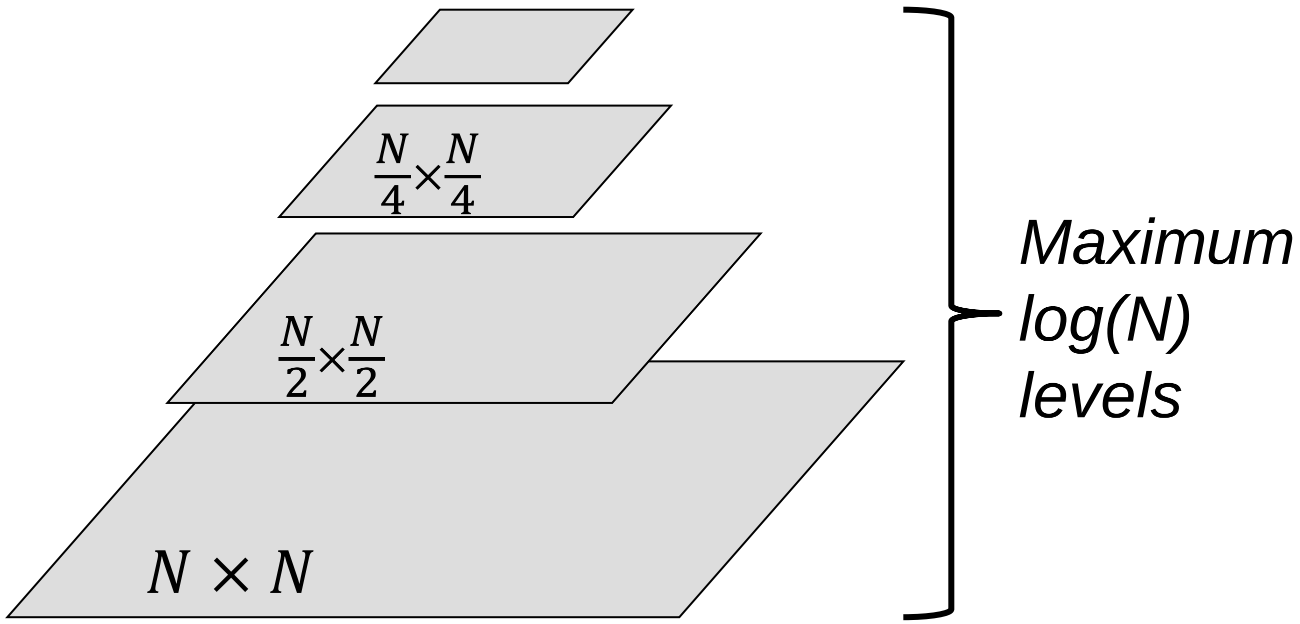
Image Pyramids



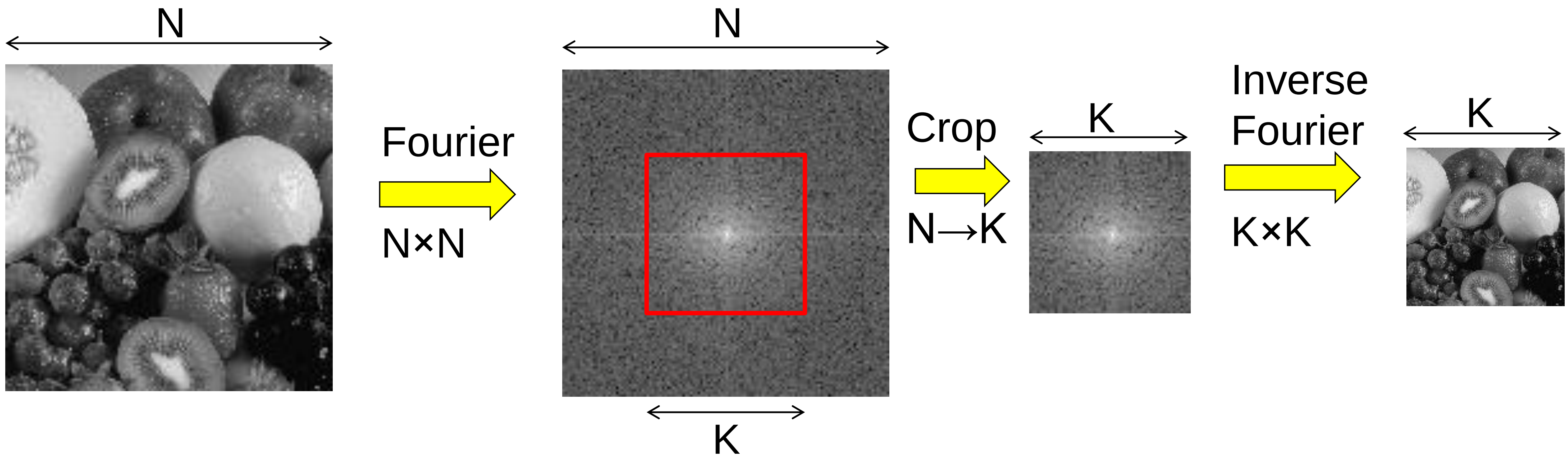
Number of pixels in this pyramid

$$N^2 + \frac{1}{4}N^2 + \frac{1}{16}N^2 + \cdots = 1\frac{1}{3}N^2$$

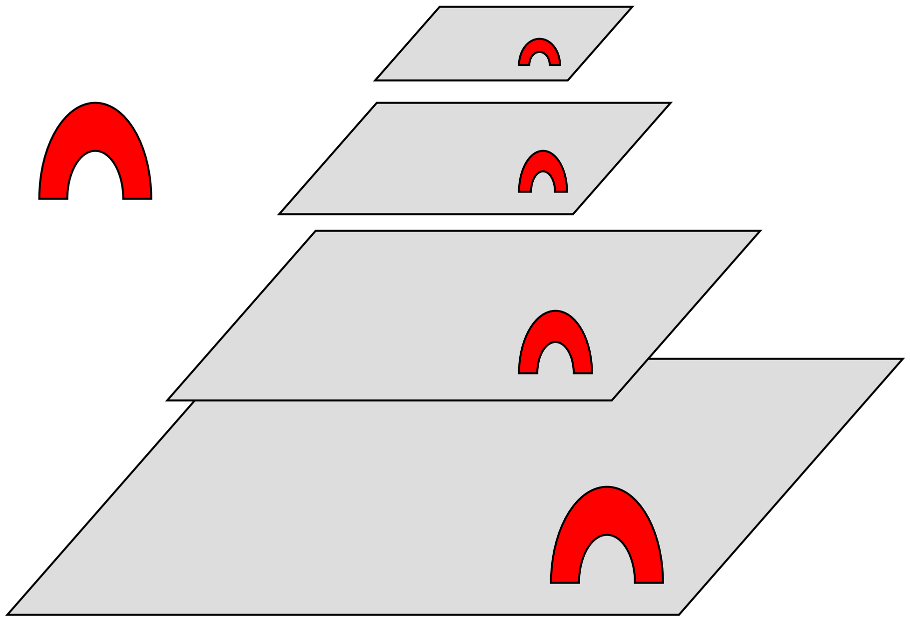
Image Resizing

- While we will only talk about resizing by ½, all scales are possible.
- Resizing by ½: Blur & Subsample every 2^{nd} pixel in every 2^{nd} row. E.g. - From 1024 x 1024 to 512 x 512
  - Convolution in image domain
- Arbitrary resizing: Use Fourier Transform
  - E.g. - From 1024 x 1024 to 712 x 712

Arbitrary Resizing with Fourier (N→K) (Reminder)



