Texture synthesis Algorithms: Fourier transform, patches, wavelet transform, and deep neural networks

Bruno Galerne
bruno.galerne@univ-orleans.fr

Institut Denis Poisson, **Université d'Orléans**, Université de Tours, CNRS

Module Image de l'École Doctorale I2S
Cours "Algorithmes de synthèse de textures"
Mercredi 17 avril 2019

Outline

Outline

What is a texture?

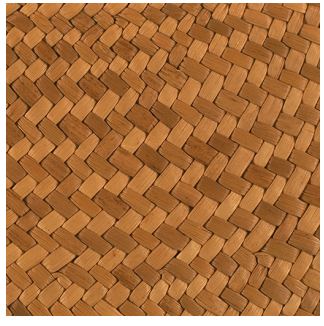
A minimal definition of a **texture** image is an “image containing repeated patterns” [Wei et al., 2009].
The family of patterns reflects a certain amount of randomness, depending on the nature of the texture.
Two main subclasses:

- The ***micro-textures***.



- The ***macro-textures***, constitued of small but discernible objects.



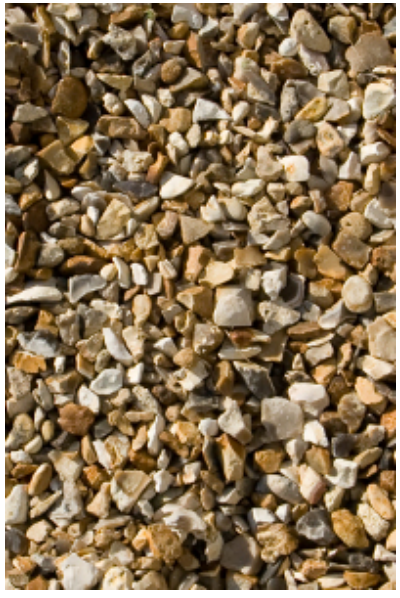


Textures and scale of observation

Depending on the **viewing distance**, the same objects can be perceived either as

- a micro-texture,
- a macro-texture,
- a collection of individual objects.

Micro-texture

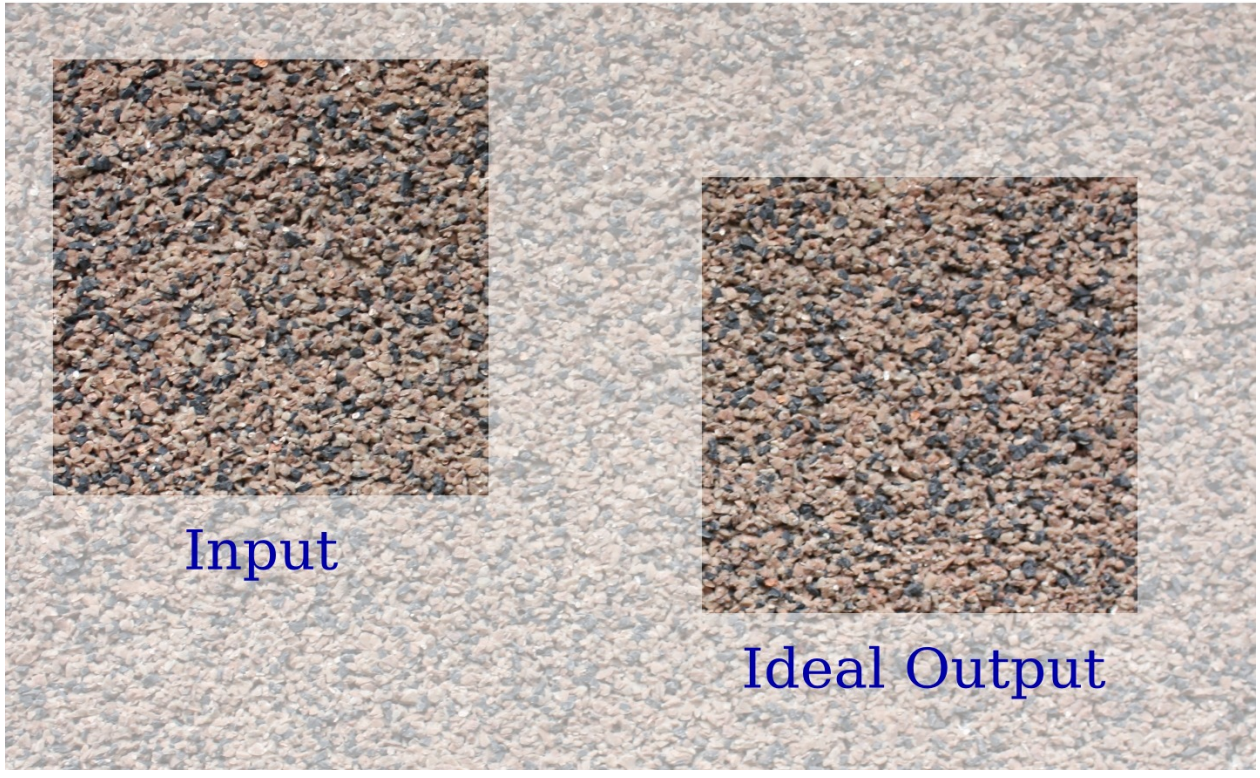


Macro-texture

Some pebbles

Texture synthesis

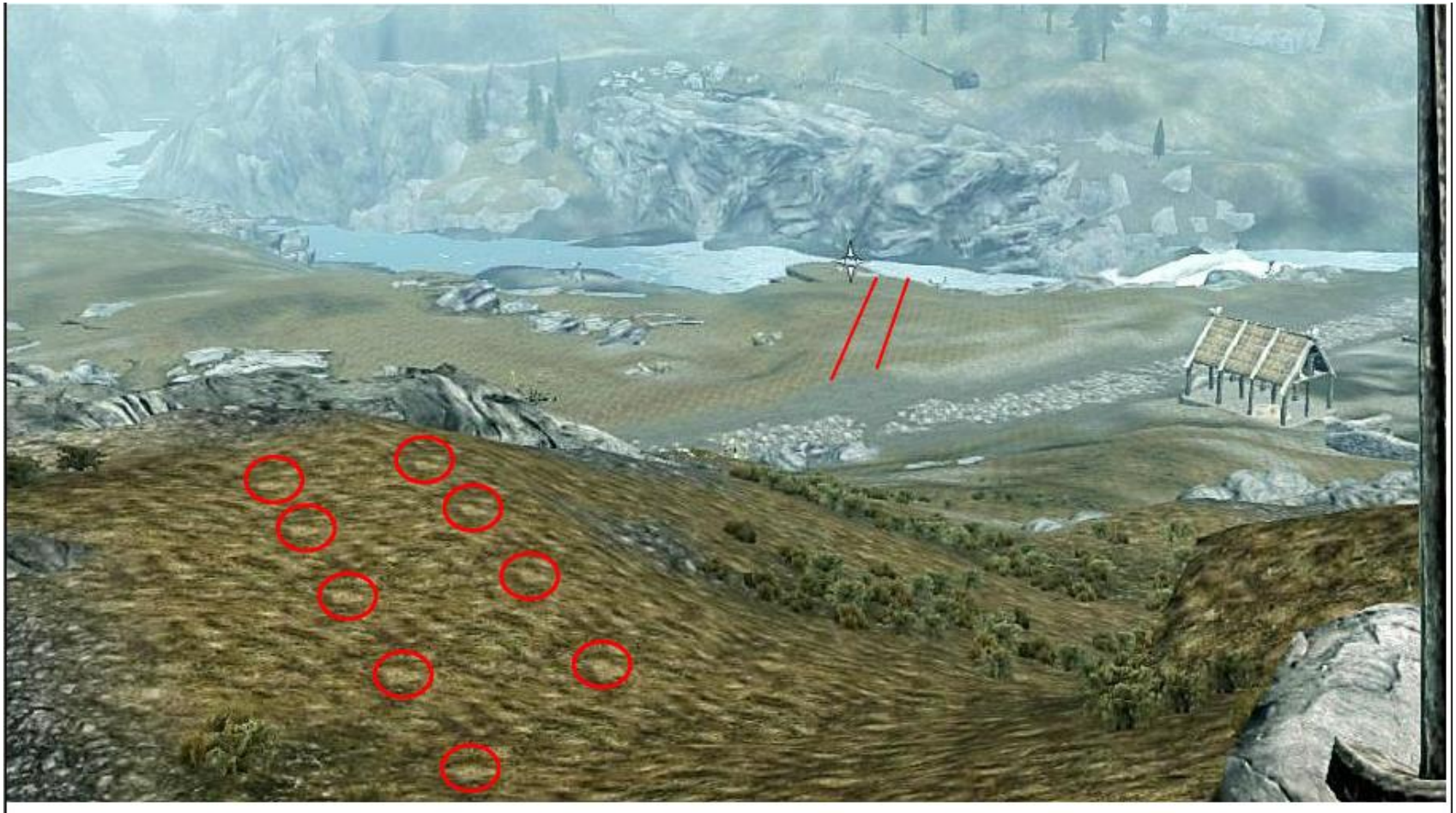
Texture Synthesis: Given an input texture image, produce an output texture image being both visually similar to and pixel-wise different from the input texture.



The output image should ideally be perceived as another part of the same large piece of homogeneous material the input texture is taken from.

Texture synthesis: Motivation

- Important problem in the industry of virtual reality (video games, movies, special effects,. . .).
- Periodic repetition is not satisfying !



2011: Skyrim (Bethesda) screenshot from Three Parts Theory

Texture synthesis algorithms

Two main kinds of algorithm:

1. Texture synthesis using statistical constraints:

   Algorithm:

   1.1 Extract some meaningful “statistics” from the input image (e.g. distribution of colors, of Fourier coefficients, of wavelet coefficients,...).

   1.2 Compute a “random” output image having the same statistics: start from a white noise and alternatively impose the “statistics” of the input.

   Properties:

   + Perceptually stable
   - Generally not good enough for macro-textures

2. Neighborhood-based synthesis algorithms (or “copy-paste” algorithms):

   Algorithm:

   - Compute sequentially an output texture such that each patch of the output corresponds to a patch of the input texture.
   - Many variations have been proposed: scanning orders, grow pixel by pixel or patch by patch, multiscale synthesis, optimization procedure,...

   Properties:

   + Synthesize well macro-textures
   - Can have some speed and stability issue, hard to set parameter...

Texture synthesis algorithms

Two main kinds of algorithm:

1. Texture synthesis using statistical constraints:

   Algorithm:

   1.1 Extract some meaningful “statistics” from the input image (e.g. distribution of colors, of Fourier coefficients, of wavelet coefficients,...).

   1.2 Compute a “random” output image having the same statistics: start from a white noise and alternatively impose the “statistics” of the input.

   Properties:

   + Perceptually stable
   - Generally not good enough for macro-textures

2. Neighborhood-based synthesis algorithms (or “copy-paste” algorithms):

   Algorithm:

   - Compute sequentially an output texture such that each patch of the output corresponds to a patch of the input texture.
   - Many variations have been proposed: scanning orders, grow pixel by pixel or patch by patch, multiscale synthesis, optimization procedure,...

   Properties:

   + Synthesize well macro-textures
   - Can have some speed and stability issue, hard to set parameter...

Texture synthesis algorithms

Two main kinds of algorithm:

1. Texture synthesis using statistical constraints:

   Algorithm:

   1.1 Extract some meaningful “statistics” from the input image (e.g. distribution of colors, of Fourier coefficients, of wavelet coefficients,...).

   1.2 Compute a “random” output image having the same statistics: start from a white noise and alternatively impose the “statistics” of the input.

   Properties:

   + Perceptually stable
   - Generally not good enough for macro-textures

2. Neighborhood-based synthesis algorithms (or “copy-paste” algorithms):

   Algorithm:

   - Compute sequentially an output texture such that each patch of the output corresponds to a patch of the input texture.
   - Many variations have been proposed: scanning orders, grow pixel by pixel or patch by patch, multiscale synthesis, optimization procedure,...

   Properties:

   + Synthesize well macro-textures
   - Can have some speed and stability issue, hard to set parameter...

Texture synthesis by phase randomization

We begin with **Random Phase Noise (RPN)** and
Asymptotic Discrete Spot Noise (ADSN)

[Galerne, Gousseau and Morel, 2011 (a)]
[Galerne, Gousseau and Morel, 2011 (b)]

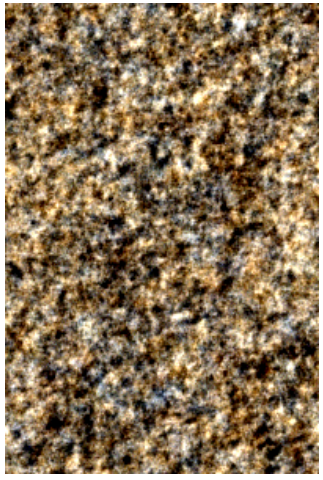
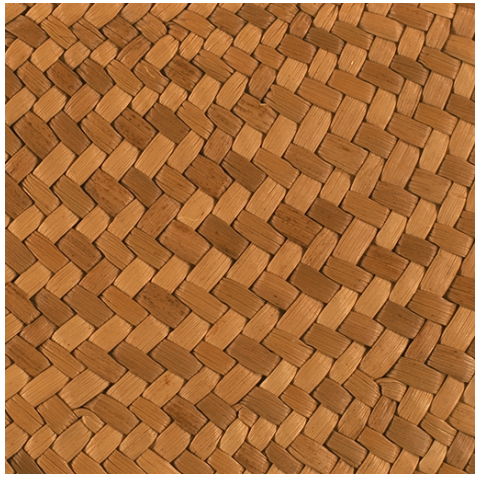
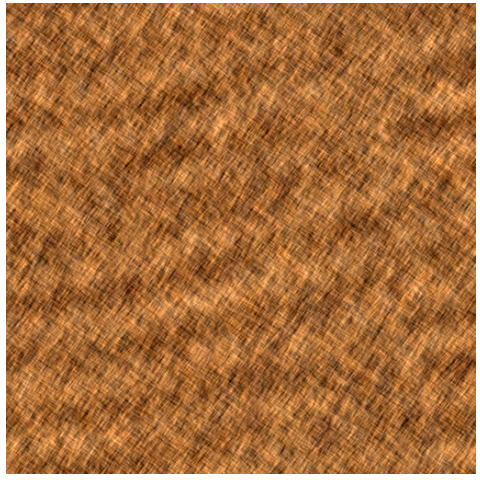
- It belongs to the first category: texture synthesis by statistical constraints.
- Here the “statistics” are the moduli of the Fourier coefficients.

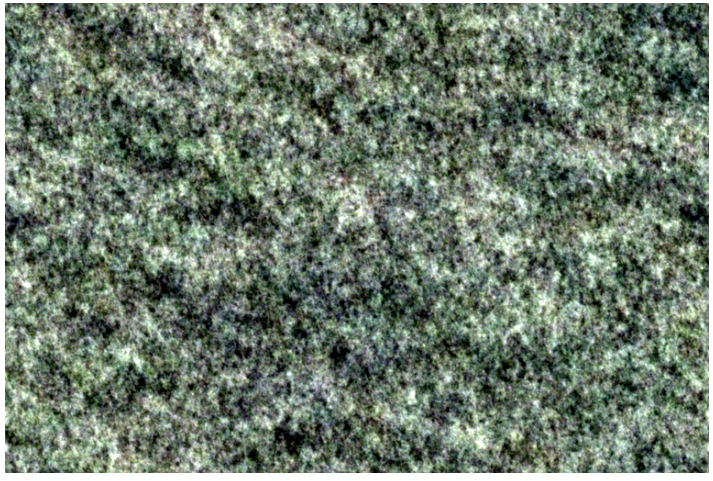
Texture synthesis by phase randomization

- Successful examples with micro-textures

Texture synthesis by phase randomization

- Failure examples with macro-textures



- Perceptual evaluation only: There is no “perception distance”

Outline

Framework

- We work with discrete digital images $u \in \mathbb{R}^{M \times N}$ indexed on the set $\Omega = \{0, \ldots, M-1\} \times \{0, \ldots, N-1\}$.
- Each image is extended by periodicity:

$$u(k, l) = u(k \mod M, l \mod N) \quad \text{for all } (k, l) \in \mathbb{Z}^2.$$

- Consequence: Translation of an image:

Discrete Fourier transform of digital images

- Image domain: $\Omega = \{0, \ldots, M-1\} \times \{0, \ldots, N-1\}$
- Fourier domain $\hat{\Omega}$: the frequency 0 is placed at the center:

$$\hat{\Omega} = \left\{-\frac{M}{2}, \ldots, \frac{M}{2} - 1\right\} \times \left\{-\frac{N}{2}, \ldots, \frac{N}{2} - 1\right\}.$$

Definition:

- The **discrete Fourier transform (DFT)** of u is the **complex-valued** image $\hat{u}$ defined by:

$$\hat{u}(s,t) = \frac{1}{MN} \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} u(k,l) e^{-\frac{2iks\pi}{M}} e^{-\frac{2ilt\pi}{N}}, \quad (s,t) \in \hat{\Omega}.$$

- $|\hat{u}|$: **Fourier modulus** of u.
- $\arg(\hat{u})$: **Fourier phase** of u.

Symmetry property:

- Since u is real-valued, $\hat{u}(-s,-t) = \overline{\hat{u}(s,t)}$.
  $\Rightarrow$ the modulus $|\hat{u}|$ is even and the phase $\arg(\hat{u})$ is odd.

Discrete Fourier transform of digital images

- Image domain: $\Omega = \{0, \ldots, M-1\} \times \{0, \ldots, N-1\}$
- Fourier domain $\hat{\Omega}$: the frequency 0 is placed at the center:

$$\hat{\Omega} = \left\{-\frac{M}{2}, \ldots, \frac{M}{2} - 1\right\} \times \left\{-\frac{N}{2}, \ldots, \frac{N}{2} - 1\right\}.$$

Definition:

- The **discrete Fourier transform (DFT)** of u is the **complex-valued** image $\hat{u}$ defined by:

$$\hat{u}(s,t) = \frac{1}{MN} \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} u(k,l) e^{-\frac{2iks\pi}{M}} e^{-\frac{2ilt\pi}{N}}, \quad (s,t) \in \hat{\Omega}.$$

- $|\hat{u}|$: **Fourier modulus** of u.
- $\arg(\hat{u})$: **Fourier phase** of u.

Symmetry property:

- Since u is real-valued, $\hat{u}(-s,-t) = \overline{\hat{u}(s,t)}$.
  $\Rightarrow$ the modulus $|\hat{u}|$ is even and the phase $\arg(\hat{u})$ is odd.

Discrete Fourier transform of digital images

- Image domain: $\Omega = \{0, \ldots, M-1\} \times \{0, \ldots, N-1\}$
- Fourier domain $\hat{\Omega}$: the frequency 0 is placed at the center:

$$\hat{\Omega} = \left\{-\frac{M}{2}, \ldots, \frac{M}{2} - 1\right\} \times \left\{-\frac{N}{2}, \ldots, \frac{N}{2} - 1\right\}.$$

Definition:

- The **discrete Fourier transform (DFT)** of u is the **complex-valued** image $\hat{u}$ defined by:

$$\hat{u}(s,t) = \frac{1}{MN} \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} u(k,l) e^{-\frac{2iks\pi}{M}} e^{-\frac{2ilt\pi}{N}}, \quad (s,t) \in \hat{\Omega}.$$

- $|\hat{u}|$: **Fourier modulus** of u.
- $\arg(\hat{u})$: **Fourier phase** of u.

Symmetry property:

- Since u is real-valued, $\hat{u}(-s,-t) = \overline{\hat{u}(s,t)}$.
  $\Rightarrow$ the modulus $|\hat{u}|$ is even and the phase $\arg(\hat{u})$ is odd.

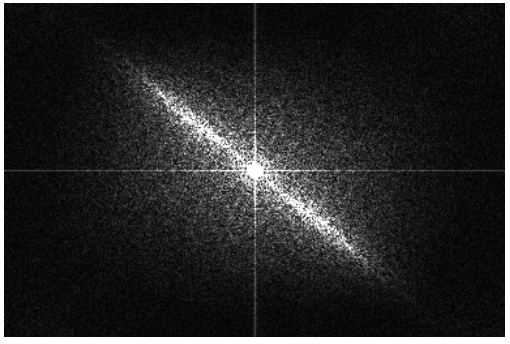
Discrete Fourier transform of digital images

Symmetry property:

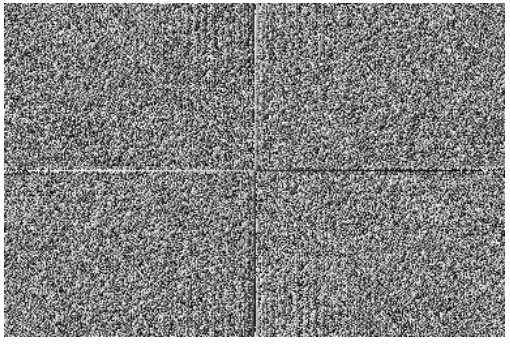
- $|\hat{u}|$: **Fourier modulus** of u is even.
- $\arg(\hat{u})$: **Fourier phase** of u is odd.

Visualization of the DFT:

Image u



Modulus $|\hat{u}|$



Phase $\arg(\hat{u})$

- Interpretation of frequency content

Computation:

- The Fast Fourier Transform algorithm computes $\hat{u}$ in $\mathcal{O}(MN \log(MN))$ operations.
- Efficient FFT implementation: **FFTW** library, a `C/C++` library (used in Matlab).

FFTW = **F**astest **F**ourier **T**ransform in the **W**est

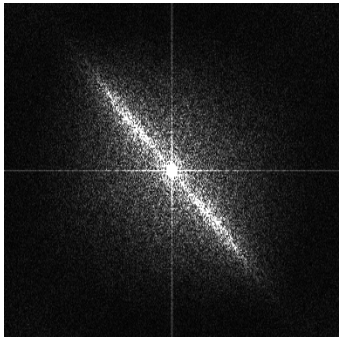
Discrete Fourier transform of digital images

Symmetry property:

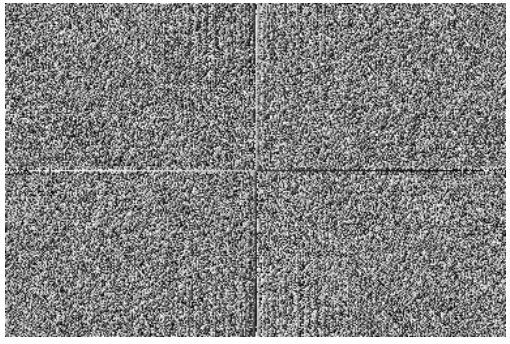
- $|\hat{u}|$: **Fourier modulus** of u is even.
- $\arg(\hat{u})$: **Fourier phase** of u is odd.

Visualization of the DFT:

Image u



Modulus $|\hat{u}|$



Phase $\arg(\hat{u})$

- Interpretation of frequency content

Computation:

- The Fast Fourier Transform algorithm computes $\hat{u}$ in $\mathcal{O}(MN\log(MN))$ operations.
- Efficient FFT implementation: **FFTW** library, a `C/C++` library (used in Matlab).

FFTW = **F**astest **F**ourier **T**ransform in the **W**est

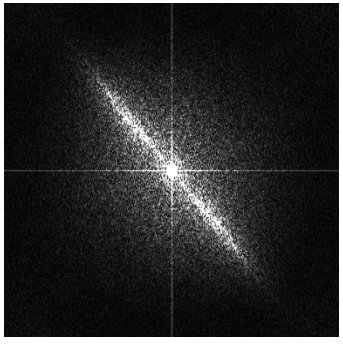
Discrete Fourier transform of digital images

Symmetry property:

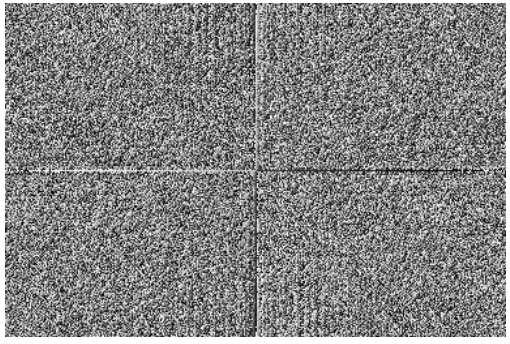
- $|\hat{u}|$: **Fourier modulus** of u is even.
- $\arg(\hat{u})$: **Fourier phase** of u is odd.

Visualization of the DFT:

Image u



Modulus $|\hat{u}|$



Phase $\arg(\hat{u})$

- Interpretation of frequency content

Computation:

- The Fast Fourier Transform algorithm computes $\hat{u}$ in $\mathcal{O}(MN\log(MN))$ operations.
- Efficient FFT implementation: **FFTW** library, a `C/C++` library (used in Matlab).

FFTW = **F**astest **F**ourier **T**ransform in the **W**est

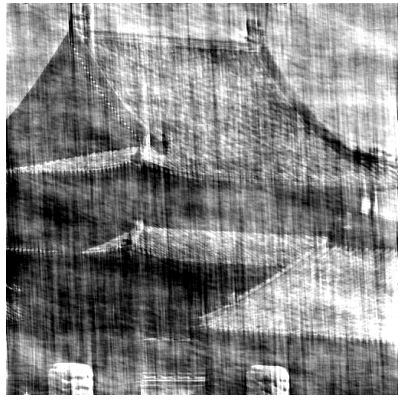
Modulus and phase of a digital image

Exchanging the modulus and the phase of two images:
[Oppenheim and Lim, 1981]

Image 1

Image 2

Modulus of 1
& phase of 2



Modulus of 2
& phase of 1

- Geometric contours are mostly contained in the phase.

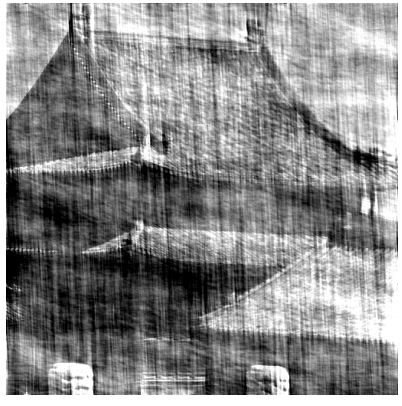
Modulus and phase of a digital image

Exchanging the modulus and the phase of two images:
[Oppenheim and Lim, 1981]

Image 1

Image 2

Modulus of 1
& phase of 2



Modulus of 2
& phase of 1

- Geometric contours are mostly contained in the phase.

