Advanced Computer Graphics
CS 563: *Acceleration Algorithms*

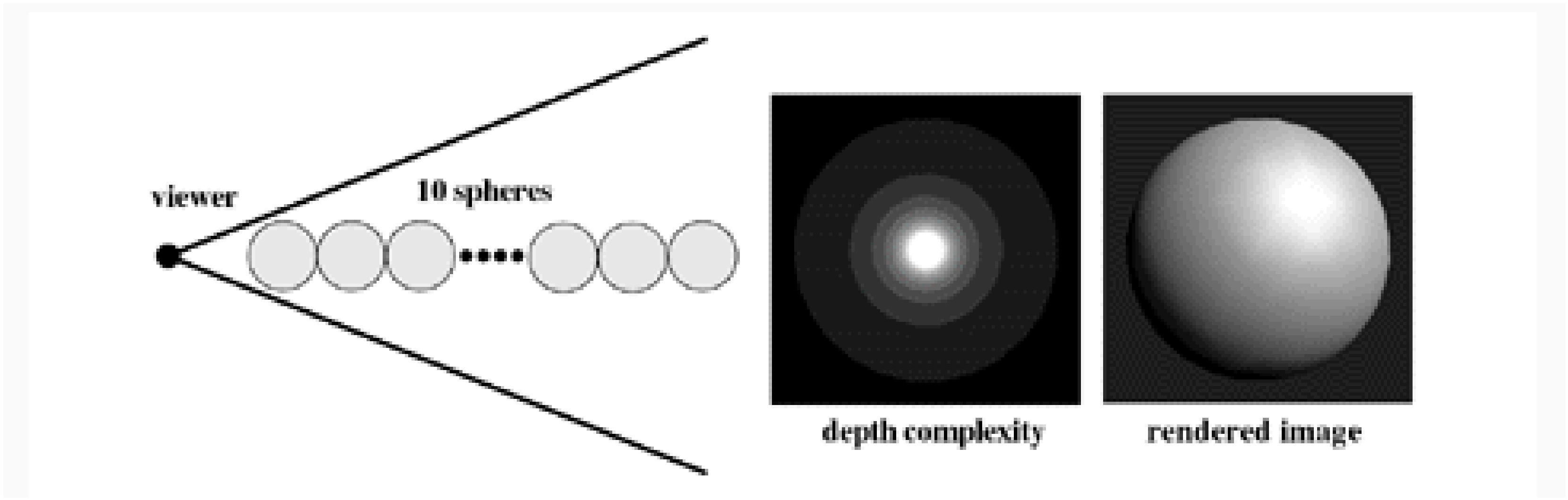
Frederik Clinckemaillie

Computer Science Dept.

Worcester Polytechnic Institute (WPI)

Problem with Z-buffer

- Using Z-buffer can cause pixels to be overwritten several times.
  - High depth complexity



Occlusion Culling

- Attempts to cull away occluded objects
- Removes objects from scene before going through pipeline
- Types:
  - Point-based
    - Visibility calculated from single point
  - Cell-based
    - Visibility calculated for all points in view cell
    - Can be reused for a few frames

Occlusion Culling

```
1: **OcclusionCullingAlgorithm** (G)
2: $O_R$=empty
3: P =empty
4: for each object g in G
5:     if(isOccluded(g,$O_R$))
6:         Skip(g)
7:     else
8:         Render(g)
9:         Add(g,P)
10:        if(LargeEnough(P))
11:            Update($O_r$, P)
12:            P =empty
13:        end
14:    end
15: end
```

P:set of potential occluders
G: Objects in the scene
O_r:occlusion representation

Hardware Occlusion Queries

- Occurs in image space
- Bounding volume polygons are tested against z-buffer
- Count of number of pixels n in which polygons are visible is returned
  - n = 0: polygon is occluded
  - N is small: May be discarded
  - N can determine LOD
- Performance: 100% increase in speed.

Other Hardware Occlusion Techniques

- MeiBner et Al.
  - Uses occlusion queries with hierarchical data structure
  - Nodes are sorted in front-to-back order before occlusion testing
- Klosowski and Silva
  - Developed a constant-frame-rate algorithm
  - Prioritized-layered projection
  - Estimated visible polygons of a scene incrementally
  - Not a conservative algorithm

