Projects

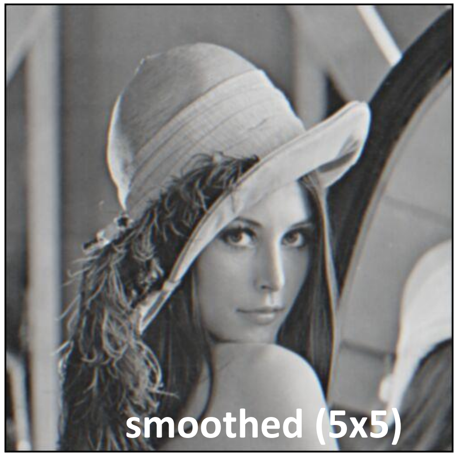
- Project 1a due this Friday

- Project 1b will go out on Friday
  - to be done in pairs
  - start looking for a partner now

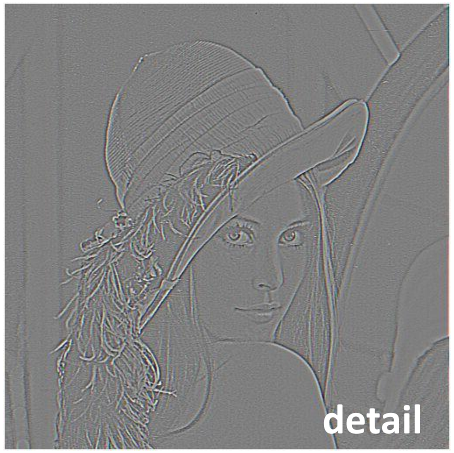
Sharpening revisited

What does blurring take away?

–

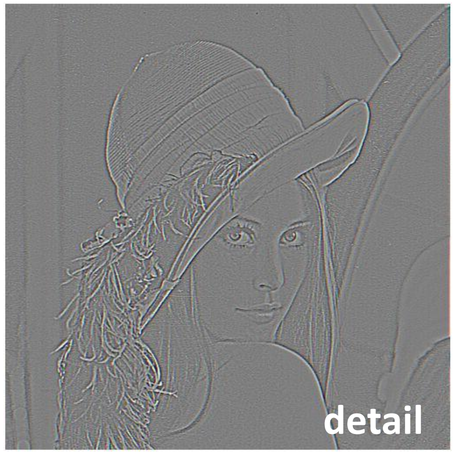


=

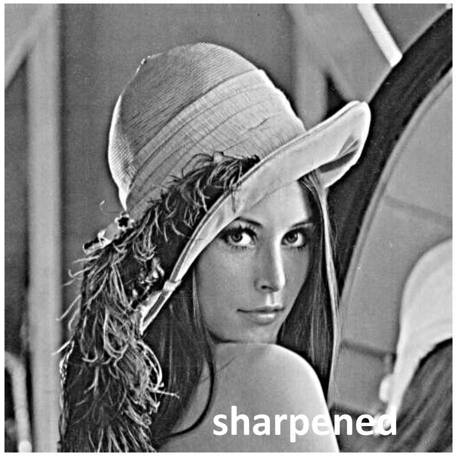


Let’s add it back:

+ α



=



Source: S. Lazebnik

Sharpening revisited

original image

blur kernel

$$A - A * H = B$$

$$A + \alpha B = C$$

sharpened image

$$A + \alpha\left(A - A * H\right) = C$$

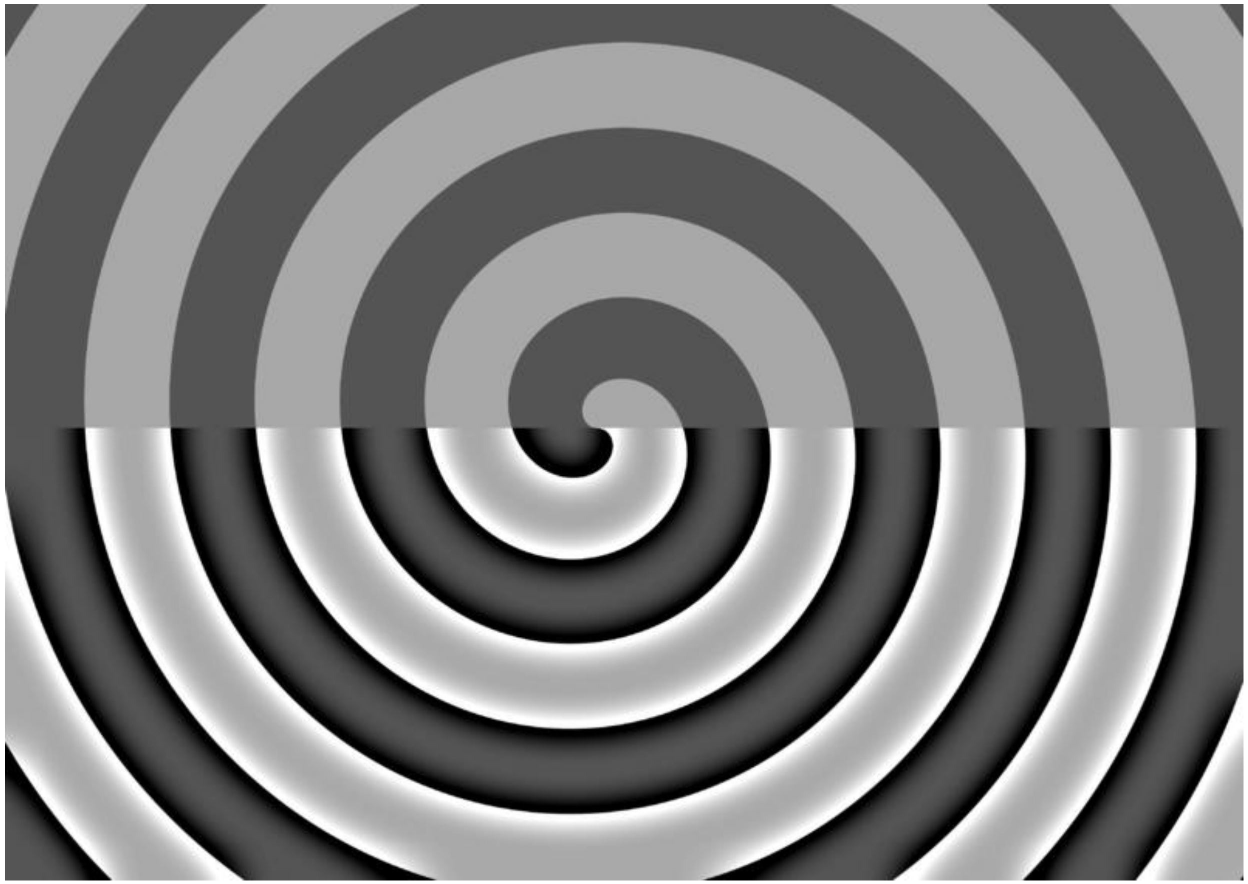
$$A * \delta + \alpha A * \delta - \alpha A * H = C$$

$$A * \boxed{\left(\delta - \alpha\left(H - \delta\right)\right)} = C$$

Sharpen filter

unfiltered

filtered



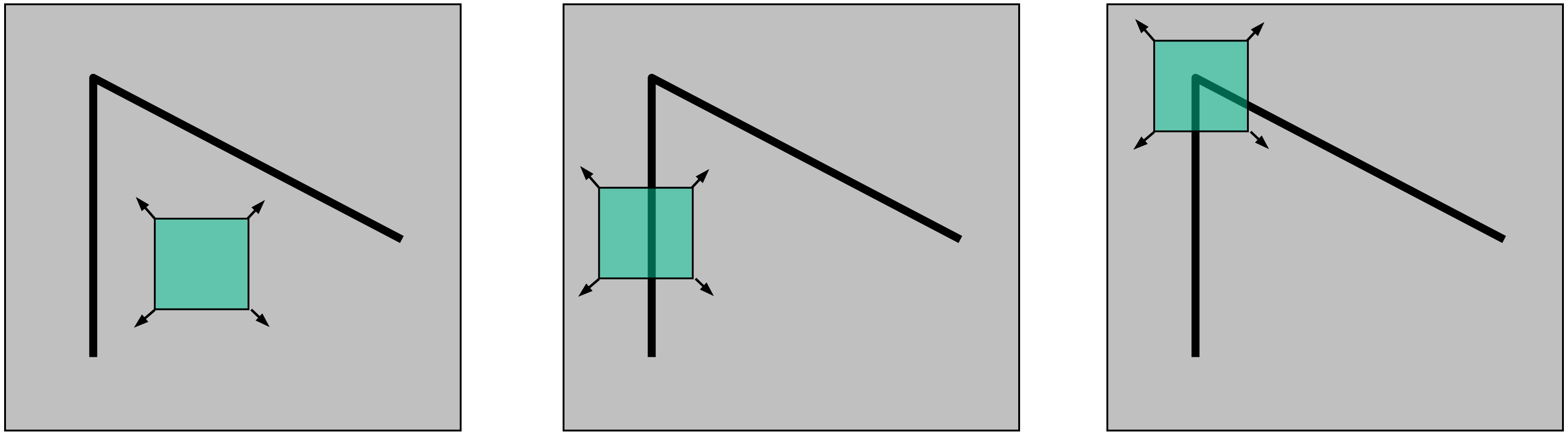
Source: N. Snavely

Corner detection

Corner detection

Local measure of feature uniqueness

- How does the window change when you shift it?
- Shifting the window in *any direction* causes a *big change*



"flat" region:
no change in all directions

"edge":
no change along the edge direction

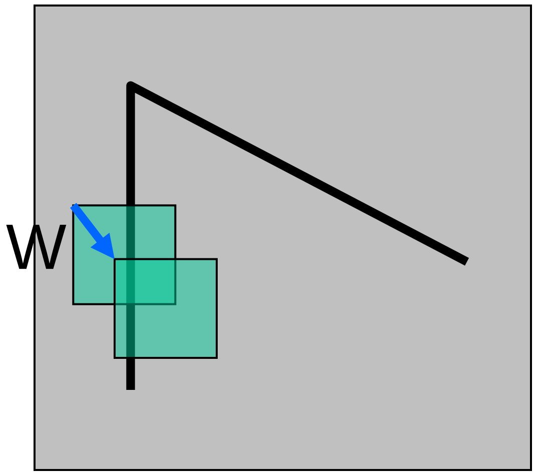
"corner":
significant change in all directions

Slide adapted from Darya Frolova, Denis Simakov, Weizmann Institute.

