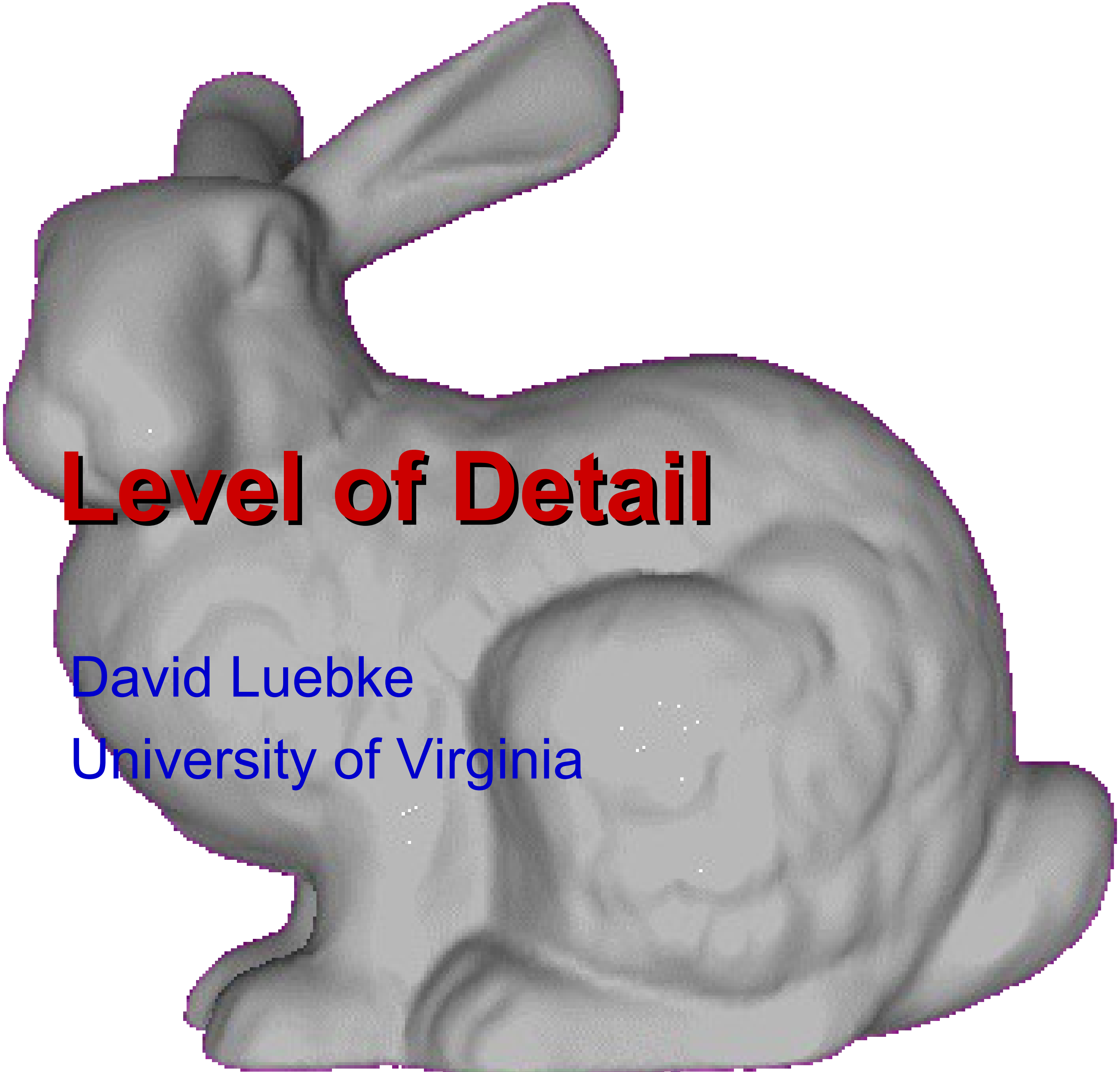
Notice

- Next Tuesday: Final exam
- Next Thursday: Guest speaker
  - NHN platform team director
    - 권택순
  - Will talk about the game development process in NHN
  - 자대생 출석체크 할 것임 .
  - 한글 수업

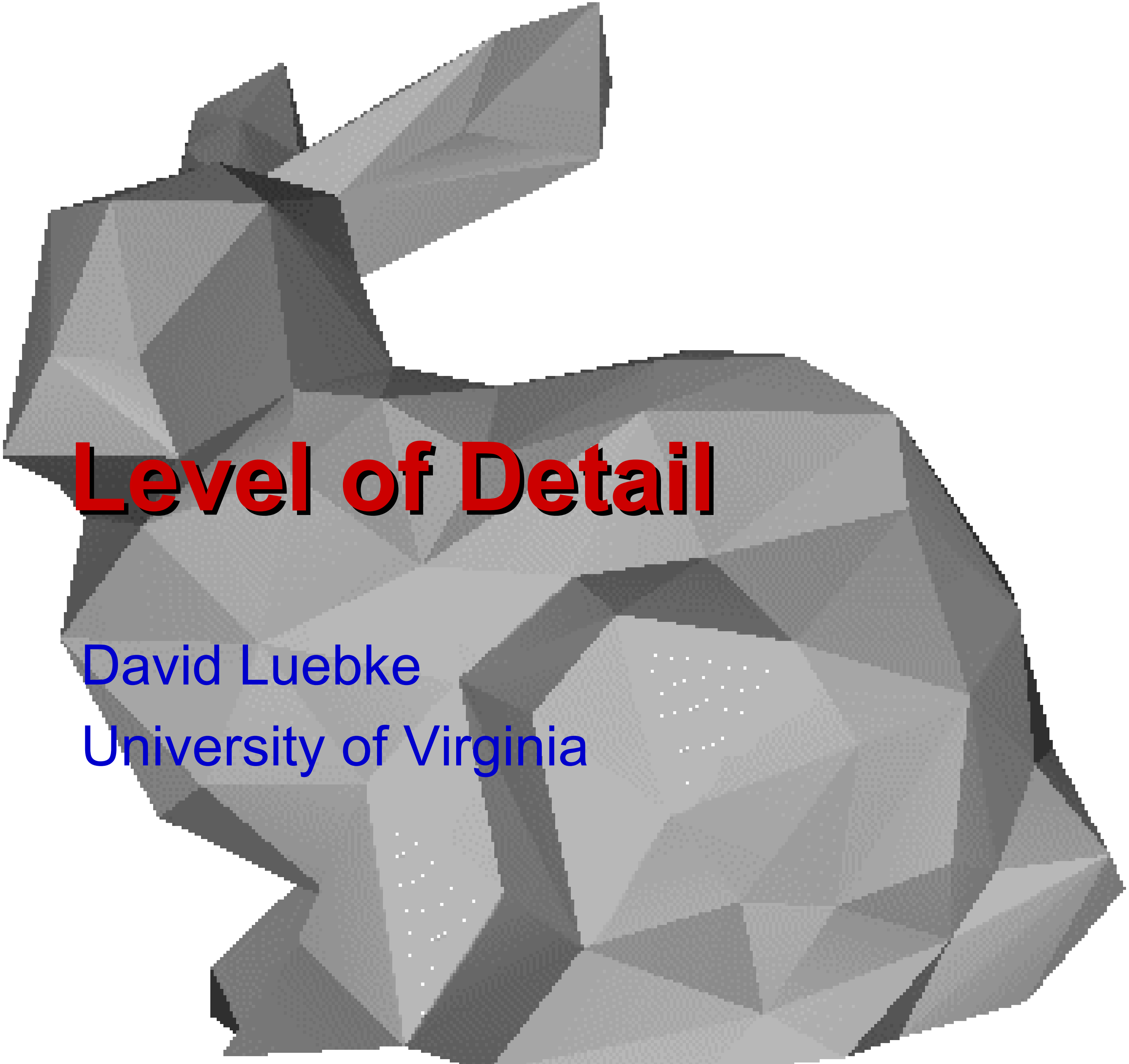


Level of Detail

David Luebke
University of Virginia

Level of Detail

David Luebke
University of Virginia



Level of Detail

David Luebke

University of Virginia

Level of Detail

David Luebke
University of Virginia

Introduction

- Level of detail (LOD) is an important tool for maintaining interactivity
  - Focuses on the fidelity / performance tradeoff
  - Not the only tool! Complementary with:
    - 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. mesh reduction, multiresolution modeling, …

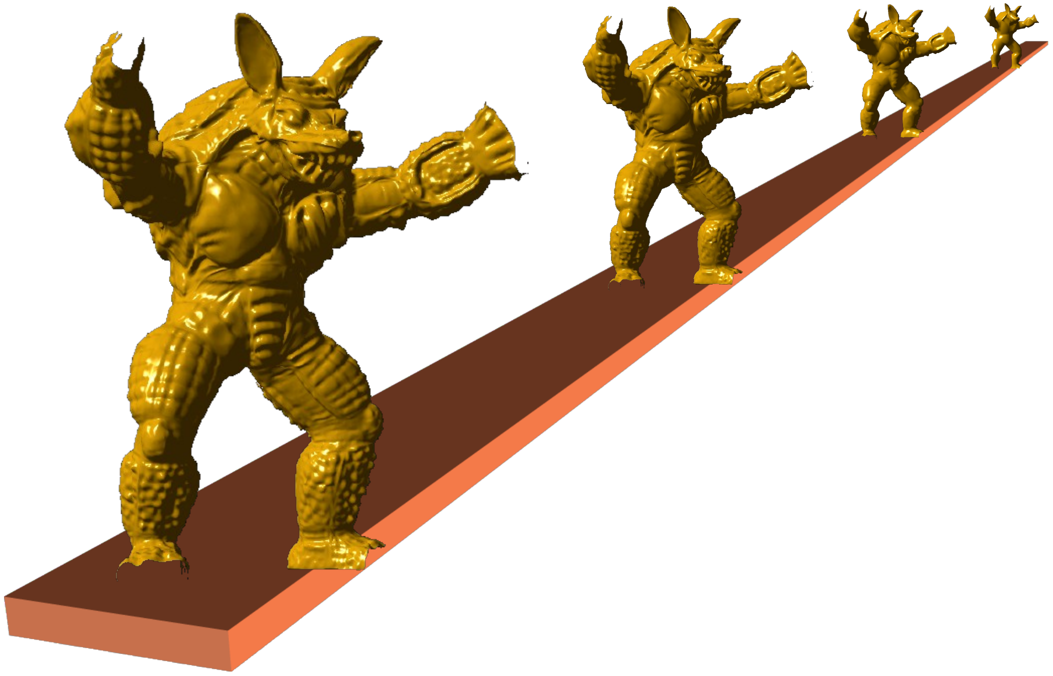
Level of Detail

- A recurring theme in computer graphics: trade fidelity for performance
  - Reduce *level of detail* of distant, small, or unimportant objects

