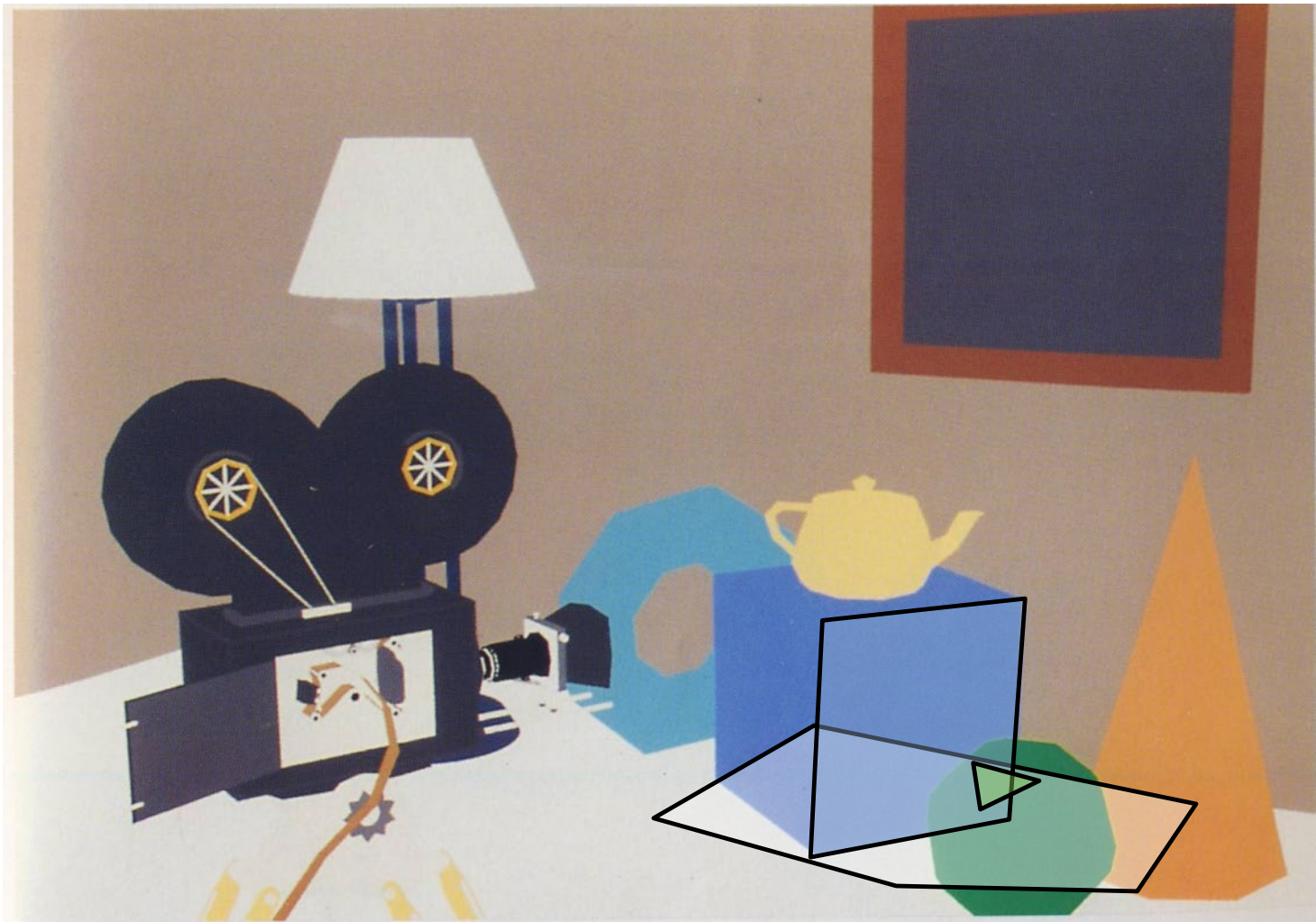
Hidden Surface Removal

Transformation/Viewing

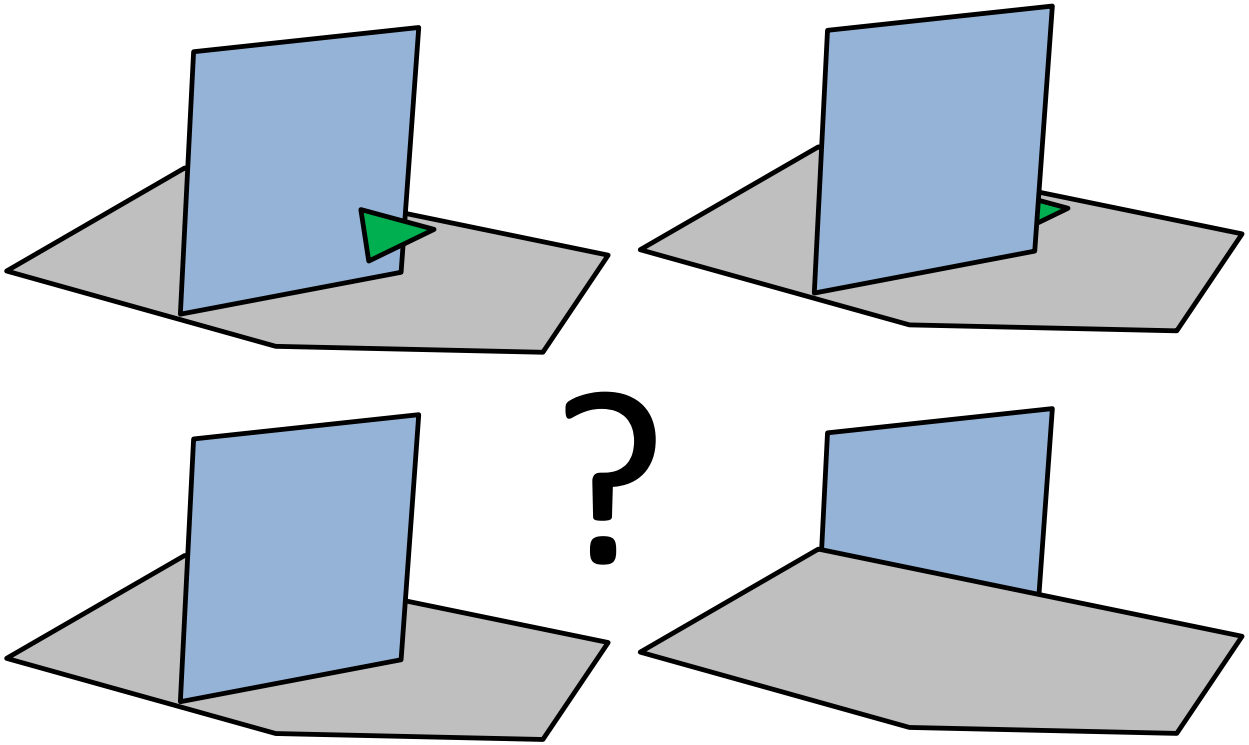


Scan Convert Algorithm

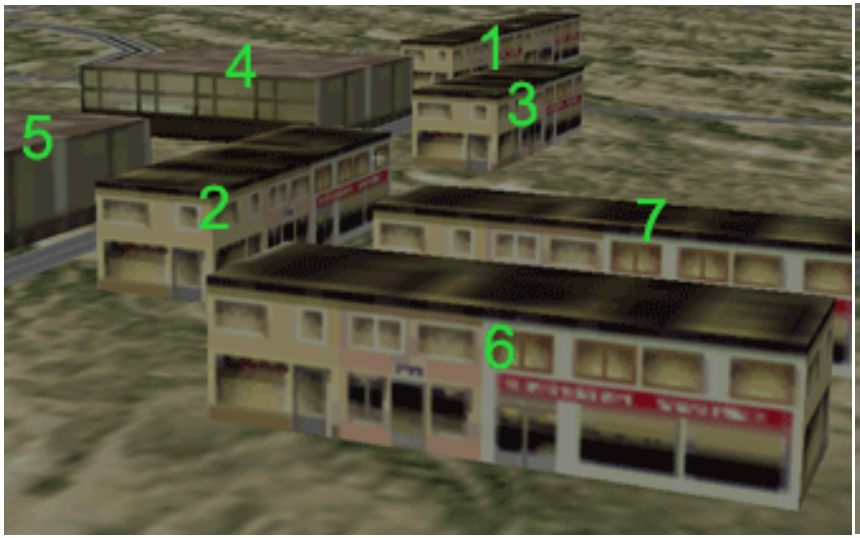
- But how do we know the order?

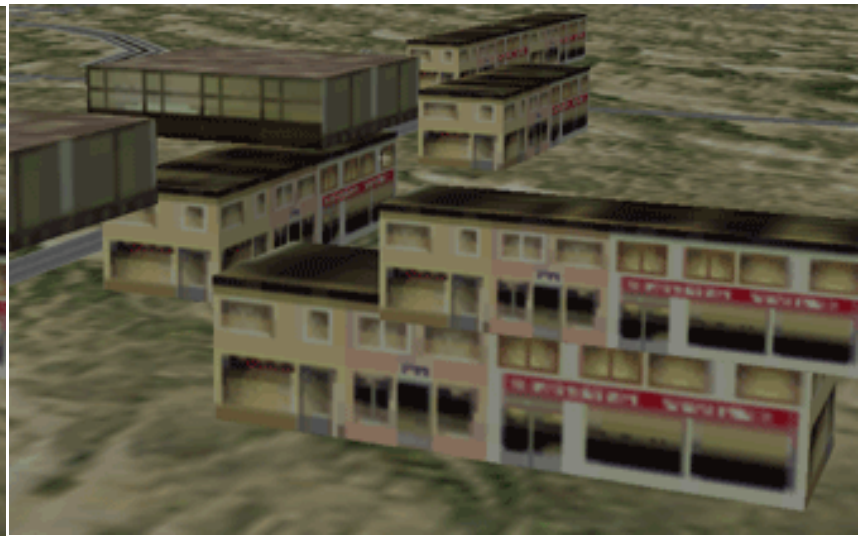


Drawing Order?



