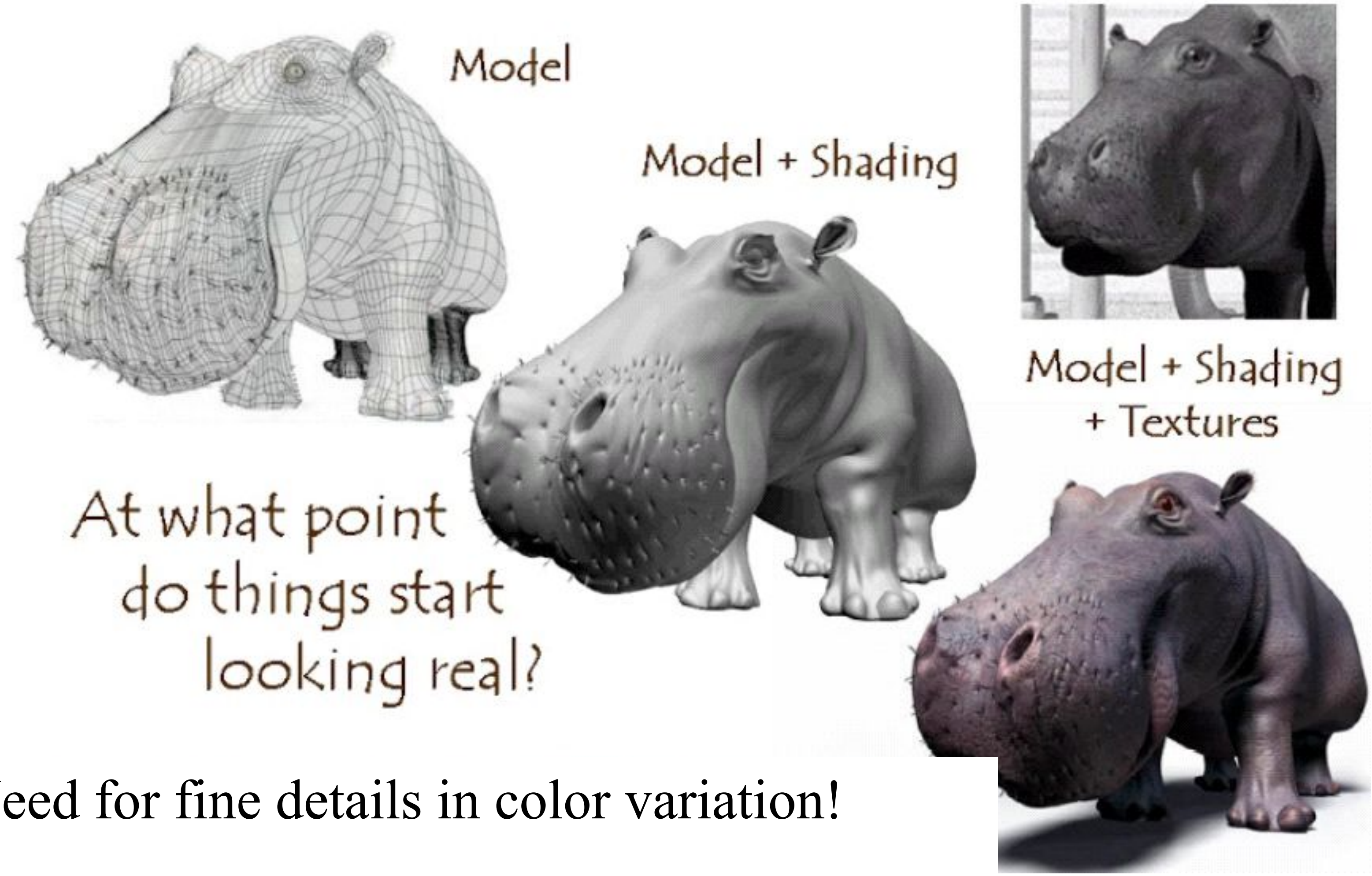
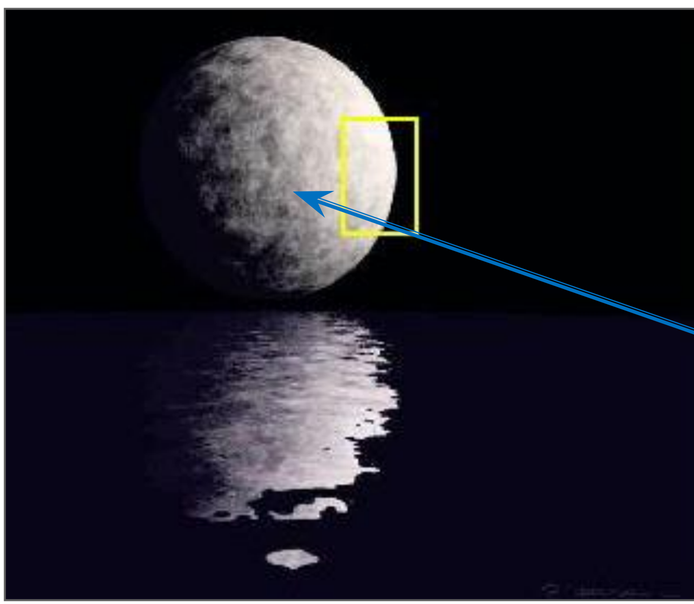
1. Quest for Visual Realism



Need for fine details in color variation!

Motivations
2. Limit the number of polygons

Problem: Everything cannot be modeled at the scale of geometry!



Micro-polygons would be needed!

Textures

Modeling changes of material

Enable the material attributes to vary inside a face

– The local attribute will be used during rendering

Examples of attributes: color, shininess, normals, transparency…

Still a single face, but several colors

Textures

Modeling changes of material

Enable the material attributes to vary inside a face

- The local attribute will be used during rendering

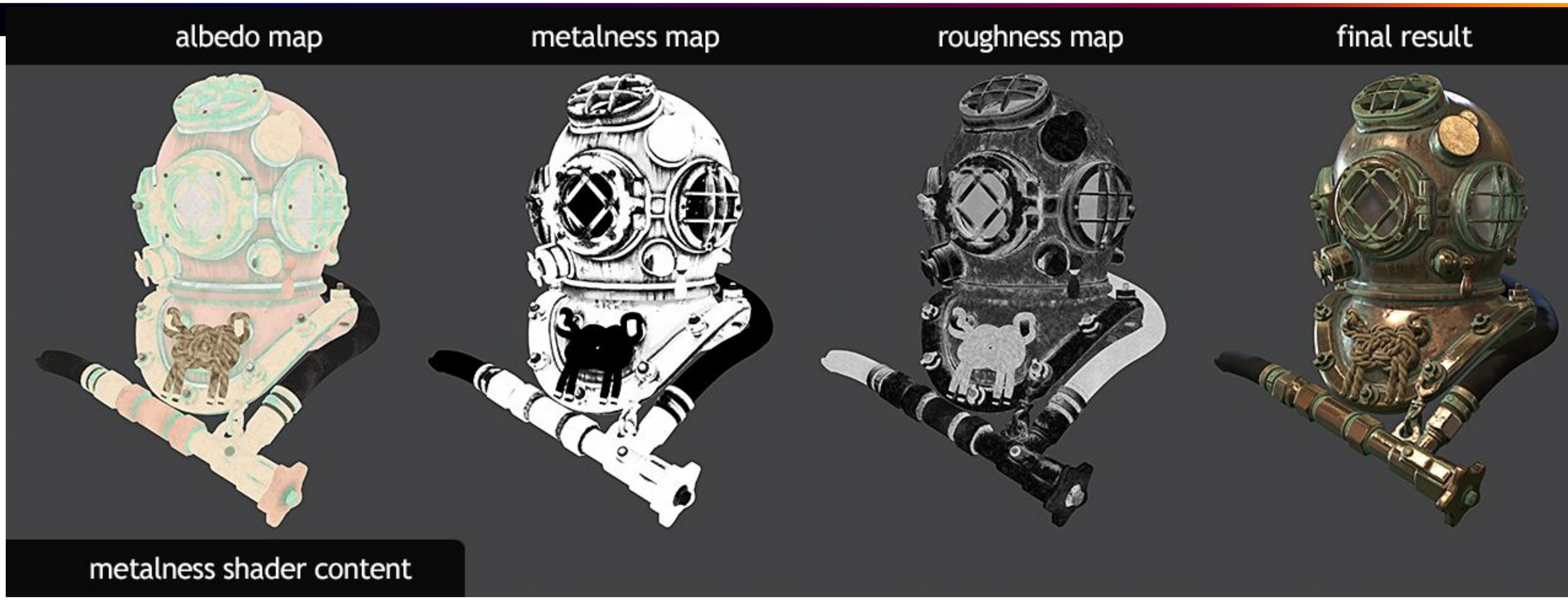
Examples of attributes: color, shininess, normals, transparency…

Typically:

$$L(\mathbf{p} \rightarrow \mathbf{e}) = K_a + \sum_s [K_d(\mathbf{n} \cdot \boldsymbol{\ell}) + K_s(\mathbf{r} \cdot \mathbf{e})^n] I_s$$

ambiant diffuse specular

Textures
Modeling changes of material



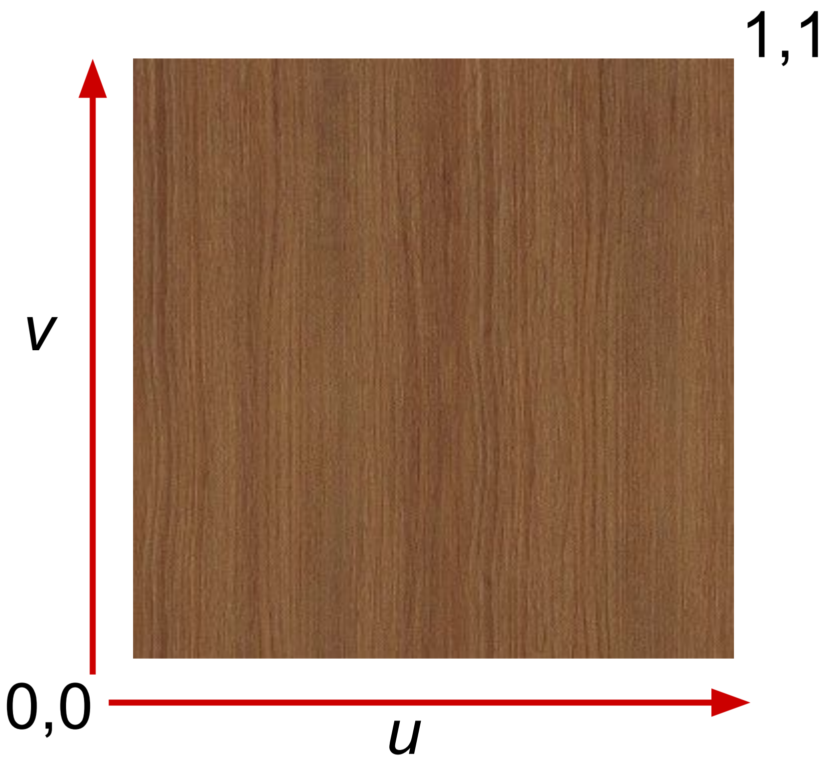
https://www.marmoset.co/posts/pbr-texture-conversion/

$$L(\mathbf{p} \rightarrow \mathbf{e}) = K_a + \sum_s [K_d(\mathbf{n} \cdot \boldsymbol{\ell}) + K_s(\mathbf{r} \cdot \mathbf{e})^n] I_s$$

ambiant　　diffuse　　specular

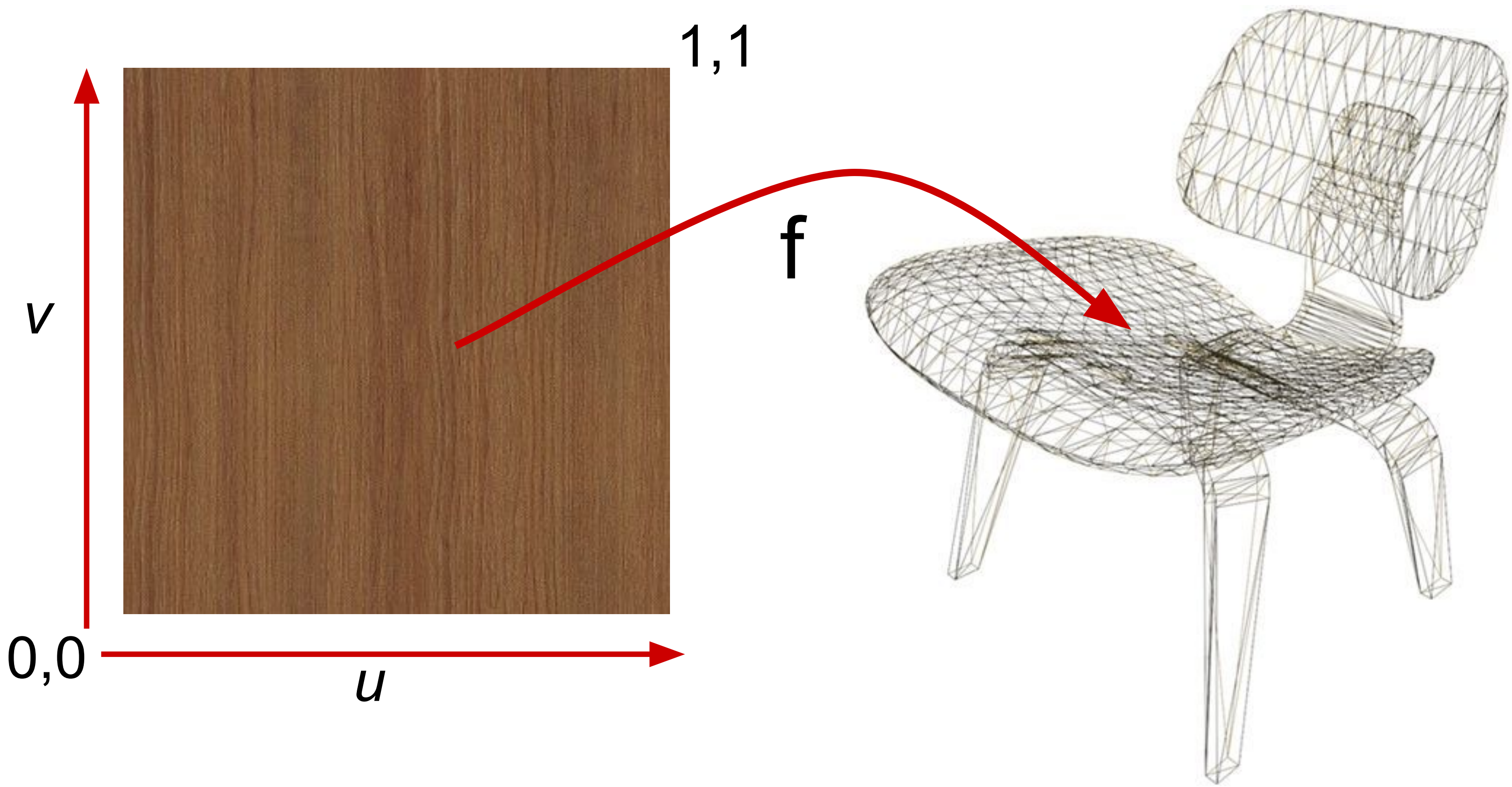
2D Textures: picture of attributes + mapping

