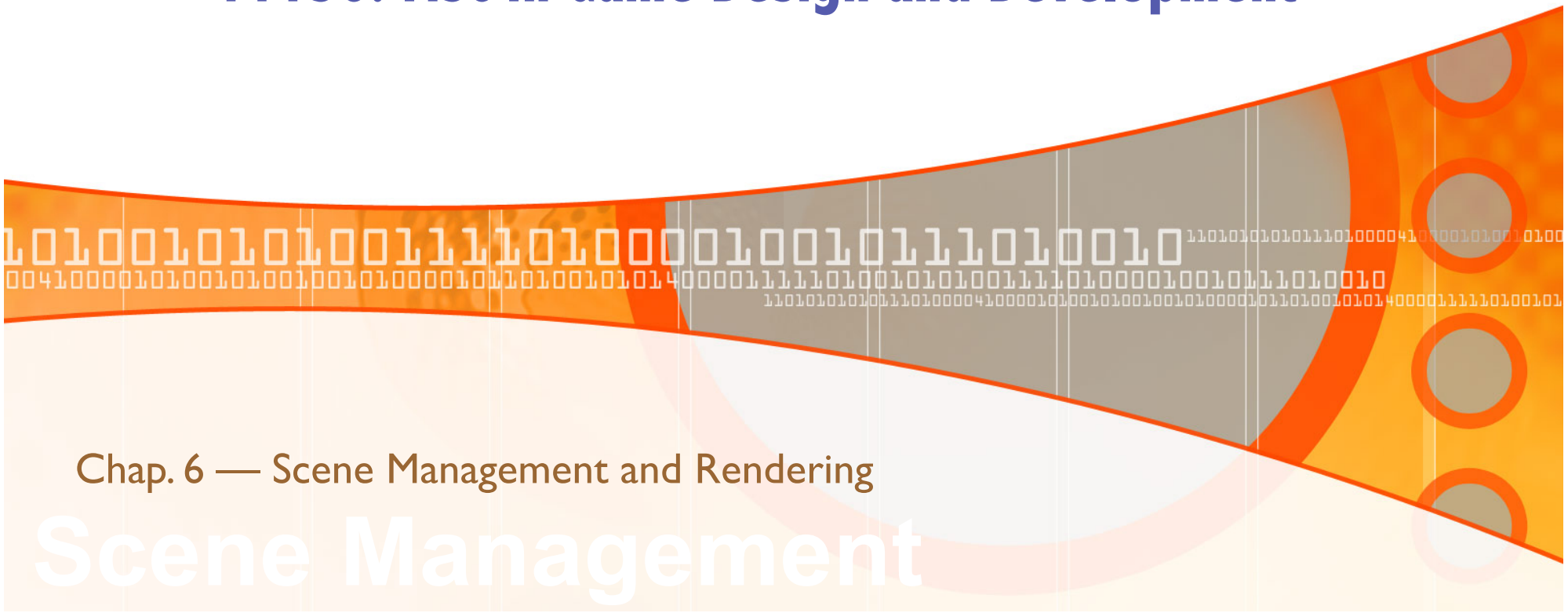
# Video Game Technologies

## 11498: MSc in Computer Science and Engineering

## 11156: MSc in Game Design and Development



Chap. 6 — Scene Management and Rendering

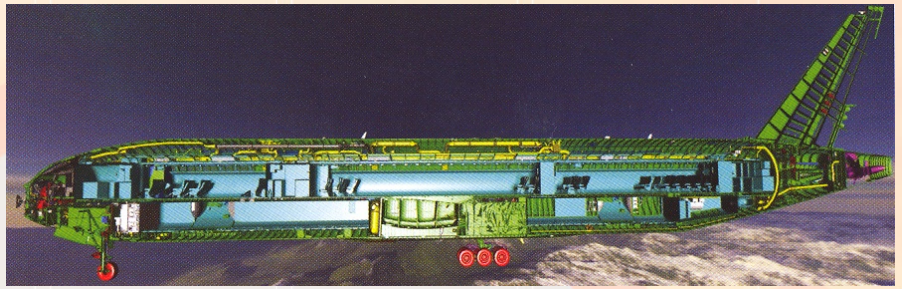
Scene Management

# Overview

- Scene Management *vs* Rendering
  - This chapter is about rendering acceleration techniques for complex scenes, i.e. scenes with large amounts of geometry
- Spatial Data Structures
  - At the core of such techniques/algorithms we find spatial data structures
- Culling Techniques
  - From those data structures, we are able to design culling techniques to determine the visibility of scene objects
- Level of Detail (LOD)
  - LOD techniques also help us to reduce the complexity of rendering scene objects.