249,924 polys   62,480 polys   7,809 polys   975 polys

Level of Detail

- A recurring theme in computer graphics: trade fidelity for performance
  - Reduce *level of detail* of distant, small, or unimportant objects



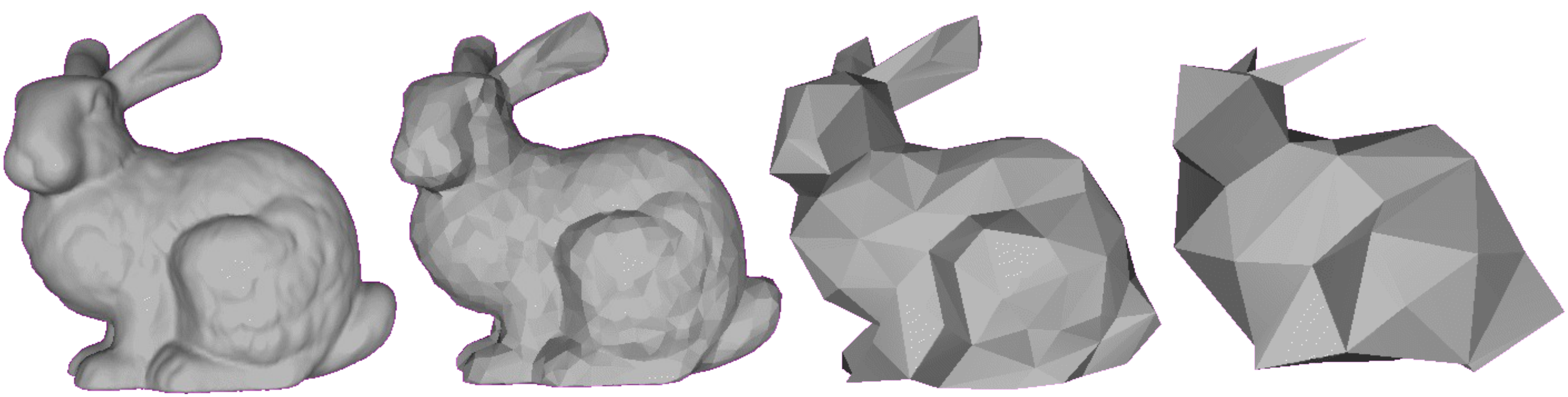
Level of Detail:
Motivation

- Big models!
  - St. Matthew: 372 million polygons
  - David: 1 billion polygons

Courtesy Digital Michelangelo Project

Level of Detail:
Traditional LOD In A Nutshell

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

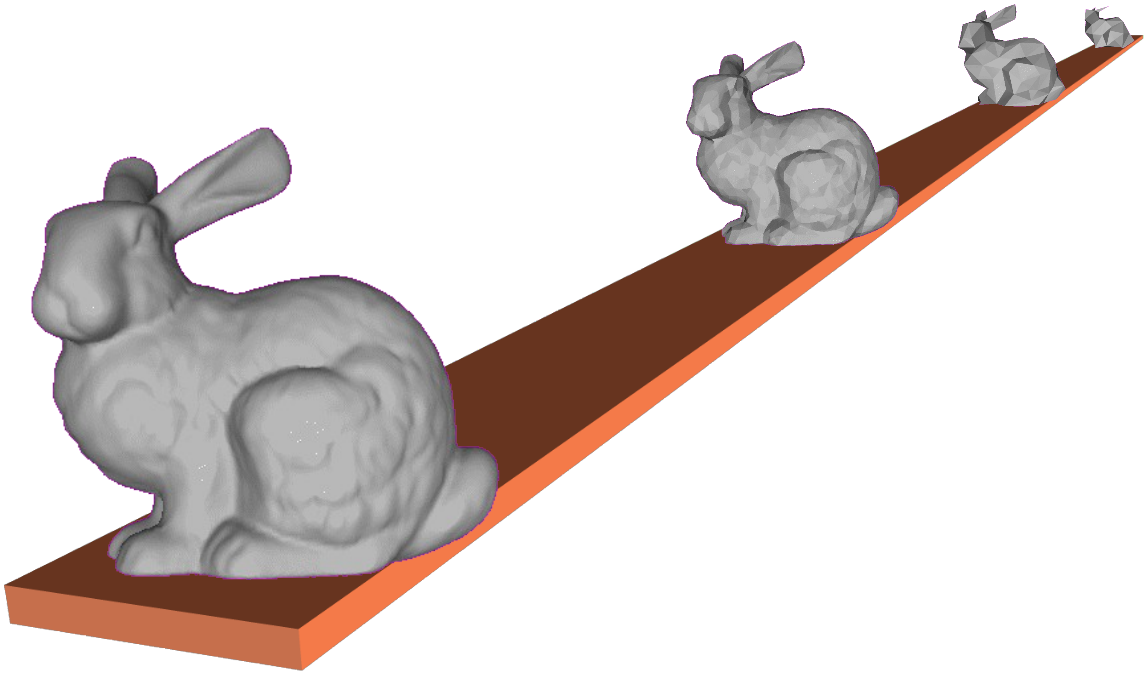


69,451 polys   2,502 polys   251 polys   76 polys

Courtesy Stanford 3D Scanning Repository

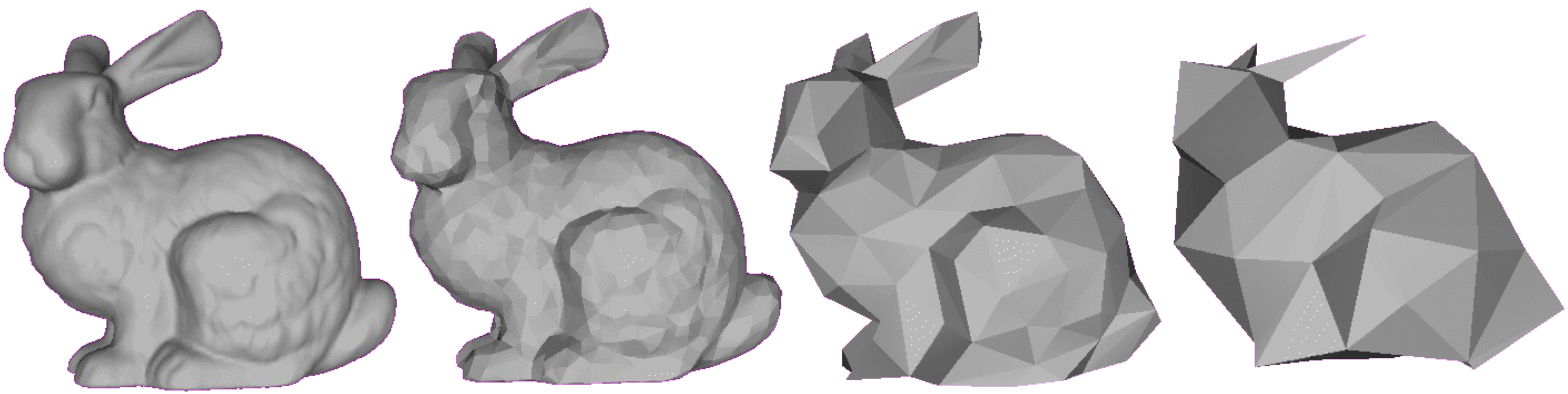
Level of Detail:
Traditional LOD In A Nutshell

- Distant objects use coarser LODs:



Level of Detail:
The Big Questions

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

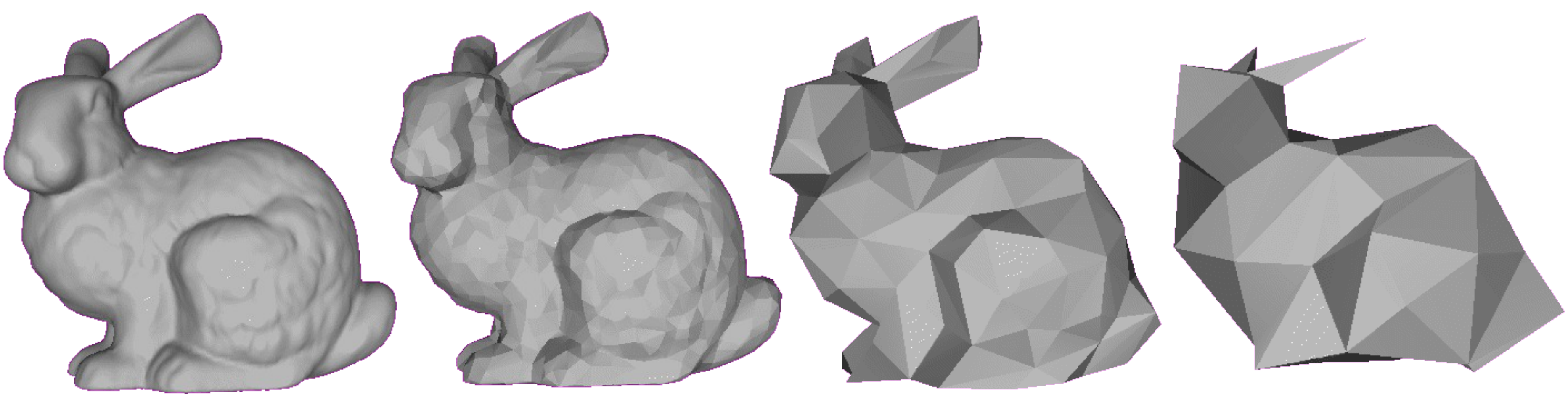


69,451 polys 2,502 polys 251 polys 76 polys

Courtesy Stanford 3D Scanning Repository

Level of Detail:
The Big Questions

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



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

Courtesy Stanford 3D Scanning Repository

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

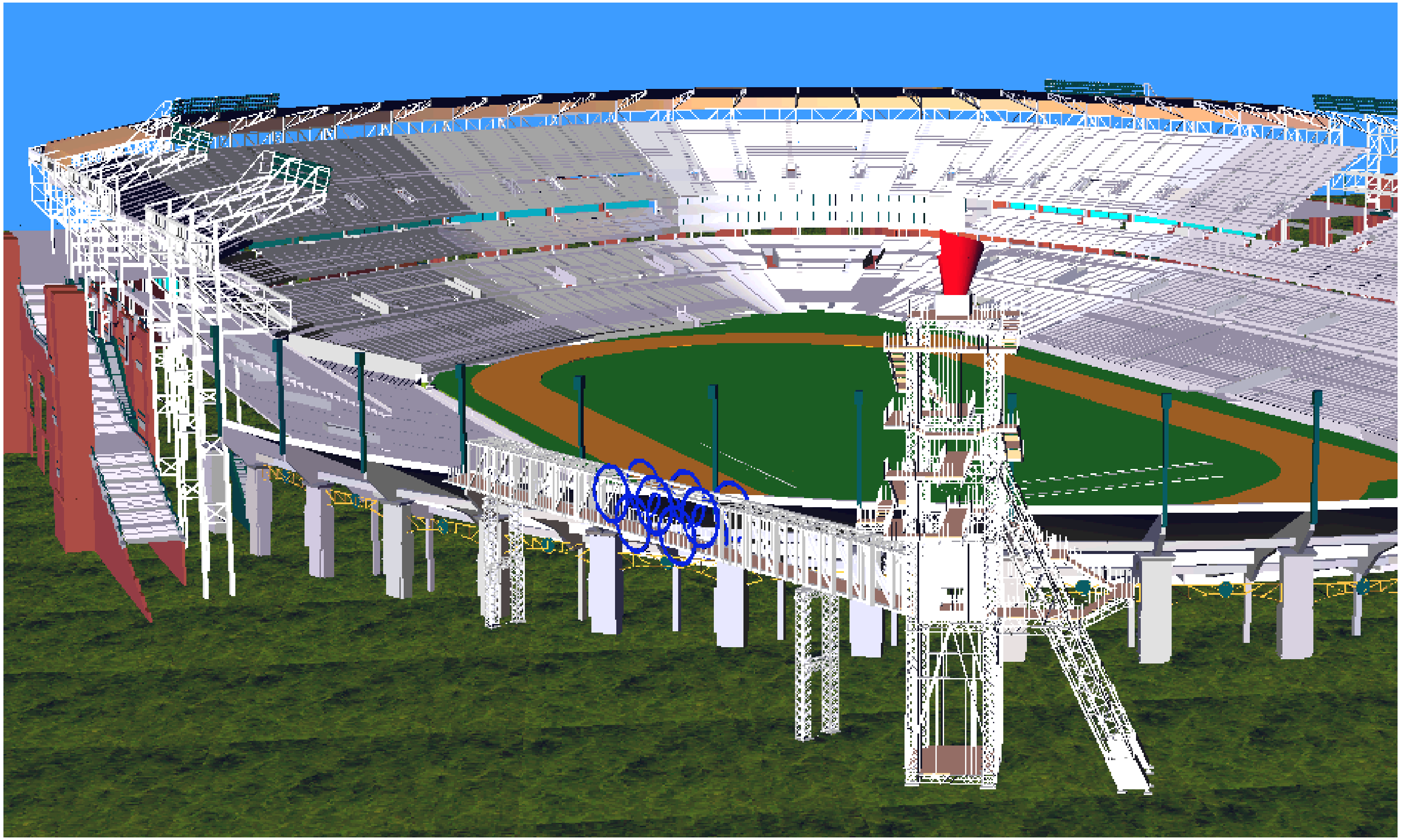
- Fits modern graphics hardware well
  - Easy to compile each LOD into triangle strips, display lists, vertex arrays, …
  - These render *much* faster than unorganized triangles on today's hardware (3-5 x)

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

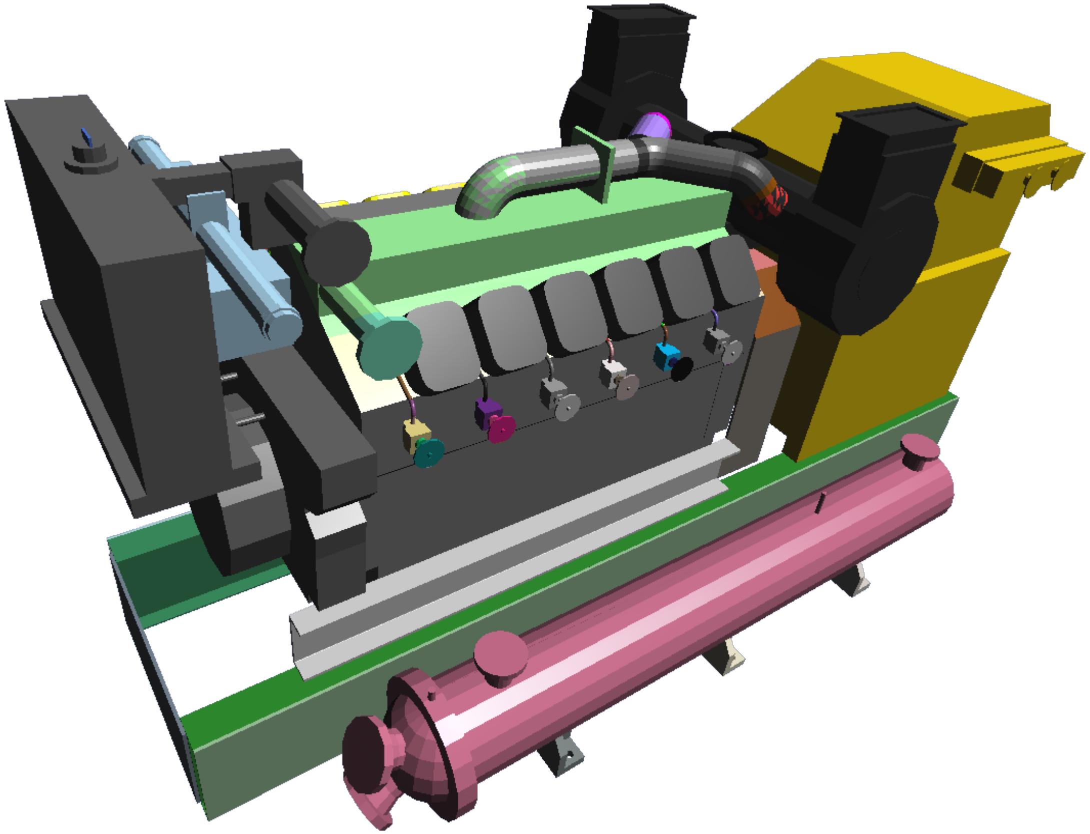
Large objects must be subdivided



Courtesy IBM and ACOG

Drastic Simplification:
The Problem With Small Objects

Small objects must be combined



Courtesy Electric Boat

Drastic Simplification

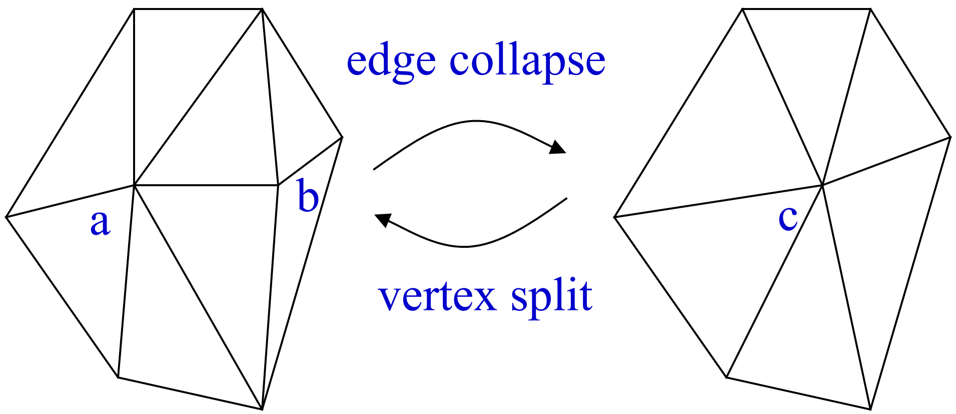
- For drastic simplification:
  - Large objects must be subdivided
  - Small objects must be combined
- Difficult or impossible with discrete LOD
- *So what can we do?*

Continuous Level of Detail

- 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*.

