



Texture Synthesis

Misha Kazhdan

(601.457/657)

An Image Synthesizer. Perlin, 1985

Texture Synthesis by Non-Parametric Sampling. Efros and Leung, 1999

Image Quilting for Texture Synthesis and Transfer. Efros and Freeman, 2001

Wang Tiles for Image and Texture Generation. Cohen et al., 2003

What is a texture?



Courtesy Paul Bourke



What is a texture?

Texture is an image that exhibits:

- Stationarity – different regions “look similar”



Courtesy Paul Bourke



What is a texture?

Texture is an image that exhibits:

- Stationarity – different regions “look similar”
- Locality – structure is defined in a small neighborhood



Courtesy Paul Bourke



What is a texture?

Texture is an image that exhibits:

- Stationarity – different regions “look similar”
- Locality – structure is defined in a small neighborhood



Note:

Any image can be texture-mapped.

We are focusing on images that are qualitatively *textures*.



How can we get textures?

Photographs

Manual texture synthesis

Automatic texture synthesis

- Procedural generation
- Extrapolation

Photographs



Easy and fast (if we can find the texture we want)!

What if our photo is not big enough?



Courtesy NVIDIA



Photographs

Easy and fast (if we can find the texture we want)!

What if our photo is not big enough?

- Stretching changes scale, image quality



Courtesy NVIDIA

Photographs

Easy and fast (if we can find the texture we want)!

What if our photo is not big enough?

- Stretching changes scale, image quality
- Tiling looks repetitive (and can generate seams)

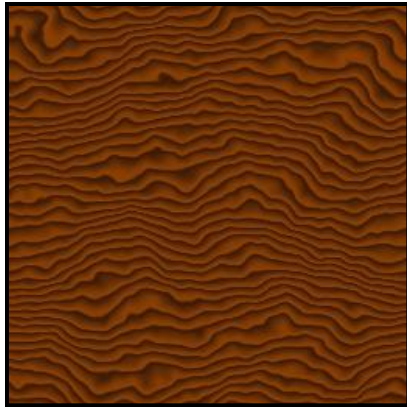


Manual Texture Synthesis

There are “texture painters”...

- × Time consuming
- × Difficult

Automatic Texture Synthesis

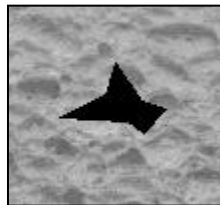


How do we
create this

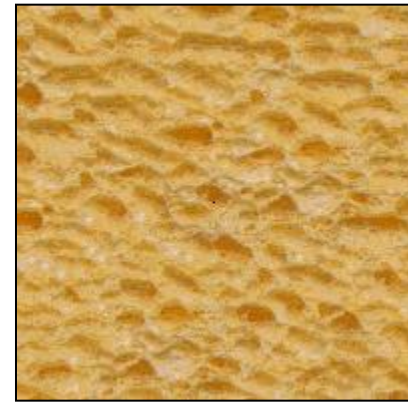
Ex nihilo



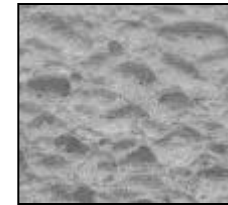
How do we go
from this...



Or from this...



...to this?



...to this?

Ex materia



Procedural Textures

Generated algorithmically instead of by an artist

Good for certain natural phenomena:

- Wood grain
- Marble
- Fire
- Etc.



Perlin-noise Textures

Key Idea:

Textures in nature (e.g. wood grain, mountain ranges, marble, clouds, fire) are composed of layers of detail

- » Individual layers appear as random signals of a fixed **frequency**
- » The textures are determined by the **amplitudes** with which the different frequencies contribute

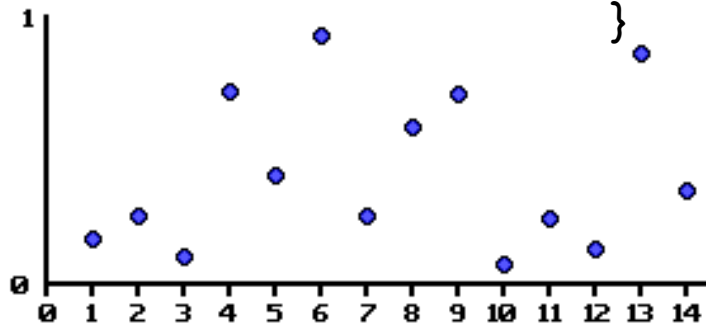


Perlin-noise Textures (Per Layer)

We need:

- Noise
- Interpolation

```
void init( float noise[] , int n , float amp )  
{  
    for( int i=0 ; i<n ; i++ ) noise[i] = Random() * amp;  
}  
  
float sample( float x , const float noise[] , int n )  
{  
    x *= n;  
    int ix = (int)floor( x );  
    return Interpolate( noise[ix] , noise[ix+1] , x-ix );  
}
```



Noise



Interpolation

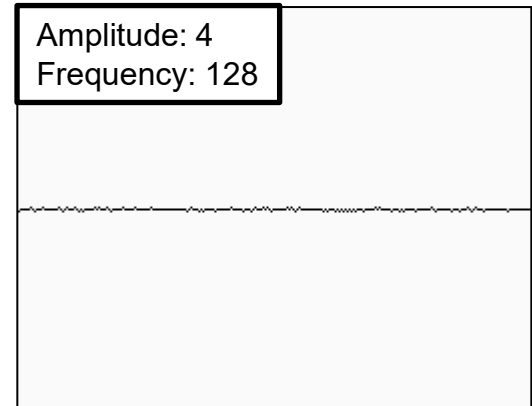
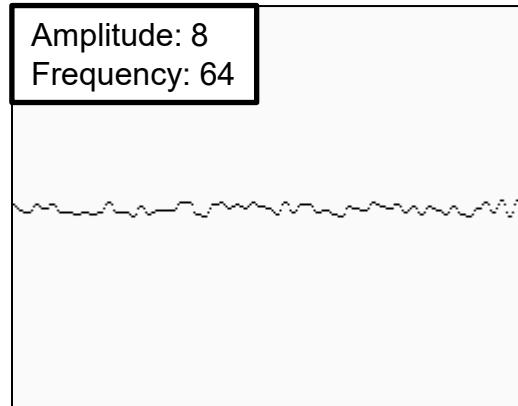
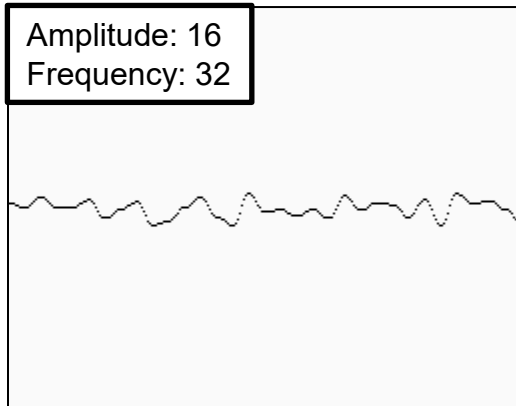
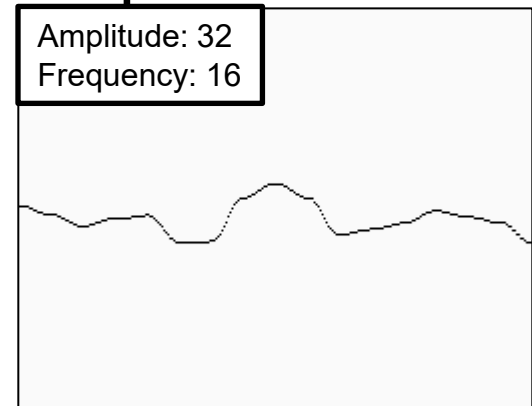
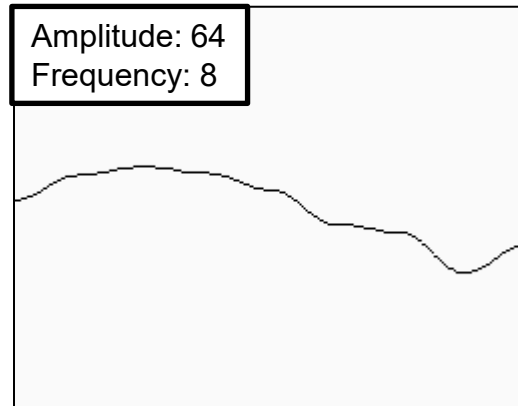
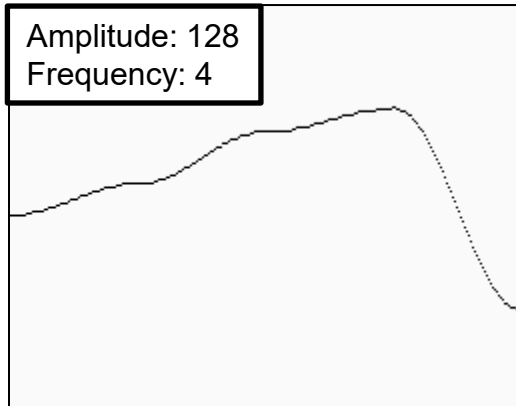
Frequency/Resolution := samples/interval
Amplitude := Magnitude of the random number

Courtesy Hugo Elias



Perlin-noise Textures

Sum noise at different resolutions/amplitudes



Standardly:

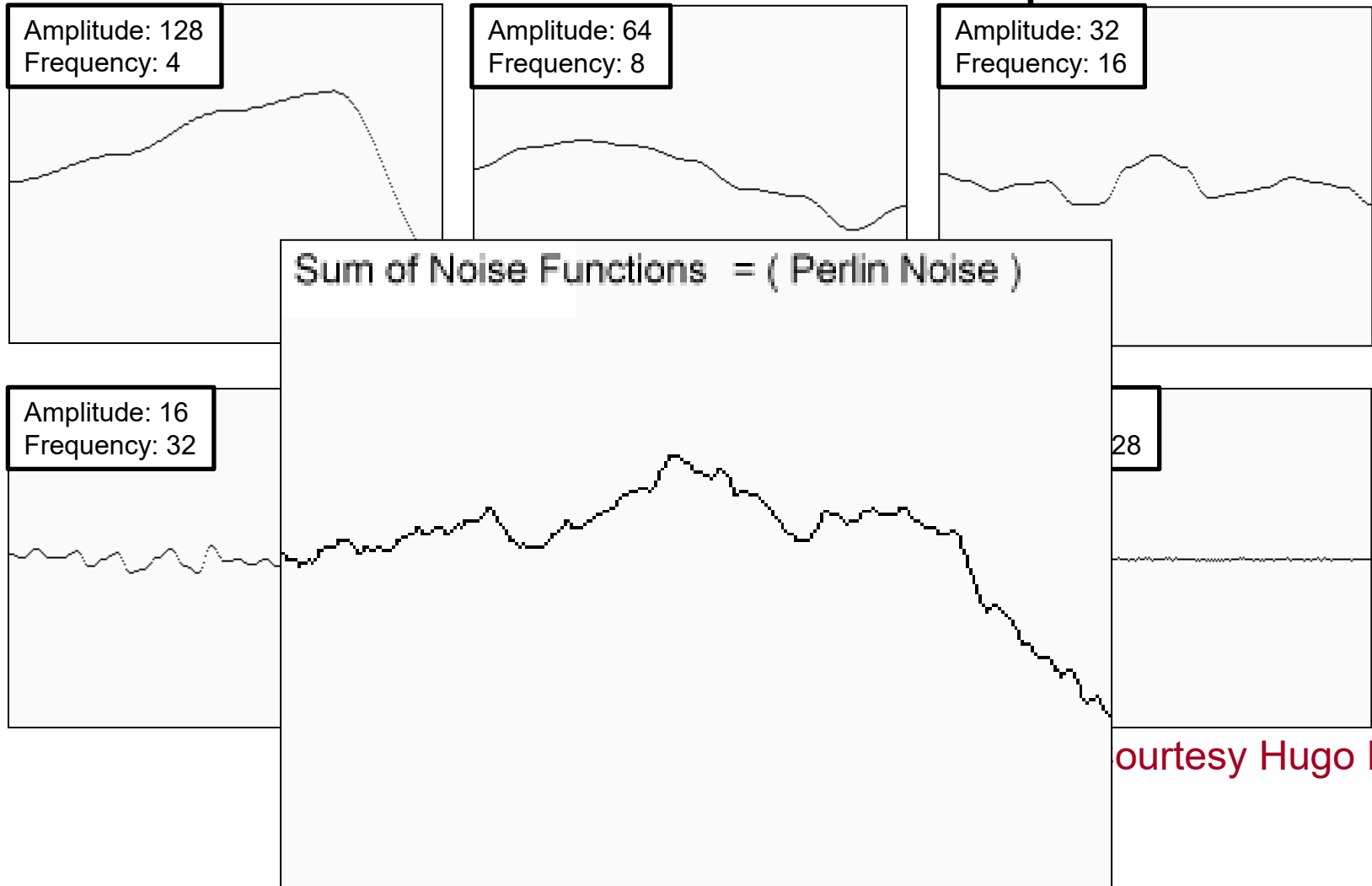
- Frequencies are powers of two
- Amplitude decreases by a constant factor with frequency.

