Computer Graphics for Digital Games
Video Game Technologies

14475: MSc in Computer Science and Engineering
13828: MSc in Game Design and Development

Chap. 4 — Terrain Modeling

Terrain Modeling

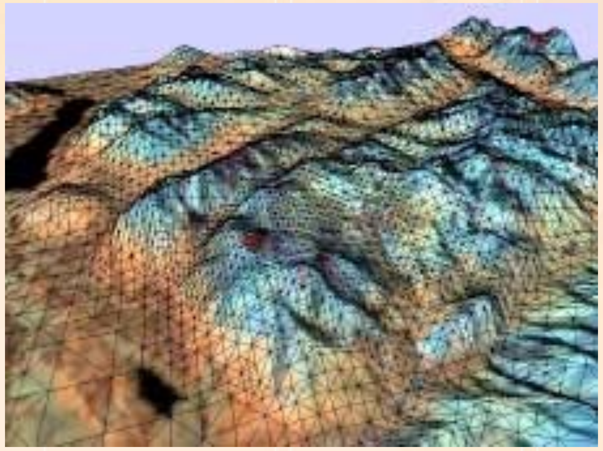
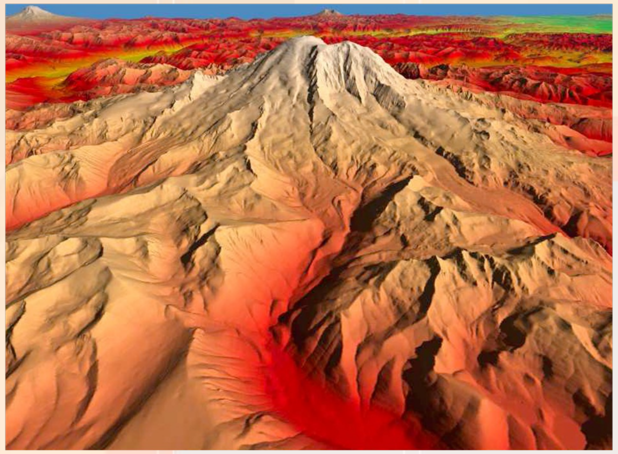
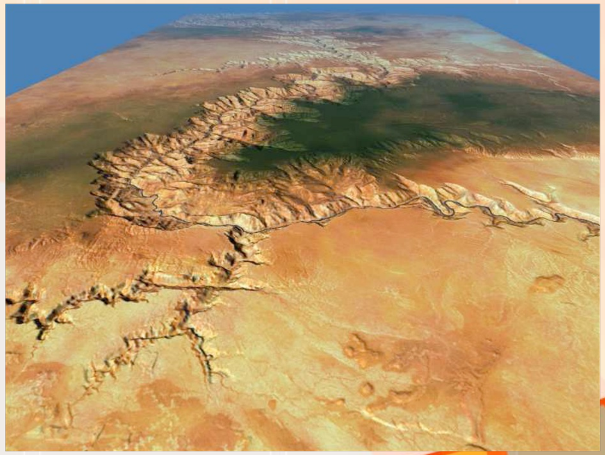
Overview

- Overview of terrain algorithms.
- Categories of terrain algorithms: discrete and continuous LODs.
- Issues in terrain generation and modeling.
- Classes of Continuous LODs: regular grids & heighfields, TINs, and voxel-based LODs.
- LEA algorithm.
- ROAM algorithm.
- Diamond-square algorithm.
- Final considerations.

http://www.cs.utah.edu/vissim/terrain.html

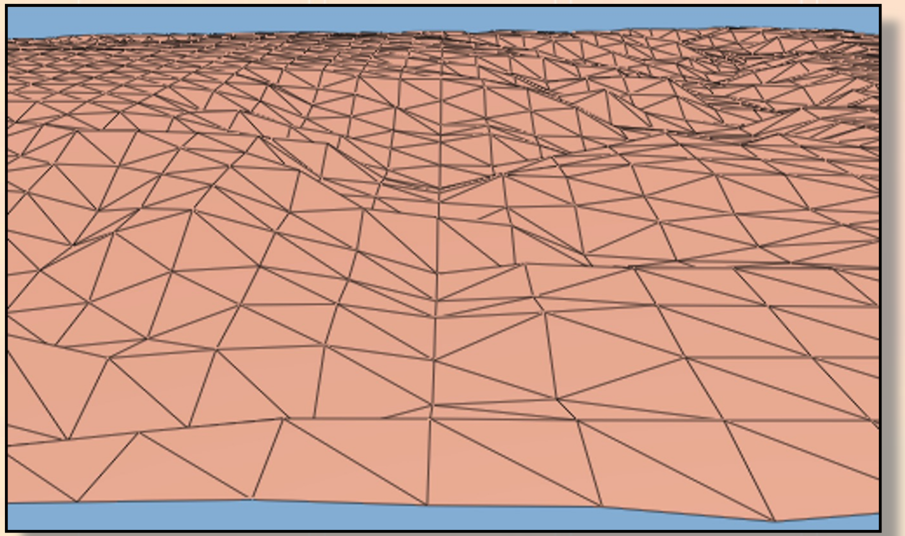
Introduction / motivation

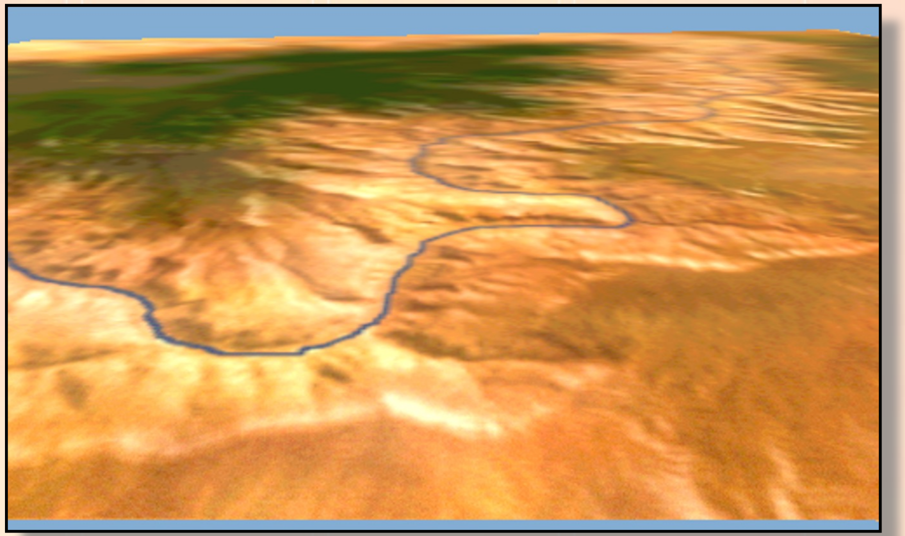
- Terrain is obviously important to many games
- As a model, it is very large
- Creating every point explicitly by hand is not feasible, so automated *terrain generation* methods are common
- When rendering, some of the terrain is close, and other parts are far away, leading to *terrain LOD* algorithms

