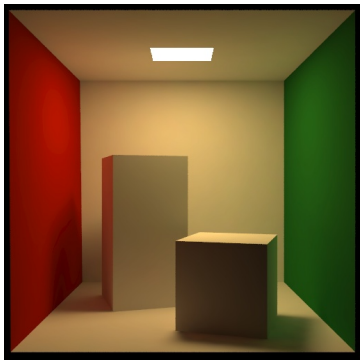
Rendering Really Large Scenes

Thanks to Siddhartha Chaudhuri and Vladlen Koltun

15 polygons

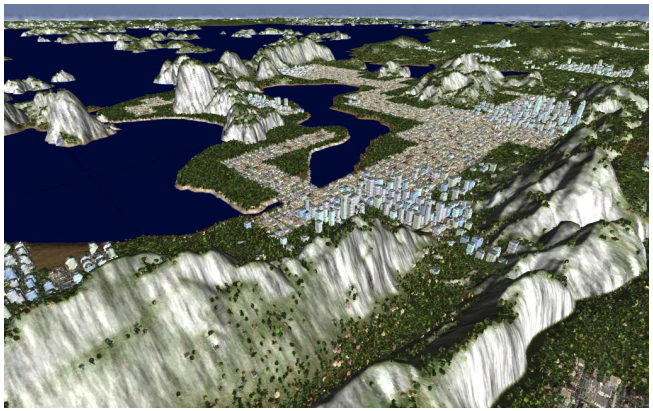


~500,000 polygons

~3,000,000 polygons

Crysis, 2007

~25,000,000,000 polygons



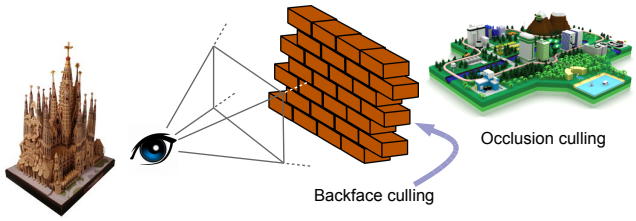
Chaudhuri and Koltun, 2009

Largescale Rendering Cheat Sheet

- Don't render what you can't see
- Don't render what the display can't resolve
- People won't notice small errors, especially in background objects
- **If all else fails, fog is your best friend :)**

Don't render what you can't see

- Rasterizing invisible objects is wasteful
- Detect such objects early and ignore (***cull***) them



Difficulty

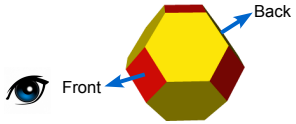
Backface < Frustum <<< Occlusion

Backface Culling

- Drop faces on the far side of object meshes
  - Assume face normals consistently point **inside-out**
  - Back faces have **normals pointing away** from the camera
- OpenGL:

```
glEnable(GL_CULL_FACE);
glCullFace(GL_BACK);
```

  - Why would anyone want `glCullFace(GL_FRONT)`?
  - Uses **vertex winding order** to determine front and back faces, not the vertex normals you pass in!
- When does this scheme for back face culling fail?



Frustum Culling

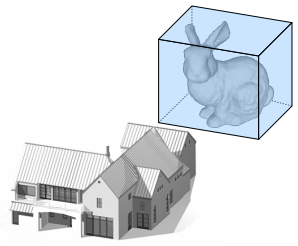
- Test each object against the view frustum
  - Much faster: **test the bounding box** instead
    - If object is visible, no frustum plane can have all 8 corners on invisible side
- Optimization:
  - **Group objects hierarchically**
    - ***Octree*** (or ***quadtree*** for 2.5D scenes)
    - ***Binary Space Partitioning (BSP) tree*** (or a restricted version called a ***kd-tree***)
    - ***Bounding box/sphere hierarchy***
  - Traverse tree top-down and ignore subtrees whose roots fail the bounding box test

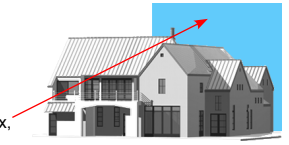
Occlusion Culling

- Whole subject by itself
- General idea: Ignore background objects covered (***occluded***) by foreground objects
  - ***(Hardware) Occlusion queries***
  - ***From-region visibility***
  - ***Portal-based rendering***

Hardware Occlusion Queries

- Part of OpenGL/D3D API
- At any time, pretend to draw a dummy shape (say the bounding box of a complex object) and check if any pixels are affected
- Accelerated by hierarchical z-buffer
- Works for dynamic scenes





Unoccluded pixels of bounding box, so object is potentially visible

From-Region Visibility

- Preprocessing:
  - Break scene up into regions
  - For each region, compute a ***potentially visible set (PVS)*** of objects
- Runtime:
  - Detect the region containing the observer
  - Render the objects in the corresponding PVS
- PVS is usually quite conservative, so further culling is needed

For gaming, this is performed in a preprocessing stage

Case Study: Quake

- Preprocessing:
  - Level map preprocessed into BSP-tree
  - Each leaf node stores potentially visible polygons from that region
- Runtime:
  - Leaf node containing player detected by searching the tree (very fast)
  - PVS of polygons for this node are rendered
  - (BSP-tree is NOT used for back-to-front rendering!)

