Modeling Terrain Geometry

CPS344 Spring 2015

March 3, 5

A quick note

- Many of the links are straight from the class page for this lecture
- Looks like Professor Duvall is still deciding what to do with the due date, so ¯_(ツ)_/¯
- Stop me with questions

What is it?

Dictionary.com:

ter·rain

/təˈrān/

noun

1. a stretch of land, especially with regard to its physical features.
2. …

Motivation

- “critical scene component”
  - But we need the model first!

- Modeling existing terrain (e.g. GIS)
- Generating terrain algorithmically

Height map

- 2D array (image) whose values are heights

- A very typical model,
  has some limitations (e.g. can’t do tunnels)
- Pinscreen is a good analogue
- Easy to turn this into a mesh

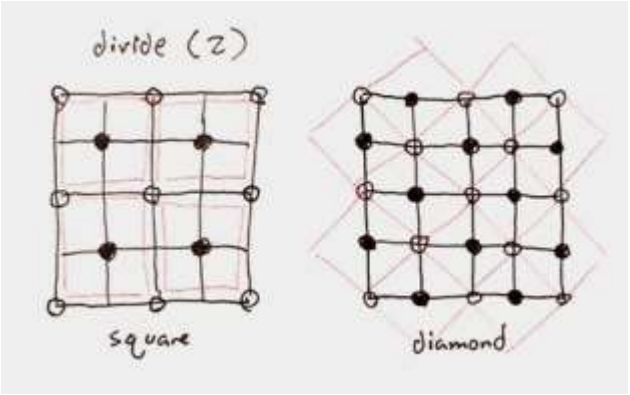
Flight simulator overview

- Starting skeleton is the surface modeling demo Professor Duvall showed last time

- Diamond-square algorithm (subdivisions)
- Perlin noise for offsets
- Normals for new faces and vertices
- Skybox (with collision detection)
- Level of detail
- Of course, some camera controls

Diamond-square algorithm

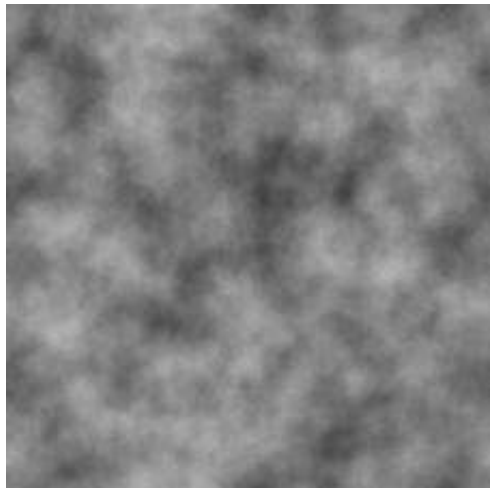
- Produces a fractal terrain
- General idea: [recursively] take a quad, move its midpoint up/down a little bit
  - The “diamond” step lets us hit every grid point
  - Notice: every quad we subdivide has had its corner heights computed
  - The amount you move it by is scaled by the side length, so *the initial amounts matter more*
  - Recurse on the four resulting quads



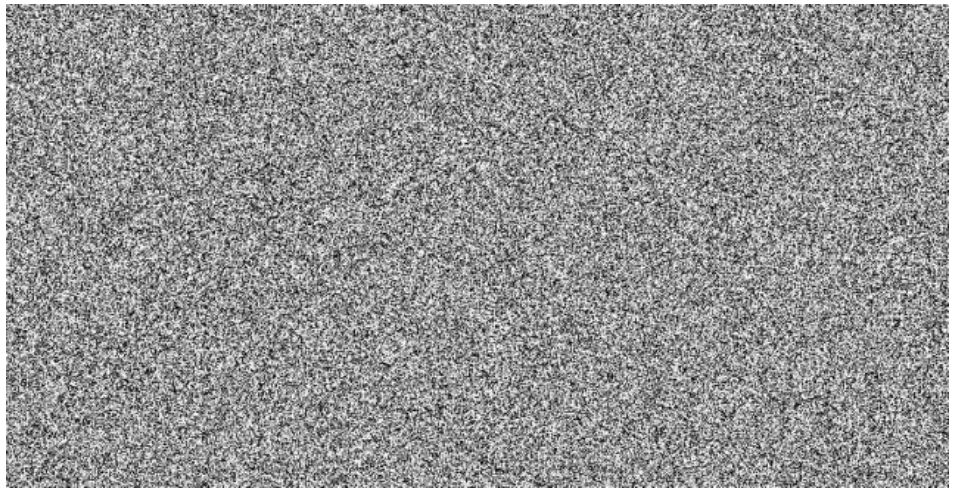
- http://en.wikipedia.org/wiki/Diamond-square_algorithm
- http://www.playfuljs.com/realistic-terrain-in-130-lines/
- https://code.google.com/p/fractalterraingeneration/wiki/Diamond_Square

