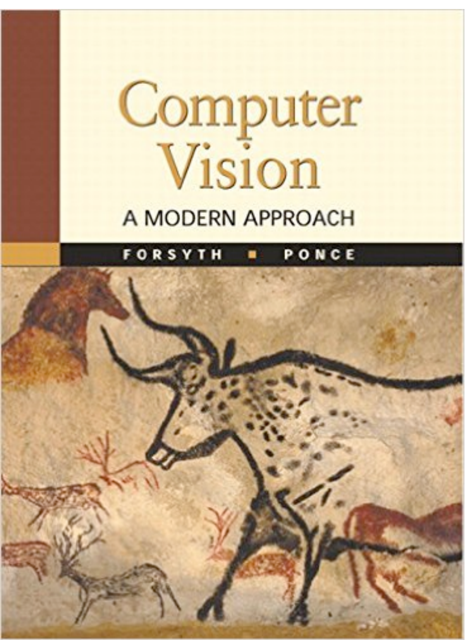
6.819 / 6.869: Advances in Computer Vision

Mid-level vision:

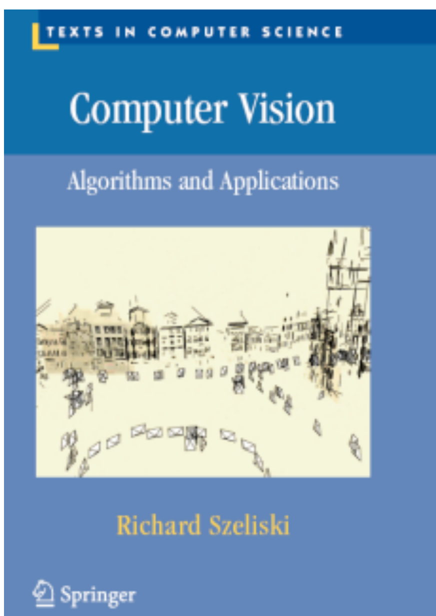
Texture and Shape Synthesis



Website:
http://6.869.csail.mit.edu/fa15/

Instructor: Aude Oliva

Lecture TR 9:30AM – 11:00AM
(Room 34-101)



Shape and Texture


GRANITE BAY FARMS
Organic Red Raspberry Vinaigrette
Organic Spicy Mango Salad Dressing


La Bernardine
La Bernardine
La Bernardine


Bandana's
SPICY BAR-B-Q SAUCE
SWEET & SMOKY BAR-B-Q SAUCE
HOT BAR-B-Q SAUCE
Original BAR-B-Q SAUCE


TASTY WINES
VELVETY
SPICY
BODY

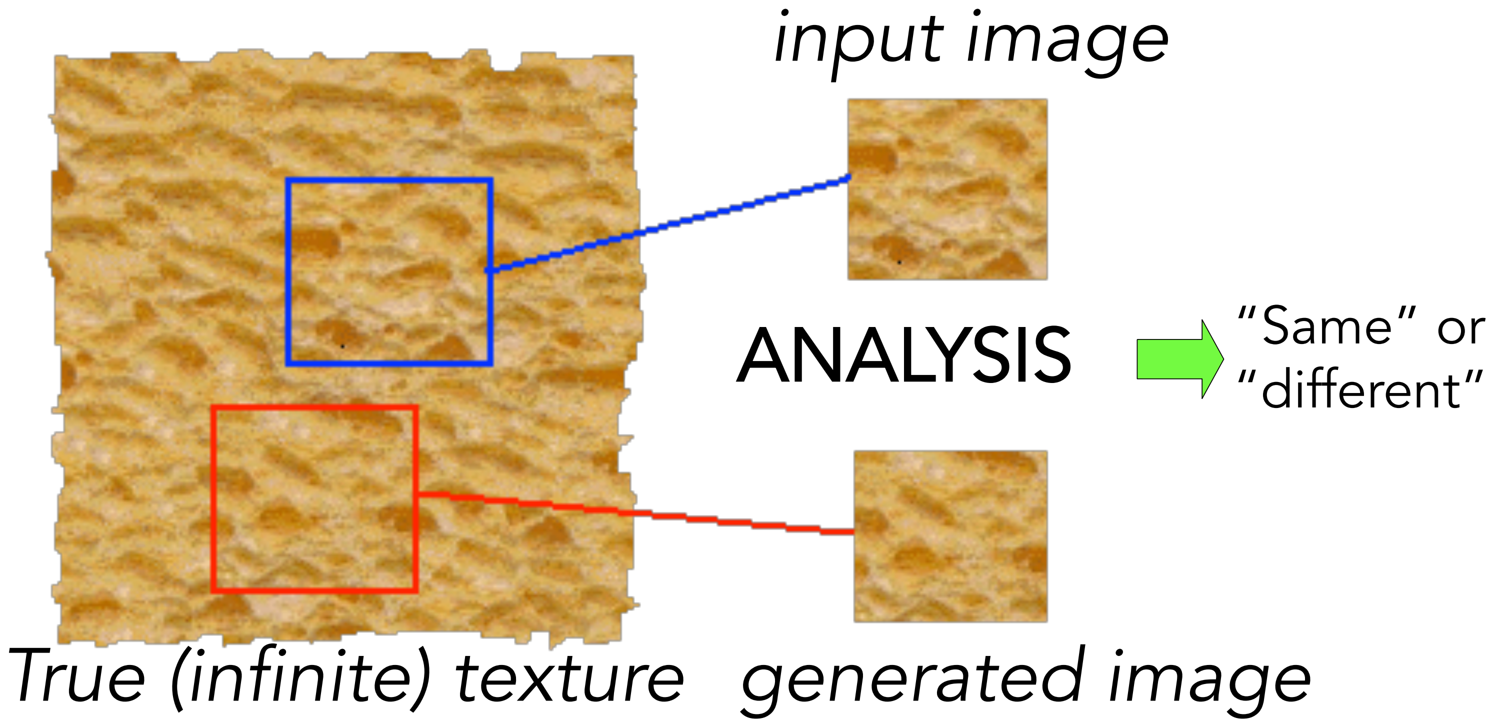
Two Categories of Textures

- 1) <u>determinist or regular textures</u> : determined by a set of <u>primitives</u> and a placement rule (e.g. a tile floor). Those are determined by repeated elements or groups of elements.
- 2) <u>stochastic textures</u>: do not have easily identifiable primitives (e.g. granite, sand).

When are two textures similar?

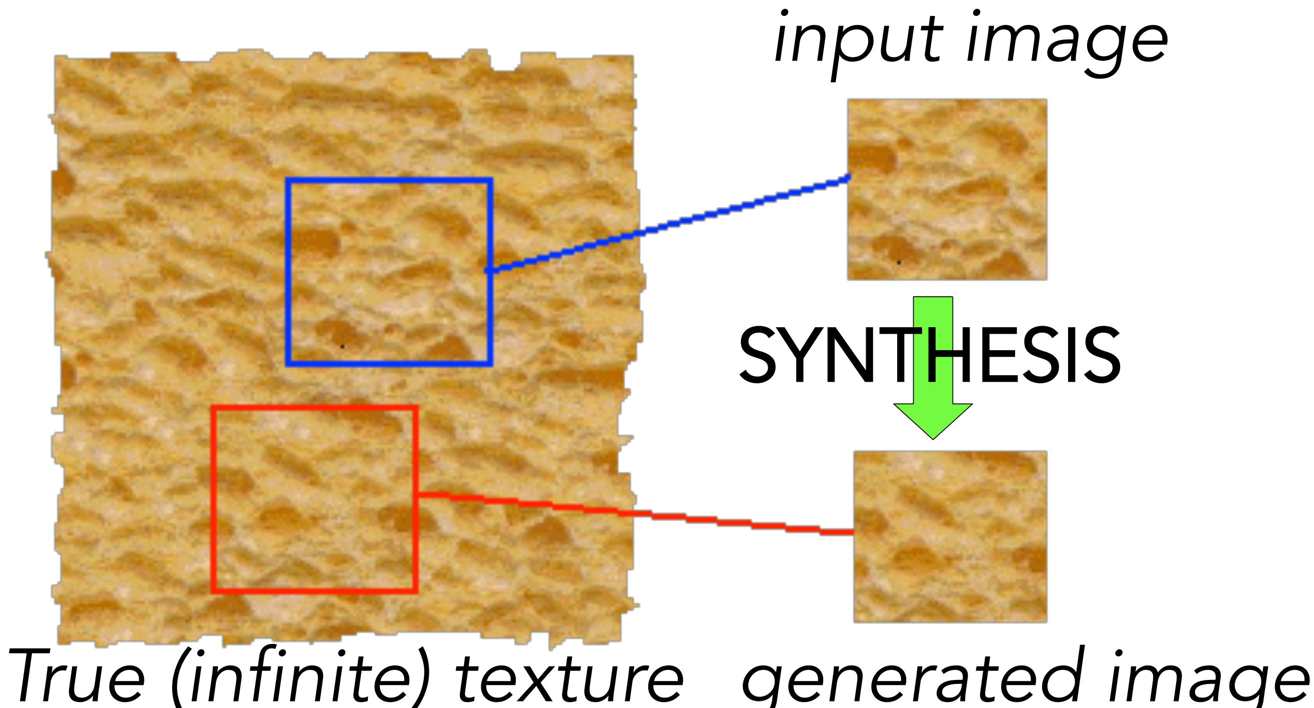
All these images are different instances of the same texture
We can differentiate between them, but they seem generated by the same process

Texture Analysis



Compare textures and decide if they're made of the same "stuff".

Texture Synthesis



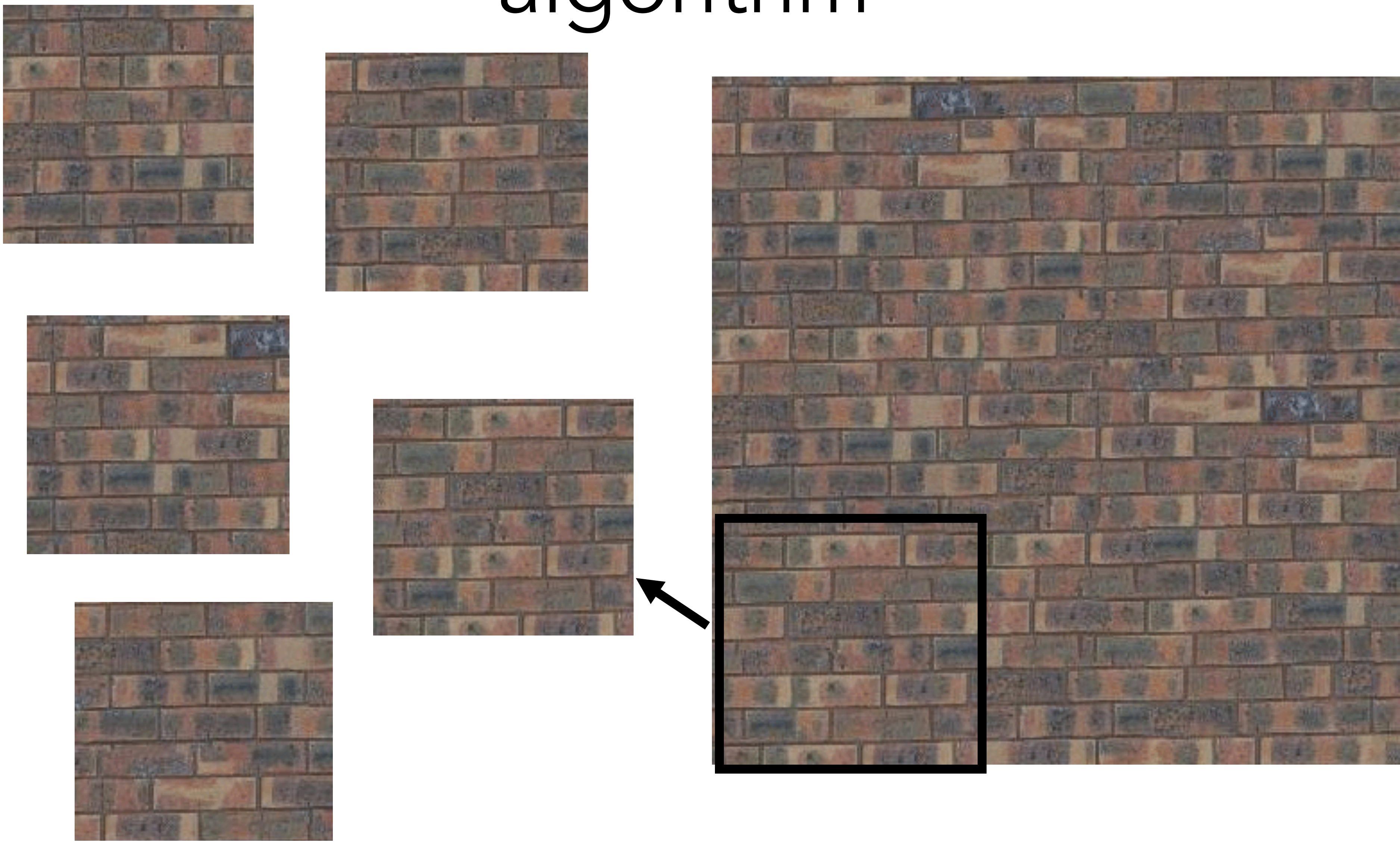
Given a finite sample of some texture, the goal is to synthesize other samples from that same texture

- The sample needs to be "large enough“

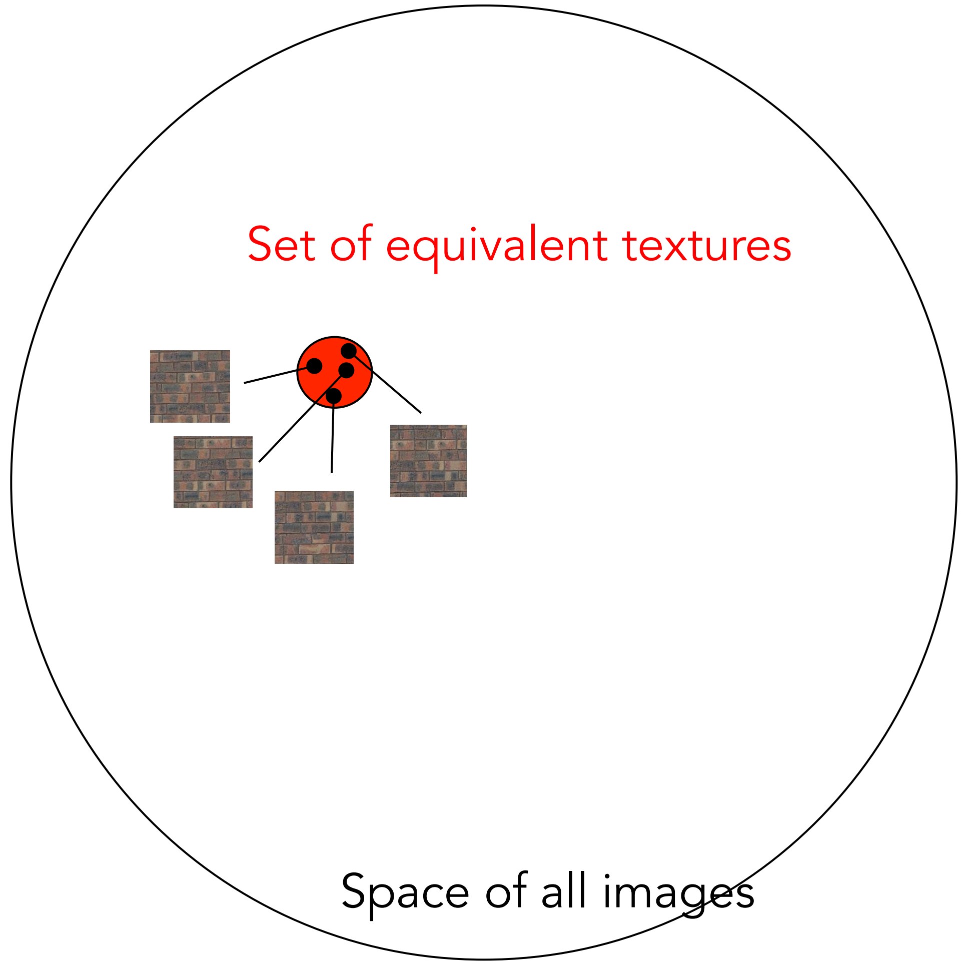
Two big families of models

I-Parametric models of filter outputs

The trivial texture synthesis algorithm

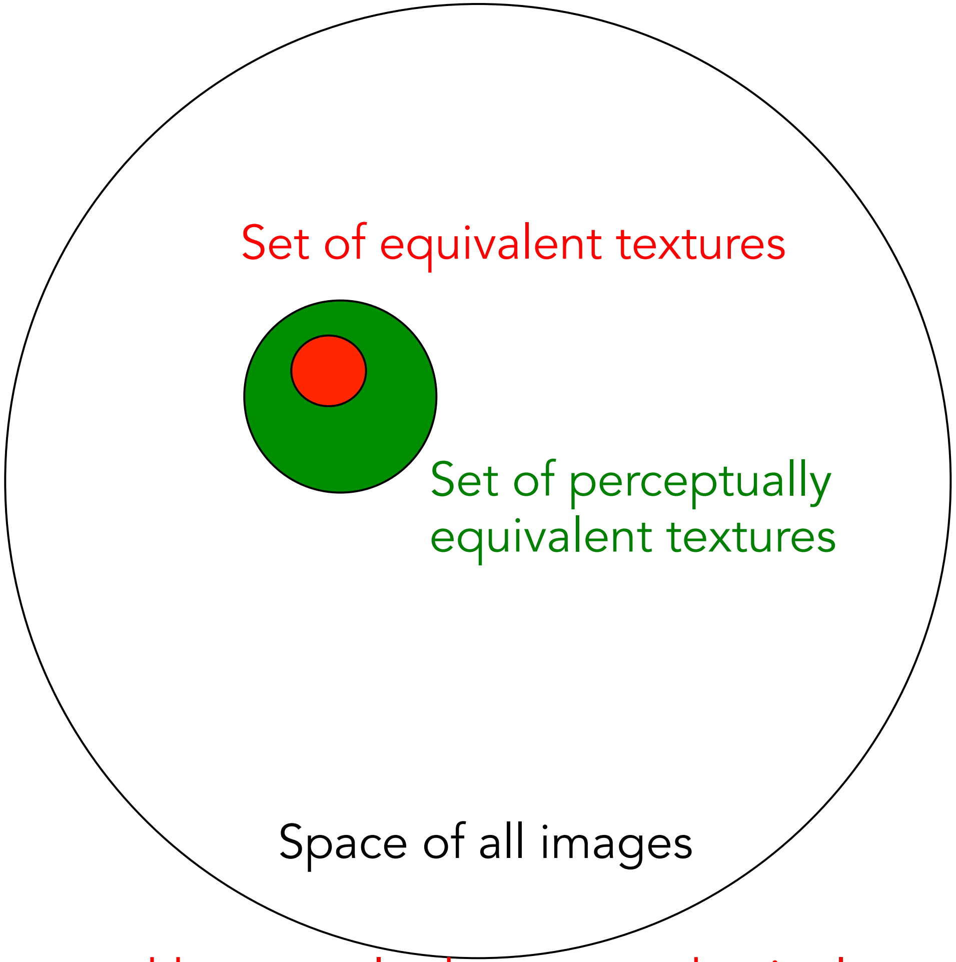


Texture synthesis and representation



Set of equivalent textures: generated by exactly the same physical process

Texture synthesis and representation



Set of equivalent textures: generated by exactly the same physical process

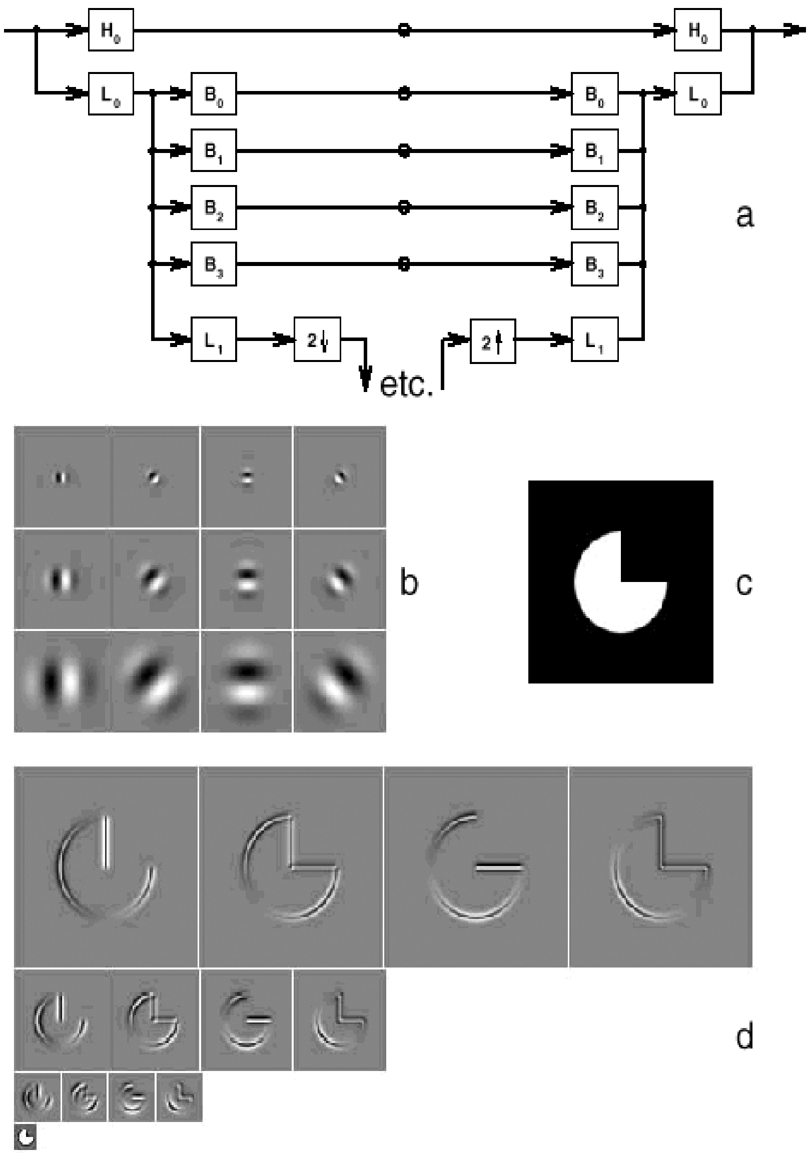
Set of perceptually equivalent textures: "well, they just look the same to me"

Pyramid-Based Texture Analysis/Synthesis

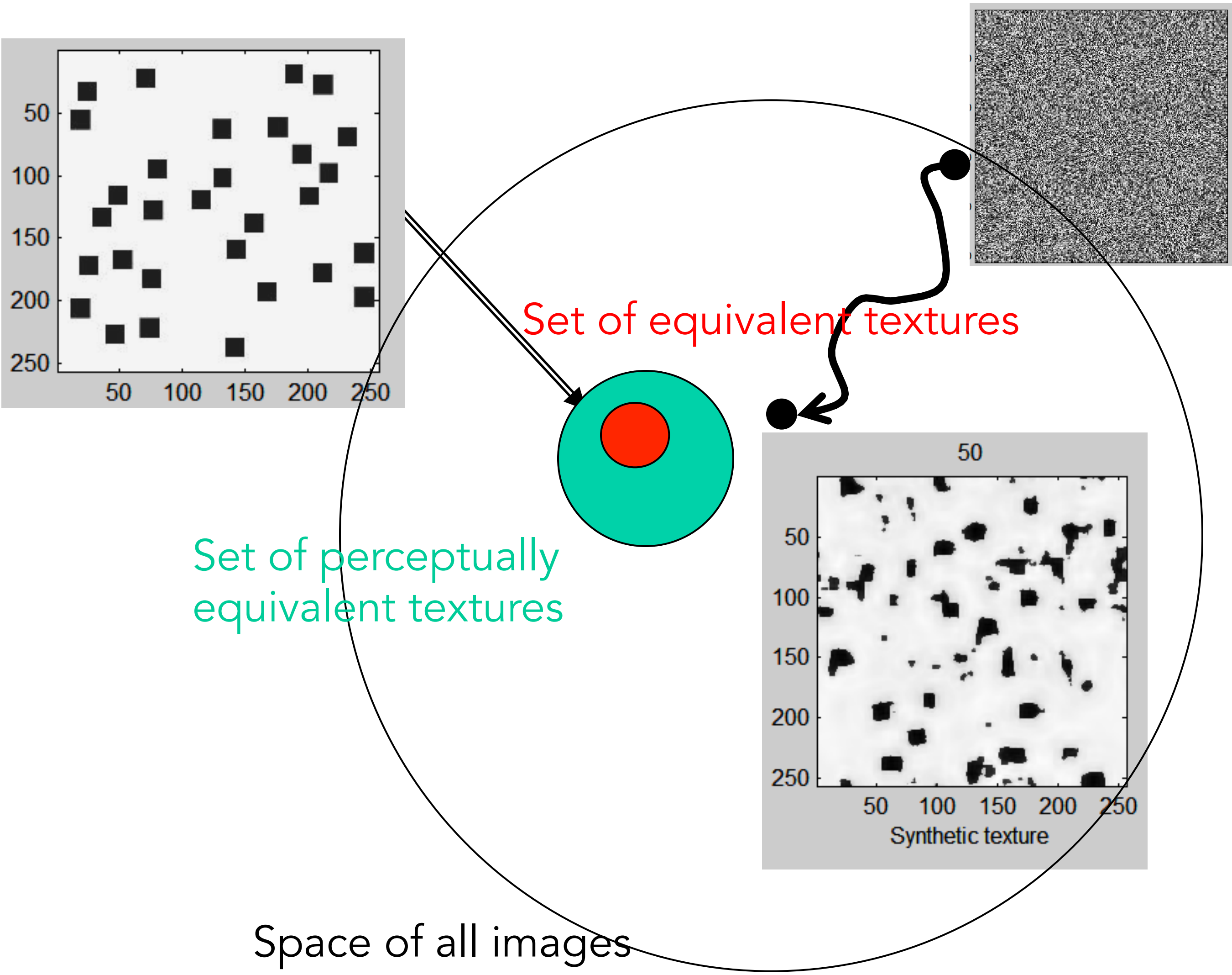
David J. Heeger[‡]
Stanford University

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

SIGGRAPH 1994



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



Overview of the algorithm

HeegerBergenTexture

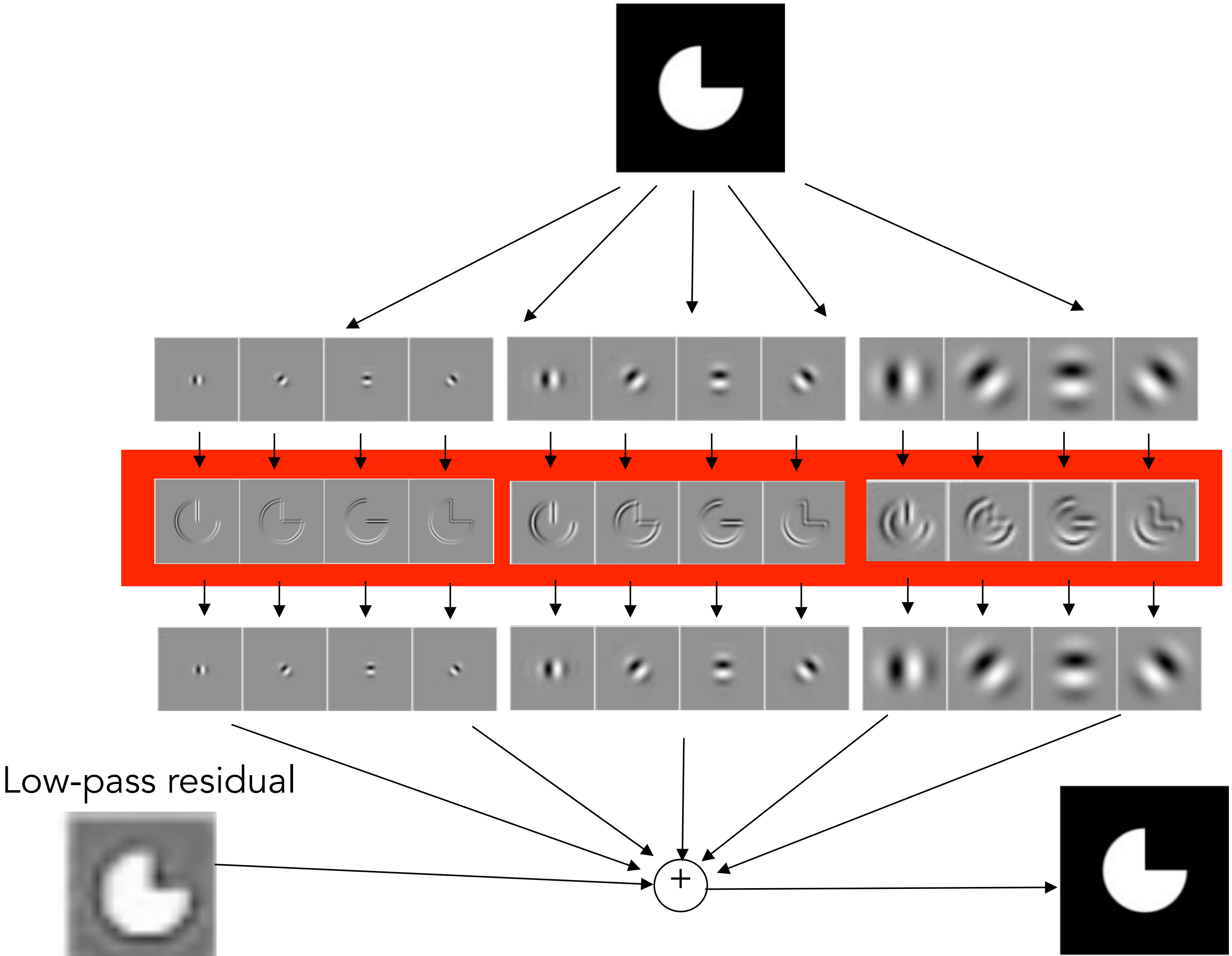
```
Match-texture(noise,texture)
  Match-Histogram (noise,texture)
  analysis-pyr = Make-Pyramid (texture)
  Loop for several iterations do
     synthesis-pyr = Make-Pyramid (noise)
     Loop for a-band in subbands of analysis-pyr
          for s-band in subbands of synthesis-pyr
          do
      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

Two main tools:

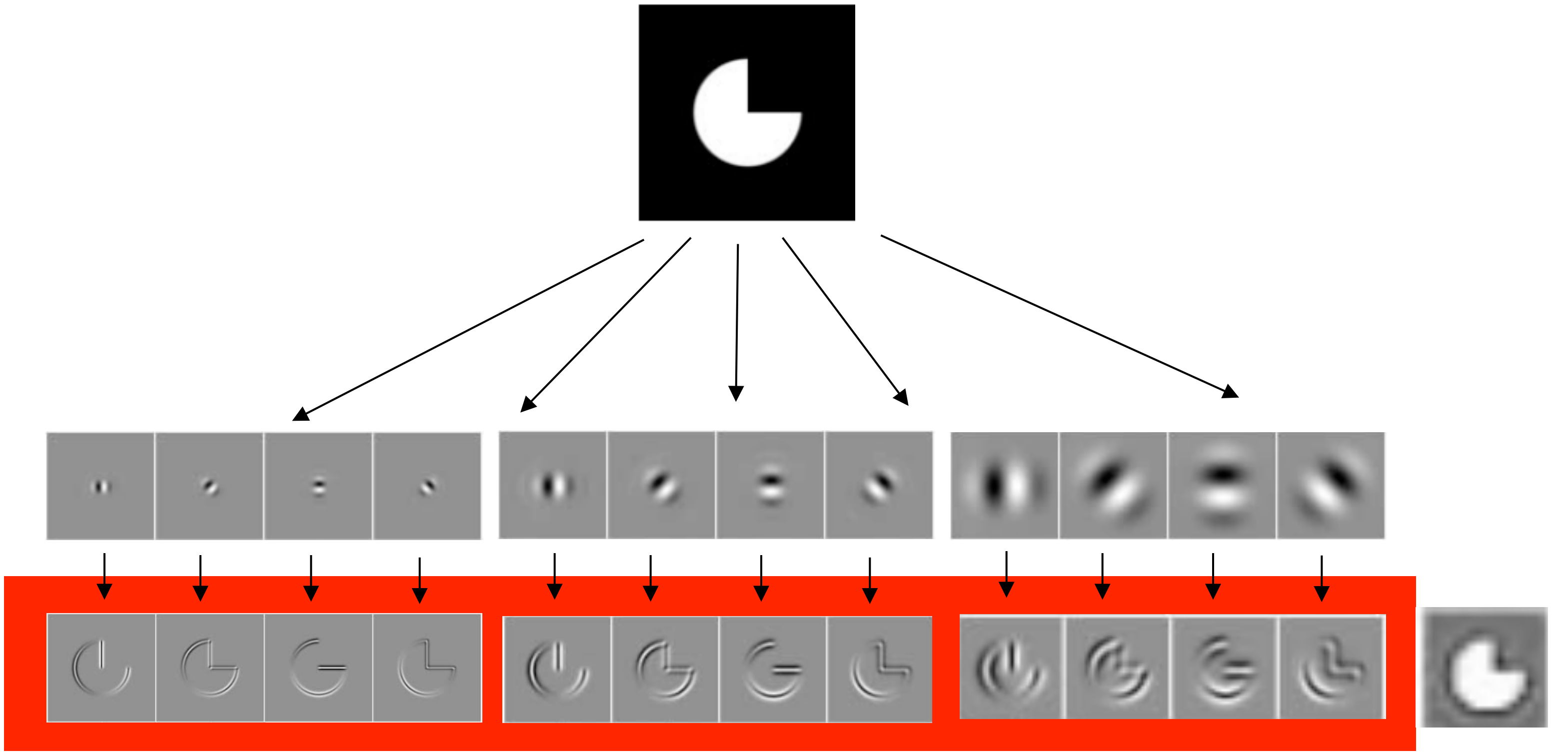
1- steerable pyramid

2- matching histograms

1-The steerable pyramid

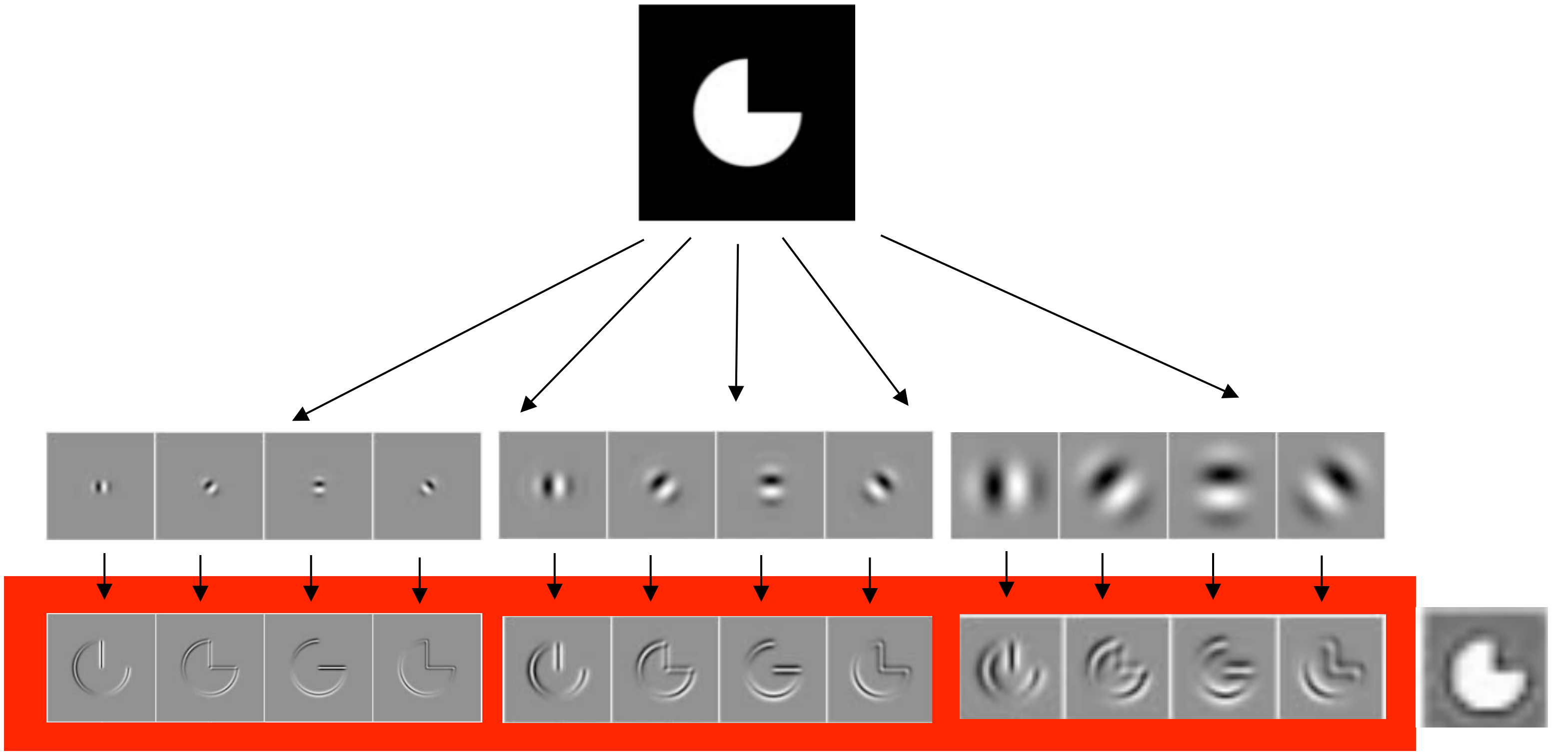


1-The steerable pyramid



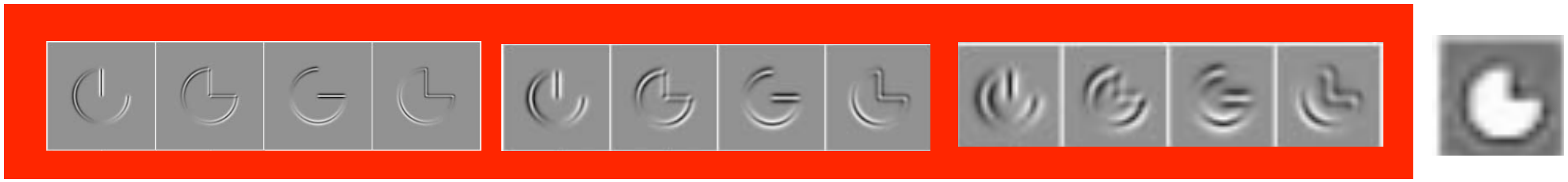
But why do I want to represent images like this?

1-The steerable pyramid



Argument used by H & B: Statistical measures in the subband representation seem to provide a "distance" between textures that correlates with human perception better than pixel-based representations.

1-The steerable pyramid



In general seems a good idea to have a representation that:

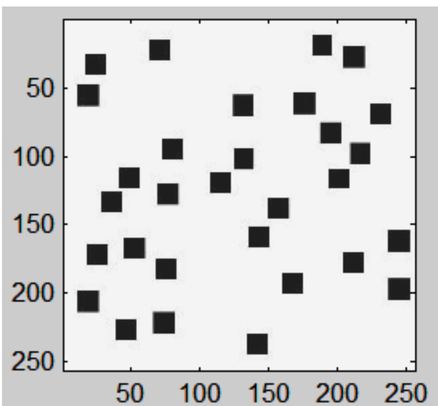
-Preserves all image information (we can go back to the image)

-Provides more independent channels of information than pixel values (we can mess with each band independently)

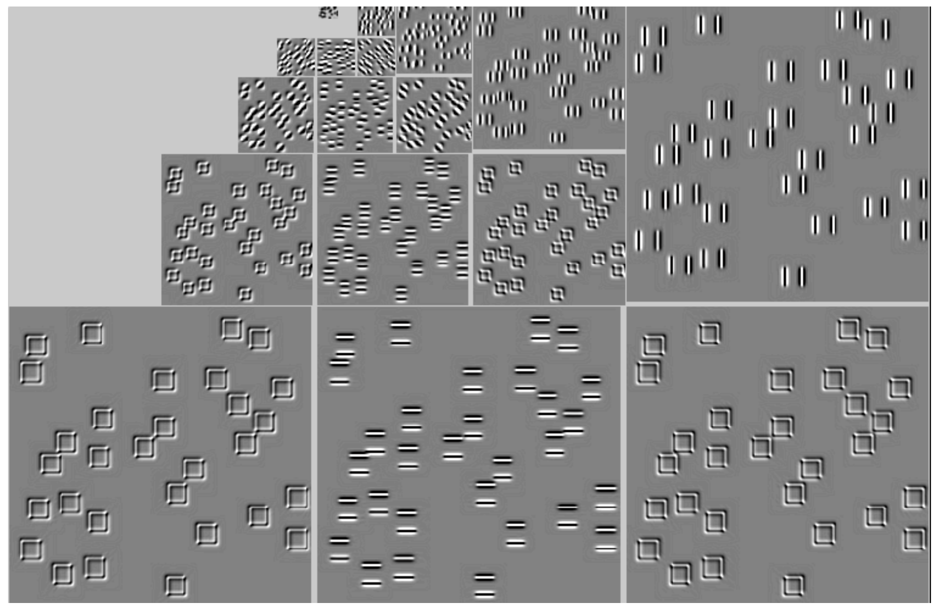
But all this is just indirectly related to the texture synthesis task. But let assume is good enough…

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)
```