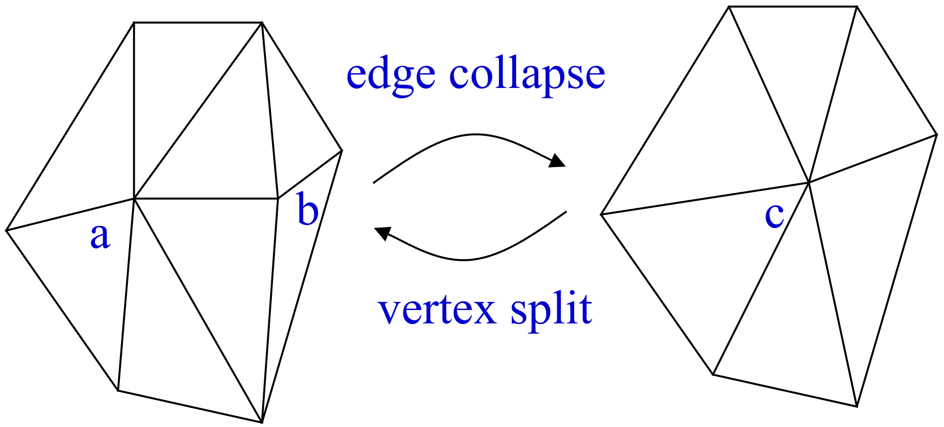
Continuous Level of Detail

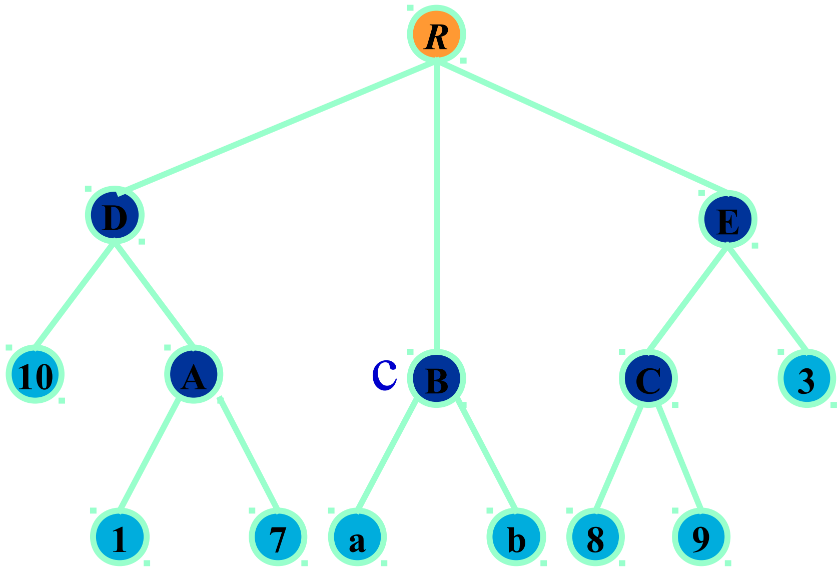
- Edge collapsing introduced by [Hoppe93]



Continuous Level of Detail

- Edge collapsing introduced by [Hoppe93]

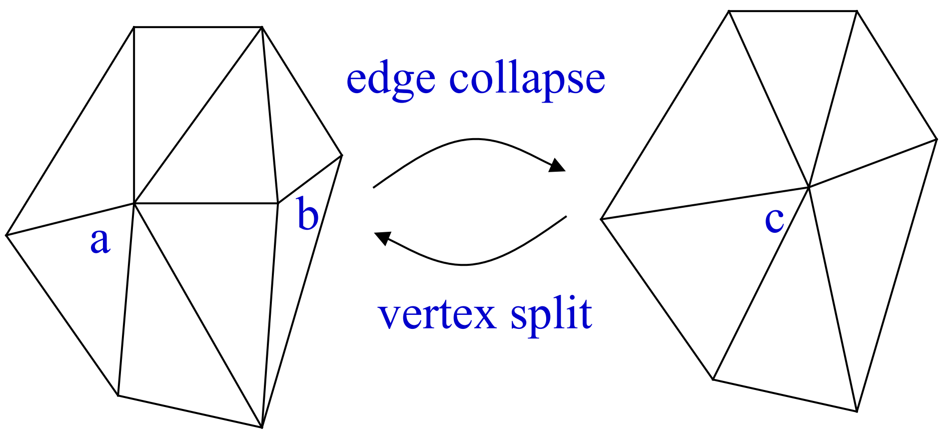


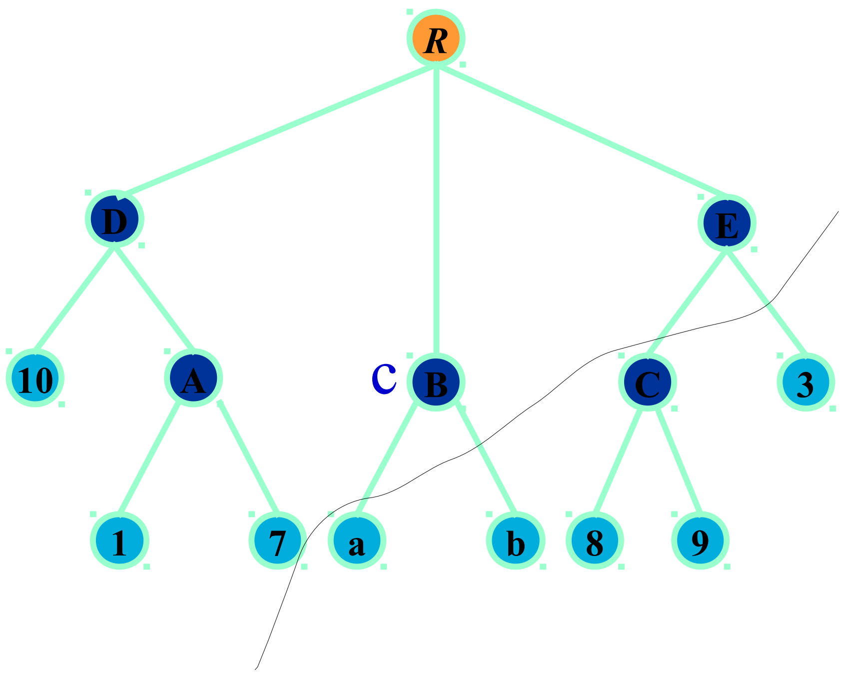


Vertex hierarchy

Continuous Level of Detail

- Edge collapsing introduced by [Hoppe93]





A cut defines an LOD

Continuous LOD: Advantages

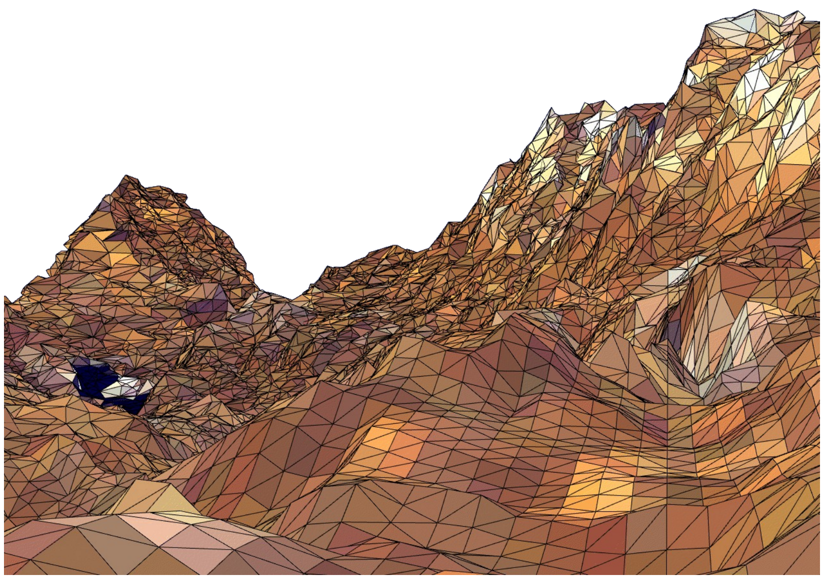
- Better granularity → better fidelity
  - LOD is specified exactly, not chosen from a few pre-created options
  - Thus objects use no more polygons than necessary, which frees up polygons for other objects
  - Net result: better resource utilization, leading to better overall fidelity/polygon

Continuous LOD: Advantages

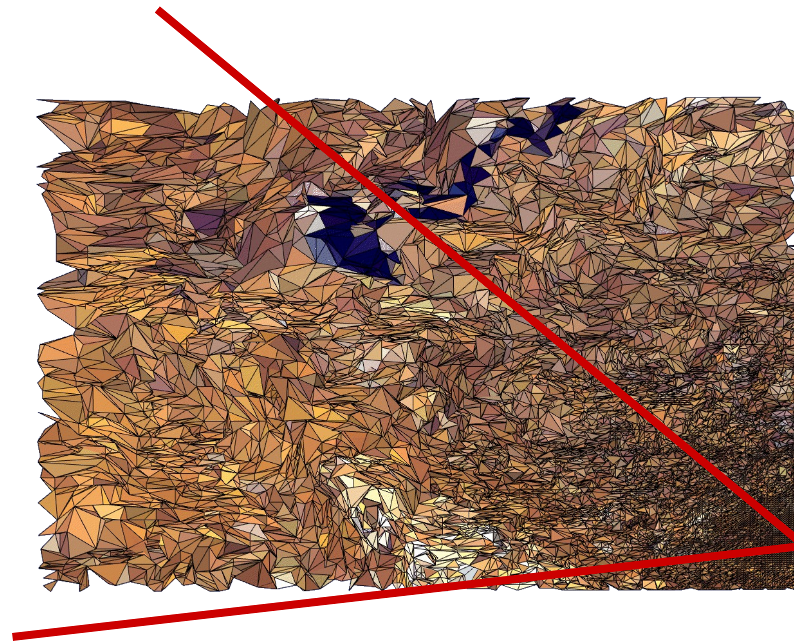
- Better granularity → smoother transitions
  - Switching between traditional LODs can introduce visual “popping” effect
  - Continuous LOD can adjust detail gradually and incrementally, reducing visual pops