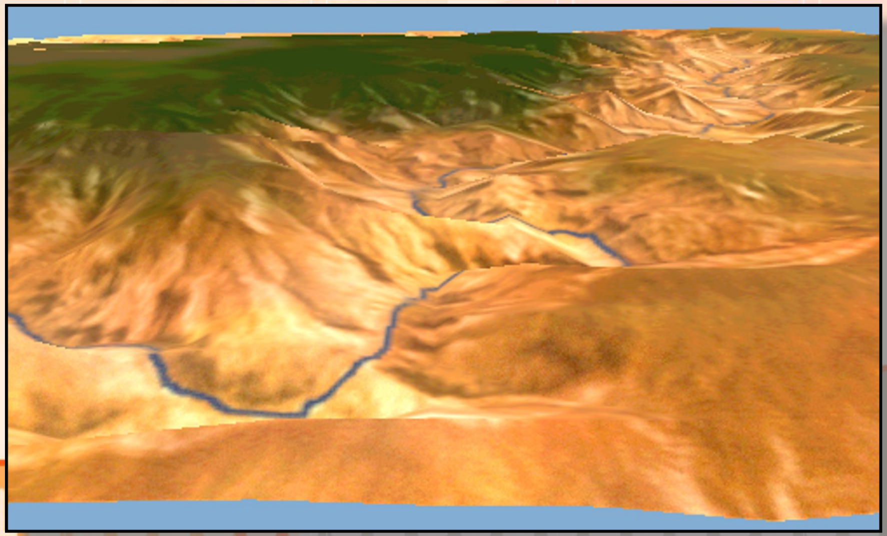




http://www.vterrain.org/LOD/Papers/

Terrain model: example

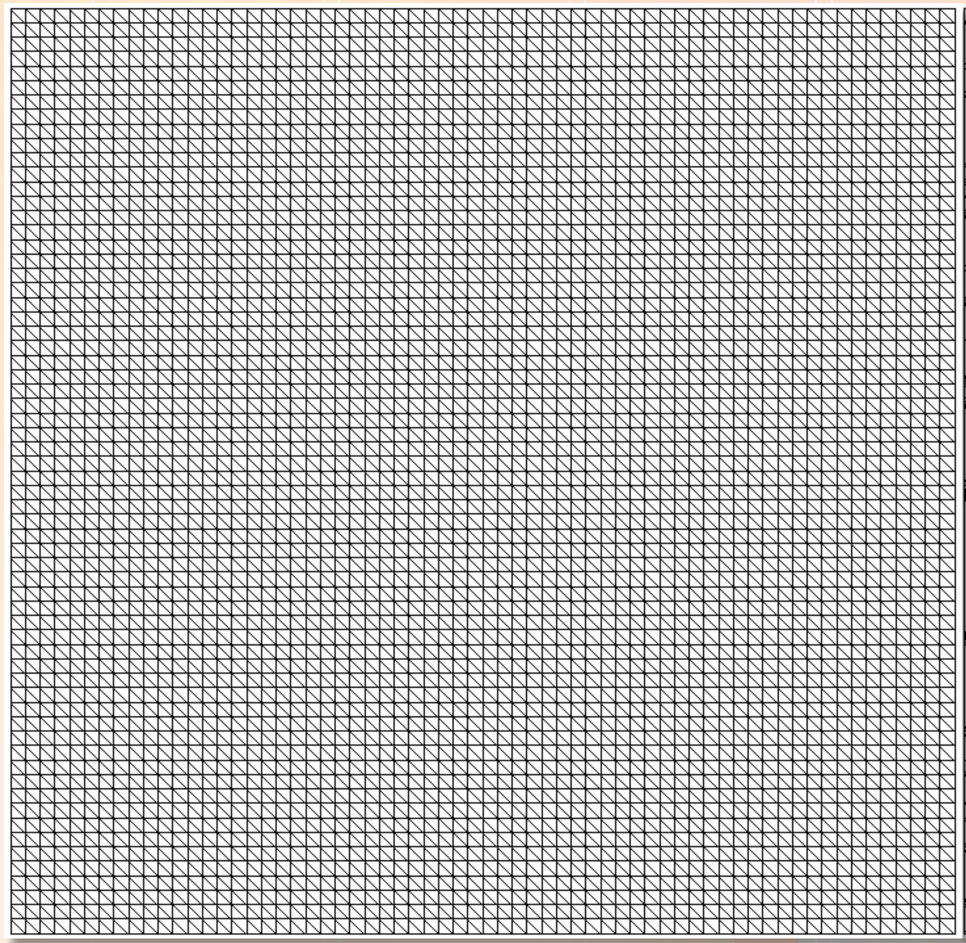

triangle mesh

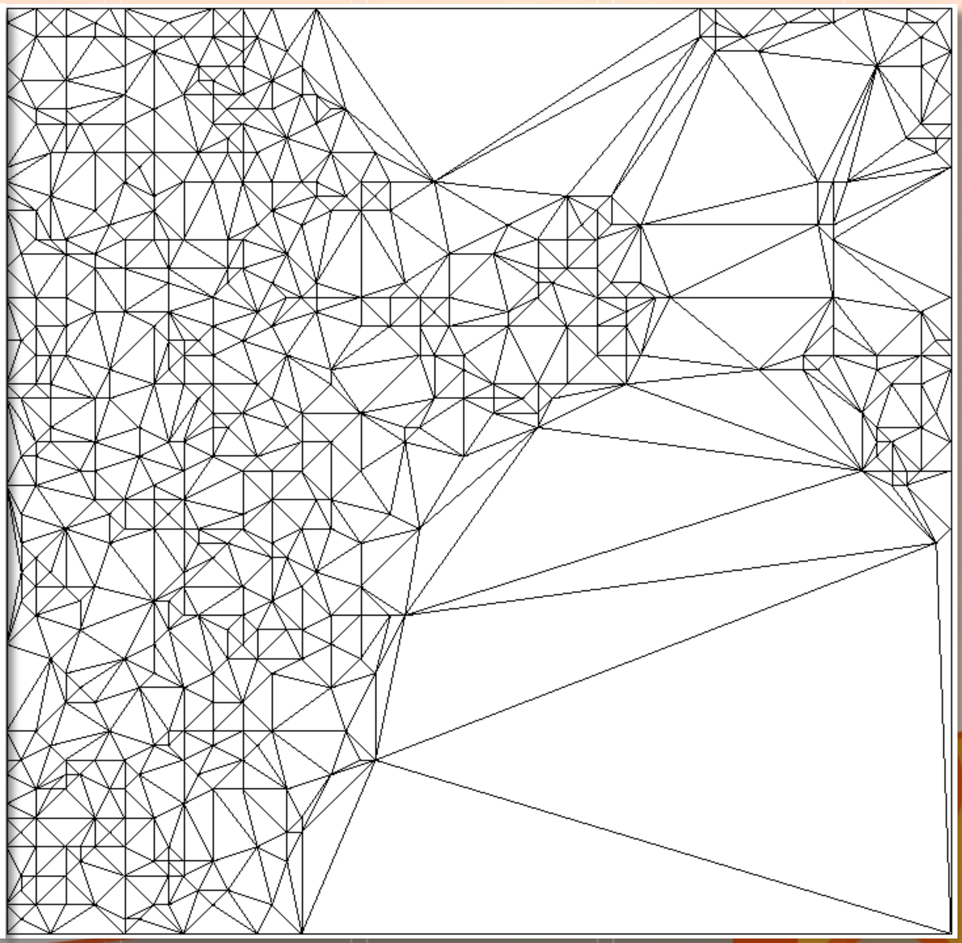

texture image



Types of data structures

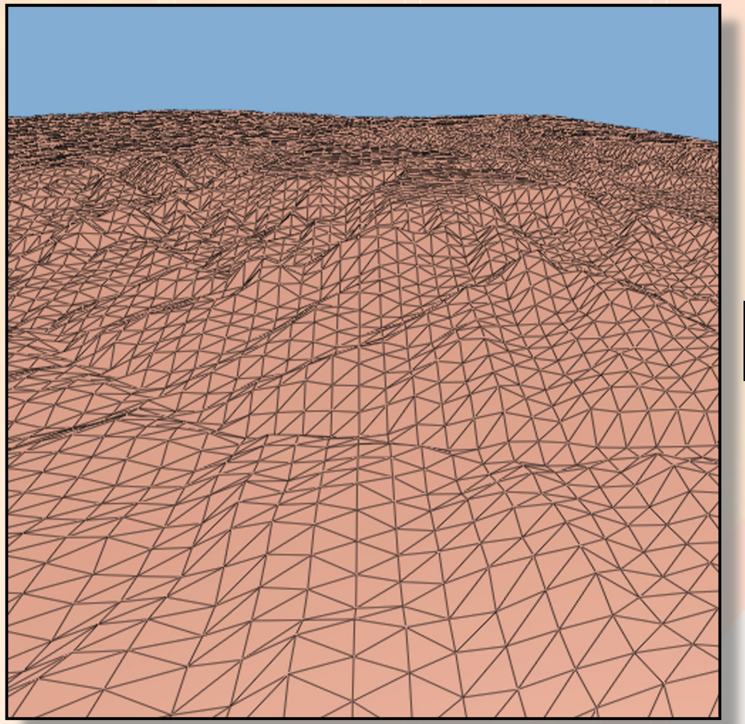
- Regular Grid + Height Field
- Triangulated Irregular Networks (TIN)