Courtesy Hugo Elias



Perlin-noise Textures

Sum noise at different resolutions/amplitudes

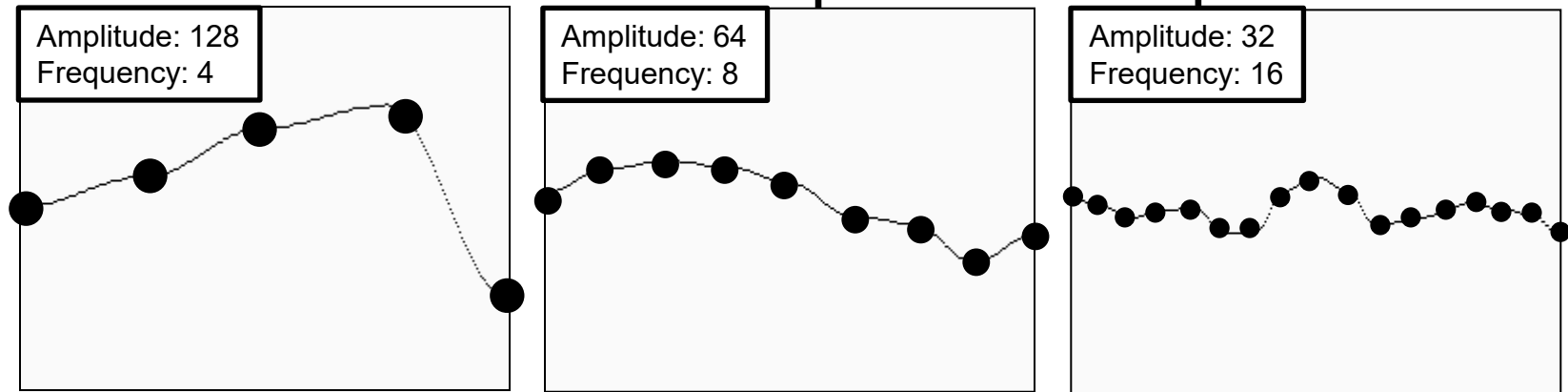


courtesy Hugo Elias



Perlin-noise Textures

Sum noise at different frequencies/amplitudes



How much data would we need to *store* the texture?

If we sample at resolution n , we need $2n$ values:

- n at the finest level
- $n/2$ at the next level,
- etc.

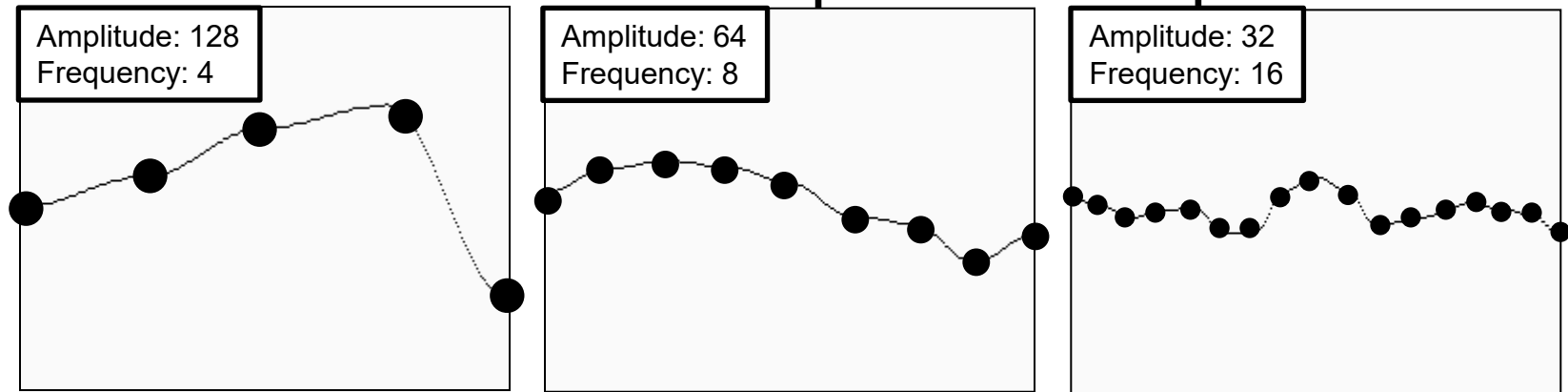
In d dimensions, $O(n^d)$.

```
void init( float noise[], int n, float amp )
{
    for( int i=0 ; i<n ; i++ ) noise[i] = Random() * amp;
}
float sample( float x, const float noise[] )
{
    x *= n;
    int ix = (int)floor( x );
    return Interpolate( noise[ix], noise[ix+1], x-ix );
}
```



Perlin-noise Textures

Sum noise at different frequencies/amplitudes



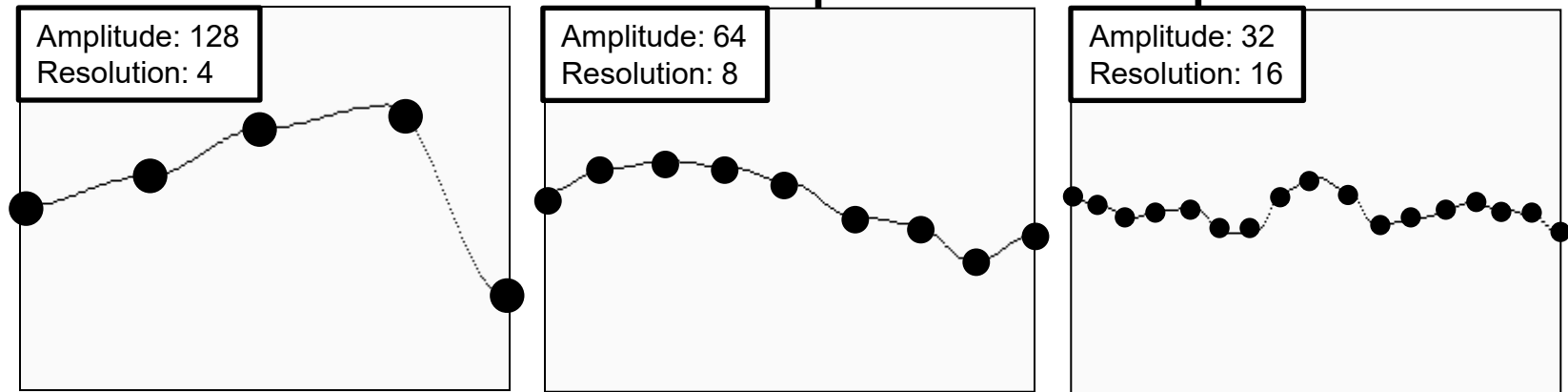
How much data do we need to *sample* the texture?

```
void init( float noise[] , int n , float amp )
{
    for( int i=0 ; i<n ; i++ ) noise[i] = Random() * amp;
}
float sample( float x , const float noise[] )
{
    x *= n;
    int ix = (int)floor( x );
    return Interpolate( noise[ix] , noise[ix+1] , x-ix );
}
```



Perlin-noise Textures

Sum noise at different frequencies/amplitudes



How much data do we need to *sample* the texture?

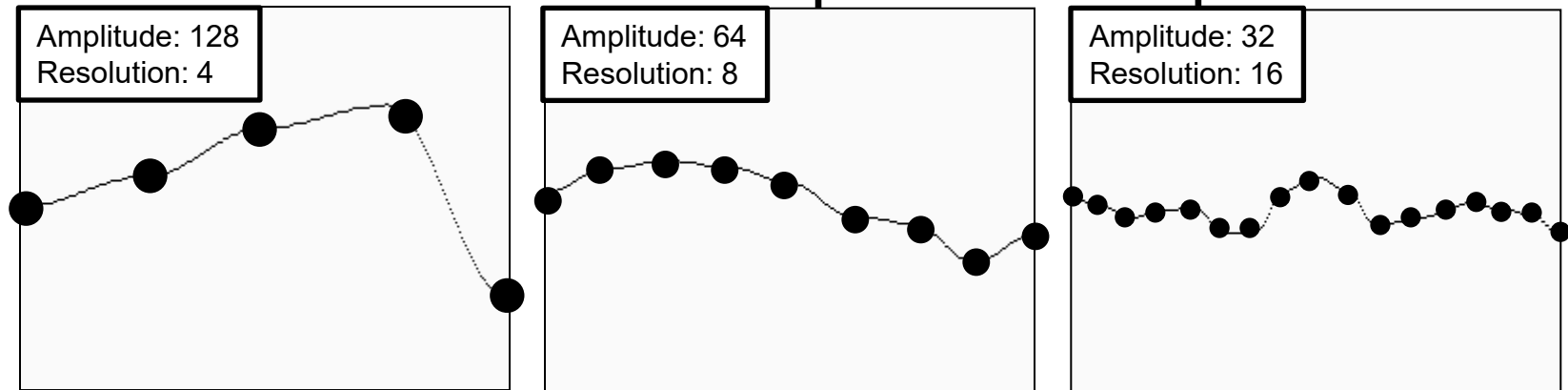
If we can get the random number generator to generate the same random value at index i , we only need to know the amplitudes at the different frequencies.

```
(float) sample(float x, int n, float amp)
{
    x *= n;
    int ix = (int)floor(x);
    srand(ix);
    float nx0 = Random() * amp;
    srand(ix+1);
    float nx1 = Random() * amp;
    return Interpolate(nx0, nx1, x-ix);
}
```



Perlin-noise Textures

Sum noise at different frequencies/amplitudes



How much *computation* is required to get the value at a point?