View-Dependent LOD: Examples

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



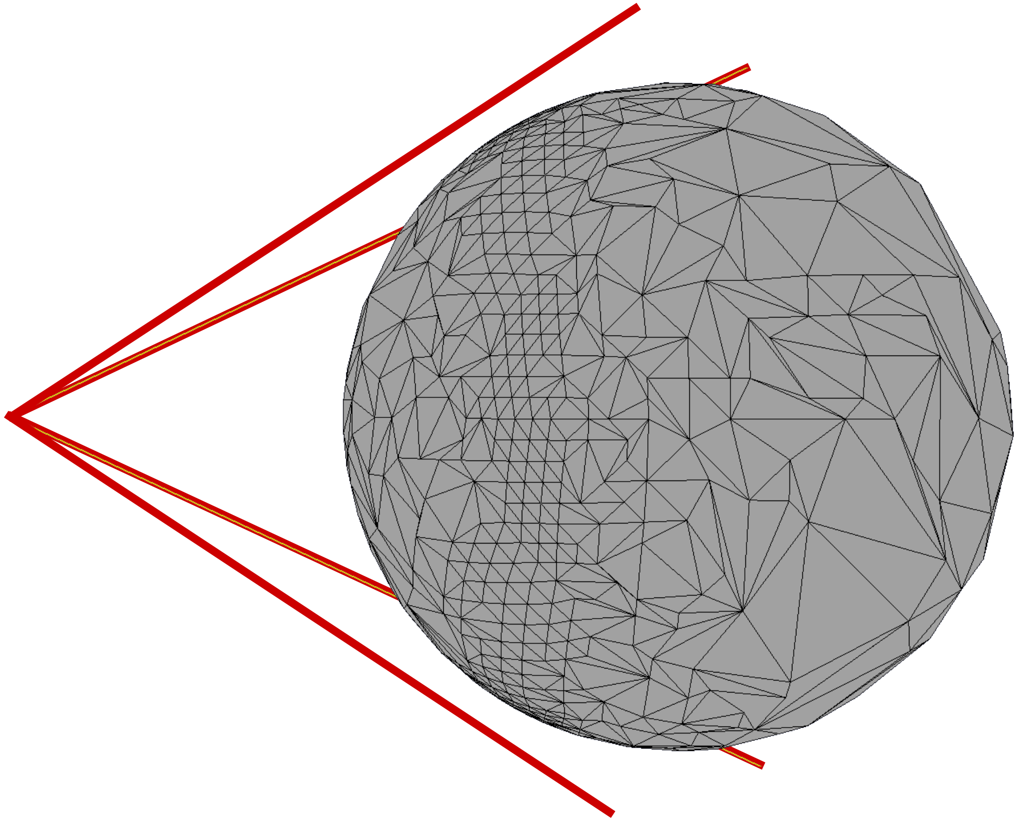
View from eyepoint



Birds-eye view

View-Dependent LOD: Examples

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



View-Dependent LOD: Advantages

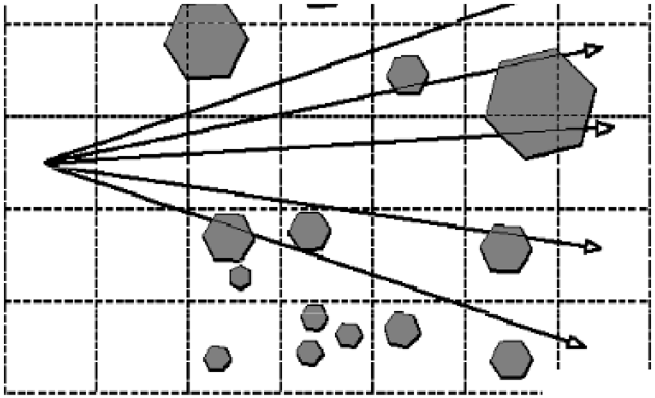
- Even better granularity
  - Allocates polygons where they are most needed, within as well as among objects
  - Enables even better overall fidelity
- Enables drastic simplification of very large objects
  - Example: stadium model
  - Example: terrain flyover

Fundamental LOD issue: where in the scene to allocate detail?

- 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
- *How might we implement this in a scene-graph based system?*



Choosing LODs

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

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)

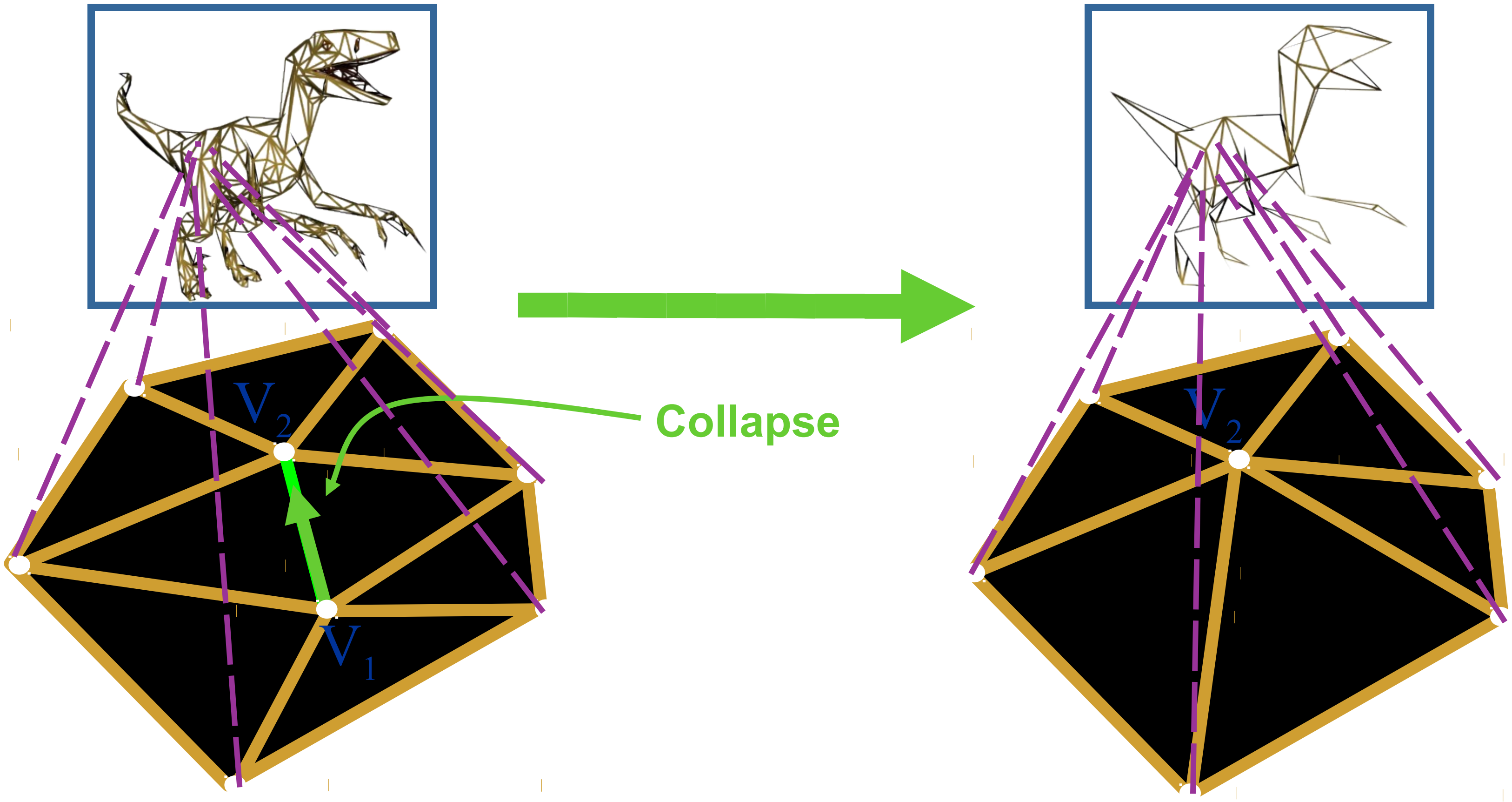
Choosing LODs:
Funkhouser & Sequin, SIGGRAPH 93

- 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 lg n)
    - Can use incremental algorithm to exploit coherence

Generating LODs

- Simplification operator:
  - Edge collapse
    - Full edge collapse
      - Better fidelity (show why)
    - Half edge collapse
      - Less memory
      - Sort vertices, tris into VAR array for fast rendering
    - Vertex-pair merge a.k.a. “virtual edge collapse”
      - Merge separate objects