- Planar image I (u,v)



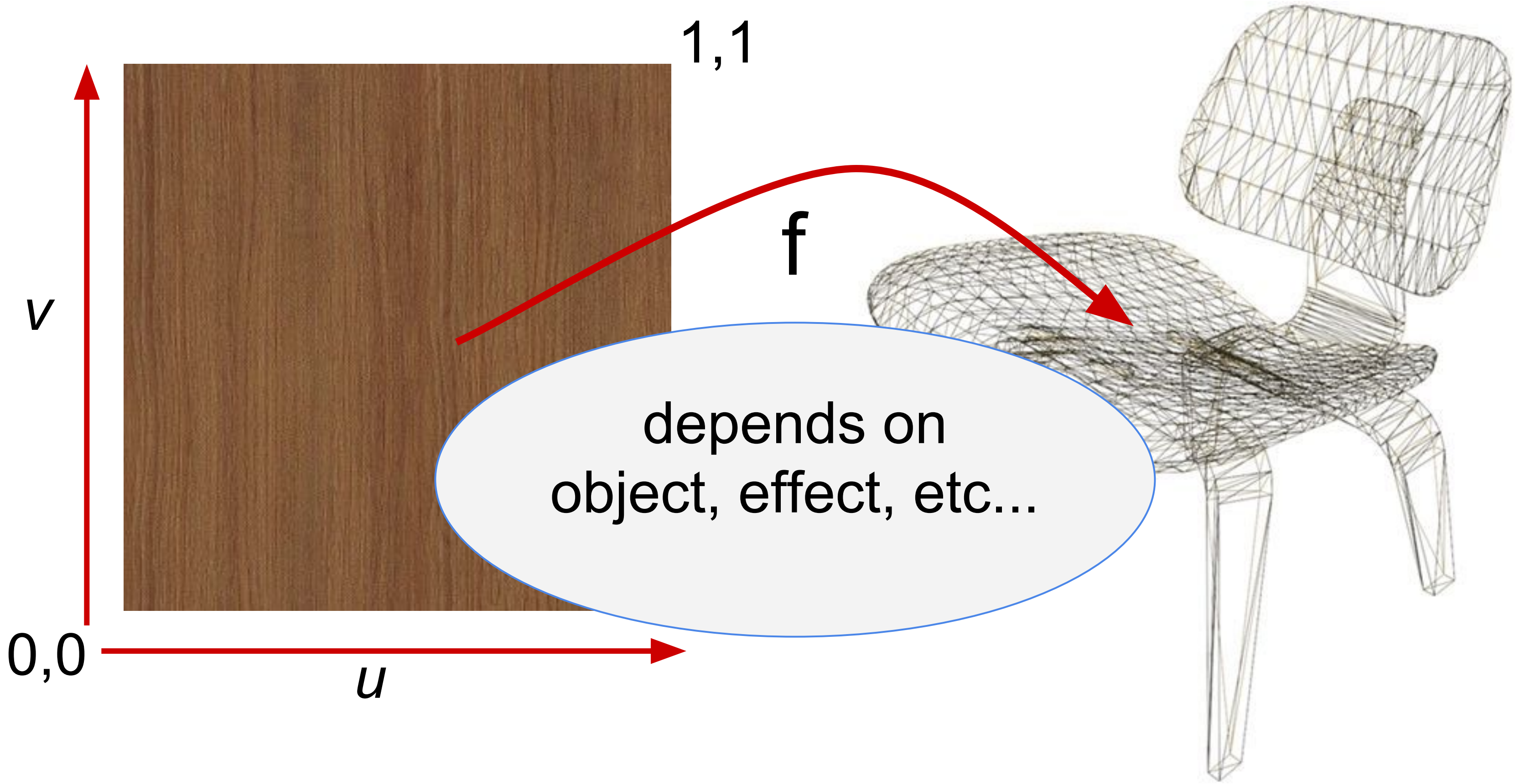
2D Textures: picture of attributes + mapping

- Planar image I (u,v)+ mapping f: P(x,y,z) → (u,v)



2D Textures: picture of attributes + mapping

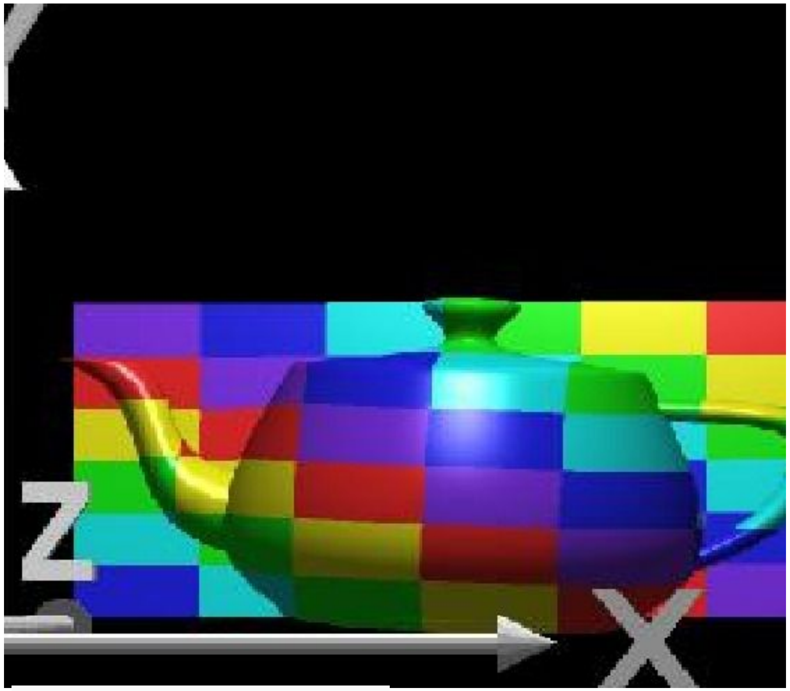
- Planar image I (u,v)+ mapping f: P(x,y,z) → (u,v)

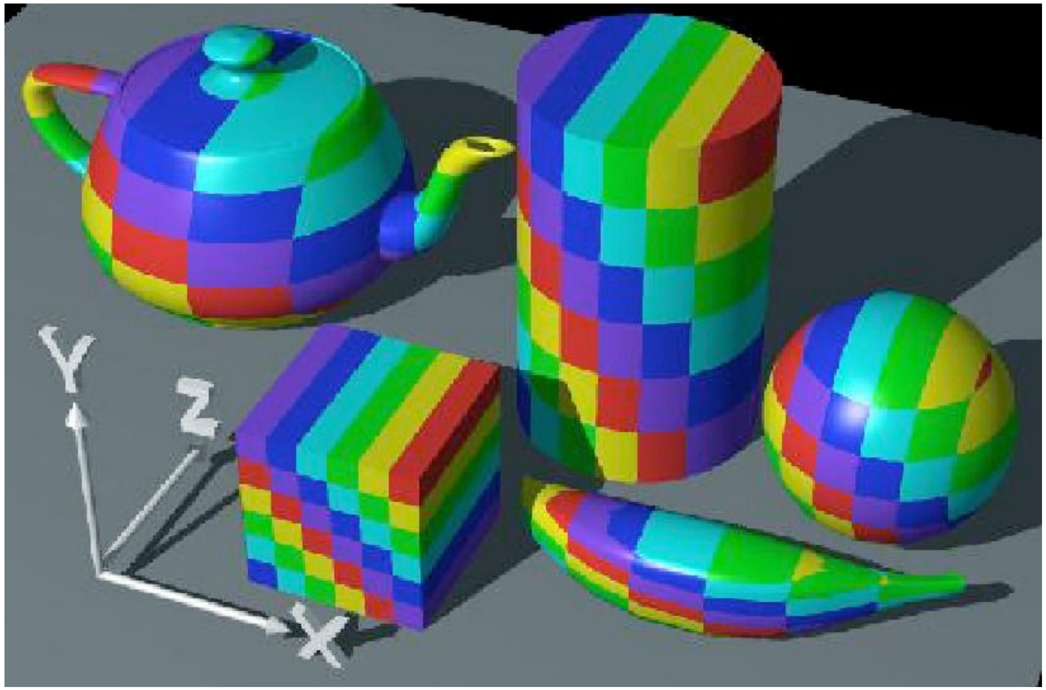


Mapping function: flat/planar mapping

f: (x,y,z) → [0,1] x [0,1]

- Planar mapping: **f (x,y,z) = (x , y)**

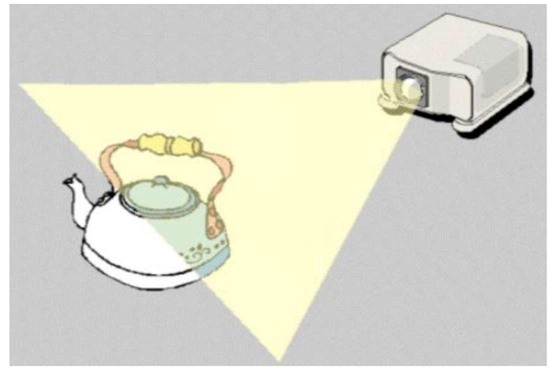




Rosalee Wolfe / siggraph.org

Mapping function: projective mapping

Map the texture like with a slide projector



Advantage

- No need for texture coordinates!

Inc

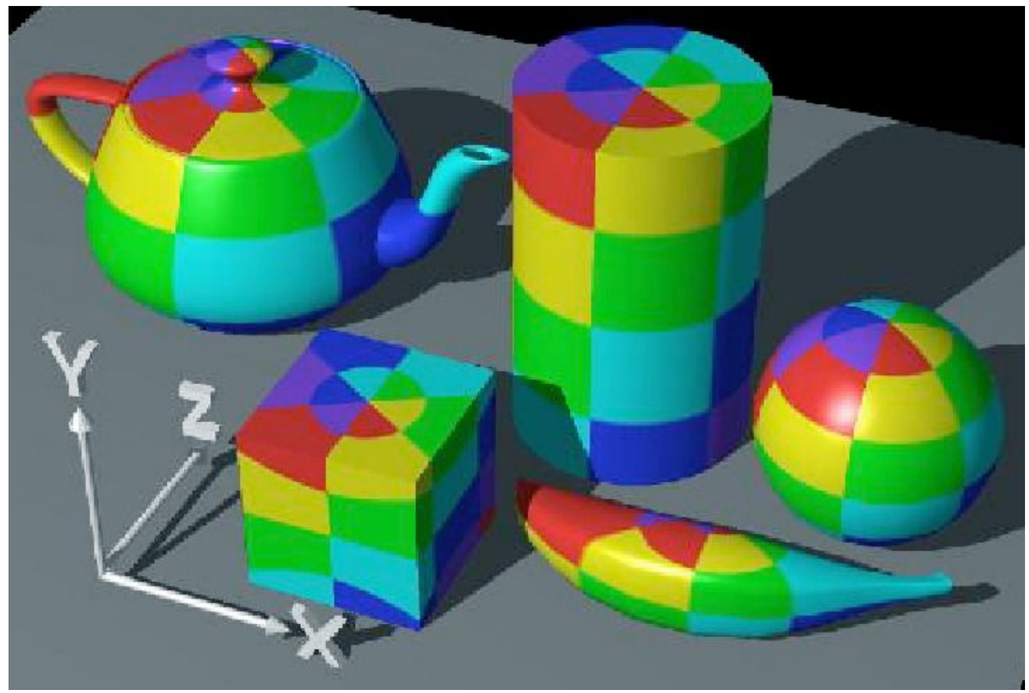
- One viewpoint
- Distortions
- Blending

Original photograph with marked edges | Recovered model | Model edges projected onto photograph | Synthetic rendering

Mapping function: spherical mapping

f: (x,y,z) → [0,1] x [0,1]

