CAP5415 Computer Vision
Spring 2003

Khurram Hassan-Shafique

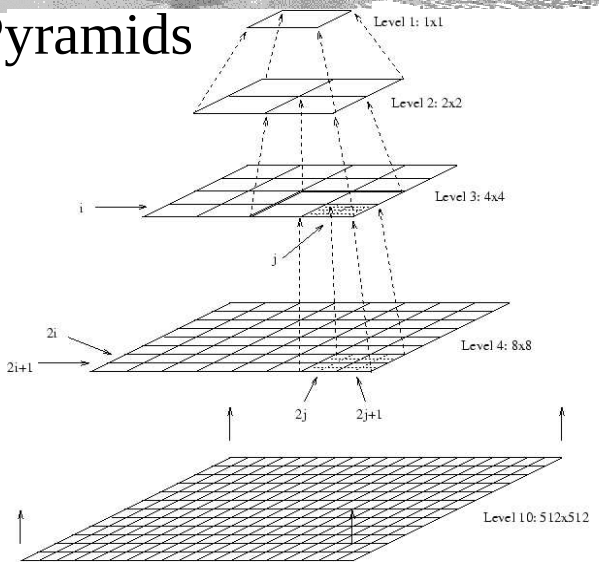
Scaled representations

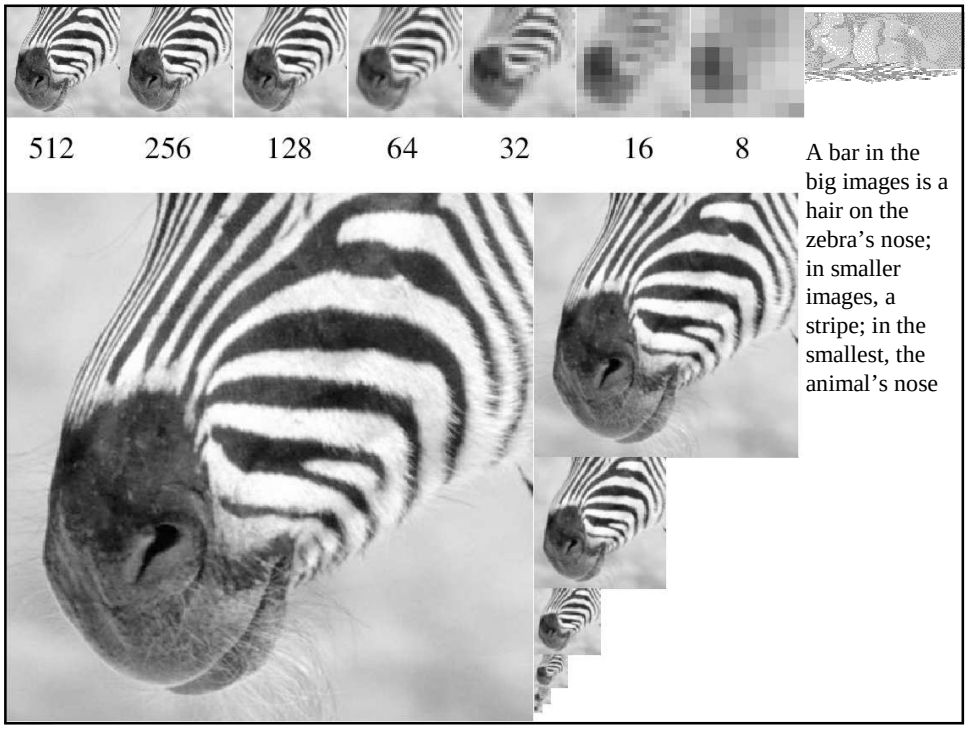
- Big bars (resp. spots, hands, etc.) and little bars are both interesting
  - Stripes and hairs, say
- Inefficient to detect big bars with big filters
  - And there is superfluous detail in the filter kernel
- Alternative:
  - Apply filters of fixed size to images of different sizes
  - Typically, a collection of images whose edge length changes by a factor of 2 (or root 2)
  - This is a pyramid (or Gaussian pyramid) by visual analogy

Gaussian Pyramids

- Very useful for representing images
- Image Pyramid is built by using multiple copies of image at different scales.
- Each level in the pyramid is ¼ of the size of previous level
- The highest level is of the highest resolution
- The lowest level is of the lowest resolution

