Lecture 24:

Image Processing

Computer Graphics and Imaging
UC Berkeley CS184/284A

Credit: Kayvon Fatahalian created the majority of these lecture slides

Case Study: JPEG Compression

JPEG Compression: The Big Ideas

Low-frequency content is predominant in images of the real world

The human visual system is:

- Less sensitive to detail in chromaticity than in luminance
- Less sensitive to high frequency sources of error

Therefore, image compression of natural images can:

- Reduce perceived error by localizing error into high frequencies, and in chromaticity

Slide credit: Pat Hanrahan

Y'CbCr Color Space

Y'CbCr color space

- This is a perceptually-motivated color space akin to L*a*b* that we discussed in the color lecture
- Y' is luma (lightness), Cb and Cr are chroma channels (blue-yellow and red-green difference from gray)

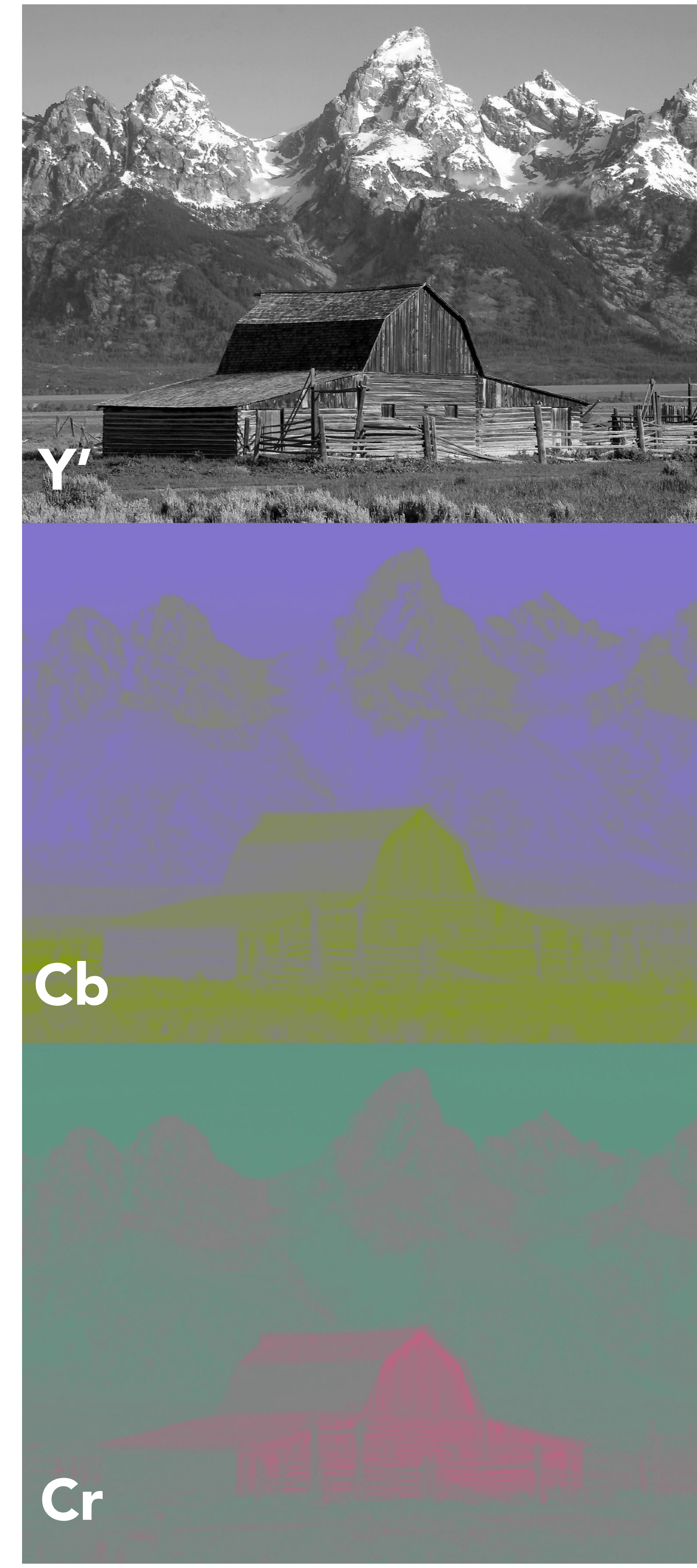


Image credit: Wikipedia

*Omitting discussion of nonlinear gamma encoding in Y' channel

Example Image

Original picture

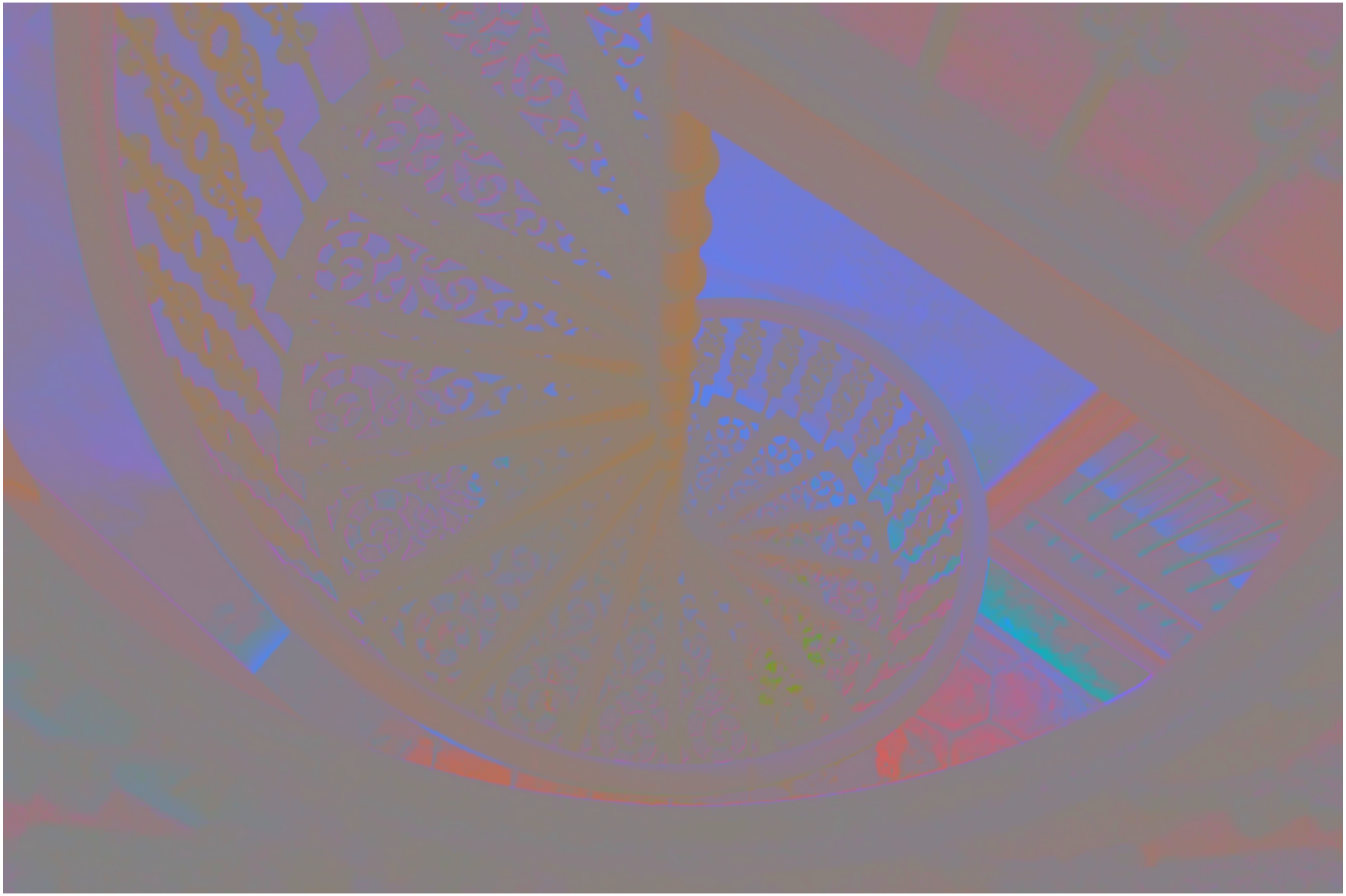
Y' Only (Luma)

Luma channel

Downsampled Y’

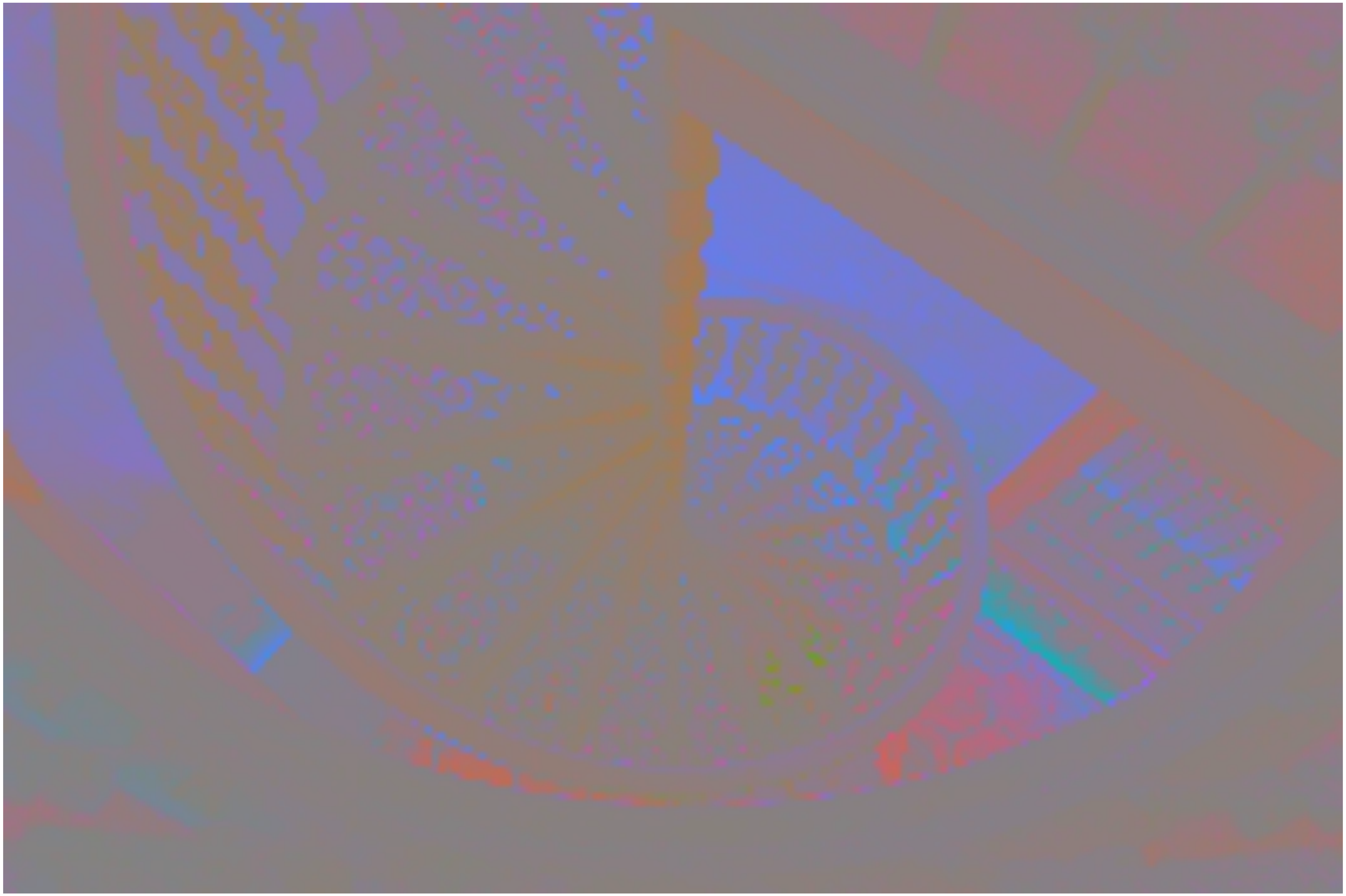
4x4 downsampled luma channel

CbCr Only (Chroma)



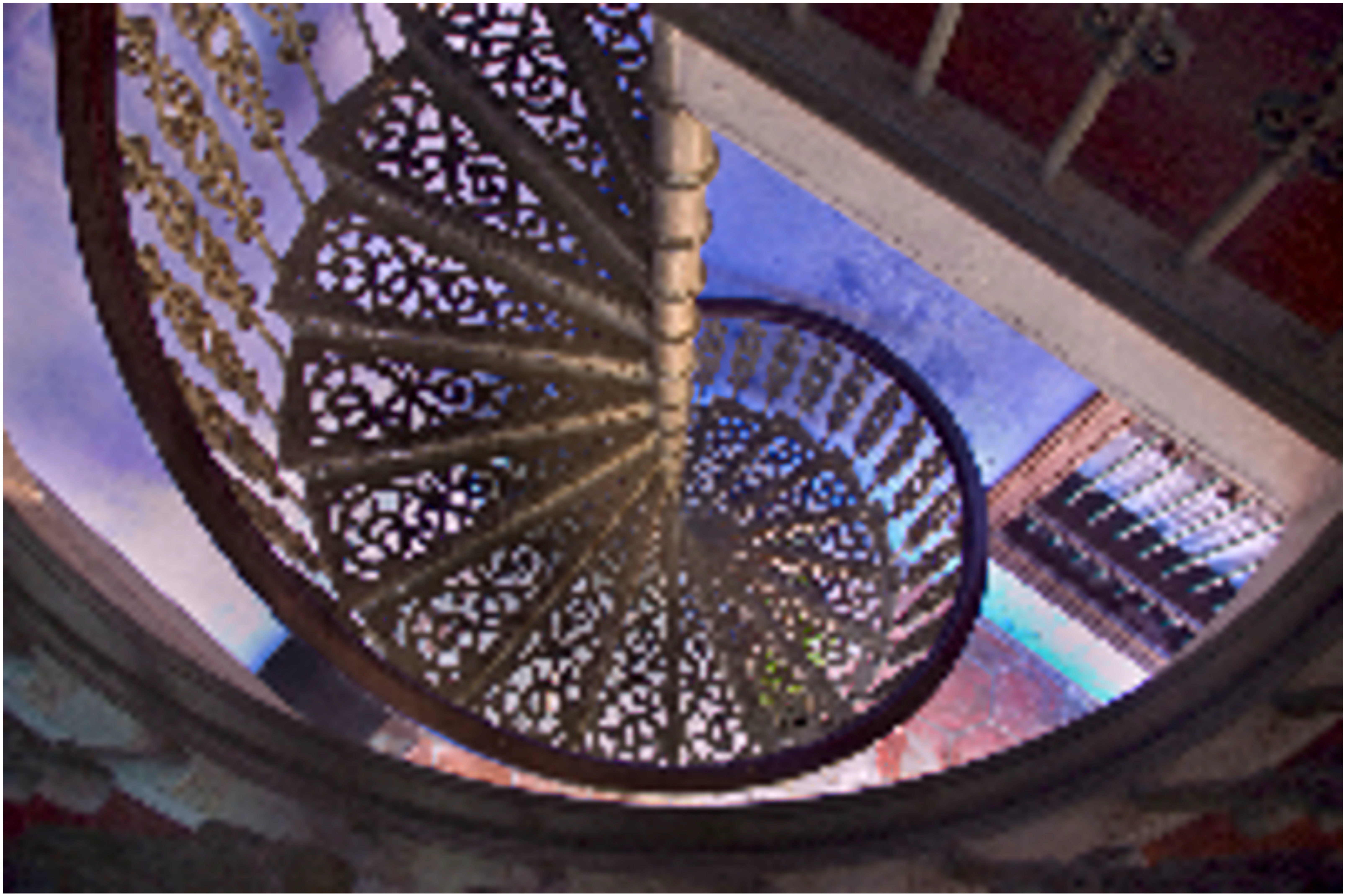
CbCr channels

Downsampled CbCr



4x4 downsampled CbCr channels

Example: Compression in Y' Channel



4x4 downsampled Y', full-resolution CbCr

Example: Compression in CbCr Channels

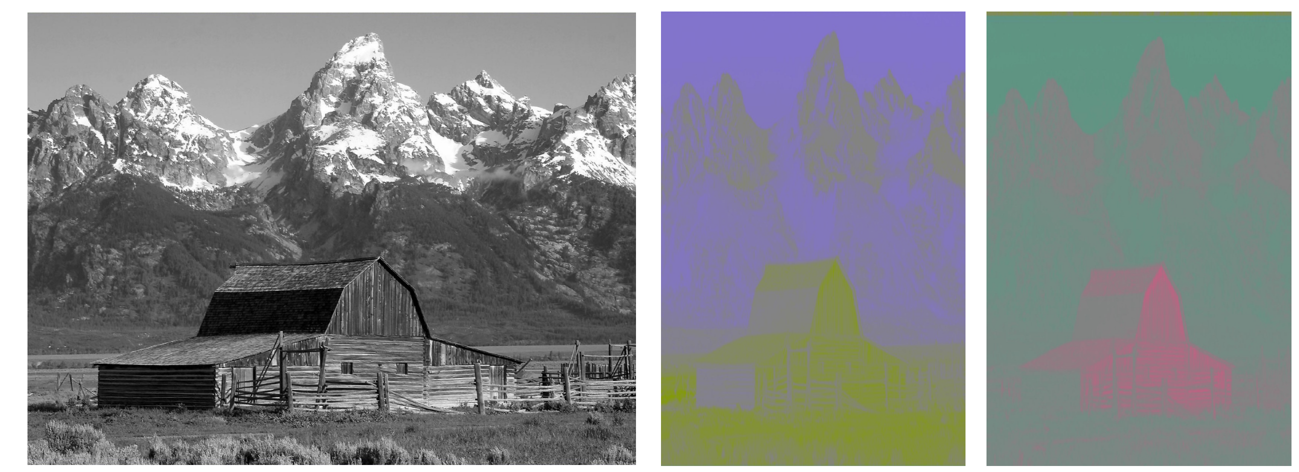
Full-resolution Y', 4x4 down sampled CbCr

Original Image

JPEG: Chroma Subsampling in Y'CbCr Space

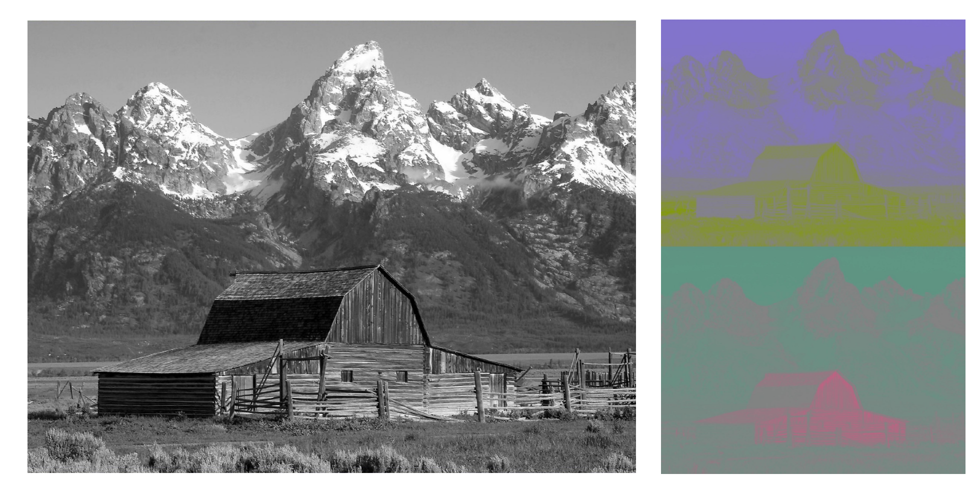
Subsample chroma channels
(e.g. to 4:2:2 or 4:2:0 format)

4:2:2 representation: (retain 2/3 values)



- Store Y' at full resolution
- Store Cb, Cr at half resolution in horizontal dimension

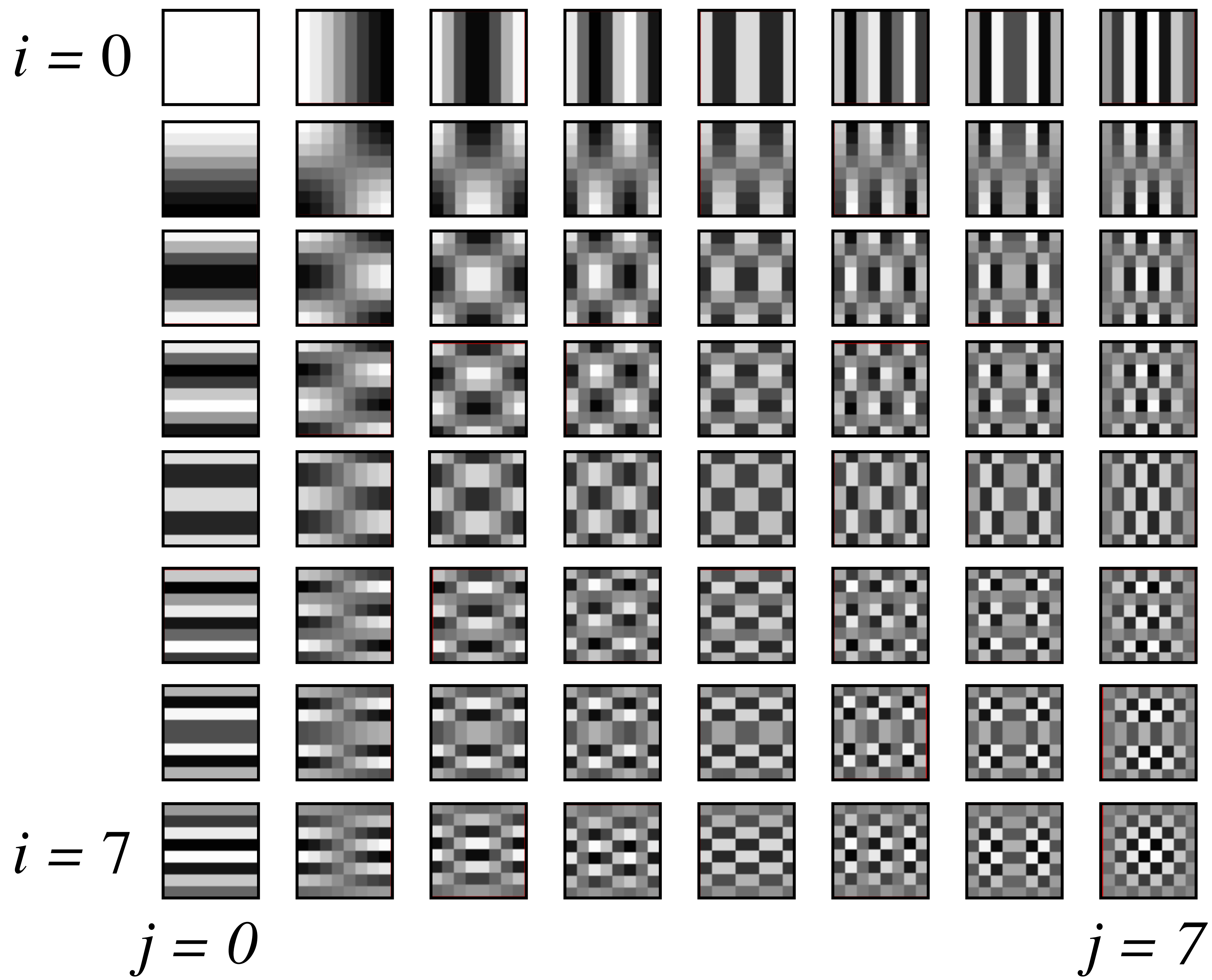
4:2:0 representation: (retain 1/2 values)



