i3D 2006
Symposium on Interactive 3D Graphics and Games
Sponsored by ACM SIGGRAPH

# Triangle Order Optimization for Efficient Graphics Hardware Computation Culling

Diego Nehab
Princeton

Joshua Barczak
UMBC/ATI Research

Pedro V. Sander
ATI Research

# Outline of talk

- Problem statement
  - The rendering pipeline
  - Two key hardware optimizations
  - Fine tuned triangle orderings
- Our algorithm
  - Unified triangle order
  - Planar patch clustering
  - Inter-cluster ordering

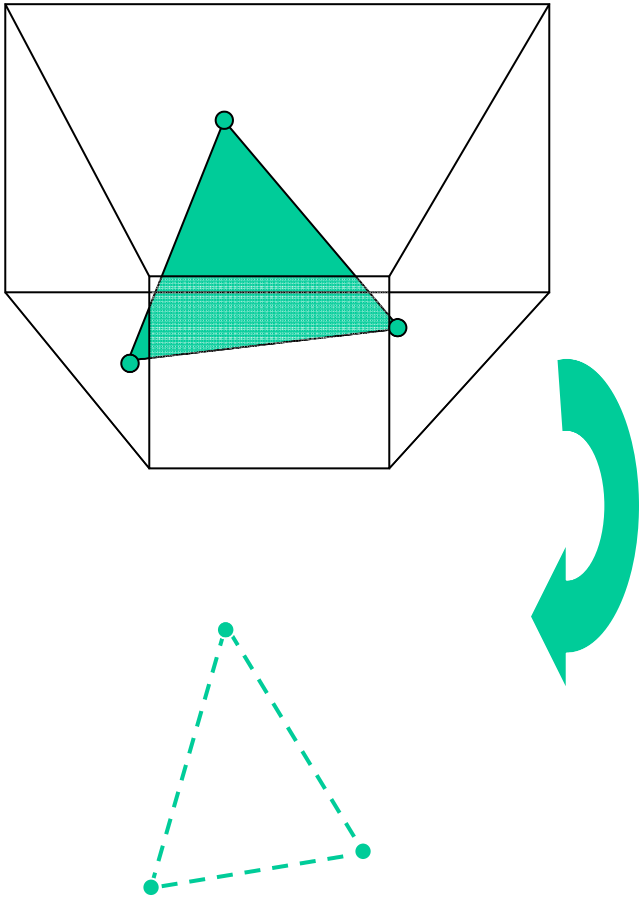
# The rendering pipeline

- Programmable stages
    - **Vertex processing**
    - Pixel processing

# Vertex programs

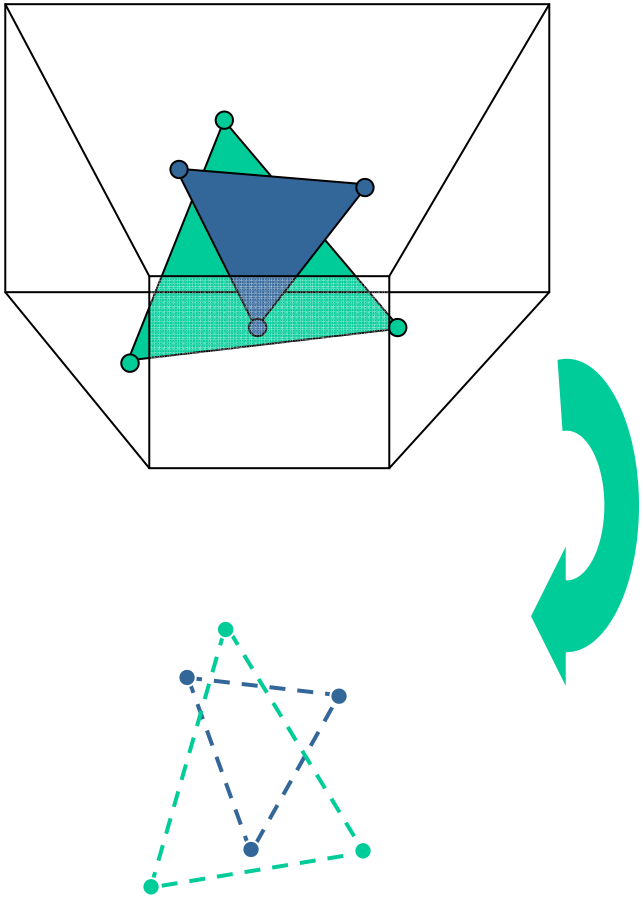
- Each vertex causes a program to be run
- Programs usually perform
  - Model, World, Projection Transforms
  - Skinning for animation
  - Per-vertex lighting (Gouraud shading)
- Can dominate rendering cost
  - Too many vertices
  - Expensive operations

# Vertex reuse



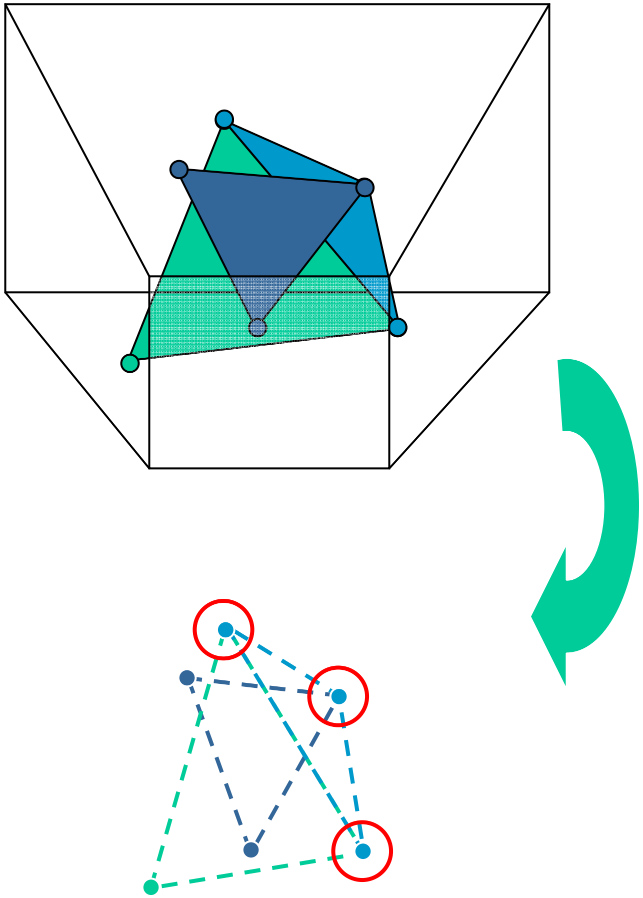
- Triangles reference vertices
- Referenced vertices are transformed by costly programs
- Hardware optimization:
    - Vertex cache
    - Reuse transformed vertices
- Software strategy:
    - Order triangles to maximize vertex locality
    - [Deering 1995]
      [Chow 1997]
      [Hoppe 1999]
      …