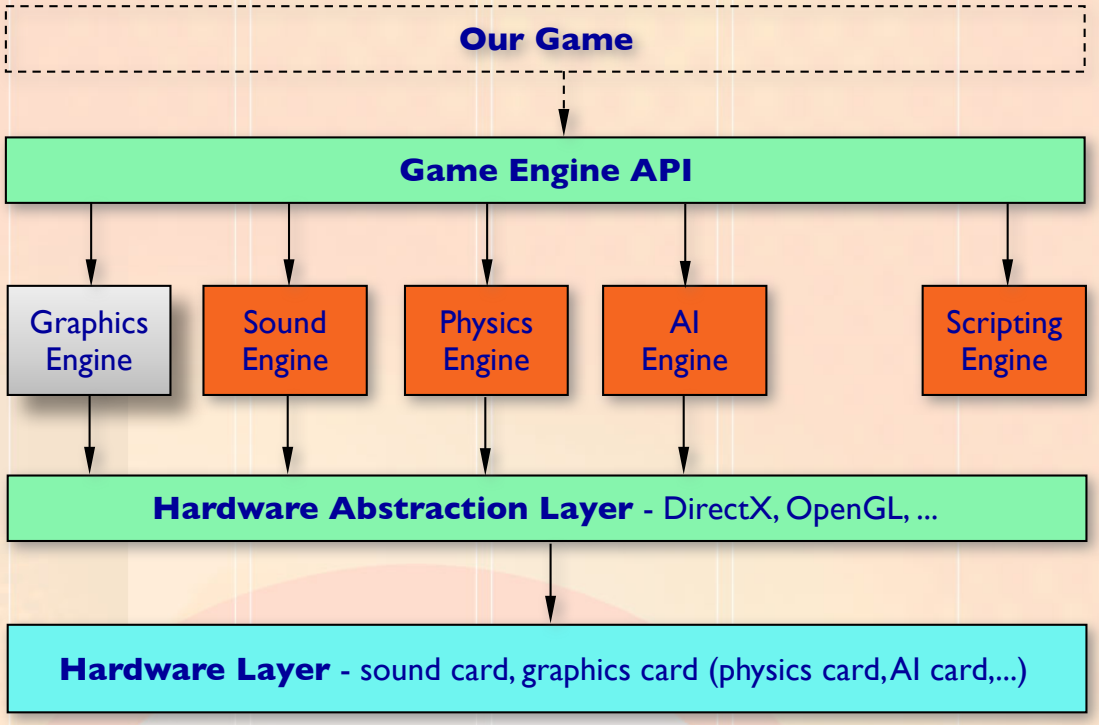
# Complex scenes: the Boeing 777 example

- A real-time renderer aims at satisfying four main performance goals:
  - More frames per second (between 65-80 fps)
  - Higher resolution and sampling rates
    - But more pixels more computations and time taken
  - More realistic materials and lighting
    - But more realism implies more computations and time spent
  - Increased scene complexity
    - Boeing 777 has 132,500 unique parts and 3,000,000 fasteners, which yields a polygonal model with over 500,000,000 polygons
    - Neither z-buffering nor ray tracing can handle such huge models without using acceleration techniques that reduce the number of computations



# Scene management and rendering

- The scene management system maintains a world full of objects and determines what gets drawn and in what order.
- The scene management layer deals primarily with objects.
- The rendering layer deals primarily with triangles and graphics state.



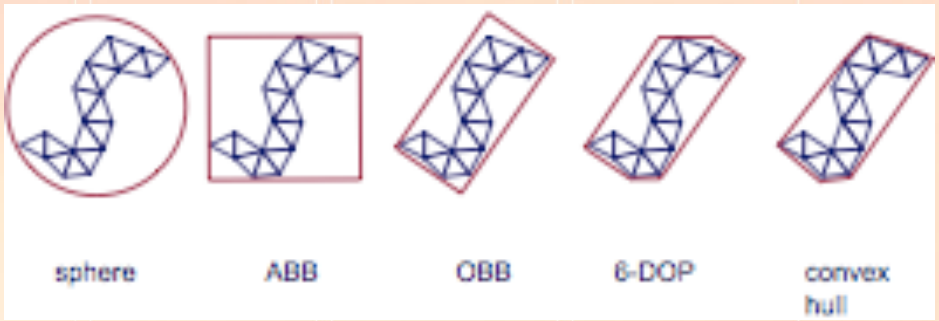
# Spatial Data Structures

# Spatial data structures

- ***Definition***: Data structure that organizes geometry in 2D or 3D or in some n-dimensional space
- ***Goal***: They are used to speed up queries about proximity and overlapping of geometric objects in a scene. Such spatial data structures are usually hierarchical to speed up queries, what leads to an improvement from O(n) to O(log n).
- ***Applications***: These queries are used in many operations:
  - Faster real-time rendering
  - Faster intersection testing
  - Faster collision detection
  - Faster ray tracing and global illumination
- ***Types***:
  - Bounding volume hierarchies (**BVH**s)
  - Binary space partitioning (**BSP**) trees
  - **Quadtrees**
  - **Octrees**

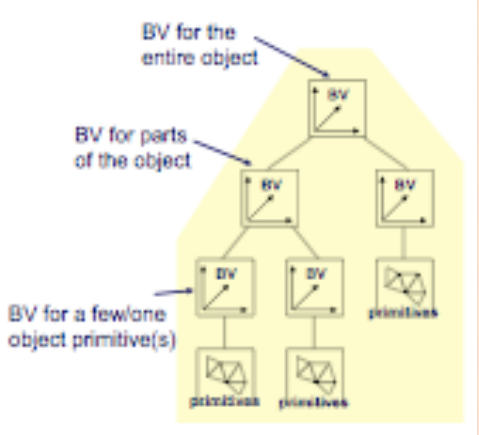
# Bounding Volume (BV)

- A ***bounding volume*** is a volume that encloses a set of objects
- It should be a simple geometrical shape.
- Examples:
  - Sphere
  - Axis-Aligned Bounding Box (AABB)
  - Oriented Bounding Box (OBB)
  - Discrete Oriented Polytope (*k*-DOP)
- A bounding volume does not contribute to the rendering image. Instead, it works as a proxy in place of the bounded objects to speed up rendering, selection, and other computations and queries.
- For example, BVH is often used for hierarchical view frustum culling.

