Image texture synthesis using spot noise and phase randomization

Bruno Galerne
bruno.galerne@parisdescartes.fr

MAP5, Université Paris Descartes

Master MVA
Cours “Méthodes stochastiques pour l’analyse d’images”
Lundi 20 février 2017

Outline

Outline

What is a texture?

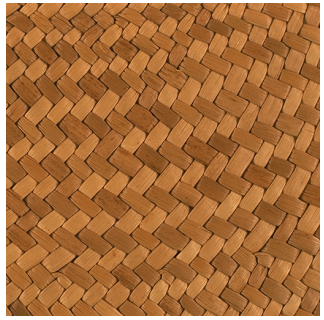
A minimal definition of a **texture** image is an “image containing repeated patterns” [Wei et al., 2009].
The family of patterns reflects a certain amount of randomness, depending on the nature of the texture.
Two main subclasses:

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



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



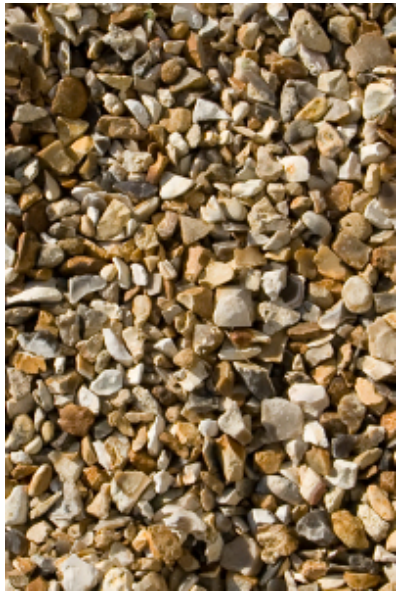


Textures and scale of observation

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

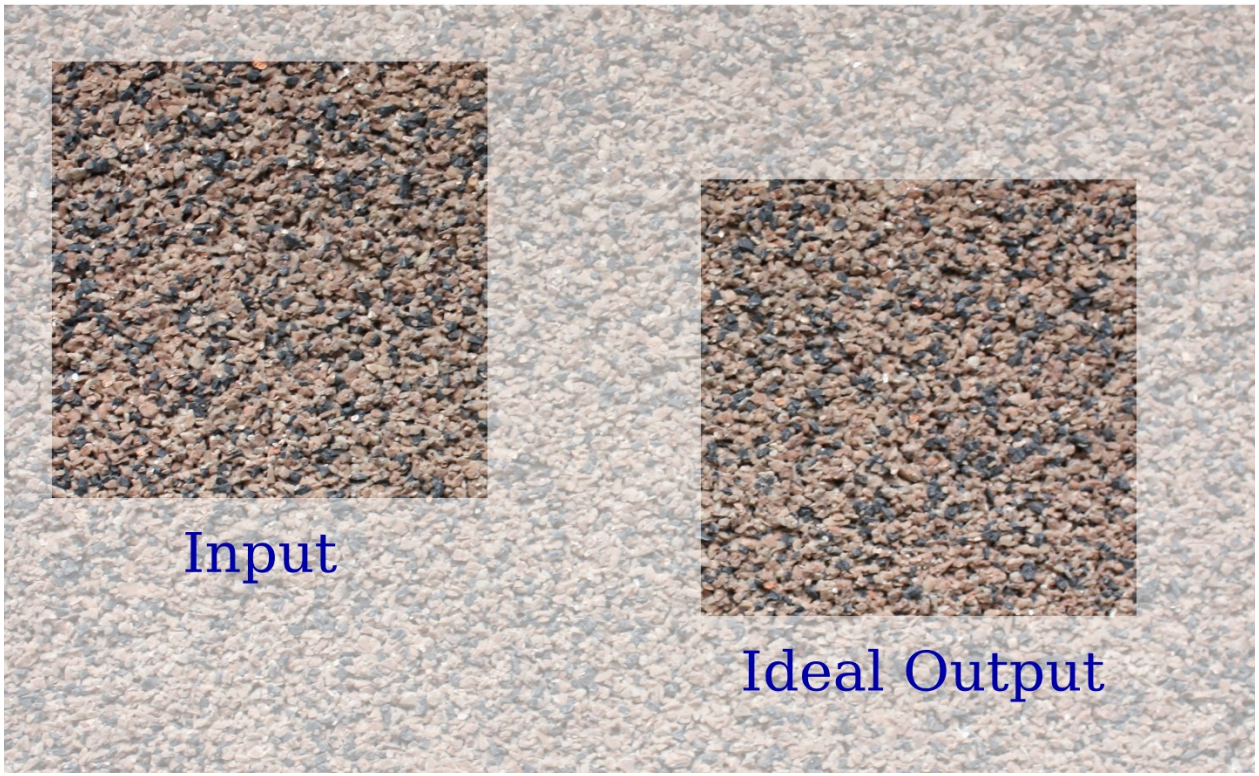


Macro-texture

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 algorithms

Two main kinds of algorithm:

1. Texture synthesis using statistical constraints:

   Algorithm:

   1. Extract some meaningful “statistics” from the input image (e.g. distribution of colors, of Fourier coefficients, of wavelet coefficients,...).
   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...
   - See next course (March, 17) for more details.

Texture synthesis algorithms

Two main kinds of algorithm:

1. Texture synthesis using statistical constraints:

   Algorithm:

   1. Extract some meaningful “statistics” from the input image (e.g. distribution of colors, of Fourier coefficients, of wavelet coefficients,...).
   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...
   - See next course (March, 17) for more details.

Texture synthesis algorithms

Two main kinds of algorithm:

1. Texture synthesis using statistical constraints:

   Algorithm:

   1. Extract some meaningful “statistics” from the input image (e.g. distribution of colors, of Fourier coefficients, of wavelet coefficients,...).
   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...
   - See next course (March, 17) for more details.

Heeger-Bergen algorithm [Heeger and Bergen, 1995]

Statistical constraints:

- Histogram of colors.
- Histogram of wavelet coefficients at each scale.

Algorithm: Alternating projections into the constraints starting from a white noise image.

Texture synthesis by phase randomization

What about the **Random Phase Noise (RPN)** and
Asymptotic Discrete Spot Noise (ADSN) presented today ?

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

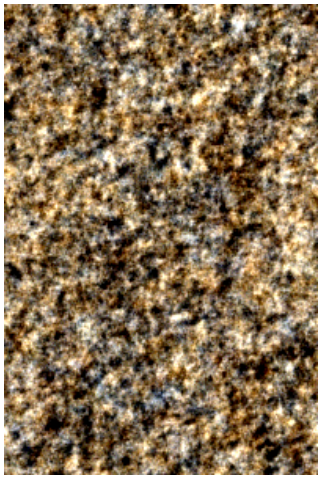
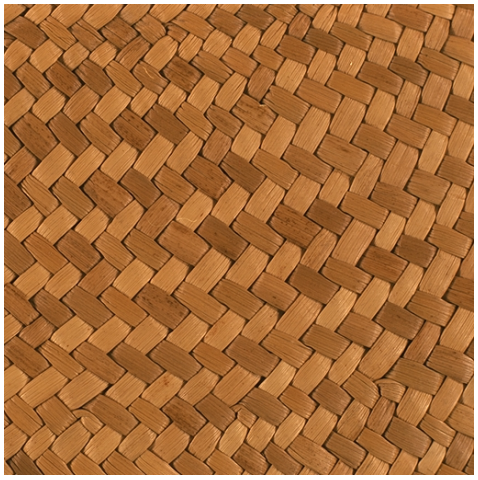
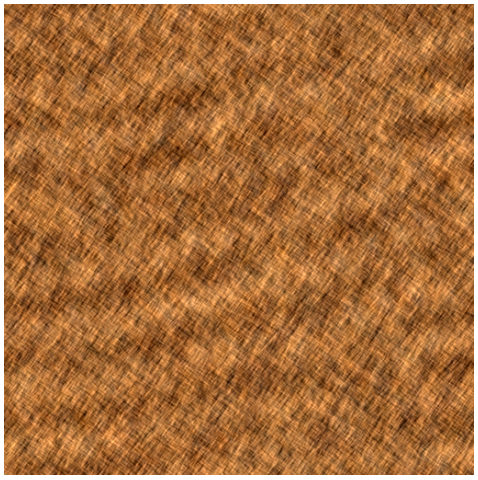
- It belongs to the first category: texture synthesis by statistical constraints.
- Here the “statistics” are the moduli of the Fourier coefficients.