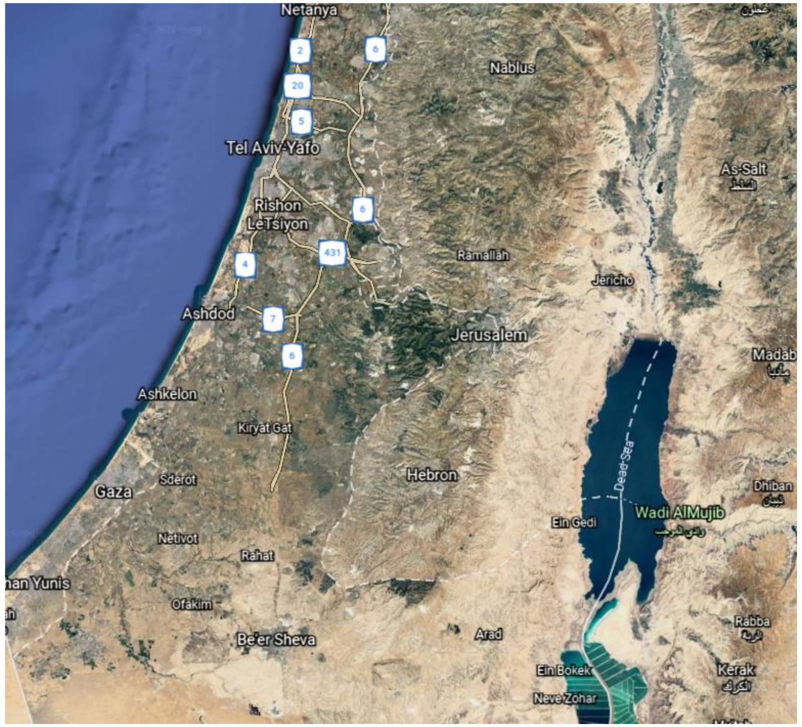
Use 1: Efficient Visual Search

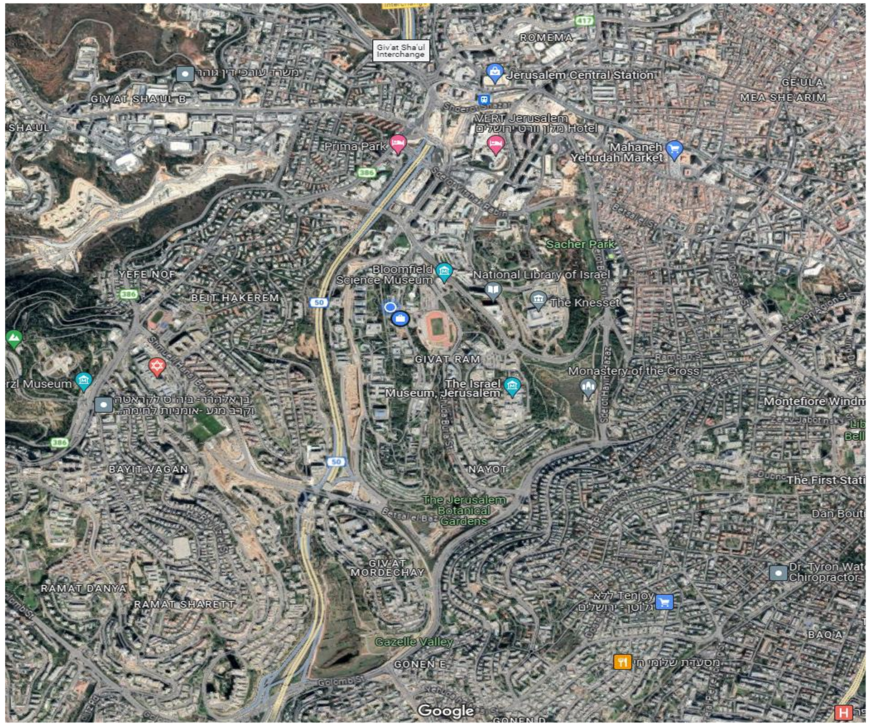


Search Area (pixels)	Pattern Size	Total Operations
$32^2(2^{10})$	$4^2\ (2^4)$	2^{14}
$64^2(2^{12})$	$8^2\ (2^6)$	2^{18}
$128^2(2^{14})$	$16^2\ (2^8)$	2^{22}
$256^2(2^{16})$	$32^2\ (2^{10})$	2^{26}

- Pyramids: Start the search in a small image
- Given an estimate from a lower resolution level, search area is small in higher resolution levels (e.g. ±1)
- Complexity at higher resolution: 9*pattern size

Search Example: Find this building in Google Earth (30M * 30M pixels)





More Applications for Pyramids

- Browsing in Image Databases: Multiple images or Videos
- Motion Computation; Stitching; More… (Later in Course)

Scale by 1/2

Scale by 1/5

Image Resizing

Reduce:

1. Blur (sometimes can be **decomposed**: Horizontal & Vertical)
   – E.g. Convolve with a 3×3 filter $(\frac{1}{4}, \frac{1}{2}, \frac{1}{4}) * (\frac{1}{4}, \frac{1}{2}, \frac{1}{4})^T$

   $$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

   or a 5×5 filter $\frac{1}{256}(1, 4, 6, 4, 1) * (1, 4, 6, 4, 1)^T$ … or larger

   $$\frac{1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

2. Sub-sample
   – Select only every 2nd pixel in every 2nd row

Expand:

1. Zero Padding $(a_1, 0, a_2, 0, a_3, 0, \ldots)$
2. Blur
   - Note: Expand blur needs different normalizations due to zero padding!
   - Is zero padding followed by blur with $(\frac{1}{2}, 1, \frac{1}{2})$ OK?

Blur Kernels

Commonly Used – **Decomposable** Binomial Coefficients, odd length:

- Odd number of coefficients (have a center pixel)
- Sum of coefficients is normalized to 1
- Fast to compute:
  - Binomial - using shift and integer add; Decomposable: 2N instead of N^2
- Asymptotically similar to a Gaussian

1 2 1 / 4

1 4 6 4 1 / 16

1 6 15 20 15 6 1 / 64

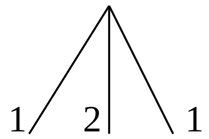
$(1\ 1) * \ldots^{2k} \ldots * (1\ 1) \quad / \ 2^{2k}$

Decomposition of Kernels

$$P * \frac{1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix} = P * \frac{1}{16}\begin{bmatrix} 1 \\ 4 \\ 6 \\ 4 \\ 1 \end{bmatrix} * \frac{1}{16}[1\ 4\ 6\ 4\ 1]$$

- Saving Computations (E.g. 5 x 5 kernel)
  - Naïve Computation: blur by 5x5 (25 multiplications)
  - If kernel can be decomposed to horizontal and vertical components
  - Blur columns first (5 multiplications), then blur rows
  - 10 multiplications instead of 25

