Topic 07:

Multi-scale image transforms

- Gaussian & Laplacian pyramids
- Linear image transforms
- Haar wavelet transform

Topic 07:

Multi-scale image transforms

- Gaussian & Laplacian pyramids
- Linear image transforms
- Haar wavelet transform

Gaussian pyramid

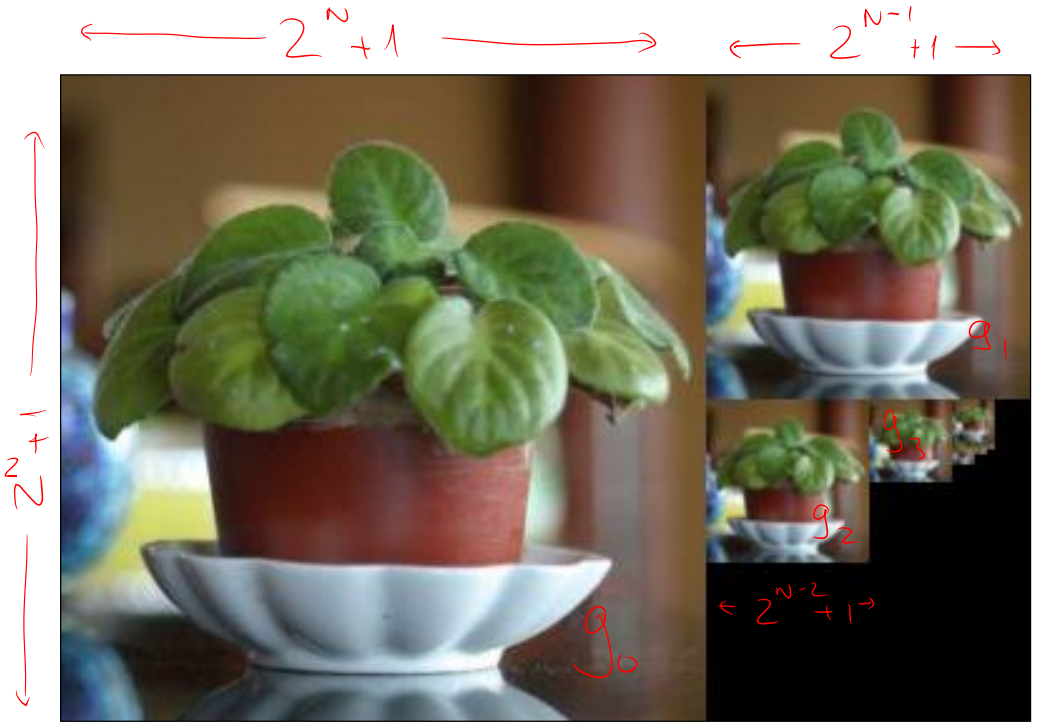
Goal: Develop representation to decompose images into information at multiple scales, to extract features or structures of interest, to attenuate noise

Input:

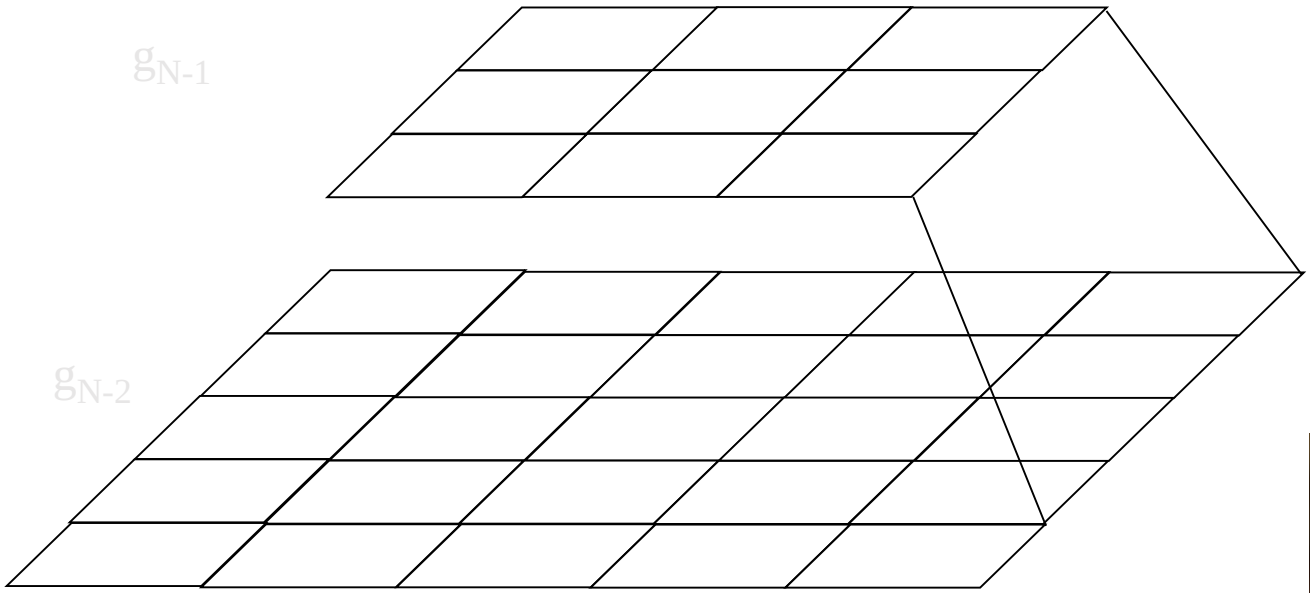
Image I of size $(2^N+1) \times (2^N+1)$

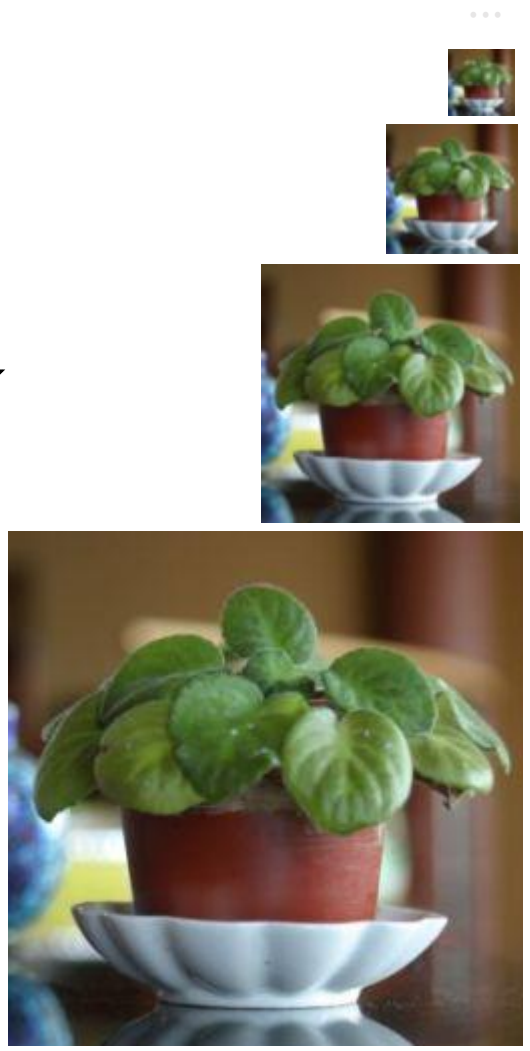
Output:

- N images $g_0, \ldots, g_{N-1}$
- g_ℓ has size $(2^{N-\ell}+1) \times (2^{N-\ell}+1)$



why a “pyramid”?

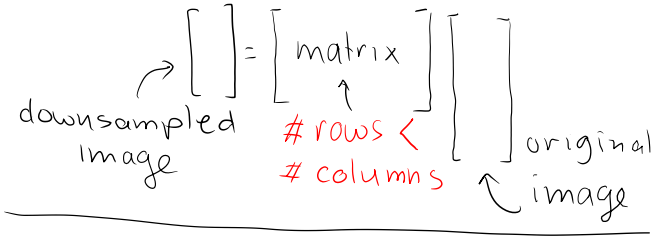




Gaussian pyramid

The representation is based on 2 basic operations

Aside: Downsampling = any linear transformation



1. **Smoothing**

   Smooth the image with a sequence of smoothing filters each of which has twice the radius of the previous one

2. **Downsampling**

   Reduce image size by 1/2 after each smoothing

operation #1: recursive smoothing with filter w

Original photo g_0

g_0

$$\hat{g}_1 = w * g_0$$

a 5x5 filter

$$\hat{g}_1(i,j) = \sum_{m=-2}^{2} \sum_{n=-2}^{2} w(m,n) \cdot g_0(i-m, j-n)$$

operation #1: recursive smoothing with filter w

$\hat{g}_1$

$$\hat{g}_2 = w * \hat{g}_1$$

$$= w * (w * g_0)$$

$$= (\underbrace{w * w}) * g_0$$

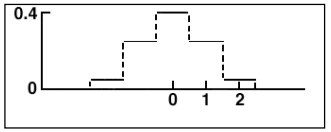
can be thought of as a filter

$h = w * w$

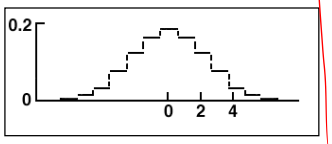
whose radius is twice that of w

In 1D:

w:



w*w:



operation #1: recursive smoothing with filter w

$\hat{g}_2$

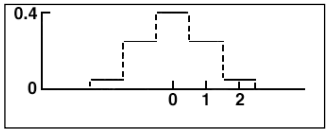
$$\hat{g}_3 = w * \hat{g}_2$$

$$= (w * w * w) * g_0$$

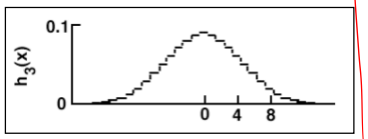
radius is 4 times that of w

In 1D:

w:



w*w*w:



operation #1: recursive smoothing with filter w

$\hat{g}_3$

$$\hat{g}_4 = w * \hat{g}_3$$

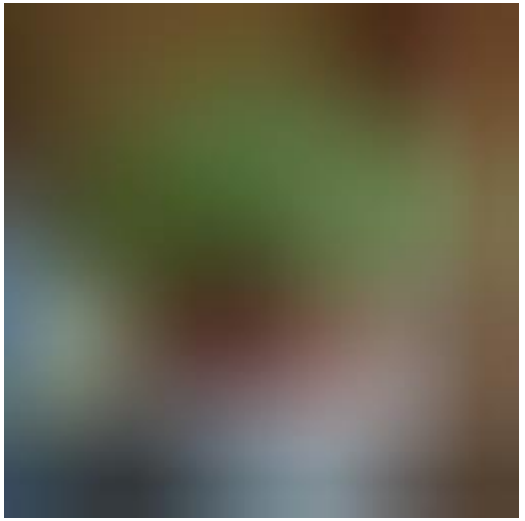
$$= (w * w * w * w) * g_0$$

operation #1: recursive smoothing with filter w

$\hat{g}_4$

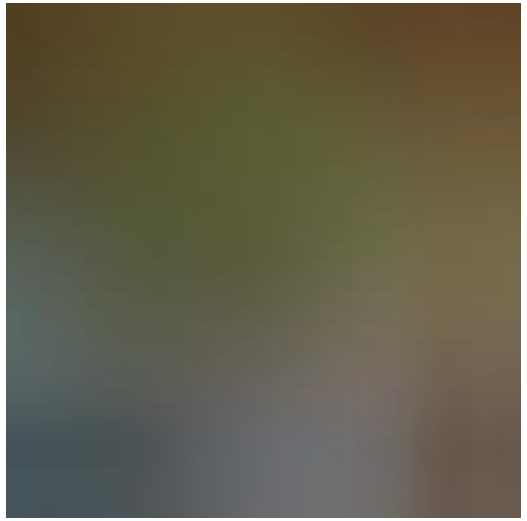
operation #1: recursive smoothing with filter w

$\hat{g}_s$



operation #1: recursive smoothing with filter w

$\hat{g}_6$



operation #1: recursive smoothing with filter w

$\hat{g}_7$

deriving the smoothing filter in 1D

① $\hat{w}$ always has 5 elements (a.k.a "5-tap" filter)

② $\hat{w}$ symmetric about 0:

$$\hat{w} = \begin{bmatrix} c \\ b \\ a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 1/4 - a/2 \\ 1/4 \\ a \\ 1/4 \\ 1/4 - a/2 \end{bmatrix}$$

