Image processing basics

Andy Warhol, *Marilyn Diptych*, 1962 (source)

Many slides adapted from
Alyosha Efros, Derek Hoiem

Image processing basics: Outline

- Images as sampled functions
- Sampling and reconstruction, aliasing
- Image resampling, interpolation
- Image transformations

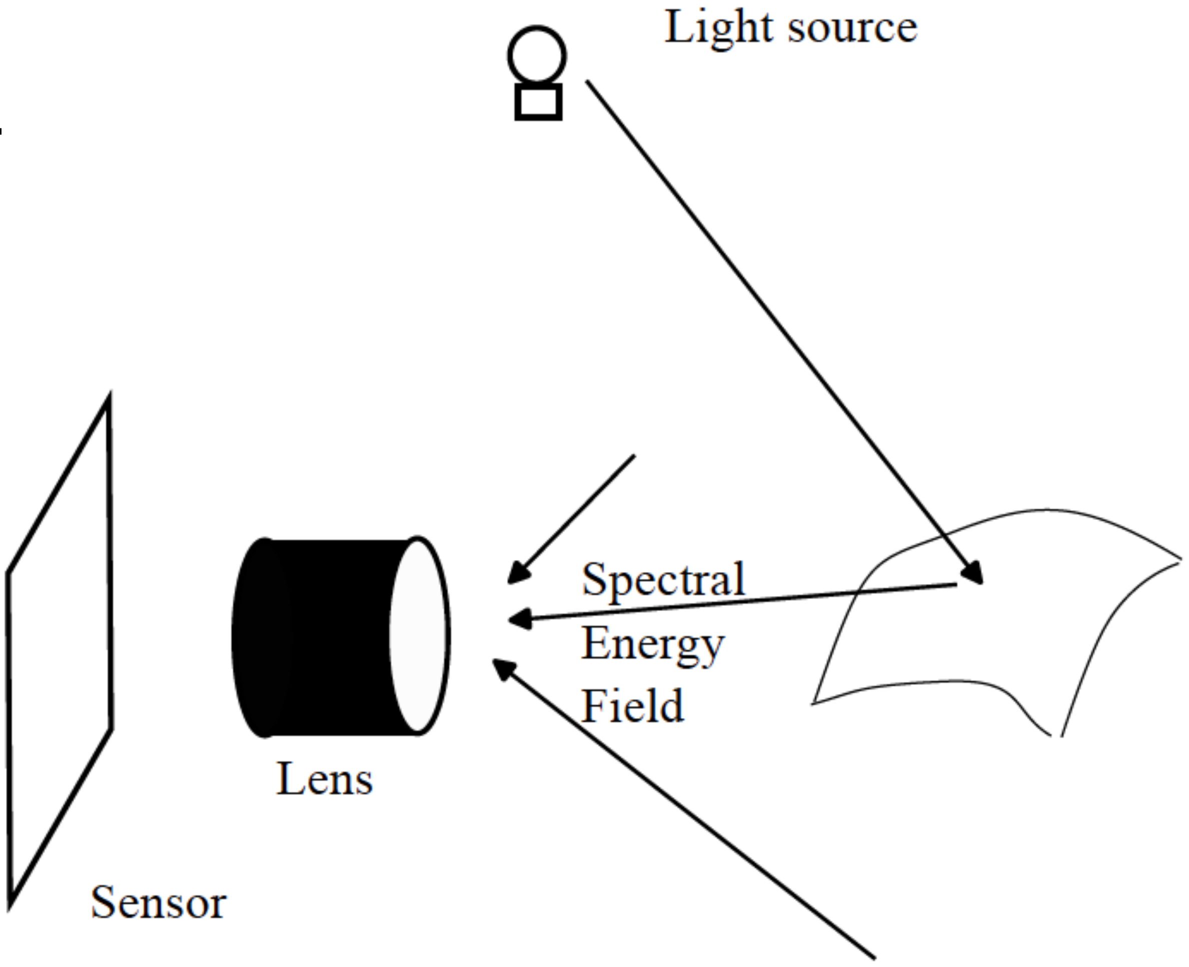
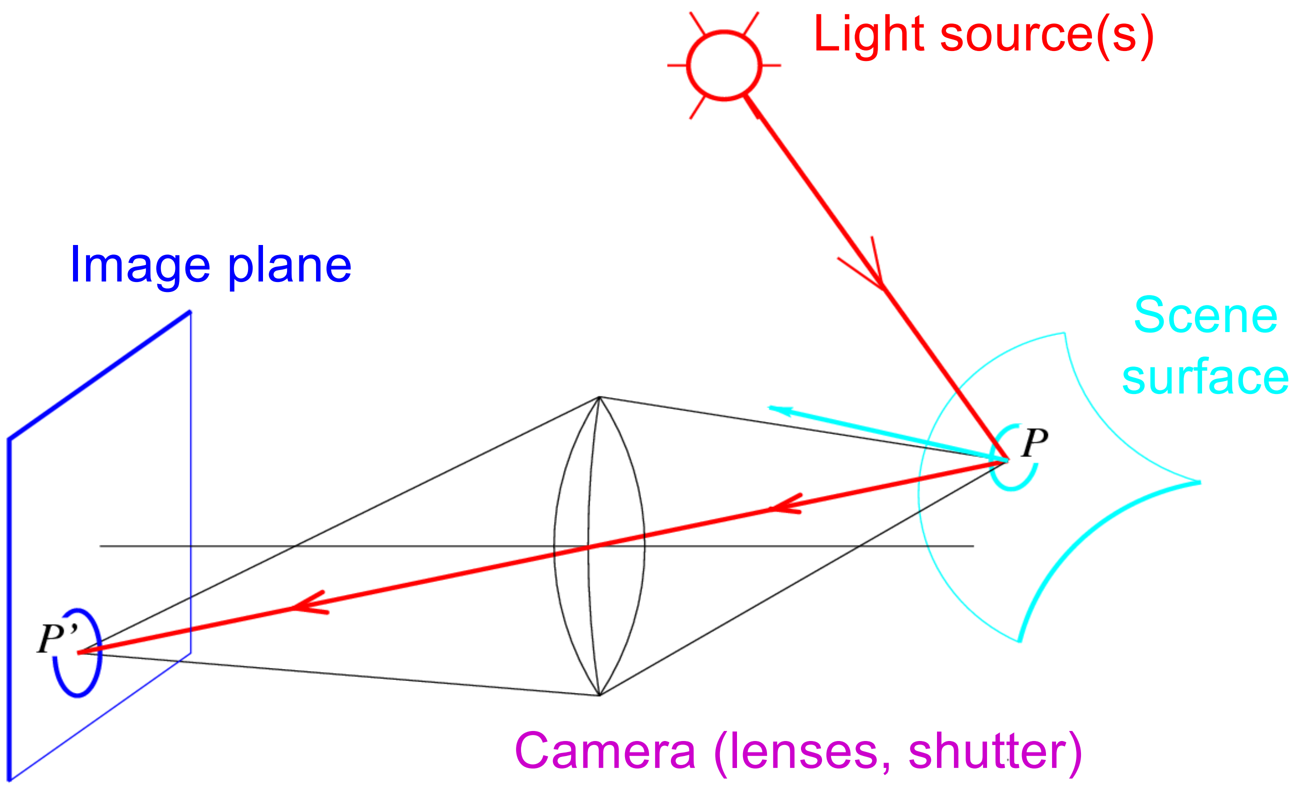


FIGURE 2.1: *A high-level model of imaging. Light leaves light sources and reflects from surfaces. Eventually, some light arrives at a camera and enters a lens system. Some of that light arrives at a photosensor inside the camera.*

Image formation (preview)

- What determines the brightness of an image pixel?



Light energy arriving at P' per unit time is quantified by ***irradiance*** (Watts per sq. meter)

Strictly speaking, we need ***spectral irradiance** to account for light across different wavelengths*

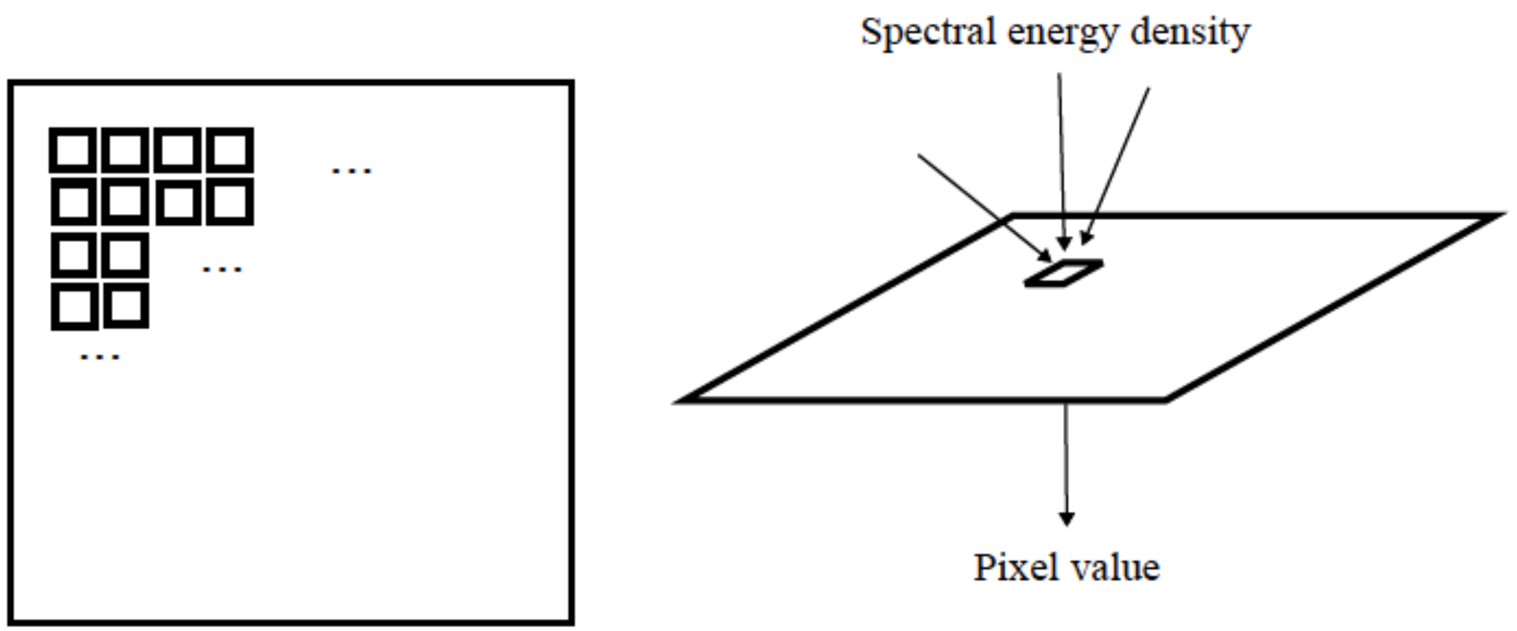
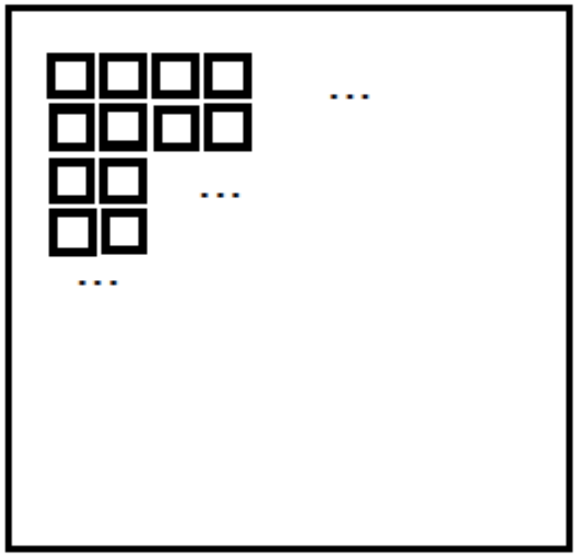


FIGURE 2.2: *The photosensor is divided into a grid of pixels, which are small sensitive locations. Each pixel receives an incoming spectral energy field, and turns it into a number. This number is typically a weighted average over a position in the sensor, a very small range of incoming directions and a large range of wavelengths. Each pixel is at a different position in the sensor, and the lens system and camera geometry ensure that each sees a different set of incoming directions, so that the averages produce a coherent image.*



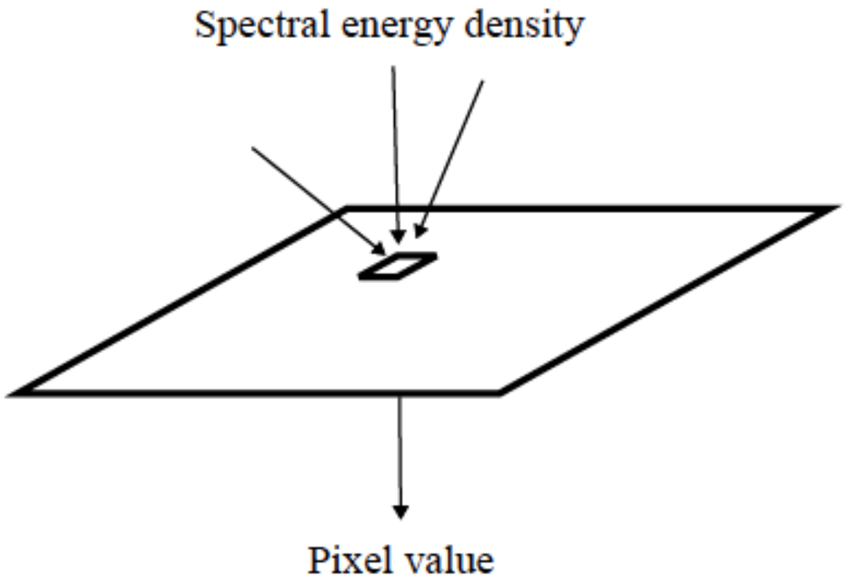


FIGURE 2.2: *The photosensor is divided int*
sitive locations. Each pixel receives an inc
into a number. This number is typically a
sensor, a very small range of incoming dire
Each pixel is at a different position in the
geometry ensure that each sees a different
averages produce a coherent image.

$$p_{ij} = k\int_{\mathcal{P}}\int_{\mathcal{W}}\int_{\Lambda} E(\mathbf{u},\omega,\lambda)w(\mathbf{u},\omega,\lambda)d\mathbf{u}d\omega d\lambda$$

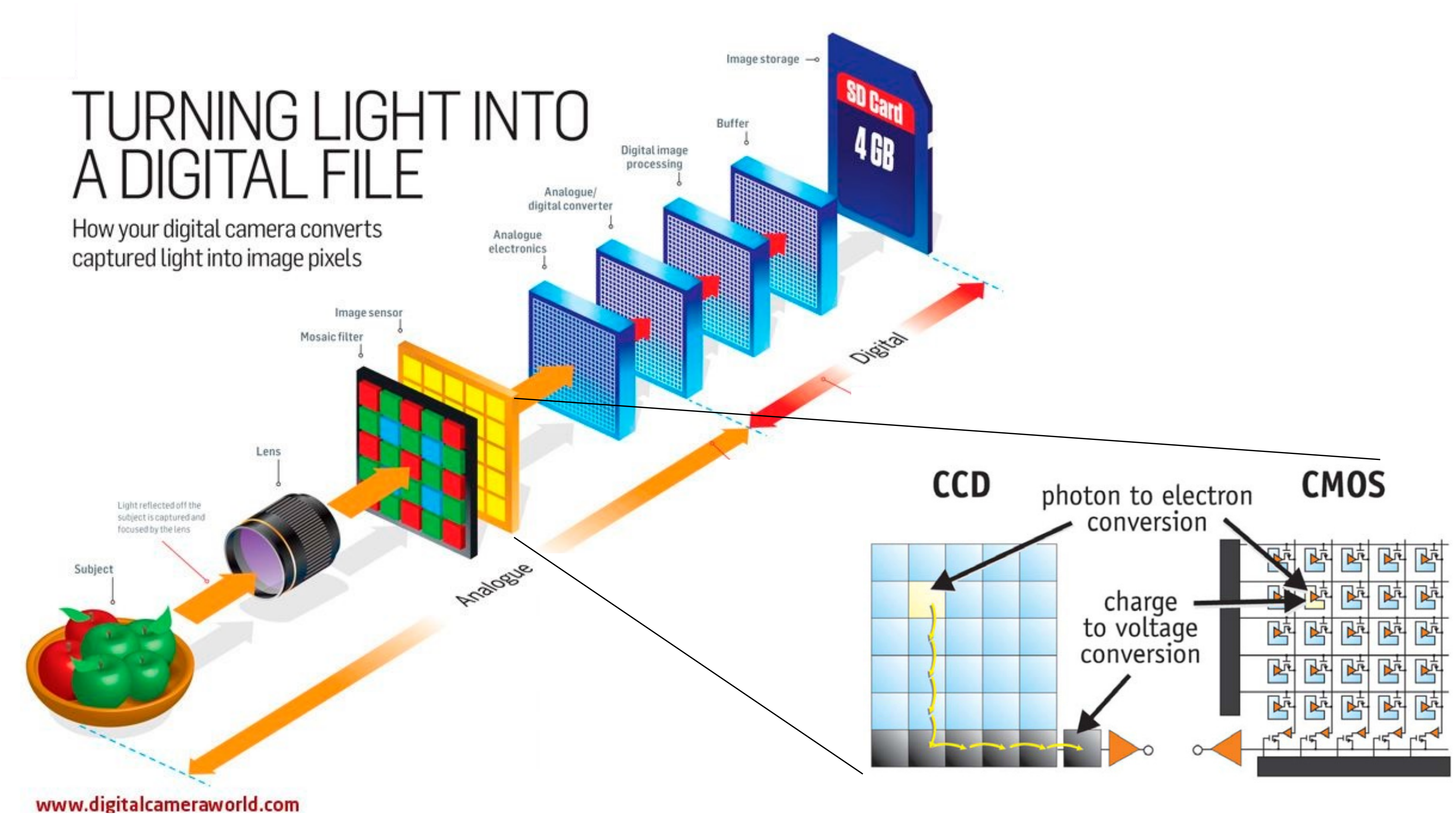
here w is the weight function or sensitivity of the sensor

$$= k\int_{\Lambda} E([i\Delta x, j\Delta y, 0], \omega, \lambda)w(\lambda)d\mathbf{u}d\lambda$$

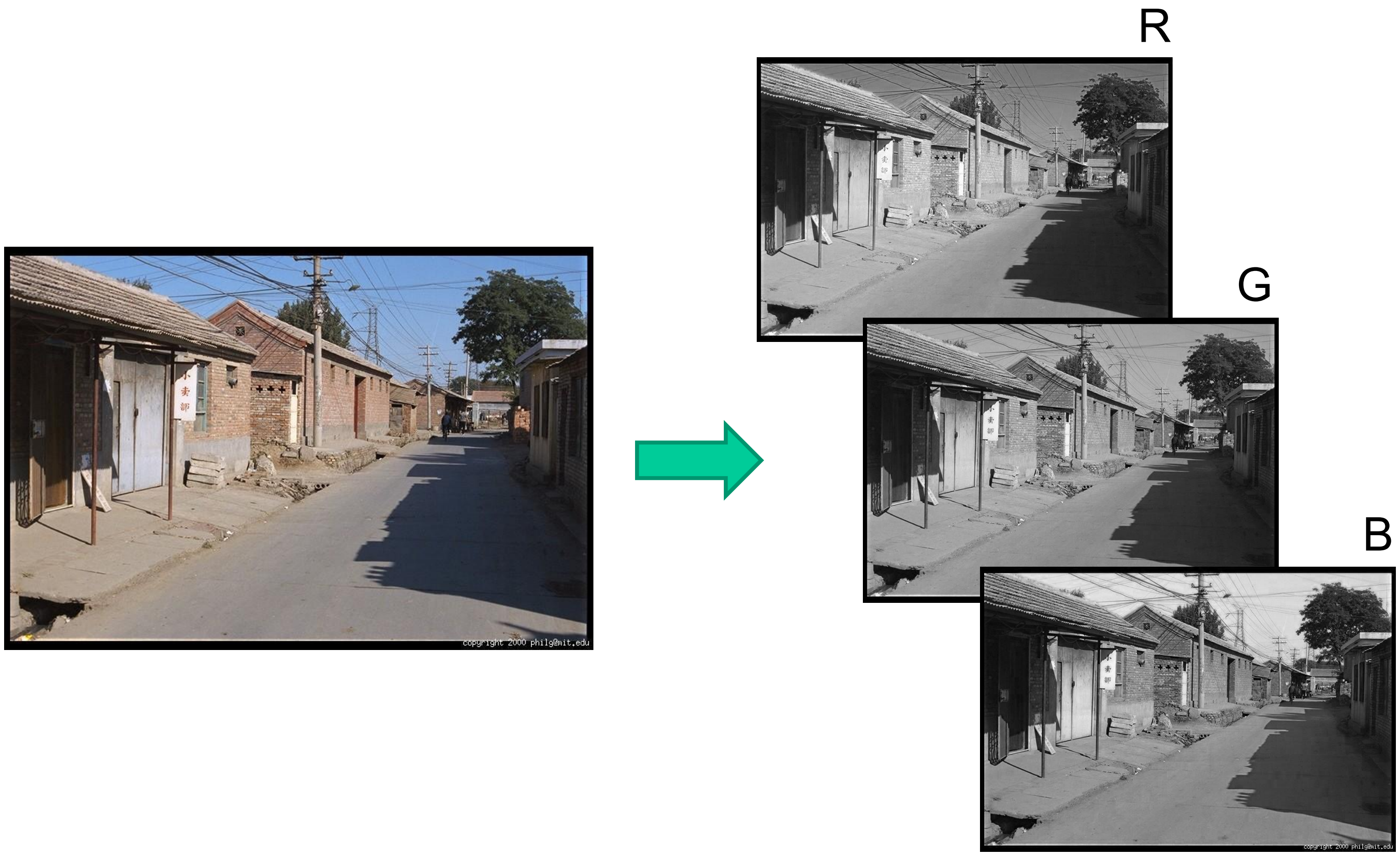
because the averages are over very small ranges

$$= k\Phi(i\Delta x, j\Delta y, 0)$$

Images as sampled functions


TURNING LIGHT INTO A DIGITAL FILE
How your digital camera converts captured light into image pixels
Image storage
SD Card
4 GB
Buffer
Digital image processing
Analogue/ digital converter
Analogue electronics
Image sensor
Mosaic filter
Lens
Light reflected off the subject is captured and focused by the lens
Subject
Digital
Analogue
www.digitalcameraworld.com
CCD
photon to electron conversion
CMOS
charge to voltage conversion