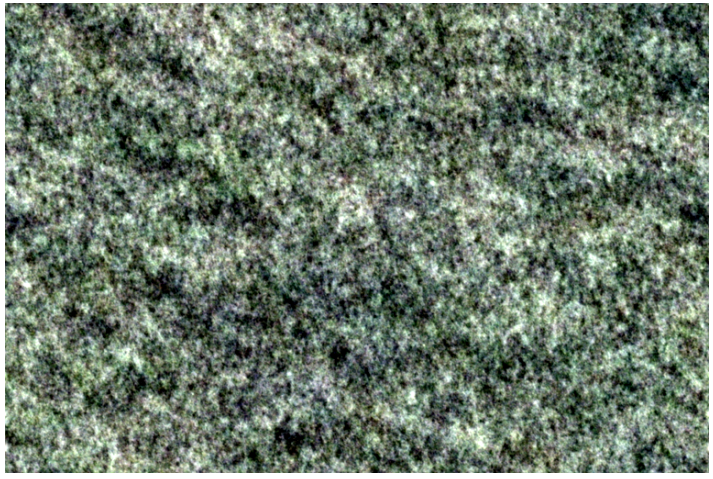
Texture synthesis by phase randomization

- Successful examples with micro-textures

Texture synthesis by phase randomization

- Failure examples with macro-textures



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.

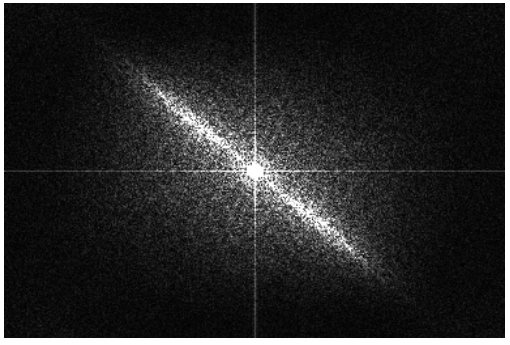
Discrete Fourier transform of digital images

Symmetry property:

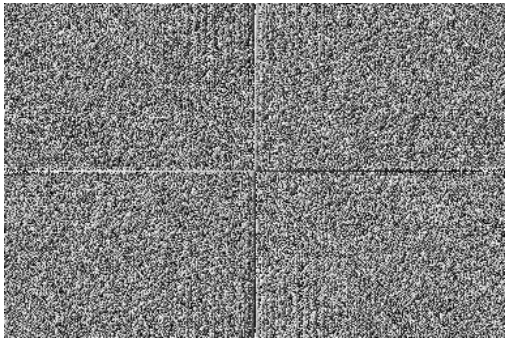
- $|\hat{u}|$: **Fourier modulus** of u is even.
- $\arg(\hat{u})$: **Fourier phase** of u is odd.

Visualization of the DFT:

Image u



Modulus $|\hat{u}|$



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

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

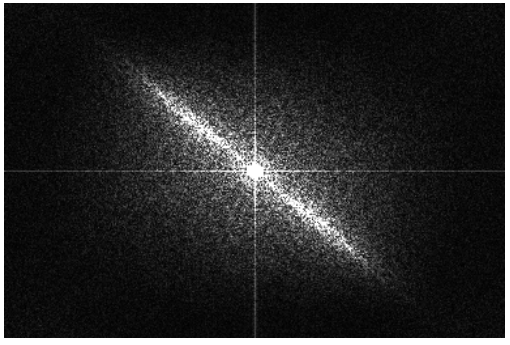
Discrete Fourier transform of digital images

Symmetry property:

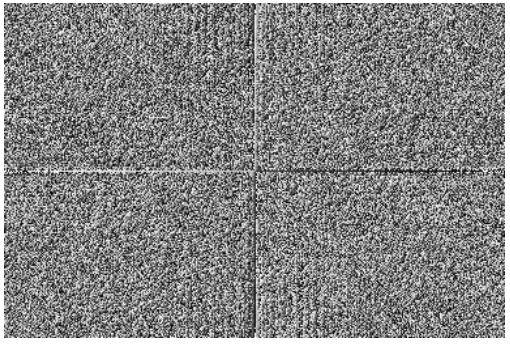
- $|\hat{u}|$: **Fourier modulus** of u is even.
- $\arg(\hat{u})$: **Fourier phase** of u is odd.

Visualization of the DFT:

Image u



Modulus $|\hat{u}|$



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

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

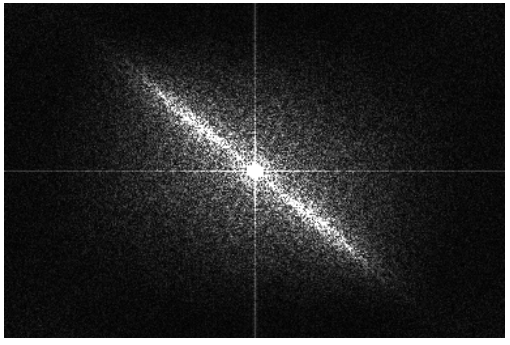
Discrete Fourier transform of digital images

Symmetry property:

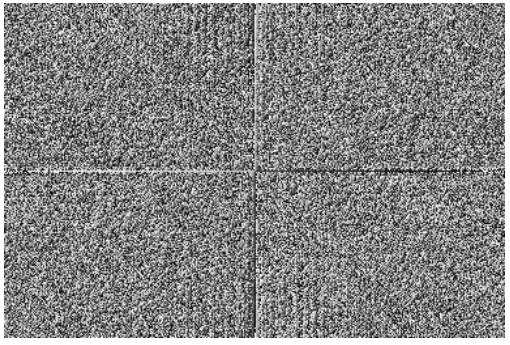
- $|\hat{u}|$: **Fourier modulus** of u is even.
- $\arg(\hat{u})$: **Fourier phase** of u is odd.

Visualization of the DFT:

Image u



Modulus $|\hat{u}|$



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

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

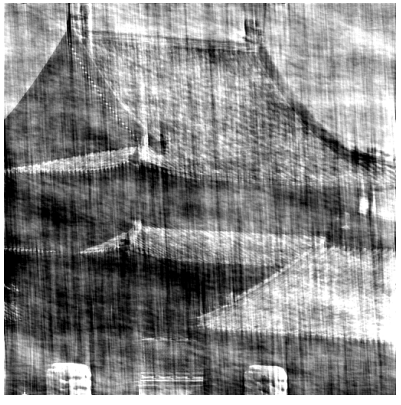
Modulus and phase of a digital image

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.