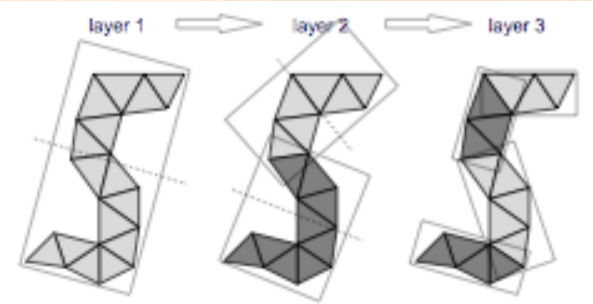


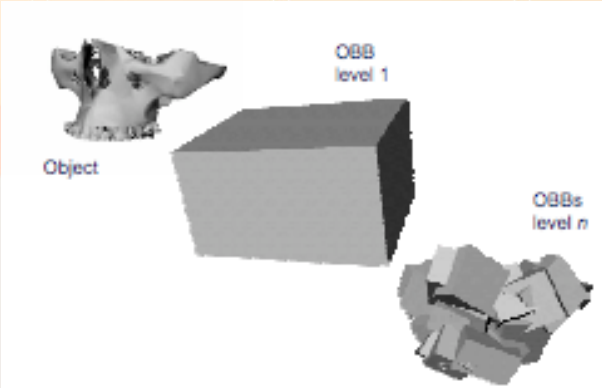
# Bounding Volume Hierarchies (BVH)

http://www.win.tue.nl/~hermanh/stack/bvh.pdf

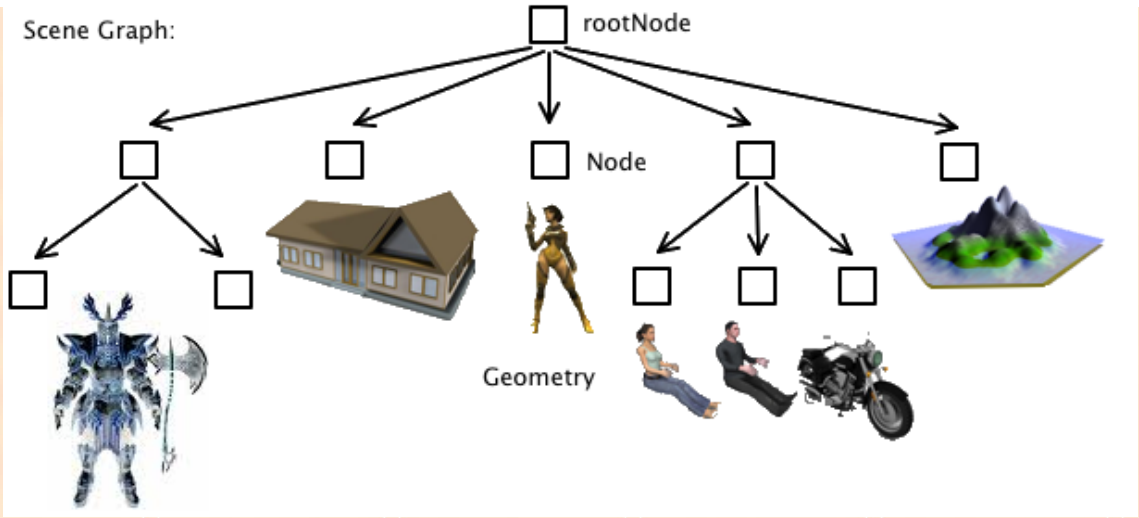



- Bounding volume tree (BV tree):
  - A tree of bounding volumes
  - Nodes contain bounding volume information
  - Leaves additionally contain object primitive information
- BVH generation:
  - Subdivision of bounding volumes to generate a hierarchy
  - Improved object approximation at higher levels