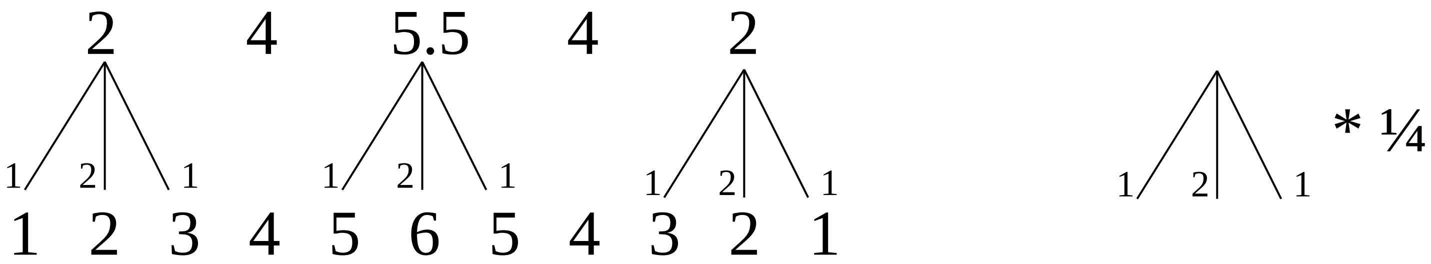
Reduce: Blur & Sub-sample



* ¼

1 2 3 4 5 6 5 4 3 2 1

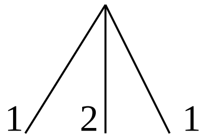
Reduce: Blur & Sub-sample



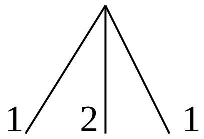
Expand: Zero-Pad & Blur

0   2   0   4   0 5.5 0 4   0   2   0

1   2   3   4   5   6   5   4   3   2   1

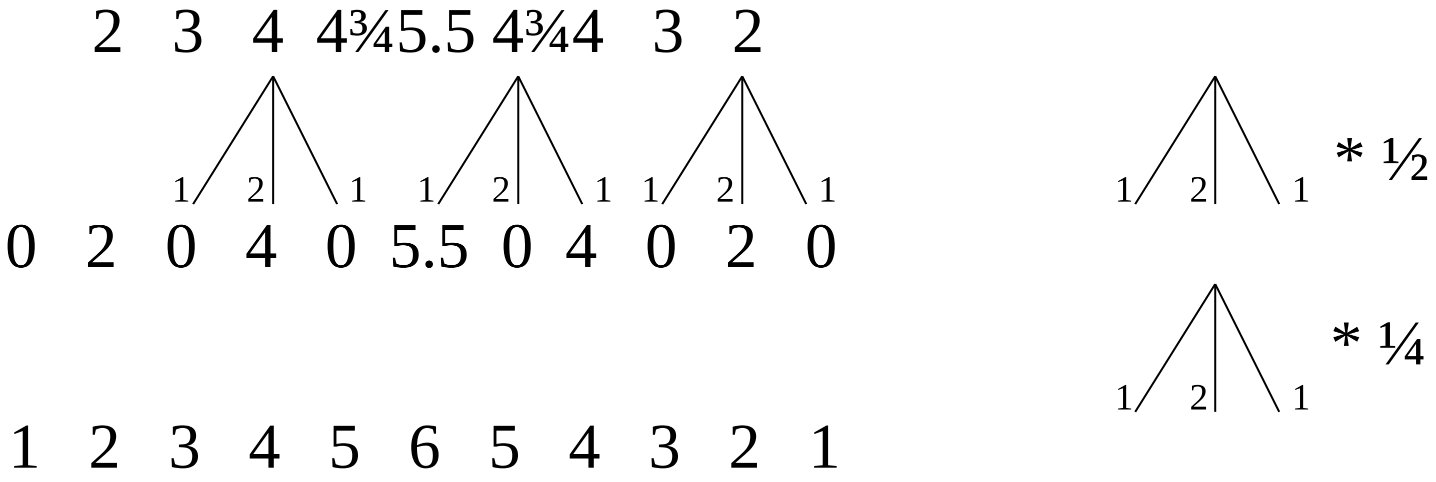


* ½



* ¼

Expand: Zero-Pad & Blur



Handling Image Boundaries

Never Cyclic…

$\boxed{}$ 2 3 4 4¾ 5.5 4¾ 4 3 2

