



## Course Plan



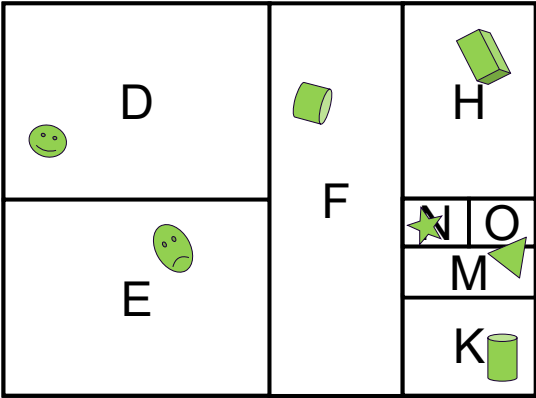
- lec. 1: **Introduction** ●
- lec. 2: **Mathematics** for 3D Games ●●●●●
- lec. 3: **Scene Graph** ●
- lec. 4: **Game 3D Physics** ●●●● + ●●●●●
- lec. 5: **Game Particle Systems** ●
- lec. 6: **Game 3D Models** ●●
- lec. 7: **Game Textures** ●●
- lec. 8: **Game 3D Animations** ●●●
- lec. 9: **Game 3D Audio** ●
- lec. 10: **Networking** for 3D Games ●
- lec. 11: **Artificial Intelligence** for 3D Games ●
- lec. 12: **Game 3D Rendering Techniques** ●●



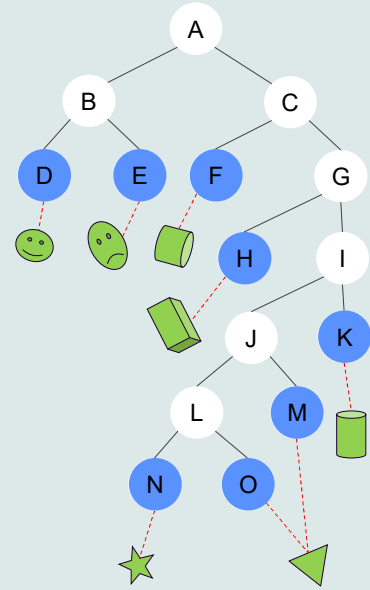
Let's continue the discussion on the **spatial indexing structures** for **collision detection**

63

## kD-trees



the scene



64

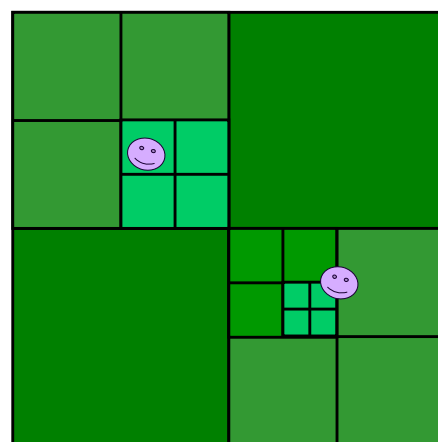
## kD-trees



- Hierarchical structure: a tree
  - each node: a subpart of the 3D space
  - root: all the world
  - child nodes: partitions of the father
  - objects linked to leaves
- kD-tree:
  - binary tree
  - each node: split over one dimension (in 3D: X,Y,Z)
  - variant:
    - each node optimizes (and stores) which dimension, or
    - always same order: e.g. X then Y then Z
  - variant:
    - each node optimizes the split point, or
    - always in the middle

