CSc 165
Computer Game Architecture

10 - Terrain

Overview

- Basic Issues
- Height Maps
  - Concepts
  - Generation Algorithms
  - Image-based Methods
- Terrain Size
  - Culling, Paging, & Level of Detail (LOD)
- TAGE Terrain Support

Outdoor Terrains

Terrain: *a "world object" defining the ground*

- not a texture
- not a skybox

issues:

- outdoors - ground is very seldom "flat"
- sky box ground won't work -- moves with player
- two scenarios: "walkover" vs. "flyover"

  both have issues to deal with
- viewer proximity to detail

Terrain generation

How do we create the desired terrain ?

Terrain rendering

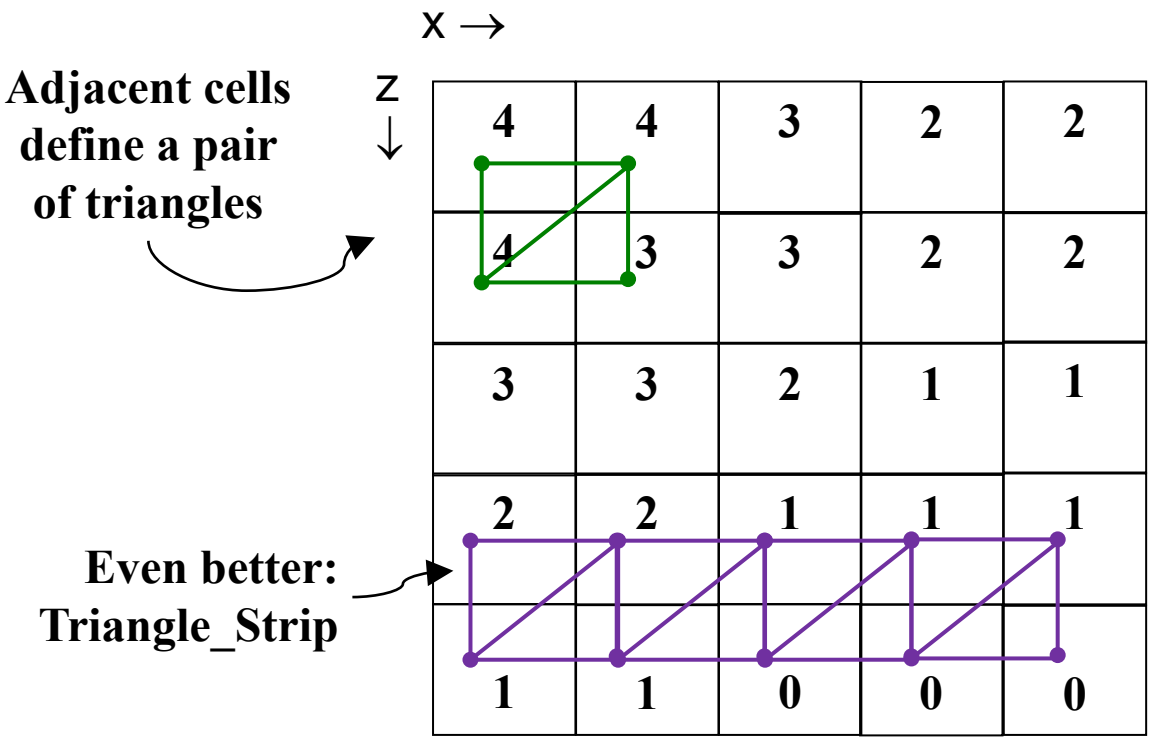
There can be millions of polygons

Terrain appearance

How do we assign color/lighting to the terrain ?

Height Maps

A 2D grid of numbers representing heights



Drawing a triangle:

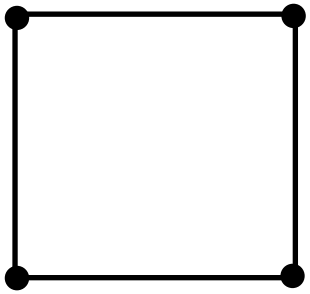
```
Vx = col
Vz = row
Vy = height[row,col]
vertex = (Vx, Vy, Vz)
```

Height Map Generation

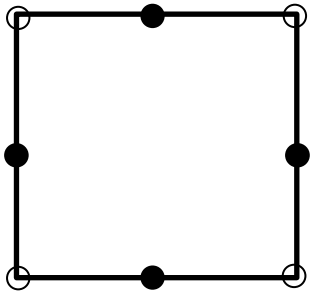
- Algorithmic
  - Functional description, e.g. $h(x,z) = 10 * \sin\left(\frac{x}{24}\right) + 7\cos\left(\frac{z-50}{18}\right)$
  - Midpoint displacement
  - Diamond-Square
  - Fault line
  - Hill raising
- Image-based

Height Map Generation Algorithms

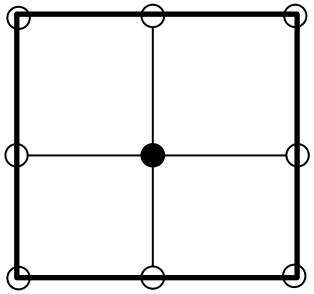
"Midpoint Displacement"