Modulus and phase of a digital image

Exchanging the modulus and the phase of two images:
[Oppenheim and Lim, 1981]

Image 1

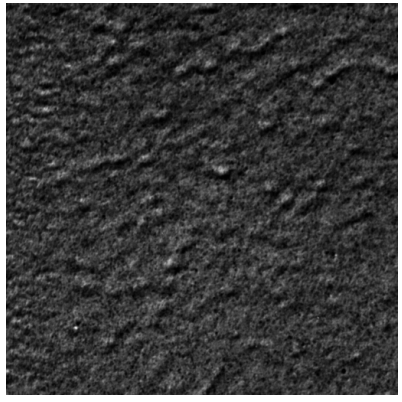
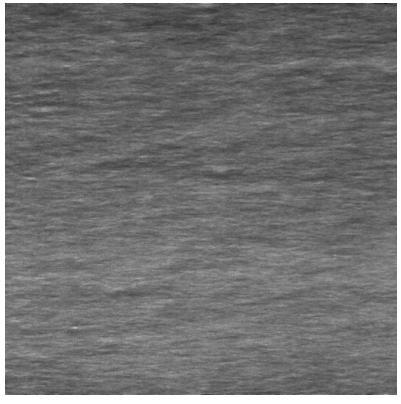


Image 2

Modulus of 1
& phase of 2



Modulus of 2
& phase of 1

- Textures are mostly contained in the modulus.

Modulus and phase of a digital image

Exchanging the modulus and the phase of two images:
[Oppenheim and Lim, 1981]

Image 1

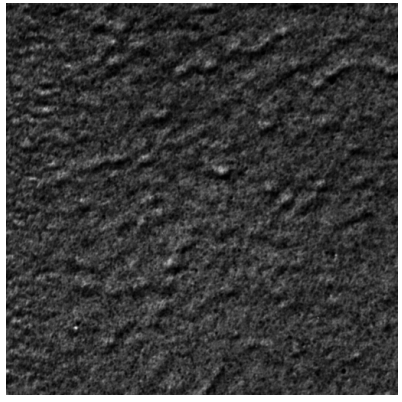
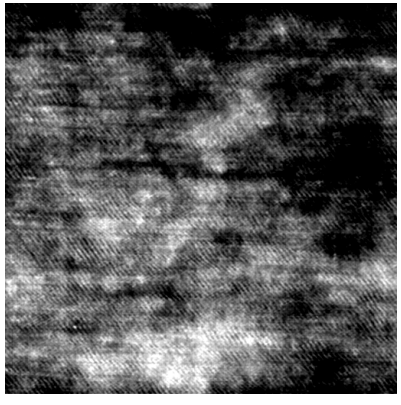
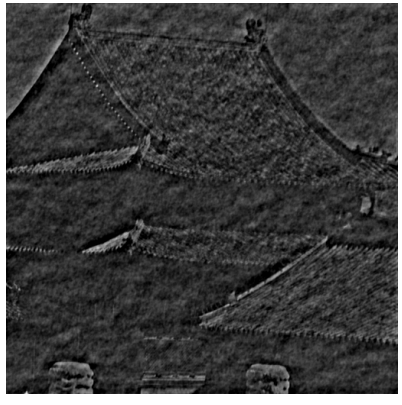


Image 2

Modulus of 1
& phase of 2





Modulus of 2
& phase of 1

- Geometric contours are mostly contained in the phase.
- Textures are mostly contained in the modulus.

Random phase textures

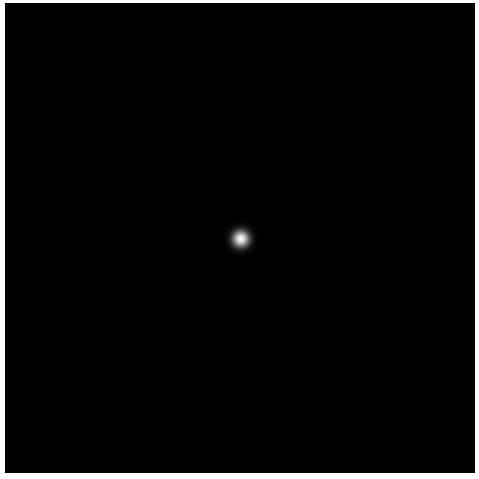
- We call *random phase texture* any image that is perceptually invariant to phase randomization.
- Phase randomization = replace the Fourier phase by a random phase.
- Definition: A random field $\theta : \hat{\Omega} \to \mathbb{R}$ is a ***random phase*** if
  1. Symmetry: θ is odd:
  $$\forall (s,t) \in \hat{\Omega}, \theta(-s,-t) = -\theta(s,t).$$
  2. Distribution: Each component $\theta(s,t)$ is
     - uniform over the interval $]-\pi, \pi]$ if $(s,t) \notin \{(0,0), (\frac{M}{2},0), (0,\frac{N}{2}), (\frac{M}{2},\frac{N}{2})\}$,
     - uniform over the set $\{0,\pi\}$ otherwise.
  3. Independence: For each subset $\mathcal{S} \subset \hat{\Omega}$ that does not contain distinct symmetric points, the r.v. $\{\theta(s,t) | (s,t) \in \mathcal{S}\}$ are independent.

- Property: The Fourier phase of a Gaussian white noise X is a random phase.
- (Lazy) simulation: In Matlab, `theta = angle(fft2(randn(M,N)))`.
- *Random phase textures* constitute a "limited" subclass of the set of textures.

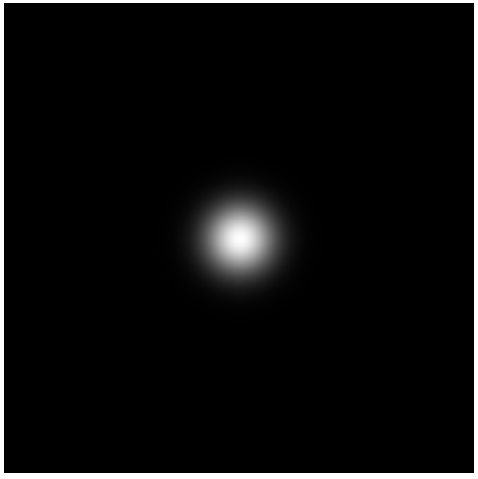
Random Phase Noise (RPN)

- Texture synthesis algorithm: ***random phase noise*** **(RPN)**: [van Wijk, 1991]

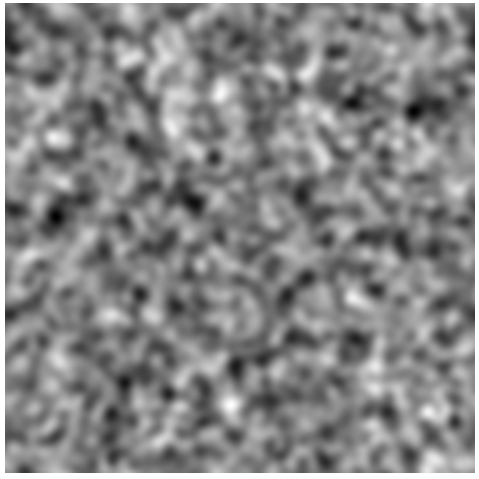
1. Compute the DFT $\hat{h}$ of the input h
2. Compute a random phase θ using a pseudo-random number generator
3. Set $\hat{Z} = \left|\hat{h}\right| e^{i\theta}$ (or $\hat{Z} = \hat{h}e^{i\theta}$)
4. Return Z the inverse DFT of $\hat{Z}$



Original image h



Modulus $\left|\hat{h}\right|$



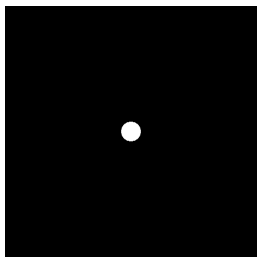
RPN associated with h

Discrete spot noise [van Wijk, 1991]

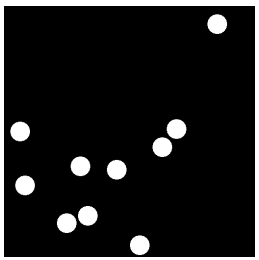
- Let h be a discrete image called *spot*.
- Let (X_k) be a sequence of random translation vectors which are i.d.d. and uniformly distributed over Ω.
- The **discrete spot noise of order** n **associated with** h is the random image

$$f_n(x) = \sum_{k=1}^{n} h(x - X_k).$$

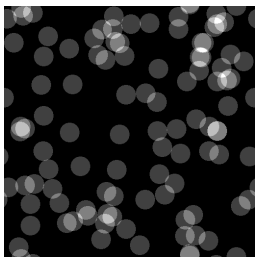
(translations with periodic boundary conditions)



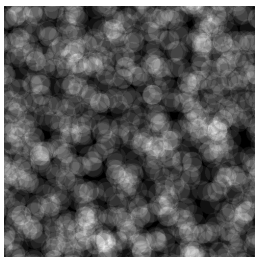
Spot h



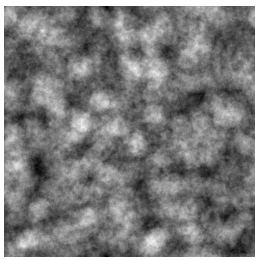
$n = 10$



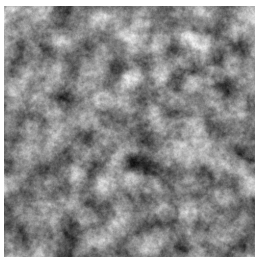
$n = 10^2$



$n = 10^3$

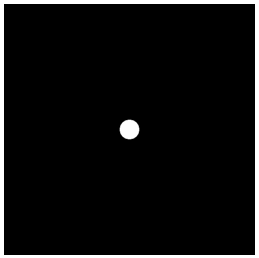


$n = 10^4$

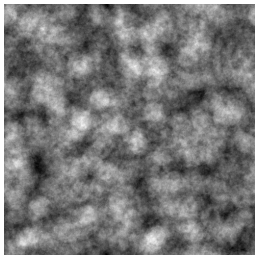


$n = 10^5$

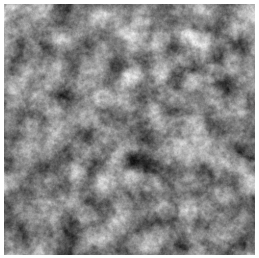
Limit of the DSN model ?



Spot h



$n = 10^4$



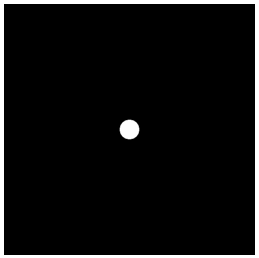
$n = 10^5$

?

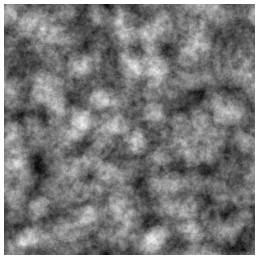
$n = +\infty$

- For texture synthesis we are more particularly interested in the limit of the *DSN*: the ***asymptotic discrete spot noise*** (***ADSN***).
- The *DSN* of order n, $f_n(x) = \sum_k h(x - X_k)$, is the sum of the n i.i.d. random images $h(\cdot - X_k)$.
- **Central limit theorem for random vectors:**...

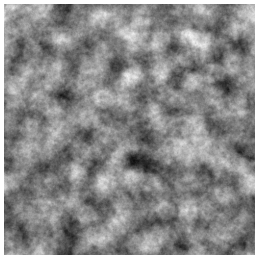
Limit of the DSN model ?



Spot h



$n = 10^4$



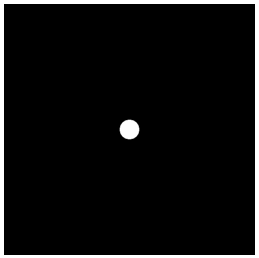
$n = 10^5$

?

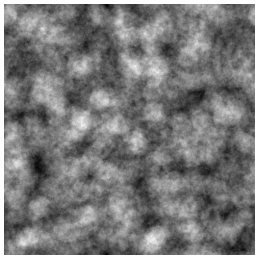
$n = +\infty$

- For texture synthesis we are more particularly interested in the limit of the *DSN*: the ***asymptotic discrete spot noise*** (***ADSN***).
- The *DSN* of order n, $f_n(x) = \sum_k h(x - X_k)$, is the sum of the n i.i.d. random images $h(\cdot - X_k)$.
- **Central limit theorem for random vectors:...**

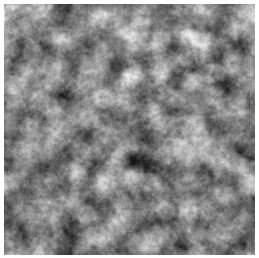
Limit of the DSN model ?



Spot h



$n = 10^4$



$n = 10^5$

?

$n = +\infty$

- For texture synthesis we are more particularly interested in the limit of the *DSN*: the ***asymptotic discrete spot noise*** **(*ADSN*)**.
- The *DSN* of order n, $f_n(x) = \sum_k h(x - X_k)$, is the sum of the n i.i.d. random images $h(\cdot - X_k)$.
- **Central limit theorem for random vectors:**...

Basics of Gaussian random vectors

Gaussian random vectors in 1D:

- $Y = (Y_1, \ldots, Y_N)^T \in \mathbb{R}^N$ is a Gaussian random vector if every linear combination of the component of Y has a Gaussian distribution :

$$\forall \alpha \in \mathbb{R}^N, \quad \langle Y, \alpha \rangle \sim \mathcal{N}(m, \sigma^2) \text{ for some } m \text{ and } \sigma^2.$$

- The expectation $\mu \in \mathbb{R}^N$ of Y is the vector $\mu = \mathbb{E}(Y)$, i.e. for all $i \in \{1, \ldots, N\}$, $\mu_i = \mathbb{E}(Y_i)$.
- The covariance of Y is the matrix $C \in \mathbb{R}^{N \times N}$ such that

$$C(i,j) = \text{Cov}(Y_i, Y_j) = \mathbb{E}((Y_i - \mu_i)(Y_j - \mu_j)).$$

- The covariance is symmetric and positive

$$\forall \alpha = (\alpha_1, \ldots, \alpha_N) \in \mathbb{R}^N, \quad \sum_{i,j=1}^{N} \alpha_i \alpha_j C(i,j) \geq 0 \quad \text{(this is just Var}(\langle Y, \alpha \rangle) \geq 0)$$

- Gaussian vector distributions are characterized by their expectation μ and covariance matrix C, one denotes the distribution by $\mathcal{N}(\mu, C)$.
- **If C is invertible**, $Y \sim \mathcal{N}(\mu, C)$ has density

$$f_Y(x) = \frac{1}{\sqrt{(2\pi)^N \det(C)}} \exp\left(-\frac{1}{2}(x-\mu)^T C^{-1}(x-\mu)\right)$$

Basics of Gaussian random vectors

Theorem (Central limit theorem for random vectors)

If $(X_n)_{n\geq 1}$ is a sequence of iid random vectors with expectation μ and , then

$$\left(\frac{\left(\sum_{k=1}^{n} X_k\right) - n\mu}{\sqrt{n}}\right)_n \quad \textit{converges in distribution to} \quad \mathcal{N}(0, C).$$

Gaussian random vectors and linear application:

- If $Y_1 \in \mathbb{R}^N$ has distribution $\mathcal{N}(\mu_1, C_1)$ and $A \in \mathbb{R}^{M\times N}$ then $Y_2 = AY_1 \in \mathbb{R}^M$ is Gaussian with

$$\mathbb{E}(Y_2) = A\mathbb{E}(Y_1) = A\mu \quad \text{and} \quad \mathrm{Cov}(Y_2) = A\,\mathrm{Cov}(Y_1)A^T = AC_1A^T.$$

Simulation Gaussian random vectors:

Given a mean vector μ and a covariance matrix C :

1. Compute a matrix A such that $C = AA^T$
   (eg Cholesky decomposition or squareroot of C)
2. Generate a Gaussian white noise vector $X \sim \mathcal{N}(0, I_N)$
   (`randn` in Matlab)
3. Return $Y = \mu + AX$.

Basics of Gaussian random vectors

Theorem (Central limit theorem for random vectors)

If $(X_n)_{n\geq 1}$ is a sequence of iid random vectors with expectation μ and , then

$$\left(\frac{\left(\sum_{k=1}^{n} X_k\right) - n\mu}{\sqrt{n}}\right)_n \quad \textit{converges in distribution to} \quad \mathcal{N}(0, C).$$

Gaussian random vectors and linear application:

- If $Y_1 \in \mathbb{R}^N$ has distribution $\mathcal{N}(\mu_1, C_1)$ and $A \in \mathbb{R}^{M\times N}$ then $Y_2 = AY_1 \in \mathbb{R}^M$ is Gaussian with

$$\mathbb{E}(Y_2) = A\mathbb{E}(Y_1) = A\mu \quad \text{and} \quad \mathrm{Cov}(Y_2) = A\,\mathrm{Cov}(Y_1)A^T = AC_1A^T.$$

Simulation Gaussian random vectors:

Given a mean vector μ and a covariance matrix C :

1. Compute a matrix A such that $C = AA^T$ (eg Cholesky decomposition or squareroot of C)
2. Generate a Gaussian white noise vector $X \sim \mathcal{N}(0, I_N)$ (`randn` in Matlab)
3. Return $Y = \mu + AX$.

Basics of Gaussian random vectors

Theorem (Central limit theorem for random vectors)

If $(X_n)_{n\geq 1}$ is a sequence of iid random vectors with expectation μ and , then

$$\left(\frac{\left(\sum_{k=1}^{n} X_k\right) - n\mu}{\sqrt{n}}\right)_n \quad \textit{converges in distribution to} \quad \mathcal{N}(0, C).$$

Gaussian random vectors and linear application:

- If $Y_1 \in \mathbb{R}^N$ has distribution $\mathcal{N}(\mu_1, C_1)$ and $A \in \mathbb{R}^{M\times N}$ then $Y_2 = AY_1 \in \mathbb{R}^M$ is Gaussian with

$$\mathbb{E}(Y_2) = A\mathbb{E}(Y_1) = A\mu \quad \text{and} \quad \mathrm{Cov}(Y_2) = A\,\mathrm{Cov}(Y_1)A^T = AC_1A^T.$$

Simulation Gaussian random vectors:

Given a mean vector μ and a covariance matrix C :

1. Compute a matrix A such that $C = AA^T$
   (eg Cholesky decomposition or squareroot of C)
2. Generate a Gaussian white noise vector $X \sim \mathcal{N}(0, I_N)$
   (`randn` in Matlab)
3. Return $Y = \mu + AX$.

Basics of Gaussian random vectors

Gaussian random vectors in 2D:

- Same story with the pixel indexes for the coordinates : $Y = (Y(x))_{x \in \Omega}$.
- The covariance matrix has two indexes : $C = (C(x, y)_{x,y \in \Omega}$.
- For (even small) images, in general the covariance matrix cannot be stored ! One needs to limit to simple models : sparse covariance, stationary distributions,. . .

Stationary random vectors in 2D:

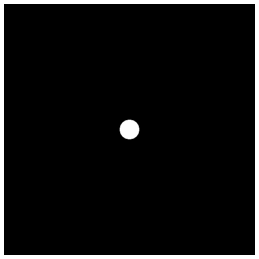
- A random vector Y is stationary if Y and its translations have the same distribution.
- If Y is stationary then $\mathbb{E}(Y)$ is a constant vector ($\mathbb{E}(Y(x)) = \mathbb{E}(Y(y))$) and

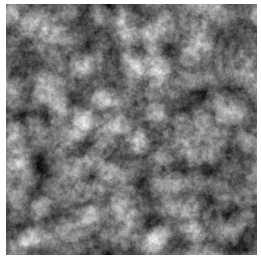
$$C(x, y) = C(x - y, 0)$$

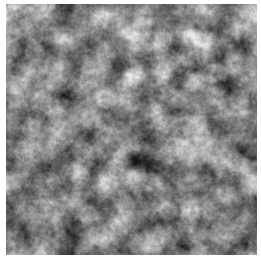
is a “circulant matrix”. Then the covariance can be stored in a single image $c(x) = C(x, 0)$ so that

$$C(x, y) = c(x - y), \quad x, y \in \Omega.$$

Limit of the DSN model ?


Spot h


$n = 10^4$

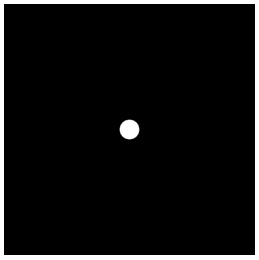

$n = 10^5$

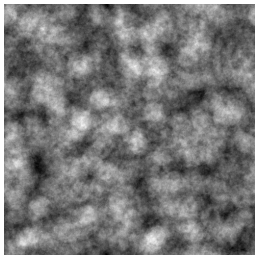
?

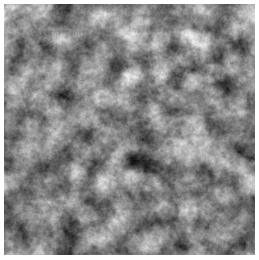
$n = +\infty$

- For texture synthesis we are more particularly interested in the limit of the *DSN*: the ***asymptotic discrete spot noise*** (***ADSN***).
- The *DSN* of order n, $f_n(x) = \sum_k h(x - X_k)$, is the sum of the n i.i.d. random images $h(\cdot - X_k)$.
- **Central limit theorem for random vectors:**
  The sequence of random images $\left(\frac{f_n - n\mathbb{E}(h(\cdot - X_1))}{\sqrt{n}} \right)_{n \in \mathbb{N}^*}$ converges in distribution towards the **Gaussian random vector** $Y = (Y(x))_{x \in \Omega}$ with zero mean and covariance $\mathrm{Cov}(h(\cdot - X_1))$.

Limit of the DSN model ?


Spot h


$n = 10^4$

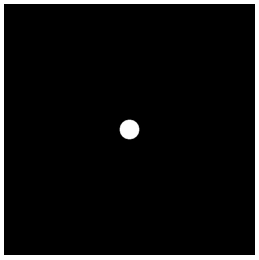

$n = 10^5$

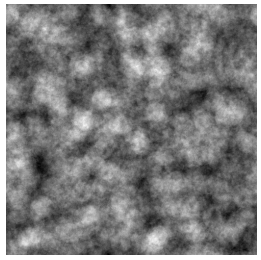
?

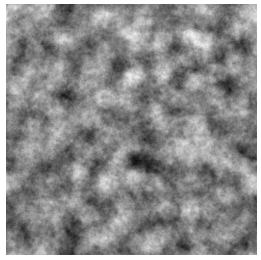
$n = +\infty$

- For texture synthesis we are more particularly interested in the limit of the *DSN*: the ***asymptotic discrete spot noise*** (***ADSN***).
- The *DSN* of order n, $f_n(x) = \sum_k h(x - X_k)$, is the sum of the n i.i.d. random images $h(\cdot - X_k)$.
- **Central limit theorem for random vectors:**
  The sequence of random images $\left(\frac{f_n - n\mathbb{E}(h(\cdot - X_1))}{\sqrt{n}} \right)_{n \in \mathbb{N}^*}$ converges in distribution towards the **Gaussian random vector** $Y = (Y(x))_{x \in \Omega}$ with zero mean and covariance $\mathrm{Cov}(h(\cdot - X_1))$.

Limit of the DSN model ?


Spot h


$n = 10^4$


$n = 10^5$

?

$n = +\infty$

- For texture synthesis we are more particularly interested in the limit of the *DSN*: the ***asymptotic discrete spot noise*** (***ADSN***).
- The *DSN* of order n, $f_n(x) = \sum_k h(x - X_k)$, is the sum of the n i.i.d. random images $h(\cdot - X_k)$.
- **Central limit theorem for random vectors:**
  The sequence of random images $\left(\frac{f_n - n\mathbb{E}(h(\cdot - X_1))}{\sqrt{n}} \right)_{n \in \mathbb{N}^*}$ converges in distribution towards the **Gaussian random vector** $Y = (Y(x))_{x \in \Omega}$ with zero mean and covariance $\mathrm{Cov}(h(\cdot - X_1))$.

Asymptotic discrete spot noise (*ADSN*)

Expectation of the random translations:

$$\begin{aligned}\mathbb{E}(h(x - X_1)) &= \sum_{y \in \Omega} h(x - y)\mathbb{P}(X_1 = y) \\ &= \sum_{y \in \Omega} h(x - y)\frac{1}{MN} \\ &= \frac{1}{MN}\sum_{z \in \Omega} h(z) \\ &= \text{mean of } h.\end{aligned}$$

- $\mathbb{E}(h(x - X_1)) = m$, where m is the mean of h.

Asymptotic discrete spot noise (*ADSN*)

Covariance of the random translations: Let $x, y \in \Omega$,

$$\begin{aligned}
\mathrm{Cov}(h(x - X_1), h(y - X_1)) &= \mathbb{E}((h(x - X_1) - m)(h(y - X_1) - m)) \\
&= \sum_{z \in \Omega} (h(x - z) - m)(h(y - z) - m)\mathbb{P}(X_1 = z) \\
&= \frac{1}{MN} \sum_{z \in \Omega} (h(x - z) - m)(h(y - z) - m) \\
&= C_h(x, y).
\end{aligned}$$

- $\mathrm{Cov}(h(x - X_1), h(y - X_1)) = C_h(x, y)$ where C_h is the **autocorrelation** of h:

$$C_h(x, y) = \frac{1}{MN} \sum_{t \in \Omega} (h(x - t) - m)\,(h(y - t) - m)\,, \quad (x, y) \in \Omega.$$

Asymptotic discrete spot noise (*ADSN*)

- For texture synthesis we are more particularly interested in the limit of the *DSN*: the ***asymptotic discrete spot noise*** (***ADSN***).

Expectation and covariance of the random translations:

- $\mathbb{E}(h(x - X_1)) = m$, where m is the arithmetic mean of h.
- $\mathrm{Cov}(h(x - X_1), h(y - X_1)) = C_h(x - y)$ where C_h is the autocorrelation of h:

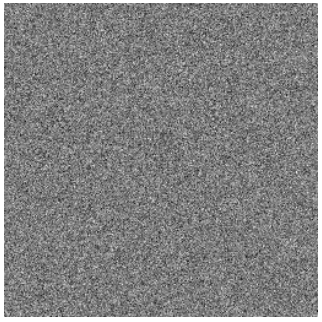
$$C_h(x, y) = \frac{1}{MN} \sum_{t \in \Omega} (h(x - t) - m)\,(h(y - t) - m)\,, \quad (x, y) \in \Omega.$$

Definition of *ADSN*:

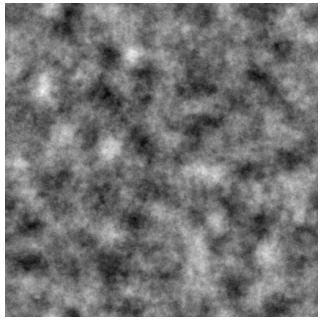
- The *ADSN* associated with h is the Gaussian vector $\mathcal{N}(0, C_h)$.

Simulation of the ADSN

Definition of *ADSN*: the *ADSN* associated with h is the Gaussian vector $\mathcal{N}(0, C_h)$.