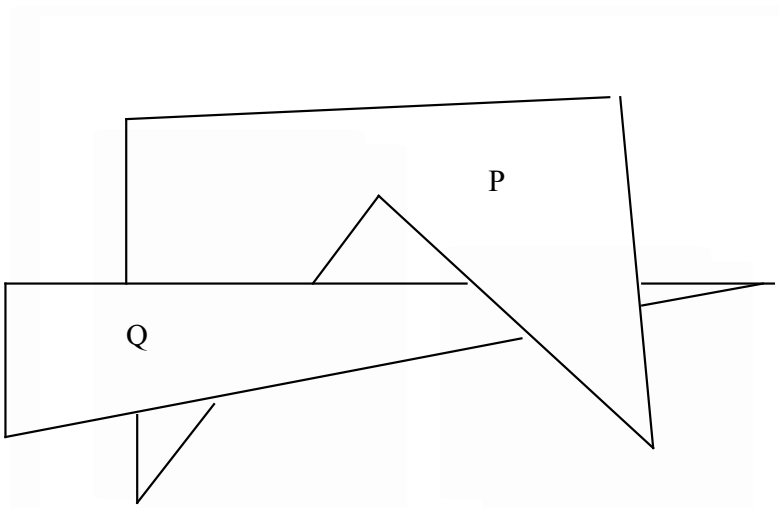
Wrong Drawing Order

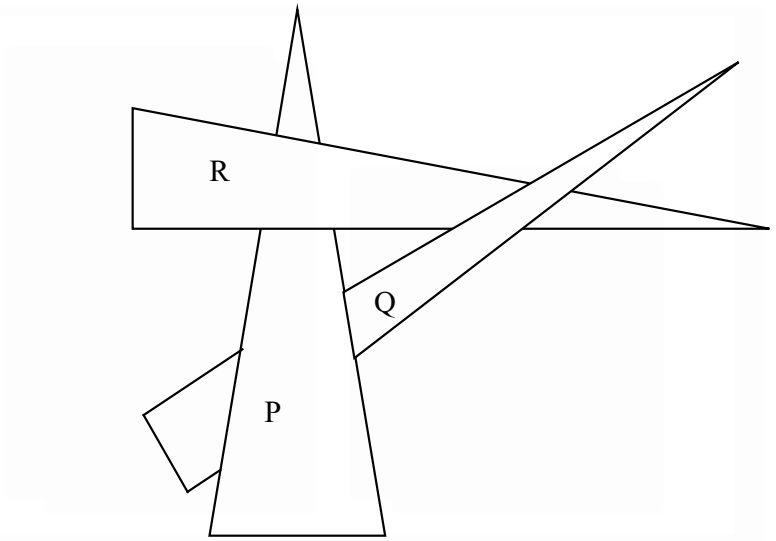




However

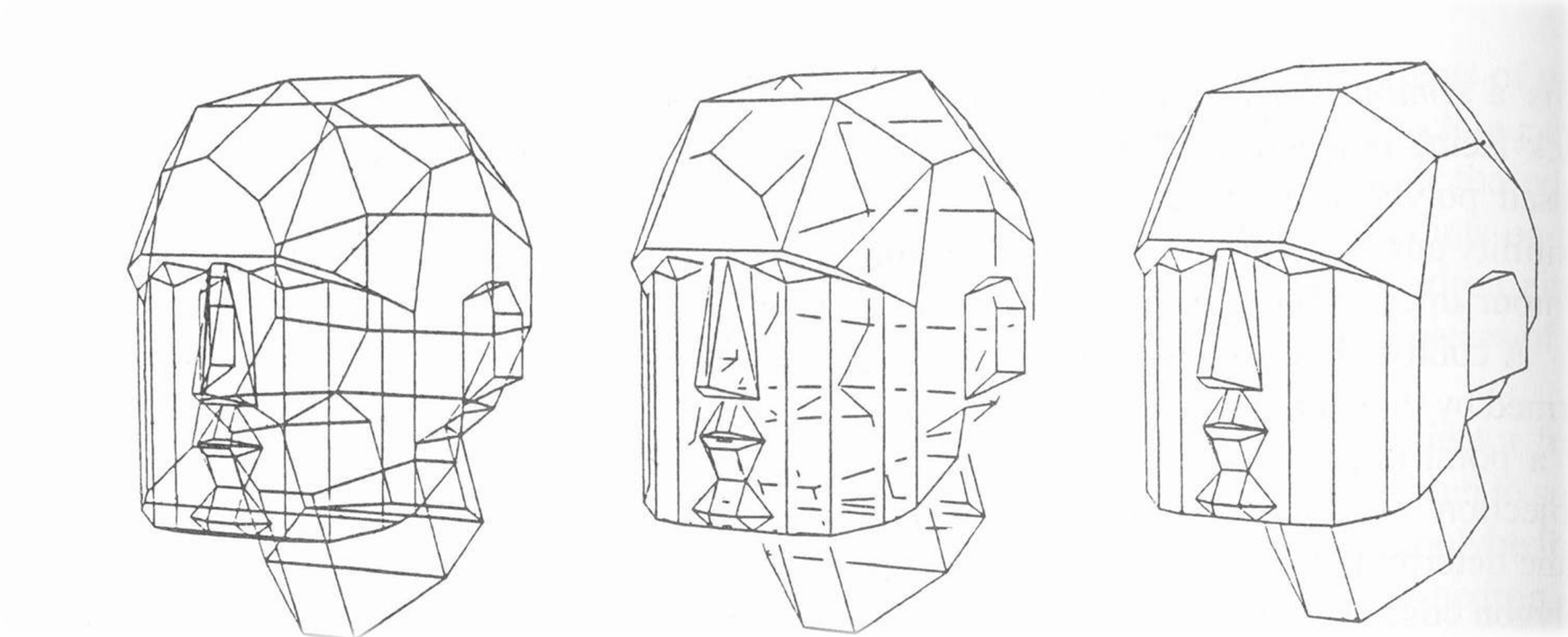
- Sometime, the drawing order does not exist





Hidden Surface Removal (HSR)

- Managing the polygons so that they are drawn in the right order



