Level of Detail:
A Brief Overview

Daniel G. Aliaga

Introduction

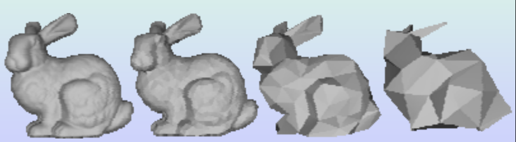
- Level of detail (LOD) is an important tool for maintaining interactivity
  - Focuses on the fidelity / performance tradeoff
  - Not the only tool! Complementary with:
    - Parallel rendering
    - Occlusion culling
    - Image-based rendering [etc]

Level of Detail:
The Basic Idea

- The problem:
  - Geometric datasets can be too complex to render at interactive rates
- One solution:
  - Simplify the polygonal geometry of small or distant objects
  - Known as *Level of Detail* or *LOD*
    - A.k.a. polygonal simplification, geometric simplification, mesh reduction, decimation, multiresolution modeling, …

Level of Detail:
Traditional LOD In A Nutshell

- Create *levels of detail* (*LODs*) of objects:

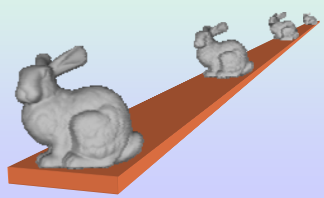


69,451 polys | 2,502 polys | 251 polys | 76 polys

Courtesy Stanford 3D Scanning Repository

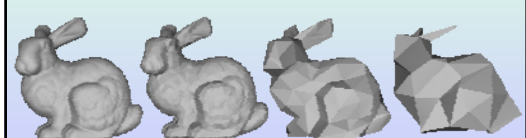
Level of Detail:
Traditional LOD In A Nutshell

- Distant objects use coarser LODs:



Level of Detail:
The Big Questions

- *How to represent and generate simpler versions of a complex model?*

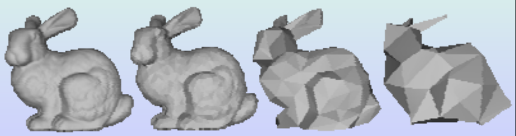


69,451 polys | 2,502 polys | 251 polys | 76 polys

Courtesy Stanford 3D Scanning Repository

Level of Detail:
The Big Questions

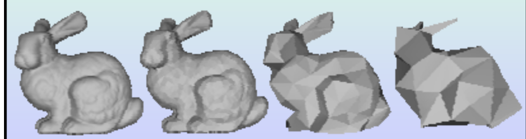
- *How to evaluate the fidelity of the simplified models?*



Courtesy Stanford 3D Scanning Repository

Level of Detail:
The Big Questions

- *When to use which LOD of an object?*



Courtesy Stanford 3D Scanning Repository

Some Background

- History of LOD techniques
  - Early history: Clark (1976), flight simulators
  - Handmade LODs → automatic LODs
  - LOD run-time management: reactive → predictive (Funkhouser)
- LOD frameworks
  - Discrete (1976)
  - Continuous (1996)
  - View-dependent (1997)

Traditional Approach:
Discrete Level of Detail

- Traditional LOD in a nutshell:
  - Create LODs for each object separately in a preprocess
  - At run-time, pick each object's LOD according to the object's distance (or similar criterion)
- Since LODs are created offline at fixed resolutions, we call this *discrete LOD*

Discrete LOD:
Advantages

- Simplest programming model; decouples simplification and rendering
  - LOD creation need not address real-time rendering constraints
  - Run-time rendering need only pick LODs

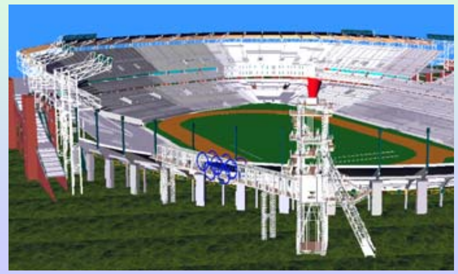
Discrete LOD:
Advantages

- Fits modern graphics hardware well
  - Easy to compile each LOD into triangle strips, display lists, vertex arrays, …
  - These render *much* faster than unorganized triangles on today's hardware (3-5 x)

Discrete LOD:
Disadvantages

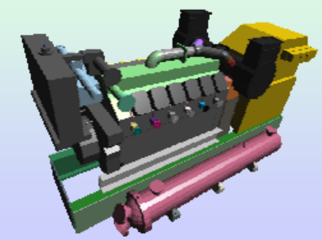
- So why use anything but discrete LOD?
- Answer: sometimes discrete LOD not suited for *drastic simplification*
- Some problem cases:
  - Terrain flyovers
  - Volumetric isosurfaces
  - Super-detailed range scans
  - Massive CAD models

Drastic Simplification:
The Problem With Large Objects



Courtesy IBM and ACOG

Drastic Simplification:
The Problem With Small Objects



Courtesy Electric Boat

Drastic Simplification

- For drastic simplification:
  - Large objects must be subdivided
  - Small objects must be combined
- Difficult or impossible with discrete LOD
- *So what can we do?*

Continuous Level of Detail

- A departure from the traditional discrete approach:
  - Discrete LOD: create individual levels of detail in a preprocess
  - Continuous LOD: create data structure from which a desired level of detail can be extracted *at run time*.

