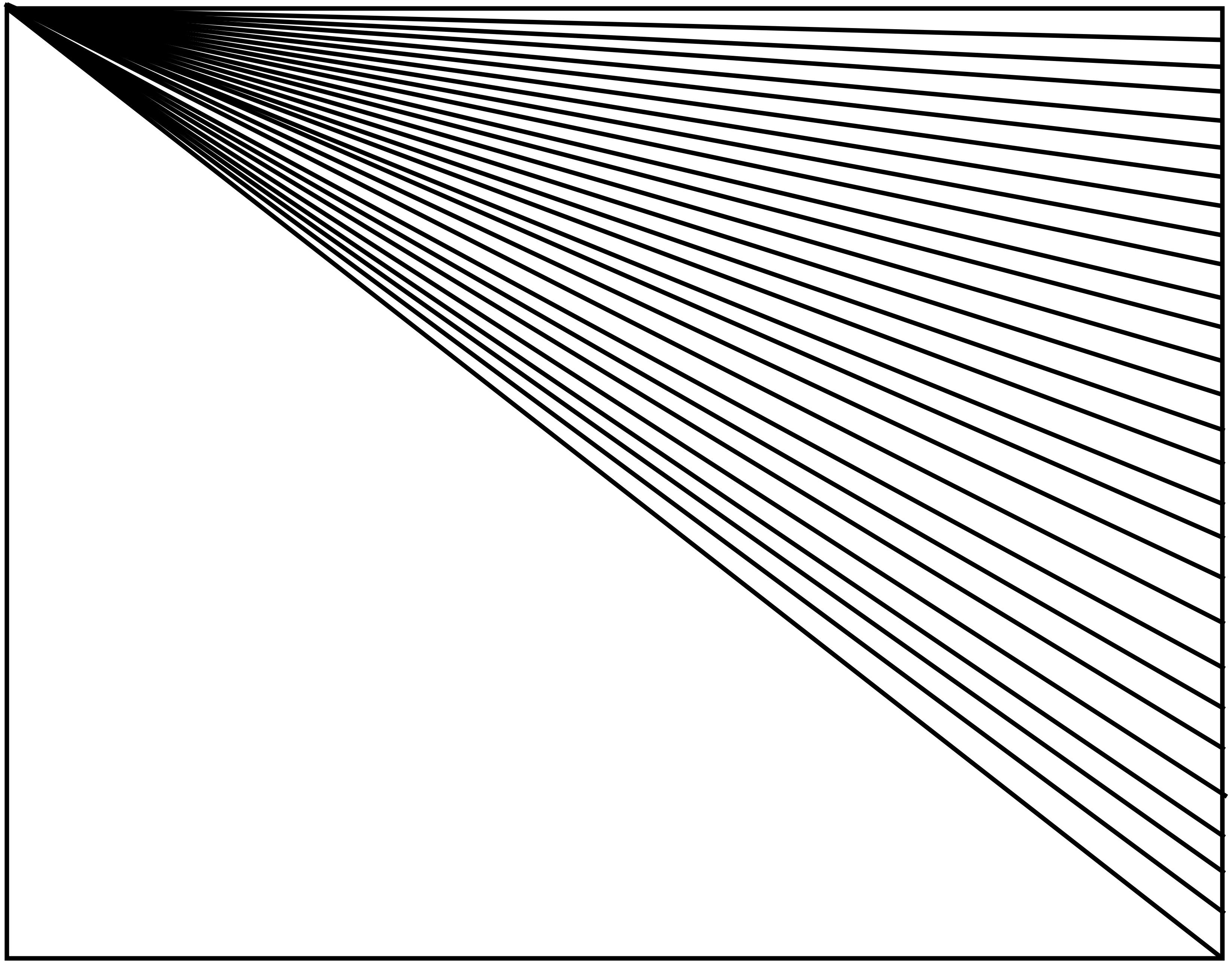
Resizing and resampling

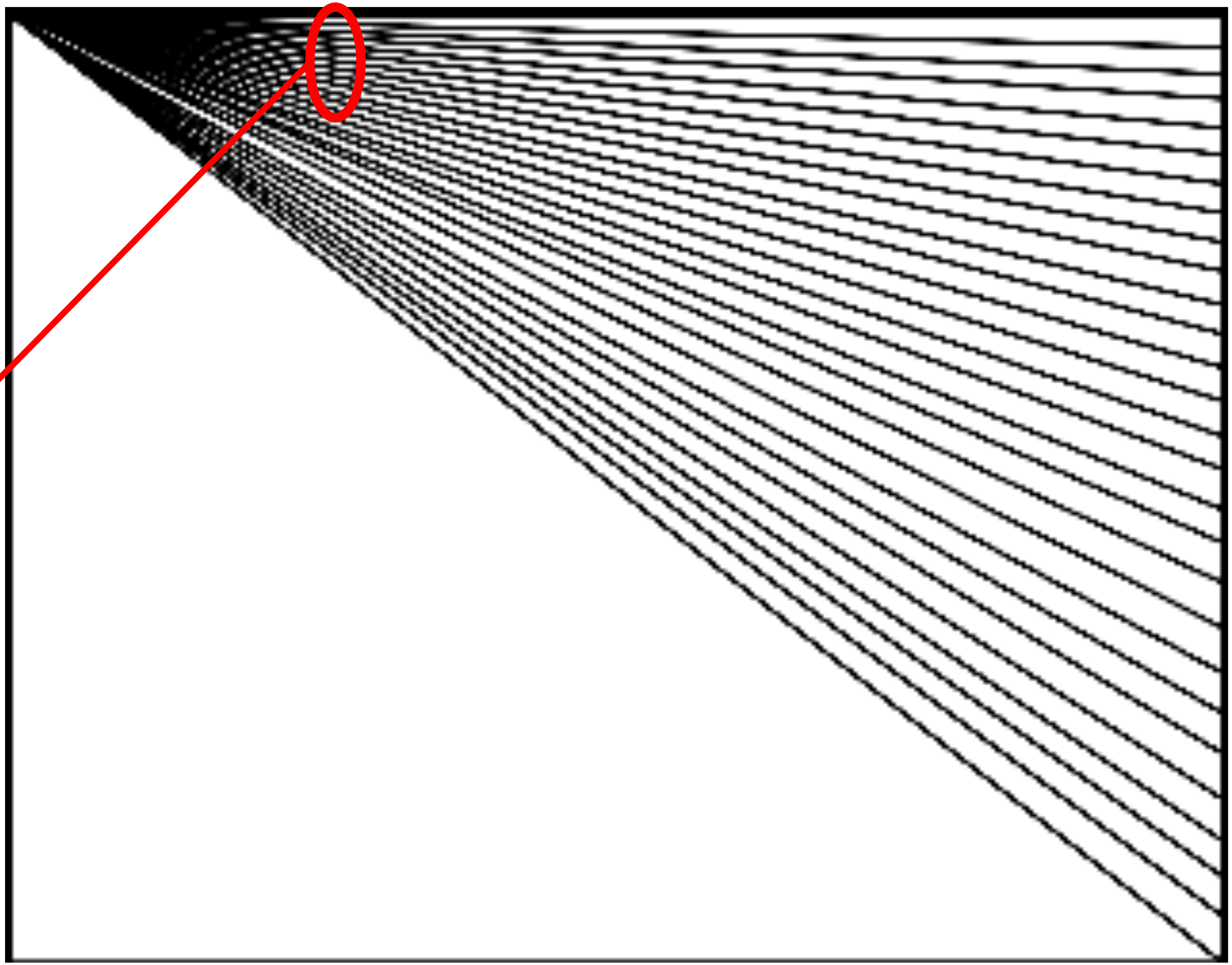
Aliasing

- Images are made up of high frequency and low frequency components
- High frequency components: pixel-to-pixel details
- Low frequency components: high-level structure
- What subsampling should do: remove pixel-to-pixel details, keep high-level structure
- What naïve subsampling does: converts pixel-to-pixel details to new coarse structures → problem

Aliasing

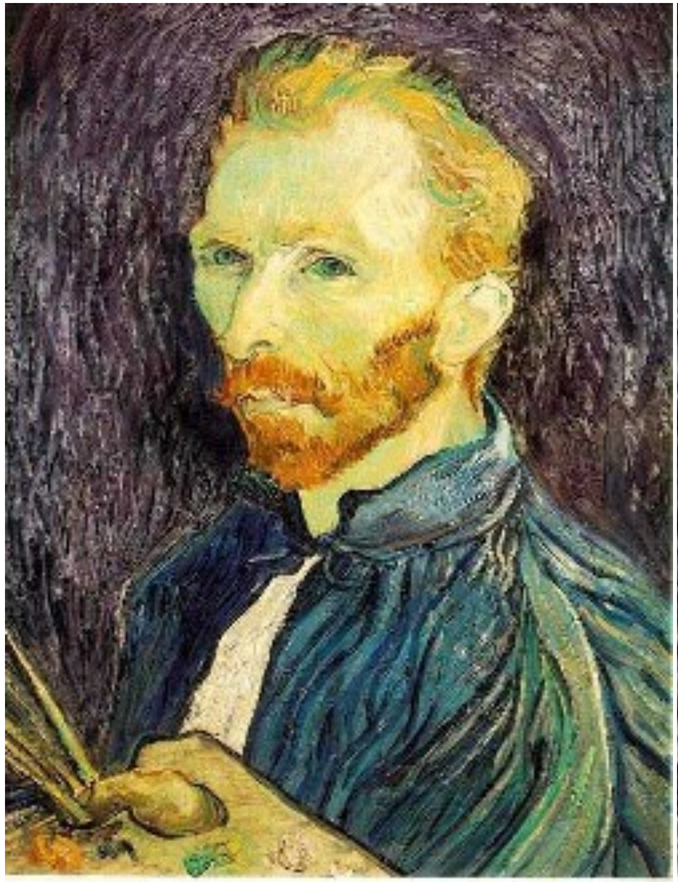


Aliasing



Aliasing artifacts

Image sub-sampling



1/2

1/4 (2x zoom)

1/16 (4x zoom)

Why does this look so crufty? Aliasing!