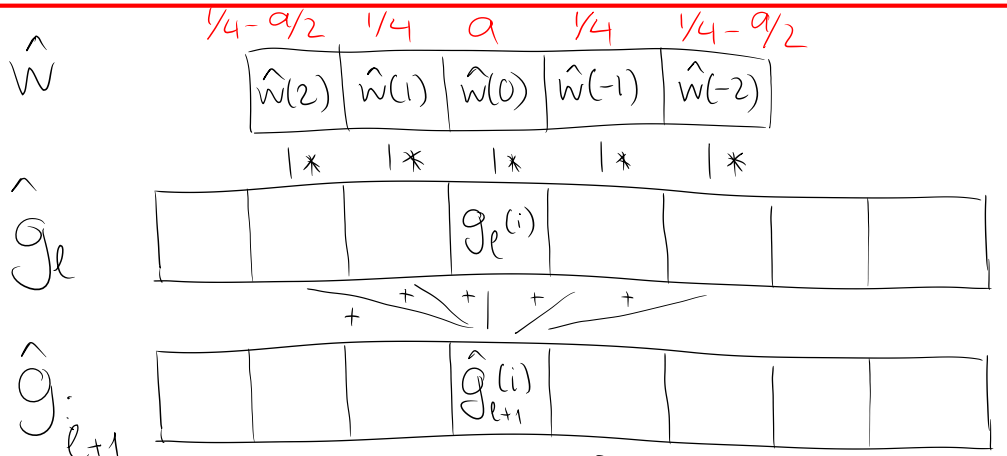
③ applying $\hat{w}$ to a constant image does not change it

$\Leftrightarrow$

$$\sum_{m=-2}^{2} \hat{w}(m) = 1$$

$\Leftrightarrow$

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



$$\hookrightarrow \sum_{m=-2}^{2} \hat{w}(m) \cdot \hat{g}_\ell(i-m)$$

④ Equal contribution.

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

To satisfy Criteria 1–4 we have 2 eqs & 3 unknowns

$\Rightarrow$ a remains free parameter

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

usually $a \in [0.3, 0.6]$

operation #2: recursive downsampling

Since $\hat{g}_1$ contains less image detail, we downsample it by 2 (i.e. store every other pixel)

$$g_1(i,j) = \hat{g}_1(2i, 2j)$$

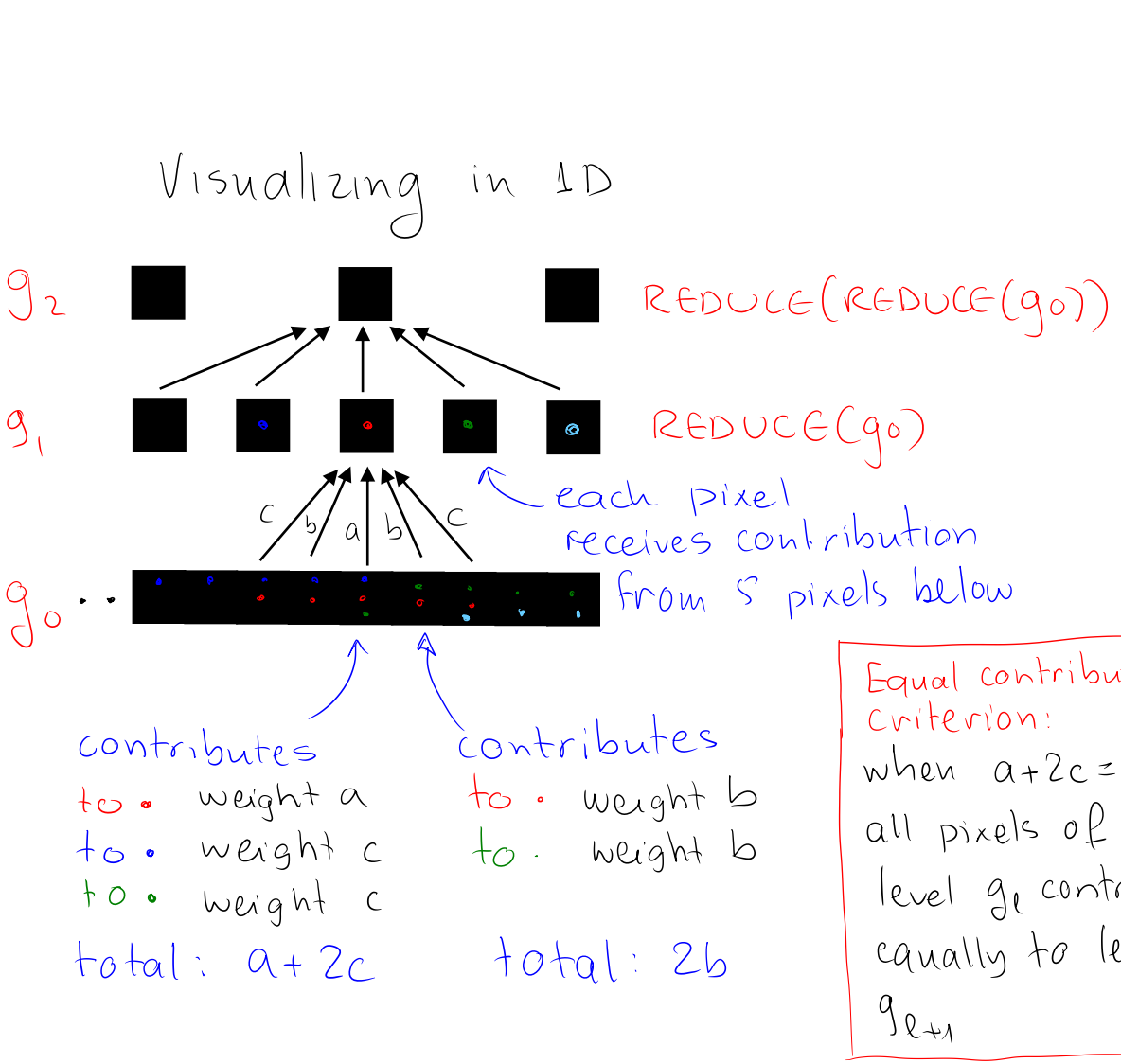
$\hat{g}_1$

$2^N + 1$

g_1

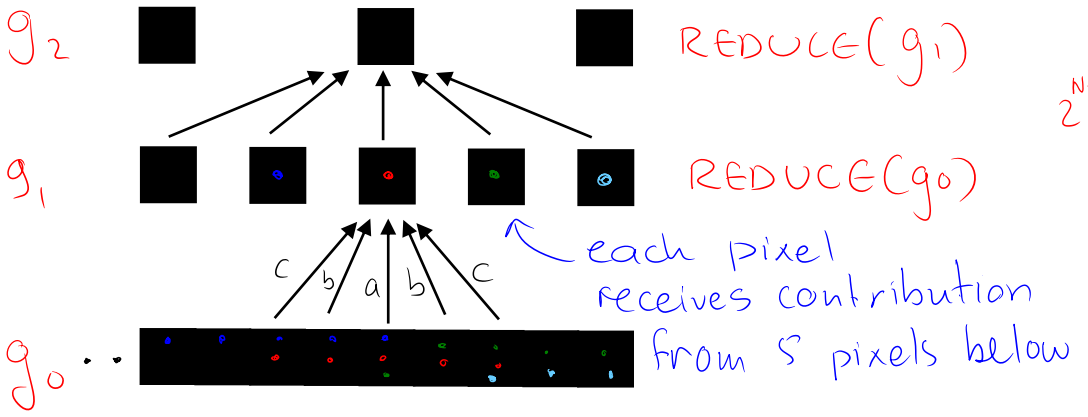
$2^{N-1} + 1$

the REDUCE() function



the REDUCE() function

Visualizing in 1D



REDUCE function in matrix notation (1D)

$$\begin{matrix} 2^{N-1}+1 \\ \updownarrow \end{matrix}\begin{bmatrix} g_1 \end{bmatrix} = \underbrace{\begin{bmatrix} 1\,0\,0\cdots & 0 \\ 0\,0\,1\cdots & 0 \\ 0\,0\,0\,0\,1\cdots & 0 \end{bmatrix}}_{\text{downsampling matrix } D_0} \underbrace{\begin{bmatrix} c\,b\,a\,b\,c & 0 \\ & c\,b\,a\,b\,c \\ & \ddots \\ 0 & \ddots \end{bmatrix}}_{\text{convolution matrix } C_0} \begin{bmatrix} g_0 \end{bmatrix}$$

($\leftarrow 2^N+1 \rightarrow$ over the D_0 matrix; $\leftarrow 2^N+1 \rightarrow$ over the C_0 matrix)

$$g_1 = D_0 \cdot C_0 g_0$$

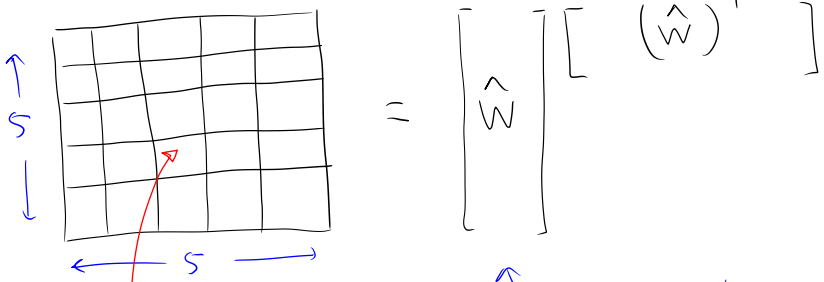
General expression:

$$g_{l+1} = D_l \cdot C_l \cdot g_l$$

smoothing in 2D

$\hat{g}_1 = w * g_0$

w is a separable smoothing filter defined by $\hat{w}$



$w(m,n) = \hat{w}(m) \cdot \hat{w}(n)$

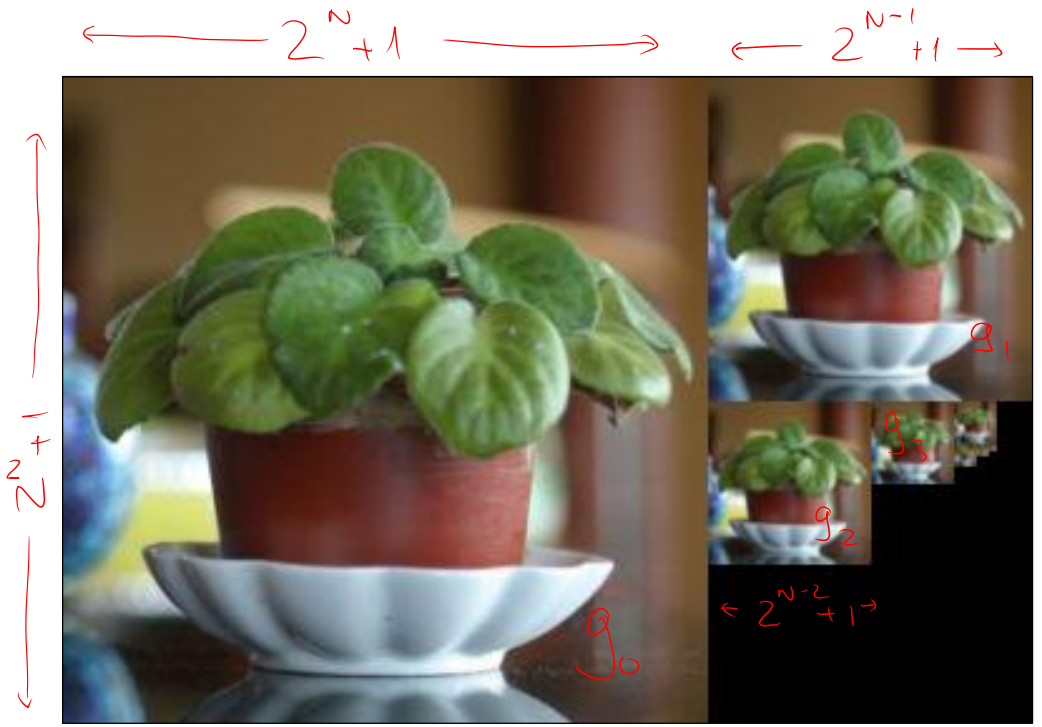
g_0

Exploiting separability to compute $\hat{g}_1$:

1. Convolve each row of g_0 with $\hat{w}$
2. Convolve the columns of the result with $\hat{w}$ again

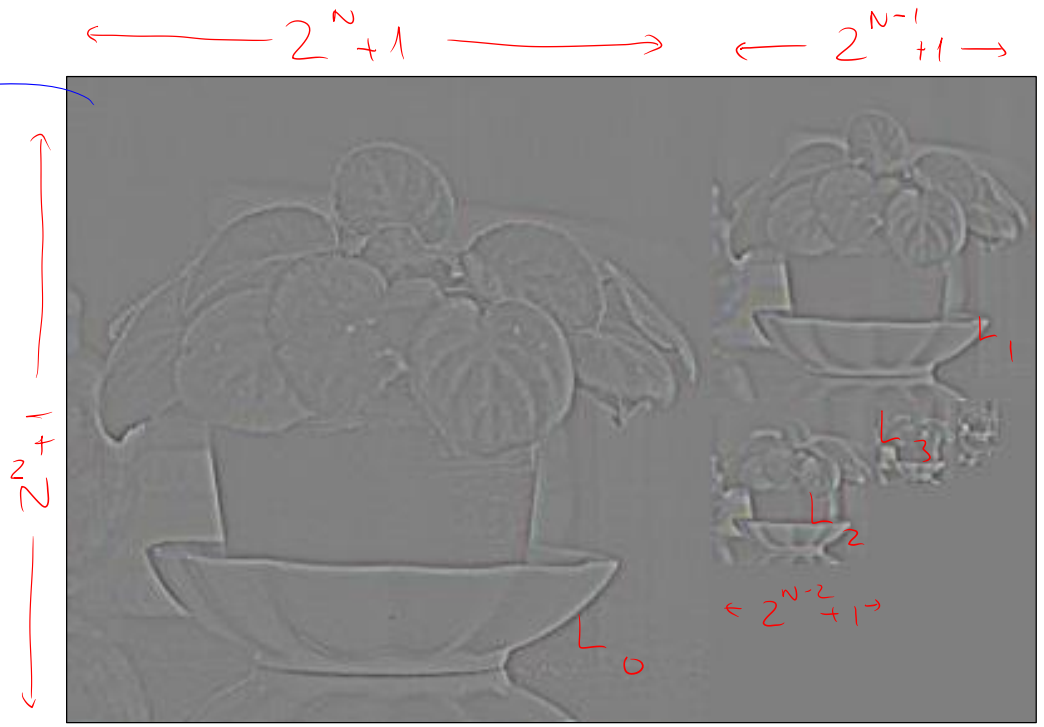
the Laplacian Pyramid

Idea: Rather than store the smoothed images, store only the difference between levels g_k and g_{k+1}



the Laplacian Pyramid

most values are 0 (gray) ⇒ can be quantized & represented with few bits

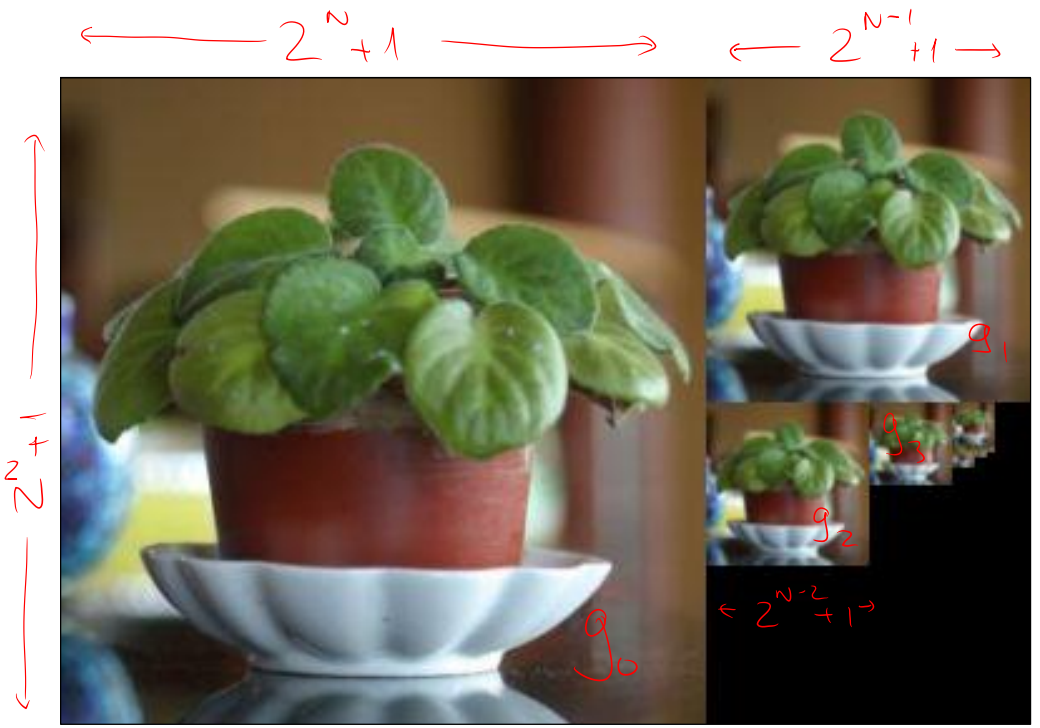


the Laplacian Pyramid

Idea: Rather than store the smoothed images, store only the difference between levels g_ℓ and $g_{\ell+1}$

- To do this, we must compare adjacent levels g_ℓ and $g_{\ell+1}$
- But $g_{\ell+1}$, g_ℓ are not the same size!

  $\Rightarrow$ Expand $g_{\ell+1}$ to make it equal size to g_ℓ



operation #3: the EXPAND() function

$\leftarrow 2^{N-\ell}+1 \rightarrow$

g_ℓ

EXPAND( )

$\Longrightarrow$

$\leftarrow 2^{N-\ell+1}+1 \rightarrow$

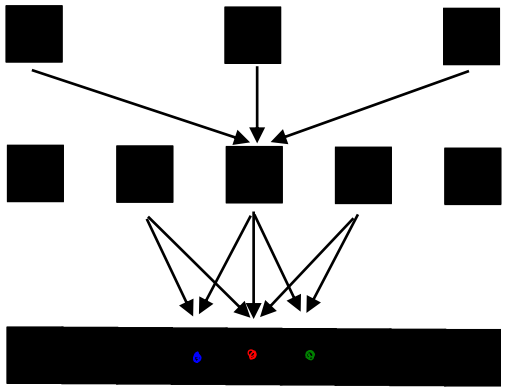
EXPAND(g_ℓ)

$$\text{EXPAND}(g_\ell) = 4\sum_{m=-2}^{2}\sum_{n=-2}^{2} w(m,n)\, g_\ell\left(\frac{i-m}{2}, \frac{j-n}{2}\right)$$

the EXPAND() function

- The function upsamples level g_ℓ by doubling its size from $(2^{N-\ell}+1)\times(2^{N-\ell}+1)$ to $(2^{N-\ell+1}+1)\times(2^{N-\ell+1}+1)$

Visualizing in 1D



g_1

EVEN pixels receive contribution from 3 pixels above them (total weight = $2c+a=1/2$)

$\uparrow$

$\uparrow$

ODD pixels receive contribution from 2 pixels above them (total weight = $2b=1/2$)

$\text{EXPAND}(g_1)$

General expression in 1D

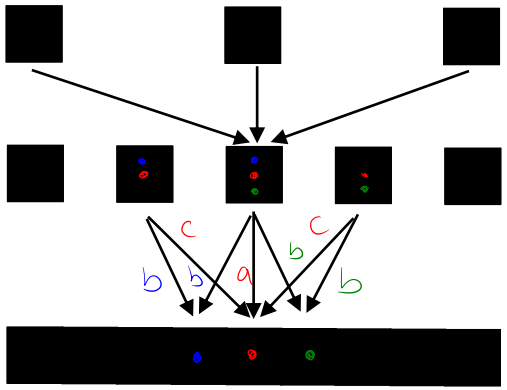
$$\text{EXPAND}(g_\ell)(i) = 2\sum_{m=-2}^{2} \hat{w}(m)\cdot g_\ell\left(\frac{i-m}{2}\right)$$

evaluated only for i, m where $\frac{i-m}{2}$ is an integer

the EXPAND() function

- The function upsamples level g_ℓ by doubling its size from $(2^{N-\ell}+1)\times(2^{N-\ell}+1)$ to $(2^{N-\ell+1}+1)\times(2^{N-\ell+1}+1)$

Visualizing in 1D



EVEN pixels receive contribution from 3 pixels above them (total weight = $2c+a=1/2$)

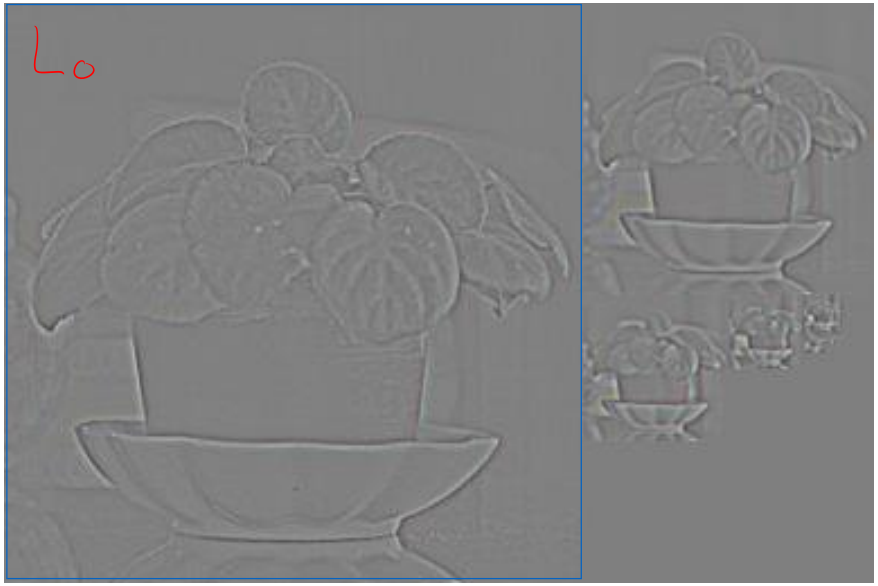
ODD pixels receive contribution from 2 pixels above them (total weight = $2b=1/2$)

EXPAND function in matrix notation (1D)

$$\underbrace{\begin{bmatrix} EXPAND(g_1) \end{bmatrix}}_{2^N+1 \times 1} = \underbrace{\begin{bmatrix} c & b & a & b & c & 0 & \cdots \\ & c & b & a & b & c & \cdots \\ & & & \ddots & & & \end{bmatrix}}_{\text{convolution matrix } C_0} \underbrace{\begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 0 & \cdots & 0 \\ & & \cdots & & \end{bmatrix}}_{\text{upsampling matrix } (D_0)^T} \underbrace{\begin{bmatrix} g_1 \end{bmatrix}}_{2^{N-1}+1}$$

