MSc Computer Games and Entertainment

Maths & Graphics Unit 2011/12

Lecturer: Gareth Edwards

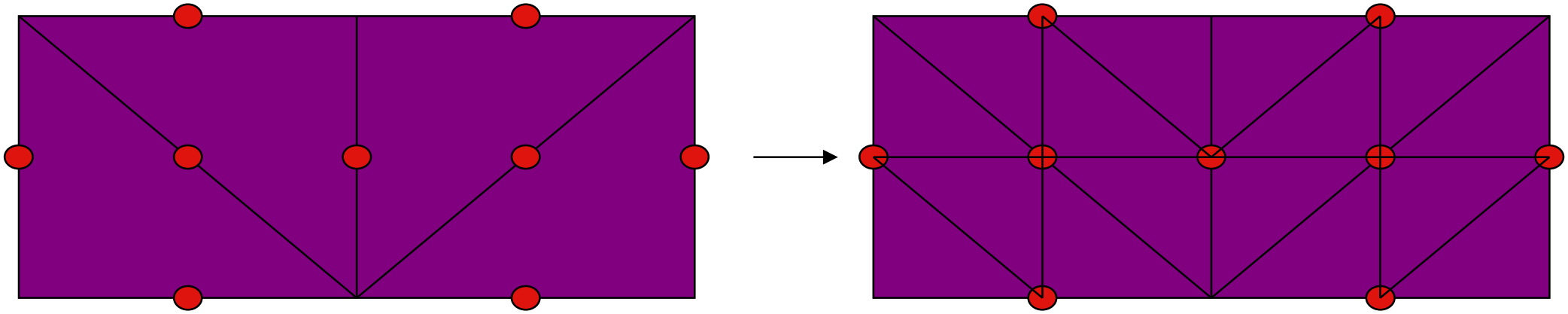
Subdivision Of Triangular Terrain Mesh

Breckon, Chenney, Hobbs, Hoppe, Watts

Fractal Terrain

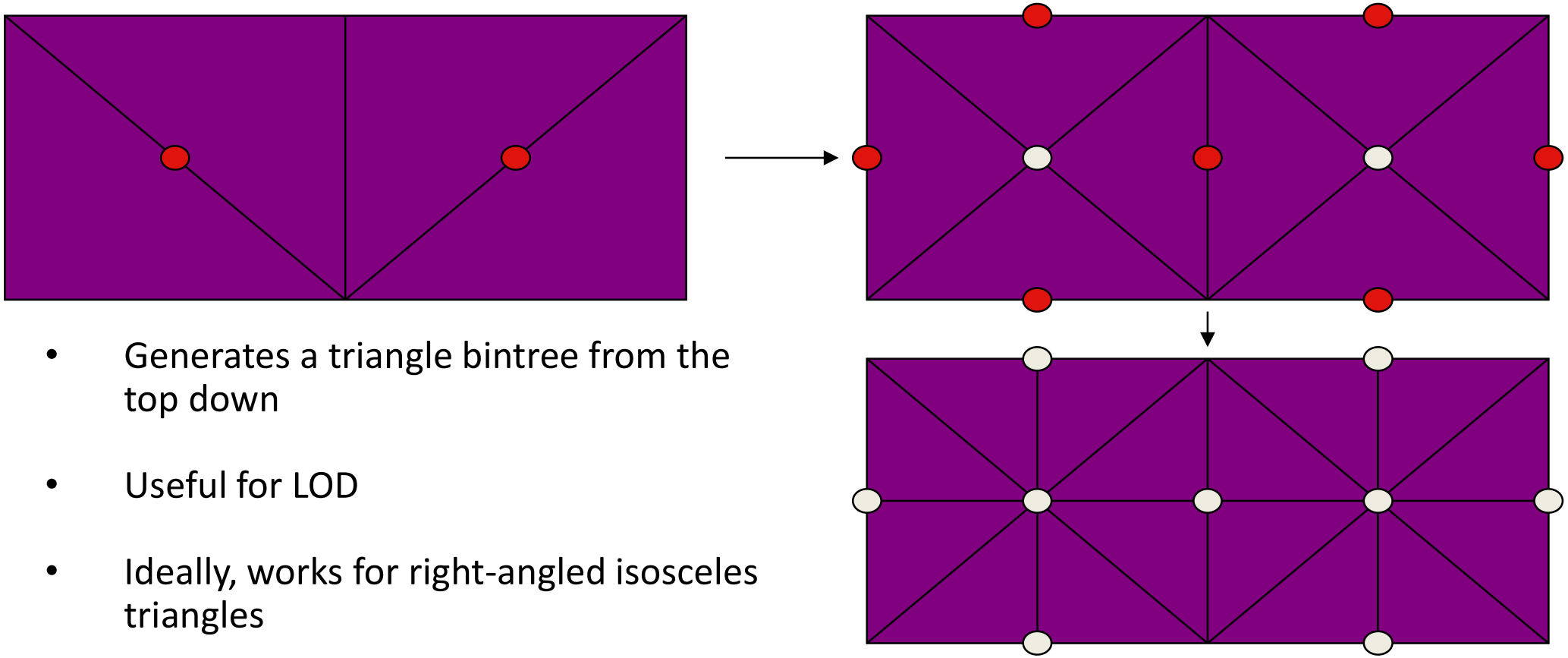
- Based on subdivision of a course polygon mesh
- Each subdivision adds detail to the mesh in a carefully controlled random way
- Algorithm (starting with a triangular mesh):
  - Split each edge, and shift the new vertex up or down by a random amount
  - Subdivide the triangles using the new vertices
  - Repeat
- There are also algorithms for quadrilateral meshes

Subdivision Method 1



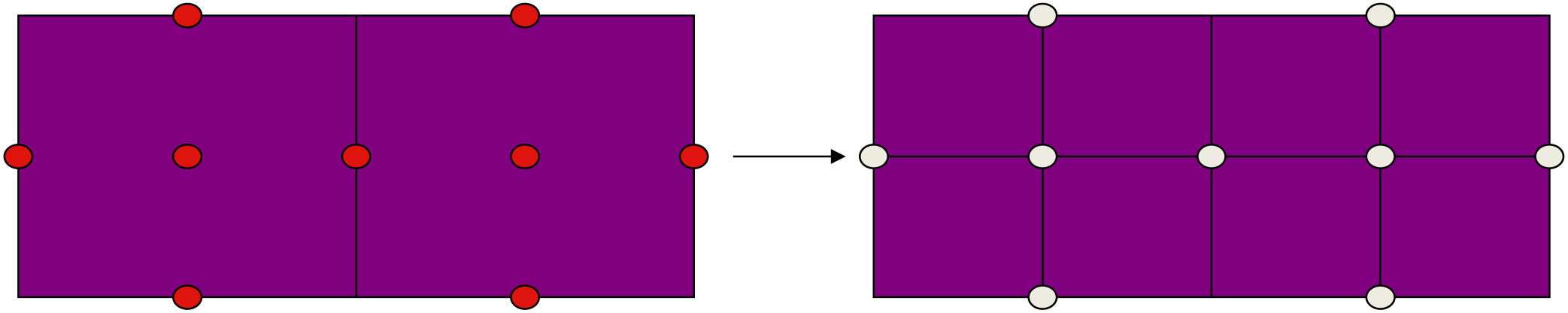
- Works on any triangular mesh - does not have to be regular or have equal sized triangles.

