اصول پردازش تصویر

Principles of Image Processing

مصطفی کمالی تبریزی

۱۰ و ۱۵ آذر ۱۳۹۹

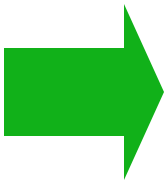
جلسه بیست یکم و دوم

Texture Synthesis

Many slides adopted from Derek Hoiem.

Texture Synthesis

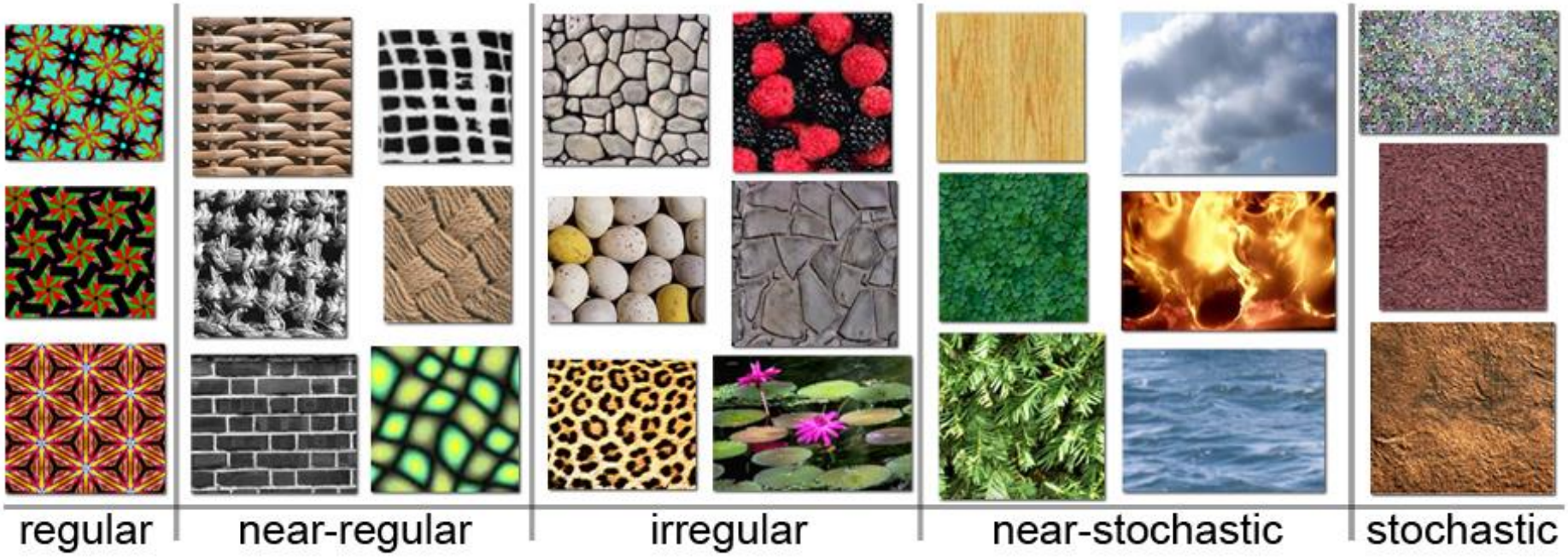
- Goal: create new samples of a given texture
- Many applications: virtual environments, hole-filling, texturing surfaces, …



resize

texture synthesis

The Challenge



Need to model the whole spectrum:

from repeated to stochastic texture

One Idea: Build Probability Distributions

Basic idea

1. Compute statistics of input texture
   (e.g., histogram of edge filter responses)
2. Generate a new texture that keeps those same statistics



- D. J. Heeger and J. R. Bergen. (*SIGGRAPH 1995*)
  Pyramid-based texture analysis/synthesis.
- E. P. Simoncelli and J. Portilla. (*ICIP 1998*)
  Texture characterization via joint statistics of wavelet coefficient magnitudes.

One Idea: Build Probability Distributions

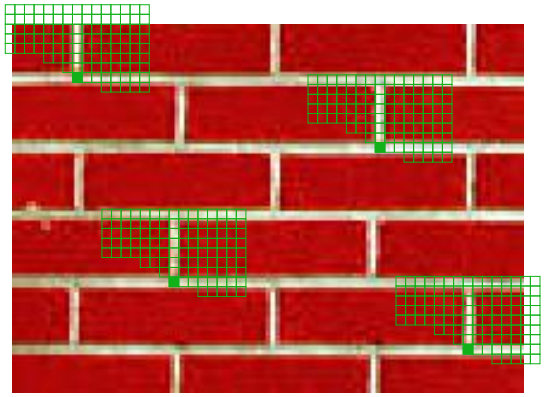
But it (usually) doesn't work

- Probability distributions are hard to model well

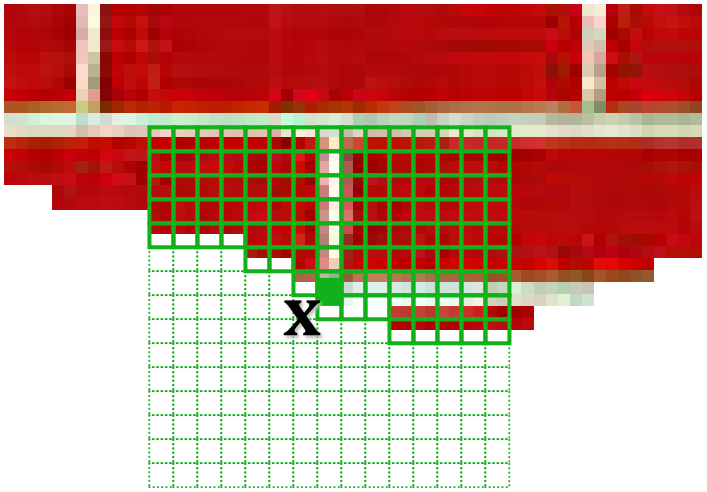
Input

Synthesized

Synthesizing One Pixel



input image



synthesized image

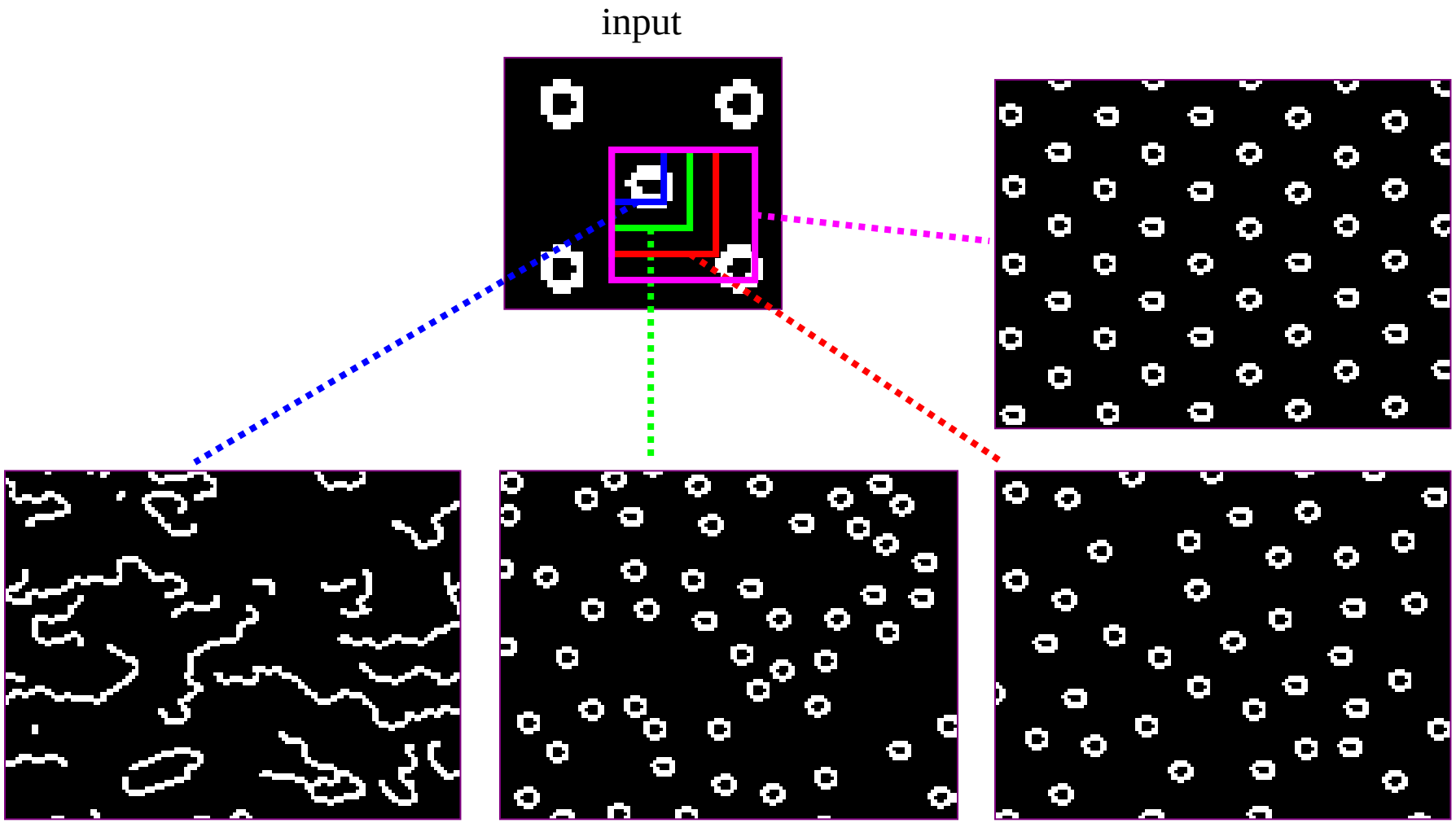
- What is $p(\mathbf{x} \mid neighborood\ of\ pixels\ around\ \mathbf{x})$?
- Find all the windows in the image that match the neighborhood
- To synthesize **x**
  - pick one matching window at random
  - assign **x** to be the center pixel of that window
- An **exact** neighbourhood match might not be present, so find the **best** matches using **SSD error** and randomly choose between them, preferring better matches with higher probability

Alyosha Efros, ICCV 1999

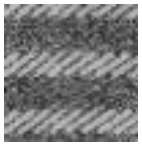
Details

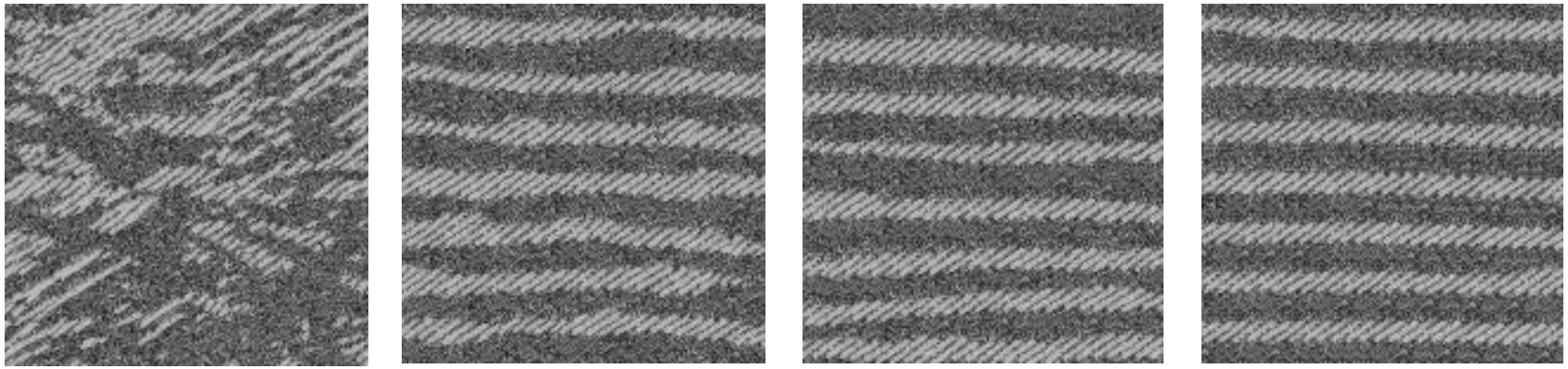
- How to match patches?
  - Gaussian-weighted SSD
    (more emphasis on nearby pixels)
- What order to fill in new pixels?
  - “Onion skin” order: pixels with most neighbors are synthesized first
  - To synthesize from scratch, start with a randomly selected small patch from the source texture
- How big should the patches be?

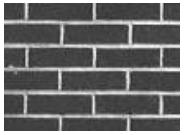
Size of Neighborhood Window

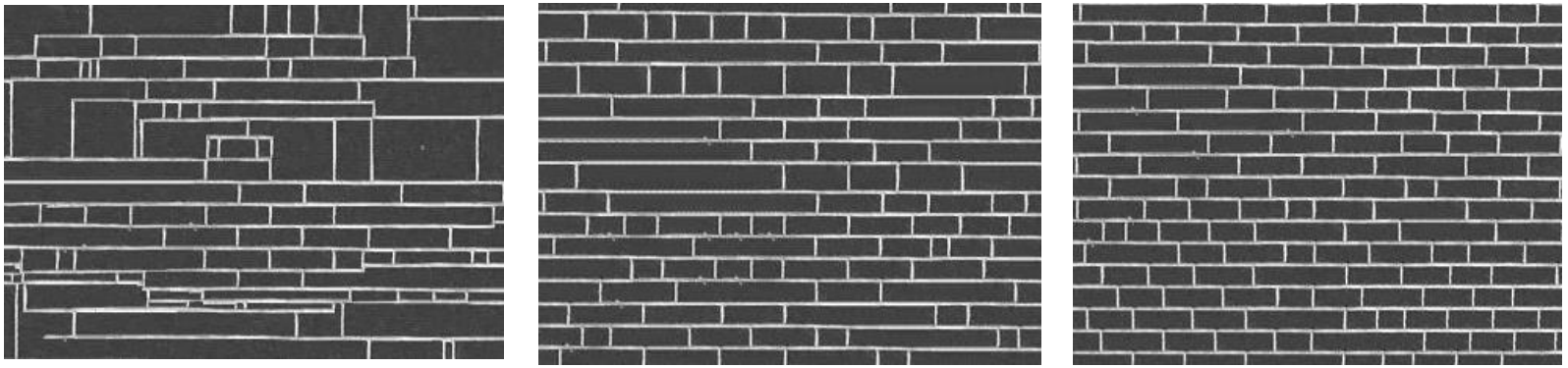


Varying Window Size









Increasing window size

Texture Synthesis Algorithm

- While image not filled

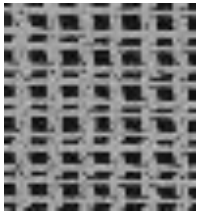
  1. Get unfilled pixels with filled neighbors, sorted by number of filled neighbors

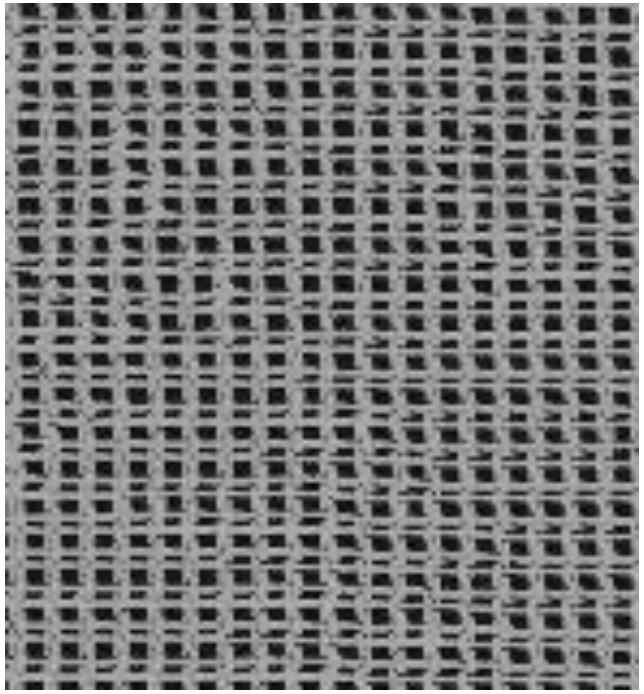
  2. For each pixel, get top n matches based on visible neighbors
     - Patch Distance: Gaussian-weighted SSD

