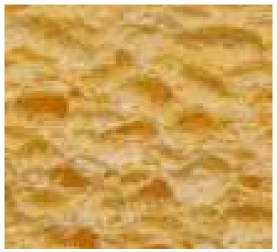
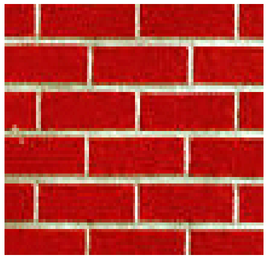
纹理合成和图像缩放

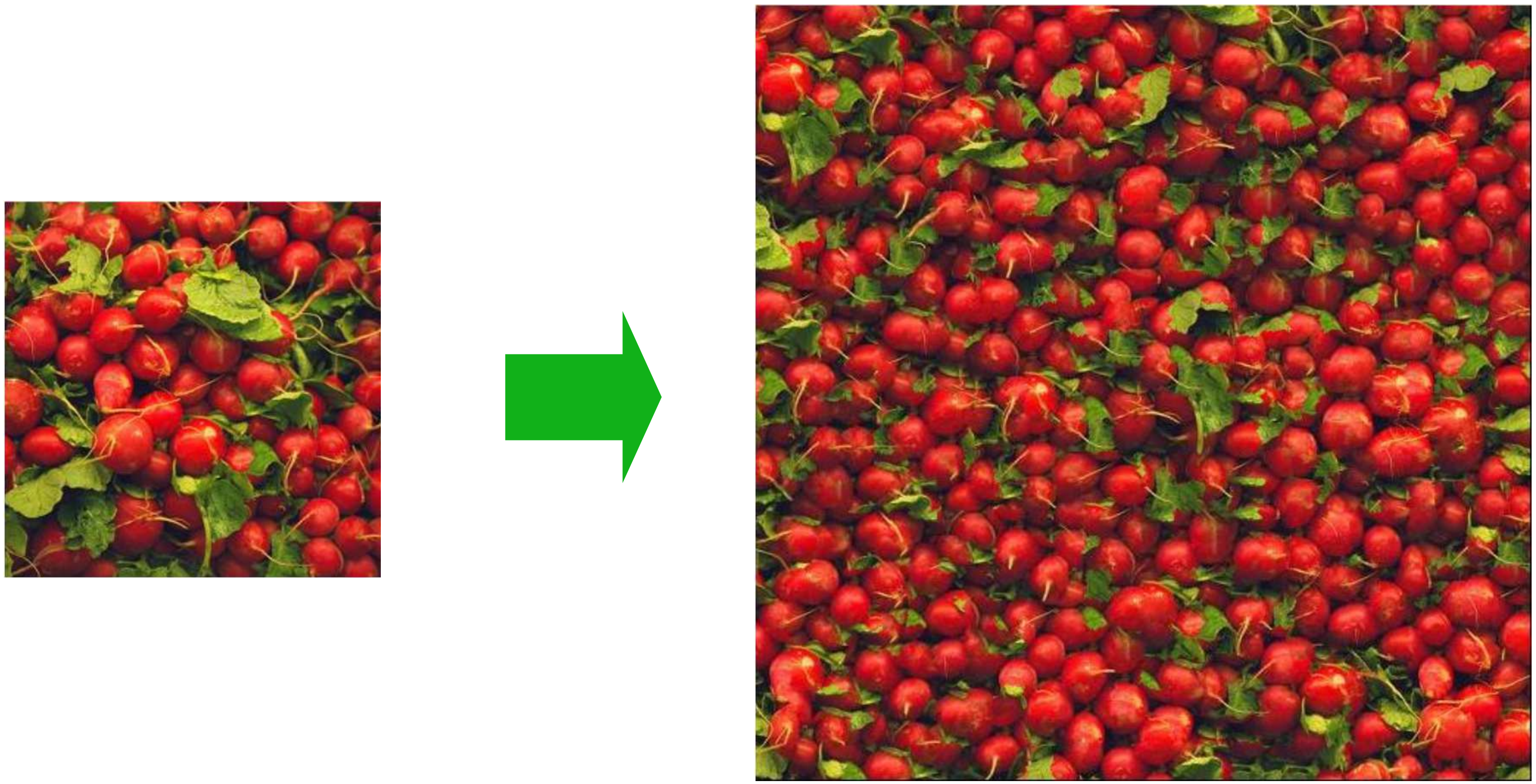
Texture

- Spatially repeating patterns

Texture Synthesis

- Create new samples of a given texture

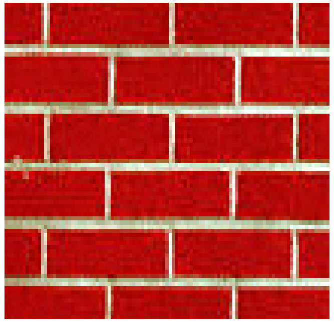


Texture Synthesis

- Useful
  - Repairing (inpainting)
  - Resizing
  - Texturing objects
  - …

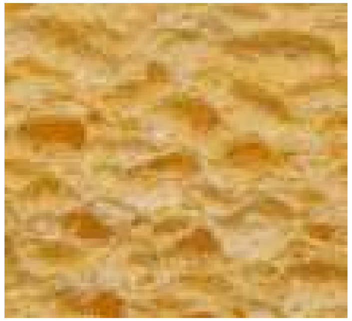
Texture Synthesis

- Challenging
  - Need to model the whole spectrum: from repeated to stochastic texture



repeated

stochastic



Both?

Alexei A. Efros and Thomas K. Leung

TEXTURE SYNTHESIS BY NON-PARAMETRIC SAMPLING

Statistical modeling

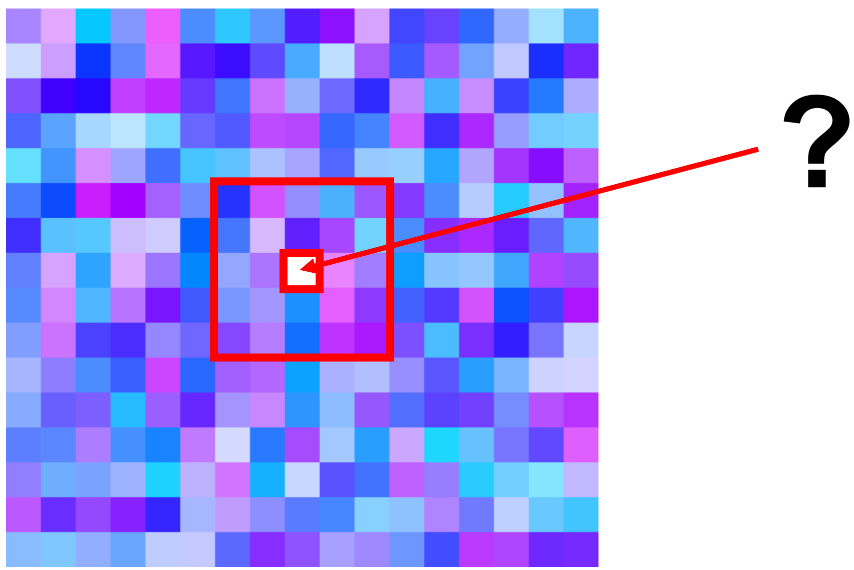
- Markov property:
  - Every pixel is only correlated with its neighborhood

$$P(\text{pixel}|\text{rest of image}) = P(\text{pixel}|N(\text{pixel}))$$

- Markov random field

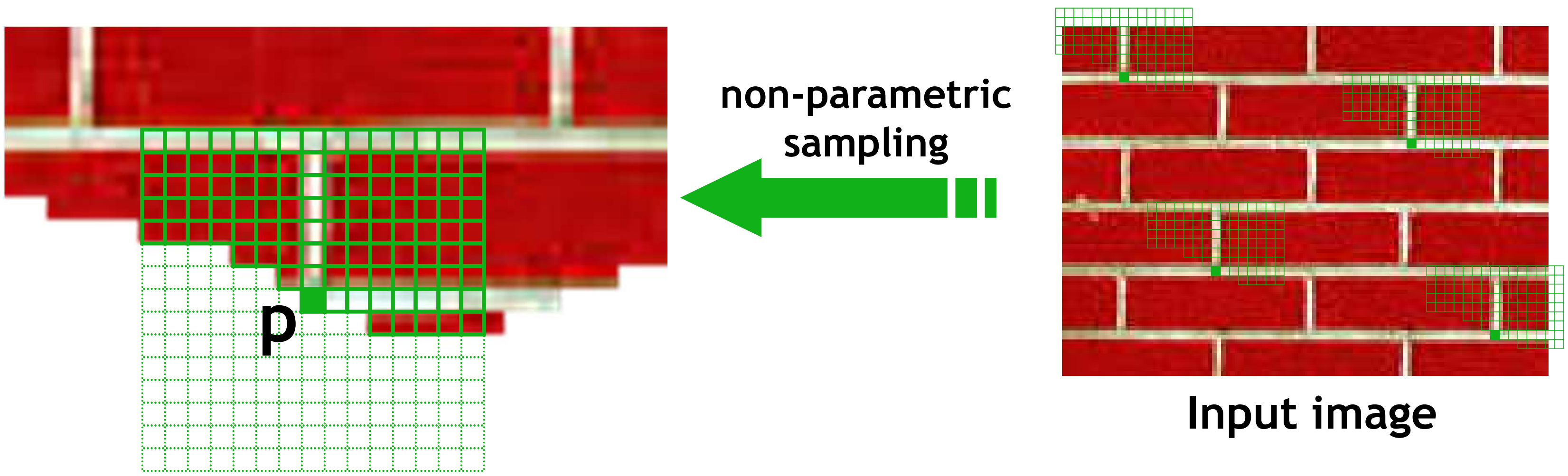
Statistical modeling

- Build the distribution of texture patch then missing pixel = argmax P(p, N(p))
  - Computationally expensive!



Non-parametric sampling

- Search the input texture for all sufficiently similar neighborhoods and pick on match at random

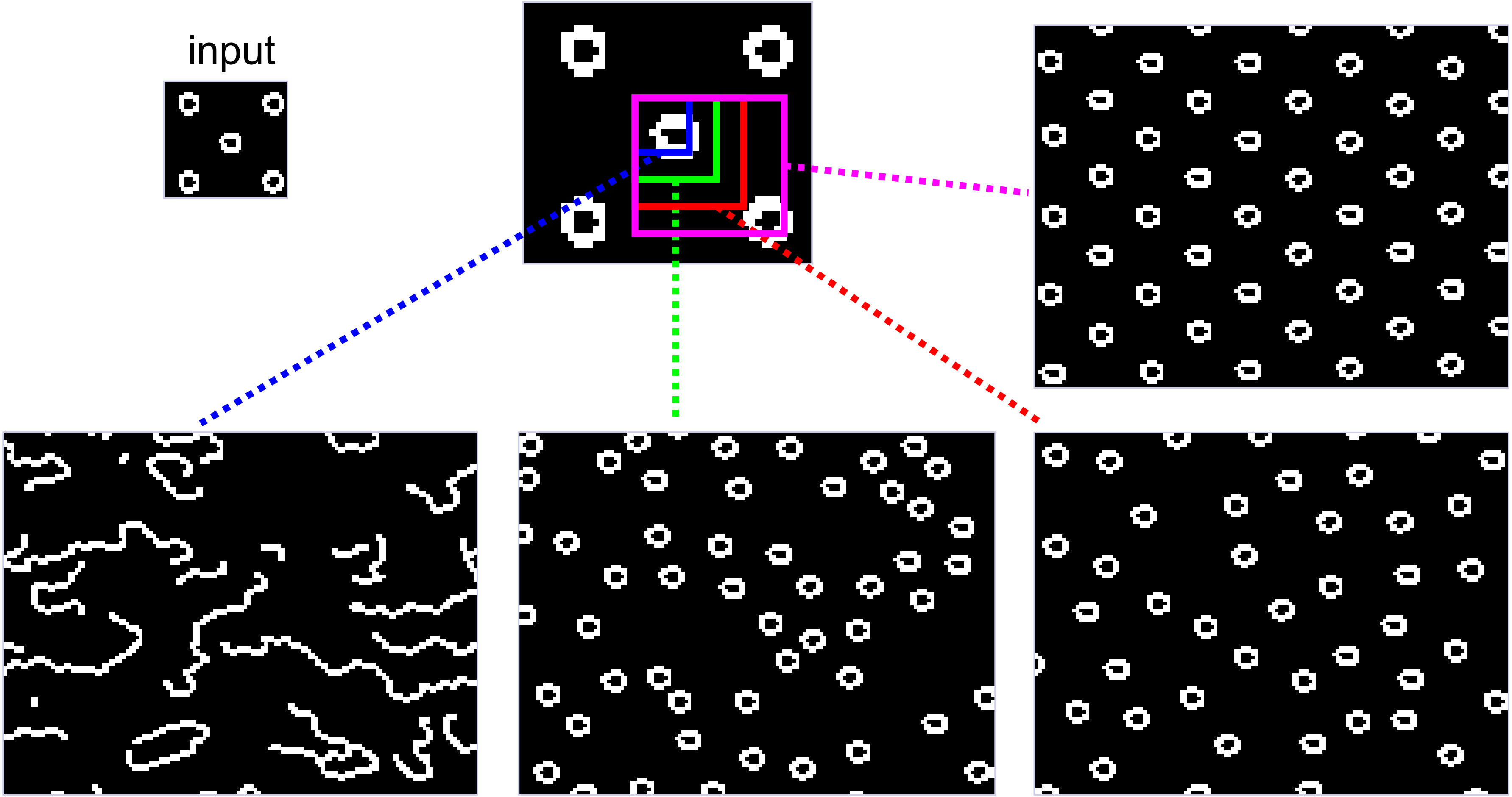


Synthesizing a pixel

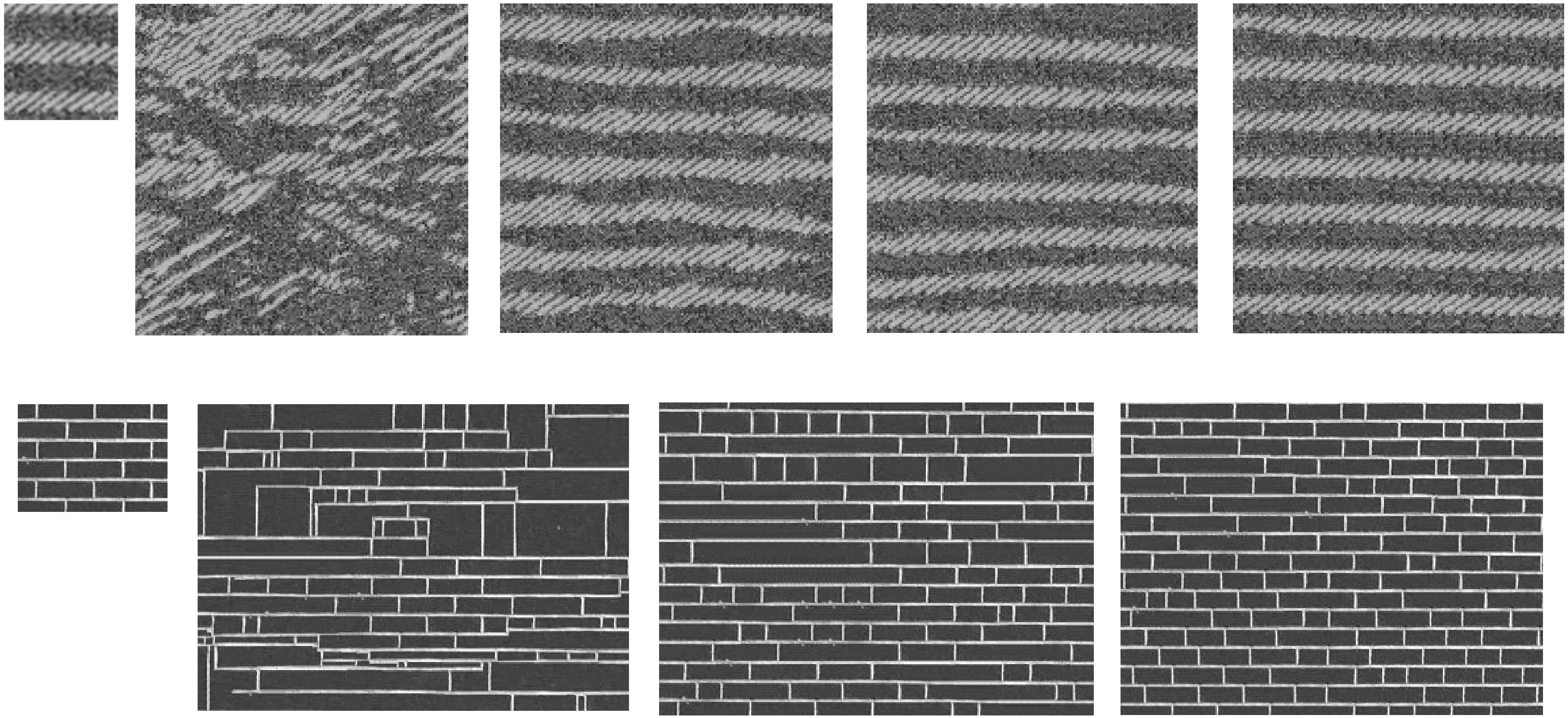
Details

- Random sampling from the set of candidates vs. picking the best candidate
- Initialization
  - Start with a few rows of white noise and grow in scanline order
  - Start with a “seed” in the middle and grow outward in layers
- Hole filling: growing is in “onion skin” order
  - Within each “layer”, pixels with most neighbors are synthesized first
  - Normalize error by the number of known pixels
  - If no close match can be found, the pixel is not synthesized until the end

