

# Generative models for images I: Introduction and texture synthesis

---

Bruno Galerne

`bruno.galerie@univ-orleans.fr`

**Master MVA 2024-25**

Tuesday January 14, 2025

Institut Denis Poisson

**Université d'Orléans**, Université de Tours, CNRS

Institut universitaire de France (IUF)

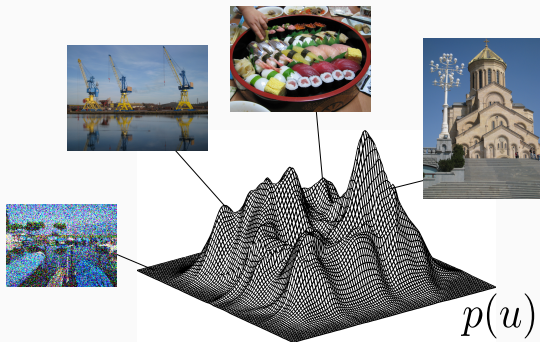
## **Introduction on generative models**

---



# Generative models

1. Model and/or learn a distribution  $p(u)$  on the space of images.



(source: Charles Deledalle)

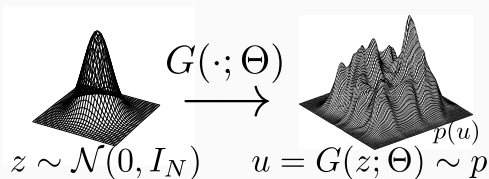
The images may represent:

- different instances of the same texture image,
- all images naturally described by a dataset of images,
- any image

2. Generate samples from this distribution.

# Generative models

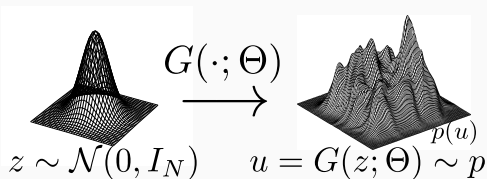
1. Model and/or learn a distribution  $p(u)$  on the space of images.
2. Generate samples from this distribution.



- $z$  is a generic source of randomness, often called the latent variable.
- If  $G(\cdot; \Theta)$  is known, then  $p = G(\cdot; \Theta)_{\#} \mathcal{N}(0, I_n)$  is the push-forward of the latent distribution.

# Generative models

1. Model and/or learn a distribution  $p(u)$  on the space of images.
2. Generate samples from this distribution.



- $z$  is a generic source of randomness, often called the latent variable.
- If  $G(\cdot; \Theta)$  is known, then  $p = G(\cdot; \Theta)_{\#} \mathcal{N}(0, I_n)$  is the push-forward of the latent distribution.

The generator  $G(\cdot; \Theta)$  can be:

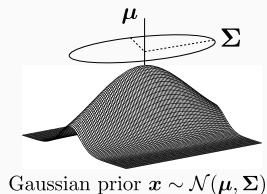
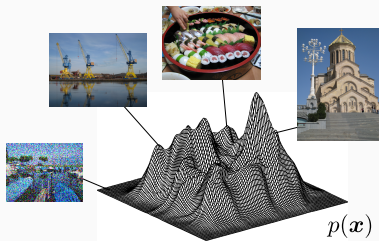
- A deterministic function (e.g. convolution operator),
- A neural network with learned parameter,
- An iterative optimization algorithm (gradient descent,...),
- A stochastic sampling algorithm (e.g. MCMC, Langevin diffusion,...).

# Image generation: Gaussian model

- Consider a **Gaussian model** for the distribution of images  $\mathbf{x}$  with  $d$  pixels:

$$\mathbf{x} \sim \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{\sqrt{(2\pi)^d |\boldsymbol{\Sigma}|}} \exp \left[ -(\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}) \right]$$

- $\boldsymbol{\mu}$ : mean image,
- $\boldsymbol{\Sigma}$ : covariance matrix of images.



