

Image texture synthesis using spot noise and phase randomization

Bruno Galerne

`bruno.galerie@parisdescartes.fr`

MAP5, Université Paris Descartes

Cours traitement d'images en ligne,
Vendredi 28 octobre 2011

Joint work with

Yann Gousseau (LTCI, Télécom ParisTech)

Jean-Michel Morel (CMLA, ENS Cachan)

Outline

- 1 Texture synthesis
- 2 Discrete Fourier transform of digital images
- 3 Random phase noise (RPN)
- 4 Asymptotic discrete spot noise (ADSN)
- 5 *RPN* and *ADSN* as texture synthesis algorithms
- 6 Numerical experiments
- 7 Propositions of IPOL projects

Outline

- 1 Texture synthesis
- 2 Discrete Fourier transform of digital images
- 3 Random phase noise (RPN)
- 4 Asymptotic discrete spot noise (ADSN)
- 5 *RPN* and *ADSN* as texture synthesis algorithms
- 6 Numerical experiments
- 7 Propositions of IPOL projects

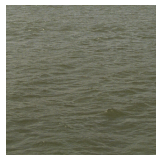
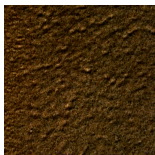
What is a texture?

A minimal definition of a **texture** image is an “image containing repeated patterns” [Wei et al., 2009].

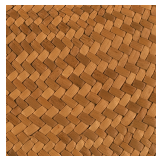
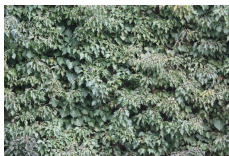
The family of patterns reflects a certain amount of randomness, depending on the nature of the texture.

Two main subclasses:

- The **micro-textures**.



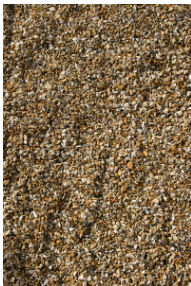
- The **macro-textures**, constituted of small but discernible objects.



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



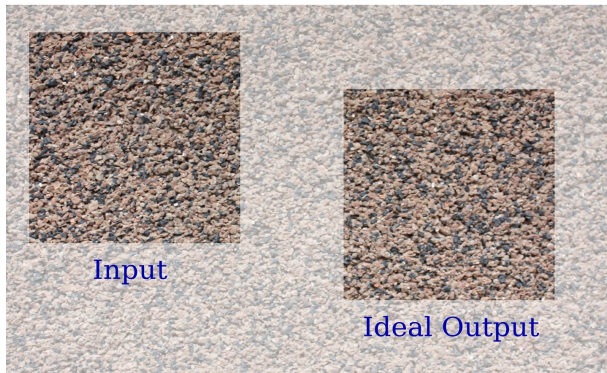
Macro-texture



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
- Numbers of parameters

(For more details, we refer to Yann Gousseau’s course.)

Heeger-Bergen algorithm [Heeger and Bergen, 1995]

Statistical constraints:

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



Efros-Leung algorithm [Efros and Leung, 1999]

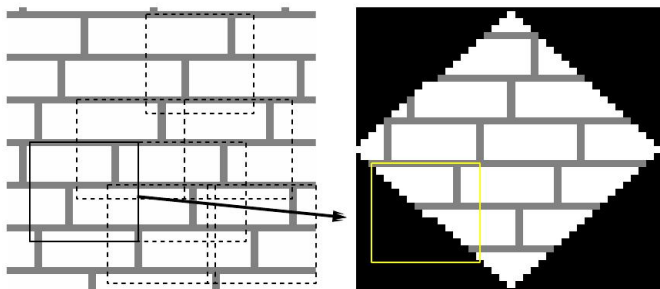
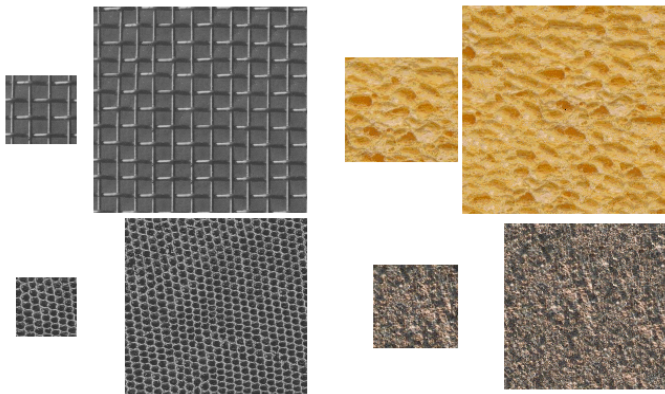