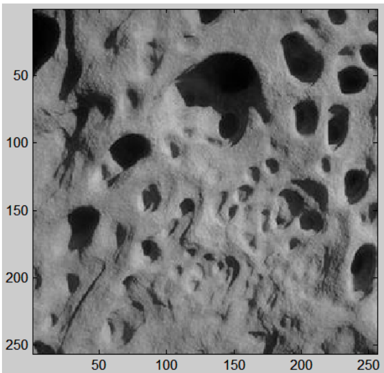
Two main tools:

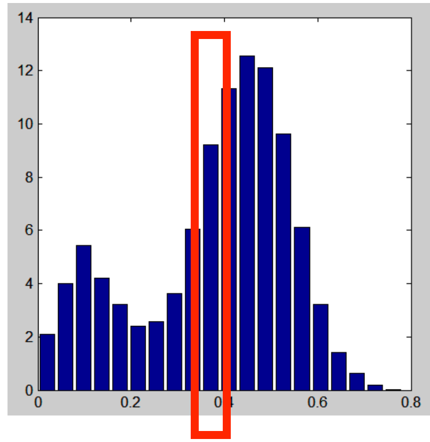
1- steerable pyramid

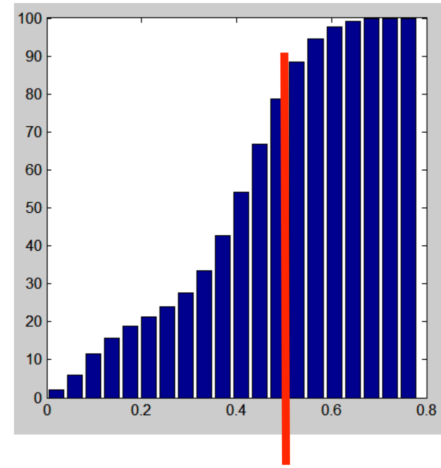
2- matching histograms

2-Matching histograms

Cumulative histogram

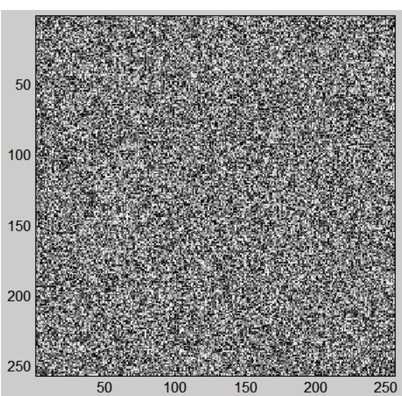


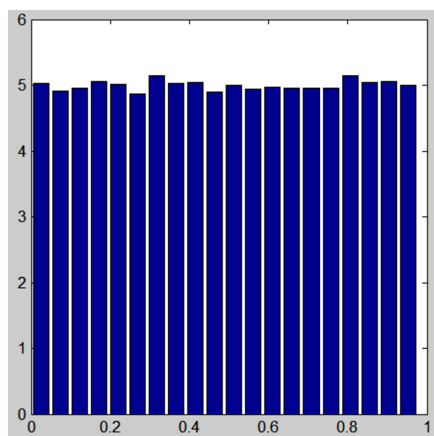


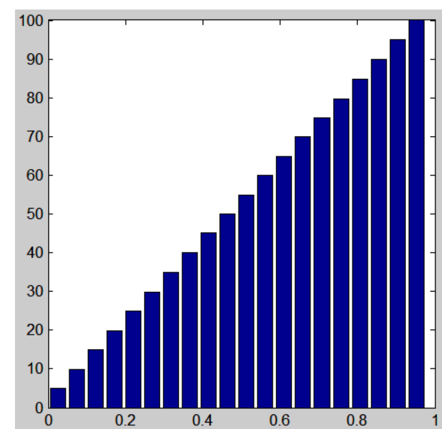


9% of pixels have an intensity value within the range[0.37, 0.41]

75% of pixels have an intensity value smaller than 0.5



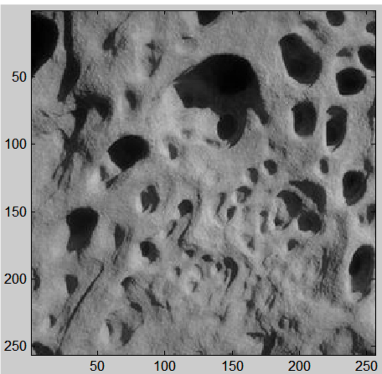


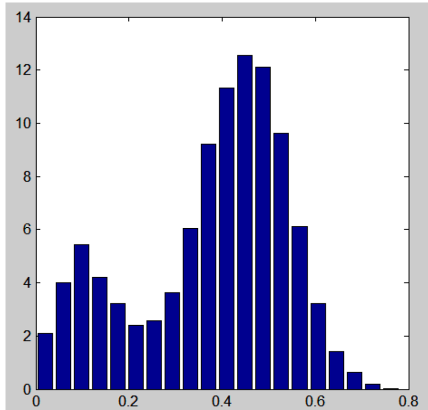


5% of pixels have an intensity value within the range[0.37, 0.41]

2-Matching histograms

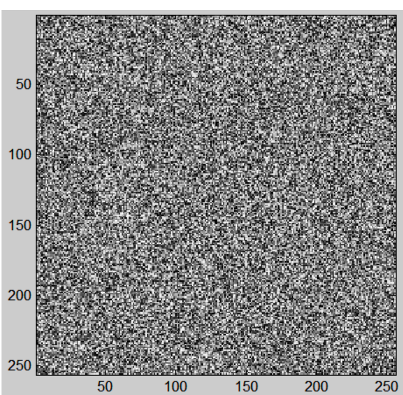
Z(x,y)

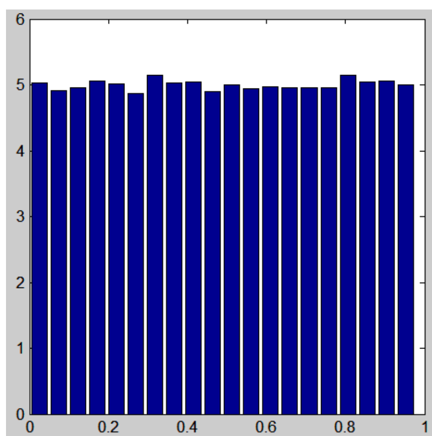




?

Y(x,y)





We look for a transformation of the image Y

Y' = f (Y)

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

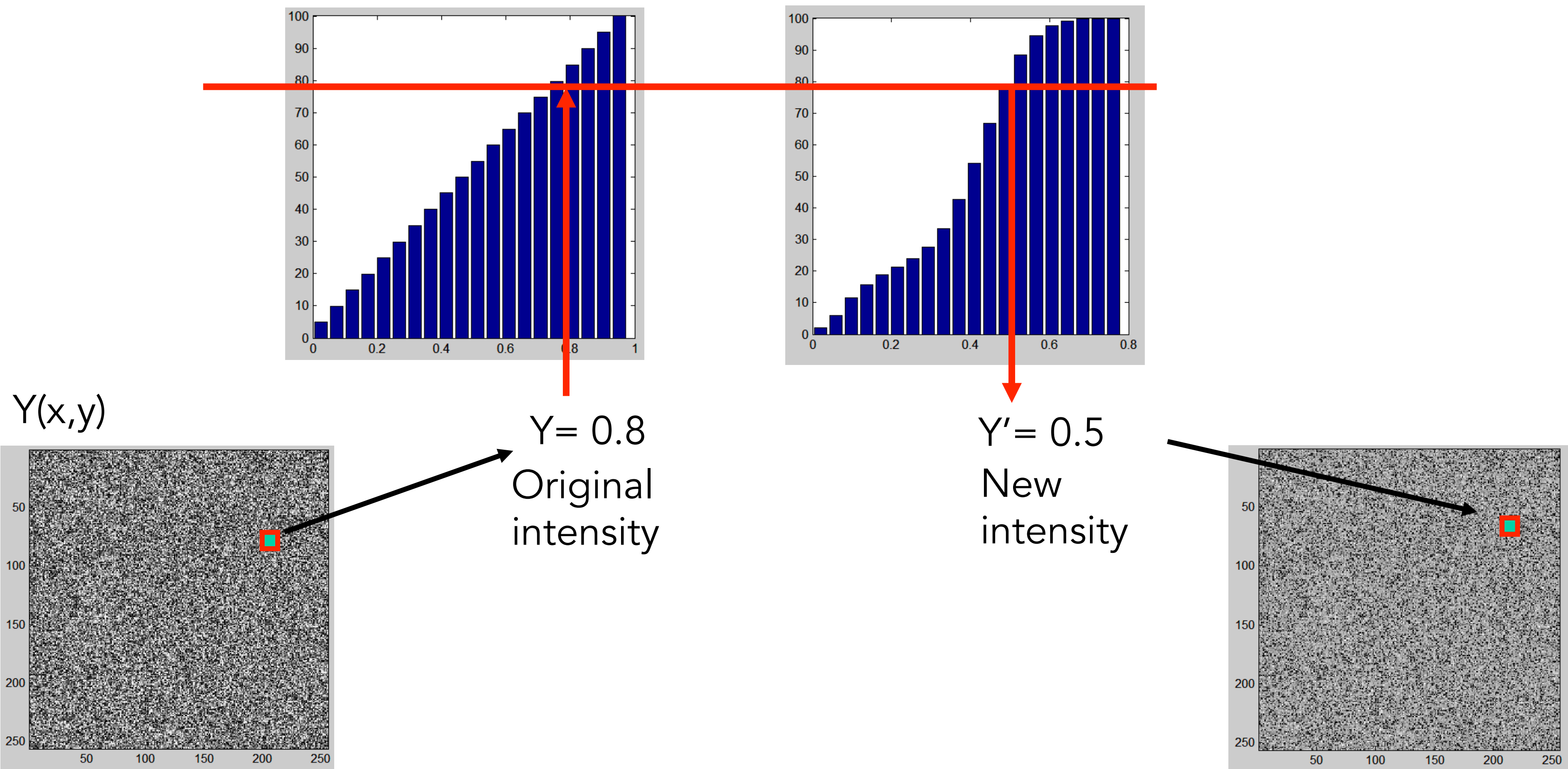
Problem: there are infinitely many functions that can do this transformation.

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

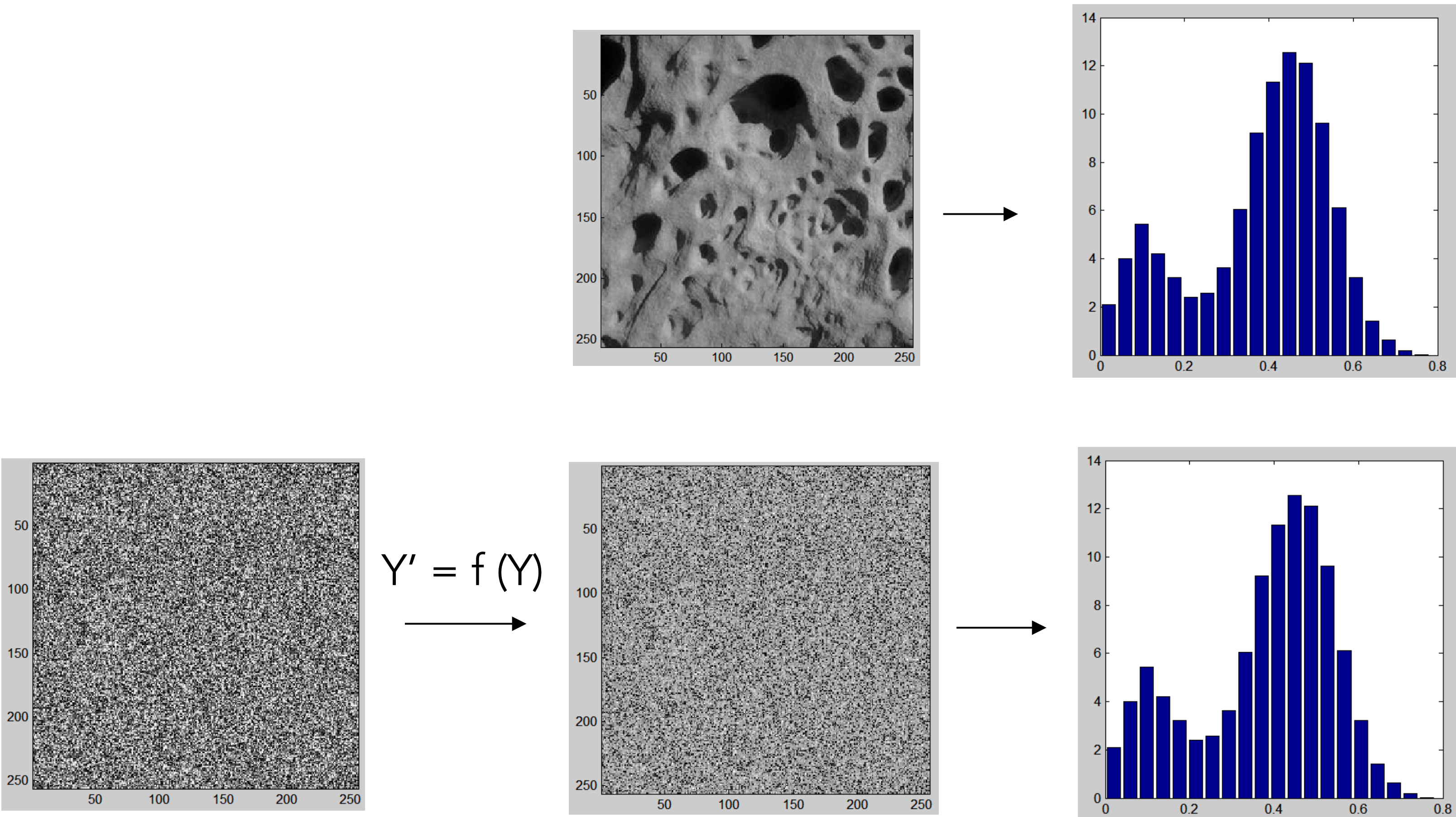
2-Matching histograms

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

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

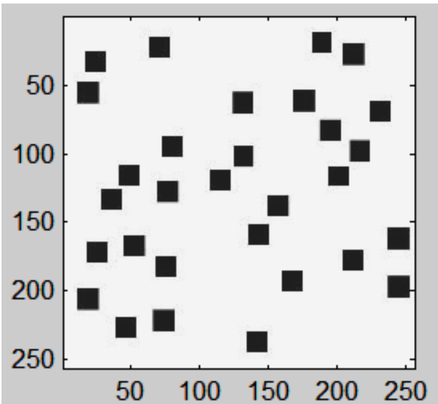


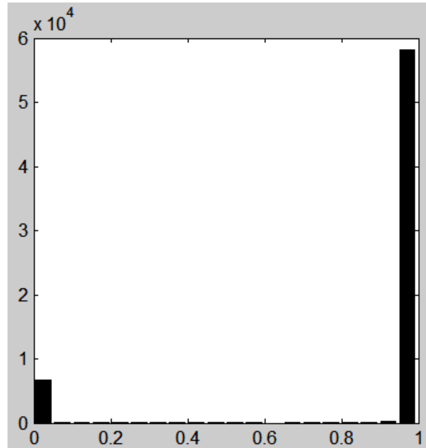
2-Matching histograms

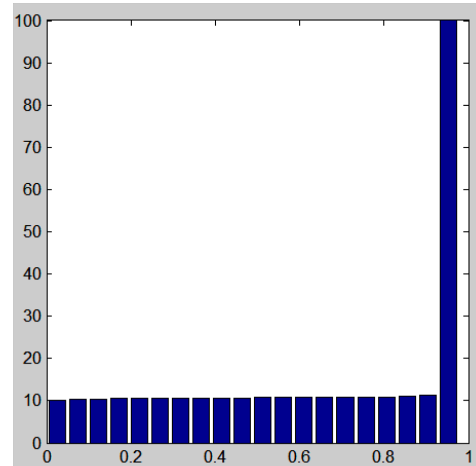


Another example: Matching histograms

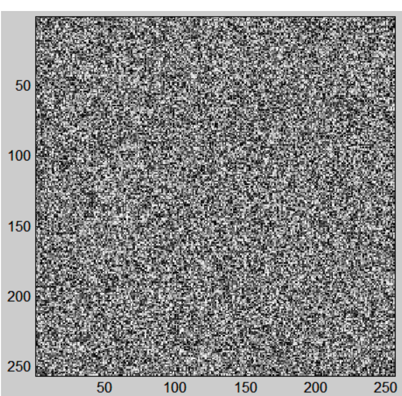
Cumulative histogram

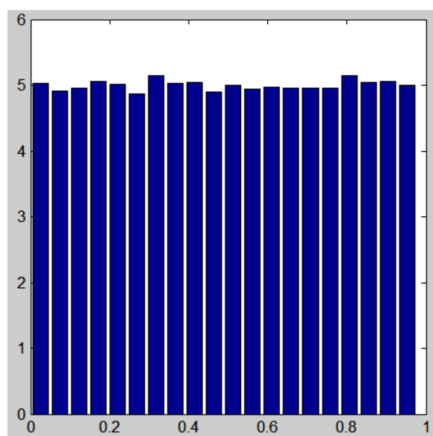


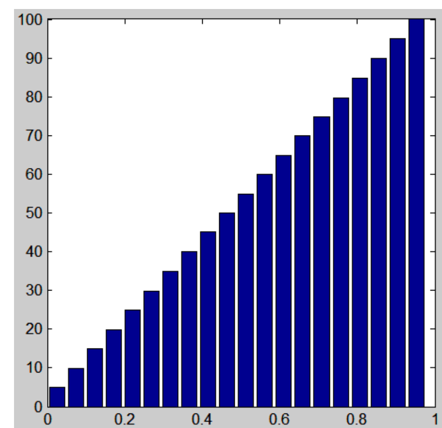




10% of pixels are black and 90% are white



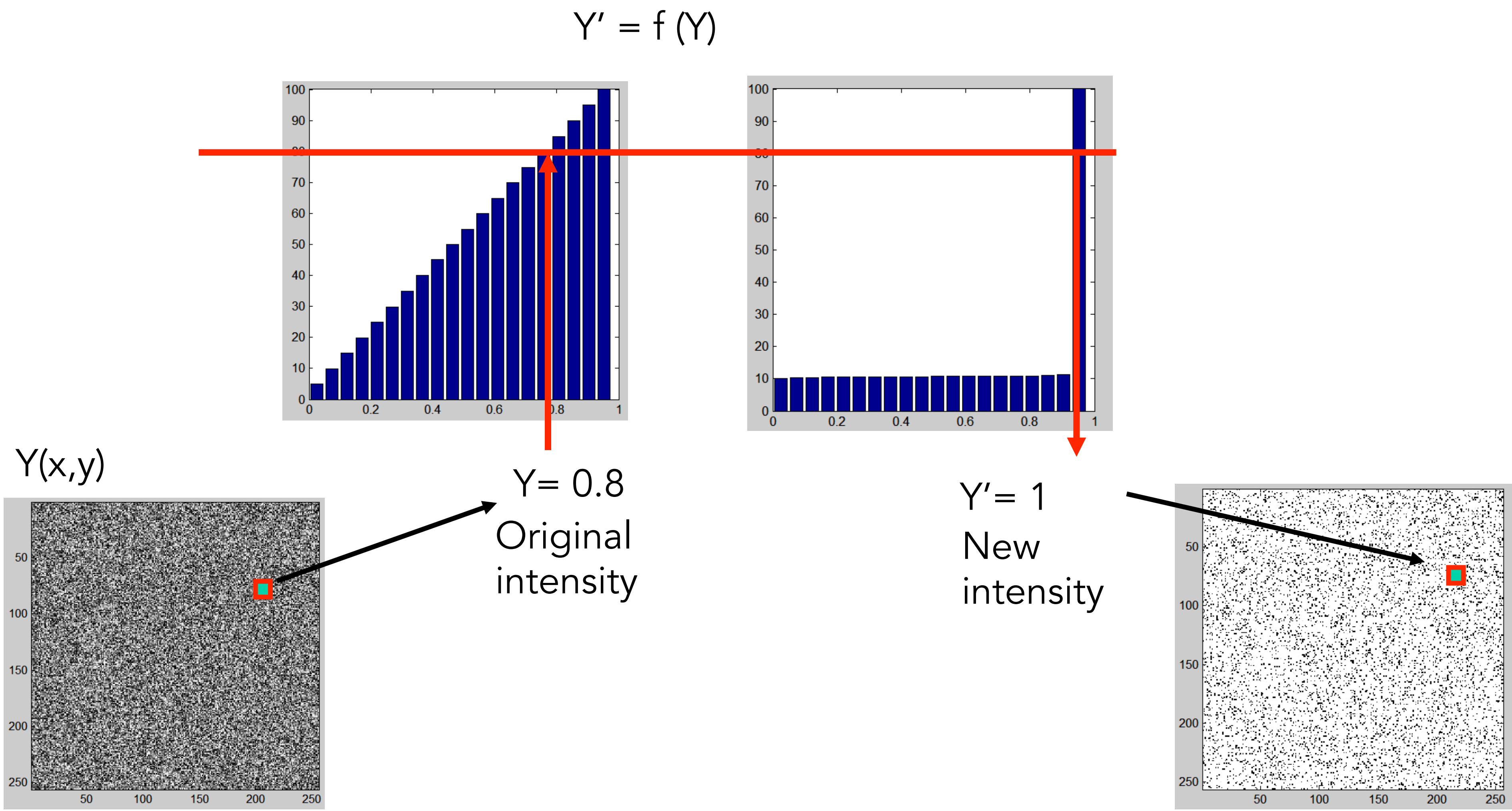




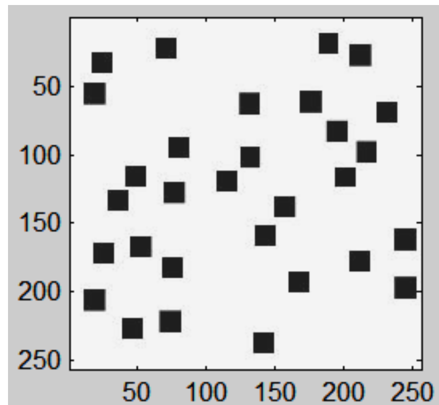
5% of pixels have an intensity value within the range[0.37, 0.41]

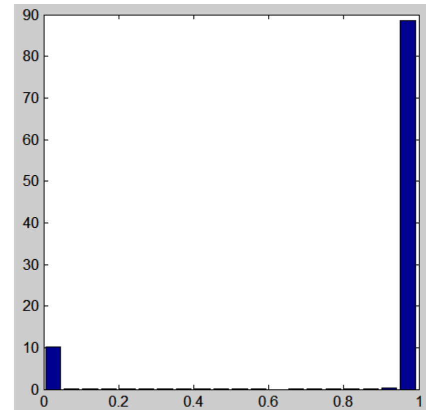
Another example: Matching histograms

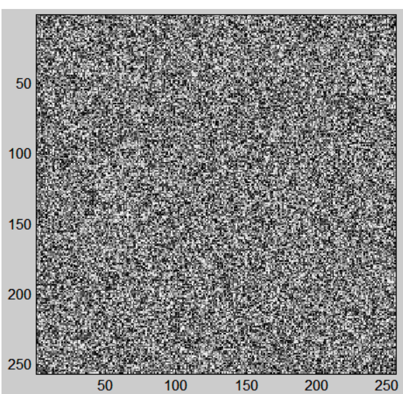
The function f is just a look up table: it says, change all the pixels of value Y into a value f(Y).



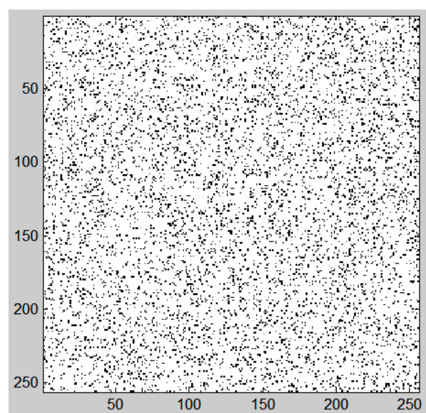
Another example: Matching histograms

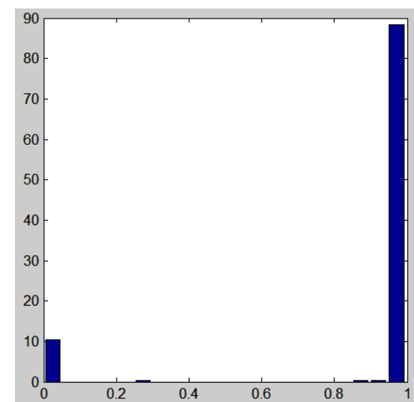






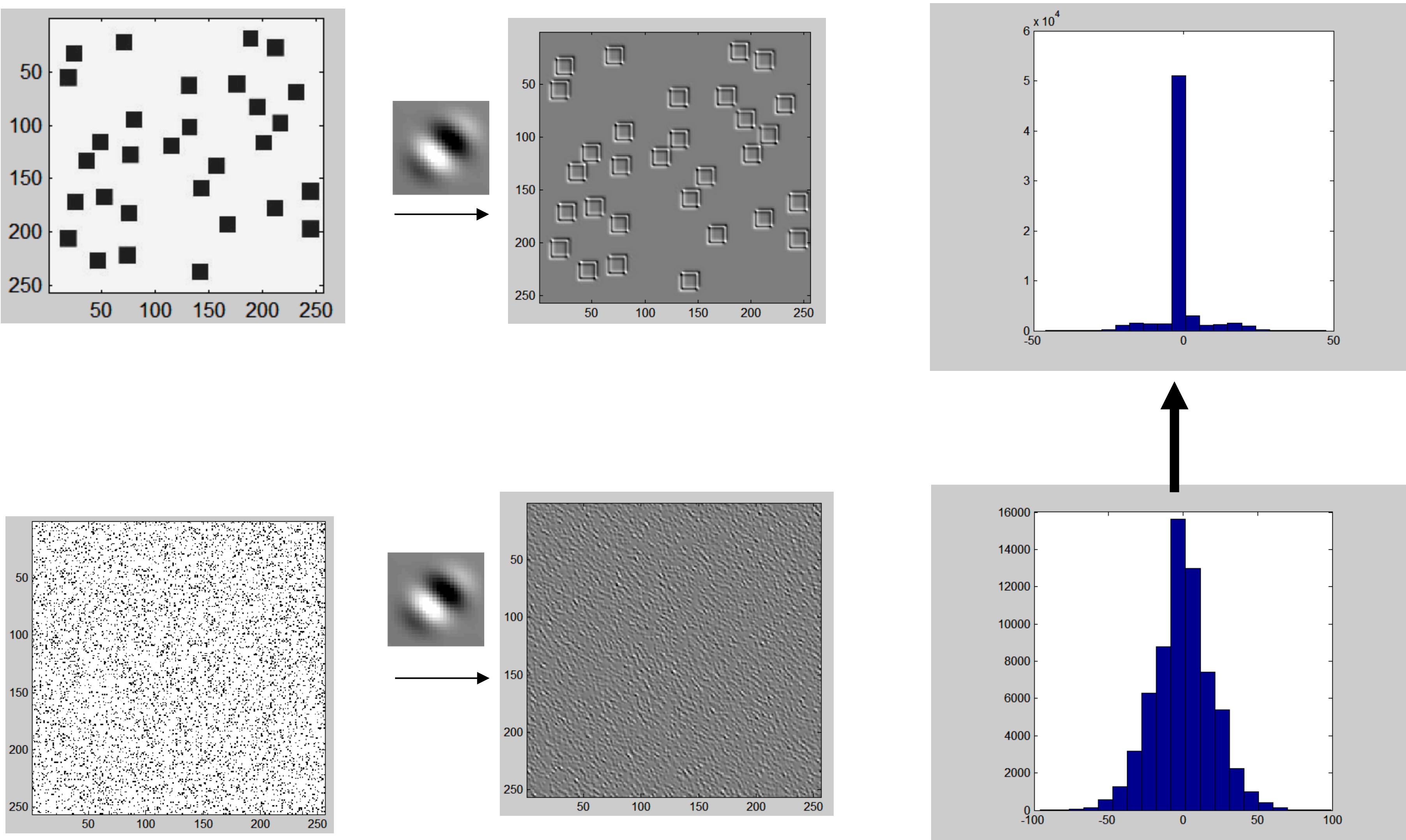
$Y' = f\,(Y)$



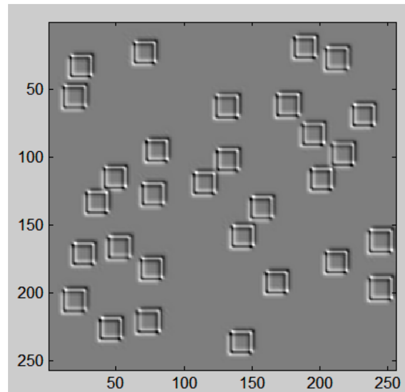


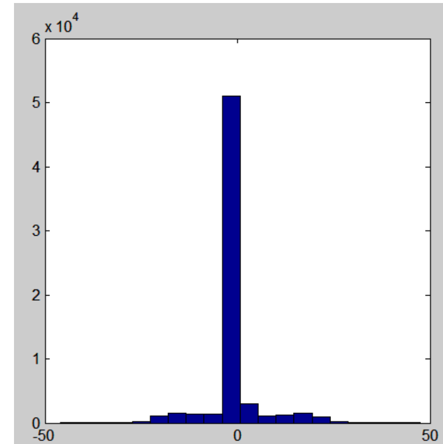
In this example, f is a step function.

