


CS 563 Advanced Topics in Computer Graphics
Culling and Acceleration Techniques Part 1

by Mark Vessella

Introduction

- Acceleration Techniques
- Spatial Data Structures
- Culling

Outline for the Night

- Bounding Volume Hierarchies (BVH's)
- Binary Space Partitioning (BSP) trees
- Octrees
- Scene Graphs
- Culling
  - Backface
  - Clustered Backface
  - Hierarchal View Frustum
  - Portal
  - Occlusion

Background

- Artificial Intelligence
- Computer Graphics
- Geometry
- Visualization Skills

Why?

- The faster we can do computer graphics the more realistic we can make it look
- Throughout computer history there has always been applications for increased performance
- Today's computer graphics will (hopefully) look primitive 10 years from now
  - Just like 1995's graphics look to us now

Spatial Data Structures

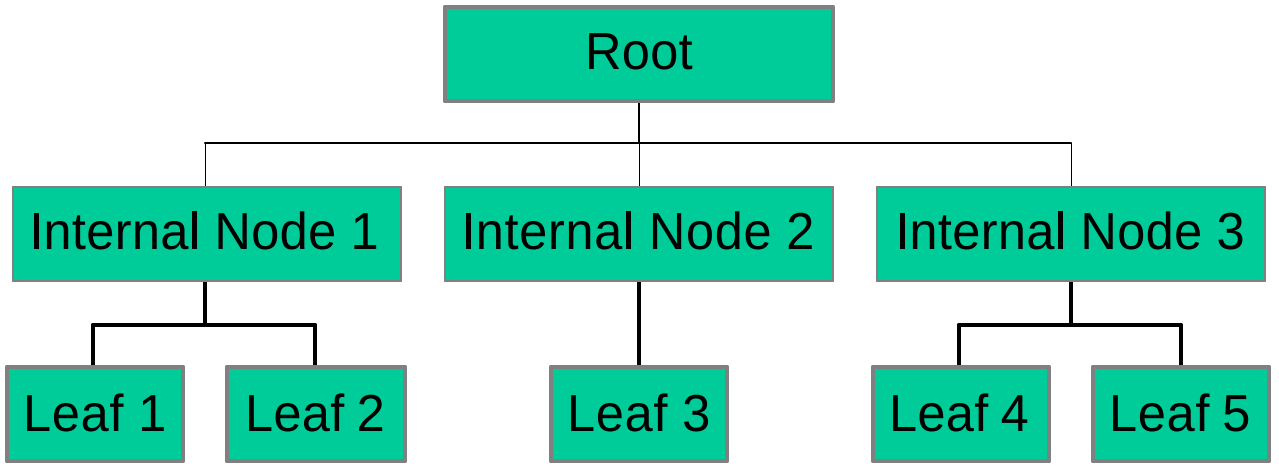
- “A spatial data structure is one that organizes geometry in some n-dimensional space.”[1]

  Courtesy of Tomas Akenine-Moller and Eric Haines, Real-Time Rendering

- Usually organized in a hierarchy
  - The top level node encloses the one below it

Bounding Volume Hierarchies (BVH's)

- A volume that encloses a set of objects
- Simpler geometric shape than the object it encloses
- Root, internal nodes, leaves:



- The root contains the whole scene
- The leaves contain to the objects in the scene

Properties of BVH's

- k-ary - Each internal node has k children
  - k=2, 4, and 8 are common
- Balanced – All leafs are at the height h or h – 1
- Full tree - All leafs are at the same height h

$$\text{nodes} = k^0 + k^1 + \cdots k^{h-1} + k^h = \frac{k^{h+1} - 1}{k - 1}$$

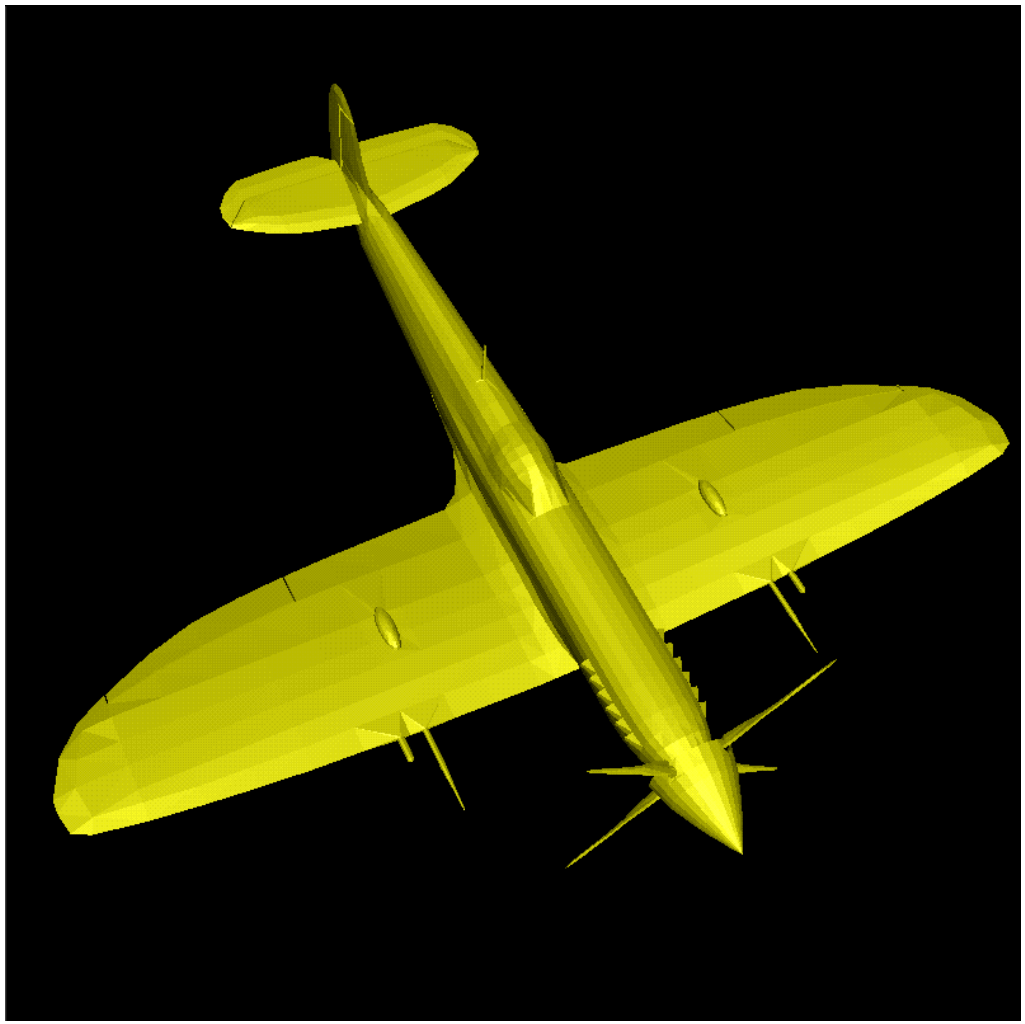
$$\text{leaves} = k^h$$

$$\text{internal nodes} = \text{nodes} - \text{leaves} = \frac{k^h - 1}{k - 1}$$

Types of BVH's

- Spheres
- Axis-aligned bounding boxes (AABB)
- Oriented bounding boxes (OBB)
- k-DOPs (Discrete Oriented Polytope)
  - Uses k/2 vectors to surround an object
  - AABB is the case of a k-DOP with k = 4

