Champs aléatoires pour la synthèse de textures
Random fields for texture synthesis

Bruno Galerne
bruno.galerne@univ-orleans.fr

Journées communes **GéoSto MIA 2022**
22-23 sept. 2022 Saint-Étienne-du-Rouvray (France)

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

Texture synthesis

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.

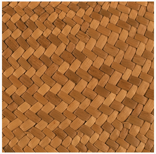
Two main subclasses:

- 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

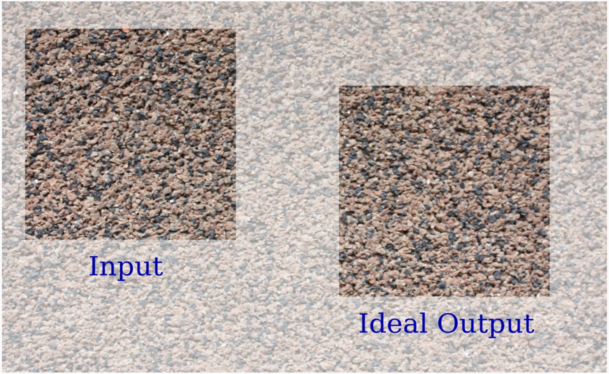
- a micro-texture,
- a macro-texture,
- a collection of individual objects.

Micro-texture

Macro-texture

Some pebbles

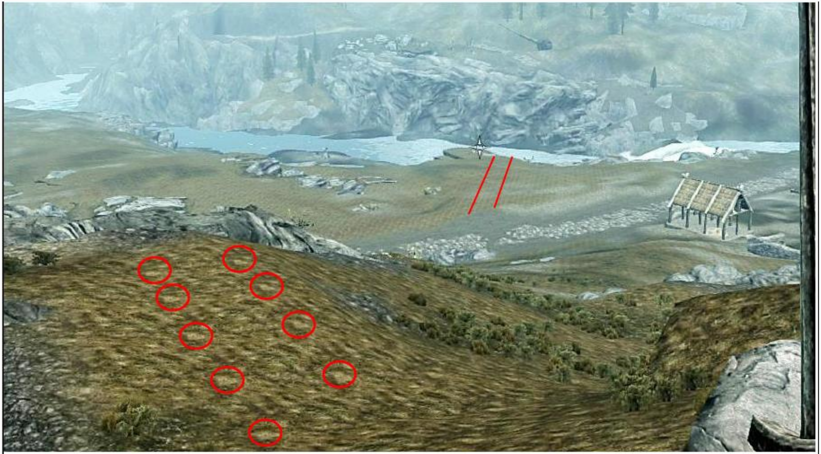
Texture Synthesis: Given an input texture image, produce an output texture image being both visually similar to and pixel-wise different from the input texture.



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

Texture synthesis: Motivation

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



2011: Skyrim (Bethesda) *screenshot from Three Parts Theory*

Texture synthesis algorithms

Two main kinds of algorithm:

Texture synthesis algorithms

Two main kinds of algorithm:

1. Texture synthesis using statistical constraints:

   Algorithm:

   1.1 Extract some meaningful "statistics" from the input image (e.g. distribution of colors, of Fourier coefficients, of wavelet coefficients,...).
   1.2 Compute a "random" output image having the same statistics: start from a white noise and alternatively impose the "statistics" of the input.

   Properties:

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

Texture synthesis algorithms

Two main kinds of algorithm:

1. Texture synthesis using statistical constraints:

   Algorithm:

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

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

   Properties:

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

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

   Algorithm:

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

   Properties:

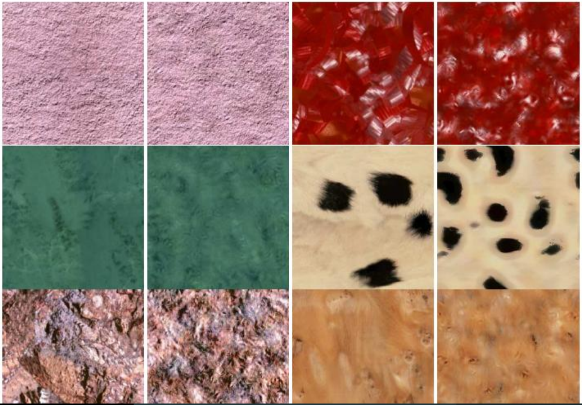
   + Synthesize well macro-textures
   - Can have some speed and stability issue, hard to set parameter, local verbatim copy...

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

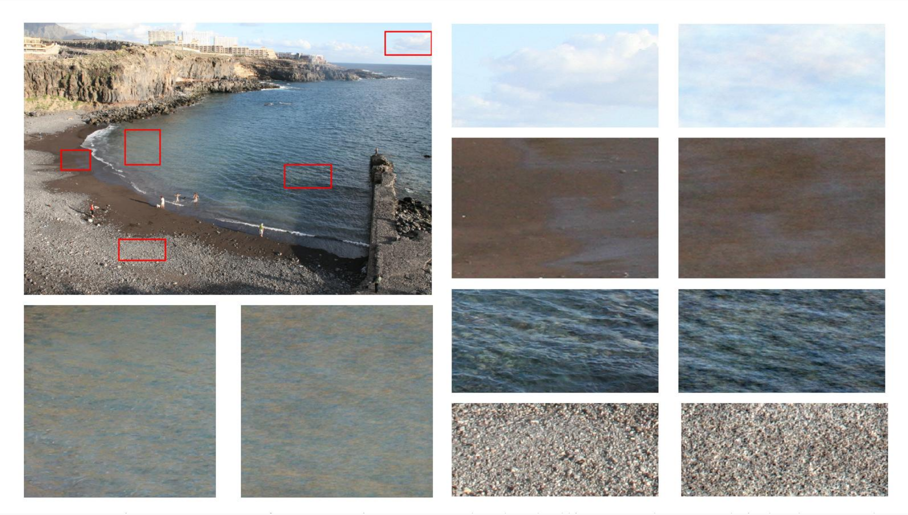
What about the **Random Phase Noise (RPN)** and **Asymptotic Discrete Spot Noise (ADSN)** presented today ?

(Galerne et al., 2011b)
(Galerne et al., 2011a)

- It belongs to the first category: texture synthesis by statistical constraints.
- Here the "statistics" are the moduli of the Fourier coefficients.
- It simply corresponds to a stationary Gaussian random field.

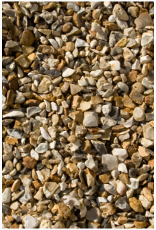
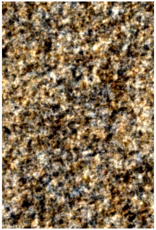
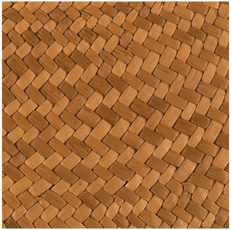
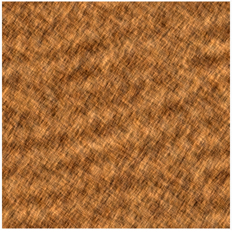
Texture synthesis by phase randomization

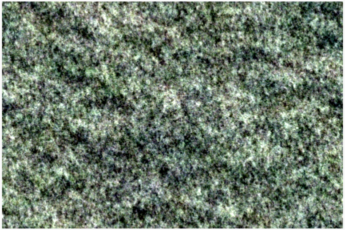
- Successful examples with micro-textures



Texture synthesis by phase randomization

- Failure examples with macro-textures

Discrete Fourier transform of digital images

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:

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

- Consequence: Periodic translations:

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:

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

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.

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.

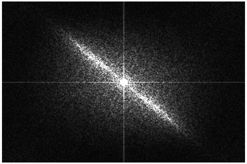
Discrete Fourier transform of digital images

Symmetry property:

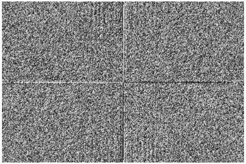
- $|\hat{u}|$: **Fourier modulus** of u is even.
- $\arg(\hat{u})$: **Fourier phase** of u is odd.

Visualization of the DFT:

Image u



Modulus $|\hat{u}|$



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

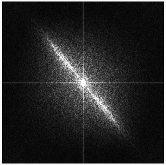
Discrete Fourier transform of digital images

Symmetry property:

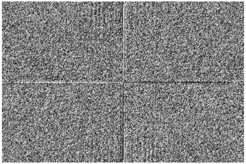
- $|\hat{u}|$: **Fourier modulus** of u is even.
- $\arg(\hat{u})$: **Fourier phase** of u is odd.

Visualization of the DFT:

Image u



Modulus $|\hat{u}|$



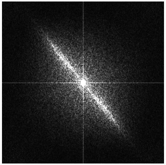
Phase $\arg(\hat{u})$

Symmetry property:

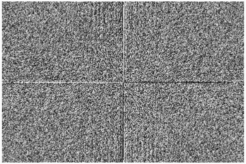
- $|\hat{u}|$: **Fourier modulus** of u is even.
- $\arg(\hat{u})$: **Fourier phase** of u is odd.

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.

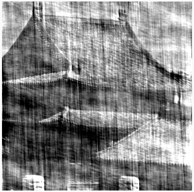
Modulus and phase of a digital image

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

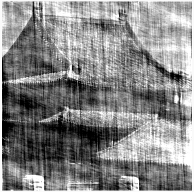
Modulus and phase of a digital image

Exchanging the modulus and the phase of two images: (Oppenheim and Lim, 1981)

Image 1

Image 2

Modulus of 1
& phase of 2



Modulus of 2
& phase of 1

- Geometric contours are mostly contained in the phase.

Modulus and phase of a digital image

Exchanging the modulus and the phase of two images: (Oppenheim and Lim, 1981)

Image 1

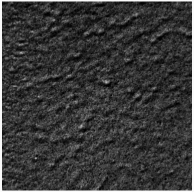
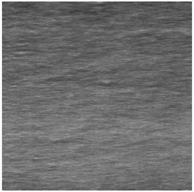


Image 2

Modulus of 1
& phase of 2



Modulus of 2
& phase of 1

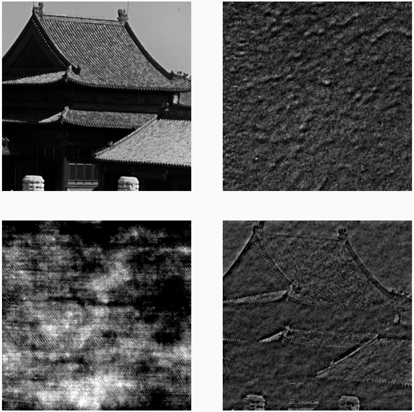
- Textures are mostly contained in the modulus.

Modulus and phase of a digital image

Exchanging the modulus and the phase of two images: (Oppenheim and Lim, 1981)

Image 1

Image 2



Modulus of 1 & phase of 2

Modulus of 2 & phase of 1

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

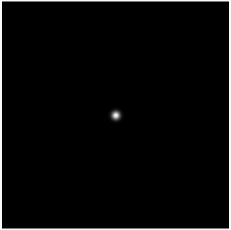
Random phase noise (RPN)

Random phase textures

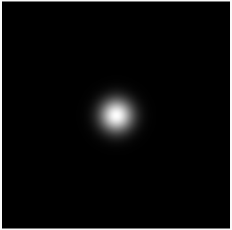
- We call *random phase texture* any image that is perceptually invariant to phase randomization.
- Phase randomization = replace the Fourier phase by a random phase.
- Definition: A random field $\theta : \hat{\Omega} \to \mathbb{R}$ is a ***random phase*** if
  1. Symmetry: θ is odd:
  $$\forall (s,t) \in \hat{\Omega}, \theta(-s,-t) = -\theta(s,t).$$
  2. Distribution: Each component $\theta(s,t)$ is
     - uniform over the interval $]-\pi,\pi]$ if $(s,t) \notin \left\{(0,0), \left(\frac{M}{2},0\right), \left(0,\frac{N}{2}\right), \left(\frac{M}{2},\frac{N}{2}\right)\right\}$,
     - 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.

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

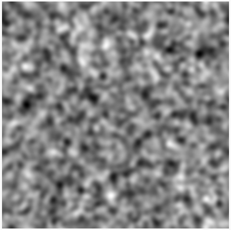
1. Compute the DFT $\hat{h}$ of the input h
2. Compute a random phase θ using a pseudo-random number generator
3. Set $\hat{Z} = \left|\hat{h}\right| e^{i\theta}$ (or $\hat{Z} = \hat{h}e^{i\theta}$)
4. Return Z the inverse DFT of $\hat{Z}$



Original image h



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



RPN associated with h

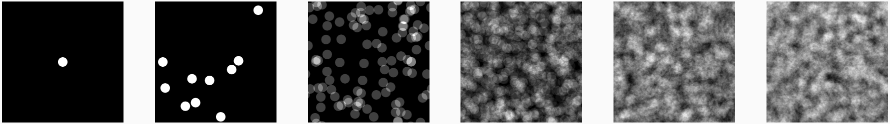
Asymptotic discrete spot noise (ADSN)

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

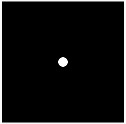
$$f_n(x) = \sum_{k=1}^{n} h(x - X_k).$$

(translations with periodic boundary conditions)

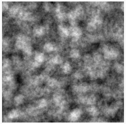


Spot h | $n = 10$ | $n = 10^2$ | $n = 10^3$ | $n = 10^4$ | $n = 10^5$

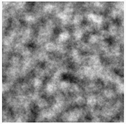
Limit of the DSN model ?



Spot h



$n = 10^4$



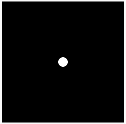
$n = 10^5$

?

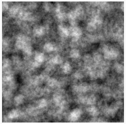
$n = +\infty$

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

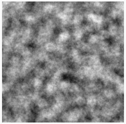
Limit of the DSN model ?



Spot h



$n = 10^4$



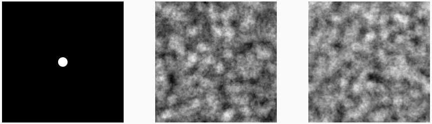
$n = 10^5$

?