Digital color image



Images in Python

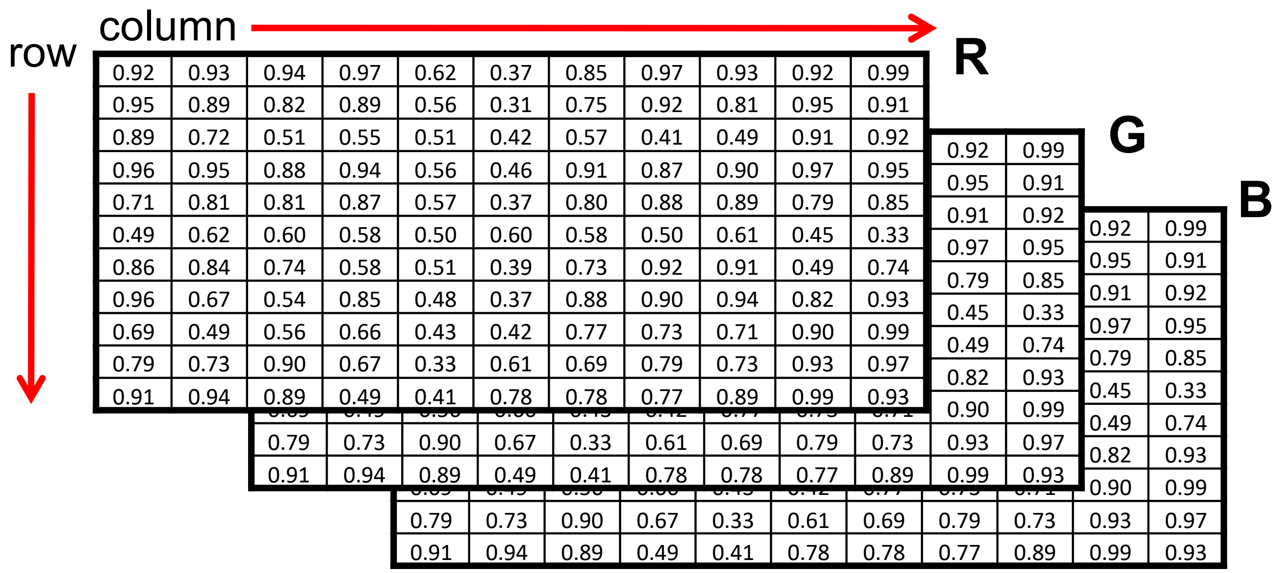
```
im = cv2.imread(filename)                  # read image
im = cv2.cvtColor(im, cv2.COLOR_BGR2RGB) # order channels as RGB
im = im / 255                              # values range from 0 to 1
```

RGB image `im` is a H x W x 3 matrix (numpy.ndarray)

`im[0,0,0]` is the top-left pixel value in R-channel

`im[y, x, c]` is the value y+1 pixels down, x+1 pixels to right in the cth channel

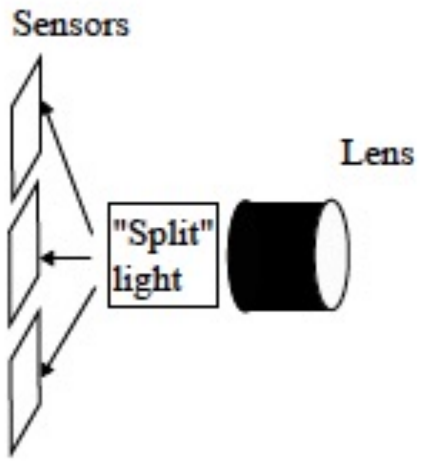
`im[H-1, W-1, 2]` is the bottom-right pixel in B-channel

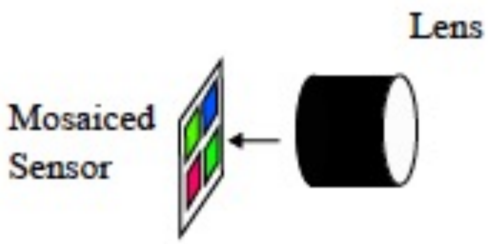


0.92	0.93	0.94	0.97	0.62	0.37	0.85	0.97	0.93	0.92	0.99
0.95	0.89	0.82	0.89	0.56	0.31	0.75	0.92	0.81	0.95	0.91
0.89	0.72	0.51	0.55	0.51	0.42	0.57	0.41	0.49	0.91	0.92
0.96	0.95	0.88	0.94	0.56	0.46	0.91	0.87	0.90	0.97	0.95
0.71	0.81	0.81	0.87	0.57	0.37	0.80	0.88	0.89	0.79	0.85
0.49	0.62	0.60	0.58	0.50	0.60	0.58	0.50	0.61	0.45	0.33
0.86	0.84	0.74	0.58	0.51	0.39	0.73	0.92	0.91	0.49	0.74
0.96	0.67	0.54	0.85	0.48	0.37	0.88	0.90	0.94	0.82	0.93
0.69	0.49	0.56	0.66	0.43	0.42	0.77	0.73	0.71	0.90	0.99
0.79	0.73	0.90	0.67	0.33	0.61	0.69	0.79	0.73	0.93	0.97
0.91	0.94	0.89	0.49	0.41	0.78	0.78	0.77	0.89	0.99	0.93

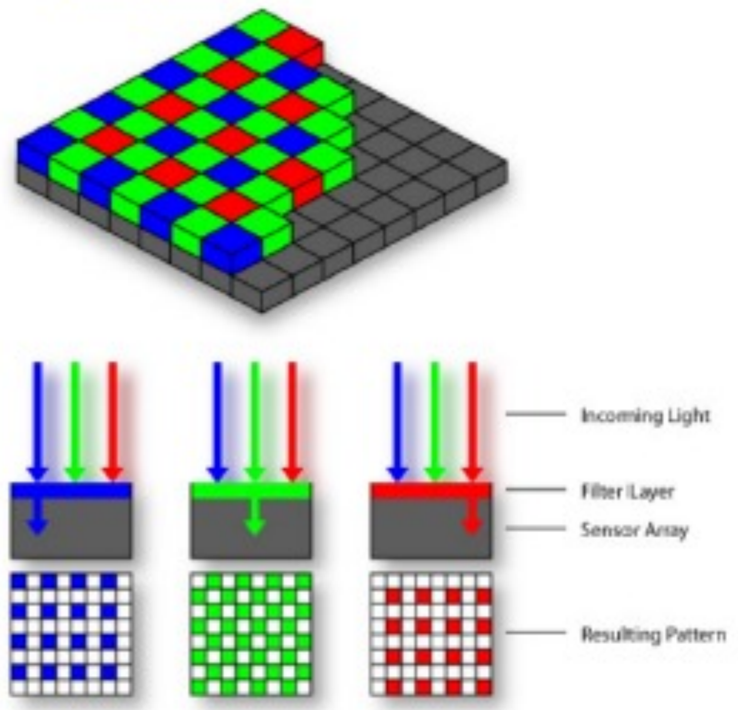
How are the three color channels acquired?

Acquiring 3 channels



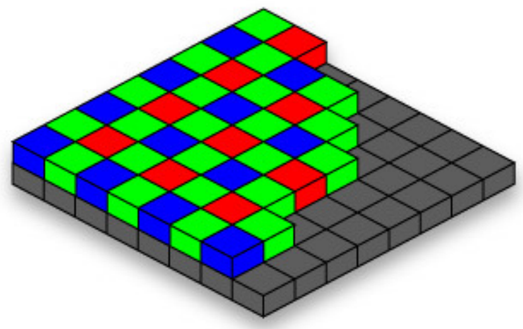


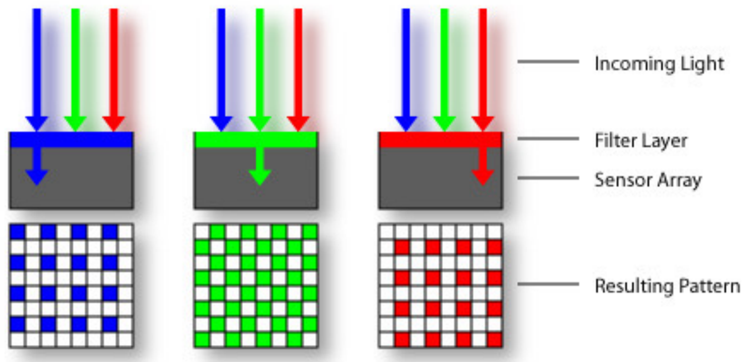
Bayer grid (1976)



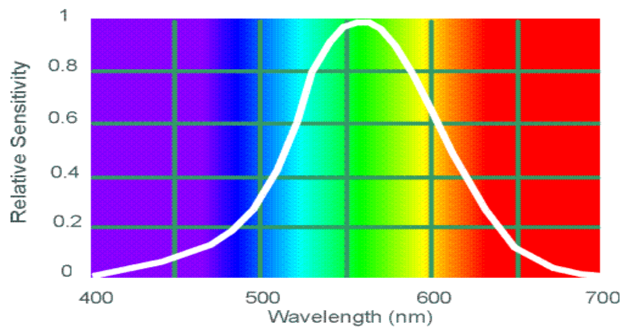
How are the three channels acquired?

Bayer grid (1976)





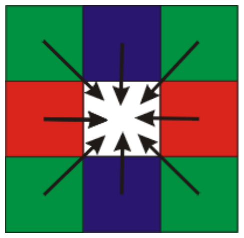
Why more green?



Human Luminance Sensitivity Function

Demosaicing:

Estimation of missing components from neighboring values

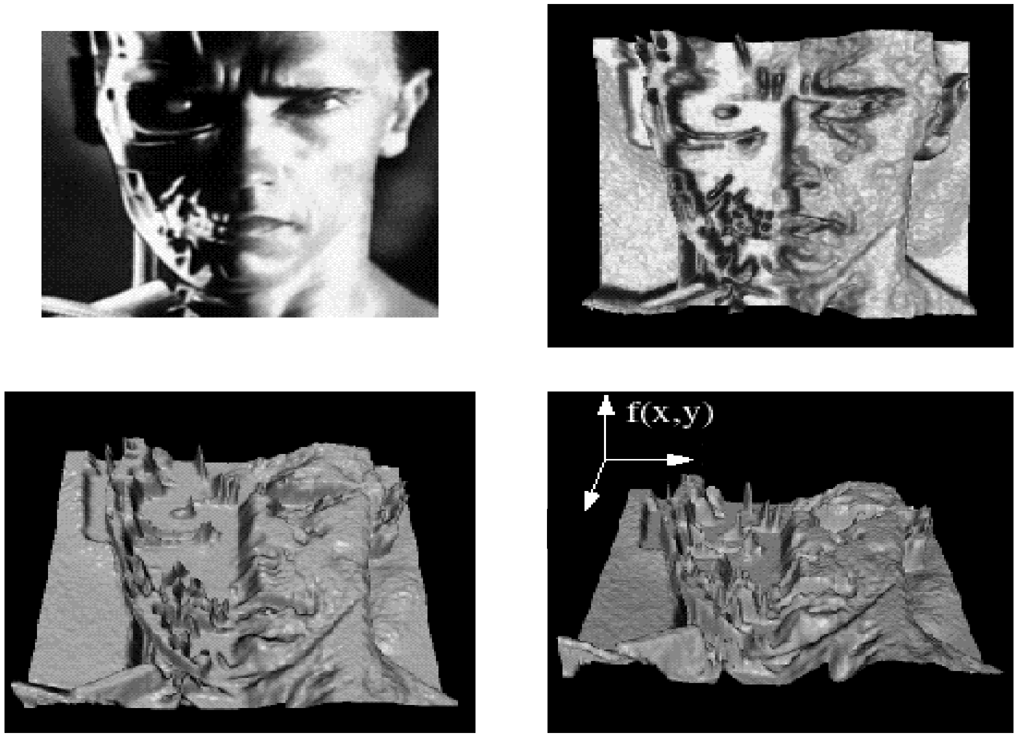


HOW? - shortly!

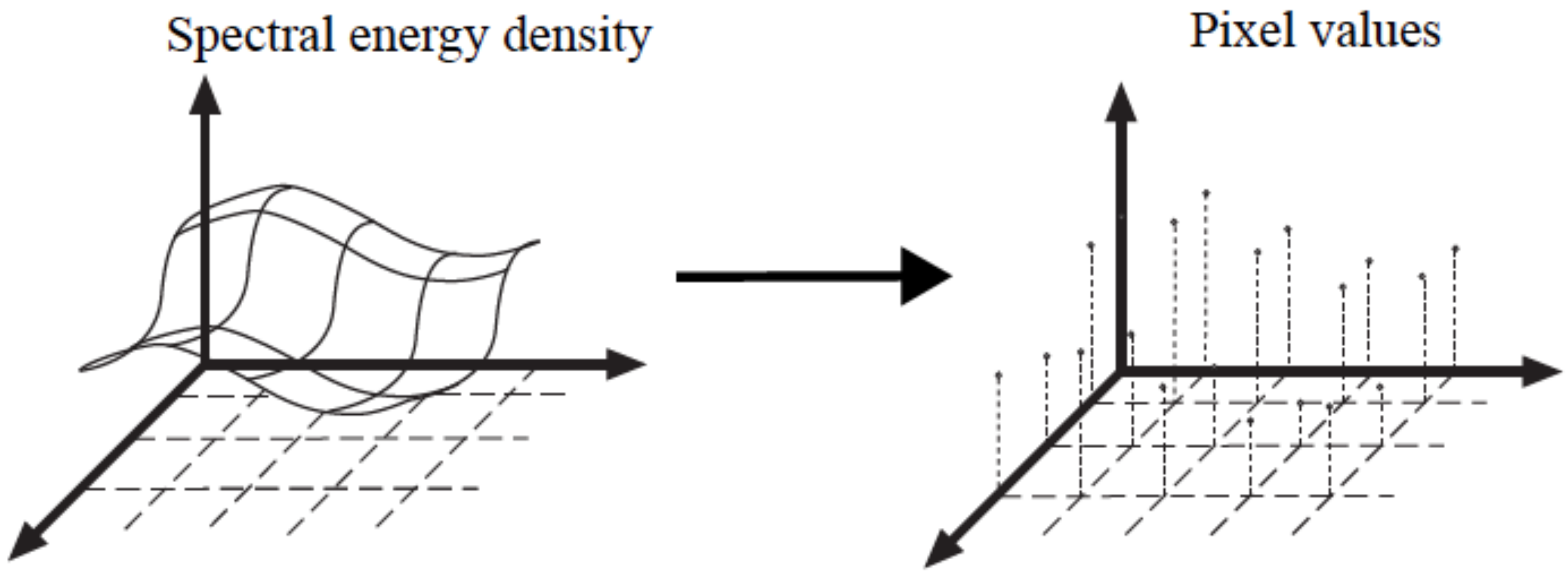
Source: Wikipedia

Images as sampled functions

- We like to think of a digital image as a sampled representation of a continuous function $f(x, y)$ defined over a continuous 2D domain

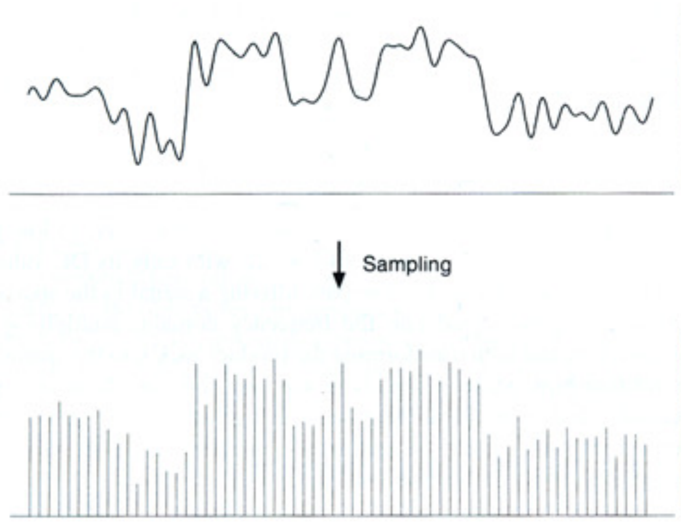


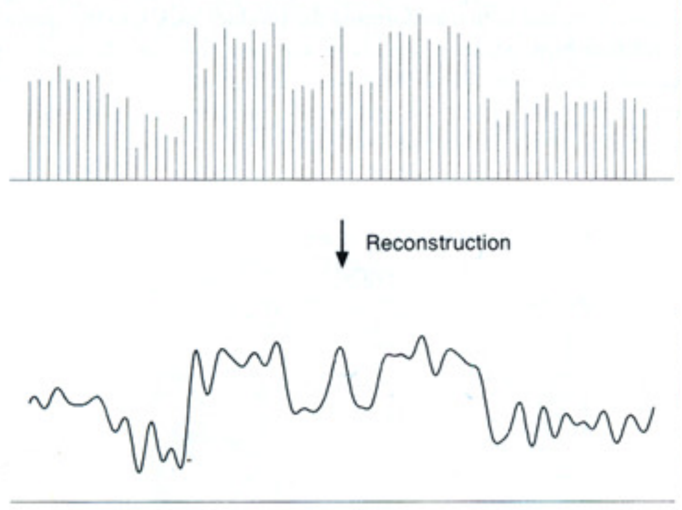
Sampling



Sampling and reconstruction

- Sampling: recording the function's values at a discrete set of locations
- Reconstruction: converting a sampled representation back into a continuous function by "guessing" what happens between the samples

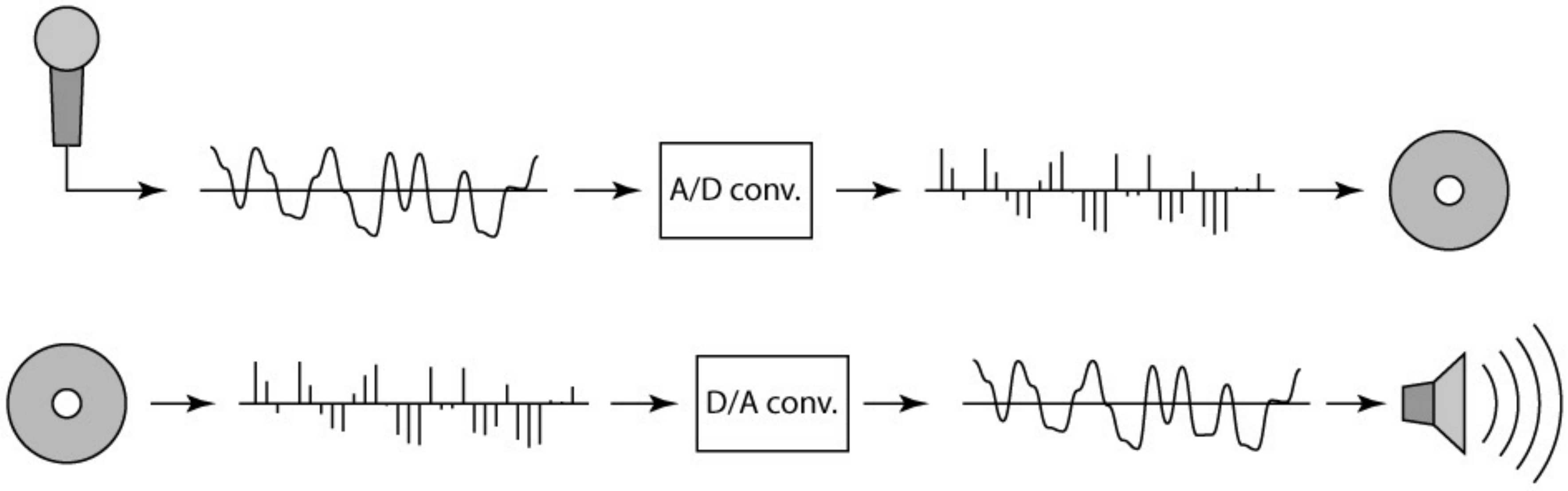




Source: S. Marschner (via A. Efros)

1D example: Digital audio

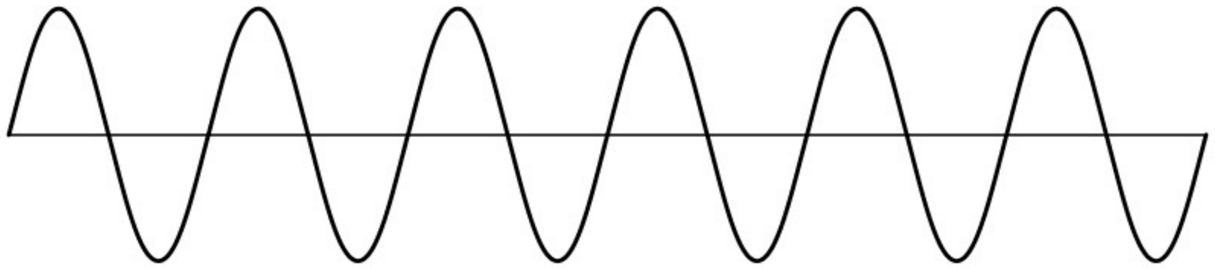
- Recording: sound to analog to samples to disc
- Playback: disc to samples to analog to sound again



Source: S. Marschner (via A. Efros)

Sampling and reconstruction

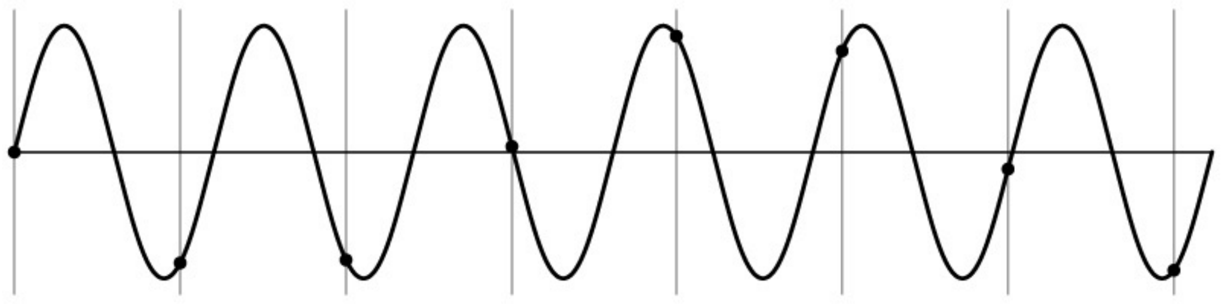
- Simple example: a sine wave



Source: S. Marschner (via A. Efros)