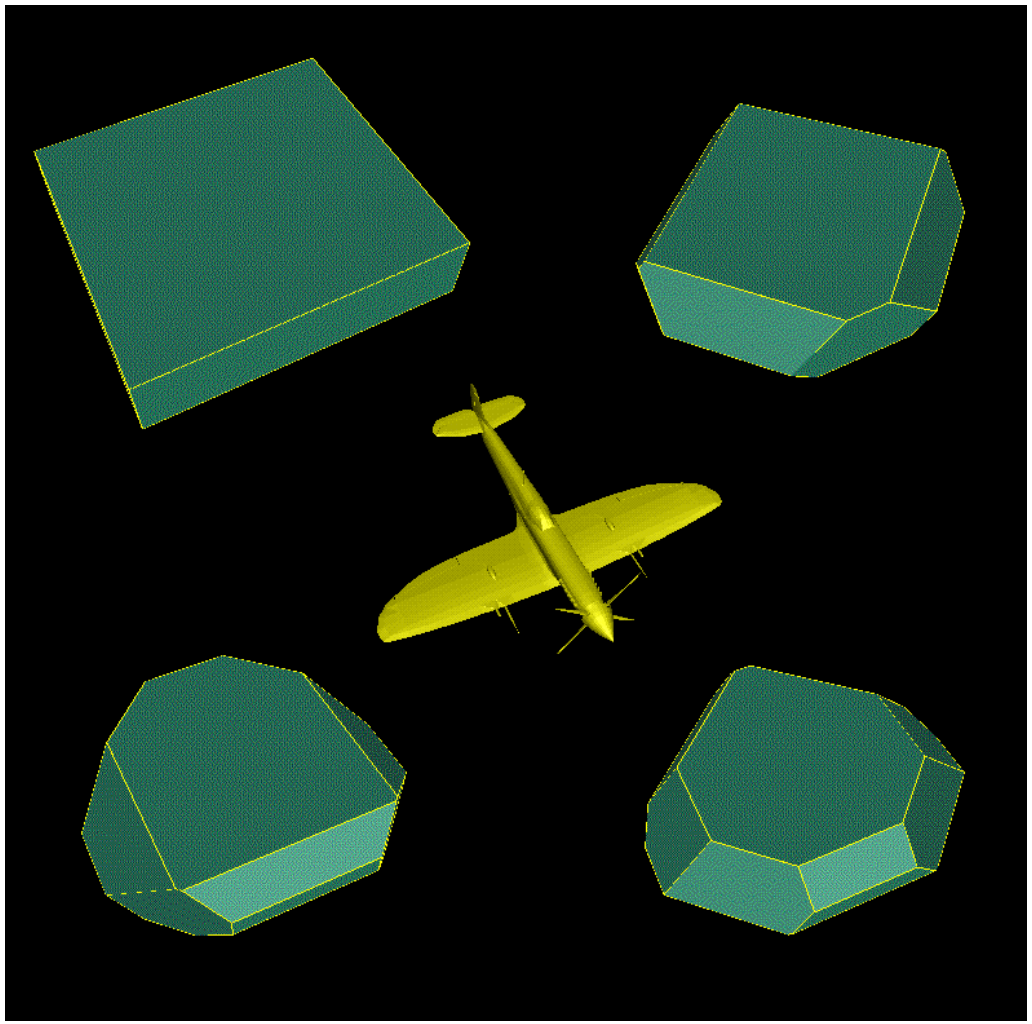
Example BVH – k-DOP



Sample image

Image courtesy of http://www.cosy.sbg.ac.at/~held/projects/collision/bvt.html

Example BVH – k-DOP(cont.)

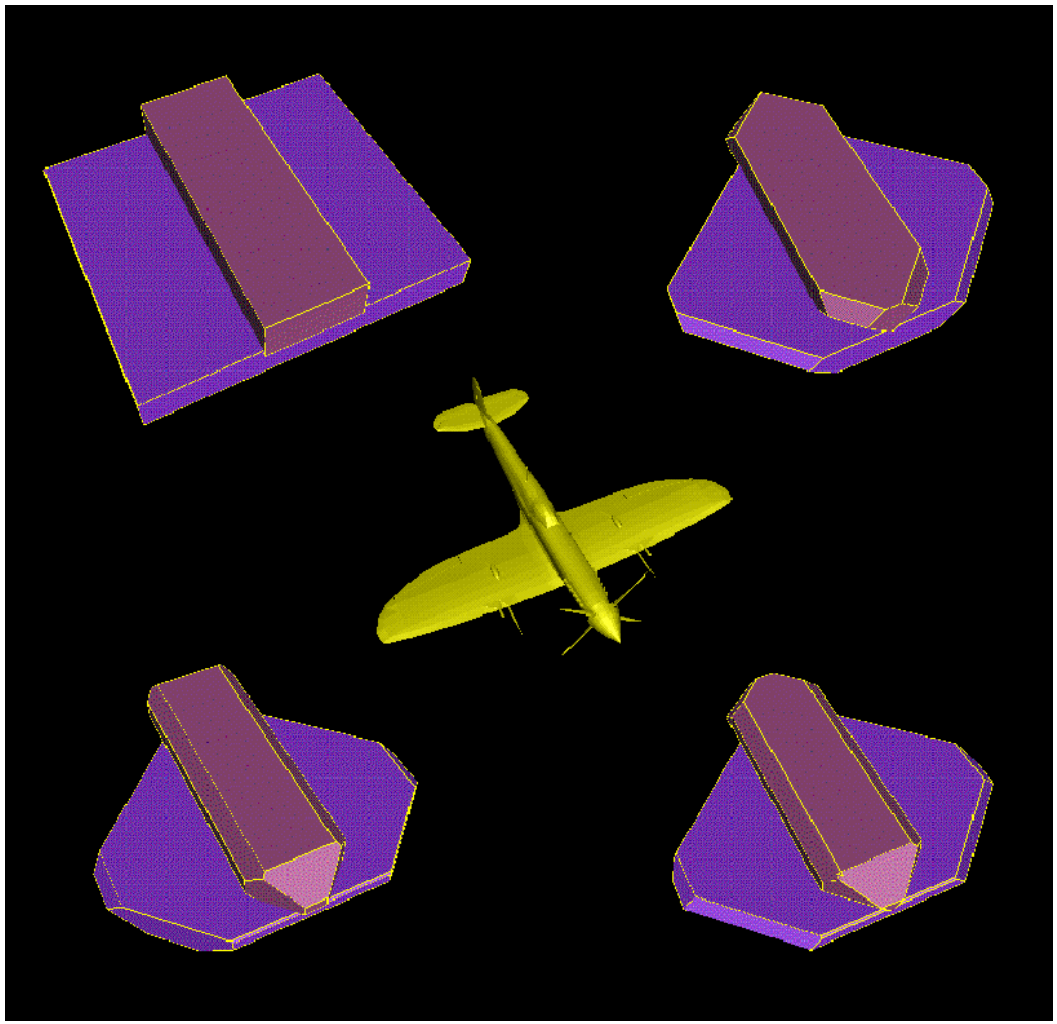


k=6, 14, 18, 26

Level 0

Image courtesy of http://www.cosy.sbg.ac.at/~held/projects/collision/bvt.html

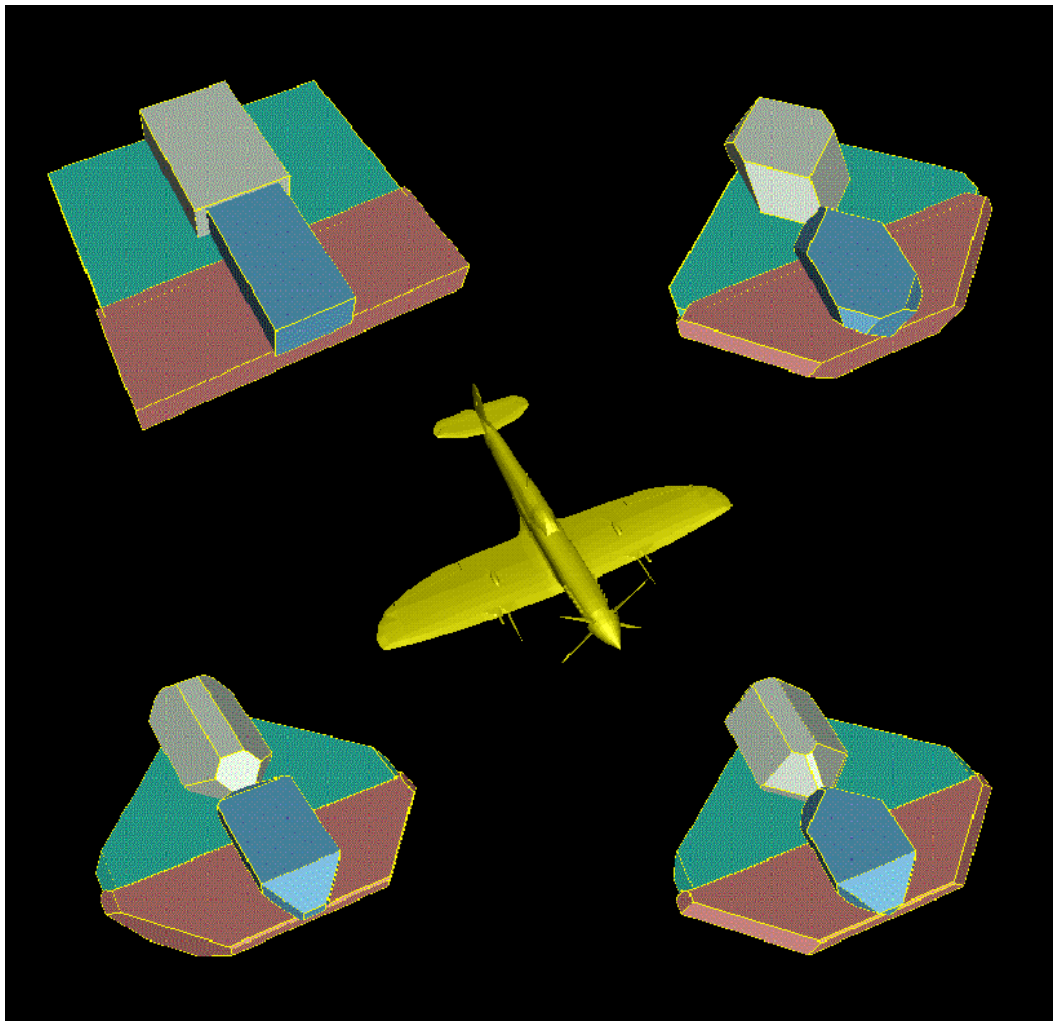
Example BVH – k-DOP(cont.)



k=6, 14, 18, 26
Level 1

Image courtesy of http://www.cosy.sbg.ac.at/~held/projects/collision/bvt.html

Example BVH – k-DOP(cont.)