$n = +\infty$

- 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 $h(\cdot - X_k)$.

Limit of the DSN model ?



Spot h | $n = 10^4$ | $n = 10^5$ | **?** $n = +\infty$

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

Basics of Gaussian random vectors

Gaussian random vectors in 1D:

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

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

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

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

- The covariance is symmetric and positive

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

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

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

Basics of Gaussian random vectors

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

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

Basics of Gaussian random vectors

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

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

Gaussian random vectors and linear application:

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

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

Basics of Gaussian random vectors

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

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

Gaussian random vectors and linear application:

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

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

Simulation Gaussian random vectors:

Given a mean vector μ and a covariance matrix C :

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

Basics of Gaussian random vectors

Gaussian random vectors in 2D:

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

Stationary random vectors in 2D:

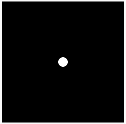
- A random vector Y is stationary if Y and its translations have the same distribution.
- If Y is stationary then $\mathbb{E}(Y)$ is a constant vector ($\mathbb{E}(Y(x)) = \mathbb{E}(Y(y))$) and

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

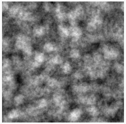
is a “circulant matrix”. Then the covariance can be stored in a single image $c(x) = C(x,0)$ so that

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

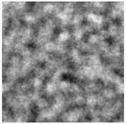
Limit of the DSN model ?



Spot h



$n = 10^4$

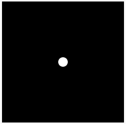


$n = 10^5$

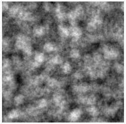
?

$n = +\infty$

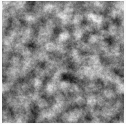
- For texture synthesis we are more particularly interested in the limit of the *DSN*: the ***asymptotic discrete spot noise*** (***ADSN***).



Spot h



$n = 10^4$



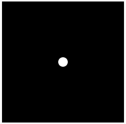
$n = 10^5$

?

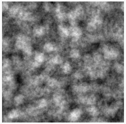
$n = +\infty$

- 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 $h(\cdot - X_k)$.

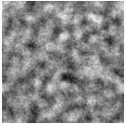
Limit of the DSN model ?



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*:

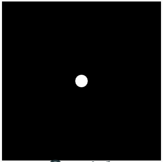
- The *ADSN* associated with h is the Gaussian vector $\mathcal{N}(0, C_h)$.

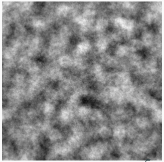
Definition of *ADSN*: the *ADSN* associated with h is the Gaussian vector $\mathcal{N}(0, C_h)$.

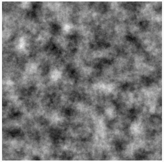
Convolution product: $(f * g)(x) = \sum_{y \in \Omega} f(x-y) g(y), \; x \in \Omega$.

Simulation of the *ADSN*:

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


Spot h


DSN, $n = 10^5$


ADSN

ADSN Simulation

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

- Y is obtained from X in applying a linear map. Since X is a Gaussian vector, Y is also a Gaussian vector.

ADSN Simulation

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

- Y is obtained from X in applying a linear map. Since X is a Gaussian vector, Y is also a Gaussian vector.
- One just needs to show that $\mathbb{E}(Y(x)) = 0$ and $\mathrm{Cov}(Y(x), Y(y)) = C_h(x, y)$.

ADSN Simulation

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

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

ADSN Simulation

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

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

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

ADSN Simulation

Simulation Gaussian random vectors:

Given a mean vector μ and a covariance matrix C :

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

Remark:

- Here with

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

  we just showed that the linear operator

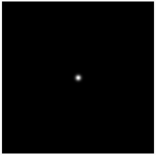
$$A = \text{“convolution by } \frac{1}{\sqrt{MN}}(h - m)\text{”}$$

  satisfies $AA^T = C_h$ (as would the Cholesky decomposition).

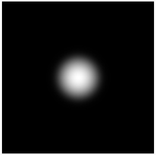
Differences between *RPN* and *ADSN*

Proposition:

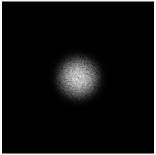
- *RPN* and *ADSN* both have a random phase.
- The Fourier modulus of *RPN* is the one of h.
- The Fourier modulus of *ADSN* is the pointwise multiplication between $\left|\hat{h}\right|$ and a Rayleigh noise.



Spot h

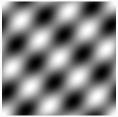


RPN Modulus

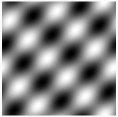


ADSN Modulus

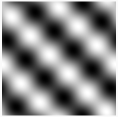
- ***RPN* and *ADSN* are two different processes.**



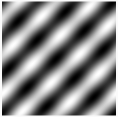
Spot h



RPN



An *ADSN*



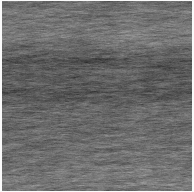
Another *ADSN*

RPN and *ADSN* as texture synthesis algorithms

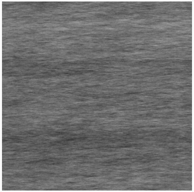
RPN and *ADSN* associated to texture images

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

Original image



RPN



ADSN

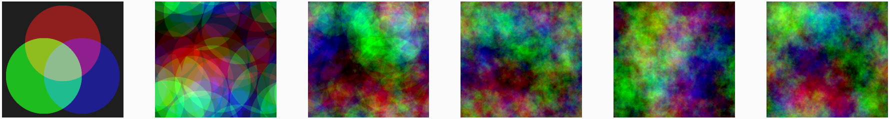
- ... But several developments are necessary to derive texture synthesis algorithms from sample.

Extension to color images

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

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

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



Spot h | $n = 10$ | $n = 10^2$ | $n = 10^3$ | $n = 10^4$ | color *ADSN*

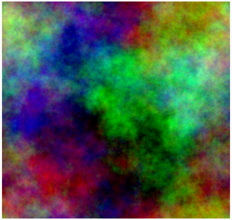
- Phase of color *ADSN*: The same random phase is added to the Fourier transform of each color channel.

Extension to color images

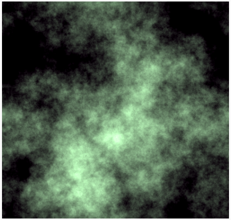
- **Color *RPN*:** By analogy, the *RPN* associated with a color image $h = (h_r, h_g, h_b)$ is the color image obtained by **adding the same random phase** to the Fourier transform of each color channel.

Original image h

Color *RPN*



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

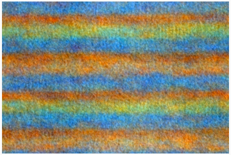


$$\hat{h} = \begin{pmatrix} |\hat{h}_R| e^{i\varphi_R} \\ |\hat{h}_G| e^{i\varphi_G} \\ |\hat{h}_B| e^{i\varphi_B} \end{pmatrix} \qquad \hat{Z} = \begin{pmatrix} |\hat{h}_R| e^{i(\varphi_R + \theta)} \\ |\hat{h}_G| e^{i(\varphi_G + \theta)} \\ |\hat{h}_B| e^{i(\varphi_B + \theta)} \end{pmatrix} \qquad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

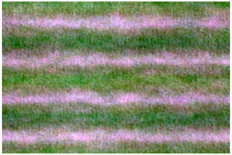
Extension to color images

- Another example with a real-world texture.

Original image h



Color *RPN*



"Wrong *RPN*"

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

Original image h

Color *RPN*

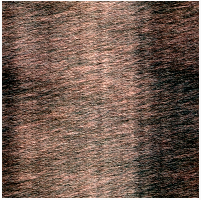
"Wrong *RPN*"

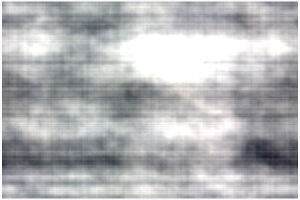
Avoiding artifacts due to non periodicity

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

Spot h

ADSN





Avoiding artifacts due to non periodicity

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

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

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

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

These two Laplacians only differ at the border:

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

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

FFT-based Poisson Solver

Periodic Poisson problem: Find the image p such that

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

In the Fourier domain, this system becomes:

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

Algorithm to compute the periodic component:

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

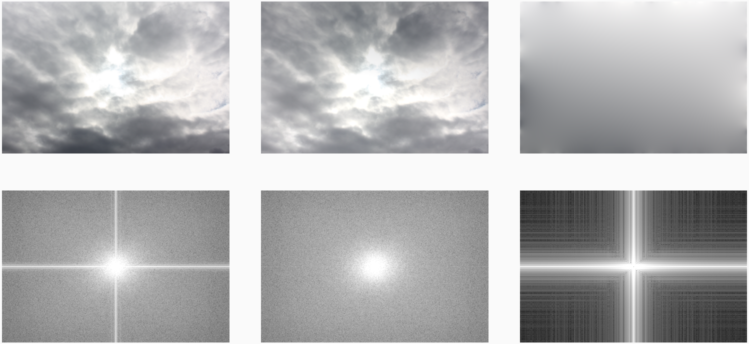
$$\begin{cases} \hat{p}(s,t) = \frac{\widehat{\Delta_i h}((s,t))}{-4+2\cos\left(\frac{2s\pi}{M}\right)+2\cos\left(\frac{2t\pi}{N}\right)} & \text{for } (s,t) \in \hat{\Omega} \setminus \{(0,0)\} \\ \hat{p}(0,0) = m \end{cases}$$

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

Periodic component: effects on the Fourier modulus

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

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

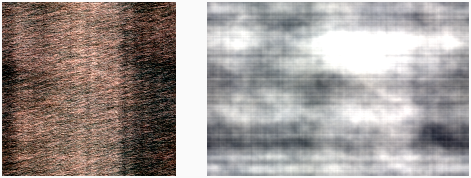


- The application $\mathrm{per} : h \mapsto p$ filters out the "cross structure" of the spectrum.

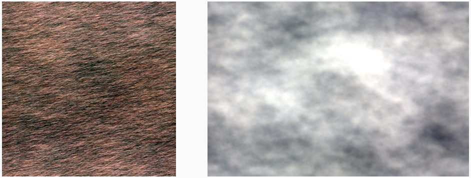
Avoiding artifacts due to non periodicity

Spot h

ADSN(h)



ADSN(p)



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

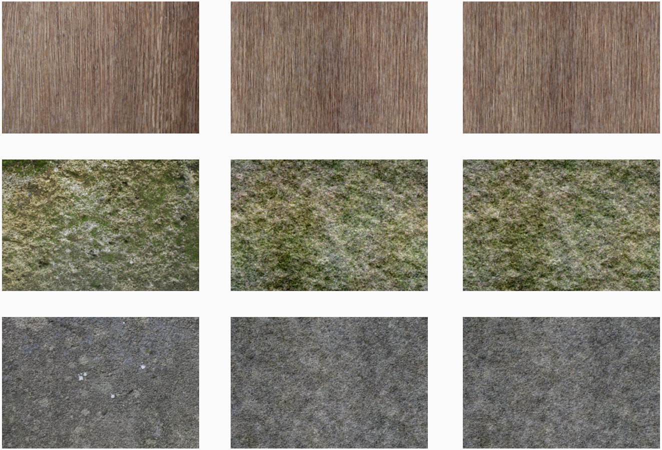
RPN 3

- [ON LINE DEMO]

Numerical results: similarity of the textures

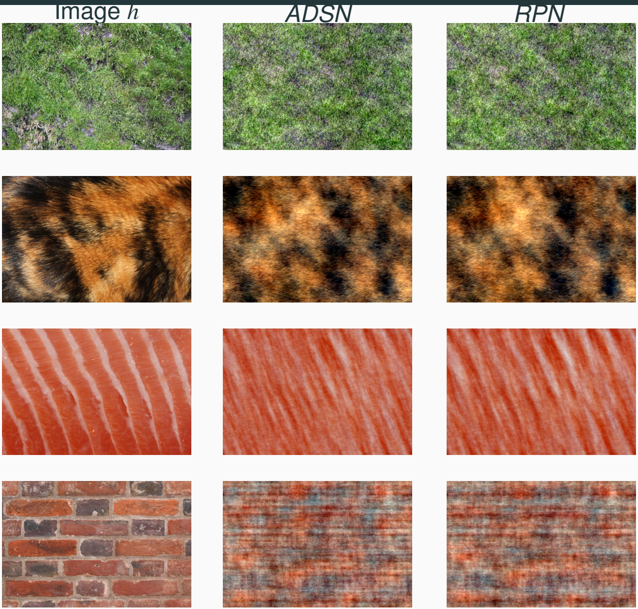
- In order to compare both algorithms, the same random phase is used for *ADSN* and *RPN*.

Image h | *ADSN* | *RPN*



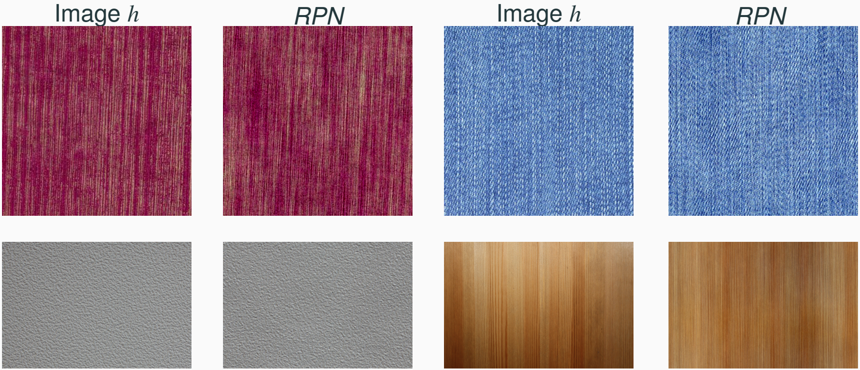
- Both algorithms produce visually similar textures.

Numerical results: non random phase textures



Some other examples of well-reproduced textures...

- We only display the *RPN* result.