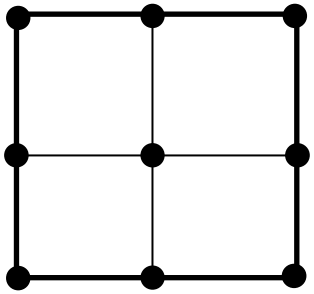
Assign random height values to four corners



Assign height values to edge midpoints by averaging corners and adding small displacement

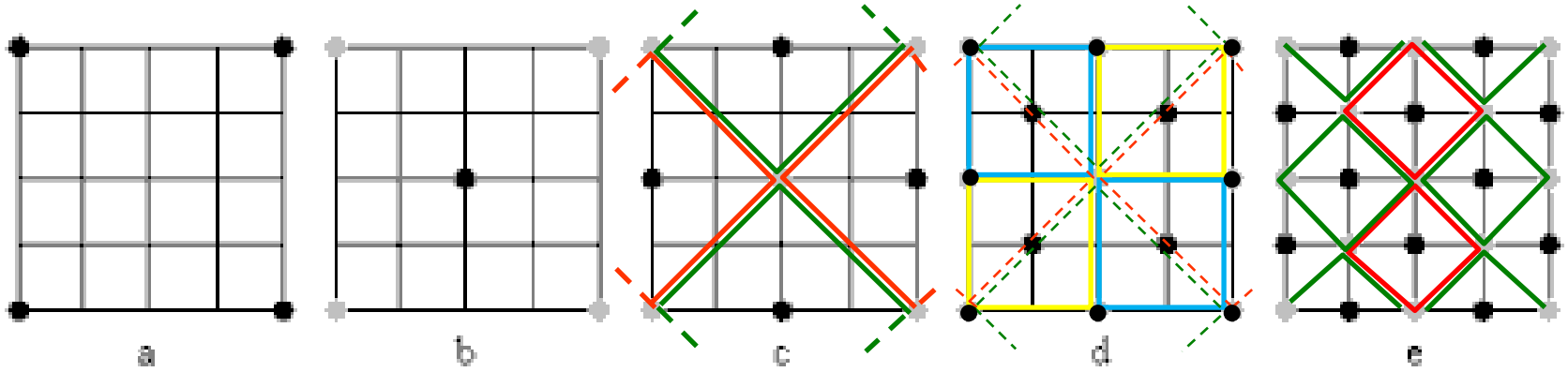


Assign height value to center point by averaging edge midpoints and adding small displacement



Repeat recursively for each smaller square

Diamond-Square



Choose outermost square

Square center = avg of corners + random offset

Form diamonds from square corners and middle

Diamond middles = average of corners + random offset

Form squares of half-size

Square centers = avg of corners + *smaller* offset

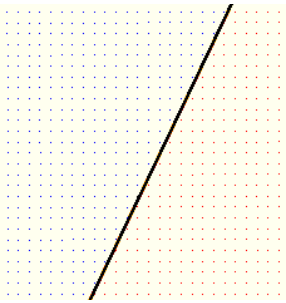
Form diamonds

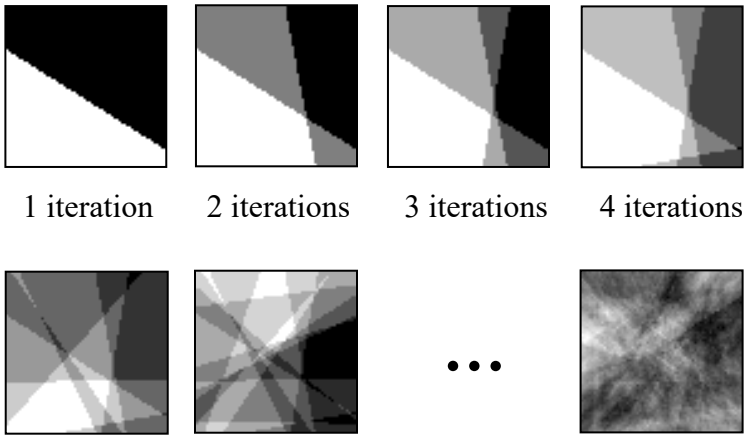
Diamond middles = average of corners + offset

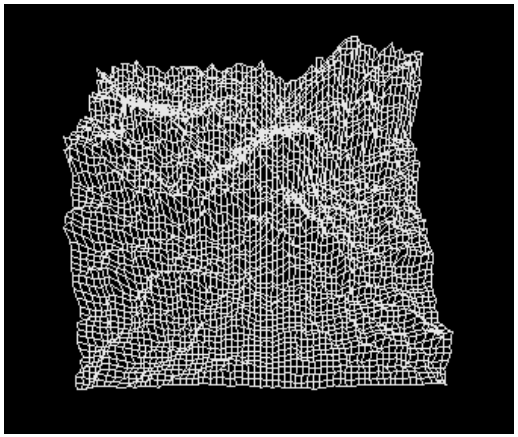
Image credit: Generating Random Fractal Terrain, Paul Martz, http://www.gameprogrammer.com/fractal.html

Fault Line

- Generate random fault line
- Increase heights on one side, decrease on the other
- Repeat