Figure 1. Algorithm Overview. Given a sample texture image (left), a new image is being synthesized one pixel at a time (right). To synthesize a pixel, the algorithm first finds all neighborhoods in the sample image (boxes on the left) that are similar to the pixel's neighborhood (box on the right) and then randomly chooses one neighborhood and takes its center to be the newly synthesized pixel.

Efros-Leung algorithm [Efros and Leung, 1999]

Some successful results:



Known problems:

- Can produce “garbage” results.
- Exhaustive searching is really slow.

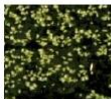
Wei-Levoy algorithm [Wei and Levoy, 2000]

Similar to Efros-Leung, but:

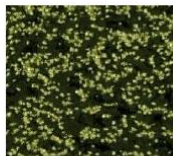
- Neighborhood always have the same size.
⇒ Should be easier to accelerate.
- Multi-resolution procedure.
⇒ More stable than Efros-Leung (?).



(a)



(b)



(c)



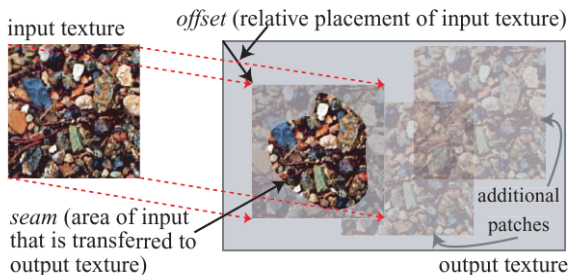
(d)



Patch based algorithms

Instead of synthesizing pixel by pixel one can synthesize patch by patch. Several algorithms using this idea will be proposed as **projects**:

- **Image quilting** [Efros, Freeman, 2001]: copy of squared patches and search of a “seamless cut” by dynamical programming.
- **Graphcut textures** [Kwatra *et al.*, 2003]: copy of large patches and search of a “seamless cut” by graphcut techniques.
- **Texture optimization** [Kwatra *et al.*, 2005]: merging of large patches by energy minimization.



Graphcut textures

Texture synthesis by phase randomization

What about the **random phase noise** presented today?

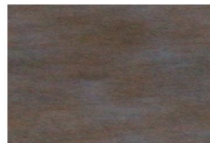
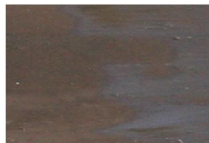
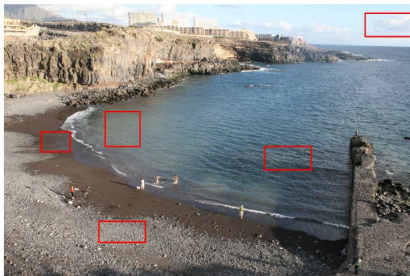
[Galerie, Gousseau and Morel, 2011 (a)]

[Galerie, 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.