Sampling and reconstruction

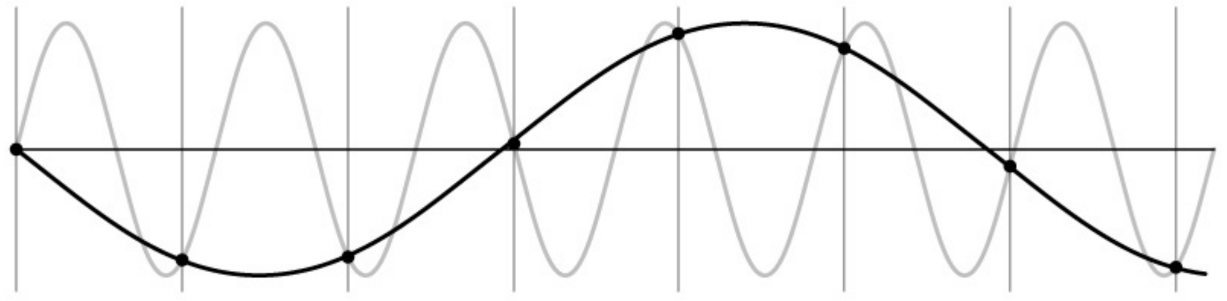
- Simple example: a sine wave
- What if we “missed” things between the samples?
    - Unsurprising result: information is lost



Source: S. Marschner (via A. Efros)

Sampling and reconstruction

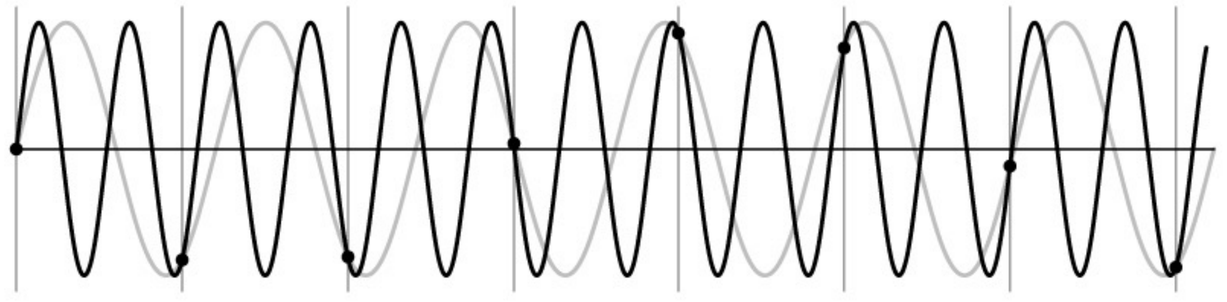
- Simple example: a sine wave
- What if we “missed” things between the samples?
  - Unsurprising result: information is lost
  - Surprising result: indistinguishable from lower frequencies



Source: S. Marschner (via A. Efros)

Sampling and reconstruction

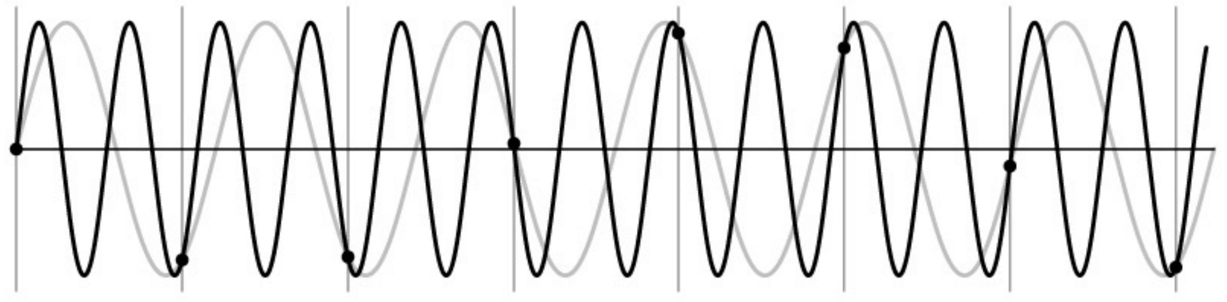
- Simple example: a sine wave
- What if we “missed” things between the samples?
  - Unsurprising result: information is lost
  - Surprising result: indistinguishable from lower frequencies (or even higher frequencies)



Source: S. Marschner (via A. Efros)

Sampling and reconstruction

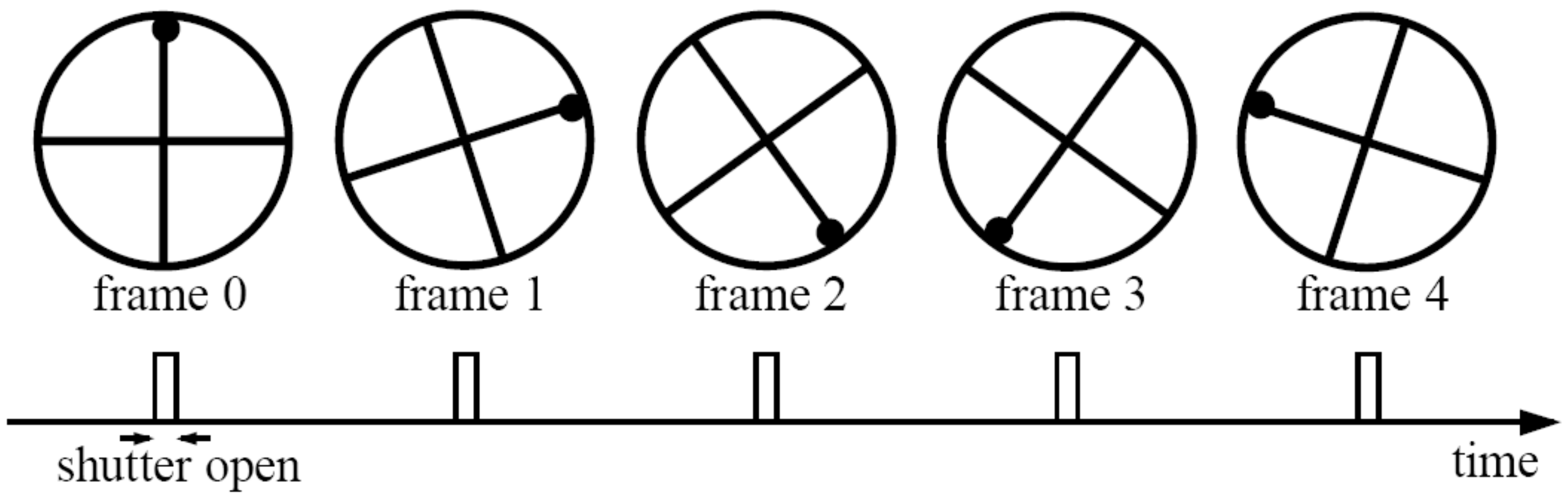
- Simple example: a sine wave
- What if we "missed" things between the samples?
  - Unsurprising result: information is lost
  - Surprising result: indistinguishable from lower frequencies (or even higher frequencies)
  - *Aliasing*: signal "traveling in disguise" as other frequencies



Source: S. Marschner (via A. Efros)

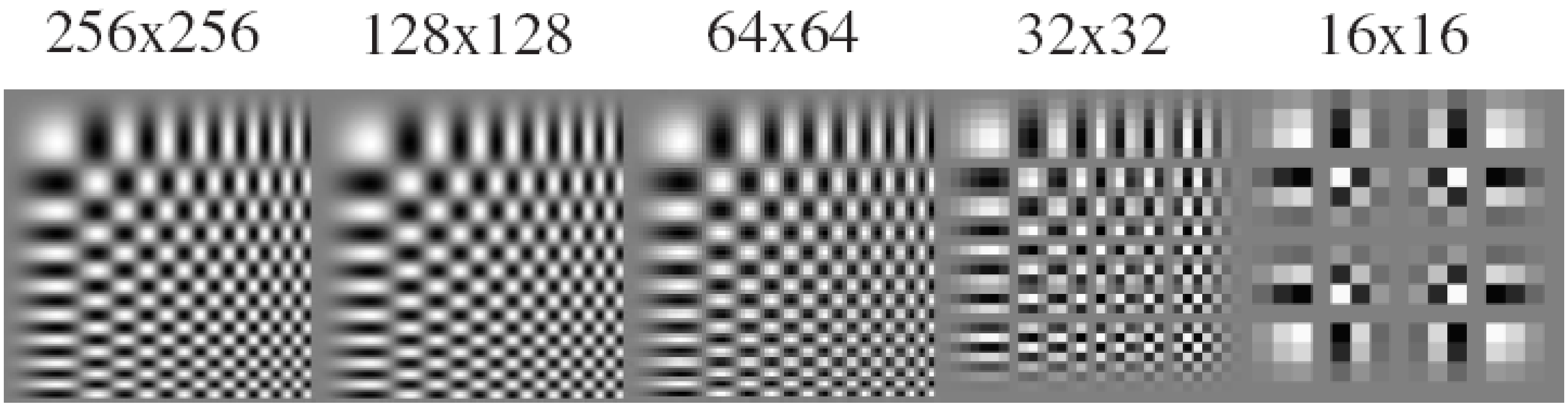
Wagon wheel effect

- Without dot, wheel appears to be slowly rotating backwards!



Source: S. Seitz (via A. Efros)

Aliasing in images



What’s happening?

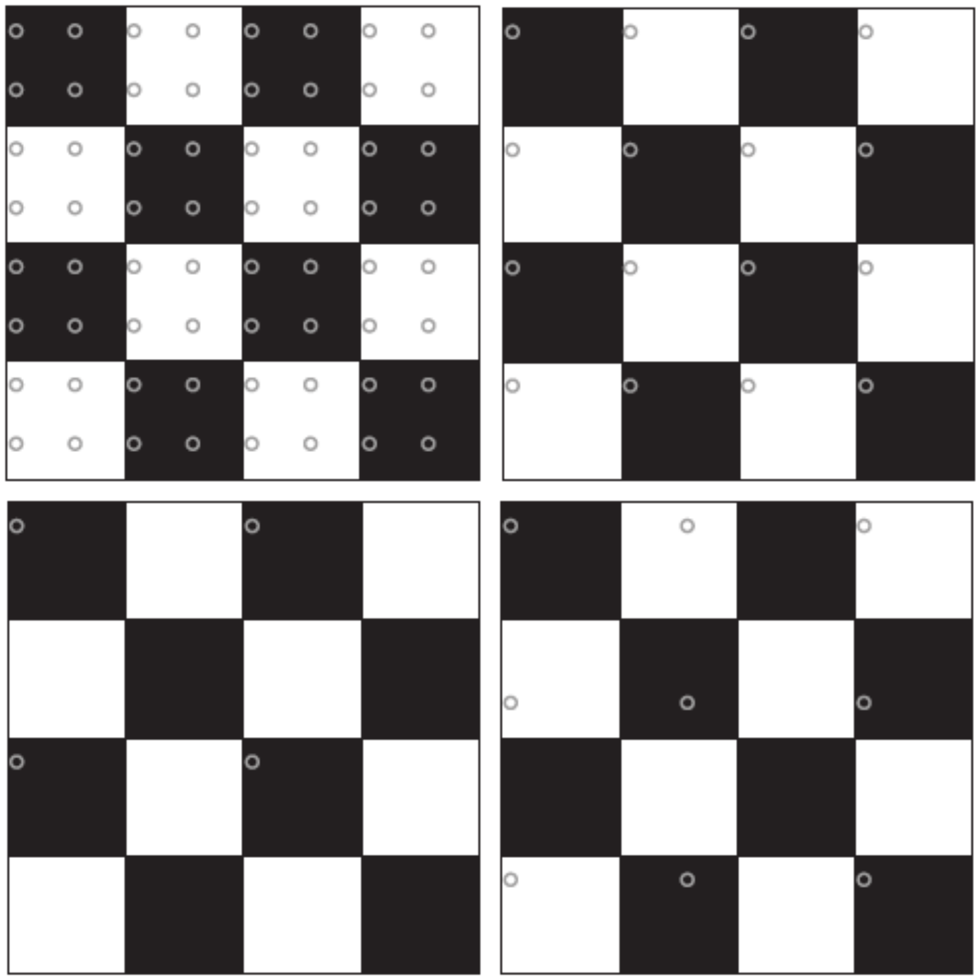
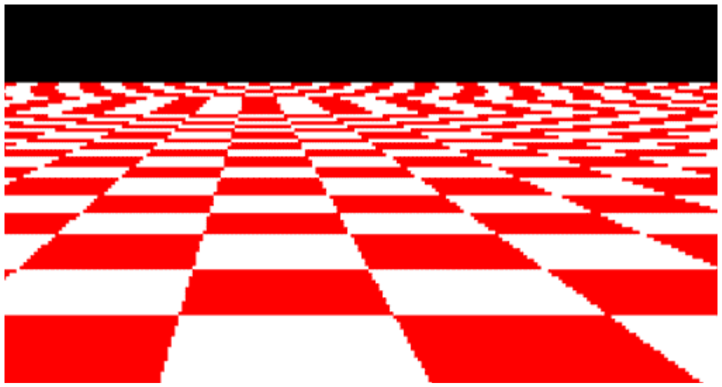
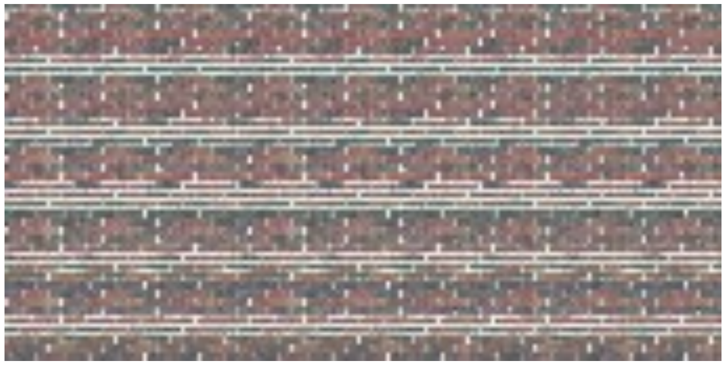


FIGURE 4.3: *The two checkerboards on the* **top** *illustrate a sampling procedure that appears to be successful (whether it is or not depends on some details that we will deal with later). The gray circles represent the samples; if there are sufficient samples, then the samples represent the detail in the underlying function. The sampling procedures shown on the* **bottom** *are unequivocally unsuccessful; the samples suggest that there are fewer checks than there are. This illustrates two important phenomena: first, successful sampling schemes sample data often enough; and second, unsuccessful sampling schemes cause high-frequency information to appear as lower-frequency information.*

Aliasing “in the wild”

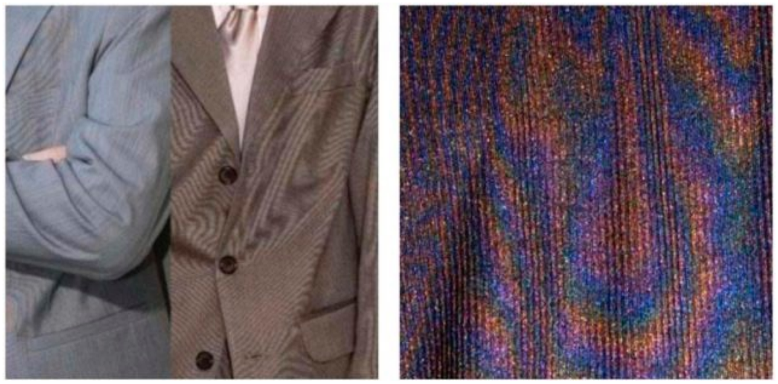
Disintegrating textures



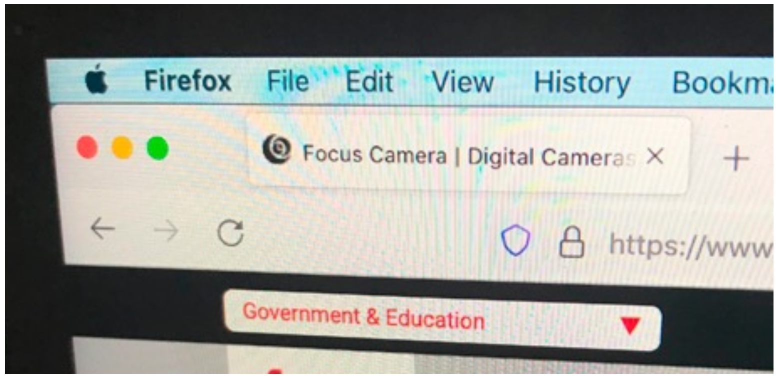


Source

Moire patterns, false color



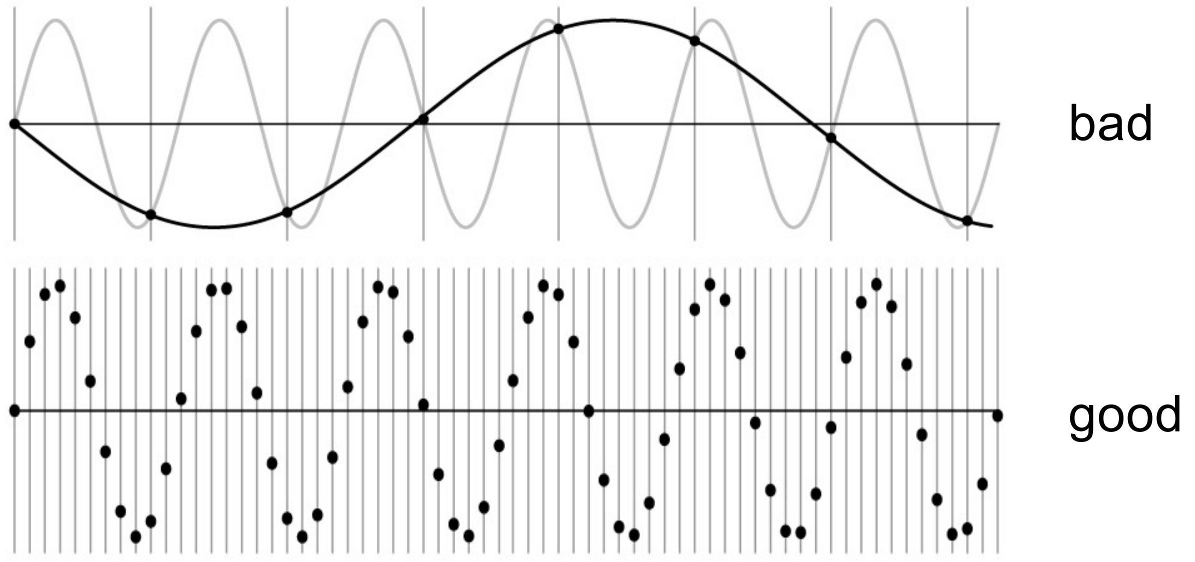
Source



Source

Nyquist-Shannon sampling theorem

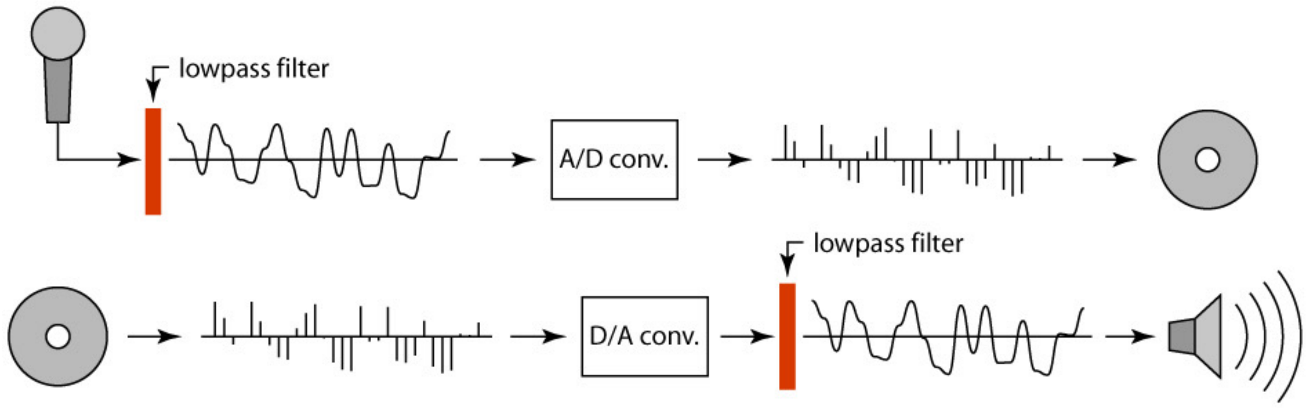
- When sampling a signal at discrete intervals, the sampling frequency must be at least *twice* the maximum frequency of the input signal to allow us to reconstruct the original perfectly from the sampled version



https://en.wikipedia.org/wiki/Nyquist-Shannon_sampling_theorem

Anti-aliasing

- What are possible solutions?
  - Sample more often (if you can)
  - Get rid of all frequencies that are greater than half the new sampling frequency
    - Will lose information, but that's better than aliasing
    - How to get rid of high frequencies?
      - Apply a smoothing or *low-pass* filter (later)



Why should you care about anti-aliasing?