Binary Space Partitioning

Binary Space Partitioning

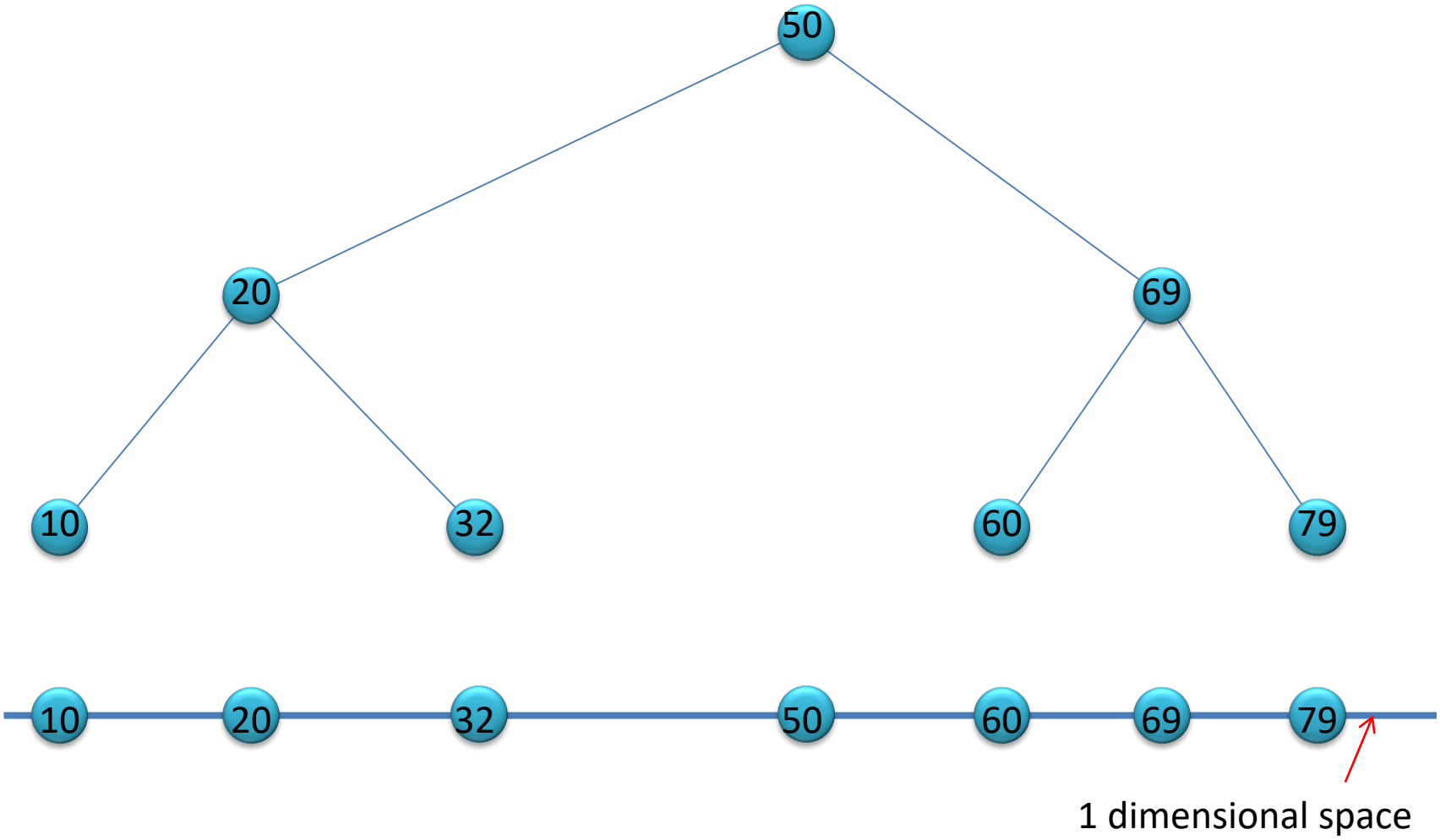
- An industrial standard for real time 3D games such as FPS
  - Doom, Counterstrike, Quake, etc…
- Input:
  - An environment modeled by polygons