Linear interp. needs two values per level. With $L = \log(n)$ levels:

- Generate $2L$ random values
- Interpolate between the L pairs
- Sum the L interpolations

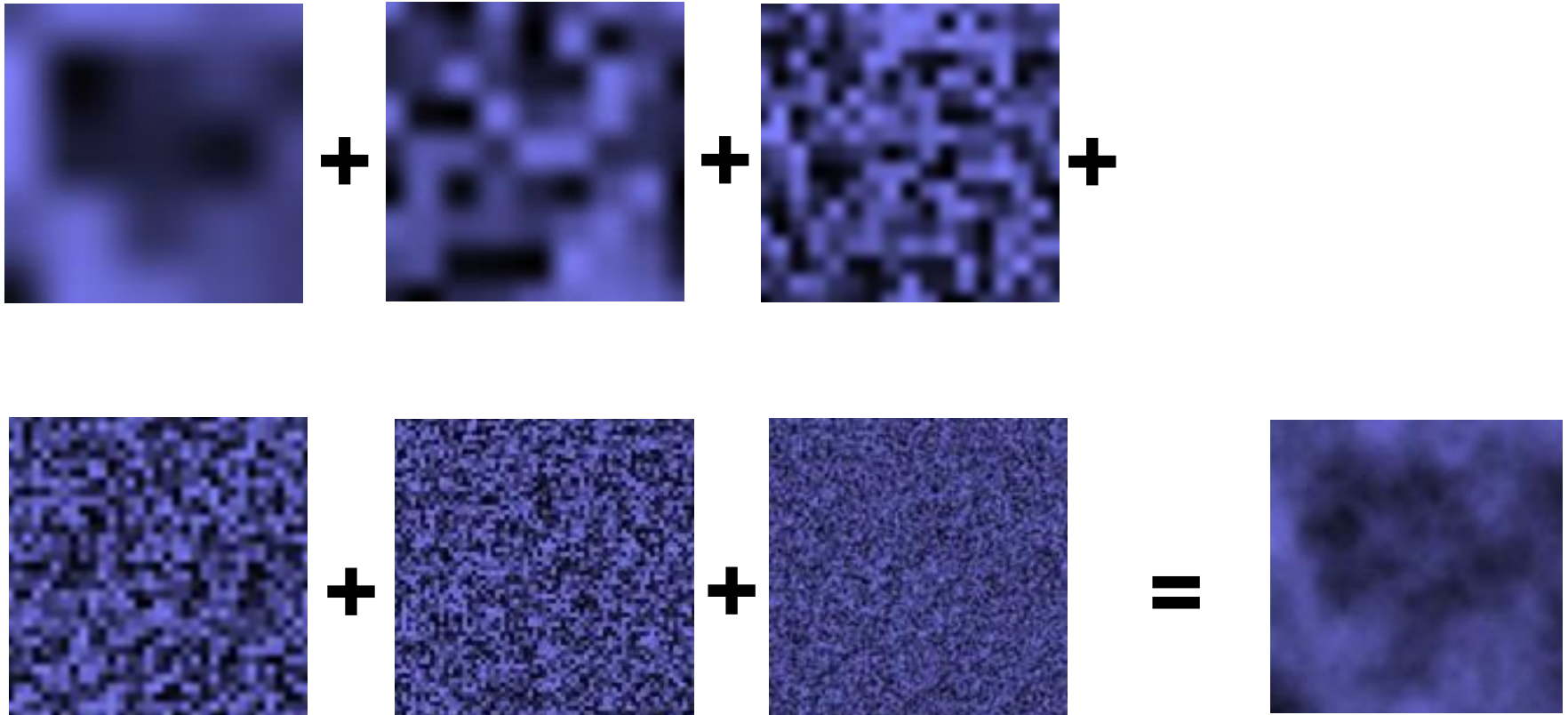
In d dimensions, $O(2^d L)$.

```
(float) float sample( float x , int n , float amp )  
{  
    x *= n;  
    int ix = (int)floor( x );  
    srand( ix );  
    float nx0 = Random() * amp;  
    srand( ix+1 );  
    float nx1 = Random() * amp;  
    return Interpolate( nx0 , nx1 , x-ix );  
}
```



Perlin-noise Textures: Nebula

Same idea with 2D images

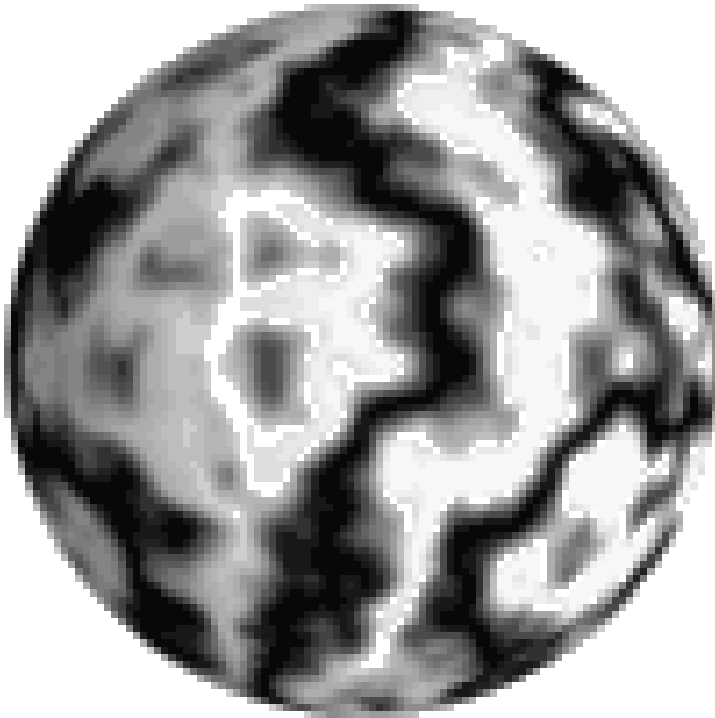


Courtesy Hugo Elias



Perlin-noise Textures: Marble/Wood

And even 3D textures



Hugo Elias

Note:

Can introduce anisotropy – using different amplitudes for the x -, y -, and z -directions.



Procedural Textures

Pros

- ✓ Constant memory overhead
- ✓ Can be computed efficiently $O(2^d L)$

Cons

- ✗ Only good for certain natural phenomena

Automatic Texture Synthesis



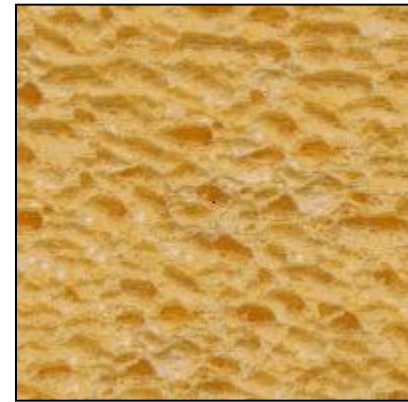
How do we
create this

Ex nihilo

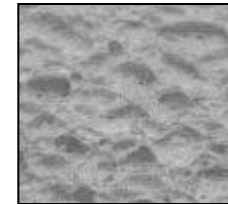
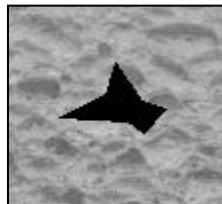


How do we go
from this...

Or from this...



...to this?



...to this?

Ex materia



Markov Models: Text

Assume we have:

- A fixed alphabet (a through z)
- An input text such as *agggcagcgggcg*

A 0th-order Markov Model:

- Assign probabilities to the characters based on the frequency of their occurrence in the input text:

$$P(a) = \frac{2}{13} \quad P(c) = \frac{3}{13} \quad P(g) = \frac{8}{13}$$

- Assuming characters are independent, generate new text by “coin-flipping”.



Markov Models: Text

But each character *is not* independent of previous characters!

A k^{th} -order Markov Model:

- Assigns probabilities to a character's occurrence that depends on the previous k characters.



Markov Models: Text

Assume we have input text with:

- 100 occurrences of *th*
 - » 50 of which followed by *e* (*the*, *then*, etc.)
 - » 25 of which followed by *i* (*this*, *thin*, etc.)
 - » 20 of which followed by *a* (*that*, *thank*, etc.)
 - » 5 of which followed by *o* (*though*, *thorn*, etc.)

2nd-order Markov model predicts that:

$$P(e|th) = \frac{1}{2} \quad P(i|th) = \frac{1}{4} \quad P(a|th) = \frac{1}{5} \quad P(o|th) = \frac{1}{20}$$

Given a probabilistic model and seed, we can generate new text!



Markov Models: Text

Snippet of original text: “As You like it” by Shakespeare:

DUKE SENIOR:

Now, my co-mates and brothers in exile,
Hath not old custom made this life more sweet
Than that of painted pomp? Are not these woods
More free from peril than the envious court?
Here feel we but the penalty of Adam,
The seasons' difference, as the icy fang
And churlish chiding of the winter's wind,
Which, when it bites and blows upon my body,
Even till I shrink with cold, I smile and say
'This is no flattery: these are counsellors
That feelingly persuade me what I am.'



Markov Models: Text

Snippet of generated text with 6th-order Markov Model:

DUKE SENIOR:

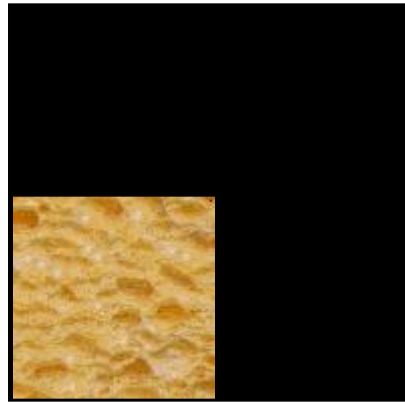
Now, my co-mates and thus bolden'd, man, how now,
monsieur Jaques, Unclaim'd of his absence, as the holly!
Though in the slightest for the fashion of his absence, as
the only wear.



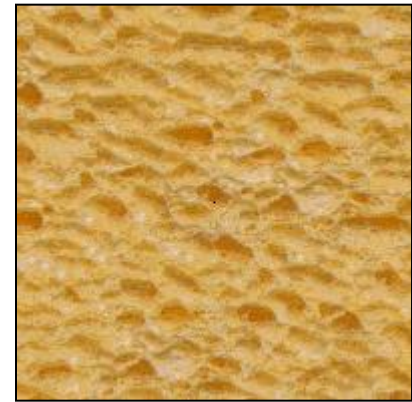
Markov Models: Images



Use this as
original "text"



and this as seed



to get this result!

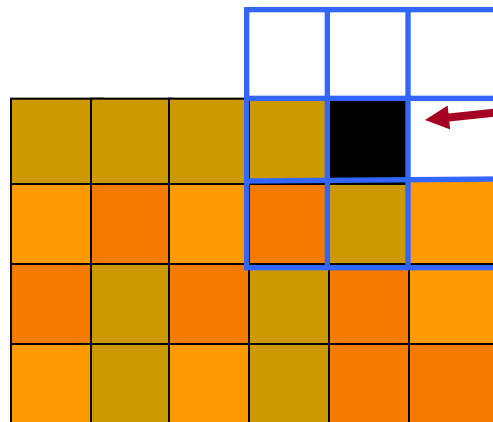


Figure out values
of new pixels
based on
surrounding
known pixels



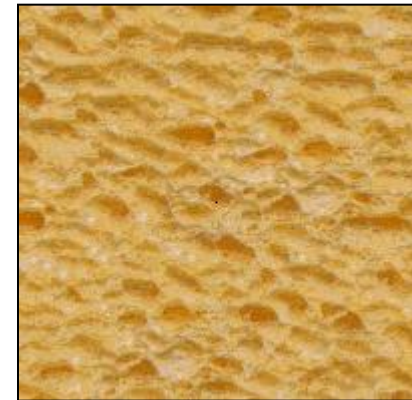
Markov Models: Images



Use this as
original “text”



and this as seed



to get this result!

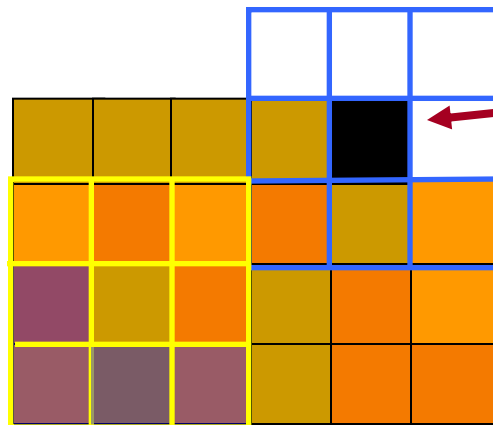


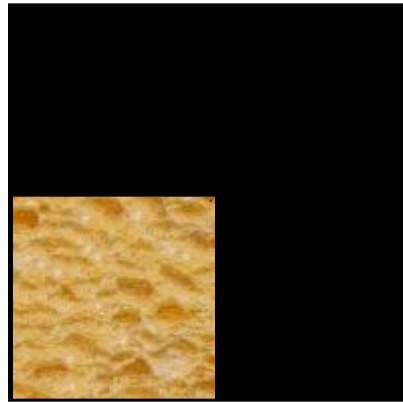
Figure out values
of new pixels
based on
surrounding
known pixels



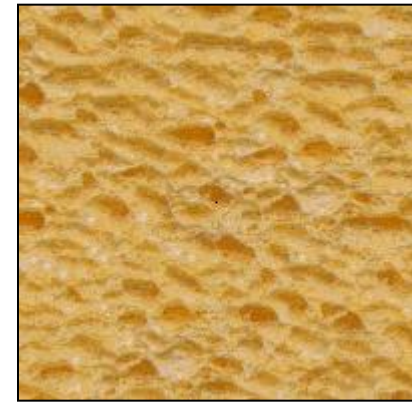
Markov Models: Images



Use this as
original "text"



and this as seed



to get this result!