# Vertex reuse



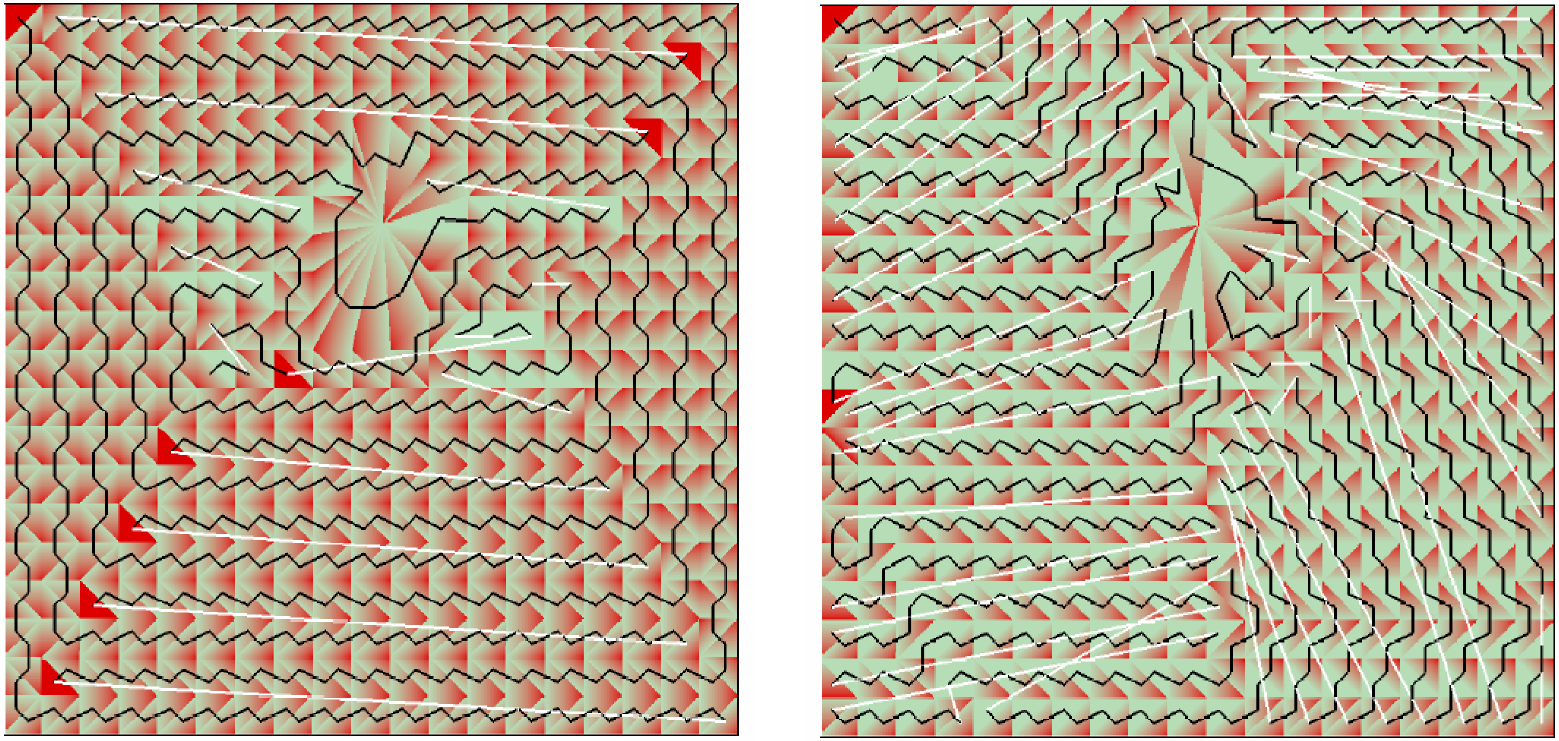
- Triangles reference vertices
- Referenced vertices are transformed by costly programs
- Hardware optimization:
  - Vertex cache
  - Reuse transformed vertices
- Software strategy:
  - Order triangles to maximize vertex locality
  - [Deering 1995]<br>[Chow 1997]<br>[Hoppe 1999]<br>…

# Vertex reuse



- Triangles reference vertices
- Referenced vertices are transformed by costly programs
- Hardware optimization:
  - Vertex cache
  - Reuse transformed vertices
- Software strategy:
  - Order triangles to maximize vertex locality
  - [Deering 1995]
    [Chow 1997]
    [Hoppe 1999]
    …

# Vertex cache efficiency



Long strips – 0.99 v/t

Hoppe 1999 – 0.60 v/t

Best possible – 0.5 v/t
Worst possible – 3 v/t

# Cache miss (demo)

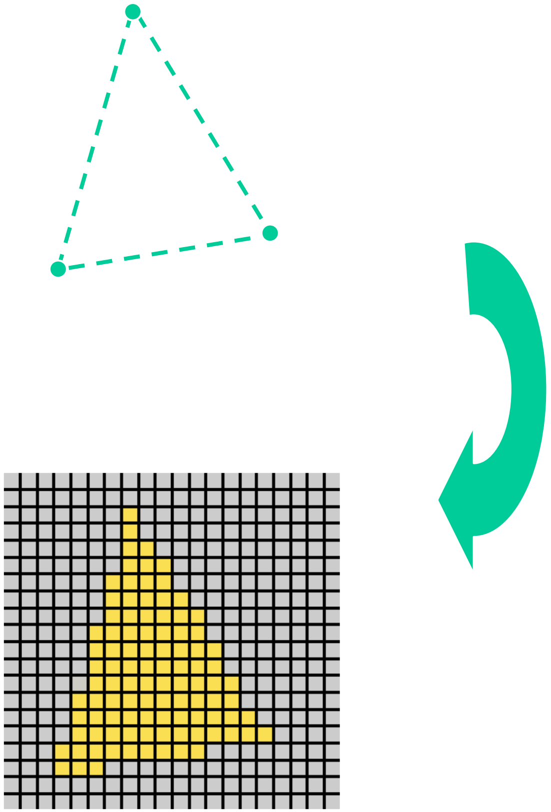
# The rendering pipeline

- Programmable stages
  - **Vertex processing**
  - Pixel processing

# Pixel programs

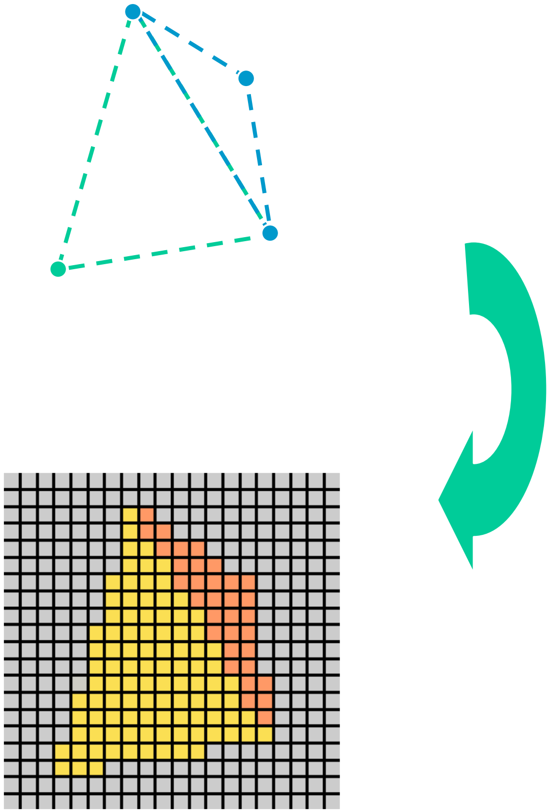
- Each pixel causes a program to be run
- Programs usually perform
  - Texture lookups
  - Shadow mapping
  - Reflections
  - Per-pixel lighting (Phong shading)
- Can dominate rendering cost
  - Too much overdraw
  - Expensive operations