  3. Randomly select one of the matches and copy pixel from it

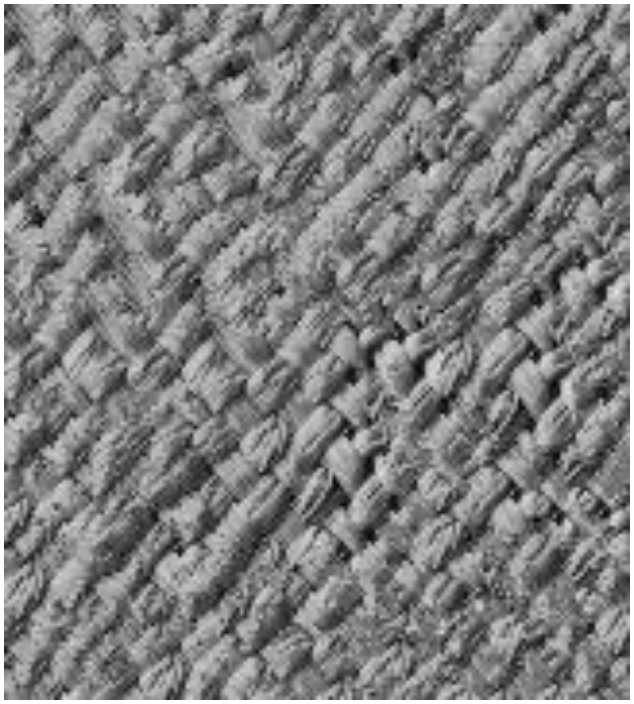
Synthesis Results

French canvas





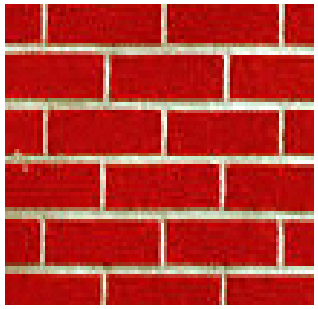
Raffia weave

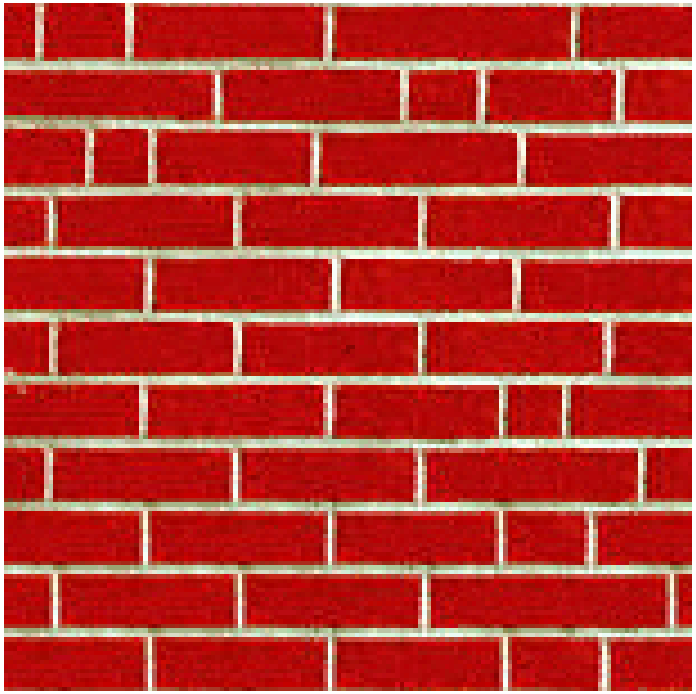


More Results

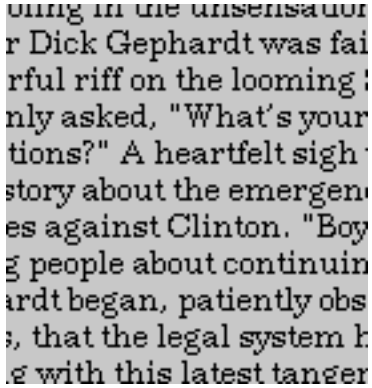
White bread

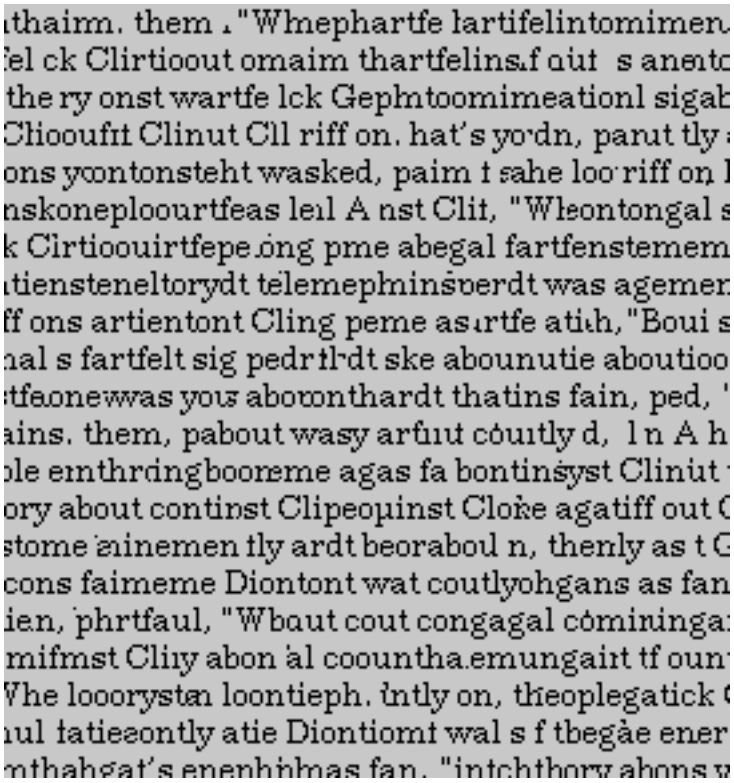
Brick wall



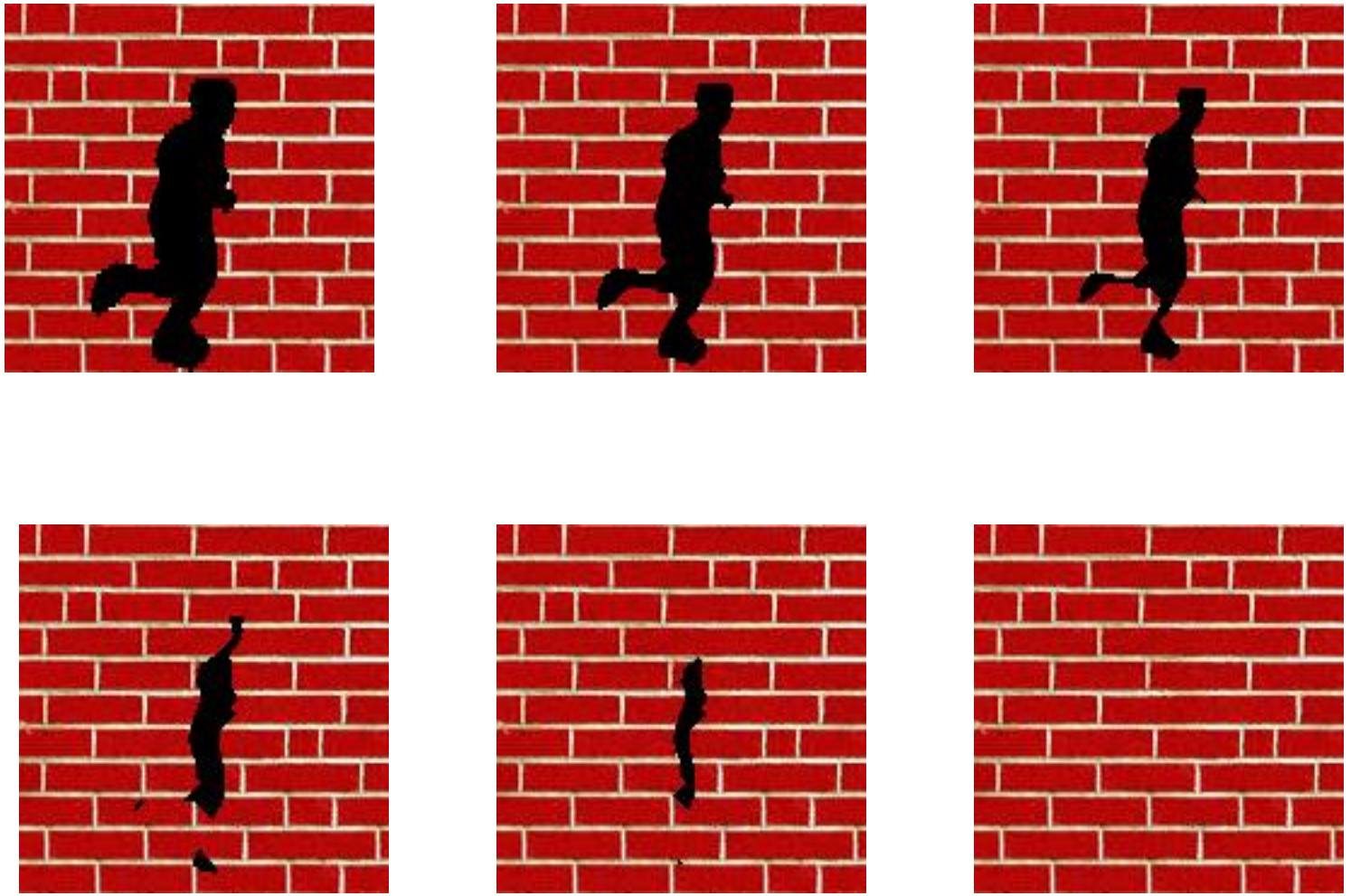


Homage to Shannon

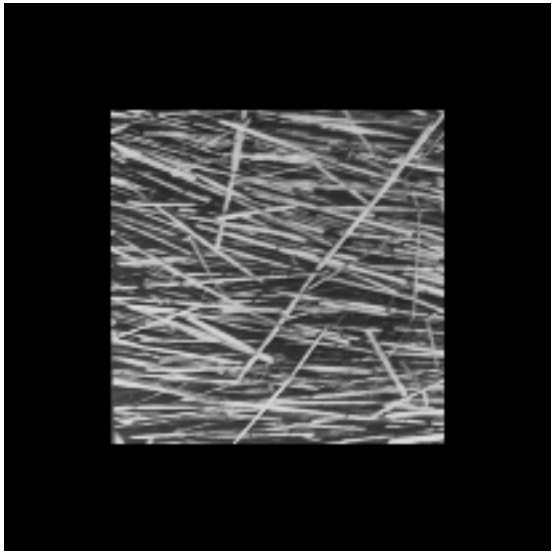


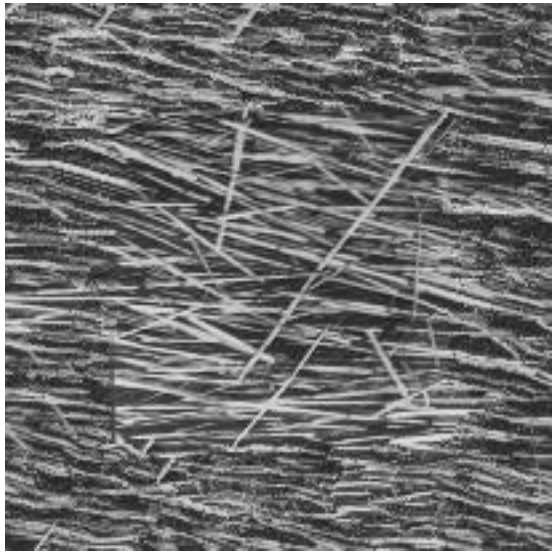


Hole Filling



Extrapolation

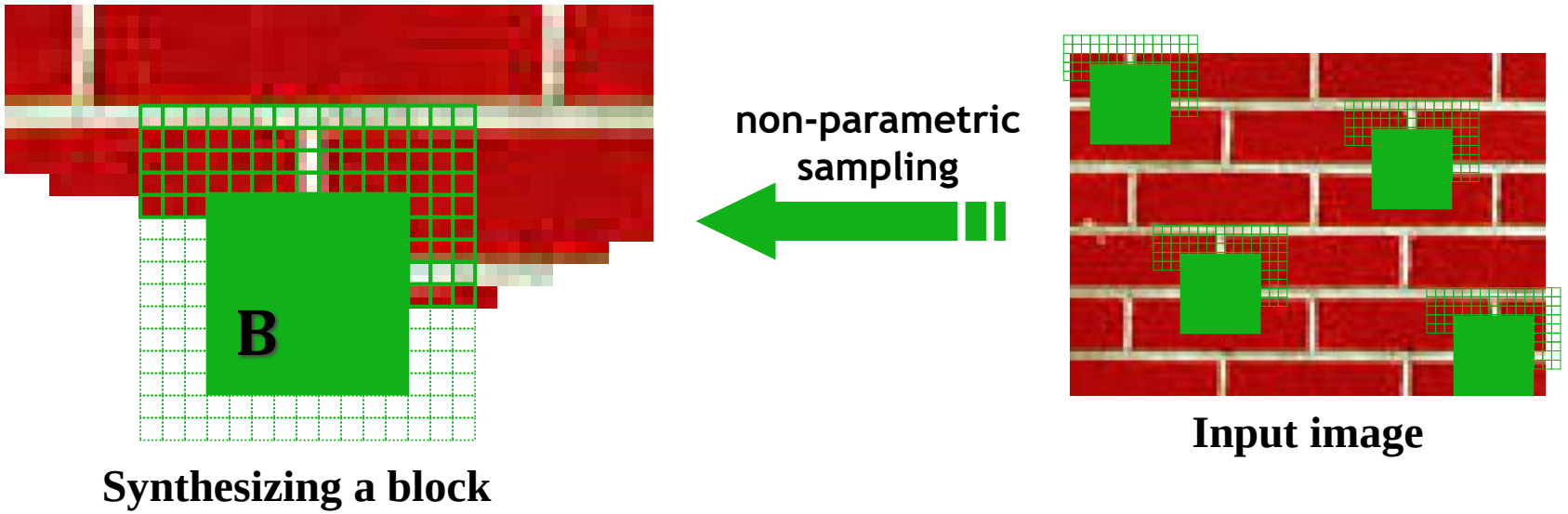




Summary

- The Efros & Leung texture synthesis algorithm
  - Very simple
  - Surprisingly good results
  - Synthesis is easier than analysis!
  - …but very slow

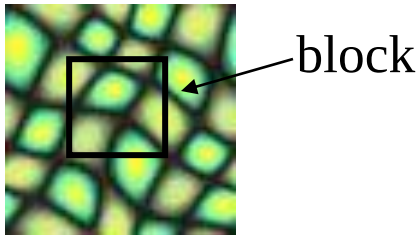
Image Quilting [Efros & Freeman 2001]



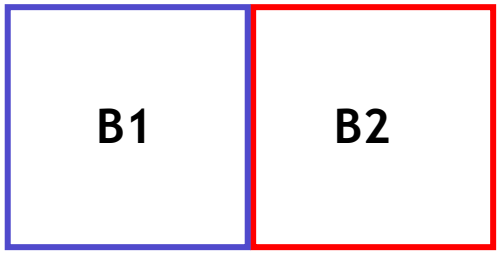
- Observation: neighbor pixels are highly correlated

Idea: unit of synthesis = block

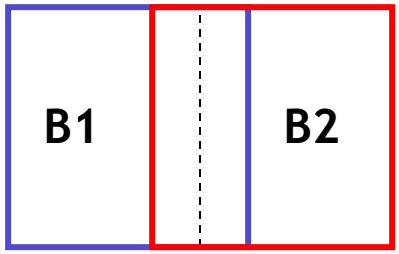
- Exactly the same but now we want $p(\mathbf{B} \mid N(\mathbf{B}))$
- Much faster: synthesize all pixels in a block at once



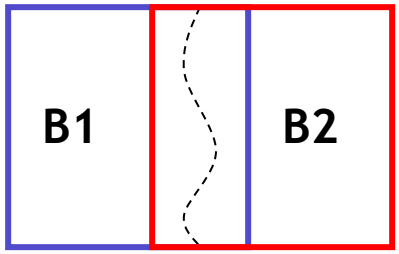
Input texture



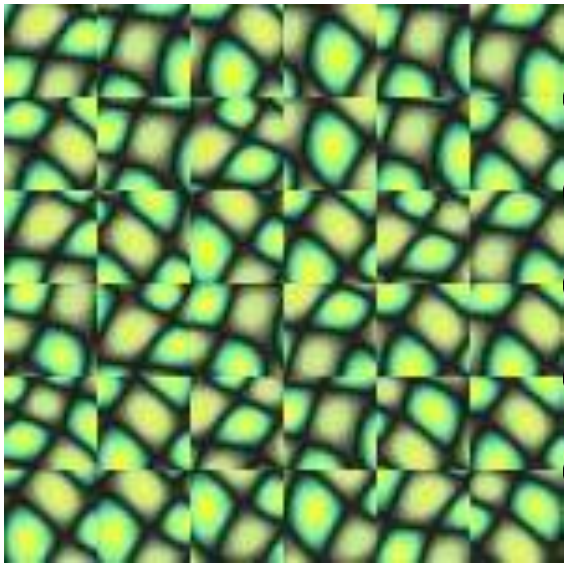
Random placement of blocks

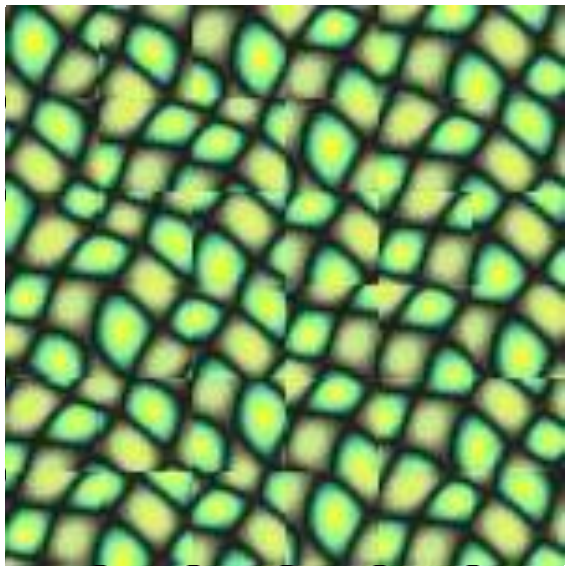


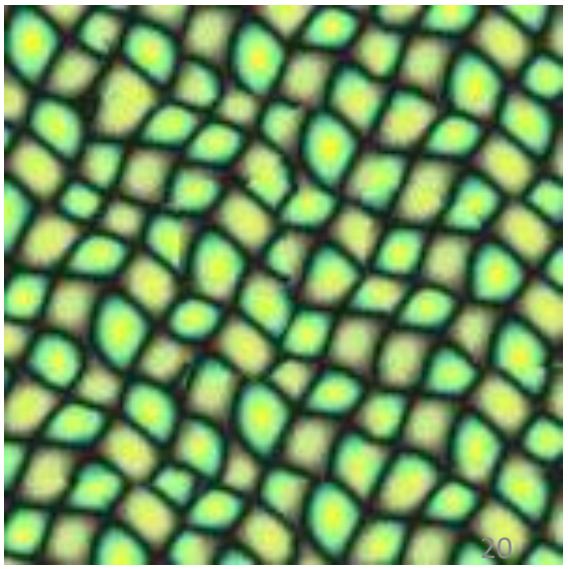
Neighboring blocks constrained by overlap