Varying Window Size

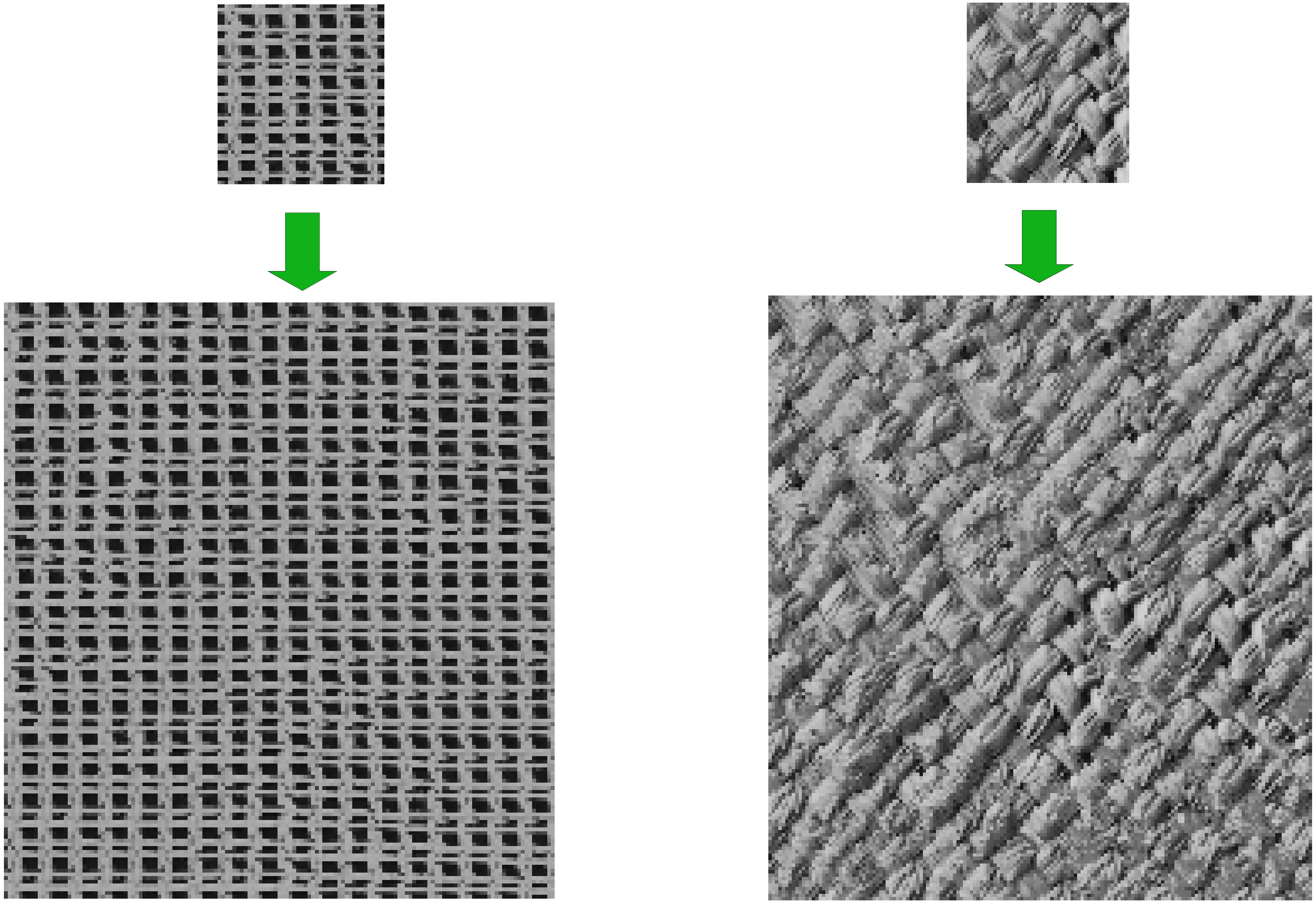


Varying Window Size

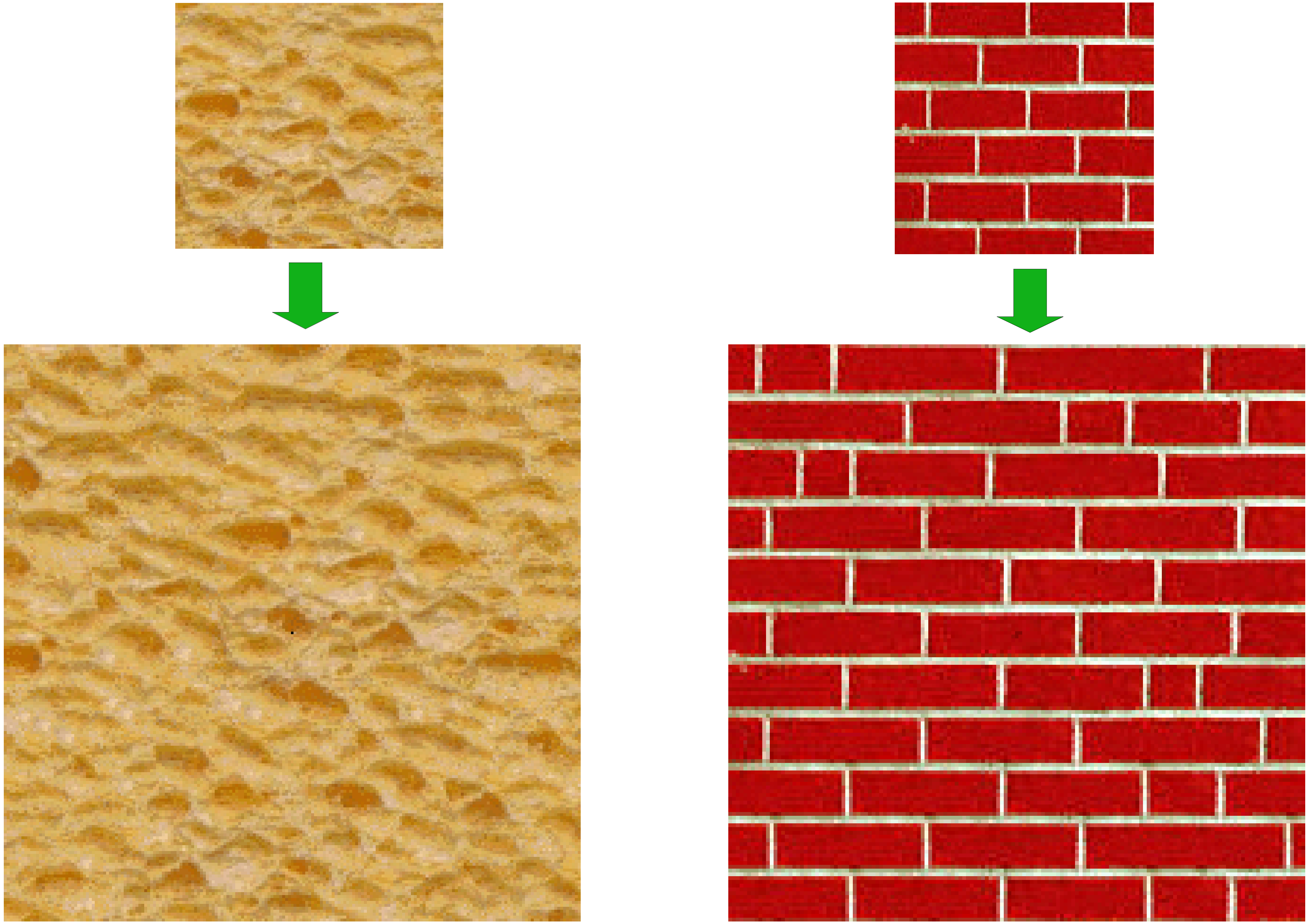


Increasing window size

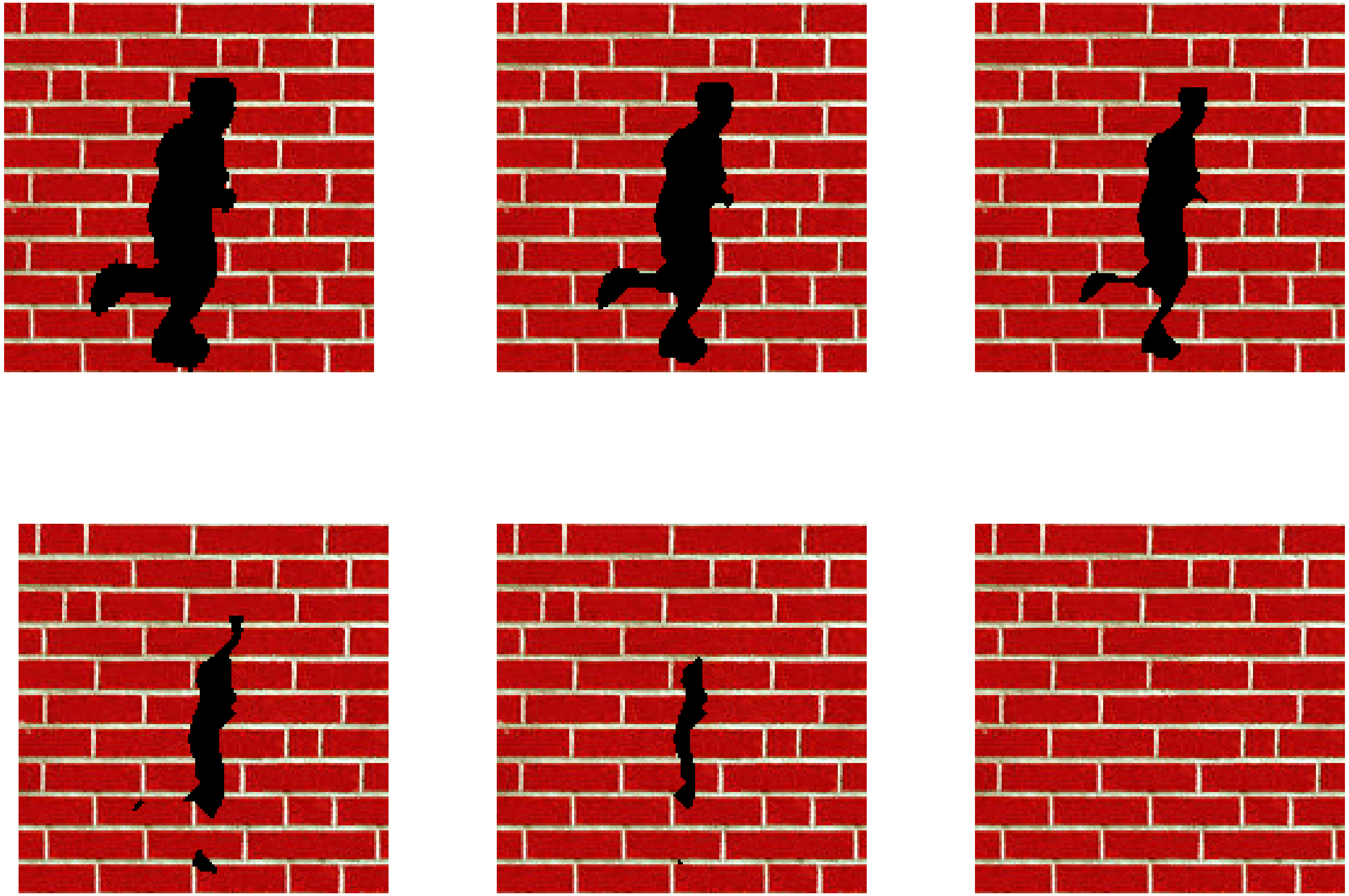
Results



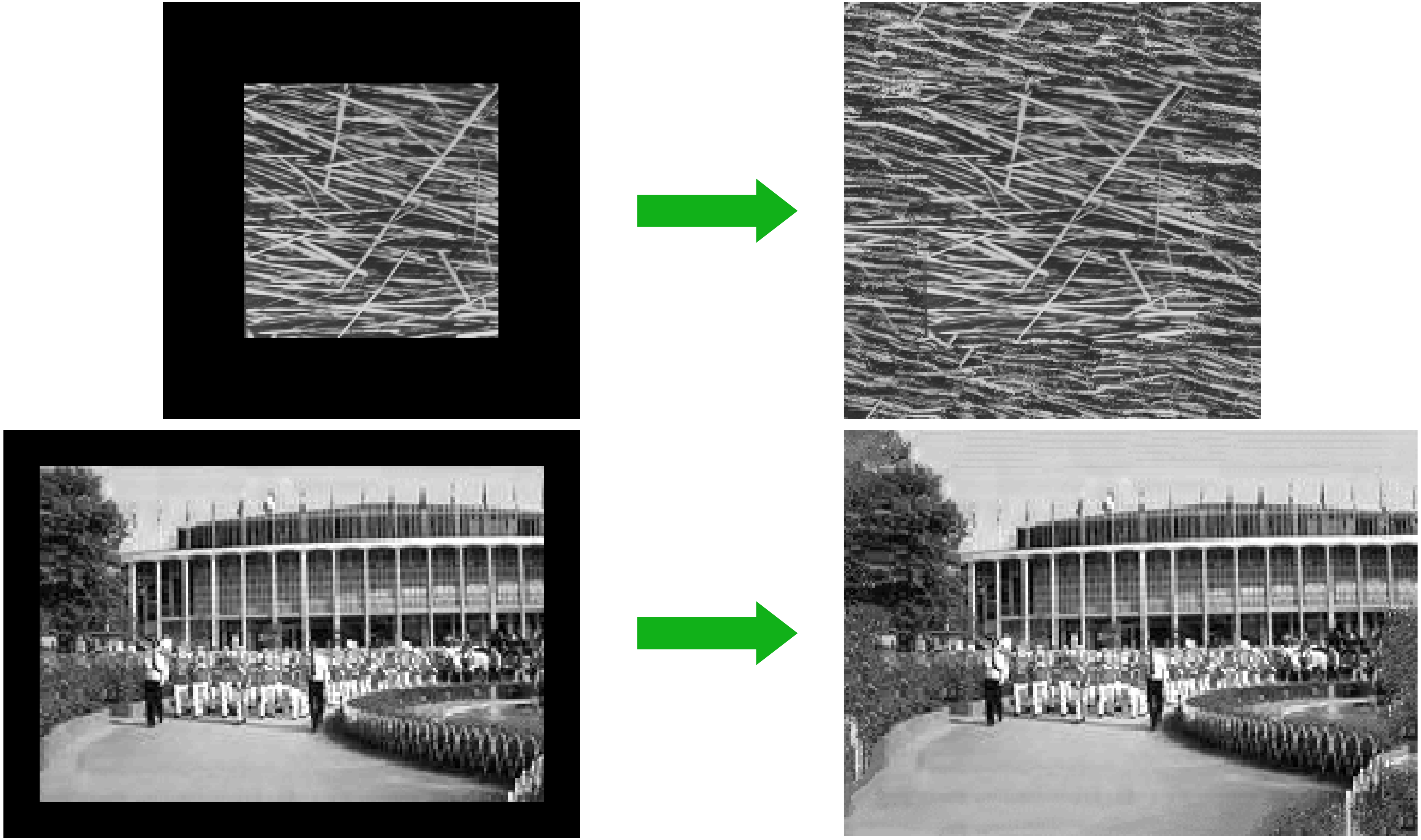
Results



Hole filling

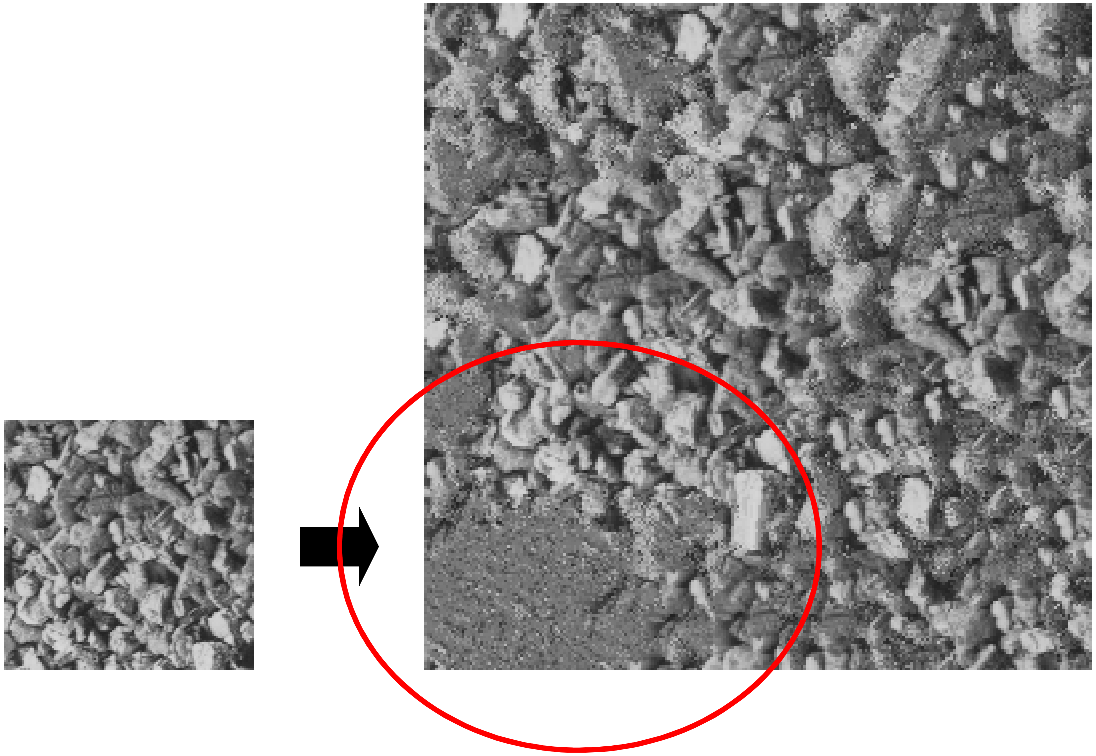


Extrapolation



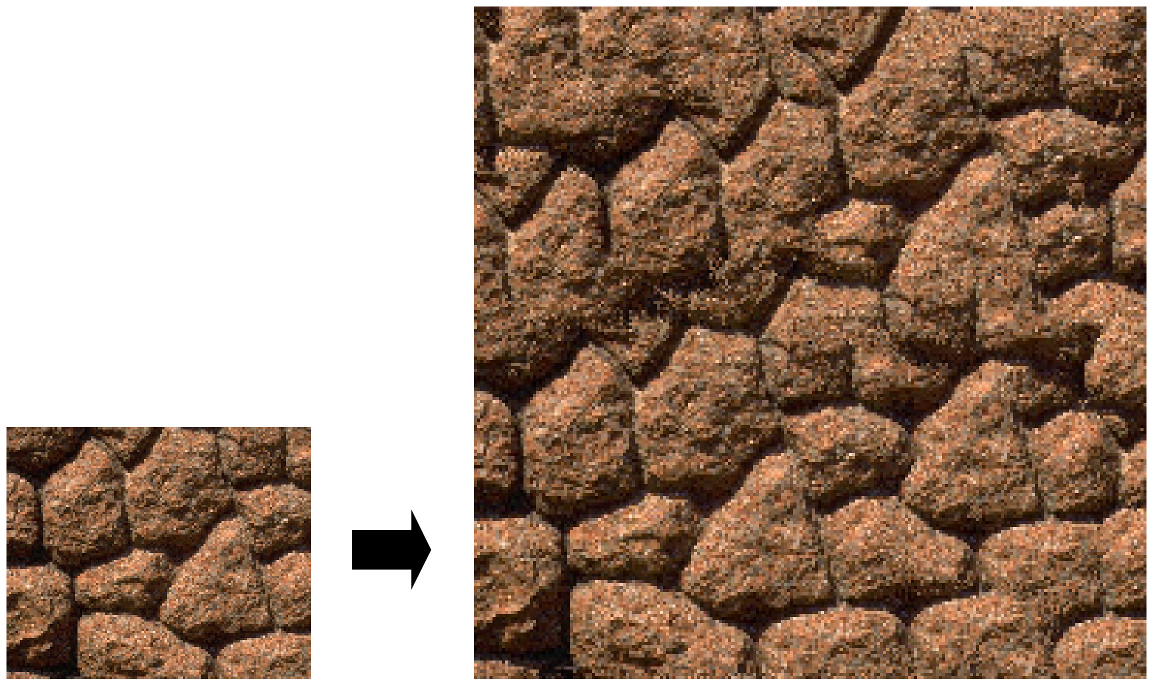
Failure cases

- A local method
  - may trapped to produce garbage



Failure cases

- A local method
  - or copying a pattern



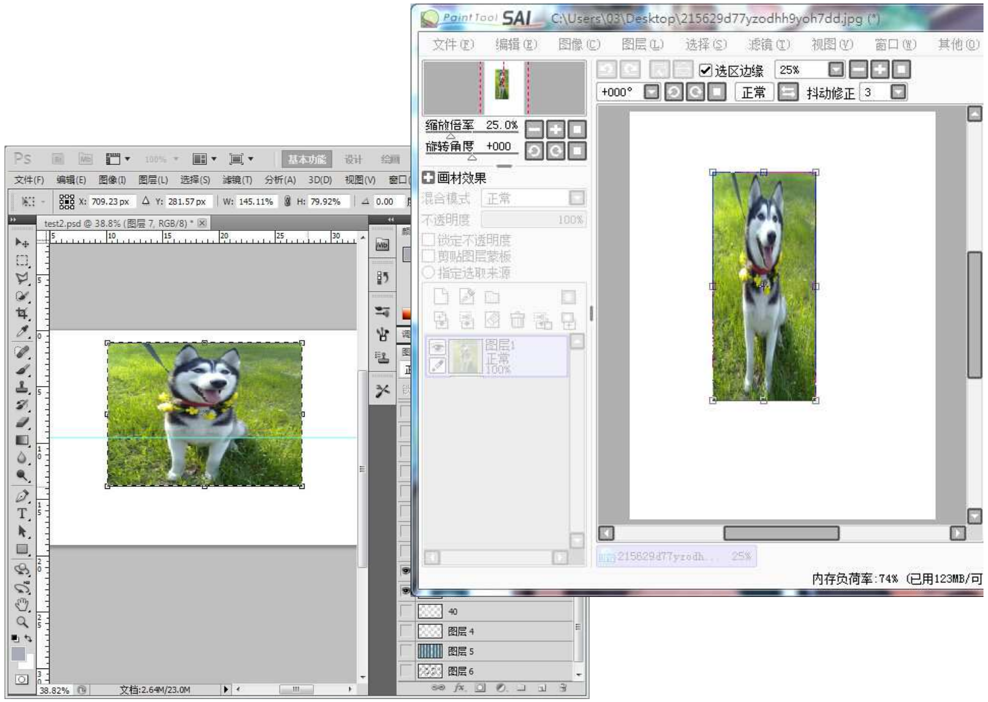
Failure cases

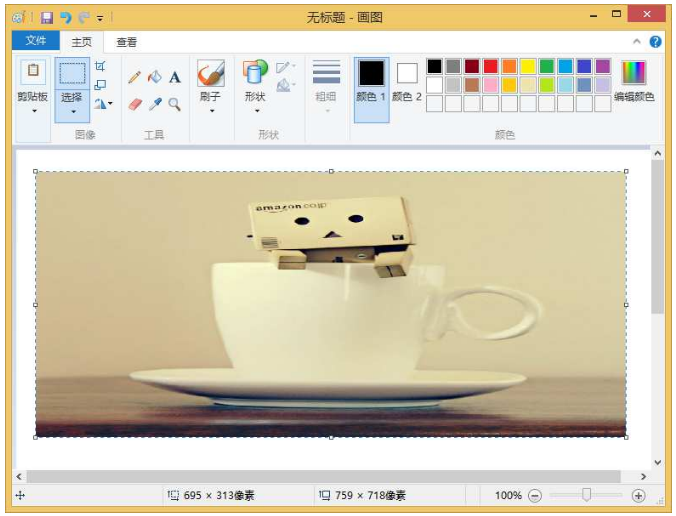
- A local method
  - Global idea: optimize MRF with BP or GC!

- Original work is very slow!
  - Finding matches is expensive