Subdivision Method 2



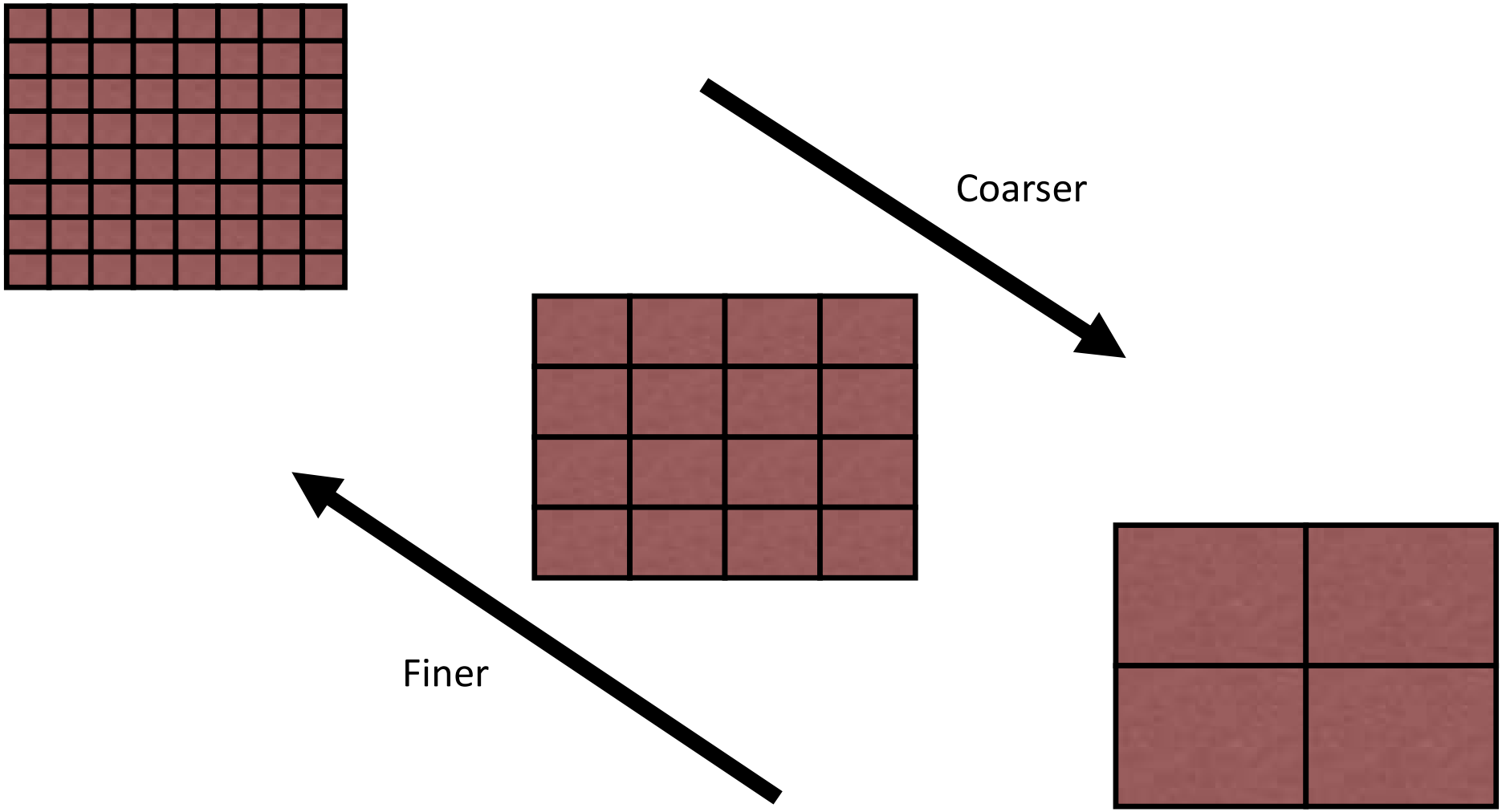
- Generates a triangle bintree from the top down
- Useful for LOD
- Ideally, works for right-angled isosceles triangles