Minimal error boundary cut

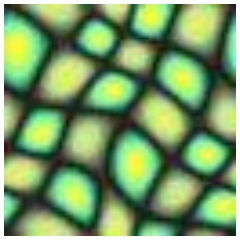


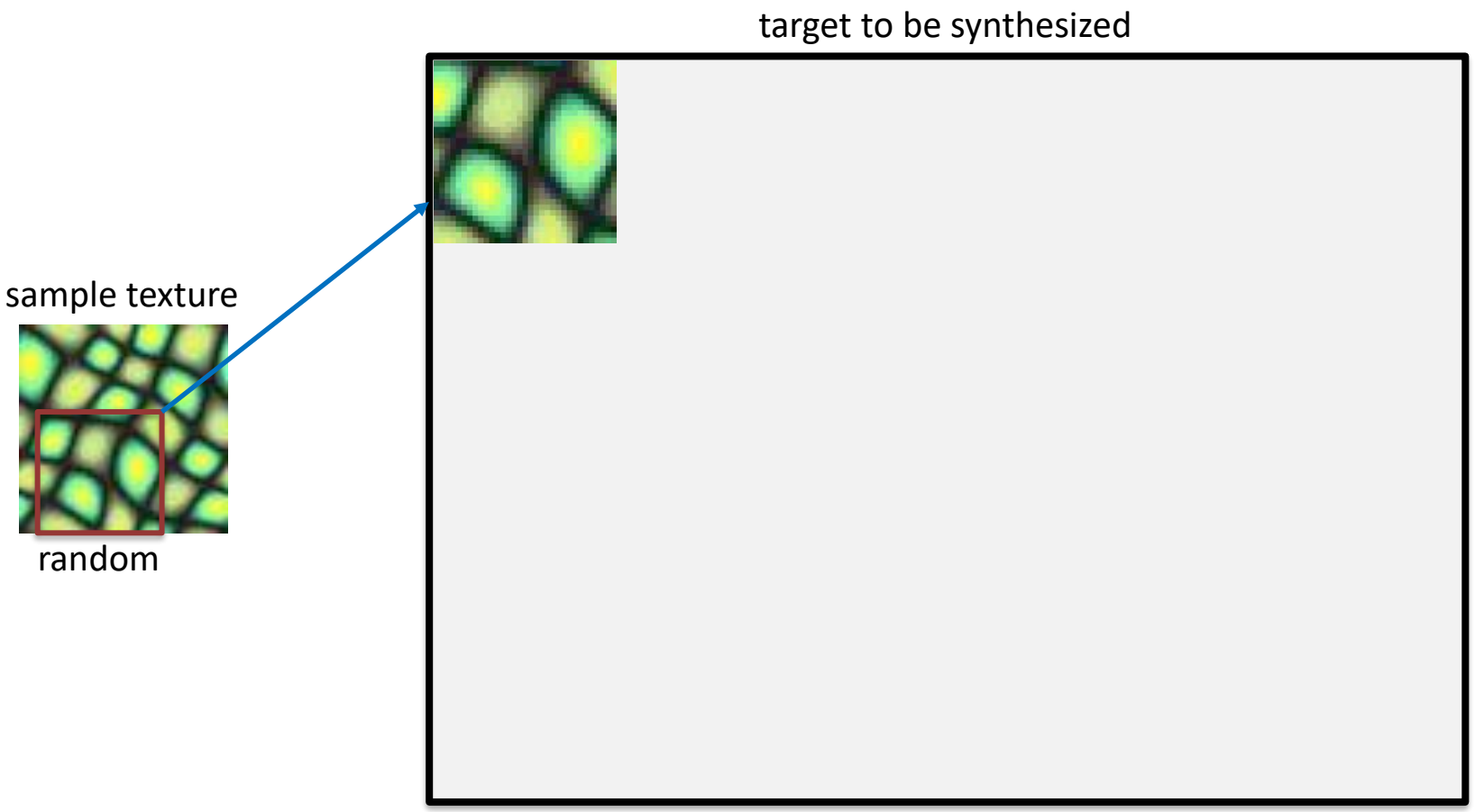




target to be synthesized

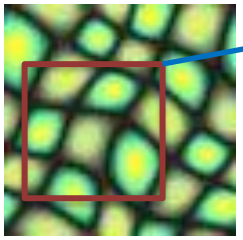
sample texture

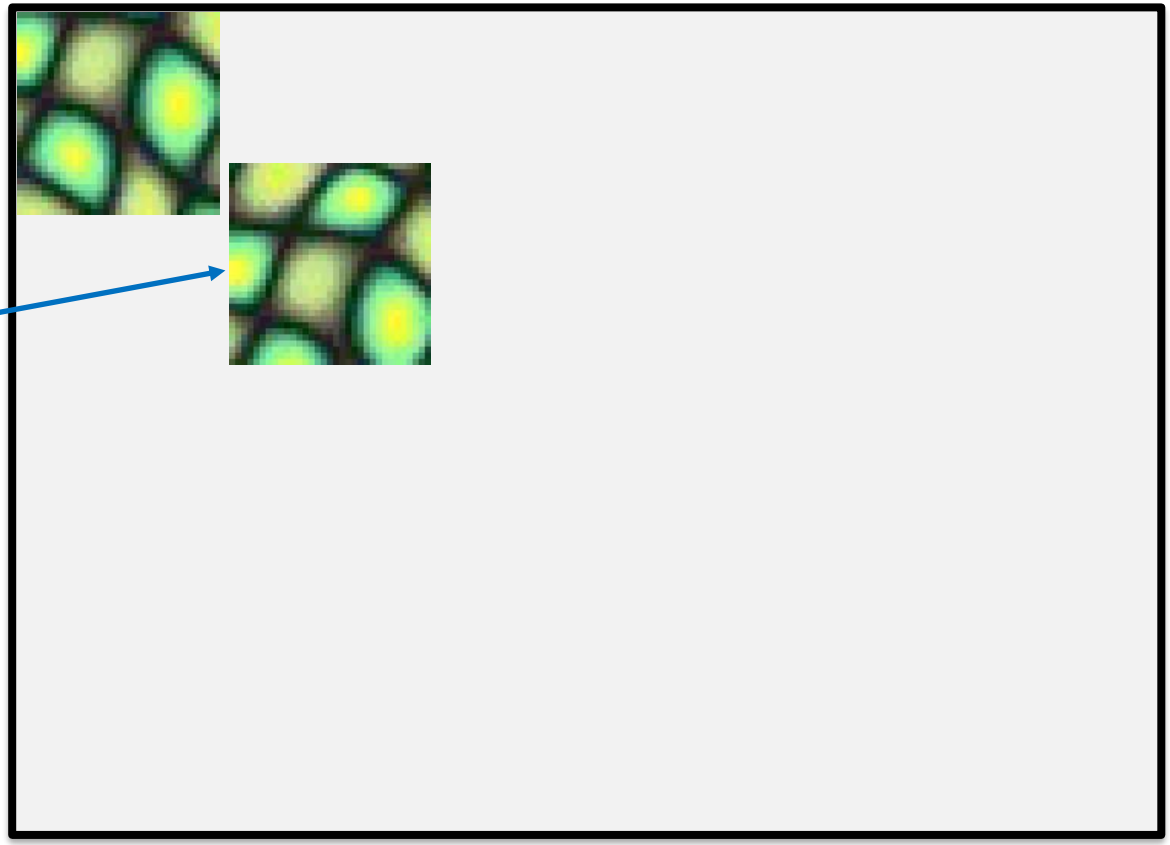



target to be synthesized
sample texture
random

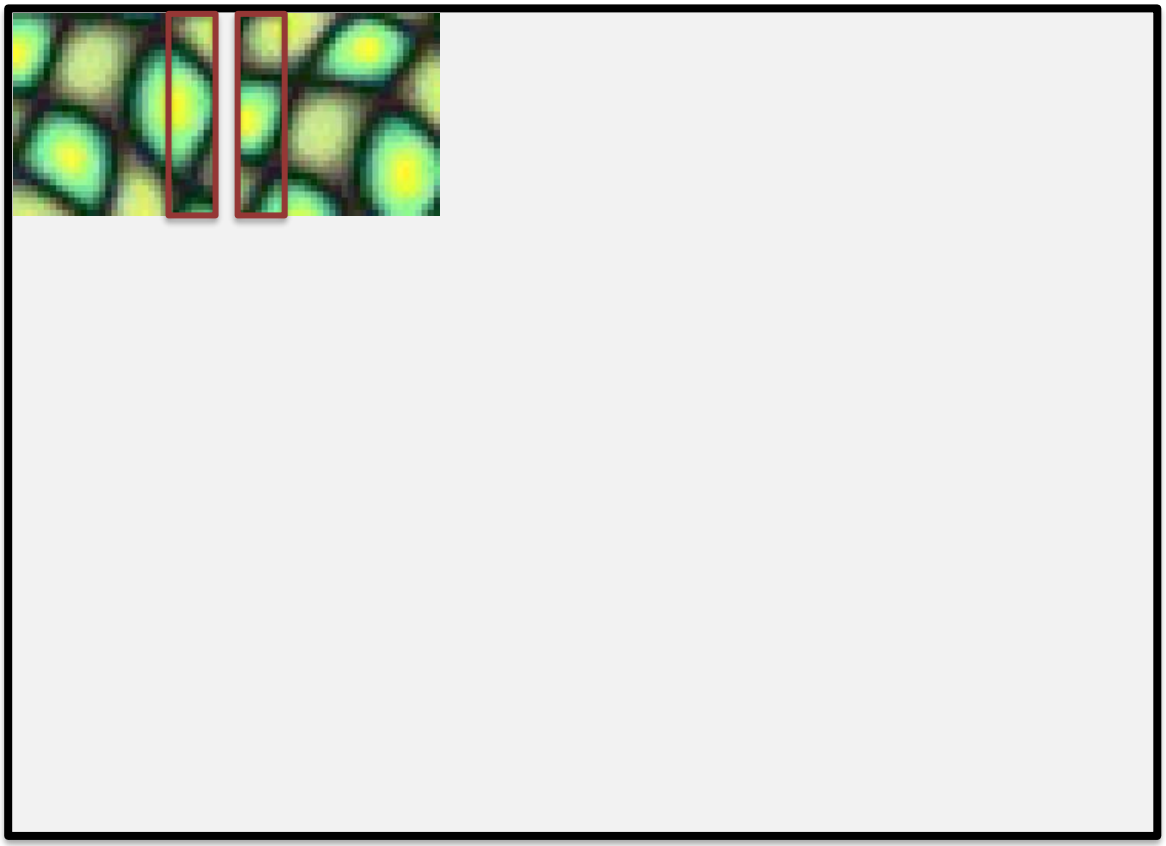
target to be synthesized

sample texture

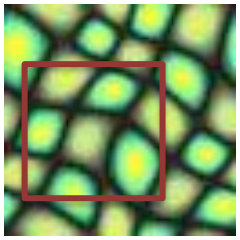




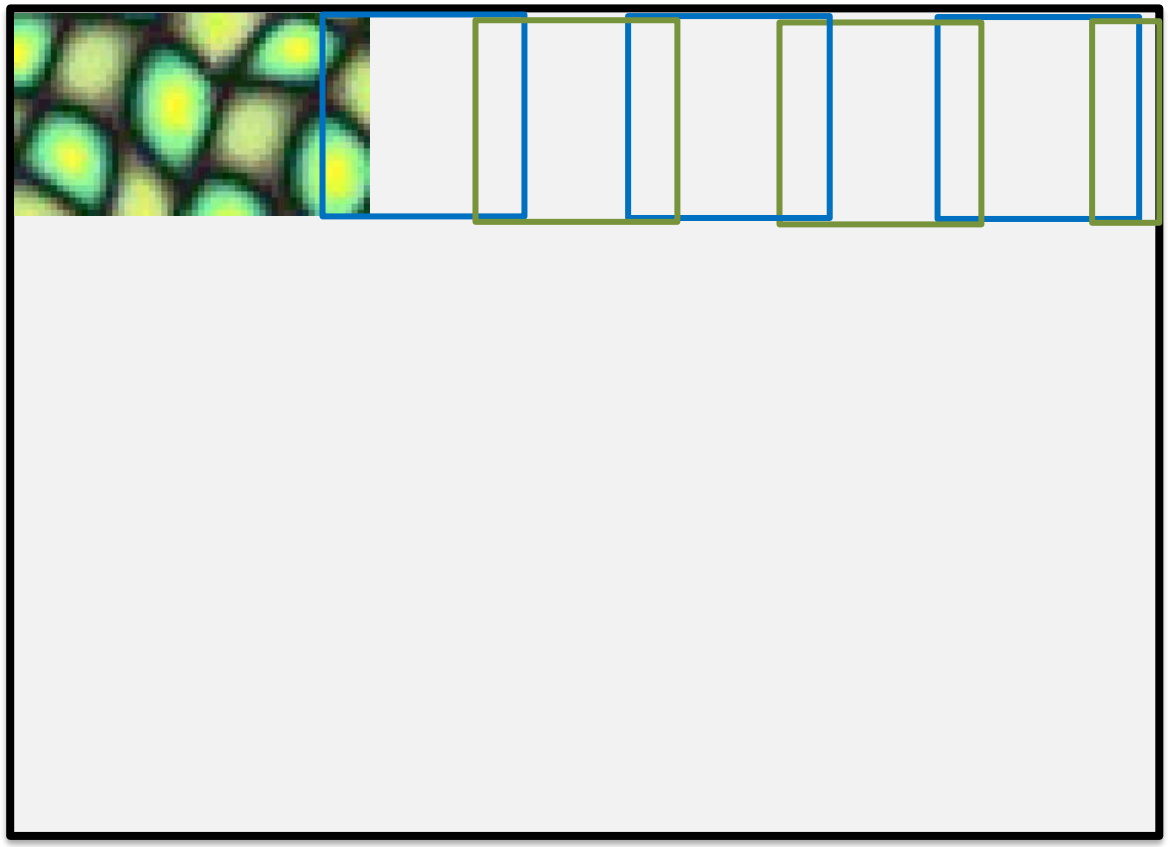
target to be synthesized