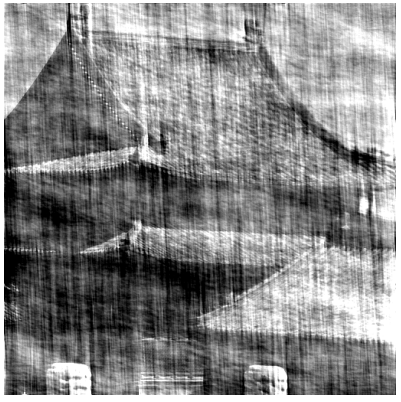
Modulus and phase of a digital image

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

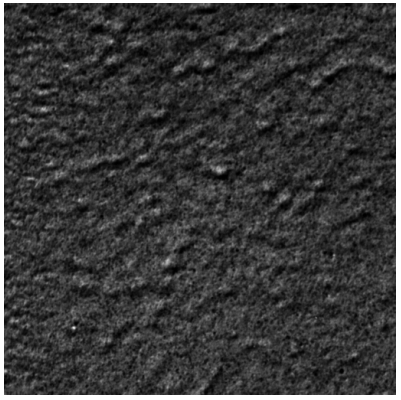
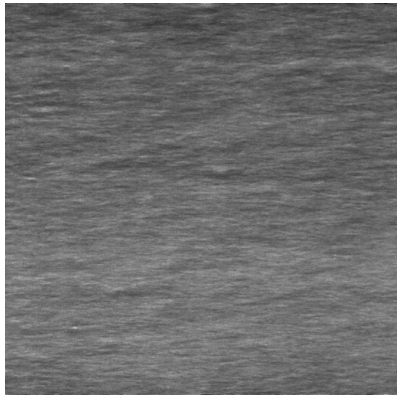


Image 2

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

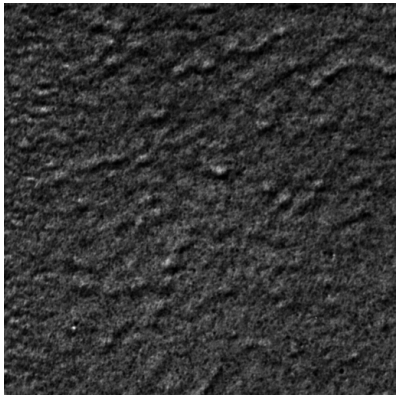
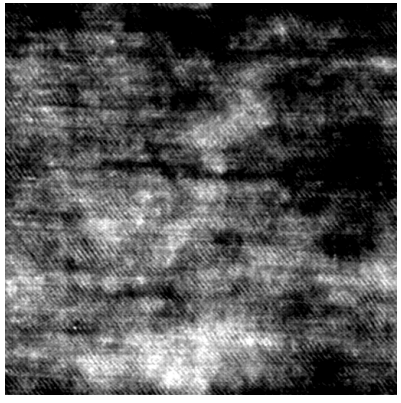
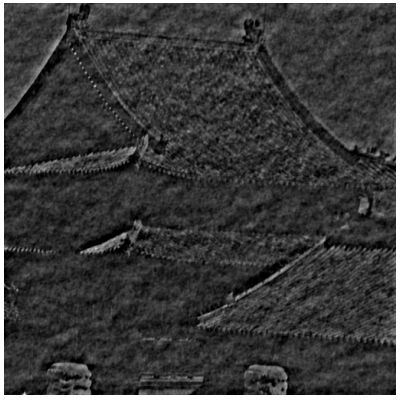


Image 2

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.

Outline

Random phase textures

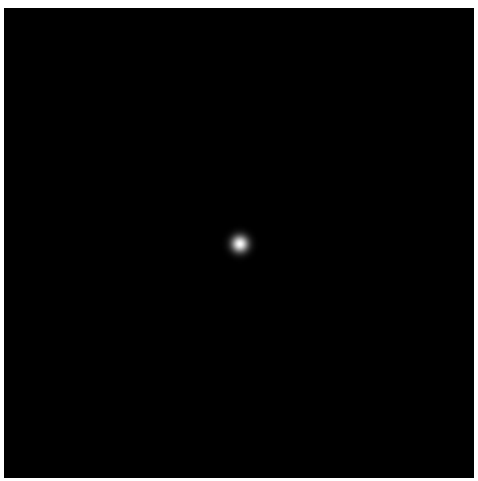
- We call *random phase texture* any image that is perceptually invariant to phase randomization.
- Phase randomization = replace the Fourier phase by a random phase.
- Definition: A random field $\theta : \hat{\Omega} \to \mathbb{R}$ is a ***random phase*** if
  1. Symmetry: θ is odd:
  $$\forall (s,t) \in \hat{\Omega}, \theta(-s,-t) = -\theta(s,t).$$
  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.

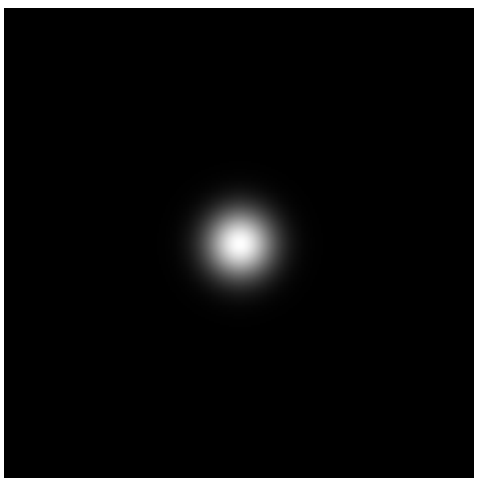
Random Phase Noise (RPN)

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

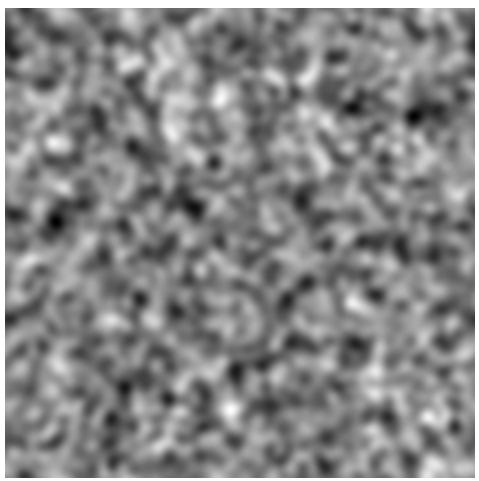
1. Compute the DFT $\hat{h}$ of the input h.
2. Compute a random phase θ using a pseudo-random number generator.
3. Set $\hat{Z} = \left|\hat{h}\right| e^{i\theta}$ (or $\hat{Z} = \hat{h}e^{i\theta}$).
4. Return Z the inverse DFT of $\hat{Z}$.



Original image h



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



RPN associated with h

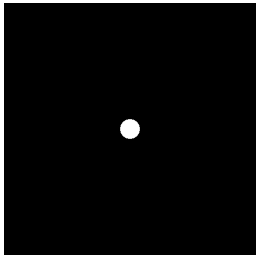
Outline

Discrete spot noise [van Wijk, 1991]

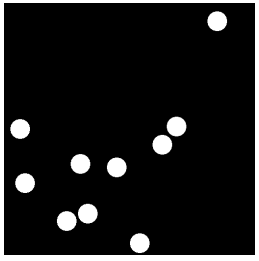
- Let h be a discrete image called *spot*.
- Let (X_k) be a sequence of random translation vectors which are i.d.d. and uniformly distributed over Ω.
- The **discrete spot noise of order** n **associated with** h is the random image

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

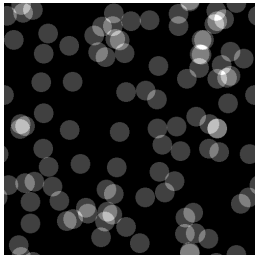
(translations with periodic boundary conditions)



