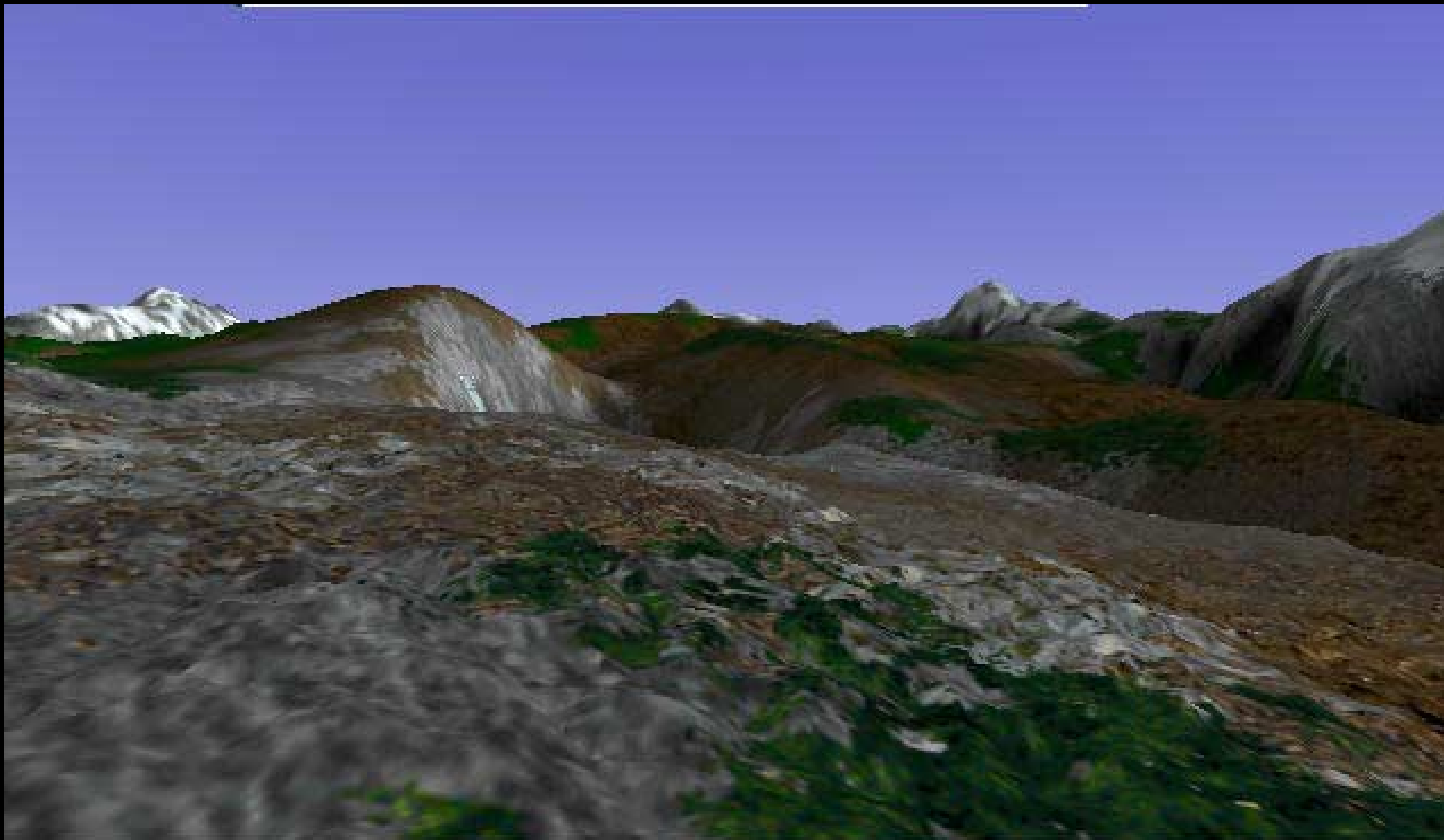
Terrain Rendering Research for Games

Jonathan Blow
Bolt Action Software
jon@bolt-action.com

Lecture Agenda

- Introduction to the problem
- Survey of established algorithms
- Problems with established algorithms
- How we solved these problems
- Future work