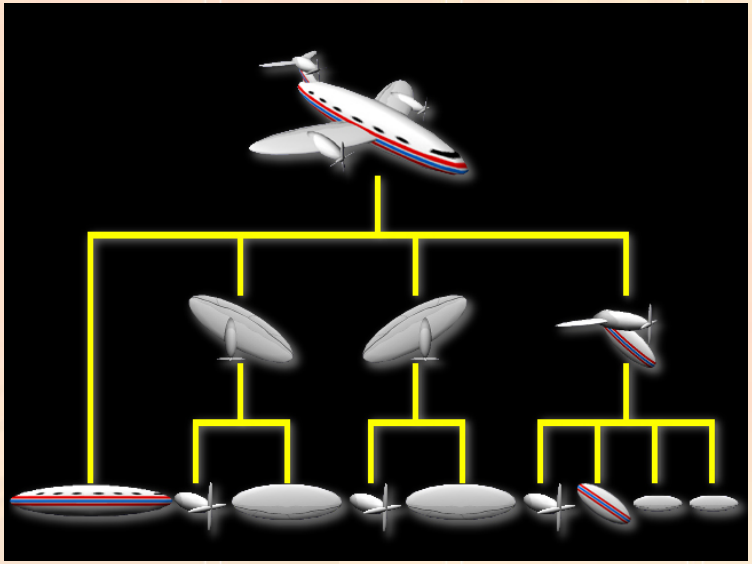



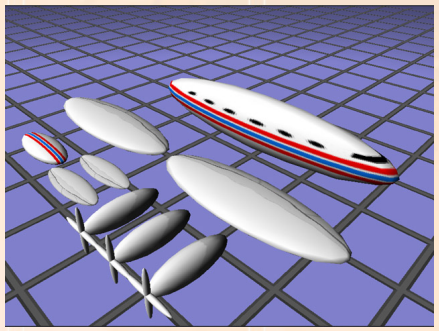
# Scene Graph



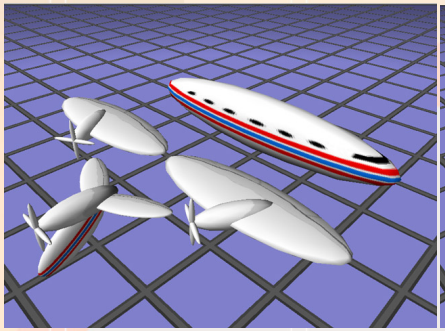

- BVH is the data structure that is used most often to wrap every single connected object of a scene, simply because:
  - It is simple to grasp
  - It is easy to program
- However, a BVH stores just geometry
  - Rendering is more than geometry
- The scene graph is an extended BVH with:
  - Lights, textures, transforms (translations, rotations, etc), and more …
  - *It represents the 3D world* (of the game)

# Building a toy airplane with a scene graph: example

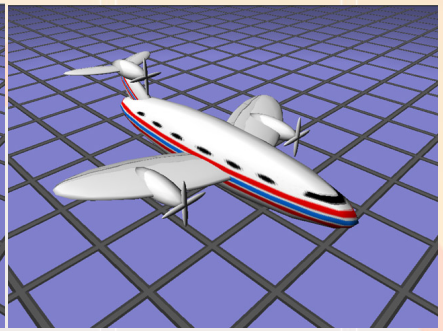




Put the parts on the table



Assemble together the related part



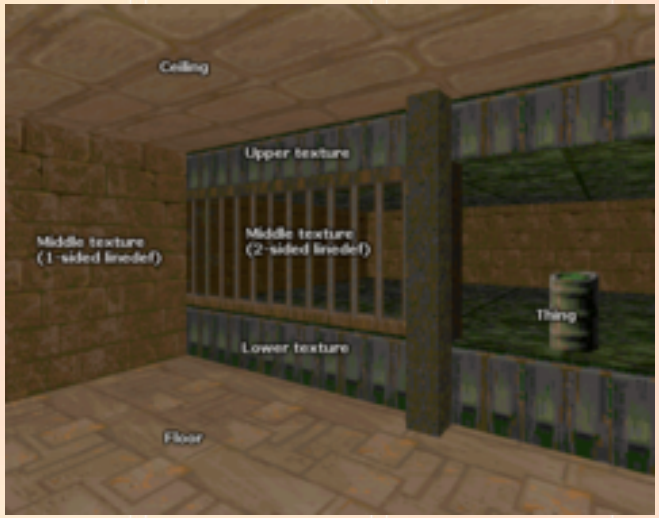
Assemble those into the final plane

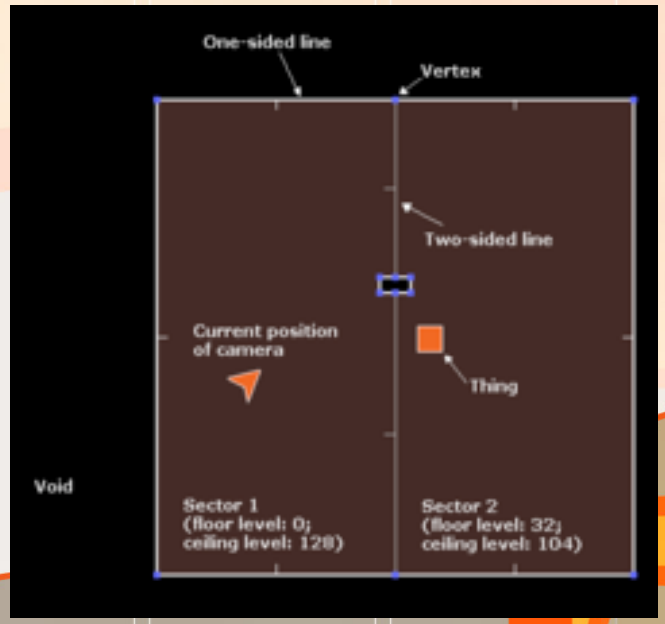
http://pesona.mmu.edu.my/~ypwong/virtualreality/java3d_tutorial_dave/slides/mt0000.htm

http://pesona.mmu.edu.my/~ypwong/virtualreality/java3d_tutorial_dave/slides/mt0026.htm

