Finishing Up Assignment 3: Raytracing II

COS 426: Computer Graphics (Spring 2022)

Henry Wang, Edward Yang

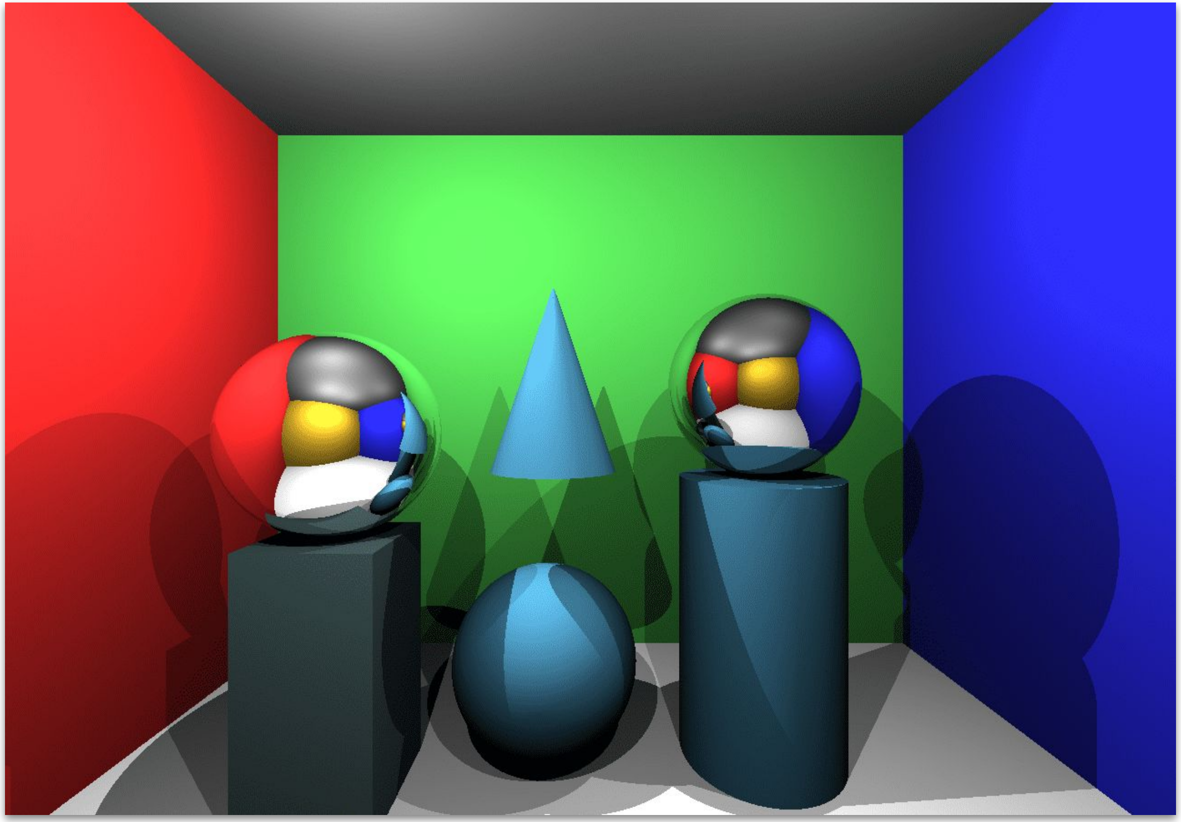
Agenda

- Lighting
  - Hard Shadows
  - Soft Shadows
  - Transmission
- Materials
  - Checkerboard
  - Phong
  - Natural Textures
    - Perlin Noise
    - Procedural Generation
- Advanced Path Tracing

Hard Shadows

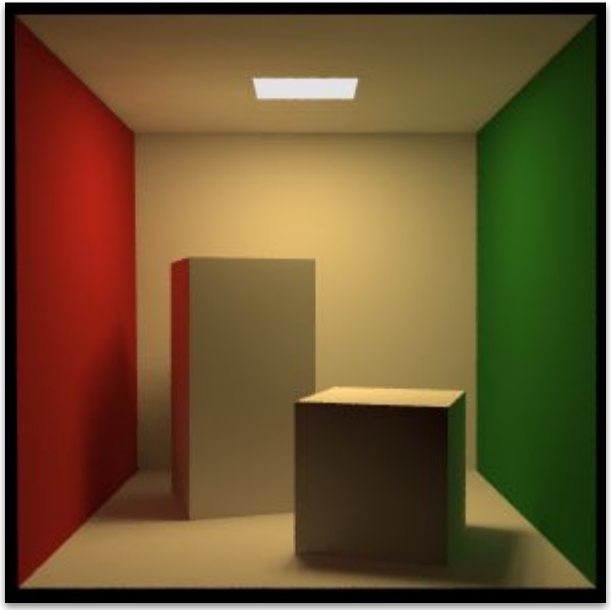
- Points should not sample the contribution from a light that they cannot “see”
- First cast a ray from the intersection point to the light in question (use `rayIntersectScene`) and determine the length of this **shadow ray**
- If this distance is “more than a little less” than the distance between the intersection point and the light, the point is in shadow.

Hard Shadows



Soft Shadows

- Lights in real life aren’t “points”
- Certain spots can now see fractional portions of a light
- This creates shaded areas (penumbra) around shadows (umbra)
    - Penumbra = *paene* + *umbra* (almost + shadow)
    - Just like “peninsula”

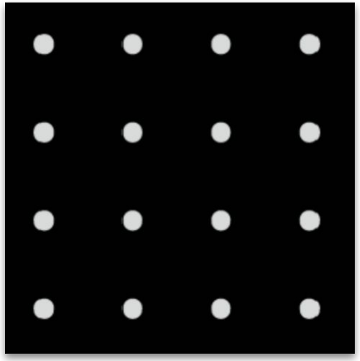


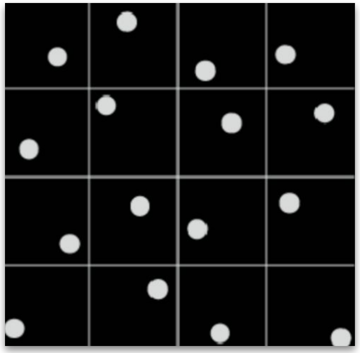
Soft Shadows

- For soft shadows in A3, pretend each light is actually a spherical orb of some radius (you can tune this parameter)
- Cast multiple shadow rays per light from points sampled on the light's surface. Return the estimated fractional contribution of the light (proportion of shadow rays that made it to the light)