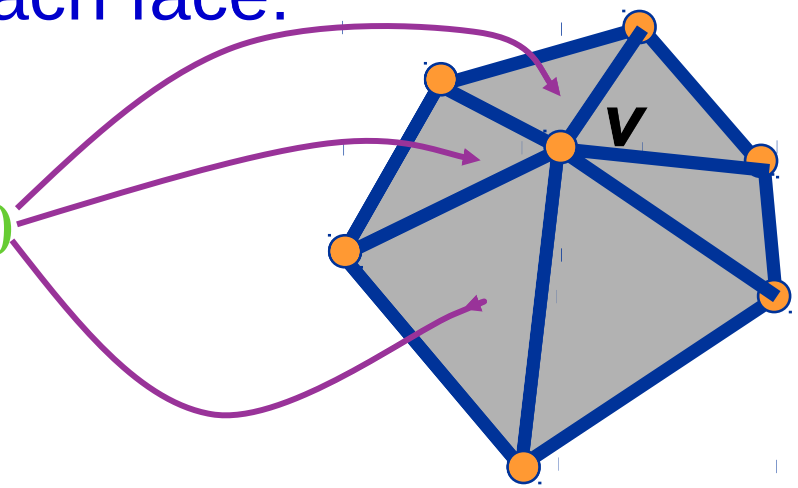
Edge Collapse Algorithm



Quadric Error Metric

- Minimize distance to all planes at a vertex
- Plane equation for each face:

$$p: \quad Ax + By + Cz + D = 0$$



Distance to vertex ***v*** :

$$p^T \cdot v = \begin{bmatrix} A & B & C & D \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Squared Distance At a Vertex

$$\Delta(v) = \sum_{p \in planes(v)} (p^T v)^2$$

$$= \sum_{p \in planes(v)} (v^T p)(p^T v)$$

$$= \sum_{p \in planes(v)} v^T (pp^T) v$$

$$= v^T \left(\sum_{p \in planes(v)} pp^T \right) v$$

Optimal Vertex Placement

- Each vertex has a quadric error metric Q associated with it
  - Error is zero for original vertices
  - Error nonzero for vertices created by merge operation(s)
- Minimize Q to calculate optimal coordinates for placing new vertex
  - Details in paper
  - Authors claim 40-50% less error

View-Dependent LOD: Algorithms

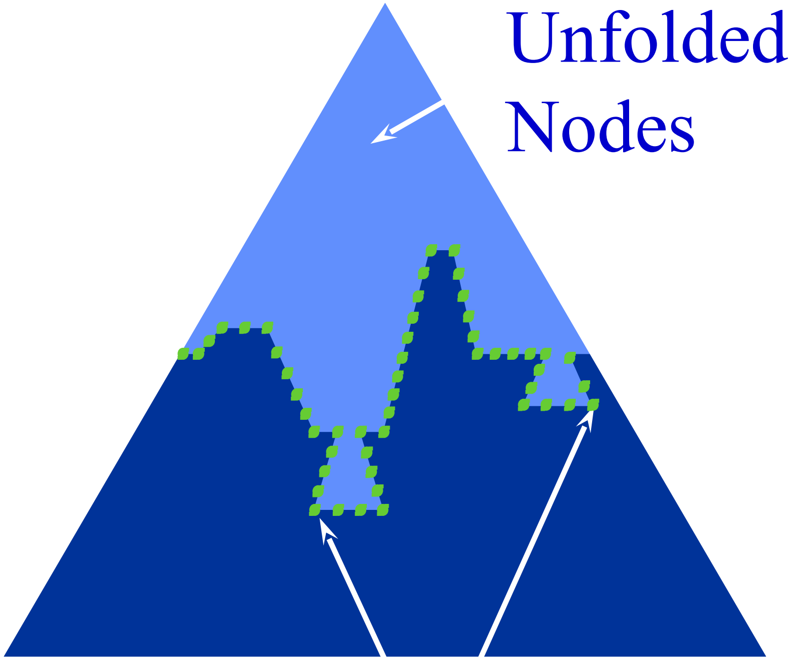
- Many good published algorithms:
  - *Progressive Meshes* by Hoppe [SIGGRAPH 96, SIGGRAPH 97, …]
  - *Hierarchical Dynamic Simplification* by Luebke & Erikson [SIGGRAPH 97]
  - *Multitriangulation* by DeFloriani et al
  - Others…

Temporal Coherence

- Exploit the fact that frame-to-frame changes are small
- One example:
  - Vertex tree

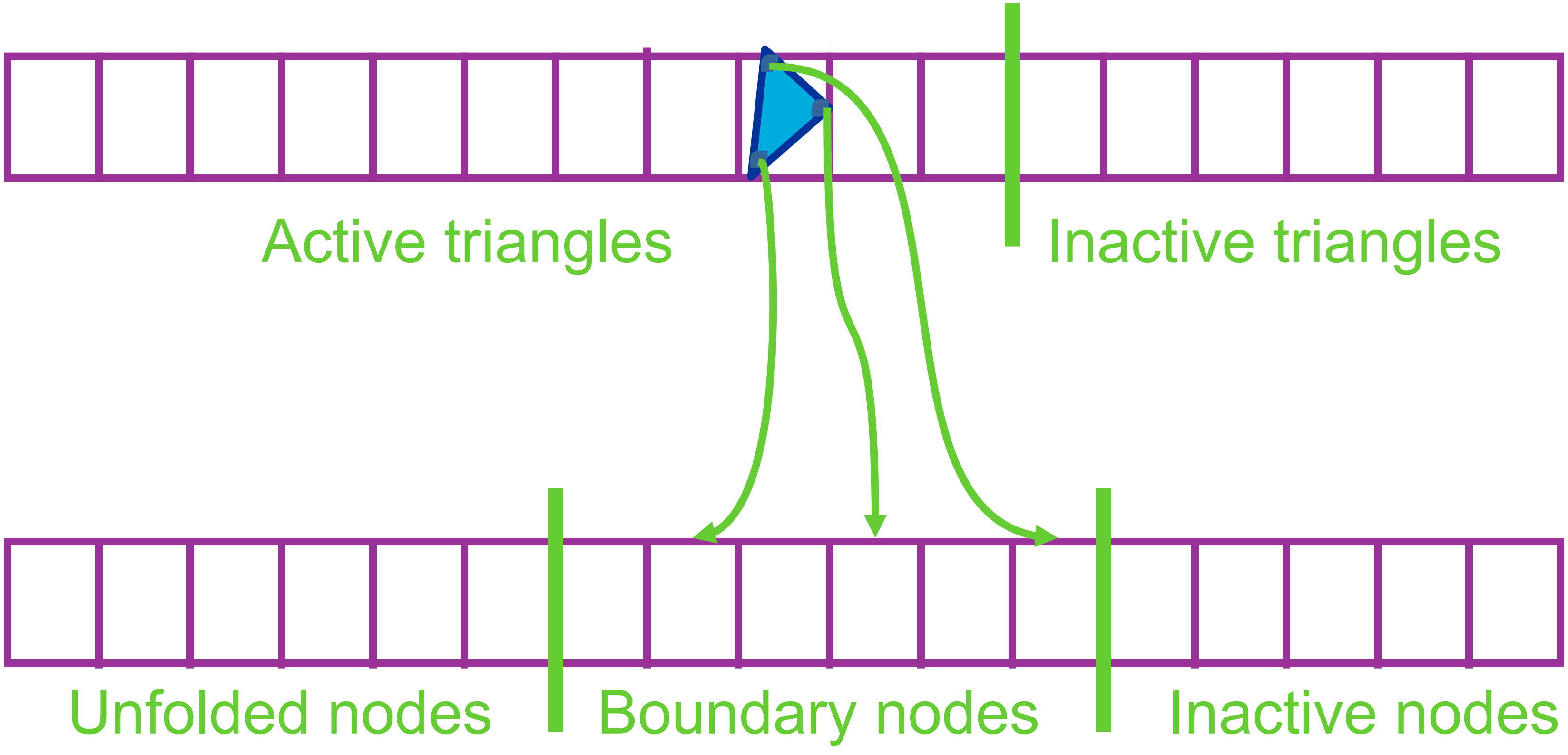
Exploiting Temporal Coherence

- Vertex Tree
  - Few nodes change per frame
  - Don’t traverse whole tree
  - Do local updates only at *boundary nodes*



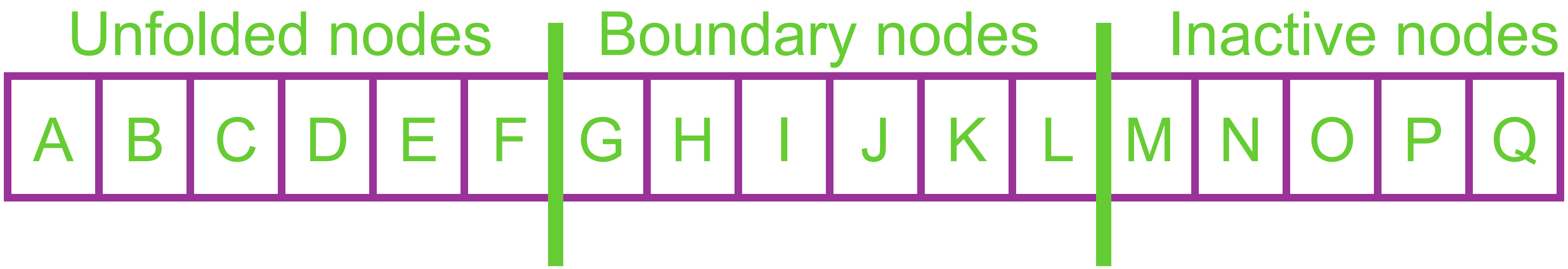
Optimizing For Rendering

- Idea: maintain geometry in coherent arrays



Optimizing For Rendering

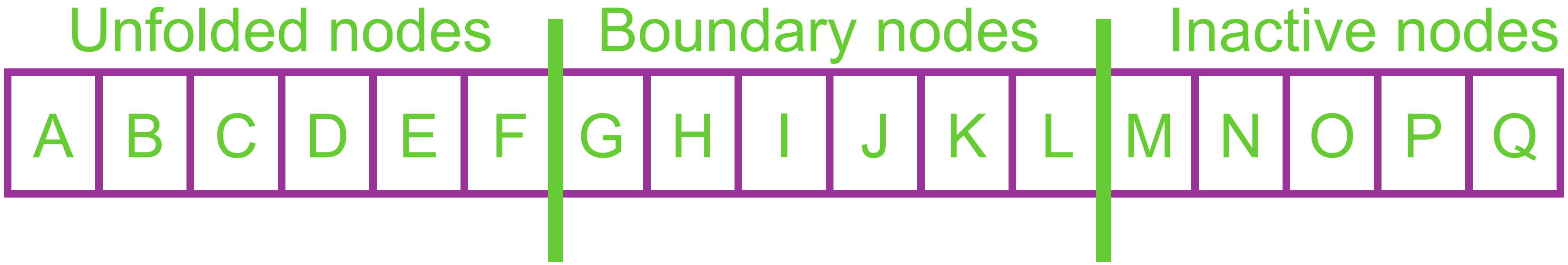
- Idea: use swaps to maintain coherence



Fold node D:

Optimizing For Rendering

- Idea: use swaps to maintain coherence

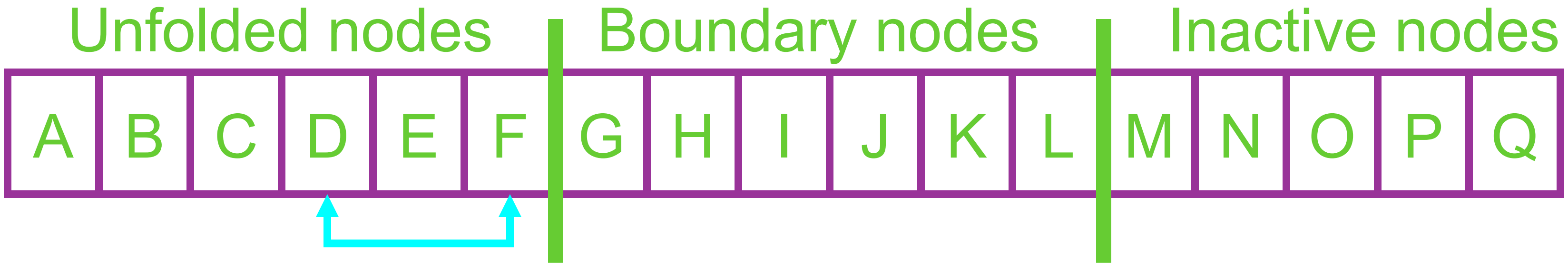


Fold node D:

Swap D with F

Optimizing For Rendering

- Idea: use swaps to maintain coherence

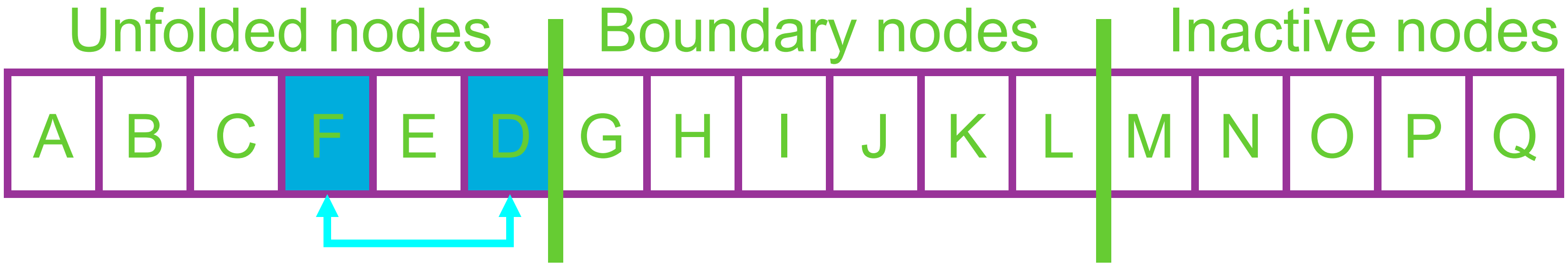


Fold node D:

Swap D with F

Optimizing For Rendering

- Idea: use swaps to maintain coherence

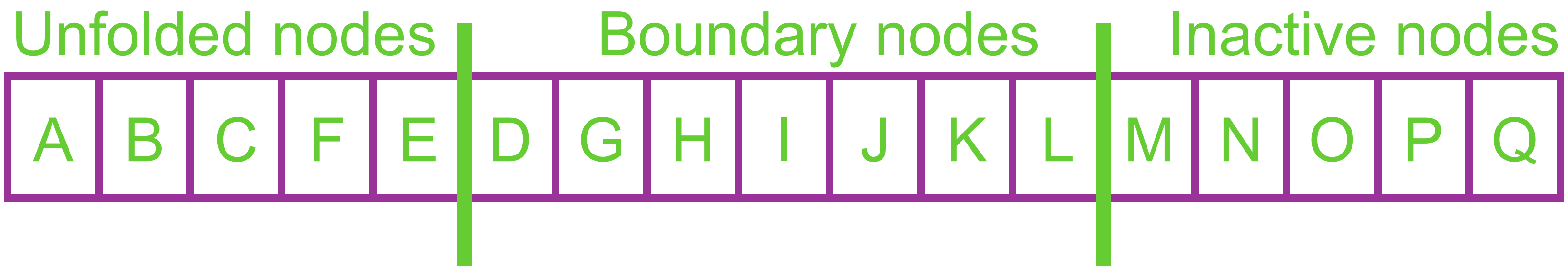


Fold node D:

Swap D with F

Optimizing For Rendering

- Idea: use swaps to maintain coherence

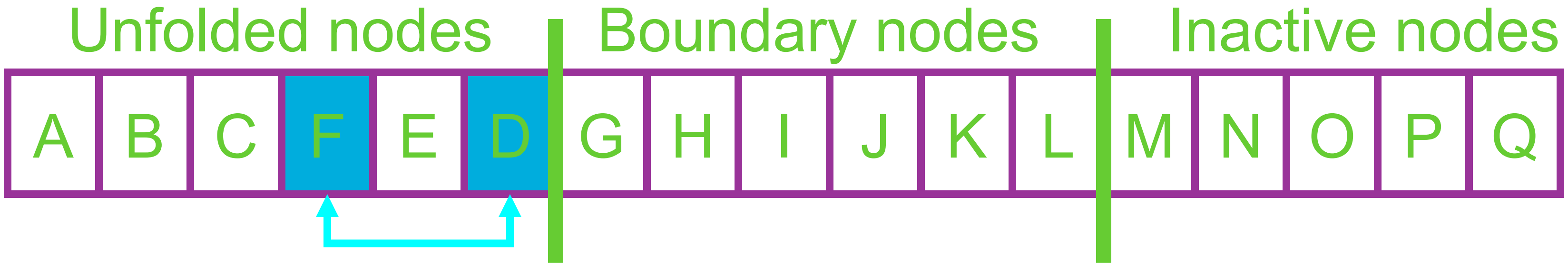


Fold node D:

Move Unfolded/Boundary Marker

Optimizing For Rendering

- Idea: use swaps to maintain coherence

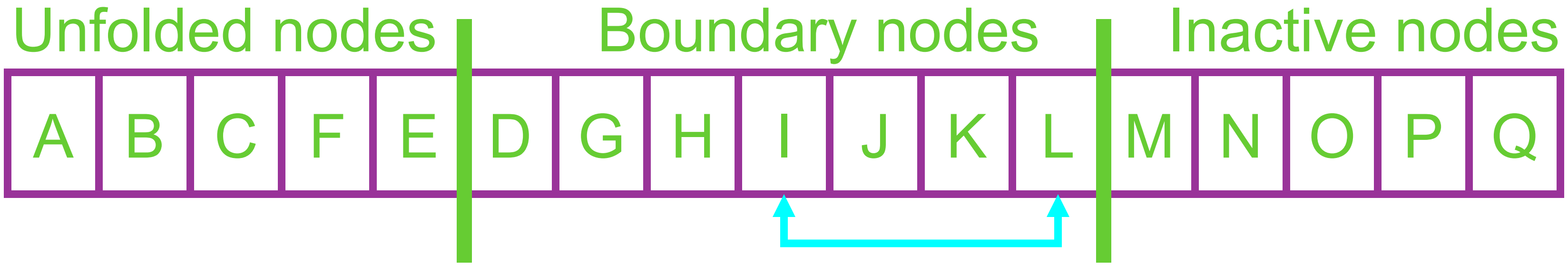


Fold node D:

Swap D with F

Optimizing For Rendering

- Idea: use swaps to maintain coherence

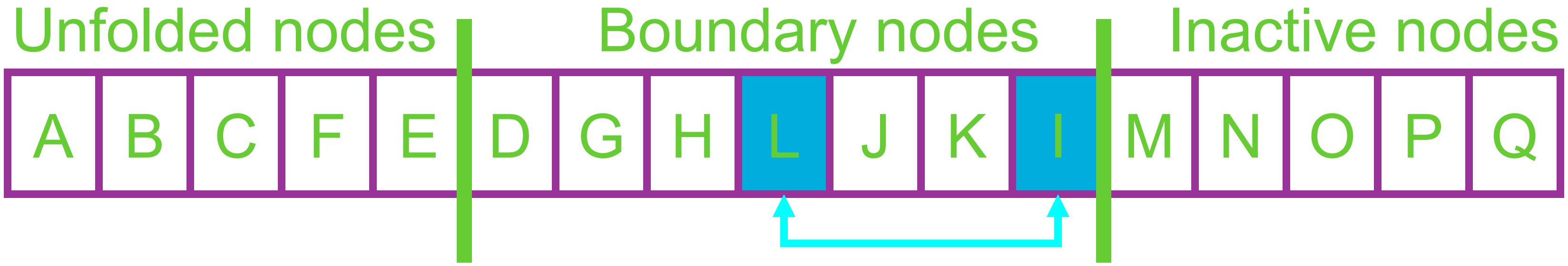


Fold node D:

Deactivate D’s children (swap w/ last boundary node)