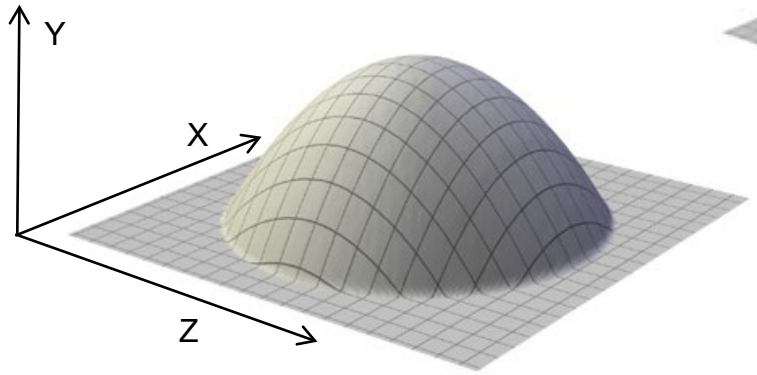


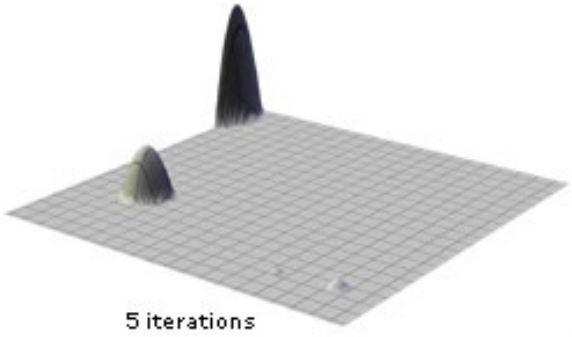
Resulting HeightMap

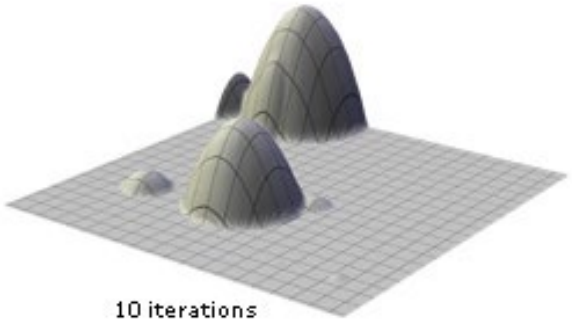
Image credit:
www.lighthouse3d.com

Hill-Raising

- Choose a random point and radius
- Raise a hill with the chosen radius at the point
- Repeat







A single hill at (x1,z1) with radius r :

$$y(x_2, z_2) = r^2 - \left((x_2 - x_1)^2 + (z_2 - z_1)^2\right)$$

Image credit http://www.stuffwithstuff.com/robot-frog/3d/hills/hill.html

Image-based Height Fields

Basic Idea: use image pixels as “height”