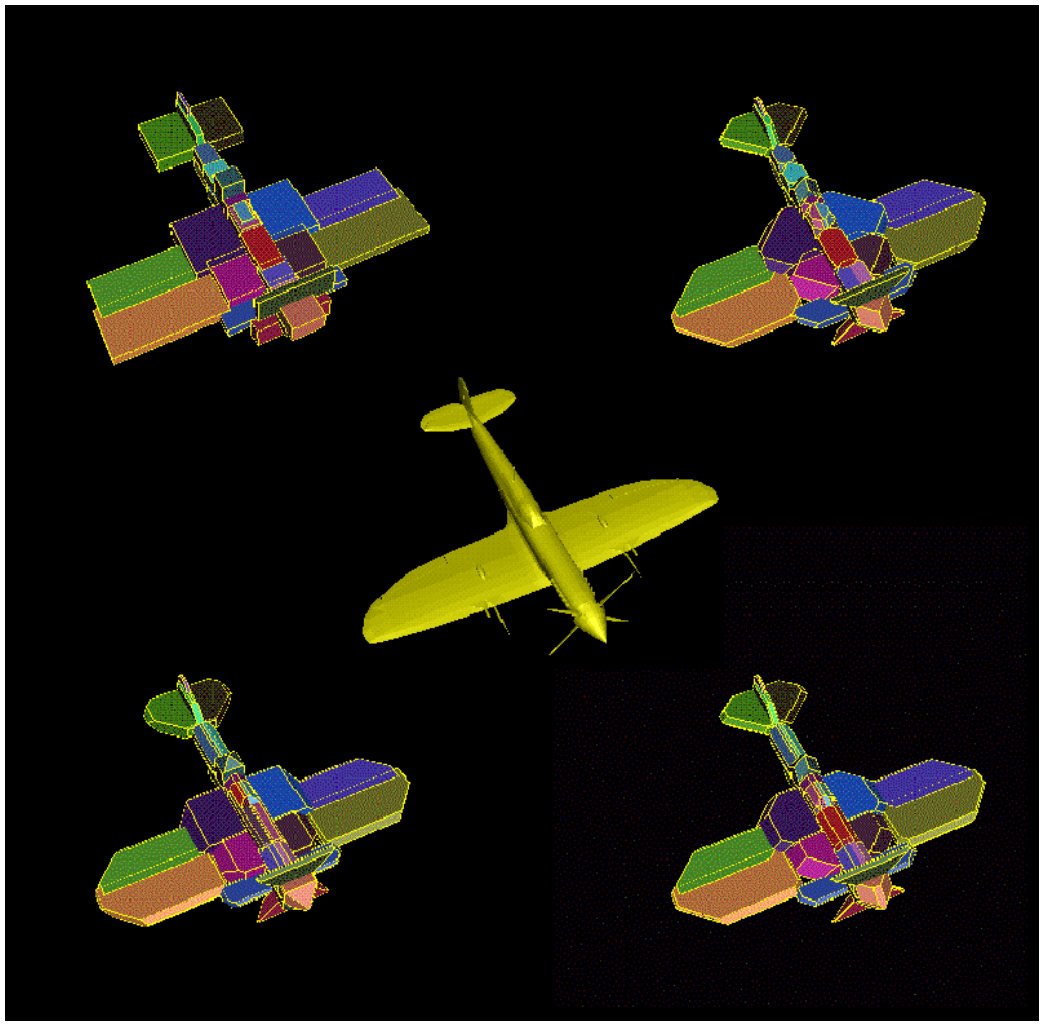


k=6, 14, 18, 26

Level 2

Image courtesy of http://www.cosy.sbg.ac.at/~held/projects/collision/bvt.html

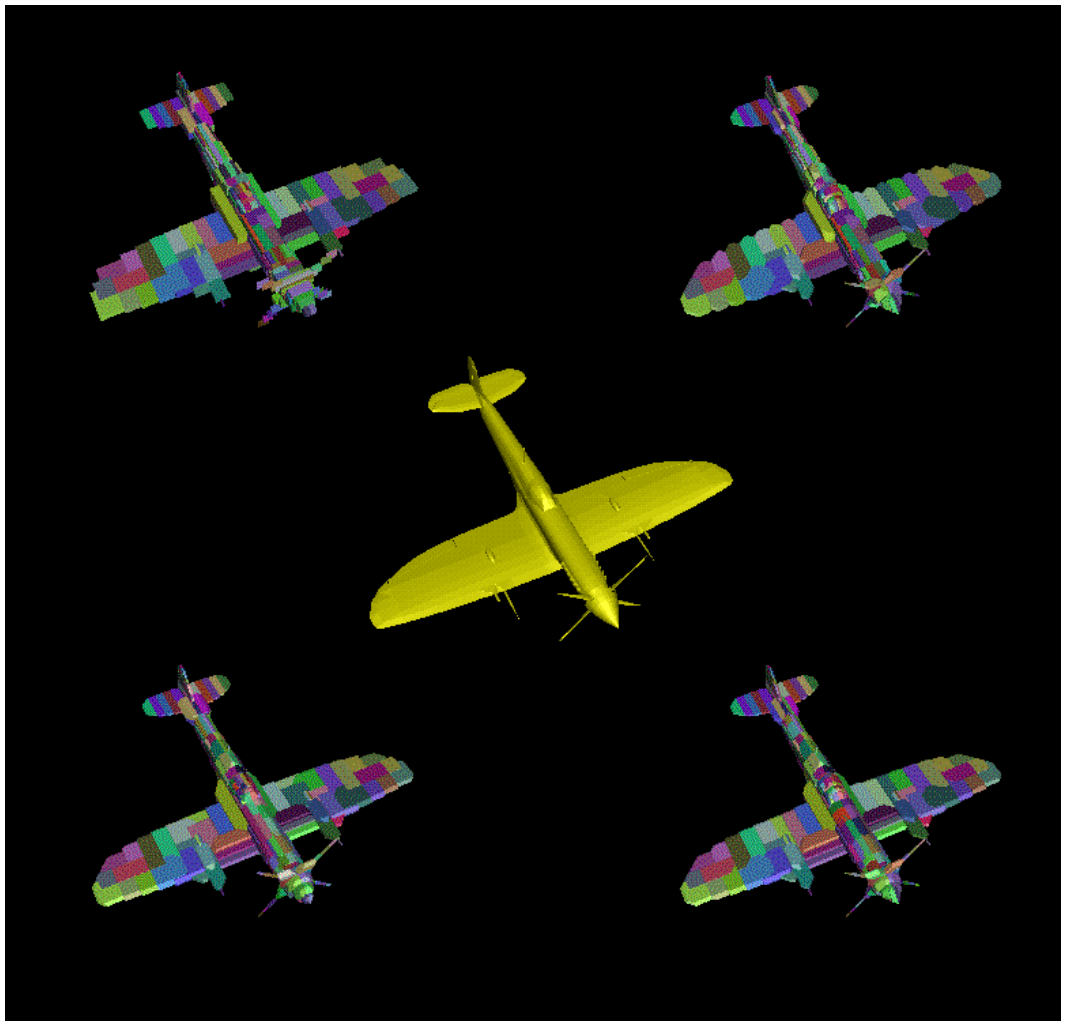
Example BVH – k-DOP(cont.)



k=6, 14, 18, 26
Level 5

Image courtesy of http://www.cosy.sbg.ac.at/~held/projects/collision/bvt.html

Example BVH – k-DOP(cont.)