Optimizing For Rendering

- Idea: use swaps to maintain coherence

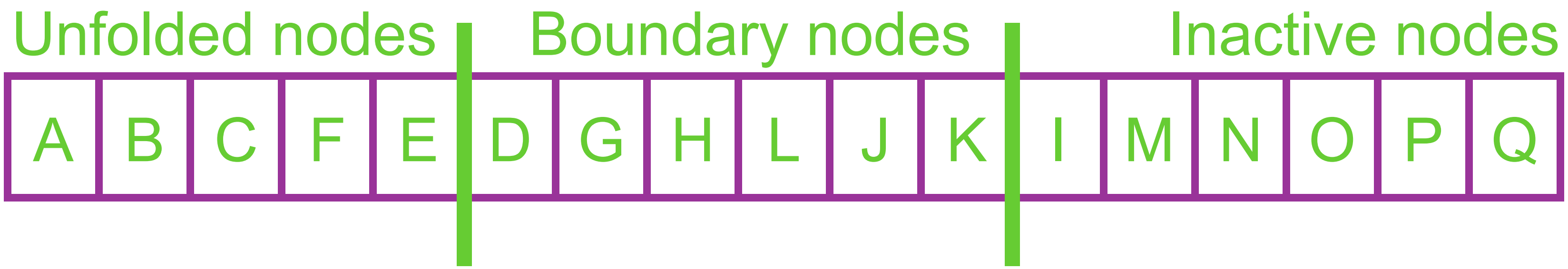


Fold node D:

Deactivate D’s children (swap w/ last boundary node)

Optimizing For Rendering

- Idea: use swaps to maintain coherence

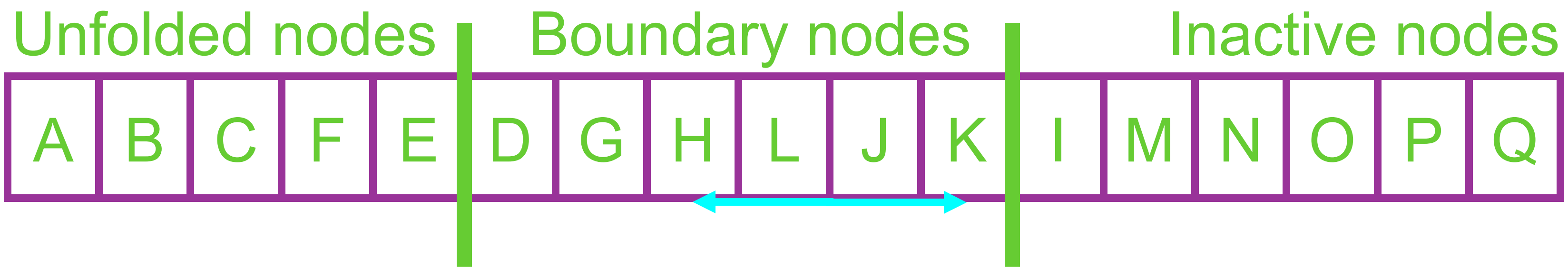


Fold node D:

Deactivate D’s children (swap w/ last boundary node)

Optimizing For Rendering

- Idea: use swaps to maintain coherence

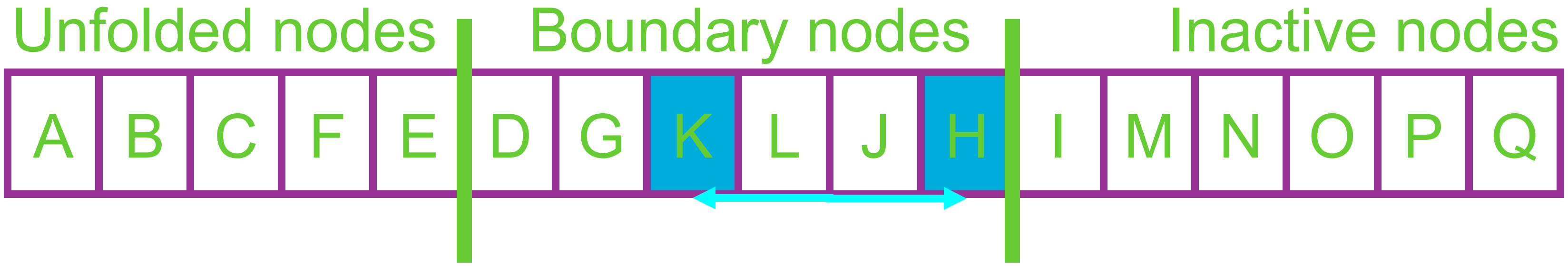


Fold node D:

Deactivate D’s children (swap w/ last boundary node)

Optimizing For Rendering

- Idea: use swaps to maintain coherence

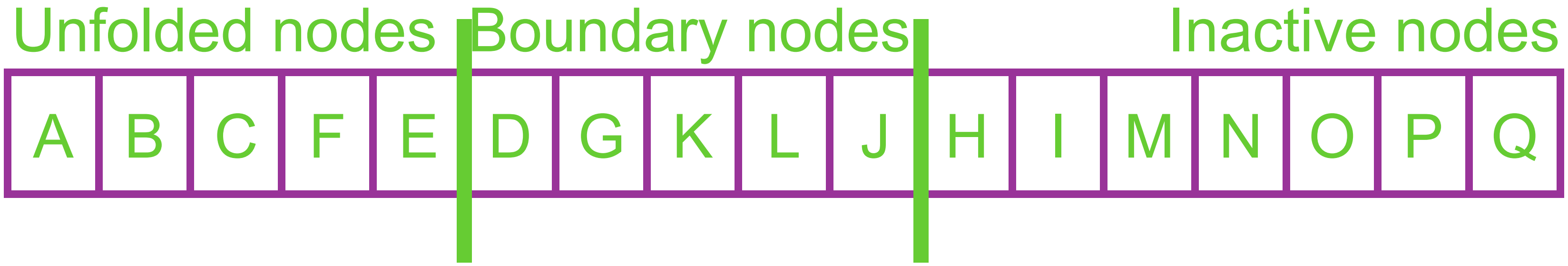


Fold node D:

Deactivate D’s children (swap w/ last boundary node)

Optimizing For Rendering

- Idea: use swaps to maintain coherence

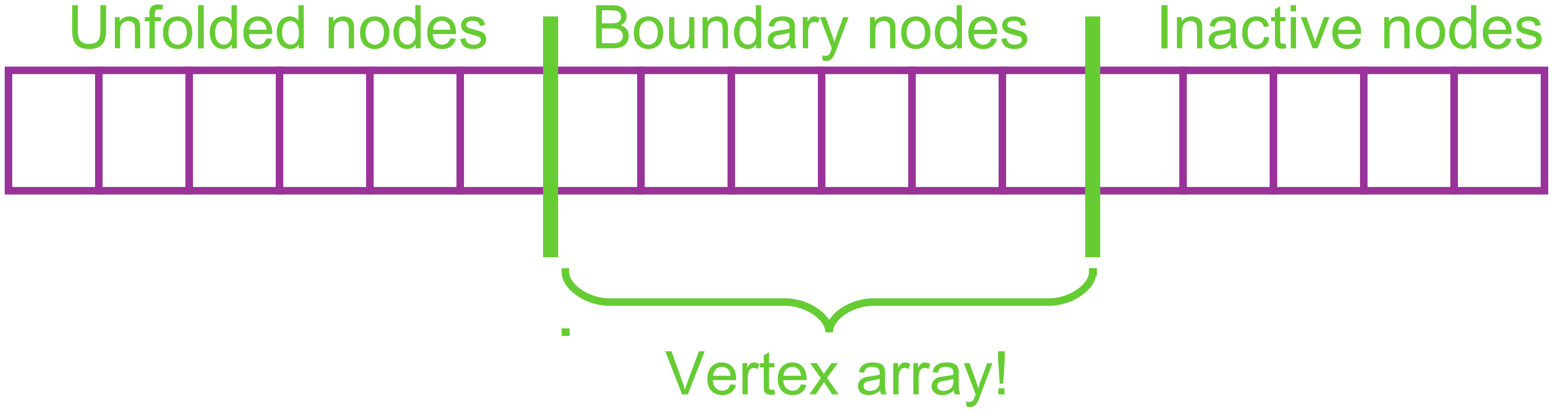


Fold node D:

Deactivate D’s children (swap w/ last boundary node)

Optimizing For Rendering:
Vertex Arrays

- Biggest win: vertex arrays



  - Actually, keep separate parallel arrays for rendering data (coords, colors, etc)

View-Dependent Versus Discrete LOD

- View-dependent LOD is superior to traditional discrete LOD when:
  - Models contain very large individual objects (e.g., terrains)
  - Simplification must be completely automatic (e.g., complex CAD models)
  - Experimenting with view-dependent simplification criteria

View-Dependent Versus Discrete LOD

- Discrete LOD is often the better choice:
  - Simplest programming model
  - Reduced run-time CPU load
  - Easier to leverage hardware:
    - Compile LODs into vertex arrays/display lists
    - Stripe LODs into triangle strips
    - Optimize vertex cache utilization and such

View-Dependent Versus Discrete LOD

- Applications that may want to use:
  - Discrete LOD
    - Video games (but much more on this later…)
    - Simulators
    - Many walkthrough-style demos
  - Dynamic and view-dependent LOD
    - CAD design review tools
    - Medical & scientific visualization toolkits
    - Terrain flyovers (e.g. google earth)

Continuous LOD:
The Sweet Spot?

- Continuous LOD may be the right compromise on modern PC hardware
  - Benefits of fine granularity without the cost of view-dependent evaluation
  - Can be implemented efficiently with regard to
    - Memory
    - CPU
    - GPU

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

Implementation: *VDSlib*

A public-domain view-dependent simplification and rendering package

Available at *http://vdslib.virginia.edu*