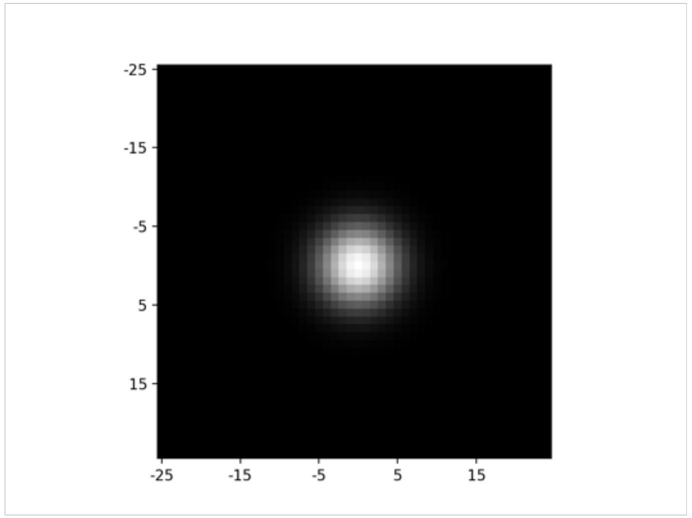
Source: S. Seitz

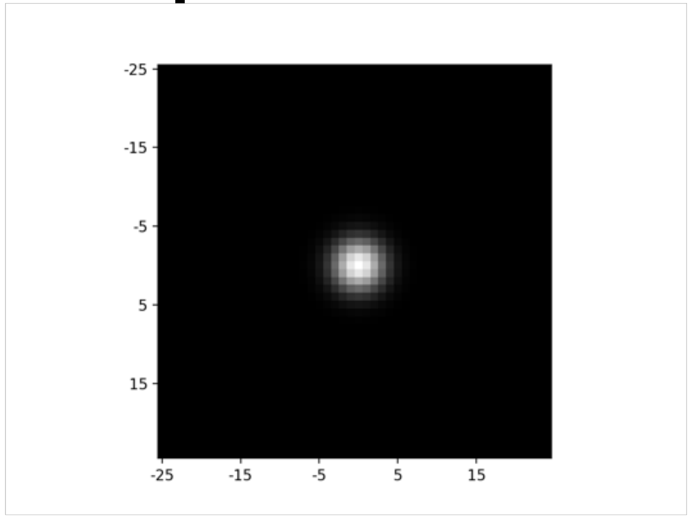
How to avoid aliasing

- To recover a sinusoid, need to sample at least twice per cycle
- For a general image, need to sample at least twice the rate of the highest frequency component
- **Nyquist sampling theorem:** $2\nu_{max} < \nu_{sample}$
- To subsample, *remove high frequency components*
- To remove high frequency components, *blur the image with a Gaussian*