Subdivision Method 3

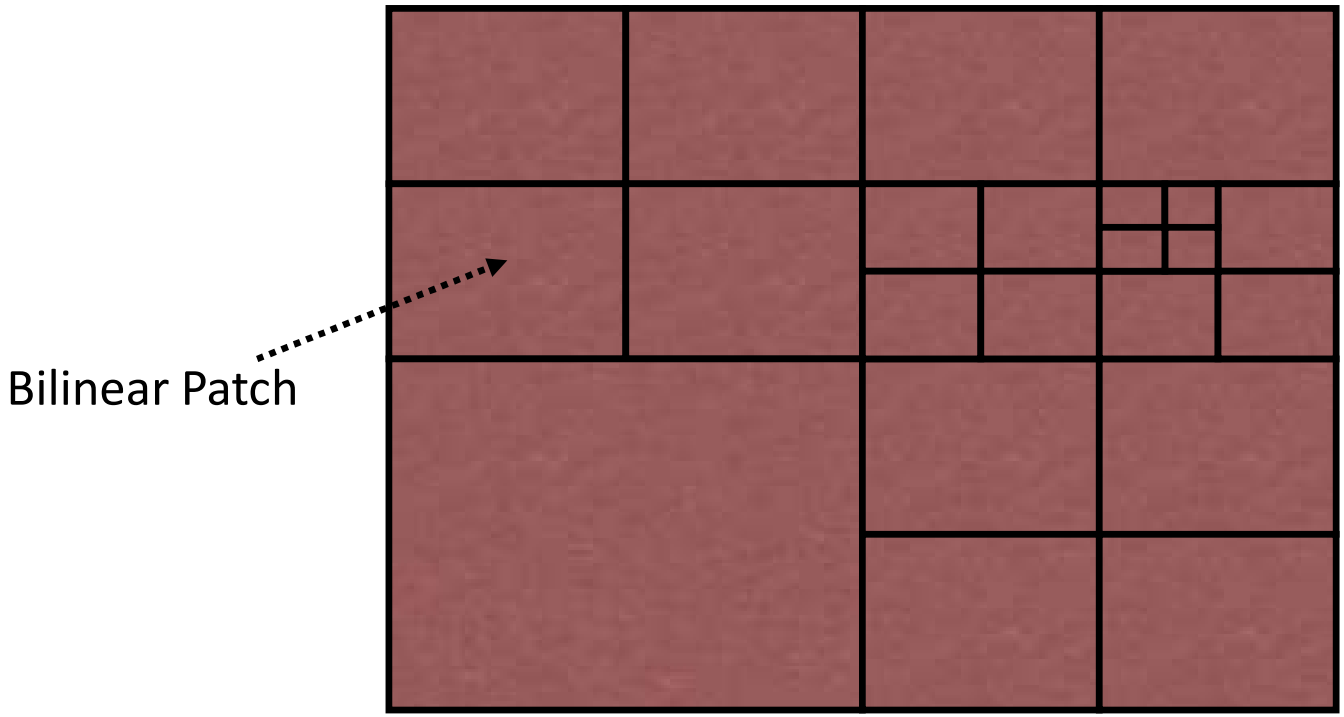


- Assume quadrilateral meshes

Rendering Terrain

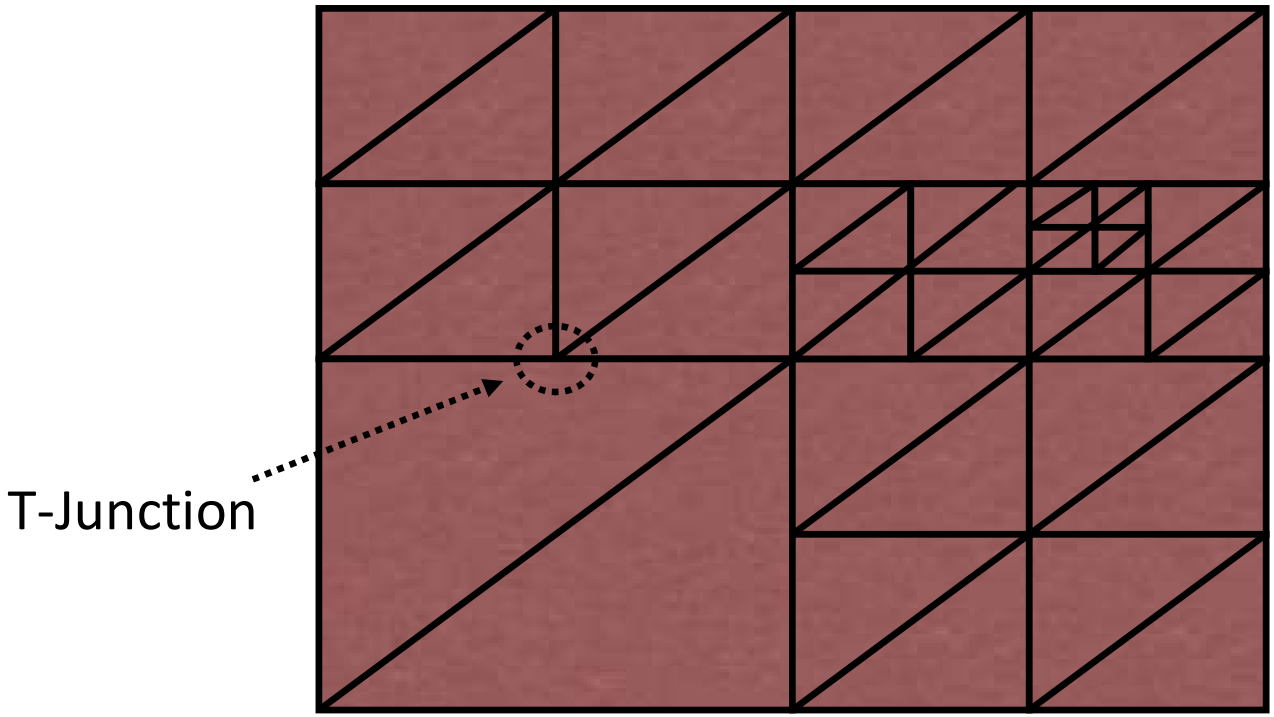


Rendering Terrain



Adaptive Representation

Rendering Terrain

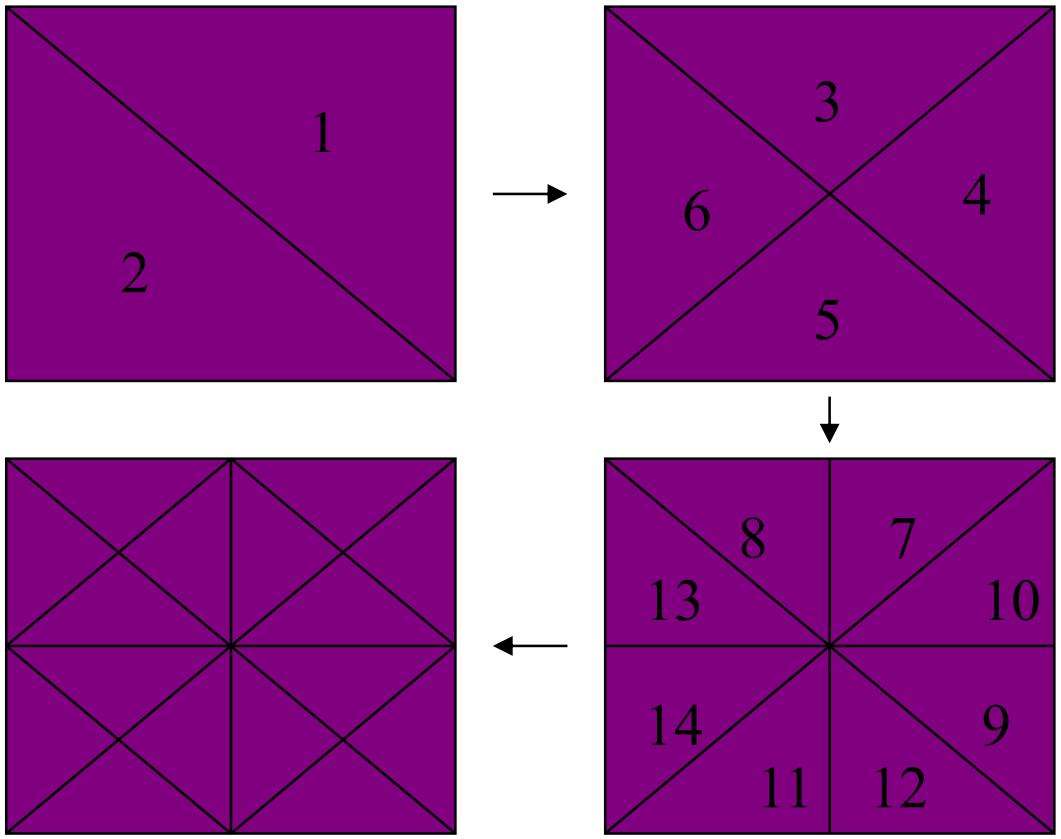


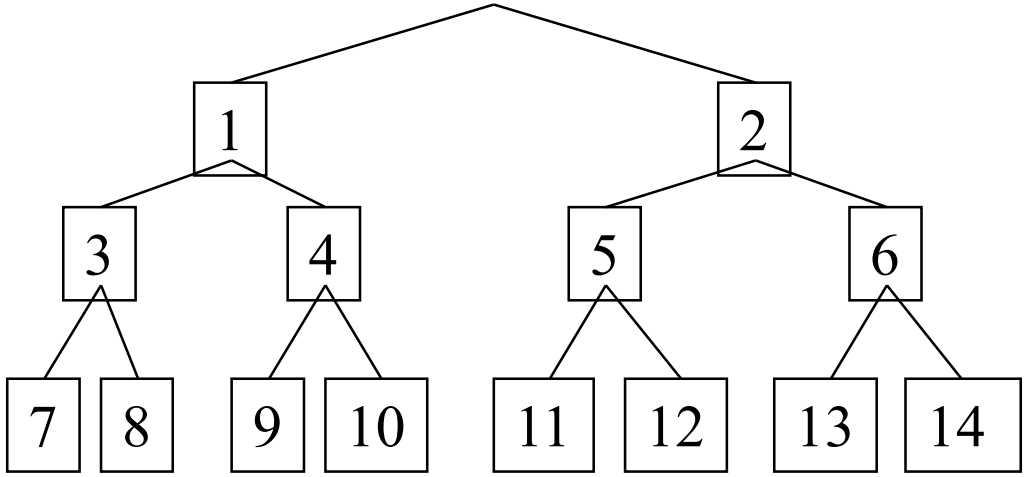
Adaptive Representation

Triangle Binary Trees (Bintree)

- Bintrees in which:
  - Each node represents a right-angled isosceles triangle
  - Each node has two children formed by splitting from the right angle vertex to the midpoint of the baseline
  - The leaf nodes use vertices from the original height field
- Another way is to build a spatial partitioning tree, but particularly well suited to simplification algorithms:
  - Easy to maintain neighbor information
  - Easy to avoid T-vertices