1 2 1   1 2 1 * ½

$\boxed{?}$ 0 2 0 4 0 5.5 0 4 0 2 0

Reflect on last pixel	?=2
Zero padding	?=0
Duplicate last pixel.	?=0

Gaussian Pyramid

Iterated Reduction of the Image

G_n — $Reduce\{G_{n-1}\}$

G_3 — $Reduce\{G_2\}$

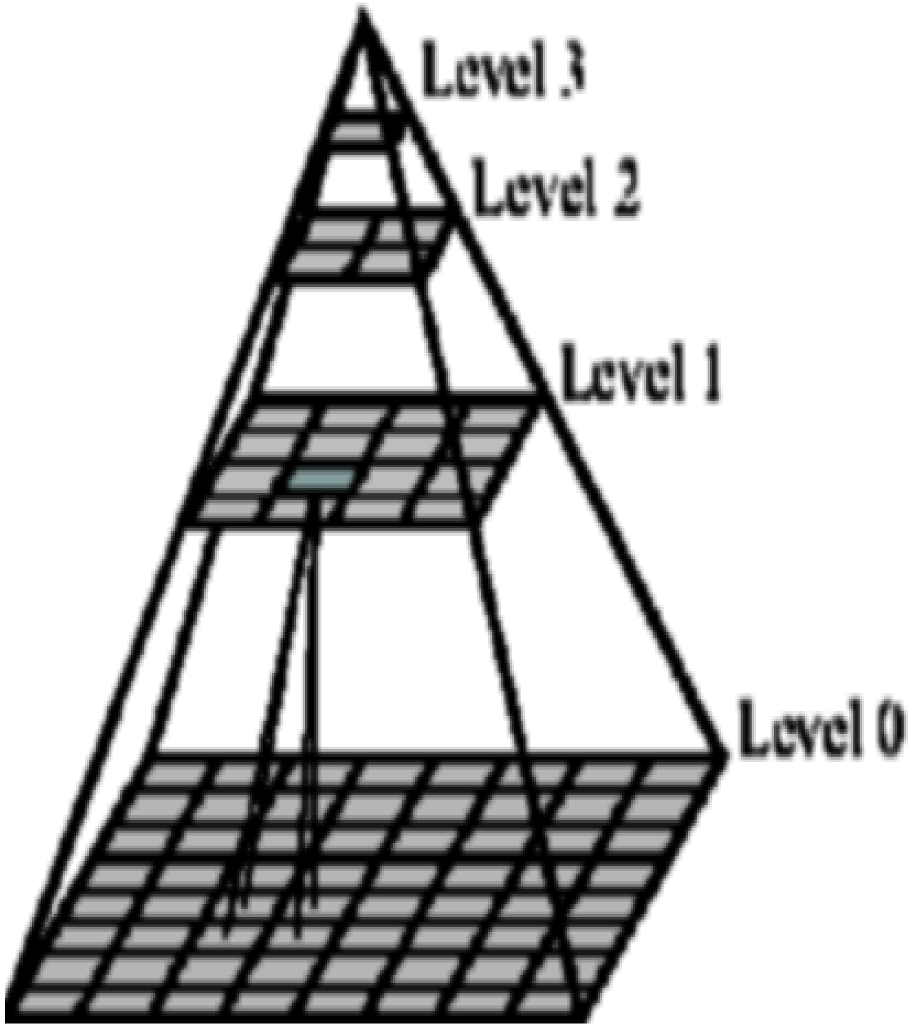
G_2 — $Reduce\{G_1\}$

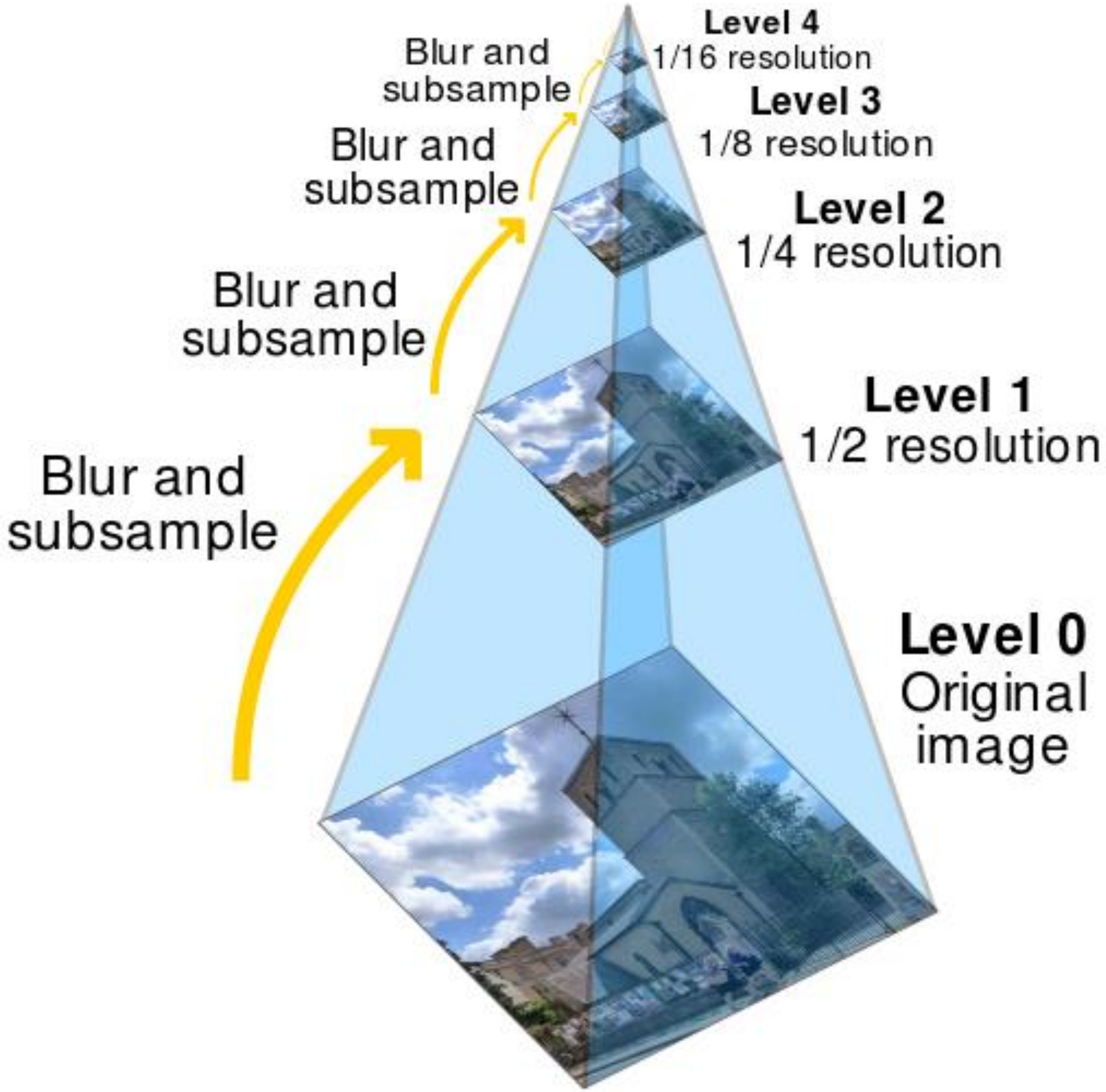
G_1 — $Reduce\{G_0\}$

G_0 — Original Image

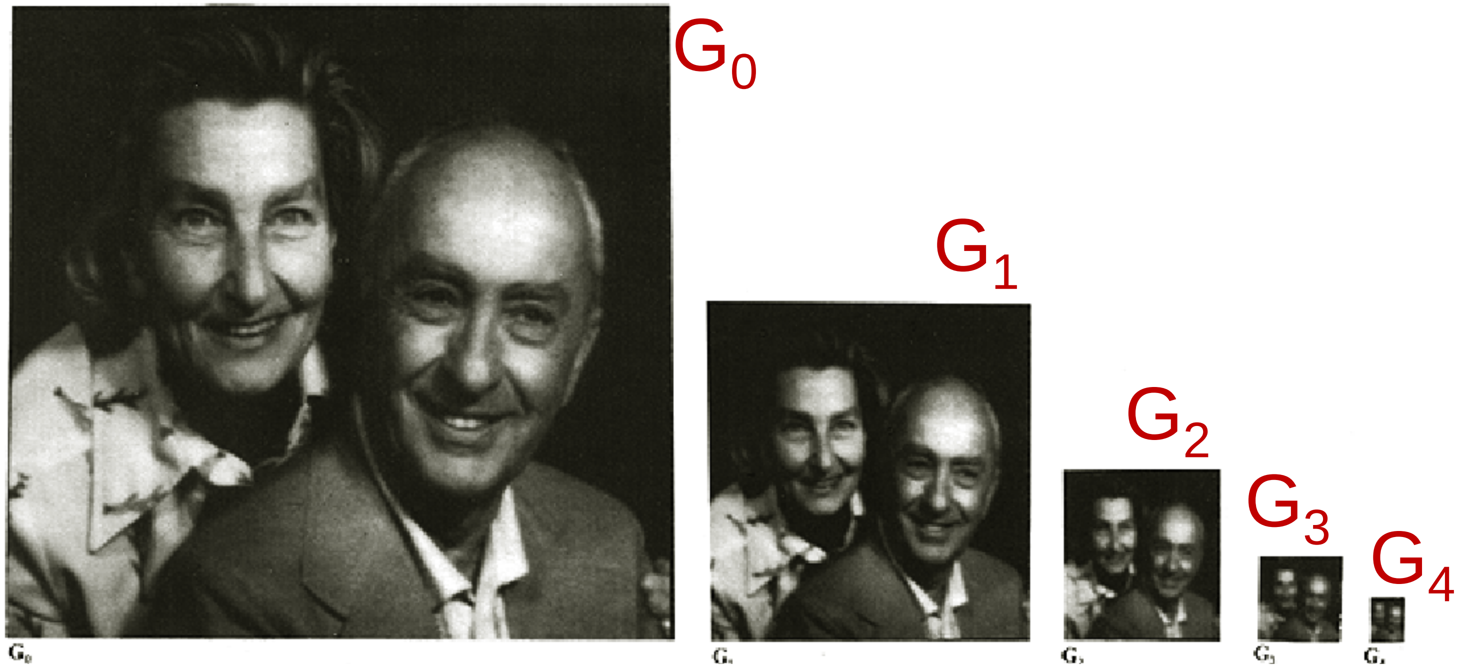
5-Level Gaussian Pyramid

(Wikipedia)

