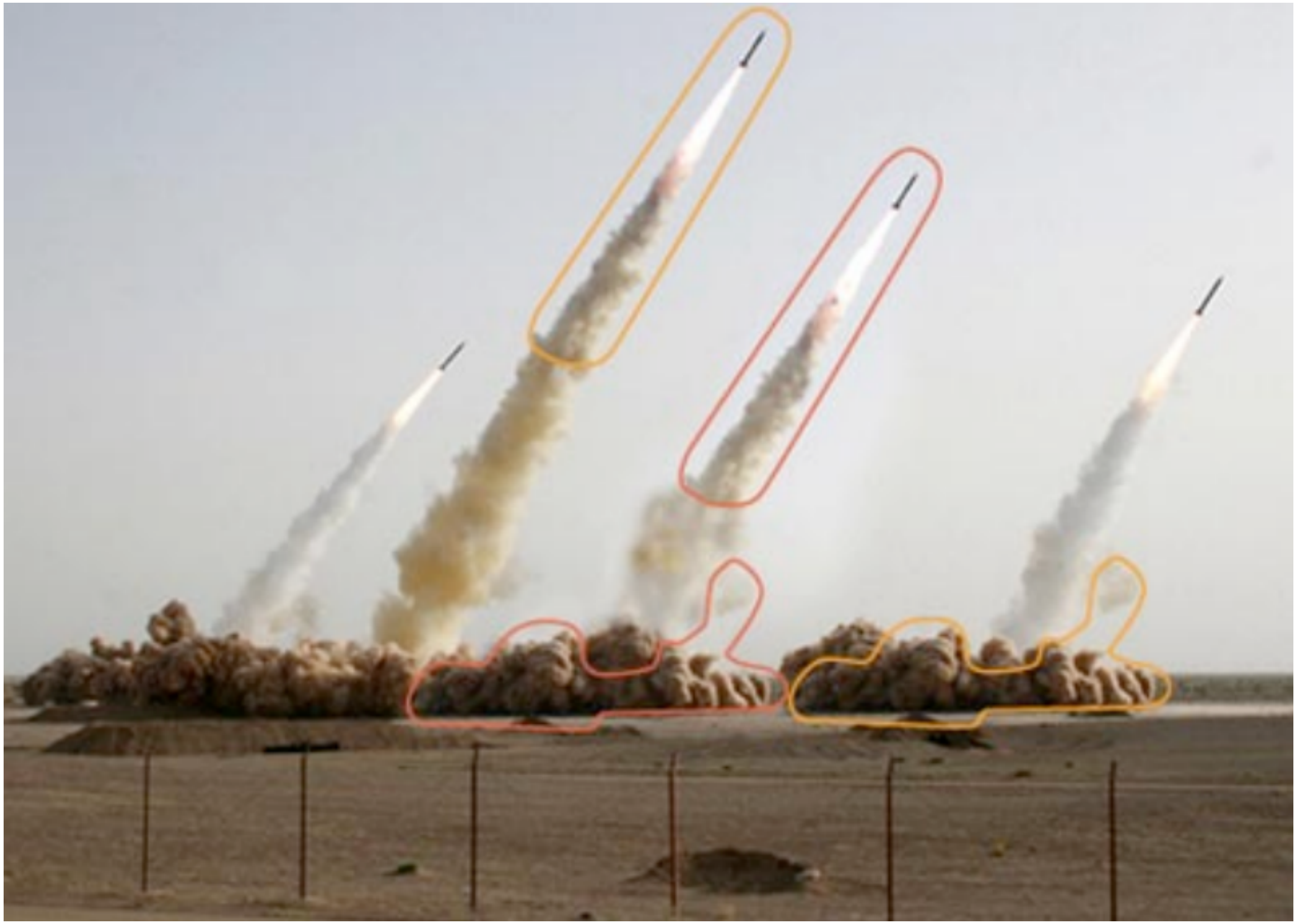
Texture

CS 419

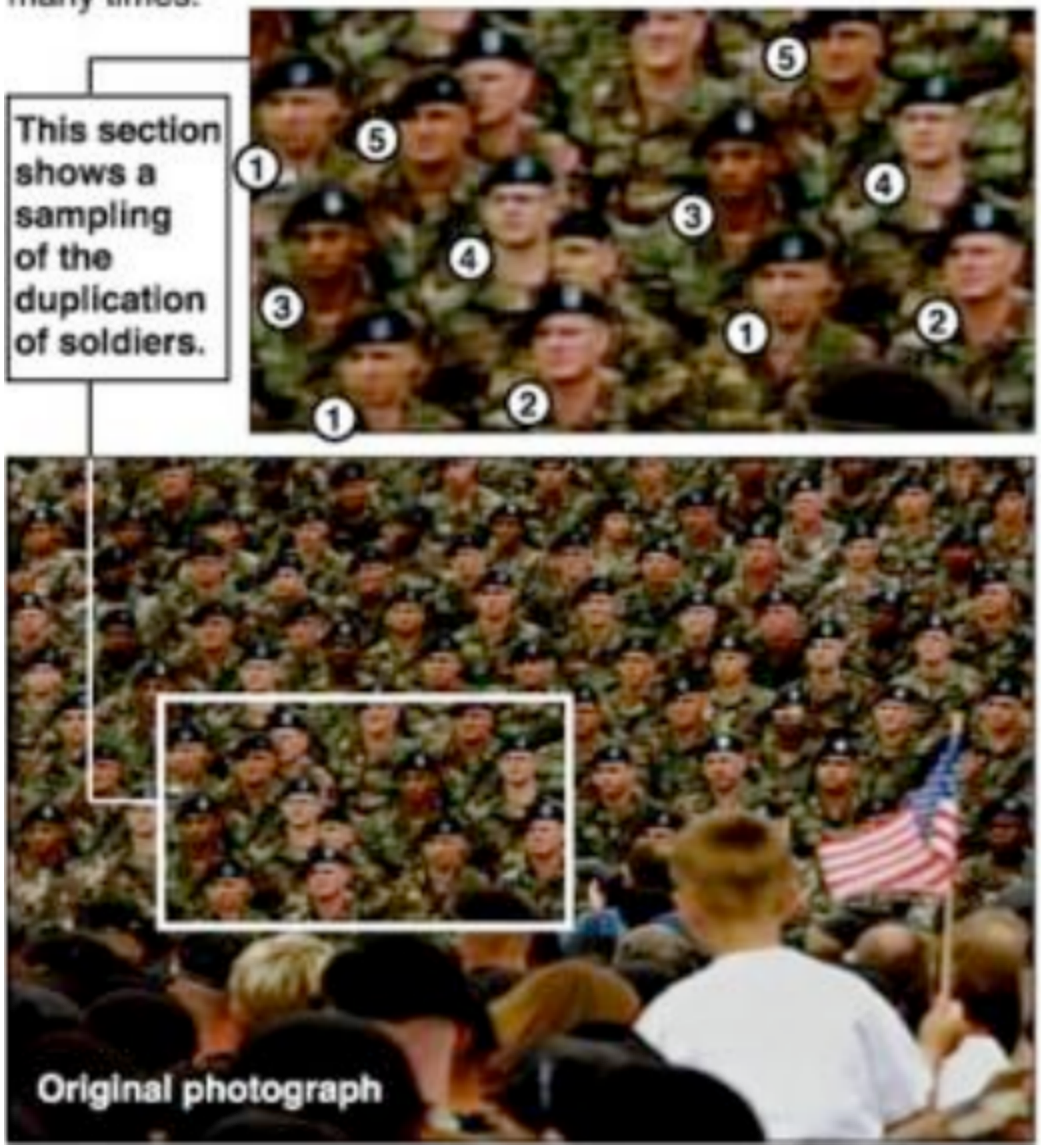
Slides by Zicheng Liao (Courtesy of Ali Farhadi)

Texture scandals

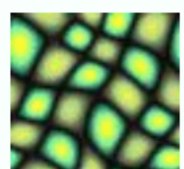


Bush campaign digitally altered TV ad

President Bush's campaign acknowledged Thursday that it had digitally altered a photo that appeared in a national cable television commercial. In the photo, a handful of soldiers were multiplied many times.



What is Texture?

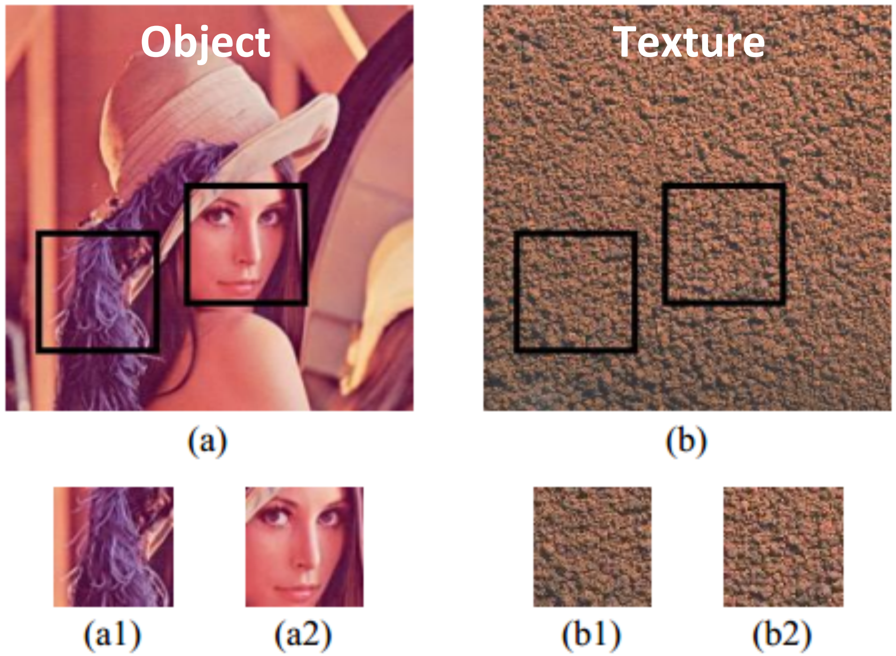


Texture spectrum

Diversity: color, scale, content, caused by geometry or color

Regularity **Randomness**

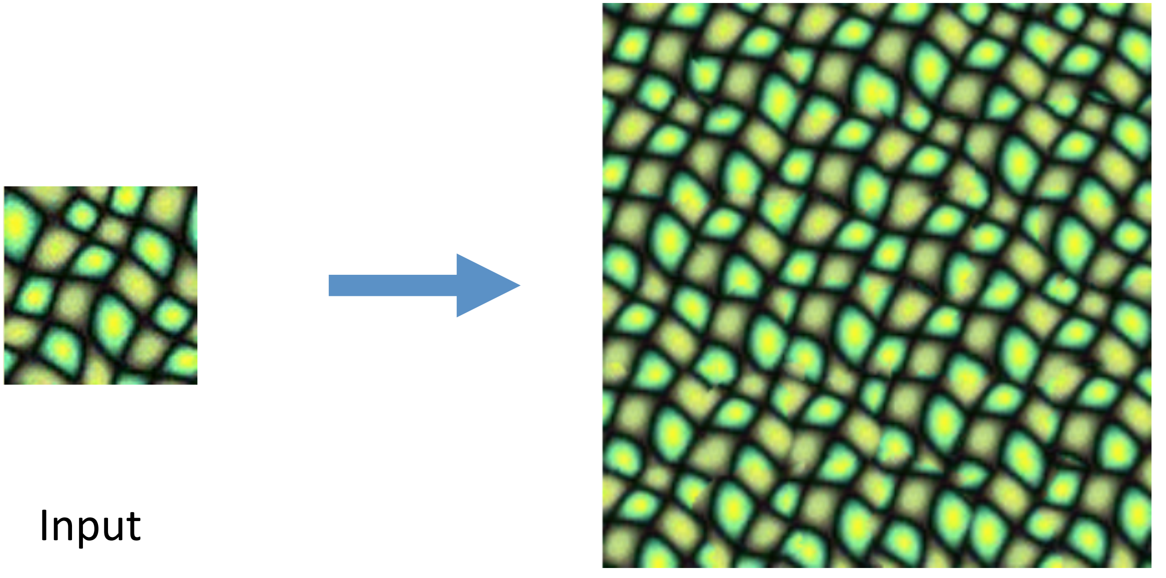
How textures differ from objects



- ***Stationary***: similar views as window moves around in (b), because the *statistical distribution is spatially invariant*.
- ***Locality***: each pixel in (b) is only related to a small set of neighbors.

[Wei & Levoy SIGGRAPH2000]

Texture synthesis



Input

Output

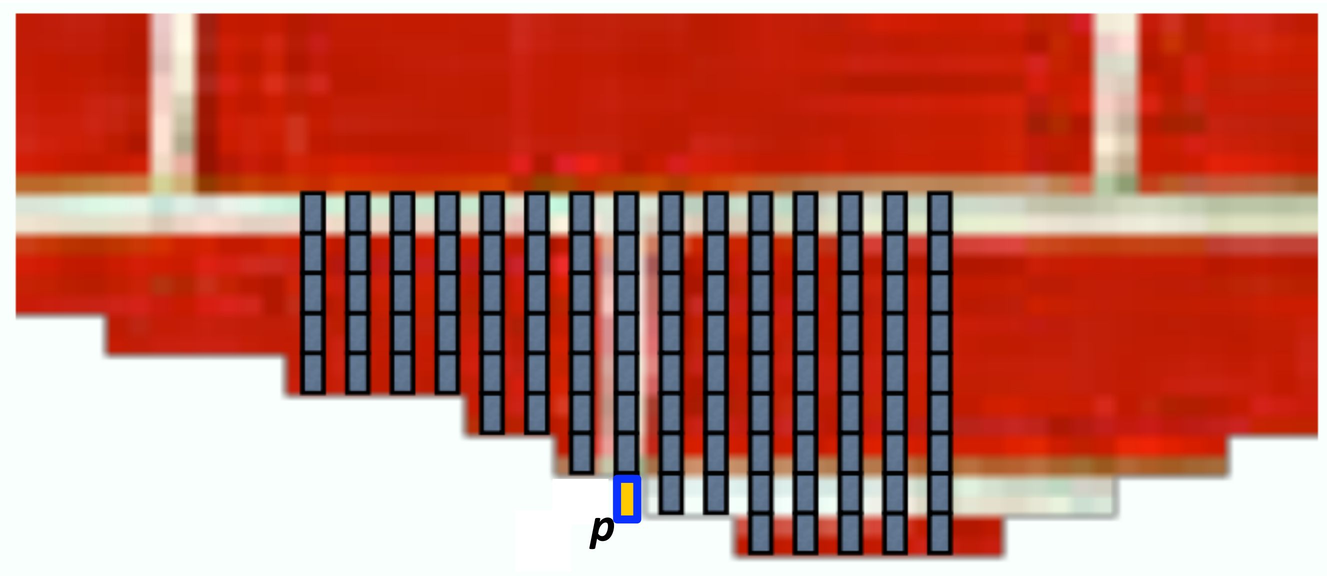
Two crucial algorithmic points

- Nearest neighbor search
- Dynamic programming

Algorithm I: NN + sampling

- How to determine a unknown pixel value?
  - **Ask neighbors** (locality)

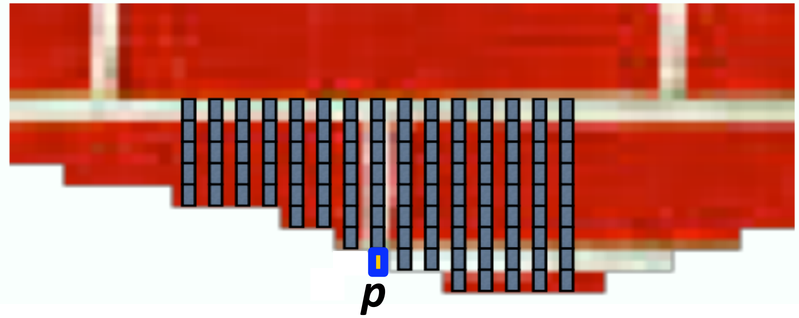
- Find pixels with similar neighbors in the input, and then randomly take one

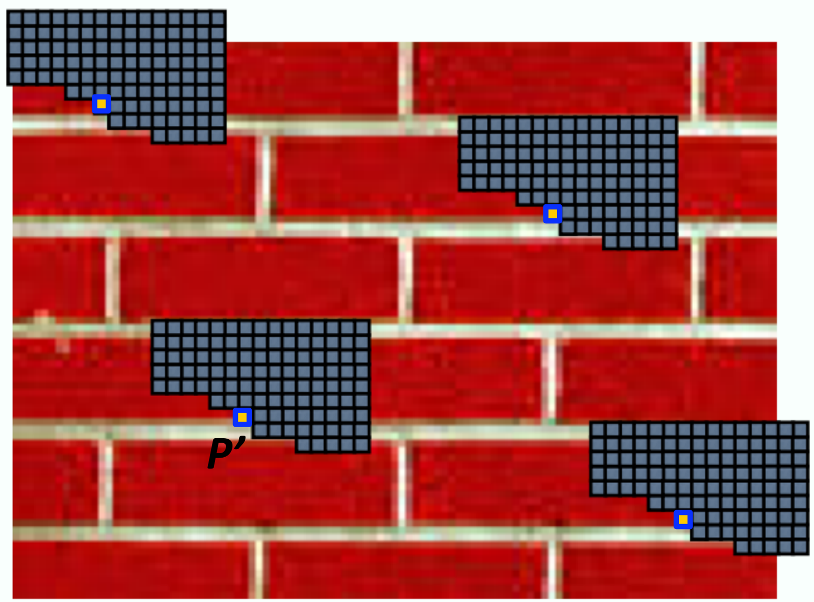


[Efros & Leung ICCV99]

NN patch match

- Find a few pixels with similar neighbors



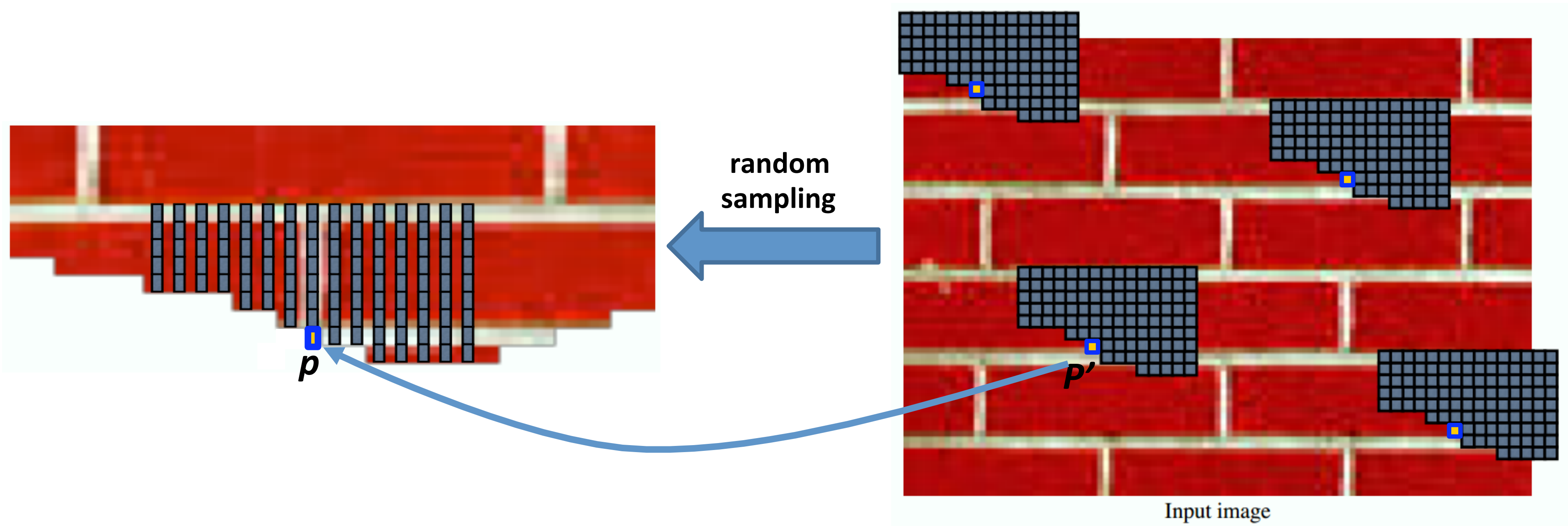


Input image

[Efros & Leung ICCV99]

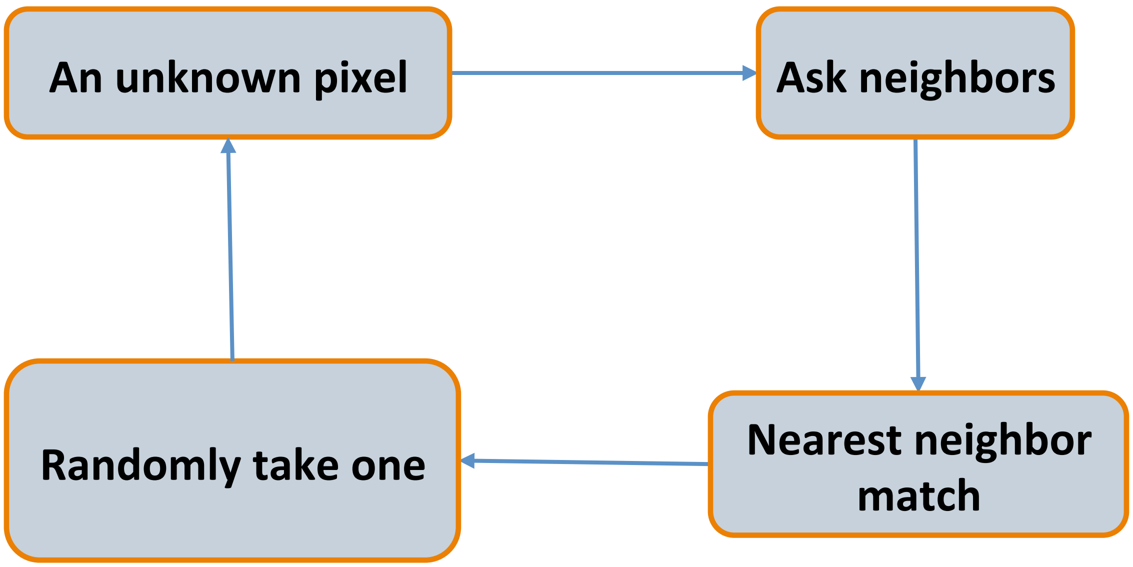
Randomly Sample

- Find a few pixels with similar neighbors
- Randomly take one matched pixel



[Efros & Leung ICCV99]