# Overdraw



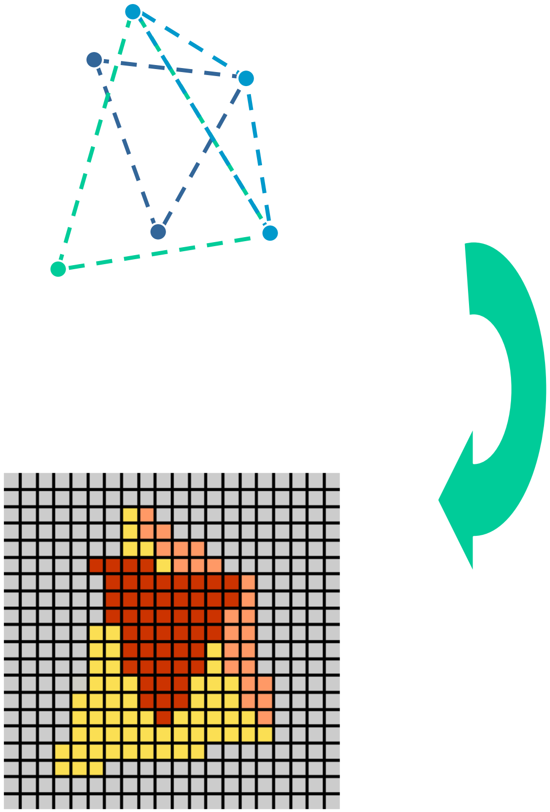
- Triangles are rasterized in order
- Colors are computed by costly programs
- Hardware optimization:
  - Early Z-culling
  - Skip execution off hidden pixels
- Software strategy:
  - Front-to-back rendering?
  - Minimizes overdraw

# Overdraw



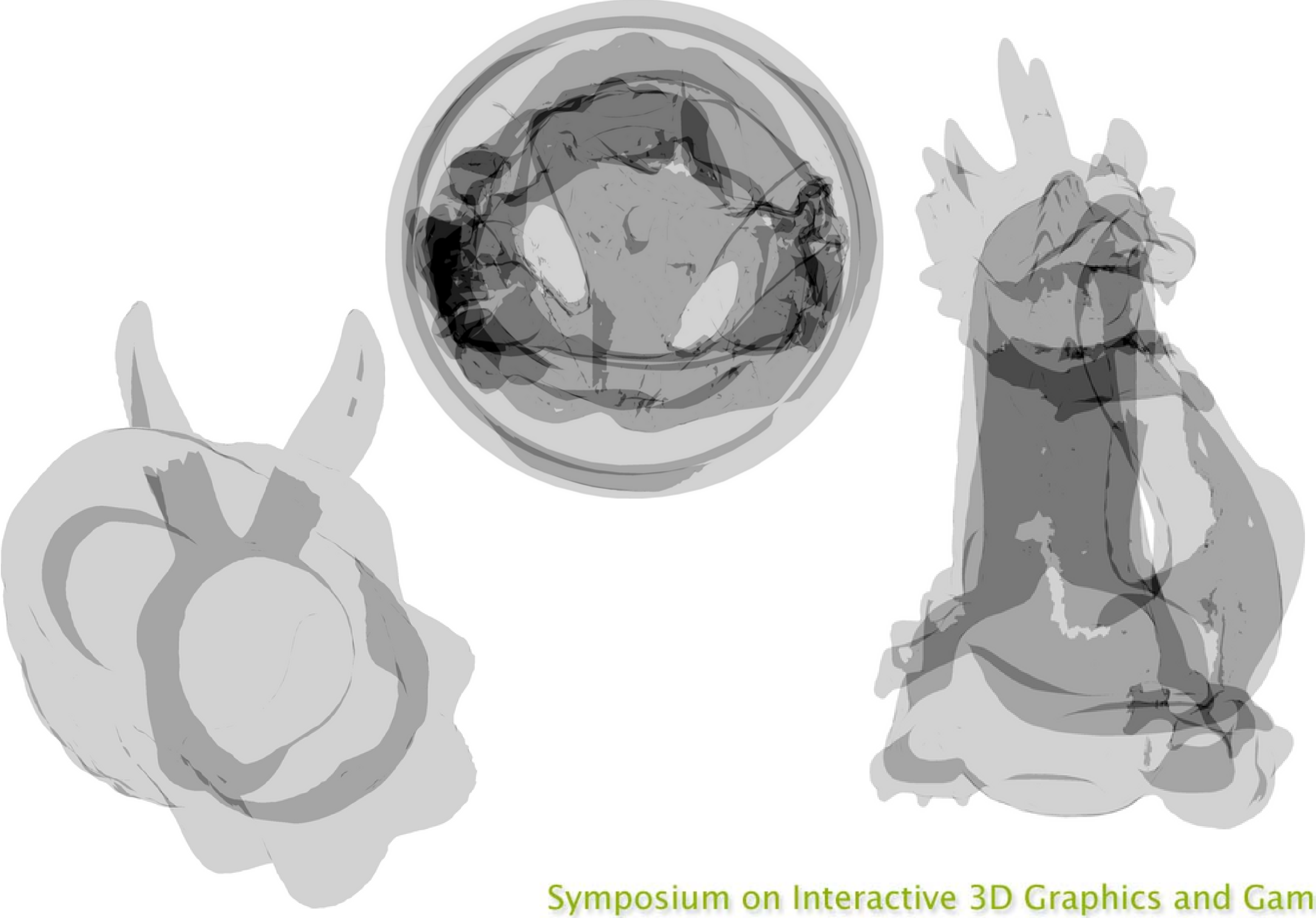
- Triangles are rasterized in order
- Colors are computed by costly programs
- Hardware optimization:
  - Early Z-culling
  - Skip execution off hidden pixels
- Software strategy:
  - Front-to-back rendering?
  - Minimizes overdraw

# Overdraw



- Triangles are rasterized in order
- Colors are computed by costly programs
- Hardware optimization:
  - Early Z-culling
  - Skip execution off hidden pixels
- Software strategy:
  - Front-to-back rendering?
  - Minimizes overdraw