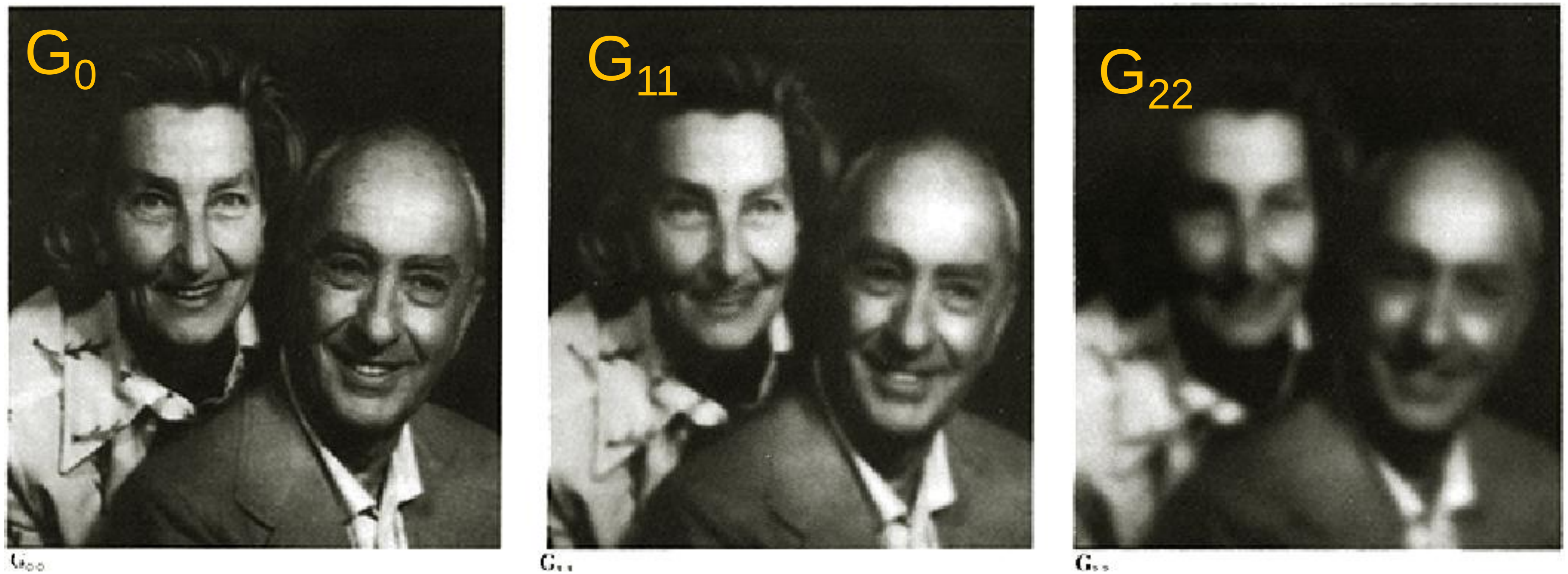




Gaussian Pyramid



Resize All Images to G_0 by repeated Expand



Laplacian Pyramid

Represents The Information Lost in Each Gaussian Level

	Gaussian	Laplacian		
Top	G_n	L_n	$L_4 = G_n$	
	G_3	L_3	$L_3 = G_3 - Expand\{G_4\}$	$G_4 = Reduce\{G_3\}$
	G_2	L_2	$L_2 = G_2 - Expand\{G_3\}$	
	G_1	L_1	$L_1 = G_1 - Expand\{G_2\}$	
Original	G_0	L_0	$L_0 = G_0 - Expand\{G_1\}$	

$$L_n + L_{n-1} = Expand\{L_n\} + L_{n-1} =$$

$$= Expand\{G_n\} + (G_{n-1} - Expand\{G_n\}) = G_{n-1}$$

$$\sum_{i=k}^{n} L_i = G_k$$

$$\sum_{i=0}^{n} L_i = G_0$$

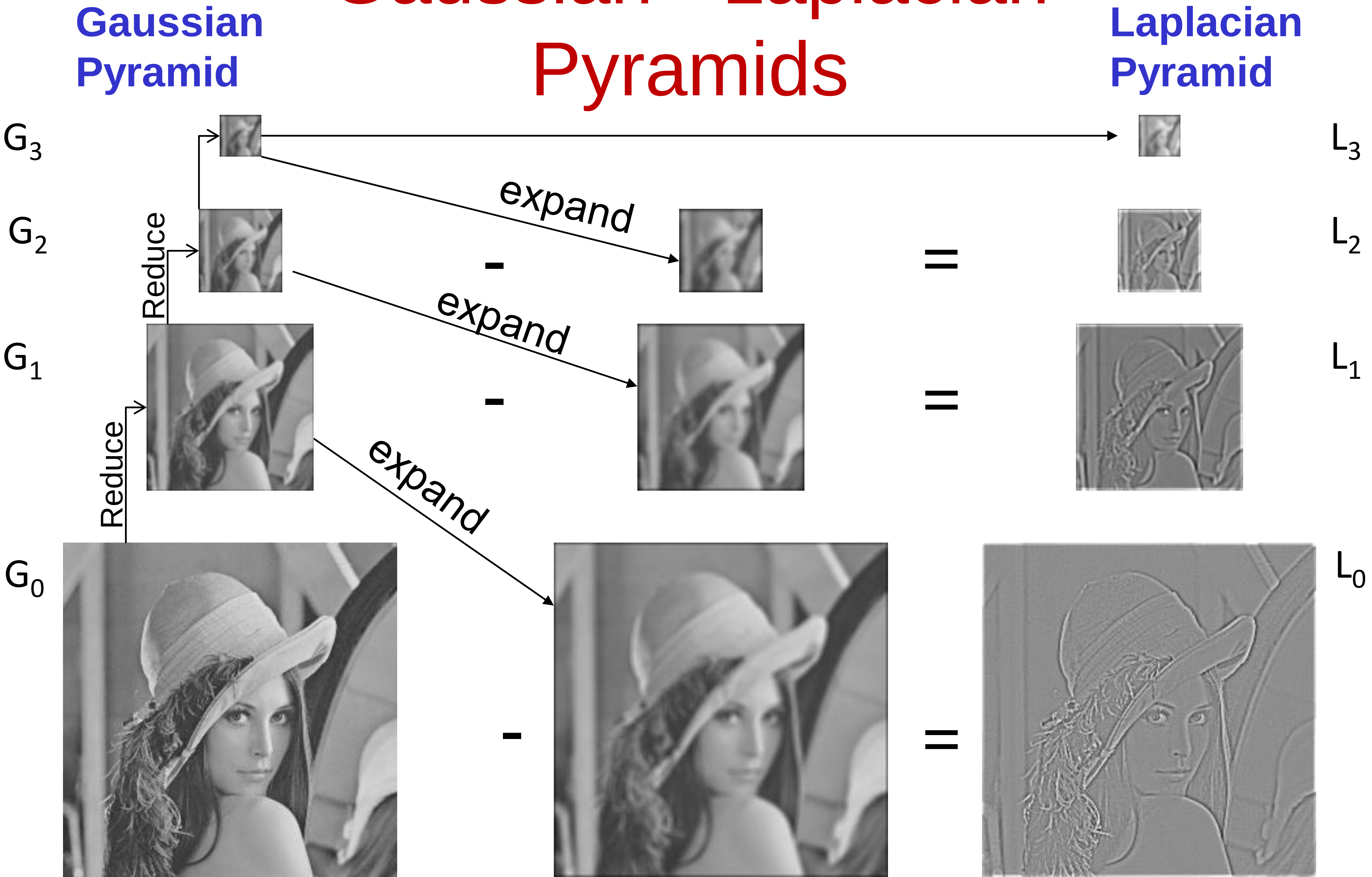
G_0

G_1

$Expand\{G_1\}$

$L_0 = G_0 - Expand\{G_1\}$

Gaussian - Laplacian Pyramids



Pyramid Compression (Burt, Adelson)

- Build a Laplacian Pyramid
- Quantize pyramid values to 3-5 values
  - Optimal Quantization (Future Lecture…)
- Compress using Entropy Compression
  - (Huffman, Lempel-Ziv) (Future Lecture…)
- Reconstruct normally

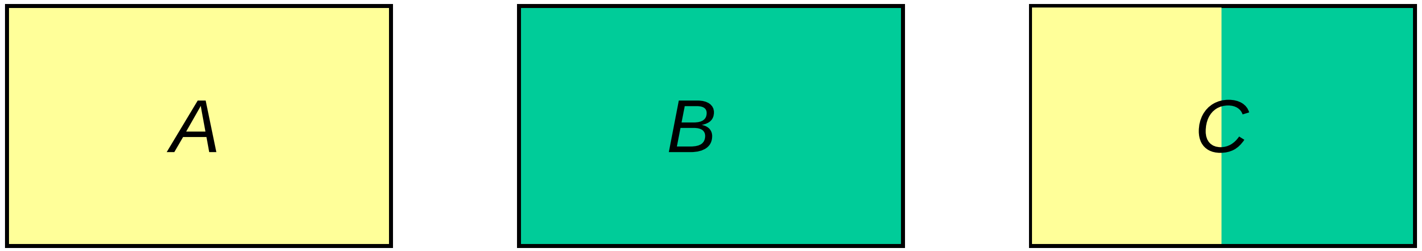
Pyramid Compression (Burt, Adelson)

65K bytes 8 bits/pixel	8K bytes 1 bits/pixel	4K bytes 0.5 bits/pixel

(a) (b) (c)

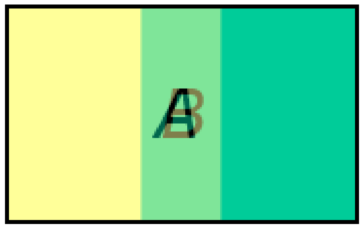
Fig. 5. *Pyramid data compression. The original image represented at 8 bits per-pixel is shown in (a). The node values of tbe Laplacian pyramid representation of this image were quantitized to obtain effective data rates of 1 b/p and 1/2 b/p. Reconstructed images (b) and (c) show relatively little degradation.*