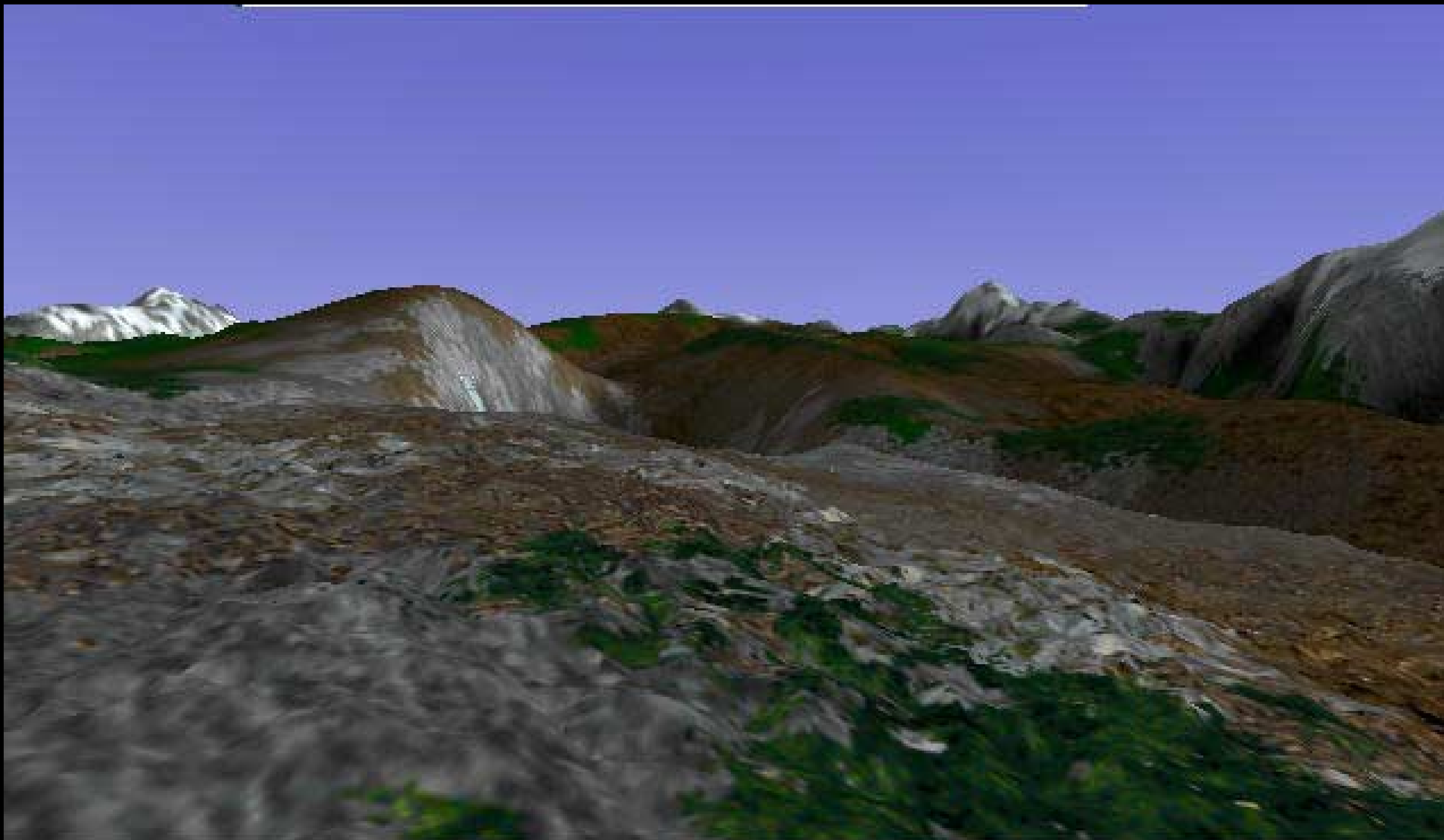


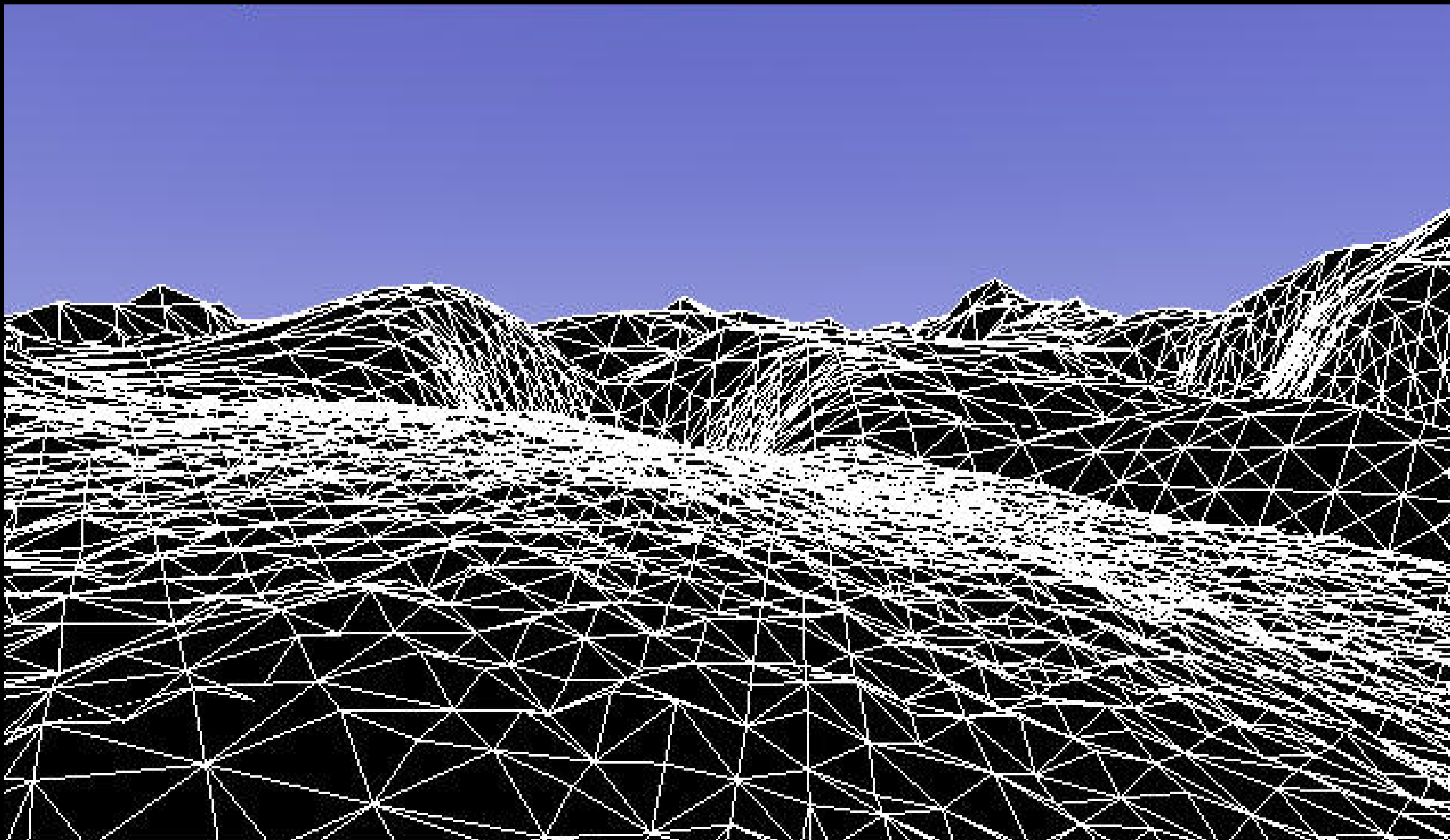
Gameplay goals for a terrain engine

- Large enough to travel around for hours
- Detailed when seen at a human scale
- Dynamic modification of terrain data
- Runs at high, **stable** frame rates
- Need fast rotation of viewpoint

Technical goals to support gameplay

- Level-of-detail management (static or continuous?)
- A lot of polygons (2^{31} in un-reduced terrain, 70,000+ in a given tessellation)
- Rendered polygons economically represent the terrain
- Near-field detail





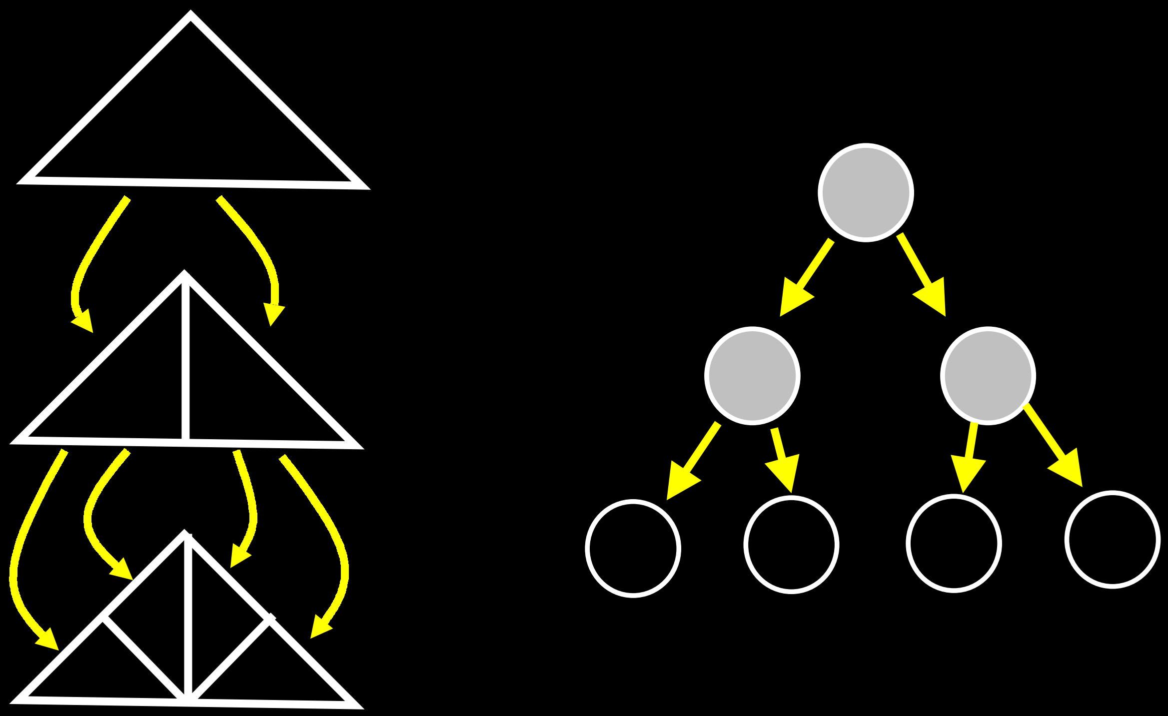
Chief terrain CLOD papers:

- Lindstrom-Koller (SIGGRAPH ‘96)
- ROAM (M. Duchaineau *et al*, IEEE Visualization ‘97)
- Rottger *et al*

Geometry management

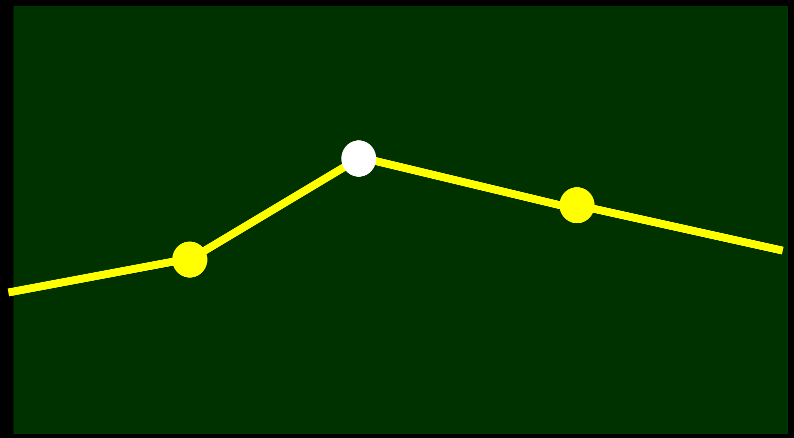
- Our previous games used variants of Lindstrom-Koller
- We wanted to switch to ROAM for increased versatility and efficiency.
- We’ll now survey both of these systems.