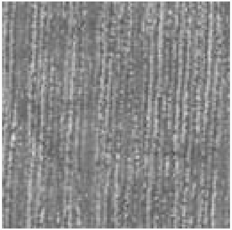
- Much more examples of success and failures on the IPOL webpage: `http://www.ipol.im/pub/algo/ggm_random_phase_texture_synthesis/`

Texton

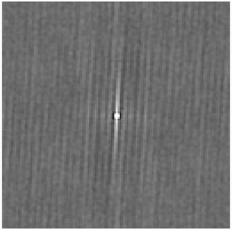
Texton associated with a texture

We work here with gray-level images.

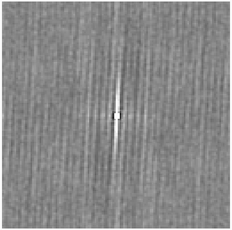
- RPN and ADSN models associated with h only depends of the Fourier modulus of h.
- Definition: The *texton* t_h associated with h is the image with the same modulus as h and with zero phase (Desolneux et al., 2015).



Input h



Texton t_h (log scale)



Texton t_h (thresholded)

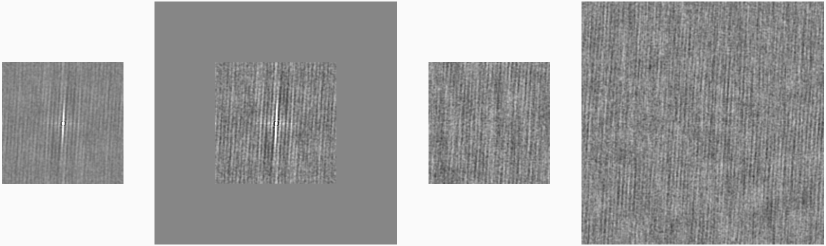
- Concentrated in zero: Compact representation of the texture model
- Interesting tool for analysis:

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*$(\tilde{t}_h)$

Interests and limitations

Interests: The CADSN reproduces most natural micro-textures. It is a fast and reliable algorithm.

Stationary Gaussian texture model: Well-defined mathematical model that as seen several developments:

- Definition of the canonical texton (Desolneux et al., 2012)
- Gaussian texture mixing using optimal transport barycenter (Xia et al., 2014)

Interests and limitations

Interests: The CADSN reproduces most natural micro-textures. It is a fast and reliable algorithm.

Stationary Gaussian texture model: Well-defined mathematical model that as seen several developments:

- Definition of the canonical texton (Desolneux et al., 2012)
- Gaussian texture mixing using optimal transport barycenter (Xia et al., 2014)
- **Microtexture inpainting through Gaussian conditional simulation** (Galerne et al., 2016) (Galerne and Leclaire, 2017)(Galerne and Leclaire, 2016)
- **Procedural noise by example** (Galerne et al., 2012, 2017)

Works related to RPN model:

- Similarity between RPN and ADSN models used in (Blanchet and Moisan, 2012; Leclaire and Moisan, 2015)
- Extension of the RPN model in a continuous setting (random field) (Ronsin et al., 2016)

Interests and limitations

Limitations of Gaussian model:

- Gaussian textures are limited: no geometric contours!
- The model is not robust to non stationarity, perspective effects, ...

Interests and limitations

Limitations of Gaussian model:

- Gaussian textures are limited: no geometric contours!
- The model is not robust to non stationarity, perspective effects, ...

Limitations due to FFT simulation:

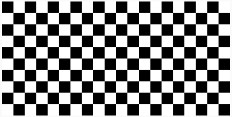
- The method is global: The whole texture image has to be computed.
- It produces periodic images with a fixed size which cannot be extended a posteriori.

Procedural noise

- A **procedural texture** is a program $x \mapsto f(x)$, where $f(x)$ is the gray-level of some texture at point $x \in \mathbb{R}^2$.
- **Continuous texture model** defined over the whole plane $\mathbb{R}^2$

Example:

A checkerboard is obtained by
$f(x_1, x_2) = (\mathrm{mod}(\lfloor x_1 \rfloor, 2) \neq \mathrm{mod}(\lfloor x_2 \rfloor, 2))$



Main interest:

- *Compact* representation (in terms of memory)
- On the fly parallel evaluation of the texture: ideal for GPU
- Easiest to map on surfaces than raster texture images (no interpolation issue)

Procedural noise

- To generate irregular patterns, **procedural noise** models have been developed: Perlin Noise (Perlin, 1985), Wavelet Noise (Cook and DeRose, 2005), Gabor noise (Lagae et al., 2009).

$$\text{Procedural noise: } x \mapsto n(x)$$

- They produce random but spatially coherent textures.
- One controls the texture appearance through their **power spectrum** (= frequency content).
- They are "easily" mapped onto (parameterized) surfaces.

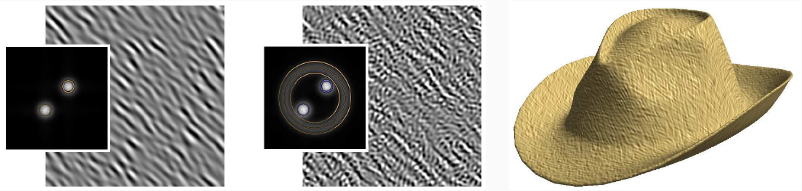


Illustration: ***Gabor noise*** (Lagae et al., 2009)

Noise by example

Procedural noise by example:

- Determine the parameters of a procedural noise that visually reproduces a given texture image.

Procedural noise and Gaussian random fields:

- A procedural noise based on the shot noise model converges in distribution towards a stationary Gaussian random field when the number of summed functions tends to infinity.
- All procedural noises are approximately Gaussian (Lagae et al., 2010).

Noise by example

Procedural noise by example:

- Determine the parameters of a procedural noise that visually reproduces a given texture image.

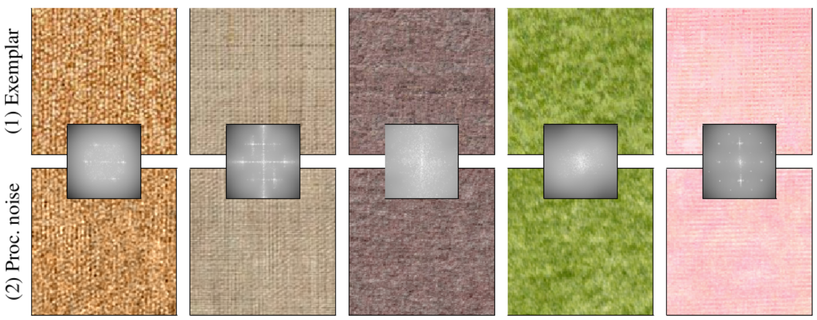
Procedural noise and Gaussian random fields:

- A procedural noise based on the shot noise model converges in distribution towards a stationary Gaussian random field when the number of summed functions tends to infinity.
- All procedural noises are approximately Gaussian (Lagae et al., 2010).

When restricting to Gaussian textures, the noise by example problem becomes well-posed:

Determine a procedural noise whose power spectrum
is close to the one of the input texture

- With **Gabor noise by example** (Galerne et al., 2012) we demonstrated that it was possible to reproduce any Gaussian texture with a procedural noise.



- However, the resulting algorithm was quite involved for both analysis and synthesis.
- Requires 1 sec. for generating a full HD 1920×1080 image.

Texton noise

Poisson point process

Poisson distribution with parameter $\lambda \in (0, +\infty)$: $X \sim \mathcal{P}(\lambda)$ if

$$\forall n \in \mathbb{N},\ \mathbb{P}(X = n) = \frac{\lambda^n e^{-\lambda}}{n!}.$$

Definition (Poisson point process)
Let Φ be a point process on $\mathbb{R}^d$ and let μ be its intensity measure. Φ is a *Poisson (point) process* on $\mathbb{R}^d$ if:

(i) For any disjoint Borel subsets $A_1, A_2, \dots, A_n \subset \mathbb{R}^d$, the random variables $\Phi(A_1), \Phi(A_2), \dots, \Phi(A_n)$ are mutually independent.

(ii) For all Borel subset $A \subset \mathbb{R}^d$, $\Phi(A)$ has the Poisson distribution with parameter $\mu(A) \in [0, +\infty]$, that is $\Phi(A) \sim \mathcal{P}(\mu(A))$.

Stationary Poisson process on R^2: Π_λ is the Poisson point process with intensity measure $\mu = \lambda \mathcal{L}^2$. The intensity λ is the mean number of point per unit area.

Sum over a Poisson point process: Campbell formula

Theorem (Campbell's Theorem)

Let Φ be a Poisson process on $\mathbb{R}^d$ with mean measure μ, and let $f : \mathbb{R}^d \to \mathbb{R}$ be a measurable function. Then the sum

$$\Sigma = \sum_{X \in \Phi} f(X)$$

is absolutely convergent with probability 1 if and only if

$$\int_{\mathbb{R}^d} \min\left(|f(x)|, 1\right) \mu(dx) < +\infty. \tag{1}$$

If this condition holds, then

$$\mathbb{E}\left(e^{\theta\Sigma}\right) = \exp\left(\int_{\mathbb{R}^d} \left(e^{\theta f(x)} - 1\right) \mu(dx)\right)$$

for any complex θ for which the integral on the right converges (e.g. θ is pure imaginary). Moreover

$$\mathbb{E}\left(\Sigma\right) = \int_{\mathbb{R}^d} f(x)\mu(dx) \quad \textit{and} \quad \mathrm{Var}(\Sigma) = \int_{\mathbb{R}^d} f(x)^2 \mu(dx) \tag{2}$$

Poisson shot noise

Model: Single kernel shot noise on $\mathbb{R}^2$

$$f_\lambda(x) = \sum_{x_j \in \Pi_\lambda} h(x - x_j)$$

- $\Pi_\lambda \subset \mathbb{R}^2$ is a Poisson point process with intensity $\lambda > 0$
- $h : \mathbb{R}^2 \to \mathbb{R}$ is called the kernel.

By Campbell formula,

$$\mathbb{E}(f_\lambda(x)) = \lambda \int_{\mathbb{R}^2} h(y)dy \quad \text{and} \quad \mathrm{Cov}(f_\lambda(x+\tau), f_\lambda(x)) = \lambda \int_{\mathbb{R}^2} h(y+\tau)h(y)dy.$$

Theorem (Normal convergence of high density shot noise)

Suppose that $\int_{\mathbb{R}^2} |h(y)|^k \, dy < +\infty$ *for* $k = 1$ *and* $k = 2$*. Then, as* λ *tends to* $+\infty$*, the family of normalized shot noise* $g_\lambda(x) = \frac{f_\lambda(x) - \mathbb{E}(f_\lambda)}{\sqrt{\lambda}}$ *converges in the sense of finite dimensional distributions to a stationary Gaussian random field having null expectation and covariance function*

$$C(\tau) = \int_{\mathbb{R}^2} h(y+\tau)h(y)dy, \quad \tau \in \mathbb{R}^d.$$

Texton noise

- **Motivation:** Propose the simplest (and fastest) noise model that enables to reproduce any Gaussian texture.

Texton noise

- **Motivation:** Propose the simplest (and fastest) noise model that enables to reproduce any Gaussian texture.
- **Model:** Single kernel shot noise on $\mathbb{R}^2$

$$f_\lambda(x) = \sum_{x_j \in \Pi_\lambda} h(x - x_j)$$

  - $\Pi_\lambda \subset \mathbb{R}^2$ is a Poisson point process with intensity $\lambda > 0$
  - the kernel $h : \mathbb{R}^2 \to \mathbb{R}^d$ ($d = 1$ or 3) is called ***texton***

Texton noise

- **Motivation:** Propose the simplest (and fastest) noise model that enables to reproduce any Gaussian texture.
- **Model:** Single kernel shot noise on $\mathbb{R}^2$

$$f_\lambda(x) = \sum_{x_j \in \Pi_\lambda} h(x - x_j)$$

  - $\Pi_\lambda \subset \mathbb{R}^2$ is a Poisson point process with intensity $\lambda > 0$
  - the kernel $h : \mathbb{R}^2 \to \mathbb{R}^d$ ($d = 1$ or 3) is called ***texton***
- The texton h is a **bilinearly interpolated image**:

$$h(x) = \sum_{k \in \mathbb{Z}^2} \alpha(k)\psi(x - k), \quad x \in \mathbb{R}^2,$$

  where ψ is the bilinear interpolation kernel

Texton noise

- **Motivation:** Propose the simplest (and fastest) noise model that enables to reproduce any Gaussian texture.
- **Model:** Single kernel shot noise on $\mathbb{R}^2$

$$f_\lambda(x) = \sum_{x_j \in \Pi_\lambda} h(x - x_j)$$

  - $\Pi_\lambda \subset \mathbb{R}^2$ is a Poisson point process with intensity $\lambda > 0$
  - the kernel $h : \mathbb{R}^2 \to \mathbb{R}^d$ ($d = 1$ or 3) is called ***texton***
- The texton h is a **bilinearly interpolated image**:

$$h(x) = \sum_{k \in \mathbb{Z}^2} \alpha(k)\psi(x - k), \quad x \in \mathbb{R}^2,$$

  where ψ is the bilinear interpolation kernel

Noise evaluation:

- Fast evaluation of the bilinear interpolation $x \mapsto h(x)$ on GPU (*texture fetch*)
- On-the-fly parallel simulation of the Poisson proccess:
  Based on a grid partition where each cell has its own pseudo-random number generator (Lagae et al., 2009)

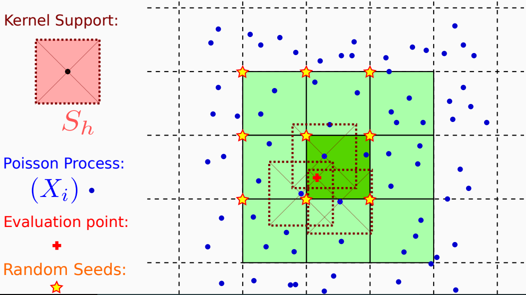
Parallel Evaluation

$$f_\lambda(x) = \sum_{x_j \in \Pi_\lambda} h(x - x_j) = h * P_\lambda(x)$$

- On a finite domain, the simulation can be done by direct summation (N_{imp} operations per pixels) or FFT convolution (on a larger domain).