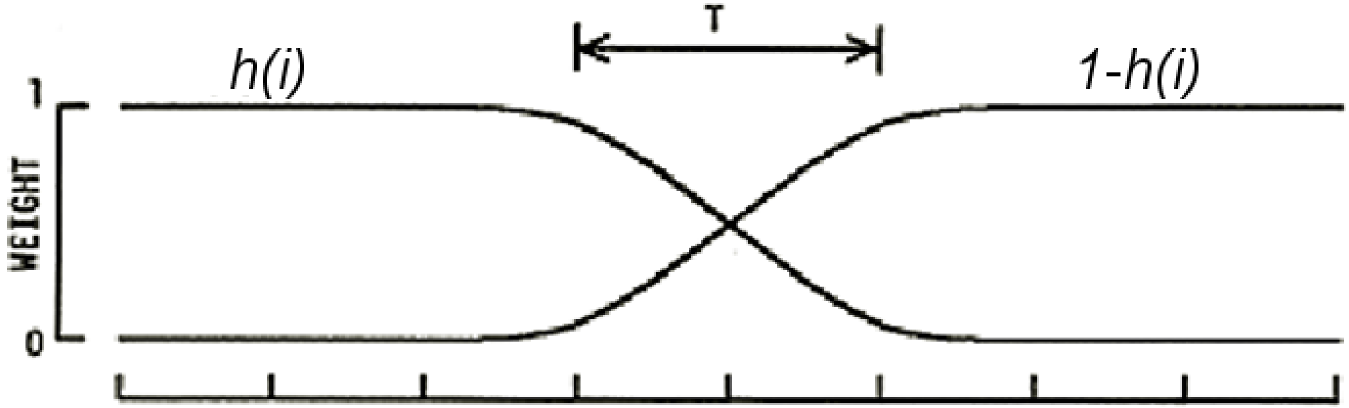
Picture Merging with Spline

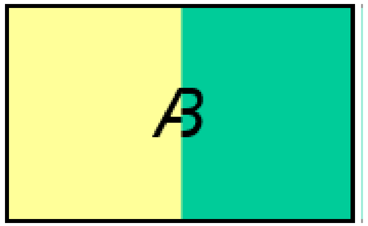


For every row y:

$$C(x,y) = h(x)\,A(x,y) + (1-h(x))\,B(x,y)$$







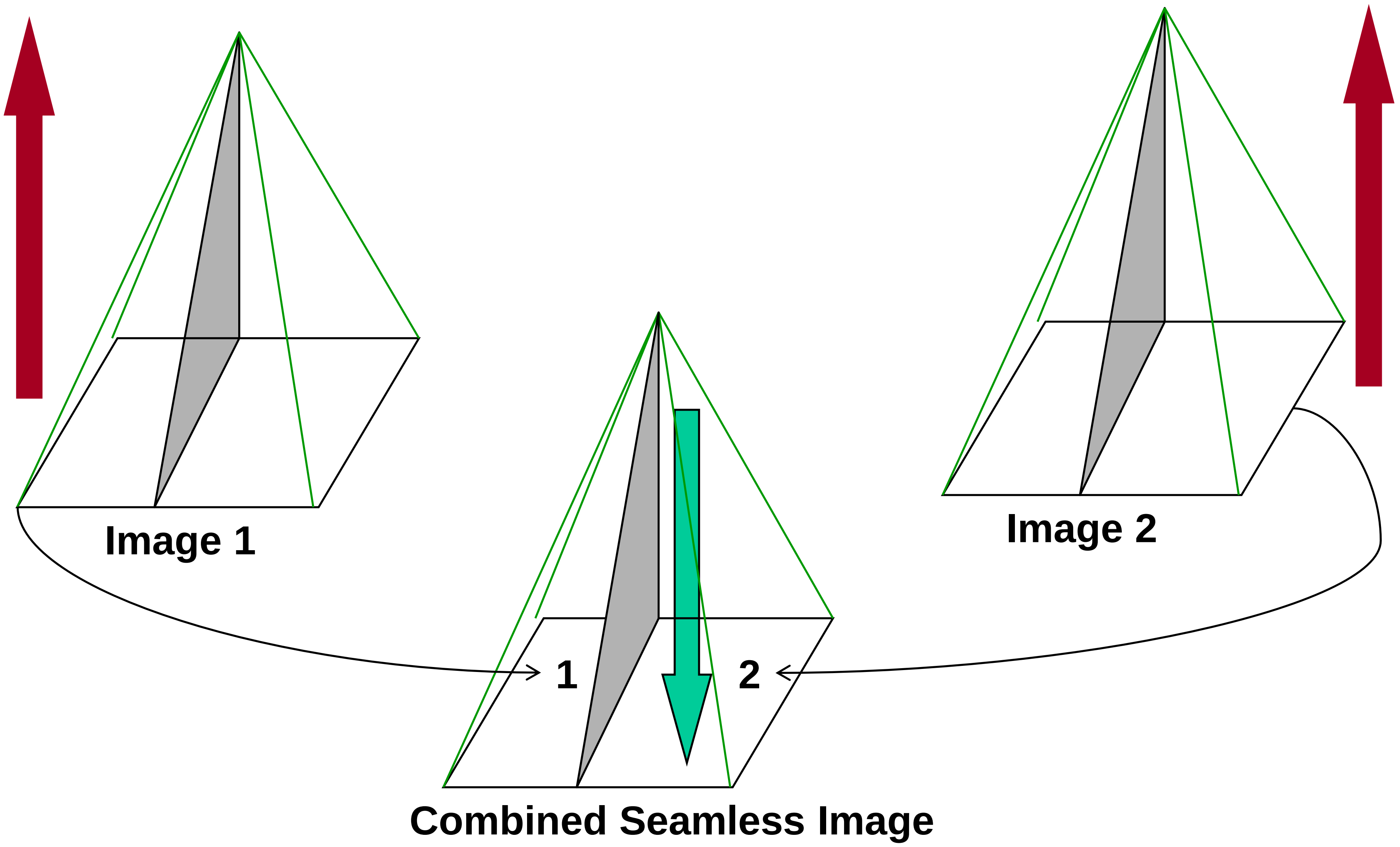
Multiresolution Pyramid Spline

- Given two images *A* and *B* to be splined in middle
- Construct Laplacian Pyramid L_a and L_b
- Create a third Laplacian Pyramid L_c where for <u>each</u> <u>level *k*</u>:

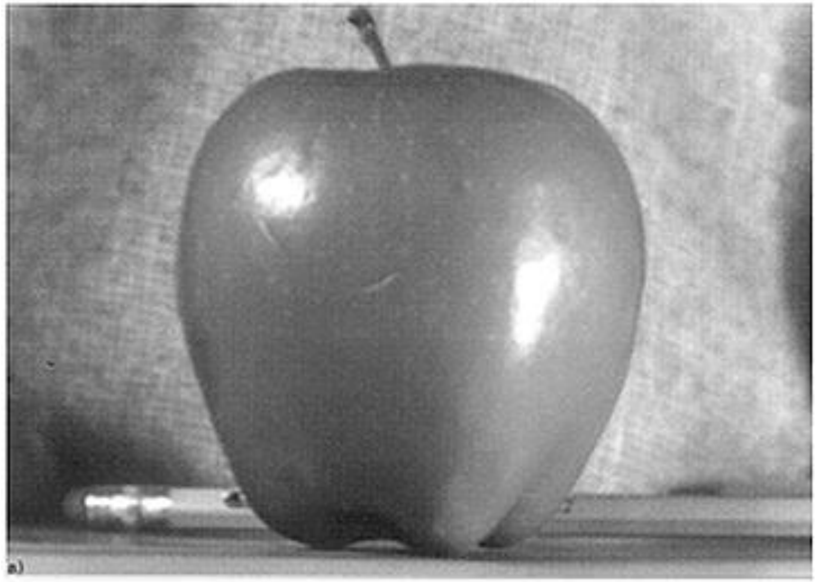
$$L_c(i,j) = \begin{cases} L_a(i,j) & if\ \ i < width/2 \\ \dfrac{L_a(i,j) + L_b(i,j)}{2} & if\ \ i = width/2 \\ L_b(i,j) & if\ \ i > width/2 \end{cases}$$