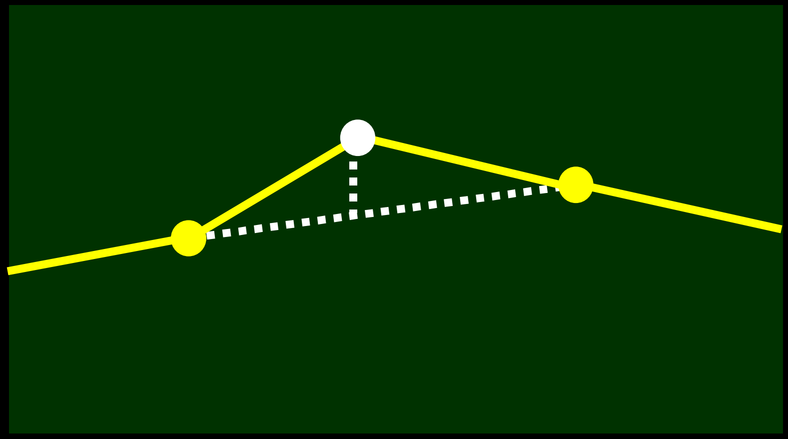
Lindstrom-Koller and ROAM both use a binary triangle tree.



Lindstrom-Koller algorithm

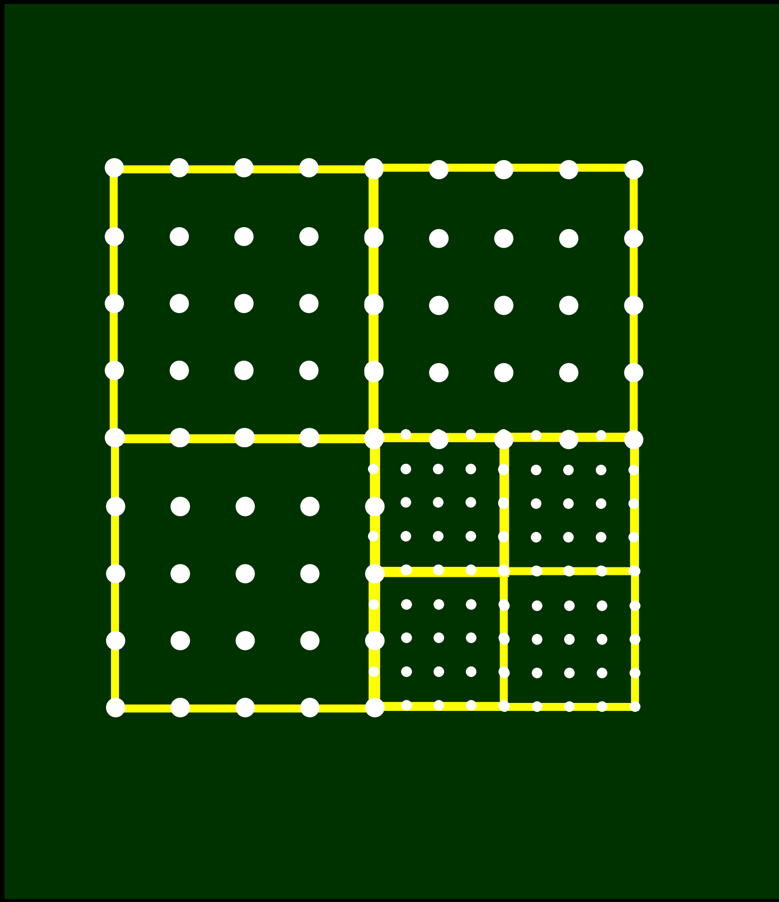
- Operates bottom-up on a height field.
- Considers vertex-removal error projected to the viewport.
- If the projection is small, we can remove the vertex.





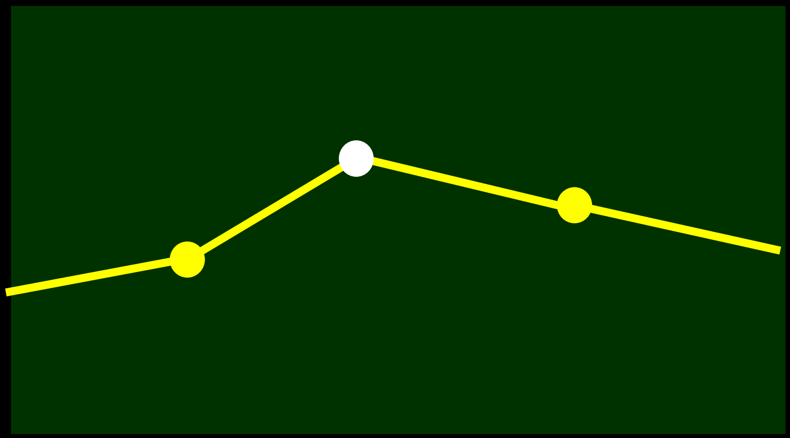
Lindstrom-Koller: frame coherence

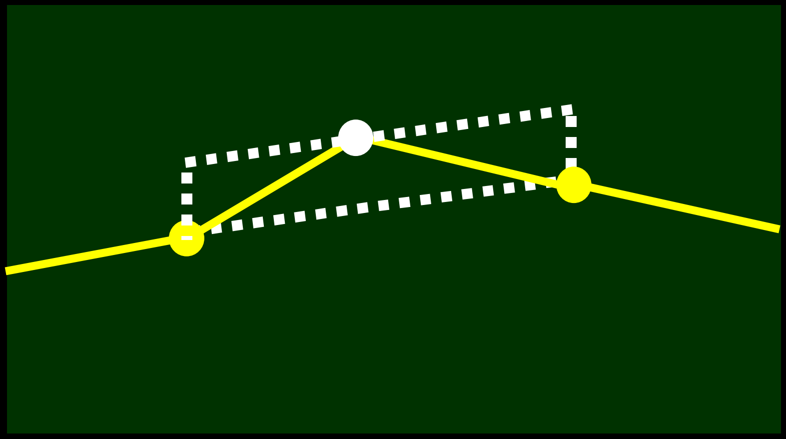
- Vertices are grouped into blocks, sorted by error value.
- Reduces the number of vertices evaluated each frame.



ROAM algorithm

- Operates top-down on bounding volumes.
- Considers the projection of each bounding volume to the screen.
- If the projection is large, we subdivide the volume.



