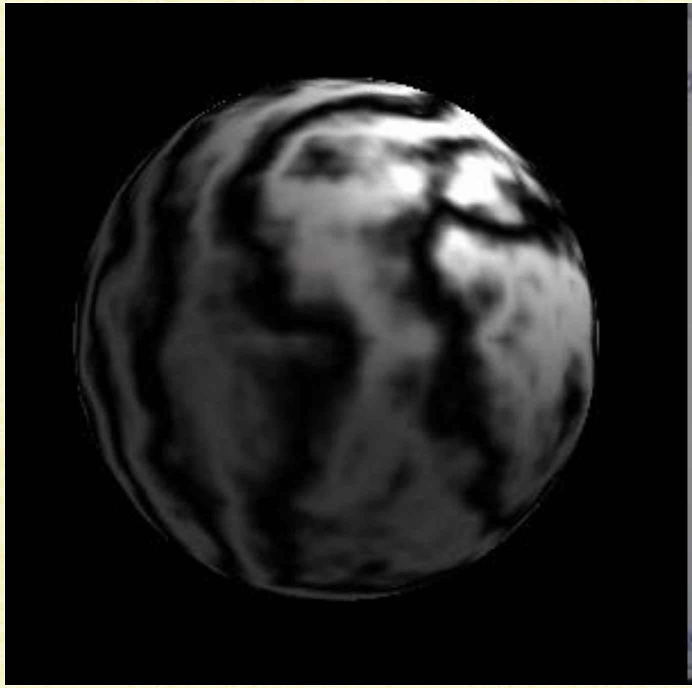
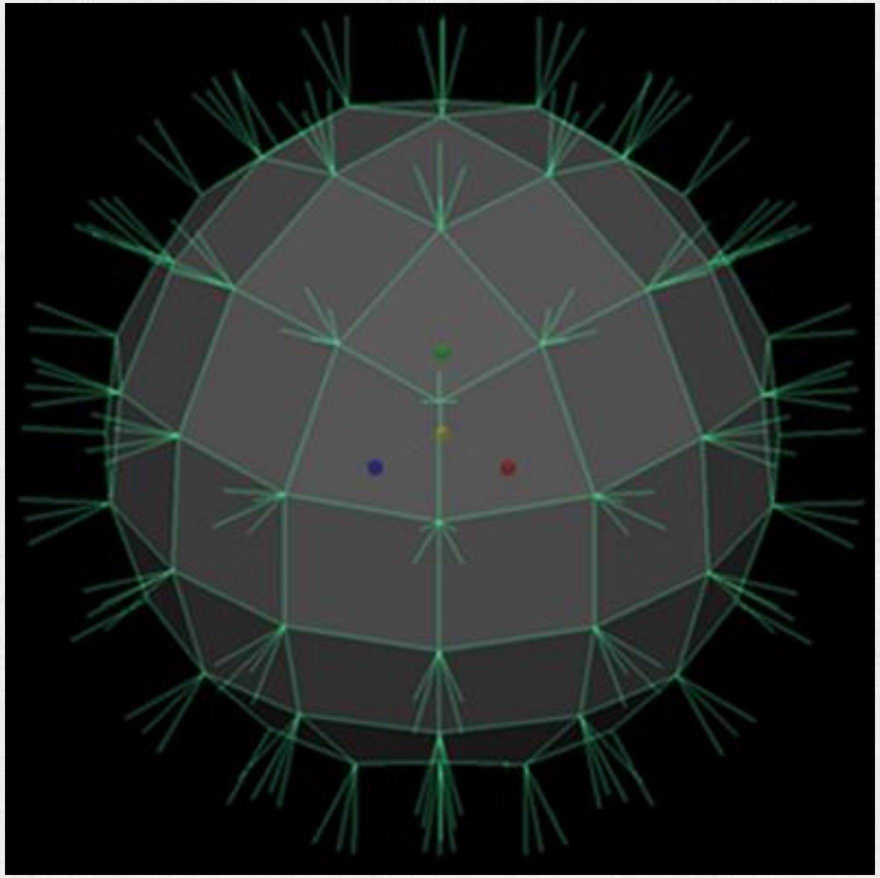
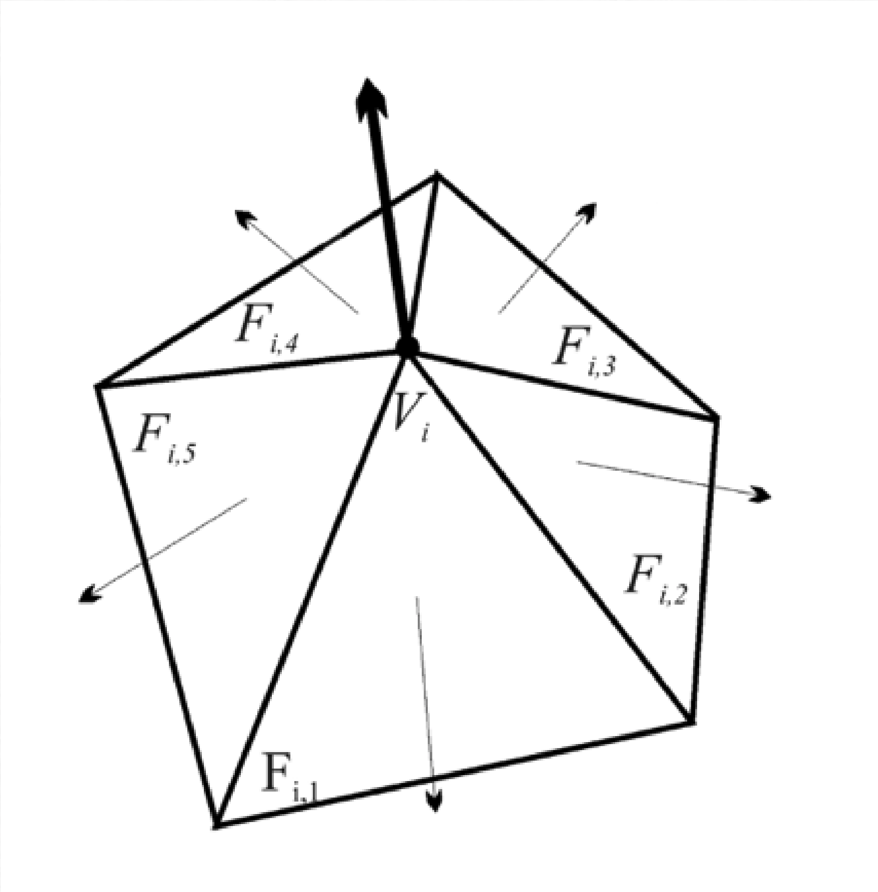
More Texture Mapping



Recall: Averaged Vertex Normals

- Each vertex belongs to a number of triangles, each with their own normal
- Averaging those normals (weighted averaging, based on: area, angle, etc.) gives a unique normal for each vertex

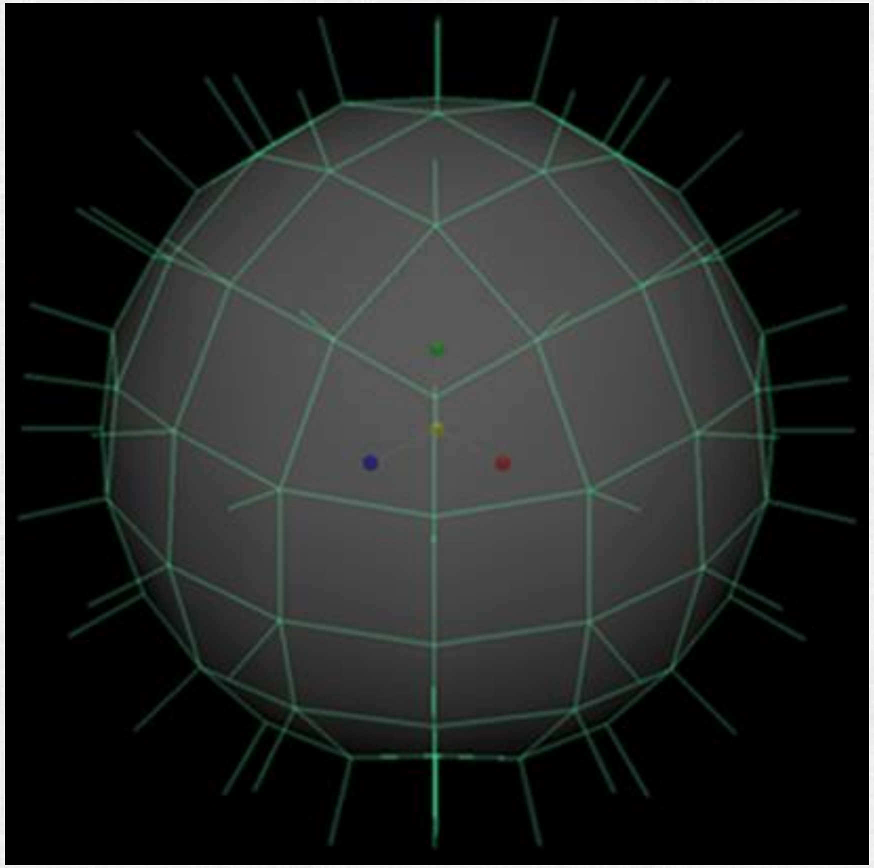


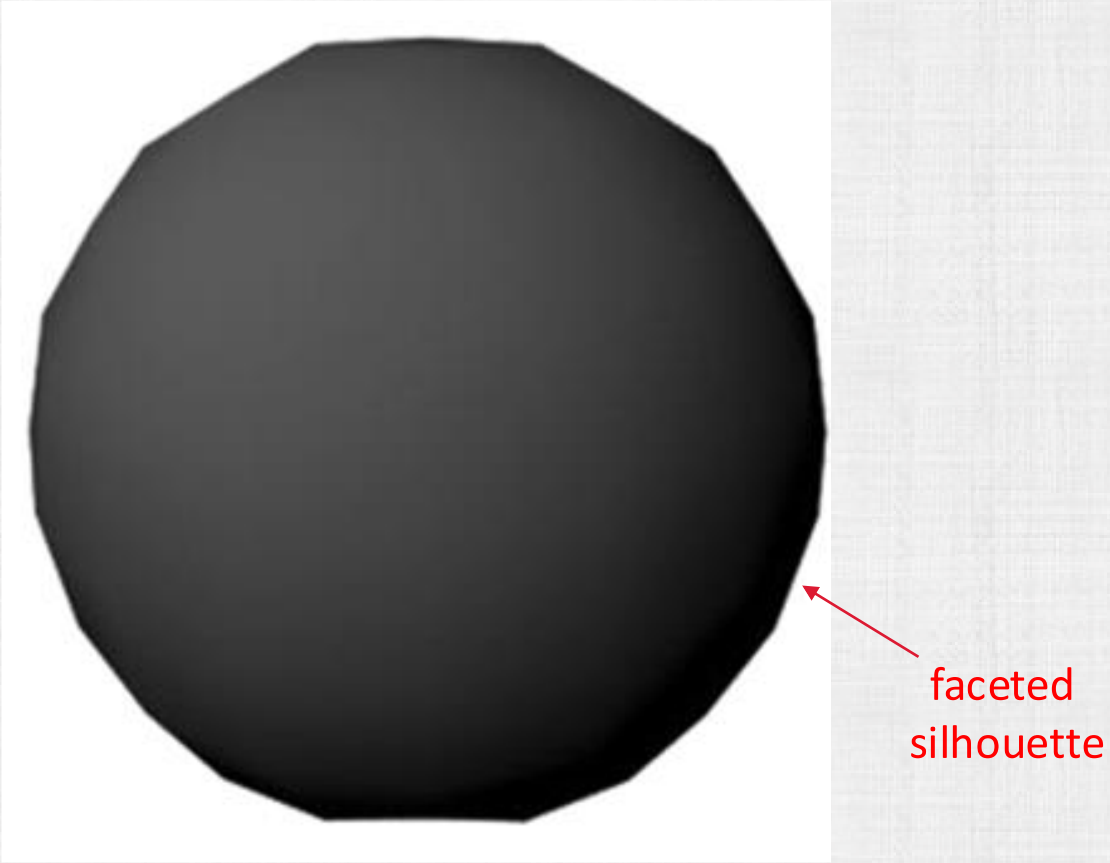


Recall: Smooth Shading

- Use barycentric weights to interpolate averaged vertex normals to the interior of a triangle:

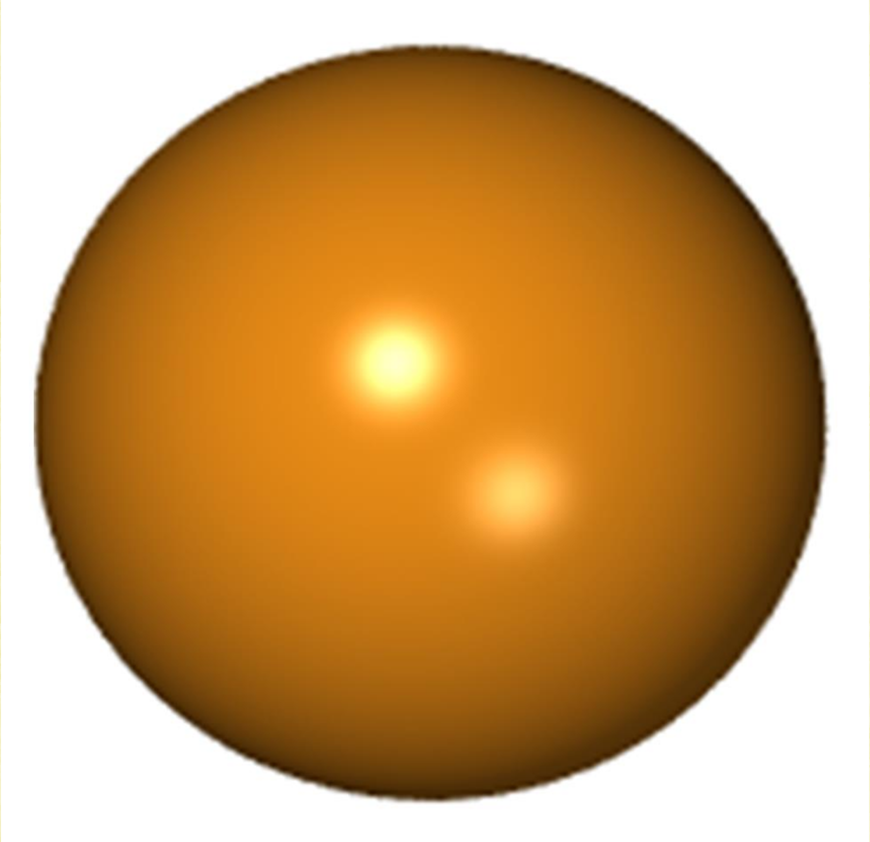
$$\hat{N}_p = \frac{\alpha_0 \hat{N}_0 + \alpha_1 \hat{N}_1 + \alpha_2 \hat{N}_2}{\left\| \alpha_0 \hat{N}_0 + \alpha_1 \hat{N}_1 + \alpha_2 \hat{N}_2 \right\|_2}$$