The steps in a single iteration



[Efros & Leung ICCV99]

Algorithm

1. Grow from the border of an input texture
2. For each unknown pixel $\boldsymbol{p}$ on the boundary
3. Gather the neighborhood centered at $\boldsymbol{p}$: $\boldsymbol{N}(\boldsymbol{p})$
4. Find patches $\boldsymbol{N}(\boldsymbol{p}')$ from input: $\boldsymbol{d}(\boldsymbol{N}(\boldsymbol{p}), \boldsymbol{N}(\boldsymbol{p}')) < (\boldsymbol{1} + \epsilon)\boldsymbol{d}(\boldsymbol{N}(\boldsymbol{p}), \boldsymbol{N}_{\boldsymbol{best}})$
5. Randomly take a $\boldsymbol{p}'$ to fill in $\boldsymbol{p}$

\epsilon = 0.1

[Efros & Leung ICCV99]

Discussion

1. Grow from the border of an input texture
2. For each unknown pixel $\boldsymbol{p}$ on the boundary
3. Gather the neighborhood centered at $\boldsymbol{p}$: $\boldsymbol{N}(\boldsymbol{p})$
4. Find patches $\boldsymbol{N}(\boldsymbol{p}')$ from input: $\boldsymbol{d}(\boldsymbol{N}(\boldsymbol{p}), \boldsymbol{N}(\boldsymbol{p}')) < (\boldsymbol{1} + \epsilon)\boldsymbol{d}(\boldsymbol{N}(\boldsymbol{p}), \boldsymbol{N}_{\boldsymbol{best}})$
5. Randomly take a $\boldsymbol{p}'$ to fill in $\boldsymbol{p}$

\epsilon = 0.1

- How does the choice of \epsilon effect the result?
  - Not much, some algorithm even set it to 0.
- Other than random sample?
  - Use the distance to bias the sampling, favor better matches

[Efros & Leung ICCV99]

Discussion (2)

1. Grow from the border of an input texture
2. For each unknown pixel $\boldsymbol{p}$ on the boundary
3. Gather the neighborhood centered at $\boldsymbol{p}$: $\boldsymbol{N}(\boldsymbol{p})$
4. Find patches $\boldsymbol{N}(\boldsymbol{p}')$ from input: $\boldsymbol{d}(\boldsymbol{N}(\boldsymbol{p}), \boldsymbol{N}(\boldsymbol{p}')) < (\mathbf{1} + \epsilon)\boldsymbol{d}(\boldsymbol{N}(\boldsymbol{p}), \boldsymbol{N}_{\boldsymbol{best}})$
5. Randomly take a $\boldsymbol{p}'$ to fill in $\boldsymbol{p}$

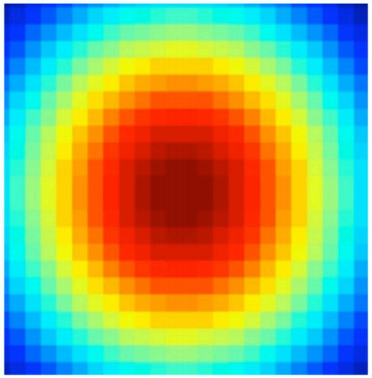
\epsilon = 0.1

- Distance metric
- Window size
- Order to synthesize

[Efros & Leung ICCV99]

Patch distance metric

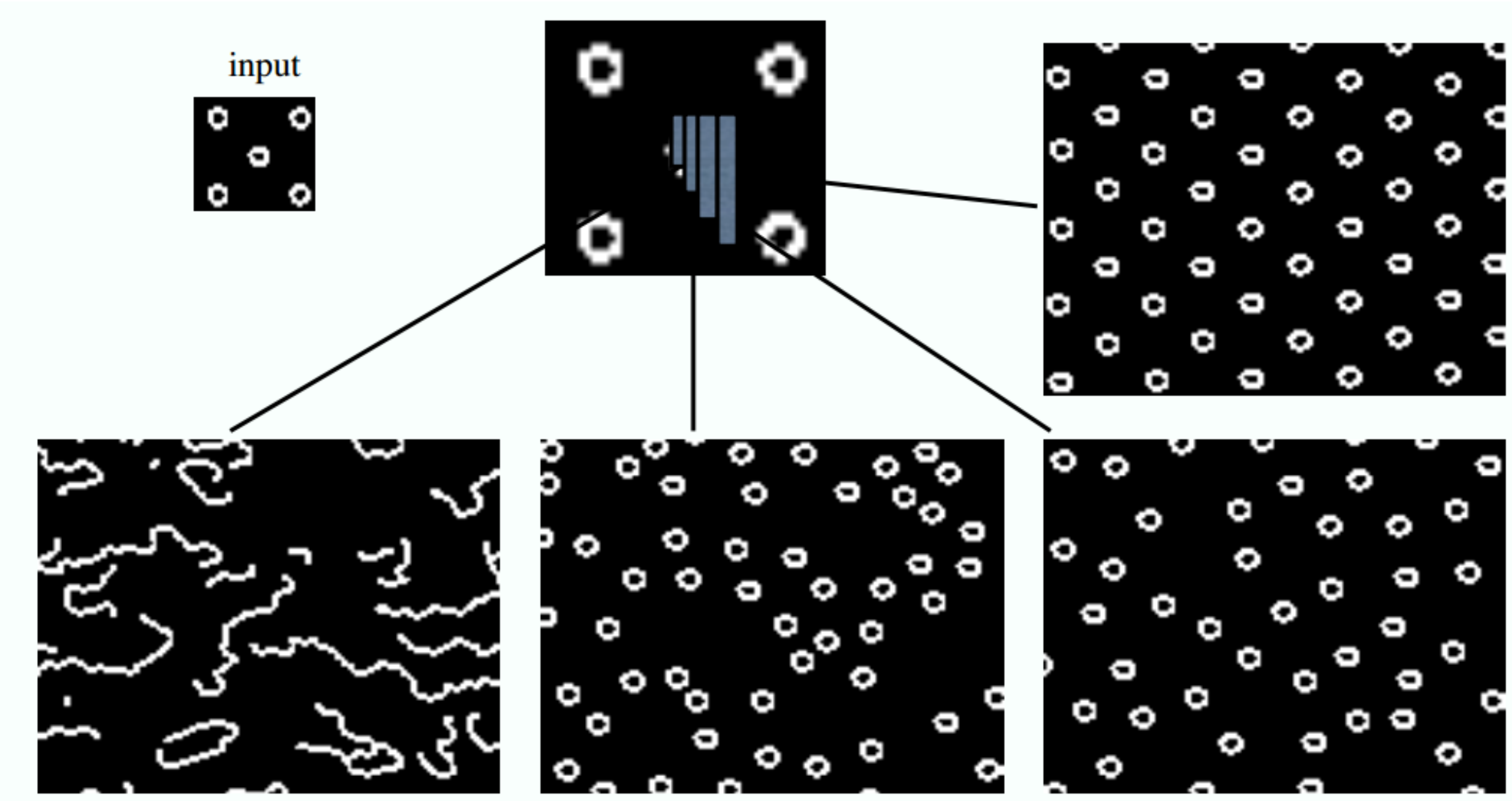
- Normalized sum of squared difference
- Further neighbors take less weight
  - Gaussian mask
  - Preserve local structure



[Efros & Leung ICCV99]

Window Size

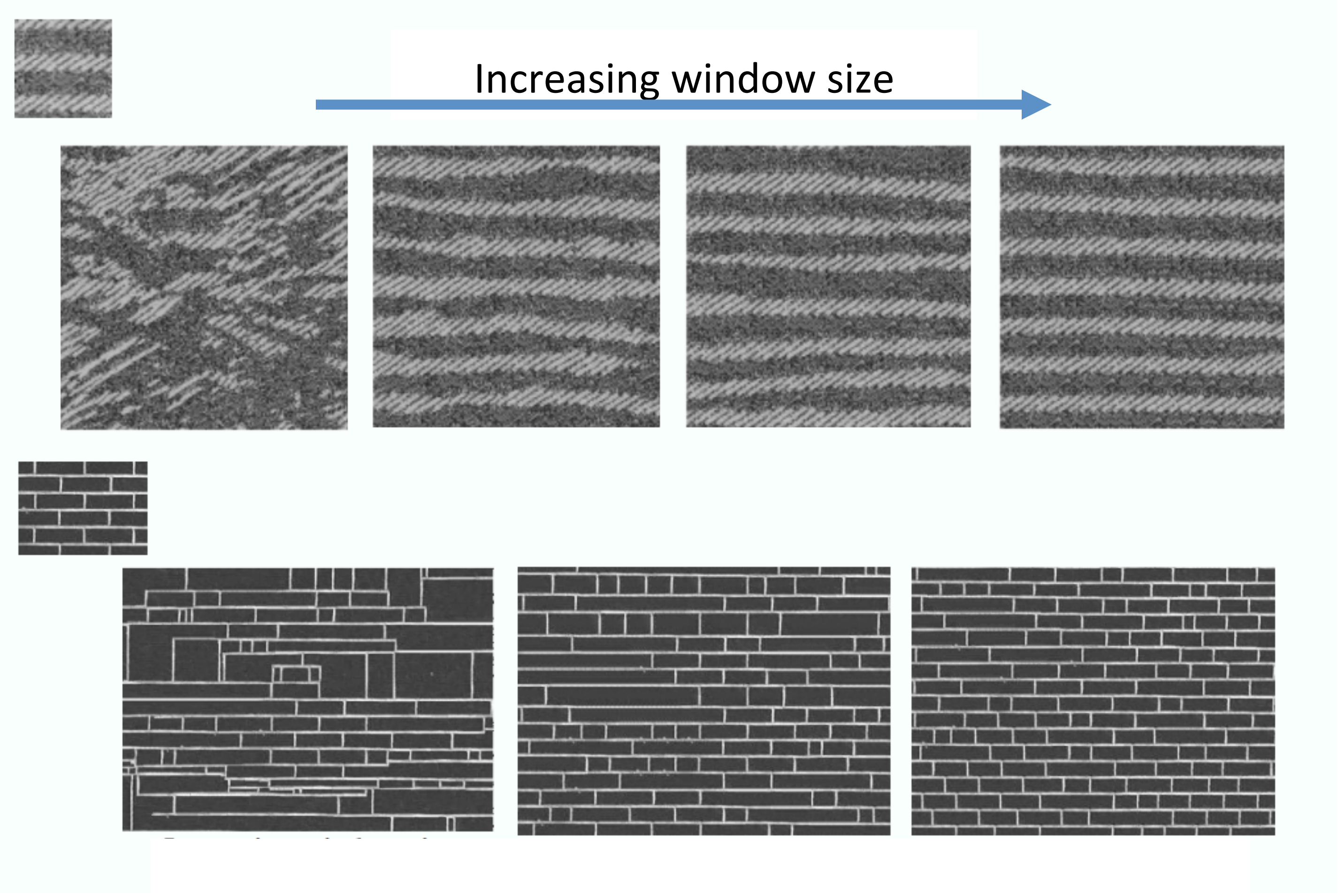
- **Effect of window size on the results**



[Efros & Leung ICCV99]

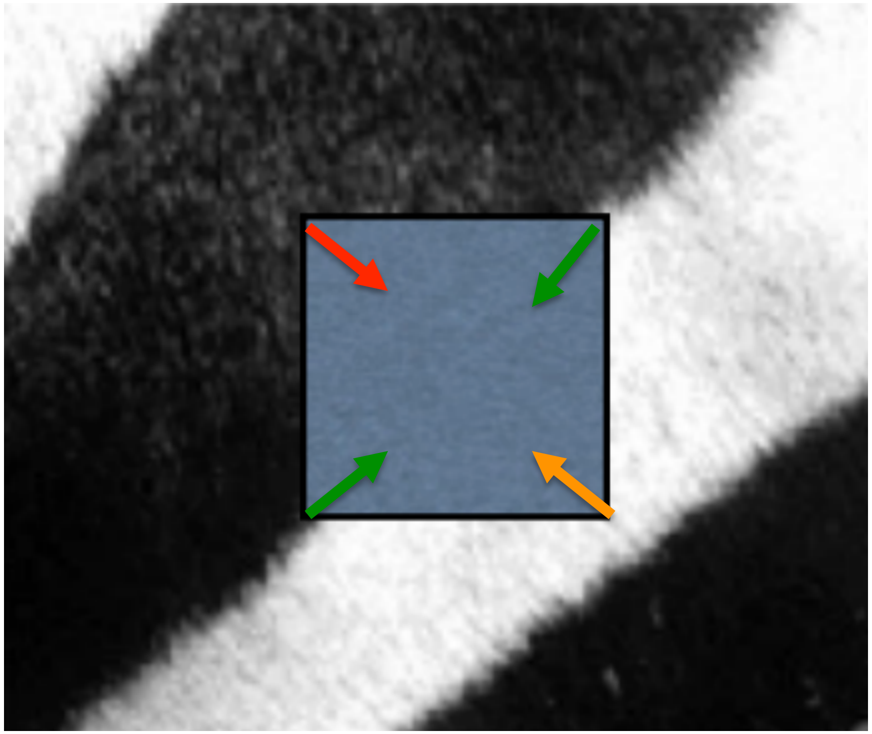
Window Size

- **Control the degree of randomness**



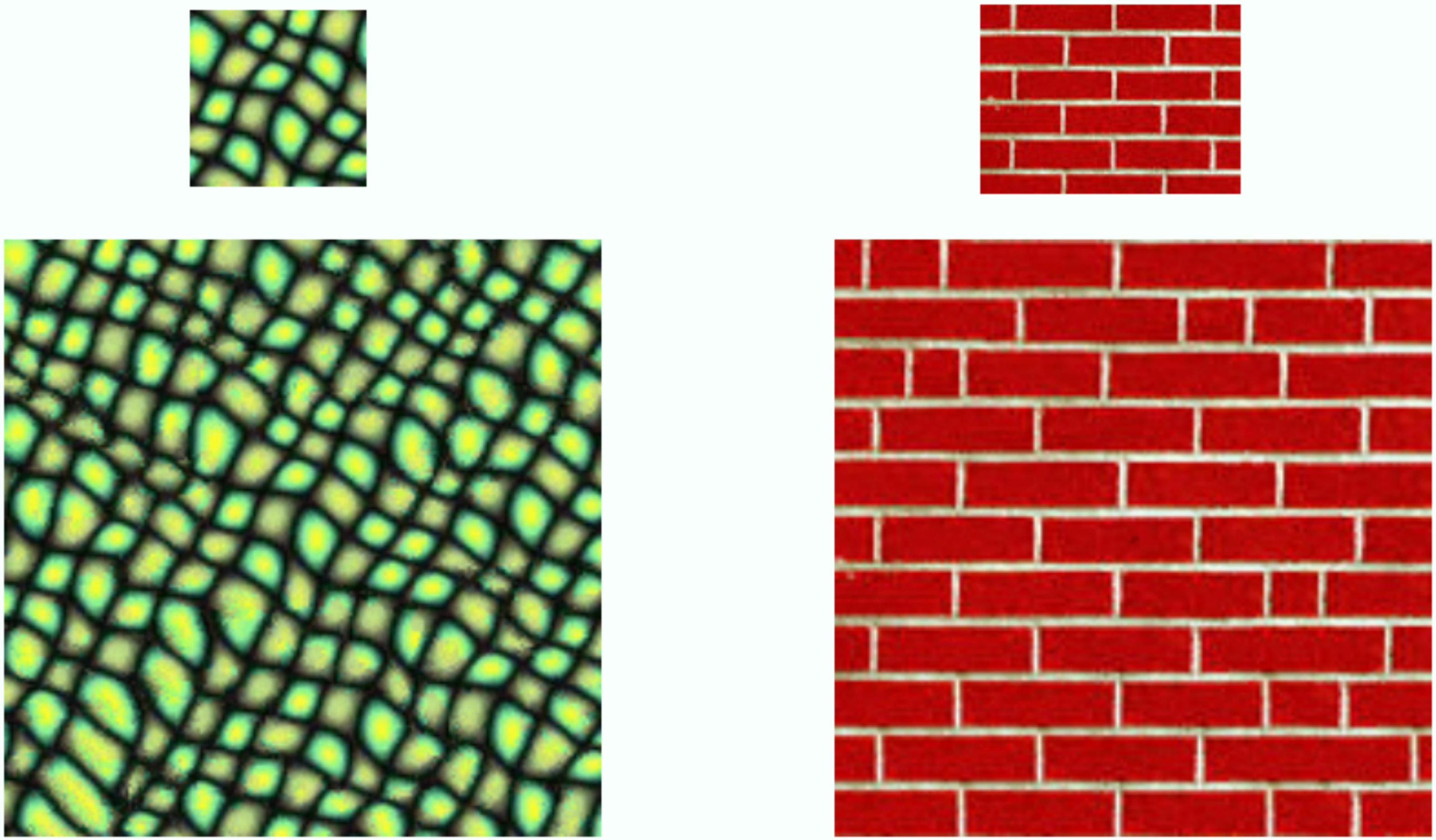
[Efros & Leung ICCV99]

The order matters



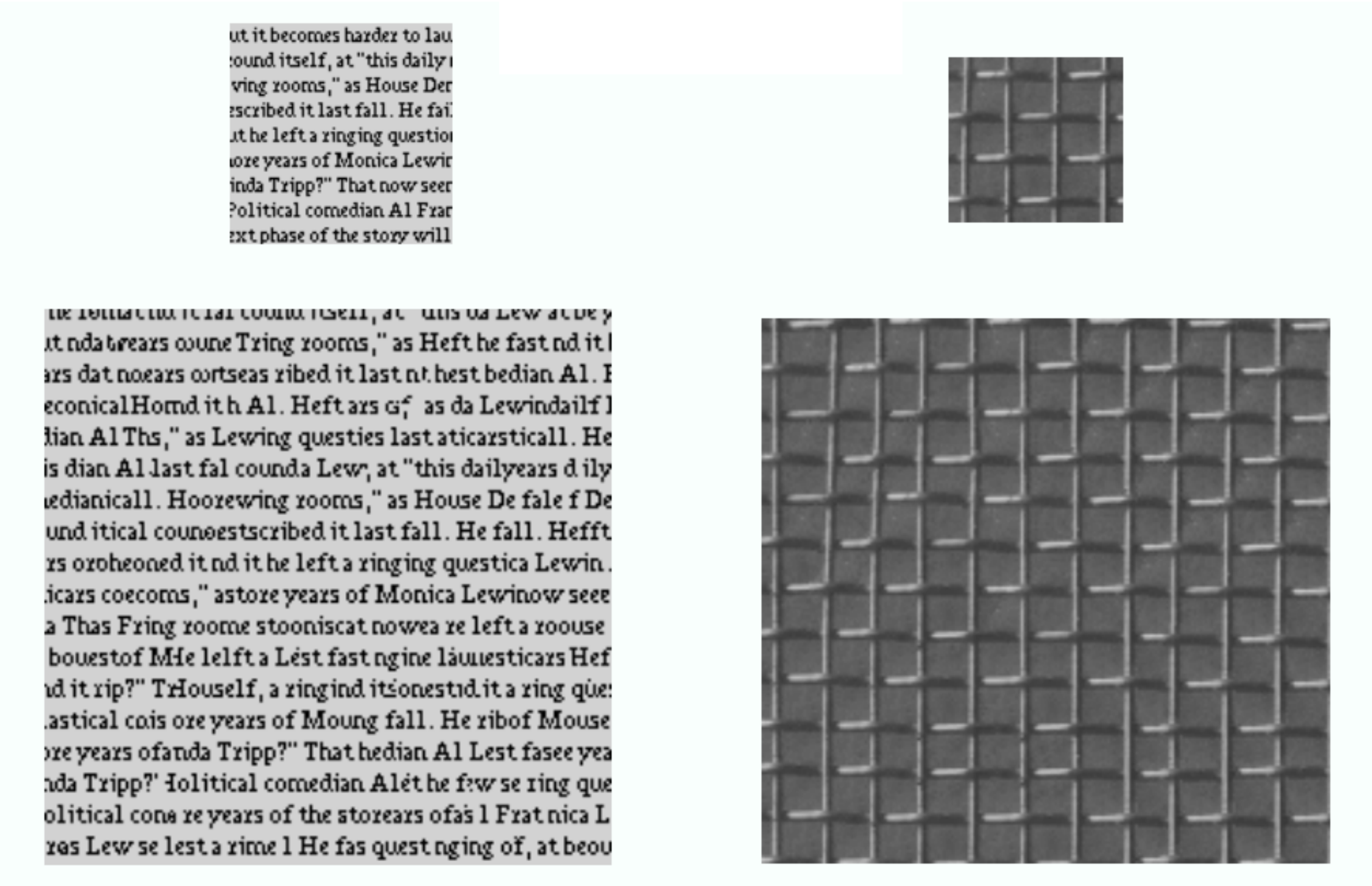
[Efros & Leung ICCV99]