Parallel evaluation using a grid-based local Poisson simulation

- The Poisson point process Π_λ is simulated locally in a reproducible way using pseudo-random number generators seeded using cell coordinates (Lagae et al., 2009).



Texton noise

- Given an input image u, find the interpolation coefficient α such that the normalized shot noise $\frac{f_\lambda - \lambda \int_{\mathbb{R}^2} h(x)dx}{\sqrt{\lambda}}$ reproduces u (or at least its Gaussian version)

Texton noise

- Given an input image u, find the interpolation coefficient α such that the normalized shot noise $\frac{f_\lambda - \lambda \int_{\mathbb{R}^2} h(x)dx}{\sqrt{\lambda}}$ reproduces u (or at least its Gaussian version)

Target noise power spectrum:

- Sampling consistency: The sampling of the noise over the grid $\mathbb{Z}^2$ must have the same covariance as the ADSN model associated with u:

$$|\hat{\alpha}(\xi)|^2 \, \hat{b}(\xi) = \left|\hat{h}_u(\xi)\right|^2, \quad \xi \in [-\tfrac{1}{2}, \tfrac{1}{2}]^2$$

where b is the $\mathbb{Z}^2$-sampling of the cubic spline kernel $\psi * \psi$.
- This equation does not have solution α with compact support.

Texton noise

- Given an input image u, find the interpolation coefficient α such that the normalized shot noise $\frac{f_\lambda - \lambda \int_{\mathbb{R}^2} h(x)dx}{\sqrt{\lambda}}$ reproduces u (or at least its Gaussian version)

Target noise power spectrum:

- Sampling consistency: The sampling of the noise over the grid $\mathbb{Z}^2$ must have the same covariance as the ADSN model associated with u:

$$|\hat{\alpha}(\xi)|^2 \, \hat{b}(\xi) = \left|\hat{h}_u(\xi)\right|^2, \quad \xi \in [-\tfrac{1}{2}, \tfrac{1}{2}]^2$$

where b is the $\mathbb{Z}^2$-sampling of the cubic spline kernel $\psi * \psi$.
- This equation does not have solution α with compact support.

Computing texton noise coefficients:

- Alternate projection algorithm to compute coefficients α s.t. (Galerne et al., 2014)
  1. α has support in S.
  2. $|\hat{\alpha}(\xi)|^2 \, \hat{b}(\xi) \approx \left|\hat{h}_u(\xi)\right|^2$ for all $\xi \in [-\frac{1}{2}, \frac{1}{2}]^2$.

Results for texton noise

- "Visual Gaussian convergence" with a **mean number of impact of 30**.

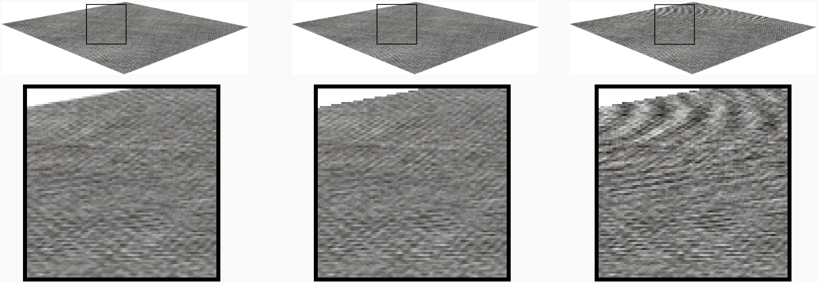
Results for texton noise

Performance: OpenGL implementation runs at 100 fps for full HD (1920×1080) on a Nvidia Quadro K5000 (1536 Cuda cores).

Antialisaing filtering:

- Antialiasing filtering is mandatory when applying noise on a surface.
- Simply calling the standard filtering procedures for the bilinear texton (stored as a GPU texture), texton noise enables fast and accurate filtering.

Ideal hypersampled noise | Filtered texton noise | Unfiltered texton noise



Surface noise

- Texton noise allows for *surface noise* as proposed in (Lagae et al., 2009) to apply the noise on the surface without a parameterization.

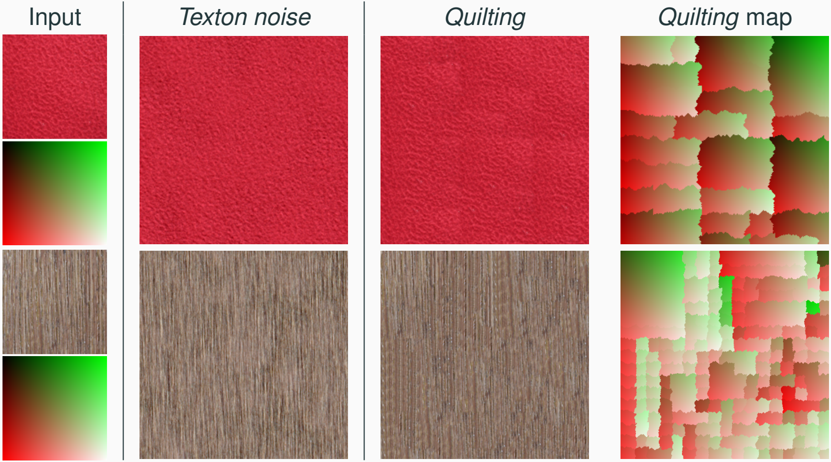
Spatially varying texture mixing

- Based on (Xia et al., 2014) (Wasserstein barycenter between Gaussian distributions), we propose a *spatially varying texture mixing* thanks to the local support of the texton.

Comparison

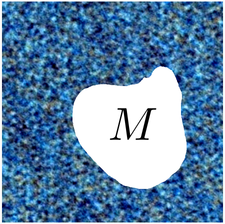
Is synthesizing Gaussian textures useful ?

- Gaussian micro-textures are not “easy” for patch-based methods !
- Comparison with *image quilting* (Efros and Freeman, 2001) (Raad and Galerne, 2017)

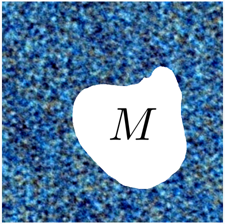


Microtexture inpainting

- Inpainting consists in **filling missing regions of an image**.
- In the case of random texture models, inpainting can be formulated as **conditional simulation**
- **Notation:**
  - $\Omega \subset \mathbb{Z}^2$: image domain
  - $M \subset \Omega$: mask
  - u: input texture known only on $\Omega \setminus M$

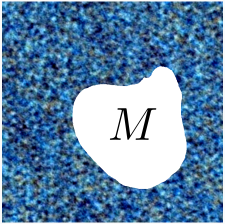


- Inpainting consists in **filling missing regions of an image**.
- In the case of random texture models, inpainting can be formulated as **conditional simulation**
- **Notation:**
  - $\Omega \subset \mathbb{Z}^2$: image domain
  - $M \subset \Omega$: mask
  - u: input texture known only on $\Omega \setminus M$



Inpainting of a Gaussian texture:

- Inpainting consists in **filling missing regions of an image**.
- In the case of random texture models, inpainting can be formulated as **conditional simulation**
- **Notation:**
  - $\Omega \subset \mathbb{Z}^2$: image domain
  - $M \subset \Omega$: mask
  - u: input texture known only on $\Omega \setminus M$



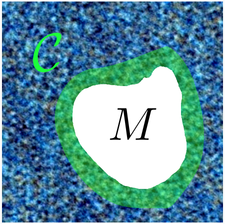
Inpainting of a Gaussian texture:

1. Estimation of an ADSN model U from the masked input u.

$$U = \mathrm{moy}(u) + h_u * X \quad \text{where} \quad h_u = \frac{1}{\sqrt{|\Omega \setminus M|}}(u - \mathrm{moy}(u))$$

Microtexture inpainting

- Inpainting consists in **filling missing regions of an image**.
- In the case of random texture models, inpainting can be formulated as **conditional simulation**
- **Notation:**
  - $\Omega \subset \mathbb{Z}^2$: image domain
  - $M \subset \Omega$: mask
  - u: input texture known only on $\Omega \setminus M$
  - $\mathcal{C}$ a set of conditioning points



Inpainting of a Gaussian texture:

1. Estimation of an ADSN model U from the masked input u.

$$U = \text{moy}(u) + h_u * X \quad \text{where} \quad h_u = \frac{1}{\sqrt{|\Omega \setminus M|}}(u - \text{moy}(u))$$

2. Conditional simulation of U knowing that $U_{|\mathcal{C}} = u_{|\mathcal{C}}$ (using kriging...)

Gaussian conditional sampling using kriging estimation

- Let $(F(x))_{x\in\Omega}$ be a Gaussian vector **with mean zero** and covariance

$$\Gamma(x,y) = \text{Cov}(F(x),F(y)) = \mathbb{E}(F(x)F(y)), \quad x,y\in\Omega.$$

- The (simple) **kriging estimation** is defined by

$$F^*(x) = \mathbb{E}(\, F(x) \mid F(c),\ c\in\mathcal{C}).$$

- There exists $(\lambda_c(x))_{c\in\mathcal{C}}$ such that $F^*(x) = \sum_{c\in\mathcal{C}} \lambda_c(x) F(c)$.

Theorem: F^* and $F - F^*$ are independent. (see e.g. (Lantuéjoul, 2002))

Consequence: A conditional sample of F given $F_{|\mathcal{C}} = \varphi$ can be obtained as

$$F \mid F_{|\mathcal{C}} = \varphi \quad \sim \quad \underbrace{\varphi^*}_{\textbf{Kriging component}} \quad + \quad \underbrace{F - F^*}_{\textbf{Innovation component}} \quad .$$

- The **kriging coefficients** $\Lambda = (\lambda_c(x))_{\substack{x\in\Omega\\ c\in\mathcal{C}}}$ satisfy $\Gamma_{|\Omega\times\mathcal{C}} = \Lambda\Gamma_{|\mathcal{C}\times\mathcal{C}}$.

- We use the pseudo-inverse of $\Gamma_{|\mathcal{C}\times\mathcal{C}}$: $\boxed{\Lambda = \Gamma_{|\Omega\times\mathcal{C}}\Gamma^{\dagger}_{|\mathcal{C}\times\mathcal{C}}}$

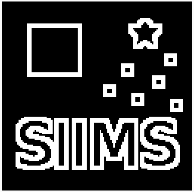
Inpainting of a Gaussian texture

1. Estimation of an ADSN model U from masked input u.
2. Conditional simulation of U knowing that $U_{|\mathcal{C}} = u_{|\mathcal{C}}$:

$$\text{Compute} \quad v = \text{mean}(u) + \underbrace{(u - \text{mean}(u))^{\star}}_{\text{Kriging component}} + \underbrace{U - U^{\star}}_{\text{Innovation component}}$$

Original

Masked input



Conditioning points $\mathcal{C}$

Kriging component

Innovation component

Inpainted texture

Efficient algorithm

- First version presented at ICASSP used explicit matrices to compute

$$\varphi^* = \Gamma_{|\Omega\times\mathcal{C}}\Gamma^{\dagger}_{|\mathcal{C}\times\mathcal{C}}\varphi.$$

- Suitable only for (very) small images !

Efficient algorithm

- First version presented at ICASSP used explicit matrices to compute

$$\varphi^* = \Gamma_{|\Omega\times\mathcal{C}}\Gamma^{\dagger}_{|\mathcal{C}\times\mathcal{C}}\varphi.$$

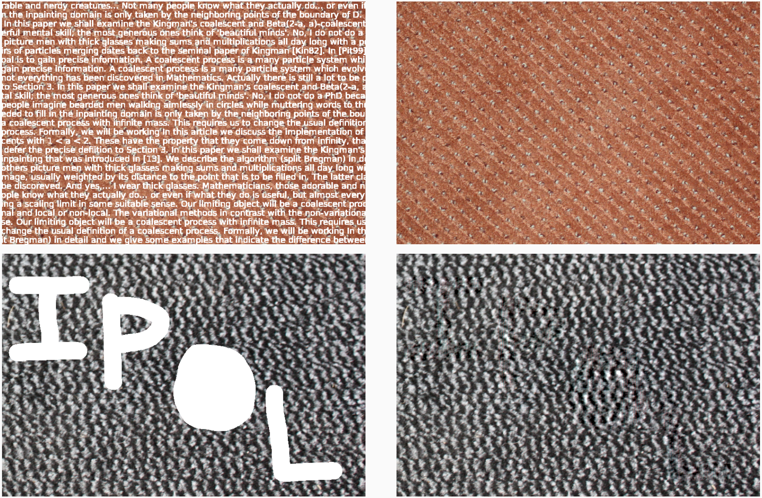
- Suitable only for (very) small images !

Scalable Implementation:

- The covariance Γ is the autocorrelation of $h_u = \frac{1}{\sqrt{|\Omega\setminus M|}}(u - \mathrm{moy}(u))$.
- All matrix-vector multiplication with restrictions of Γ can be done using FFT-based convolution.
- Computing $\Gamma^{\dagger}_{|\mathcal{C}\times\mathcal{C}}\varphi$ done using conjugate gradient descent (CGD).
- Each CGD iteration has the cost of a couple of convolutions (and does not depend on the number of points to fill !)
- In practice, 1000 iterations gives a good approximate solution.
- On-line demo with only 100 iterations (Galerne and Leclaire, 2016).
- It turns out that using a 3 pixel wide boundary for $\mathcal{C}$ is visually good enough, and better for the conditioning of the linear system.

