Fast View-Dependent Level-of-Detail Rendering Using Cached Geometry

Josh Levenberg
UC Berkeley
http://www.technomagi.com/josh/vis2002/

Outline

- Problem statement
- Review of ROAM, an existing level-of-detail algorithm
- Binary Triangle Trees (BTTs), the mesh representation in ROAM
- CABTT, my modification to ROAM for improved performance
  - Continuity
  - Error metric
  - Amount of aggregation
- Conclusion

Goal

- Render a large height field at interactive rates on consumer hardware
- For games, flight simulations
- Want to take advantage of modern and future consumer video cards double in speed every 6 months
- Height fields are memory efficient, 32 million triangles in 32 MB

Algorithm Comparison

Brute force	CABTT	ROAM
6,200k tris 1-2 fps geometry bound	150k tris 120 fps	60k tris 18 fps CPU bound

Speeds are for a 4k x 4k height field

Puget Sound data set from Lindstrom and Pascucci's web site

32 million triangles before view culling

1.7 GHz Pentium 4m, NVIDIA GeForce 4 440 Go

ROAM (Review)

- *ROAMing Terrain: Real-time Optimally Adapting Meshes* by Duchaineau et al. IEEE Vis 1997
- View-dependent level of detail
- Maintains a crack-free mesh using a *Binary Triangle Tree* (BTT)
- Incrementally modifies mesh from frame-to-frame
- Uses prioritized work queues to control work per frame

ROAM

Advantages

- Few triangles to achieve error bound
- Fine control over error threshold, time spent per frame

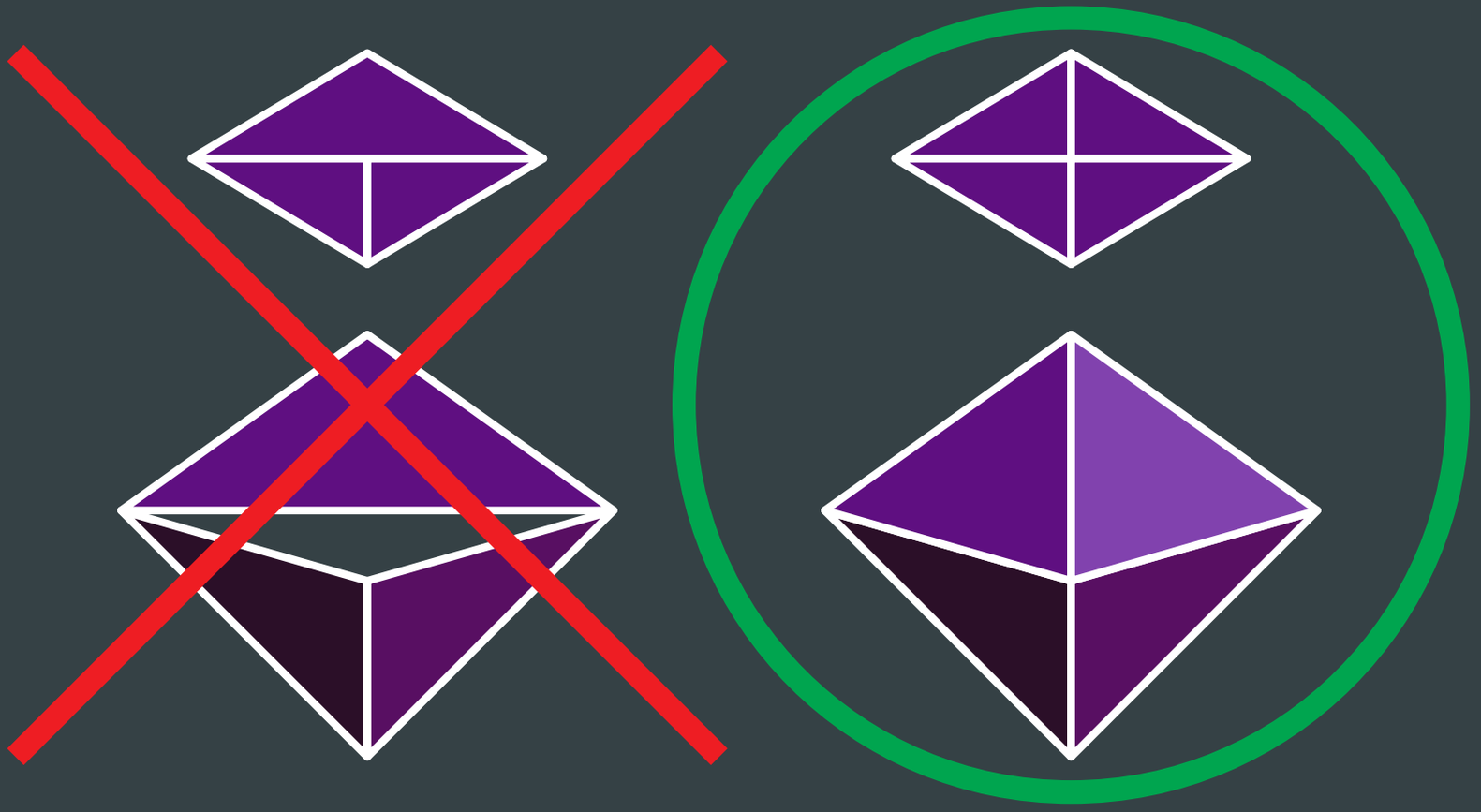
Disadvantages

- Lots of per-triangle processing: CPU bound
- Can’t cache on video card

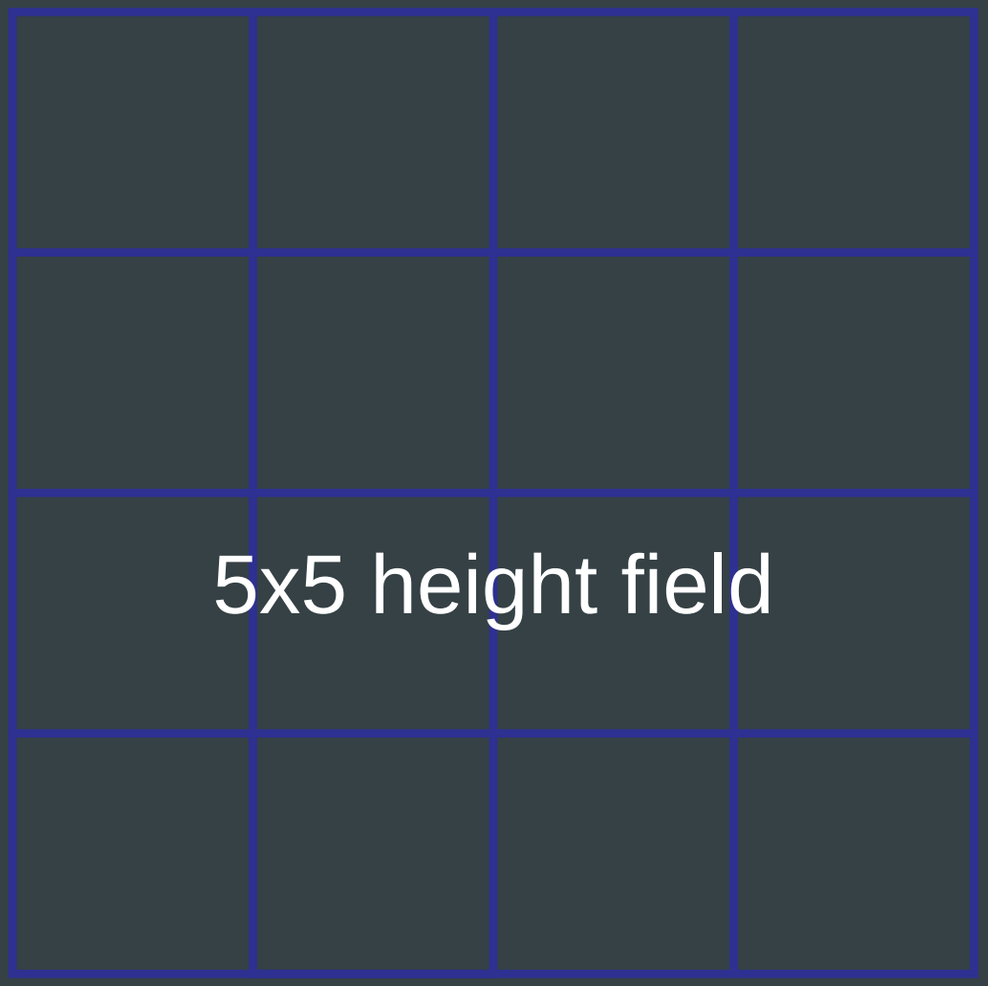
Binary Triangle Trees

- AKA: triangle bintrees, right triangular irregular networks, newest-vertex-bisection meshes, BTTs
- Usually for Terrain, but works with base mesh + displacement map