Some results



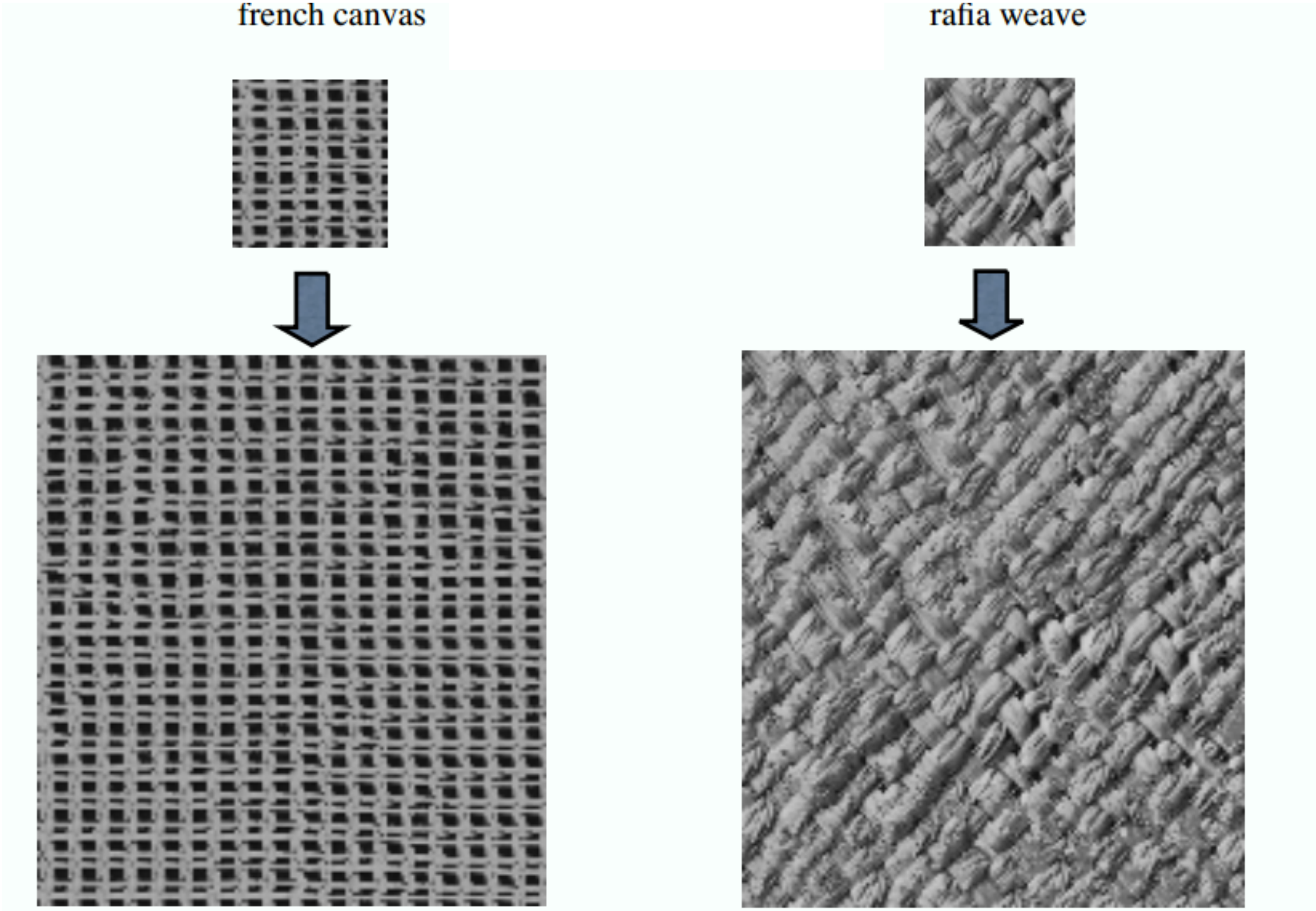
[Efros & Leung ICCV99]

More results



[Efros & Leung ICCV99]

More results

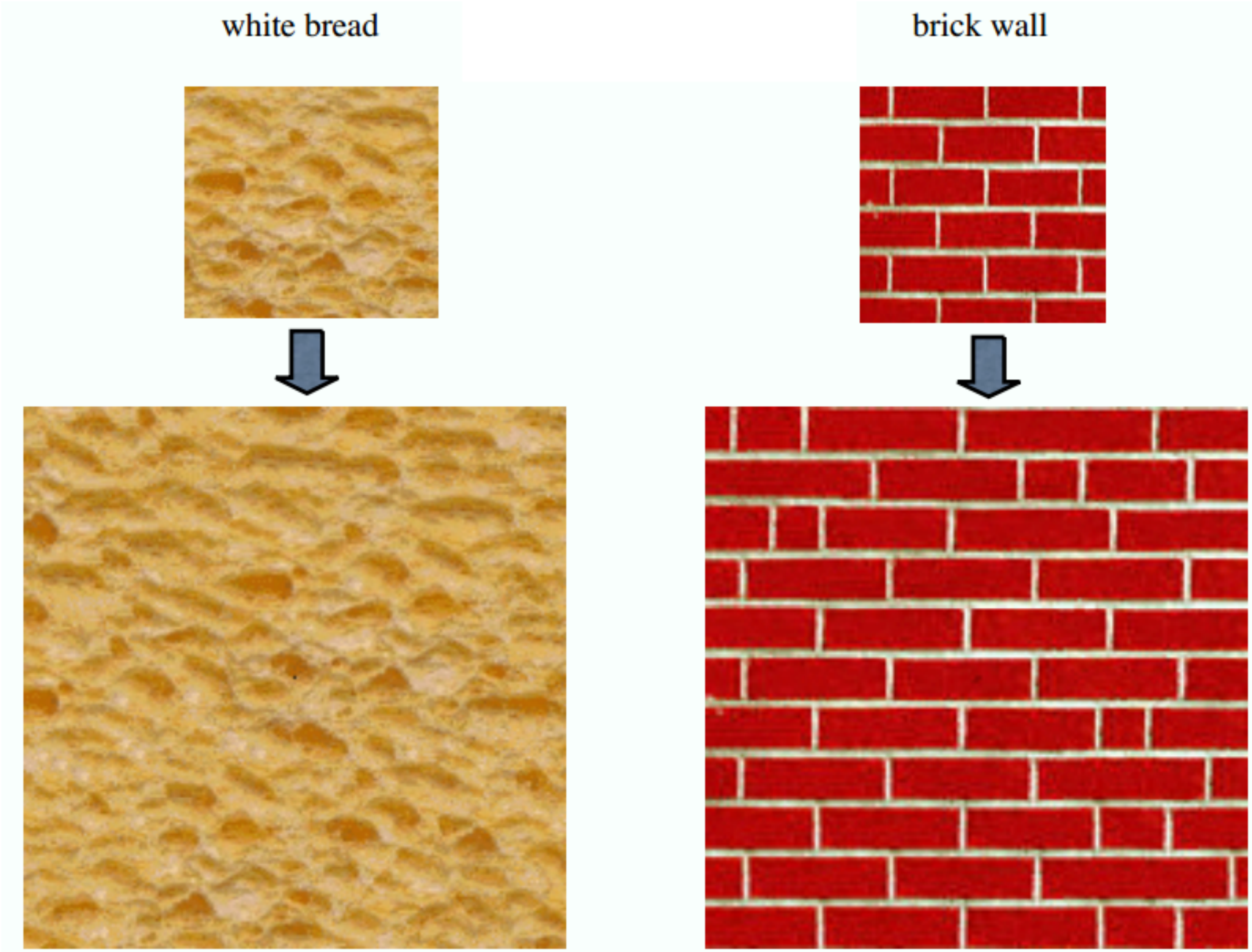


[Efros & Leung ICCV99]

Some results

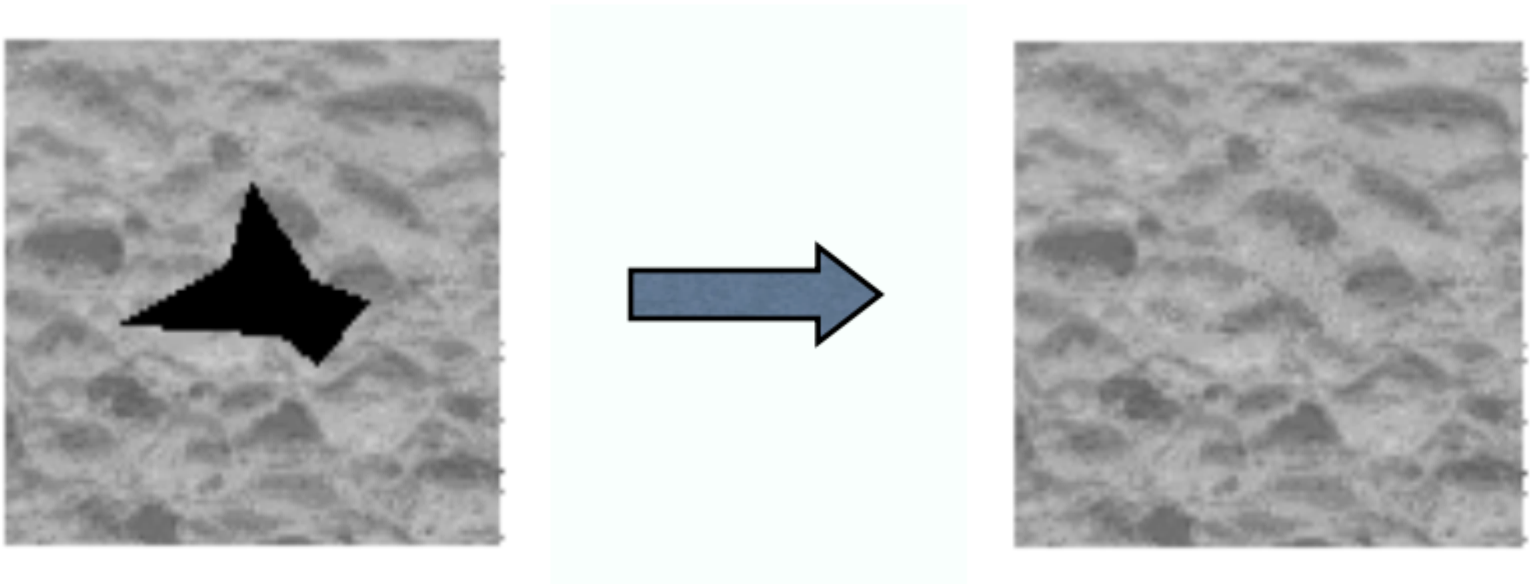
[Efros & Leung ICCV99]

Some results



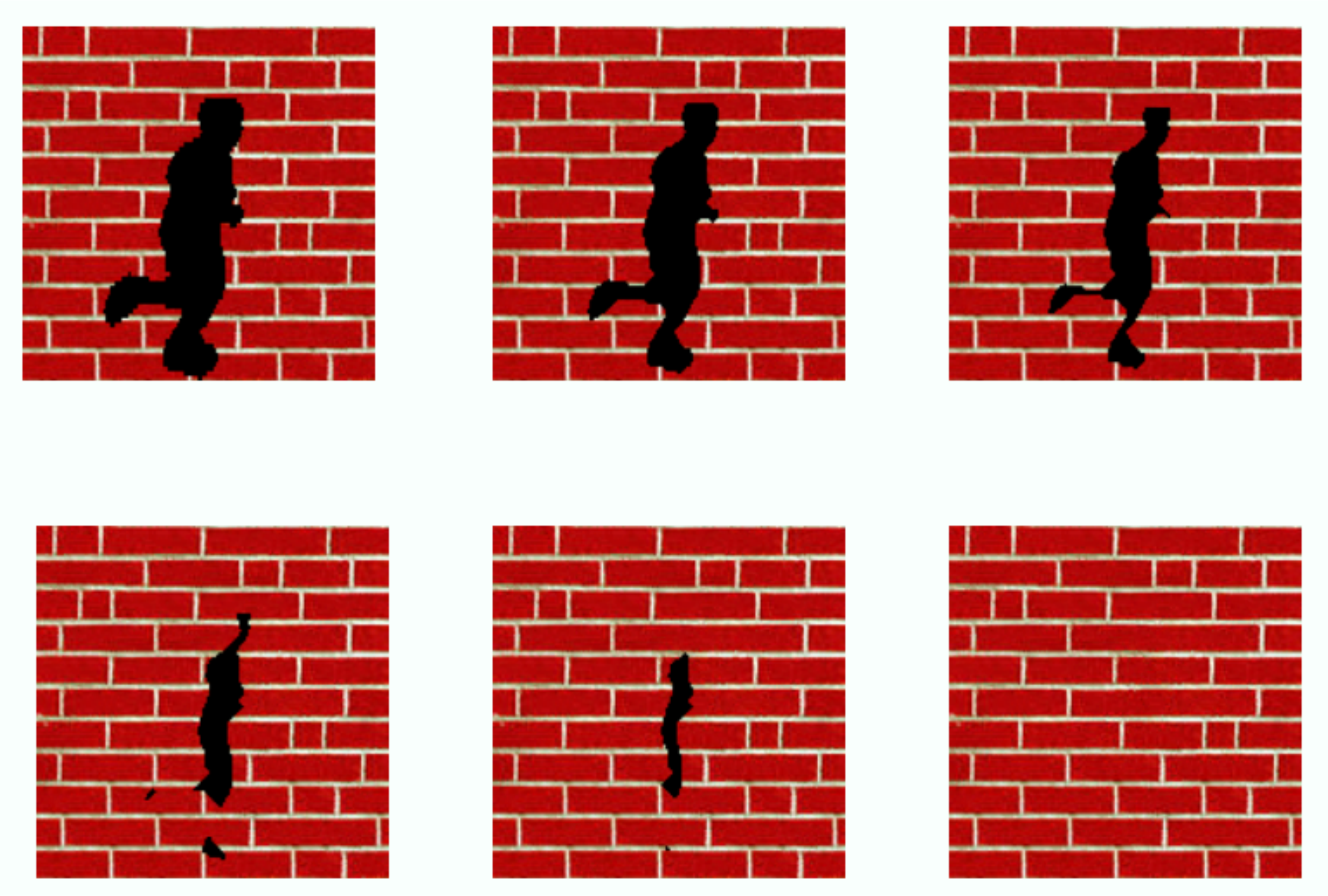
[Efros & Leung ICCV99]

Hole Filling



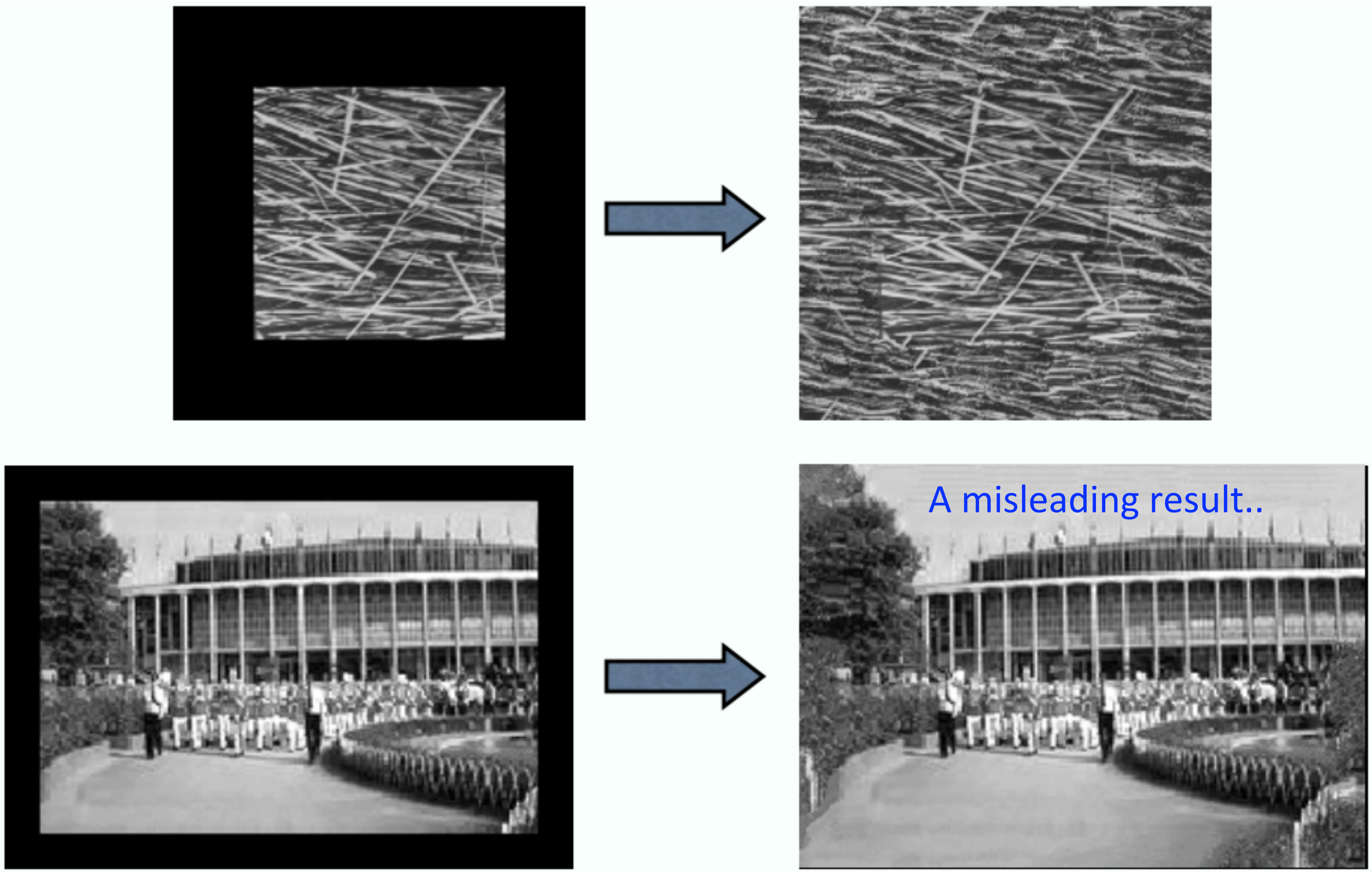
[Efros & Leung ICCV99]

Hole Filling



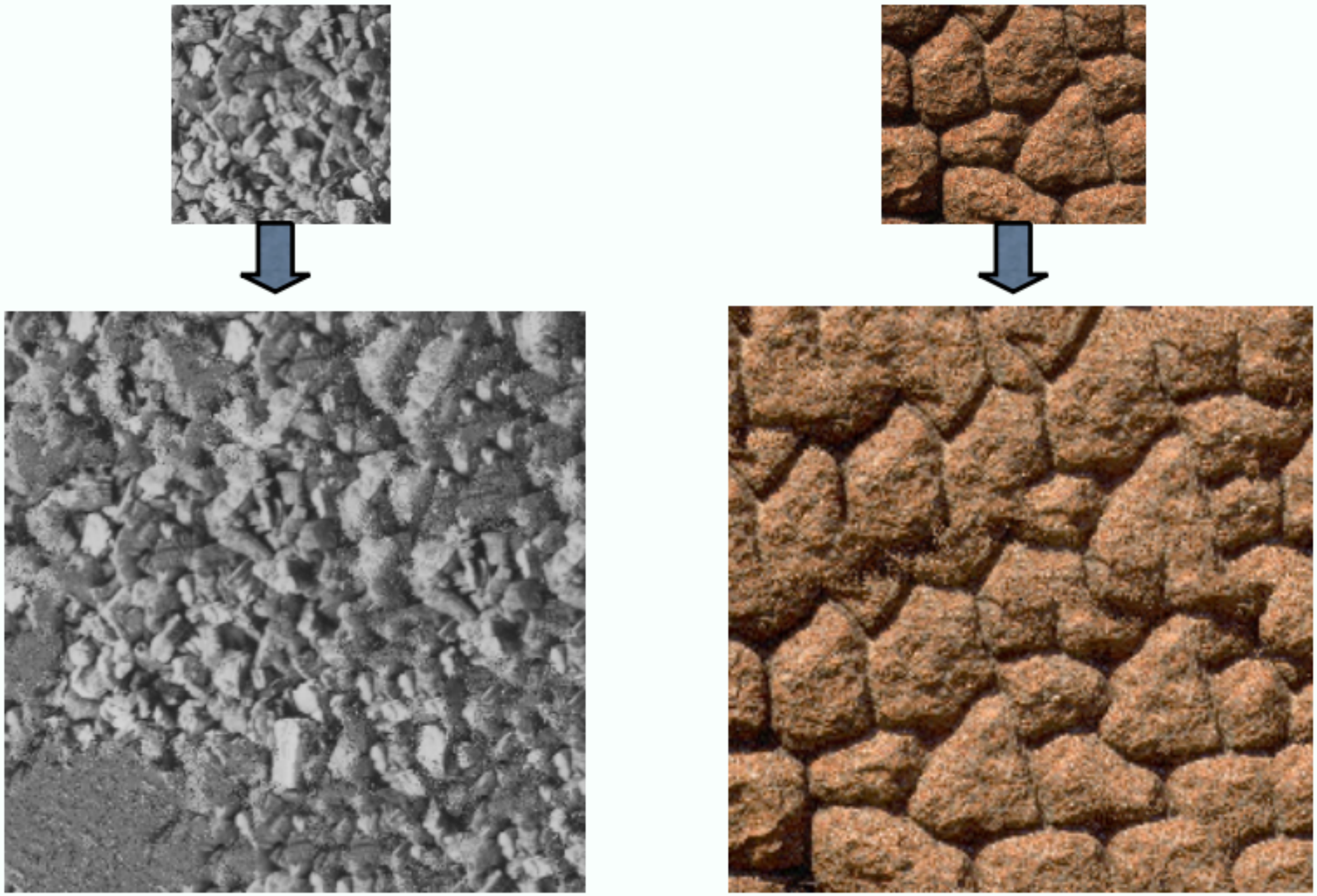
[Efros & Leung ICCV99]

Extrapolation



[Efros & Leung ICCV99]

Failure Cases



Growing garbage   Verbatim copying

[Efros & Leung ICCV99]

Pros and Cons

- Very simple
- Easy to implement: 32 lines of matlab code!
- Works well for a variety of synthetic and real-world textures

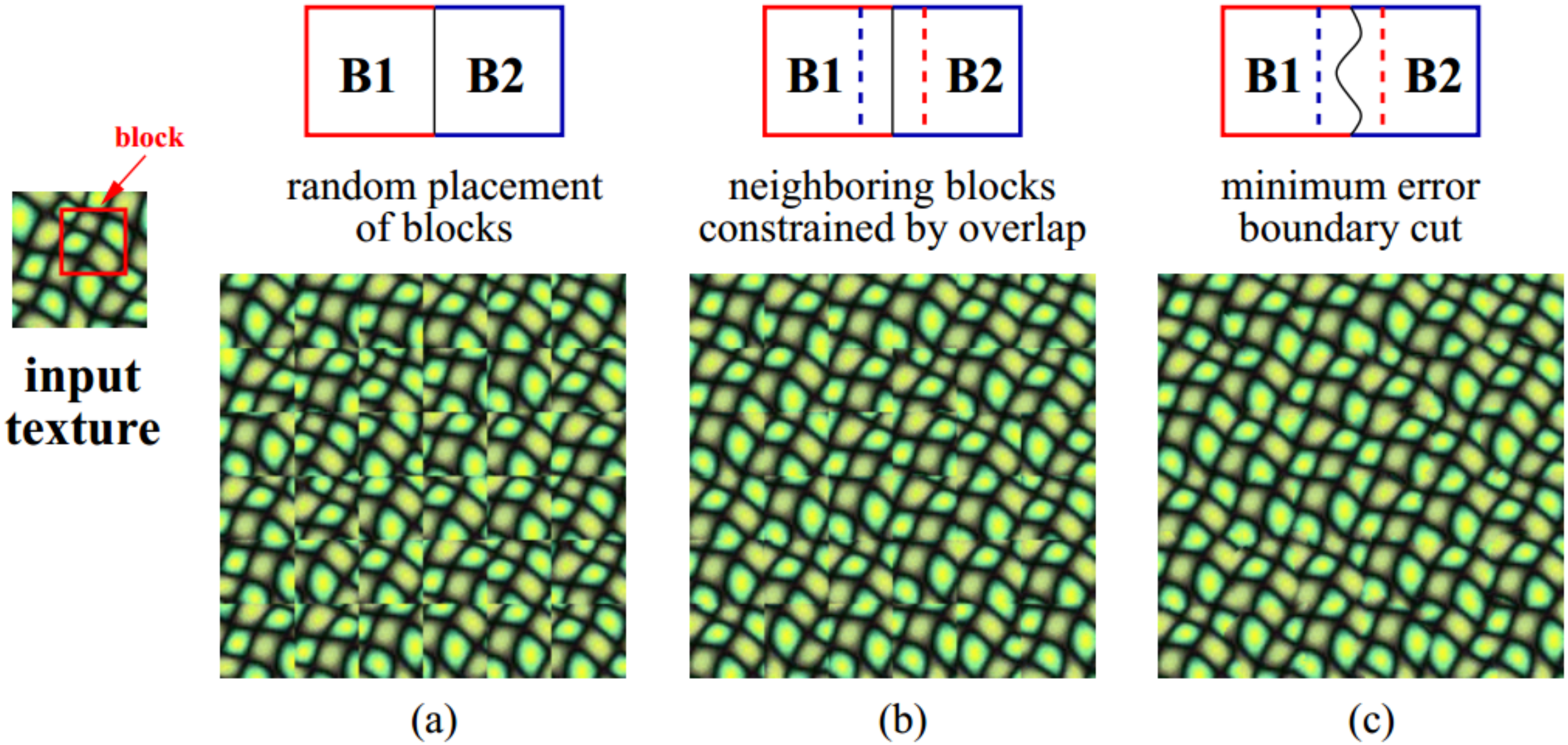
- VERY VERY slow!