Soft Shadows: Sampling

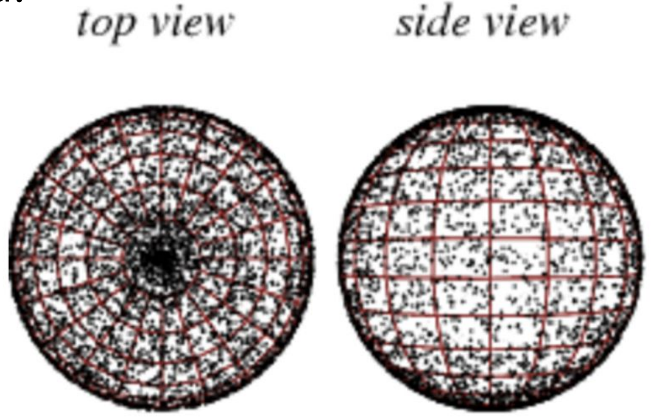
- In raytracing, uniform sampling a grid is almost always bad
  - It creates grid-like visual artifacts
  - This applies to more than soft shadows
- Instead, randomly sample the grid by applying **jitter** (a random offset) to each point
  - Better than "just random points" because more even distribution
  - Therefore, faster convergence to correct (non-noisy) value!





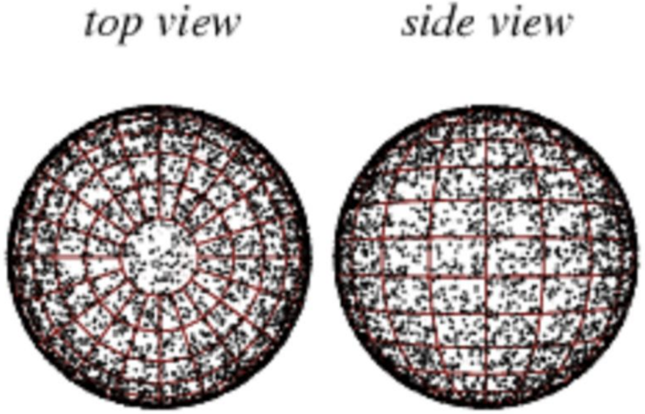
Soft Shadows: Sampling

- For A3, you'll need to sample points on a spherical shell
- Be careful about how you parameterize the surface
  - Let's try spherical coordinates and sampling θ [0, 2π) and ϕ [0, π)
  - This is not good!



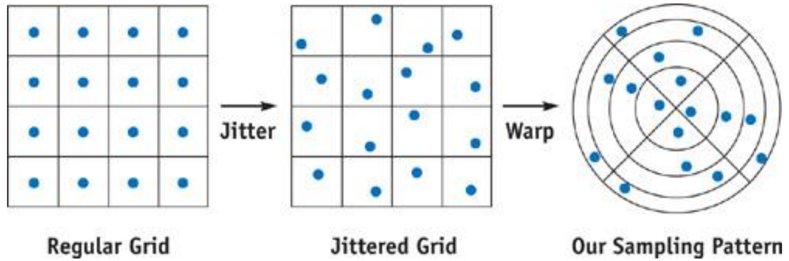
Soft Shadows: Sampling

- For A3, you'll need to sample points on a spherical shell
- Be careful about how you parameterize the surface
  - Let's try sampling by **solid angle**
  - Yay! It's even!



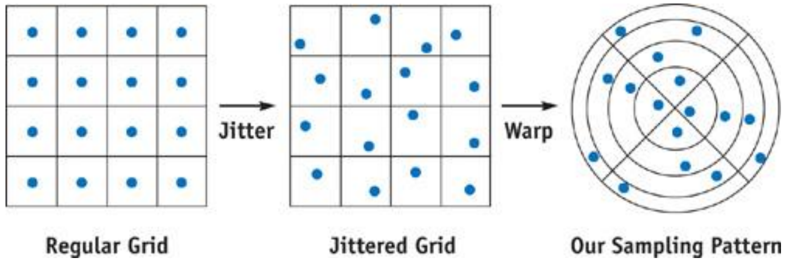
Soft Shadows: Sampling

- First split the unit area [0, 1) x [0, 1) into an N x N grid, where N is the square root of the number of samples you want to make
- For the (minX, minY) of each tile (uniform sampling offset into corner)
- Apply jitter by offsetting the corner position by 1/(N) in the *x* and *y* directions, independently (stratified random sampling)
    - 1/N is the width and height of each grid tile, so this random position falls anywhere within the tile with uniform probability
- Map each random sample in [0, 1] x [0, 1] to the space you want to sample.

