Lecture 7

Spatial Pyramids

6.869/6.819 Advances in Computer Vision

Spring 2022

Bill Freeman, Phillip Isola

Today’s lecture

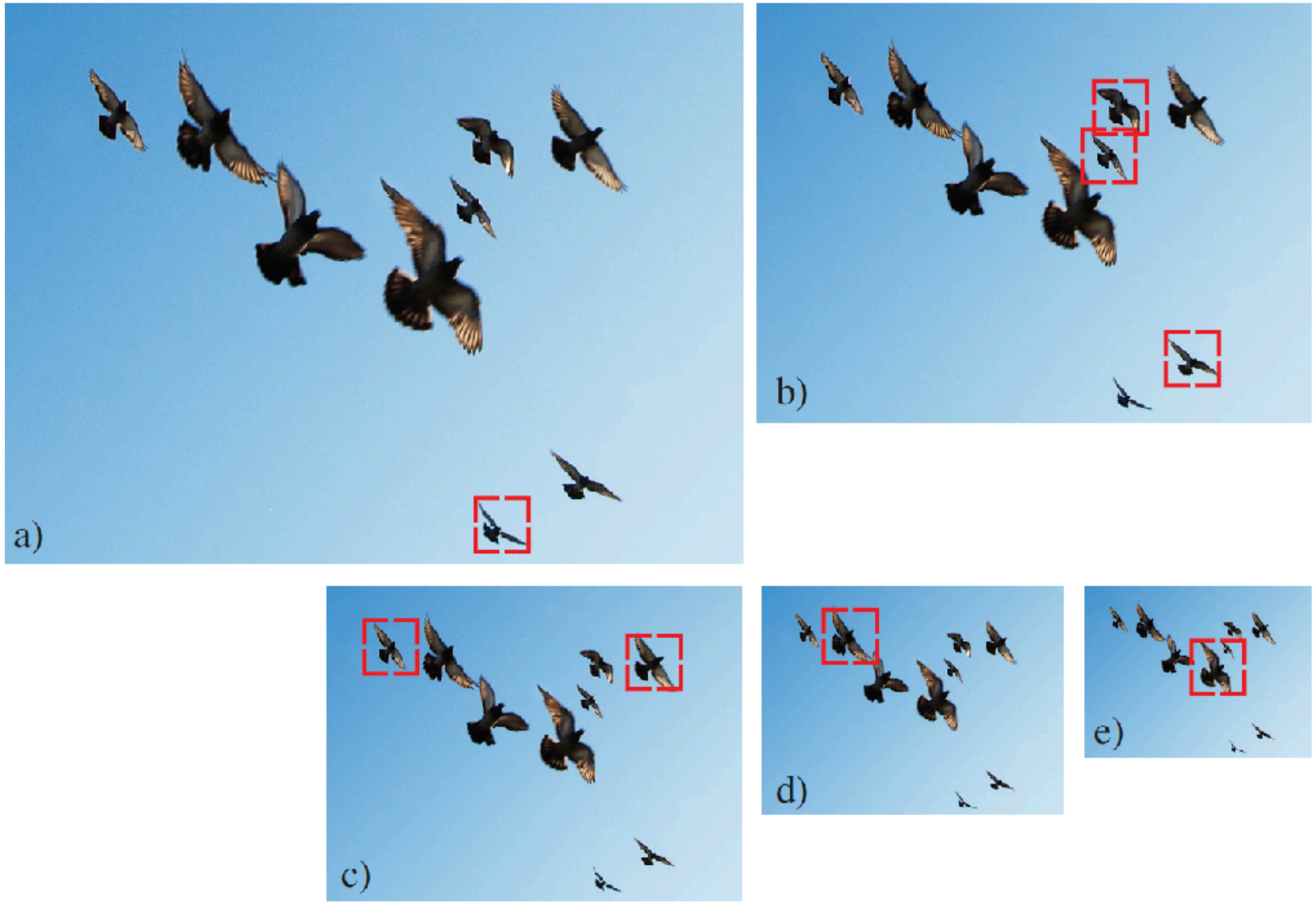
Problem set 2 due today
Problem set 3 released today
Reading: course notes, chapter 20

- Gaussian pyramid
  - application: recognition
  - neural network algorithms
- Laplacian pyramid
  - application in image blending
  - neural network algorithms
- Steerable pyramid
  - application in texture synthesis
  - neural network algorithms

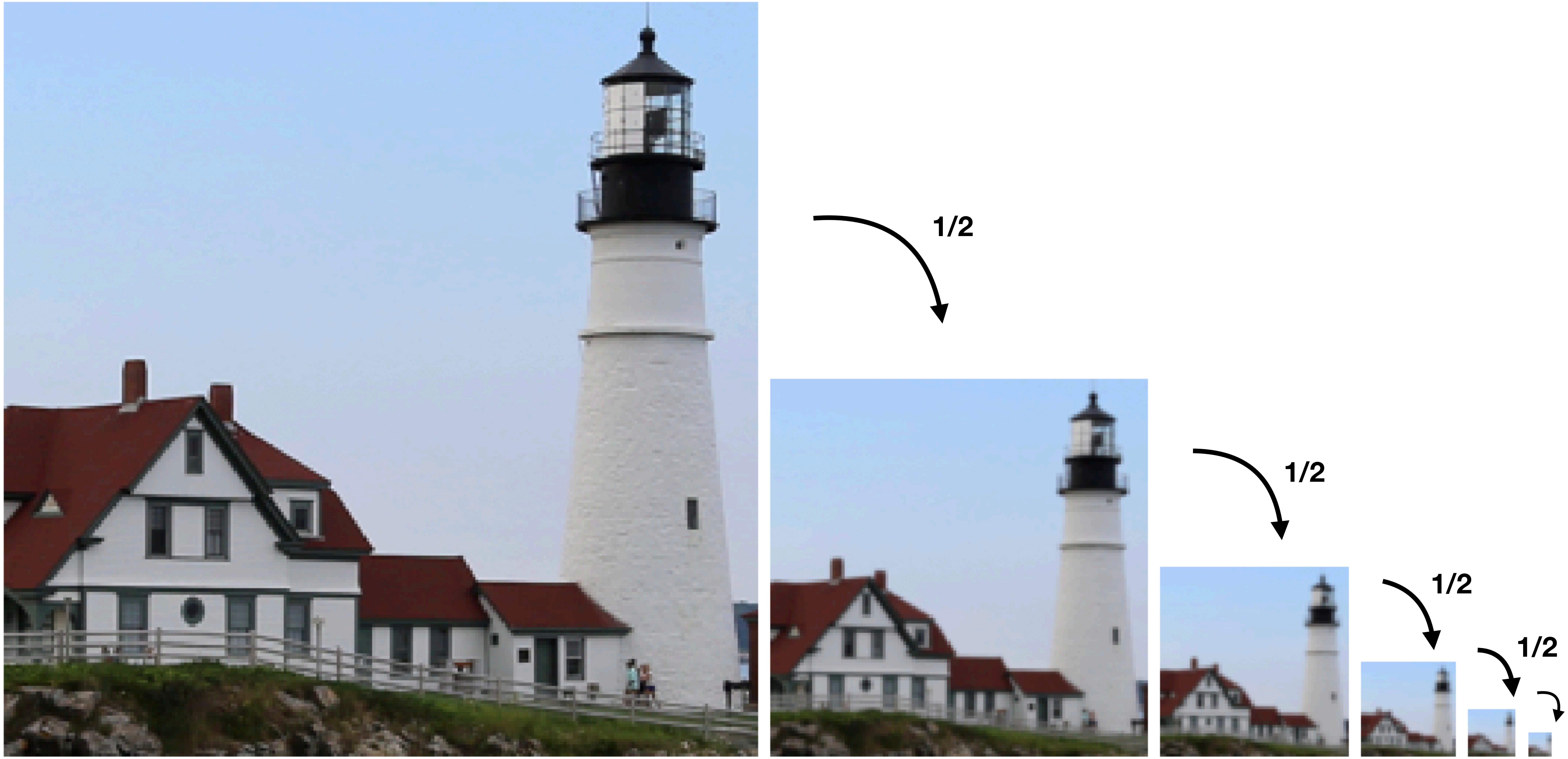
We need translation invariance

We need translation and scale invariance

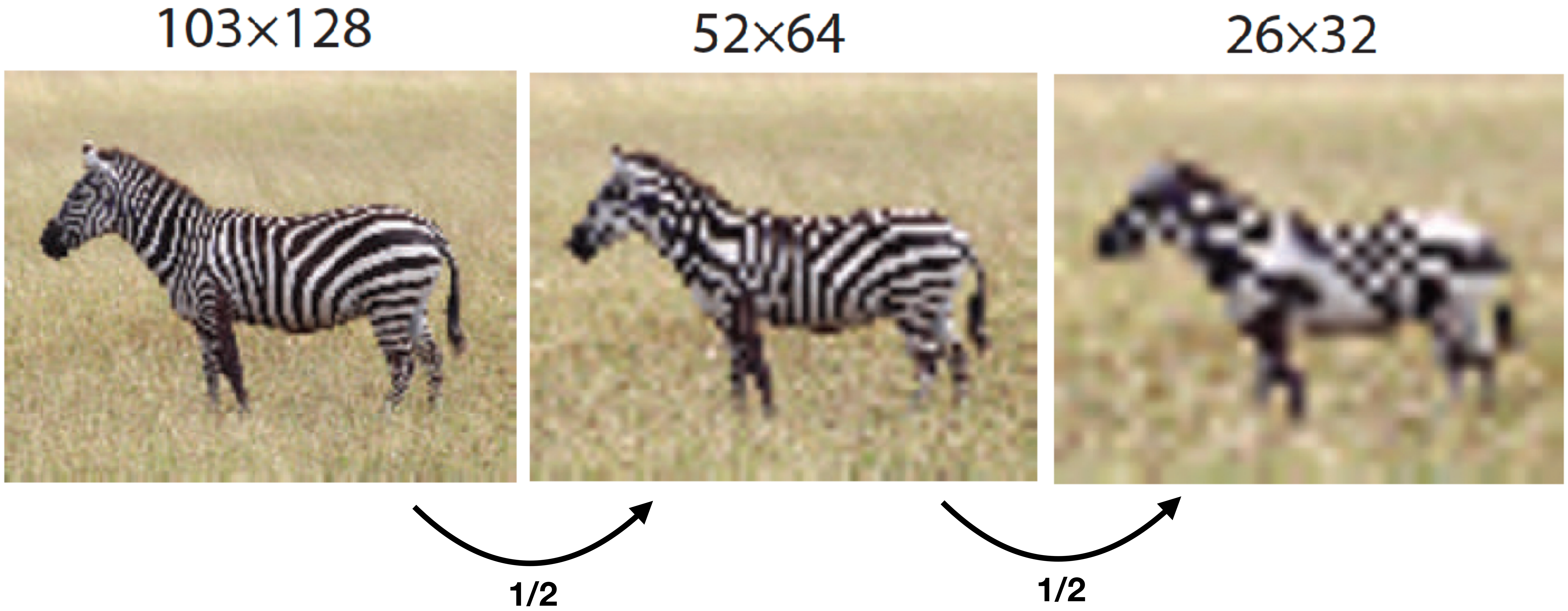
Image pyramids



Gaussian Pyramid



Subsampling and aliasing



Subsampling without blurring

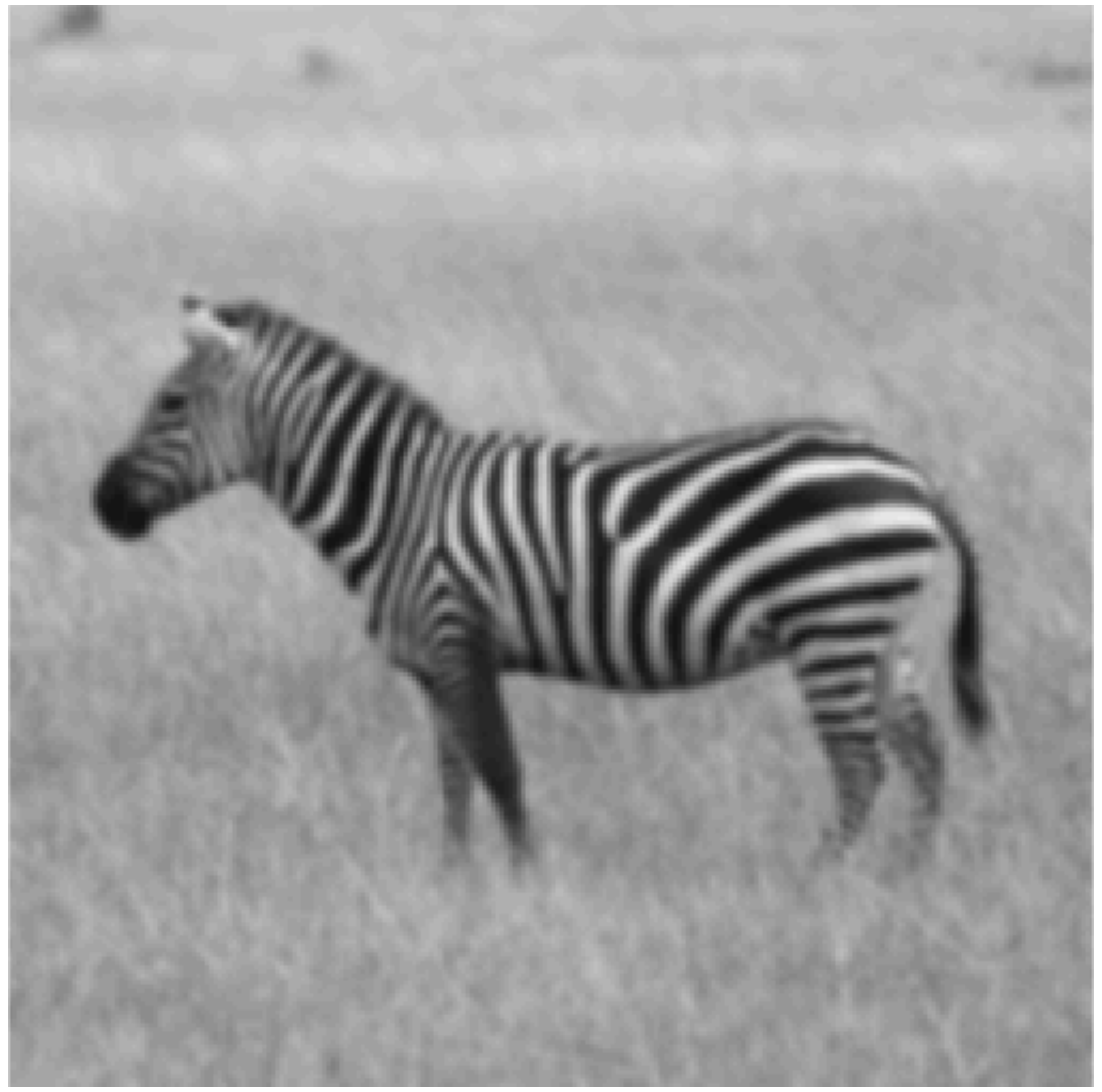
The Gaussian pyramid

For each level

1. Blur input image with a Gaussian (actually, binomial) low-pass filter

[1, 4, 6, 4, 1]

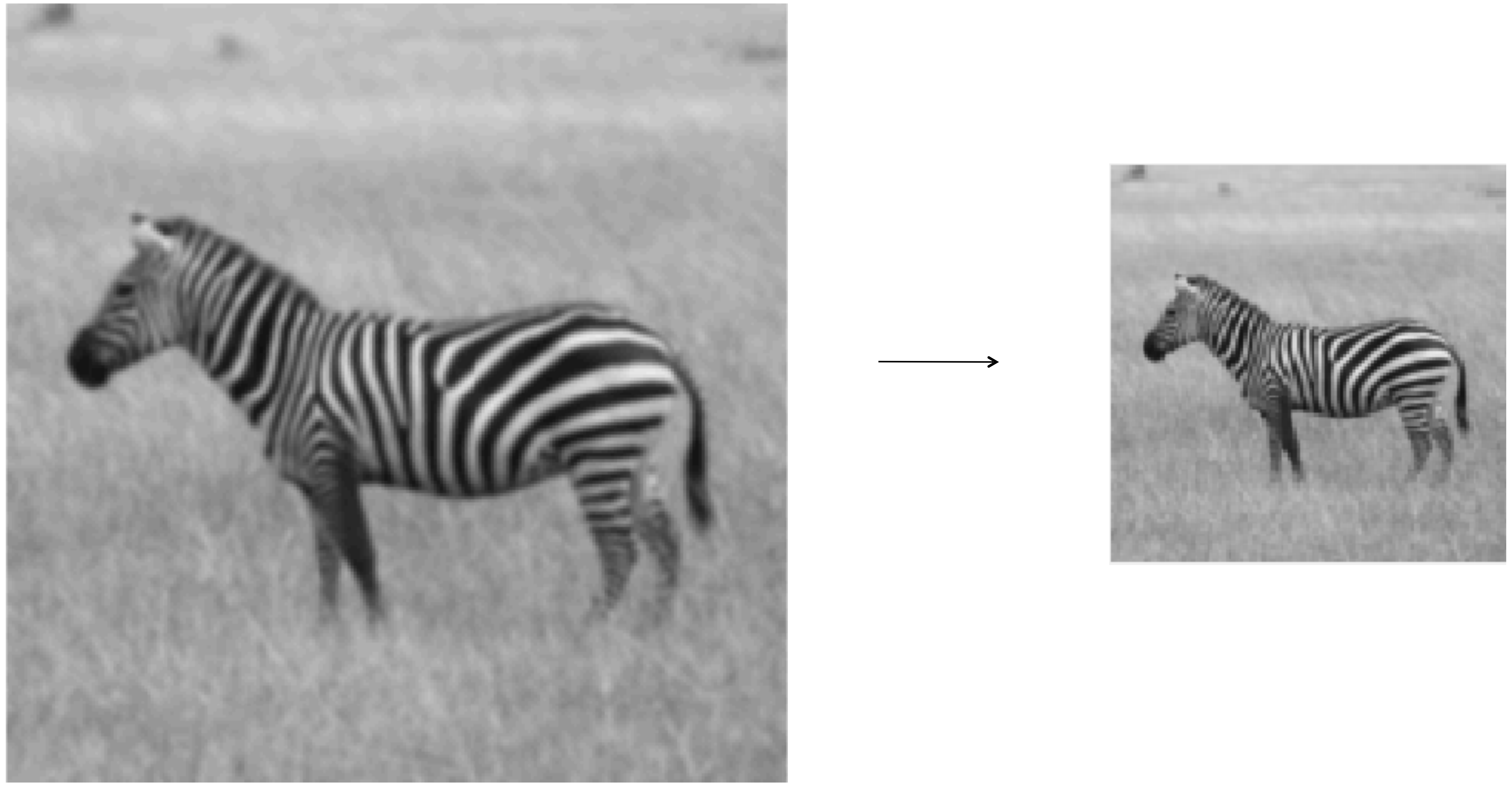
$$\bigcirc [1, 4, 6, 4, 1] \bigcirc \begin{bmatrix}1\\4\\6\\4\\1\end{bmatrix} \longrightarrow$$



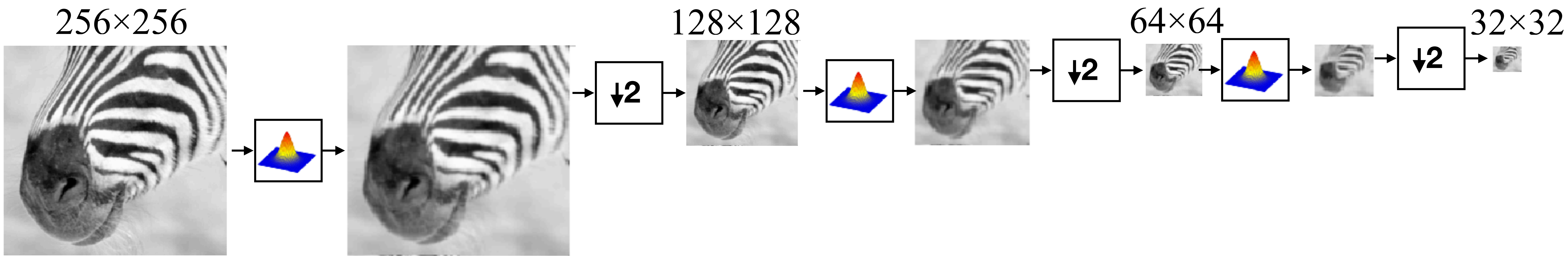
The Gaussian pyramid

For each level

1. Blur input image with a Gaussian filter
2. Downsample image



The Gaussian pyramid



The Gaussian pyramid

512×512

(original image)

256×256

128×128

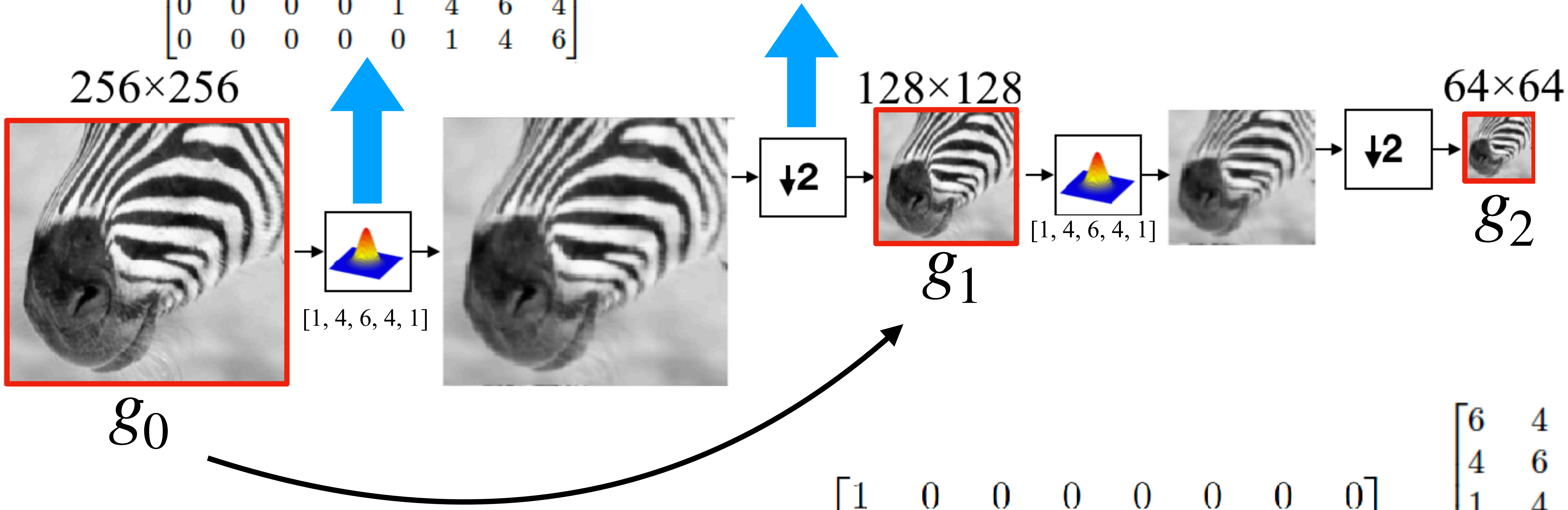
64×64

32×32

The Gaussian pyramid operators, in matrix form

$$\frac{1}{16}\begin{bmatrix}6&4&1&0&0&0&0&0\\4&6&4&1&0&0&0&0\\1&4&6&4&1&0&0&0\\0&1&4&6&4&1&0&0\\0&0&1&4&6&4&1&0\\0&0&0&1&4&6&4&1\\0&0&0&0&1&4&6&4\\0&0&0&0&0&1&4&6\end{bmatrix}$$

$$\begin{bmatrix}1&0&0&0&0&0&0&0\\0&0&1&0&0&0&0&0\\0&0&0&0&1&0&0&0\\0&0&0&0&0&0&1&0\end{bmatrix}$$



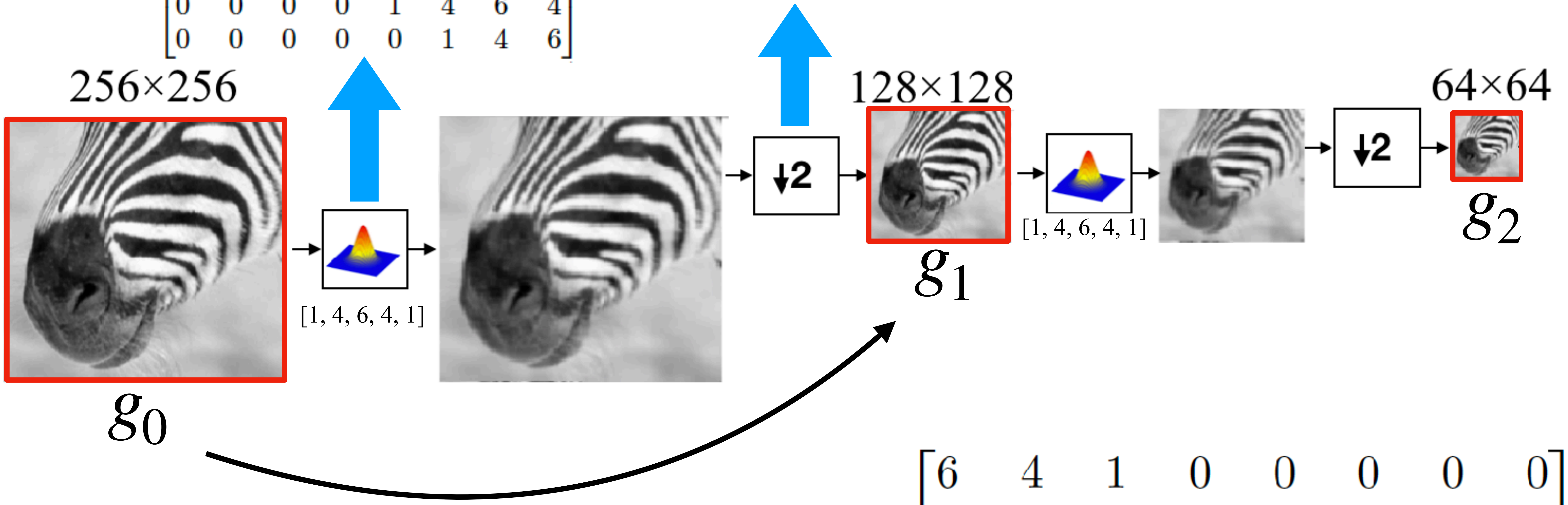
$$g_1 = G_0 g_0$$

$$G_0 = \begin{bmatrix}1&0&0&0&0&0&0&0\\0&0&1&0&0&0&0&0\\0&0&0&0&1&0&0&0\\0&0&0&0&0&0&1&0\end{bmatrix}\frac{1}{16}\begin{bmatrix}6&4&1&0&0&0&0&0\\4&6&4&1&0&0&0&0\\1&4&6&4&1&0&0&0\\0&1&4&6&4&1&0&0\\0&0&1&4&6&4&1&0\\0&0&0&1&4&6&4&1\\0&0&0&0&1&4&6&4\\0&0&0&0&0&1&4&6\end{bmatrix}$$

The Gaussian pyramid operators, in matrix form

$$\frac{1}{16}\begin{bmatrix} 6 & 4 & 1 & 0 & 0 & 0 & 0 & 0 \\ 4 & 6 & 4 & 1 & 0 & 0 & 0 & 0 \\ 1 & 4 & 6 & 4 & 1 & 0 & 0 & 0 \\ 0 & 1 & 4 & 6 & 4 & 1 & 0 & 0 \\ 0 & 0 & 1 & 4 & 6 & 4 & 1 & 0 \\ 0 & 0 & 0 & 1 & 4 & 6 & 4 & 1 \\ 0 & 0 & 0 & 0 & 1 & 4 & 6 & 4 \\ 0 & 0 & 0 & 0 & 0 & 1 & 4 & 6 \end{bmatrix}$$

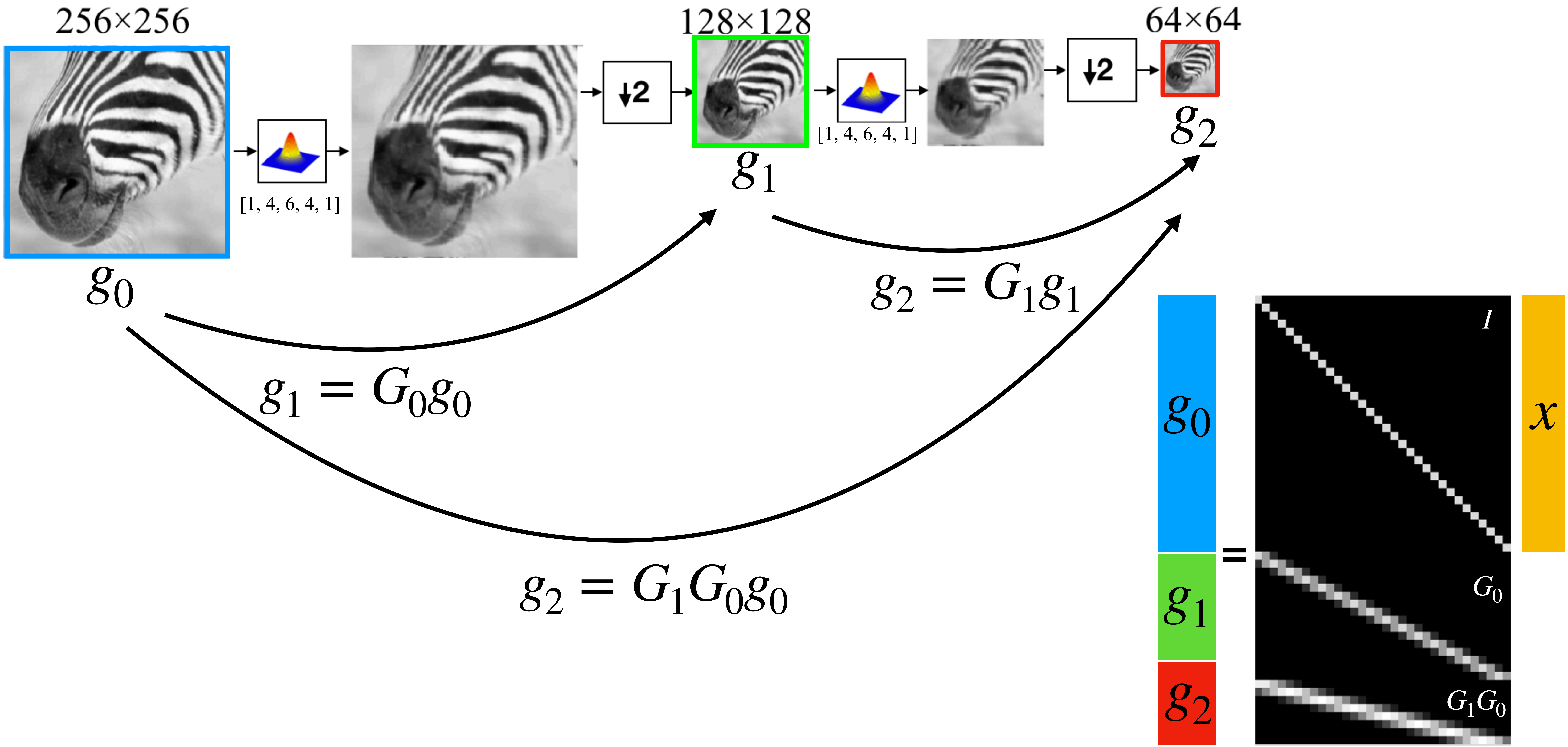
$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$



$$g_1 = G_0 g_0$$

$$G_0 = \frac{1}{16}\begin{bmatrix} 6 & 4 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 4 & 6 & 4 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 4 & 6 & 4 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 4 & 6 & 4 \end{bmatrix}$$

The Gaussian pyramid as a matrix multiplication

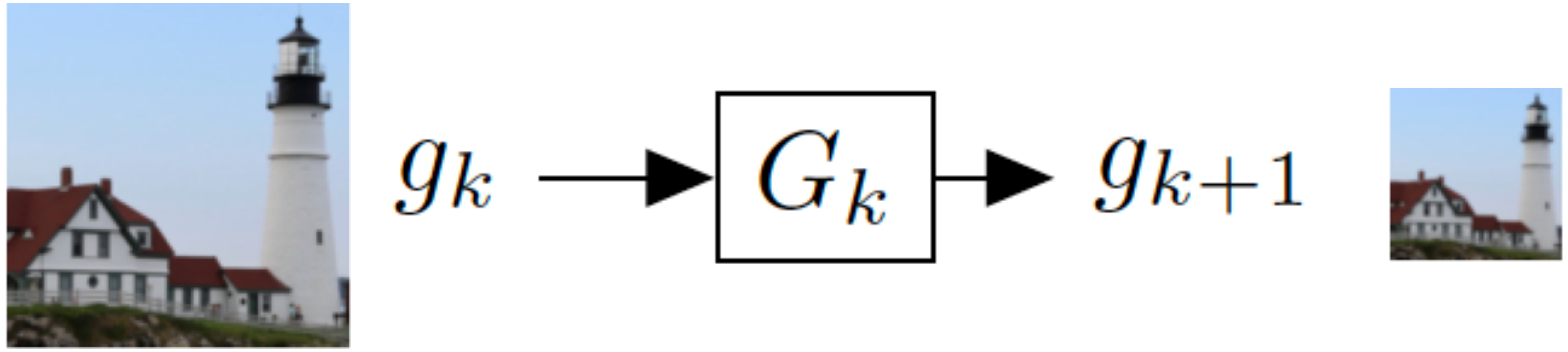


$$g_1 = G_0 g_0$$

$$g_2 = G_1 g_1$$

$$g_2 = G_1 G_0 g_0$$

The Gaussian pyramid summary



For each level

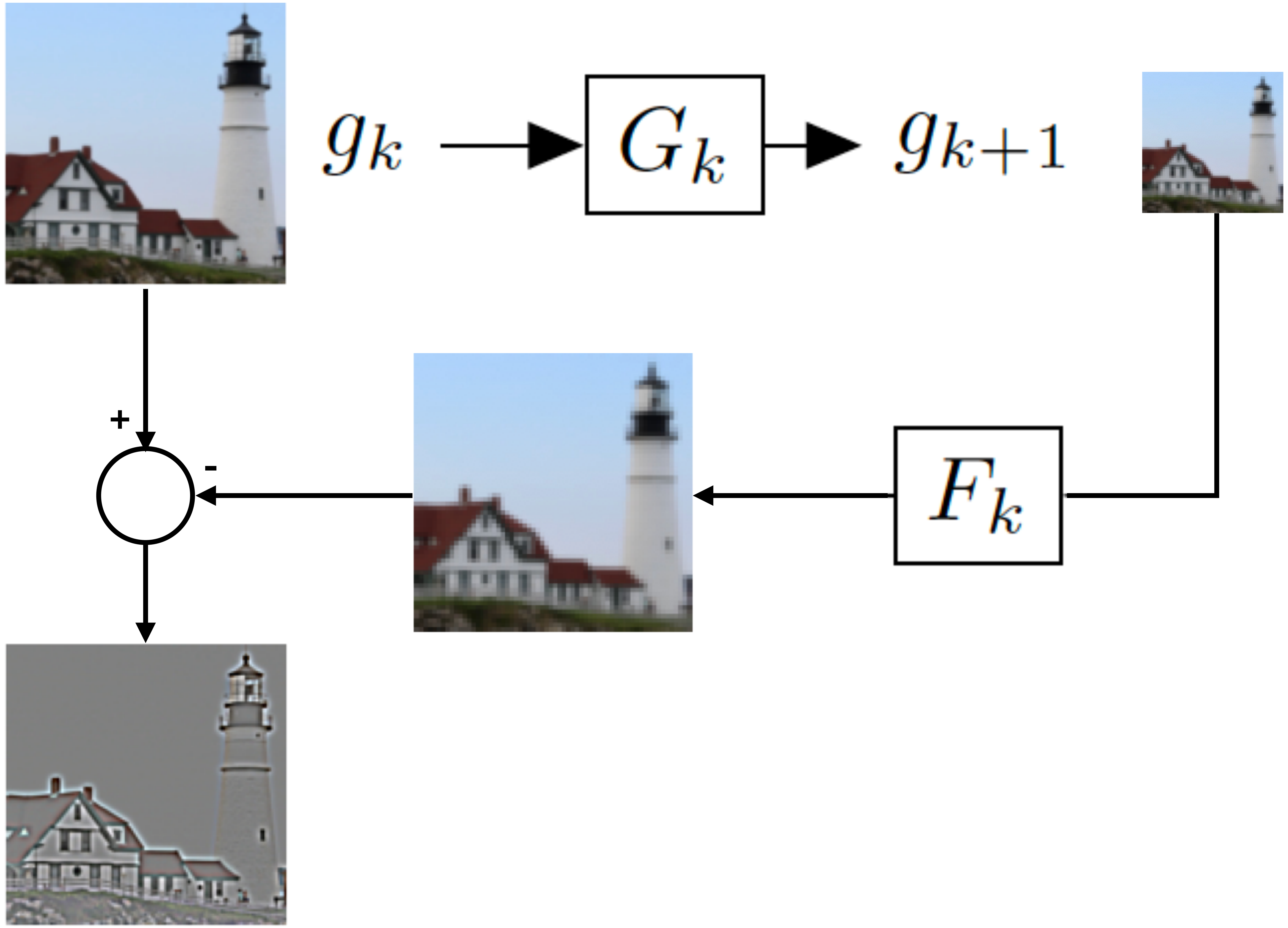
1. Blur input image with a Gaussian filter
2. Downsample image

Gaussian pyramid applications

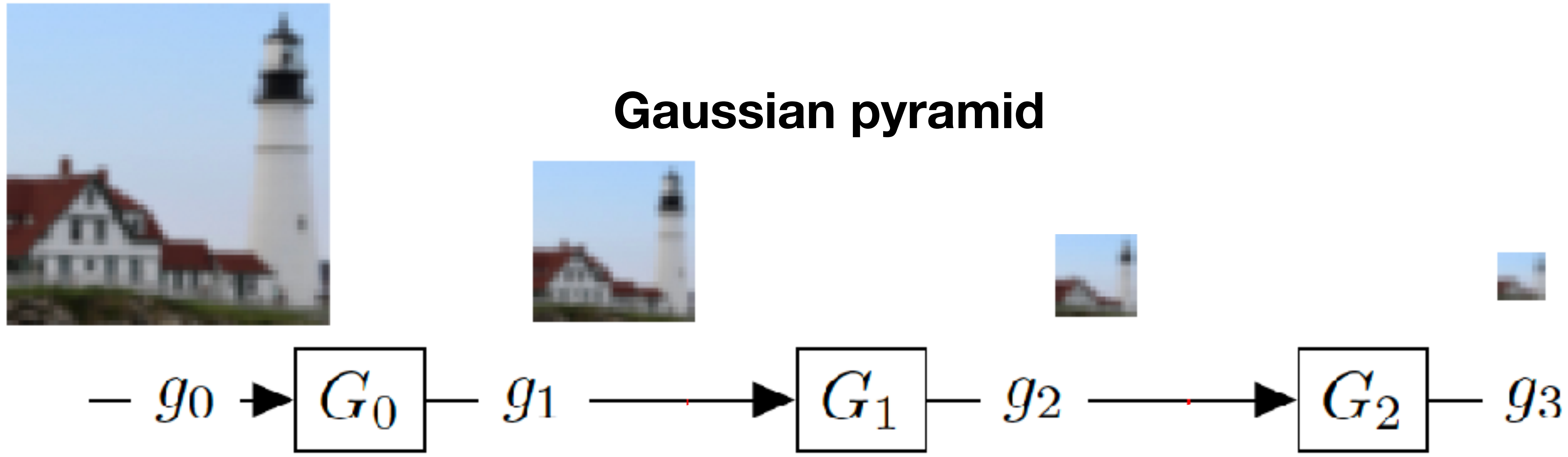
- Object recognition
- Neural Network image synthesis

The Laplacian Pyramid

Compute the difference between upsampled Gaussian pyramid level k+1 and Gaussian pyramid level k.

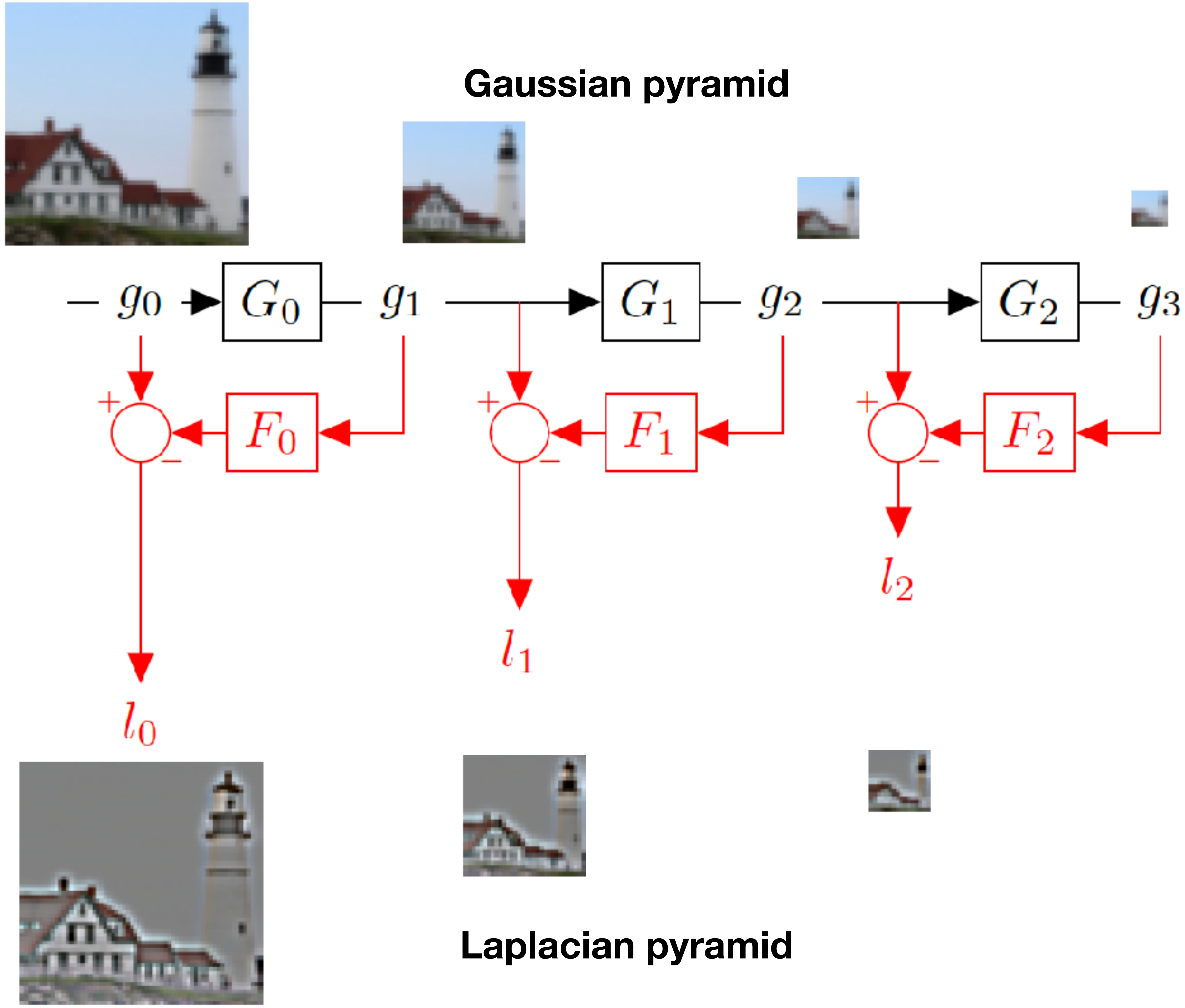


The Laplacian Pyramid



Gaussian pyramid

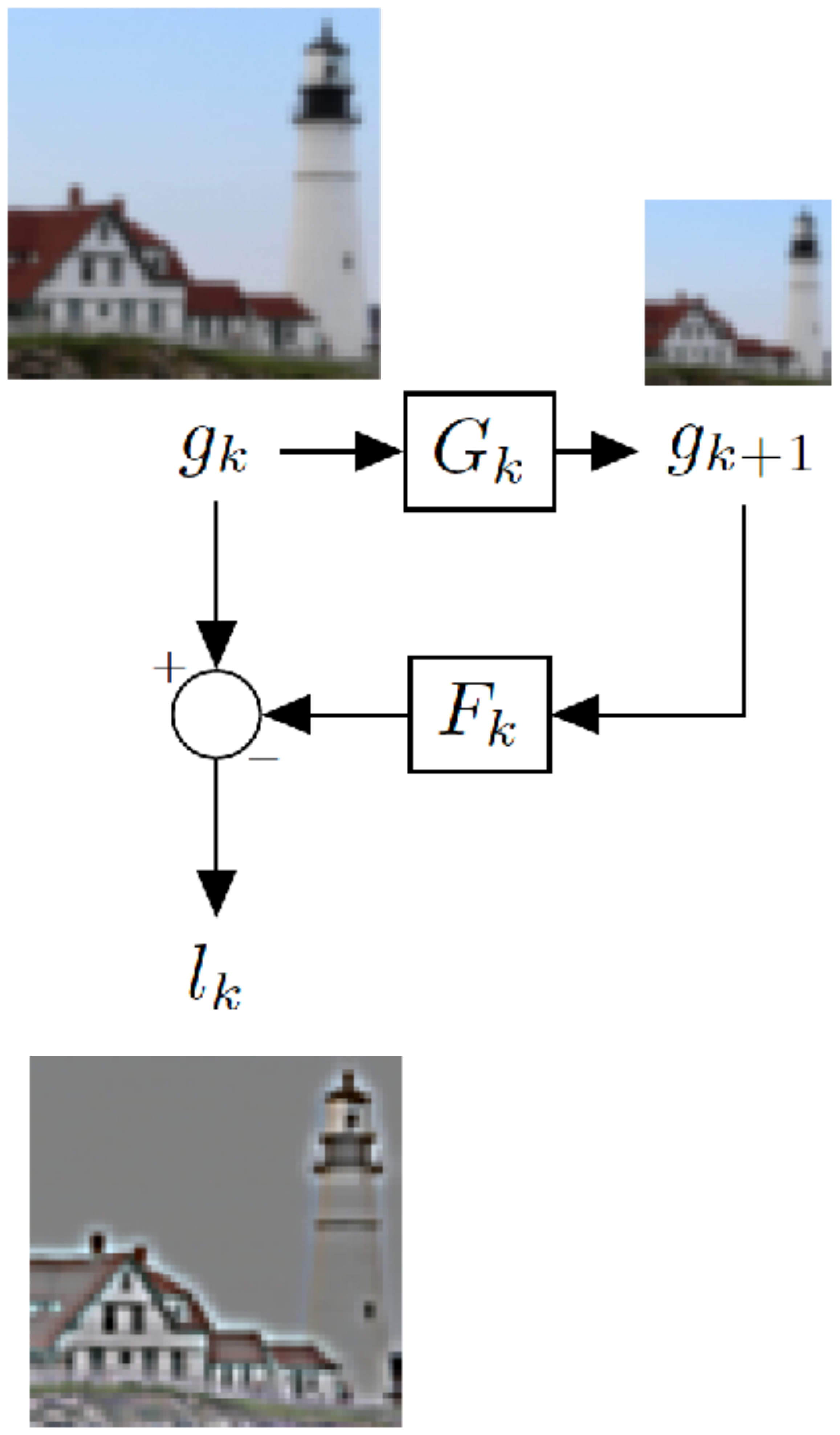
The Laplacian Pyramid



Gaussian pyramid

Laplacian pyramid

The Laplacian Pyramid operators in matrix form



Blurring and downsampling:

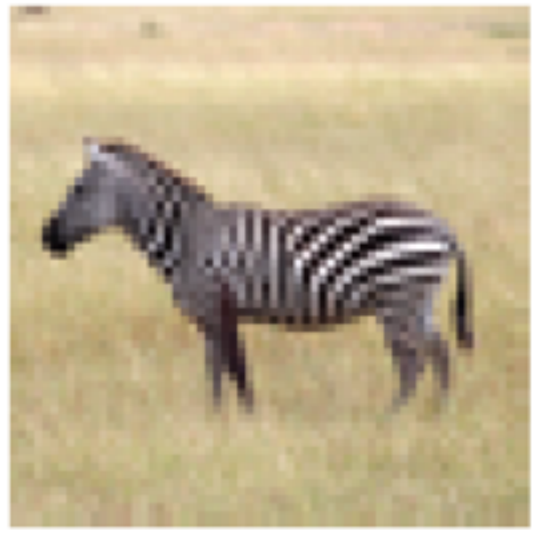
$$G_0 = \underset{\text{(Downsampling by 2)}}{\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}} \frac{1}{16} \underset{\text{(blur)}}{\begin{bmatrix} 6 & 4 & 1 & 0 & 0 & 0 & 0 & 0 \\ 4 & 6 & 4 & 1 & 0 & 0 & 0 & 0 \\ 1 & 4 & 6 & 4 & 1 & 0 & 0 & 0 \\ 0 & 1 & 4 & 6 & 4 & 1 & 0 & 0 \\ 0 & 0 & 1 & 4 & 6 & 4 & 1 & 0 \\ 0 & 0 & 0 & 1 & 4 & 6 & 4 & 1 \\ 0 & 0 & 0 & 0 & 1 & 4 & 6 & 4 \\ 0 & 0 & 0 & 0 & 0 & 1 & 4 & 6 \end{bmatrix}}$$

Upsampling and blurring:

$$F_0 =$$

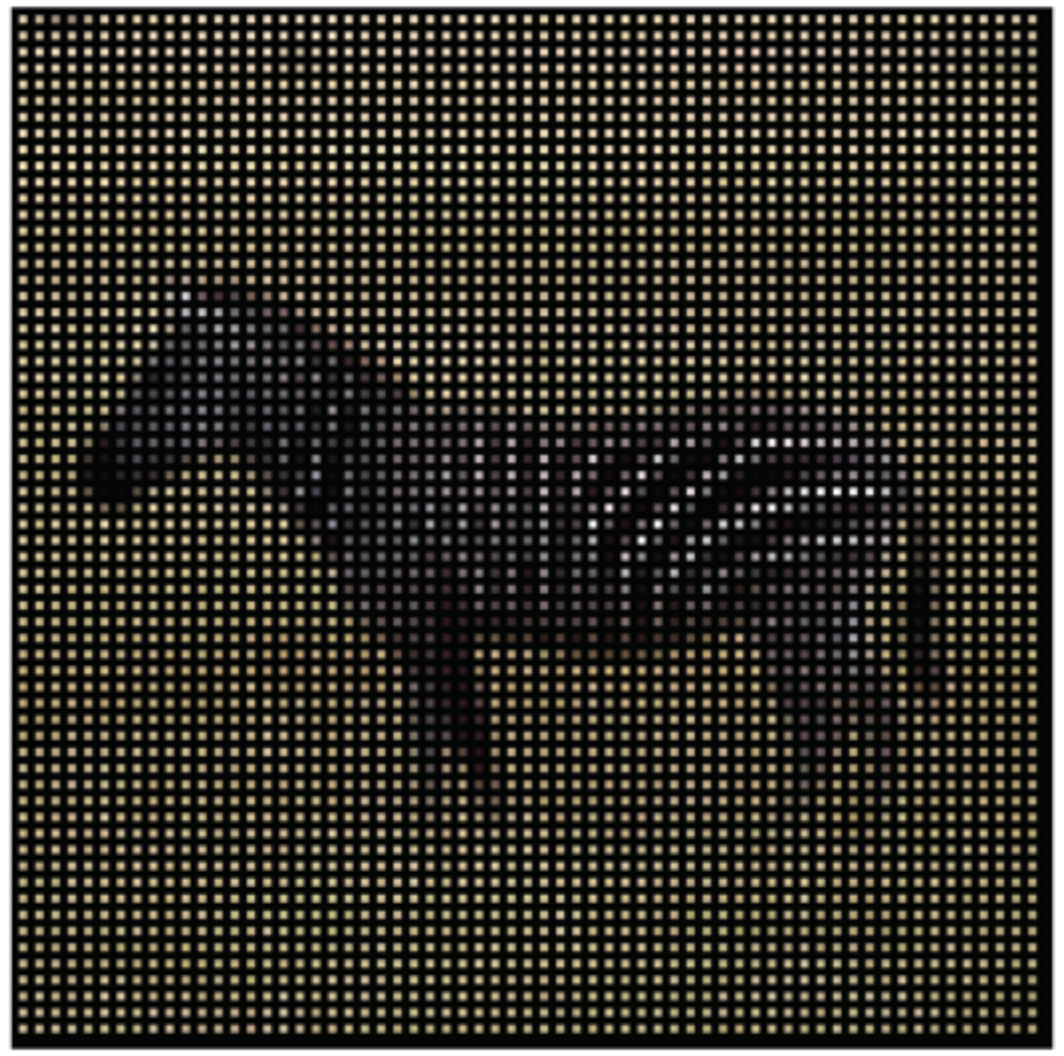
Upsampling

64x64



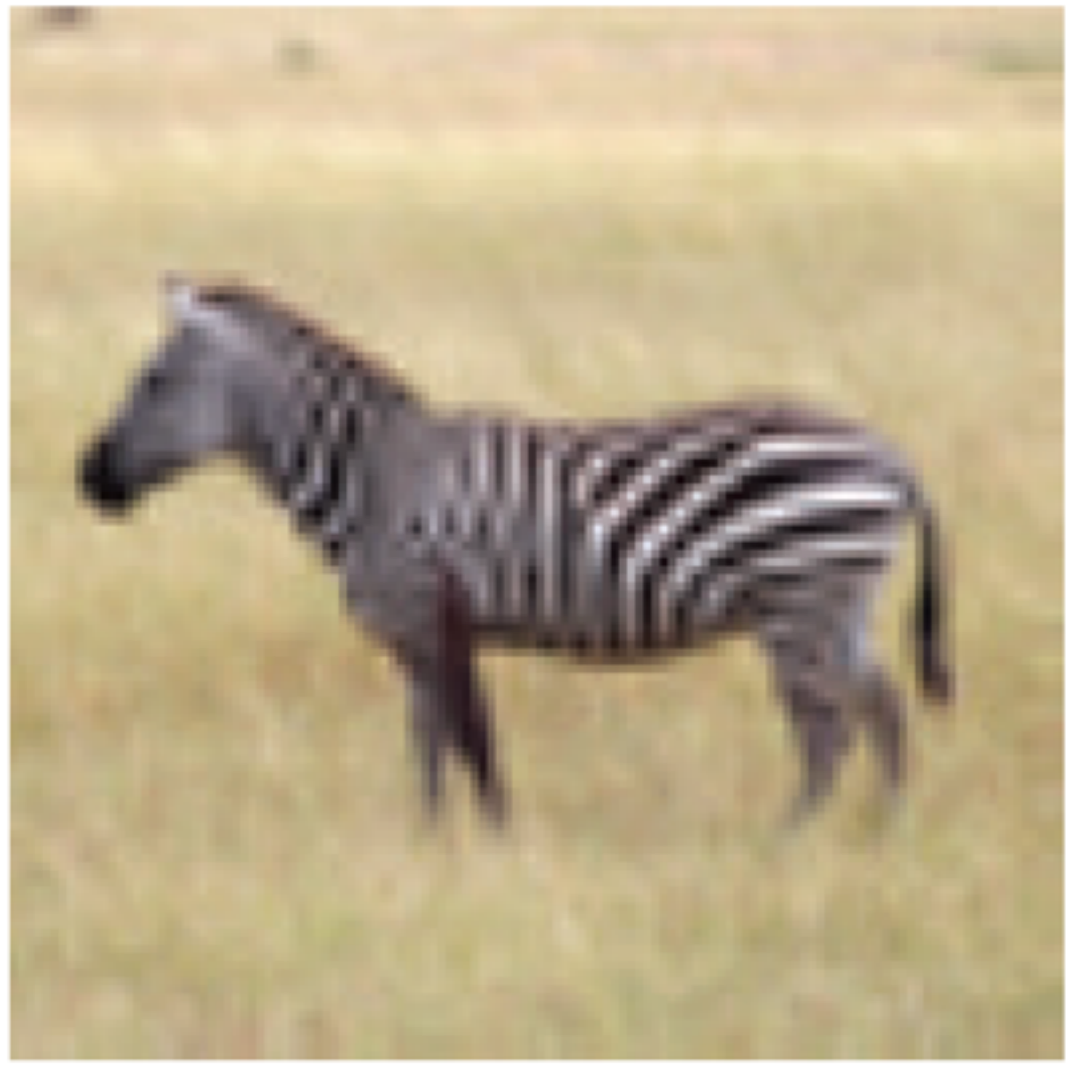
Insert zeros

128x128

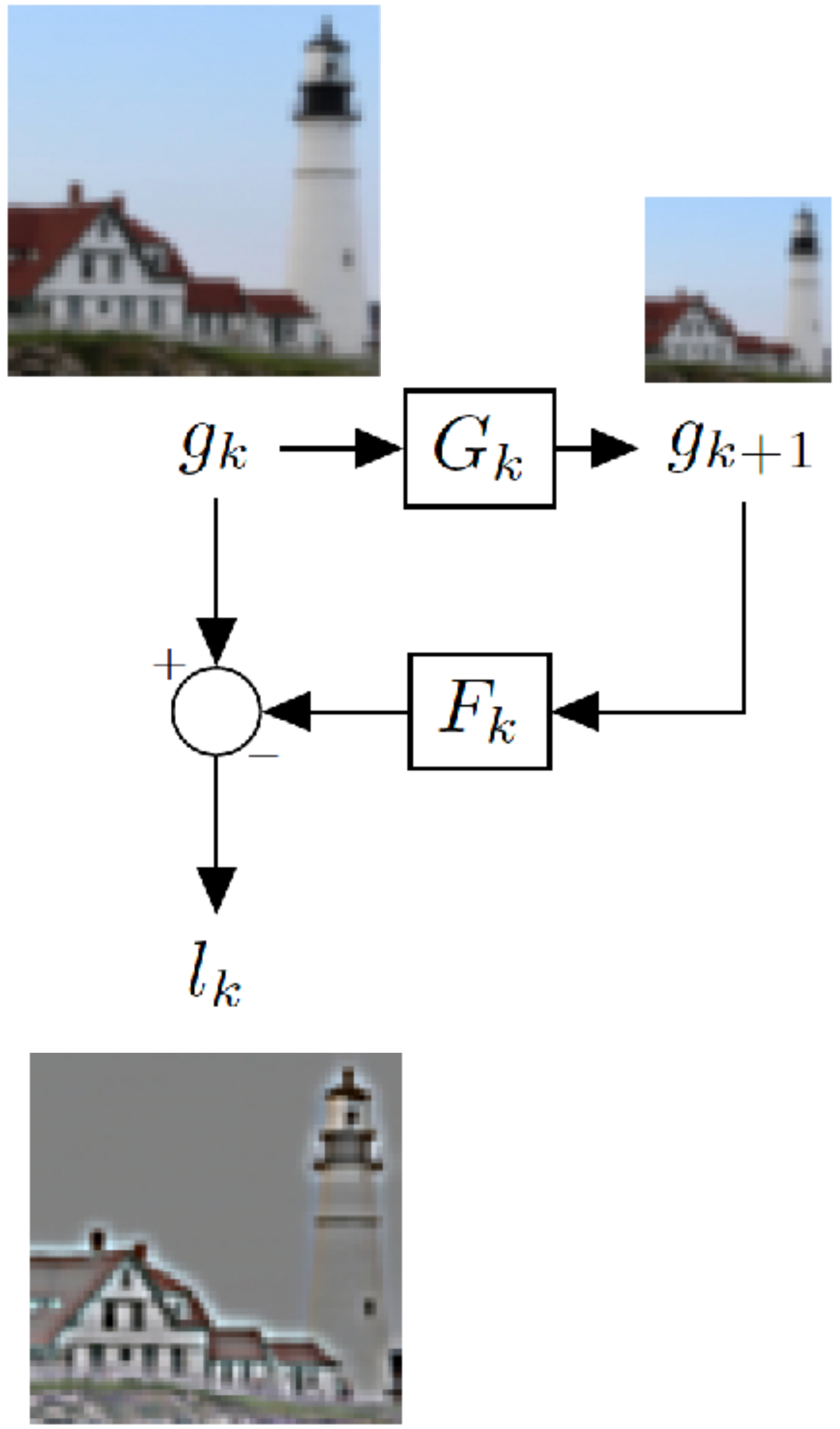


$$\bigcirc \begin{bmatrix} 0.25 & 0.5 & 0.25 \\ 0.5 & 1 & 0.5 \\ 0.25 & 0.5 & 0.25 \end{bmatrix} =$$