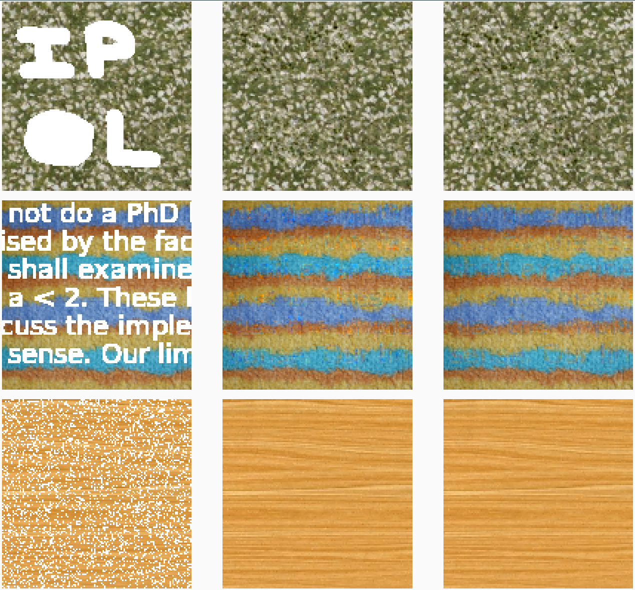
Results: Large problems

Masked texture | Inpainted texture



- Results are satisfying as soon as the Gaussian model is well estimated.

Results: Failures



Input | 100 CGD it. | 1000 CGD it.

Comparison with path-based methods

- Unfair comparison: Other algorithms are **not limited to textures !**

Original

Gaussian inpainting

Kriging component

(Arias et al., 2011)

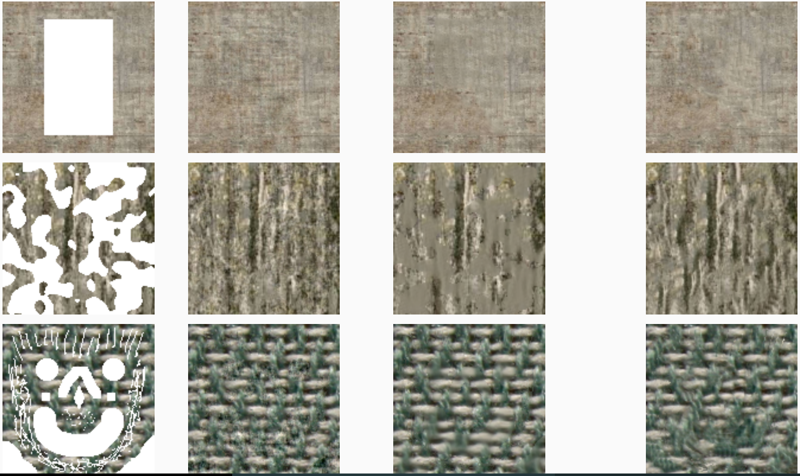
(Buyssens et al., 2015)

(Newson et al., 2014)

- Thanks to the covariance estimation, the Gaussian inpainting is consistent regarding long range correlations.

Comparison with path-based methods

- Our algorithm often gives better results when inpainting a stationary texture, even if the texture is not Gaussian.
- Inpainting textures is not an easy task.



Stochastic superresolution: (Lugmayr et al., 2020) “SRFlow: Learning the Super-Resolution Space with Normalizing Flow”

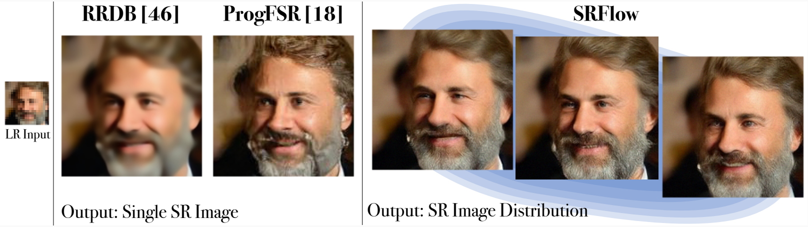


Fig. 1. While prior work trains a deterministic mapping, SRFlow learns the distribution of photo-realistic HR images for a given LR image. This allows us to explicitly account for the ill-posed nature of the SR problem, and to sample diverse images. ($8\times$ upscaling)

Superresolution of Gaussian textures: (work in progress with Emile Pierret)

- Study a base case of stochastic super-resolution.

Idea:

- $X \sim \mathcal{N}(0, \Gamma) \in \mathbb{R}^{n \times n}$ a Gaussian stationary process : $X = t * W$ with $W \sim \mathcal{N}(0, I_d)$ (ADSN model).
- $Y = AX \in \mathbb{R}^{sn \times sn}$, ZOOM-out of X with $s = 1/2, 1/4, \ldots, r = 1/s = 2, 4, \ldots$
- Sample $X | AX = Y$

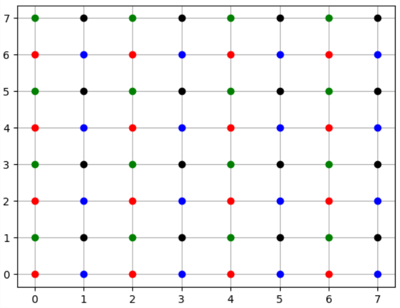
X is Gaussian. Consequently, $\mathbb{E}(X|AX)$ is Gaussian and there exits $\Lambda \in \mathbb{R}^{n \times sn}$ such that $\mathbb{E}(X|AX) = \Lambda^T AX$.

Proposition
Let $\Lambda \in \mathbb{R}^{s\Omega \times \Omega}$ such that $\mathbb{E}(X|AX) = \Lambda^T AX$, Λ verifies the equation :

$$A\Gamma A^T \Lambda = A\Gamma$$

Proposition (Reduction of the number of the systems to solve)
Only r^2 columns of Λ are necessary to express $\mathbb{E}(X|AX)$. More precisely, Λ^T is a convolution on the lattices generated by (k,ℓ) for $k,\ell \in \{0, r-1\}$ and for $i,k,j,\ell \in \mathbb{N}$ such that $x = (i+kr, j+\ell r) \in \Omega$ by $\check{\lambda}(i,j)$ and :

$$A\Gamma A^T \left((J_{sn}^T)^\ell \otimes (J_{sn}^T)^k \right) \lambda(i,j) = (A\Gamma A^T)\lambda(x,y) = A\Gamma_{\Omega\times\{(x,y)\}}.$$



$s = 1/2$

- Λ is a convolution on each lattices generated by (k,ℓ) for $k,\ell \in [\![0, r-1]\!]$
- $\Lambda \in \mathbb{R}^{(n)^2\times(sn)^2}$
- Λ applies a convolution with a $(sn \times sn)$ image on each lattice of size $(sn \times sn)$ of $\tilde{X}$.
- Needs to store $r(sn)^2 = sn^2$ values. $(= n^2/2, n^2/4, \dots)$

Superresolution of Gaussian textures: Results for $s = 1/4$

Input

Conditional HR

Superresolution of Gaussian textures: Results for $s = 1/4$

Input

Conditional HR

Input

Conditional HR

Limitations:

- Limited to stationary textures.
- The added HR grain is independent of the kriging component.

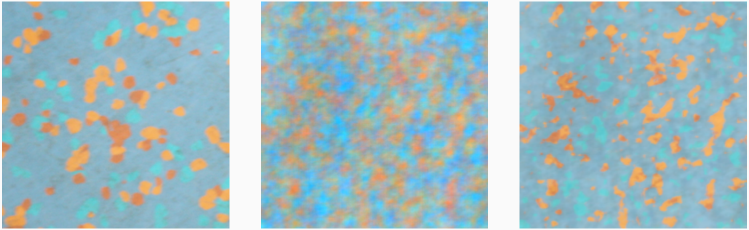
Semi-discrete Optimal Transport

Goal: Exemplar-based synthesis of structured textures.

Design a model that

- has statistical guarantees,
- allows for fast and parallel synthesis.

Main idea: Extend the Gaussian model with an adapted local transformation.



Exemplar — Gaussian field — Locally transformed Gaussian field

Reference: (Galerne et al., 2018)

Related works

Optimal transport for imaging applications

- Image matching [Rabin et al., 2009]
- Color transfer [Rabin et al., 2011], [Bonneel et al., 2015]
- Image segmentation [Papadakis et al., 2015]
- Shape interpolation [Solomon et al., 2015]
- Texture synthesis and mixing
  [Xia et al., 2014] [Tartavel et al., 2016] [Gutierrez et al., 2017]

Semi-discrete optimal transport

- Least-squares assignment [Aurenhammer, Hoffmann, Aronov, 1998]
- Numerical solution based on multiscale L-BFGS
  [Mérigot, 2011], [Lévy, 2015]
- Iterative scheme to get an ε-approximate solution [Kitagawa, 2014]
- Stochastic gradient descent [Genevay, Cuturi, Peyré, Bach, 2016]
- Damped Newton algorithm
  [Kitagawa, Mérigot, Thibert, 2017] [Mérigot, Meyron, Thibert, 2017]

Semi-discrete Optimal Transport

Let us consider two probability measures on $X, Y \subset \mathbb{R}^D$

- $\mu(dx) = \rho(x)dx$ **absolutely continuous** measure on X with pdf ρ
- $\nu = \sum_{j=1}^{J} \nu_j \delta_{y_j}$ **discrete** probability measure on $Y = \{y_j,\ 1 \leqslant j \leqslant J\}$

We consider the following **semi-discrete optimal transport** problem

$$\inf \int_X \|x - T(x)\|^2 d\mu(x) \qquad \text{(OT)}$$

where inf is taken over all measurable maps $T : X \to Y$ such that $\nu = T_\sharp \mu$.

Recall the definition of the push-forward measure

$$\forall A \in \mathcal{B}(\mathbb{R}^D), \quad T_\sharp \mu(A) = \mu(T^{-1}(A)).$$

Power cells

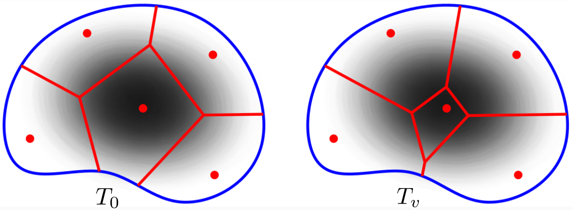
To solve this problem, for $v \in \mathbb{R}^J$ we consider the mapping

$$T_v(x) = \underset{y_j}{\text{Argmin}} \, \|x - y_j\|^2 - v_j$$

NB: When $v = 0 \to$ true nearest-neighbor (NN).

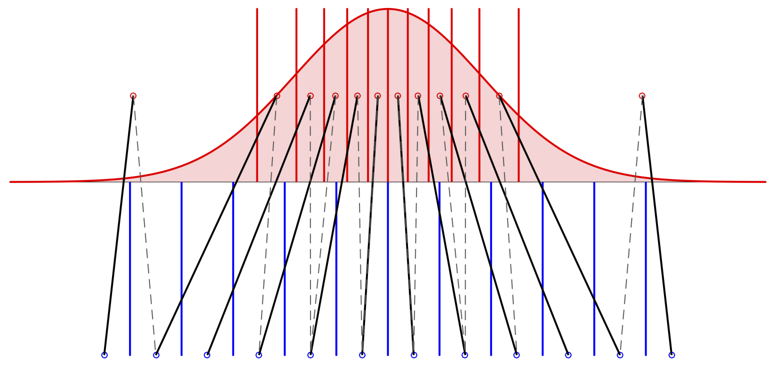
This mapping corresponds to a “power diagram”

$$\text{Pow}_v(y_j) = \{\, x \in \mathbb{R}^D \mid \forall k \neq j, \; \|x - y_j\|^2 - v_j < \|x - y_k\|^2 - v_k \,\}.$$



[Credits Kitagawa et al. 2017]

An example in 1D



Power diagram : from Gaussian to discrete uniform.

Blue: True Voronoi cells (NN assignment T_0)

Red: Power cells (optimal assignment T_v)

Dual Problem

The following theorem is due to [Aurenhammer, Hoffmann, Aronov, 1998].

See also [Kitagawa, Mérigot, Thibert, 2017].

Theorem
A solution to (OT) is given by T_v where v maximizes the $\mathcal{C}^1$ concave function

$$H(v) = \int_{\mathbb{R}^D} \left(\min_j \|x - y_j\|^2 - v_j\right) d\mu(x) + \sum_j \nu_j v_j,$$

whose gradient is given by $\dfrac{\partial H}{\partial v_j} = -\mu(\mathrm{Pow}_v(y_j)) + \nu_j$.

NB: H is not strictly concave.

Corollary
The following statements are equivalent

- v *is a global maximizer of* H
- T_v *is an optimal transport map between* μ *and* ν
- $(T_v)_\sharp \mu = \nu$

Stochastic Optimization

Writing $H(v) = \mathbb{E}_{X\sim\mu}[h(X,v)]$ where

$$h(x,v) = \left(\min_j \|x - y_j\|^2 - v_j\right) + \sum_j \nu_j v_j \ ,$$

$$\frac{\partial h}{\partial v_j}(x,v) = -\mathbf{1}_{\mathrm{Pow}_v(y_j)}(x) + \nu_j \ .$$

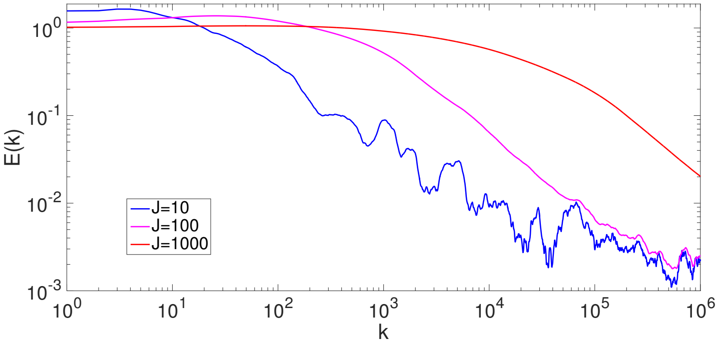
We maximize with average stochastic gradient ascent [Genevay et al., 2016]:

$$\begin{cases} \tilde{v}^k & = \tilde{v}^{k-1} + \frac{C}{\sqrt{k}} \nabla_v h(x^k, \tilde{v}^{k-1}) \quad \text{where } x^k \sim \mu \\ v^k & = \frac{1}{k}(\tilde{v}^1 + \ldots + \tilde{v}^k). \end{cases}$$

Theorem (Convergence guarantee)

$$\max H - \mathbb{E}[H(v^k)] = \mathcal{O}\left(\frac{\log k}{\sqrt{k}}\right).$$

ASGD for Semi-discrete OT
Convergence in dimension 1

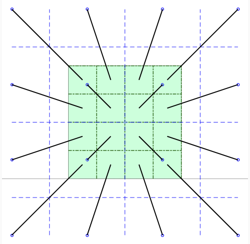


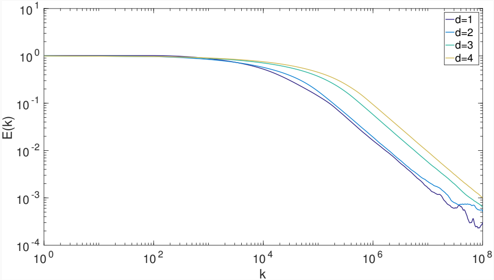
Transport in 1D from Gaussian to discrete uniform on J points

Evolution of $E(k) = \frac{\|v^k - v^\star\|}{\|v^\star\|}$

where $v^\star$ is the closed-form solution

ASGD for Semi-discrete OT
Convergence in dimension > 1





$$E(k) = \frac{\|v^k - v^\star\|}{\|v^\star\|}$$

where $v^\star$ is the closed-form solution

Optimal Transport in Patch Space

Many successes with patch-based texture synthesis

- [Efros, Leung, 1999], [Wei, Levoy, 2000]
- [Efros, Freeman, 2001]
- [Kwatra et al., 2003]
- [Lefebvre, Hoppe, 2005]
- [Raad et al., 2016]
- [Li, Wand, 2016]
- and many others...