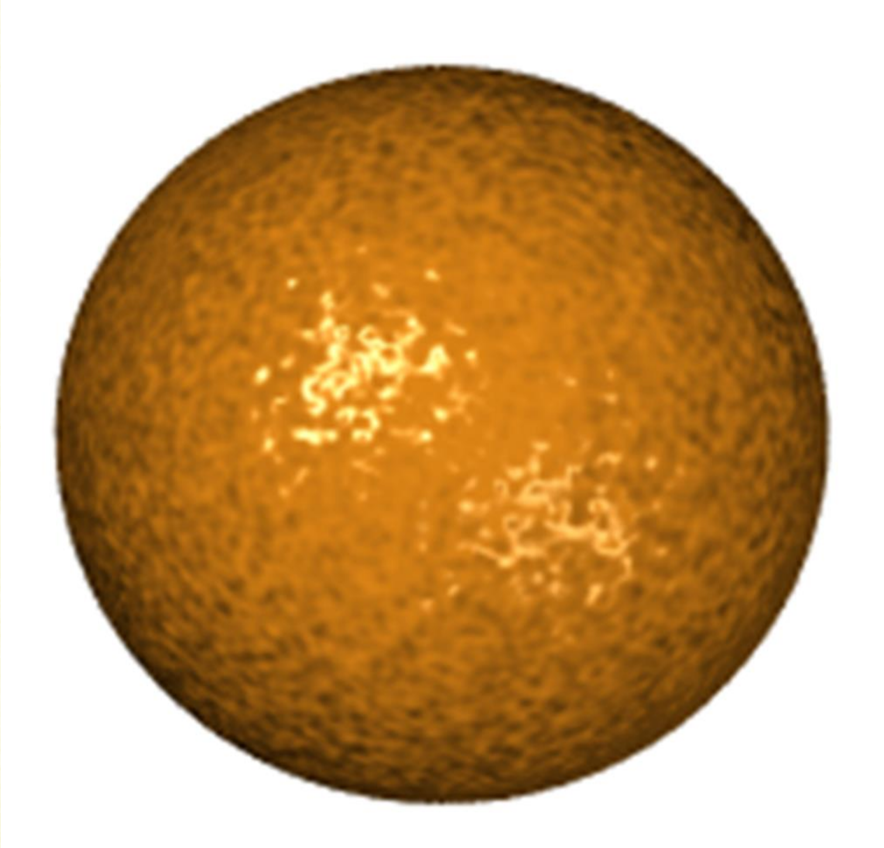


Perturbing the Normal

- Store a normal vector in the texture (instead of a color)
- A perturbed normal can be used to make "fake" geometric details



using real normal



using fake normal

Normal Map

- A normalized vector has each component in $[-1,1]$, so can convert back and forth to a color format via:

$$(R,G,B) = 255\frac{\vec{N}+(1,1,1)}{2} \quad \text{and} \quad \vec{N} = \frac{2}{255}(R,G,B) - (1,1,1)$$

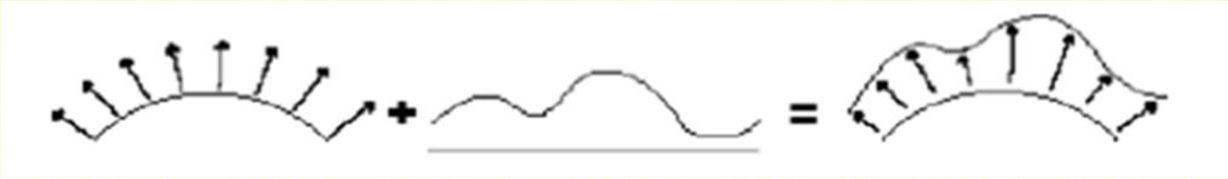
normal mapping on a plane
(note the variation in specular highlights created by variation of the normal)

Bump Map (variation of a Normal Map)

- Single-channel (grey-scale) height map h_{ij}, representing the height at location (u_i, v_j)
- The tangent plane at a point (u_i, v_j, h_{ij}) is: $-\frac{\partial h(u_i,v_j)}{\partial u}(u - u_i) - \frac{\partial h(u_i,v_j)}{\partial v}(v - v_j) + (h - h_{ij}) = 0$
- So, the outward pointing (non-unit) normal is: $\left(-\frac{\partial h(u_i,v_j)}{\partial u}, -\frac{\partial h(u_i,v_j)}{\partial v}, 1\right)$
    - Compute the partial derivatives via finite differences: $\frac{\partial h(u_i,v_j)}{\partial u} = \frac{h_{i+1,j} - h_{i-1,j}}{u_{i+1} - u_{i-1}}$ and $\frac{\partial h(u_i,v_j)}{\partial v} = \frac{h_{i,j+1} - h_{i,j-1}}{v_{j+1} - v_{j-1}}$

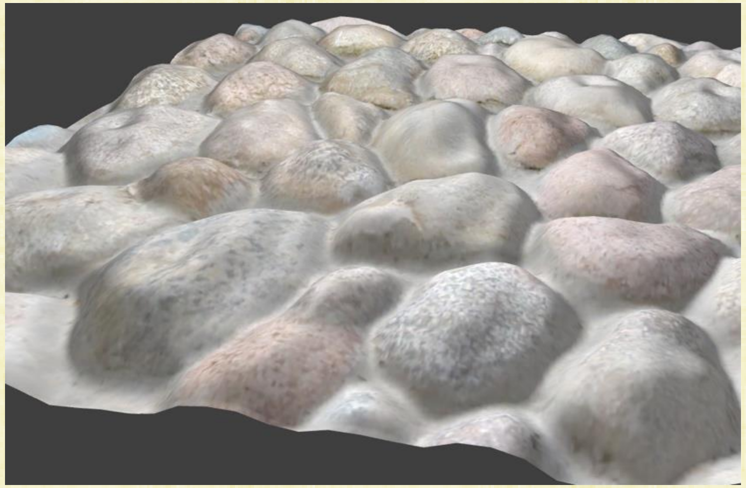
Displacement Mapping

- Subdivide geometry at render time, and use a height map $h(u, v)$ to perturb vertices in the normal direction
- Pros: self-occlusion, self-shadowing, detailed silhouettes
- Cons: expensive, requires adaptive tessellation, still need bump/normal map for sub-triangle detail



original geometry $\boldsymbol{h(u, v)}$ displacement map

bump map



displacement map

Displacement Mapping

bump map

displacement map

Inferencing Displacement Maps

- Train a neural network to inference a texture (based on the shape of a skeleton)
- Use the texture as a displacement map to drive cloth deformations