Gaussian white noise: pixels are independent and have Gaussian distribution

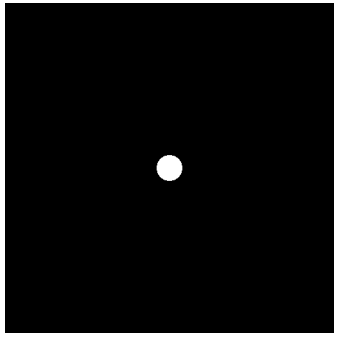


Gaussian vector: pixels have Gaussian distribution and are correlated

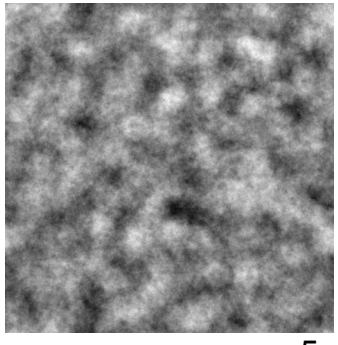
Convolution product: $(f * g)(x) = \sum_{y \in \Omega} f(x-y)g(y),\ x \in \Omega.$

Simulation of the *ADSN*:

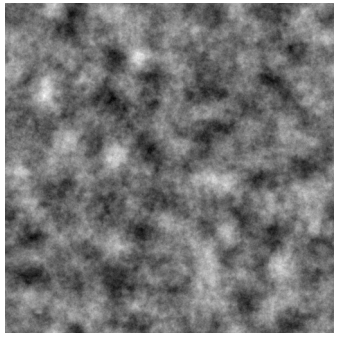
- Let $h \in \mathbb{R}^{M \times N}$ be a an image, m be the mean of h and X be a Gaussian white noise image.
- The random image $\frac{1}{\sqrt{MN}}(h-m) * X$ is the *ADSN* associated with h.



Spot h



DSN, $n = 10^5$



ADSN

ADSN Simulation

Proof of $Y = \frac{1}{\sqrt{MN}}(h - m) * X \sim \mathcal{N}(0, C_h)$.

- Y is obtained from X in applying a linear map. Since X is a Gaussian vector, Y is also a Gaussian vector.
- One just needs to show that $\mathbb{E}(Y(x)) = 0$ and $\mathrm{Cov}(Y(x), Y(y)) = C_h(x, y)$.
- By linearity, $\mathbb{E}(Y(x)) = \frac{1}{\sqrt{MN}}(h - m) * \mathbb{E}(X)(x) = 0$.
- Let $x, y \in \Omega$,

$$\mathrm{Cov}(Y(x), Y(y)) = \mathbb{E}(Y(x)Y(y))$$

$$= \frac{1}{MN}\mathbb{E}\left(\sum_{s\in\Omega}(h(s-x)-m)X(s)\sum_{t\in\Omega_{M,N}}(h(t-y)-m)X(t)\right)$$

$$= \frac{1}{MN}\sum_{s,t\in\Omega}(h(s-x)-m)(h(t-y)-m)\underbrace{\mathbb{E}(X(s)X(t))}_{=1 \text{ if } s=t \text{ and } 0 \text{ otherwise}}$$

$$= \frac{1}{MN}\sum_{s\in\Omega}(h(s-x)-m)(h(t-y)-m)$$

$$= C_h(x, y)$$

ADSN Simulation

Proof of $Y = \frac{1}{\sqrt{MN}}(h - m) * X \sim \mathcal{N}(0, C_h)$.

- Y is obtained from X in applying a linear map. Since X is a Gaussian vector, Y is also a Gaussian vector.
- One just needs to show that $\mathbb{E}(Y(x)) = 0$ and $\mathrm{Cov}(Y(x), Y(y)) = C_h(x, y)$.
- By linearity, $\mathbb{E}(Y(x)) = \frac{1}{\sqrt{MN}}(h - m) * \mathbb{E}(X)(x) = 0$.
- Let $x, y \in \Omega$,

$$\mathrm{Cov}(Y(x), Y(y)) = \mathbb{E}(Y(x)Y(y))$$

$$= \frac{1}{MN}\mathbb{E}\left(\sum_{s\in\Omega}(h(s-x)-m)X(s)\sum_{t\in\Omega_{M,N}}(h(t-y)-m)X(t)\right)$$

$$= \frac{1}{MN}\sum_{s,t\in\Omega}(h(s-x)-m)(h(t-y)-m)\underbrace{\mathbb{E}(X(s)X(t))}_{=1 \text{ if } s=t \text{ and } 0 \text{ otherwise}}$$

$$= \frac{1}{MN}\sum_{s\in\Omega}(h(s-x)-m)(h(t-y)-m)$$

$$= C_h(x, y)$$

ADSN Simulation

Proof of $Y = \frac{1}{\sqrt{MN}}(h - m) * X \sim \mathcal{N}(0, C_h)$.

- Y is obtained from X in applying a linear map. Since X is a Gaussian vector, Y is also a Gaussian vector.
- One just needs to show that $\mathbb{E}(Y(x)) = 0$ and $\mathrm{Cov}(Y(x), Y(y)) = C_h(x, y)$.
- By linearity, $\mathbb{E}(Y(x)) = \frac{1}{\sqrt{MN}}(h - m) * \mathbb{E}(X)(x) = 0$.
- Let $x, y \in \Omega$,

$$\mathrm{Cov}(Y(x), Y(y)) = \mathbb{E}(Y(x)Y(y))$$

$$= \frac{1}{MN}\mathbb{E}\left(\sum_{s\in\Omega}(h(s-x)-m)X(s)\sum_{t\in\Omega_{M,N}}(h(t-y)-m)X(t)\right)$$

$$= \frac{1}{MN}\sum_{s,t\in\Omega}(h(s-x)-m)(h(t-y)-m)\underbrace{\mathbb{E}(X(s)X(t))}_{=1 \text{ if } s=t \text{ and } 0 \text{ otherwise}}$$

$$= \frac{1}{MN}\sum_{s\in\Omega}(h(s-x)-m)(h(t-y)-m)$$

$$= C_h(x, y)$$

ADSN Simulation

Proof of $Y = \frac{1}{\sqrt{MN}}(h - m) * X \sim \mathcal{N}(0, C_h)$.

- Y is obtained from X in applying a linear map. Since X is a Gaussian vector, Y is also a Gaussian vector.
- One just needs to show that $\mathbb{E}(Y(x)) = 0$ and $\mathrm{Cov}(Y(x), Y(y)) = C_h(x, y)$.
- By linearity, $\mathbb{E}(Y(x)) = \frac{1}{\sqrt{MN}}(h - m) * \mathbb{E}(X)(x) = 0$.
- Let $x, y \in \Omega$,

$$\begin{aligned}
\mathrm{Cov}(Y(x), Y(y)) &= \mathbb{E}(Y(x)Y(y)) \\
&= \frac{1}{MN}\mathbb{E}\left(\sum_{s\in\Omega}(h(s-x)-m)X(s)\sum_{t\in\Omega_{M,N}}(h(t-y)-m)X(t)\right) \\
&= \frac{1}{MN}\sum_{s,t\in\Omega}(h(s-x)-m)(h(t-y)-m)\underbrace{\mathbb{E}(X(s)X(t))}_{=1 \text{ if } s=t \text{ and } 0 \text{ otherwise}} \\
&= \frac{1}{MN}\sum_{s\in\Omega}(h(s-x)-m)(h(t-y)-m) \\
&= C_h(x, y)
\end{aligned}$$

ADSN Simulation

Simulation Gaussian random vectors:

Given a mean vector μ and a covariance matrix C :

1. Compute a matrix A such that $C = AA^T$
   (eg Cholesky decomposition or squareroot of C)
2. Generate a Gaussian white noise vector $X \sim \mathcal{N}(0, I_N)$
   (`randn` in Matlab)
3. Return $Y = \mu + AX$.

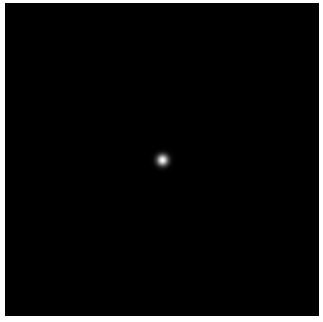
Remark:

- Here with
$$Y = \frac{1}{\sqrt{MN}} (h - m) * X \sim \mathcal{N}(0, C_h)$$
we just showed that the linear operator
$$A = \text{“convolution by } \frac{1}{\sqrt{MN}} (h - m)\text{”}$$
satisfies $AA^T = C_h$ (as would the Cholesky decomposition).

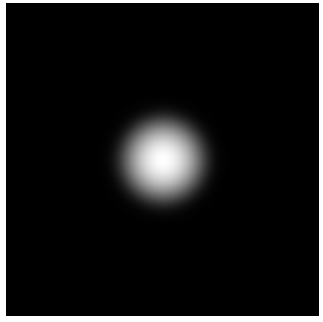
Differences between *RPN* and *ADSN*

Proposition:

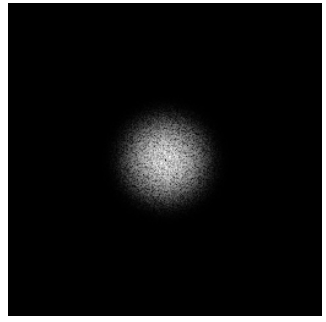
- *RPN* and *ADSN* both have a random phase.
- The Fourier modulus of *RPN* is the one of h.
- The Fourier modulus of *ADSN* is the pointwise multiplication between $\left|\hat{h}\right|$ and a Rayleigh noise.



Spot h

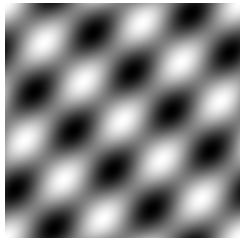


RPN Modulus

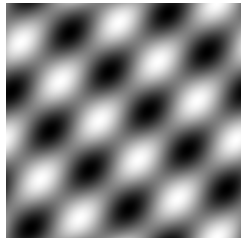


ADSN Modulus

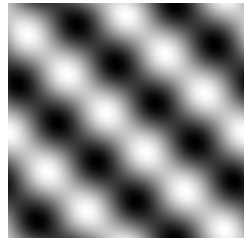
- ***RPN* and *ADSN* are two different processes.**



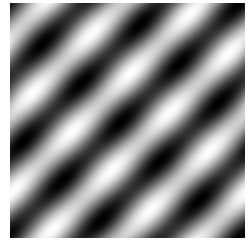
Spot h



RPN



An *ADSN* realization

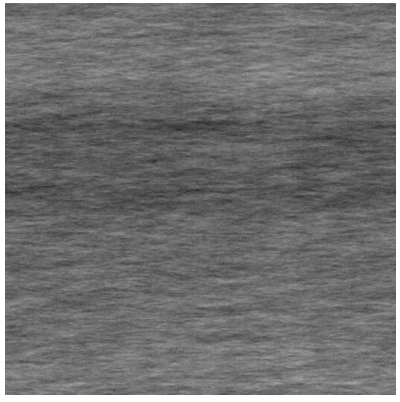


Another *ADSN* realization

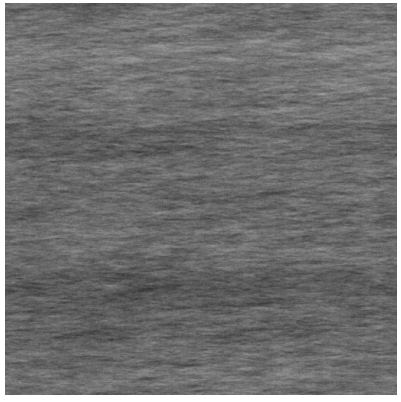
RPN and *ADSN* associated to texture images

- We add the original mean to *RPN* and *ADSN* realizations.
- *RPN* and *ADSN* are texture models with same mean and same covariance than the original image *h*.
- Some textures are relatively well reproduced by *RPN* and *ADSN*.

Original image



RPN



ADSN

- ... But several developments are necessary to derive texture synthesis algorithms from sample.

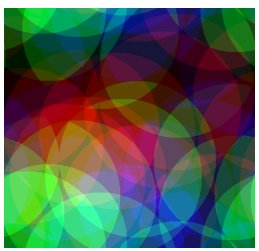
Extension to color images

- We use the RGB color representation for color images.
- **Color *ADSN*:** The definition of Discrete Spot Noise extends to color images $h = (h_r, h_g, h_b)$.
- The color *ADSN* Y is the limit Gaussian process obtained in letting the number of spots tend to $+\infty$. It is simulated by:

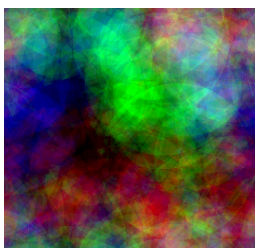
$$Y = \frac{1}{\sqrt{MN}} \begin{pmatrix} (h_r - m_r \mathbf{1}) * X \\ (h_g - m_g \mathbf{1}) * X \\ (h_b - m_b \mathbf{1}) * X \end{pmatrix}, \quad X \text{ a Gaussian white noise.}$$

- One convolves each color channel with the **same** Gaussian white noise X.

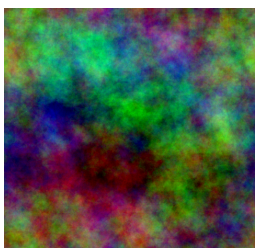
Spot h



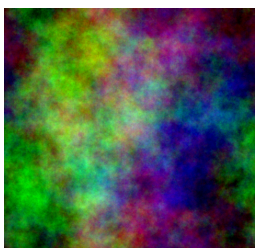
$n = 10$



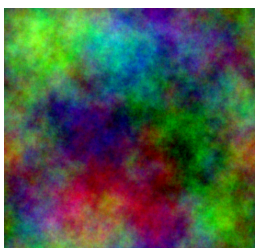
$n = 10^2$



$n = 10^3$



$n = 10^4$



color *ADSN*

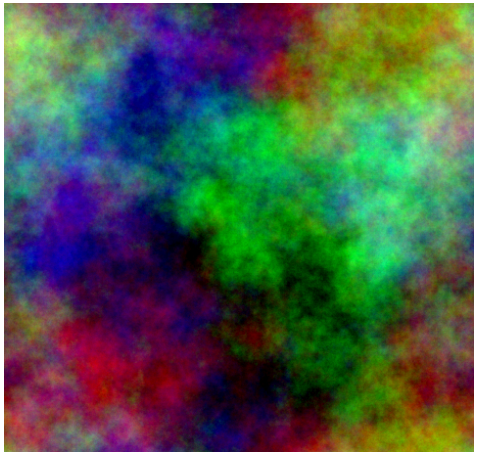
- Phase of color *ADSN*: The same random phase is added to the Fourier transform of each color channel.

Extension to color images

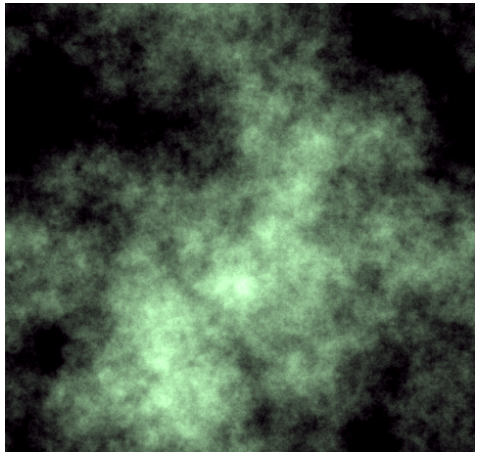
- **Color *RPN*:** By analogy, the *RPN* associated with a color image $h = (h_r, h_g, h_b)$ is the color image obtained by **adding the same random phase** to the Fourier transform of each color channel.

Original image h

Color *RPN*



"Wrong *RPN*": each channel has the same random phase

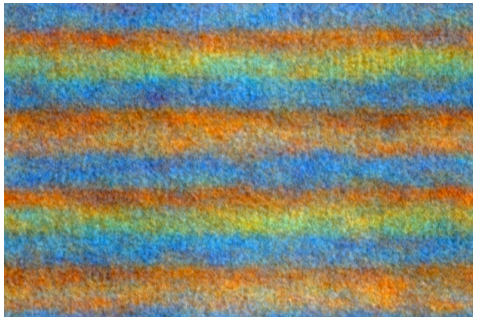


$$\hat{h} = \begin{pmatrix} |\hat{h}_R| e^{i\varphi_R} \\ |\hat{h}_G| e^{i\varphi_G} \\ |\hat{h}_B| e^{i\varphi_B} \end{pmatrix} \qquad \hat{Z} = \begin{pmatrix} |\hat{h}_R| e^{i(\varphi_R+\theta)} \\ |\hat{h}_G| e^{i(\varphi_G+\theta)} \\ |\hat{h}_B| e^{i(\varphi_B+\theta)} \end{pmatrix} \qquad \hat{Z}_W = \begin{pmatrix} |\hat{h}_R| e^{i\theta} \\ |\hat{h}_G| e^{i\theta} \\ |\hat{h}_B| e^{i\theta} \end{pmatrix}$$

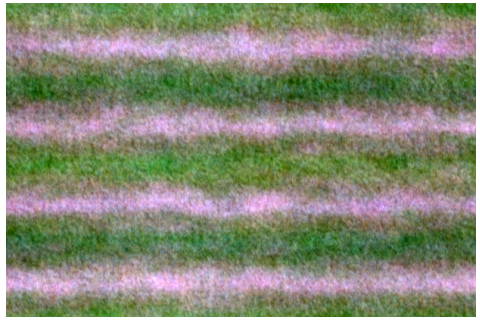
Extension to color images

- Another example with a real-world texture.

Original image h



Color *RPN*



"Wrong *RPN*"

- Preserving the original phase displacement between the color channels is essential for color consistency.
- ...however for most monochromatic textures, there is no huge difference.

Original image h

Color *RPN*

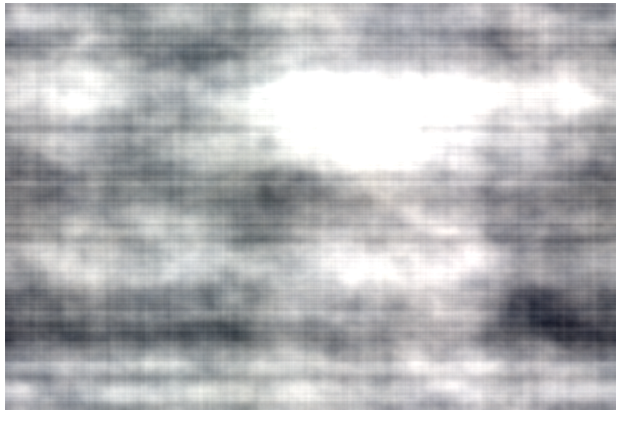
"Wrong *RPN*"

Avoiding artifacts due to non periodicity

- Both *ADSN* and *RPN* algorithms are based on the fast Fourier transform (FFT).
  $\Longrightarrow$ implicit hypothesis of periodicity
- Using non periodic samples yields important artifacts.

Spot *h*

ADSN



Avoiding artifacts due to non periodicity

- Our solution: Force the periodicity of the input sample.
- The original image h is replaced by its **periodic component** $p = \text{per}(h)$, see L. Moisan's course [Moisan, 2011].
- Definition of the periodic component p of h: p unique solution of

$$\begin{cases} \Delta p = \Delta_i h \\ \text{mean}(p) = \text{mean}(h) \end{cases}$$

where, noting N_x the neighborhood of $x \in \Omega$ for 4-connexity:

$$\Delta f(x) = 4f(x) - \sum_{y \in N_x} f(y) \quad \text{and} \quad \Delta_i f(x) = |N_x \cap \Omega| \, f(x) - \sum_{y \in N_x \cap \Omega} f(y).$$

These two Laplacians only differ at the border:

- Δ: discrete Laplacian with periodic boundary conditions
- Δ_i: discrete Laplacian without periodic boundary conditions (index i for interior)

- p is "visually close" to h (same Laplacian).
- p is fastly computed using the FFT...

FFT-based Poisson Solver

Periodic Poisson problem: Find the image p such that

$$\begin{cases} \Delta p = \Delta_i h \\ \text{mean}(p) = \text{mean}(h) \end{cases}$$

In the Fourier domain, this system becomes:

$$\begin{cases} \left(4 - 2\cos\left(\frac{2s\pi}{M}\right) - 2\cos\left(\frac{2t\pi}{N}\right)\right)\hat{p}(s,t) = \widehat{\Delta_i h}(s,t), \ (s,t) \in \hat{\Omega} \setminus \{(0,0)\}, \\ \hat{p}(0,0) = \text{mean}(h). \end{cases}$$

Algorithm to compute the periodic component:

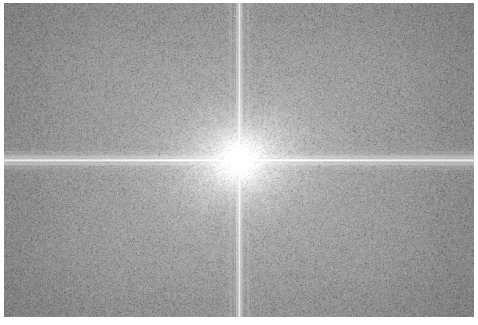
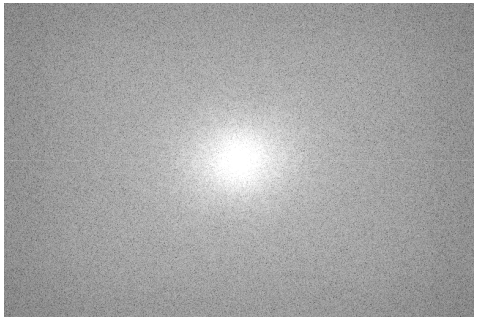
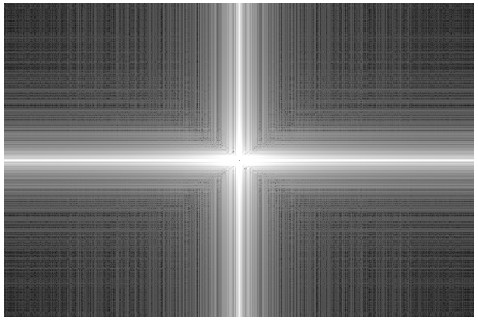
1. Compute $\Delta_i h$ the discrete Laplacian of h.
2. Compute $m = \text{mean}(h)$.
3. Compute $\widehat{\Delta_i h}$ the DFT of $\Delta_i h$ using the forward FFT.
4. Compute the DFT $\hat{p}$ of p defined by

$$\begin{cases} \hat{p}(s,t) = \frac{\widehat{\Delta_i h}((s,t))}{-4+2\cos\left(\frac{2s\pi}{M}\right)+2\cos\left(\frac{2t\pi}{N}\right)} & \text{for } (s,t) \in \hat{\Omega} \setminus \{(0,0)\} \\ \hat{p}(0,0) = m \end{cases}$$

5. Compute p using the backward FFT (if necessary).

Periodic component: effects on the Fourier modulus

- p is “visually close” to h (same Laplacian).

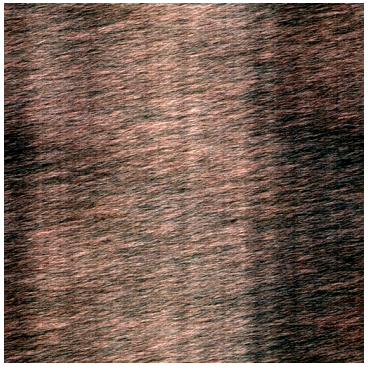
	Image h	Periodic component $p = \mathrm{per}(h)$	Smooth component $s = h - p$ $(+m)$
Images			
Fourier modulus			

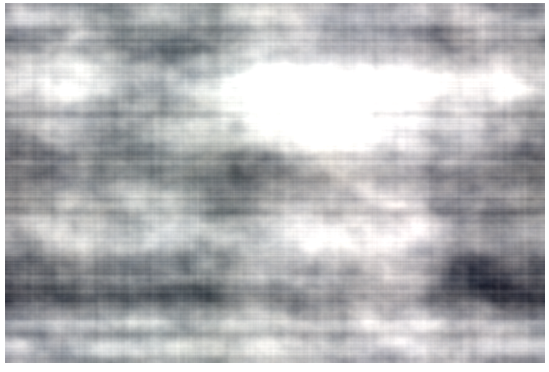
- The application per : $h \mapsto p$ filters out the “cross structure” of the spectrum.

Avoiding artifacts due to non periodicity

Spot *h*

ADSN(*h*)



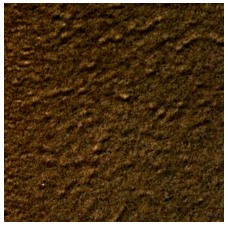


ADSN(*p*)

Synthesizing textures having arbitrary large size

Ad hoc solution: To synthesize a texture larger than the original spot h, one computes an "equivalent spot" $\tilde{h}$:

- Copy $p = \text{per}(h)$ in the center of a constant image equal to the mean of h.
- Normalize the variance.
- Attenuate the transition at the inner border.



Spot h

Equivalent spot $\tilde{h}$

$RPN(h)$

$RPN\left(\tilde{h}\right)$

- Not really rigorous... The envelope changes the covariance.

Properties of the resulting algorithms

- Both algorithms are fast, with the complexity of the fast Fourier transform [$\mathcal{O}(MN \log(MN))$].
- Visual stability: All the realizations obtained from the same input image are visually similar.