  - To speed-up
    - Li-Yi Wei and Marc Levoy, "Fast Texture Synthesis using Tree-structured Vector Quantization," SIGGRAPH 2000

Image resizing

- One of the most useful image operations
  - Virtually available in all image software



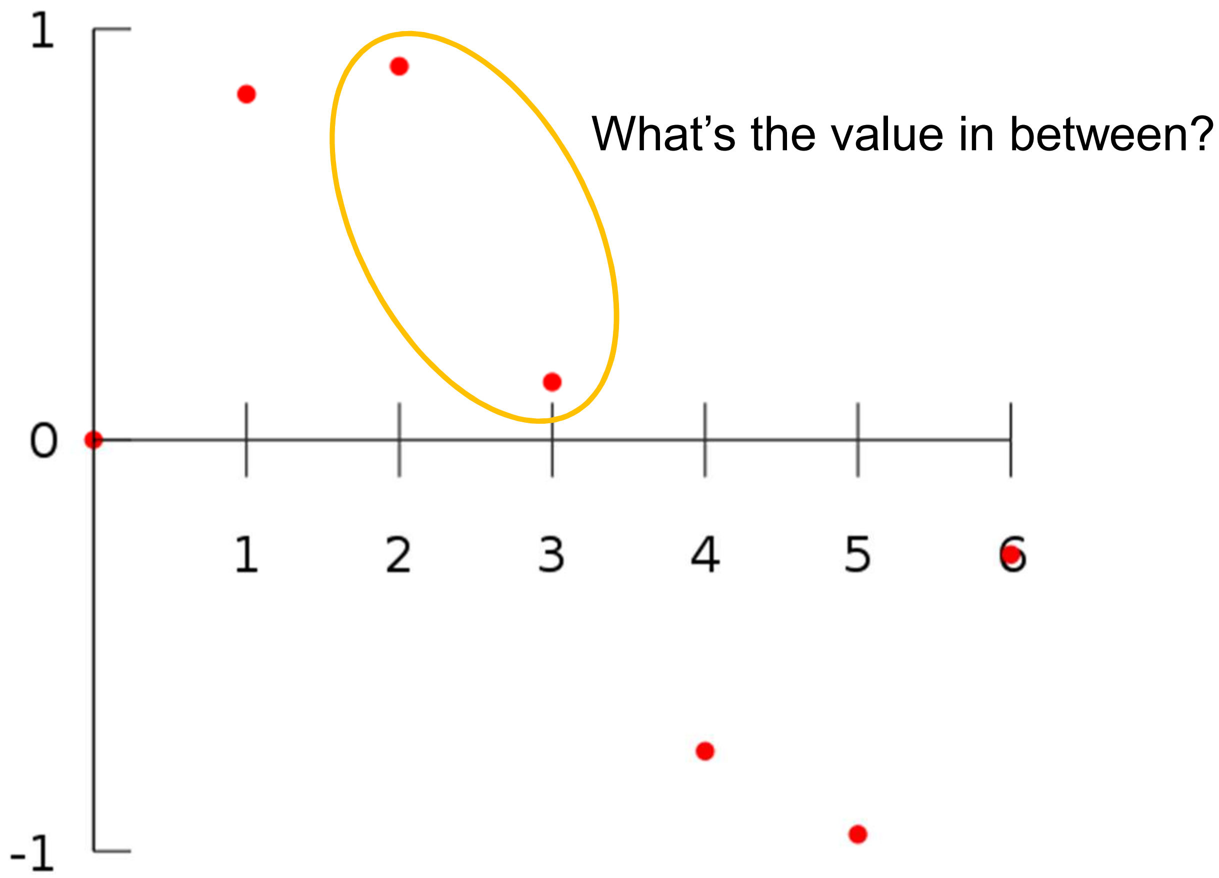


Imagine a great many huge bunch of screenshots of image editing software here …

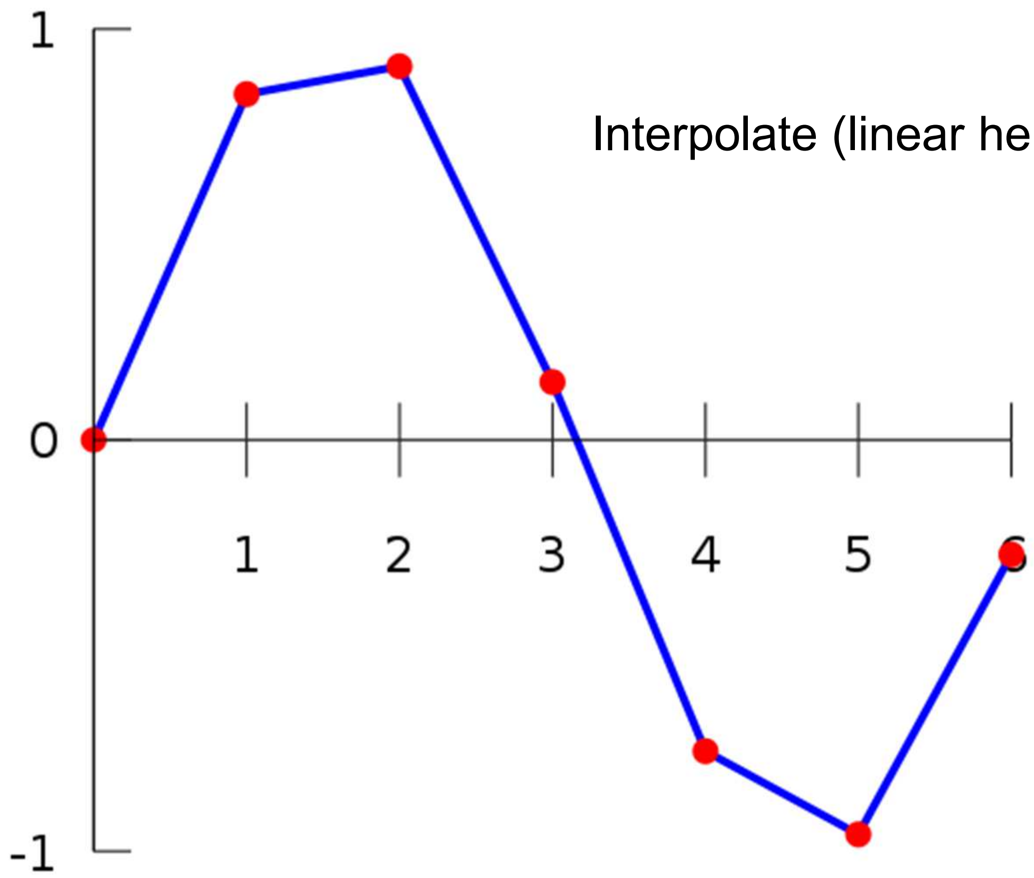
Magnification

- If $n \times m \rightarrow 2n \times 2m$
  - 4x pixels (3x more)
  - Where to find these missing pixels?

Interpolation

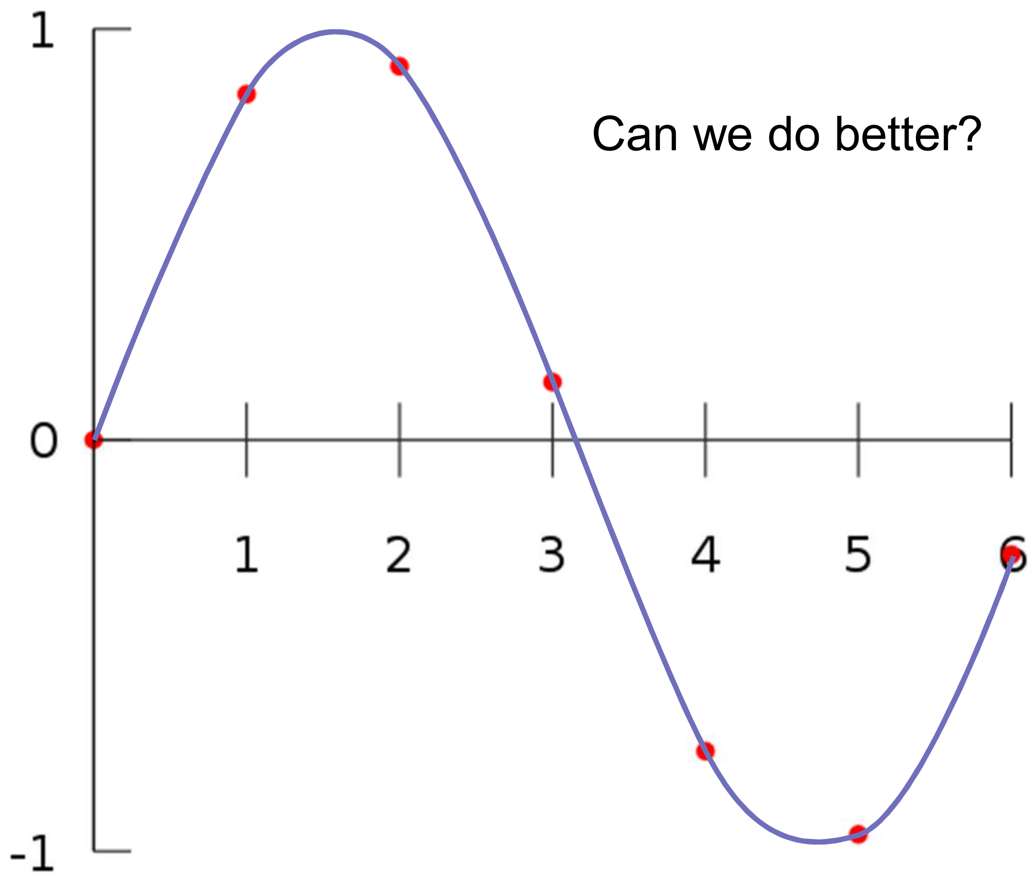


Interpolation



Interpolate (linear here)

Interpolation



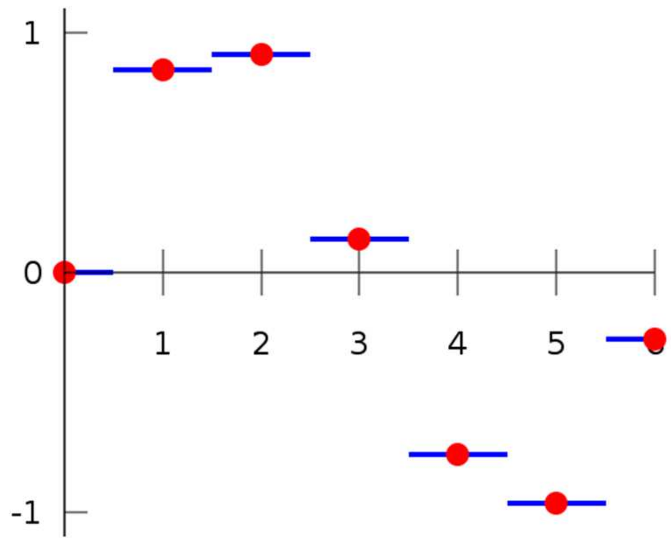
Can we do better?