Patches 11×11

Here we will use **optimal transport in patch space**, inspired by

- Texture classification by analysis of the patch distribution [Varma, Zissermann, 2003]
- Texture optimization for synthesis [Kwatra et al., 2005]
- Parallel controllable texture synthesis [Lefebvre, Hoppe, 2005]

Transformed Gaussian Field

- We start from the **Gaussian model**

$$U = \bar{u} + t_u * W \quad \text{where} \begin{cases} \bar{u} = \frac{1}{|\Omega|} \sum u(x), \\ t_u = \frac{1}{\sqrt{|\Omega|}} (u - \bar{u}) \mathbf{1}_\Omega \end{cases}$$

  and where W is a normalized Gaussian white noise on $\mathbb{Z}^2$.

- We then apply a **local transform**

$$\forall x \in \mathbb{Z}^2, \quad P_x = T(U_{|x+\omega}),$$
$$\forall x \in \mathbb{Z}^2, \quad V(x) = \frac{1}{|\omega|} \sum_{z \in \omega} P_{x-z}(z).$$

  where $\omega = \{0, \dots, w-1\}^2$ be the patch domain and where

$$T : \mathbb{R}^D \longrightarrow \mathbb{R}^D \quad \left(D = dw^2\right).$$

Optimal Transport in Patch Space

We choose the local transform T that realizes an optimal transport between

- μ distribution of the Gaussian patch $U_{|\omega}$
- $\nu = \sum_{j=1}^{J} \nu_j \delta_{p_j}$ where $\nu_j = \frac{1}{J}$ and $p_1, \dots, p_J$ are $J = 1000$ patches of u.

i.e. which solves the following **semi-discrete optimal transport** problem

$$\min \int_{\mathbb{R}^D} \|p - T(p)\|^2 d\mu(p) \qquad \text{(OT)}$$

We compute an optimal assignment

$$T_v(p) = \underset{p_j}{\text{Argmin}} \|p - p_j\|^2 - v_j$$

by running 10^6 iterations of stochastic gradient descent.

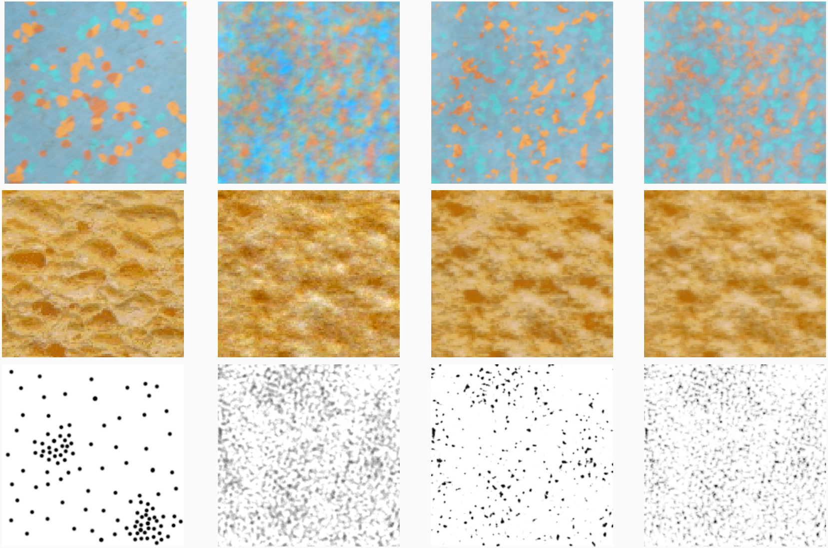
Guarantees

- V is a stationary random field on $\mathbb{Z}^2$
- Medium-range correlations are imposed in the Gaussian model.
- The patch distribution is reimposed with the local transform T.

Proposition (Long-range independence)
V satisfies the following property: for all $A, B \subset \mathbb{Z}^2$

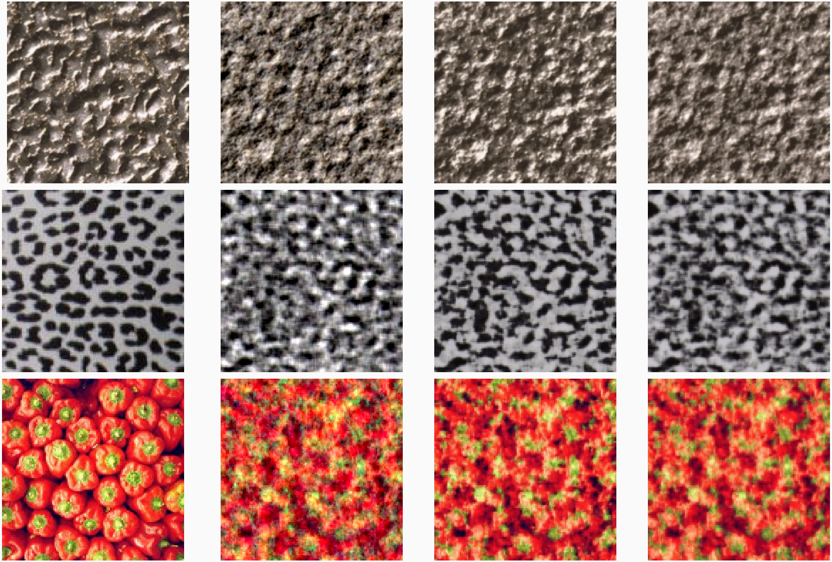
$$(A - B) \cap (\text{Supp}(t_u * \widetilde{t_u}) + 4\omega) = \varnothing \implies V_{|A}, V_{|B} \textit{ are independent.}$$

Monoscale Synthesis with 3×3 patches

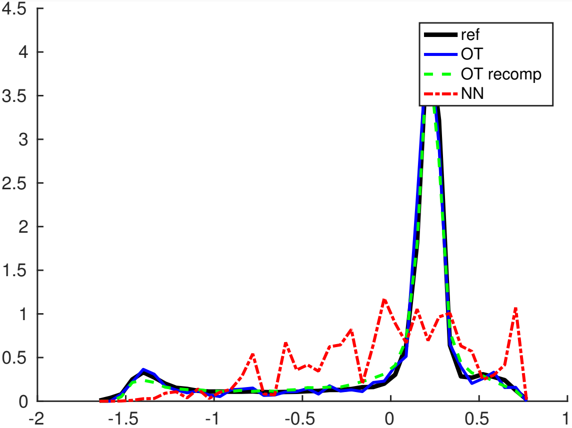


Original | Gaussian | Local transform (OT) | Local transform (NN)

Monoscale Synthesis with 3×3 patches


Original | Gaussian | Local transform (OT) | Local transform (NN)

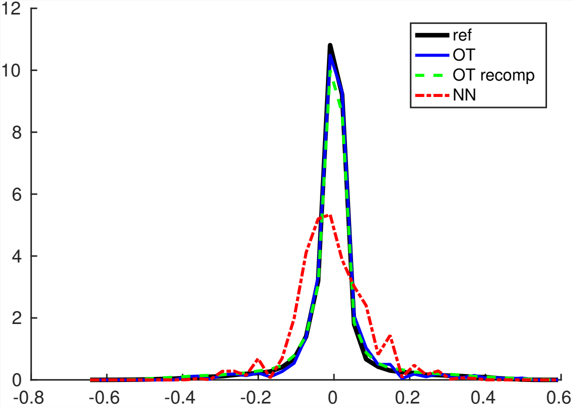
Output Patch Distribution
on the first Principal Components



3×3 patches

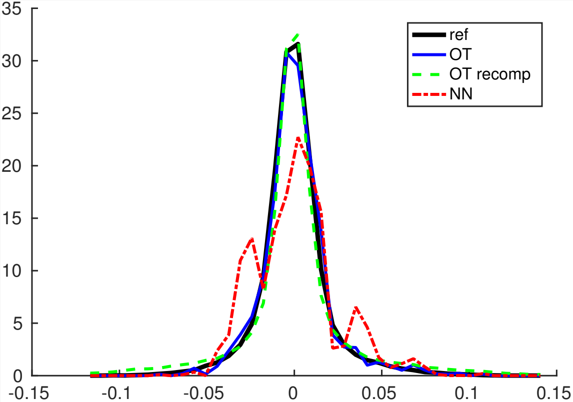
PC1

Output Patch Distribution on the first Principal Components



3×3 patches
PC3

Output Patch Distribution on the first Principal Components



3×3 patches
PC8

Multiscale input and target patch distributions

We compute the exemplar u^ℓ and target patch distribution ν^ℓ at different scales $\ell = 0, \dots, L$.

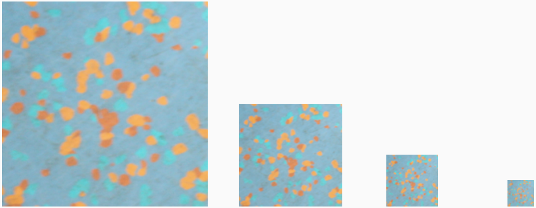


Image u^0 Target ν^0 — Image u^1 Target ν^1 — Image u^2 Target ν^2 — Image u^3 Target ν^3

And we compute one local transform at each scale, and perform synthesis recursively from coarse scale to fine scale.

From one scale to another

We initialize with a synthesis U^L with the Gaussian model.

Suppose we have a current synthesis U^ℓ at scale ℓ.

- Fit a Gaussian mixture model μ^ℓ to the empirical patch distribution of U^ℓ
- Compute optimal transport map T^ℓ from μ^ℓ to ν^ℓ
- Apply T^ℓ to each patch and recompose

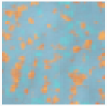
$$\forall x \in 2^\ell \mathbb{Z}^2, \quad V^\ell(x) = \frac{1}{|\omega|} \sum_{h \in 2^\ell \omega} T^\ell(U^\ell_{|x-h+2^\ell\omega})(h)$$

$$\text{i.e.} \quad \forall x \in 2^\ell \mathbb{Z}^2, \quad V^\ell(x) = \frac{1}{|\omega|} \sum_{h \in 2^\ell \omega} u^\ell\big(Y^\ell(x-h)+h\big)$$

- Upsample using the same patches at the coarser scale

$$\forall x \in 2^\ell \mathbb{Z}^2,\ \forall s \in \{0, 2^{\ell-1}\}^2, \quad U^{\ell-1}(x+s) = \frac{1}{|\omega|} \sum_{h \in 2^\ell \omega} u^{\ell-1}\big(Y^\ell(x-h)+s\big).$$

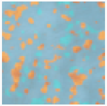
From one scale to another (illustration)



U^ℓ

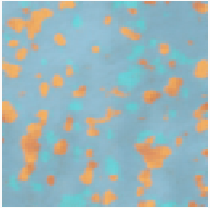
Estimate GMM μ^ℓ

$\xrightarrow[\text{Patch transform} T^\ell]{}$



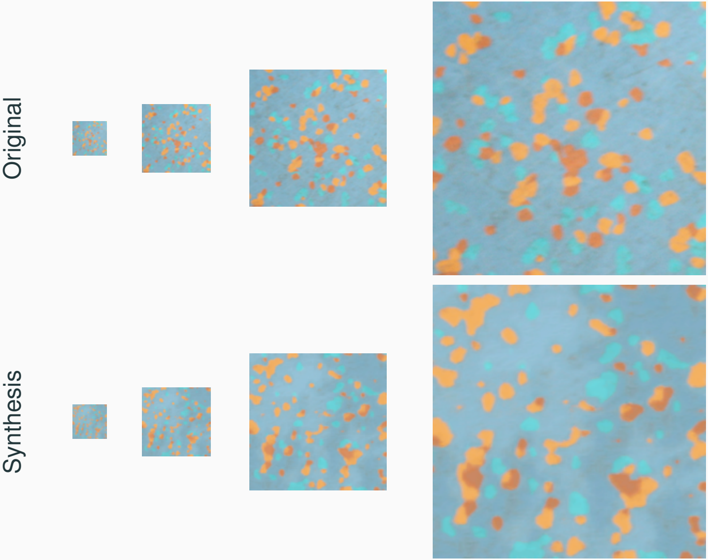
V^ℓ

$\xrightarrow[\text{Upsample}]{}$



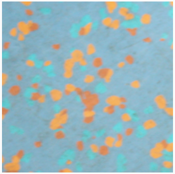
$U^{\ell-1}$

Multiscale synthesis



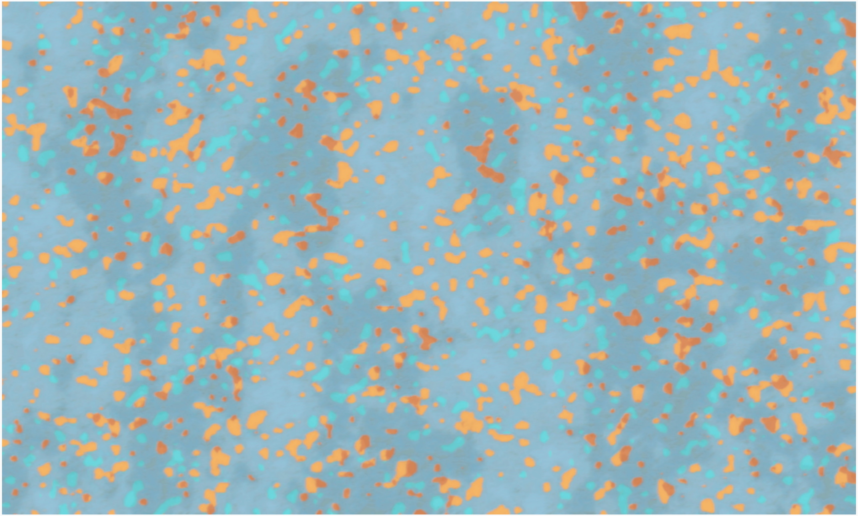
Remarks on Multiscale Model

- Long-range independence persists.
- At each scale, patches are transformed independtly
  $\rightarrow$ allows for parallel computations
- The parameters of the local transforms can be computed offline.
  $\rightarrow$ allows for very fast synthesis!
- The memory footprint is reasonably low