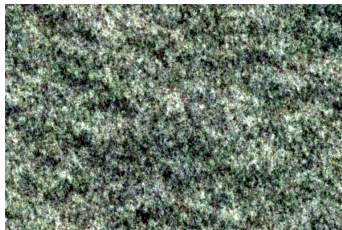
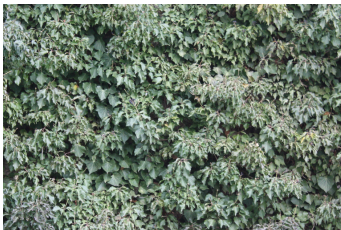
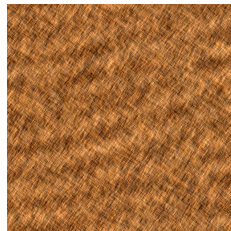
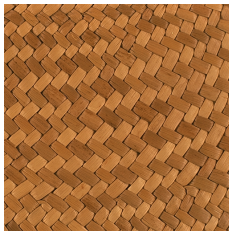
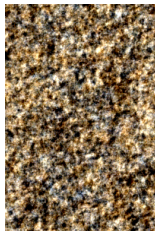
Texture synthesis by phase randomization

- Successful examples with micro-textures:



Texture synthesis by phase randomization

- Failure examples with macro-textures:



Outline

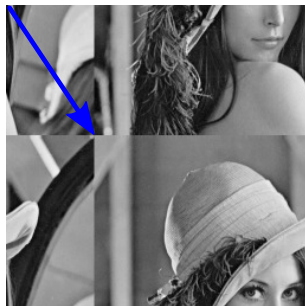
- 1 Texture synthesis
- 2 Discrete Fourier transform of digital images**
- 3 Random phase noise (RPN)
- 4 Asymptotic discrete spot noise (ADSN)
- 5 *RPN* and *ADSN* as texture synthesis algorithms
- 6 Numerical experiments
- 7 Propositions of IPOL projects

Framework

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

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

- Consequence: Translation of an image:



Discrete Fourier transform of digital images

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

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

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:

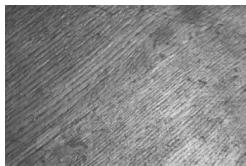
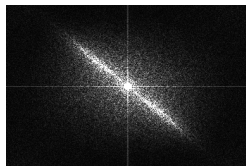


Image u



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

Modulus and phase of a digital image

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

Image 1

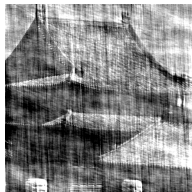


Image 2

Modulus of 1
& phase of 2



Modulus of 2
& phase of 1



- Geometric contours are mostly contained in the phase.

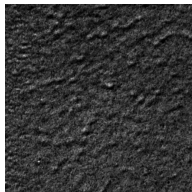
Modulus and phase of a digital image

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

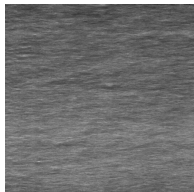
Image 1



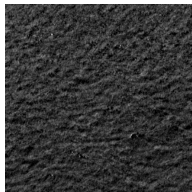
Image 2



Modulus of 1
& phase of 2



Modulus of 2
& phase of 1



- Textures are mostly contained in the modulus.

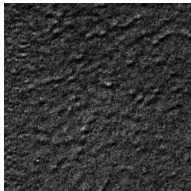
Modulus and phase of a digital image

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

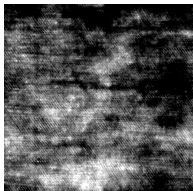
Image 1



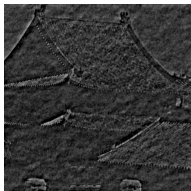
Image 2



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

- 1 Texture synthesis
- 2 Discrete Fourier transform of digital images
- 3 Random phase noise (RPN)**
- 4 Asymptotic discrete spot noise (ADSN)
- 5 *RPN* and *ADSN* as texture synthesis algorithms
- 6 Numerical experiments
- 7 Propositions of IPOL projects

Random phase textures

- First Julesz theory: The Fourier modulus is a key feature for the perception of a texture.
- 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 $\theta : \hat{\Omega} \rightarrow \mathbb{R}$ is a **random phase** if

① **Symmetry:** θ is odd:

$$\forall (\mathbf{s}, t) \in \hat{\Omega}, \theta(-\mathbf{s}, -t) = -\theta(\mathbf{s}, t).$$