Triangle Bintree Example

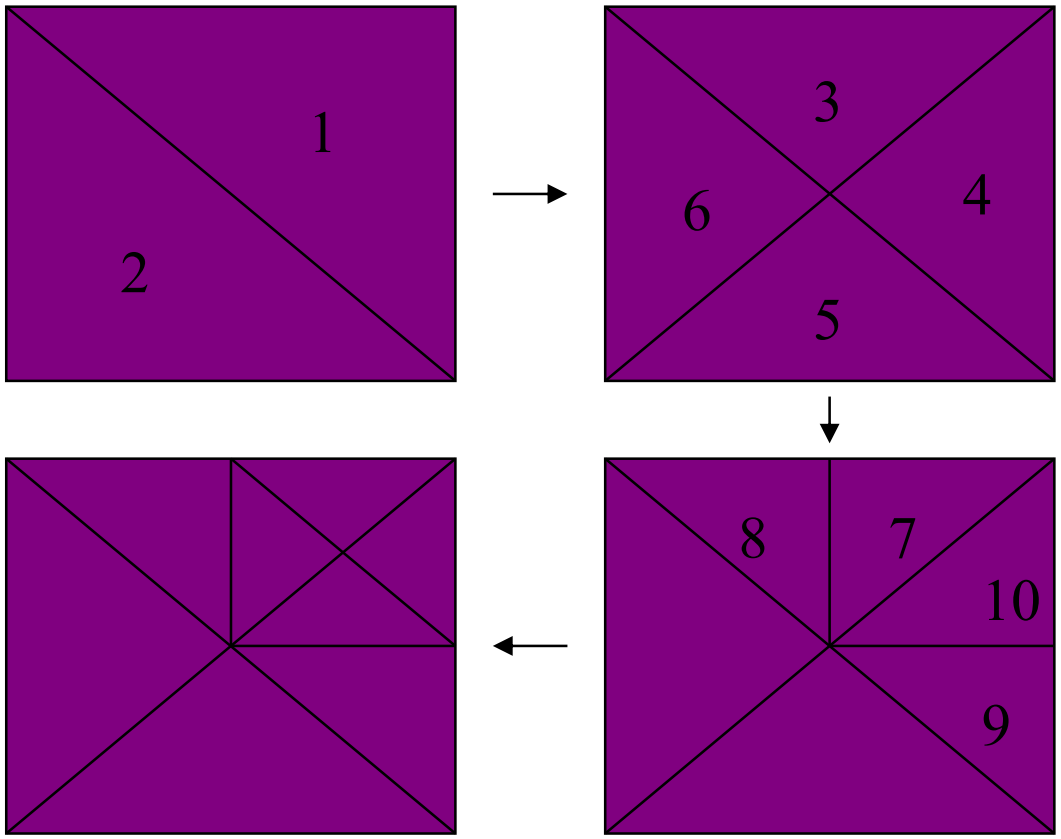


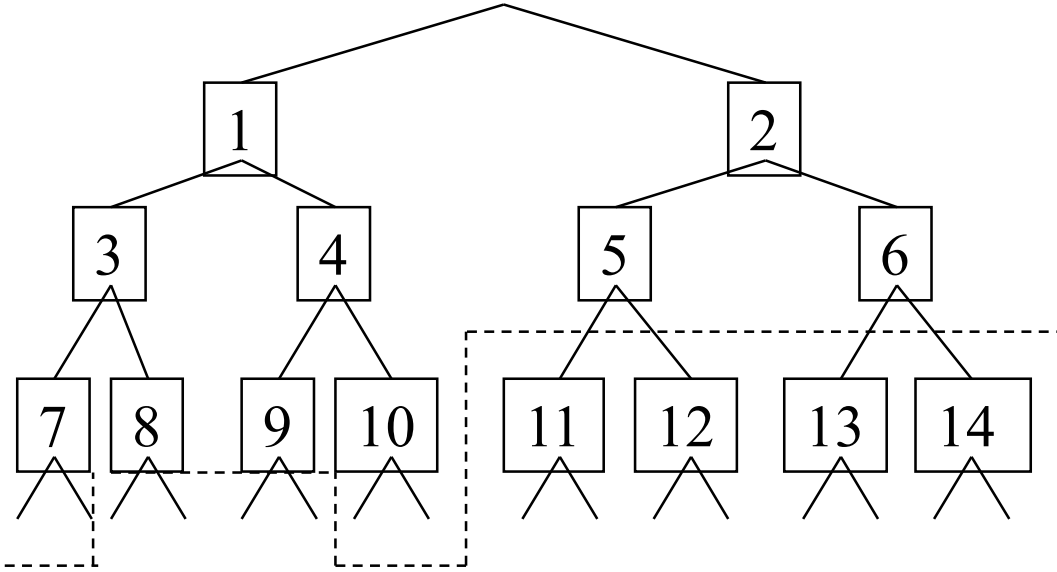


Bintree Data Structure

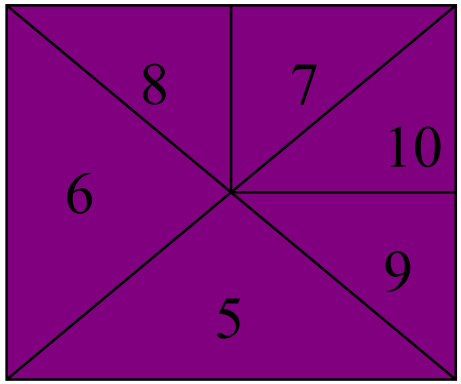
- Parent and child pointers
- Neighbors:
  - A left neighbor, a right neighbor, and a base neighbor
  - Note that the base and right angle give us a way to orient the triangle
  - Neighbors are not necessarily at your own level
- Later, error bounds that say how much variation in height there is in your children

Cuts





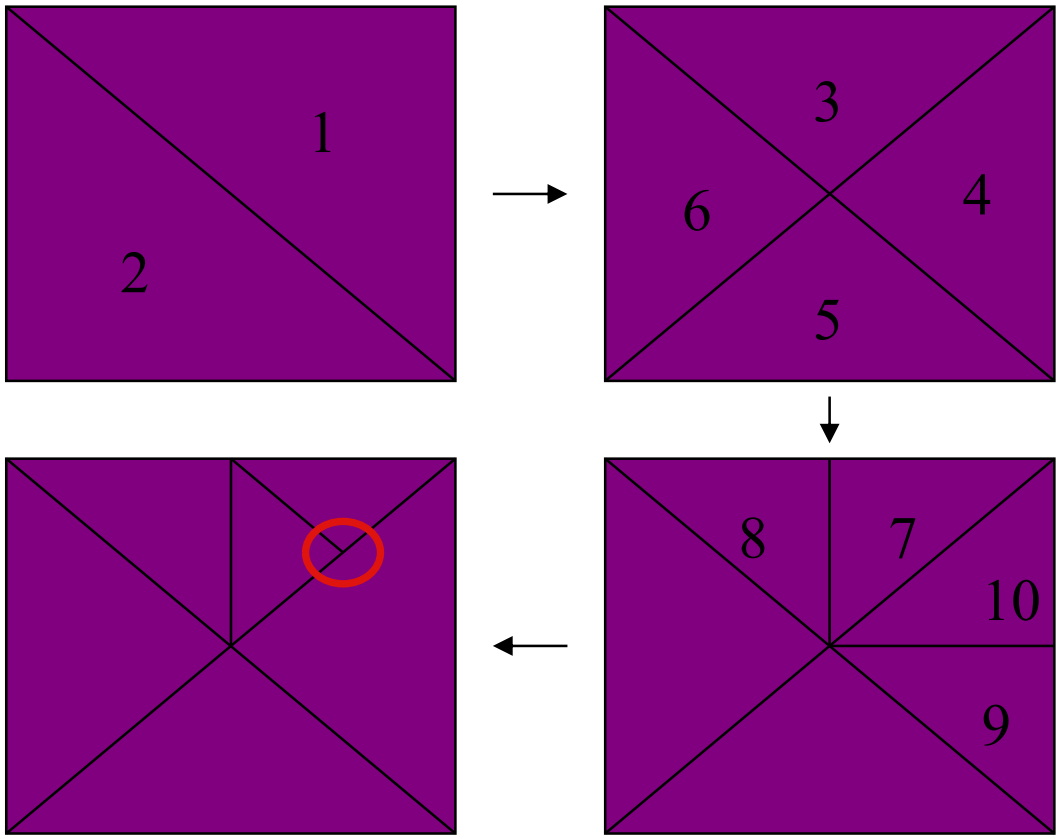
Neighbours

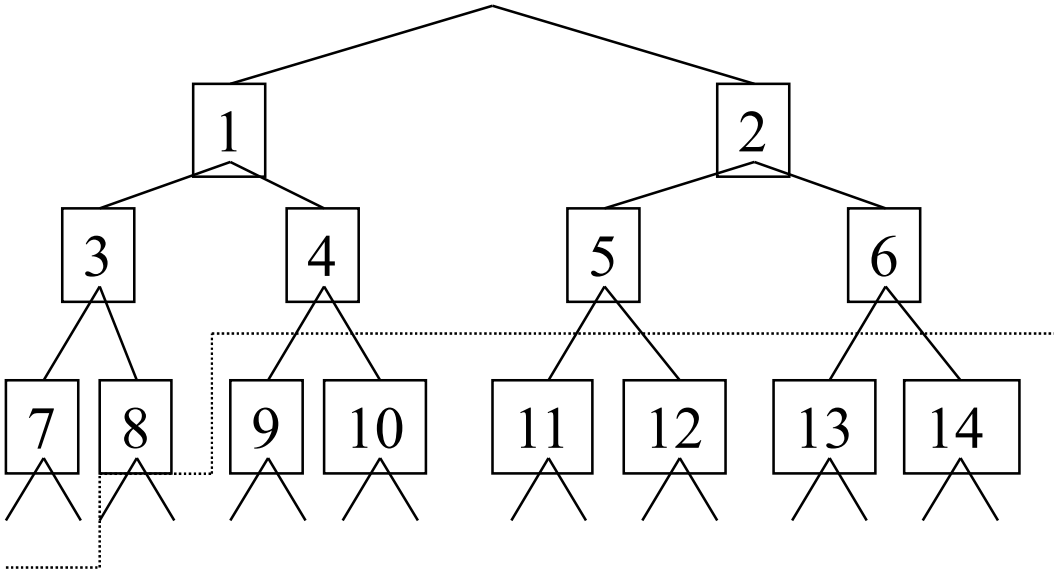


- 5: left neighbor 6, right neighbor 9
- 6: left neighbor 8, right neighbor 5
- 7: left neighbor 8, base neighbor 10
- 8: base neighbor 6, right neighbor 7
- 9: base neighbor 5, left neighbor 10
- 10: base neighbor 7, right neighbor 9

- Note that 8 is 6's left neighbor but 6 is 8's base neighbor
  - If you are someone's left/right/base neighbor they are not always your right/left/base neighbor
- In other words, neighbors need not come from the same level in the tree

Cuts are not always equal!





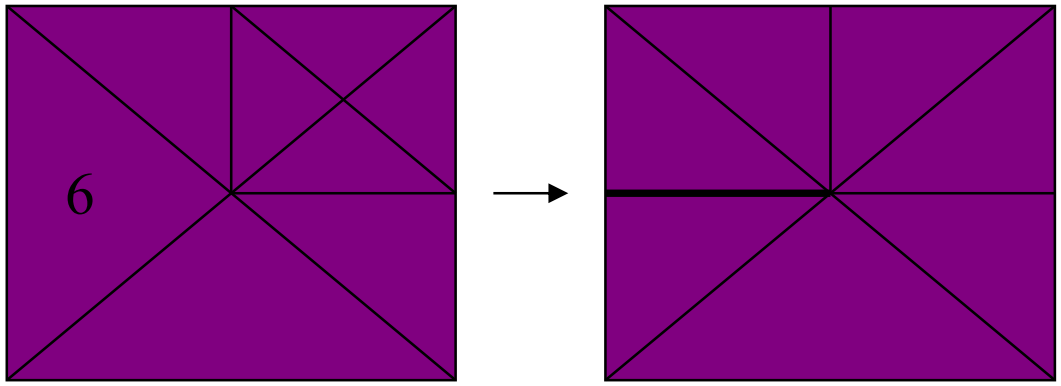
Note the T-vertex - causes cracks in rendering