128x128



The Laplacian Pyramid operators in matrix form



Blurring and downsampling:

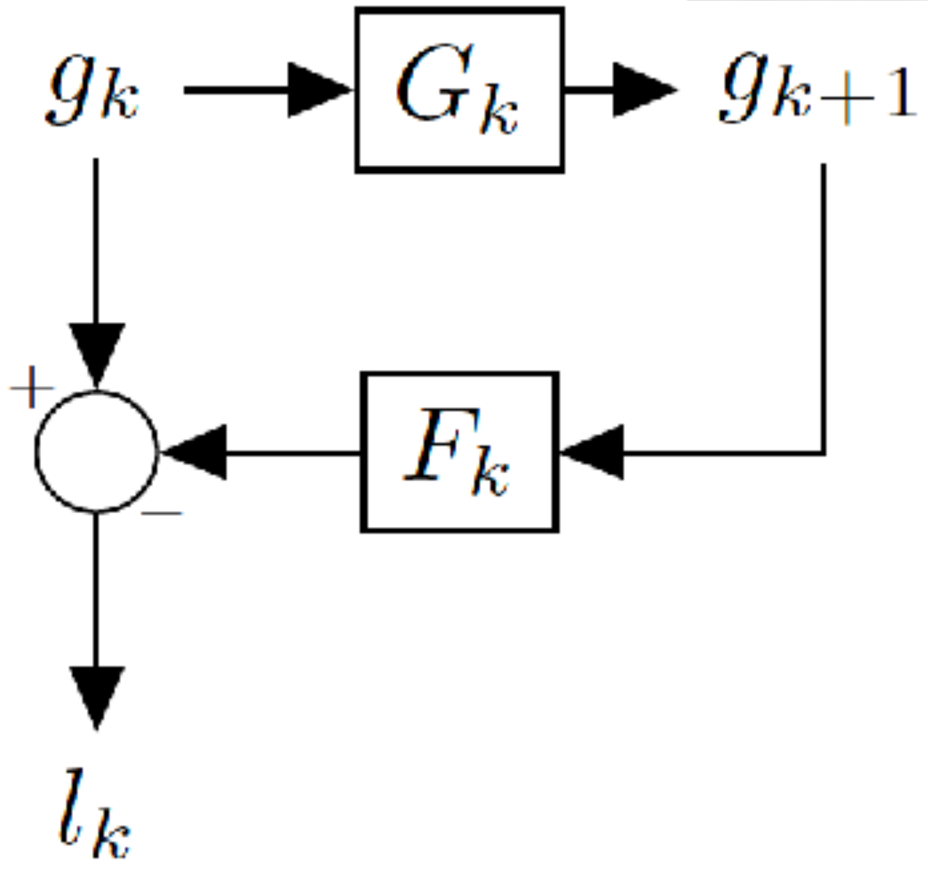
$$G_0 = \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}}_{\text{(Downsampling by 2)}} \frac{1}{16} \underbrace{\begin{bmatrix} 6 & 4 & 1 & 0 & 0 & 0 & 0 & 0 \\ 4 & 6 & 4 & 1 & 0 & 0 & 0 & 0 \\ 1 & 4 & 6 & 4 & 1 & 0 & 0 & 0 \\ 0 & 1 & 4 & 6 & 4 & 1 & 0 & 0 \\ 0 & 0 & 1 & 4 & 6 & 4 & 1 & 0 \\ 0 & 0 & 0 & 1 & 4 & 6 & 4 & 1 \\ 0 & 0 & 0 & 0 & 1 & 4 & 6 & 4 \\ 0 & 0 & 0 & 0 & 0 & 1 & 4 & 6 \end{bmatrix}}_{\text{(blur)}}$$

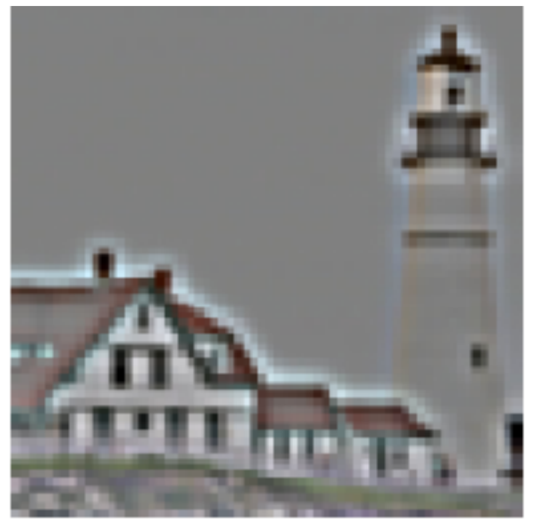
Upsampling and blurring:

$$F_0 = \frac{1}{8} \underbrace{\begin{bmatrix} 6 & 4 & 1 & 0 & 0 & 0 & 0 & 0 \\ 4 & 6 & 4 & 1 & 0 & 0 & 0 & 0 \\ 1 & 4 & 6 & 4 & 1 & 0 & 0 & 0 \\ 0 & 1 & 4 & 6 & 4 & 1 & 0 & 0 \\ 0 & 0 & 1 & 4 & 6 & 4 & 1 & 0 \\ 0 & 0 & 0 & 1 & 4 & 6 & 4 & 1 \\ 0 & 0 & 0 & 0 & 1 & 4 & 6 & 4 \\ 0 & 0 & 0 & 0 & 0 & 1 & 4 & 6 \end{bmatrix}}_{\text{(blur)}} \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}}_{\text{(Upsampling by 2)}}$$

$$\boxed{l_0 = (I_0 - F_0 G_0) g_0}$$

The Laplacian Pyramid

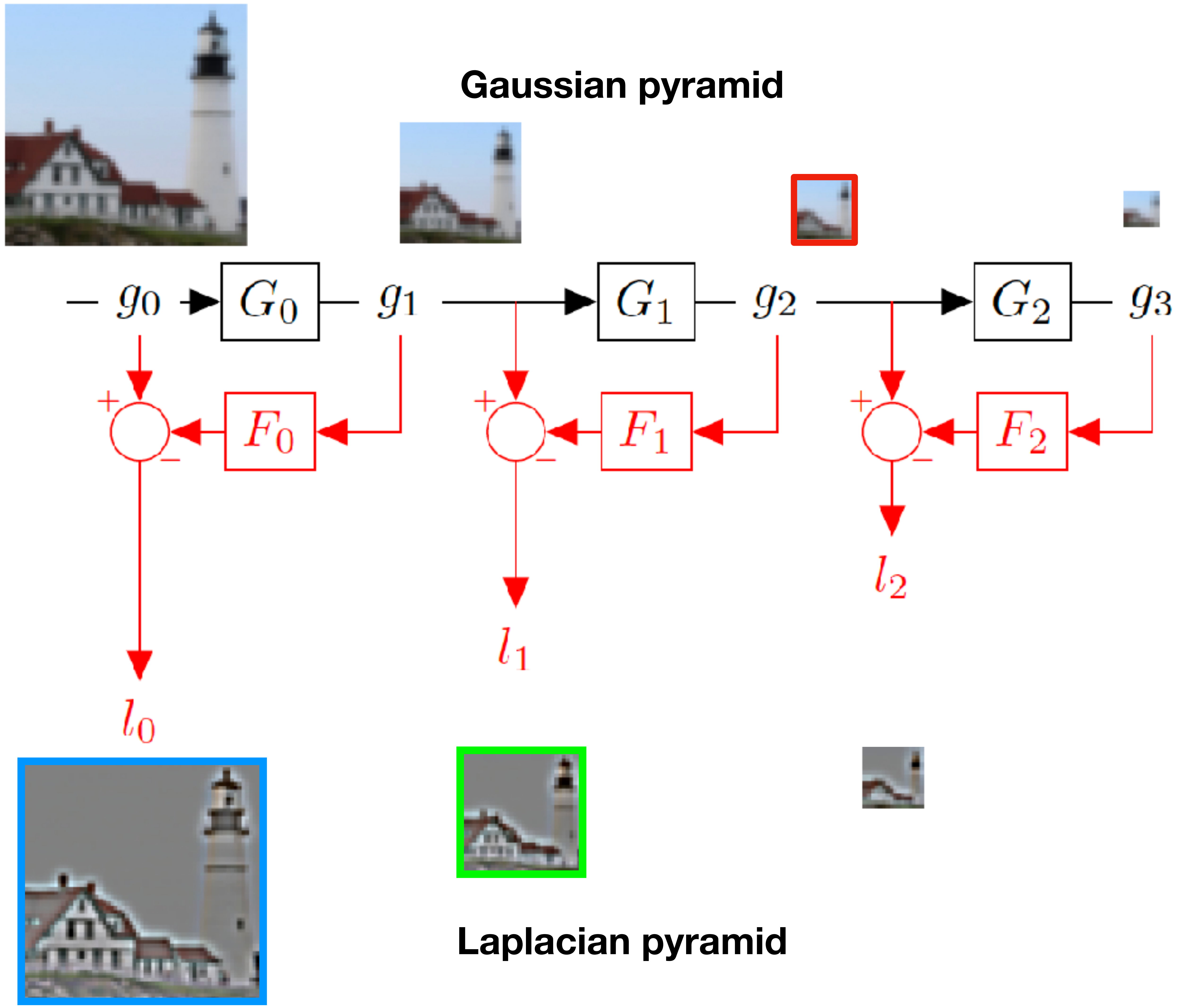




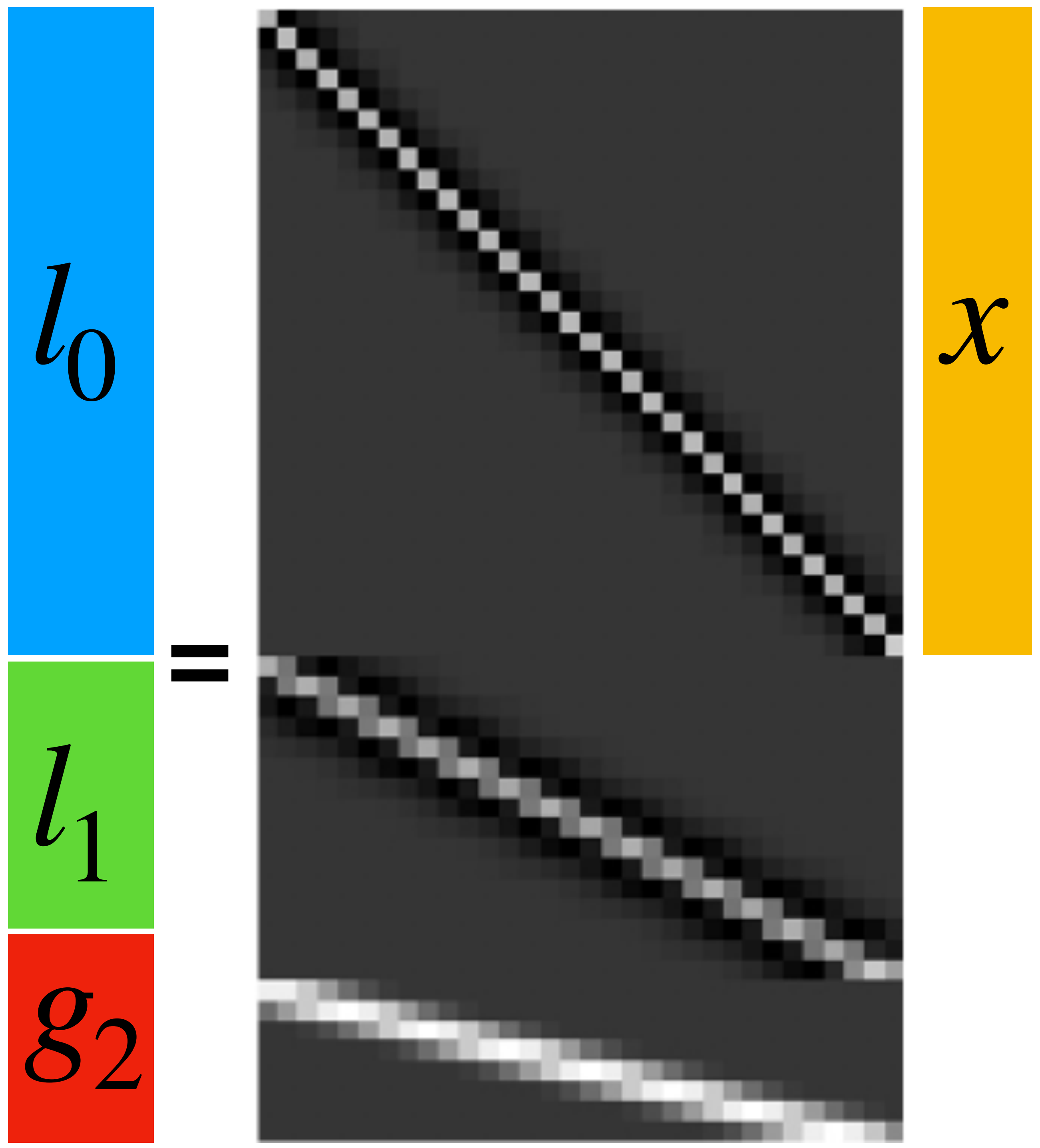
$$\boxed{l_0 = (I_0 - F_0 G_0) g_0}$$

$$= \frac{1}{256}\begin{bmatrix} 182 & -56 & -24 & -8 & -2 & 0 & 0 & 0 \\ -56 & 192 & -56 & -32 & -8 & 0 & 0 & 0 \\ -24 & -56 & 180 & -56 & -24 & -8 & -2 & 0 \\ -8 & -32 & -56 & 192 & -56 & -32 & -8 & 0 \\ -2 & -8 & -24 & -56 & 180 & -56 & -24 & -8 \\ 0 & 0 & -8 & -32 & -56 & 192 & -56 & -32 \\ 0 & 0 & -2 & -8 & -24 & -56 & 182 & -48 \\ 0 & 0 & 0 & 0 & -8 & -32 & -48 & 224 \end{bmatrix} x$$

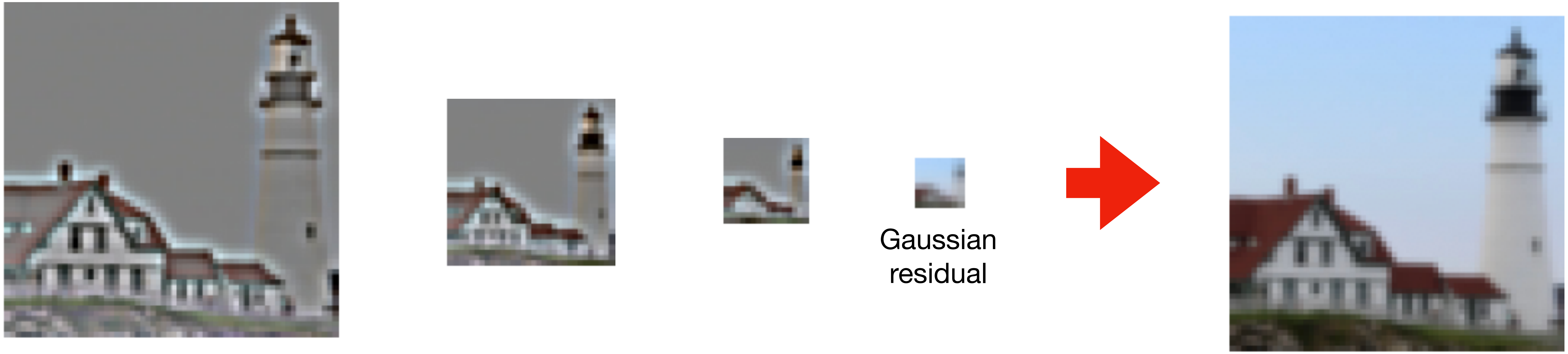
The Laplacian Pyramid shown in matrix form


Gaussian pyramid

Laplacian pyramid



Inverting the Laplacian Pyramid

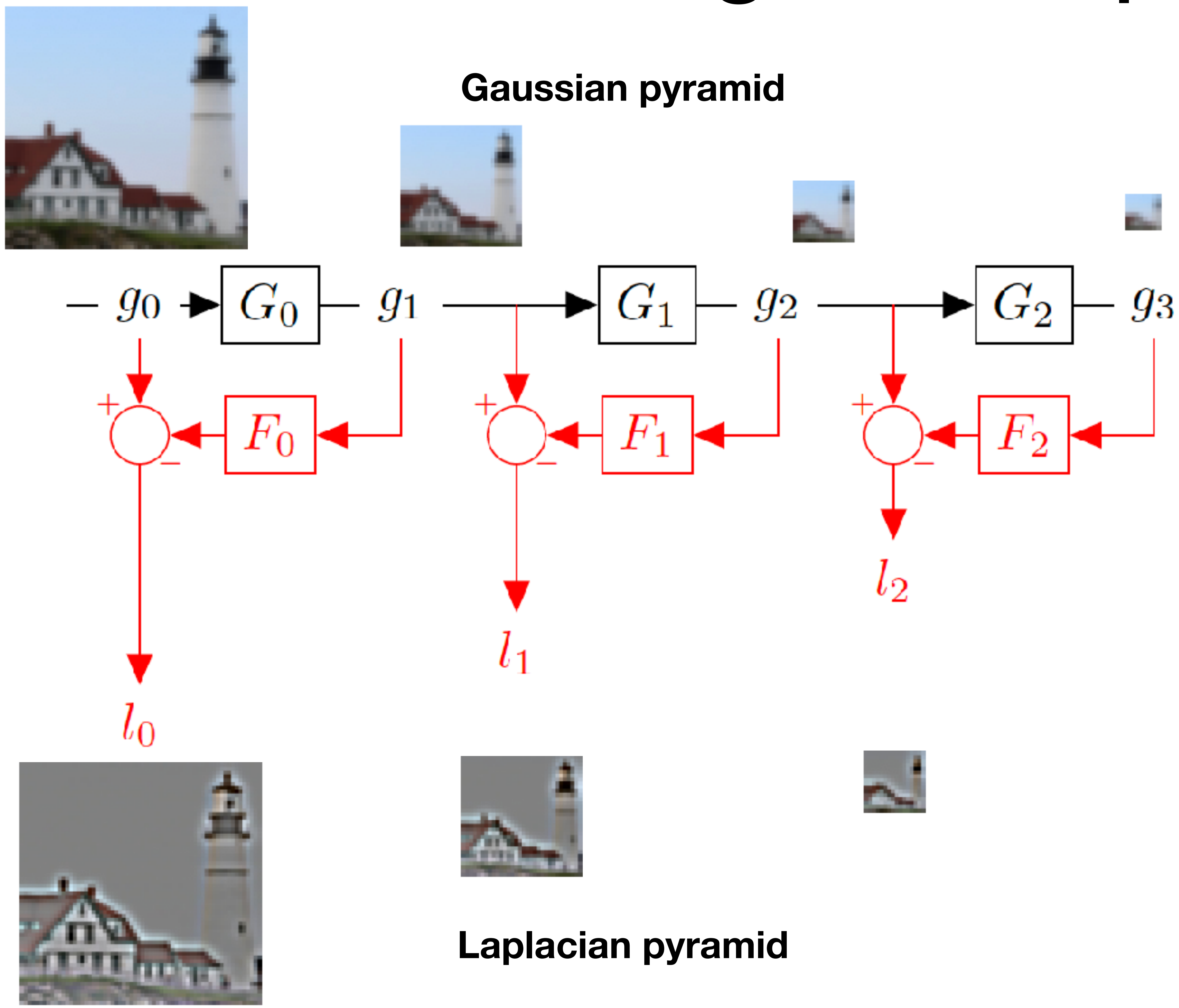


Gaussian residual

Laplacian pyramid

Can we invert the Laplacian Pyramid?

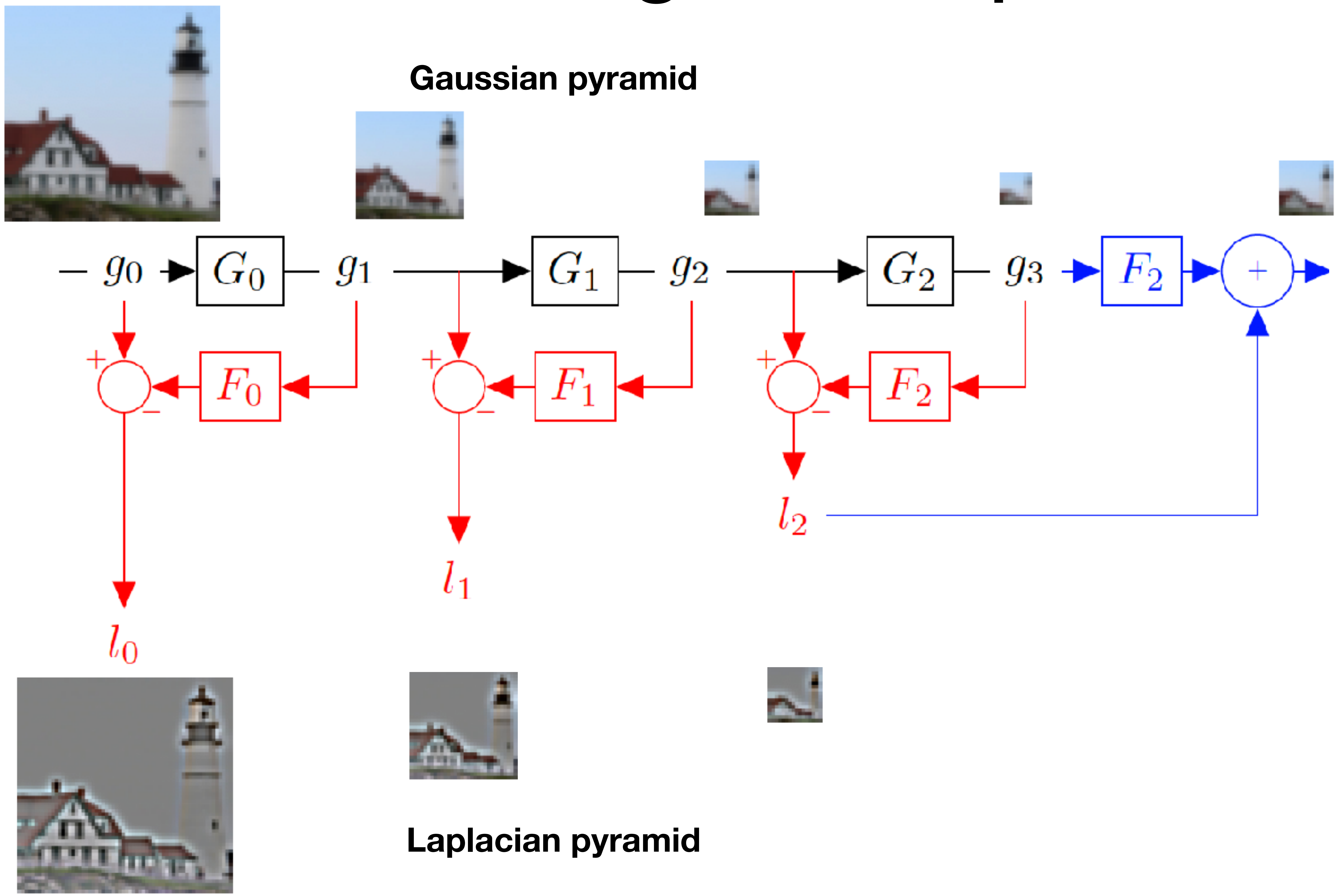
Inverting the Laplacian Pyramid



Gaussian pyramid

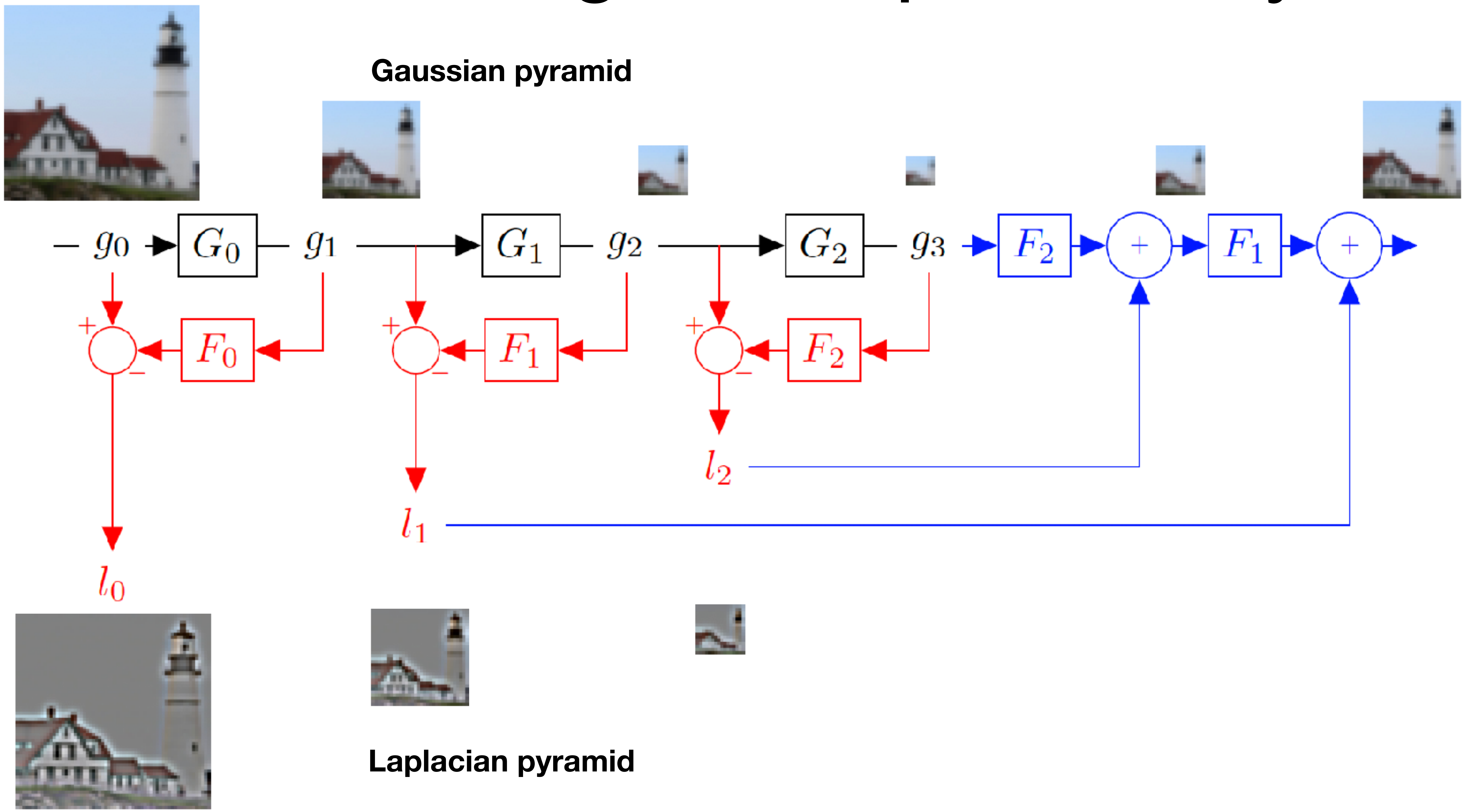
Laplacian pyramid

Inverting the Laplacian Pyramid


Gaussian pyramid

Laplacian pyramid

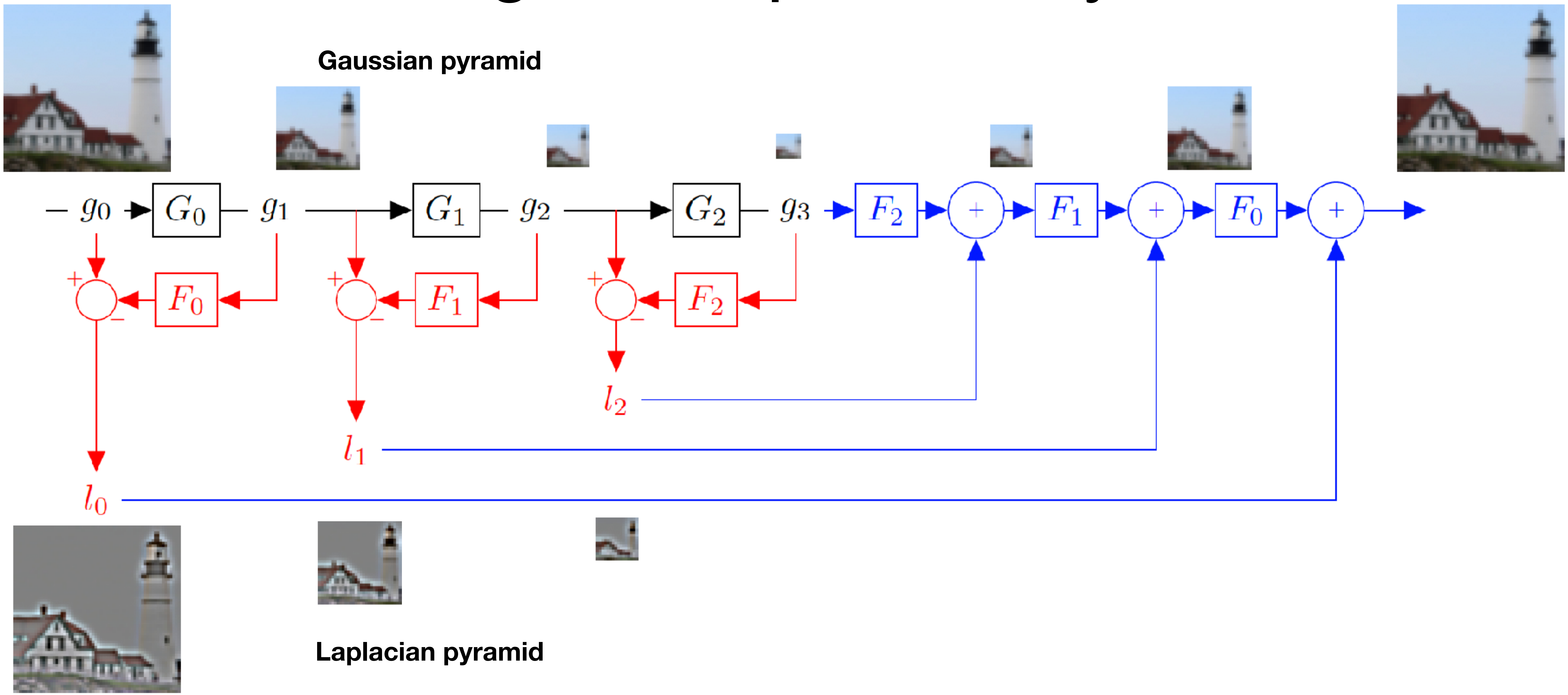
Inverting the Laplacian Pyramid



Gaussian pyramid

Laplacian pyramid

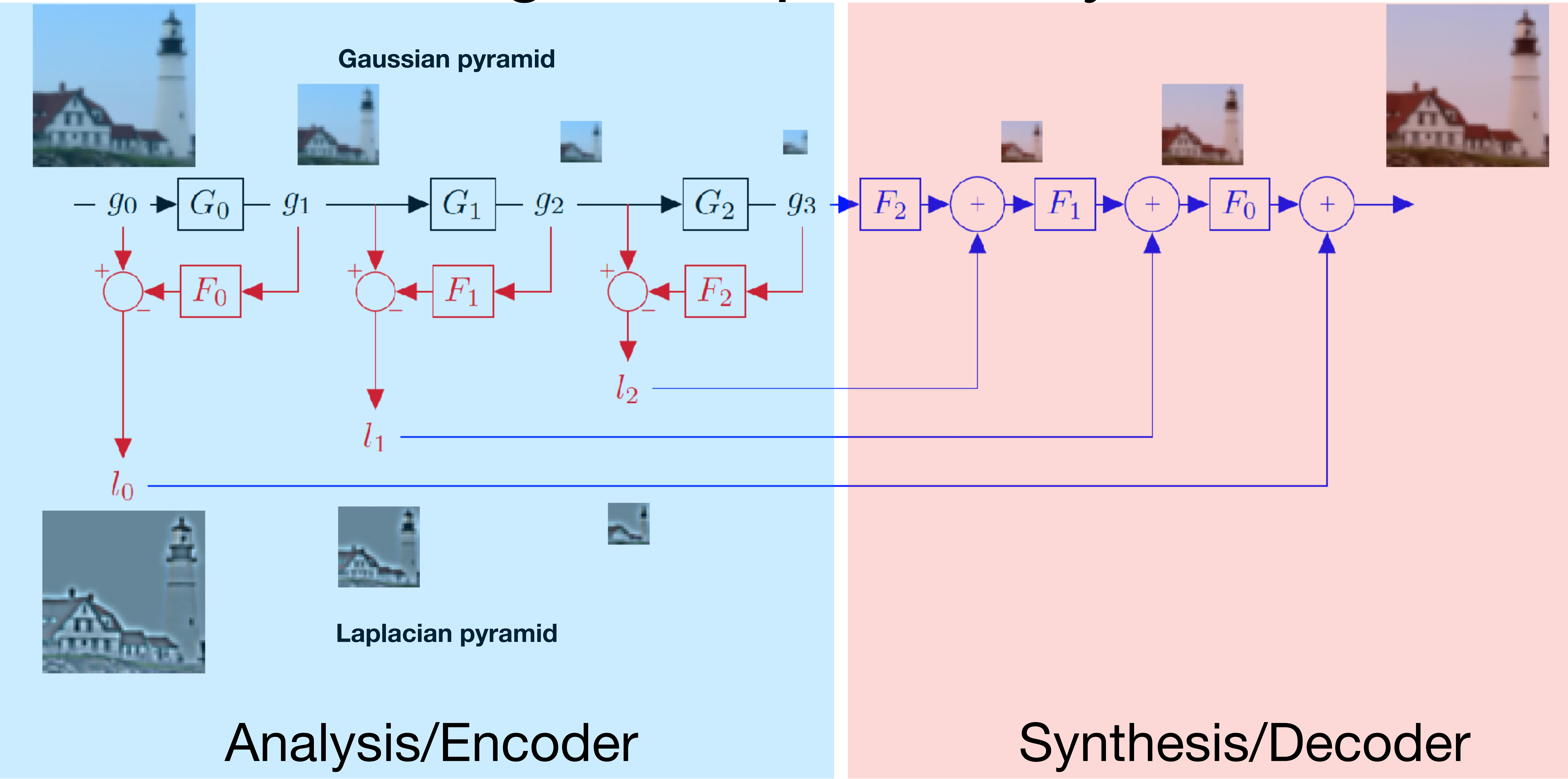
Inverting the Laplacian Pyramid



Gaussian pyramid

Laplacian pyramid

Inverting the Laplacian Pyramid



Laplacian pyramid applications

- Texture synthesis
- Image compression
- Noise removal
- Computing image features (e.g., SIFT)
- Image Blending…

Image Blending

Image Blending

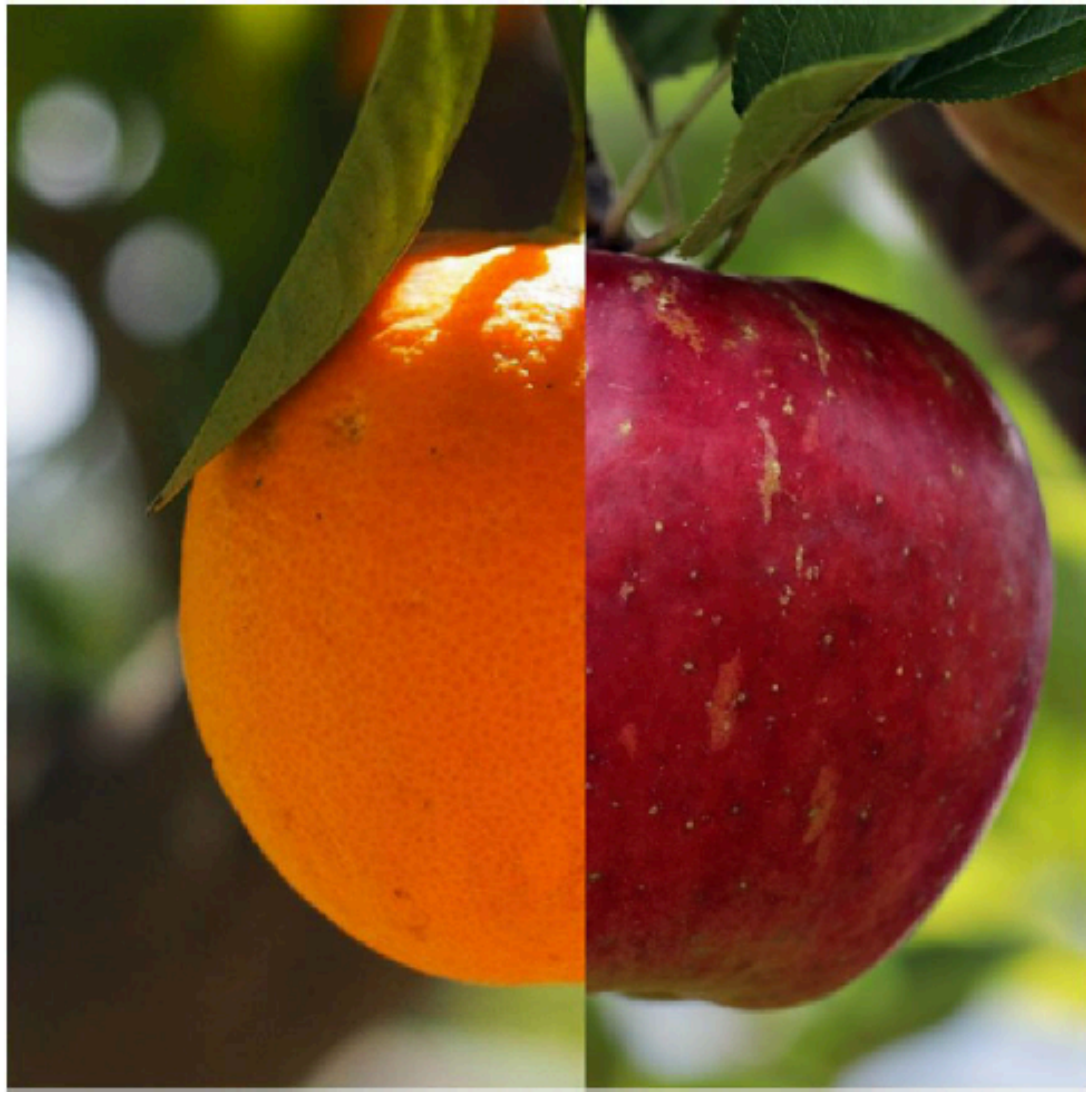
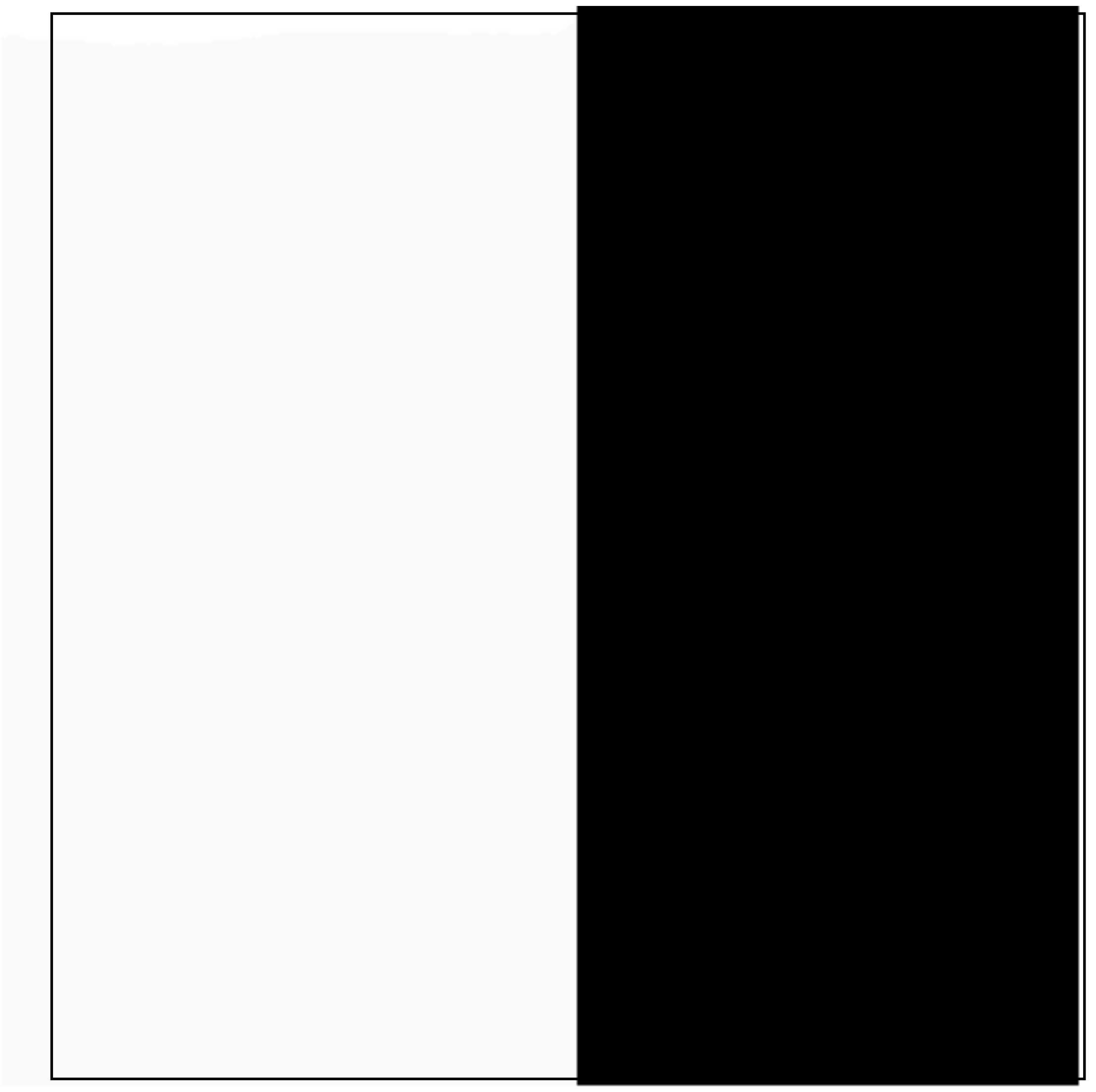