# Binary space partitioning (BSP)

- *Doom* makes use of a system known as binary space partitioning (BSP).
- It is because of this that it is not possible to move the walls in *Doom*; while doors and lifts move up and down, none of them ever move sideways.
- The ***level*** is divided up into a binary tree: each location in the tree is a "node" which represents a particular area of the level (with the root node representing the entire level).
- At each branch of the tree there is a dividing line which divides the area of the node into two subnodes. At the same time, the dividing line divides linedefs into line segments called "segs".



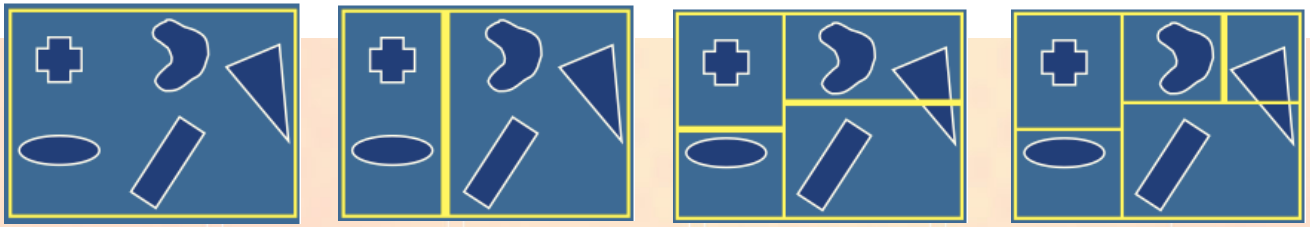



# Binary Space Partitioning (BSP) Trees

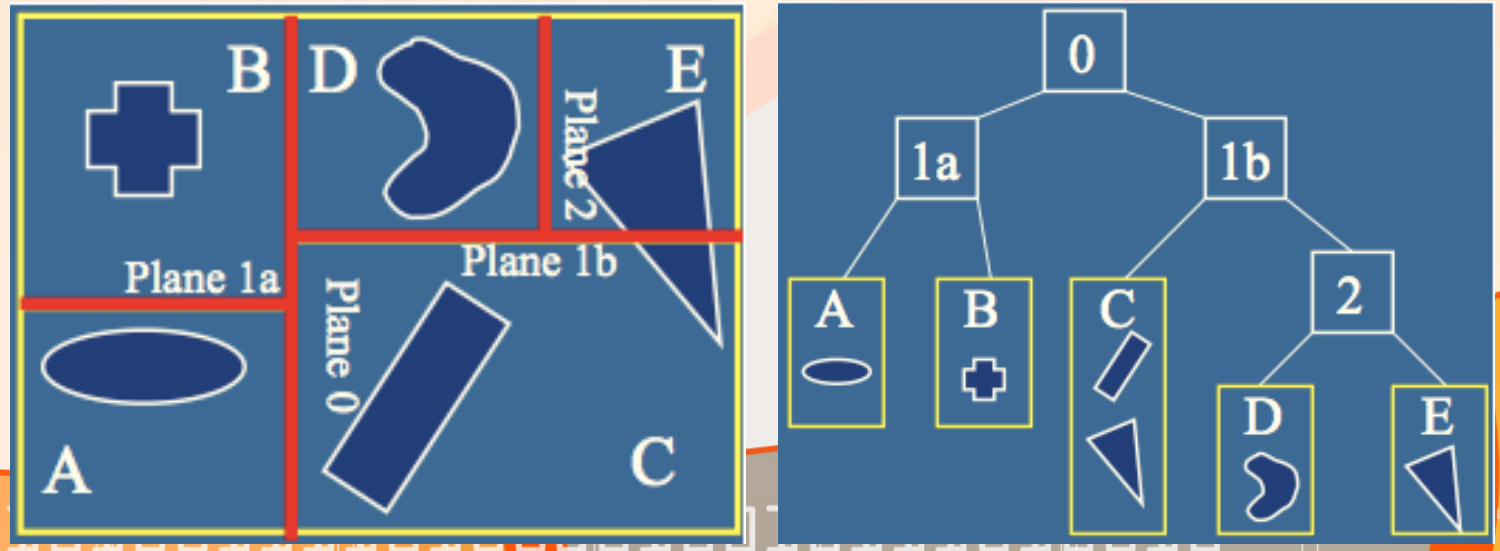
- ***Types***:
  - Axis-aligned BSP tree
  - Polygon-aligned BSP tree
- ***Idea***:
  - Divide space with a plane
  - Sort geometry into the space it belongs
  - Done recursively
- If traversed in a certain way, we can get the geometry sorted along an axis
  - Exact for polygon-aligned
  - Approximately for axis-aligned

http://www.gamedev.net/page/resources/_/technical/graphics-programming-and-theory/bsp-tree-faq-r657