Continuous LOD:
Advantages

- Better granularity → better fidelity
  - LOD is specified exactly, not chosen from a few pre-created options
  - Thus objects use no more polygons than necessary, which frees up polygons for other objects
  - Net result: better resource utilization, leading to better overall fidelity/polygon

Continuous LOD:
Advantages

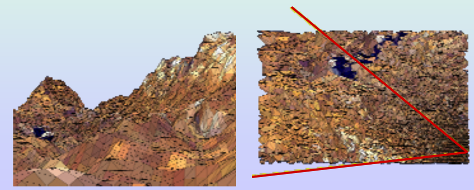
- Better granularity → smoother transitions
  - Switching between traditional LODs can introduce visual "popping" effect
  - Continuous LOD can adjust detail gradually and incrementally, reducing visual pops
    - Can even *geomorph* the fine-grained simplification operations over several frames to eliminate pops [Hoppe 96, 98]

Continuous LOD:
Advantages

- Supports progressive transmission
  - *Progressive Meshes [Hoppe 97]*
  - *Progressive Forest Split Compression [Taubin 98]*
- Leads to *view-dependent LOD*
  - Use current view parameters to select best representation *for the current view*
  - Single objects may thus span several levels of detail

View-Dependent LOD:
Examples

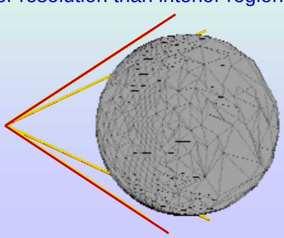
- Show nearby portions of object at higher resolution than distant portions



View from eyepoint   Birds-eye view

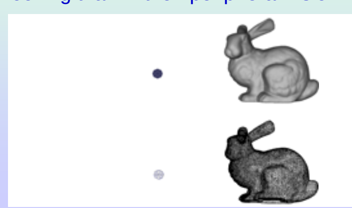
View-Dependent LOD:
Examples

- Show silhouette regions of object at higher resolution than interior regions



View-Dependent LOD:
Examples

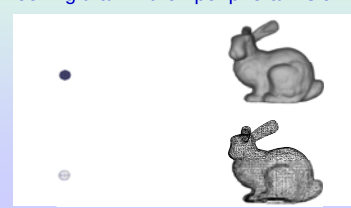
- Show more detail where the user is looking than in their peripheral vision:



34,321 triangles

View-Dependent LOD:
Examples

- Show more detail where the user is looking than in their peripheral vision:



11,726 triangles

View-Dependent LOD: Advantages

- Even better granularity
  - Allocates polygons where they are most needed, within as well as among objects
  - Enables even better overall fidelity
- Enables drastic simplification of very large objects
  - Example: stadium model
  - Example: terrain flyover

Summary: LOD Frameworks

- Discrete LOD
  - Generate a handful of LODs for each object
- Continuous LOD (CLOD)
  - Generate data structure for each object from which a spectrum of detail can be extracted
- View-dependent LOD
  - Generate data structure from which an LOD specialized to the current view parameters can be generated on the fly.
  - One object may span multiple levels of detail
- (Hierarchical LOD)
  - Aggregate objects into assemblies with their own LODs

Choosing LODs: LOD Run-Time Management

- Fundamental LOD issue: where in the scene to allocate detail?
  - For discrete LOD this equates to choosing which LOD will represent each object
  - Run every frame on every object; keep it fast

Choosing LODs

- *Describe a simple method for the system to choose LODs*
  - Assign each LOD a range of distances
  - Calculate distance from viewer to object
  - Use corresponding LOD

Choosing LODs

- *What's wrong with this simple approach?*
  - Visual "pop" when switching LODs can be disconcerting
  - Doesn't maintain constant frame rate; lots of objects still means slow frame times
  - Requires someone to assign switching distances by hand
  - Correct switching distance may vary with field of view, resolution, etc.
- *What can we do about each of these?*

Choosing LODs Maintaining constant frame rate

- One solution: scale LOD switching distances by a "bias"
  - Implement a feedback mechanism:
    - If last frame took too long, decrease bias
    - If last frame took too little time, increase bias
  - Dangers:
    - Oscillation caused by overly aggressive feedback
    - Sudden change in rendering load can still cause overly long frame times

Choosing LODs: Maintaining constant frame rate

- A better (but harder) solution: predictive LOD selection
- For each LOD estimate:
  - *Cost* (rendering time)
  - *Benefit* (importance to the image)

Choosing LODs: Maintaining constant frame rate

- A better (but harder) solution: predictive LOD selection
- For each LOD estimate:
  - *Cost* (rendering time)
    - # of polygons
    - How large on screen
    - Vertex processing load (e.g., lighting) OR
    - Fragment processing load (e.g., texturing)
  - *Benefit* (importance to the image)

Choosing LODs: Maintaining constant frame rate

- A better (but harder) solution: predictive LOD selection
- For each LOD estimate:
  - *Cost* (rendering time)
  - *Benefit* (importance to the image)
    - Size: larger objects contribute more to image
    - Accuracy: no of verts/polys, shading model, etc.
    - Priority: account for inherent importance
    - Eccentricity: peripheral objects harder to see
    - Velocity: fast-moving objects harder to see
    - Hysteresis: avoid flicker; use previous frame state

Choosing LODs:

- Given a fixed time budget, select LODs to maximize benefit within a cost constraint
  - Variation of the knapsack problem
  - *What do you think the complexity is?*
    - A: NP-Complete (like the 0-1 knapsack problem)
  - In practice, use a greedy algorithm
    - Sort objects by benefit/cost ratio, pick in sorted order until budget is exceeded
    - Guaranteed to achieve at least 50% optimal sol'n
    - Time: O(n log n)
    - Can use incremental algorithm to exploit coherence