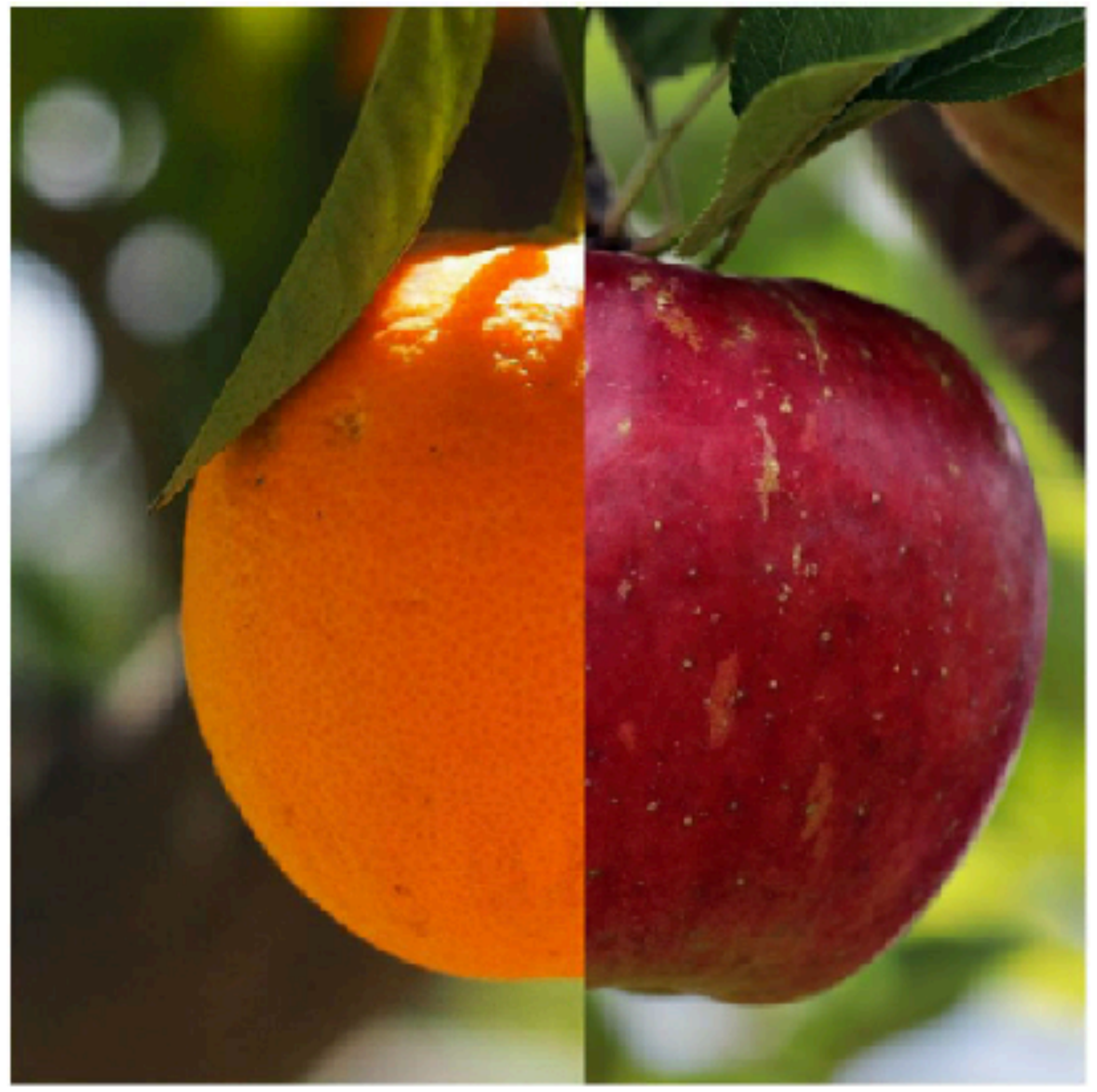
Image Blending

I^A

I^B



m

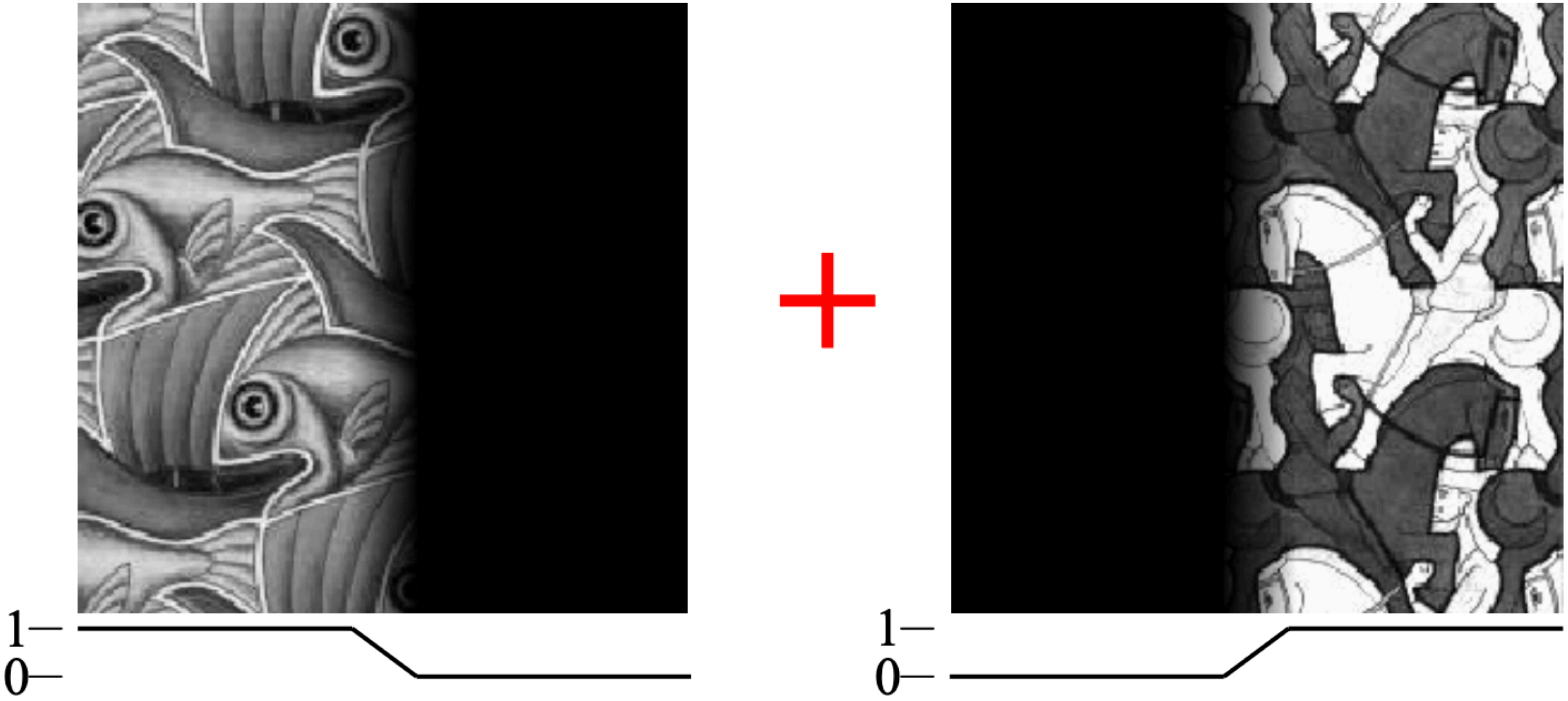


I

$$I = m * I^A + (1 - m) * I^B$$

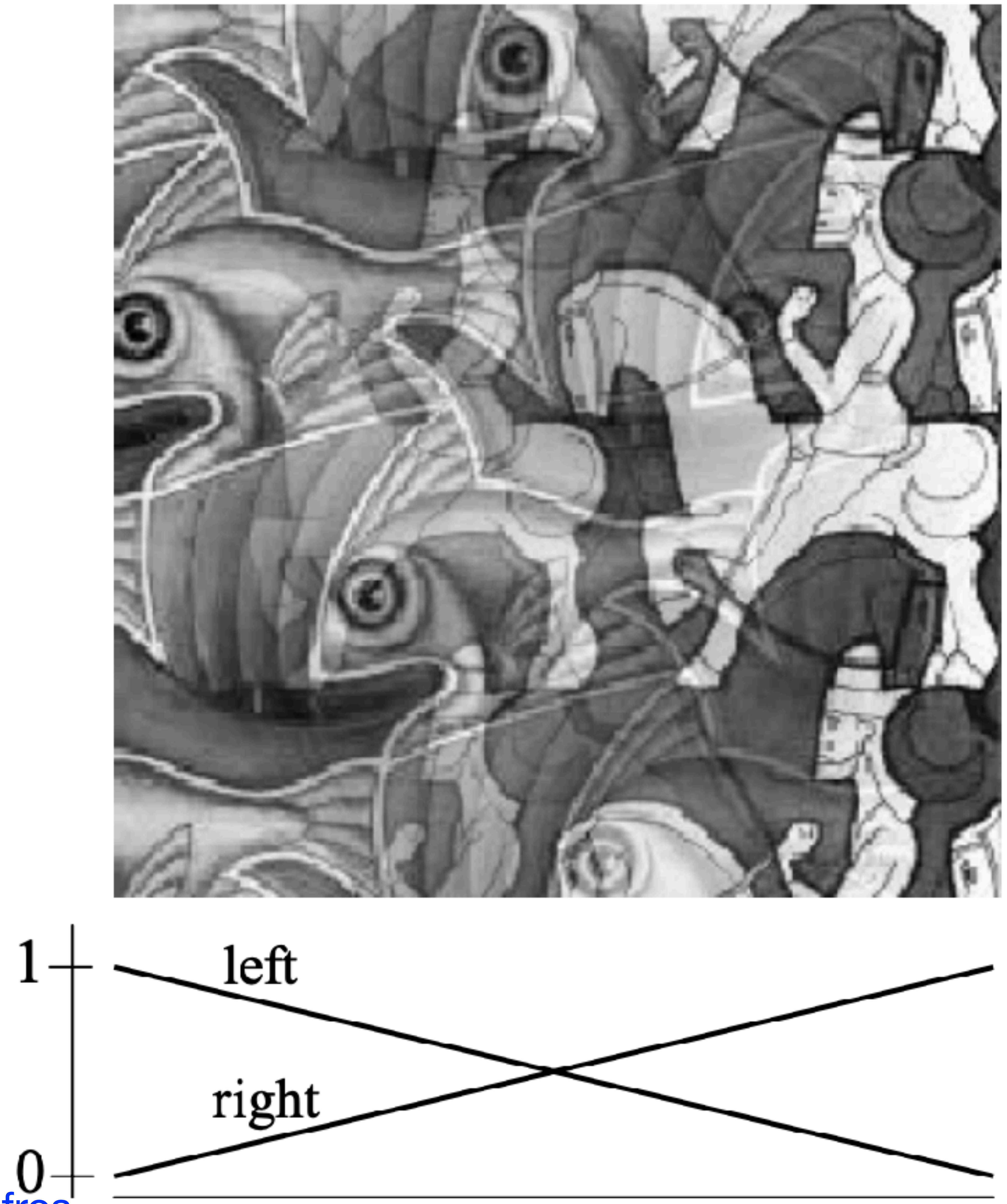
Image Blending

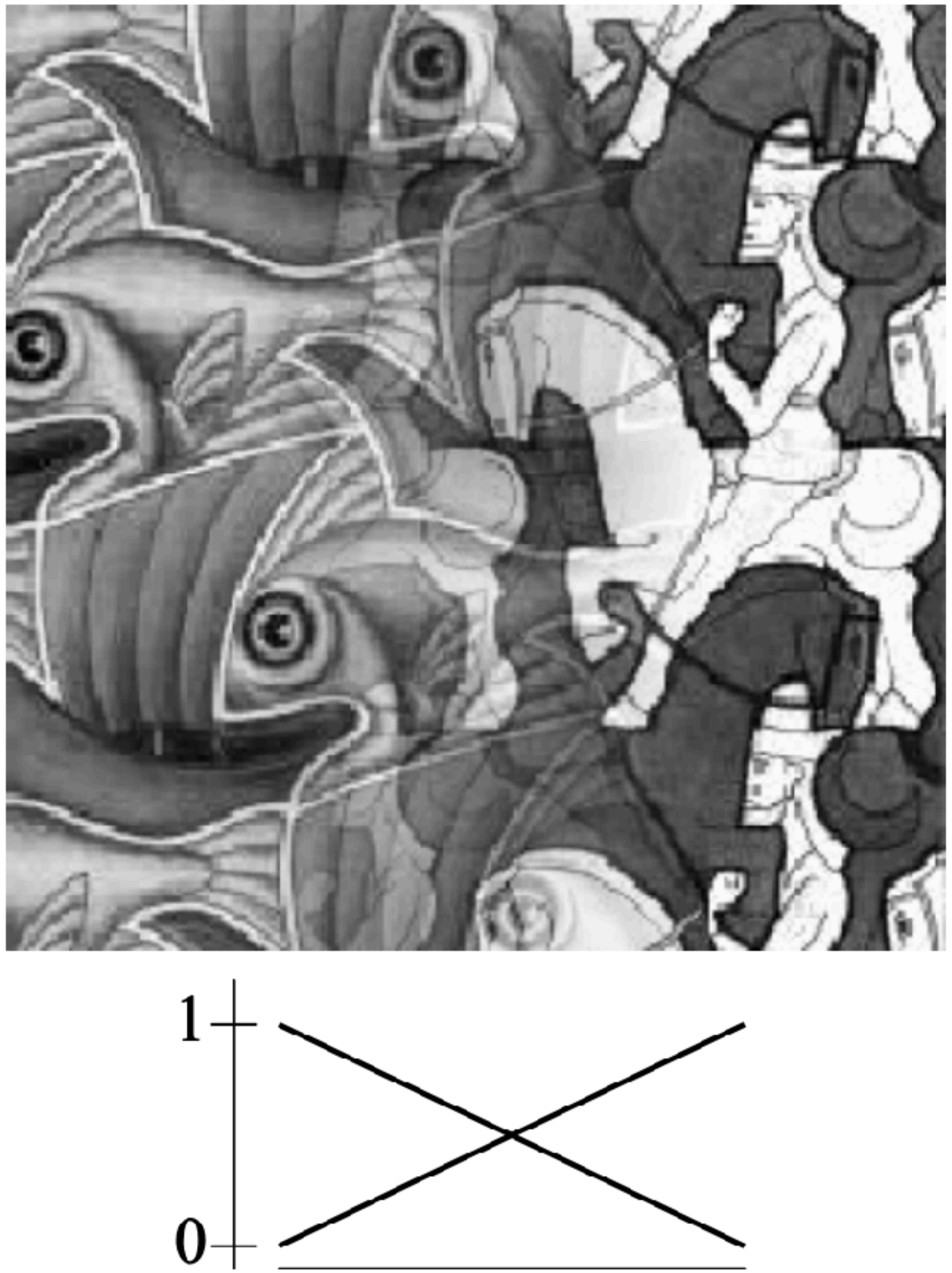
Feathering



Slide by A. Efros

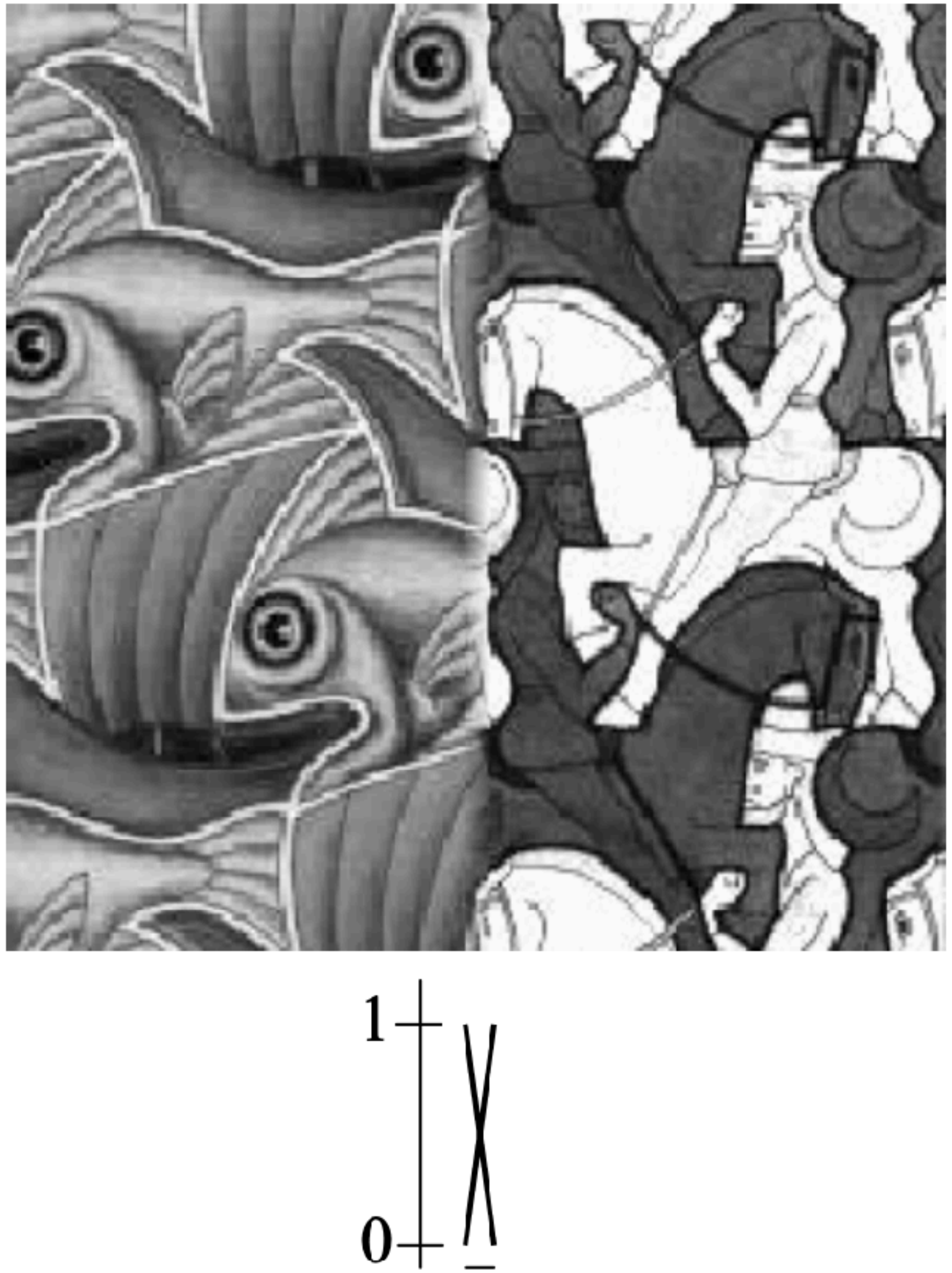
Affect of Window Size

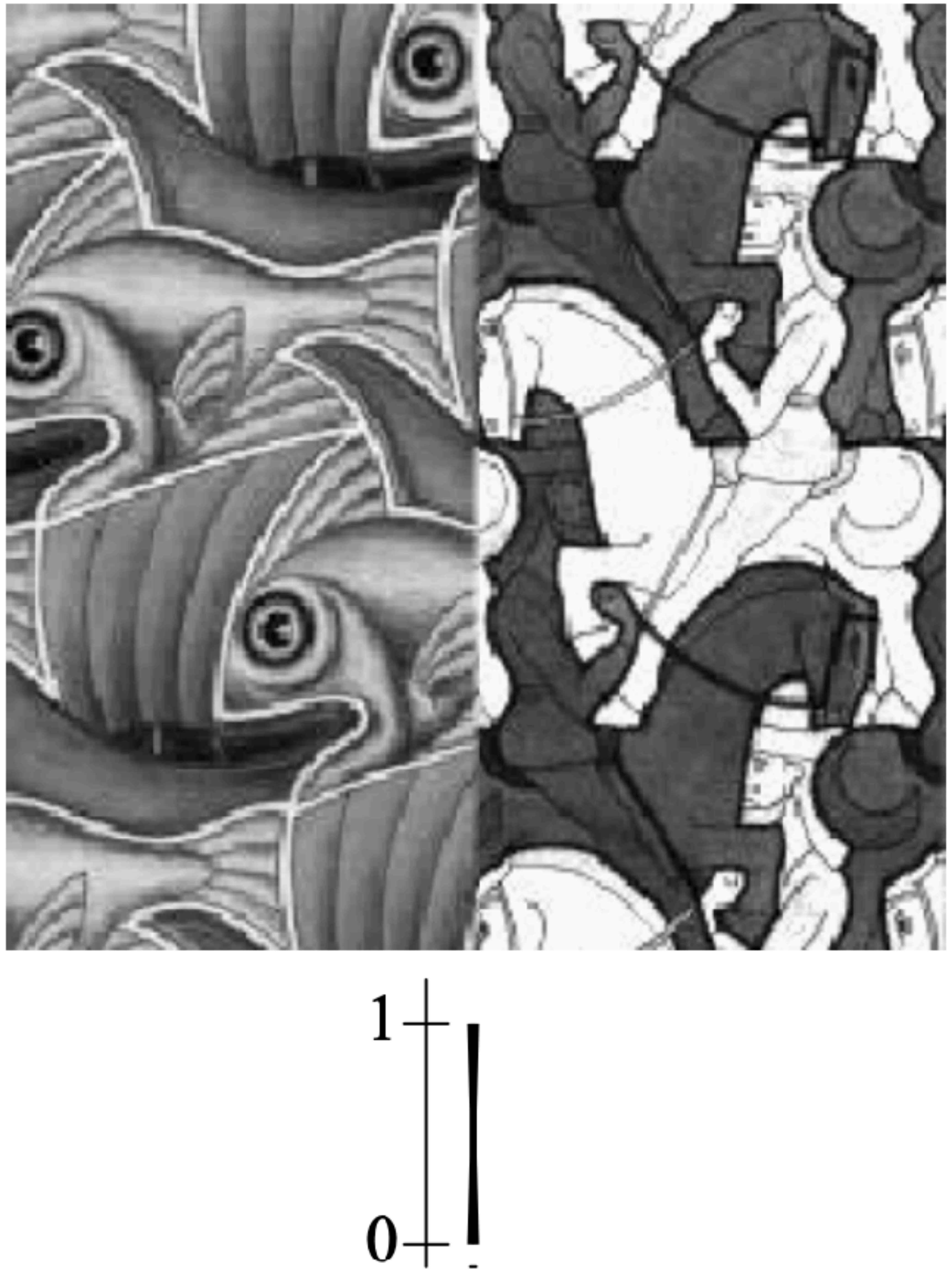




Slide by A. Efros

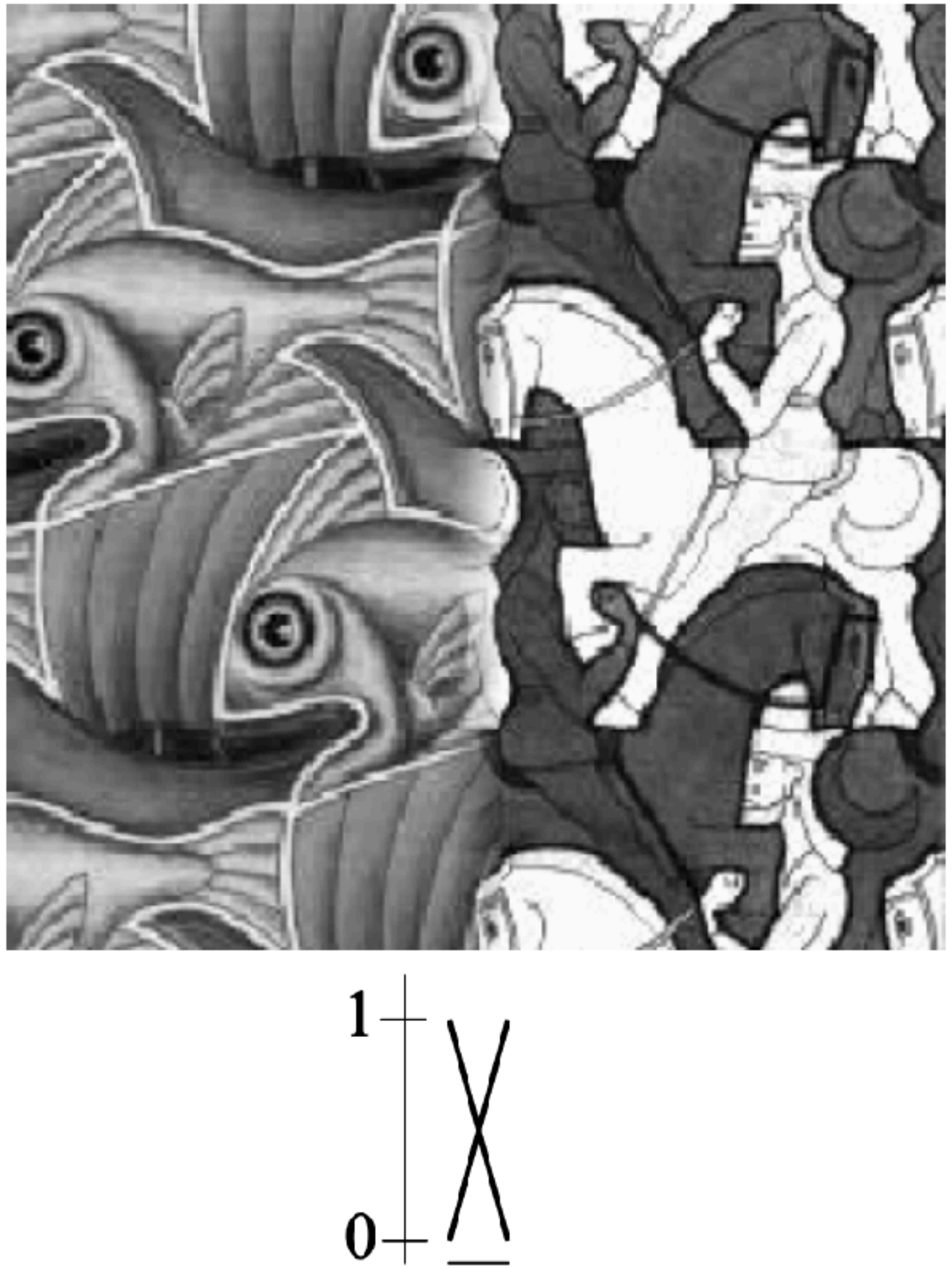
Affect of Window Size





Slide by A. Efros

Good Window Size



“Optimal” Window: smooth but not ghosted

Slide by A. Efros

What is the Optimal Window?

To avoid seams

- window >= size of largest prominent feature

To avoid ghosting

- window <= 2*size of smallest prominent feature

Natural to cast this in the *Fourier domain*

- largest frequency <= 2*size of smallest frequency
- image frequency content should occupy one “octave” (power of two)

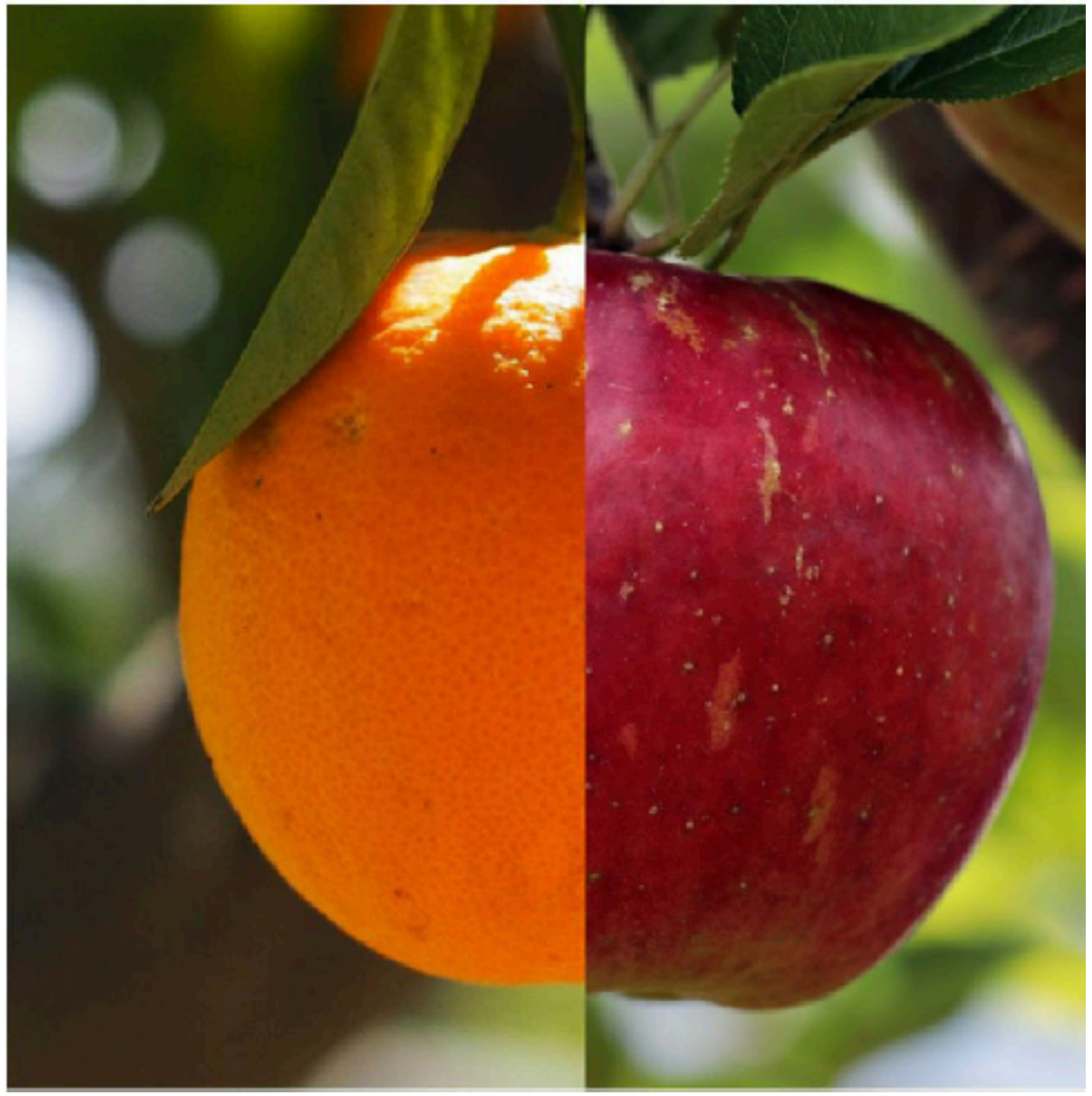
Slide by A. Efros

Image Blending with the Laplacian Pyramid

l_0^A
l_1^A
l_2^A
l_3^A
l_4^A
l_5^A
l_6^A
g_7^A
l_0^B
l_1^B
l_2^B
l_3^B
l_4^B
l_5^B
l_6^B
g_7^B
m_0
m_1
m_2
m_3
m_4
m_5
m_6
m_7

$$l_k = l_k^A * m_k + l_i^B * (1 - m_k)$$

Image Blending with the Laplacian Pyramid



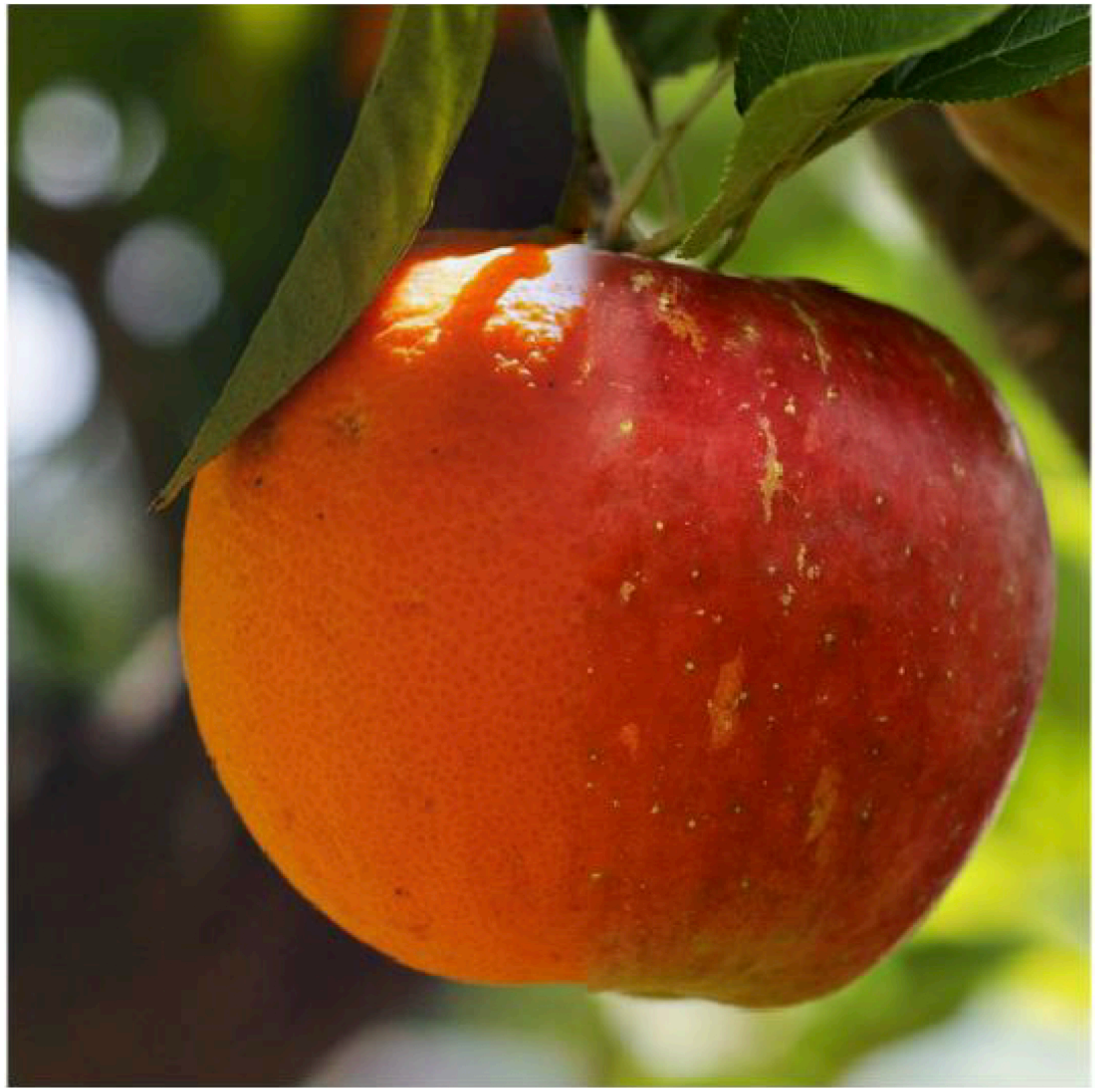
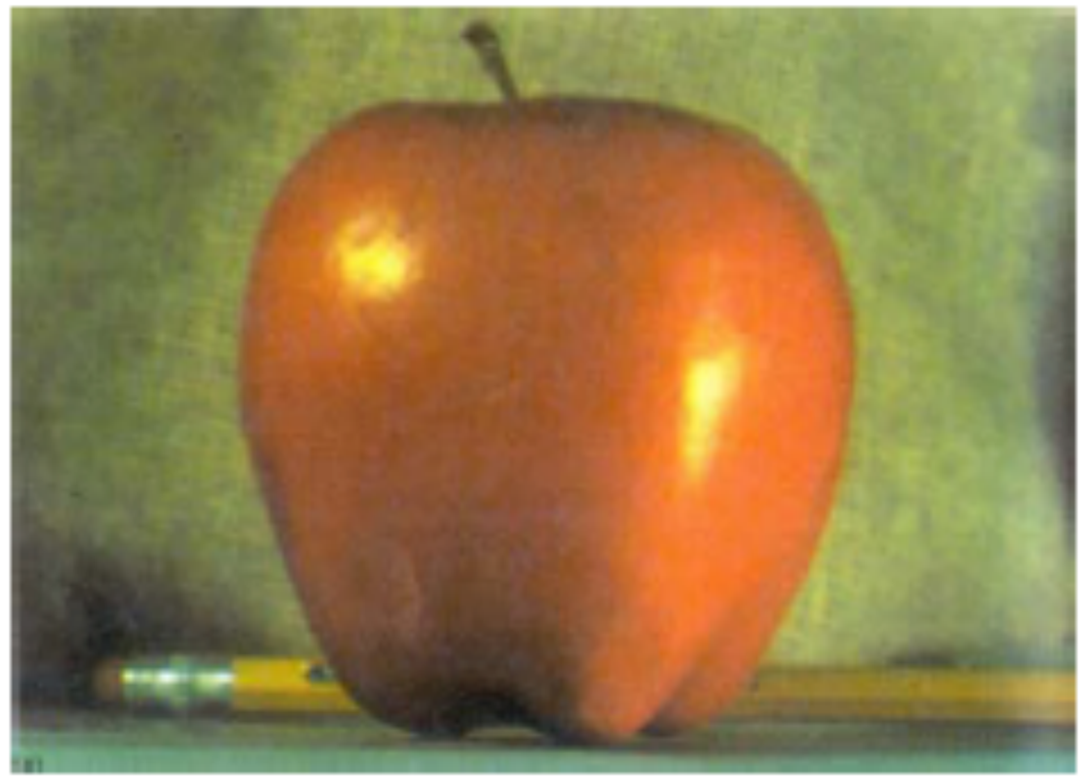


Image Blending with the Laplacian Pyramid

- Build Laplacian pyramid for both images: LA, LB
- Build Gaussian pyramid for mask: G
- Build a combined Laplacian pyramid.
- Collapse L to obtain the blended image.



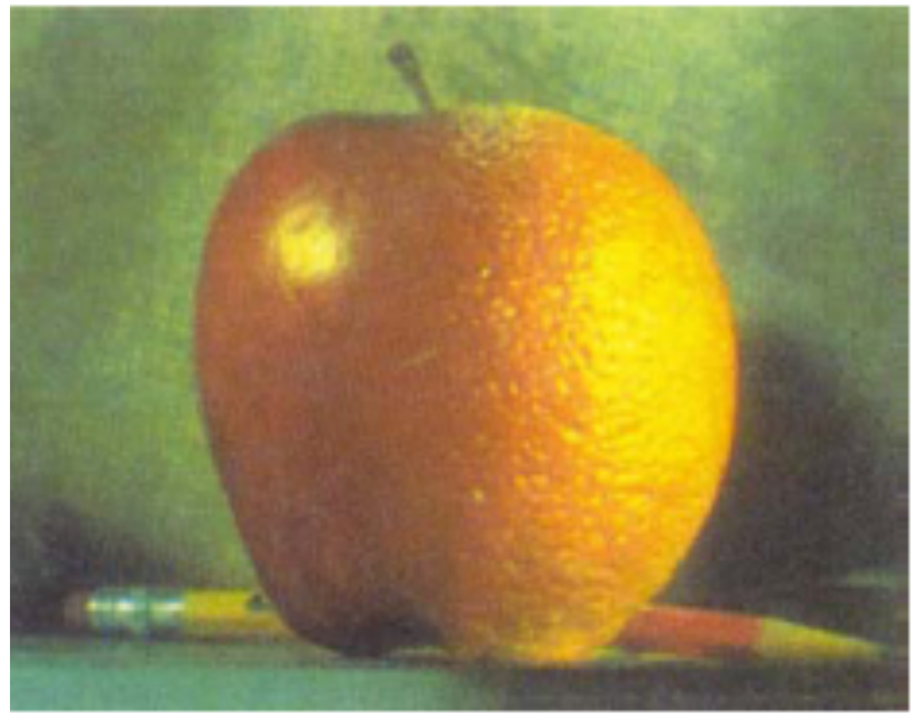
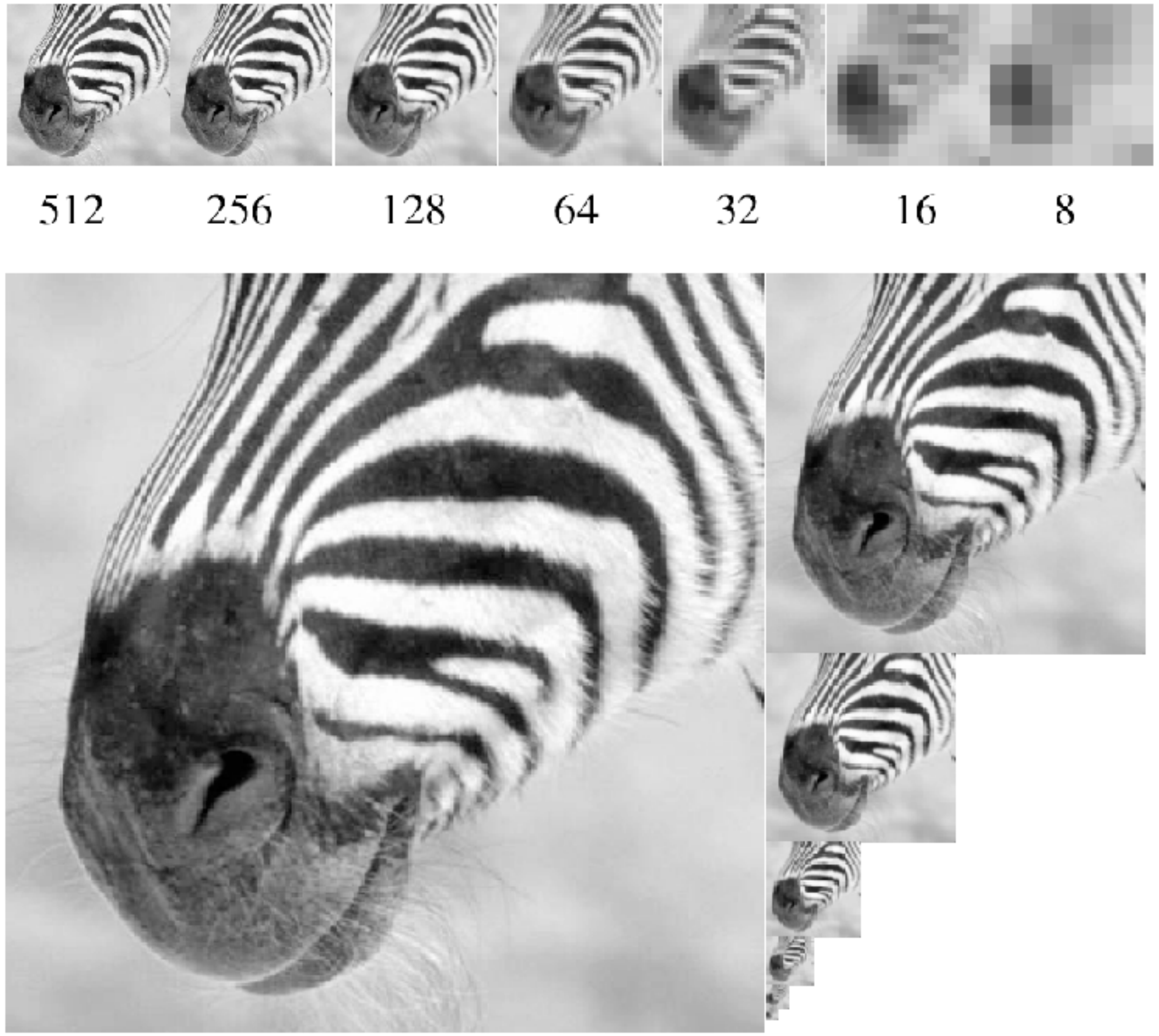
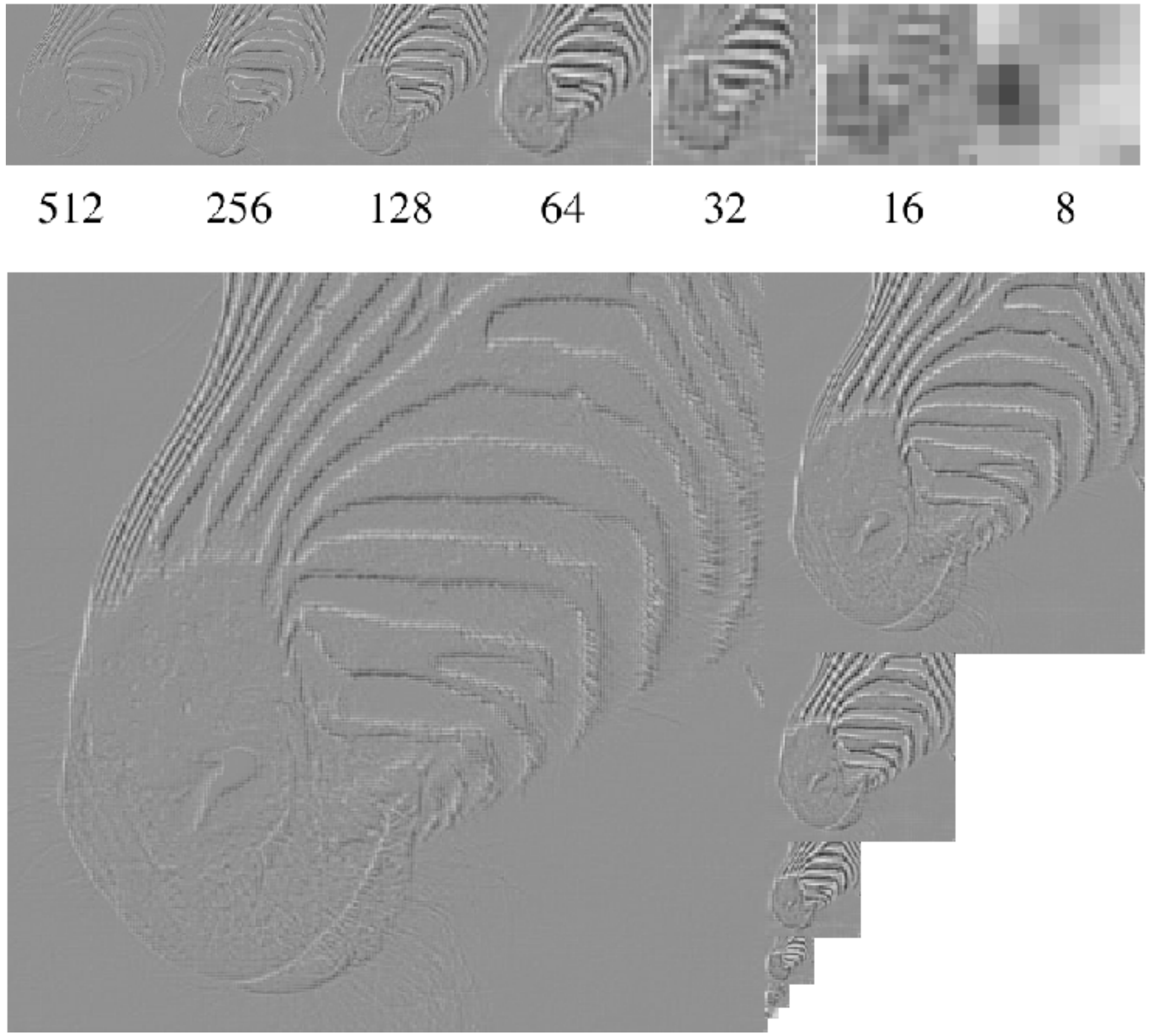


Image pyramids





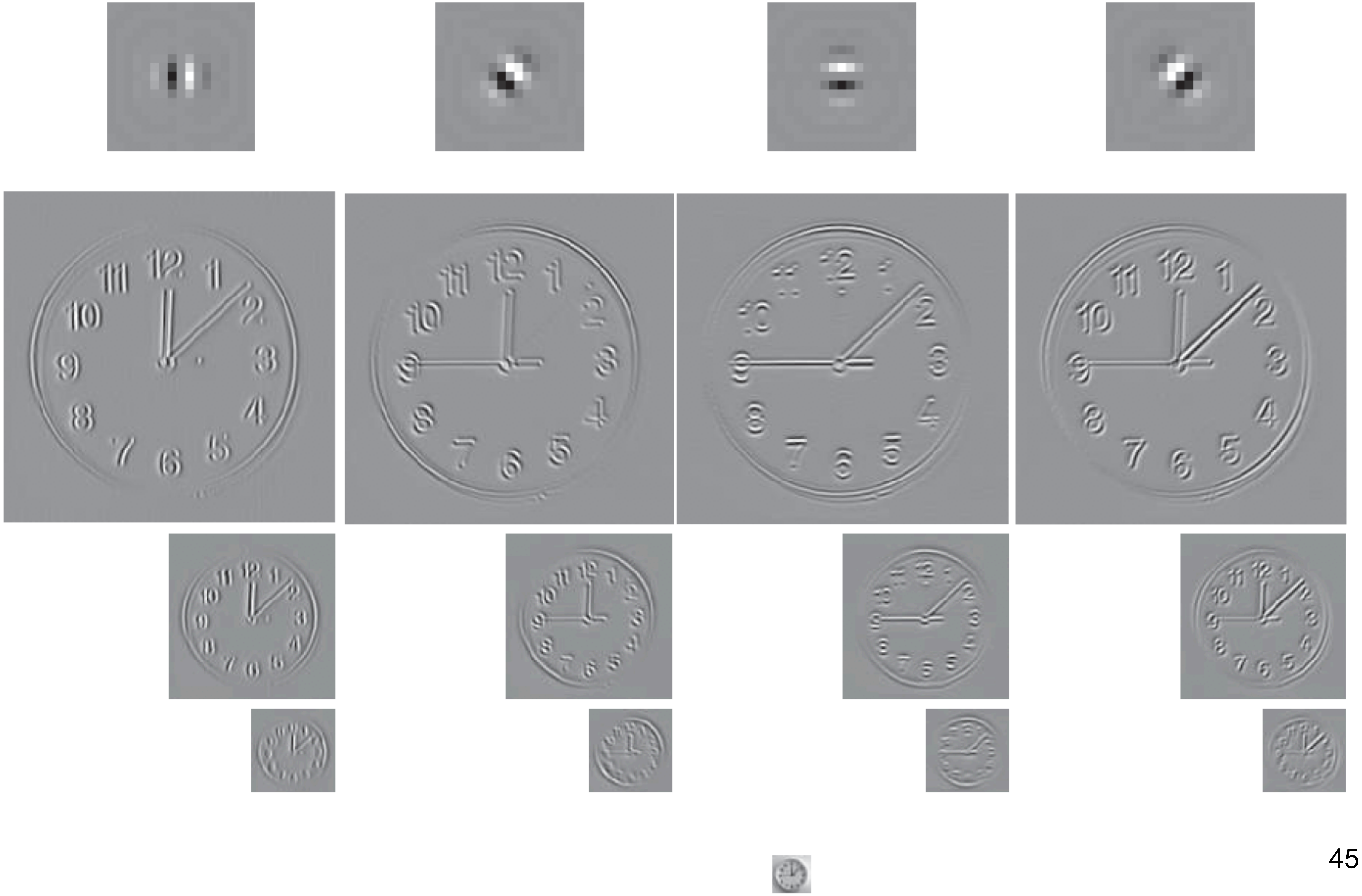
Gaussian Pyr

Laplacian Pyr

(And many more: QMF, steerable, …Convnets!)

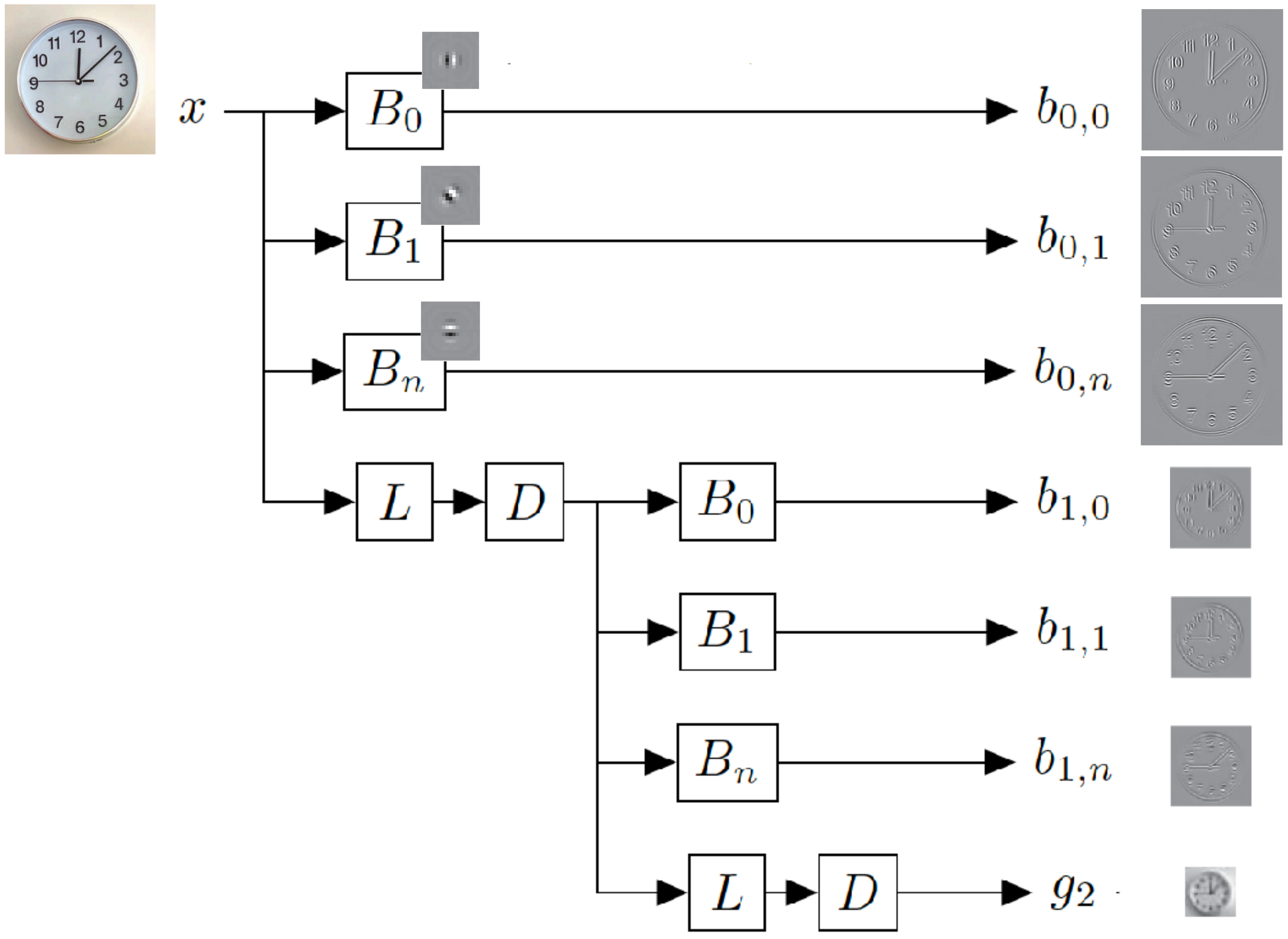
Orientations

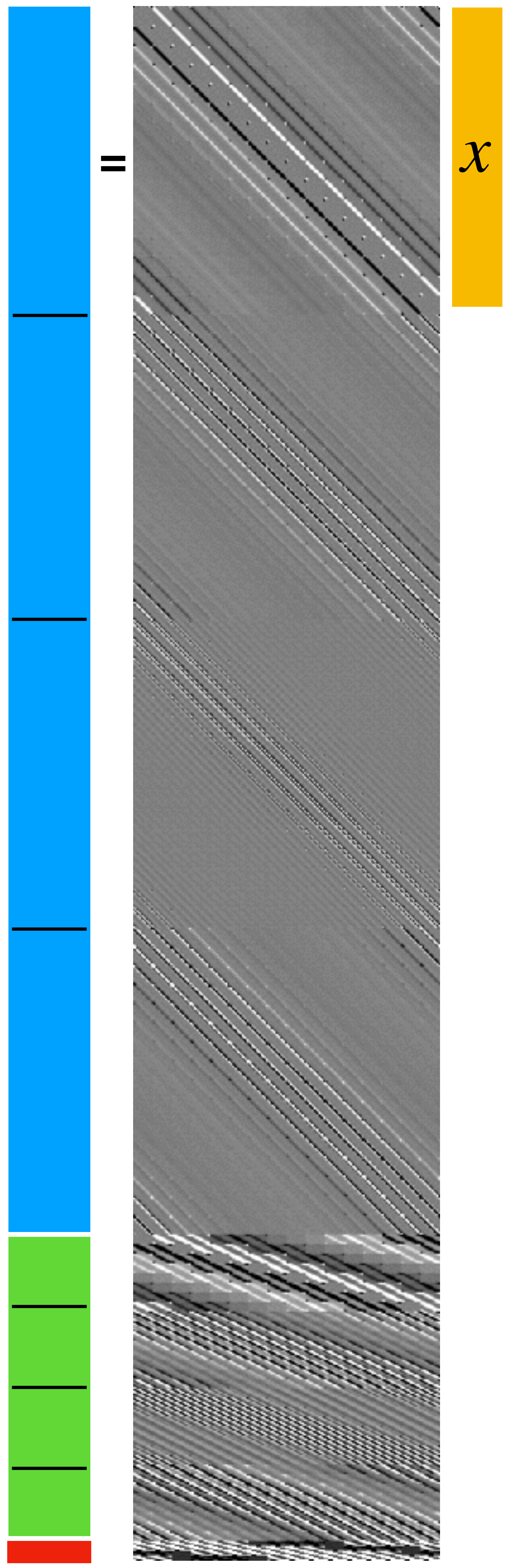
Steerable Pyramid



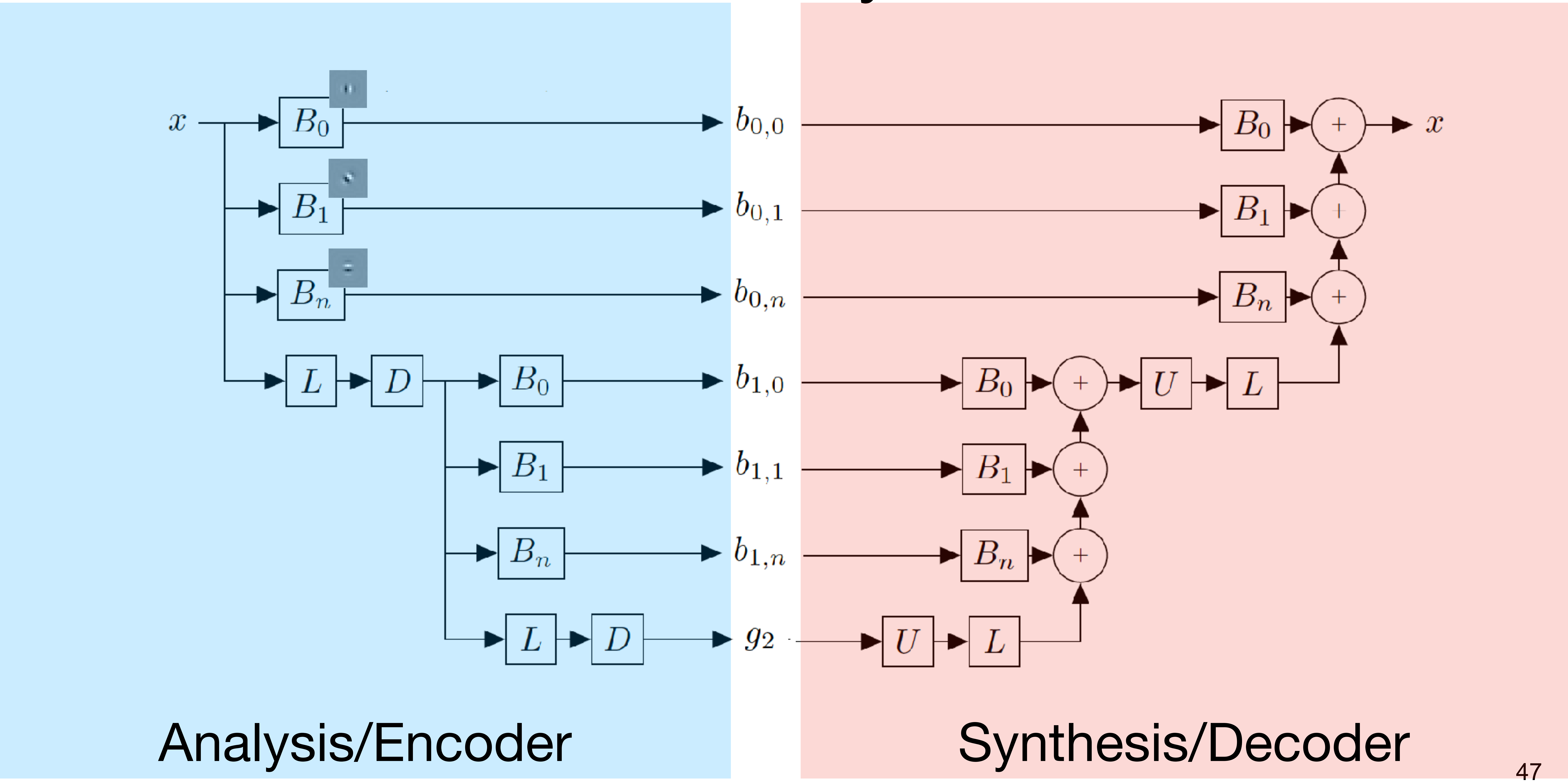
similar to Gabor filters, similar to early filters in human visual system, similar to early filters of trained neural networks: Oriented, spatially localized, multi-scale, multi-phase filter bank.