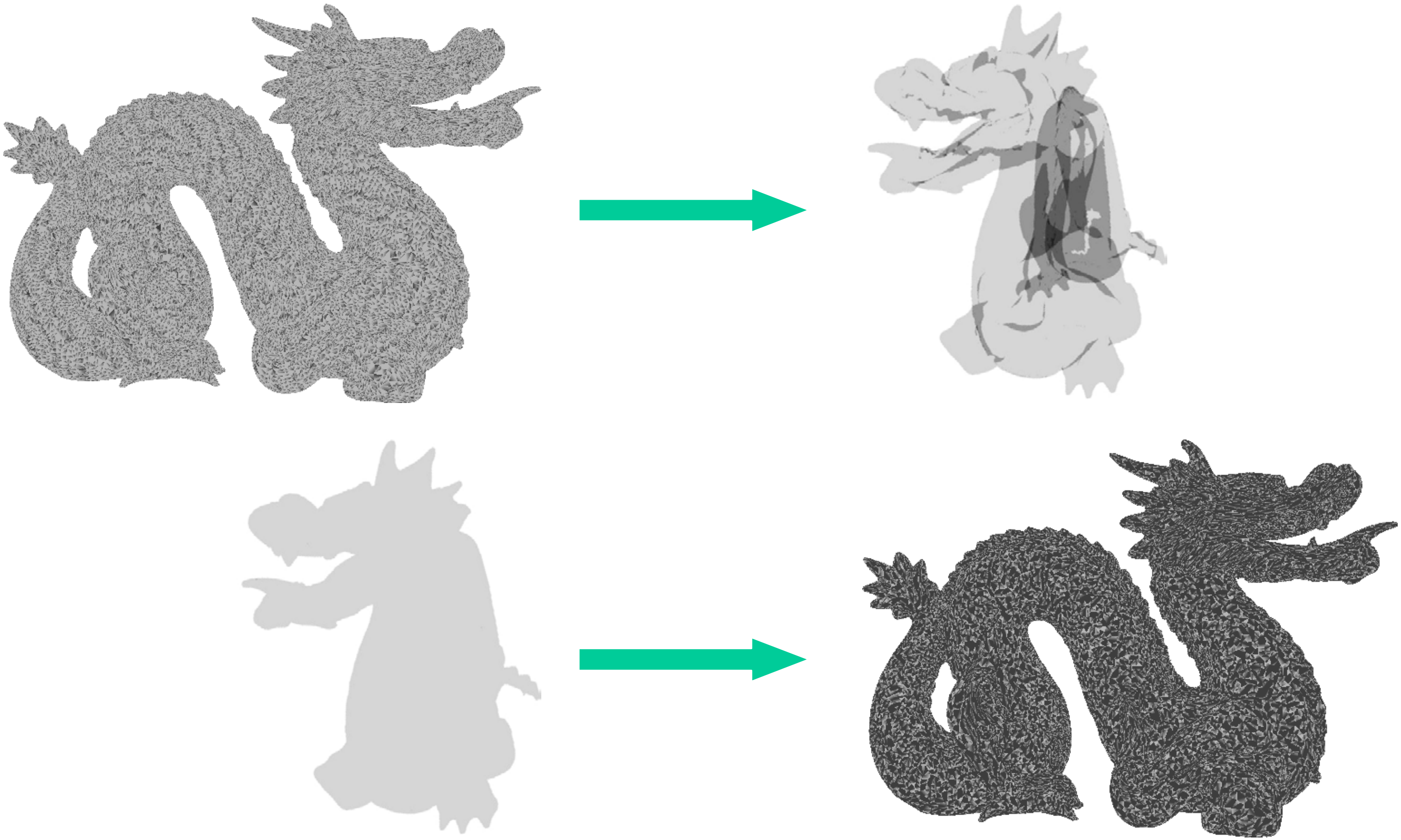
# Overdraw (demo)



# What is the ideal scenario?

- Applications can be vertex-bound, pixel-bound or both (depending on viewpoint)
- Want to preserve mesh locality
- Want to eliminate overdraw
- But how?

# Goals seem incompatible



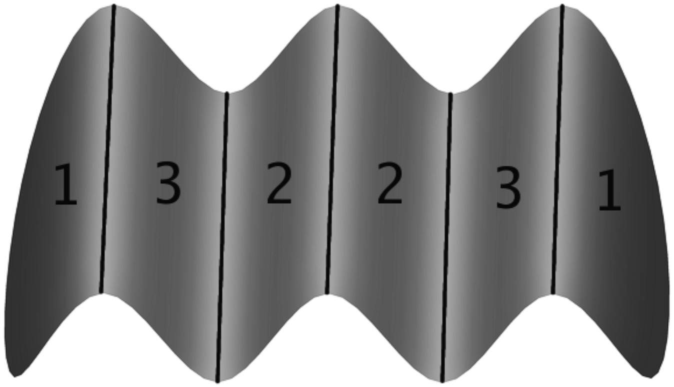
# Alternatives

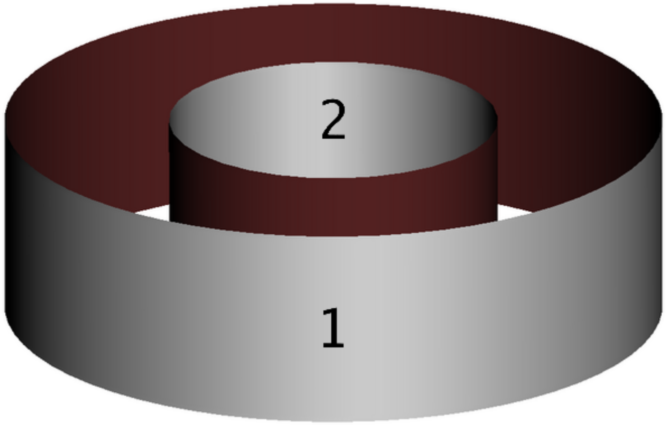
- Dynamic depth-sort
    - Can be too expensive
    - Destroys mesh locality
- Sorting per object
    - Does not eliminate intra-object overdraw
- Z-buffer priming
    - Can be too expensive

# Our goal

- Simple solution
- Good in both vertex and pixel bound scenarios
- Transparent to application

# Insight: View Independent Ordering





- Back-face culling is often used
- Convex objects have *no* overdraw, regardless of viewpoint
- Might be possible even for concave objects!

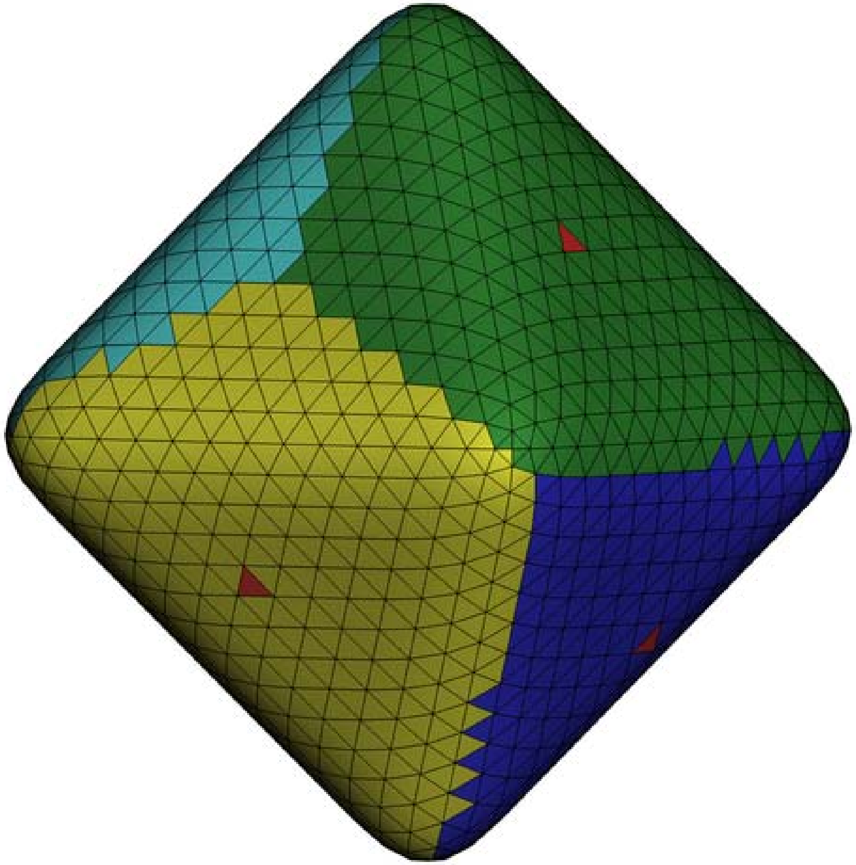
# Algorithm overview

- Cluster mesh into planar patches
    - Lloyd-Max relaxation
- Sort clusters to minimize overdraw
    - Minimum feedback arc set
- Optimize for locality within clusters
    - Off-the-shelf [Hoppe 1999, …]