② **Distribution:** Each component $\theta(\mathbf{s}, t)$ is

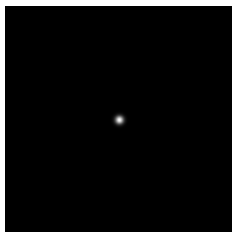
- uniform over the interval $]-\pi, \pi]$ if $(\mathbf{s}, t) \notin \{(0, 0), (\frac{M}{2}, 0), (0, \frac{N}{2}), (\frac{M}{2}, \frac{N}{2})\}$,
- uniform over the set $\{0, \pi\}$ otherwise.

③ **Independence:** For each subset $\mathcal{S} \subset \hat{\Omega}$ that does not contain distinct symmetric points, the r.v. $\{\theta(\mathbf{s}, t) | (\mathbf{s}, t) \in \mathcal{S}\}$ are independent.

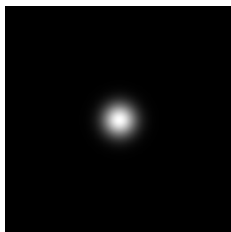
- *Random phase textures* constitute a “limited” subclasse of the set of textures.

Random Phase Noise (RPN)

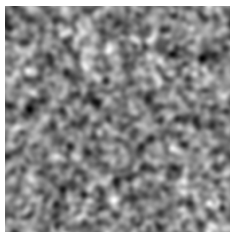
- Texture synthesis algorithm: **random phase noise (RPN)**: [van Wijk, 1991]
- 1 Compute the DFT \hat{h} of the input h .
- 2 Compute a random phase θ using a pseudo-random number generator.
- 3 Set $\hat{Z} = |\hat{h}| e^{i\theta}$ (or $\hat{Z} = \hat{h}e^{i\theta}$).
- 4 Return Z the inverse DFT of \hat{Z} .



Original image h



Modulus $|\hat{h}|$



RPN associated with
 h

Outline

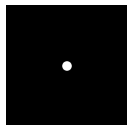
- 1 Texture synthesis
- 2 Discrete Fourier transform of digital images
- 3 Random phase noise (RPN)
- 4 Asymptotic discrete spot noise (ADSN)**
- 5 *RPN* and *ADSN* as texture synthesis algorithms
- 6 Numerical experiments
- 7 Propositions of IPOL projects

Discrete spot noise [van Wijk, 1991]

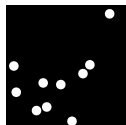
Texture model with two main characteristics: linear superimposition and invariance by translation of the objects.

- 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

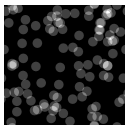
$$f_n(x) = \sum_{k=1}^n h(x - X_k).$$



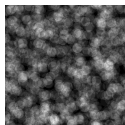
Spot h



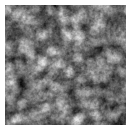
$n = 10$



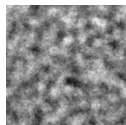
$n = 10^2$



$n = 10^3$



$n = 10^4$



$n = 10^5$

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

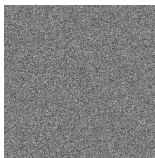
$$C_h(x, y) = \frac{1}{MN} \sum_{t \in \Omega} (h(x - t) - m)(h(y - t) - m), \quad (x, y) \in \Omega.$$

- 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: $\frac{f_n - m}{\sqrt{n}}$ converges towards the Gaussian random vector defined on Ω with zero mean and covariance C_h .

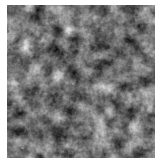
Definition of ADSN: the ADSN associated with h is the Gaussian vector $\mathcal{N}(0, C_h)$.

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

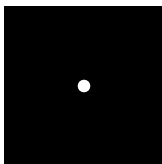


Gaussian vector: pixels have Gaussian distribution and are correlated

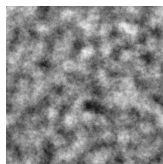
Convolution product: $(f * g)(x) = \sum_{y \in \Omega} f(x - y)g(y)$, $x \in \Omega$.