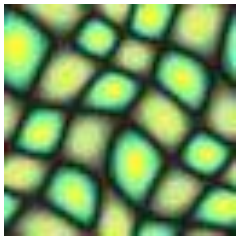
sample texture



target to be synthesized

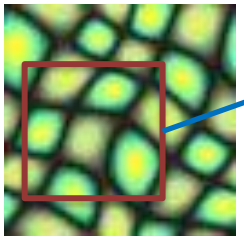


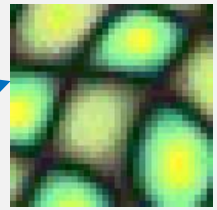
sample texture



target to be synthesized

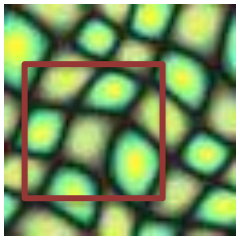
sample texture



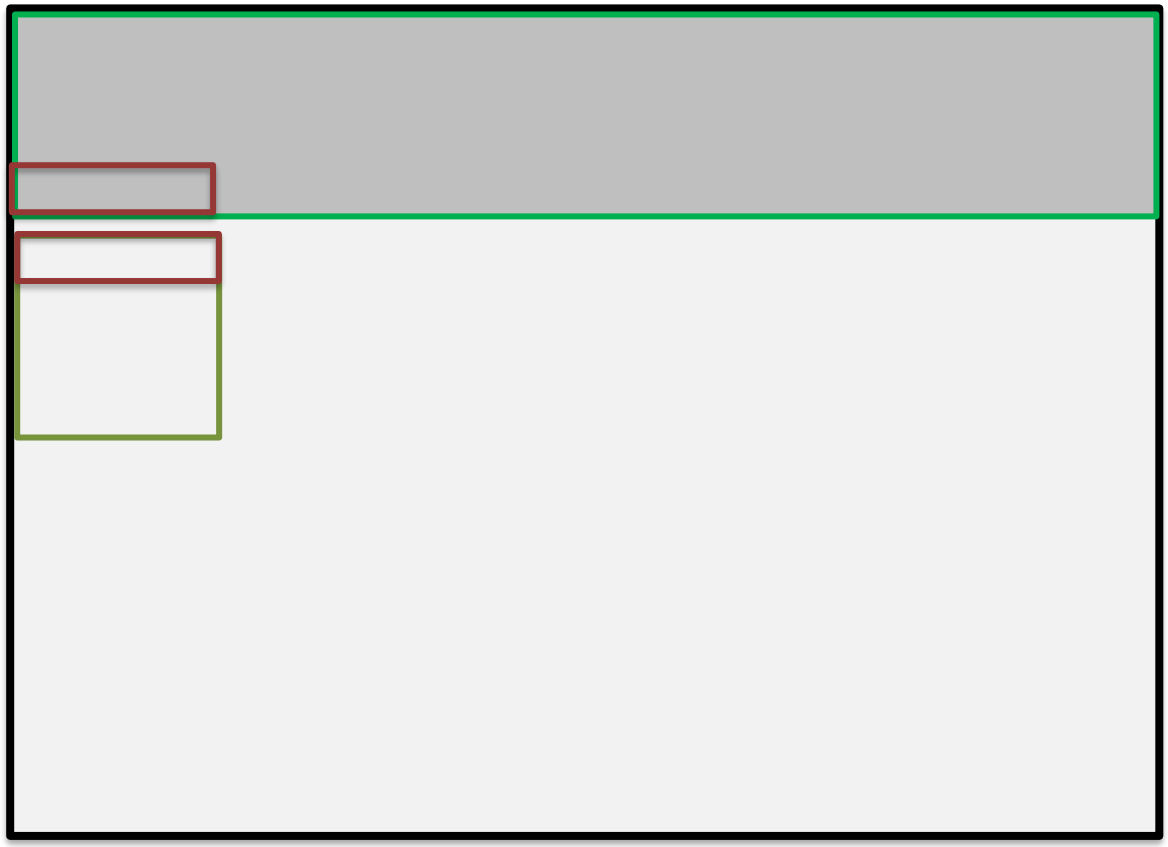


target to be synthesized

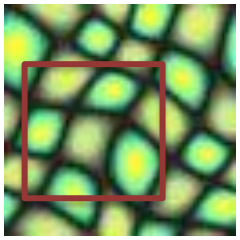
sample texture



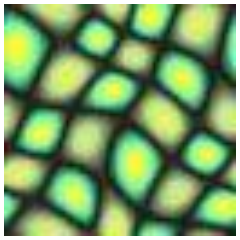
target to be synthesized



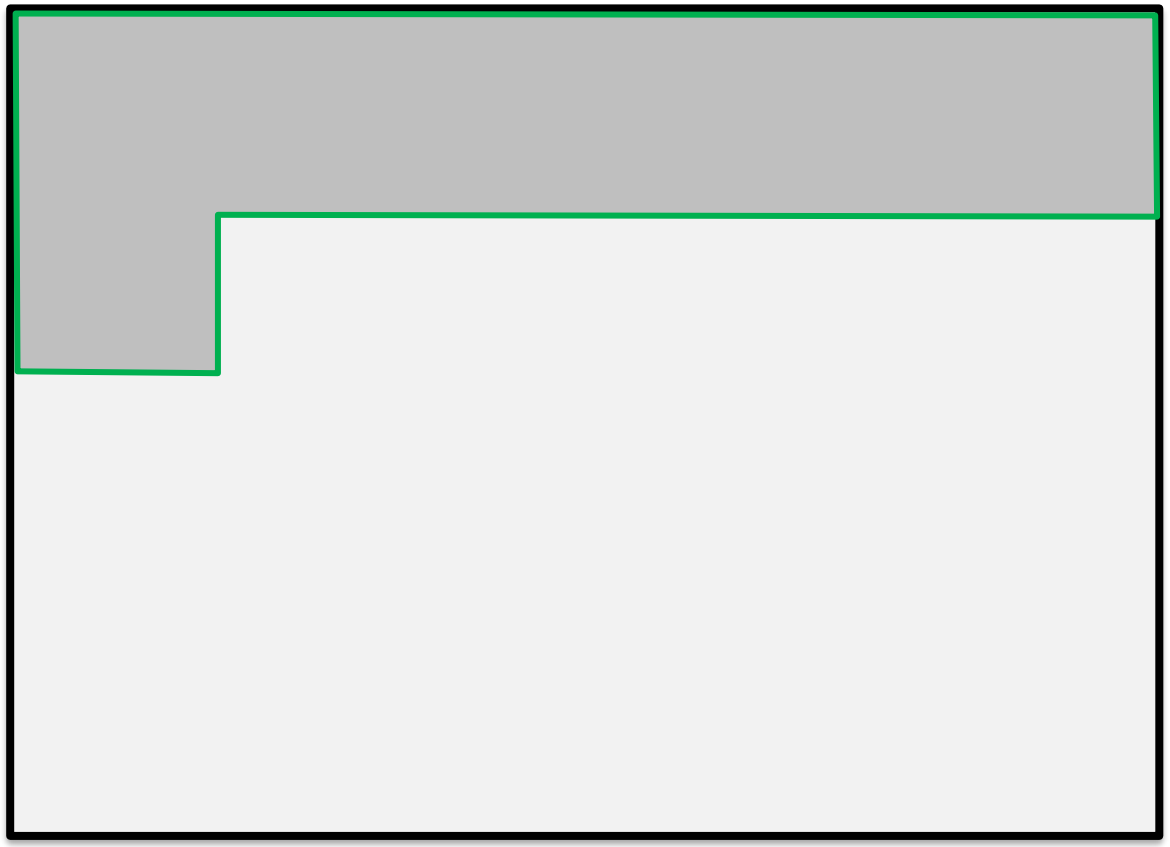