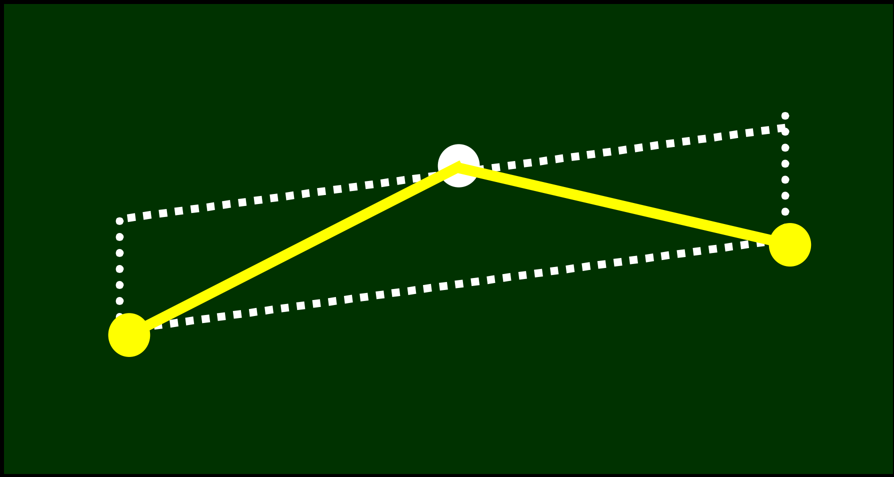


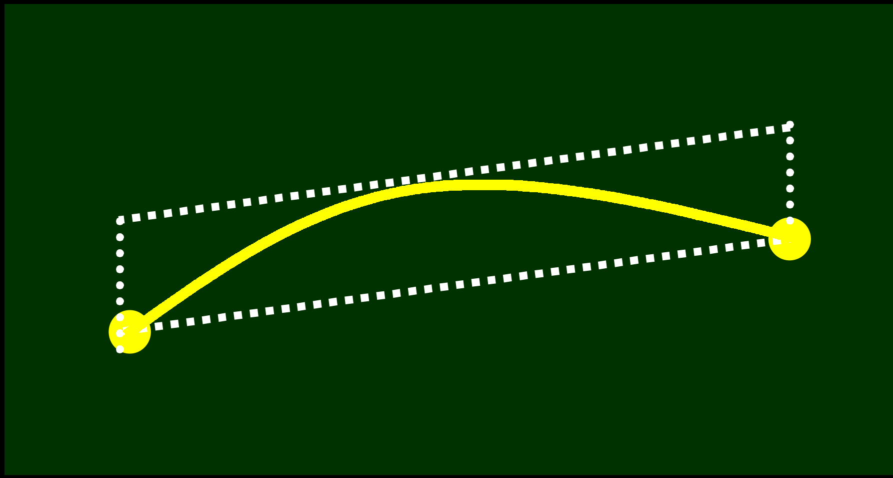
ROAM: frame coherence

- Two priority queues: split queue, merge queue
- Highest-priority wedges are split and merged to maintain equilibrium triangle count.
- Priorities modified according to viewpoint motion.

Why we were so excited about ROAM

- Because it's top-down, it does not dictate the form of your terrain data.
- We could use terrain consisting of Bezier patches with displacement maps.





The binary triangle tree is an excellent tessellator for curved surface terrain.

- Most people who work on Bezier surface terrain use rectangular subdivision.
- BTTs provide easier crack fixing and tighter resolution adaptation.
- The height map guys and the Bezier patch guys just don’t talk to each other?

We implemented ROAM

- It ran slowly -- didn’t scale.
- Spent a long time trying to optimize it.
- Other game developers have had similar problems.
- Games that use ROAM-style algorithms usually throw away the frame-coherence portions. This results in “split-only ROAM”.

The Evil Feedback Loop

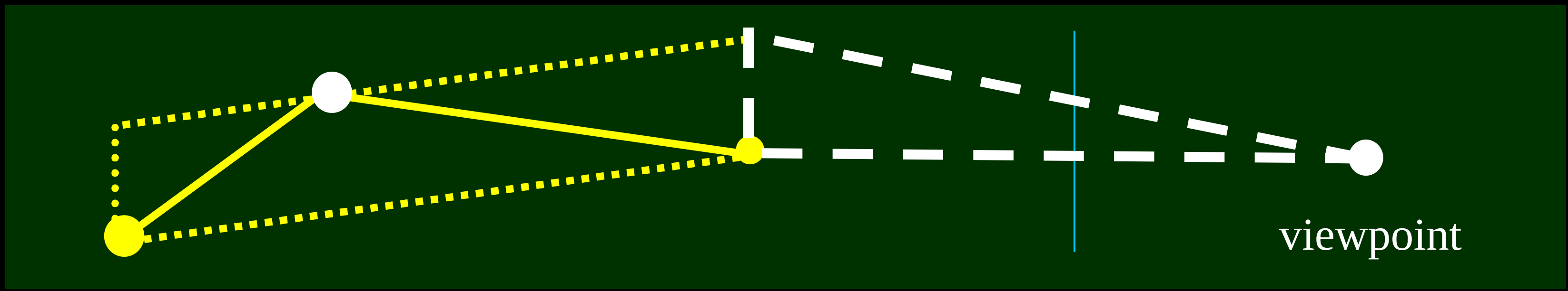
- The longer you take to simulate a frame, the further the viewpoint moves in that frame.
- Thus the algorithm has to do more work next frame: longer simulation time.
- There’s a catastrophe point where you can no longer keep up with real time: frame rate plummets toward 0.

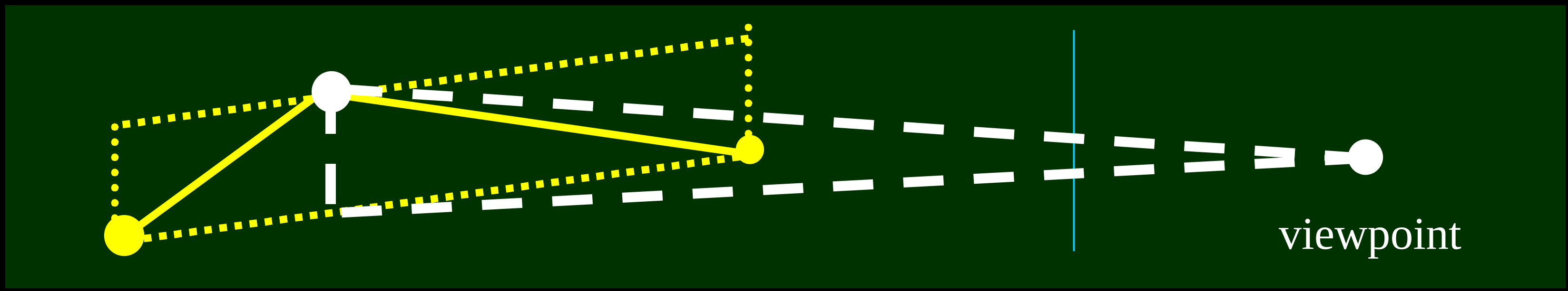
Minor improvements to increase ROAM tessellation accuracy

- Separate wedge ascent and descent
- Child-volume bounding versus contained-vertex bounding
- We were able to decrease polygon output by 40% for our data set.

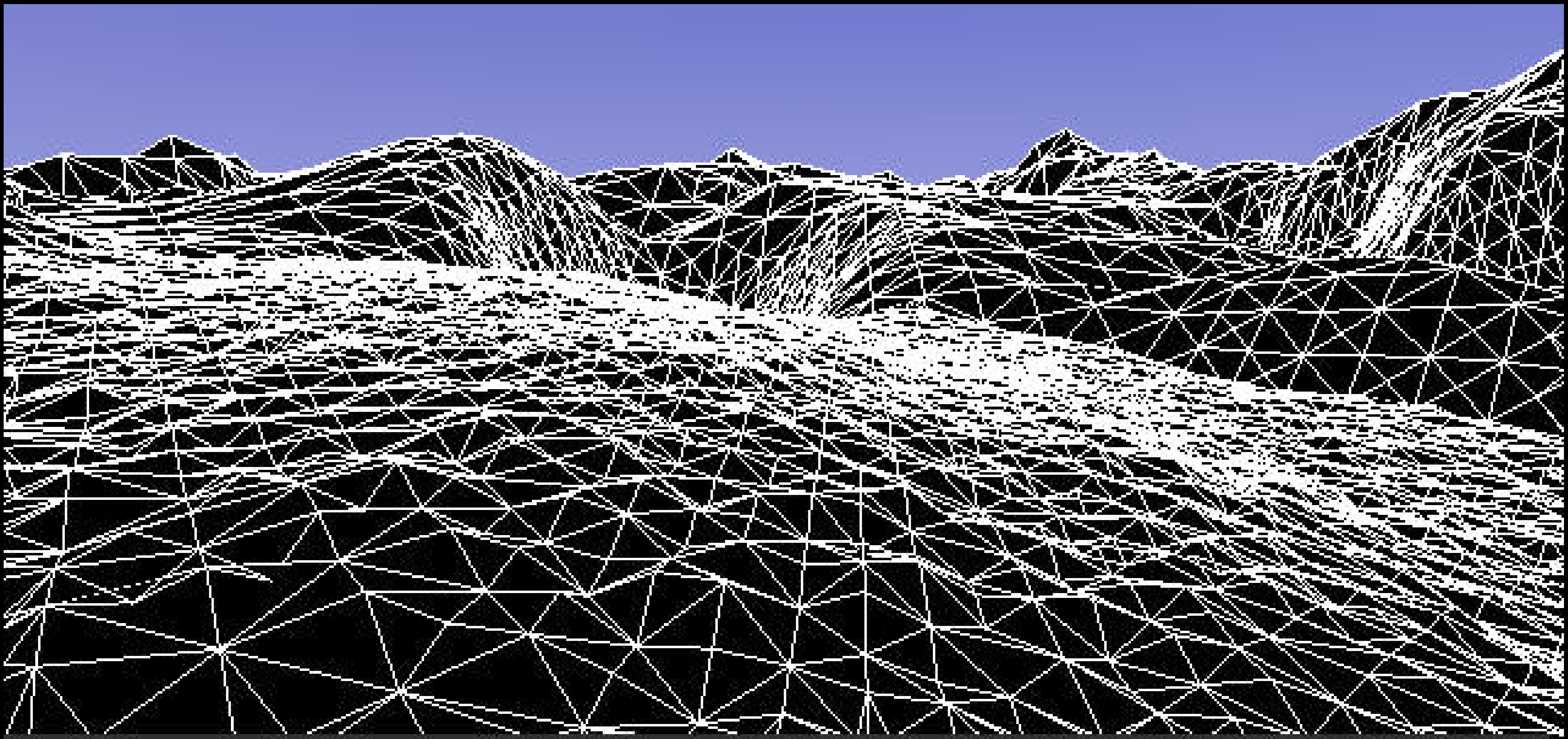
The problem with top-down terrain rendering systems