Simulation of the ADSN:

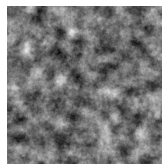
- Let $h \in \mathbb{R}^{M \times N}$ be an image, m be the mean of h and X be a Gaussian white noise image.
- The random image $\frac{1}{\sqrt{MN}} (h - m) * X$ is the ADSN associated with h .



Spot h



DSN, $n = 10^5$

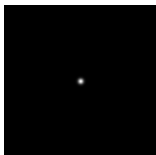


ADSN

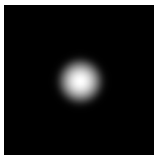
Differences between *RPN* and *ADSN*

Proposition:

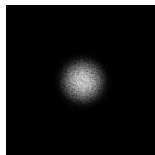
- *RPN* and *ADSN* both have a random phase.
- The Fourier modulus of *RPN* is the one of h .
- The Fourier modulus of *ADSN* is the pointwise multiplication between $|\hat{h}|$ and a Rayleigh noise.



Spot h



RPN Modulus



ADSN Modulus

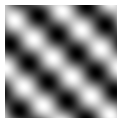
- ***RPN* and *ADSN* are two different processes.**



Spot h



RPN



An *ADSN*
realization



Another *ADSN*
realization

Outline

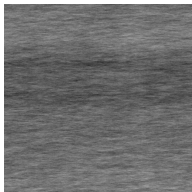
- 1 Texture synthesis
- 2 Discrete Fourier transform of digital images
- 3 Random phase noise (RPN)
- 4 Asymptotic discrete spot noise (ADSN)
- 5 *RPN* and *ADSN* as texture synthesis algorithms**
- 6 Numerical experiments
- 7 Propositions of IPOL projects

RPN and *ADSN* associated to texture images

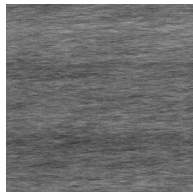
- Some textures are relatively well reproduced by *RPN* and *ADSN*.



Original image



RPN



ADSN

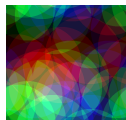
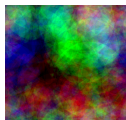
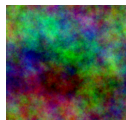
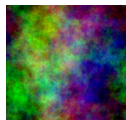
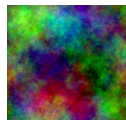
- ... But several developments are necessary to derive texture synthesis algorithms from sample.

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

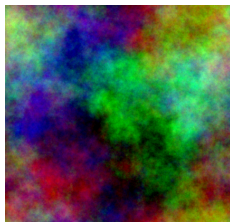
- **Color RPN:** By analogy, the *RPN* associated with a color image $h = (h_r, h_g, h_b)$ is the color image obtained by **adding the same random phase** to the Fourier transform of each color channel.

Original image h



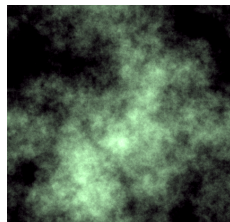
$$\hat{h} = \begin{pmatrix} |\hat{h}_R| e^{i\varphi_R} \\ |\hat{h}_G| e^{i\varphi_G} \\ |\hat{h}_B| e^{i\varphi_B} \end{pmatrix}$$

Color *RPN*



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

“Wrong *RPN*”: each channel has the same random phase



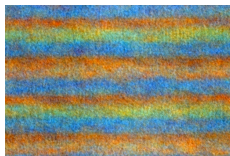
$$\hat{z}_W = \begin{pmatrix} |\hat{h}_R| e^{i\theta} \\ |\hat{h}_G| e^{i\theta} \\ |\hat{h}_B| e^{i\theta} \end{pmatrix}$$

Extension to color images

- Another example with a real-world texture.