Steerable Pyramid





Steerable Pyramid



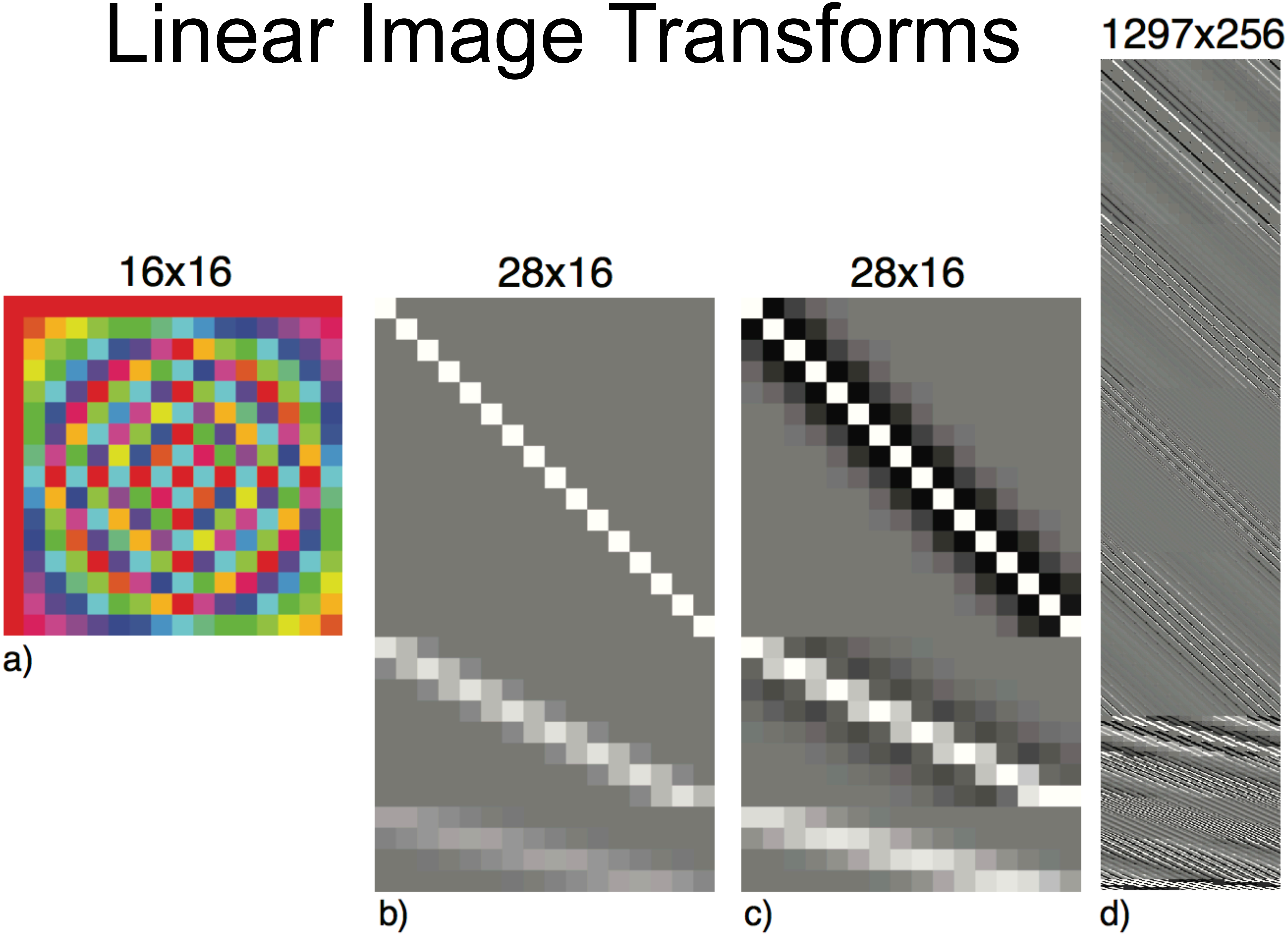
Analysis/Encoder

Synthesis/Decoder

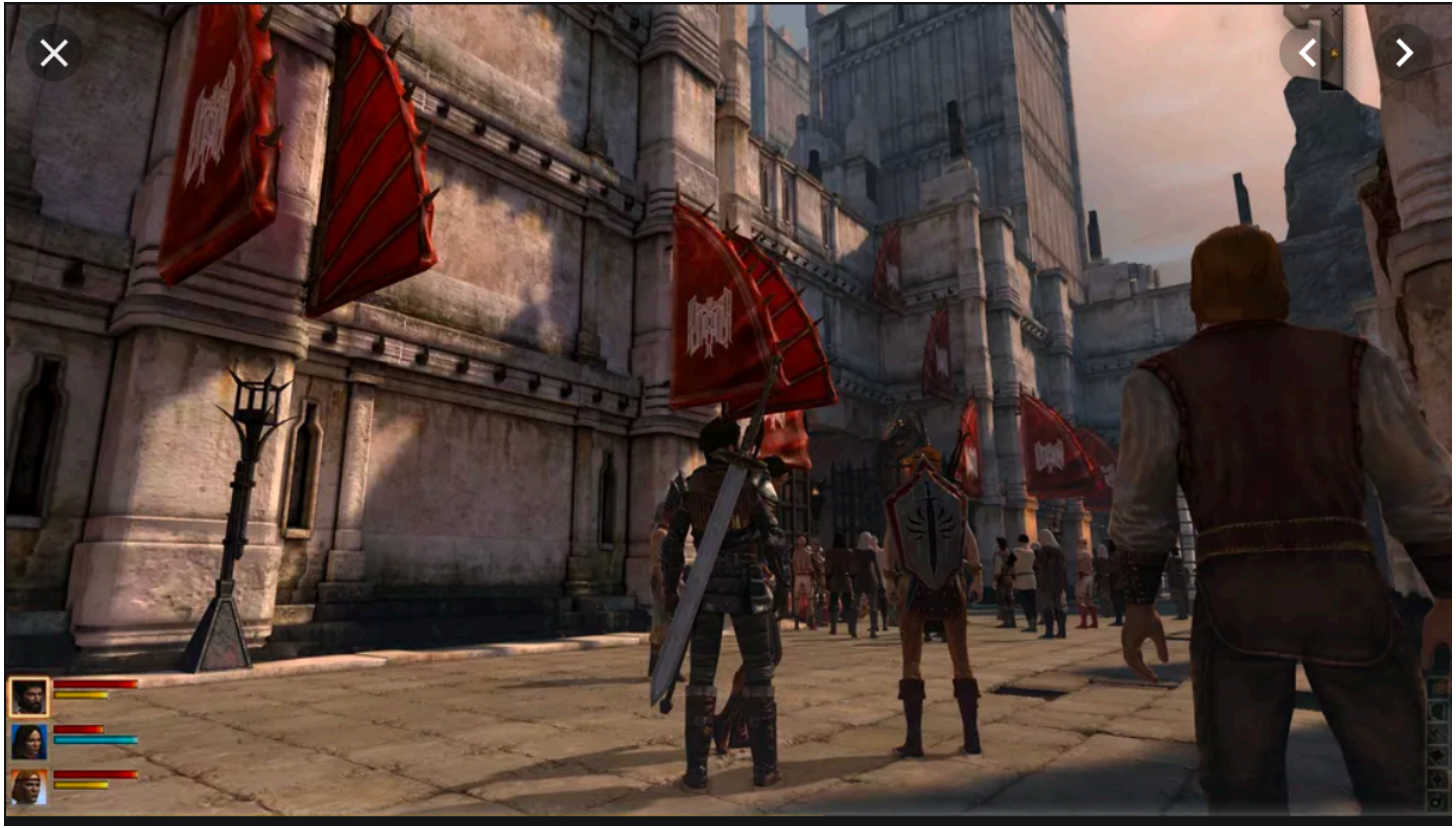
Steerable pyramid applications

- Texture synthesis
- Noise removal
- Motion analysis
- Motion synthesis, motion magnification

Linear Image Transforms



Making textures



La Bernardine

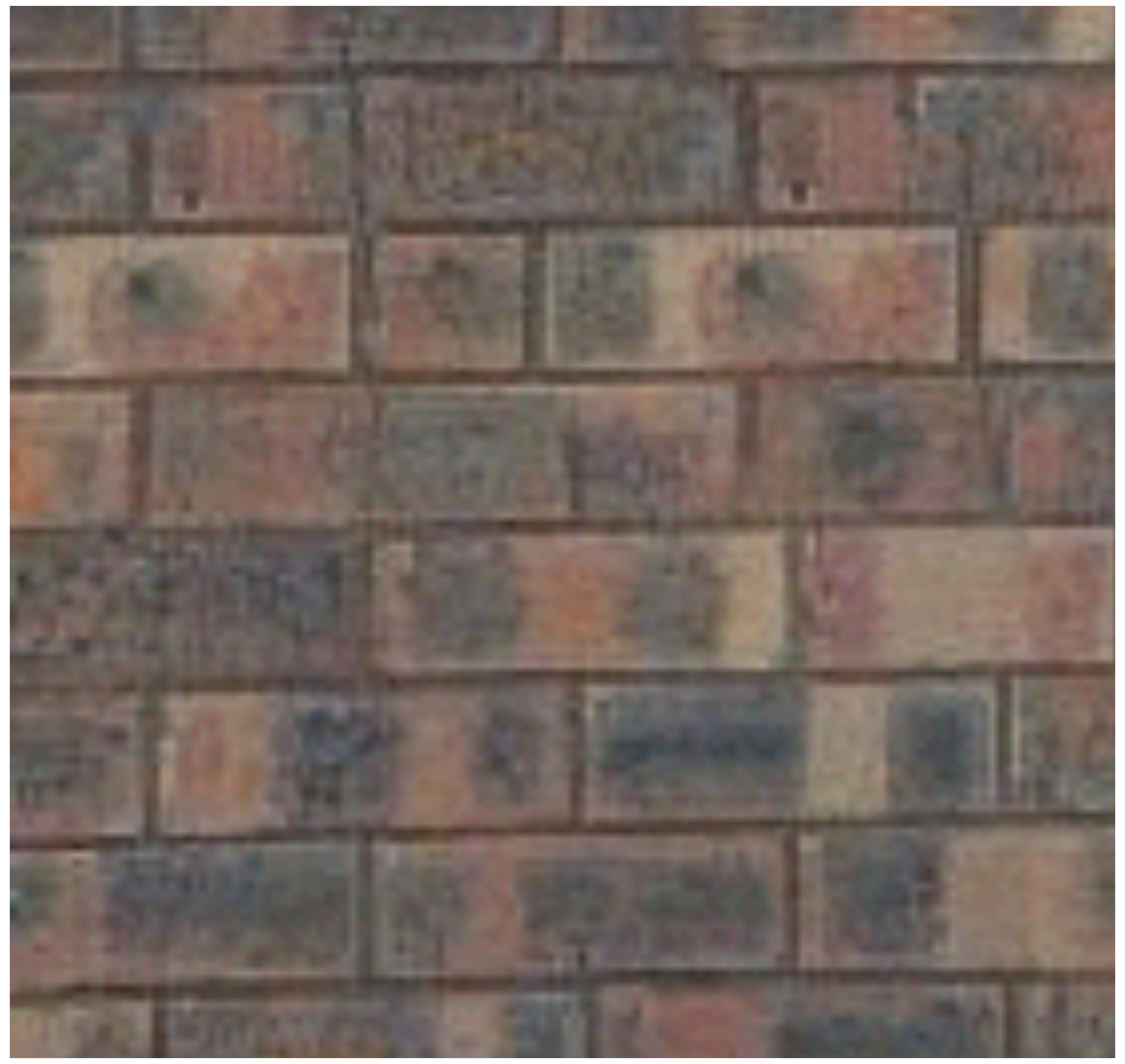
SPICY BAR-B-Q SAUCE
HOT
Original BAR-B-Q SAUCE

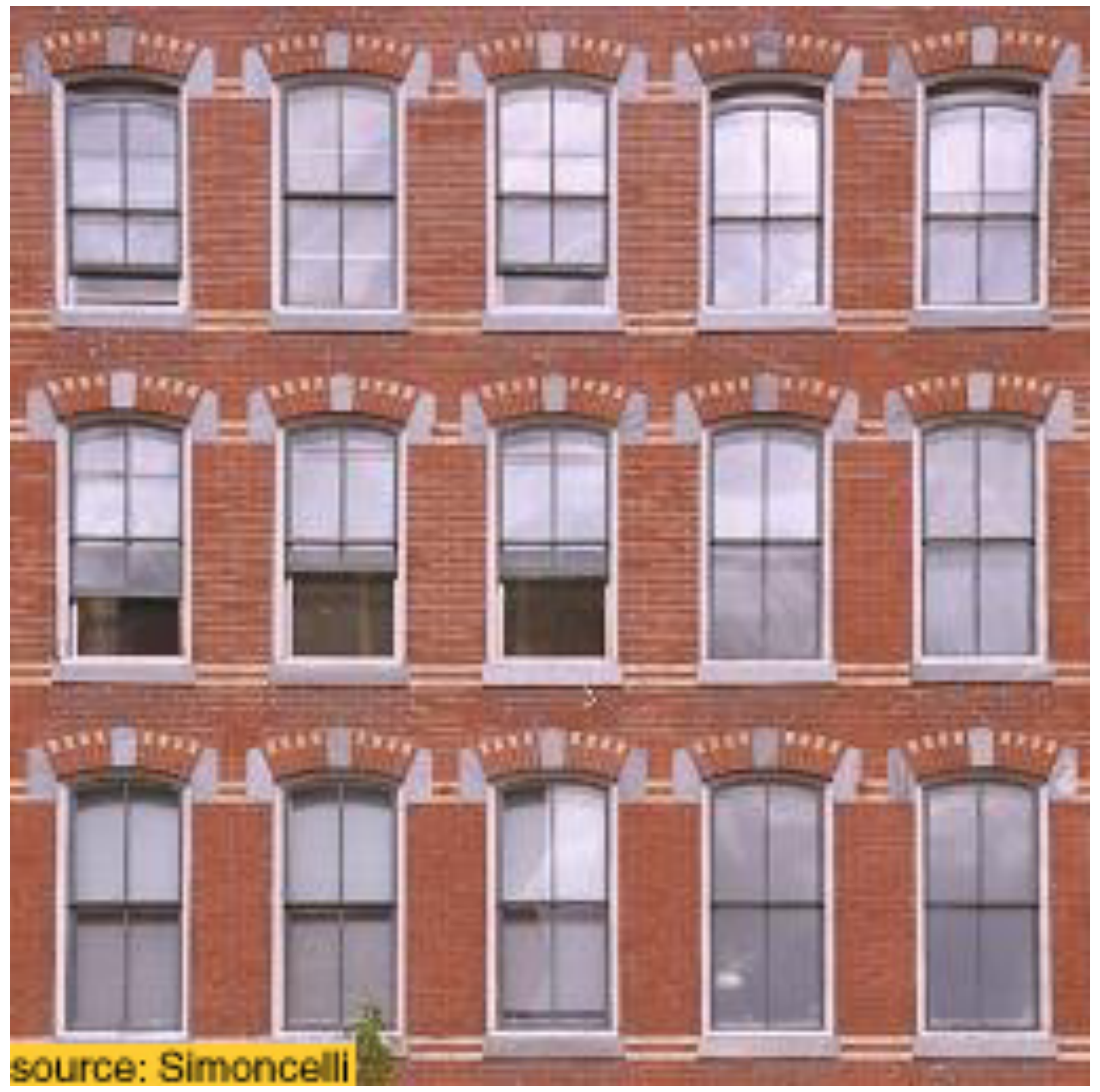
Coca-Cola

Examples of Textures

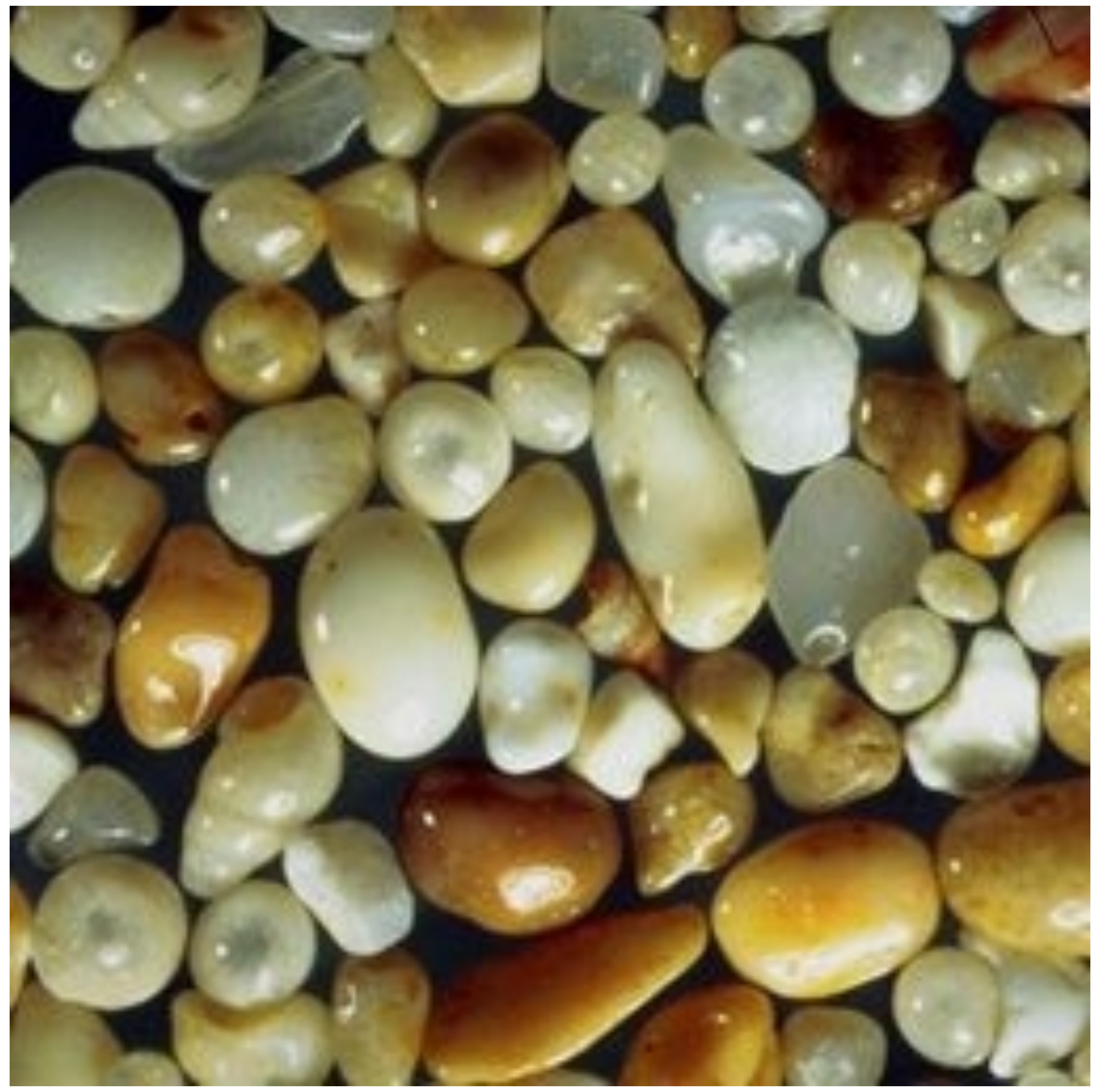
Stationary

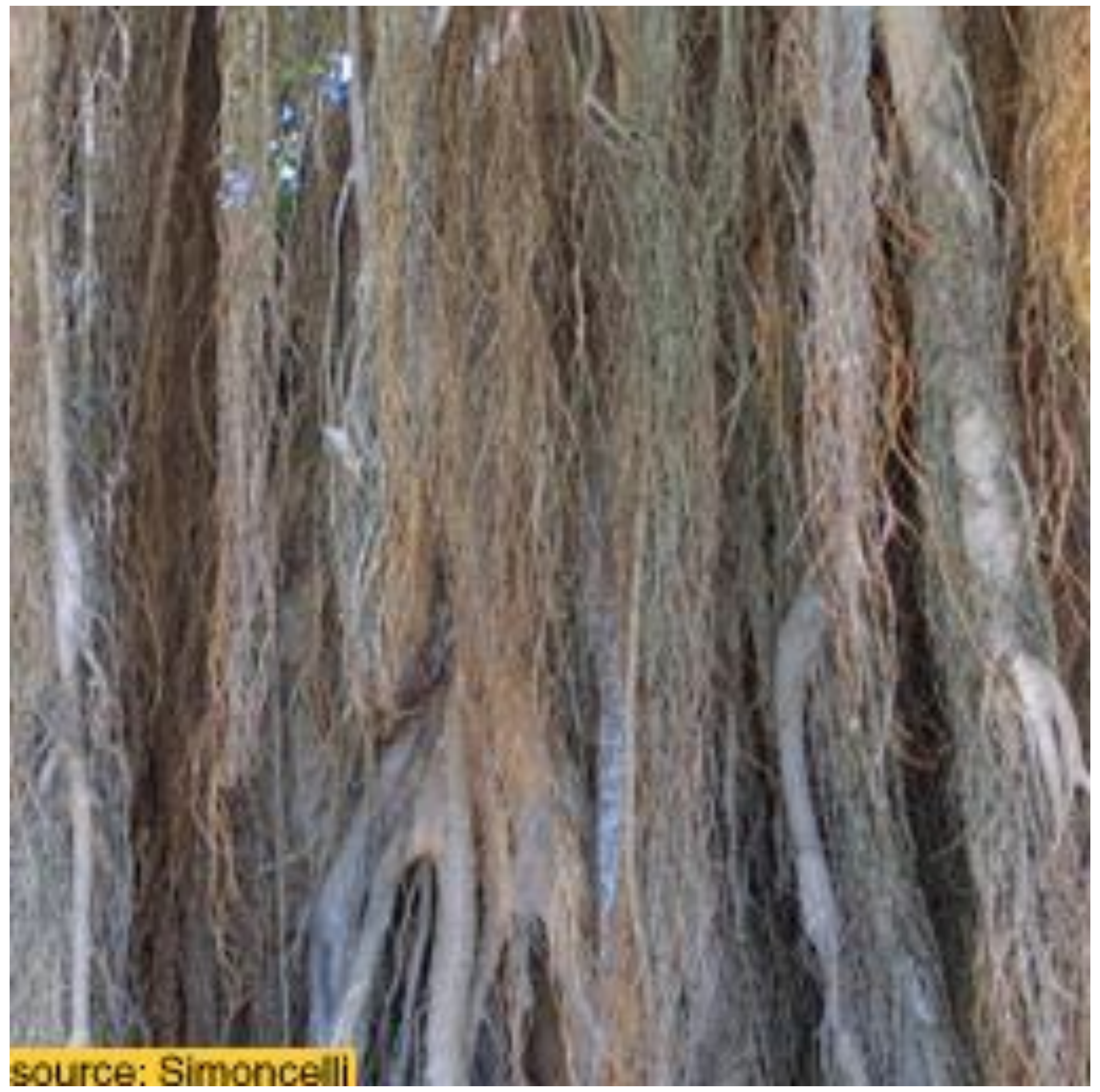
Stochastic



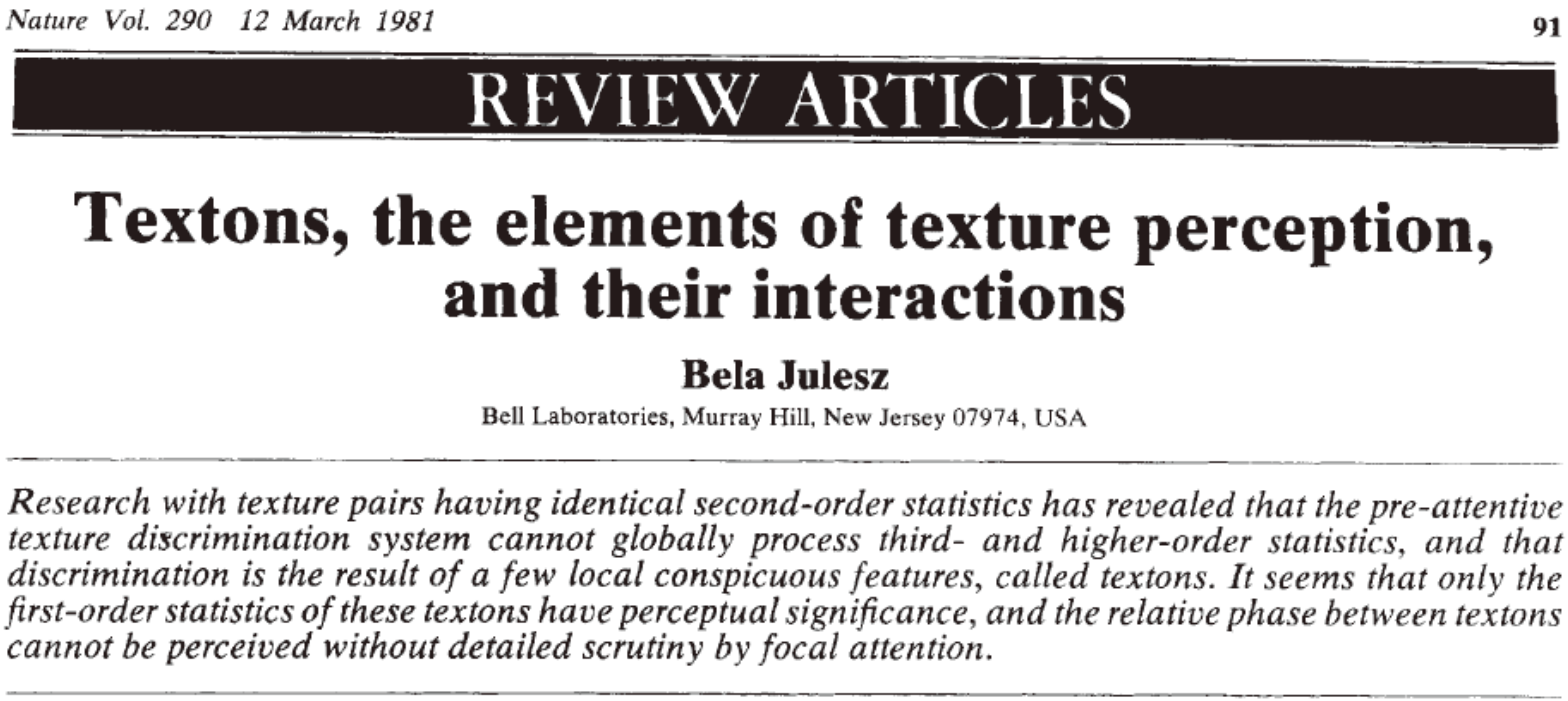


source: Simoncelli





source: Simoncelli



Nature Vol. 290 12 March 1981 91

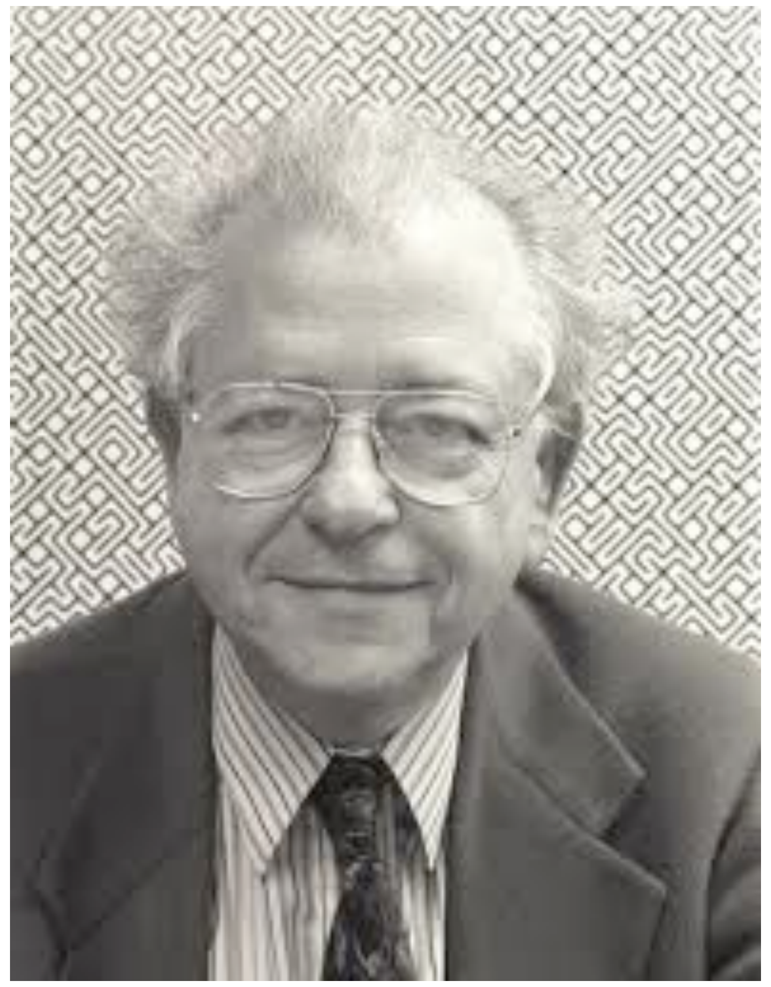
REVIEW ARTICLES

Textons, the elements of texture perception, and their interactions

Bela Julesz

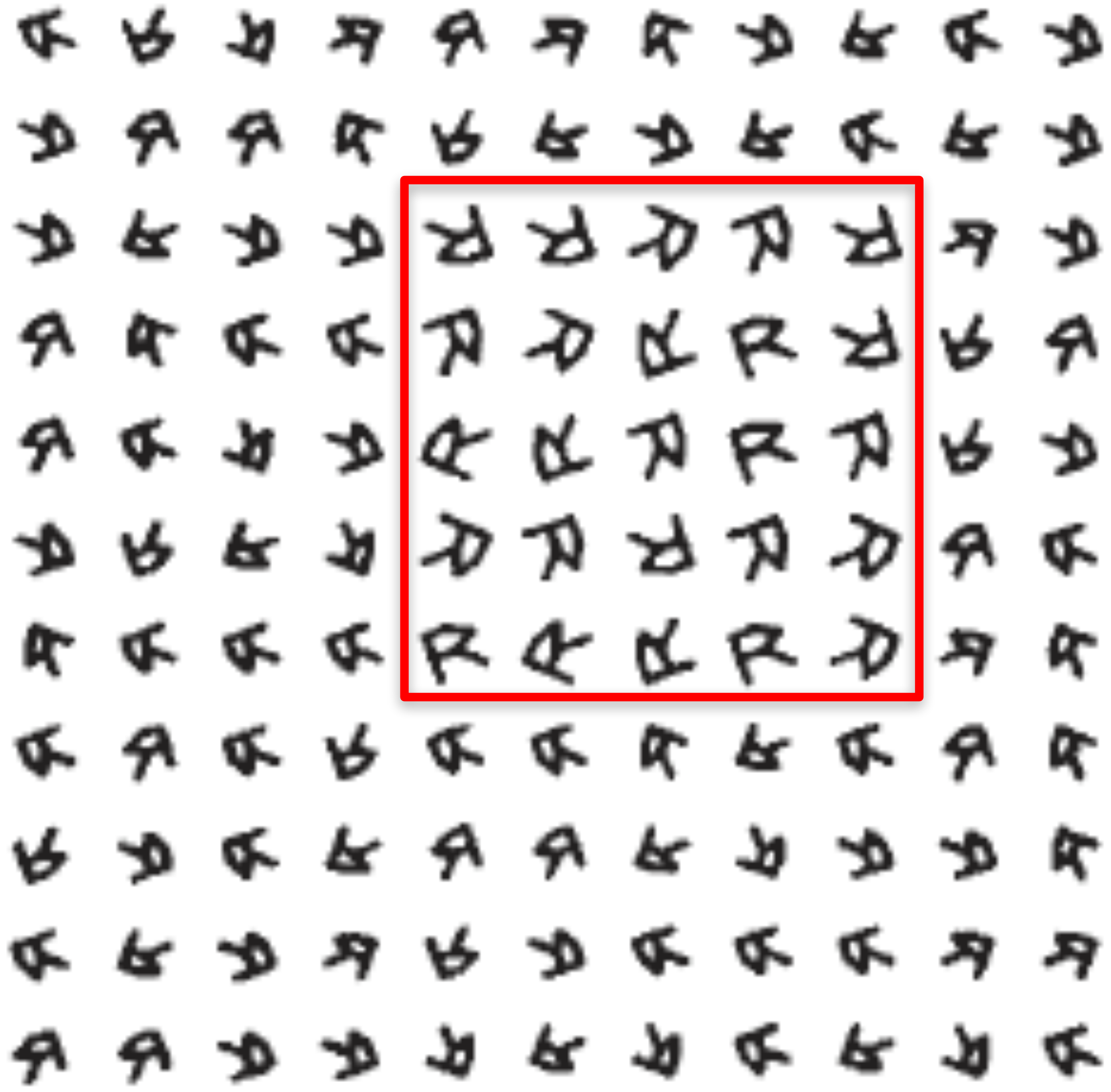
Bell Laboratories, Murray Hill, New Jersey 07974, USA

Research with texture pairs having identical second-order statistics has revealed that the pre-attentive texture discrimination system cannot globally process third- and higher-order statistics, and that discrimination is the result of a few local conspicuous features, called textons. It seems that only the first-order statistics of these textons have perceptual significance, and the relative phase between textons cannot be perceived without detailed scrutiny by focal attention.



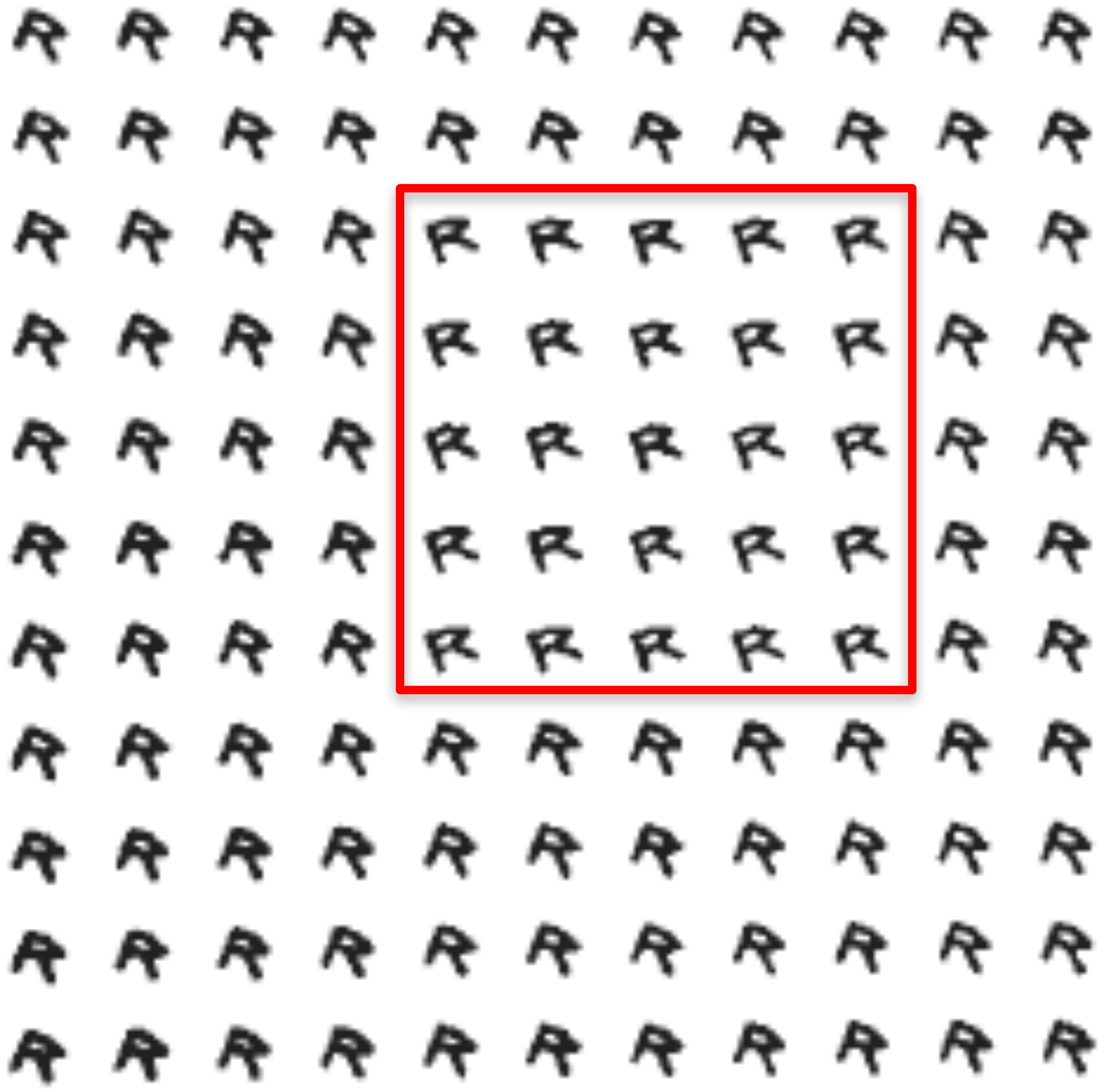
Bela Julesz, "Textons, the Elements of Texture Perception, and their Interactions". Nature 290: 91-97. March, 1981.

Pre-attentive texture discrimination



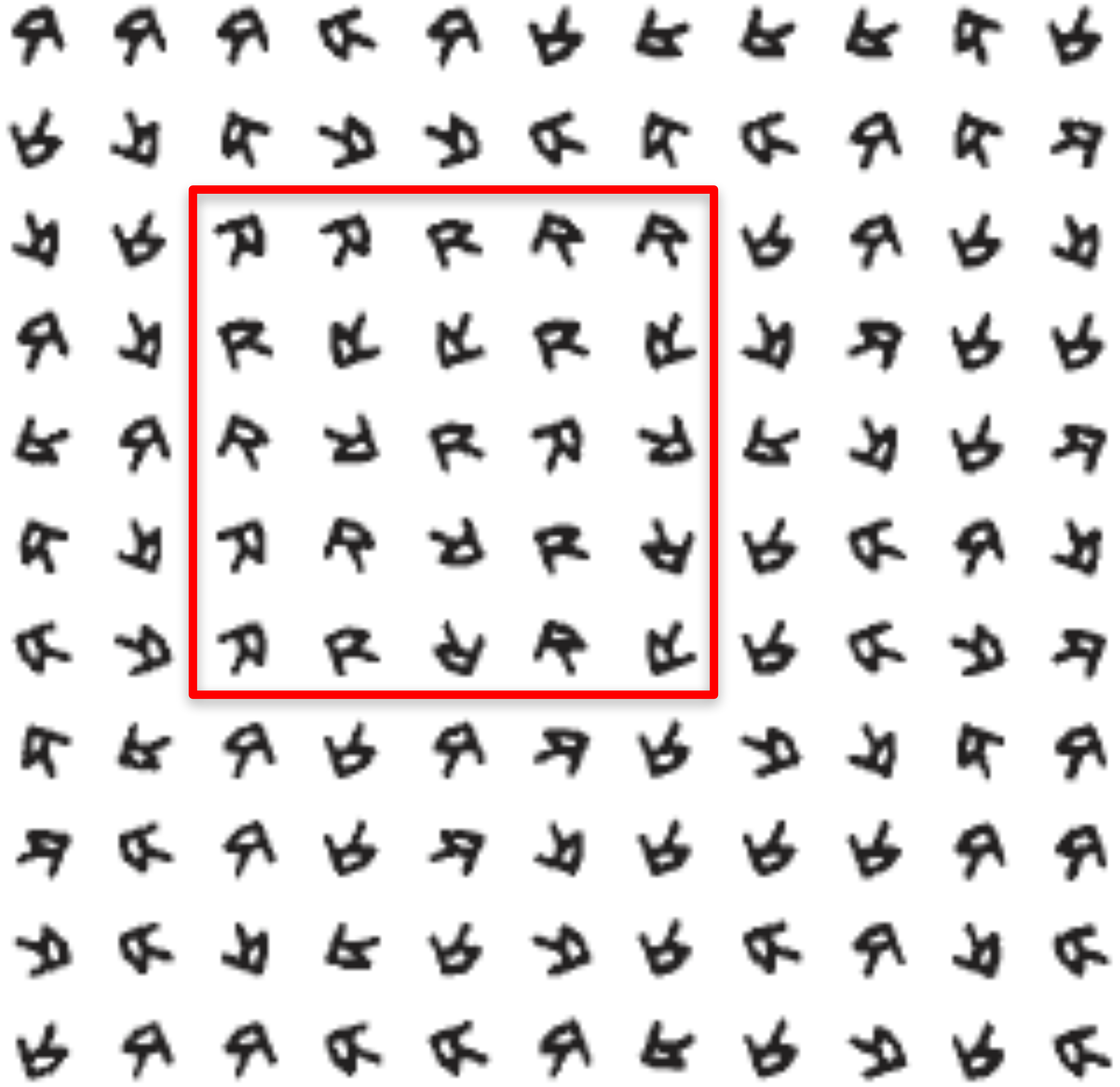
Bela Julesz, "Textons, the Elements of Texture Perception, and their Interactions". Nature 290: 91-97. March, 1981.

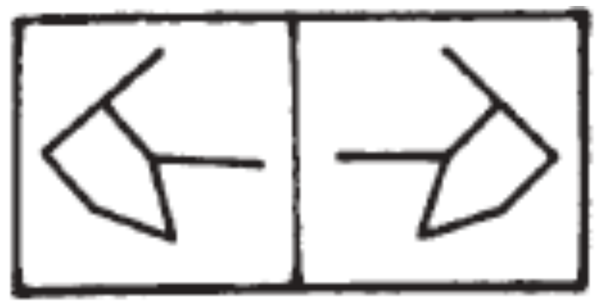
Pre-attentive texture discrimination



Bela Julesz, "Textons, the Elements of Texture Perception, and their Interactions". Nature 290: 91-97. March, 1981.

Pre-attentive texture discrimination





This texture pair is pre-attentively indistinguishable. Why?

Bela Julesz, "Textons, the Elements of Texture Perception, and their Interactions". Nature 290: 91-97. March, 1981.

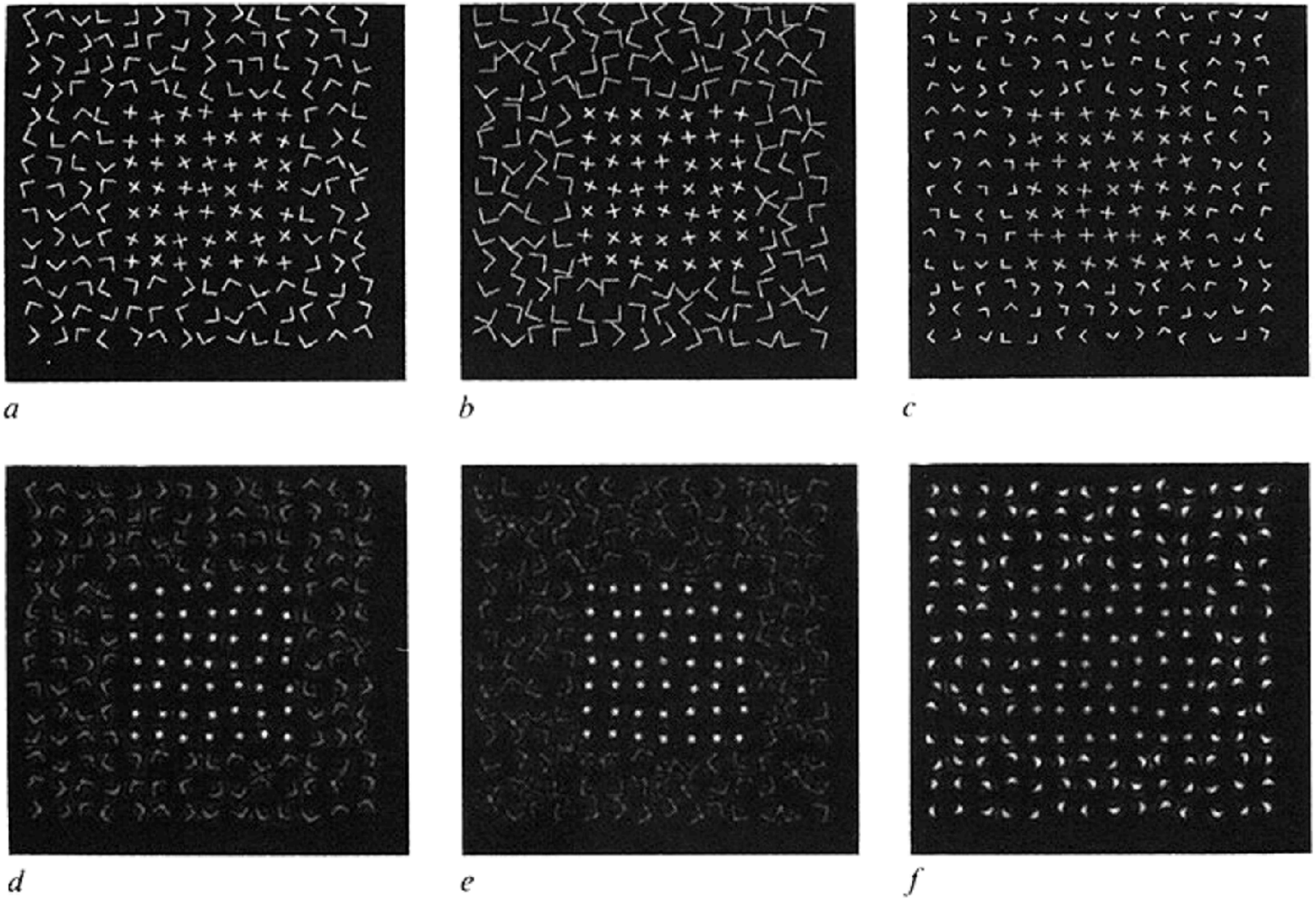


Fig. 1 *Top row,* Textures consisting of Xs within a texture composed of Ls. The micropatterns are placed at random orientations on a randomly perturbed lattice. *a*, The bars of the Xs have the same length as the bars of the Ls. *b*, The bars of the Ls have been lengthened by 25%, and the intensity adjusted for the same mean luminance. Discriminabitity is enhanced. *c*, The bars of the Ls have been shortened by 25%, and the intensity adjusted for the same mean luminance. Discriminabitity is impaired. *Bottom row*: the responses of a size-tuned mechanism *d*, response to image *a*; *e*, response to image *b*; *f*; response to image *c*.

Early vision and texture perception

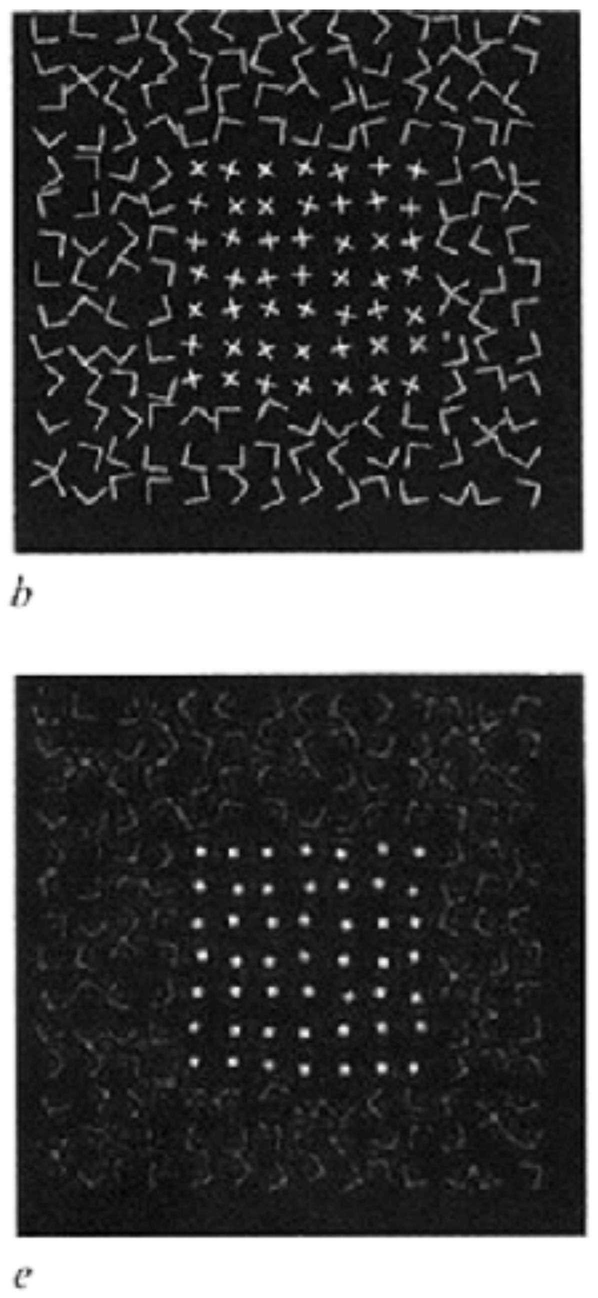
James R. Bergen* & Edward H. Adelson**

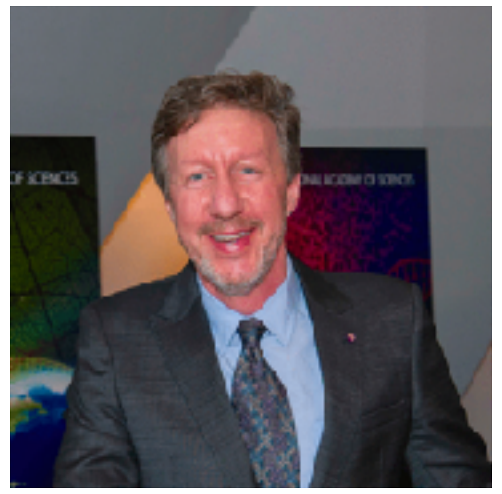
* SRI David Sarnoff Research Center, Princeton,
New Jersey 08540, USA
** Media Lab and Department of Brain and Cognitive Science,
Massachusetts Institute of Technology, Cambridge,
Massachusetts 02139, USA

Reprinted from Nature, Vol. 333. No. 6171. pp. 363-364, 26 May 1988

Jim Bergen’s conjecture

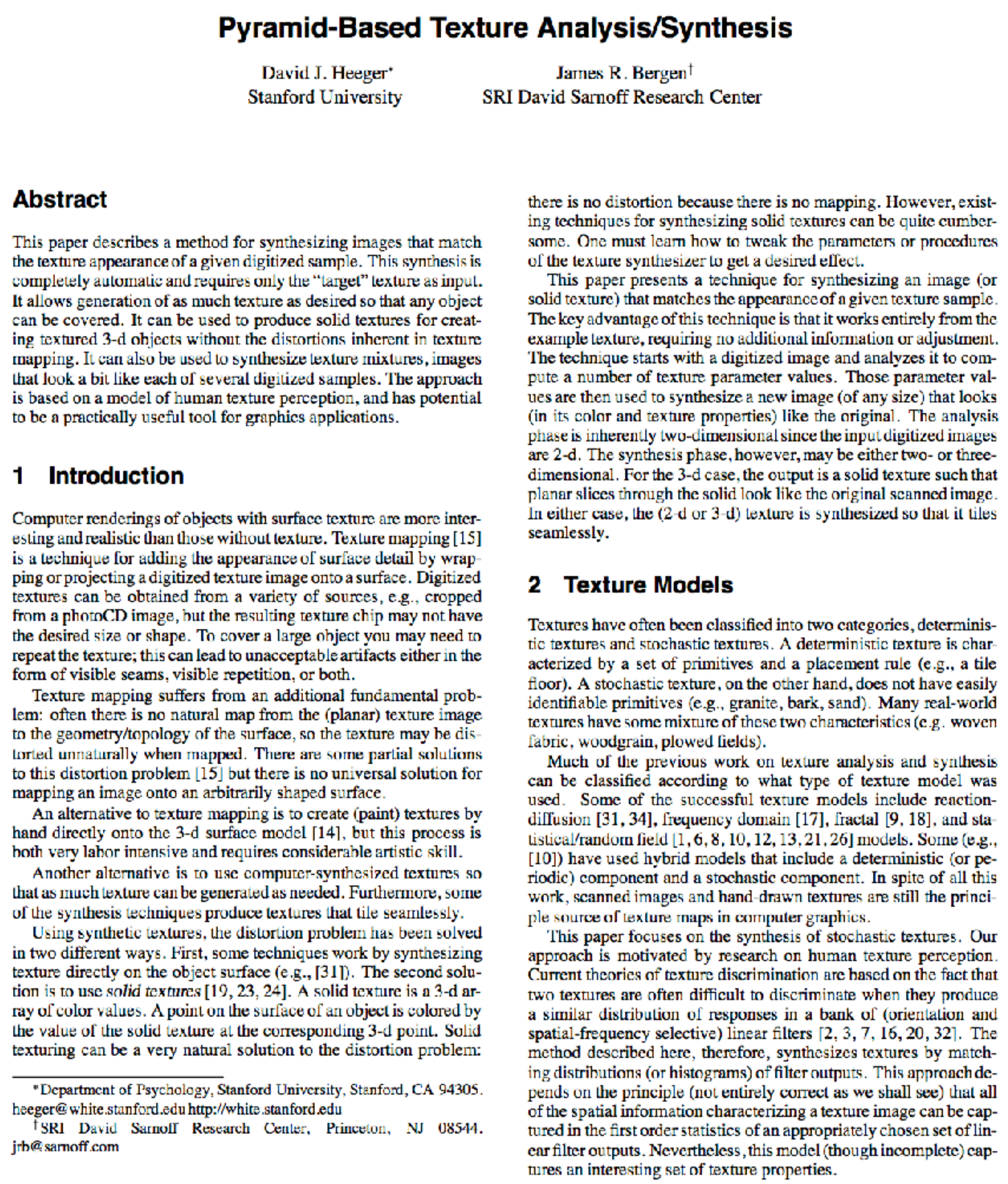
“If matching the mean amplitude of a bandpass spatial filter’s response goes a little way towards mimicking human texture perception, then maybe matching the histogram of the responses of filters (the marginal statistics) will do an even better job of capturing human texture perception.”





Dave Heeger’s response

“You mean if I took a steerable pyramid of some random noise, and forced the histograms of each subband level to match those of some target texture, that the modified noise image would then look like that texture??? No way! I’ll prove it to you; here, let me try it out… hmm, gee, that worked pretty well…”



Pyramid-Based Texture Analysis/Synthesis

David J. Heeger[*]
Stanford University

James R. Bergen[†]
SRI David Sarnoff Research Center

Abstract

This paper describes a method for synthesizing images that match the texture appearance of a given digitized sample. This synthesis is completely automatic and requires only the "target" texture as input. It allows generation of as much texture as desired so that any object can be covered. It can be used to produce solid textures for creating textured 3-d objects without the distortions inherent in texture mapping. It can also be used to synthesize texture mixtures, images that look a bit like each of several digitized samples. The approach is based on a model of human texture perception, and has potential to be a practically useful tool for graphics applications.

1 Introduction

Computer renderings of objects with surface texture are more interesting and realistic than those without texture. Texture mapping [15] is a technique for adding the appearance of surface detail by wrapping or projecting a digitized texture image onto a surface. Digitized textures can be obtained from a variety of sources, e.g., cropped from a photoCD image, but the resulting texture chip may not have the desired size or shape. To cover a large object you may need to repeat the texture; this can lead to unacceptable artifacts either in the form of visible seams, visible repetition, or both.

Texture mapping suffers from an additional fundamental problem: often there is no natural map from the (planar) texture image to the geometry/topology of the surface, so the texture may be distorted unnaturally when mapped. There are some partial solutions to this distortion problem [15] but there is no universal solution for mapping an image onto an arbitrarily shaped surface.

An alternative to texture mapping is to create (paint) textures by hand directly onto the 3-d surface model [14], but this process is both very labor intensive and requires considerable artistic skill.

Another alternative is to use computer-synthesized textures so that as much texture can be generated as needed. Furthermore, some of the synthesis techniques produce textures that tile seamlessly.

Using synthetic textures, the distortion problem has been solved in two different ways. First, some techniques work by synthesizing texture directly on the object surface (e.g., [31]). The second solution is to use *solid textures* [19, 23, 24]. A solid texture is a 3-d array of color values. A point on the surface of an object is colored by the value of the solid texture at the corresponding 3-d point. Solid texturing can be a very natural solution to the distortion problem: there is no distortion because there is no mapping. However, existing techniques for synthesizing solid textures can be quite cumbersome. One must learn how to tweak the parameters or procedures of the texture synthesizer to get a desired effect.

This paper presents a technique for synthesizing an image (or solid texture) that matches the appearance of a given texture sample. The key advantage of this technique is that it works entirely from the example texture, requiring no additional information or adjustment. The technique starts with a digitized image and analyzes it to compute a number of texture parameter values. Those parameter values are then used to synthesize a new image (of any size) that looks (in its color and texture properties) like the original. The analysis phase is inherently two-dimensional since the input digitized images are 2-d. The synthesis phase, however, may be either two- or three-dimensional. For the 3-d case, the output is a solid texture such that planar slices through the solid look like the original scanned image. In either case, the (2-d or 3-d) texture is synthesized so that it tiles seamlessly.

2 Texture Models

Textures have often been classified into two categories, deterministic textures and stochastic textures. A deterministic texture is characterized by a set of primitives and a placement rule (e.g., a tile floor). A stochastic texture, on the other hand, does not have easily identifiable primitives (e.g., granite, bark, sand). Many real-world textures have some mixture of these two characteristics (e.g. woven fabric, woodgrain, plowed fields).

Much of the previous work on texture analysis and synthesis can be classified according to what type of texture model was used. Some of the successful texture models include reaction-diffusion [31, 34], frequency domain [17], fractal [9, 18], and statistical/random field [1, 6, 8, 10, 12, 13, 21, 26] models. Some (e.g., [10]) have used hybrid models that include a deterministic (or periodic) component and a stochastic component. In spite of all this work, scanned images and hand-drawn textures are still the principle source of texture maps in computer graphics.

This paper focuses on the synthesis of stochastic textures. Our approach is motivated by research on human texture perception. Current theories of texture discrimination are based on the fact that two textures are often difficult to discriminate when they produce a similar distribution of responses in a bank of (orientation and spatial-frequency selective) linear filters [2, 3, 7, 16, 20, 32]. The method described here, therefore, synthesizes textures by matching distributions (or histograms) of filter outputs. This approach depends on the principle (not entirely correct as we shall see) that all of the spatial information characterizing a texture image can be captured in the first order statistics of an appropriately chosen set of linear filter outputs. Nevertheless, this model (though incomplete) captures an interesting set of texture properties.