inferenced texture image

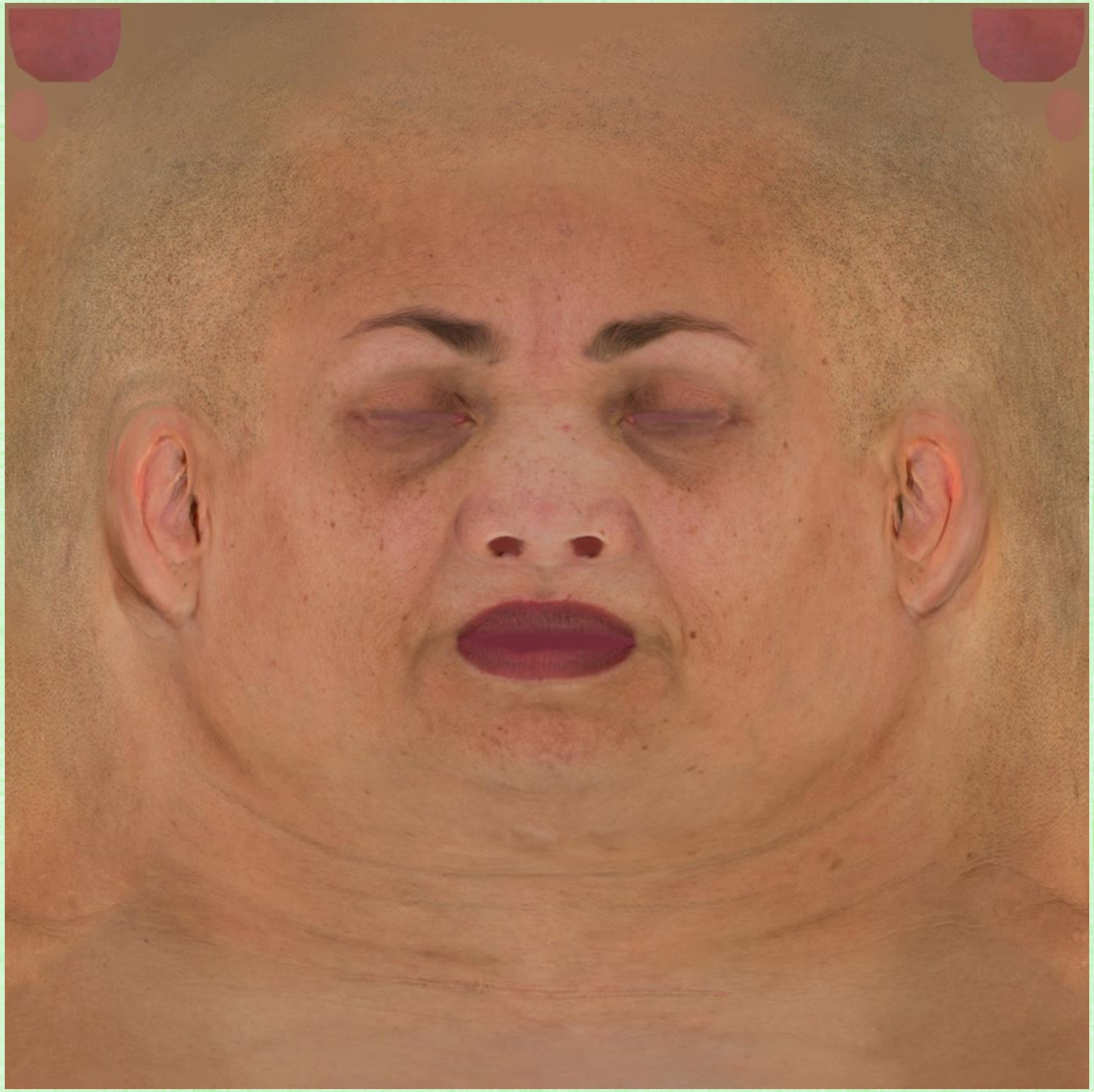
vertices look up colors

displace vertices to get folding & wrinkling

Texture Acquisition via Imaging

Light Stage

Texture Acquisition via Imaging



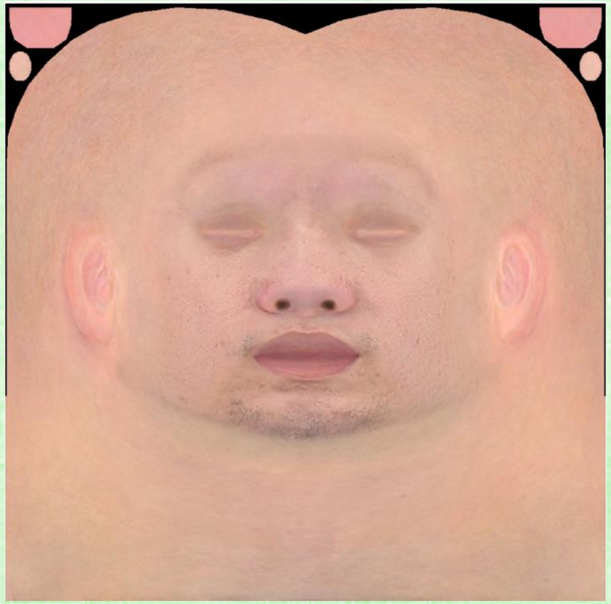
Face Texture

A Democratized Approach using Gaussian Splatting

cell phone images

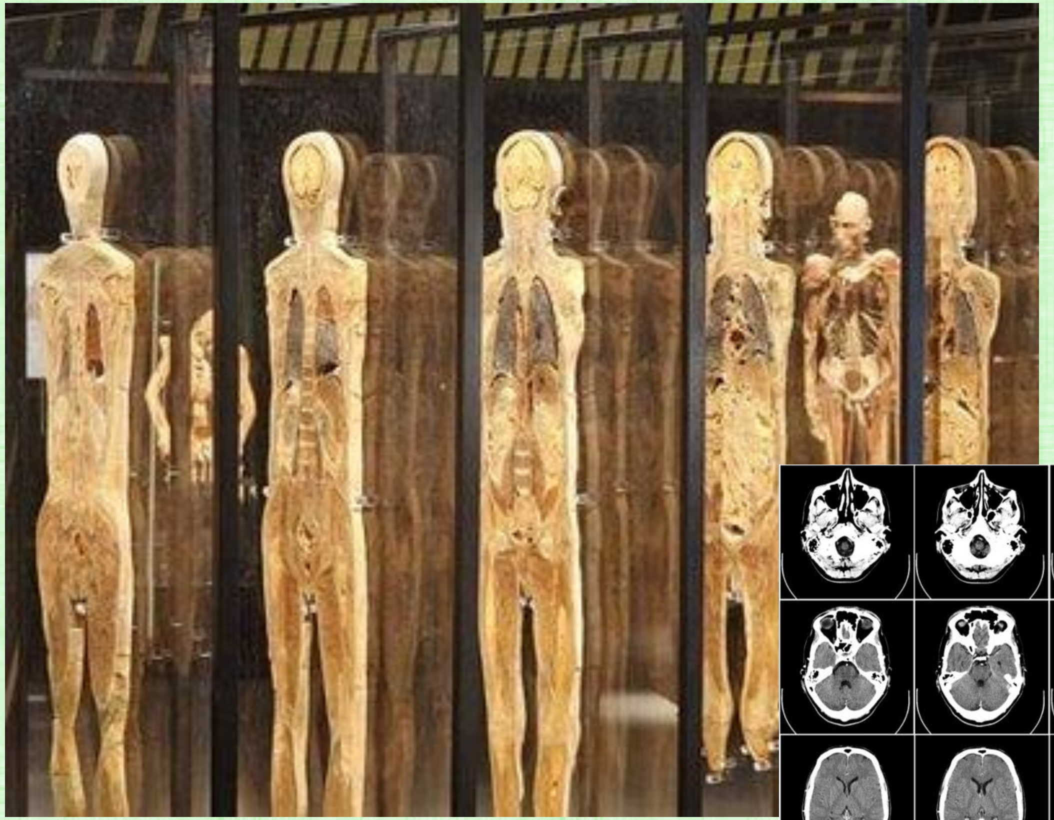
de-lit images
via portrait editing

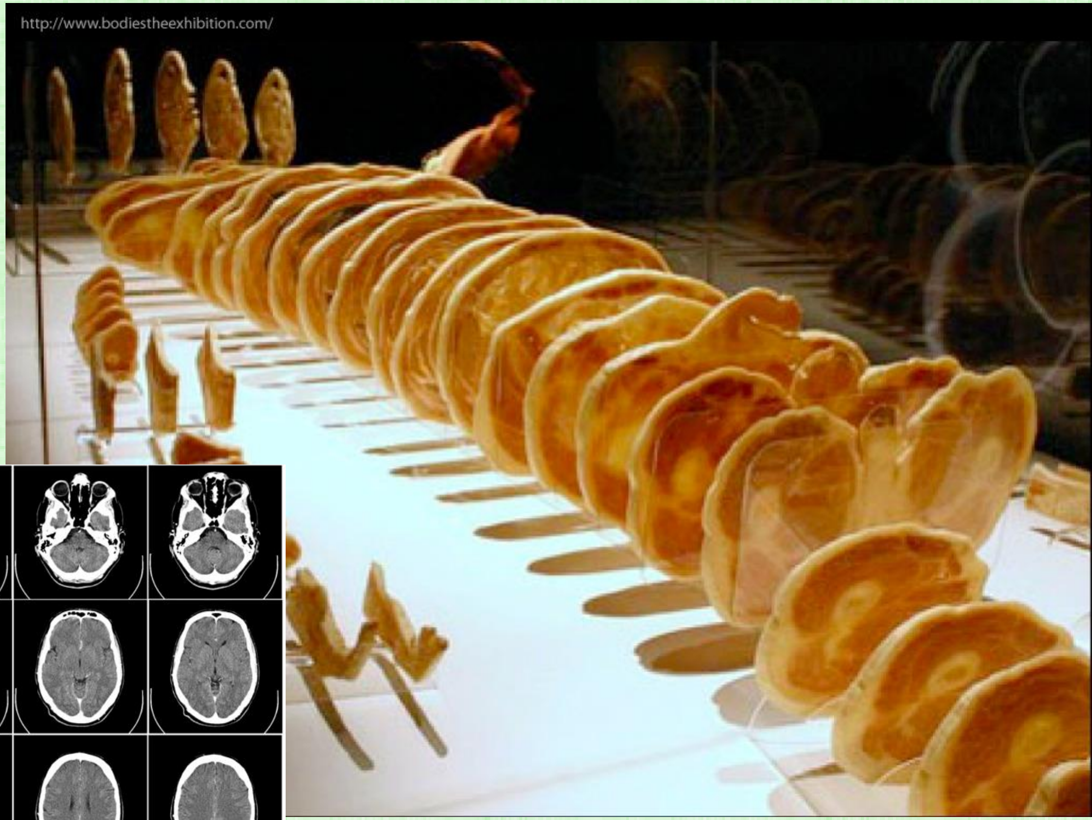
de-lit texture

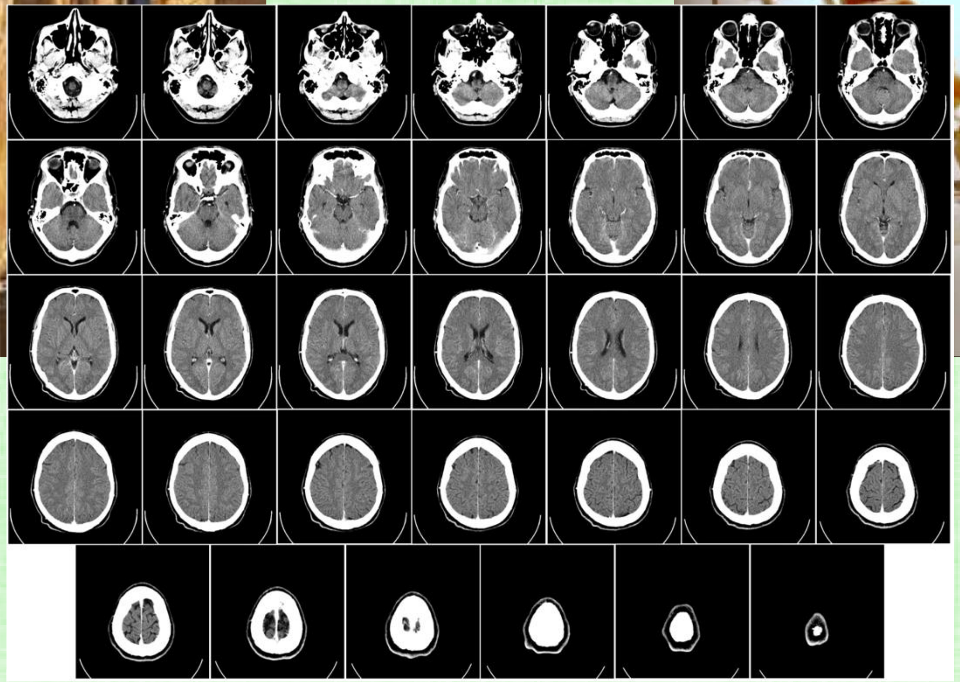


in-painted texture
(removing hair/eyebrows, etc.)

Texture Acquisition via Medical Imaging

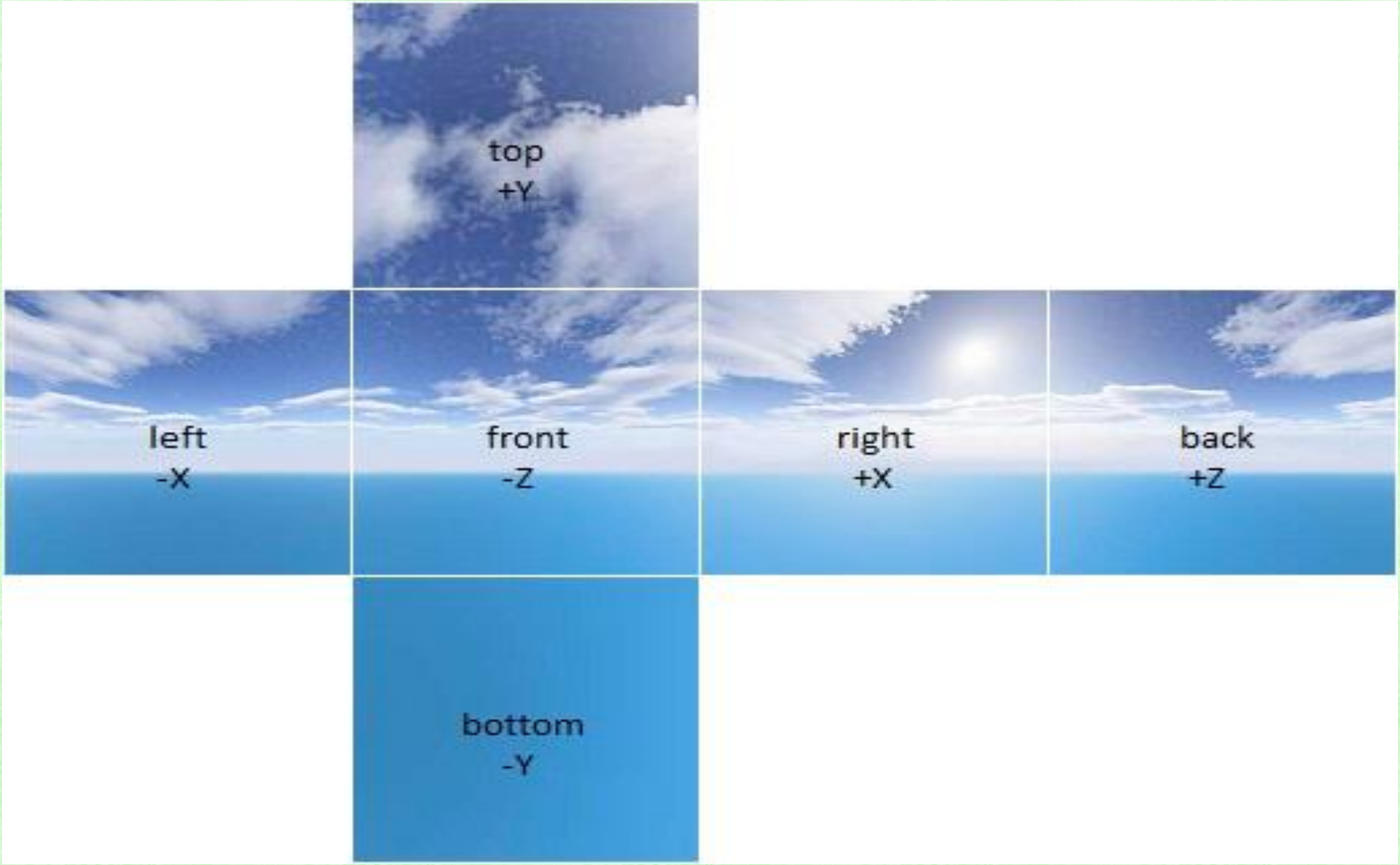






Sky Boxes

- Take pictures of the sky, and use them as textures on the inside of a cube (or hemisphere, etc.)



Recall: Measuring Incoming Light

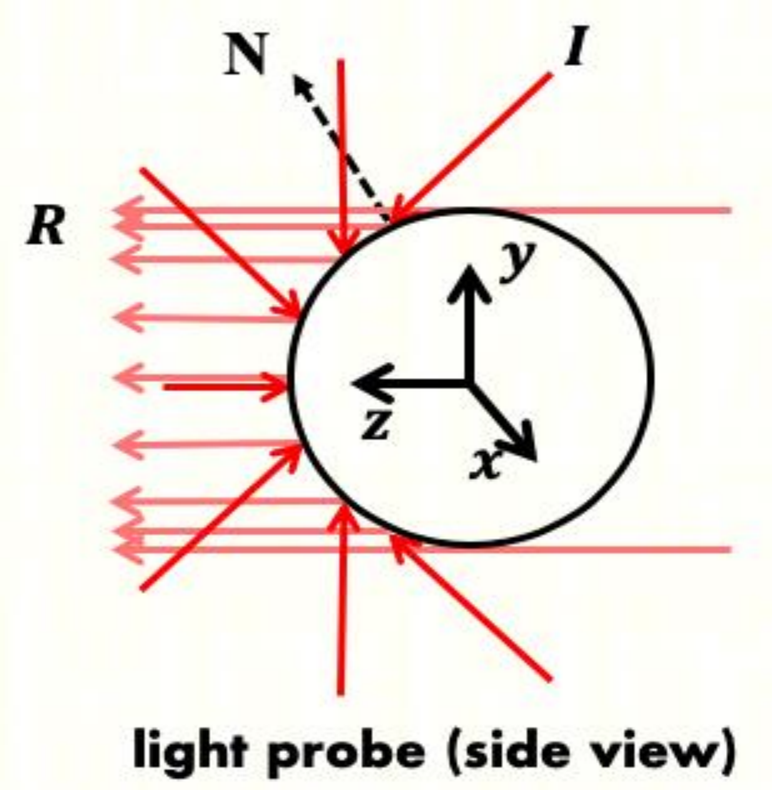
- Light Probe: a small reflective chrome sphere
- Photograph it, in order to record the incoming light (at its location) from all directions

Recall: Using the Measured Incoming Light

- The measured incoming light can be used to render a synthetic object with realistic lighting

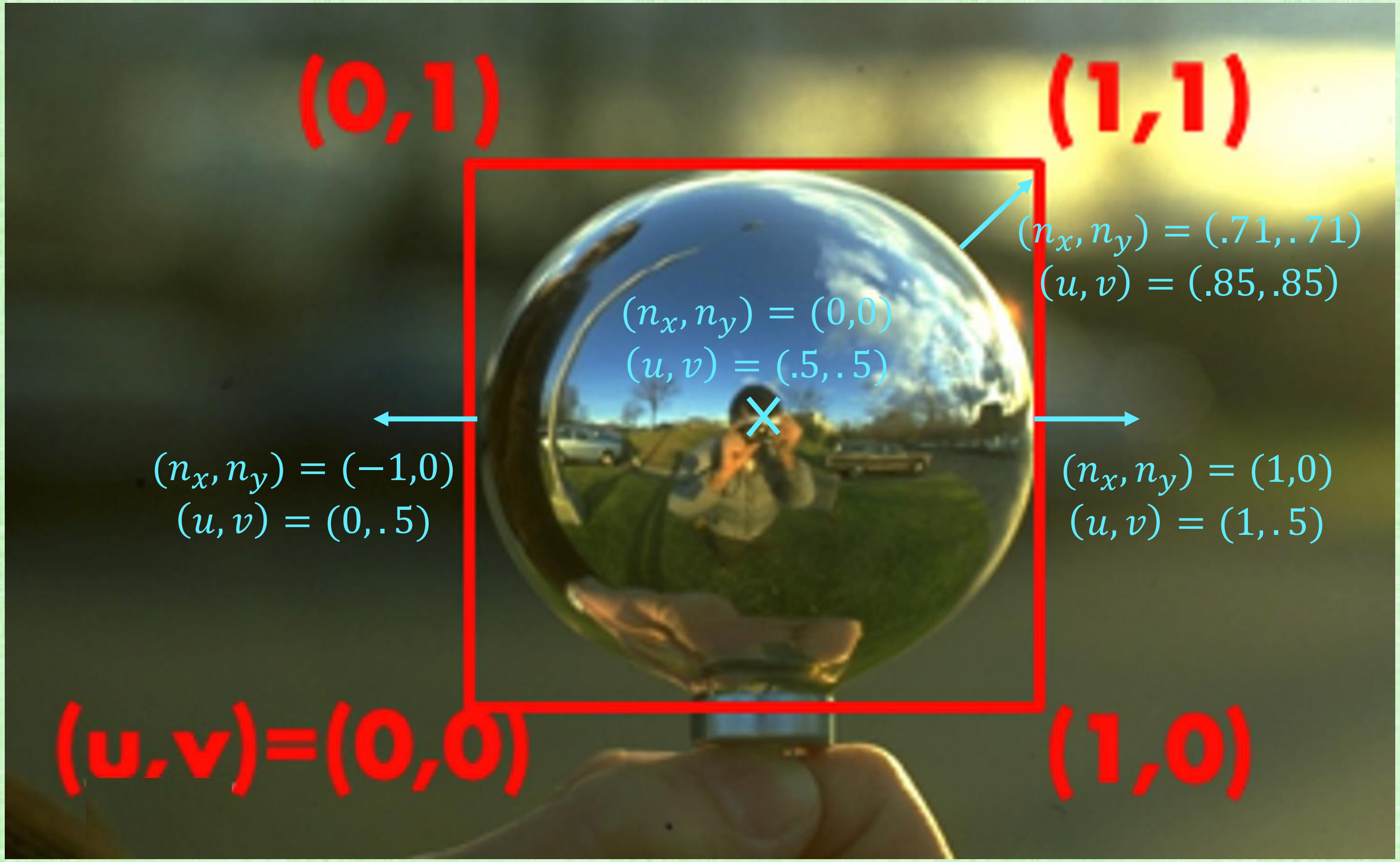
Environment Mapping

- Place a coordinate system at the center of the sphere; then, the surface normal is: $\boldsymbol{N} = \frac{(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})}{\sqrt{\boldsymbol{x}^2+\boldsymbol{y}^2+\boldsymbol{z}^2}}$
- Let $\boldsymbol{R}$ be the direction from the light probe to the camera
- $\boldsymbol{I}$ and $\boldsymbol{R}$ are equal-angle from $\boldsymbol{N}$ (because of mirror reflection)
- Thus, $\boldsymbol{N}$ has a one-to-one correspondence with $\boldsymbol{I}$ (i.e., $\boldsymbol{N}$ encodes the incoming light direction)