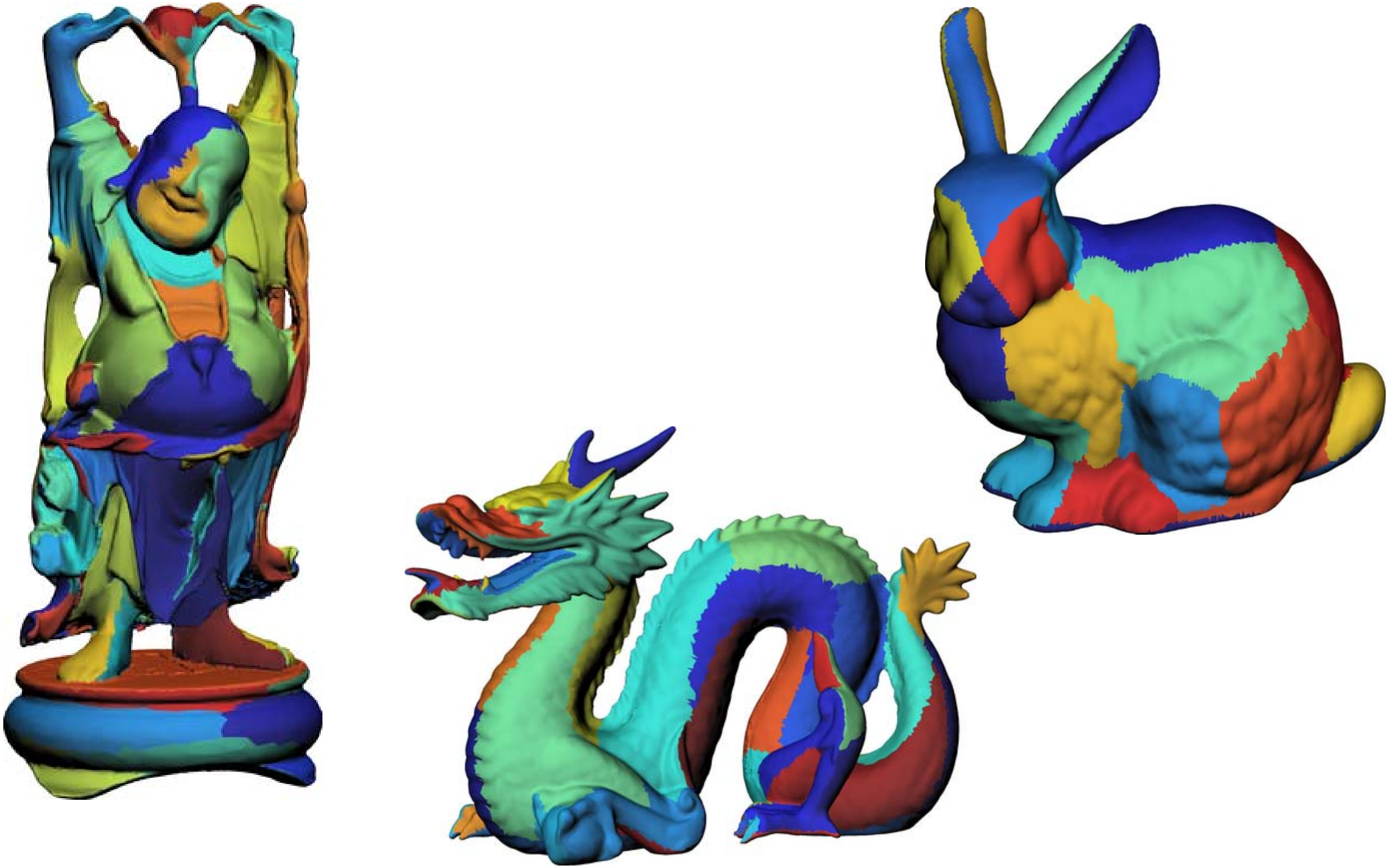
# Planar patch clustering

- Start with one random triangle seed
- Repeat
  - Grow clusters from seeds
    - Dijkstra on dual graph
    - Penalize normal variation
  - Move seeds to centroids of clusters
    - Reverse Dijkstra
  - Add new seed on last visited triangle
- Based on Sander 2003, different metric

# Step-by-step (demo)



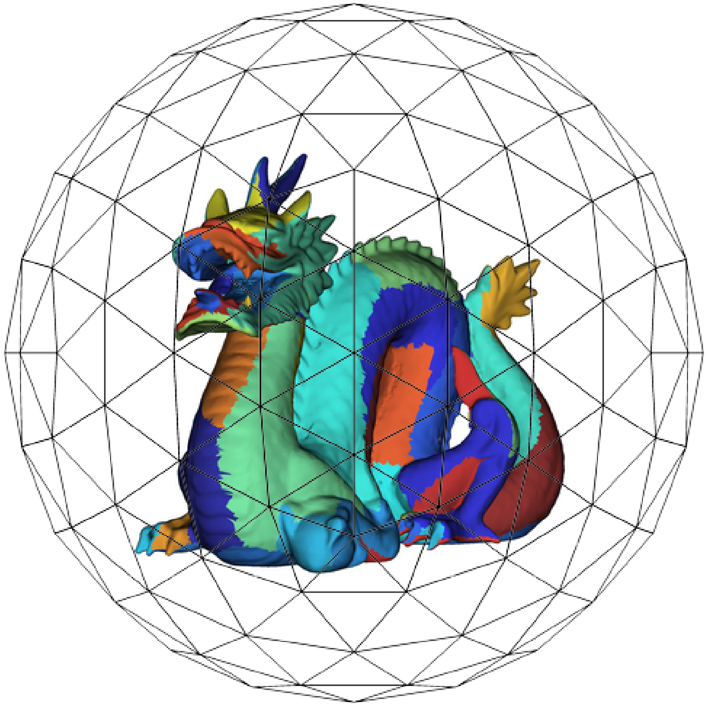
# Real meshes (demo)



# Inter-cluster ordering

- Want to start with clusters that are likely to occlude others
- Encode priority information into a partial order graph
- For each pair of clusters, edge gives cost of drawing a cluster before the other
- Sort clusters respecting partial orders
- Should minimize overdraw

# Finding the partial order graph



- For each pair of clusters
- Render in both orders
- Compare number of pixels generated
  - Occlusion query
- Repeat for many camera positions
- Add *one* arc with net pixel difference between clusters