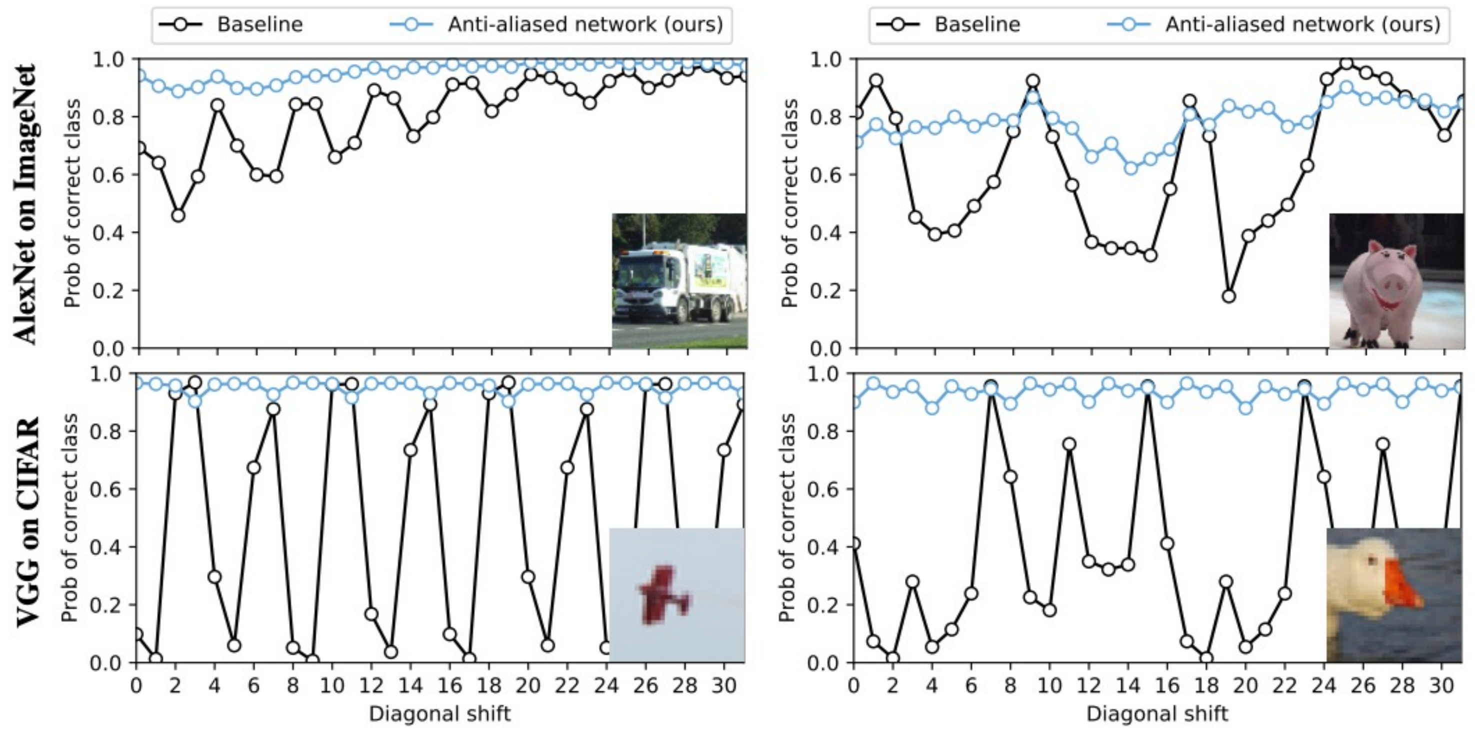


Figure 1. **Classification stability for selected images.** Predicted probability of the correct class changes when shifting the image. The baseline (black) exhibits chaotic behavior, which is stabilized by our method (blue). We find this behavior across networks and datasets. Here, we show selected examples using AlexNet on ImageNet **(top)** and VGG on CIFAR10 **(bottom)**. Code and anti-aliased versions of popular networks are available at `https://richzhang.github.io/antialiased-cnns/`.

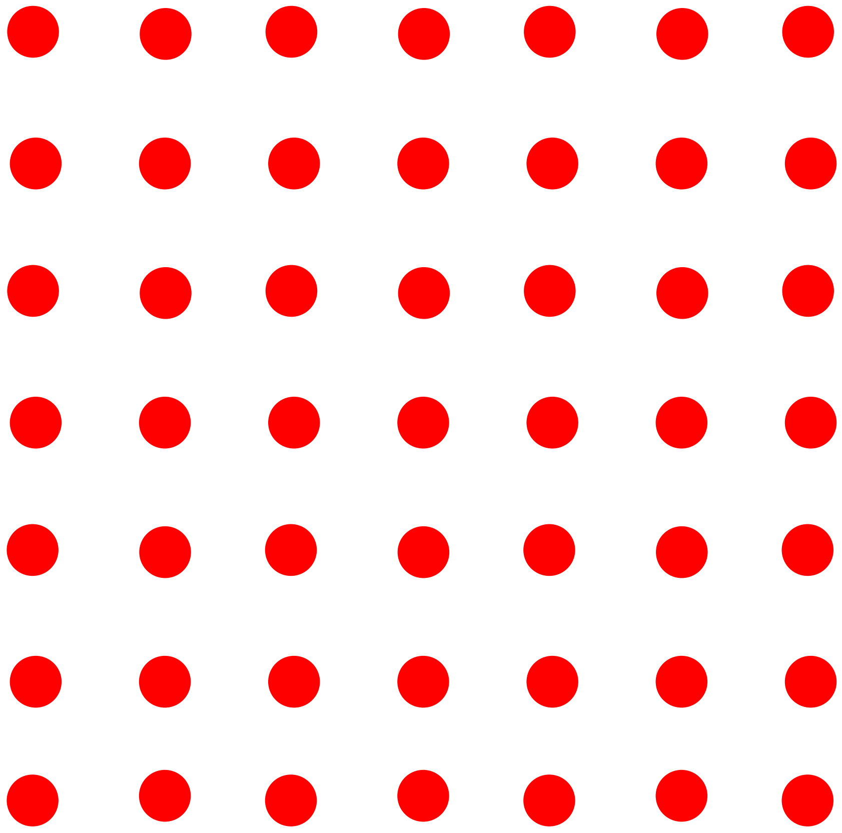
R. Zhang. Making convolutional networks shift-invariant again. ICML 2019

Image processing basics: Outline

- Images as sampled functions
- Sampling and reconstruction, aliasing
- Image resampling, interpolation

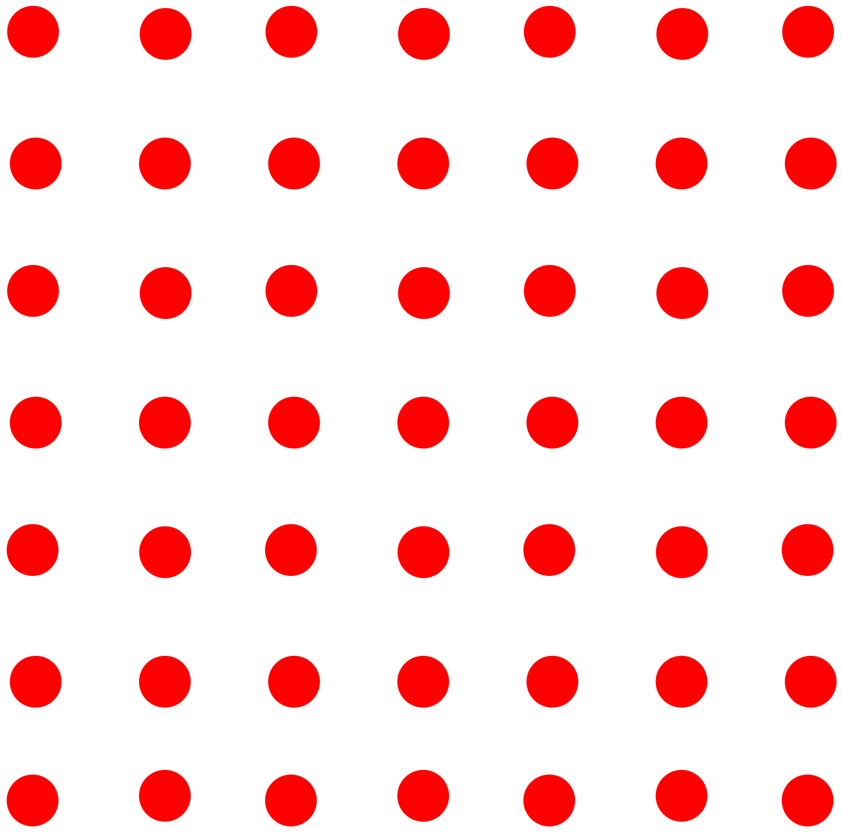
Subsampling an image

- How do we reduce the size of an image by a factor of two?



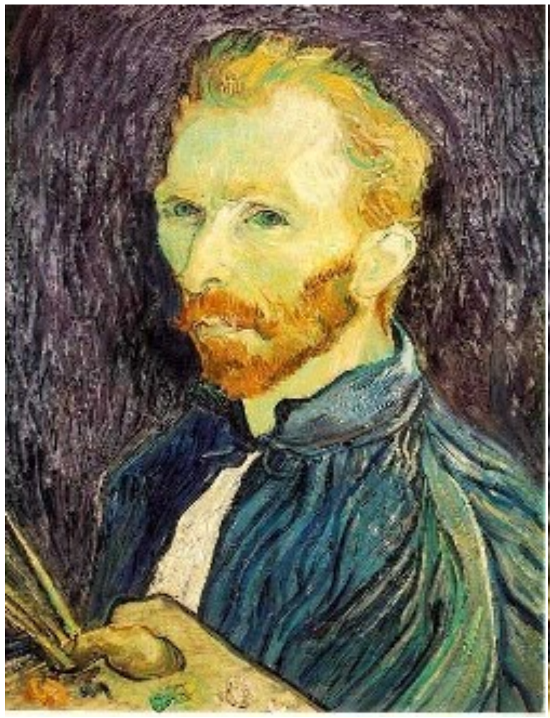
Subsampling an image

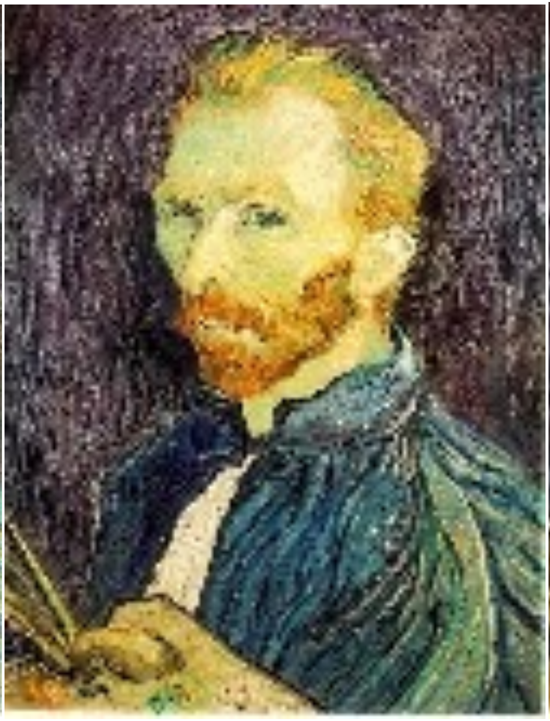
- How do we reduce the size of an image by a factor of two?

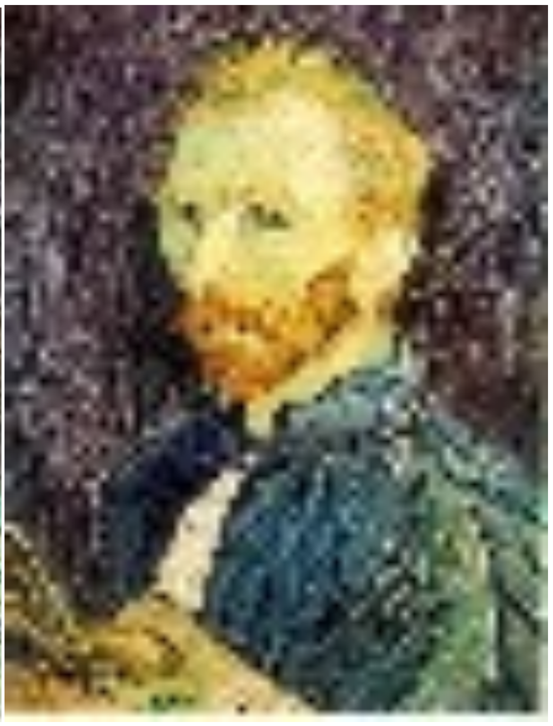


How about throwing away every other row and column to create a half-size image?

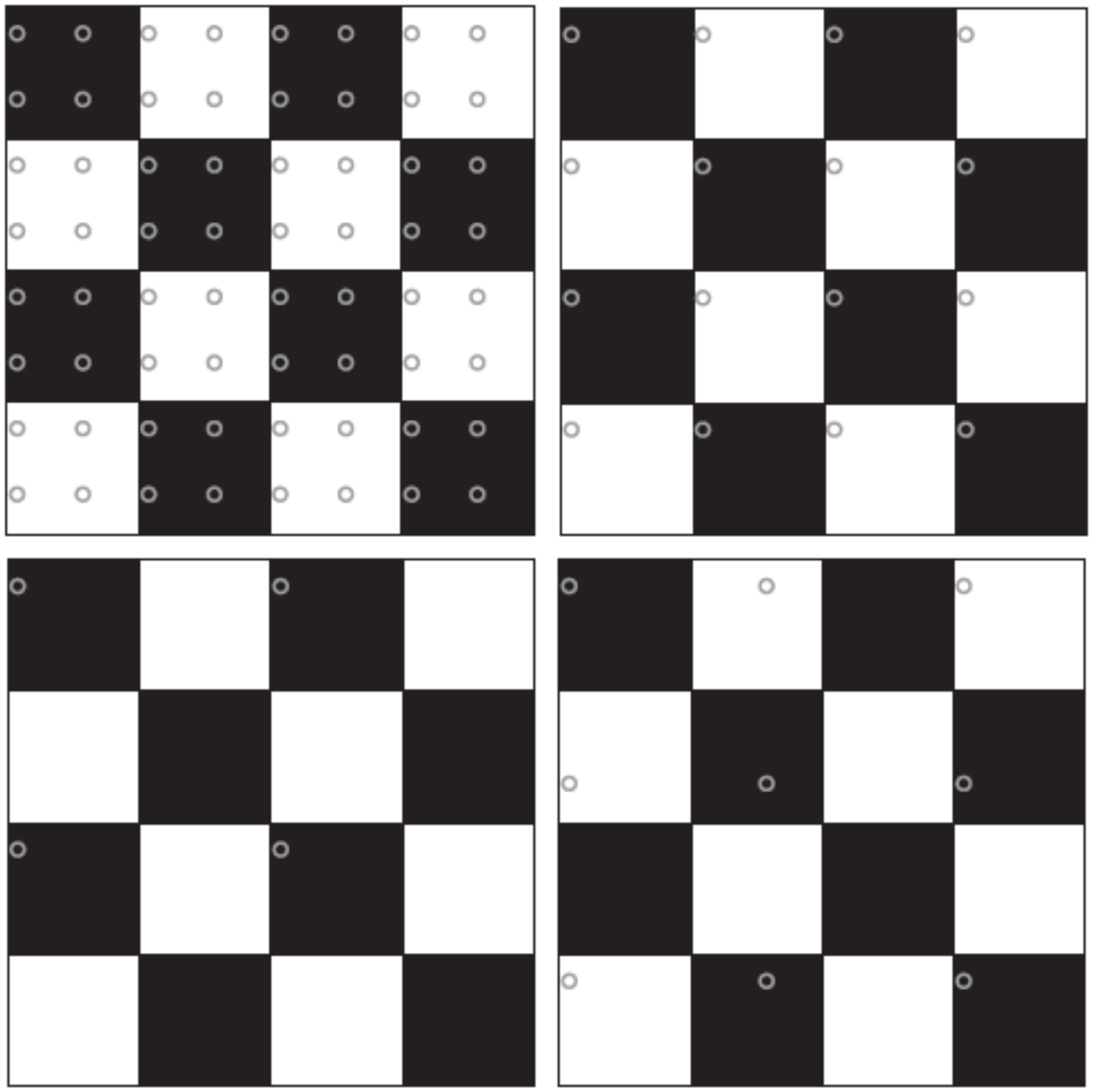
Subsampling without pre-filtering


1/2

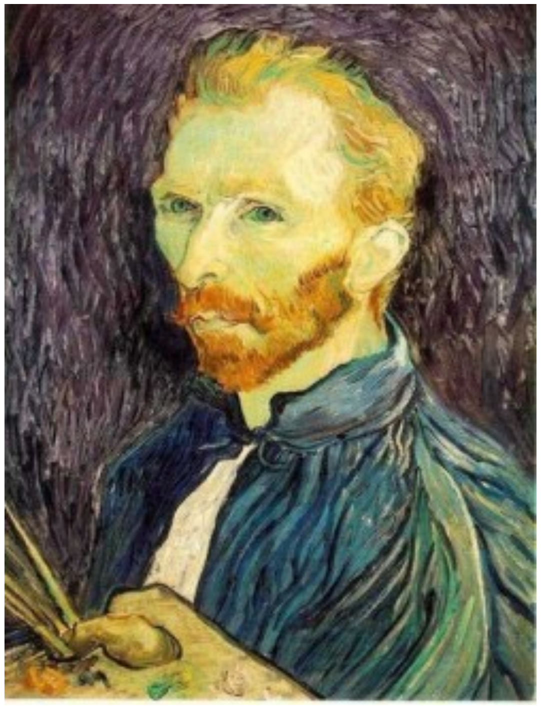

1/4 (2x zoom)

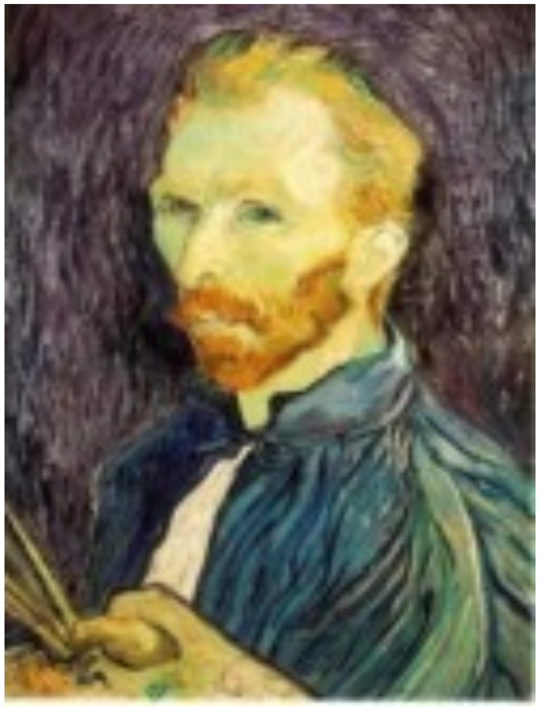

1/8 (4x zoom)

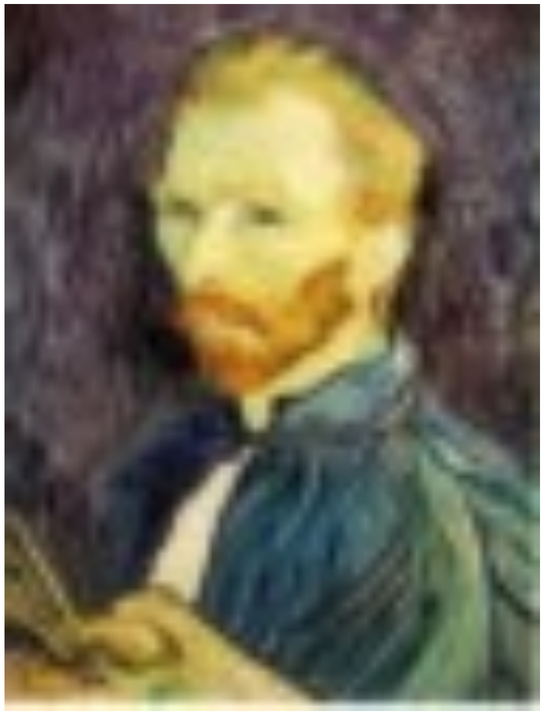
Source: S. Seitz (via D. Hoiem)



Subsampling with pre-filtering


1/2

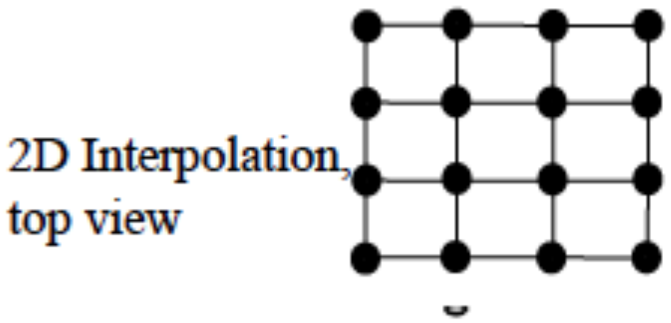

1/4


1/8

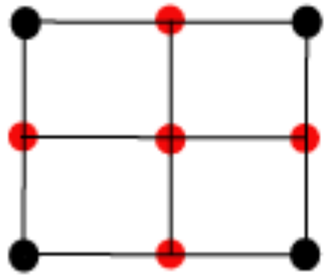
- Image is smoothed with a *Gaussian filter* before subsampling

Source: S. Seitz (via D. Hoiem)

Interpolation

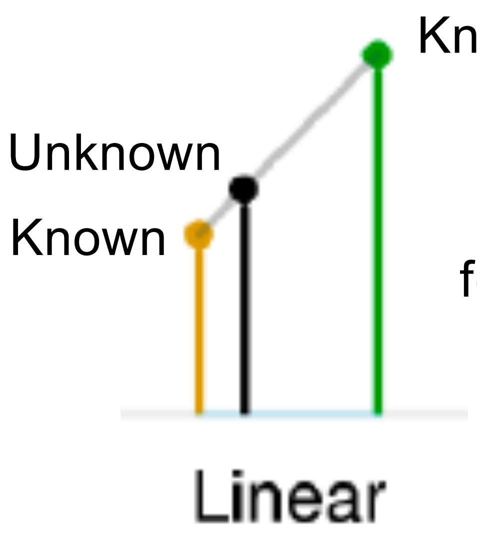


4x4



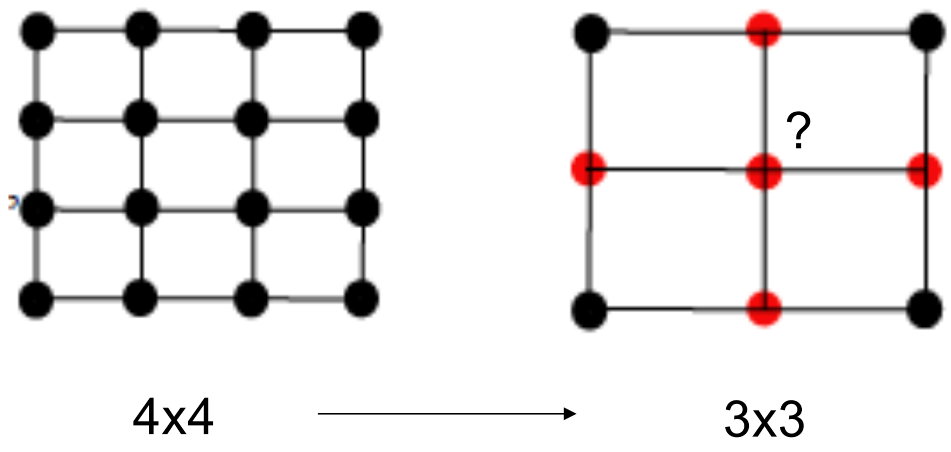
Downsample to 3x3:
Must supply red pixels
Procedures:
-nearest neighbor value
-bilinear interpolation
-other interpolation