sample texture



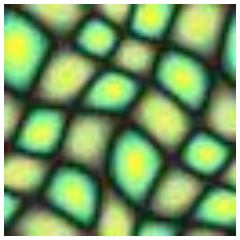
sample texture



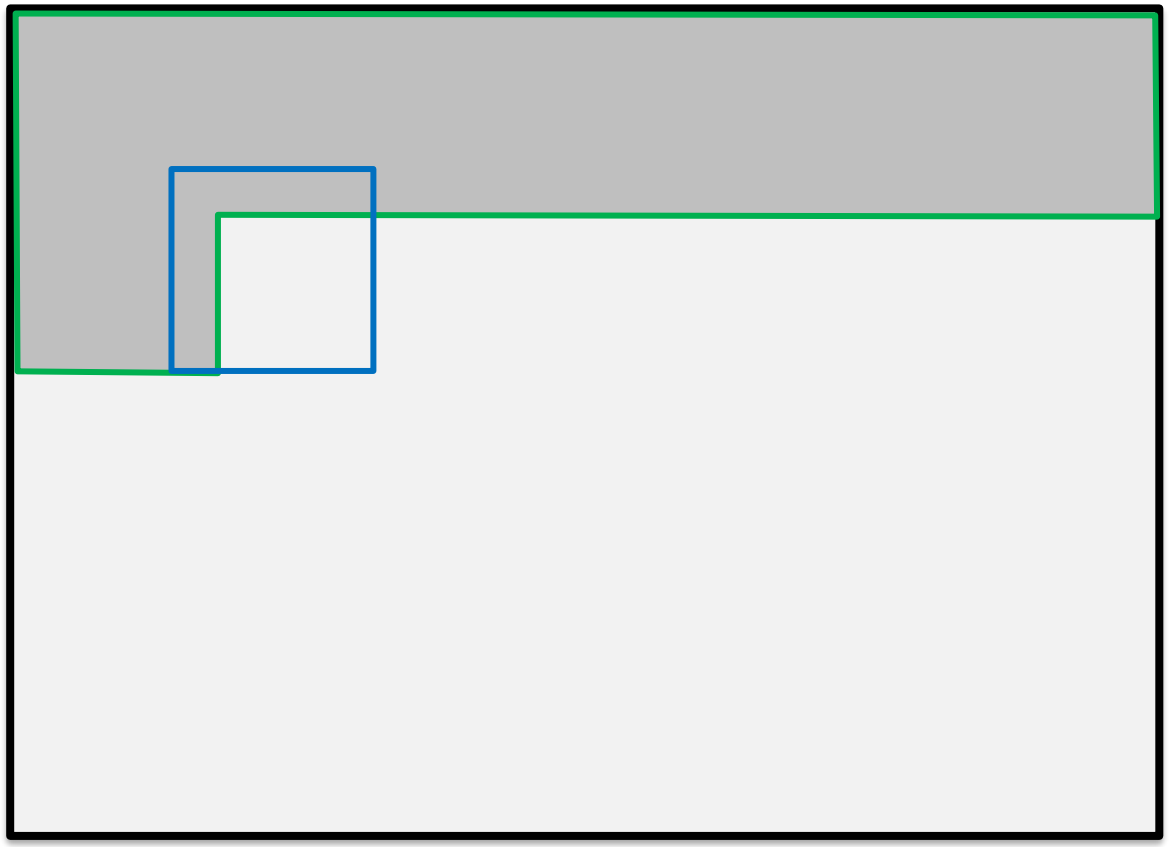
target to be synthesized



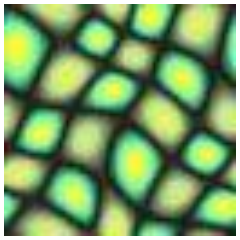
sample texture



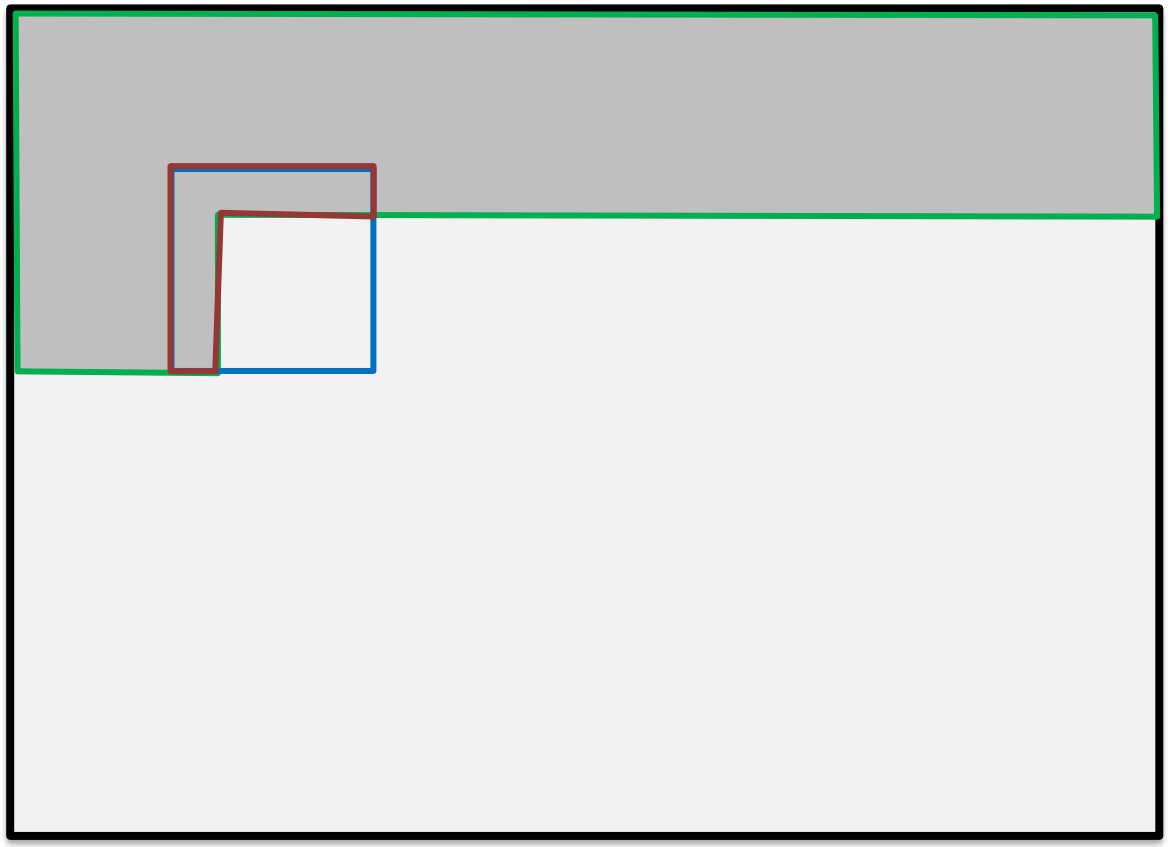
target to be synthesized



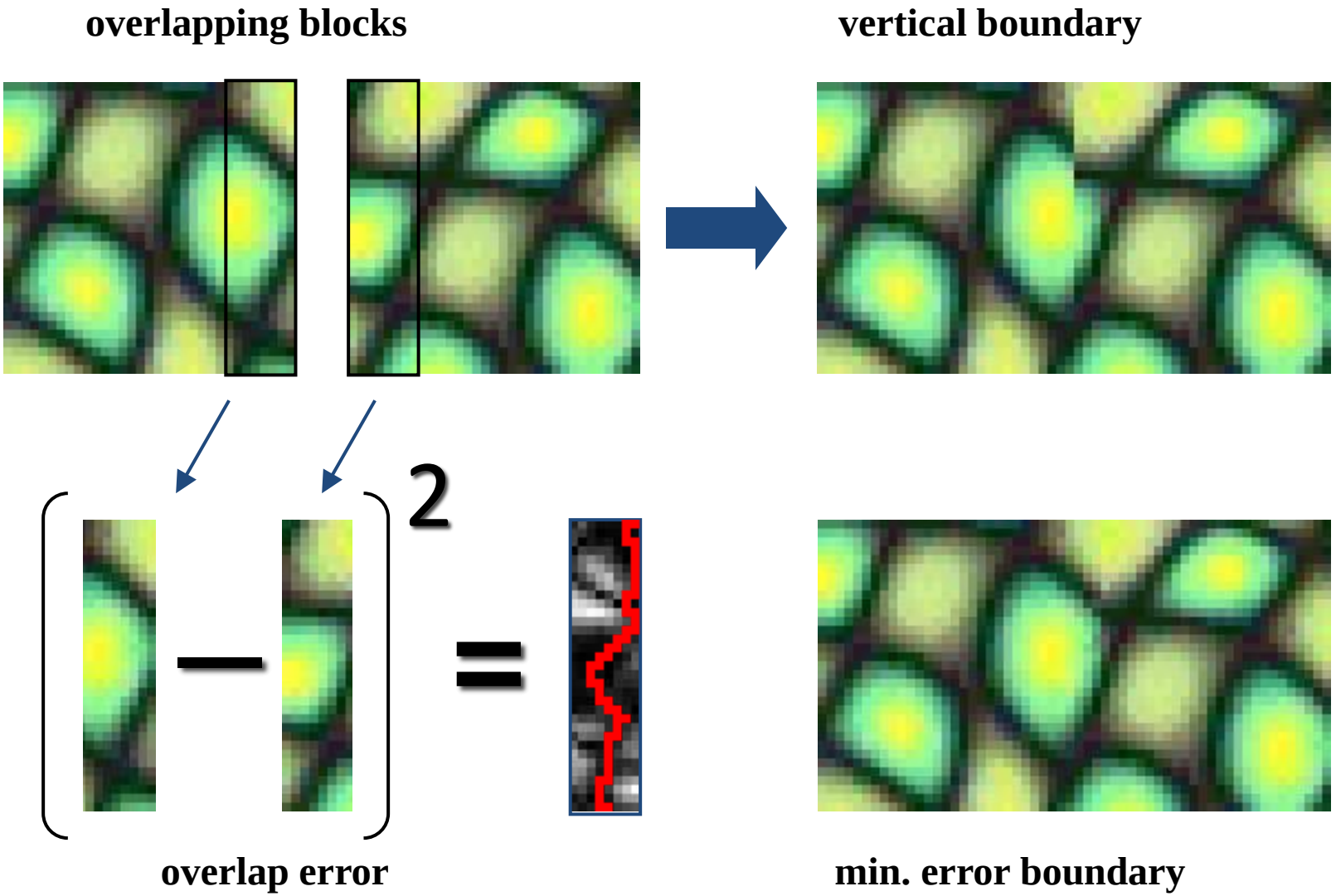
sample texture



target to be synthesized

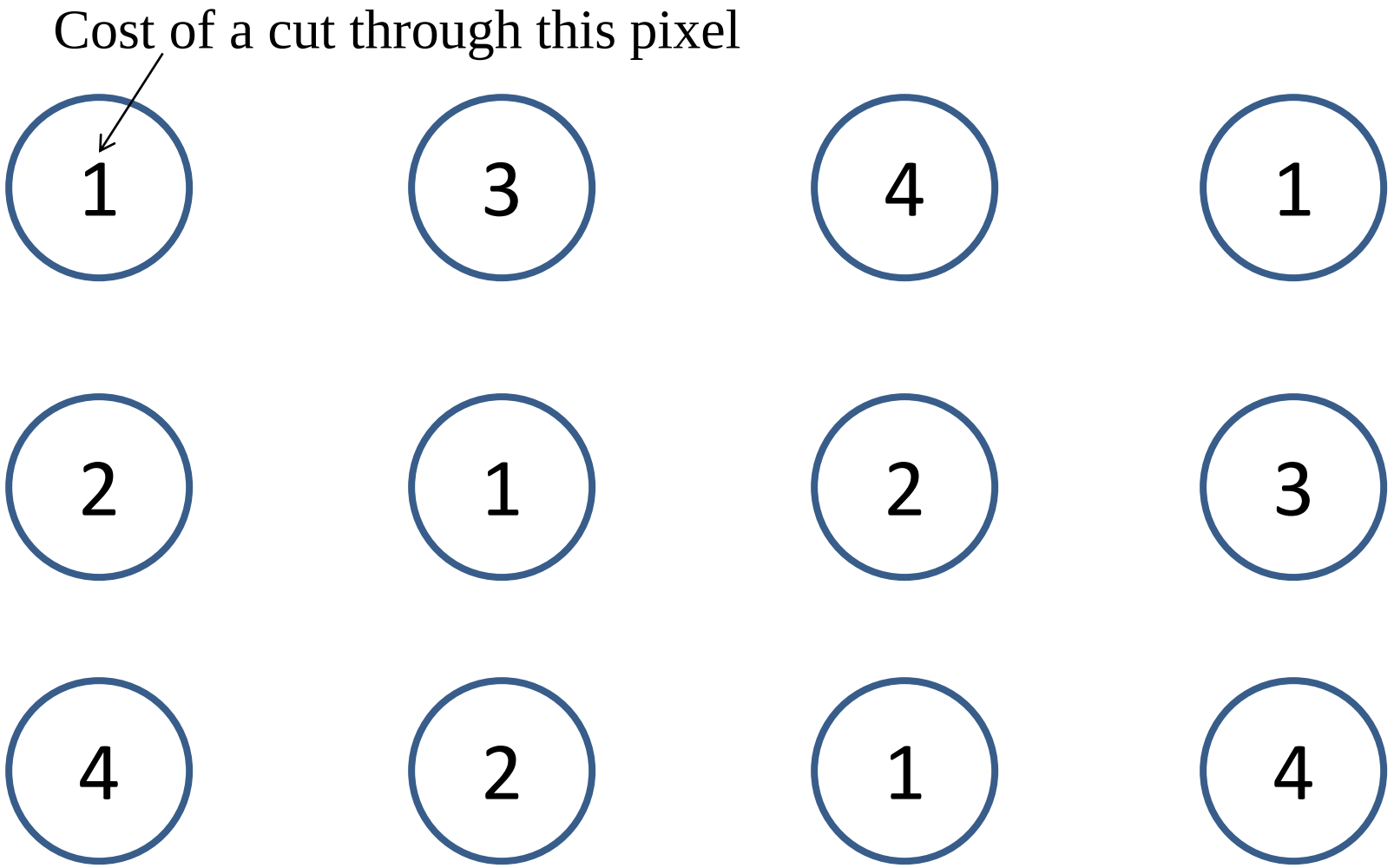


Minimal Error Boundary

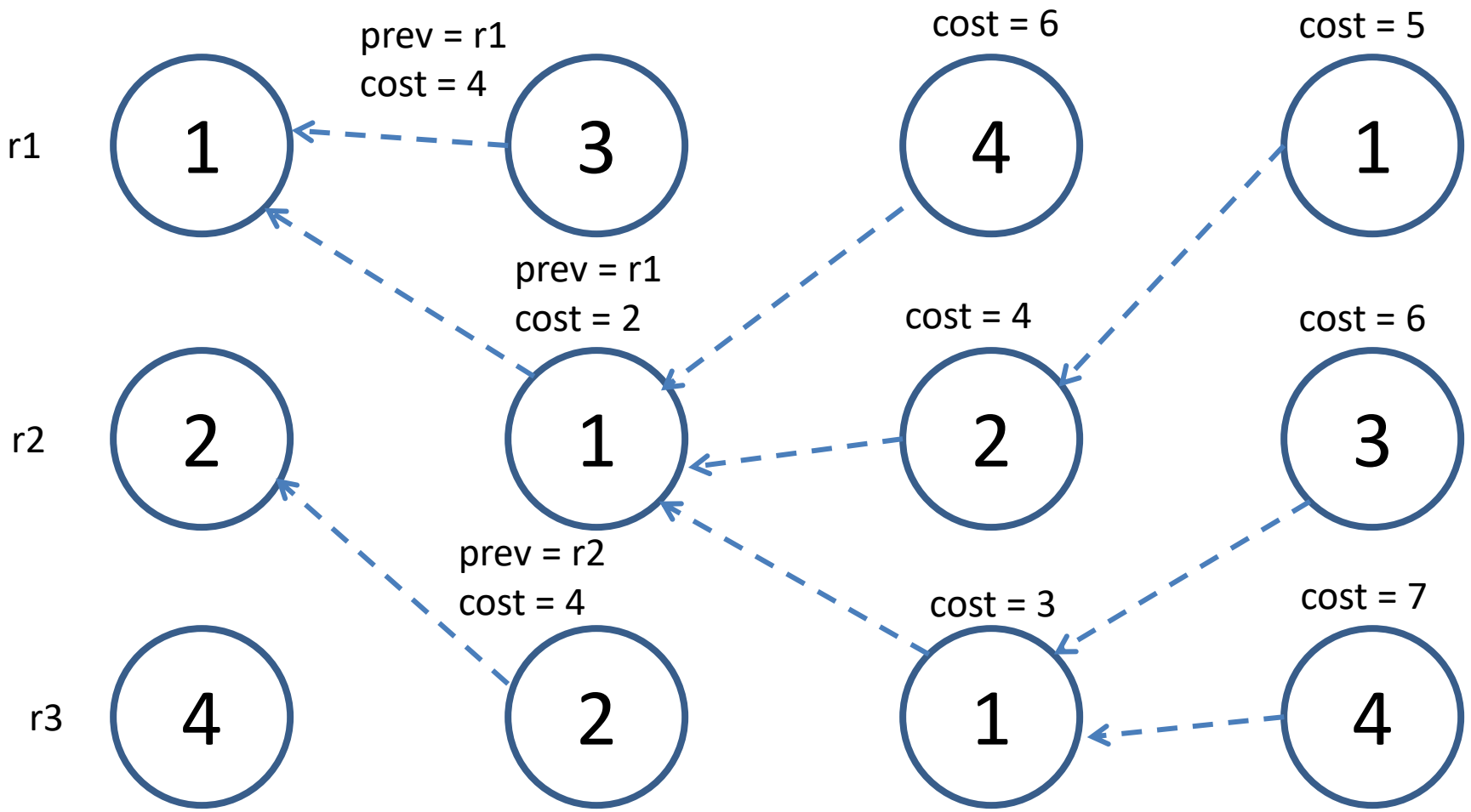


Slide: Alyosha Efros