- The bounding volumes hide information about the position of the maximal error.
- In making pessimistic assumptions about the projection, they sacrifice tess. efficiency.

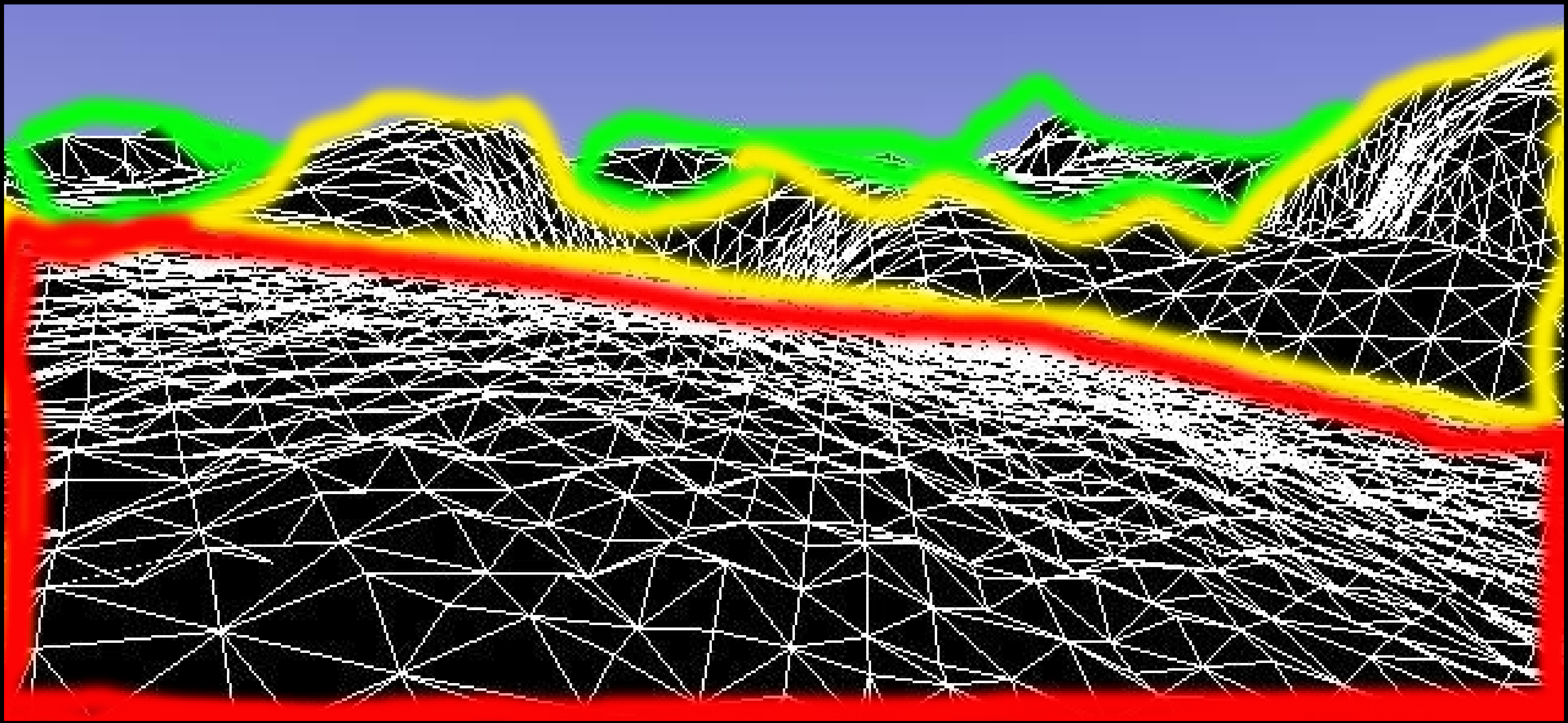




In an LOD’d scene, polygons tend to be roughly the same size in screen pixels.



A large percentage of polygons are small and close (50%? 60%?)



ROAM hindered by the basic nature of LOD

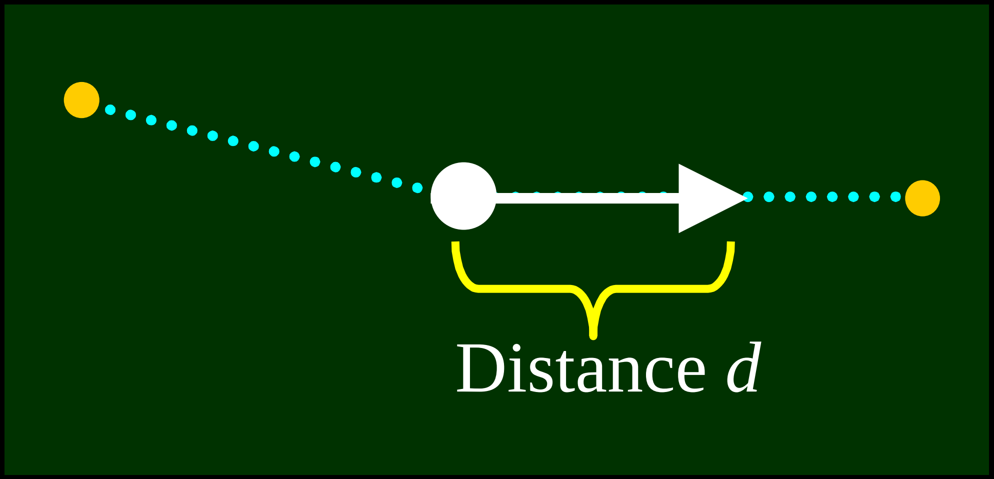
- We cannot get good priority bounds on polygons that are nearby.
- Polygons that are nearby comprise 50% of our tessellation.
- This hurts.

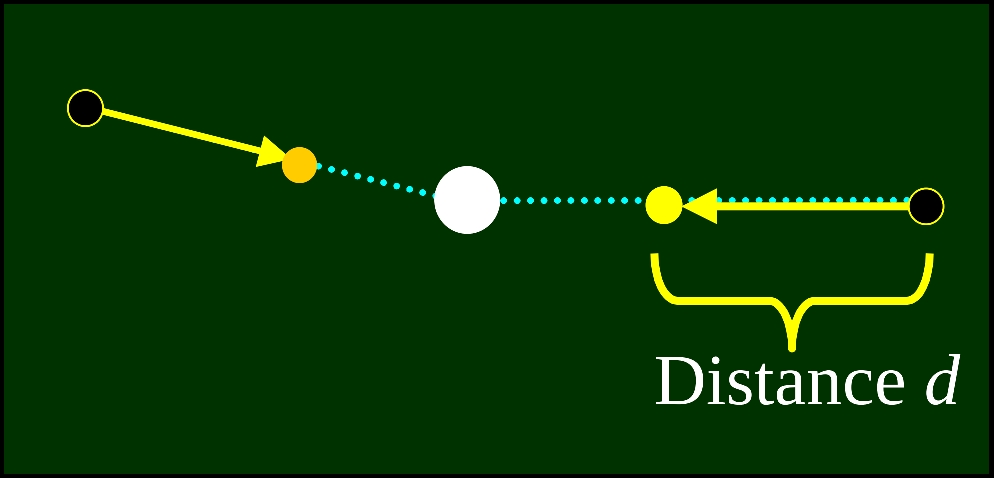
ROAM's running time

- The ROAM paper states it's O(n),
  n = number of LOD operations per frame.
- ROAM priority queues perform sorting.
- ROAM is actually O(mlogm),
  m = number of triangles in tessellation.