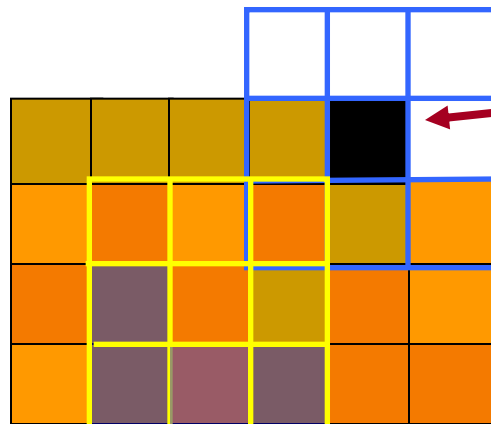


Figure out values
of new pixels
based on
surrounding
known pixels



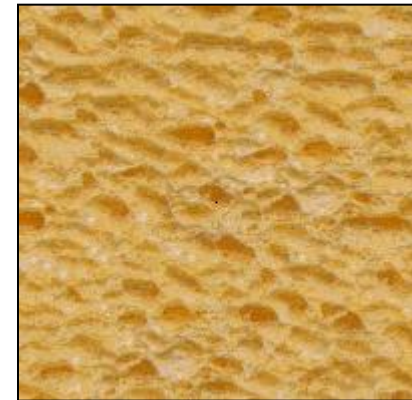
Markov Models: Images



Use this as
original “text”



and this as seed



to get this result!

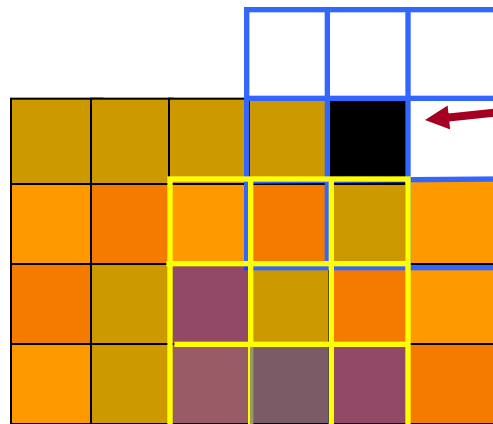


Figure out values
of new pixels
based on
surrounding
known pixels



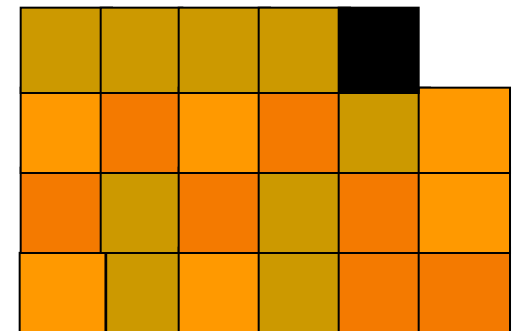
Markov Models: Images

Problems:

Not likely to find one, let alone multiple, exact matches

Solution:

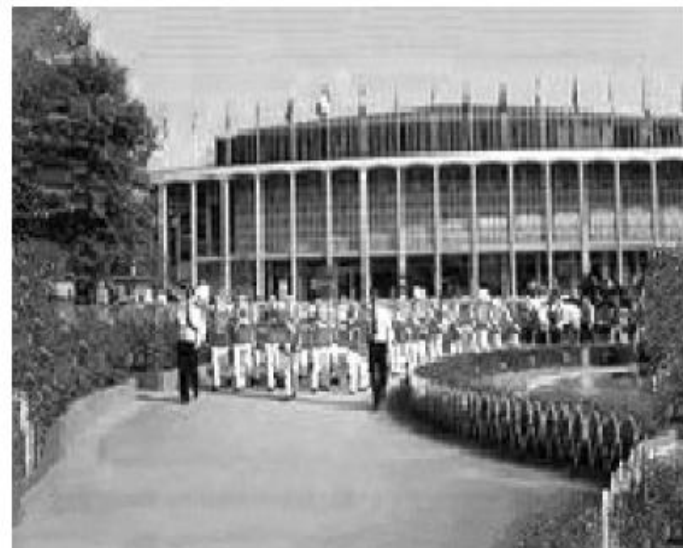
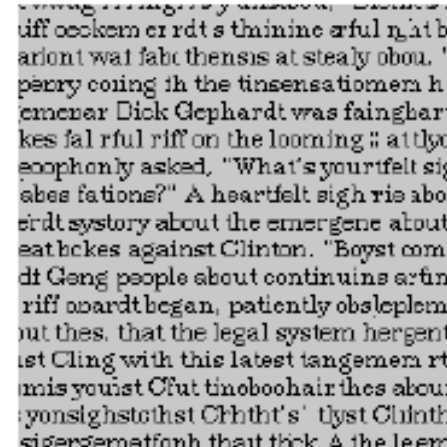
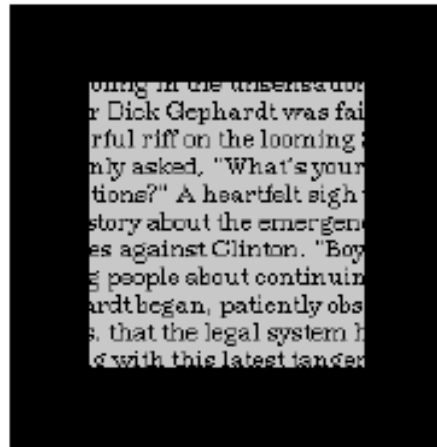
Randomly choose among best N matches with probability based on match quality





Markov Models: Images

Examples:





Markov Models: Images

Pros:

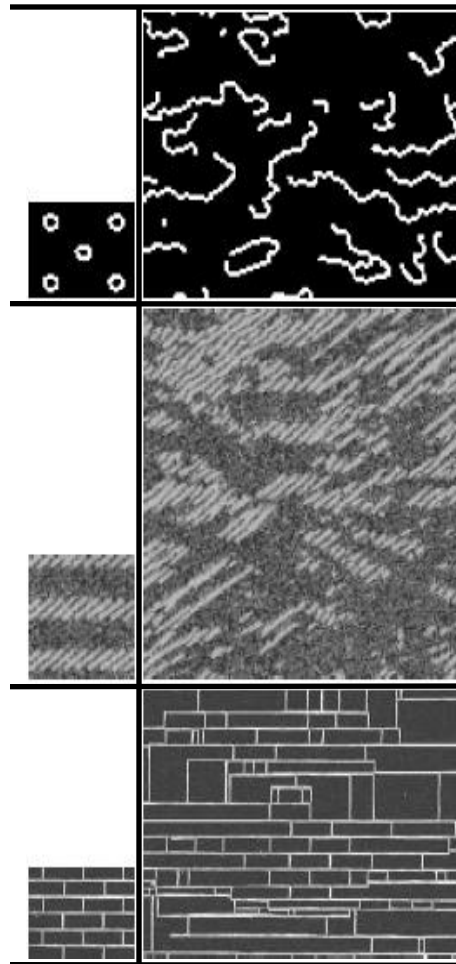
- ✓ Conceptually simple/sound
- ✓ Often produces good results
- ✓ Never chooses a pixel/color NOT found in source

Cons:

- ✗ Need to choose correct window size



Markov Models: Images

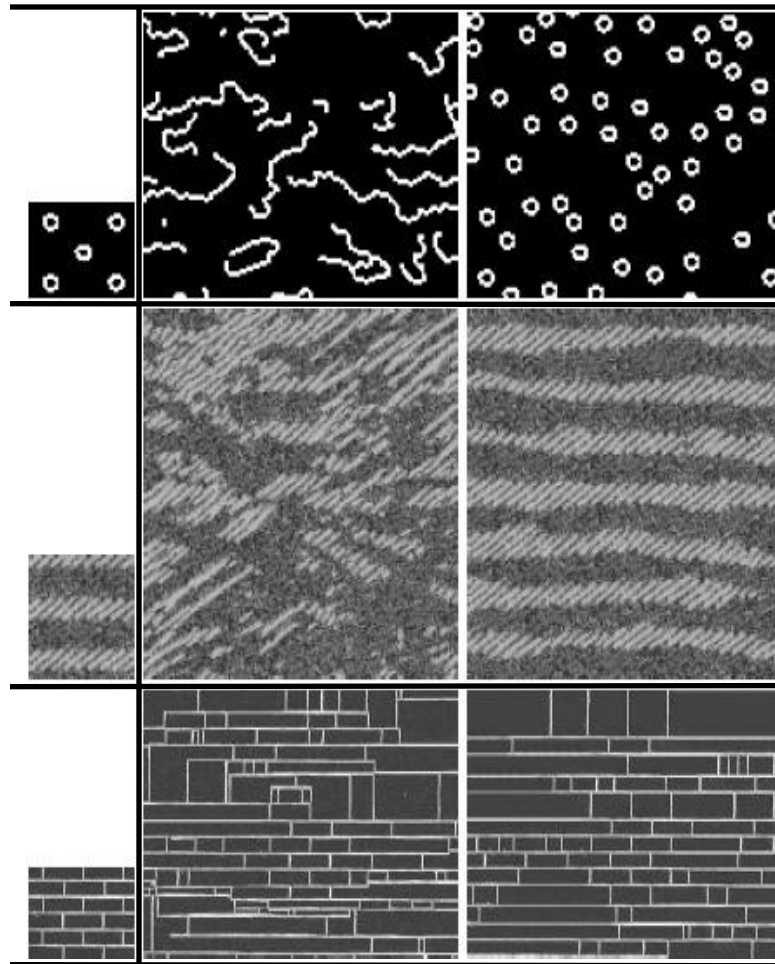


Increasing window size 

Courtesy Alexei Efros



Markov Models: Images

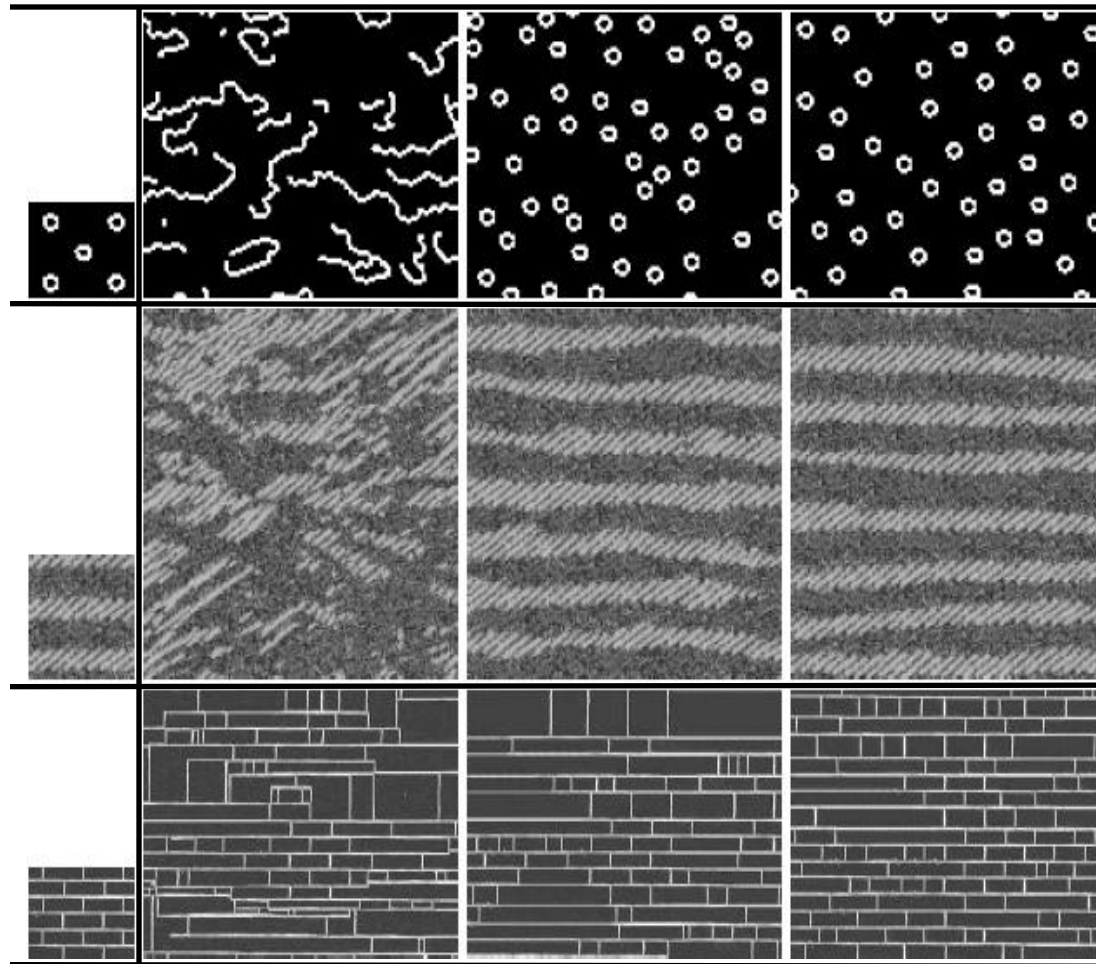


Increasing window size →

Courtesy Alexei Efros



Markov Models: Images

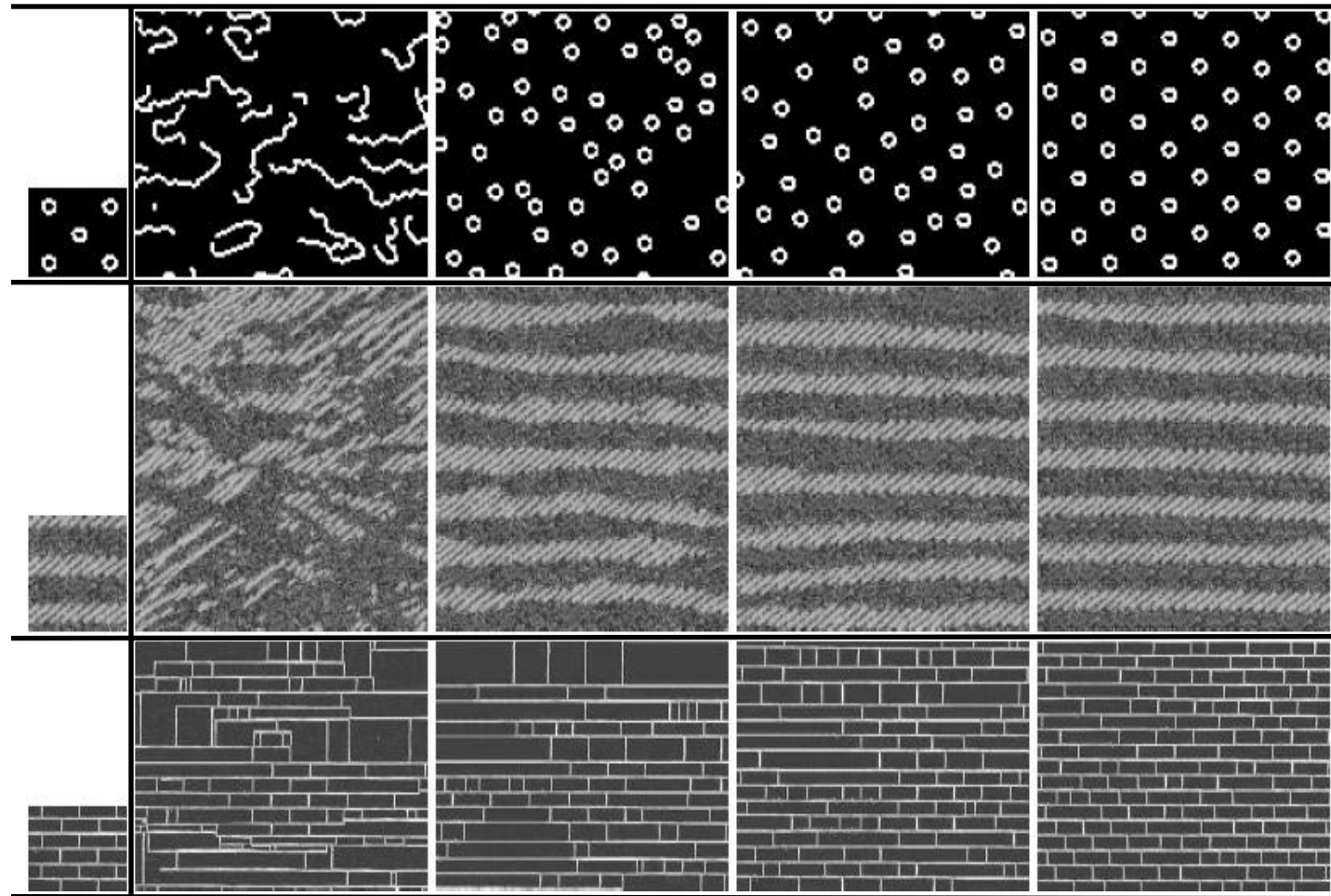


Increasing window size →

Courtesy Alexei Efros



Markov Models: Images



Increasing window size →

Courtesy Alexei Efros



Markov Models: Images

Pros:

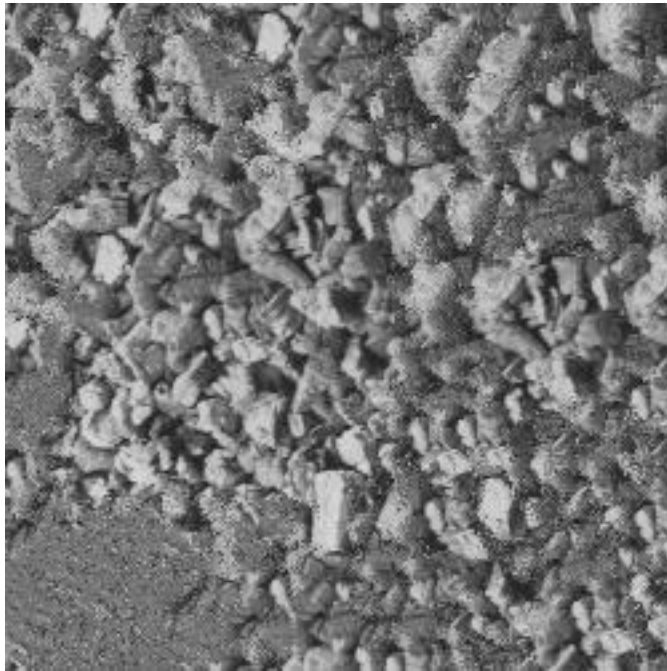
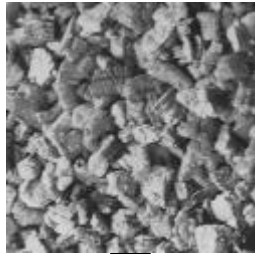
- ✓ Conceptually simple/sound
- ✓ Often produces good results
- ✓ Never chooses a pixel/color NOT found in source

Cons:

- ✗ Need to choose correct window size
- ✗ Slow! (increasing window size makes this worse)
 - » See [Barnes, '09] for acceleration techniques



Markov Models: Images



Growing garbage



Verbatim copying

Courtesy Alexei Efros



Markov Models: Images

Pros:

- ✓ Conceptually simple/sound
- ✓ Often produces good results
- ✓ Never chooses a pixel/color NOT found in source

Cons:

- ✗ Need to choose correct window size
- ✗ Very slow! (increasing window size makes this worse)
- ✗ Doesn't always work (can get stuck in a rut)
- ✗ **The size of the output is proportional to the size of the output**

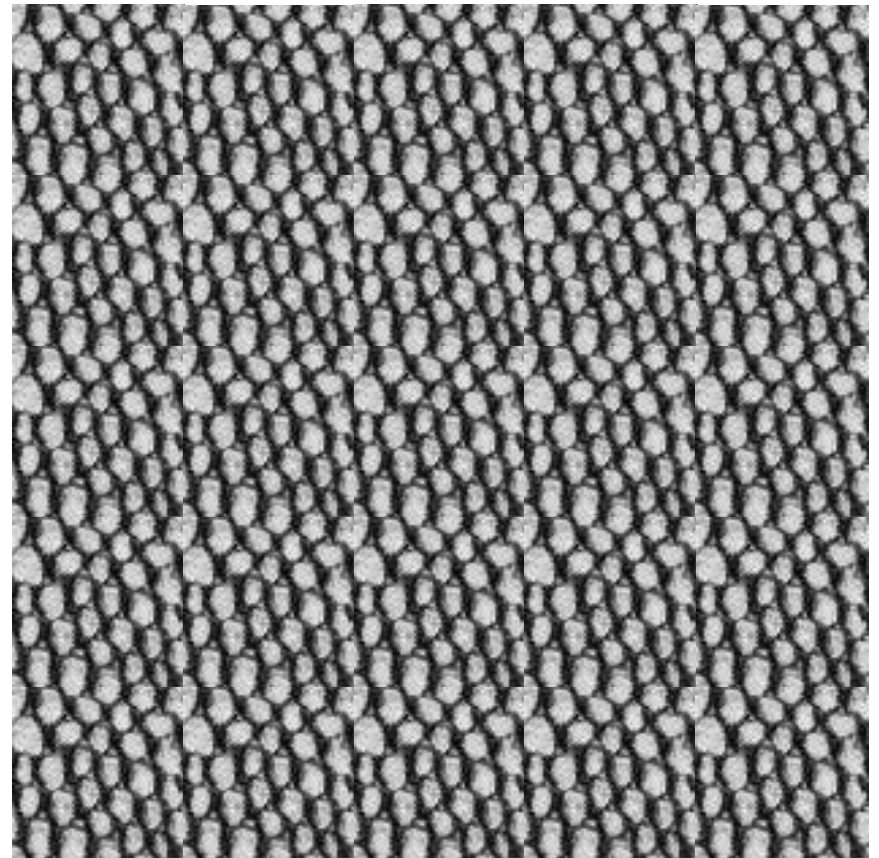
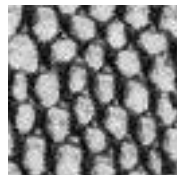


Wang Tiles

Can we use a small amount of texture memory to generate large textures?