Spot h

RPN 1

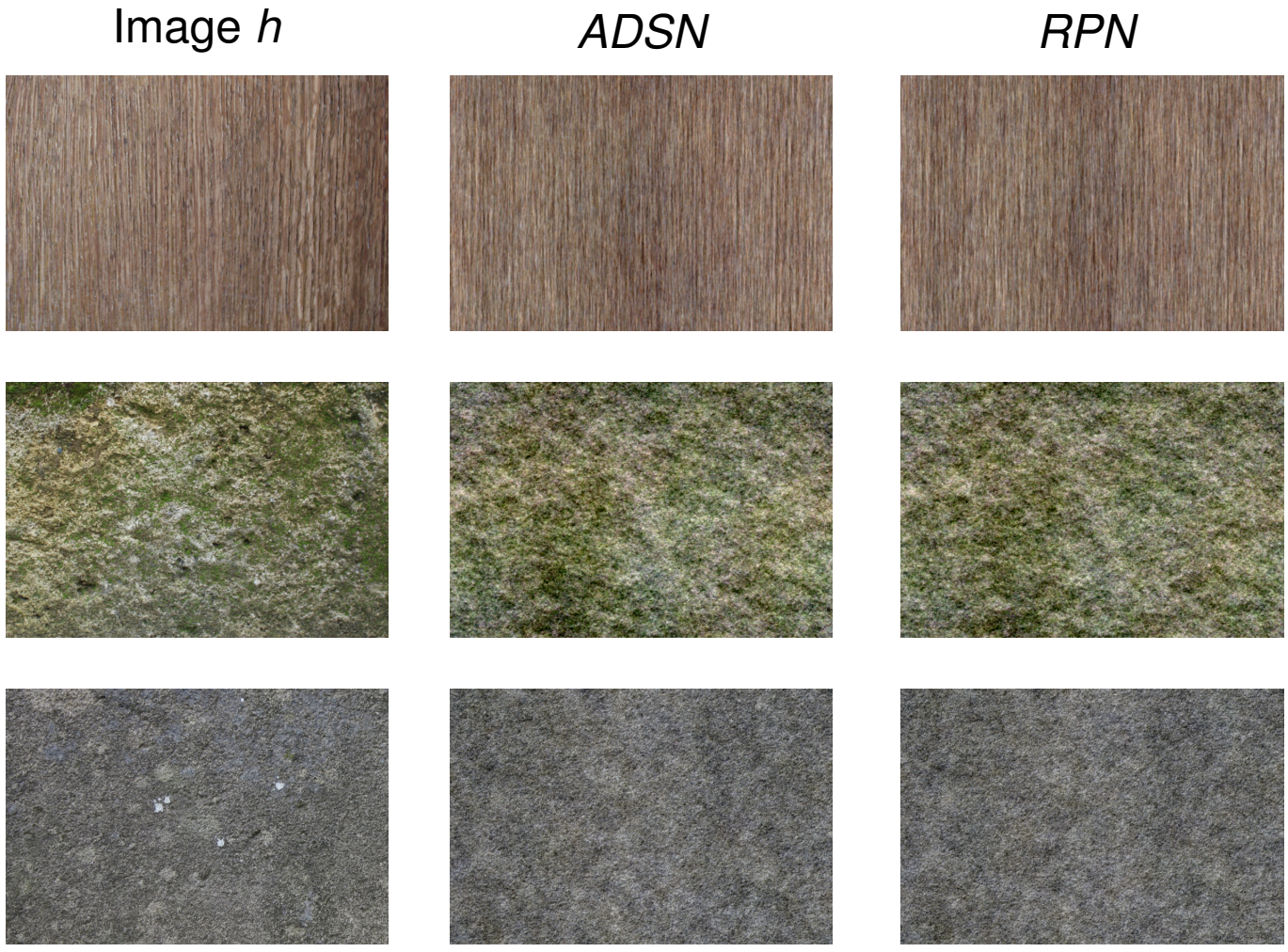
RPN 2

RPN 3

- [ON LINE DEMO]

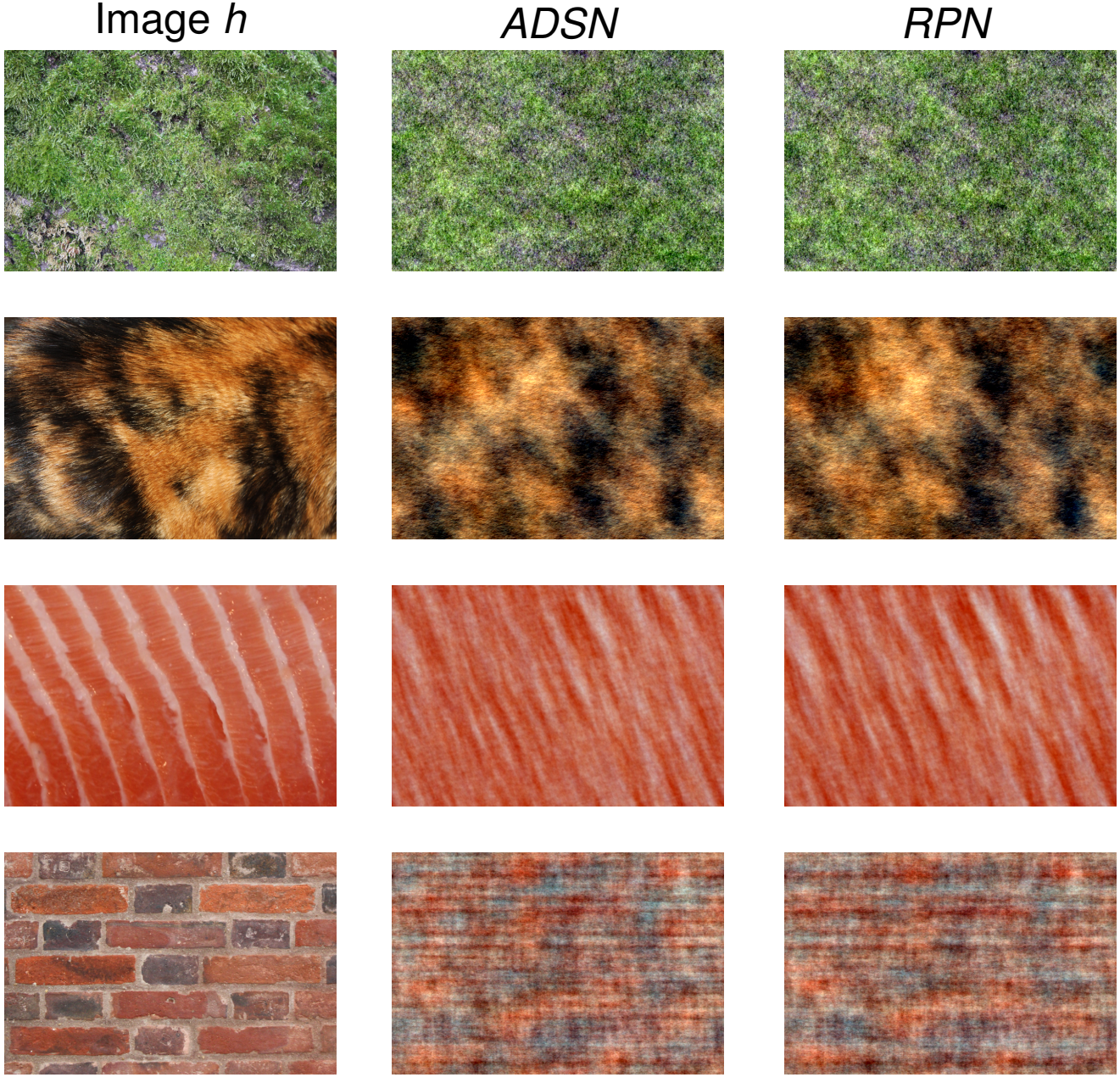
Numerical results: similarity of the textures

- In order to compare both algorithms, the same random phase is used for *ADSN* and *RPN*.



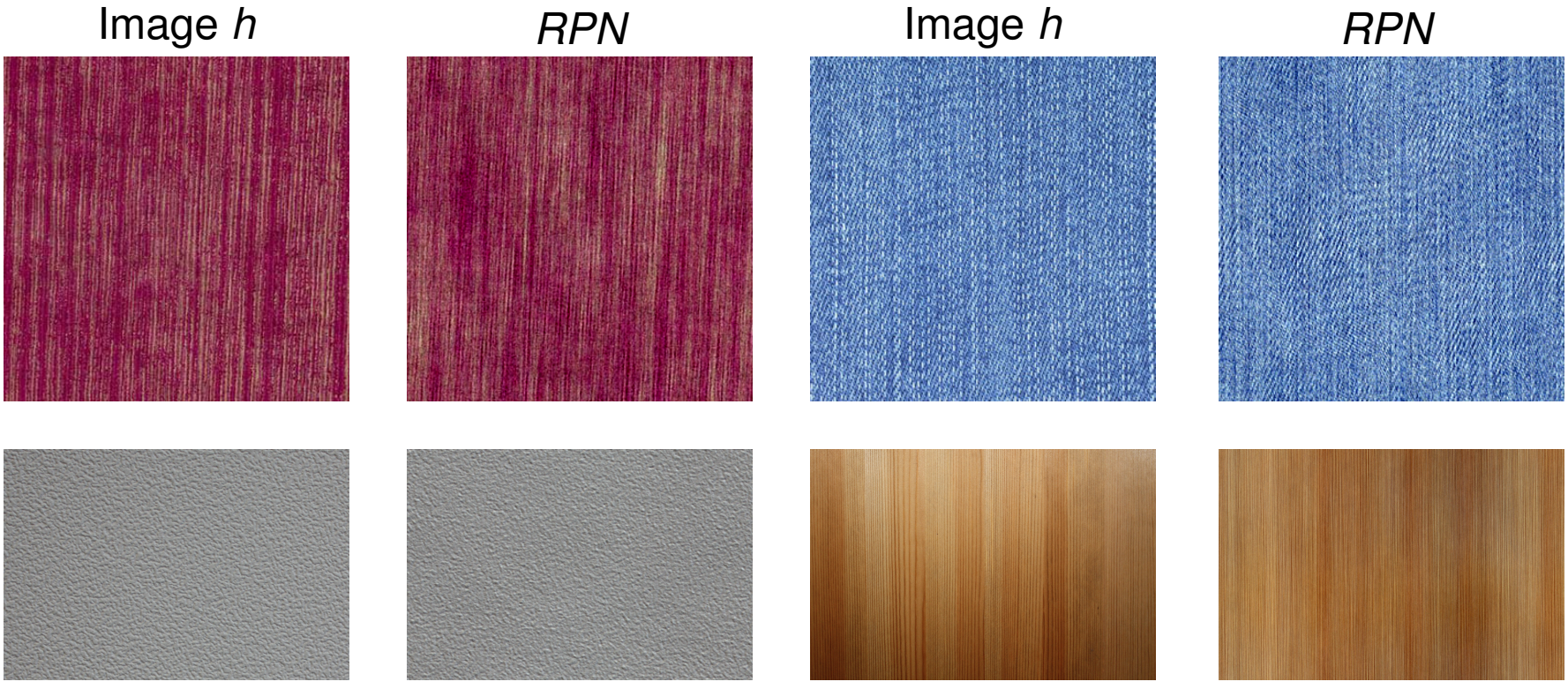
- Both algorithms produce visually similar textures.

Numerical results: non random phase textures



Some other examples of well-reproduced textures...

- We only display the *RPN* result.



- Much more examples of success and failures on the IPOL webpage: `http://www.ipol.im/pub/algo/ggm_random_phase_texture_synthesis/`

Conclusion

Summary:

- *Random phase noise* and *asymptotic discrete spot noise* have been mathematically defined and theoretically compared.
- Both corresponding texture synthesis algorithms are fast, visually stable, and produce visually similar results.
- Both algorithms reproduce relatively well a certain class of textures: the micro-textures.

Limitations:

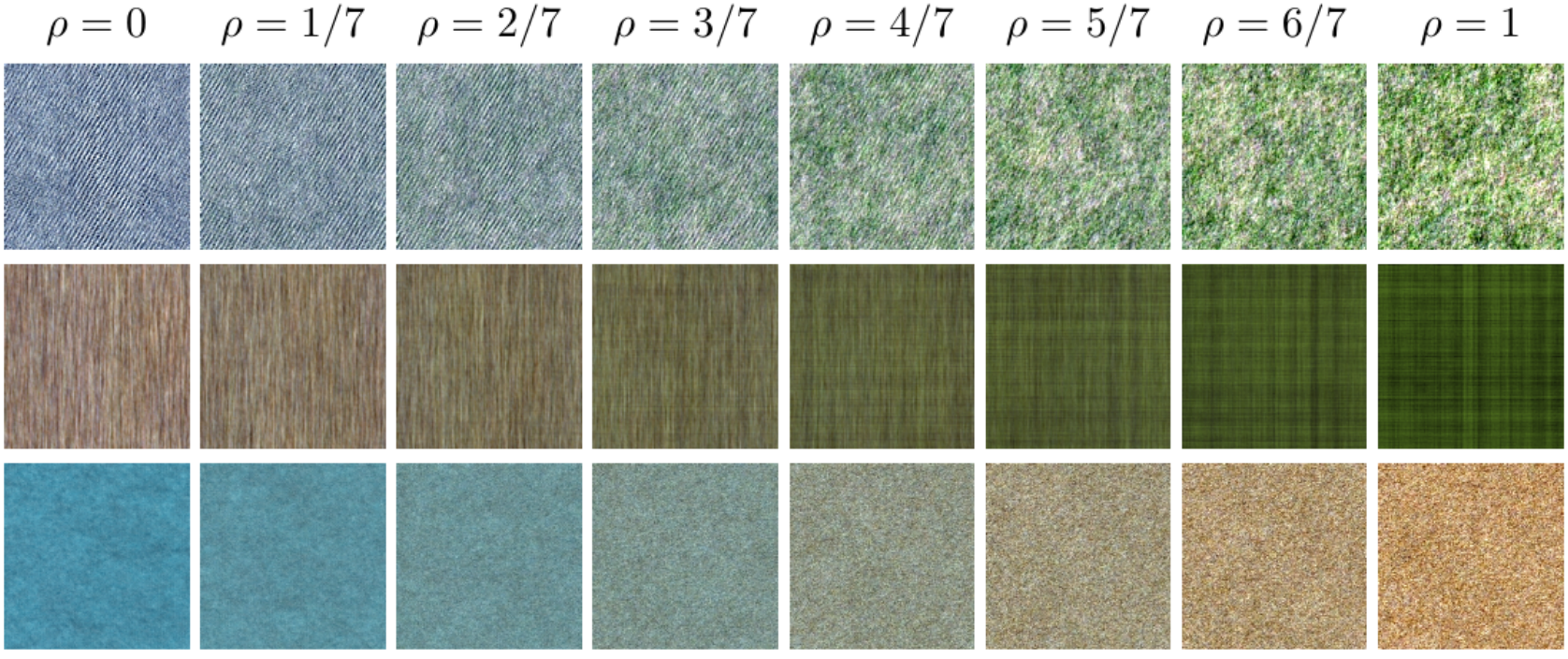
- The models are limited to a restrictive class of textures.
- The algorithms are not robust to non stationarities, perspective effects, ...
- The method is global: the whole texture image has to be computed (in constrast with noise models from computer graphics).

Gaussian textures are a well understood mathematical models: This allows for several extensions:

- Texture mixing
- Noise models from example in graphics
- Texture inpainting

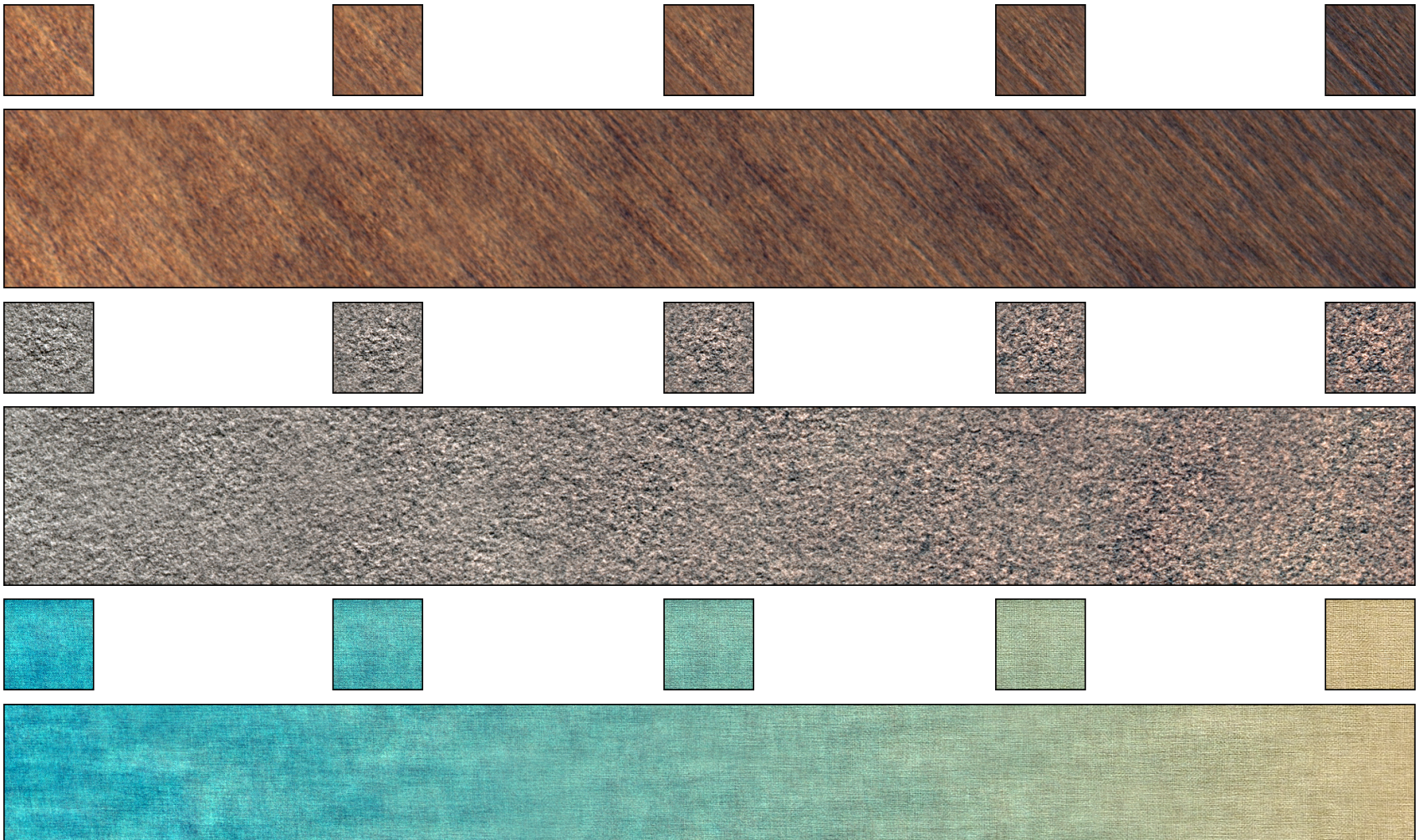
Mixing of Gaussian textures

- Using optimal transport distance, one can define barycenters between Gaussian texture models (ie a shortest covariance path between two covariances).
- This gives a practical and rigorous solution for Gaussian texture mixing [Xia et al, 2011].



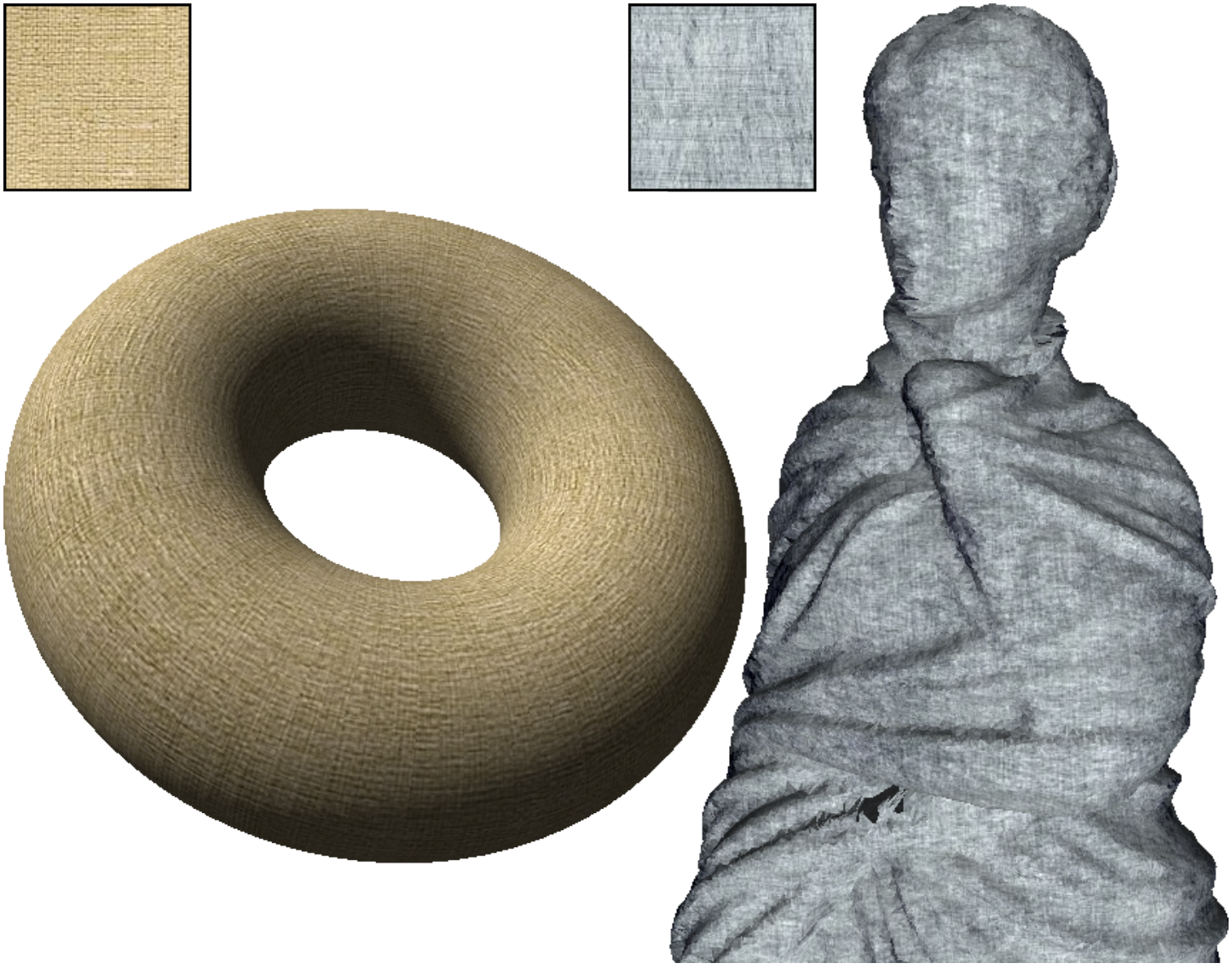
Mixing of Gaussian textures

- Using optimal transport distance, one can define barycenters between Gaussian texture models (ie a shortest covariance path between two covariances).
- This gives a practical and rigorous solution for Gaussian texture mixing [Xia et al, 2011].
- Another example from [Galerne, Leclaire, Moisan, 2017]



Gaussian texture models for 3D graphics

- Gaussian texture models can be extended as procedural noise models for 3D graphics [Galerne, Leclaire, Moisan, 2017]



Outline

Using texture patches

Switch to powerpoint...

Outline

Heeger-Bergen algorithm

References:

- Original paper:
  D. J. Heeger and J. R. Bergen, Pyramid-based texture analysis/synthesis, SIGGRAPH '95, 1995
- Article and demo IPOL [Briand et al. 2014].

Statistical constraints:

- Histogram of colors
- Histogram of "wavelet" coefficients at each scale, more precisely the steerable pyramid transform [Simoncelli et al 1992]

Algorithm:

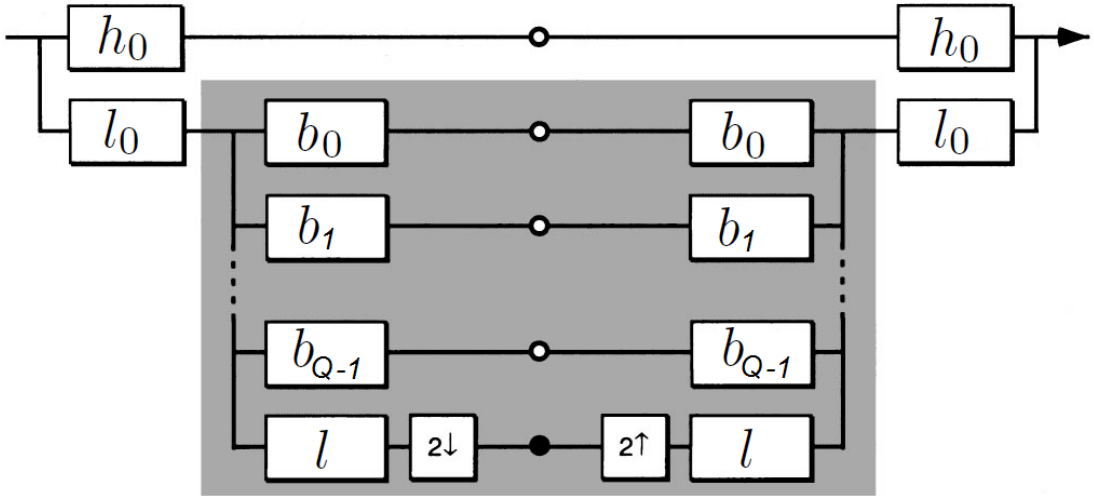
- Alternating projections into the constraints starting from a white noise image

Two main tools:

- Steerable pyramid decomposition and reconstruction
- Histogram matching

Steerable pyramid decomposition

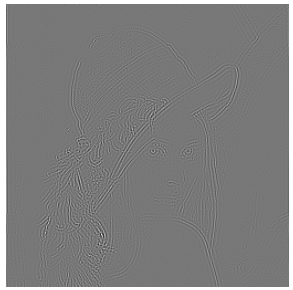
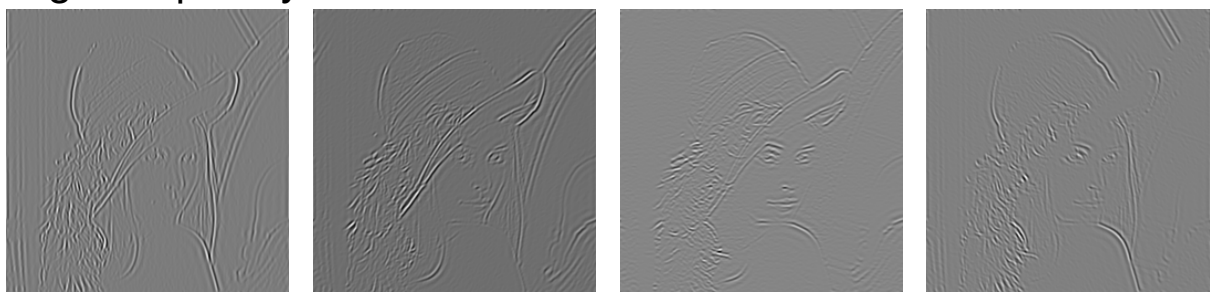
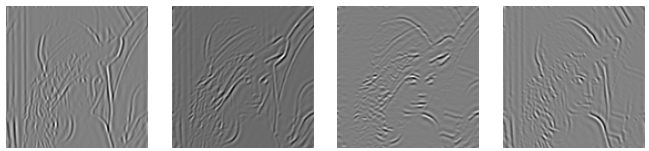
Diagram for the steerable pyramid:



- Different filters:
  - h_0: high-pass filter
  - l_0 and l: low-pass filters
  - $b_0, b_1, \ldots, b_{Q-1}$: Q oriented bandlimited filters
- The left part corresponds to the steerable pyramid image decomposition.
- The right part corresponds to the image reconstruction from the pyramid.
- The dark dot illustrates that the multi-orientation analysis contained in the gray rectangular area is performed on the subsampled images.
- This recursive step is performed until the number of desired pyramid scales P is reached.