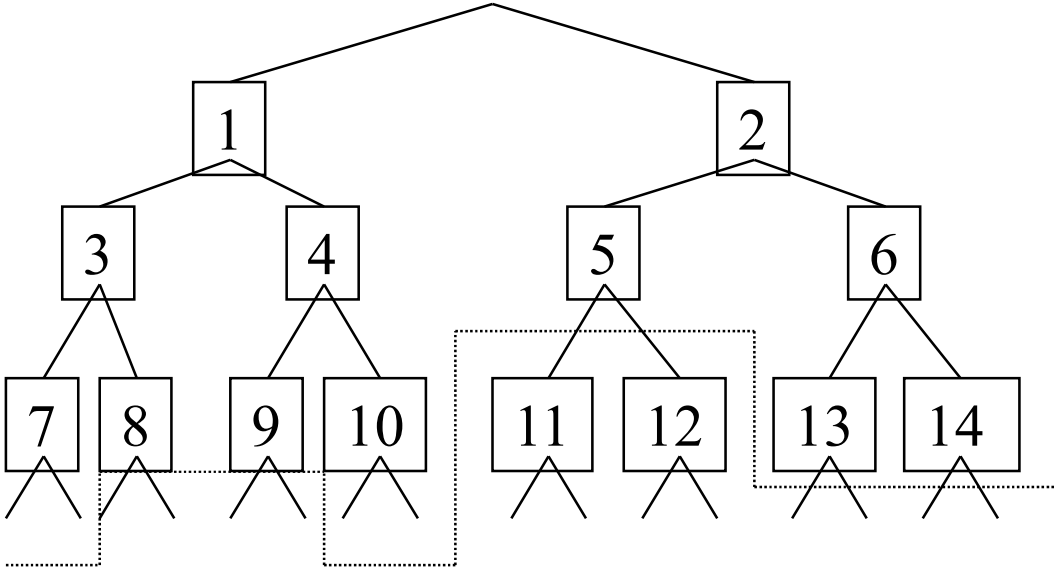
Generating Cuts

- Cuts are generated by a sequence of split or merge steps:
  - Split: Drop the cut below to include your children
  - Merge: Lift the cut up above two children
- To avoid T-vertices, some splits lead to other, *forced*, splits
- An LOD algorithm chooses which steps to apply to generate a particular triangle count or error rate

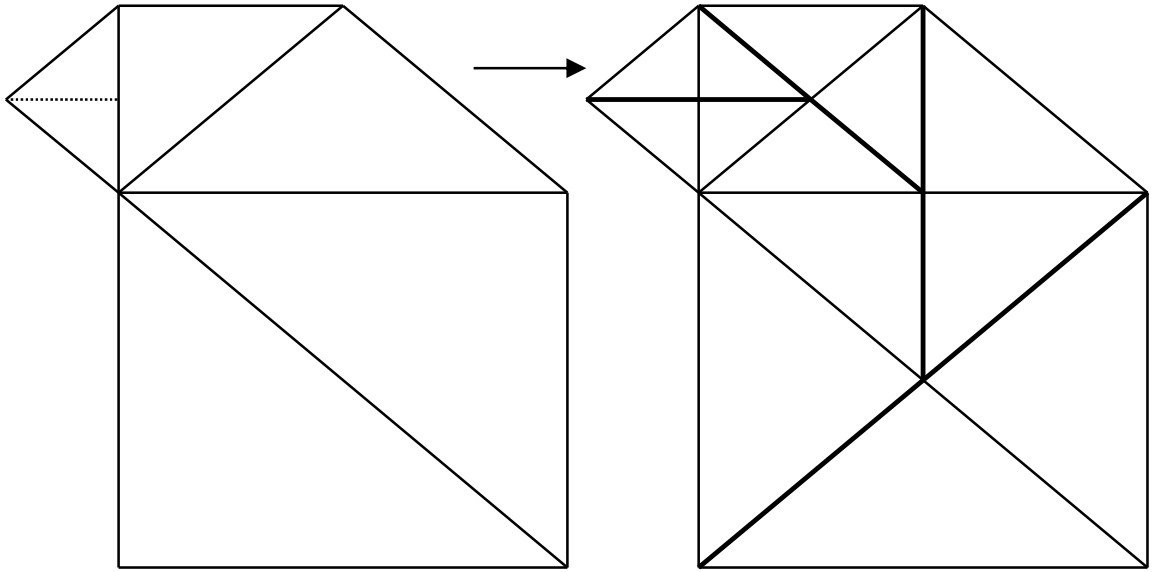
A Split

- A split cuts a triangle in two by splitting its base edge:
  - If the base edge is on a boundary, just split, as shown
  - If the base edge is shared, additional splits are forced
- Add a new triangle to the mesh





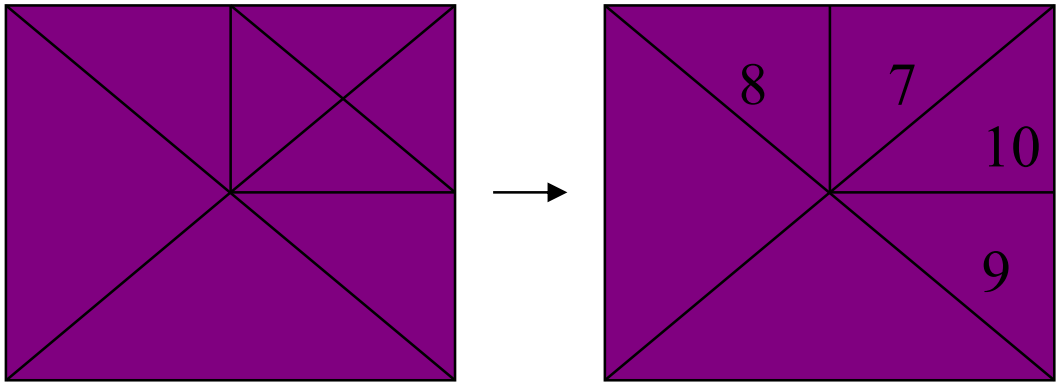
Forced Splits

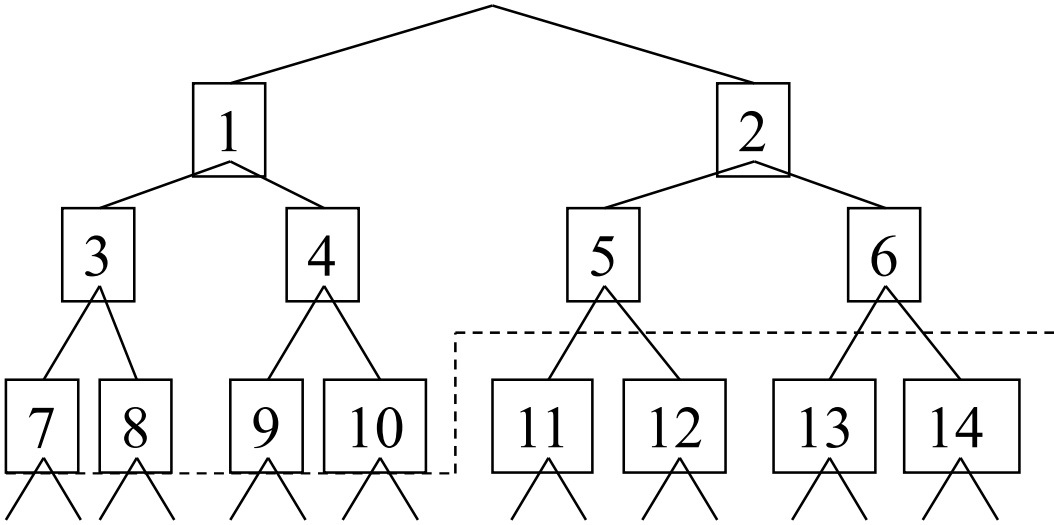


- Triangles are always split along their base
- Hence, must also be able to split the base neighbor
  - Requires neighbors to be mutual base neighbors
  - If they are not base neighbors, even more splits are needed
  - Simple recursive formulation

Merges

- A diamond is a merge candidate if the children of it's members are in the triangulation:
  - The children of the 7-10 diamond below are candidates
  - Look for parents of sibling leaf nodes that are base neighbors or have no base neighbors
- Reduces the triangle count





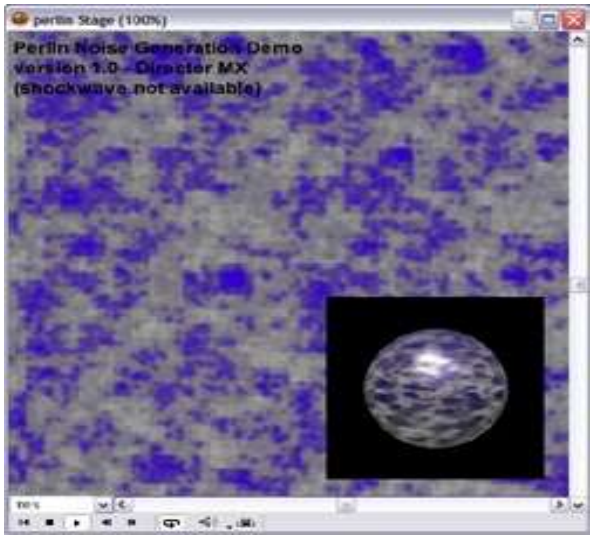
Refinement LOD Algorithm

- Start with the base mesh
- Repeatedly split triangles until done:
  - Stop when a specific triangle count is reached, or …
  - Stop when error is below some amount
- To guide the split order, assign priorities to each split and always do the one with the highest priority
  - After each split, update priorities of affected triangles
  - Sample priority: High priority to splits that will reduce big errors
- What is the complexity of this?
- A similar algorithm works by simplifying the mesh through merge operations. Why choose one over the other?