Corner detection: the math

Consider shifting the window W by (u,v)

- how do the pixels in W change?
- compare each pixel before and after by summing up the squared differences (SSD)
- this defines an SSD "error" of *E(u,v)*:



$$E(u,v) = \sum_{(x,y)\in W} \left[I(x+u,y+v) - I(x,y)\right]^2$$

Small motion assumption

Taylor Series expansion of I:

$$I(x+u, y+v) = I(x,y) + \frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v + \text{higher order terms}$$

If the motion (u,v) is small, then first order approx is good

$$I(x+u, y+v) \approx I(x,y) + \frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v$$

$$\approx I(x,y) + [I_x \ \ I_y] \begin{bmatrix} u \\ v \end{bmatrix}$$

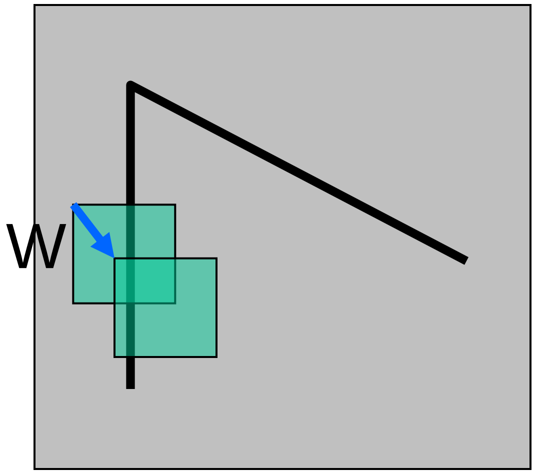
shorthand: $I_x = \frac{\partial I}{\partial x}$

Plugging this into the formula on the previous slide…

Corner detection: the math

Consider shifting the window W by (u,v)

- how do the pixels in W change?
- compare each pixel before and after by summing up the squared differences
- this defines an "error" of E(u,v):

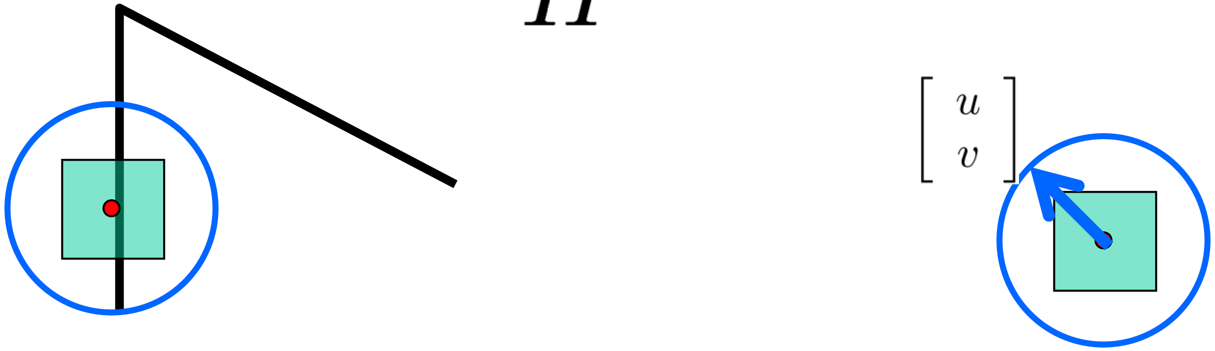


$$
\begin{aligned}
E(u,v) &= \sum_{(x,y)\in W} \left[I(x+u,y+v) - I(x,y)\right]^2 \\
&\approx \sum_{(x,y)\in W} \left[I(x,y) + [I_x \; I_y]\begin{bmatrix} u \\ v \end{bmatrix} - I(x,y)\right]^2 \\
&\approx \sum_{(x,y)\in W} \left[[I_x \; I_y]\begin{bmatrix} u \\ v \end{bmatrix}\right]^2
\end{aligned}
$$

Corner detection: the math

This can be rewritten:

$$E(u,v) = \sum_{(x,y)\in W} [u \ v] \underbrace{\left[\begin{array}{cc} I_x^2 & I_x I_y \\ I_y I_x & I_y^2 \end{array}\right]}_{H} \left[\begin{array}{c} u \\ v \end{array}\right]$$



For the example above

- You can move the center of the green window to anywhere on the blue unit circle
- Which directions will result in the largest and smallest E values?
- We can find these directions by looking at the eigenvectors of *H*

Quick eigenvalue/eigenvector review

The **eigenvectors** of a matrix **A** are the vectors **x** that satisfy:

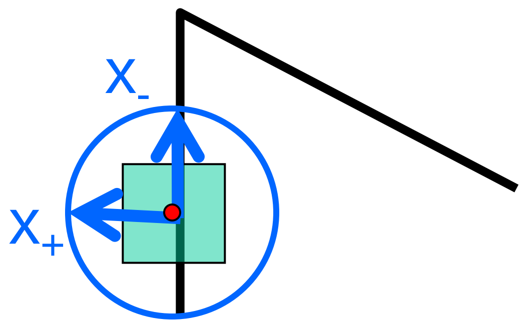
$$Ax = \lambda x$$

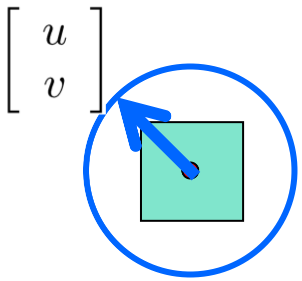
The scalar λ is the **eigenvalue** corresponding to **x**

Corner detection: the math

This can be rewritten:

$$E(u,v) = \sum_{(x,y)\in W} [u\ v] \underbrace{\begin{bmatrix} I_x^2 & I_xI_y \\ I_yI_x & I_y^2 \end{bmatrix}}_{H} \begin{bmatrix} u \\ v \end{bmatrix}$$





Eigenvalues and eigenvectors of H

- Define shifts with the smallest and largest change (E value)
- x_+ = direction of largest increase in E.
- λ_+ = amount of increase in direction x_+
- x_- = direction of smallest increase in E.
- λ- = amount of increase in direction x_+

$$\begin{aligned} Hx_+ &= \lambda_+ x_+ \\ Hx_- &= \lambda_- x_- \end{aligned}$$

Corner detection: the math

How are λ_+, x_+, λ_-, and x_- relevant for corner detection?

- What's our "corneriness" scoring function?

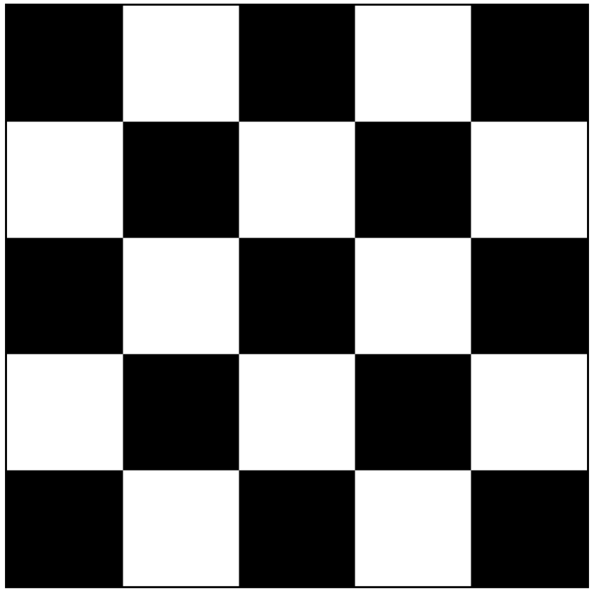
Corner detection: the math

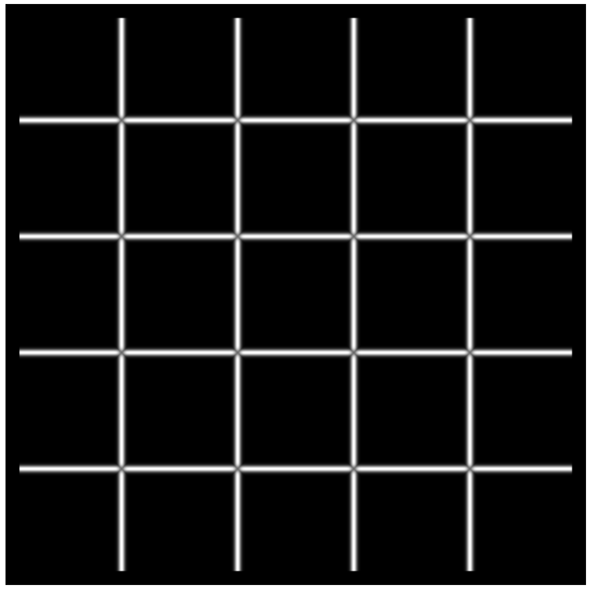
How are λ_+, x_+, λ_-, and x_- relevant for corner detection?

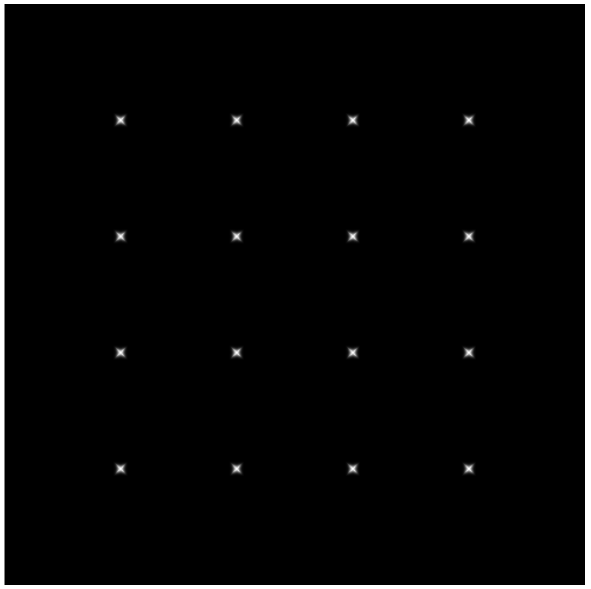
- What's our "corneriness" scoring function?