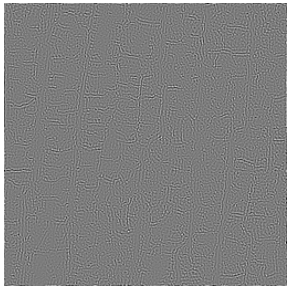
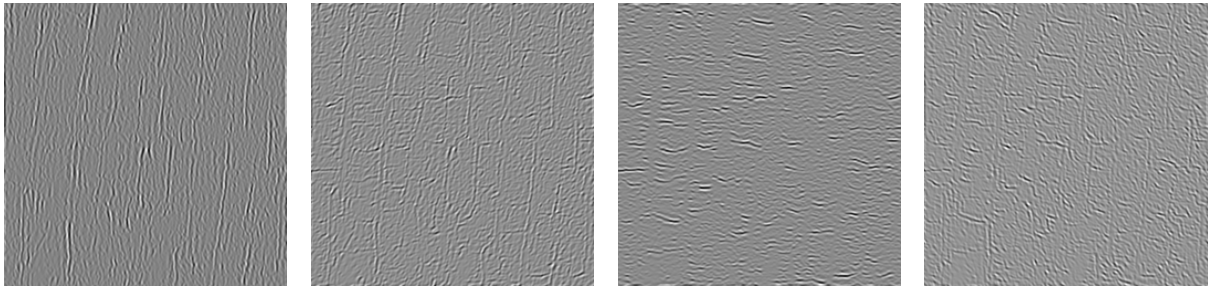
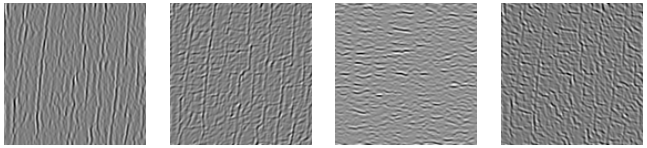
Steerable pyramid decomposition

- Steerable pyramid decomposition of a texture image with two scales and four orientations.

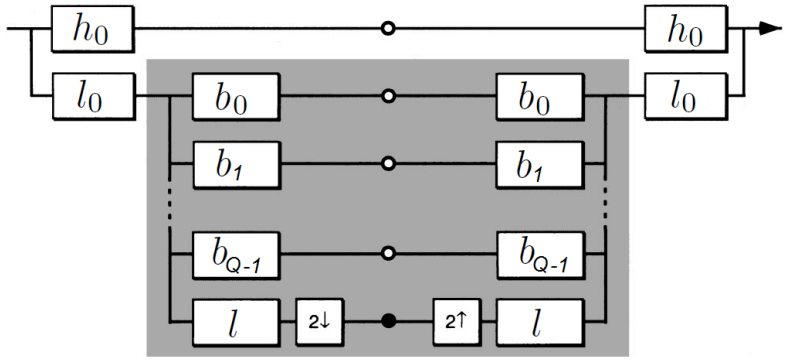
Original image	**Associated steerable pyramid**
	 High frequency residual  1st scale of oriented subbands with angle $\theta = 0$, $\frac{\pi}{4}$, $\frac{\pi}{2}$, and $\frac{3\pi}{4}$  2nd scale of oriented subbands with angle $\theta = 0$, $\frac{\pi}{4}$, $\frac{\pi}{2}$, and $\frac{3\pi}{4}$  Low frequency residual

Steerable pyramid decomposition of a texture

- Steerable pyramid decomposition of a texture image with two scales and four orientations.

Original image	Associated steerable pyramid
	 High frequency residual  1st scale of oriented subbands with angle $\theta = 0$, $\frac{\pi}{4}$, $\frac{\pi}{2}$, and $\frac{3\pi}{4}$  2nd scale of oriented subbands with angle $\theta = 0$, $\frac{\pi}{4}$, $\frac{\pi}{2}$, and $\frac{3\pi}{4}$  Low frequency residual

Steerable pyramid decomposition



- Filters are defined analytically in polar coordinates in the Fourier domain e.g.

$$L(r,\theta) = L(r) = \begin{cases} 1 & \text{if } r \leqslant \frac{\pi}{4}, \\ \cos(\frac{\pi}{2}\log_2(\frac{4r}{\pi})) & \text{if } \frac{\pi}{4} \leqslant r \leqslant \frac{\pi}{2}, \\ 0 & \text{if } r \geqslant \frac{\pi}{2}, \end{cases}$$

- The decomposition corresponds to a (soft) paving of the Fourier domain (similar to a filter bank of Gabor filters).

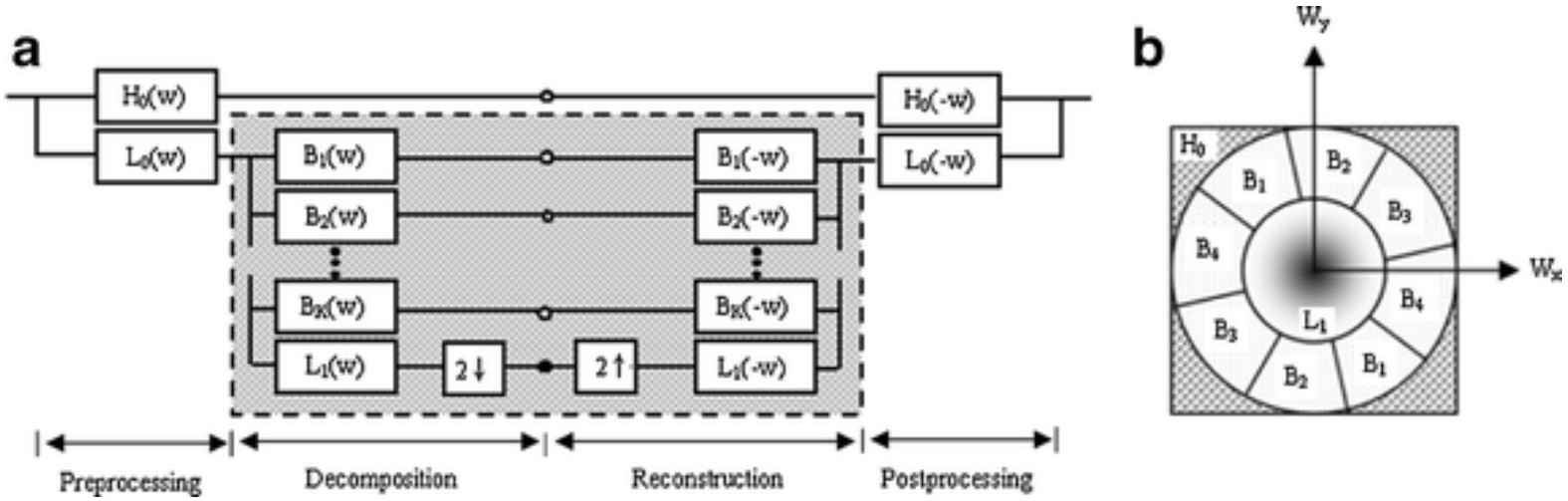


Image credits [Wang et al 2014]

Steerable pyramid reconstruction

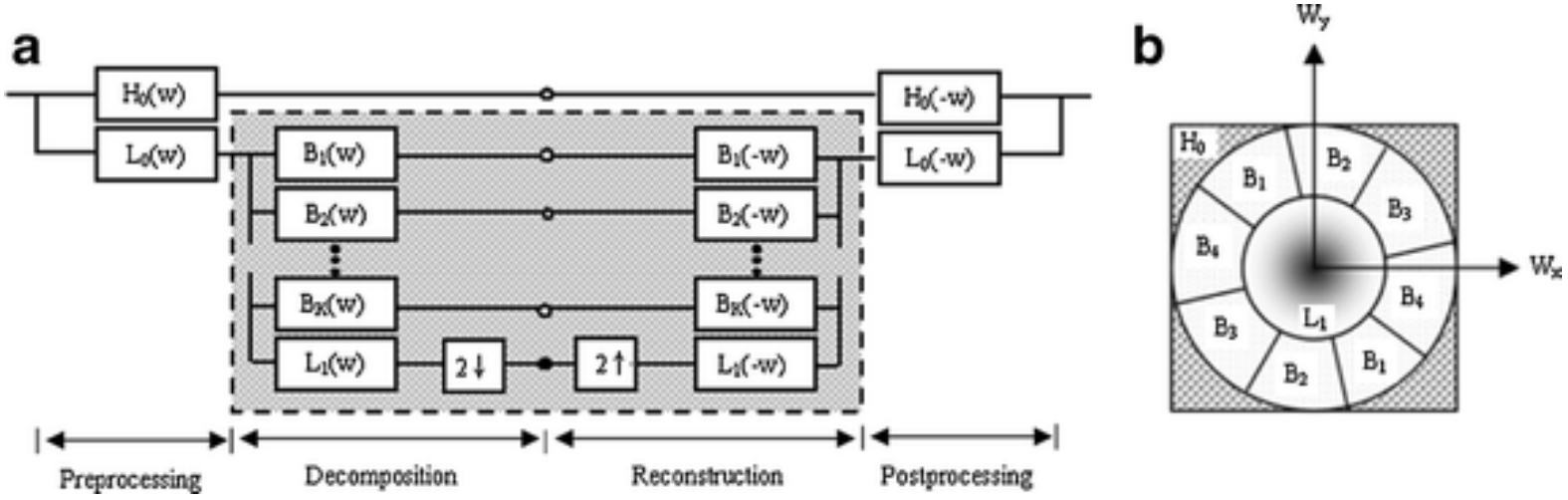


Image credits [Wang et al 2014]

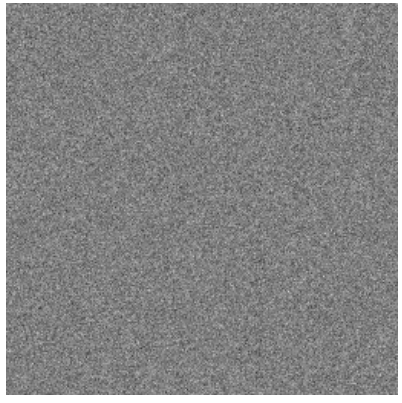
- The steerable filters are designed so that each stage of the diagram has a flat system response.

$$H_0(r,\theta)^2 + L_0(r,\theta)^2 = 1 \quad \text{and} \quad \sum_{k=0}^{Q-1} B_q(r,\theta)^2 + L(r,\theta)^2 = 1.$$

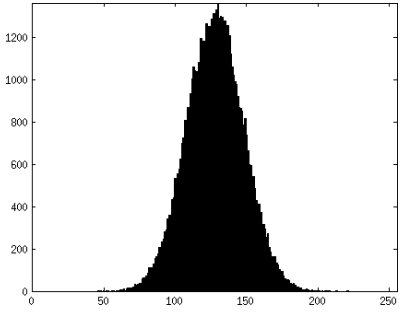
- Denote by A the matrix of the pyramid decomposition operator, then the matrix of the reconstruction operator is simply the transpose A^T.
- But the flat response ensures that $A^T A = Id$, so that A^T is also the pseudo-inverse of A: $A^\dagger = \left(A^T A\right)^{-1} A^T = A^T$.
- This is important here since the reconstruction operator will be applied to pyramids that are outside the range A.

Histogram matching

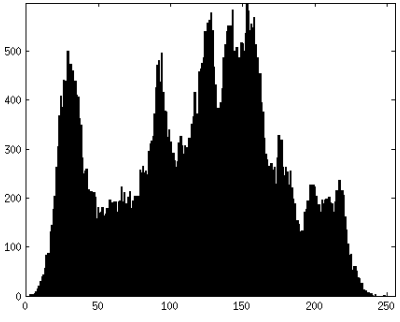
- Example of histogram matching: The histogram of a Gaussian white noise is matched with the histogram of the Lena image.

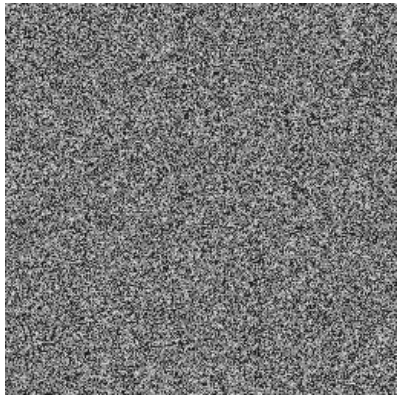


Input image

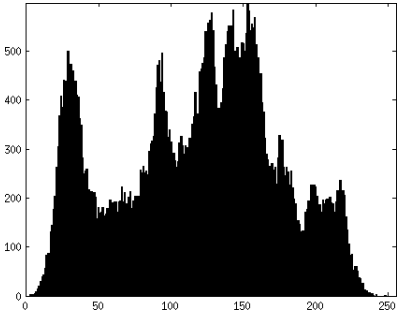


Reference image



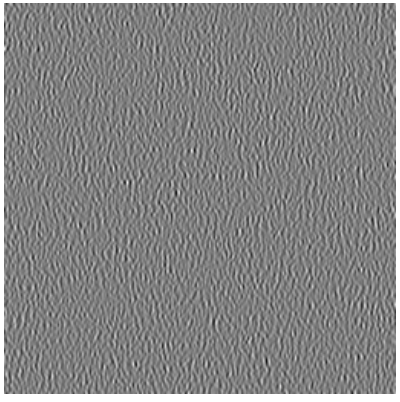


Output image

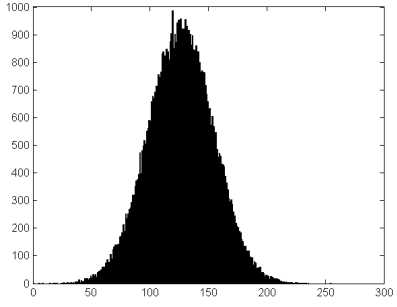


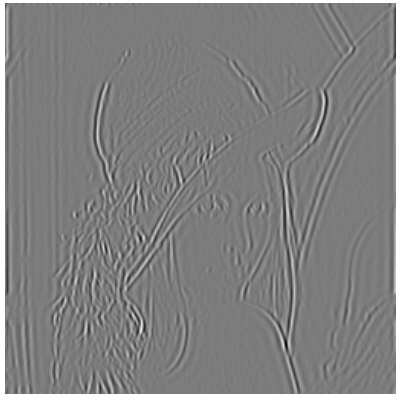
Histogram matching

- Example of histogram matching of a pyramid image: The histogram of an oriented pyramid image of a Gaussian white noise is matched with the corresponding image in Lena's pyramid

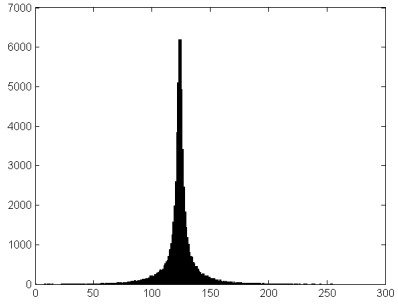


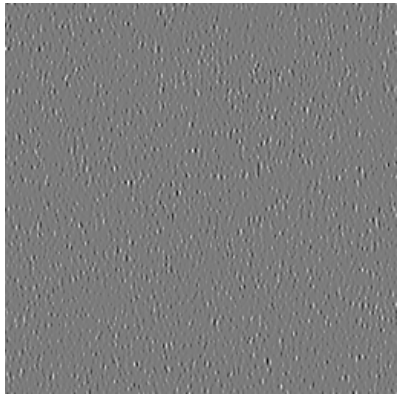
Input image



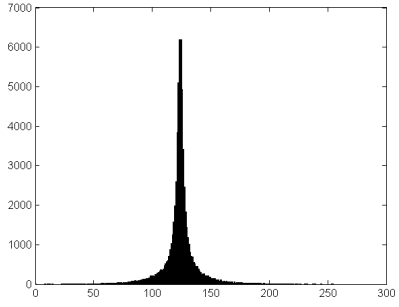


Reference image





Output image



Histogram matching

Algorithm 1: Histogram matching

Input : Input image u, reference image v (both images have size $M \times N$)
Output: Image u having the same histogram as v (the input u is lost)

1. Define $L = MN$ and describe the image as arrays of length L (*e.g.* by reading them line by line).
2. **Sort the reference image v:**
3. Determine the permutation τ such that $v_{\tau(1)} \leq v_{\tau(2)} \leq \cdots \leq v_{\tau(L)}$.
4. **Sort the input image u:**
5. Determine the permutation σ such that $u_{\sigma(1)} \leq u_{\sigma(2)} \leq \cdots \leq u_{\sigma(L)}$.
6. **Match the histogram of u:**
7. **for** rank $k = 1$ **to** L **do**
8. $\quad u_{\sigma(k)} \leftarrow v_{\tau(k)}$ (the k-th pixel of u takes the gray-value of the k-th pixel of v).
9. **end**

$$\forall k \in \Omega, \quad u_{\sigma(k)} \leftarrow v_{\tau(k)} \quad \Longleftrightarrow \quad \forall j \in \Omega, \quad u_j \leftarrow v_{\tau(\sigma^{-1}(j))}$$

- **Optimal assignment:** $\tau \circ \sigma^{-1}$ corresponds to the optimal transport plan between the two discrete measures $\sum_{k\in\Omega} \delta_{u_k}$ and $\sum_{k\in\Omega} \delta_{v_k}$.

$$\tau \circ \sigma^{-1} = \operatorname{argmin}_{\sigma\in\Sigma_L} \sum_{k\in\Omega} |u_k - v_{\sigma(k)}|^2$$

Heeger and Bergen algorithm

Algorithm 2: Heeger-Bergen texture synthesis algorithm for grayscale images (without extension)

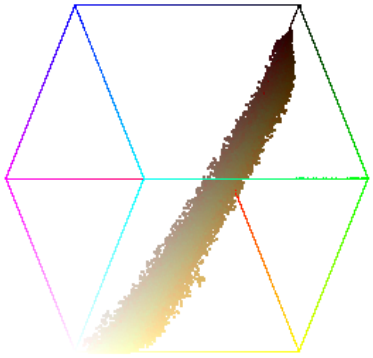
Input : Number of scales P, number of orientations Q, texture image u of size $M \times N$ such that M and N are multiples of 2^P, number of iterations N_{iter}

Output: Texture image v of size $M \times N$

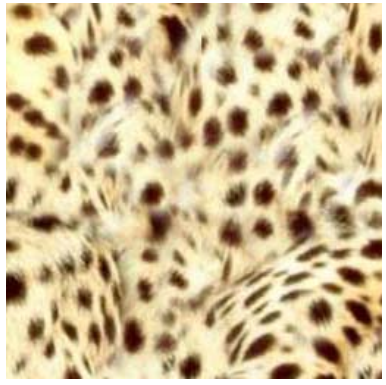
1. **Input analysis:**
2. Compute and store the steerable pyramid with P scales and Q orientations of the input texture u.
3. **Output synthesis:**
4. Initialize v with a Gaussian white noise.
5. Match the gray-level histogram of v with the gray-level histogram of u.
6. **for** iteration $i = 1$ **to** N_{iter} **do**
7. Compute the steerable pyramid of v.
8. For each of the $PQ + 2$ images of this pyramid, apply histogram matching with the corresponding image of the pyramid of u.
9. Apply the image reconstruction algorithm to this new histogram-matched pyramid and store the obtained image in v.
10. Match the gray-level histogram of v with the gray-level histogram of the input u.
11. **end**
12. Return v.

Heeger and Bergen algorithm for color textures

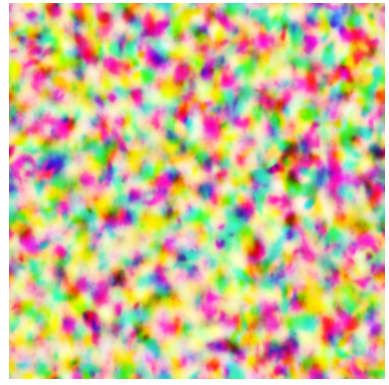
- The Heeger-Bergen algorithm is designed for grayscale images (one channel).
- The principal limitation comes from the histogram matching algorithm.
- Applying the algorithm to each channel of an RGB image does not work: Indeed the output color channels are independent !



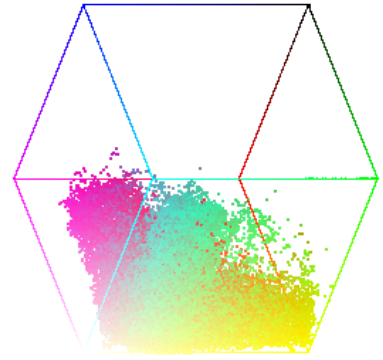
Input color cube



Input texture



Output texture



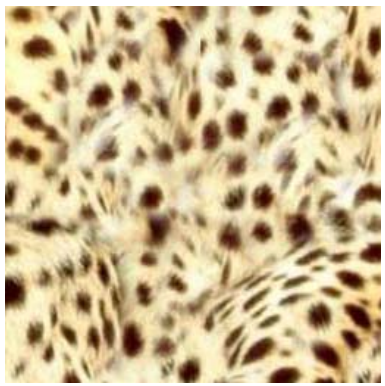
Output color cube

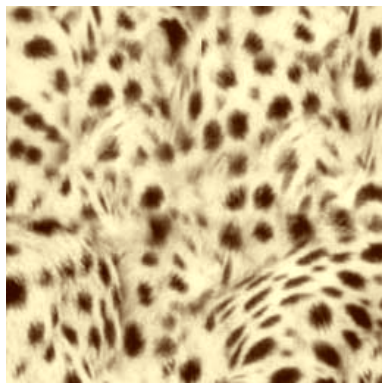
- RGB cube visualizatiton from the IPOL paper and demo [Lisani et al 2011]
- Solution: Change the color space so that the independence of the color channels is acceptable.

Heeger and Bergen algorithm for color textures

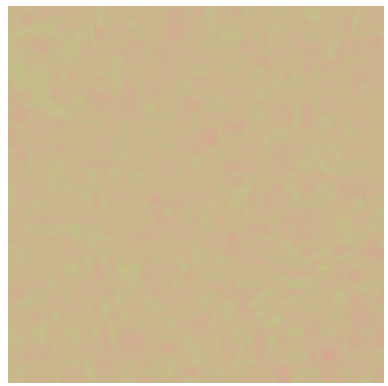
PCA color space of an image:

- Compute the PCA of the colors seen as a point cloud in $\mathbb{R}^3$.


Color texture

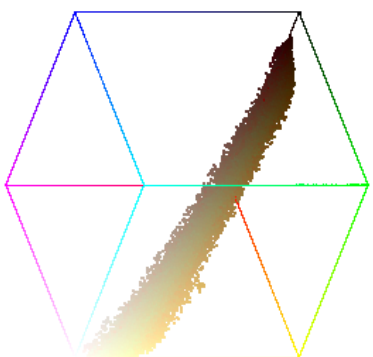

1st PC

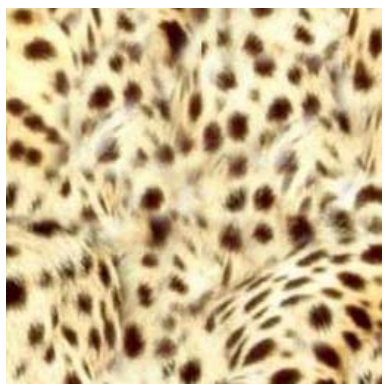

2nd PC

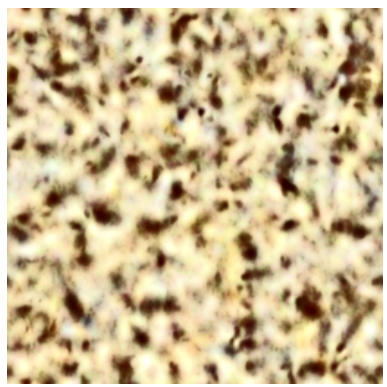

3rd PC

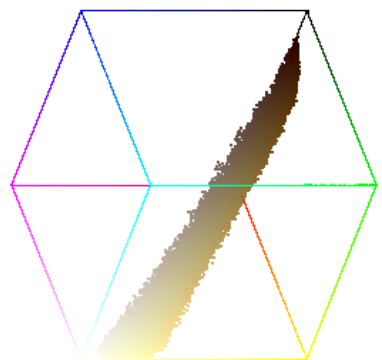
- In general, the dynamic of the texture is mainly contained in the first principal component.

Independent synthesis in PCA color space:


Input color cube


Input texture


Output texture


Output color cube

Heeger and Bergen algorithm: Results

Heeger and Bergen algorithm: Parameters

Several parameters:

- The number of iterations N_{iter} of the synthesis algorithm.
- The number of scales P of the steerable pyramid decomposition.
- The number of orientations Q of the steerable pyramid decomposition (not critical when higher than 4).
- The edge handling option (not discussed here...).

Heeger and Bergen algorithm: Parameters

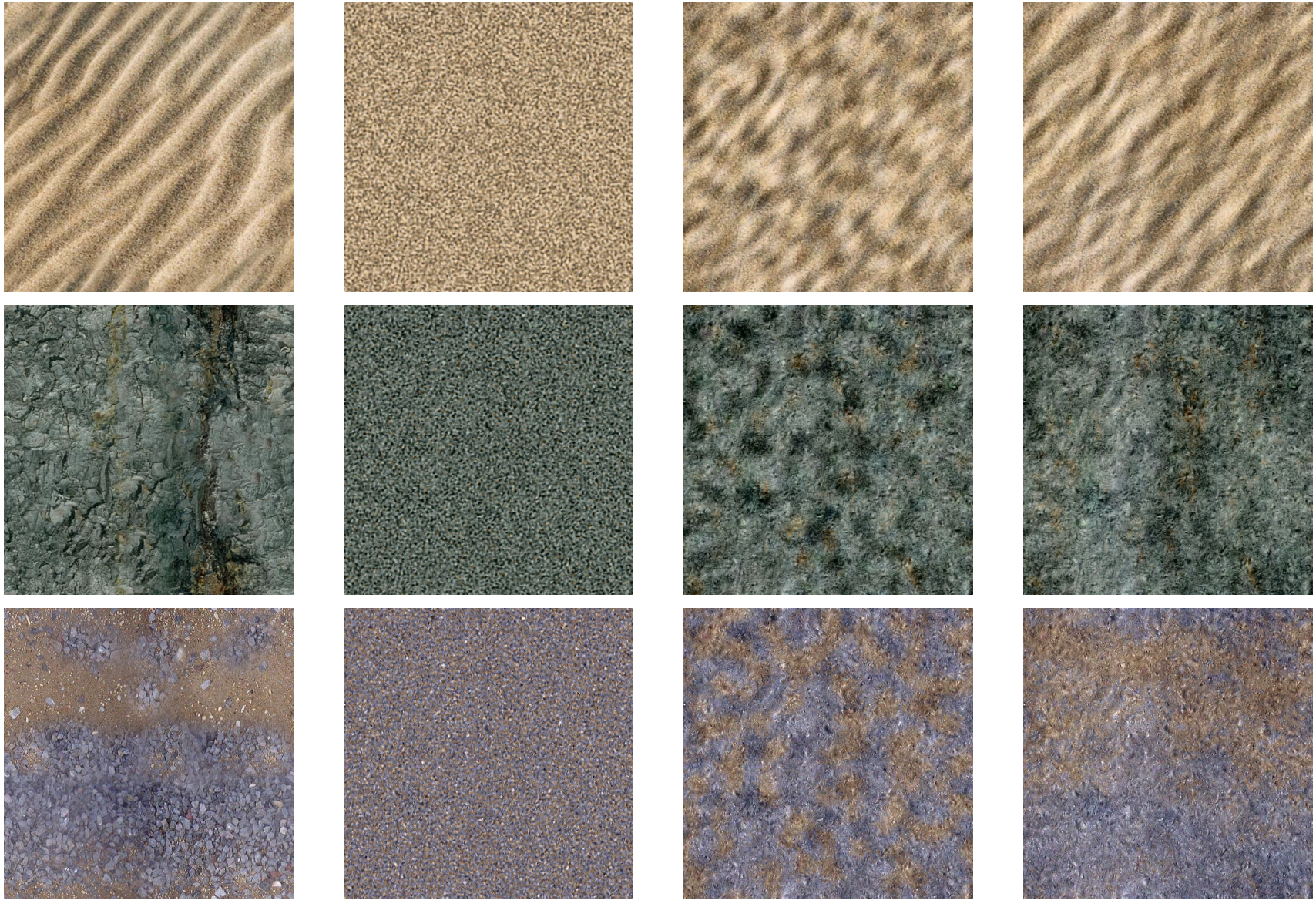
Influence of the number of iterations: From left to right: original image, result with $N_{iter} = 1$, 5 and 10.