ROAM's lack of directionality is a problem.

- We don't know where wedges are relative to the viewpoint; only how "distant" they are.
- Priorities of all wedges decrement at the same rate… even wedges you are moving away from.





General problem with established CLOD algorithms: Weak correlation

- The algorithms use 1-dimensional correlation between vertices to gain speed (within-block sorting in LK, sorted priority queues in ROAM)
- They spend CPU resolving ambiguities in this 1D ordering.
- We could do better correlating in 3D.

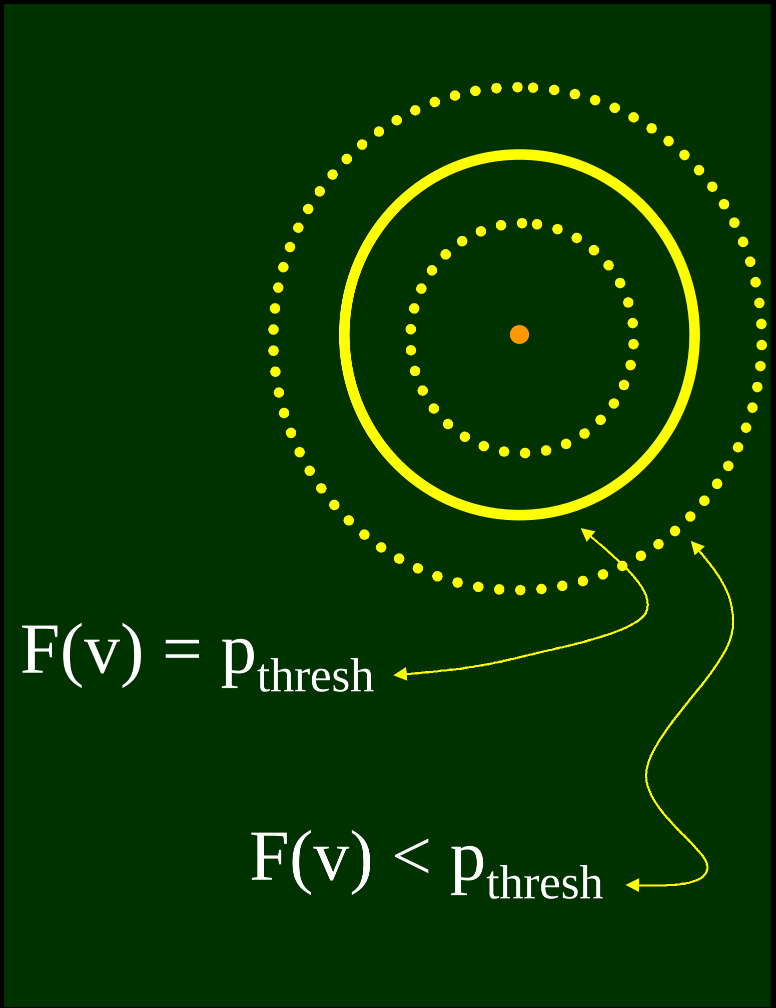
$$F_n(\mathbf{v}) = p_n$$

$$p_n < p_{thresh}?$$

- F maps the 3-dimensional argument v into the one-dimensional result p
- There are 3 dimensions’ worth of information in F but we see only the 1D shadow of that in p.
- Every point in p represents an infinite number of points in F aliased together.

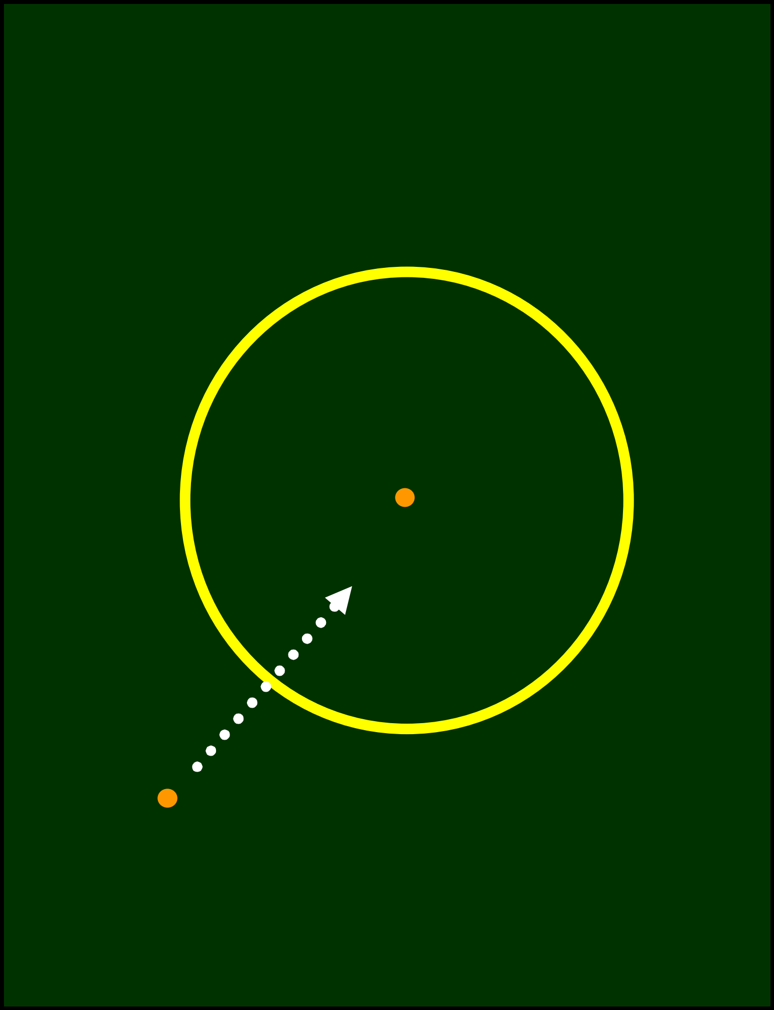
Changing the way we think about the projected error.

- Rather than evaluating $F_n(v)$, we look at F_n itself.
- The set of points for which $F_n(v) = p_{thresh}$ forms a boundary surface in 3D space.
- This is an *isosurface* of the implicit function F_n.