# Axis-aligned BSP Trees

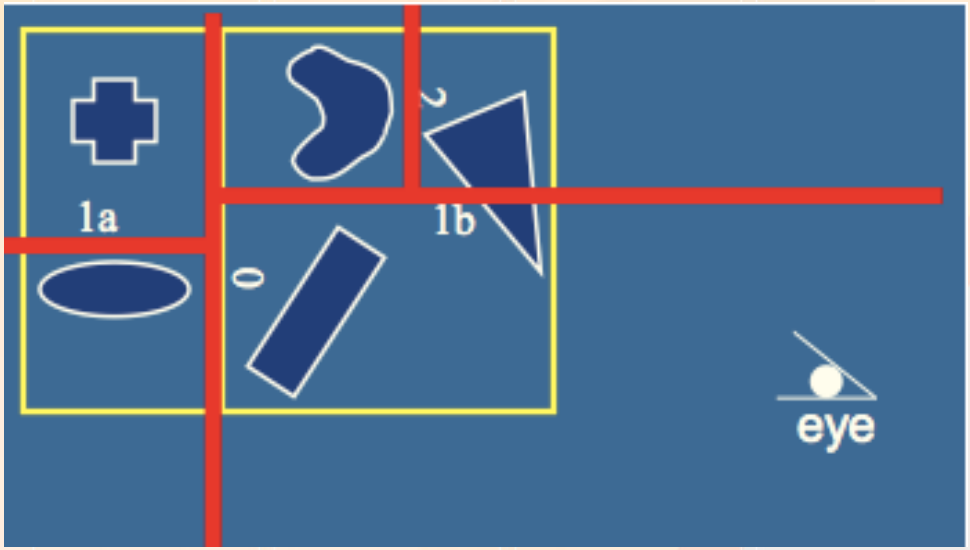


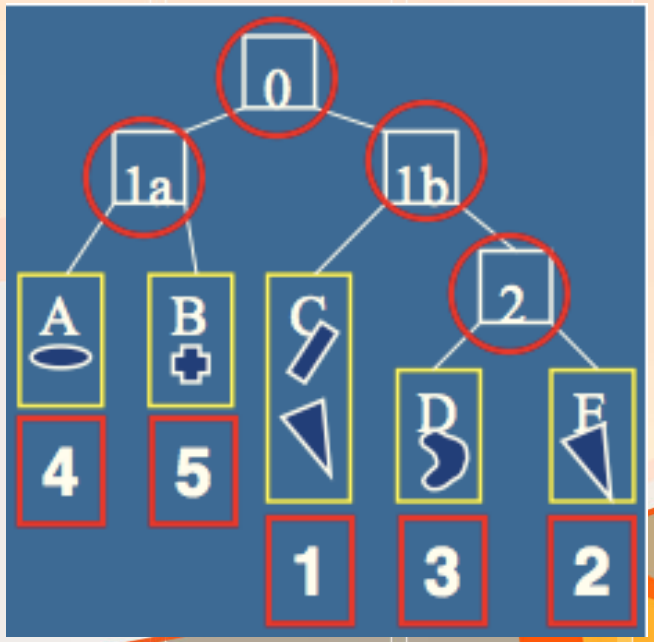
- Can only make a splitting plane along x,y, or z
- Each internal node holds a divider plane
- Leaves hold geometry
- Differences compared to BVH
  - Encloses entire space and provides sorting
  - The BV hierarchy can be constructed in any way (no sort)
  - BVHs can use any desirable type of BVT



# Axis-aligned BSP Trees: Rough Sorting

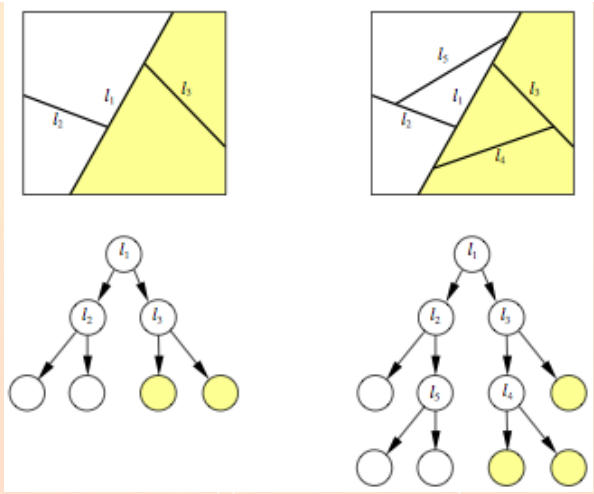
- Test the planes against the point of view
- Test recursively from root
- Continue on the ”hither” side to sort front to back
- Works in the same way for polygon- aligned BSP trees --- but that gives exact sorting