  (A nearly identical idea was proposed in 1981 by Barber but discarded due to computational intractability)

- Idea
  - **A patch a time**, instead of a pixel

[Efros & Leung ICCV99]

Image Quilting: Patch-based method



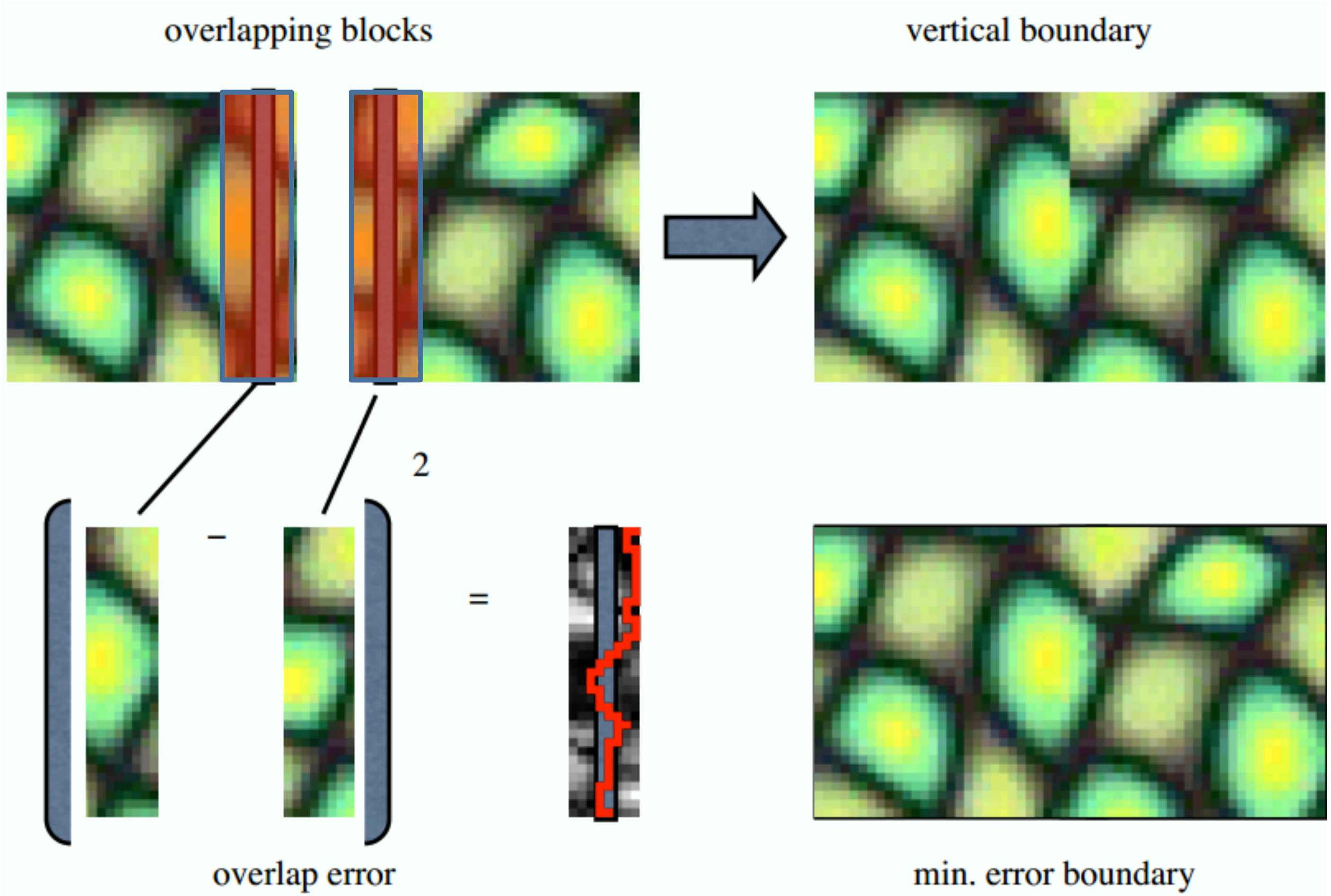
(a) random blocks concatenated together

(b) Blocks overlap, new block is chosen so that the overlap regions best agree

(c) A minimum cost (optimal) path is computed within the overlap

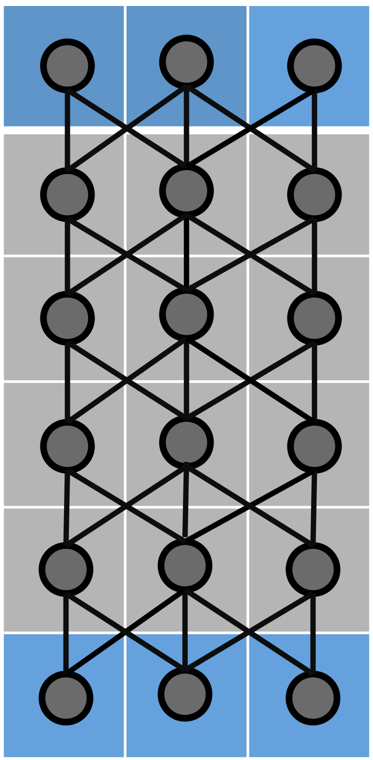
[Efros & Freeman SIGGRAPH01]

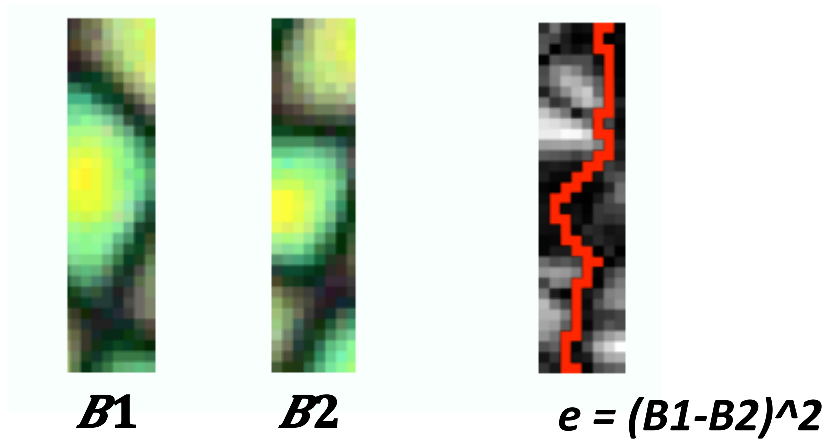
Curved path VS vertical path



[Efros & Freeman SIGGRAPH01]

How to find the optimal path?





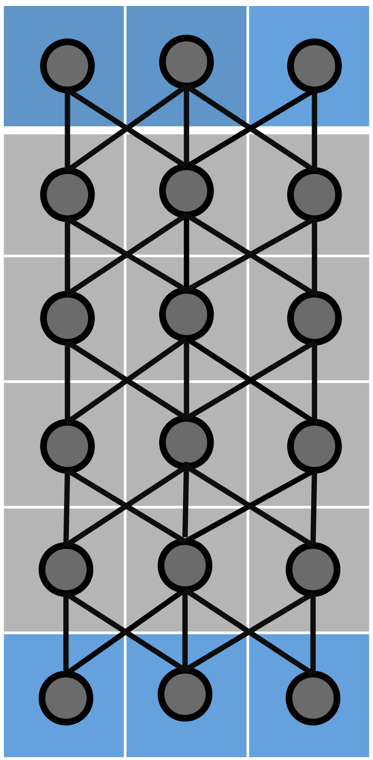
- Brute force: exponential number of paths
- Greedy algorithm? No..
- Key observation: **every optimal sub-path is part of an optimal full path**

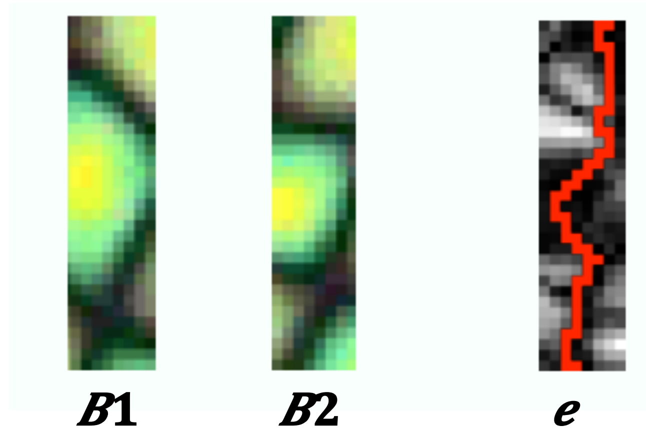
  ➔ ***Dynamic programming***

http://community.topcoder.com/tc?module=Static&d1=tutorials&d2=dynProg
(A nice dynamic programming tutorial)

[Efros & Freeman SIGGRAPH01]

Dynamic programming





Initialize: $e_{i,j} = \left(B1_{ij} - B2_{ij}\right)^2$

for $i = 2{:}h$; for $j = 1{:}w$

$$E_{i,j} = e_{ij} + min\left(E_{i-1,j-1}, E_{i-1,j}, E_{i-1,j+1}\right)$$

$$k_{i,j} = argmin\left(E_{i-1,j-1}, E_{i-1,j}, E_{i-1,j+1}\right)$$

end; end

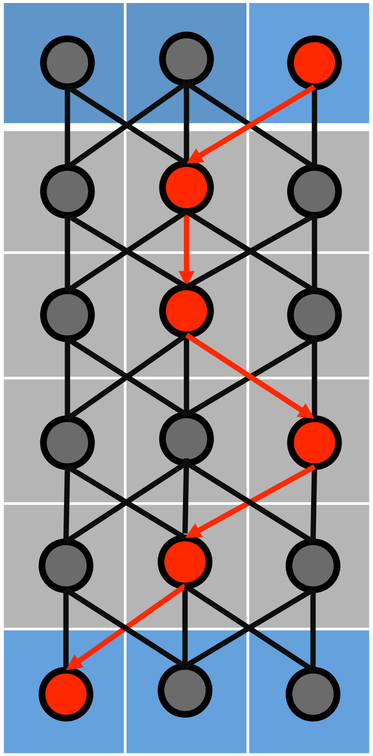
e_{i,j}: node cost at pixel (i,j)

E_{i, j}: optimal path cost up to node (i, j)

k_{i,j}: index to the optimal (next) sub-path

[Efros & Freeman SIGGRAPH01]

Dynamic programming

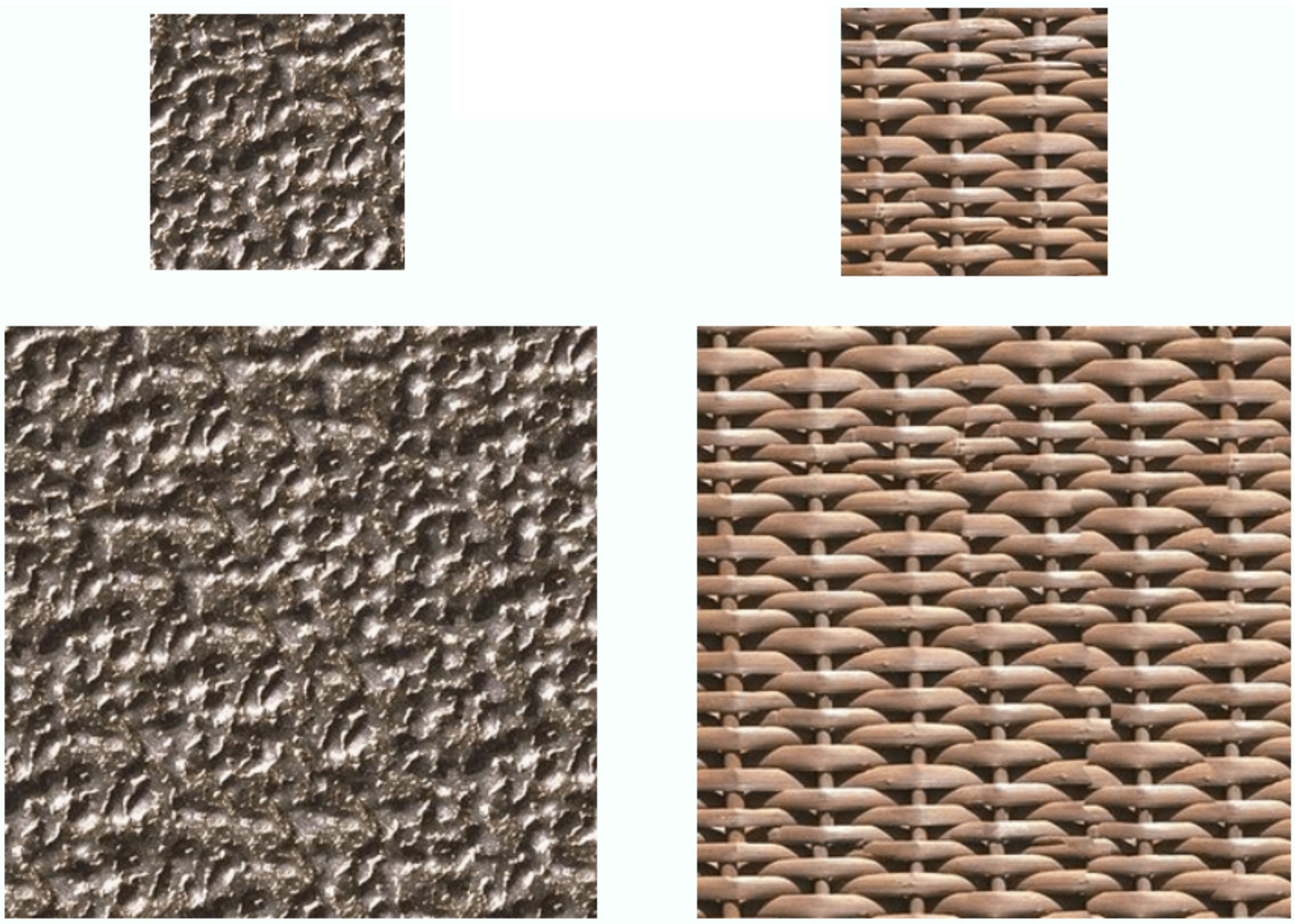


1. Compute path costs: start from the bottom, iteratively go up, end at the top

2. Get optimal path: compare the path cost of nodes on the top row, find the minimum cost node, and use k_{i,j} to trace back for the optimal path down to the bottom

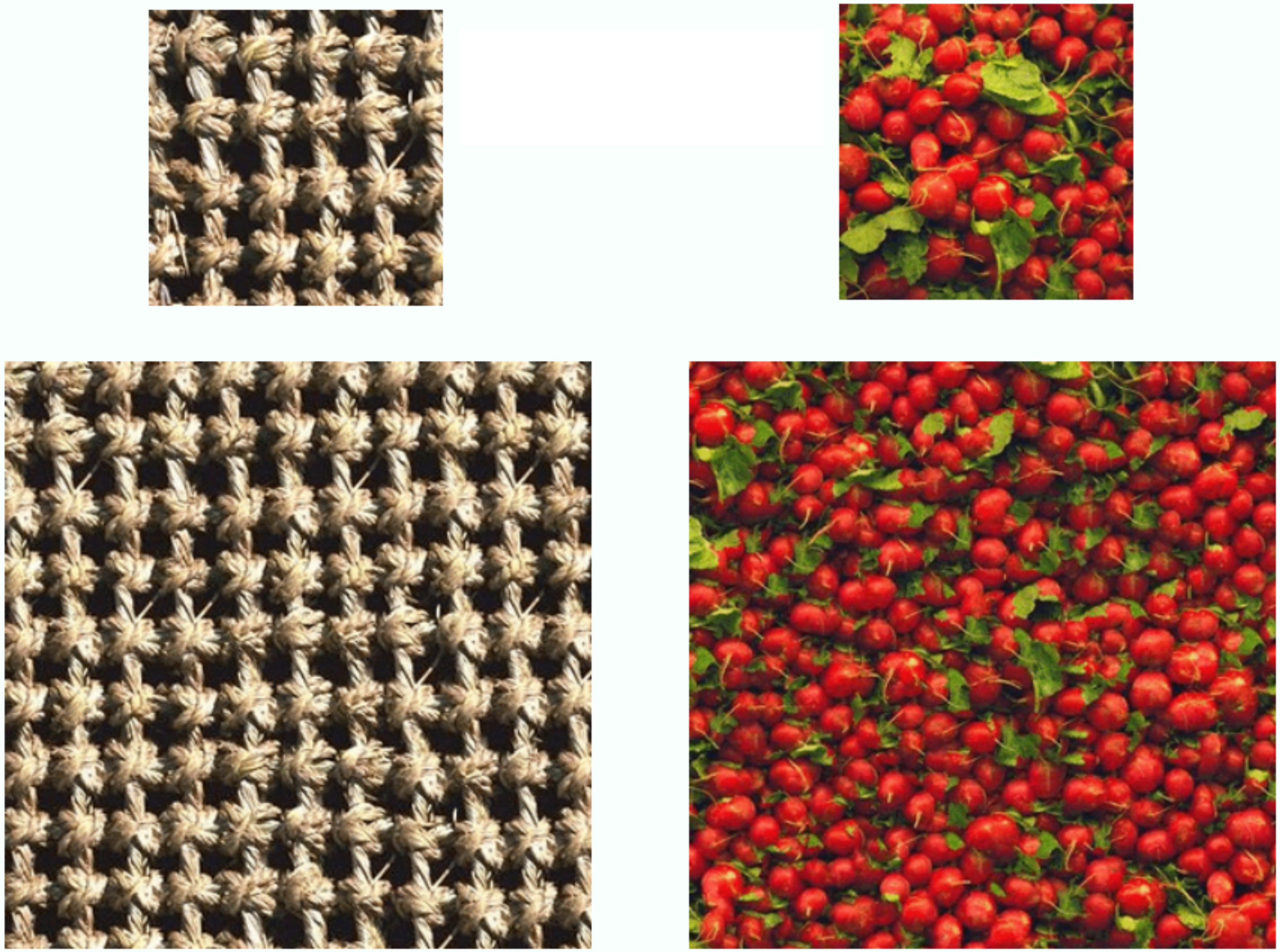
[Efros & Freeman SIGGRAPH01]