- Store Y' at full resolution
- Store Cb, Cr at half resolution in both dimensions

JPEG: Discrete Cosine Transform (DCT)

$$\text{basis}[i,j] = \cos\left[\pi \frac{i}{N}\left(x+\frac{1}{2}\right)\right] \times \underset{j}{\cos}\left[\pi \frac{j}{N}\left(y+\frac{1}{2}\right)\right]$$



In JPEG, Apply discrete cosine transform (DCT) to each 8x8 block of image values

DCT computes projection of image onto 64 basis functions:

basis[i, j]

DCT applied to 8x8 pixel blocks of Y' channel, 16x16 pixel blocks of Cb, Cr (assuming 4:2:0)

JPEG Quantization: Prioritize Low Frequencies

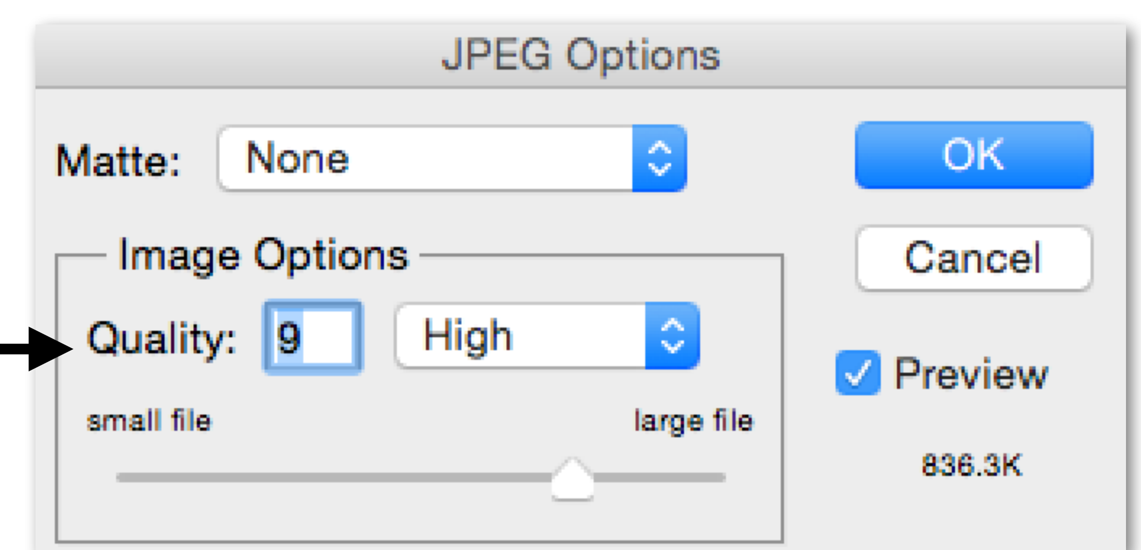
$$\begin{bmatrix} -415 & -30 & -61 & 27 & 56 & -20 & -2 & 0 \\ 4 & -22 & -61 & 10 & 13 & -7 & -9 & 5 \\ -47 & 7 & 77 & -25 & -29 & 10 & 5 & -6 \\ -49 & 12 & 34 & -15 & -10 & 6 & 2 & 2 \\ 12 & -7 & -13 & -4 & -2 & 2 & -3 & 3 \\ -8 & 3 & 2 & -6 & -2 & 1 & 4 & 2 \\ -1 & 0 & 0 & -2 & -1 & -3 & 4 & -1 \\ 0 & 0 & -1 & -4 & -1 & 0 & 1 & 2 \end{bmatrix} \Big/ \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}$$

Result of DCT
(image encoded in cosine basis)

Quantization Matrix

$$= \begin{bmatrix} -26 & -3 & -6 & 2 & 2 & -1 & 0 & 0 \\ 0 & -2 & -4 & 1 & 1 & 0 & 0 & 0 \\ -3 & 1 & 5 & -1 & -1 & 0 & 0 & 0 \\ -4 & 1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Changing JPEG quality setting in your favorite photo app modifies this matrix ("lower quality" = higher values for elements in quantization matrix)



Quantization produces small values for coefficients (only a few bits needed per coefficient)

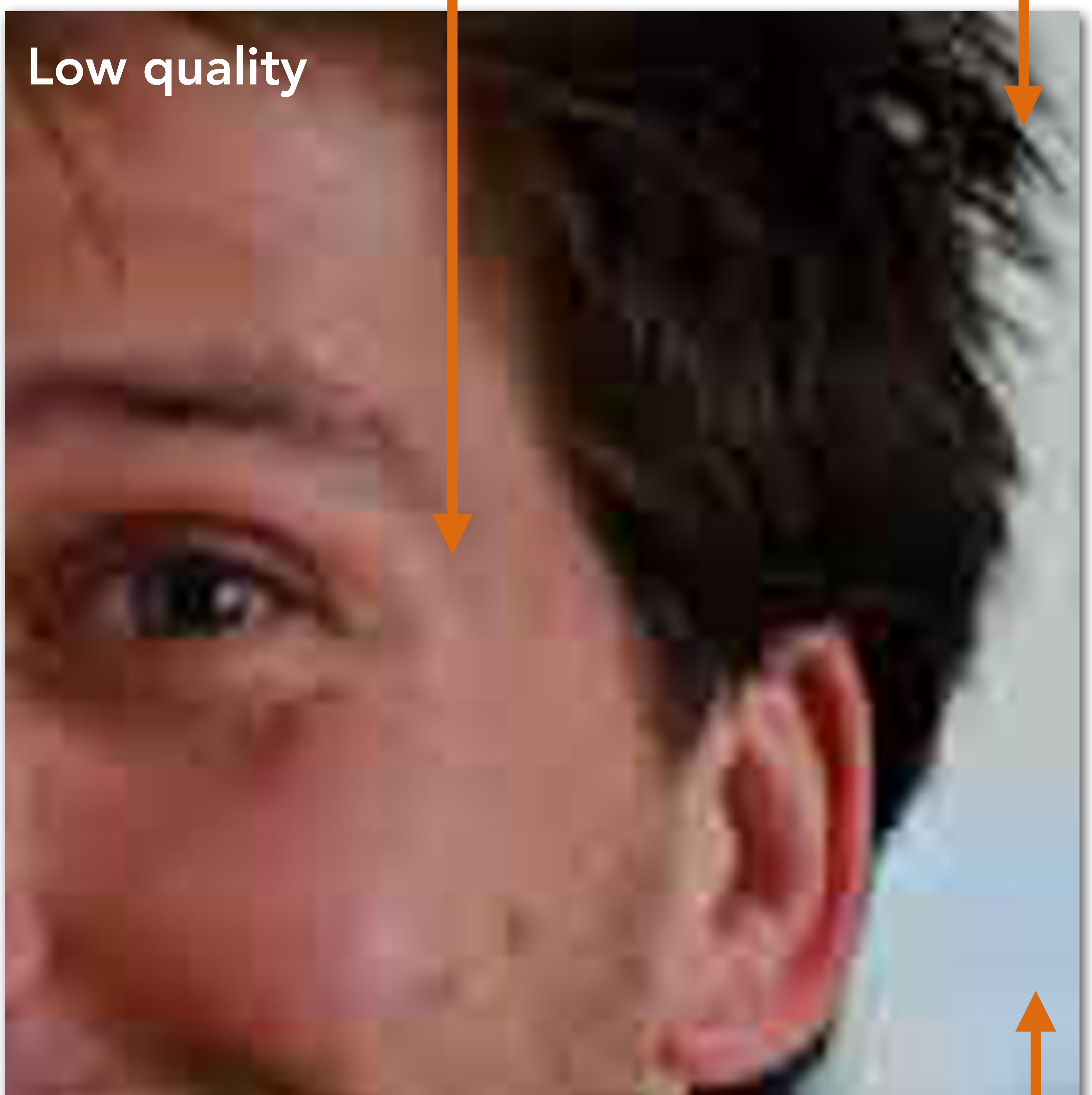
Observe: quantization zeros out many coefficients

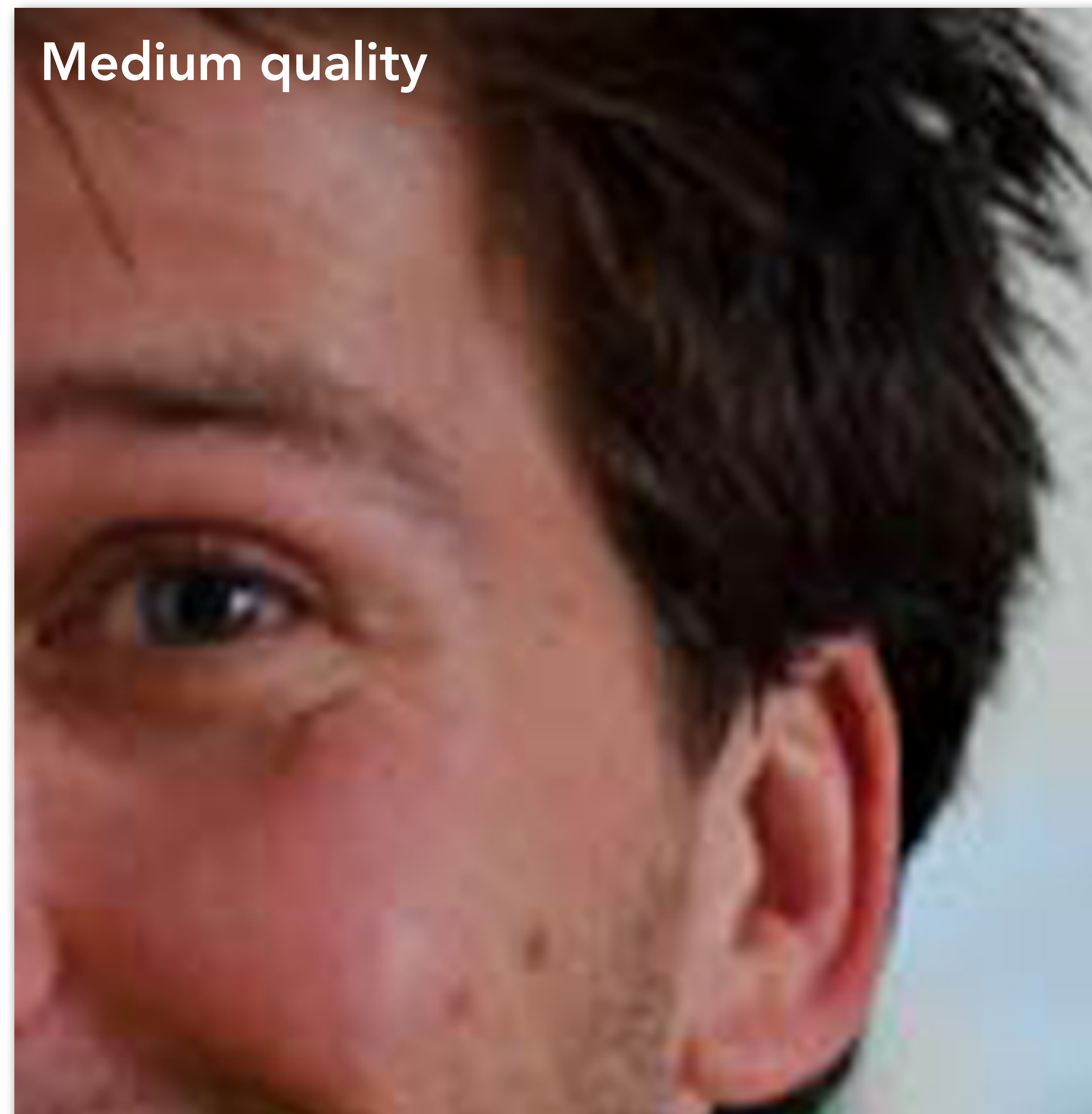
Slide credit: Wikipedia, Pat Hanrahan

JPEG: Compression Artifacts

Noticeable 8x8 pixel block boundaries

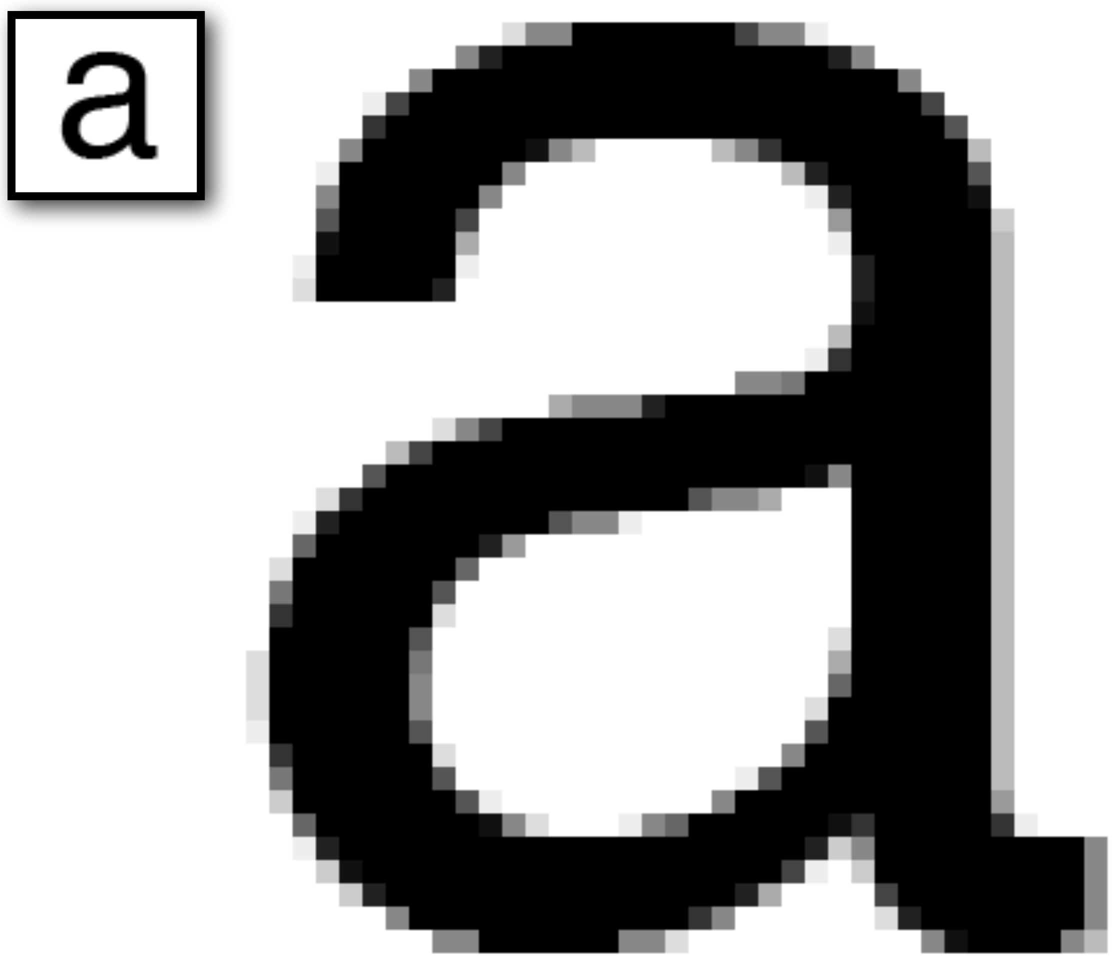
Noticeable error near large color gradients





Low-frequency regions of image represented accurately even under high compression

JPEG: Compression Artifacts



Original

Quality Level

Quality Level

Quality Level

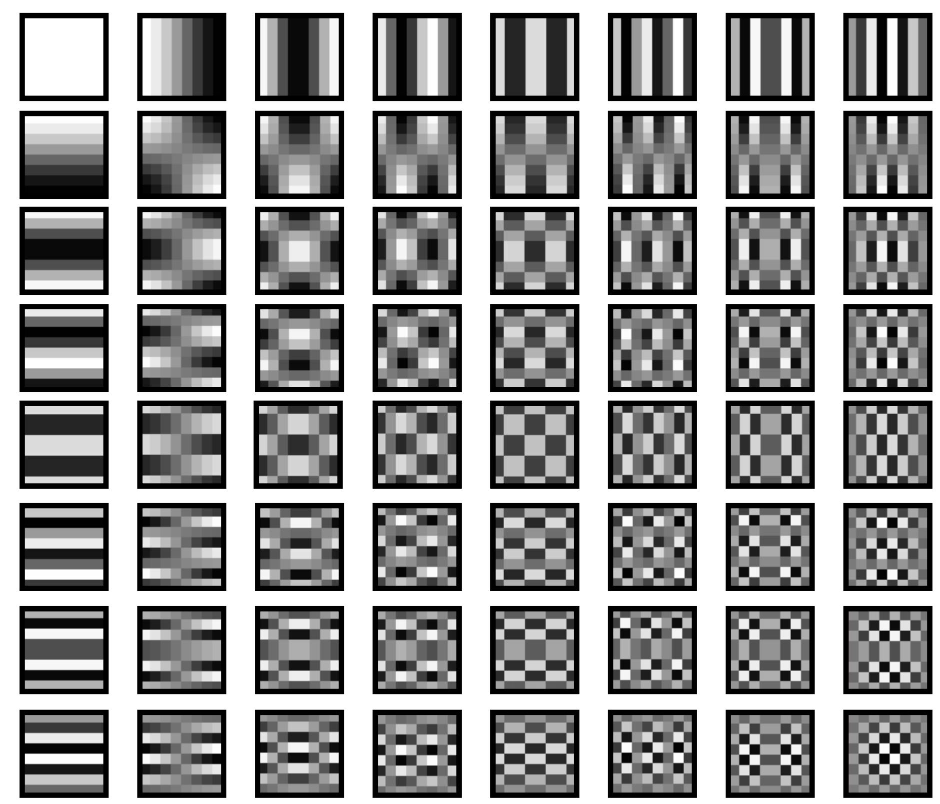
Quality Level

Why might JPEG compression not be a good compression scheme for line-based illustrations or rasterized text?