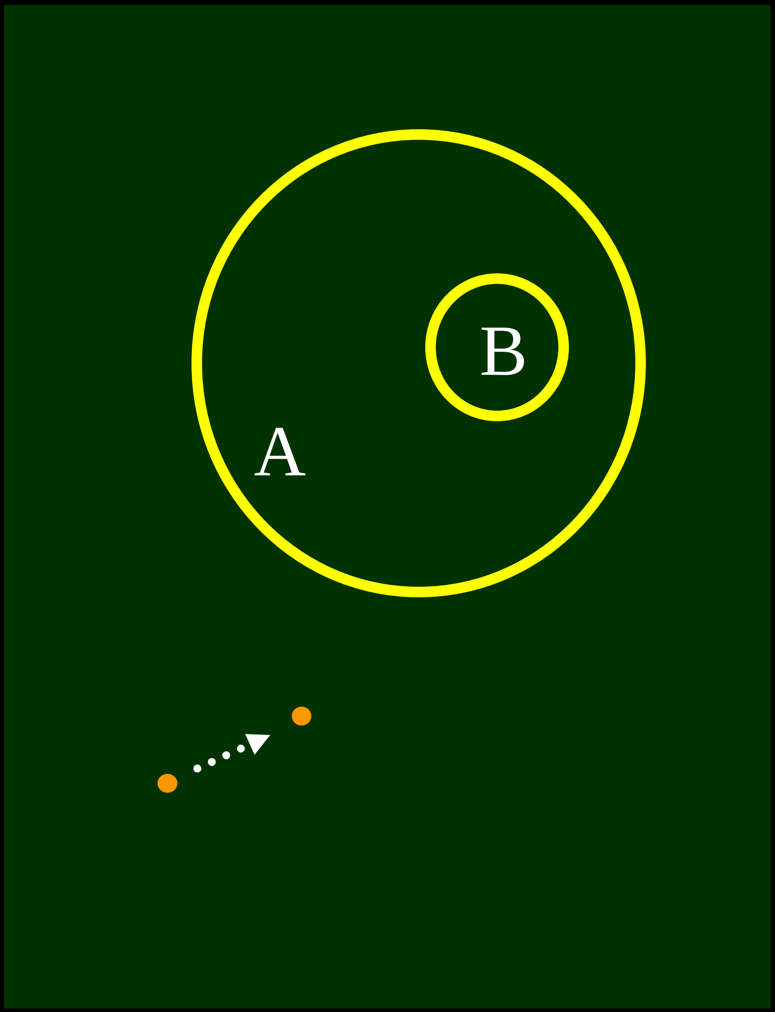
Isosurface LOD testing

- When the viewpoint crosses into an isosurface, enable the vertex.
- When the viewpoint crosses back out, disable the vertex.



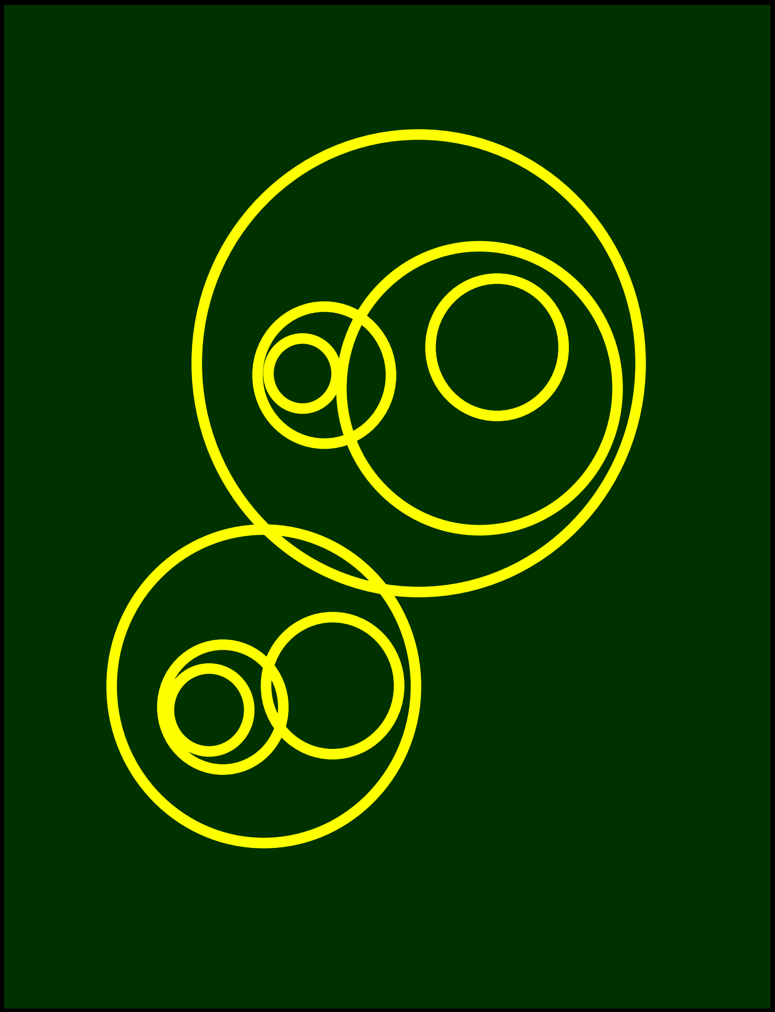
How we gain efficiency

- If B is contained in A, the viewpoint cannot enter B without first crossing A.



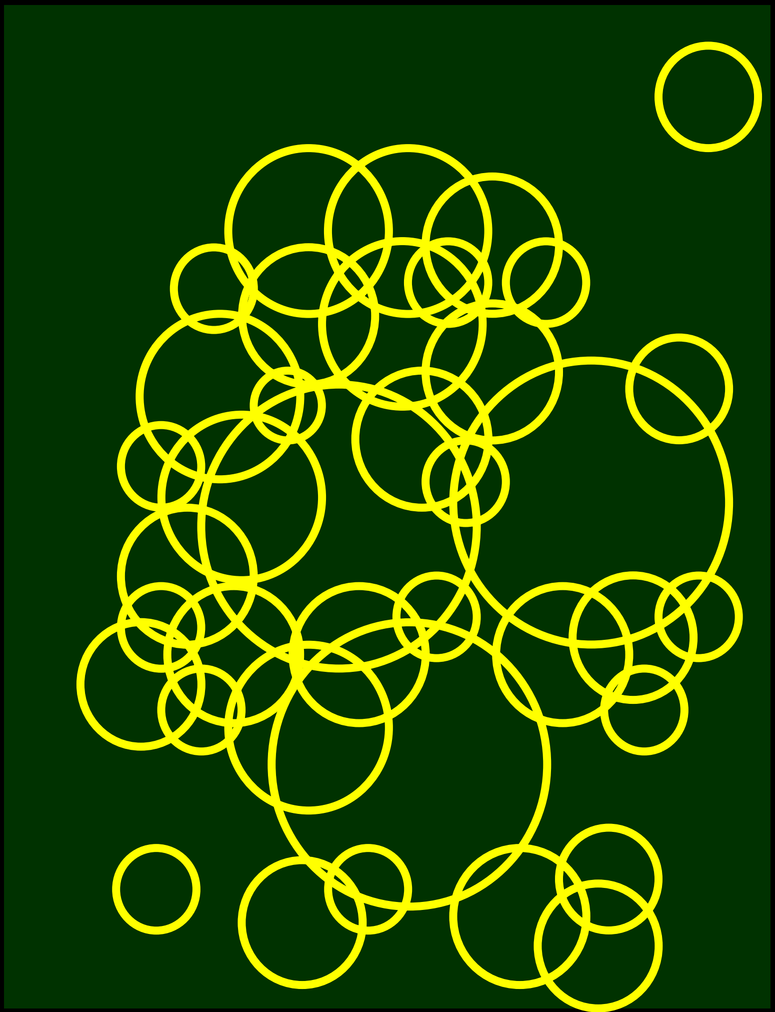
How we gain efficiency

- We store the isosurfaces in a tree. We only descend into nodes when the viewpoint crosses an isosurface.
- Statistically, terrains will exhibit a lot of natural hierarchy.
- Split tree, merge tree