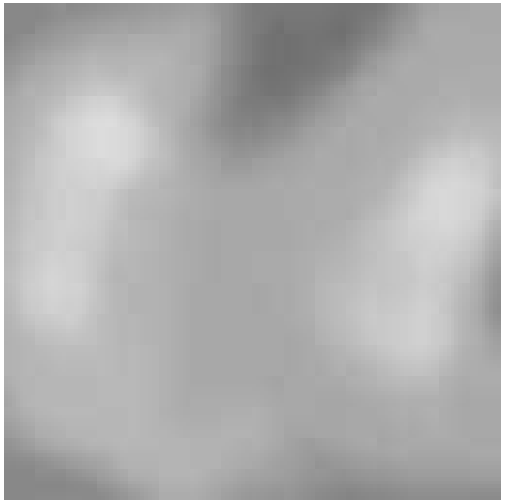
Most common form: *“gray-scale”*

0 = black = low height

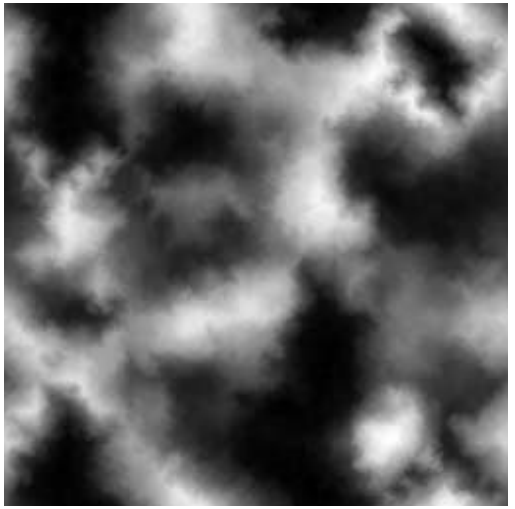
1 = 255 = white = high height

Easy to create in any paint program

Terrain Map Examples



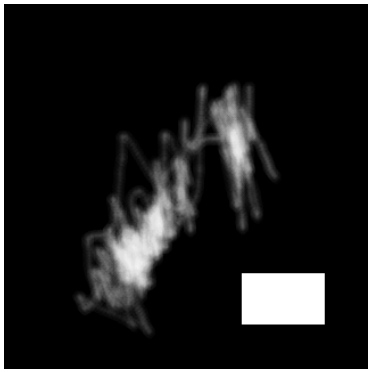
Low variation in height



Wide variation in height

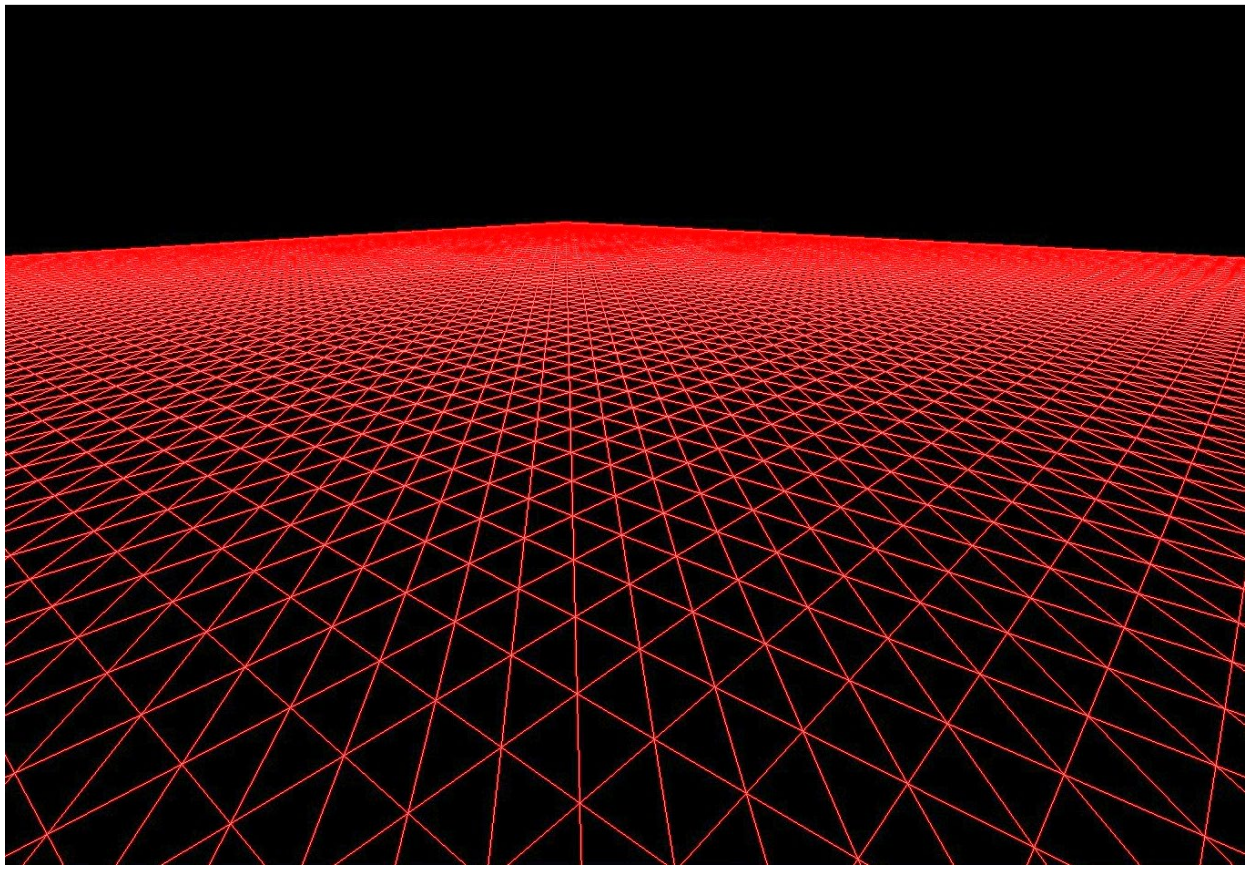
Hardware Support

height map:



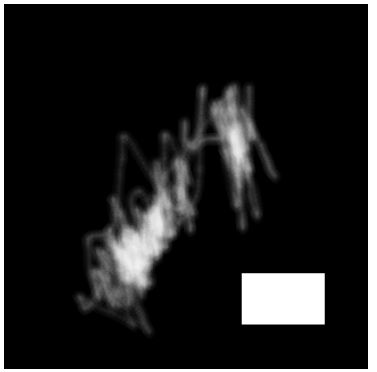
Vertex shader example (TAGE):

Starts with a large grid of vertices: (100x100)