Want *E(u,v)* to be *large* for small shifts in *all* directions

- the *minimum* of *E(u,v)* should be large, over all unit vectors [u v]
- this minimum is given by the smaller eigenvalue (λ_-) of *H*


I


λ_+


λ_-

Corner detection summary

Here’s what you do

- Compute the gradient at each point in the image
- Create the *H* matrix from the entries in the gradient
- Compute the eigenvalues.
- Find points with large response (λ_- > threshold)
- Choose those points where λ_- is a local maximum as corners

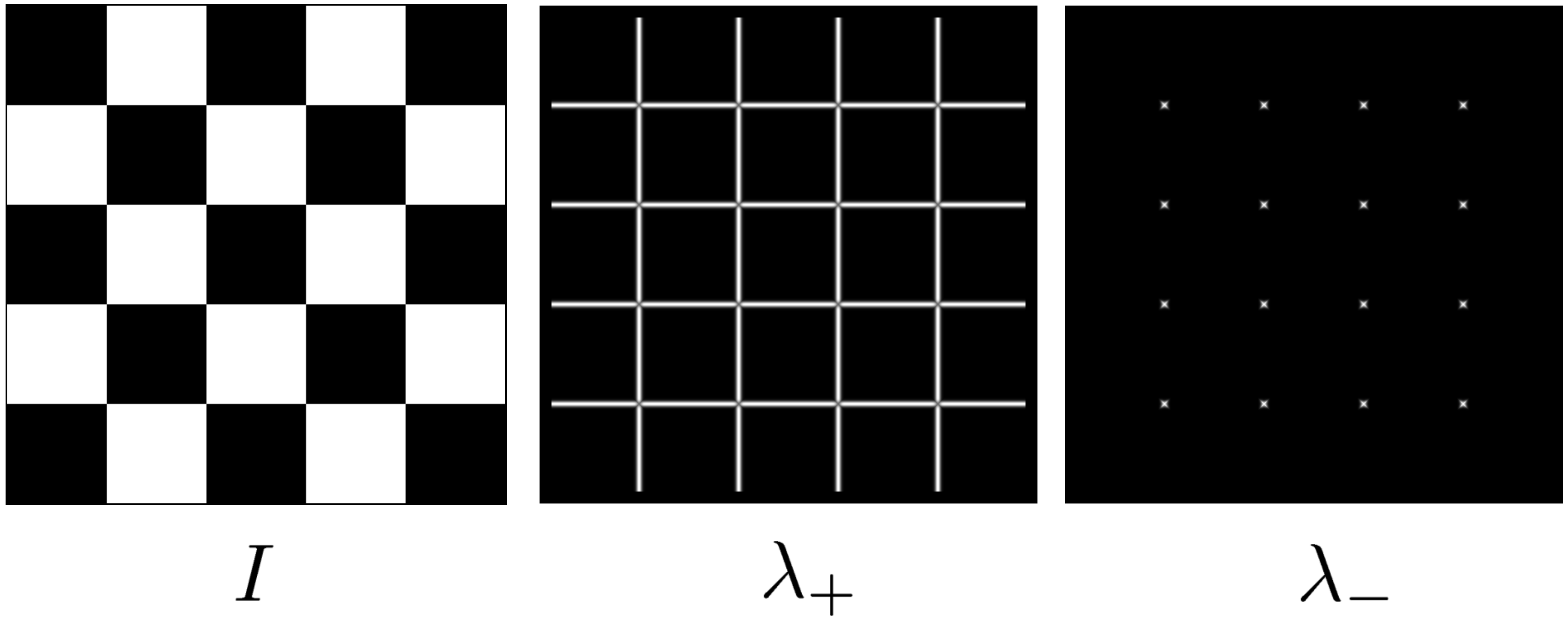


Image Sampling

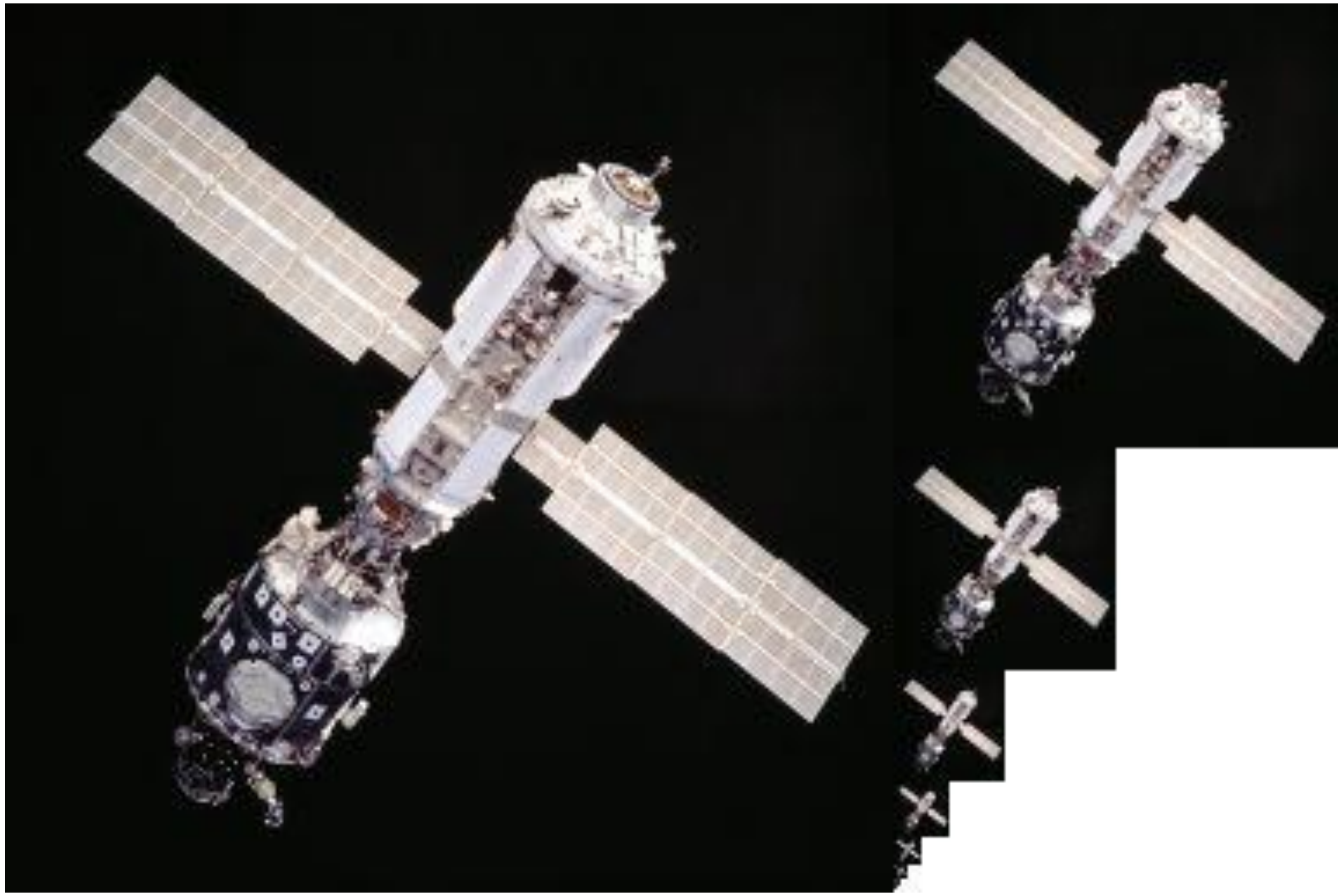
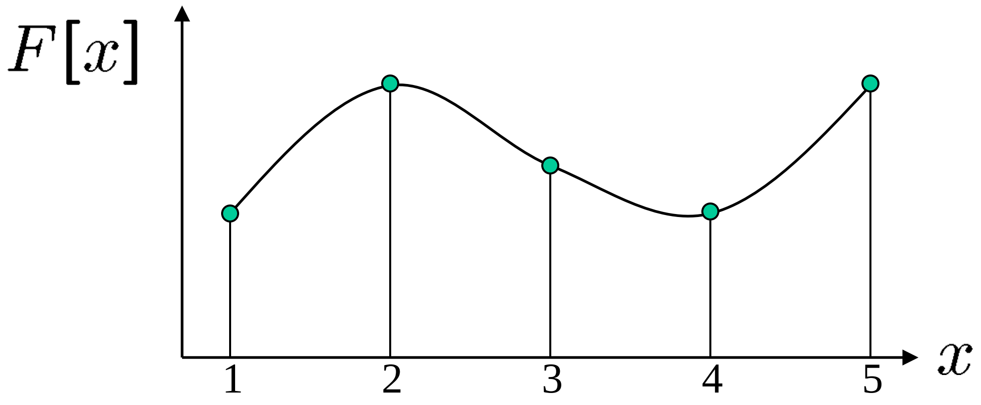


Image resampling

How can we change the size of an image?



here distance d between samples = 1

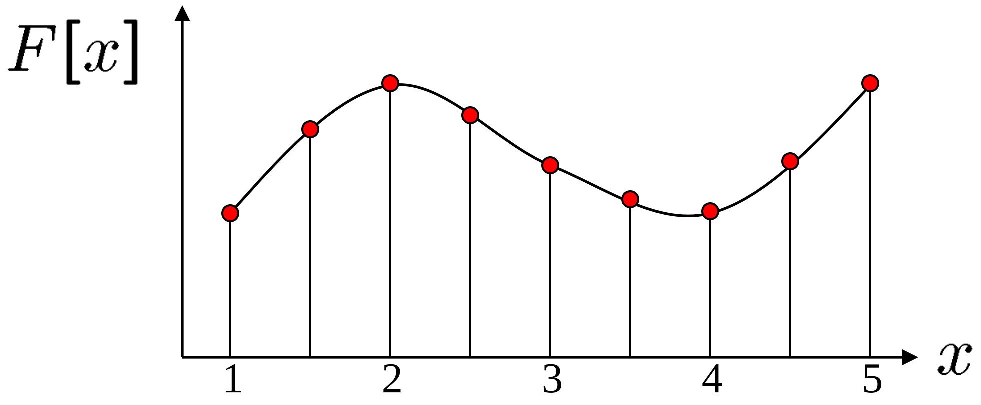
Recall how a digital image is formed

$$F[x, y] = \text{quantize}\{f(xd, yd)\}$$

- It is a discrete point-sampling of a continuous function
- If we could somehow reconstruct the original function, any new image could be generated, at any resolution and scale

Image resampling

How can we change the size of an image?



here distance d
between samples = 1

Recall how a digital image is formed

$$F[x,y] = \text{quantize}\{f(xd, yd)\}$$

- It is a discrete point-sampling of a continuous function
- If we could somehow reconstruct the original function, any new image could be generated, at any resolution and scale

Convolution revisited

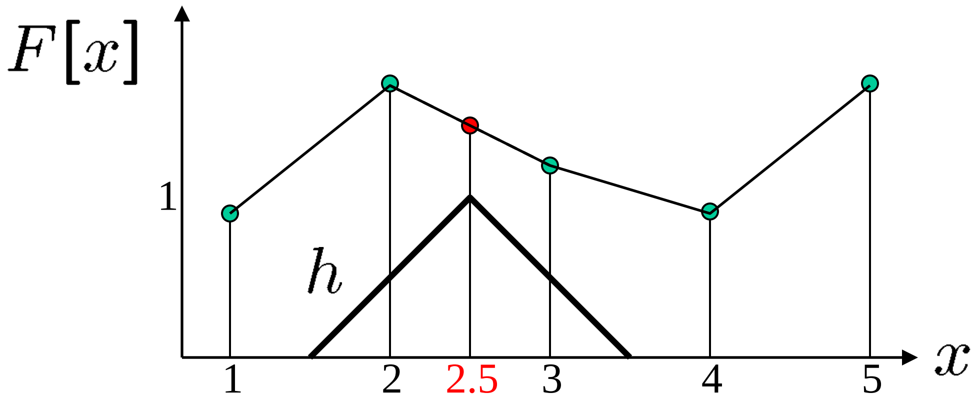
We can also apply filters to *continuous* images.

$$g(x, y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} h(u, v) f(x - u, y - v) du dv$$

Image resampling

So what to do if we don't know f

- Answer: guess an approximation $\tilde{f}$
- Can be done in a principled way: filtering



here distance d between samples = 1

Image reconstruction

- Convert F to a continuous function

$$f_F(x) = F(\tfrac{x}{d}) \text{ when } \tfrac{x}{d} \text{ is an integer, 0 otherwise}$$

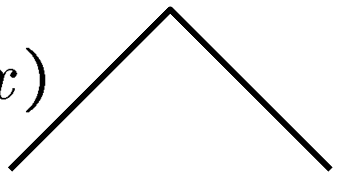
- Reconstruct by cross-correlation:

$$\tilde{f} = h \otimes f_F$$

Resampling filters

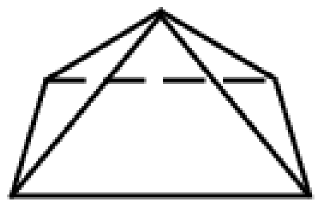
What does the 2D version of this hat function look like?