Gaussian Pyramids

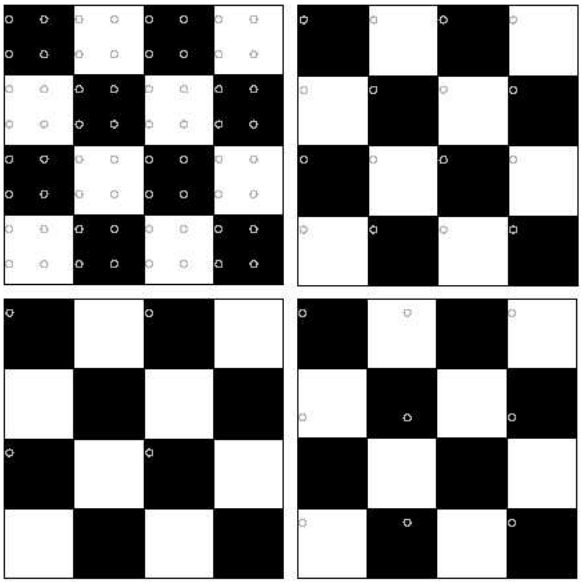




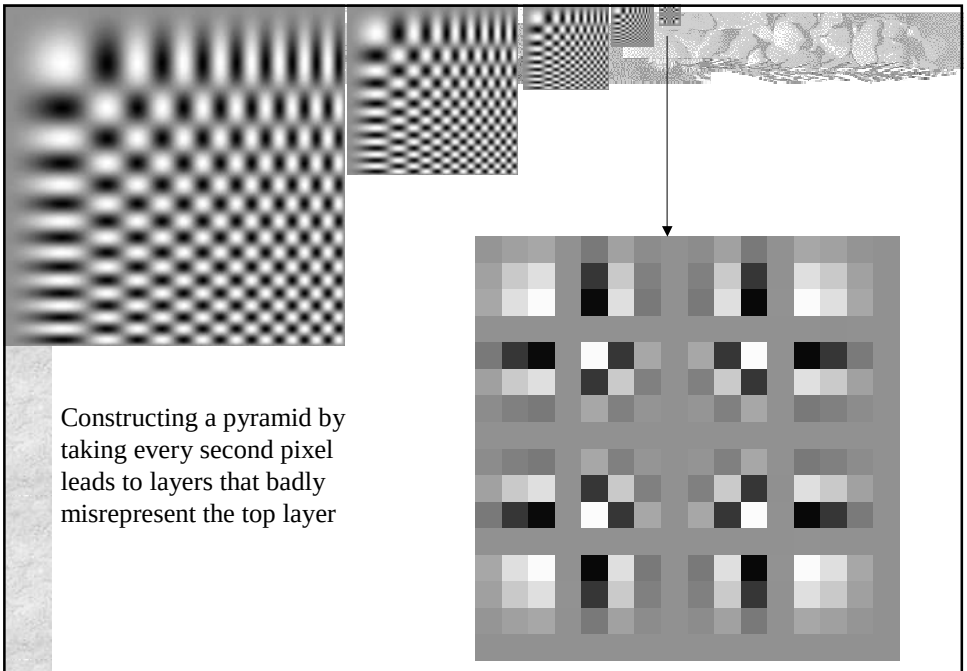
A bar in the big images is a hair on the zebra's nose; in smaller images, a stripe; in the smallest, the animal's nose

Aliasing

- Can't shrink an image by taking every second pixel
- If we do, characteristic errors appear
  - In the next few slides
  - Typically, small phenomena look bigger; fast phenomena can look slower
  - Common phenomenon
    - Wagon wheels rolling the wrong way in movies
    - Checkerboards misrepresented in ray tracing
    - Striped shirts look funny on colour television



Resample the checkerboard by taking one sample at each circle. In the case of the top left board, new representation is reasonable.
Top right also yields a reasonable representation.
Bottom left is all black (dubious) and bottom right has checks that are too big.



Constructing a pyramid by taking every second pixel leads to layers that badly misrepresent the top layer

Linear image transformations

- In analyzing images, it's often useful to make a change of basis.

transformed image

$$\vec{F} = U\vec{f}$$ ← Vectorized image

Fourier transform, or
Wavelet transform, or
Steerable pyramid transform

Self-inverting transforms

Same basis functions are used for the inverse transform

$$\vec{f} = U^{-1}\vec{F}$$

$$= U^{+}\vec{F}$$

U transpose and complex conjugate

Fourier Transform

Continuous : $F(g(x,y))(u,v) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} g(x,y)e^{-i2\pi(ux+vy)}dxdy$

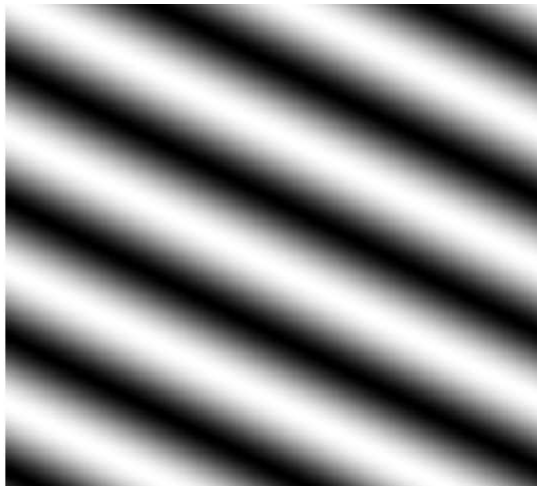
Discrete $F[m,n] = \sum_{k=0}^{M-1}\sum_{l=0}^{N-1} f[k,l]e^{-\pi i\left(\frac{km}{M}+\frac{\ln}{N}\right)}$

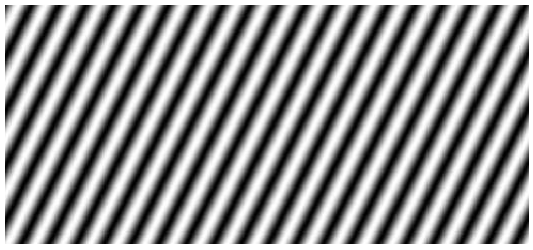
The Fourier Transform

- Represent function on a new basis
  - Think of functions as vectors, with many components
  - We now apply a linear transformation to transform the basis
    - dot product with each basis element
- In the expression, u and v select the basis element, so a function of x and y becomes a function of u and v
- basis elements have the form $e^{-i2\pi(ux+vy)}$

