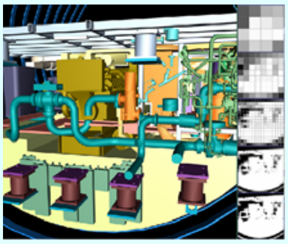
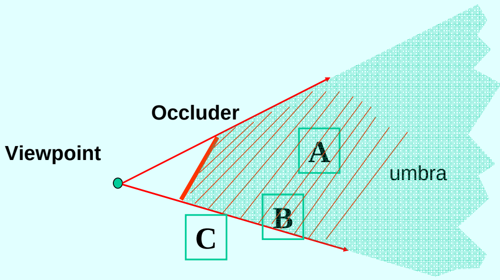
## Image Space Occlusion Culling





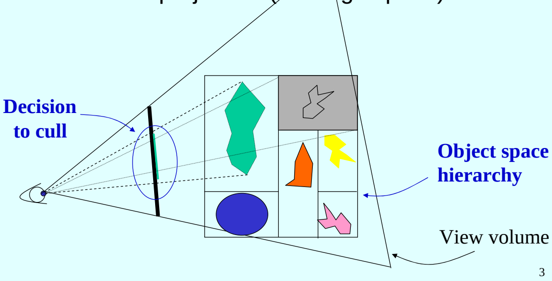
## Hudson et al, SoCG 97






## What Methods are Called Image-Space?

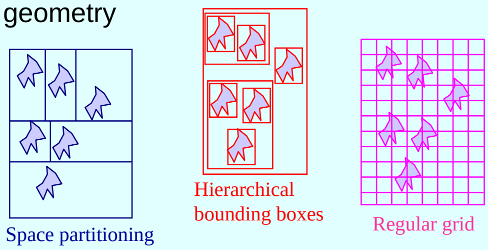
- Those where the decision to cull or render is done after projection (in image space)






## Ingredients of an Image Space Method

- An object space data structure that allows fast queries to the complex geometry

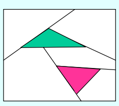





## An image space representation of the occlusion information

- Discrete
  - Z-hierarchy
  - Occlusion map hierarchy
  - Hardware – Occlusion Query

- Continuous
  - BSP tree
  - Image space extends





## General Outline of Image Space Methods

- During the (Top-down, front-to-back) traversal of the scene hierarchy do:
  - compare each node against the view volume.
  - if not culled, test node for occlusion against **occlusion map**.
  - if still not culled, render objects/occluders **augmenting the occlusion map**

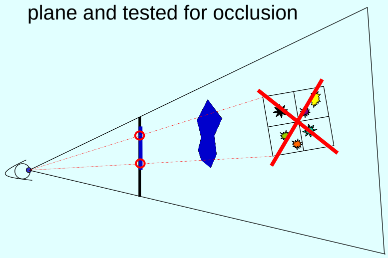
## Occlusion Map

**Set of Occluders** **Occlusion Map**

8

## Testing a Node for Occlusion

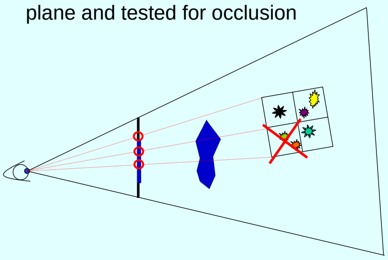
- If the box representing a node is not visible then nothing in it is either
- The faces of the box are projected onto the image plane and tested for occlusion



9

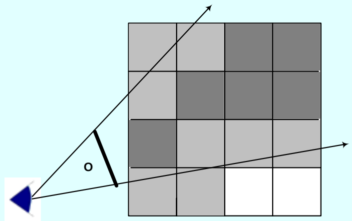
## Testing a Node for Occlusion

- If the box representing a node is not visible then nothing in it is either
- The faces of the box are projected onto the image plane and tested for occlusion



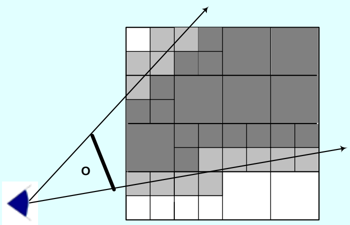
10

## Hierarchical Tests



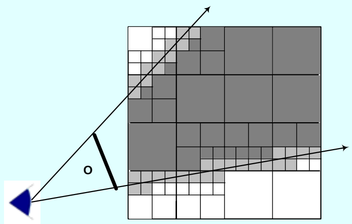
11

## Hierarchical Tests



12

## Hierarchical Tests



13

## Differences of Algorithms

- The most important differences between the various approaches are:
  - the representation of the (augmented) occlusion map and,
  - the method of testing the hierarchy for occlusion