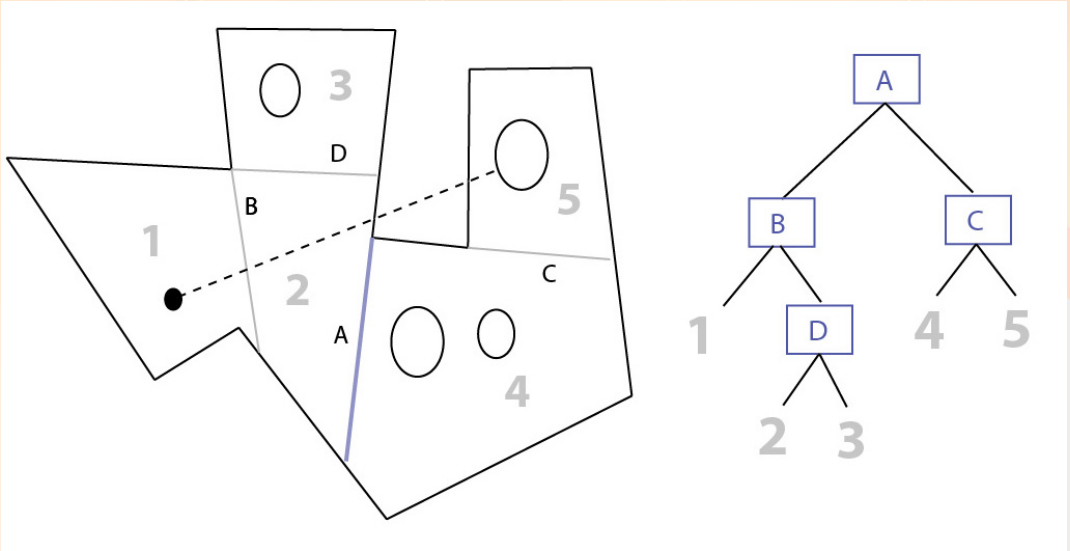


# Polygon-aligned BSP Trees

- Allows exact sorting
- Very similar to axis-aligned BSP tree
  - But the splitting plane are now located in the planes of the triangles





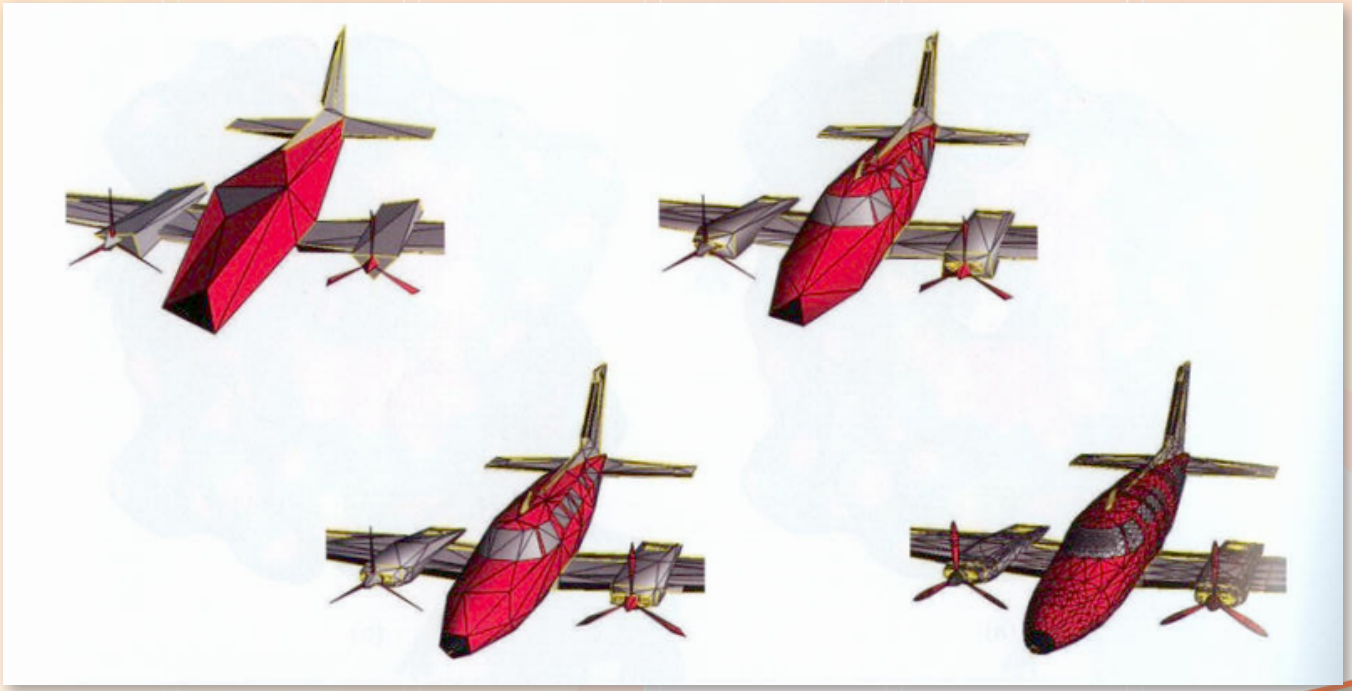
# Level of Detail (LoD)

# Discrete LoDs

- Several discrete LODs are prepared off line and dynamically switched based on simple distance comparisons (ideally it should take the camera FOV and resolution into account too)
- Tried and true method used for many years in flight simulation
- Basically no CPU overhead for algorithm itself
- Hardware is well adapted to rendering static display lists
- Additional memory required for storing LODs, but all low LODs together are usually smaller than the high LOD
- Can use sophisticated mesh decimation tools off line for preparing models
- If desired, techniques exist for cross dissolving (fading) between LODs
- Bottom line: very fast and very simple (quick & dirty)