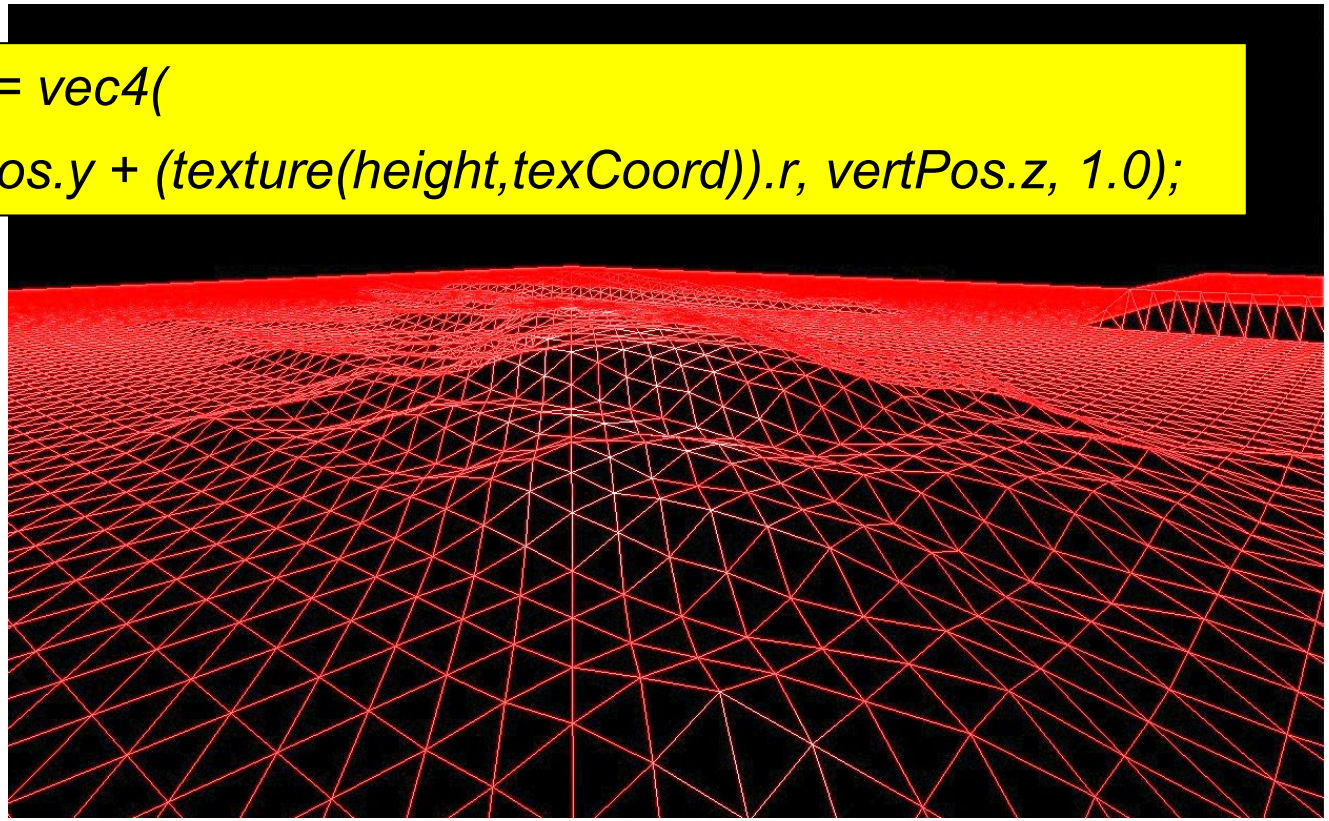
Hardware Support

height map:



Vertex shader raises vertex locations based on height map:

```
vec4 newVertPos = vec4(
    vertPos.x,  vertPos.y + (texture(height,texCoord)).r, vertPos.z, 1.0);
```



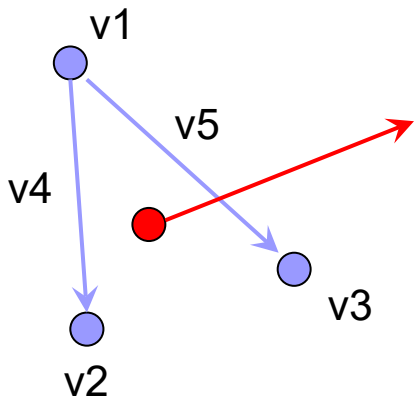
(shown here in wireframe mode)

Estimating Normal Vectors

After moving vertices, new normal vectors need to be created so that lighting will work properly.

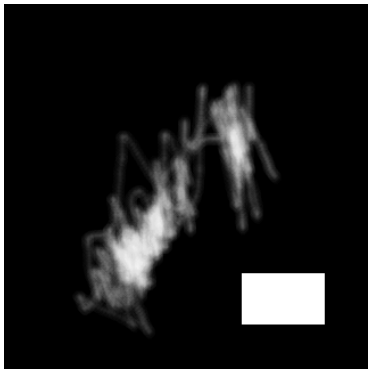
In TAGE, this is done in the Fragment Shader:

```
vec3 estimateNormal(float offset, float heightScale)
{  float h1 = heightScale * texture(height, vec2(tc.s, tc.t+.01)).r;
   float h2 = heightScale * texture(height, vec2(tc.s-.01, tc.t-.01)).r;
   float h3 = heightScale * texture(height, vec2(tc.s+.01, tc.t-.01)).r;
   vec3 v1 = vec3(0, h1, -1);
   vec3 v2 = vec3(-1, h2, 1);
   vec3 v3 = vec3(1, h3, 1);
   vec3 v4 = v2-v1;
   vec3 v5 = v3-v1;
   vec3 normEst = normalize(cross(v4,v5));
   return normEst;
}
```