Original 256×256

Multiscale Results

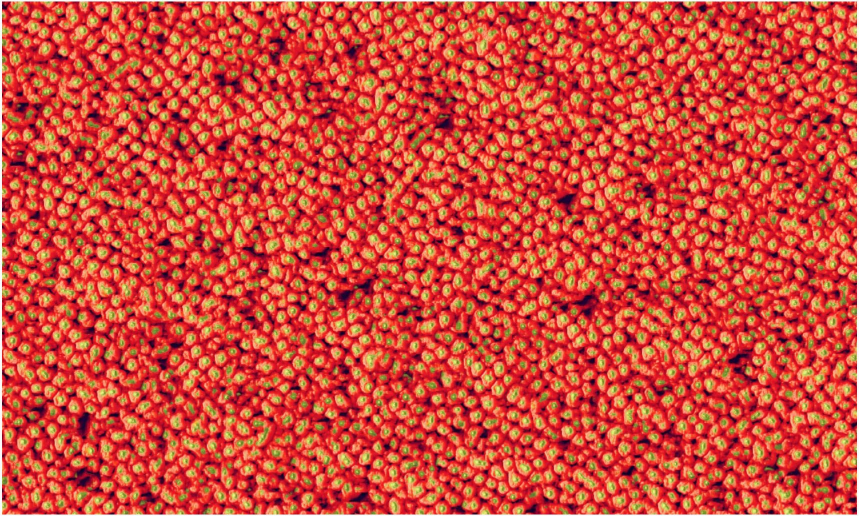


Synthesis 1280×768

Multiscale Results

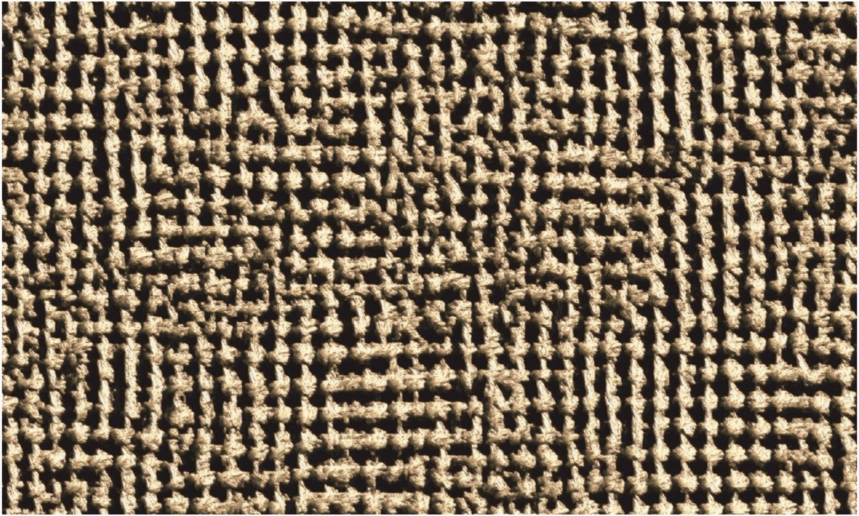
Original 128×128

Multiscale Results



Synthesis 1280×768

Original 192×192

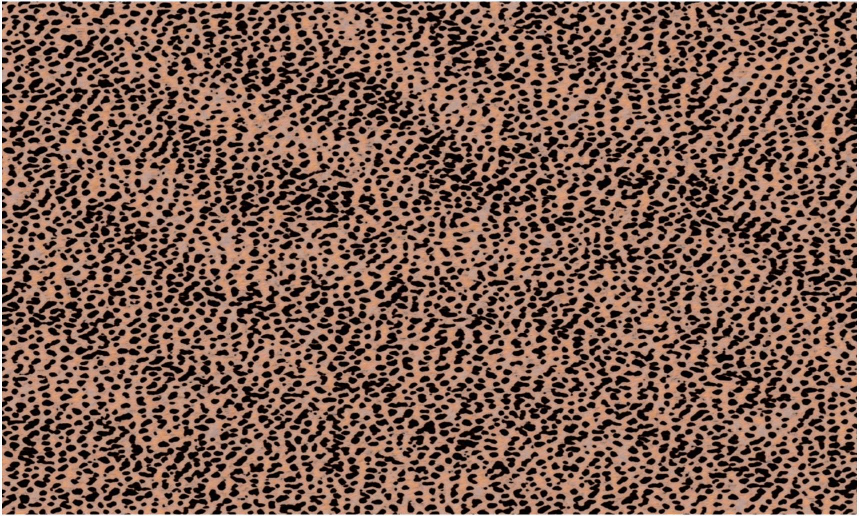


Synthesis 1280×768

Multiscale Results

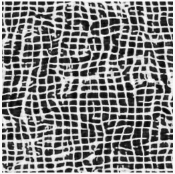
Original 200×202

Multiscale Results



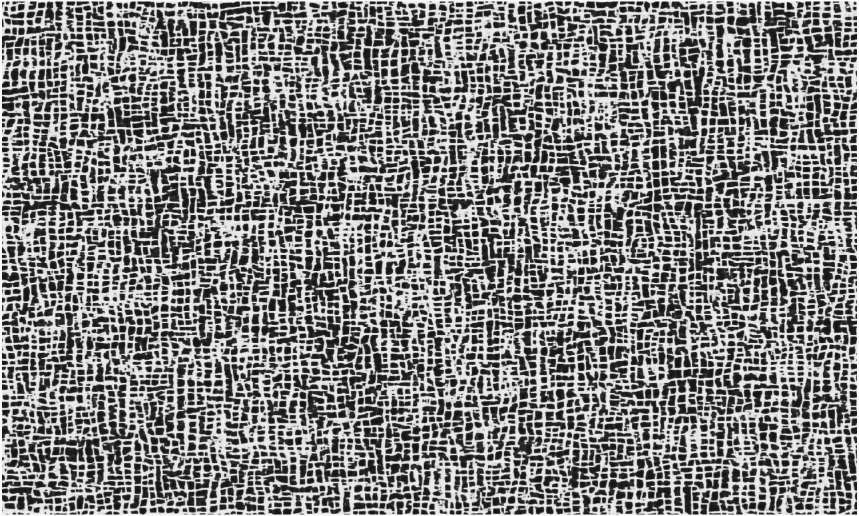
Synthesis 1280×768

Multiscale Results



Original 256×256

Multiscale Results

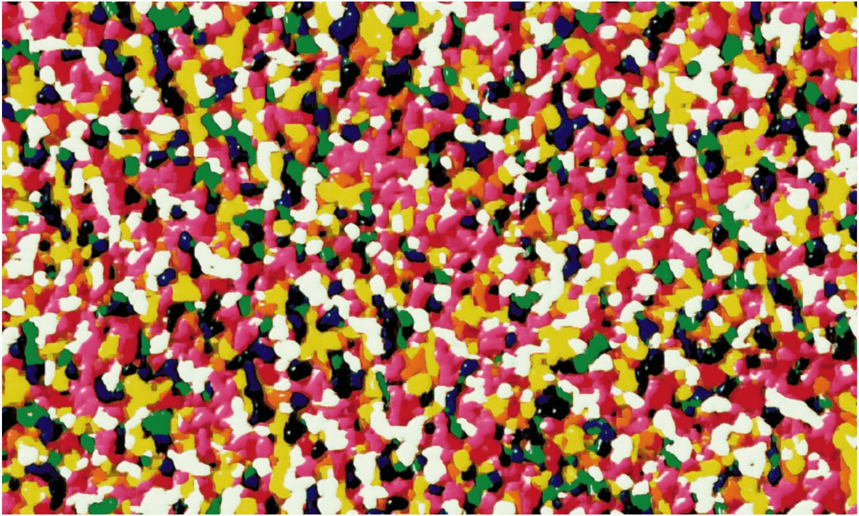


Synthesis 1280×768

Multiscale Results

Original 200×200

Multiscale Results



Synthesis 1280×768

Multiscale Results

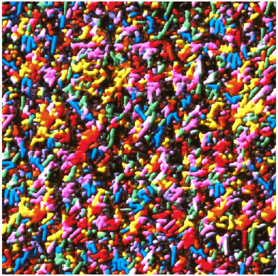
Original 200×200

Multiscale Results

Synthesis 1280×768

Comparison

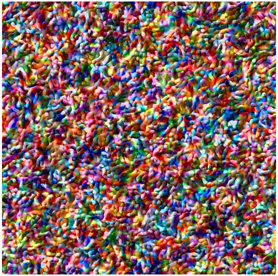
Original (512×512)



Multiscale OT (6 scales)

[Gatys et al.]

[Raad et al.]

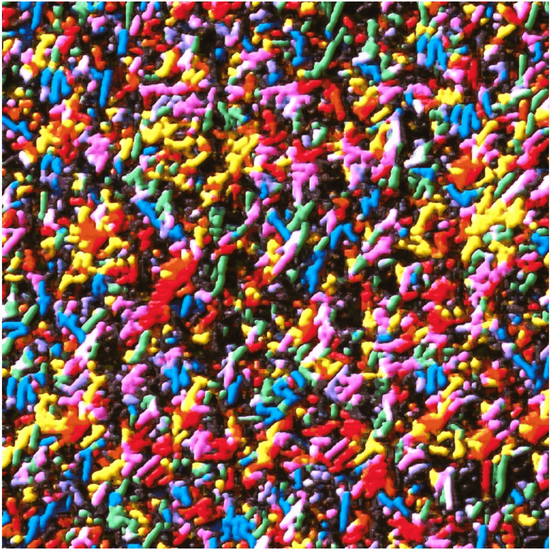


[Portilla & Simoncelli]

Comparison

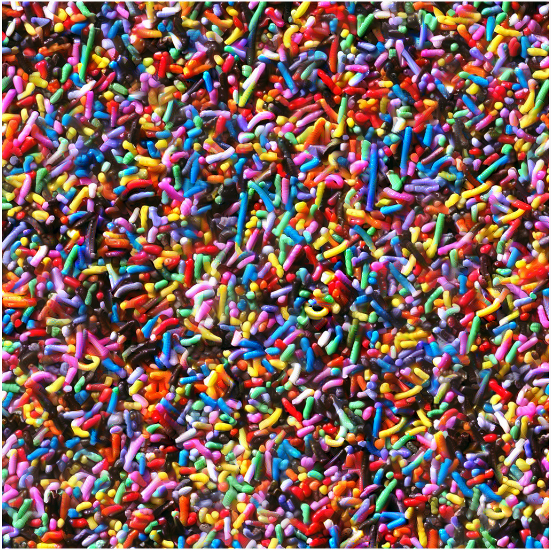
Original

Comparison



Multiscale OT

Comparison

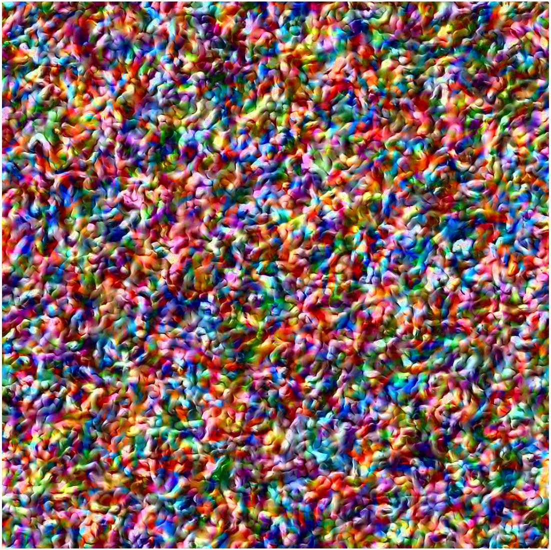


[Gatys et al.]

Comparison

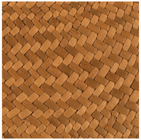
[Raad et al.]

Comparison

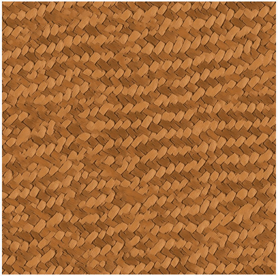


[Portilla & Simoncelli]

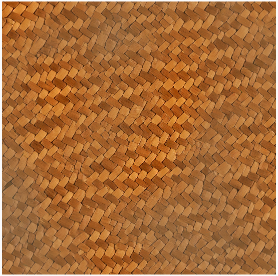
Comparison



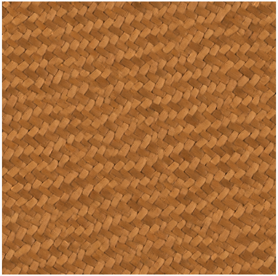
Original (512×512)



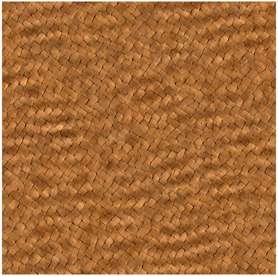
Multiscale OT (6 scales)



[Gatys et al.]