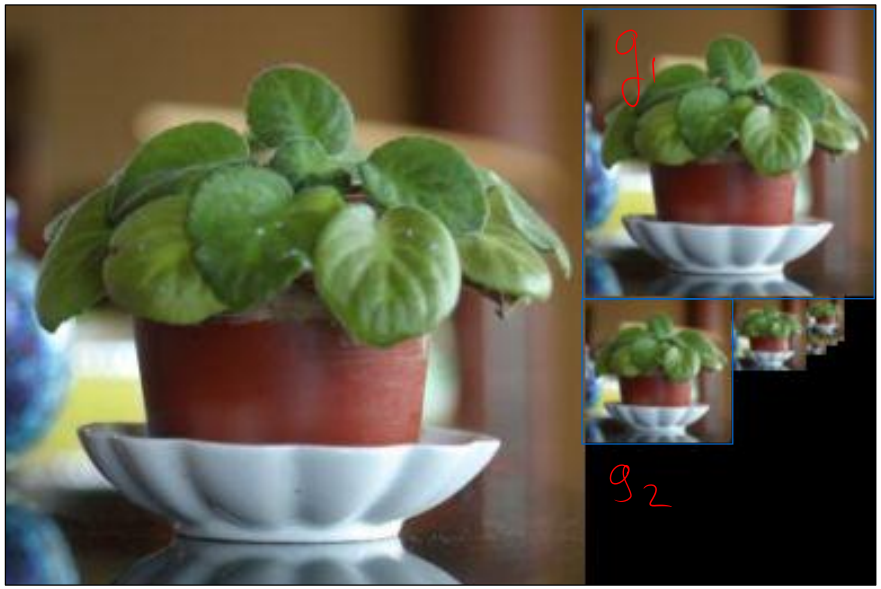
$$EXPAND(g_1) = C_0 \cdot (D_0)^T g_1$$

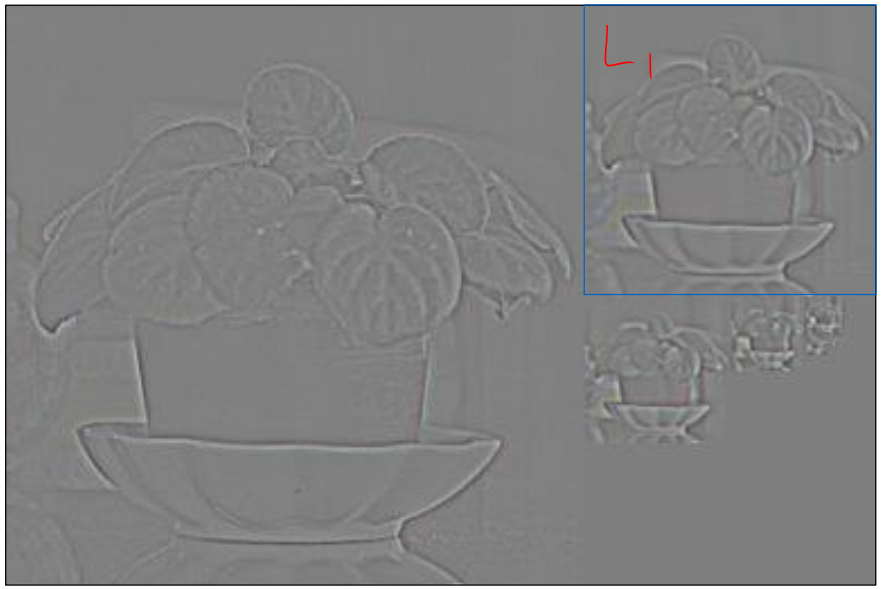
the Laplacian Pyramid



the Laplacian Pyramid

$L_1 = g_1 - \text{EXPAND}(g_2)$





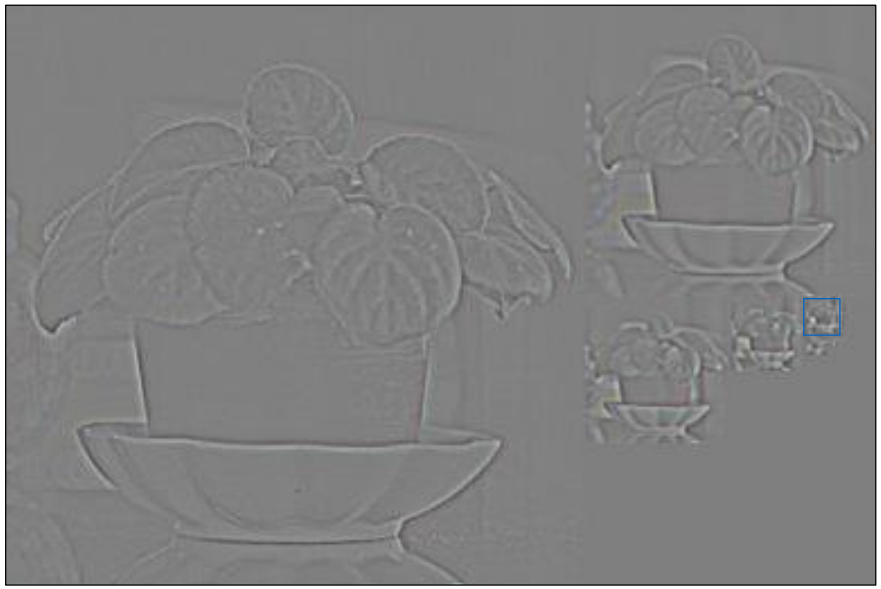
the Laplacian Pyramid

$L_2 = g_2 - \text{EXPAND}(g_3)$

the Laplacian Pyramid

Often we use a truncated Laplacian pyramid by storing images $L_0, L_1, \ldots, L_K, g_{K+1}$ for $K+1 < N$

Base case: store g_N



application: progressive transmission

RECEIVER

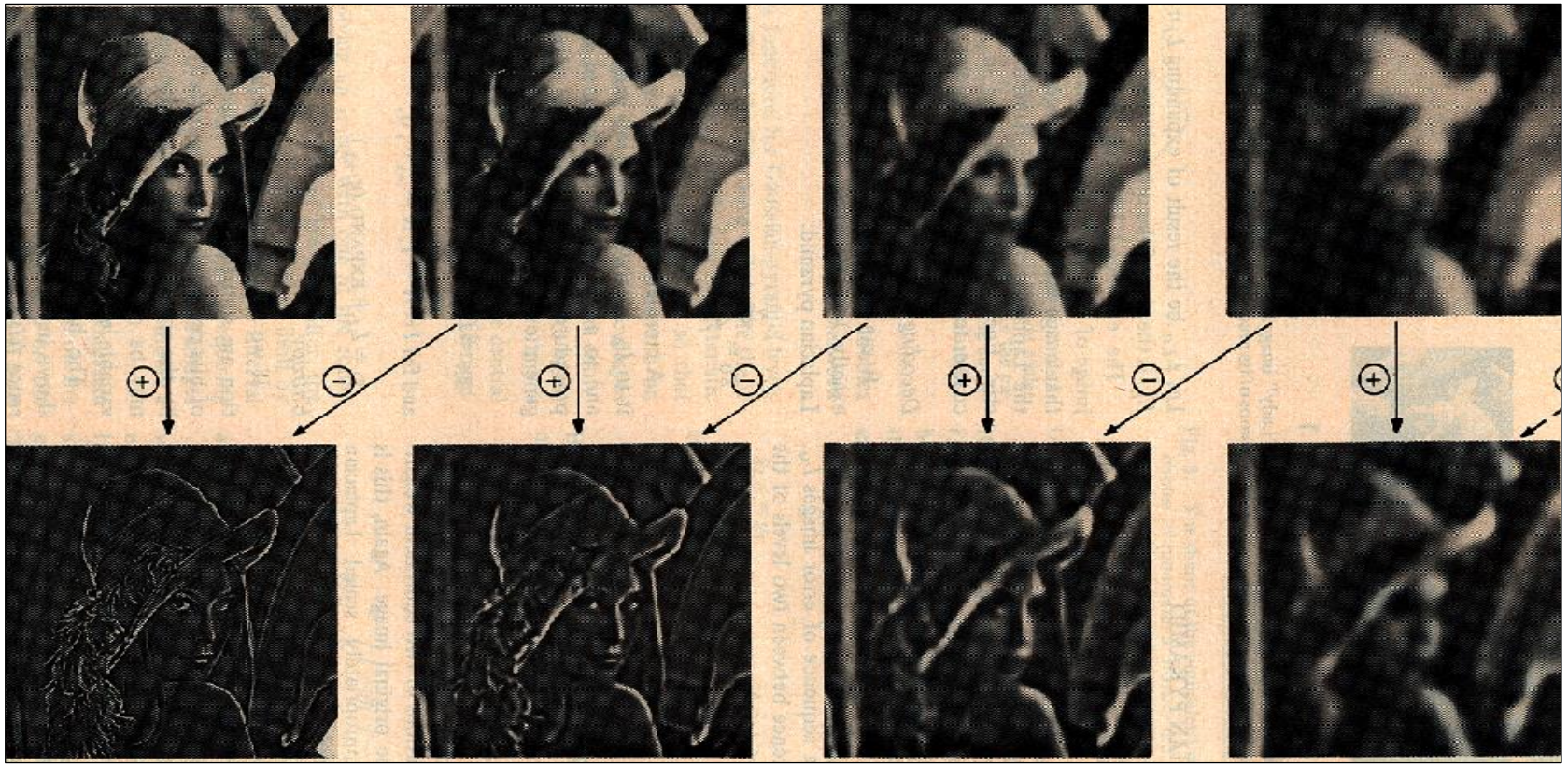
TRANSMITTER

③ Display $d_3 = d_2 + \text{EXPAND}^{K-2}(L_{K-2})$

② Display $d_2 = d_1 + \text{EXPAND}^{K-1}(L_{K-1})$

① Display $d_1 = \text{EXPAND}^{K}(g_K)$

① Transmit g_K



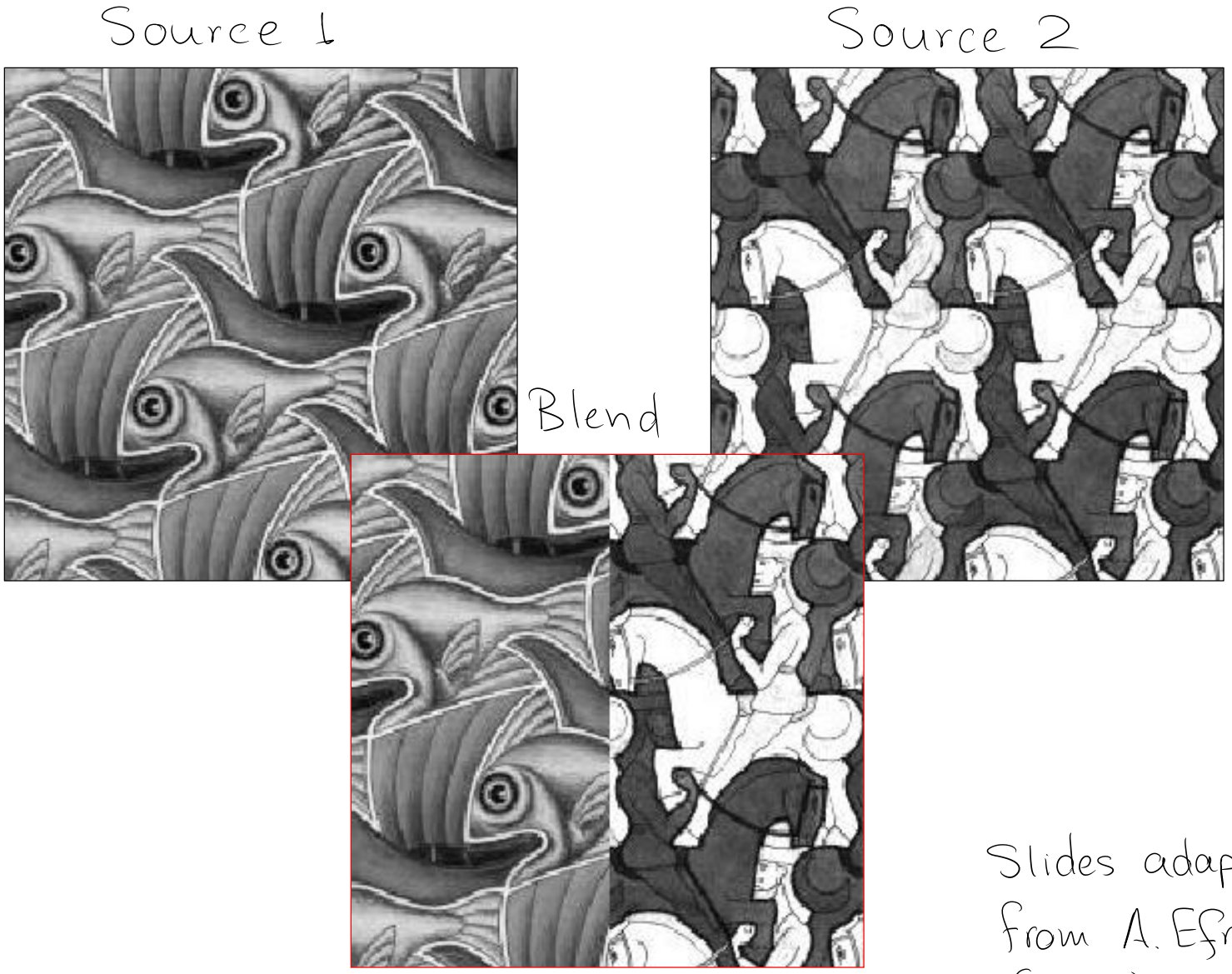
⑤ Transmit L_{K-4}

④ Transmit L_{K-3}

③ Transmit L_{K-2}

② Transmit L_{K-1}

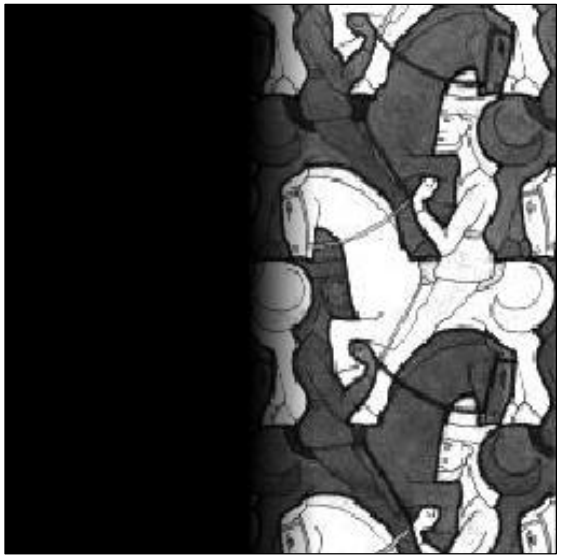
application: Laplacian image blending



Slides adapted from A. Efros (CMU)