k=6, 14, 18, 26
Level 8

Image courtesy of http://www.cosy.sbg.ac.at/~held/projects/collision/bvt.html

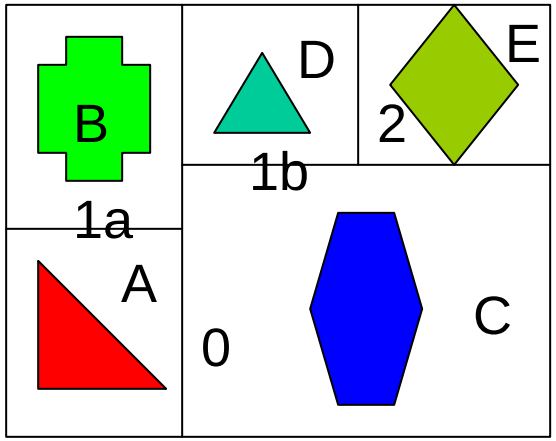
Binary Space Partitioning (BSP) trees

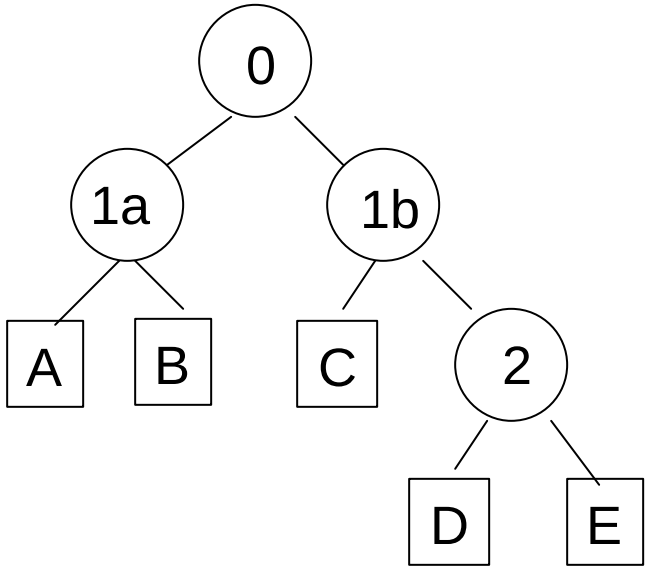
- They come in two main forms:
  - Axis-aligned
  - Polygon-aligned
- The geometrical contents (objects in the scene) of the trees can be sorted while in BVH's they can't

Axis-Aligned BSP Trees

- General Procedure
  - Enclose the scene in an Axis-aligned Bounding Box (AABB)
  - Recursively subdivide the box into smaller and smaller pieces
- Different techniques for dividing the boxes
  - Split each axis in order, i.e. first x, then y, and then z. Repeat.
  - Split the longest edge

Example Axis-Aligned BSP Trees

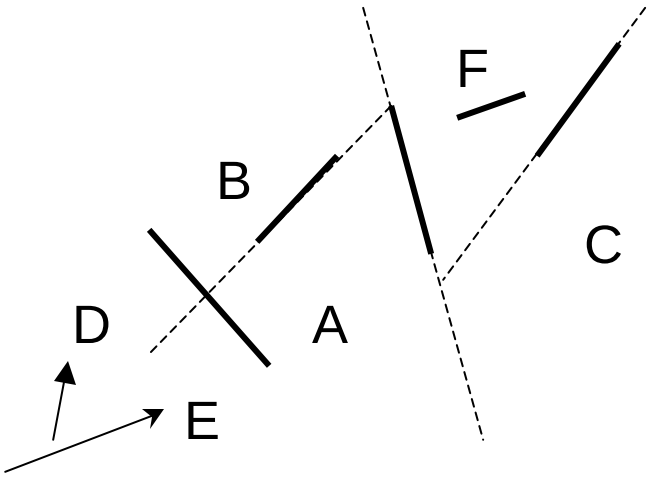


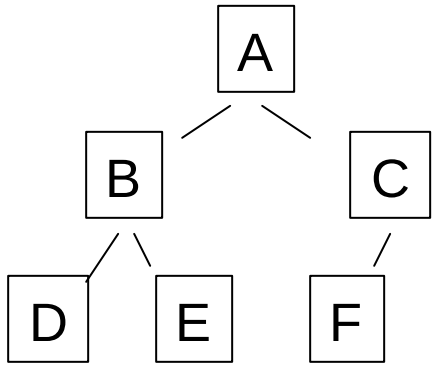


Courtesy of Spatial Data Structures and Culling Techniques, Han-Wei Shen

Polygon-Aligned BSP Trees

- A polygon is chosen as the divider
  - Many strategies for finding the polygons
- Results in a back-to-front (or front-to-back) order
  - The “Painters Algorithm” can be used
    - No Z-buffer required





result of split

Courtesy of Spatial Data Structures and Culling Techniques, Han-Wei Shen

Octrees

- Similar to axis aligned bounding box except all 3 axis of the bounding box are split simultaneously down the middle
  - This uniform splitting is known as “regular” which can make it more efficient in some situations

Procedure

- Start with the entire scene in an axis-aligned box
- Recursively split the box until a criteria is met
  - Maximum recursion level
  - Threshold reached for the number of primitives

Octrees Demo

- http://njord.umiacs.umd.edu:1601/users/brabec/quadtree/rectangles/cifquad.html

Problem with Octrees

- Some objects will be in more than 1 leaf node
- Solutions
  - Use the smallest box that the object will be contained in
    - A small object at the center will be in a large box
  - Split the object up
    - Results in more objects => more computations

Solution – Loose Octrees

- Increase the size of each bounding box
- The centroid of the box remains the same

Scene Graphs

- Add other characteristics to the tree
  - Textures, transforms, levels-of-detail, light sources
  - The nodes will hold these other characteristics
  - The leaves will still hold the geometric objects
- Example
  - You want to have a wheel of a car turn
  - Put a transform to rotate into an internal node
  - The leaf will contain the wheel

Directed Acyclic Graph(DAG)

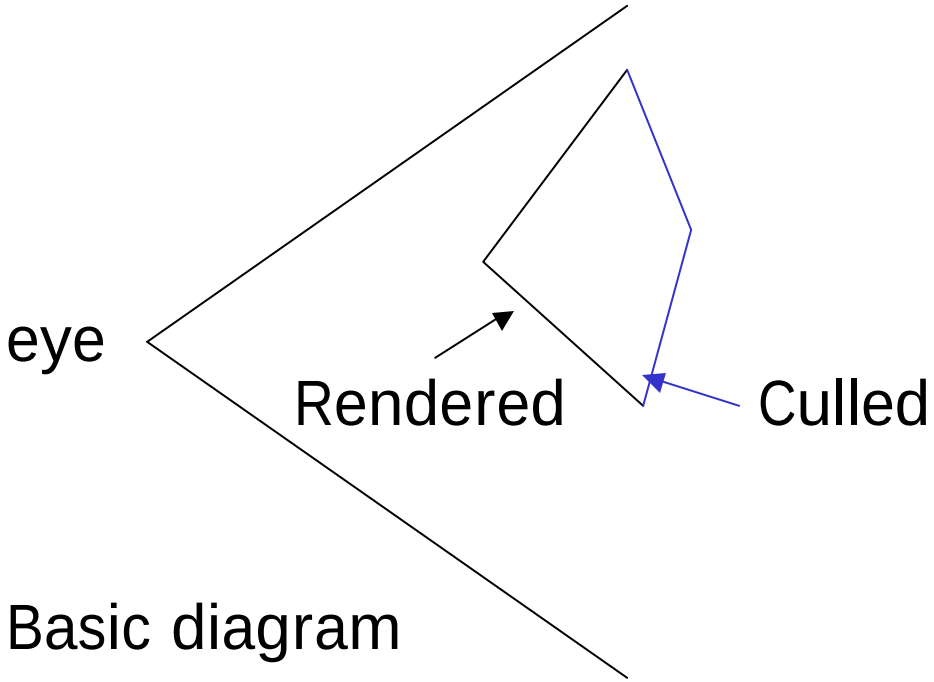
- When several nodes point to the same child node
  - Can’t have loops of cycles
  - Directional – Parent to child only
- Update the nodes on the way from the root to the leaves
- Update the bounding volume on the way from the leaves to the root
- In general DAG’s complicate matters and are avoided