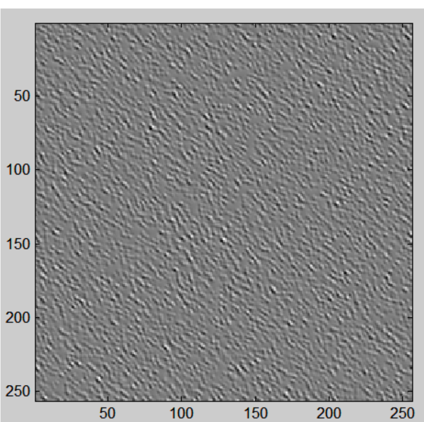
Matching histograms of a subband



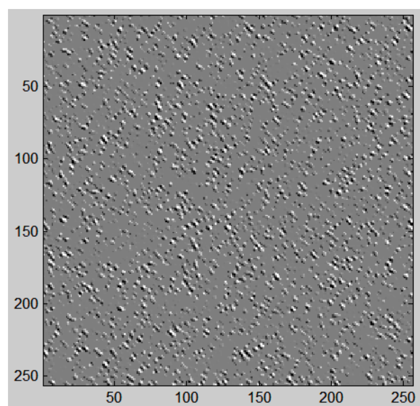
Matching histograms of a subband

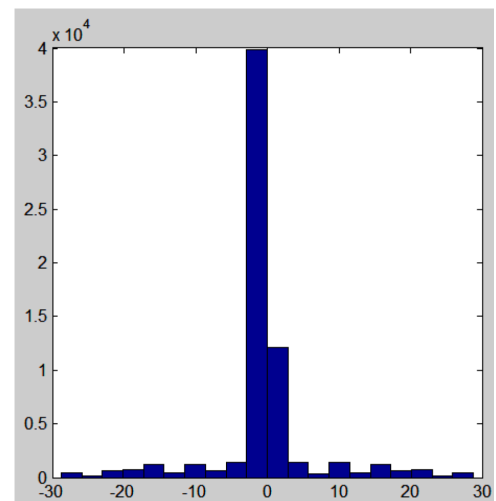






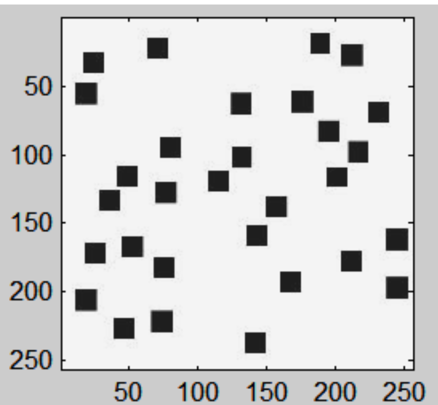
Y’ = f (Y)



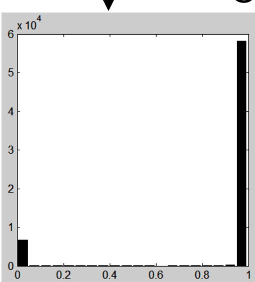


Texture analysis

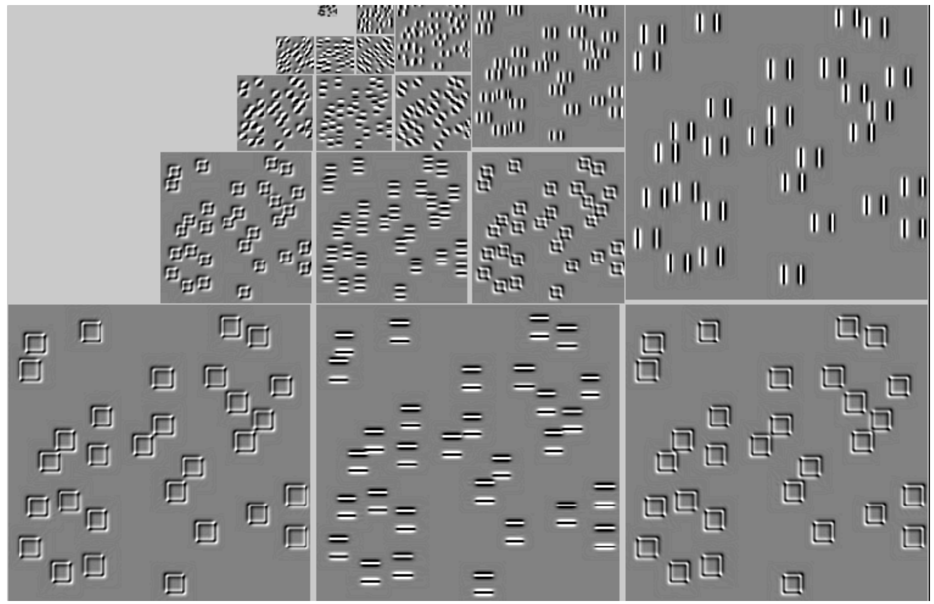
Input texture



(histogram)

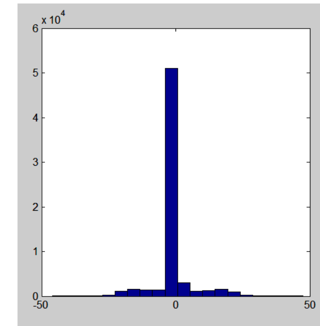


Wavelet decomposition (steerable pyr)



(Steerable pyr; Freeman & Adelson, 91)

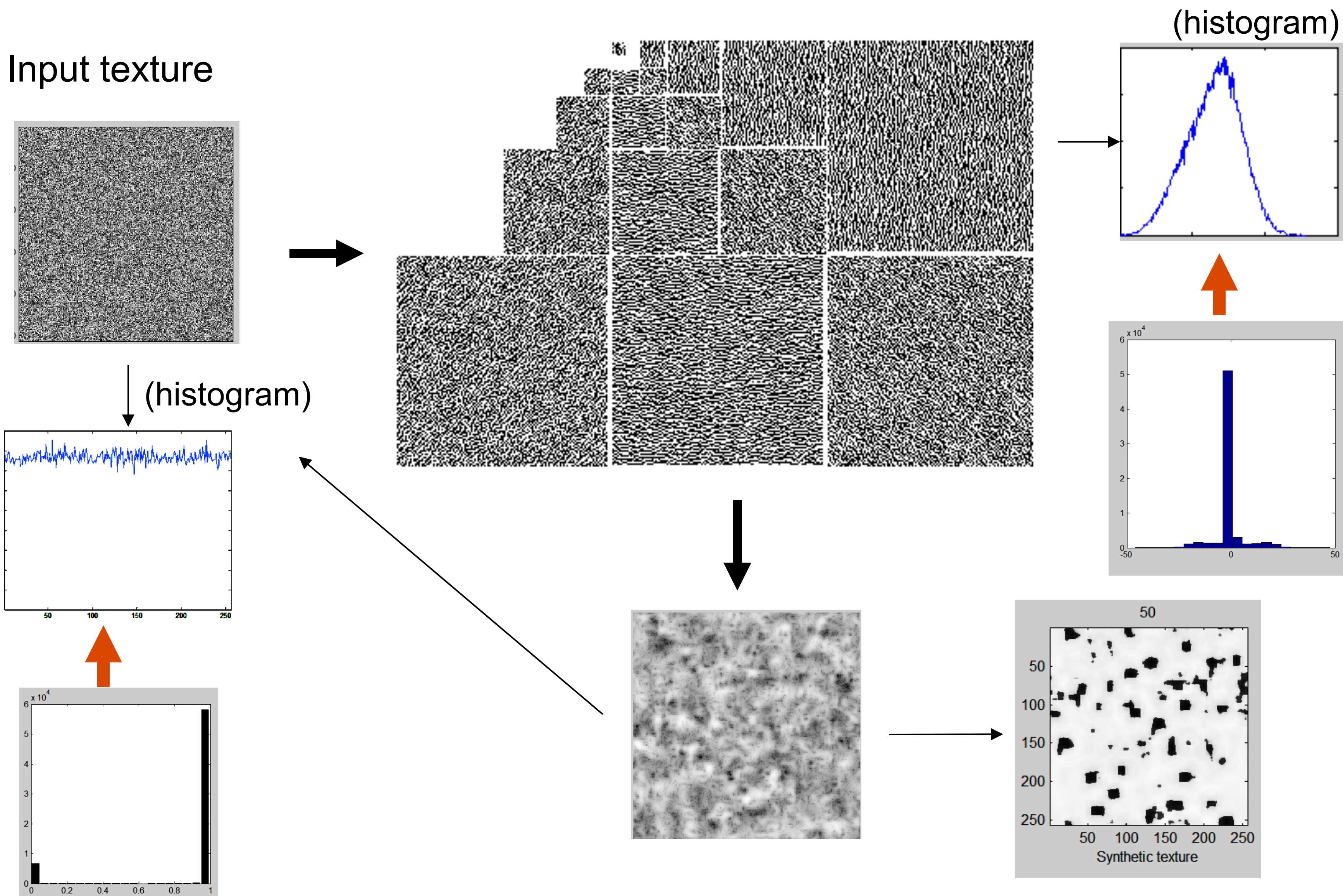
(histogram)

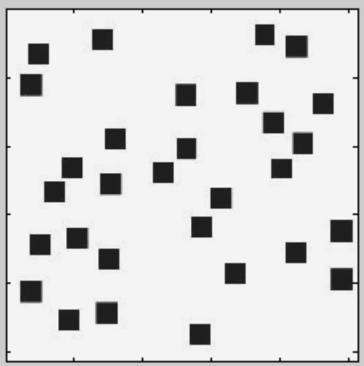


The texture is represented as a collection of marginal histograms.

Texture synthesis

Heeger and Bergen, 1995



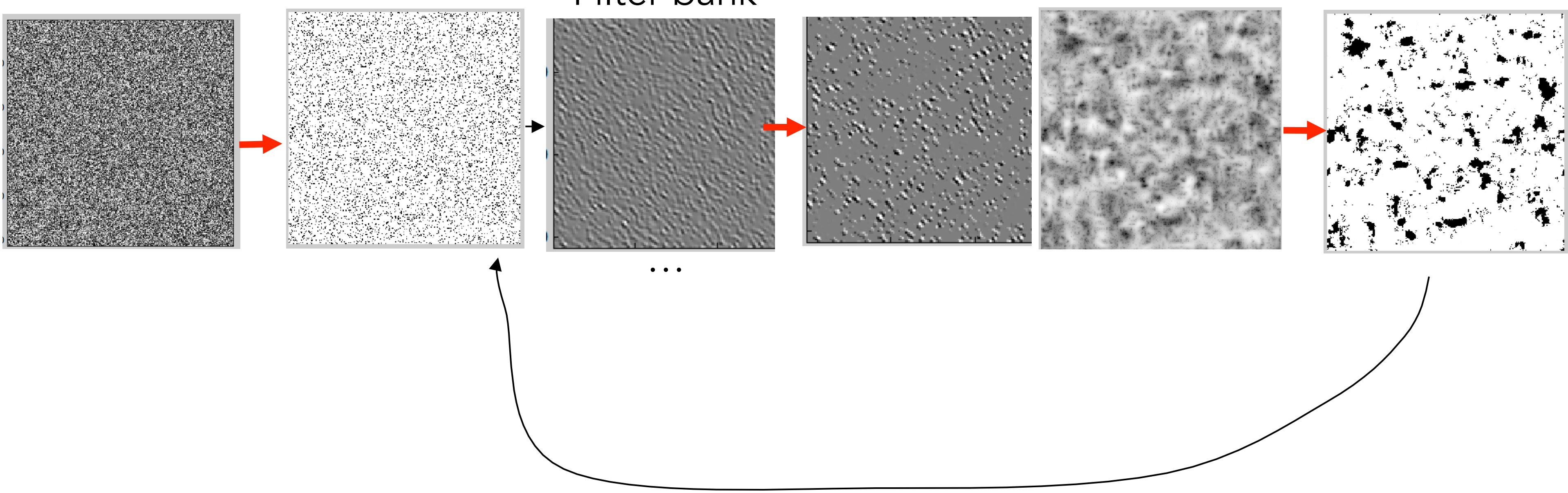


Why does it work? (sort of)

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

Iteration 0

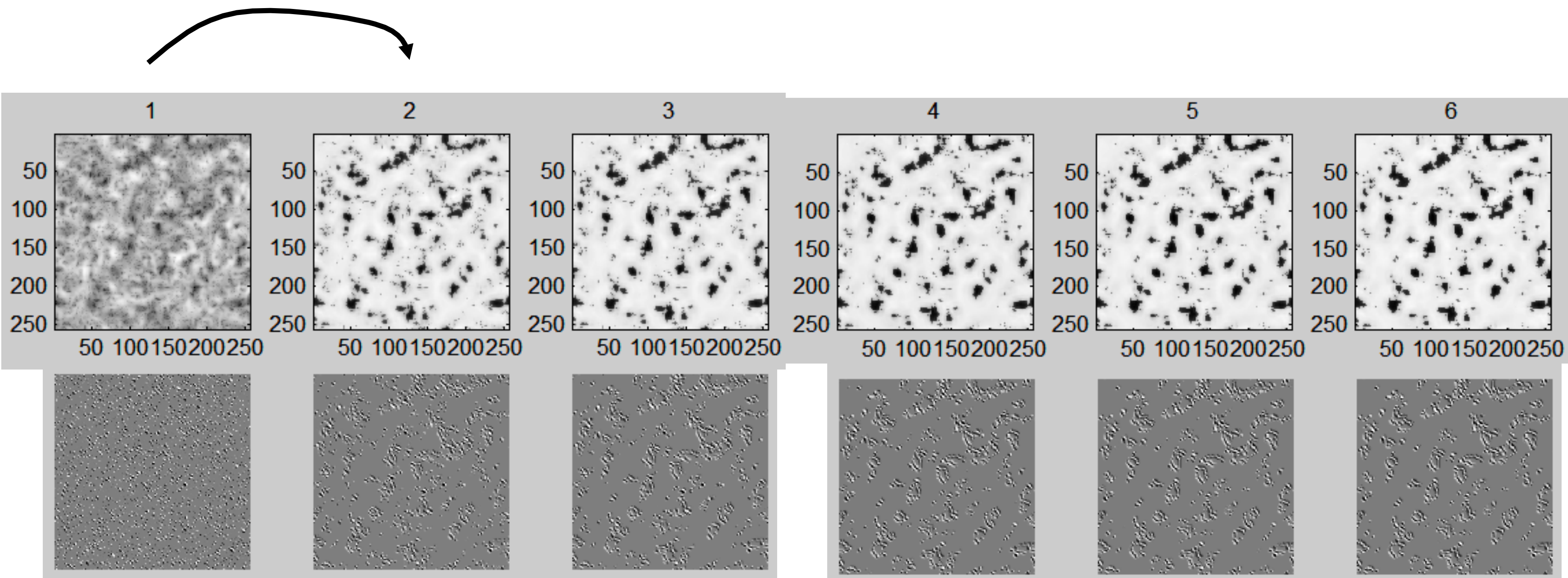
Filter bank

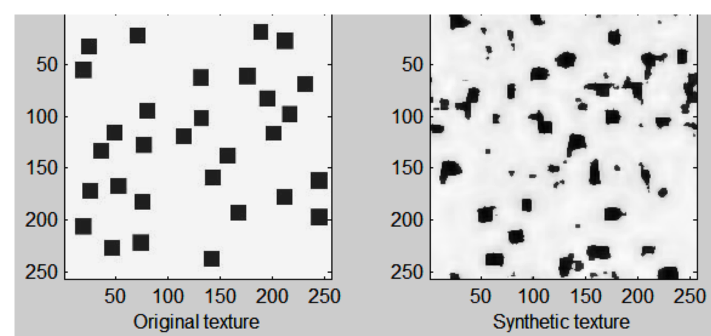


...

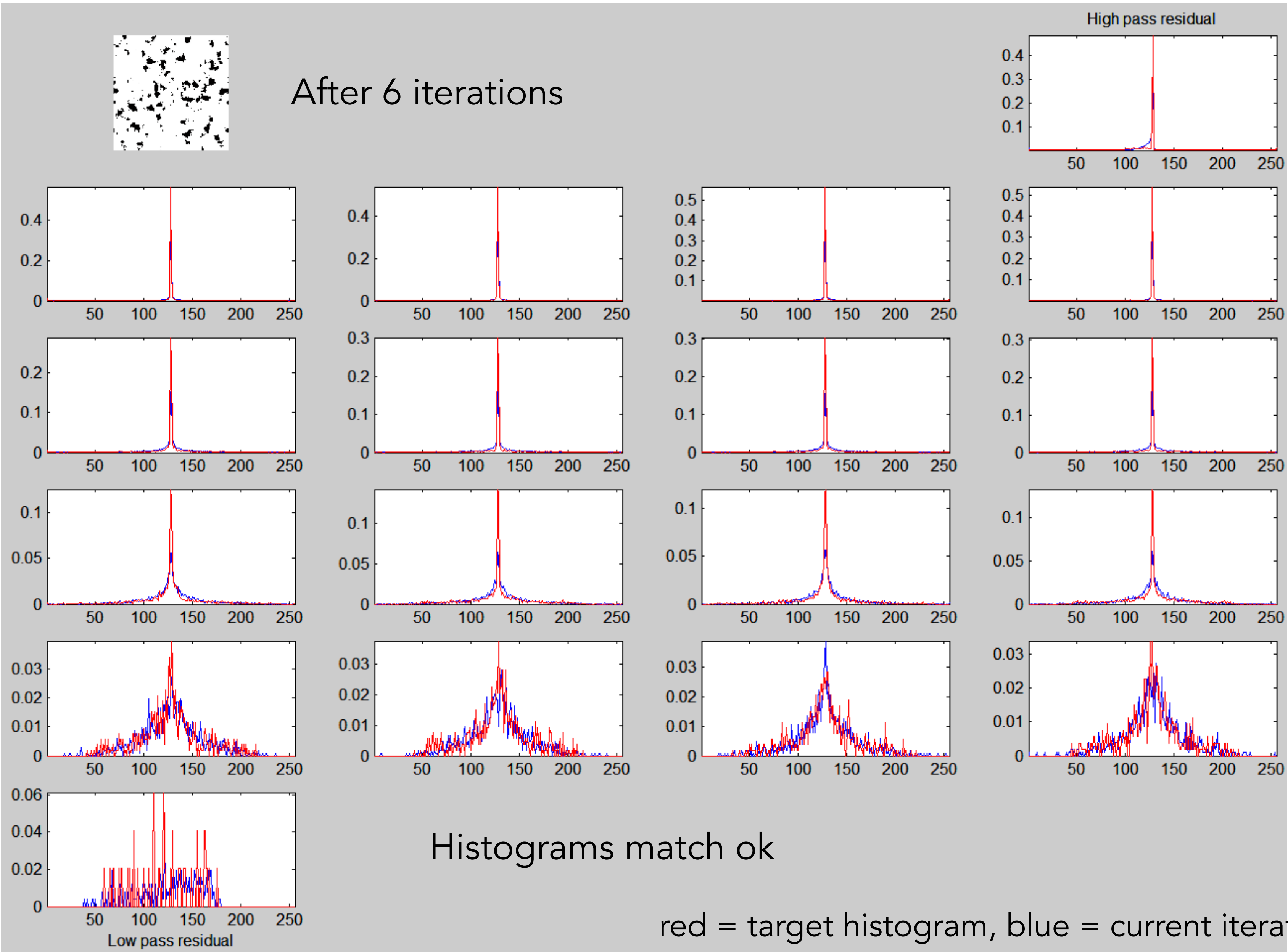
Why does it work? (sort of)

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





Why does it work? (sort of)

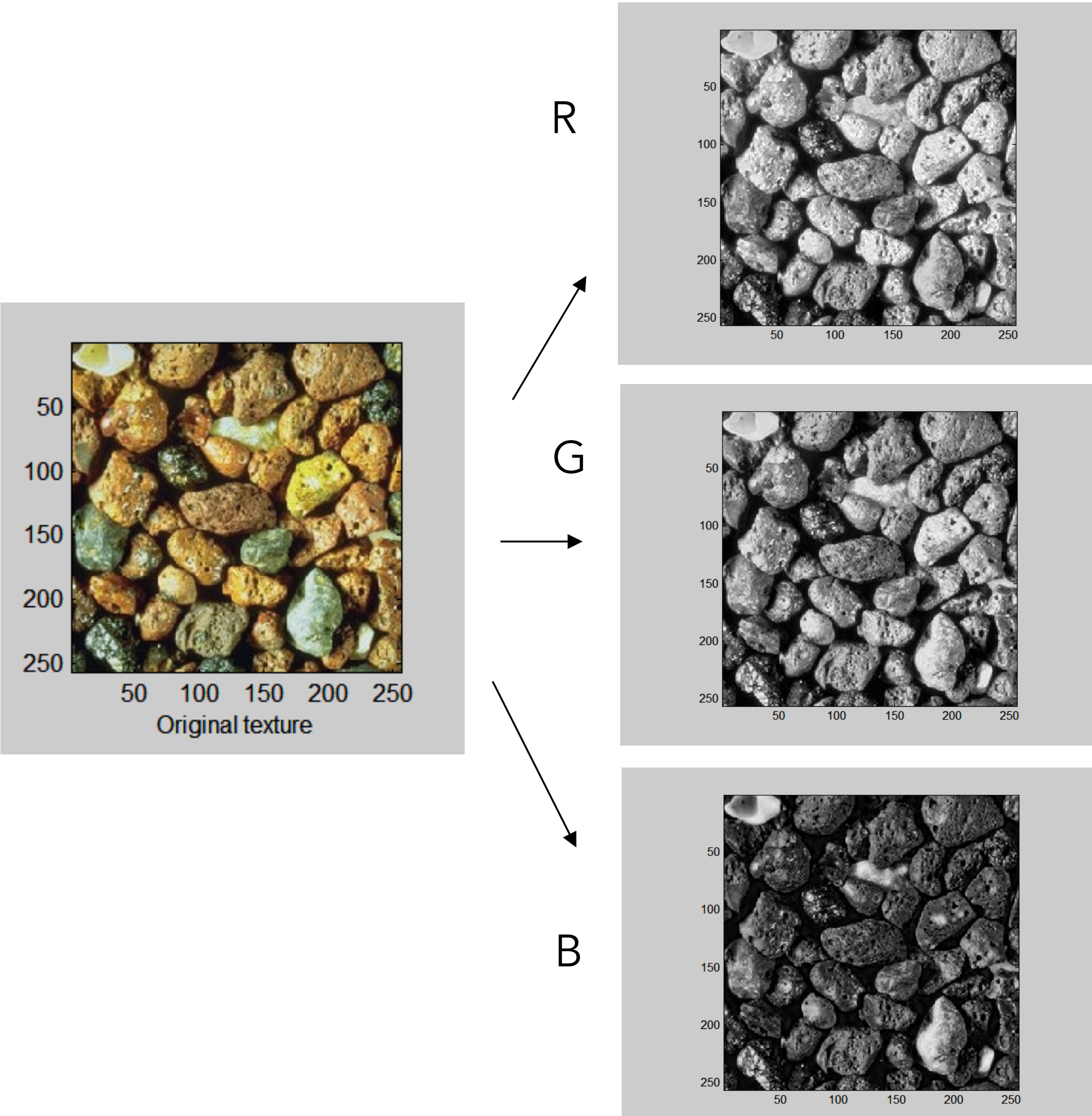


After 6 iterations

Histograms match ok

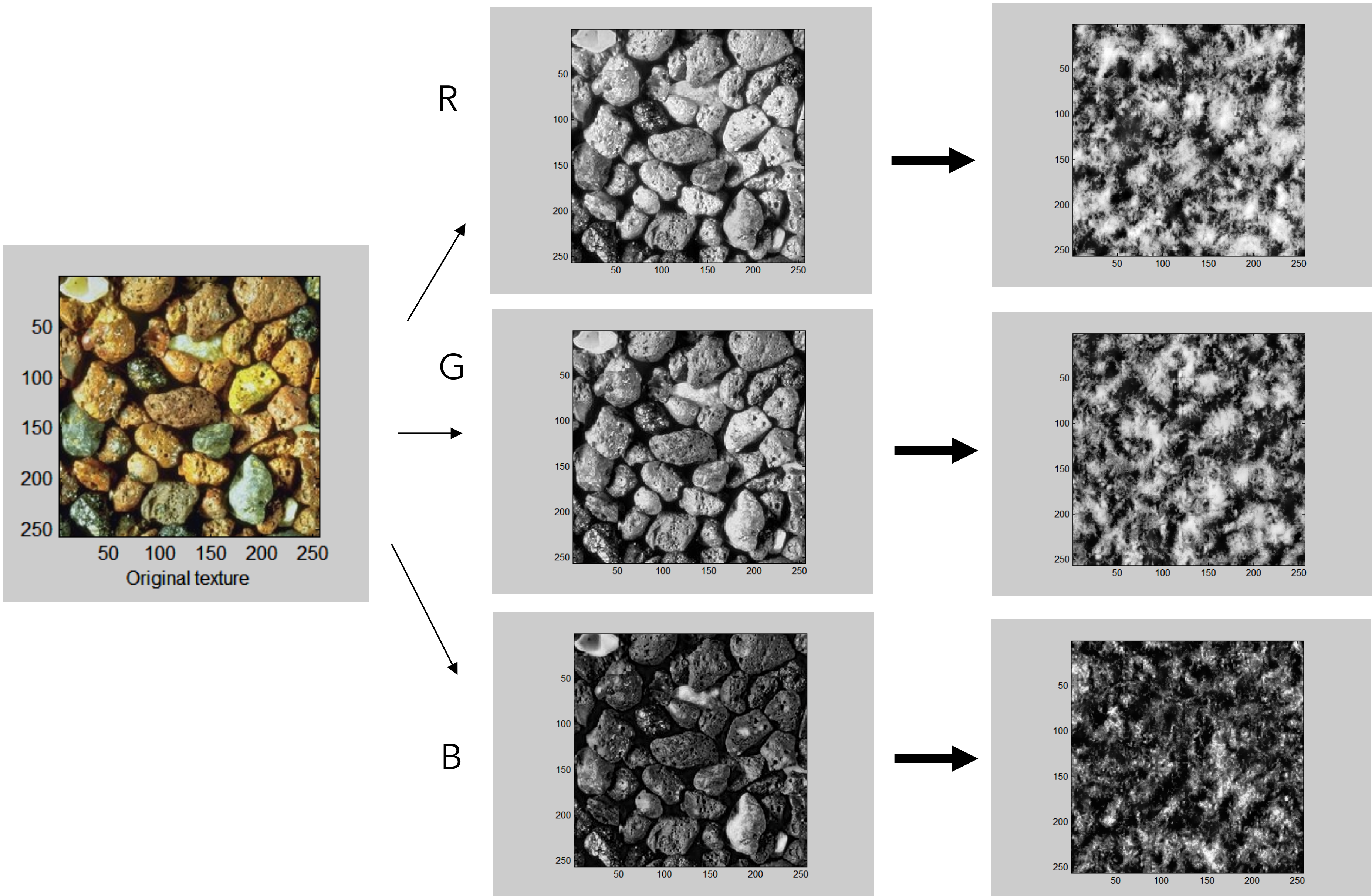
red = target histogram, blue = current iteration

Color textures



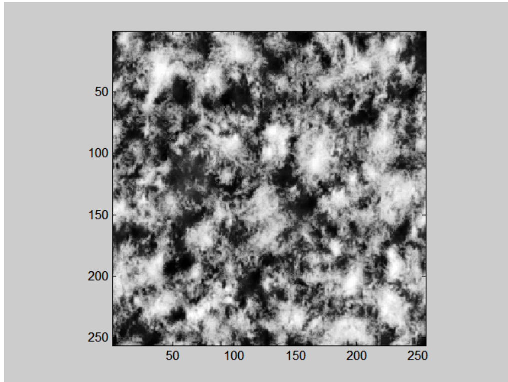
Three textures

Color textures

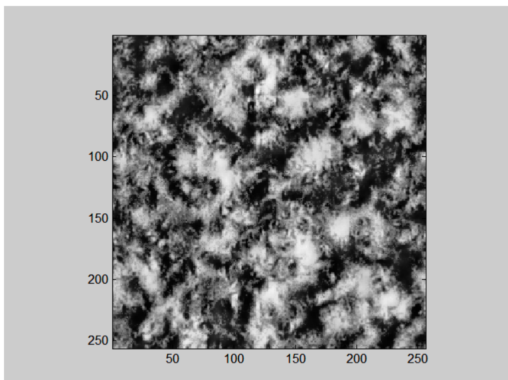


Color textures

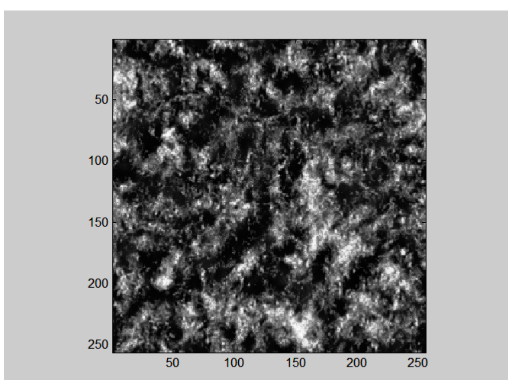
R



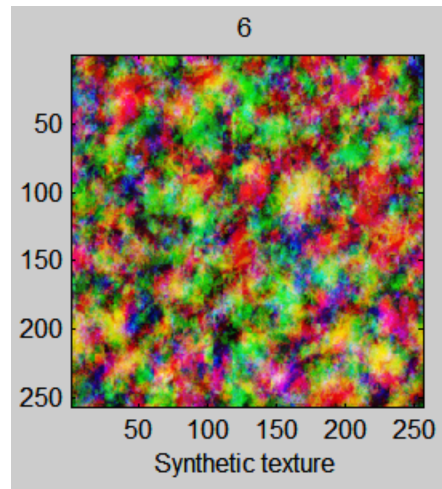
G

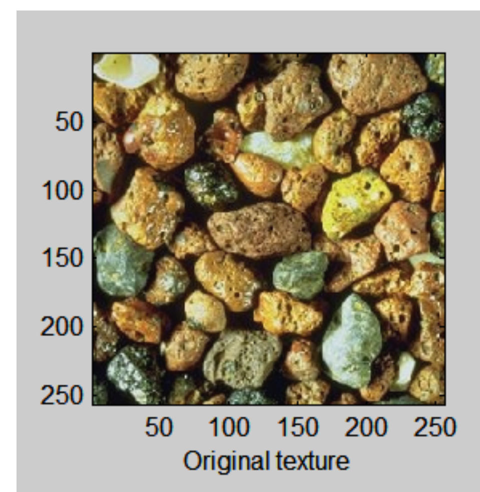


B



This does not work

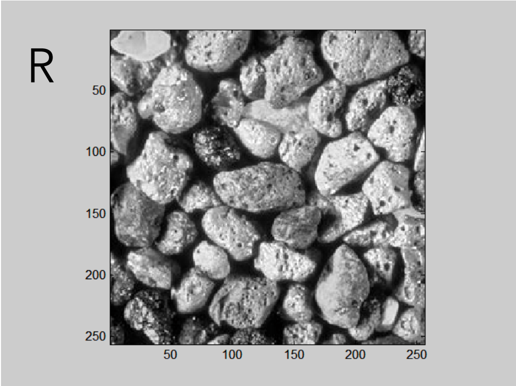


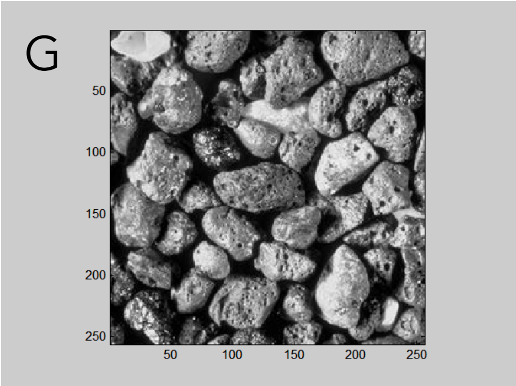


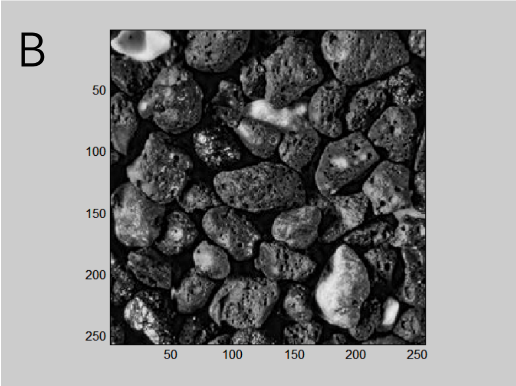
Color textures

Problem: we create new colors not present in the original imag

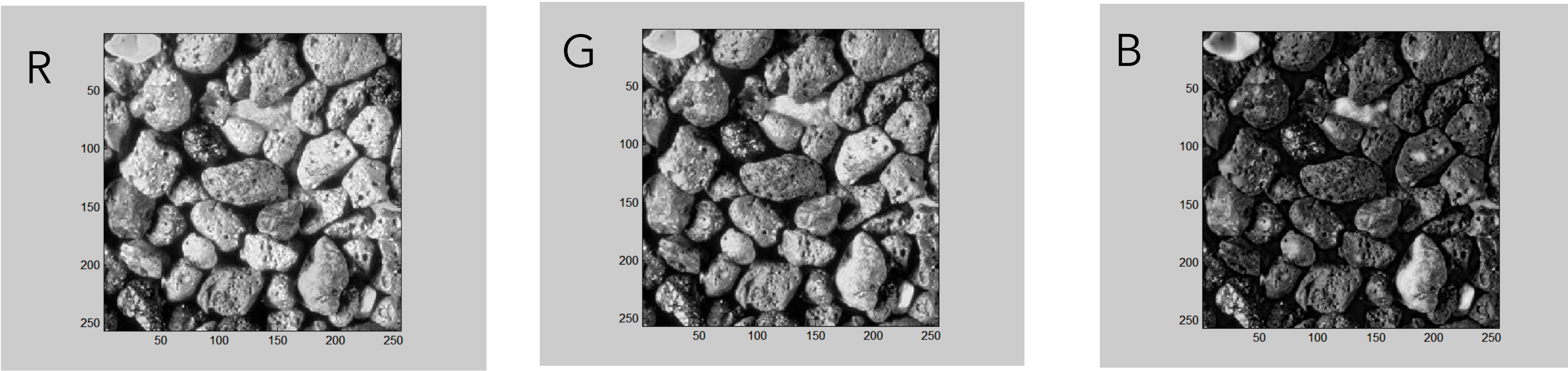
Why? Color channels are not independent.