14

## Hierarchical Z-Buffer (HZB)

*(Ned Greene, Michael Kass 93)*

- An extension of the Z-buffer VSD algorithm
- It follows the outline described above.
- Scene is arranged into an octree which is traversed top-to-bottom and front-to-back.
- During rendering an occlusion map is incrementally built.
- Octree nodes are compared against occlusion map.
- The occlusion map is a z-pyramid…

15

## OpenGL Assisted Culling

*(Bartz et al C&G99)*

- Similar in principle to HZB but instead of creating a z-pyramid:
  - set up OpenGL so that it doesn't modify the z-buffer and it writes into the stencil whenever the depth test succeeds
  - render the bounding box of the geometry and check the stencil buffer to see if at all visible
- Requires a lot of hardware access

16

## HP Hardware implementation

- Before rendering an object, scan-convert its bounding box
- Special purpose hardware are used to determine if any of the covered pixels passed the z-test
- If not, the object is occluded

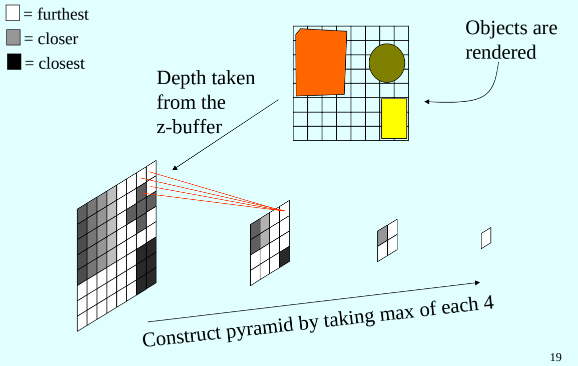
17

## The Z-Pyramid

- The content of the Z-buffer is the finest level in the pyramid
- Coarser levels are created by grouping together four neighbouring pixels and keeping the largest z-value
- The coarsest level is just one value corresponding to overall max z

18

## The Z-Pyramid



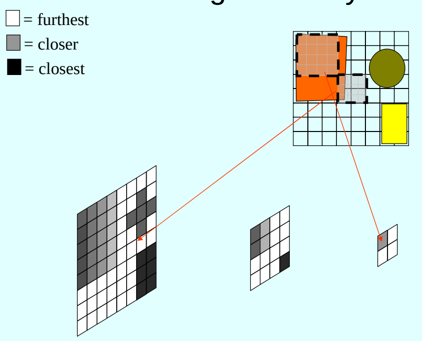

19

## Using the Z-Pyramid

- To determine whether a polygon (e.g. a face of an octree node) is occluded:
  - find the finest-level of the pyramid whose pixel covers the image-space box of the polygon
  - compare their z-values
    - if polygon z > pyramid z, then stop => occluded
    - else descent down the z-pyramid and repeat

20

## Using The Z-Pyramid




21

## Maintaining the Z-Pyramid

- Ideally every time an object is rendered causing a change in the Z-buffer, this change is propagated through the pyramid
- However this is not a practical approach

22

## More Realistic Implementation

- Make use of frame-to-frame coherence:
  - at start of each frame render the nodes that were visible in previous frame
  - read the z-buffer and construct the z-pyramid
  - now traverse the octree using the z-pyramid for occlusion but without updating it

23

## HZB: discussion

- It provides good acceleration in very dense scenes
- Getting the necessary information from the Z-buffer is costly
- A hardware modification was proposed for making it real-time

24

## Hierarchical Occlusion Maps

*(Hansong Zhang et.al 97)*

Similar idea to HZB but:

- they separate the coverage information from the depth information, two data structures
  - hierarchical occlusion maps
  - depth (several proposals for this)

25

## What is Occlusion Map Pyramid?

- A hierarchy of occlusion maps (HOM)
- At the finest level it's just a bit map with
  - 1 where it is transparent and
  - 0 where it is opaque (occluded)
- Higher levels are half the size in each dimension and store gray-scale values
- Records average opacities for blocks of pixels
- Represents occlusion at multiple resolutions

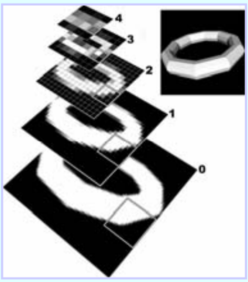
28

## Occlusion Map Pyramid

64 x 64 32 x 32 16 x 16

29

## Occlusion Map Pyramid




31

## Problem Decomposition

•

When is an object occluded by another object?