Other Hardware Occlusion Techniques

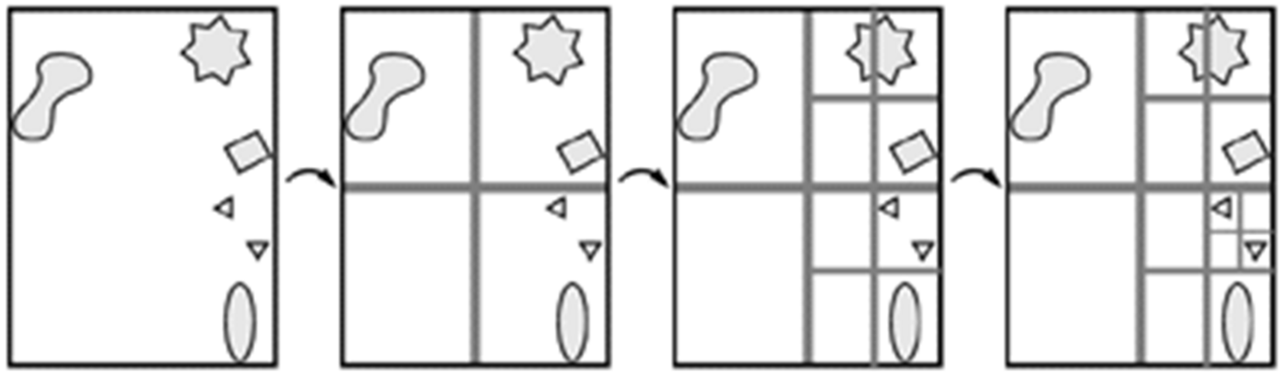
- Sekulic
  - Took advantage of temporal coherence
  - Results of occlusion are checked one frame later
  - Visible objects are rechecked every few frames

Issues with Occlusion Culling

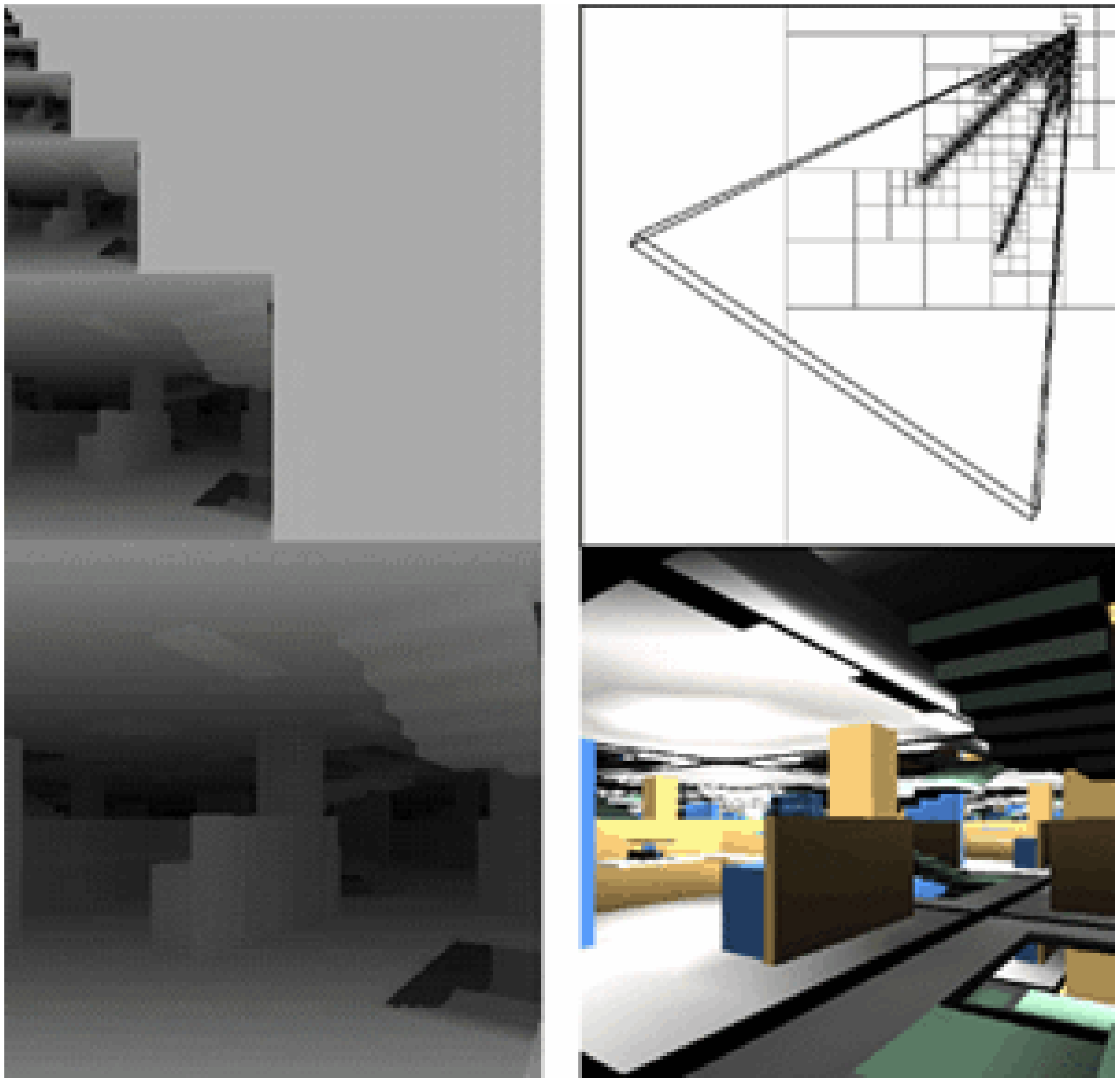
- Using occlusion algorithms can cost time if everything is visible
  - Cutting algorithm back if it is not helping
  - Statistical methods help determine usefulness
- Determining occluders