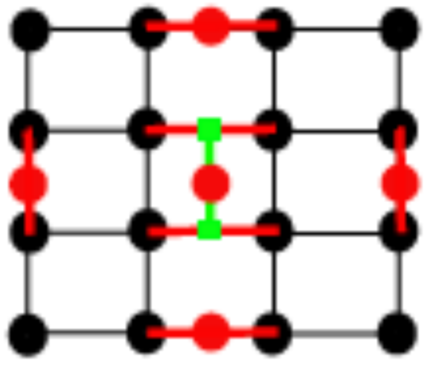
Linear and Bilinear Interpolation



$f(t)=(1-t)f(0)+t\ f(1)$

In 1D





Bilinear interpolation

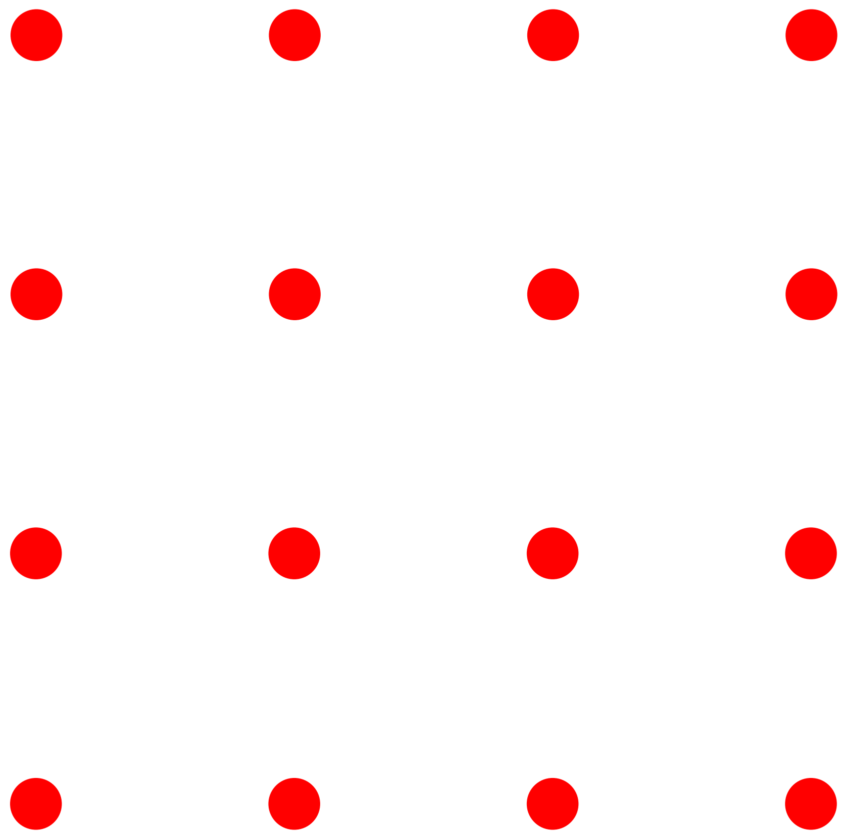
We want a value at $i + \delta i, j + \delta j$, where i and j are integers; $0 < \delta i < 1$; and $0 < \delta j < 1$. Write v_{ij} for the value at i, j. Then use

$$v = v_{ij}(1 - \delta i)(1 - \delta j) + v_{i+1,j}(\delta i)(1 - \delta j) + v_{i,j+1}(1 - \delta i)(\delta j) + v_{i+1,j+1}(\delta i)(\delta j).$$

Notice that if δi and δj are both zero, then $v = v_{ij}$; if they are both one, $v = v_{i+1,j+1}$; and v will interpolate the value at the other two corners, too. By a little manipulation, you can show that this procedure boils down to: predict a value for $i + \delta i, j$ using a linear interpolate; predict a value for $i + \delta i, j + \delta j$ using a linear interpolate; now linearly interpolate between these two to get a value for $i + \delta i, j + \delta j$. Modern hardware is particularly efficient at bilinear interpolation, and any reasonable software environment will be able to do this for you.

Upsampling an image

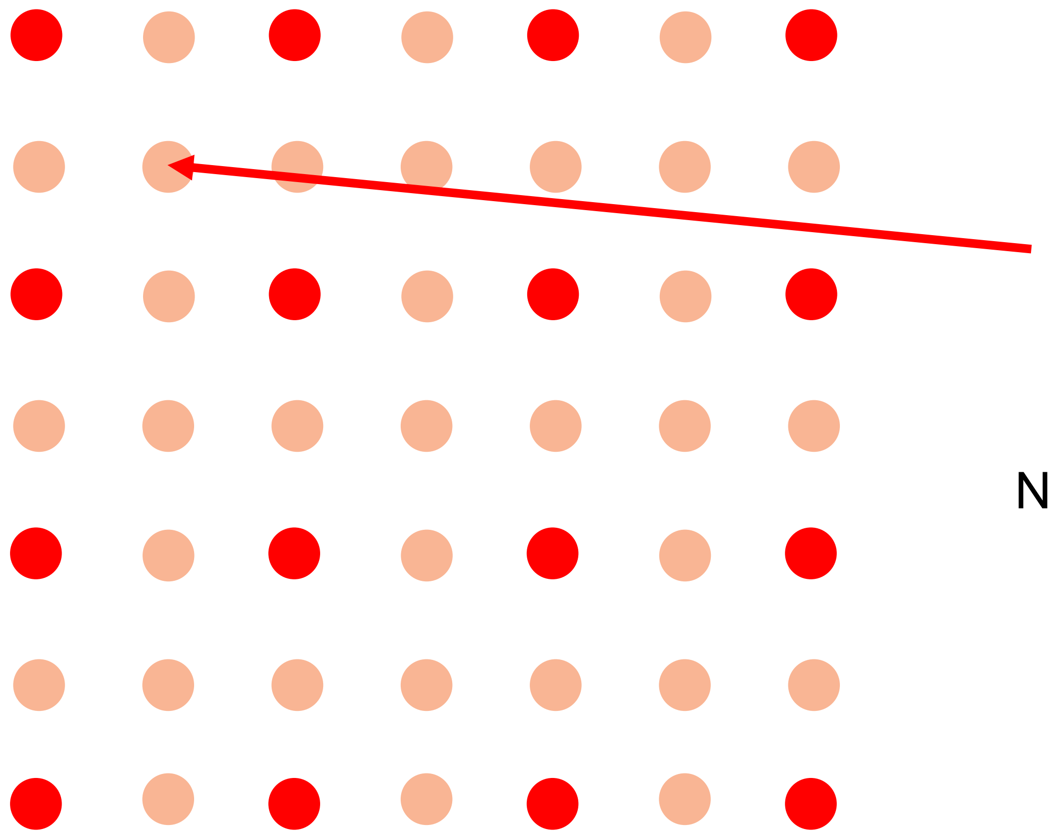
- How do we *increase* the size of an image by a factor of two?



Let’s increase the resolution of the sampling grid!

Upsampling an image

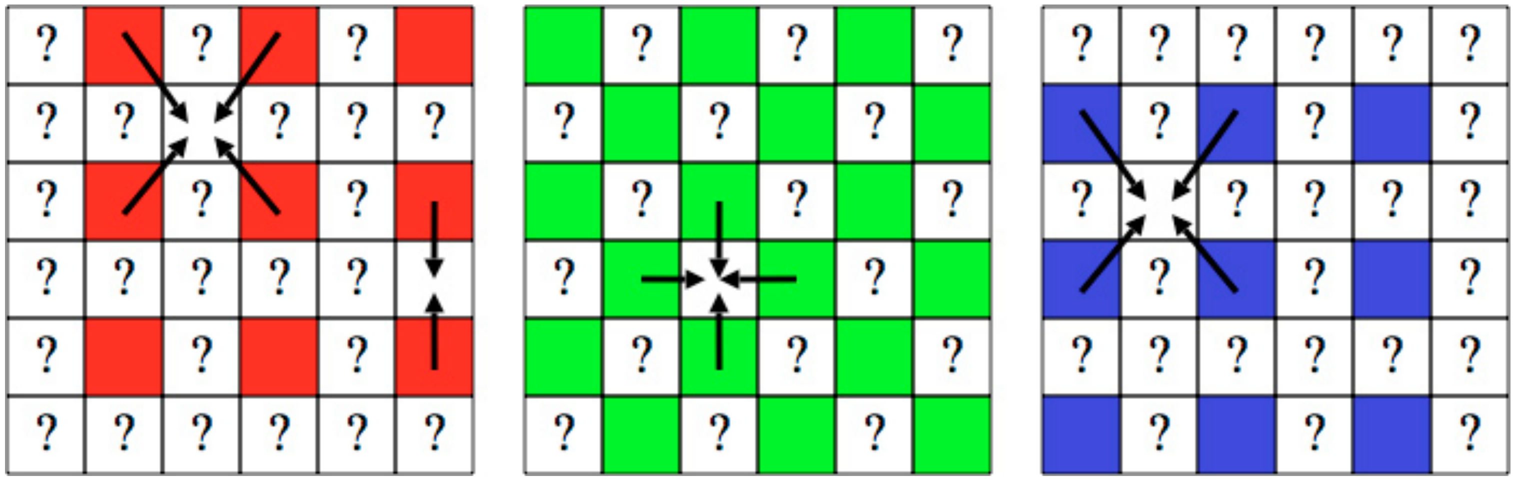
- How do we *increase* the size of an image by a factor of two?



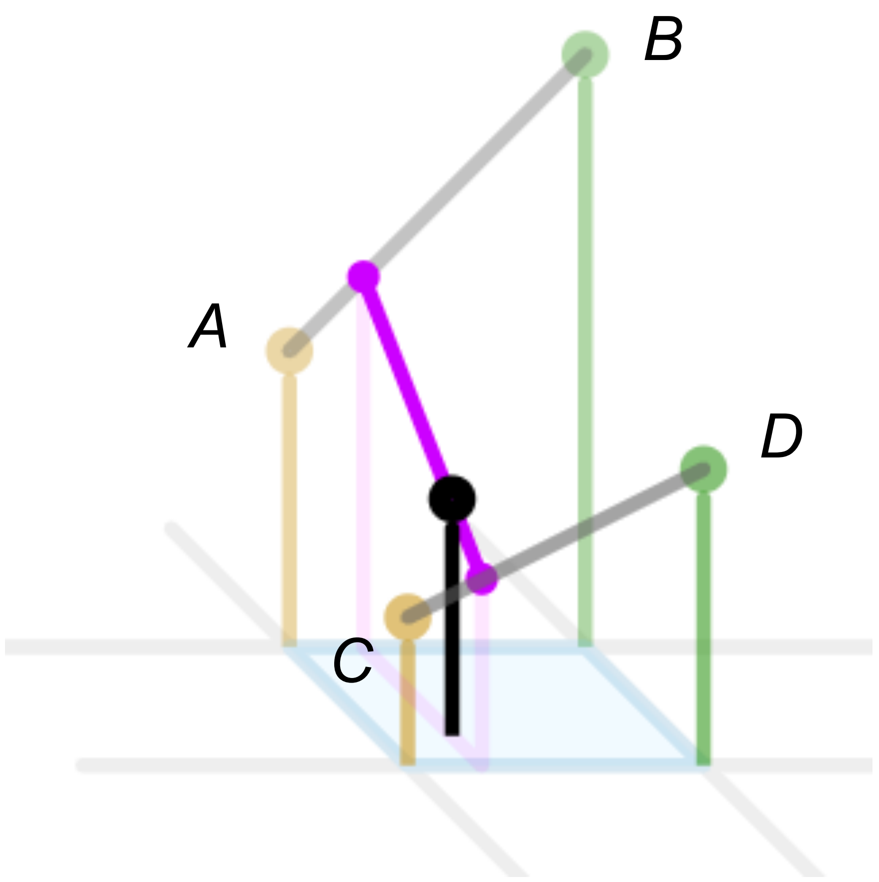
What should this value be?

Need to *interpolate*!

Application: Demosaicing

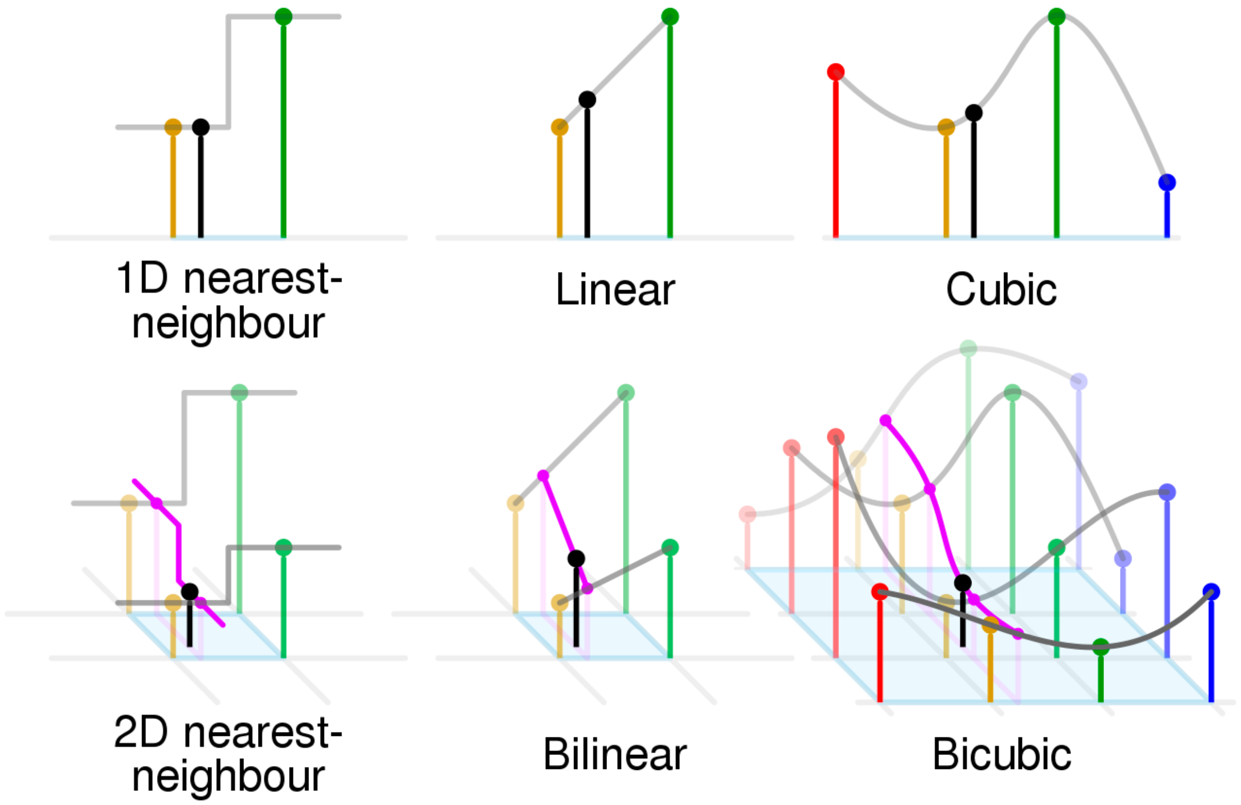


Bilinear interpolation more generally



http://en.wikipedia.org/wiki/Bilinear_interpolation

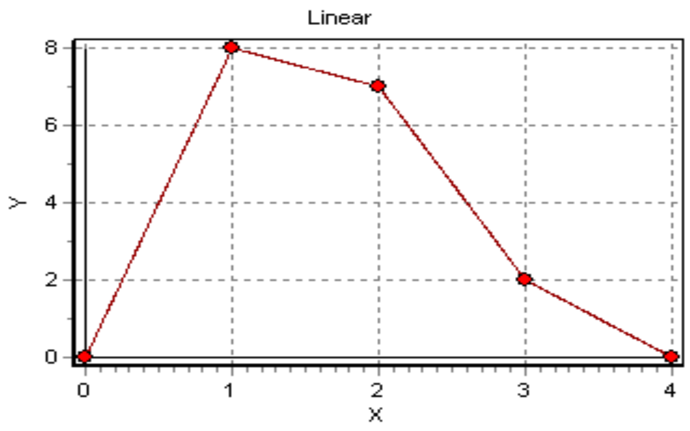
Other kinds of interpolation



Source: Wikipedia

Interpolation and function extrema

- When you use linear interpolation, extrema of the image function can only occur at the original sample points
- What about nonlinear interpolation?



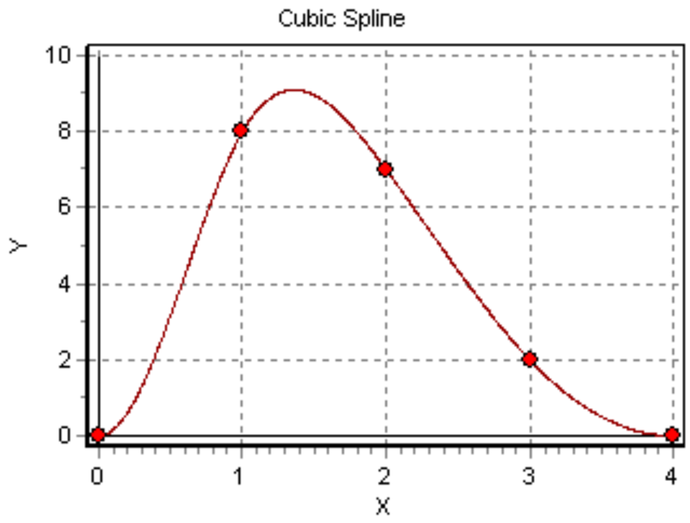


Image source

Image processing basics: Outline

- Images as sampled functions
- Sampling and reconstruction, aliasing
- Image resampling, interpolation
- Image transformations