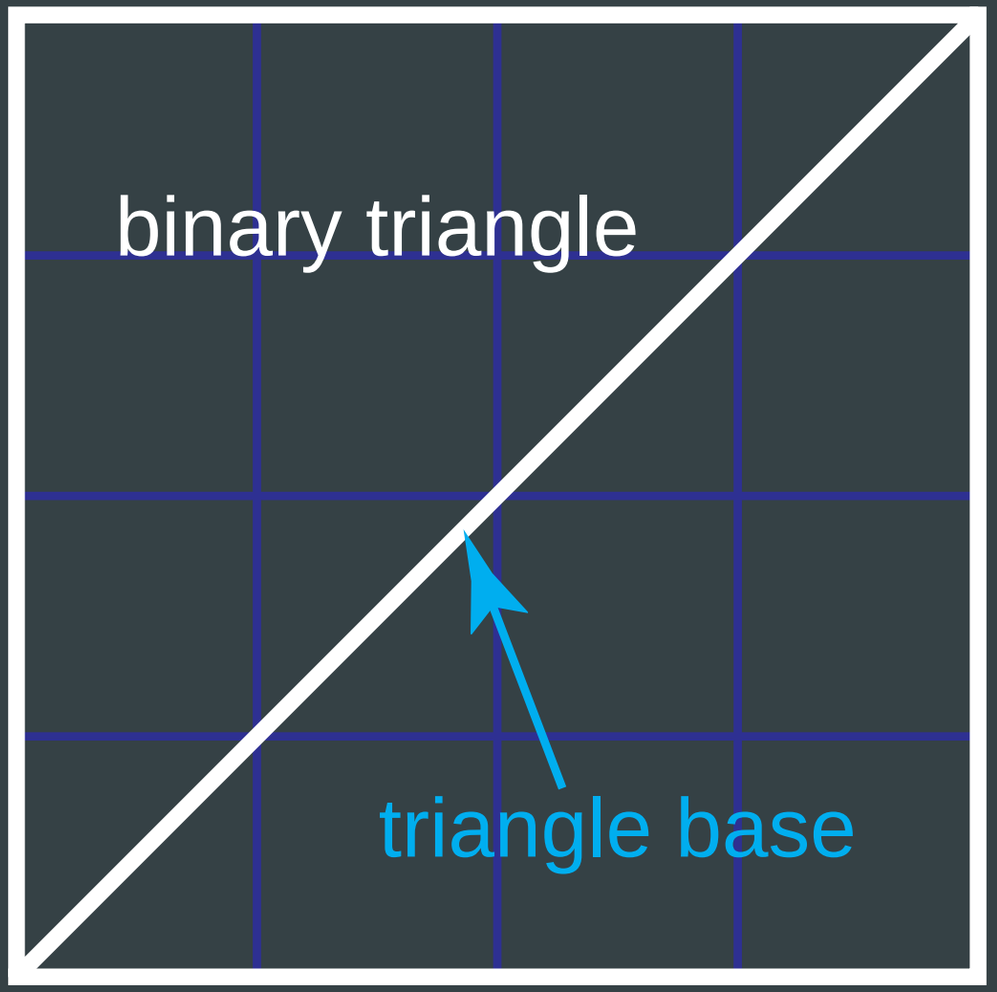
Don’t want T-Junctions



Example

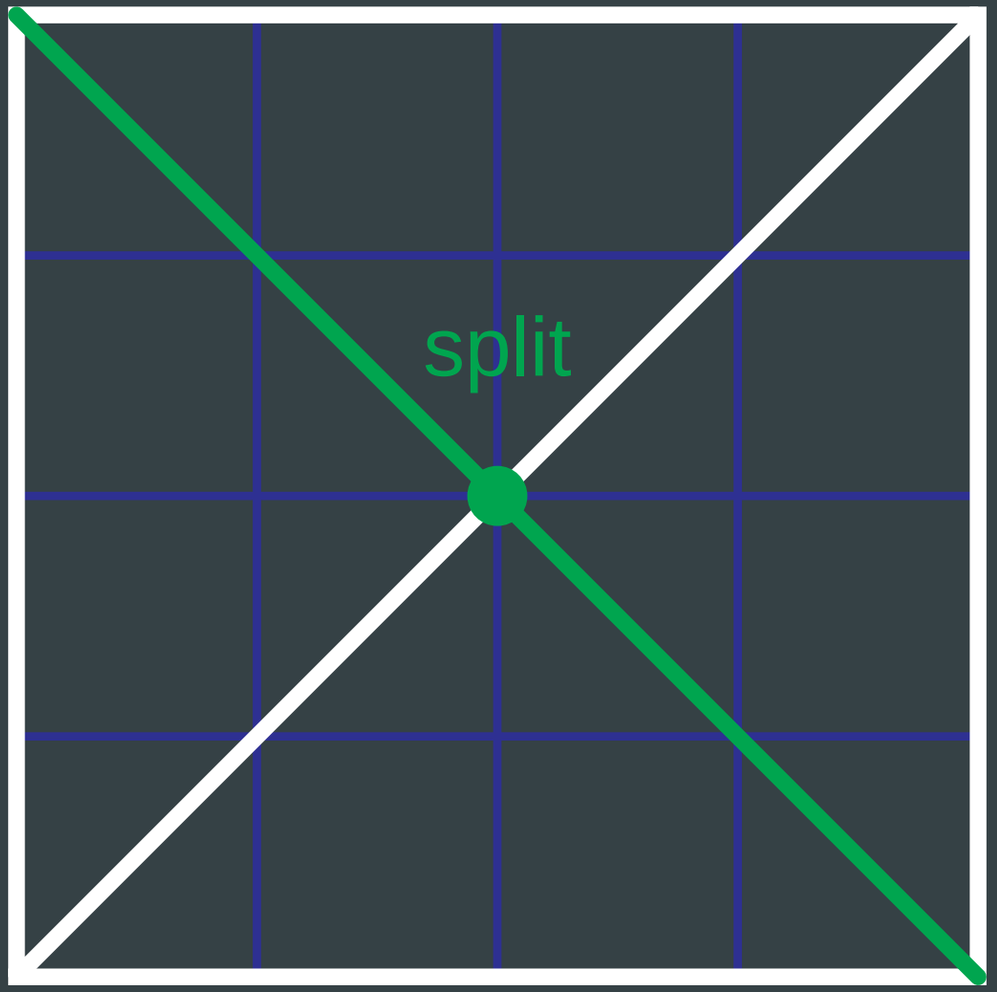


Diamond

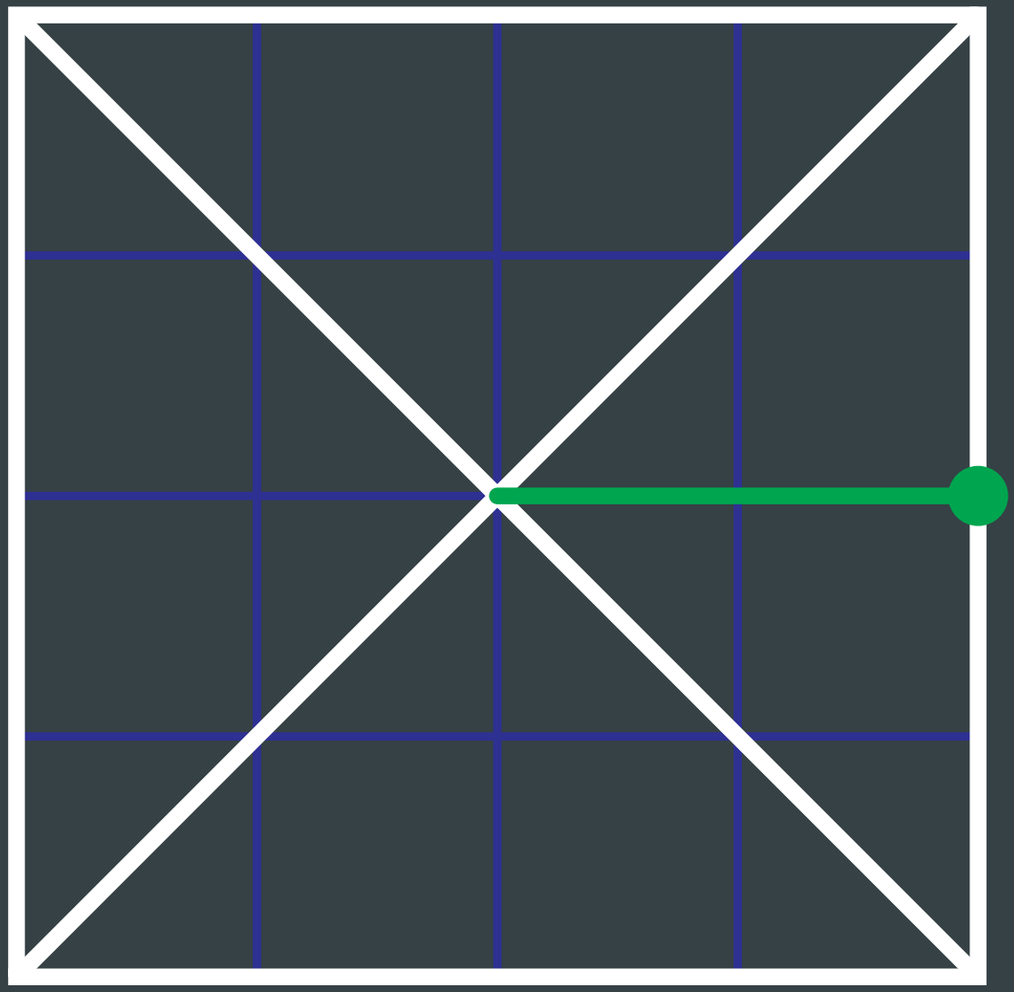


base mesh = initial triangles

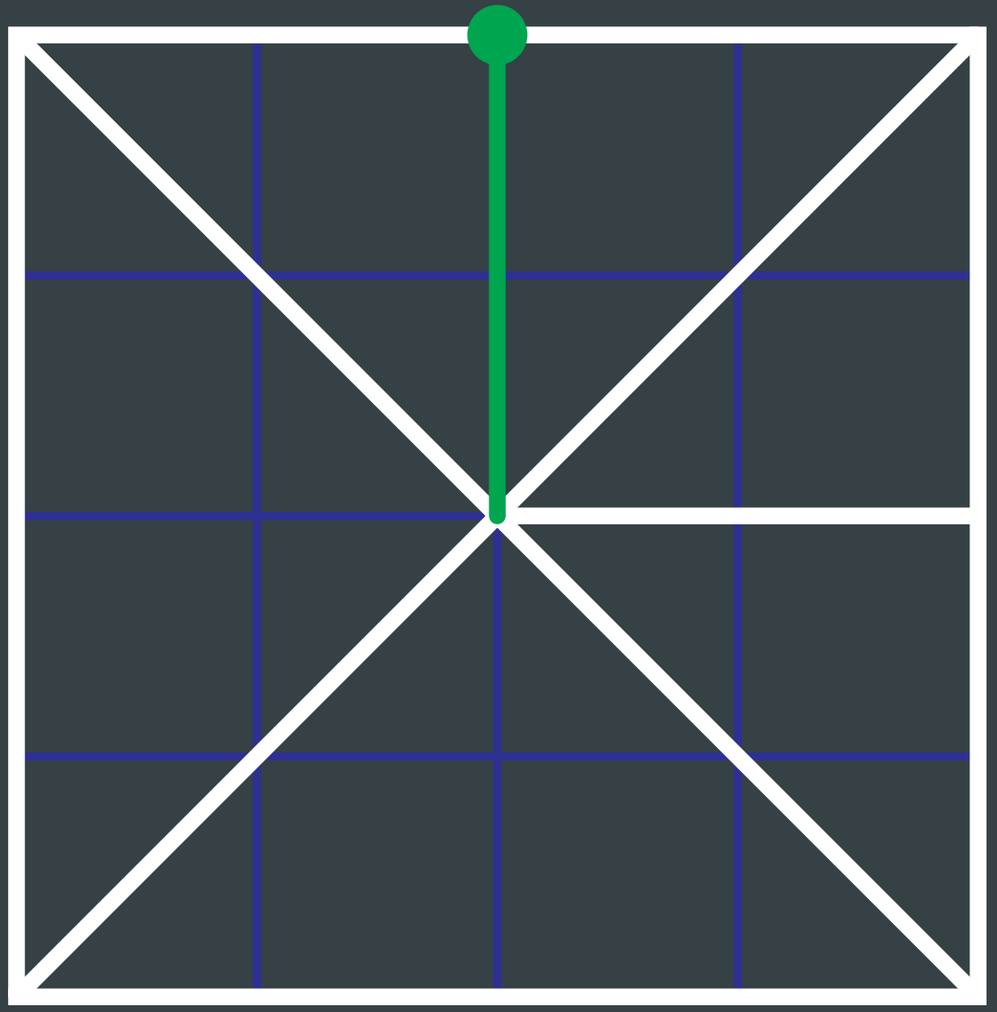
Split of diamond



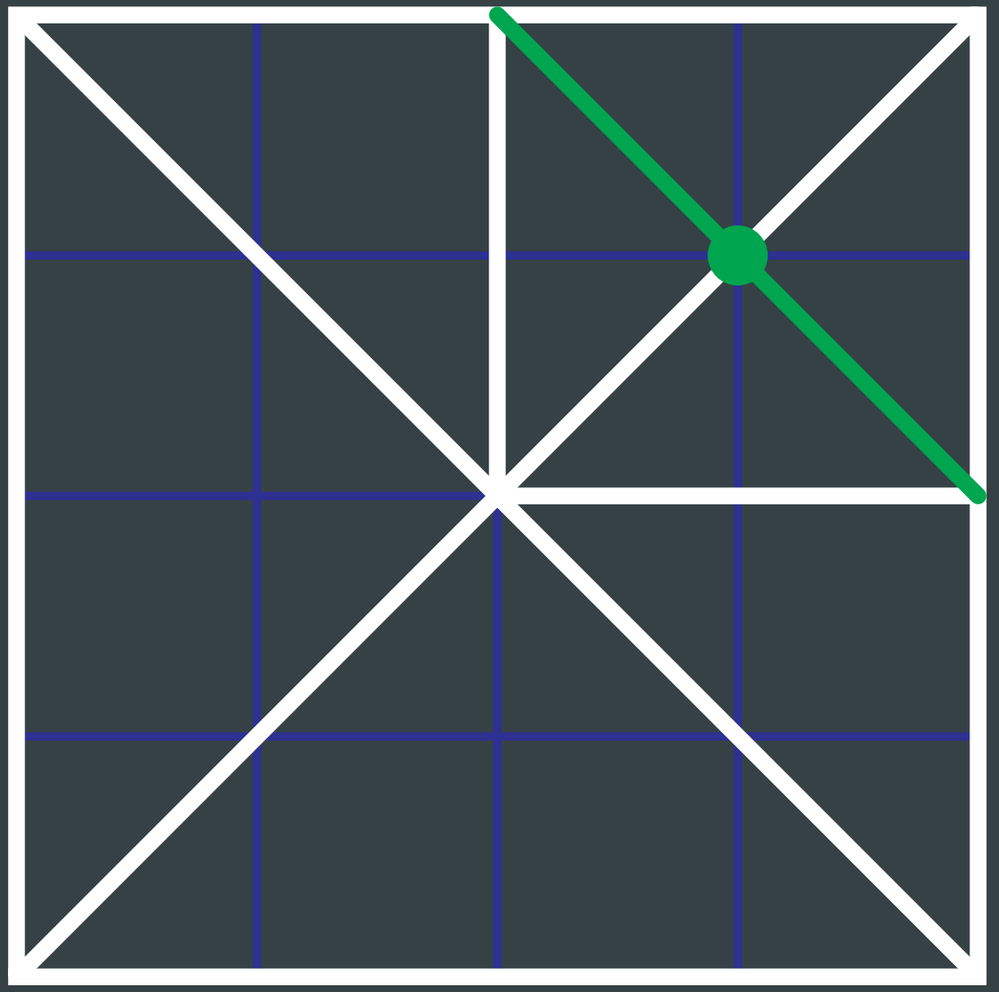
Split at edge



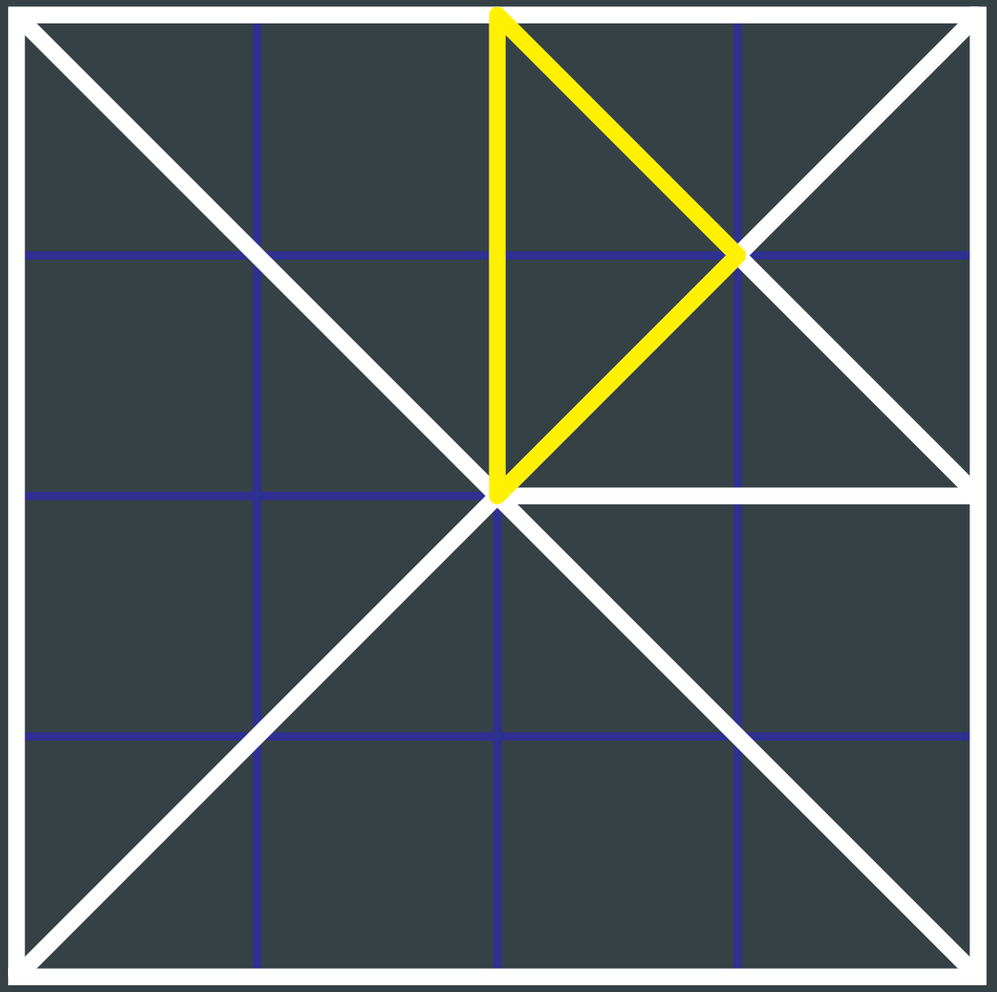
Another split



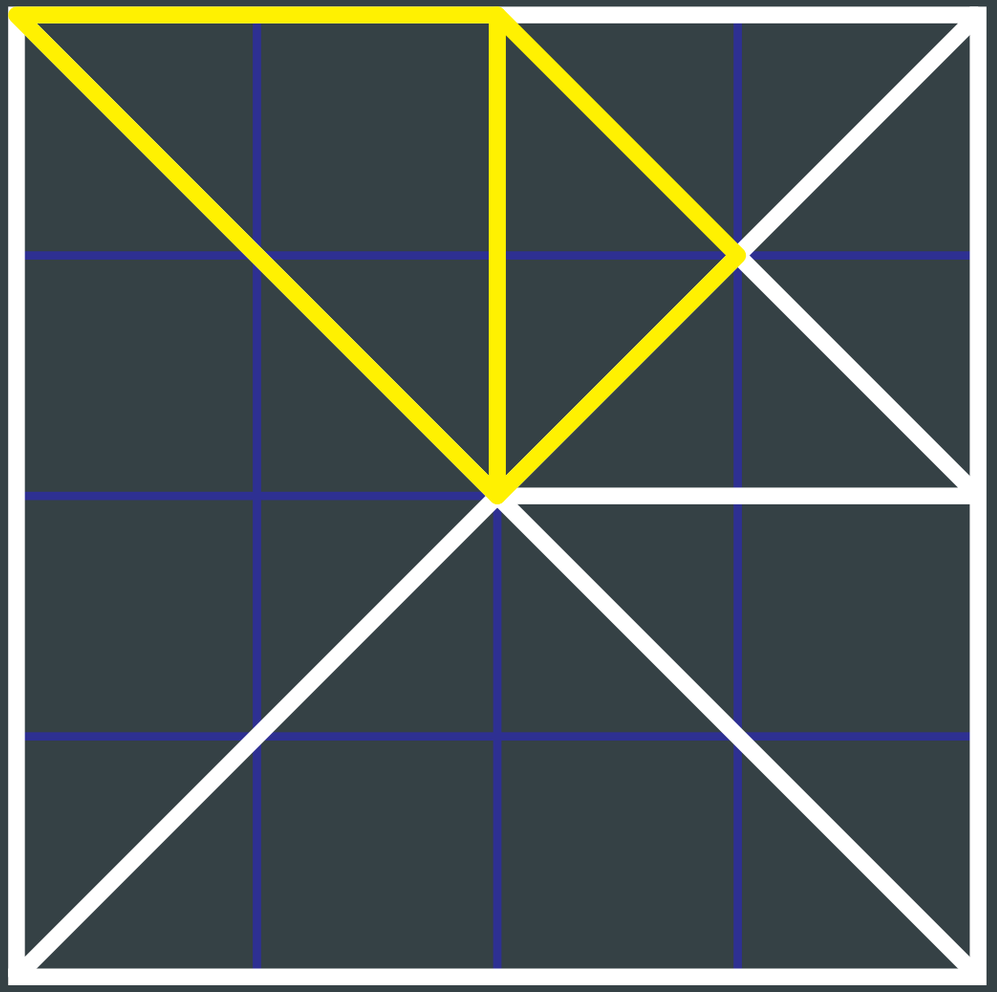
Another split



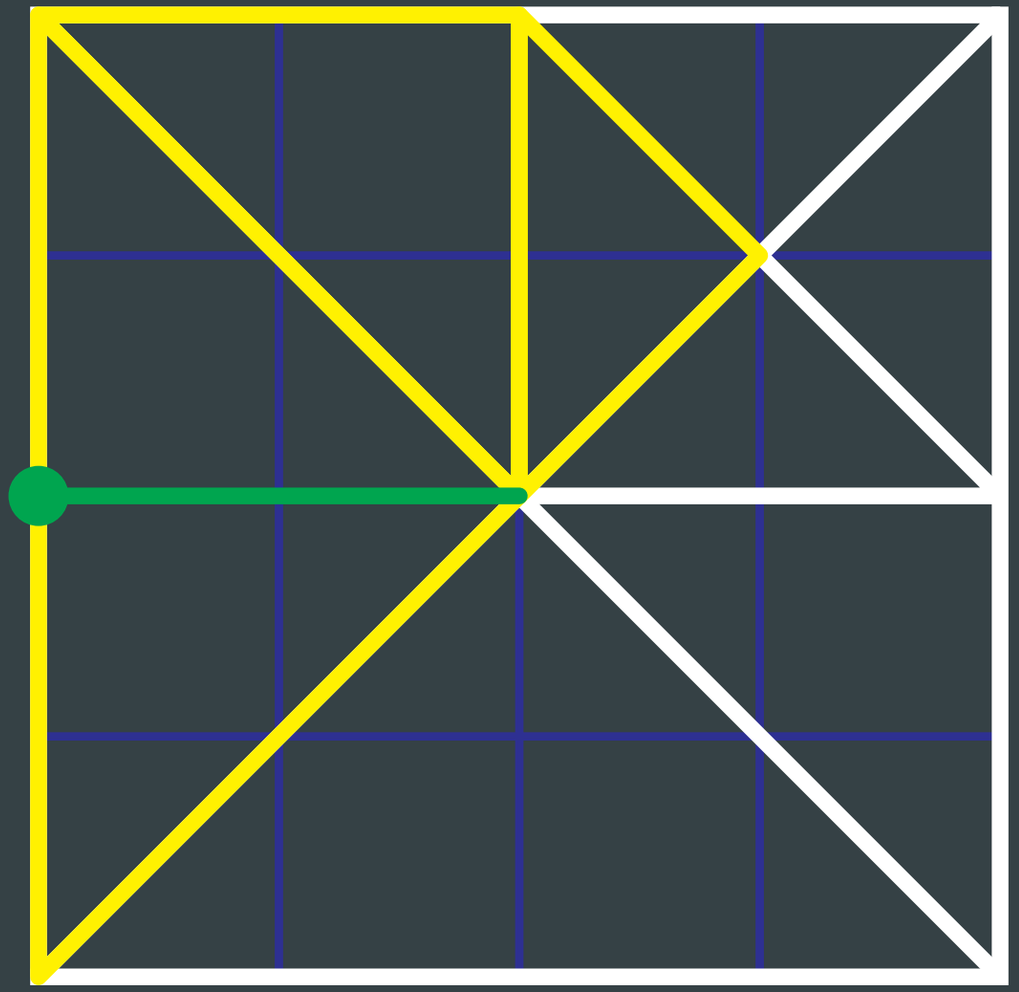