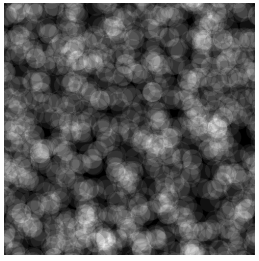
Spot h



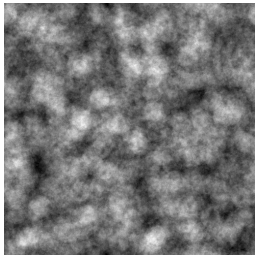
$n = 10$



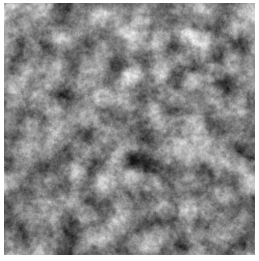
$n = 10^2$



$n = 10^3$

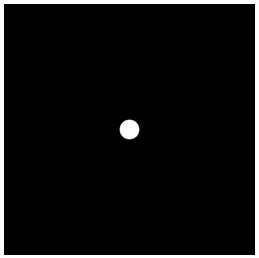


$n = 10^4$

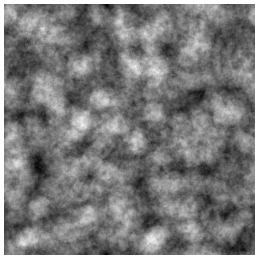


$n = 10^5$

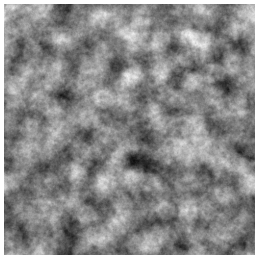
Limit of the DSN model ?



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:**
  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:

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

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:

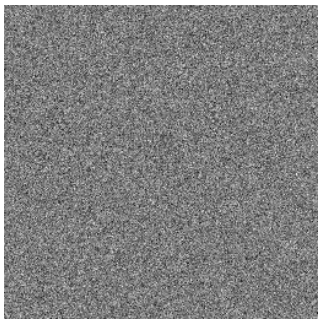
$$C_h(x, y) = \frac{1}{MN} \sum_{t \in \Omega} (h(x - t) - m)\,(h(y - t) - m)\,, \quad (x, y) \in \Omega.$$

Definition of *ADSN*:

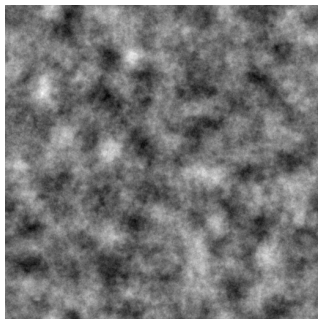
- The *ADSN* associated with h is the Gaussian vector $\mathcal{N}(0, C_h)$.

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

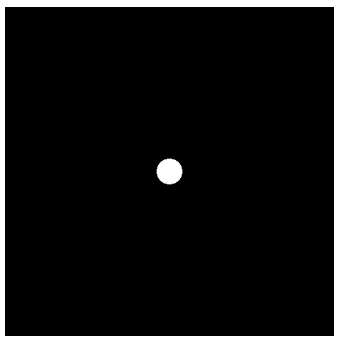


Gaussian vector: pixels have Gaussian distribution and are correlated

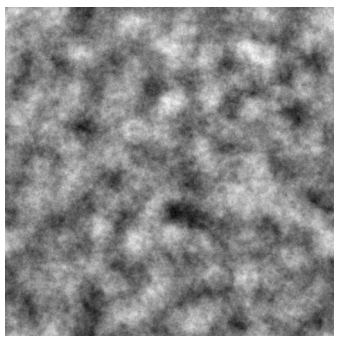
Convolution product: $(f * g)(x) = \sum_{y \in \Omega} f(x-y)g(y), \ x \in \Omega.$

Simulation of the *ADSN*:

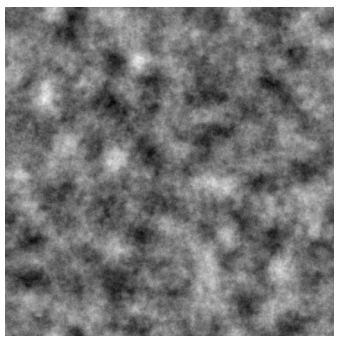
- Let $h \in \mathbb{R}^{M \times N}$ be a an image, m be the mean of h and X be a Gaussian white noise image.
- The random image $\frac{1}{\sqrt{MN}}(h-m) * X$ is the *ADSN* associated with h.



Spot h



DSN, $n = 10^5$



ADSN

ADSN Simulation

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

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

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

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

- 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 $x, y \in \Omega$,

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

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

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

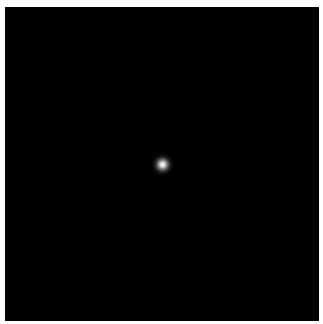
$$= \frac{1}{MN}\sum_{s\in\Omega}(h(s-x) - m)(h(t-y) - m)$$

$$= C_h(x, y)$$

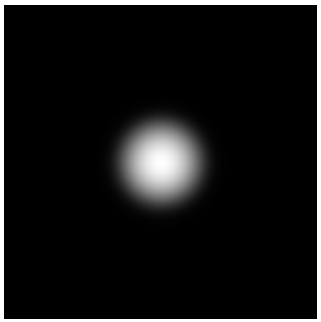
Differences between *RPN* and *ADSN*

Proposition:

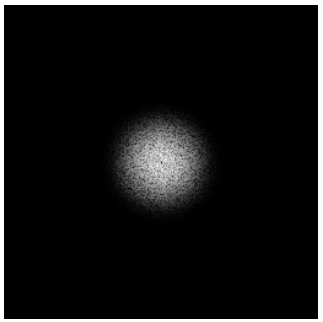
- *RPN* and *ADSN* both have a random phase.
- The Fourier modulus of *RPN* is the one of h.
- The Fourier modulus of *ADSN* is the pointwise multiplication between $\left|\hat{h}\right|$ and a Rayleigh noise.



Spot h

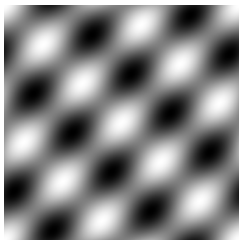


RPN Modulus

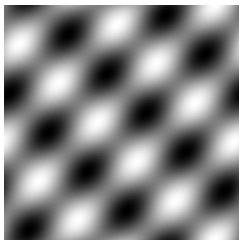


ADSN Modulus

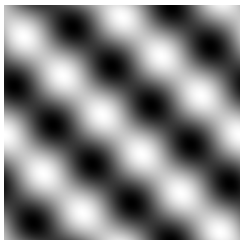
- ***RPN* and *ADSN* are two different processes.**



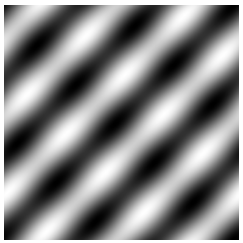
Spot h



RPN



An *ADSN* realization



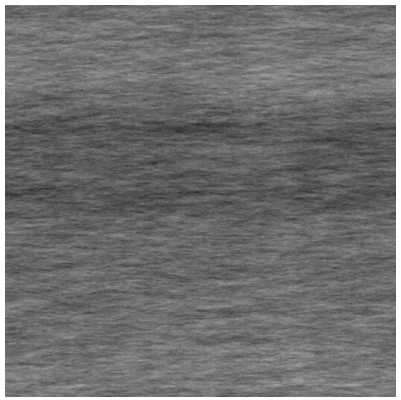
Another *ADSN* realization

Outline

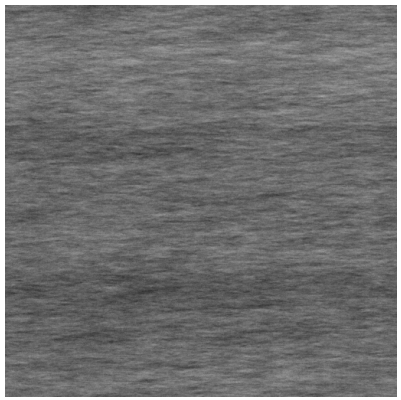
RPN and *ADSN* associated to texture images

- We add the original mean to *RPN* and *ADSN* realizations.
- 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.