- Spherical mapping: **f (θ,ψ) = (θ/2π , (π/2 – ψ) / π/4)**

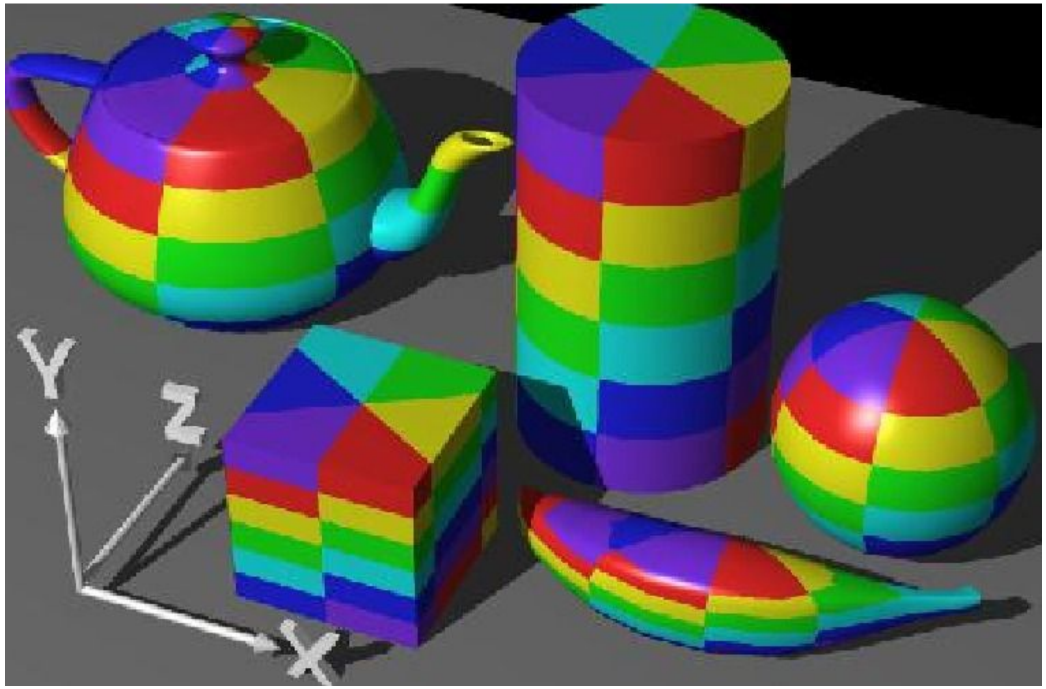


Rosalee Wolfe / siggraph.org

Mapping function: cylindrical mapping

f: $(x,y,z) \rightarrow [0,1] \times [0,1]$

- Cylindrical mapping: **$f(\theta,z) = (\theta/2\pi, z)$**

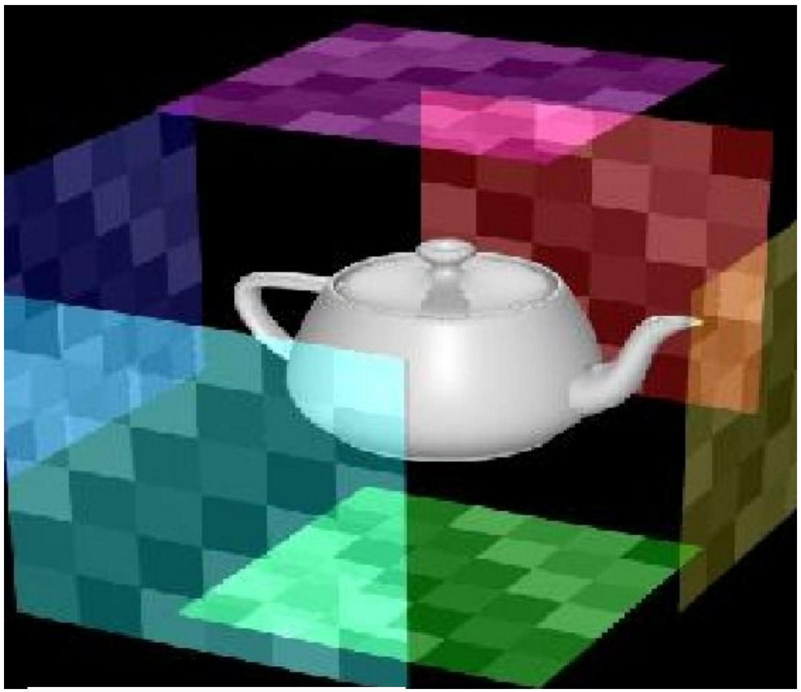


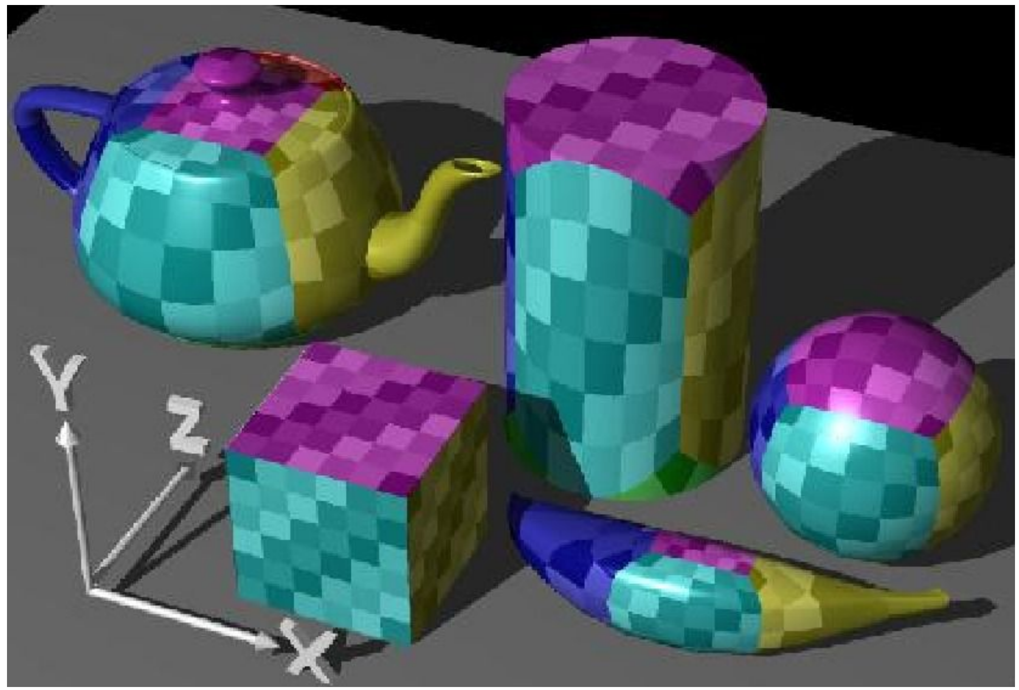
Rosalee Wolfe / siggraph.org

Mapping function: cube mapping

f: (x,y,z) → [0,1] x [0,1]

- Cube mapping: **depends on x,y,z signs**



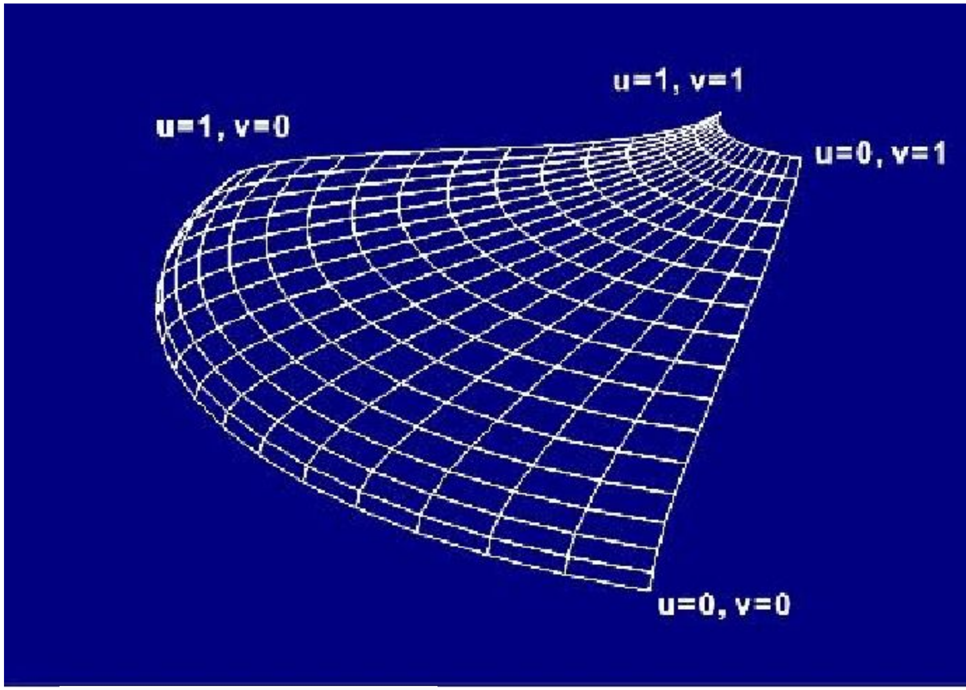


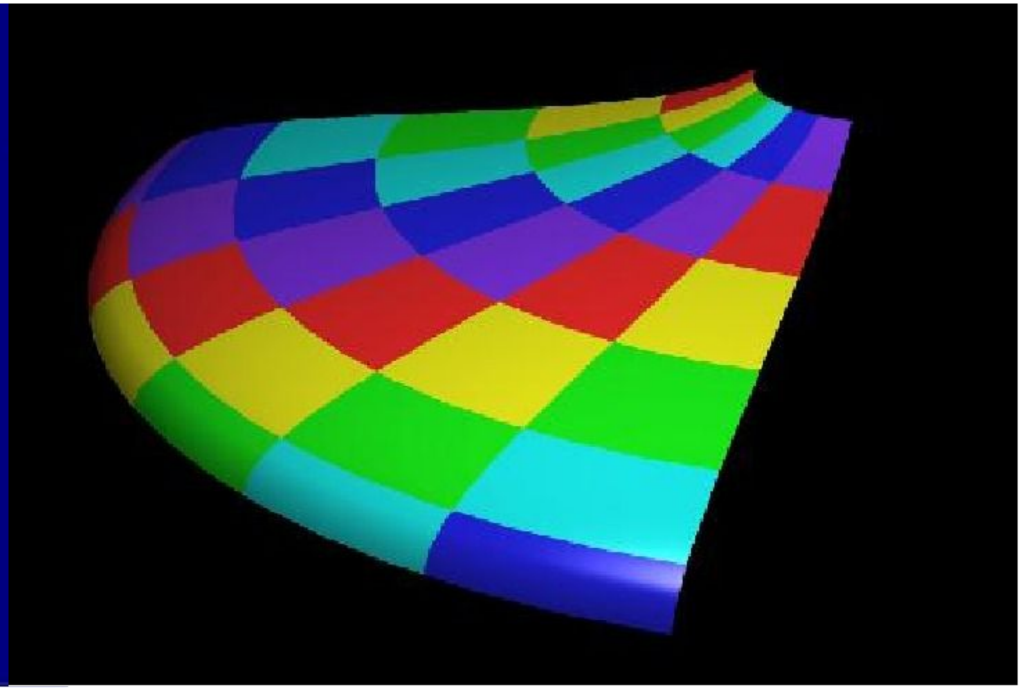
Rosalee Wolfe / siggraph.org

Mapping function: parametric mapping

$f: (x,y,z) \rightarrow [0,1] \times [0,1]$

- Parametric mapping: $f(S(u,v)) = (u,v)$



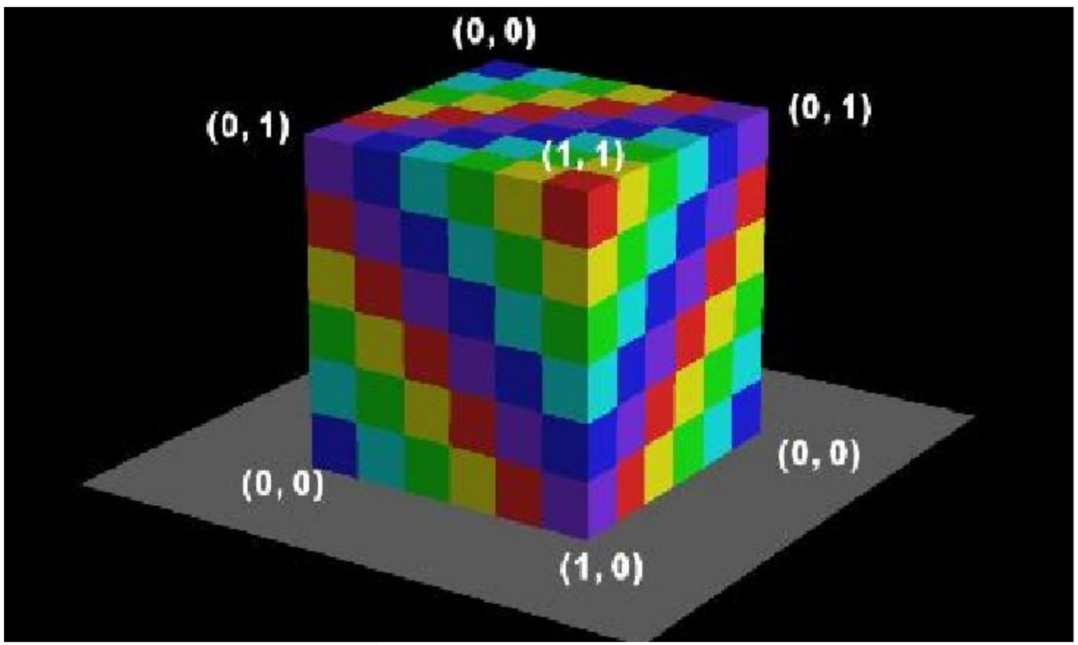


Rosalee Wolfe / siggraph.org

Mapping function: uv mapping

f: (x,y,z) $\rightarrow$ [0,1] x [0,1]