Original image h

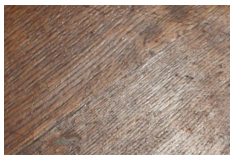


Color RPN

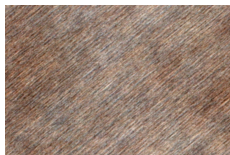


“Wrong RPN ”

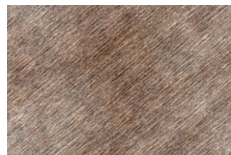
- Preserving the original phase displacement between the color channels is essential for color consistency.
- ...however for most monochromatic textures, there is no huge difference.



Original image h



Color RPN

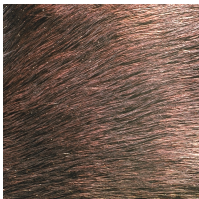


“Wrong RPN ”

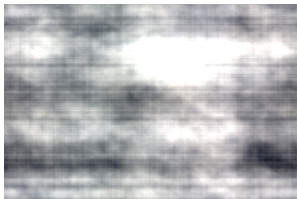
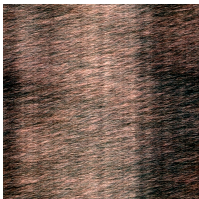
Avoiding artifacts due to non periodicity

- Both *ADSN* and *RPN* algorithms are based on the fast Fourier transform (FFT).
⇒ 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-connectivity:

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

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

Poisson problem:

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

In the **Fourier domain**, this system becomes:

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

Algorithm to compute the periodic component:

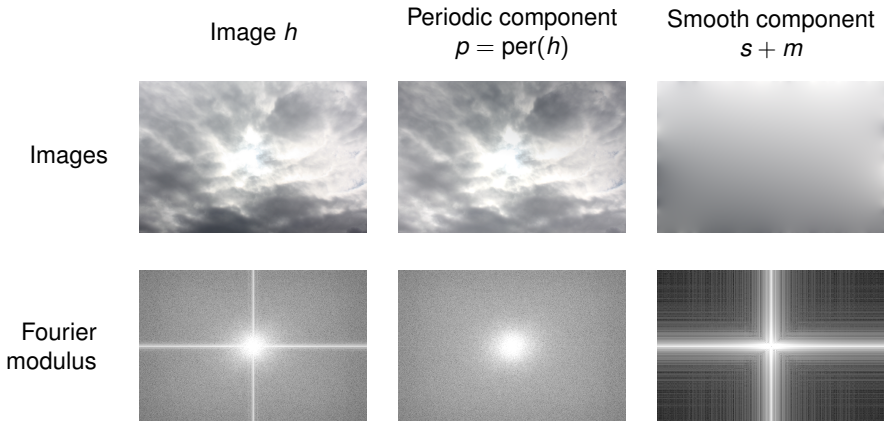
- 1 Compute $\Delta_i h$ the discrete Laplacian of h .
- 2 Compute $m = \text{mean}(h)$.
- 3 Compute $\widehat{\Delta_i h}$ the DFT of $\Delta_i h$ using the forward FFT.
- 4 Compute the DFT \hat{p} of p defined by

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

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

Periodic component: effects on the Fourier modulus

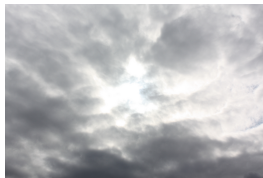
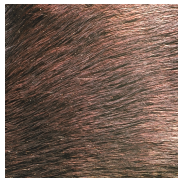
- p is “visually close” to h (same Laplacian).



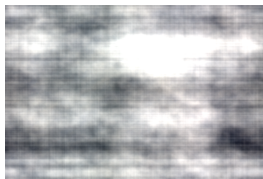
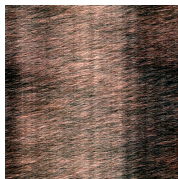
- The application $\text{per} : h \mapsto p$ filters out the “cross structure” of the spectrum.