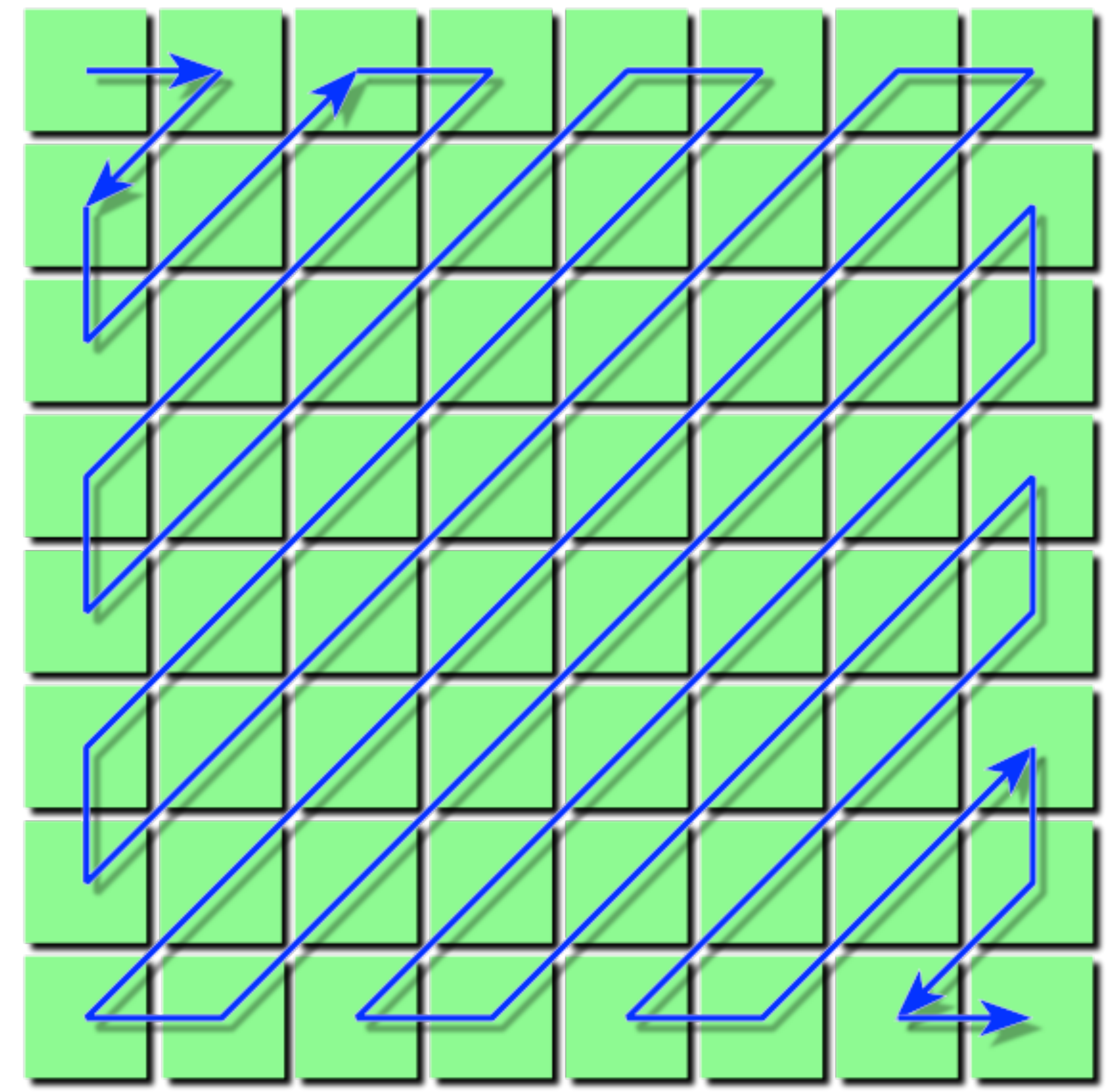
Lossless Compression of Quantized DCT Values

$$\begin{bmatrix} -26 & -3 & -6 & 2 & 2 & -1 & 0 & 0 \\ 0 & -2 & -4 & 1 & 1 & 0 & 0 & 0 \\ -3 & 1 & 5 & -1 & -1 & 0 & 0 & 0 \\ -4 & 1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Quantized DCT Values



Basis functions



Reordering

Entropy encoding: (lossless)

- Reorder values
- Run-length encode (RLE) 0’s
- Huffman encode non-zero values

Image credit: Wikipedia

JPEG Compression Summary

Convert image to Y'CbCr color space

Downsample CbCr (to 4:2:2 or 4:2:0) (information loss occurs here)

For each color channel (Y', Cb, Cr):

- For each 8x8 block of values
  - Compute DCT
  - Quantize results (information loss occurs here)
  - Reorder values
  - Run-length encode 0-spans
  - Huffman encode non-zero values

Theme: Exploit Perception in Visual Computing

JPEG is an example of a general theme of exploiting characteristics of human perception to build efficient visual computing systems

We are perceptually insensitive to color errors:

- Separate luminance from chrominance in color representations (e.g, Y'CbCr) and compress chrominance

We are less perceptually sensitive to high-frequency error

- Use a frequency-based encoding (cosine transform) and compress high-frequency values

We perceive lightness non-linearly (not discussed in this lecture)

- Encode pixel values non-linearly to match perceived brightness using gamma curve

Basic Image Processing Operations

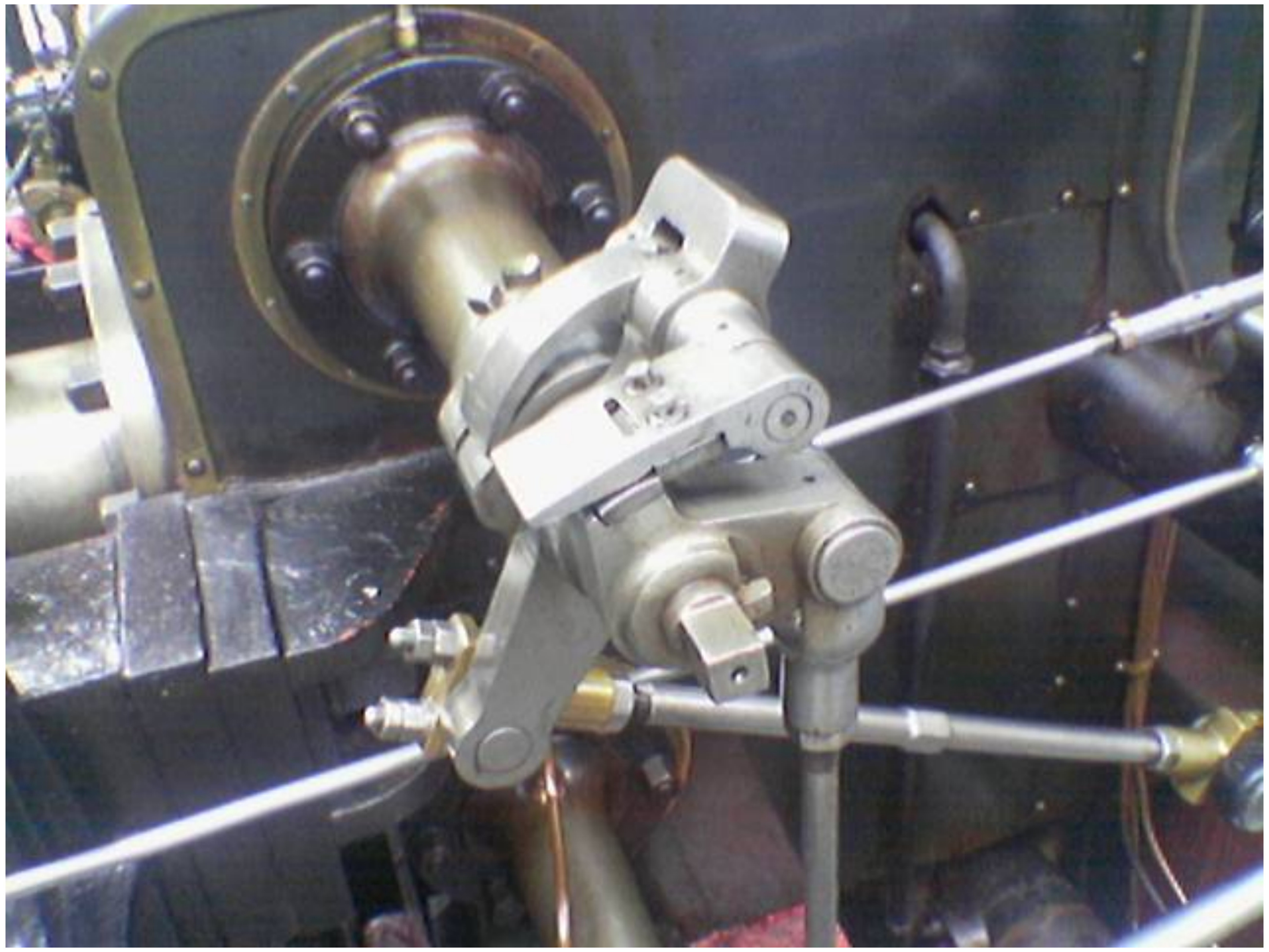
Example Image Processing Operations

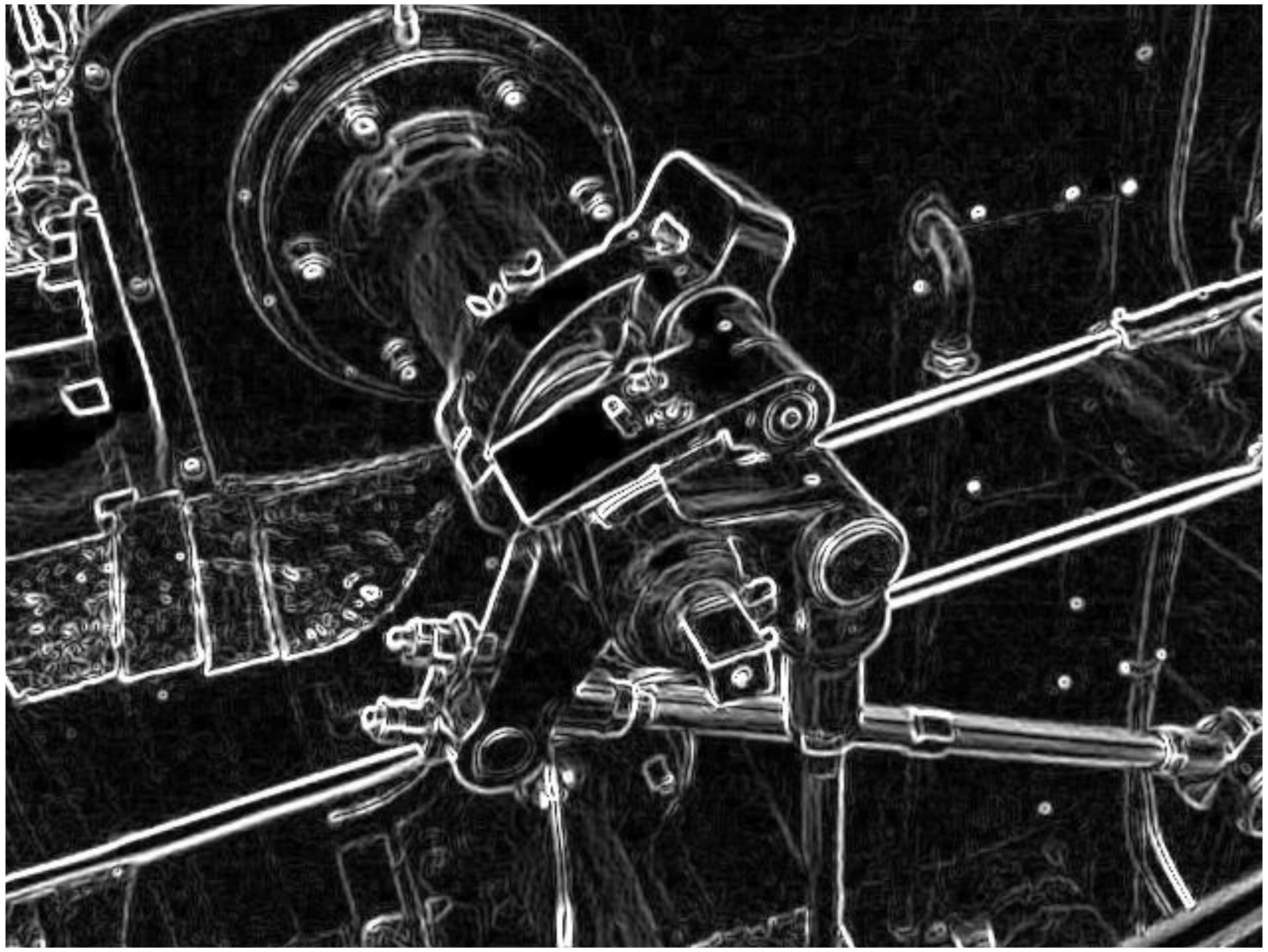
Blur

Example Image Processing Operations

Sharpen

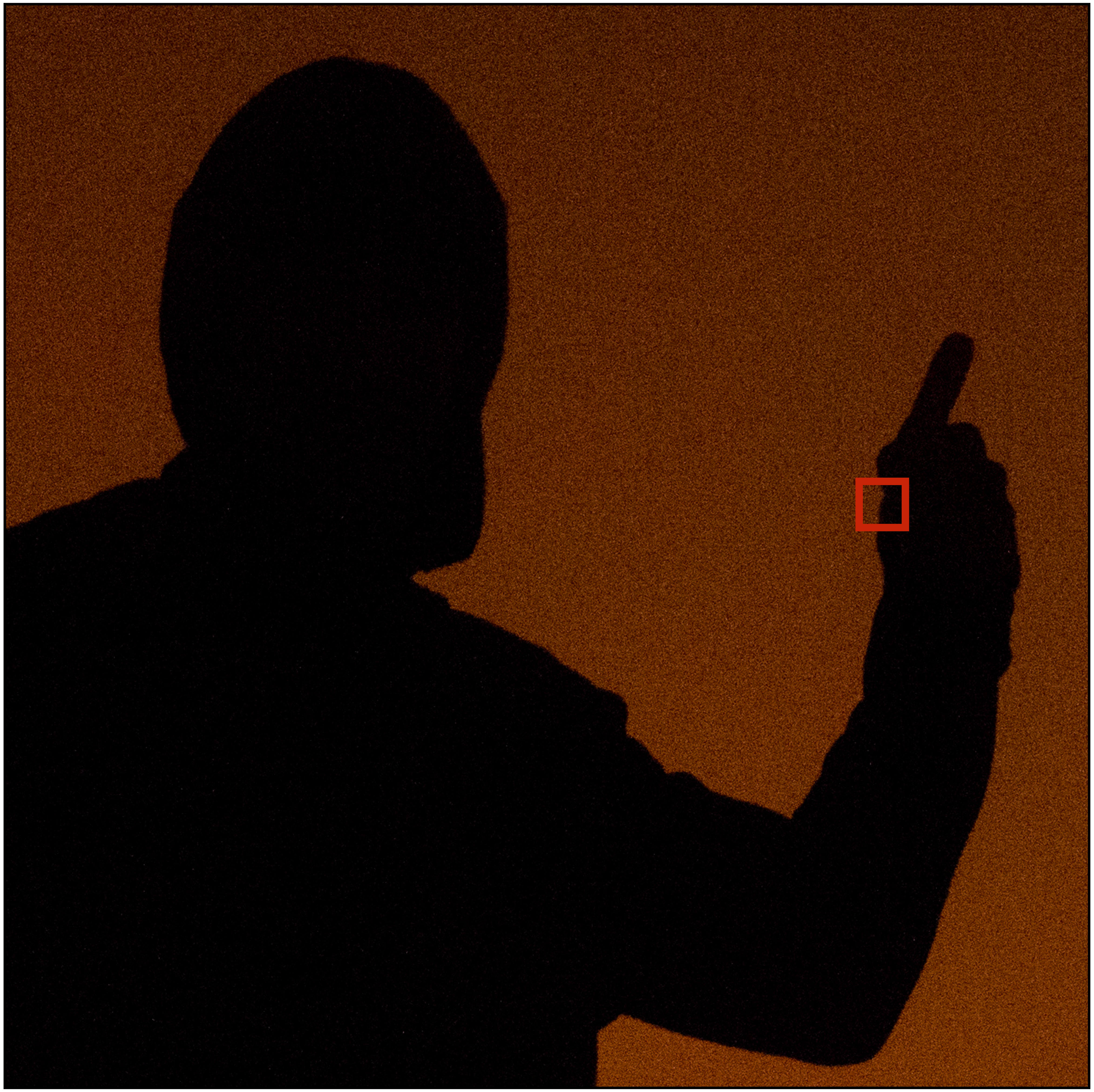
Edge Detection

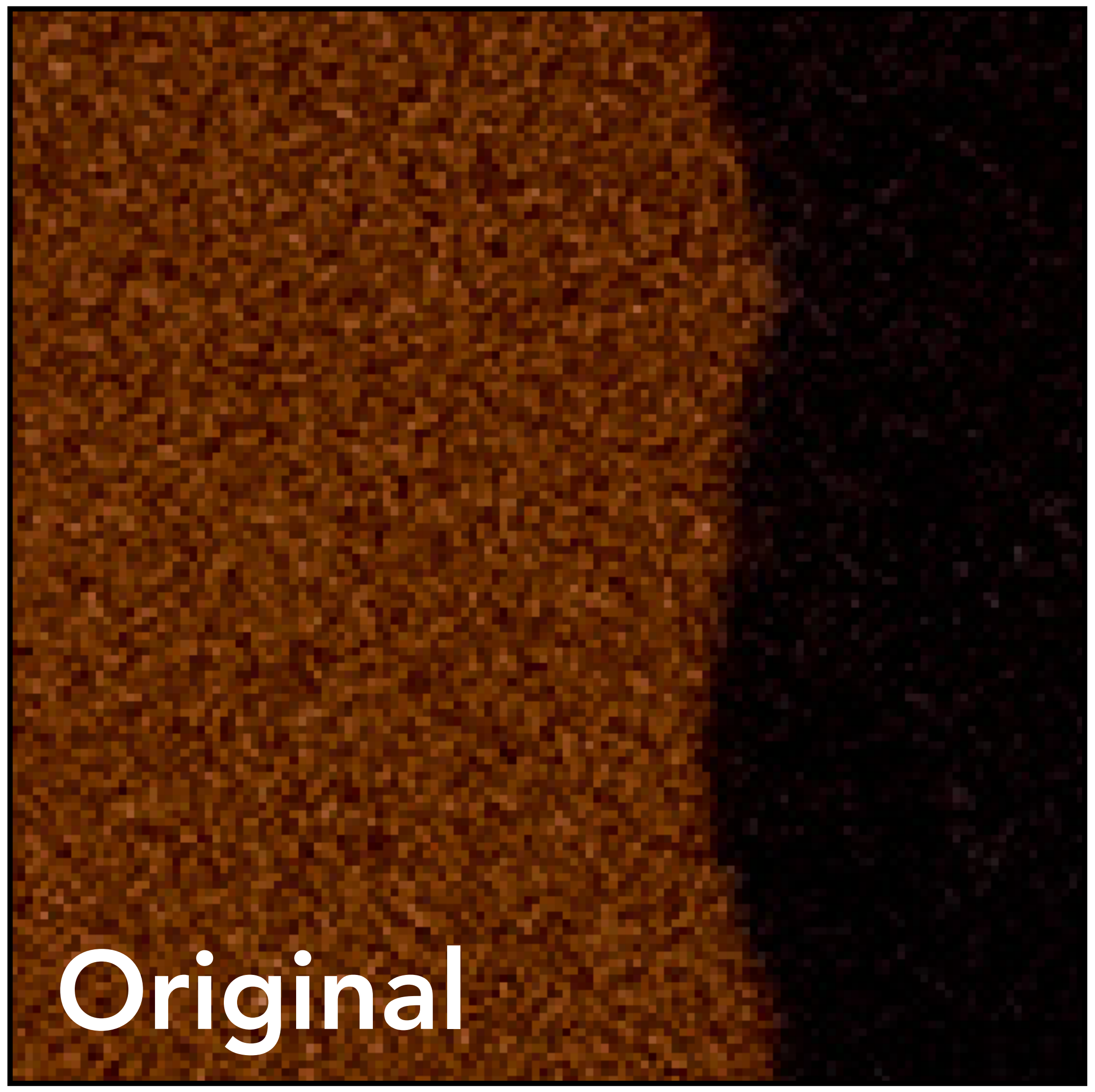




A "Smarter" Blur (Preserves Crisp Edges)

Denoising



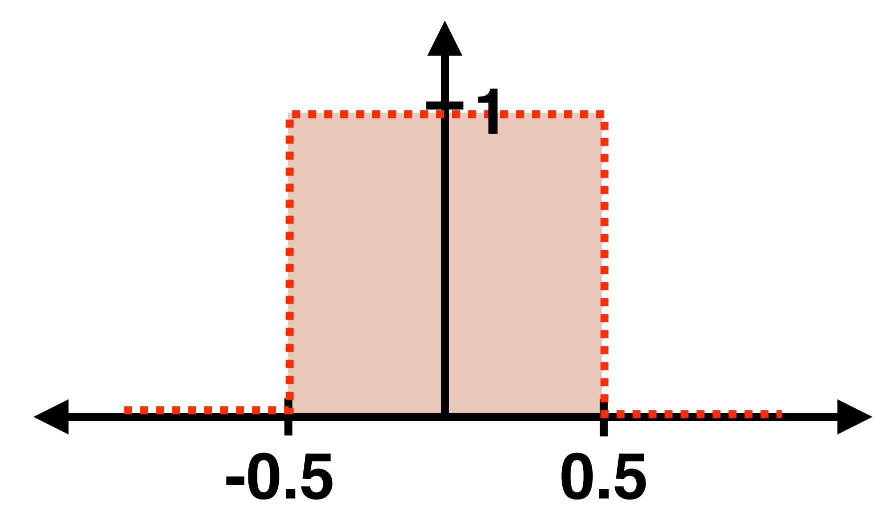


Review: Convolution

$$\underbrace{(f * g)(x)}_{\text{output signal}} = \int_{-\infty}^{\infty} \underbrace{f(y)}_{\text{filter}}\underbrace{g(x-y)}_{\text{input signal}}dy$$

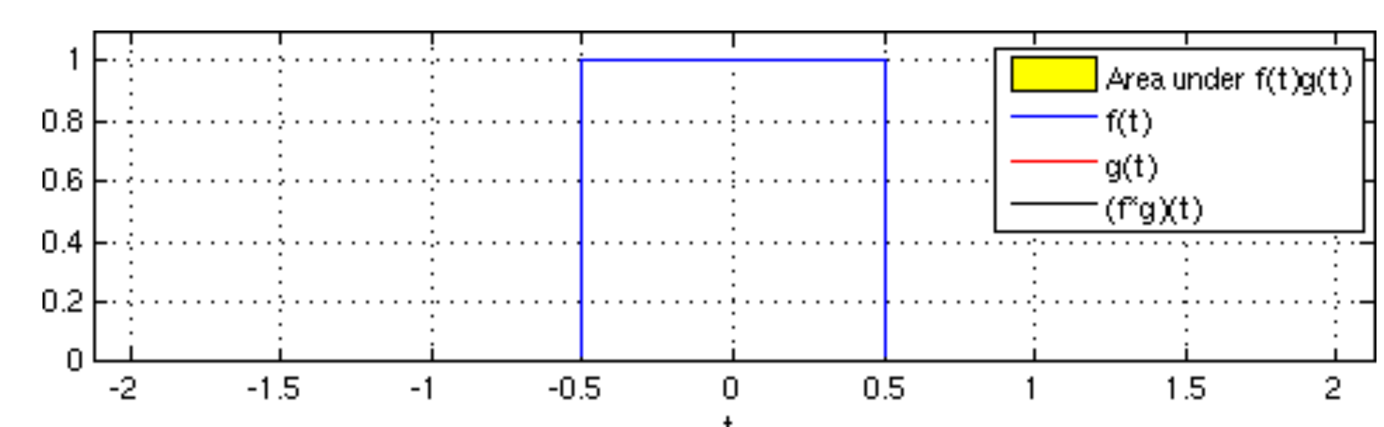
Example: convolution with "box" function:

$$f(x) = \begin{cases} 1 & |x| \leq 0.5 \\ 0 & otherwise \end{cases}$$



$$(f * g)(x) = \int_{-0.5}^{0.5} g(x-y)dy$$

f * g is a "smoothed" version of g



* In this gif f and g are swapped

Discrete 2D Convolution

$$\underbrace{(f * I)(x, y)}_{\text{output image}} = \sum_{i,j=-\infty}^{\infty} \underbrace{f(i, j)}_{\text{filter}} \underbrace{I(x - i, y - j)}_{\text{input image}}$$

Consider $f(i, j)$ that is nonzero only when: $-1 \leq i, j \leq 1$

Then:

$$(f * g)(x, y) = \sum_{i,j=-1}^{1} f(i, j) I(x - i, y - j)$$

And we can represent f(i,j) as a 3x3 matrix of values.