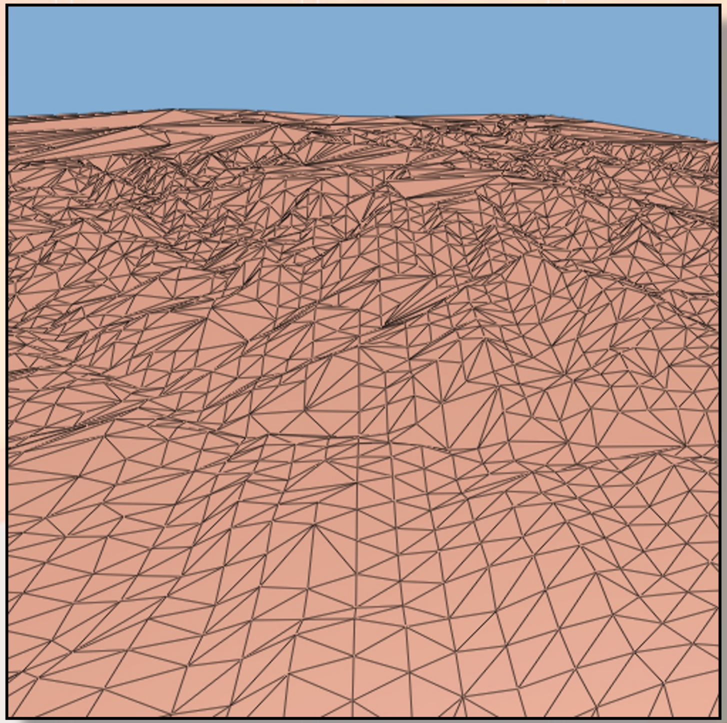


Mesh simplification

- Why do we need simplify a mesh?



Full Mesh



Simplified Mesh

Terrain algorithms: categories

- Discrete LOD

  Generate a handful of LOD meshes for each object

- Continuous LOD

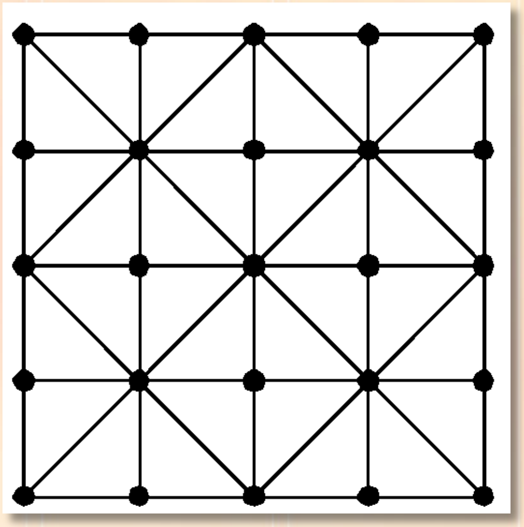
  Generate a single mesh for terrain from which a spectrum of detail can be extracted.

  – **Grid-based**:
    - continuous LOD rendering by Lindstrom
    - Real-time Optimally Adapting Meshes (ROAM)

  – **TIN-based**
    - progressive meshes

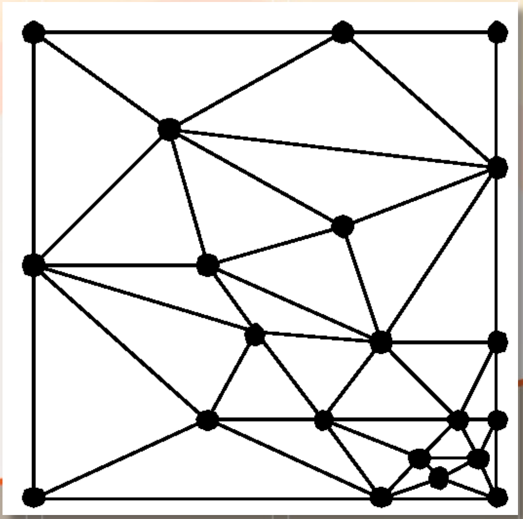
  – **Voxel-based**
    - marching cubes (transvoxel algorithm, 2010)

✸ Grid-based



$w(v)=(x,y,z(x,y))$

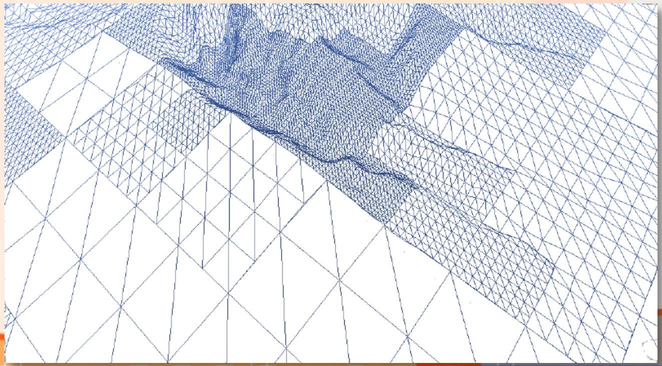
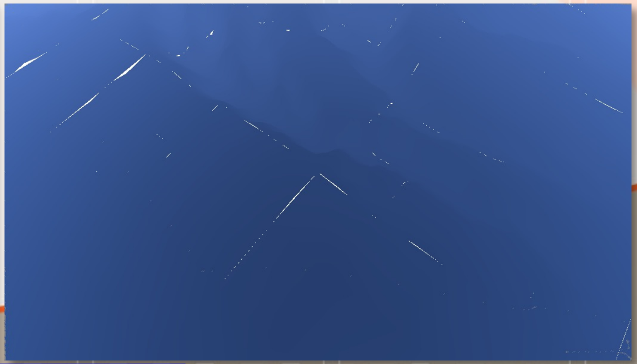
✸ TIN-based



$v=(x,y,z)$

Issues in terrain modeling

- ***Size***
  - Terrains are often modeled as huge meshes that may not fit in memory.
- ***Popping***
  - Terrain (vertex) *popping effect* as the level of detail changes;
  - This is particularly notorious for discrete LODs.
- ***Cracks***
  - Since the terrain consists of varying level of detail regions, when two regions of different detail levels meet, their edges do not match up perfectly
  - Cracks may jeopardize the pathfinding algorithms in the sense that a crack may be mistaken as a boundary of the game world.

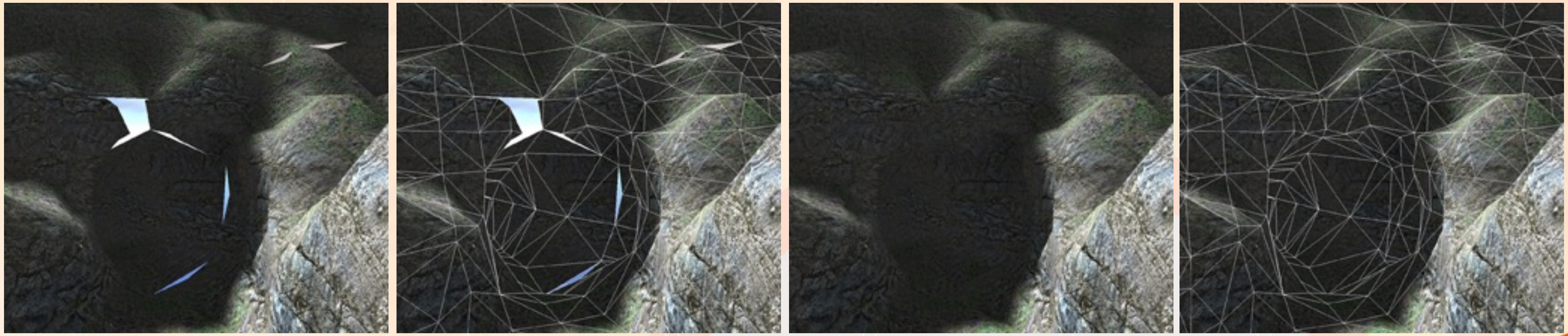


mesh with cracks

http://www.gamedev.net/topic/486286-terrain-rendering-simple-stitching-algorithm-to-avoid-cracks-picts-and-pseudo-code/

Issues in terrain modeling (cont'd.)