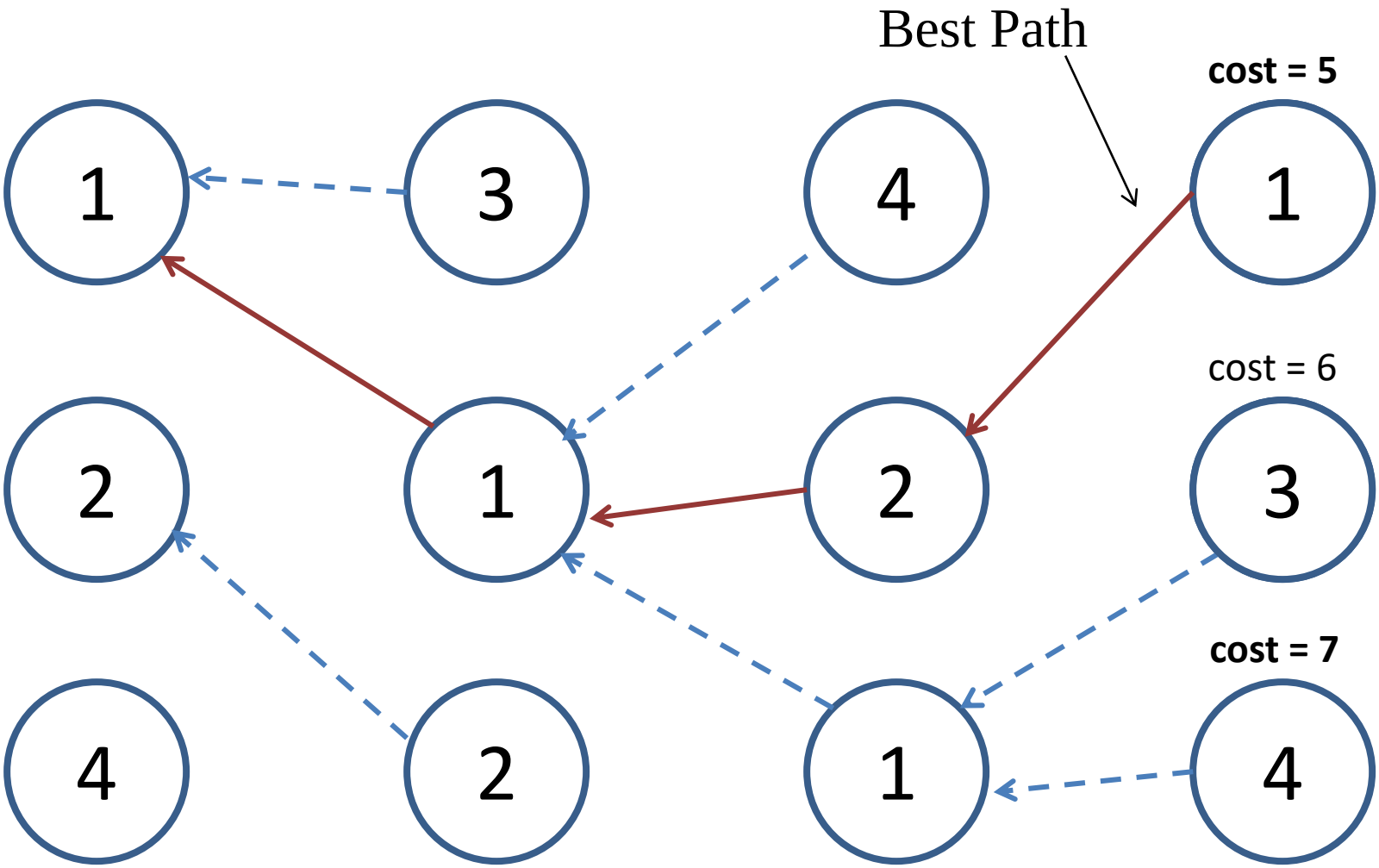
Solving for Minimum Cut Path



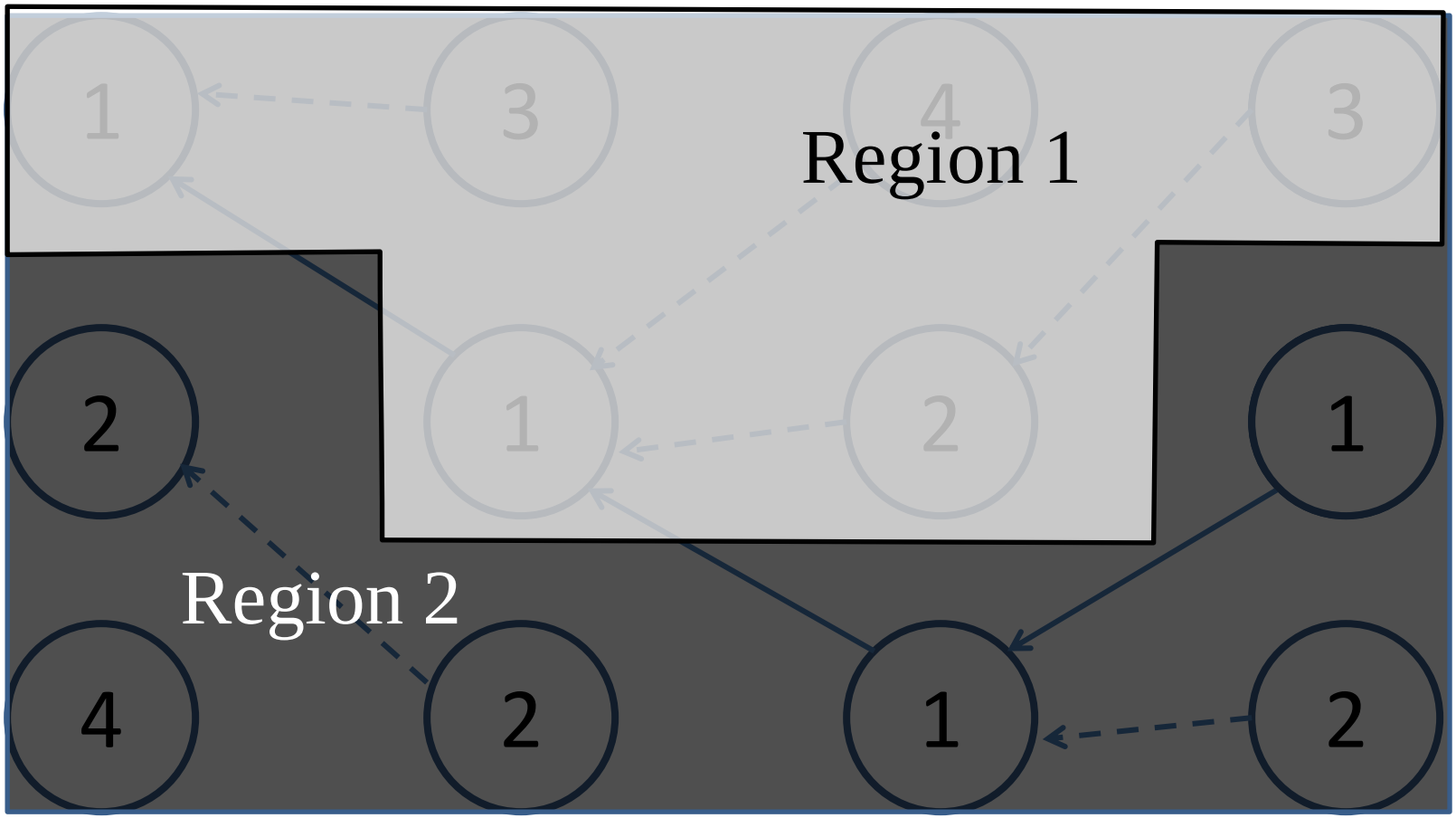
Solving for Minimum Cut Path



Solving for Minimum Cut Path

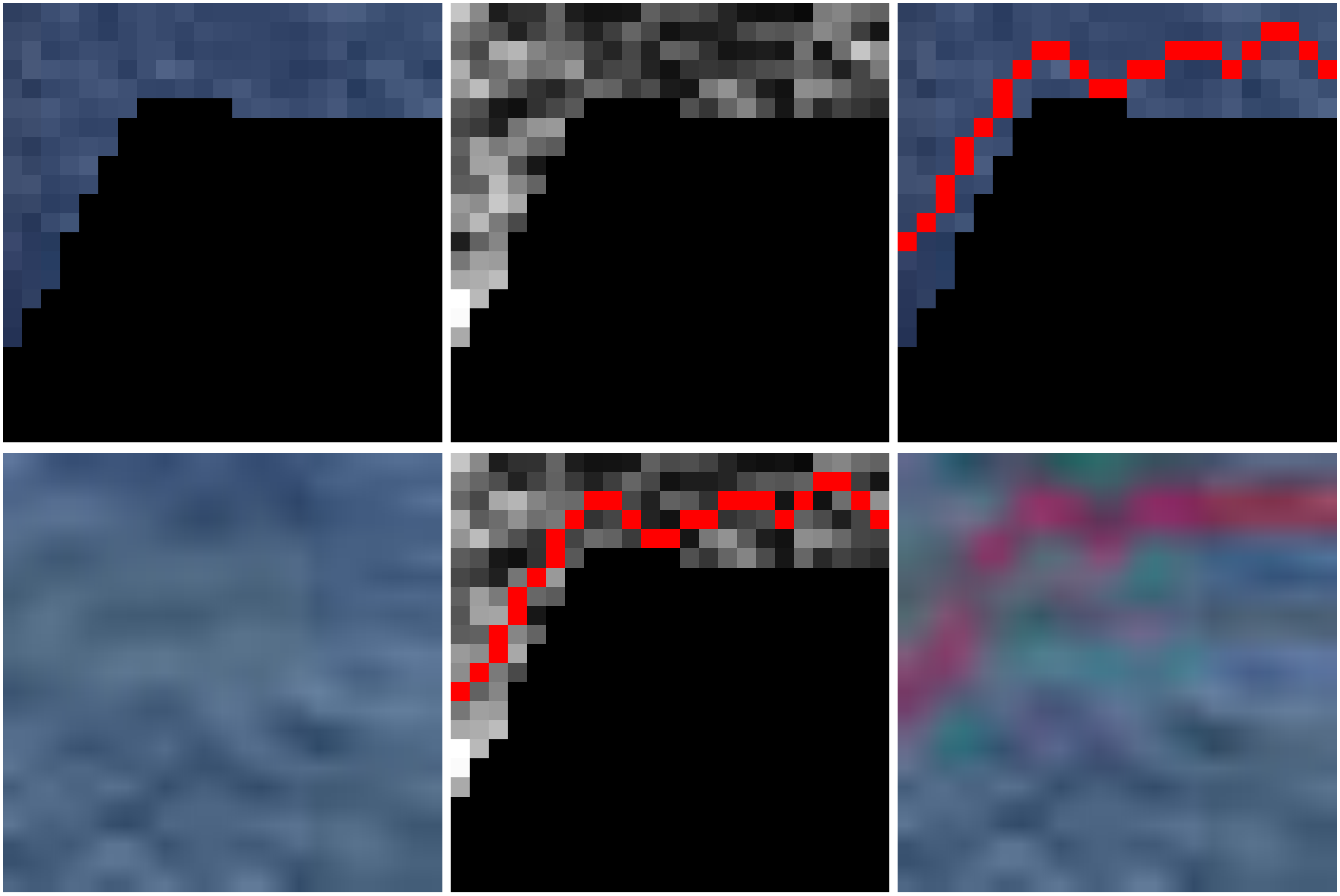


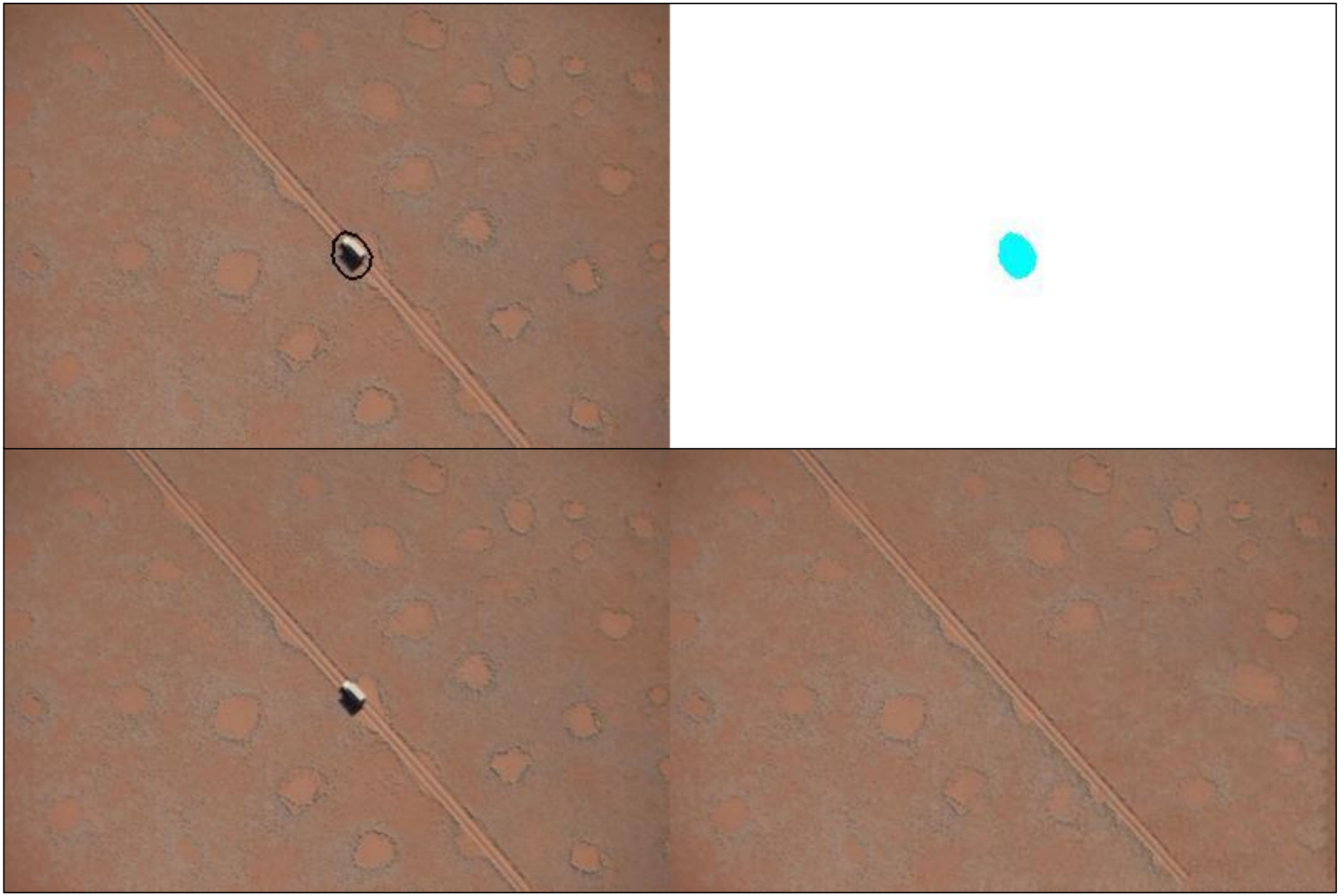
Solving for Minimum Cut Path

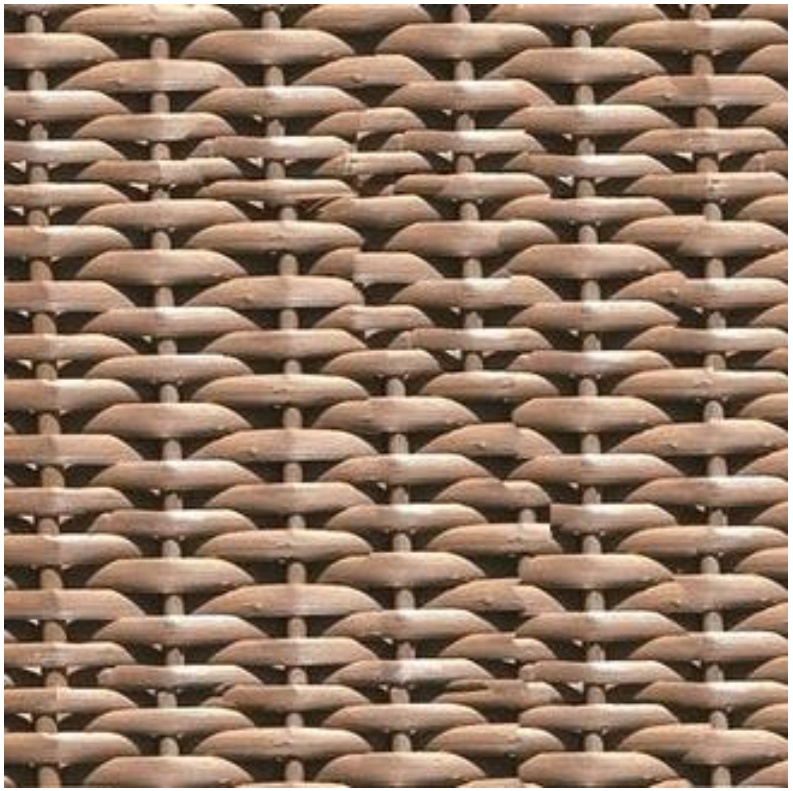


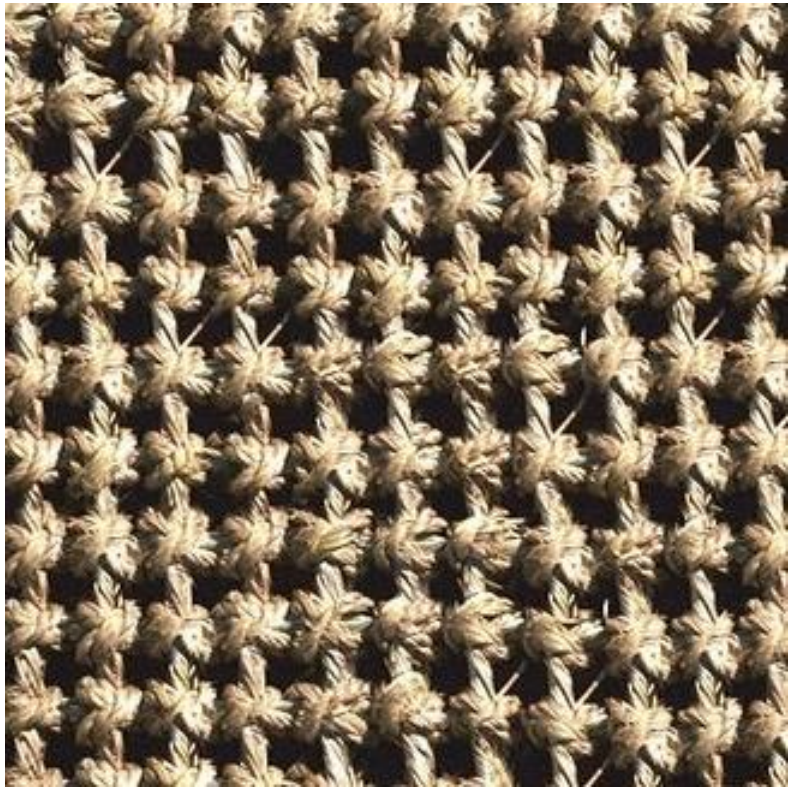
Mask Based on Best Path

Image Inpainting



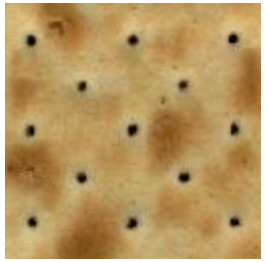
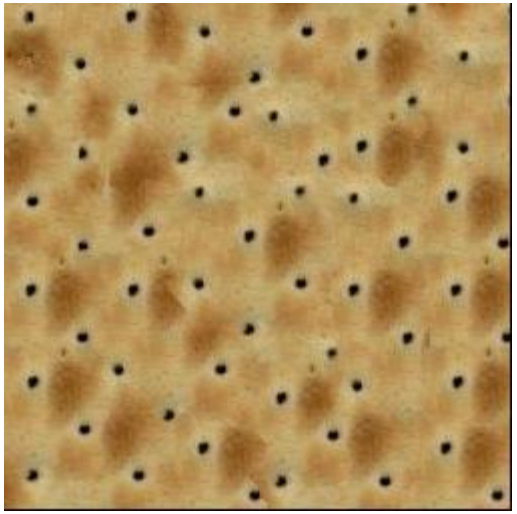