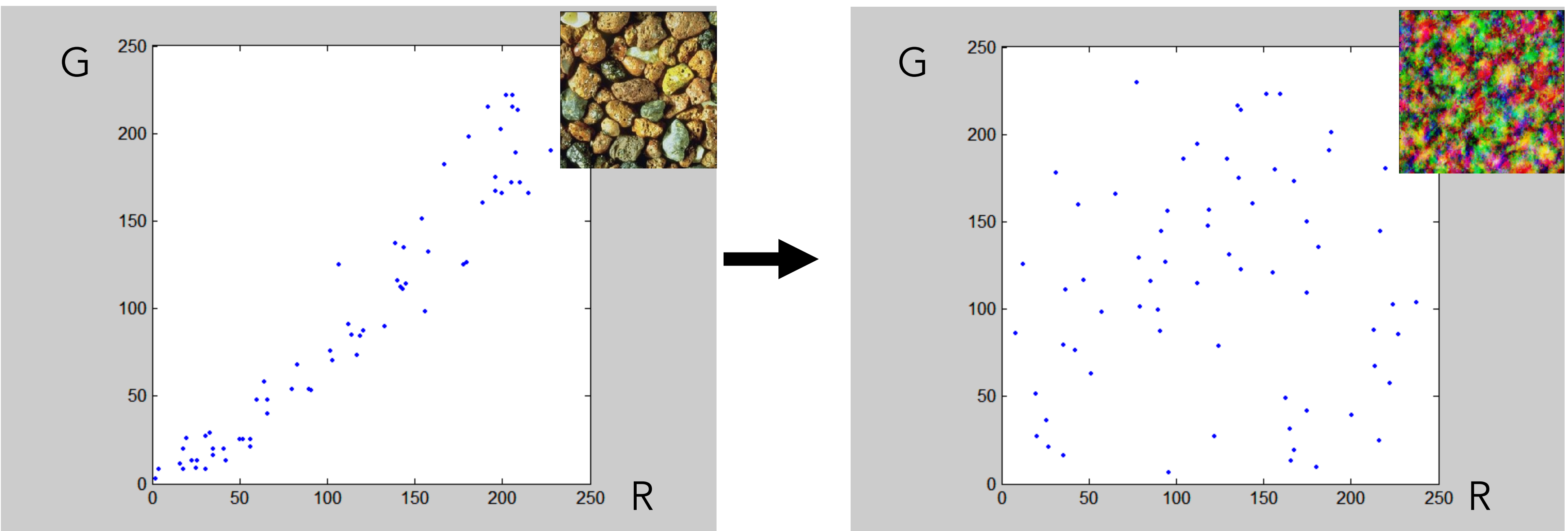




Principal Components Analysis (PCA) and decorrelation

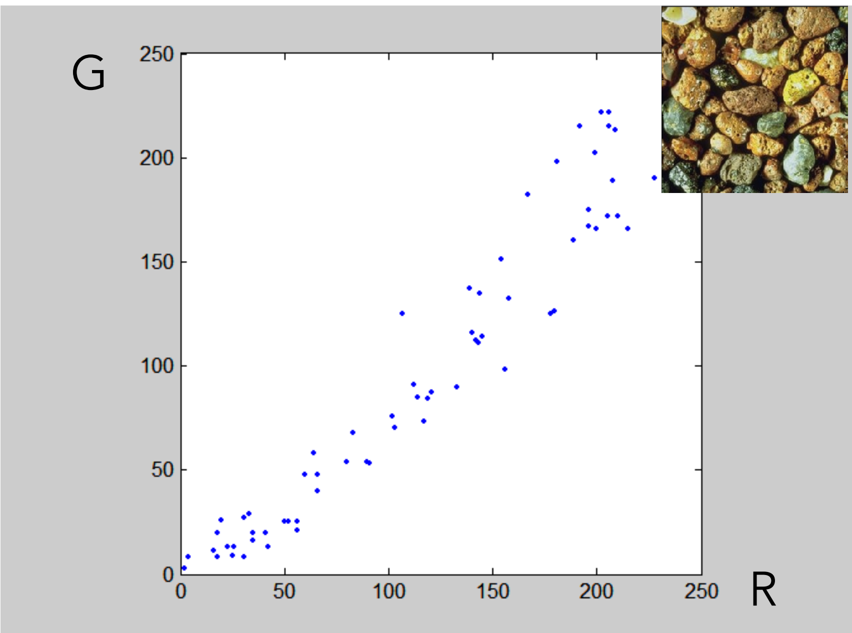


In the original image, R and G are correlated, but, after synthesis,…

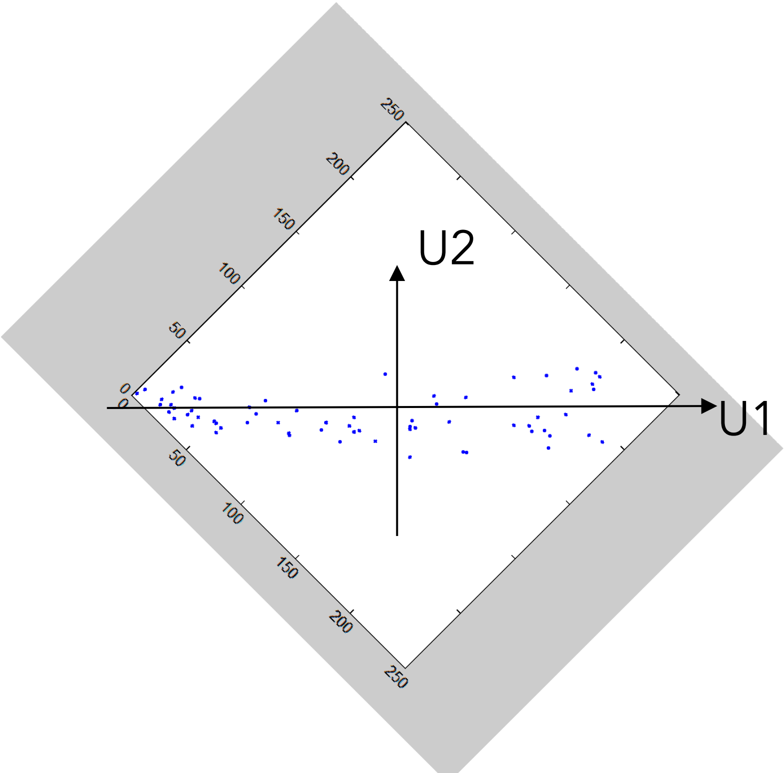


PCA and decorrelation

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

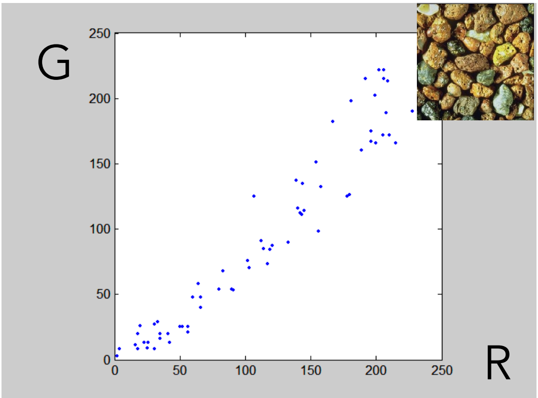


Rotation



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

PCA and decorrelation



correlation(R,G)

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

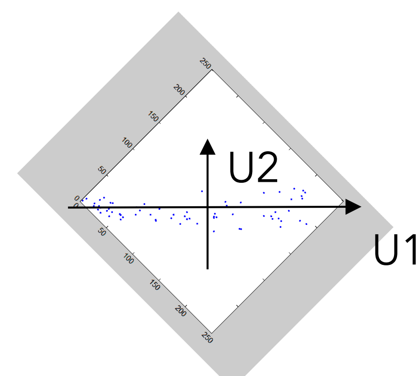
PCA finds the principal directions of variation of the data.
It gives a decomposition of the covariance matrix as:

$$C = D\,D'$$

$$D = \begin{matrix} 0.6347 & 0.6072 & 0.4779 \\ 0.6306 & -0.0496 & -0.7745 \\ 0.4466 & -0.7930 & 0.4144 \end{matrix}$$

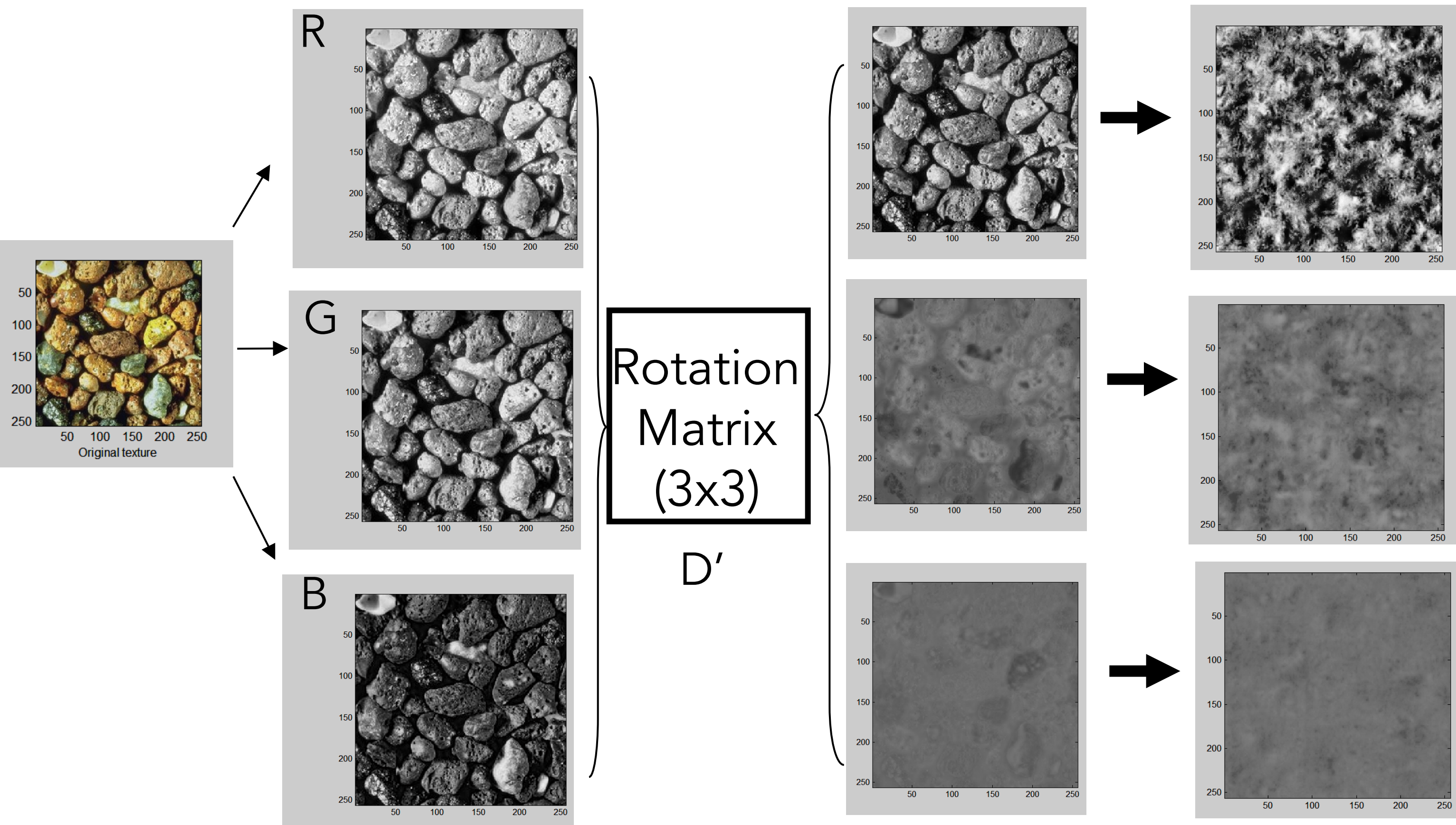
By transforming the original data (RGB) using D we get:

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



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

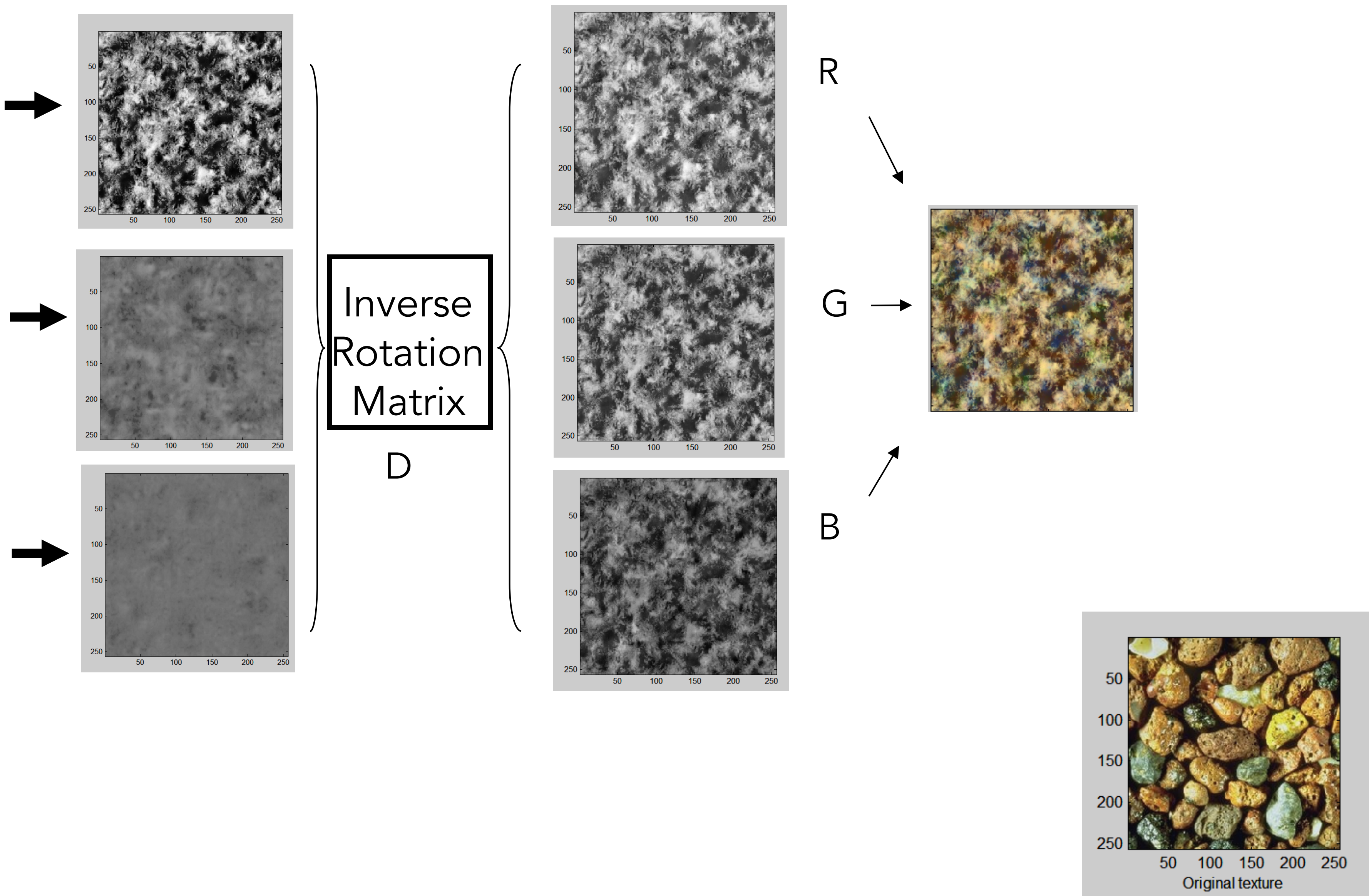
Color textures



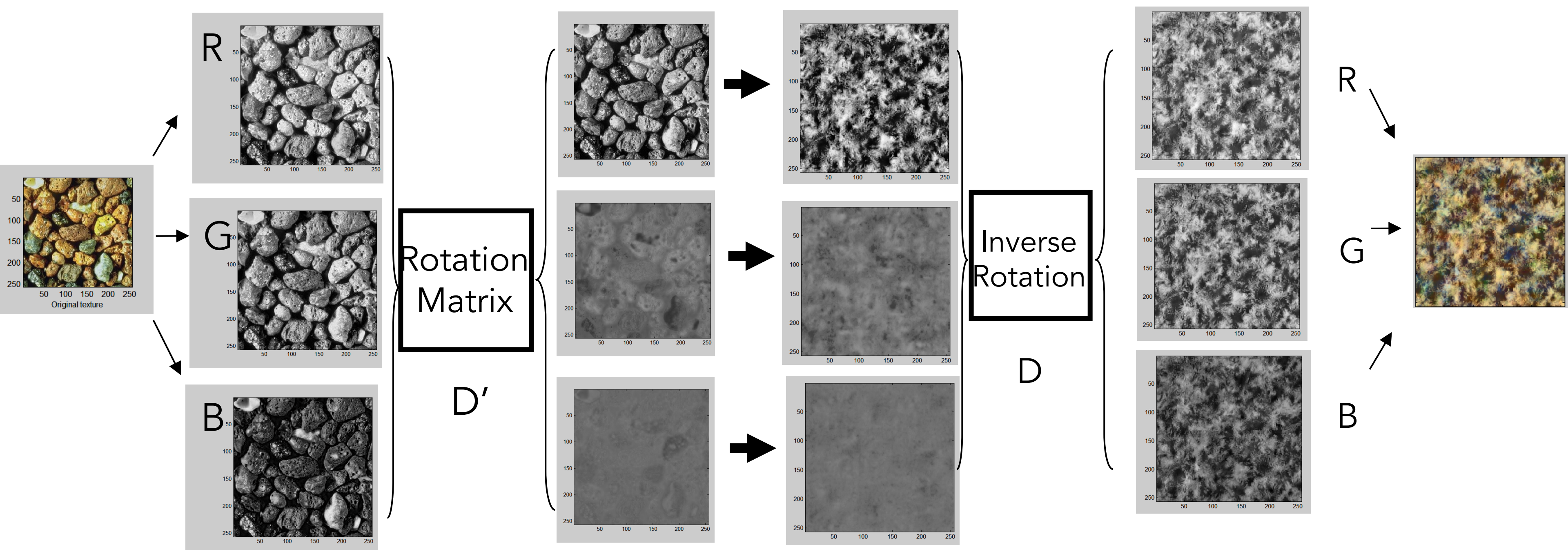
These three textures look similar (high dependency)

These three textures Look less similar (lower dependency)

Color textures



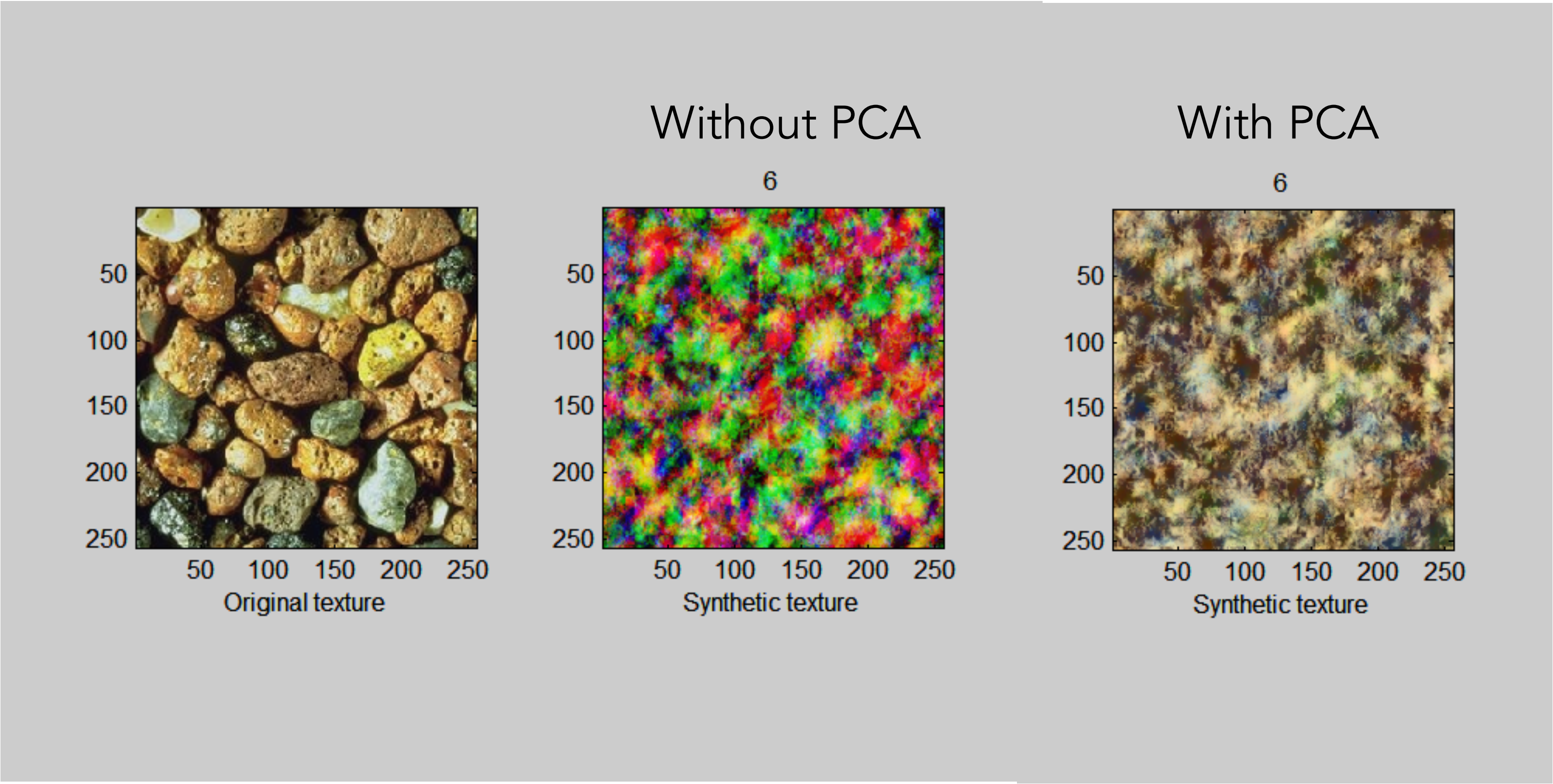
Color textures



These three textures
look similar
(high dependency)

These three textures
Look less similar
(lower dependency)

Color channels



Examples from the paper

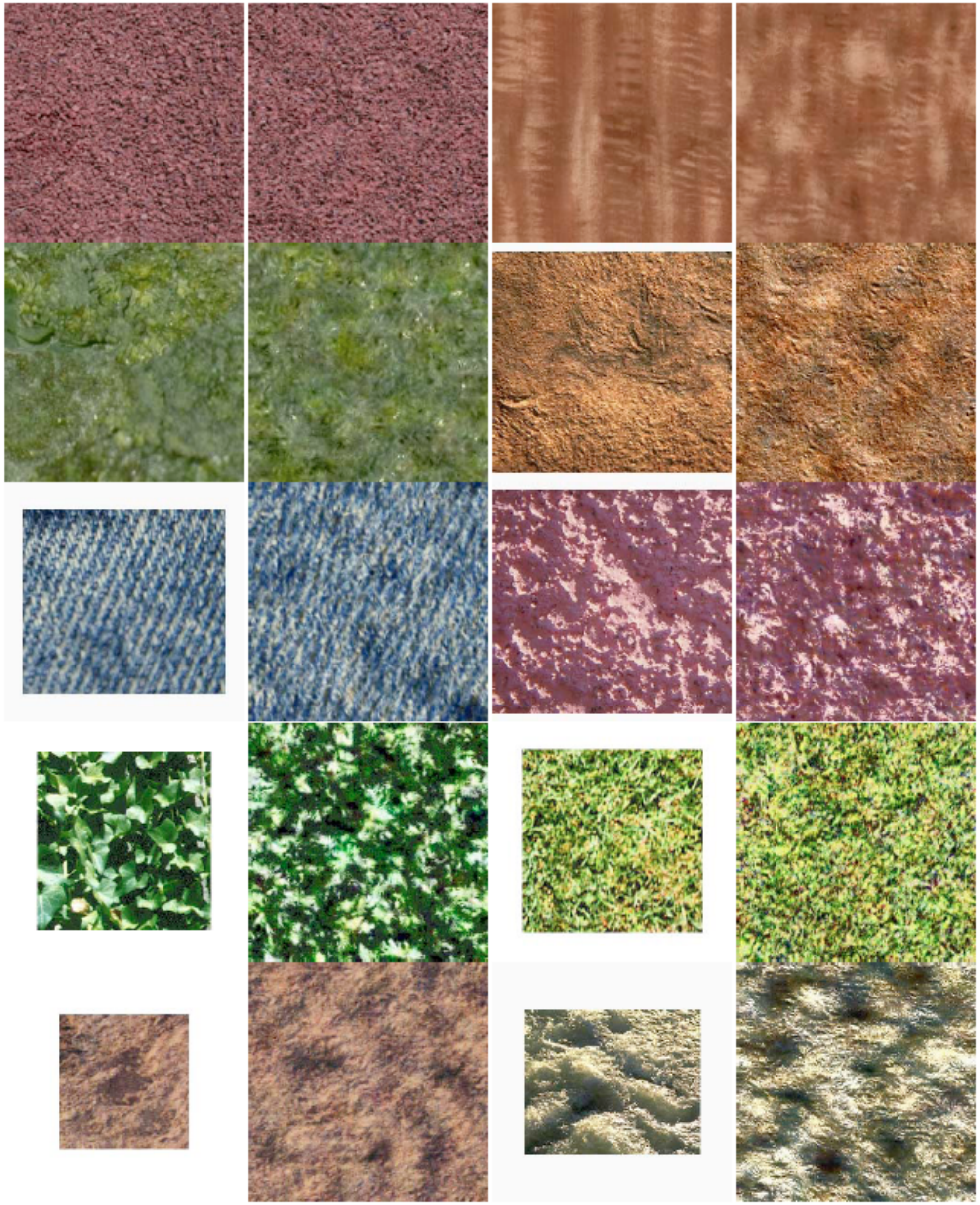
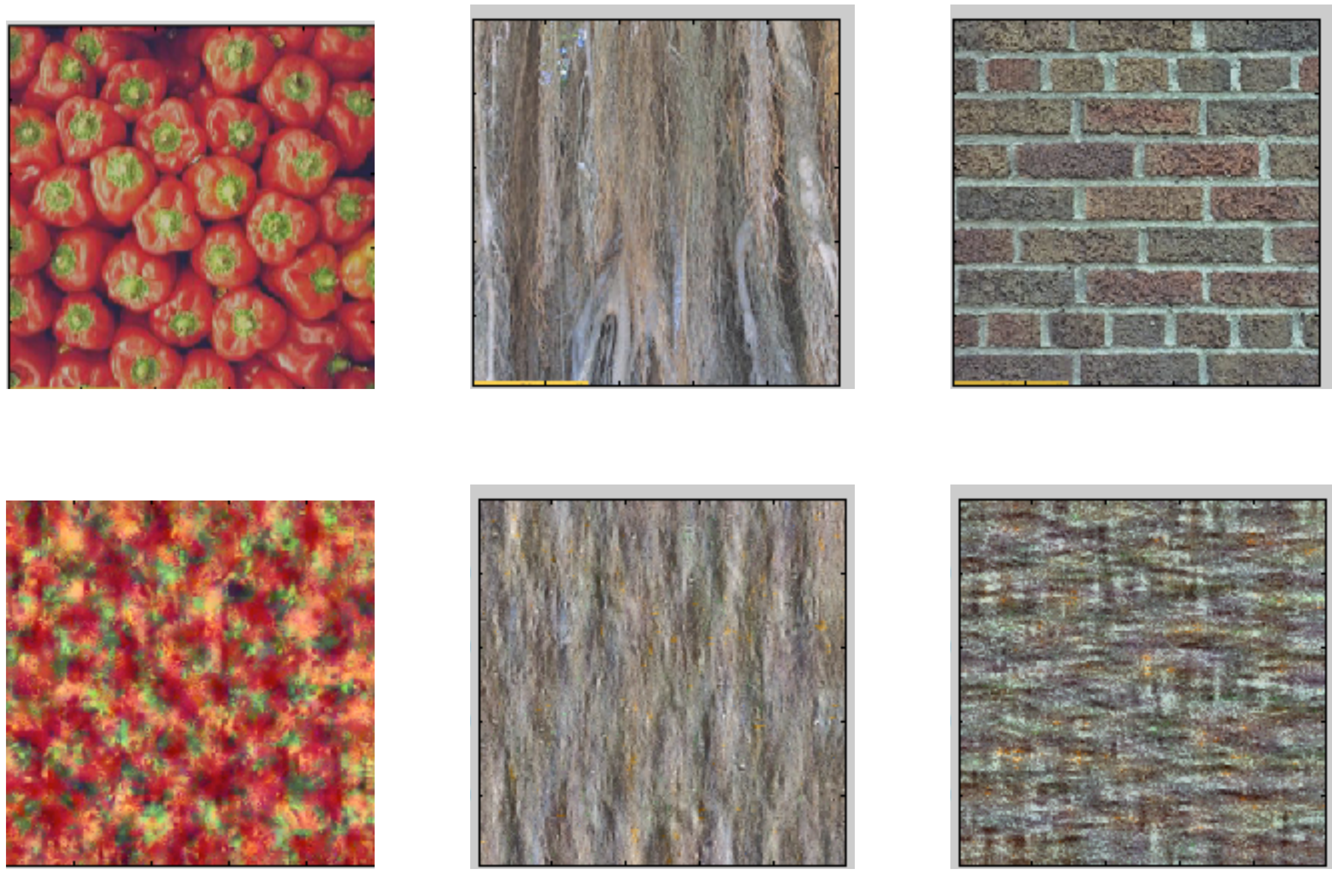


Figure 4: In each pair left image is original and right image is synthetic: red gravel, figured sepele wood, brocolli, bark paper, denim, pink wall, ivy, grass, sand, surf.

Examples not from the paper

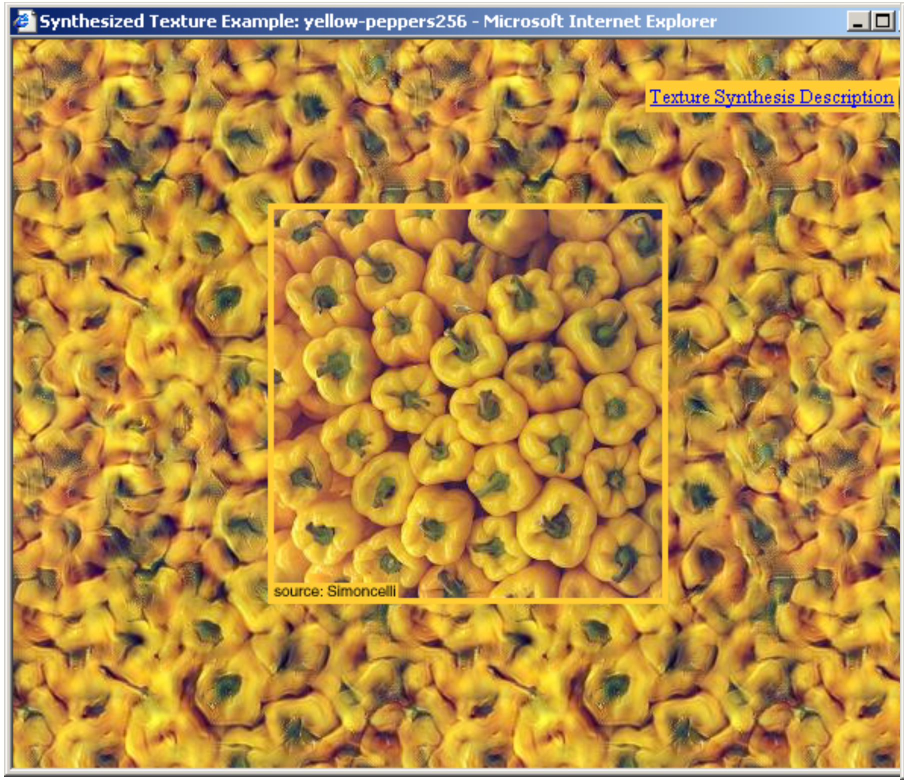
Input texture

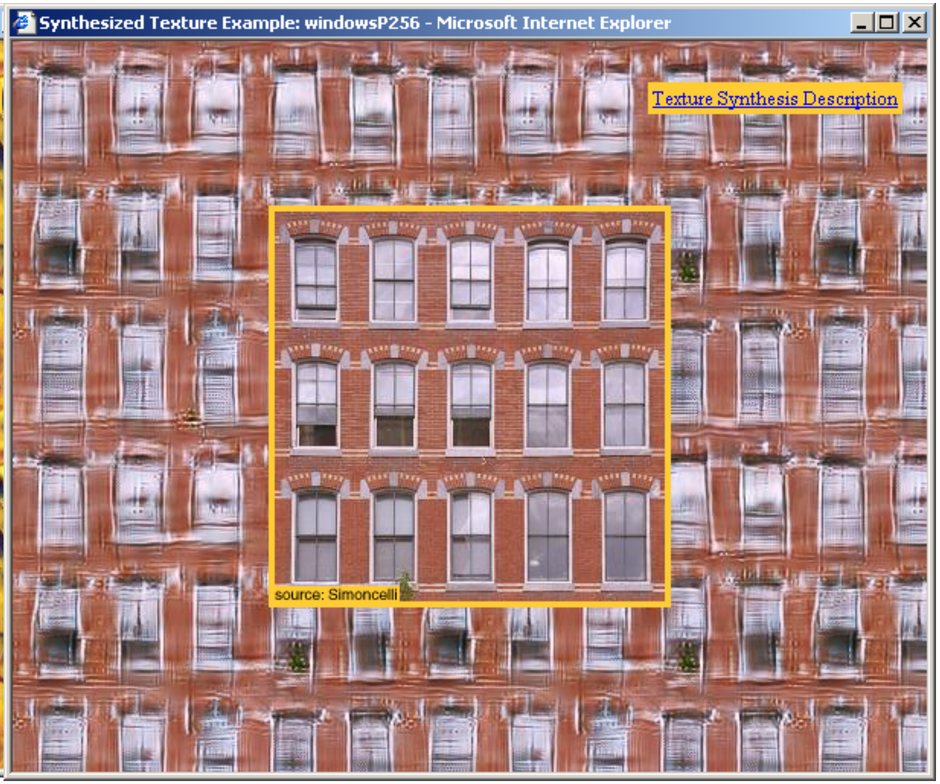
Synthetic texture



It does not keep much of the structure for these textures

Portilla and Simoncelli (2001)





Same principle than previous method but using more statistics

Four statistics

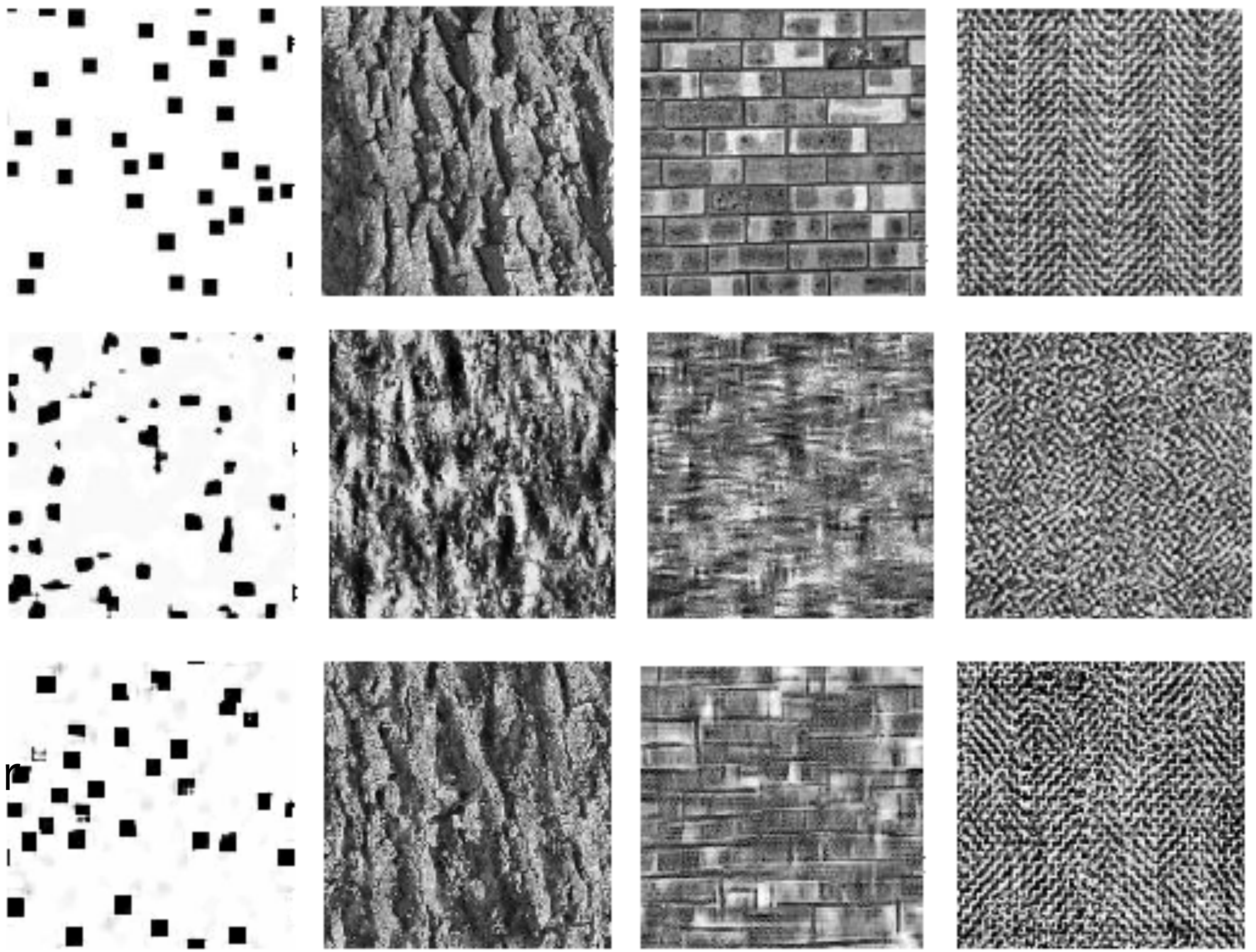
- Marginal Statistics
- Coefficient Correlation
- Magnitude Correlation
- Cross-Scale Phase Statistics

Texture analysis and synthesis

Original

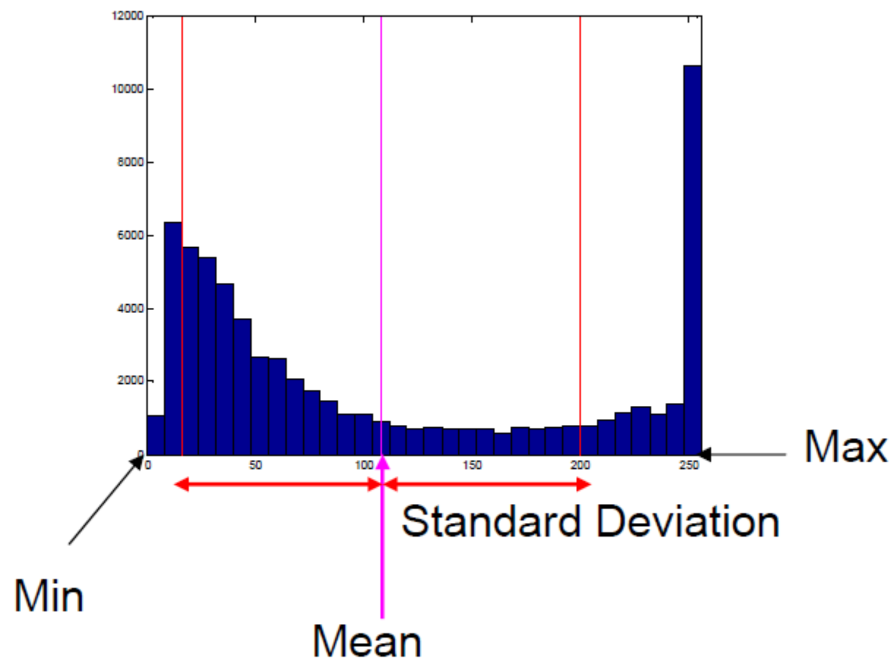
Marginal Histograms
(Heeger-Bergen)

Higher order statistics

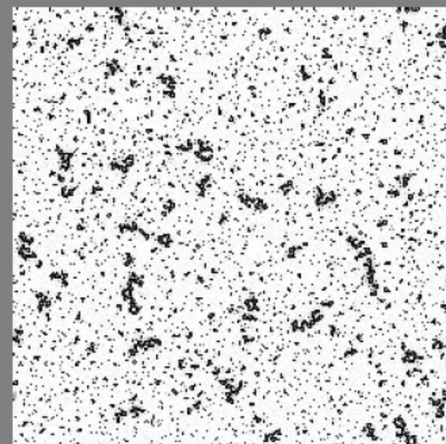


Marginal Statistics

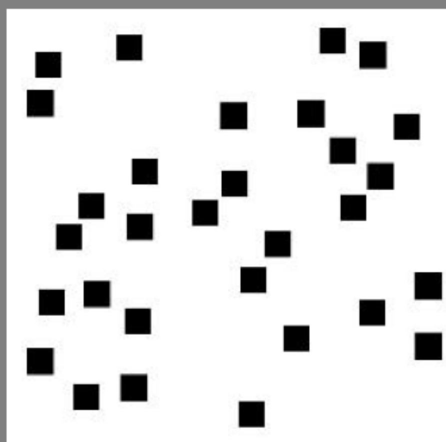
- Pixel statistics: Mean, Variance, Skew, Kurtosis, Min and Max