[*] Department of Psychology, Stanford University, Stanford, CA 94305. heeger@white.stanford.edu http://white.stanford.edu

[†] SRI David Sarnoff Research Center, Princeton, NJ 08544. jrb@sarnoff.com

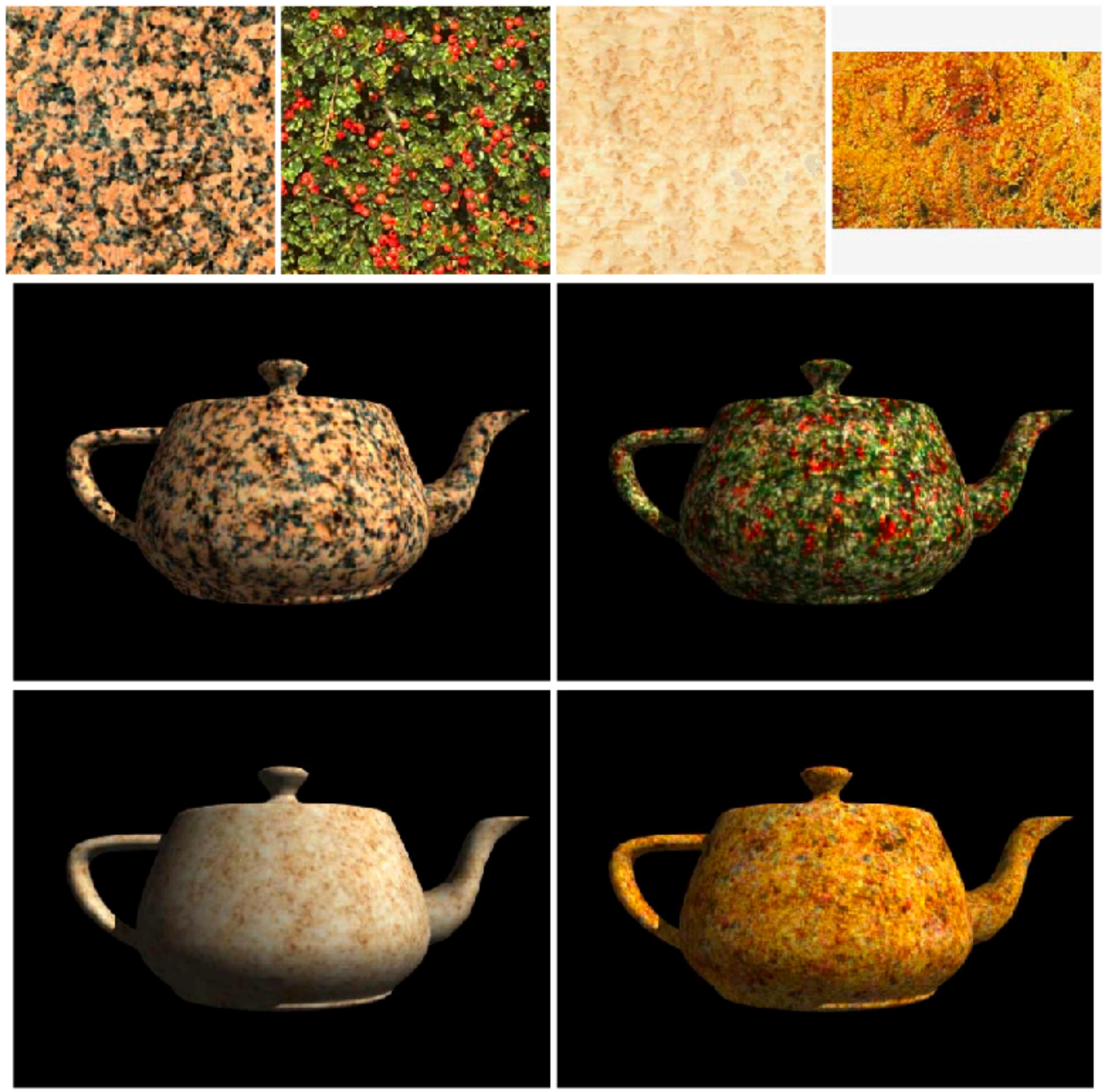
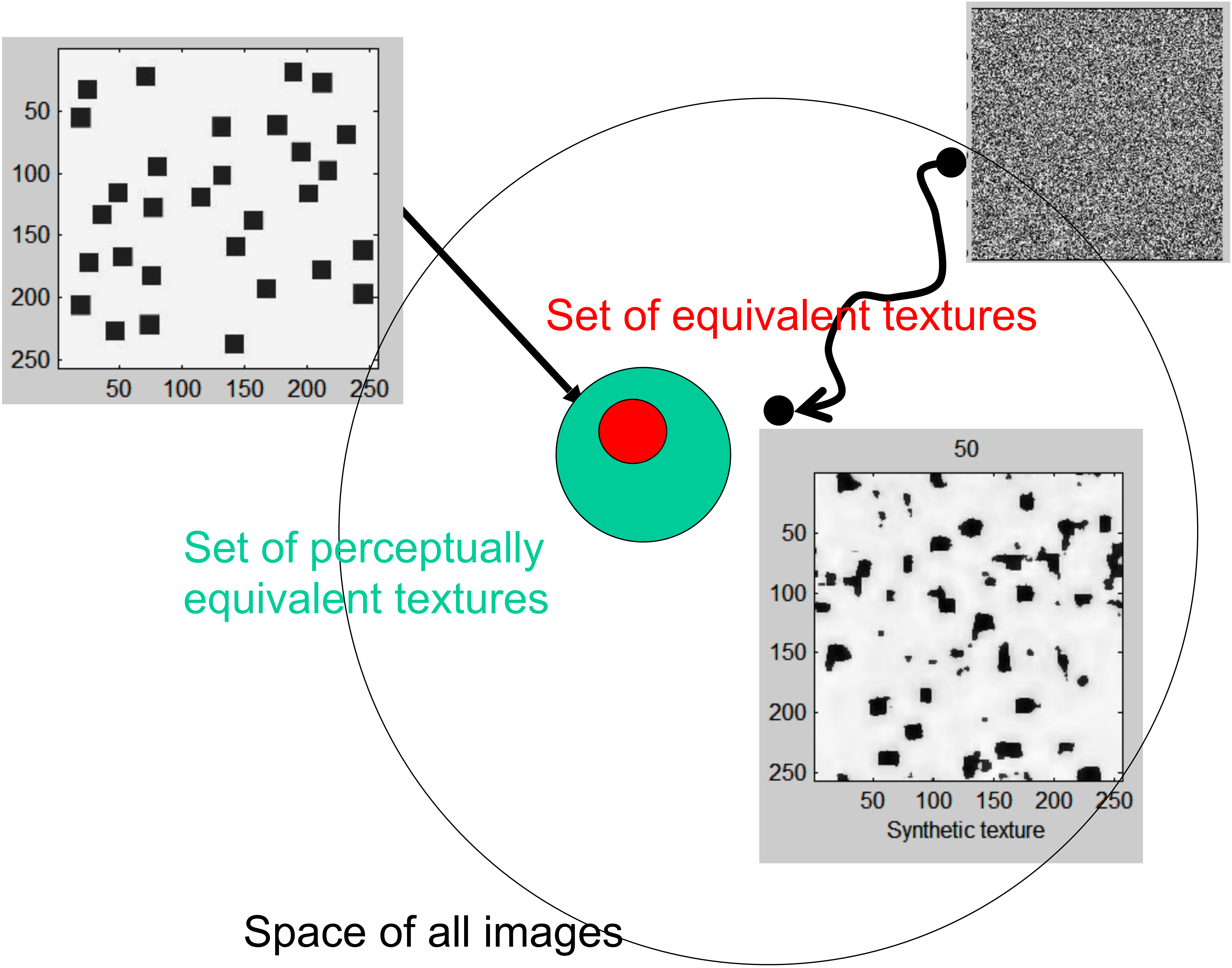


Figure 5: (Top Row) Original digitized sample textures: red granite, berry bush, figured maple, yellow coral. (Bottom Rows) Synthetic solid textured teapots.

https://www.cns.nyu.edu/heegerlab/content/publications/Heeger-siggraph95.pdf

SIGGRAPH 1995

The main idea: it works by ‘kind of’ projecting a random image into the set of equivalent textures

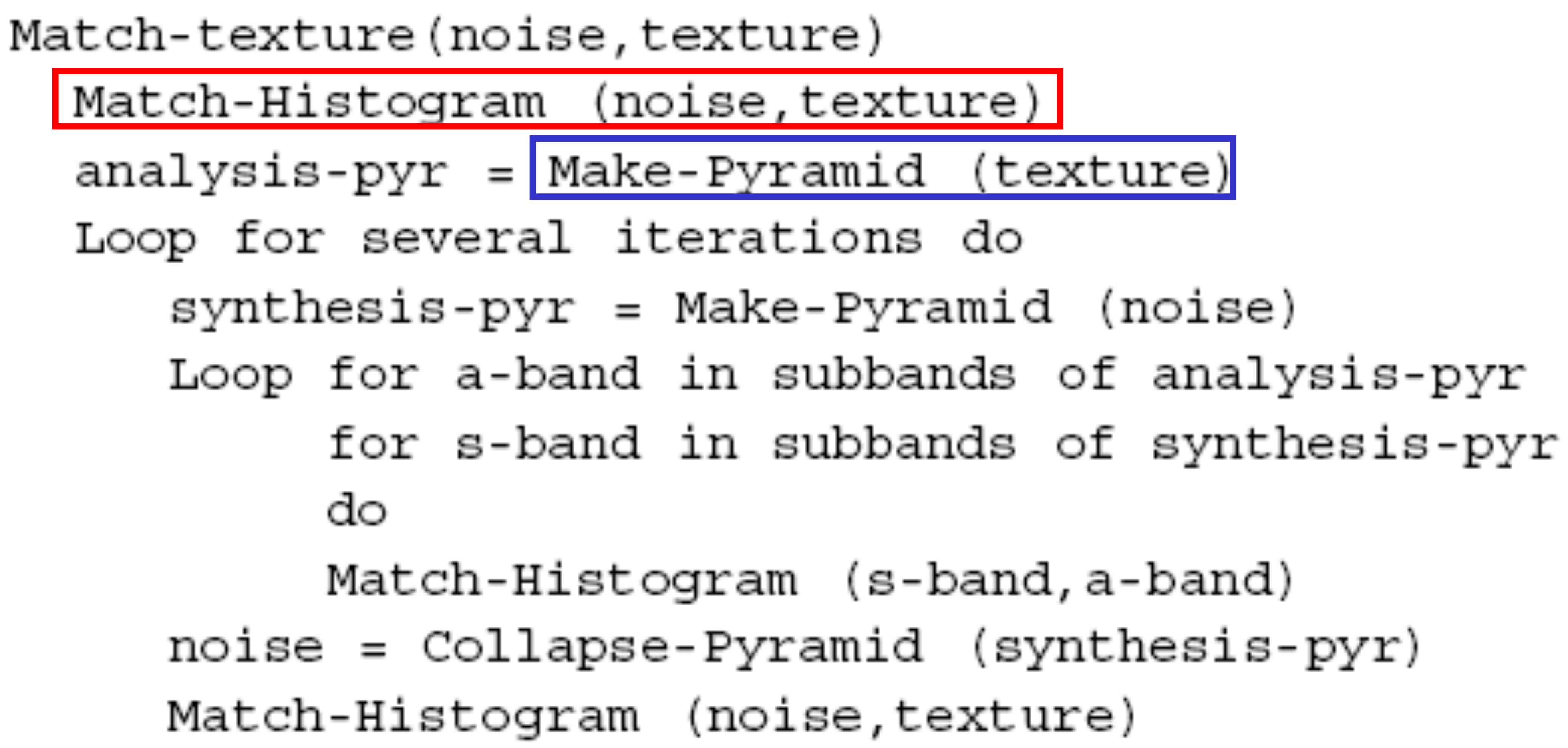


Set of equivalent textures

Set of perceptually equivalent textures

Space of all images

Overview of the algorithm



```
Match-texture(noise,texture)
  Match-Histogram (noise,texture)
  analysis-pyr = Make-Pyramid (texture)
  Loop for several iterations do
     synthesis-pyr = Make-Pyramid (noise)
     Loop for a-band in subbands of analysis-pyr
          for s-band in subbands of synthesis-pyr
          do
          Match-Histogram (s-band,a-band)
     noise = Collapse-Pyramid (synthesis-pyr)
     Match-Histogram (noise,texture)
```

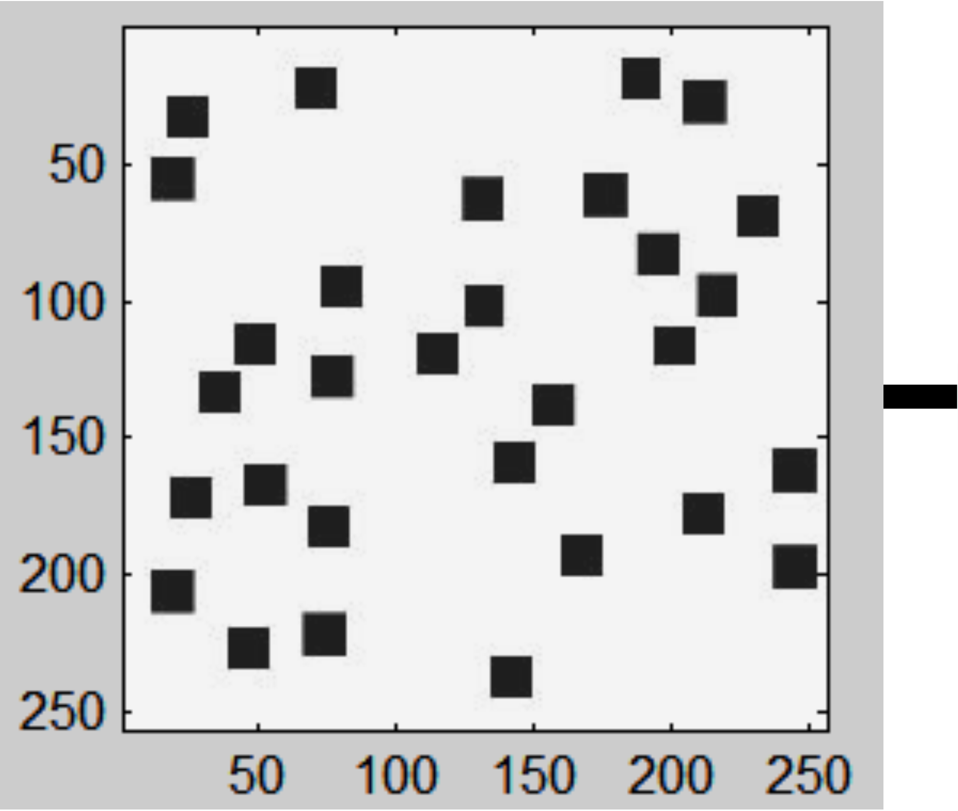
Two main tools:

1- steerable pyramid

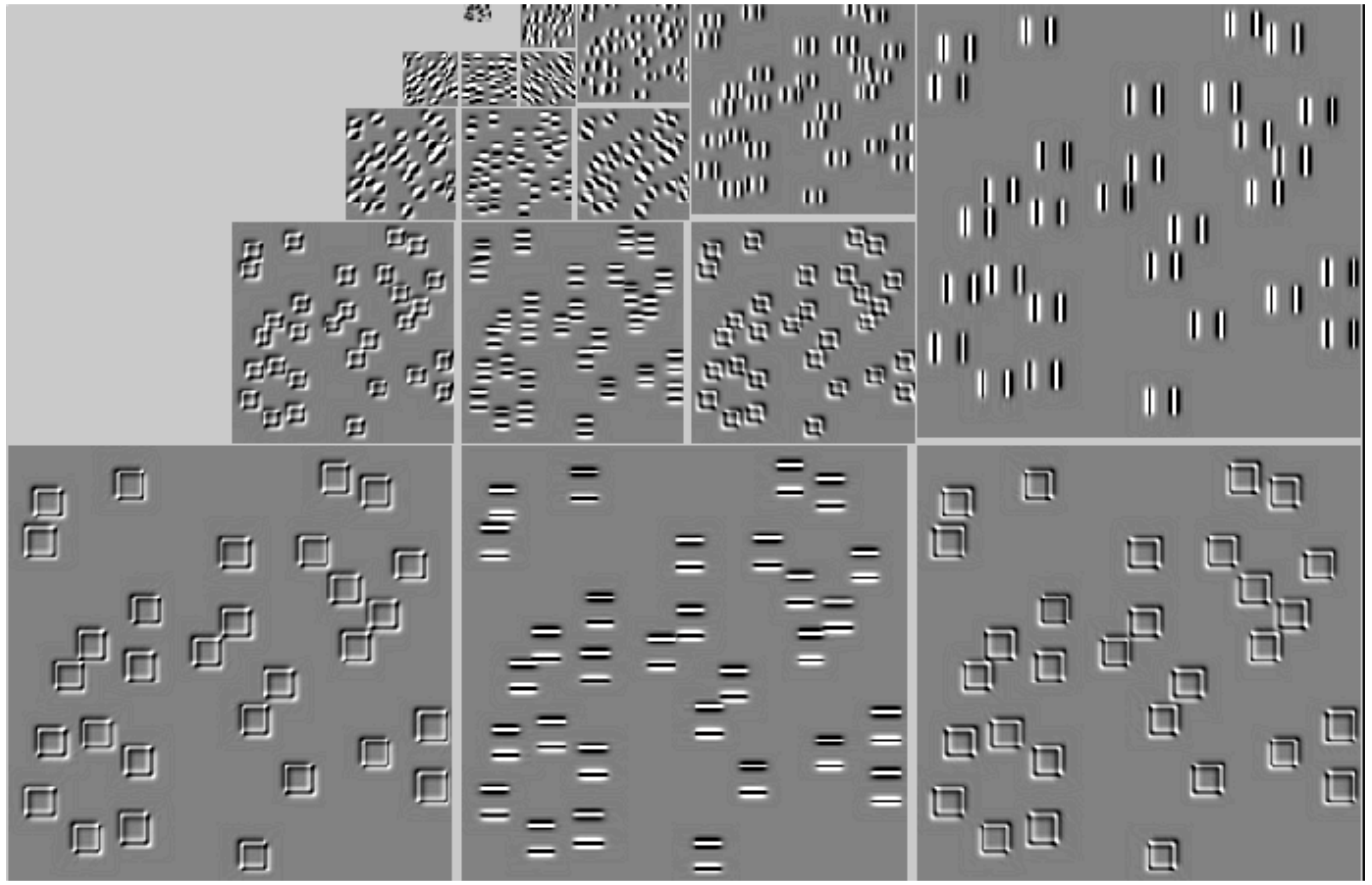
2- matching histograms

1-The steerable pyramid

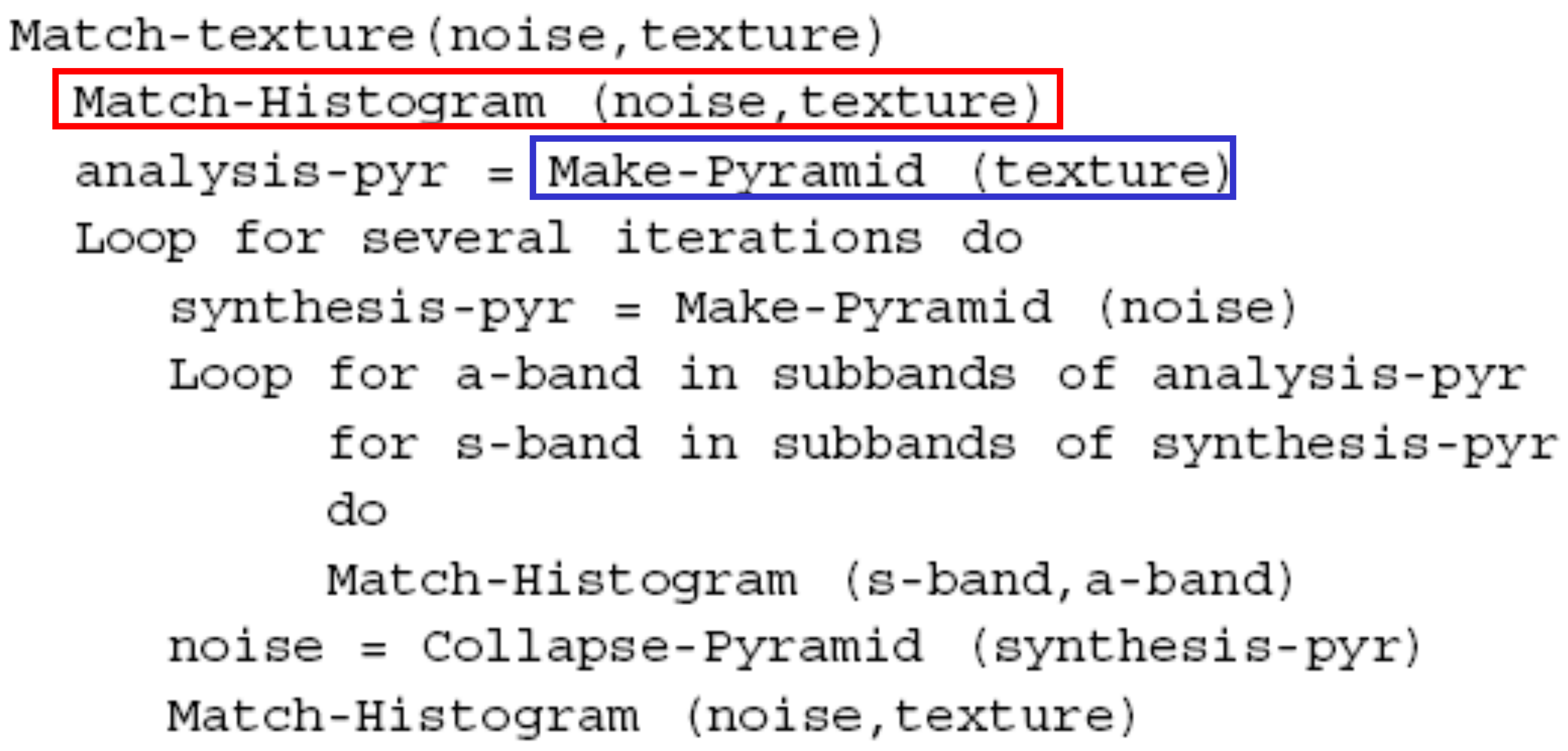
Input texture



Steerable pyr



Overview of the algorithm



```
Match-texture(noise,texture)
  Match-Histogram (noise,texture)
  analysis-pyr = Make-Pyramid (texture)
  Loop for several iterations do
     synthesis-pyr = Make-Pyramid (noise)
     Loop for a-band in subbands of analysis-pyr
          for s-band in subbands of synthesis-pyr
          do
          Match-Histogram (s-band,a-band)
     noise = Collapse-Pyramid (synthesis-pyr)
     Match-Histogram (noise,texture)
```

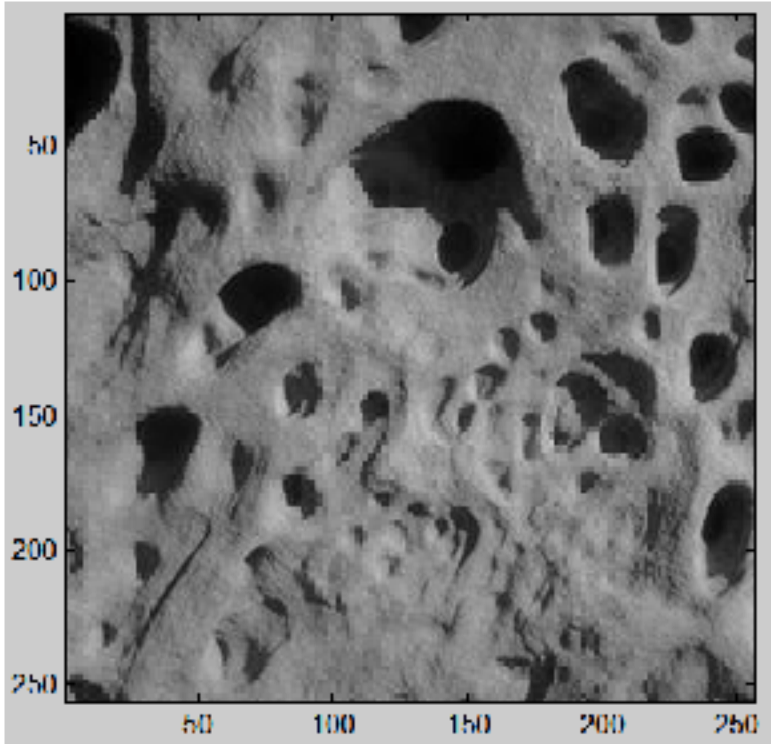
Two main tools:

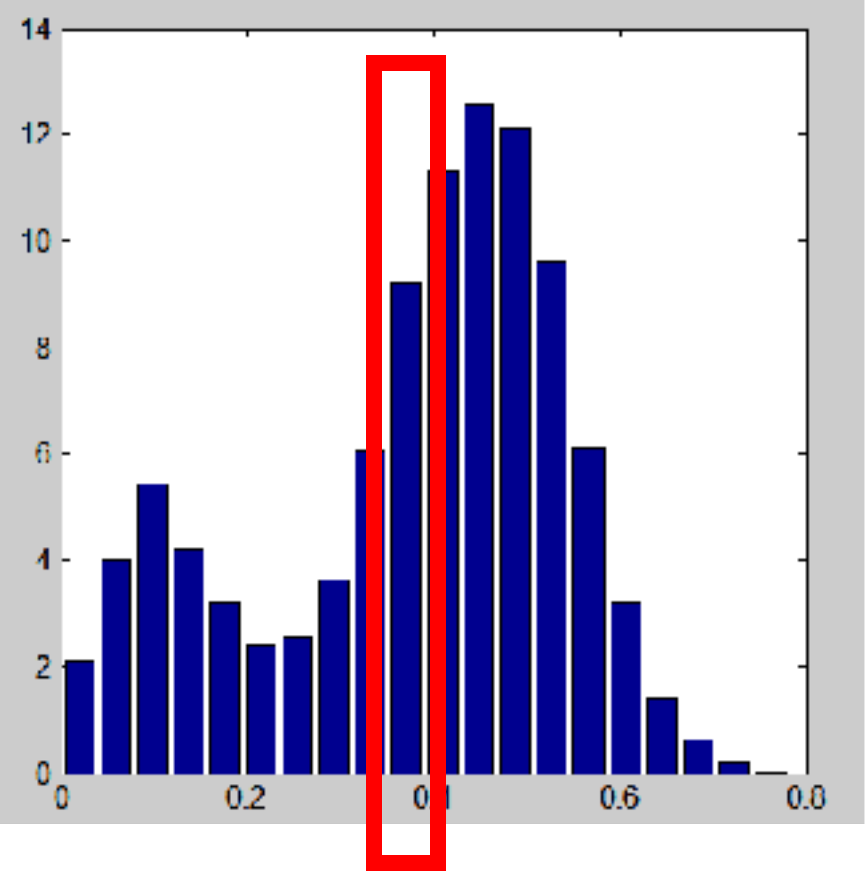
1- steerable pyramid

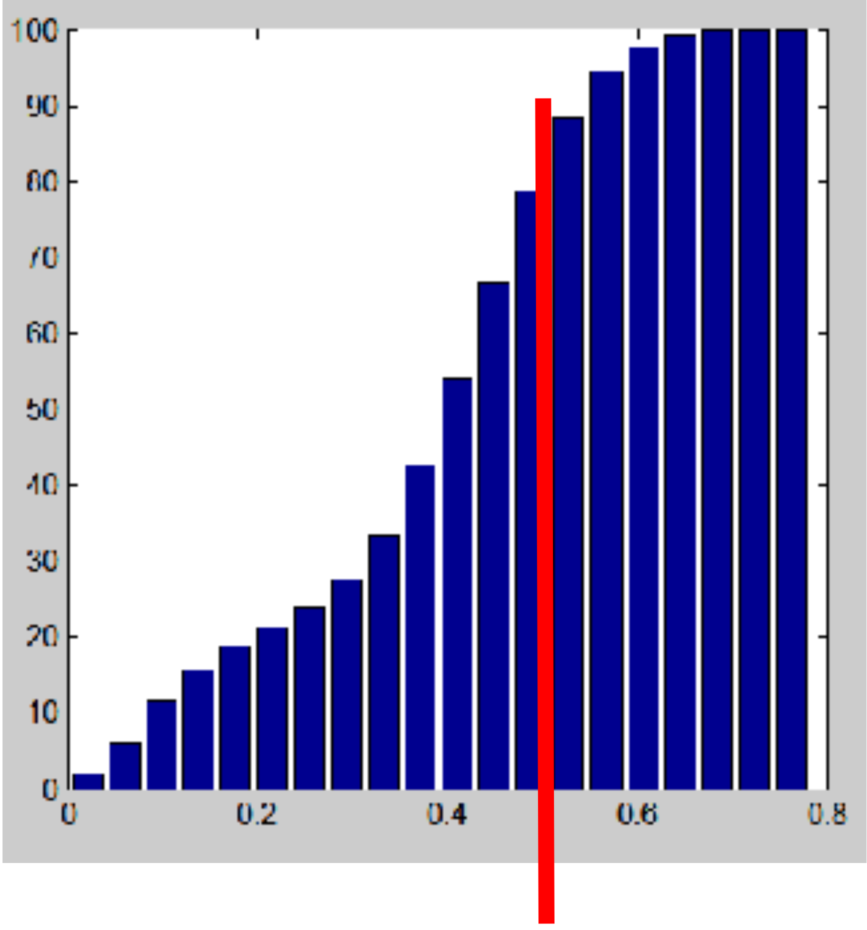
2- matching histograms

2-Matching histograms

Histograms **Cumulative Histograms**

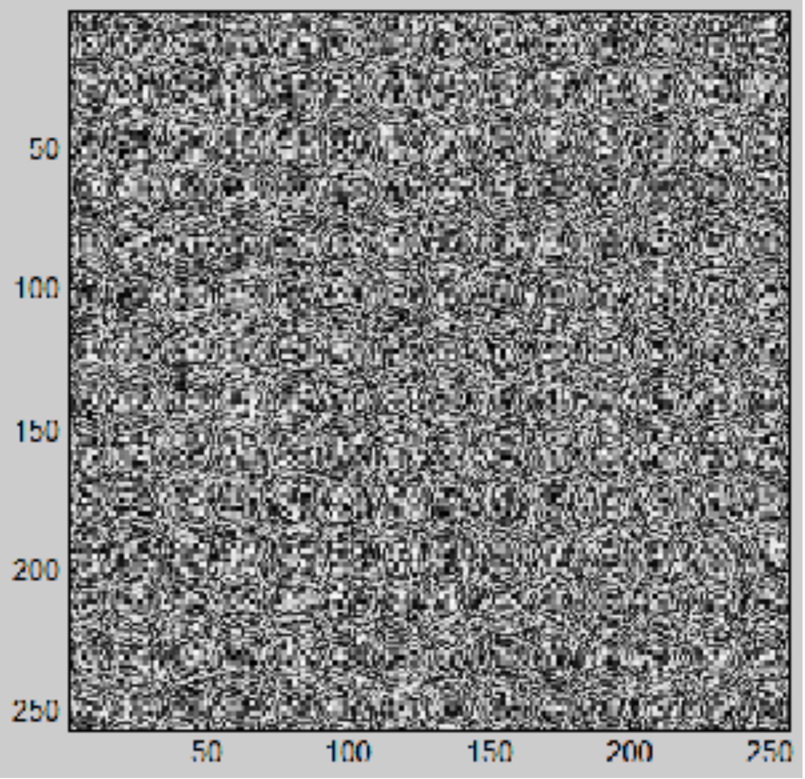


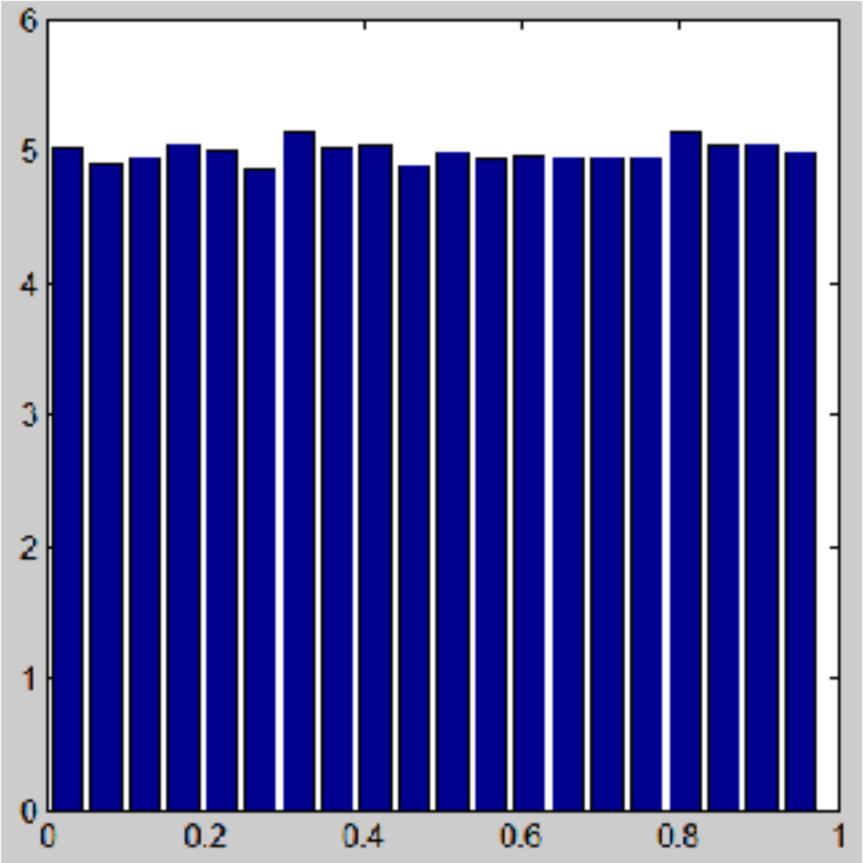


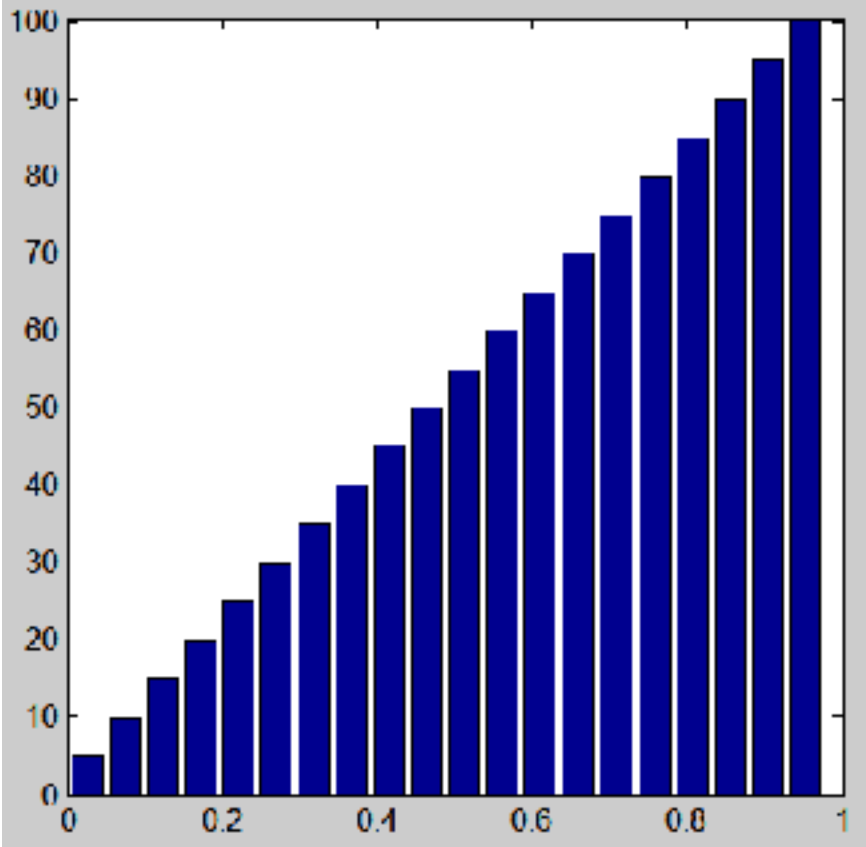


9% of pixels have an intensity value within the range[0.37, 0.41]

75% of pixels have an intensity value smaller than 0.5



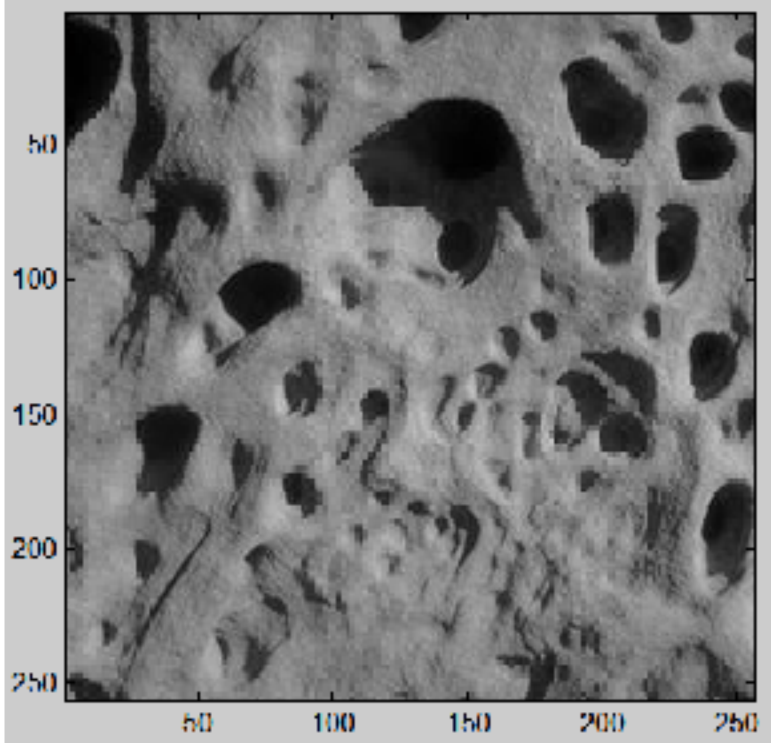


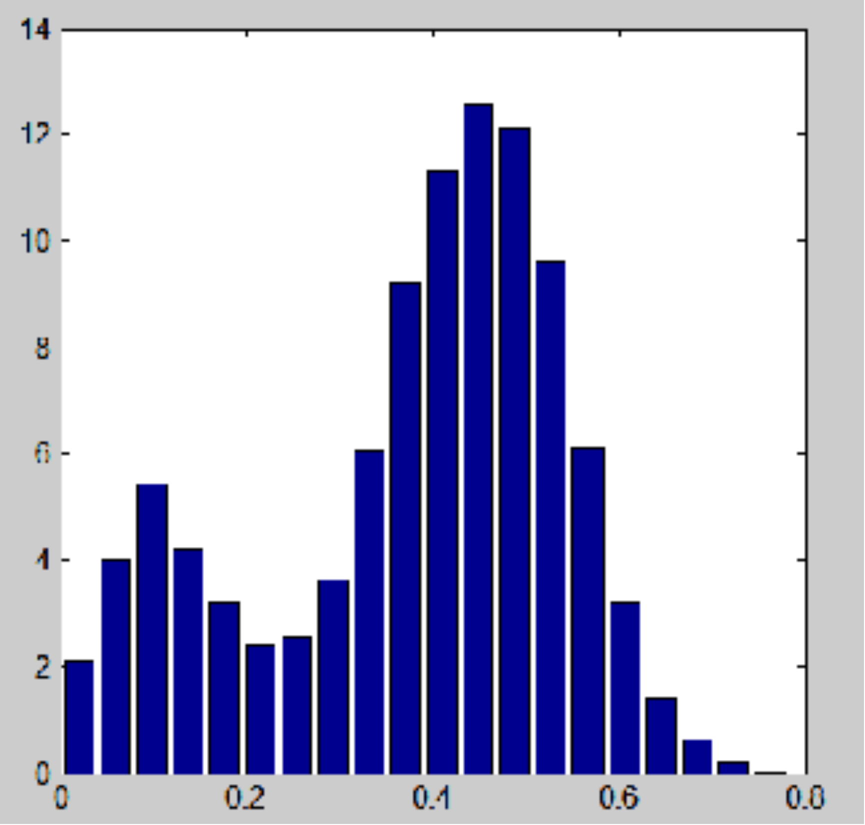


5% of pixels have an intensity value within the range[0.37, 0.41]

2-Matching histograms

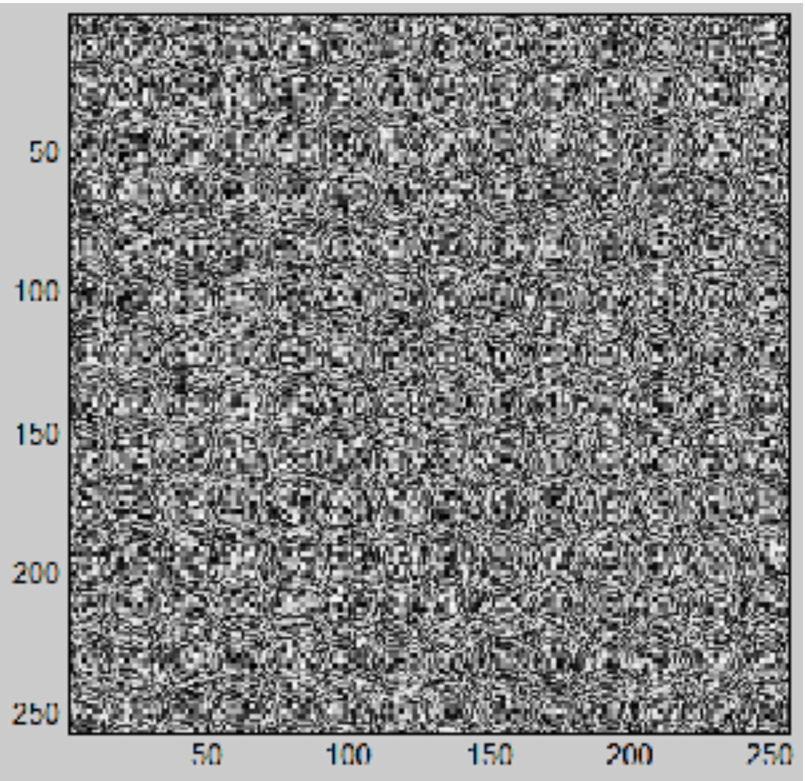
Z(x,y)

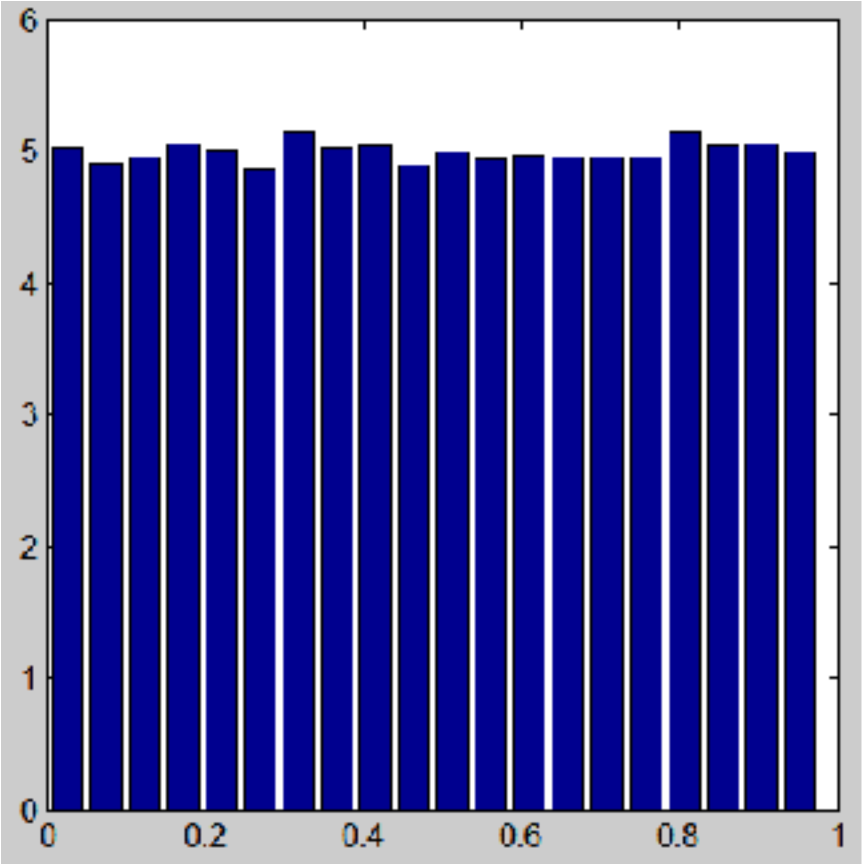




?

Y(x,y)





We look for a transformation of the image Y

Y’ = f (Y)

Such that
Hist(Y) = Hist(f(Z))

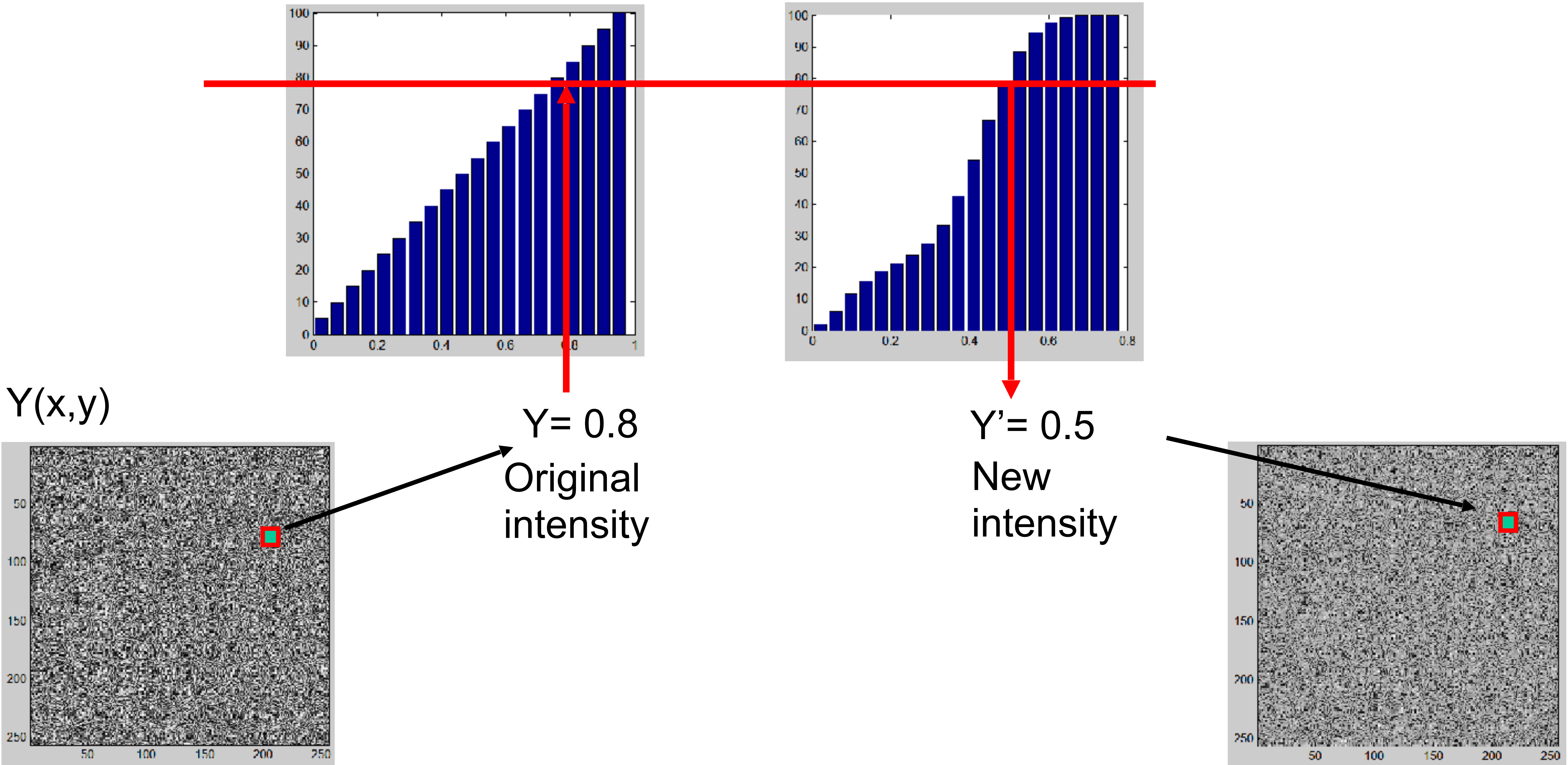
Problem: there are infinitely many functions that can do this transformation.

A natural choice is to use *f* being:
- pointwise non linearity
- stationary
- monotonic (most of the time invertible)

2-Matching cumulative histograms

The function f is just a look up table: it says, change all the pixels of value Y into a value f(Y).

$$Y' = f\ (Y)$$

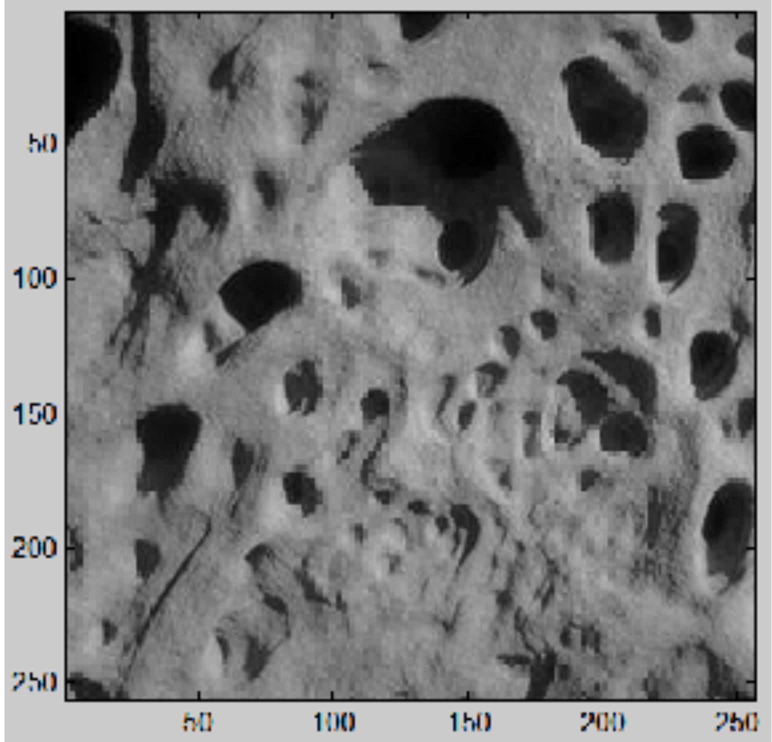


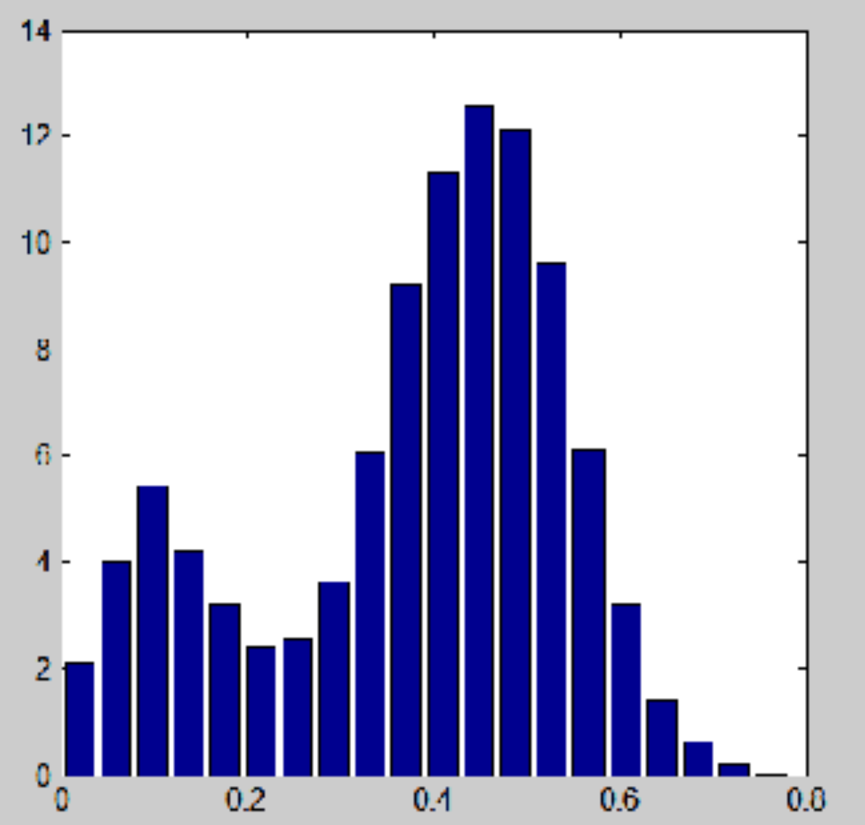
Y(x,y)

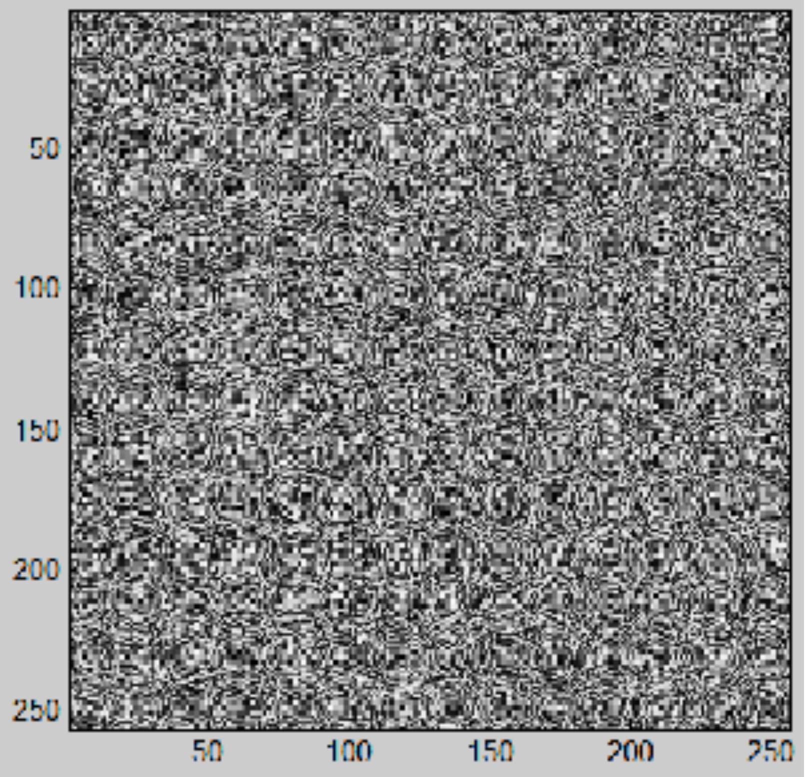
Y= 0.8
Original intensity

Y'= 0.5
New intensity

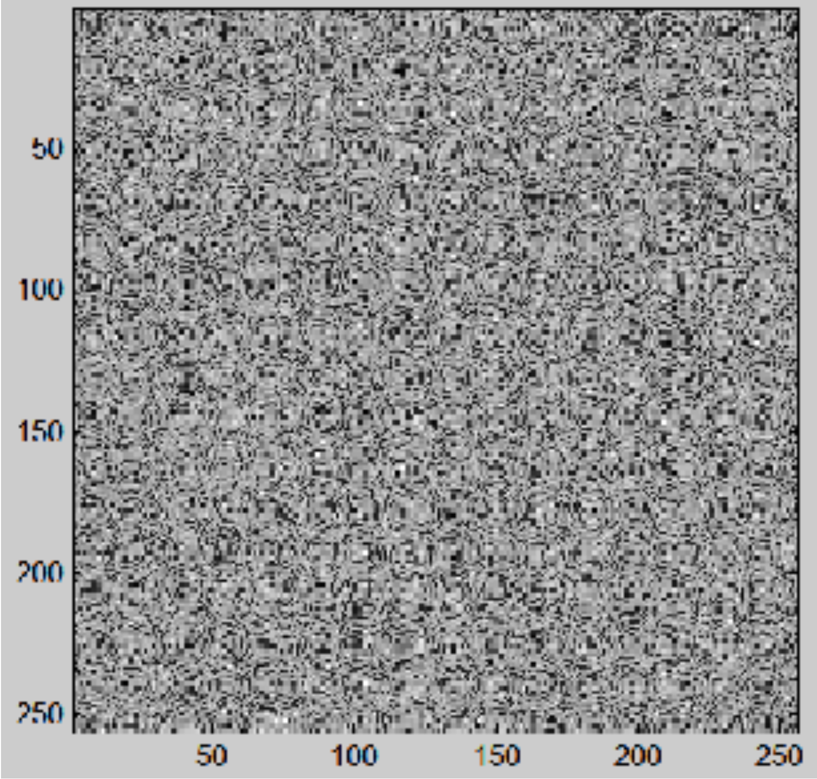
2-Matching histograms

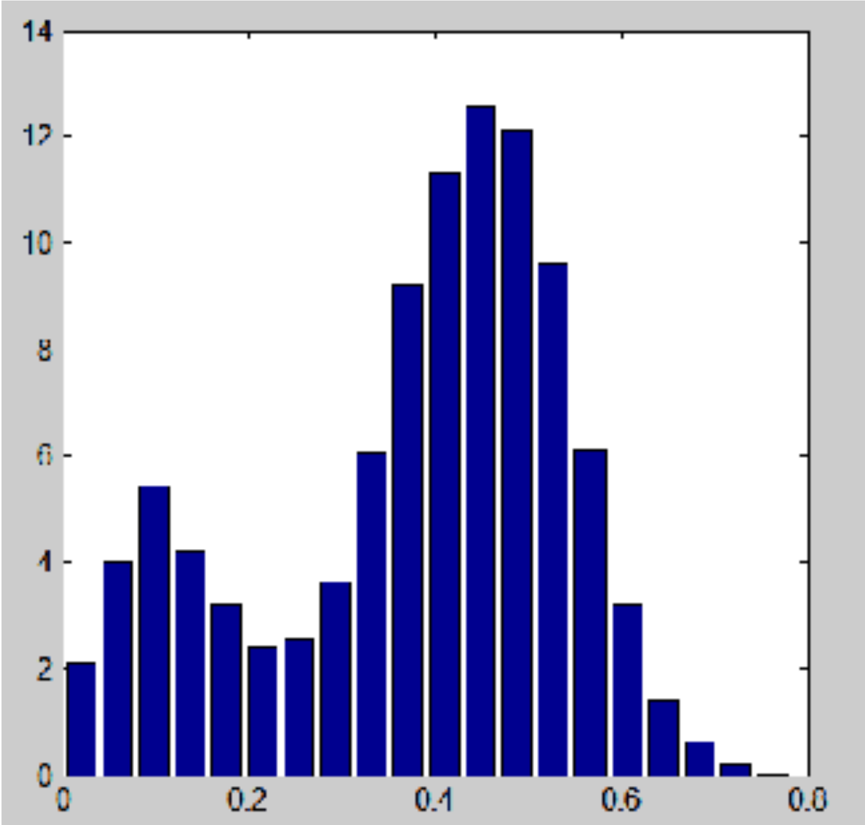




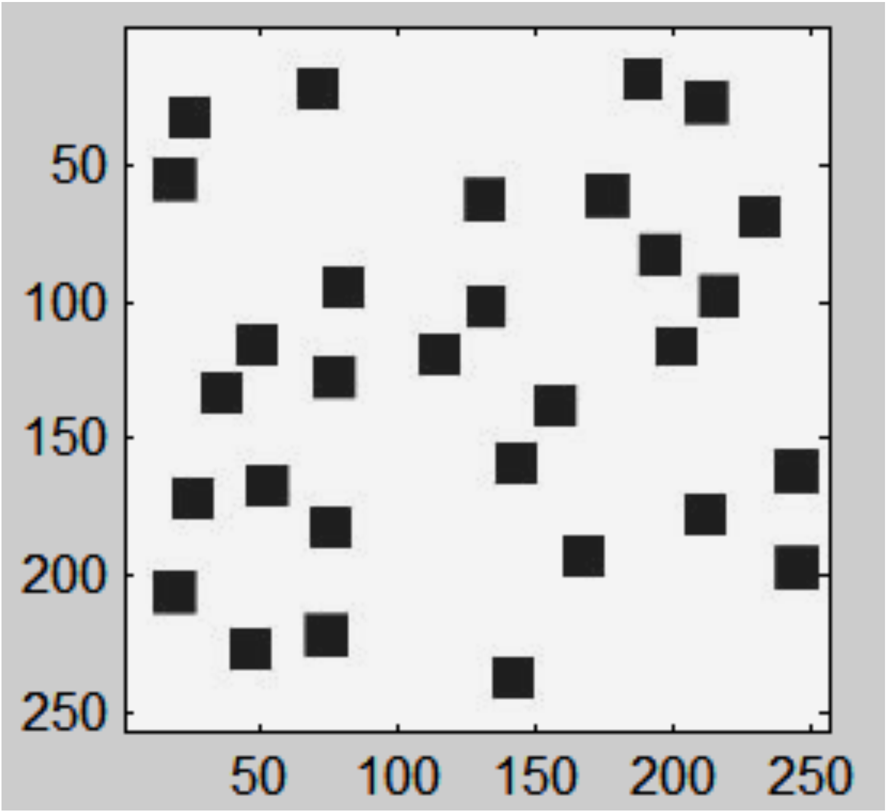


Y' = f (Y)