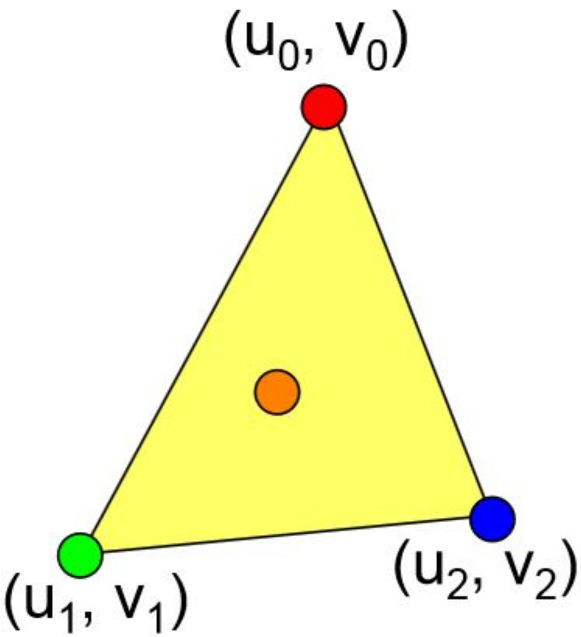
- UV mapping: define uv at each vertex and exploit rasterization

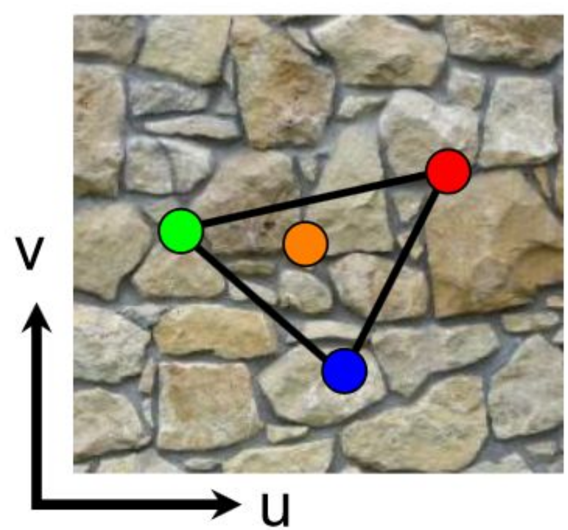


Rosalee Wolfe / siggraph.org

2D Textures: picture of attributes + mapping

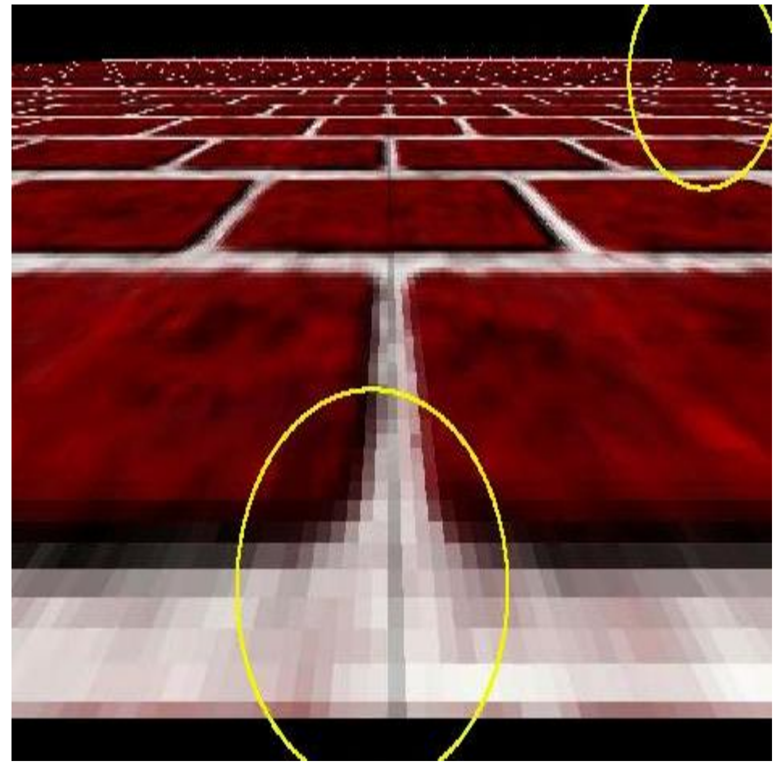
- Planar image I (u,v) + mapping f: P(x,y,z) → (u,v)
- Store: mesh point + normal + texture coordinates (u,v)
- In a face, interpolate (u,v) using barycentric coords





Aliasing problems

Brick wall picture mapped on one face



At the front

- The texture pixels can be seen

At the back

- Many colors in the same pixel
- The one at the center is picked!

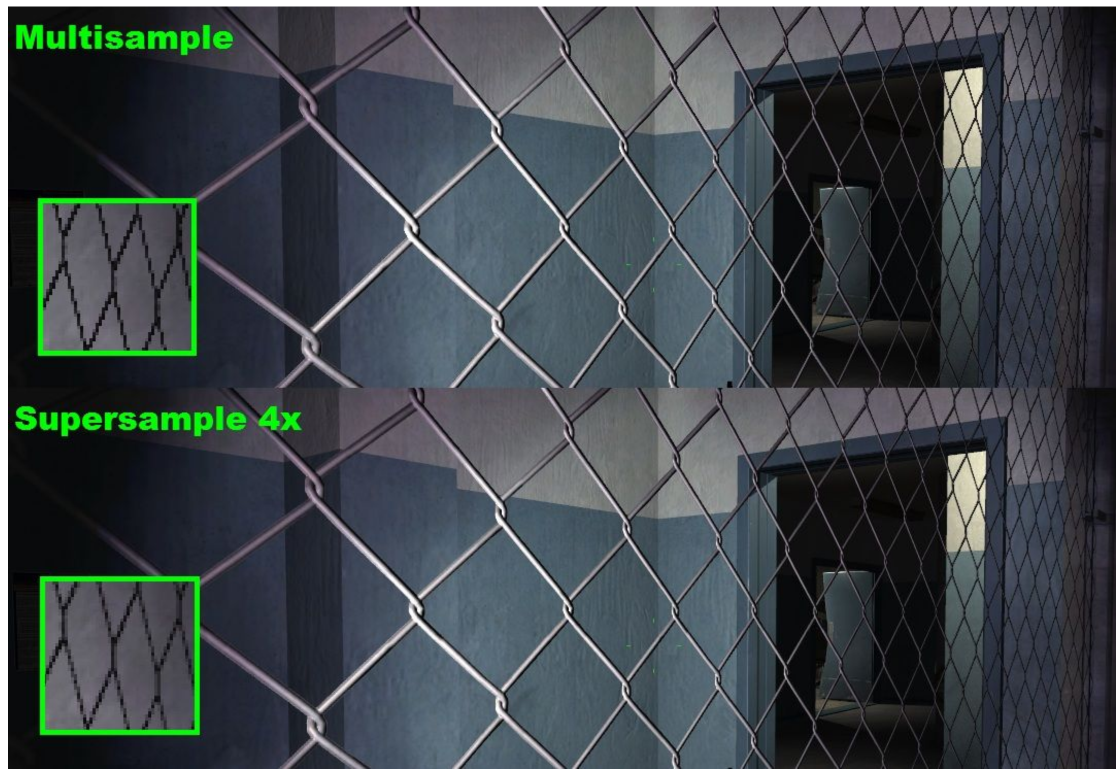
Aliasing problems

Solution 1: Pre-filtering

- compute multiple samples per pixel
- and average result

Advantage: "ground truth"

Drawback: expensive!



Aliasing problems

Solution 2: Pre-filtering

- Pre-compute an "image pyramid" (mip-map) of the texture: down sampling
- Pick the best texture resolution while rendering (rasterization phase)

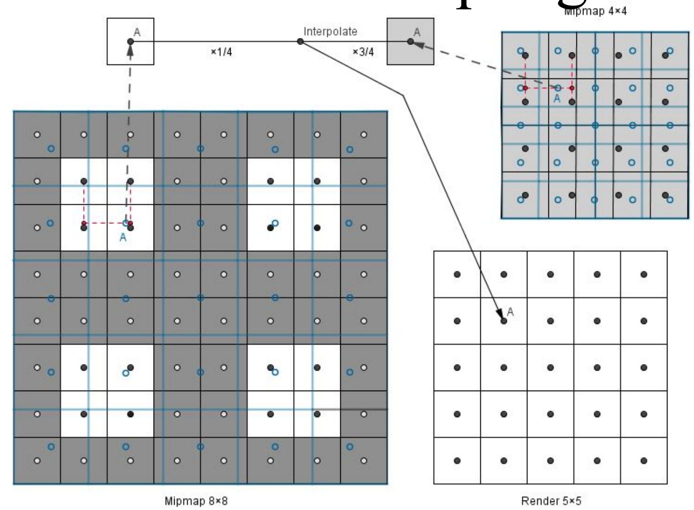
Advantage: fast
Drawback: incorrect filter

256x256 128x128 64x64 32x32

mip-map sampling

- Multiple options
  - Nearest scale, nearest neighbor texel sampling
  - Nearest scale, bilinear texel sampling
  - Trilinear sampling
- Trilinear sampling
  - Find nearest two scales
  - Bilinear sampling in each scale
  - Linear interpolation of the result

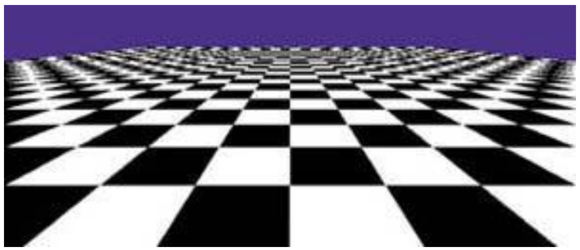
Trilinear sampling



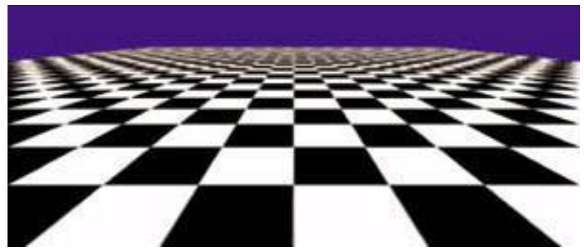
https://cglearn.codelight.eu/pub/textures-and-sampling

Mip-mapping

- Andrew Flavell has a nice (old) article on mip-mapping
  http://www.gamasutra.com/view/feature/131708/runtime_mipmap_filtering.php



Without mip-mapping



With mip-mapping

Aliasing problems

Solution 3: Post-filtering

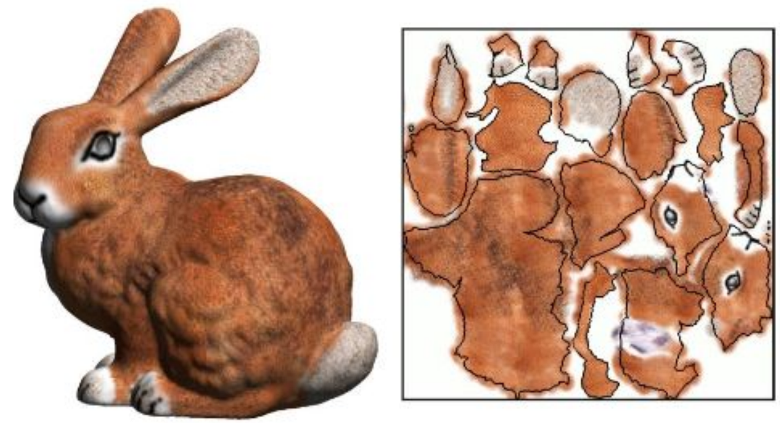
- screen-space anti-aliasing (SSAA)
- multiple algorithms
- more and more used

Advantage: fast, GPU friendly

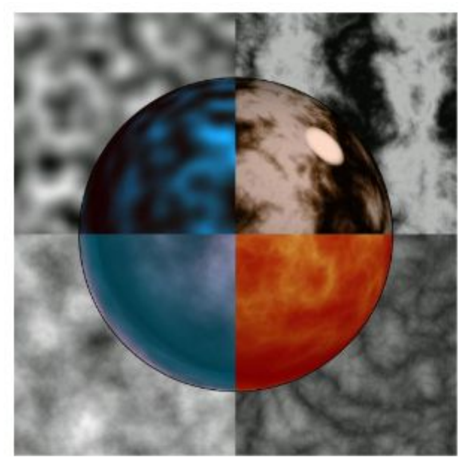
Drawback: cannot handle all types of artifacts

Creating a texture

- From real data
  - colors/normals/coefs,
  - stored in 2D images

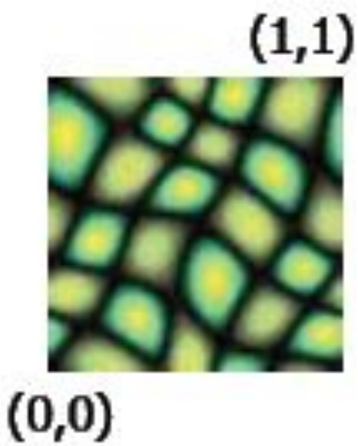


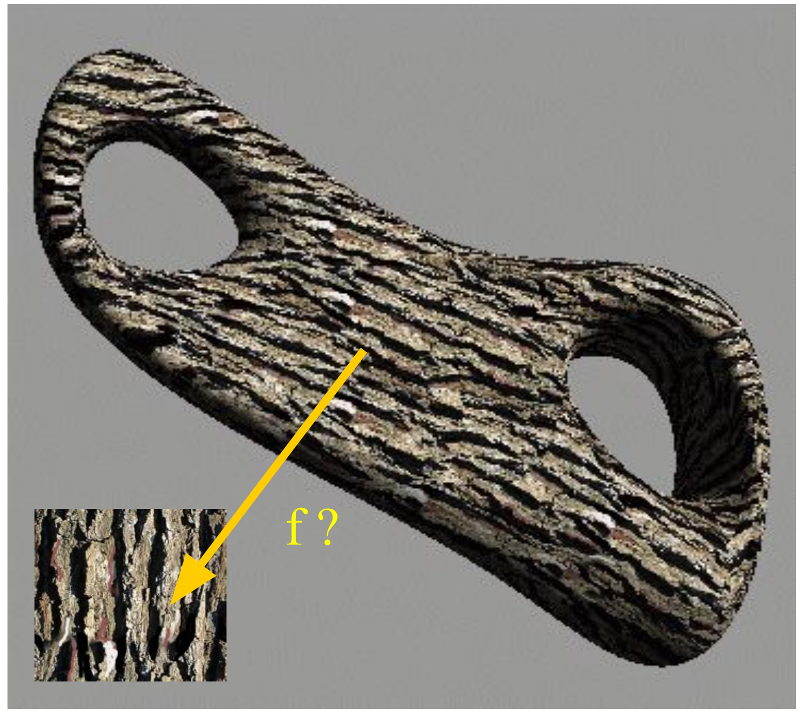
- Proceduraly
  - using a small program
  - usually on the GPU