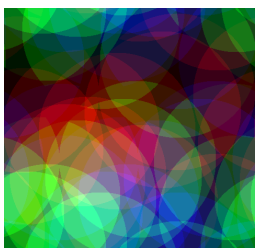
Extension to color images

- 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:

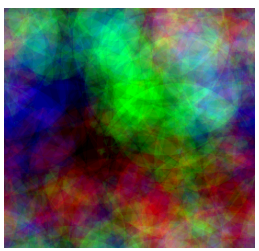
$$Y = \frac{1}{\sqrt{MN}} \begin{pmatrix} (h_r - m_r \mathbf{1}) * X \\ (h_g - m_g \mathbf{1}) * X \\ (h_b - m_b \mathbf{1}) * X \end{pmatrix}, \quad X \text{ a Gaussian white noise.}$$

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

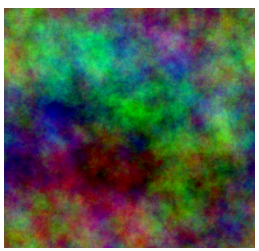
Spot h



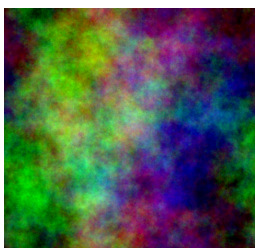
$n = 10$



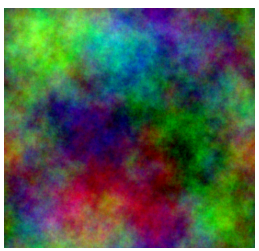
$n = 10^2$



$n = 10^3$



$n = 10^4$



color *ADSN*

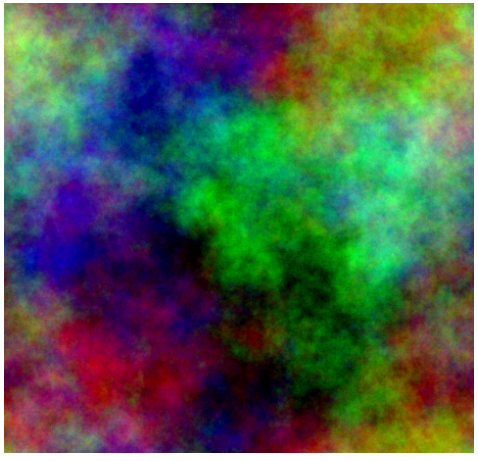
- Phase of color *ADSN*: The same random phase is added to the Fourier transform of each color channel.

Extension to color images

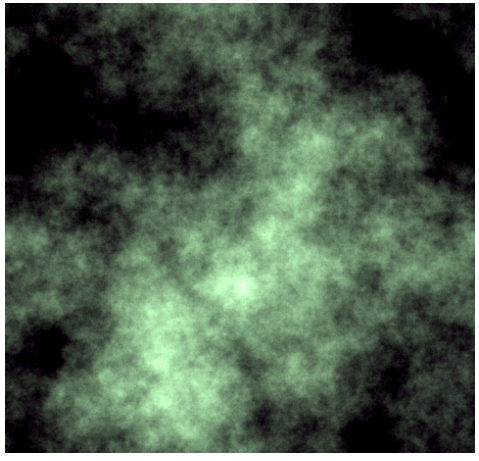
- **Color *RPN*:** By analogy, the *RPN* associated with a color image $h = (h_r, h_g, h_b)$ is the color image obtained by **adding the same random phase** to the Fourier transform of each color channel.

Original image h

Color *RPN*



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

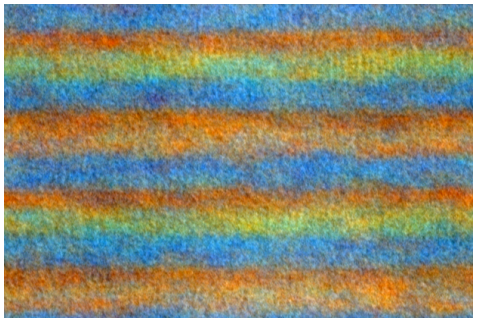


$$\hat{h} = \begin{pmatrix} |\hat{h}_R| e^{i\varphi_R} \\ |\hat{h}_G| e^{i\varphi_G} \\ |\hat{h}_B| e^{i\varphi_B} \end{pmatrix} \qquad \hat{Z} = \begin{pmatrix} |\hat{h}_R| e^{i(\varphi_R+\theta)} \\ |\hat{h}_G| e^{i(\varphi_G+\theta)} \\ |\hat{h}_B| e^{i(\varphi_B+\theta)} \end{pmatrix} \qquad \hat{Z}_W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

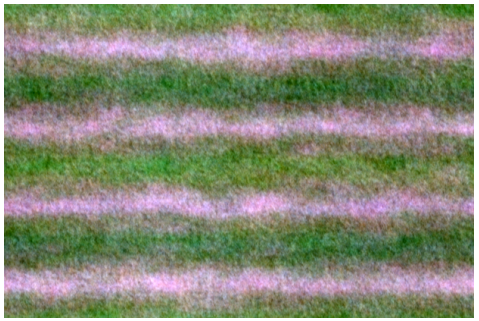
Extension to color images

- 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*

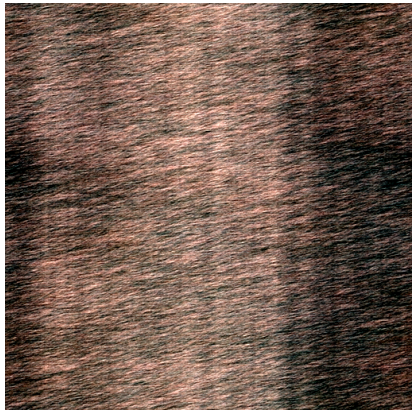
"Wrong *RPN*"

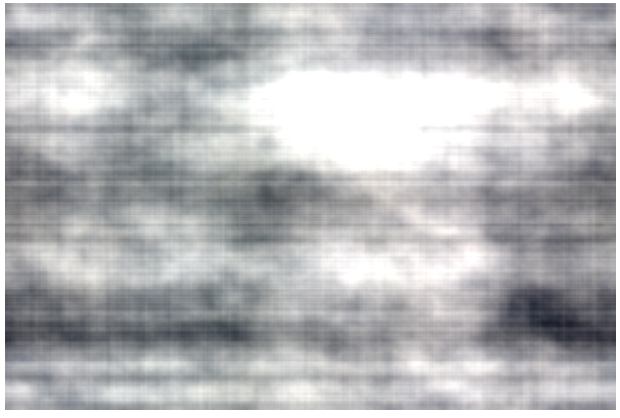
Avoiding artifacts due to non periodicity

- 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 conditions
- Δ_i: discrete Laplacian without periodic 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:

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

Algorithm to compute the periodic component:

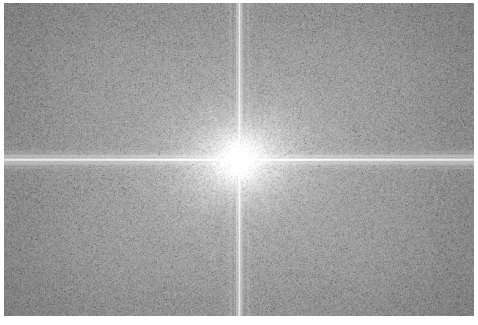
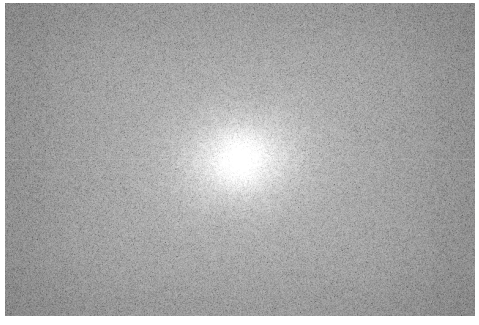
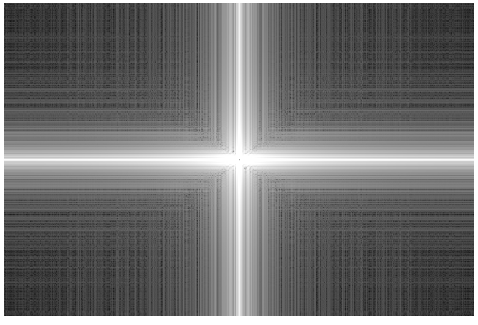
1. Compute $\Delta_i h$ the discrete Laplacian of h.
2. Compute $m = \text{mean}(h)$.
3. Compute $\widehat{\Delta_i h}$ the DFT of $\Delta_i h$ using the forward FFT.
4. Compute the DFT $\hat{p}$ of p defined by

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

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

Periodic component: effects on the Fourier modulus

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

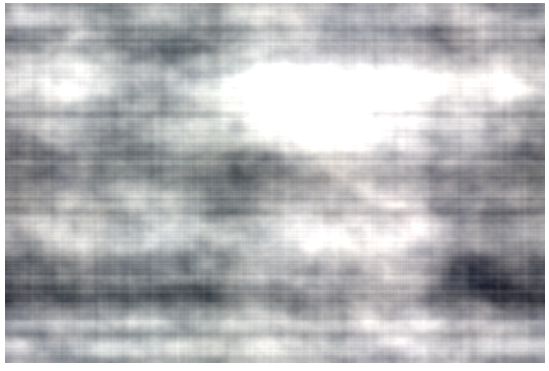
	Image h	Periodic component $p = \mathrm{per}(h)$	Smooth component $s = h - p$ $(+m)$
Images			
Fourier modulus			

- 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 $p = \mathrm{per}(h)$ 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

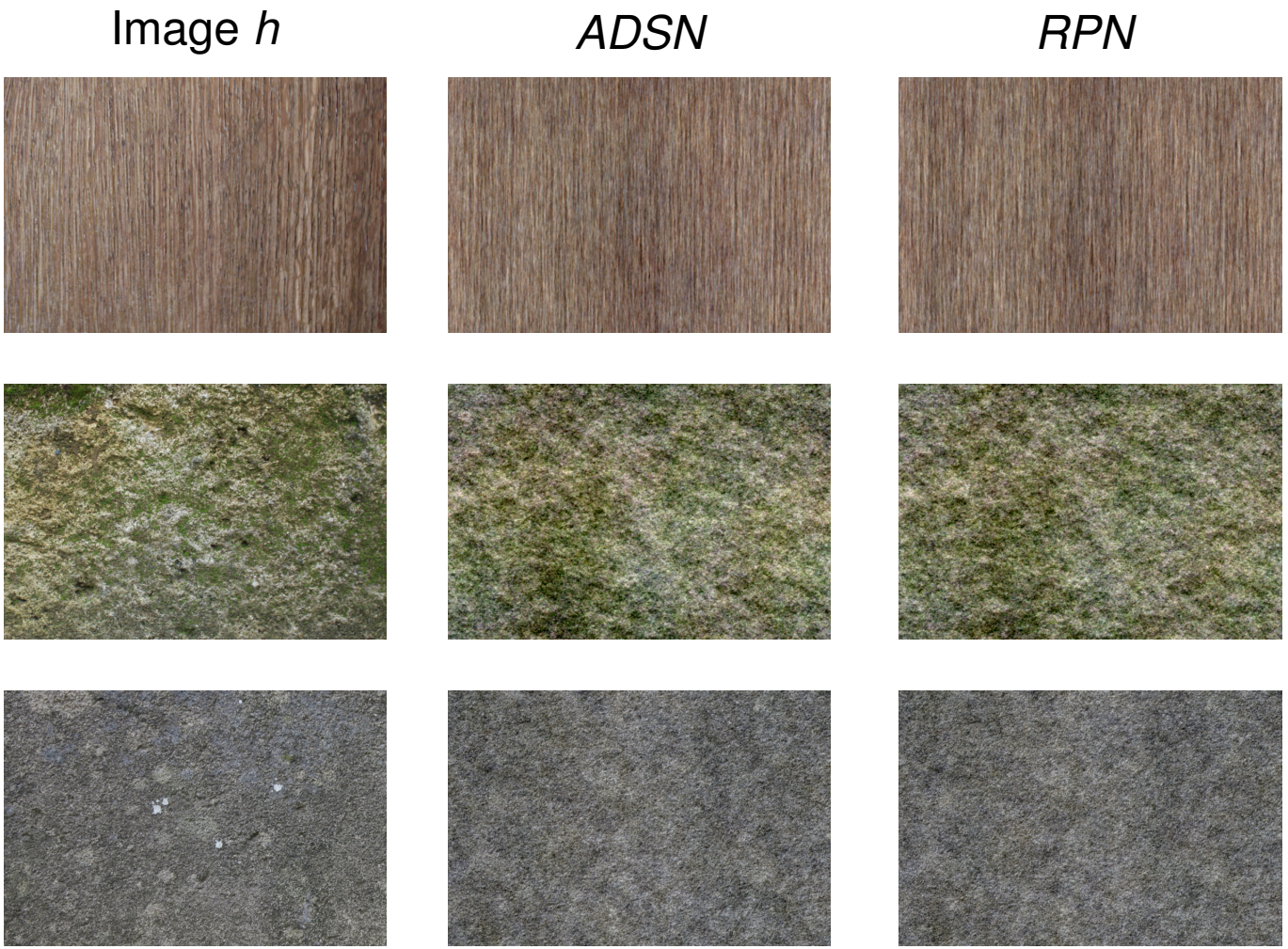
RPN 3

- [ON LINE DEMO]