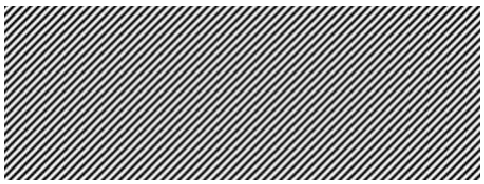
$$F(g(x,y))(u,v) = \iint_{R^2} g(x,y)e^{-i2\pi(ux+vy)}dxdy$$

To get some sense of what basis elements look like, we plot a basis element --- or rather, its real part --- as a function of x,y for some fixed u, v. We get a function that is constant when (ux+vy) is constant. The magnitude of the vector (u, v) gives a frequency, and its direction gives an orientation. The function is a sinusoid with this frequency along the direction, and constant perpendicular to the direction.





Here u & v are larger than the previous slide



Larger than the upper example

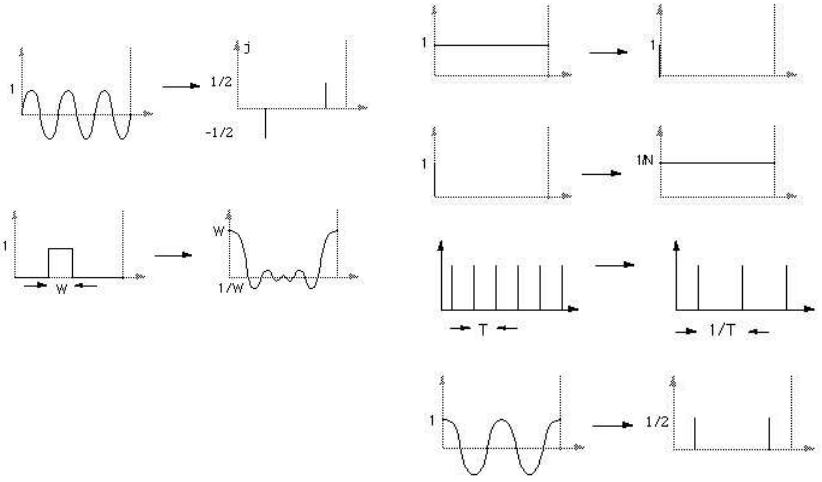
Phase and Magnitude

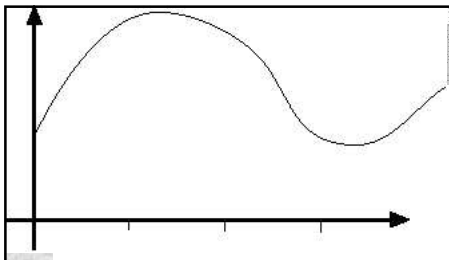
- Fourier transform of a real function is complex
  - difficult to plot, visualize
  - instead, we can think of the phase and magnitude of the transform
- Phase is the phase of the complex transform
- Magnitude is the magnitude of the complex transform
- Curious fact
  - all natural images have about the same magnitude transform
  - hence, phase seems to matter, but magnitude largely doesn't
- Demonstration
  - Take two pictures, swap the phase transforms, compute the inverse - what does the result look like?

Various Fourier Transform Pairs

- Important facts
  - The Fourier transform is linear
  - There is an inverse FT
  - if you scale the function's argument, then the transform's argument scales the other way. This makes sense --- if you multiply a function's argument by a number that is larger than one, you are stretching the function, so that high frequencies go to low frequencies
  - The FT of a Gaussian is a Gaussian.
- The convolution theorem
  - The Fourier transform of the convolution of two functions is the product of their Fourier transforms
  - The inverse Fourier transform of the product of two Fourier transforms is the convolution of the two inverse Fourier transforms
- There's a table in the book.

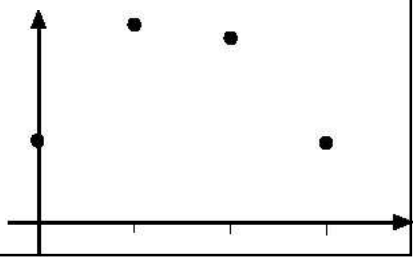
Various Fourier Transform Pairs

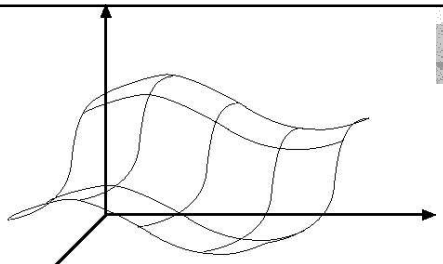




Sampling in 1D

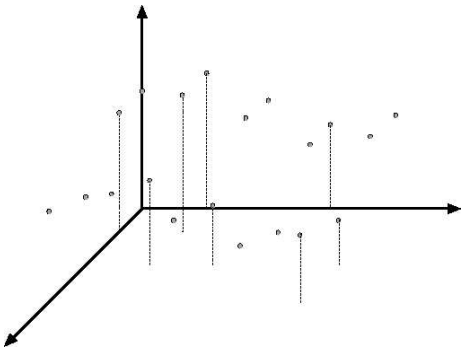
Sampling in 1D takes a continuous function and replaces it with a vector of values, consisting of the function's values at a set of sample points. We'll assume that these sample points are on a regular grid, and can place one at each integer for convenience.