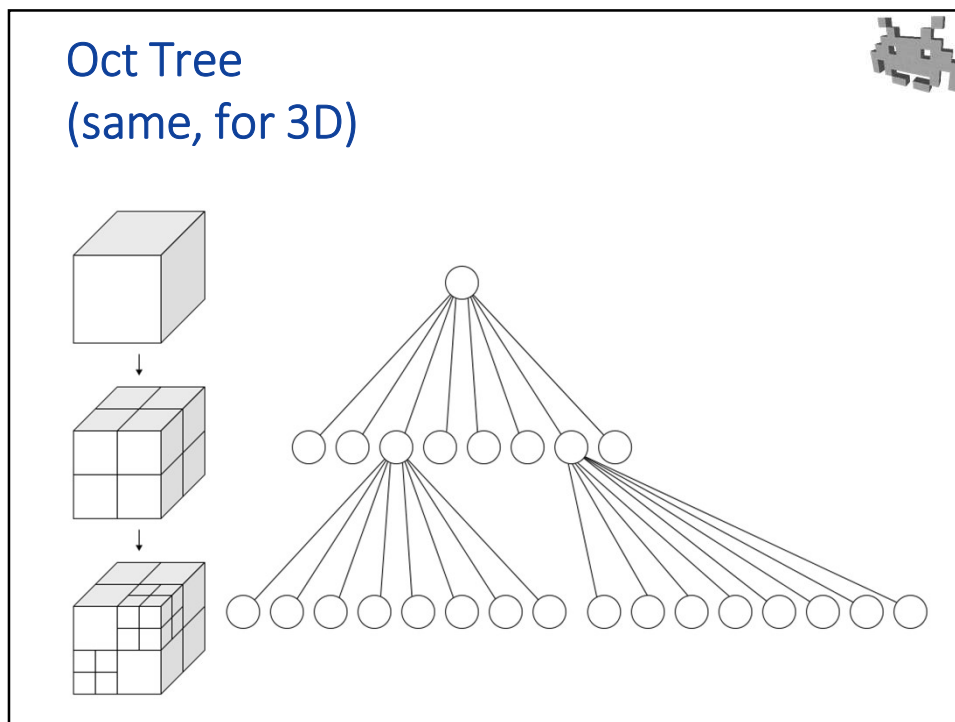
65

## Quad-Tree (in 2D)



the (2D) world

66

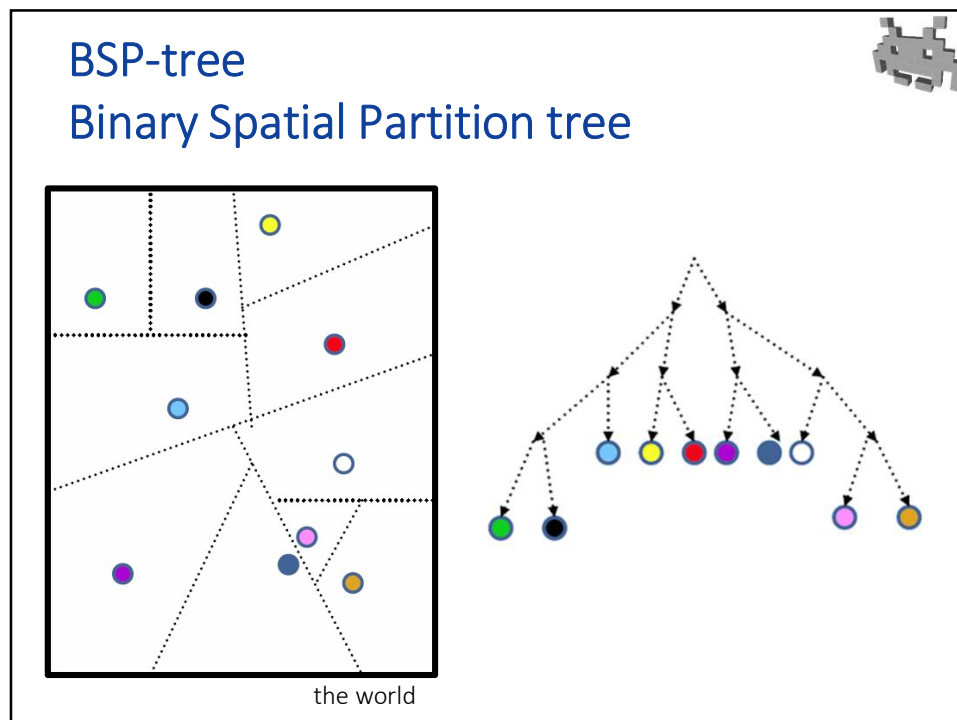


67

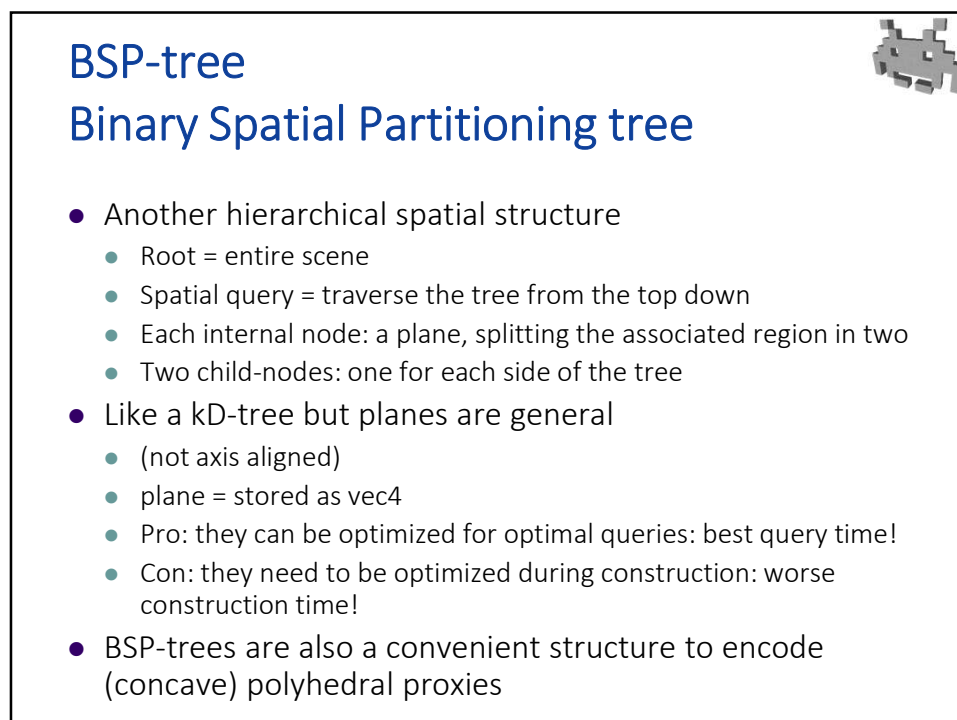
### Quad trees (in 2D) Oct trees (in 3D)

- Similar to kD-trees, but:
  - tree: branching factor: 4 (in 2D) or 8 (in 3D)
  - each node: splits into all dimensions at once, (in the middle)
- Construction (just as kD-trees):
  - continue splitting until a end nodes has few enough objects (or limit level reached)