Tiling:

- discontinuities
- repetitive

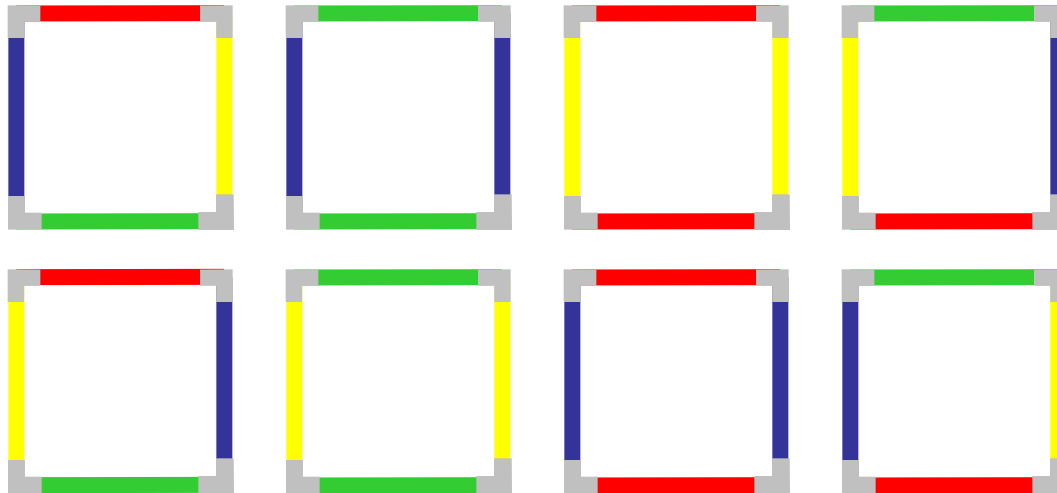




Wang Tiles

Key Idea:

Given a set of colors, and a sufficiently large set of square tiles whose edges are marked with one of these colors....:

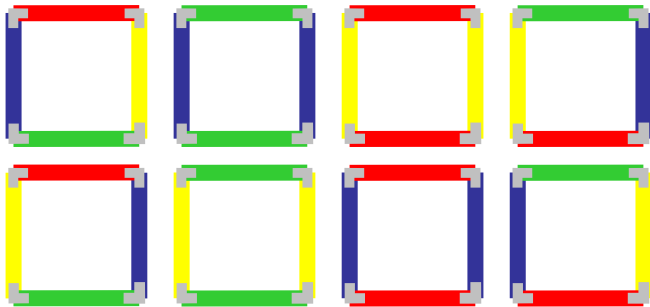




Wang Tiles

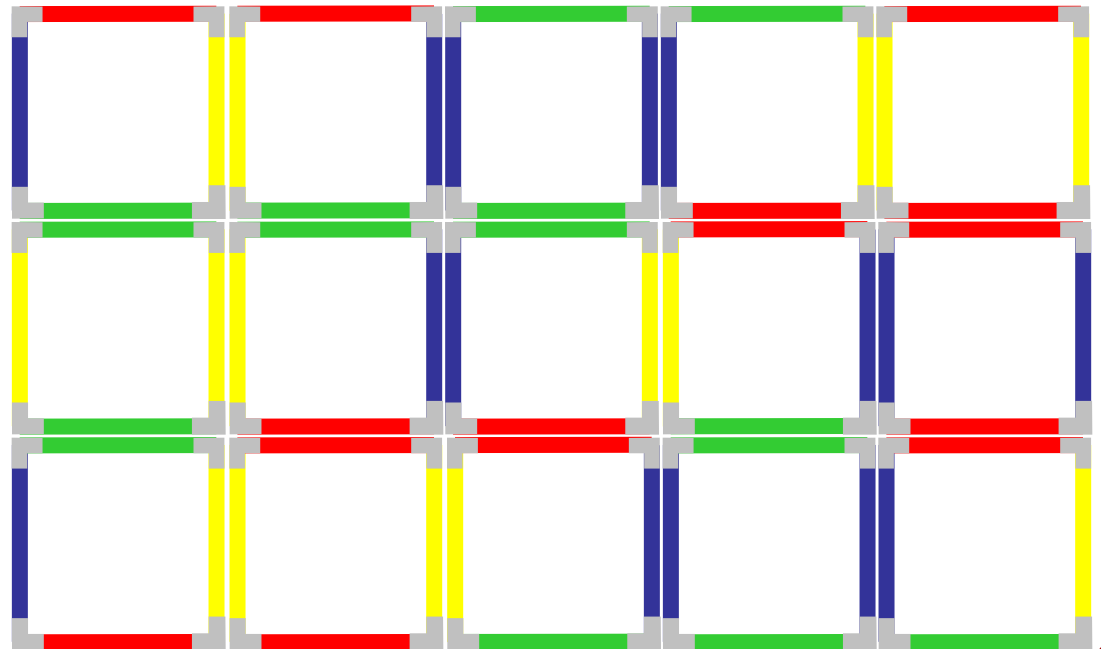
Key Idea:

...the plane can be tiled w/ edge-matching squares:



Base Tiles

Tiled Image

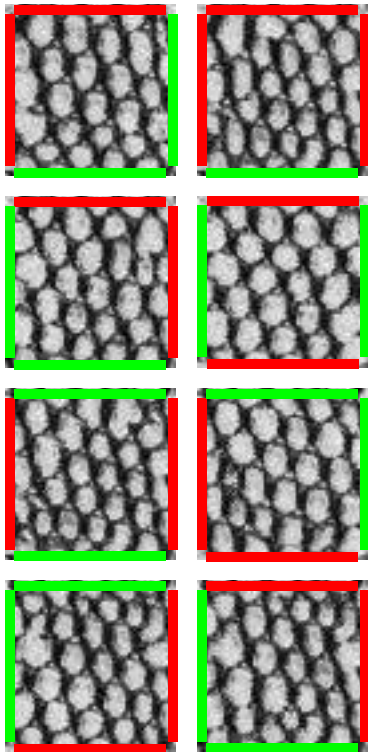




How Wang Tile Works

Application:

- Associate a single texture to each tile



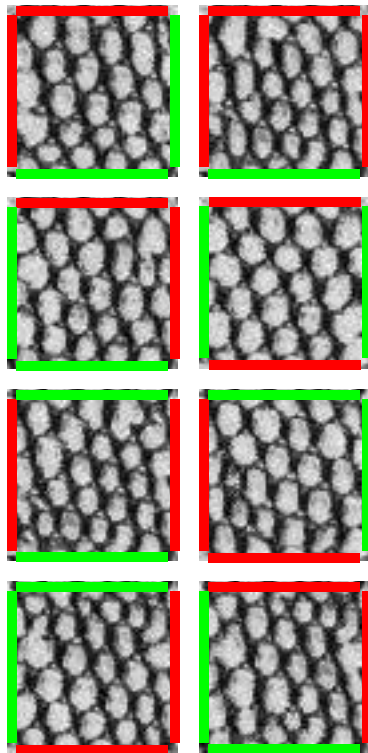
Input tiles



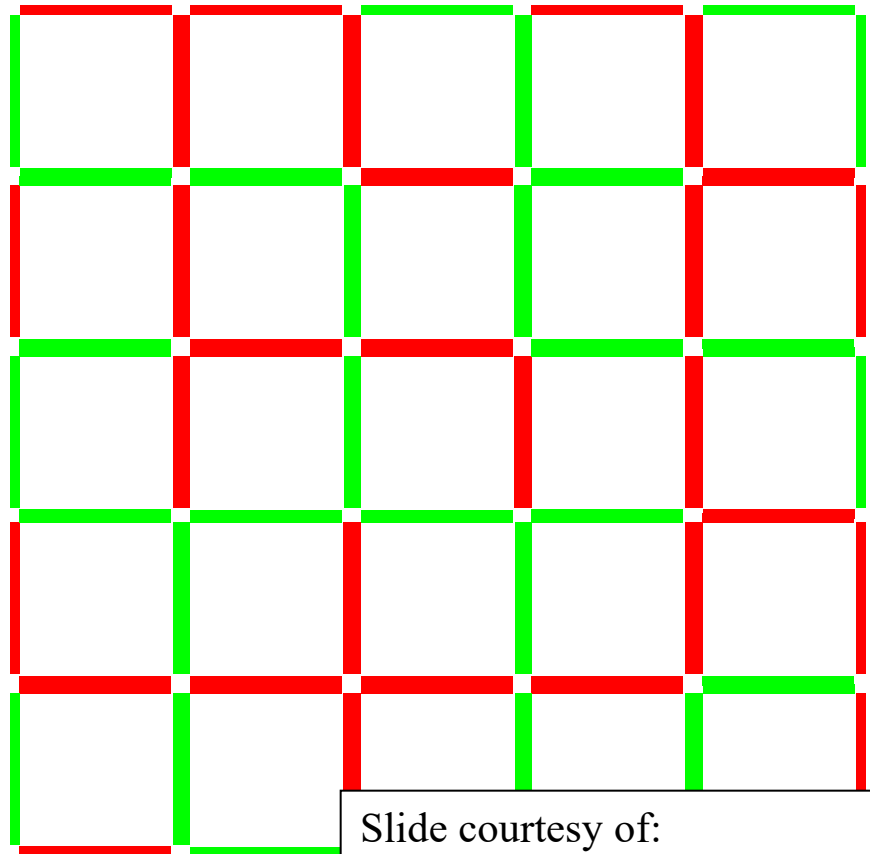
How Wang Tile Works

Application:

- Associate a texture to each tile
- Given a Wang tiling of the plane we get a new texture



Input tiles



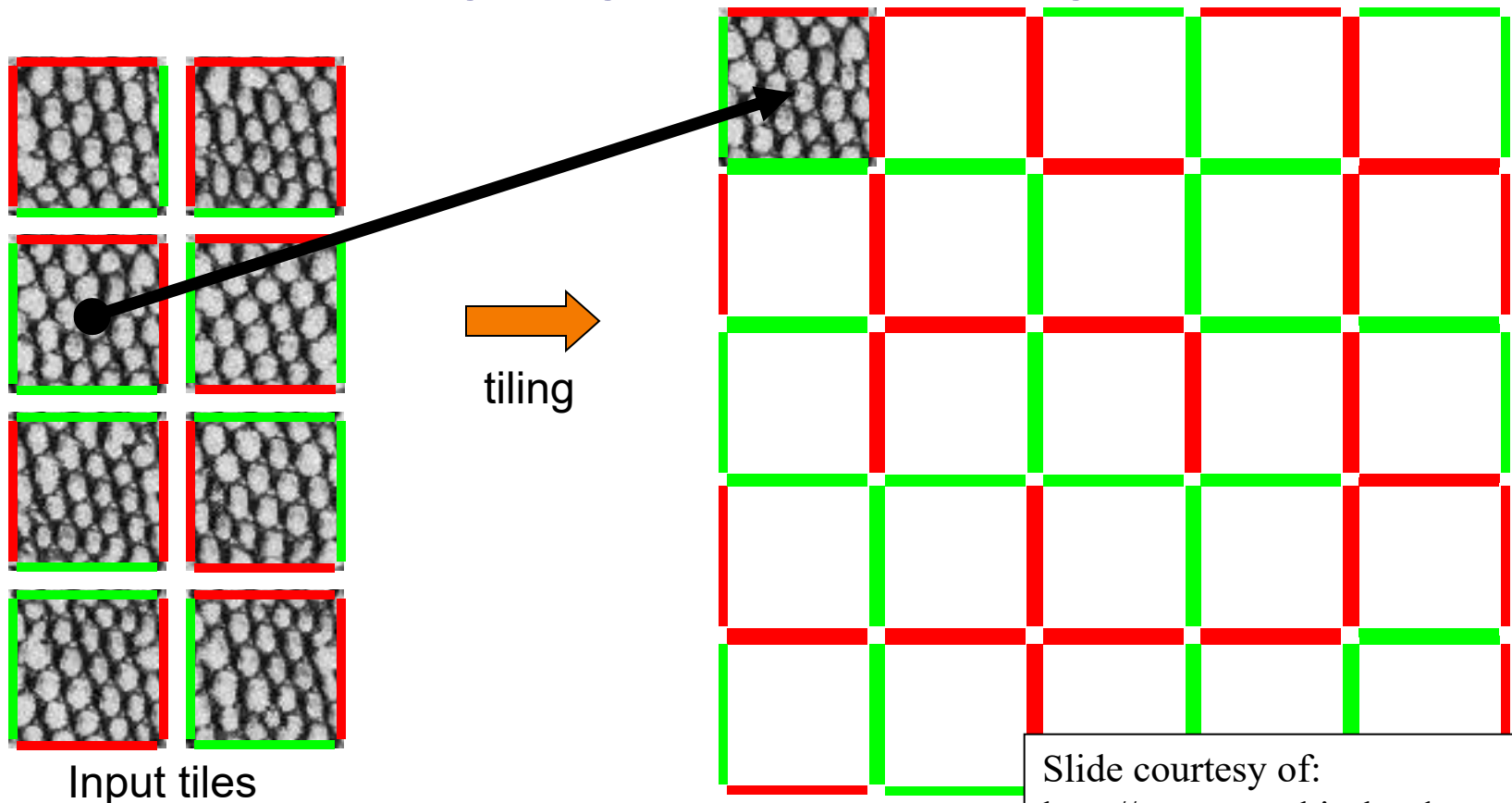
Slide courtesy of:
<http://www.graphicshardware.org>



How Wang Tile Works

Application:

- Associate a texture to each tile
- Given a Wang tiling of the plane we get a new texture

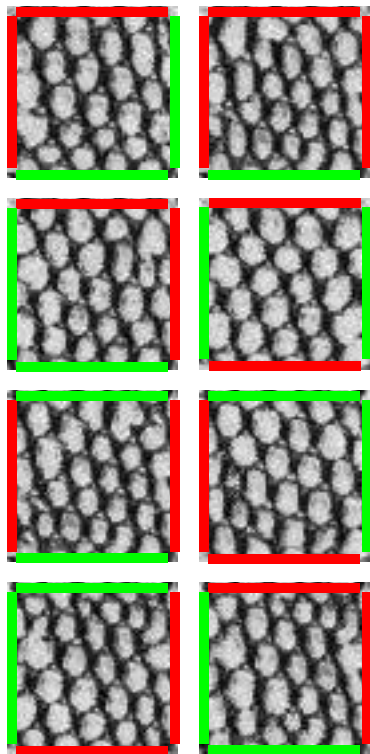




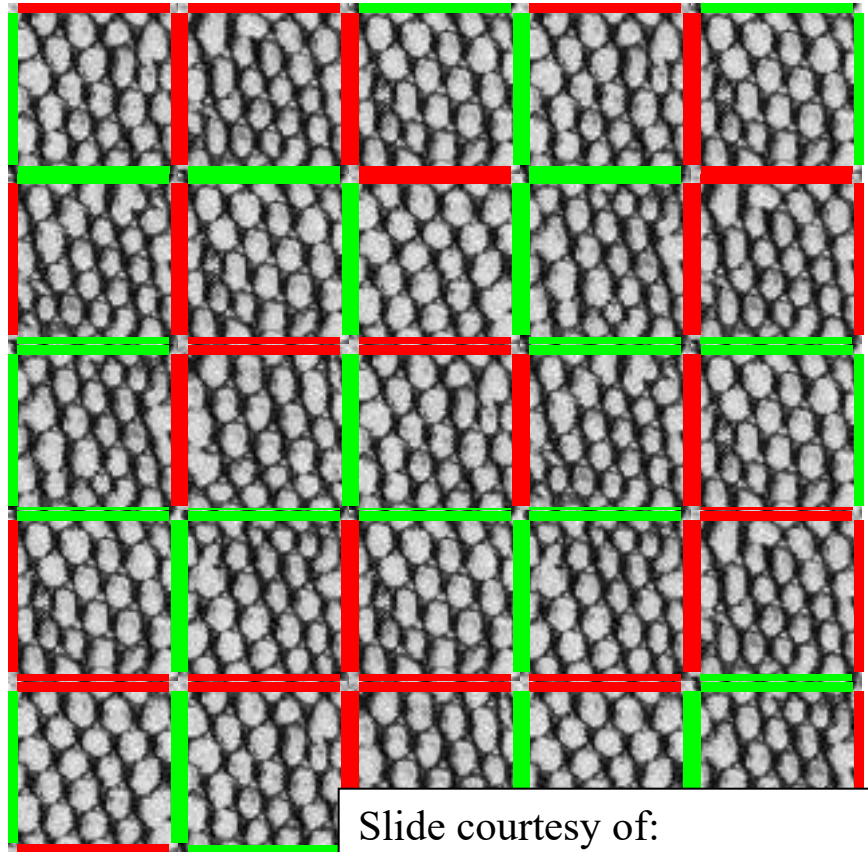
How Wang Tile Works

Application:

- Associate a texture to each tile
- Given a Wang tiling of the plane we get a new texture



Input tiles



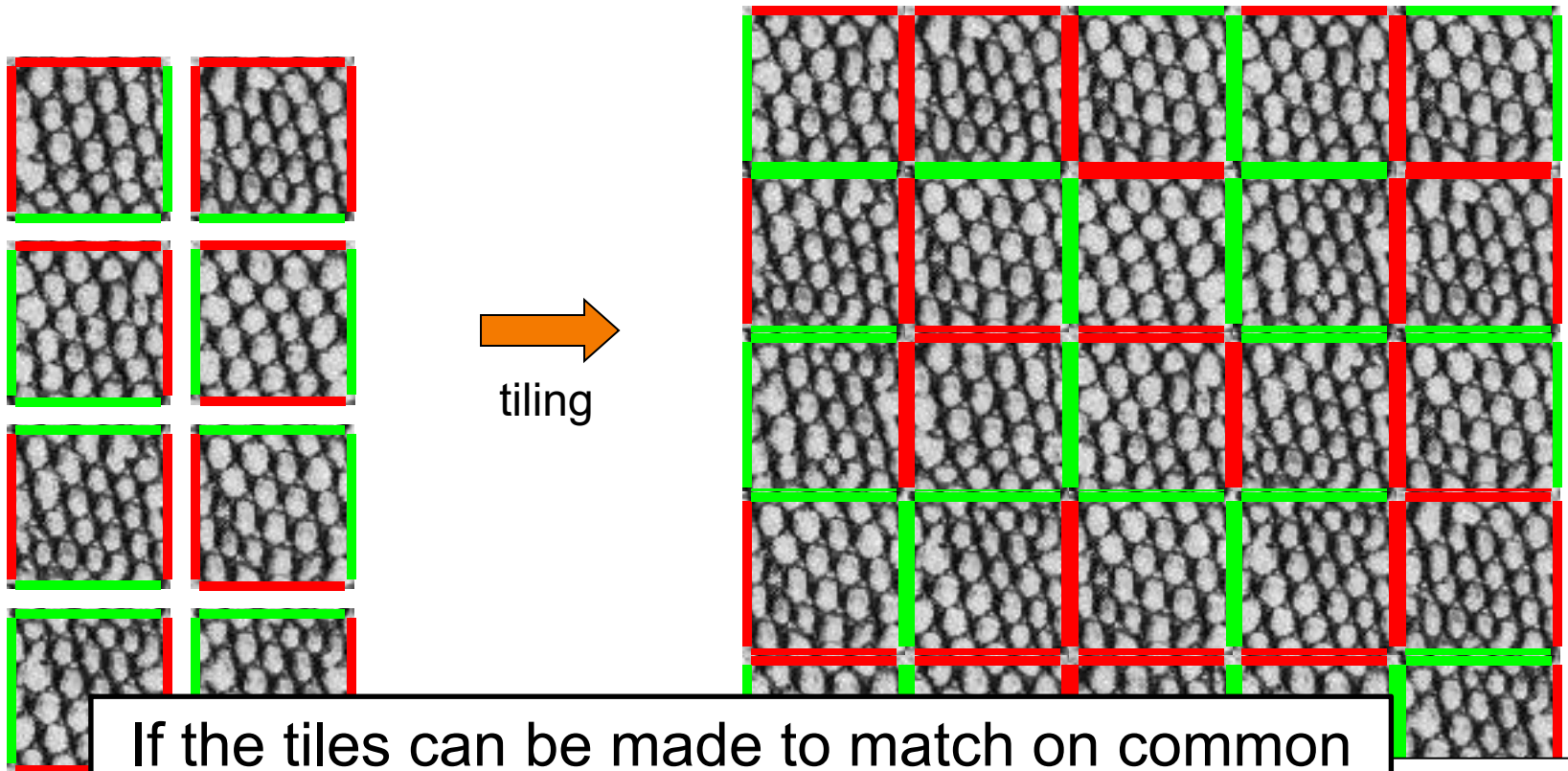
Slide courtesy of:
<http://www.graphicshardware.org>



How Wang Tile Works

Application:

- Associate a texture to each tile
- Given a Wang tiling of the plane we get a new texture



If the tiles can be made to match on common color edges, the texture will be seamless.



Wang Tiles

Tile Complexity:

For the texture not to appear repetitive, we need to have (random) choice in choosing tiles.

How many tiles do we need, assuming k different colors on the edges?

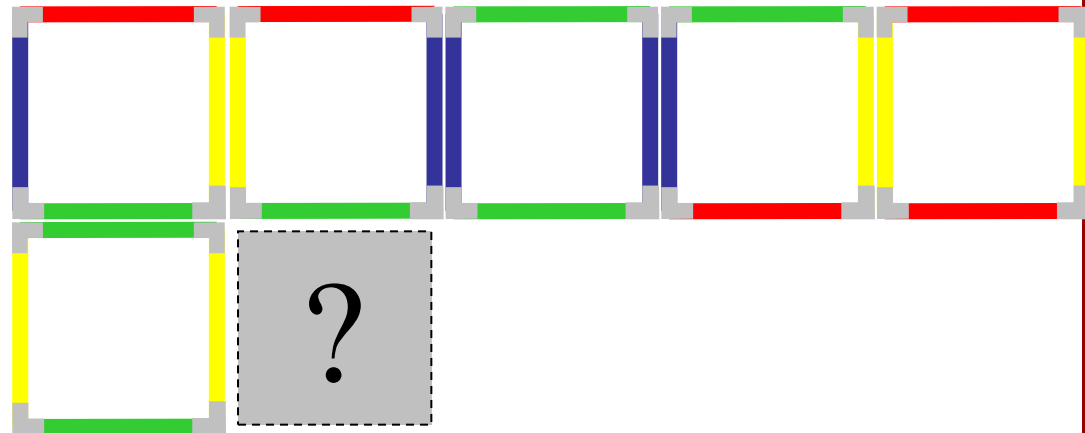


Wang Tiles

Tile Complexity:

In general, we have two restrictions when we introduce a new tile – the colors of the West and North edges.

Tiled Image





Wang Tiles

Tile Complexity:

In general, we have two restrictions when we introduce a new tile – the colors of the West and North edges.

- ⇒ For k colors, we need to have at least k^2 tiles to be able to find one that will fit.
- ⇒ To be able to make a (random) choice each time, we need to have at least $2k^2$ tiles.

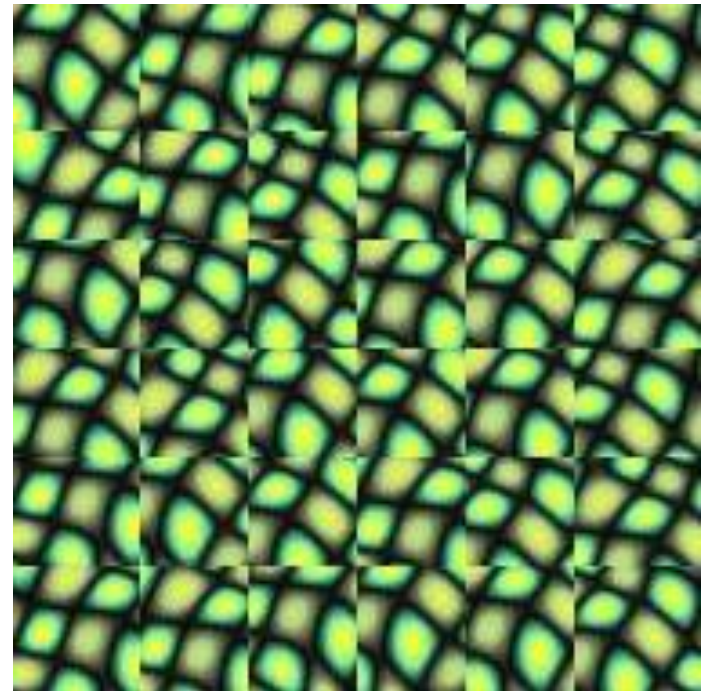


Wang Tiles

Tile Generation:

To generate seamless textures, tiles must match on common color edges.