Fractal Terrain Details

- The original mesh vertices don't move, so it defines the overall shape of the terrain (mountains, valleys, etc)

- There are options for choosing where to move the new vertices
  - Uniform random offset
  - Normally distributed offset – small motions more likely
  - Procedural rule – eg *Perlin noise*
    - *making patterns from pseudo-random numbers*
- If desired, boundary vertices can be left unmoved, to maintain the boundary edge



Fractal Terrains

Very jagged terrain

Populating Terrain

- Coloring terrain:
  - Paint texture maps
  - Base color on height (with some randomness)

- Trees:
  - Paint densities, or randomly set density
  - Then place trees randomly within regions according to density

- Rivers (and lakes):
  - Trace local minima, and estimate catchment areas (more later…)

Media: Terrain Paint v3.0

Terrain Generation Trade-Offs

- Control versus Automation:
  - Painting gives most control
  - Fractal terrain next best control because you can always specify more points
  - Random methods give little control - generate lots and choose the one you like

- Generate on-the-fly:
  - Random points and fractal terrain could be generated on the fly, but fractal terrain generation is quite slow
  - Tiling's can also be generated on the fly

Media: Unity 3D Terrain with Forest by NS Design

Static LOD

- Depending on the roughness of the terrain and the application, 5%-50% of the vertices and triangles can be removed:
  - Consider: with 536,805,378 triangles MAYBE more than 200,000,000 triangles to draw in best case (rough surface).
- Frustum culling further reduces number of triangles to draw
- In most cases we still draw the terrain at full resolution near the far plane

View Dependent Dynamic LOD

- Dynamic simplification of visible part of the terrain
- A mountain observed from a distance of 10 km requires a higher tessellation than when observed from a distance of 100 km
- The quality of the tessellation can be changed at run time to achieve constant frame rates
- Terrains can be altered at run time

Terrain LOD

- Terrain poses problems for static LOD methods:
  - Must have high resolution in the near field, and low resolution in the distance, all in one model
- Dynamic LOD methods are the answer:
  - All based on the idea of *cuts* through a tree of potential simplifications
- ROAM algorithm is a good example:
  - Other continuous LOD algorithms are similar in style
- An alternative is to create fixed LODs for sub-regions and figure out how to join them together

Terrain LOD Algorithms

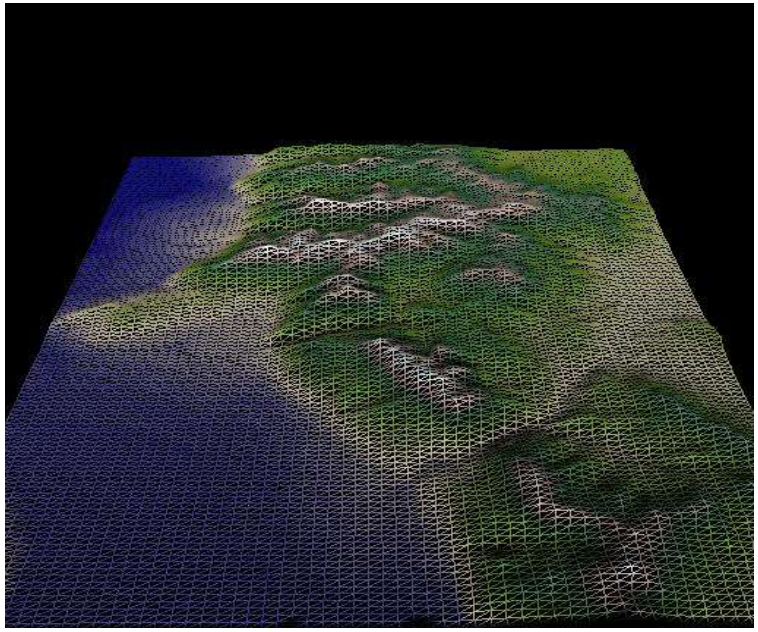
- Triangle bintree based
  - ‘ROAMing Terrain: Real-time Optimally Adapting Meshes’ – Duchaineau et al.
- Quad tree based
  - e.g. ‘Real-Time, Continuous Level of Detail Rendering of Height Fields’ – Lindstrom et al.
- Progressive mesh based
  - e.g. ‘Smooth View-Dependent Level-of-Detail Control and its Application to Terrain Rendering’ – Hoppe
- Geo Mip-mapping
  - ‘Fast Terrain Rendering Using Geometrical MipMapping’ – de Boer

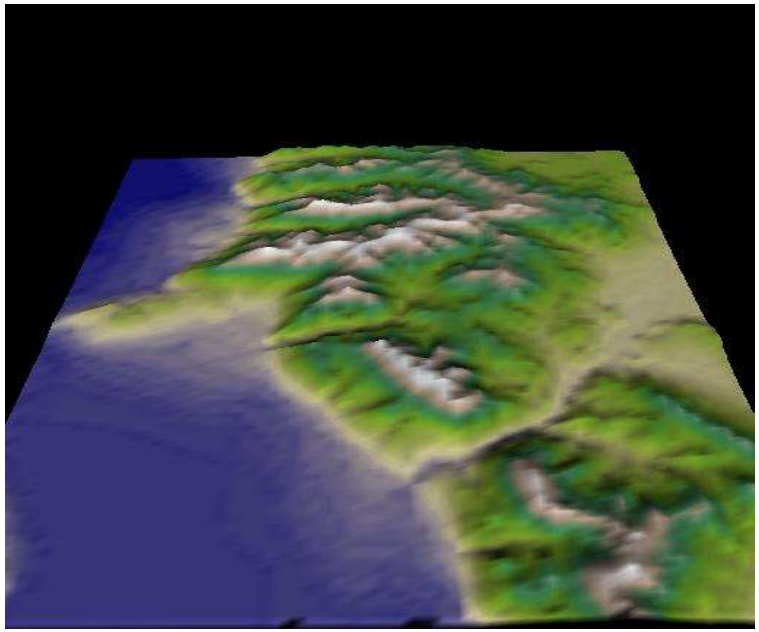
Rendering Fractal Landscape

- Alternative methods:
  - Polygon rendering using graphics hardware
    - Tesselate the height field
    - Draw each triangle separately or
    - Perform adaptive level-of-detail rendering
  - Ray-tracing
    - A) Tesselate the height field and trace the triangle mesh
    - B) Directly trace the fractal (QEAB)

Rendering Fractal Landscapes

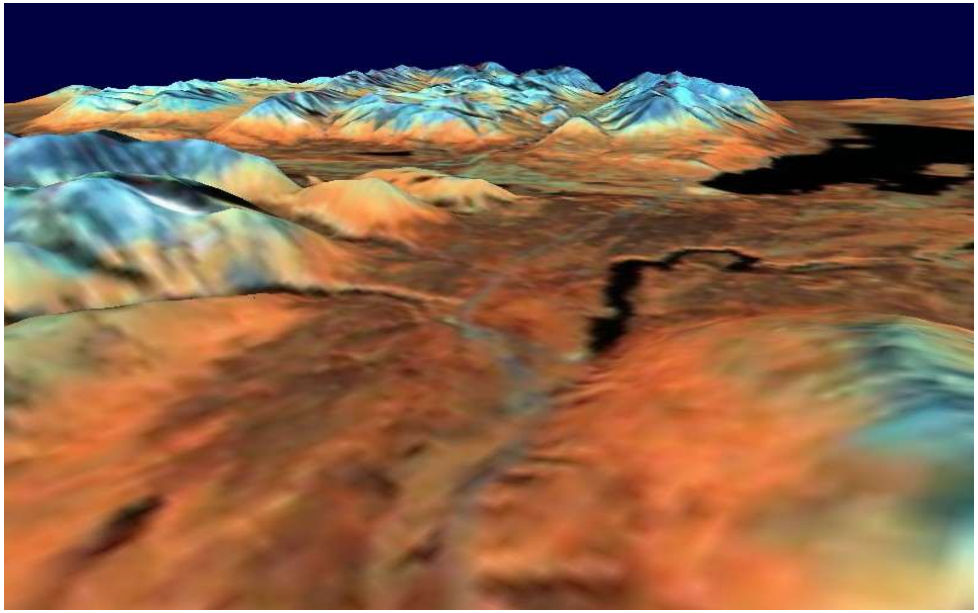
- Polygon rendering
  - Tesselation is given implicitely or can be generated
  - A height field over a rectangular domain, values are given at discrete sample points