Hierarchical Z-buffering

- Scene model is held in octree
  - Objects in scenes are divided into 2x2x2 smaller boxes if the number of primitives exceeds a certain number
  - Best for static scenes
- Z-buffer maintained as a Z-pyramid
  - Each z value is the farthest z in 2x2 window of previous level



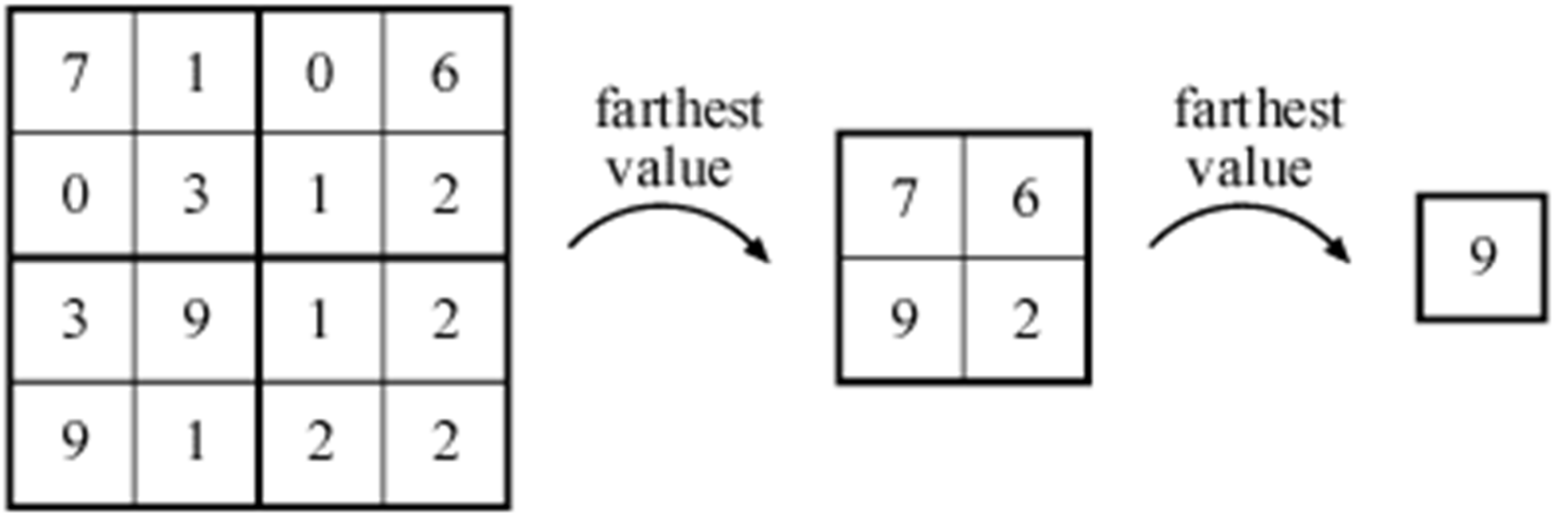
Hierarchical Z-buffering



Hierarchical Z-buffering

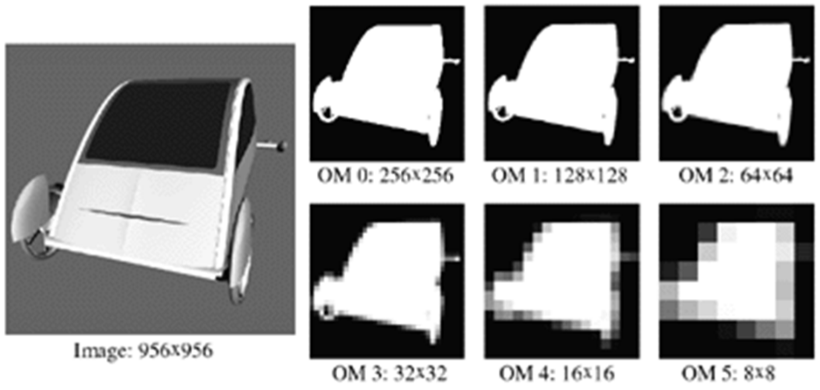
- Octree is traversed in front-to-back order
- Bounding box of the octree is tested against Z-pyramid
  - Begin at coarsest Z-pyramid cell that encloses the box's screen projection
  - If nearest depth(z_{near}) is farther, box is occluded
  - Else, finer level of Z-pyramid is used
- If Octree node is visible, child nodes are examined.