- Preprocess the environment and produce a BSP tree
- Output
  - Base on the BSP tree, output a drawing order from the furthest away to the nearest polygon

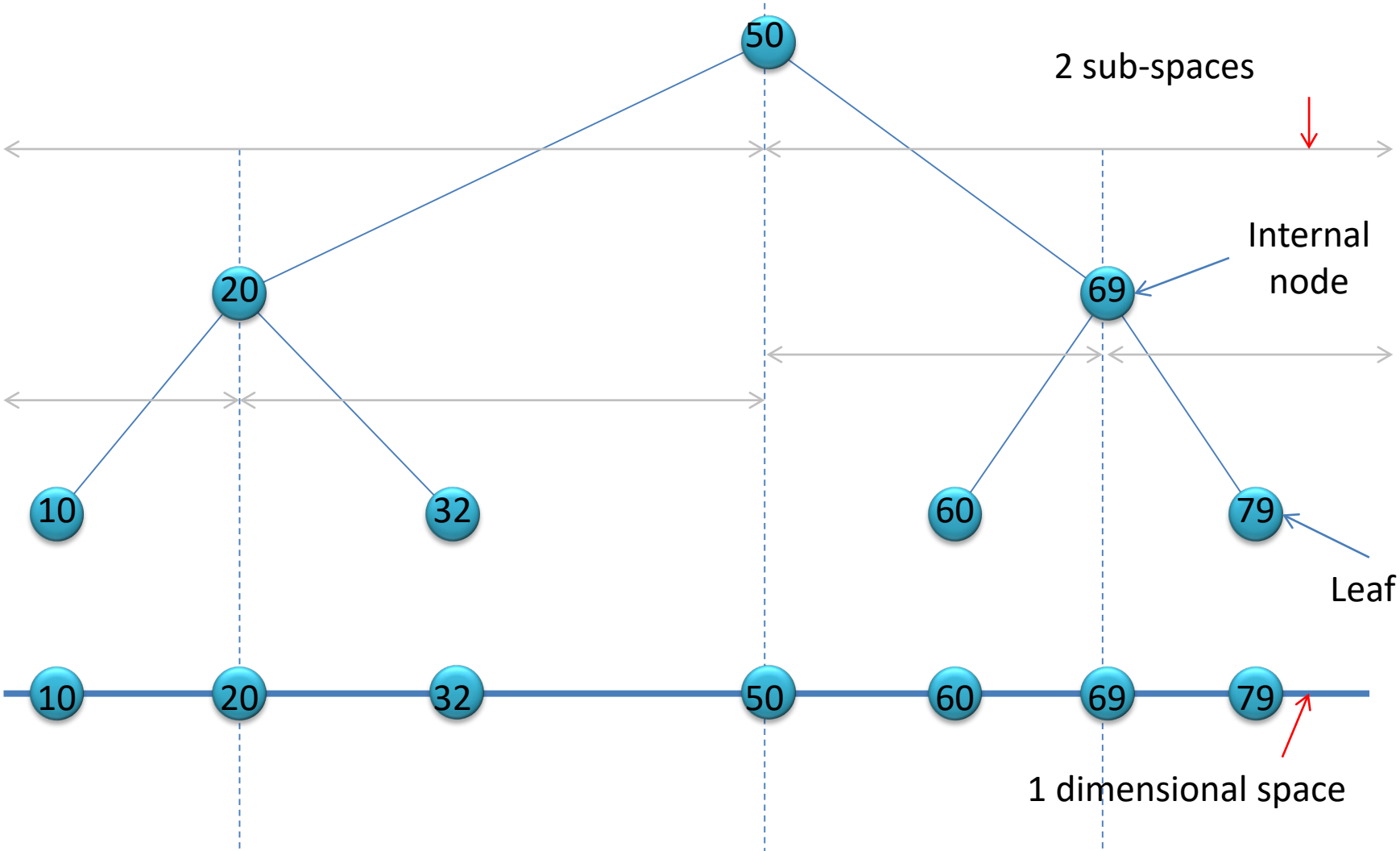
The First FPS

- There are free versions on web and iPad/iPhone
  - The iPad version is totally free on the Japanese iStore

Recap: Binary Search Tree



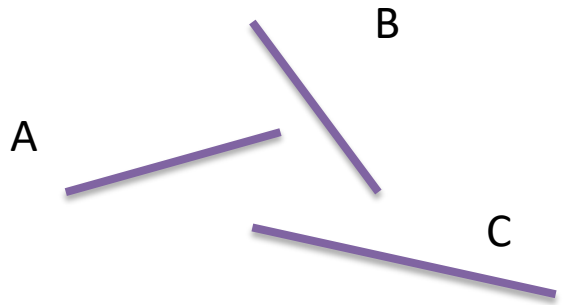
Recap: Binary Search Tree



Type 1b: Binary Space Partitioning

- Idea: Subdivide the entire space by a binary tree
  - Each internal node is a division of a partition/space
  - Each leaf is a part of the space with only **one** polygon