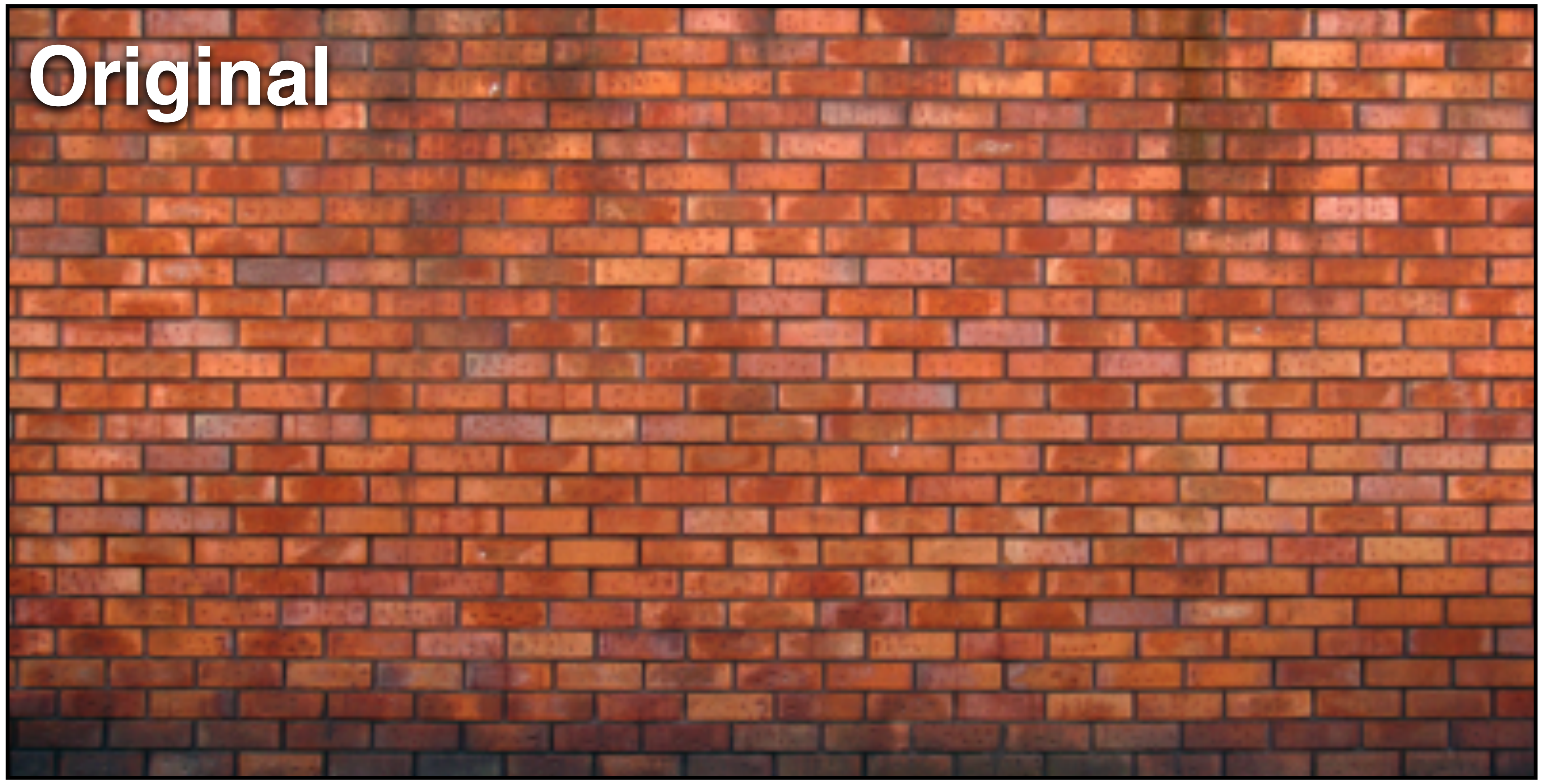
These values are often called “filter weights” or the “kernel”.

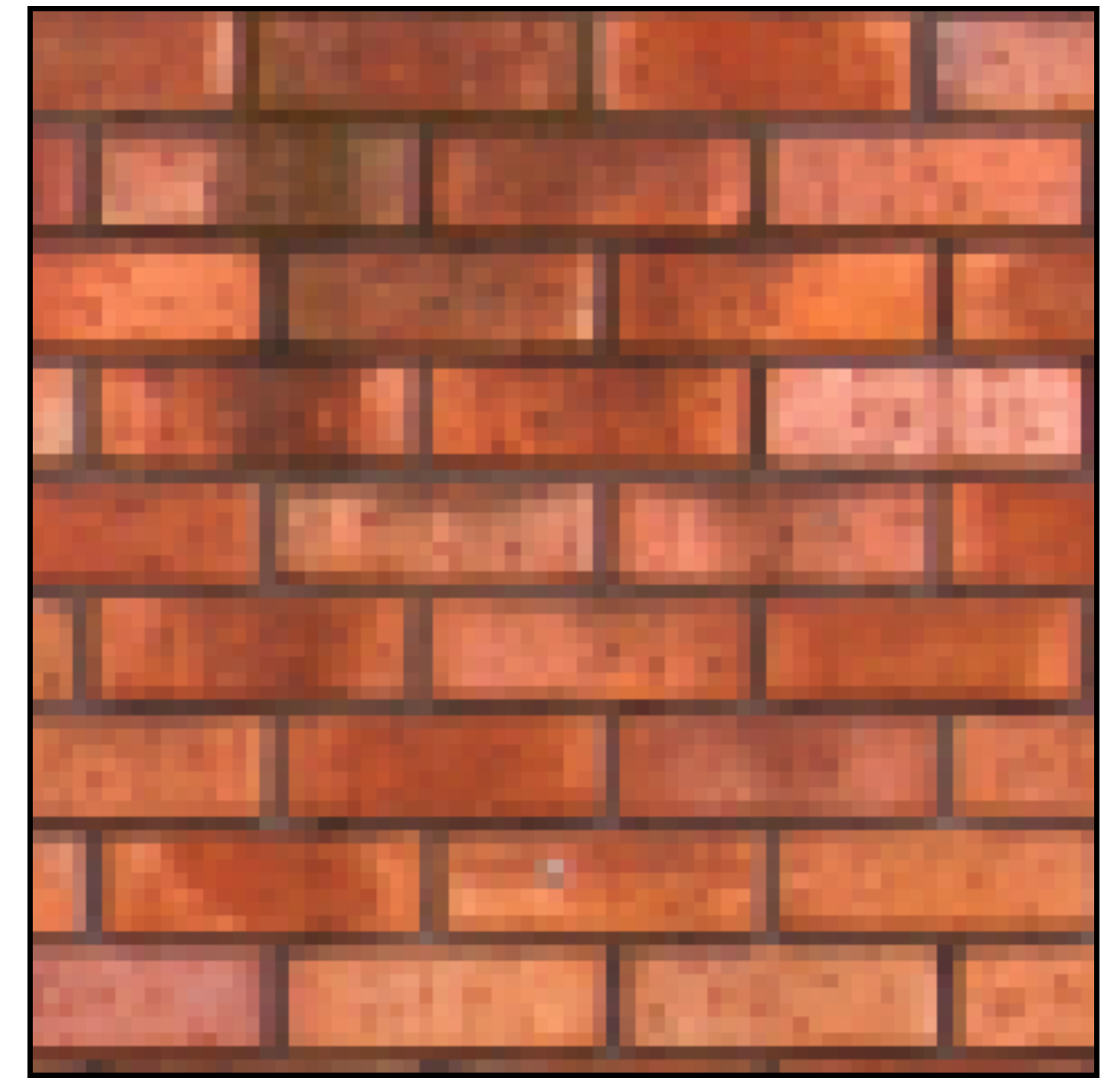
Simple 3x3 Box Blur

```
float input[(WIDTH+2) * (HEIGHT+2)];
float output[WIDTH * HEIGHT];

float weights[] = {1./9, 1./9, 1./9,
                   1./9, 1./9, 1./9,
                   1./9, 1./9, 1./9};

for (int j=0; j<HEIGHT; j++) {
  for (int i=0; i<WIDTH; i++) {
    float tmp = 0.f;
    for (int jj=0; jj<3; jj++)
      for (int ii=0; ii<3; ii++)
        tmp += input[(j+jj)*(WIDTH+2) + (i+ii)] * weights[jj*3 + ii];
    output[j*WIDTH + i] = tmp;
  }
}
```

Will ignore boundary pixels today and assume output image is smaller than input (makes convolution loop bounds much simpler to write)

7x7 Box Blur





Gaussian Blur

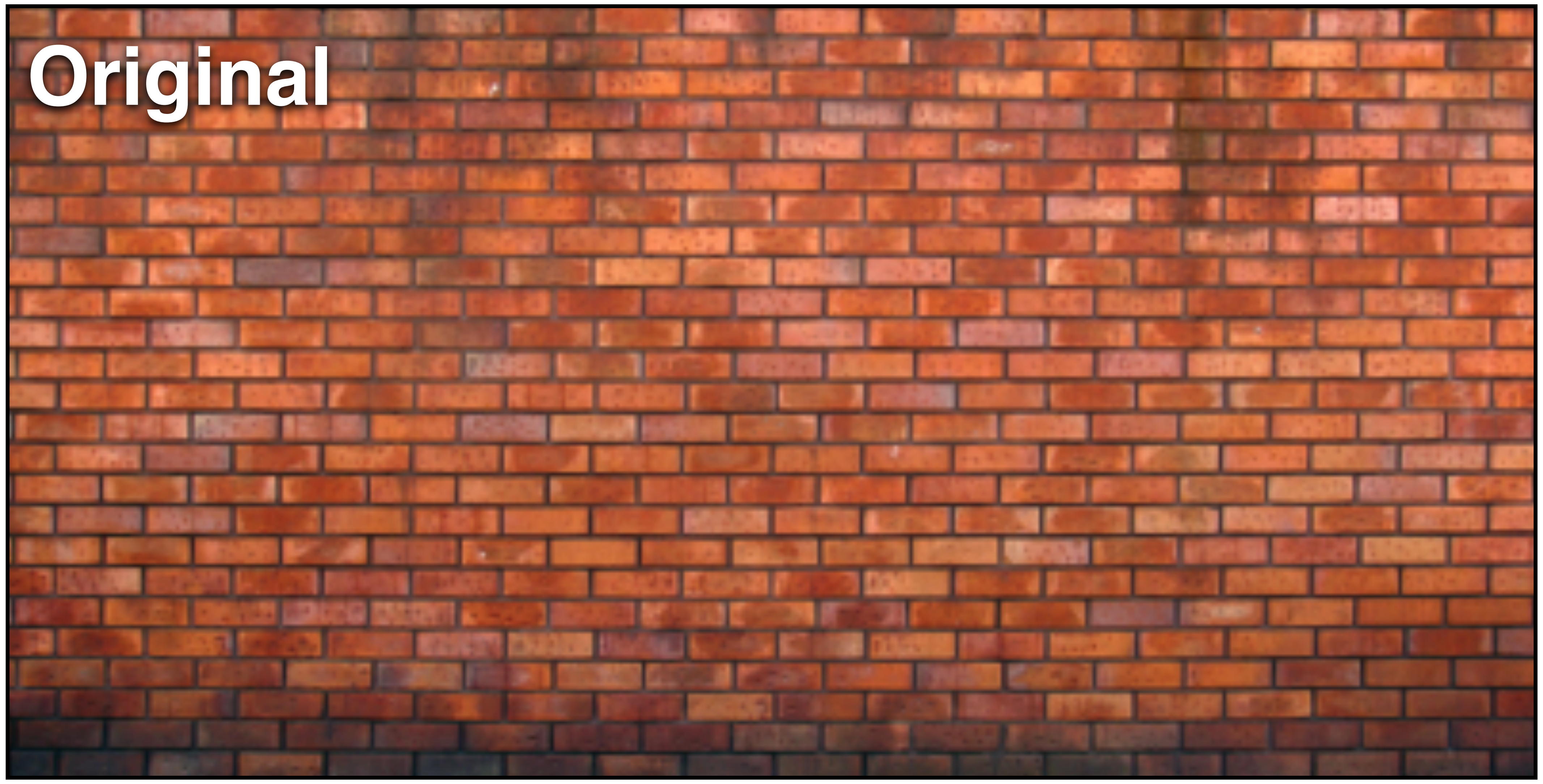
Obtain filter coefficients from sampling 2D Gaussian

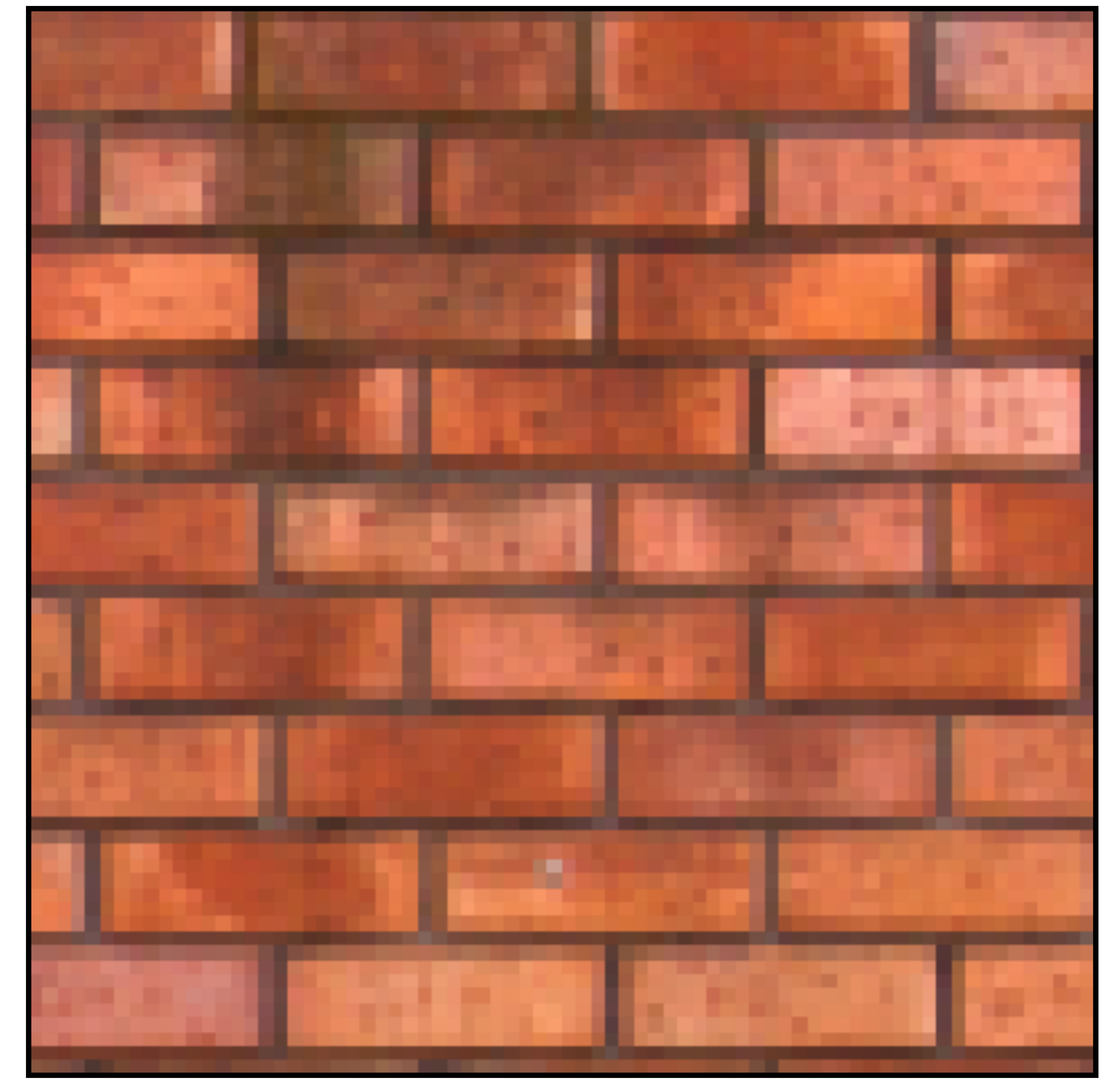
$$f(i,j) = \frac{1}{2\pi\sigma^2} e^{-\frac{i^2+j^2}{2\sigma^2}}$$

- Produces weighted sum of neighboring pixels (contribution falls off with distance)
  - Truncate filter beyond certain distance

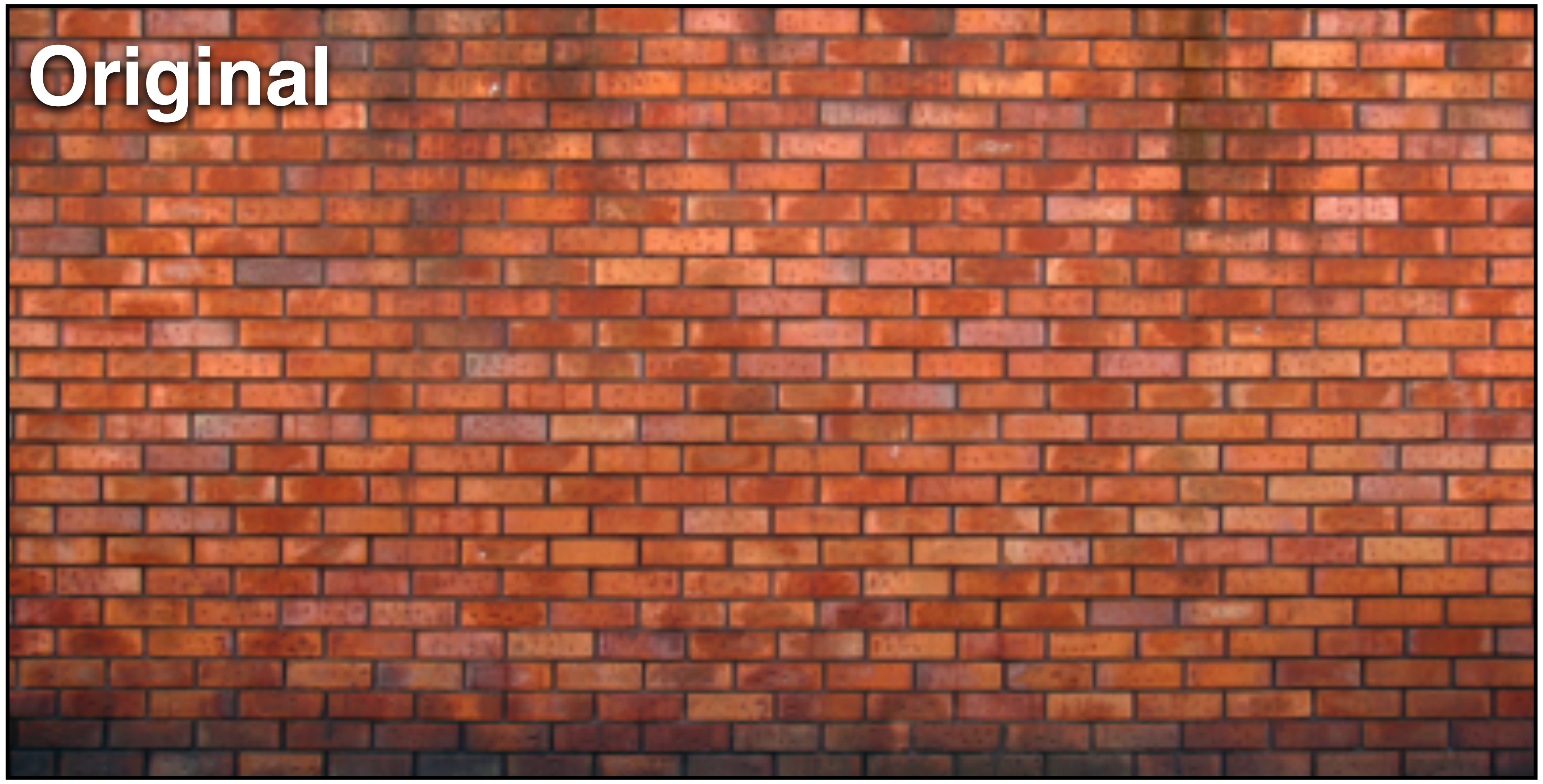
$$\begin{bmatrix} .075 & .124 & .075 \\ .124 & .204 & .124 \\ .075 & .124 & .075 \end{bmatrix}$$