Outline

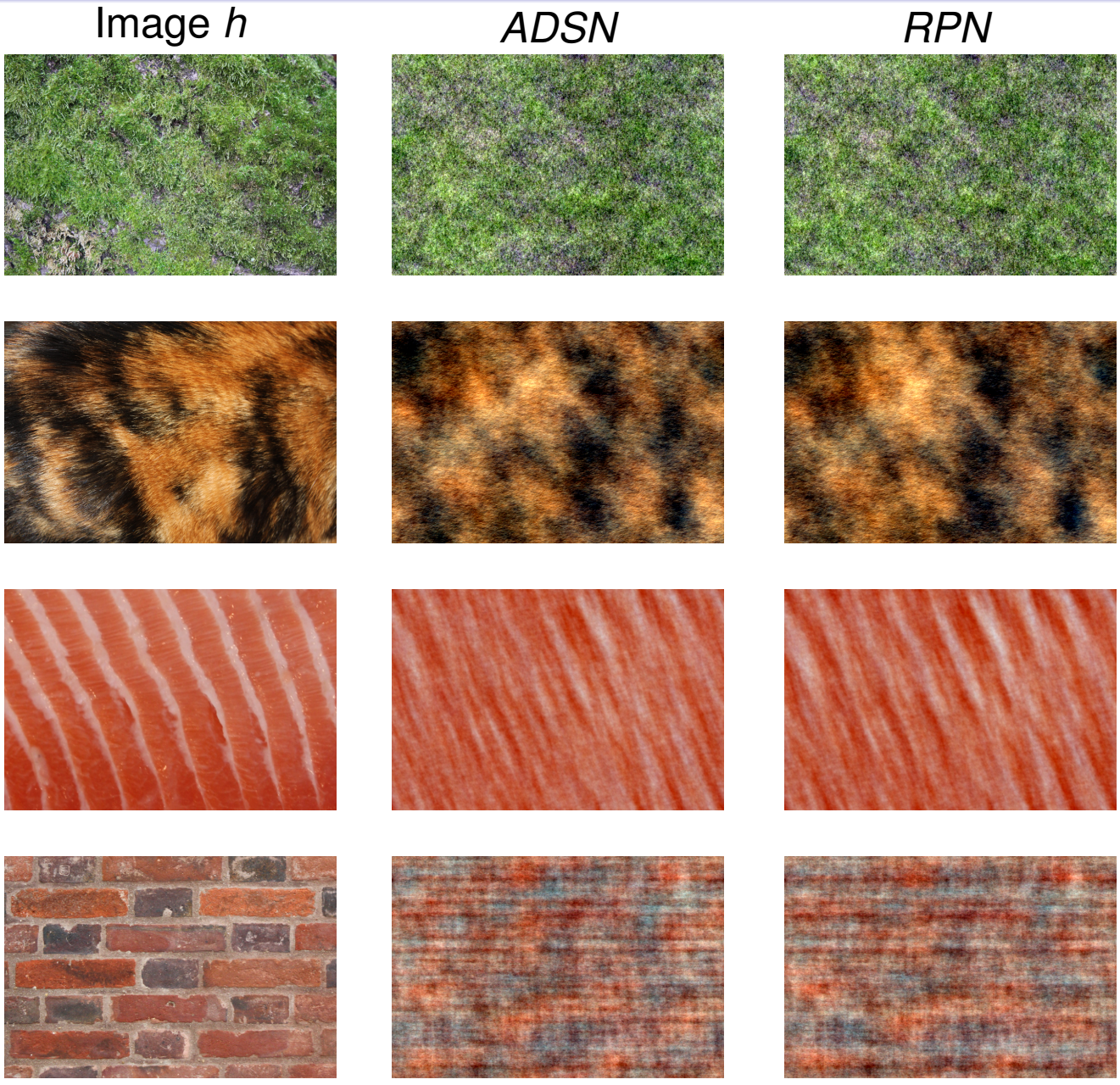
Numerical results: similarity of the textures

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



- Both algorithms produce visually similar textures.

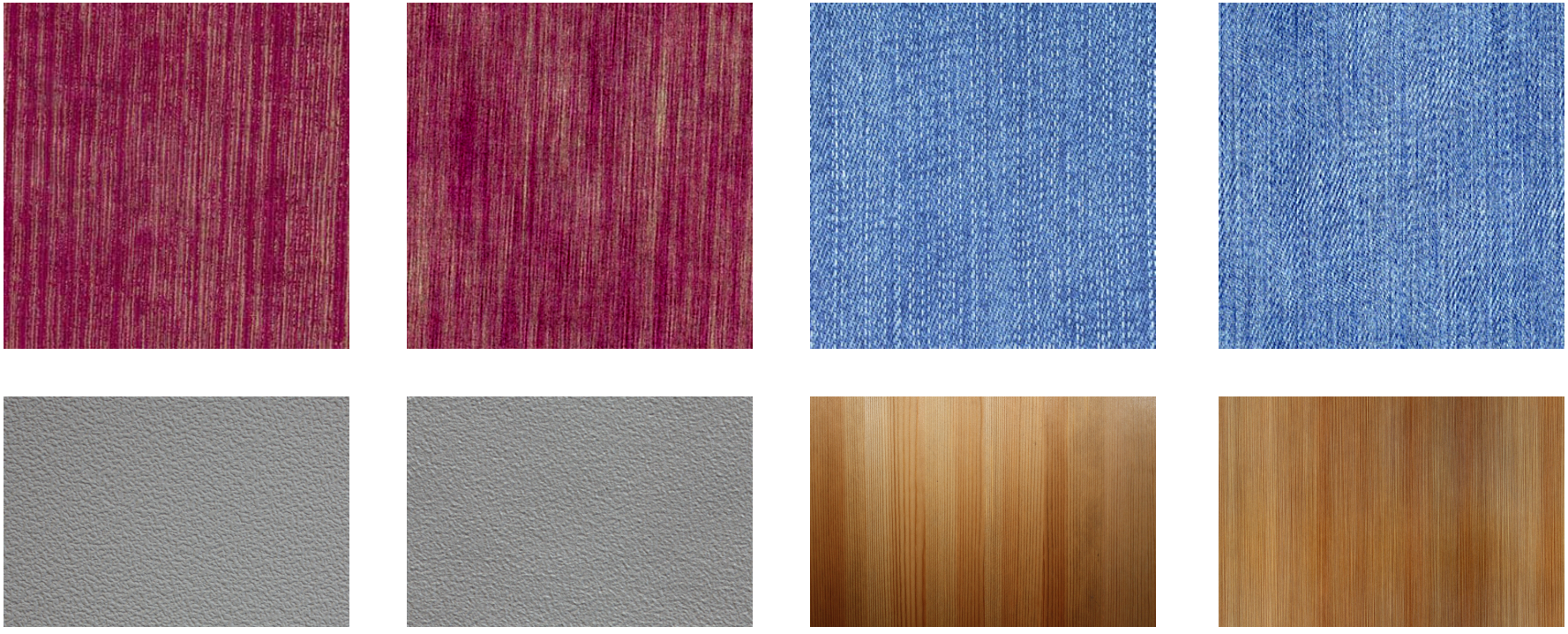
Numerical results: non random phase textures



Some other examples of well-reproduced textures...

- We only display the *RPN* result.

Image *h* | *RPN* | Image *h* | *RPN*



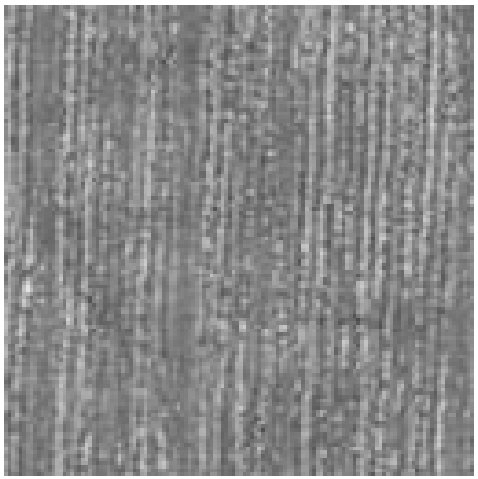
- Much more examples of success and failures on the IPOL webpage: `http://www.ipol.im/pub/algo/ggm_random_phase_texture_synthesis/`

Outline

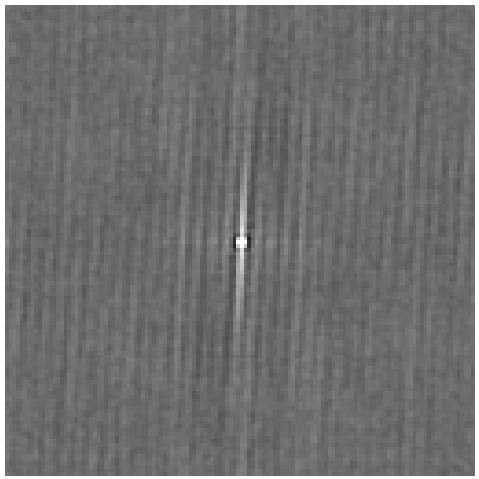
Texton associated with a texture

We work here with gray-level images.