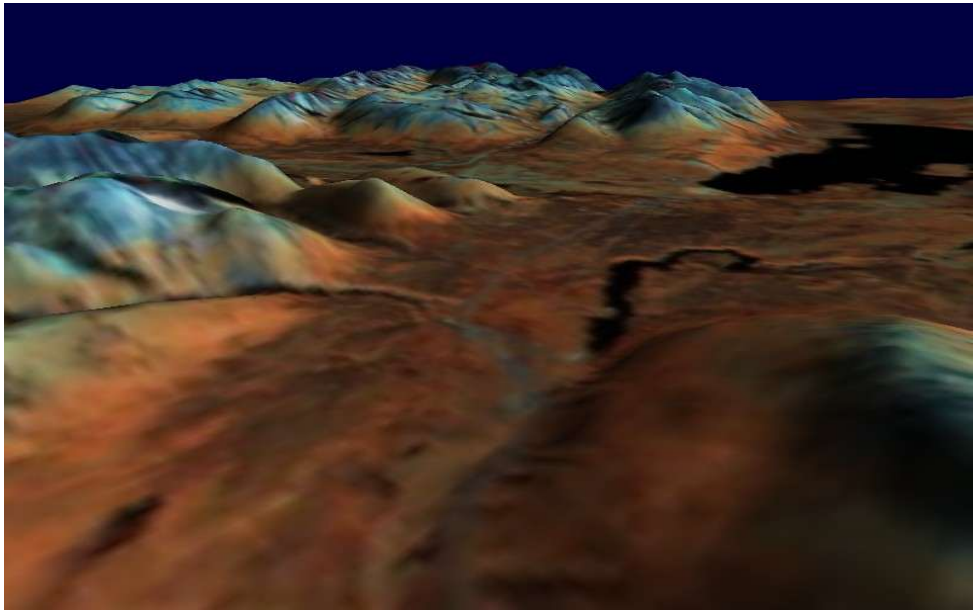


Rendering Fractal Landscapes

- Rendering lit and textured triangles
  - Vertices, colors, texture coords, normals



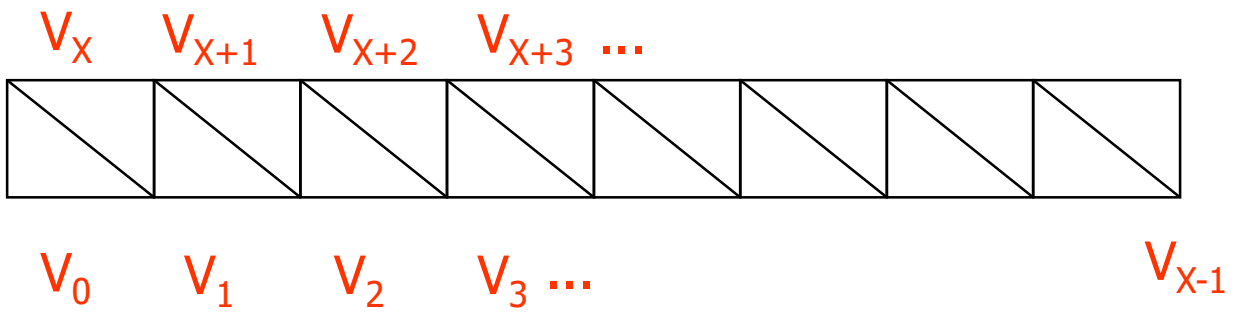
Textured only



Textured and shaded

Rendering Fractal Landscape

- Rendering triangles using graphics hardware
  - Render triangle strips: $v_0, v_X, v_1, v_{X+1}, v_2, v_{X+1}, \ldots$

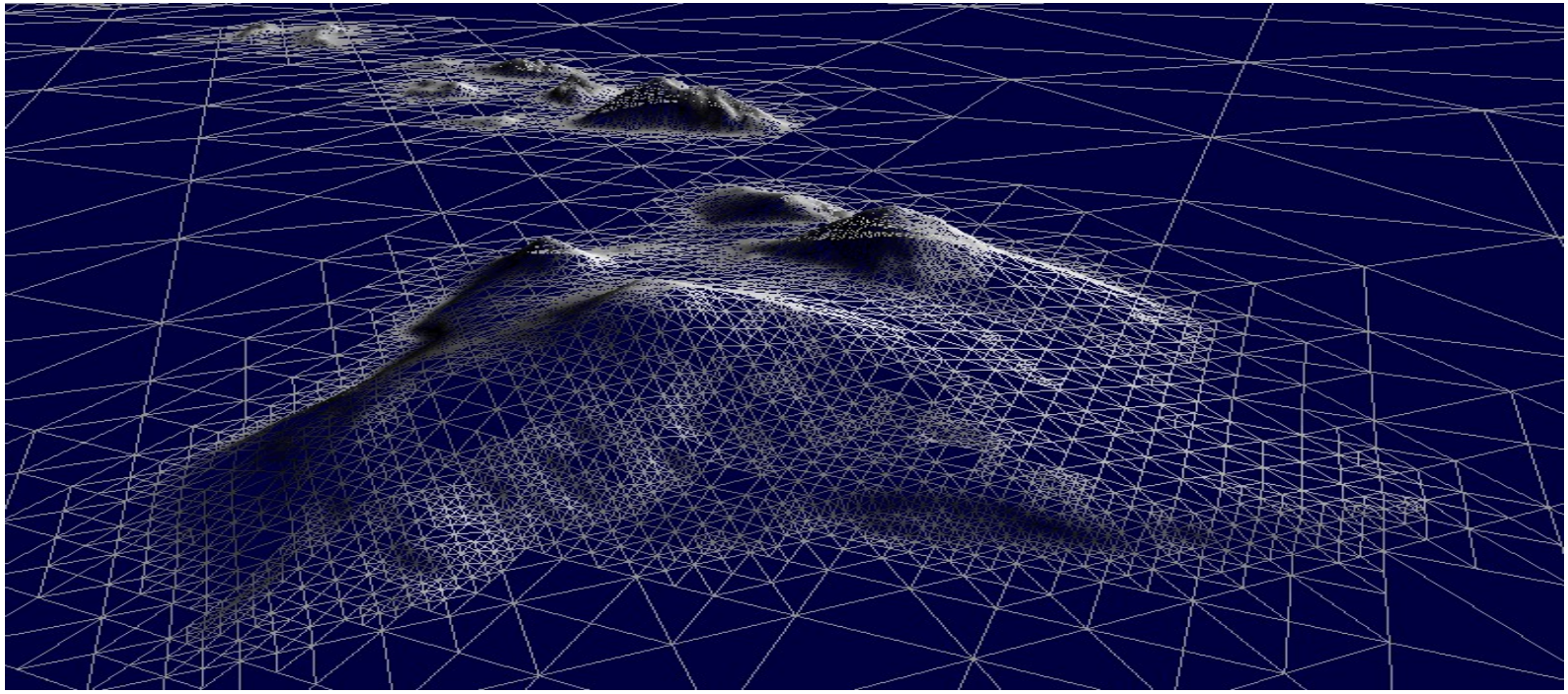


    - Triangle defined by new point and previous two points
    - Each vertex is rendered only once

  - Still have to send every vertex even in smooth regions
  - Popping artefacts where triangles below pixel size

Rendering Fractal Landscapes

- LOD-Rendering
  - Adaptively refine the mesh with regard to the current view

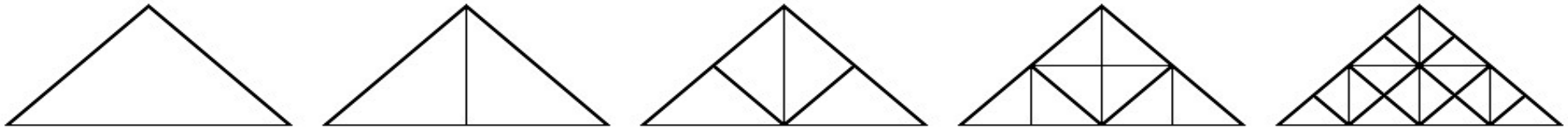


Rendering Fractal Landscapes

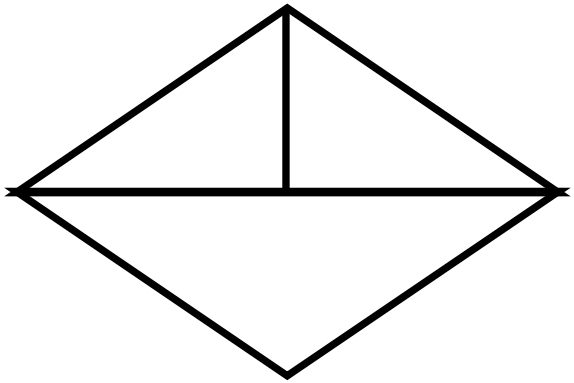
- LOD-Rendering:
  - Top-down or bottom-up approach
  - Hierarchically partition the mesh
  - Start with coarse resolution and adaptively refine until desired level-of-detail is reached
  - Start with original mesh and successively merge triangles
  - Consider world space and/or screen space deviation
  - Re-build mesh for every frame