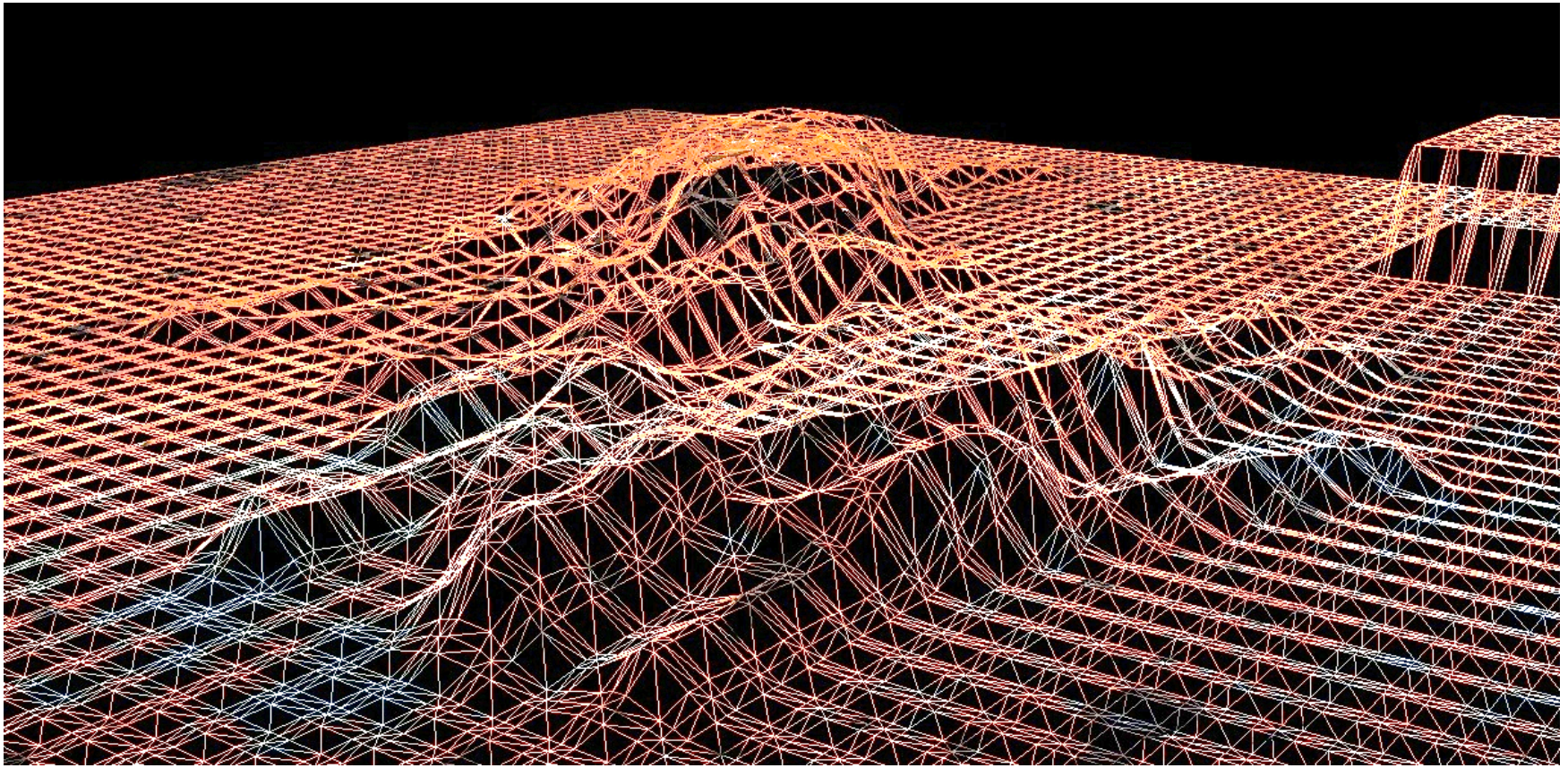


Hardware Support

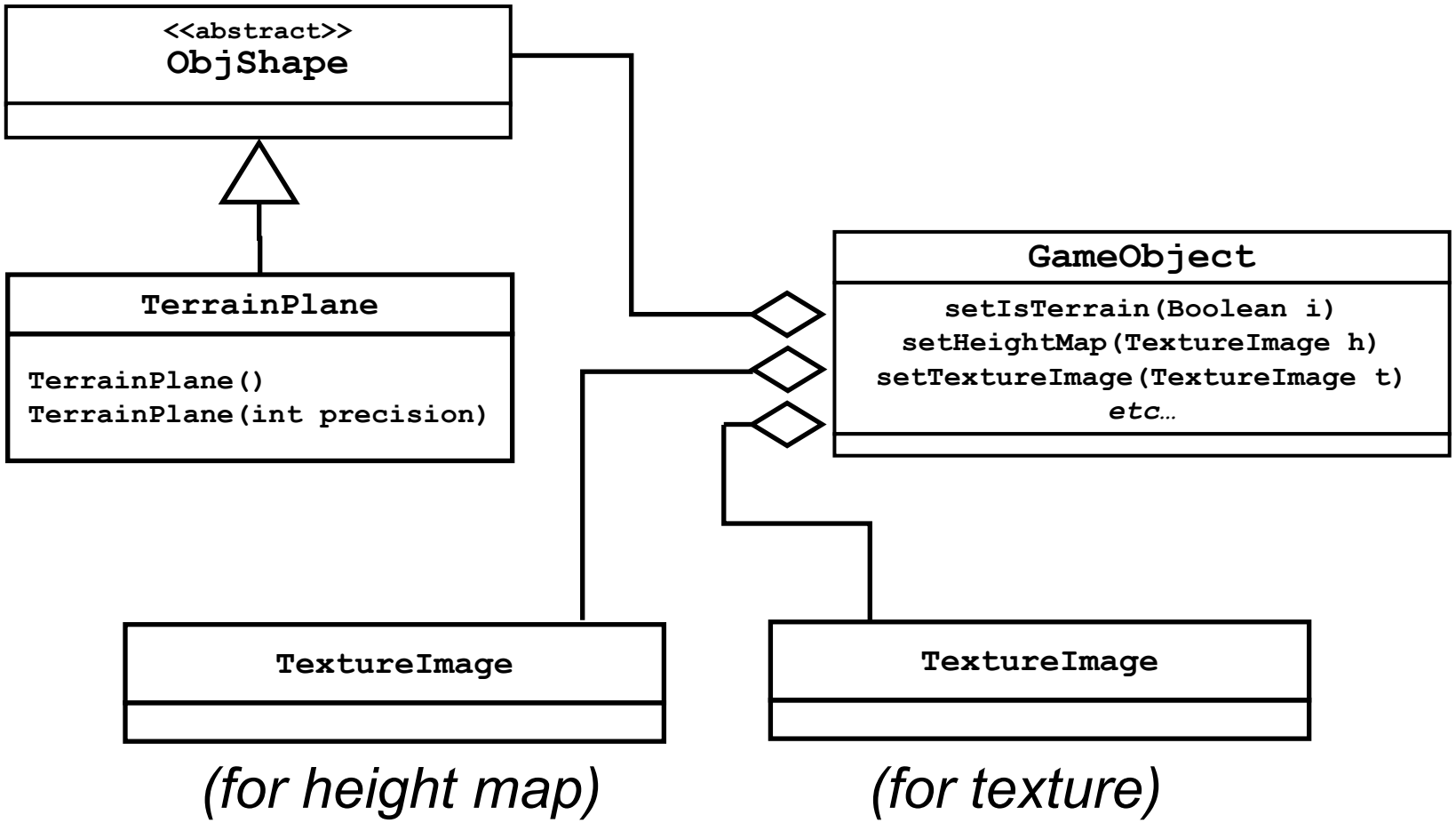
height map:



Tessellation shader example (RAGE):



TAGE Terrain Classes



using the TAGE terrain classes

- Create height map
  - typically a grayscale jpg image
- Instantiate a TerrainPlane
  - use default precision, or specify a precision
- Instantiate a GameObject
  - specify the terrainPlane as the shape (in constructor)
  - setIsTerrain(true)
  - setHeightMap(*the heightmap image*)