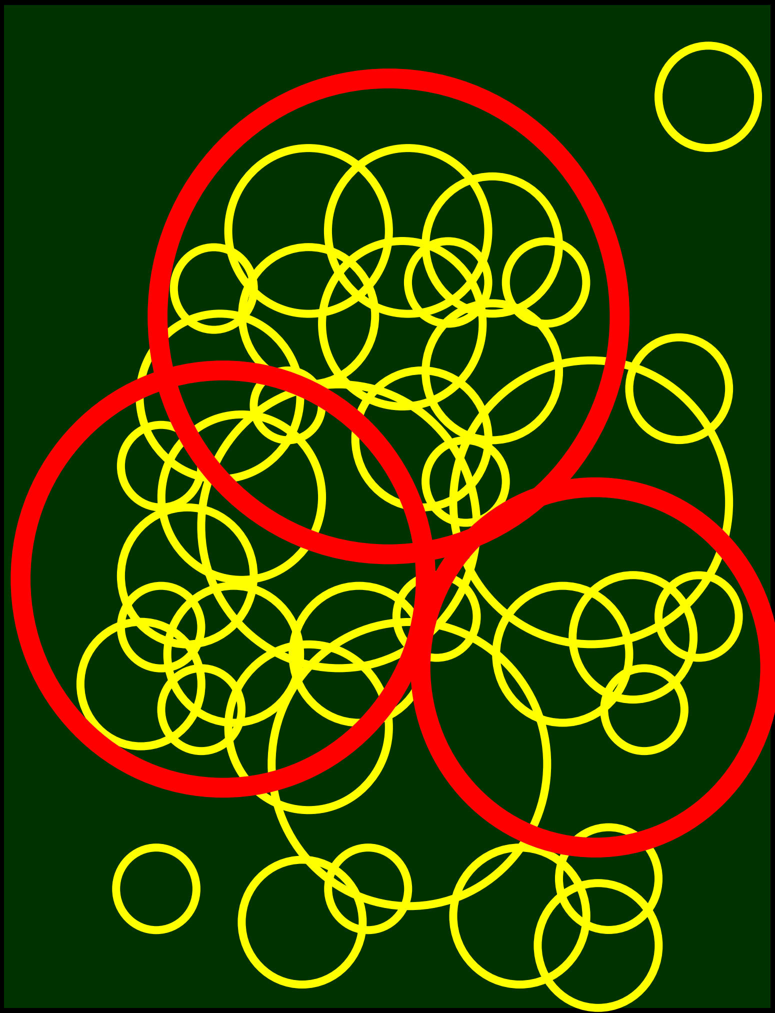
Clustering

- At the root level of the isosurface tree we will have thousands of intersecting surfaces.
- We introduce extra bounding volumes to cluster these nodes together.



Clustering

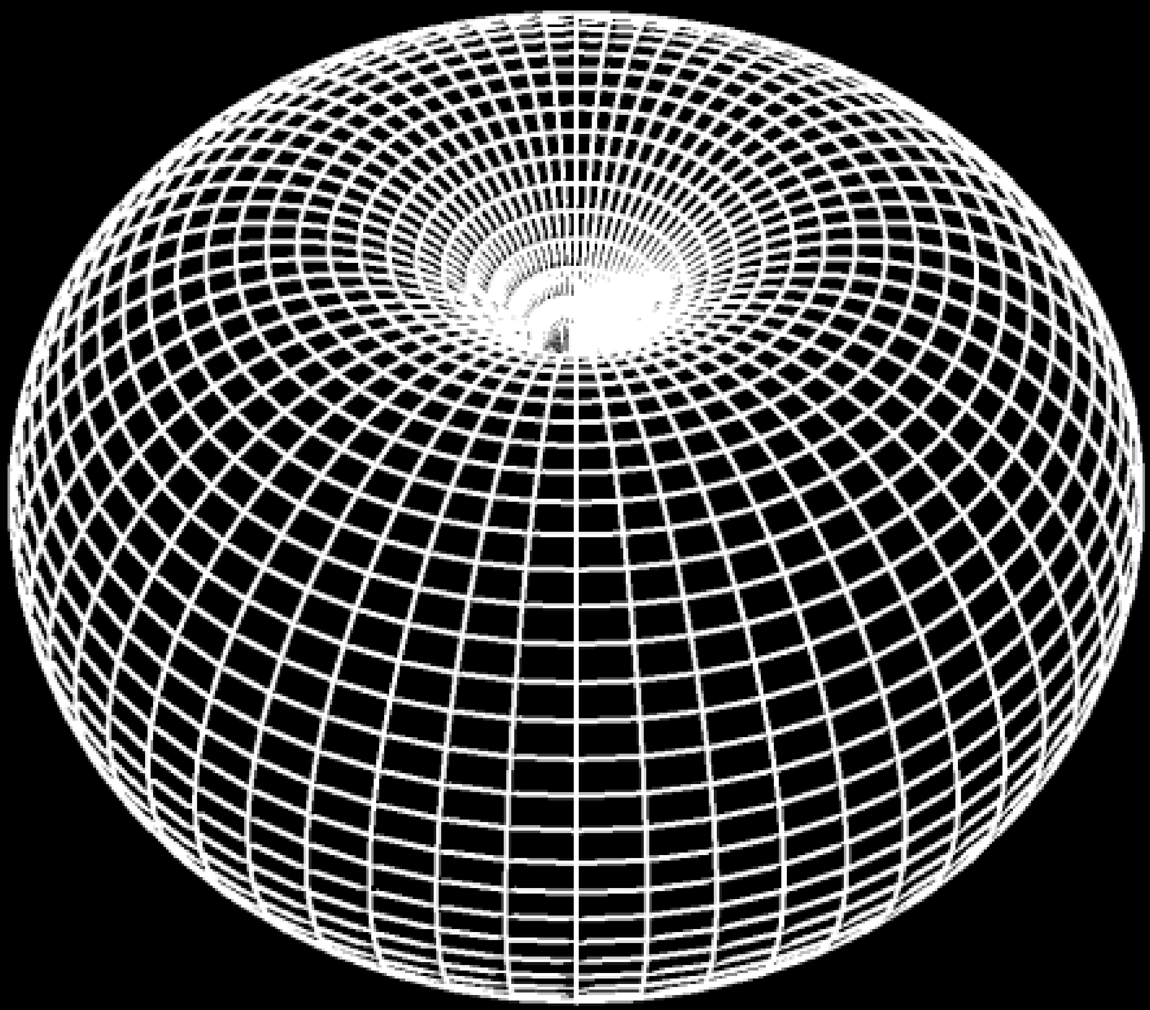
- At the root level of the isosurface tree we will have thousands of intersecting surfaces.
- We introduce extra bounding volumes to cluster these nodes together.



Performance

- The number of node traversals is O() the number of LOD operations required that frame, with a slight overhead for cluster nodes. "You only pay for what you get (mostly)".
- To make things extra speedy, we use spherical isosurfaces. However, the basic algorithm works with surfaces of any shape.

Lindstrom-Koller isosurface



Performance

- 2^{31} triangles, 50k in tessellation, 12k in frustum
- Quickly moving viewpoint.
- ROAM gave us 1fps, unstable performance.
- Lindstrom better at 8fps, moderately stable.
- Isosurfaces fastest at 30+fps, very stable.

Tessellation improvement

- 640x480 rendering, 3-pixel error
- ROAM-style wedges: 86284 triangles
- Direct isosurfaces: 56234 triangles

Tessellation improvement

- How low an error bound can we hit with a budget of 100,000 triangles?

- ROAM-style wedges: 2.75 pixels
- Direct isosurfaces: 2.17 pixels

Vertex buffers

- Pack vertices into an array that gets shipped off to the card.
- Expensive to create or modify; cheap to render.
- Difficult to use vertex buffers with dynamic LOD.

Using the isosurfaces to predict mean-time-to-modification

- We can make vertex buffers out of isosurfaces that don’t come very close to the viewpoint.
- We cluster these vertex buffers spatially so we can do reasonable frustum culling.
- Software buffers to take care of The Other 50%.

Changing the basic rendering method

- Now to render the scene, we begin at the root of the isosurface tree and work our way downward.
- We no longer use the binary triangle tree for rendering; so we phase it out entirely.

Future work

Future work:
The binary triangle tree?

- The binary triangle tree causes extra triangle splits to fix cracks.
- How much overhead does this produce?
- Some algorithms like Rottger’s and Ulrich’s triangulate quadtree blocks instead. They have differing crack fixing policies.
- What is the relationship between crack fixing policy, tessellation density, and scene quality?

Future work: A better error metric?

- Lindstrom-Koller uses vertical displacement to measure error. Garland/Heckbert use normal displacement.
- Is linear displacement even a good error metric? What are other (non-ad-hoc) options?
- We need a metric that judges the algorithm’s final output. (cf. PSNR)

Future work: Batched LOD operations?

- Our LOD decision-making is fast enough now to be negligible for our target detail levels.
- However our algorithm still suffers a catastrophe at high viewpoint speeds due to the aggregate cost of all the split/merge operations per frame.
- Some way to perform many splits/merges at once would be good.

Terrain Rendering Research for Games

Jonathan Blow
Bolt Action Software
jon@bolt-action.com