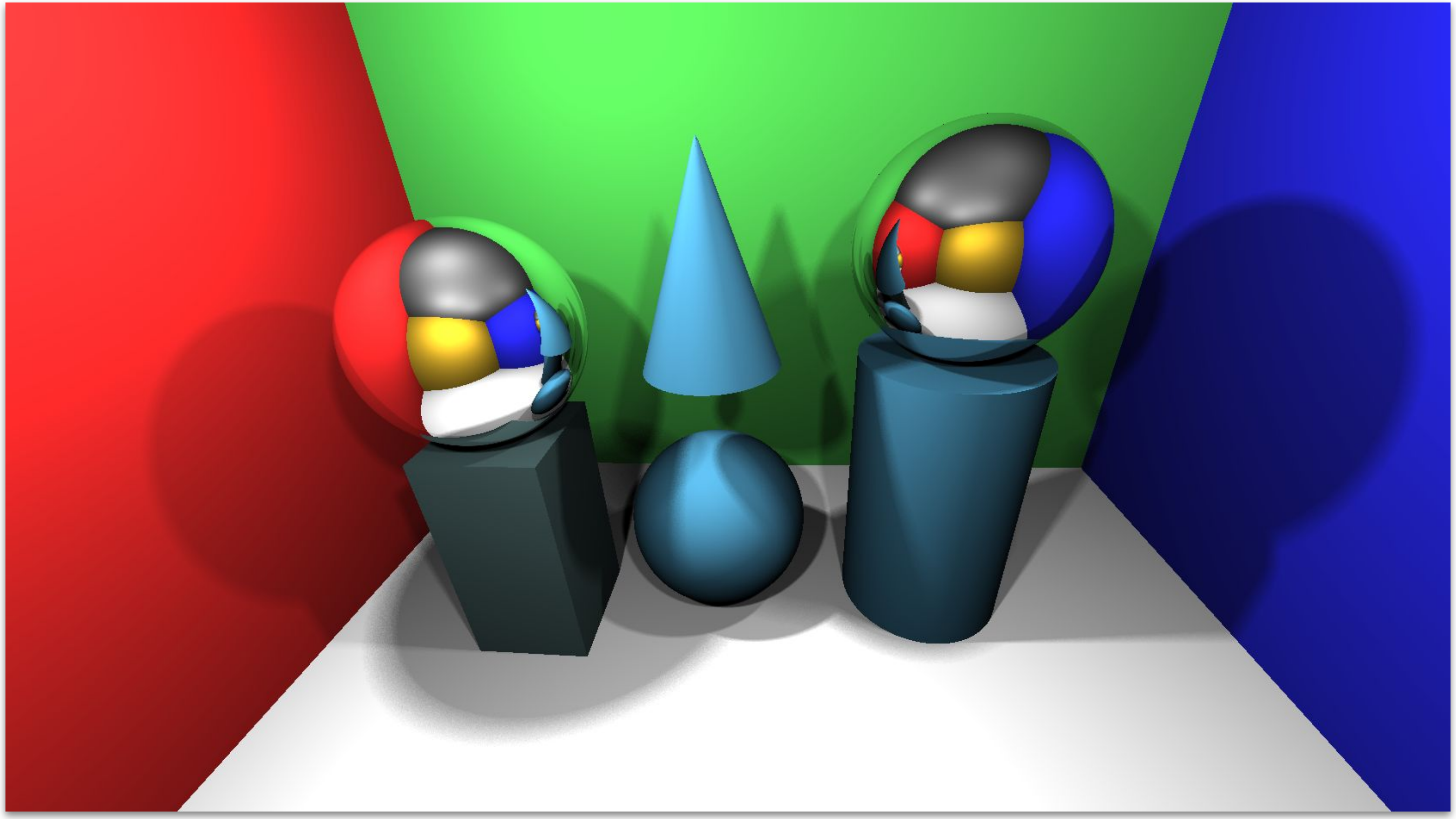


Soft Shadows: Sampling

- For point picking on a sphere using two samples drawn from [0, 1) x [0, 1) uniformly at random:
  - Map **x** to **θ ∈ [0, 2π)**
  - Map **y** to **u ∈ [-1, 1)**
  - sample = (sqrt(1 - u^2)cos(θ), sqrt(1 - u^2)sin(θ), u)

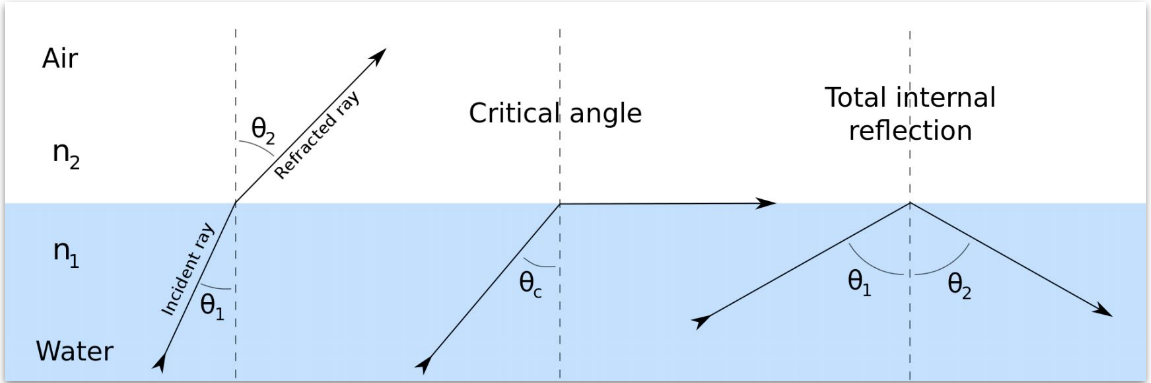


Soft Shadows



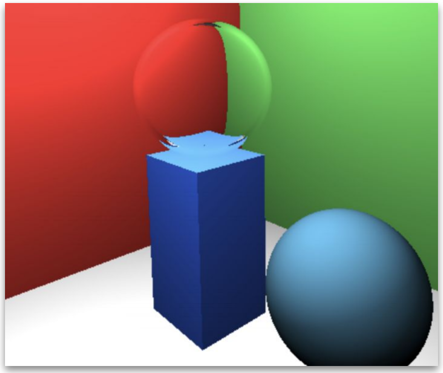
Transmission

- Transparent materials like glass **refract** light
- When light passes from one refractive medium to another, the angle of refraction is determined by the ratio of the **refractive indices** between the media
- Use **Snell's law** to compute the direction of the outgoing ray



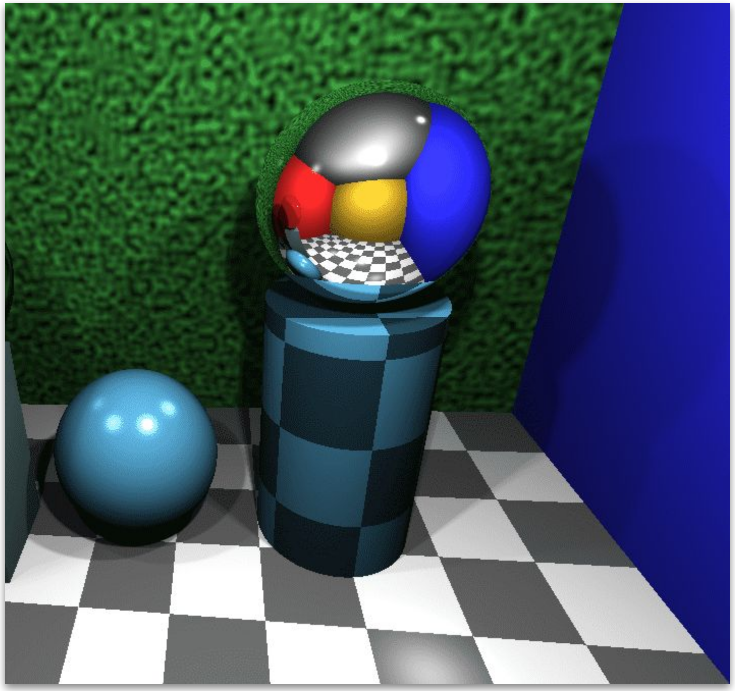
Transmission

- In reality, most transmissive materials also reflect light, as determined by the Fresnel equations from E&M
  - Hard to compute, so industry path-tracers tend to use Schlick's approximation to estimate the T/R ratio
  - Don't worry about this for A3, unless you want to go the distance!
- At some point, Snell's law will break down because it is impossible to take an arcsin of a value greater than one. This threshold is the **critical angle**
- Angles at or above the critical angle cause **total internal reflection**.
  - You will need to handle this
  - **Return a mirror bounce!**