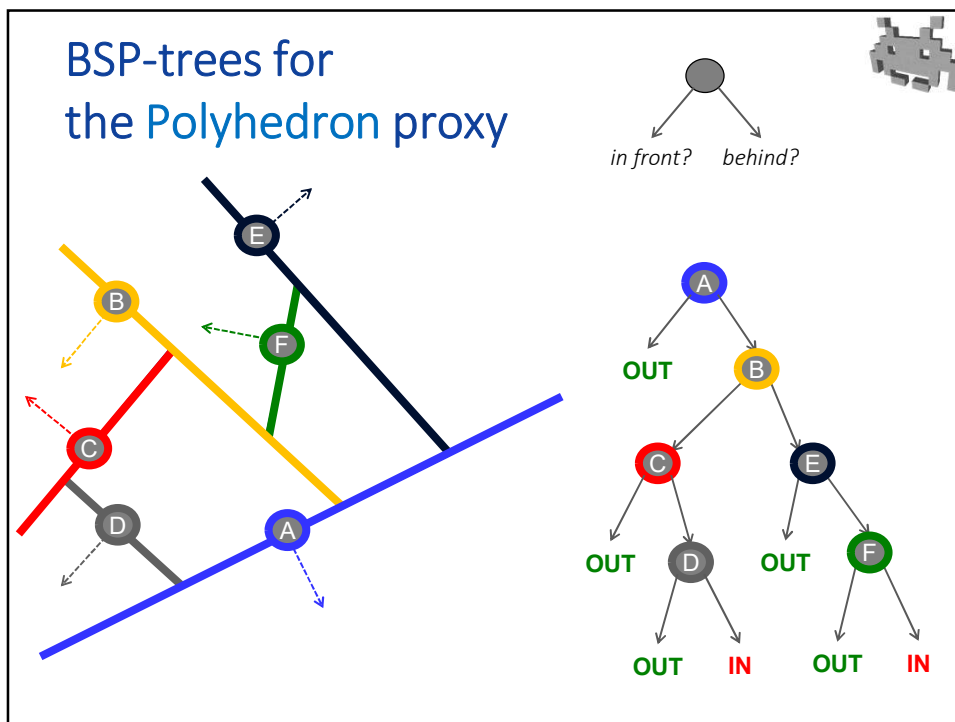
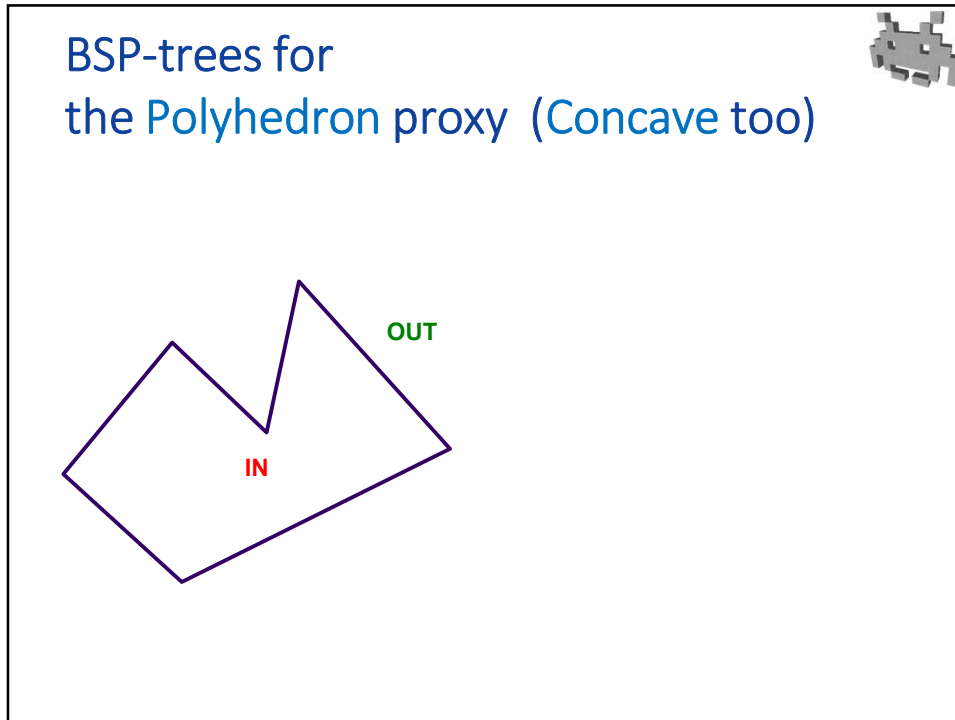
68