Avoiding artifacts due to non periodicity

Spot h



$ADSN(h)$



$ADSN(p)$



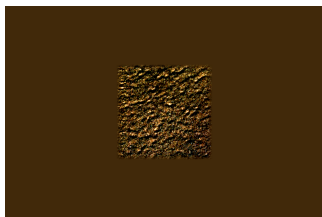
Synthesizing textures having arbitrary large size

To synthesize a texture larger than the original spot h , one computes an “equivalent spot” \tilde{h} :

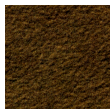
- Copy $p = \text{per}(h)$ in the center of a constant image equal to the mean of h .
- Normalize the variance.
- Attenuate the transition at the inner border.



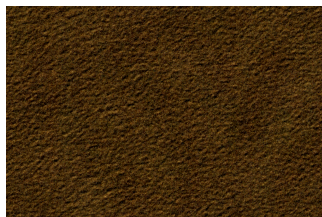
Spot h



Equivalent spot \tilde{h}



$RPN(h)$



$RPN(\tilde{h})$

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

- [\[ON LINE DEMO\]](#)

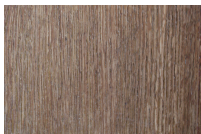
Outline

- 1 Texture synthesis
- 2 Discrete Fourier transform of digital images
- 3 Random phase noise (RPN)
- 4 Asymptotic discrete spot noise (ADSN)
- 5 *RPN* and *ADSN* as texture synthesis algorithms
- 6 Numerical experiments**
- 7 Propositions of IPOL projects

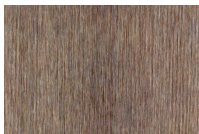
Numerical results: similarity of the textures

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

Image h



ADSN



RPN



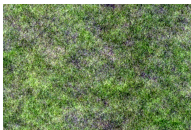
- Both algorithms produce visually similar textures.

Numerical results: non random phase textures

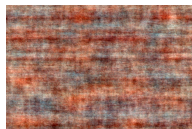
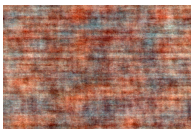
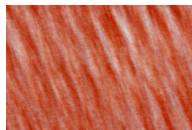
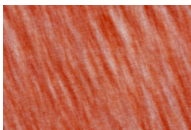
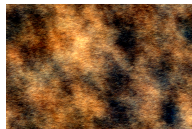
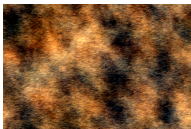
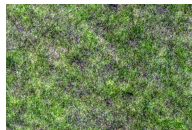
Image h



ADSN



RPN



Some other examples of well-reproduced textures...

- We only display the *RPN* result.

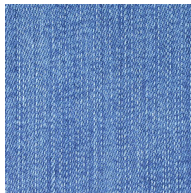
Image h



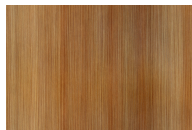
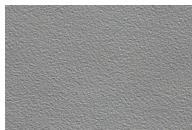
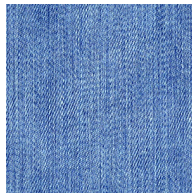
RPN



Image h



RPN



- Much more examples of success and failures on the IPOL webpage:
http://www.ipol.im/pub/alg/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 contrast with noise models from computer graphics).

Outline

- 1 Texture synthesis
- 2 Discrete Fourier transform of digital images
- 3 Random phase noise (RPN)
- 4 Asymptotic discrete spot noise (ADSN)
- 5 *RPN* and *ADSN* as texture synthesis algorithms
- 6 Numerical experiments
- 7 Propositions of IPOL projects

Patch based algorithms

Three classic algorithms are proposed in project:

- **Image quilting** [Efros, Freeman, 2001]: copy of squared patches and search of a “seamless cut” by dynamical programming.
- **Graphcut textures** [Kwatra *et al.*, 2003]: copy of large patches and search of a “seamless cut” by graphcut techniques.
- **Texture optimization** [Kwatra *et al.*, 2005]: merging of large patches by energy minimization.