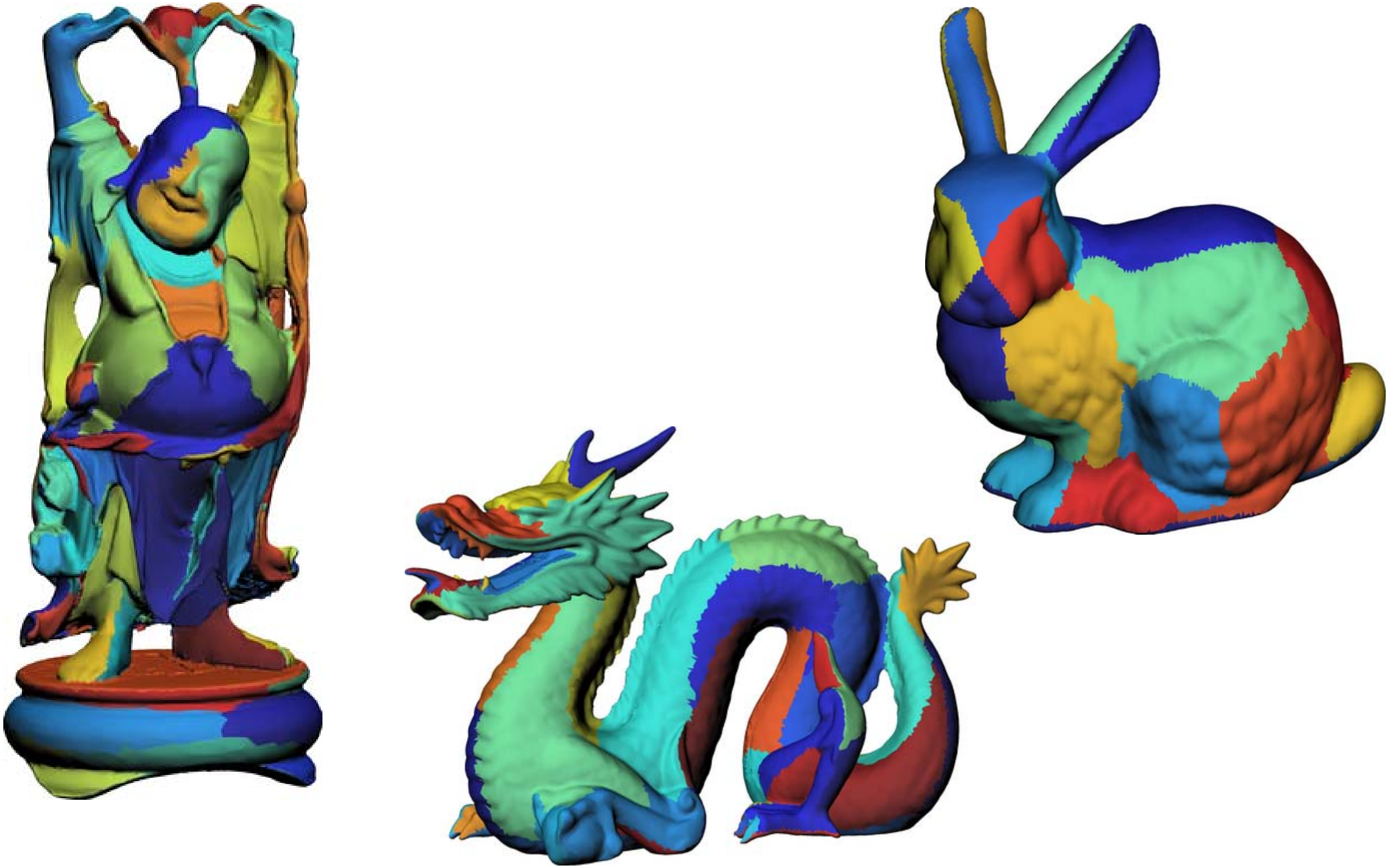
# Minimum Feedback Arc Set

- Graphs are usually dense (1/2 of all edges)
- Lack of cycles makes problem trivial
  - Topological sort
- Cycles make the problem untreatable
  - APX-Hard
- We don't need the optimal solution
- Simple O(n) heuristic is good enough [Skiena 1997]
- Agrees with topological sort

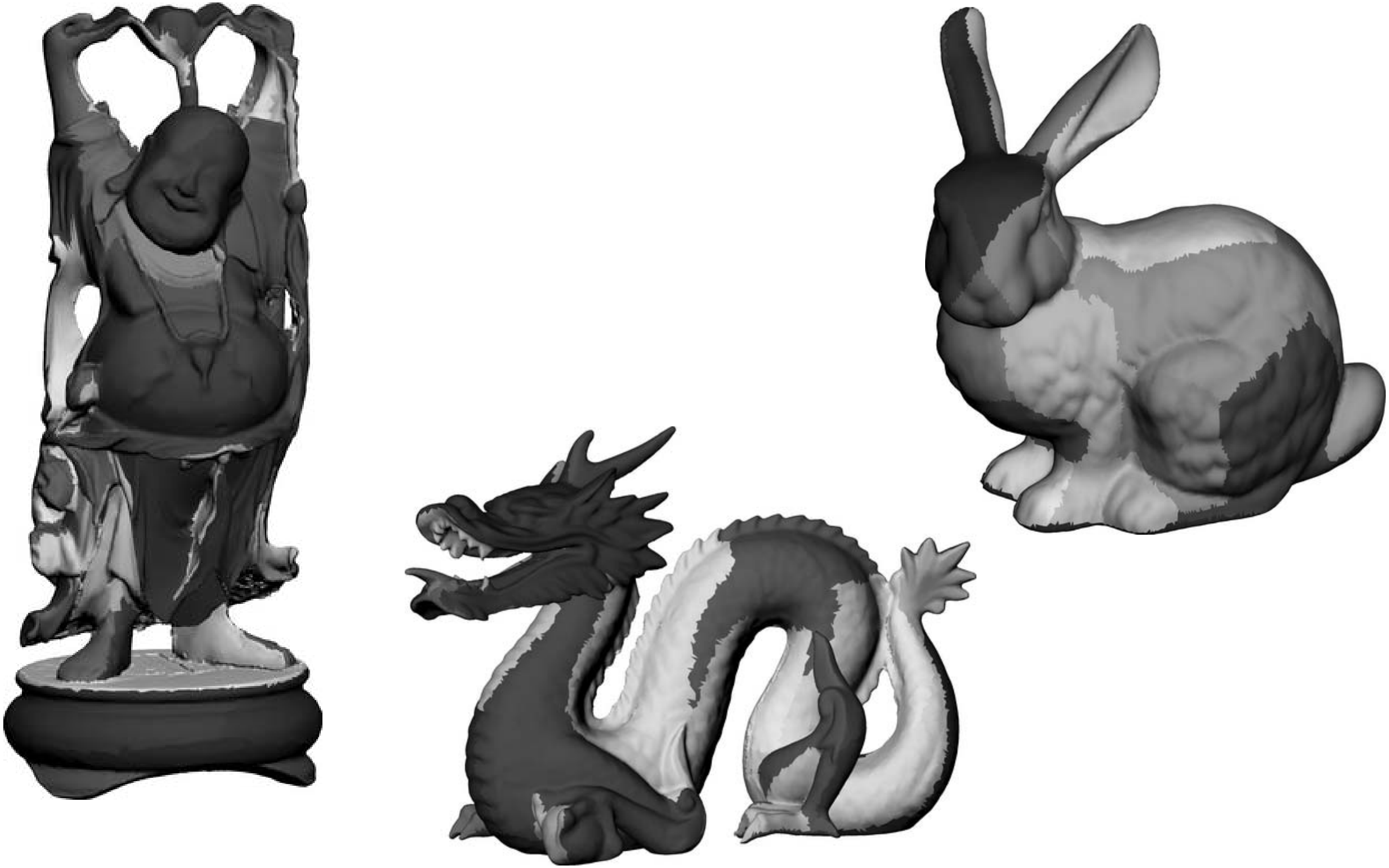
# Heuristic

- Nodes with no incoming arc
  - Output first
- Nodes with no outgoing arc
  - Output last
- On output, remove node and arcs
- At some point, nodes will have both incoming and outgoing arcs
- Output node with greatest net edge cost to the appropriate end of list
- Repeat