- ***Caves, overhangs, and arches***
  - Voxel-based terrain systems have been becoming popular because they rid off the topographical limitations of traditional elevation-based terrain systems, being so possible to create more complex structures like caves, overhangs, and arches..

http://www.terathon.com/voxels/



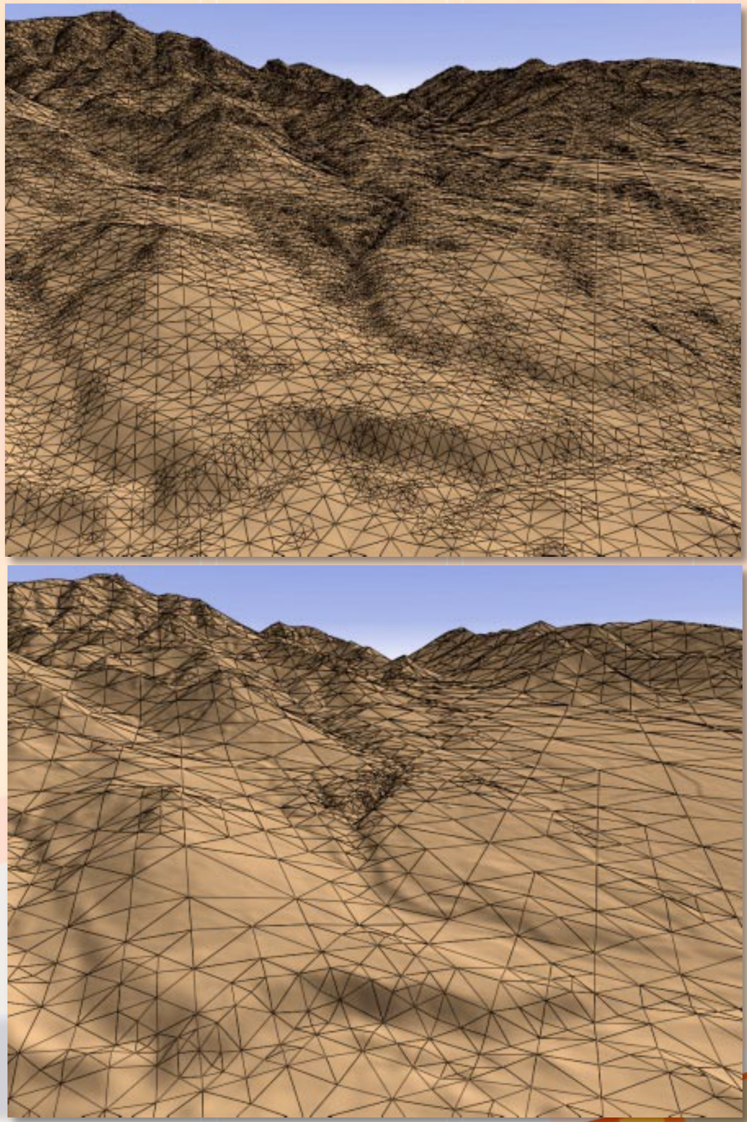
P. Lindstrom, D. Koller, W. Ribarsky, L. Hodges, N. Faust, and G. Turner. ***Real-Time, Continuous Level of Detail Rendering of Height Fields****. In Proceedings of the ACM SIGGRAPH'96, pp. 109-118, August 1996.*

Lindstrom et al. (LEA) Algorithm

http://www.gamasutra.com/view/feature/131841/continuous_lod_terrain_meshing_.php?print=1

LEA algorithm: key concepts

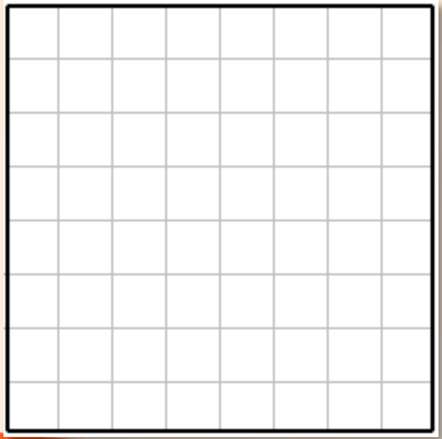
- Continuous LOD for height fields
- Uses a binary vertex tree
- Frame-to-frame coherence
- Introduced user-controllable screen space error threshold

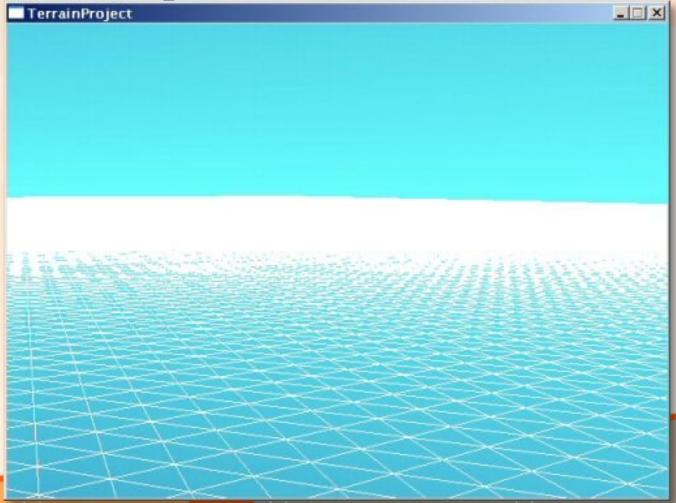

Terrain surface tessellations corresponding to projected geometric error thresholds of one (top) and four (below) pixels

LEA: bottom-up LOD algorithm

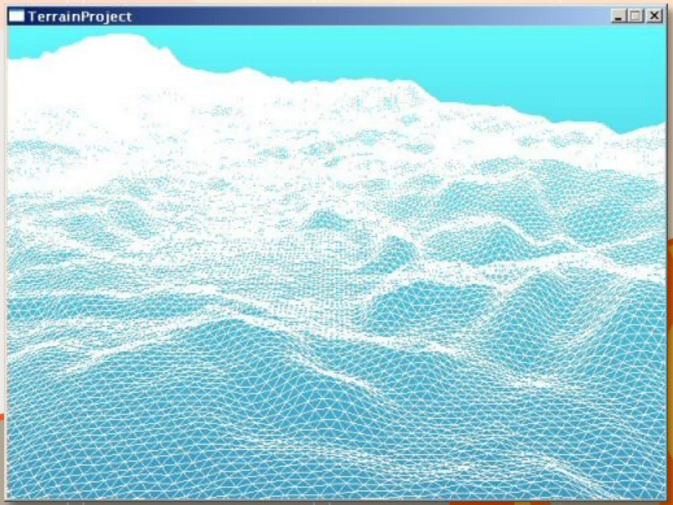
- The terrain is a heightfield represented by a squared 2D grid elevated from the x-y plane
- The surface corresponding to the heightfield (before simplification) is represented as a symmetric triangle mesh.
- We start from the most refined mesh to obtain a coarser mesh ➔ BOTTOM-UP.
- *Simplification algorithm*:
  - We are dealing with very large terrains; for example, a medium size grid 512 x 512 squares comes to 262,144 squares, each of which is divided into two triangles for rendering. So, we're looking at 524,288 total triangles. Each polygon is made up of three vertices (points), and so we send 1,572,864 vertices for rendering. That is a lot of data.