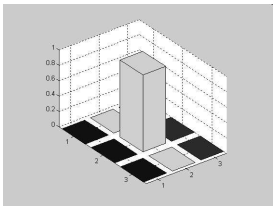


Sampling in 2D

Sampling in 2D does the same thing, only in 2D. We'll assume that these sample points are on a regular grid, and can place one at each integer point for convenience.



Convolution

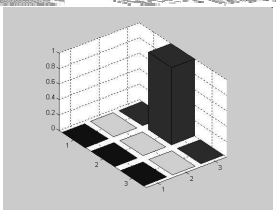


*

0	0	0
0	1	0
0	0	0

=

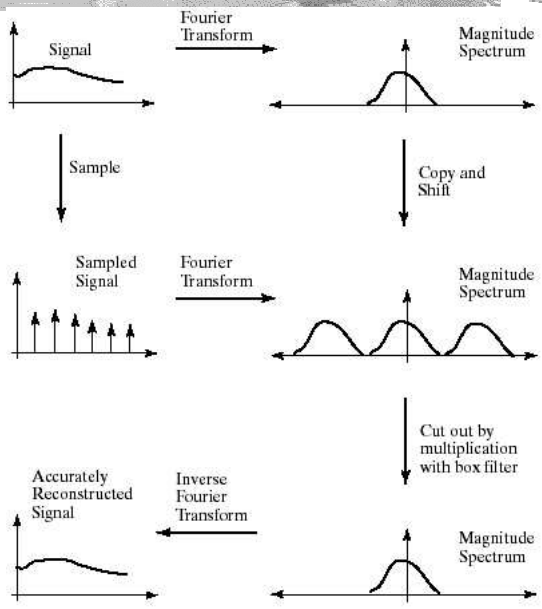
Convolution

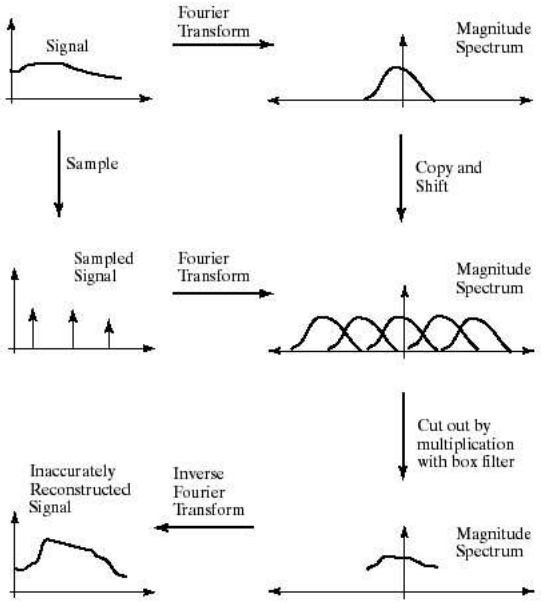


*

0	0	0
0	0	1
0	0	0

=





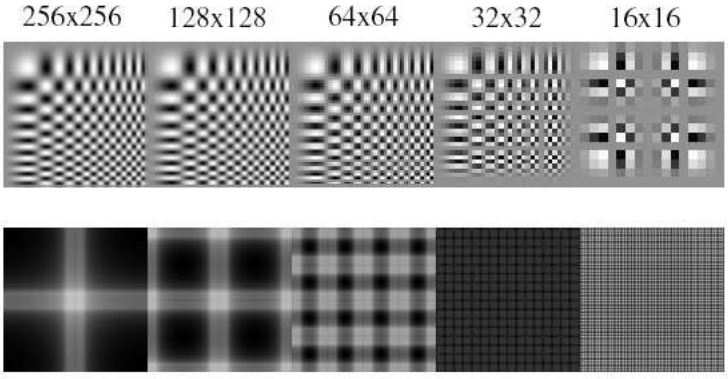
Nyquist Theorem

- In order for a band-limited (i.e., one with a zero power spectrum for frequencies $f > B$) baseband ($f > 0$) signal to be reconstructed fully, it must be sampled at a rate $f \geq 2B$. A signal sampled at $f = 2B$ is said to be Nyquist sampled, and $f = 2B$ is called the Nyquist frequency. No information is lost if a signal is sampled at the Nyquist frequency, and no additional information is gained by sampling faster than this rate.

Smoothing as low-pass filtering

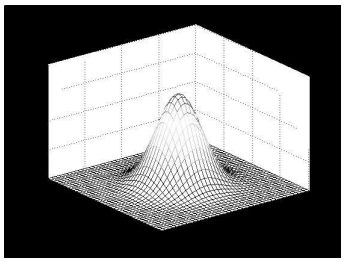
- The message of the NT is that high frequencies lead to trouble with sampling.
- Solution: suppress high frequencies before sampling
  - multiply the FT of the signal with something that suppresses high frequencies
  - or convolve with a low-pass filter
- A filter whose FT is a box is bad, because the filter kernel has infinite support
- Common solution: use a Gaussian
  - multiplying FT by Gaussian is equivalent to convolving image with Gaussian.