Marginal Statistics

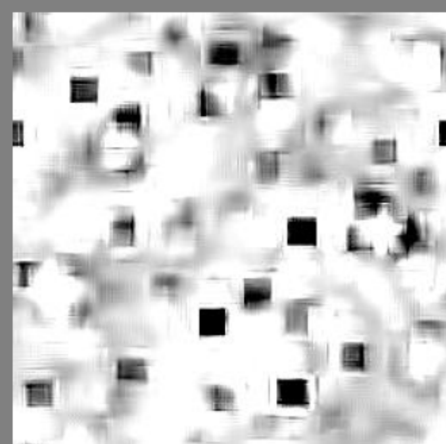


Marginals only



Original

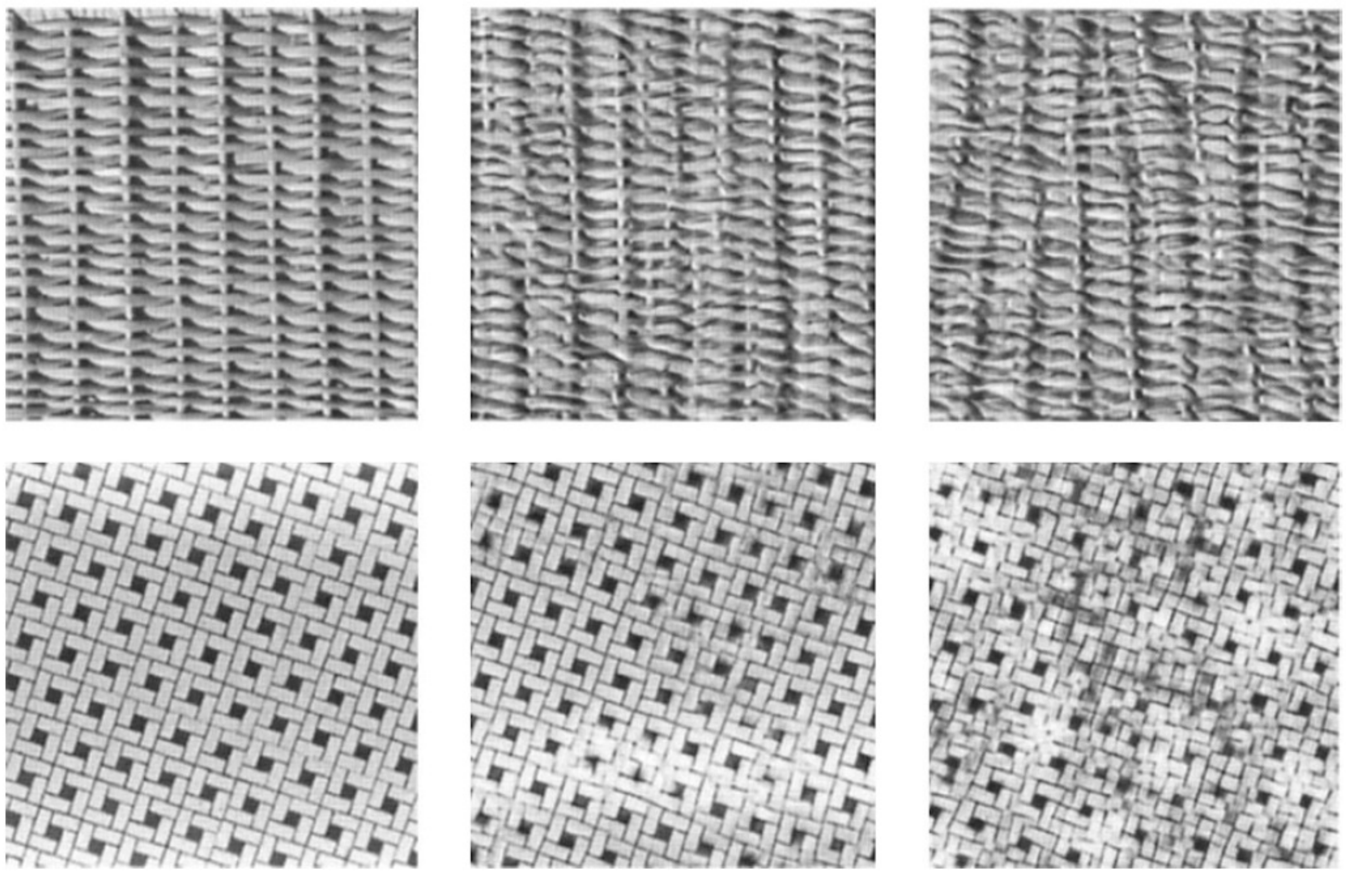
Full set



All except marginals

(2) Coefficient correlation

It captures periodic or globally oriented structure (within a neighborhood size, e.g. 9 pixels). The local correlation of each subband. It characterizes the salient spatial frequencies and the regularity of the texture, as represented by periodic or globally oriented structure



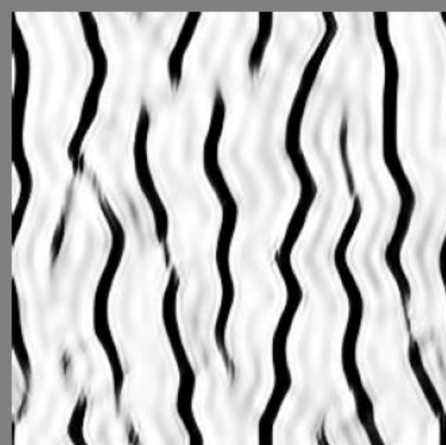
All parameters

All but coefficient correlation

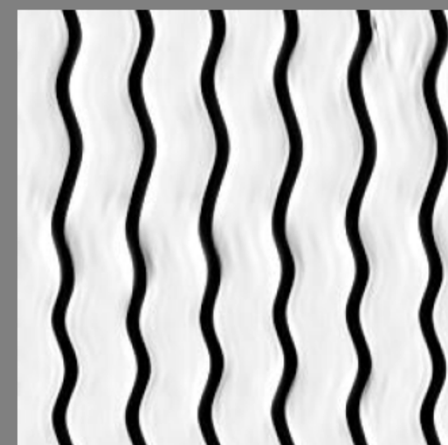
Raw coefficient correlation

Original

Raw corr + Marginals



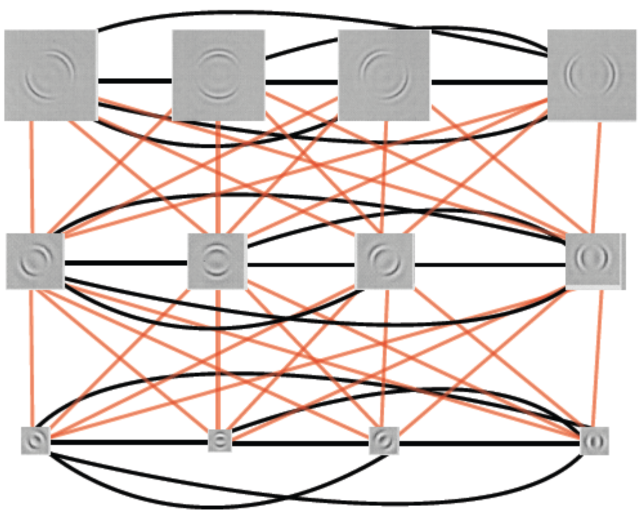
All except raw corr



Full set

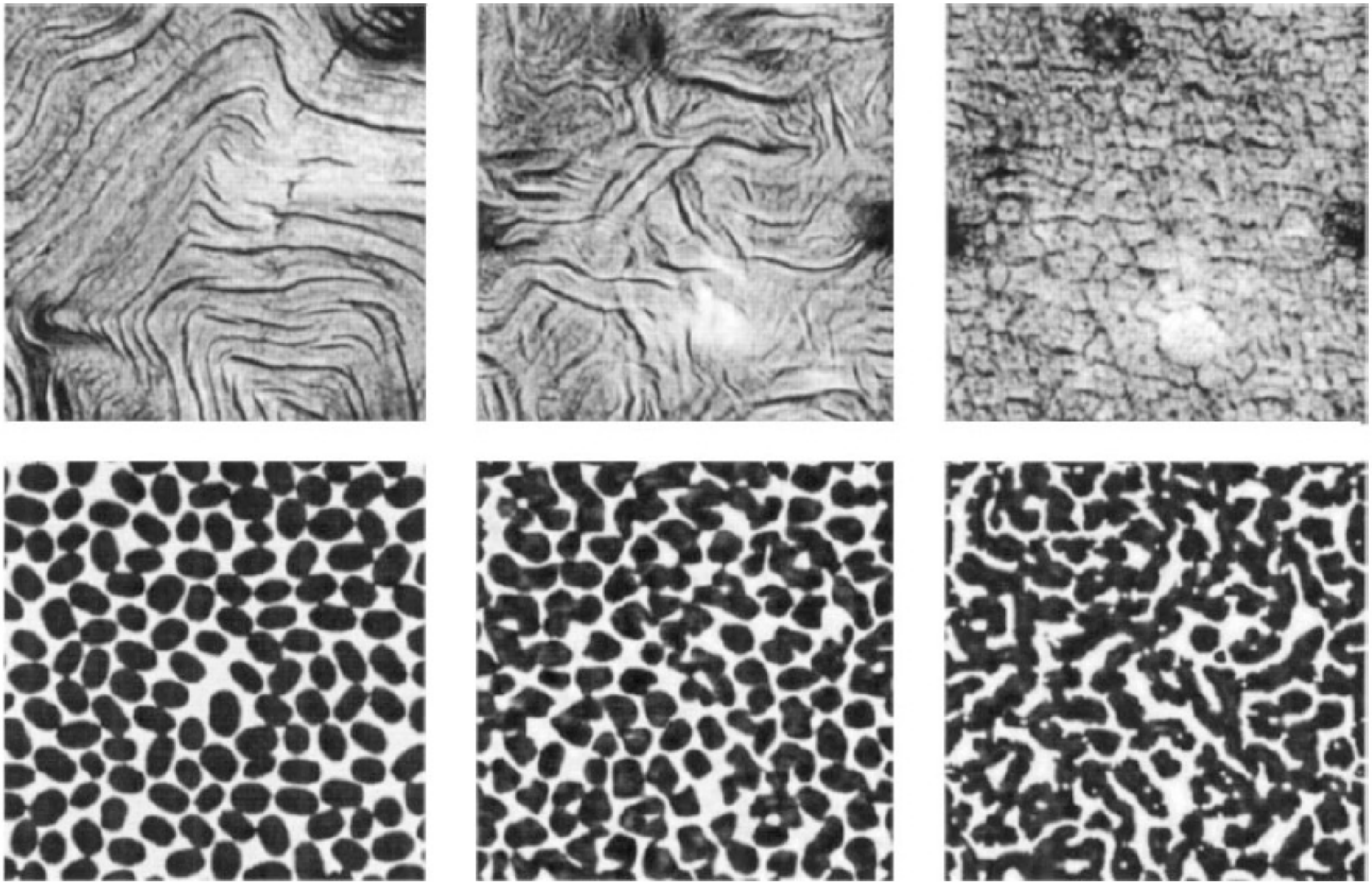
(3) Magnitude correlation

Capture structure (edges, bars, corners) and “second-order” textures.
cross-correlation of each subband magnitudes with those of other orientations at the same scale, and cross-correlation of each subband magnitude with all orientations at a coarser scale.



Black = Cousin Cross-correlation
Red = Parent Cross-correlation

(3) Magnitude correlation

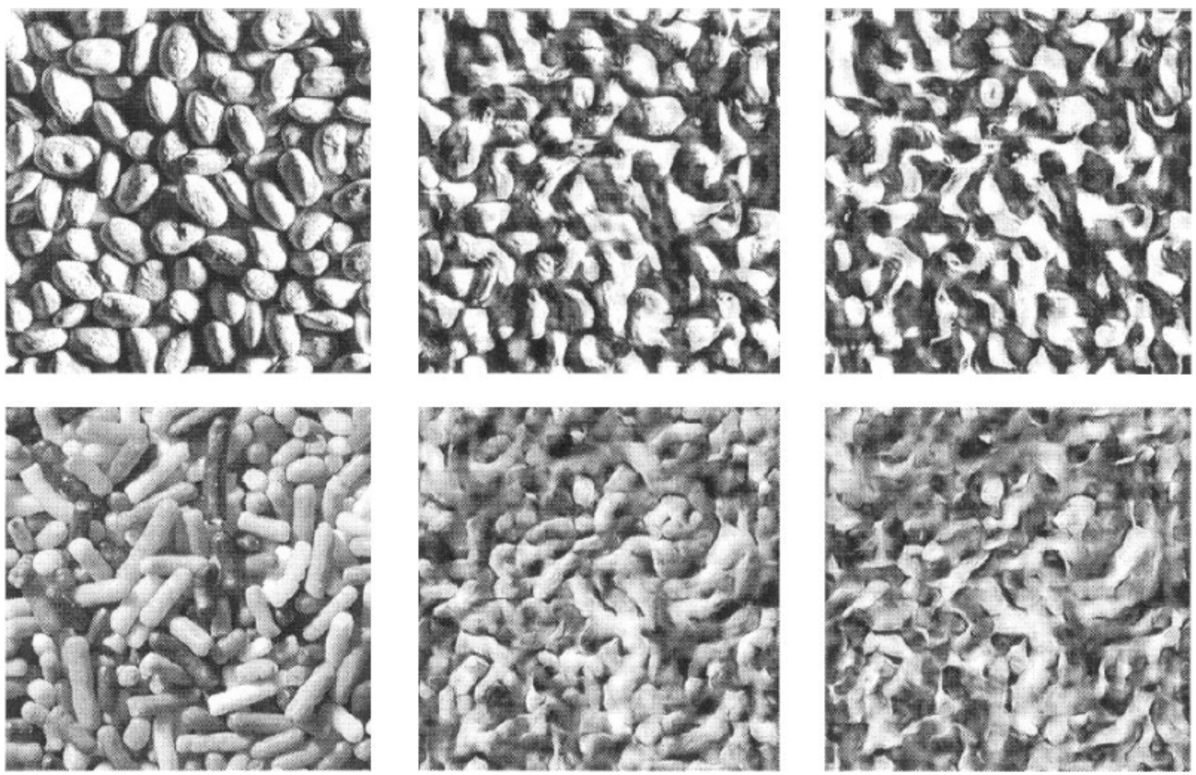


All parameters

All but magnitude correlation

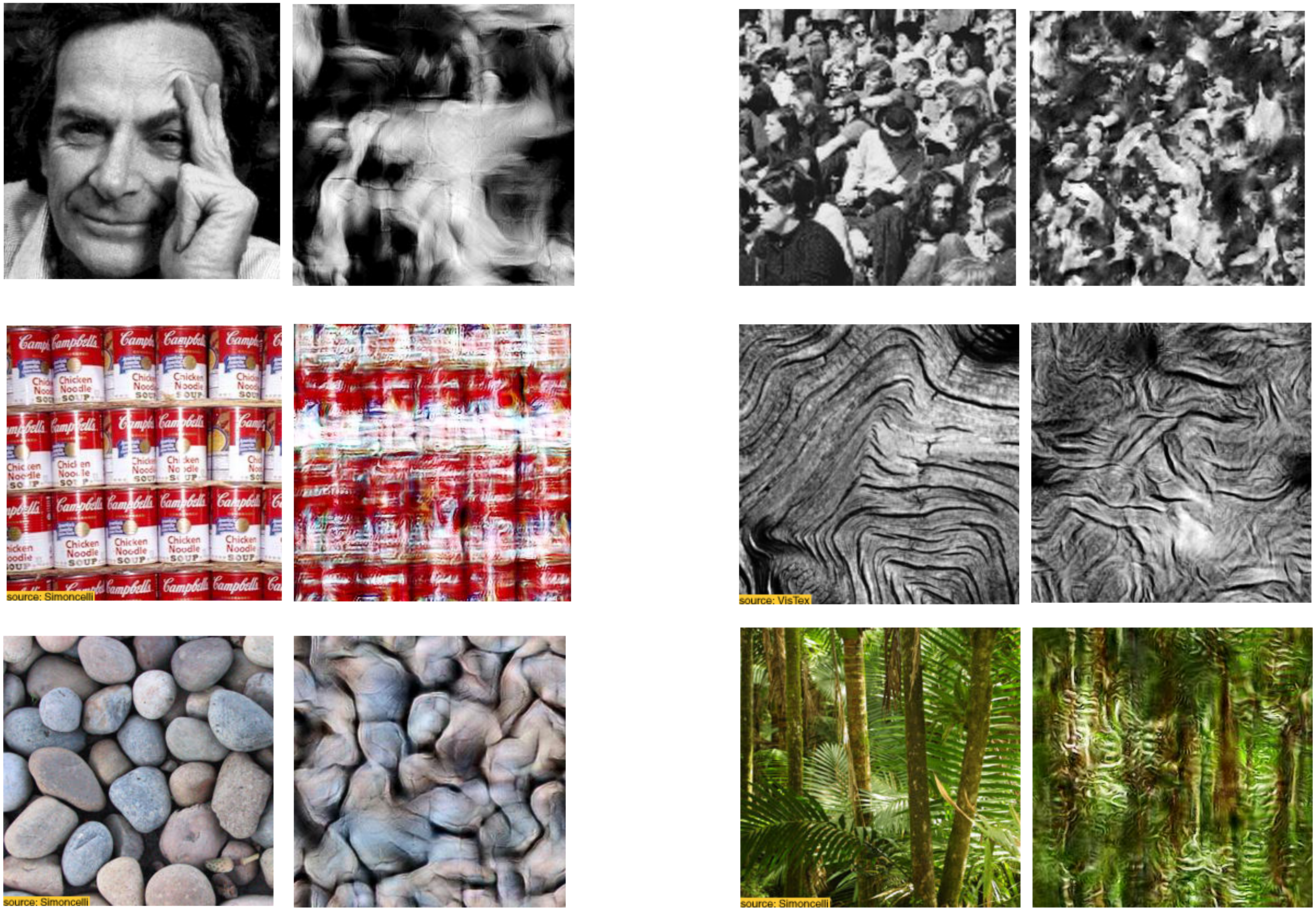
(4) Cross-scale phase statistics

<u>Cross-scale phase statistics:</u> Distinguishes edges from lines. Help represented gradients/lighting effects. A local representation of the phase (position), in order to represent edges and lines. Important to represent 3dimensional aspect and shadows, and more generally gradients due to lighting effects.

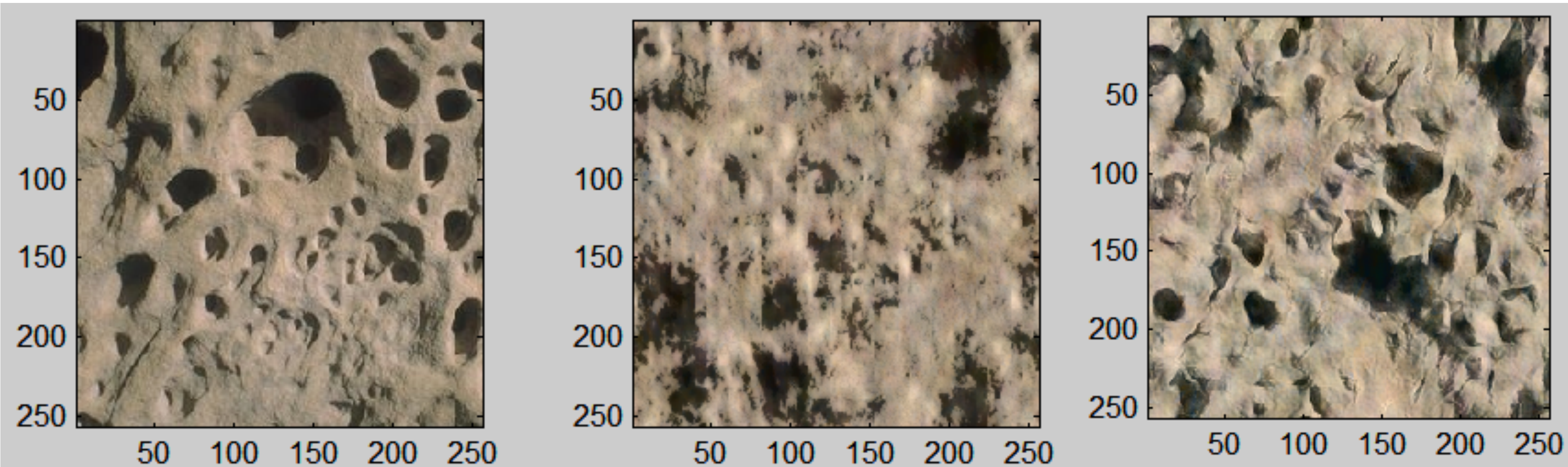


All parameters   All but phase statistics

Portilla & Simoncelli



Portilla & Simoncelli



Heeger & Bergen

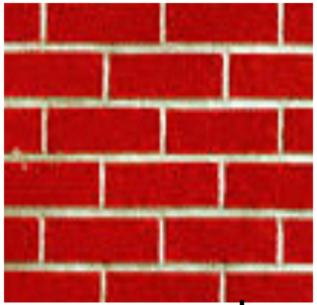
Portilla & Simoncelli

Two big families of models

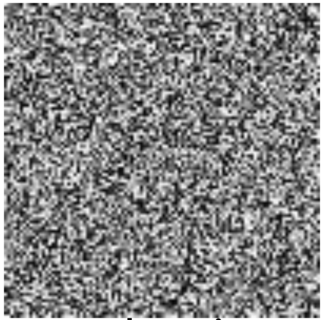
II-Example-based non-parametric models

The Challenge

- Texture analysis: how to capture the essence of texture?
- Need to model the whole spectrum: from repeated to stochastic texture
- This problem is at intersection of vision, graphics, statistics, and image compression



repeated



stochastic

Both?

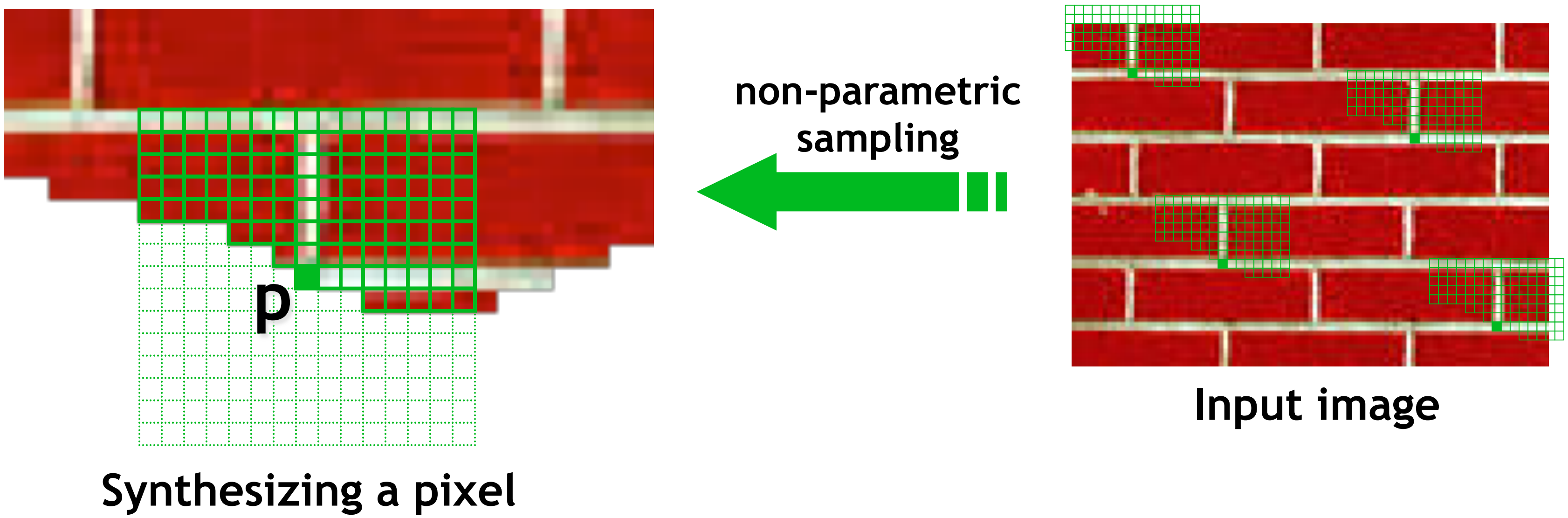
IEEE International Conference on Computer Vision, Corfu, Greece, September 1999

Texture Synthesis by Non-parametric Sampling

Alexei A. Efros and Thomas K. Leung
Computer Science Division
University of California, Berkeley
Berkeley, CA 94720-1776, U.S.A.
{efros,leungt}@cs.berkeley.edu

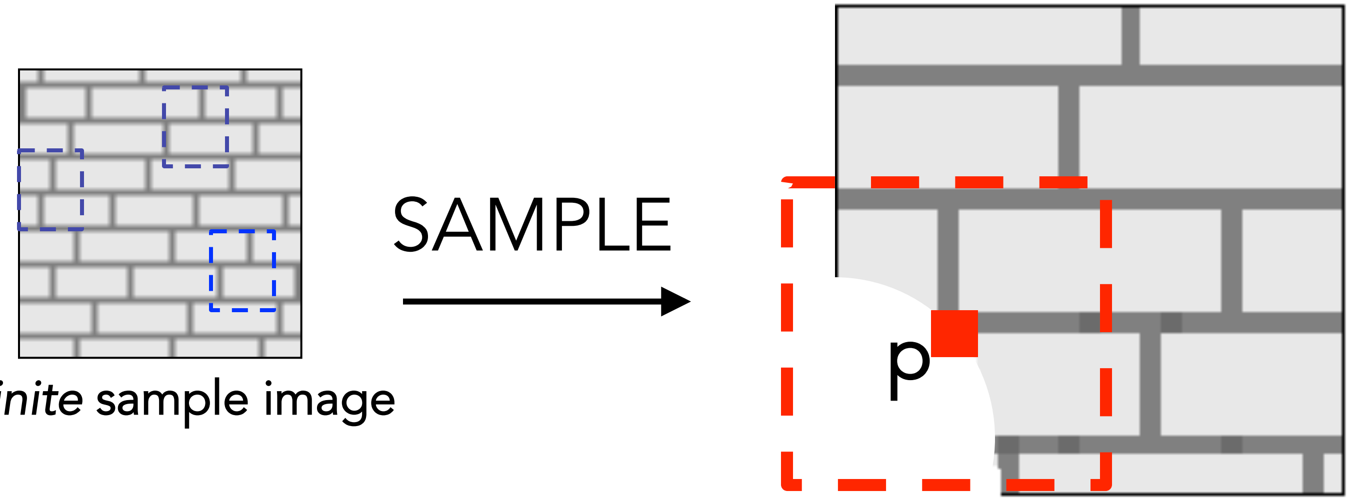
See section 9.3 Forsyth Ponce textbook (2003) – pdf given

Efros & Leung Algorithm



– *Search the input image* for all similar neighborhoods pixels to p

Non parametric texture synthesis



finite sample image

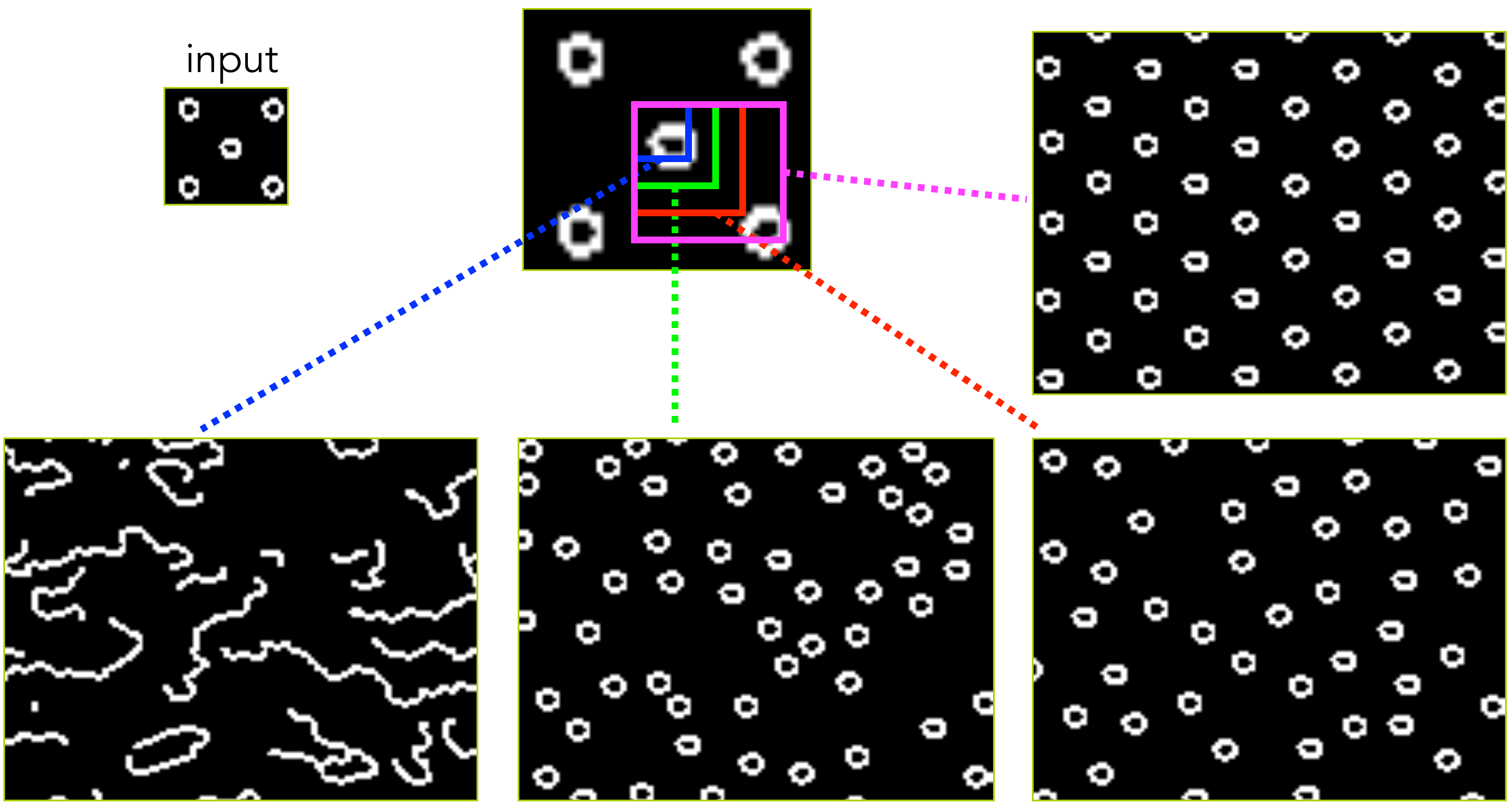
Generated image

- let's directly search the input image for all similar neighbourhoods pixels to produce a histogram for p

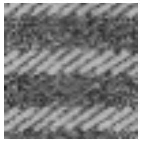
Growing Texture

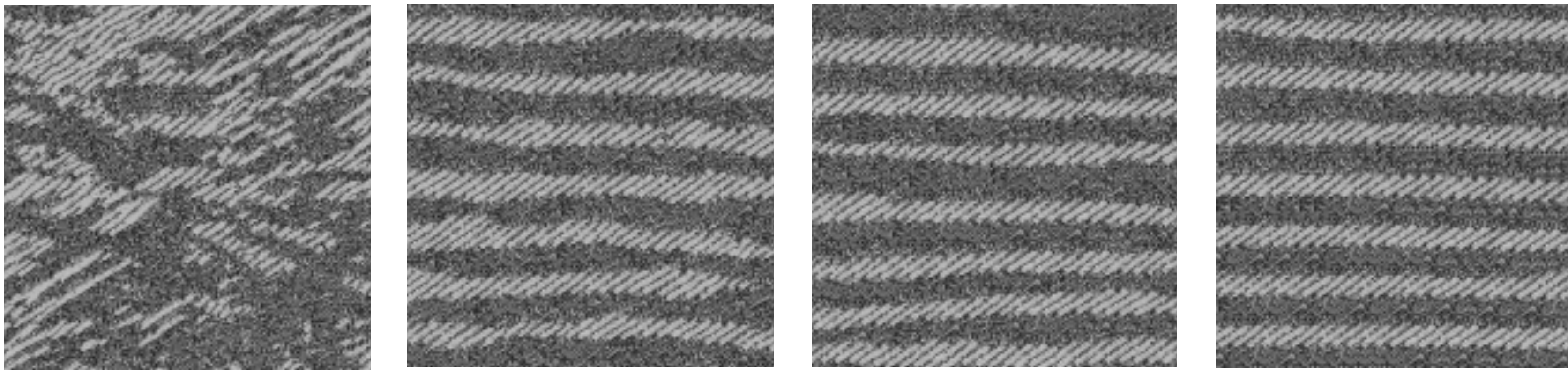
- Starting from the initial configuration, we "grow" the texture one pixel at a time
- The size of the neighbourhood window is a parameter that specifies how stochastic (random) the user believes this texture to be
- To grow from scratch, we use a random 3x3 patch from input image as seed.
- Pixels with most neighbors are synthesized first. If no close match can be found, the pixel is not synthesized until the end