Creating a texture

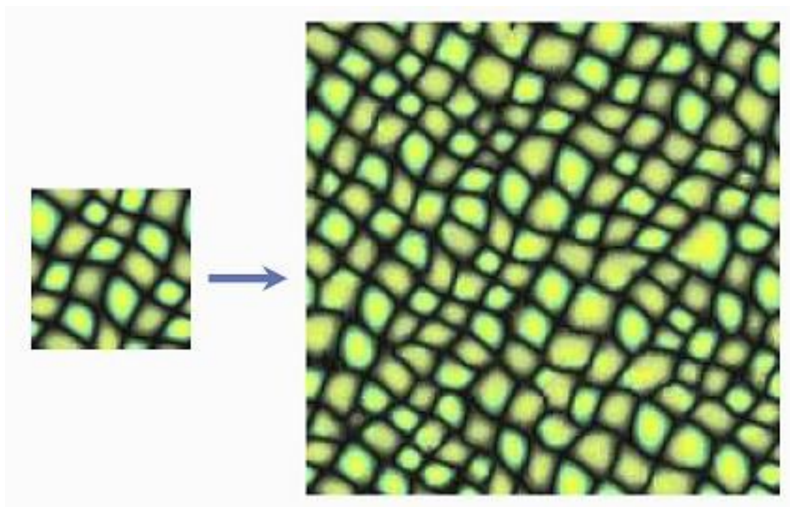
- From real data
  - Paint a self similar texture
  - use torus topology for textures





Creating a texture

- From real data
  - texture synthesis
  - analyse a small sample
  - generate a large similar texture

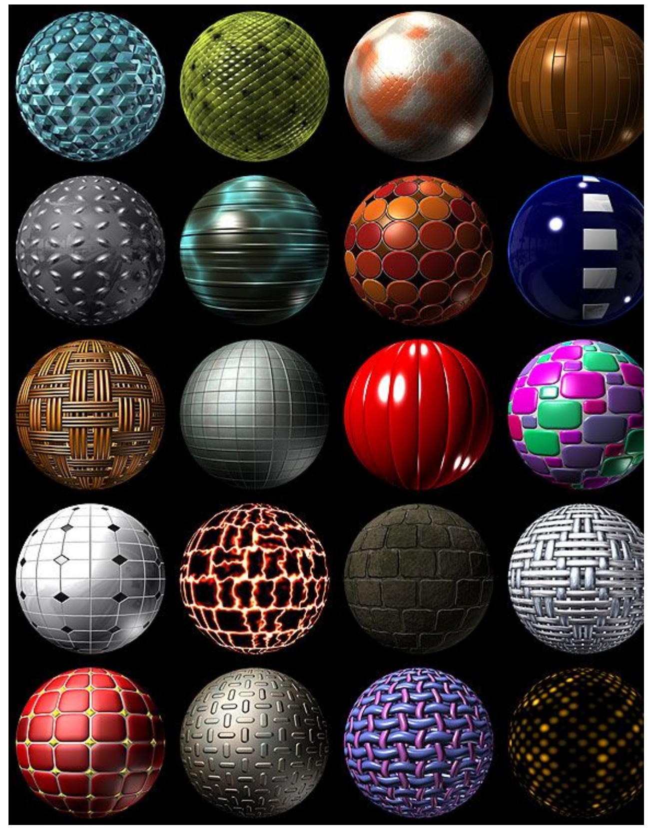


Creating a texture

- From real data
  - re-shading problem

Creating a texture

- Proceduraly
  - combination of simple functions

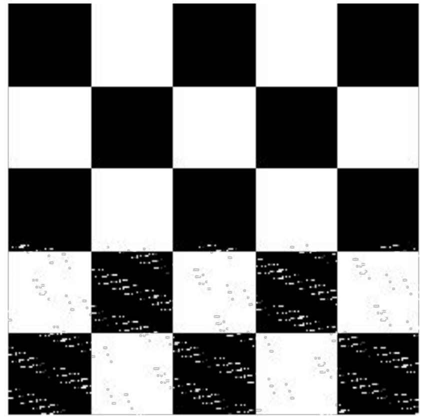


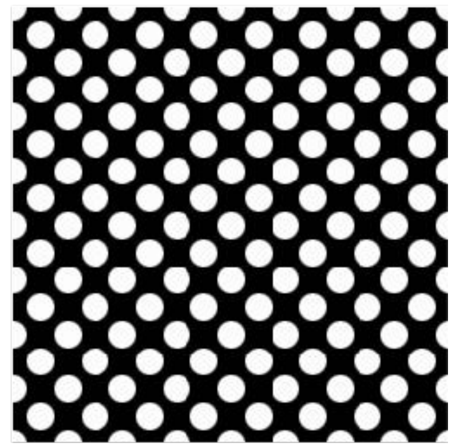
+ Easy to implement
+ Compact
+ Infinite resolution

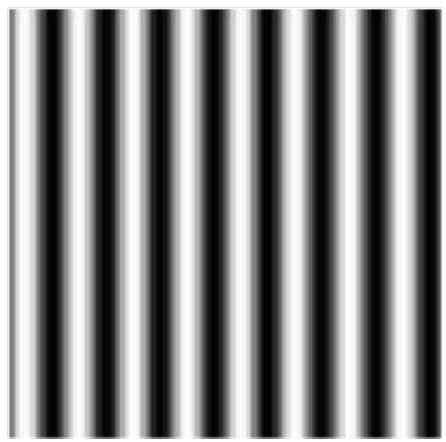
- Non-intuitive
- Difficult to match existing textures

Procedural textures

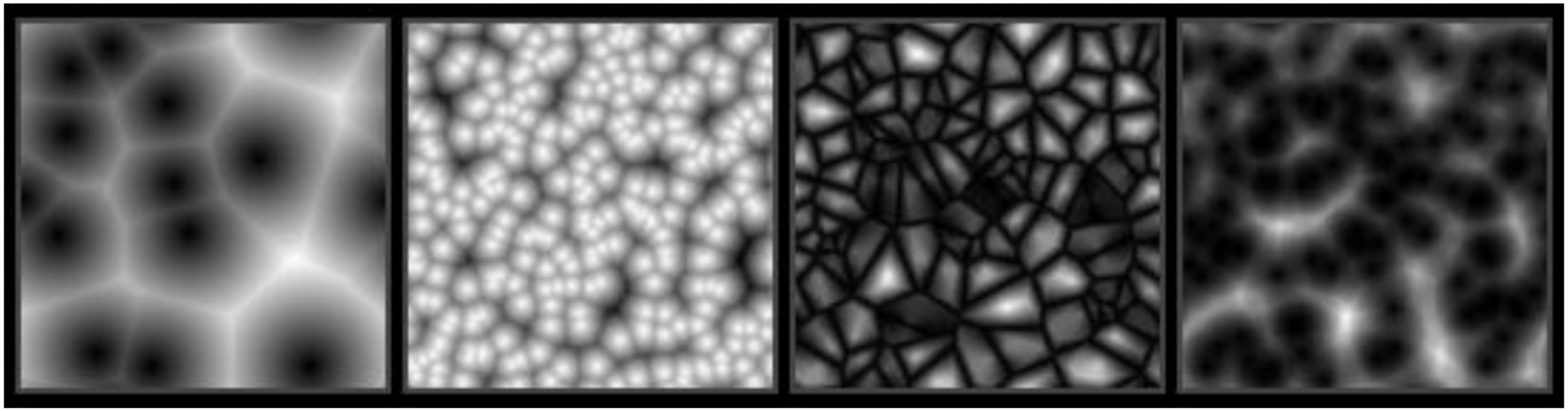
- Combination of simple functions
  - Mod
  - Clamp
  - Mix
  - Sin / cos / tan
  - Pow
  - Exp
  - Etc…







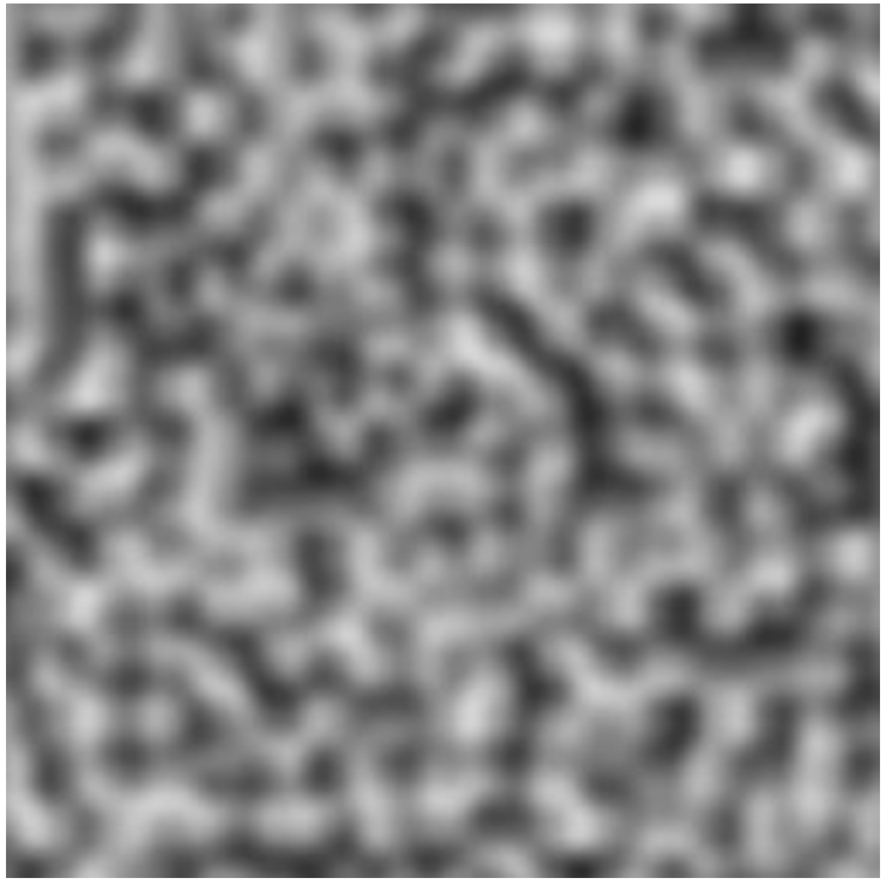
Procedural cellular textures



- Generate a bunch of random points
- For each pixel
  - Find the nearest distance to the nearest couple points
  - Use these values to determine a color
- Voronoi-like

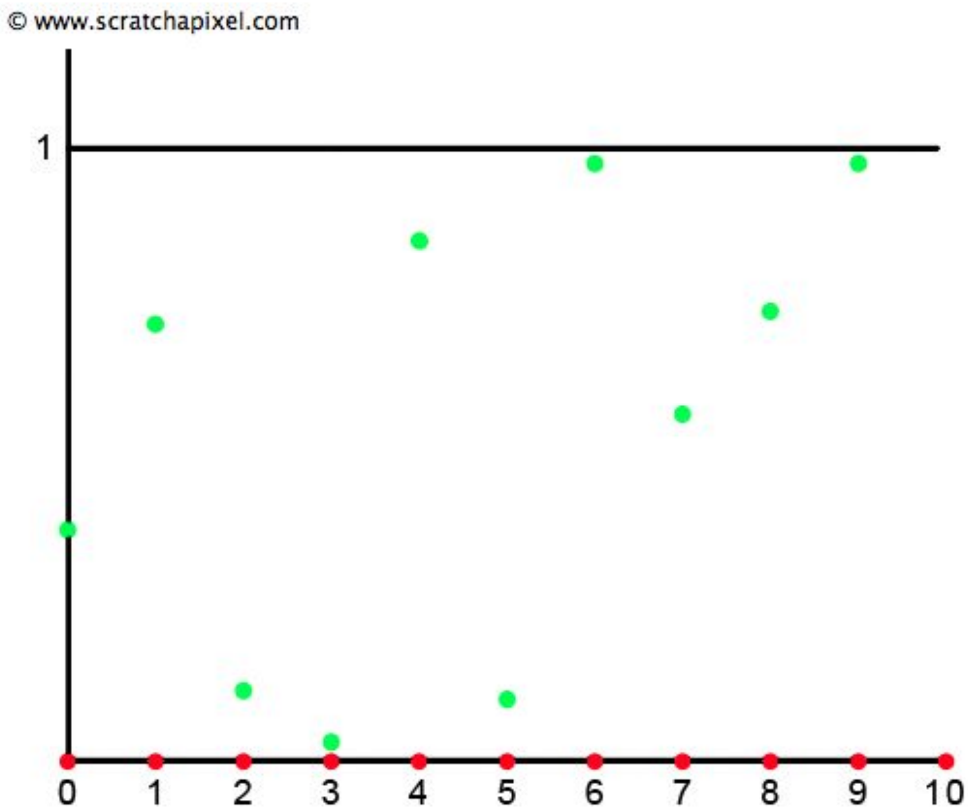
Perlin textures

- Requirements
  - Pseudo random
  - Arbitrary dimension
  - Smooth
  - Band pass (one scale)
  - Little memory usage
  - Implicit evaluation