Sampling without smoothing. Top row shows the images, sampled at every second pixel to get the next; bottom row shows the magnitude spectrum of these images.



Gaussian Filter

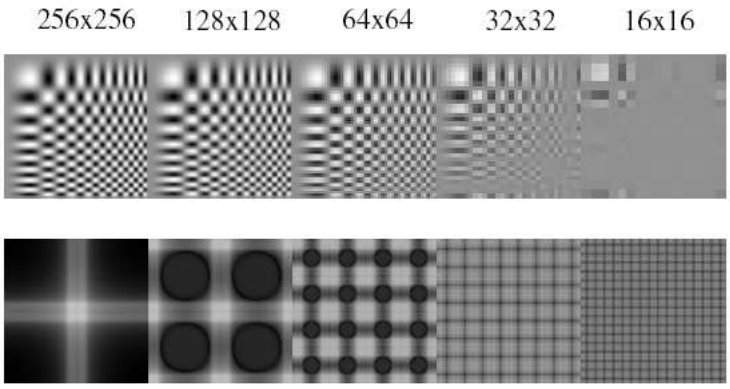
$$G_\sigma(x,y)=\frac{1}{2\pi\sigma^2}\exp\left(-\frac{\left(x^2+y^2\right)}{2\sigma^2}\right)$$



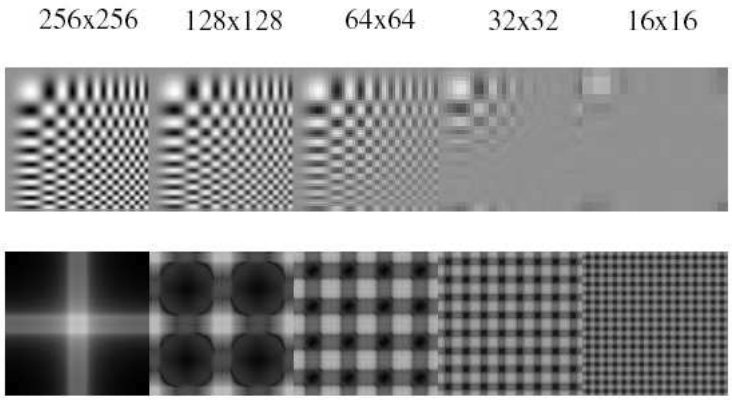
$$H(i,j)=\frac{1}{2\pi\sigma^2}\exp\left(-\frac{\left((i-k-1)^2+(j-k-1)^2\right)}{2\sigma^2}\right)$$

where $H(i,j)$ is $(2k+1)\times(2k+1)$ array

Sampling with smoothing. Top row shows the images. We get the next image by smoothing the image with a Gaussian with sigma 1 pixel, then sampling at every second pixel to get the next; bottom row shows the magnitude spectrum of these images.



Sampling with smoothing. Top row shows the images. We get the next image by smoothing the image with a Gaussian with sigma 1.4 pixels, then sampling at every second pixel to get the next; bottom row shows the magnitude spectrum of these images.



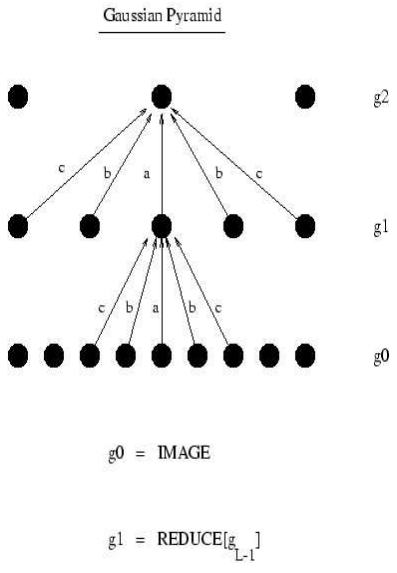
Applications of scaled representations

- Search for correspondence
  - look at coarse scales, then refine with finer scales
- Edge tracking
  - a “good” edge at a fine scale has parents at a coarser scale
- Control of detail and computational cost in matching
  - e.g. finding stripes
  - terribly important in texture representation

The Gaussian pyramid

- Smooth with gaussians, because
  - a gaussian*gaussian=another gaussian
- Synthesis
  - smooth and sample
- gaussians are low pass filters

Reduce (1D)



Reduce (1D)

$$g_l(i) = \sum_{m=-2}^{2} \hat{w}(m) g_{l+1}(2i+m)$$

$$g_l(2) = \hat{w}(-2) g_{l+1}(4-2) + \hat{w}(-1) g_{l+1} \hat{w}(4-1) + \hat{w}(0) g_{l+1}(4) + \hat{w}(1) g_{l+1}(4+1) + \hat{w}(2) g_{l+1}(4+2)$$

$$g_l(2) = \hat{w}(-2) g_{l+1}(2) + \hat{w}(-1) g_{l+1} \hat{w}(3) + \hat{w}(0) g_{l+1}(4) + \hat{w}(1) g_{l+1}(5) + \hat{w}(2) g_{l+1}(6)$$