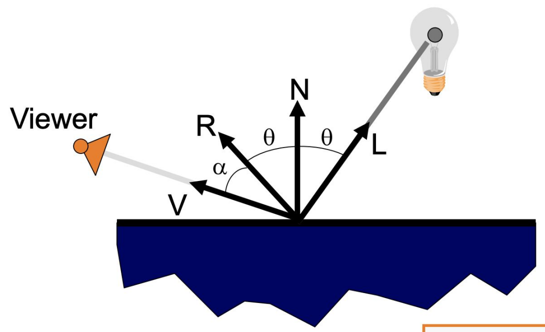
Checkerboard

- Given your intersection point, quantize it to the unit grid, and then scale the result (optional).
  - Might need to add eps to the intersection point to prevent speckeling
- Depending on the parity of its x, y, z coords summed up, choose a color!
- Result should look like a 3D chess board



Phong Reflectance Model

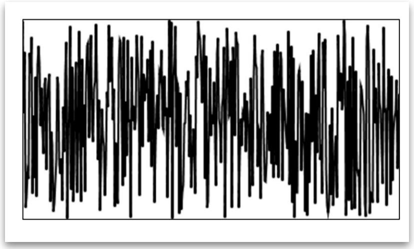
- Many non-reflective surfaces still reflect concentrated blobs of light, known as specular highlights
- The Phong reflectance model approximates this look
  - Physically motivated, but not physically accurate (doesn't conserve energy!)



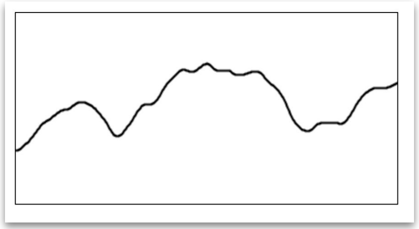
$$\boxed{I_S = K_S (V \cdot R)^n I_L}$$

Natural Textures

- How can we use randomness to procedurally generate natural textures?
- Uniform randomness doesn't look good
  - Too machine-like



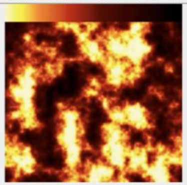
- Key insight: the scale of artifacts due to the randomness should be locally similar

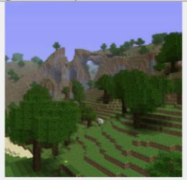


Perlin Noise

- Developed by Ken Perlin; won an academy award
  - Used in video games and CGI for natural textures
  - Used to help generate procedural landscapes
  - Now belongs to a class of type of **gradient noise** functions

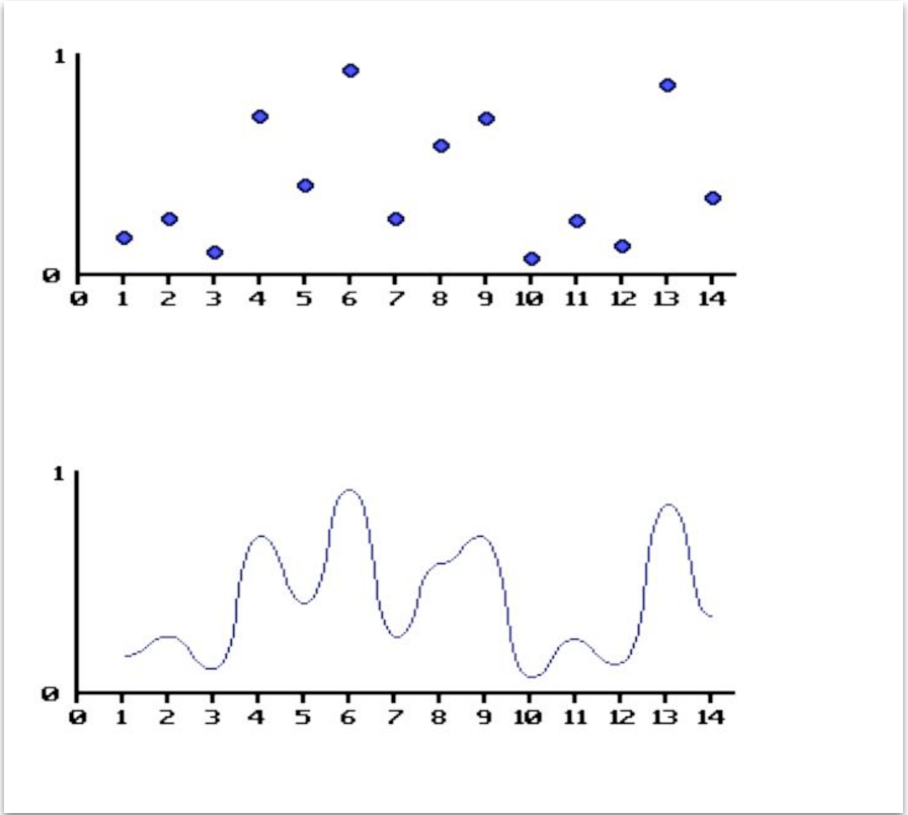
Using noise as an offset to create handwritten lines.


By applying a simple gradient, a procedural fire texture can be created.


erhaps the quintessential use of Perlin noise today, terrain can be created with cave
and caverns using a modified Perlin Noise implementation.

Perlin Noise

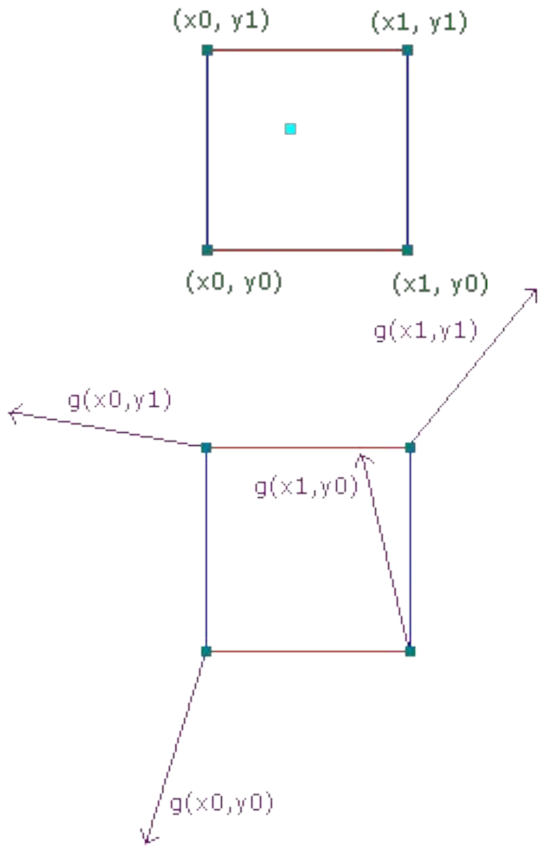
- Key idea:
  - First generate random points
  - Then smoothly interpolate (fade) between the points