http://www.shamusyoung.com/twentysidedtale/?p=141


2D grid



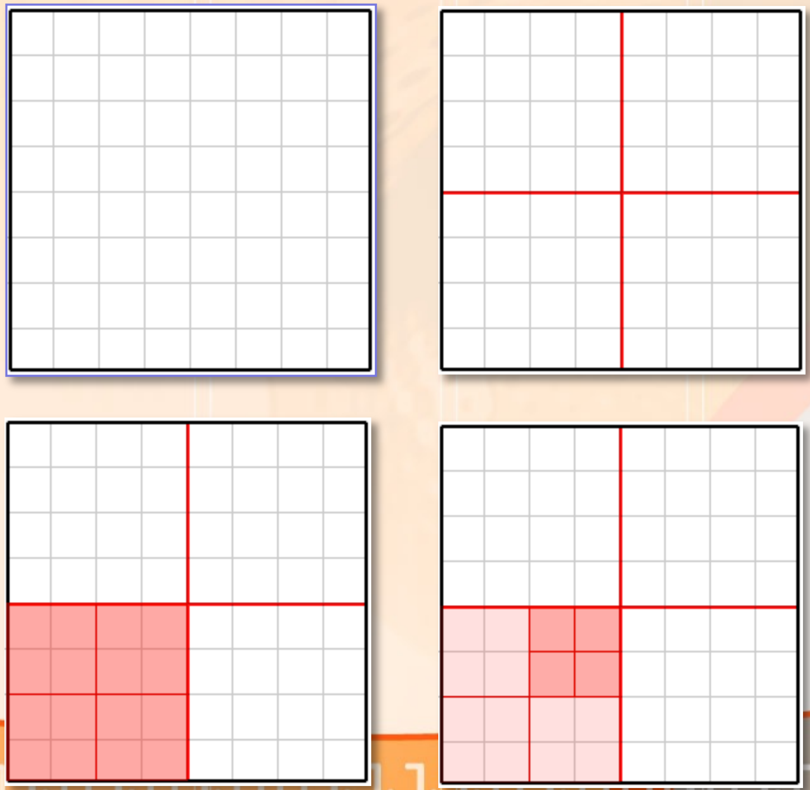
2D symmetric triangle grid

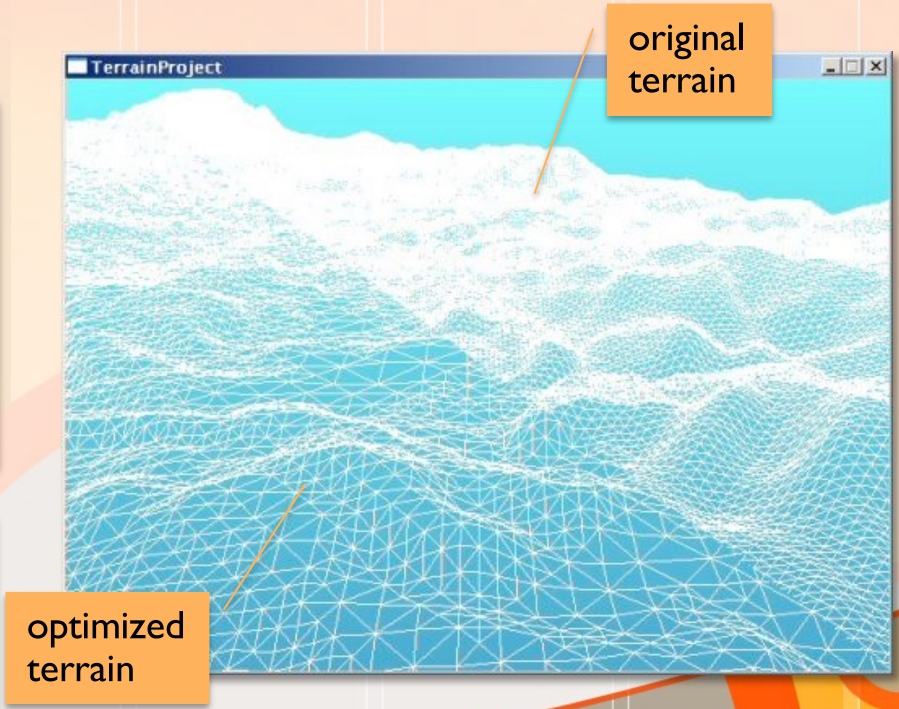


terrain as a heightfield

LEA's simplification algorithm:
1st reduction strategy of triangle mesh

- Criterion: **FLATNESS**
  - Reduce **flat areas** into few polygons, and keep complex areas (slopes) with more polygons ➔ GLOBAL STRATEGY
  - HOW? ➔ using QUADTREE data structure to represent the recursive partition of the terrain into quadrants.

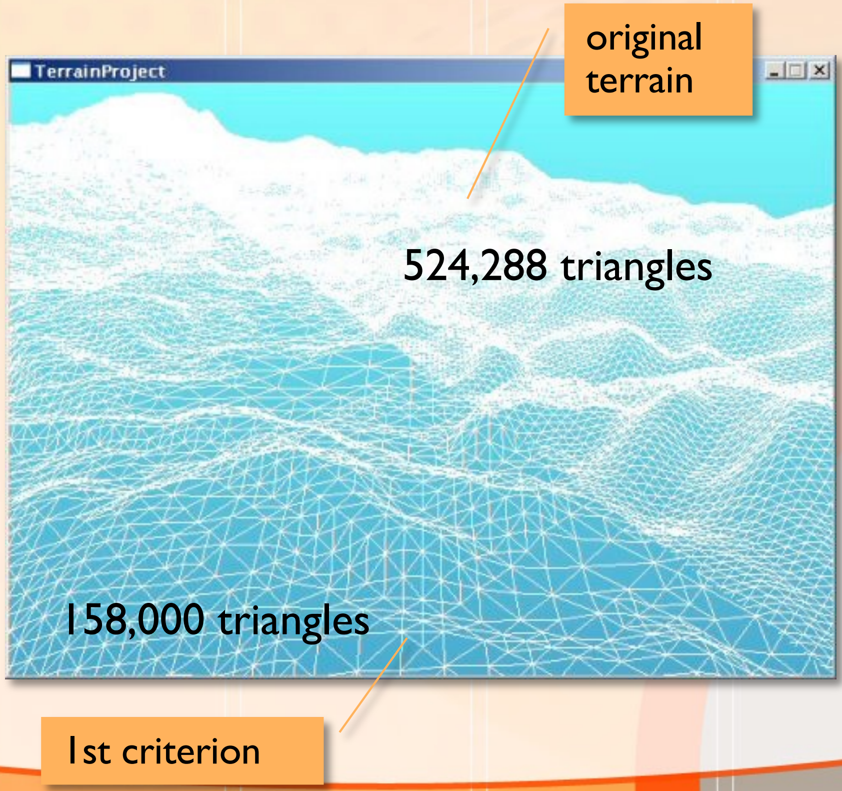


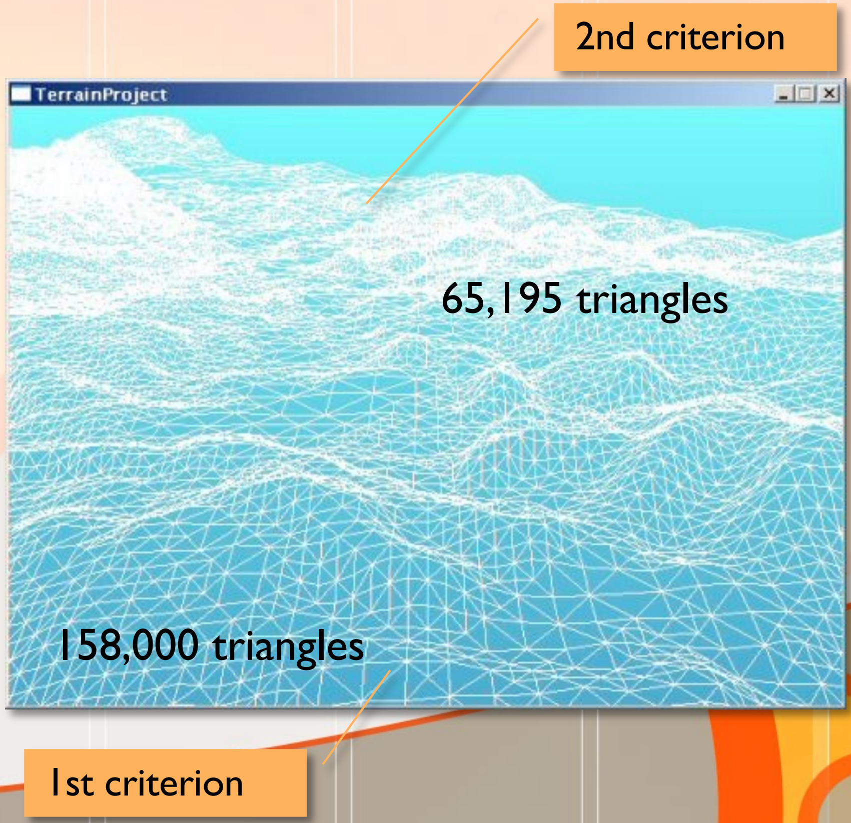


The new terrain is 158,000 triangles.
So, we've removed an amazing two-thirds of the triangles from the scene.

LEA's simplification algorithm:
2nd reduction strategy of triangle mesh

- <u>Criterion</u>: **DISTANCE**
  - With the distance to the viewer, a lot less polygons are necessary to pipeline into the graphics card. We're all the way down to 65,195 polygons! That is a reduction of about 88% in relation to the original terrain.