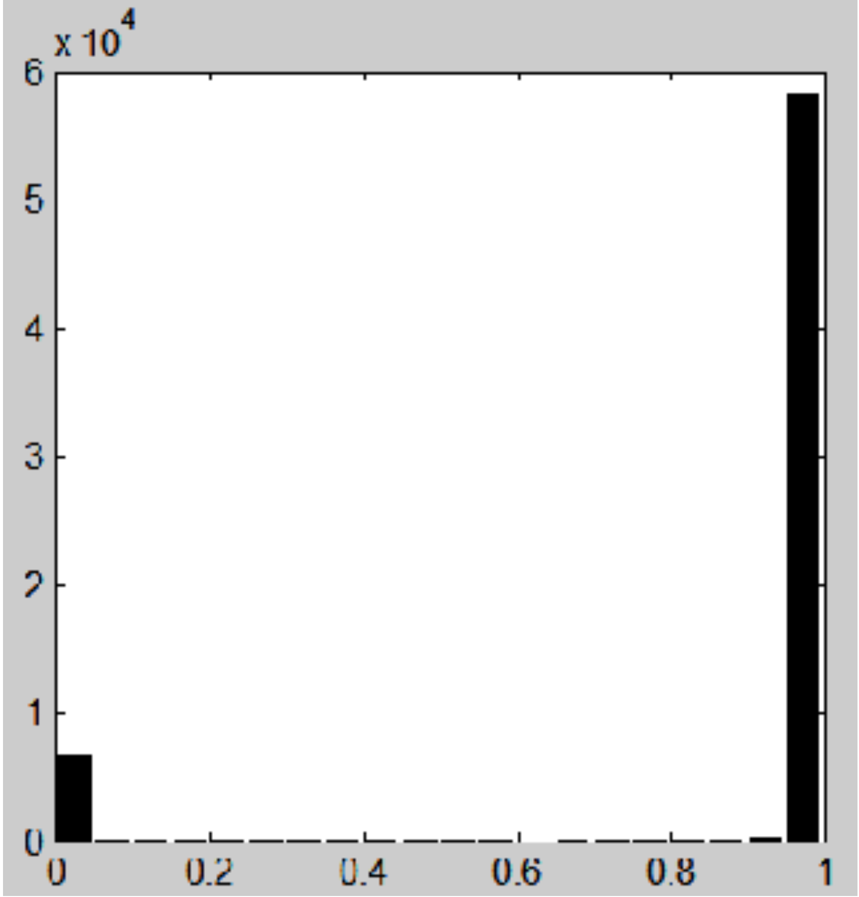




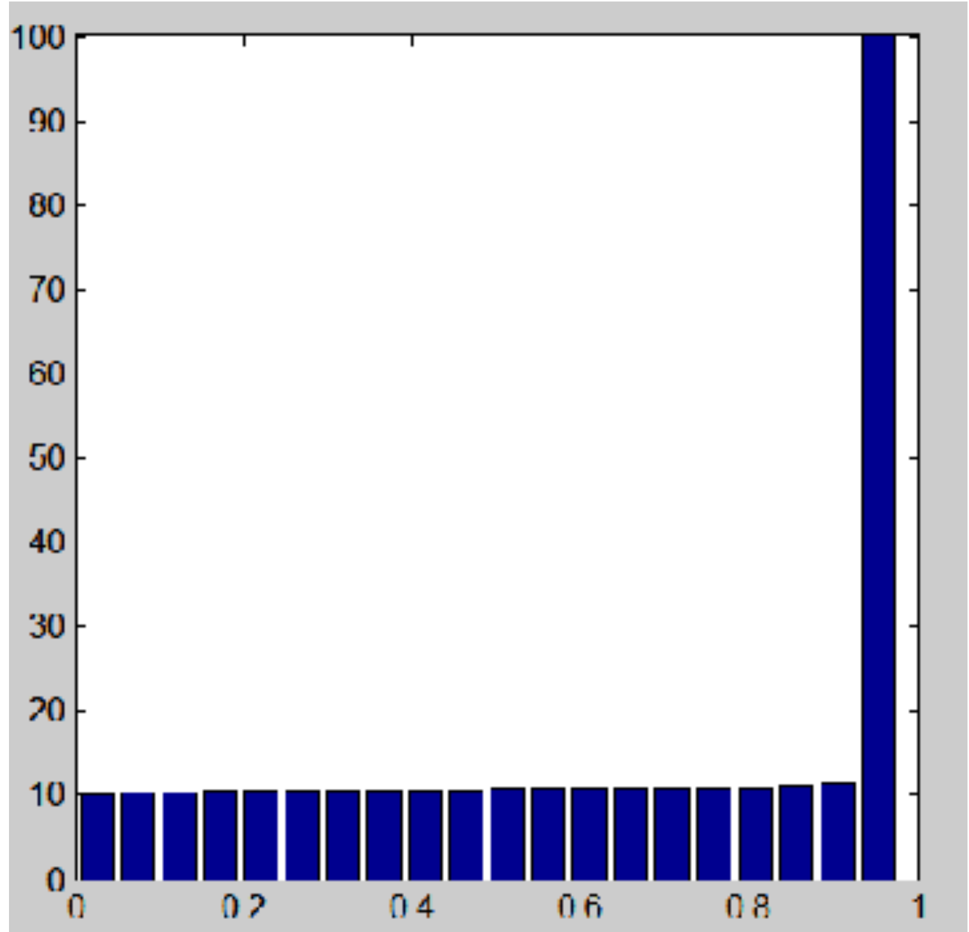
Matching histograms example



Histograms



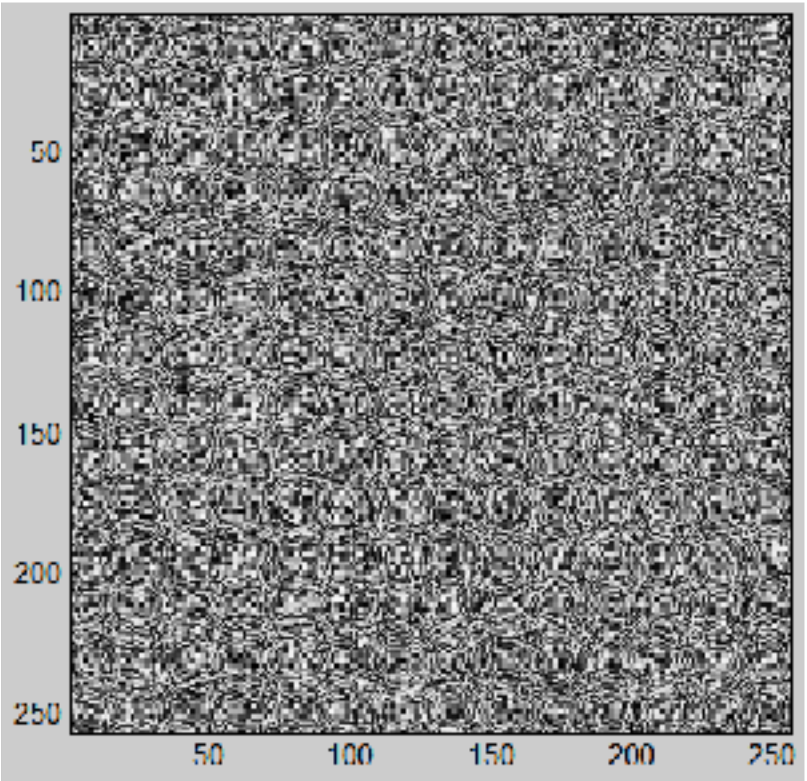
Cumulative Histograms

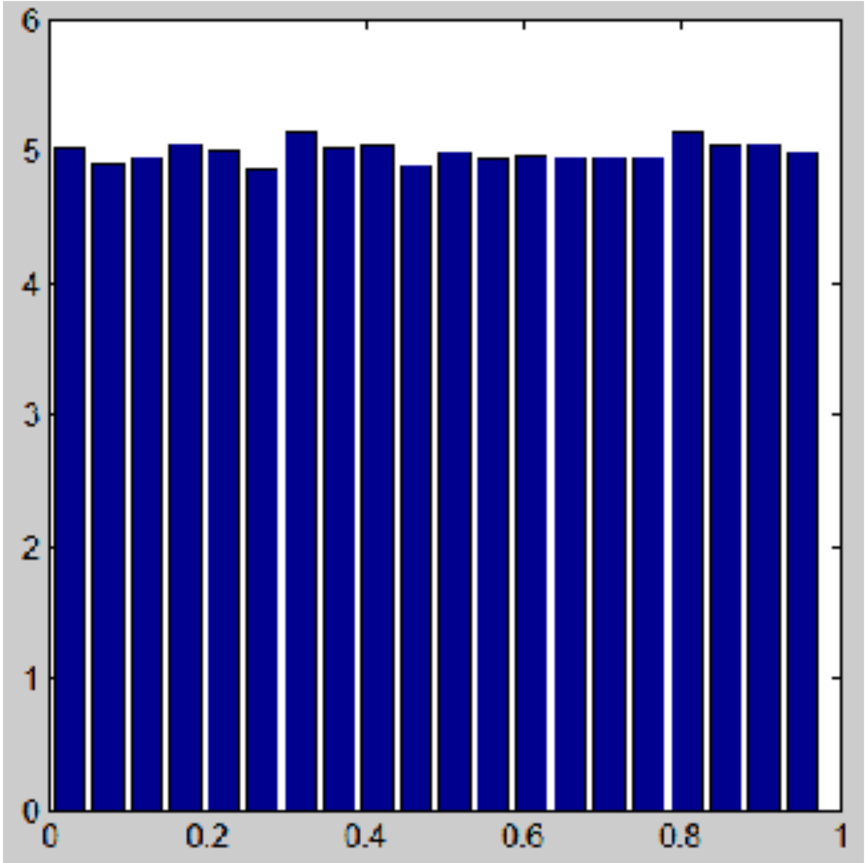


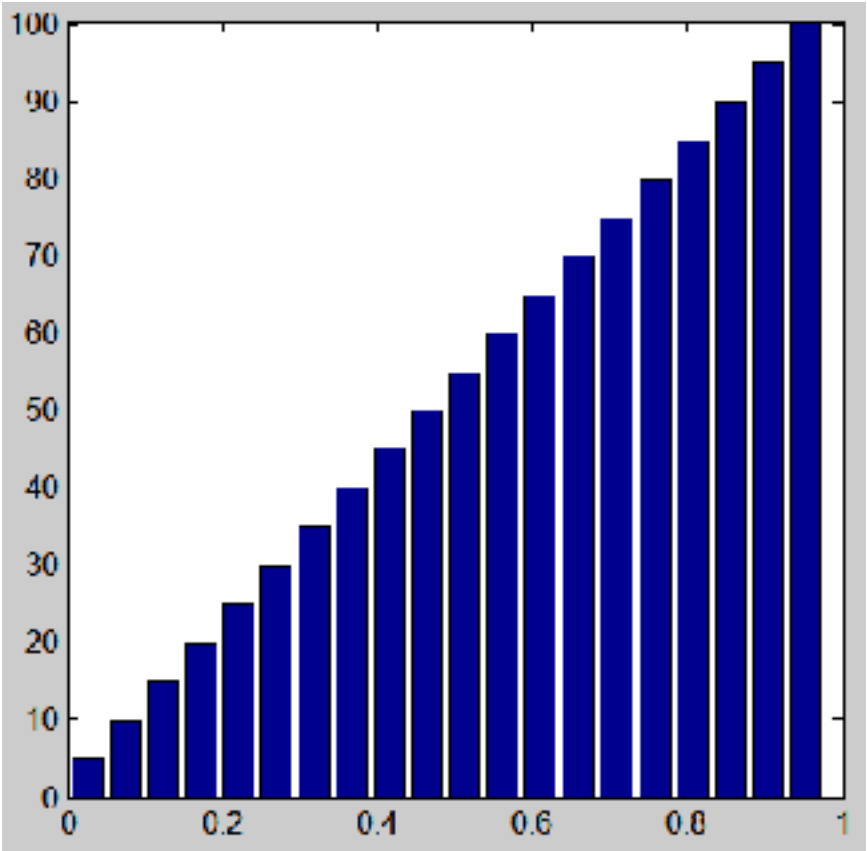
?

?

10% of pixels are black
and 90% are white





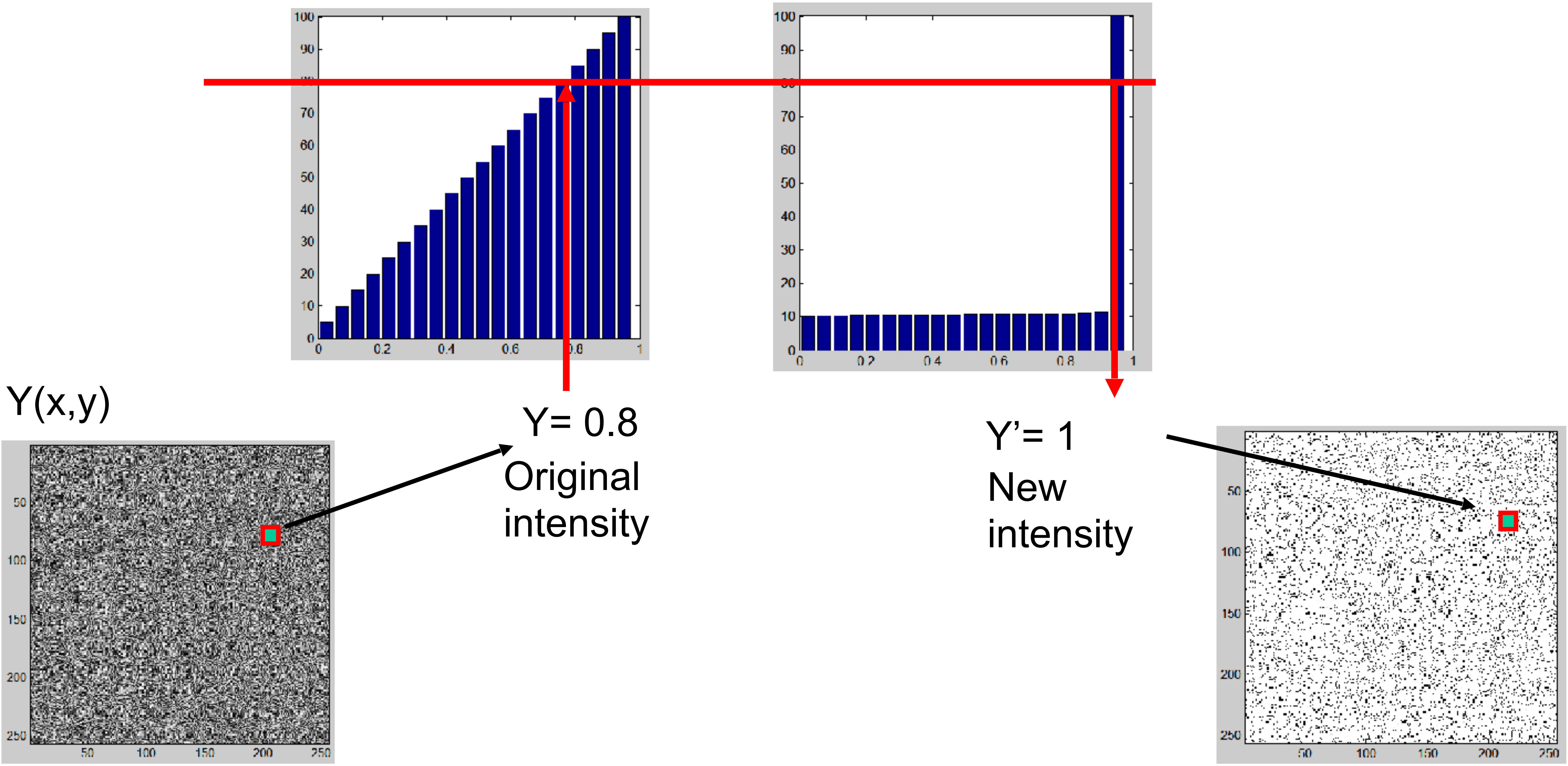


5% of pixels have an intensity value
within the range[0.37, 0.41]

Matching histograms example

The function f is just a look up table: it says, change all the pixels of value Y into a value f(Y).

Y’ = f (Y)



Y(x,y)

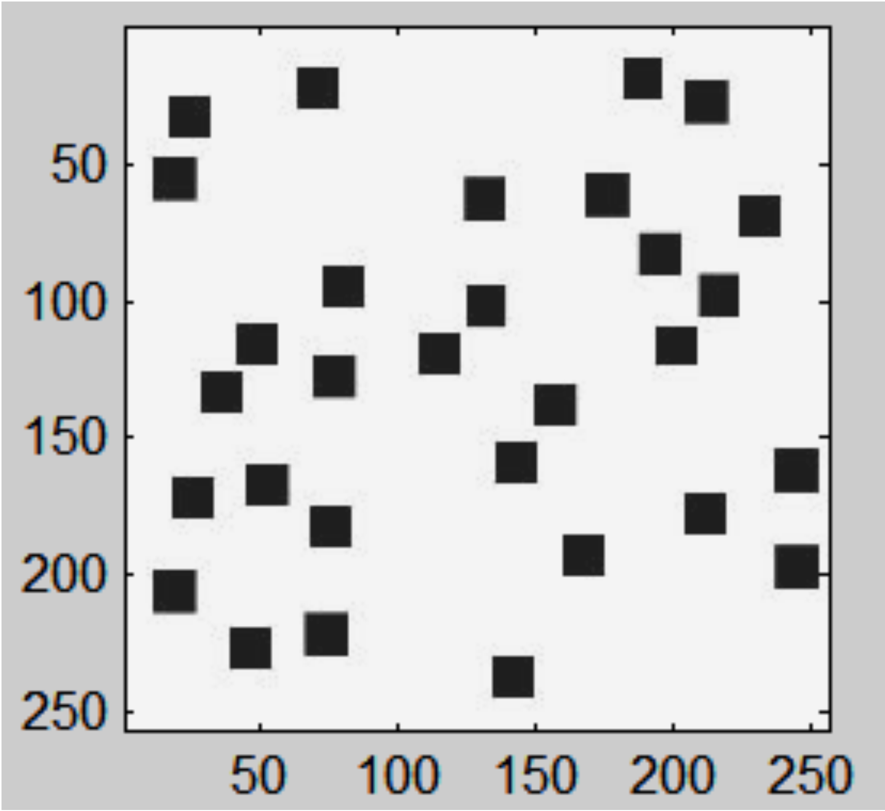
Y= 0.8

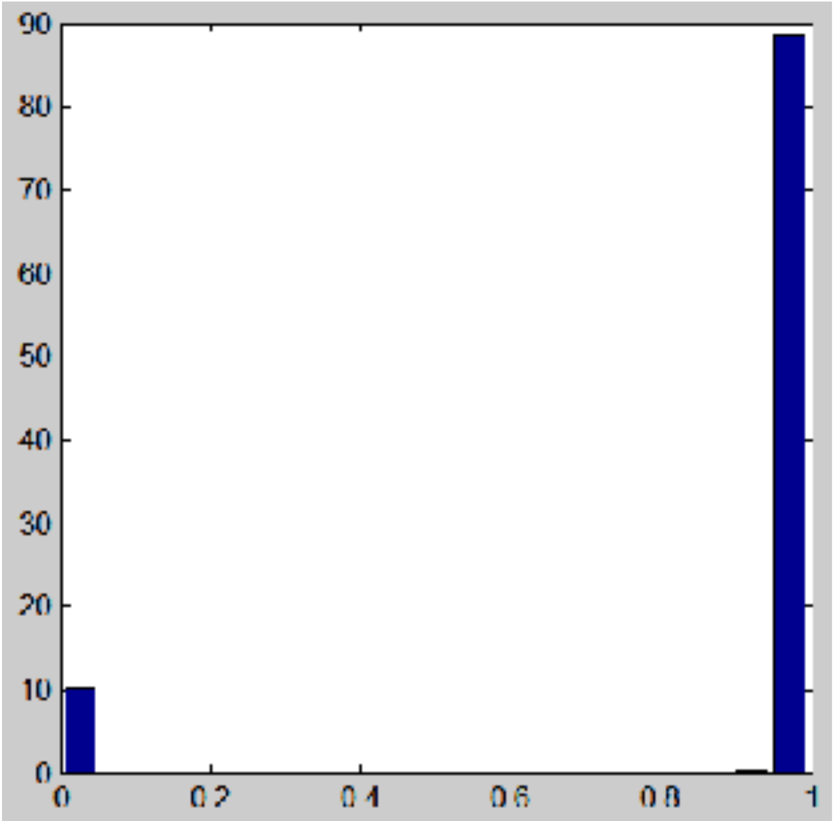
Original intensity

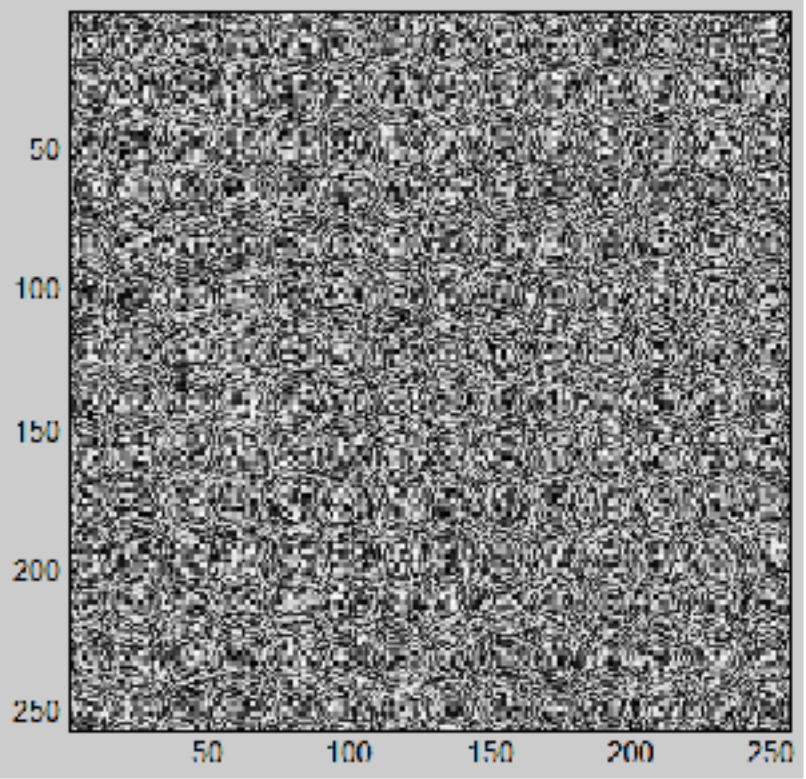
Y’= 1

New intensity

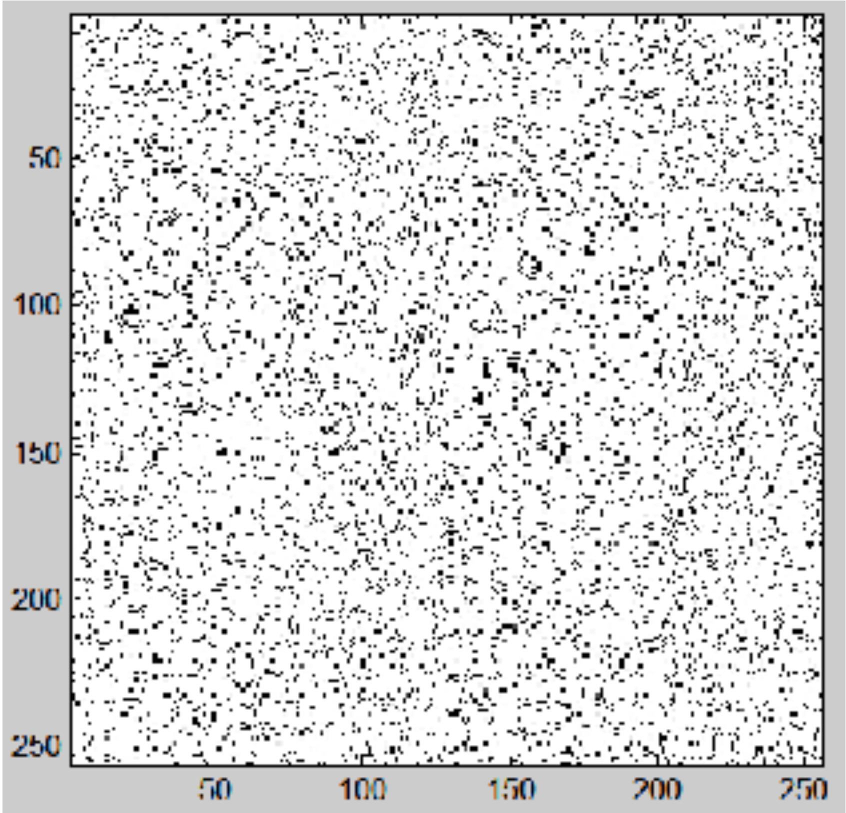
Matching histograms example

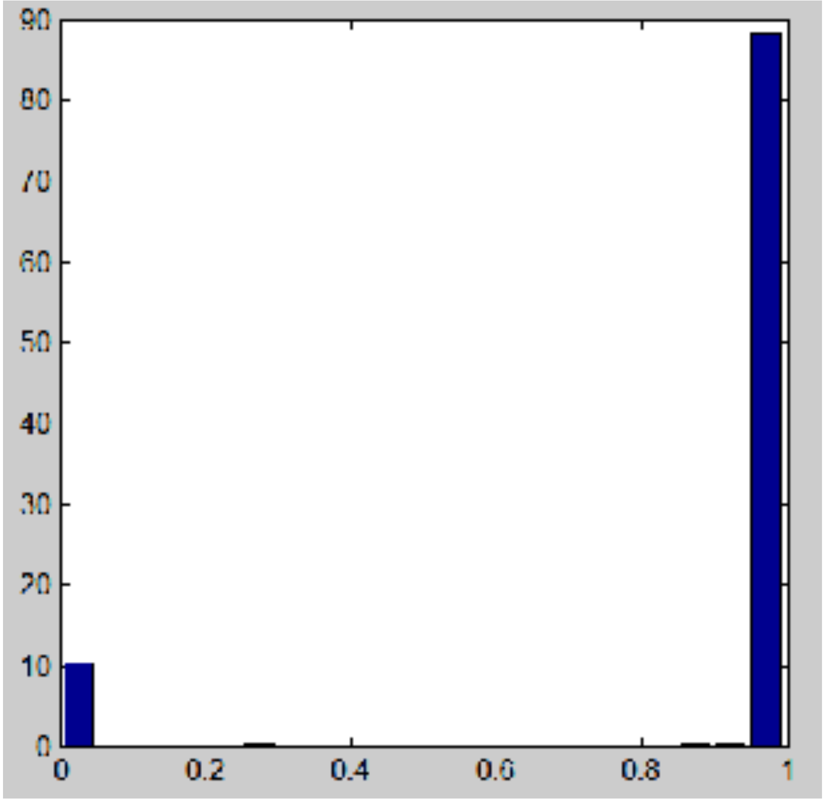






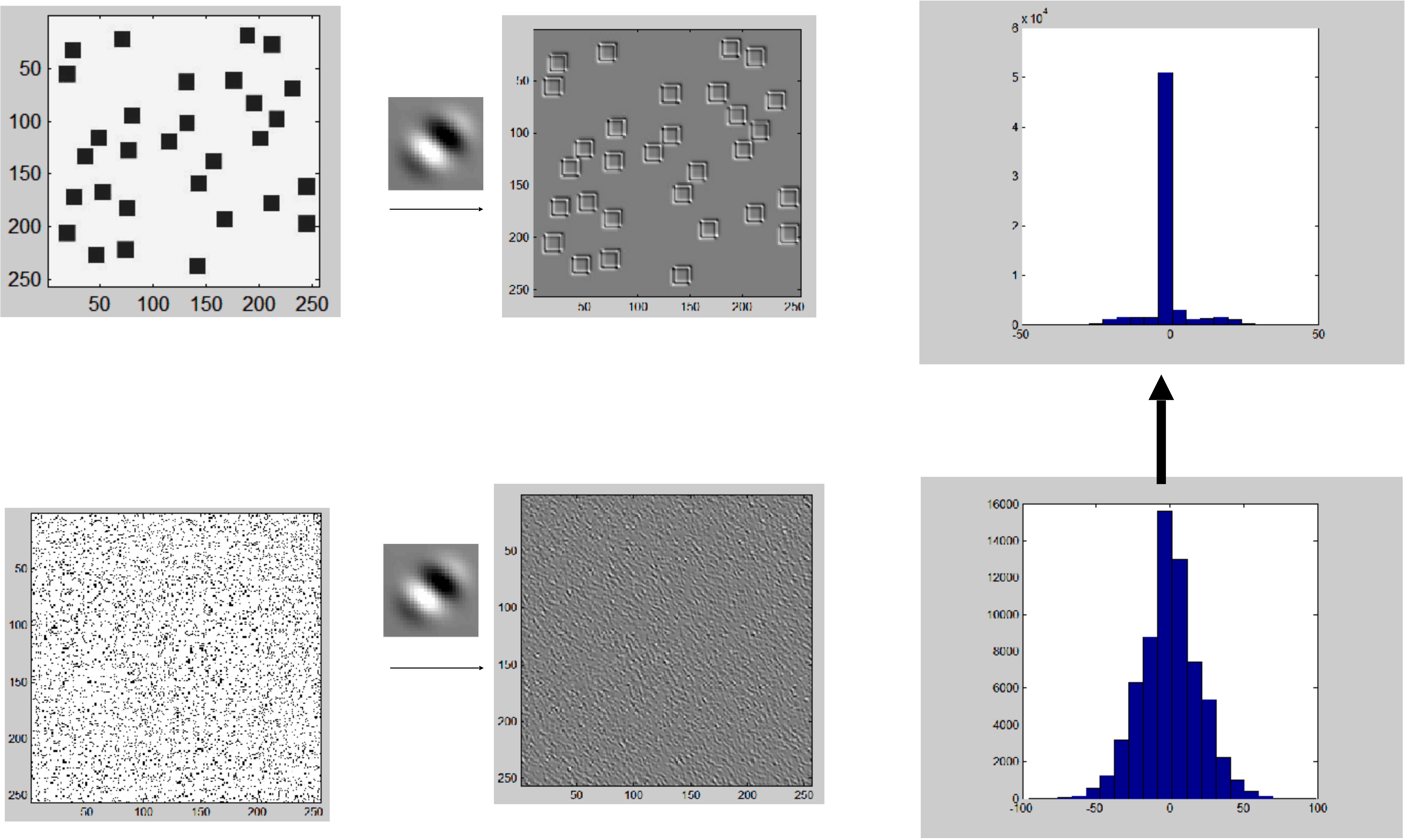
Y’ = f (Y)



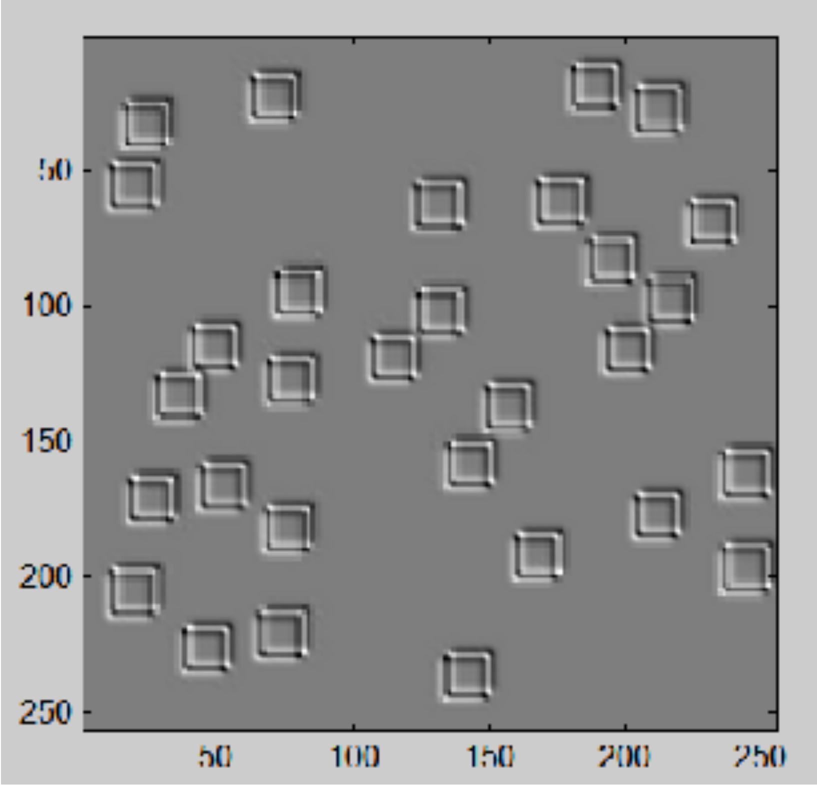


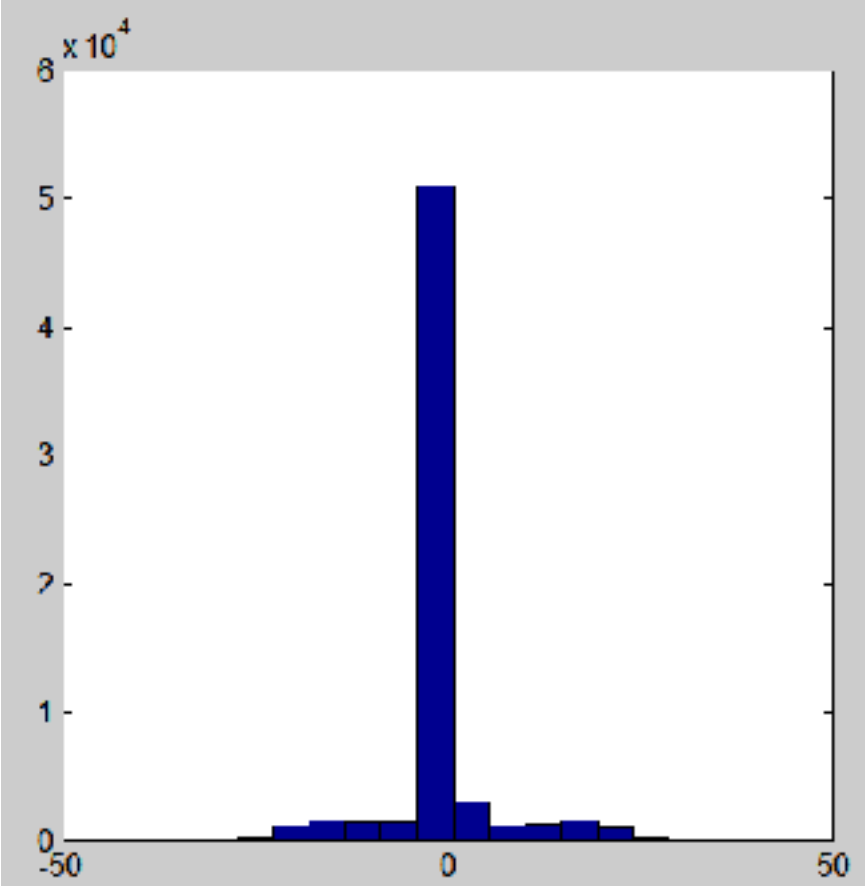
In this example, f is a step function.

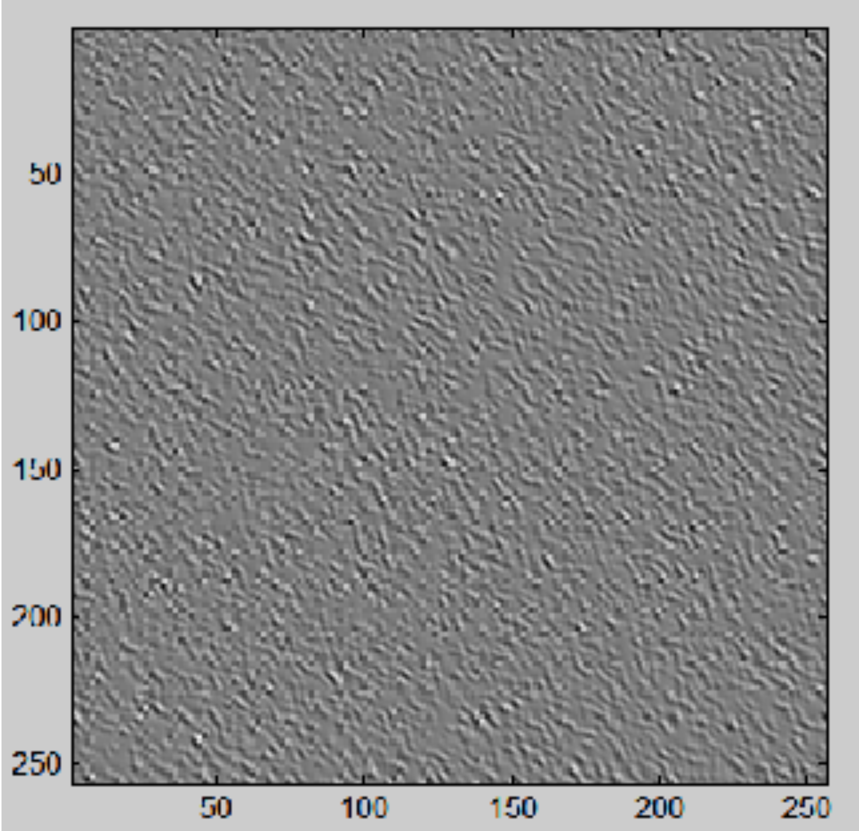
Matching histograms of a subband



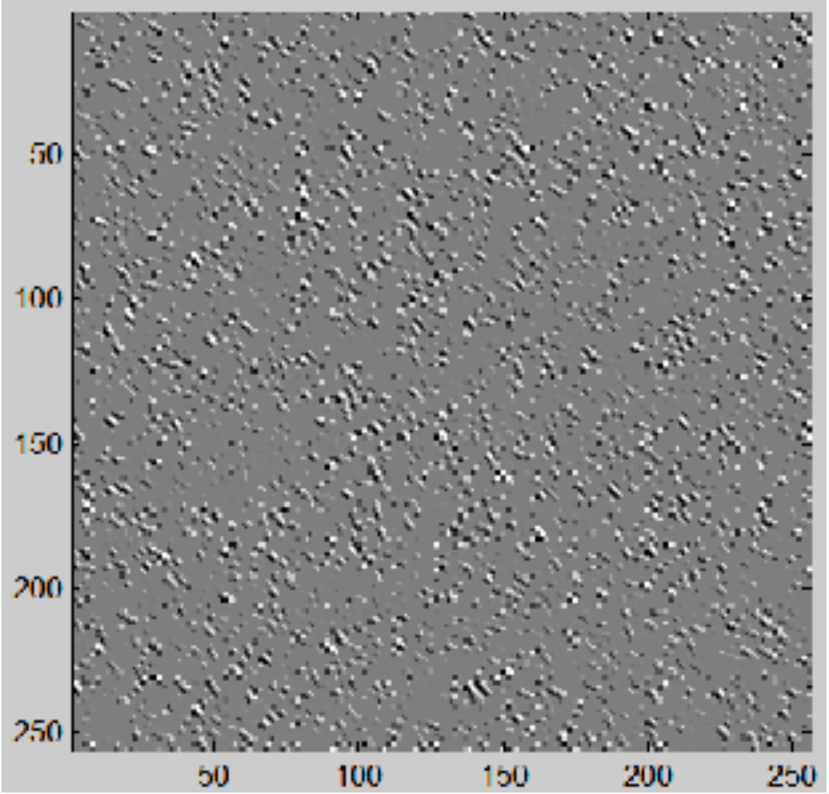
Matching histograms of a subband

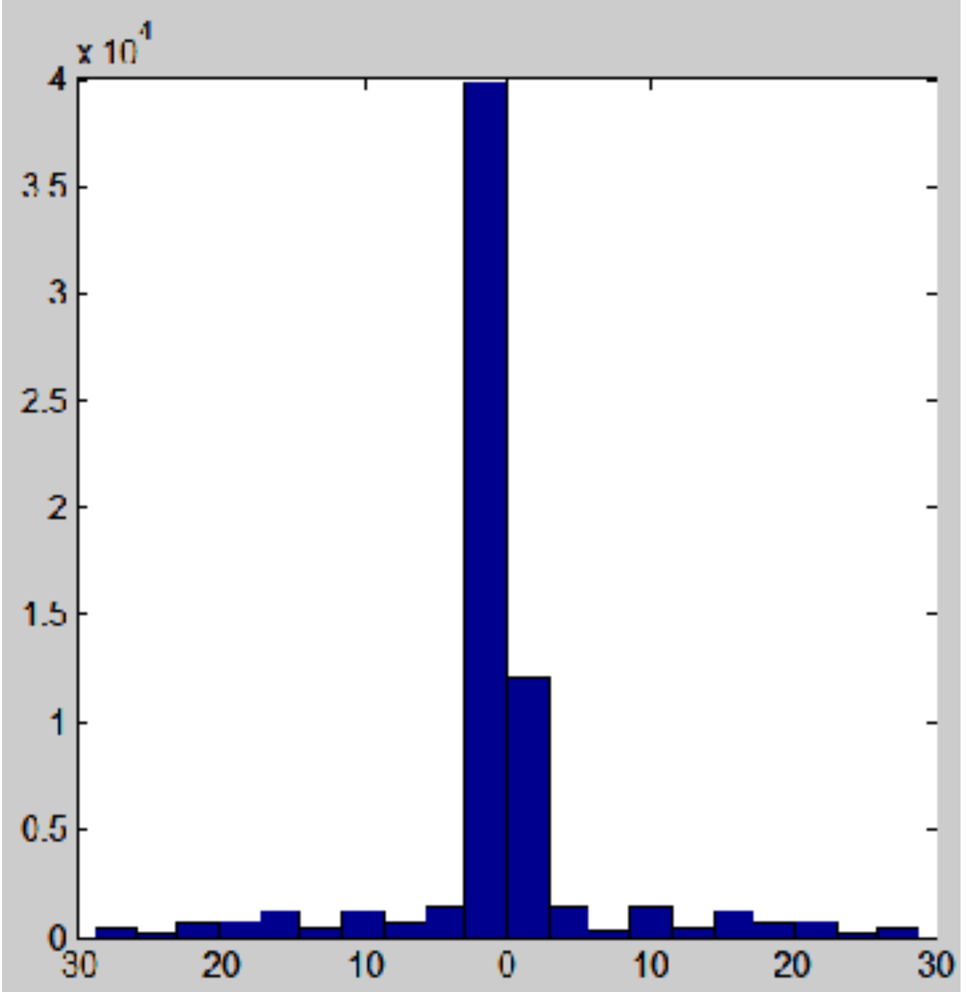






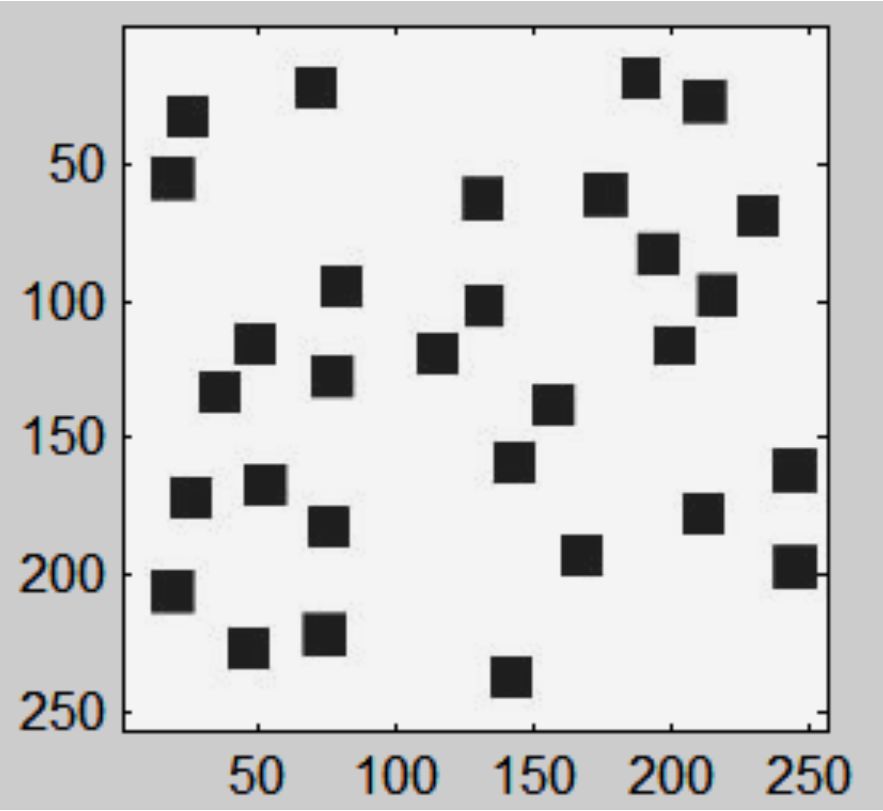
Y’ = f (Y)



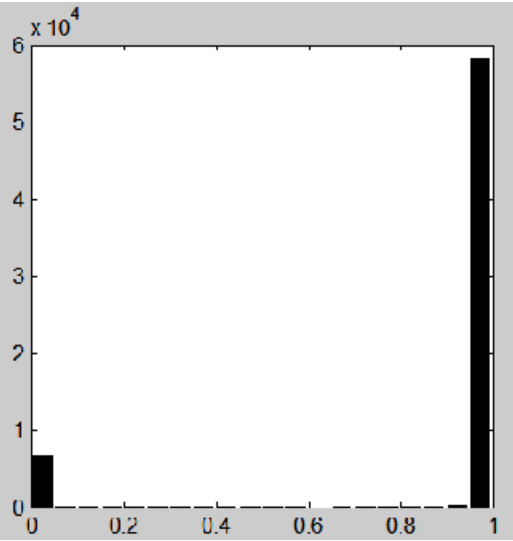


Texture analysis

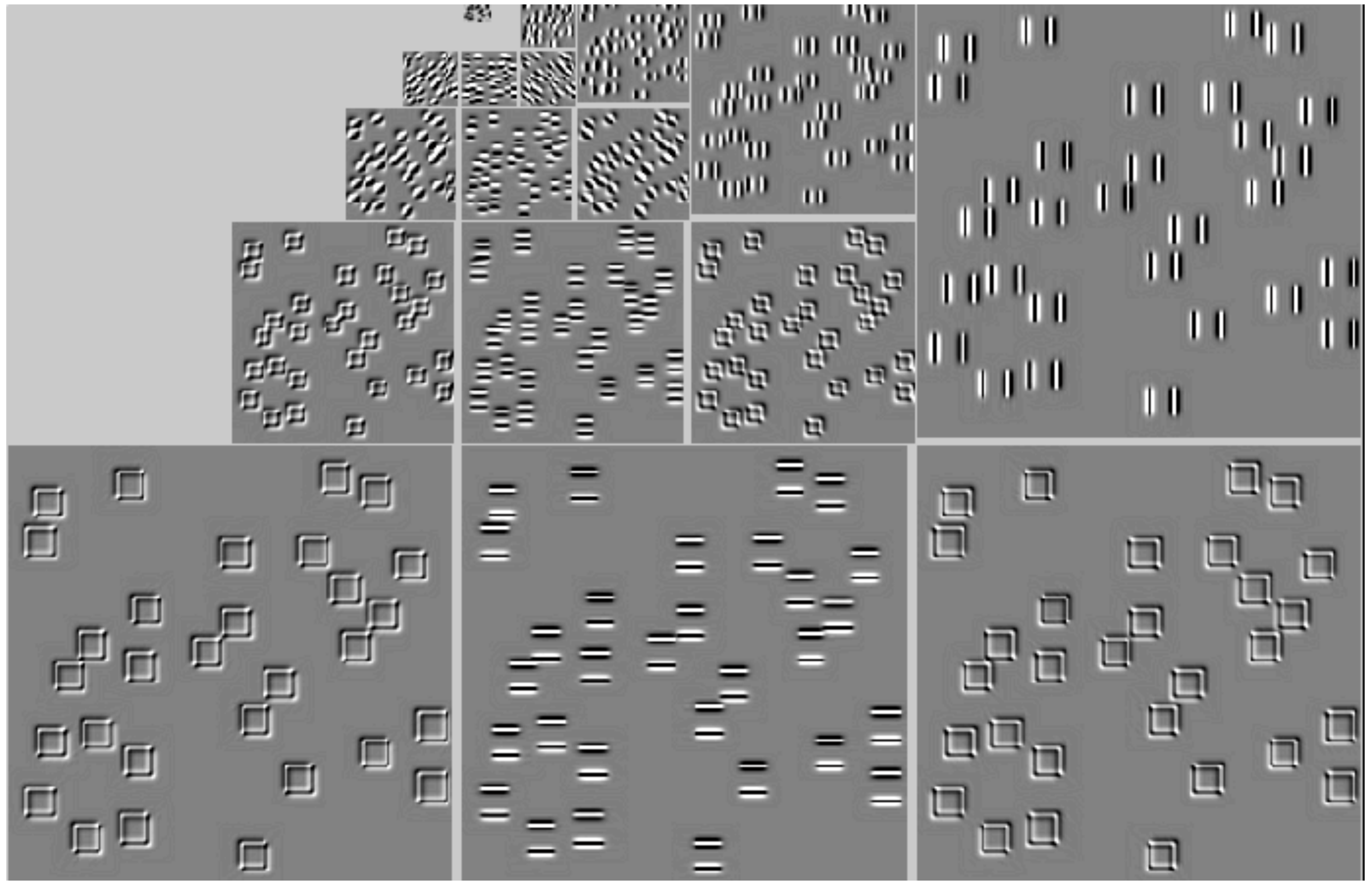
Input texture



(histogram)

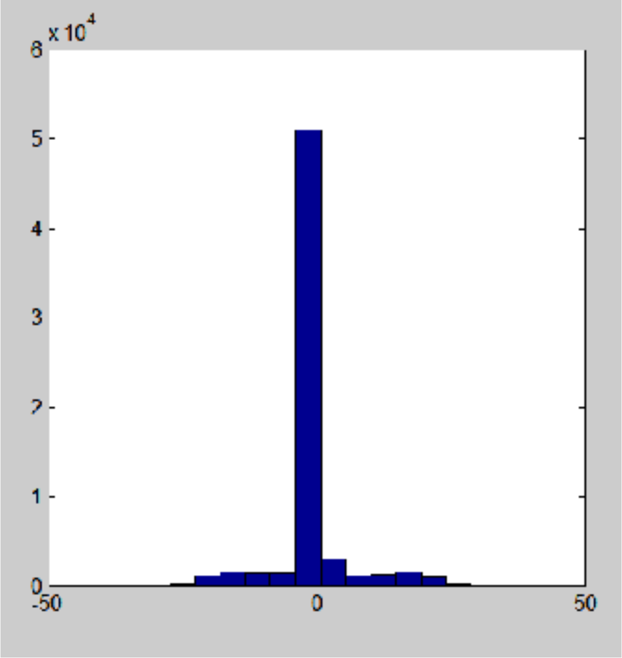


Wavelet decomposition (steerable pyr)



(Steerable pyr; Simoncelli & Freeman, ’95)

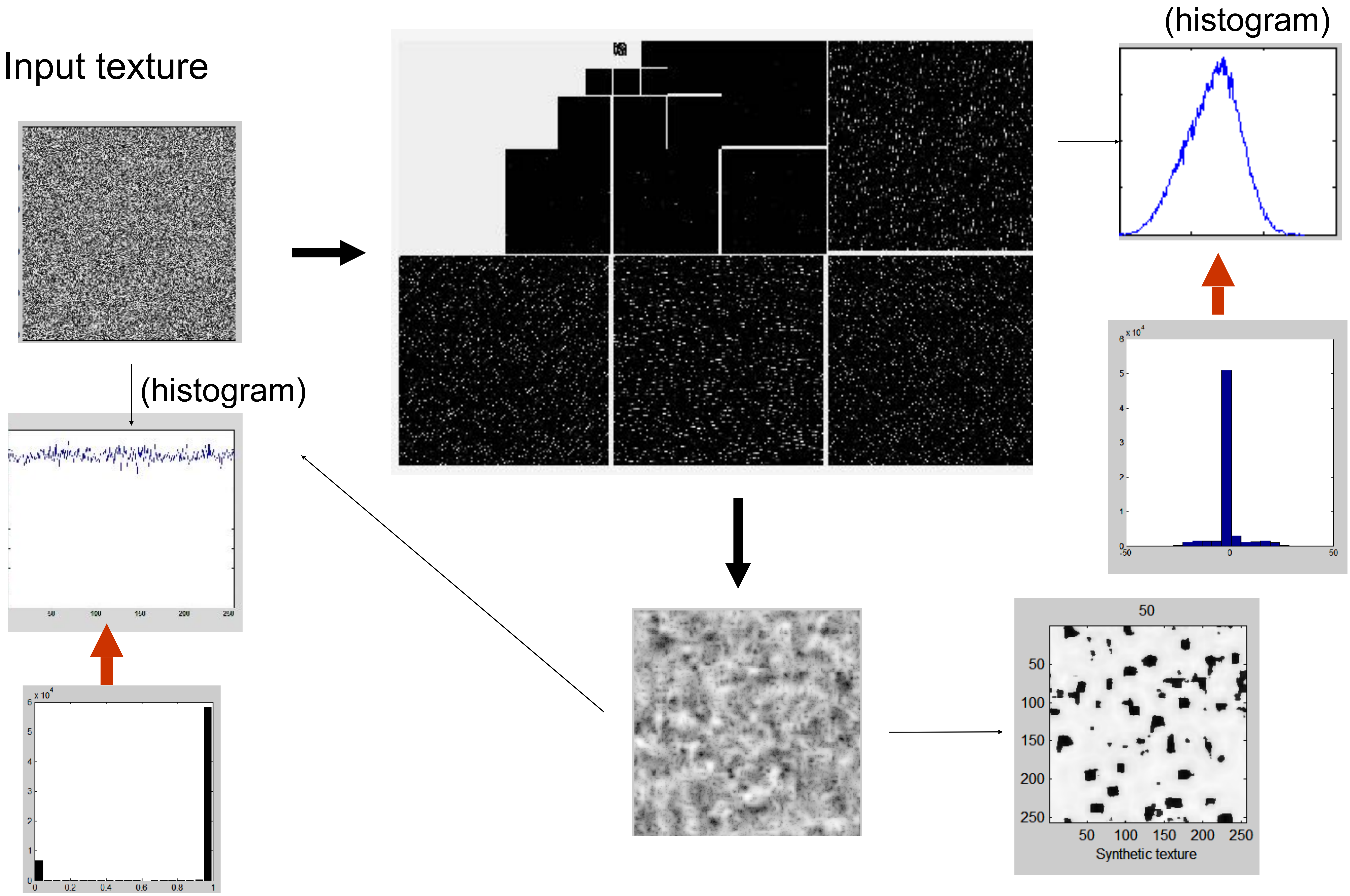
(histogram)



The texture is represented as a collection of marginal histograms.