Image interpolation

- Nearest-Neighbor Interpolation
- Bilinear Interpolation
- Bicubic Interpolation
- …

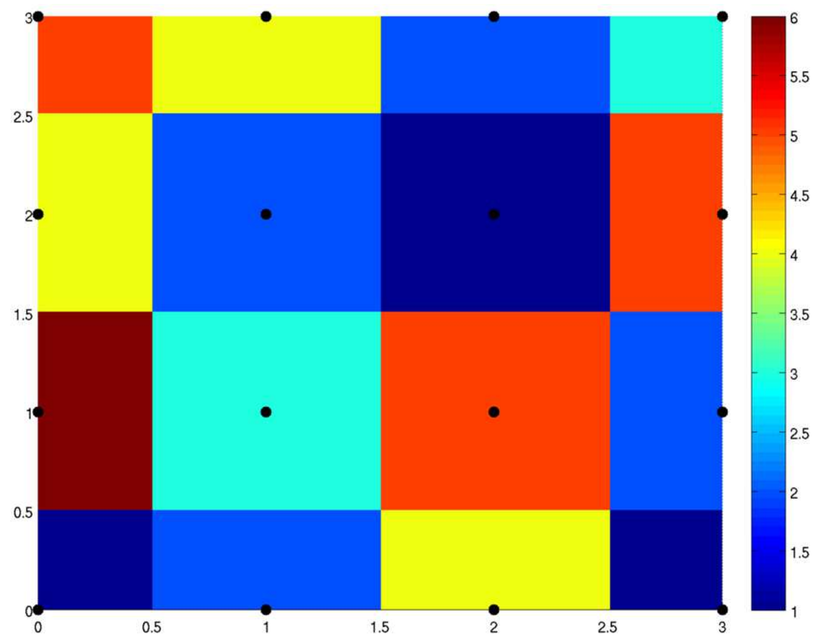
Nearest-Neighbor

- Just take the value from nearest sampling



Nearest-Neighbor

- "Blocky"

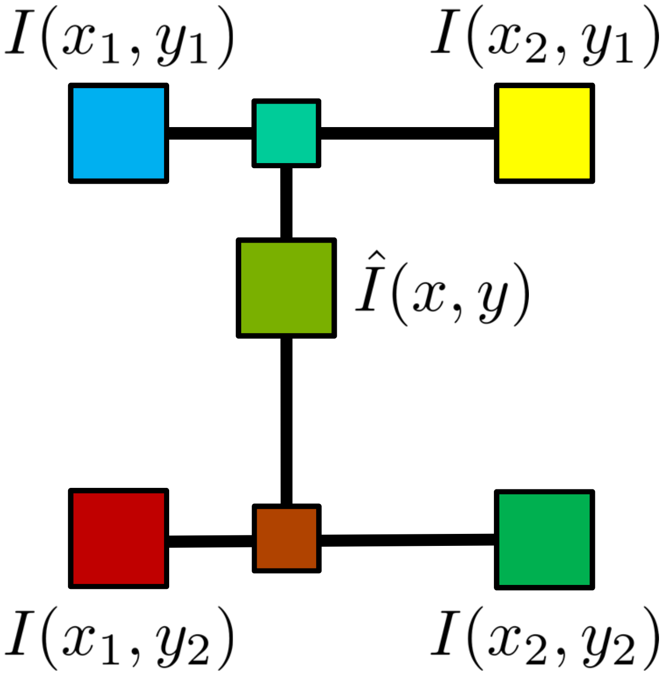


Nearest-Neighbor

- NN usually produces blocky result
  - Useful when zoom-in and manipulating pixels
  - But visually awful
  - Value “jumps” in the unknown domain

Bilinear

- Linearly change in the unknown domain, left to right, top to bottom.



$$[c_0, c_1](\alpha) := (1 - \alpha)c_0 + \alpha c_1$$

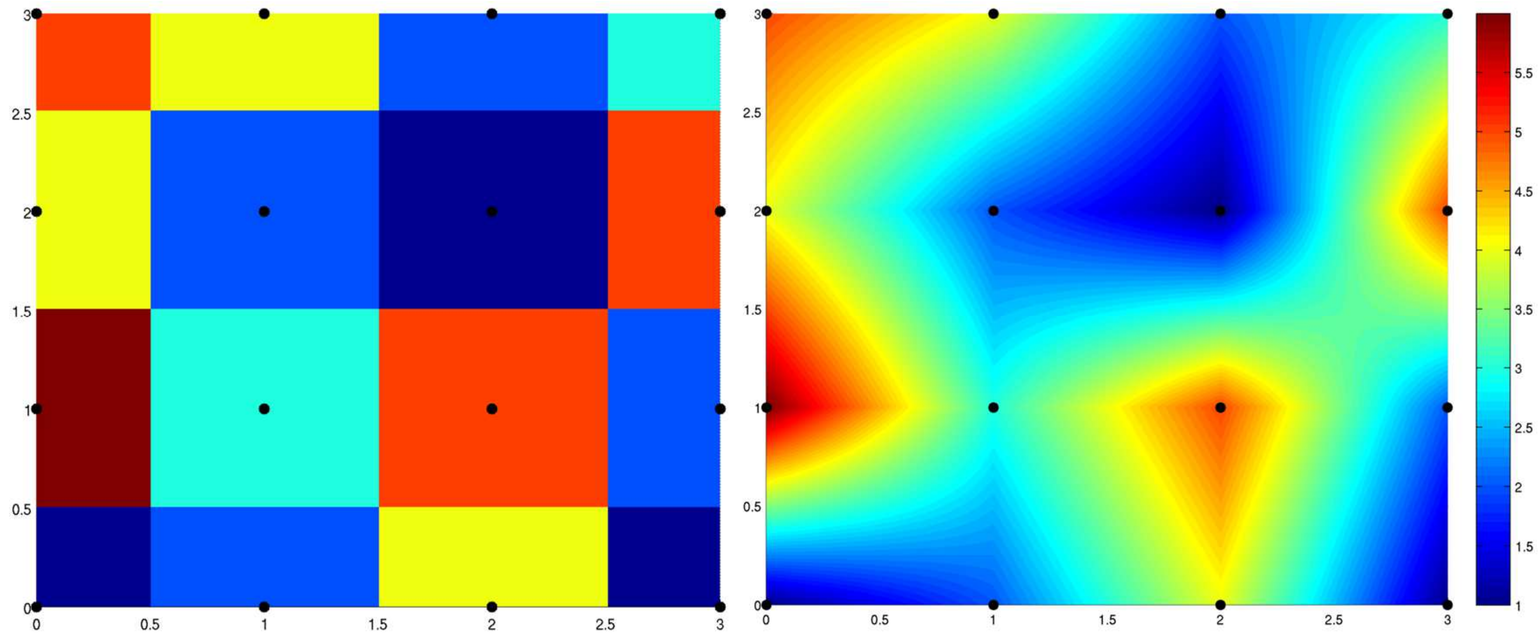
$$\underline{\hat{I}(x, y_1)} = [\underline{I(x_1, y_1)}, \underline{I(x_2, y_1)}]\left(\frac{x - x_1}{x_2 - x_1}\right)$$

$$\underline{\hat{I}(x, y_2)} = [\underline{I(x_1, y_2)}, \underline{I(x_2, y_2)}]\left(\frac{x - x_1}{x_2 - x_1}\right)$$

$$\underline{\hat{I}(x, y)} = [\underline{\hat{I}(x, y_1)}, \underline{\hat{I}(x, y_2)}]\left(\frac{y - y_1}{y_2 - y_1}\right)$$

Bilinear

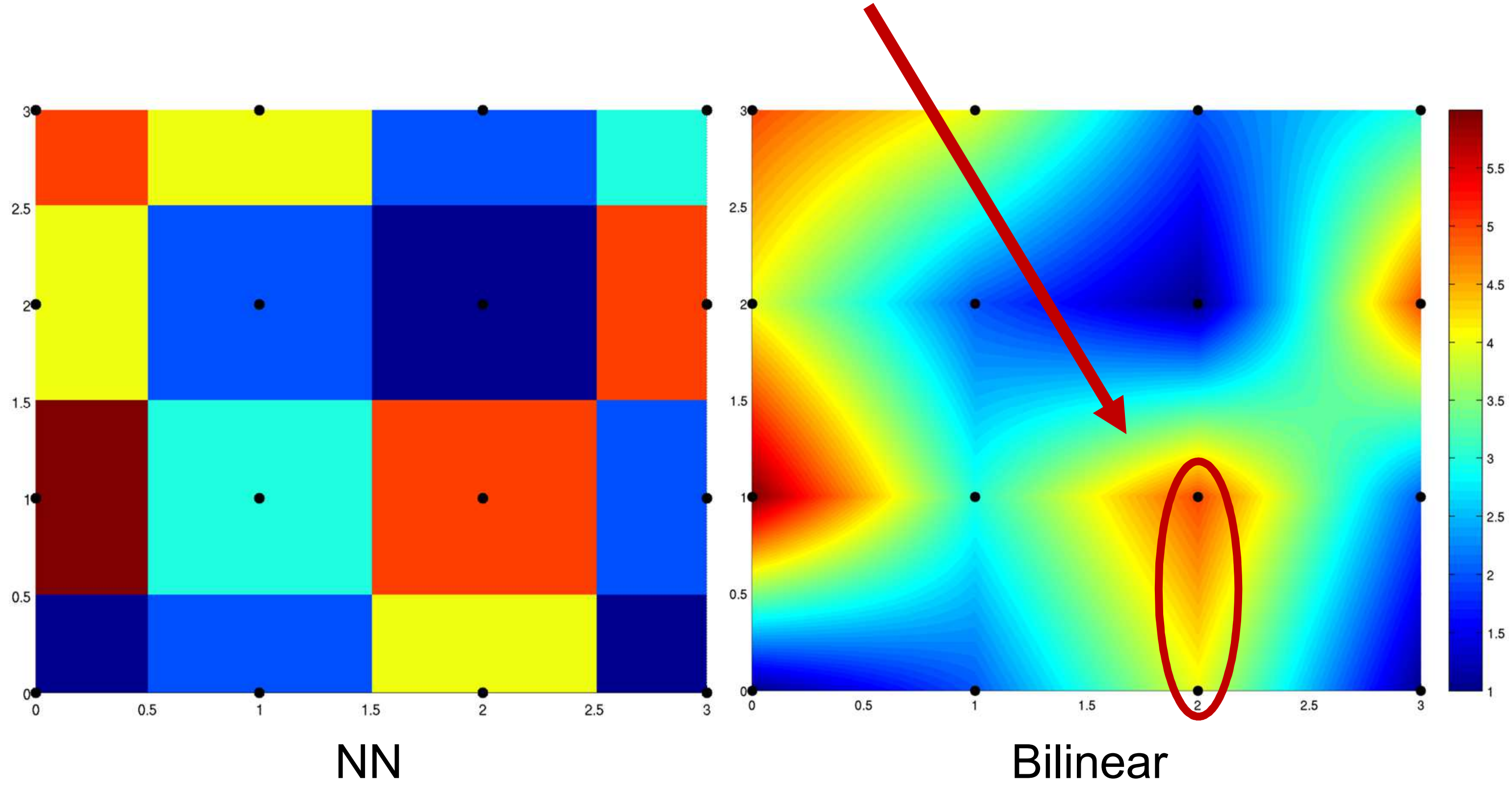
- No more blocky



NN Bilinear

Bilinear

- No more blocky
  - but derivatives are not continuous



Bicubic

- Constraint the derivatives on 4 corners

$$\hat{I}(x,y) = \sum_{i=0}^{3}\sum_{j=0}^{3} a_{ij}x^i y^j$$

$$\hat{I}_x(x,y) = \sum_{i=1}^{3}\sum_{j=0}^{3} a_{ij}ix^{i-1} y^j$$

$$\hat{I}_y(x,y) = \sum_{i=0}^{3}\sum_{j=1}^{3} a_{ij}x^i jy^{j-1}$$

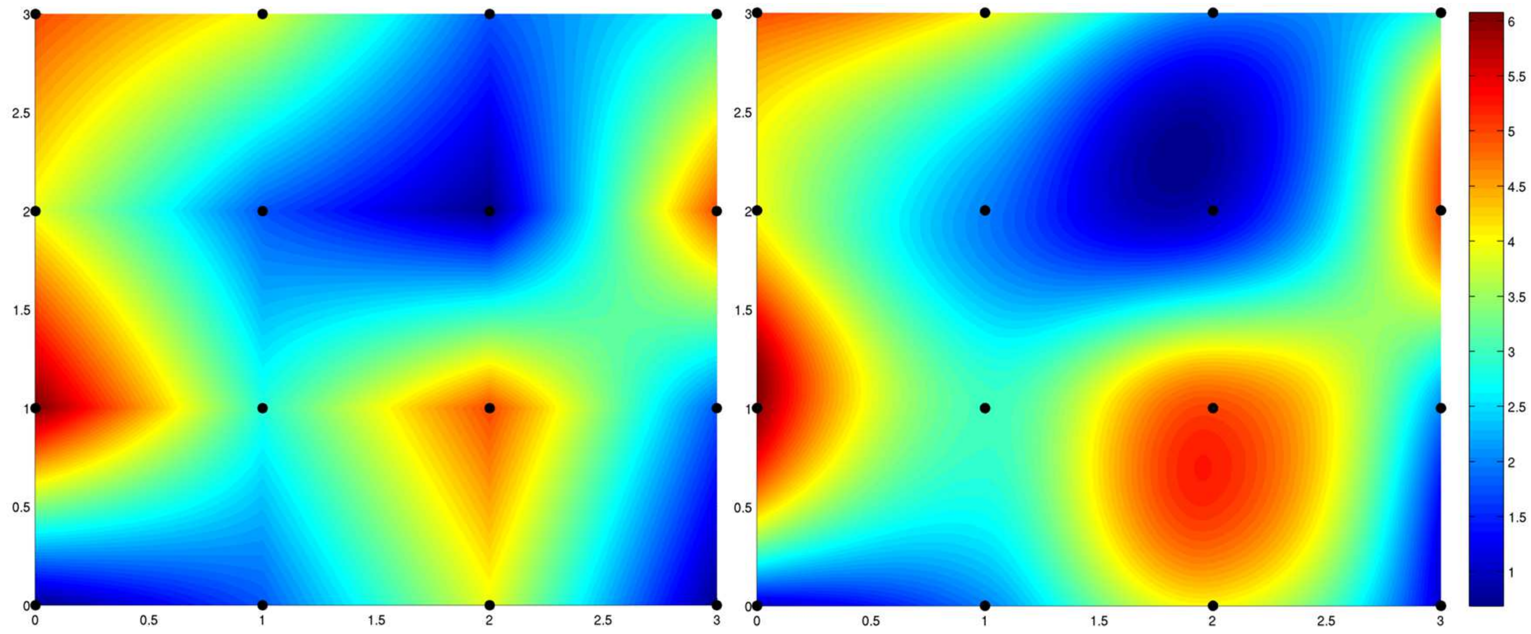
$$\hat{I}_{xy}(x,y) = \sum_{i=1}^{3}\sum_{j=1}^{3} a_{ij}ix^{i-1} jy^{j-1}$$

ex (constraint dx at top-right):

$$\hat{I}_x(1,0) = a_{10} + 2a_{20} + 3a_{30} = \underset{\text{known}}{I_x(1,0)}$$