- Generally 5 iterations is enough for “visual convergence”

Heeger and Bergen algorithm: Parameters

Influence of the number of scales P: Result with $P = 1$, 4 and 8.

- Generally the maximal number avoid a blotchy artefact due to incoherent low frequency residual.



Portilla and Simoncelli algorithm

References:

J. Portilla and E. Simoncelli, *A parametric texture model based on joint statistics of complex wavelet coefficients*, IJCV, 40 (2000)

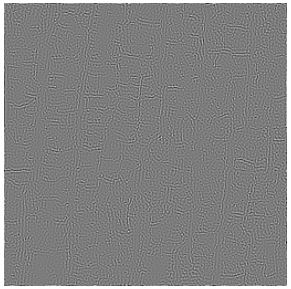
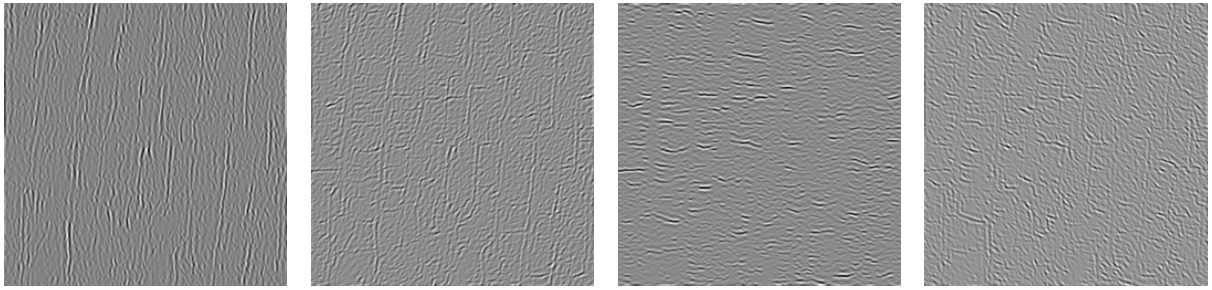
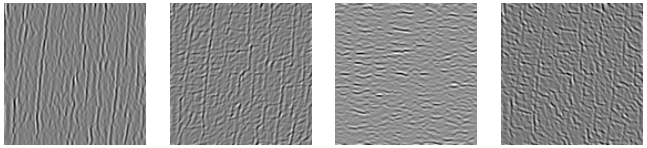
A must read paper !

General ideas:

- In Heeger-Bergen the pyramid images are treated independently, but they are clearly highly correlated.
- Introduce joint statistics within neighborhood pixels, with neighborhood on spatial grid but also across scales.
- Select a small set of statistics (moments) and show that each of them is important.
- The texture synthesis is a gradient descent algorithm starting from a white noise image.

Steerable pyramid decomposition of a texture

- Steerable pyramid decomposition of a texture image with two scales and four orientations.

Original image	Associated steerable pyramid
	 High frequency residual  1st scale of oriented subbands with angle $\theta = 0$, $\frac{\pi}{4}$, $\frac{\pi}{2}$, and $\frac{3\pi}{4}$  2nd scale of oriented subbands with angle $\theta = 0$, $\frac{\pi}{4}$, $\frac{\pi}{2}$, and $\frac{3\pi}{4}$  Low frequency residual

Portilla and Simoncelli

From [Portilla and Simoncelli 2000].

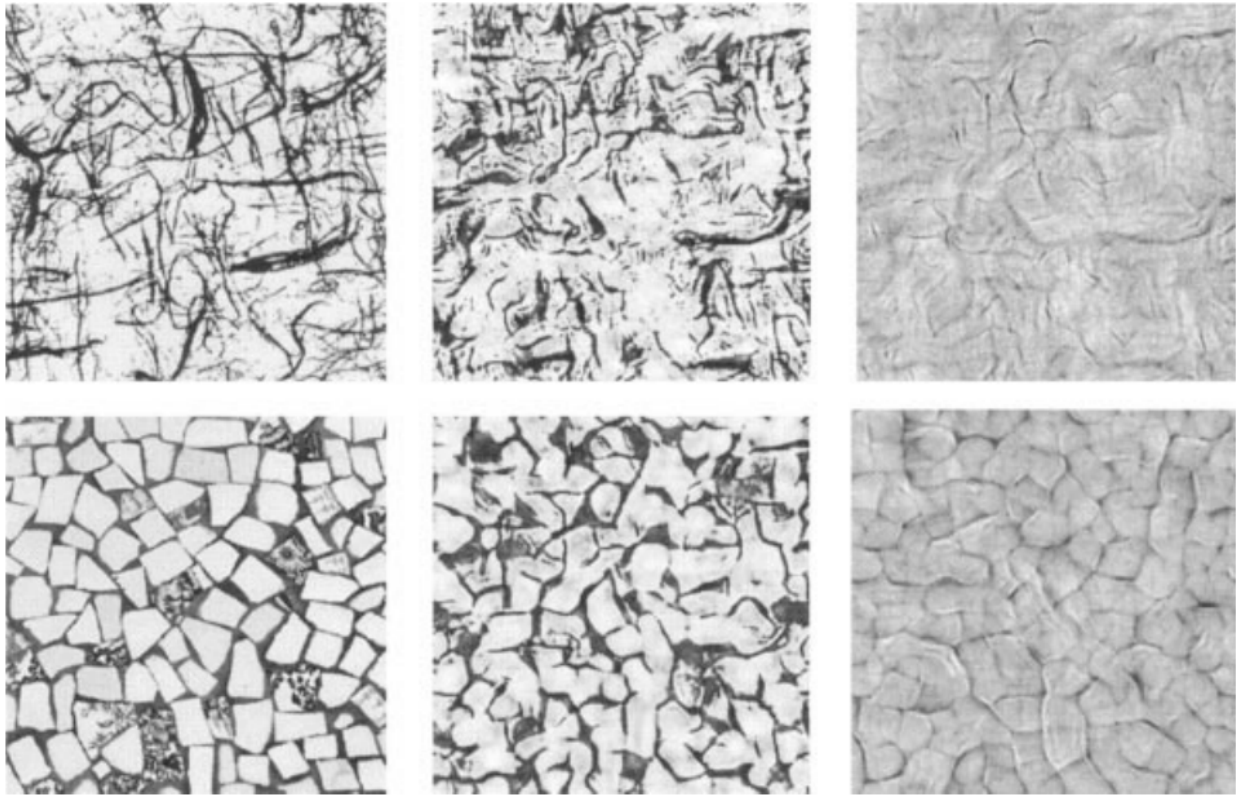


Figure 3. Necessity of marginal constraints. Left column: original texture images. Middle: Images synthesized using full constraint set. Right: Images synthesized using all but the marginal constraints.

Portilla and Simoncelli

From [Portilla and Simoncelli 2000].

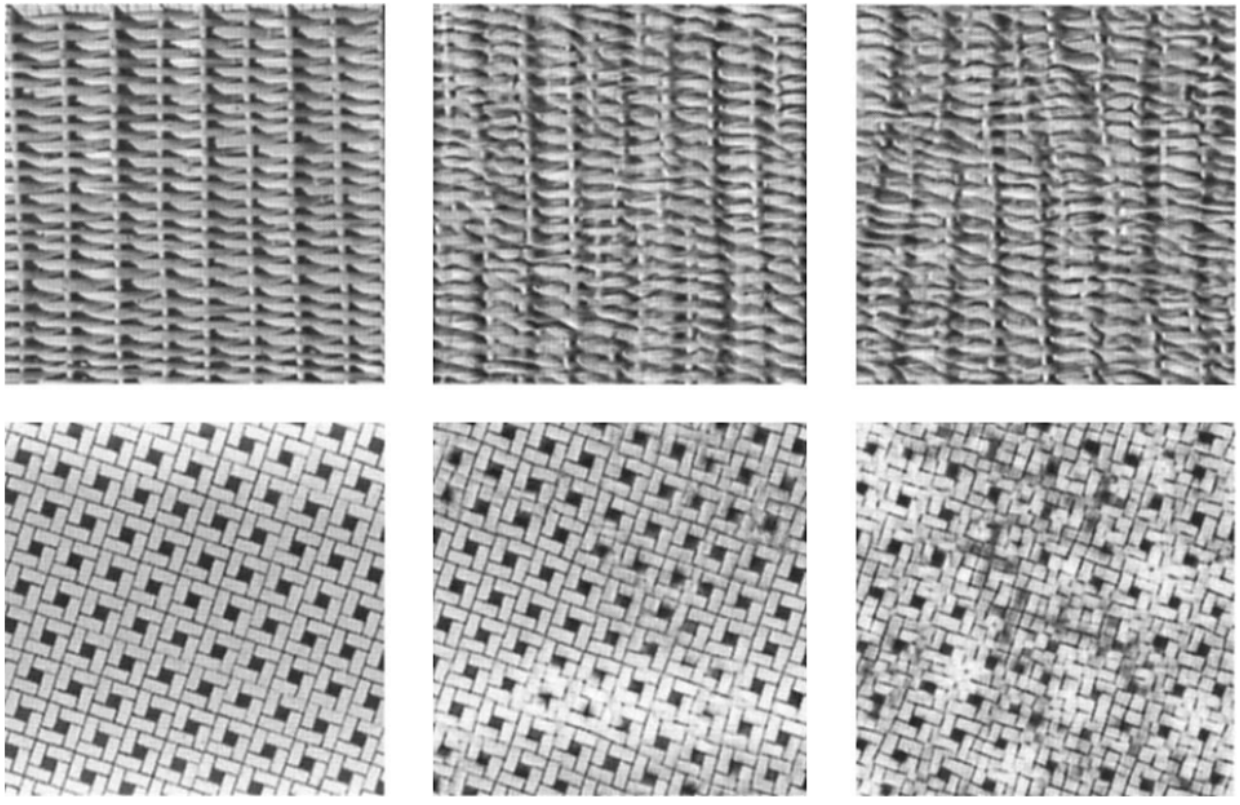


Figure 4. Necessity of raw autocorrelation constraints. Left column: original texture images. Middle: Images synthesized using full constraint set. Right: Images synthesized using all but the autocorrelation constraints.

Portilla and Simoncelli

From [Portilla and Simoncelli 2000].

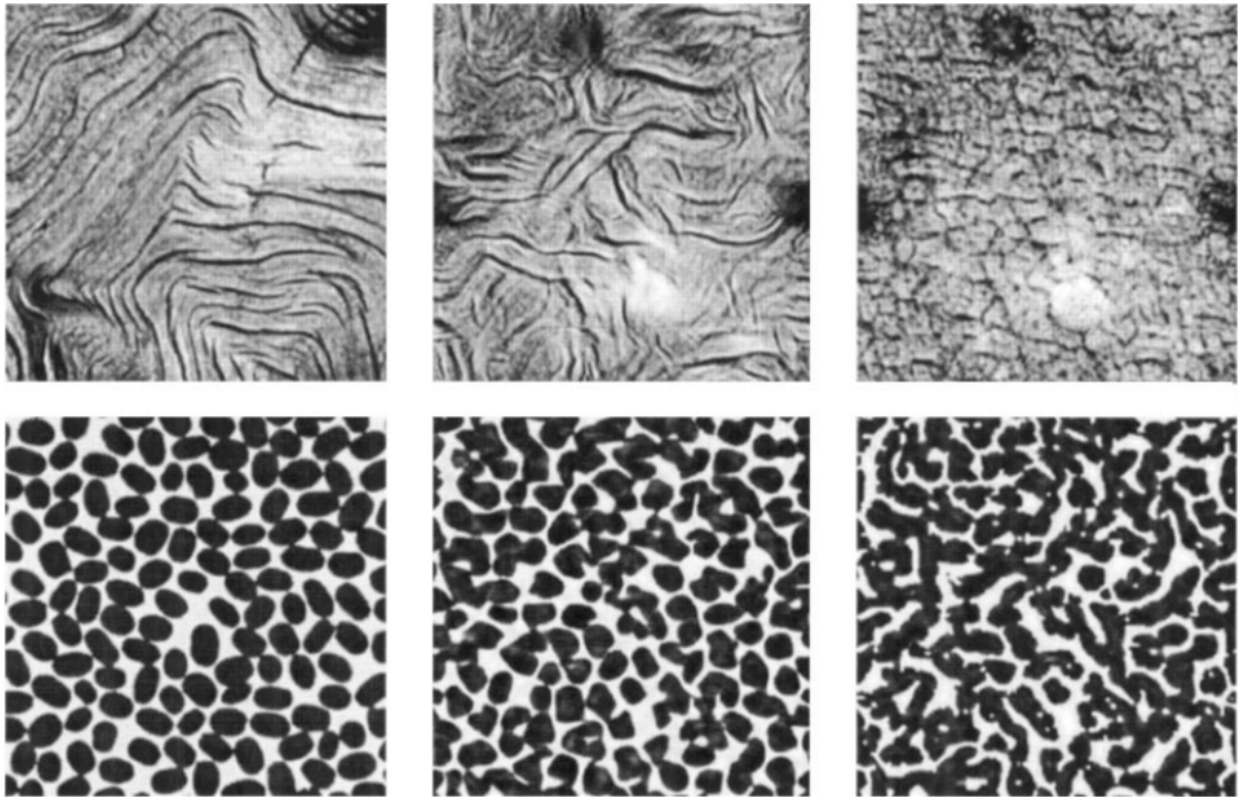


Figure 6. Necessity of magnitude correlation constraints. Left column: original texture images. Middle: Images synthesized using full constraint set. Right: Images synthesized using all but the magnitude auto- and cross-correlation constraints.

Portilla and Simoncelli

From [Portilla and Simoncelli 2000].

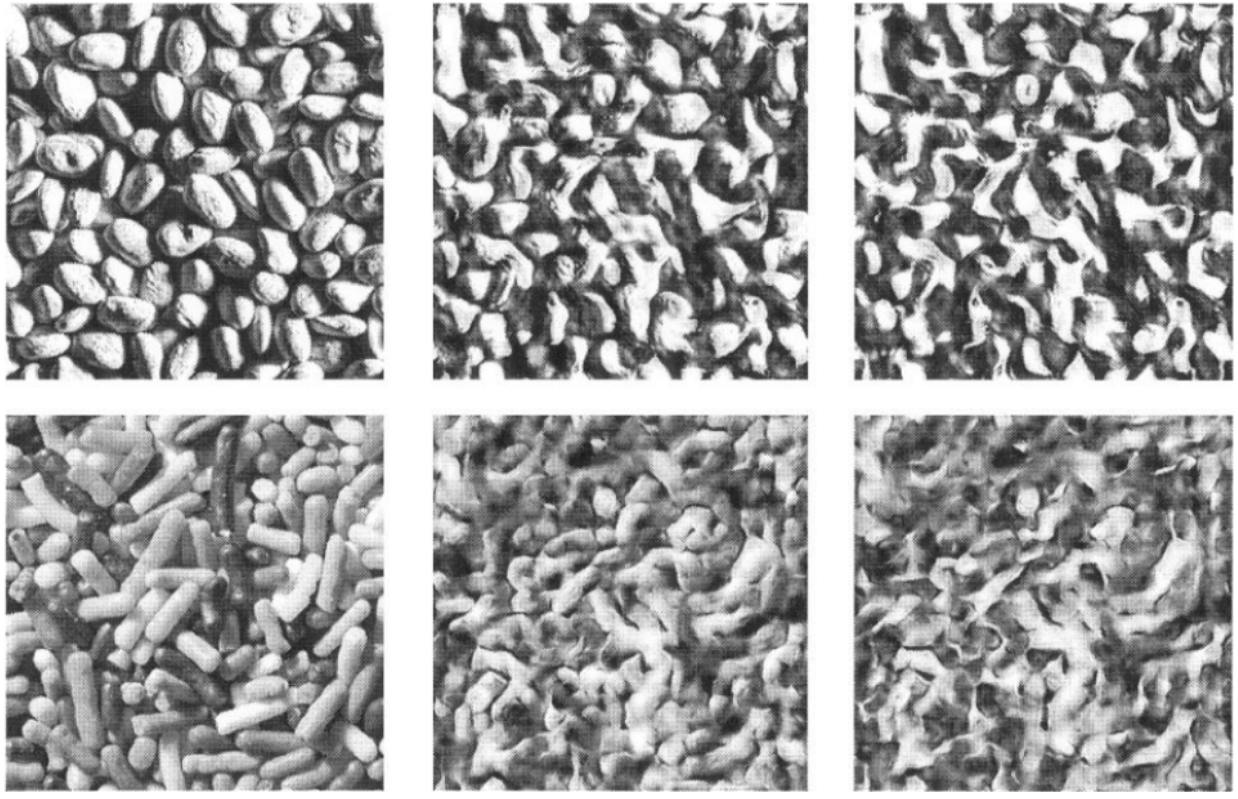


Figure 8. Necessity of cross-scale phase constraints. Left column: original texture images. Middle: Images synthesized using full constraint set. Right: Images synthesized using all but the cross-scale phase constraints.

Portilla and Simoncelli

- Very impressive results.
- Stayed state of the art for statistical texture synthesis for a long time.
- Patch-based method were introduced at the same period.
- Here the textures are synthesized with only 710 parameters.

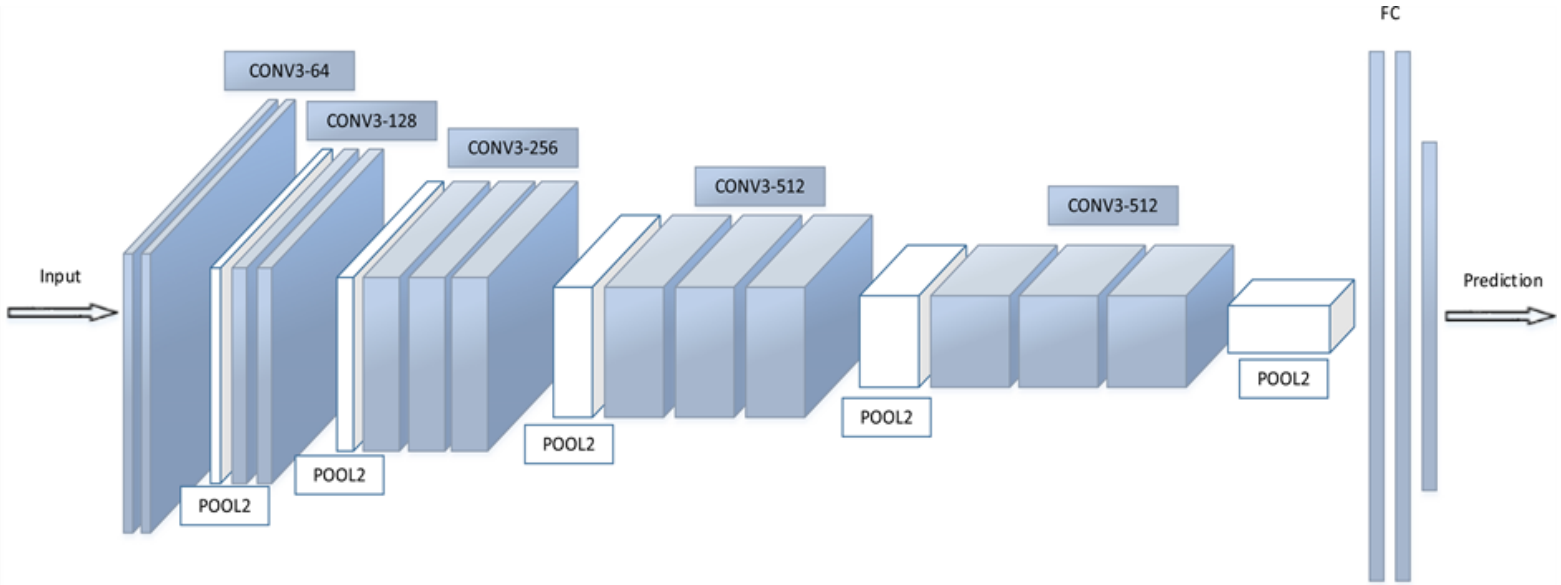
Outline

Gatys et al algorithm

References: L. Gatys, A. S. Ecker, and M. Bethge, *Texture synthesis using convolutional neural networks*, in Advances in Neural Information Processing Systems, 2015

Convolutional Neural Networks (CNN)

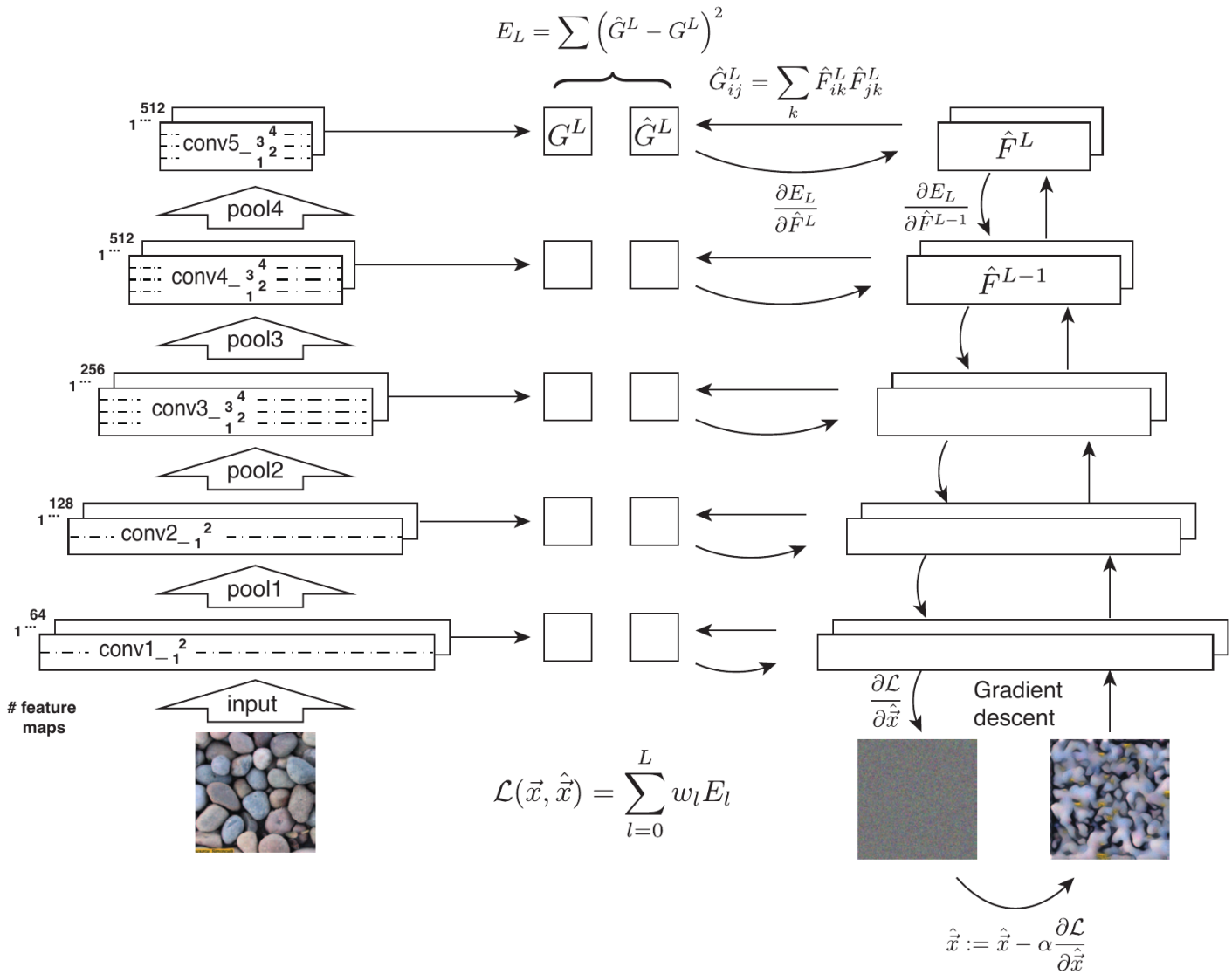
- Main idea: Use the feature layers of a trained deep CNN, namely VGG 19 [Simonyan and Zisserman], as statistics.
- VGG 19 [Simonyan and Zisserman] was trained for image classification.
- It only uses 3×3 convolution kernels followed by RELU (= positive part) and max-pooling.



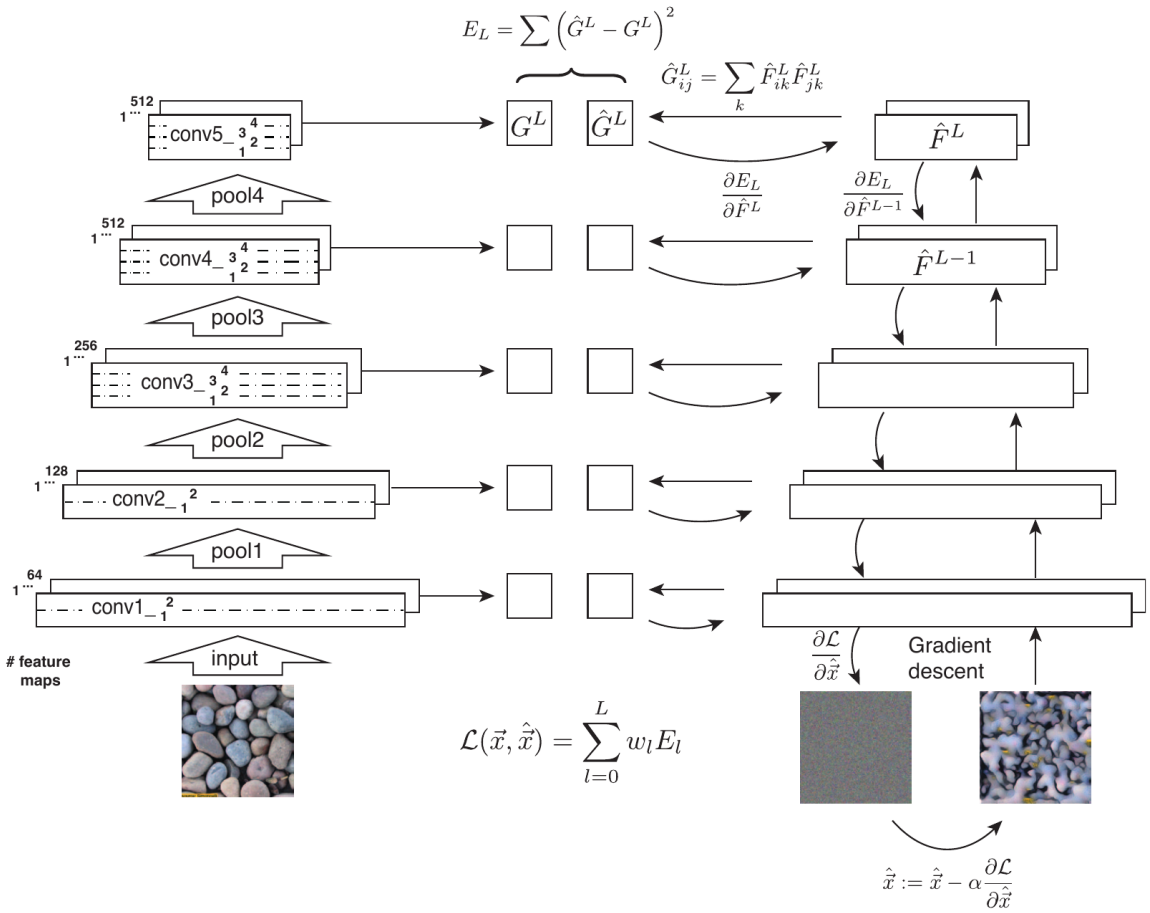
- For texture analysis, we only use the “pool” layers.
- We do not use the last “fully connected” layers that perform classification.
- VGG 19 is understood as a multiscale nonlinear transform adapted to natural images.