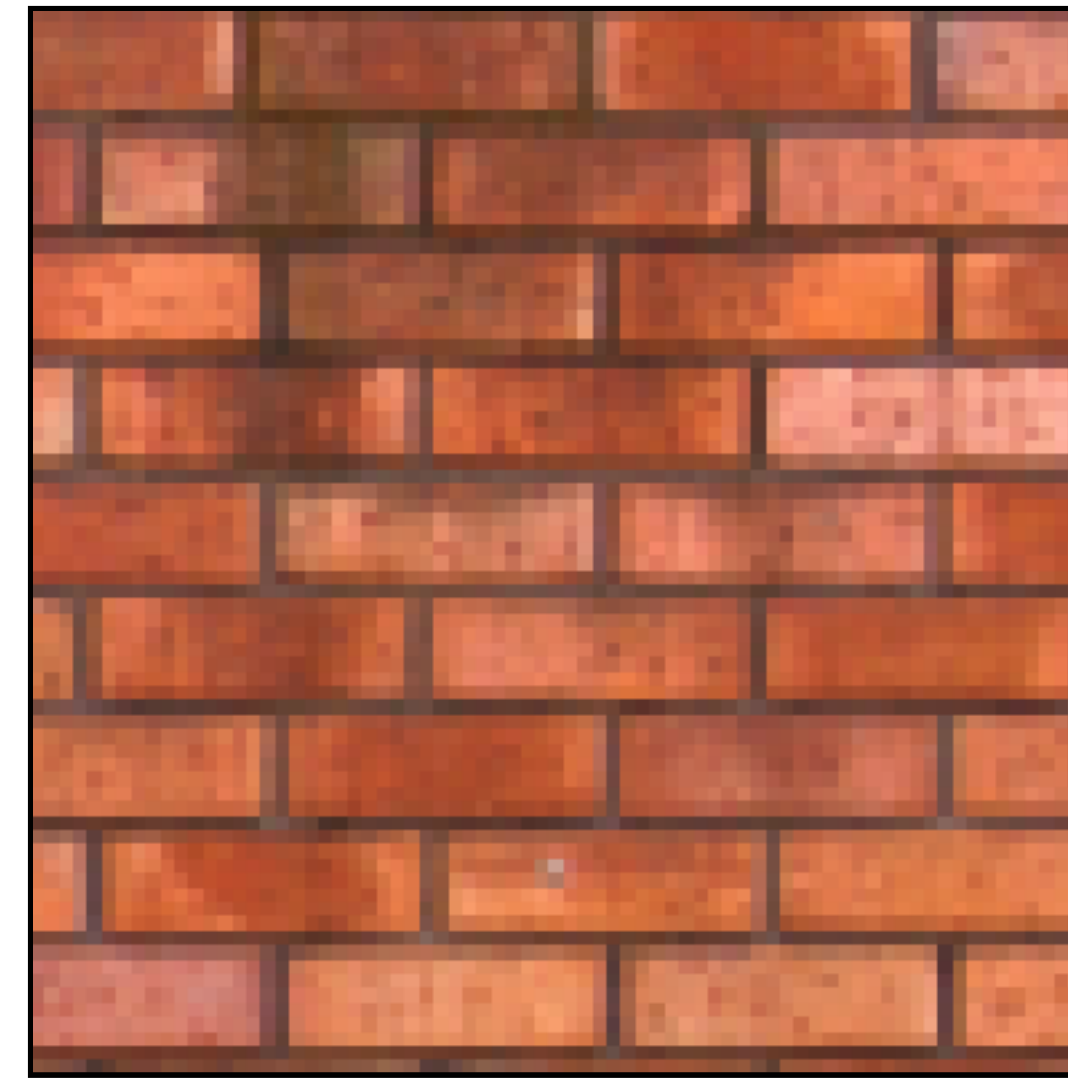
7x7 Gaussian Blur

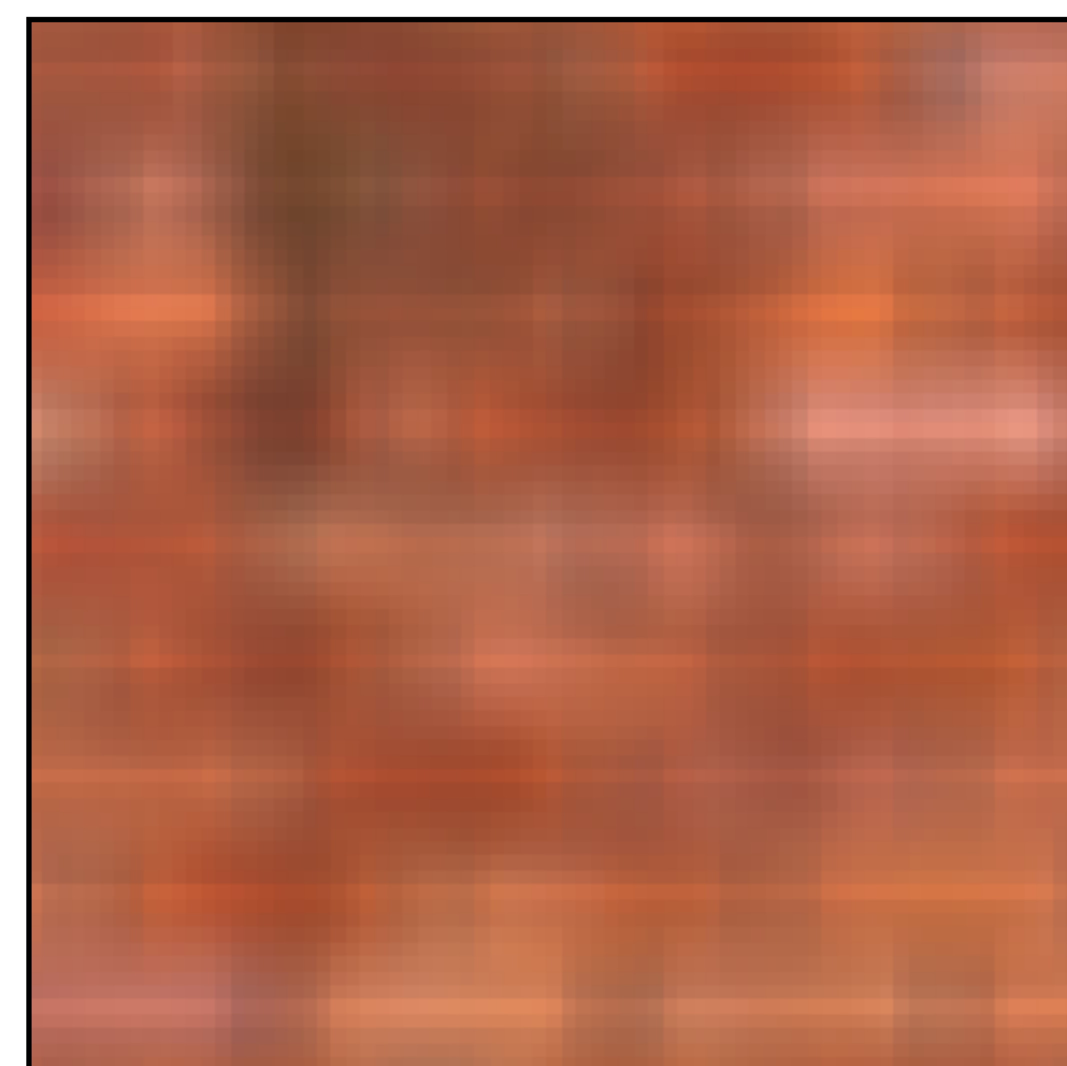




Compare: 7x7 Box Blur





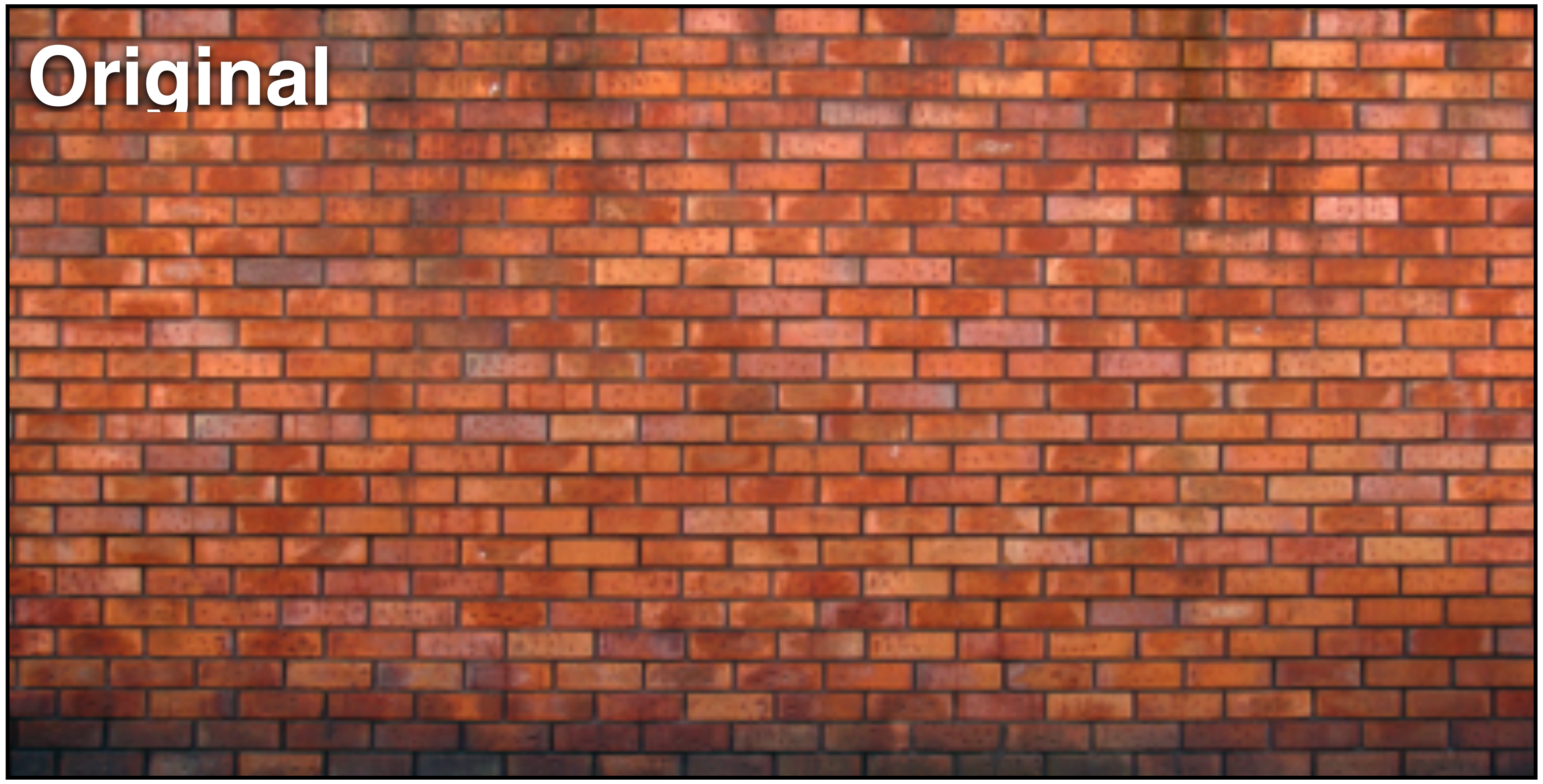


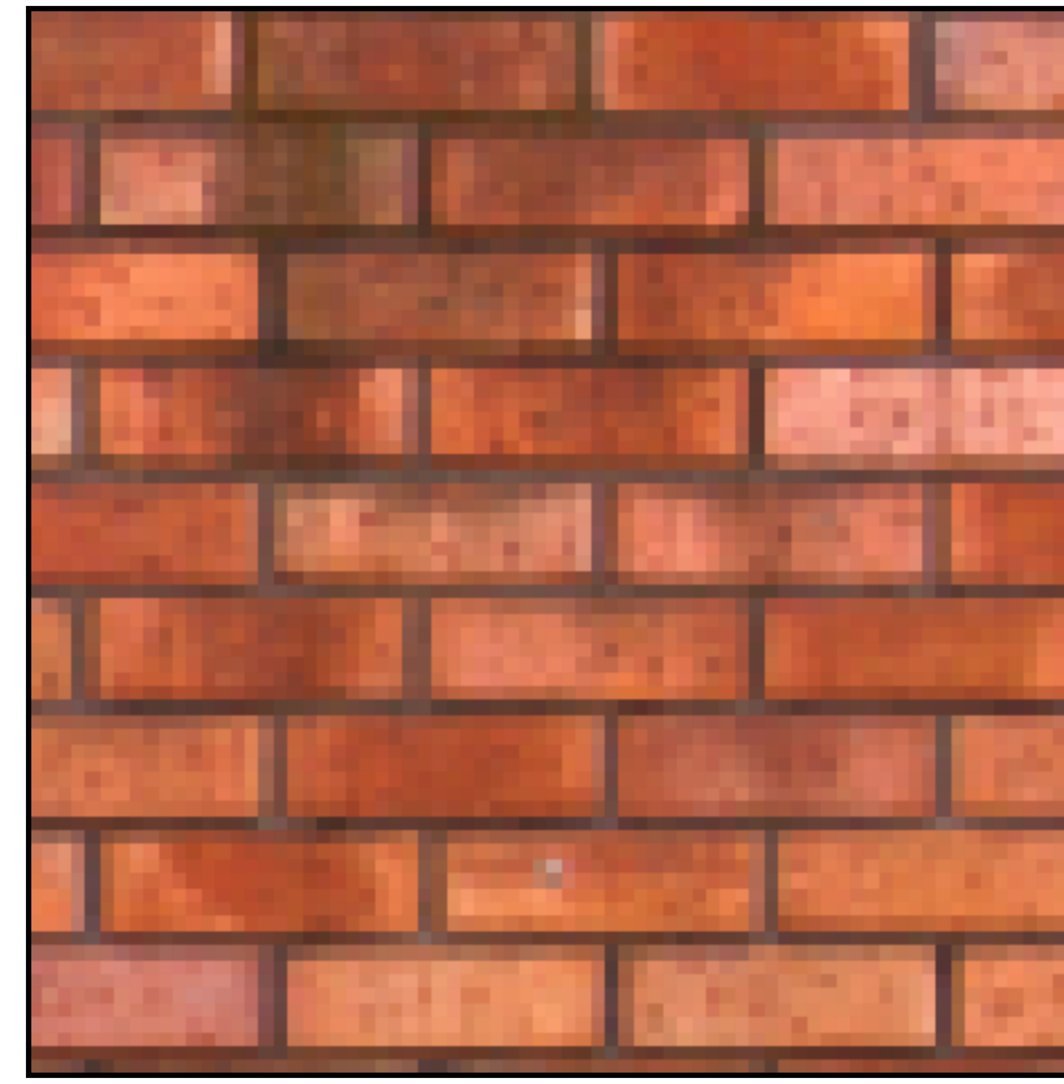
What Does Convolution with this Filter Do?

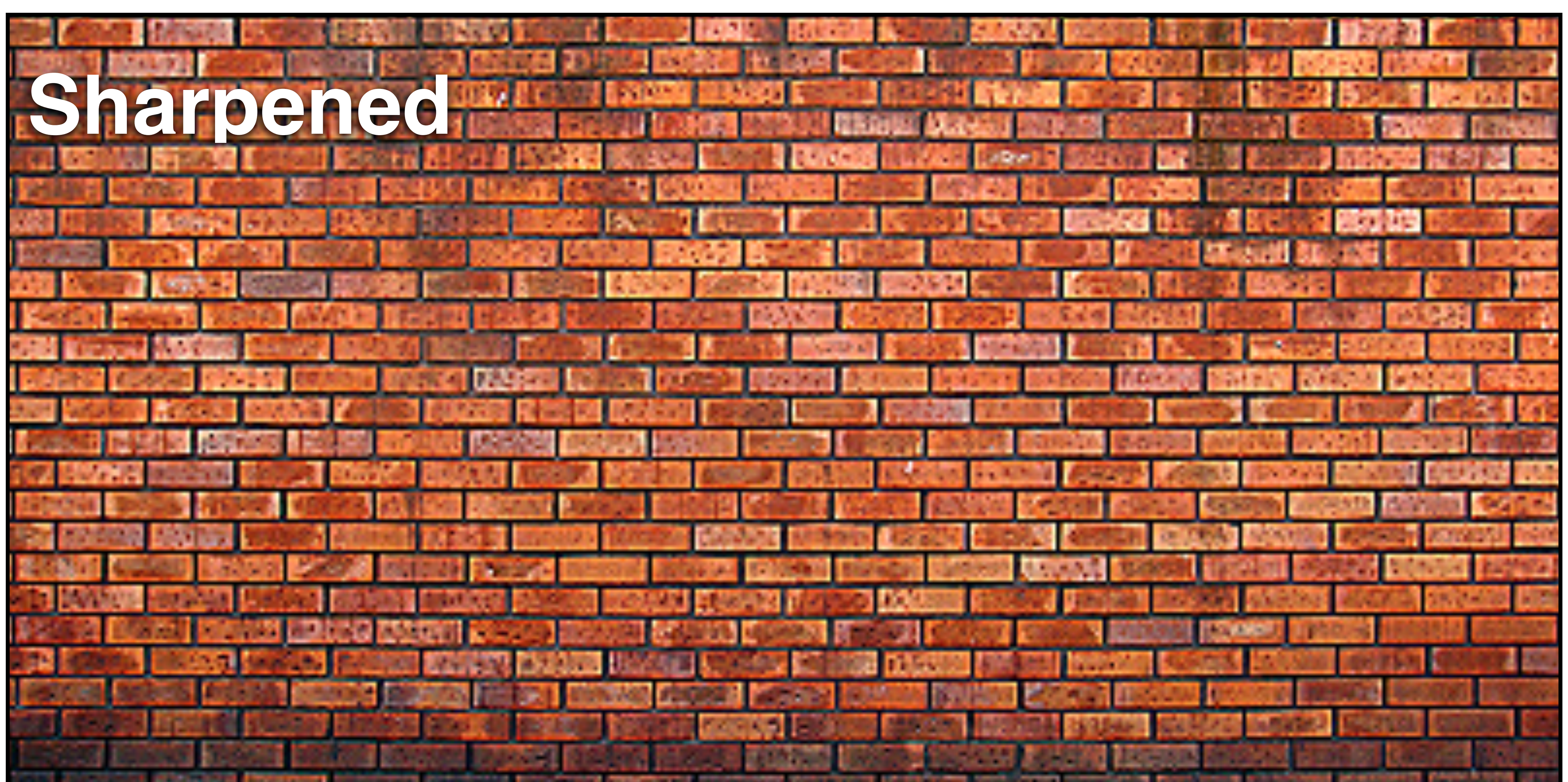
$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

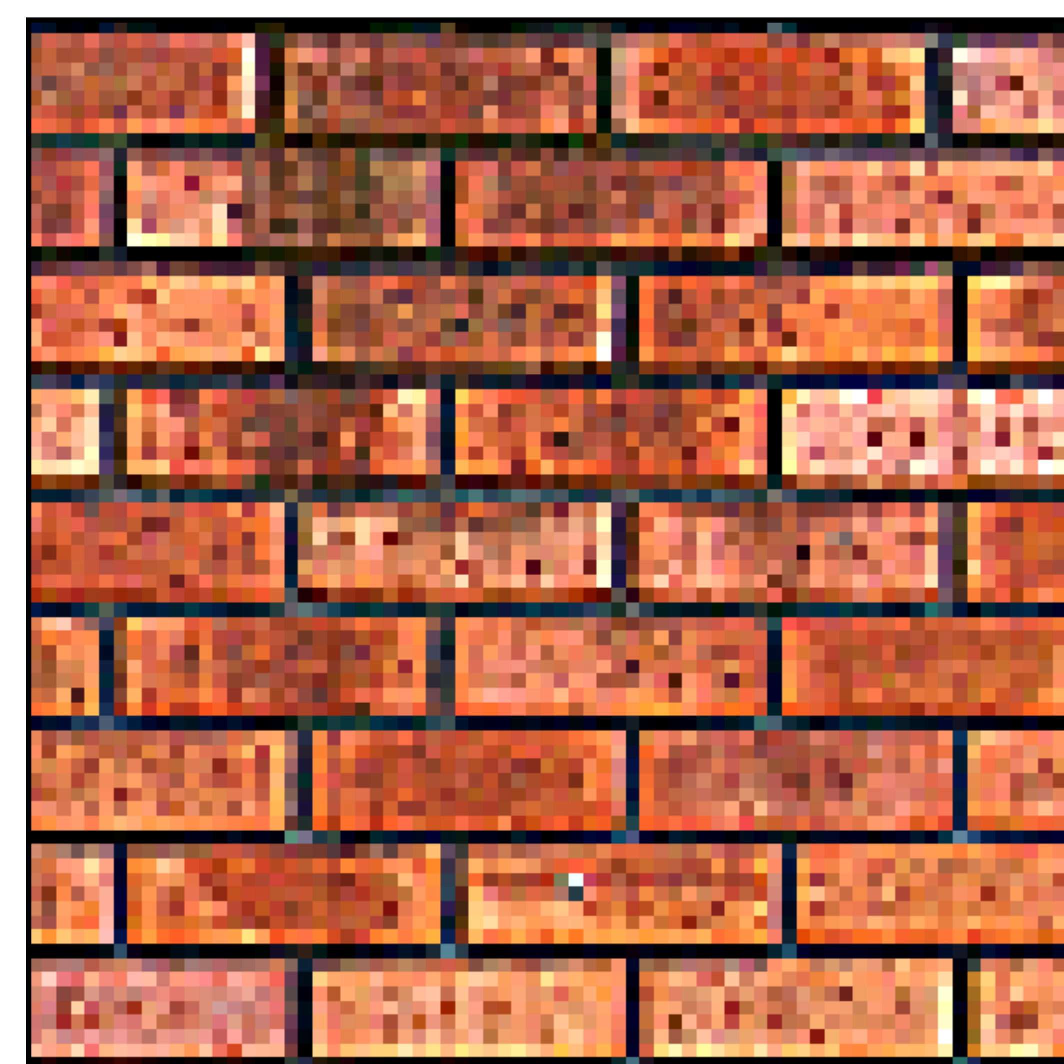
Sharpens image!

3x3 Sharpen Filter









What Does Convolution with these Filters Do?