Image transformations

Downsampling

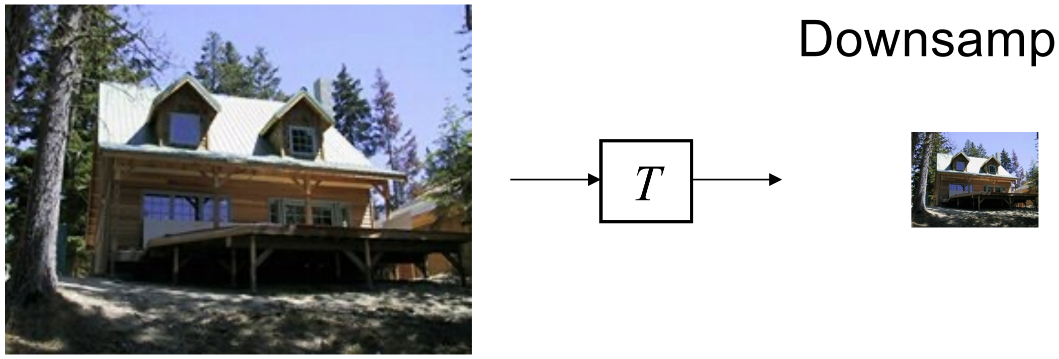


Image transformations

Upsampling

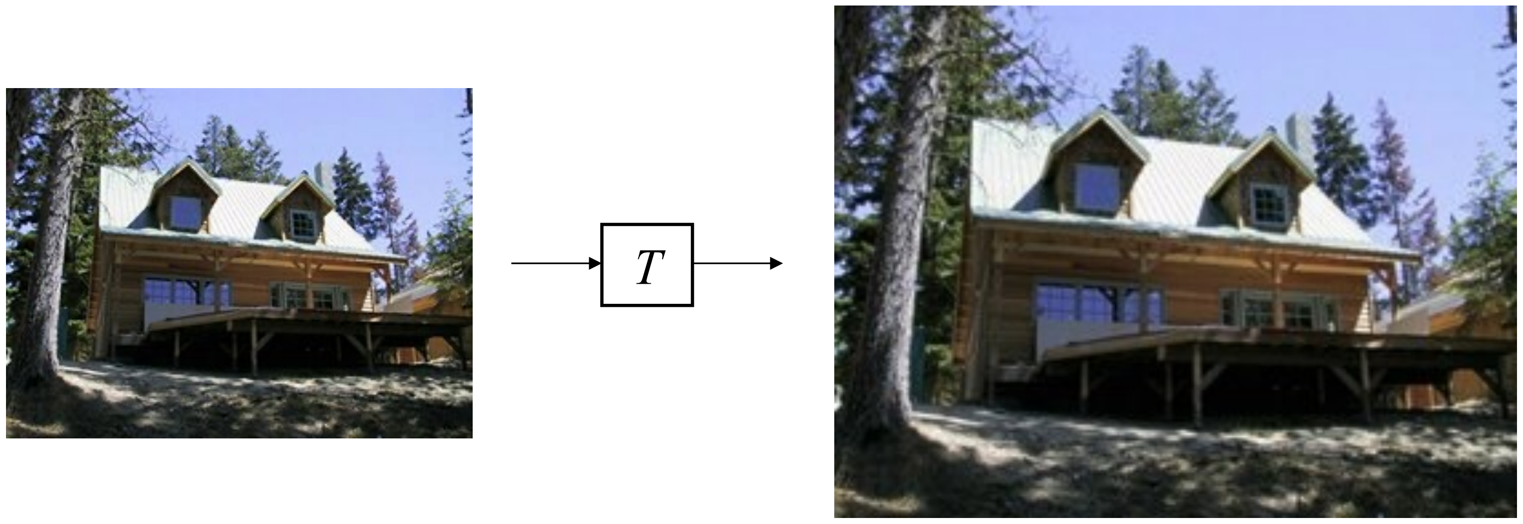


Image transformations

Contrast change

$$\longrightarrow \boxed{T} \longrightarrow$$

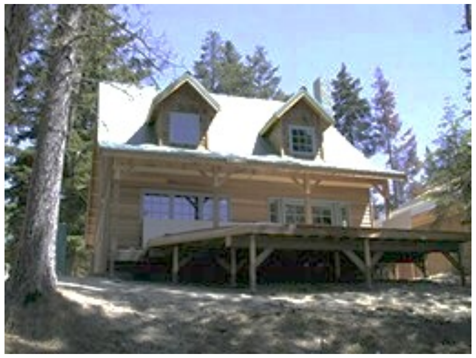


Image transformations

Blurring/sharpening

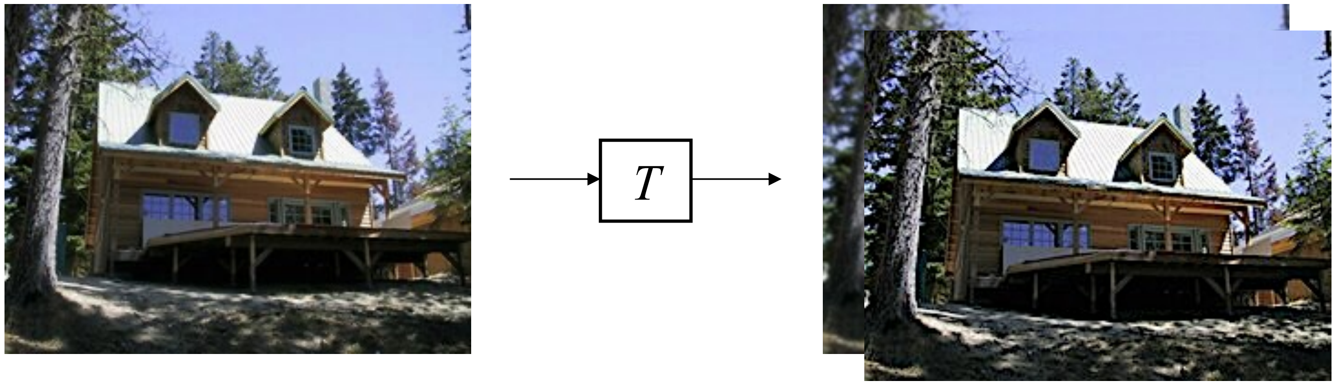
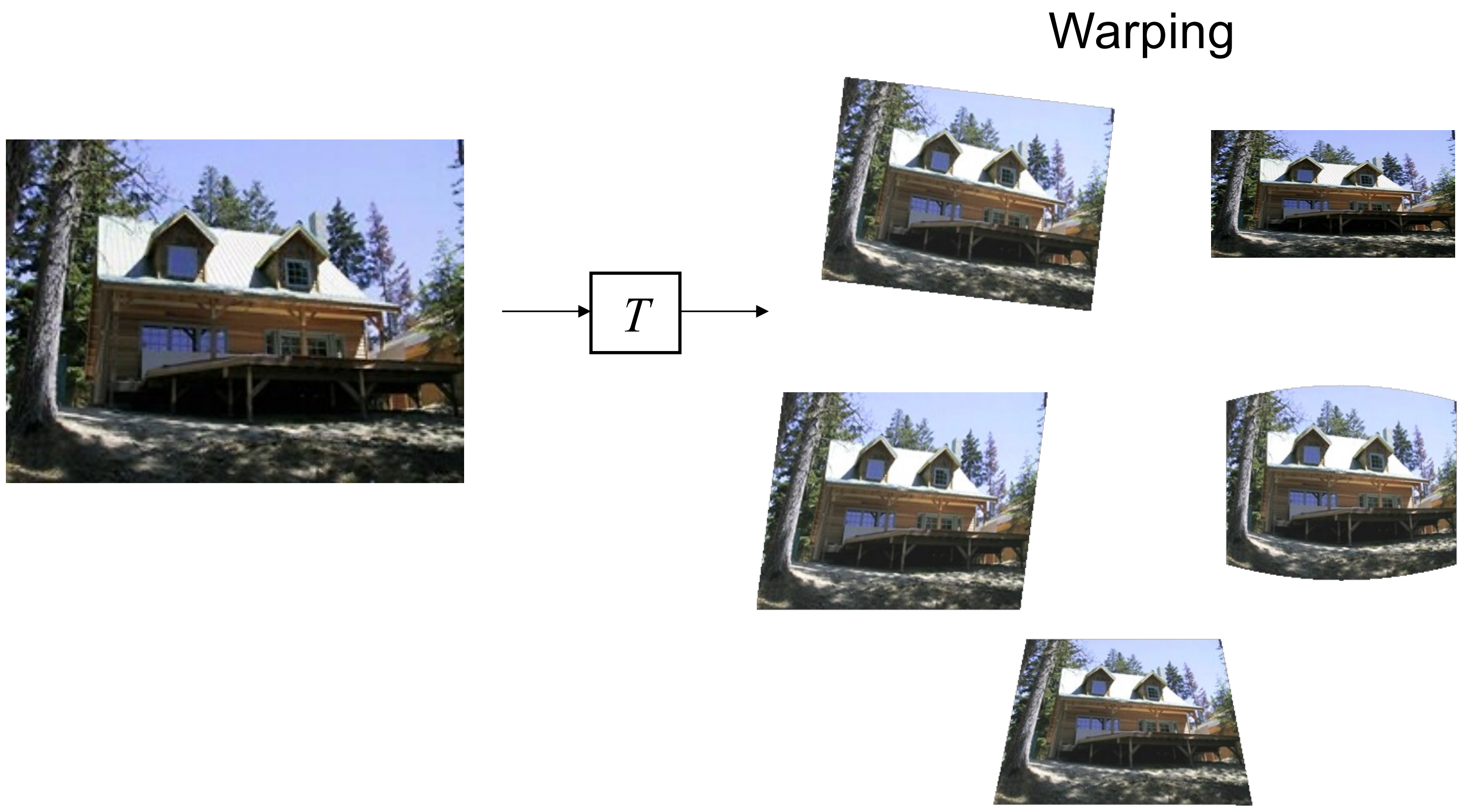


Image transformations



Point processing

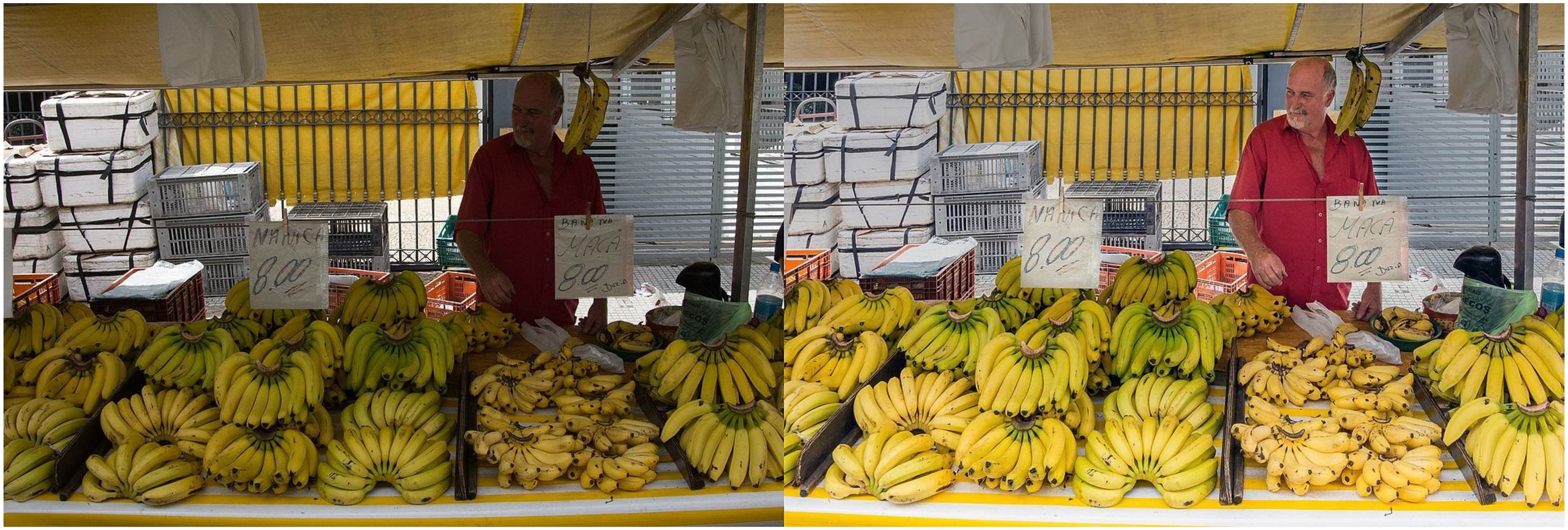
- Change ***range*** of image:
  - Apply the same function to each pixel value

- Many applications

- Common
  - Linear sensors produce strange pictures
    - The world has high dynamic range, 8 bits is too few
    - Typically 12 bit sensor, 8 bit image
  - Most cameras process sensor results before they report them

From wikipedia

From wikipedia



From wikipedia

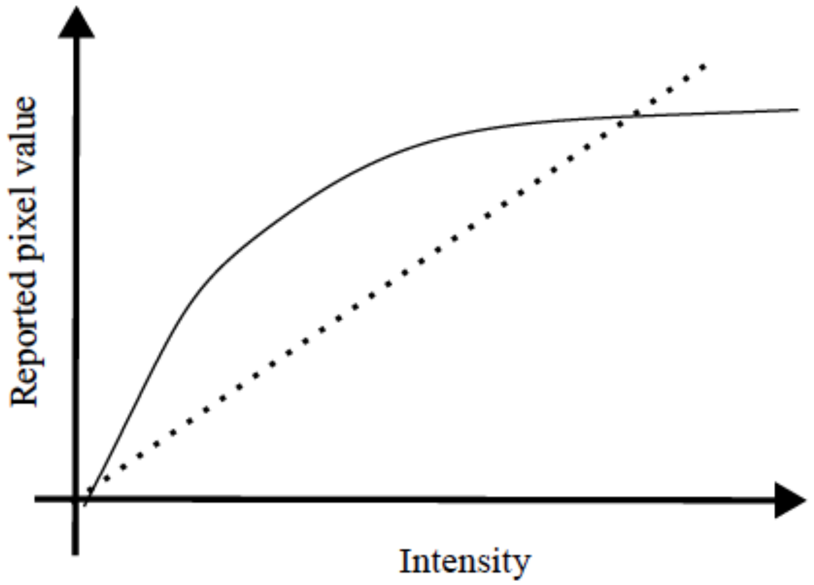


FIGURE 2.7: *A typical camera response function, mapping the response a linear sensor would compute to the output recorded by the camera. Notice that locations that would be quite dark for a linear sensor will be lighter; but as the linear sensor gets very bright, the output recorded by the camera grows slowly. This means that the range of outputs is smaller than the range of inputs, which is helpful for practical cameras. This response function is typically located deep in the camera's electronics. Typical consumer cameras apply a variety of transforms before reporting an image, though one can often persuade cameras to produce an untransformed, linear response image (a RAW file).*

Negative

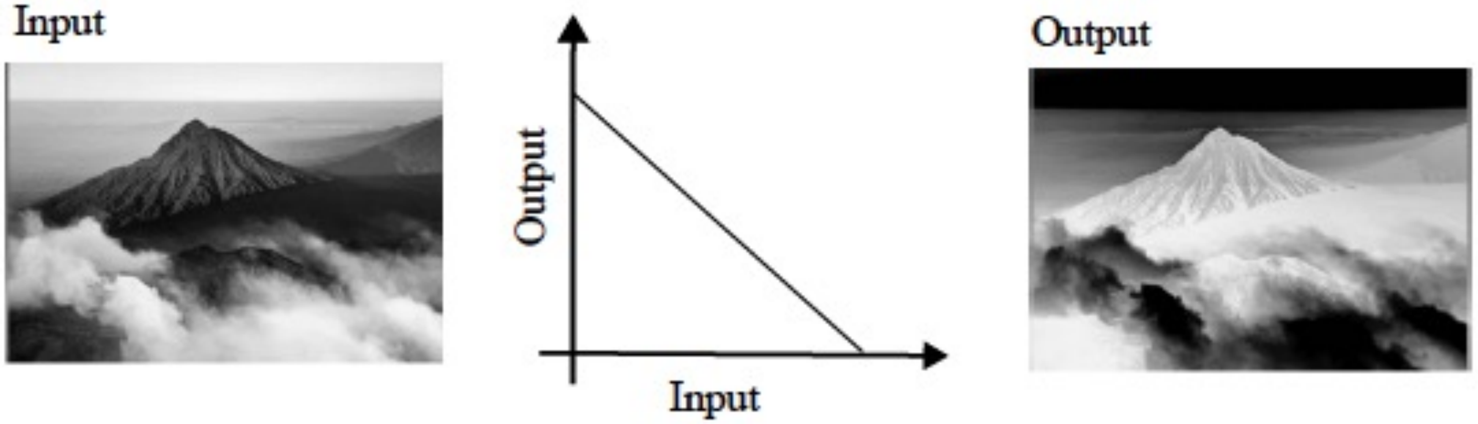


FIGURE 2.8: *Mapping individual pixel values using the mapping in the* center *will transform the image on the* left *to that on the* right. *This function maps light pixel values to dark ones, and vice versa, and is often called a negative.*

Contrast adjustment

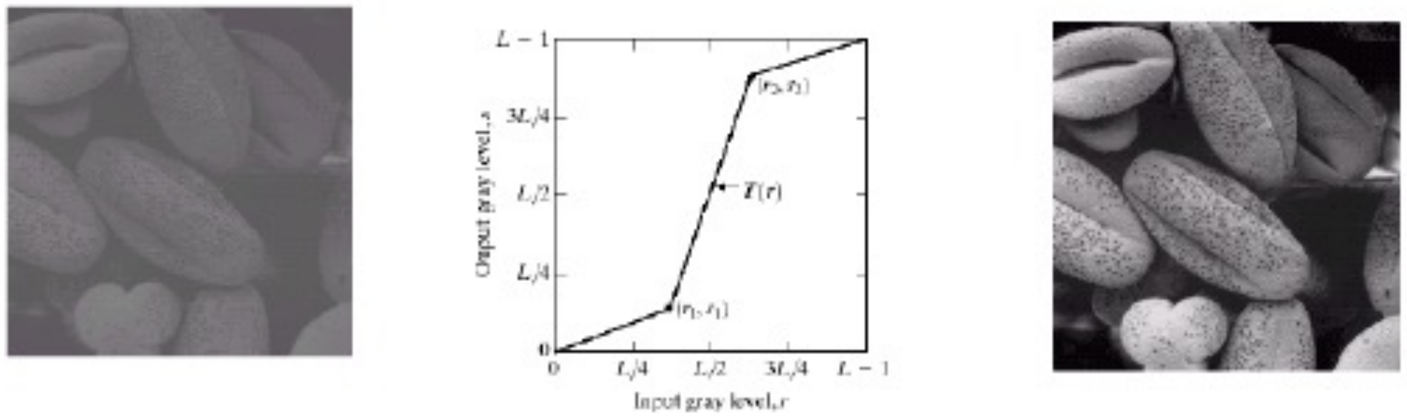
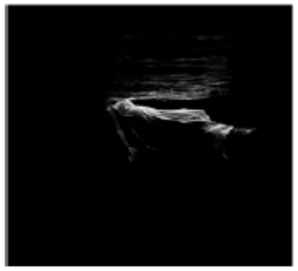


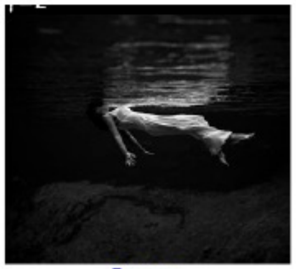
FIGURE 2.9: *Mapping individual pixel values using the mapping in the* center *will transform the image on the* left *to that on the* right. *Notice how the mapping compresses the range of dark and light pixels, and expands that of mid-range pixels, so adjusting the contrast of the image.*

Gamma correction

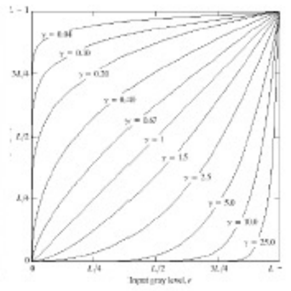
$\gamma = 2$



Original



Source



$\gamma = 1/2$

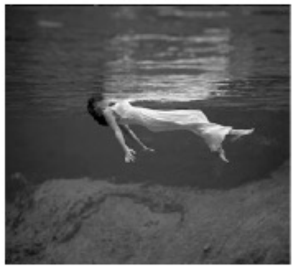


FIGURE 2.10: *Many imaging and rendering devices have a response that is a power of the input, so that output = Cinput$^{\gamma}$, where γ is a parameter of the device. One can simulate this effect by applying a transform like those shown in the* **center** *(curves for several values of γ). Note that you can remove the effect of such a transform –* gamma correct *the image – by applying another such transform with an appropriately chosen γ. The image on the* **left** *is transformed to the two examples on the* **right** *with different γ values.*

Image filtering

- Roughly speaking, replace image value at x with some function of values in its spatial neighborhood $N(x)$:

$$g(x) = T(f(N(x)))$$

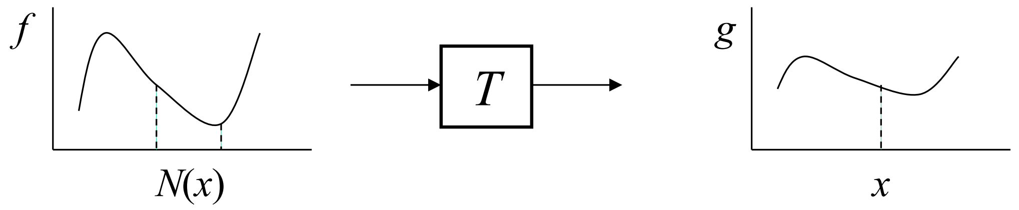
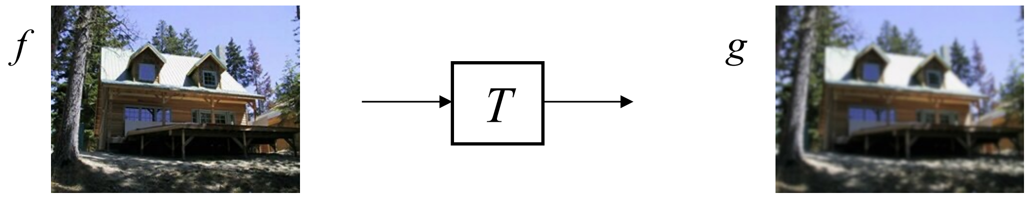


Image filtering

- Roughly speaking, replace image value at x with some function of values in its spatial neighborhood $N(x)$:

$$g(x) = T(f(N(x)))$$



- Examples: smoothing, sharpening, edge detection, etc.

Image warping

- Change ***domain*** of image:

$$x' = T(x), \qquad g(x') = f(x)$$

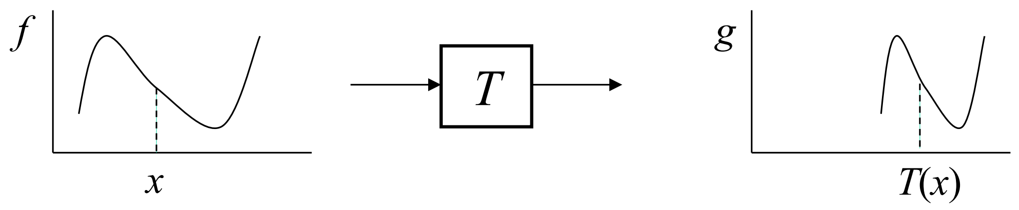


Image warping

- Change ***domain*** of image:

$$x' = T(x), \qquad g(x') = f(x)$$

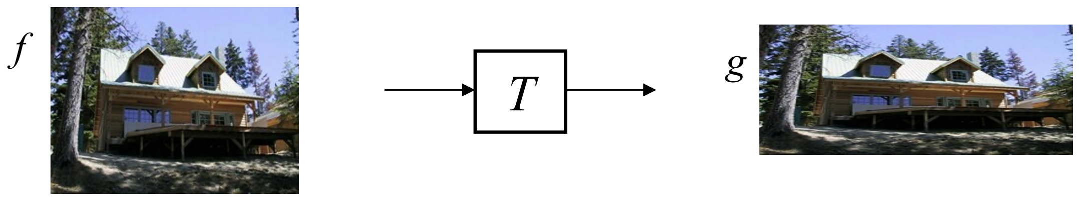


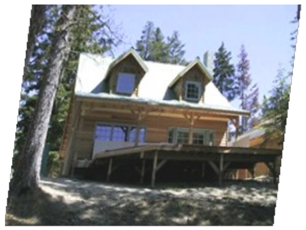
Image warping

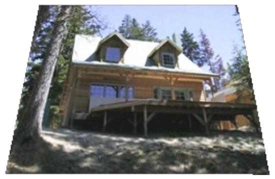
- Examples of *global parametric* warps:

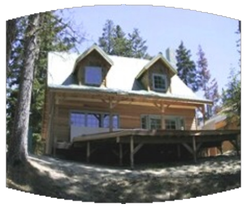
translation

rotation

scaling (uniform or non-uniform)


affine


perspective *homography*


cylindrical (not covered in this class)