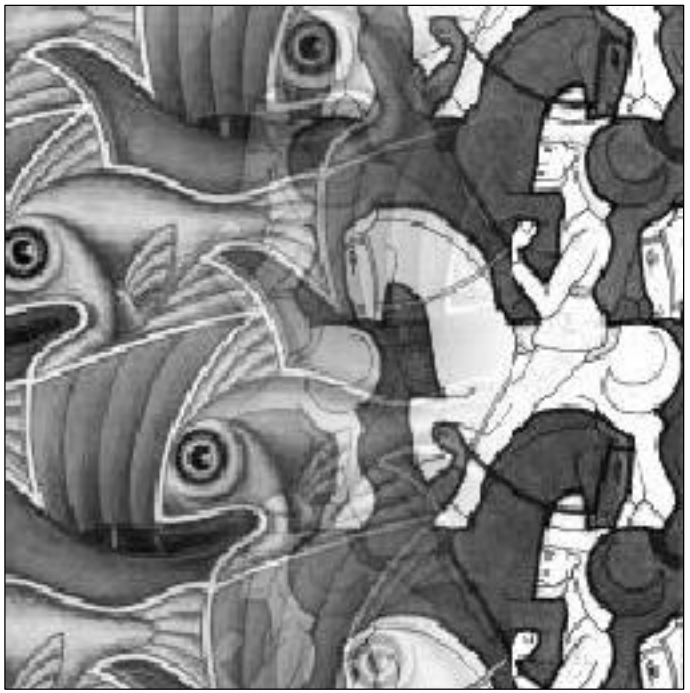
feathering

+



=

feathering: effect of Window Size



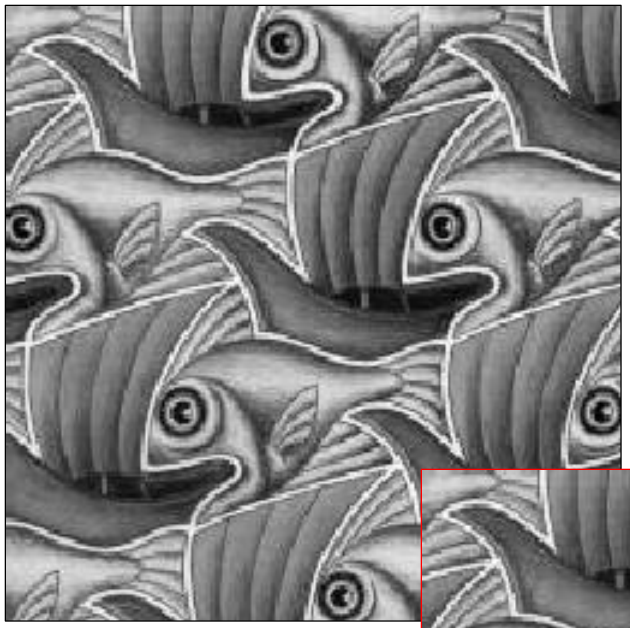
Ghosting is most apparent here

feathering: effect of Window Size

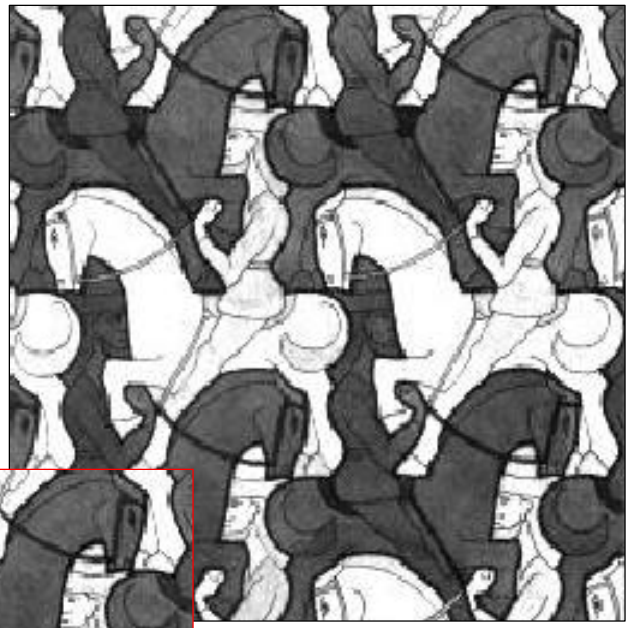
seam most visible here

Laplacian image blending (Burt & Adelson, 1983)

Source A



Source B



Blend

Main challenge:

- Minimize ghosting without making visible seams

Slides adapted from A. Efros (CMU)

pyramid blending algorithm

Input: Source images A, B & binary matte M
Output: Blended image S

A

B

pyramid blending algorithm

Input: Source images A, B & binary matte M
Output: Blended image S

① Compute A's Laplacian pyramid: $AL_0, \ldots, AL_{N-1}, Ag_N$

③ Compute M's Gaussian pyramid: $Mg_0, \ldots, Mg_N$

② Compute B's Laplacian pyramid: $BL_0, \ldots, BL_{N-1}, Bg_N$

pyramid blending algorithm

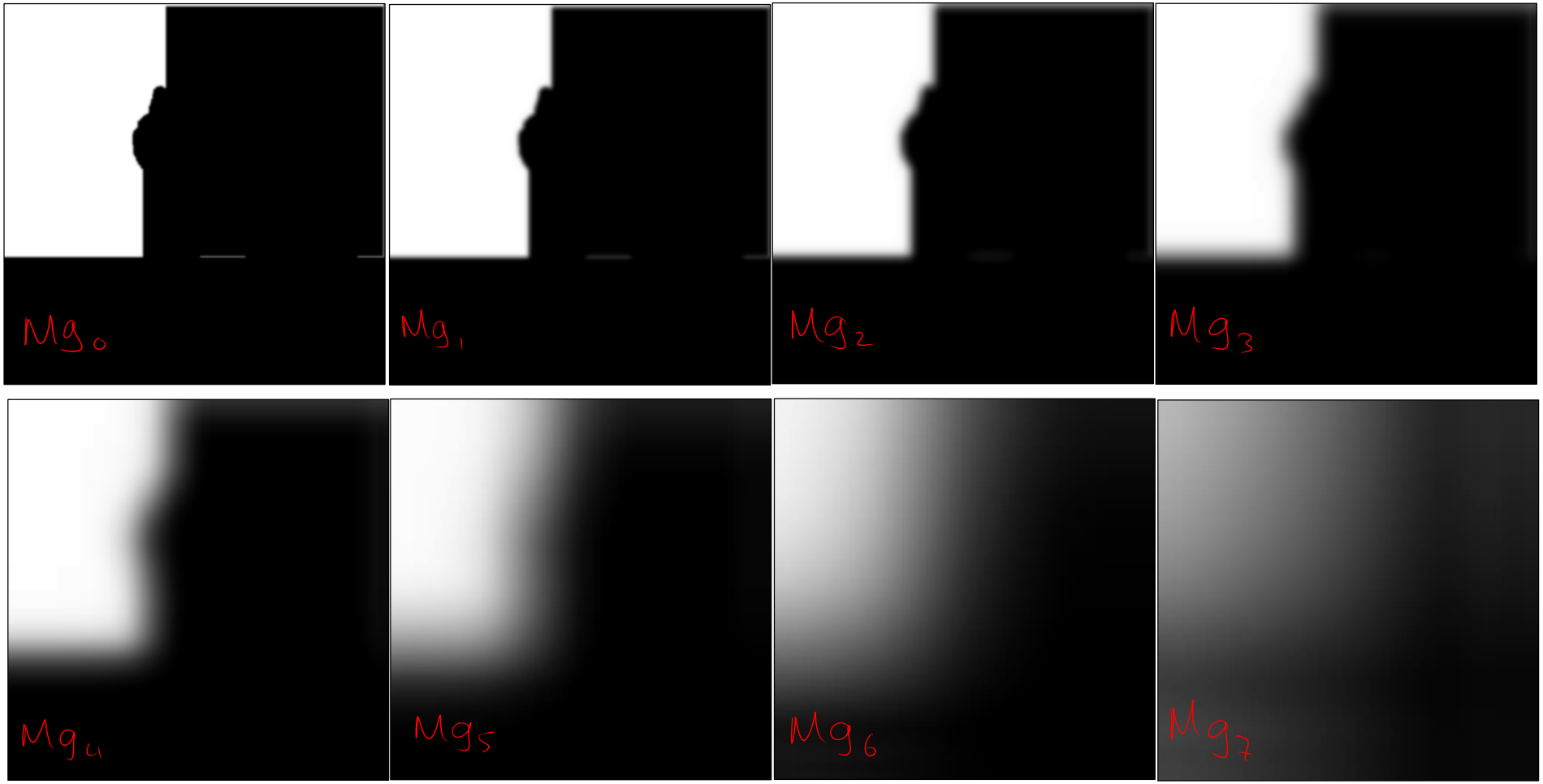
④ Compute the Laplacian pyramid, $SL_0, SL_1, \ldots SL_{N-1}, Sg_N$ by applying the matting equation with matte Mg_ℓ:

$$SL_\ell(i,j) = Mg_\ell(i,j)\,AL_\ell(i,j) + (1 - Mg_\ell(i,j))\,BL_\ell(i,j)$$

pyramid blending algorithm: intuition

The algorithm effectively uses a different alpha matte for each level of detail

↑detail ⇒ ↓feathering window ↓detail ⇒ ↑feathering window



pyramid blending algorithm

⑤ Compute level 0 of the Gaussian pyramid of S from $SL_0, \ldots, SL_{N-1}, Sg_N$

Result S'