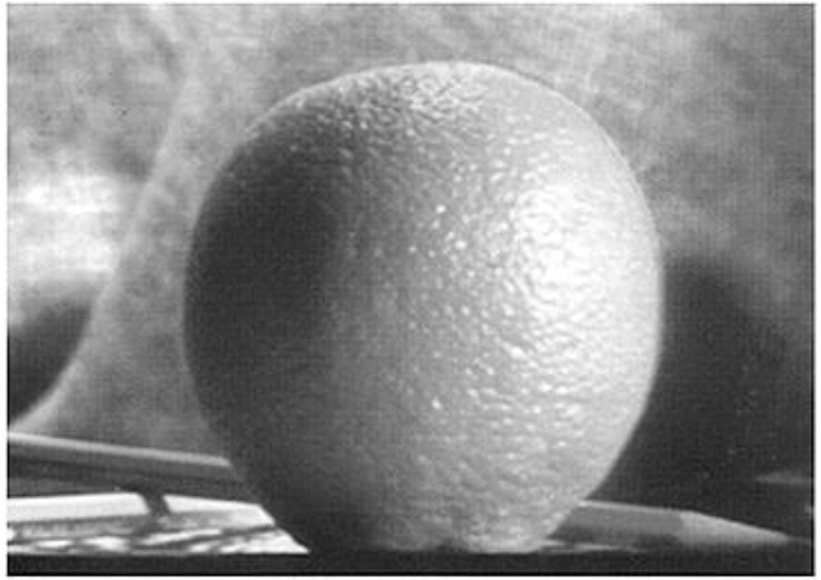
- Sum all levels in L_c to get the blended image

Image Merging with Laplacian Pyramids

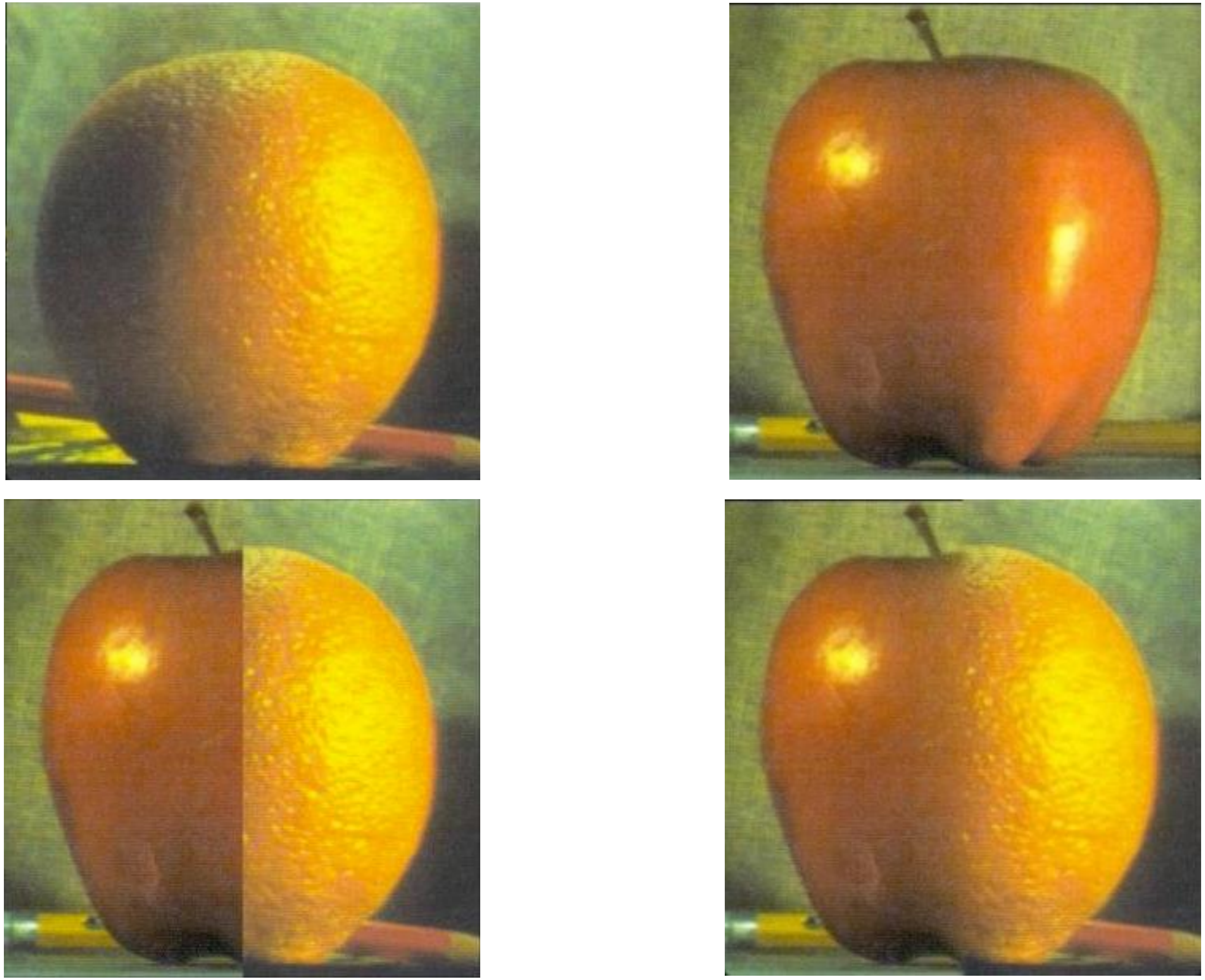


Pyramid Blending Example 1





Pyramid Blending Example 1



Pyramid Blending Arbitrary Shape

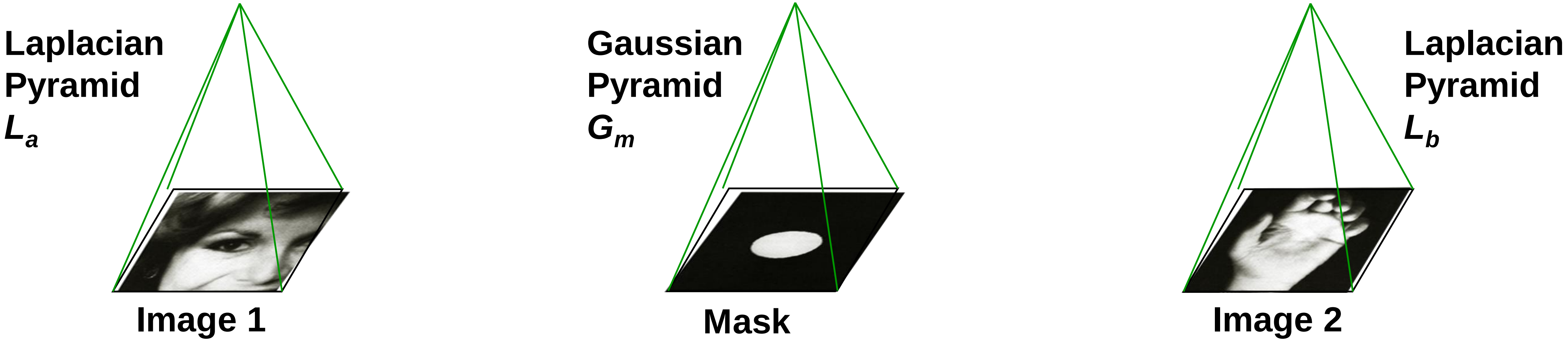
- Given two images *A* and *B*, and a binary mask *M*
- Construct Laplacian Pyramids L_a and L_b
- Construct a Gaussian Pyramid from mask M - G_m
- Create a third Laplacian Pyramid L_c where for each level *k*