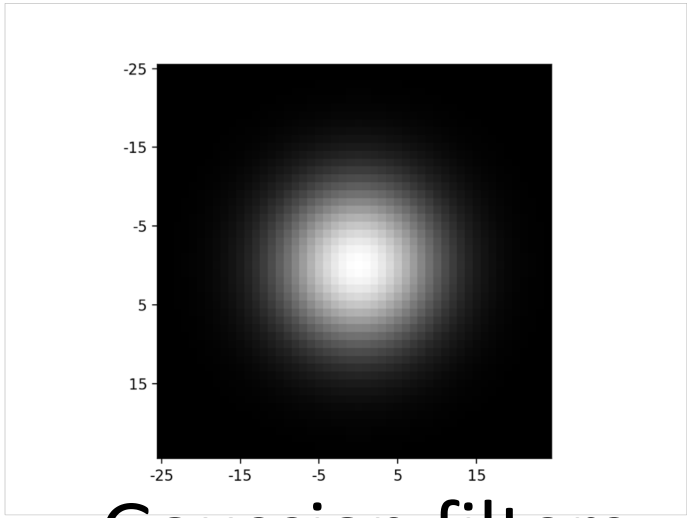
Image

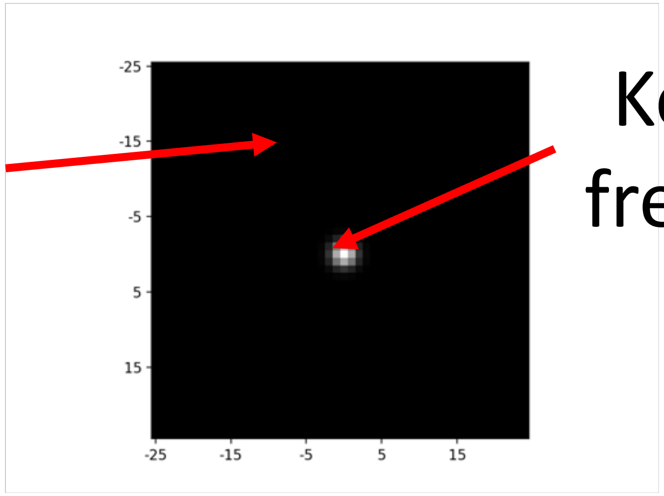
Fourier spectrum





Zeros out high frequencies





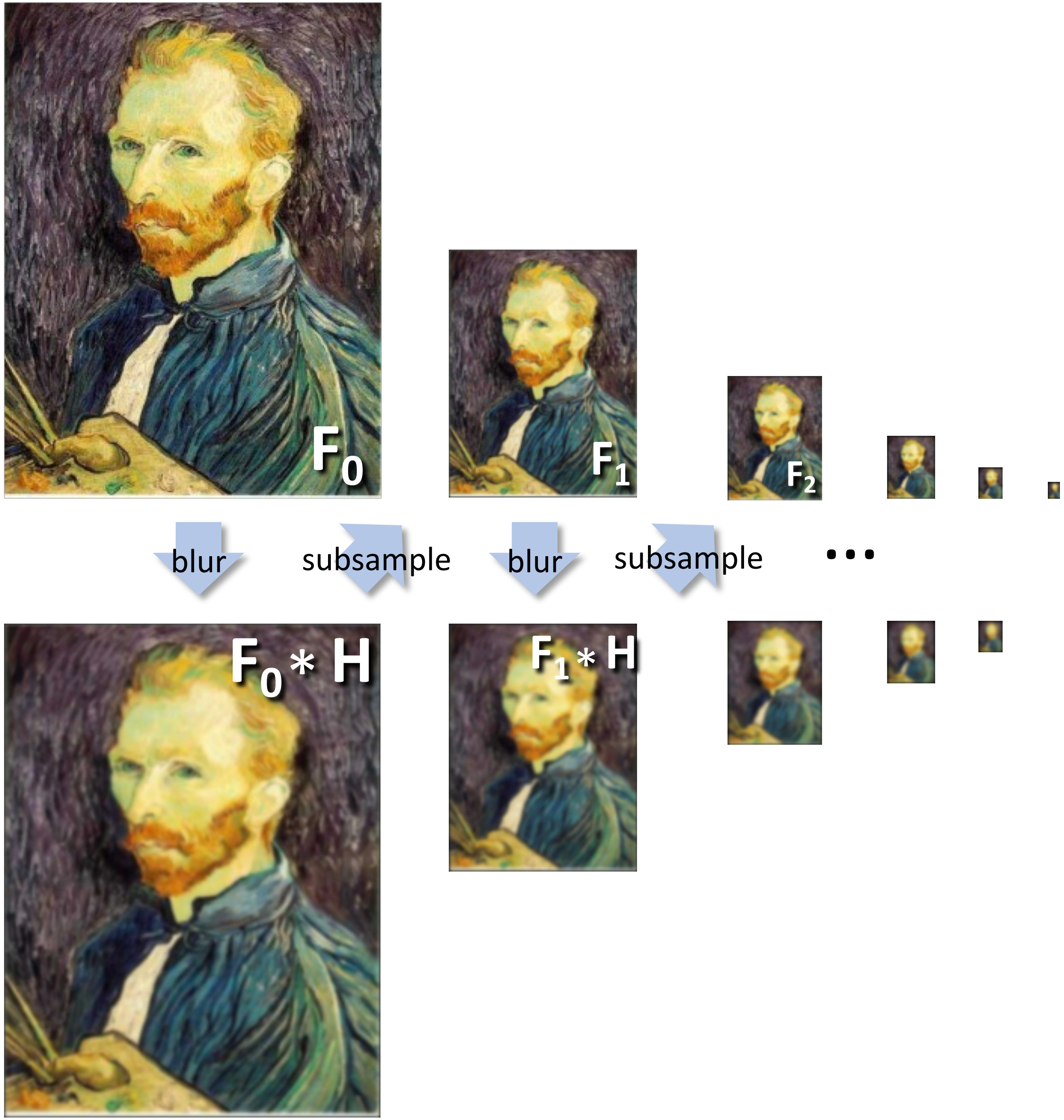
Keeps low frequencies

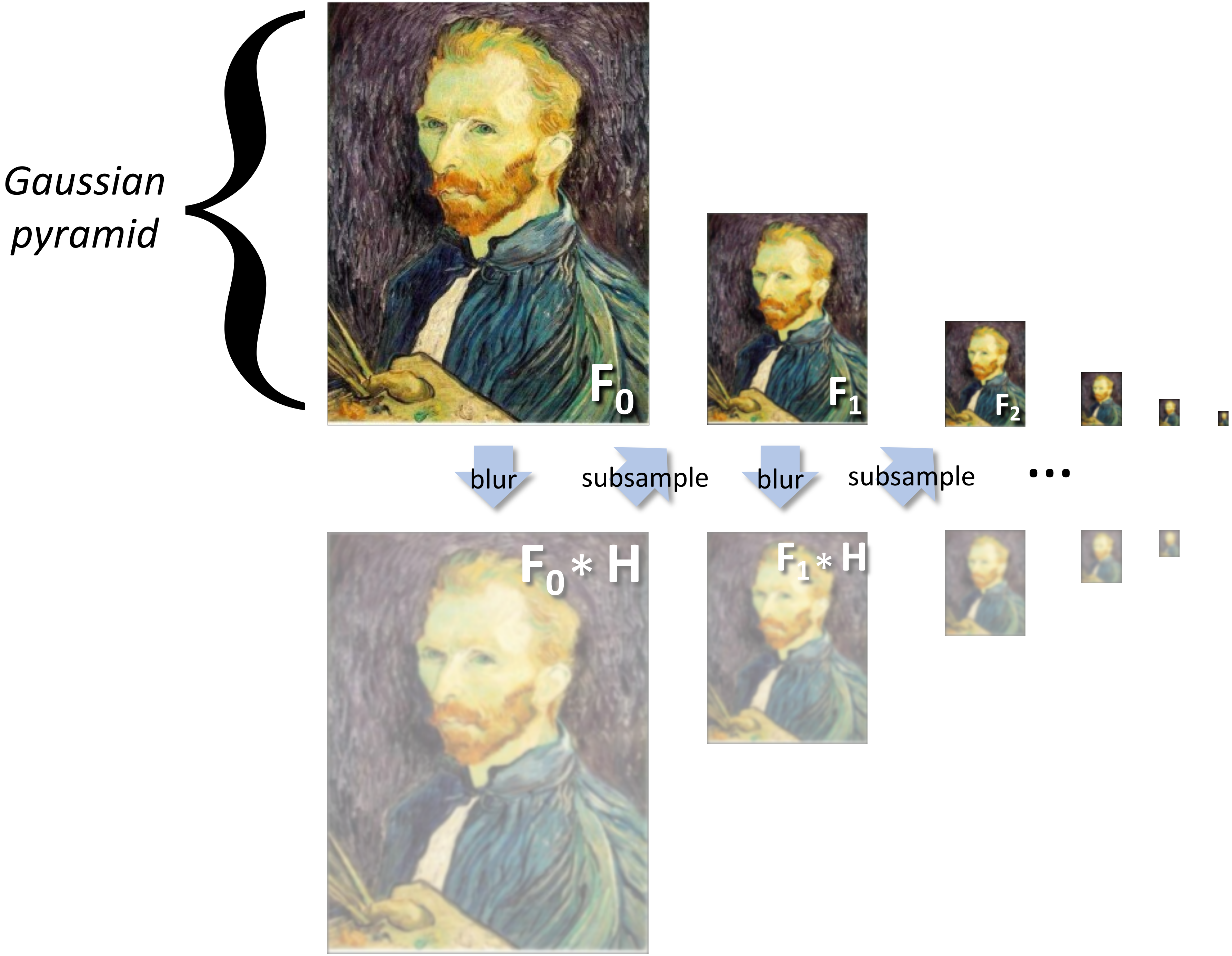
Gaussian filters

Fourier transform

Gaussian pre-filtering

- Solution: filter the image, *then* subsample

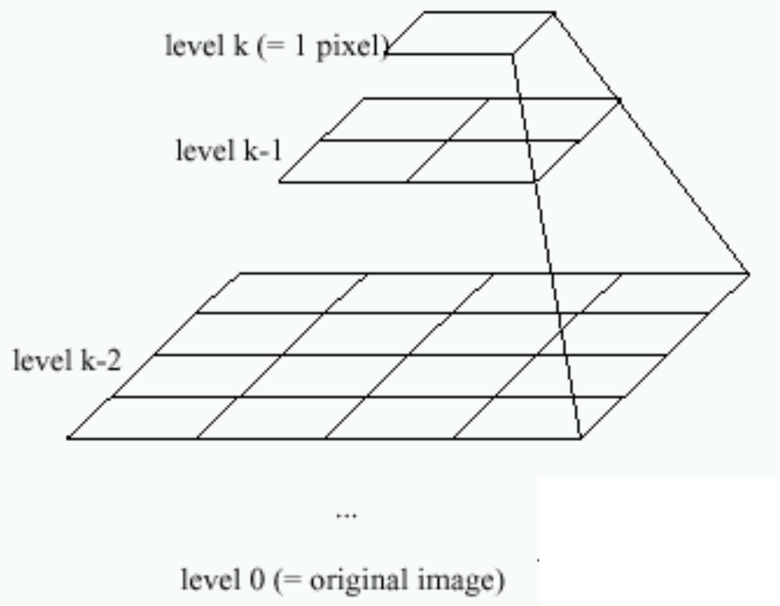




Gaussian pyramid
F0
F1
F2
blur
subsample
blur
subsample
...
F0 * H
F1 * H

Gaussian pyramids
[Burt and Adelson, 1983]

Idea: Represent NxN image as a "pyramid" of
$1\times1, 2\times2, 4\times4, \ldots, 2^k \times 2^k$ images (assuming $N=2^k$)

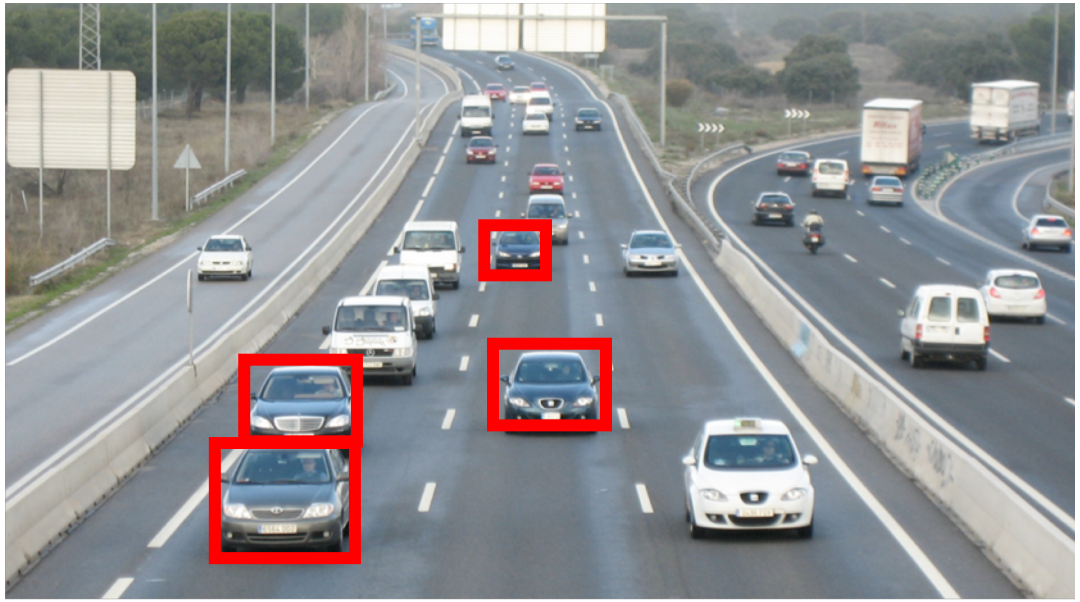


- In computer graphics, a *mip map* [Williams, 1983]

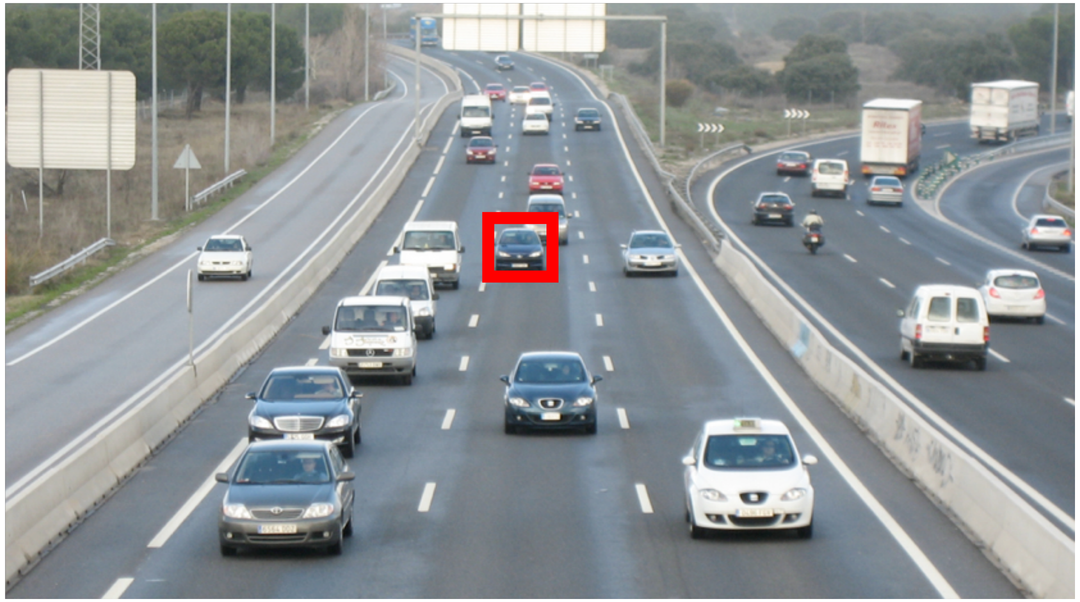
Gaussian Pyramids have all sorts of applications in computer vision