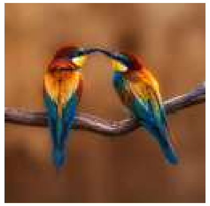
Bicubic

- Solve16 coefficients a_{ij} from 16 constraints

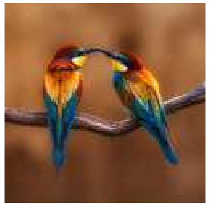


Bilinear Bicubic

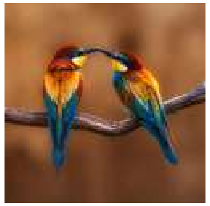
Nearest Neighbor



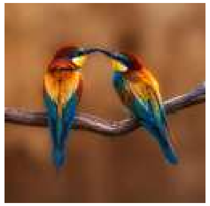
Bilinear



Bicubic

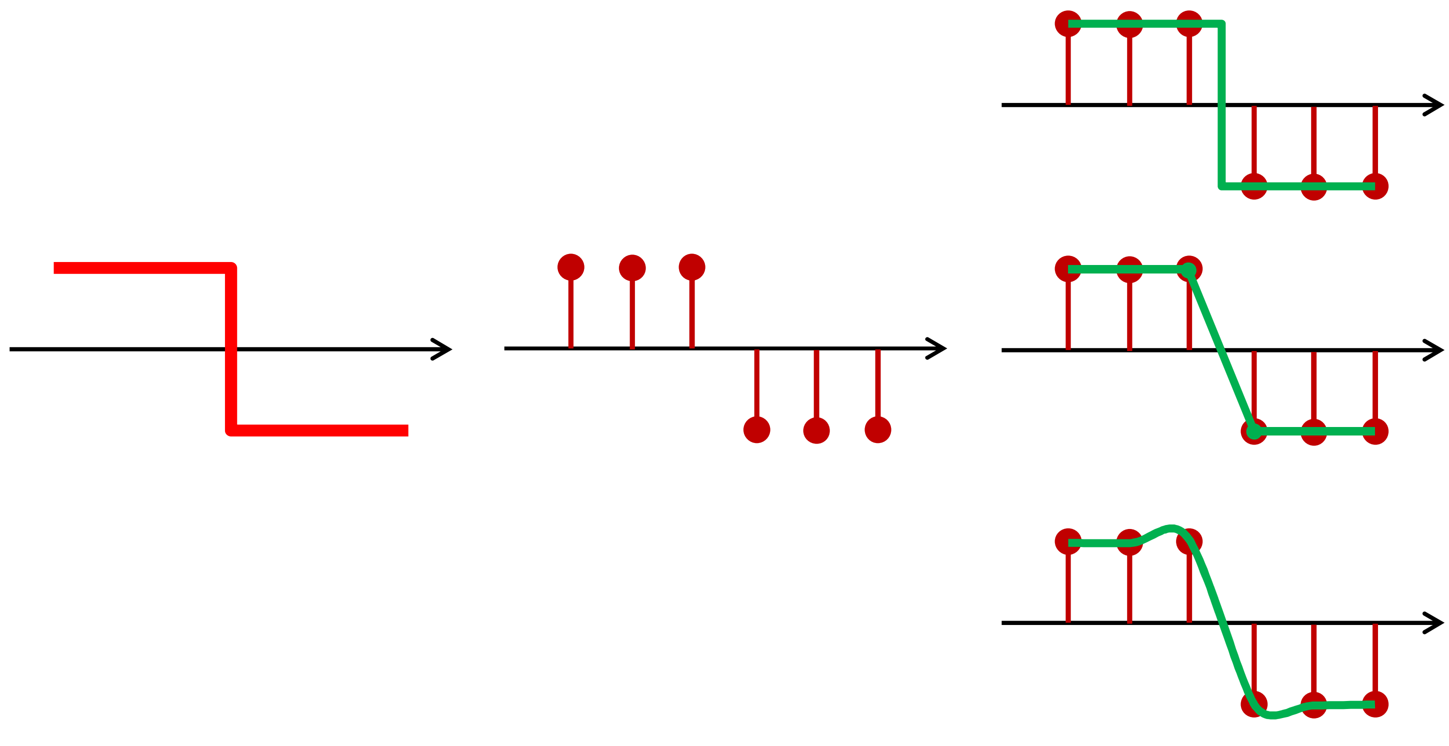


True



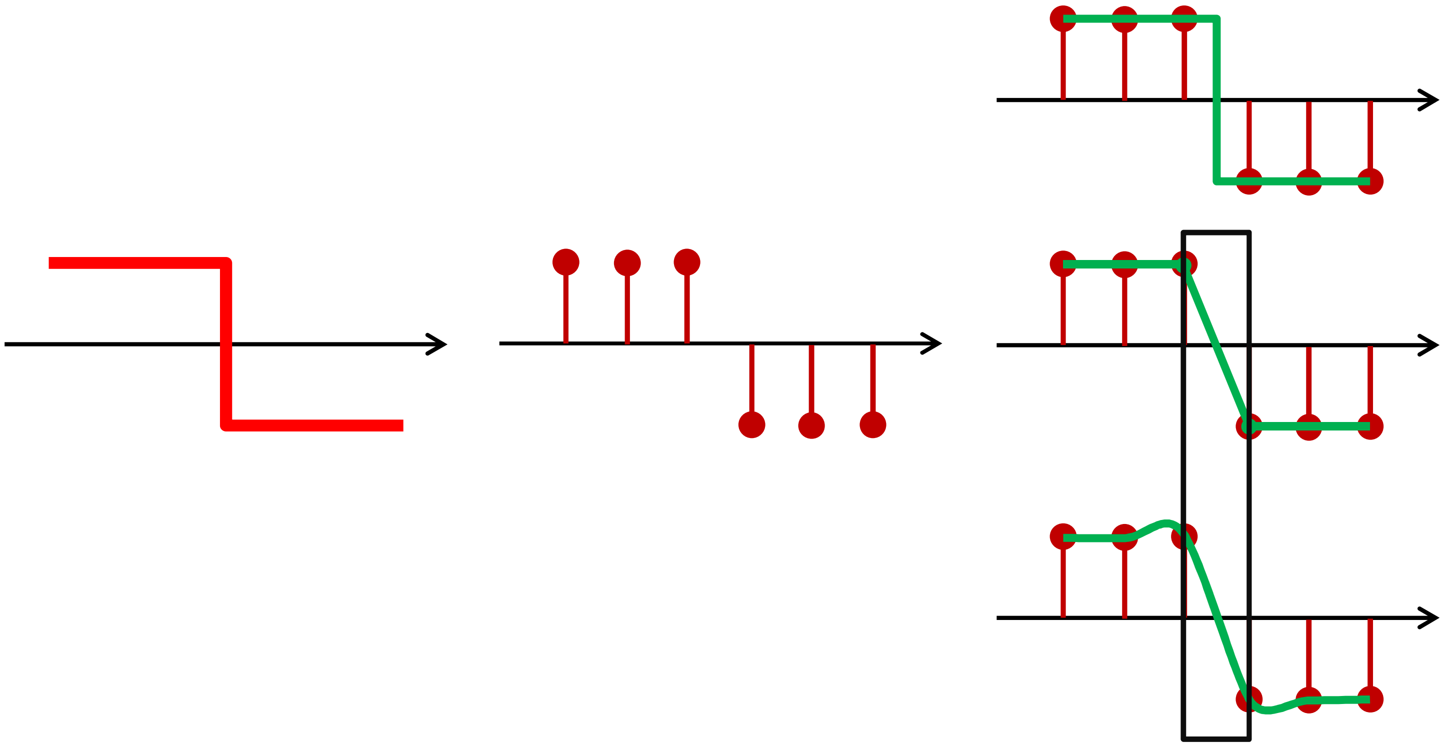
Interpolation

- Takes only neighborhood information



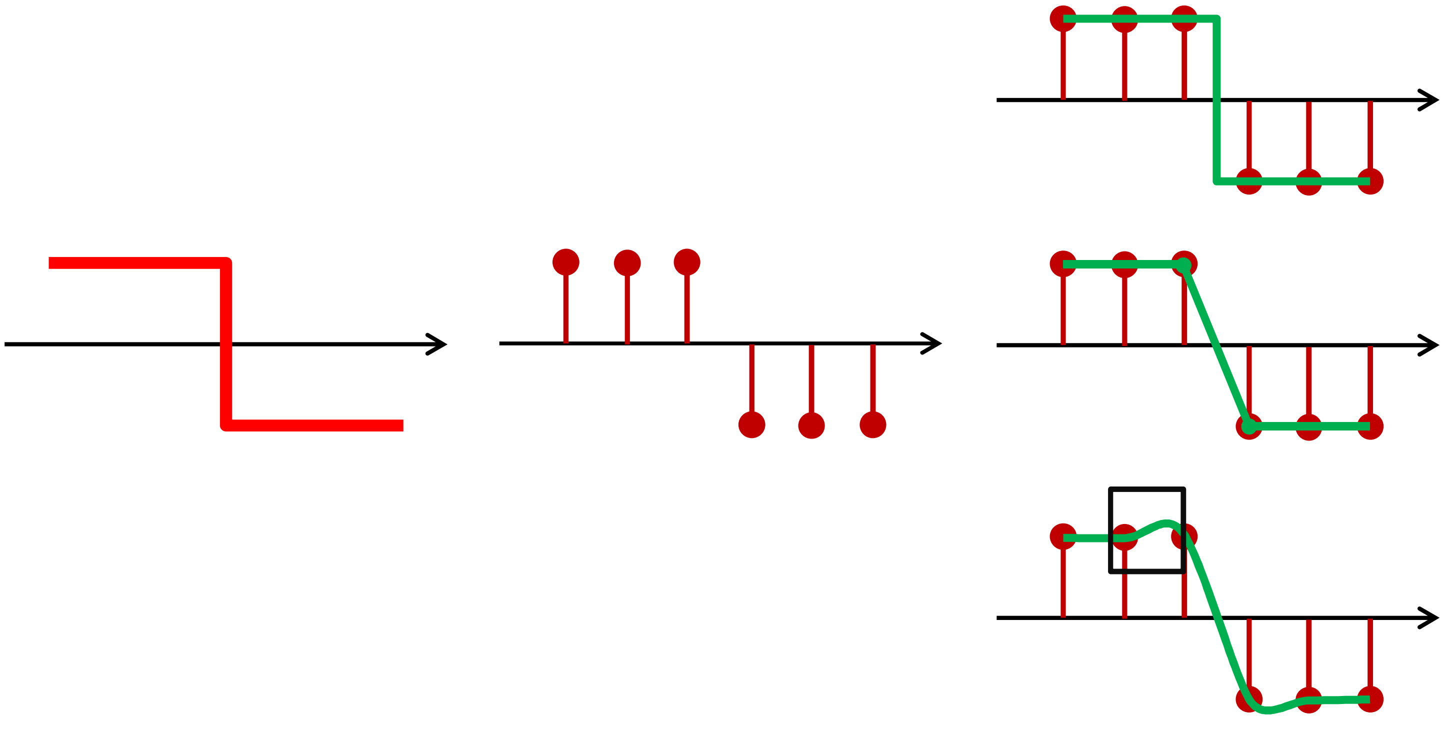
Interpolation

- Takes only neighborhood information
  - Blurring



Interpolation

- Takes only neighborhood information
  - Ringing



Super-resolution

- Try obtaining the high-detailed image without blurring/ringing/other artifacts…

Super-resolution

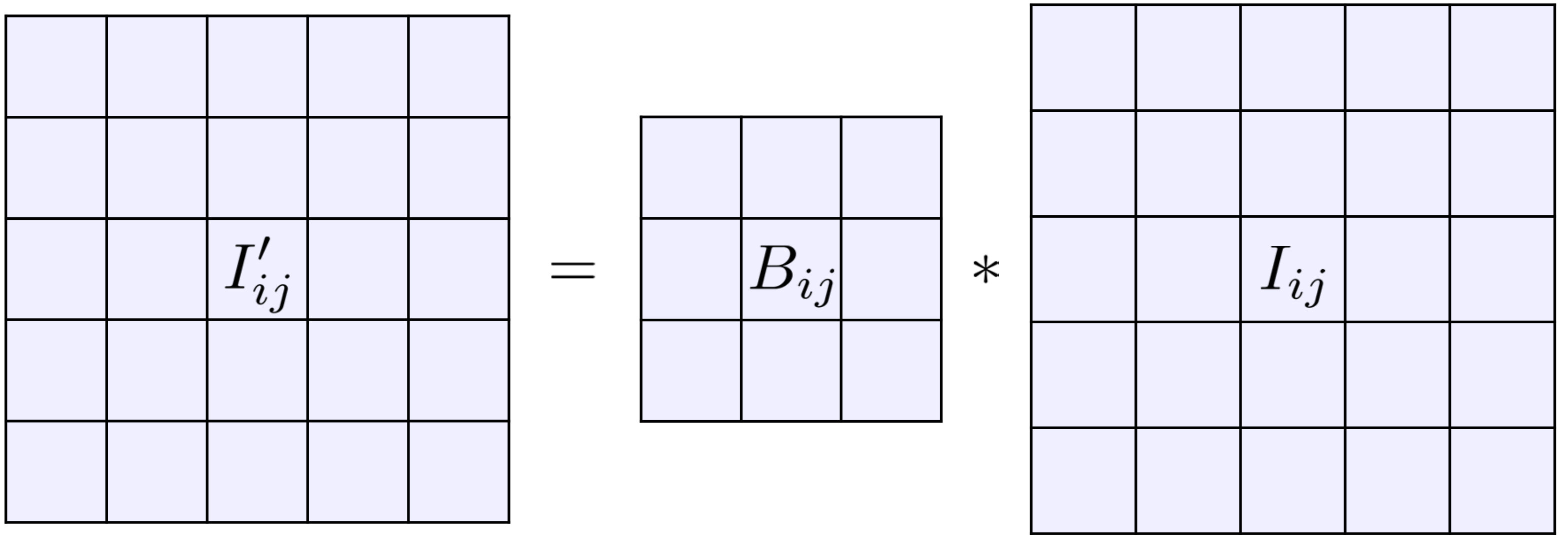
- Given a low-res image, blurred and subsampled from true high-res original, recover the original.

$$J = (B * I) \downarrow$$

  - I – High resolution image
  - J – Low resolution image
  - B – Blur kernel
  - $(\cdot) \downarrow$ – Subsampling process

Super-resolution

- Blur and subsample are linear:

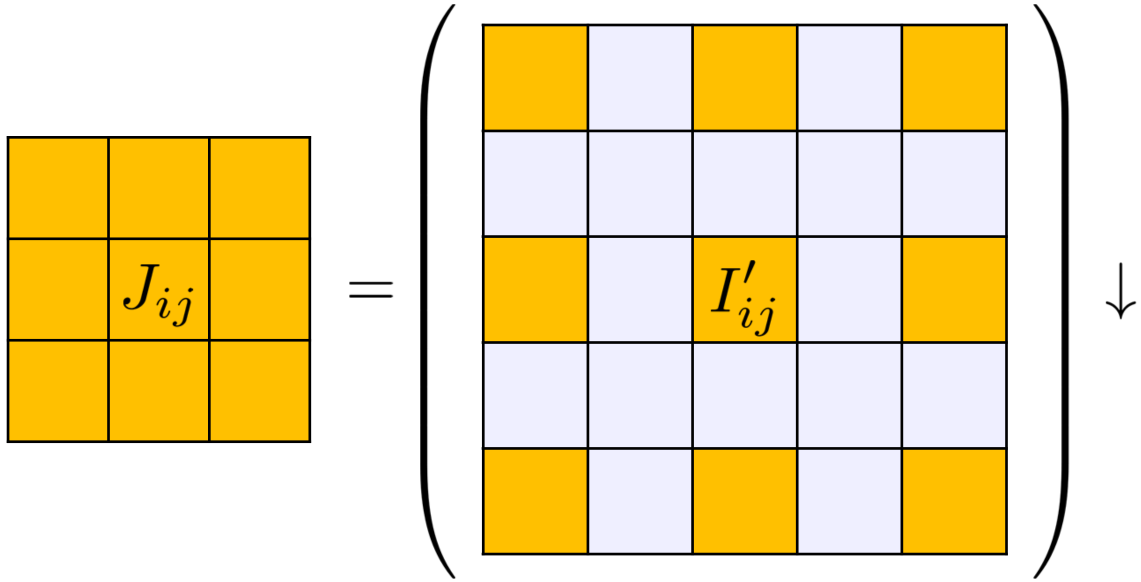


$$I'_{ij} = \sum_p \sum_q B_{pq} I_{i-p,j-q}$$

$$\mathrm{vec}(I') = M_B \mathrm{vec}(I)$$

Super-resolution

- Blur and subsample are linear:



$$\mathrm{vec}(J) = M_S \mathrm{vec}(I')$$

Super-resolution

- Super-resolution → solve linear equation?

$$\mathrm{vec}(J) = M_S \mathrm{vec}(I') = M_S M_B \mathrm{vec}(I)$$

  - Under constrained
    - Infinitely many solutions
    - Ill-posed

Super-resolution

- By having many low-res image of the same original
  - Multi-image super-resolution

$$\begin{cases} \mathrm{vec}(J_1) = M_1 \mathrm{vec}(I) \\ \mathrm{vec}(J_2) = M_2 \mathrm{vec}(I) \\ \vdots \\ \mathrm{vec}(J_n) = M_n \mathrm{vec}(I) \end{cases}$$

Super-resolution

- Or if we have prior knowledge of I
  - Prior-regularized super-resolution

$$\max P(I|J)$$

  - But where to get prior statistics?