Perlin noise

- A function from points to reals (in our use case, heights)
- http://gamedev.stackexchange.com/questions/68168/perlin-noise-help



Perlin noise



White noise

Perlin noise

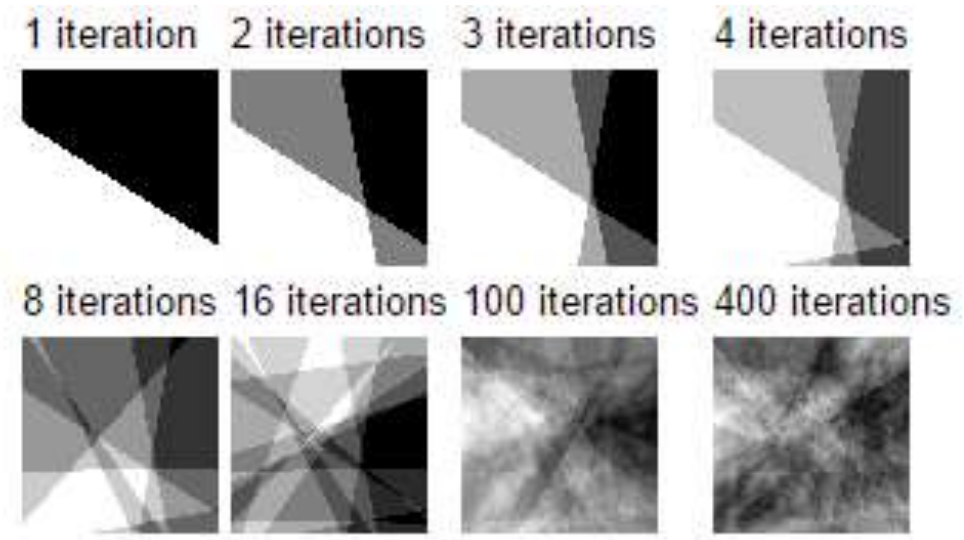
- Algorithm:
  - fix a grid
  - assign a (pseudo) random gradient vector for each grid point (2D direction)
  - On a query for point p:
    - Locate the grid cell containing p
    - Take the dot product between the distance vector between p and each cell corner
    - Return an interpolated value between the dot products
- For your project, these are the offsets you're choosing in the diamond-square algorithm
- Implemented for you: jogl_jars/src/framework/ImprovedNoise.java

Perlin noise

- Why go through all this trouble?
  - Smooth noise (the function is continuous, i.e. the noise is *coherent*)
  - With a fixed pseudorandom seed (choice of gradients), the output is reproducible
- Additionally:
  - “Details” in the noise are of a controlled *frequency* (by grid size)
  - Summing Perlin noise of different frequencies gives noise with a controlled *frequency range*
  - Functions of Perlin noise (e.g. 1/|noise|) can be used to simulate other random patterns, like wood grains, marble, fire, clouds
- http://webstaff.itn.liu.se/~stegu/TNM022-2005/perlinnoiselinks/perlin-noise-math-faq.html
- http://stackoverflow.com/questions/18511932/questions-regarding-perlin-noise-how-it-works