Results



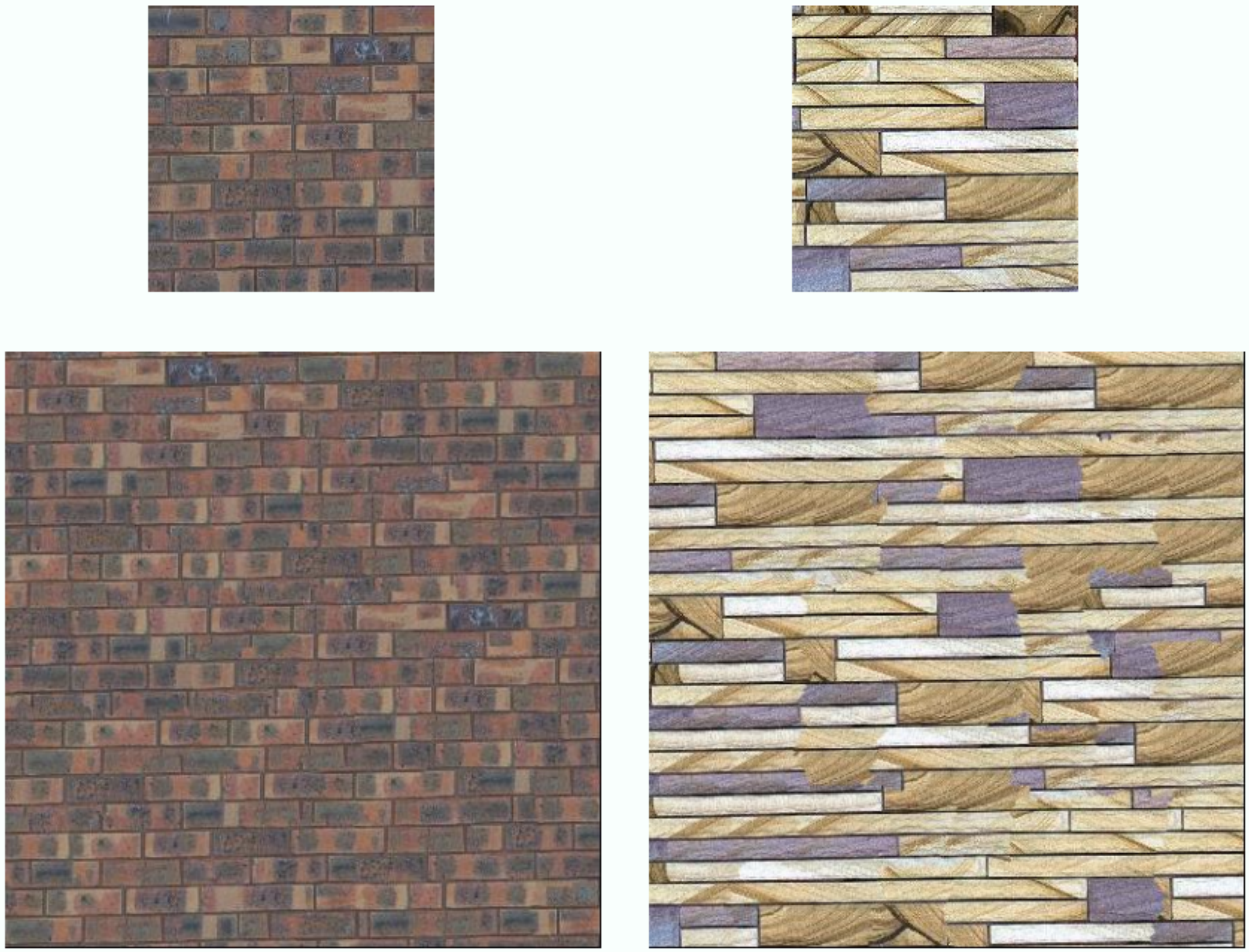
[Efros & Freeman SIGGRAPH01]

More results



[Efros & Freeman SIGGRAPH01]

More results

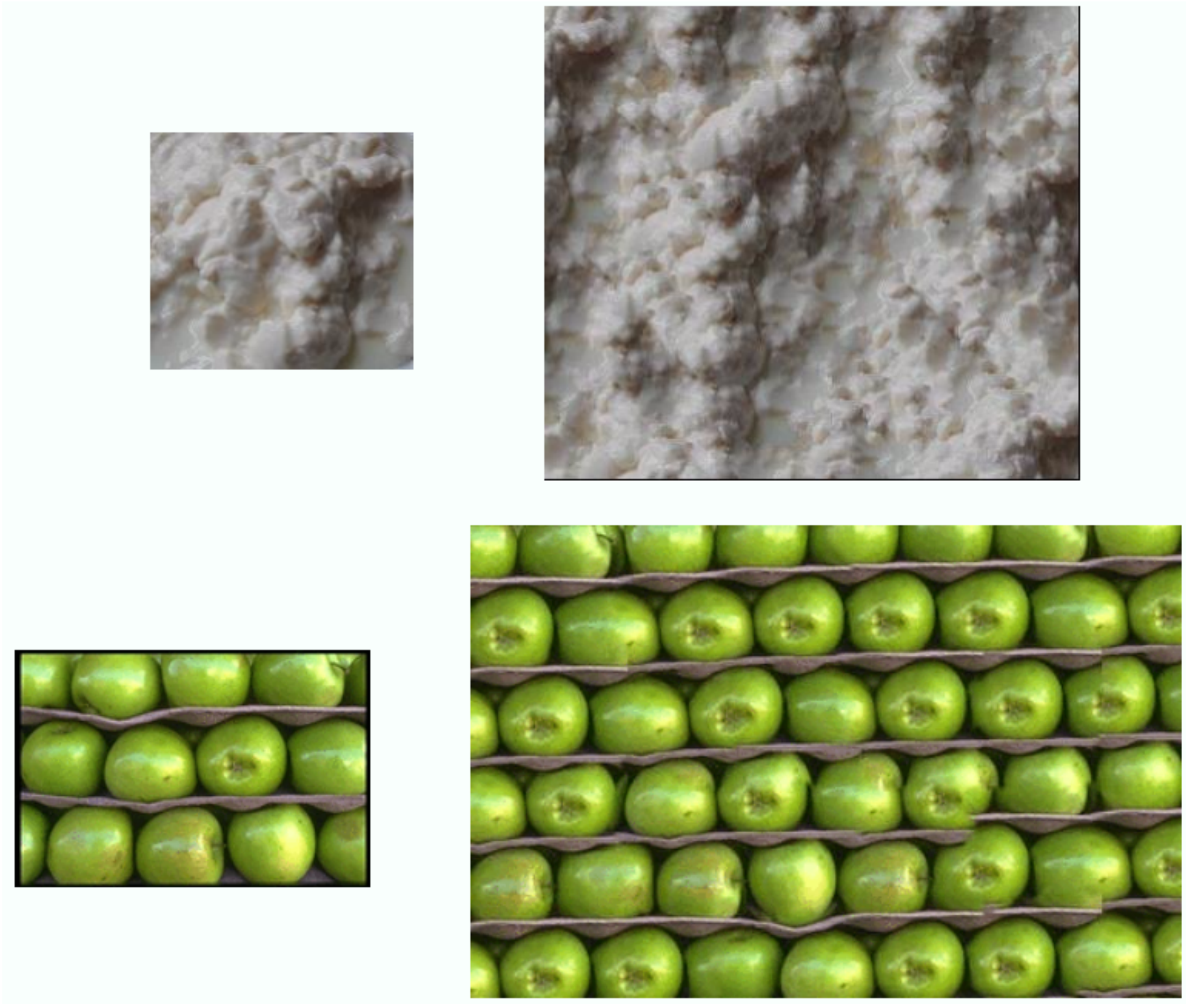


[Efros & Freeman SIGGRAPH01]

More results

[Efros & Freeman SIGGRAPH01]

More results

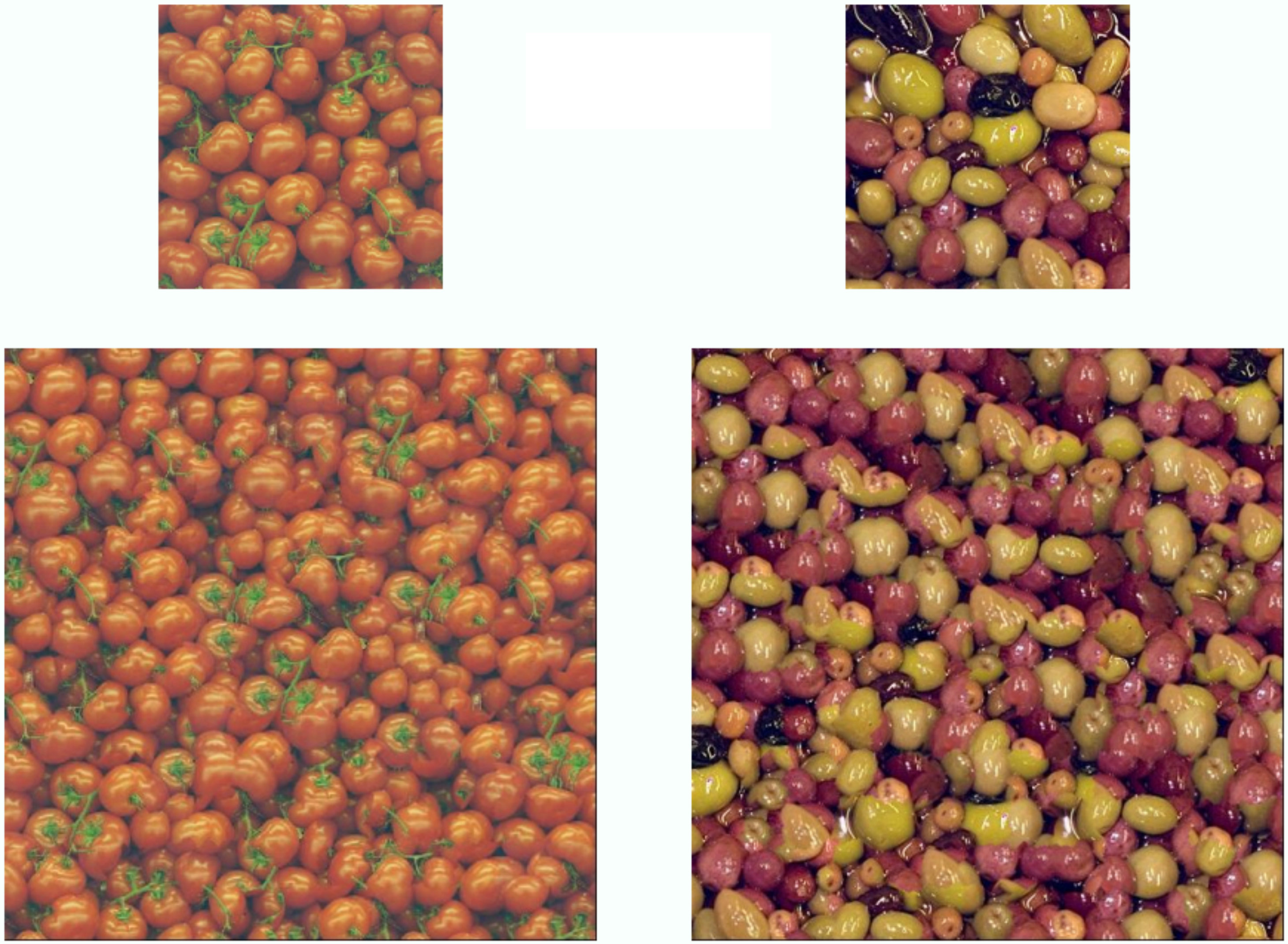


[Efros & Freeman SIGGRAPH01]

More results

[Efros & Freeman SIGGRAPH01]

Failure cases

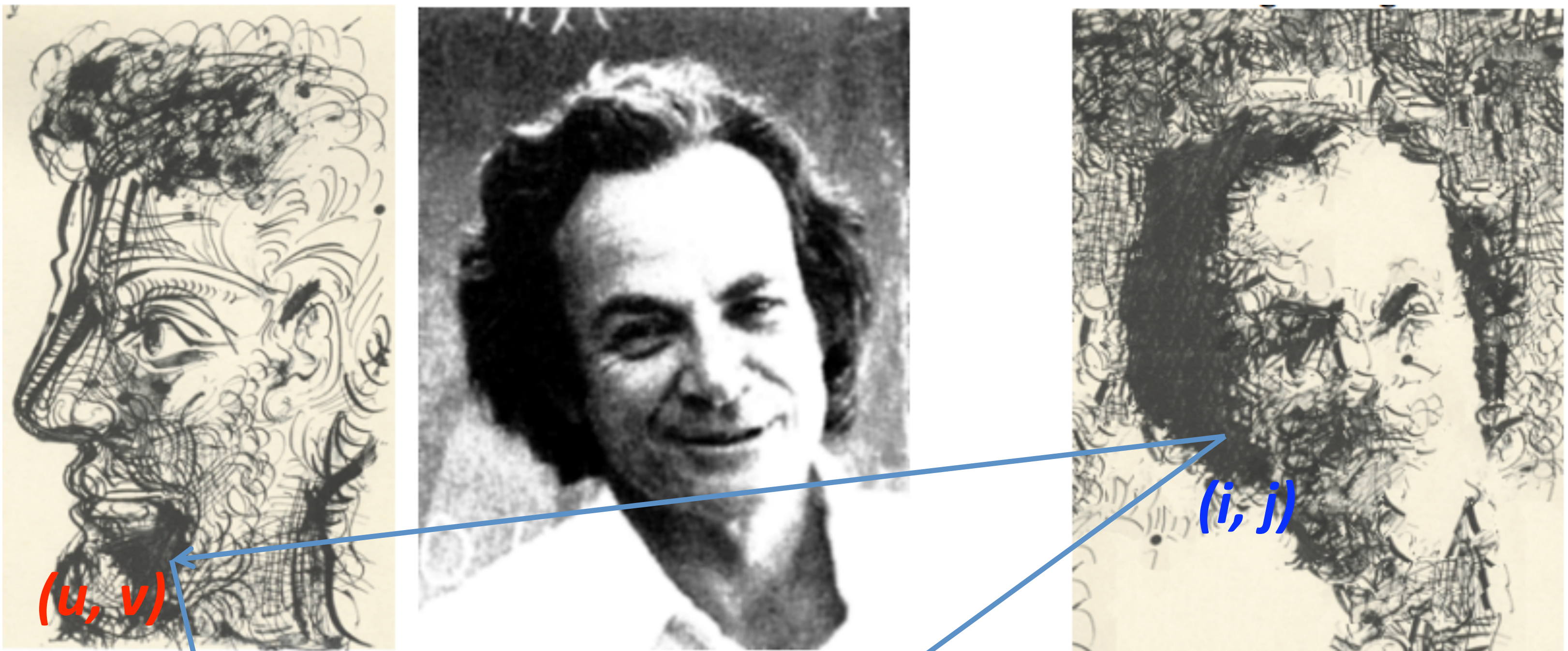


[Efros & Freeman SIGGRAPH01]

Texture Transfer

[Efros & Freeman SIGGRAPH01]

Texture Transfer



source texture | target image | texture transfer result

correspondence maps

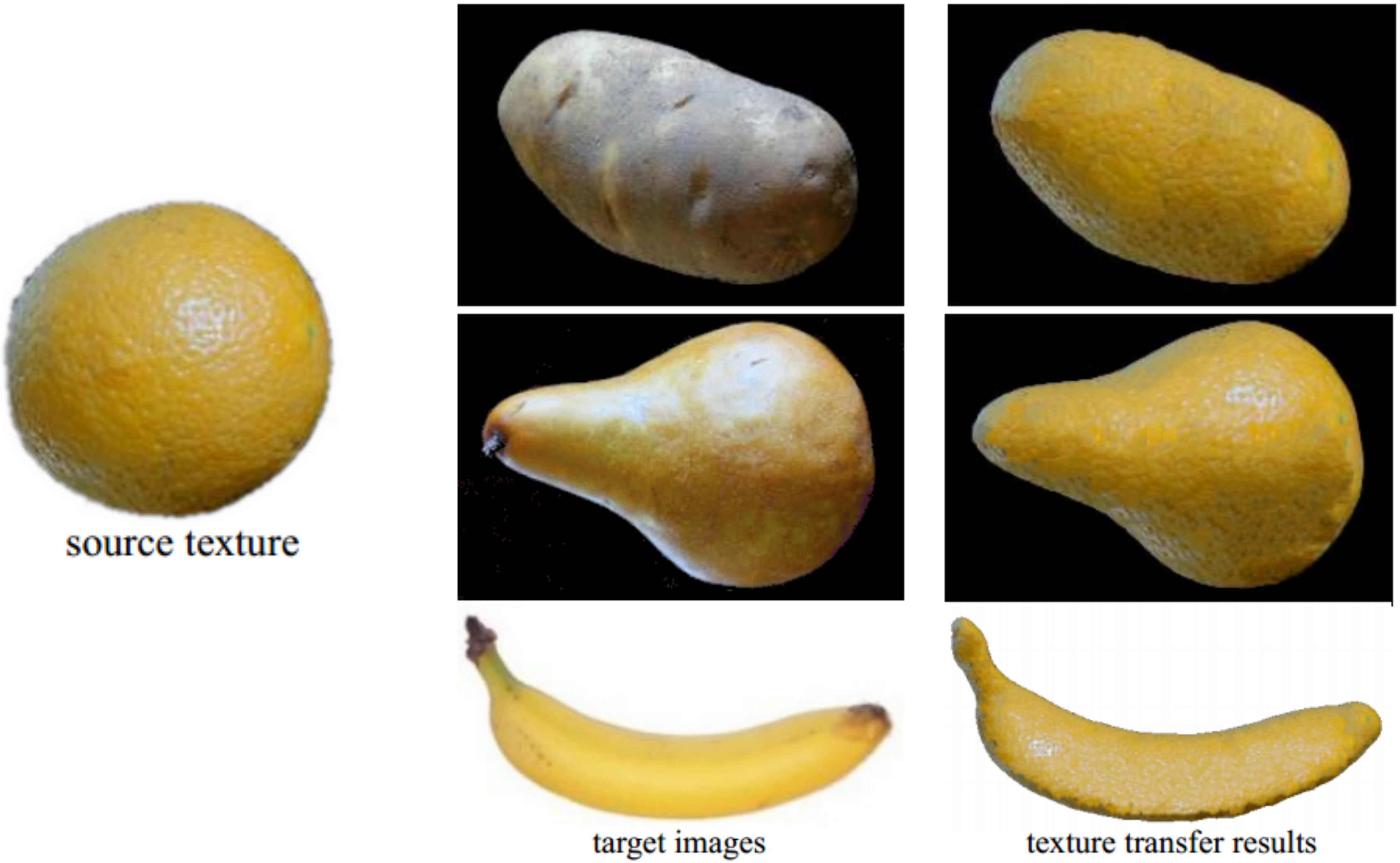
$$\boldsymbol{e_{i,j}} = \boldsymbol{\alpha}\left(\boldsymbol{B1_{ij}} - \boldsymbol{B2_{ij}}\right)^2 + (\mathbf{1} - \boldsymbol{\alpha})\left(\boldsymbol{Sc_{uv}} - \boldsymbol{Tc_{i,j}}\right)^2$$

(u,v): the coordinate of patch $B2_{i,j}$ *in the source texture*

Sc: source correspondence map

Tc: target correspondence map

More results

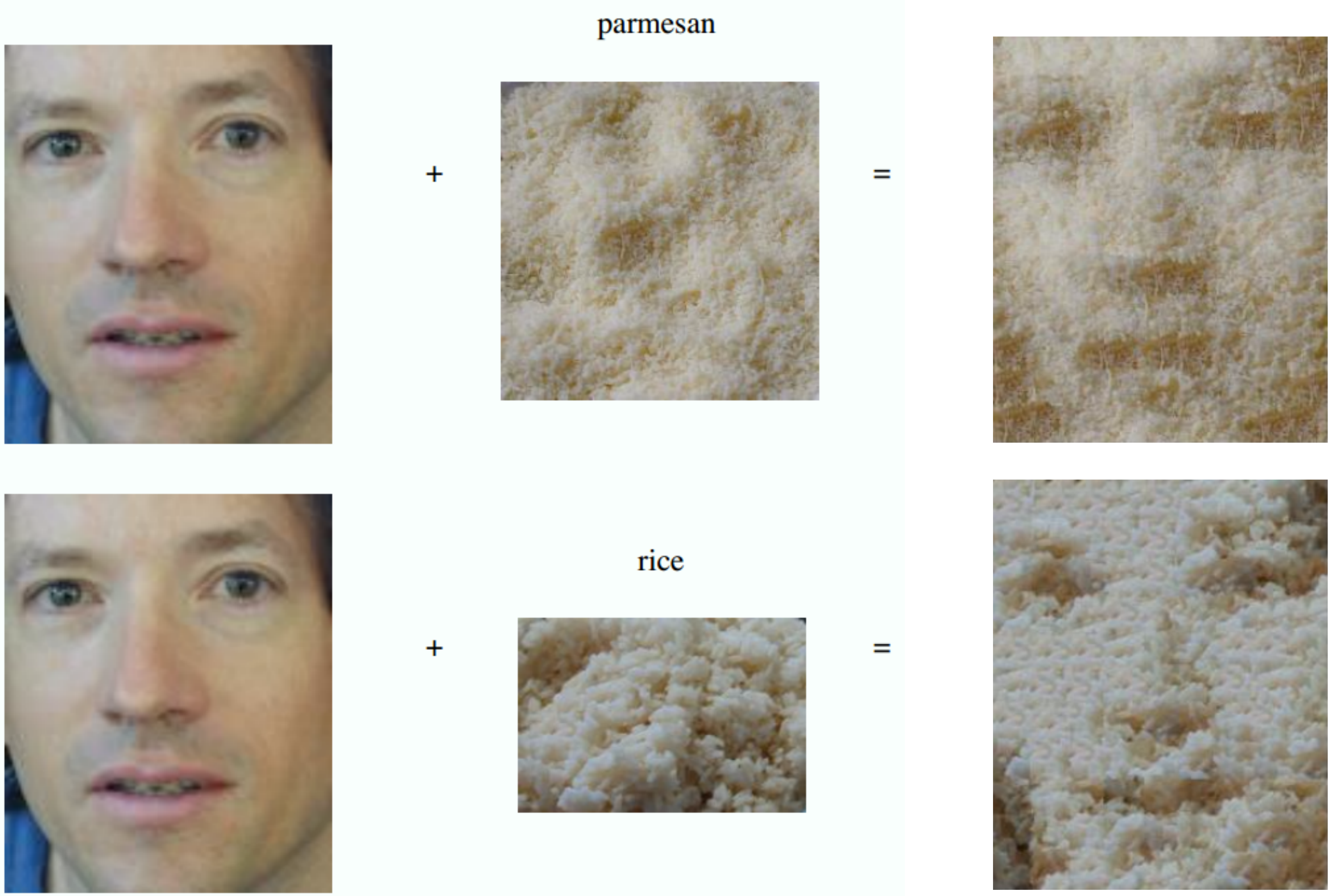


[Efros & Freeman SIGGRAPH01]

More results

[Efros & Freeman SIGGRAPH01]

More results



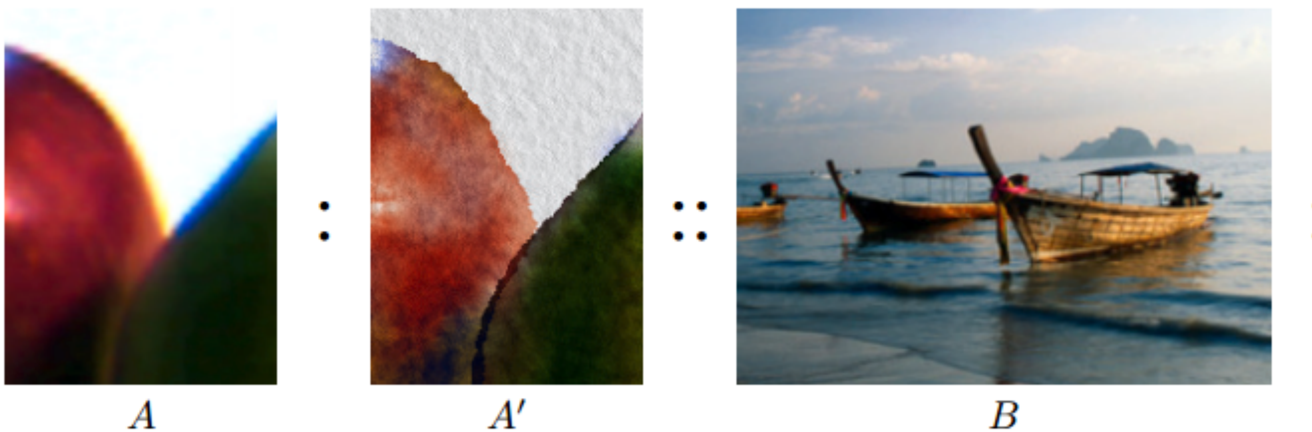
[Efros & Freeman SIGGRAPH01]

Pros and Cons

- Very simple
- Easy to implement
- Work well
- Fast!