Force split



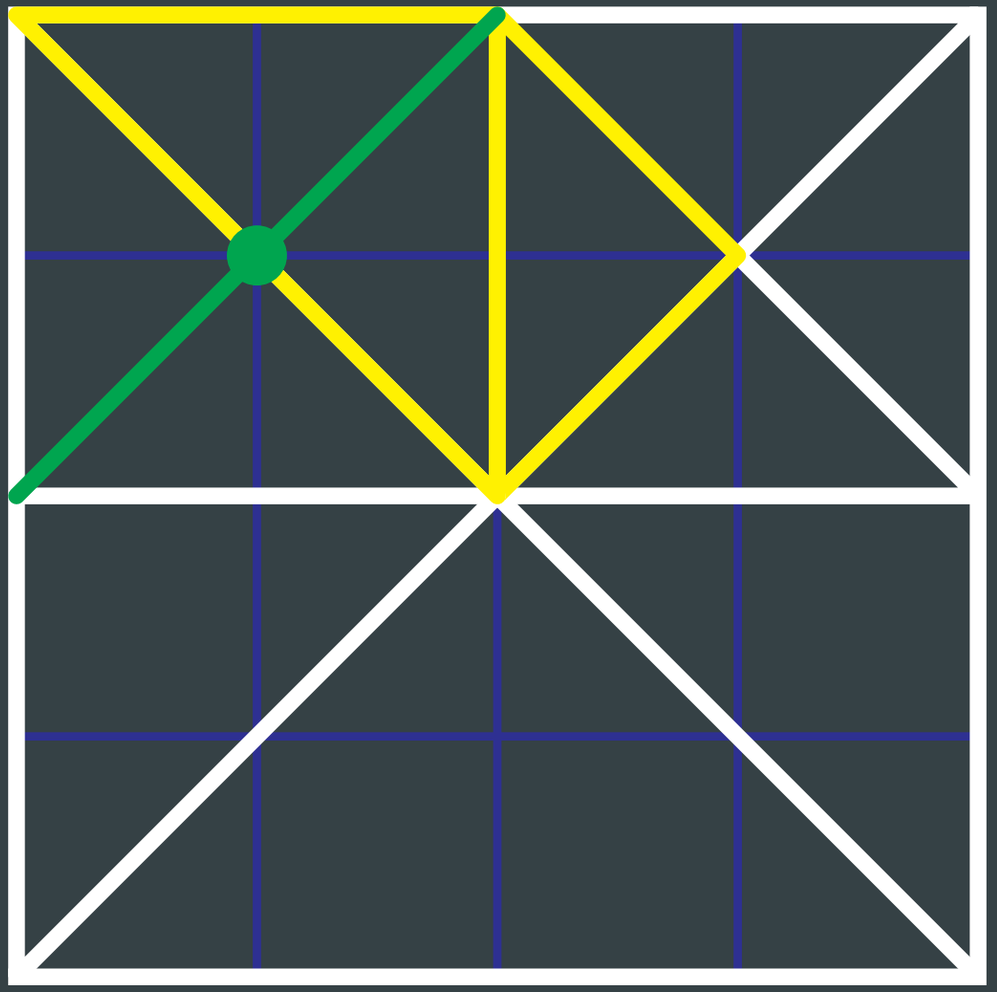
Force split



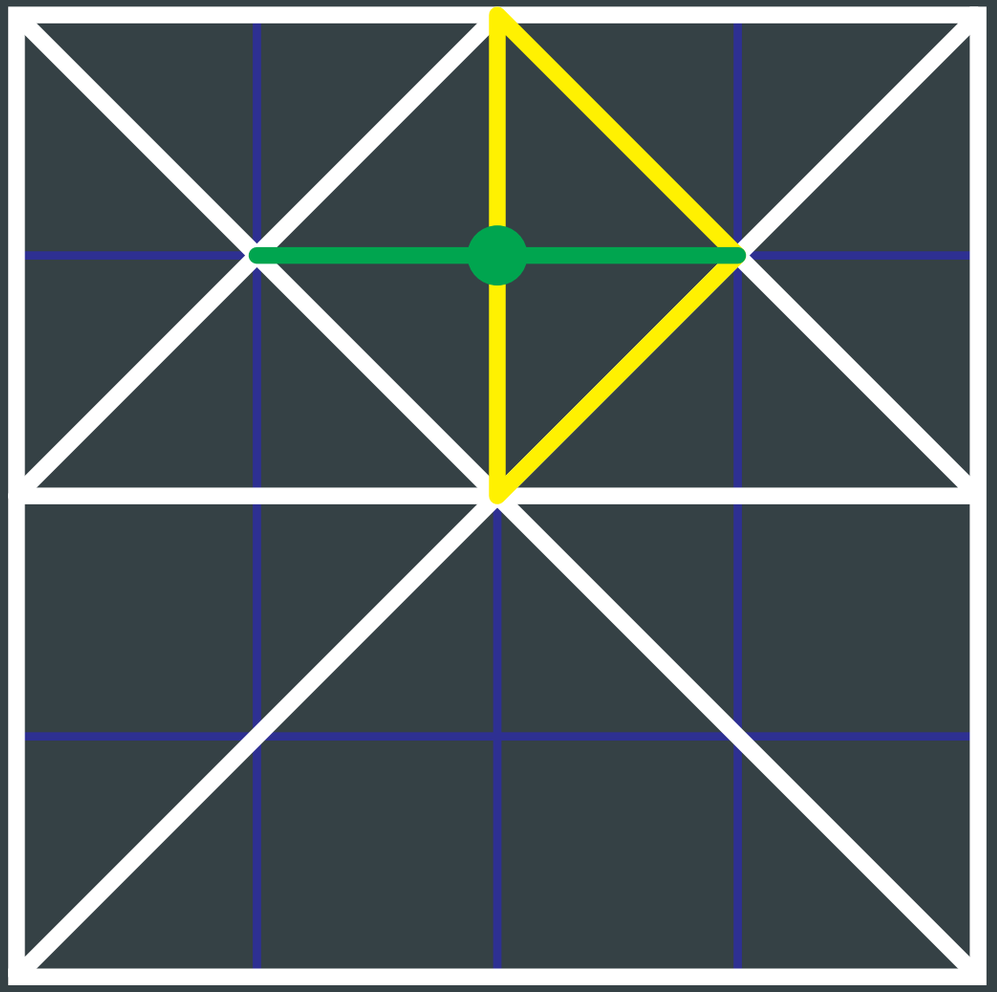
Force split

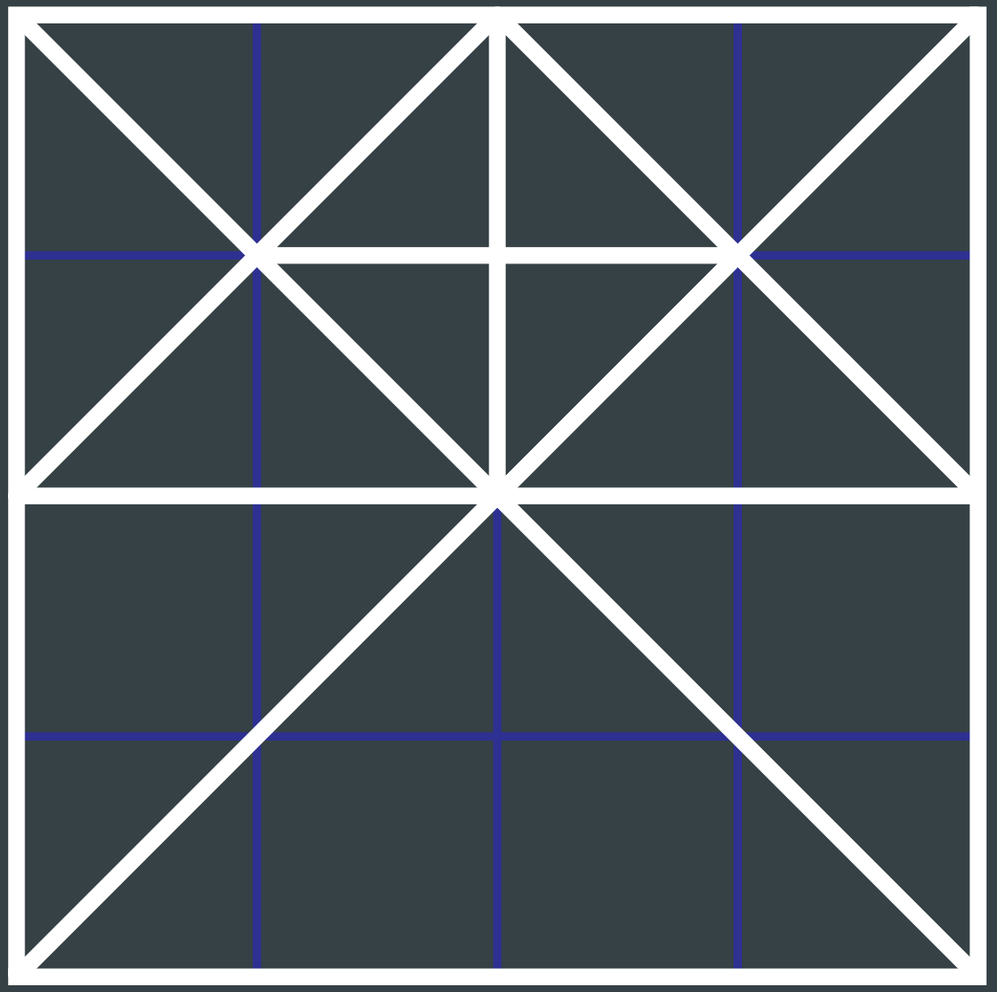


Force split

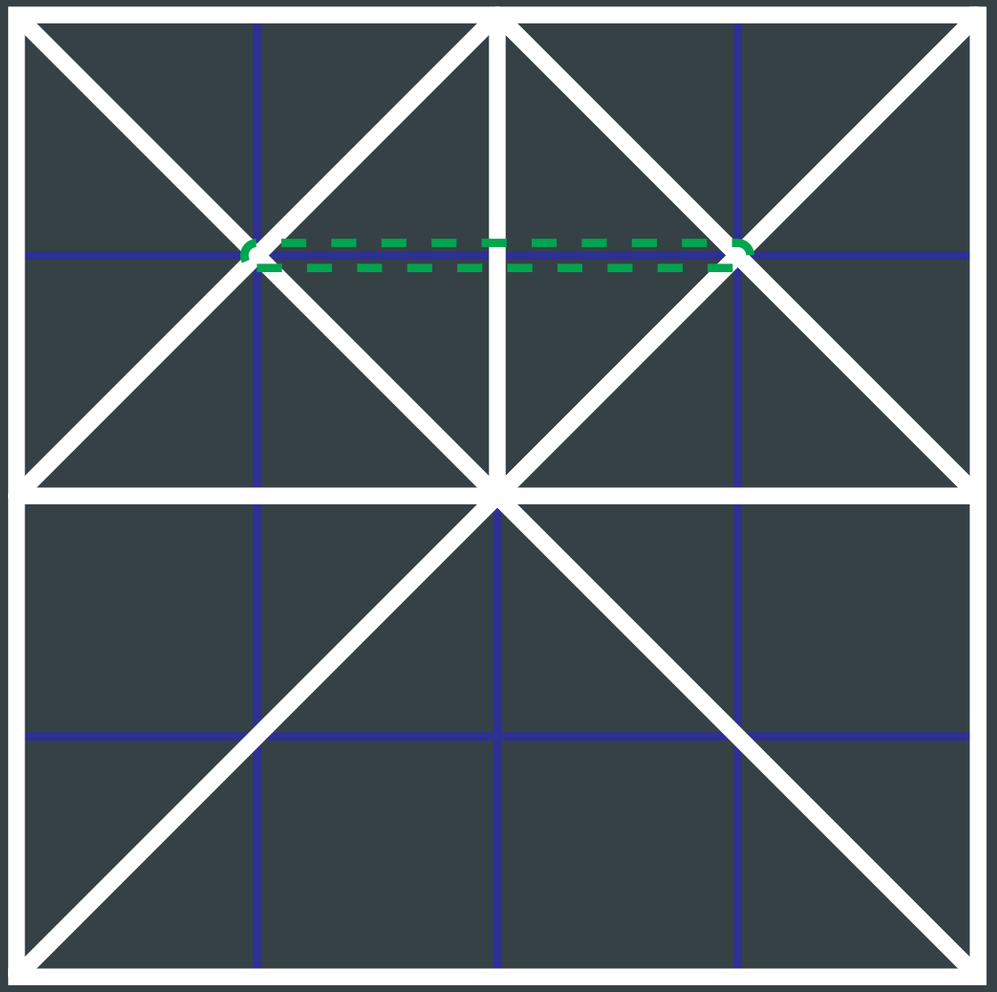


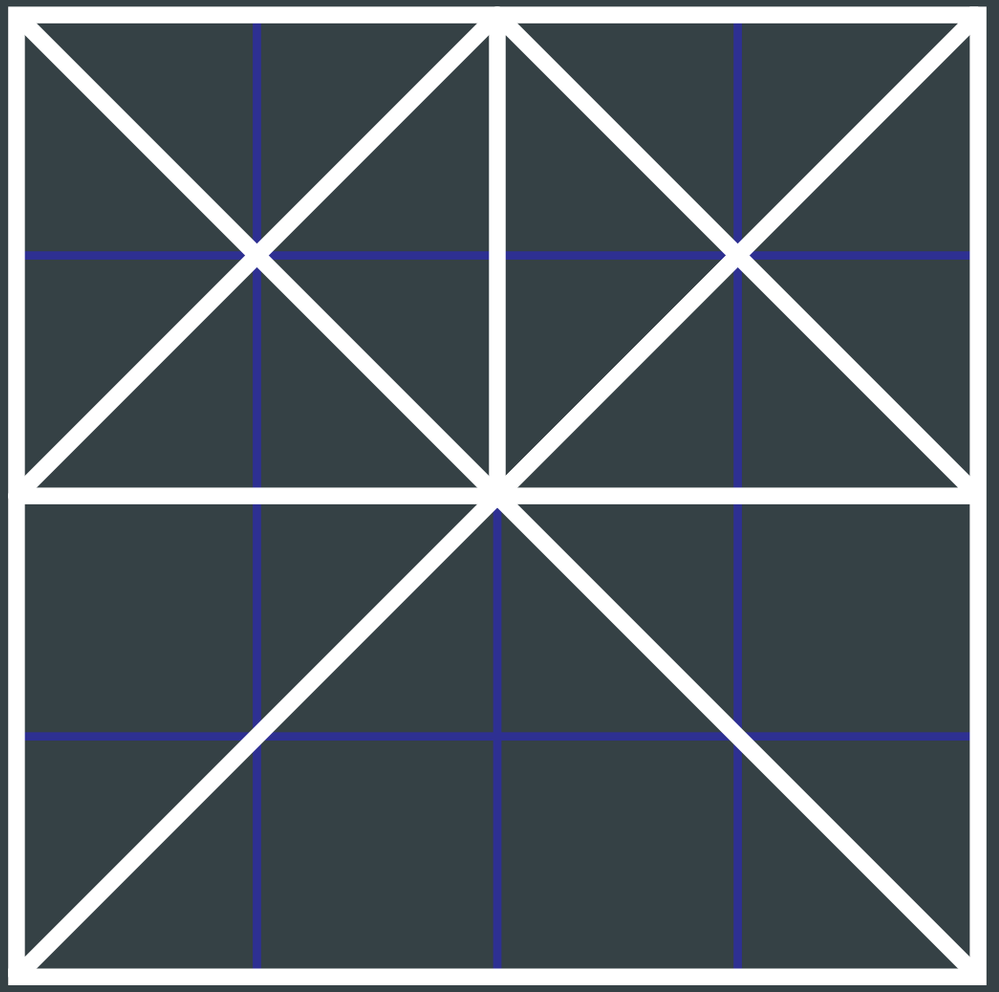
Force split



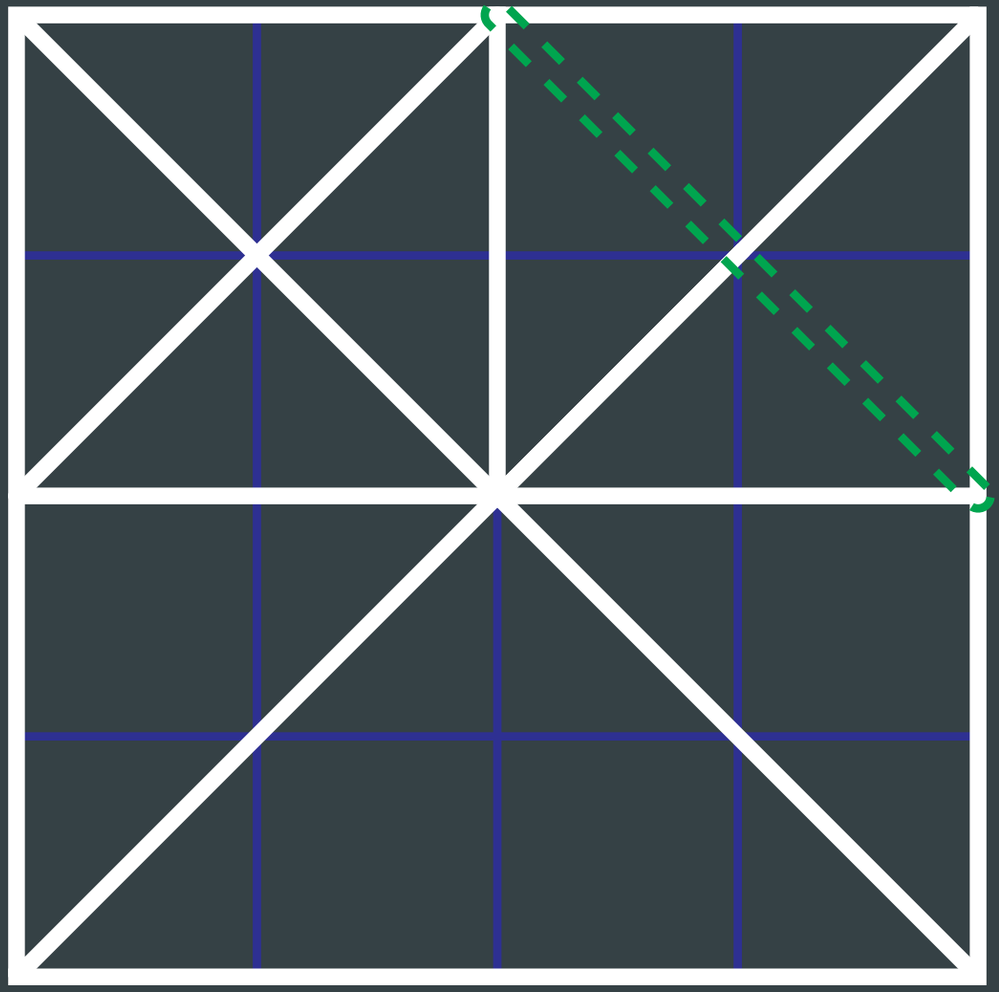


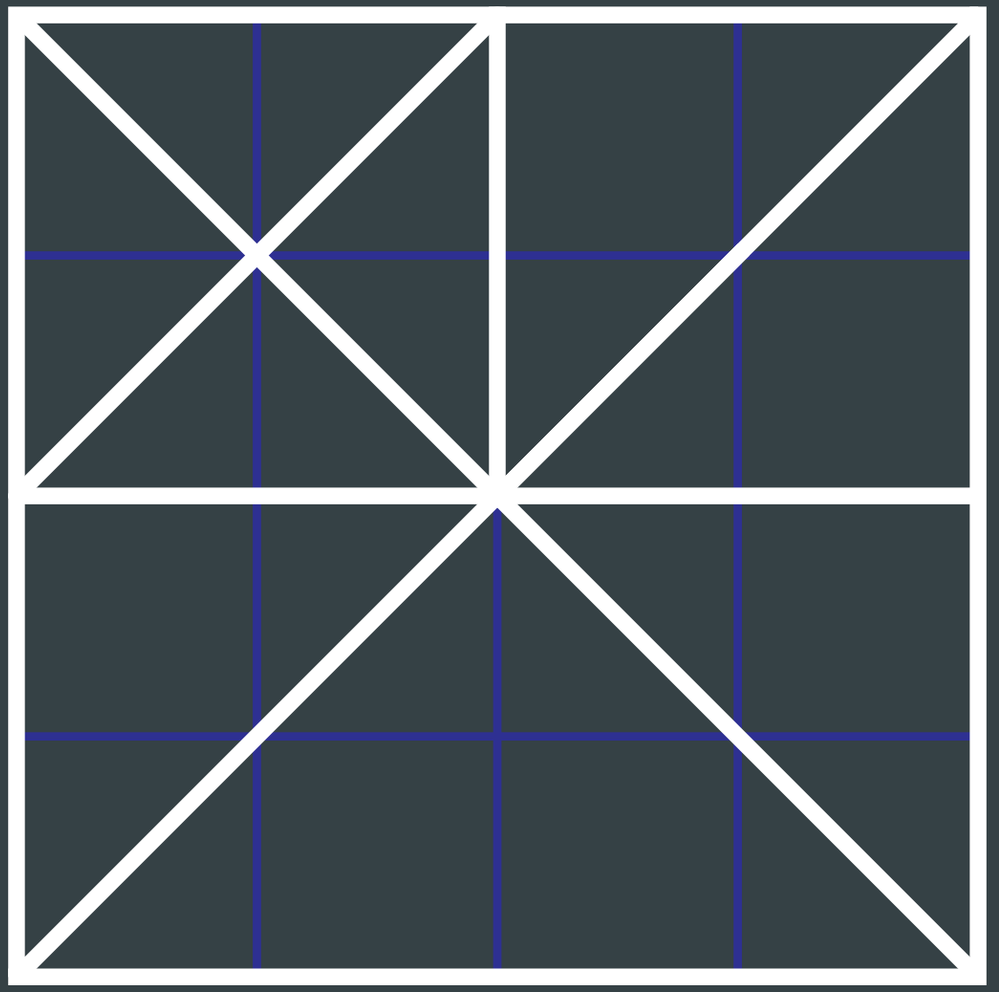
Merge





Another merge





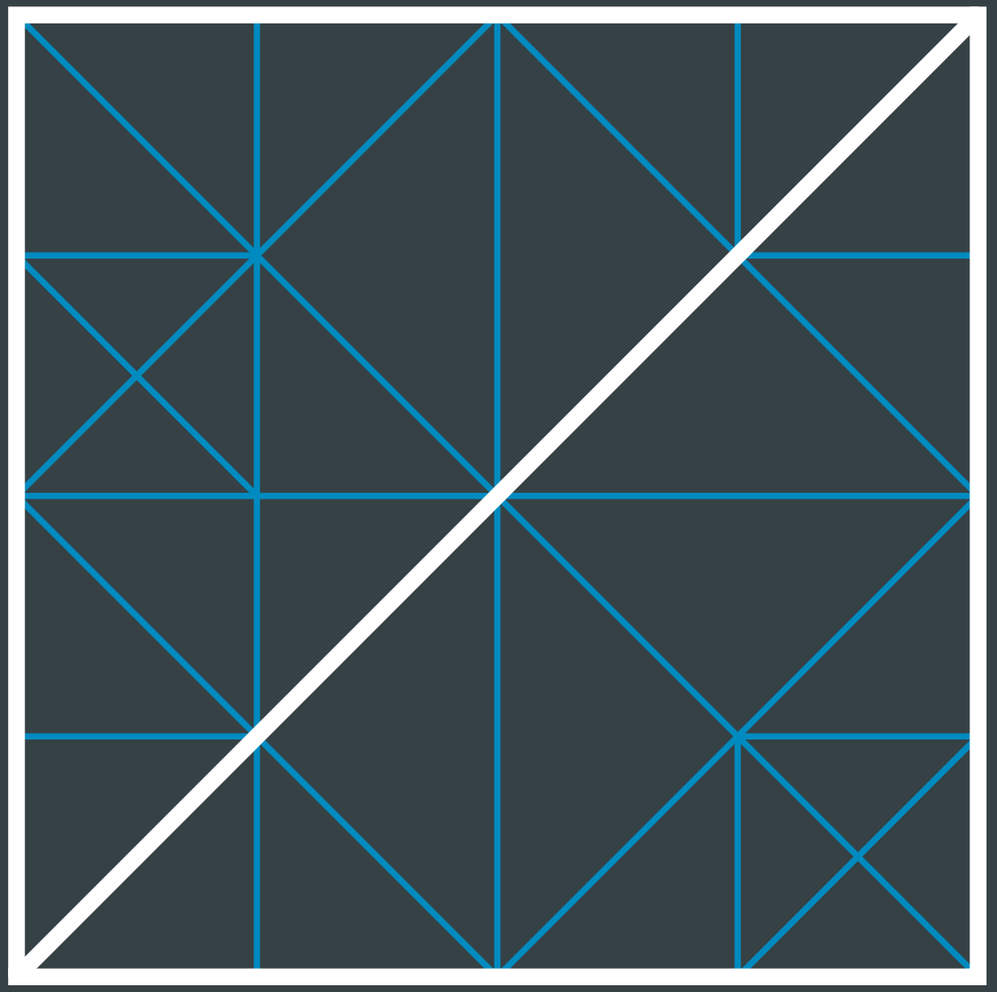
ROAM render loop