Aside: a few other terrain algorithms

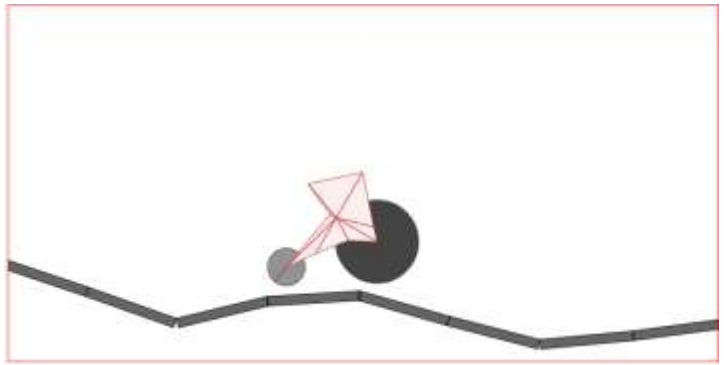
- Fault algorithm
  - Iteratively bisect the entire surface, move one side up and one side down
- http://www.lighthouse3d.com/opengl/terrain/index.php3?fault



Aside: a few other terrain algorithms

- Genetic algorithms for terrain generation
  - A designer (algorithm or person) selects the best to “reproduce”
  - Mutation, natural selection – models an evolutionary process
- http://www.hindawi.com/journals/ijcgt/2009/125714/

- Example of a genetic algorithm: http://rednuht.org/genetic_cars_2/



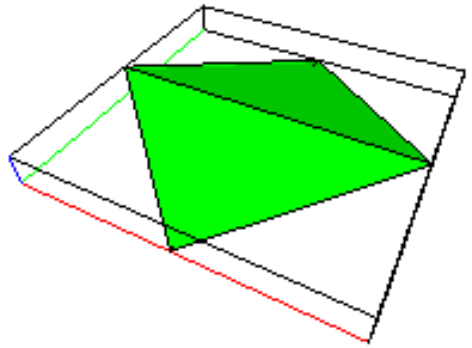
Review: meshes (and mesh data structures)

- Vertices, edges, faces, and so on
- http://www.cs.duke.edu/courses/compsci344/spring15/classwork/10_terrain/mesh.pdf
- http://www.flipcode.com/archives/The_Half-Edge_Data_Structure.shtml

- For your project: simple polygons (quads), 3D surface mesh
- Your data structure only needs to support:
  - Finding adjacent faces (for vertex normals)
  - Adding new faces (i.e. for the refinement)

Review: approximating normals

- The faces coming out of your subdivision scheme might not be planar



"Sliver Tetrahedra." From the Los Alamos Grid Toolbox documentation. https://lagrit.lanl.gov/docs/QUALITY_sliver_cap_needle_wedge.html

- Note: if you triangulate, of course they're planar!
  - Planes are uniquely defined by three points (i.e. a triangle)
  - When faces are planar, we can just use cross products between the edges

Review: approximating normals

- Newell's method – approximating normal for polygons
  - Where the polygons need not be planar
- https://courses.cit.cornell.edu/cs417-land/SECTIONS/normals.html

$$n_x = \sum_{\substack{i=1 \\ i<m \\ j=i+1}}^{\substack{m \\ i=m \\ j=1}} (y_i - y_j) \cdot (z_i + z_j)$$

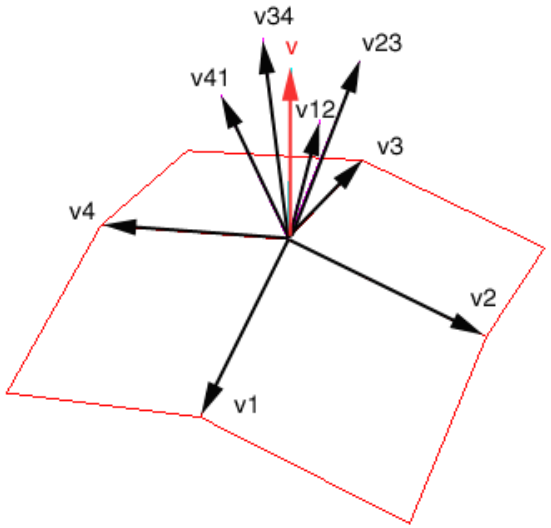
$$n_y = \sum_{\substack{i=1 \\ i<m \\ j=i+1}}^{\substack{m \\ i=m \\ j=1}} (z_i - z_j) \cdot (x_i + x_j)$$

$$n_z = \sum_{\substack{i=1 \\ i<m \\ j=i+1}}^{\substack{m \\ i=m \\ j=1}} (x_i - x_j) \cdot (y_i + y_j)$$

- This is what we ask that you use for the project

Review: approximating normals

- To approximate vertex normals, most schemes involve “averaging” normals of the adjoining faces
  - Sum over them and normalize the result
- OpenGL needs vertex normals to do smooth shading



- http://www.lighthouse3d.com/opengl/terrain/index.php3?normals

Before we start

- Project deadline is indeed extended:
  - Due Sunday, March 22 (one week after end of spring break)
  - There's some sort of progress check tomorrow on Friday
    - "significant update... one basic feature"
  - Also...