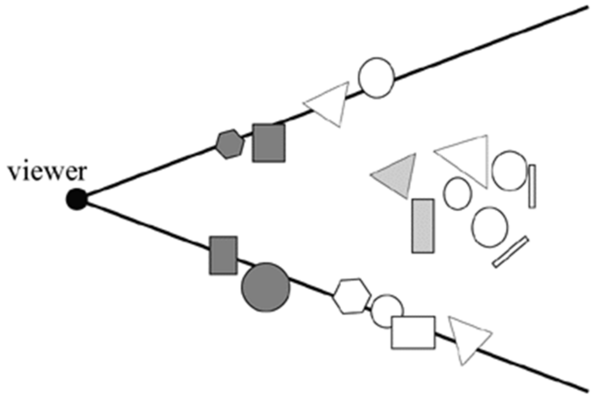
Hierarchical Z-buffering



Other Occlusion Culling Algorithms

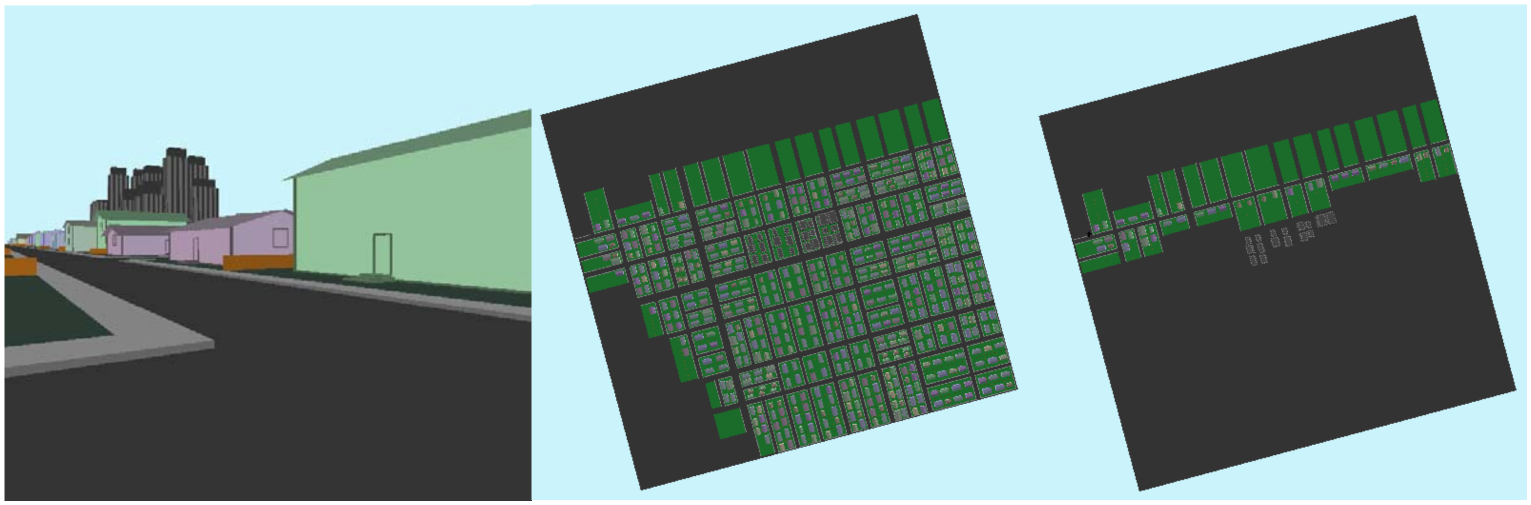
- Hierarchical Occlusion Map
  - Offers approximate occlusion culling
  - Opacity threshold
    - Used at each level of hierarchical depth buffer
    - Objects are culled if too little of them are visible
  - Number of Occluders is limited
    - Creation of HOM can be bottleneck





Other Occlusion Culling Algorithms

- Occlusion Horizons
  - Used to render urban or mountain scenes
  - Scenes are rendered front to back and the horizon drawn is tracked
  - Algorithm is point based



Level of Detail

- Use simpler versions of objects if they make smaller contributions to the image
- LOD algorithms have three parts:
  - Generation: Models of different details are generated
  - Selection: Chooses which model should be used depending on criteria
  - Switching: Changing from one model to another
- Can be used for models, textures, shading and more