Culling

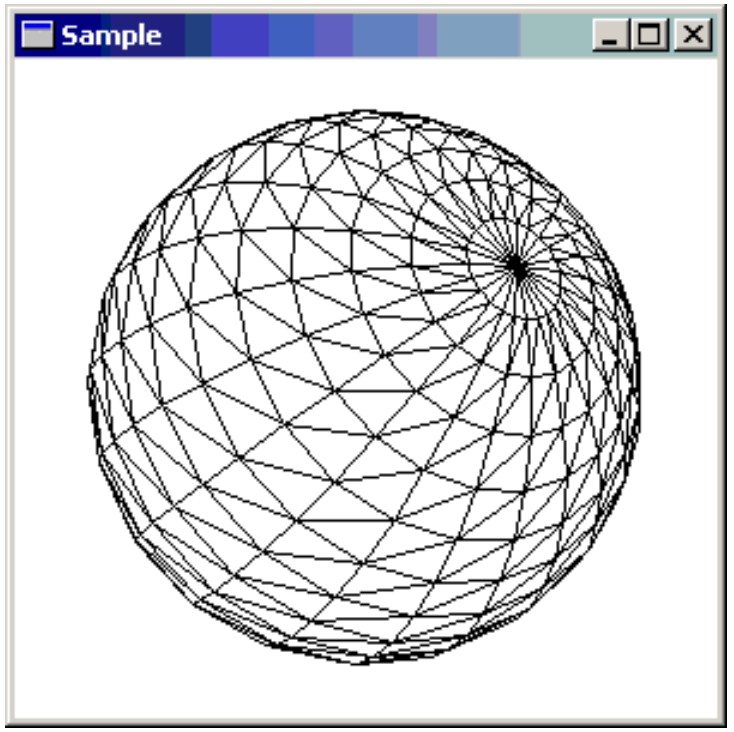
- The removal of objects that don't contribute to the final scene
- It can take place anywhere in the pipeline
  - The earlier in the pipeline the better
- Ideally only the Exact Visible Set (EVS) will be sent down the pipeline
  - Not practical
- Typical algorithms Potentially Visible Set (PVS)
  - Conservative – Fully includes the EVS
  - Approximate – May have errors but is faster

Backface Culling

- Removal of the backfacing portion of an opaque object



Basic diagram



Back portion of sphere is removed

Sample image courtesy of http://gpwiki.org/index.php/3D:Backface_Culling

Backface Culling - Procedure 1

- You are given the orientation of the vertices
- Compute the normal of the projected polygon

$$\vec{n} = (\vec{V}_1 - \vec{V}_0) \times (\vec{V}_2 - \vec{V}_0)$$

- If the z-component is negative then it is backfacing
- Implemented immediately after screen-mapping
- Decreases the load on the rasterizer
- Increases the load on the geometry stage

Backface Culling - Procedure 2

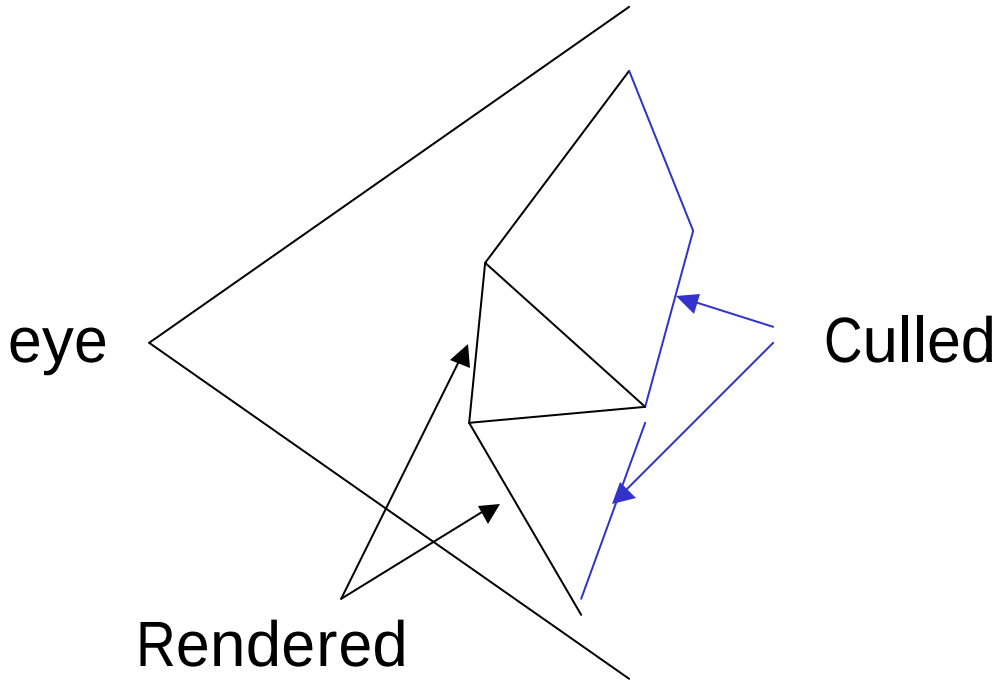
- Create a vector from an arbitrary point on the plane in which the polygon lies to the viewers position
- Compute the dot product of this vector and the normal of the polygon
- The sign of the dot product determines how it is facing (if it is negative it is backfacing and vice-versa)
- Can be done after the model transform or after both the model and view transforms
- It is done earlier in the geometric stage

Backface Culling Conclusion

- The tests are essentially the same – just that they are done at different places in the pipeline
- The first test is safer because edge-on polygons will be back-facing in eye-space and front-facing in screen-space
  - This is caused by round-off

Clustered Backface Culling

- The removal of a whole set of polygons with just one test



Clustered Backface Culling - Procedure

- Create a cone containing all of a set of polygons normals and points
- Will leave you with $\vec{n}$ and a (half-angle of the cone)
- If

$$\vec{n} \cdot \left(\frac{e - f}{\|e - f\|} \right) \geq \sin(a), \text{front facing}$$

$$-\vec{n} \cdot \left(\frac{e - b}{\|e - b\|} \right) \geq \sin(a), \text{back facing}$$

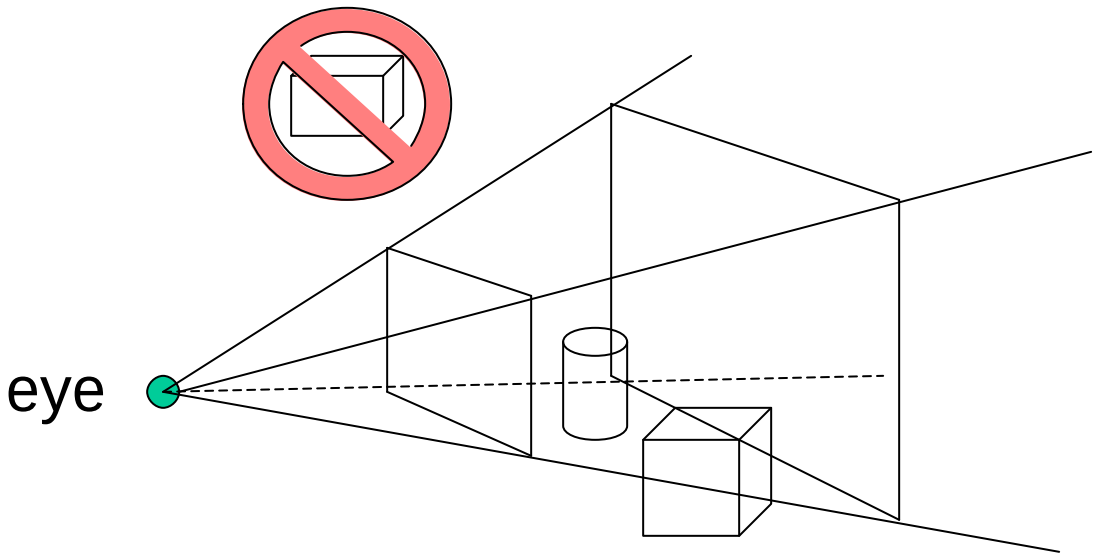
  - e is the location of the viewer
  - f and b are the apexes of the front and backfacing cones

Clustered Backface Culling cont.

- Can also be used to avoid lighting calculations on surfaces away from light source
- There are other techniques (p. 362 of RT Rendering)

Hierarchical View Frustum Culling

- Only objects totally or partially inside the view frustum need to be rendered
- The BV's outside the view frustum do not need to be sent down the pipeline



Courtesy of Spatial Data Structures and Culling Techniques, Han-Wei Shen

Procedure

- Create a BV Hierarchy
- Start at the root
- **If** the BV is outside the frustum don't recurse any further
- **Else if** it intersects a node test its children
- When you get to a leaf send it down the pipeline
  - It is still not guaranteed to be in the frustum
    - Clipping takes care of that
- **Else** the BV is fully inside the frustum then all its contents must be inside the frustum
  - Frustum testing is not needed for the rest of the tree

Hierarchical View Frustum Culling cont.