69



70



## BSP-tree

### Binary Spatial Partitioning tree

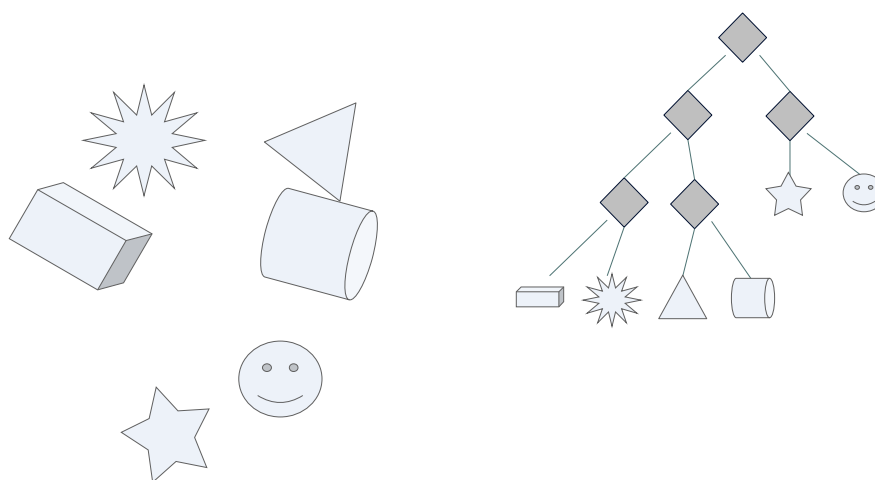


- Another variant
  - a binary tree (like the kD-tree)
    - root = all scene (like kD-tree)
  - but, each node is split by an *arbitrary* plane
    - (or a *line*, in 2D)
    - plane is stored at node, as  $(n_x, n_y, n_z, k)$
  - planes can be optimized for a given scene
    - e.g. to go for a 50%-50% object split at each node
    - e.g. to exactly *one* object at leaves
      - (assuming it is always possible to split any two apart – reasonable assumption)
- Another use: to test (Generic) Polyhedron proxy:
  - note: with planes defined in its **object space**
  - each leaf: inside or outside
    - (no need to store them: left-child = in, right-child = out)
  - tree precomputed for a given Collision Object