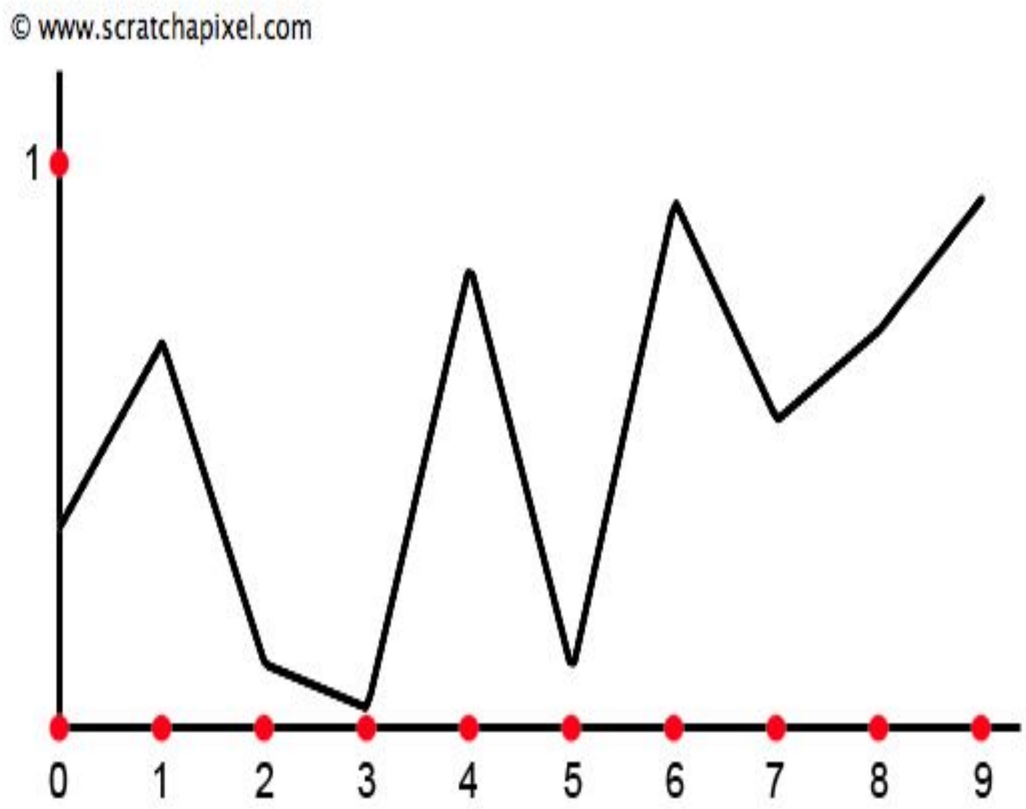
Perlin textures

- Distribute random values at particular locations (a grid)...



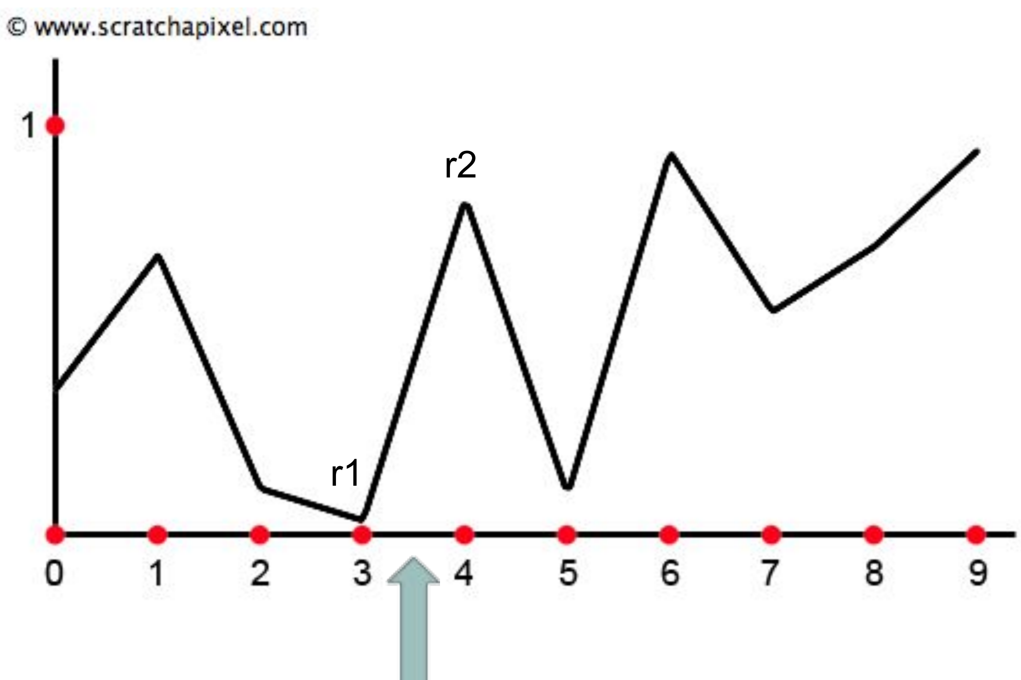
Perlin textures

- Distribute random values at particular locations (a grid)...
- … and interpolate



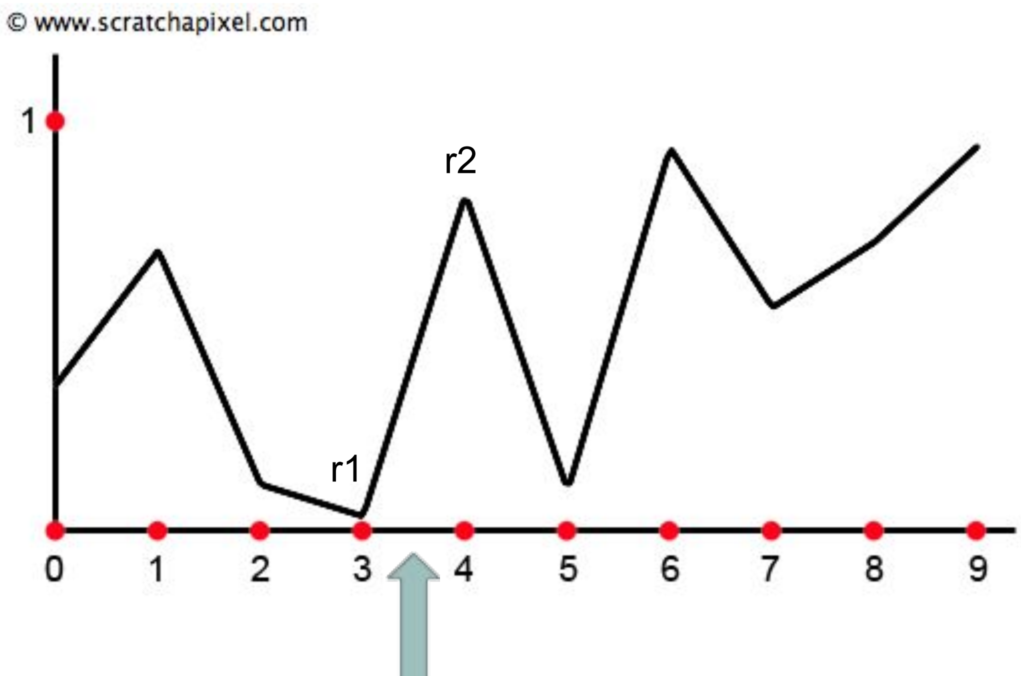
Perlin textures

- At a given point:



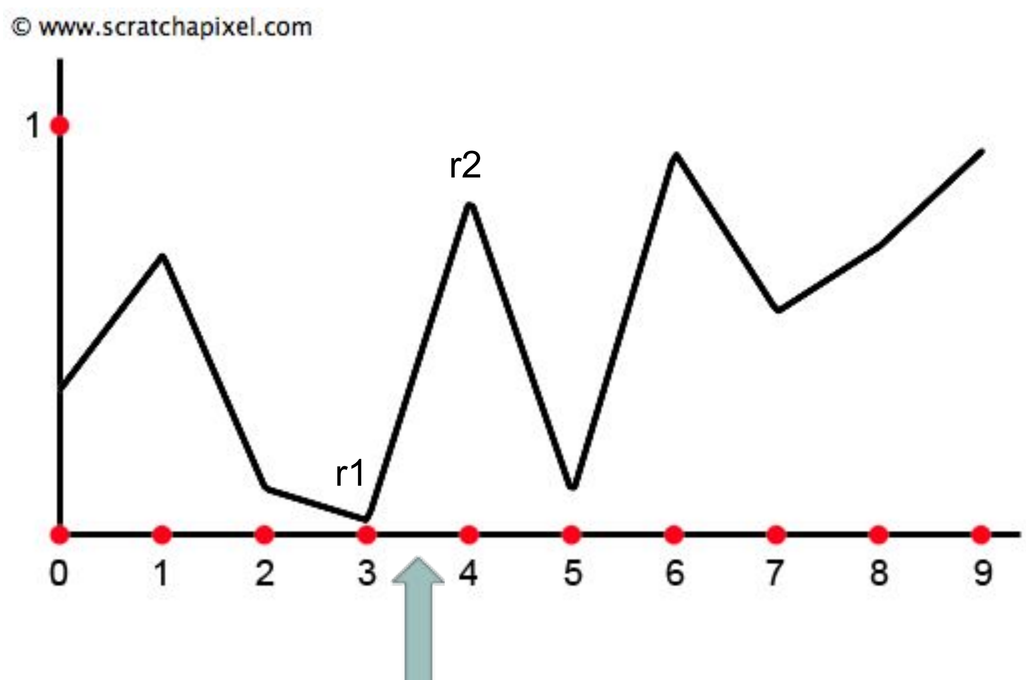
Perlin textures

- At a given point:
  - Get the associated 2 random values?



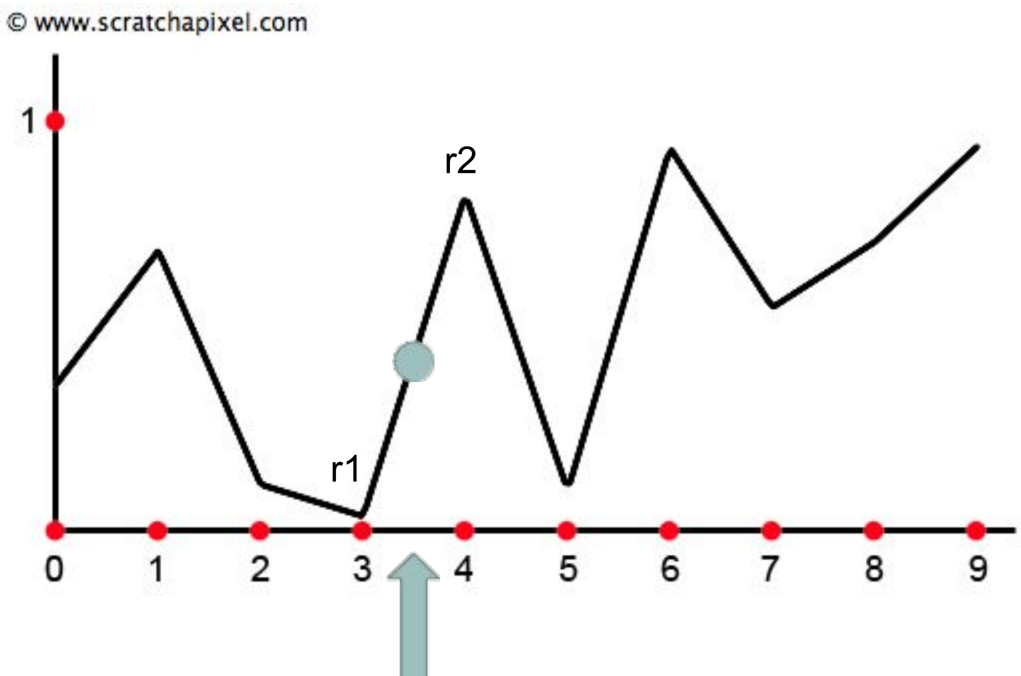
Perlin textures

- At a given point:
  - Get the associated 2 random values?
    - Pseudo random function
    - Precomputed in an array



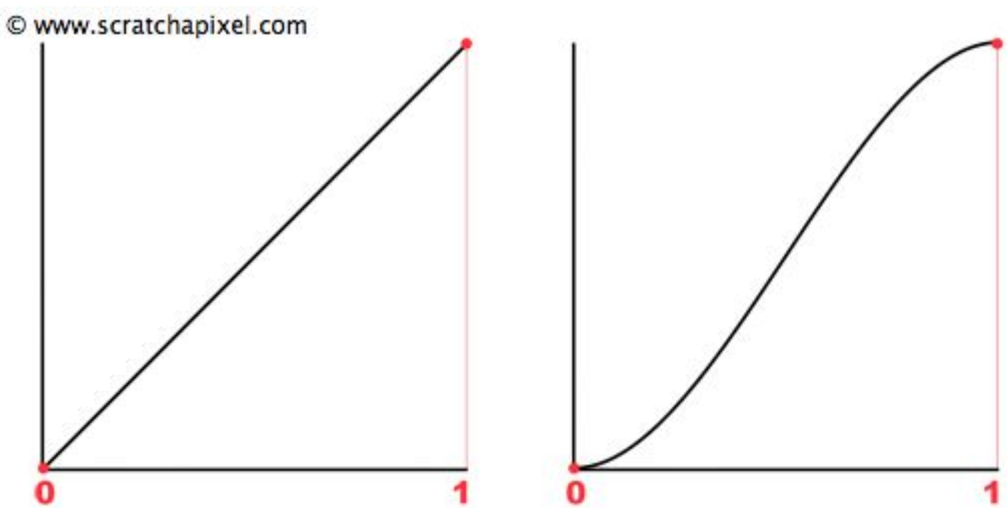
Perlin textures

- At a given point:
  - Get the associated 2 random values?
    - Pseudo random function
    - Precomputed in an array
  - Get relative position of x (between 0 and 1)
  - mix!