- Operates in the application stage(CPU)
  - Geometry and rasterizer stages can benefit
- For a large scene only a small portion of it will be seen
- BSP trees – Used mainly for static scenes because it takes too long to update the corresponding data structure
- Polygon aligned, axis aligned, and octrees can be used
- Can exploit frame-to-frame coherency

Portal Culling

- Used in architectural models and 3D games
  - Can also create mirror reflections or one-way portals used in games
- Do view frustum culling through each portal, (window, door, ...) in a scene

Portal Culling - Procedure

- Render the geometry of the room you are in using the view frustum culling
- Go through each portal and render the additional geometry using the frustum created by the portal
- Continue until there are no more portals to go through

Portal Culling Diagram

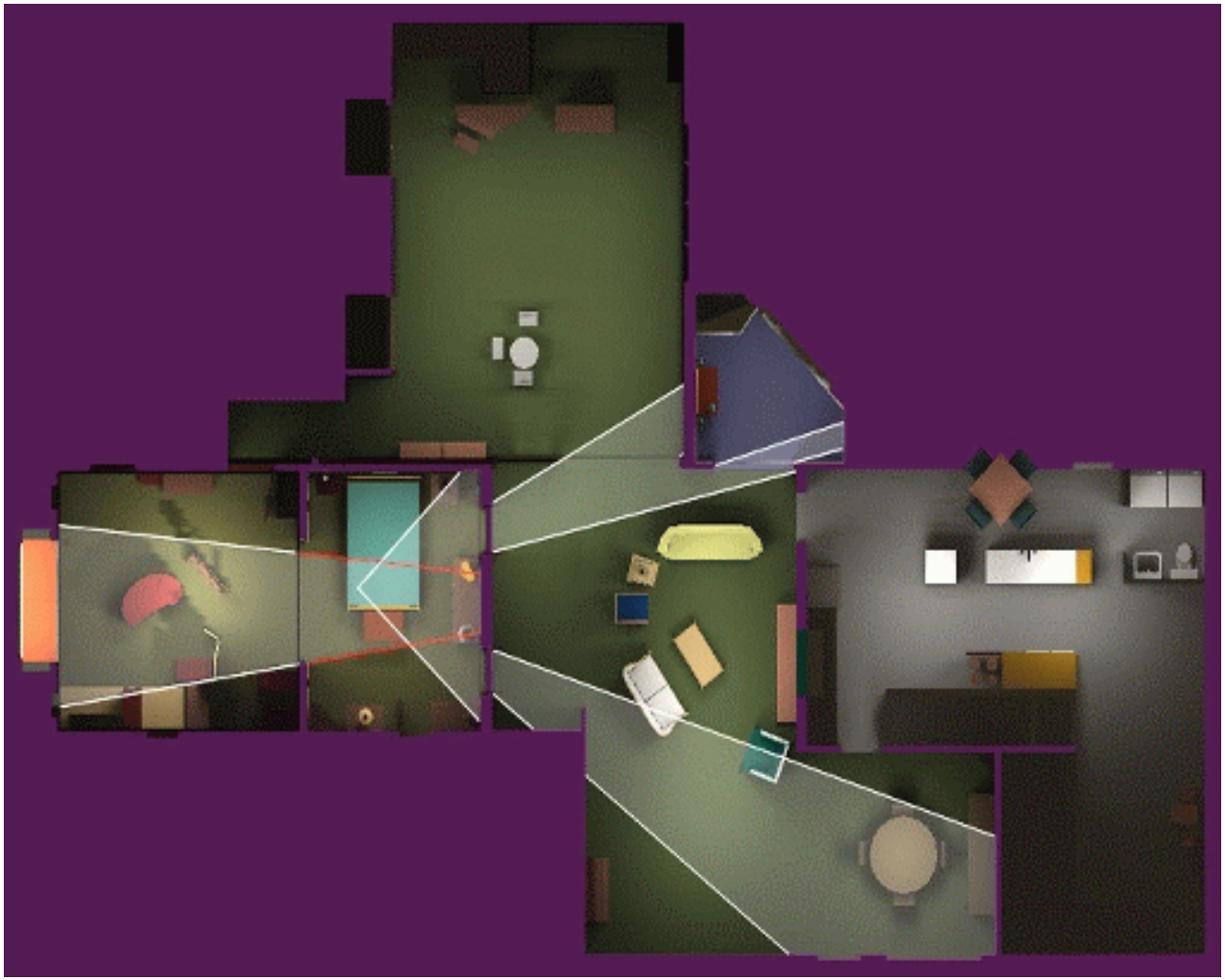


Image Courtesy of http://www.cs.virginia.edu/~luebke/publications/images/portals.plate2.gif

Scene from Splinter Cell

Image Courtesy of http://www.xbox.com/media/games/tomclancysplintercell/sim-tomclancysplintercell-0005.jpg

Occlusion Culling

Main Idea: Cull items based on depth to avoid the processing of objects that are eventually occluded by objects closer to the viewer.

Techniques:

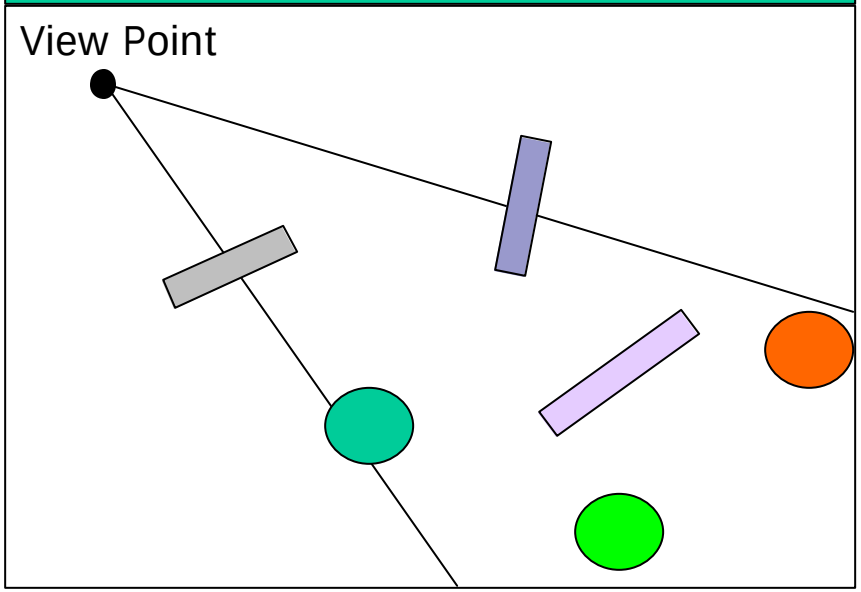
- Point Based
- Cell Based
- Image Space
  - Occlusion Query
  - Hierarchical Z-Buffer
  - The **H**ierarchical **O**cclusion **M**ap

Cell & Point Based

Main idea: Intuitive techniques that employs a different viewer reference points for occlusion culling

- These techniques are easy to implement, but could be more effective

Objects occluded are invisible to a single view point.



Objects occlude are invisible to all points on cell.

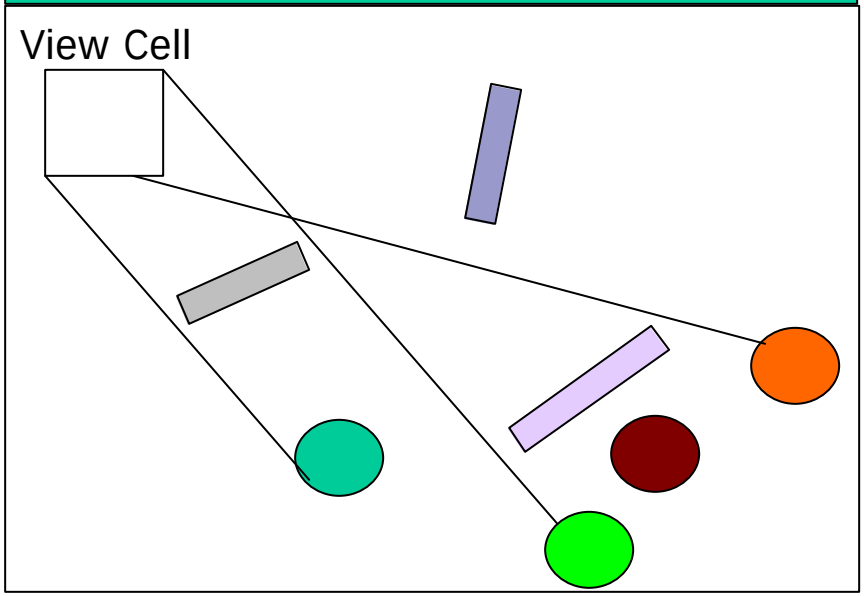


Image Space Based

Main Idea: "Visibility testing in 2D after some projection*"

Techniques:

- Hardware Occlusion Query
- Hierarchical Z-Buffer
- H.O.M.