# Progressive Meshes

- Automated mesh decimation and dynamic reconstruction technique
- Can take any model and then render it dynamically with any number of polygons (less than or equal to the original number)
- Requires about 1/3 extra memory (or more, depending on details of the implementation)



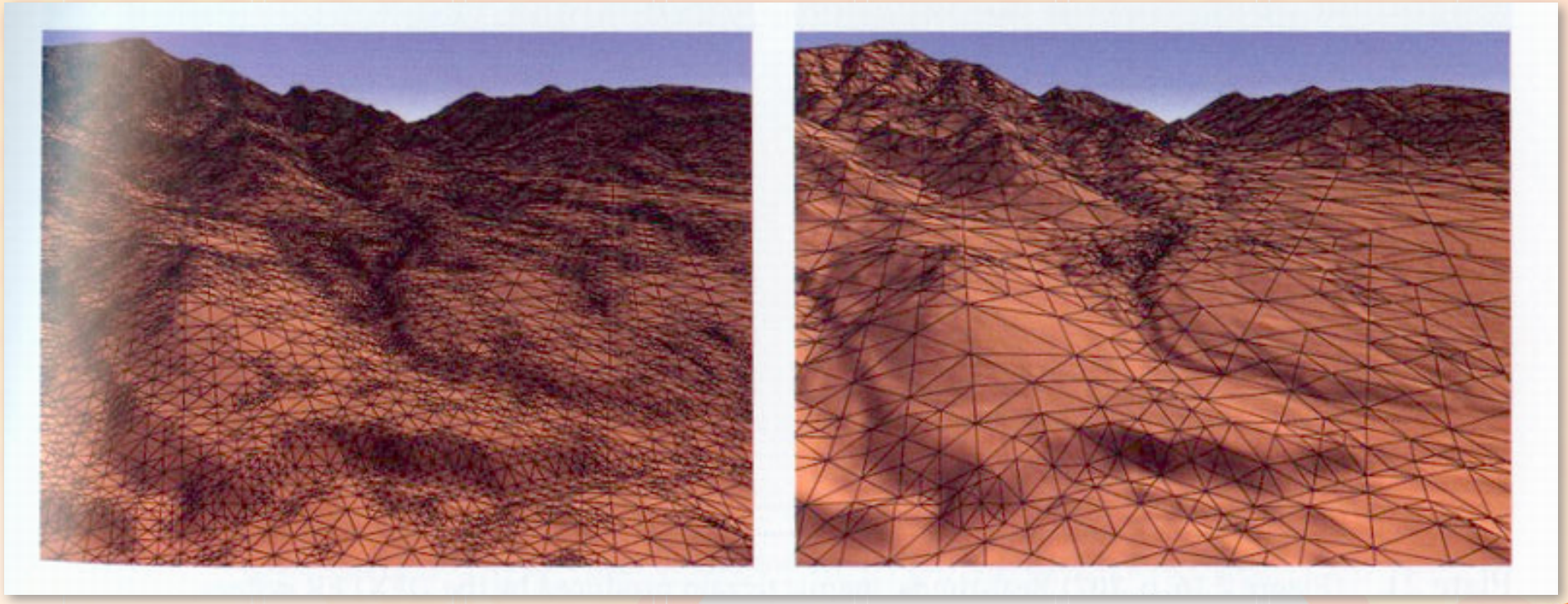
# Progressive mesh performance

- Rendering requires custom processing at the per-vertex and per-triangle level, and so the algorithm, while pretty efficient, has trouble competing with the simpler discrete LOD technique. In other words, even though it may be able to render fewer triangles, the per-triangle cost is higher, usually making the overall approach slower.
- Neat technology and useful research, but hasn't been too successful in video games just yet. Future graphics hardware may have better support for progressive meshes, and so they may be more important later. Also, mesh decimation technology is still very useful in off line modeling.

# Patches and subdivision surfaces

- Geometry is defined by a control mesh that explicitly describes a high order surface
- Surface is dynamically tessellated into polygons, based on camera distance and other visual quality controls
- Patches & subdivision surfaces allow for smooth, rounded surfaces to be described without using tons of memory
- Overall approach suffers from similar drawbacks to progressive meshes as far as its usefulness to games

# Terrain rendering



# LoD issues

- Color & lighting (normals) pops are more visible than geometry or silhouette pops
- Camera is often moving and there are often many dynamic objects in the scene. This can help mask LOD transitions
- Ideally, LOD should be pixel error based
- Load balancing:
  - LOD tolerances can be dynamically adjusted to provide a stable framerate
  - Uses timers to analyze how long the last frame took to render, then adjusts LODs so that current frame makes the most of available CPU time

# Summary

- Scene Management *vs* Rendering
  - Performance goals, scene graphs, and differences between scene manager and render
- Spatial Data Structures
  - Bounding volume hierarchies (BVHs), binary space partitioning (BSP) trees, quadtrees, and octrees
- Culling Techniques
  - Definition  and algorithms
- Level of Detail (LOD)
  - Discrete LODs, progressive meshes, multiresolution schemes, terrain modeling