Texture synthesis

Heeger and Bergen, 1995

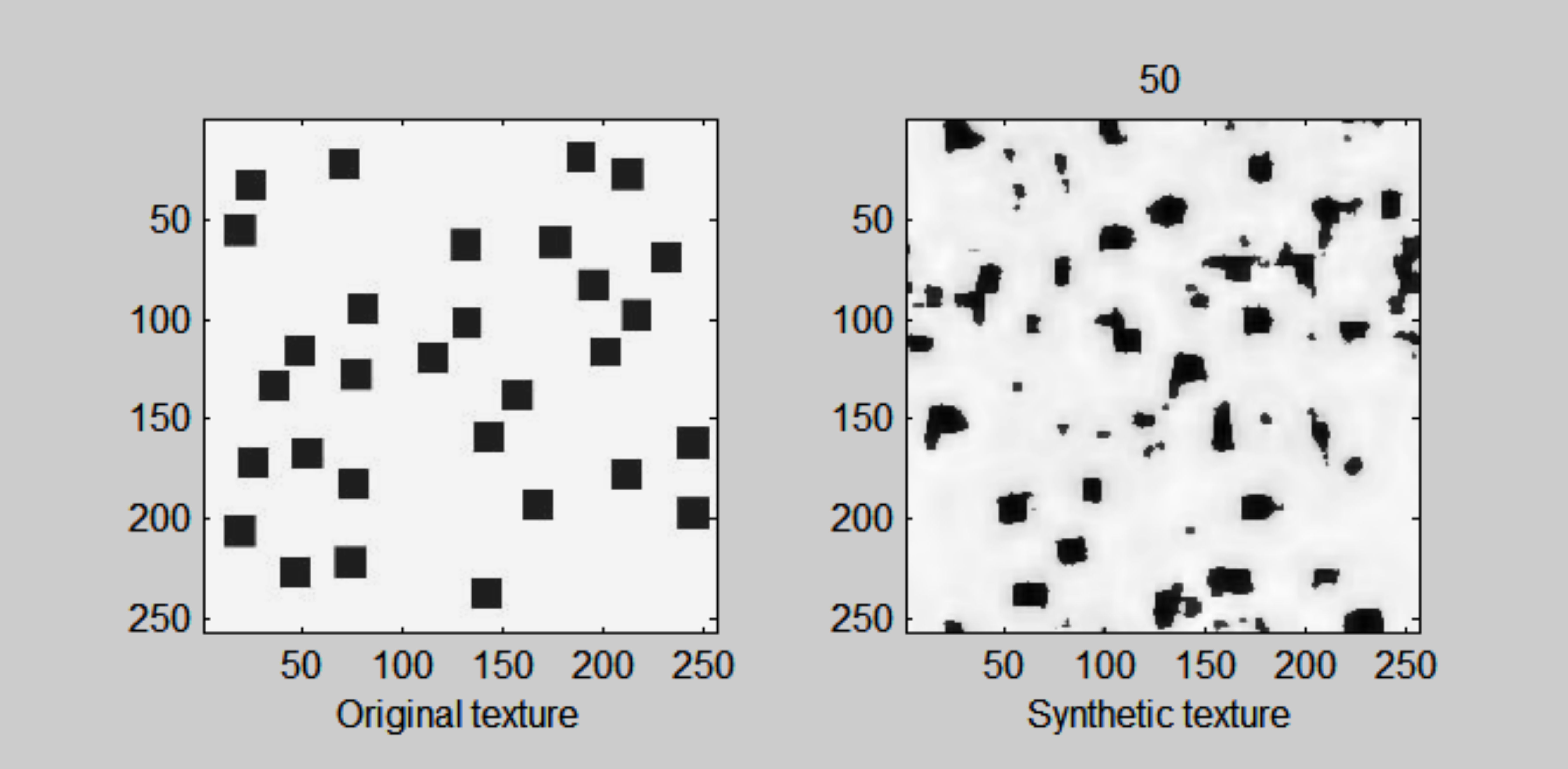


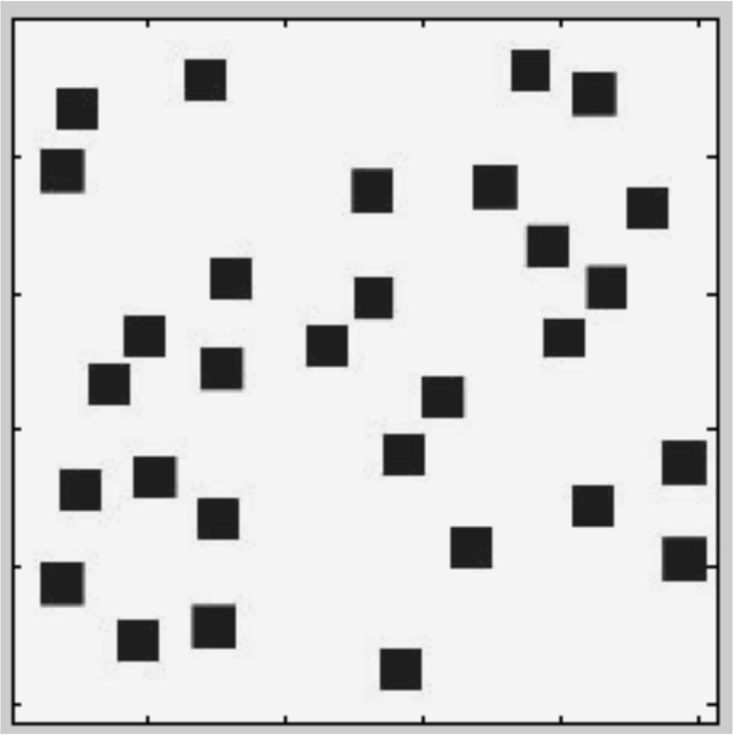
Input texture

(histogram)

(histogram)

Why does it work? (sort of)



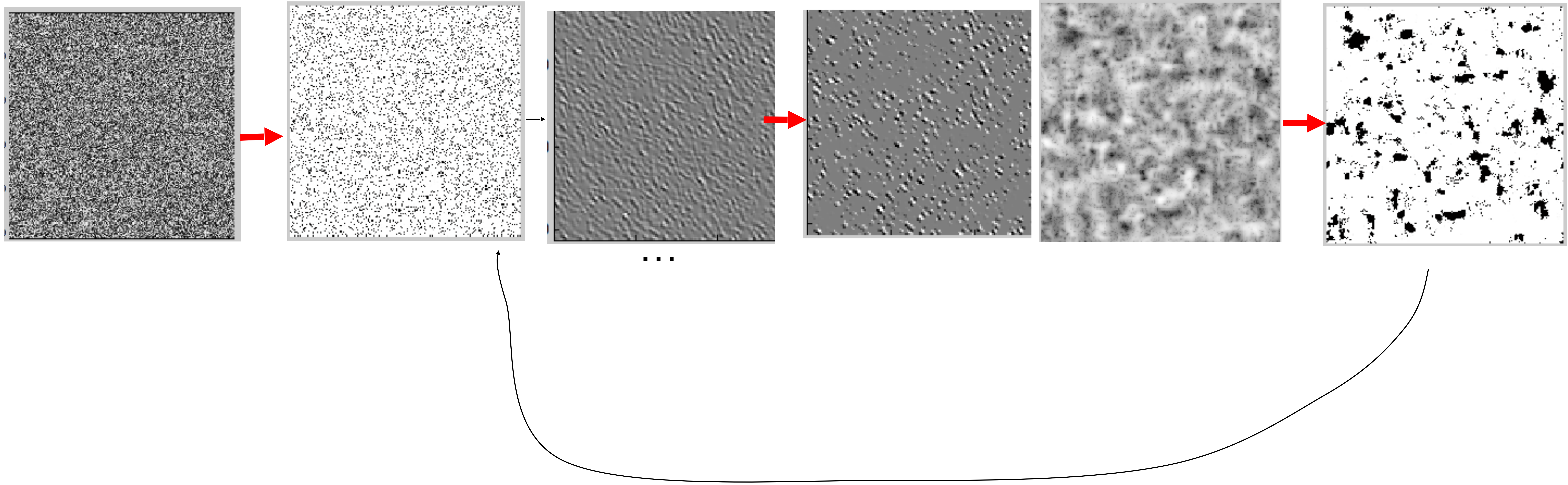


Why does it work? (sort of)

The black and white blocks appear by thresholding (f) a blobby image

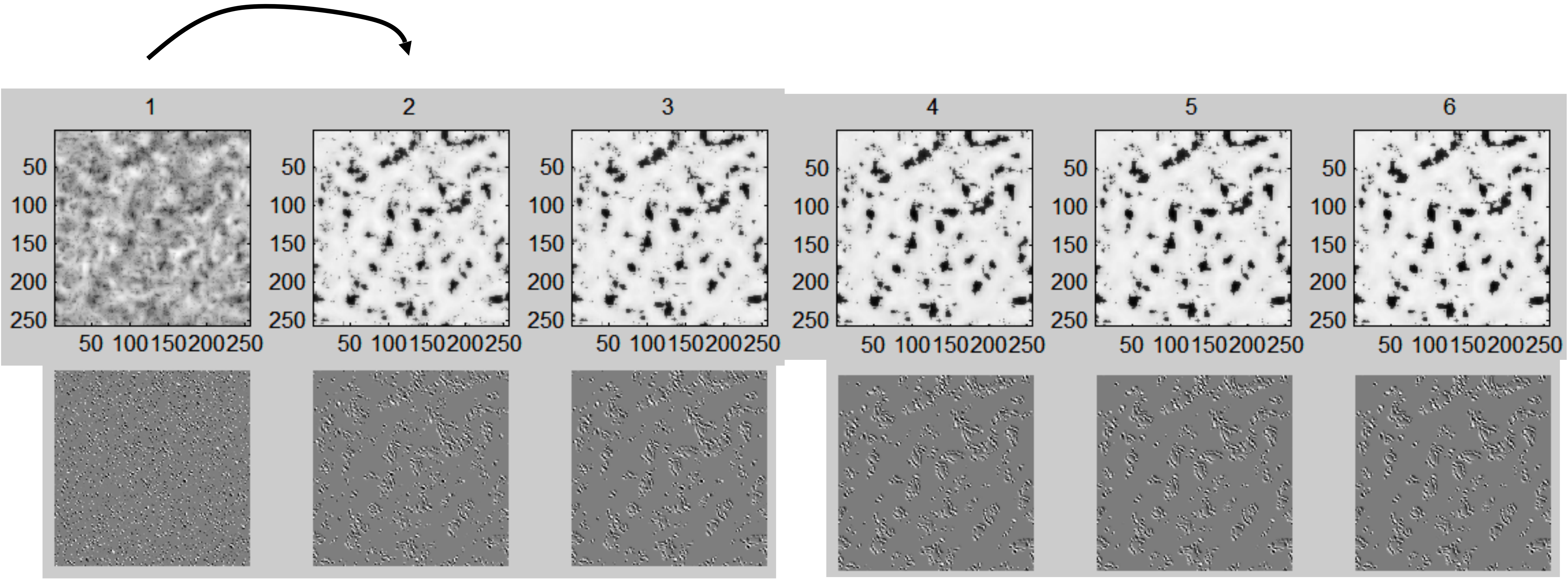
Iteration 0

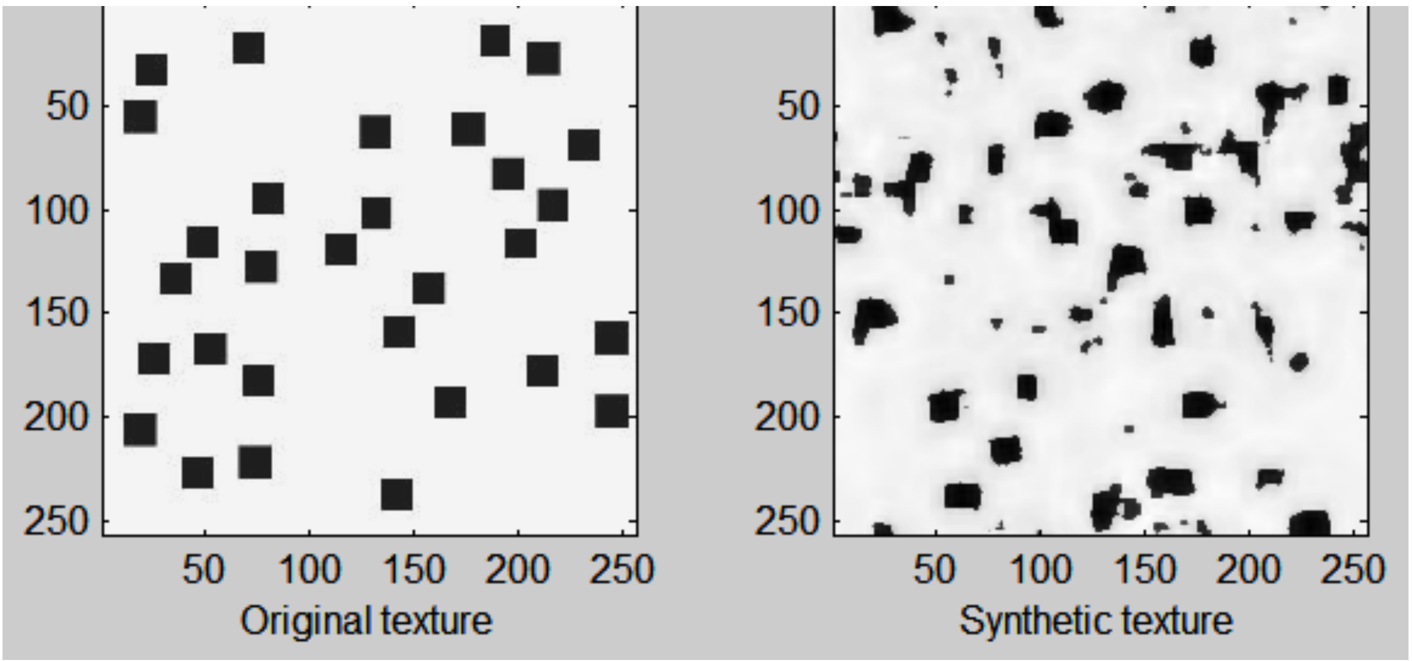
Filter bank



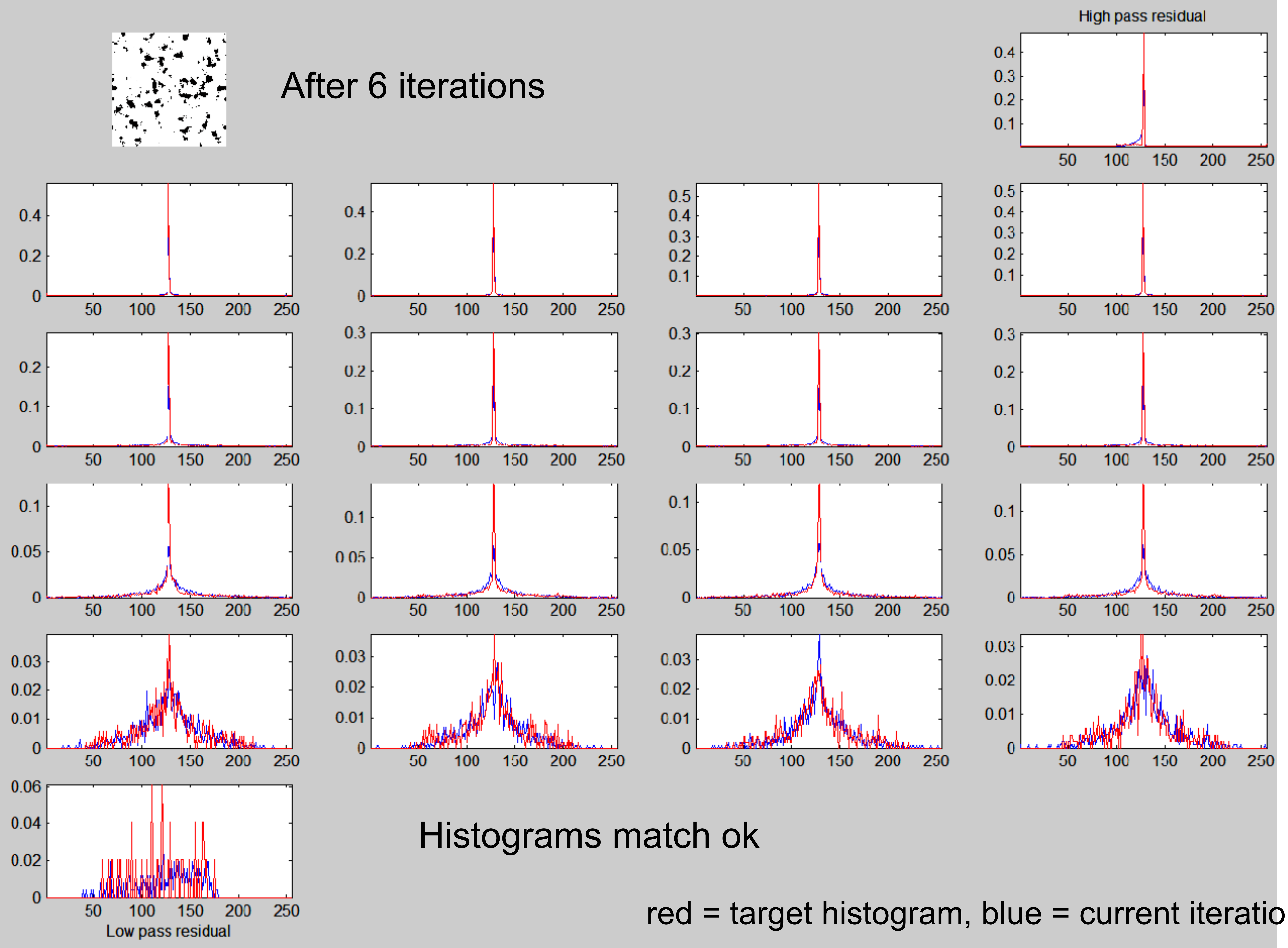
Why does it work? (sort of)

The black and white blocks appear by thresholding (f) a blobby image





Why does it work? (sort of)

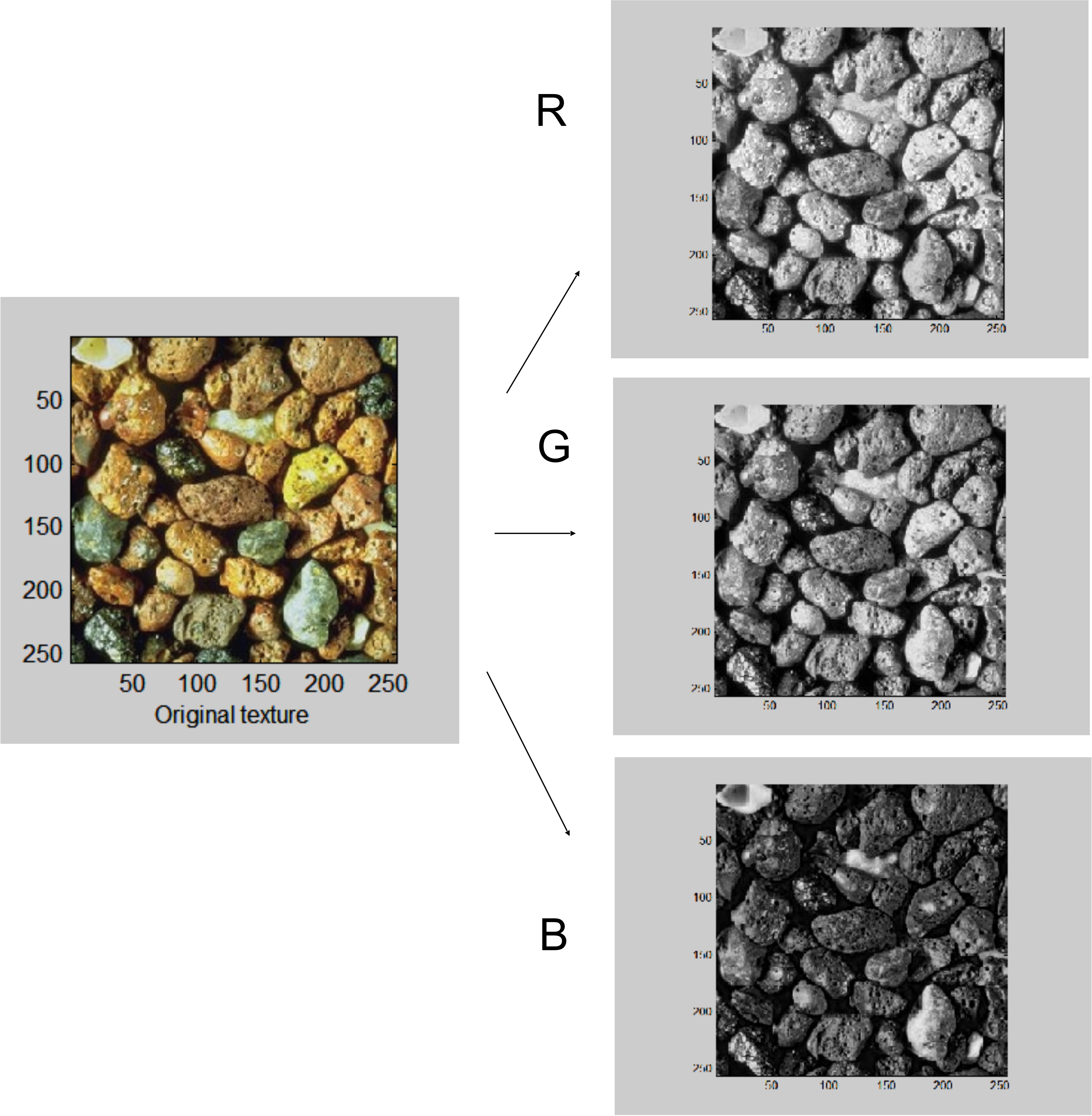


After 6 iterations

Histograms match ok

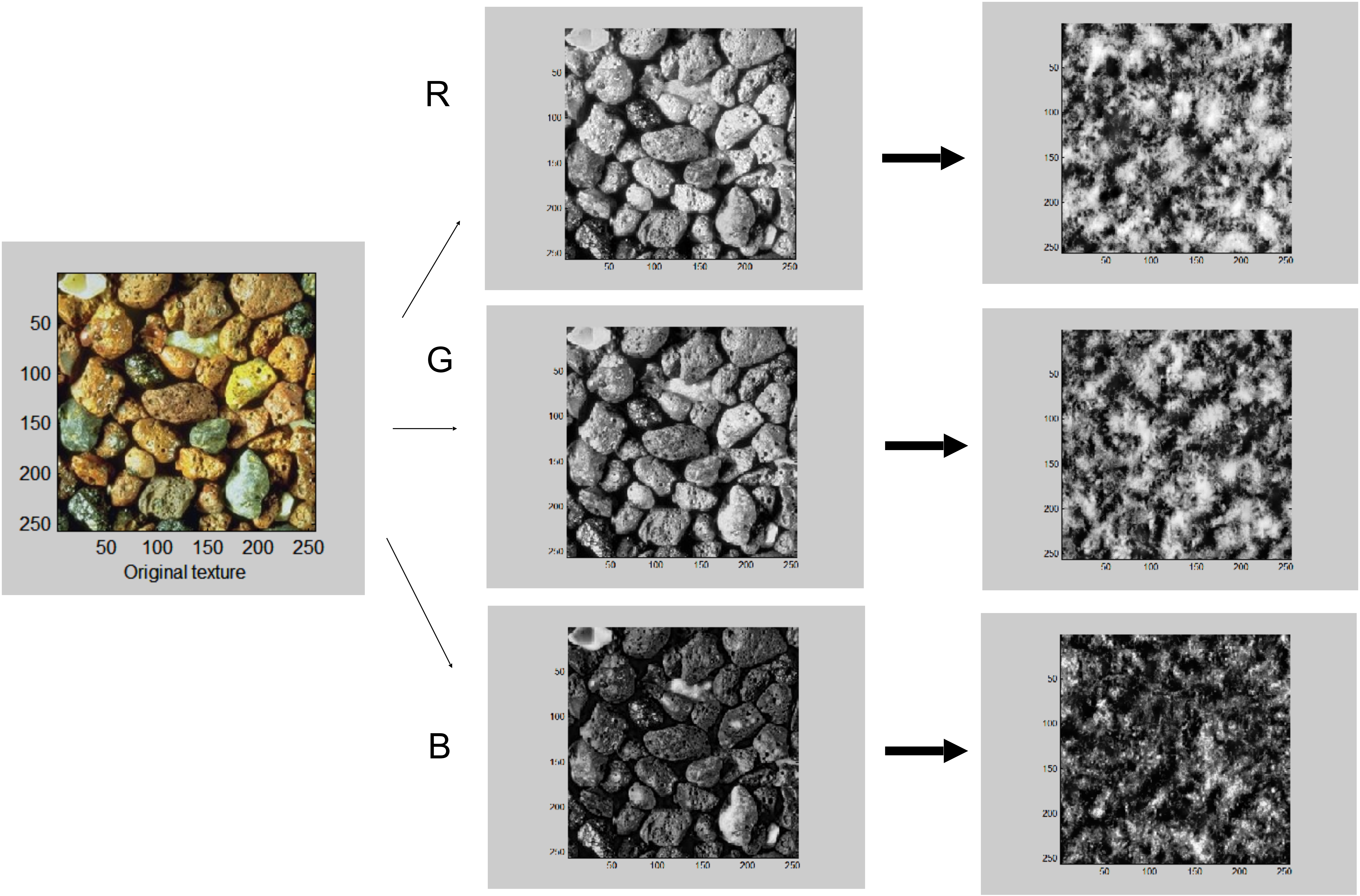
red = target histogram, blue = current iteration

Color textures



Three textures

Color textures



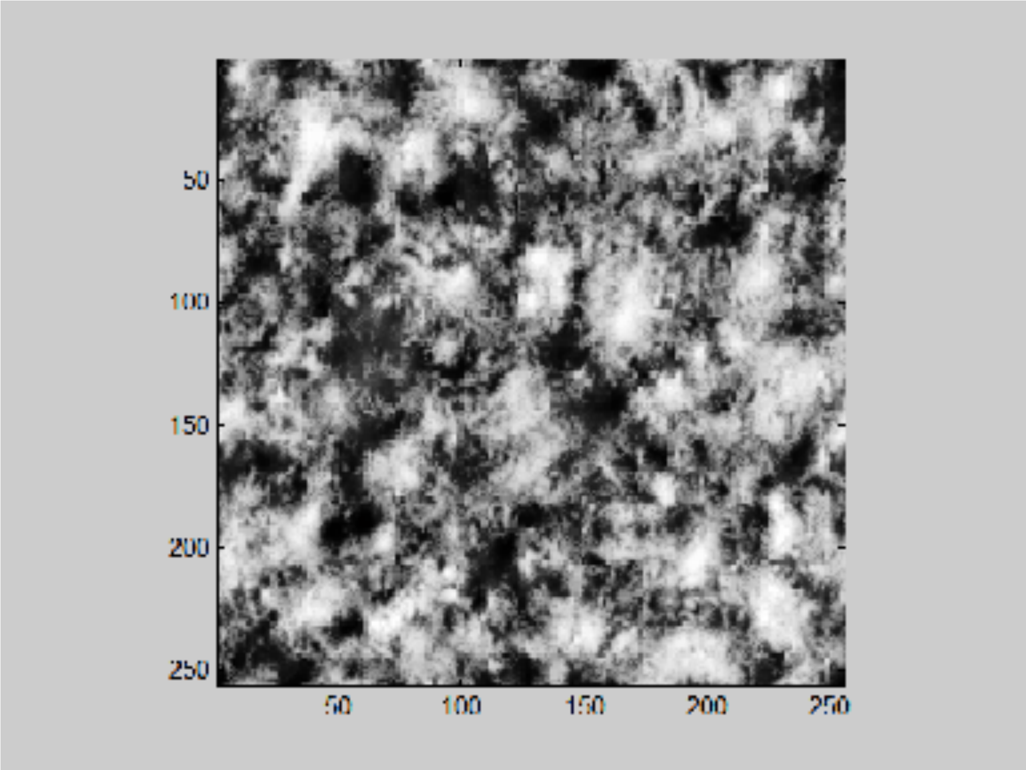
R

G

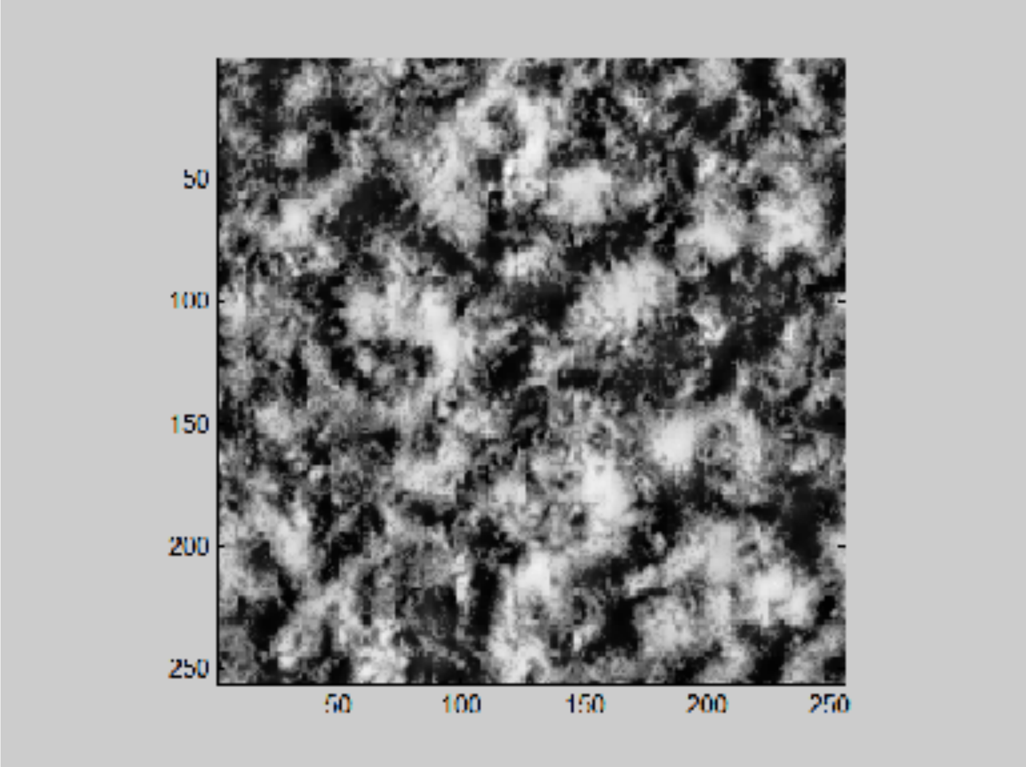
B

Color textures

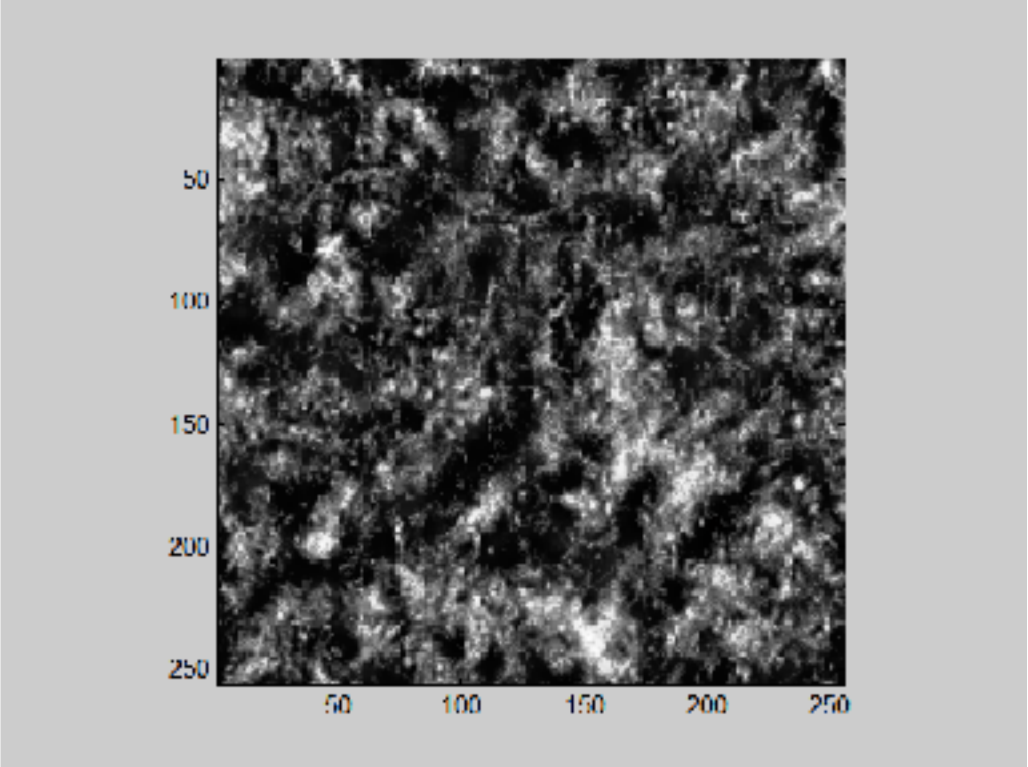
R



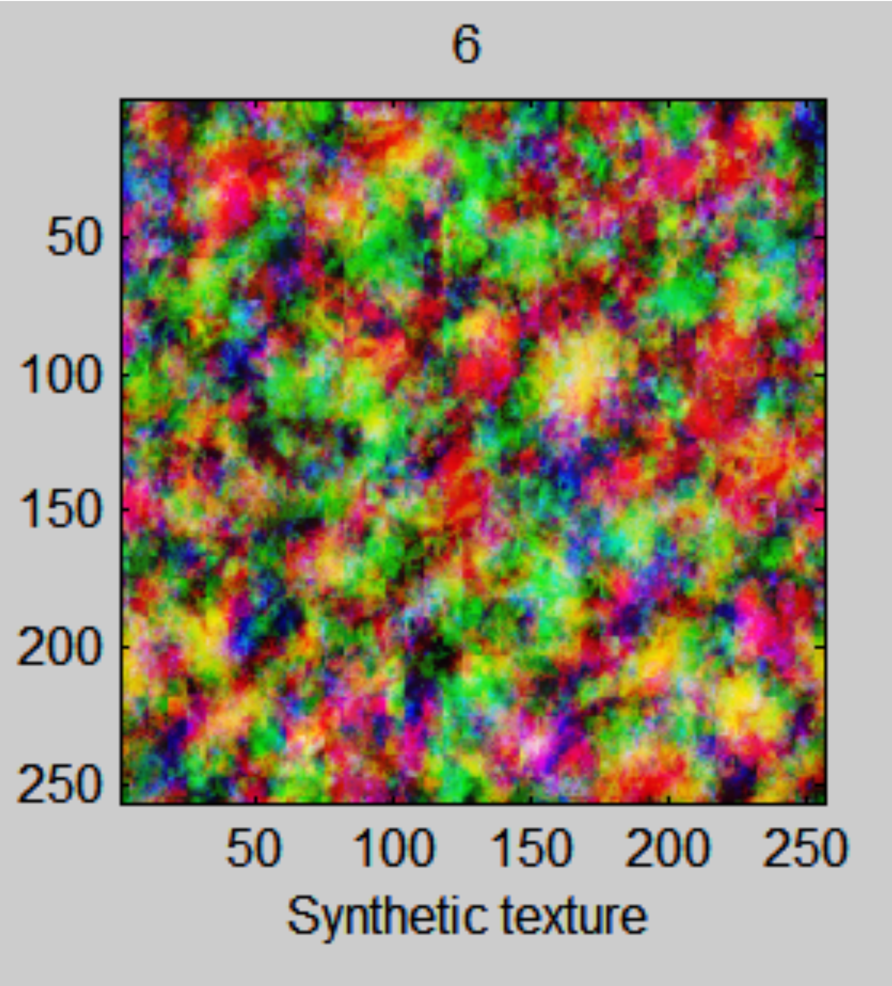
G



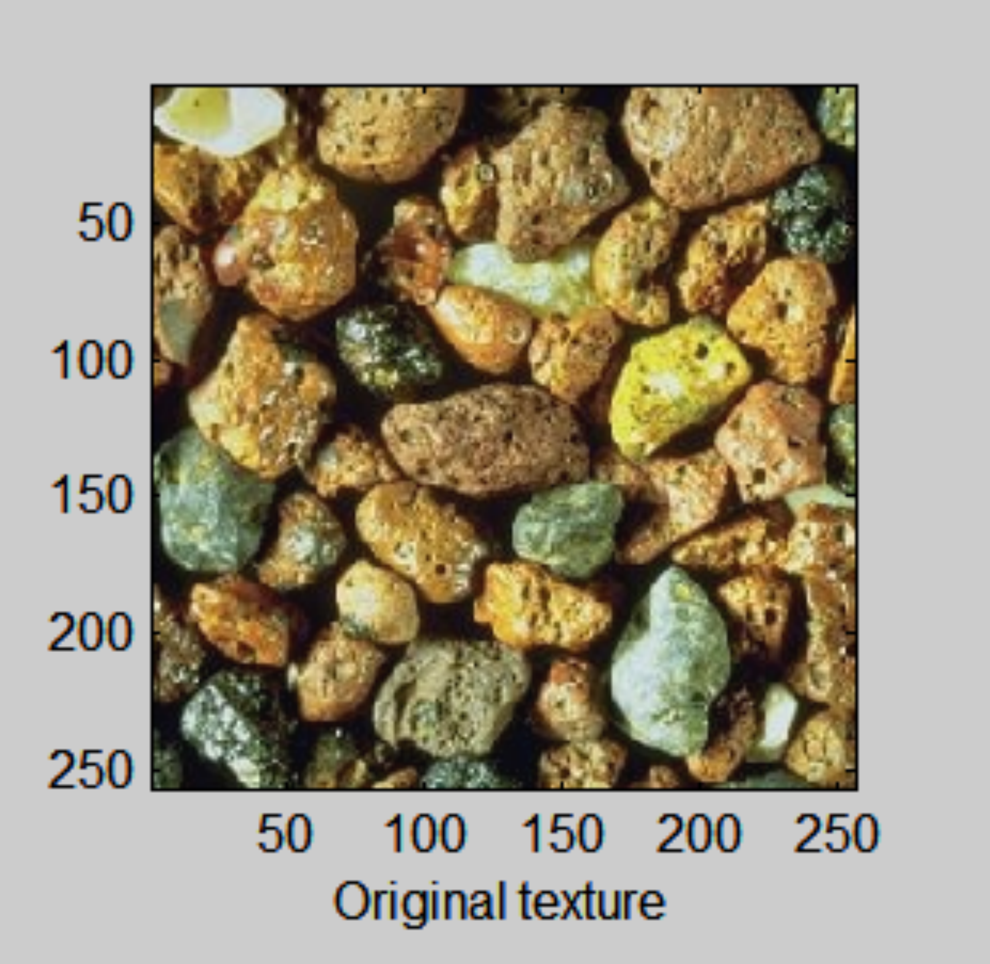
B



This does not work



Synthetic texture

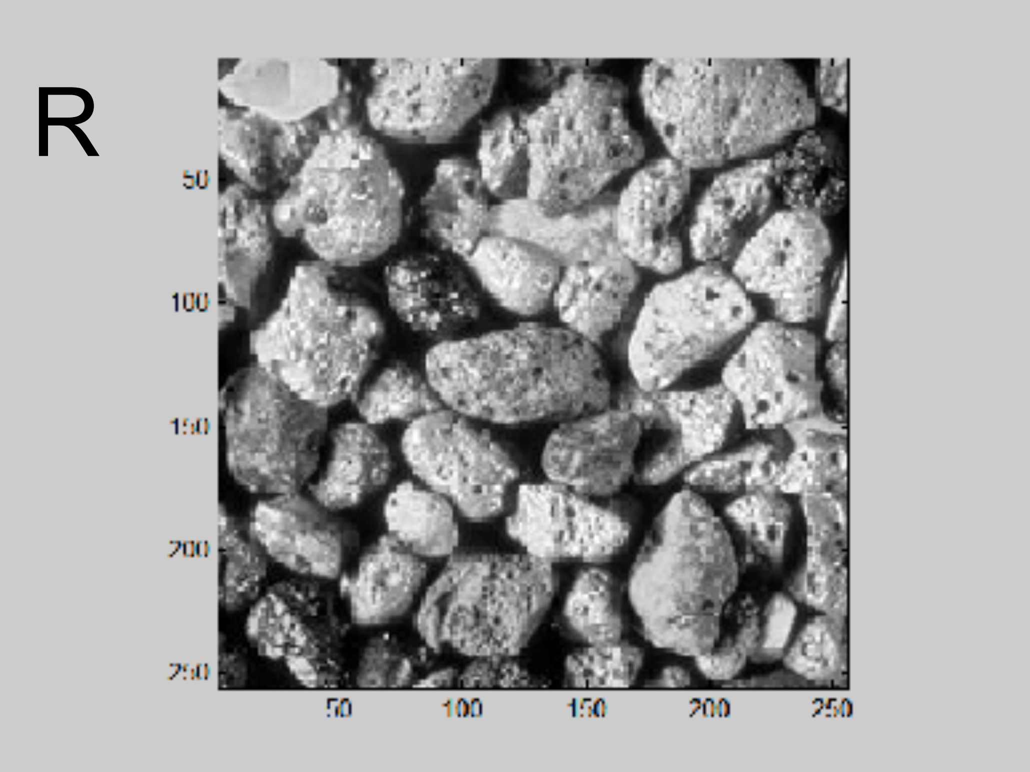


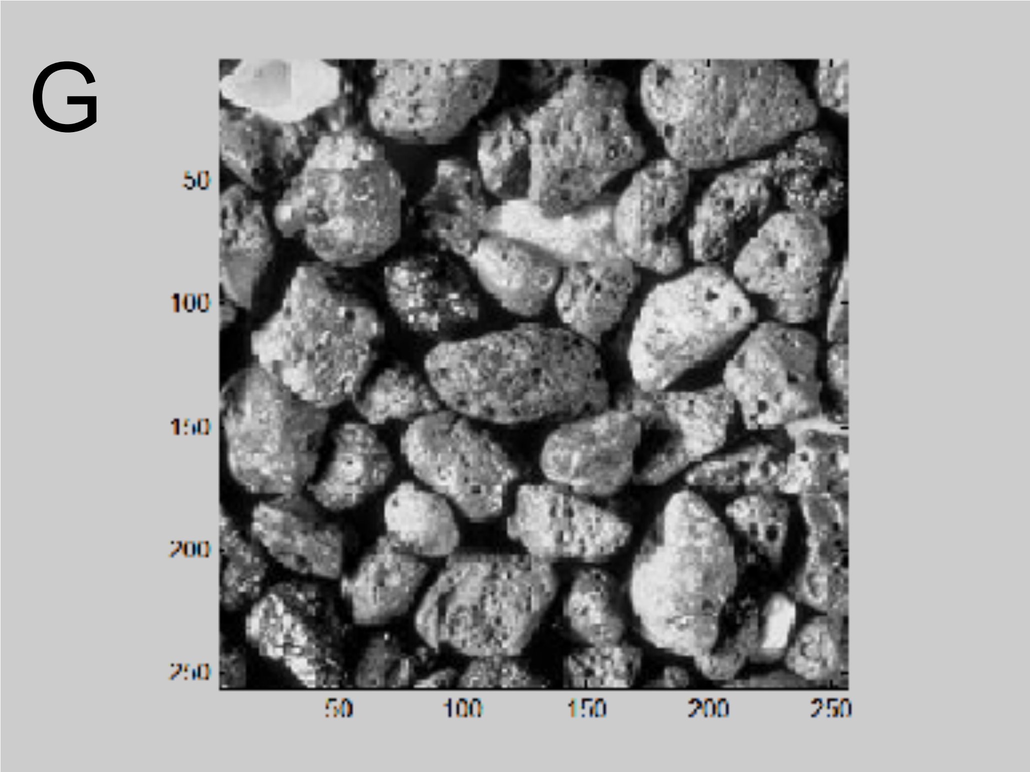
Original texture

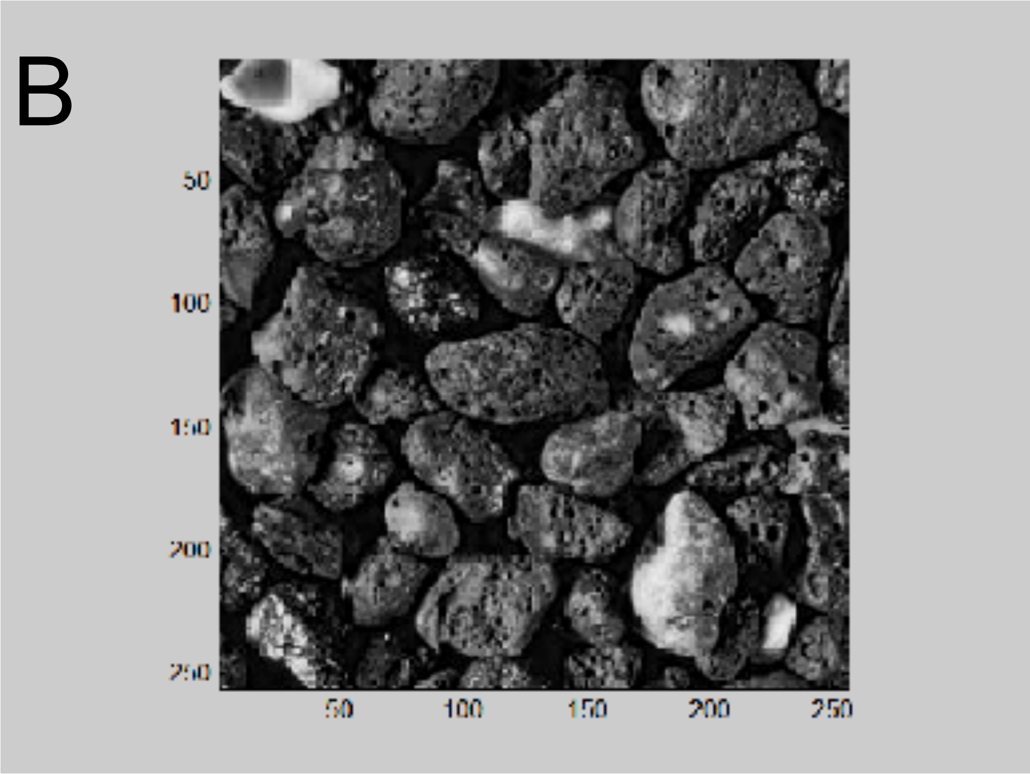
Color textures

Problem: we create new colors not present in the original image.

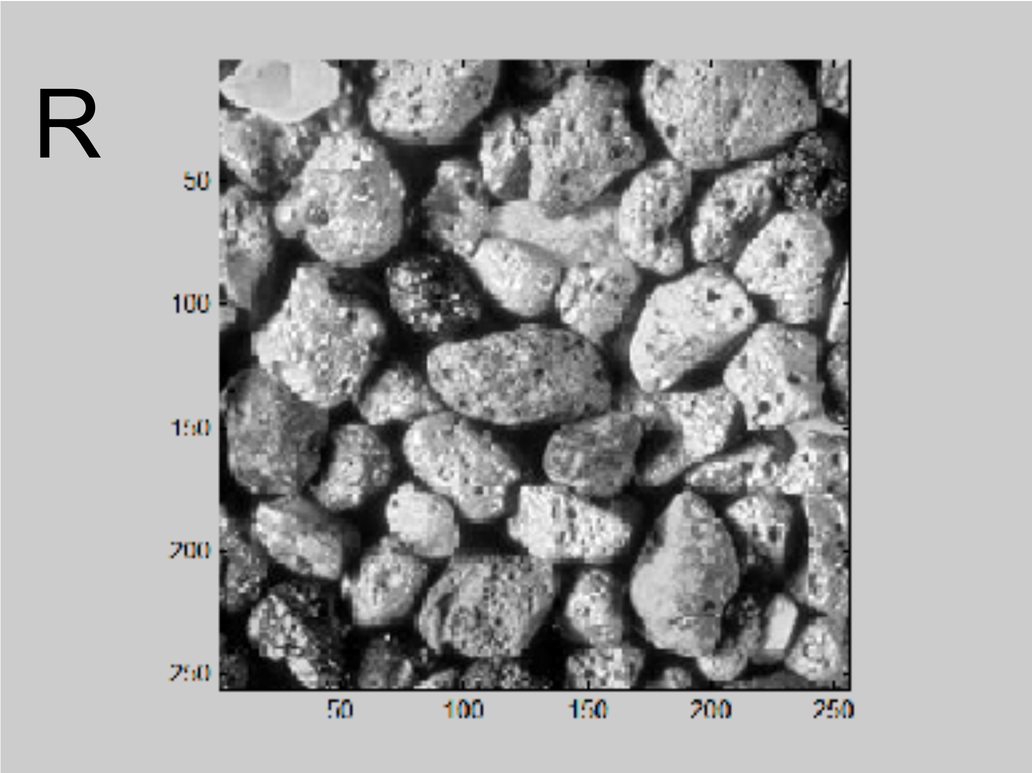
Why? Color channels are not independent.

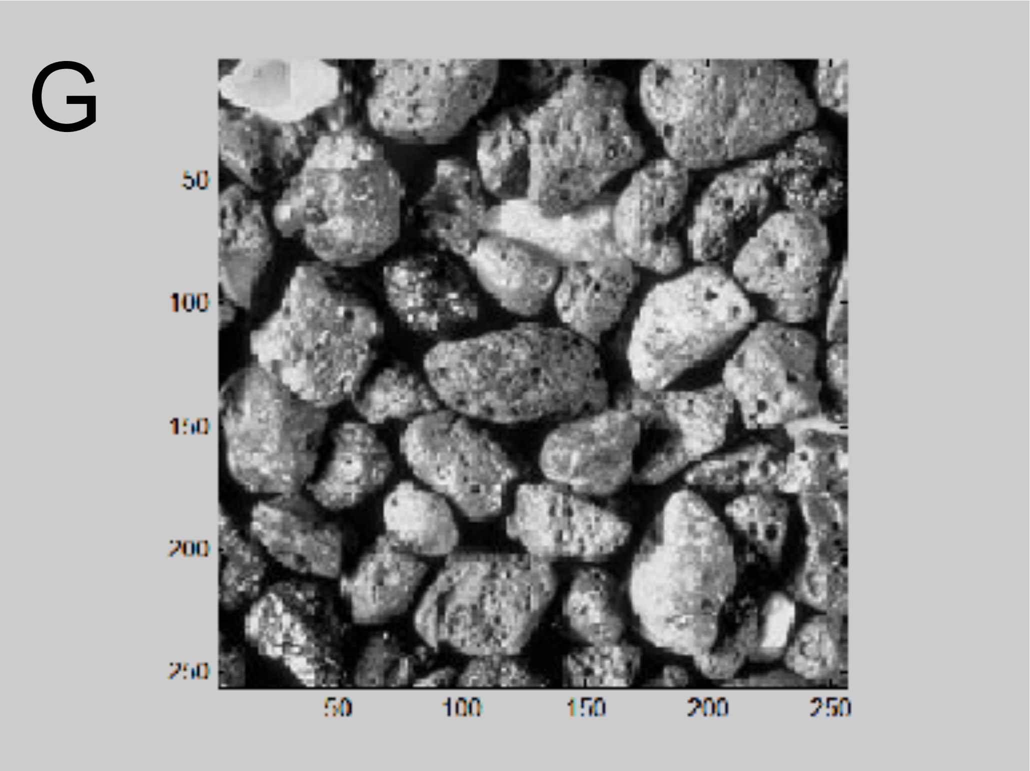

R

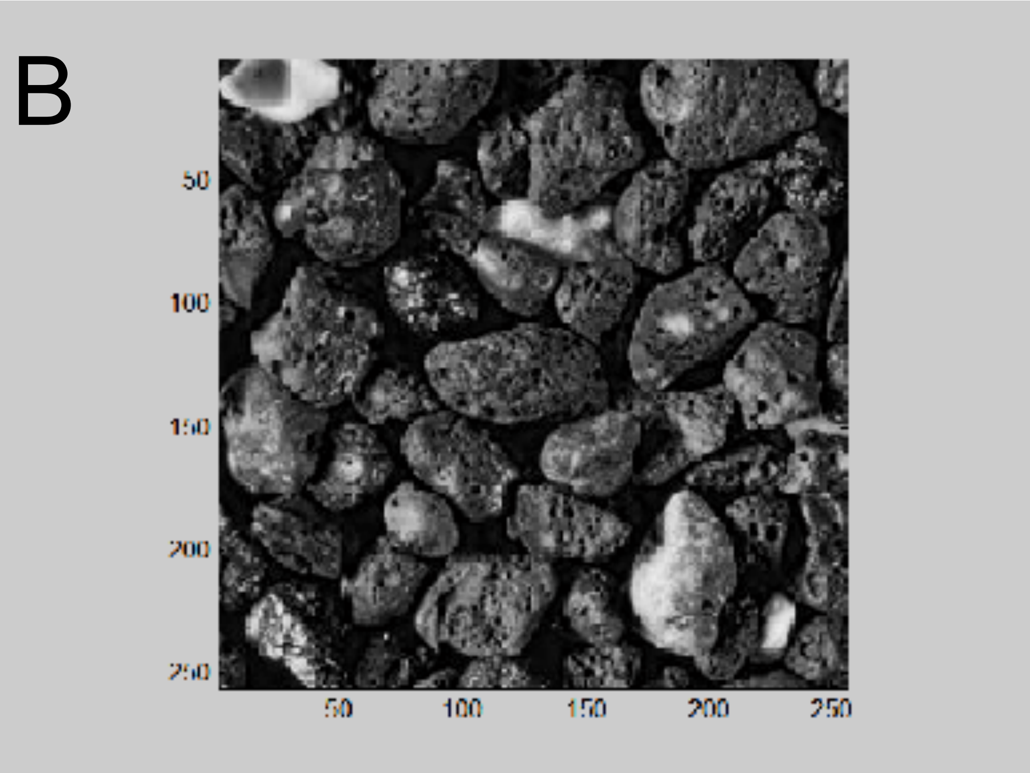

G


B

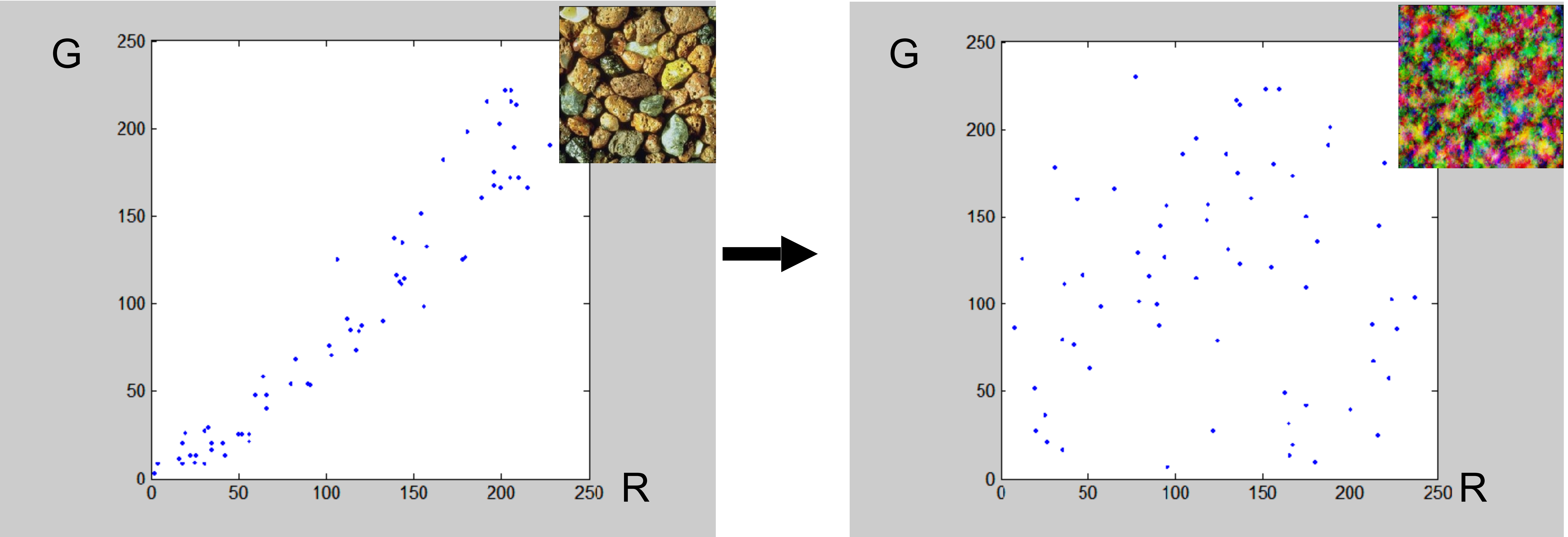
PCA and decorrelation





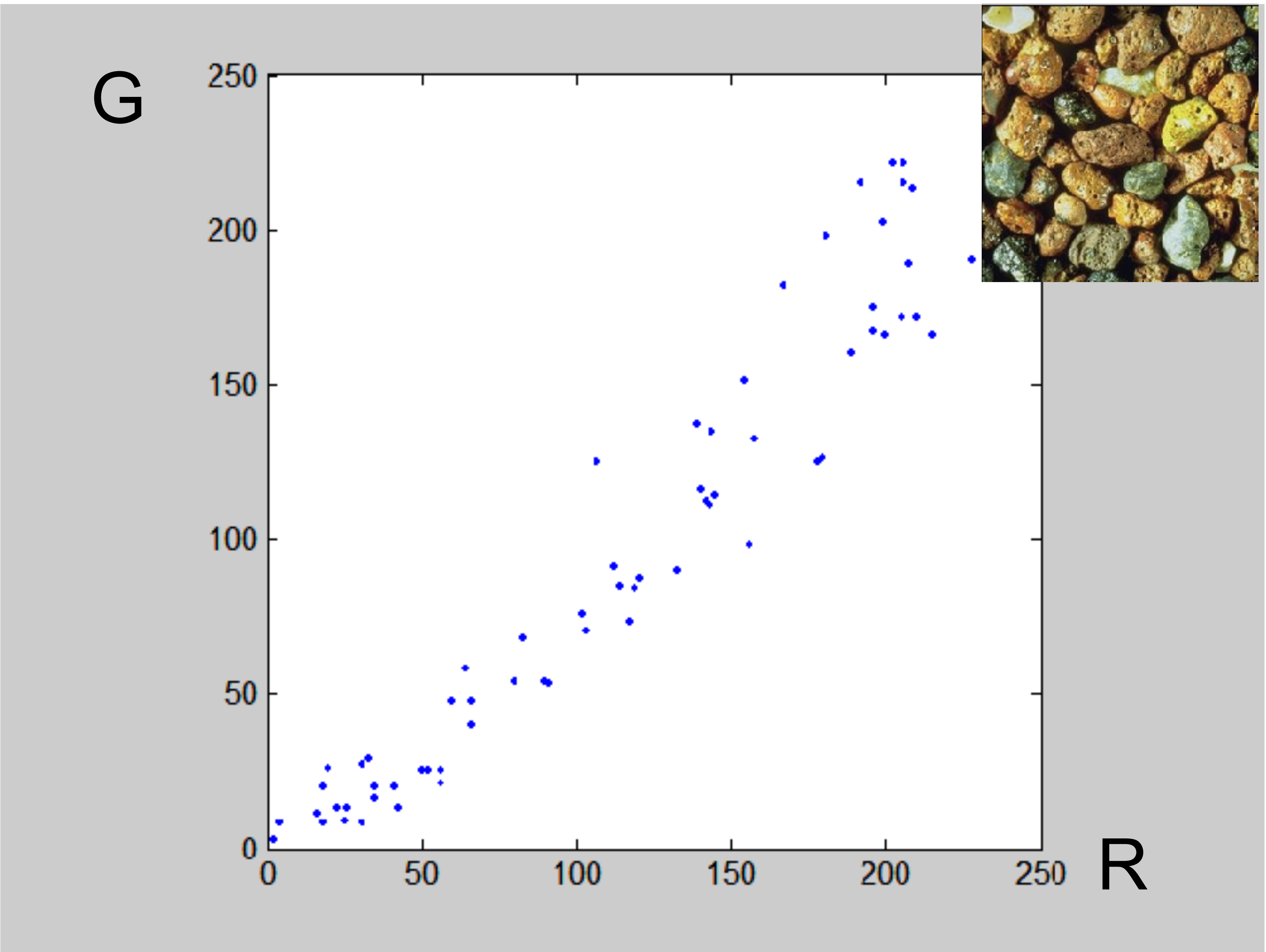


In the original image, R and G are correlated, but, after synthesis,…

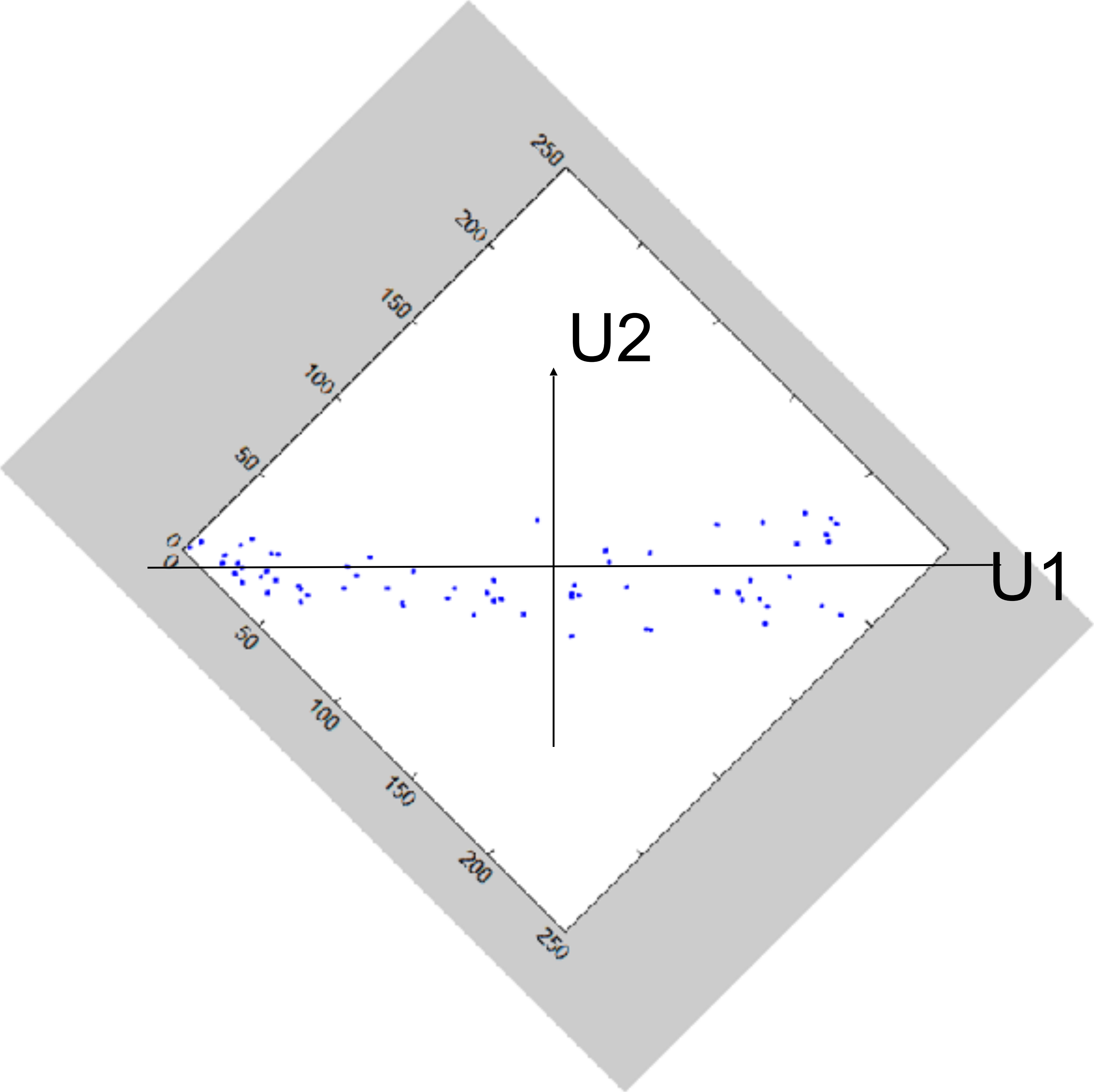


PCA and decorrelation

The texture synthesis algorithm assumes that the channels are independent.
What we want to do is some rotation

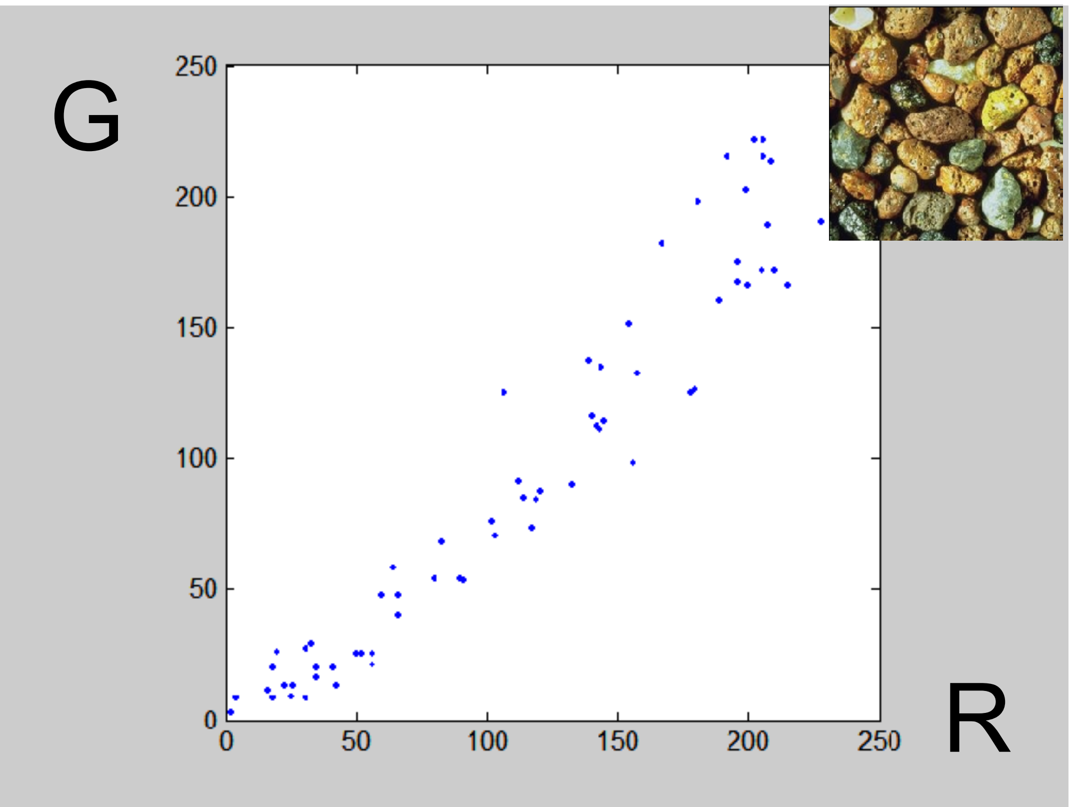


Rotation



See that in this rotated space, if I specify one coordinate the other remains unconstrained.