Environment Mapping

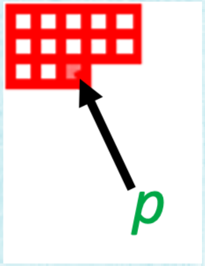
- Compute the normal $(\boldsymbol{n_x}, \boldsymbol{n_y}, \boldsymbol{n_z})$ at the point on the geometry that is being rendered
- Use $\boldsymbol{n_x}$ and $\boldsymbol{n_y}$ (in the interval [-1, 1]) to obtain texture coordinates $(\boldsymbol{u}, \boldsymbol{v}) = \frac{1}{2}(\boldsymbol{n_x} + \mathbf{1}, \boldsymbol{n_y} + \mathbf{1})$
- Then, the incoming light can be "looked up" in the texture (which is a picture of the chrome sphere)



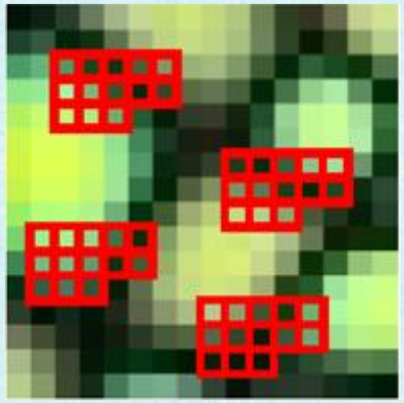
Environment Mapping

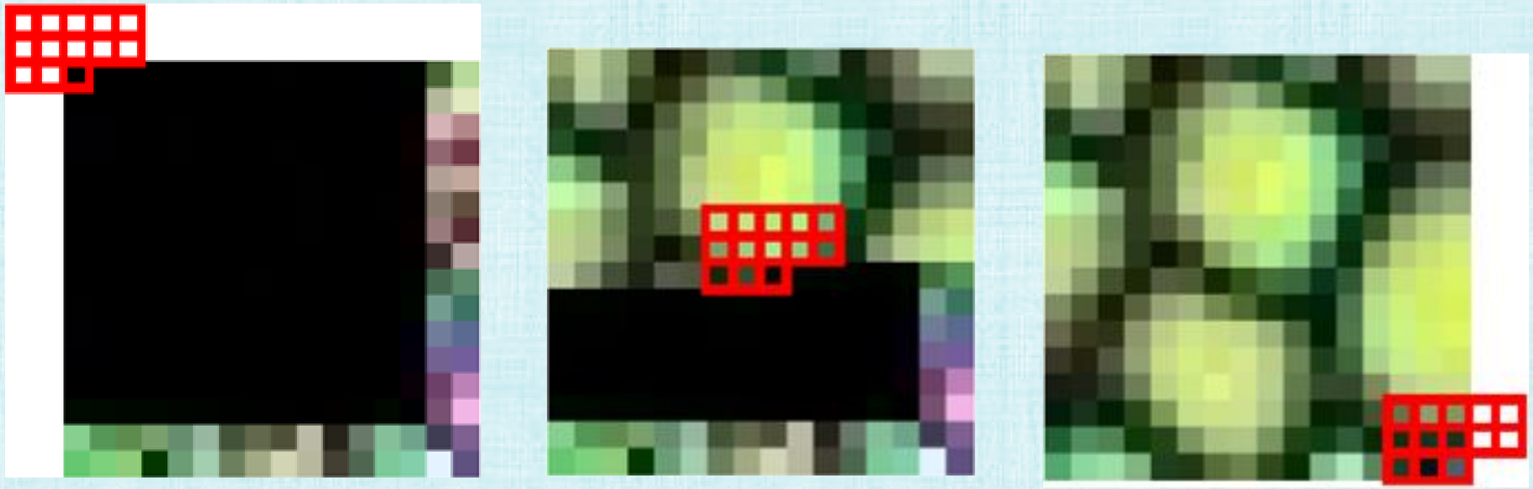
Texture Synthesis: Pixel Based

- Create a larger texture (one pixel at a time) from a small sample (using its structural content)
- Generate the texture in a raster scan ordering
- To generate the texture for pixel *p*:
  - Compare *p*'s neighboring pixels in the (red) stencil to all potential choices in the sample
  - Choose the stencil position with the smallest difference to fill pixel *p*
- When the stencil needs values outside the domain, use periodic boundaries (so, fill the last few rows/columns with random values)



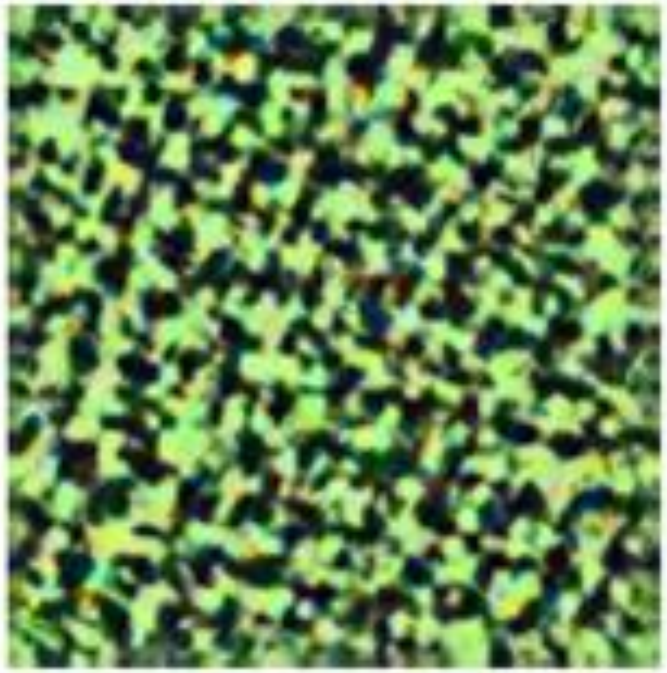
stencil


texture sample

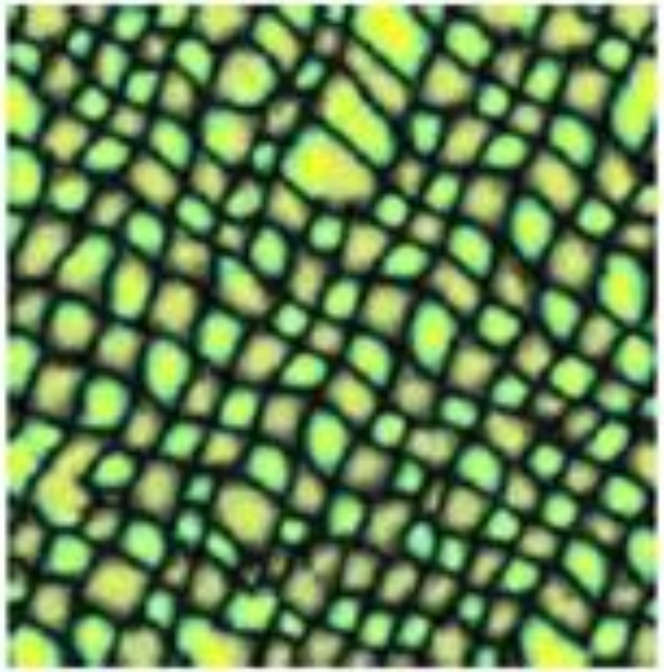

raster scan ordering (with randomly generated periodic boundaries)

Texture Synthesis: Pixel Based

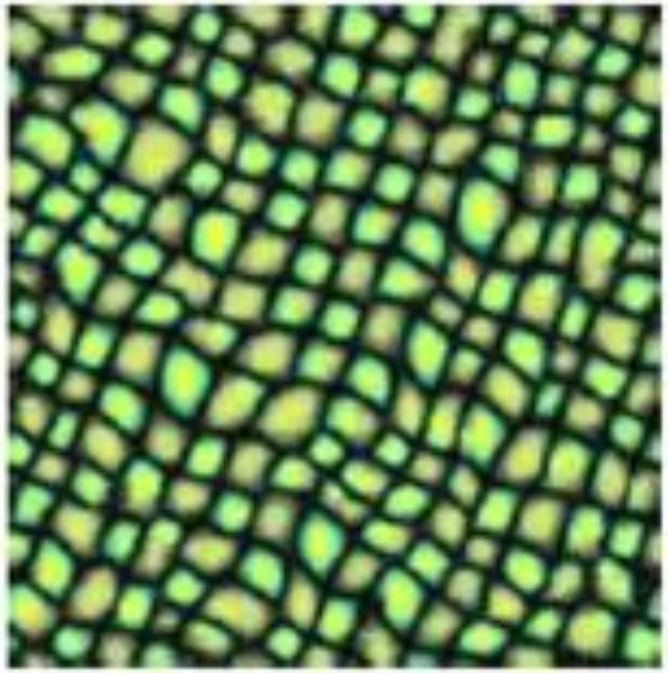
Sample



Heeger and Bergen



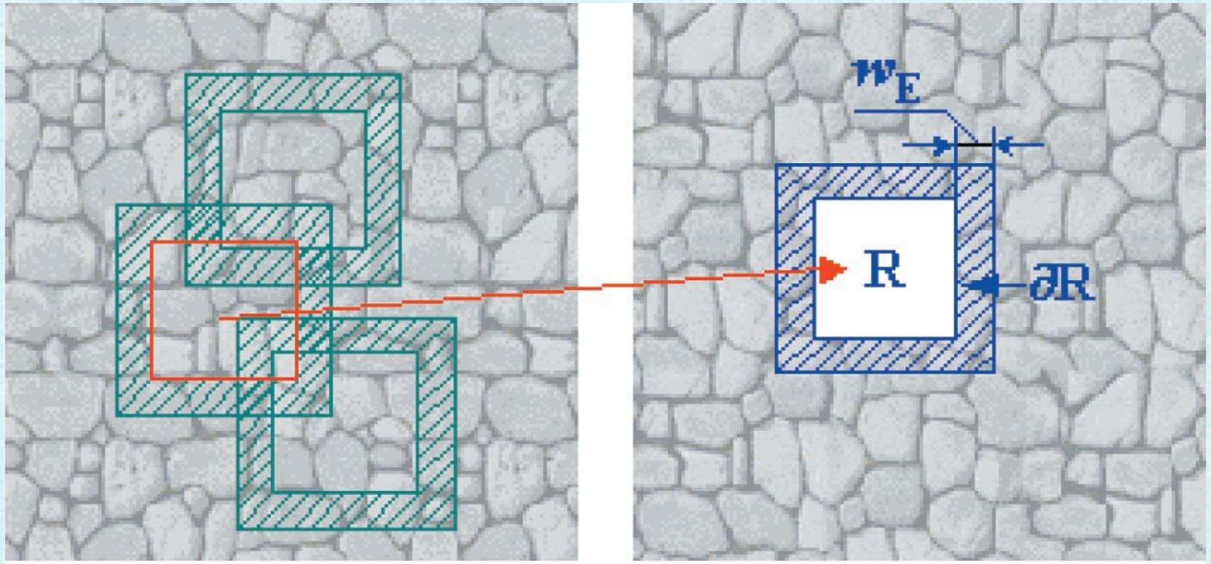
Efros and Leung



Wei and Levoy

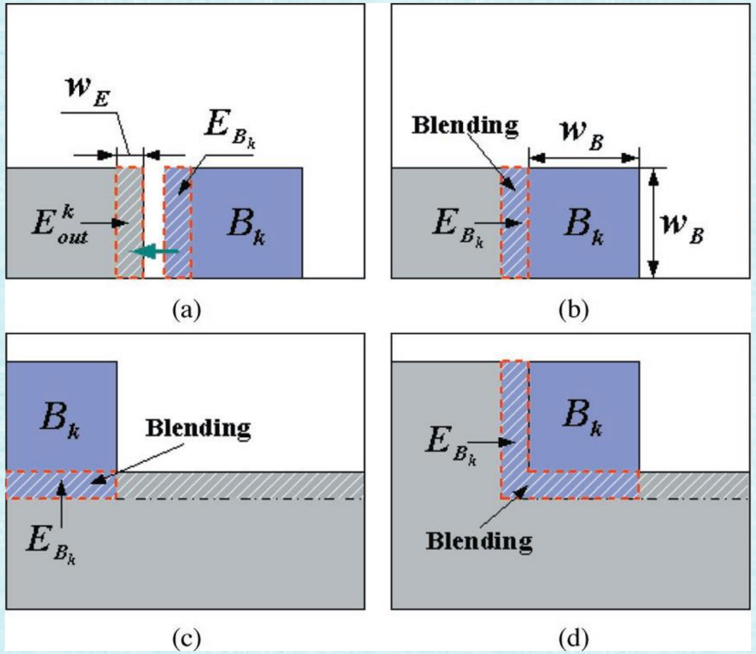
Texture Synthesis: Patch Based

- For each patch:
  - Search the original sample to find the candidate that best matches the overlap boundaries
  - Choose the best candidate
  - Blend overlapped regions to remove “seams”