Level of Detail

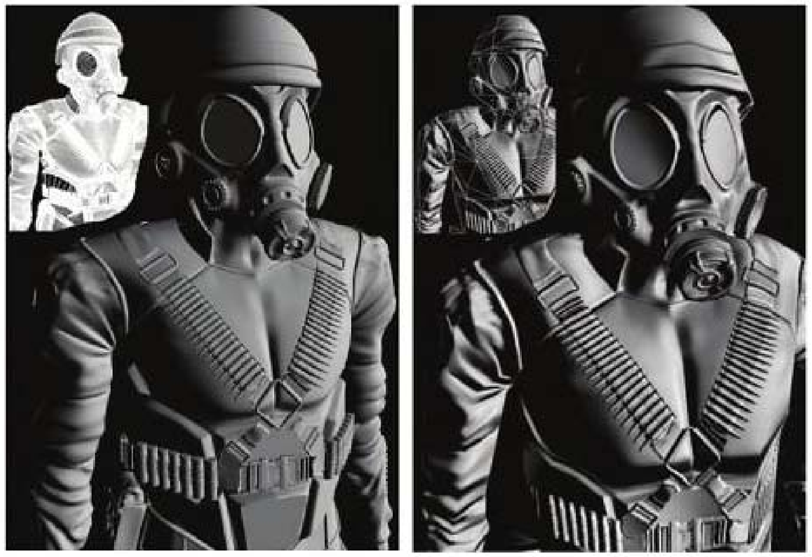
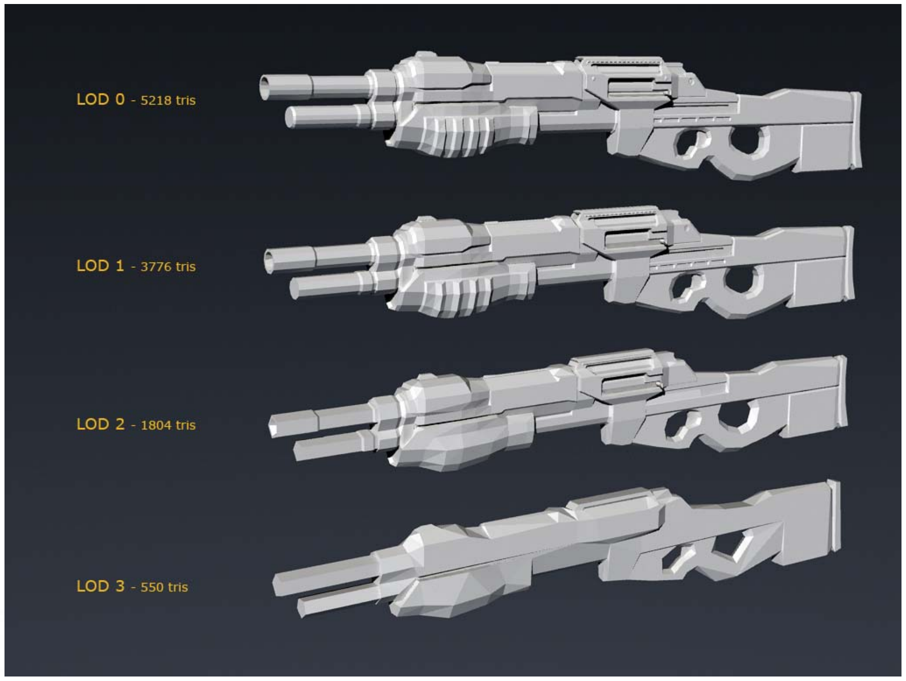


Figure 14.21. On the left, the original model consists of 1.5 million triangles. On the right, the model has 1100 triangles, with surface details stored as heightfield textures and rendered using relief mapping. *(Image courtesy of Natalya Tatarchuk, ATI Research, Inc.)*



LOD Switching

- Discrete Geometry LODs
  - LOD is switched suddenly from one frame to the next
- Blend LODs
  - Two LODs are blended together over time
  - New LOD is faded by increasing alpha value from 0 to 1
  - More expensive than rendering one LOD
  - Faded LODs are drawn last to avoid distant objects drawing over the faded LOD

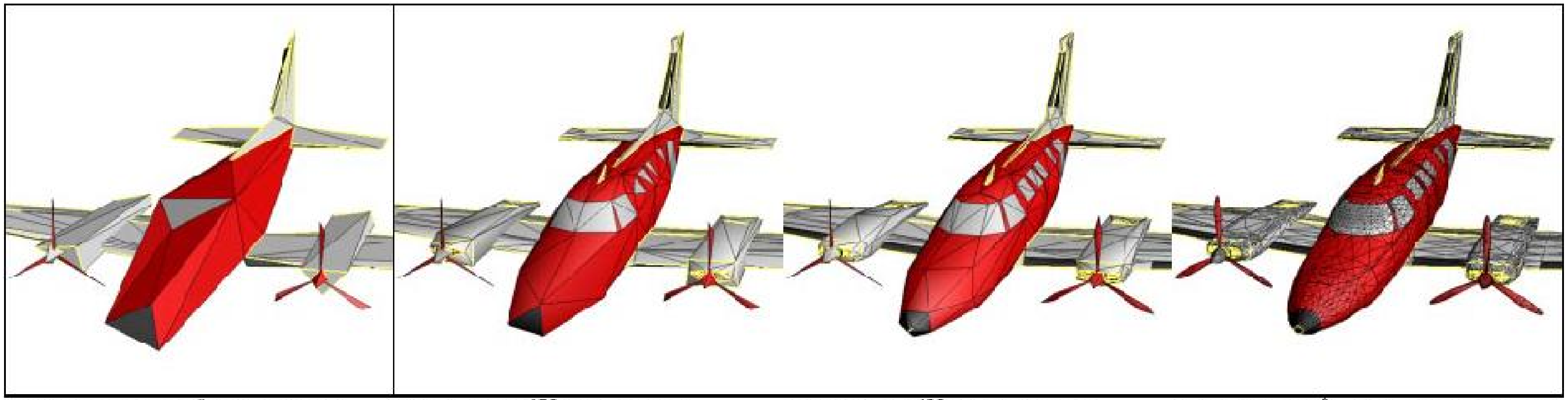
LOD Switching (cont.)