$h(x)$

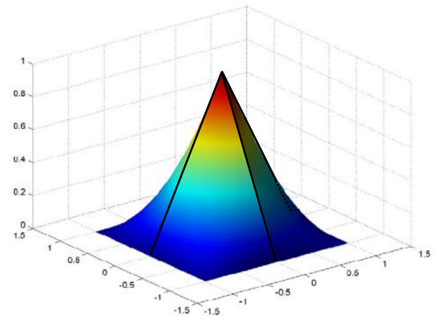


performs
linear interpolation

$h(x, y)$



(tent function) performs
bilinear interpolation

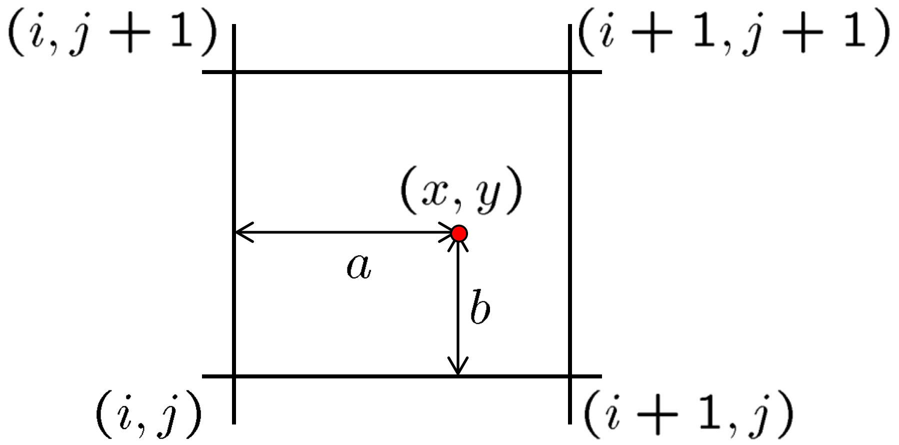


Better filters give better resampled images

- Bicubic is common choice
  - fit 3rd degree polynomial surface to pixels in neighborhood
  - can also be implemented by a convolution

Bilinear interpolation

A simple method for resampling images



$$
\begin{aligned}
f(x, y) = \quad & \underline{\qquad\qquad\qquad} \; f[i, j] \\
& \underline{\qquad\qquad\qquad} \; f[i+1, j] \\
& \underline{\qquad\qquad\qquad} \; f[i+1, j+1] \\
& \underline{\qquad\qquad\qquad} \; f[i, j+1]
\end{aligned}
$$

Downsampling

This image is too big to fit on the screen. How can we reduce it?

How to generate a half-sized version?

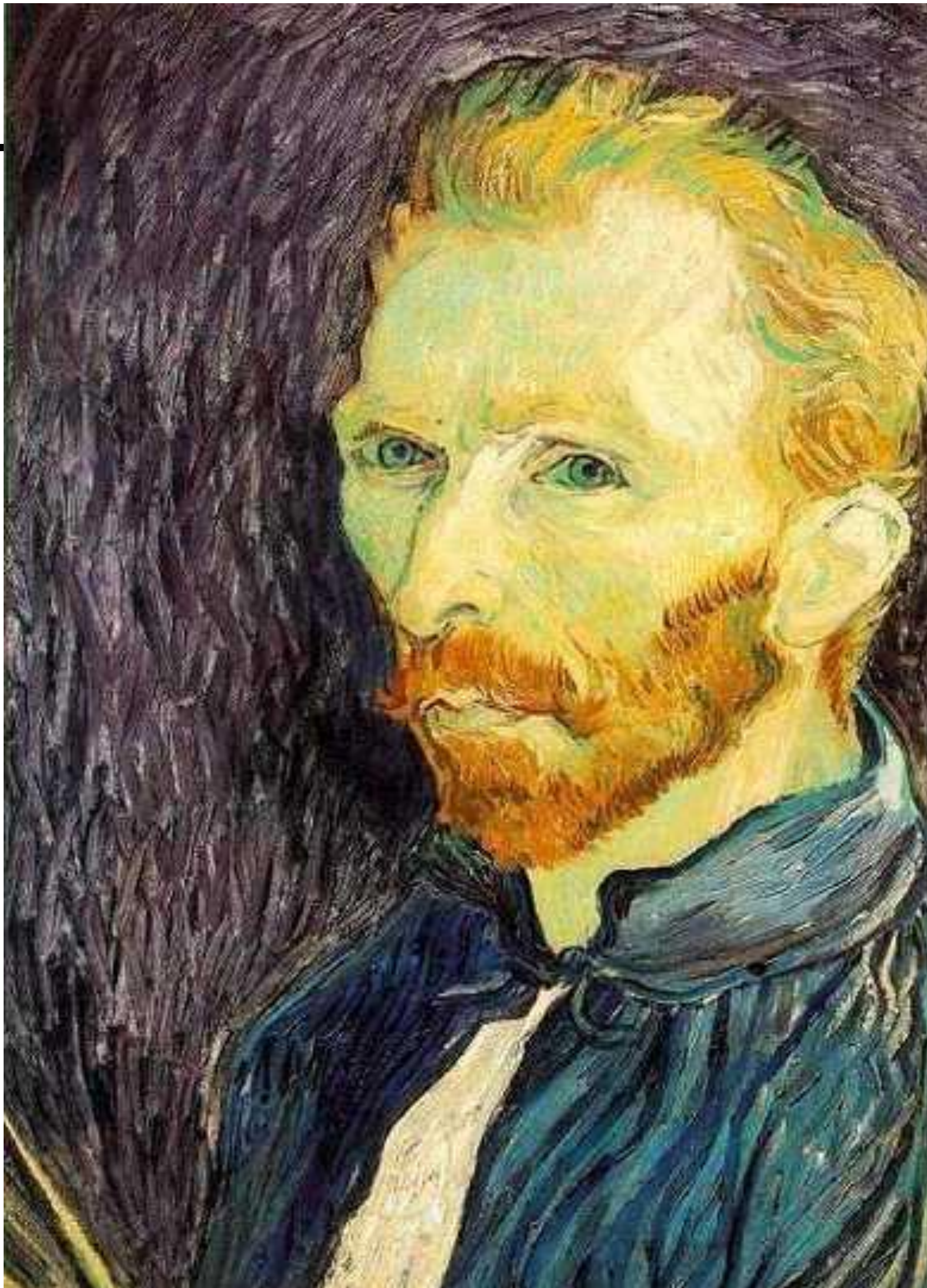
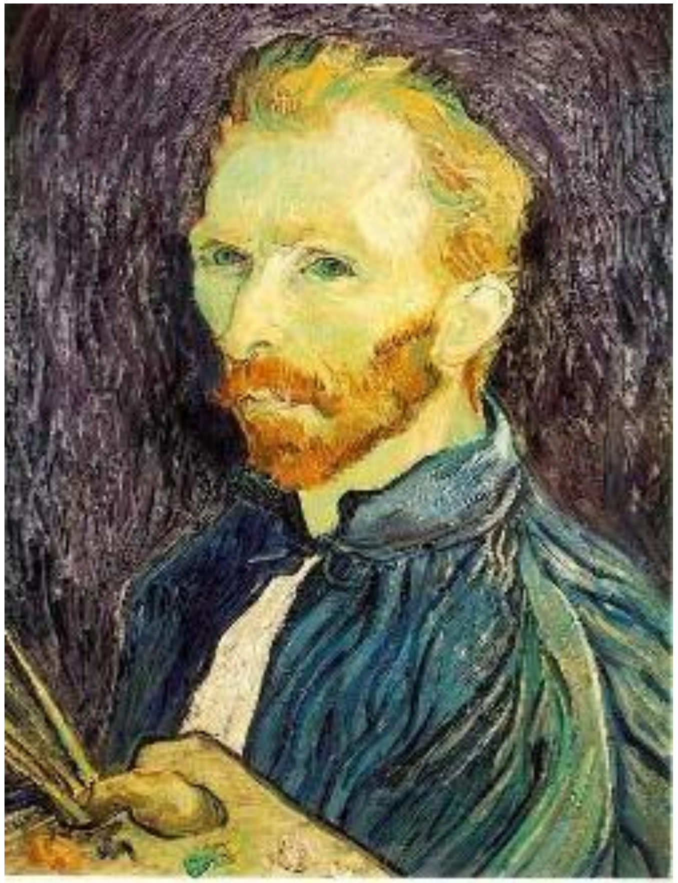