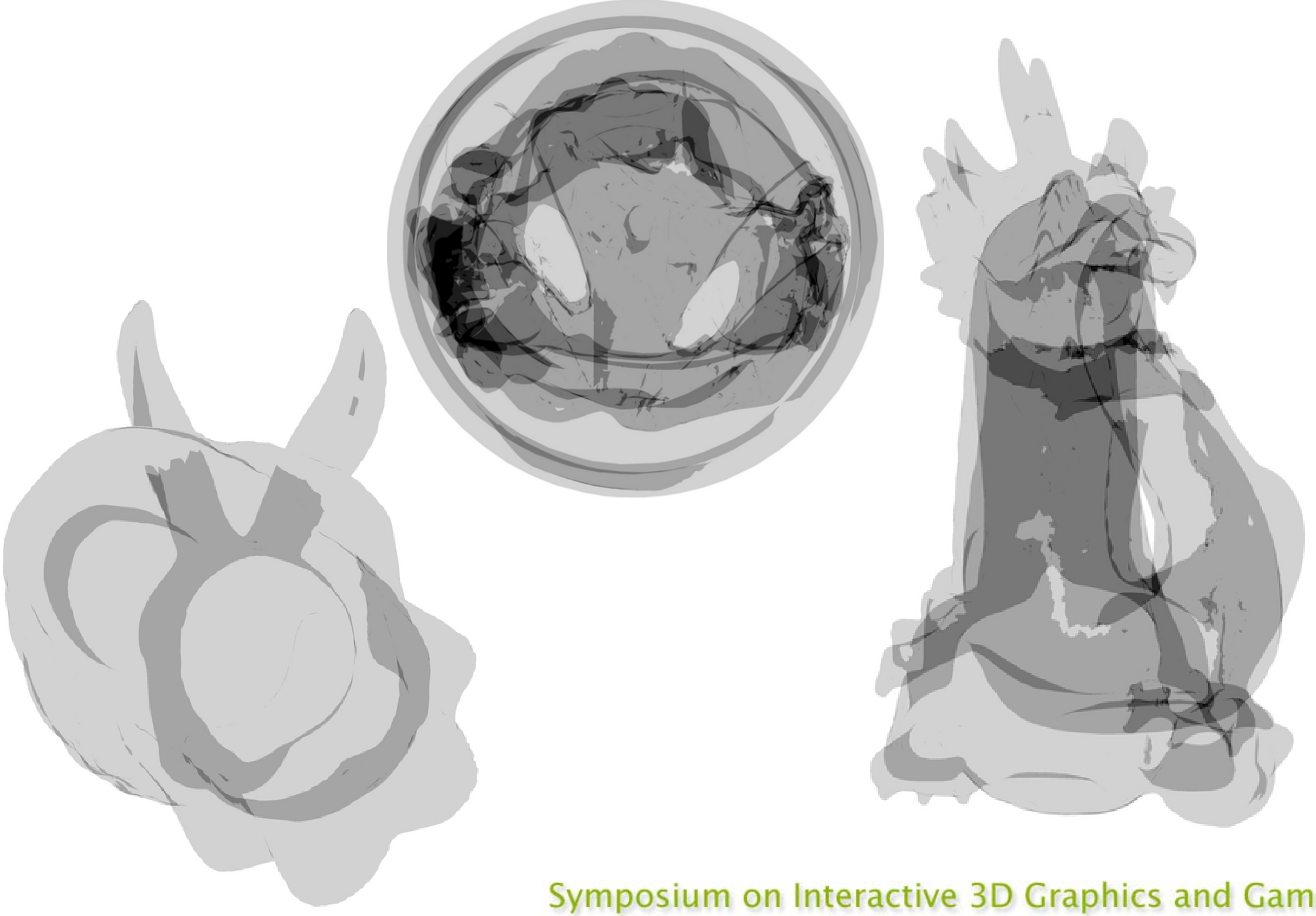
# Planar patch clustering



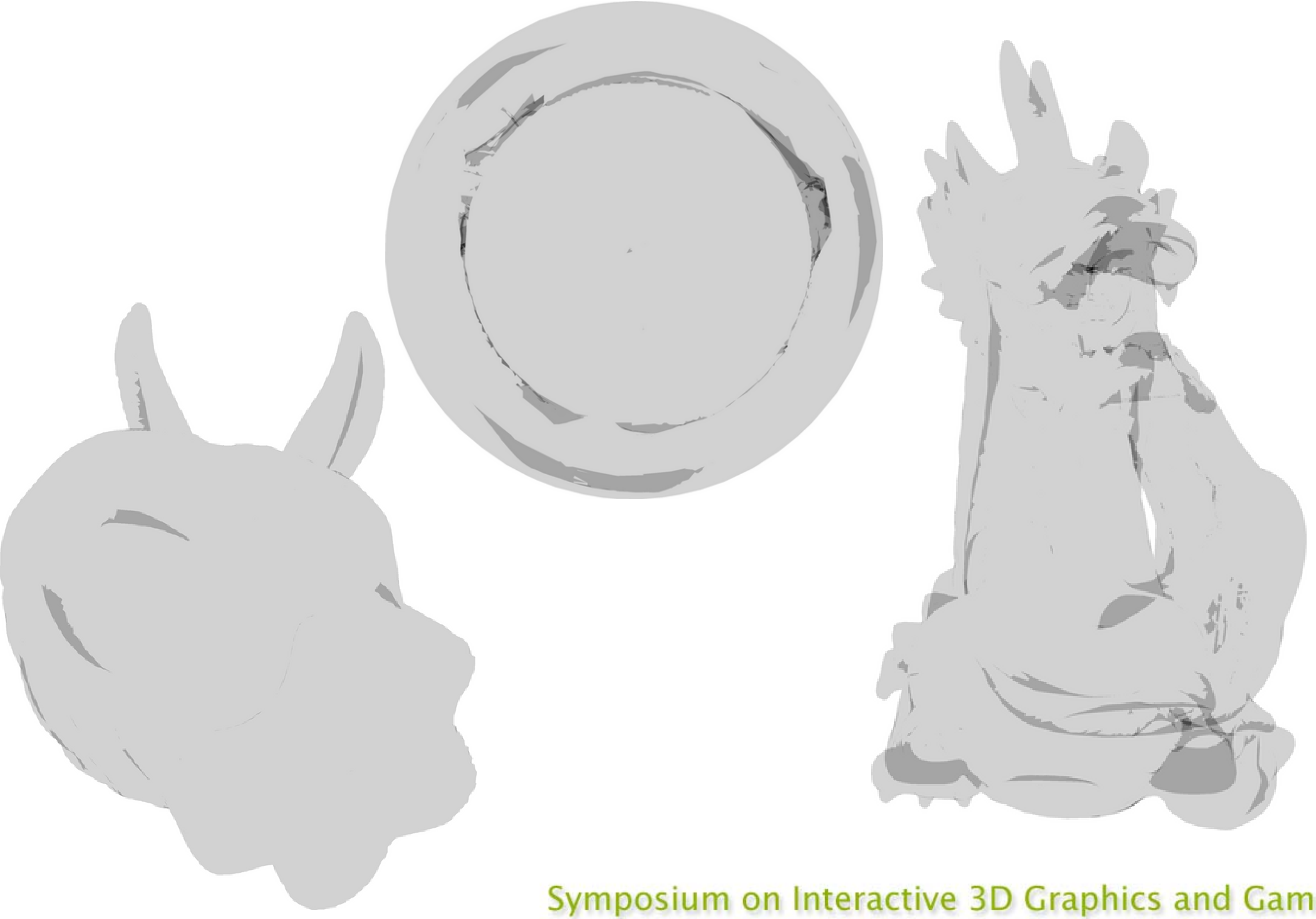
# Cluster ordering (demo)