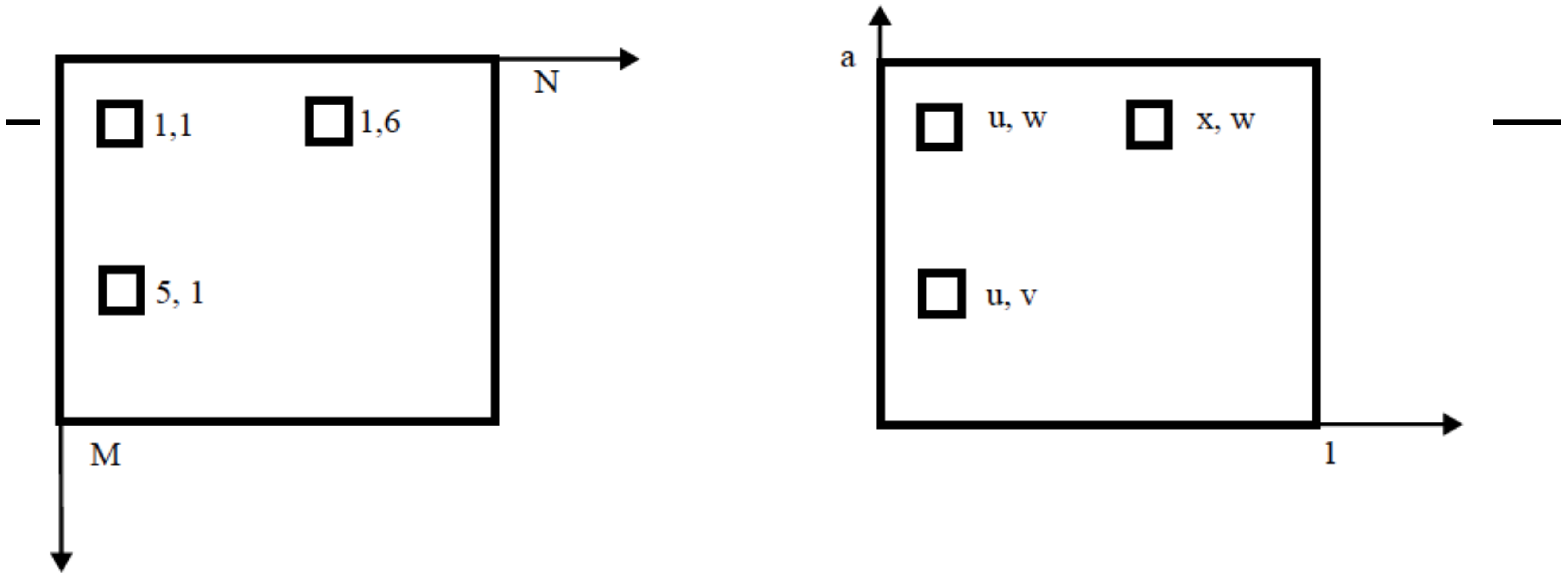
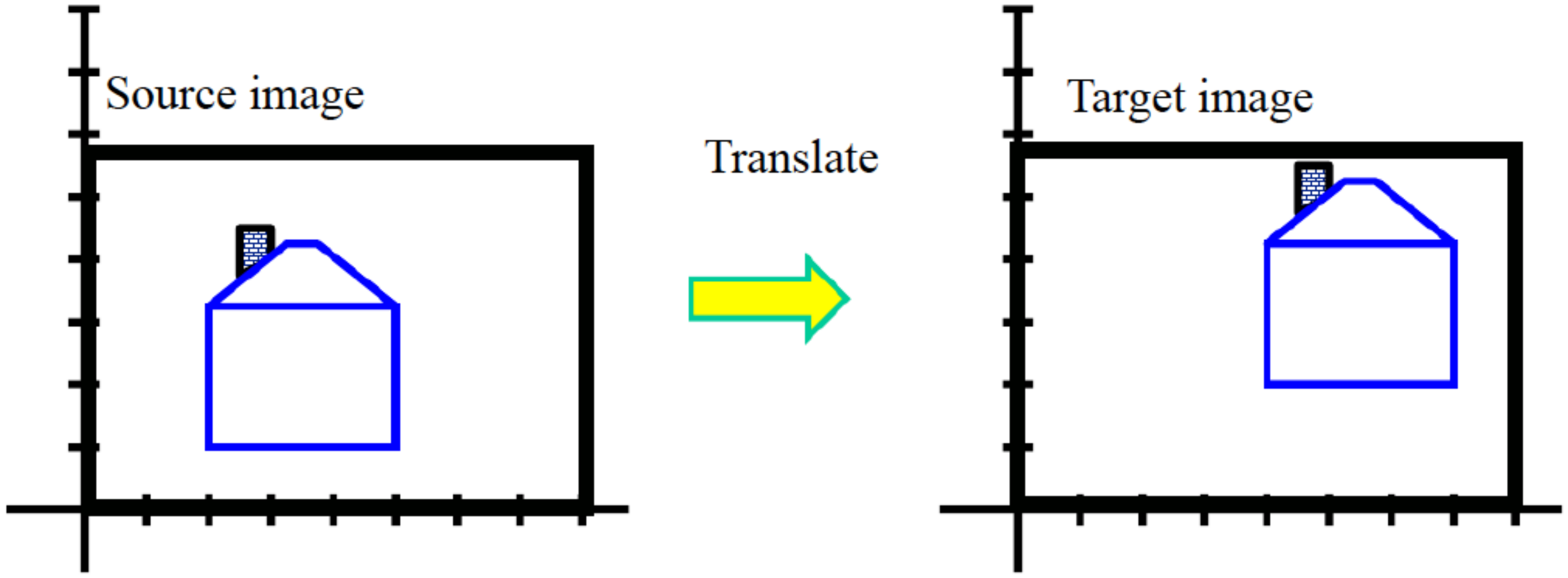


FIGURE 2.11: *Two common coordinate systems for images. On the* **left**, *the origin is at the top left corner, and we count in pixels. This is an $M \times N$ image. I will use the convention $\mathcal{I}_{ij}$ for points in this coordinate system, so the top right pixel is $\mathcal{I}_{16}$. On the right, the origin is at the bottom left, and the coordinate axes are more familiar. It is a good idea to use a range from $0 - 1$ (rather than $0 - M$) in this coordinate system, but if the image is not square one direction will run from 0 to a. I will use the convention $\mathcal{I}(x, y)$ for points in this coordinate system, so that the bottom left pixel is $\mathcal{I}(u, v)$.*

Translation

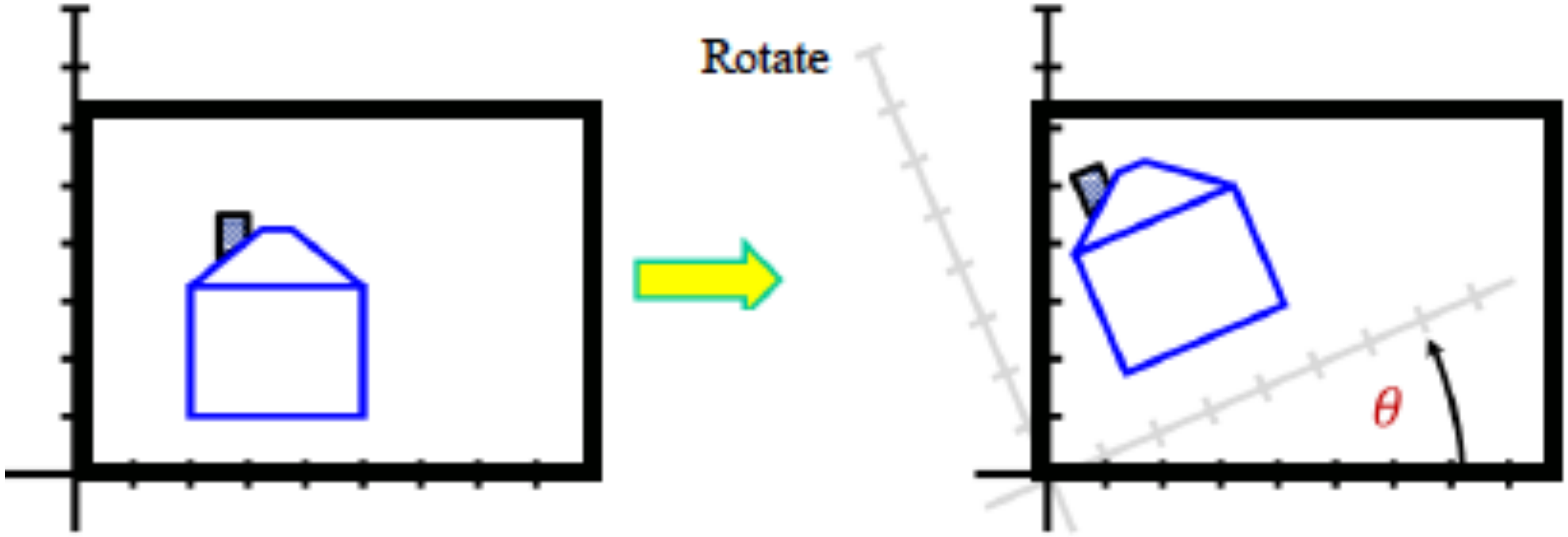
Write $\mathcal{S}$ for a source image and $\mathcal{T}$ for a target image. Many important transformations have the form $\mathcal{T}(u(x,y), v(x,y)) = \mathcal{S}(x,y)$. Simpler examples include:



- **Translating an image** where $u(x,y) = x + t_x$, $v(x,y) = y + t_y$ (check that if $t_x > 0, t_y > 0$, the image moves up and to the right, as in the figure).

Rotation

Write $\mathcal{S}$ for a source image and $\mathcal{T}$ for a target image. Many important transformations have the form $\mathcal{T}(u(x,y), v(x,y)) = \mathcal{S}(x,y)$. Simpler examples include:



- **Rotating an image** where $u(x,y) = x\cos\theta - y\sin\theta$, $v(x,y) = x\sin\theta + y\cos\theta$ (check that if $\theta > 0$, the rotation is counterclockwise, as in the figure).

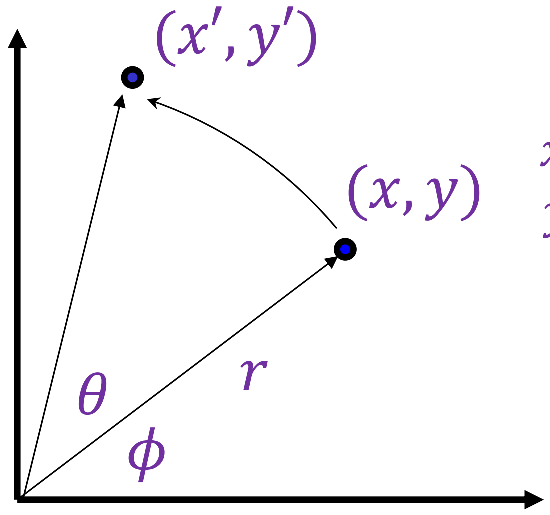
Rotation

- Rotate the image by an angle of θ about the origin:

$$x' = r\cos(\phi+\theta) = r\cos(\phi)\cos(\theta) - r\sin(\phi)\sin(\theta) = x\cos(\theta) - y\sin(\theta)$$
$$y' = r\sin(\phi+\theta) = r\sin(\phi)\cos(\theta) + r\cos(\phi)\sin(\theta) = x\sin(\theta) + y\cos(\theta)$$

Apply trig identities

Substitute

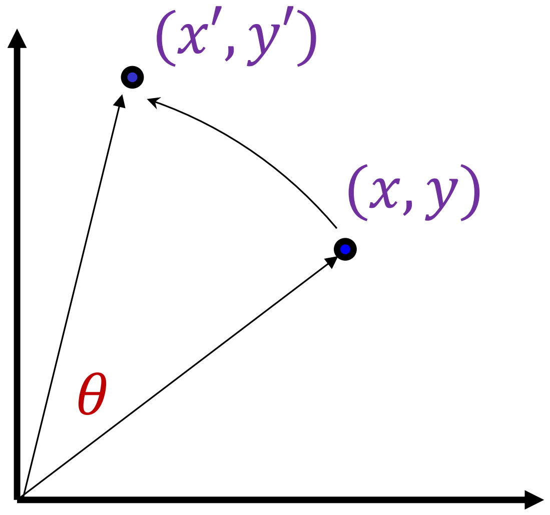


$$x = r\cos(\phi)$$
$$y = r\sin(\phi)$$

Rotation

- Rotate the image by an angle of θ about the origin:

$$\boxed{\begin{aligned} x' &= x\cos(\theta) - y\sin(\theta) \\ y' &= x\sin(\theta) + y\cos(\theta) \end{aligned}}$$



In matrix form:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

Rotation

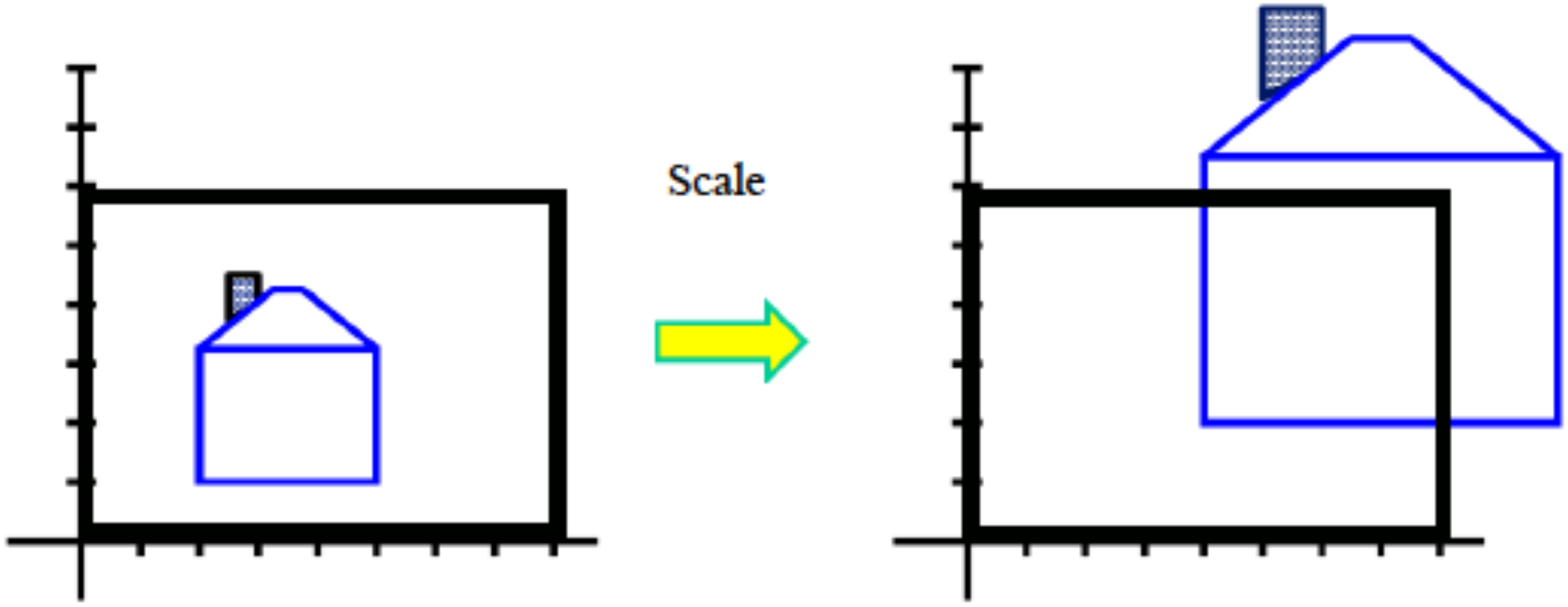
- 2D rotation in matrix form:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \underbrace{\begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}}_{R(\theta)} \begin{pmatrix} x \\ y \end{pmatrix}$$

- Note: even though $\cos(\theta)$ and $\sin(\theta)$ are *nonlinear* functions of θ, x' and y' are *linear* combinations of x and y
- What is the inverse transformation?
  - Rotation by $-\theta$
  - For rotation matrices, $R^{-1} = R^T$

Uniform scaling

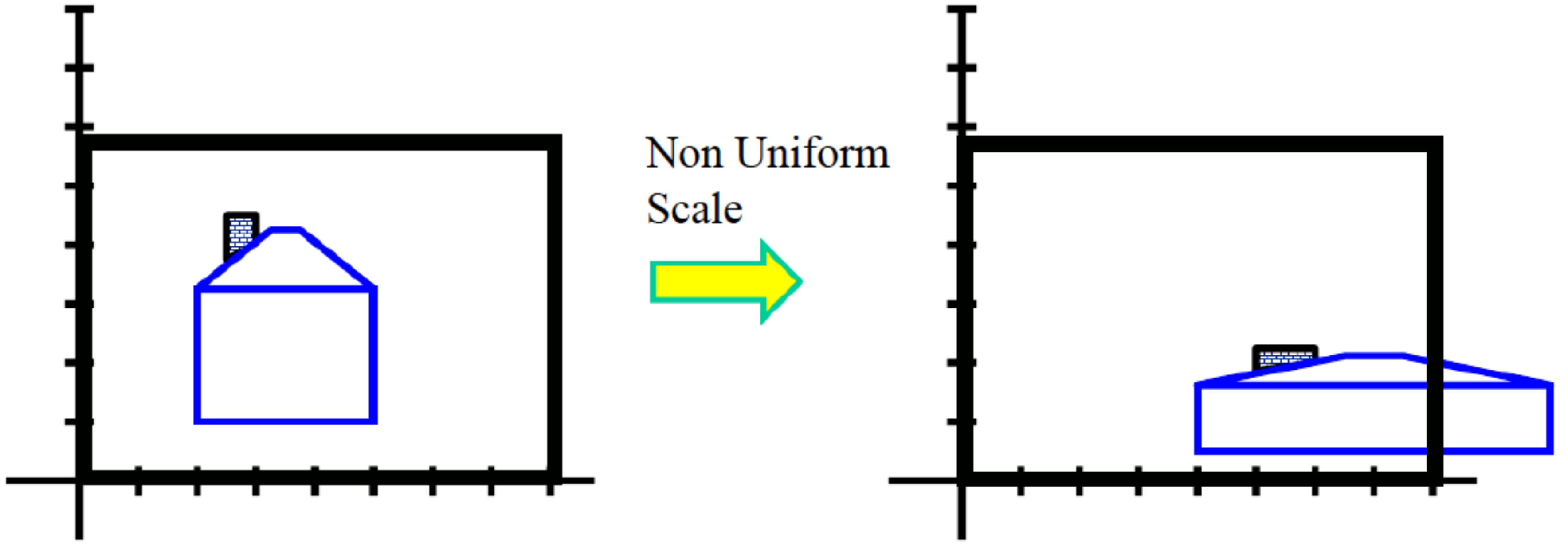
Write $\mathcal{S}$ for a source image and $\mathcal{T}$ for a target image. Many important transformations have the form $\mathcal{T}(u(x,y), v(x,y)) = \mathcal{S}(x,y)$. Simpler examples include:



- **Scaling an image uniformly** where $u(x,y) = sx$, $v(x,y) = sy$ (check that if $s > 1$, the image gets bigger, and if $s < 1$, it gets smaller, as in the figure).

Non-uniform scaling

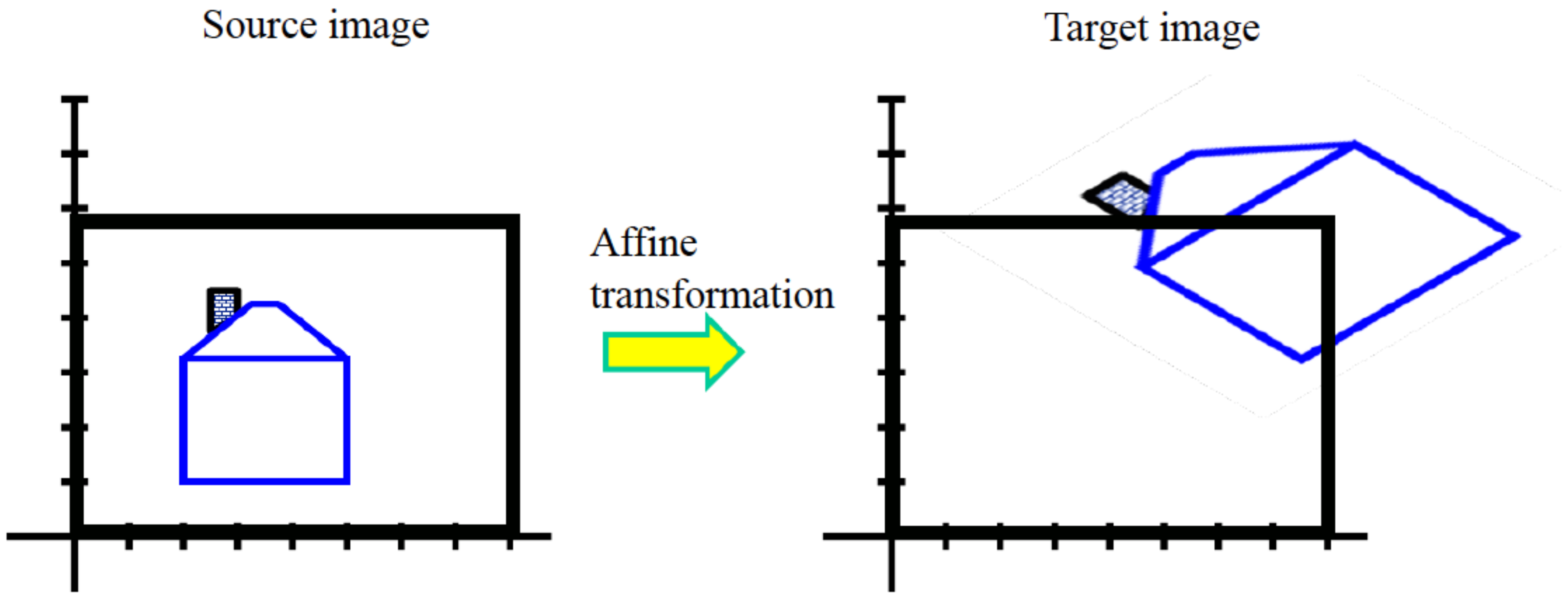
Write $\mathcal{S}$ for a source image and $\mathcal{T}$ for a target image. Many important transformations have the form $\mathcal{T}(u(x,y), v(x,y)) = \mathcal{S}(x,y)$. Simpler examples include:



- **Scaling an image non-uniformly** where $u(x,y) = sx$, $v(x,y) = ty$.

Affine transformation

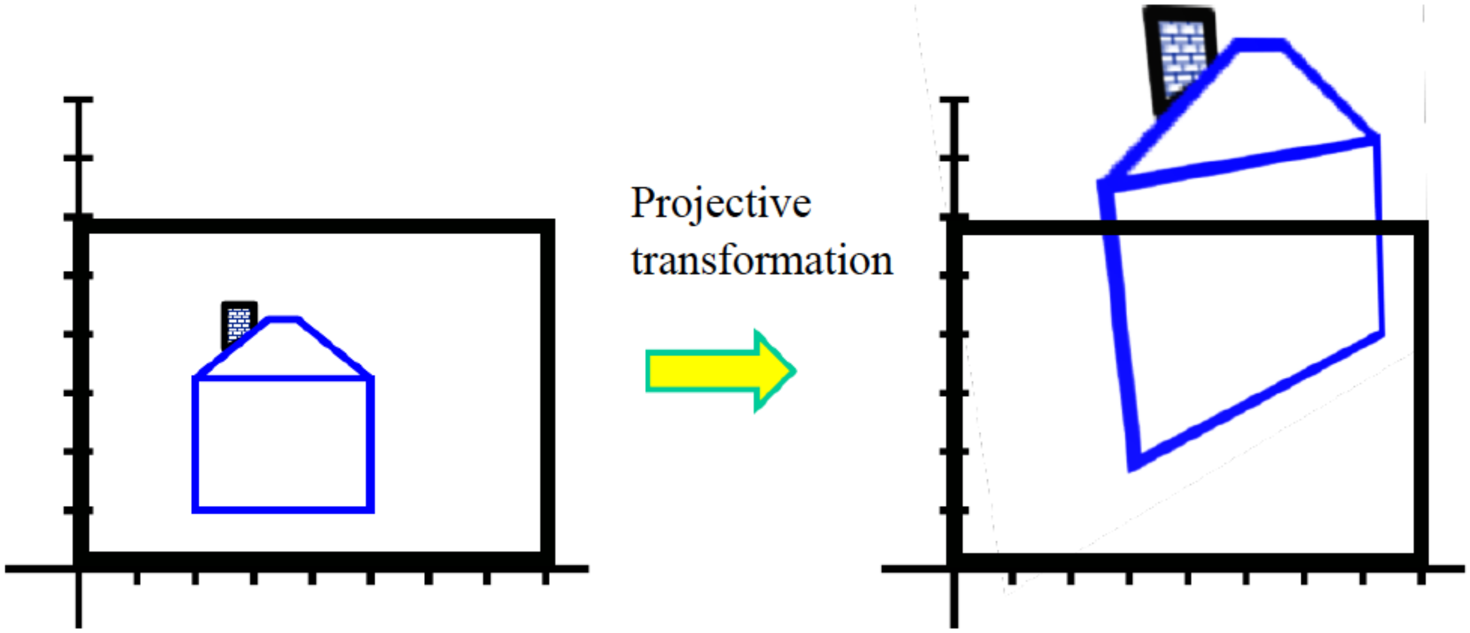
Write $\mathcal{S}$ for a source image and $\mathcal{T}$ for a target image. Many important transformations have the form $\mathcal{T}(u(x,y), v(x,y)) = \mathcal{S}(x,y)$. Simpler examples include:



- **Affine transformations** where $u(x,y) = ax + by + c$, $v(x,y) = dx + ey + f$.