Note: All of these techniques can use the benefit of hardware acceleration

General Algorithm

```
OcclusionCulling (G)
Or = empty
For each object g in G
    if (isOccluded(g, Or))
            skip g
    else
            render (g)
          update (Or)
    end
End
```

G: input graphics data
Or: occlusion hint

*Definition from *Real-Time Rendering*, Akenine-Moller, Haines

Hardware Occlusion Query

Main Idea: Query hardware to find out if a set of bounding polygons is visible based on the current contents of the Z-Buffer. Returns 0 if occluded, Else 1.

Technique:

1. Scan convert bounding polygons
2. Compare depths to Z-Buffer
3. Decision:
   - If completely hidden, object can be safely culled
   - Else render object

Note: This feature is available in both ATI and nVidia

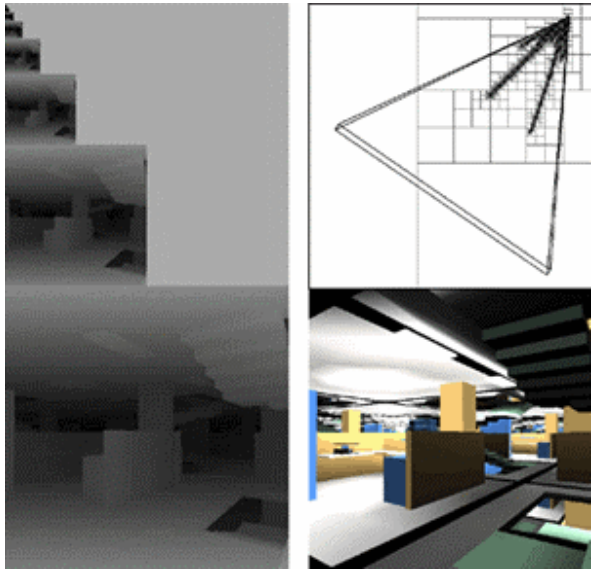
Hierarchical Z-Buffer

Main idea: Maintain the scene model in an Octree and the Z-Buffer as an Image Pyramid.

Technique:

- Scene is arranged into an Octree which is traversed top-to-bottom and front-to-back
- A Z-pyramid is incrementally built during rendering
- Compare Octree nodes and Z-pyramid for occlusion

Note: *RT Rending* states that this technique not in hardware yet, but ATI has had this implemented in their RADEON cards for some time now using a 3 level Z-Pyramid.

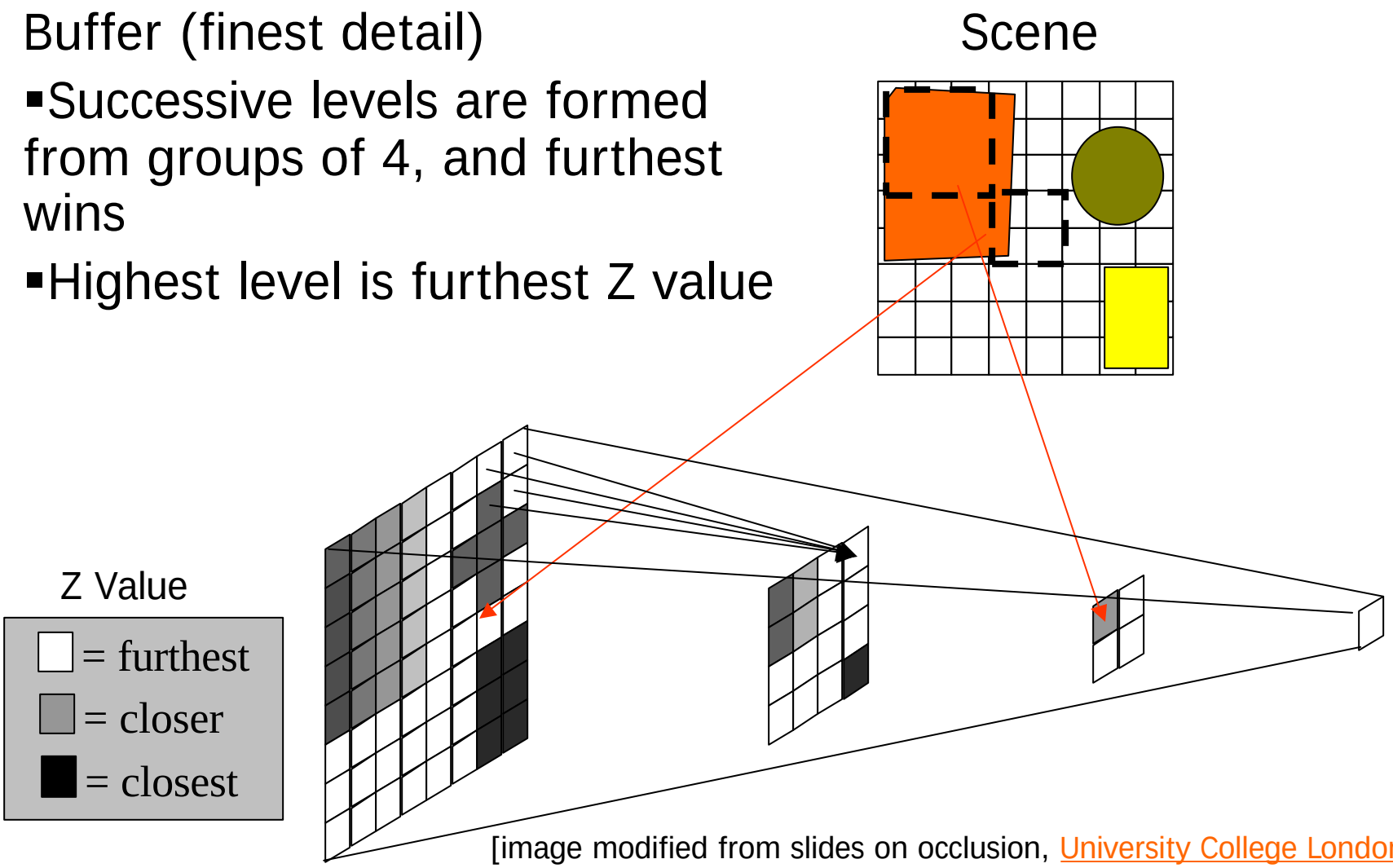


[image courtesy of Gamasutra.com]

Z-Pyramid

Technique:

- Lowest level constructed from Z-Buffer (finest detail)
- Successive levels are formed from groups of 4, and furthest wins
- Highest level is furthest Z value



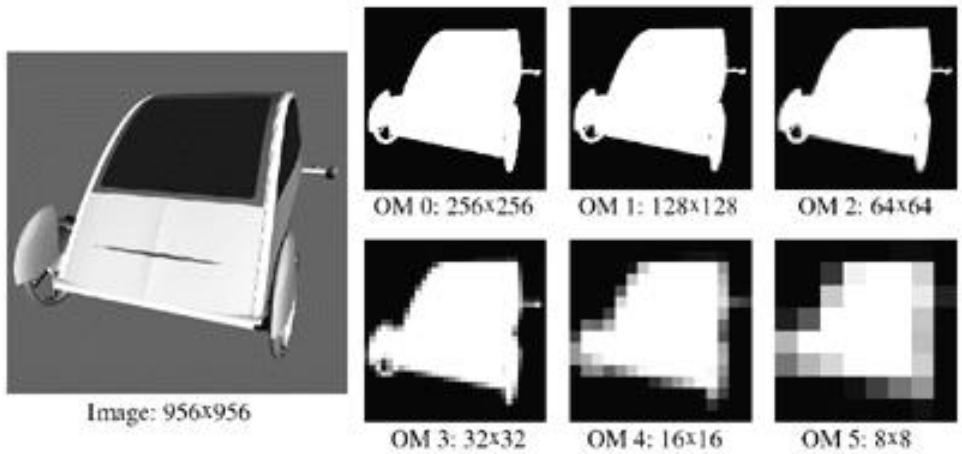
[image modified from slides on occlusion, University College London]

H.O.M

Main idea: Hierarchical Occlusion Map is built from a set of occluders rendered to occlussion maps in the form of an image pyramid

Pyramid Build Technique:

1. Clear buffer to black
2. Pick Occluders
3. Render Occluders in white to buffer
4. Recursively create higher levels using a Low Pass Filter



[image courtesy of Gamasutra.com]