- Alpha LOD
  - Alpha value of object is lowered as distance increases
  - Transparent objects are not sent through pipeline
  - Experience as much more continuous
  - Performance is only felt when object disappears
  - Requires sorting of scene because of transparency

LOD Switching (cont.)

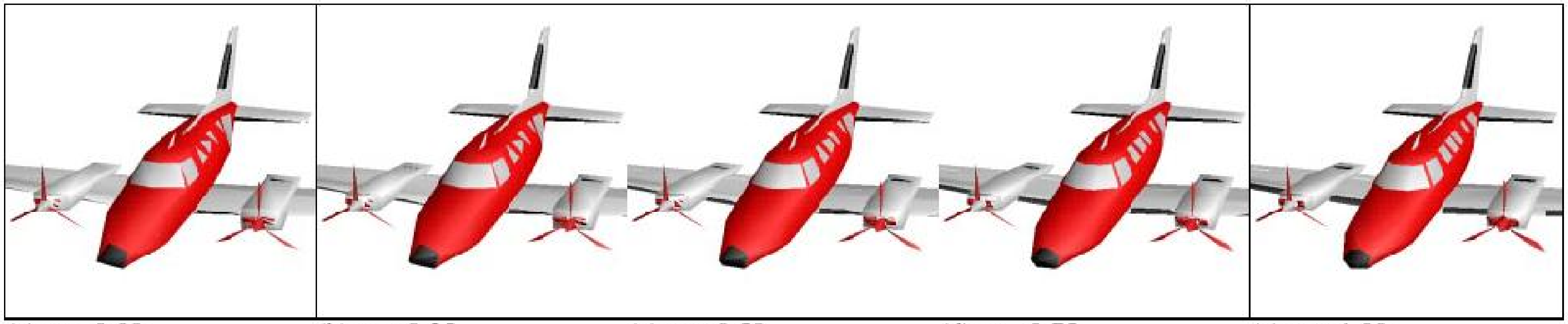
- CLODs and Geomorph LOD
  - Edges can be collapsed as distance increases
  - Process is reversible (vertex split) if deleted vertices are stored
  - Number of polygons can be based on distance (Continuous Level of Detail)
  - Geomorph LODs: a set of discrete models created by simplification with connectivity of vertices maintained
    - Smooth transitions can be done between Geomorph models

CLODs and Geomorph LODs



(a) Base mesh M^0 (150 faces) (b) Mesh M^{175} (500 faces) (c) Mesh M^{425} (1,000 faces) (d) Original $\hat{M} = M^n$ (13,546 faces)

Figure 5: The PM representation of an arbitrary mesh $\hat{M}$ captures a continuous-resolution family of approximating meshes $M^0 \dots M^n = \hat{M}$.

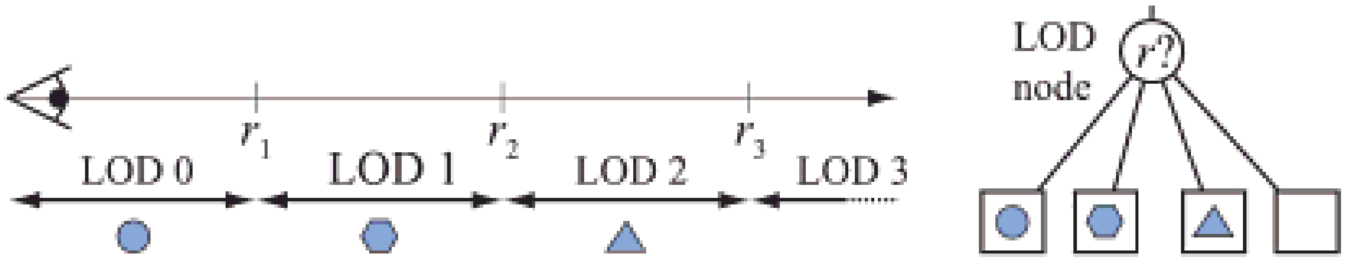


(a) $\alpha = 0.00$ (b) $\alpha = 0.25$ (c) $\alpha = 0.50$ (d) $\alpha = 0.75$ (e) $\alpha = 1.00$

Figure 6: Example of a geomorph $M^G(\alpha)$ defined between $M^G(0) \doteq M^{175}$ (with 500 faces) and $M^G(1) = M^{425}$ (with 1,000 faces).