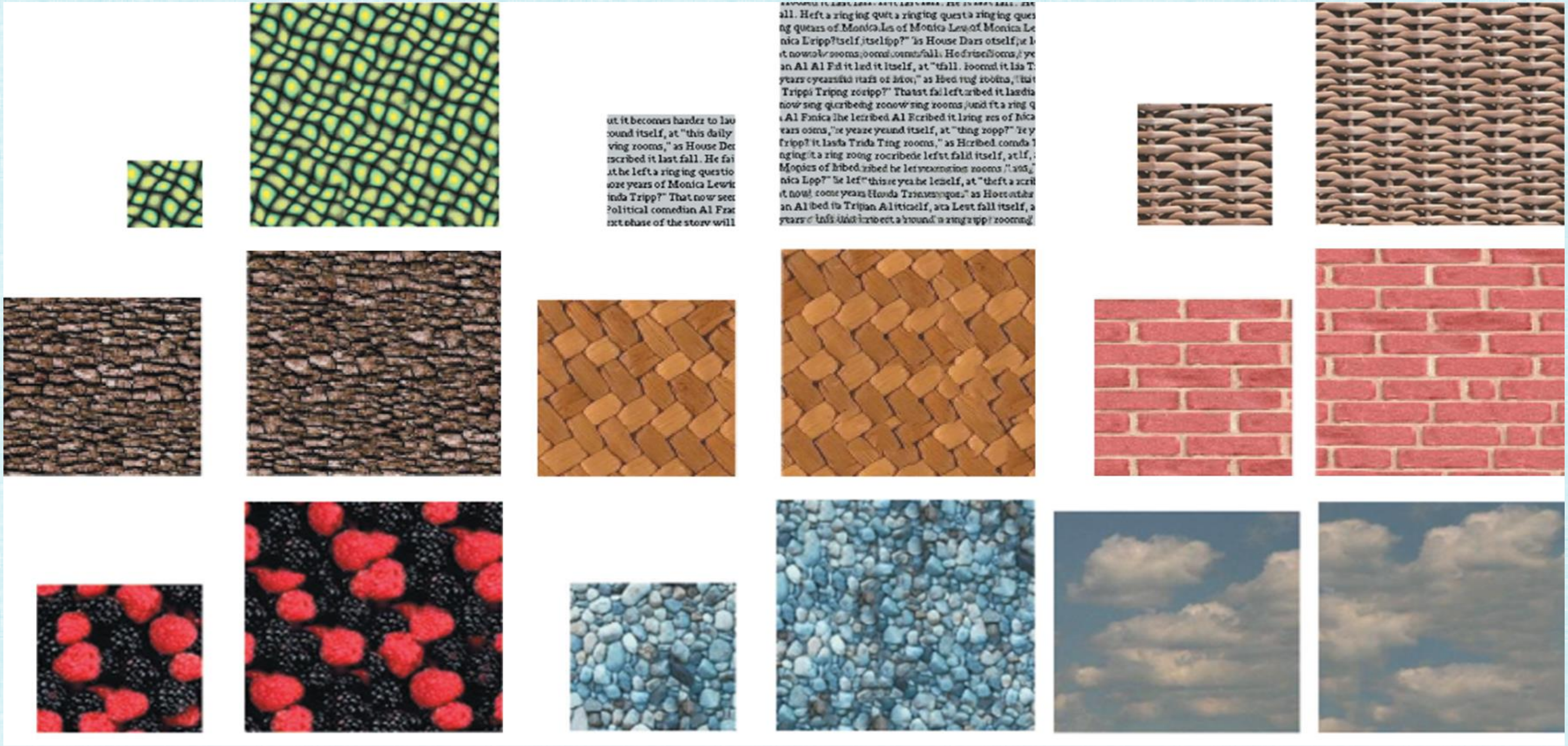


sample texture

matching boundary regions



Texture Synthesis: Patch Based

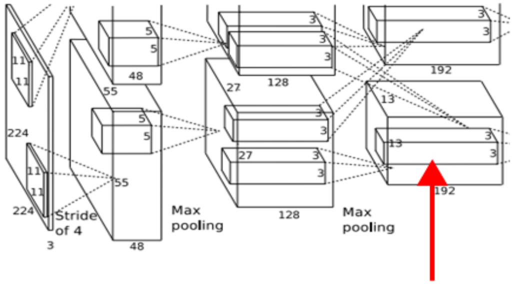


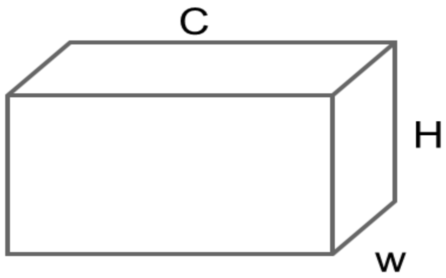
Texture Synthesis: AI Methods

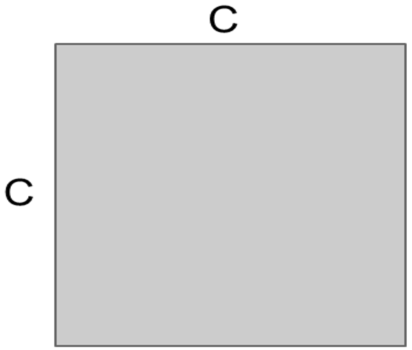
CS231N

Neural Texture Synthesis: Gram Matrix


This image is in the public domain.







Each layer of CNN gives C x H x W tensor of features; H x W grid of C-dimensional vectors

Outer product of two C-dimensional vectors gives C x C matrix measuring co-occurrence

Average over all HW pairs of vectors, giving **Gram matrix** of shape C x C

Efficient to compute; reshape features from

C x H x W to =C x HW

then compute G = FF^T

Texture Synthesis: AI Methods

CS231N

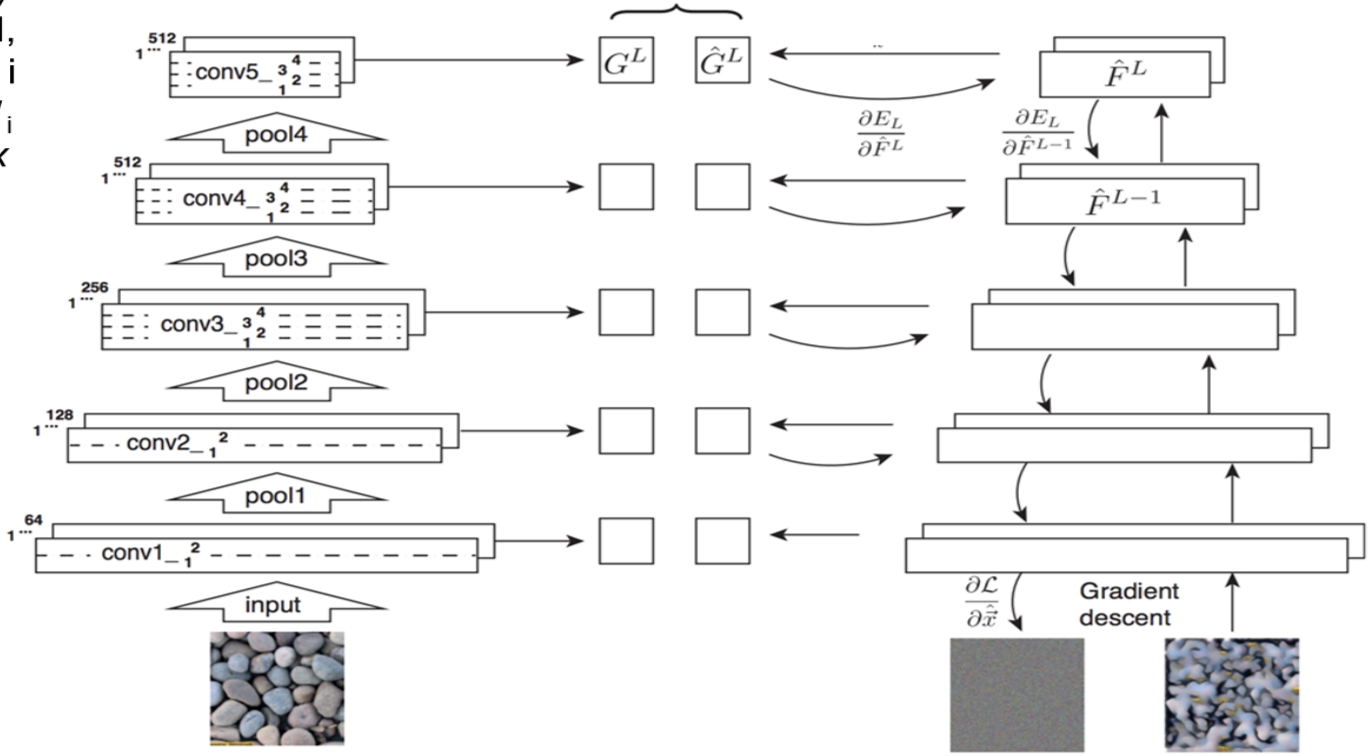
Neural Texture Synthesis

$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} \left(G_{ij}^l - \hat{G}_{ij}^l\right)^2 \qquad \mathcal{L}(\vec{x}, \hat{\vec{x}}) = \sum_{l=0}^{L} w_l E_l$$

1. Pretrain a CNN on ImageNet (VGG-19)
2. Run input texture forward through CNN, record activations on every layer; layer i gives feature map of shape $C_i \times H_i \times W_i$
3. At each layer compute the *Gram matrix* giving outer product of features:

$$G_{ij}^l = \sum_k F_{ik}^l F_{jk}^l \quad (\text{shape } C_i \times C_i)$$

4. Initialize generated image from random noise
5. Pass generated image through CNN, compute Gram matrix on each layer
6. Compute loss: weighted sum of L2 distance between Gram matrices
7. Backprop to get gradient on image
8. Make gradient step on image
9. GOTO 5

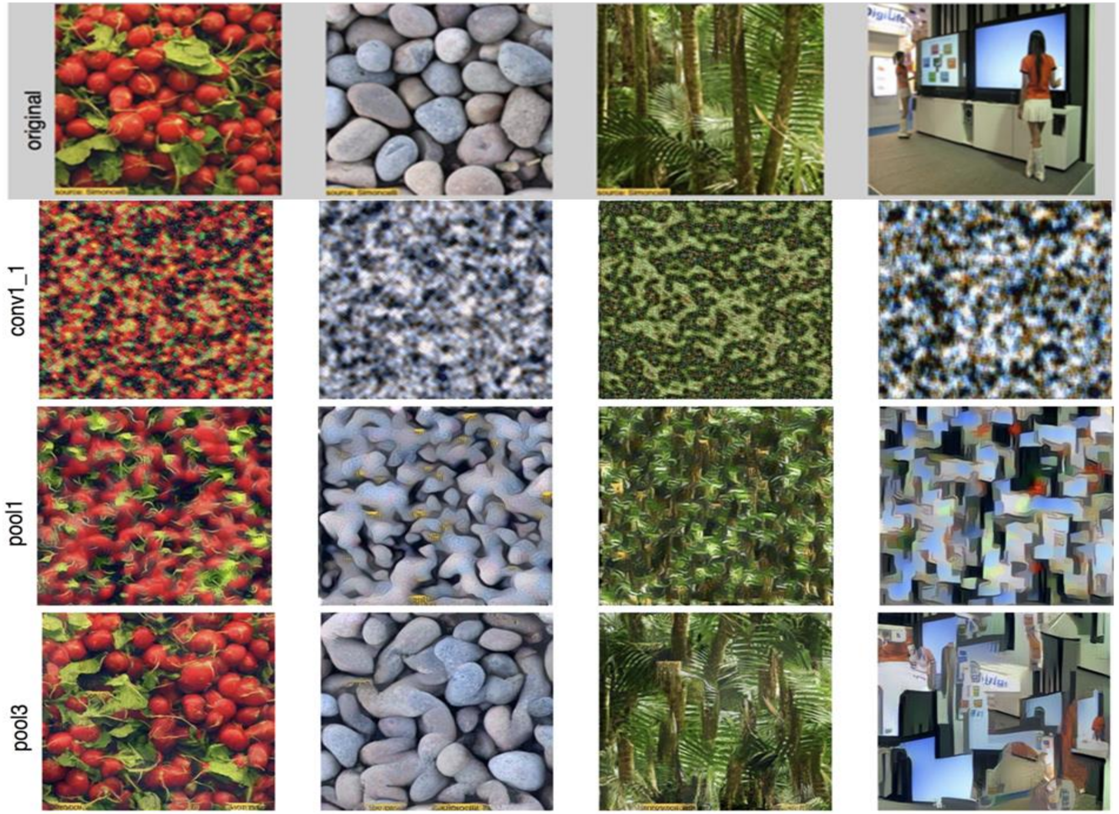


Gatys, Ecker, and Bethge, "Texture Synthesis Using Convolutional Neural Networks", NIPS 2015

Texture Synthesis: AI Methods

CS231N

Neural Texture Synthesis



Reconstructing texture from higher layers recovers larger features from the input texture

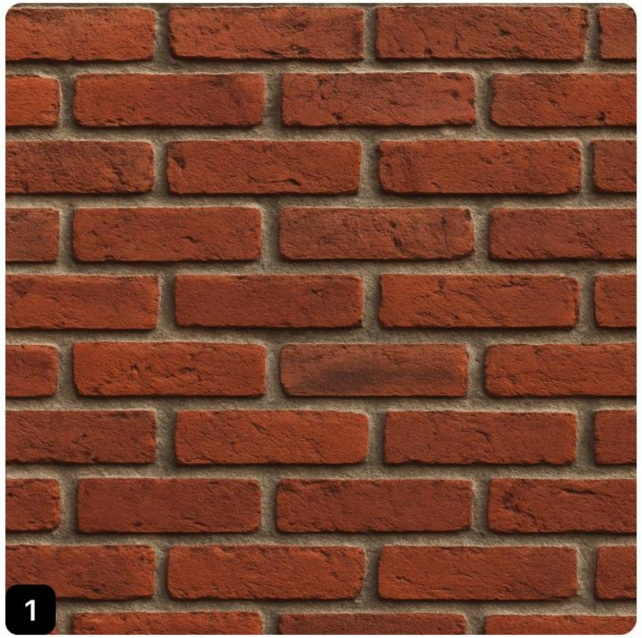
Gatys, Ecker, and Bethge, "Texture Synthesis Using Convolutional Neural Networks", NIPS 2015

Texture Synthesis: ChatGPT

generate a brick texture

Which image do you like more?