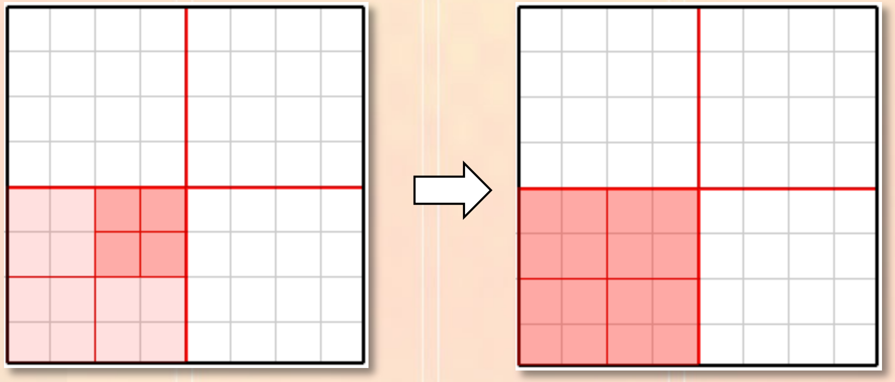


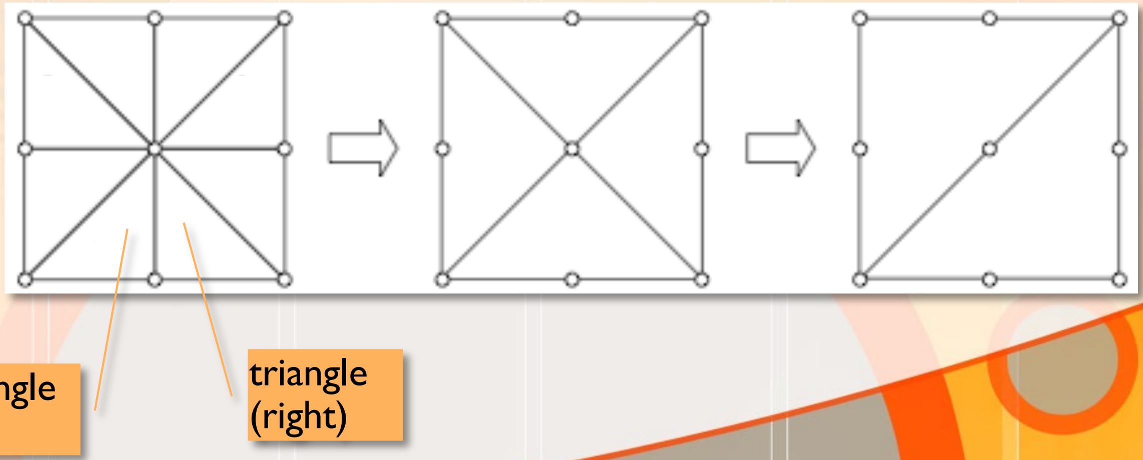
LEA's simplification algorithm:
3rd reduction strategy of triangle mesh

- Criterion: **POSITION**
  - Now that we have distance-based polygon reduction, we need the terrain to change whenever the user moves around.
  - As we fly around the terrain, there are little pauses every few seconds. Lots of games have this "stuttering" problem from time to time.
  - The problem is that we're doing the whole terrain in one go. We're sending the whole mesh to graphics system (e.g. OpenGL) at once, which just takes too long.
  - Solution: We can fix this problem by breaking the terrain into many separate sections and sending the terrain over to OpenGL a "visible" section at a time.

LEA's simplification algorithm in two steps

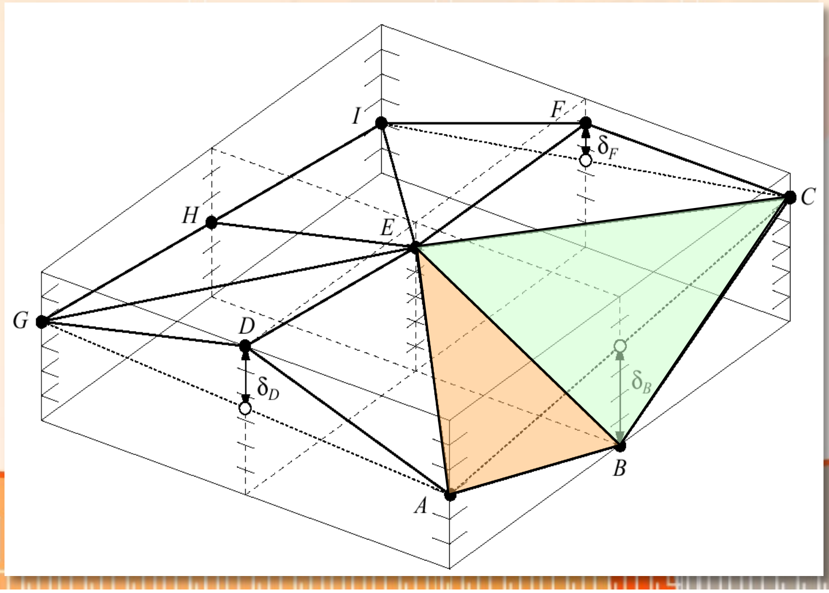
- Coarse-grained simplification (block-based simplification)
  - It operates at the quadtree level.
  - It serves to determine which discrete level of detail is needed for each block (or terrain region).
- Fine-grained simplification (vertex-based simplification)
  - It operates at the block level.
  - Coaslescence of triangle-cotriangle pairs into a single triangle.
  - Individual vertices are considered for removal so that their sharing triangles are replaced with fewer larger triangles.

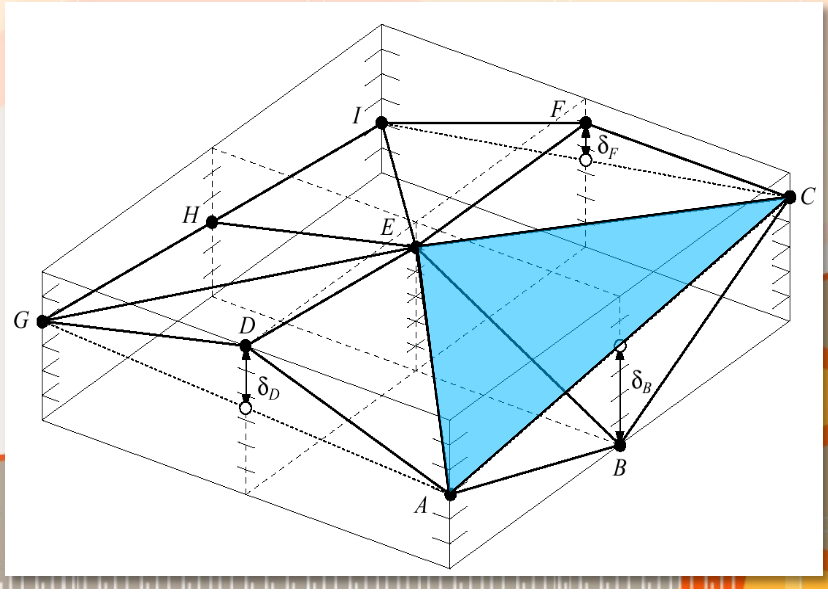




LEA's simplification algorithm: vertex-based simplification

- Criterion: **GEOMETRIC ERROR IN PIXELS**
  - It is based on geometric error in pixels. This error measures the change in slope between two triangles, i.e. FLATNESS VARIATION (see1st reduction criterion above).
  - EXAMPLE: For triangles ABE and BCE, one computes the delta value of B, which is given by vertical distance between B and the midpoint M of A and C; the segment BM is called delta segment.
  - As the delta value increases, the chance of triangle coaslescence decreases.
  - The geometric error is just given by the size of the delta segment in number of pixels when it is projected on screen.