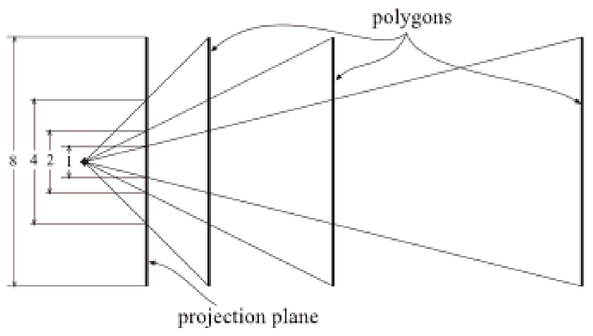
LOD Selection

- Determining which LOD to render and which to blend
- Range-Based:
  - LOD choice based on distance



LOD Selection

- Projected Area-Based
  - Estimates the projected area of the bounding volume
  - Estimating Screen-space coverage:
    - For spheres, estimation of radius is :
      $$p = \frac{nr}{\mathbf{d} \cdot (\mathbf{c} - \mathbf{v})}.$$
      - Distance is the projection of the sphere center onto the view vector (d *(c-v))
      - n: distance from the viewer to the near plan of the view frustum



LOD Selection

- Hysteresis
  - Popping can occur if the metric varies from frame to frame
  - Example: different increasing and decreasing r values

Time-Critical LOD Rendering

- Using LOD to ensure constant frame rates
- Predictive algorithm
  - Selects the LOD based on which objects are visible
- Heuristics:
  - Cost(O,L)
  - Benefit(O,L)
- Maximize $\sum_S \mathrm{Benefit}(O, L)$
- Constraint: $\sum_S \mathrm{Cost}(O, L) \leq \mathrm{TargetFrameTime}.$

Estimating the Heuristics.

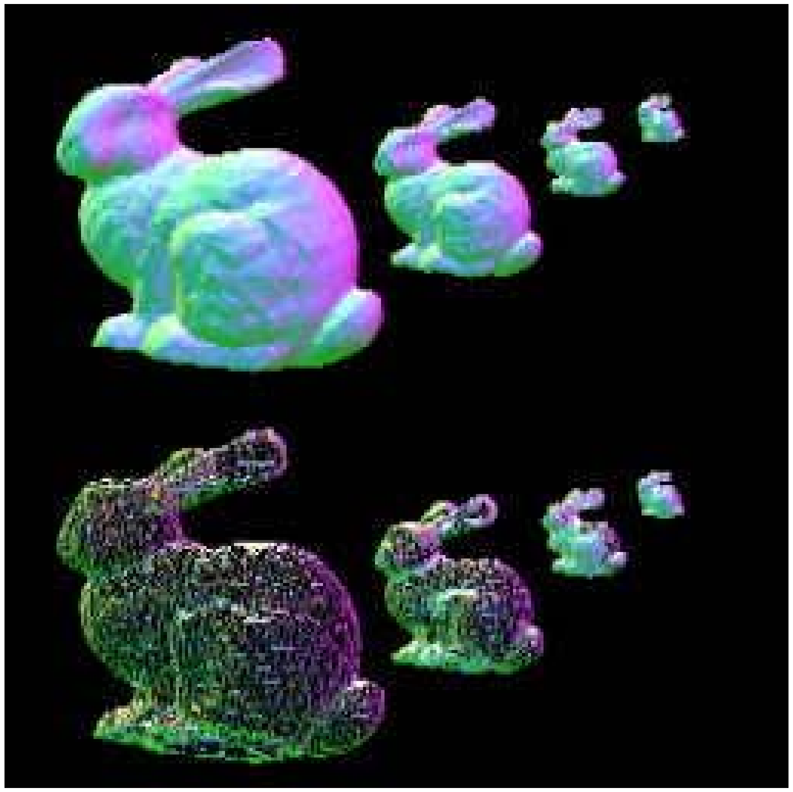
- Do not work in all cases
- Benefit function
  - Methods mentioned earlier
  - In practice: projected area of BV.
- Cost function
  - Time of LODs with different viewing parameters

Time-Critical LOD: Choosing the LODs

- All models have a simplest LOD with no primitives
  - Handles the case of too complex scenes
  - Allows focus on rendering important objects
- Problem is NP-Complete.
- Greedy algorithm to maximize Value (Benefit/Cost)
- Algorithm runs in O(nlog n)
  - N: # of objects in view
  -