Convolution Kernel

$$[w(-2), w(-1), w(0), w(1), w(2)]$$

- Symmetric

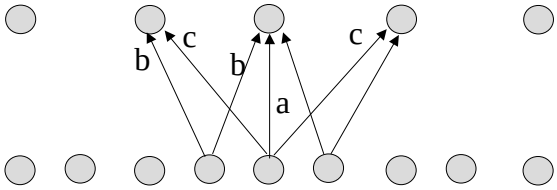
$$w(i) = w(-i) \Rightarrow [c, b, a, b, c]$$

- Sum of mask should be 1

$$a + 2b + 2c = 1$$

Convolution Kernel

- All nodes at a given level must contribute the same total weight to the nodes at the next higher level



$$a + 2c = 2b$$

Convolution Kernel

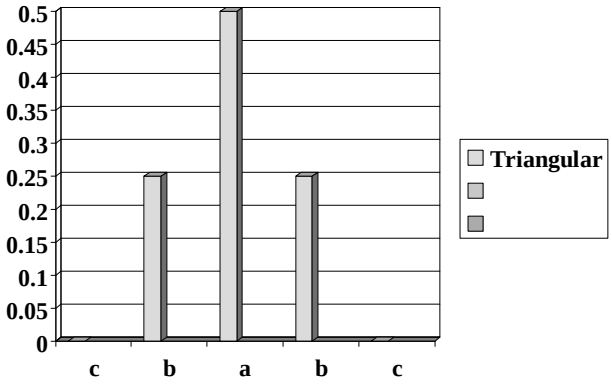
$$w(0) = a$$

$$w(-1) = w(1) = \frac{1}{4}$$

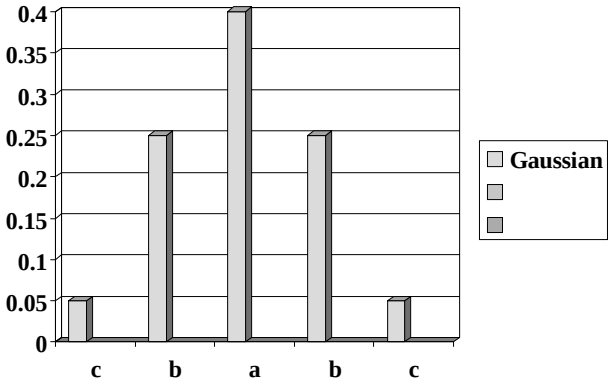
$$w(-2) = w(2) = \frac{1}{4} - \frac{a}{2}$$

a=0.4 GAUSSIAN, a=0.5 TRIANGULAR

Triangular



Approximate Gaussian



What about 2D?

- Separability of Gaussian

$$\hat{I}(x,y) = I(x,y) * G(x,y)$$

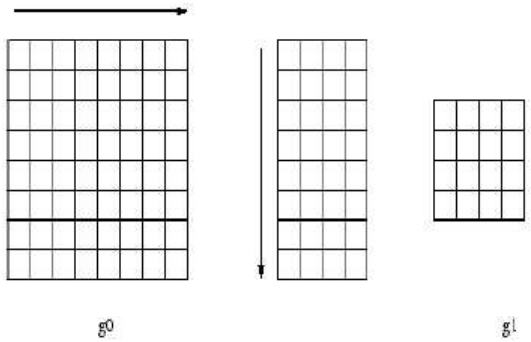
Requires $n^2 k^2$ multiplications for n by n image and k by k kernel.

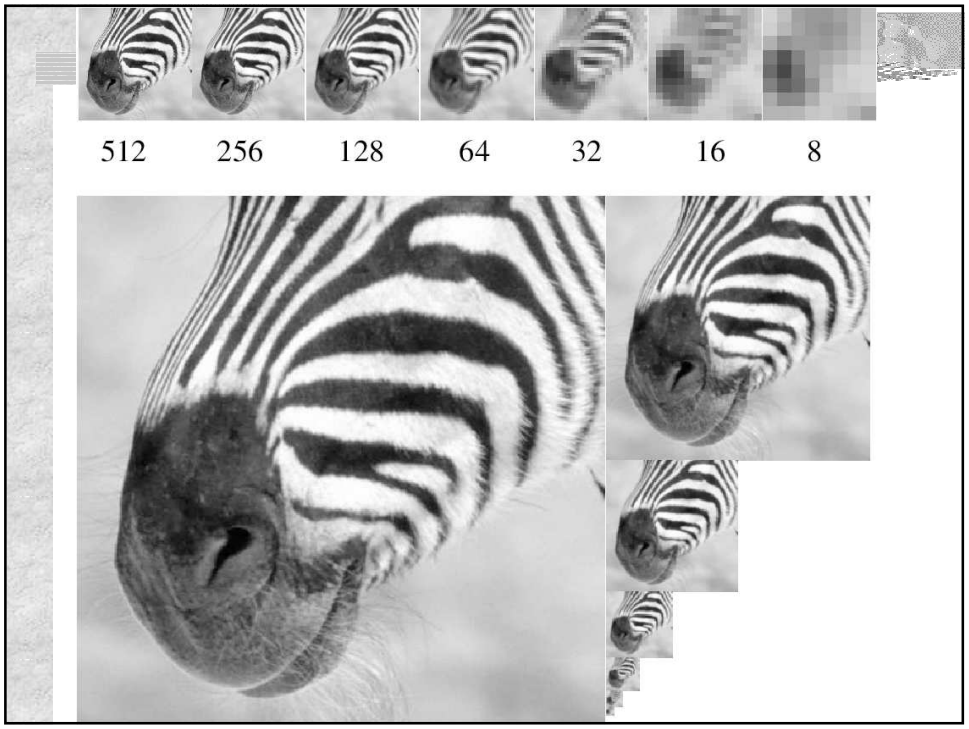
$$\hat{I}(x,y) = I(x,y) * G(x) * G(y)$$

Requires $2kn^2$ multiplications for n by n image and k by k kernel.

