LOD Unresolved Problems

- The LOD algorithms discussed previously do not perform well with large amounts of visible objects
- Consider a large number of tress on a landscape
  - The terrain LOD algorithms work well because of the highly structured terrain mesh, and the basic 2D nature of the problem
  - Per-object simplification performs poorly because it must still render something for every tree
  - Global simplification algorithms perform badly because the "spike" of each individual tree must still appear
- Recall appearance based simplification: use texture maps to capture fine detail while the geometry is simplified

Image-Based Rendering Revisited

- Image-based rendering may be applied to the problem of rendering large databases
  - Primary advantage: Rendering cost depends mostly on the number of pixels in the images, not the number of objects those pixels represent
  - Disadvantage: Have to allow a wide range of viewer motion, which exacerbates typical IBR problems (cracks, stretching…)
- Basic idea: Replace geometry with a few texture mapped polygons
  - Polygons and texture maps can be generated in a pre-process, or as required at run-time
- Environment maps are an instance of this approach

Textured Clusters

(Maciel and Shirley, 1995)

- Provide a range of representations for each object, and a benefit associated with each representation
  - Texture-mapped box is one representation, as is an average color box
  - Benefit may be view dependent – only texture one face
- Use an octree to group objects into clusters
  - Representations for a cluster are a textured and average color box
  - Benefit is derived from benefit of children (maximum of children)
- Deciding what to draw at run time is a bin-packing variation
  - Solve approximately
    - First pass computes cost, benefit and visibility, and sets initial model
    - Second pass starts at root, expands nodes in order of decreasing benefit
  - Not very scalable – looks at every representation of every object on each frame

Textured Clusters Discussion

- Coined the term “impostor”
- Benefit does not add: The benefit of two objects together may be much greater than their sum (eg a man and a gun)
- Data-requirements may be very large
  - The appearance of a cluster can change greatly with viewing position
    - Shading highlights will change rapidly
    - Bigger clusters near the root contain many objects that may show significant disparity
  - Solution: Generate the necessary textures at run-time
- The problem is easier for special cases…

Urban Scenery

(Sillion, Drettakis and Bodelet, 1997)

- Dense cities have two useful properties when viewed from the ground:
  - The view location is highly constrained to be on streets
  - When standing on a street, the only distant views are through the ends of the street
- Place an impostor at the ends of each street
  - Each impostor is a textured 3D mesh
- At run-time, render the *local model* (the street and its buildings) and an impostor (showing everything else)
- Additional space cost is linear in the number of streets
- Rendering is much faster, because a relatively constant, small number of polygons are drawn for each frame

Urban Impostors Details

- Impostors are generated in a pre-processing stage
  - Render the view from the center of each street out each end
  - Take the depth map, and triangulate it while attempting to preserve major discontinuities
  - Use the image as a texture for the triangle mesh
- Remaining problems:
  - Transitions at intersections are problematic
    - At an intersection, the relevant impostors are most incorrect
  - 3D mesh impostors stretch over what should be holes
  - More impostors per street are required for accurate occlusion effects

Multi-Layered Impostors

(Decoret, Schaufler, Sillion and Dorsey, 1999)

- Extend urban impostors to use multiple impostors per street
- Objects visible through the end of the street are grouped into layers
  - Grouping uses the maximum visible distance between two points to identify candidates for each group (nearby objects are grouped)
- At run-time, try to exploit free time by re-rendering the layers from the current viewpoint
  - Produces much better images if the viewer stays still
  - Allows for poorer (and cheaper) basic impostors, because the viewer won't see them for long

Coherent Layers

(Lengyel and Snyder, 1997)

- Hand animators use layers to reduce the number of cells to draw
  - One layer for background, one for middle ground, one for character,…
  - Background layers need to be changed less frequently than foreground, slow moving less frequently than fast moving,…
  - Layers are composited as a final step
- Coherent layers was designed to work with hardware that supports fast compositing and layer warping
- Approach:
  - Break scene into layers by hand
  - At run-time, warp some layers, re-render others
  - Composite the layers into the frame buffer (back to front)

Layered Depth Images

(Shade, Gortler, He and Szeliski, 1998)

- Store more than one color value, with depths, at each pixel
- Several ways to generate layered image
  - Standard ray tracer sampling different rays and combining results
  - Modified ray tracer returning multiple hits per pixel
  - Vision techniques from multiple views
- Rendering uses a splatting operation to push pixels onto the image plane
  - Render pixels in a specific order (McMillan and Bishop, 1995)
  - Project depth forward then back to determine the splat size
- Restricted viewing range for each image
  - Must combine several images to cover a large area (not done in paper)

Image Caching (1)

(Schaufler and Sturzlinger, 1996)

- Hierarchically subdivide world into boxes (kd-tree)
- Replace boxes with a textured rectangles at run-time
  - Given current view, render the box contents onto the rectangle
  - If not a leaf, render child boxes and use impostors to render parent
- Textures are warped in subsequent frames
  - Estimate error in warped image by considering maximum disparity between points in the box
  - Efficiency depends on the viewer rate of motion
- Hardware issues:
  - Assumes hardware clipping planes – not unusual
  - Reading frame buffer is not typically fast (some machines put the frame buffer in main memory – faster)
  - Cannot be multi-threaded (everyone needs the frame buffer)

Image Caching (2)

(Shade, Lischinski, Salesin, DeRose, Snyder, 1996)

- Same idea as previous paper, but superior implementation
- Build BSP hierarchy (actually a kd-tree in this paper)
  - Look for cuts that produce balanced trees
  - Look for cuts that produce near-square boxes
  - Look for cuts that don't split objects (prevents cracking)
  - Grow each box by 10%-20% to avoid cracks
- Use a predictive approach to decide whether it is worth producing a texture map
  - Viewer's motion is assumed bounded (not unreasonable)
  - Uses an off-line estimation of the costs of rendering the geometry and creating the texture maps

Open Problems in LOD

- Dynamic LOD! Design an LOD algorithm that is efficient for scenes that change
  - Some of the terrain rendering algorithms can handle changing terrain, but not very efficiently
  - None of the other algorithms manage deforming objects
  - Image based approaches cannot handle moving objects in the background
- More work on appearance preserving simplification
  - Current methods cannot cluster objects