- **Memoryless**
  - Cannot keep track of old solutions: A general problem for DP
- Better algorithm
  - Graph cut *[Graphcut texture, Kwatra SIGGRAPH03]*

Image Analogy



[Hertzman et al. SIGGRAPH01]

Image Analogy

[Hertzman et al. SIGGRAPH01]

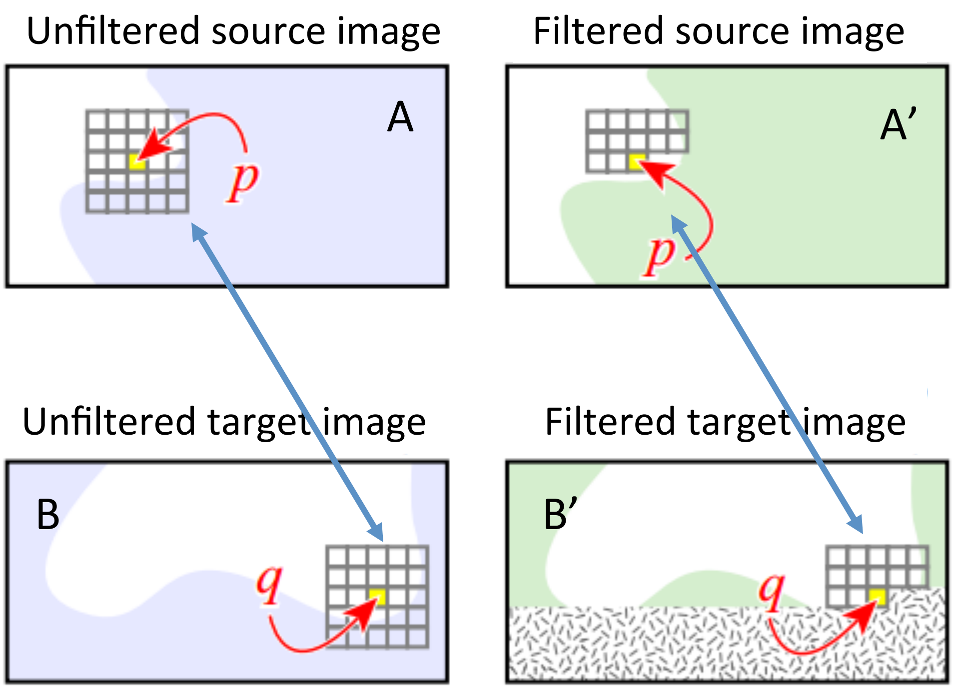
Image Analogy

Problem("IMAGE ANALOGY"): Given a pair of images A (*unfiltered source*) and filtered image A' (*filtered source*), Along with some additional unfiltered *target image* B, Synthesize a new *filtered target image* B' such that:

A : A' :: B : **B'**

[Hertzman et al. SIGGRAPH01]

Basic idea



(a simplified version)

For q in B’, find index p in source images such that
A’(p) ~ B’(q) and
A(p) ~ B(q)

Concatenate the windows of ***p*** in source pair, and that of ***q*** in the target pair, then match

[Hertzman et al. SIGGRAPH01]

Applications

- Learning a complicated filter from data
- Super-resolution
- Exemplar-based NPR
- Texture synthesis!

[Hertzman et al. SIGGRAPH01]

Training

Unfiltered source (*A*)

:

Filtered source (*A′*)

[Hertzman et al. SIGGRAPH01]

Unfiltered target (B)

Learned filtered target (B')

[Hertzman et al. SIGGRAPH01]

Unfiltered target (B)

Learned filtered target (B')

[Hertzman et al. SIGGRAPH01]

Learn to Blur

Unfiltered source (A)

Filtered source (A')

Unfiltered target (B)

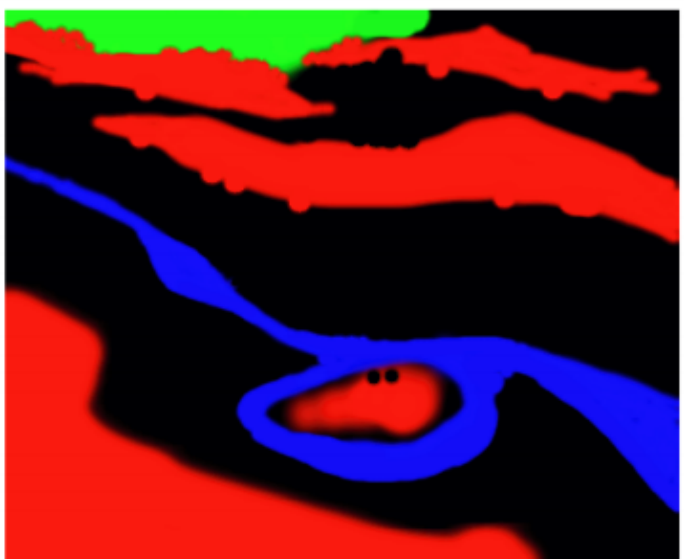
Filtered target (B')

[Hertzman et al. SIGGRAPH01]

Texture by Numbers

Unfiltered source (A)

Filtered source (A')

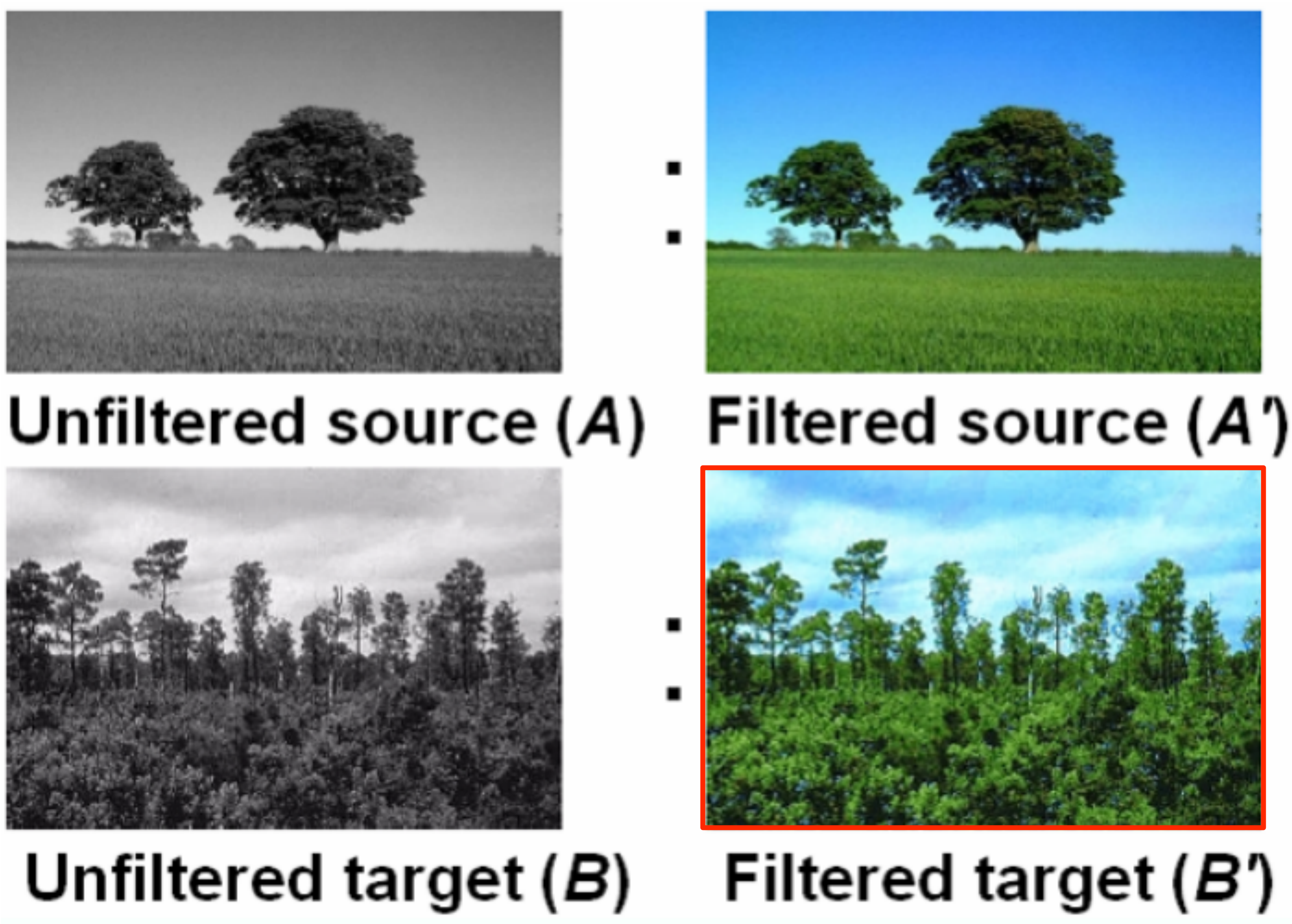


Unfiltered (B)

Filtered (B')

[Hertzman et al. SIGGRAPH01]

Colorization



[Hertzman et al. SIGGRAPH01]

Super-resolution

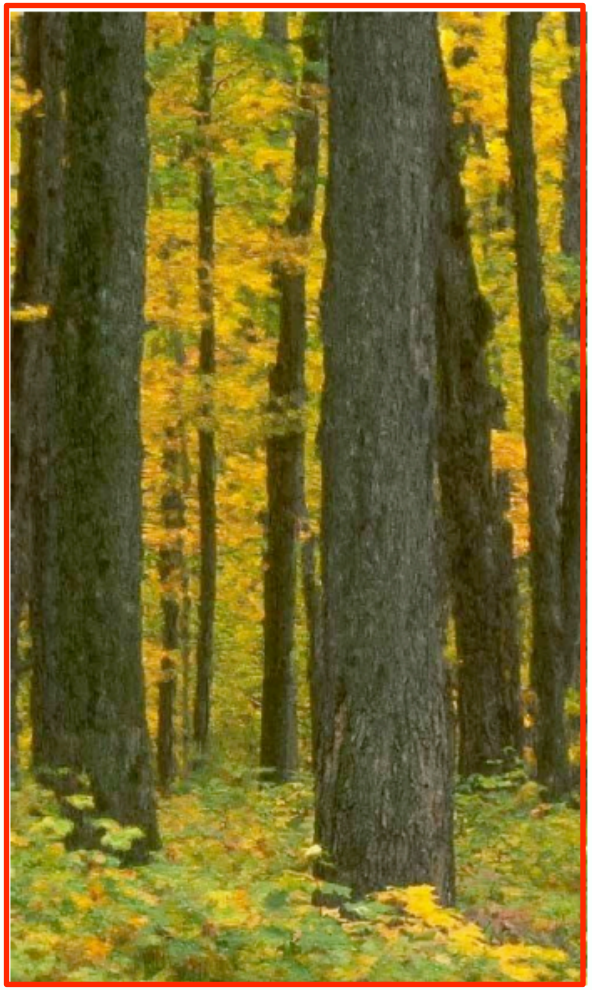
Unfiltered source

Filtered source

[Hertzman et al. SIGGRAPH01]

Super-resolution (result)

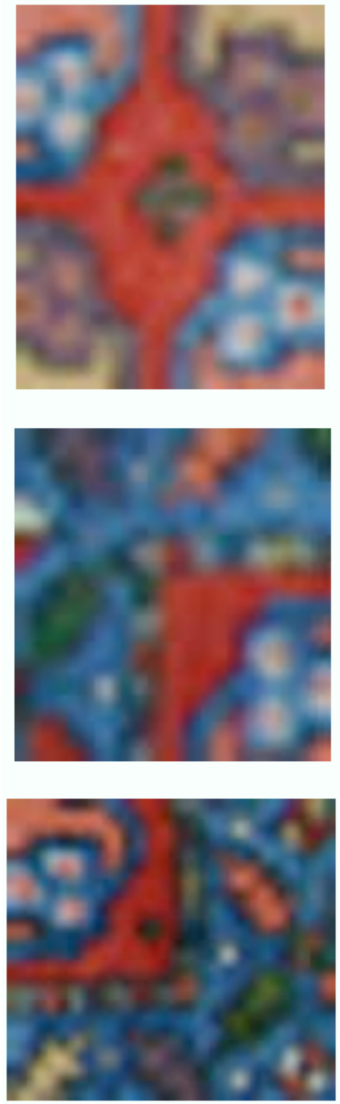
Unfiltered target



Filtered target

[Hertzman et al. SIGGRAPH01]

Super-resolution



Unfiltered source

Filtered source

[Hertzman et al. SIGGRAPH01]

Super-resolution (result)

[Hertzman et al. SIGGRAPH01]

Wrap-up: The two steps of texture synthesis

- Modeling ➔Visual fidelity
  - How to estimate the stochastic process from a given finite texture sample
  - Both used MRF model (locality and stationary)

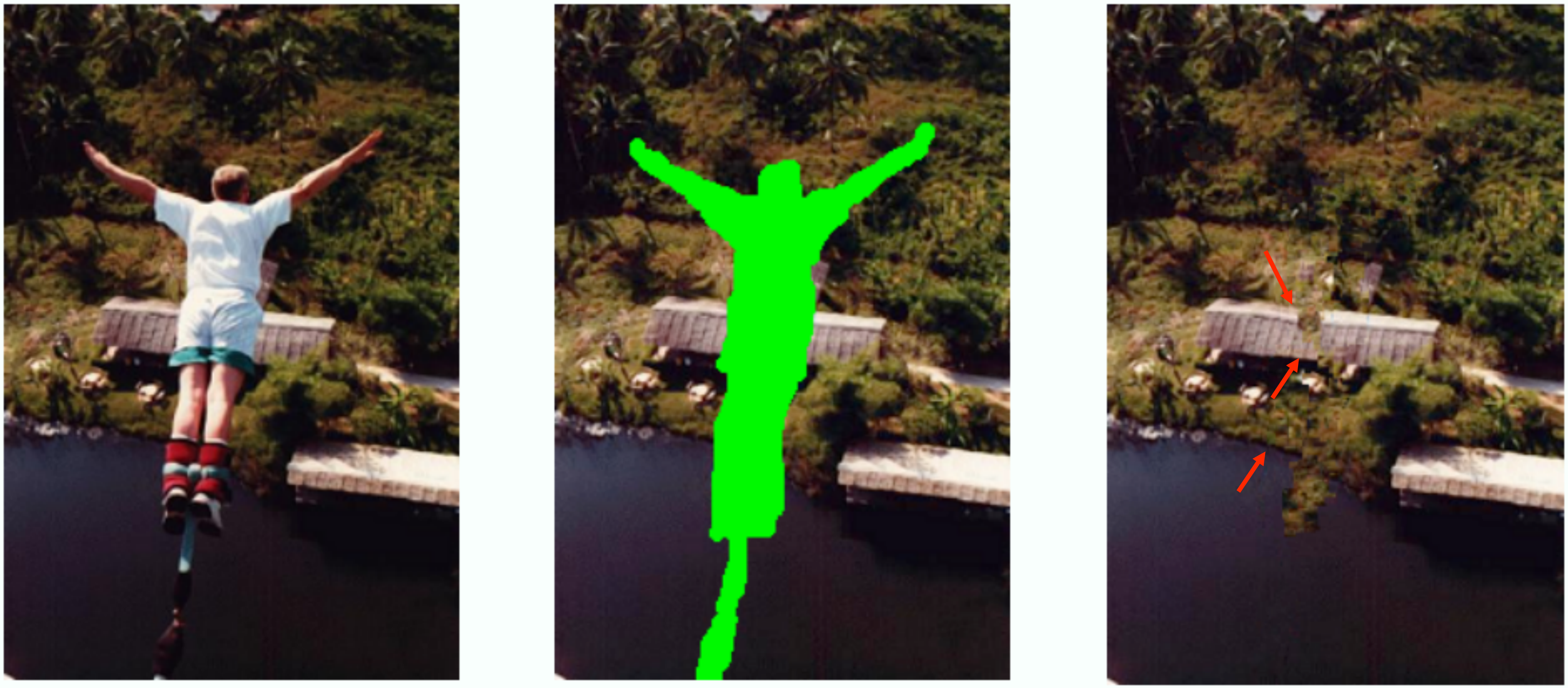
- Sampling ➔Computational cost
  - How to develop an efficient sampling procedure based on the model
  - Pixel by pixel VS patch-based

  - Sampling under guidance: texture transfer, image analogy

Two problems remains

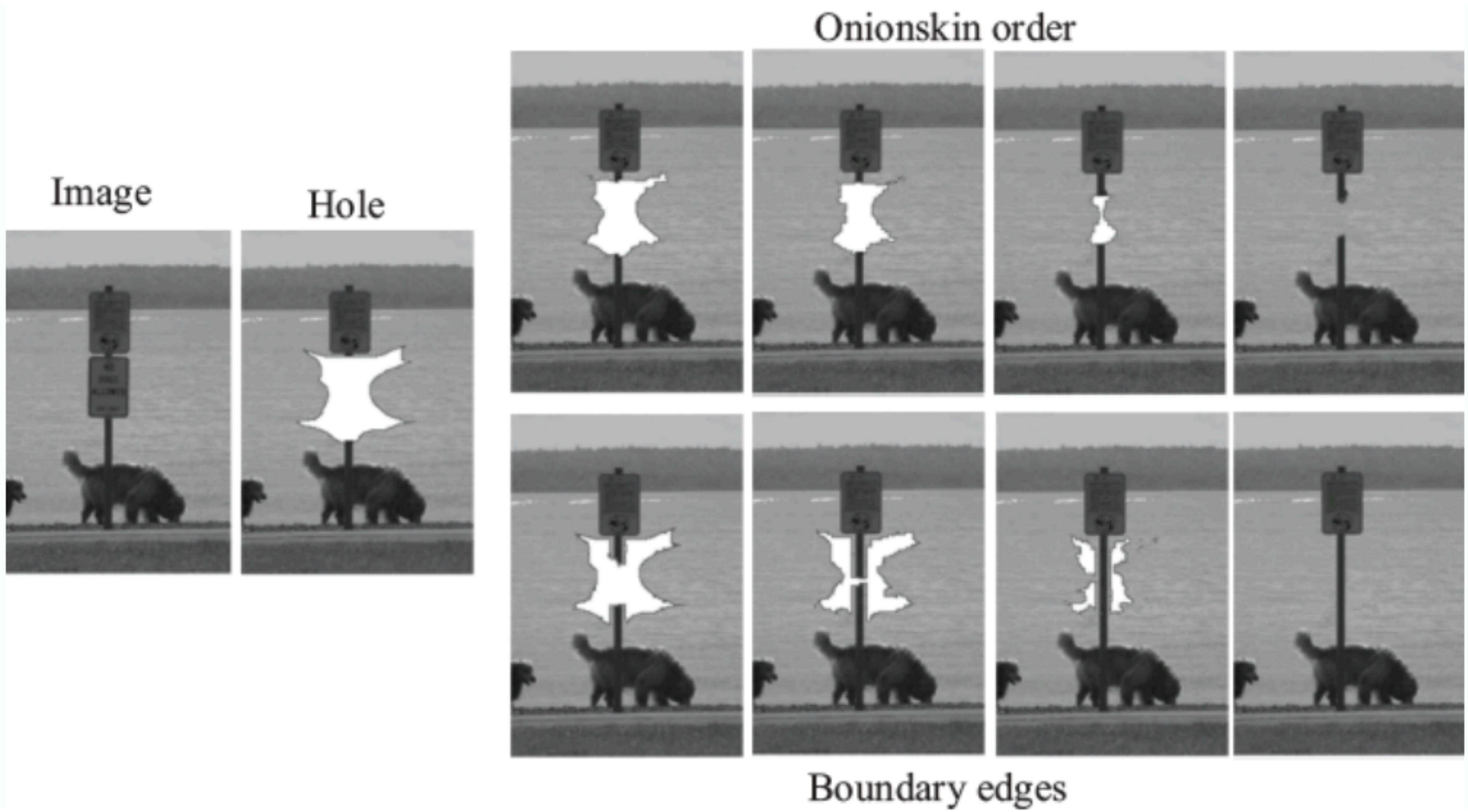
- Preserving scene structure
  - Prioritize synthesize order *[Criminisi et al. CVPR03]*
- Efficient patch matching
  - Fast approximate NN search *[Barnes et al. SIGGRAPH09]*