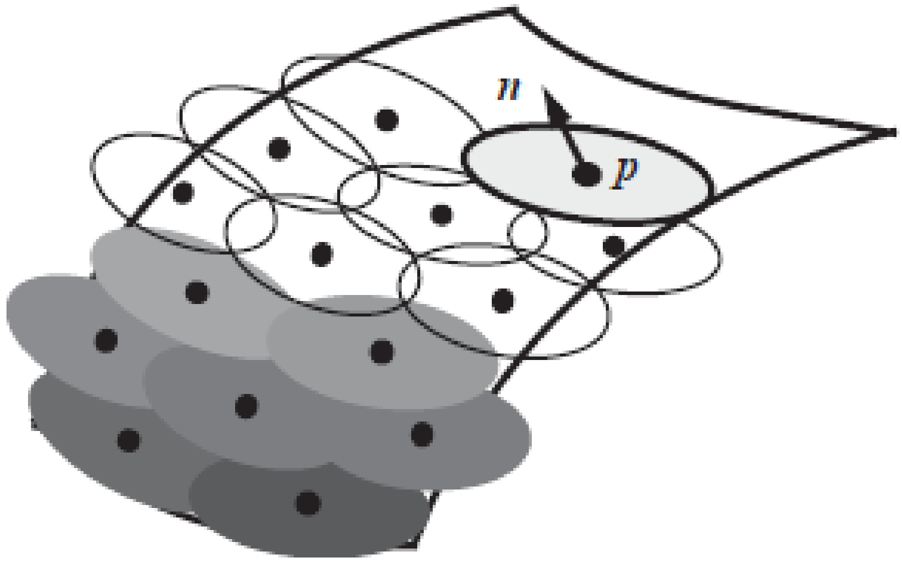
Point Rendering

- Use points as primitive
- Render surfaces as large sets of points
- Gaussian filter pass to fill gaps
  - Radius of Gaussian filter determined by point density



Point Rendering

- Points are rendered as splats
  - Shapes with a set radius
    - May be square, circles, or fuzzy spheres
- Useful when triangles cover less than one pixel in view
- Can import real world objects



Point Rendering

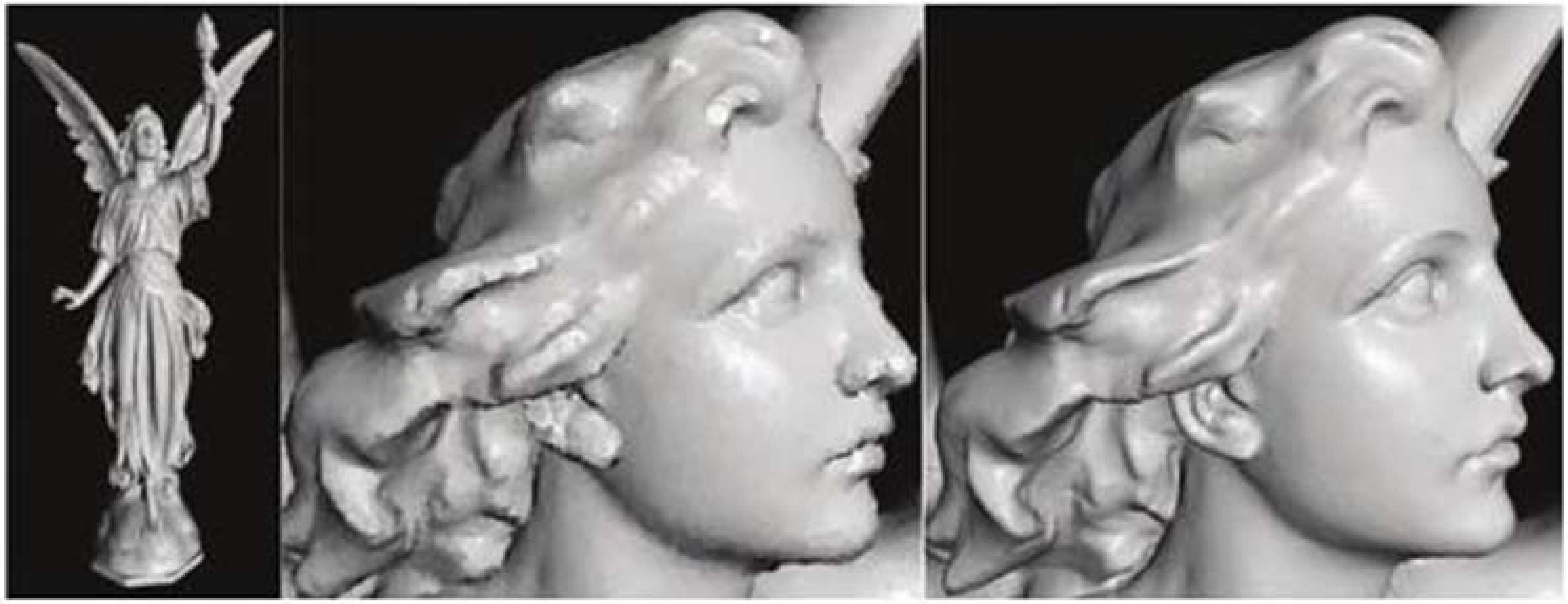


Figure 14.31. These models were rendered with point-based rendering, using circular splats. The left image shows the full model of an angel named Lucy, with 10 million vertices. However, only about 3 million splats were used in the rendering. The middle and right images zoom in on the head. The middle image used about 40,000 splats during rendering. When the viewer stopped moving, the result converged to the image shown to the right, with 600,000 splats. *(Images generated by the QSPlat program by Szymon*

Demo

- http://www.youtube.com/watch?v=VgnNgBwlz6c
- http://www.youtube.com/watch?v=ORswin2M-F4

References

- Akenine-Moller, T et Al. “Real Time Rendering”. AK Peters Ltd 2008, Natick MA.
- Mario Botsch and Leif Kobbelt. 2003. High-Quality Point-Based Rendering on Modern GPUs. In *Proceedings of the 11th Pacific Conference on Computer Graphics and Applications* (PG '03). IEEE Computer Society, Washington, DC, USA, 335-.