Gatys et al algorithm

- Each feature layer is spatially averaged and the Gram matrix is formed for each layer.
- Texture synthesis consists in minimizing the sum of the squared Frobenius norm between Gram matrices.

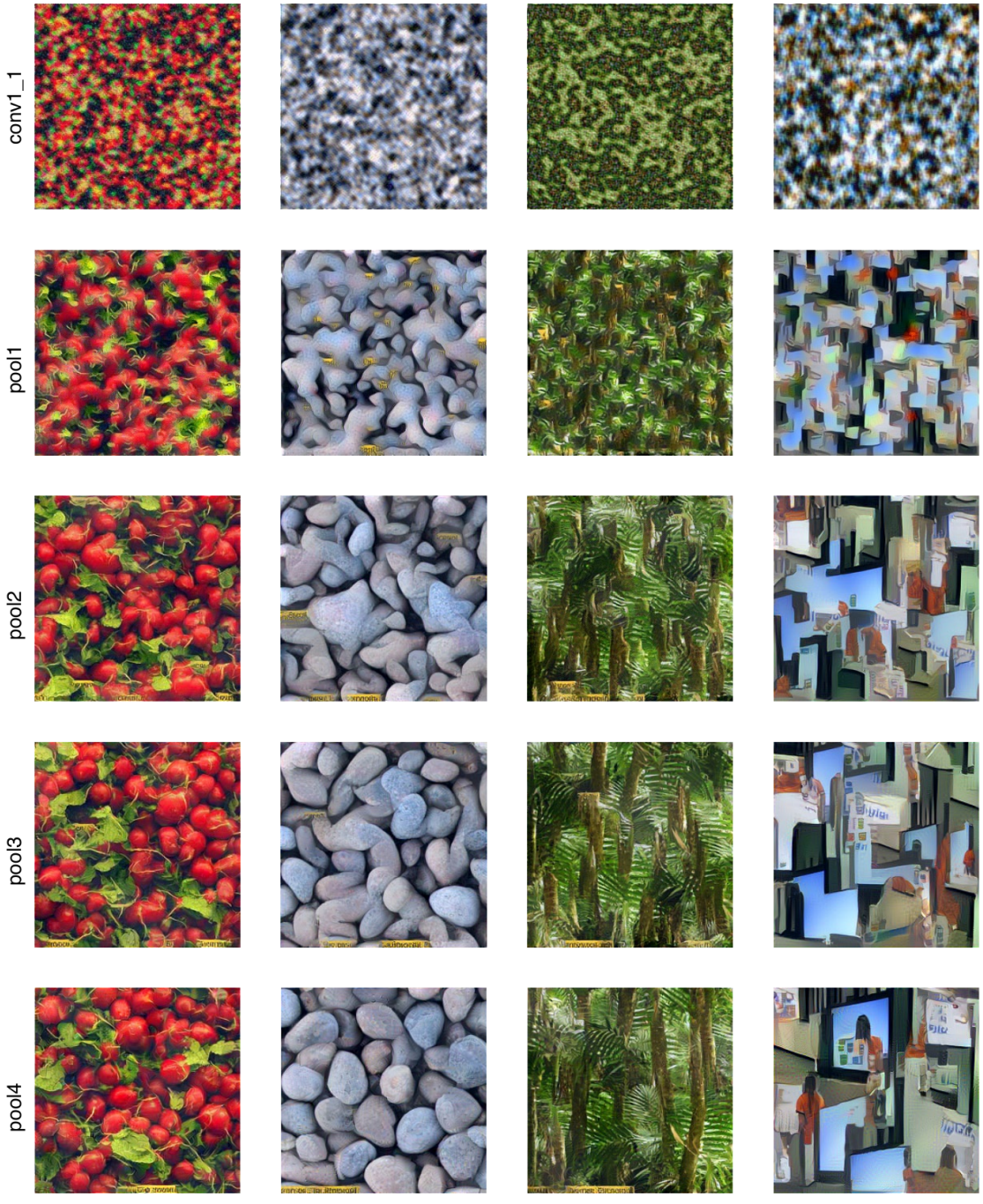


Gatys et al algorithm

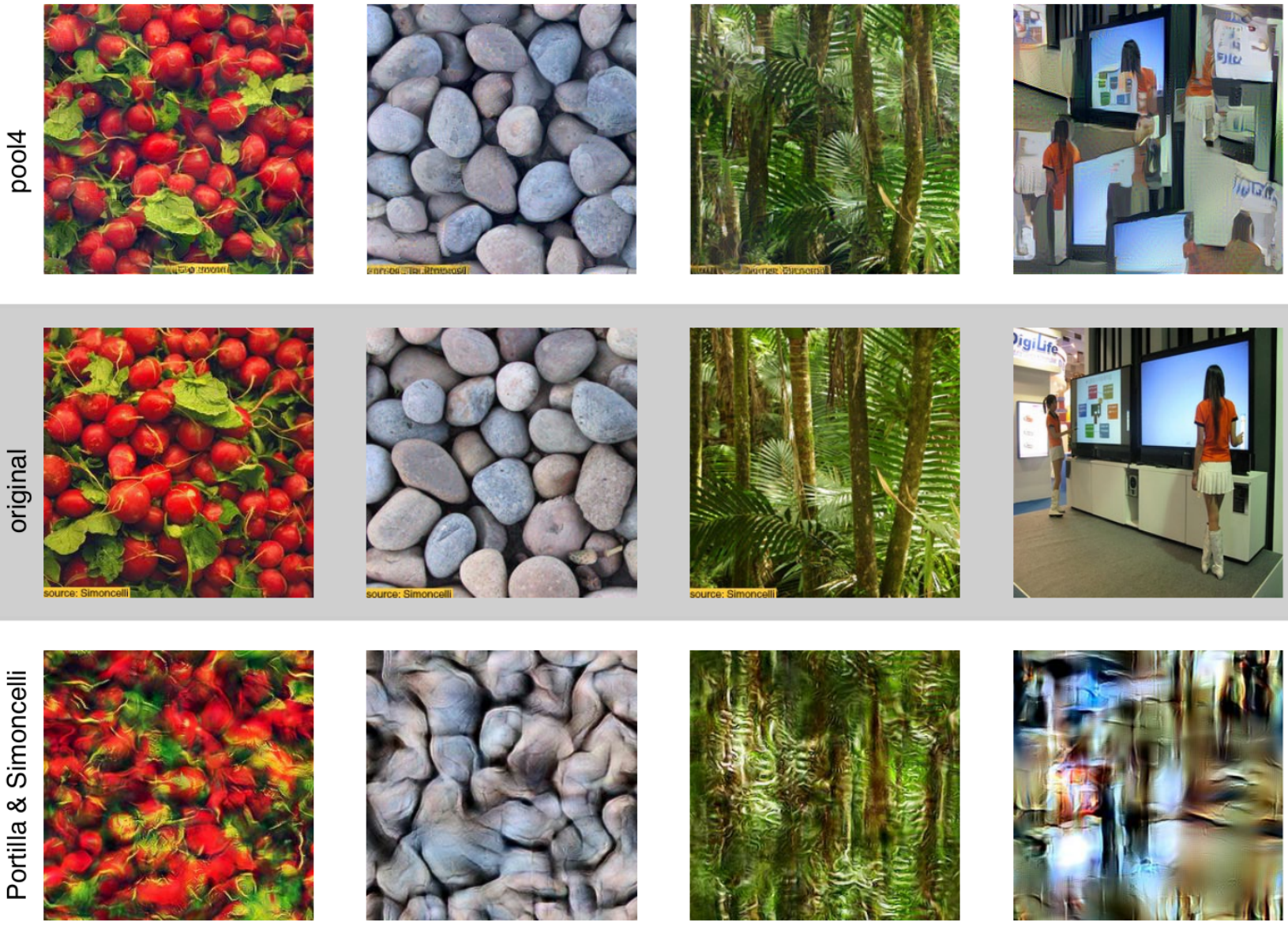


- The gradient of the energy is computed using back-propagation routines.
- The authors use a quasi-Newton algorithm: L-BFGS that stands for Limited-memory Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm.
- It consists in using a low-rank approximation of the Hessian matrix for computing the descent direction.

Gatys et al algorithm: Depth influence



Gatys et al algorithm: Results



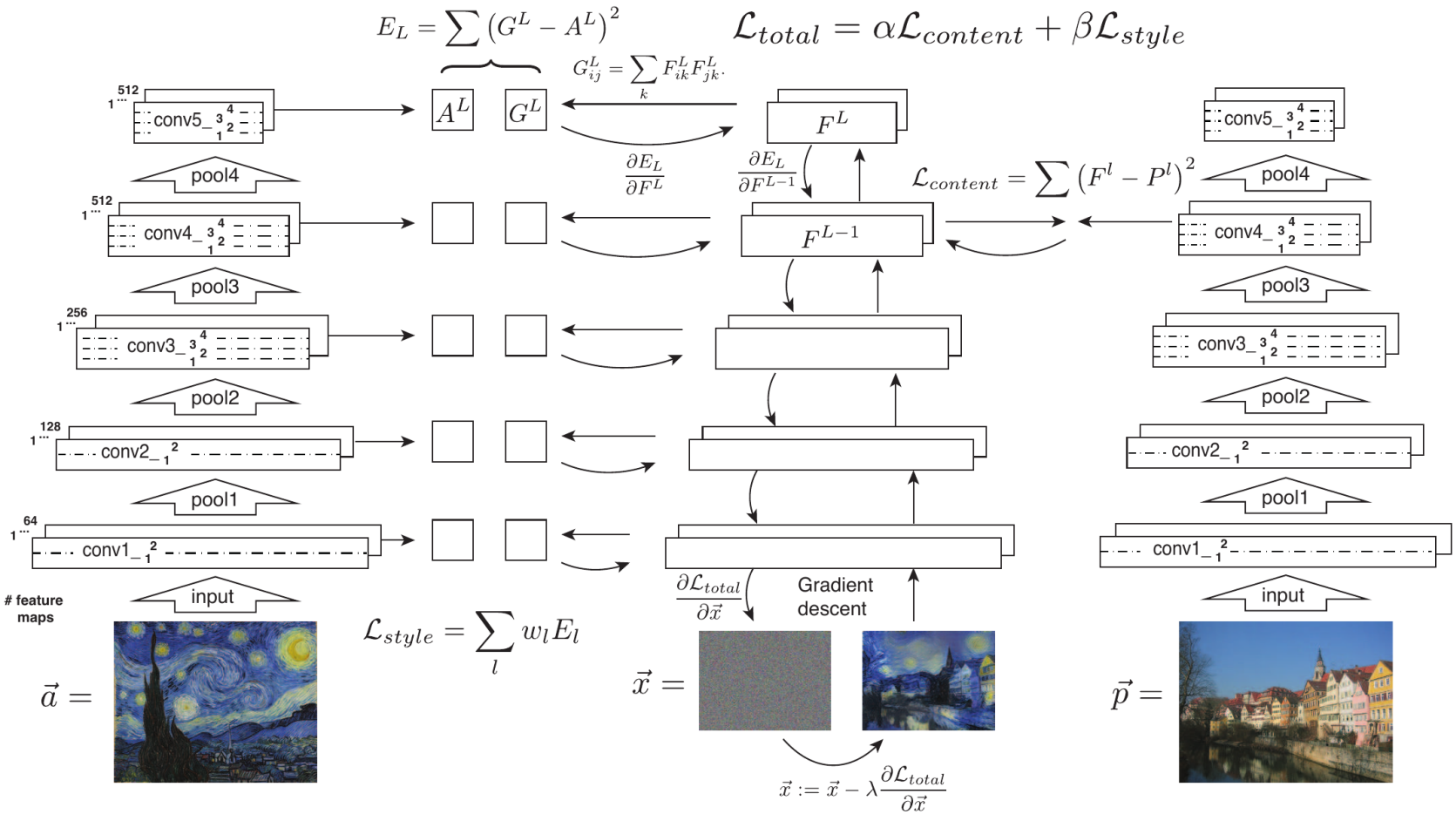
- MORE RESULTS

Gatys et al algorithm

- This algorithm is the current state of the art.
- The computational cost is really high (even with high-end GPUs it takes minutes).
- A lot of improvements have been proposed, eg by adding term to the energy or by adding correlation between layers.
- Extension for style transfer with equally impressive results.

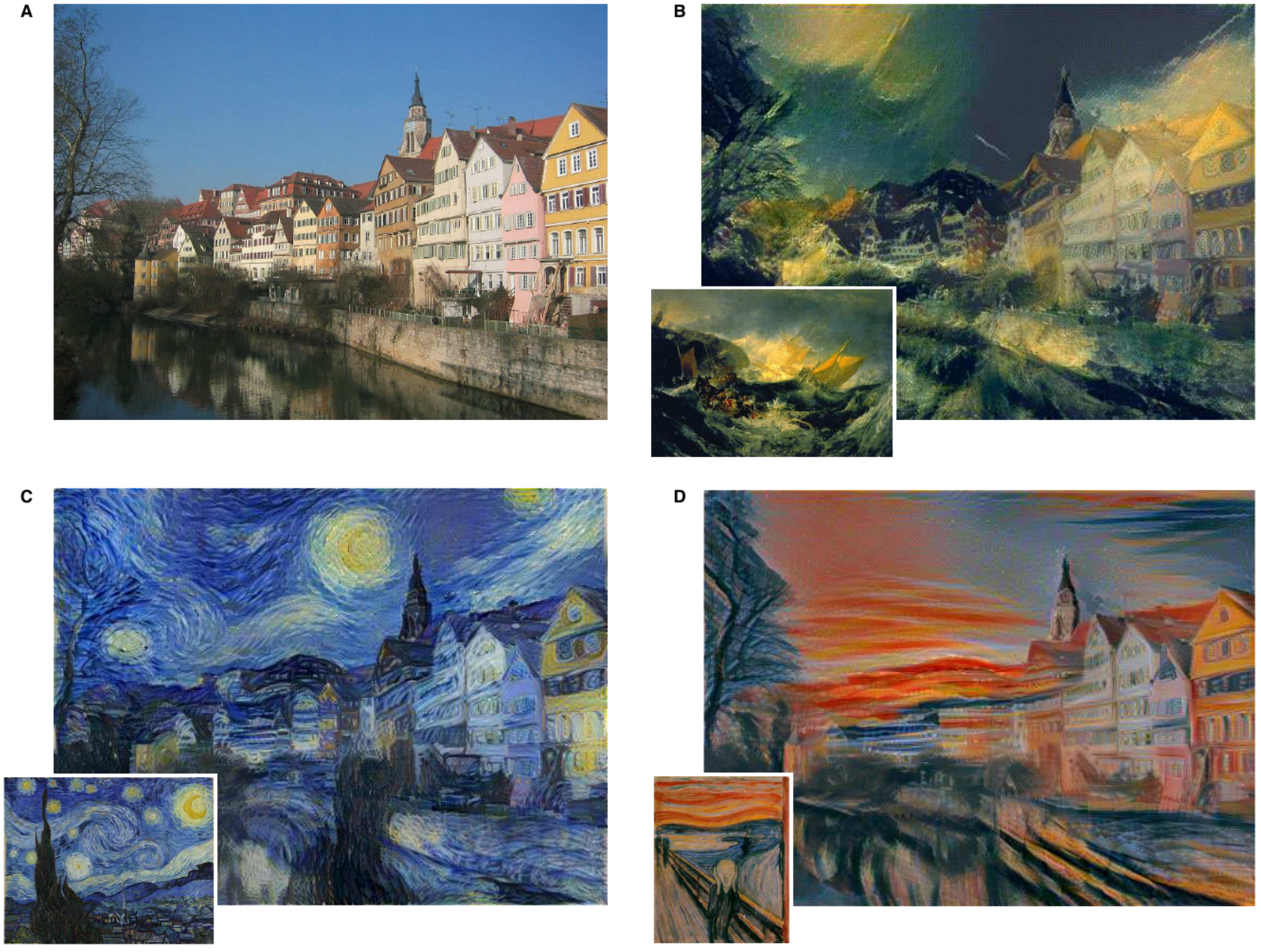
Gatys et al for style transfer

Reference: [Gatys et al 2016]



Gatys et al for style transfer

Reference: [Gatys et al 2016]

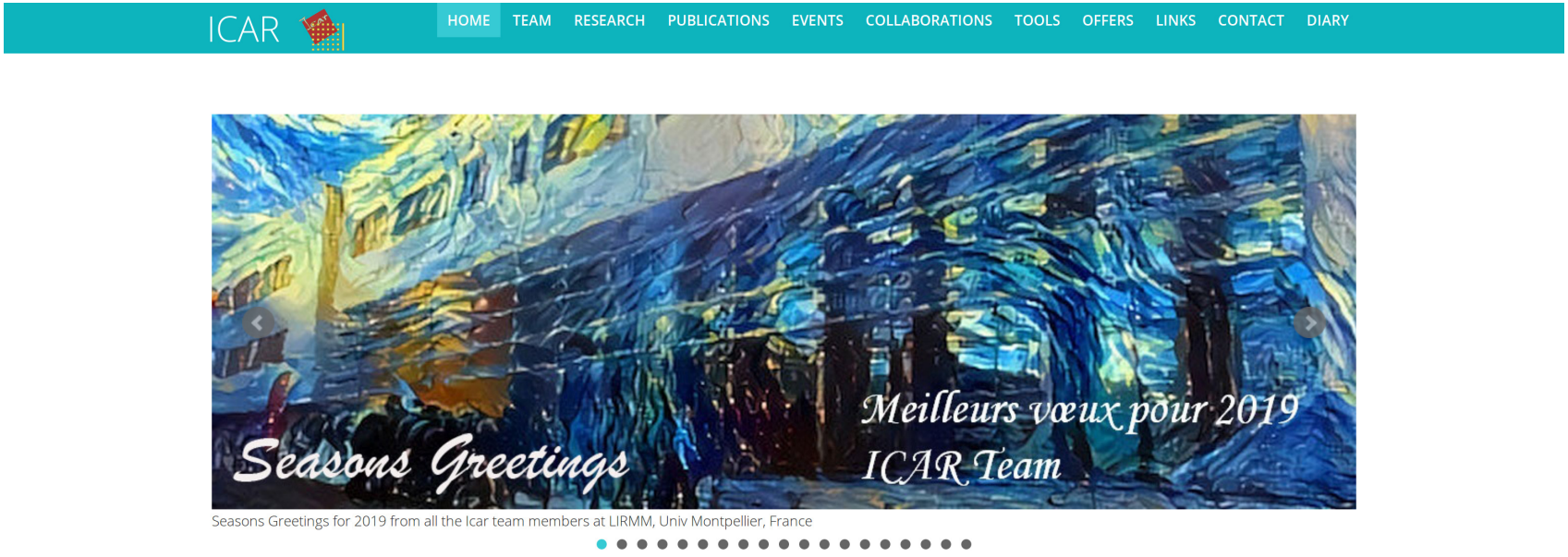


Gatys et al for style transfer

- Free online service: `https://deepart.io/` (wait 20 minutes)
- Lot of results
- Used for the lab webpage?

Gatys et al for style transfer

- Free online service: `https://deepart.io/` (wait 20 minutes)
- Lot of results
- Used for the lab webpage?

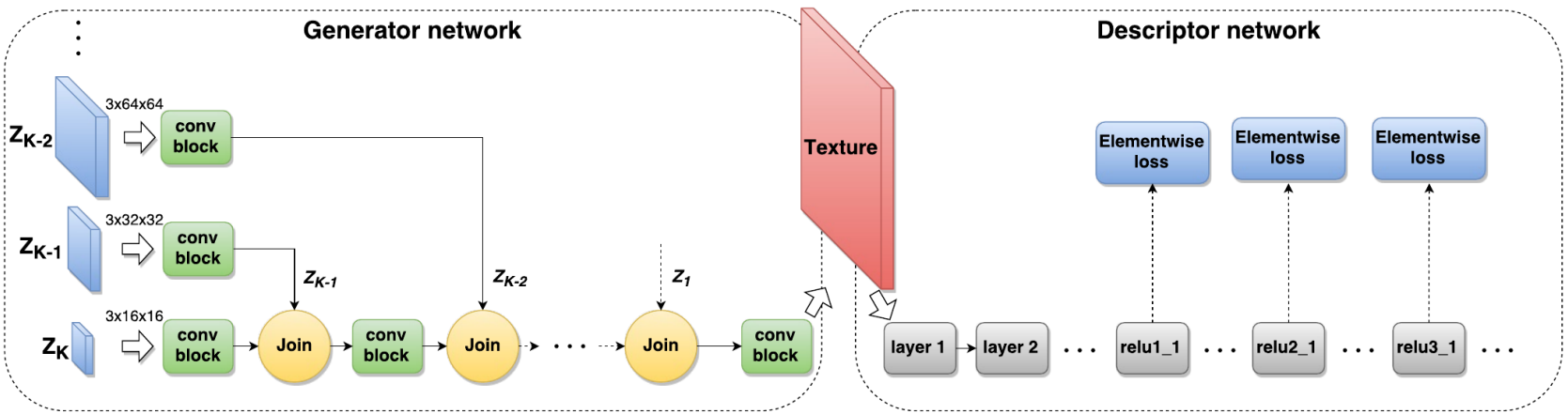


Gatys et al for style transfer and texture synthesis

- Very nice and clean PyTorch implementation:
  `https://github.com/leongatys/PytorchNeuralStyleTransfer`
- But it is **very slow** on CPU and still slow with high-end GPU (and memory consuming, e.g. 8 GB of memory for a 1024×1024 image).
- Regarding texture modeling, the **number of parameters is huge**: Textures are described by the Gram matrices and the number of elements in the Gram matrices totals 850k. That is 1000 times more than Portilla-Simoncelli !

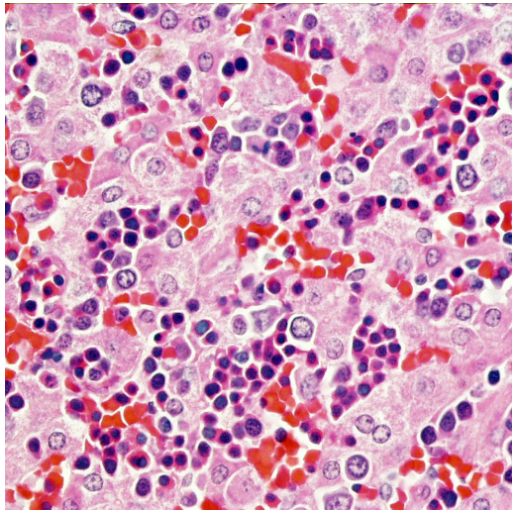
Generative networks for texture synthesis

- A workaround for speeding up synthesis is to train generative forward networks to mimic Gatys algorithm, as proposed by [Ulyanov et al. 2016].
- Then synthesis is fast but the texture quality is not as good, and a network has to be trained for each new image.

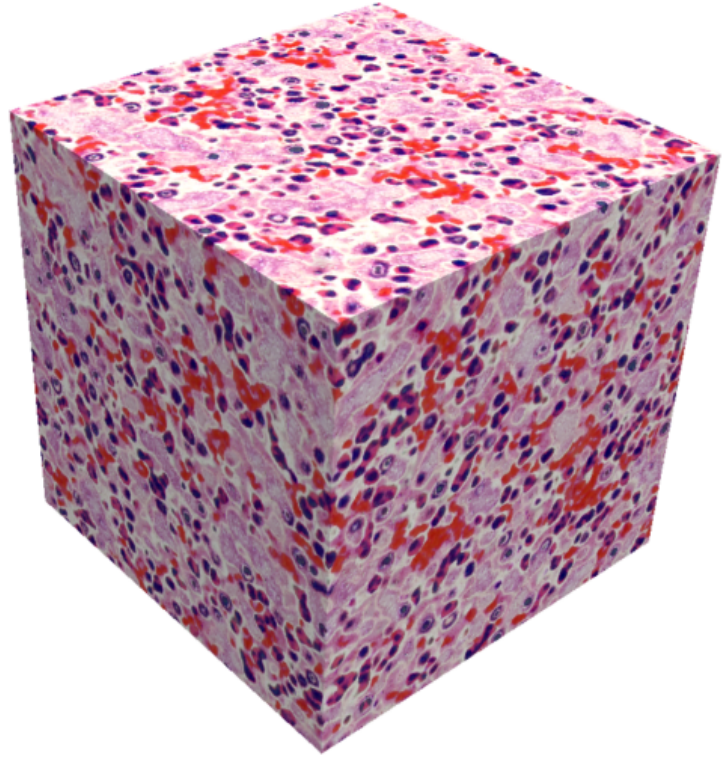


On Demand Solid Texture Synthesis Using Deep 3D Networks

- Ongoing work with Jorge Gutierrez (PhD), Julien Rabin and Thomas Hurtut [Gutierrez et al 2019]: We extend this idea for 3D texture where the Gatys approach is infeasible.