$$L_c(i,j) = G_m(i,j)L_a(i,j) + (1 - G_m(i,j))L_b(i,j)$$

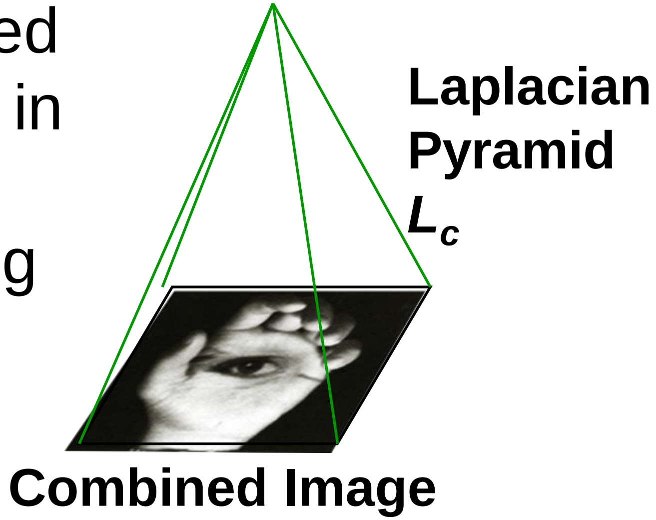
- Sum all levels L_c in to get the blended image

Pyramid Blending Example 2

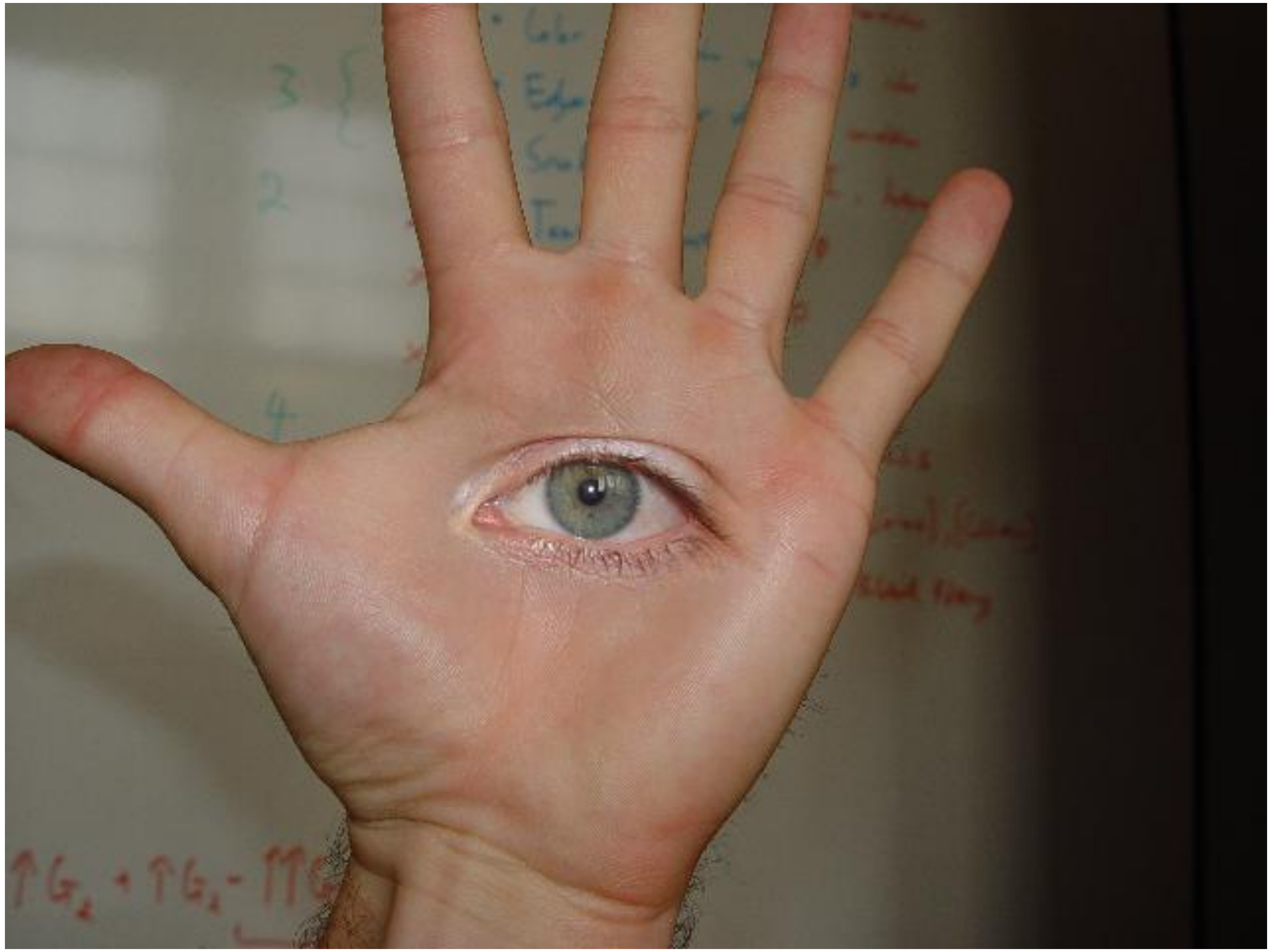
Pyramid Blending – Arbitrary Shape



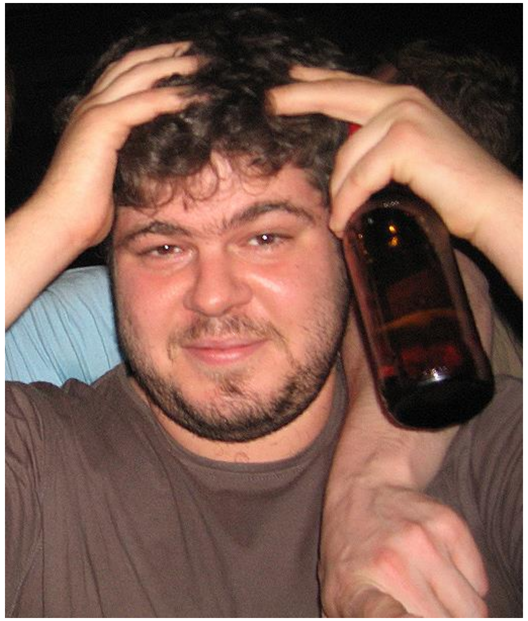
Each pixel *(i,j)* in each level in the Laplacian Pyramid L_c is created By averaging the corresponding pixels (same level and location) in L_a and L_b using the corresponding weights in G_m.
After L_c is completed, the combined image is created by summing all its levels.

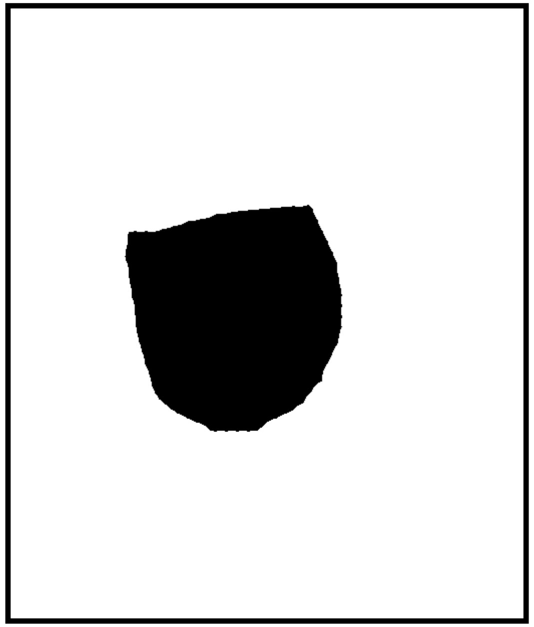


$$L_c(i,j) = G_m(i,j)L_a(i,j) + (1 - G_m(i,j))L_b(i,j)$$



Pyramid Blending Example





Pyramid Blending Example

Pyramid Blending Example