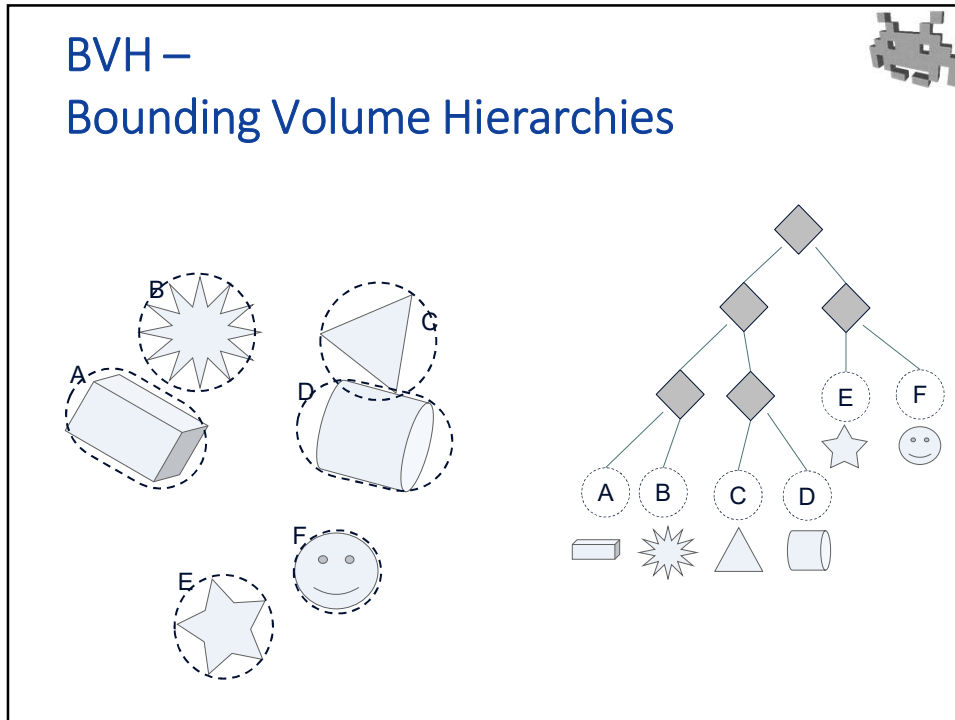
73

## BVH

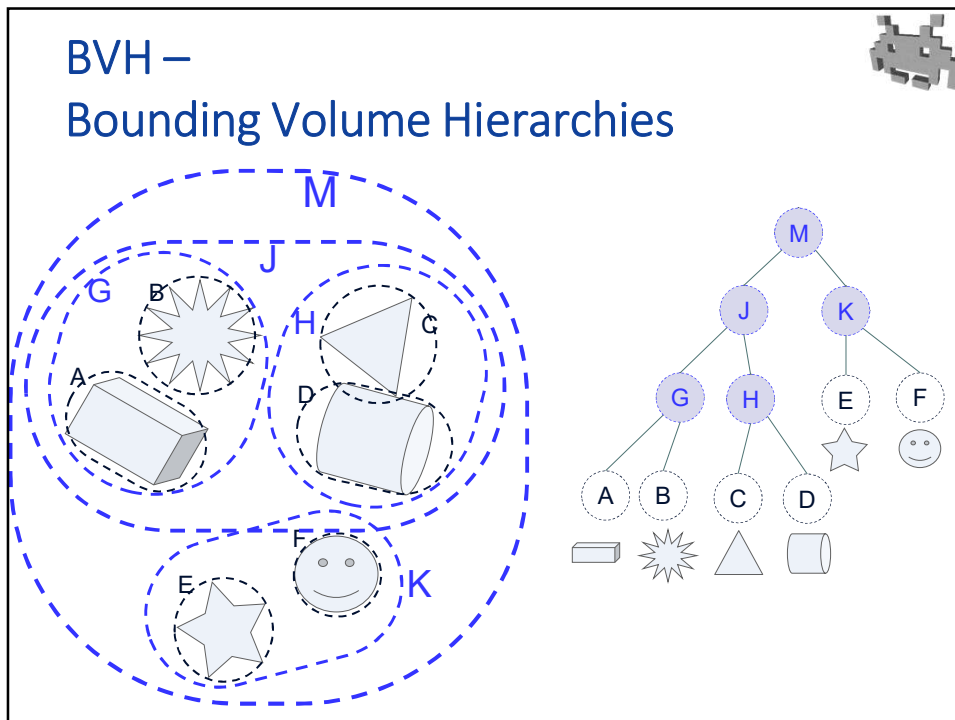
### Bounding Volume Hierarchy



74



75



76

## BVH

### Bounding Volume Hierarchy



- Idea: use the scene hierarchy given by the scene graph
  - (instead of a spatial derived one)
- associate a Bounding Volumes to each node
  - rule: a BV of a node bounds all objects in the subtree
- construction / update: quick! 😊
  - bottom-up: recursive (how?)
- using it:
  - top-down: visit (how?)
  - *note: not* a single root to leaf path
    - may need to follow *multiple* children of a node (in a BSP-tree: only one)

77

## Spatial indexing structures

### Recap



- Regular Grid
  - 😊 the most parallelizable (to update / construct / use)
  - 😊 constant time access (best!)
  - ☹ quadratic / cubic RAM space (2D, 3D) – unless hashing
- kD-tree, Oct-tree, Quad-tree
  - 😊 compact
  - 😊 simple
- BSP-tree
  - 😊 optimized splits! → best performance when accessed
  - ☹ optimized splits! → more complex construction / update
  - ideal for **static parts of the scene?**
  - (also, used for generic Polyhedral Collider)
- alternative: BVH
  - 😊 simplest construction
  - ☹ non necessarily super efficient to access
    - may need to traverse multiple children
    - if uses same hierarchy of the scene-graph: not always the best
  - ideal for **dynamic parts of the scene?**

78



## Physics Engine: an implementation problem



- Task: **Dynamics**:
  - (forces, speed and position updates...)
  - simple structures, fixed workflow
  - highly parallelizable: **GPU** possible
- Task: **Constraints Enforcement**:
  - still moderately simple structures, fixed workflow
  - problem: collision constraints not know a-priori
  - still highly parallelizable: hopefully, **GPU** possible
- Task: **Collisions Detection**:
  - non-trivial data structures, hierarchies, recursive algorithms...
  - hugely variable workflow
    - (e.g.: quick on no-collision, more work to do when the rare collisions occur)
  - difficult to parallelize: **CPU**
  - but outcome affect the other two tasks (e.g. creates constraints):
    - ==> **CPU-GPU** communication, and ==> **GPU** structures updates (problematic on many architectures)

79

## Physics: that's all folks. To gather more info...



- Erwin Coumans  
**SIGGRAPH 2015 course**  
<http://bulletphysics.org/wordpress/?p=432>
- Müller-Fischer et al.  
***Real-time physics***  
(Siggraph course notes, 2008)  
<http://www.matthiasmueller.info/realtimephysics/>

80