Inpainting



[Criminisi et al. CVPR03]

Synthesize Order Matters



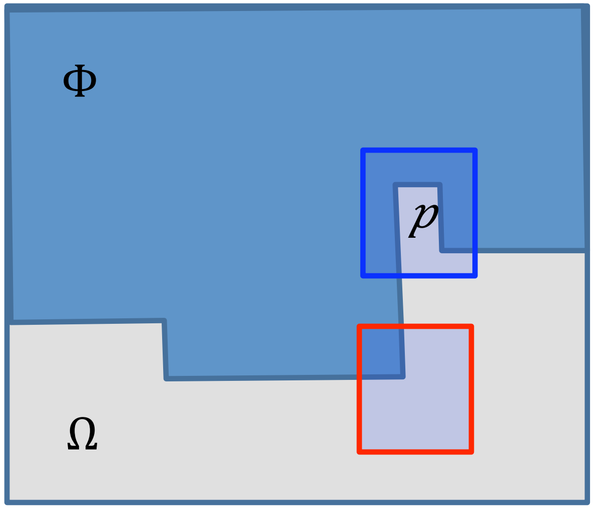
[Criminisi et al. CVPR03]

Choose the order

- Confidence
  - Favor unknown pixels with more (reliable) neighbor information

$$C(p) = \frac{\sum_{q \in \Psi_p \cap \Phi} C(q)}{|\Psi_p|}$$

Ψ_p: Neighborhood of *p*

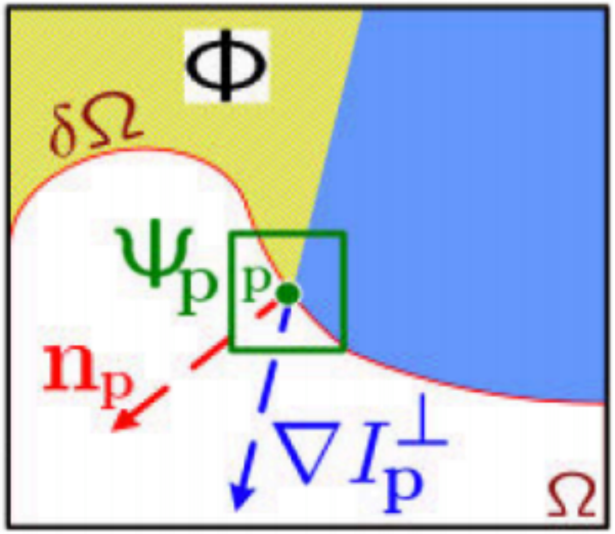


[Criminisi et al. CVPR03]

Choose the order

- Data term
  - Favor starting from strong edges (indication of high saliency for structures)

$$D(p) = \frac{|\nabla I_p^{\perp} \cdot n_p|}{\alpha}$$



[Criminisi et al. CVPR03]

Choose the order

- Priority
  - Balance between confidence and data term

$$P(p) = C(p)\ D(p)$$

- Algorithm

- Extract the manually selected initial front $\delta\Omega^0$.
- Repeat until done:

  1a. Identify the fill front $\delta\Omega^t$. If $\Omega^t = \emptyset$, exit.

  1b. Compute priorities $P(\mathbf{p}) \quad \forall \mathbf{p} \in \delta\Omega^t$.

  2a. Find the patch $\Psi_{\hat{\mathbf{p}}}$ with the maximum priority, *i.e.*, $\Psi_{\hat{\mathbf{p}}} \mid \hat{\mathbf{p}} = \arg\max_{\mathbf{p} \in \delta\Omega^t} P(\mathbf{p})$

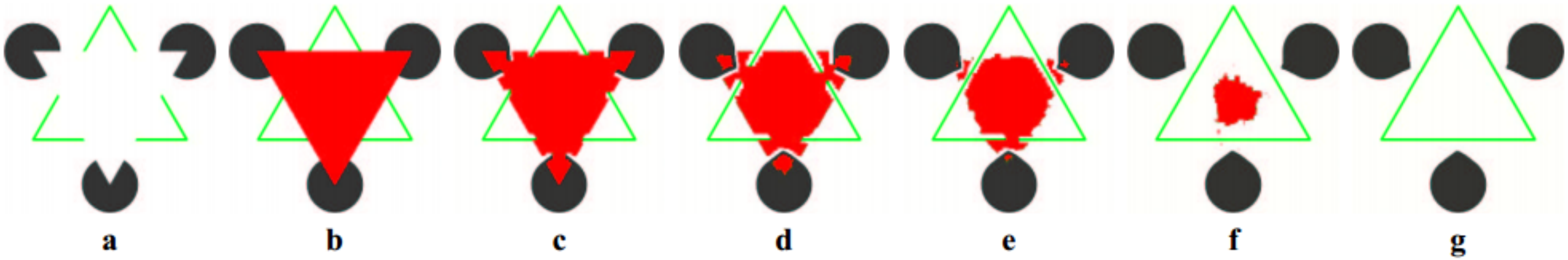
  2b. Find the exemplar $\Psi_{\hat{\mathbf{q}}} \in \Phi$ that minimizes $d(\Psi_{\hat{\mathbf{p}}}, \Psi_{\hat{\mathbf{q}}})$.

  2c. Copy image data from $\Psi_{\hat{\mathbf{q}}}$ to $\Psi_{\hat{\mathbf{p}}}$.

  3. Update $C(\mathbf{p}) \quad \forall \mathbf{p} \ |\mathbf{p} \in \Psi_{\hat{\mathbf{p}}} \cap \Omega$

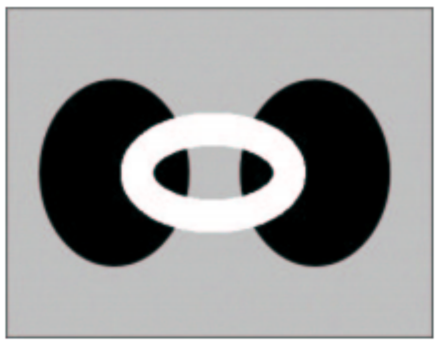
[Criminisi et al. CVPR03]

Results

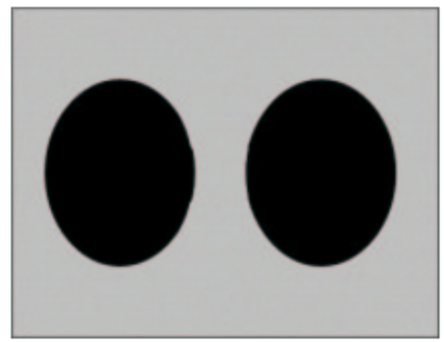


[Criminisi et al. CVPR03]

More results



Input



Result

[Criminisi et al. CVPR03]

More results

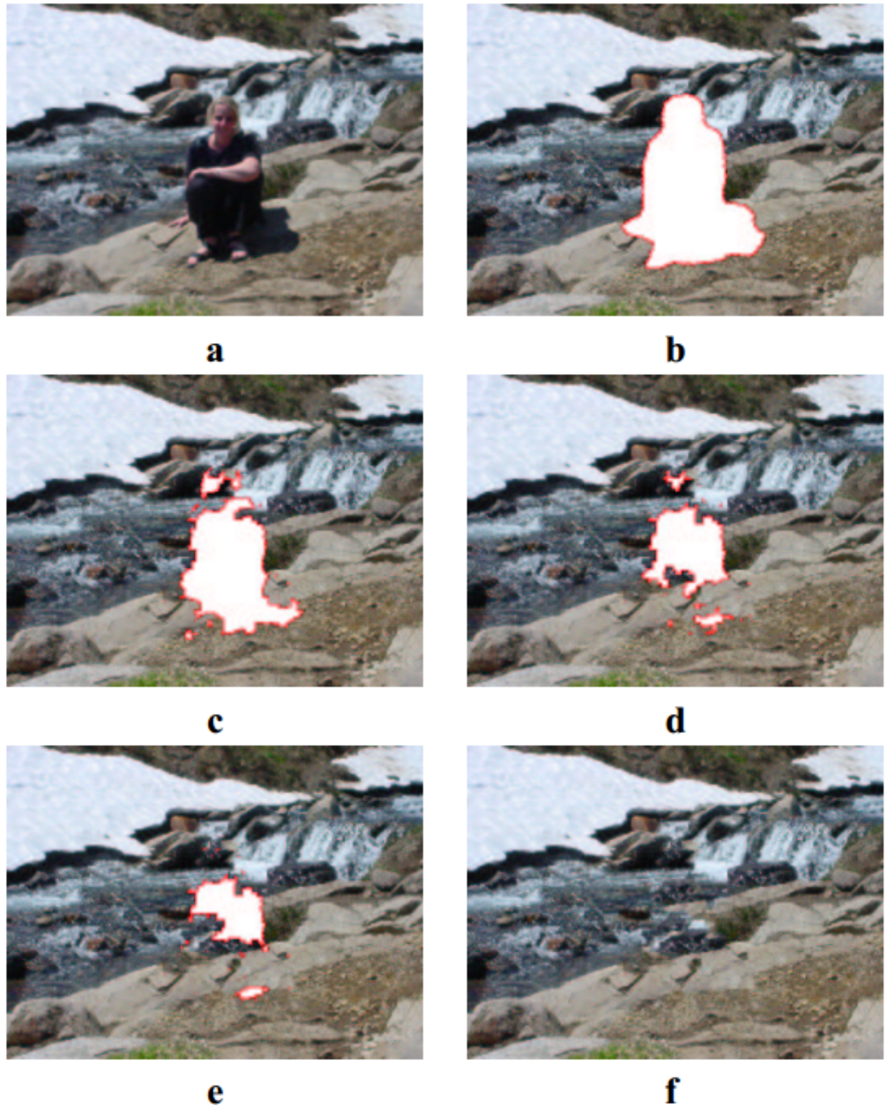
Input

Masked target region

Result

[Criminisi et al. CVPR03]

More results



[Criminisi et al. CVPR03]

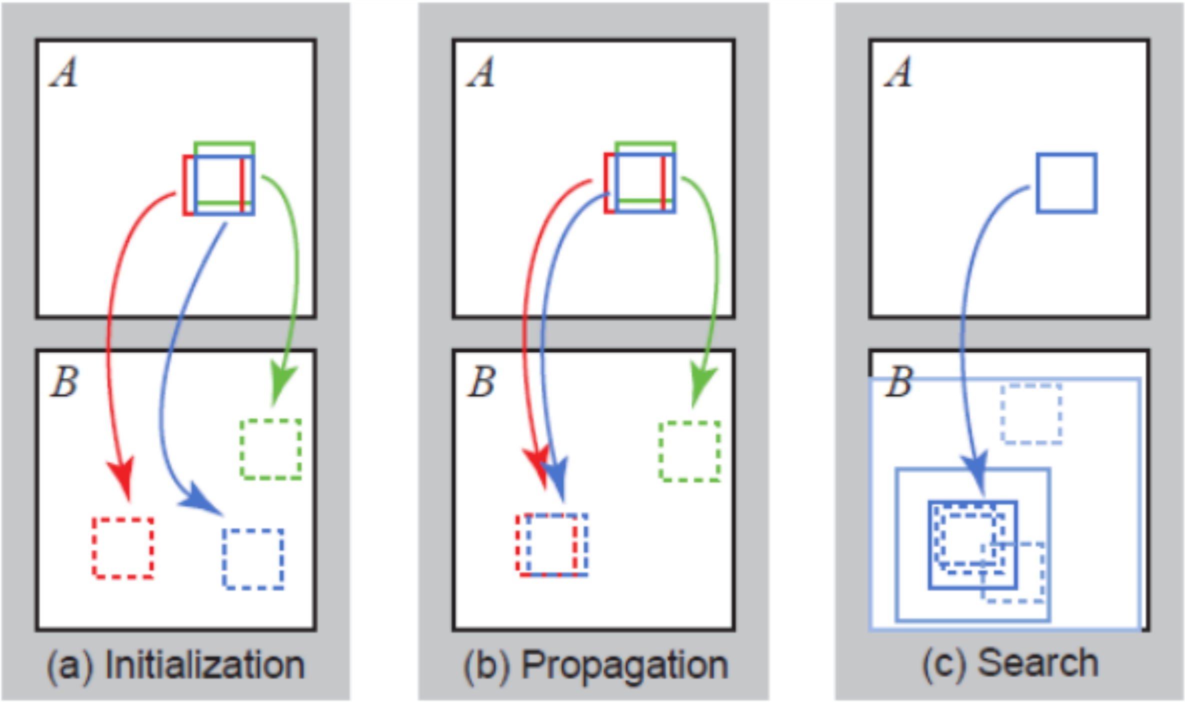
Fast Approximate NN search

- Brute force search
  - Sliding window
  - Exact answer , but too slow

- Better idea
  - **Local coherence**: nearby windows have high probability to match nearby windows in the source image

[Barnes et al. SIGGRAPH09]

Fast ANN search



(a) Random initialization,
(b) Propagate good matches → **20-100x speedup**
(c) Search nearby

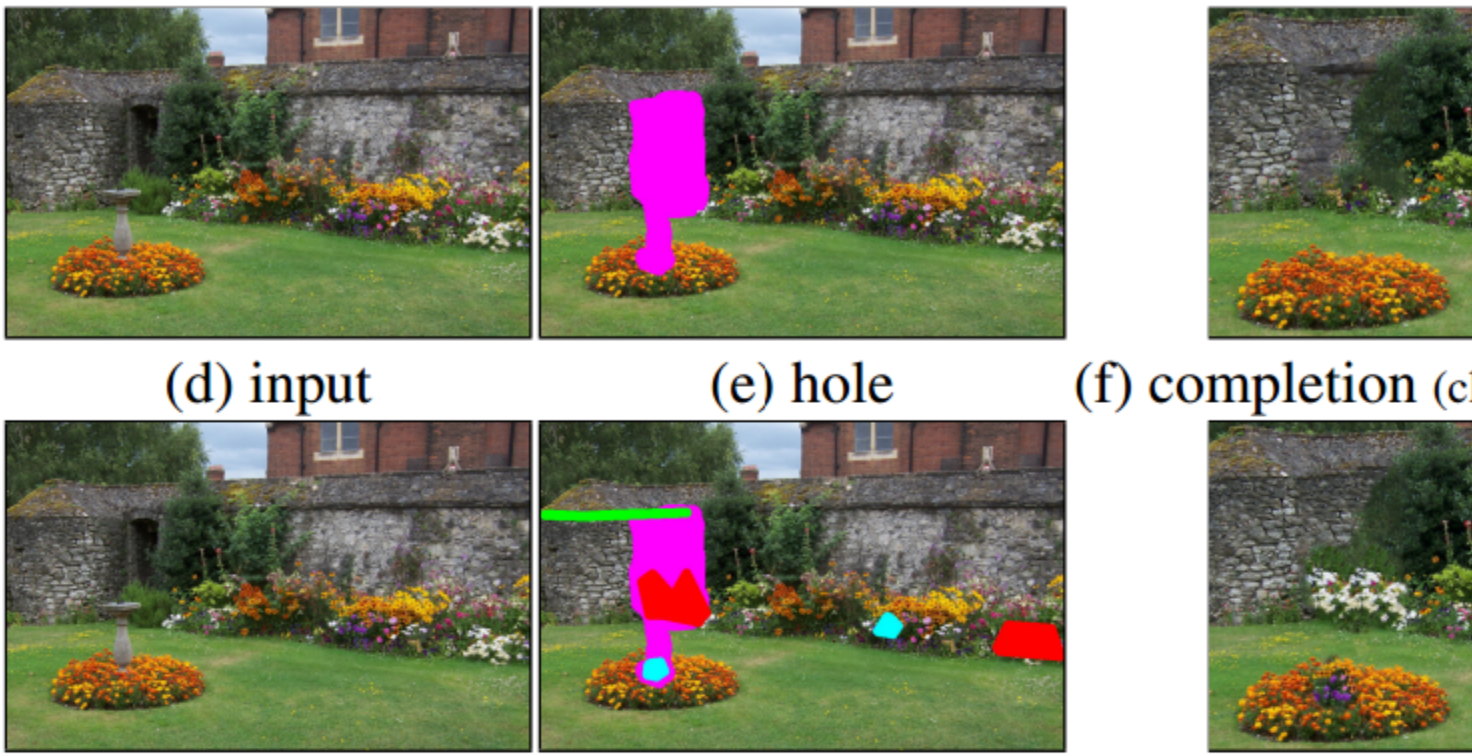
[Barnes et al. SIGGRAPH09]

Add User Constraints

- Mark an hole to fill in (standard process)
- Add extra label, partly inside the hole, partly outside
- Limit the search space for labeled pixels inside the hole to outside regions with the same label

[Barnes et al. SIGGRAPH09]

Add User Constraints

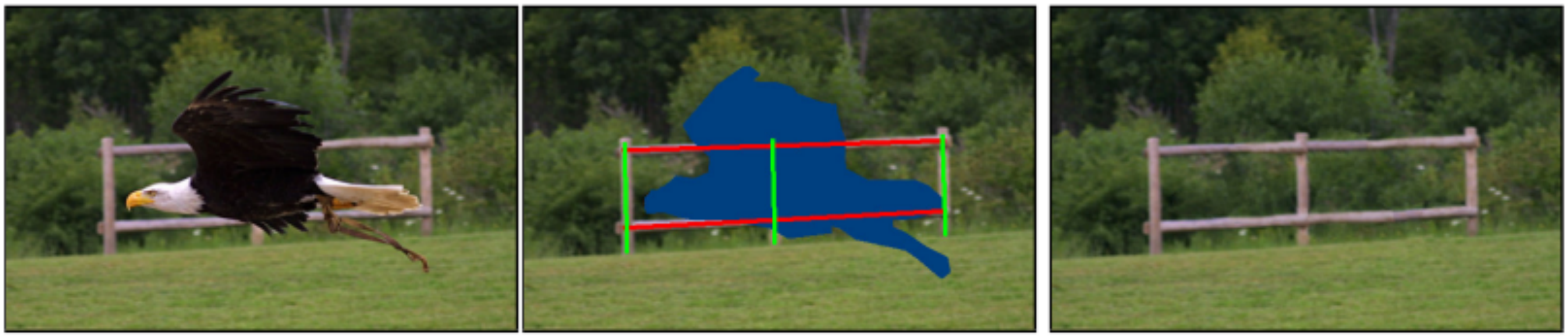


(d) input (e) hole (f) completion (close up)

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

[Barnes et al. SIGGRAPH09]

More results



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

[Barnes et al. SIGGRAPH09]

More results

(a) original (b) hole+constraints (c) hole filled

(d) constraints (e) constrained retarget (f) reshuffle

[Barnes et al. SIGGRAPH09]

Retargeting

- Make an image bigger or smaller in one direction
  - e.g. Change aspect ratio
- Traditional
  - Interpolation: content distortion
  - Cut off pixels: lose important content

- Seam Carving
  - Cut out a curve of pixels that “does not matter much”

Example

input

seam

scaling

cropping

[Avidan&Shamir. SIGGRAPH07]

Find a seam

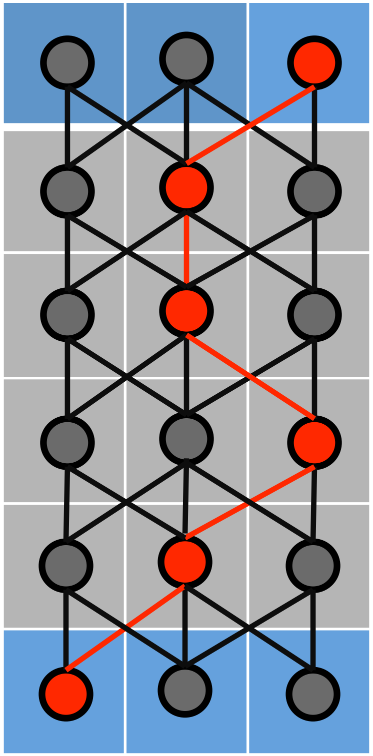
- Define an optimal seam
  - Vertical seam, horizontal seam
  - Energy
    - **Gradient**
    - Entropy, HOG, Segmentation, Saliency, Corner detector, eye gaze

$$e_1(\mathbf{I}) = \left|\frac{\partial}{\partial x}\mathbf{I}\right| + \left|\frac{\partial}{\partial y}\mathbf{I}\right|$$

- Optimal seam: A seam with minimum energy → dynamic programming

[Avidan&Shamir. SIGGRAPH07]

Find a seam



Initialize: $e_{i,j} = \left|\frac{\partial}{\partial x} I_{ij}\right| + \left|\frac{\partial}{\partial y} I_{ij}\right|$

for $i = 2 : h$; for $j = 1 : w$

$E_{i,j} = e_{ij} + min\,(E_{i-1,j-1}, E_{i-1,j}, E_{i-1,j+1})$

$k_{i,j} = argmin\,(E_{i-1,j-1}, E_{i-1,j}, E_{i-1,j+1})$

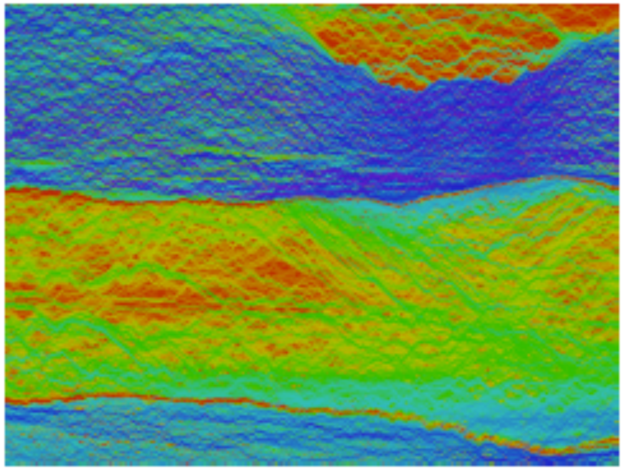
end; end

Exactly the same DP, different energy term
(Compare with the algorithm of image quilting (P33)!)