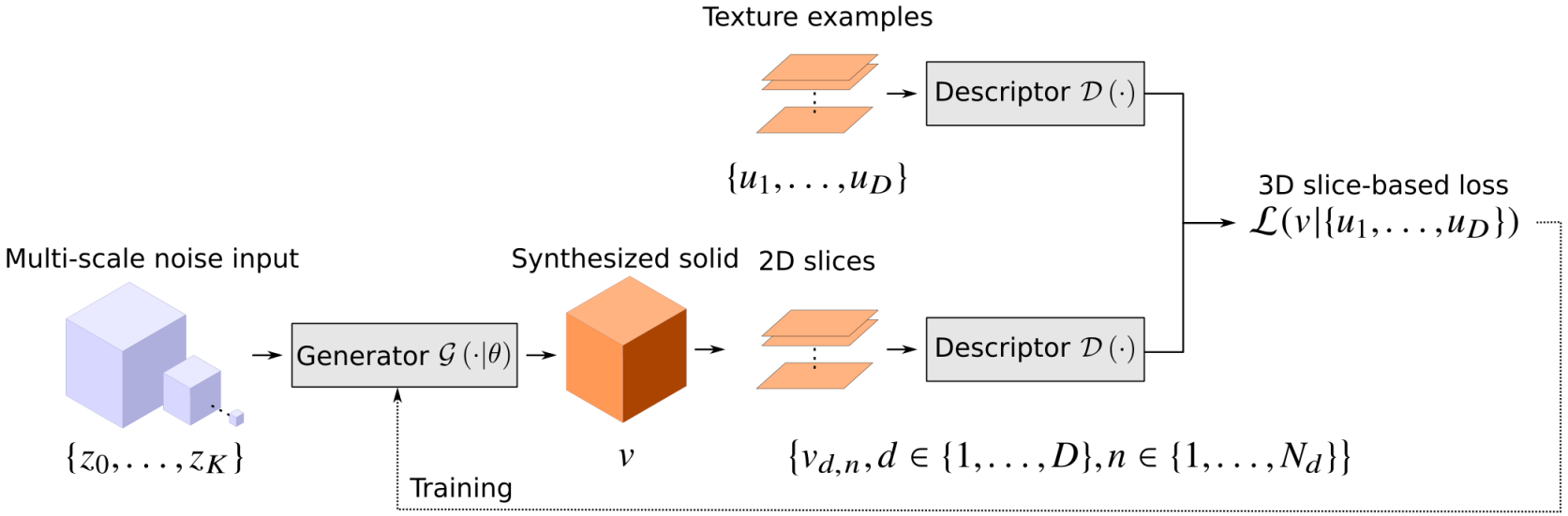
Input



Output

On Demand Solid Texture Synthesis Using Deep 3D Networks

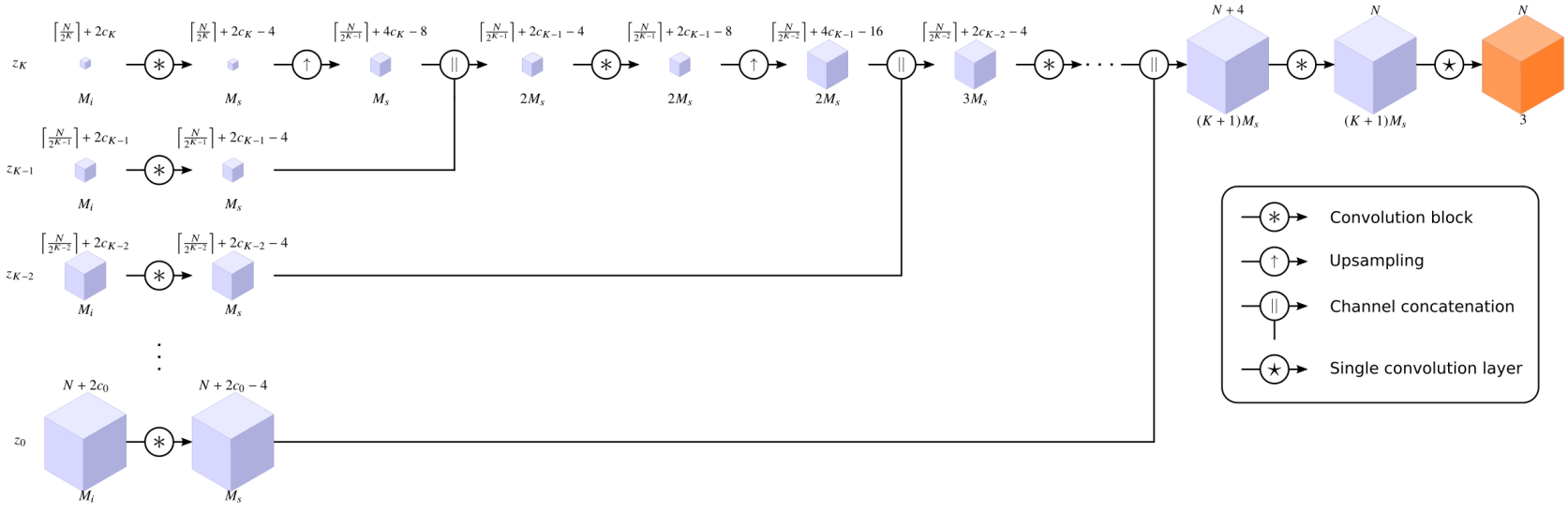
Training framework for the proposed CNN Generator network:



- The generator $\mathcal{G}(\cdot|\theta)$ with parameters θ processes a multi-scale noise input Z to produce a solid texture v
- The loss $\mathcal{L}$ compares, for each direction d, the feature statistics induced by the example u_d in the layers of the pre-trained Descriptor network $\mathcal{D}(\cdot)$ (loss of Gatys et al).

On Demand Solid Texture Synthesis Using Deep 3D Networks

Schematic of the Generator's architecture:

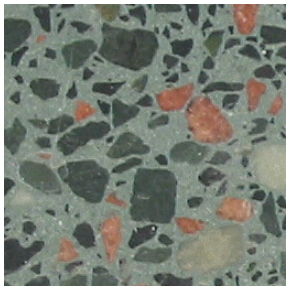


- Processes a set of noise inputs $Z = \{z_0, \ldots, z_K\}$ at $K + 1$ different scales using convolution operations and non-linear activations
- The information at different scales is combined using upsampling and channel concatenation.

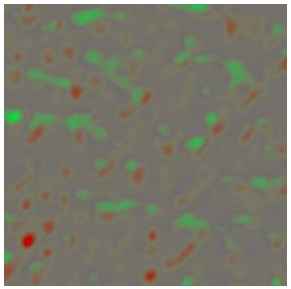
On Demand Solid Texture Synthesis Using Deep 3D Networks

Training: Find the parameters θ for a given texture

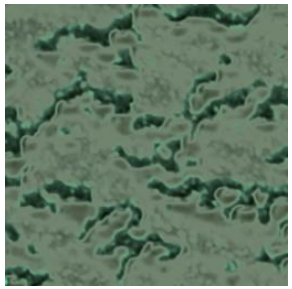
- Minimize Gatys' loss for each slice
- Exploit invariance by translation to generate batches of width one voxel only ("single-slice training scheme")
- Minimization using 3000 iteration of Adam algorithm



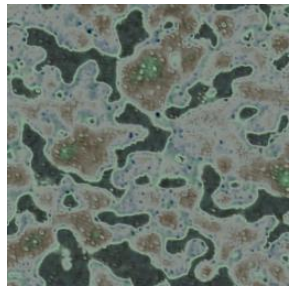
input



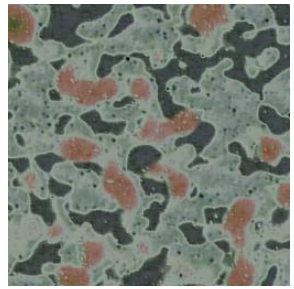
10



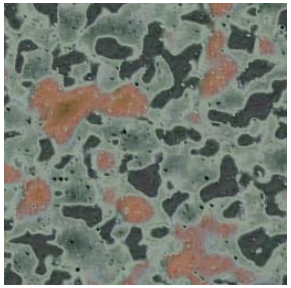
20



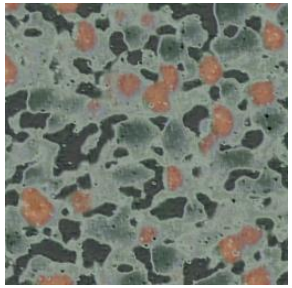
50



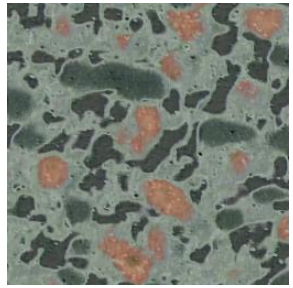
100



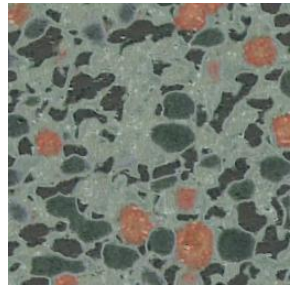
200



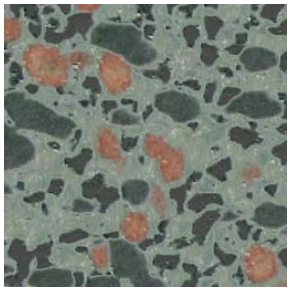
300



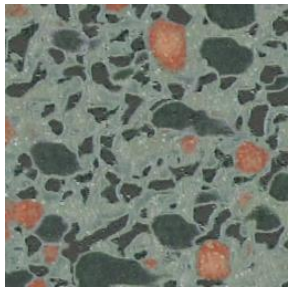
500



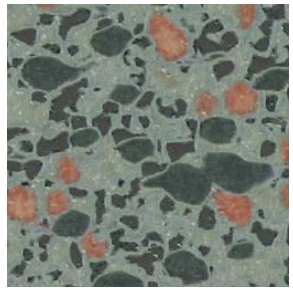
1000



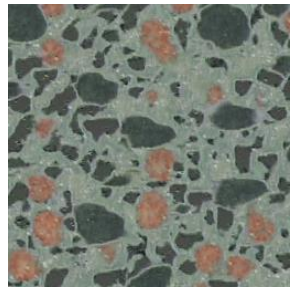
1500



2000

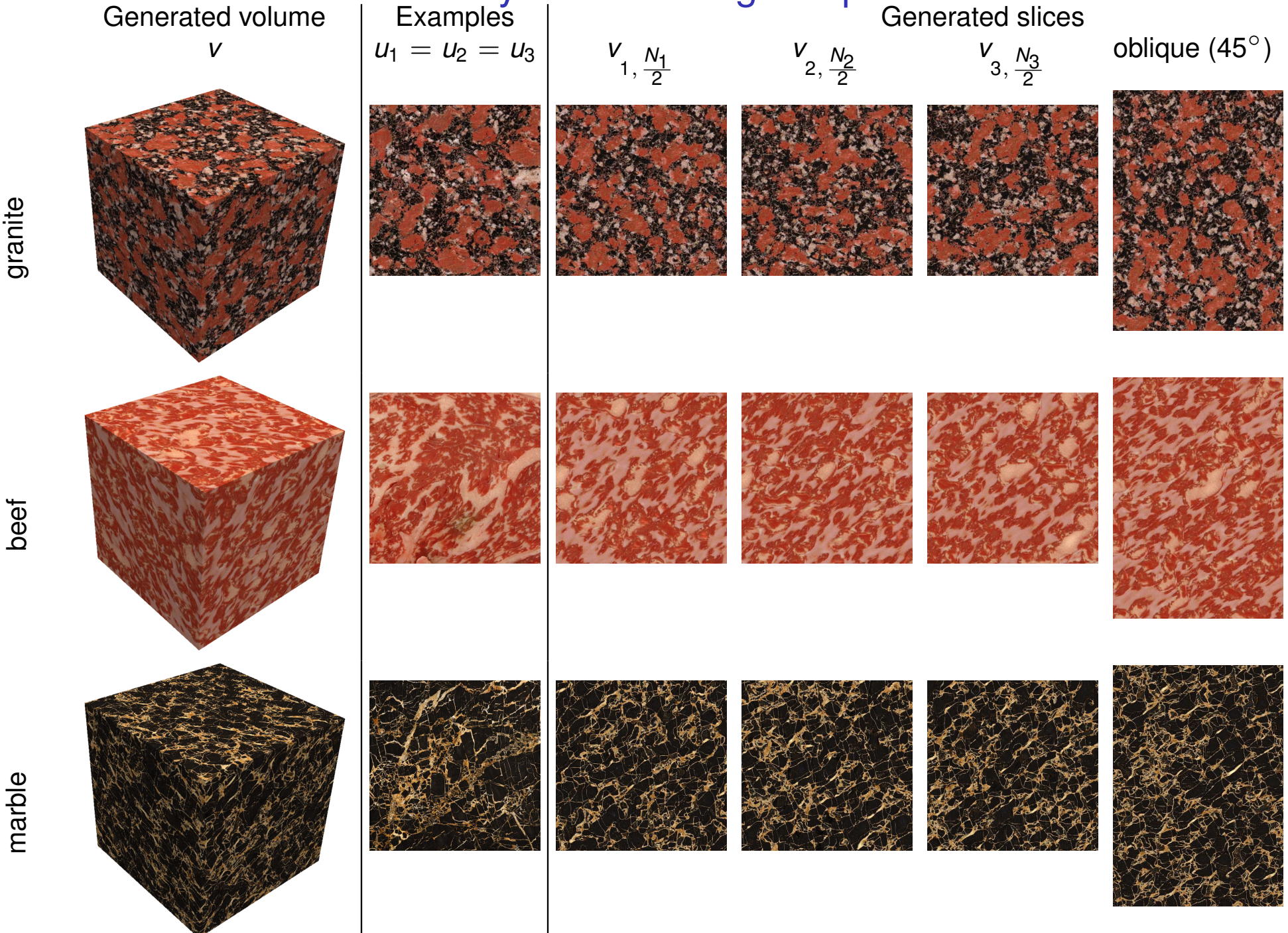


2500

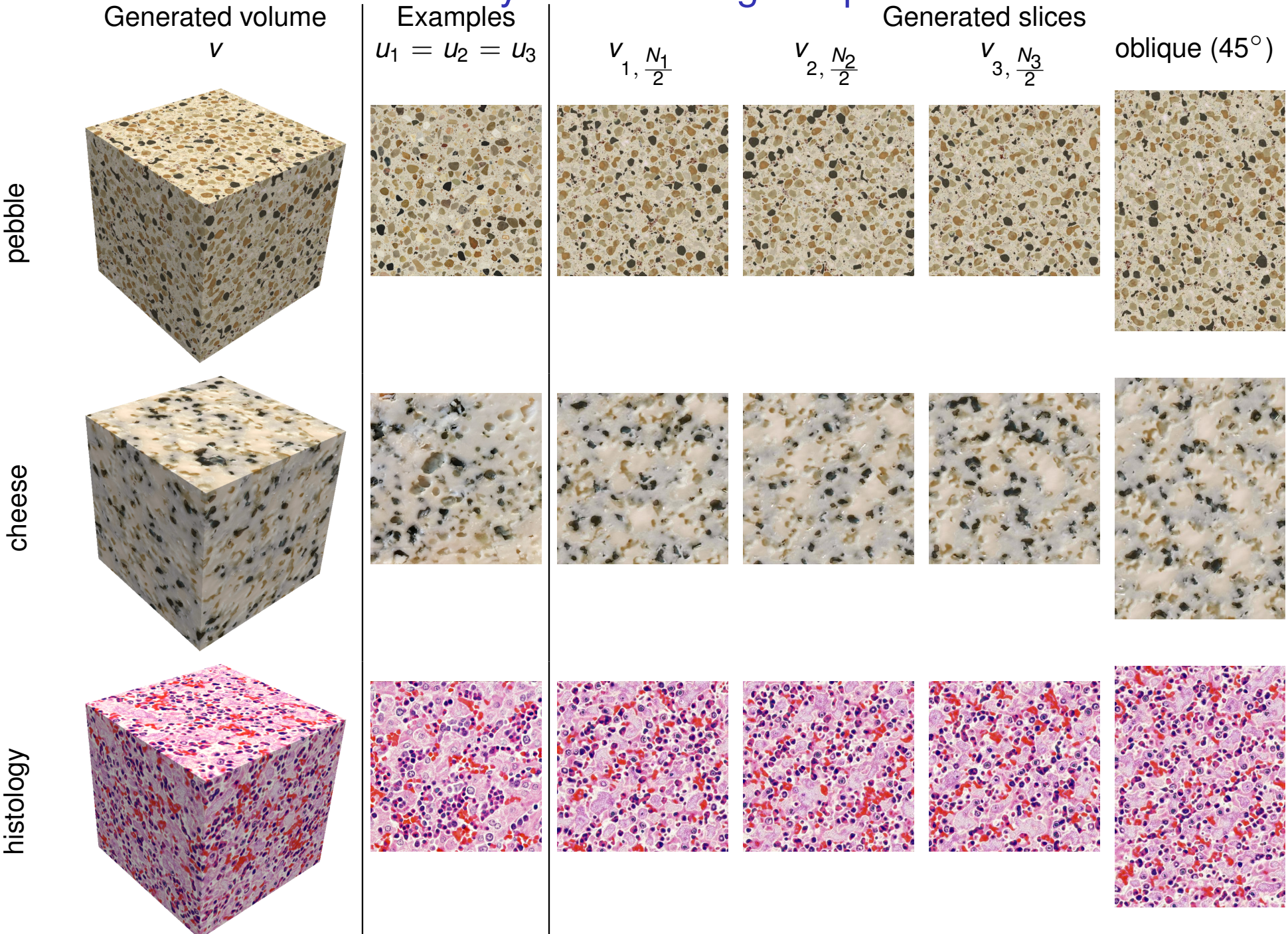


3000

On Demand Solid Texture Synthesis Using Deep 3D Networks

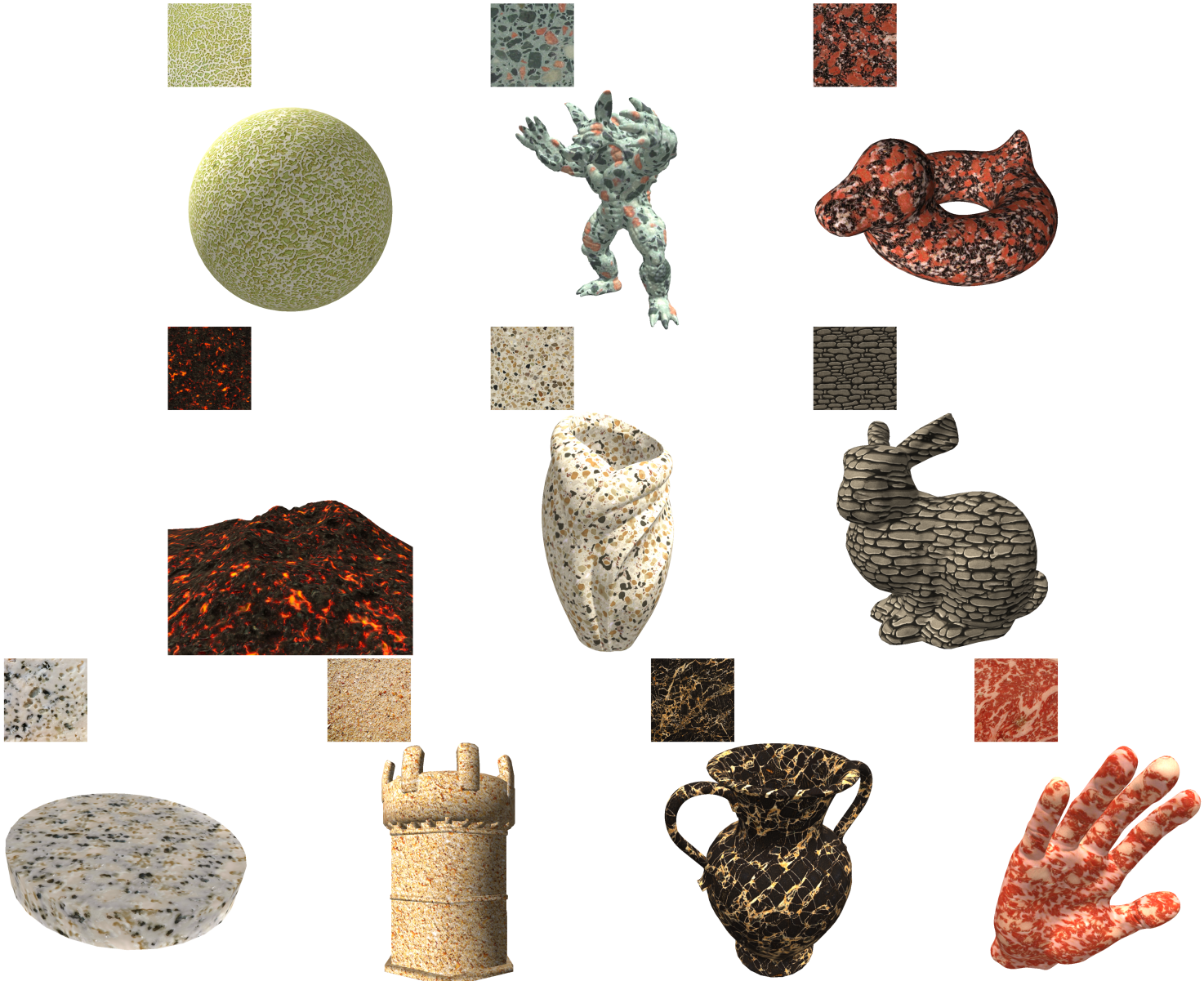


On Demand Solid Texture Synthesis Using Deep 3D Networks



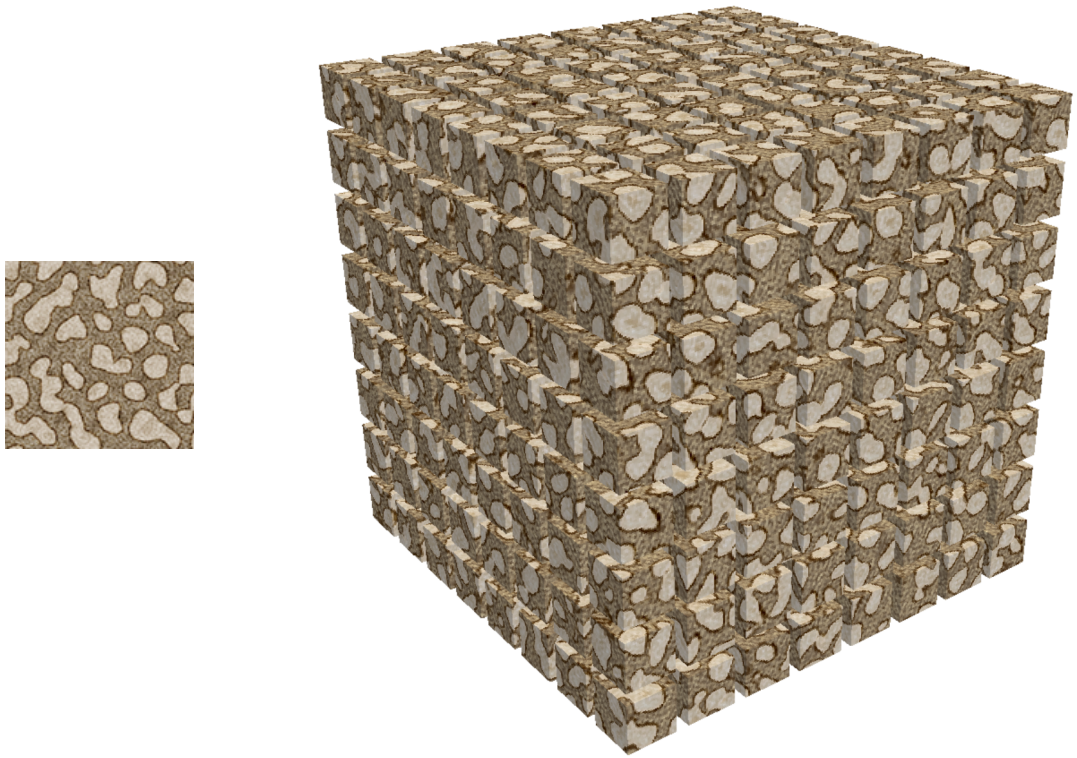
On Demand Solid Texture Synthesis Using Deep 3D Networks

- Solid textures can be used to apply textures on surfaces without parametrization.



On Demand Solid Texture Synthesis Using Deep 3D Networks

- Fast synthesis thanks to the feed forward network (1 sec for 256^3)
- On demand synthesis using a pseudo random number generator seed with spatial coordinates



- Training and synthesis with high resolution images without memory issues thanks to the single slice strategy.

Bibliographic references I


P. Brémaud, *Markov Chains: Gibbs Fields, Monte Carlo Simulation, and Queues*, Springer, 1998


T. Briand, J. Vacher, B. Galerne and J. Rabin, *The Heeger & Bergen Pyramid Based Texture Synthesis Algorithm*, Image Processing On Line, 2014


B. Galerne, A. Leclaire, and L. Moisan. Texton noise. Computer Graphics Forum, 2017


B. Galerne, Y. Gousseau, and J.-M. Morel, *Random phase textures: Theory and synthesis*, IEEE Trans. Image Process., 2011


B. Galerne, Y. Gousseau, J.-M. Morel, *Micro-Texture Synthesis by Phase Randomization*, Image Processing On Line, 2011


L. Gatys, A. S. Ecker, and M. Bethge, *Texture synthesis using convolutional neural networks*, in Advances in Neural Information Processing Systems, 2015


Gatys, L. A., Ecker, A. S., & Bethge, M. (2016, June). Image style transfer using convolutional neural networks. In Computer Vision and Pattern Recognition (CVPR), 2016 IEEE Conference on (pp. 2414-2423). IEEE.

Bibliographic references II


Gutierrez, J., Rabin, J., B. Galerne, & Hurtut, T., On Demand Solid Texture Synthesis Using Deep 3D Networks, submitted


D. J. Heeger and J. R. Bergen, *Pyramid-based texture analysis/synthesis*, SIGGRAPH '95, 1995


J.-L. Lisani, A. Buades, and J.-M. Morel. *Image color cube dimensional filtering and visualization*, Image Processing On Line, 2011.


Y. Lu, S.-C. Zhu, and Y. N. Wu, *Learning frame models using CNN filters*, in 31th conference on artificial intelligence, 2016.


L. Moisan, *Periodic plus smooth image decomposition*, J. Math. Imag. Vis., 2011


A. V. Oppenheim and J. S. Lim, *The importance of phase in signals*, Proceedings of the IEEE, 1981


J. Portilla and E. Simoncelli, *A parametric texture model based on joint statistics of complex wavelet coefficients*, IJCV, 40 (2000)


E. P. Simoncelli, W. T. Freeman, E. H. Adelson, and D. J. Heeger. *Shiftable multi-scale transforms*. IEEE Transaction on Information Theory, 1992.

Bibliographic references III


K. Simonyan and A. Zisserman, *Very deep convolutional networks for large-scale image recognition*, tech report, 2014


D. Ulyanov, V. Lebedev, A. Vedaldi, and V. Lempitsky, *Texture networks: Feed-forward synthesis of textures and stylized images*, in *ICML*, 2016, pp. 1349–1357.


Xiang-Yang Wang, Bei-Bei Zhang and Hong-Ying Yang, *Content-based image retrieval by integrating color and texture features*, Multimedia Tools and Applications, 2014


L.-Y. Wei, S. Lefebvre, V. Kwatra, and G. Turk. *State of the art in example-based texture synthesis*, Eurographics 2009, 2009.


J. J. van Wijk, *Spot noise texture synthesis for data visualization*, SIGGRAPH ’91, 1991.


G.-S. Xia, S. Ferradans, G. Peyré and J.-F. Aujol, Synthesizing and Mixing Stationary Gaussian Texture Models, SIAM J. on Imaging Science, 2014.

Merci pour votre attention