Mark Duchaineau, Murray Wolinsky, David E. Sigeti, Mark C. Millery, Charles Aldrich, and Mark B. Mineev-Weinstein. ***ROAMing Terrain: Real-time Optimally Adapting Meshes****. In Proceedings of the 8th Conference on Visualization '97 (VIS'97), IEEE Computer Society Press, 1997.*

ROAM Algorithm

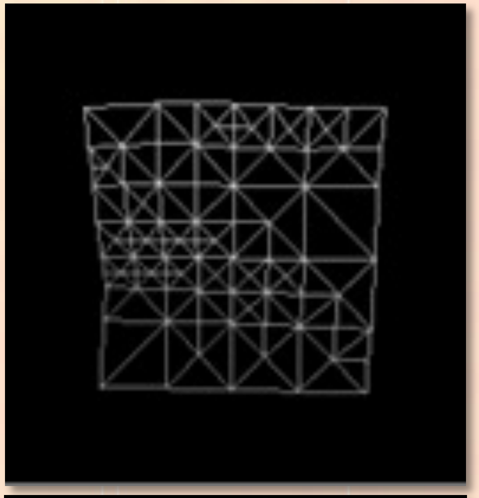
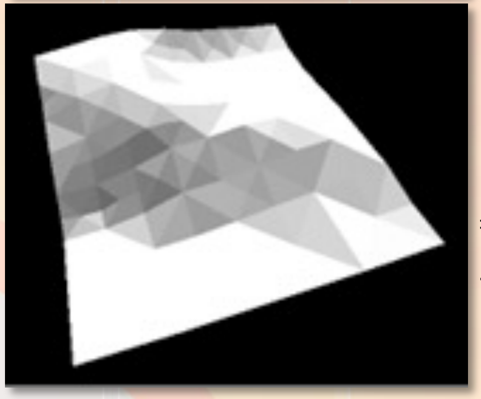
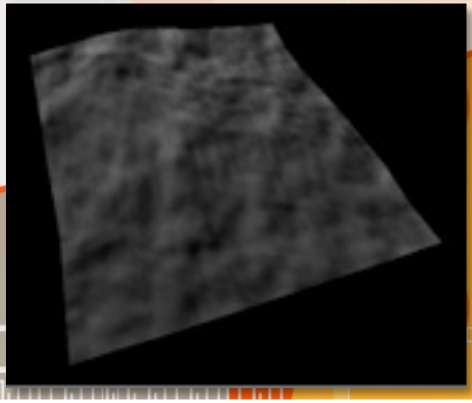
http://www.cognigraph.com/ROAM_homepage/

https://graphics.llnl.gov/ROAM/roam.pdf

http://lodbook.com/source/

ROAM (Real-Time Optimally Adapting Meshes)

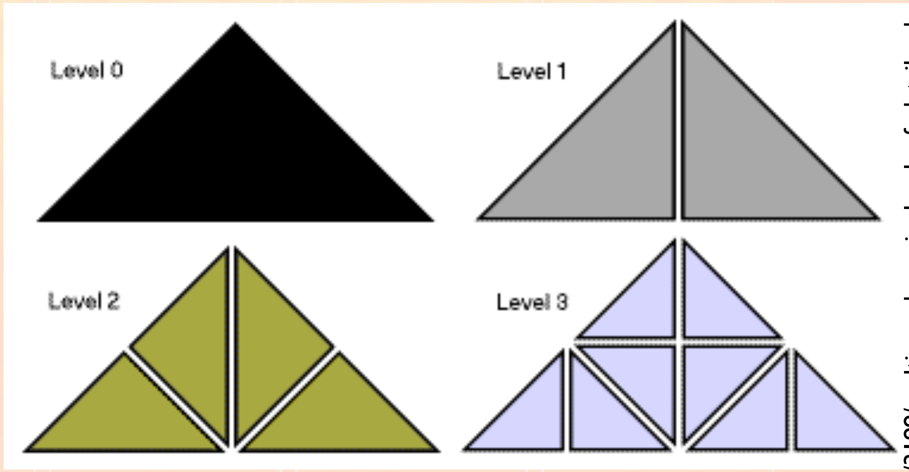
- Probably the most popular algorithm for terrain generation/modeling.
- Key concepts:
  - Height field (or height map) as for Landstrom et al.
  - Binary triangle tree structure (without cracks).
  - Split and merge operations (to refine and simplify the mesh)
- Other important concepts:
  - Two priority queues
    - One for splits and one for merge
    - Allows for frame-to-frame coherence
  - Error metrics for splits and merges
  - Geo-morphing – introduced, but rarely needed
  - Incremental triangle stripping introduced

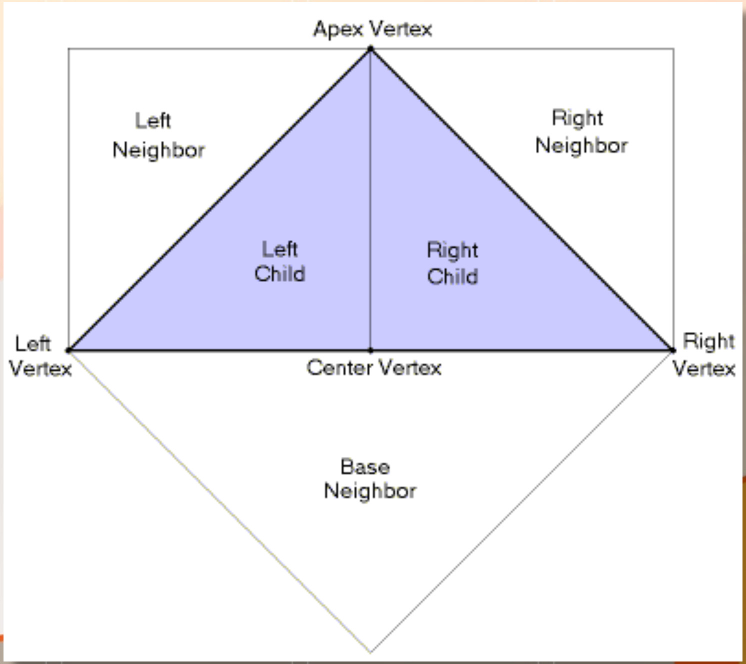
http://www.gamasutra.com/view/feature/3188/realtime_dynamic_level_of_detail_.ph

ROAM's LOD data structure

- Binary triangle tree (BTT)
  - Each patch is a simple isosceles right triangle. Splitting the triangle from its apex to the middle of its hypotenuse produces two new isosceles right triangles.
  - The splitting is recursive and can be repeated on the children until the desired level of detail is reached.



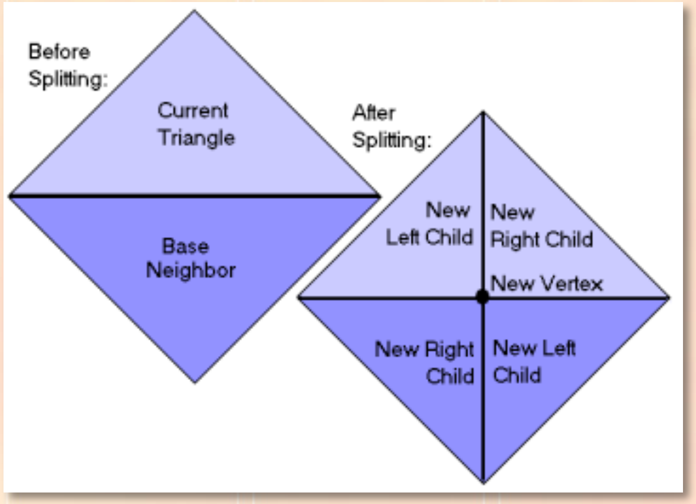
```
struct TriTreeNode {
    TriTreeNode *LeftChild;
    TriTreeNode *RightChild;
    TriTreeNode *BaseNeighbor;
    TriTreeNode *LeftNeighbor;
    TriTreeNode *RightNeighbor;
};
```