(source: Charles Deledalle)

# Image generation: Gaussian model

- Take a training dataset  $\mathcal{D}$  of images:

$$\mathcal{D} = \{\mathbf{x}_1, \dots, \mathbf{x}_N\} \\ = \left\{ \begin{array}{c} \text{img}_1, \text{img}_2, \text{img}_3, \text{img}_4, \text{img}_5, \text{img}_6, \dots \\ \times N \end{array} \right\}$$

- Estimate the mean

$$\hat{\boldsymbol{\mu}} = \frac{1}{N} \sum_i \mathbf{x}_i = \text{img}_{\text{mean}}$$

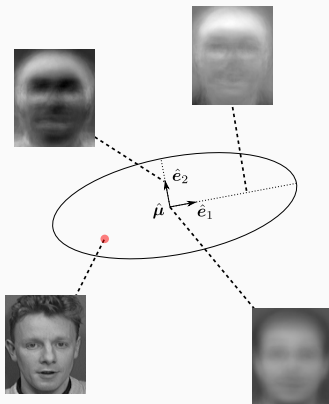
- Estimate the covariance matrix:  $\hat{\boldsymbol{\Sigma}} = \frac{1}{N} \sum_i (\mathbf{x}_i - \hat{\boldsymbol{\mu}})(\mathbf{x}_i - \hat{\boldsymbol{\mu}})^T = \hat{\mathbf{E}} \hat{\boldsymbol{\Lambda}} \hat{\mathbf{E}}^T$

$$\hat{\mathbf{E}} = \left\{ \begin{array}{c} \text{img}_1, \text{img}_2, \text{img}_3, \text{img}_4, \text{img}_5, \text{img}_6, \dots \\ \times N \end{array} \right\}$$

eigenvectors of  $\hat{\boldsymbol{\Sigma}}$ , i.e., main variation axis

# Image generation: Gaussian model

You now have learned a **generative model**:



How to generate samples from  $\mathcal{N}(\hat{\mu}, \hat{\Sigma})$ ?

$$\begin{cases} z & \sim \mathcal{N}(0, I_d) \quad \leftarrow \text{Generate random latent variable} \\ x & = \hat{\mu} + \hat{E}\hat{\Lambda}^{1/2}z \end{cases}$$

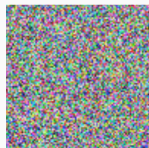


**The model does not generate realistic faces.**

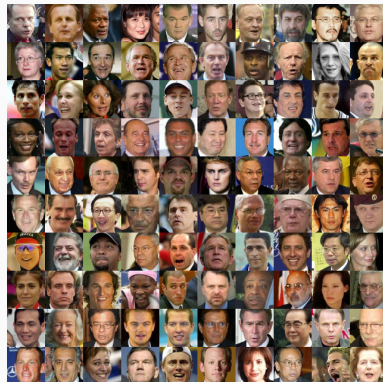
- The Gaussian distribution assumption is too simplistic.
- Each generated image is just a linear random combination of the eigenvectors (with independence !).
- The generator corresponds to a one layer linear neural network (without non-linearities).

# Image generation: Gaussian model

Noise  $\sim N(0,1)$



Generative  
Model



- Deep generative modeling consists in learning non-linear generative models to reproduce complex data such as realistic images.
- It relies on deep neural networks and several solutions have been proposed since the “Deep learning revolution” (2012).



## Texture synthesis with a stationary Gaussian model: (Galerne et al., 2011)

- Data: A single texture image  $h$ .
- Inferred distribution:  $p$  is the stationary Gaussian distribution with similar mean and covariance statistics.
- $z$  is a Gaussian white noise image (each pixel is iid with standard normal distribution).
- $G$  is a convolution operator with known parameters  $\Theta$ .

Data



Spot  $h$

Generated images



$G(z_1; \Theta)$



$G(z_2; \Theta)$



$G(z_3; \Theta)$

## Generative Adversarial Networks: (Goodfellow et al., 2014)

- Data: A database of images.
- Inferred distribution: Not explicit, push-forward measure given by generator.
- $z$  is a Gaussian array in a latent space.
- $G(\cdot; \Theta)$  is a (convolutional) neural network with parameters  $\Theta$  learned using an adversarial discriminator network  $D(\cdot; \Theta_D)$ .