Image sub-sampling

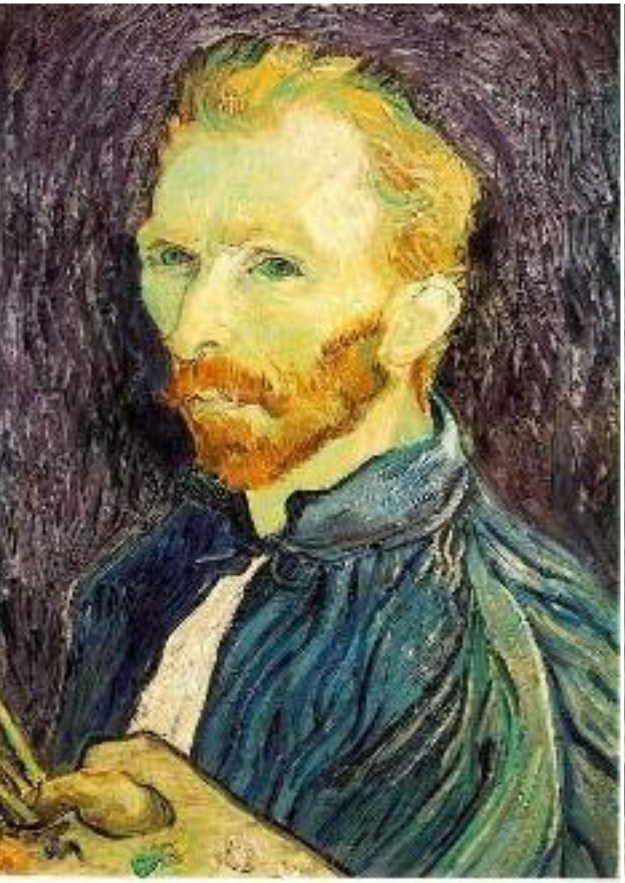


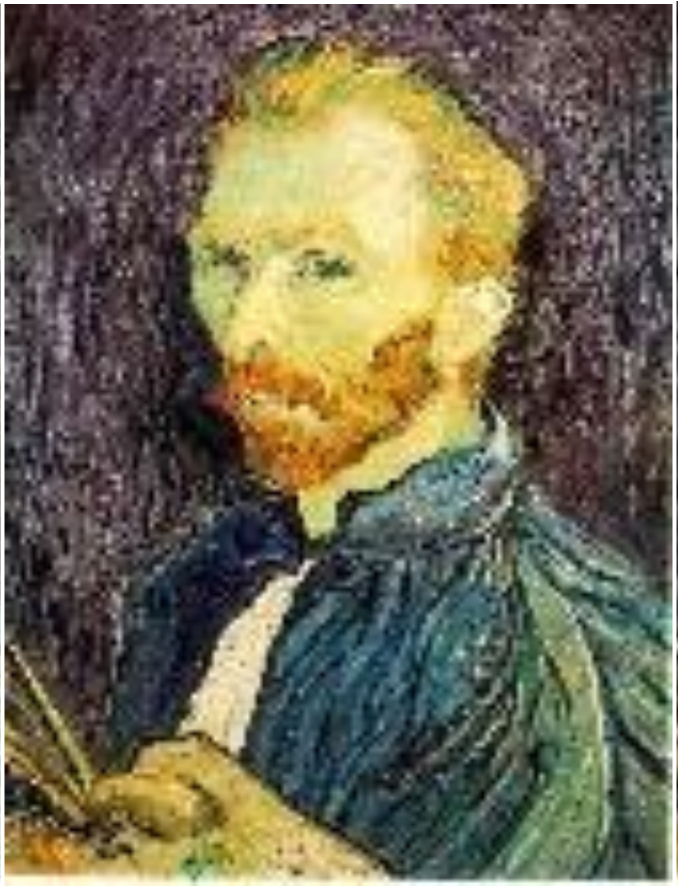
1/4

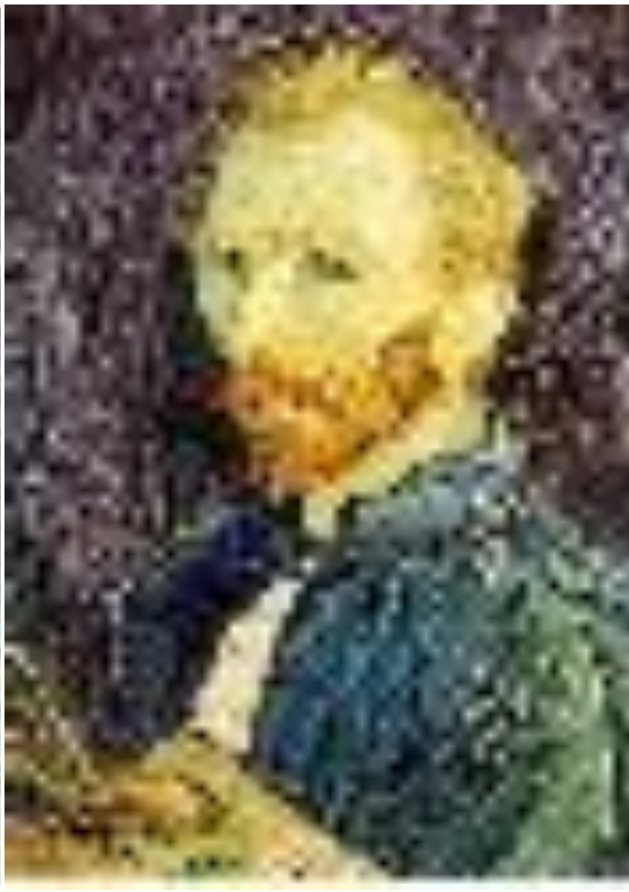
1/8

Throw away every other row and column to create a 1/2 size image
- called *image sub-sampling*

Image sub-sampling

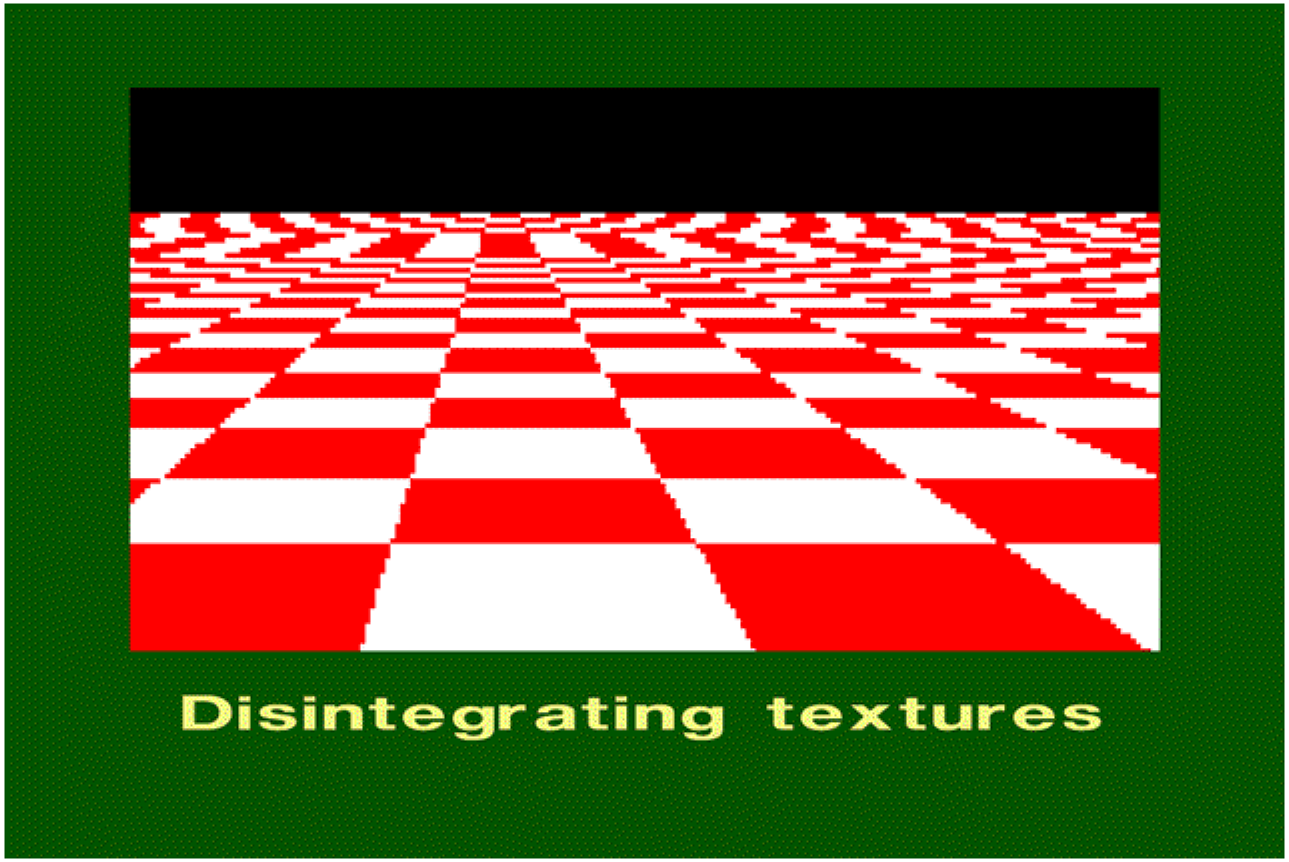

1/2


1/4 (2x zoom)

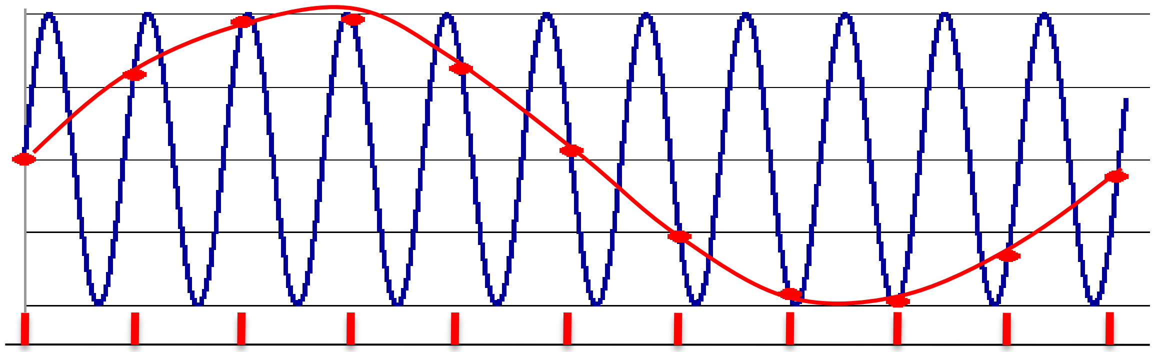

1/8 (4x zoom)

Why does this look so crufty?