- Split and merge to update BTT from last frame
- Render on-screen parts of BTT

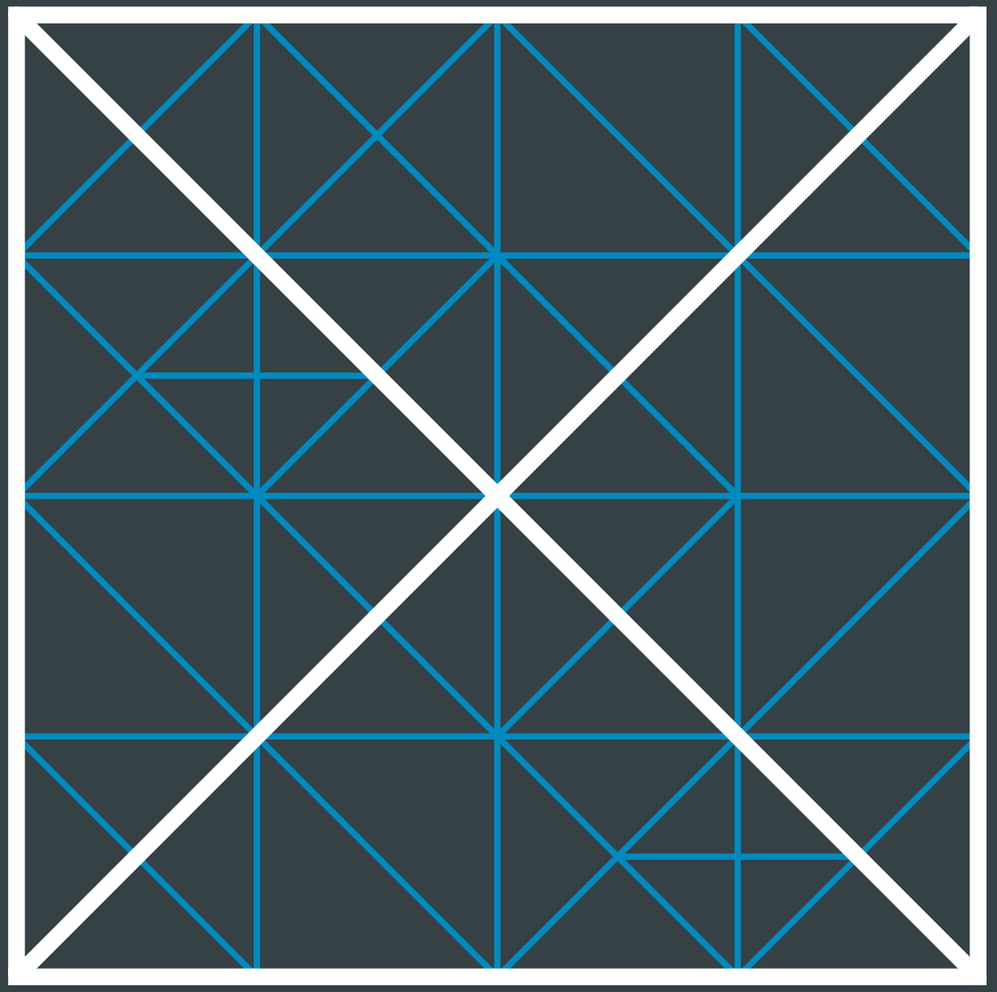
New algorithm: CABTT

- Two improvements: aggregation and caching
- Mostly avoids per-triangle work
- Can trade between CPU work and more triangles for graphics card
- Similar to RUSTiC except without having to precompute geometry
- *Aggregates* are large chunks of fixed geometry
- To change LOD, swap in more/fewer aggregates
- Aggregates may be cached