- Divide into two steps
  I. Preparation
  II. Rendering

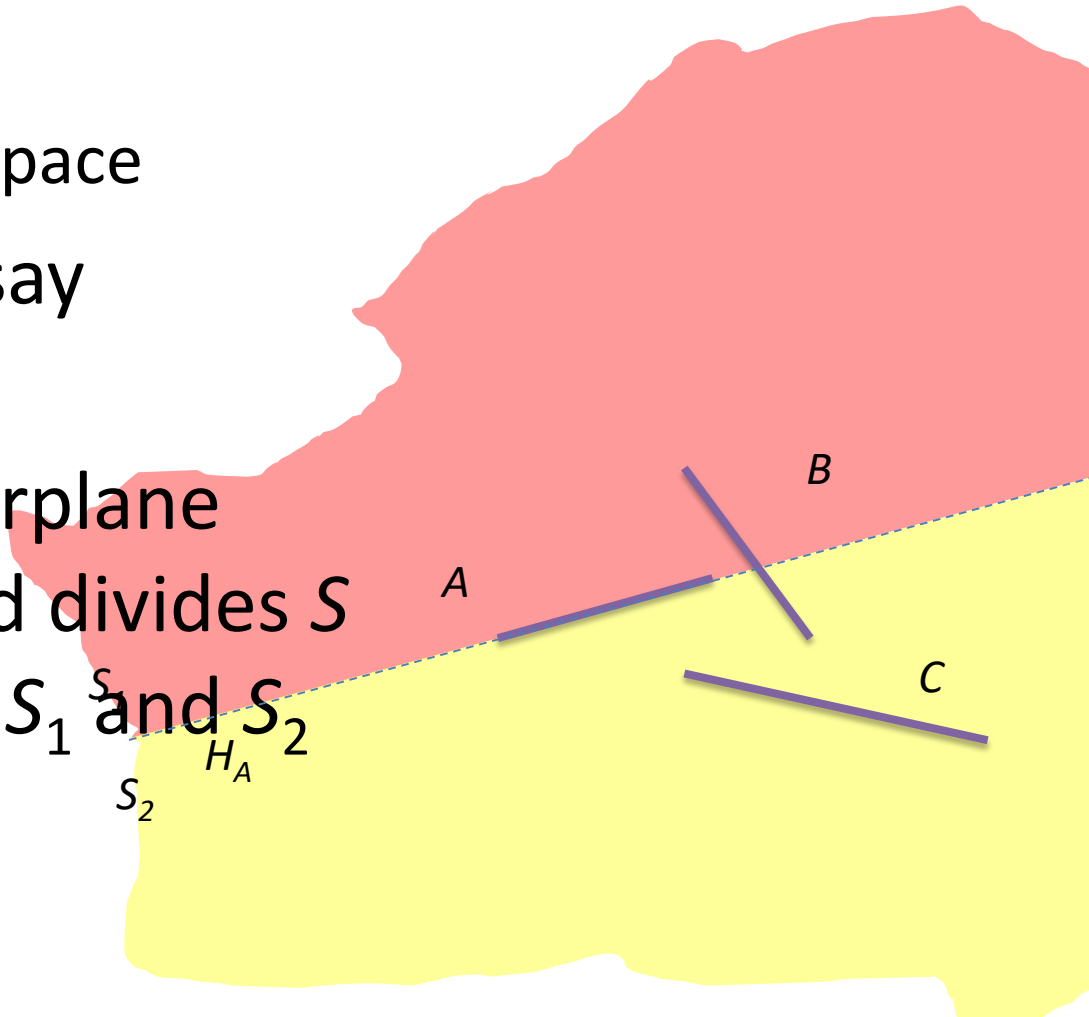


(Use a bird's eye view for illustration)

like looking at a square from above

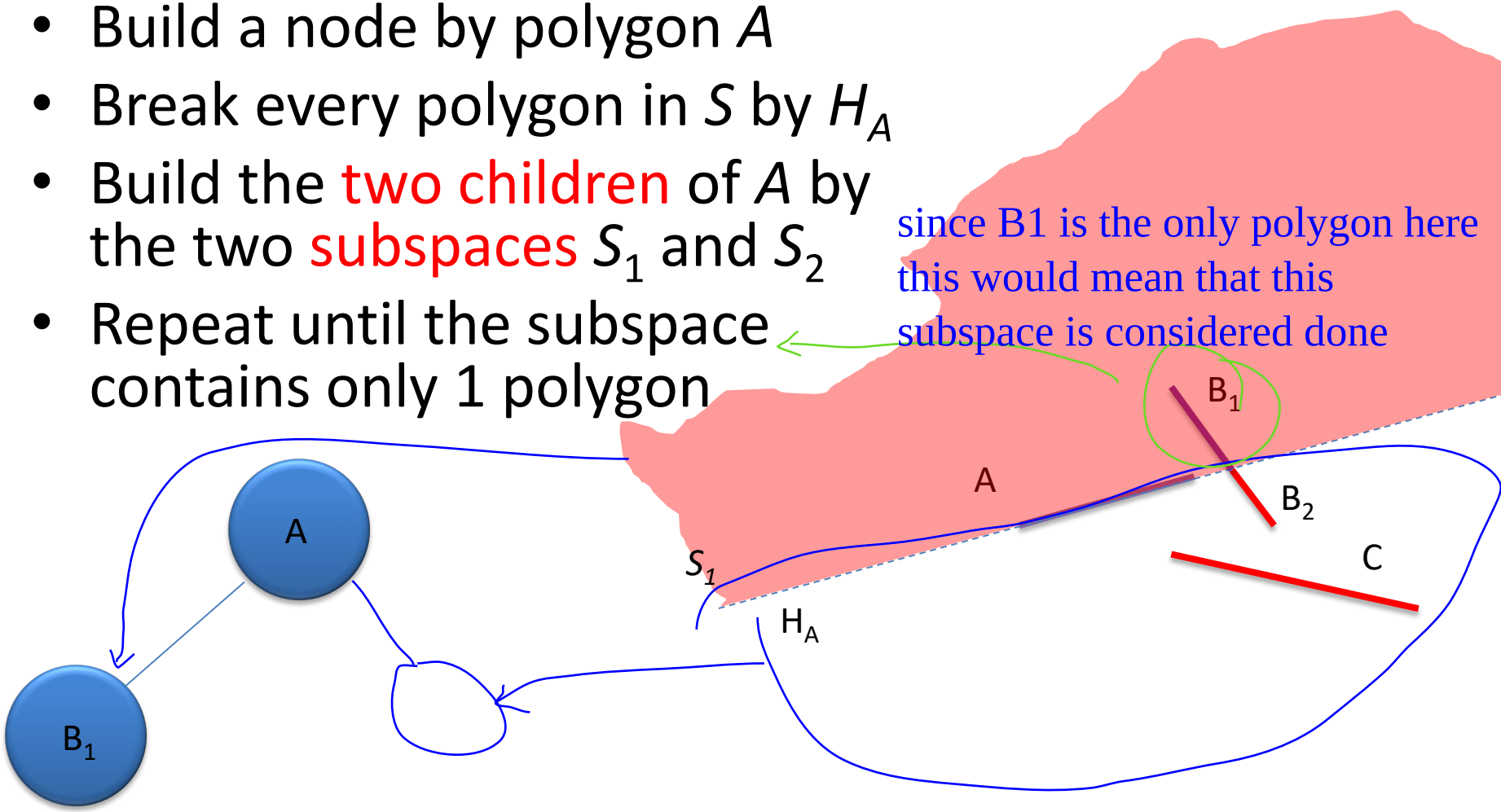
Type 1b: BSP-tree Preparation

- Initialization
  - Let S = the entire space
- Pick any polygon, say Polygon A
- Let H_A be the hyperplane that contains A and divides S into two subspace S_1 and S_2