Perlin Noise

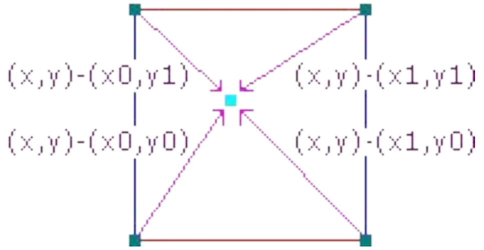
- For a 2D point:
  - Cut to grids
    - Scale this as needed to adjust sampling frequency
  - Assign each grid vertex a random gradient vector from:

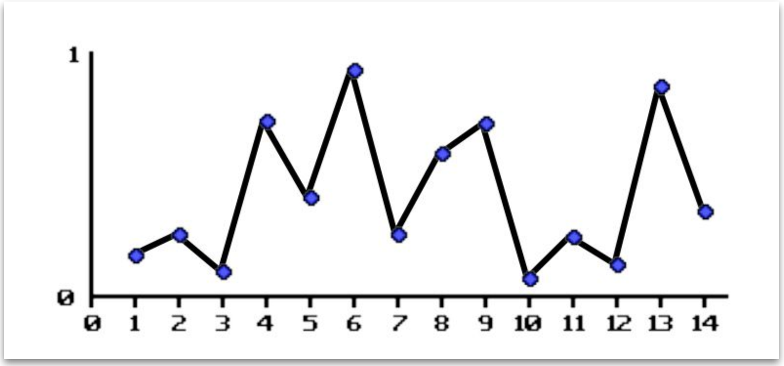
    (1,1,0),(-1,1,0),(1,-1,0),(-1,-1,0),
    (1,0,1),(-1,0,1),(1,0,-1),(-1,0,-1),
    (0,1,1),(0,-1,1),(0,1,-1),(0,-1,-1)
  - The random selection should be **determined** by grid vertex position



Perlin Noise

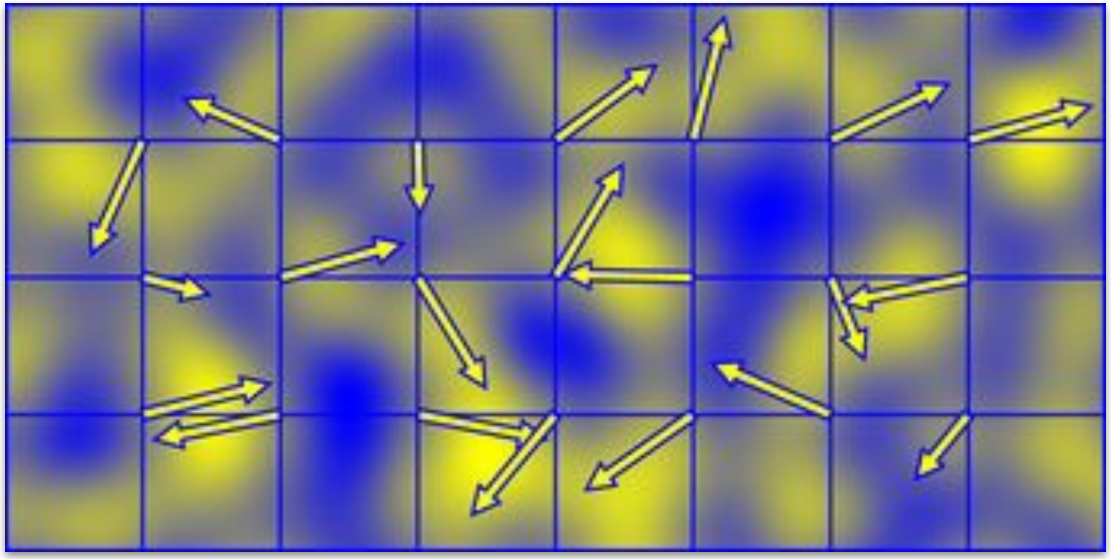
- For a 2D point (continued):
  - Find distance vectors for each grid vertex (w.r.t. the point being sampled)
  - Dot distance vectors for a given vertex with the vertex's gradient
  - Interpolate results (think bilinear sampling), but run the normal "linear" alpha through the fade function:
    - $\alpha_{new} = 6\alpha_{old}^5 - 15\alpha_{old}^4 + 10\alpha_{old}^3$





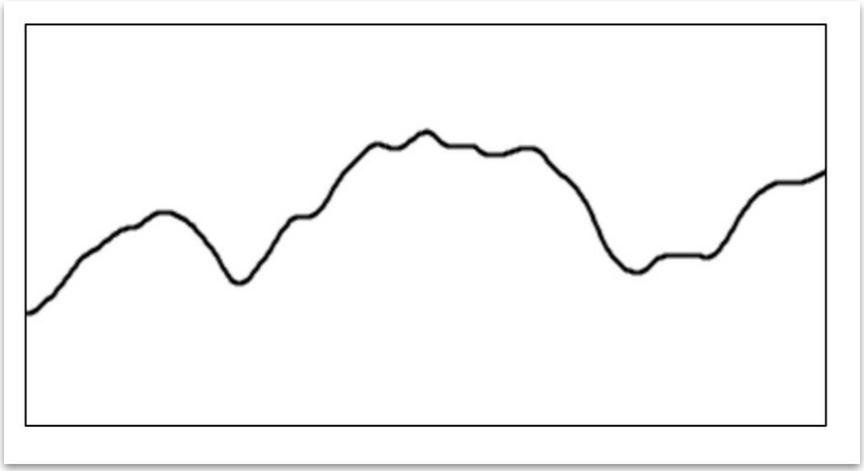
Perlin Noise

- Result (with gradients and grid visualized):



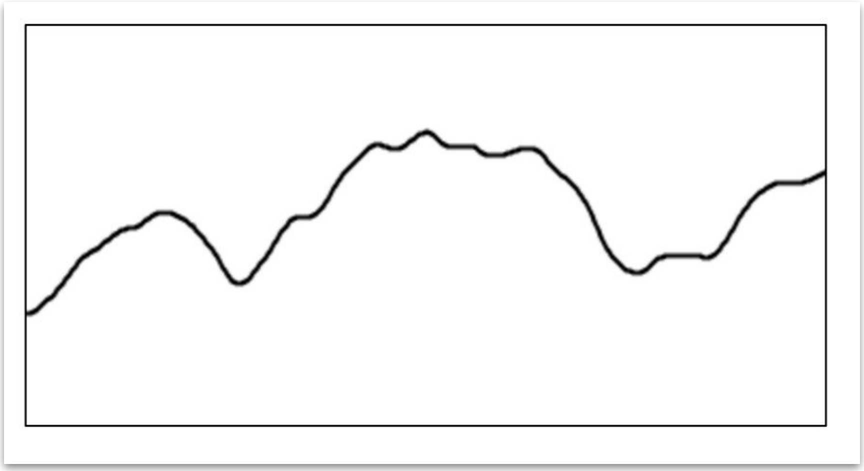
Procedural Generation

- Now let’s try to procedurally generate a mountain range!
- Does this Perlin noise look like a mountain range?