William T. Freeman, Thouis R. Jones, and Egon C. Pasztor

EXAMPLE-BASED SUPER-RESOLUTION

Basic Idea

- Missing high-res information may be learned from other images.
- Edges and corners are more important.

Training

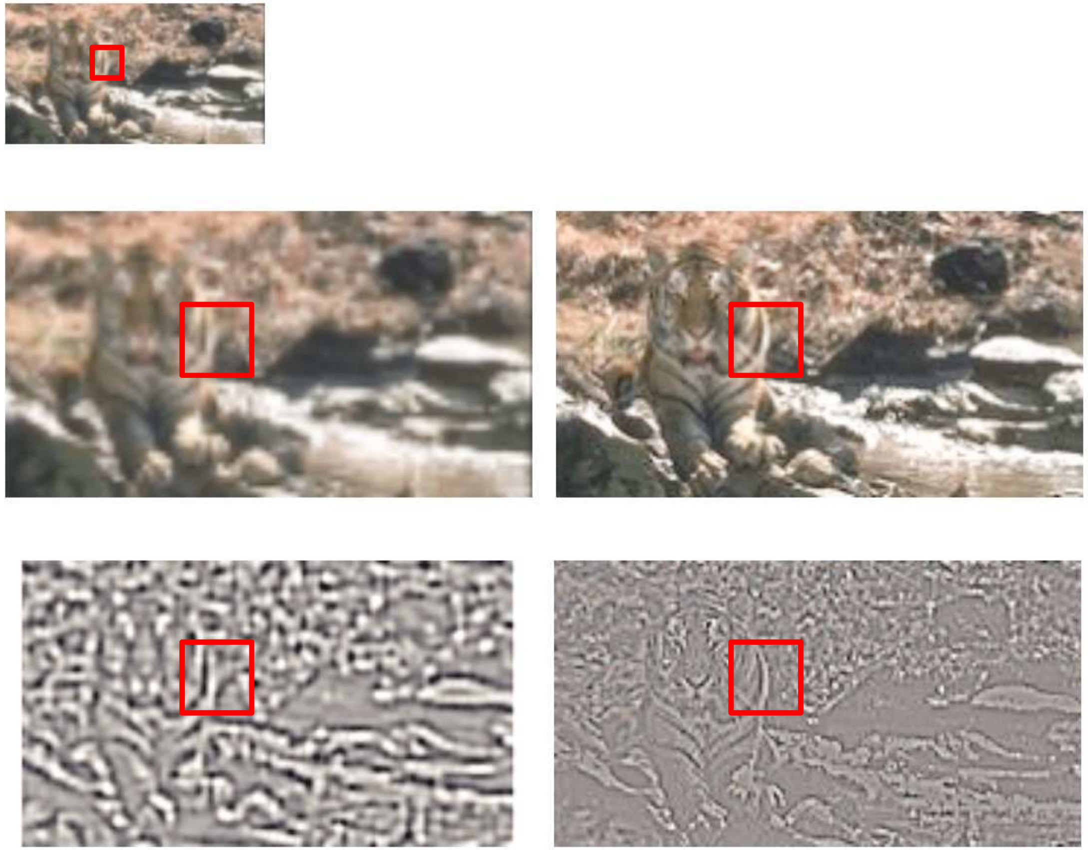
- Make database!
- Record lo-res/hi-res correspondences

Training

- Interpolate the low-res one

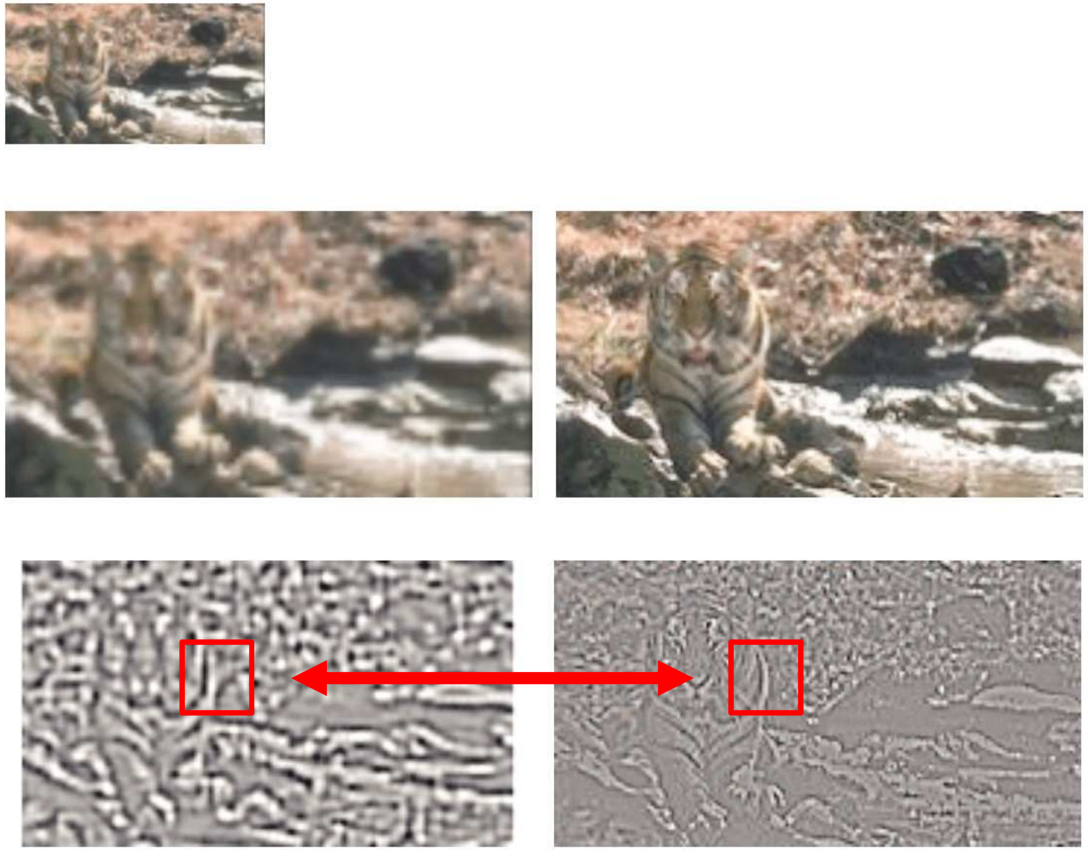
Training

- Interpolate the low-res one
- Apply hi-pass filter
  - Keep only hi-freq



Training

- Interpolate the low-res one
- Apply hi-pass filter
  - Keep only hi-freq
- Make database

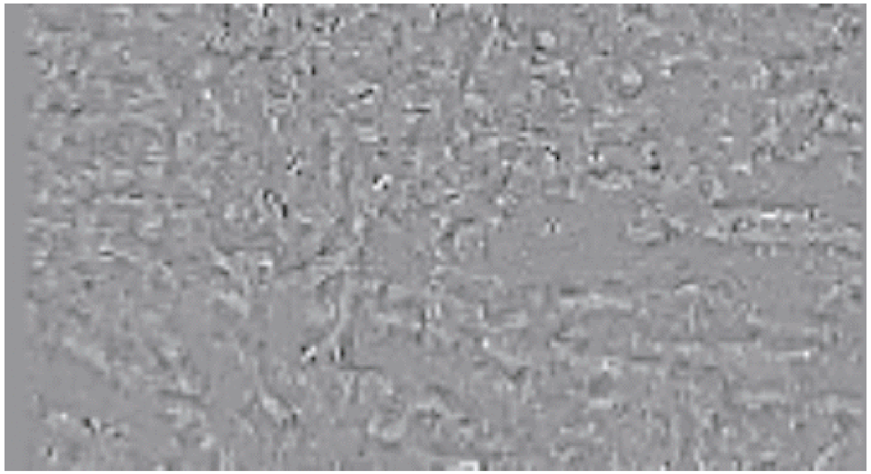


Reconstruction

- For a new low-res image:
  - Interpolate
  - Match closest low-res patch in database
  - Update with high-res counterpart

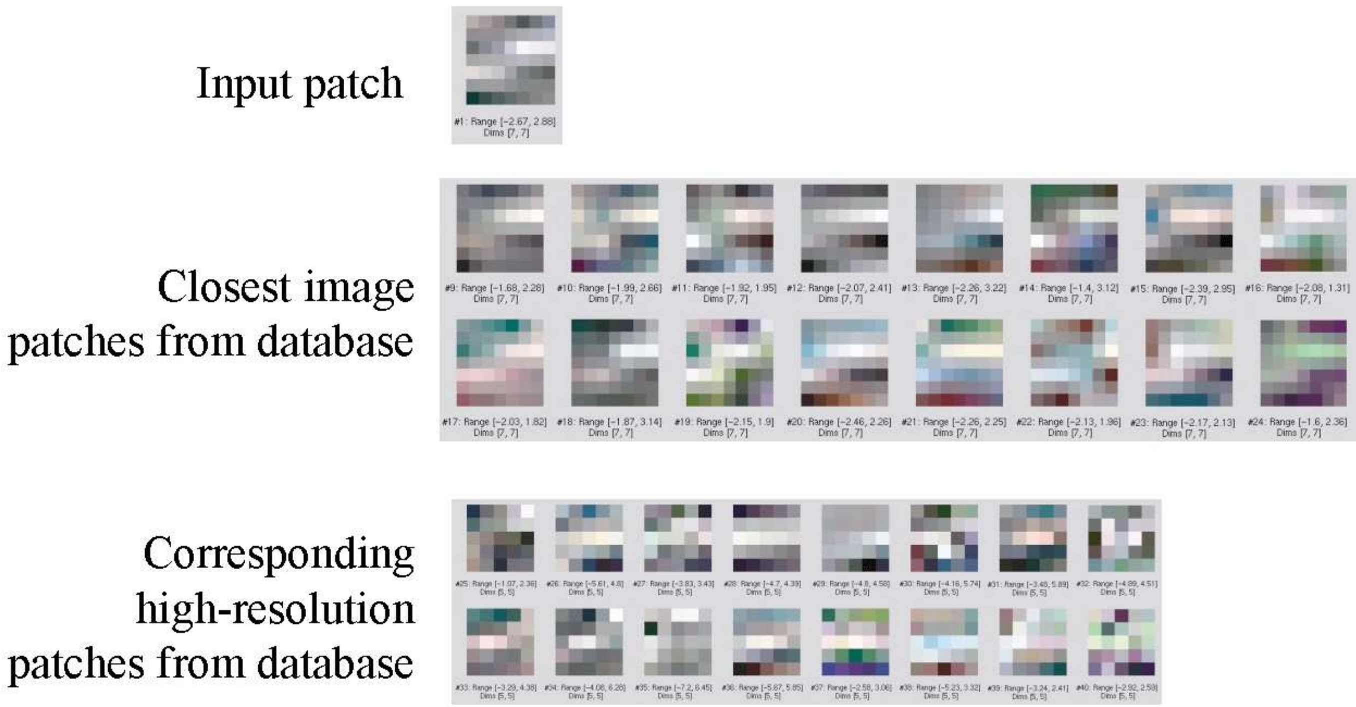
Reconstruction

- For a new low-res image:
  - Interpolate
  - Match closest low-res patch in database
  - Update with high-res counterpart
- Won’t work!



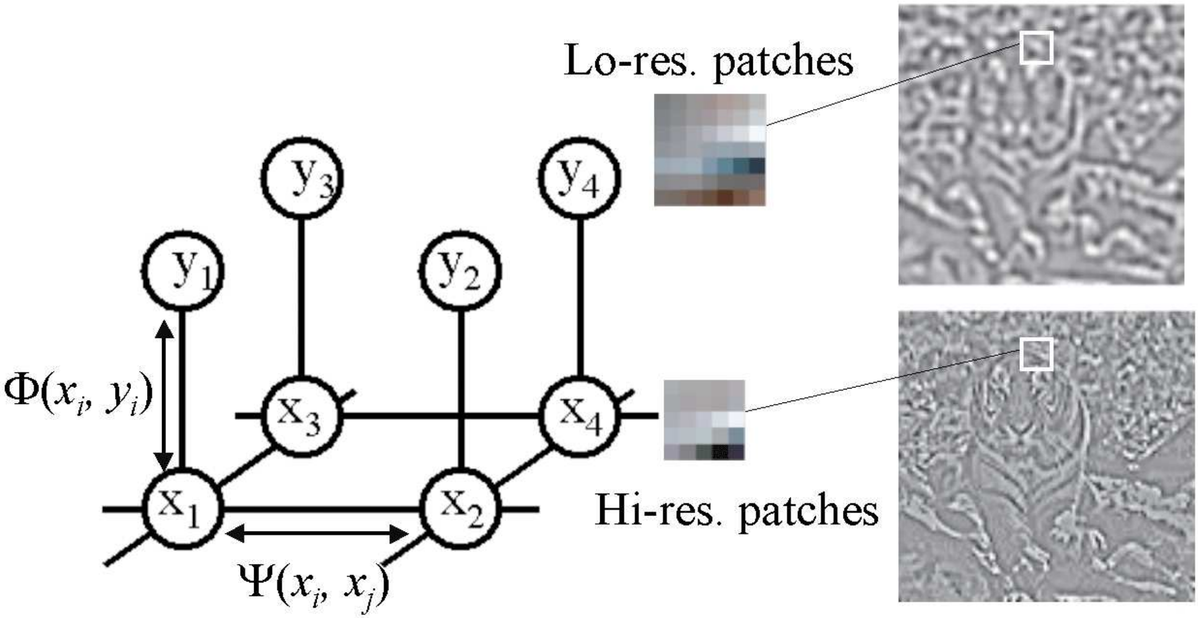
Reconstruction

- High-res patches can have large variations even when low-res ones are similar



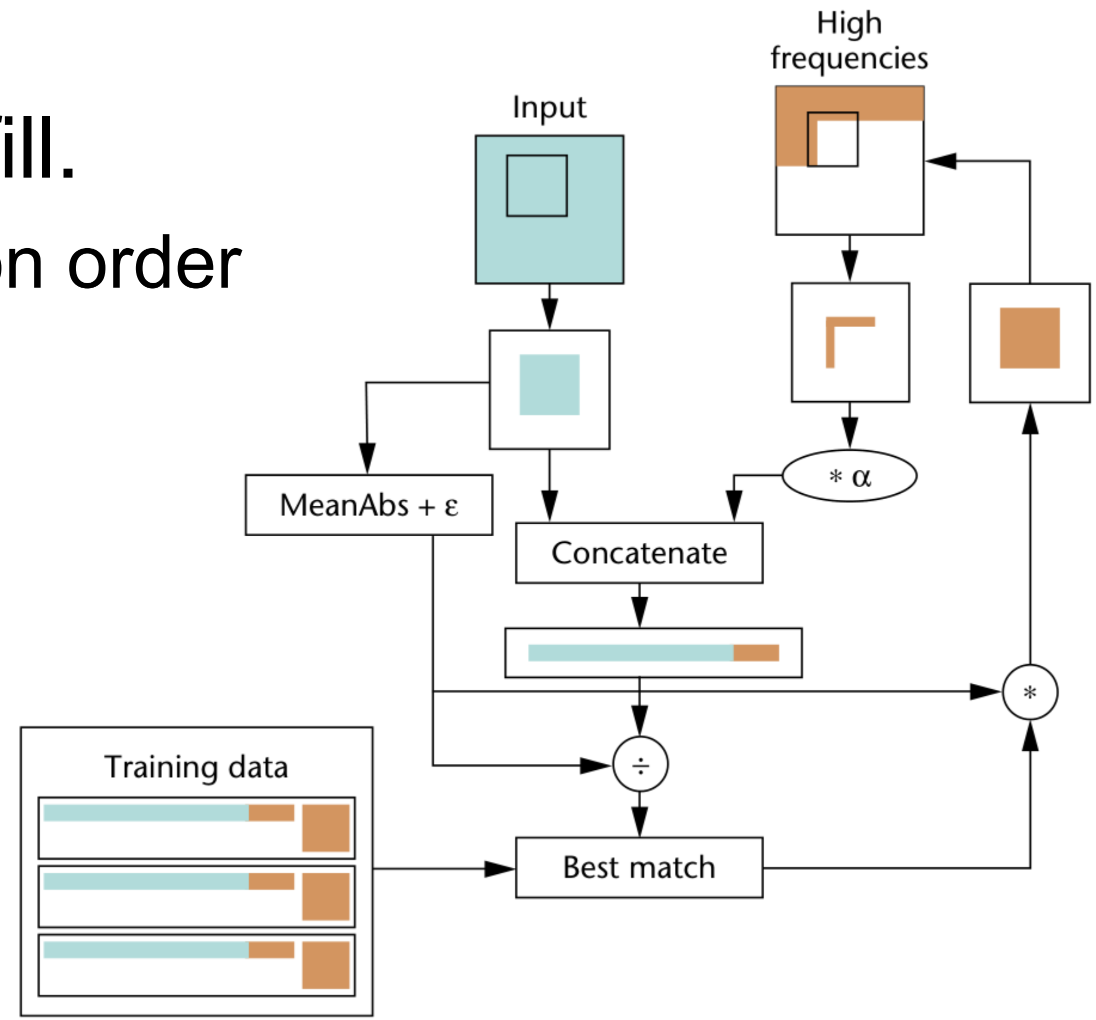
Reconstruction

- Connect neighboring patches smoothly
  - Markov random field
    - Can solve with BP