- Questions? Comments?

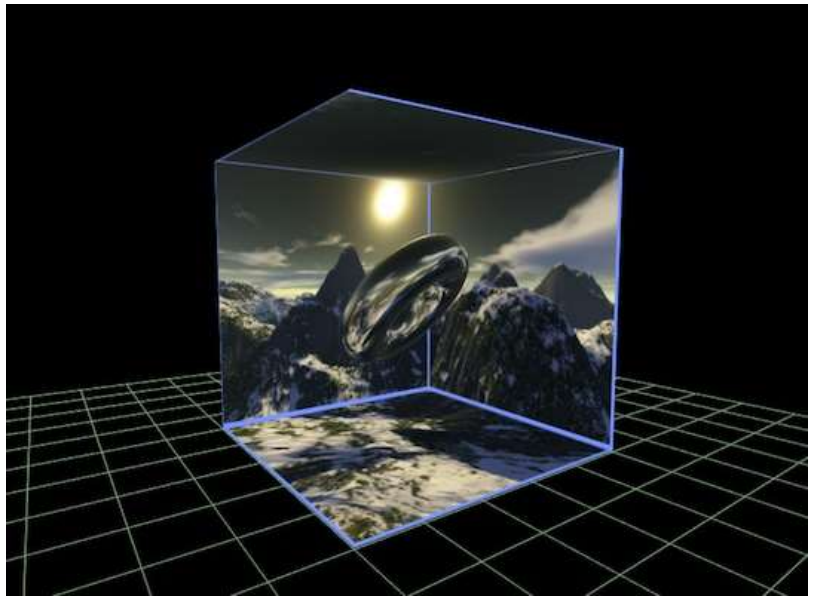
Plan

- Today we'll try to cover:
  - Skyboxes
  - Collisions (axis-aligned bounding boxes)
  - Level of Detail
  - Other resources

Skyboxes

- Idea: wrap the scene in backdrops (a texture) of the sky/whatever to create the impression of distant surroundings
  - Simulates parallax
  - Boxes (cubes) have very easy texture mappings, so often used rather than spheres/domes

From: The Truman Show (1998)



From: http://away3d.com/tutorials/Using_A_Skybox

Skyboxes

- OpenGL actually provides very nice facilities for you to do this!
- Cubemaps are a type of texture
  - https://www.opengl.org/wiki/Cubemap_Texture
  - You'll have to do the typical work of loading/binding the texture(s)
  - Of course, make an enclosing cube (geometry) to texture
  - Make sure you bind to GL_TEXTURE_CUBE_MAP rather than GL_TEXTURE_2D
  - There are also special GL variables targeting the 6 specific cube faces, look for these in the references
  - There are more details for making it better (seamless edges, depth testing, etc.) but the basic skybox should be pretty straightforward
- http://antongerdelan.net/opengl/cubemaps.html
- http://www.learnopengl.com/#!Advanced-OpenGL/Cubemaps