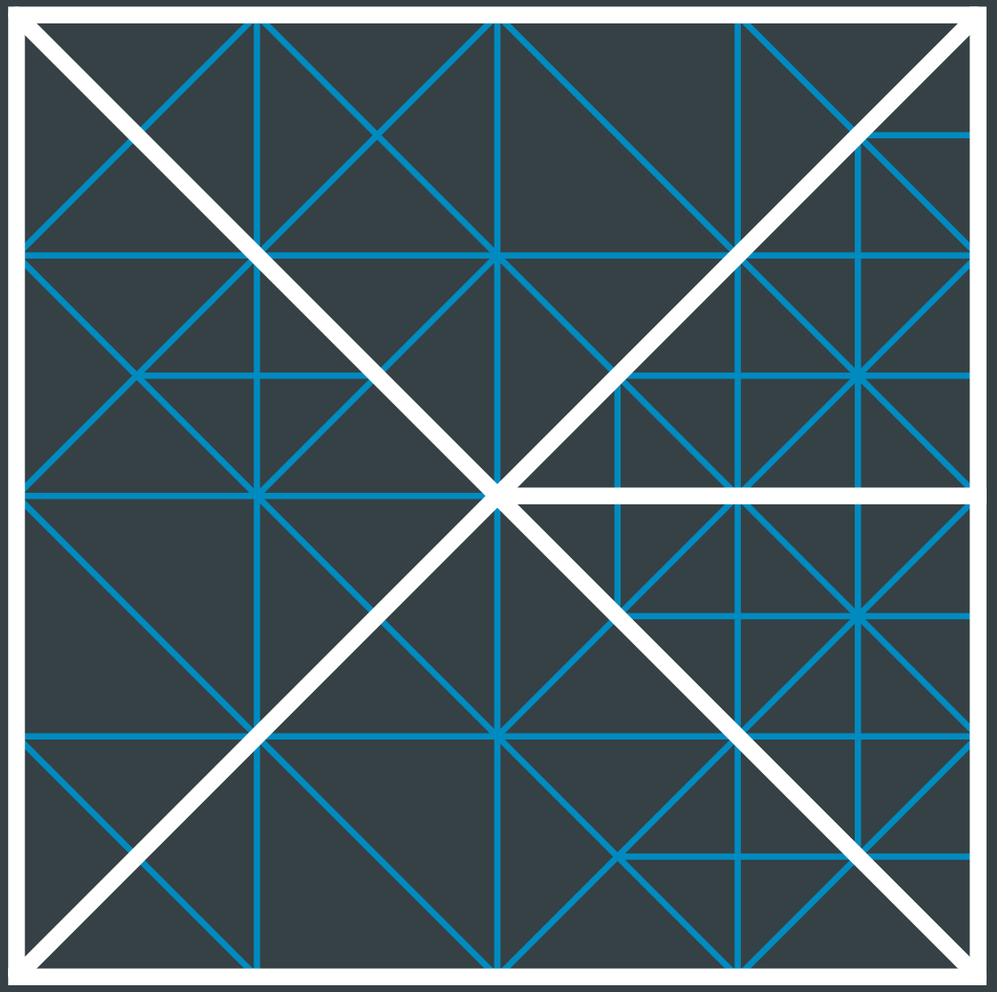
Example



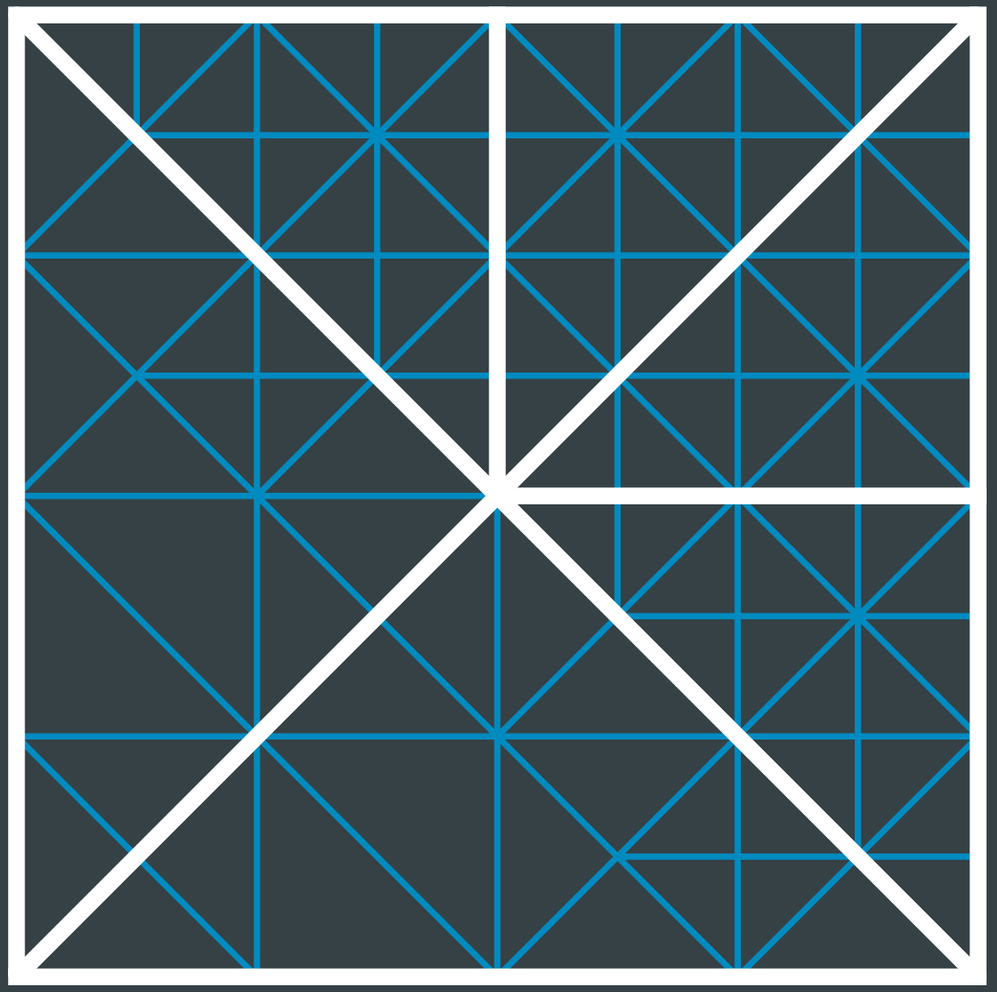
Split



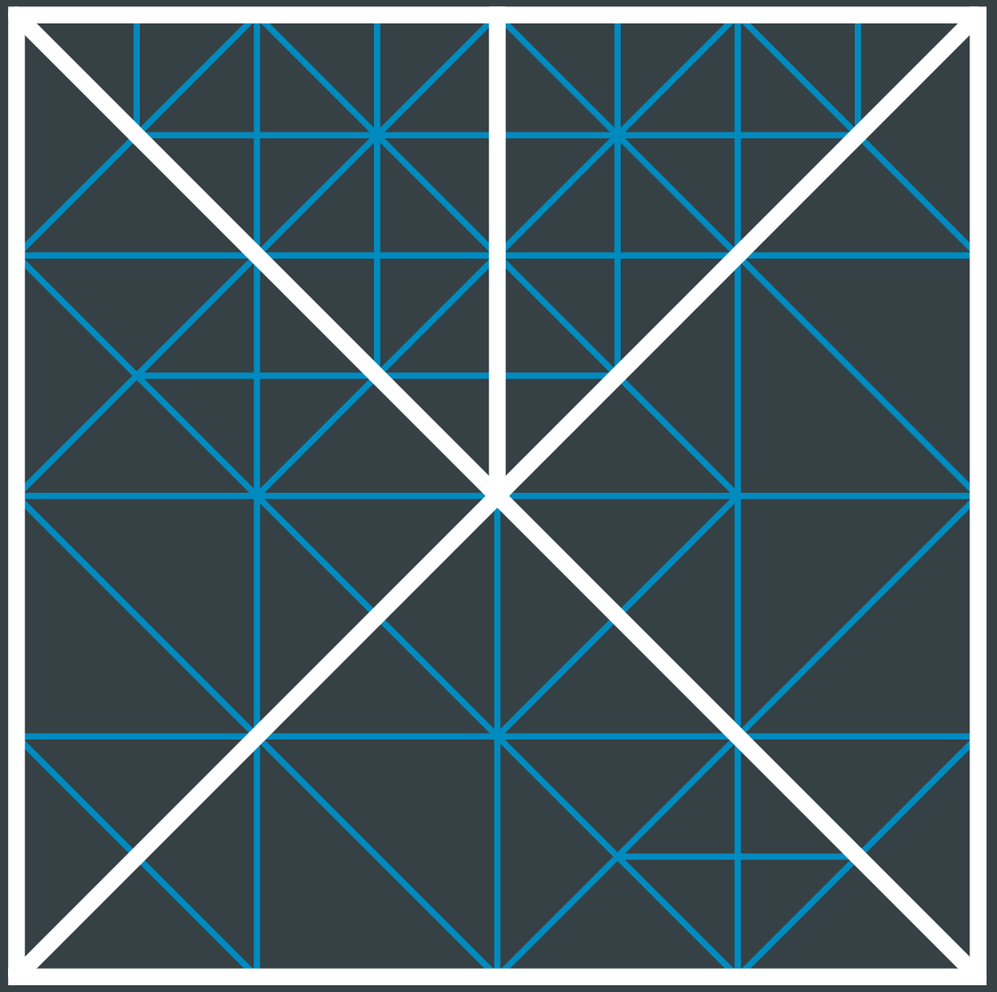
Split



Split



Merge



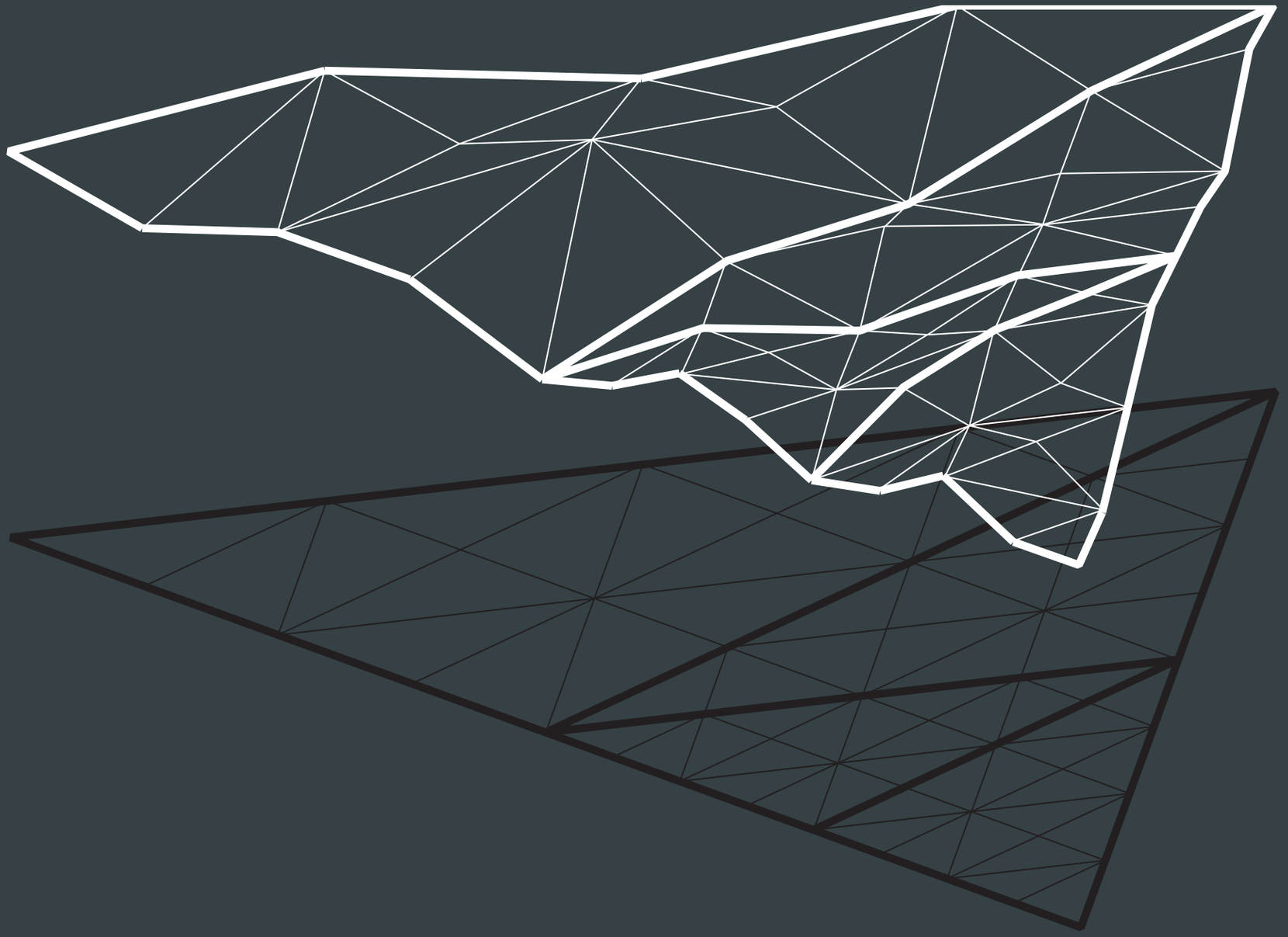
Aggregation Issues

- Continuity between adjacent aggregates
- Computing aggregate error
- Choosing amount of aggregation

Continuity

- Do not want T-junctions
- Adjacent aggregates are often at different levels of detail
- Mandate that aggregate boundaries are uniformly subdivided
- Costs $\leq$ 20%