Projective transformation

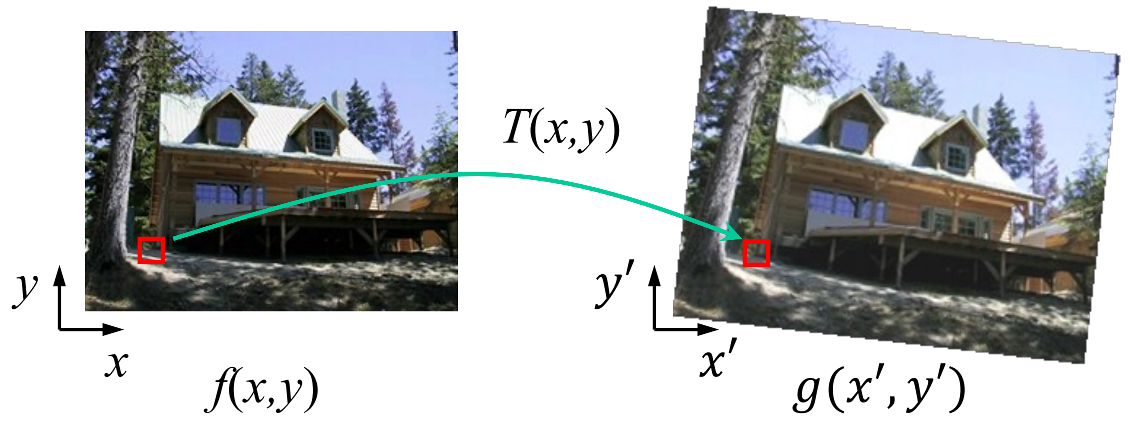
Write $\mathcal{S}$ for a source image and $\mathcal{T}$ for a target image. Many important transformations have the form $\mathcal{T}(u(x,y), v(x,y)) = \mathcal{S}(x,y)$. Simpler examples include:



- **Projective transformations** where $u(x,y) = \frac{ax+by+c}{gx+hy+i}$, $v(x,y) = \frac{dx+ey+f}{gx+hy+i}$.

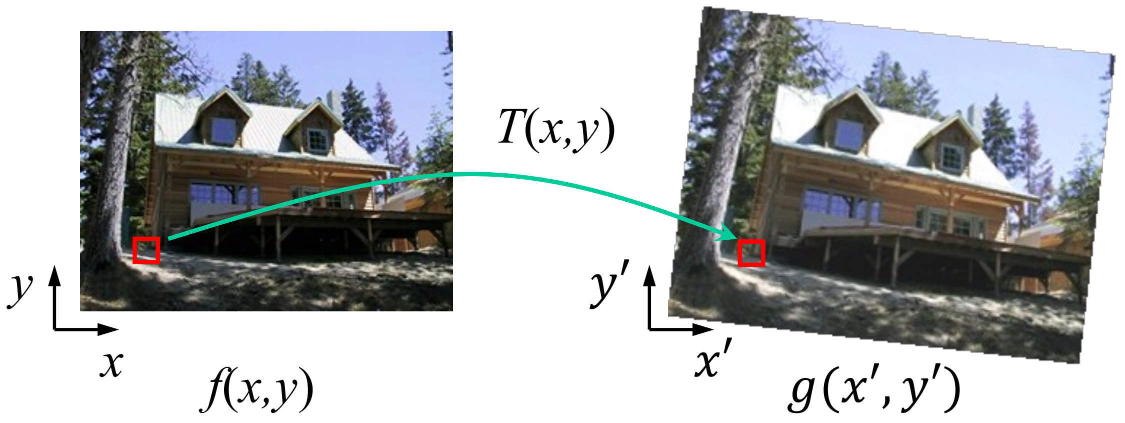
Warping pixels

- Given a coordinate transform $(x', y') = T(x, y)$ and a source image $f(x, y)$, how do we compute a transformed image $g(x', y') = f(T(x, y))$?



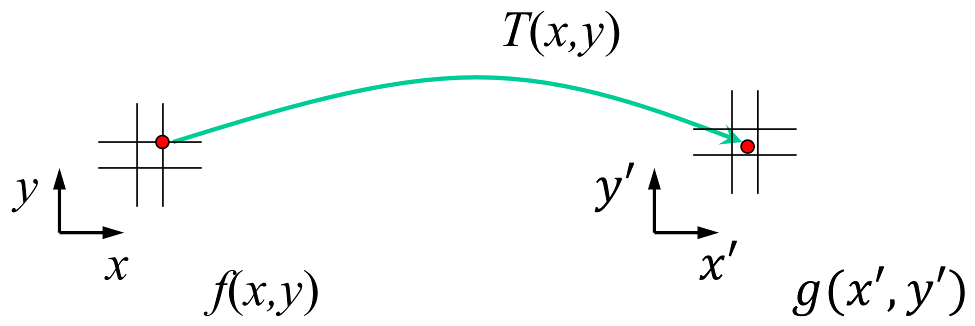
Forward warping

- Send each pixel $f(x, y)$ to its corresponding location $(x', y') = T(x, y)$ in the second image



Forward warping

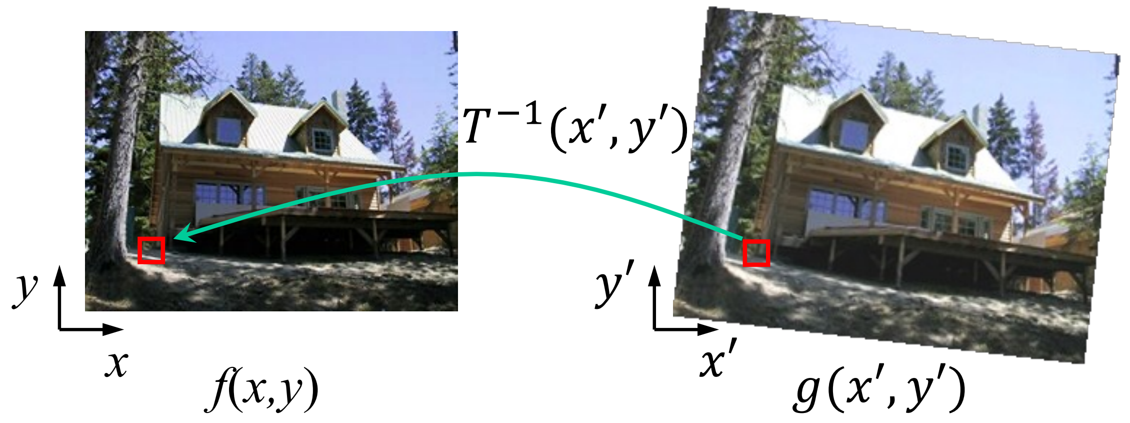
- Send each pixel $f(x, y)$ to its corresponding location $(x', y') = T(x, y)$ in the second image
  - What if (x', y') is not an integer location on the pixel grid?
  - We can “distribute” colors among the neighboring grid locations – known as *splatting*



  - *Often (usually) holes in target image!*

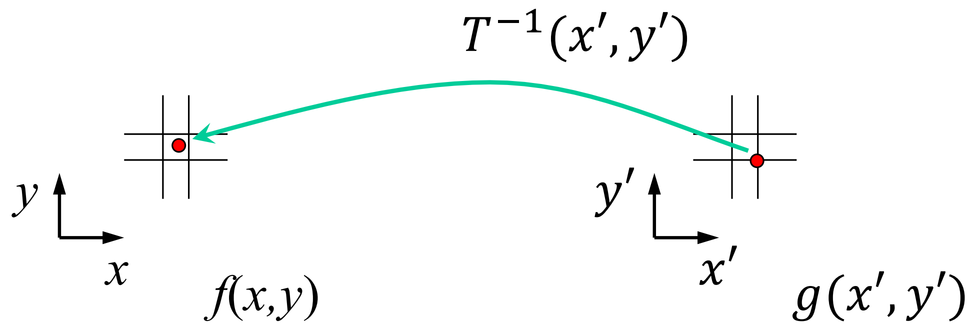
Inverse warping

- For each pixel grid location (x', y') in the second image, get the color from its corresponding location $(x, y) = T^{-1}(x', y')$ in the first image



Inverse warping

- For each pixel grid location (x', y') in the second image, get the color from its corresponding location $(x, y) = T^{-1}(x', y')$ in the first image
  - What if (x, y) is not an integer location on the original pixel grid?
  - Interpolate!



Forward vs. inverse warping

- Which is better?
  - Usually inverse: more efficient, doesn’t have a problem with holes
  - However, it requires an invertible warp function, which is not always possible

Images relative to other images:

Principal components analysis can be applied to images (sometimes useful)

Principal components analysis:

Assume we have a dataset of N d-dimensional vectors $\{\mathbf{x}\}$. This dataset has mean $\mathsf{mean}(\{\mathbf{x}\})$ and covariance $\mathsf{Covmat}(\{\mathbf{x}\})$. Principal components analysis yields a set of directions $\mathbf{p}_j$, which are eigenvectors of the covariance matrix. These directions are orthonormal (so that $\mathbf{p}_i^T\mathbf{p}_j$ is one if $i = j$ and zero otherwise.

Any data item $\mathbf{x}_i$ can be represented as

$$\mathbf{x}_i = \mathsf{mean}(\{\mathbf{x}\}) + \sum_{j=1}^{w} s_{i,j}\mathbf{p}_j.$$

Principal components analysis:

Most datasets have the remarkable property that this representation has very low error even when w is considerably smaller than d (Chapter 33.2 if you haven't seen this before). The coefficents $s_{i,j}$ have strong properties, too. First, the mean over the dataset of each coefficient is zero, so

$$\frac{1}{N}\sum_i s_{i,j} = 0$$

and second, the directions can be ordered by the variance of the coefficients. Write

$$\mathsf{var}\left(\{s\}\right)_j = \frac{1}{N}\sum_i s_{i,j}^2$$

for the variance of the j'th coefficient; then if $k > j$, $\mathsf{var}\left(\{s\}\right)_k < \mathsf{var}\left(\{s\}\right)_j$. Note that $\mathsf{var}\left(\{s\}\right)_j$ is the j'th largest eigenvalue of the covariance. The $\mathbf{p}_j$ are known as *principal components* (sometimes *loadings*) of the dataset; the $s_{i,j}$ are sometimes known as *scores*, but are usually just called *coefficients*. Forming the representation is called *principal components analysis* or *PCA*.

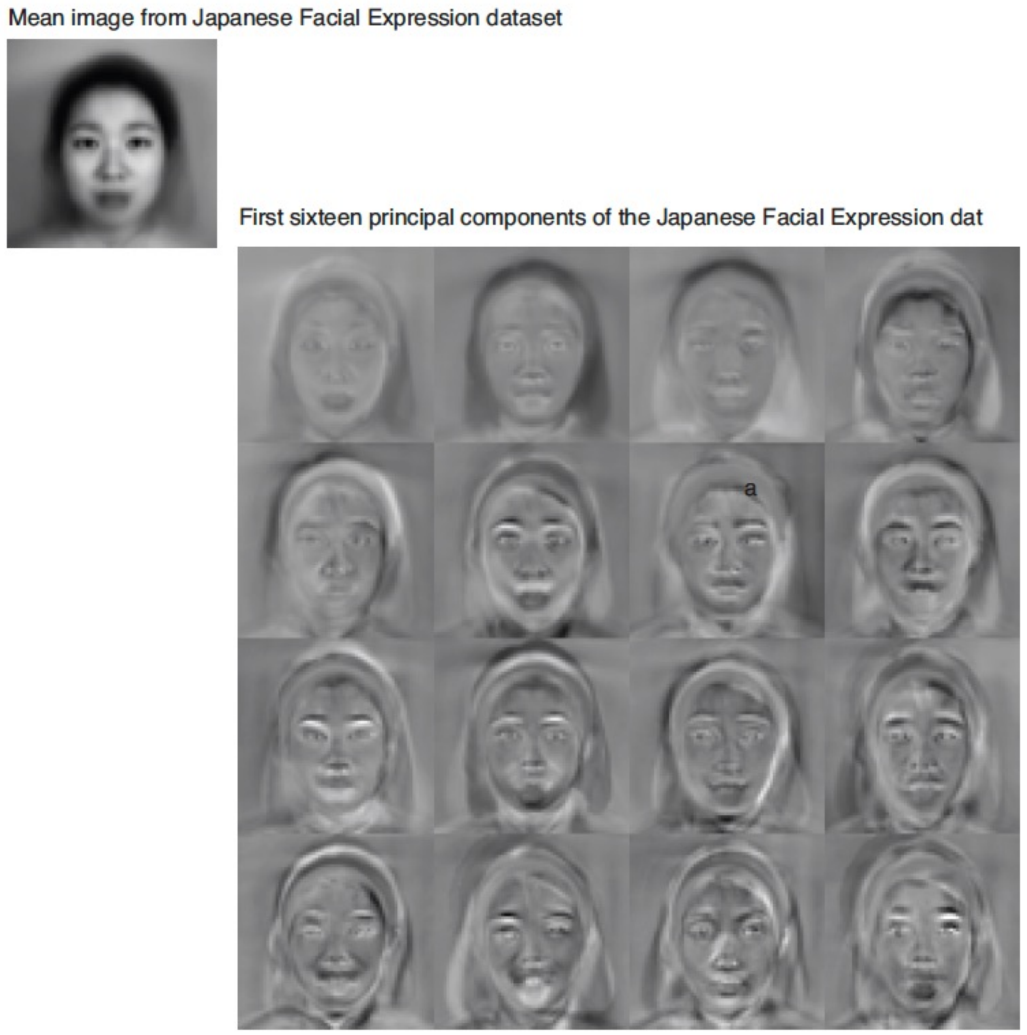
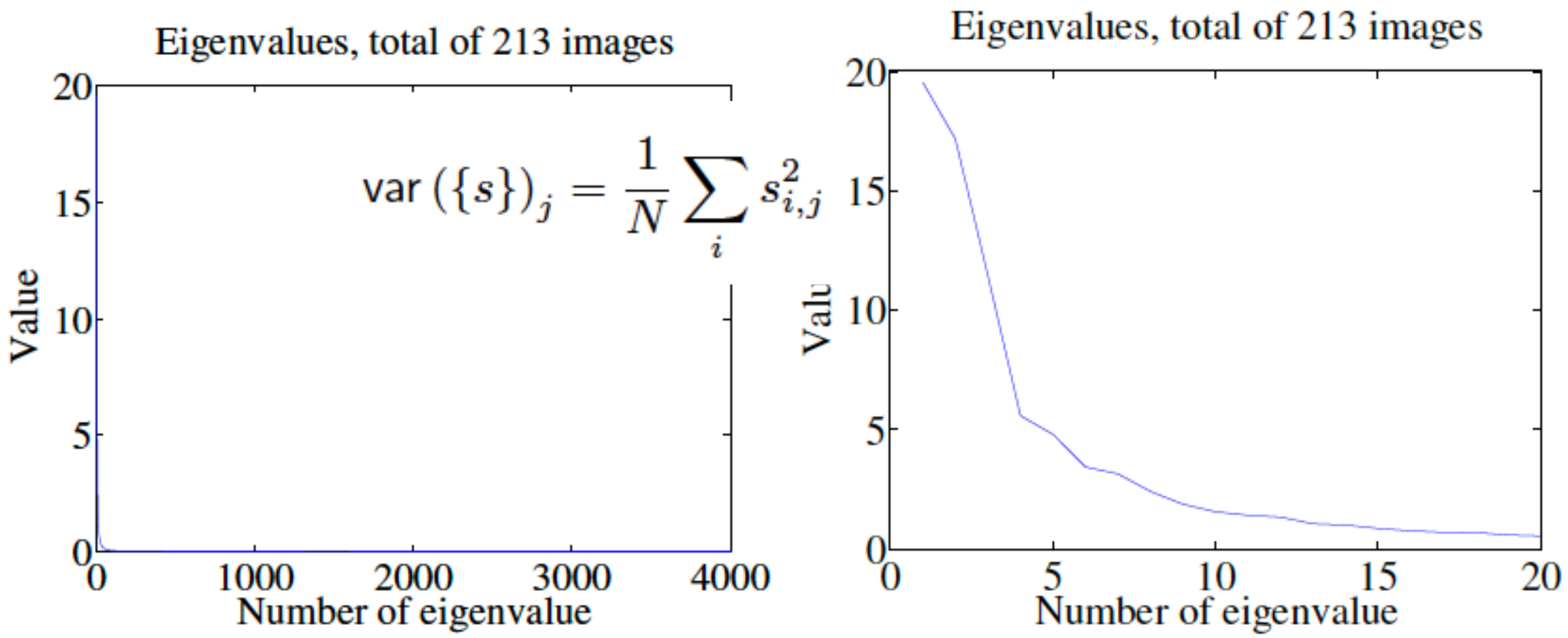

Mean image from Japanese Facial Expression dataset
First sixteen principal components of the Japanese Facial Expression dat
a

FIGURE 2.16: *The mean and first 16 principal components of the Japanese facial expression dataset.*



$$\text{var}\left(\{s\}\right)_j = \frac{1}{N}\sum_i s_{i,j}^2$$

FIGURE 2.15: *On the* **left**,*the eigenvalues of the covariance of the Japanese facial expression dataset; there are 4096, so it's hard to see the curve (which is packed to the left). On the* **right**, *a zoomed version of the curve, showing how quickly the values of the eigenvalues get small.*

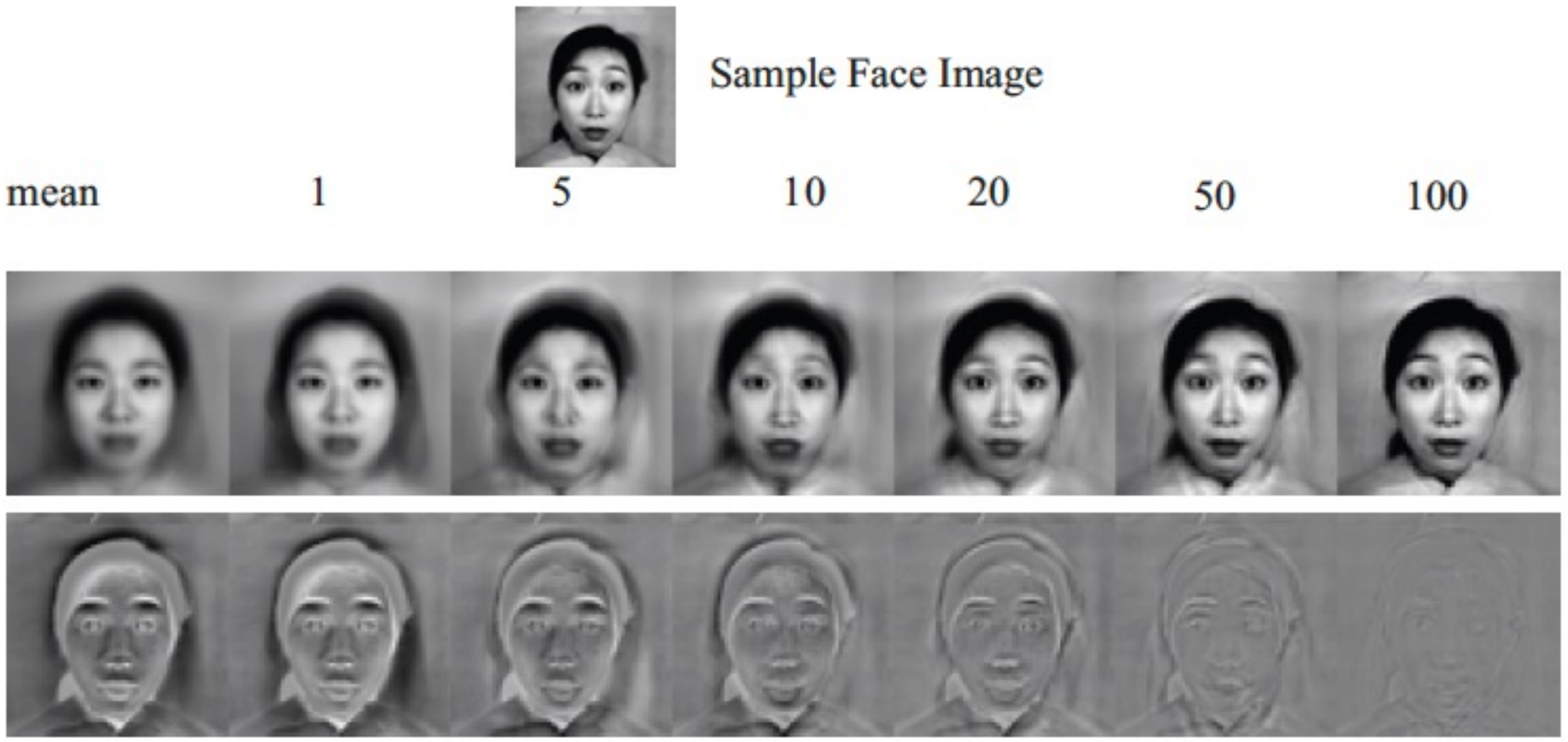


FIGURE 2.17: *Approximating a face image by the mean and some principal components; notice how good the approximation becomes with relatively few components.*