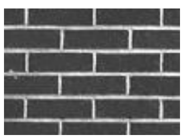
Neighborhood Window

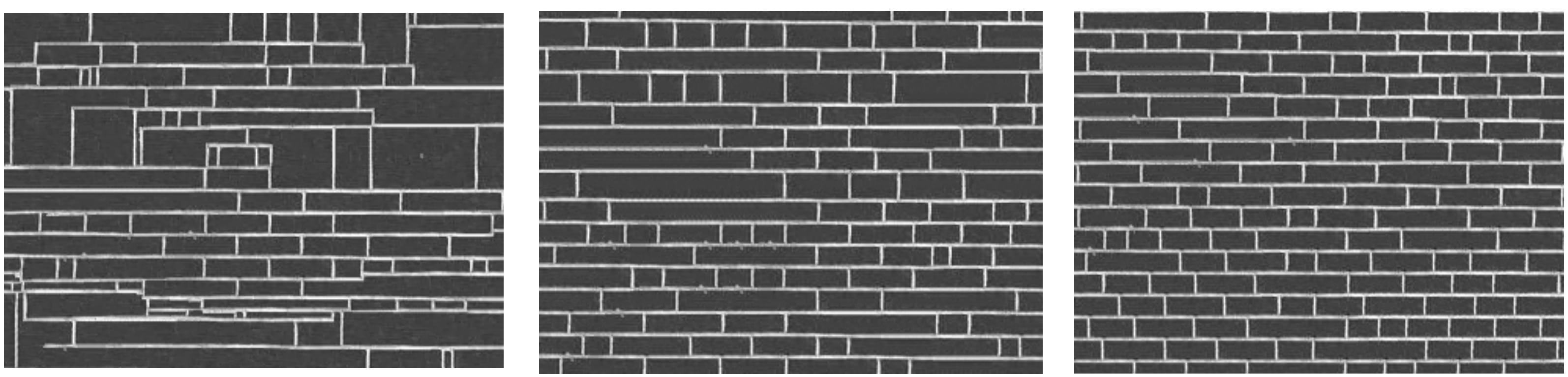


Varying Window Size





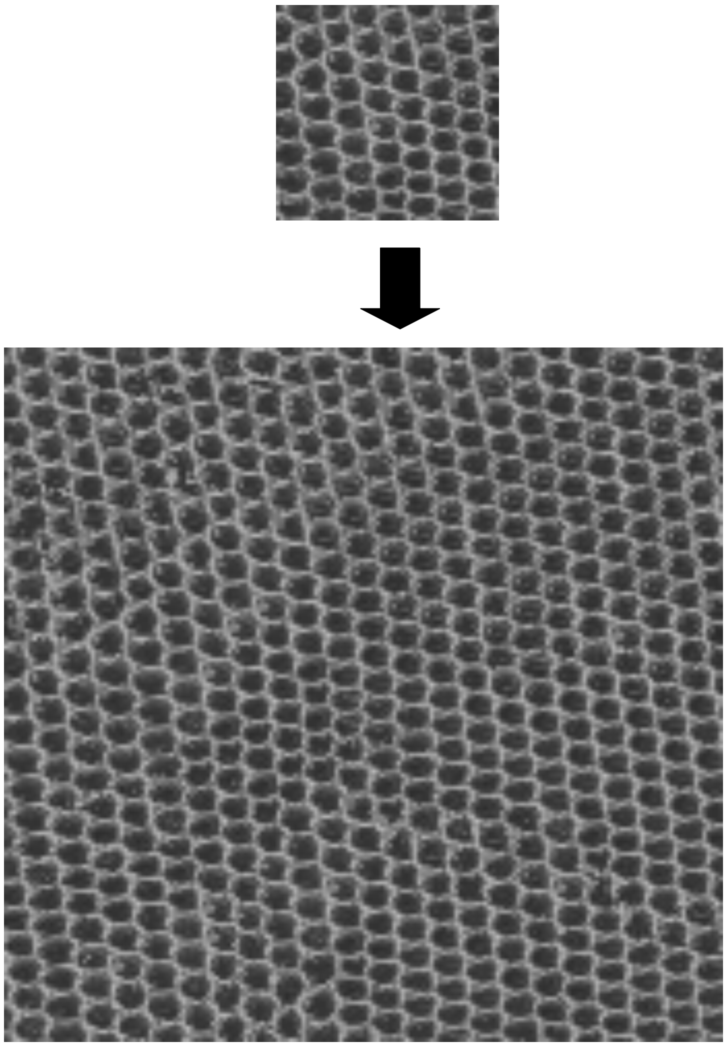




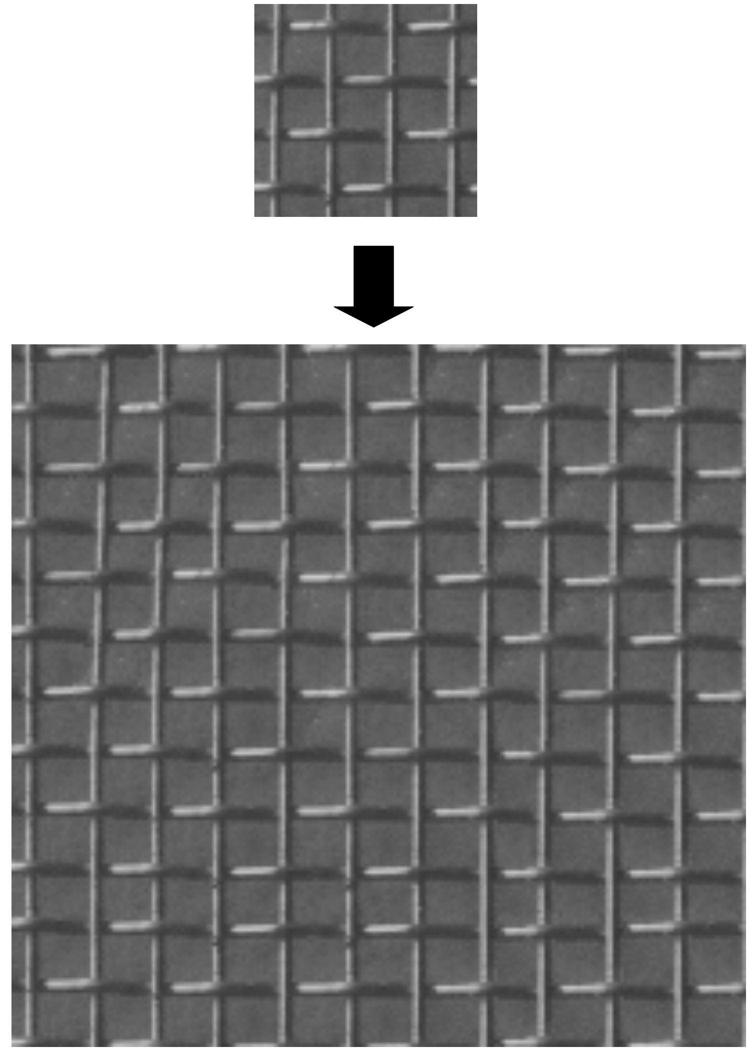
Increasing window size

Brodatz Results

reptile skin

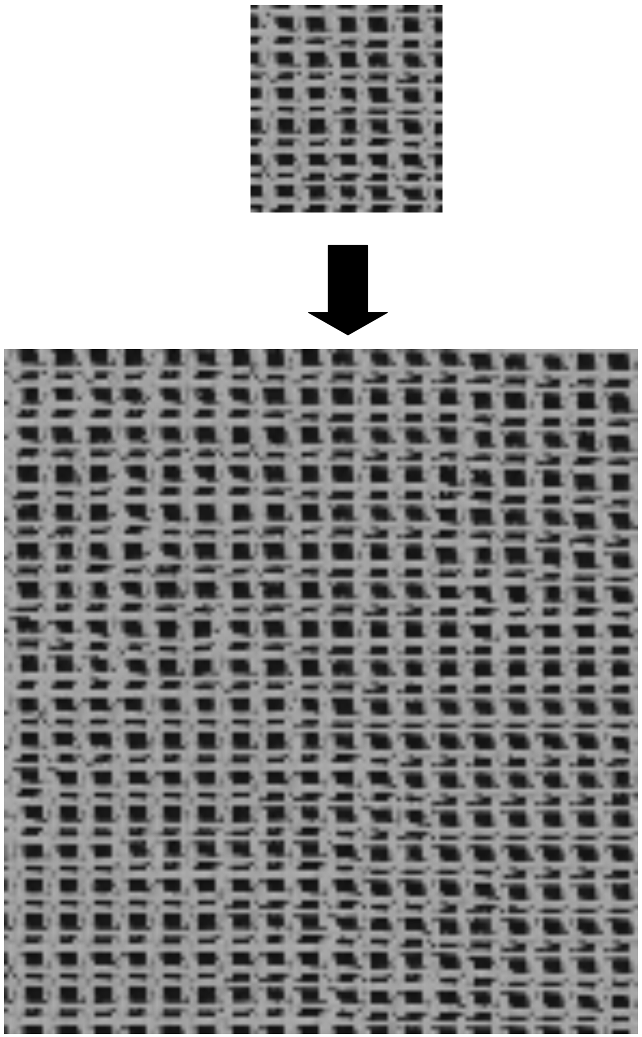


aluminum wire

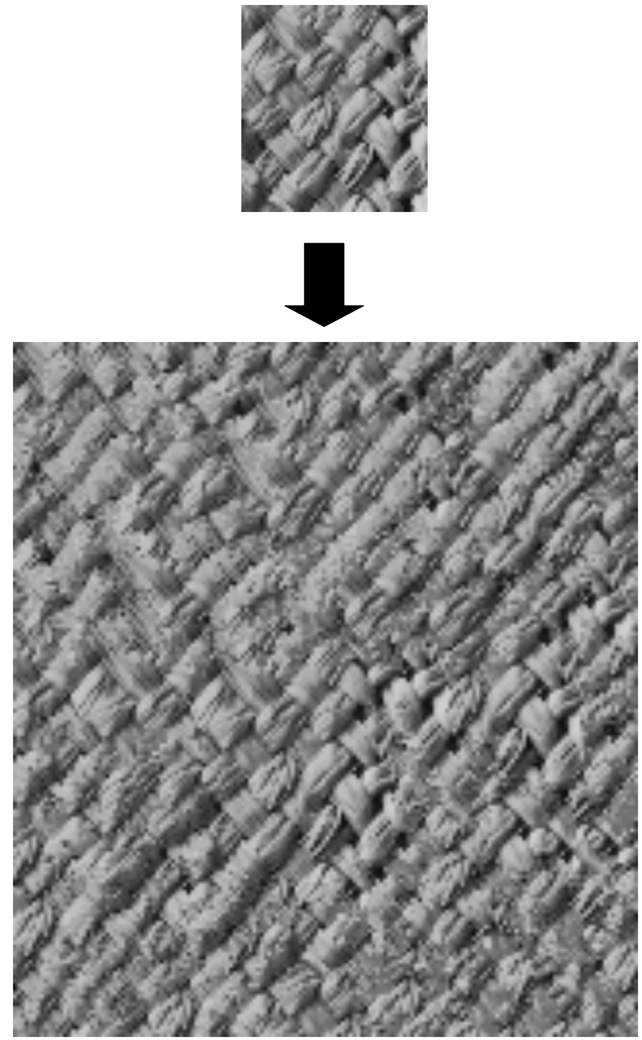


More Brodatz Results

french canvas

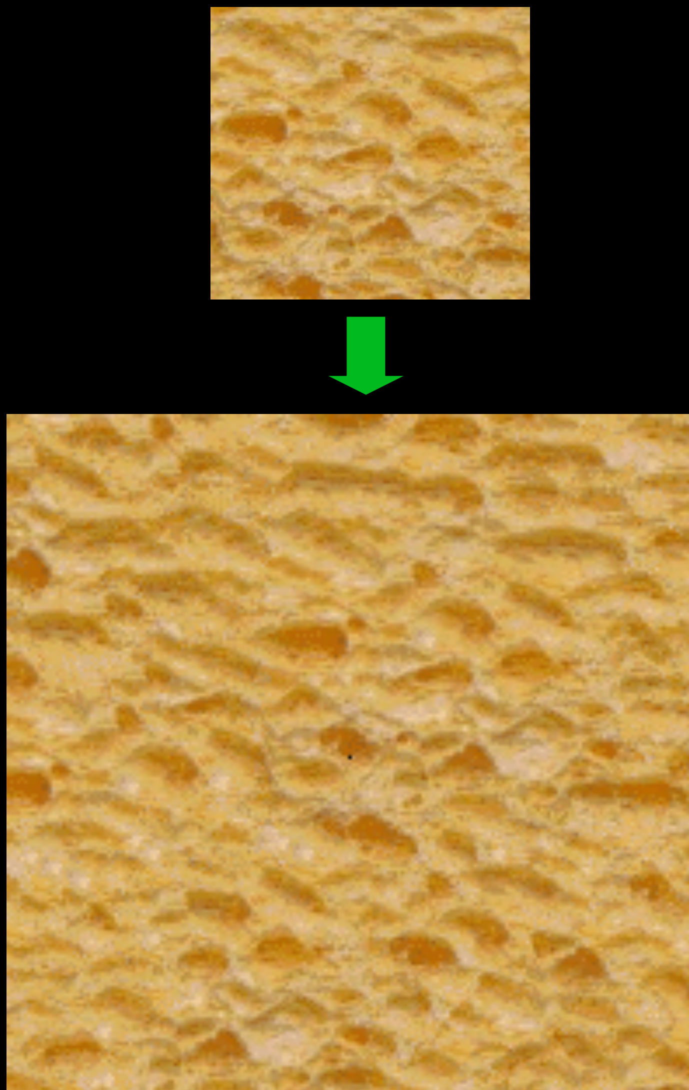


rafia weave

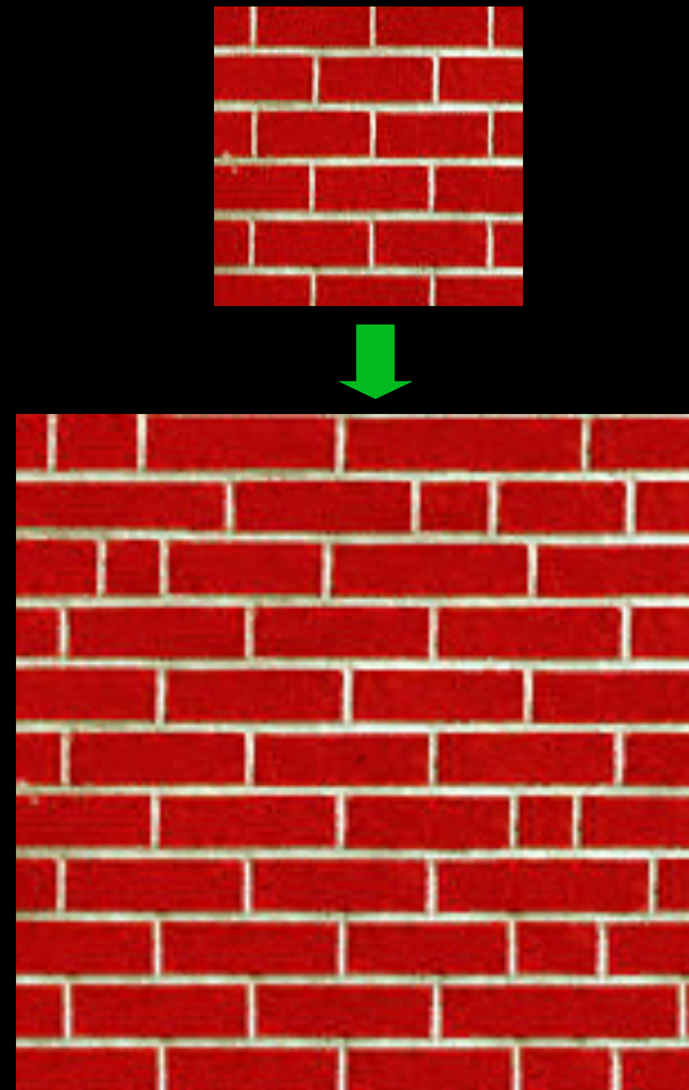


More Results

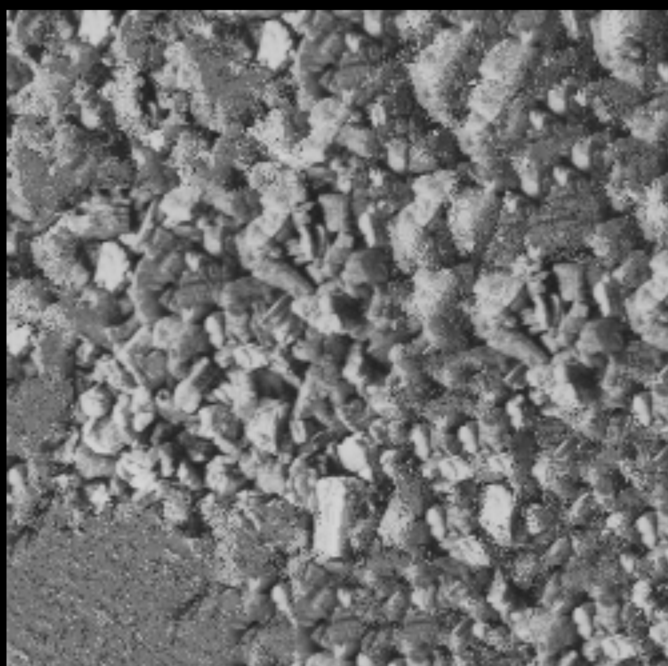
white bread



brick wall



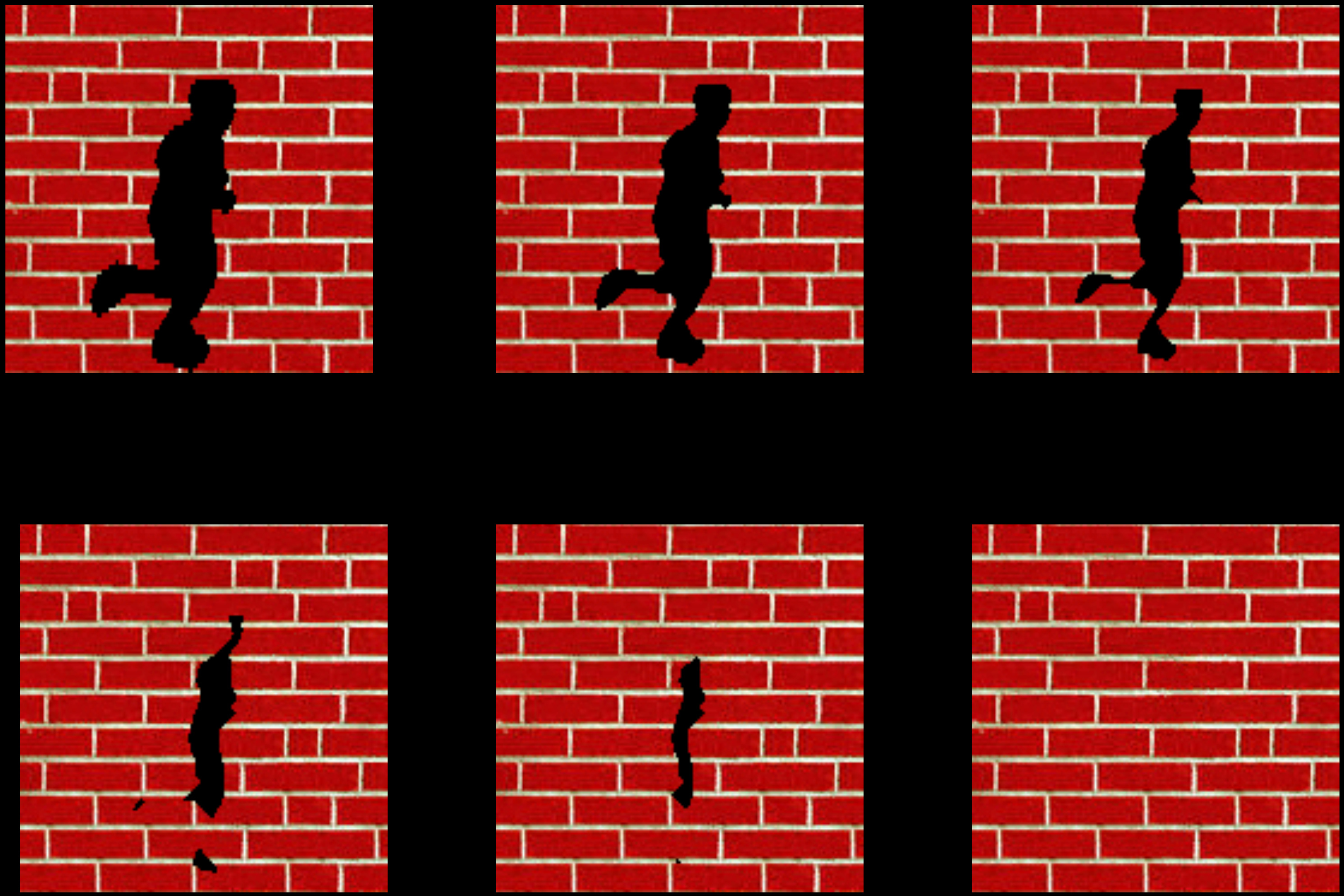
Failure Cases



Growing garbage

Verbatim copying

Hole Filling



Extrapolation

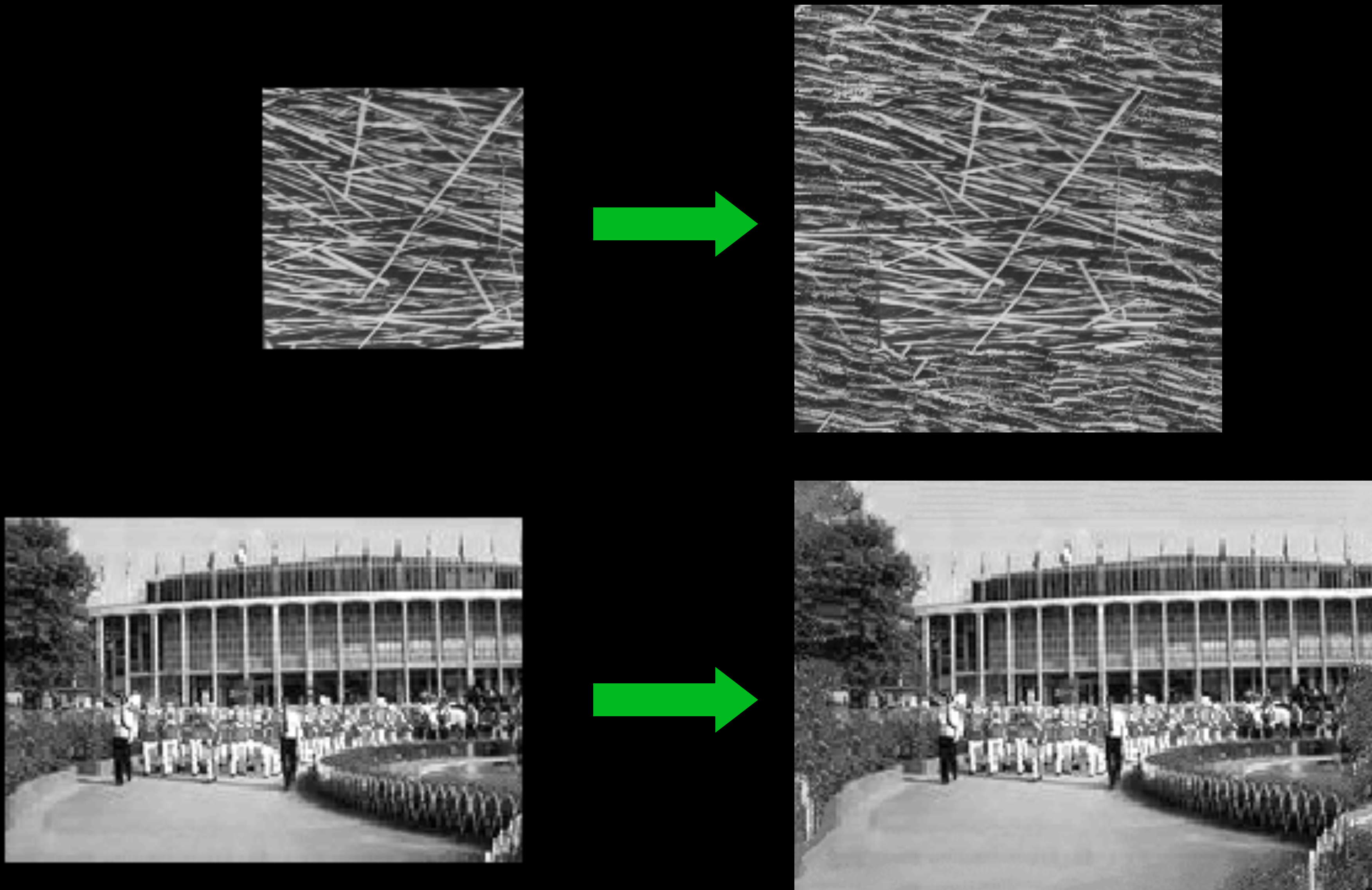
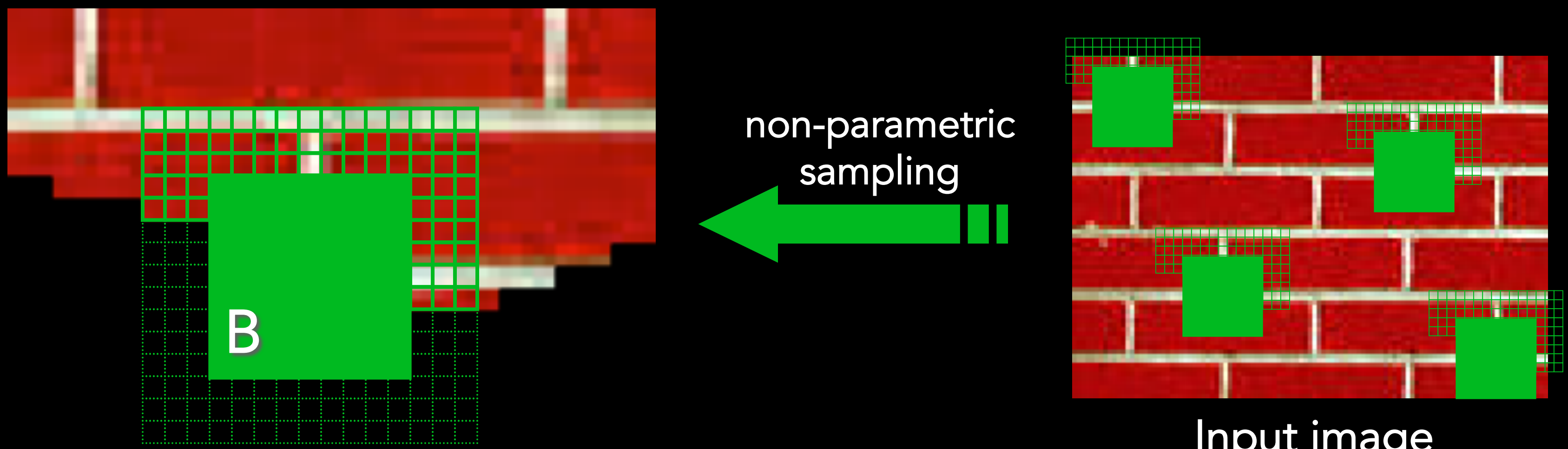


Image Quilting [Efros & Freeman]



Synthesizing a block

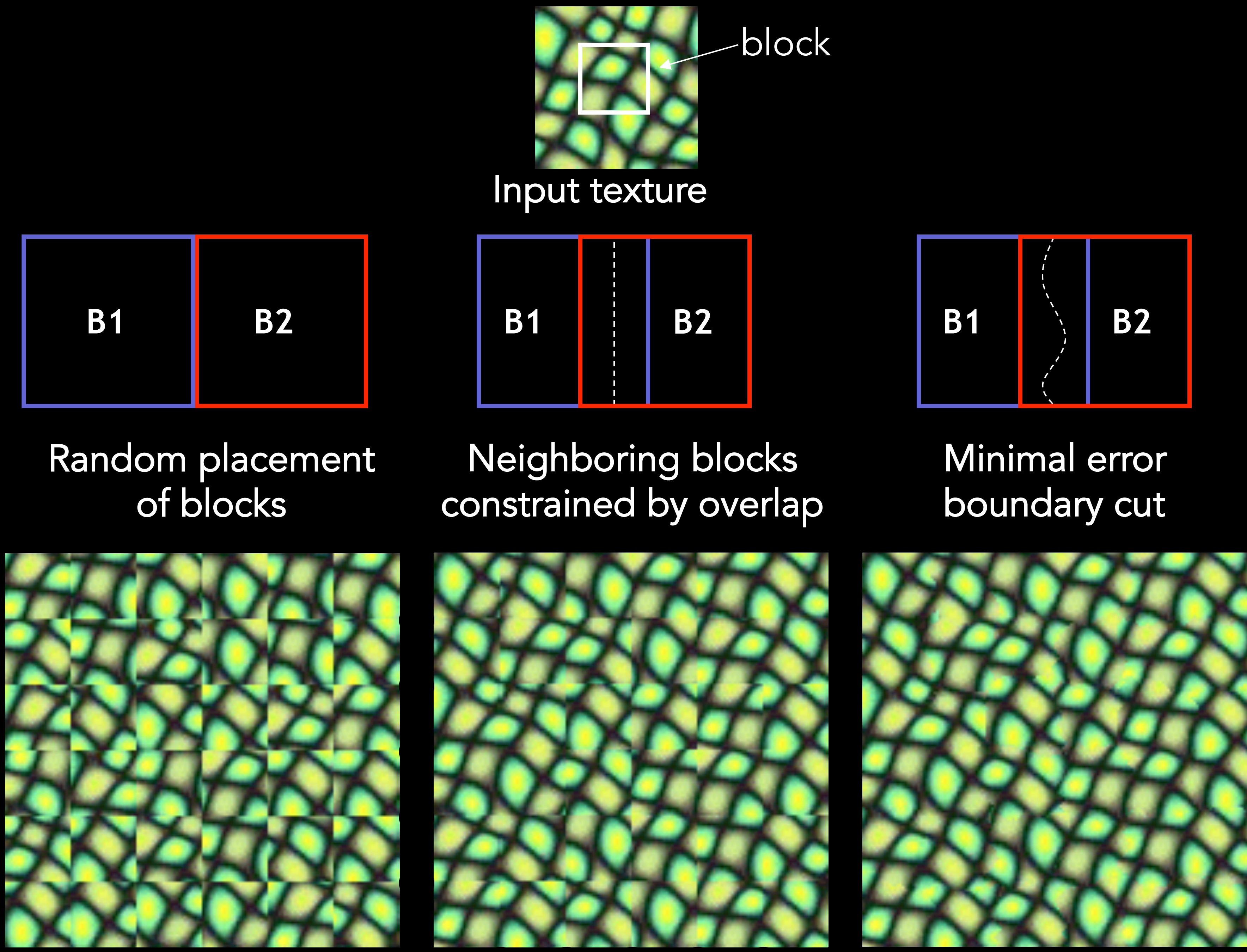
Input image

- Observation: neighbor pixels are highly correlated

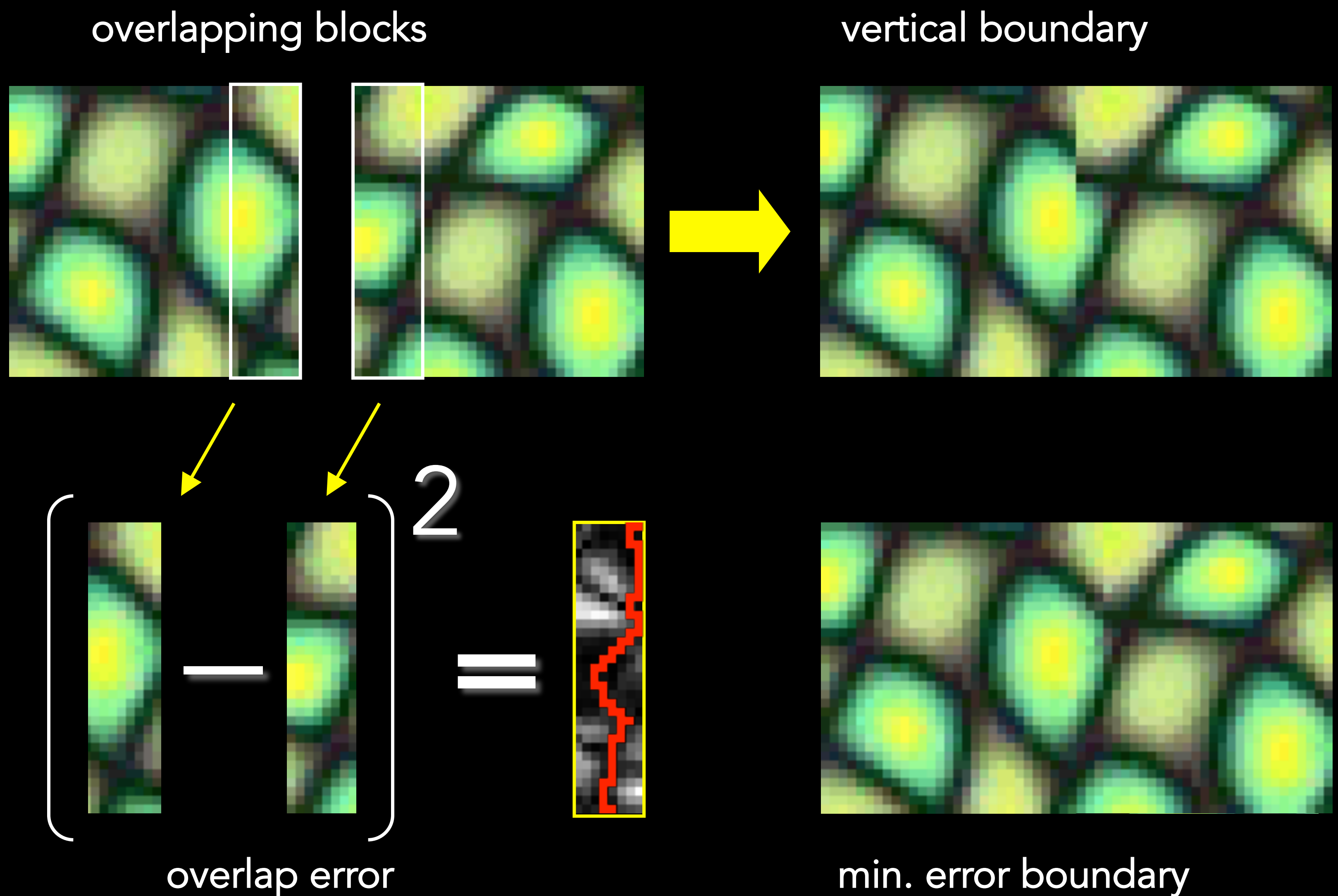
Idea: unit of synthesis = block

- Exactly the same but now we want P(**B**|N(**B**))
- Much faster: synthesize all pixels in a block at once
- Not the same as multi-scale!

http://graphics.cs.cmu.edu/people/efros/research/quilting.html


block
Input texture
B1
B2
B1
B2
B1
B2
Random placement of blocks
Neighboring blocks constrained by overlap
Minimal error boundary cut

Minimal error boundary



Texture Transfer

- Take the texture from one object and "paint" it onto another object
  - This requires separating texture and shape
  - That's HARD, but we can cheat
  - Assume we can capture shape by boundary and rough shading

- **Then, just add another constraint when sampling: similarity to underlying image at that spot**