Procedural Generation

- Does this Perlin noise look like a mountain range?



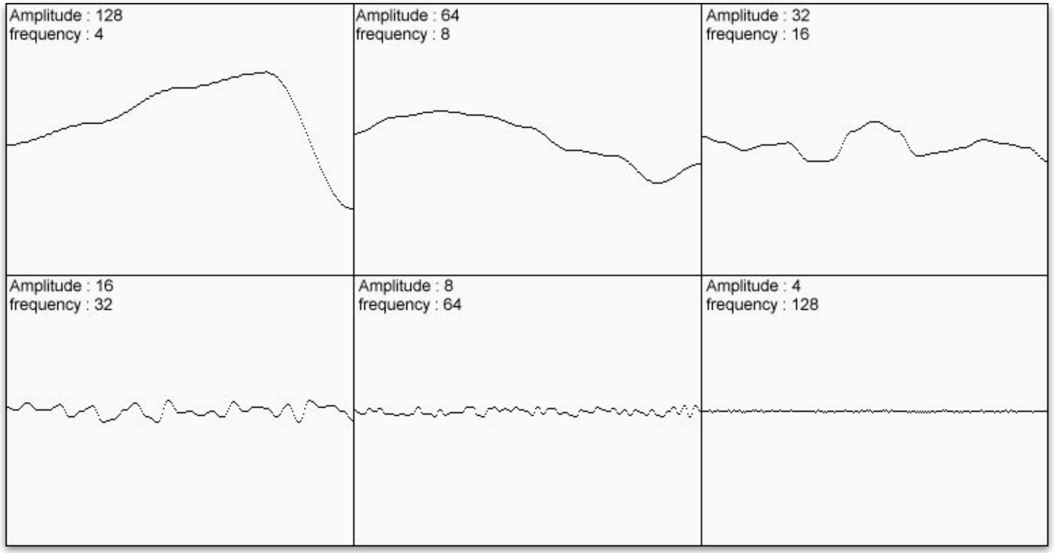
- No!
- A real landscape is like a fractal, with details as large as mountains to as small as gravel

Procedural Generation

- Key idea: add up our Perlin noise sampled for different levels (called **octaves**)
- High frequency info (like rocks and pebbles) should have low amplitudes (small height)
- Low frequency info (like mountains and valleys) should have high amplitudes (large height)

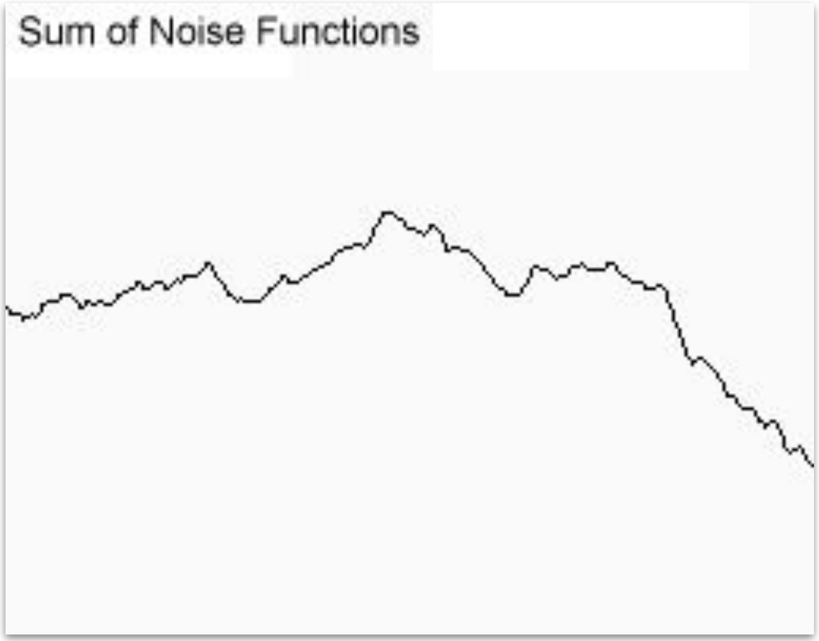
Procedural Generation

- As an example, here are some samples at different octaves:



Procedural Generation

- And here is the result:



Procedural Generation

- We have now created **fractional Brownian Motion** (or **fractal** Brownian motion), abbreviated as fBM
- Brownian motion is a random walk where the direction of your next step is independent of every other step
- fBM is similar, but your next step can be correlated or anticorrelated to your previous step

Procedural Generation

- fBM correlation is determined by the **Hurst Exponent**, H, which takes values between 0 and 1.
  - H = 0: volatile (anticorrelated)
  - H = ½: Brownian motion (uncorrelated)
  - H = 1: smooth (correlated)
- For efficiency, fBM usually uses **gain**, $G = 2^{-2H}$
  - Thus, G ranges between .5 (smooth) and 1 (rough)
  - Also called **persistence**
- Read more here:
  - http://www.iquilezles.org/www/articles/fbm/fbm.htm