Source: S. Seitz

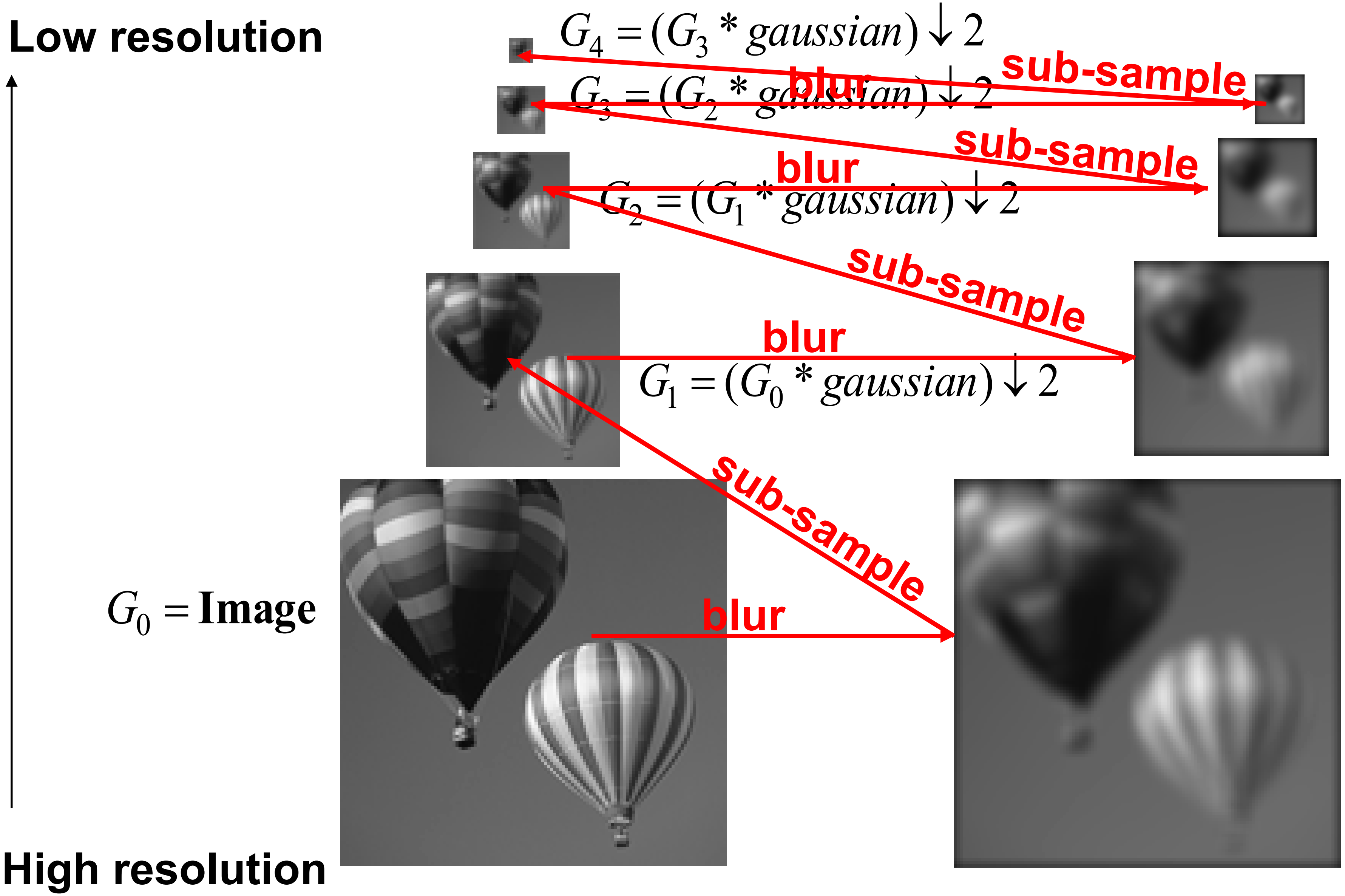
Gaussian pyramids - Searching over scales



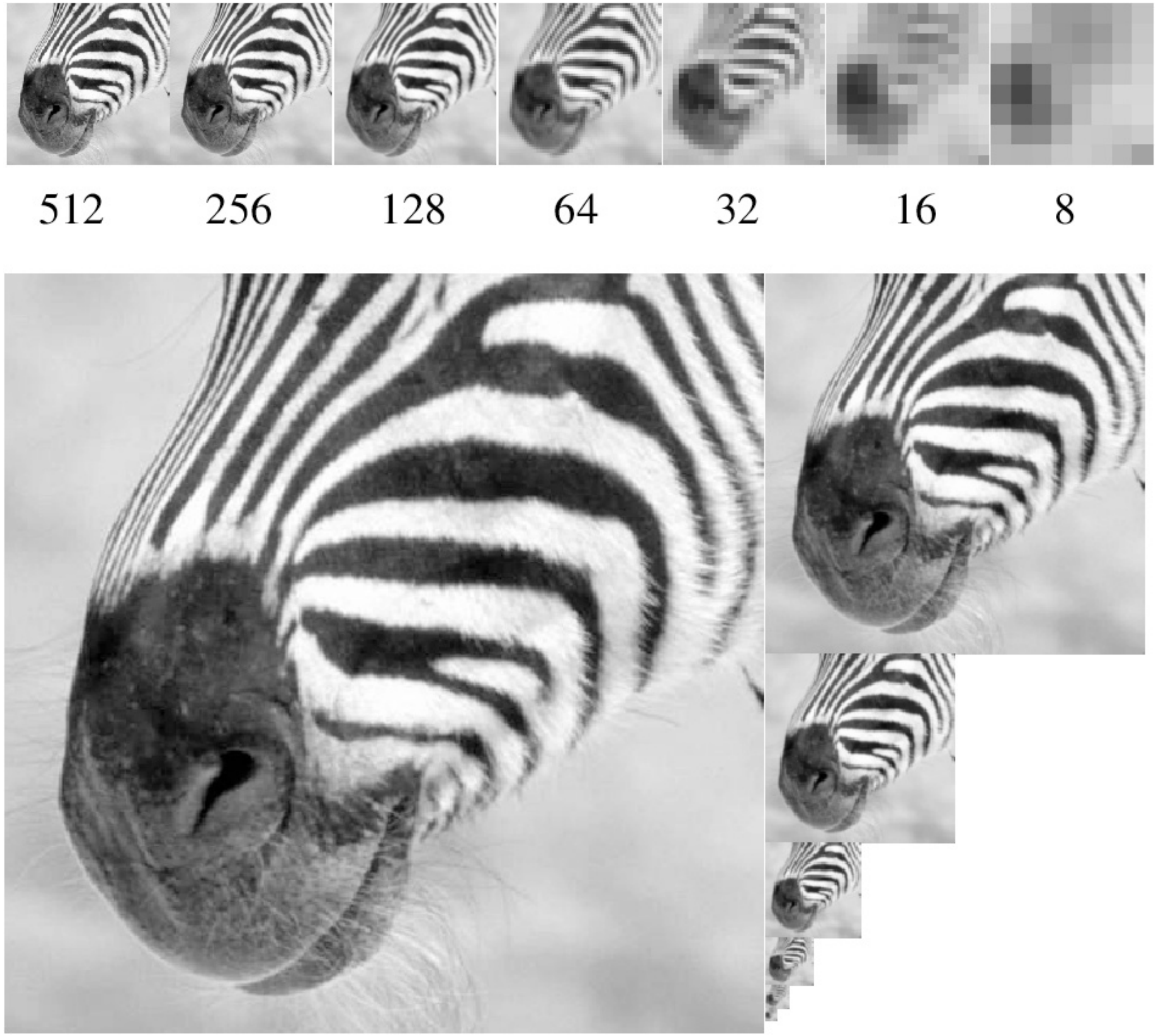
Gaussian pyramids - Searching over scales