Perlin textures

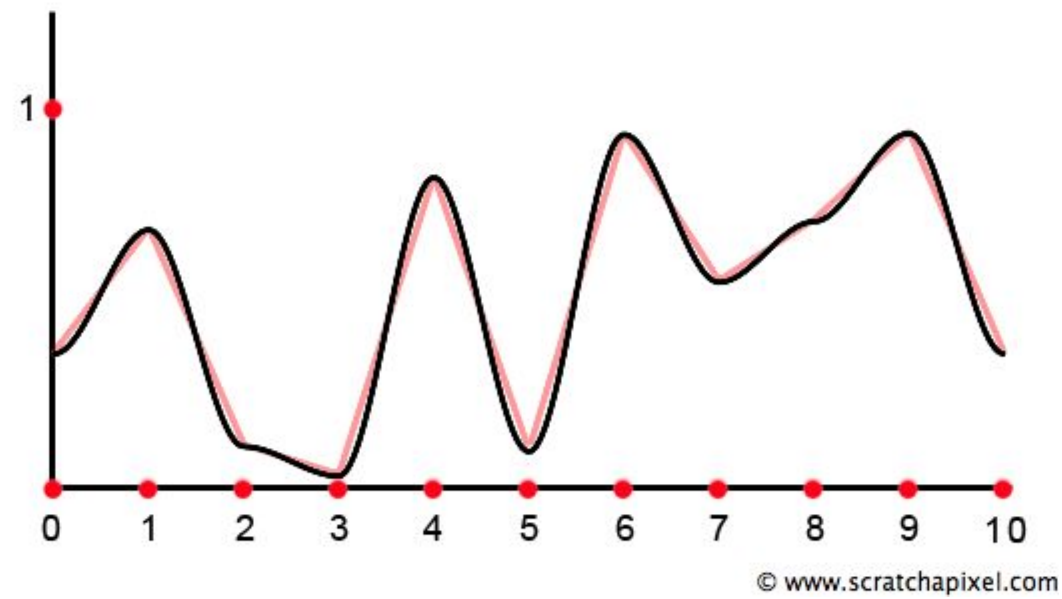
- At a given point:
  - Get the associated 2 random values?
    - Pseudo random function
    - Precomputed in an array
  - Get relative position of x (between 0 and 1)
  - mix!



S-Shaped function: $t = t^2(3 - 2t)$

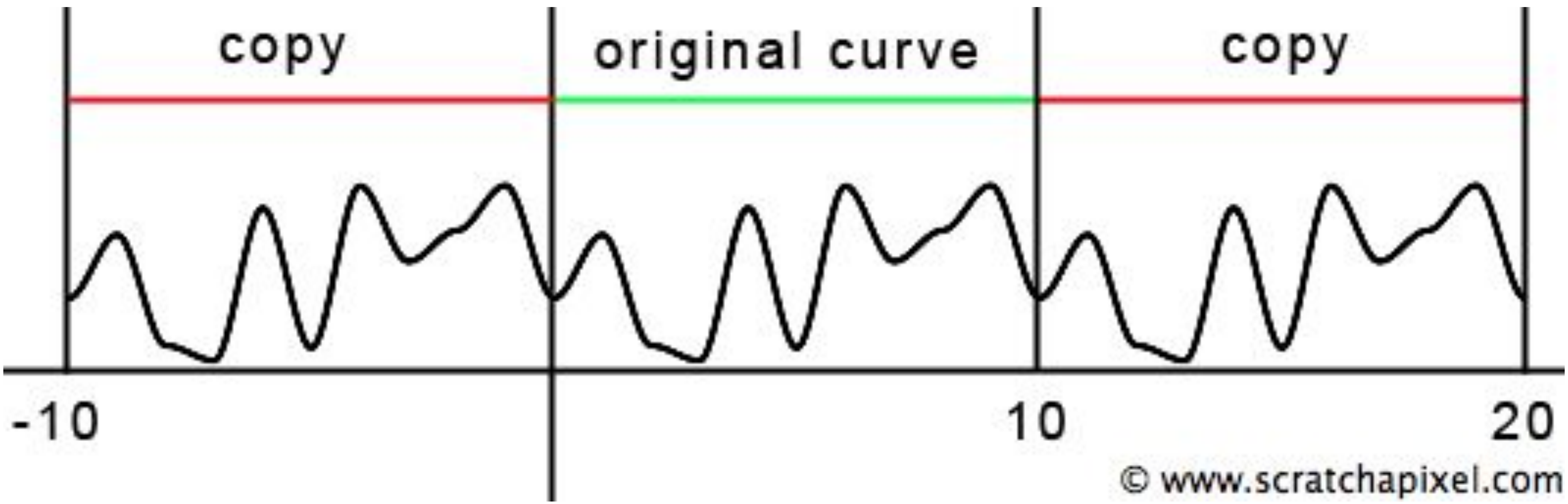
Perlin textures

- At a given point:
  - Get the associated 2 random values?
    - Pseudo random function
    - Precomputed in an array
  - Get relative position of x (between 0 and 1)
  - mix!



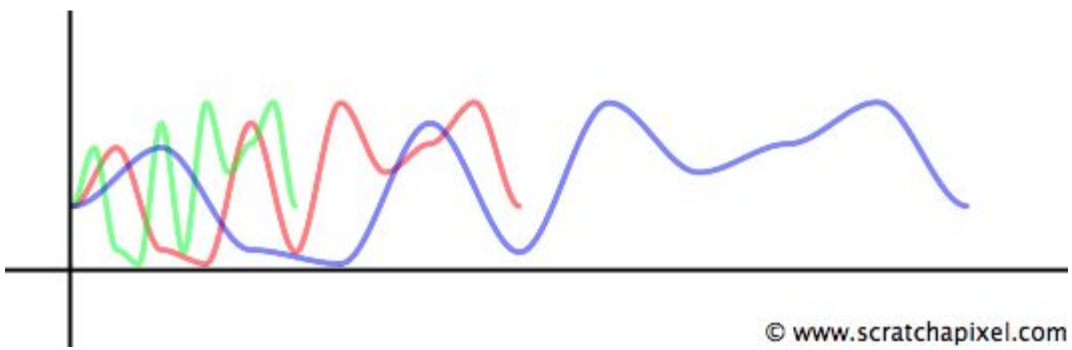
Perlin textures

- At a given point:
  - Get the associated 2 random values?
    - Pseudo random function
    - Precomputed in an array
  - Get relative position of x (between 0 and 1)
  - mix!

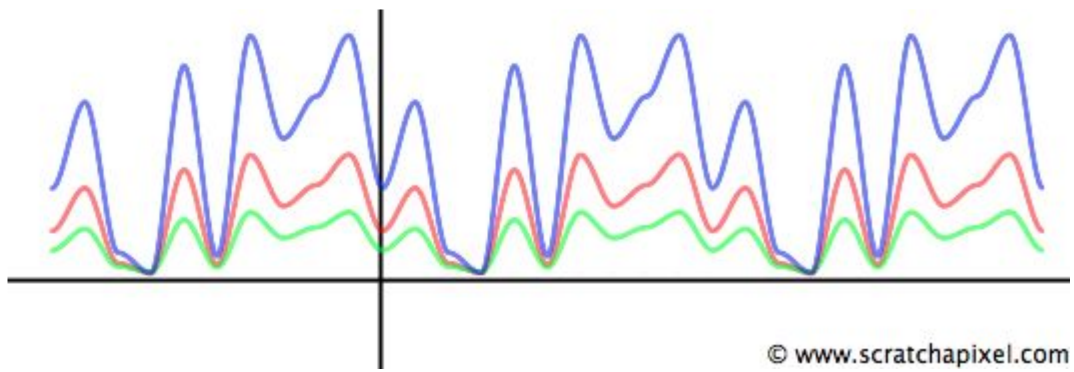


Perlin textures

- Controls
  - Frequency: evalNoise(x * freq)

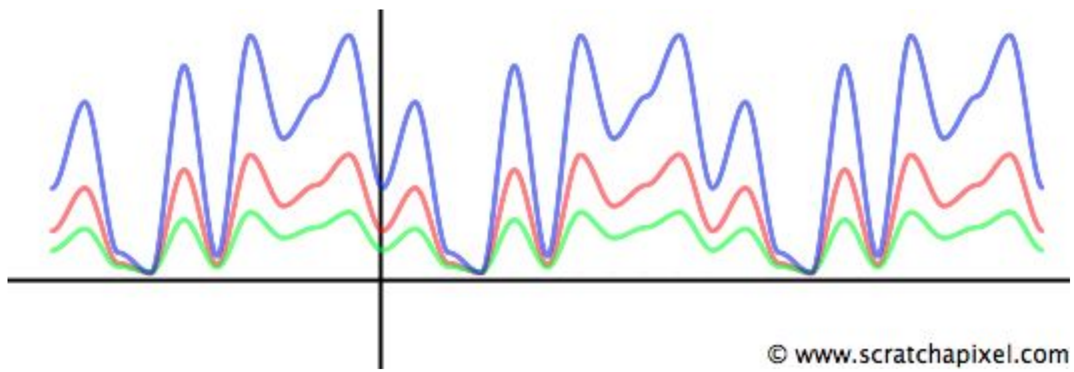


- Controls
  - Frequency: evalNoise(x * freq)
  - Amplitude: evalNoise(x) * amplitude



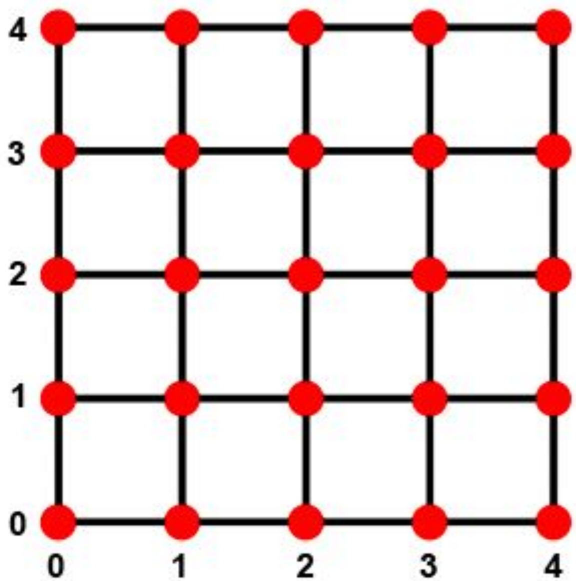
Perlin textures

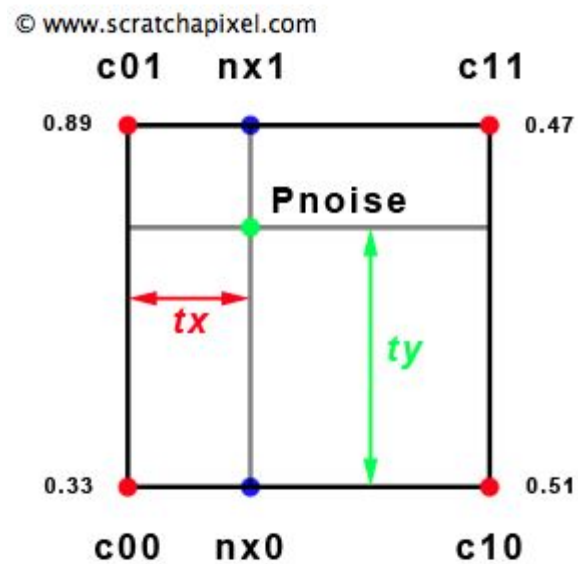
- Controls
  - Frequency: evalNoise(x * freq)
  - Amplitude: evalNoise(x) * amplitude
  - Offsetting: evalNoise(x + offset)