Even worse for synthetic images



Sampling and the Nyquist rate



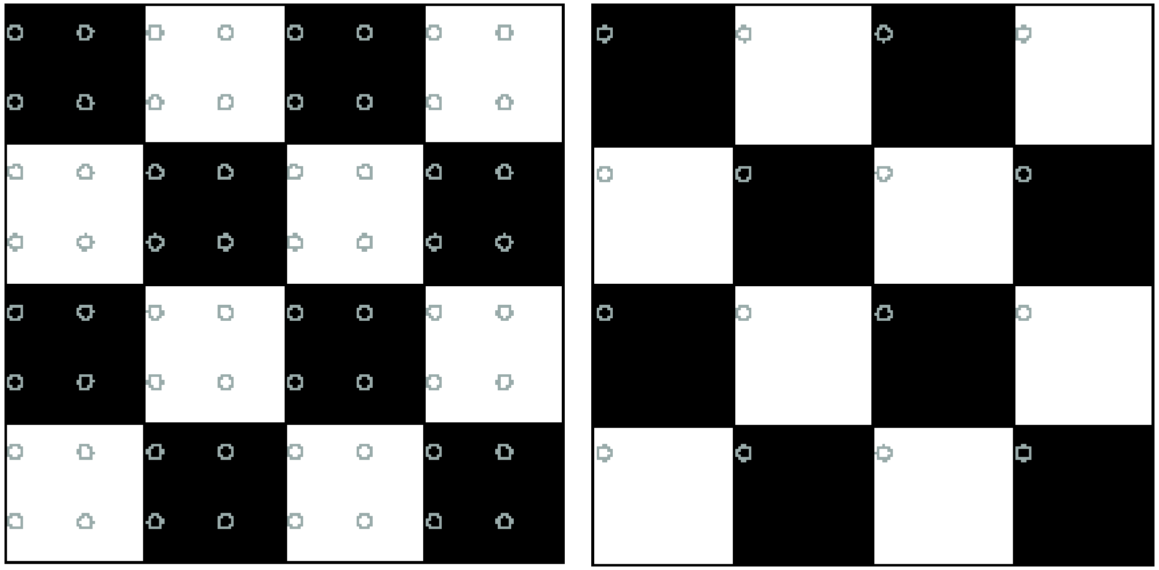
Aliasing can arise when you sample a continuous signal or image

- occurs when your sampling rate is not high enough to capture the amount of detail in your image
- Can give you the wrong signal/image—an *alias*
- formally, the image contains structure at different scales
  - called “frequencies” in the Fourier domain
- the sampling rate must be high enough to capture the highest frequency in the image

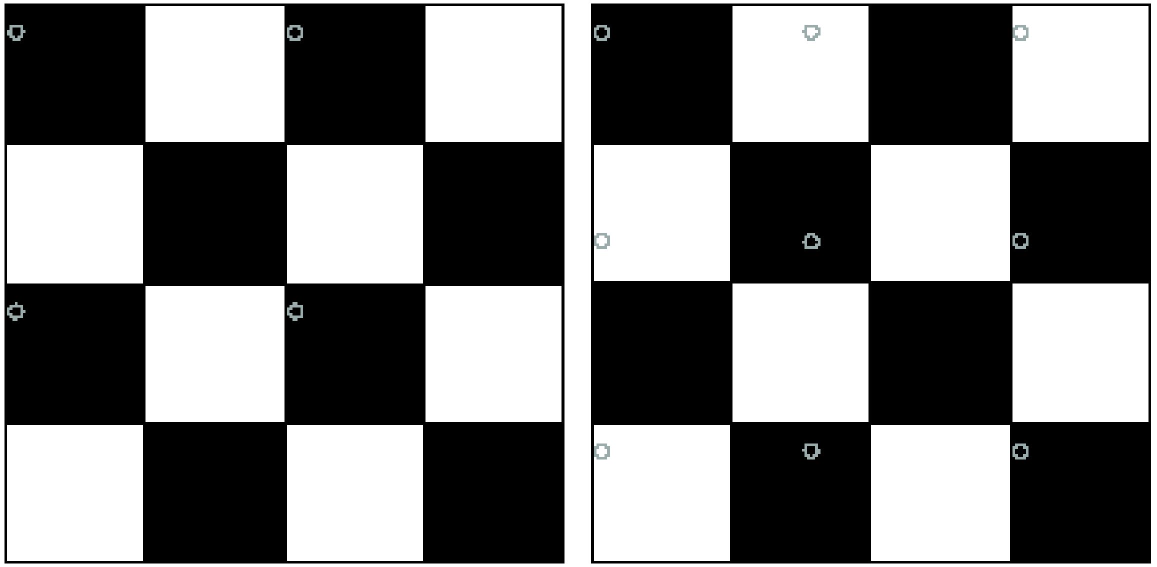
To avoid aliasing:

- sampling rate ≥ 2 * max frequency in the image
  - said another way: ≥ two samples per cycle
- This minimum sampling rate is called the **Nyquist rate**

2D example

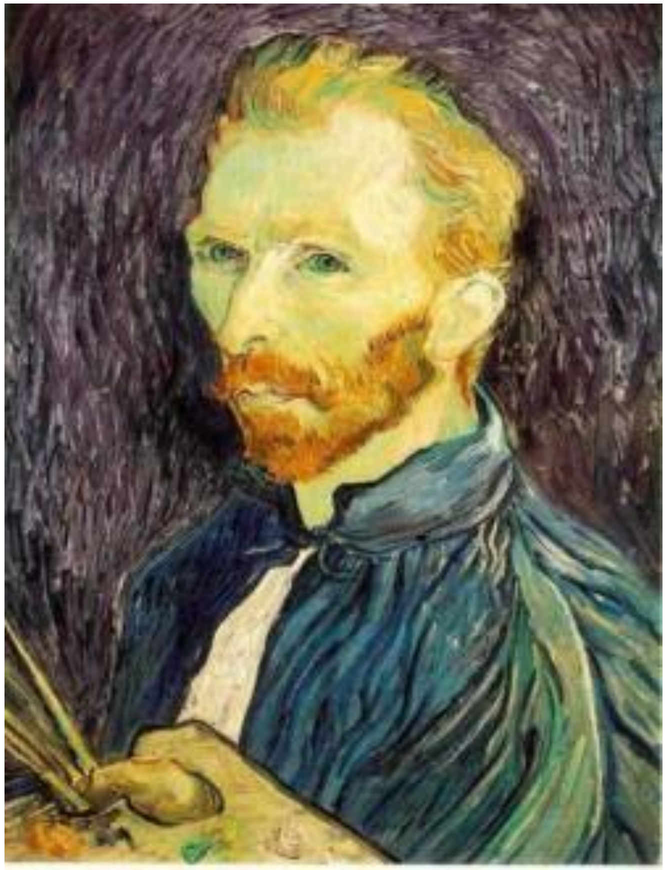


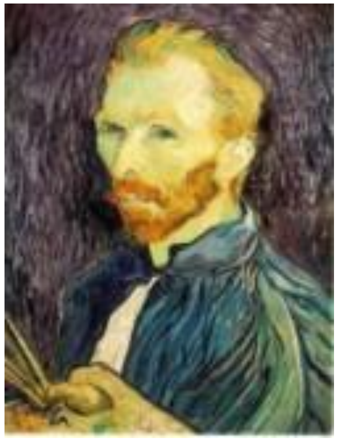
Good sampling

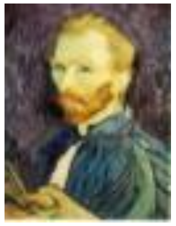


Bad sampling

Subsampling with Gaussian pre-filtering

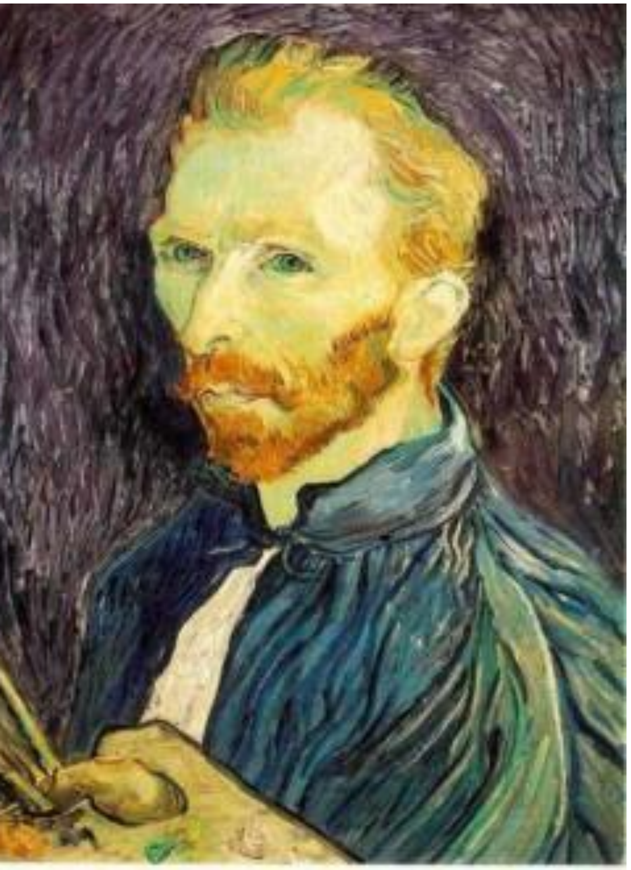
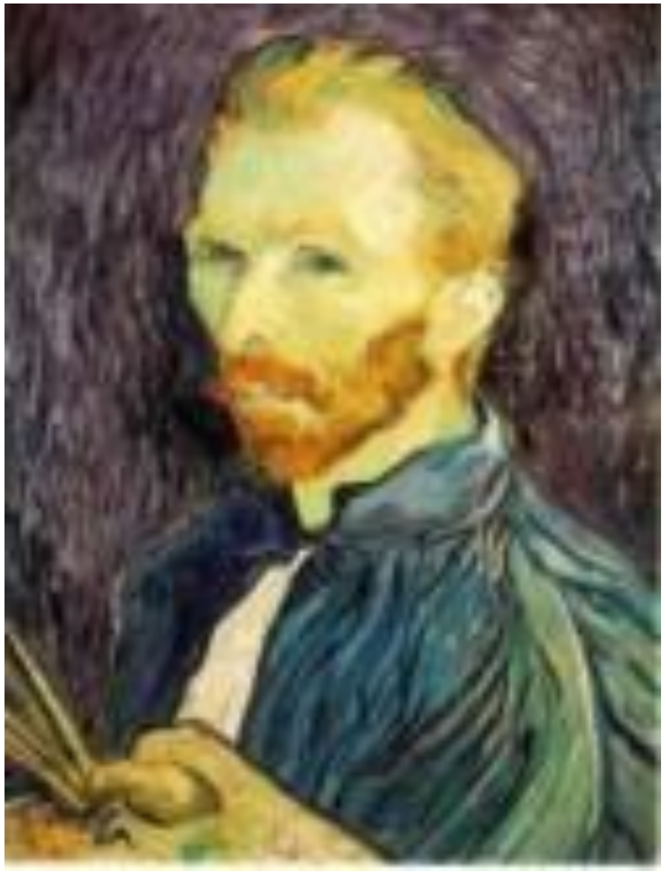
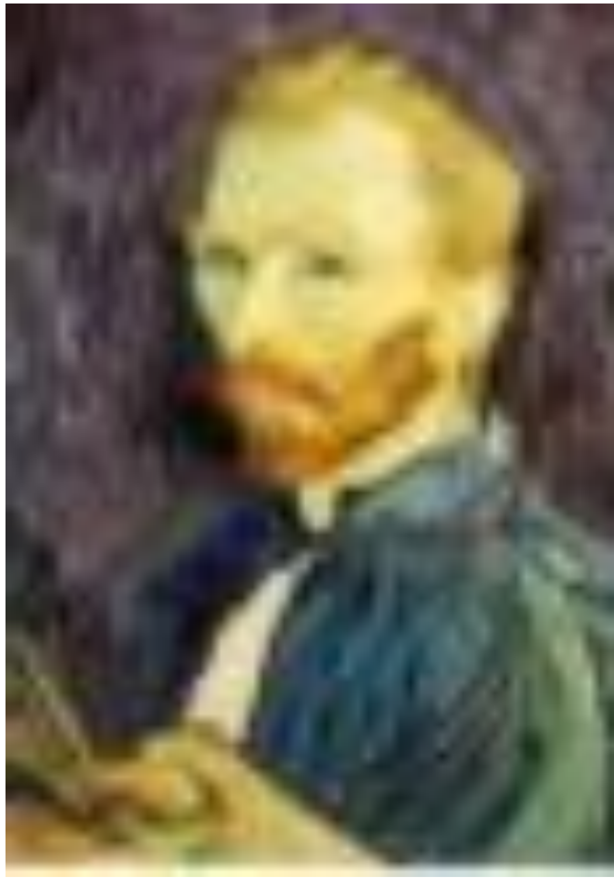