+

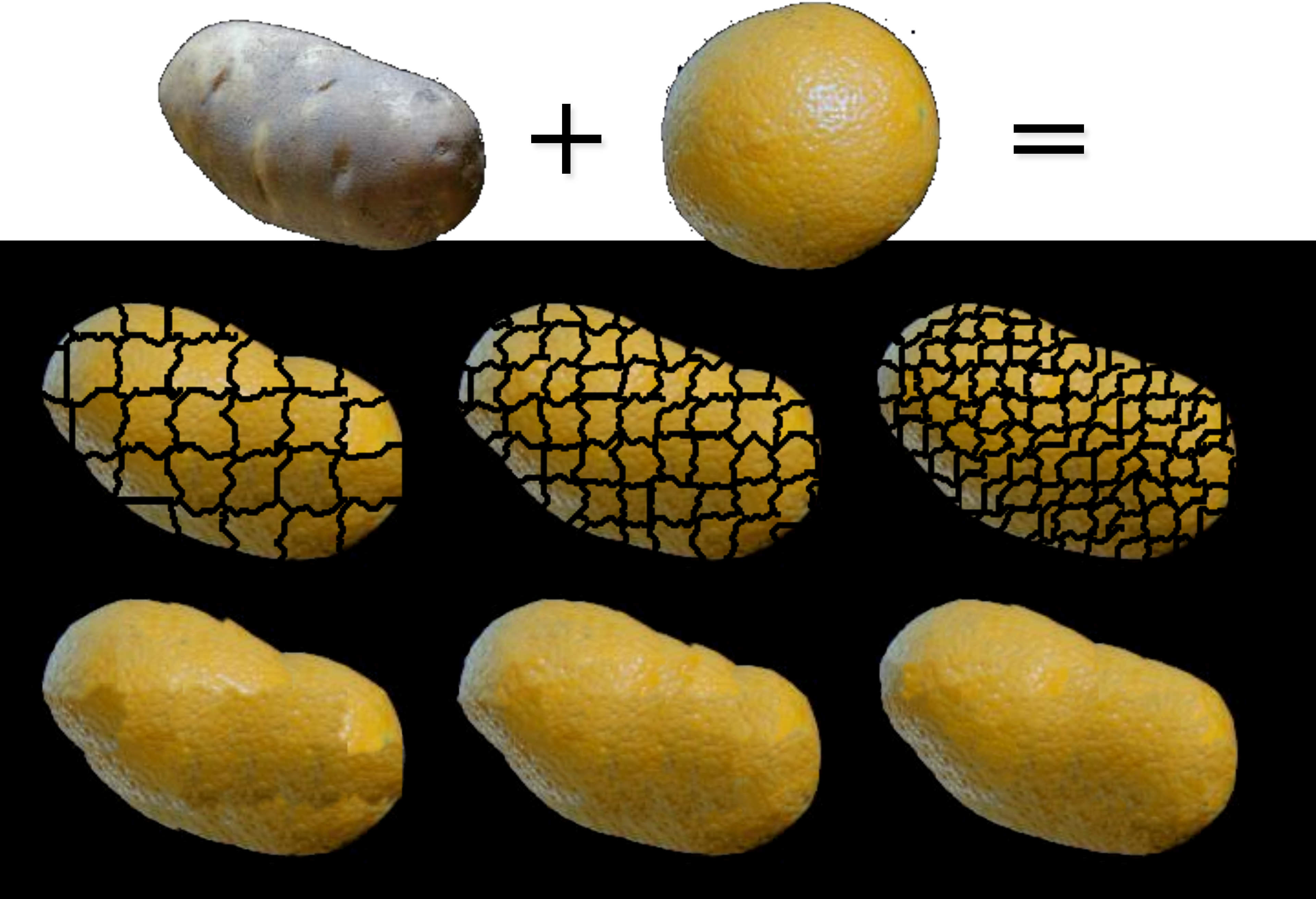
parmesan

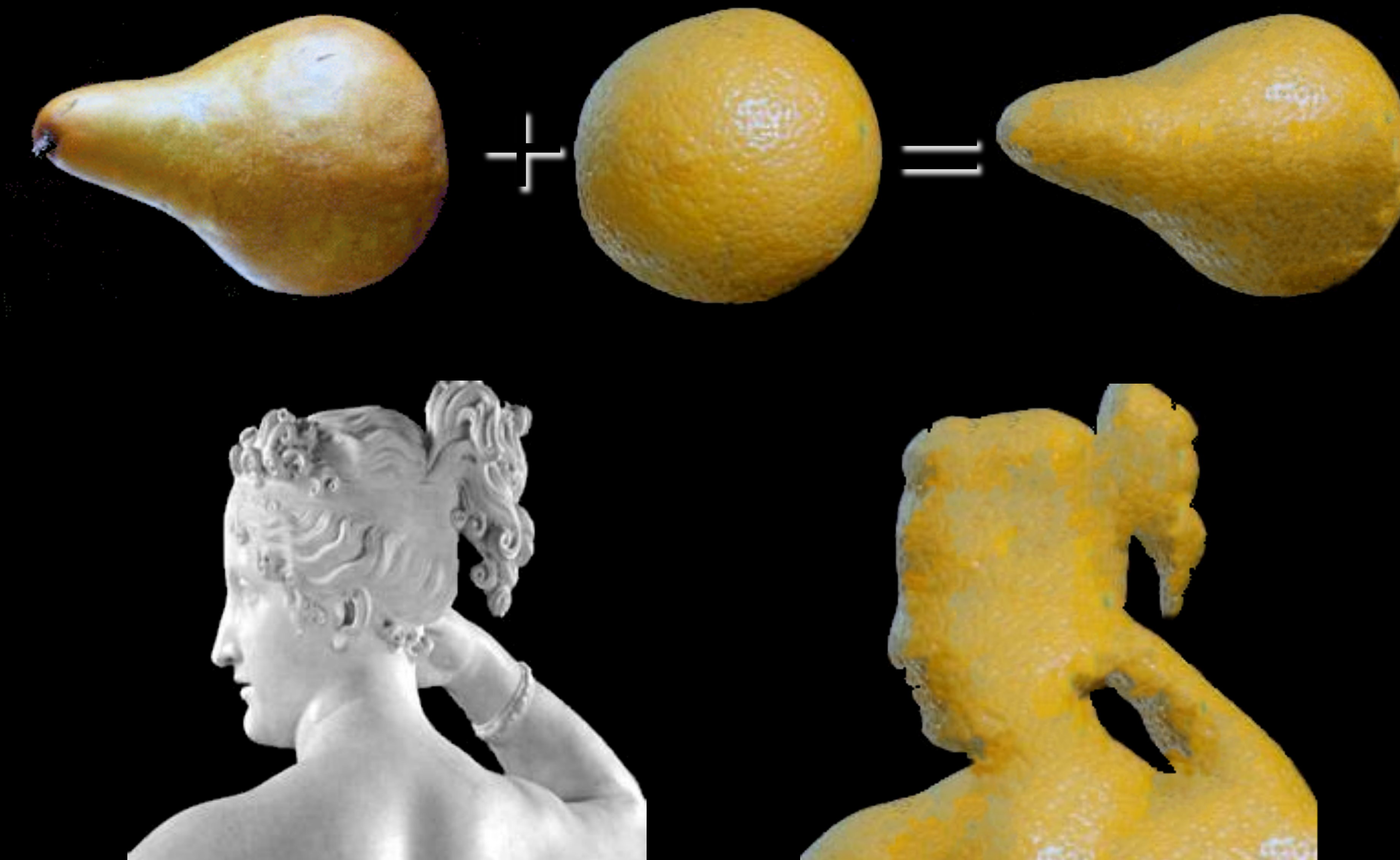
=

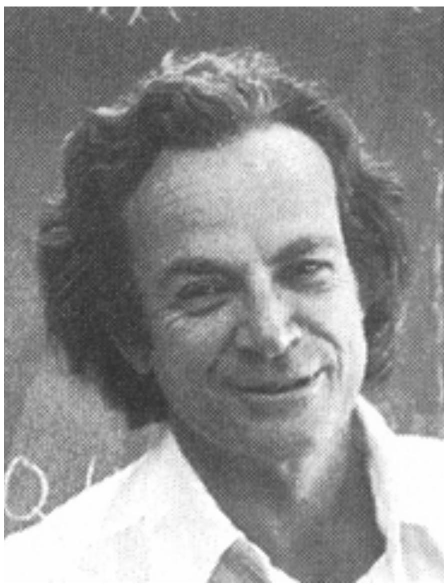
+

rice

=







+

=

Shape and Texture Synthesis

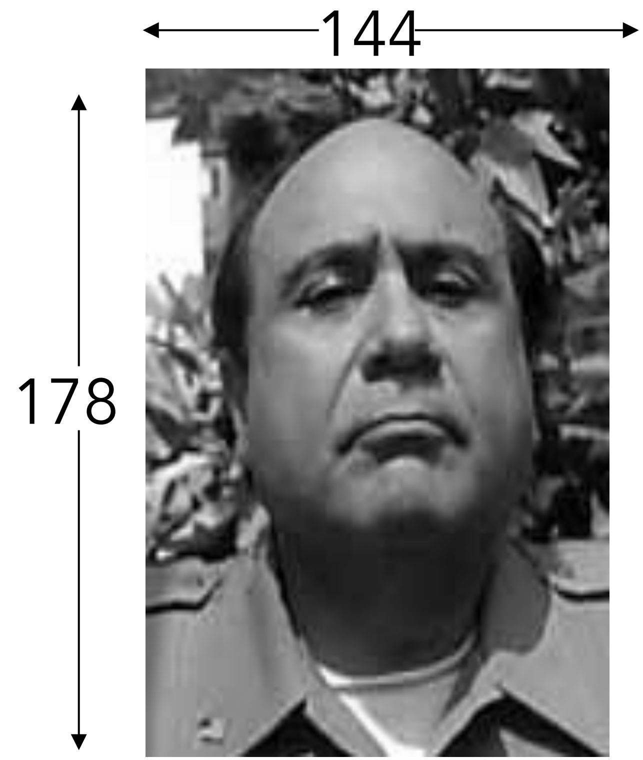
Goal of “Interpretation through synthesis”

The same idea than the texture synthesis approach:

- Represent a novel image by generating synthetic images that are as similar as possible to the target image
- Similarity is based on shape and texture (i.e. color): use of a collection of parameters that describe the image appearance (e.g. round shape, dark grey color, etc)

Pixels as Features

- A grayscale digital picture has n rows by m columns of pixels. Each pixel can have a single gray scale value (ex. 0-255 black to white).
- We can consider each pixel as a feature (or dimension) of that image.
- These features may be numerous but they are very cheap to generate.



25632 feature dimensions

Feature Extraction:
Principal Component Analysis PCA

- Use PCA to find a new set of features, from pixels, that better represents the data.

- Pick the best principal component vectors to represent the data.

What is PCA ? Ex. For Faces

- An image of a face is stored as the intensity of gray level of each pixel.

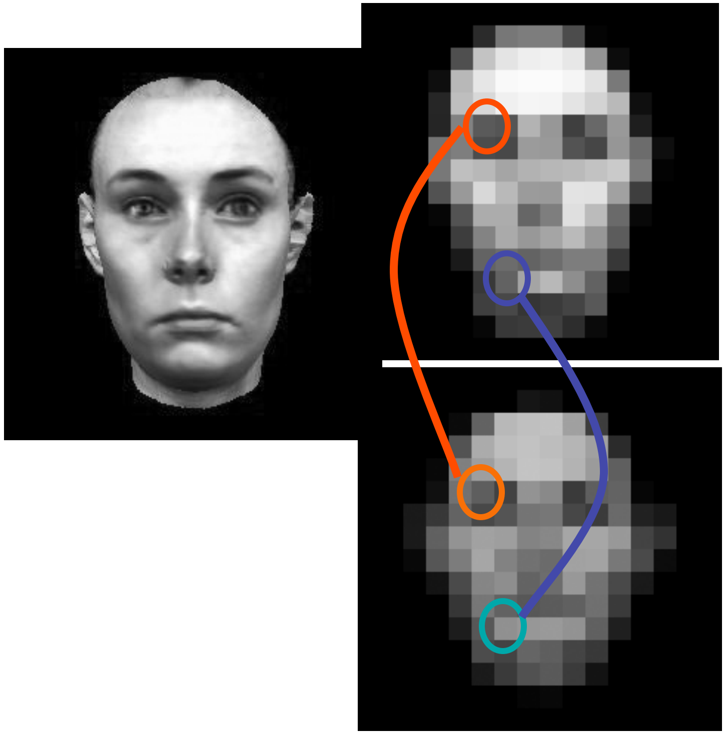
- What differences are important and what are not in a set of faces ? Can we reduce the dimension of the images (nb of pixels), while maintaining the "relevant" differences.

- One strategy: Principal components analysis

- By analyzing the statistical variation across different pixels in a large number of images, we can derived a more economical way to represent faces.

- Across a series of faces, there will be variation of the intensity shown in each pixel: by analyzing the pattern of correlation between the grey levels in all the different pixels across a series of faces, <u>the principal components of this variation</u> can be extracted.

- E.g. some men have receding hairlines, so the pixels at the upper forehead will be light (skin) while other have a full head of dark hair and the corresponding pixels may be dark.



Faces PCA example

- If a set of eigenfaces is derived from a set of face images, any face can be described as an **appropriate weighted sum of this set of eigenfaces** for analysis

- Eigenface representation is an economic method of coding large number of faces: what is stored is 1) the eigenfaces images and 2) the <u>weights</u> for each individual face.

- Eigenfaces method works only if faces are aligned. A possible method is 1) morph the faces to a common shape first, and 2) apply PCA. Then, analyses can be conducted both of the grey levels in the "shape-free" (morphed) images and on the shape vectors (the transformations needed to restore the original shape to the face).

Fig. 2. The first four 'shape free' eigenfaces.

Those represent the first 4 eigenfaces after all the 174 male faces were morphed to a common shape. There is no more variation around the bottom of the face. In this example, all 4 eigenfaces code aspects of hairstyle

[Hancock et al. al.98, Vis.Res,38,22]

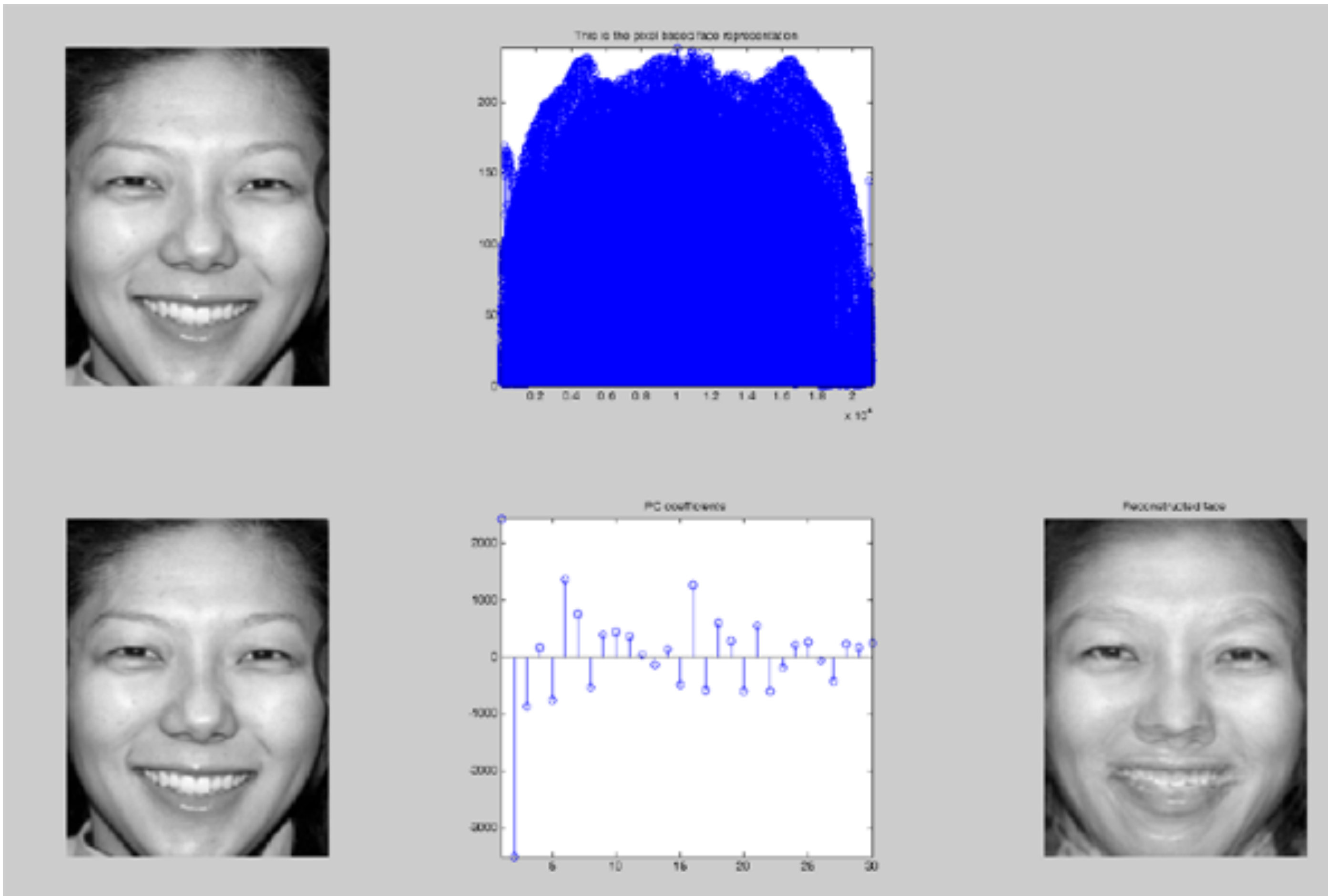
PCA Demo: Run pcaFaces.m

PCA-Faces/pcaFaces.m

Principal Components (eigenfaces) of *Emotion* dataset

Run section 4 of pcaFaces.m

Representation in a low dimensional space



Run section 5 of pcaFace.m

Reconstruction with different # of PC

Active Appearance Model

An **Active Appearance Model** (AAM) is a computer vision algorithm for matching a statistical model of object shape and appearance (texture) to a new image.

They are built during a training phase. A set of images together with coordinates of landmarks, that appear in all of the images is provided by the training supervisor.

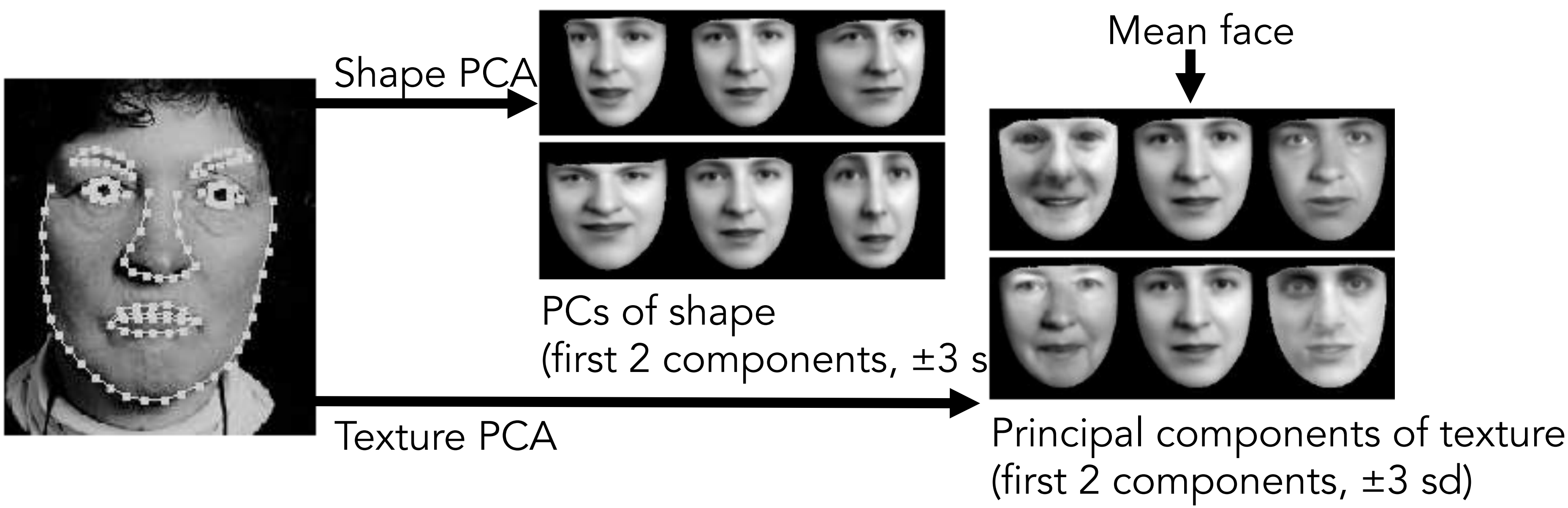
A statistical model of object appearance can be matched to an image in two steps

- (1) represent the shape of the object
- (2) represent the texture of the object

T. F. Cootes, G. J. Edwards, and C. J. Taylor. Active appearance models. *IEEE TPAMI, 23(6):681–685, 2001*

Active Appearance Models

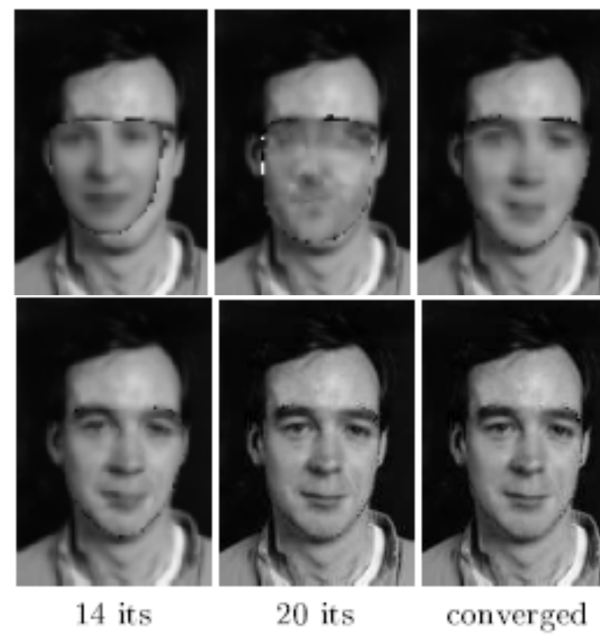
- Take a set of similar images
- Label corresponding landmark points in each image
- Warp images onto the mean shape to get shape-free texture
- Do PCA separately on shapes and textures . . .



Active Appearance Models

- Do more PCA on combined shape+texture coefficients
- Results:
  - Learn interesting things about the distribution of shapes/ textures in the object class and how they co-vary
  - Find landmark points in novel images

Principal components of combined shape+texture (first 4 components, ±3 sd)



Analysis by synthesis

AAM/readme.txt

Ingredients:

1) A large database of annotated objects.

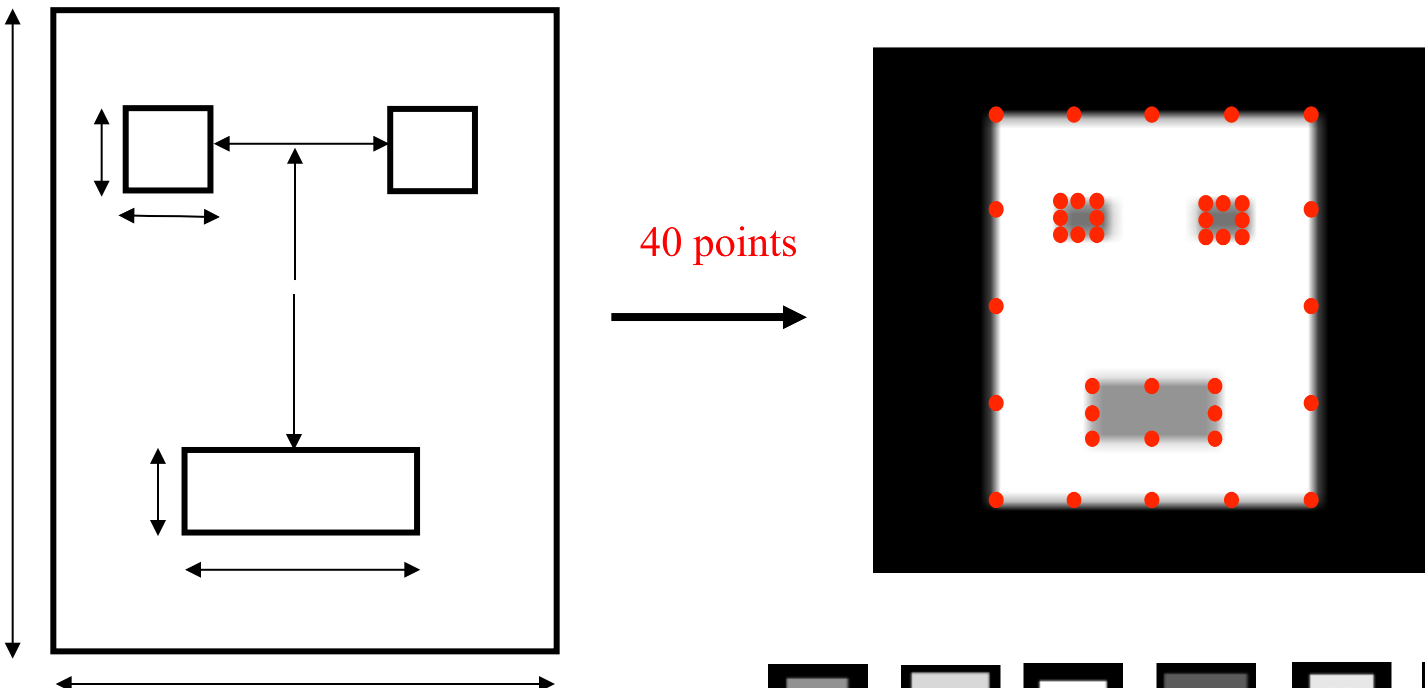
2) Synthesis method for generation of photo-realistic images from model parameters.

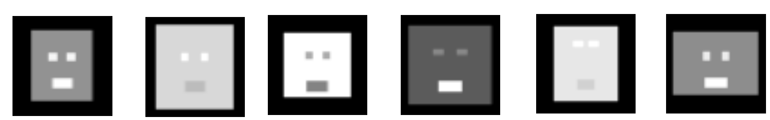
3) Analysis: extraction of model parameters from images.

Goal: Allow a prototype to vary according to some physical model

I- Robot training database

Labeling the training data set is step 1





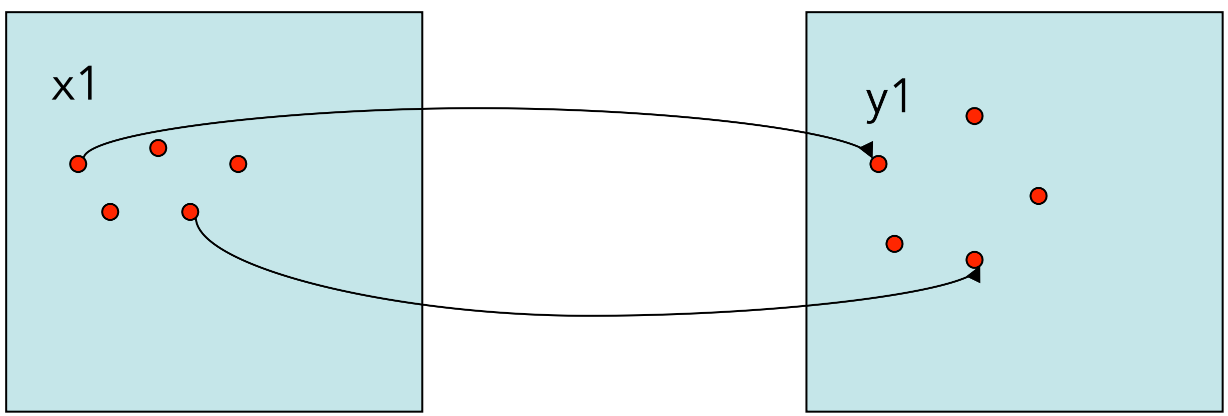
RoboFaces's parameters

AAM/labeling.m and demowarp.m

II- Image Warping

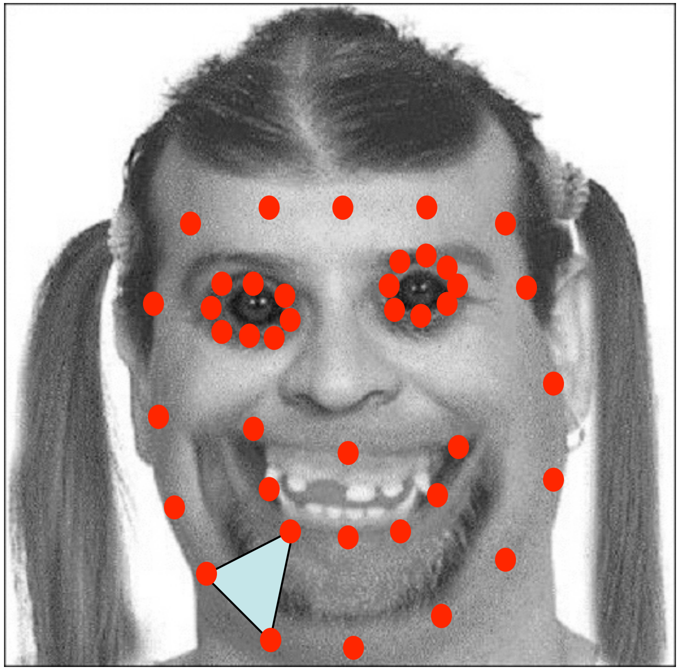
- Synthesis method for generation of photo-realistic images from model parameters

- The main building block of AAM is the image warping procedure.
- It is a function that applies a deformation to an image given a set of corresponding points:



AAM/labeling.m and demowarp.m

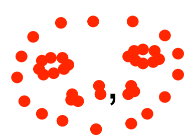
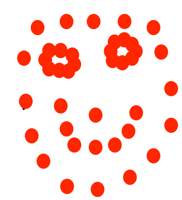
AAM/demowarp.m



Original image

Background

The Matlab implementation is limited to convex objects but this is good enough for faces.

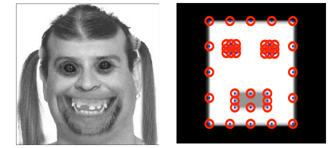
 = ImageWarp (   

background

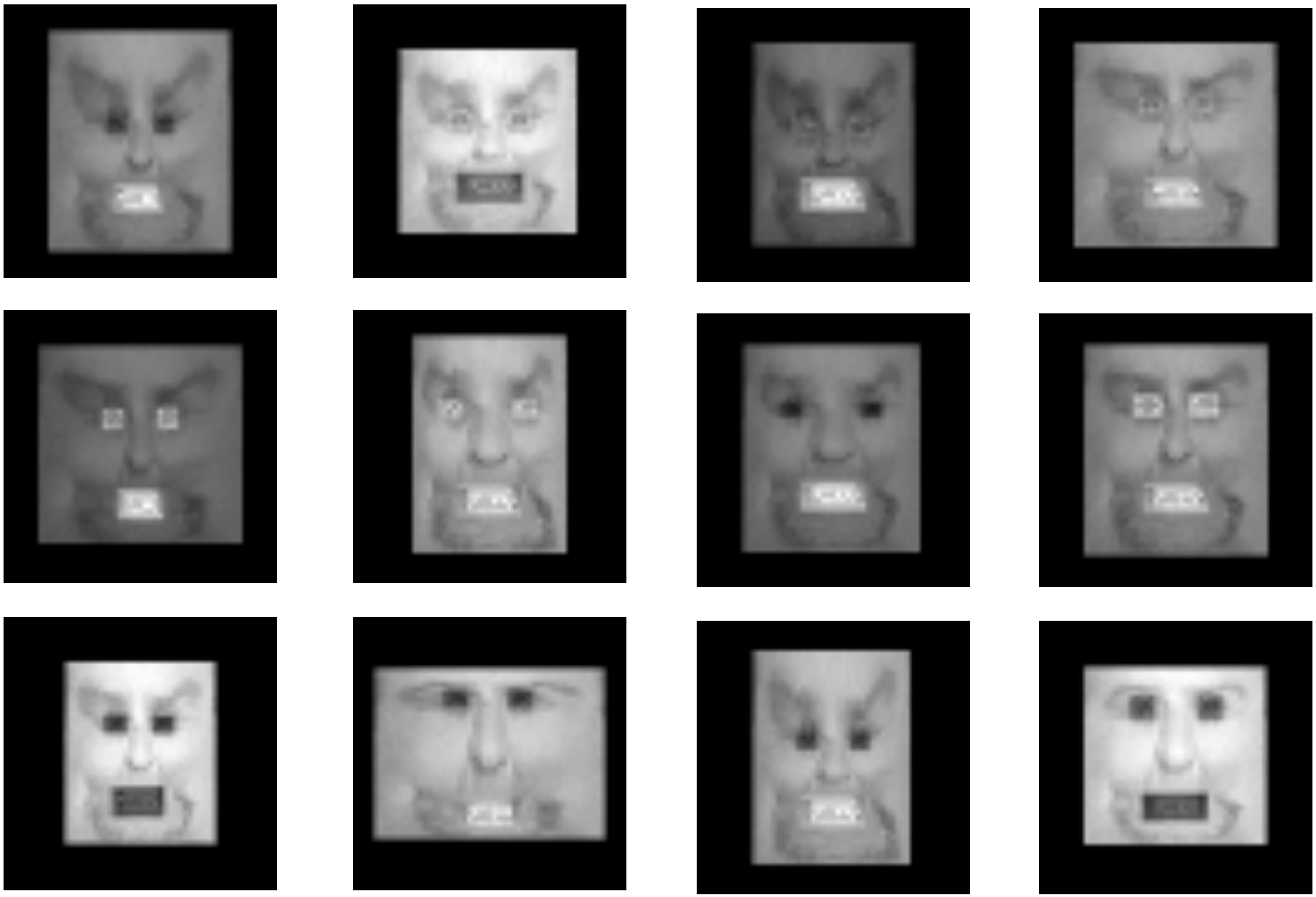
This function is used during the iterations of the AAM.

AAM/VirtualExamples.m

AAM/labeling.m

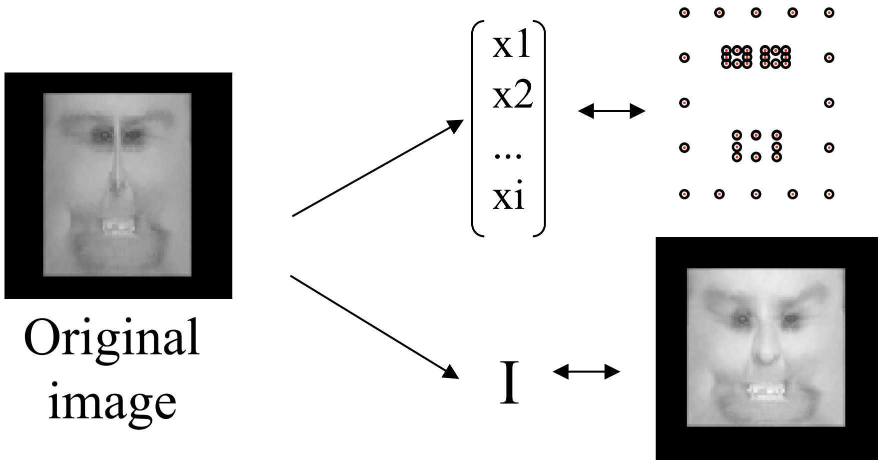


We warp a “real” face into the roboFaces in order to have more realistic images. We have same modes of variation.



III- Appearance model

- Each image is represented as (1) a collection of correspondence points (shape) and (2) a texture image normalized in shape.

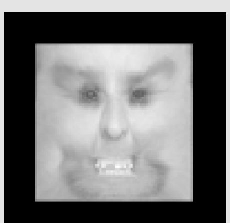


1 - Shape information (texture free)

2 - Texture information (shape free)

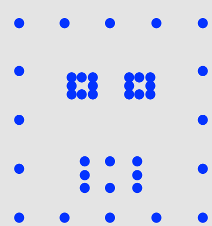
- 2 - Shape normalization is obtained by warping the image into the mean shape of the training database.