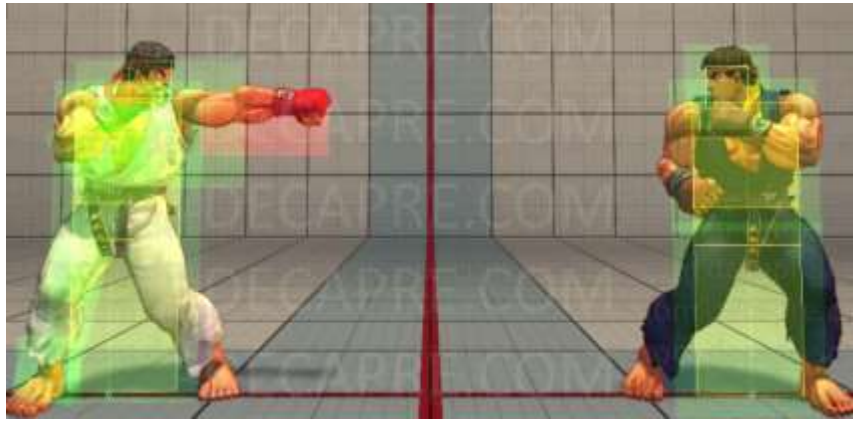
Aside: environment maps

- Used to simulate (fake) reflection and refraction
- Instead of bouncing a ray everywhere, just get color from the right place in the cubemap
  - Like a 1-step version of ray tracing reflection/refraction which only hits the background
  - Still have to do a calculation to find the reflected/refracted ray

Both from the learnopengl.com tutorial on the previous slide

Collisions

- Rather than doing a (difficult) intersection test between every piece of moving geometry, check intersections of simple shapes which surround them
- Intuition: if two objects intersect, their containers have to intersect first!



Screencap from: http://decapre.com/watch?v=MW8WwWNO0nE

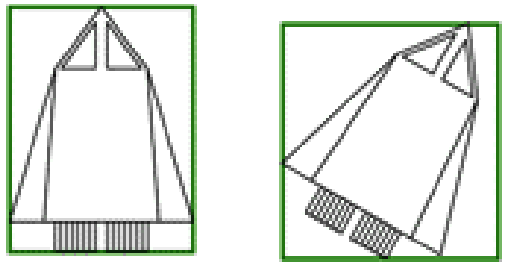
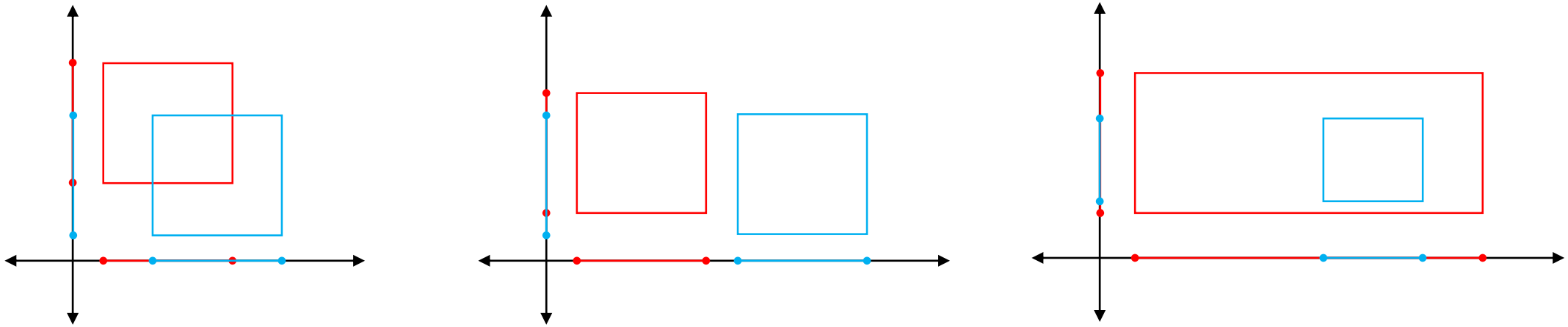


fig. 1: AABB

From: http://www.gameprogrammer.net/delphi3dArchive/collisiondetection.htm

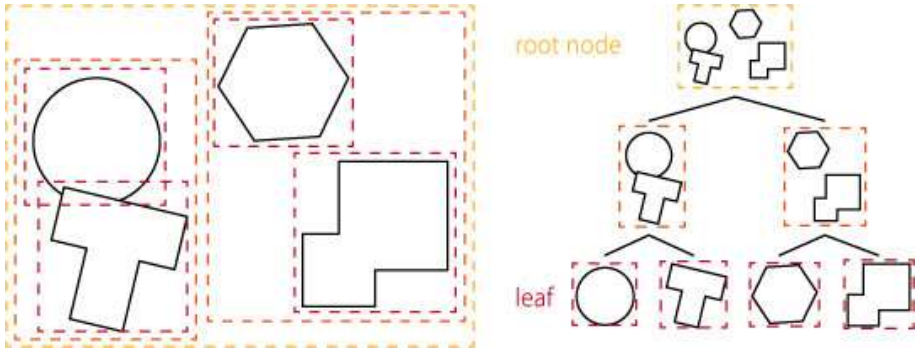
Collisions with axis-aligned bounding boxes

- We'll be using axis-aligned bounding boxes (AABB)
- Breaking that down:
  - Bounding box: a box completely containing an object (often minimal)
  - Axis-aligned: the edges of the box are parallel with the coordinate axes
- To check for collisions, we'll check if these boxes intersect instead
- Given the eight corners of each (3D) box, what's the intersection test?
  - Hint: look at the projection on each axis. What does it look like when they intersect? When they don't?