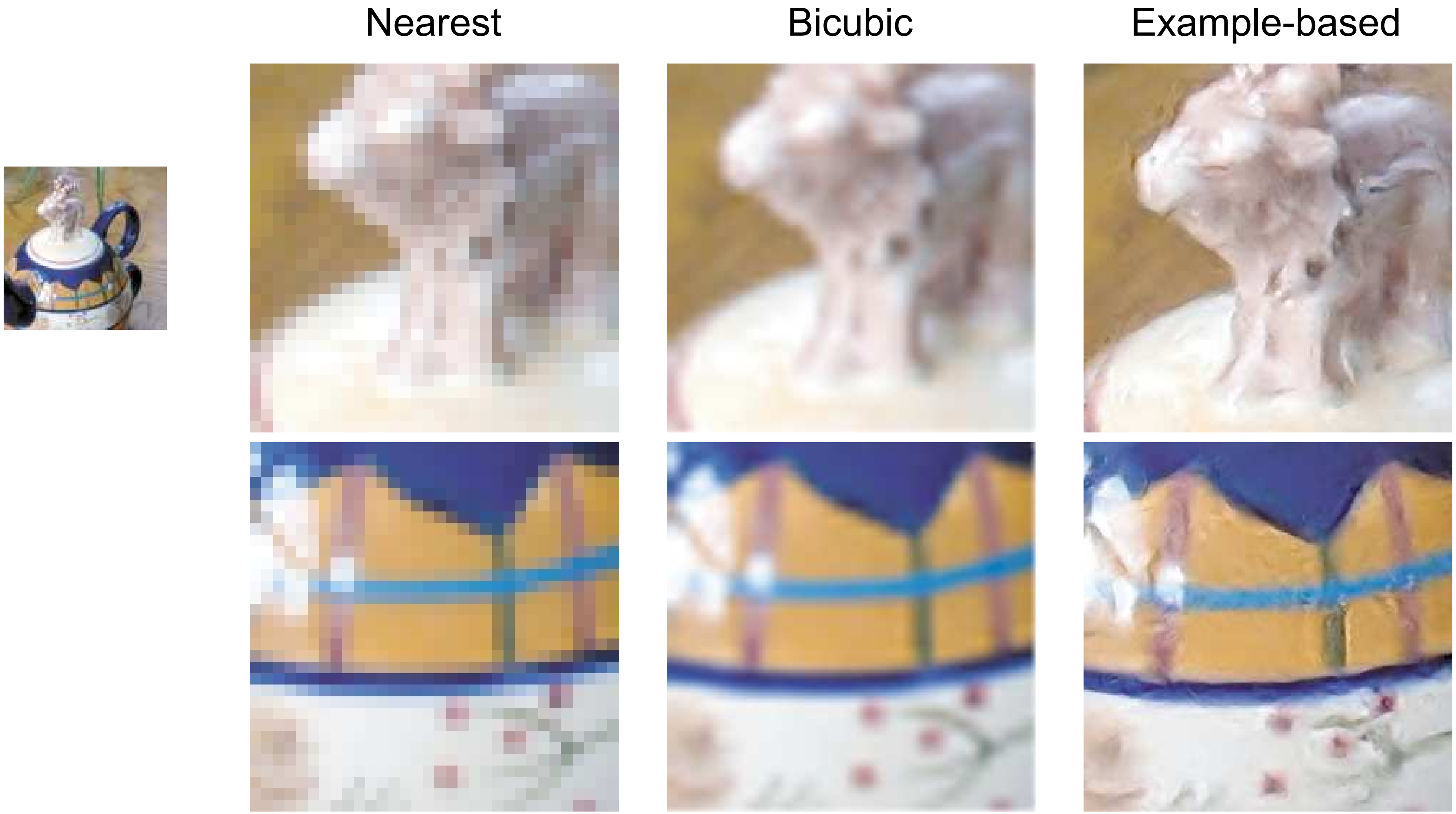


One-pass algorithm

- No BP, just fill.
  - Rasterization order



Results

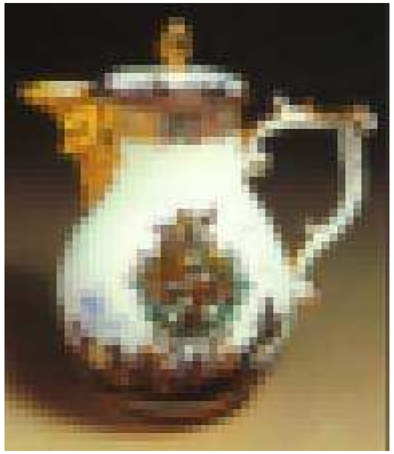


Bad training

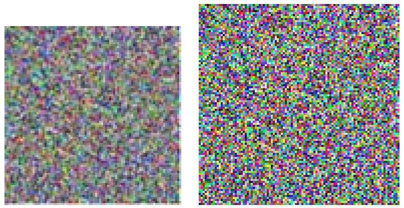
- What if the training image contains no/bad information?

Bad training

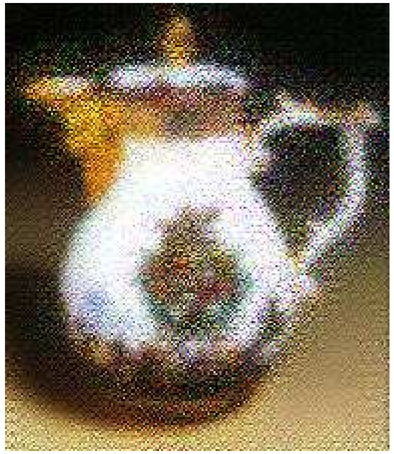
- Wrong information will be introduced



original



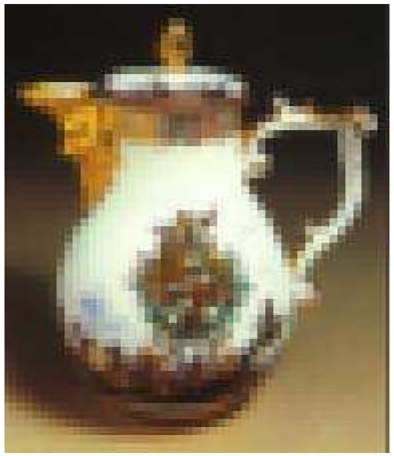
training



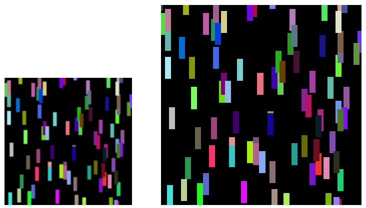
result

Bad training

- Wrong information will be introduced



original

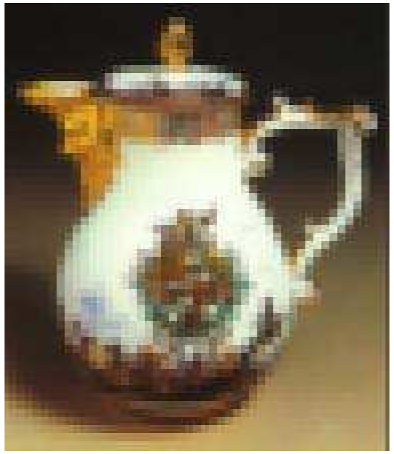


training

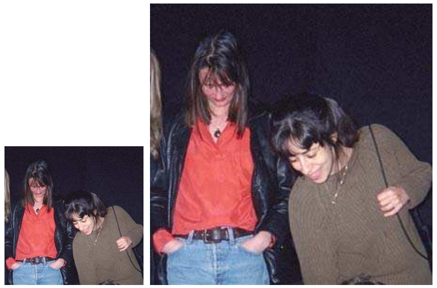
result

Bad training

- Natural image contains reasonable edges



original

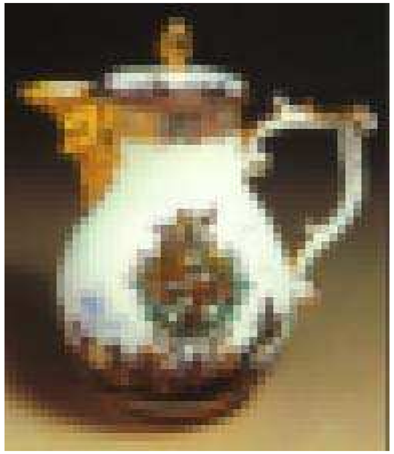


training

result

Bad training

- Natural image contains reasonable edges



original

ground truth

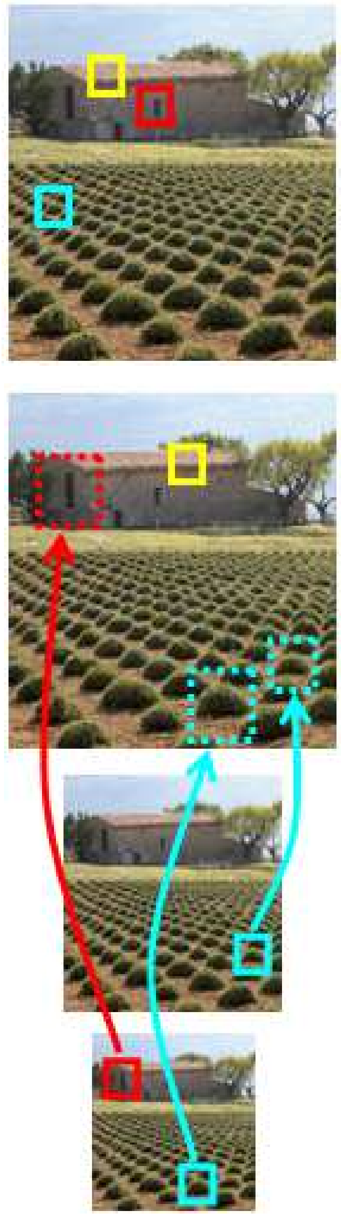
Super-resolution

- Multi-image SR
  - Many images of same scene
- Example-based SR
  - Image database from other scene
- Can we SR with exactly one image?

Daniel Glasner, Shai Bagon, and Michal Irani

SUPER-RESOLUTION FROM A SINGLE IMAGE

- Key observation:
  - “Small patches in a single natural image tend to recur many times within and across scales of the same image.”

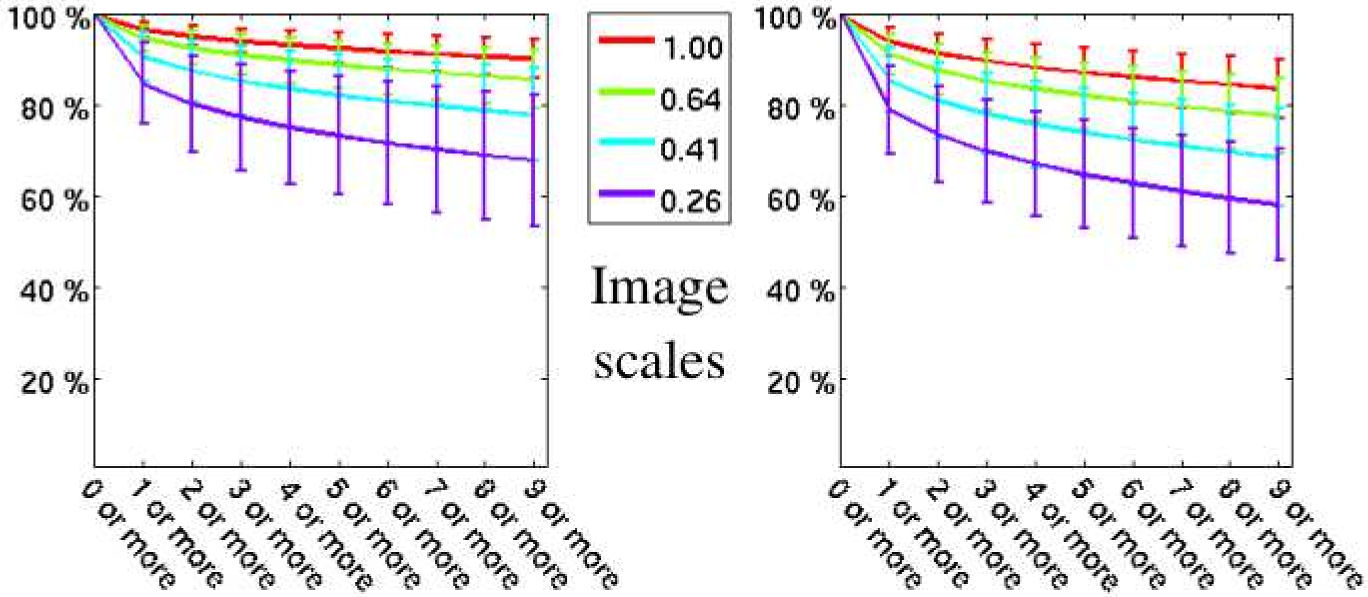


Justification

- Average patch recurrence within and across scales of a single image
  - find how many similar patches for each 5x5 patch

Justification

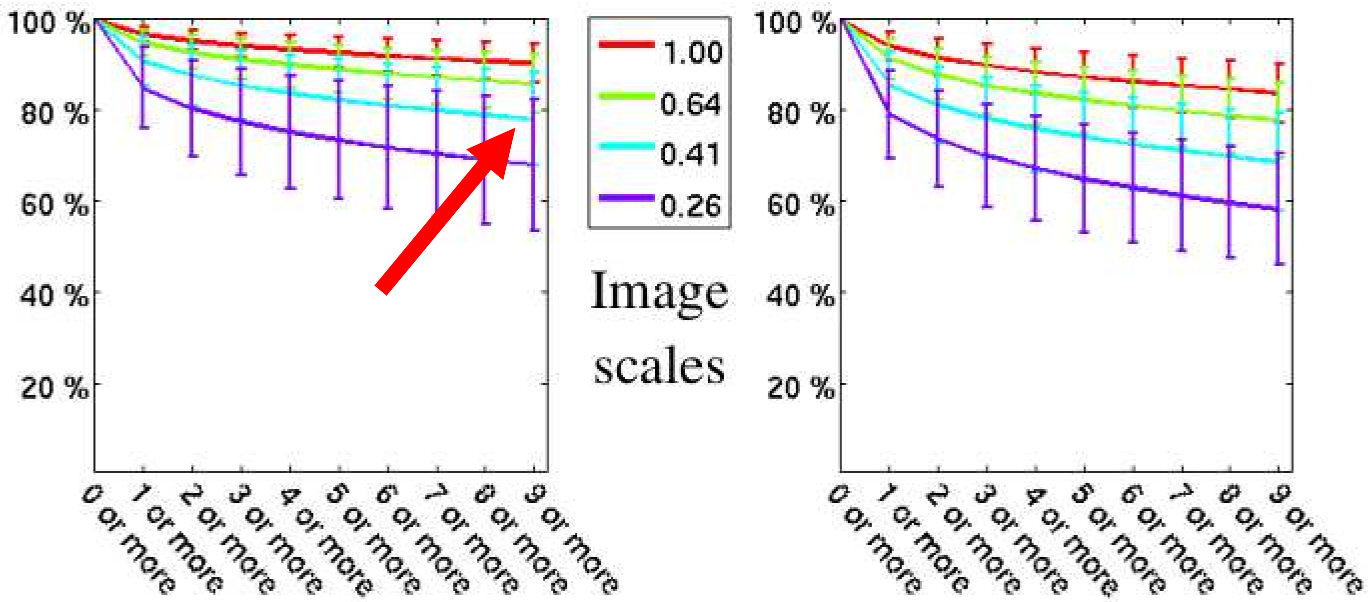
- Average patch recurrence within and across scales of a single image



(a) All image patches (b) High variance patches only

Justification

- > 90% have 9 or more at original scale
- > 80% have 9 or more at 0.41 scale



(a) All image patches (b) High variance patches only

Within scale pairs

- Multiple low-res patch of “same” high-res patch
  - Multiple-image SR (Classical SR)