The Gaussian Pyramid



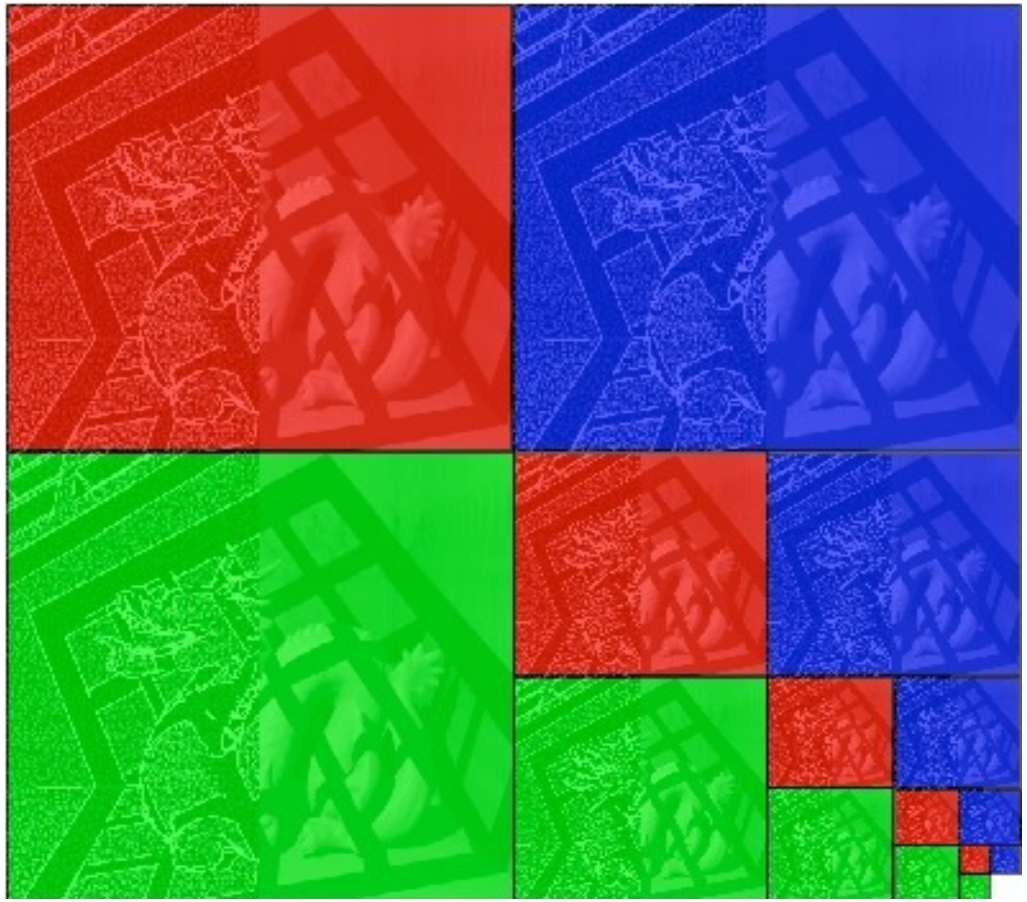
Gaussian pyramid and stack

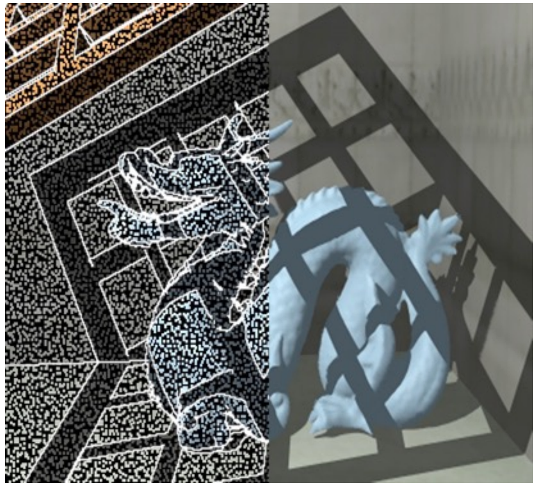


Source: Forsyth

Memory Usage

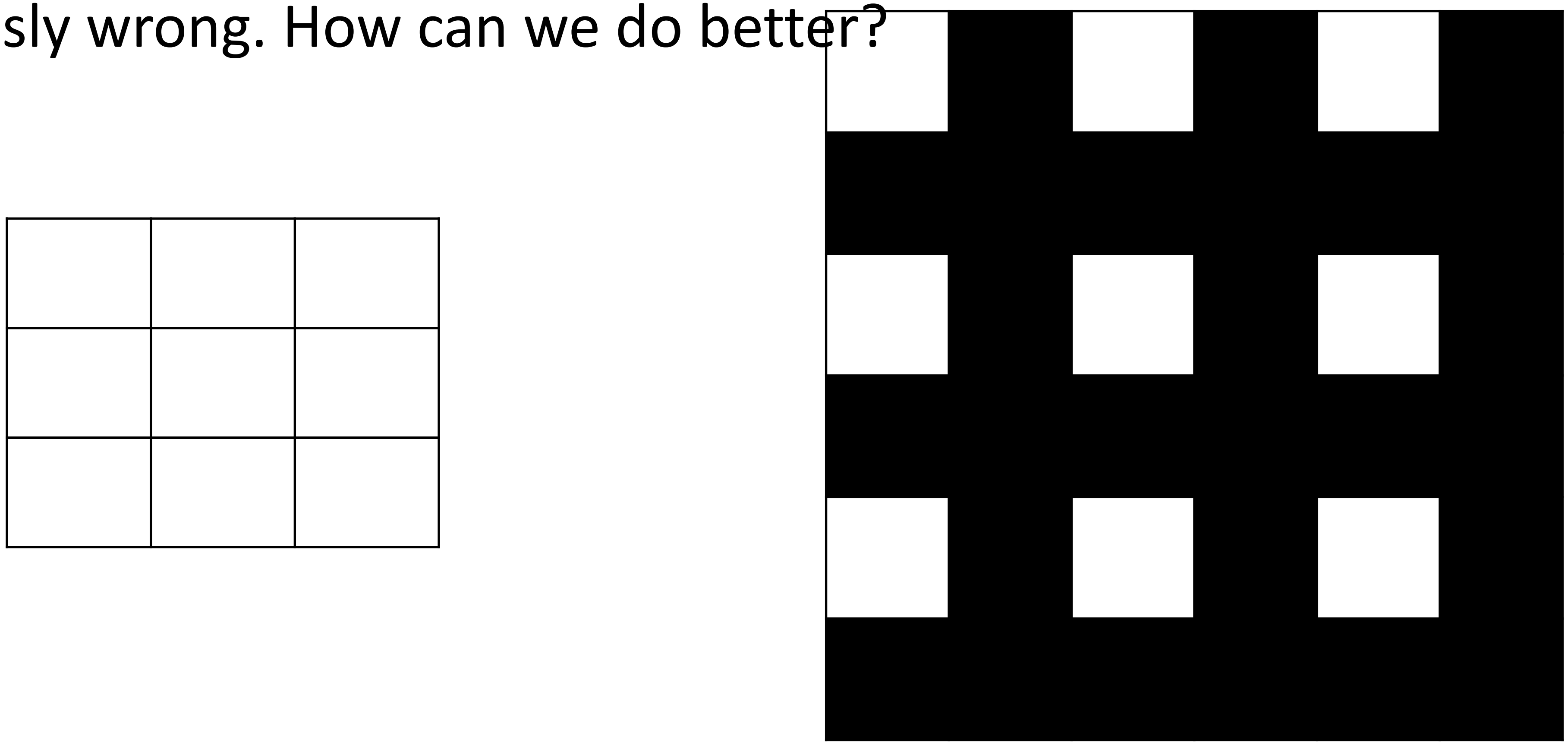
- Each color is a separate pyramid
- 3 pyramids fit into 2W x 2H image





What about upsampling?

- Simple solution: Fill rest of the pixels with zeros
- Obviously wrong. How can we do better?



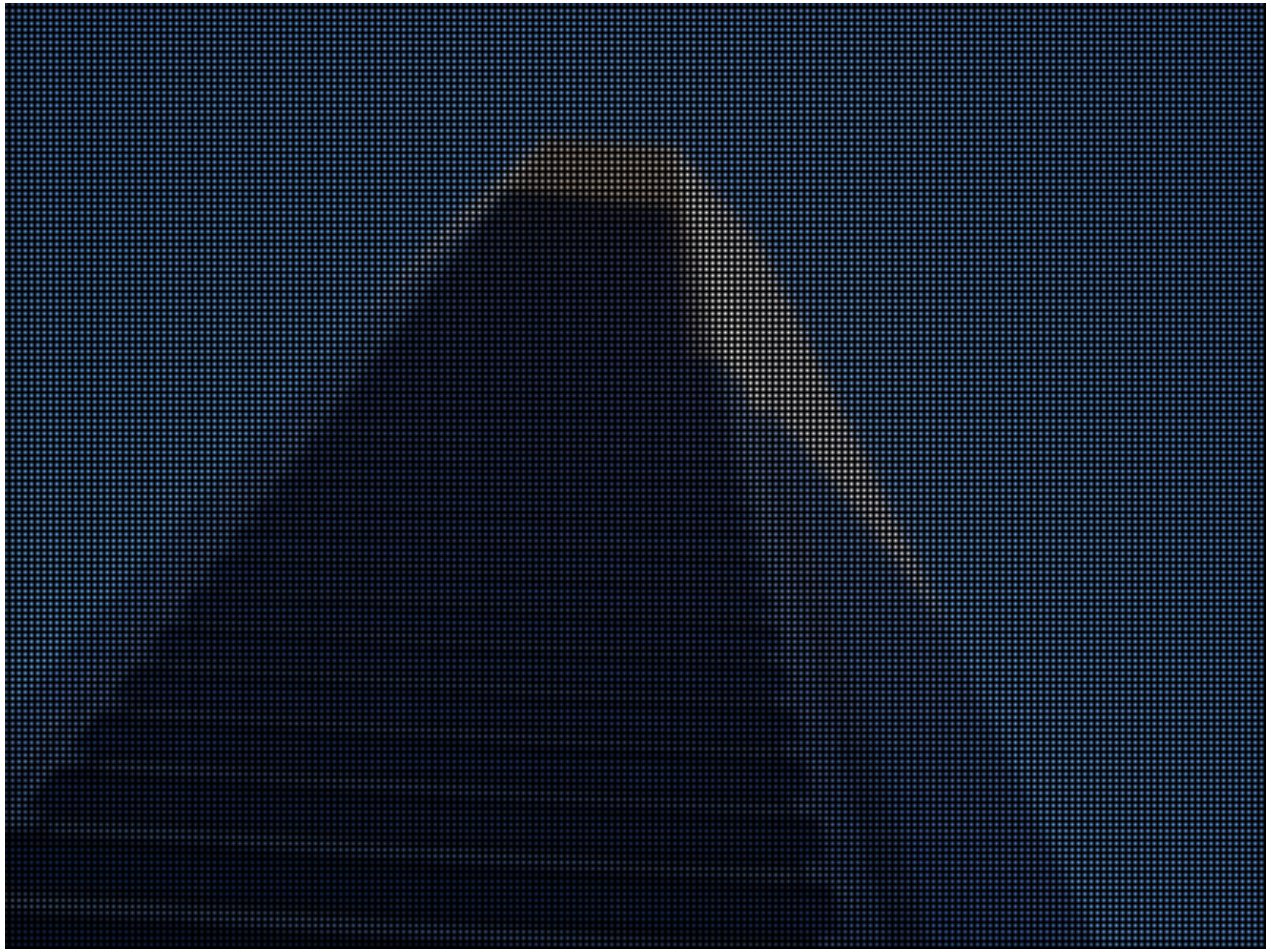
Upsampling

- Need to *interpolate* intermediate pixels. What is the best way to interpolate?
  - Find the *most likely* high-res image
- Recall: before subsampling, we removed high frequencies
- Key idea: upsampled image should not have high frequencies either
- Gaussian blur again!