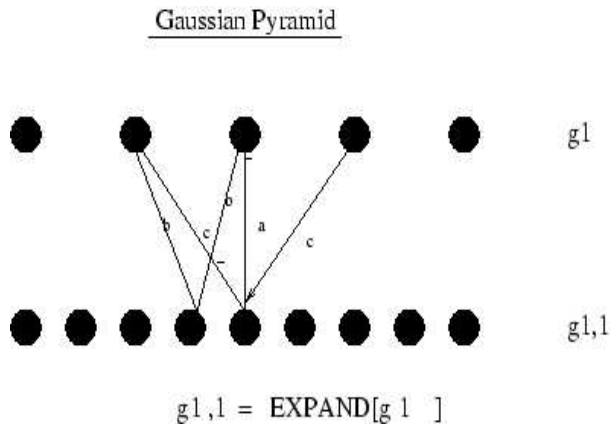
Algorithm

- Apply 1D mask to alternate pixels along each row of image.
- Apply 1D mask to alternate pixels along each colum of resultant image from previous step.




512
256
128
64
32
16
8

Expand (1D)


Gaussian Pyramid
g1
g1,1
b
c
a
b
c
g1 ,1 = EXPAND[g 1]

Expand (1D)

$$g_{l,n}(i) = \sum_{p=-2}^{2} \hat{w}(p) g_{l,n-1}\left(\frac{i-p}{2}\right)$$

$$g_{l,n}(4) = \hat{w}(-2) g_{l,n-1}\left(\frac{4-2}{2}\right) + \hat{w}(-1) g_{l,n-1}\left(\frac{4-1}{2}\right) +$$

$$\hat{w}(0) g_{l,n-1}\left(\frac{4}{2}\right) + \hat{w}(1) g_{l,n-1}\left(\frac{4+1}{2}\right) + \hat{w}(2) g_{l,n-1}\left(\frac{4+2}{2}\right)$$

$$g_{l,n}(4) = \hat{w}(-2) g_{l,n-1}(1) + \hat{w}(0) g_{l,n-1}(2) + \hat{w}(2) g_{l,n-1}(3)$$

Expand (1D)

$$g_{l,n}(i) = \sum_{m=-2}^{2} \hat{w}(p) g_{l,n-1}\left(\frac{i-p}{2}\right)$$

$$g_{l,n}(3) = \hat{w}(-2) g_{l,n-1}\left(\frac{3-2}{2}\right) + \hat{w}(-1) g_{l,n-1}\left(\frac{3-1}{2}\right) +$$

$$\hat{w}(0) g_{l,n-1}\left(\frac{3}{2}\right) + \hat{w}(1) g_{l,n-1}\left(\frac{3+1}{2}\right) + \hat{w}(2) g_{l,n-1}\left(\frac{3+2}{2}\right)$$

$$g_{l,n}(3) = \hat{w}(-1) g_{l,n-1}(1) + \hat{w}(1) g_{l,n-1}(2)$$

The Laplacian Pyramid

- Similar to edge detected images
- Most pixels are zero
- Can be used in image compression

Constructing Laplacian Pyramid

- Compute Gaussian pyramid

$$g_k, g_{k-1}, g_{k-2}, \cdots g_2, g_1$$

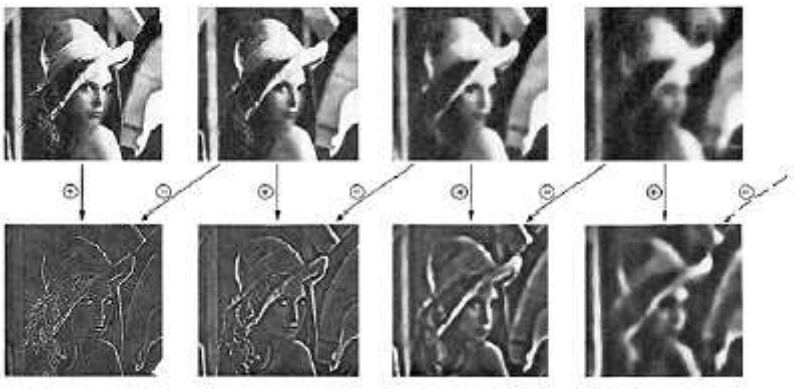
- Compute Laplacian pyramid as follows:

$$\begin{aligned} L_k &= g_k - EXPAND(g_{k-1}) \\ L_{k-1} &= g_{k-1} - EXPAND(g_{k-2}) \\ L_{k-2} &= g_{k-2} - EXPAND(g_{k-3}) \\ &\vdots \\ L_1 &= g_1 \end{aligned}$$