H.O.M Overlap Test

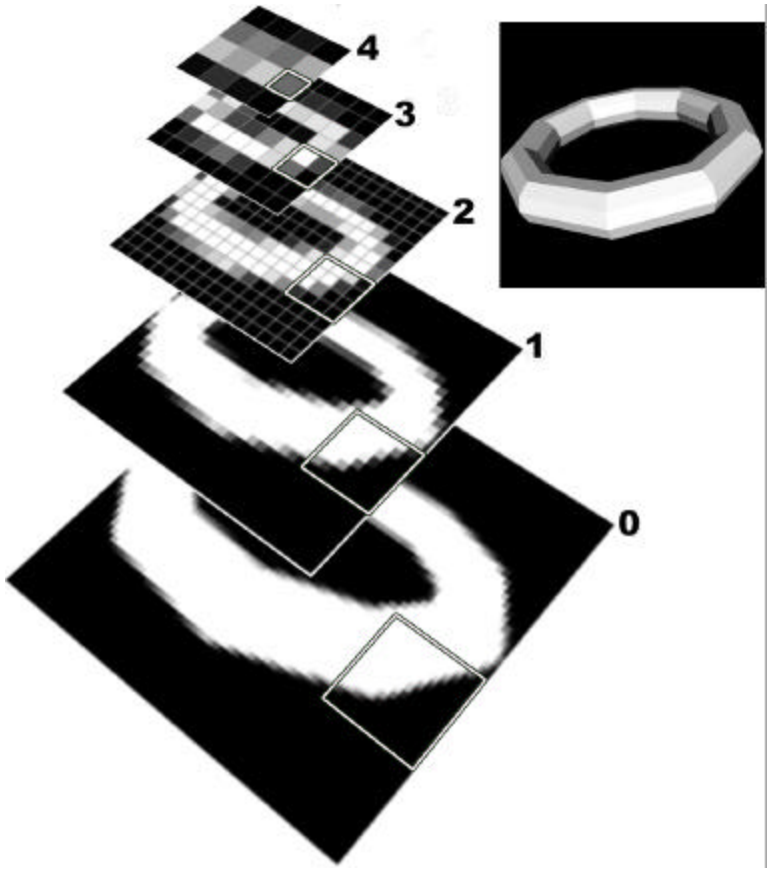
Steps:

1. Find level where pixel cover polygons bounding rectangle

2. Project Bounded Polygon against HOM for overlap test

Decision, For Each Pixel in BR:

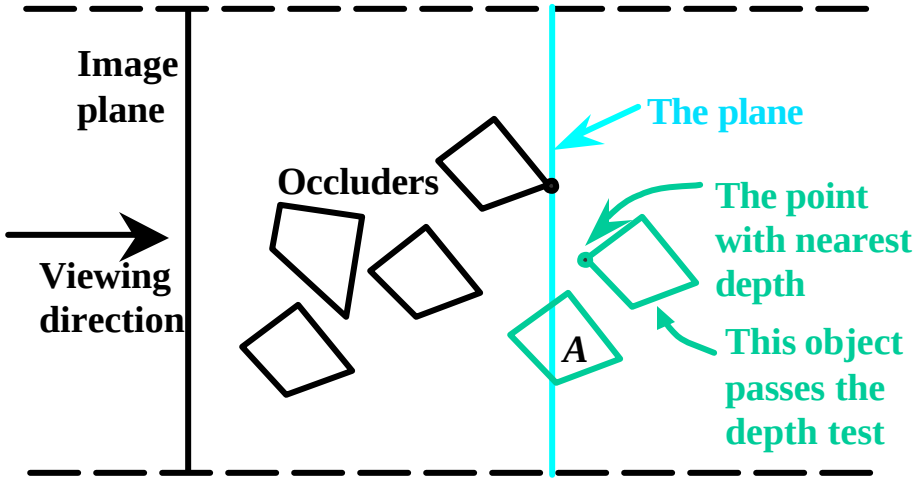
- *If fully covered and opaque then goto next object*
- *Else go down to a fine level of detail until it is decided.*



[image courtesy of Zhangh, PhD defense slides]

H.O.M Depth Test

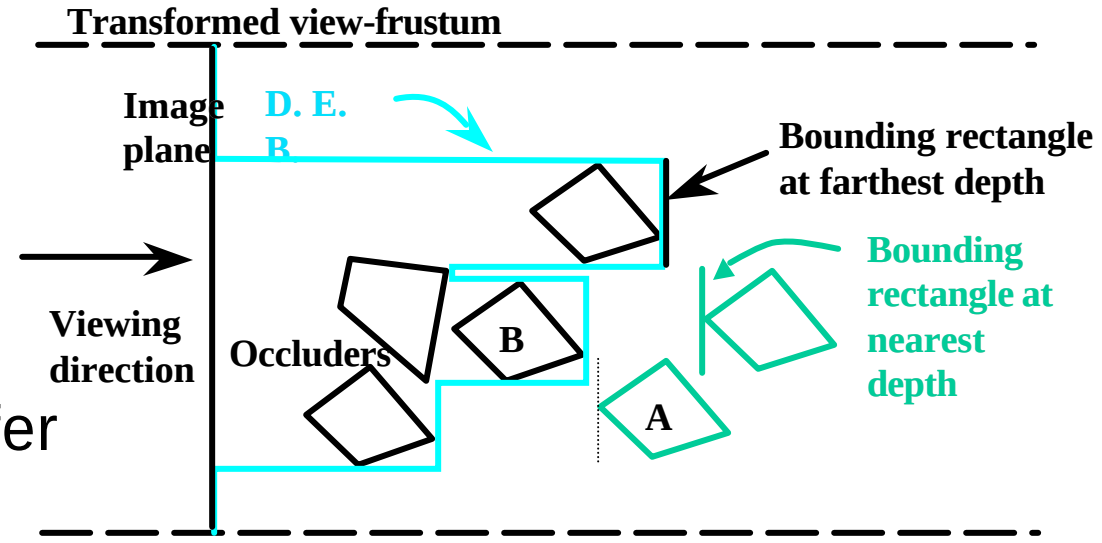
Choose A Depth Test!



Either: a single plane at furthest point of occluders

Or: uniform subdivision of image with separate depth at each partition

Or even: just the Z-buffer content



[image courtesy from slides on occlusion, University College London]

Conclusion

- There is a ton of stuff on the web describing each technique
- Not the most difficult subject
- What techniques you use will depend on your application

References

- Books
  - Tomas Akenine-Moller and Eric Haines, Real-Time Rendering, A.K Peters Ltd., pp. 346, 2002
- Papers
  - Hybrid Scene Structuring with Application to Ray Tracing, Gordon Müller and Dieter W. Fellner
- Web pages
  - http://www.cosy.sbg.ac.at/~held/projects/collision/bvt.html
  - http://www.cs.umd.edu/~brabec/quadtree/
    - This web page gives Java applets to run various spatial data structures

References

- http://www.cs.wpi.edu/~matt/courses/cs563/talks/bsp/bsp.html
- http://www.cse.ohio-state.edu/~hwshen/781/newCulling.ppt
- http://pfportals.cs.virginia.edu/
- http://www.xbox.com/media/games/tomclancysplintercell/sim-tomclancysplintercell-0005.jpg

References

- *Greene, Ned, Michael Kass, and Gavin Miller, "Hierarchical Z-Buffer Visibility", Computer Graphics (SIGGRAPH 93 Proceedings), pp. 231-238, August 1993.*

- *Scott, N., D. Olsen, and E. Gannett, "An Overview of the VISUALIZE fx Graphics Accelerator Hardware", Hewlett-Packard Journal, pp. 28-34, May 1998.* http://www.hp.com/hpj/98may/ma98a4.htm

- *Eric haines, Thomas Moller, Gamasutra Article on Occlusion Culling, Nov. 9, 1999,* http://www.gamasutra.com/features/19991109/moller_haines_01.htm

- *Slides on Occlusion Talk, University College London, http://www.cs.ucl.ac.uk/staff/A.Steed/ book_tmp/CGVE/slides/occlusion.ppt*

- Zhang, H., D. Manocha, T. Hudson, and K.E. Hoff III, "Visibility Culling using Hierarchical Occlusion Maps", *Computer Graphics (SIGGRAPH 97 Proceedings),* pp. 77-88, August 1997. http://www.cs.unc.edu/~zhangh/hom.html

- Zhang, Hansong, *Effective Occlusion Culling for the Interactive Display of Arbitrary Models*, Ph.D. Thesis, Department of Computer Science, University of North Carolina at Chapel Hill, July 1998.

- Real-Time Rendering, Tomas Moller, Eric Haines, A K Peters, 2002