Procedural Generation

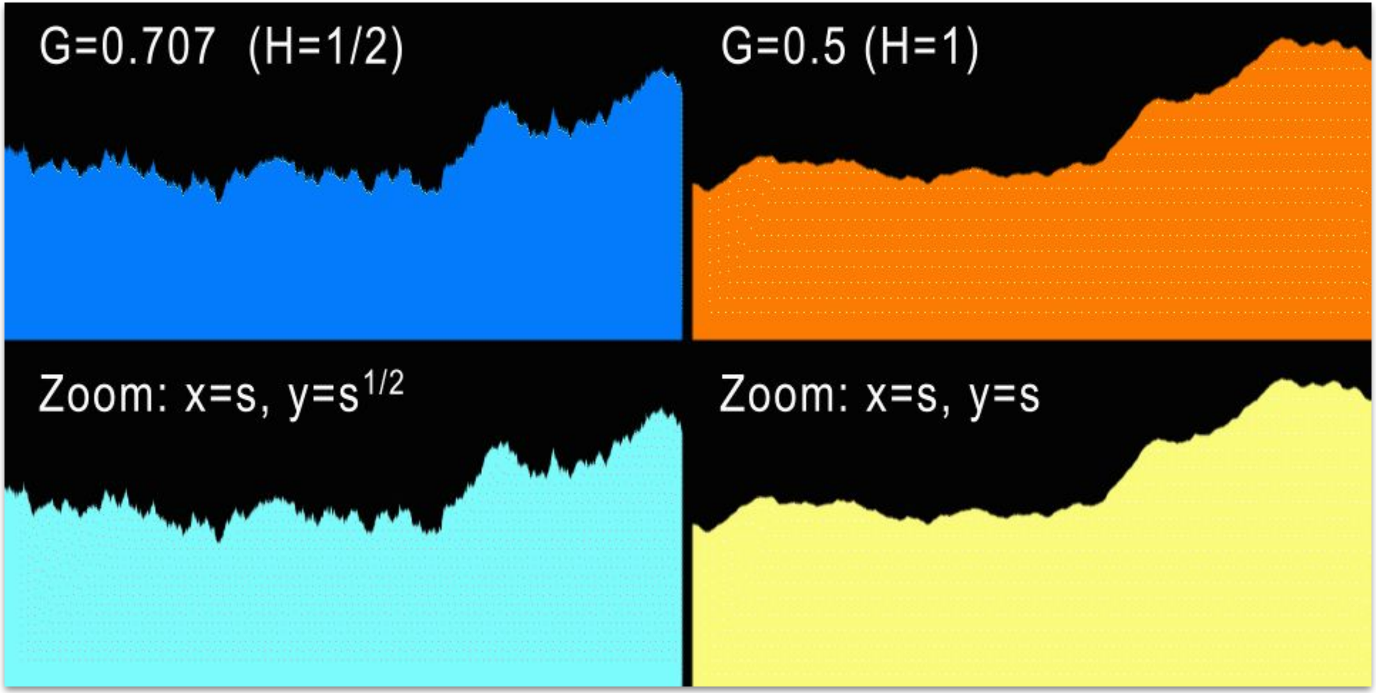
- Fractal Brownian Motion

```
// Properties
const int octaves = 8;
float lacunarity = 2.0; // How quickly width shrinks
float gain = 0.5; // How slowly height shrinks

// Initial values
float amplitude = 0.5;
float frequency = 1.;

// Loop of octaves
for (int i = 0; i < octaves; i++) {
     y += amplitude * noise(frequency*x);
     frequency *= lacunarity;
     amplitude *= gain;
}
```

- **Lacunarity** affects how quickly our frequency increases
- We can use any smooth noise function!
  - Often cheaper or more advanced noise alternative to Perlin is used

Procedural Generation

- Example of fractal behavior:

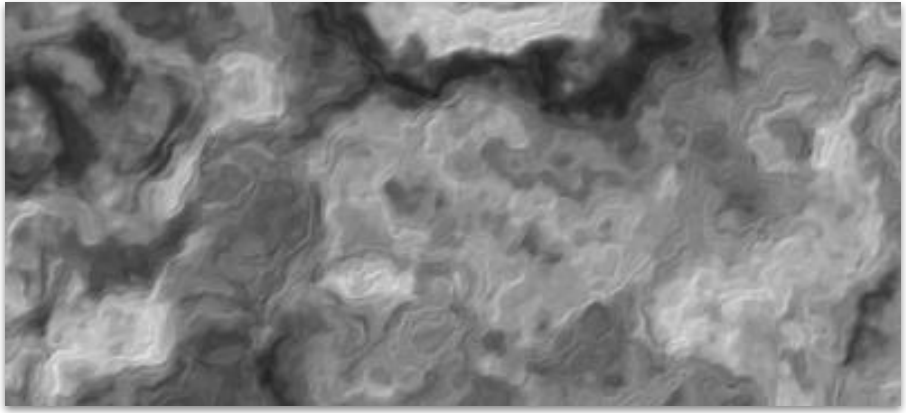


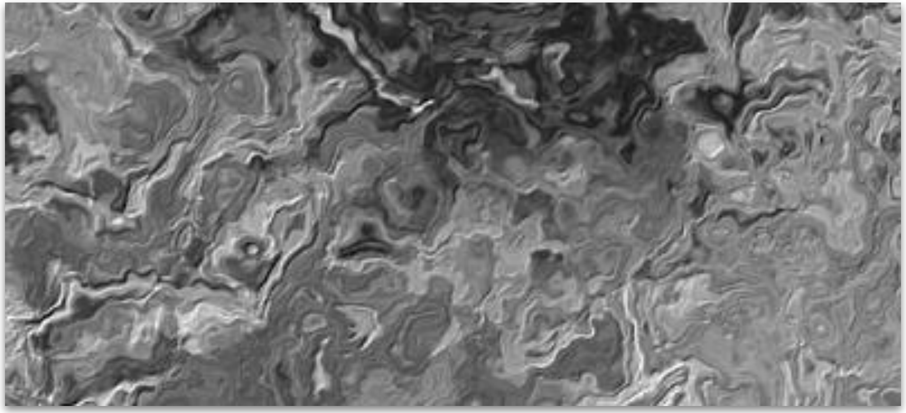
Procedural Generation

- A fractal Brownian motion texture:
  - f(p) = fbm(p)

Procedural Generation

- We can even compose our fbm functions:
  - g(p) = fbm(p + fbm(p))
  - h(p) = fbm(p +g(p))





Procedural Generation

- And we can add color!

Procedural Generation

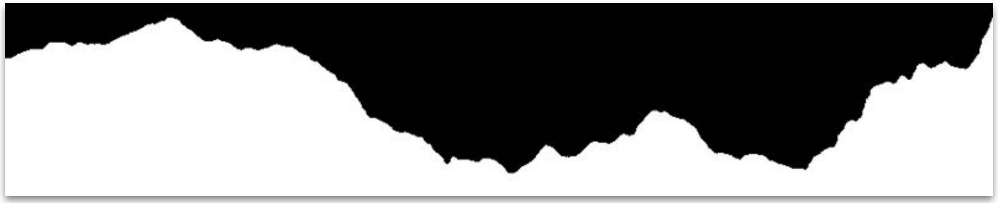
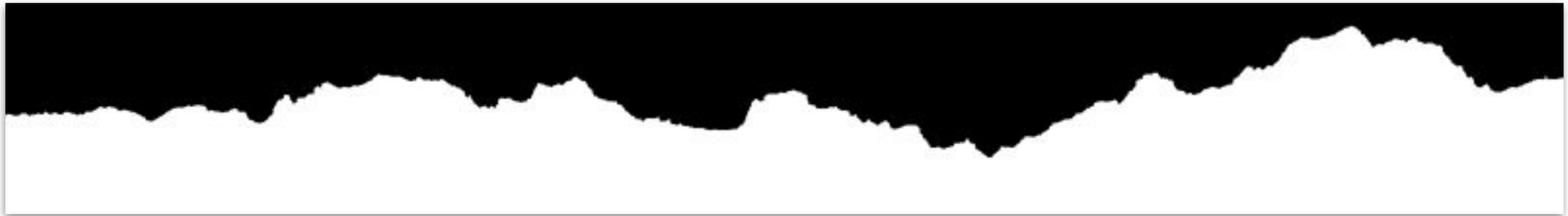
- And we can add color!