Skip



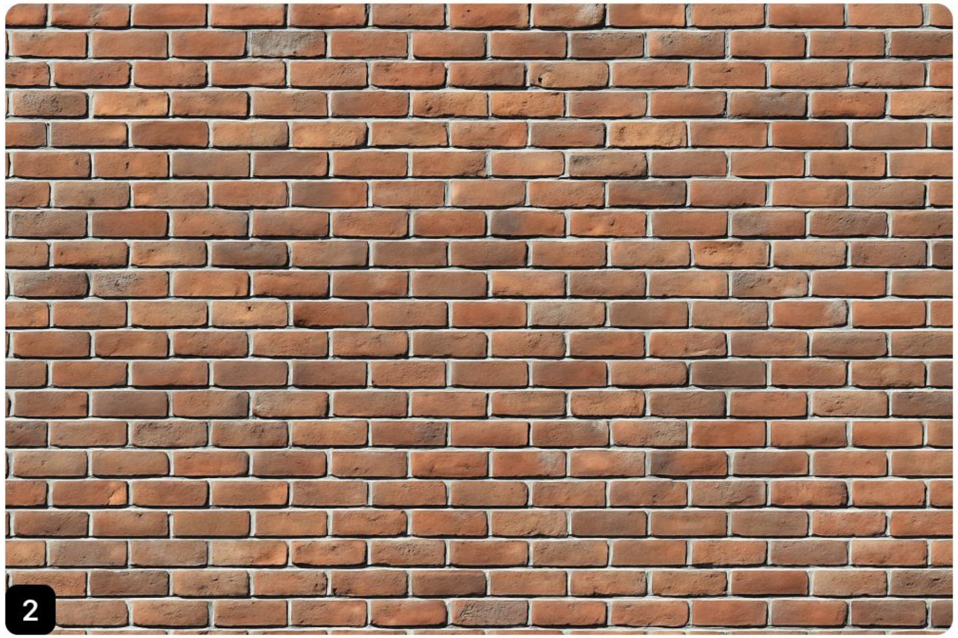
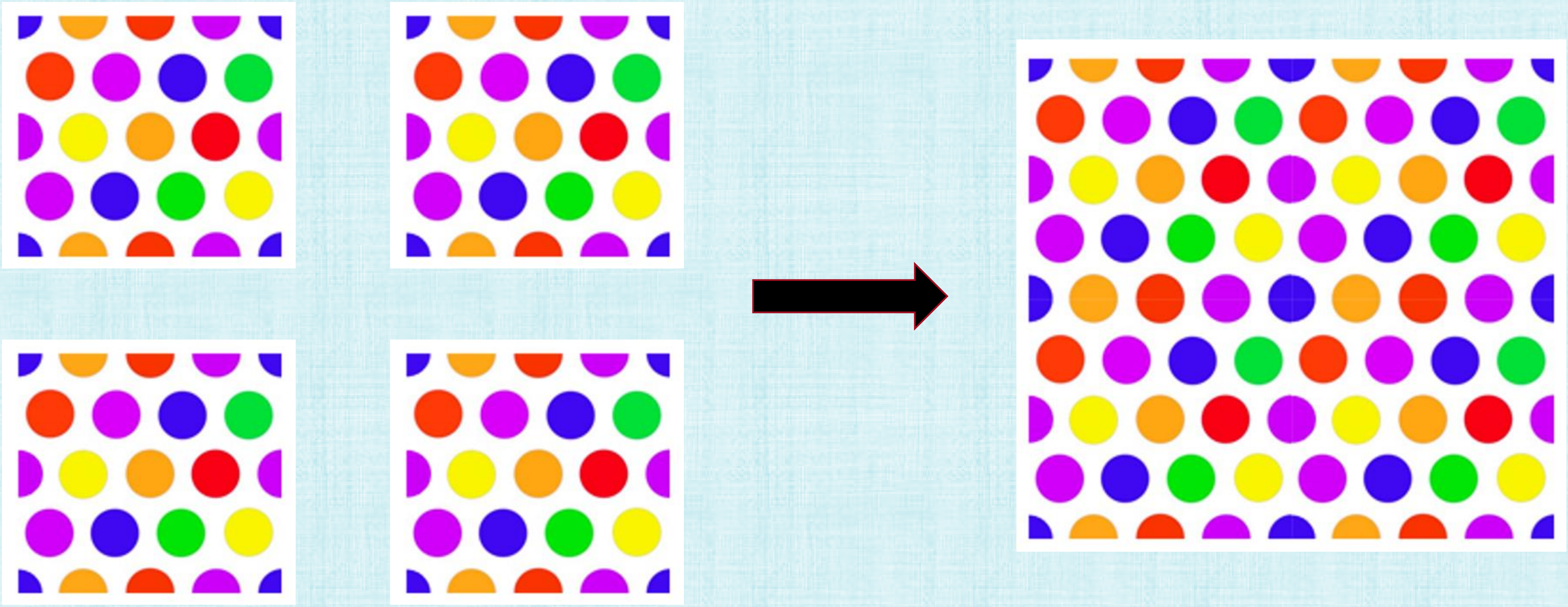


Image 1 is better

Image 2 is better

Don’t Stretch Textures!

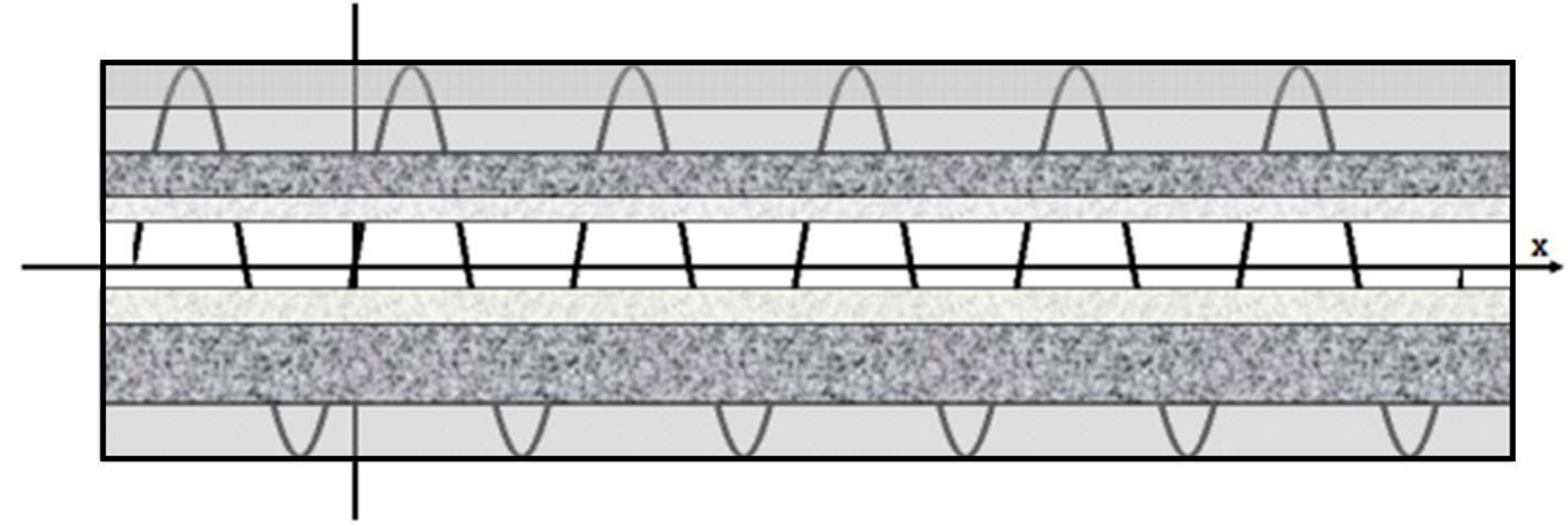
- Stretching out 10 bricks to cover an entire wall of a building is going to look unrealistic!
- Instead, tile the texture (requires periodic boundaries)



Marble Texture

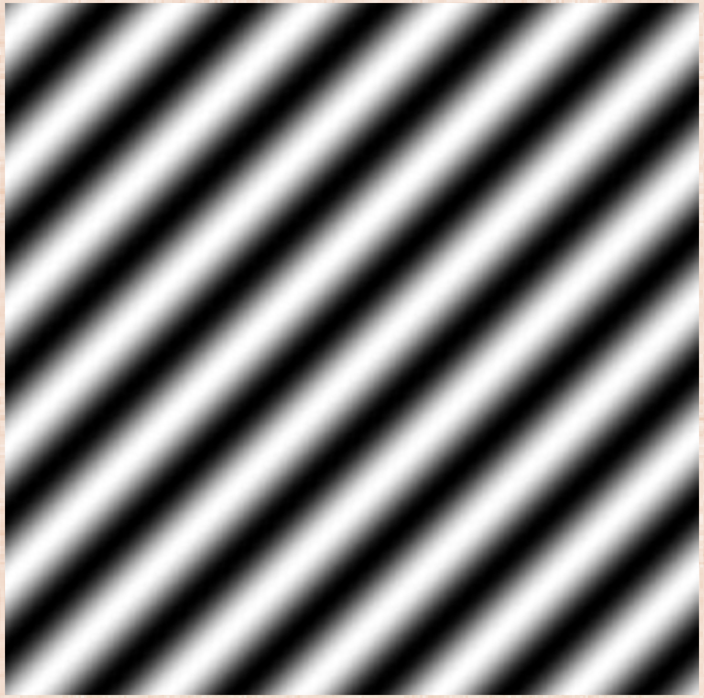
- Define layers of different colors
- Use a function to map layer colors to (u, v) texture locations
- For example:

$$marbleColor(u, v) = LayerColor\big(\sin(k_u u + k_v v)\big)$$

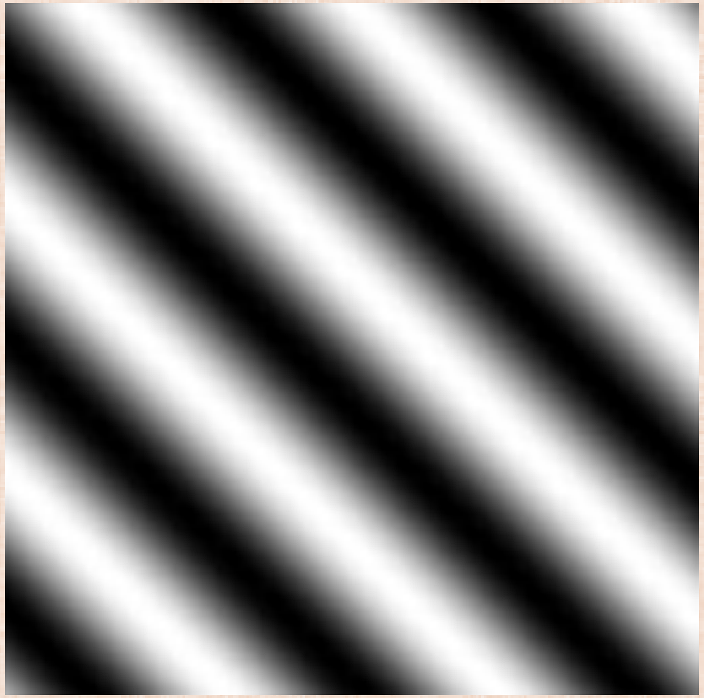


Marble Texture

- k_u and k_v are spatial frequencies
- (k_u, k_v) determines the direction, and $\frac{2\pi}{\sqrt{k_u^2+k_v^2}}$ determines the periodicity
- Problem: too regular (still need to add noise/randomness)



higher frequency



lower frequency

Perlin Noise

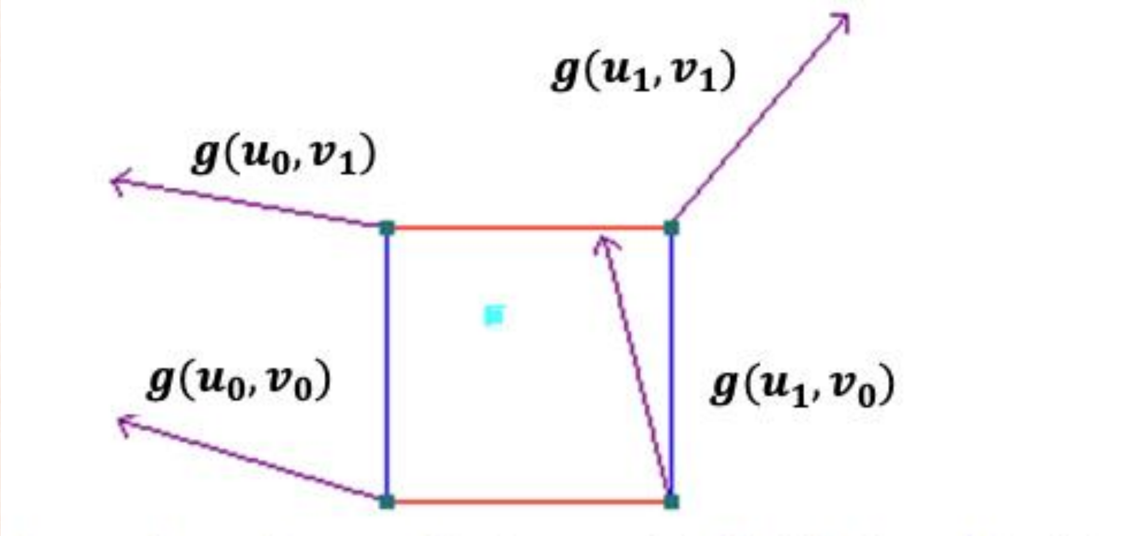
- Noise should have both coherency and structure, in order to look more natural
- Ken Perlin proposed a specific (and amazing!) method for doing this