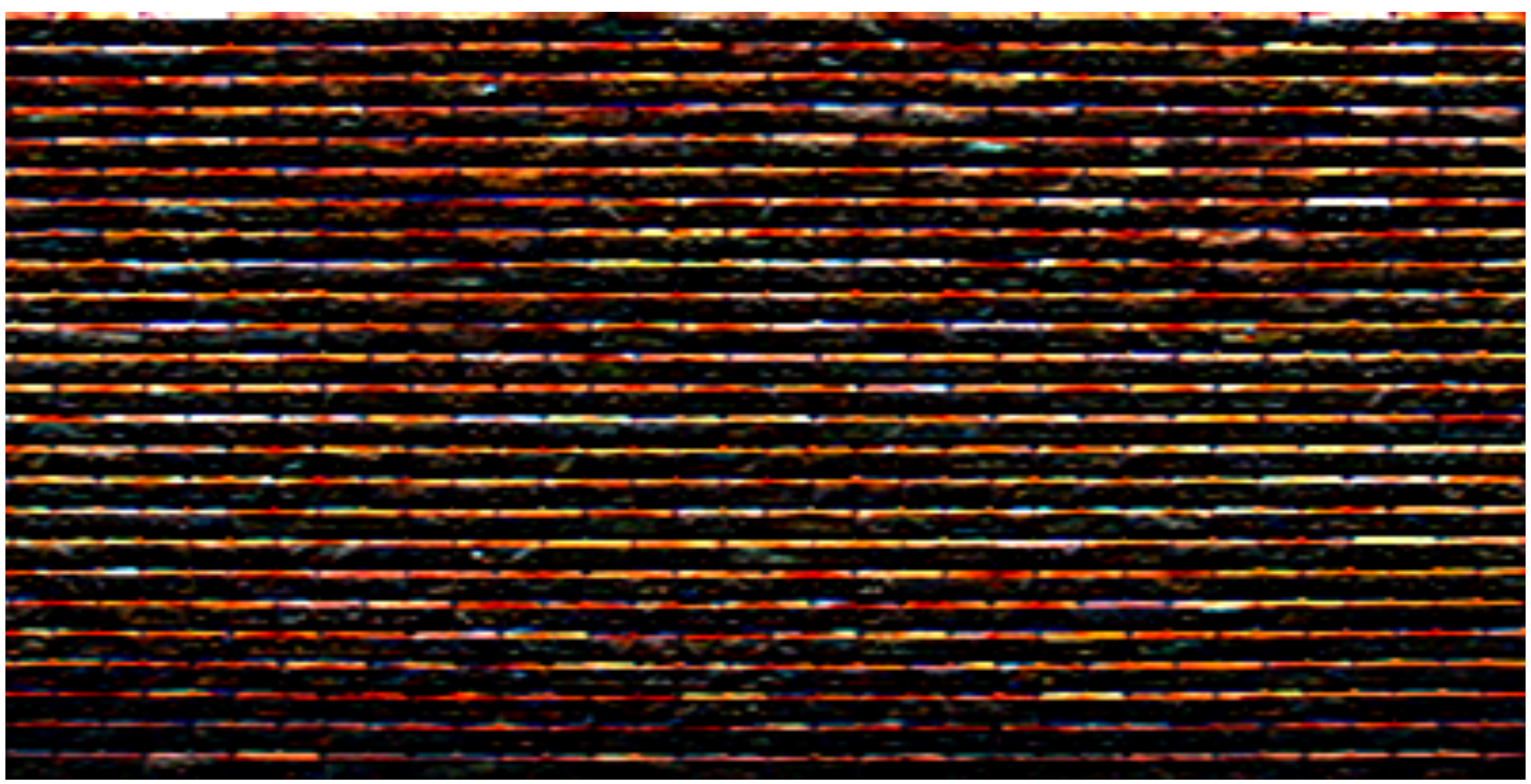
$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Extracts horizontal gradients

Extracts vertical gradients

Gradient Detection Filters

Horizontal gradients



Vertical gradients

Note: you can think of a filter as a "detector" of a pattern, and the magnitude of a pixel in the output image as the "response" of the filter to the region surrounding each pixel in the input image (this is a common interpretation in computer vision)

Sobel Edge Detection

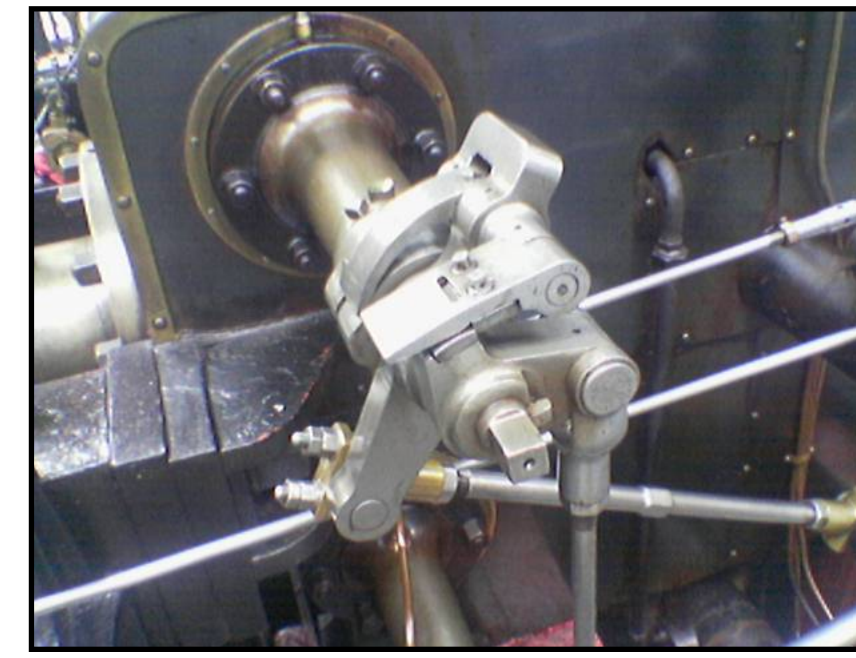
$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * I$$

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} * I$$

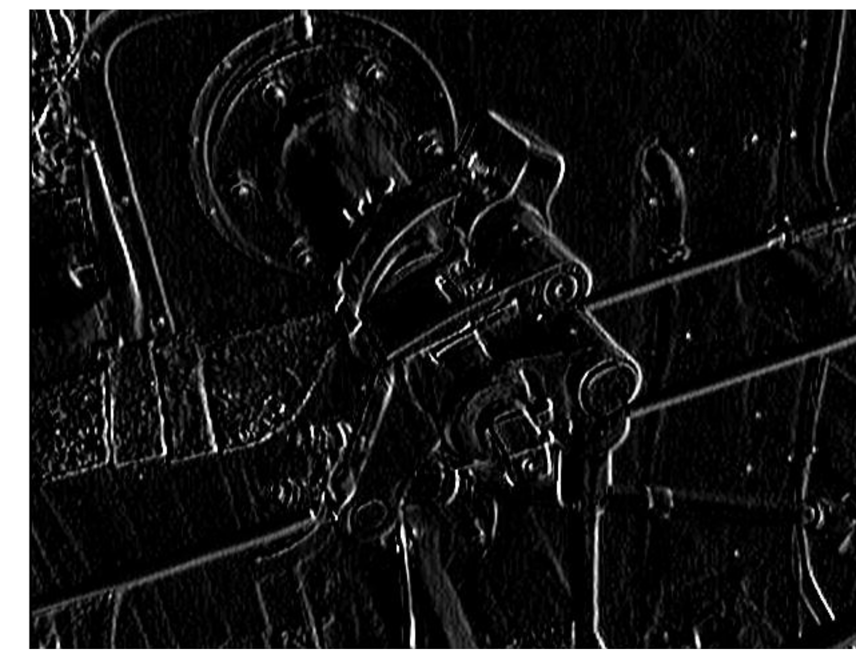
- **Find pixels with large gradients**

$$G = \sqrt{{G_x}^2 + {G_y}^2}$$

Pixel-wise operation on images

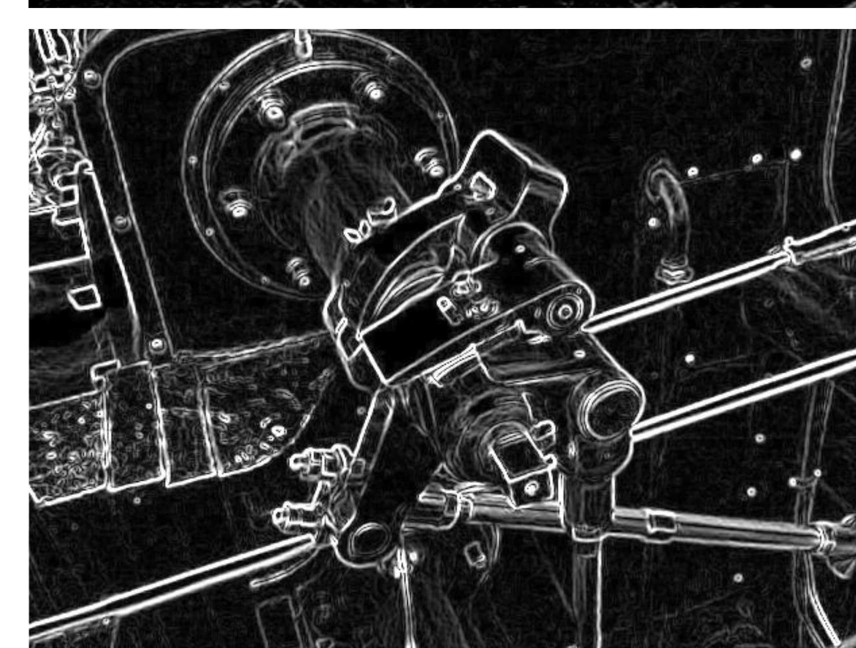


G_x



G_y

G



Algorithmic Cost of
Convolution-Based Image Processing

Cost of Convolution with N x N Filter?

```
float input[(WIDTH+2) * (HEIGHT+2)];
float output[WIDTH * HEIGHT];

float weights[] = {1./9, 1./9, 1./9,
                   1./9, 1./9, 1./9,
                   1./9, 1./9, 1./9};

for (int j=0; j<HEIGHT; j++) {
  for (int i=0; i<WIDTH; i++) {
    float tmp = 0.f;
    for (int jj=0; jj<3; jj++)
      for (int ii=0; ii<3; ii++)
        tmp += input[(j+jj)*(WIDTH+2) + (i+ii)] * weights[jj*3 + ii];
    output[j*WIDTH + i] = tmp;
  }
}
```

In this 3x3 box blur example:
Total work per image = 9 x WIDTH x HEIGHT

For N x N filter: N^2 x WIDTH x HEIGHT

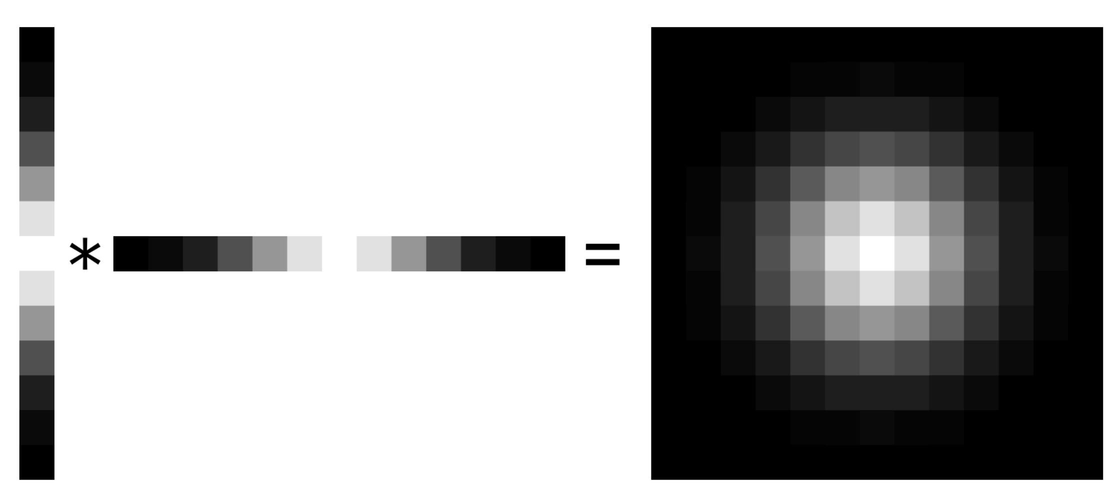
Separable Filters

A filter is separable if is the product of two other filters

- Examples: a 2D box blur

$$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\1 & 1 & 1\\1 & 1 & 1\end{bmatrix} = \frac{1}{3}\begin{bmatrix}1\\1\\1\end{bmatrix} * \frac{1}{3}\begin{bmatrix}1 & 1 & 1\end{bmatrix}$$

- Exercise: write 2D gaussian and vertical/horizontal gradient detection filters as product of 1D filters (they are separable!)



Key property: 2D convolution with separable filter can be written as two 1D convolutions!

Fast 2D Box Blur via Two 1D Convolutions

```
int WIDTH = 1024
int HEIGHT = 1024;
float input[(WIDTH+2) * (HEIGHT+2)];
float tmp_buf[WIDTH * (HEIGHT+2)];
float output[WIDTH * HEIGHT];

float weights[] = {1./3, 1./3, 1./3};

for (int j=0; j<(HEIGHT+2); j++)
  for (int i=0; i<WIDTH; i++) {
    float tmp = 0.f;
    for (int ii=0; ii<3; ii++)
      tmp += input[j*(WIDTH+2) + i+ii] * weights[ii];
    tmp_buf[j*WIDTH + i] = tmp;
  }

for (int j=0; j<HEIGHT; j++) {
  for (int i=0; i<WIDTH; i++) {
    float tmp = 0.f;
    for (int jj=0; jj<3; jj++)
      tmp += tmp_buf[(j+jj)*WIDTH + i] * weights[jj];
    output[j*WIDTH + i] = tmp;
  }
}
```

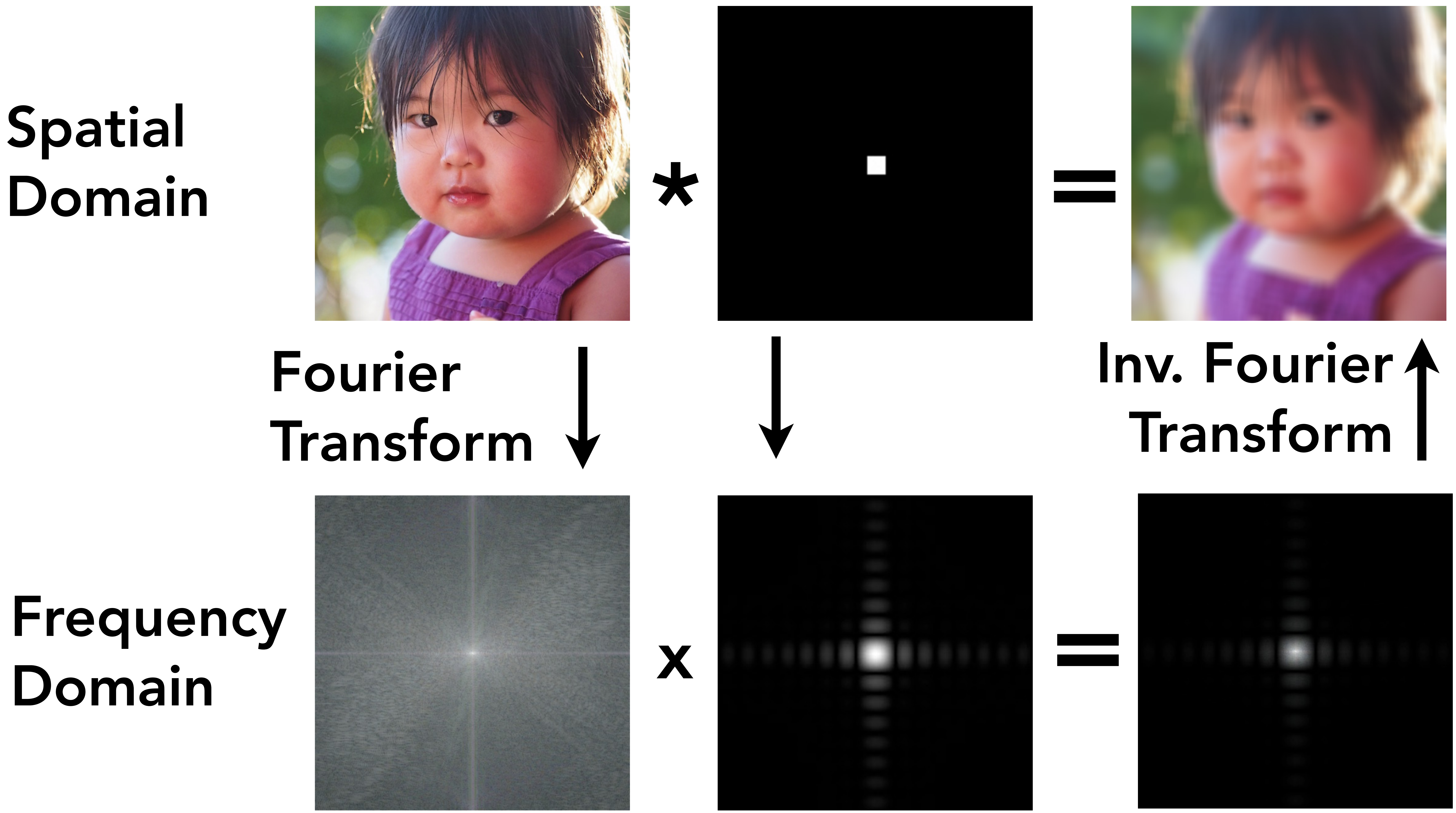
Total work per image = 6 x WIDTH x HEIGHT

For NxN filter: 2N x WIDTH x HEIGHT

Extra cost of this approach?

Storage!
Challenge: can you achieve this work complexity without incurring this cost?

Recall: Convolution Theorem



Efficiency?

When is it faster to implement a filter by convolution in the spatial domain?

When is it faster to implement a filter by multiplication in the frequency domain?

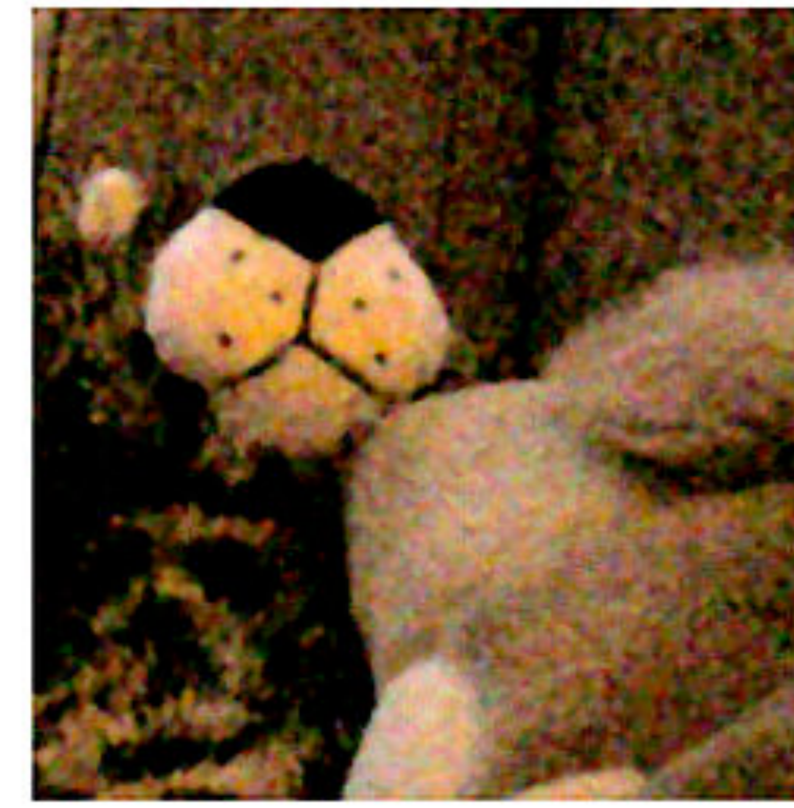
Data-Dependent Filters

Median Filter

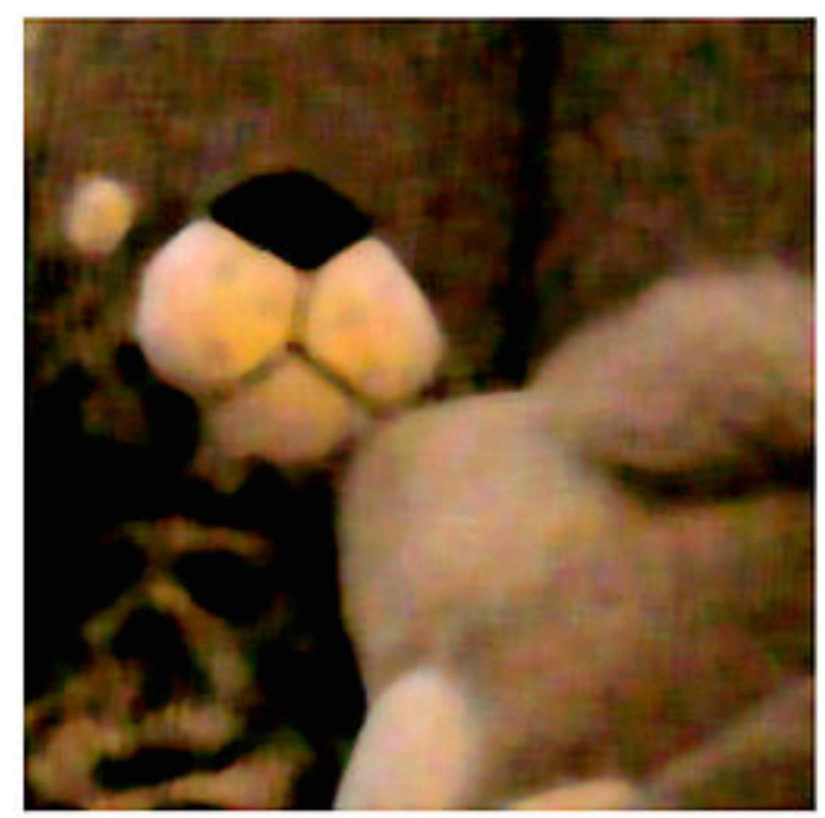
- Replace pixel with median of its neighbors
  - Useful noise reduction filter: unlike gaussian blur, one bright pixel doesn’t drag up the average for entire region
- Not linear, not separable
  - Filter weights are 1 or 0 (depending on image content)

```
uint8 input[(WIDTH+2) * (HEIGHT+2)];
uint8 output[WIDTH * HEIGHT];
for (int j=0; j<HEIGHT; j++)
  for (int i=0; i<WIDTH; i++)
    output[j*WIDTH + i] =
       // compute median of pixels
       // in surrounding 5x5 pixel window
```

original image



1px median filter



3px median filter

10px median filter

Bilateral Filter

Example use of bilateral filter: removing noise while preserving image edges

Intuition

Isotropic filtering

Anisotopic, data dependent filtering

Bilateral Filter

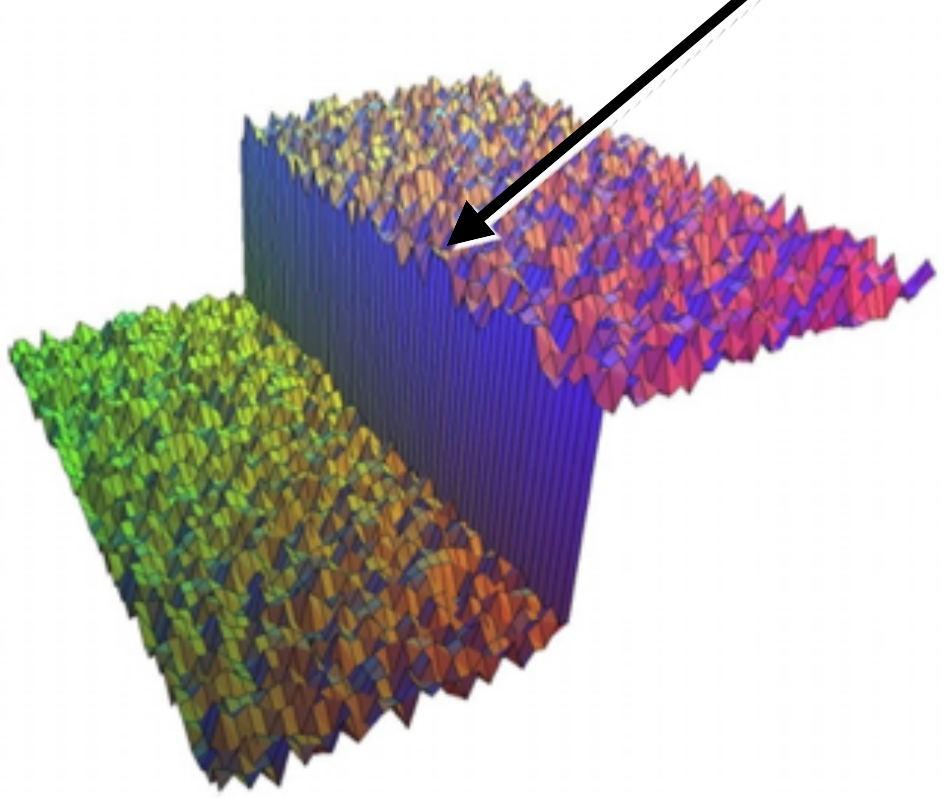
$$\mathrm{BF}[I](x,y) = \underbrace{\sum_{i,j}}_{\substack{\text{For all pixels in support}\\ \text{region of Gaussian kernel}}} f(\|\underbrace{I(x-i,y-j) - I(x,y)}_{\substack{\text{Re-weight based on difference}\\ \text{in input image pixel values}}}\|)\overbrace{G(i,j)}^{\text{Gaussian blur kernel}}\overbrace{I(x-i,y-j)}^{\text{Input image}}$$