Gaussian 1/2


G 1/4


G 1/8

Solution: filter the image, *then* subsample

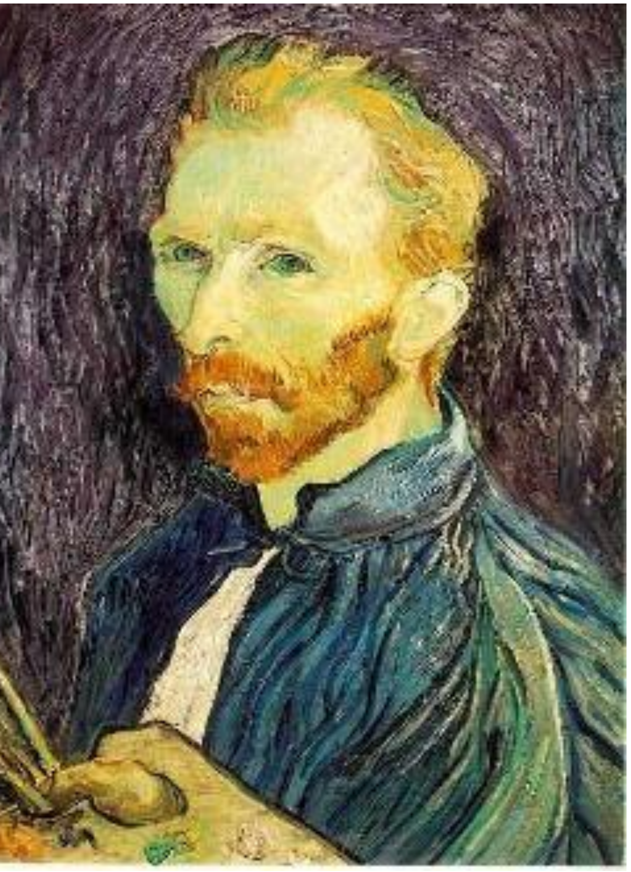
Subsampling with Gaussian pre-filtering

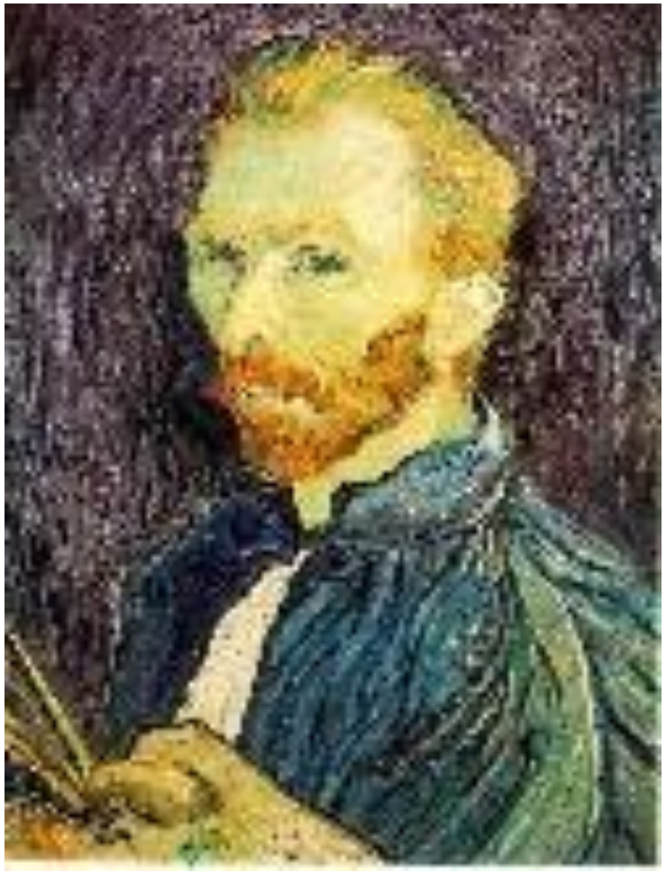




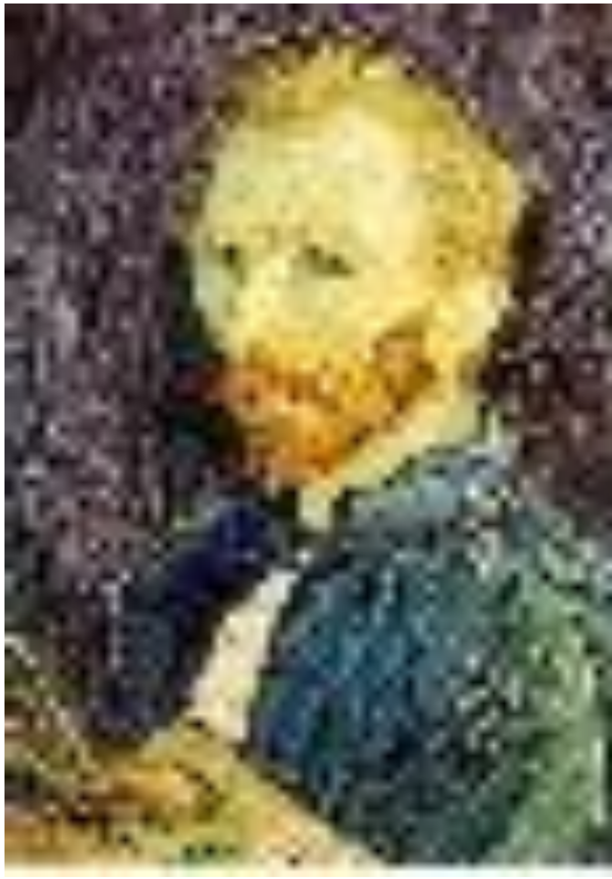
Gaussian 1/2 G 1/4 G 1/8

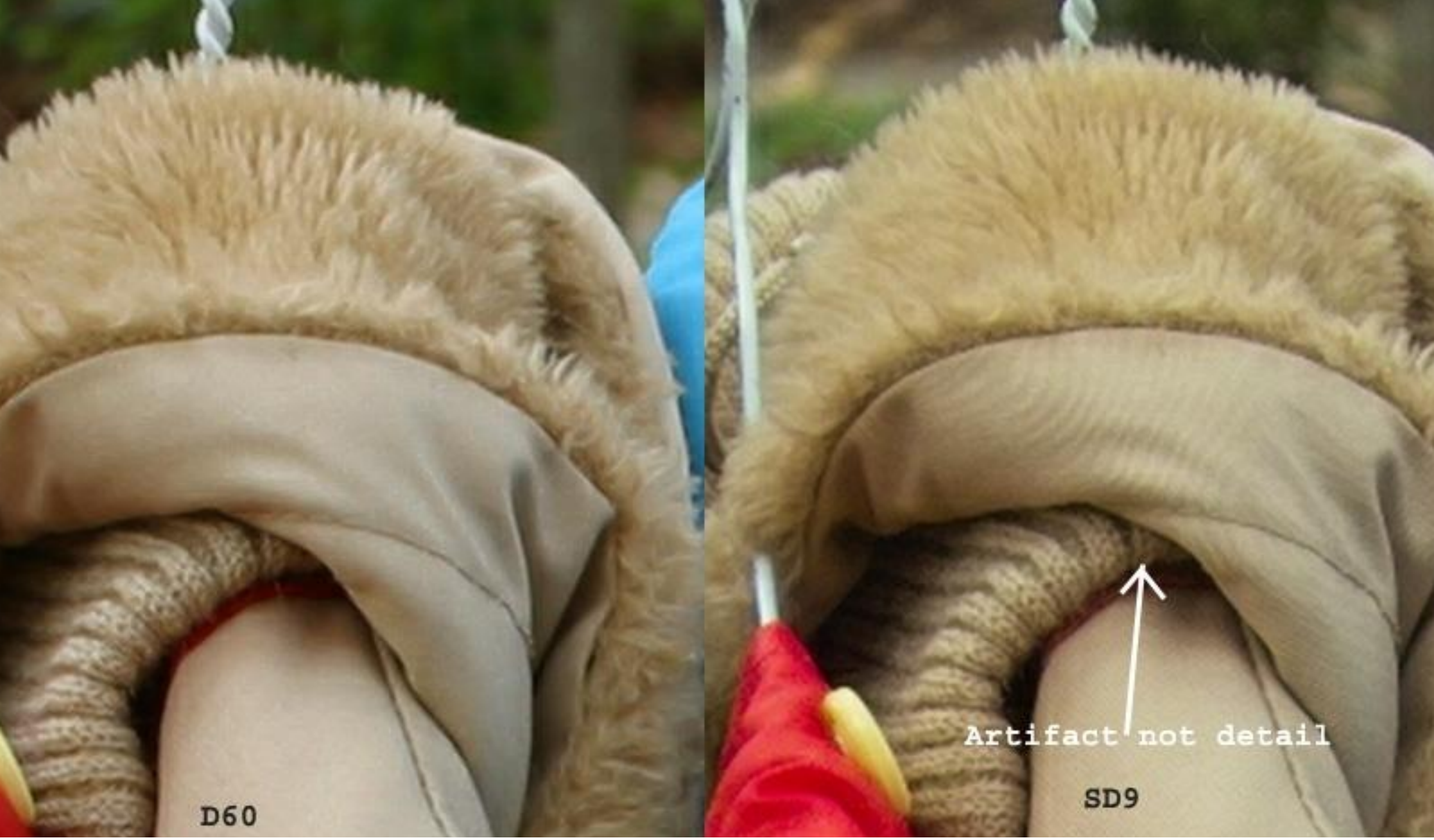
Solution: filter the image, *then* subsample

Compare with...