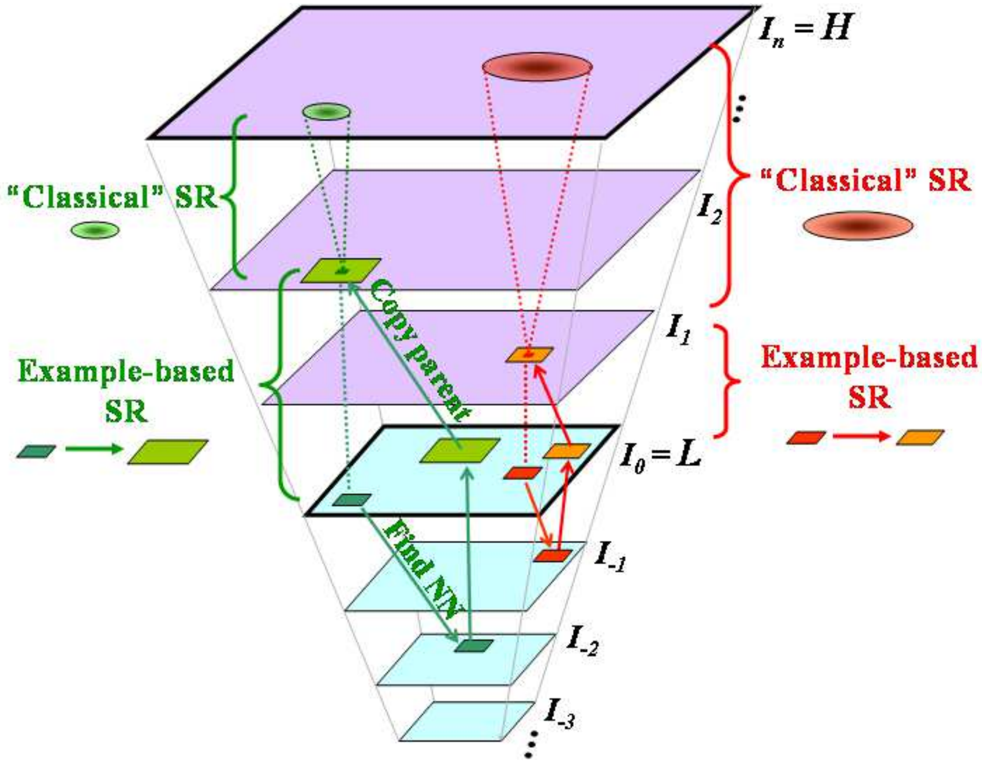
Across scale pairs

- We have low/high res correspondence
  - Example-based SR

Put together

- Within scale
  - Multiple-Image SR
- Across scale
  - Example-Based SR

- Unified SR



Results

Bicubic

Results

Single-Image SR

Bicubic

Single-Image SR


ZSHC
HSKRN
CHKRVD
HONSDCV
OKHDNRCS

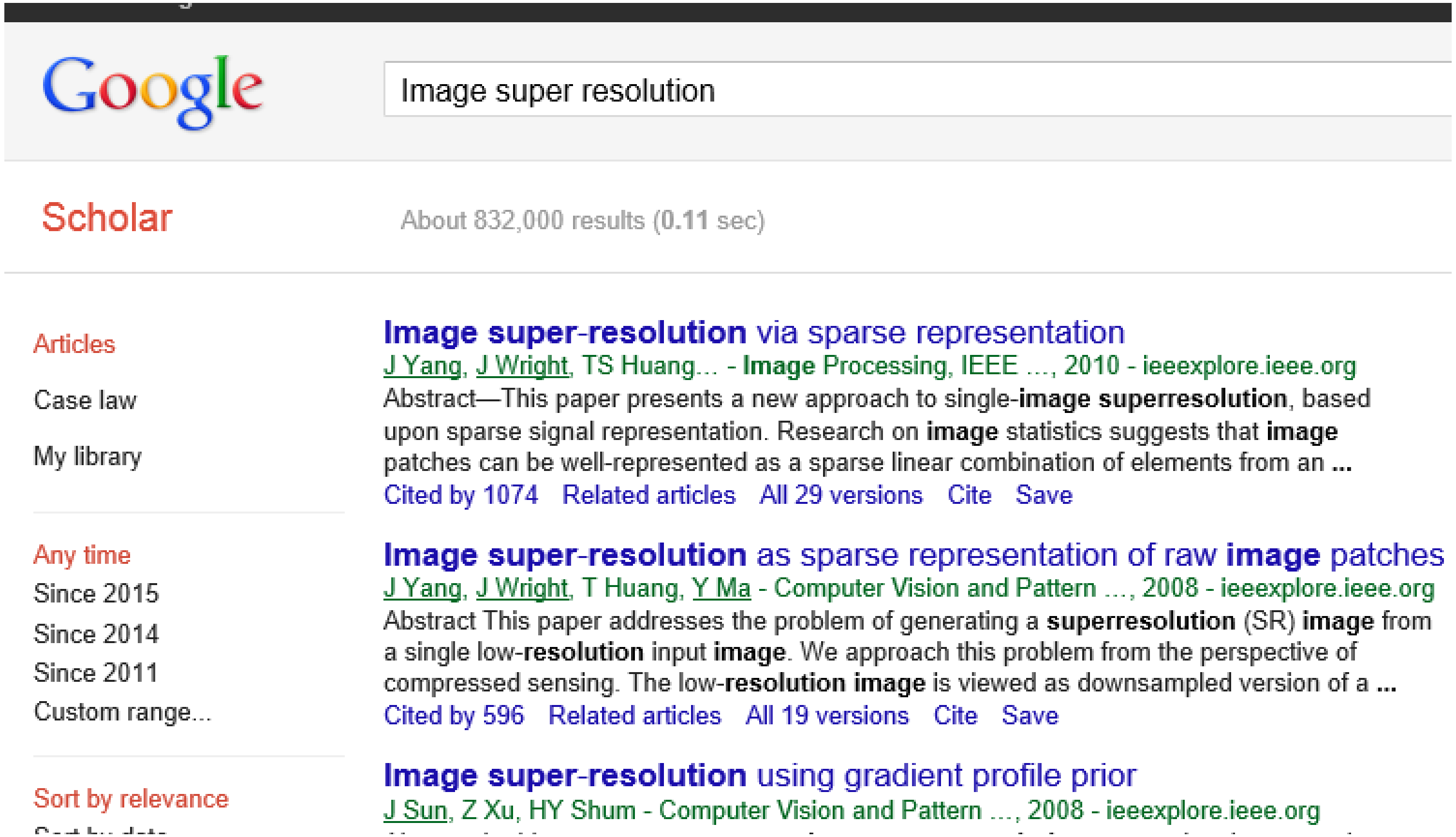

ZSHC
HSKRN
CHKRVD
HONSDCV
OKHDNRCS
VHDNKUOSRC
BDCLKZVHSROA

Bicubic

Single-Image SR

Super-resolution

- A very active area



- Aspect ratio
  - SR keeps it unchanged
  - What if image is too high or too wide?

- Aspect ratio
  - change it can be difficult

Scale

Crop right

Crop left

Shai Avidan and Ariel Shamir

SEAM CARVING FOR CONTENT-AWARE IMAGE RESIZING

Content-awareness

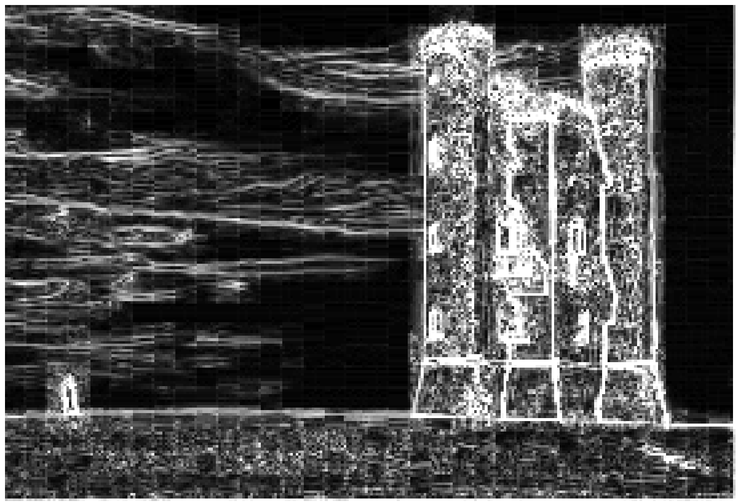
- Remove less salient area
- Assign each pixel a cost
  - Important pixel costs more if removed

Content-awareness

- Edges might be important
  - Edge energy:

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

- Important objects
  - Saliency energy
- …



Resizing

- Reduce width by n
  - Remove n pixel in each row

Resizing

- Reduce width by n
  - Remove n pixel in each row
  - The pixel with smallest cost?
    - Zig-zagging

Resizing

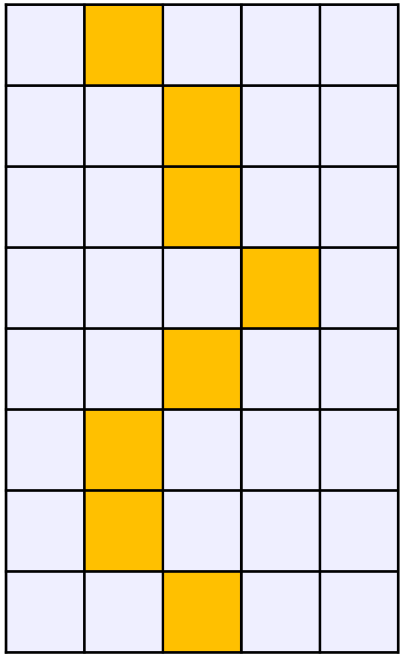
- Reduce width by n
  - Remove n pixel in each row
  - The columns of smallest total cost?
    - Still not good…

Resizing

- Reduce width by n
  - Remove n pixel in each row
  - The vertical seam of smallest total cost

Seam

- Vertical seam:
  - vertically or diagonally connected path



Seam-carving

- Reduce width by n
  - Iteratively remove n vertical beams with smallest cost
- How to find the minimum beam?

Seam-carving

- Reduce width by n
  - Iteratively remove n
    vertical beams with smallest cost

- How to find the minimal beam?
  - Dynamic-Programming

Seam-carving

- Init with pixel cost

1	3	4	1	2
2	1	3	4	1
1	3	2	1	2
4	1	2	4	5
1	4	1	2	4
2	2	2	3	1
2	3	5	1	2
4	1	4	3	1

Seam-carving

- Init with pixel cost
- Filling minimum cost from top to bottom

1 1	3 3	4 4	1 1	2 2
2	1	3	4	1
1	3	2	1	2
4	1	2	4	5
1	4	1	2	4
2	2	2	3	1
2	3	5	1	2
4	1	4	3	1

Seam-carving

- Init with pixel cost
- Filling minimum cost from top to bottom

1 1	3 3	4 4	1 1	2 2
2 3	1	3	4	1
1	3	2	1	2
4	1	2	4	5
1	4	1	2	4
2	2	2	3	1
2	3	5	1	2
4	1	4	3	1

Seam-carving

- Init with pixel cost
- Filling minimum cost from top to bottom

1 1	3 3	4 4	1 1	2 2
2 3	1 2	3	4	1
1	3	2	1	2
4	1	2	4	5
1	4	1	2	4
2	2	2	3	1
2	3	5	1	2
4	1	4	3	1

Seam-carving

- Init with pixel cost
- Filling minimum cost from top to bottom

1 1	3 3	4 4	1 1	2 2
2 3	1 2	3 4	4	1
1	3	2	1	2
4	1	2	4	5
1	4	1	2	4
2	2	2	3	1
2	3	5	1	2
4	1	4	3	1

Seam-carving

- Init with pixel cost
- Filling minimum cost from top to bottom

1 1	3 3	4 4	1 1	2 2
2 3	1 2	3 4	4 5	1 2
1	3	2	1	2
4	1	2	4	5
1	4	1	2	4
2	2	2	3	1
2	3	5	1	2
4	1	4	3	1

Seam-carving

- Init with pixel cost
- Filling minimum cost from top to bottom

1 1	3 3	4 4	1 1	2 2
2 3	1 2	3 4	4 5	1 2
1 3	3 5	2 4	1 3	2 4
4	1	2	4	5
1	4	1	2	4
2	2	2	3	1
2	3	5	1	2
4	1	4	3	1

Seam-carving

- Init with pixel cost
- Filling minimum cost from top to bottom

1 1	3 3	4 4	1 1	2 2
2 3	1 2	3 4	4 5	1 2
1 3	3 5	2 4	1 3	2 4
4 7	1 4	2 5	4 7	5 8
1 5	4 8	1 5	2 7	4 11
2 7	2 7	2 7	3 8	1 8
2 9	3 10	5 12	1 8	2 10
4 13	1 10	4 12	3 11	1 9

Seam-carving

- Init with pixel cost
- Filling minimum cost from top to bottom
- The minimal beam ends at the minimal total cost

1 1	3 3	4 4	1 1	2 2
2 3	1 2	3 4	4 5	1 2
1 3	3 5	2 4	1 3	2 4
4 7	1 4	2 5	4 7	5 8
1 5	4 8	1 5	2 7	4 11
2 7	2 7	2 7	3 8	1 8
2 9	3 10	5 12	1 8	2 10
4 13	1 10	4 12	3 11	1 9

Seam-carving

- Init with pixel cost
- Filling minimum cost from top to bottom
- The minimal beam ends at the minimal total cost
- Back-trace for min path

1 1	3 3	4 4	1 1	2 2
2 3	1 2	3 4	4 5	1 2
1 3	3 5	2 4	1 3	2 4
4 7	1 4	2 5	4 7	5 8
1 5	4 8	1 5	2 7	4 11
2 7	2 7	2 7	3 8	1 8
2 9	3 10	5 12	1 8	2 10
4 13	1 10	4 12	3 11	1 9

Seam-carving

- Init with pixel cost
- Filling minimum cost from top to bottom
- The minimal beam ends at the minimal total cost
- Back-trace for min path

1 1	3 3	4 4	1 1	2 2
2 3	1 2	3 4	4 5	1 2
1 3	3 5	2 4	1 3	2 4
4 7	1 4	2 5	4 7	5 8
1 5	4 8	1 5	2 7	4 11
2 7	2 7	2 7	3 8	1 8
2 9	3 10	5 12	1 8	2 10
4 13	1 10	4 12	3 11	1 9

Seam-carving

- Init with pixel cost
- Filling minimum cost from top to bottom
- The minimal beam ends at the minimal total cost
- Back-trace for min path

1 1	3 3	4 4	1 1	2 2
2 3	1 2	3 4	4 5	1 2
1 3	3 5	2 4	1 3	2 4
4 7	1 4	2 5	4 7	5 8
1 5	4 8	1 5	2 7	4 11
2 7	2 7	2 7	3 8	1 8
2 9	3 10	5 12	1 8	2 10
4 13	1 10	4 12	3 11	1 9

Seam-carving

- Init with pixel cost
- Filling minimum cost from top to bottom
- The minimal beam ends at the minimal total cost
- Back-trace for min path

1 1	3 3	4 4	1 1	2 2
2 3	**1 2**	3 4	4 5	1 2
1 3	3 5	2 4	1 3	2 4
4 7	**1 4**	2 5	4 7	5 8
1 5	4 8	**1 5**	2 7	4 11
2 7	2 7	**2 7**	3 8	1 8
2 9	3 10	5 12	**1 8**	2 10
4 13	1 10	4 12	3 11	**1 9**

Seam-carving

- … then remove this seam
- Repeat until n seams is removed

Discussion

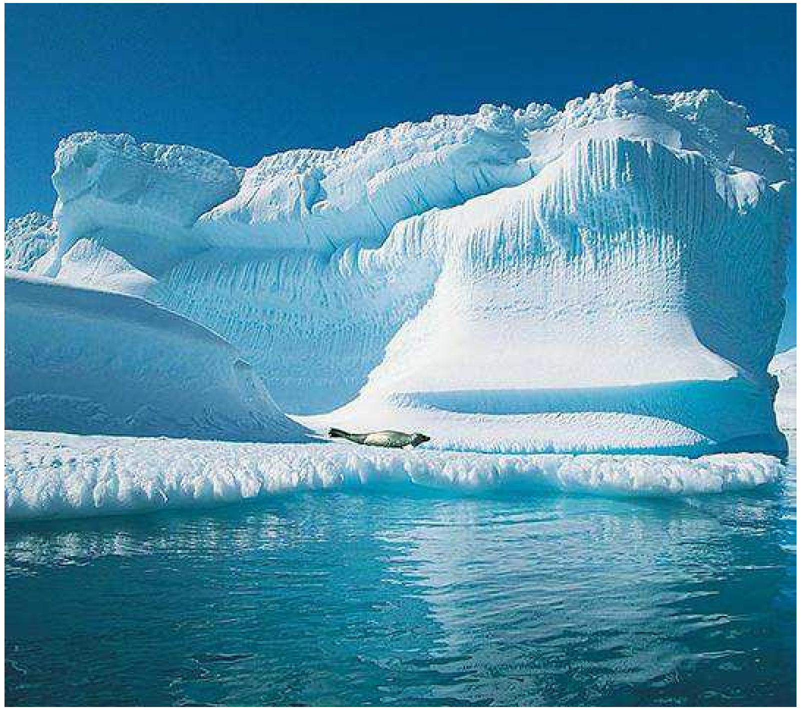
- To widen image
  - Insert beams (how?)
- Other costs
  - Protect region/unwanted region
- …

Results

Seam-Carving Scale Crop

Results

- Shrinking



narrowed

Results

- Widening

expanded

Results

- Object removal

narrowing and removing

More resizing

- Many factors are important in image
  - Edges

More resizing

- Many factors are important in image
  - Edges
  - Structure

More resizing

- Many factors are important in image
  - Edges
  - Structure
  - Symmetry

More resizing

- Many factors are important in image
  - Edges
  - Structure
  - Symmetry

- More considerate resizing methods…
  - Try combine seam-carving, cropping, scaling ?
    - M. Rubinstein, A. Shamir, S. Avidan, Multi-operator Media Retargeting, ACM ToG, 2009.