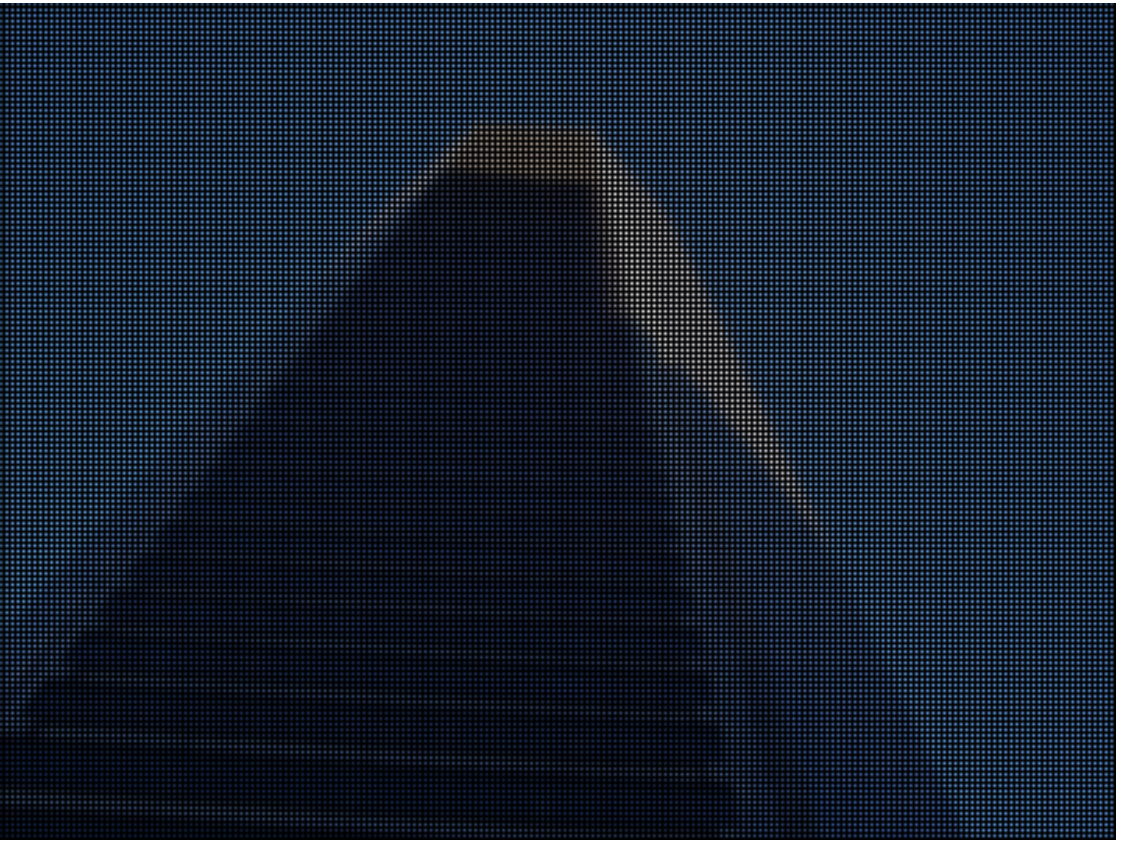
Upsampling

- Step 1: upsample and fill with 0s
- Step 2: Gaussian blur to interpolate
- Step 3: Scale correction
  - Gaussian blur is just weighted average
  - But we just introduced a bunch of zeros ==> need to scale up the resulting image