Procedural Generation

- And we can animate it:
  - For more, check out: http://www.iquilezles.org/www/articles/warp/warp.htm

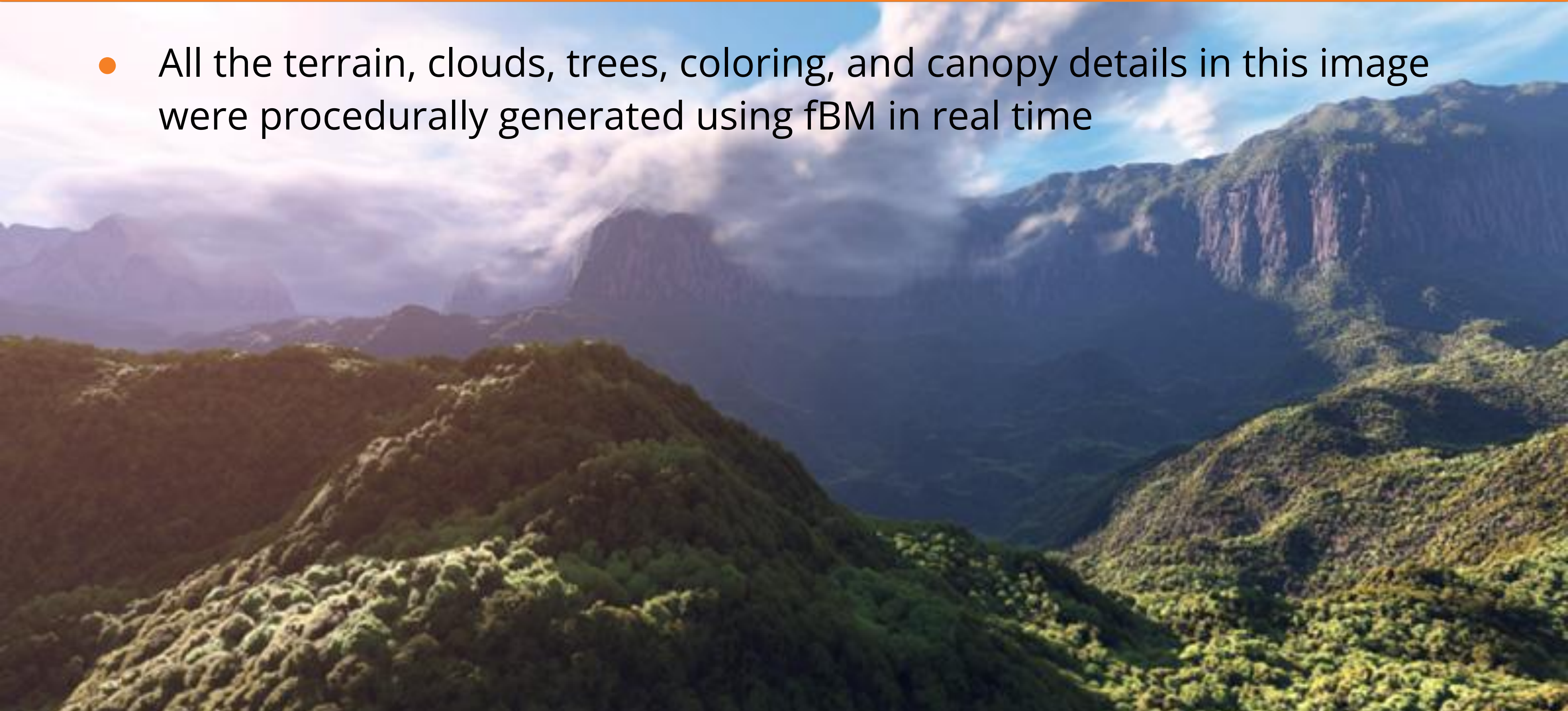
Procedural Generation

- Back to the original question: how about landscapes?
- Here are two **real** mountainous landscapes:

- Signal analysis shows they correspond to fbm with a gain of 0.5 (smooth)

Procedural Generation



- All the terrain, clouds, trees, coloring, and canopy details in this image were procedurally generated using fBM in real time

Advanced Path Tracing

- Our simple raytracer for A3 leaves out many real behaviors of light:
  - No visible lights with realistic geometries
  - No indirect illumination
    - No global illum. and no caustics (light focused by refraction)
  - No distributed/monte carlo path-tracing
    - No natural looking materials; no blurry reflections/refractions
  - Poor BRDF (bidirectional reflectance distribution function)
    - Reflections aren't physically based; materials all look plasticy
  - No sub-surface scattering (ray bounces under and out of the surface)
    - This is what makes velvet look like velvet; same for skin
  - No volumes like water or smoke
  - No camera effects like motion blur or depth of field

Advanced Path Tracing

- It would be infeasible to cover the last 20 years of path tracing in this slide deck but these two papers from Disney/Pixar are incredible. Between these papers and their references, you could probably build a super advanced path tracer on your own:
  - History of path tracing and path tracing today
  - Disney BSDF (BRDF + surface properties)
  - Disney BSDF coding walkthough
- Today, films are finishing up the shift to **single-pass path tracing** as the default rendering technique

Advanced Path Tracing

- Advanced single-pass path-tracer in a nutshell:
    - Until convergence:
        - Trace a ray from the camera
        - Evaluate the surface contribution at the intersection
        - Play russian roulette: if lose, kill the ray (base case)
        - Otherwise, randomly sample the BSDF (bidirectional scattering distribution function) for a bounced ray, and recur on that ray
        - Add the weighted contribution from that bounce

Advanced Path Tracing

- COS 526: Advanced Computer Graphics will take you up to here
- Note:
  - Fresnel effects!
  - Glossy reflection!
  - Global illumination (uses **photon mapping**)
  - Caustics!
- Unfortunately not planned for Fall 2022. :(