- Value of output pixel (x,y) is the weighted sum of all pixels in the support region of a truncated gaussian kernel
- But weight is combination of both spatial distance and intensity difference. (another non-linear, data-dependent filter)
- The bilateral filter is an “edge preserving” filter: down-weight contribution of pixels on the other side of strong edges. $f(x)$ defines what “strong edge means”
- Spatial distance weight term $f(x)$ could itself be a gaussian
    - Or very simple: $f(x) = 0$ *if* $x >$ *threshold*, 1 *otherwise*

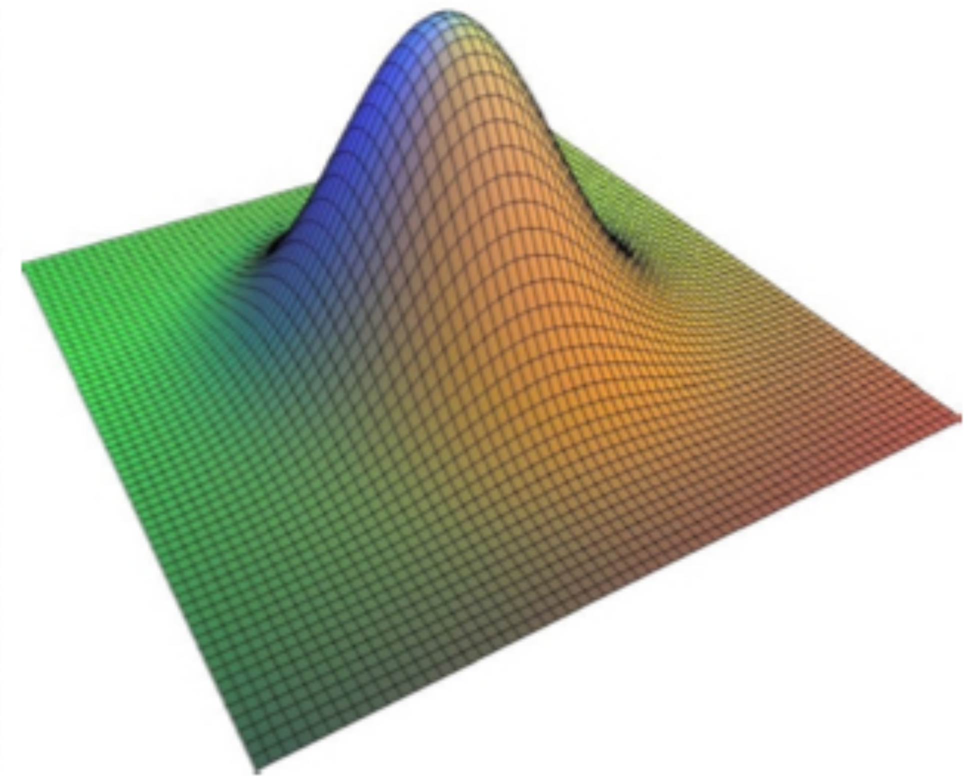
Bilateral Filter

Pixels with significantly different intensity relative to *p* contribute little to filtered result (they are on the "other side of the edge")

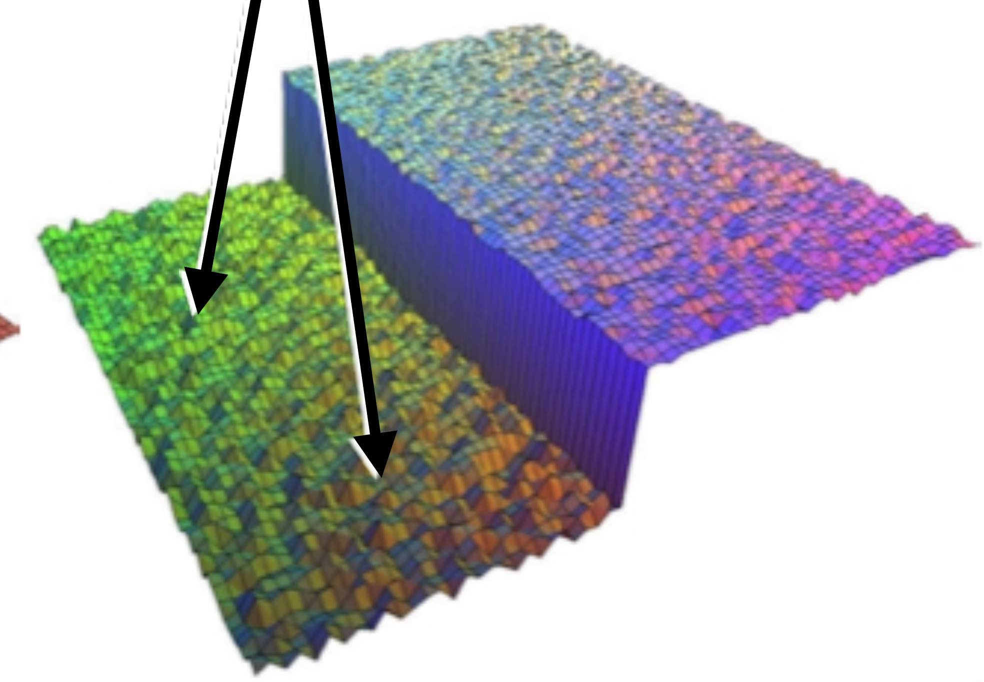
Input pixel *p*



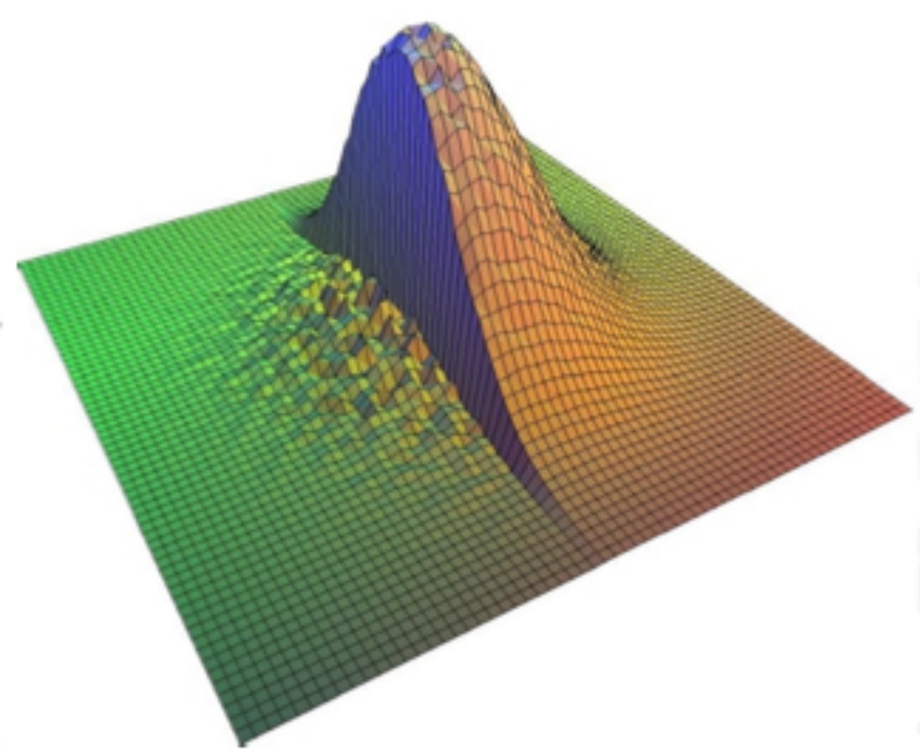
Input image



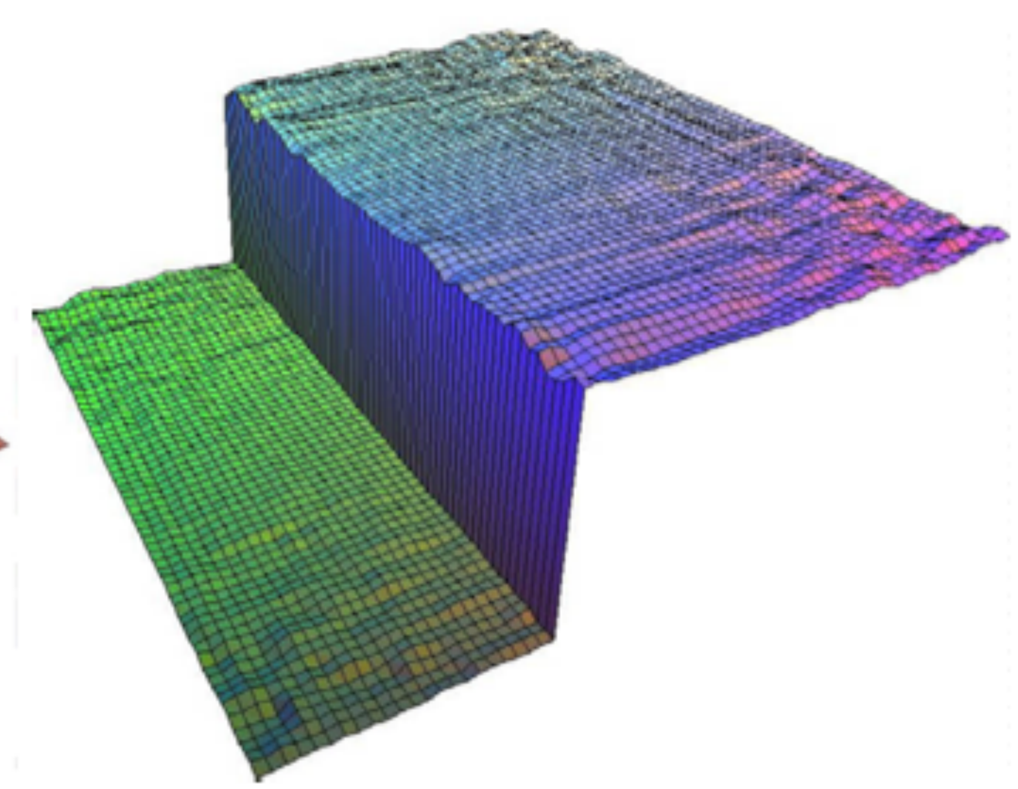
G(): gaussian about input pixel *p*



f(): Influence of support region



G x f: filter weights for pixel *p*



Filtered output image

Test your understanding: What would change on this slide if pixel p were on the lower side of the edge?

Figure credit: Durand and Dorsey, "Fast Bilateral Filtering for the Display of High-Dynamic-Range Images", SIGGRAPH 2002

Bilateral Filter: Kernel Depends on Image Content

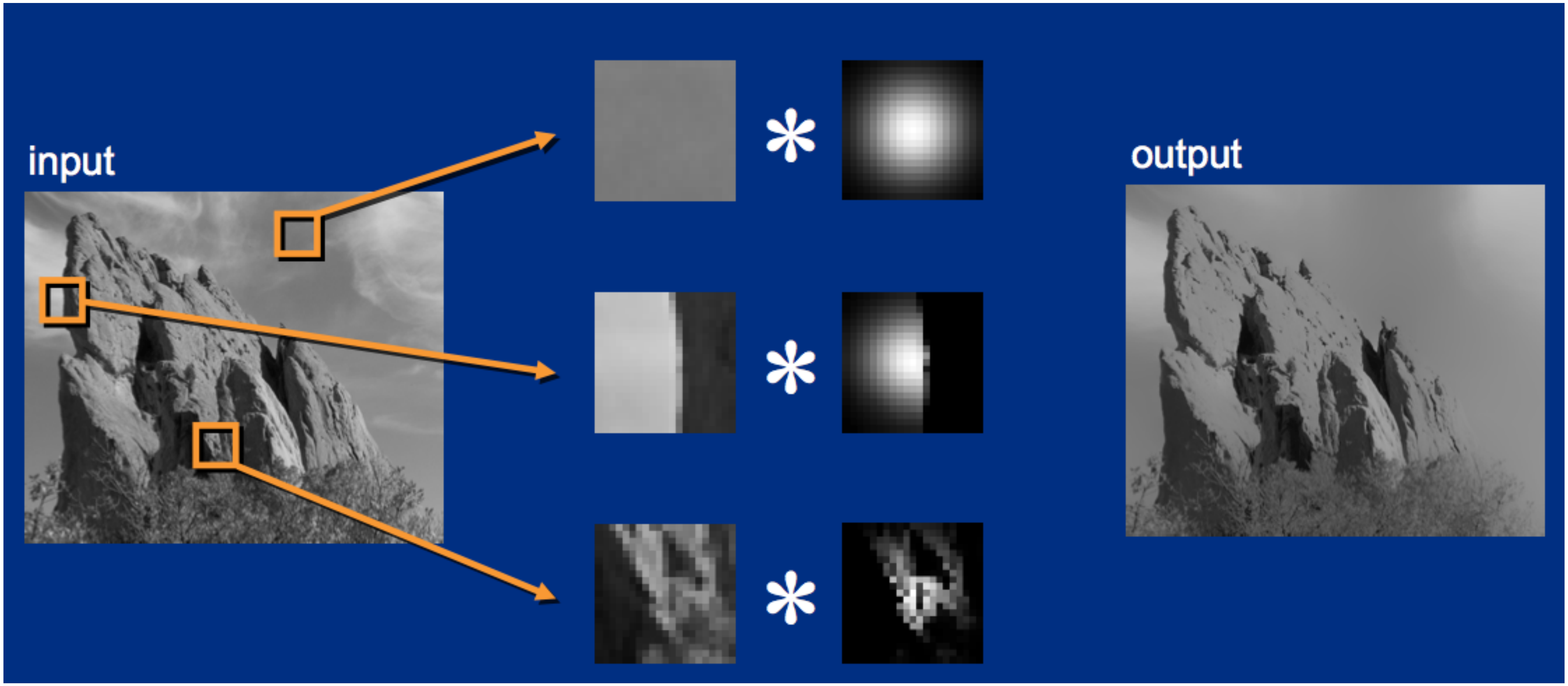


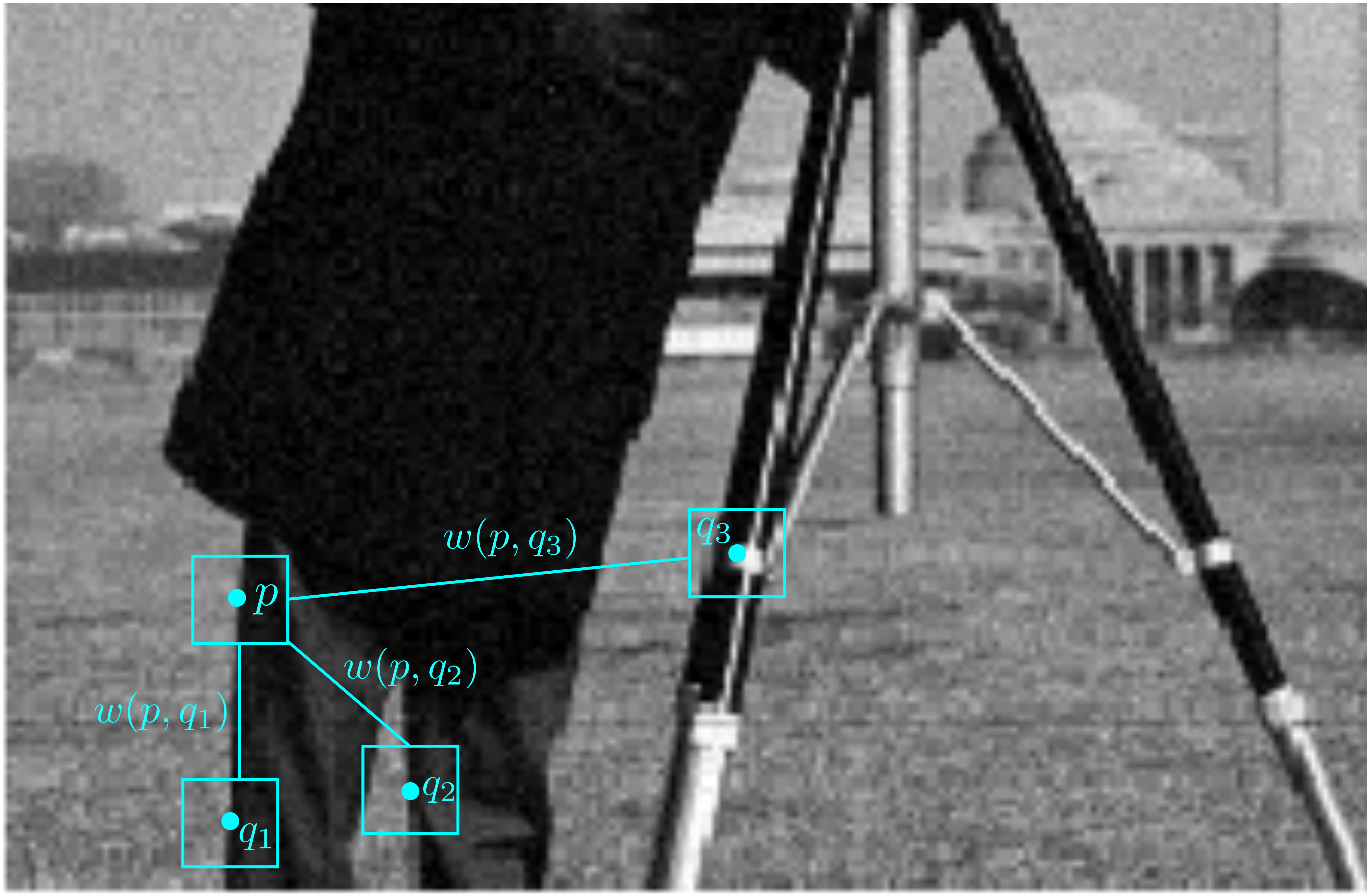
Figure credit: SIGGRAPH 2008 Course: "A Gentle Introduction to Bilateral Filtering and its Applications" Paris et al.

Data-Driven Image Processing:
“Image Manipulation by Example“

Denoising with Non-Local Means

Large weight for input pixels that have similar neighborhood as p

- Intuition: filtered result is the average of pixels "like" this one
    - Most similar pixel has no reason to be nearby at all!!
- In example below-right: q1 and q2 have high weight, q3 has low weight



Denoising Using Non-Local Means

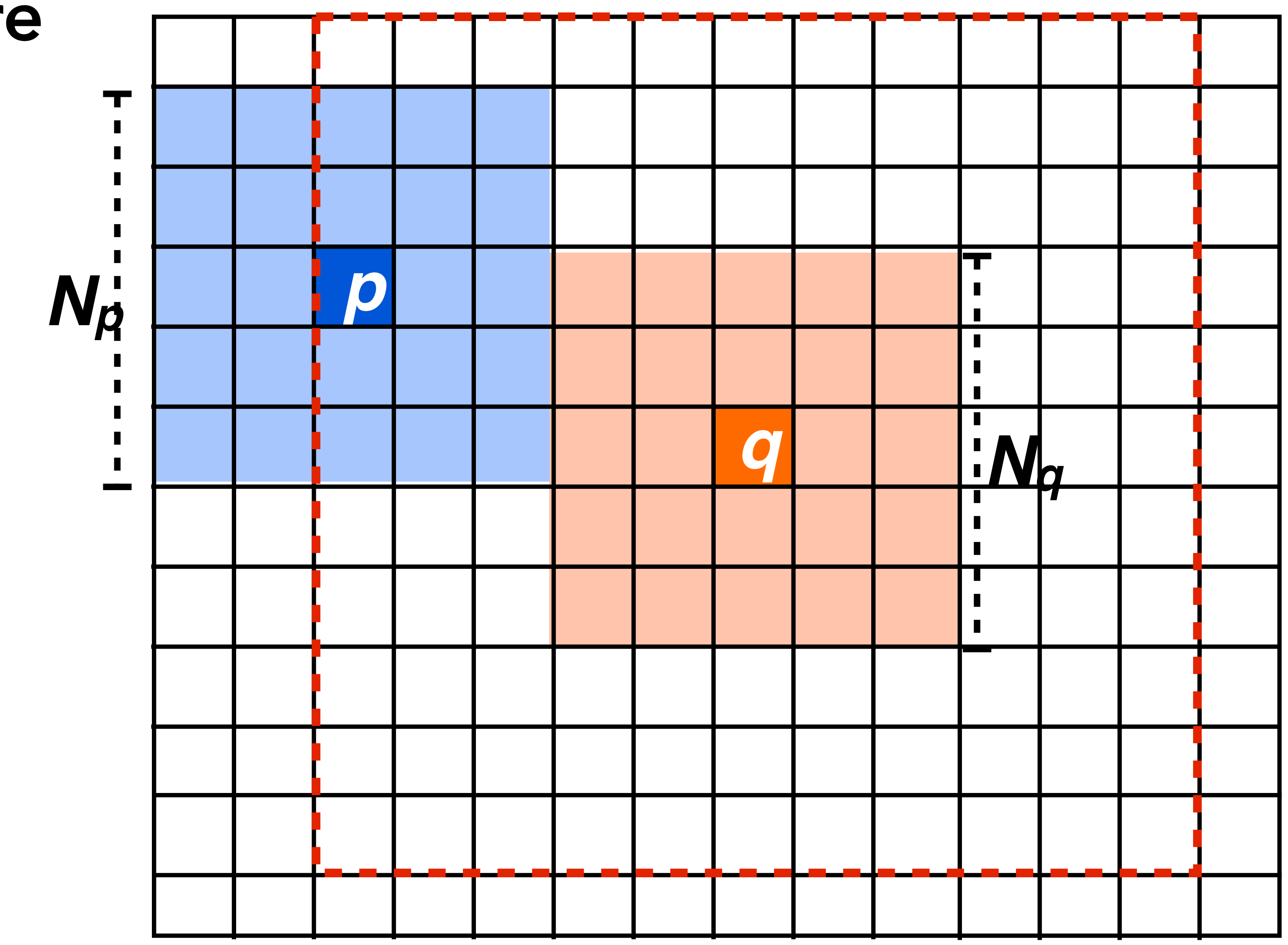
Main idea: replace pixel with average value of nearby pixels that have a similar surrounding region.