Aggregate boundaries match up

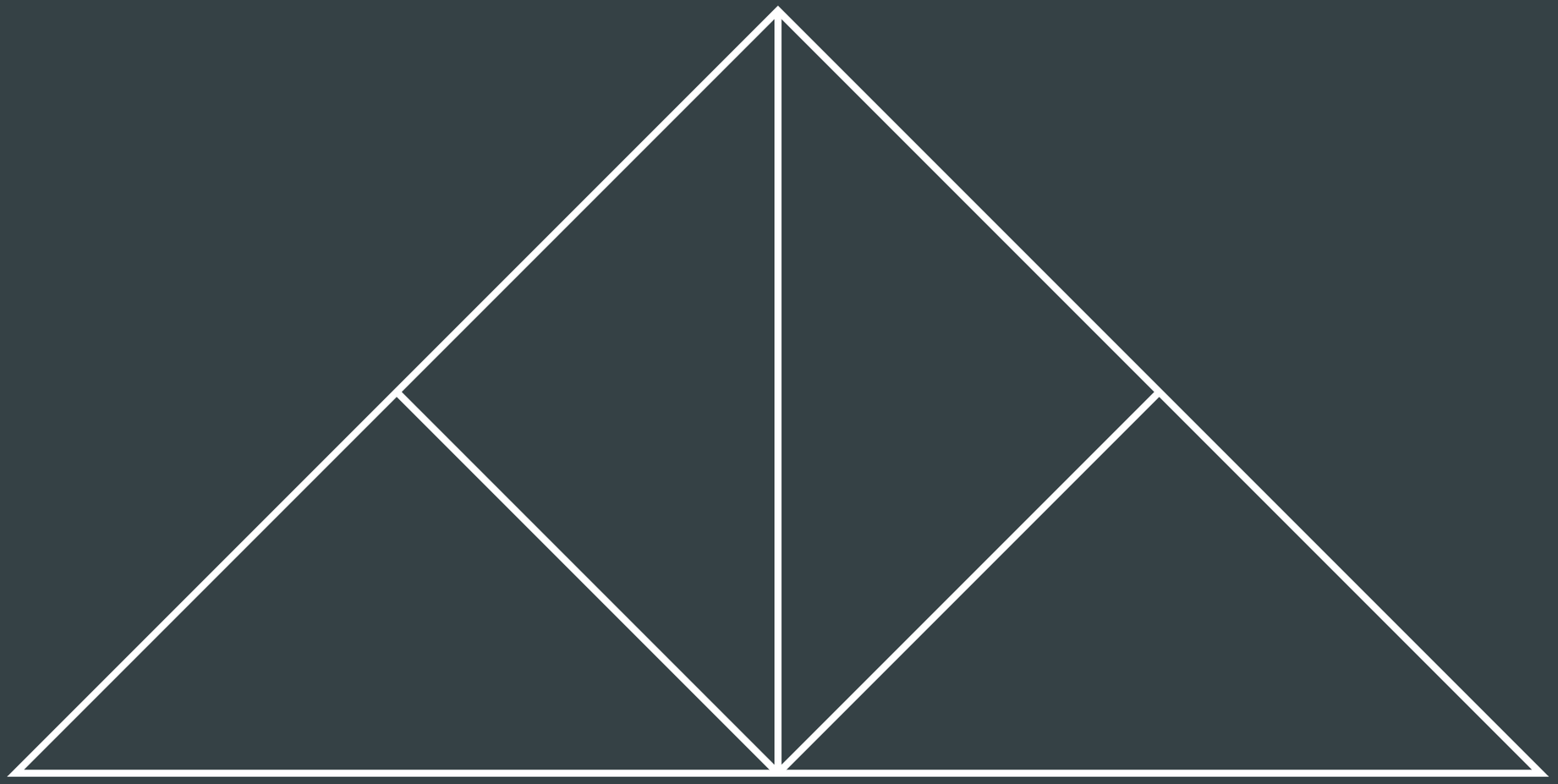


Adaptive subdivision

- Previous picture showed uniformly subdivided aggregates
- Instead use a mini-BTT to create geometry tailored to height field
- 2x improvement over uniform subdivision
  - fewer triangles per aggregate
  - lower approximation error
- “Tree of Trees”