Shape free texture



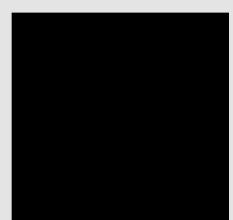
= ImageWarp (

Mean shape



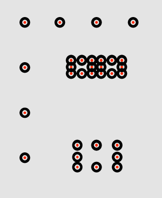
,

zeros



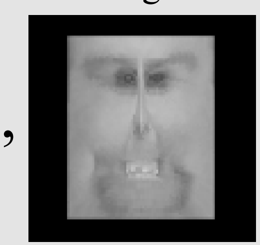
,

shape



,

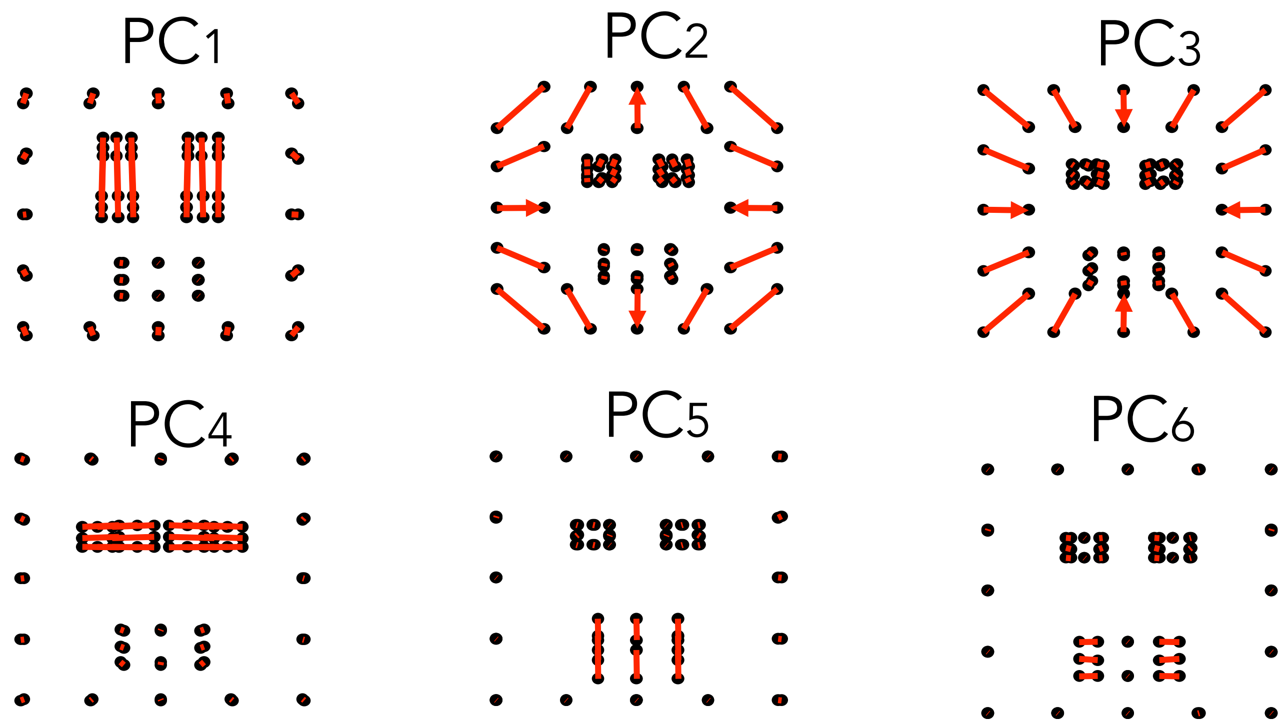
Original image



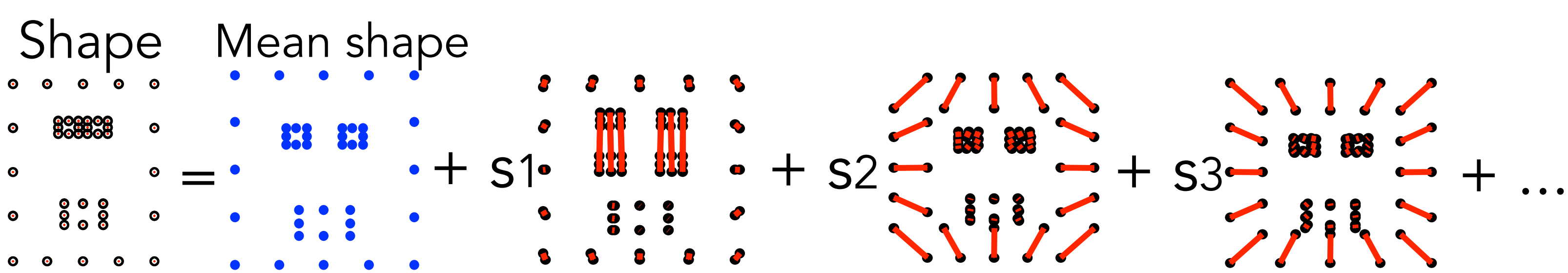
)

1 - Shape model

- PCA of shape information for the training database:



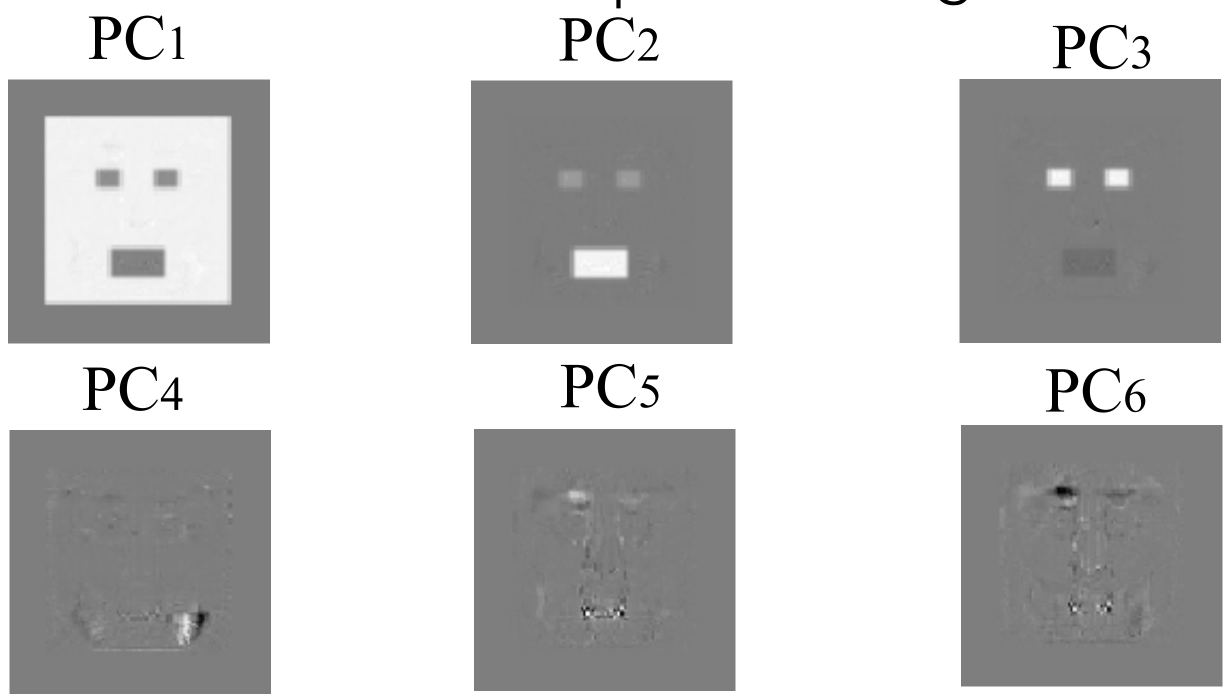
- Each shape can be decomposed as:



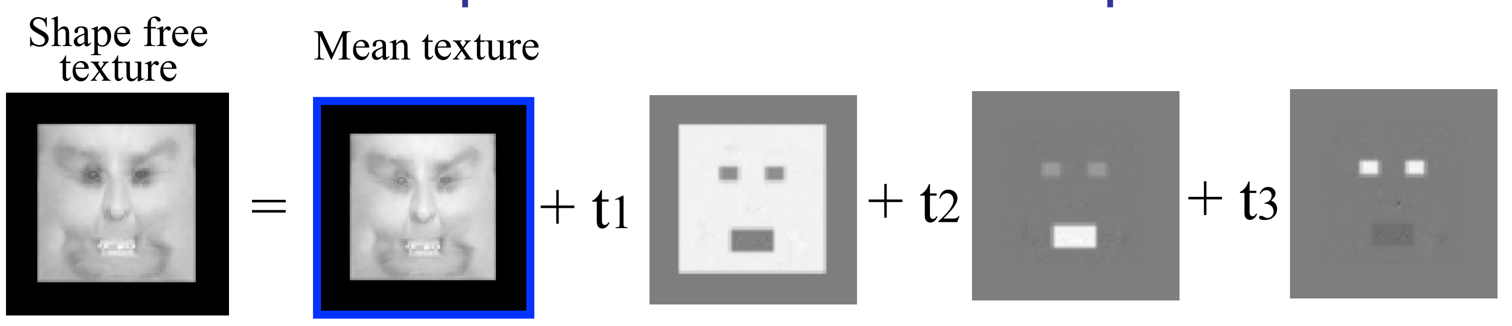
2 - Texture model

- PCA of texture information for the training database:

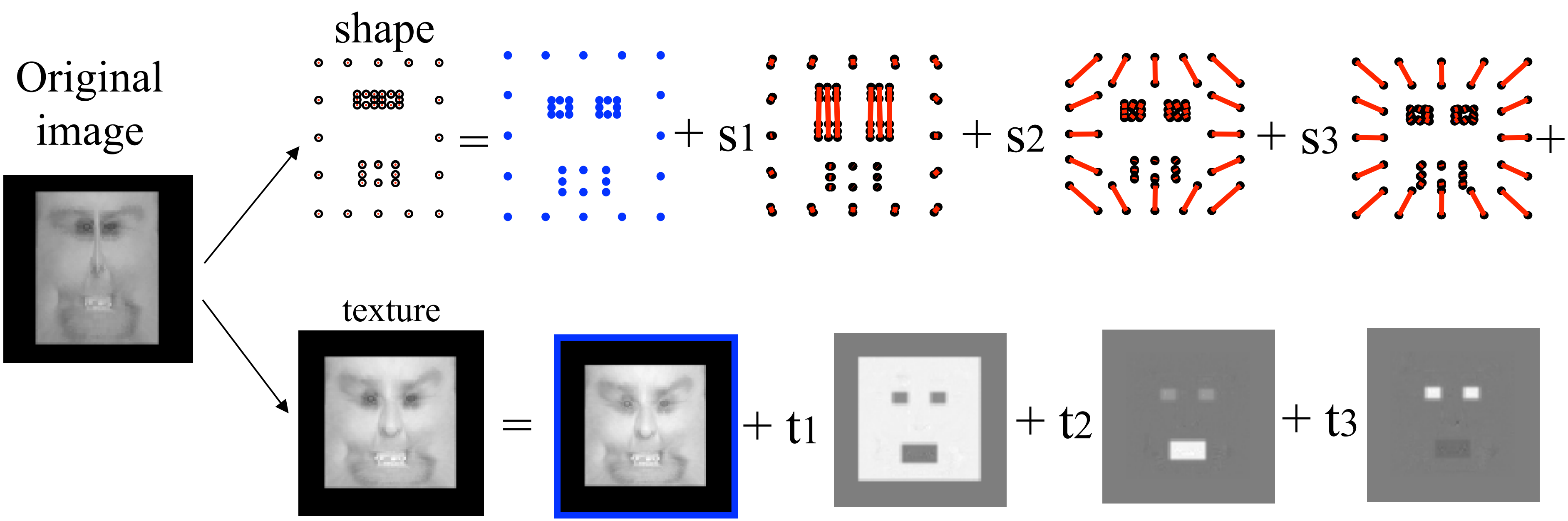
The PCA is done on the shape free images



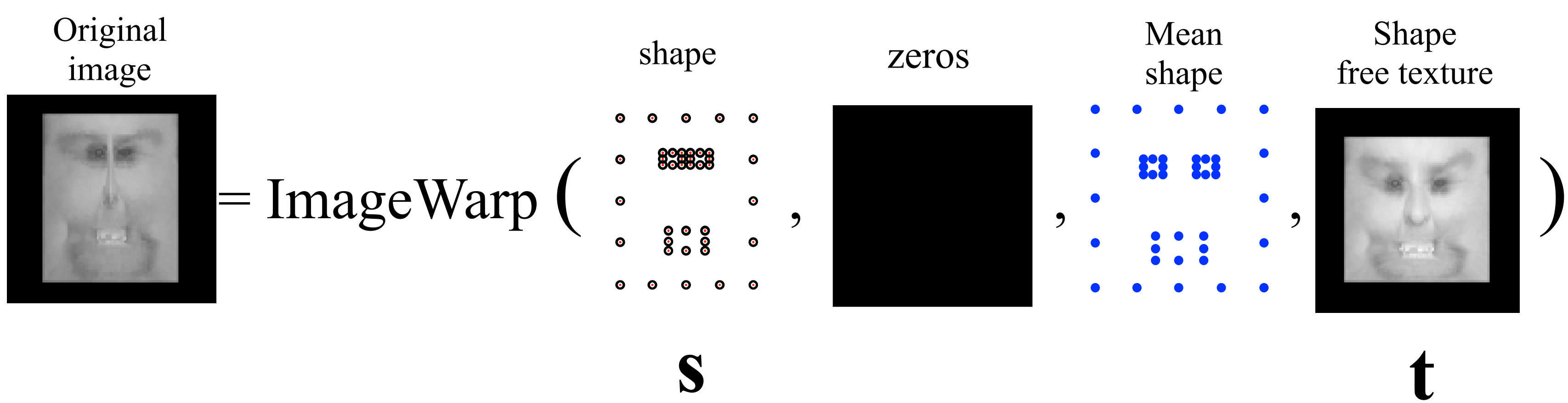
- Each texture (shape free) can be decomposed as:



Summary of Appearance Model of one image

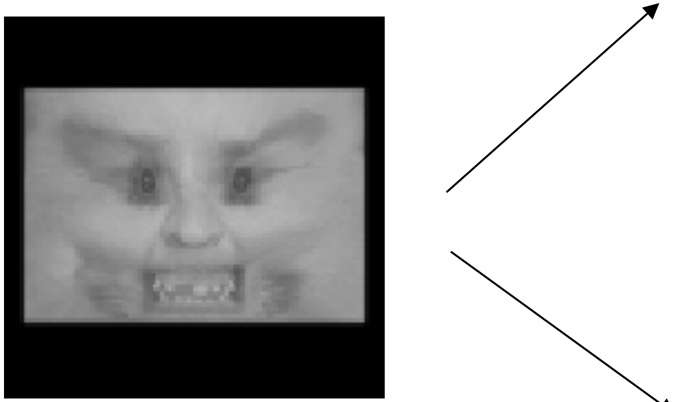


A set of model parameters encode shape and gray level variation learned from a training set



Active Appearance Model Search

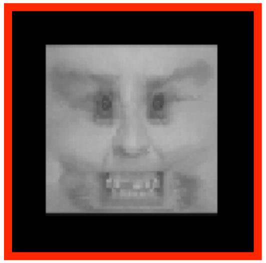
Given a "face" the model has to build an appearance model (shape + texture) that reproduces the original image.



Shape = ?

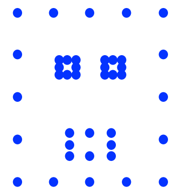
Texture = ?

This is done in an iterative procedure that tries to minimize the reconstruction error.



= ImageWarp (

Model (mean shape)

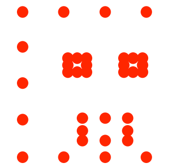


,

zeros

,

estimated shape



$\mathbf{s}_i$

,

Input Image (in shape free template)

)

This template is the goal of the AAM Algorithm: retrieve the Red points

The algorithm adjusts the points of the shape and texture templates and propose a new <u>Model template</u>

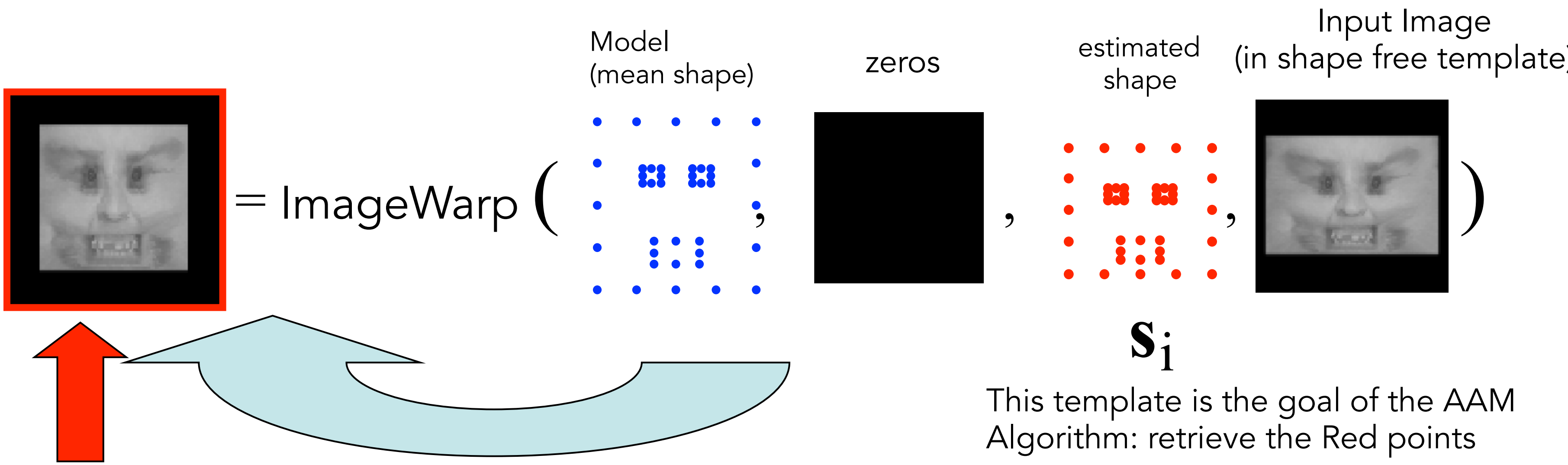
Error

AAM/learnerror.m

Two parts of the iterative procedure

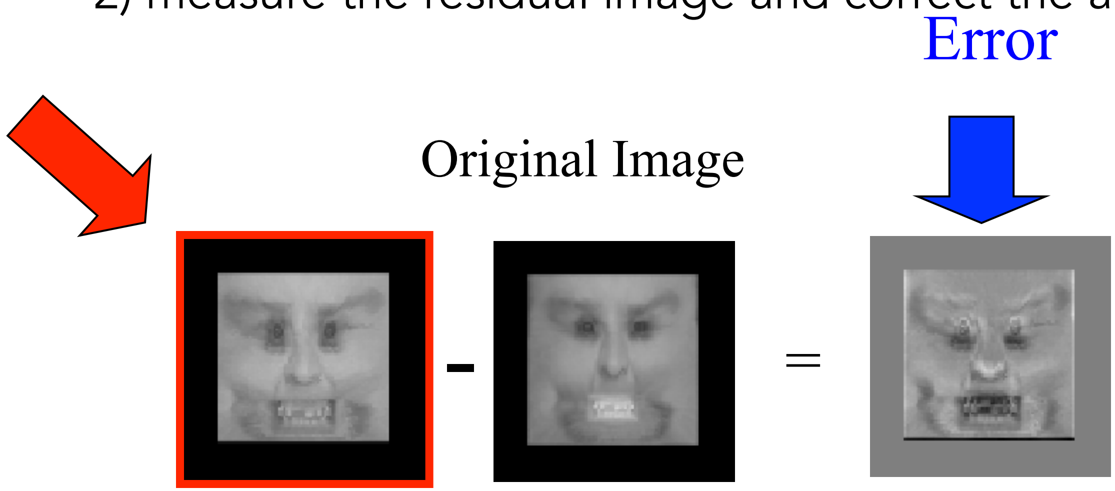
AAM/learnerror.m

1) given a set of shape parameters, warp input image into its shape free template



Result of
the warping

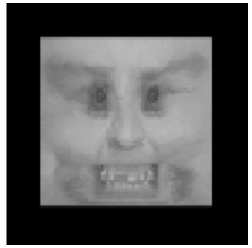
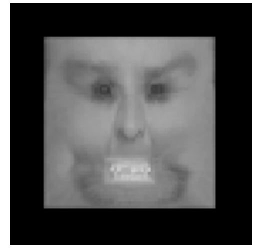
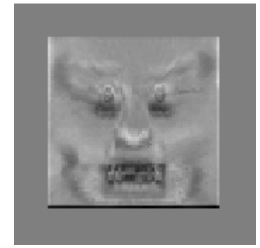
2) measure the residual image and correct the appearance model.



The residual is function of errors in both shape and texture parameters

Learning to correct model parameters

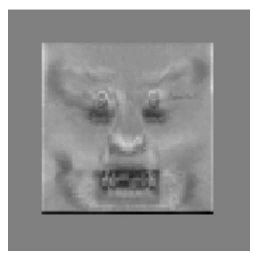
$$\begin{bmatrix}\Delta s\\ \Delta t\end{bmatrix} = F\Bigg($$

 -  =  $\Bigg)$

t_i

Linear approximation:

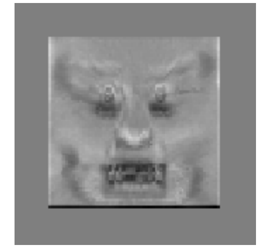
$$\begin{bmatrix}\Delta s\\ \Delta t\end{bmatrix} = \mathbf{A}$$



Column vector

Matrix **A** is learned by adding perturbations to the parameters of the training set. The residual corresponds to the difference between the image obtained with the real parameters and the one perturbed.

Learning to correct shape parameters

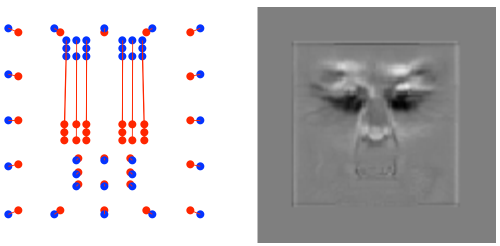
Shape parameters: $\Delta\mathbf{s} = \mathbf{A_s}$ 

vector

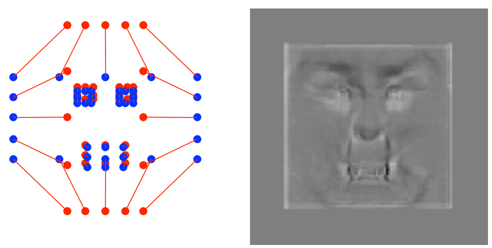
As is Rs in matlab program..

Each row of **As** describes how the residual contributes to each shape mode:

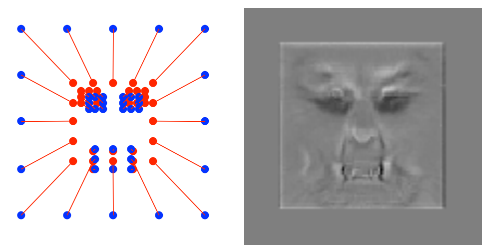
1st row of **As**



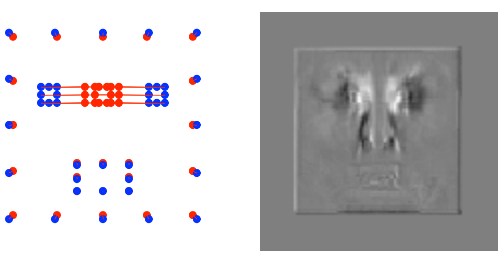
2nd row



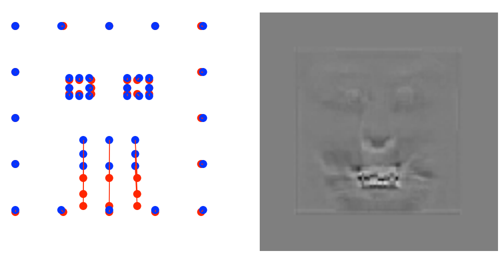
3rd row



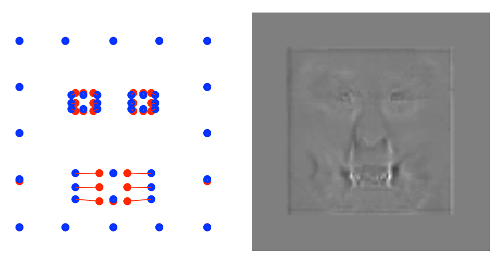
4th row



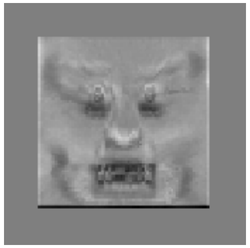
5th row



6th row



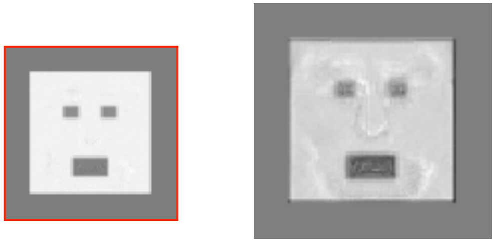
Learning to correct texture parameters

Texture parameters: $\Delta \mathbf{t} = \mathbf{A_t}$ 

vector

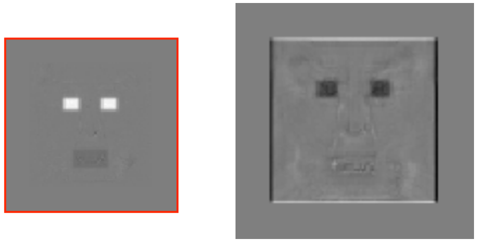
Each row of **At** describes how the residual contributes to each texture mode:

1st row of **At**



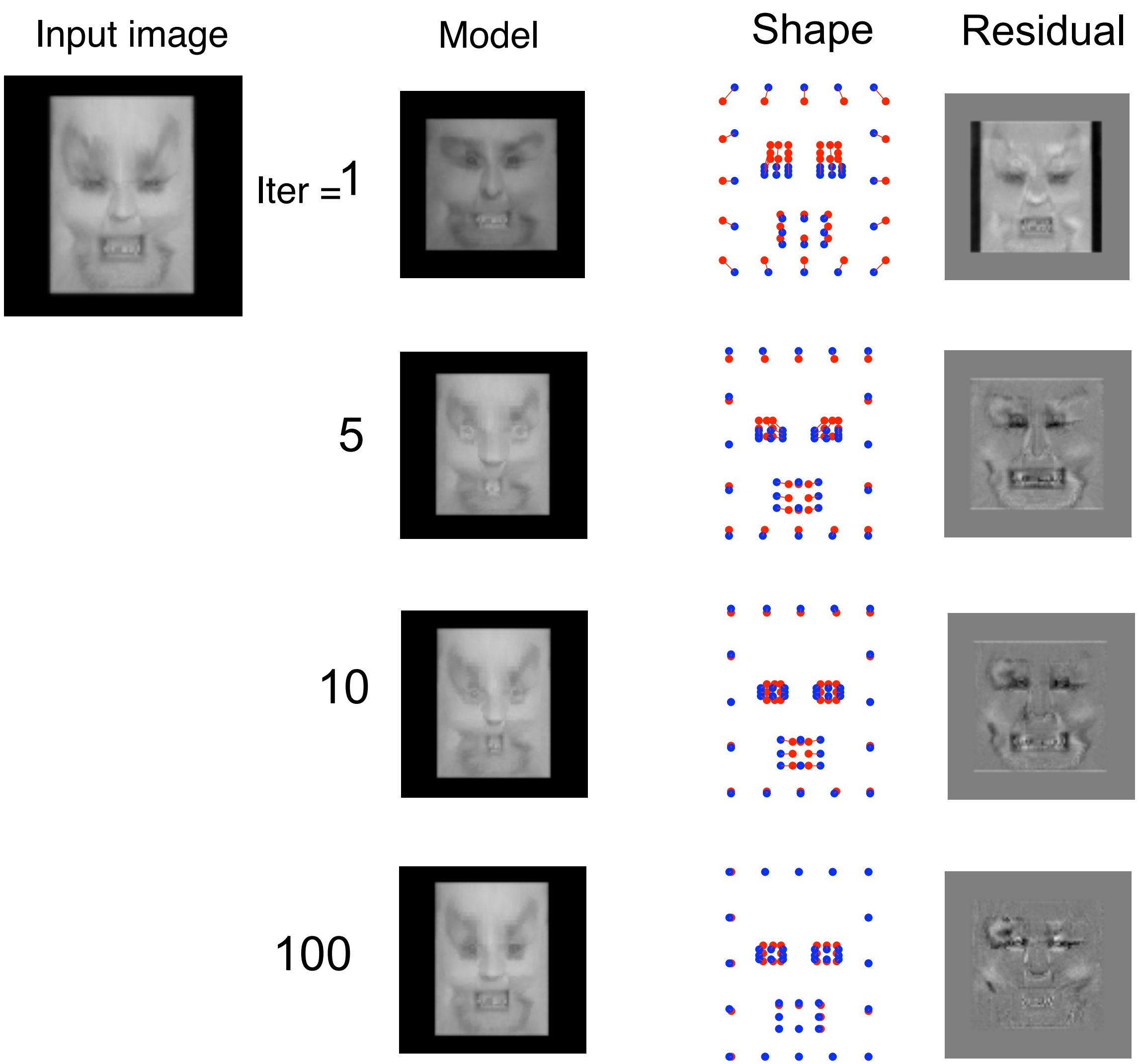
2nd row

3rd row

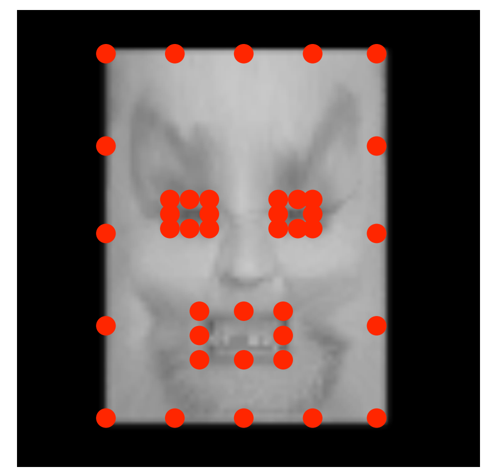


Results

AAM/detection.m

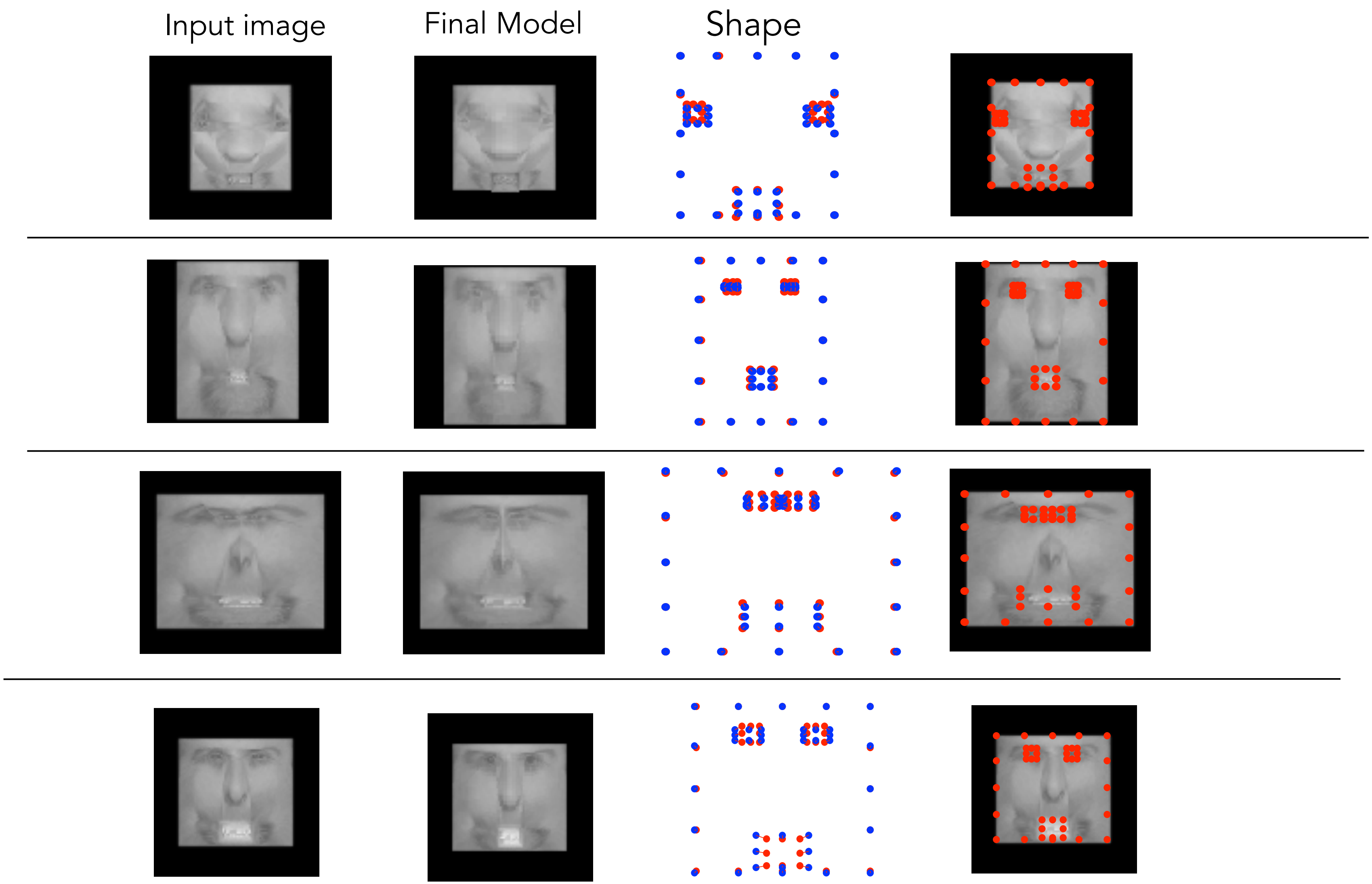


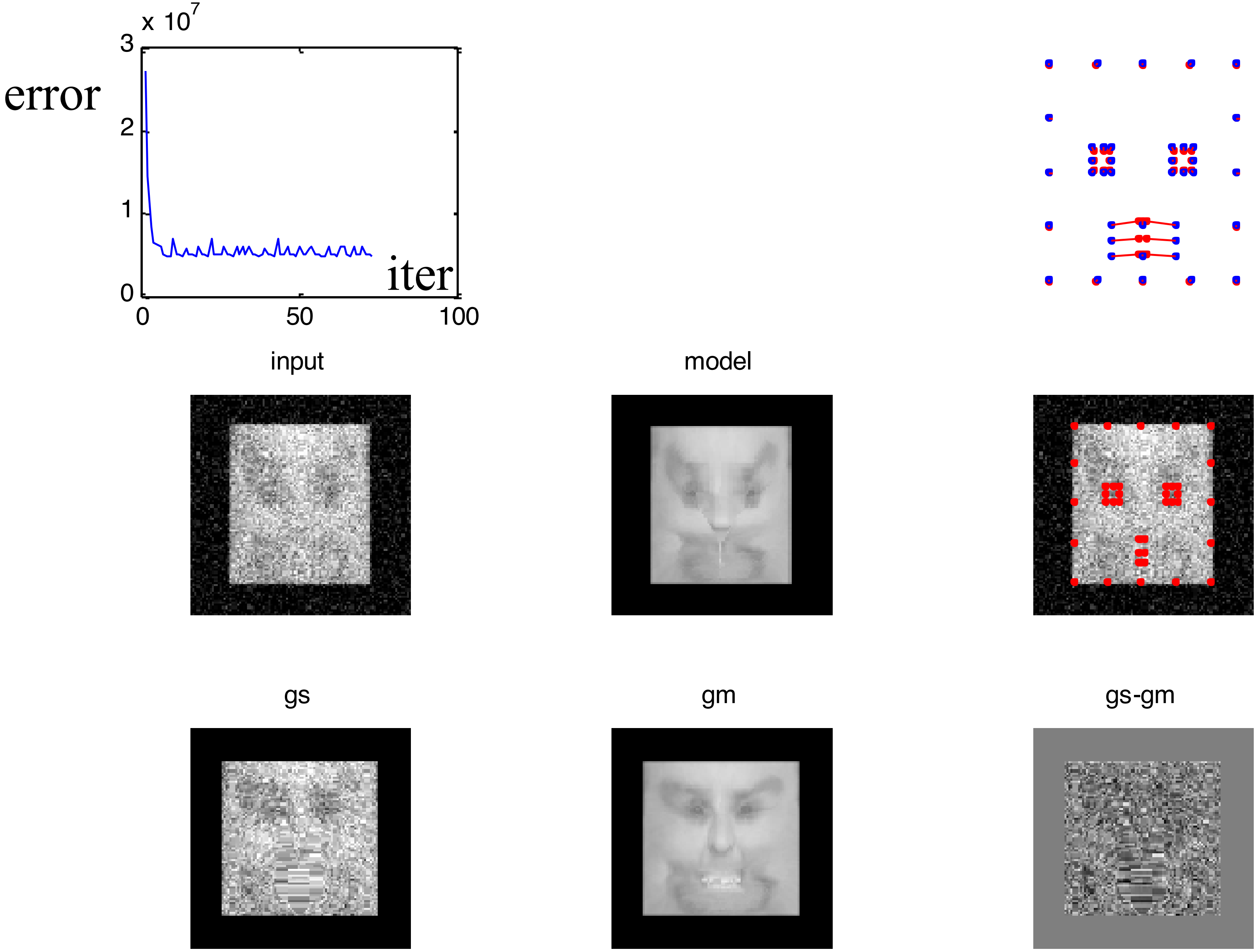
Convergence after 50 iterations



Results

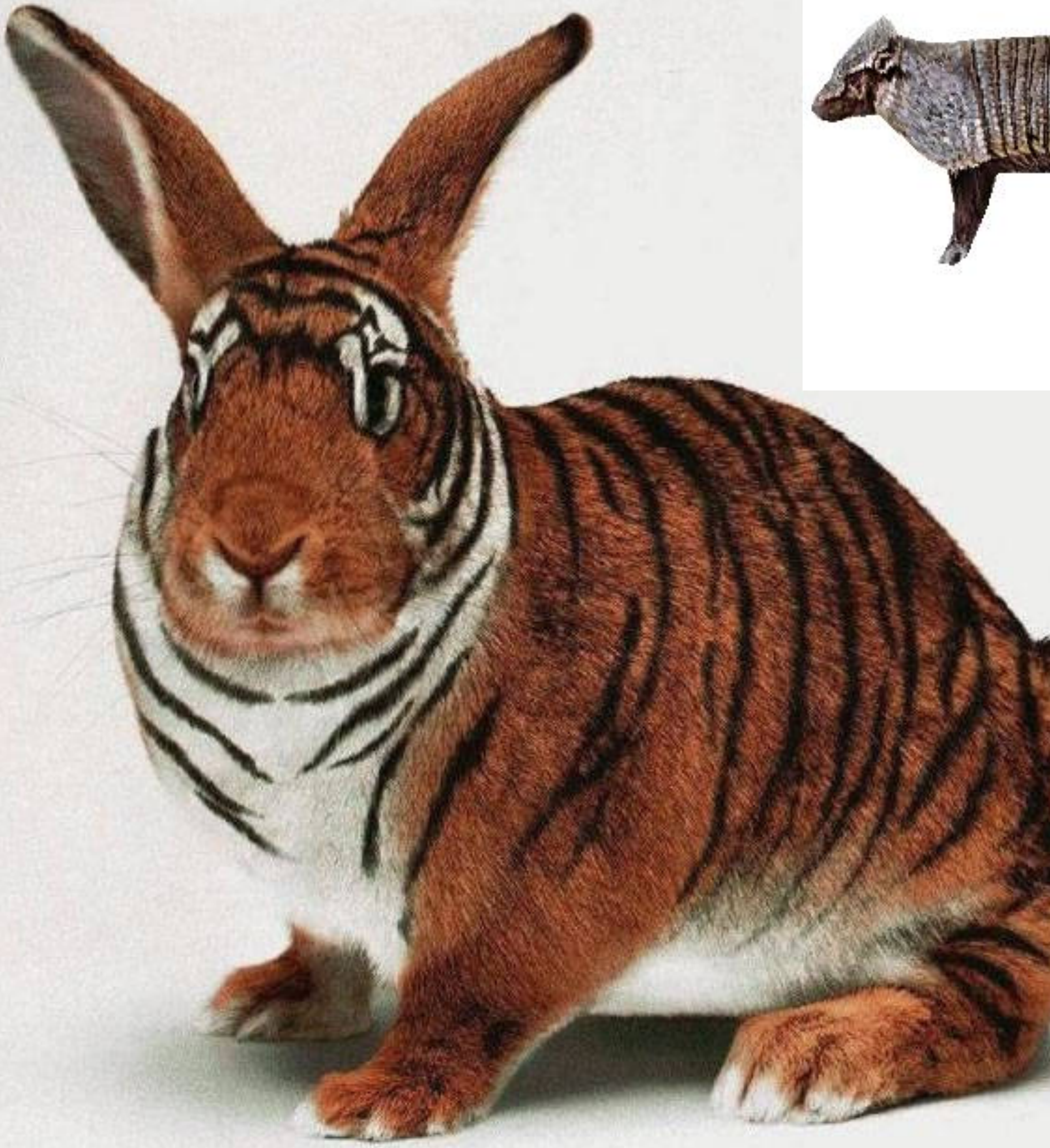
Even when the images have real parameters that deviate from the distribution of the training set, the algorithm seems to converge:





Adding priors to possible appearance parameters may prevent this.






Shape-free "animals"

- Obtained by warping each animal's shape onto the mean shape