- Assumption: images have repeating structure

$$\mathrm{NL}[I](\mathrm{p}) = \sum_{\mathrm{q} \in \mathrm{S}(\mathrm{p})} w(\mathrm{p},\mathrm{q}) I(q)$$

All points in search region about p

$$w(\mathrm{p},\mathrm{q}) = \frac{1}{C_p} e^{-\frac{\| N_p - N_q \|^2}{h^2}}$$



- Np and Nq are vectors of pixel values in square window around pixels p and q (highlighted regions in figure)
- L2 difference between Np and Nq = “similarity” of surrounding regions
- Cp is just a normalization constant to ensure weights sum to one for pixel p.
- S is the search region around p (given by dotted red line in figure)

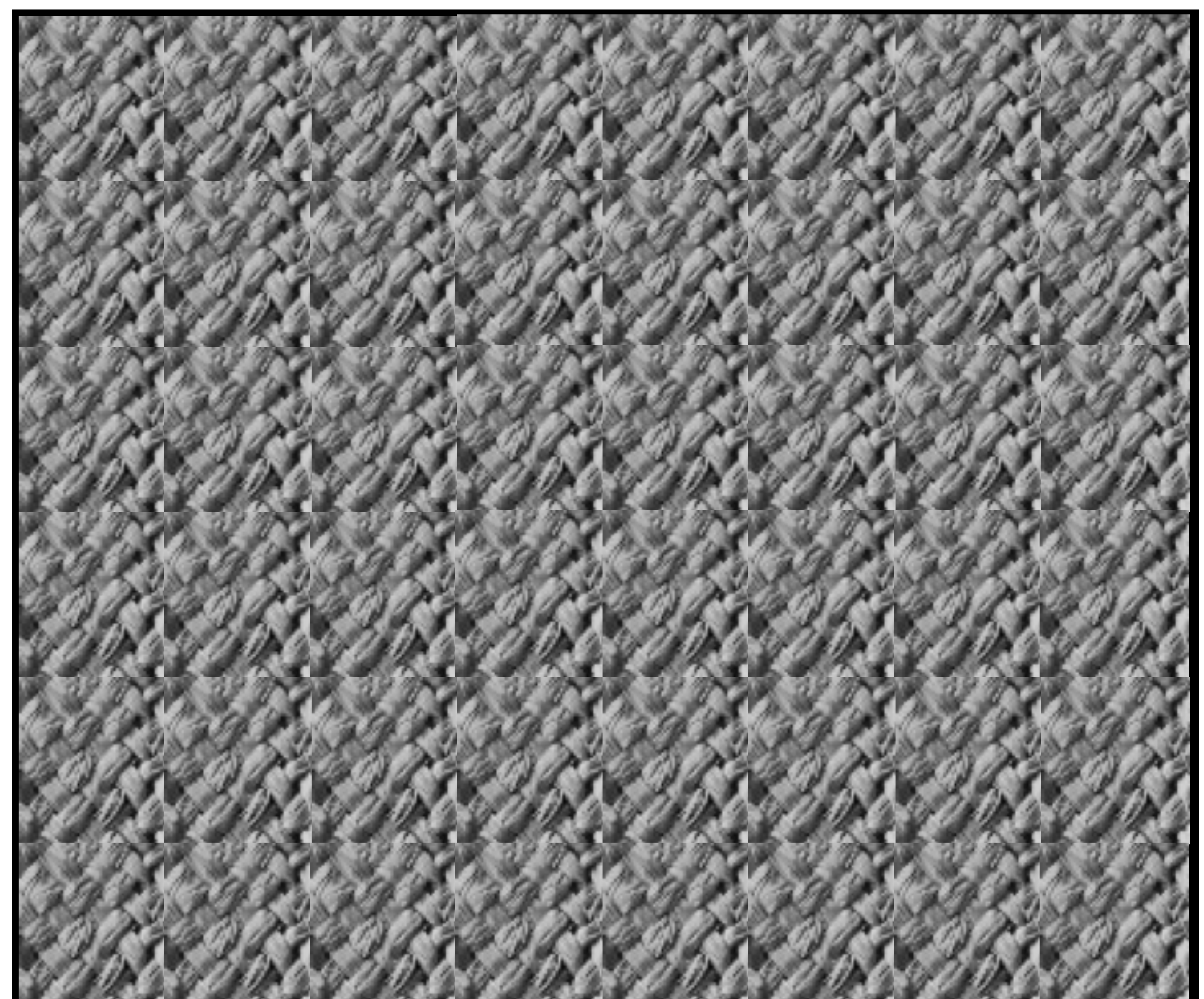
Texture Synthesis

Input: low-resolution texture image

Desired output: high-resolution texture that appears "like" the input

Source texture
(low resolution)

High-resolution texture generated by
naive tiling of low-resolution texture



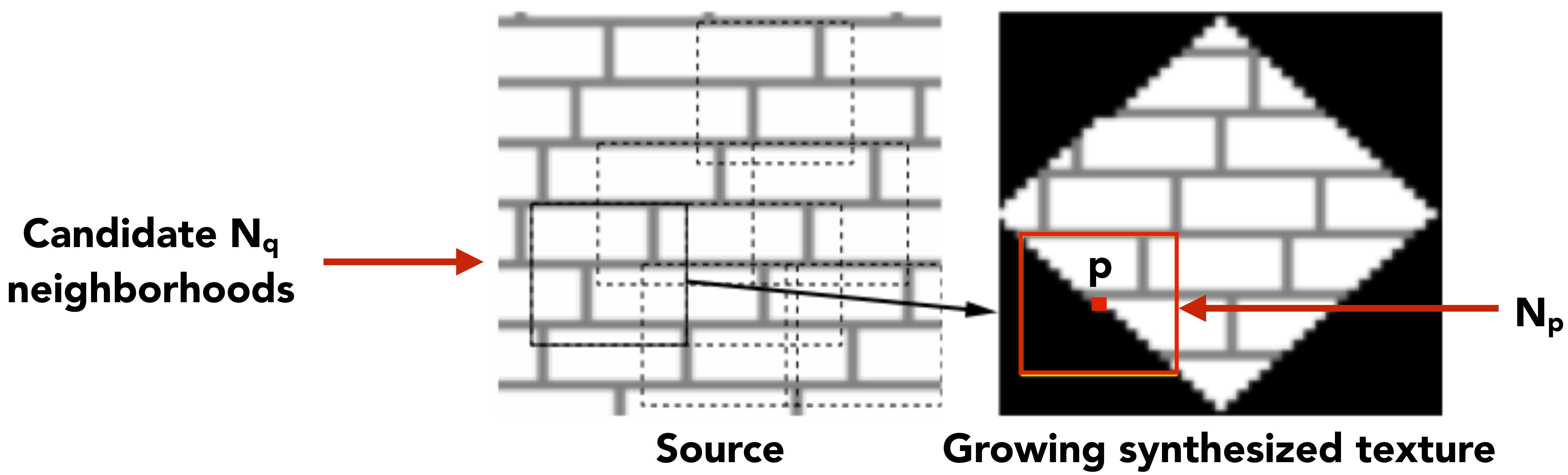
Algorithm: Non-Parametric Texture Synthesis

Main idea: For a given pixel p, find a probability distribution function for possible values of p, based on its neighboring pixels.

Define neighborhood N_p to be the NxN pixels around p

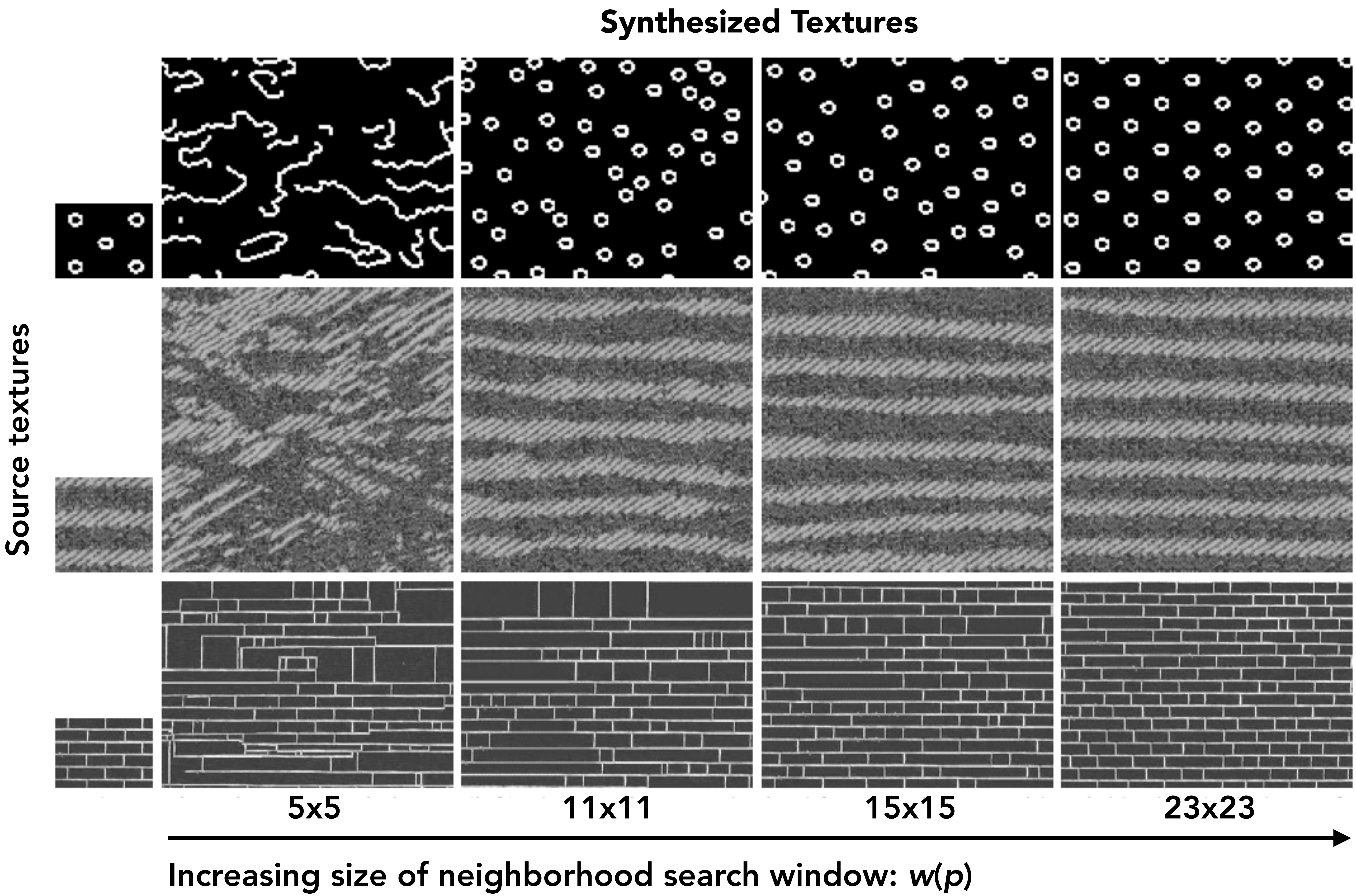
To synthesize each pixel *p*:

1. **Find other *N x N* patches (*N_q*) in the image that are most similar to *N_p***
2. **Center pixels of the closest patches are candidates for *p***
3. **Randomly sample from candidates weighted by distance *$d(N_p,N_q)$***



[Efros and Leung 99]

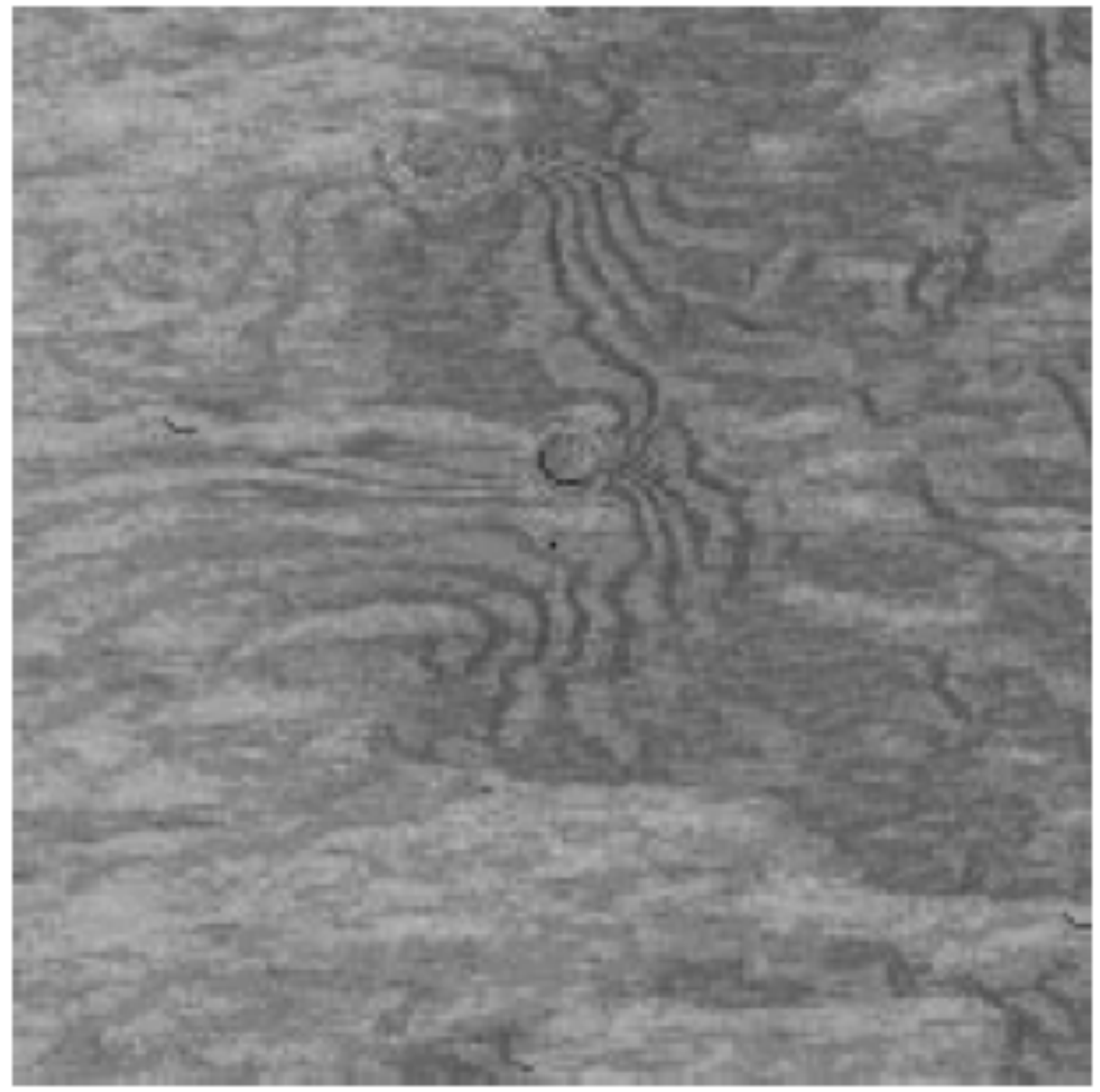
Non-Parametric Texture Synthesis



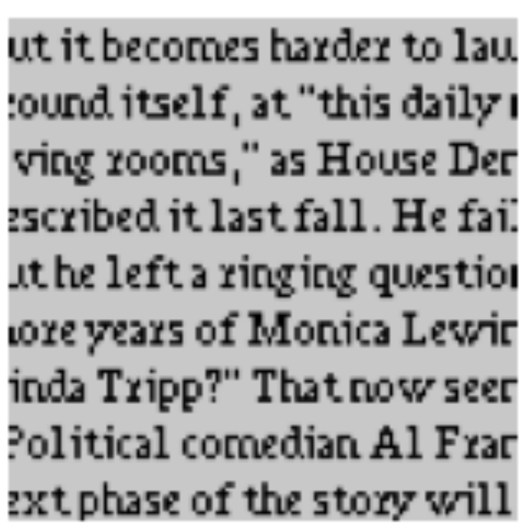
[Efros and Leung 99]

More Texture Synthesis Examples

Source textures | **Synthesized Textures**



Naive tiling solution

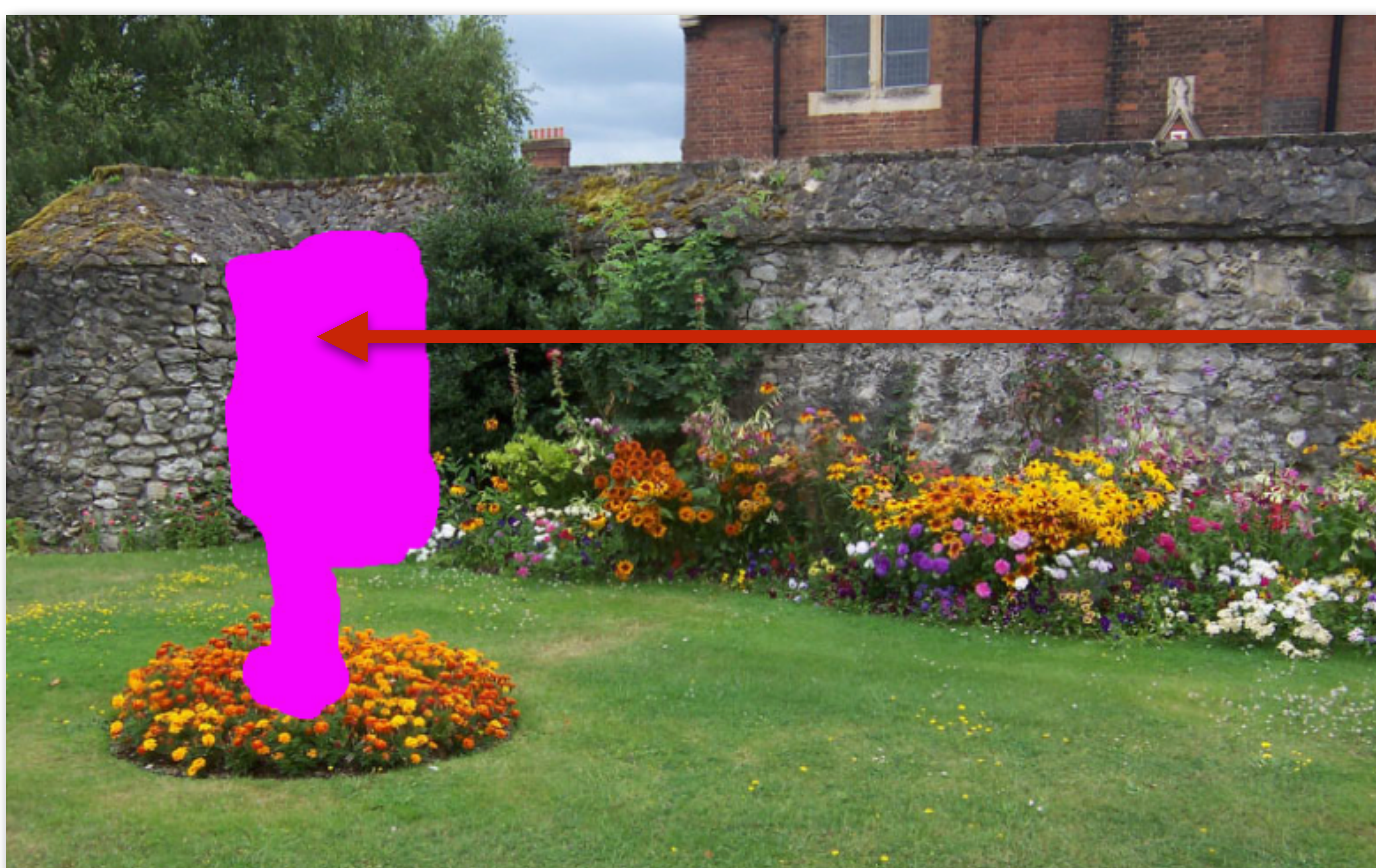


[Efros and Leung 99]

Image Completion Example

Original Image

Completion Result



Masked Region

Goal: fill in masked region with "plausible" pixel values.

See PatchMatch algorithm [Barnes 2009] for a fast randomized algorithm for finding similar patches

Image credit: [Barnes et al. 2009]

Things to Remember

JPEG as an example of exploiting perception in visual systems

- Chroma subsampling and DCT transform

Image processing via convolution

- Different operations by changing filter kernel weights
- Fast separable filter implementation: multiple 1D filters

Data-dependent image processing techniques

- Bilateral filtering, Efros-Leung texture synthesis

To learn more: consider CS194-26 "Computational Photography"

Acknowledgments

Many thanks to Kayvon Fatahalian for this lecture!