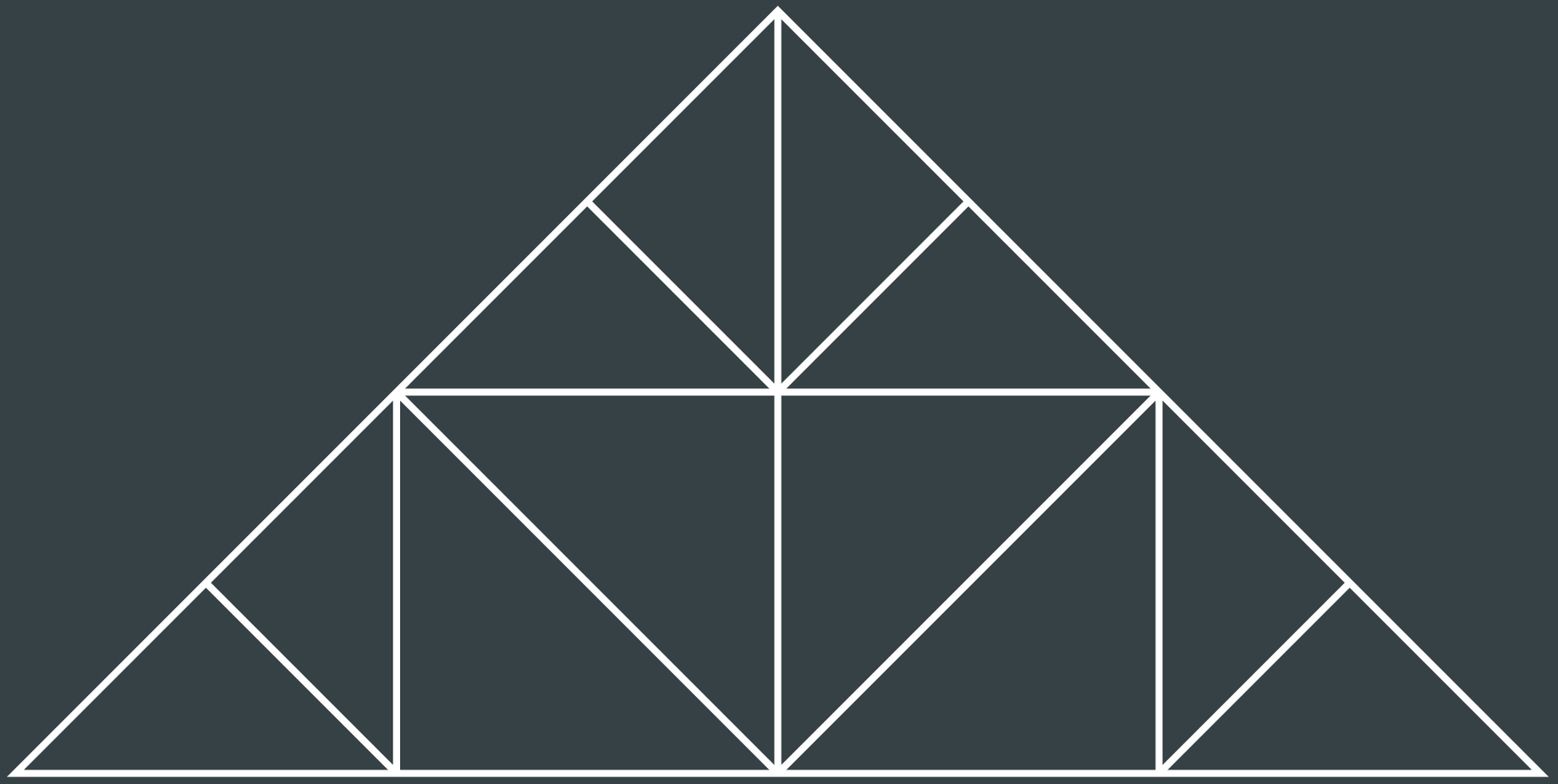
Initial subdivision of aggregates

n=1, 2 segments per edge



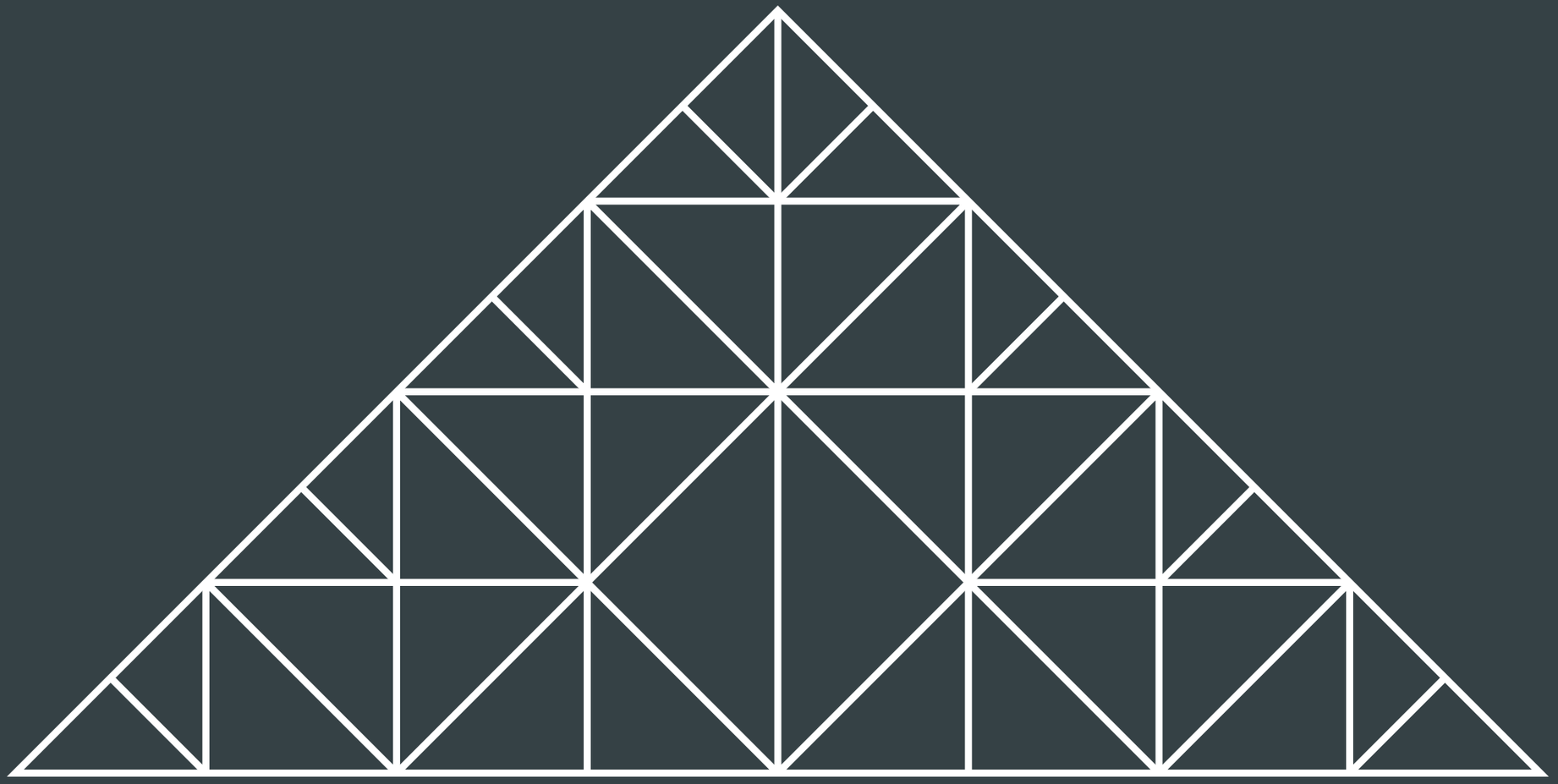
Initial subdivision of aggregates

n=2, 4 segments per edge



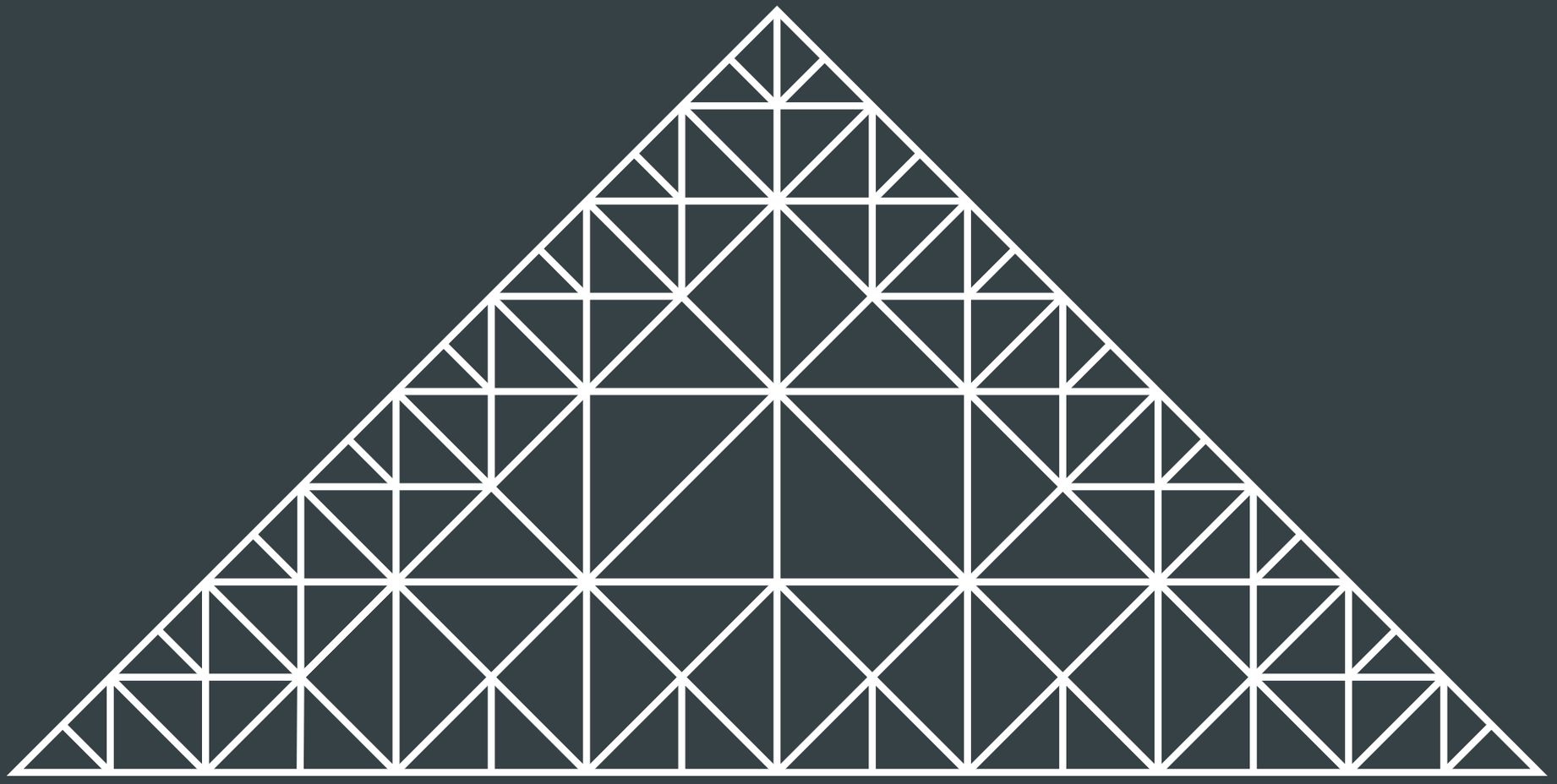
Initial subdivision of aggregates

n=3, 8 segments per edge



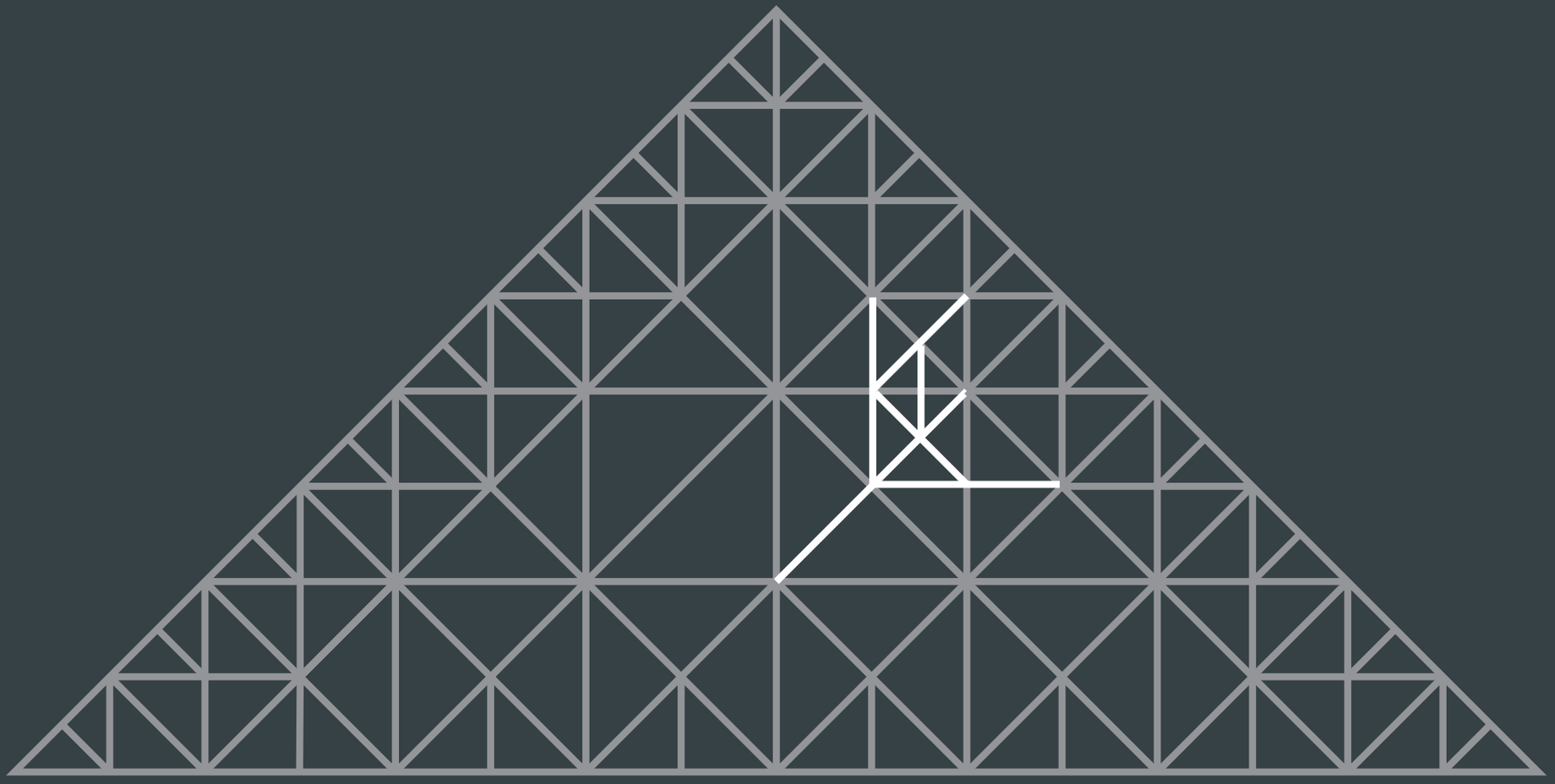
Initial subdivision of aggregates

n=4, 16 segments per edge



Interior is adaptively subdivided

Split to minimize error in interior



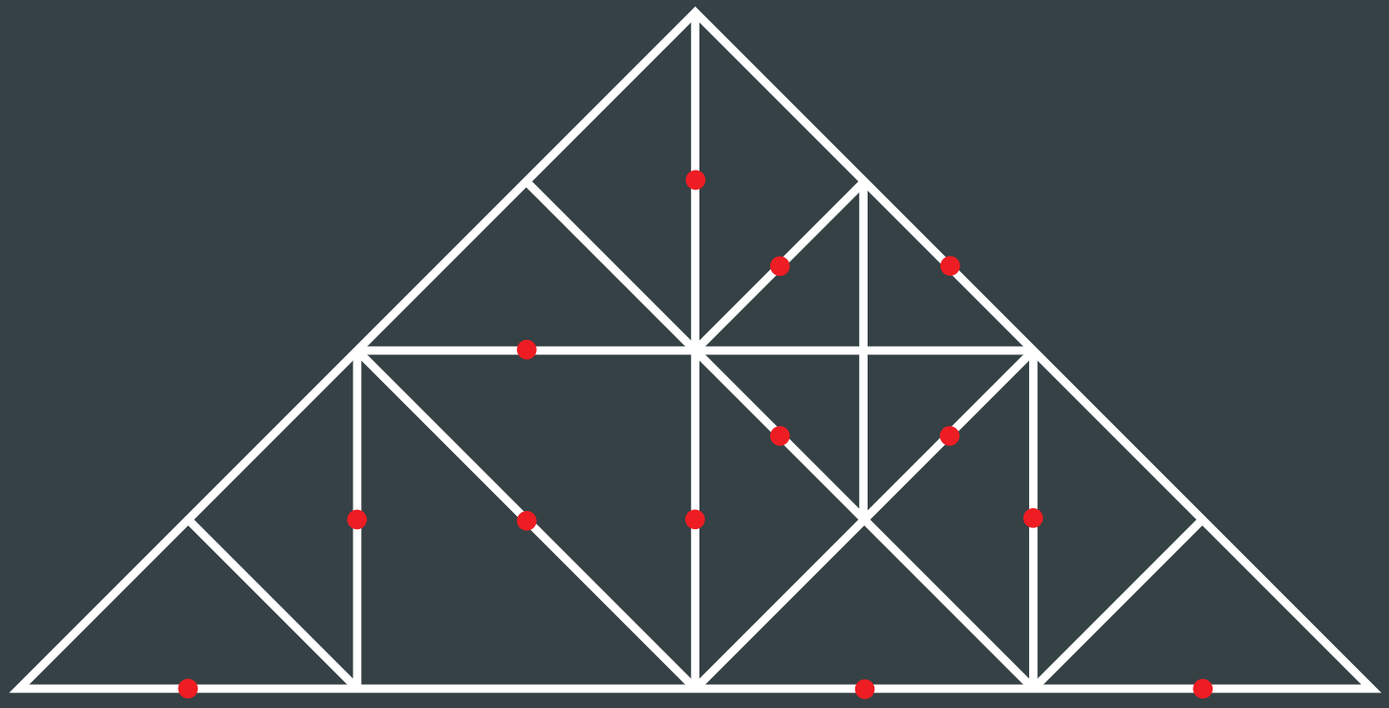
CABTT render loop

- Split and merge to update BTT from last frame
  - driven by a priority queue ordered by “camera movement until transitions are possible”
- Render on-screen parts of BTT

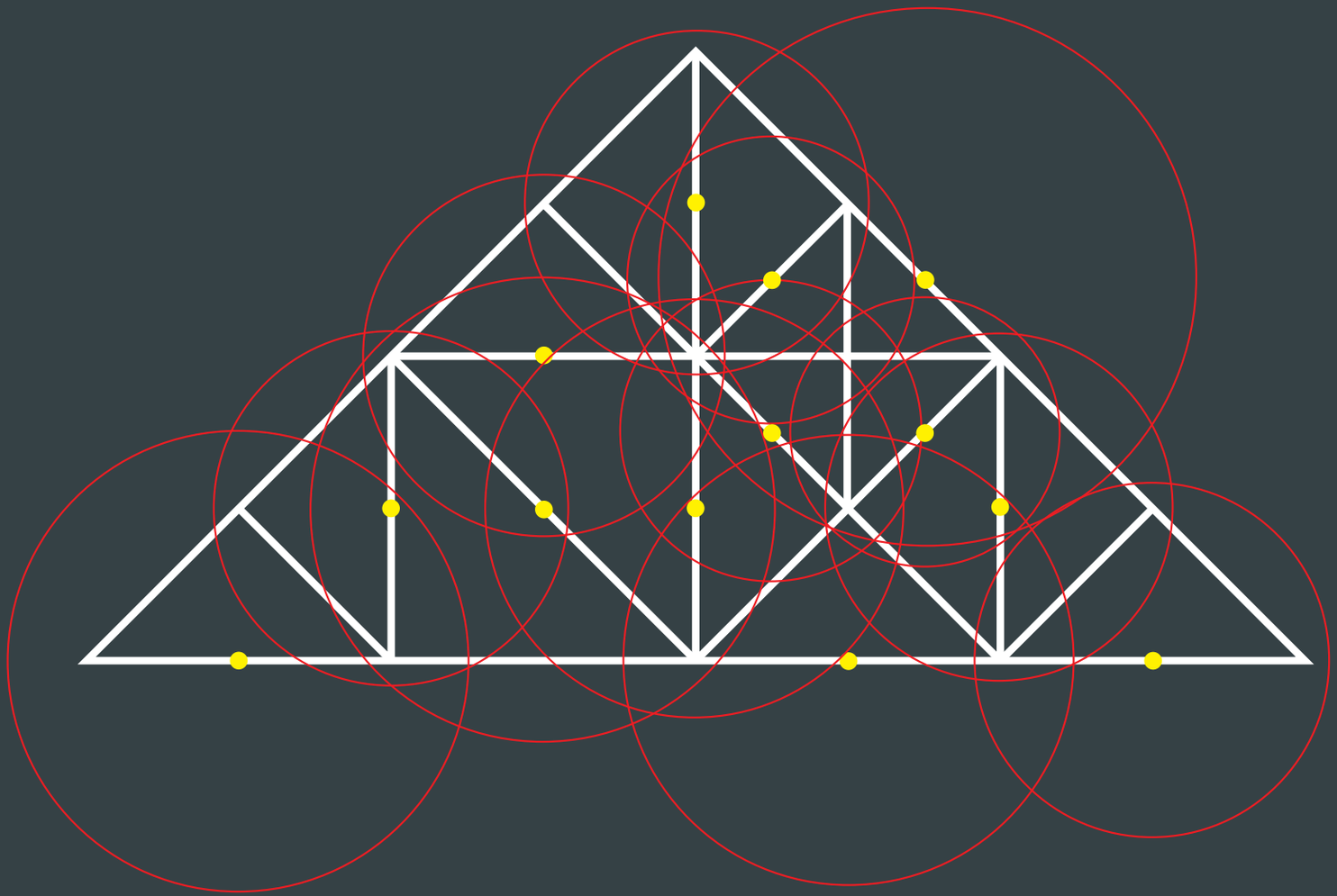
  If already cached:
  - render cache

  Else:
  - construct aggregate/mini-BTT
  - send geometry to video card
  - discard mini-BTT

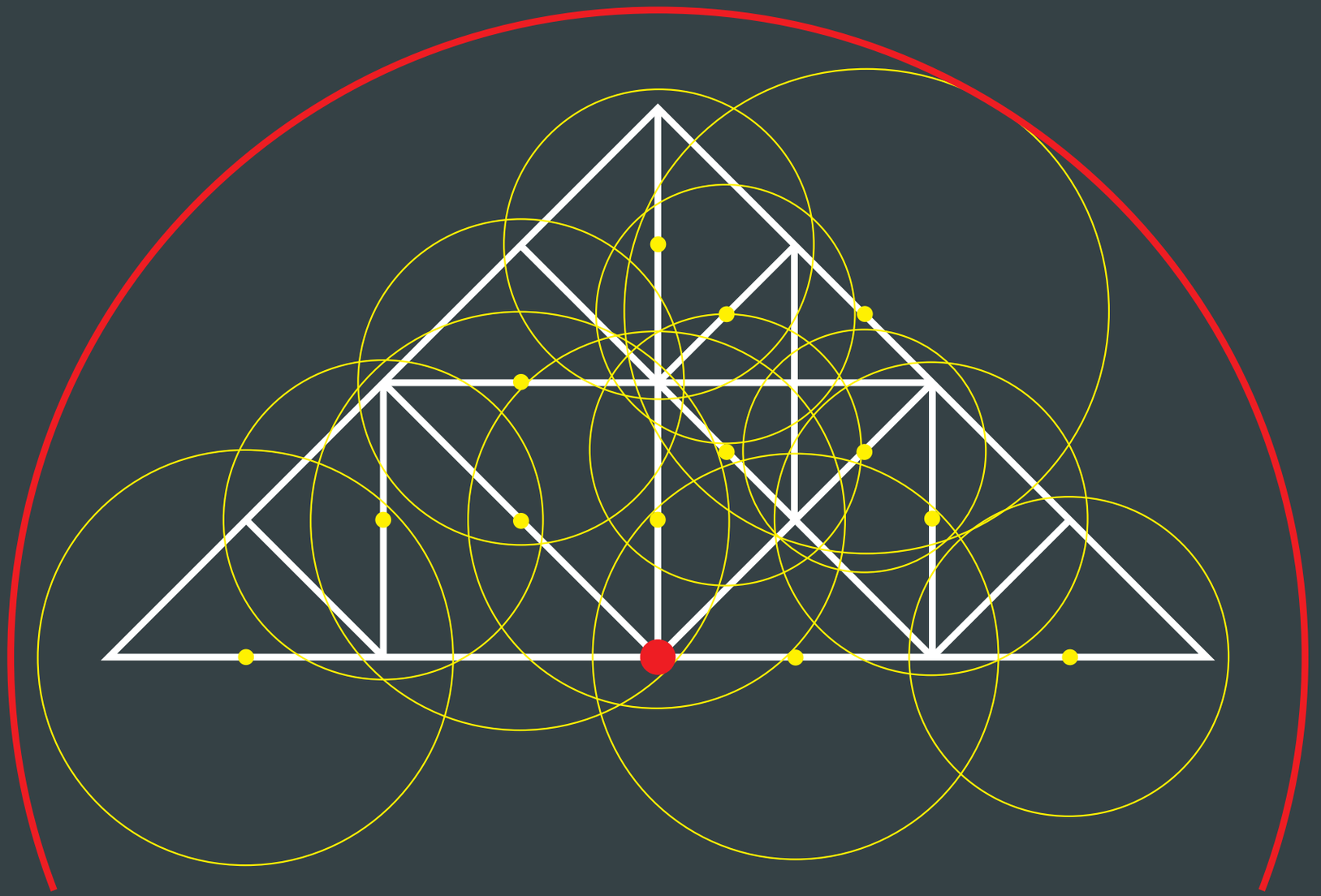
Aggregate error



Aggregate error



Aggregate error



Choosing aggregation level

- Profile a few levels at install time
- More aggregation:
  - more triangles to render
  - uses less main memory
  - less CPU processing
  - fewer splits and merges
  - more work for every split and merge
- Rule of thumb: n=4