Reconstructing Image

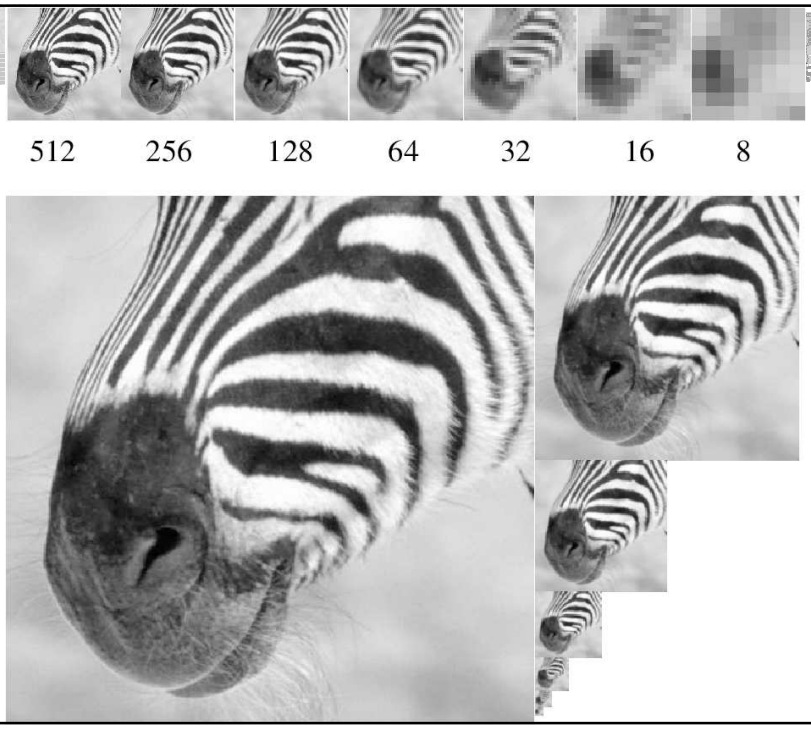
$$
\begin{aligned}
g_1 &= L_1 \\
g_2 &= EXPAND(g_1) + L_2 \\
g_3 &= EXPAND(g_2) + L_3 \\
&\vdots \\
g_k &= EXPAND(g_{k-1}) + L_k
\end{aligned}
$$

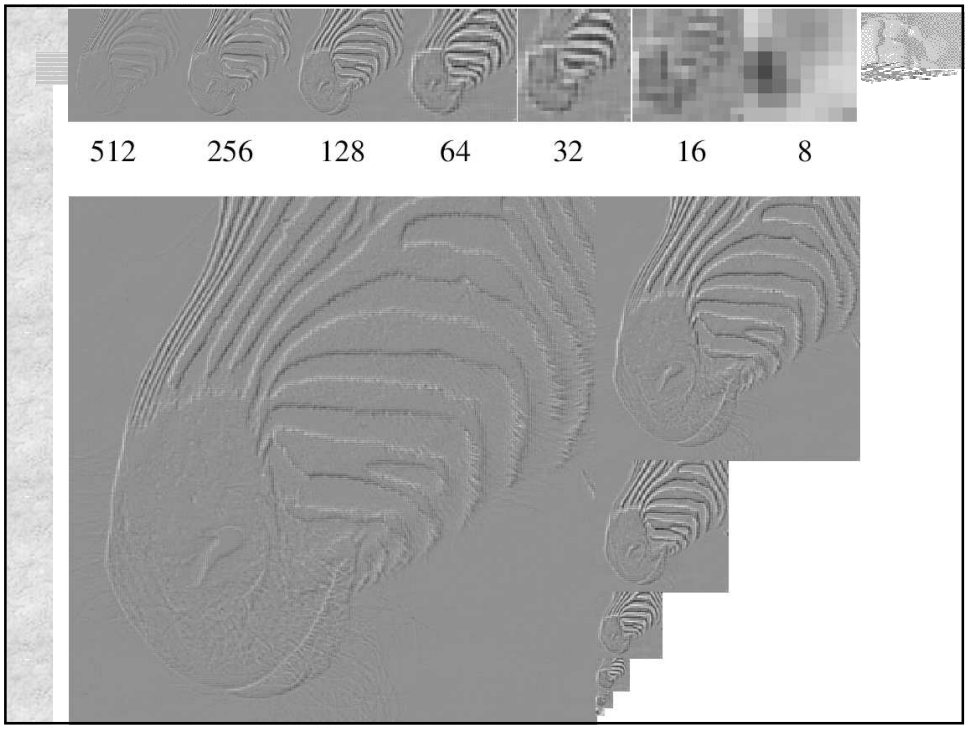
The Laplacian Pyramid



The Laplacian Pyramid

- Synthesis
  - preserve difference between upsampled Gaussian pyramid level and Gaussian pyramid level
  - band pass filter - each level represents spatial frequencies (largely) unrepresented at other levels
- Analysis
  - reconstruct Gaussian pyramid, take top layer





Suggested Reading

- Chapter 7, David A. Forsyth and Jean Ponce, "Computer Vision: A Modern Approach"