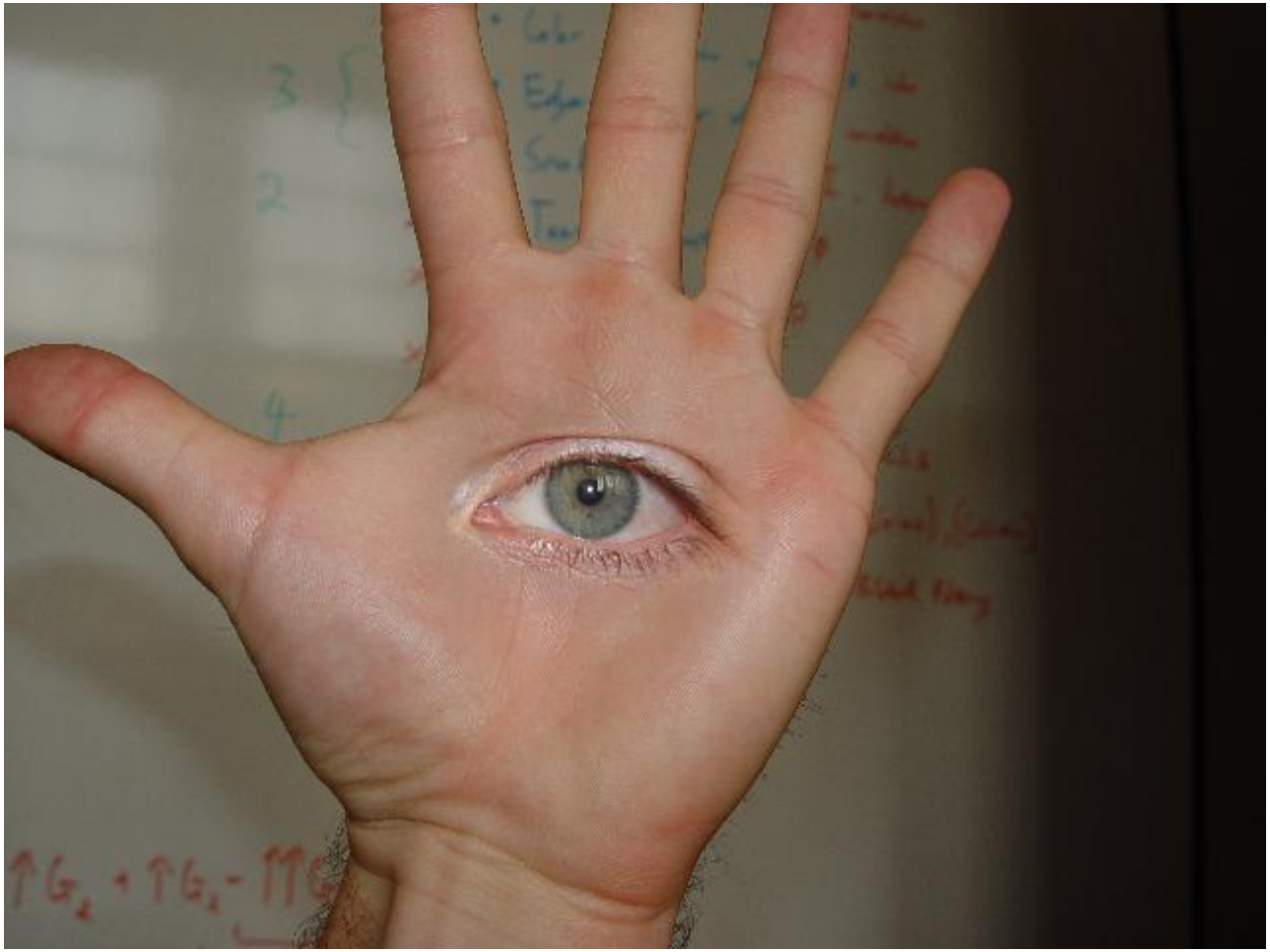
stitching photos with pyramid blending

stitching photos (no pyramid blend)

blending mismatched photos (still looks OK)

stitching mismatched photos (no pyramid blend)

Horror Photo



Topic 07:

Multi-scale image transforms

- Gaussian & Laplacian pyramids
- Linear image transforms
- Haar wavelet transform

linear transform framework

Projection Vectors: Let $\vec{\mathbf{I}}$ denote a 1D signal, or a vectorized representation of an image (so $\vec{\mathbf{I}} \in \mathcal{R}^N$), and let the transform be

$$\vec{\mathbf{a}} = \mathbf{P}^T \vec{\mathbf{I}} . \tag{1}$$

Here,

- $\vec{\mathbf{a}} = [a_0, ..., a_{M-1}] \in \mathcal{R}^M$ are the transform coefficients.
- The columns of $\mathbf{P} = [\vec{\mathbf{p}}_0, \vec{\mathbf{p}}_1, ..., \vec{\mathbf{p}}_{M-1}]$ are the projection vectors; the m^{th} coefficient, a_m, is the inner product ${\vec{\mathbf{p}}_m}^T \vec{\mathbf{I}}$
- When $\mathbf{P}$ is complex-valued, we should replace $\mathbf{P}^T$ by the conjugate transpose $\mathbf{P}^{*T}$

Sampling: The transform $\mathbf{P}^T \in \mathcal{R}^{M \times N}$ is said to be *critically sampled* when $M = N$. Otherwise it is *over-sampled* when $M > N$, or *under-sampled* when $M < N$.

Basis Vectors: For many transforms of interest there is a corresponding basis matrix $\mathbf{B}$ satisfying

$$\vec{\mathbf{I}} = \mathbf{B}\,\vec{\mathbf{a}} . \tag{2}$$

linear transform framework

Completeness

- the forward transform (1) is complete, encoding all image structure, if it is invertible.
- when critically sampled, it is complete if $\mathbf{B} = (\mathbf{P}^T)^{-1}$ exists.
- if over-sampled, the transform is complete if $rank(\mathbf{P}) = N$.
  In this case $\mathbf{B}$ is not unique – one choice is the pseudoinverse

$$\mathbf{B} = (\mathbf{P}^T\mathbf{P})^{-1}\,\mathbf{P}^T$$

- if under-sampled, then $rank(\mathbf{P}) < N$ and it is not invertible in general.

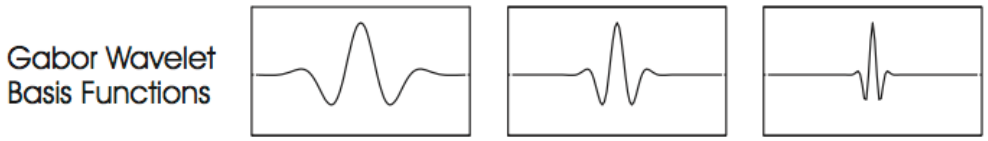
Self-Inverting

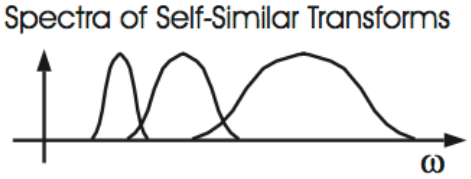
- the transform is self-inverting if $\vec{\mathbf{b}}_m = \alpha\vec{\mathbf{p}}_m$ for some constant α.
- in the critically-sampled, self-inverting case the transform is orthogonal (unitary), up to the constant α (e.g., the DFT).

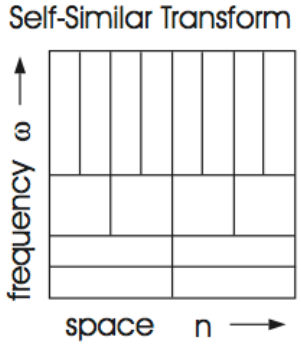
self-similar linear transforms

Goal: The filter support should grow with scale, and be well matched to scale-dependent correlation lengths in images. The represenation should exhibit scale-invariant properties, as objects project to images at different scales depending on distance from camera.

Scale Self-Similarity: Let the basis functions be dilations and translations of a "mother" function, so they all have the same shape, differing in scale and position only.







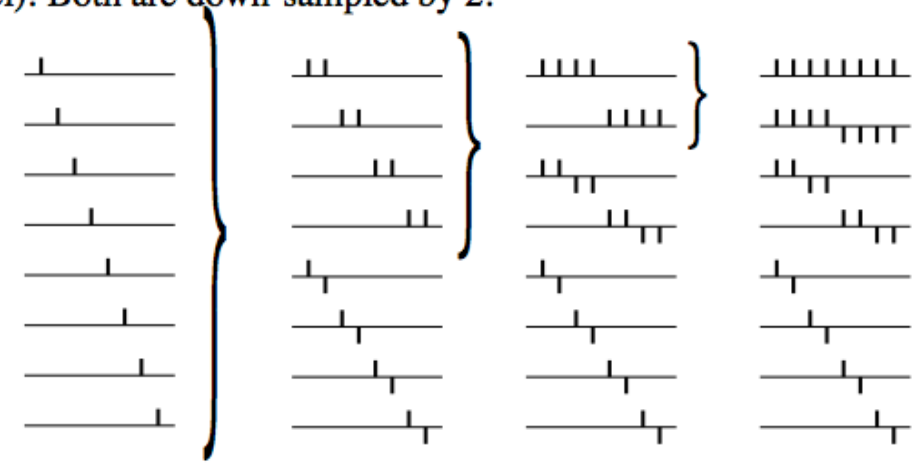
Topic 07:

Multi-scale image transforms

- Gaussian & Laplacian pyramids
- Linear image transforms
- Haar wavelet transform

Haar wavelet transform

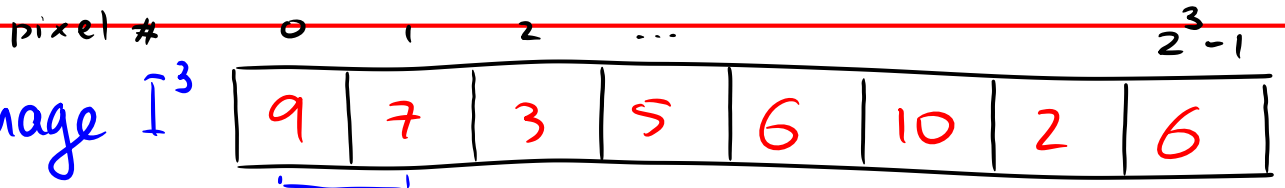
Originally described by A. Haar (1909). Each step creates two channels: one simply averages adjacent elements (i.e., low-pass channel); and one takes difference between adjacent elements (i.e., a high-pass channel). Both are down-sampled by 2.

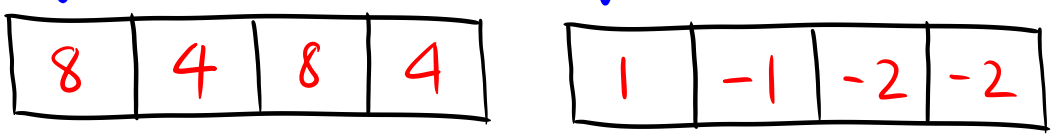


Properties:

- critically-sampled and self-inverting (orthogonal)
- local in space (compact) but not continuously differentiable
- broad ringing frequency spectrum due to top-hat spatial window, and therefore massive aliasing in each band (like blocked DCT).
- very efficient to compute with pyramid scheme and addition

1D Haar wavelet transform: recursive definition





Average I^2

(analogous to level g_0 of Gaussian pyramid)

average of first two pixels

difference of first pixel from the average

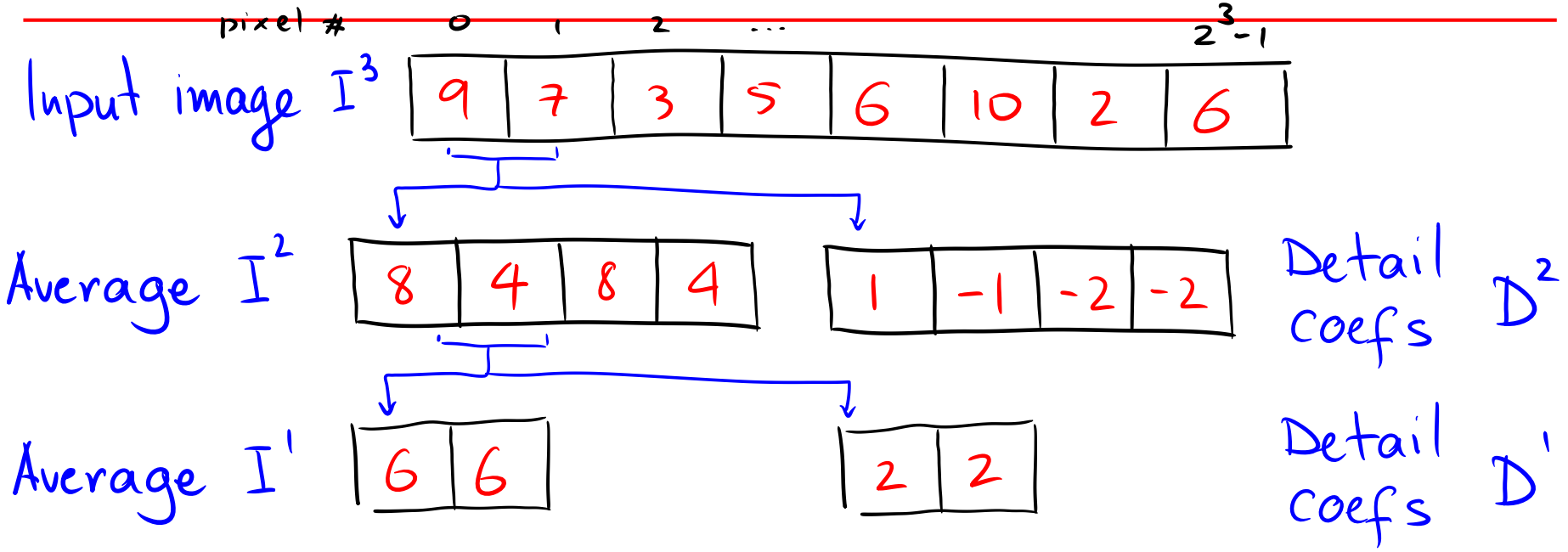
Detail coefs D^2

(analogous to level L_0 of Laplacian pyramid)

(note: we don't need to store difference of 2nd pixel from average ⇒

D^0 has ½ the size of the corresponding Laplacian level L_0!)