  - set desired terrain as for any game object
  - scale on Y axis adjusts actual heights

Following Terrain Height

- Get avatar world X and Z coordinates
- Retrieve height from terrain GameObject
- Adjust avatar y-translation appropriately

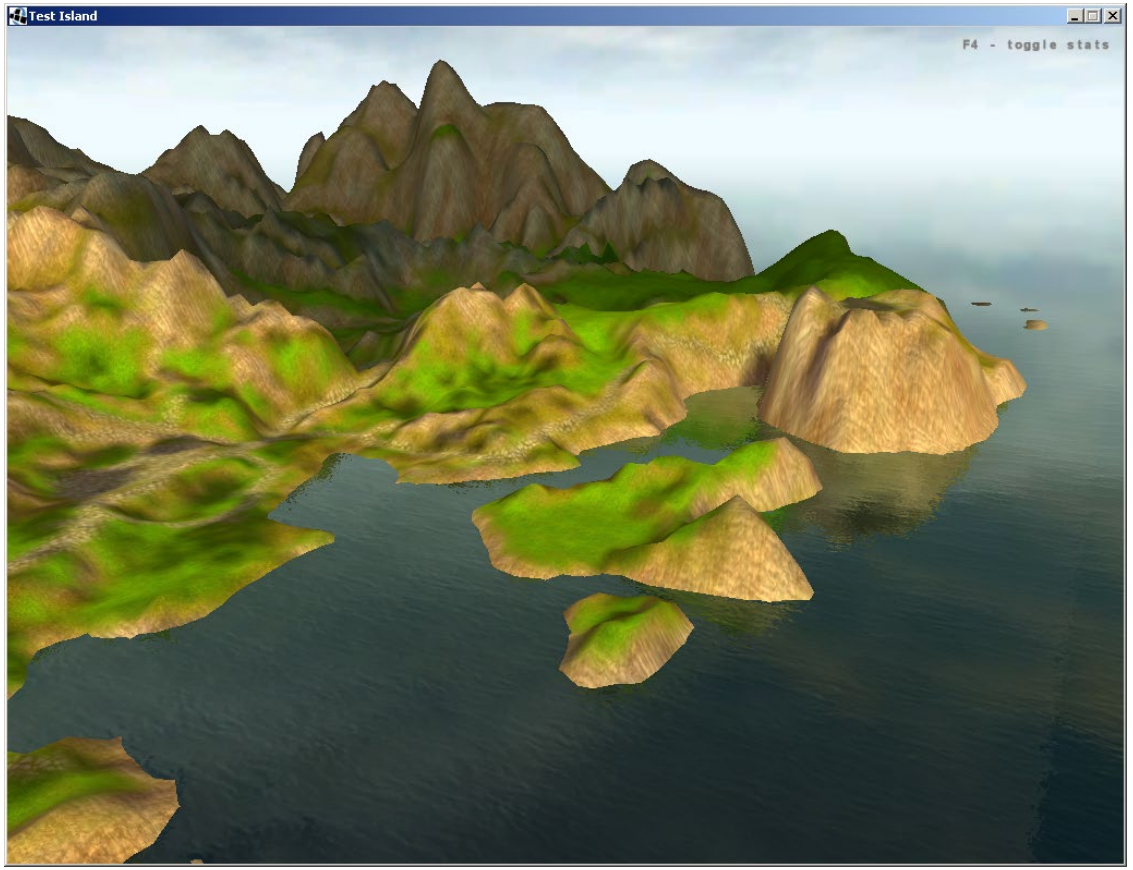
```
// update altitude of dolphin based on height map
Vector3f loc = avatar.getWorldLocation();
float height = terrObject.getHeight(loc.x(), loc.z());
avatar.setLocalLocation(new Vector3f(loc.x(), height, loc.z()));
```