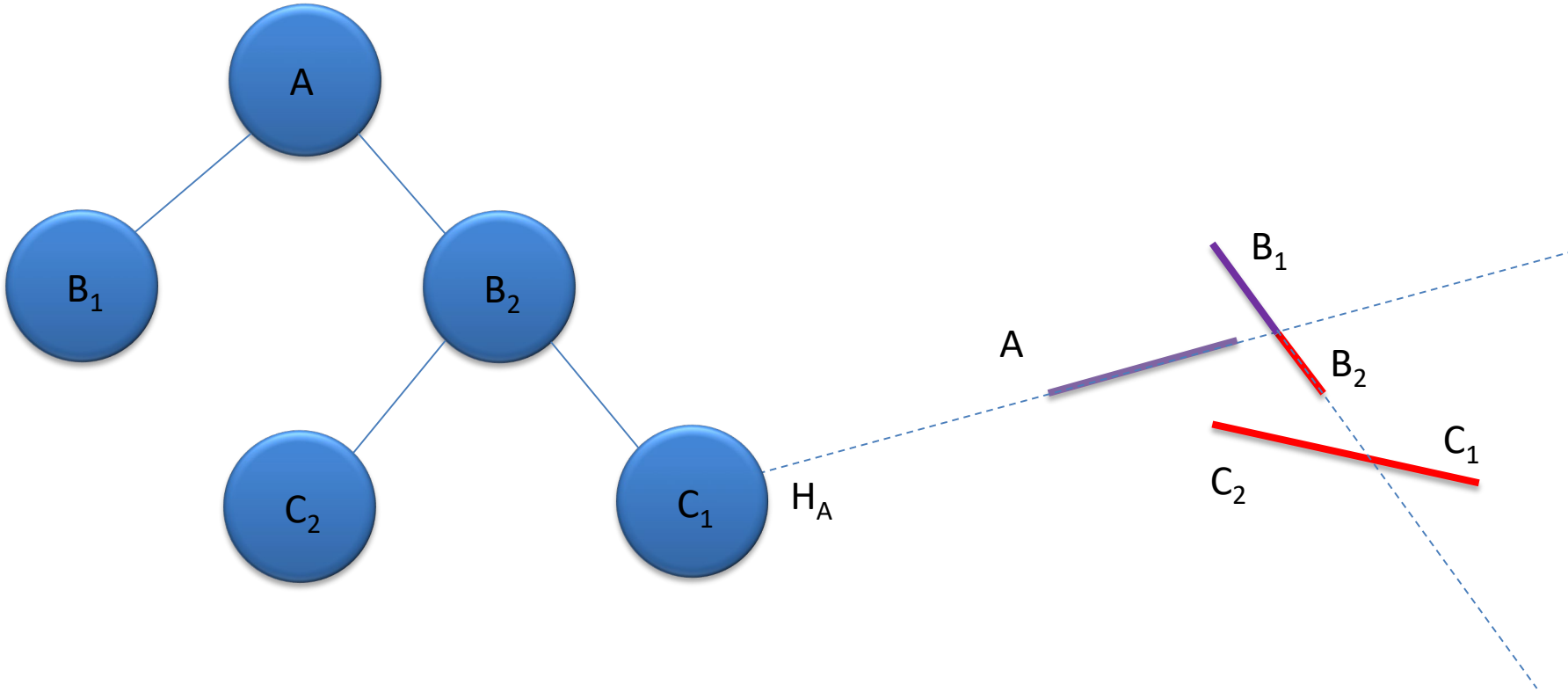
Type 1b: BSP-tree

- Build a node by polygon *A*
- Break every polygon in *S* by H_A
- Build the two children of *A* by the two subspaces S_1 and S_2
- Repeat until the subspace contains only 1 polygon



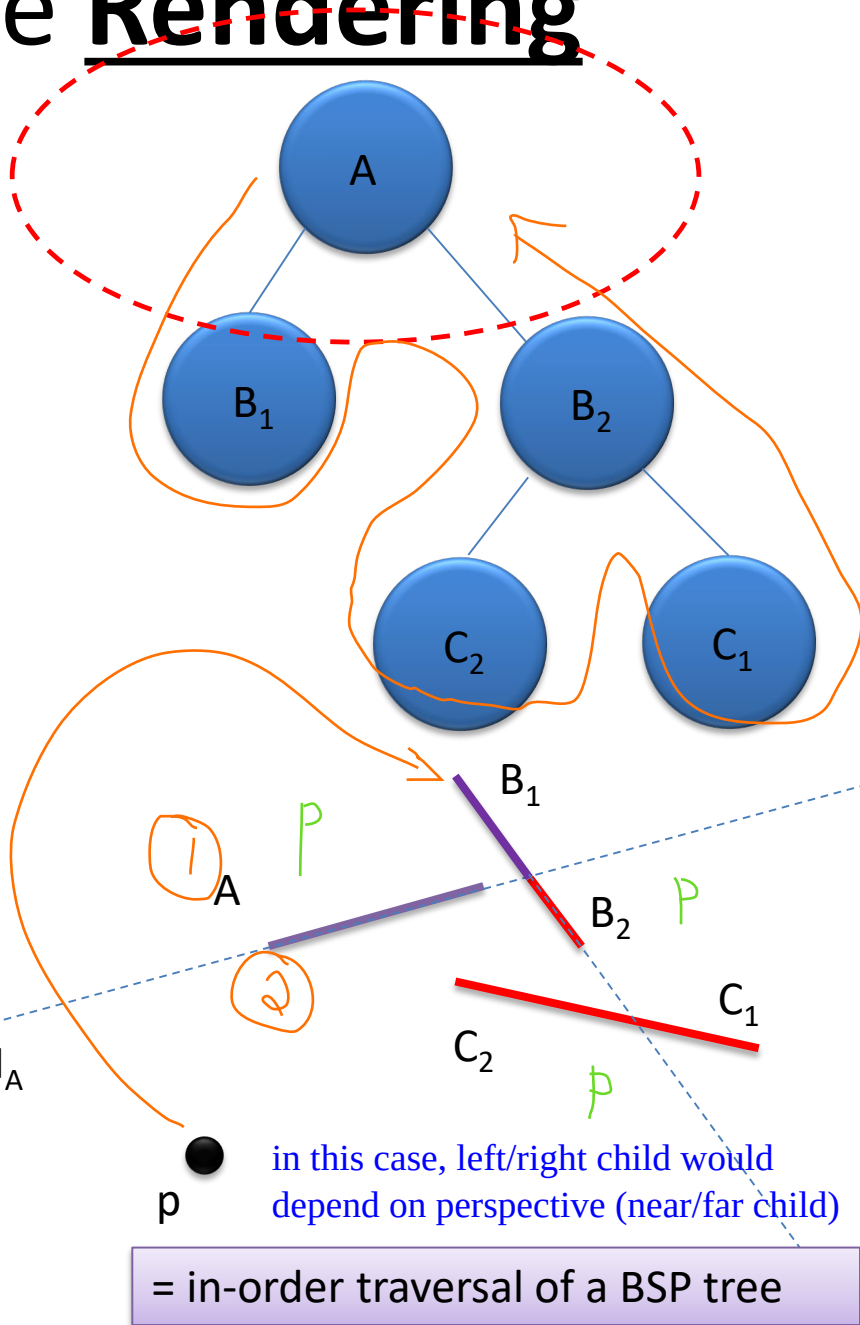
Type 1b: BSP-tree

- Prepare the BSP-tree for rendering:



Type 1b: BSP-tree **Rendering**

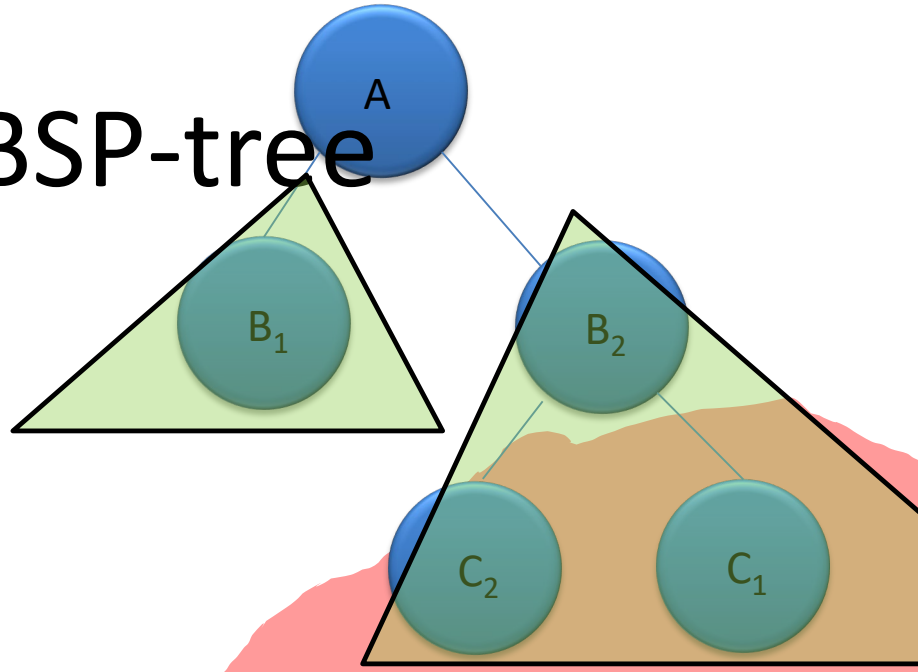
- Depending on the viewpoint *p*
- Start from the root
  - For each node there is one polygon and two sub-spaces in the two children
    1. Recursively draw the sub-tree behind the polygon from the view point *p*
    2. Draw the polygon of the node
    3. Recursively, draw the sub-tree in front of the polygon from the view point *p*

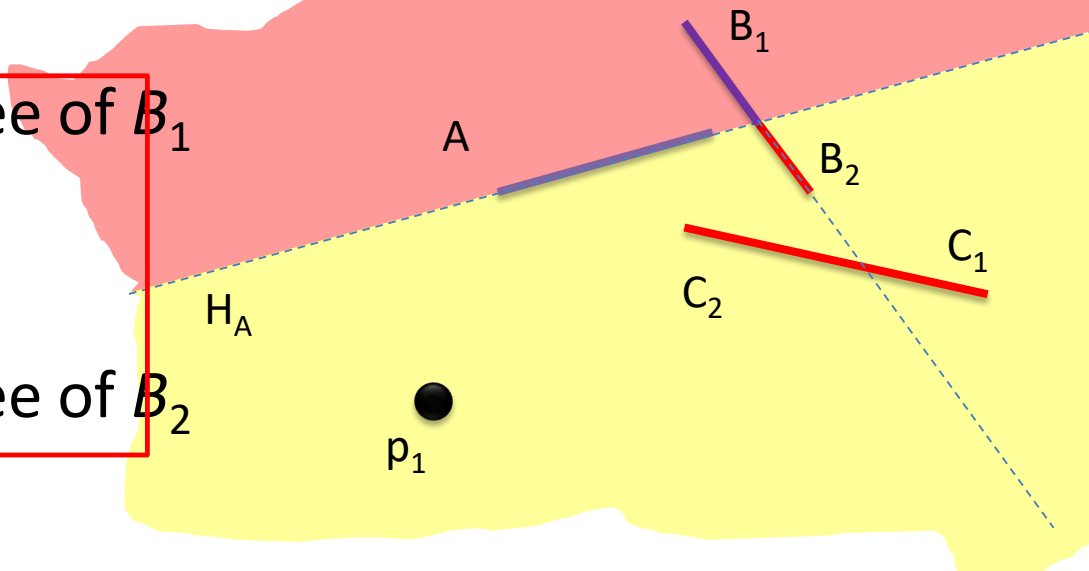


= in-order traversal of a BSP tree

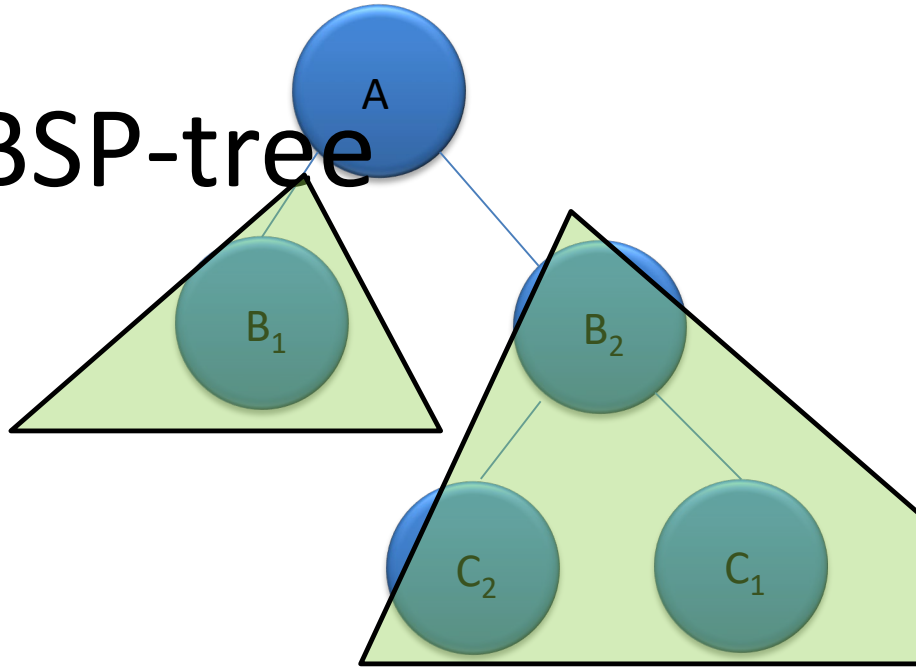
Type 1b: BSP-tree

- For the viewpoint p_1
  - Starting from A
    - The subtree of B_2 is in front of A
    - The subtree of B_1 is behind of A
  - Travserse the subtree of B_1 first
  - Then A
  - Travserse the subtree of B_2