1D Haar wavelet transform: recursive definition



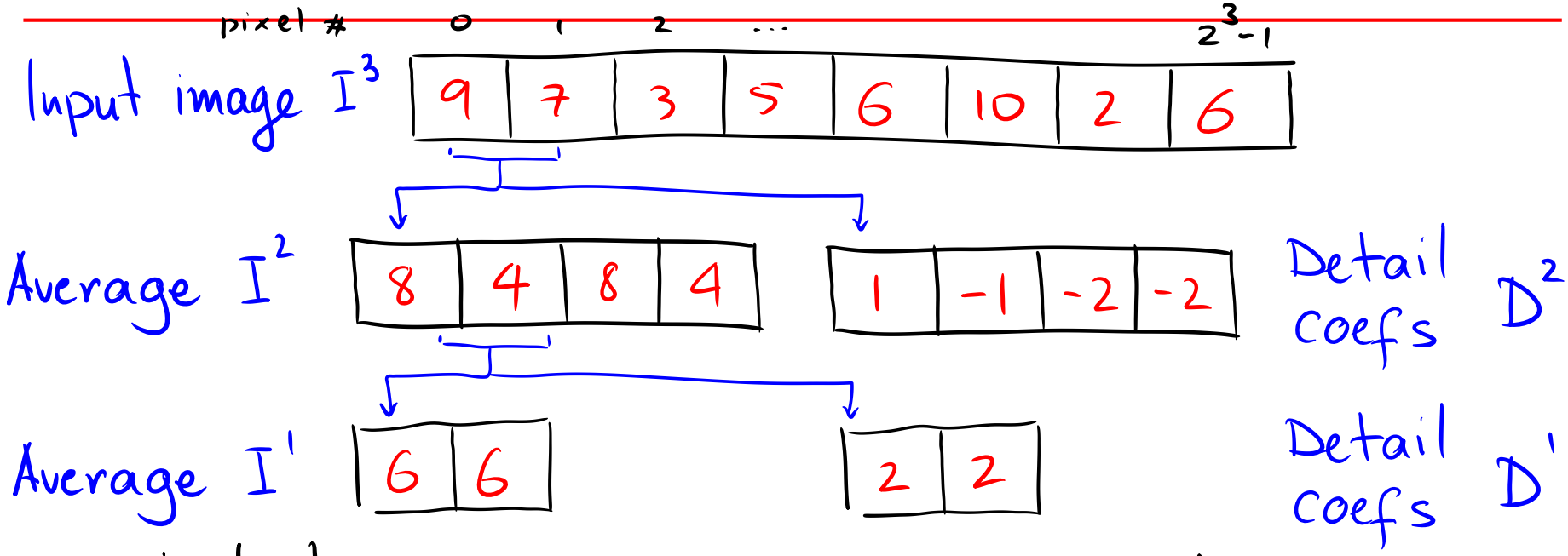
$$I_i^j = \frac{1}{2}\left(I_{2i}^{j+1} + I_{2i+1}^{j+1} \right)$$

j-th level of "pyramid" contains 2^j pixels

$$D_i^j = I_{2i}^{j+1} - \frac{1}{2}\left(I_{2i}^{j+1} + I_{2i+1}^{j+1} \right)$$

$$= \frac{1}{2}\left(I_{2i}^{j+1} - I_{2i+1}^{j+1} \right)$$

1D Haar wavelet transform: recursive definition



equivalent Convolution mask ϕ: $\frac{1}{2}$ $\frac{1}{2}$

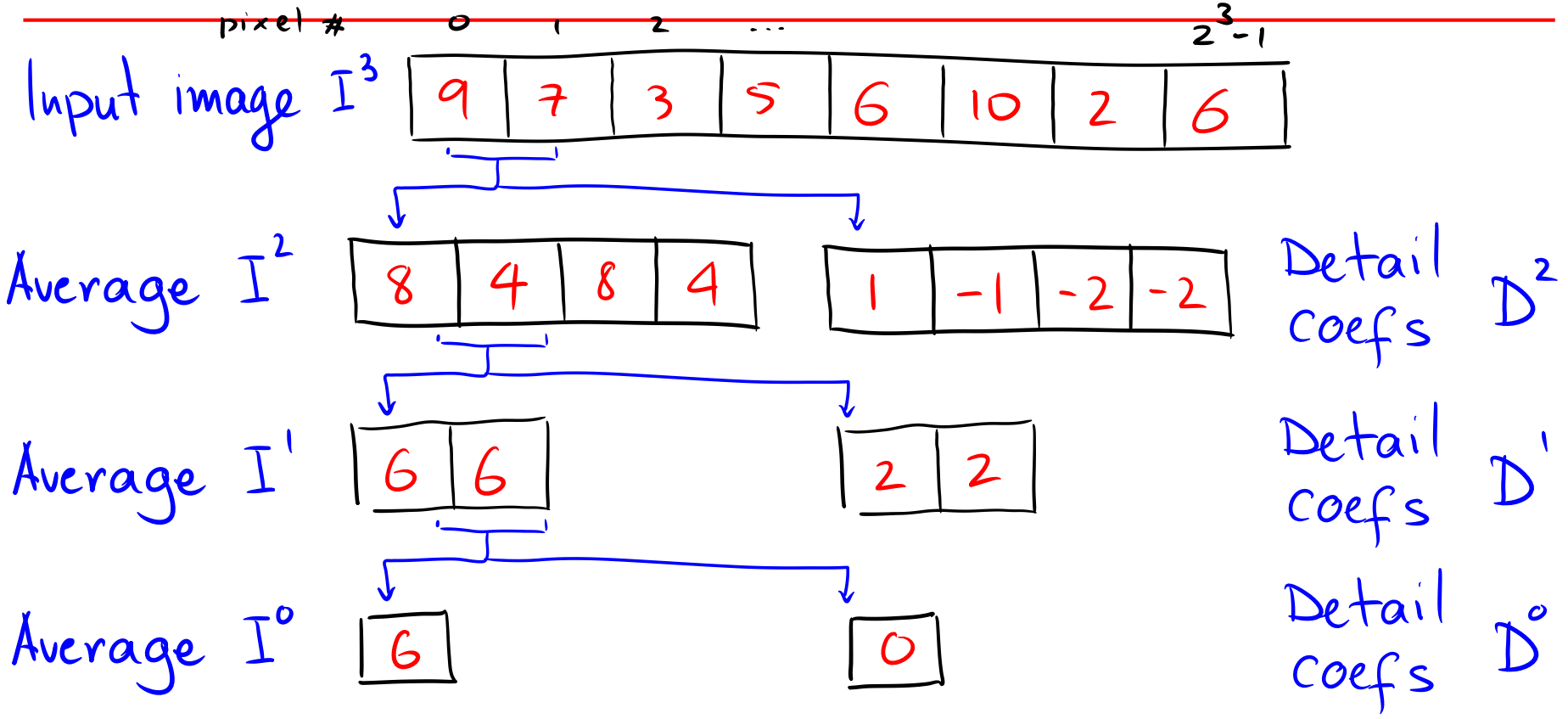
equivalent Convolution mask ψ: $-\frac{1}{2}$ $\frac{1}{2}$