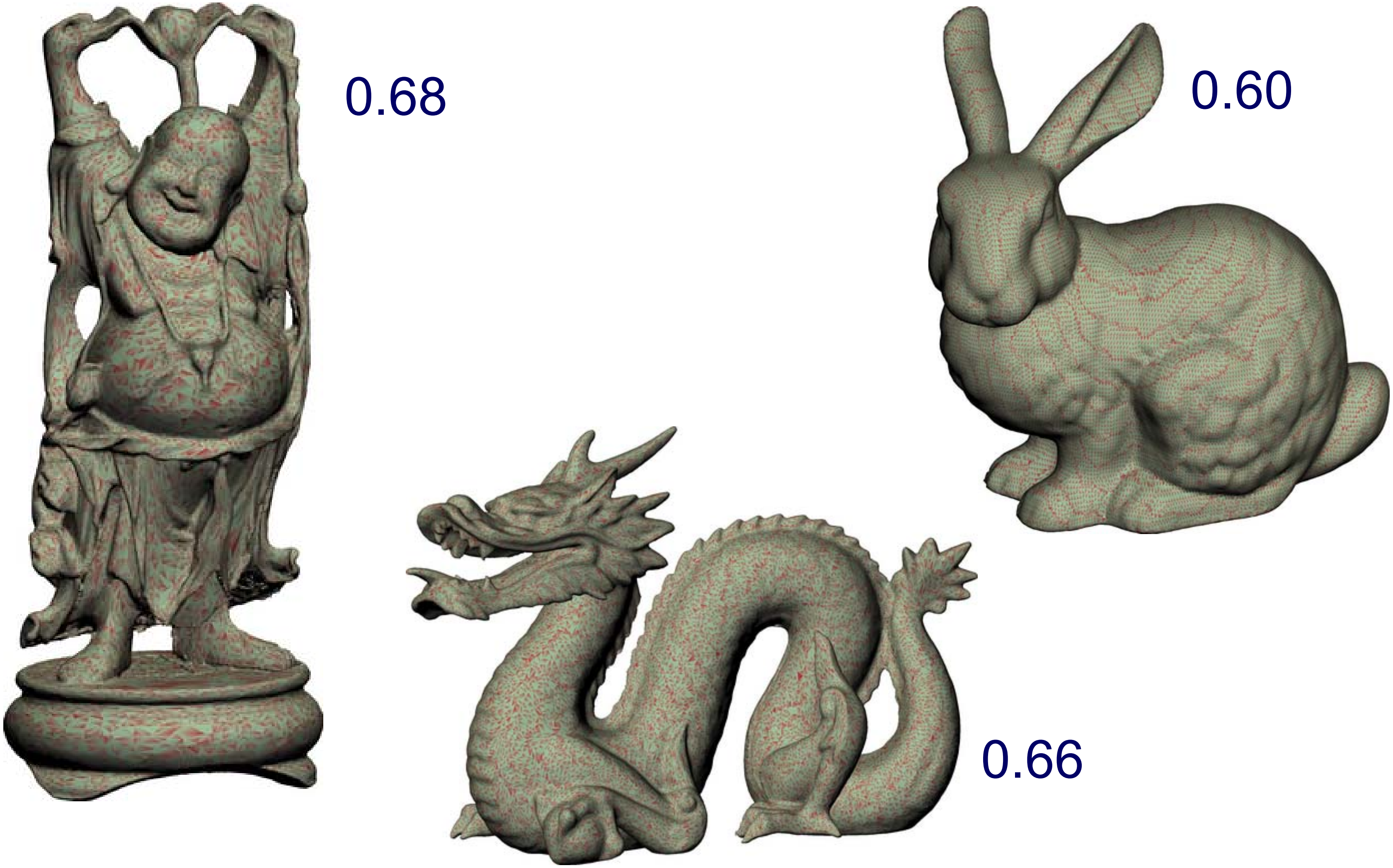
# Overdraw (full)



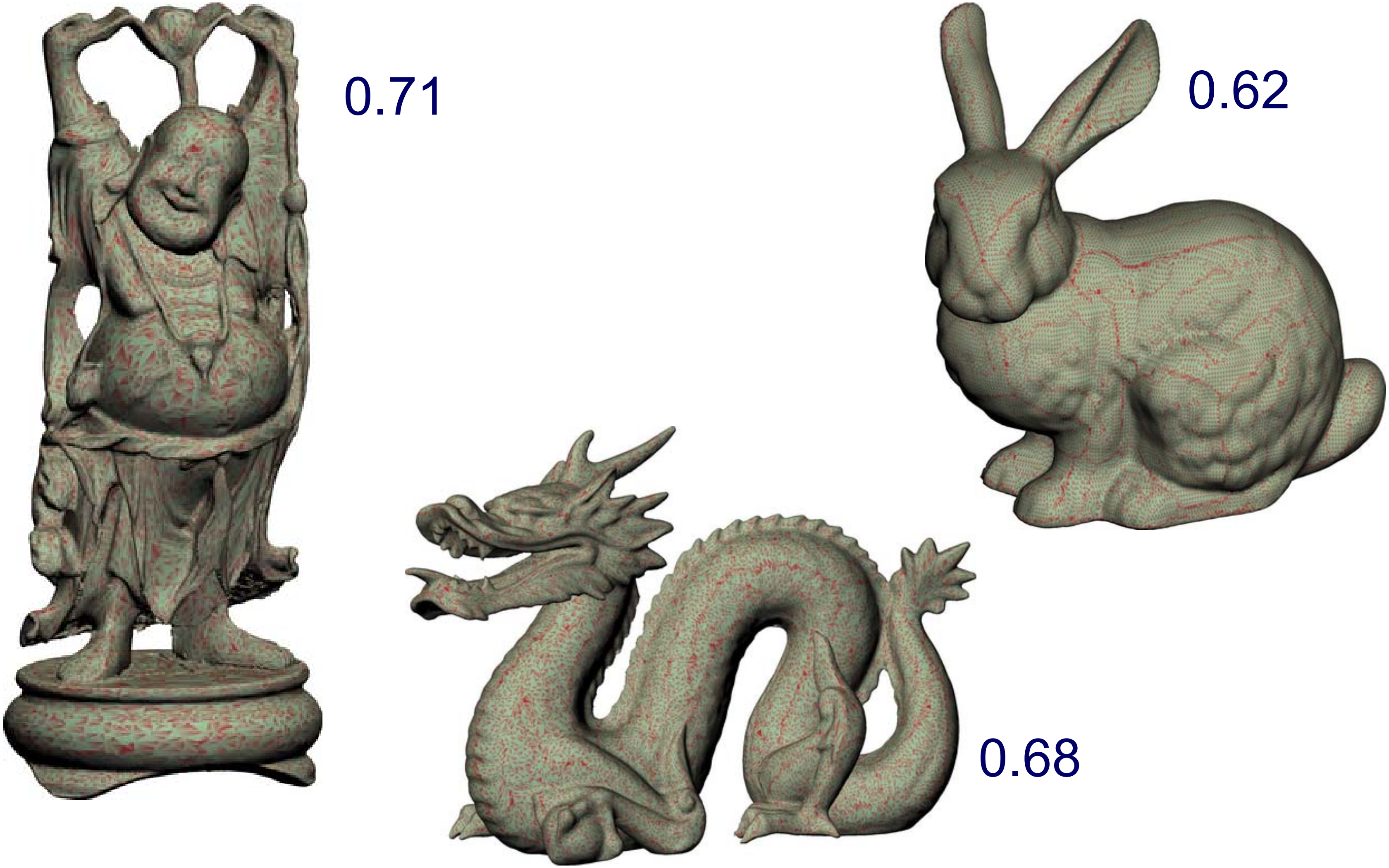
# Overdraw (clustered)



# Cache misses (full)



# Cache misses (clustered)



# Expensive shader demo

# Conclusions

- Greatly reduces overdraw
- Preserves state-of-the-art locality
- Completely automatic
- No run-time requirements
- No reason not to use! ☺