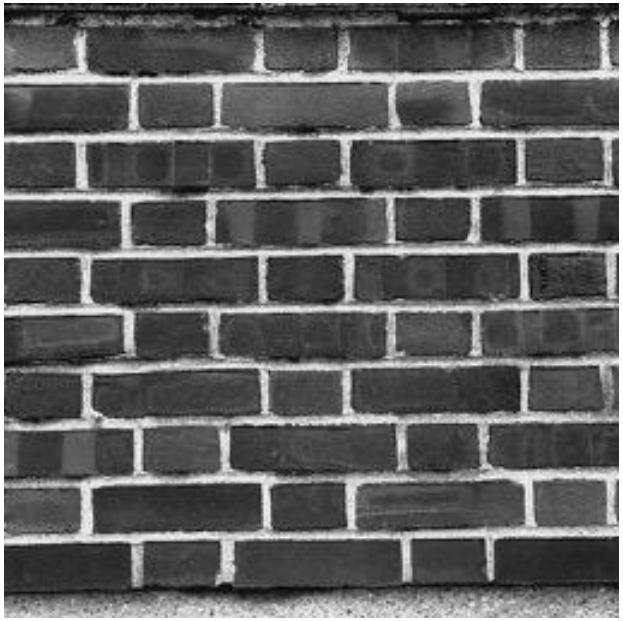

College Inn
Chicken Broth


College Inn
Chicken Broth

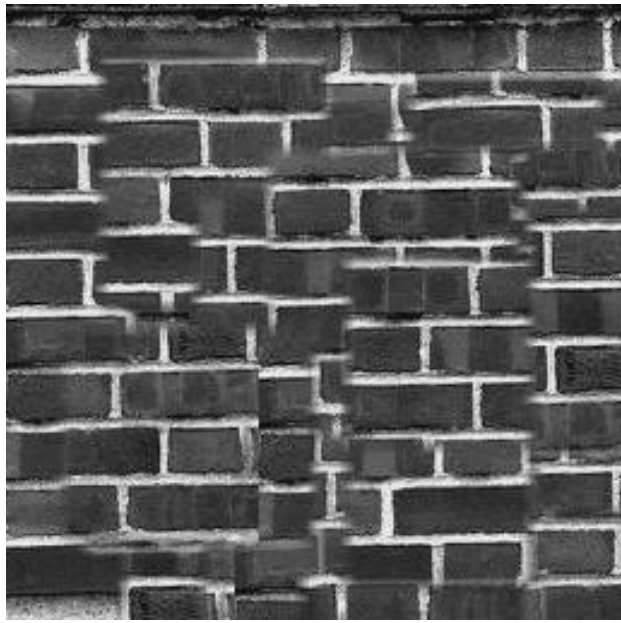




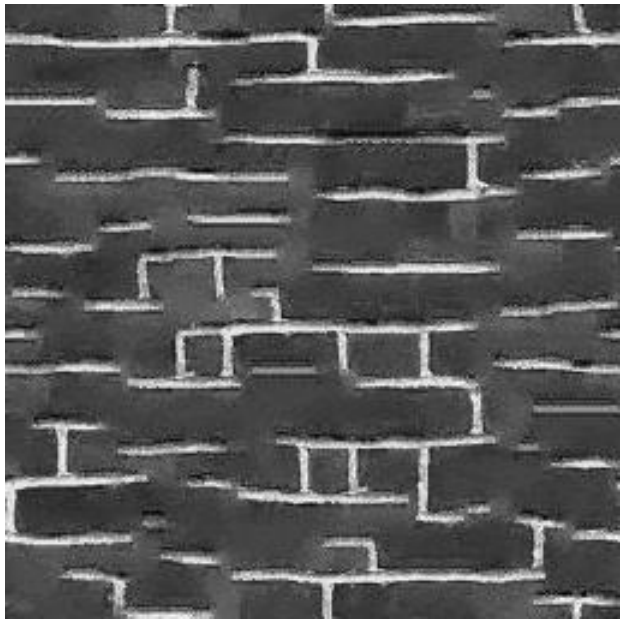


Input Image

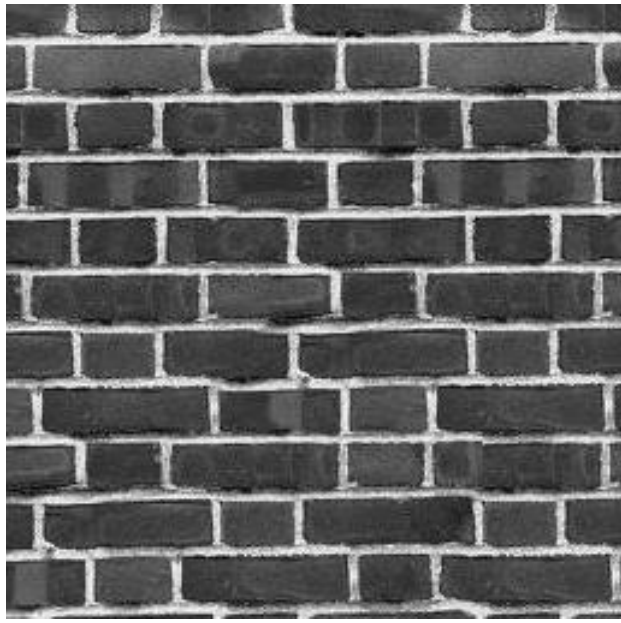
Portilla & Simoncelli



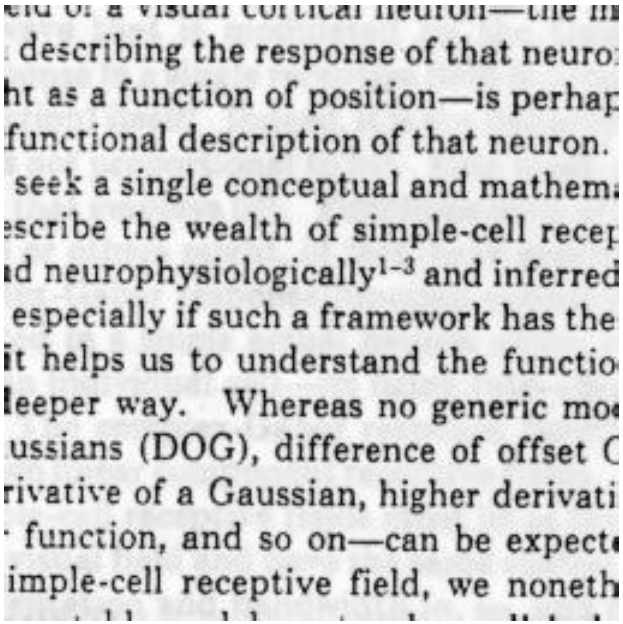
Xu, Guo & Shum



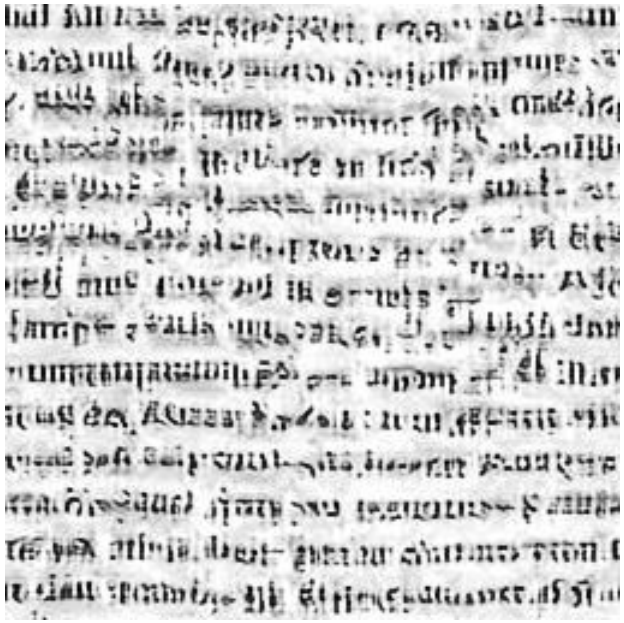
Wei & Levoy



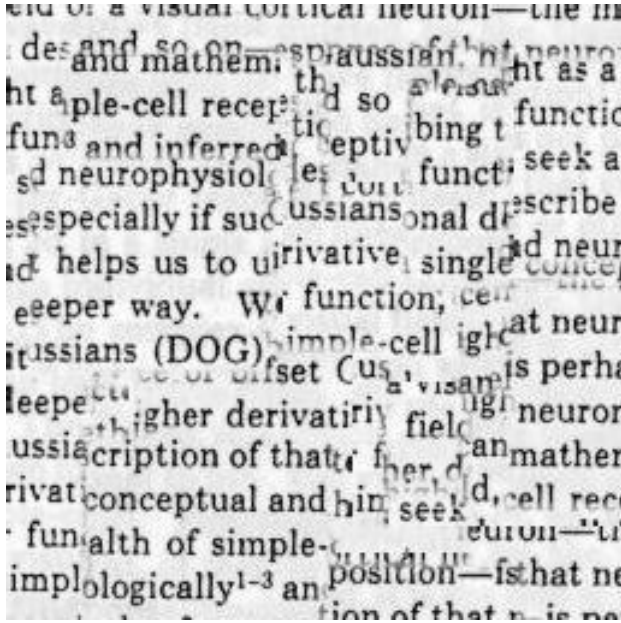
Quilting



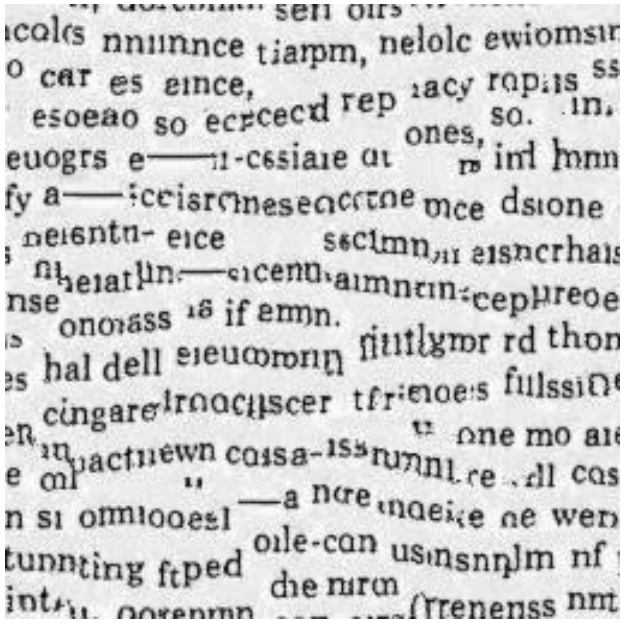
Input Image



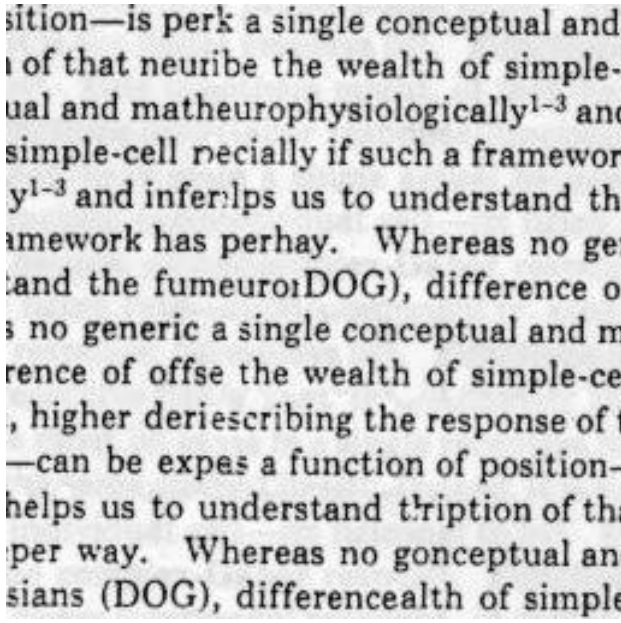
Portilla & Simoncelli



Xu, Guo & Shum



Wei & Levoy



Quilting

Failures

(Chernobyl Harvest)

In-Painting Natural Scenes



Criminisi, Perez, and Toyama. "Object Removal by Exemplar-based Inpainting," CVPR 2003

Key Idea: Filling Order Matters

In-painting Result

Image with Hole

Raster-Scan Order

Onion-Peel
(Concentric Layers)

Gradient-Sensitive
Order

Criminisi, Perez, and Toyama. "Object Removal by Exemplar-based Inpainting," CVPR 2003

Filling Order

Fill a pixel that:

1. Is surrounded by other known pixels
2. Is a continuation of a strong gradient or edge

Criminisi, Perez, and Toyama. "Object Removal by Exemplar-based Inpainting," CVPR 2003

Original

With Hole

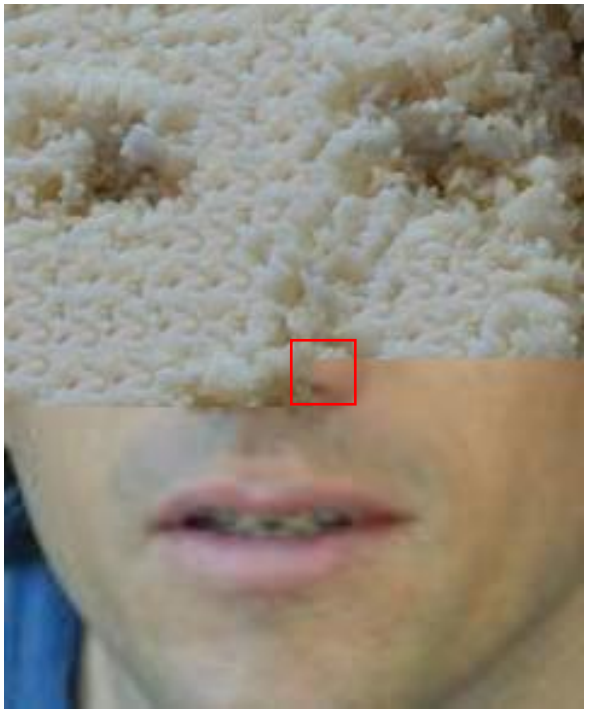
Onion-Ring Fill

Criminisi

Criminisi, Perez, and Toyama. "Object Removal by Exemplar-based Inpainting," CVPR 2003

Texture Transfer

- Take the texture from one object and “paint” it onto another object

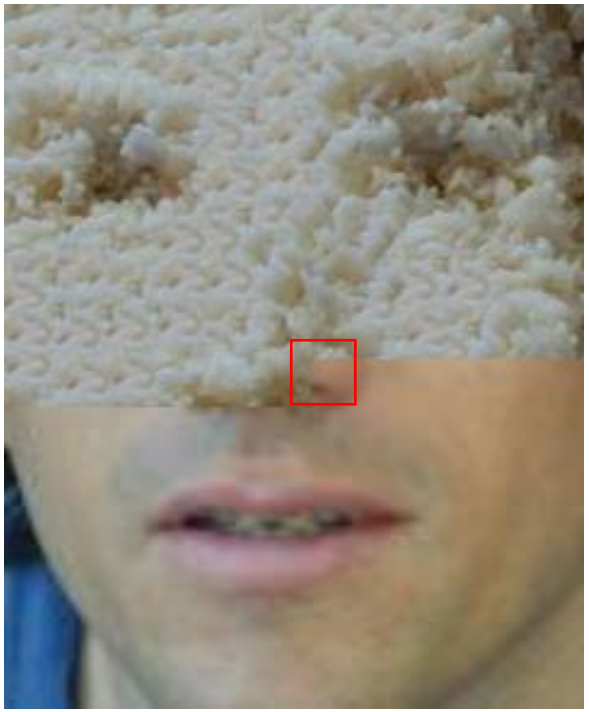


Same as texture synthesis, except an additional constraint:

1. Consistency of texture
2. Patches from texture should correspond to patches from constraint in some way. Typical example: blur luminance, use SSD for distance

Texture Transfer

- Take the texture from one object and “paint” it onto another object
  - This requires separating texture and shape
  - That’s HARD, but we can cheat
  - Assume we can capture shape by boundary and rough shading

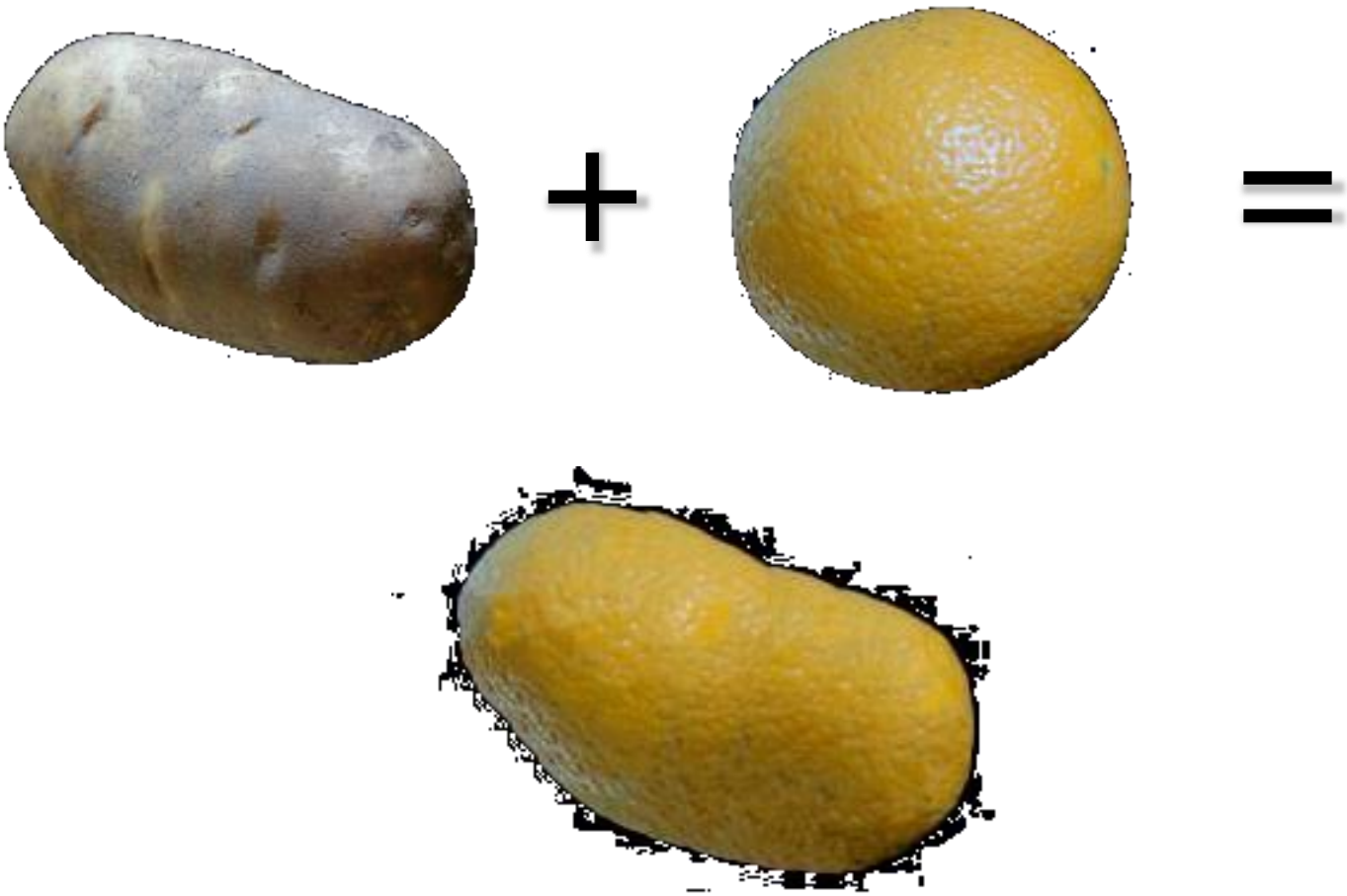


Then, just add another constraint when sampling:

Similarity to underlying image at that spot

parmesan

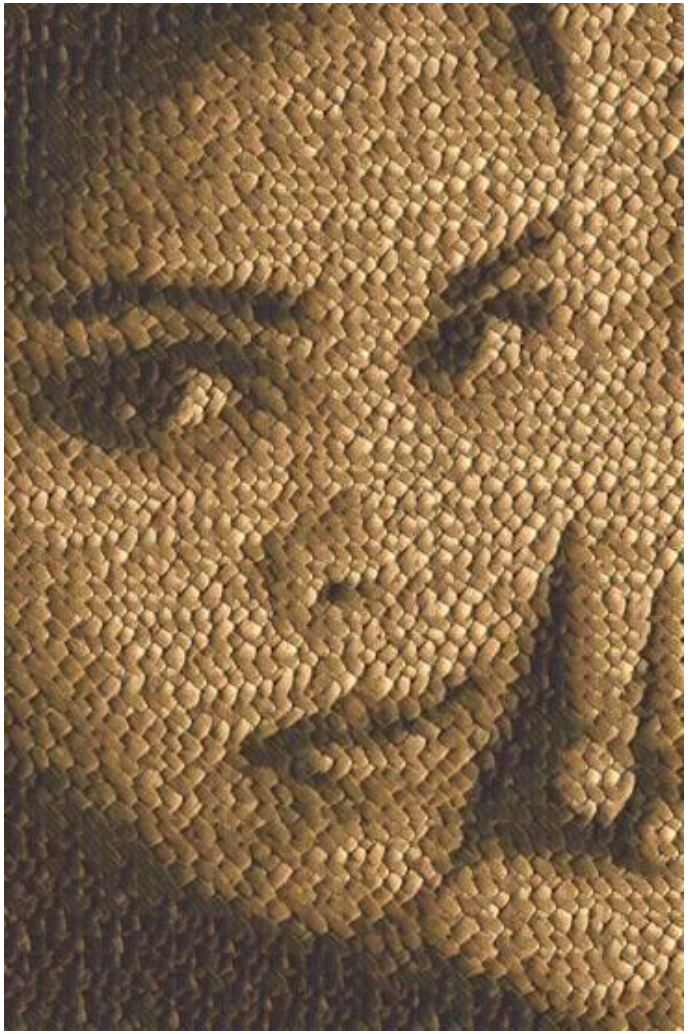
rice



Texture Synthesis and Transfer Recap

For each overlapping patch in the output image

1. Compute the cost to each patch in the sample
   - Texture synthesis: this cost is the SSD (sum of square difference) of pixel values in the overlapping portion of the existing output and sample
   - Texture transfer: cost is $\alpha\, SSD_{overlap} + (1-\alpha) SSD_{transfer}$. The latter term enforces that the source and target correspondence patches should match.
2. Select one sample patch that has a small cost
3. Find a cut through the left/top borders of the patch based on overlapping region with existing output
   - Use this cut to create a mask that specifies which pixels to copy from sample patch
4. Copy masked pixels from sample image to corresponding pixel locations in output image

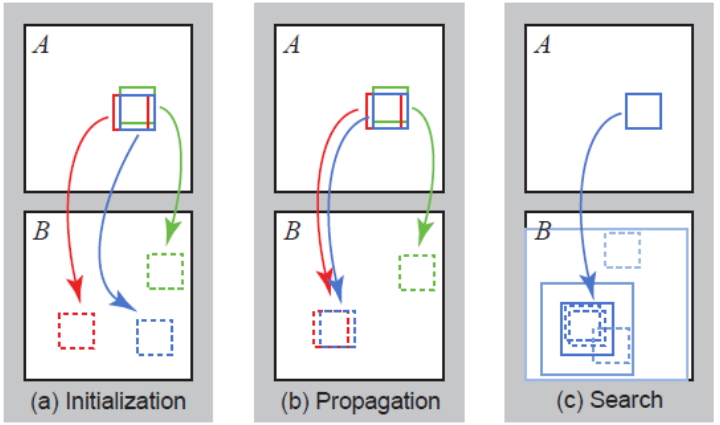


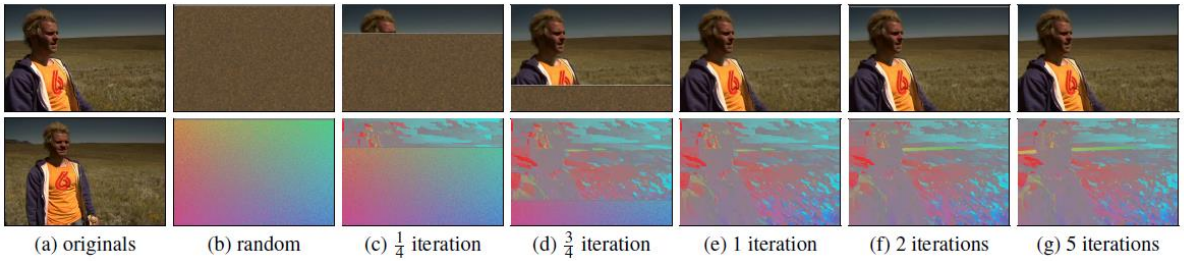
PatchMatch

Recommended!