Collisions with axis-aligned bounding boxes

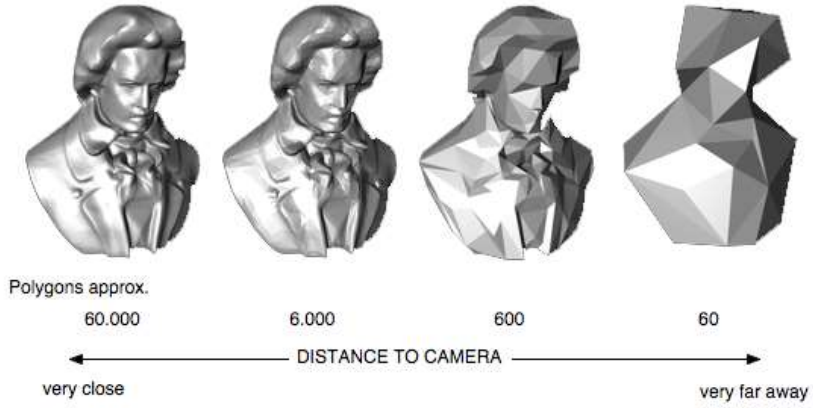
- Unanswered here:
  - how do you find the smallest (most accurate) bounding box?
  - What if the object is moving so fast it completely avoids the bounding box in one frame? (teleports to the other side)
- Even when the boxes intersect, it's not necessary that the objects actually intersect – then either:
  - We don't care, that's good enough.
  - We then decide do the intersection test for the actual geometry.
- As the previous point hinted, we can organize AABB in a hierarchical fashion over an entire scene to efficiently find the right objects to check for intersections (ray tracing gets a lot faster with this)
  - http://en.wikipedia.org/wiki/Bounding_volume_hierarchy



From: http://slis.tsukuba.ac.jp/~fujisawa.makoto.fu/lecture/iml/text/3_collision.html

Level of Detail

- If things are far away, the viewer can't make out fine details
  - Avoid rendering fine geometric details for distant objects
  - Conversely, render more detail for closer objects
  - "fidelity/performance tradeoff"
  - Other versions exist (boundaries, focal point)
- http://www.cs.duke.edu/courses/compsci344/spring15/classwork/10_terrain/LOD.pdf



From: http://www.opensg.org/projects/opensg/wiki/Tutorial/OpenSG2/NodeCores

Level of Detail

- For the project, definitely not as general
- Input is a regular grid, simply select how much to refine each input cell
  - A simple form: only refine input quads that are within D units away
  - Better: refinement (recursion) depth decreases linearly as we get farther away
  - It's fine if you decide this on a per-input cell basis, but feel free to try continuous
- What's distance in our case?
  - Since the grid is regular in x-y, just use the distance in the x-y plane

- We can still compute the complete refinement in the beginning, and only pass to OpenGL the vertices corresponding to the right granularity
  - The lower level of detail (i.e. less refined subgrid) is implicit in the completely refined mesh! Just have to select the right vertices to submit to OpenGL.

Aside: some continuous LOD algorithms

- ROAM: http://www.cognigraph.com/ROAM_homepage/
  - Queues monitoring triangles to split or merge
- SOAR: http://computation.llnl.gov/casc/SOAR/
  - View frustum culling, and some other techniques shared with ROAM

- Both of these seem to be focused on using data representation (layout in memory, indexing, etc.) in order to get their speed

More resources

- Virtual Terrain Project: http://vterrain.org/
  - On terrain generation algorithms: http://vterrain.org/Elevation/Artificial/
  - On level of detail: http://vterrain.org/LOD/Papers/

- NeHe (many, many OpenGL tutorials): http://nehe.gamedev.net/