1/2

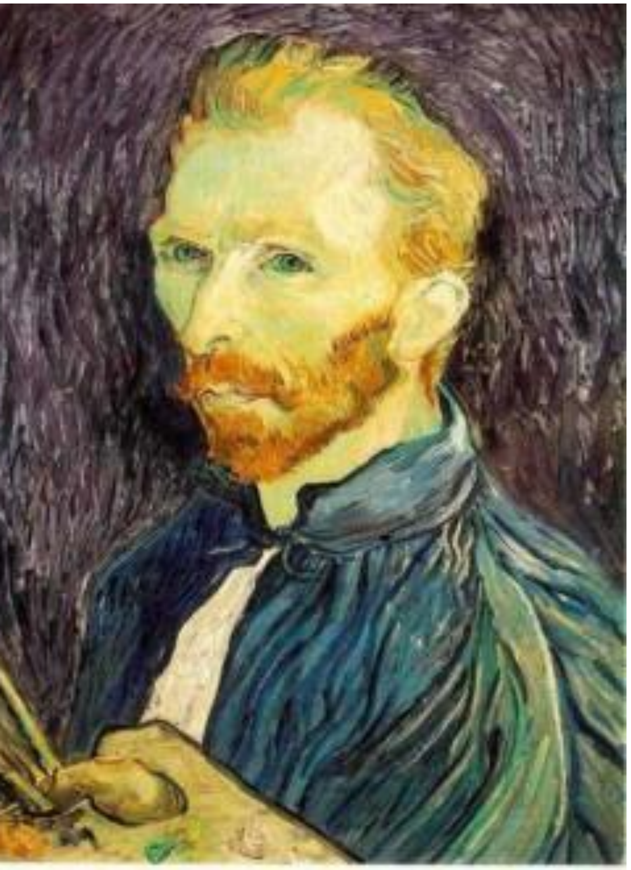

1/4 (2x zoom)

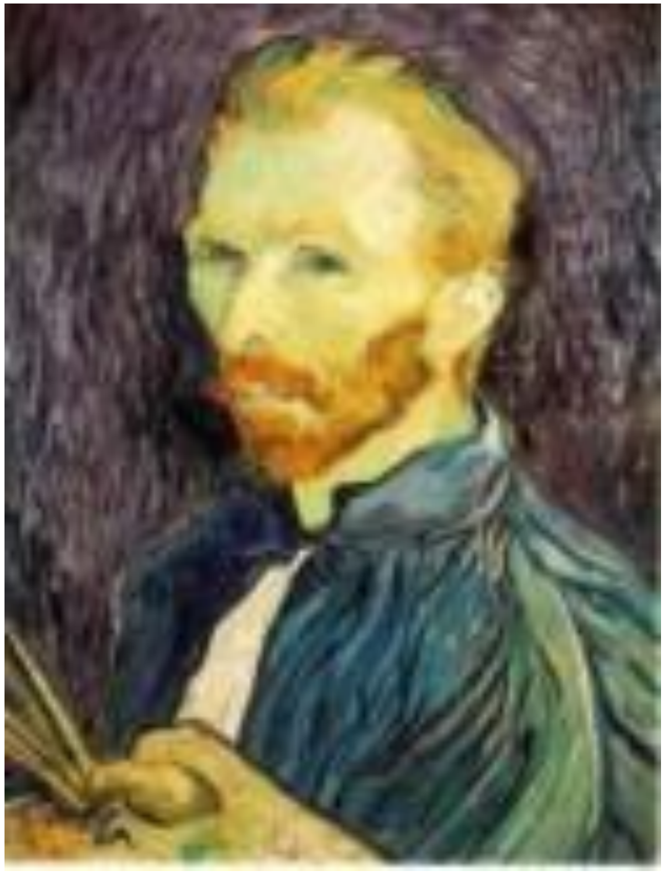

1/8 (4x zoom)

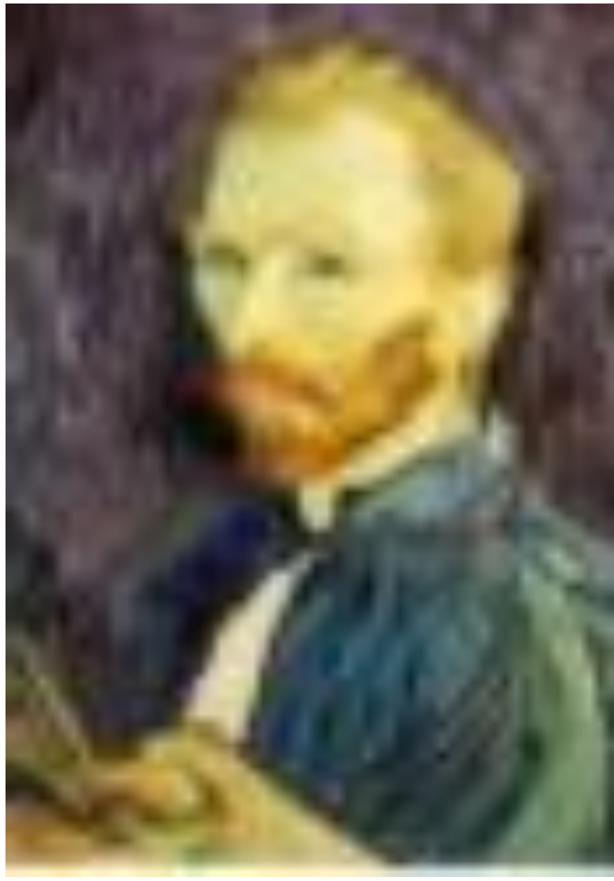


Moire patterns in real-world images. Here are comparison images by Dave Etchells of Imaging Resource using the Canon D60 (with an antialias filter) and the Sigma SD-9 (which has no antialias filter). The bands below the fur in the image at right are the kinds of artifacts that appear in images when no antialias filter is used. Sigma chose to eliminate the filter to get more sharpness, but the resulting apparent detail may or may not reflect features in the image.

Subsampling with Gaussian pre-filtering


Gaussian 1/2

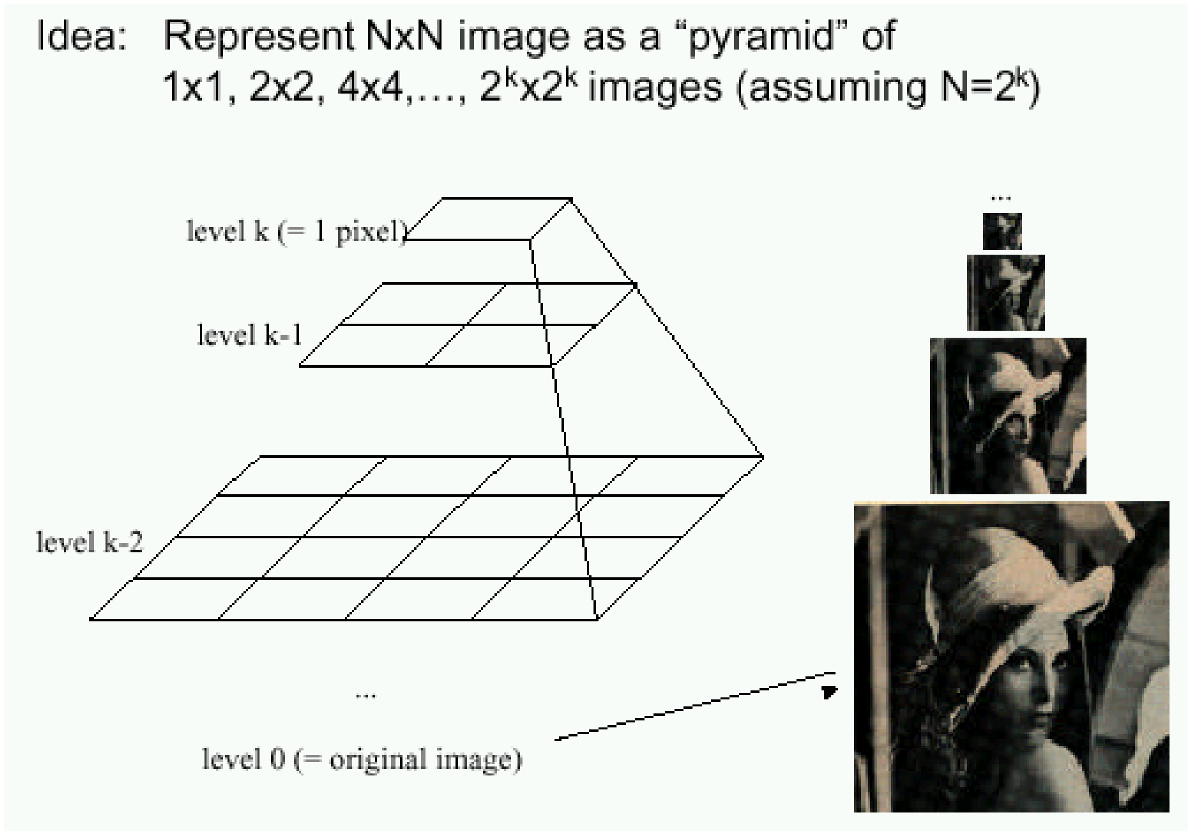

G 1/4


G 1/8

Solution: filter the image, *then* subsample

- Filter size should double when desired image size is halved. Why?
- How can we speed this up?

Some times we want many resolutions


Idea: Represent NxN image as a "pyramid" of
1x1, 2x2, 4x4,…, $2^k x 2^k$ images (assuming $N=2^k$)
level k (= 1 pixel)
level k-1
level k-2
...
level 0 (= original image)
...

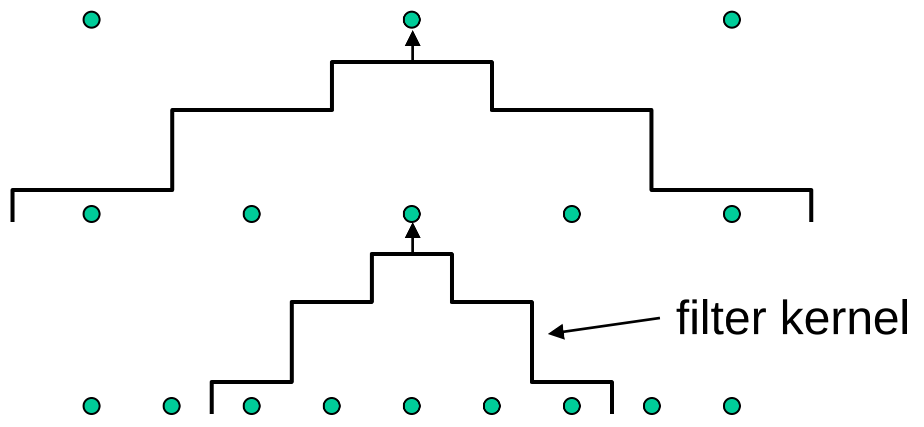
Known as a **Gaussian Pyramid** [Burt and Adelson, 1983]

- In computer graphics, a *mip map* [Williams, 1983]

Gaussian Pyramids have all sorts of applications in computer vision

- We'll talk about these later in the course

Gaussian pyramid construction



Repeat

- Filter
- Subsample

Until minimum resolution reached

- can specify desired number of levels (e.g., 3-level pyramid)

How big is the pyramid?

Each level is ¼ the size of the previous one.

If original image has *N* pixels, then total size is:

$$N + \frac{N}{4} + \frac{N}{16} + \frac{N}{64} + \cdots$$

$$N \sum_{i=0}^{k} \frac{1}{4^{i}}$$

$$< N \left(\frac{1}{1 - 1/4} \right) = \frac{4}{3} N$$

The whole pyramid is only 4/3 the size of the original image!