(painter algorithm)

Portal-Based Rendering

- Suitable for indoor environments
- Divide environment into ***cells***, connected by simple polygonal ***portals*** (doors/windows/...)
- Render:
  - Neighboring cells with visible portals (check if projected polygon is within screen limits)
  - Neighbors-of-neighbors with portals visible through the first set of portals
  - ... and so on
- Further culling possible with frusta through portals

Case Study: Unreal 2

Don't render what the display can't resolve/people won't notice

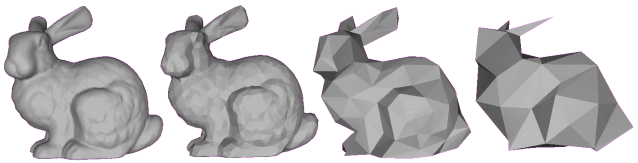
- There're only ~1 million pixels on the average screen
- Why spend precious milliseconds rendering the Taj Mahal in exquisite detail if it's only going to take up 10 pixels on the screen?
- Moral: **Simplify distant objects**

  ... assuming perspective projection
- Moral #2: Since people are usually not too interested in the background, you can **simplify over-aggressively**

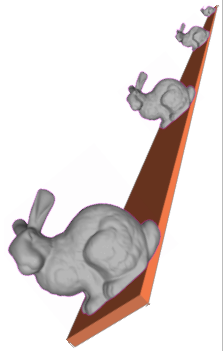
For every object, choose the **simplest possible representation** that will look nearly the same as the

Levels of Detail (LOD)

- **Coarser representations for distant objects**
  - Hierarchy of representations of the same object at different resolutions
- The same idea can also be used for textures (***mipmapping***)



69,451 polys 2,502 polys 251 polys 76 polys



Environment Maps

- Very distant stuff looks the same from anywhere within reasonable limits
- Pre-render distant objects (including the sky) out to a 360° image
- Texture-map it onto a bounding cube at runtime

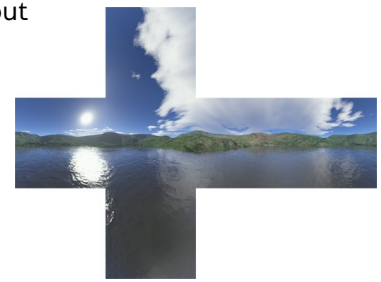
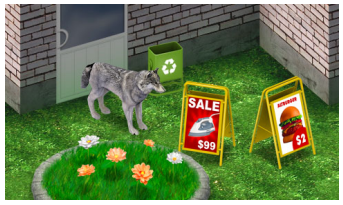


Image-Based Rendering

- Render complex objects to images and texture-map them to simple proxy shapes (***impostors***)
  - Environment mapping is a specific example
- ***Billboards/sprites***: Textured quads always facing the viewer
  - Single image is valid if viewer doesn't move much



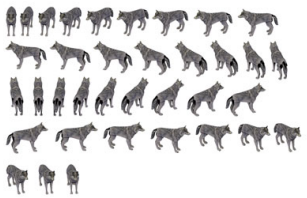
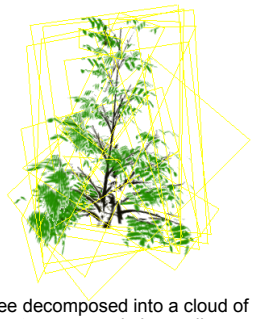


Image-Based Rendering

Image base rendering - use real images in computer-generated scene



Tree decomposed into a cloud of texture-mapped planar slices

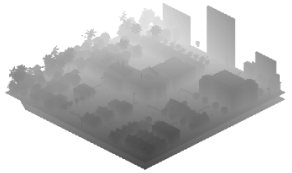
Impostors Original

Décoret, Sillion, Durand and Dorsey 2002

- **We create several (partially overlapping) images of portions of the tree.**
- **Each captures more more less a plane in the tree**
- **During rendering time, we decide which of the image planeS to use (more or more)**
- **Note the use of transparent pixels $\alpha = 0$**

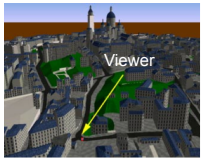
Adding Depth to Images

- Store the **depth map** as well as the color
- Impostor is **heightfield** defined by the depth map
- Fixes parallax errors (impostor is still valid when viewing position changes significantly)
- What are the drawbacks?

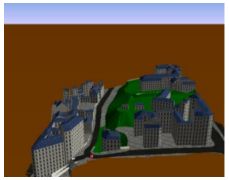


Think about maintain the image as viewer moves

Images + Geometry



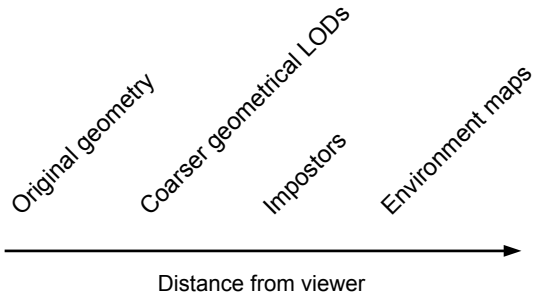
=



+

Foreground Background

LOD spectrum
(not exhaustive or exact!)



Speed Gains (back of the envelope)

- Backface culling: **~2x**
- Frustum culling: **~5x** (varies inversely with FOV)
- Occlusion culling: **can be huge, but no guarantees** at all except in special cases, e.g. portal-based indoor environments
- Levels-of-detail: a uniformly dense scene of radius r takes $\mathbf{O(\log r)}$ time to render (prove!)

Lesson of the day: invest in good LODs!!!