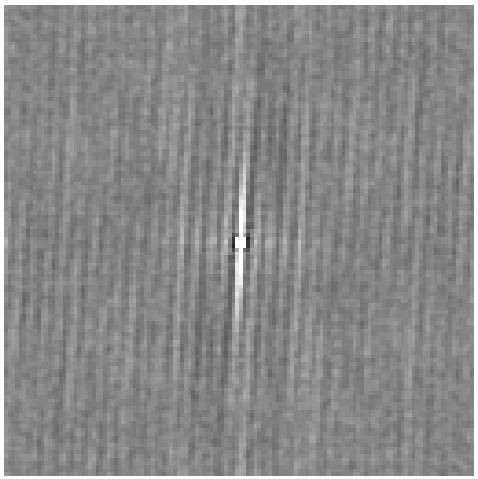
- RPN and ADSN models associated with h only depends of the Fourier modulus of h.
- Definition: The *texton* t_h associated with h is the image with the same modulus as h and with zero phase [Desolneux et al, 2012].



Input h



Texton t_h (log scale)



Texton t_h (thresholded)

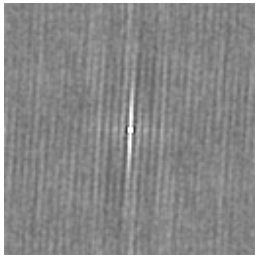
- Concentrated in zero: Compact representation of the texture model
- Interesting tool for analysis:

Same texton = same Gaussian texture

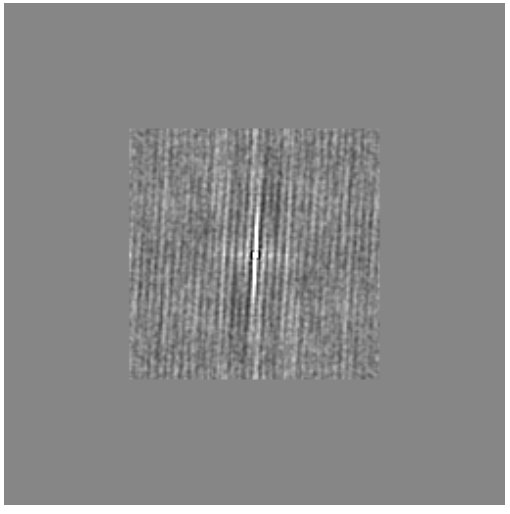
Texton for synthesizing textures having arbitrary large size

- One computes an extended texton (the texton is smallest at the boundary than the original image) :

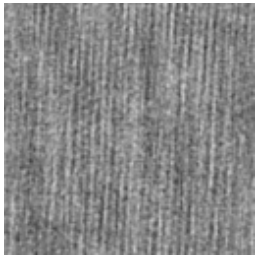
$$\tilde{t}_h = m + r(t_h - m)\mathbb{1}_\Omega$$



Texton t_h



Extended texton $\tilde{t}_h$

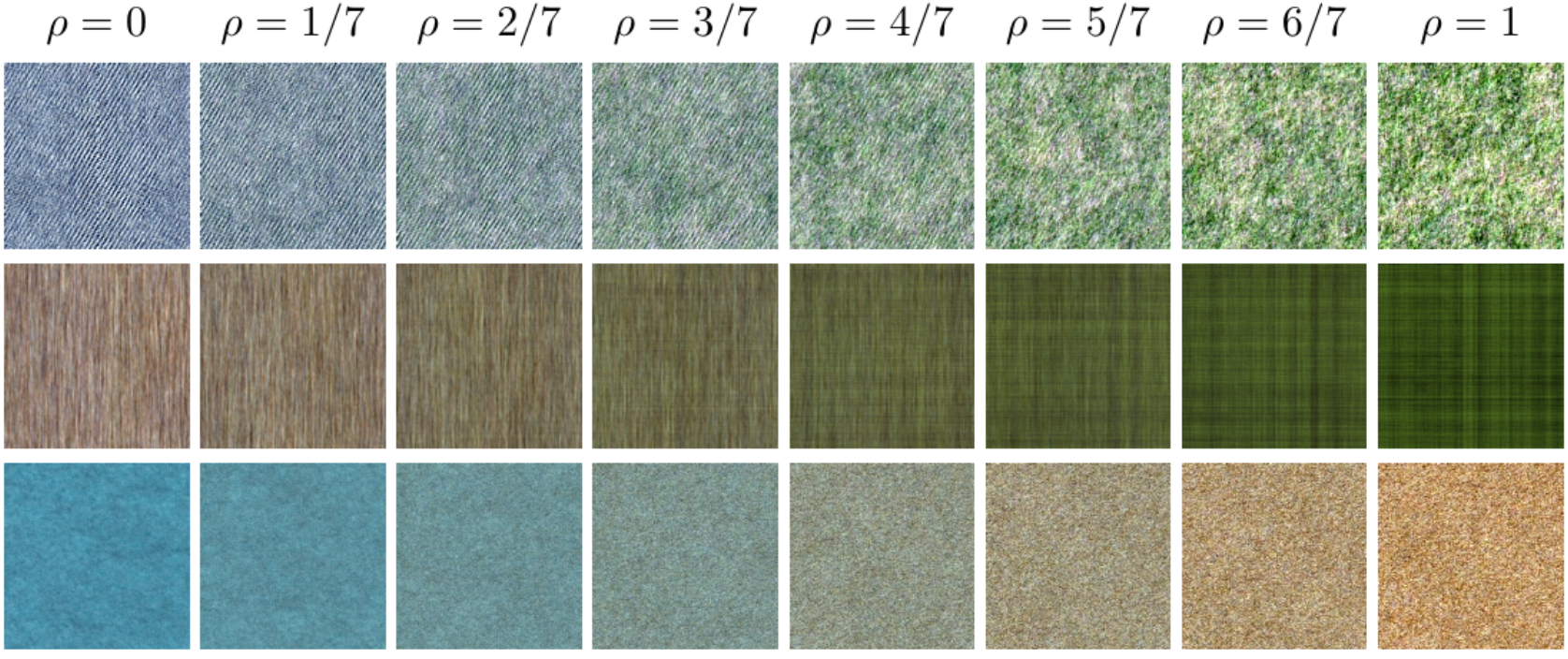


$ADSN(t_h)$

$ADSN(\tilde{t}_h)$

Mixing of Gaussian textures

- Using optimal transport distance, one can define barycenters between Gaussian texture models.
- This gives a practical and rigorous solution for Gaussian texture mixing [Xia et al., 2014].



- For gray-level images, this is simply obtained in averaging the textons...

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).

Bibliographic references I


A. Desolneux, L. Moisan, and S. Ronsin, *A compact representation of random phase and Gaussian textures*, in *ICASSP'12*


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


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


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


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.

Bibliographic references II

G.-S. Xia, S. Ferradans, G. Peyre, J-F. Aujol, *Synthesizing and mixing stationary Gaussian texture models*, SIAM Journal on Imaging Science (SIIMS), 2014.