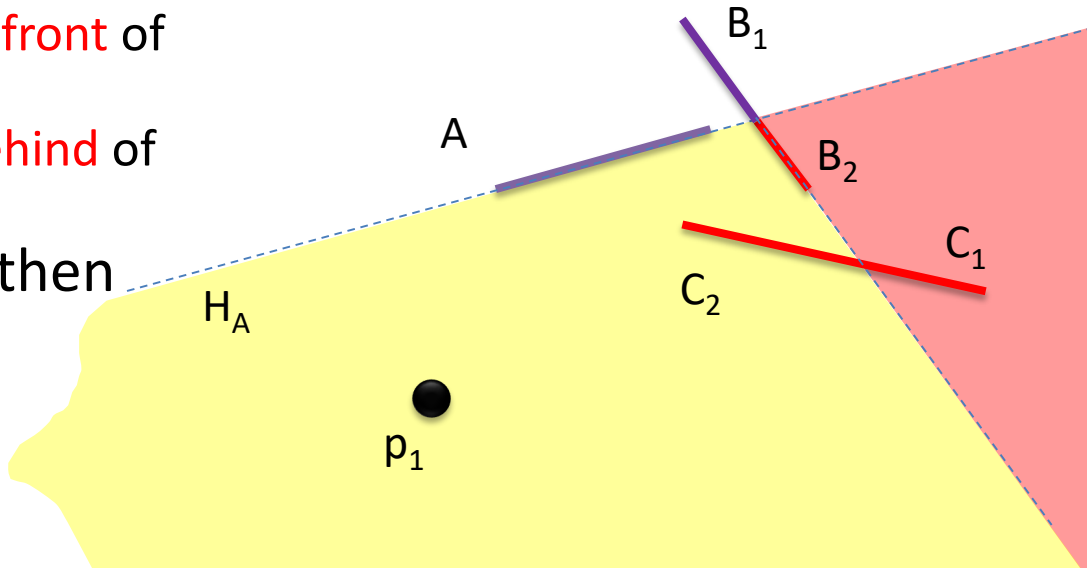




Type 1b: BSP-tree

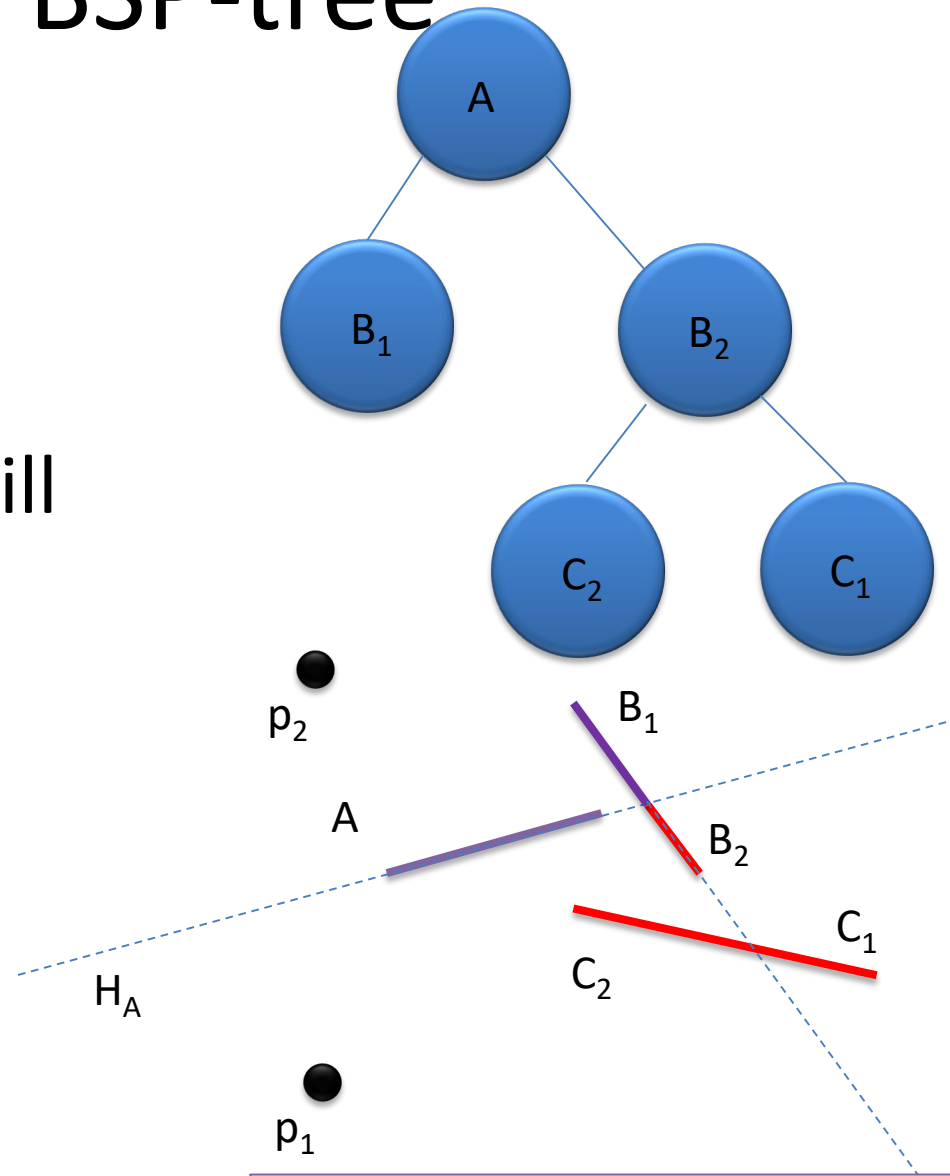


- Travsers the subtree of B_1 first
  - Output B_1
- Then output A
- Then Travserse the subtree of B_2
  - Starting from B_2
    - The subtree of C_2 is in front of B_2
    - The subtree of C_1 is behind of B_2
  - Therefore output C_1, then B_2, and finally C_2



Type 1b: BSP-tree

- For example, for the following viewpoints, their drawing order will be
  - p_1 : B_1, A, C_1, B_2 , C_2
  - p_2: C_1, B_2, C_2, A, B_1



Detailed exercises in tutorials

Type 1b: BSP-tree

- **Advantage**:
  - Once the tree is computed, the tree can handle all viewpoints without reconstructing the tree, i.e. efficient
  - Handle transparency
  - Indeed, it's a standard format (**`.BSP`** files) to store the environment for many games
    - E.g. Quake, Half-life, Call of Duty, etc
- **Disadvantages**
  - Cannot handle moving/changing environments
  - Preprocessing time for tree construction is long