Otherwise get apparent discontinuity seams





Wang Tiles

Tile Generation:

- Associate a source diamond to each colored edge (with diagonal slightly larger than the edge length)



Source



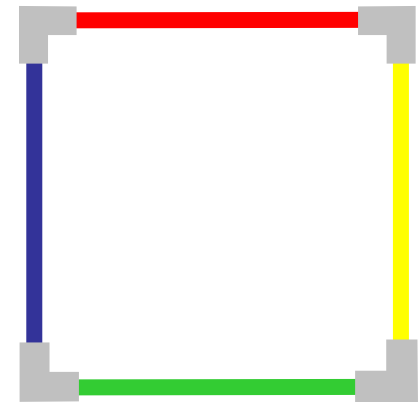
Wang Tiles

Tile Generation:

- Associate a source diamond to each colored edge (with diagonal slightly larger than the edge length)
- Given a tile, paste the diamonds onto the edges



Source





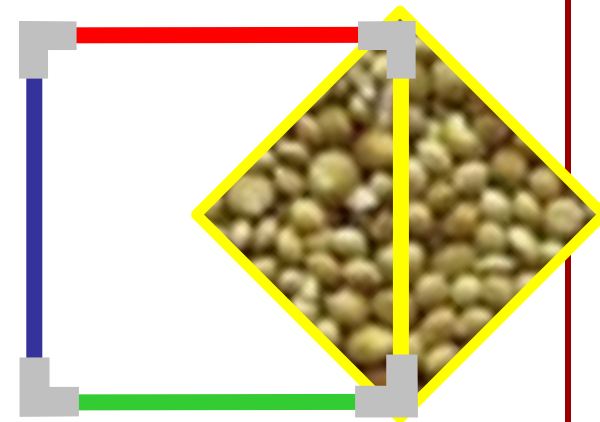
Wang Tiles

Tile Generation:

- Associate a source diamond to each colored edge (with diagonal slightly larger than the edge length)
- Given a tile, paste the diamonds onto the edges



Source





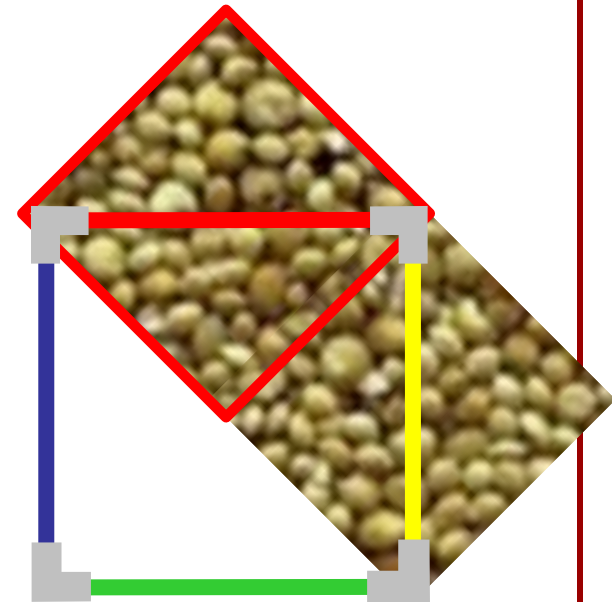
Wang Tiles

Tile Generation:

- Associate a source diamond to each colored edge (with diagonal slightly larger than the edge length)
- Given a tile, paste the diamonds onto the edges



Source





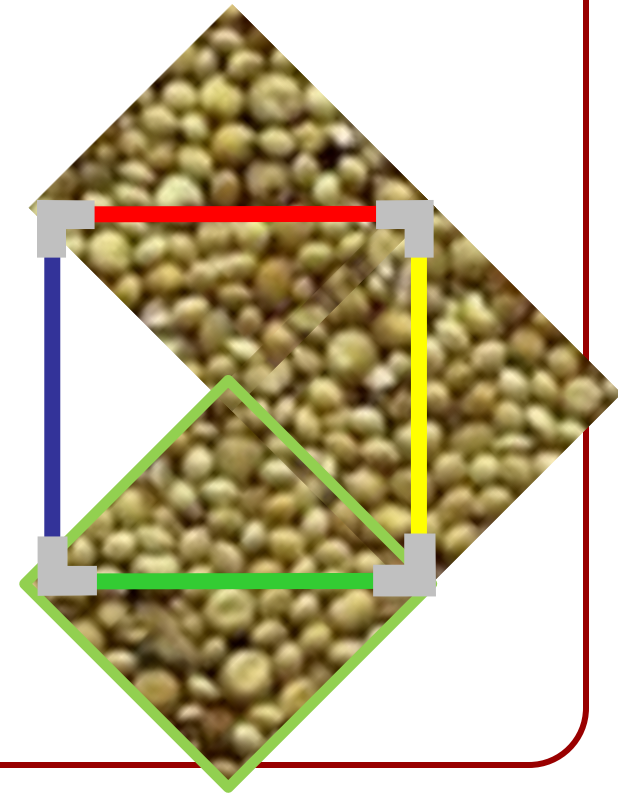
Wang Tiles

Tile Generation:

- Associate a source diamond to each colored edge (with diagonal slightly larger than the edge length)
- Given a tile, paste the diamonds onto the edges



Source





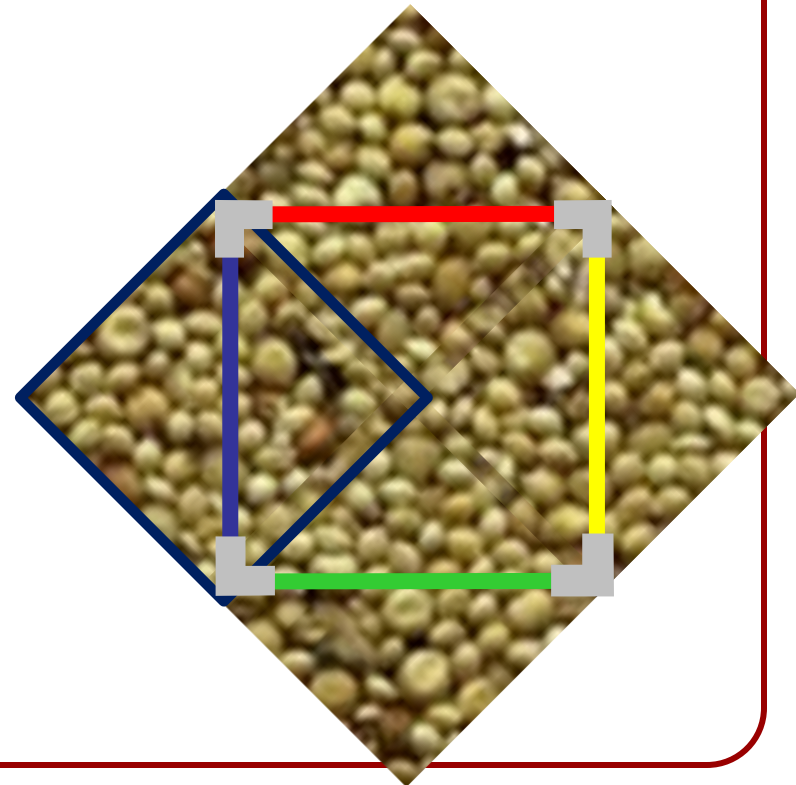
Wang Tiles

Tile Generation:

- Associate a source diamond to each colored edge (with diagonal slightly larger than the edge length)
- Given a tile, paste the diamonds onto the edges



Source





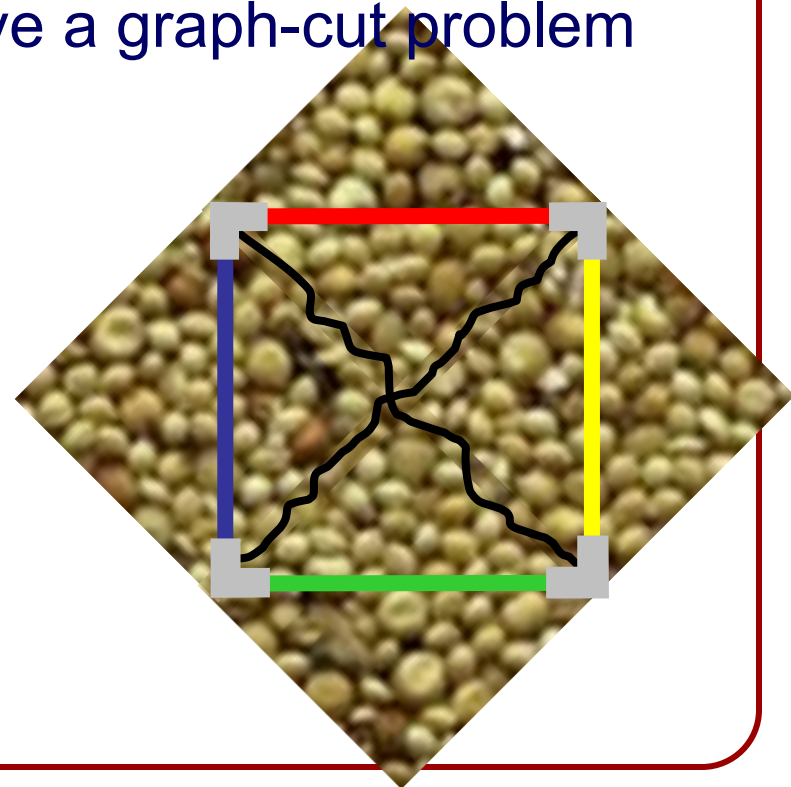
Wang Tiles

Tile Generation:

- Associate a source diamond to each colored edge (with diagonal slightly larger than the edge length)
- Given a tile, paste the diamonds onto the edges
- Quilt the overlap region $\Rightarrow$ solve a graph-cut problem



Source



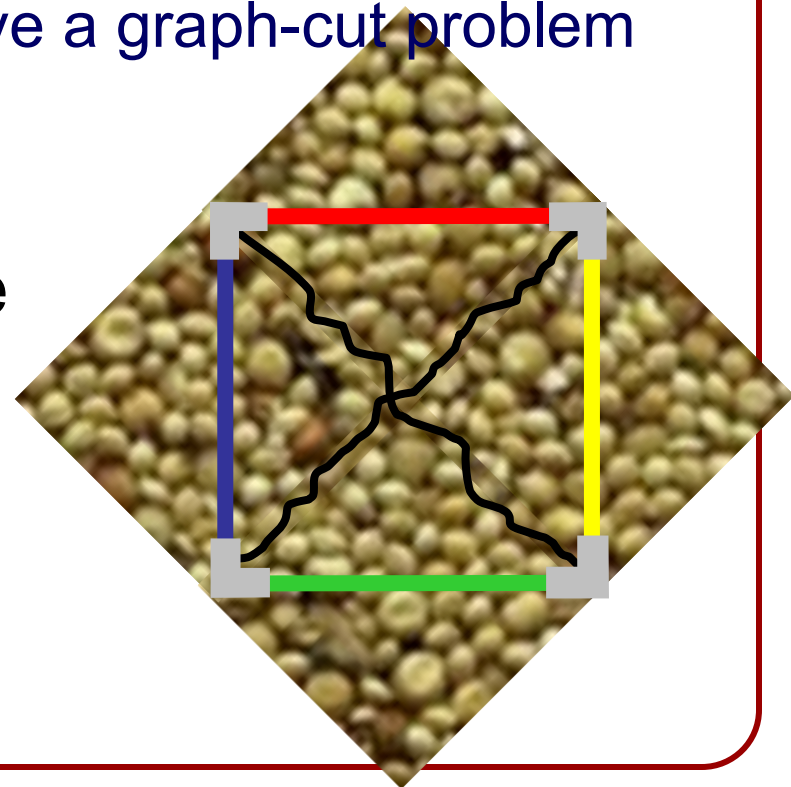


Wang Tiles

Tile Generation:

- Associate a source diamond to each colored edge (with diagonal slightly larger than the edge length)
- Given a tile, paste the diamonds onto the edges
- Quilt the overlap region $\Rightarrow$ solve a graph-cut problem

Since the two-sides of an edge come from the same diamond, they are guaranteed to meet seamlessly!



Outline

- Texture Synthesis
- Midterm Info





Midterm

Content:

Everything that we have covered up to this point:

- Image Processing
- Sampling
- Ray-Casting/Tracing
- Illumination
- Clipping
- Texture Mapping
- Texture Synthesis



Midterm

Format:

- Closed book
- Short answer questions only
- No essays
- No true/false
- No multiple choice
- In person



Midterm

Breakdown:

Six Sections:

- Image Processing
- Sampling
- Ray Tracing
- Illumination
- Texture Mapping
- Miscellany