Image quilting [Efros, Freeman, 2001]

- Copy of squared patches
- Search of a “seamless cut” by dynamical programming.
- Matlab codes available.

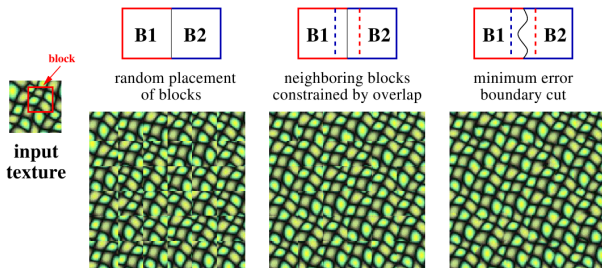


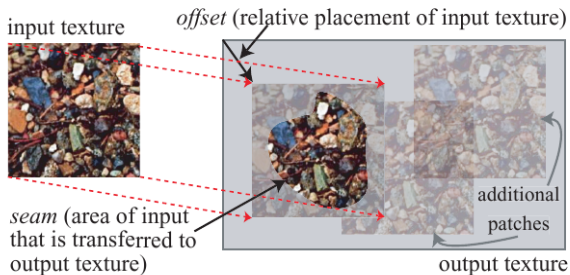
Image quilting

Project:

- Precise description of the algorithm
- Experimental study: influence of parameters, growing garbage,...

Graphcut textures [Kwatra et al., 2003]

- Copy of large neighborhoods.
- Search of a “seamless cut” by graphcut.



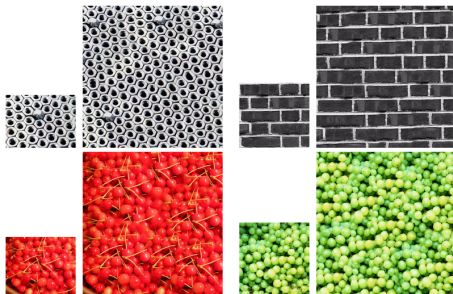
Graphcut textures

Project:

- Precise description of the algorithm
- Discuss implementation in C

Texture optimization [Kwatra *et al.*, 2005]

- Merging of neighborhoods by energy minimization.



Texture optimization

One or two Projects:








- Precise description of the algorithm
- **Project 1:** Discuss implementation in C (highlight parallel programming issues)
- **Project 2:** Compare with G. Peyré's framework [Peyre, 2009] and propose a close algorithm based on G. Peyré's matlab codes (see his Numerical Tours)

Propositions of IPOL Projects







For all the projects, please contact me by email:

`bruno.galerie@parisdescartes.fr`

Bibliographic references I

-  A. A. Efros and W. T. Freeman, *Image quilting for texture synthesis and transfer*, SIGGRAPH '01, 2001
-  A. A. Efros and T. K. Leung, *Texture synthesis by non-parametric sampling*, ICIP 1999, 1999
-  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
-  V. Kwatra and A. Schödl and I. Essa and G. Turk and A. Bobick, *Graphcut textures: image and video synthesis using graph cuts*, SIGGRAPH '03, 2003
-  V. Kwatra and I. Essa and A. Bobick and N. Kwatra *Texture optimization for example-based synthesis* SIGGRAPH '05, 2005

Bibliographic references II

-  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, vol. 69, May 1981, pp. 529–541.
-  G. Peyré, *Sparse Modeling of Textures*, J. Math. Imag. Vis., 2009
-  L.-Y. Wei, S. Lefebvre, V. Kwatra, and G. Turk. *State of the art in example-based texture synthesis*, Eurographics 2009, 2009.
-  L. Y. Wei and M. Levoy, *Fast texture synthesis using tree-structured vector quantization* SIGGRAPH '00, 2000.
-  J. J. van Wijk, *Spot noise texture synthesis for data visualization*, SIGGRAPH '91, 1991.