SVD and decorrelation



correlation(R,G)

$$C = \begin{array}{ccc} 1.0000 & 0.9303 & 0.6034 \\ 0.9303 & 0.9438 & 0.6620 \\ 0.6034 & 0.6620 & 0.5569 \end{array}$$

SVD finds the principal directions of variation of the data.
It gives a decomposition of the covariance matrix as:

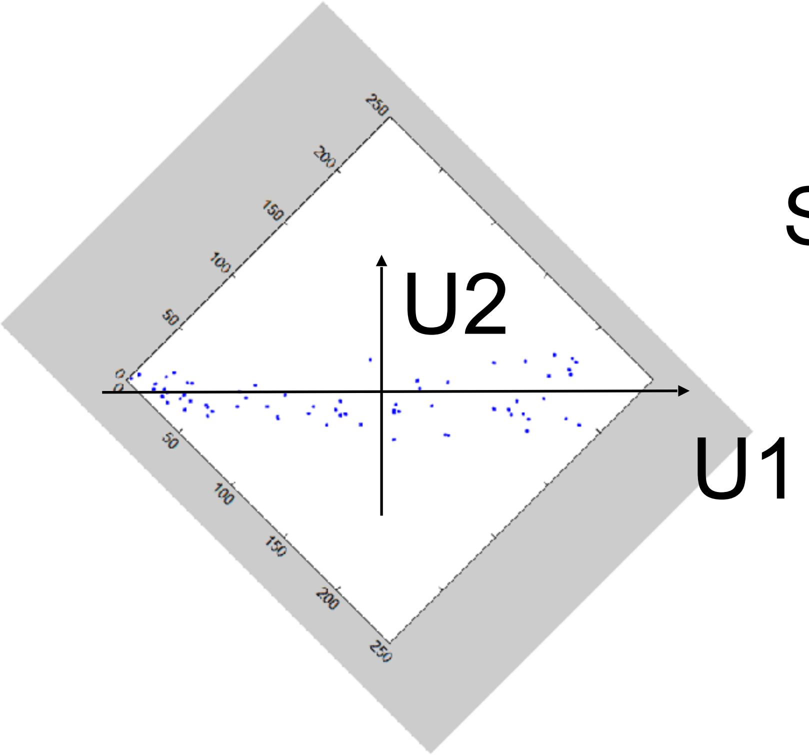
$$C = U\,S\,V'$$

$$V = \begin{array}{ccc} -0.6347 & 0.6072 & 0.4779 \\ -0.6306 & -0.0496 & -0.7745 \\ -0.4466 & -0.7930 & 0.4144 \end{array}$$

By transforming the original data (RGB) using D we get:

$$\underbrace{\boxed{\begin{array}{c} U1 \\ U2 \\ U3 \end{array}}}_{3 \times \text{Npixels}} = \underbrace{\boxed{V'}}_{3 \times 3} \; \underbrace{\boxed{\begin{array}{c} R \\ G \\ B \end{array}}}_{3 \times \text{Npixels}}$$

The new components (U1,U2,U3) are decorrelated.



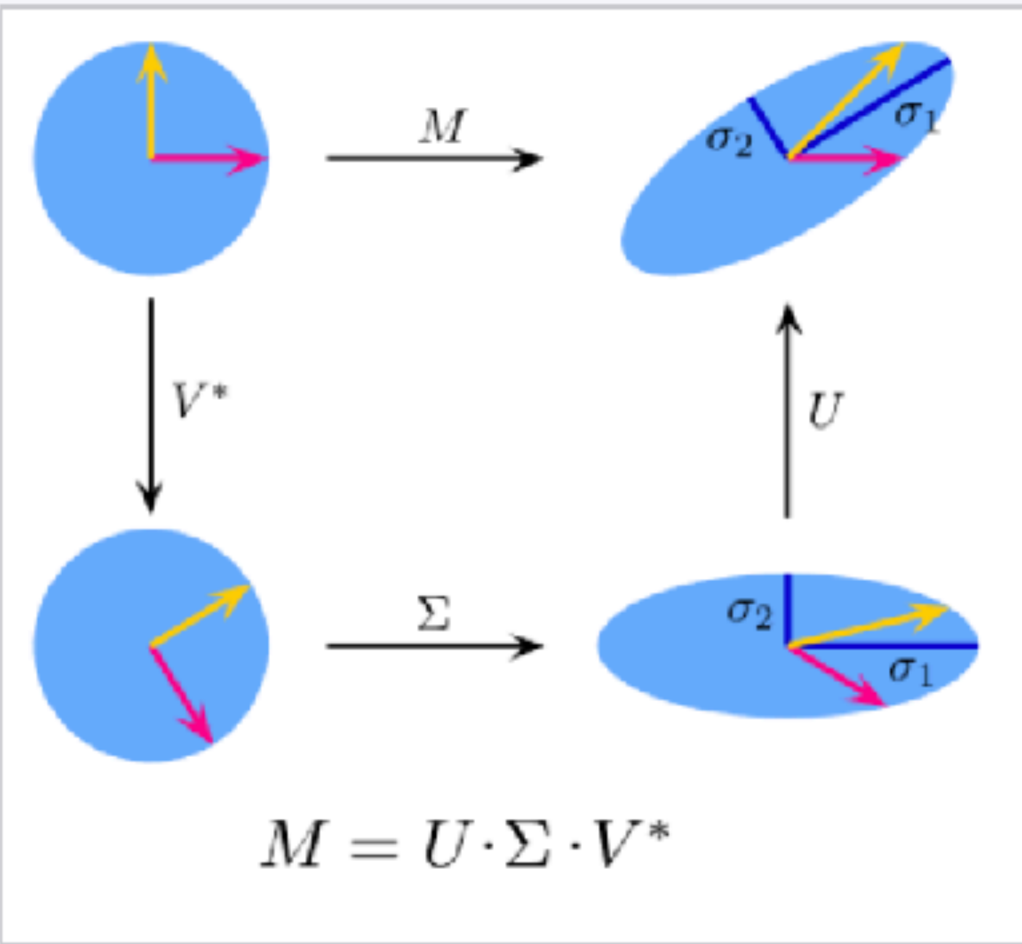


Illustration of the singular value decomposition **UΣV*** of a real 2x2 matrix **M**.

Top: The action of **M**, indicated by its effect on the unit disc *D* and the two canonical unit vectors e_1 and e_2.

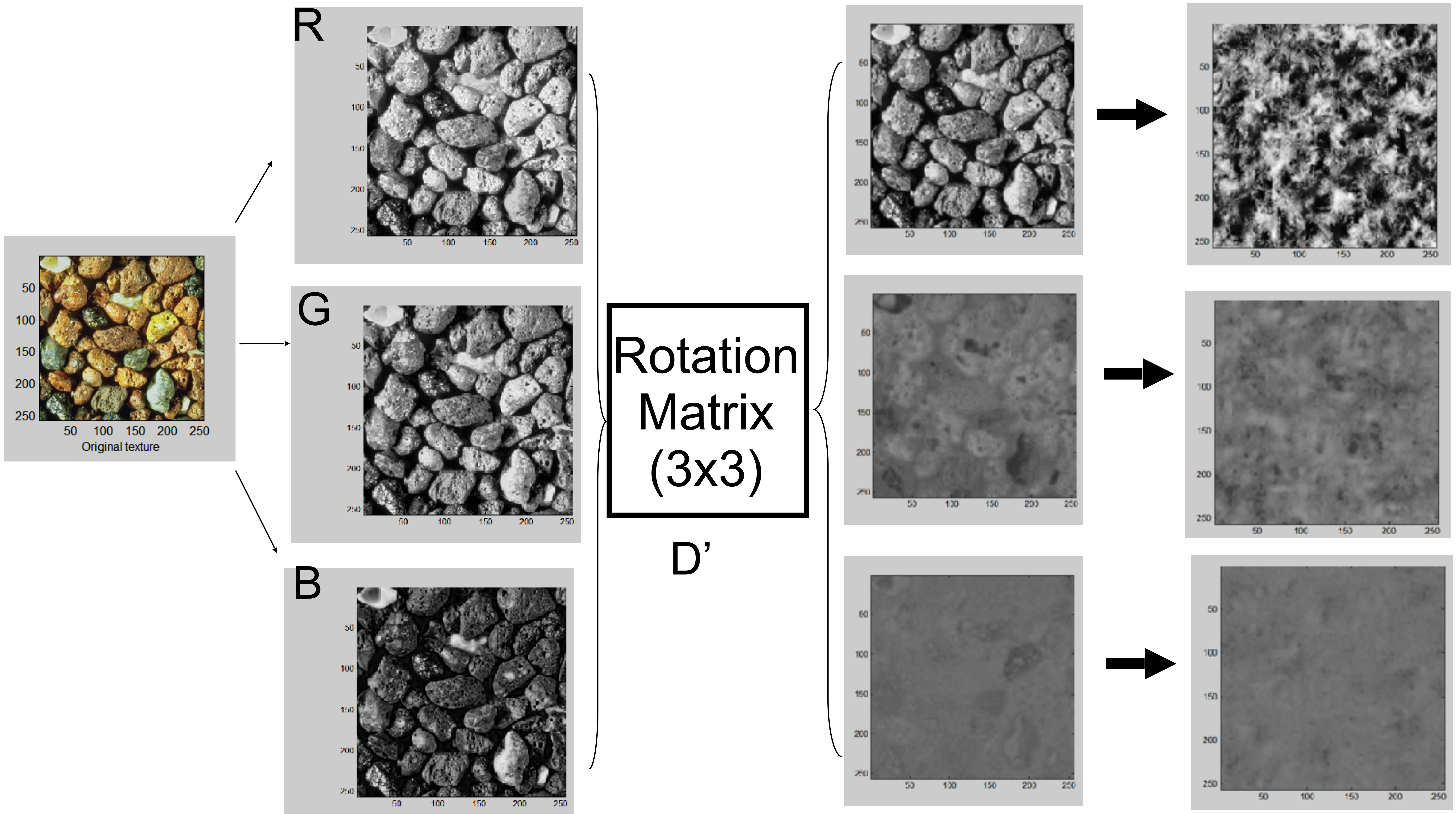
Left: The action of **V***, a rotation, on *D*, e_1, and e_2.

Bottom: The action of **Σ**, a scaling by the singular values σ_1 horizontally and σ_2 vertically.

Right: The action of **U**, another rotation.

https://en.wikipedia.org/wiki/Singular_value_decompositio

Color textures

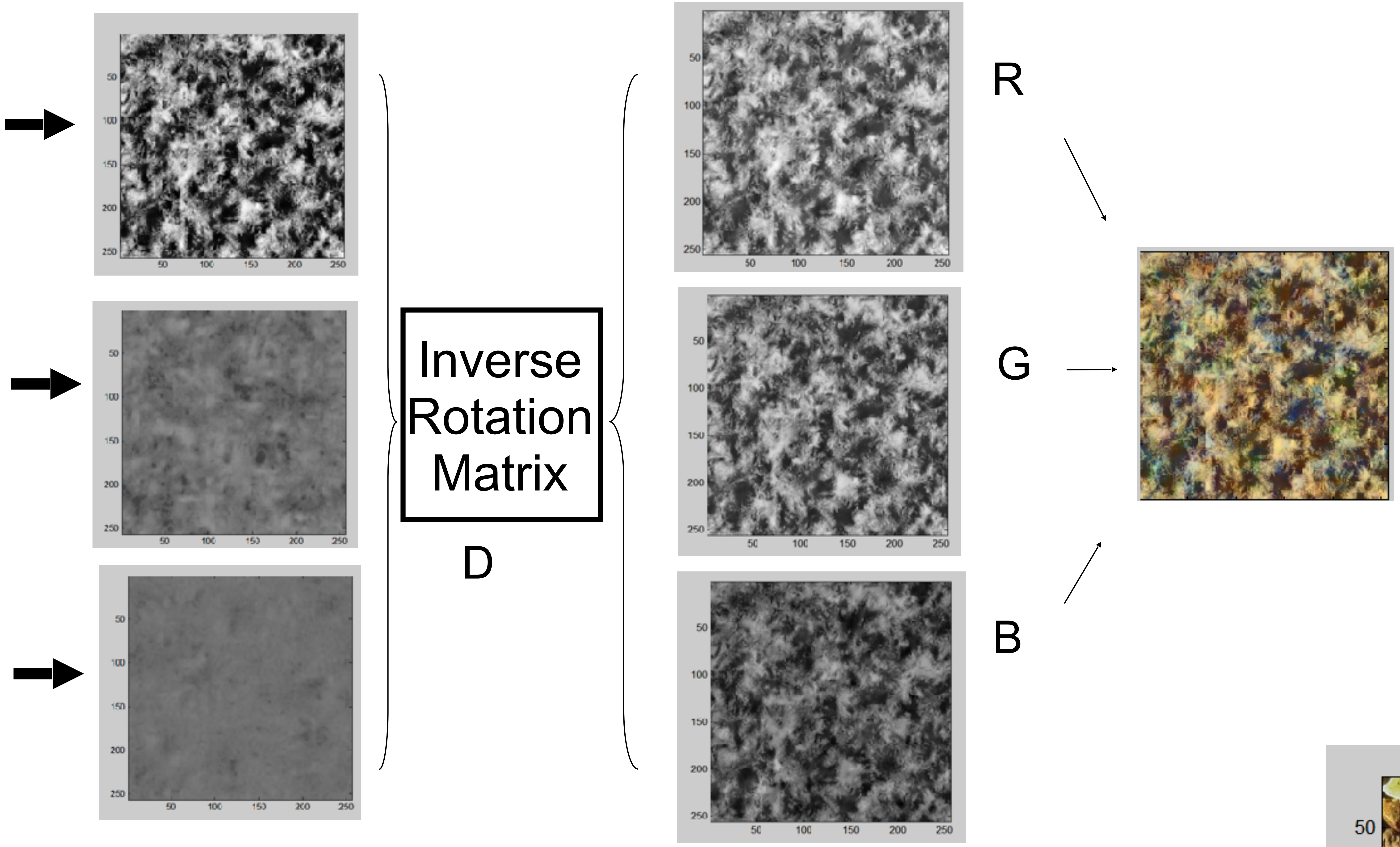


Rotation Matrix (3x3)

These three textures look similar (high dependency)

These three textures Look less similar (lower dependency)

Color textures



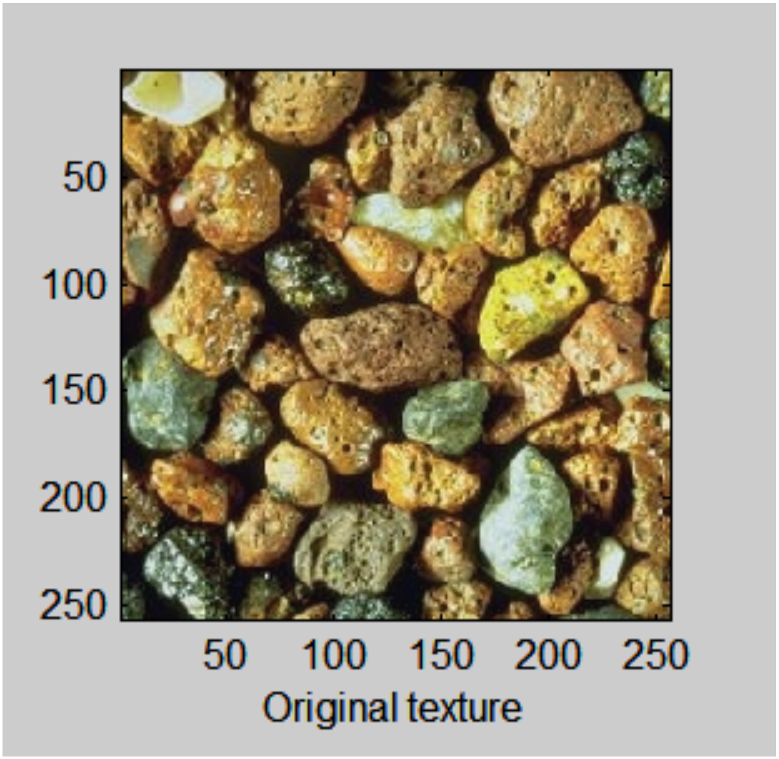
Inverse Rotation Matrix

D

R

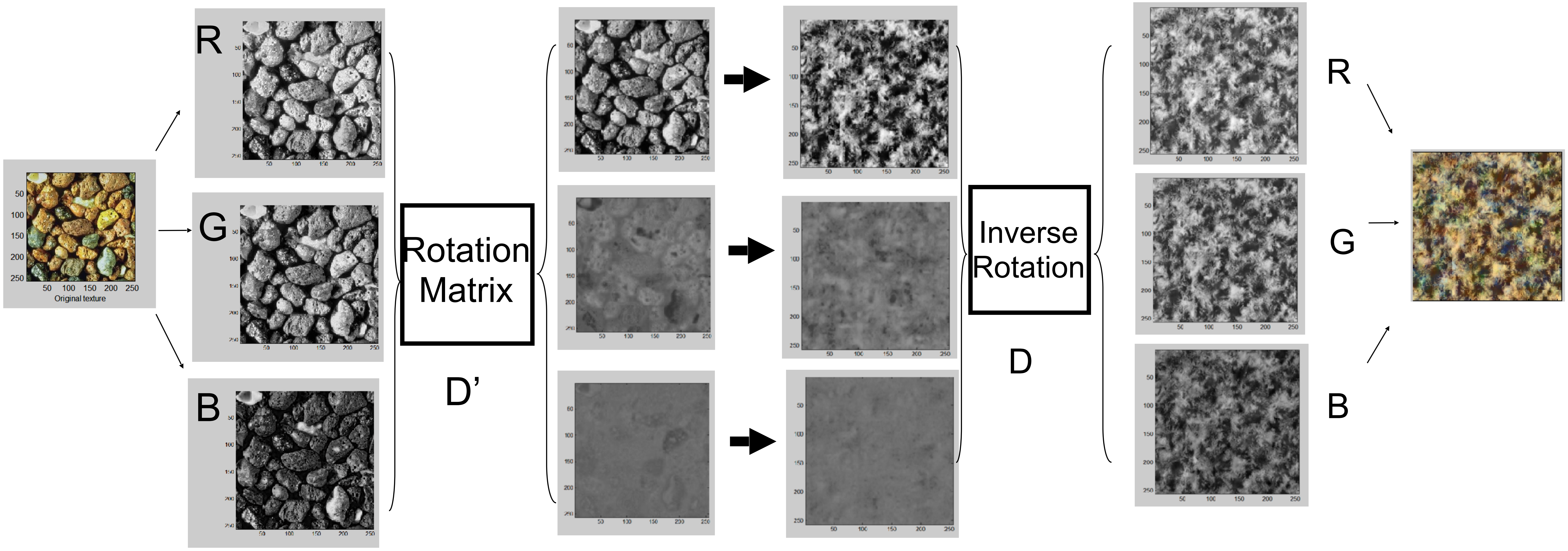
G

B



Original texture

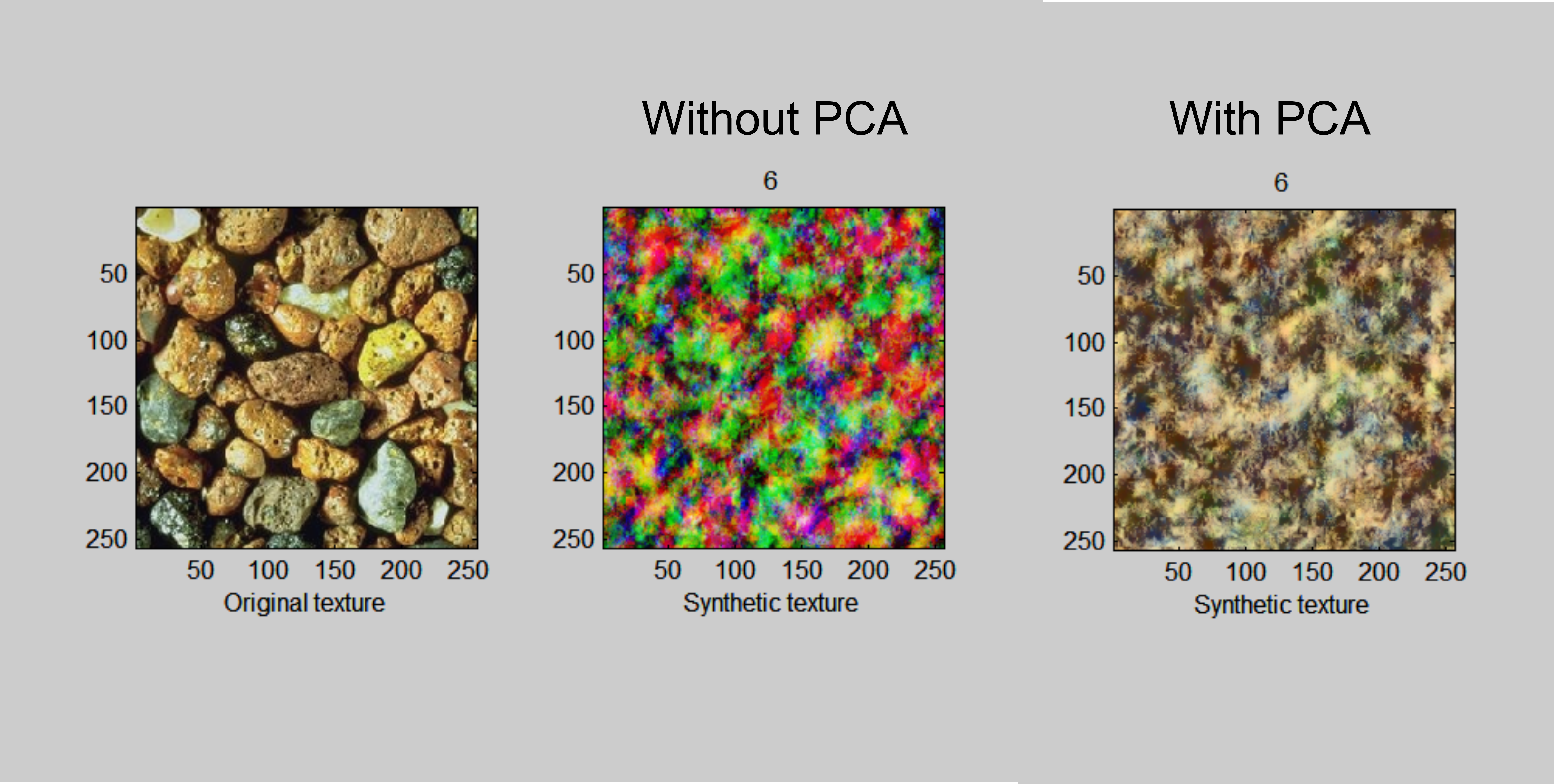
Color textures



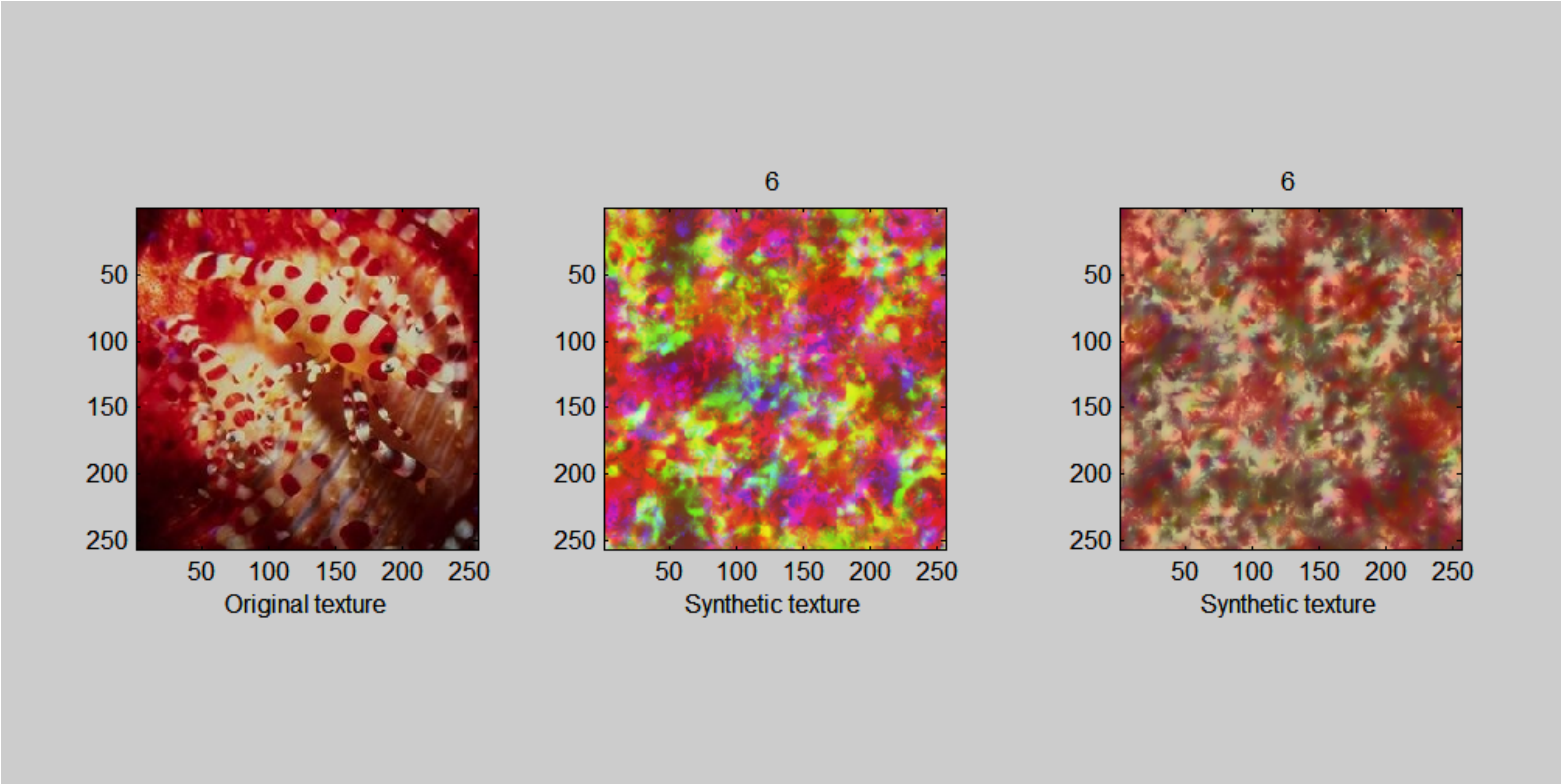
These three textures
look similar
(high dependency)

These three textures
Look less similar
(lower dependency)

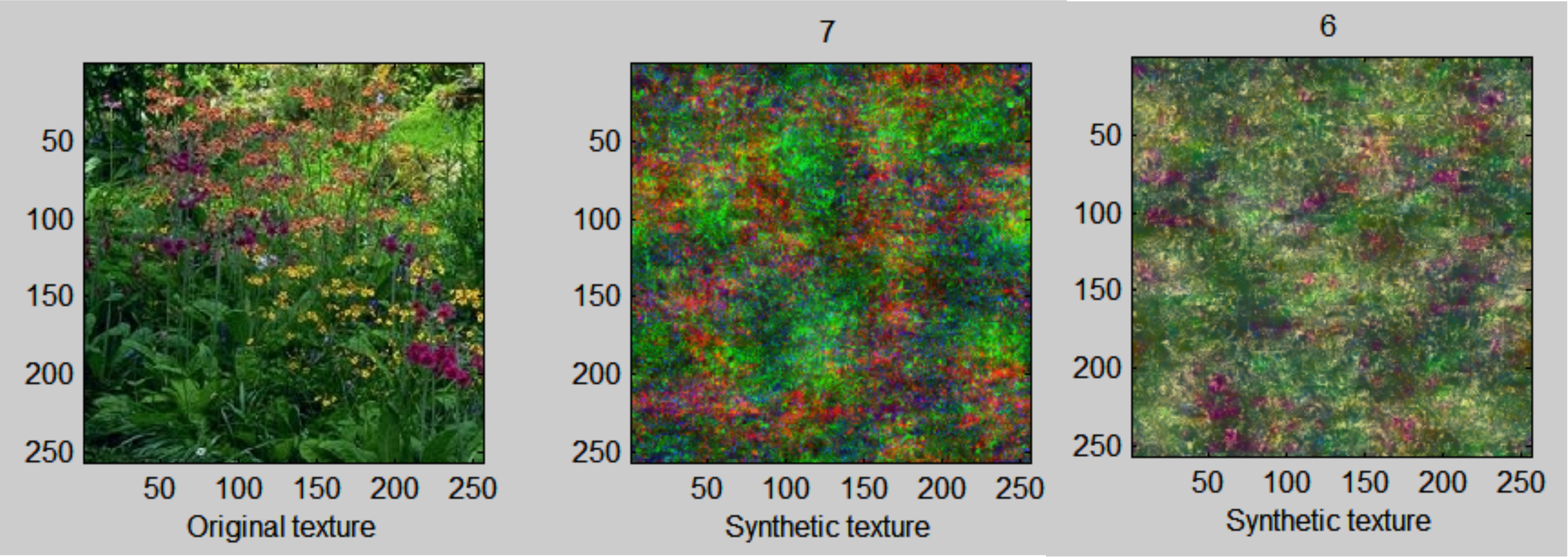
Color channels



Color channels



Color channels



Examples from the paper

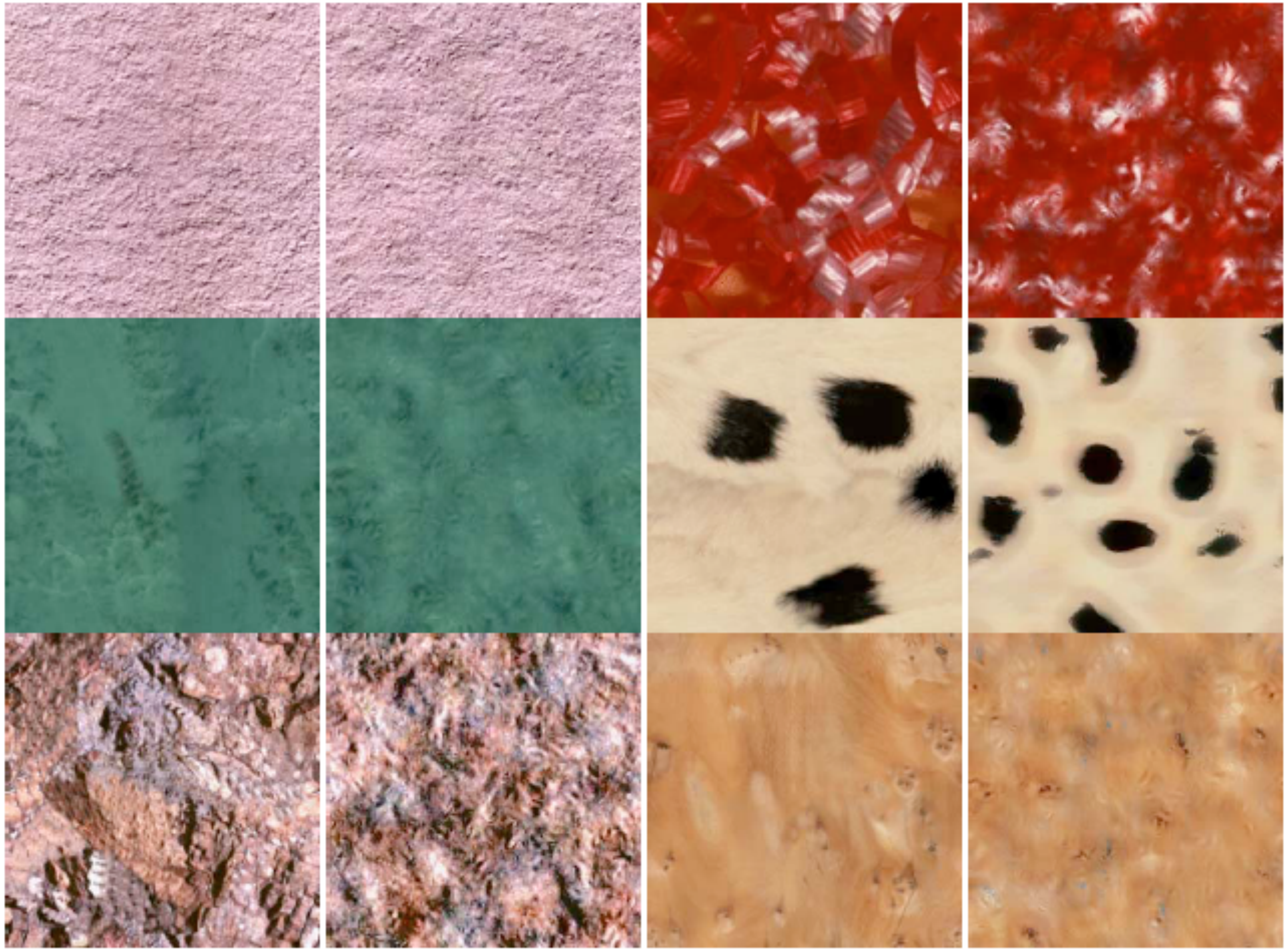


Figure 3: In each pair left image is original and right image is synthetic: stucco, iridescent ribbon, green marble, panda fur, slag stone, figured yew wood.

Heeger and Bergen, 1995

Examples from the paper

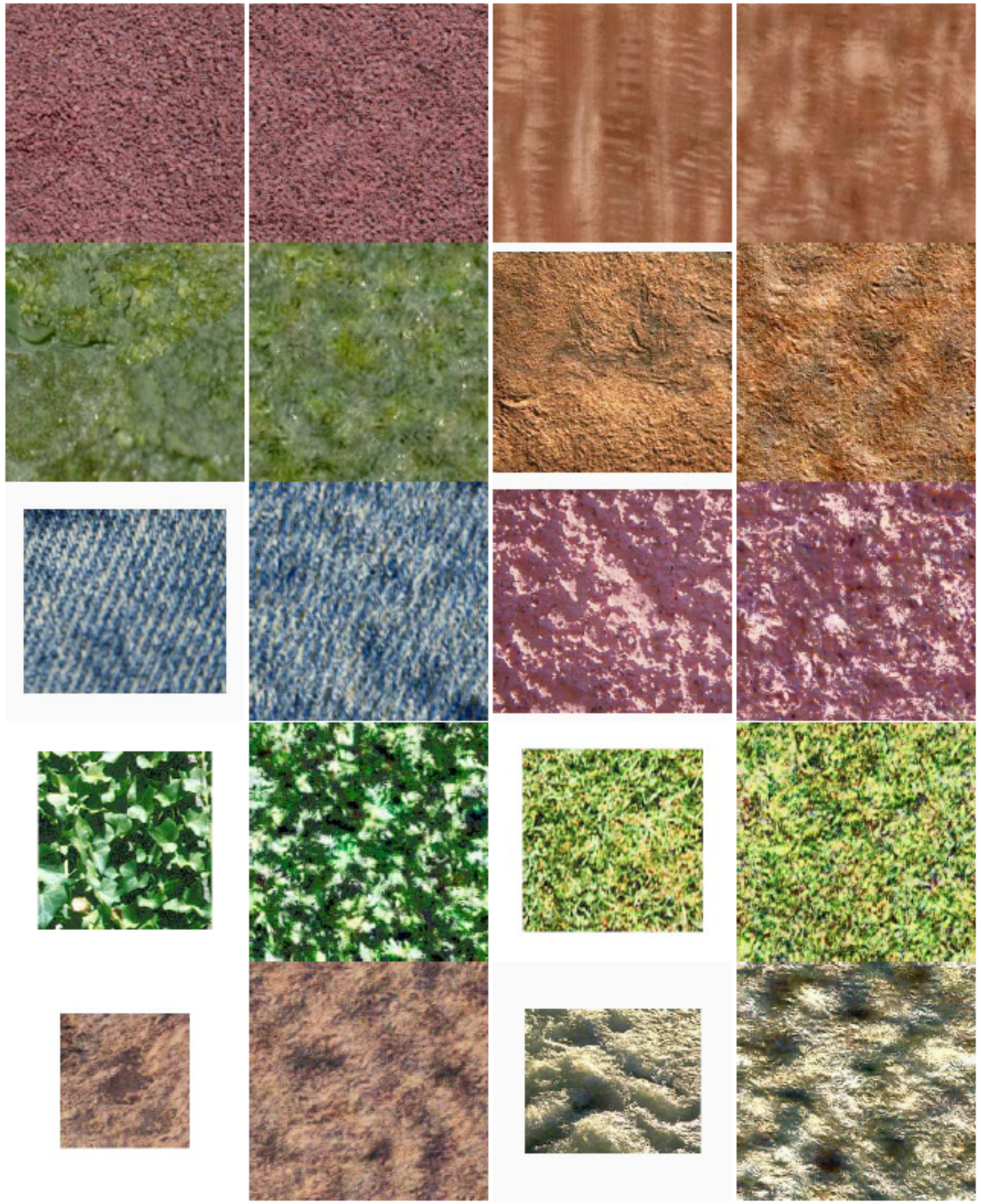
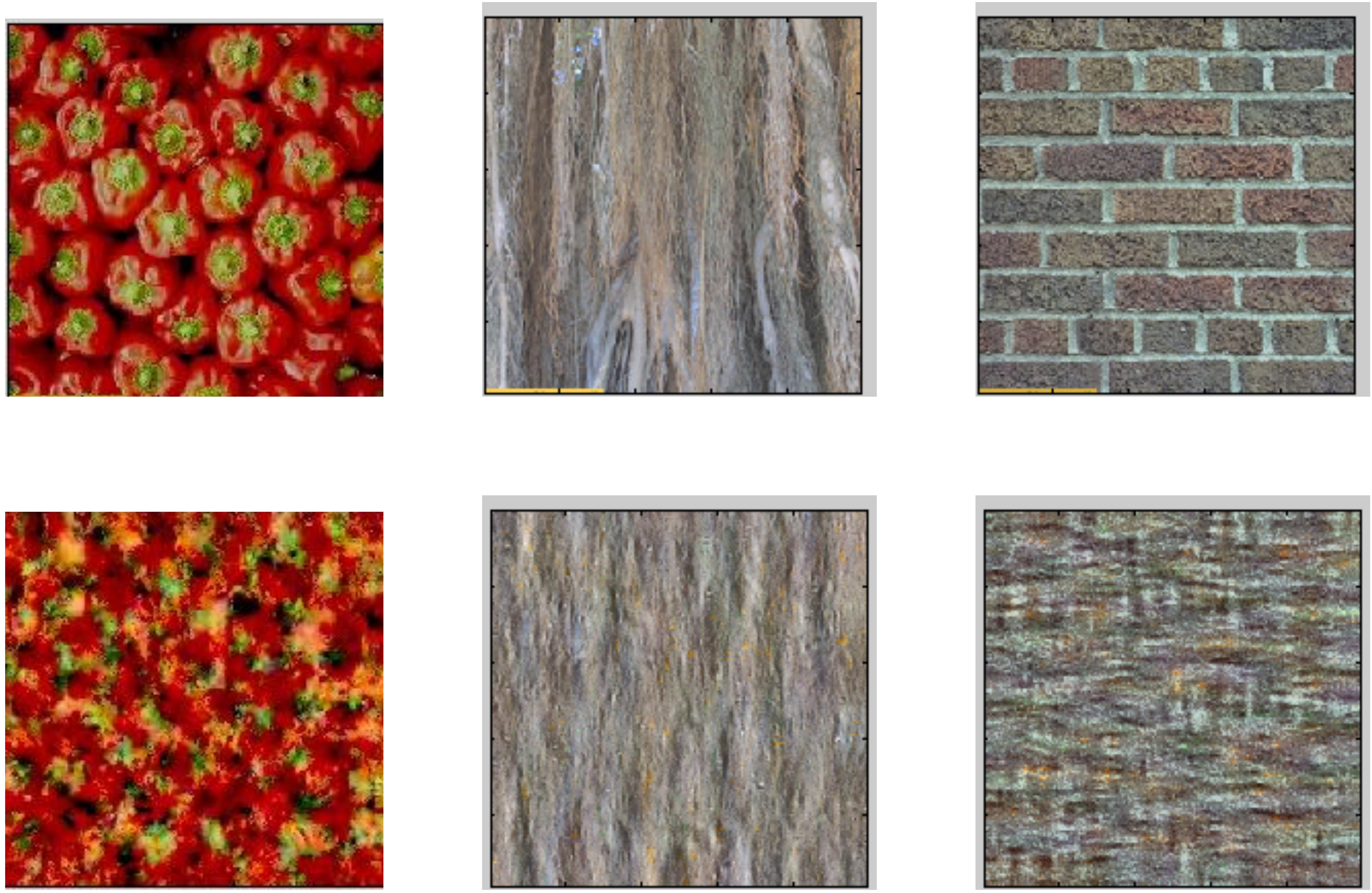


Figure 4: In each pair left image is original and right image is synthetic: red gravel, figured sepele wood, brocolli, bark paper, denim, pink wall, ivy, grass, sand, surf.

Examples not from the paper



Input texture

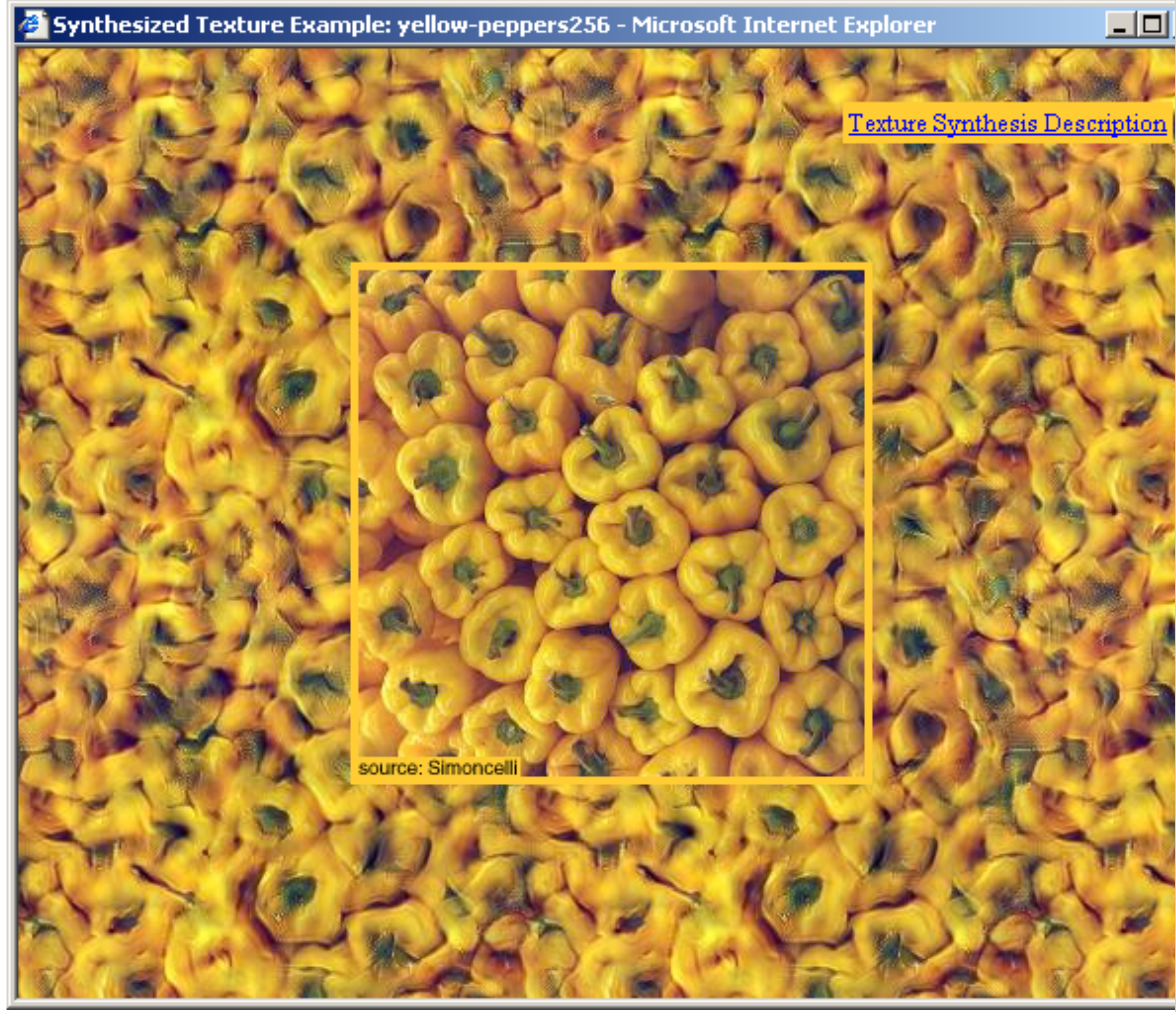
Synthetic texture

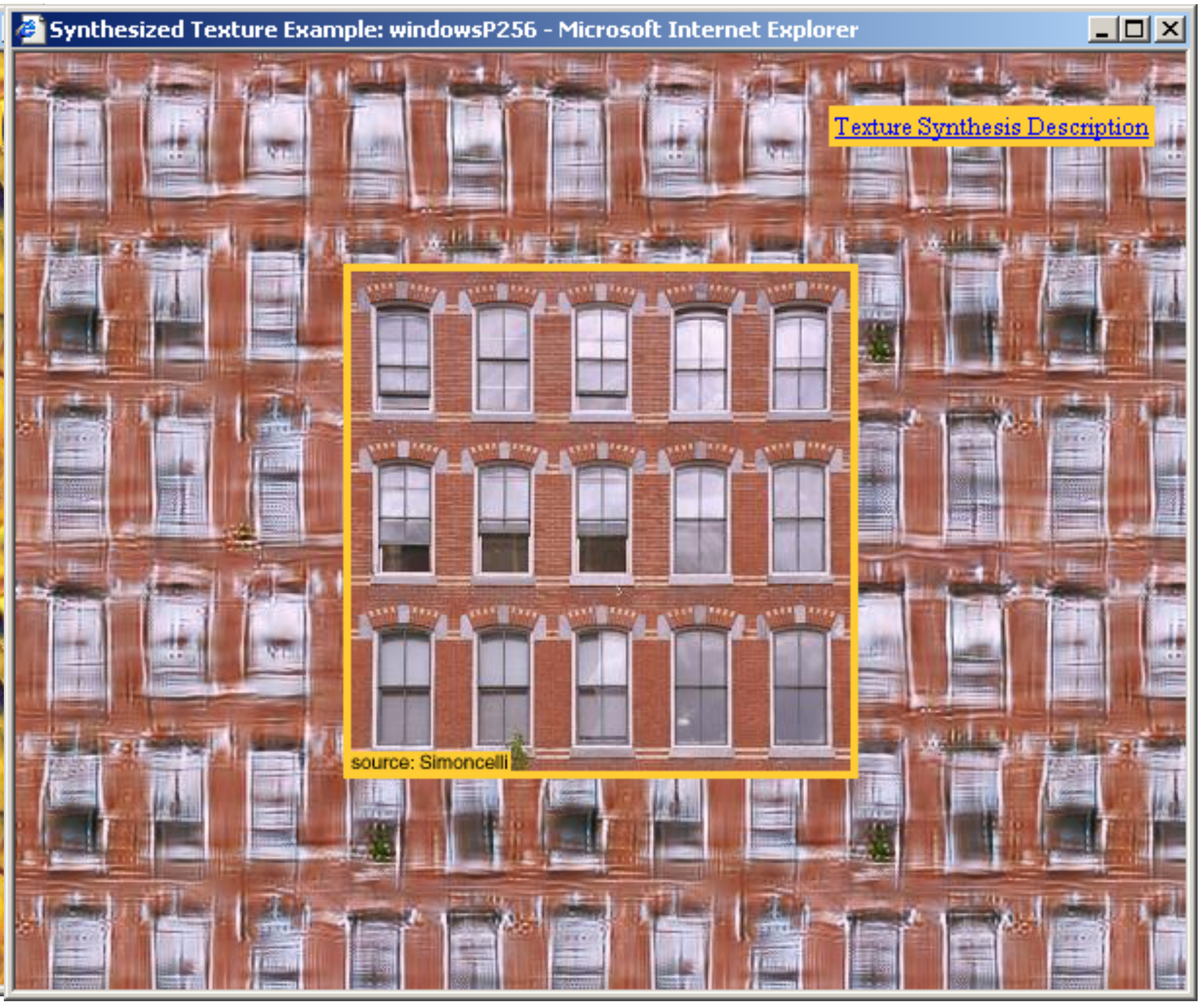
But, does it really work even when it seems to work?

Portilla and Simoncelli

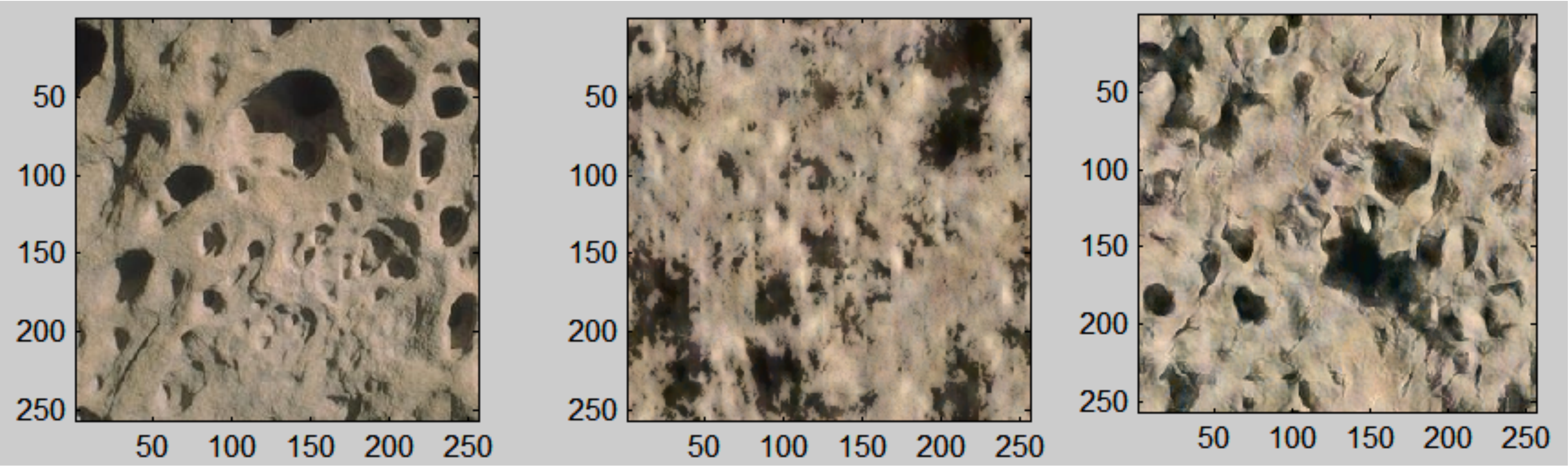
- Parametric representation, based on Gaussian scale mixture prior model for images.
- About 1000 numbers to describe a texture.
- Better results than Heeger Bergen.

Portilla and Simoncelli


Synthesized Texture Example: yellow-peppers256 - Microsoft Internet Explorer
Texture Synthesis Description
source: Simoncelli


Synthesized Texture Example: windowsP256 - Microsoft Internet Explorer
Texture Synthesis Description
source: Simoncelli

Portilla & Simoncelli



Heeger & Bergen

Portilla & Simoncelli