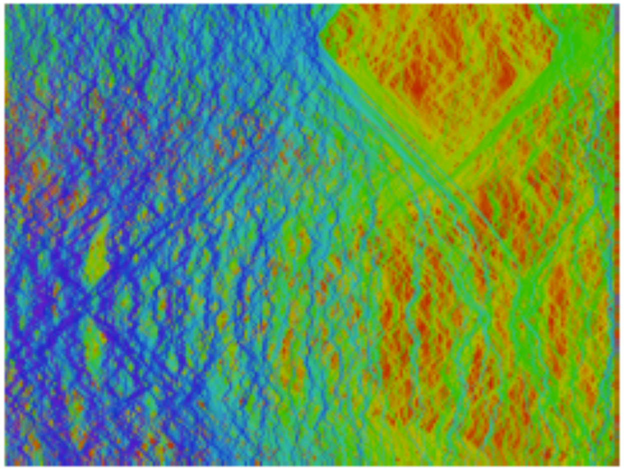
[Avidan&Shamir. SIGGRAPH07]

Visualization of seams

Image



Horizontal seams



Vertical seams

Blue: low cost seams
Red: high cost seams

Results

(a) Original (b) Crop (c) Column (d) Seam

[Avidan&Shamir. SIGGRAPH07]

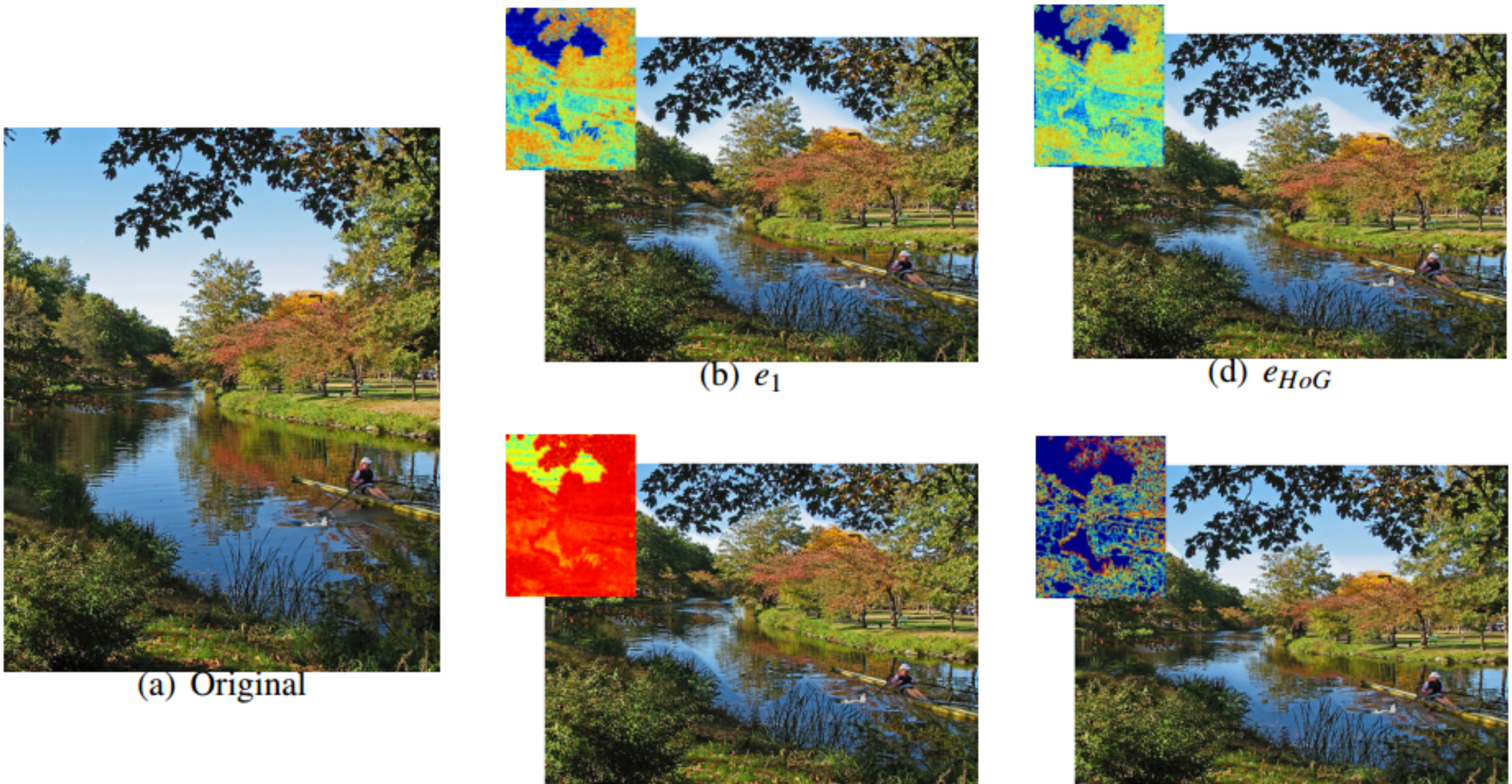
More results

input

output

[Avidan&Shamir. SIGGRAPH07]

Difference Energies make a difference



(a) Original

(b) e_1

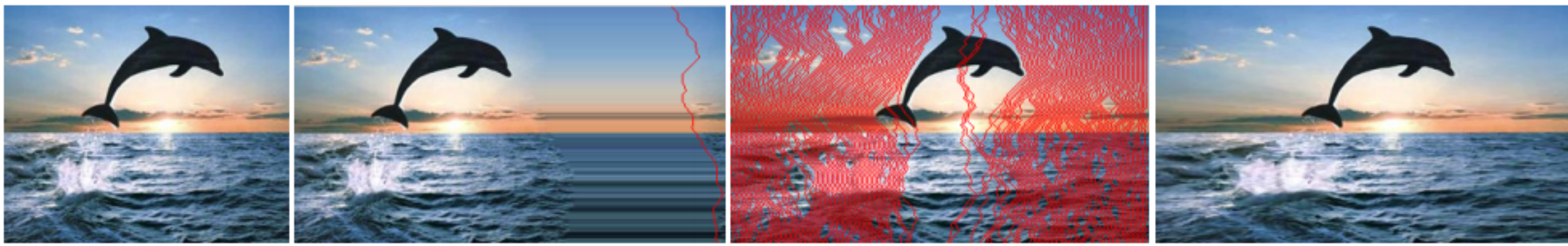
(d) e_{HoG}

(c) $e_{Entropy}$

(e) Segmentation and L_1

[Avidan&Shamir. SIGGRAPH07]

Seam insertion



(a) (b) (c) (d)

- **Finding and inserting the optimum seam will most likely insert the same seam again and again, resulting in (b)**
- **Instead, inserting the seams in order of removal (c), achieves better result (d)**

[Avidan&Shamir. SIGGRAPH07]

More results

input

widening by seam insertion

[Avidan&Shamir. SIGGRAPH07]

Seam carving and insertion

Input

Seam Carving result

Rescaling by interpolation

[Avidan&Shamir. SIGGRAPH07]

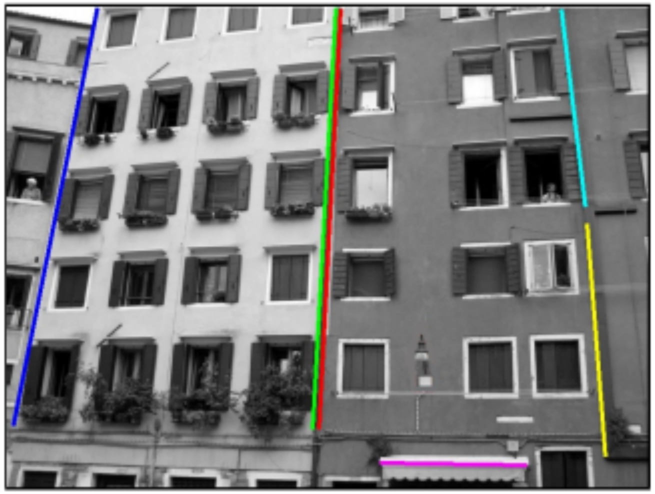
Failure case: need extra constraints

Figure 14: Retargeting the left image with e_1 alone (middle), and with a face detector (right).

[Avidan&Shamir. SIGGRAPH07]

Constrained Retargeting

Main idea: user supplied constraints (mask), high cost for seam crossing the masked region



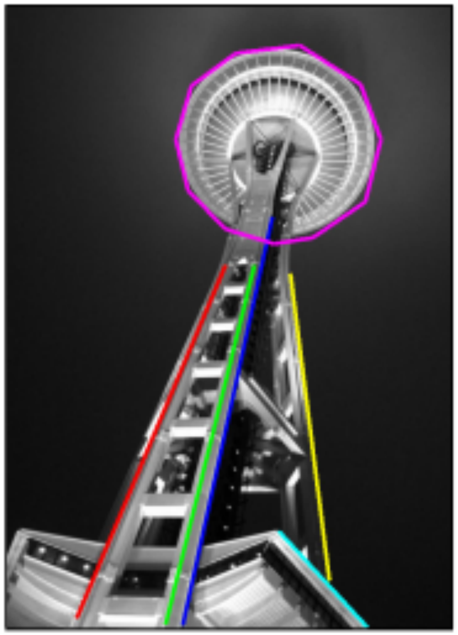
Input + constraints

Retarget
without constraints

Retarget
with constraints

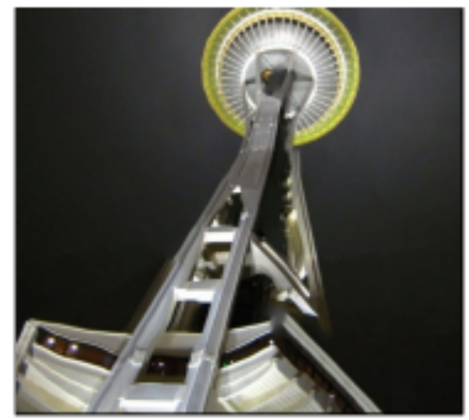
[Barnes et al. SIGGRAPH09]

More results



Input + constraints

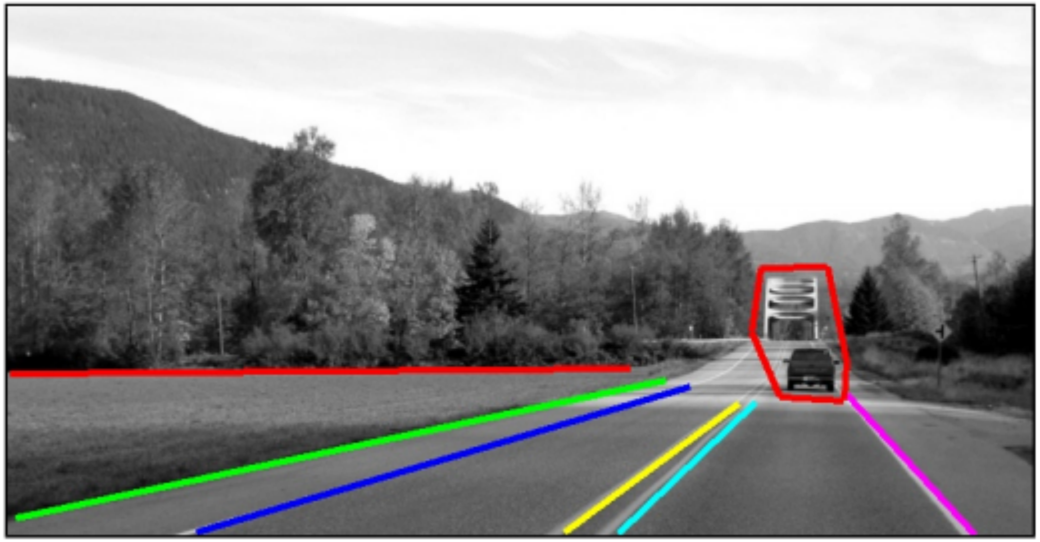
Retarget
without constraints



Retarget
with constraints

[Barnes et al. SIGGRAPH09]

More results



Input + constraints

Retarget
without constraints

Retarget
with constraints

[Barnes et al. SIGGRAPH09]

More results

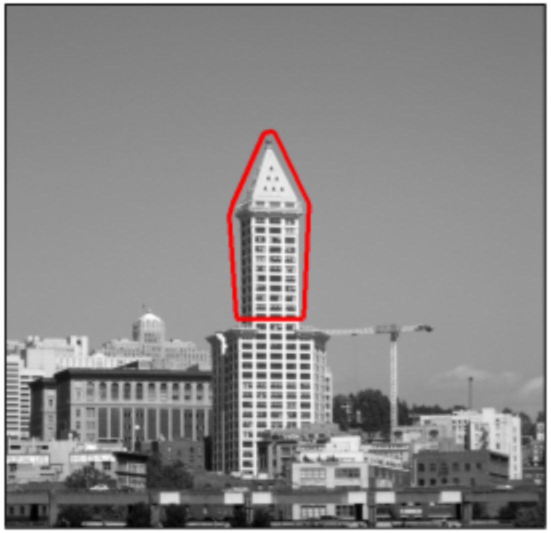
Input + constraints

Retarget
without constraints

Retarget
with constraints

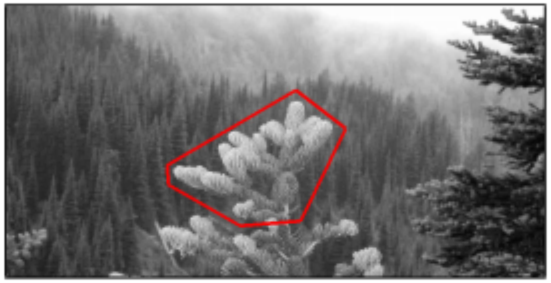
[Barnes et al. SIGGRAPH09]

Local scale editing



(a) building marked by user

(b) scaled up, preserving texture



(c) bush marked by user

(d) scaled up, preserving texture.

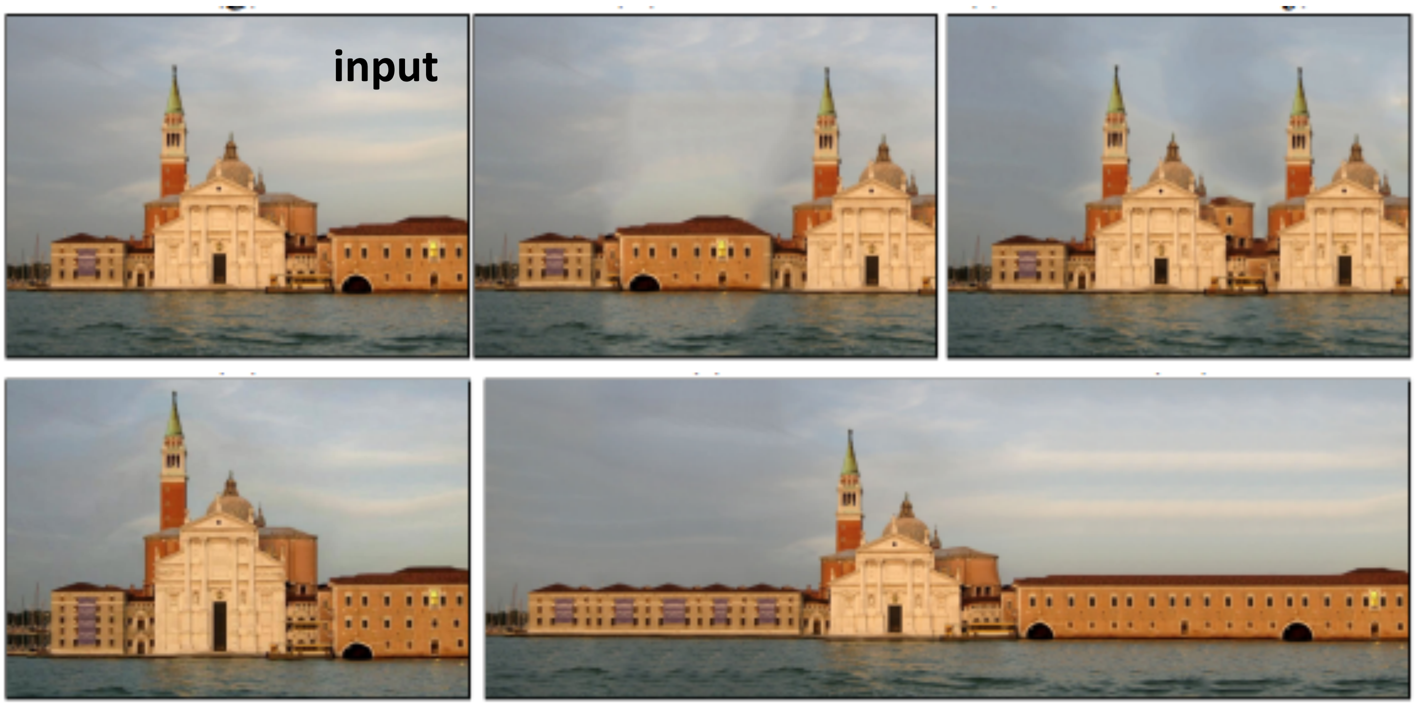
[Barnes et al. SIGGRAPH09]

Reshuffling

input Reshuffled

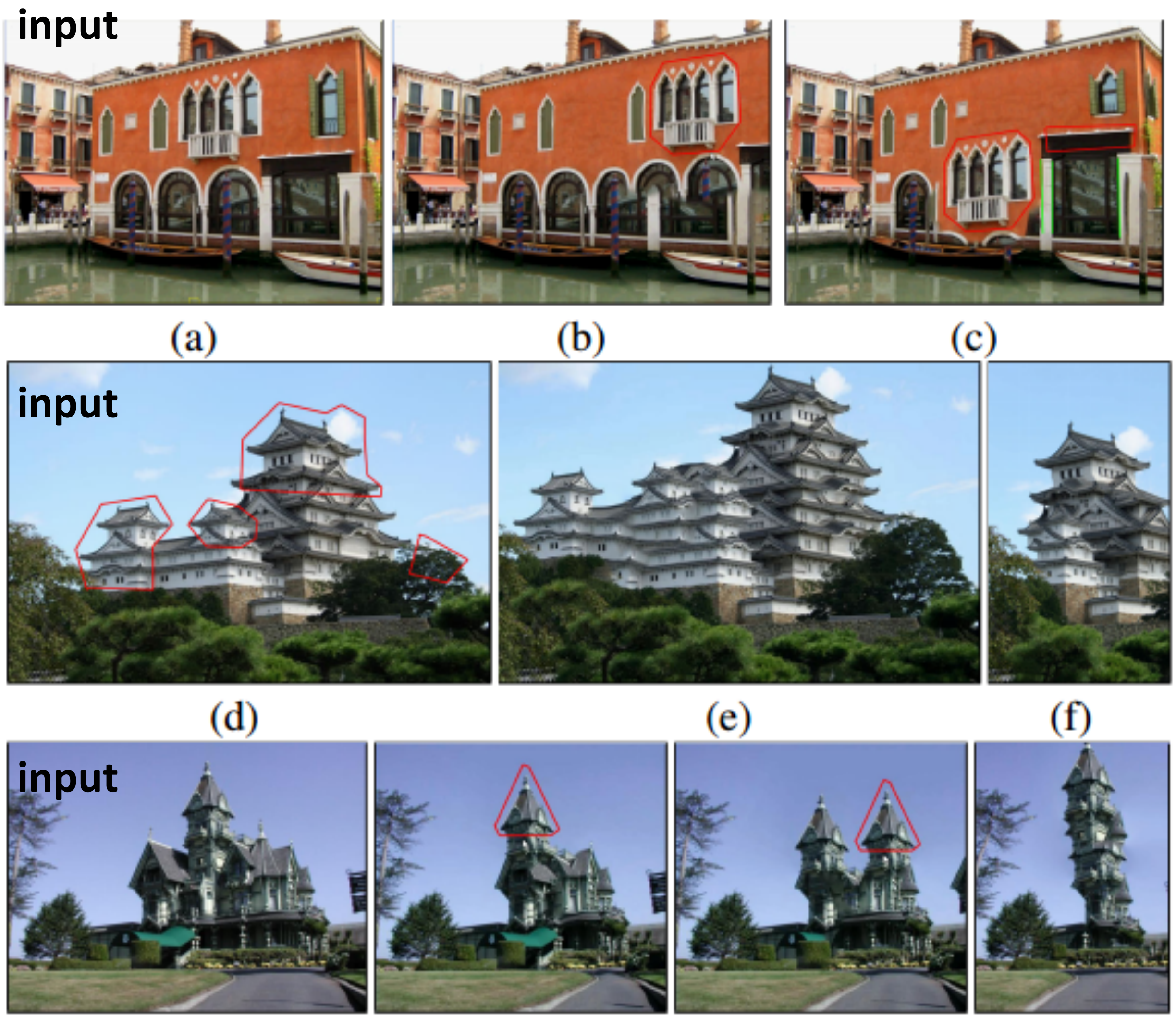
[Barnes et al. SIGGRAPH09]

More Reshuffling



[Barnes et al. SIGGRAPH09]

More Reshuffling



[Barnes et al. SIGGRAPH09]

http://gfx.cs.princeton.edu/pubs/Barnes_2009_PAR/index.php

Summary

- Texture
  - Stationary
  - Locality
- Scene
  - Texture
  - Structure (edges, object contours, etc)
- Two crucial algorithmic points
  - Nearest neighbor search
  - Dynamic programming

Summary (2)

- Patch representation
  - **RGB, luminance, CIELAB**…
  - **Gradient**, HOG, Harris corner detector, steerable filter
  - Saliency
  - Entropy
- Patch distance
  - L1 norm, L2 norm, sum of square
  - Gaussian weighted sum of square
- Patch match algorithm
  - Brute force NN
  - Approximate NN: local coherence
  - Tree structure: kd-tree (used in the image analogy method)
  - Dimension reduction: PCA, vector quantization
- Multi-scale representation
  - Gaussian pyramid (used in the image analogy method)