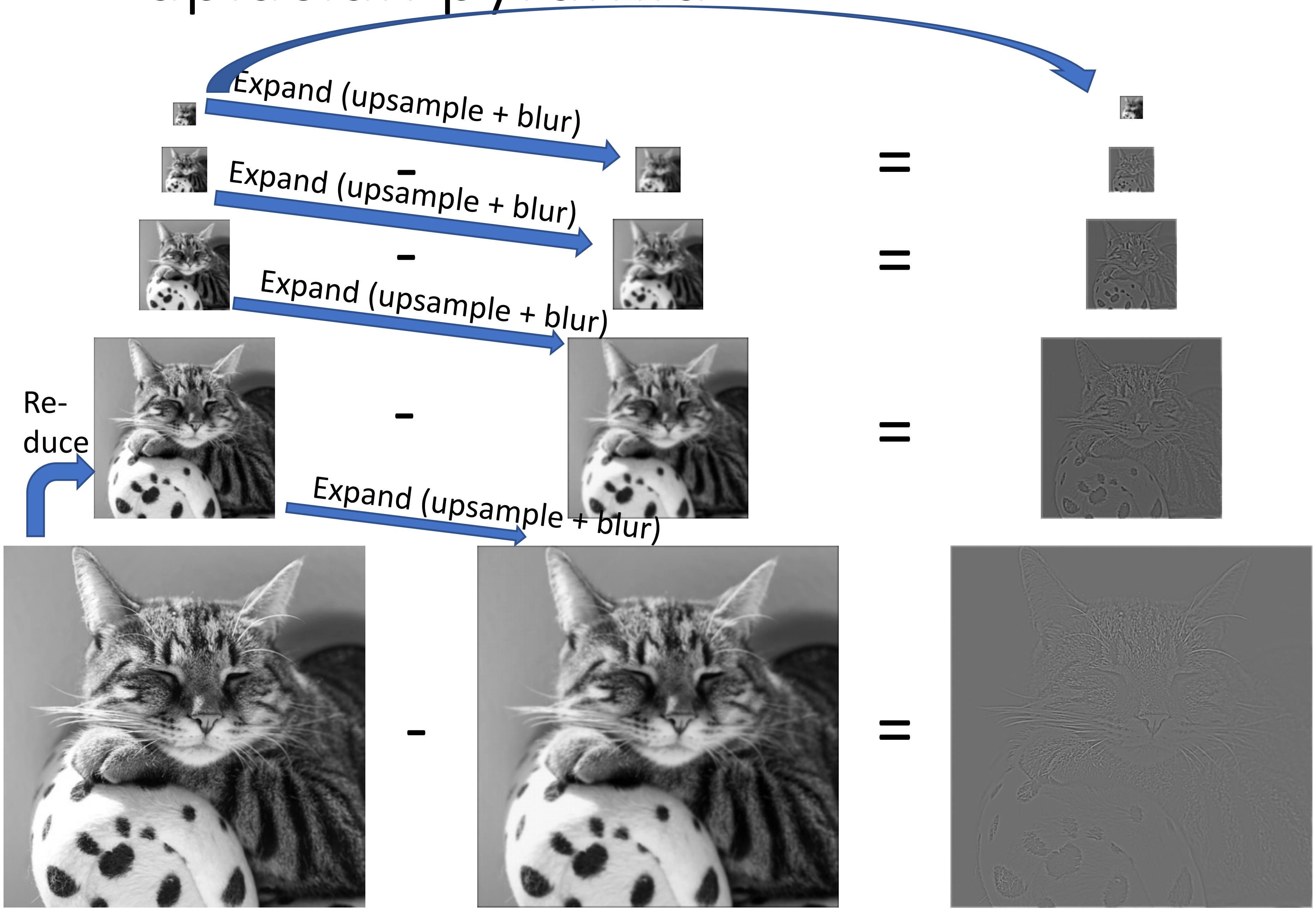
Upsampling: Step 1



Upsampling: Step 2 + 3



Laplacian pyramid



Laplacian pyramid

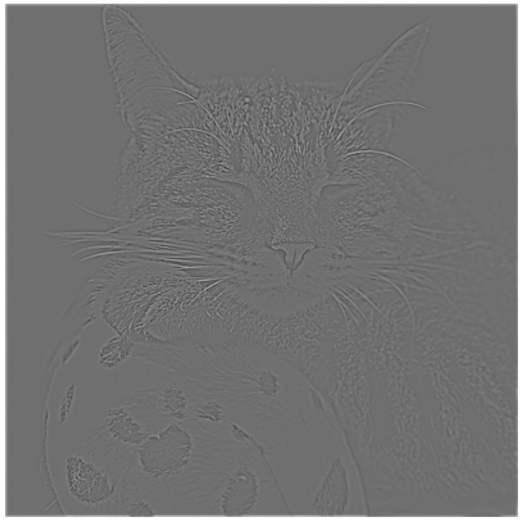
$L_4 = G_4 =$

$L_3 = G_3 - \text{expand}(G_4) =$

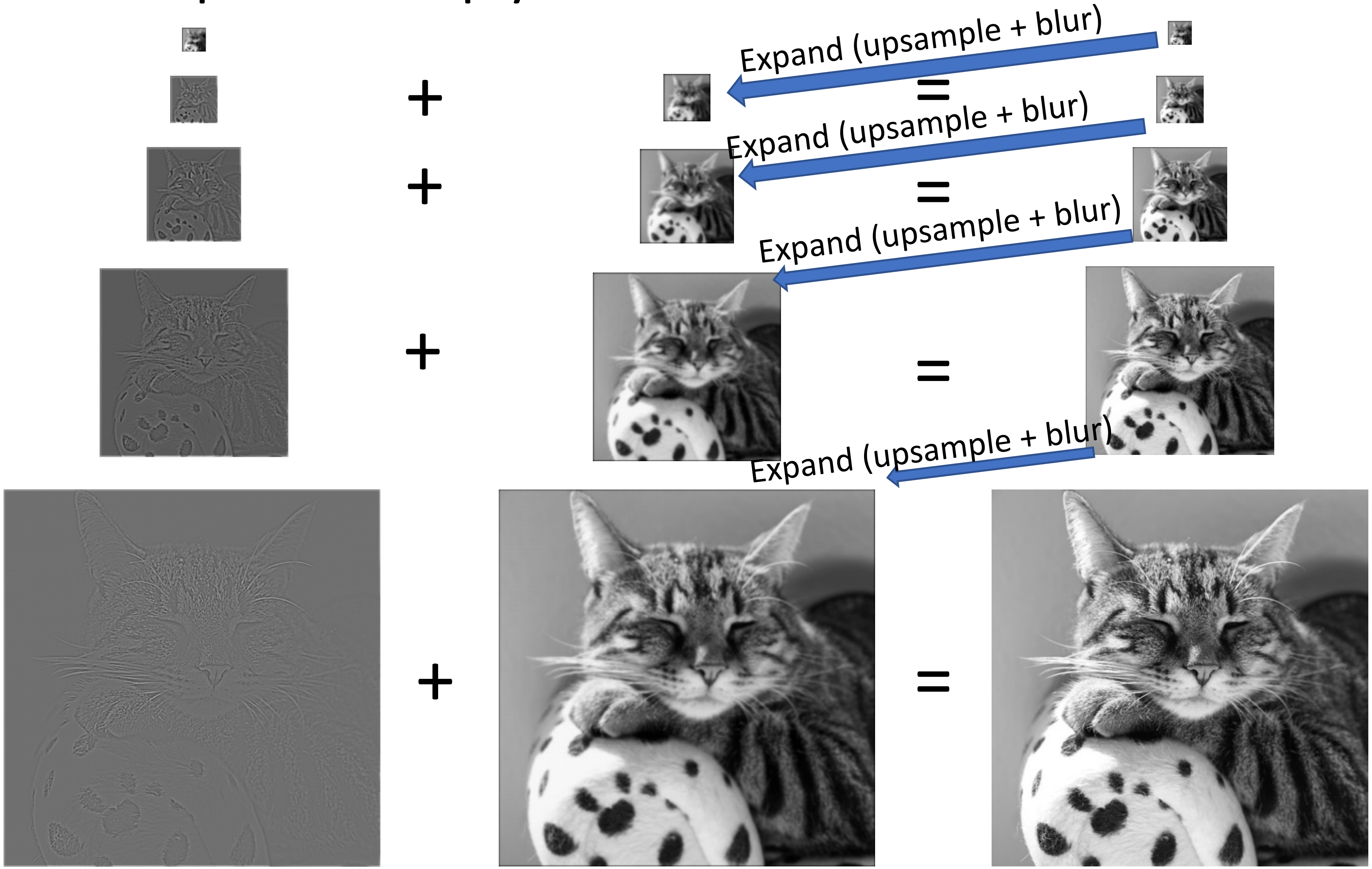
$L_2 = G_2 - \text{expand}(G_3) =$

$L_1 = G_1 - \text{expand}(G_2) =$

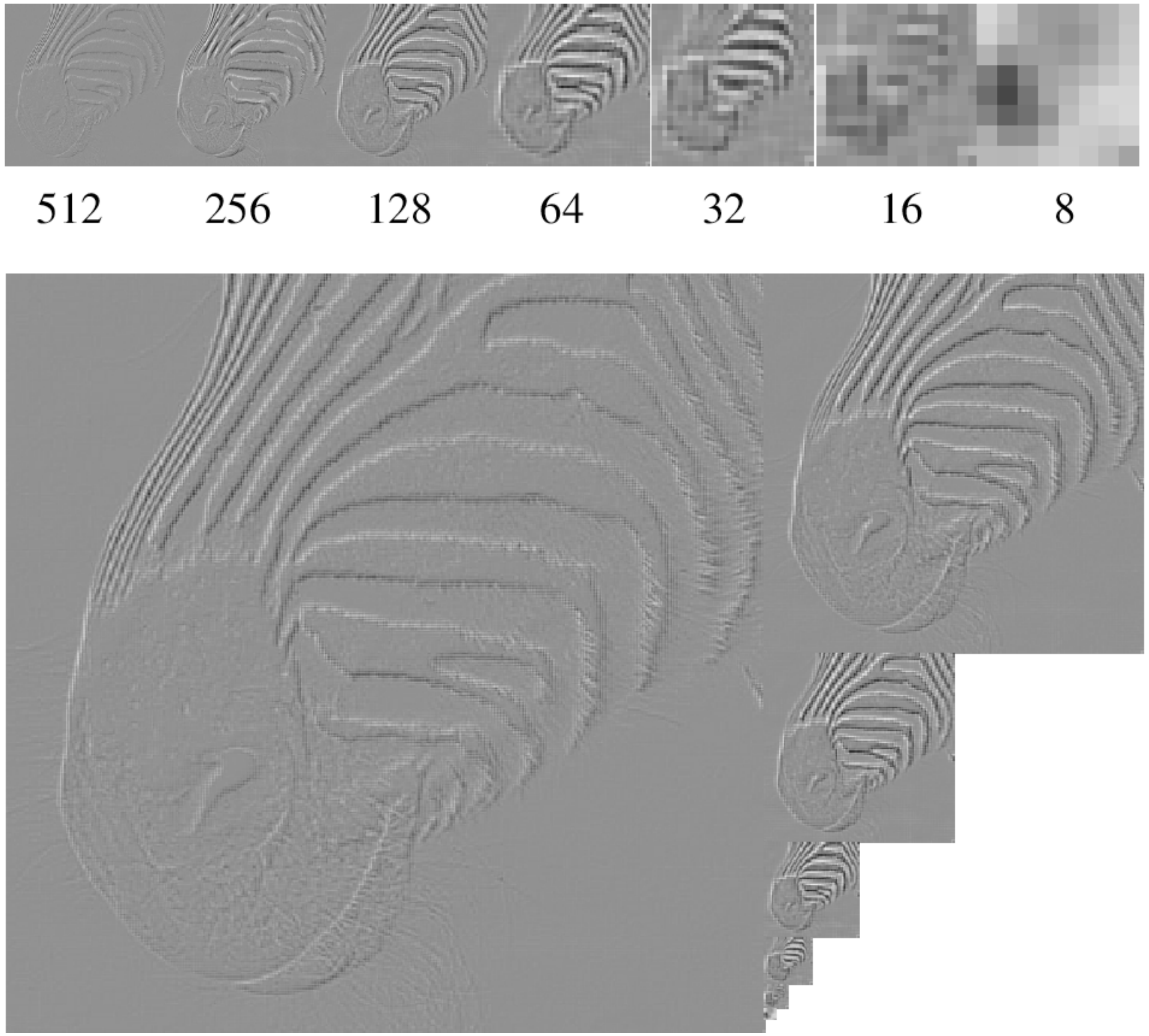
$L_0 = G_0 - \text{expand}(G_1) =$



Reconstructing the image from a Laplacian pyramid



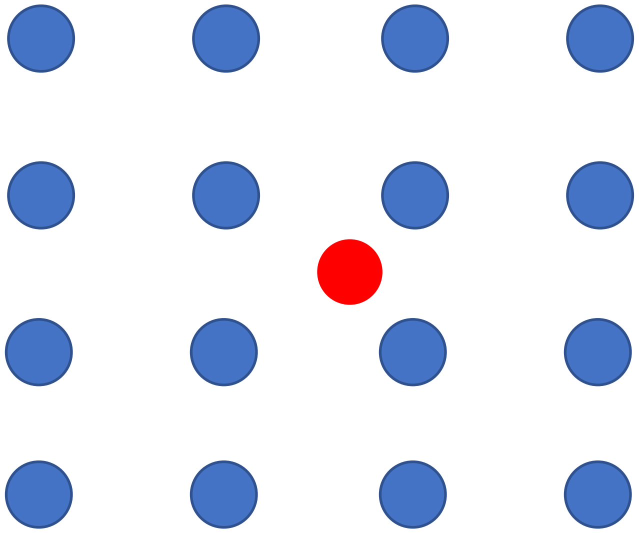
Laplacian pyramid



Source: Forsyth

Interpolation in general

- A more general question
- Given some known pixels in the image (shown in blue) how can we get the value of other pixels (shown in red)
- In our case, known pixels are in every other row/column



Interpolation in general

- Gaussian interpolation: set new pixels to be weighted combination of known pixels

$$g(x,y) = C \sum_{x'} \sum_{y'} e^{-\frac{(x-x')^2+(y-y')^2}{2\sigma^2}} f(x',y')$$