Perlin Noise

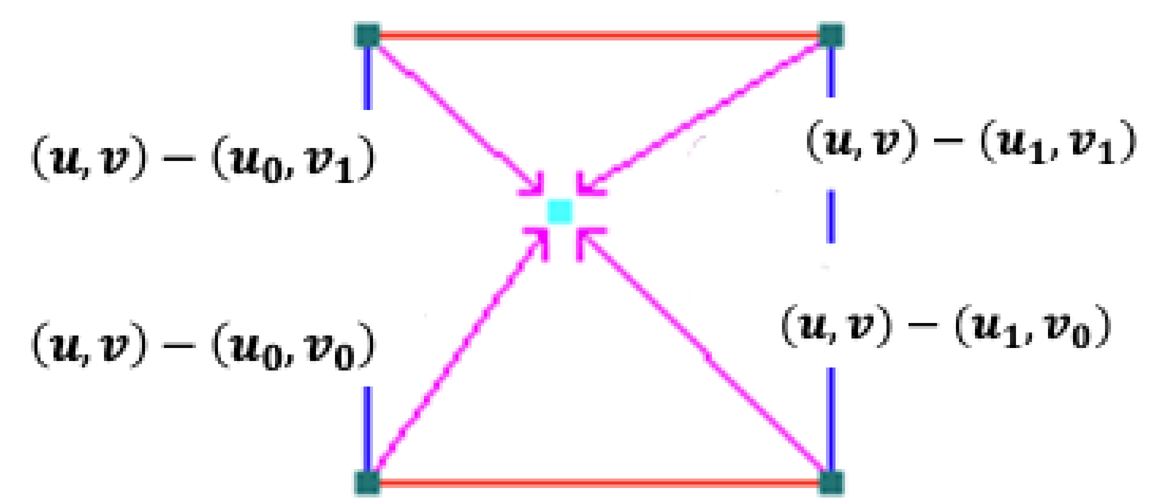
- Place a 2D grid over the texture image, and assign a random (unit) gradient $g(u_i, v_j)$ to each grid point
- For each pixel, compute the dot-products between vectors from the grid corners and the corresponding gradients
- Take a weighted average of the result:

$$noise(u,v) = \sum_{i=0,1;j=0,1} w\left(\frac{u-u_i}{\Delta u}\right) w\left(\frac{v-v_j}{\Delta v}\right) g(u_i, v_j) \cdot (u-u_i, v-v_j)$$

- Cubic weighting (for smoothness): $w(t) = 2|t|^3 - 3|t|^2 + 1$ for $t \in (-1,1)$



4 pseudorandom gradients associated with the grid points



Vectors from the grid points to (u, v)

Multiple Scales

- Natural textures tend to contain a variety of feature sizes
- Mimic this by adding together noises with different frequencies and amplitudes:

$$perlin(u, v) = \sum_k noise\big(frequency(k) * (u, v)\big) * amplitude(k)$$

- Each successive noise function is twice the frequency of the previous one:

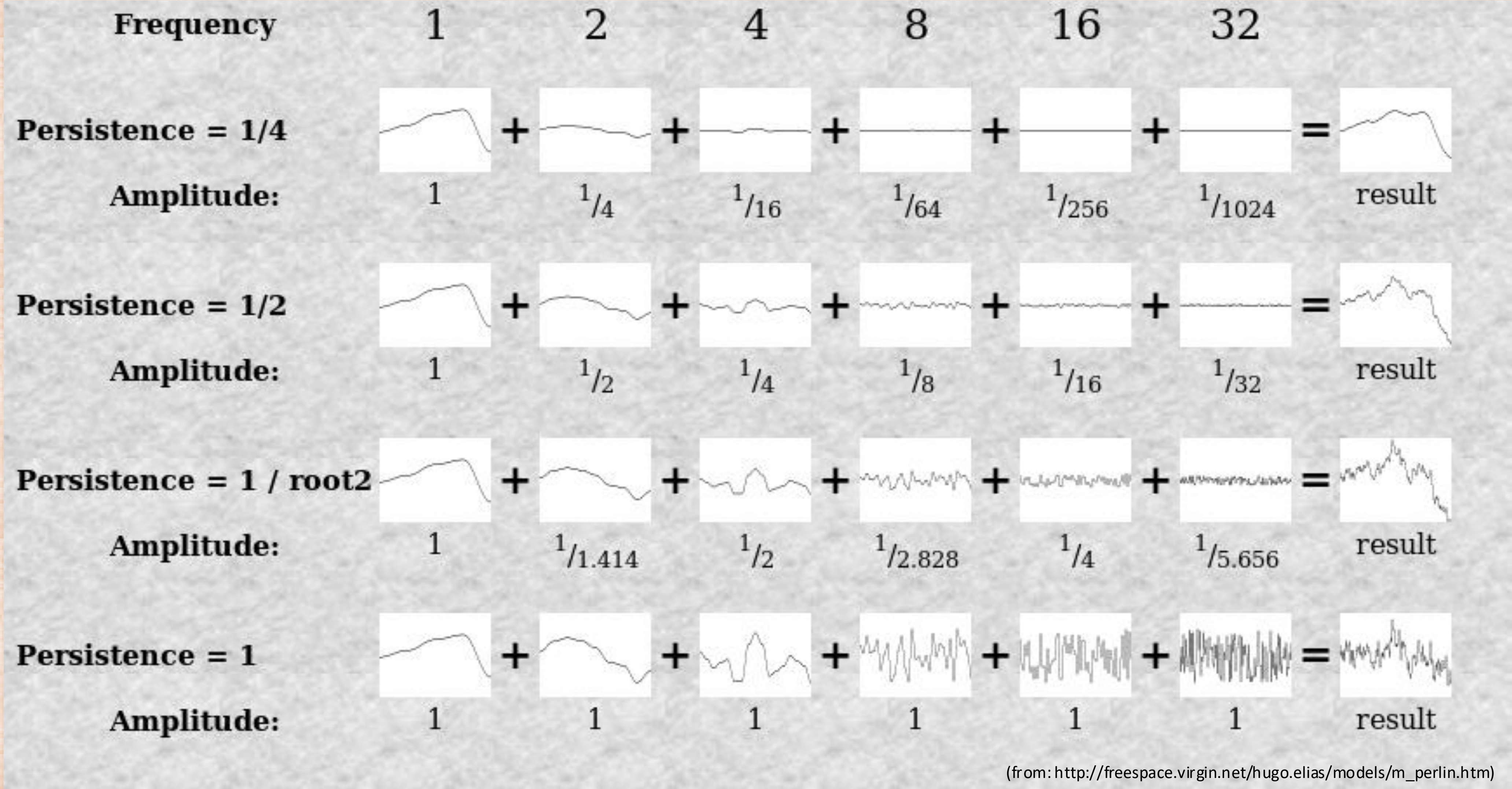
$$frequency(k) = 2^k$$

- The amplitude of higher frequencies is measured by a persistence parameter (≤ 1)
- Higher frequencies have a diminished contribution:

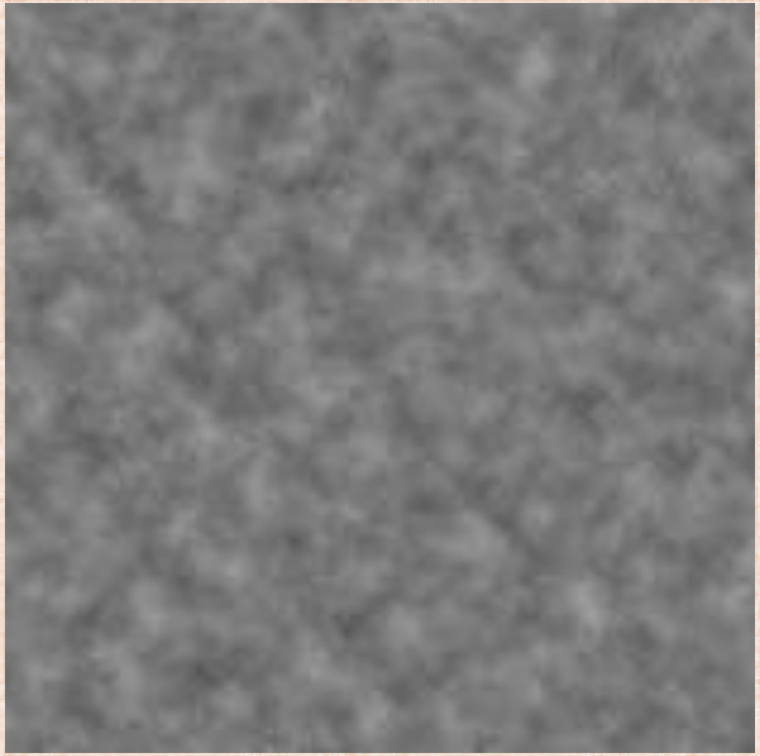
$$amplitude(k) = persistence^k$$

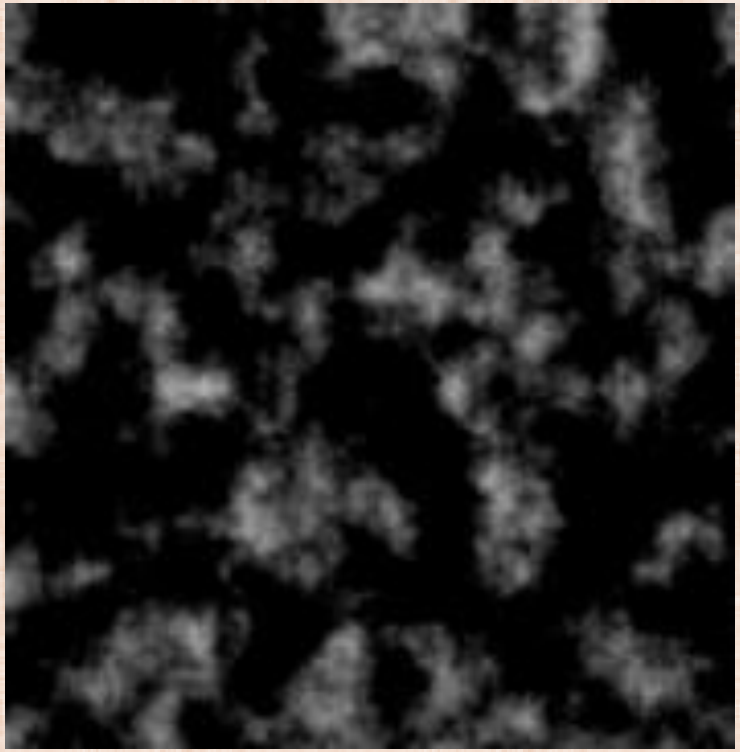
1D Examples

- Smaller persistence gives less higher frequency noise and thus a smoother result



2D Examples

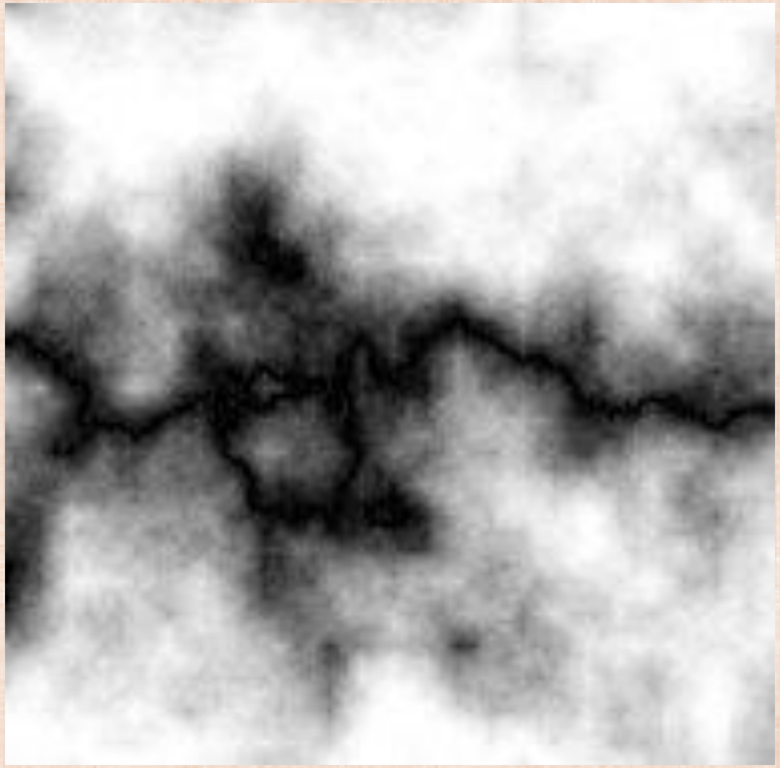


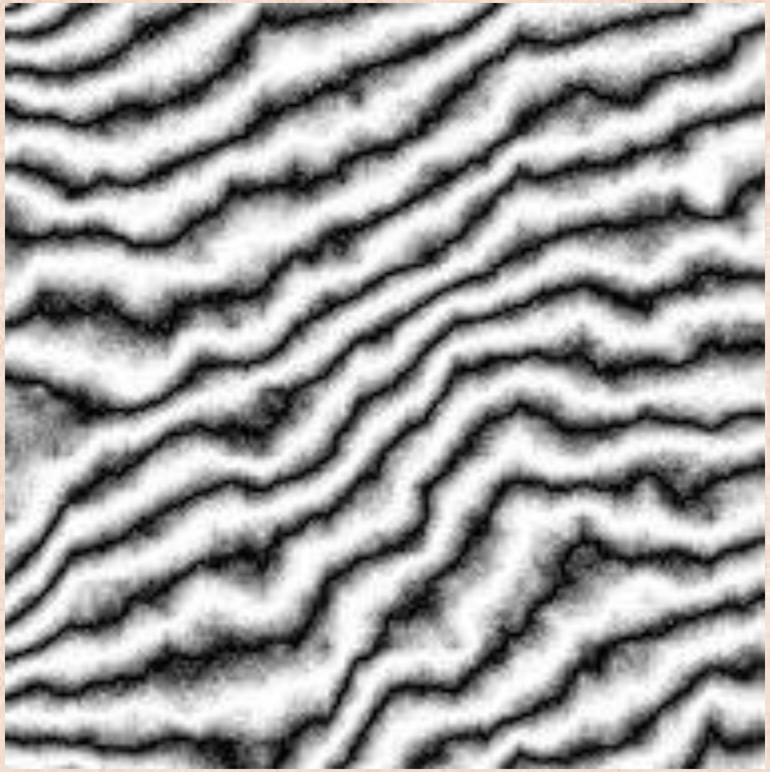


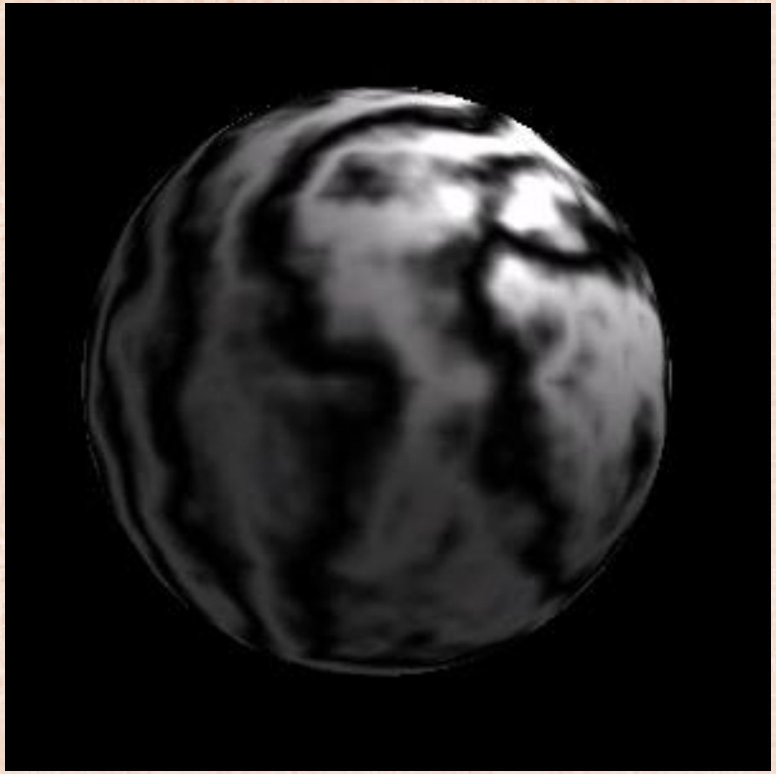
Marble Texture + Perlin Noise

- Set the value of A to scale the amount of noise:

$$marbleColor(u,v) = LayerColor\left(\sin\left(k_u u + k_v v + A * perlin(u,v)\right)\right)$$