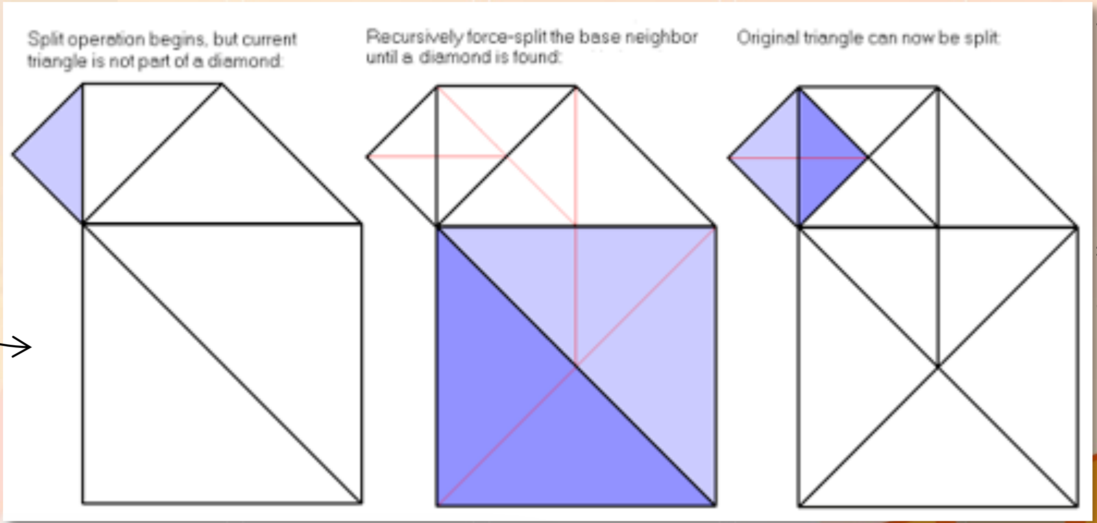


http://www.gamasutra.com/view/feature/3188/realtime_dynamic_level_of_detail_.php

ROAM triangle splitting & merging

- Three cases exist when attempting to split a node:
  - The node is part of a diamond
    - Action: split the node and its base neighbor.
  - The node is on the boundary edge of the mesh
    - Action: trivial, only split the node.
  - The node is not part of a diamond
    - Action: force split the base neighbor.




Split operation begins, but current triangle is not part of a diamond:
Recursively force-split the base neighbor until a diamond is found:
Original triangle can now be split:

http://www.gamasutra.com/view/feature/3188/realtime_dynamic_level_of_detail_.php

This technique does not cause cracks. How?

ROAM geomorphing: a technique to prevent popping effects

- Visual popping is a side effect of rendering with dynamic levels of detail when triangles are inserted or removed from the mesh.
- This distortion can be reduced to nearly unnoticeable amounts by vertex morphing, also called geomorphing, by gradually changing the rise or fall of a vertex's height from the un-split position to its new split position over the course of several frames.
- Geomorphing is given by the following interpolation formula:

$$z(t) = t \,.\, z_{\text{end}} + (1 - t) \,.\, z_{\text{start}}$$

with $t \in [0, 1]$

*Alain Fournier, Don Fussell, and Loren Carpenter (June 1982). "**Computer rendering of stochastic models**". Communications of the ACM 25 (6): 371–384.*

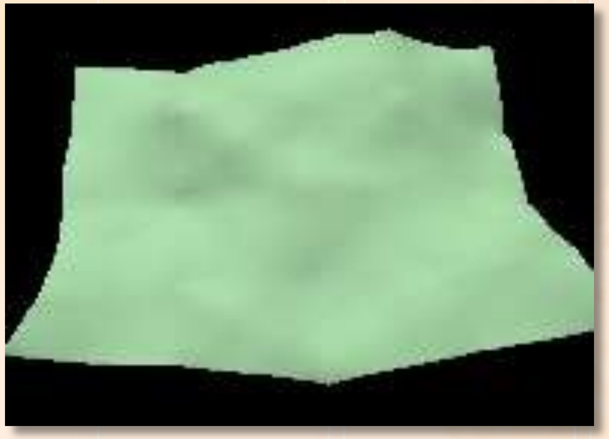
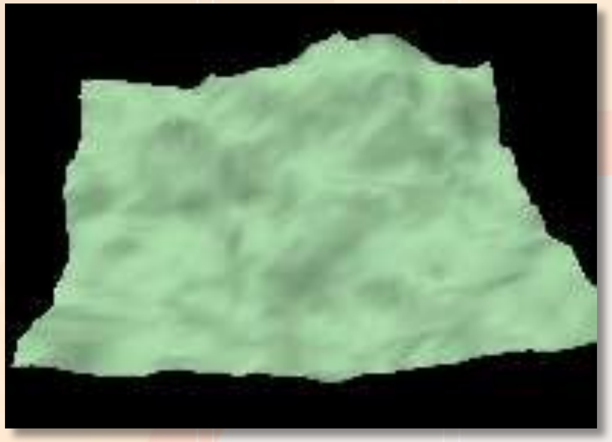
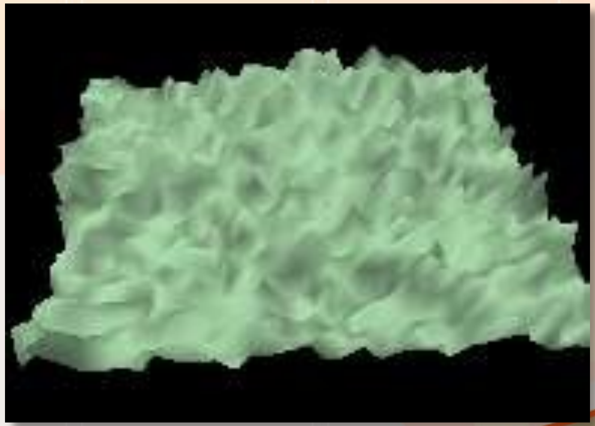
Diamond-Square Algorithm

http://en.wikipedia.org/wiki/Diamond-square_algorithm

http://paulboxley.com/blog/2011/03/terrain-generation-mark-one

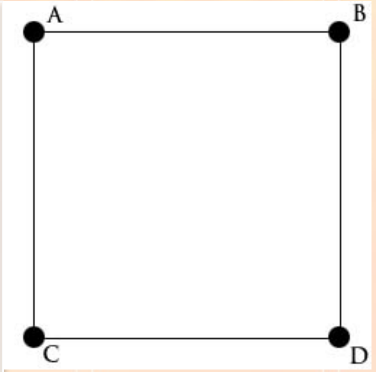
Diamond-square algorithm: context

- Also called the cloud fractal, plasma fractal or random midpoint displacement
- The 2D version of the original *midpoint displacement algorithm*
  - Therefore it also has a roughness constant
- The diamond-square algorithm works best if it is run on square grids of width 2^n
  - This ensures that the rectangle size will have an integer value at each iteration

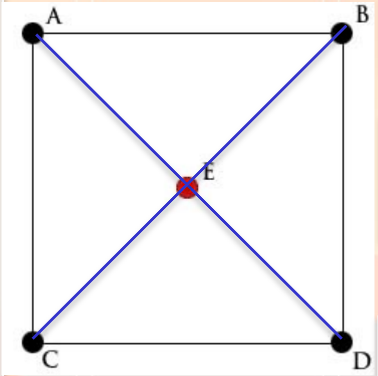




Diamond-square algorithm

- The algorithm starts with a 2 x 2 grid
  - The heights at the corners can be set to either zero, a random value or some predefined value
- ***Diamond step*** (observe the four diamond shape):
  - This step calculates the midpoint of the grid based on its corners and then adding the maximum displacement for the current iteration
- ***Square step***:
  - Calculate the midpoints of the edges between the corners
  - Since the first iteration is complete, now d is reduced by **d *= pow(2,-r)**, where r is the roughness constant.
- Repeat the previous two steps until the length of the square gets below a threshold.

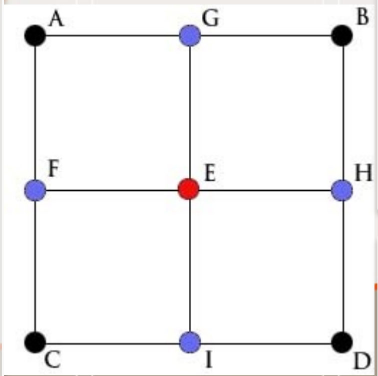


Only in height!



E = (A+B+C+D)/4 + Rand(d)

Rand(d) can generate random values between -d and +d



wrapping
G = (A+B+E+E)/4 + rand(d)
H = (B+D+E+E) /4 + rand(d)
I = (D+C+E+E)/4 + rand(d)
F = (A+C+E+E)/4 + rand(d)

Non-wrapping
G = (A+B+E)/3 +rand(d)
same for H,I,F

Summary

- Overview of terrain algorithms.
- Categories of terrain algorithms: discrete and continuous LODs.
- Issues in terrain generation and modeling.
- Classes of Continuous LODs: regular grids & heighfields, TINs, and voxel-based LODs.
- LEA algorithm.
- ROAM algorithm.
- Diamond-square algorithm.
- Final considerations.