- Other forms of interpolation: other weights

$$g(x,y) = \sum_{x'} \sum_{y'} w(x,x',y,y') f(x',y')$$

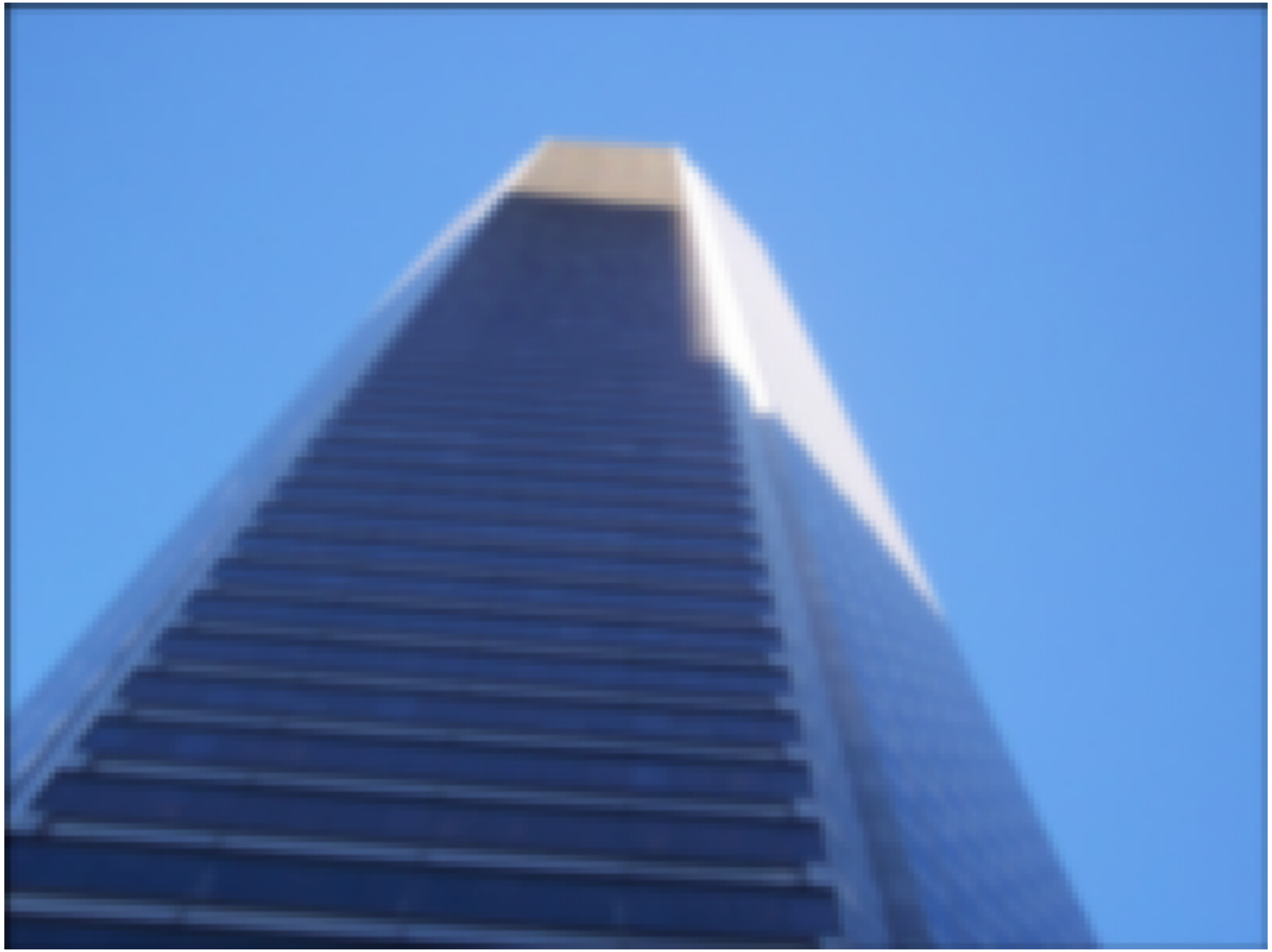
Interpolation in general

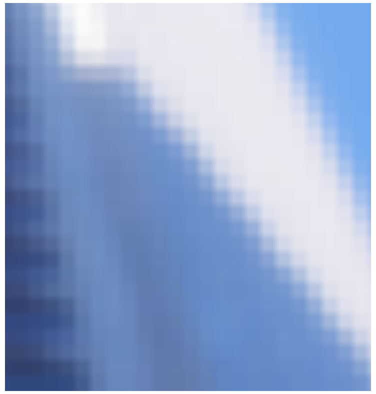
$$g(x, y) = \sum_{x'} \sum_{y'} w(x, x', y, y') f(x', y')$$

- Nearest neighbor interpolation
  - Find the nearest known pixel
  - Copy its value

$$g(x, y) = f(x^*, y^*)$$

Nearest-neighbnor interpolation



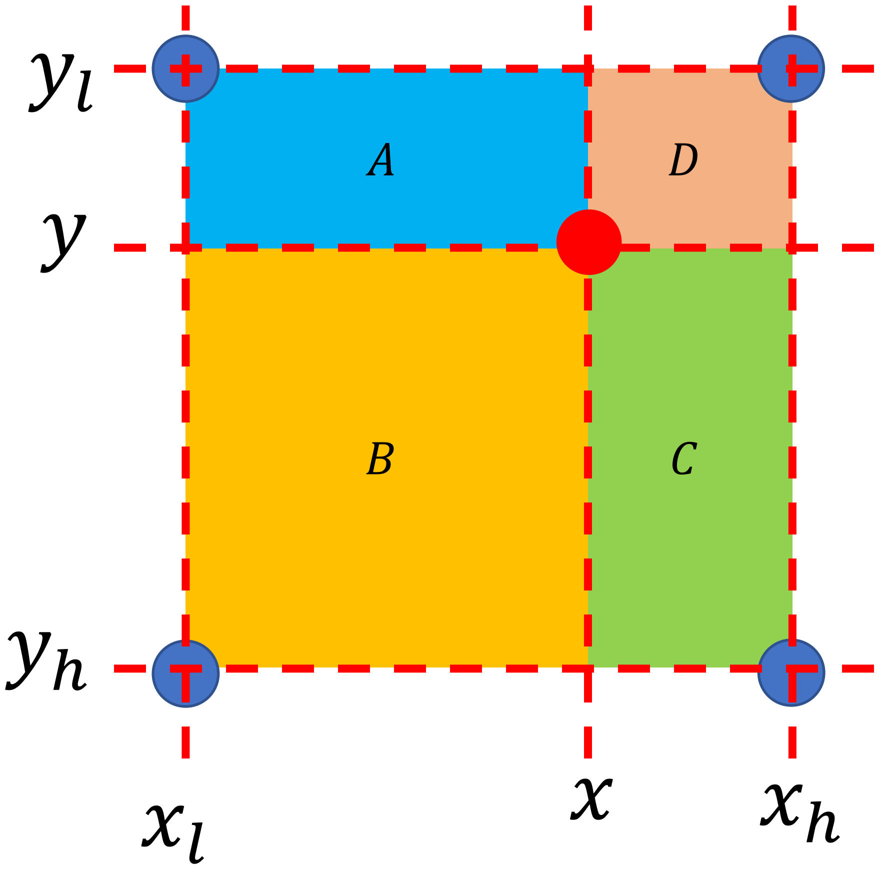


Bilinear interpolation

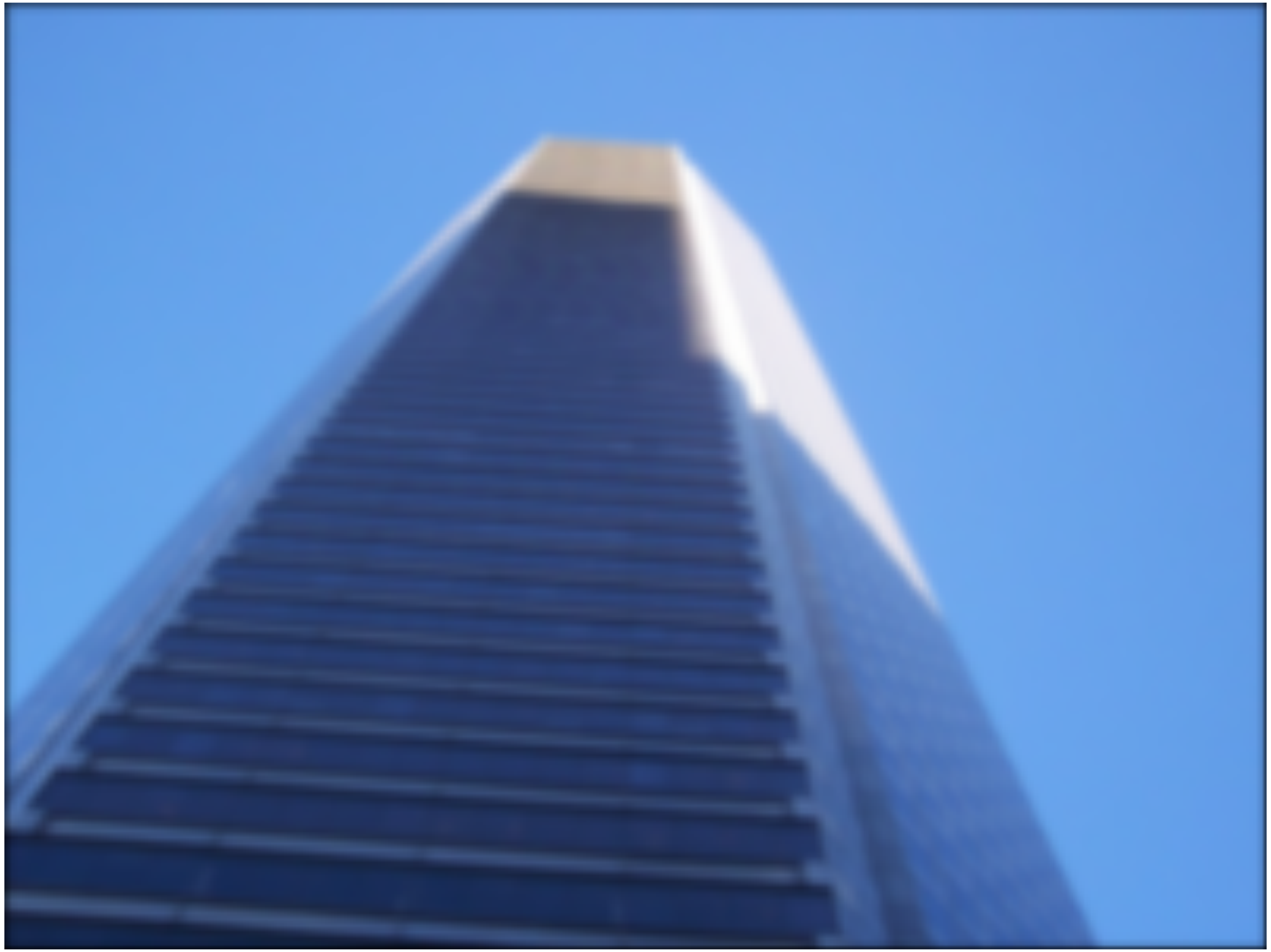
$$g(x,y) = \sum_{x'} \sum_{y'} w(x,x',y,y') f(x',y')$$

- Find the four nearest neighbors
  - $(x_l, y_l), (x_l, y_h), (x_h, y_h), (x_h, y_l)$
- Compute weighted average of the four

$$\begin{aligned} g(x,y) = & C f(x_l, y_l) \\ & + B f(x_h, y_l) \\ & + A f(x_h, y_h) \\ & + D f(x_l, y_h) \end{aligned}$$



Bilinear interpolation



Geometric transformations

- Geometric transformations involve changes to pixel *coordinates* instead of pixel *values*
- For example, resizing
  - Reducing size: $x, y \longmapsto \frac{x}{2}, \frac{y}{2}$
  - Increasing size: $x, y \longmapsto 2x, 2y$
- In general: $x, y \longmapsto T(x, y)$
- How can we do this?

Geometric transformations

- $x, y \longmapsto T(x, y)$
- Simplest solution: copy over pixel values to the new location
- $g\big(T(x, y)\big) = f(x, y)$
- Problem?
  - Only integer coordinates in f
  - So, not every pixel in g will be produced
  - Holes!

Geometric transformations

- $x, y \longmapsto T(x, y)$
- Better solution: find T^{-1}
- For every pixel of output g, set:
- $g(x, y) = f\left(T^{-1}(x, y)\right)$
- Problem: $T^{-1}(x, y)$ may not be integers
- Solution: interpolate!