Rendering Fractal Landscapes

- ROAM: Real-Time Optimally Adapting Meshes
  - Exploits triangle bintree structure: split along base edge



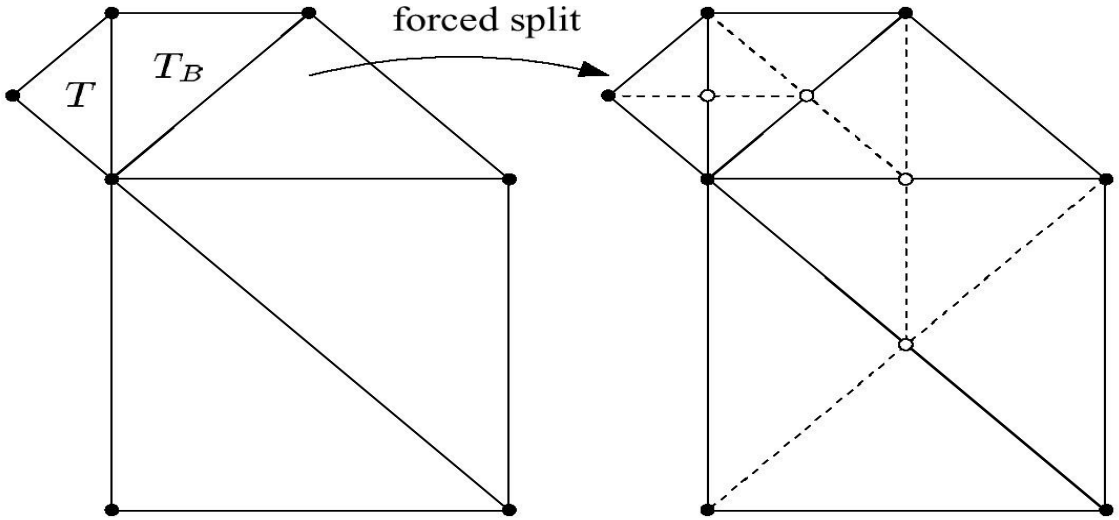
  - Valid triangulation has to be guaranteed
    - Two triangles have no overlap
    - Overlap at common vertex/common edge



Critical case

Rendering Fractal Landscapes

- The recursive split operation
  - Triangle to be split has coarser base neighbor
    - Force split of base neighbor first
    - Recursively force further splits until diamond is found
    - Diamond can be split without further splits



Rendering Fractal Landscapes

- Error criterion for split operation:
  - Deviation in world space
    - Difference between height value at center vertex and average of left and right vertex
    - Recursively pull deviation values from bottom to top
  - Deviation in screen space
    - Determine upper bound for screen space length of world space difference vector (0,d,0)

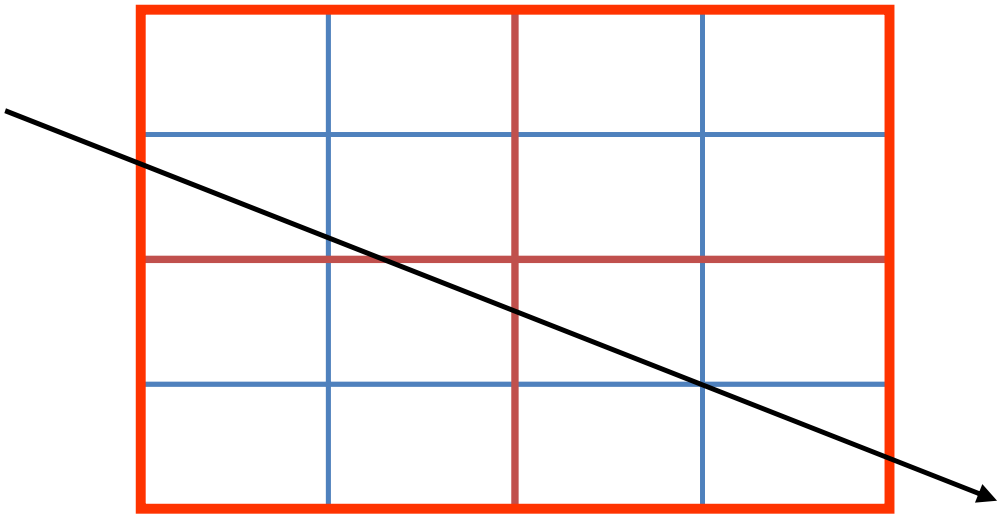
Rendering Fractal Landscapes

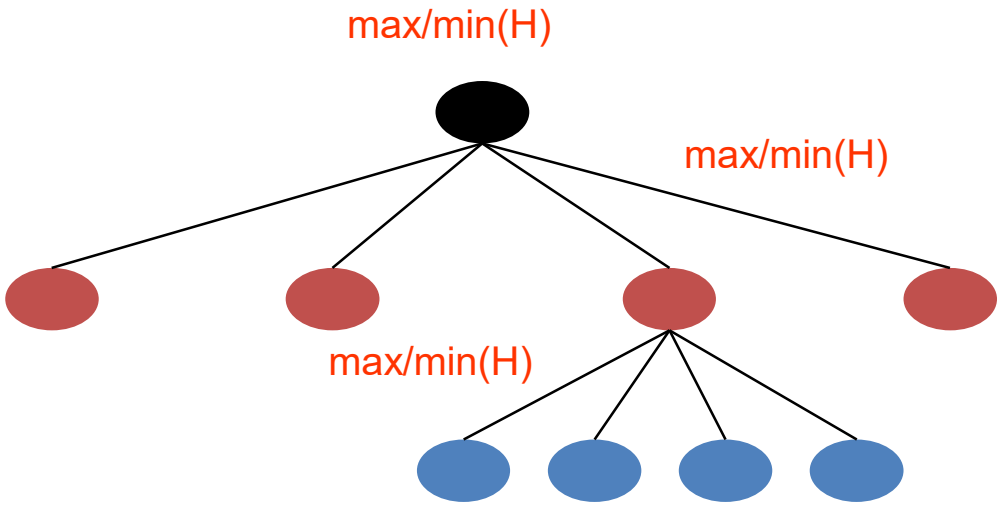
- Ray-Tracing
  - Ray-tracing triangle meshes
    - Trace rays until a triangle is hit

    - Implicit occlusion culling

Rendering Fractal Landscapes

- Ray-Tracing - performance issues
  - Mesh data structure has to be stored
  - Hierarchical representation necessary for improved intersection test
    - Octree or kD-tree





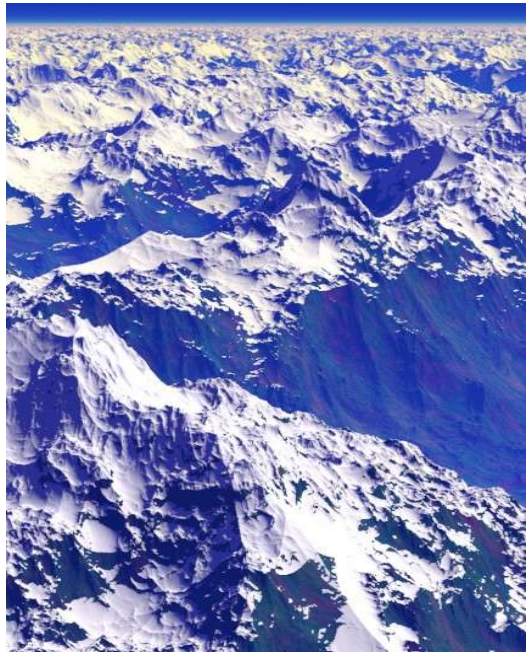
Store max/min heights within subregion

Skip regions below minimal height of ray

Employ ray coherences

Rendering Fractal Landscapes

- Rendering polygonal models – analysis
  - Aliasing due to undersampling of small features
  - No exploitation of the fractal lod-nature



Other Issues

- Terrain Texturing
- Terrain Lighting
- Camera Animation and Fly-through
- SkyBox
- Terrain following (a form of collision)
  - Maintaining characters and objects on top of Terrain

Big Picture

- The creation, population, management, movement within (and on) and rendering of triangle based terrains is one of the most important issues in real-time graphics.

Media: Virtual Worlds – Scotland Terrain Mesh

End