$$I_i^j = \frac{1}{2}\left(I_{2i}^{j+1} + I_{2i+1}^{j+1} \right)$$

$$D^j = \frac{1}{2}\left(I_{2i}^{j+1} - I_{2i+1}^{j+1} \right)$$

j-th level of "pyramid" contains 2^j pixels

1D Haar wavelet transform: recursive definition

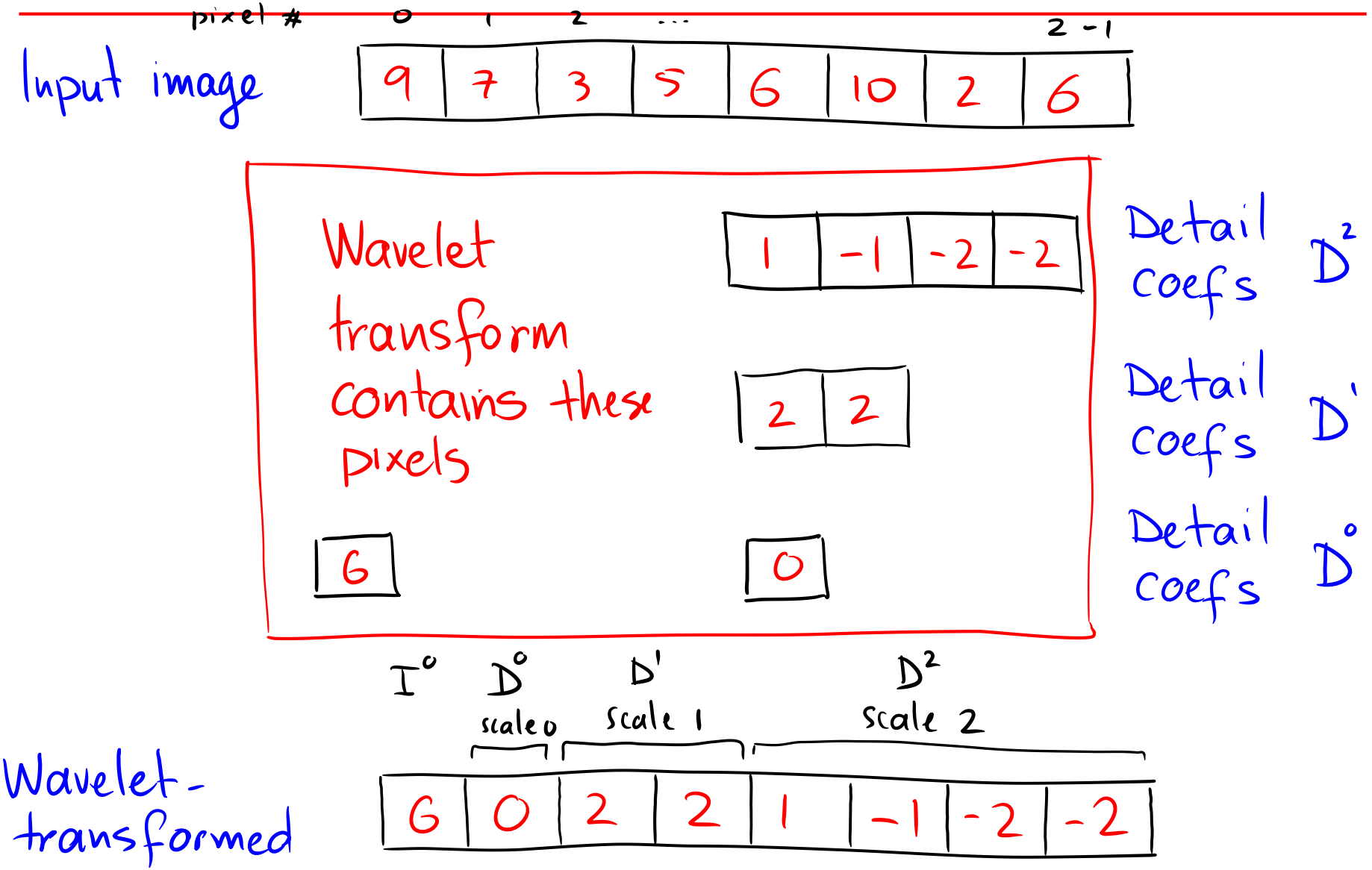


$$I_i^j = \frac{1}{2}\left(I_{2i}^{j+1} + I_{2i+1}^{j+1} \right)$$

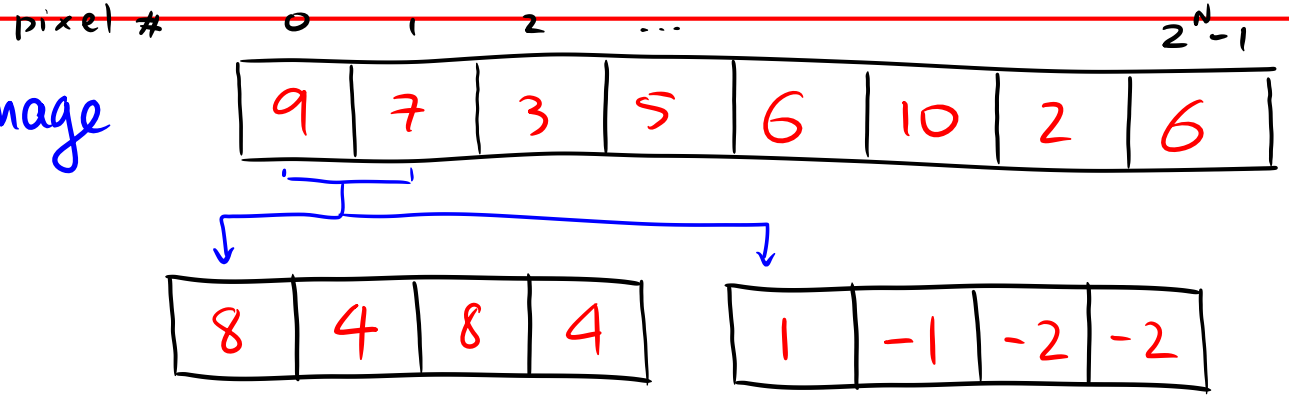
$$D^j = \frac{1}{2}\left(I_{2i}^{j+1} - I_{2i+1}^{j+1} \right)$$

j-th level of "pyramid" contains 2^j pixels

1D Haar wavelet transform: recursive definition



1D Haar wavelet transform: matrix formulation

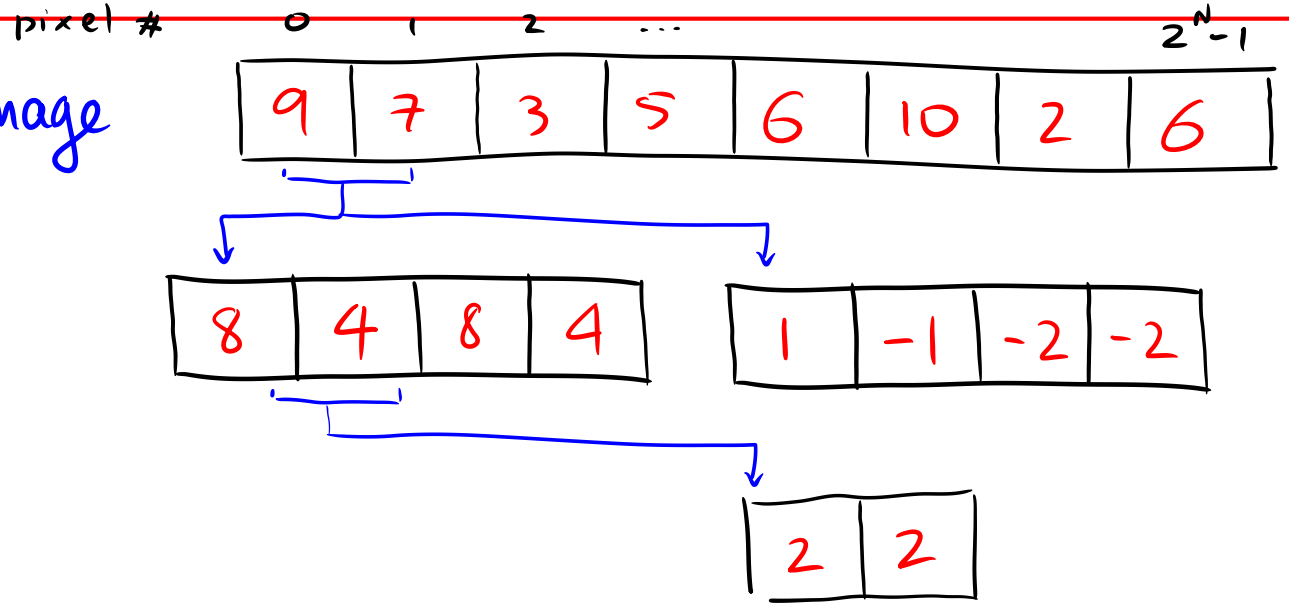


$$\begin{bmatrix} I^0 \\ D^0 \\ D^1 \\ D^2 \end{bmatrix} = \begin{bmatrix} 6 \\ 0 \\ 2 \\ 2 \\ 1 \\ -1 \\ -2 \\ -2 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} 9 \\ 7 \\ 3 \\ 5 \\ 6 \\ 10 \\ 2 \\ 6 \end{bmatrix}$$

Wavelet transformed image

Original image

1D Haar wavelet transform: matrix formulation

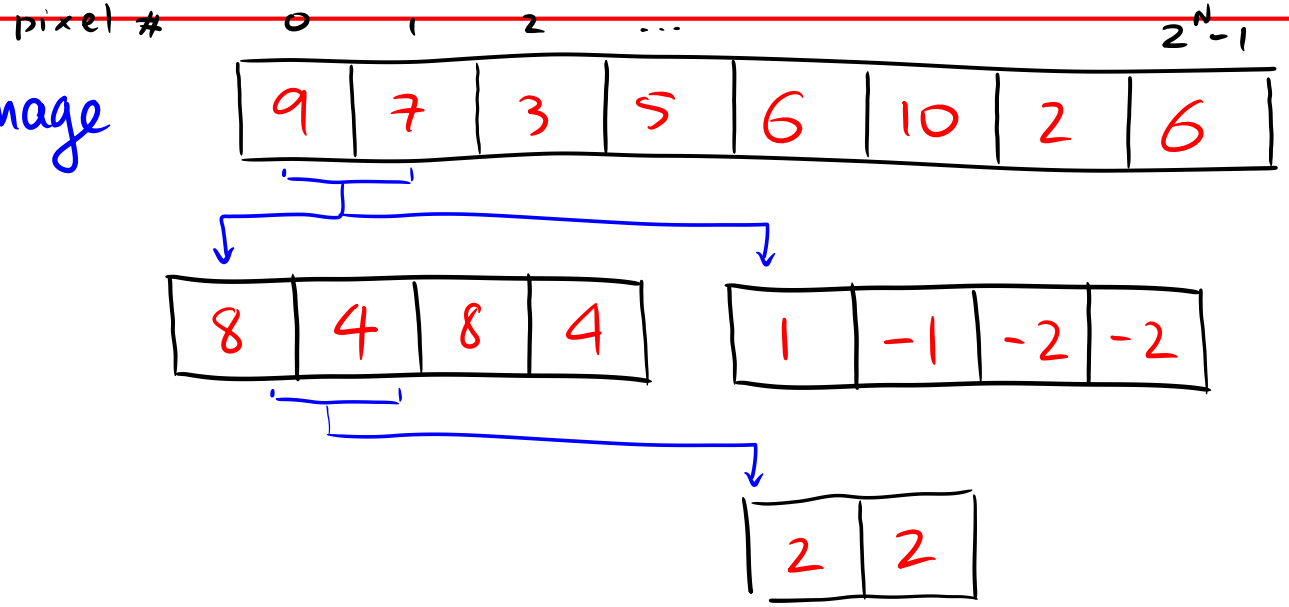


Wavelet transformed image

$$\begin{matrix} I^0 \\ D^0 \\ \\ D' \\ \\ \\ D^2 \end{matrix} \begin{bmatrix} 6 \\ 0 \\ 2 \\ 2 \\ 1 \\ -1 \\ -2 \\ -2 \end{bmatrix} = \begin{bmatrix} & & & & & & & \\ & & & & & & & \\ 1/4 & 1 & 1 & -1 & -1 & 0 & 0 & 0 & 0 \\ & 0 & 0 & 0 & 0 & 1 & 1 & -1 & -1 \\ & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 1/2 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} 9 \\ 7 \\ 3 \\ 5 \\ 6 \\ 10 \\ 2 \\ 6 \end{bmatrix}$$

Original image

1D Haar wavelet transform: matrix formulation



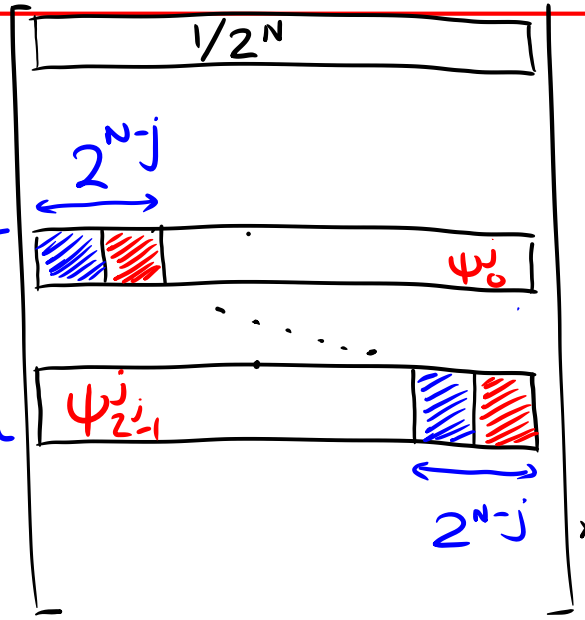
Wavelet transformed image

$$\begin{matrix} I^0 \\ D^0 \\ \\ D' \\ \\ \\ \\ D^2 \end{matrix} \begin{bmatrix} 6 \\ 0 \\ 2 \\ 2 \\ 1 \\ -1 \\ -2 \\ -2 \end{bmatrix} = \begin{matrix} 1/8 \\ 1/8 \\ 1/4 \\ \\ \\ 1/2 \\ \\ \end{matrix} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & 1 & -1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & -1 & -1 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} 9 \\ 7 \\ 3 \\ 5 \\ 6 \\ 10 \\ 2 \\ 6 \end{bmatrix}$$

Original image

the 1D Haar wavelet transform matrix

- Matrix contains $N-1$ scales
- Scale j represented by 2^j rows $\psi_0^j, \dots, \psi_{2^j-1}^j$

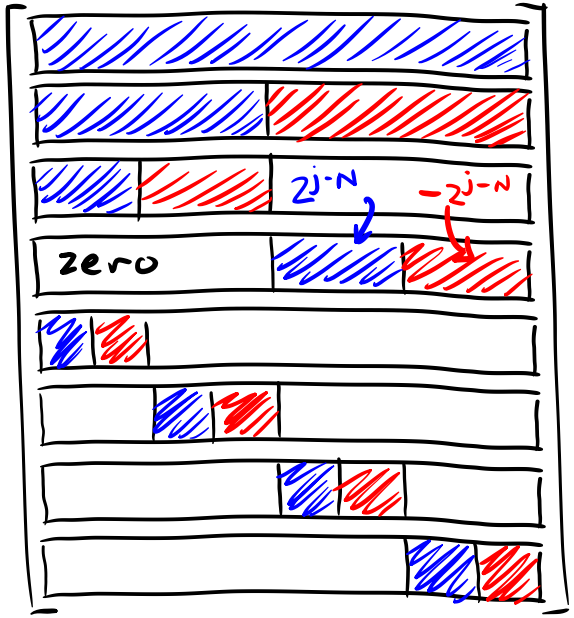


- Row ψ_j^i has $\frac{2^N}{2^j} = 2^{N-j}$ non-zero pixels
- They are pixels $x = i\,2^{N-j}, \dots, (i+1)2^{N-j}-1$ with $|\psi_j^i(x)| = \frac{1}{2^{N-j}}$

Wavelet transformed image

I^0
D^0
D^1
D^2

$$\begin{bmatrix} 6 \\ 0 \\ 2 \\ 2 \\ 1 \\ -1 \\ -2 \\ -2 \end{bmatrix} =$$



$$\begin{bmatrix} 9 \\ 7 \\ 3 \\ 5 \\ 6 \\ 10 \\ 2 \\ 6 \end{bmatrix}$$

Original image