optional:

adjust avatar tilt based on neighboring heights

Lakes and Islands

Can be simulated by adding an additional Plane shape, textured with water color:



other Terrain issues

Terrain map may not cover the world

remedies:

- repeated terrain
- scaling

Terrain is huge, renderer can't keep up

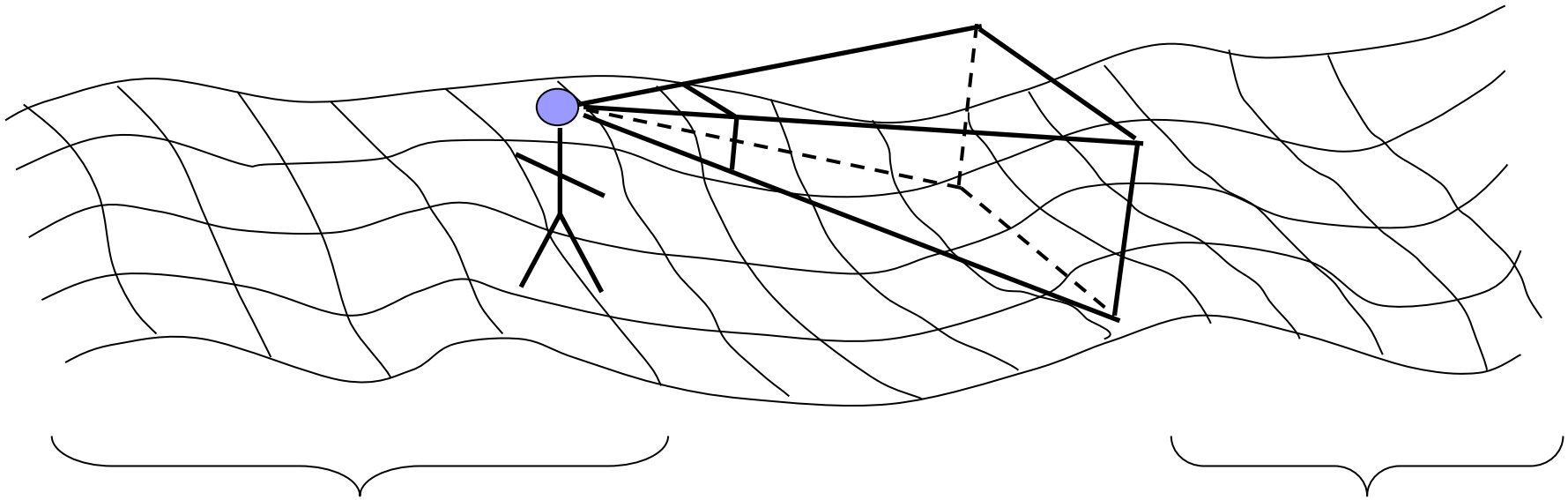
remedies:

- Frustum Culling
- Terrain Blocks and Paging
- Level of Detail (LOD – see CSc-155)
- Hardware support (e.g., tessellation)

Frustum Culling for Terrain

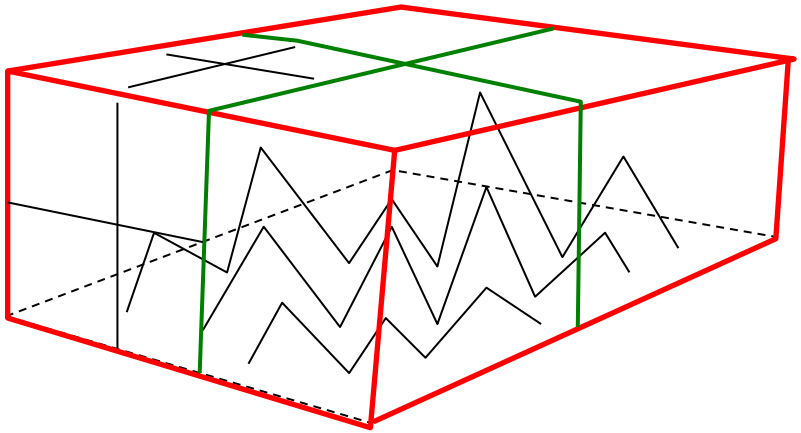
Most triangles are outside frustum

Can we avoid sending them down the pipeline?



QuadTrees For Visibility Culling

Bounding Box



HeightField