Perlin textures

- In 2D
  - needs a 2D grid
  - requires 3 interpolations instead of 1

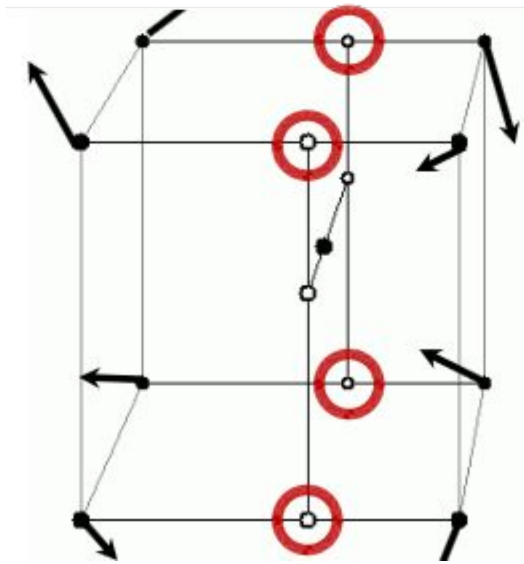




demo: https://www.shadertoy.com/view/lsf3WH

Perlin textures

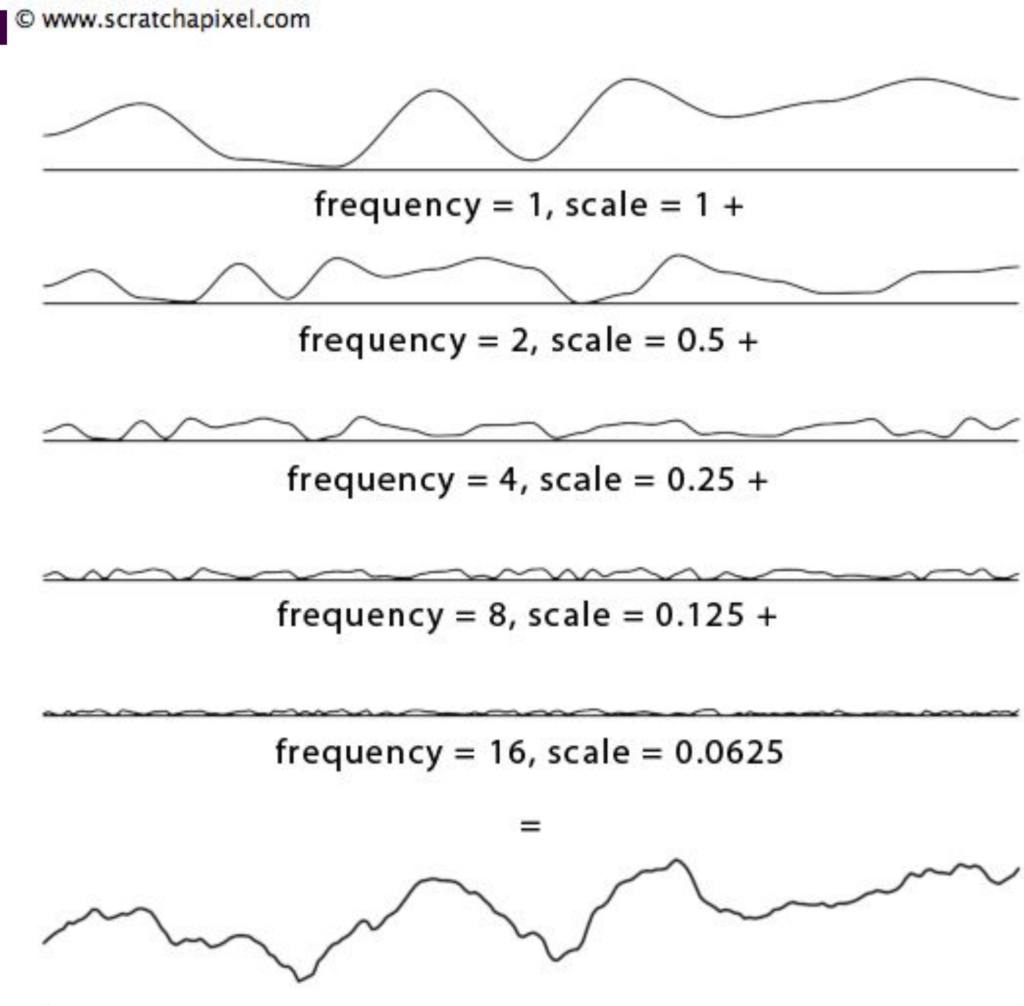
- In 3D
  - same principle, with a 3D grid
  - requires 7 interpolations



demo: https://www.shadertoy.com/view/XsXfRH

Fractal Perlin textures

- Noise at one scale = 1 octave
- Multiple octave usually used
  - Frequency multiplied by 2 each time
  - Hence the name octave
  - Different amplitudes too
- Sum of all noises =
  - Fractal noise

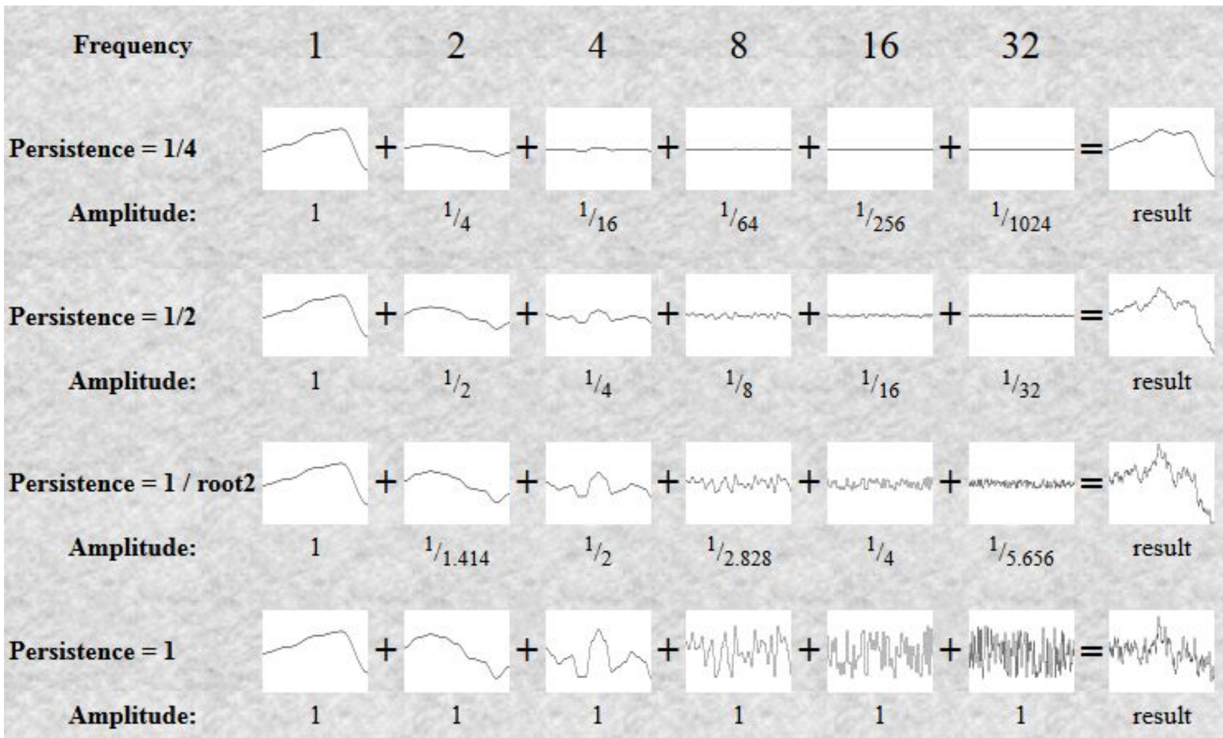


Fractal Perlin textures

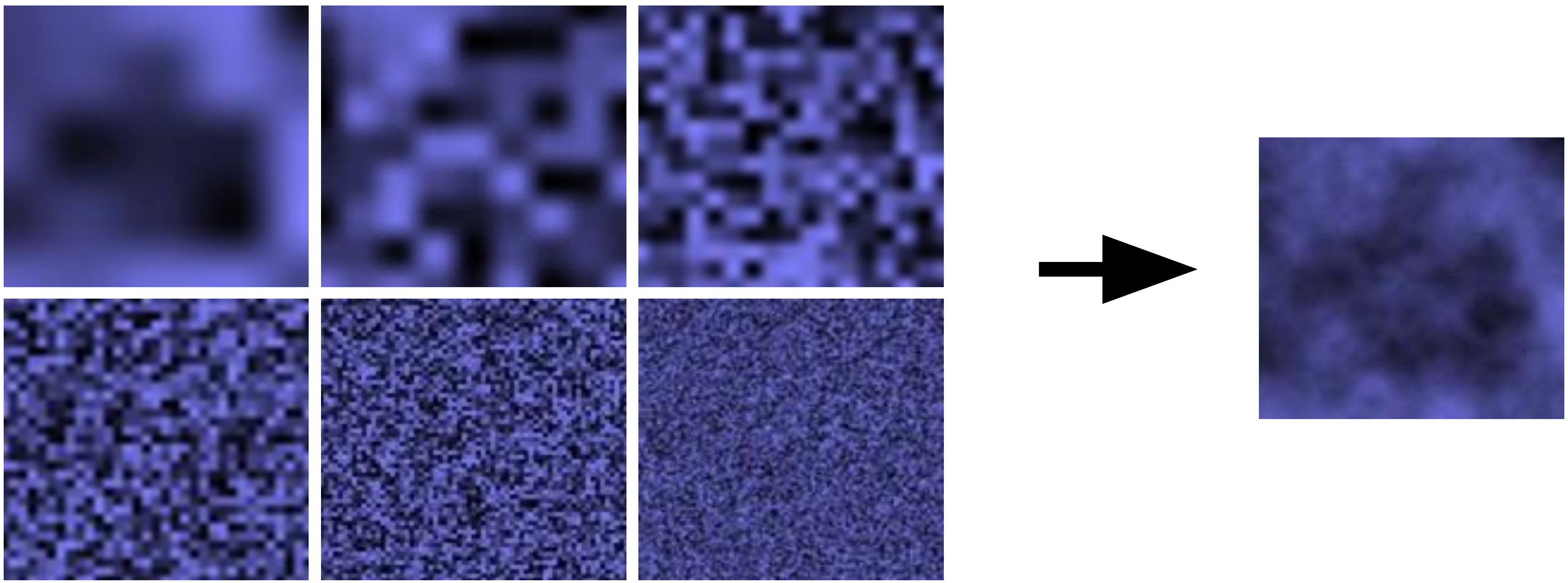
- Compute the ith texture using:

$$frequency = 2^i$$
$$amplitude = persistence^i$$

(relation between frequency and amplitude)

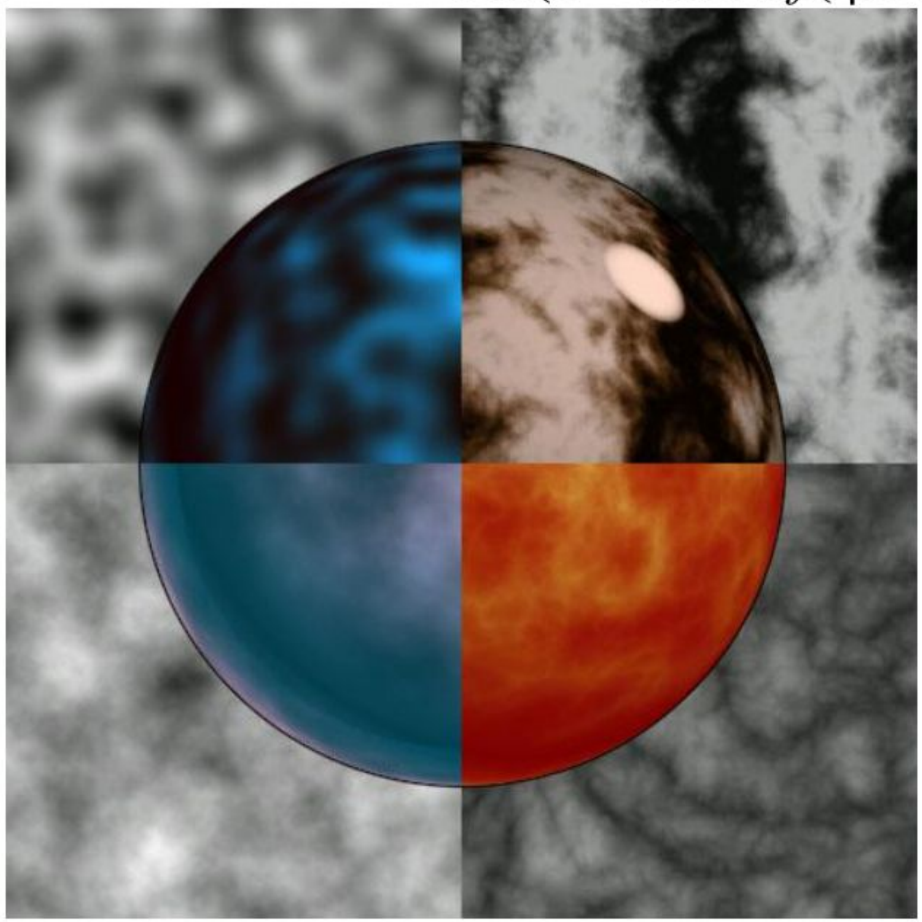


Fractal Perlin textures



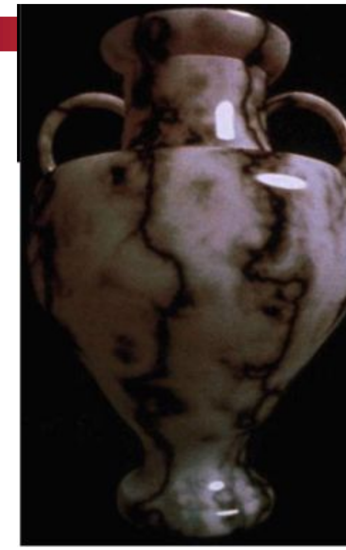
Fractal Perlin textures

•*noise* $\quad$ *sin*(*x* + sum 1/*f*(|*noise*|))



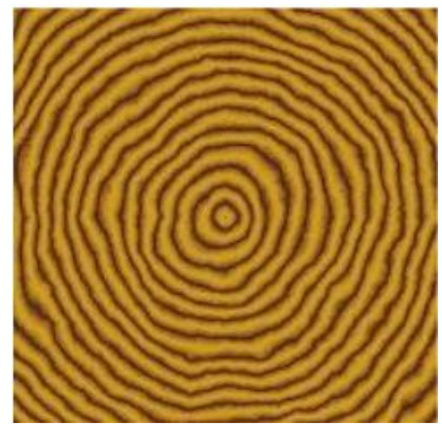
sum 1/*f*(*noise*) $\quad$ sum 1/*f*(|*noise*|)

Fractal Perlin textures



- Marble

Sin (x + Sum 1/f(noise))
=
Colormap(Sin (x + turbulence))



- Wood

Colormap(Sin (radius + turbulence))

Perlin's textures : Examples