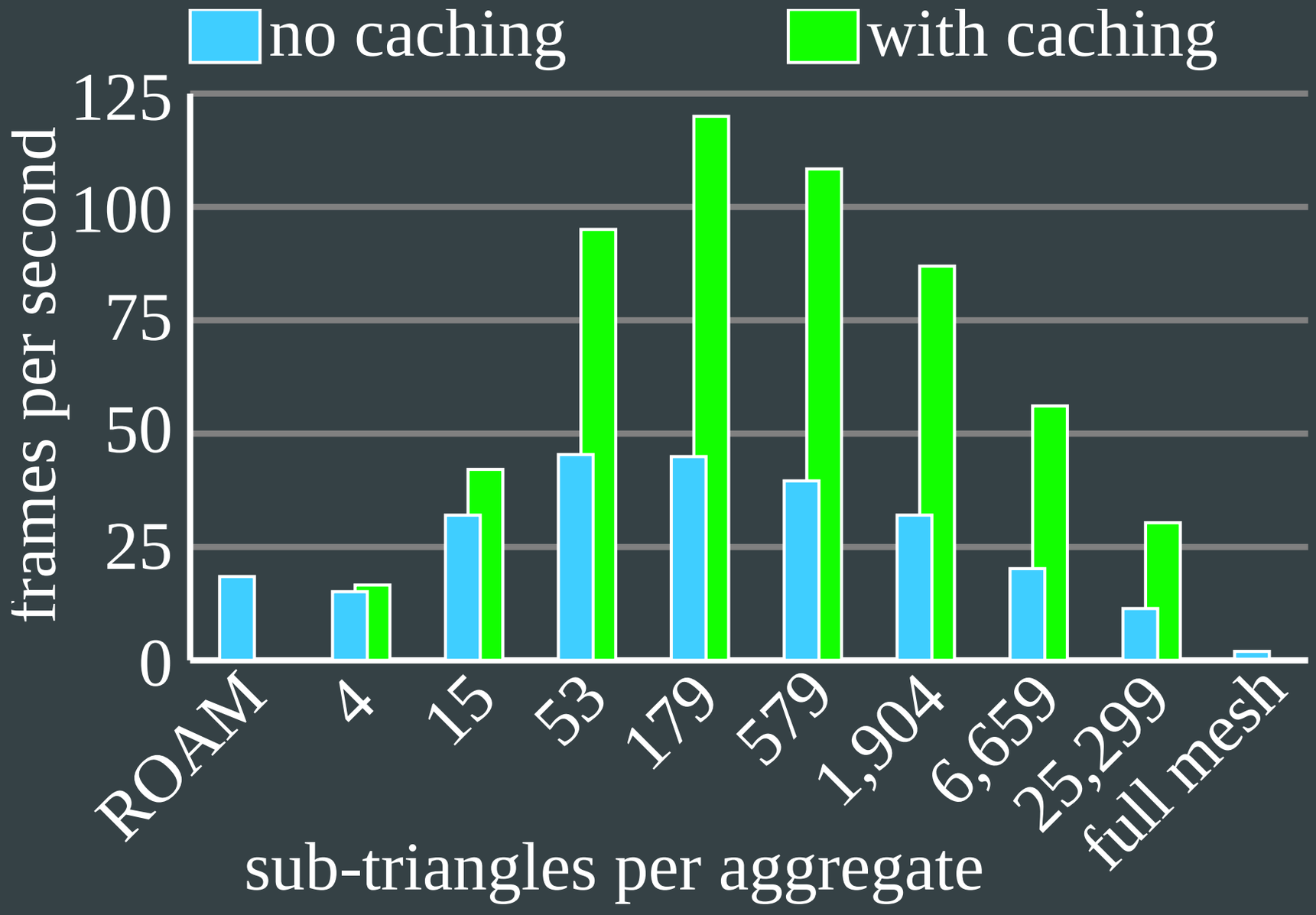
Aggregation level 4

No aggregation	$n = 4$
	180 triangles/aggregate
1 segment in boundary	16 segments in boundary
60,000 triangles	150,000 triangles
420,000 tree nodes	7,000 tree nodes
1.3 MB active data	0.3 MB active data
5500 split/merge tests per frame	90 split/merge tests per frame
300 splits per frame	4 splits per frame

Advantages of CABTT

- Greatly improved performance
  - higher frame rates
  - lower error tolerances
  - more detail up close
  - larger worlds or “to the horizon” rendering
- Adaptable to many hardware configurations
- Future proof
- Very efficient *instancing* (rendering many identical objects)
- Preserves most good features of ROAM

Performance



Disadvantages of CABTT

- 2.5x triangles to achieve same error bound (but each renders 18x faster)
- More expensive to change error threshold
- Minimum detail level
  - Could switch to a different LOD algorithm for far away objects

Future work

- Out-of-core rendering
- Other geometry
- Multiple instances

Questions?

Interactive Demonstration from 4–5pm today.

`http://www.technomagi.com/josh/vis2002/`

- Source code
- Demo
- These slides

Large data sets

- Large textures are memory intensive
  - Texture LOD: Assign a static texture to each aggregate triangle
- Simple: use memory-mapped files
  - *Visualization of Large Terrains Made Easy*, Lindstrom and Pascucci, IEEE Vis 2001
- Harder: use non-blocking I/O
  - Anticipate what data will be needed in the next several frames
  - Prevent splits until data is loaded
  - Keep aggregate error data in memory (1 float per 256 grid values)

Non-terrain models

Lots of research into base mesh + displacement:

- *Normal meshes*, Guskov et al., SIGGRAPH 2000
- *Displaced subdivision surfaces*, Lee et al., SIGGRAPH 2000
- *Fitting smooth surfaces to dense polygon meshes*, Krishnamurthy and Levoy, SIGGRAPH 1996
- Remeshing: *MAPS*, Lee et al., SIGGRAPH 1998 and Kobbelt, et al., Eurographics ‘99

Limitation: Static mesh

- Main reason: error metric
- Workaround: aggregate error could be updated in background thread
- Alternative: use an error metric that is easier to update (monotonic)
- Alternative: update error metric approximately

Past ROAM improvements

- Generally reduced CPU per triangle rendered
- Improving cache coherency (Lindstrom and Pascucci)
- Hierarchical work queue (Blow)
- Simplifying error metric (Blow)
- Storing only leaves of the tree (Blow)
- RUSTiC: a form of aggregation, but high memory and lots of precomputation

All but the last of these stores every triangle in a BTT.

Multiple instances

- Idea is to share caches between models
- Keep a map from (base triangle, leaf number) in the model to caches
- Maintain a count of how many instances are using each cache
- Still have an aggregated BTT for each model, plus a location and orientation

Uniform boundary splitting

```
SplitEdge(BttNode, Side):
        if BttNode is a leaf:
                BttNode.Split()
                if Side is not base:
                        BttNode.Child(Side).Split()
        else if Side is base:
                SplitEdge(BttNode.Child(left), right)
                SplitEdge(BttNode.Child(right), left)
        else:
                SplitEdge(BttNode.Child(Side), base)
```

Computing aggregate error

- I use spherical error volumes

$$\text{screen error} \approx C\,\text{world space error} / \text{distance to camera}$$

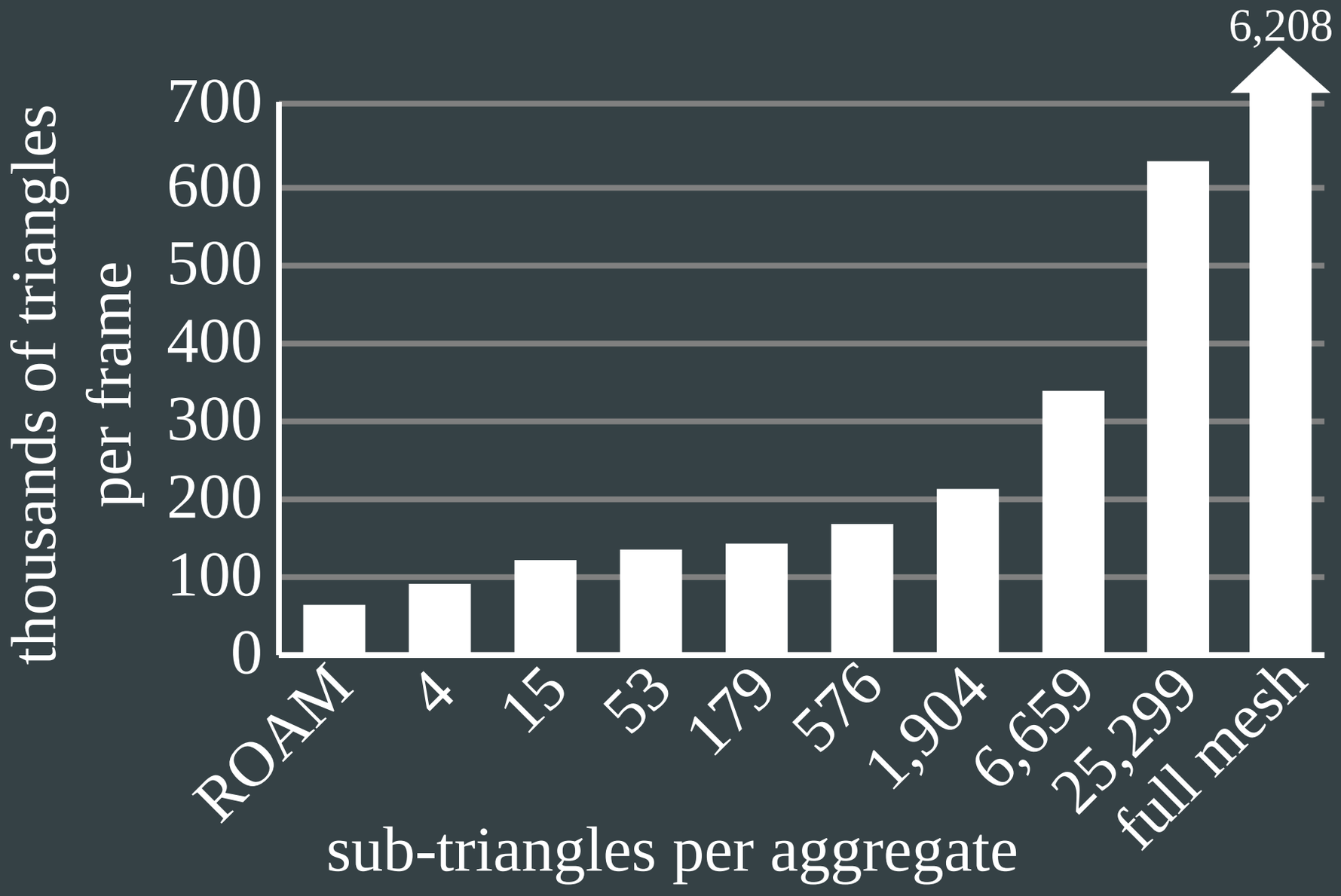
- Triangle inequality:

$$\begin{aligned}\text{view point to triangle} \leq\ & \text{view point to center} \\ & + \text{center to triangle}\end{aligned}$$

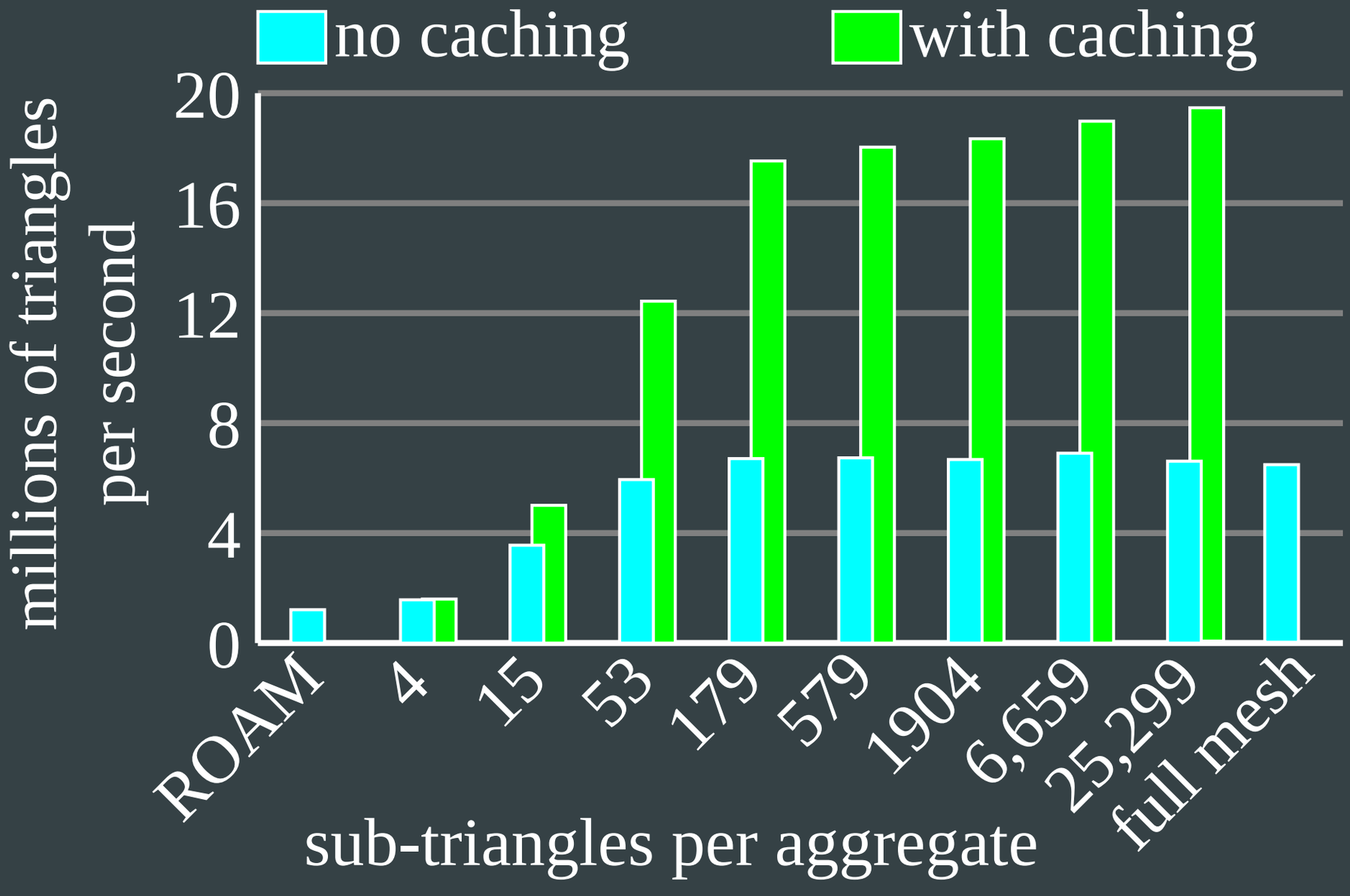
- Aggregate error contains error volumes of all sub-triangles

$$\text{Radius} = \max\left\{\text{center to triangle} + \frac{C\,\text{world space error}}{\text{screen error}}\right\}$$

Geometric complexity



Rendering speed



Performance table

	Laptop		Desktop	
	2k x 2k	4k x 4k	2k x 2k	4k x 4k
CABTT	121 fps	120 fps	67 fps	39 fps
No caching	47 fps	46 fps	25 fps	17 fps
ROAM	19 fps	18 fps	9.7 fps	6.1 fps
Full mesh	4.5 fps	1.3 fps	2.3 fps	0.6 fps
Full mesh w/ caching	14 fps			

Laptop has a 1.7 GHz Pentium 4m, NVIDIA GeForce 4 440 Go

Desktop has a 450 MHz Pentium 2, NVIDIA GeForce 2 GTS

2k x 2k mesh is a fractal synthesis

4k x 4k mesh is Puget Sound data set

Index

Don’t want T-Junctions
BTT Example
CABTT Example
Aggregate boundaries match up
Initial subdivision of aggregates
Aggregate error
Aggregation level 4
Performance (frames/second)
Questions?
Large data sets
Non-terrain models
Limitation: Static mesh
Past ROAM improvements
Multiple instances
Uniform boundary splitting
Computing aggregate error
Geometric complexity
Rendering speed
Performance table