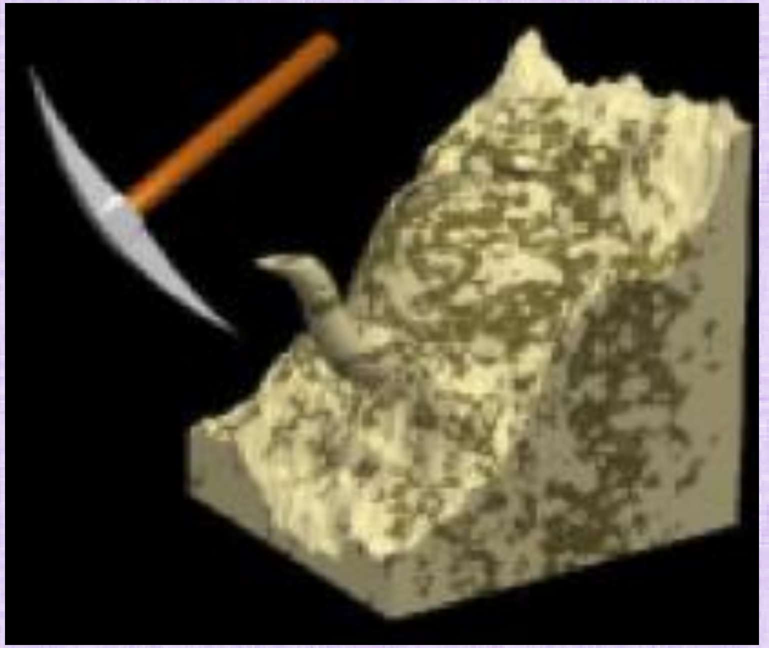


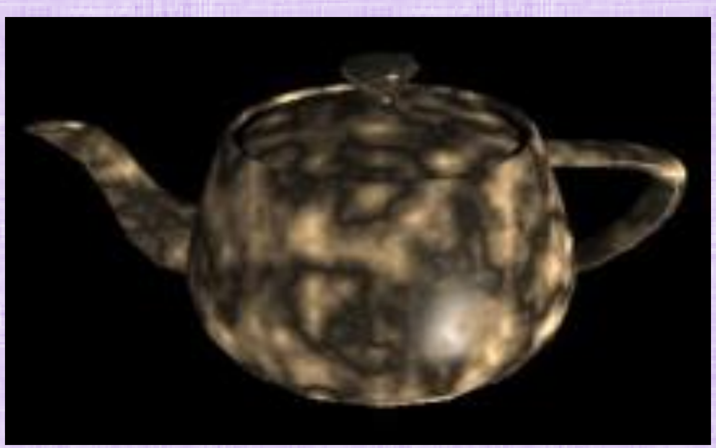


3D Marble Texture

- “Carve” an object out of a 3D texture
- Marble texture function w/Perlin noise (for 3D):

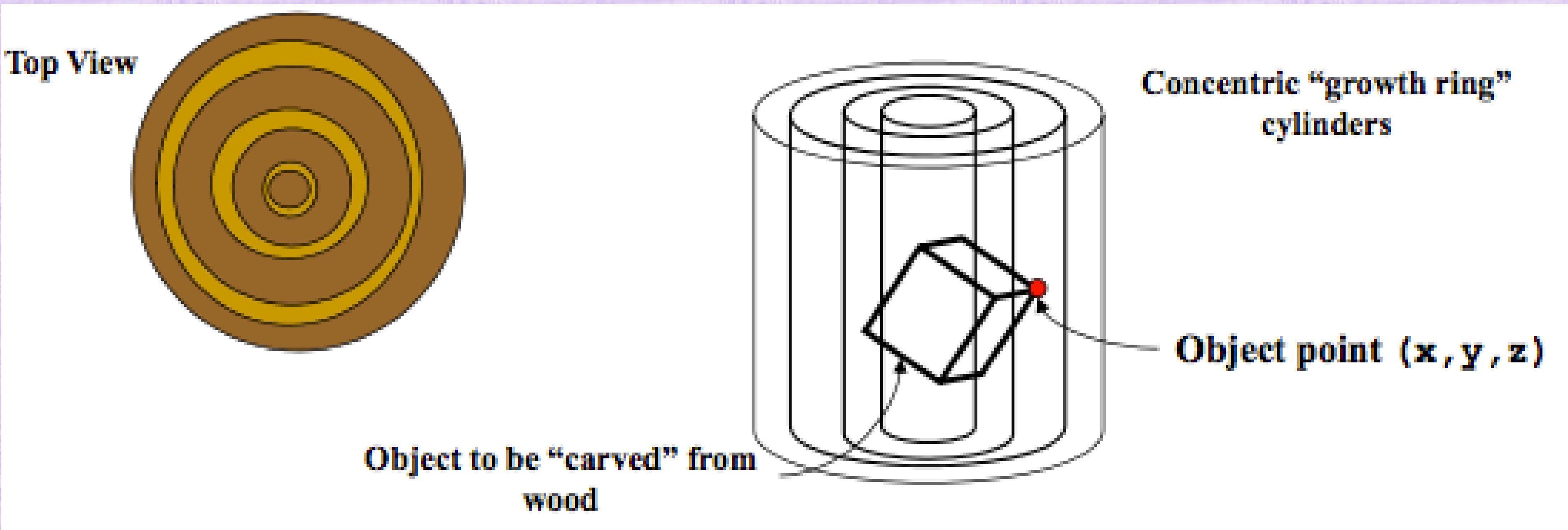
$$marbleColor(u,v,w) = LayerColor\left(\sin\left(k_u u + k_v v + k_w w + A * perlin(u,v,w)\right)\right)$$

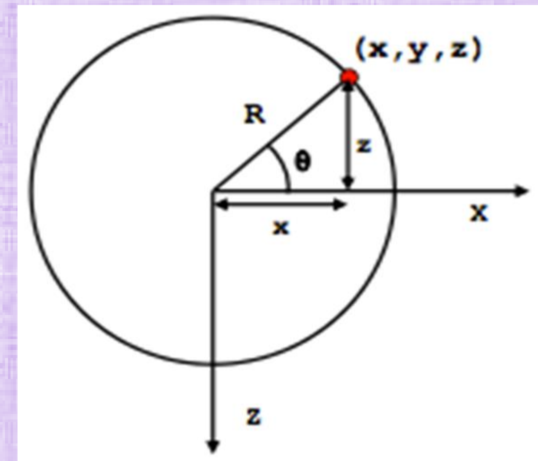




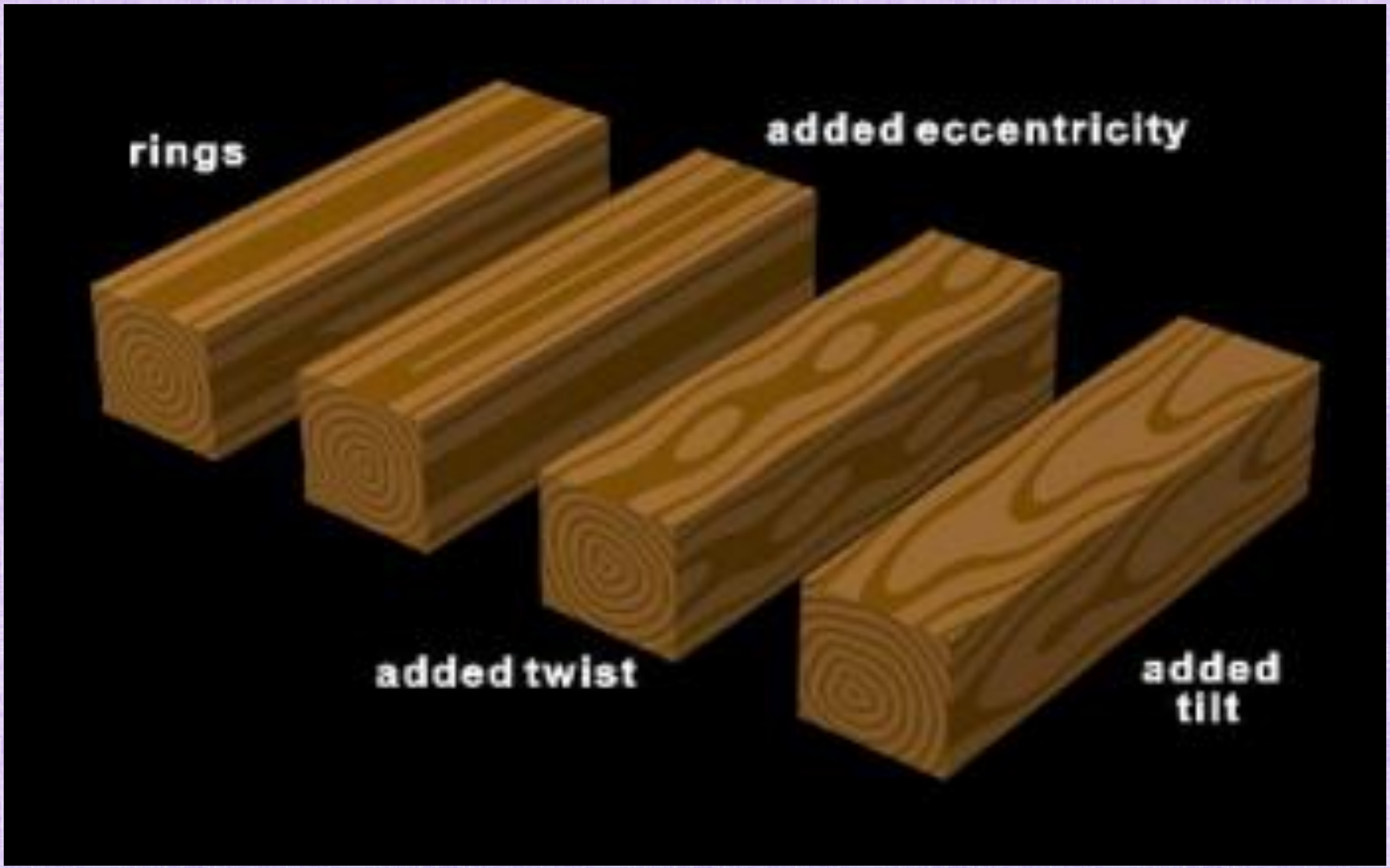
3D Wood Texture

- Procedurally generate tree rings (and cut the object out of the 3D texture)
- Cylindrical coordinates for (x, y, z) object points: $H = y, R = \sqrt{x^2 + z^2}, \theta = \tan^{-1}\left(\frac{z}{x}\right)$

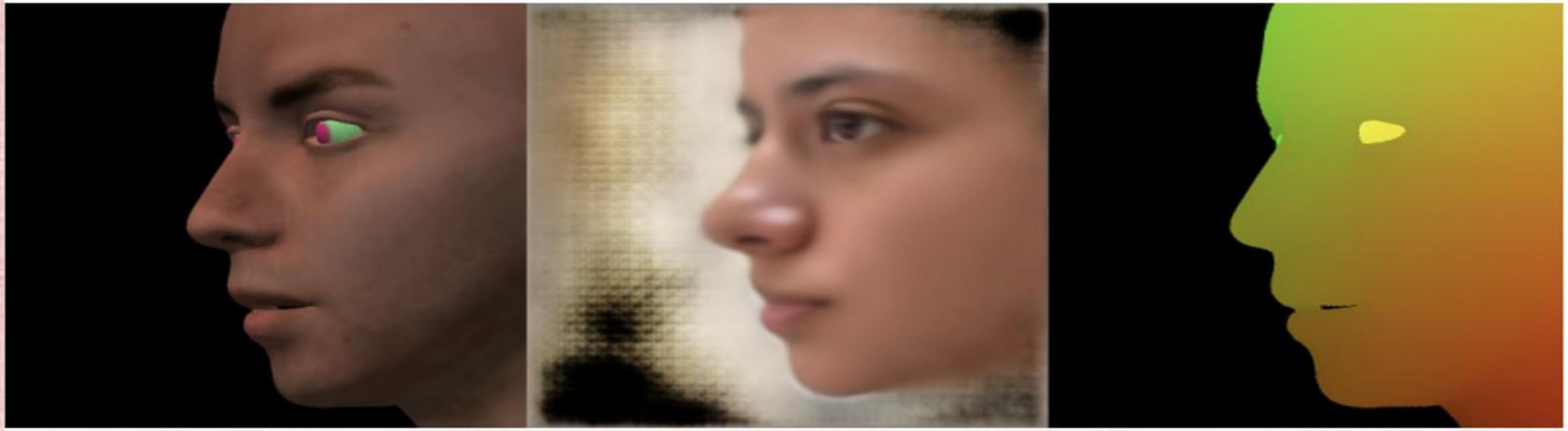




3D Wood Texture



Machine Learning: Deep Fakes



Machine Learning: Text to Video

elon musk in a space suit

3d animation