33

## Representing Occluders

Set of Occluders Occlusion Map

34

## Overlap Tests

- To test if the projection of a polygon is occluded
  - find the finest-level of the pyramid whose pixel covers the image-space box of the polygon
  - if fully covered then continue with depth test
  - else descend down the pyramid until a decision can be made

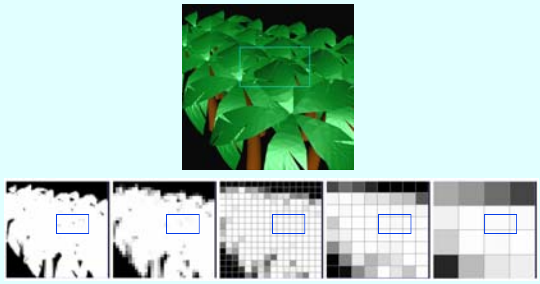
35

## Aggressive Approximate Culling

- A great advantage over the HZB
- Ignoring barely-visible objects
  - Small holes in or among objects
  - To ignore the small holes
    - Low-pass filter suppresses noise — holes "dissolve"
    - Regard "very high" opacity as fully opaque



## Aggressive Approximate culling



0 1 2 3 4



## Occluder selection

- This is a big issue relevant to most occlusion culling algorithms
- Occluder data-base -- selection criterions
  - size, redundancy, rendering complexity
  - Size of bounding boxes (when depth-estimation buffer is used)
- At run time
  - Objects inside the view volume
  - Distance-based selection with a polygon budget



## Metric for Comparing Occluder Quality

Occluder quality: $(-A\ (N * V)) / ||D||^2$

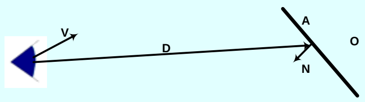
*A* : the occluder's area

N : normal

V : viewing direction

*D* : the distance between the viewpoint and the occluder center

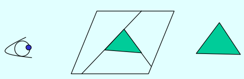
Large polygon have large area-angle.






## BSP Occlusion Culling *(Naylor GI92)*

- Both scene and occlusion information are represented as BSP trees
- Render scene in front-to-back order
- Create 2D BSP tree using the edges of the rendered polygons
- Intersect this with the scene BSP tree to find occluded regions





## NV Occlusion Query (1)

- Extension name: NV_occlusion_query
- Returns pixel count – the # of pixels that pass
- Provides an interface to issue multiple queries at once before asking for the result of any one
- Applications can now overlap the time it takes for the queries to return with other work increasing the parallelism between CPU and GPU

## NV Occlusion Query – How to Use (1)

- (Optional) Disable Depth/Color Buffers
- (Optional) Disable any other irrelevant non-geometric state
- Generate occlusion queries
- Begin $i^{th}$ occlusion query
- Render $i^{th}$ (bounding) geometry
- End occlusion query
- Do other CPU computation while queries are being made
- (Optional) Enable Depth/Color Buffers
- (Optional) Re-enable other state
- Get pixel count of $i^{th}$ query
- If (count > MAX_COUNT) render $i^{th}$ geometry



## NV Occlusion Query – How to Use (2)

- Generate occlusion queries

```
Gluint queries[N];
GLuint pixelCount;
glGenOcclusionQueriesNV(N, queries);
```

- Loop over queries

```
for (i = 0; i < N; i++) {
    glBeginOcclusionQueryNV(queries[i]);
    // render bounding box for ith geometry
    glEndOcclusionQueryNV();
}
```

- Get pixel counts

```
for (i = 0; i < N; i++) {
    glGetOcclusionQueryuivNV(queries[i], GL_PIXEL_COUNT_NV, &pixelCount);
    if (pixelCount > MAX_COUNT)
      // render ith geometry
 }
```