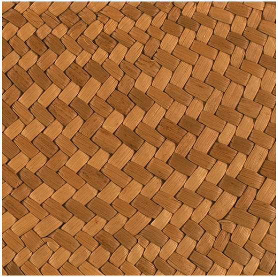


[Raad et al.]



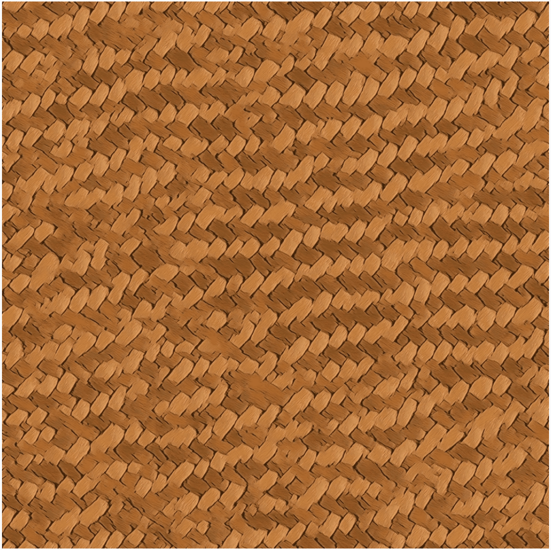
[Portilla & Simoncelli]

Comparison



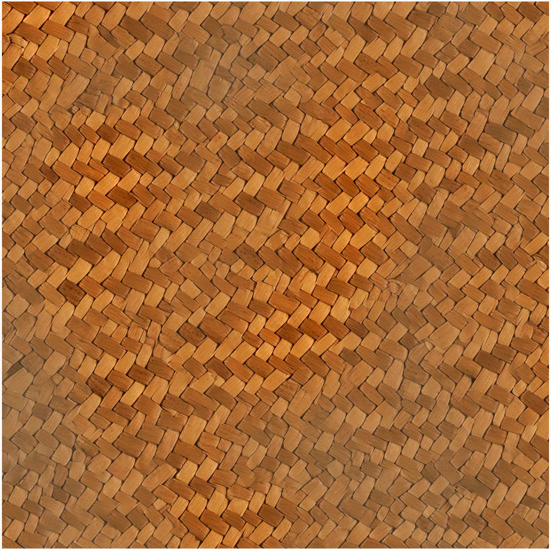
Original

Comparison



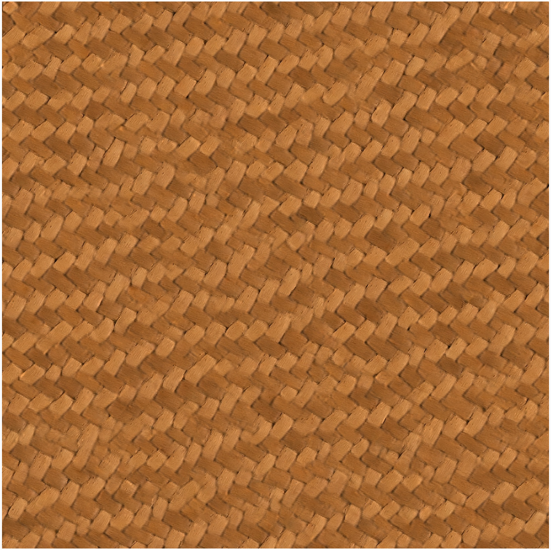
Multiscale OT

Comparison



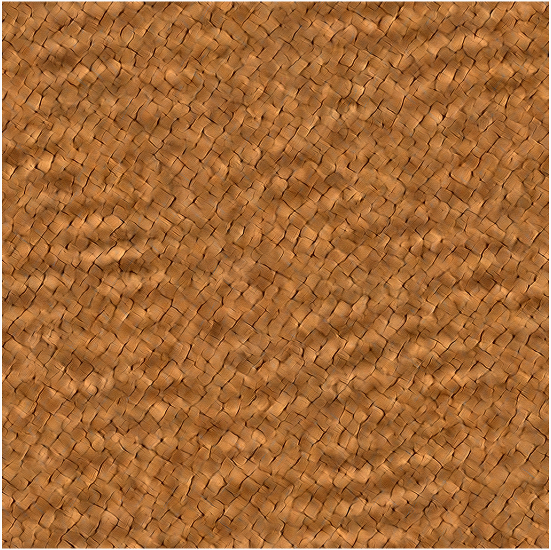
[Gatys et al.]

Comparison



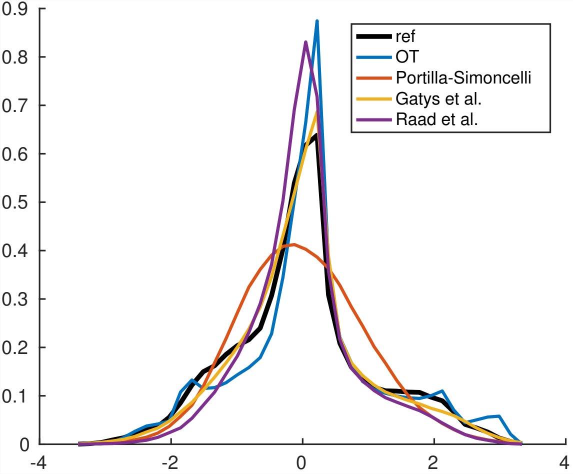
[Raad et al.]

Comparison



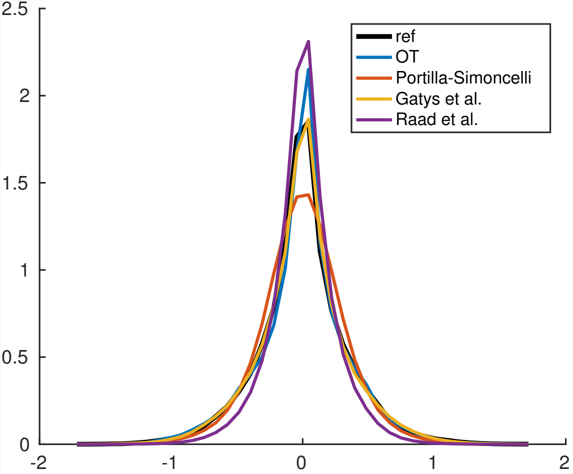
[Portilla & Simoncelli]

Output Patch Distribution on the first Principal Components



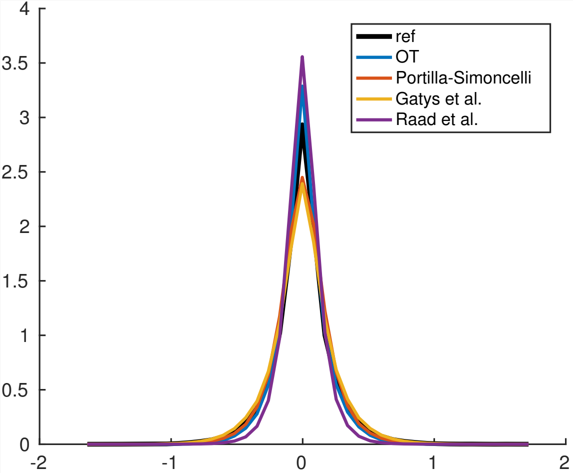
8×8 patches
PC4

Output Patch Distribution on the first Principal Components



8×8 patches
PC12

Output Patch Distribution on the first Principal Components



8×8 patches
PC18

Locally transformed Gaussian random fields

- This model of locally transformed Gaussian random field is satisfying for some macrotextures and has controlled statistical properties.
- Computing the OT plans is long and the result is only approximate (slow convergence of the ASGD).

Faster approaches have been proposed recently:

- Multiscale OT: (Leclaire and Rabin, 2021)
- OT for GMM: (Delon et al., 2022)

References

References

Arias, P., Facciolo, G., Caselles, V., and Sapiro, G. (2011). A variational framework for exemplar-based image inpainting. *International Journal of Computer Vision*, 93(3):319–347.

Blanchet, G. and Moisan, L. (2012). An explicit sharpness index related to global phase coherence. In *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*, pages 1065–1068. IEEE.

Buyssens, P., Daisy, M., Tschumperlé, D., and Lézoray, O. (2015). Exemplar-based Inpainting: Technical Review and new Heuristics for better Geometric Reconstructions. *IEEE Transactions on Image Processing*, 24(6):1809–1824.

Cook, R. L. and DeRose, T. (2005). Wavelet noise. In *ACM SIGGRAPH 2005 Papers*, SIGGRAPH '05, pages 803–811, New York, NY, USA. ACM.

Delon, J., Desolneux, A., and Leclaire, A. (2022). Transport optimal entre gmm pour la synthèse de texture. In *28ième Colloque sur le traitement du signal et des images*, number 001-071, pages p. 285–288, Nancy. GRETSI - Groupe de Recherche en Traitement du Signal et des Images.

Desolneux, A., Moisan, L., and Ronsin, S. (2012). A compact representation of random phase and Gaussian textures. In *ICASSP'12*, pages 1381–1384.

Desolneux, A., Moisan, L., and Ronsin, S. (2015). A texton for random phase and Gaussian textures.

Efros, A. A. and Freeman, W. T. (2001). Image quilting for texture synthesis and transfer. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, pages 341–346, New York, NY, USA. ACM.

Galerne, B., Gousseau, Y., and Morel, J.-M. (2011a). Micro-texture synthesis by phase randomization. *Image Processing On Line*.

Galerne, B., Gousseau, Y., and Morel, J.-M. (2011b). Random phase textures: Theory and synthesis. *IEEE Trans. Image Process.*, 20(1):257 – 267.

Galerne, B., Lagae, A., Lefebvre, S., and Drettakis, G. (2012). Gabor noise by example. *ACM Trans. Graph.*, 31(4):73:1–73:9.

Galerne, B. and Leclaire, A. (2016). An algorithm for gaussian texture inpainting. Submitted to Image Processing On Line.

Galerne, B. and Leclaire, A. (2017). Texture inpainting using efficient Gaussian conditional simulation. *SIAM Journal on Imaging Sciences*, 10(3):1446–1474.

Galerne, B., Leclaire, A., and Moisan, L. (2014). A texton for fast and flexible Gaussian texture synthesis. In *Proceedings of the 22nd European Signal Processing Conference (EUSIPCO)*, pages 1686–1690.

Galerne, B., Leclaire, A., and Moisan, L. (2016). Microtexture inpainting through Gaussian conditional simulation. In *Proceedings of IEEE Int. Conf. on Acoustics, Speech, and Signal Process. (ICASSP 2016)*.

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

Galerne, B., Leclaire, A., and Rabin, J. (2018). A texture synthesis model based on semi-discrete optimal transport in patch space. *SIAM Journal on Imaging Sciences*, 11(4):2456–2493.

Heeger, D. J. and Bergen, J. R. (1995). Pyramid-based texture analysis/synthesis. In *SIGGRAPH '95*, pages 229–238, New York, NY, USA. ACM.

Lagae, A., Lefebvre, S., Cook, R., DeRose, T., Drettakis, G., Ebert, D., Lewis, J., Perlin, K., and Zwicker, M. (2010). A survey of procedural noise functions. *Computer Graphics Forum*, 29(8):2579–2600.

Lagae, A., Lefebvre, S., Drettakis, G., and Dutré, P. (2009). Procedural noise using sparse gabor convolution. In *ACM SIGGRAPH 2009 Papers*, SIGGRAPH '09, pages 54:1–54:10, New York, NY, USA. ACM.

Lantuéjoul, C. (2002). *Geostatistical Simulation: Models and Algorithms*. Springer-Verlag, Berlin.

Leclaire, A. and Moisan, L. (2015). No-reference image quality assessment and blind deblurring with sharpness metrics exploiting Fourier phase information. *Journal of Mathematical Imaging and Vision*, 52(1):145–172.

Leclaire, A. and Rabin, J. (2021). A stochastic multi-layer algorithm for semi-discrete optimal transport with applications to texture synthesis and style transfer. *Journal of Mathematical Imaging and Vision*, 63(2):282–308.

Lugmayr, A., Danelljan, M., Gool, L. V., and Timofte, R. (2020). Srflow: Learning the super-resolution space with normalizing flow. In Vedaldi, A., Bischof, H., Brox, T., and Frahm, J., editors, *Computer Vision - ECCV 2020 - 16th European Conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part V*, volume 12350 of *Lecture Notes in Computer Science*, pages 715–732. Springer.

Moisan, L. (2011). Periodic plus smooth image decomposition. *J. Math. Imag. Vis.*, 39:161–179.

Newson, A., Almansa, A., Fradet, M., Gousseau, Y., and Pérez, P. (2014). Video Inpainting of Complex Scenes. *SIAM Journal on Imaging Sciences*, 7(4):1993–2019.

Oppenheim, A. V. and Lim, J. S. (1981). The importance of phase in signals. In *Proceedings of the IEEE*, volume 69, pages 529–541.

Perlin, K. (1985). An image synthesizer. In *Proceedings of the 12th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '85, pages 287–296, New York, NY, USA. ACM.

Raad, L. and Galerne, B. (2017). Efros and Freeman image quilting algorithm for texture synthesis. *Image Processing On Line*, 7:1–22.

Ronsin, S., Biermé, H., and Moisan, L. (2016). The Billard theorem for multiple random Fourier series. *Journal of Fourier Analysis and Applications*. to appear.

van Wijk, J. J. (1991). Spot noise texture synthesis for data visualization. In *Proceedings of the 18th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '91, pages 309–318, New York, NY, USA. ACM.

Wei, L.-Y., Lefebvre, S., Kwatra, V., and Turk, G. (2009). State of the art in example-based texture synthesis. In *Eurographics 2009, State of the Art Report, EG-STAR*. Eurographics Association.

Xia, G.-S., Ferradans, S., Peyré, G., and Aujol, J.-F. (2014). Synthesizing and mixing stationary Gaussian texture models. *SIAM J. on Imaging Science*, 8(1):476–508.