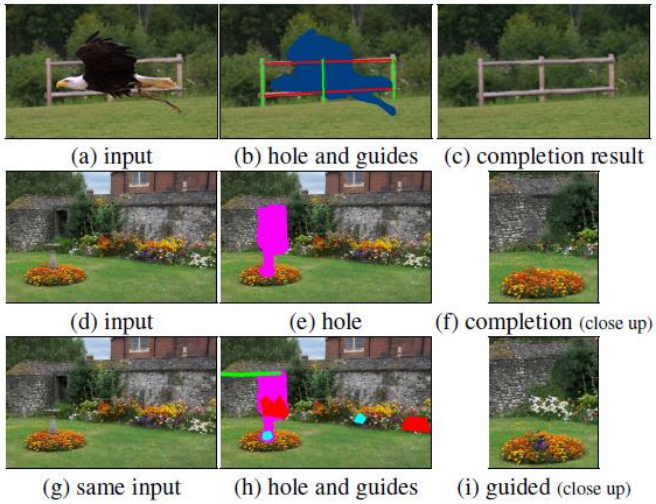
More efficient search:

1. Randomly initialize matches
2. See if neighbor's offsets are better
3. Randomly search a local window for better matches
4. Repeat 2, 3 across image several times





Reconstructing top-left image with patches from bottom-left image



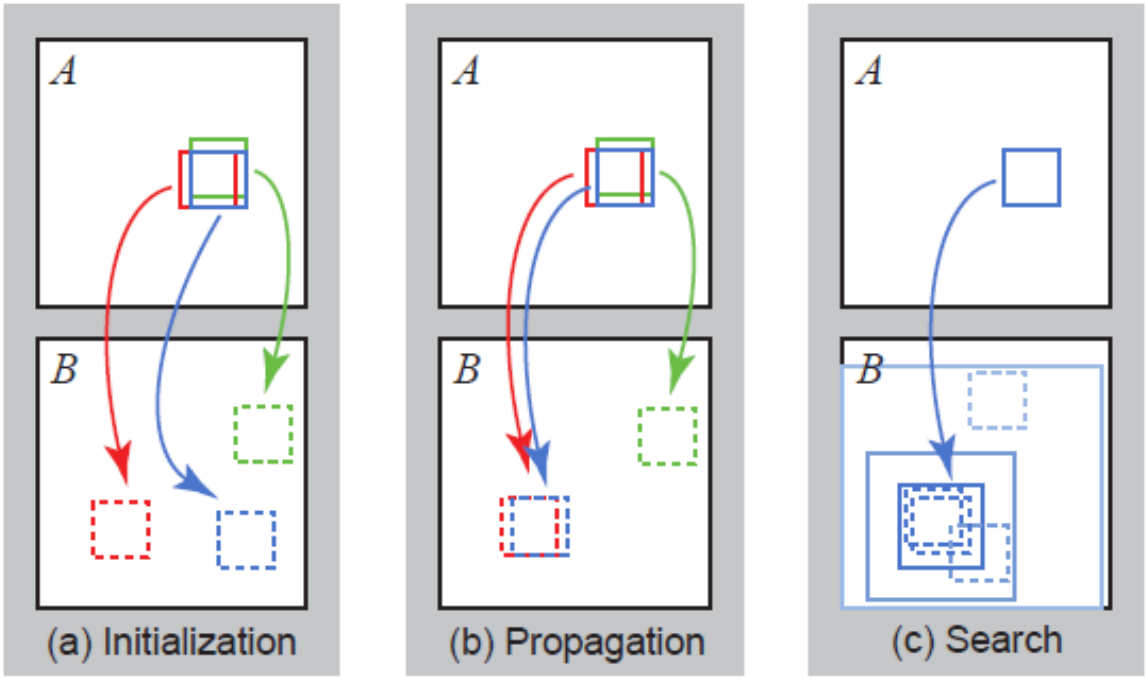
Applications to hole-filling, retargeting; constraints can guide search

Barnes et al. Patch Match, SIGGRAPH 2009
Barnes et al. Generalized Patch Match, ECCV 2010
Ehret & Arias, On the convergence of PatchMatch and its variants, CVPR 2018

PatchMatch

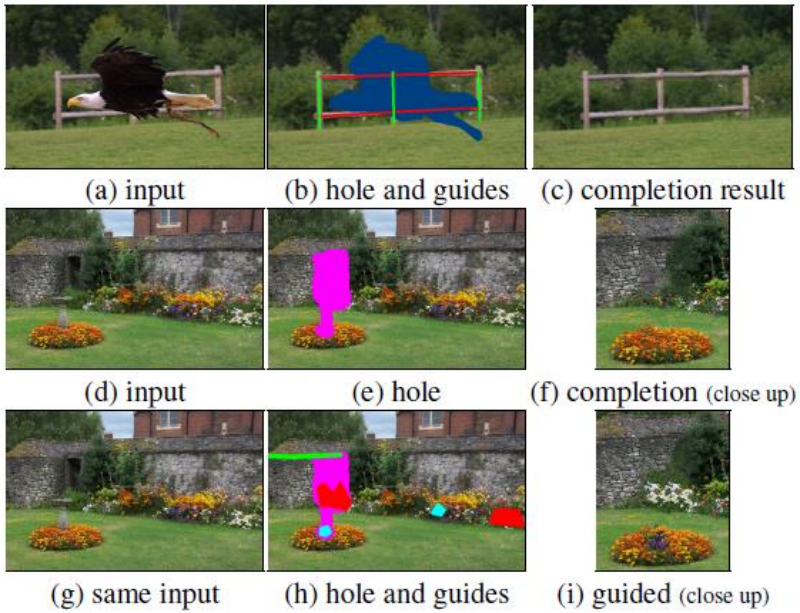
More efficient search:

1. Randomly initialize matches
2. See if neighbor's offsets are better
3. Randomly search a local window for better matches
4. Repeat 2, 3 across image several times



PatchMatch

(a) originals (b) random (c) $\frac{1}{4}$ iteration (d) $\frac{3}{4}$ iteration (e) 1 iteration (f) 2 iterations (g) 5 iterations



(a) input (b) hole and guides (c) completion result

(d) input (e) hole (f) completion (close up)

(g) same input (h) hole and guides (i) guided (close up)

Related idea: Image Analogies

A : :

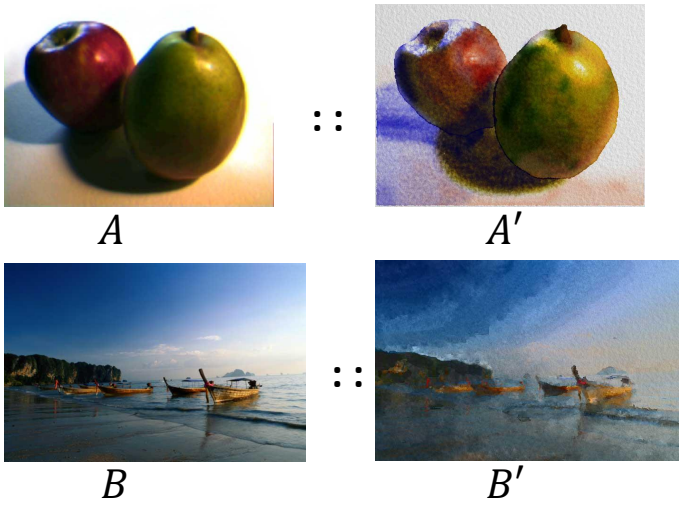
A'

B : :

B'

Image Analogies, Hertzmann et al. siggraph 2001

Image Analogies



- Define a similarity between A and B
- For each patch in B :
  - Find a matching patch in A, whose corresponding A' also fits in well with existing patches in B'
  - Copy the patch in A' to B'
- Algorithm is done iteratively, coarse-to-fine

Blur Filter

Unfiltered source (A)

Filtered source (A')

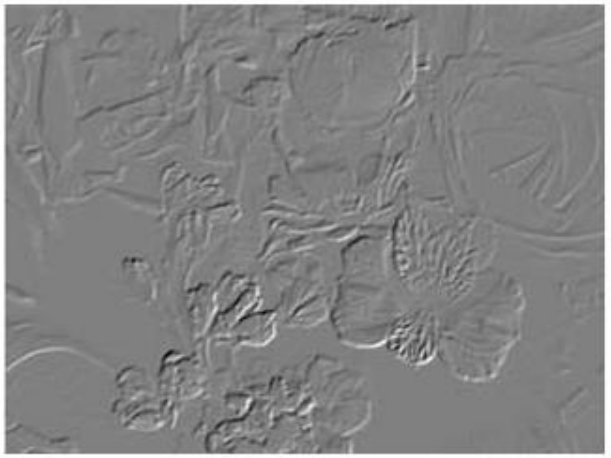
Unfiltered target (B)

Filtered target (B')

Edge Filter

Unfiltered source (*A*)

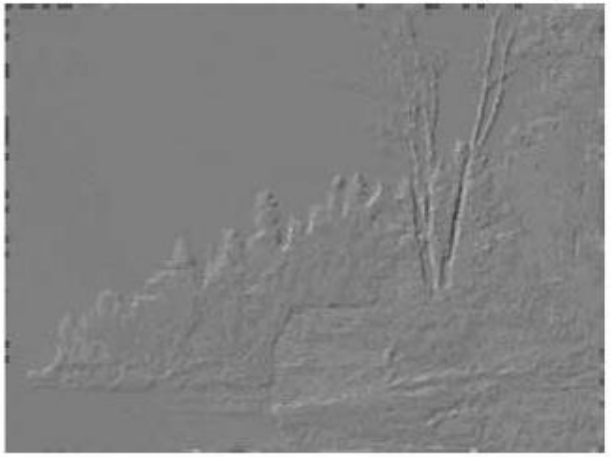
:



Filtered source (*A′*)

::

Unfiltered target (*B*)



Filtered target (*B′*)

Artistic Filters

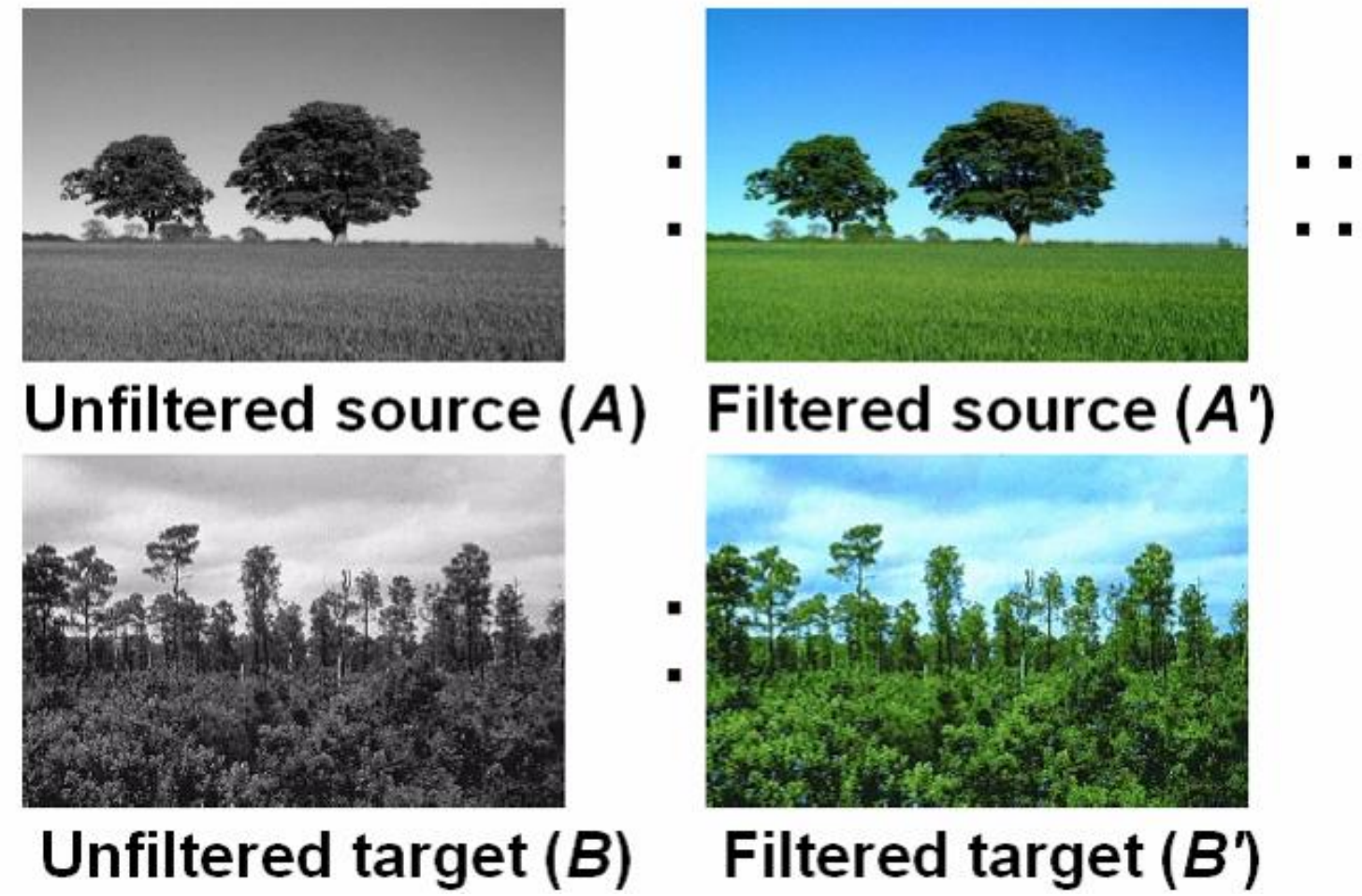
A

A'

B

B'

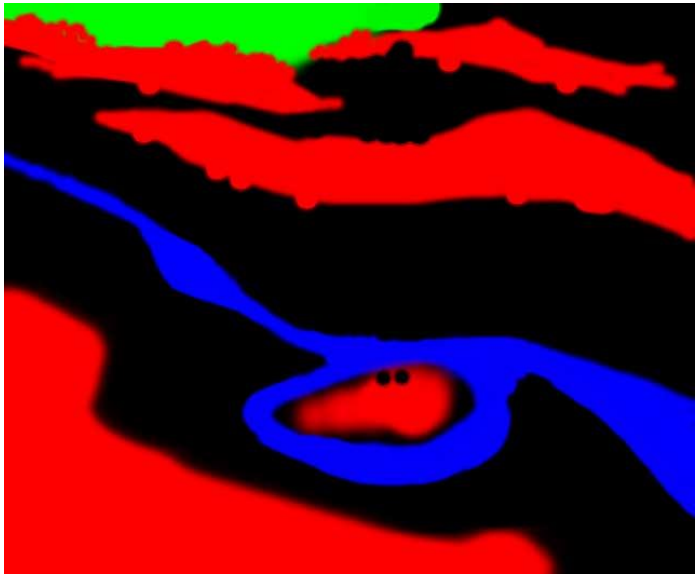
Colorization



Texture-by-Numbers

A

A’



B

B’

Super-Resolution


A


A'

Super-Resolution (result!)

B

B'

Texture Synthesis

a

b

(Image Inpainting & Hole Filling)

References

- Texture Analysis and Synthesis
  Szeliski, Section 10.5
- Efros & Leung, *Texture Synthesis by non-parametric sampling*, ICCV 1999
- Efros & Freeman, *Image Quilting for Texture Synthesis and Transfer*, SIGGRAPH 2001
- Criminisi et al, *Object Removal by Exemplar-based Inpainting*, CVPR 2003
- Barnes et al, PatchMatch, SIGGRAPH 2009
- Hertzman et al, Image Analogies, SIGGRAPH 2001