Data



MNIST: handwritten digits

Generated images



Fake images (100 epochs)

Image size:  
28×28 px

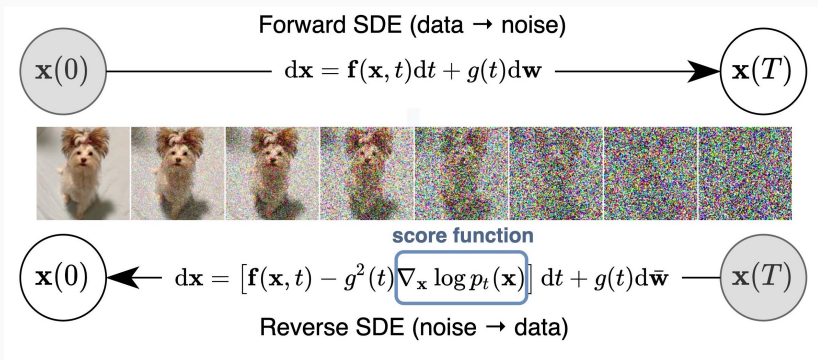
## Generative Adversarial Networks: Style GAN (Karras et al., 2019)



Image size:  
 $1024 \times 1024$  px  
(source: Karras et al.)

# Denoising diffusion probabilistic models

- Learn to revert a degradation process: Add more and more noise to an image.
- First similar model (Sohl-Dickstein et al., 2015)



(source: Yang Song)

- Probably the most promising framework these days... but things change very quickly in this field!

# Diffusion models

(Ho et al., 2020): Denoising Diffusion Probabilistic Models (DDPM): One of the first paper producing images with reasonable resolution.

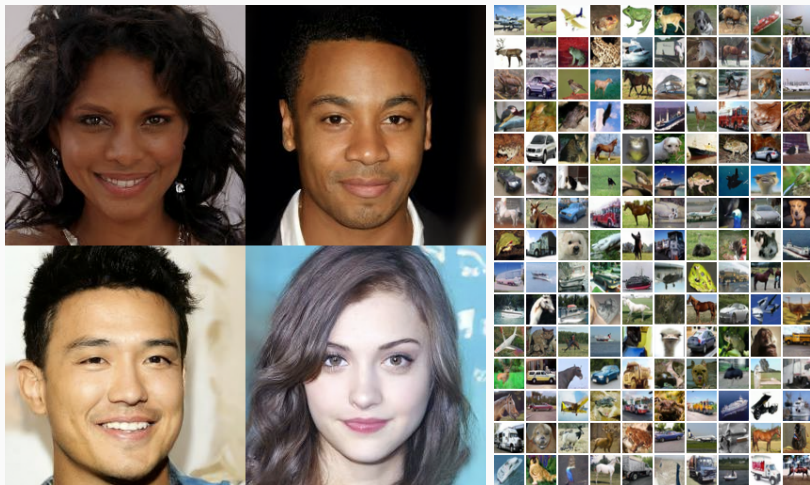


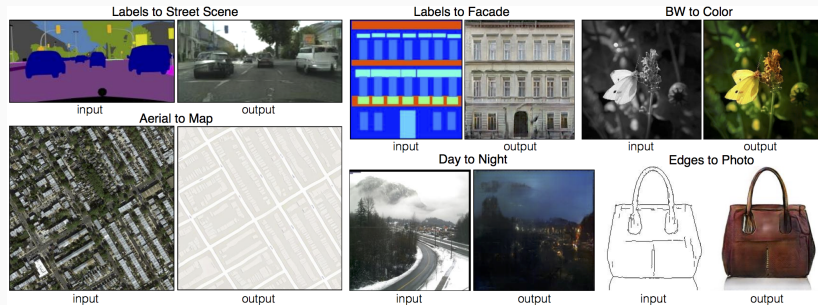
Figure 1: Generated samples on CelebA-HQ  $256 \times 256$  (left) and unconditional CIFAR10 (right)

Why generative models are interesting ?

- **Generating realistic images is important by itself** for entertainment industry (visual effects, video games, augmented reality...), design, advertising industry,...
- **Good image model leads to good image processing:** Generative models can be used as a parametric space for solving inverse problems. Example: Inpainting of a portrait image.
- Also generative models opens the way to **non trivial image manipulation** using **conditional generative models**.

# Conditional generative models: Examples

## Pix2pix: Image-to-Image Translation with Conditional Adversarial Nets (Isola et al., 2017)



- GAN conditioned on input image.
- Generator: U-net architecture
- Discriminator: Patch discriminator applied to each patch
- Opens the way for new creative tools

(source: Isola et al.)

# Conditional generative models: Examples

Latest trends using **diffusion models**: Text to image generation

- DALL·E 1 & 2: Creating Images from Text (Open AI, January 2021 and April 2022)
- Imagen, Google research (May 2022)

DALL·E 2 (Open AI)

Input: An astronaut riding a horse in a photorealistic style



Imagen (Google)

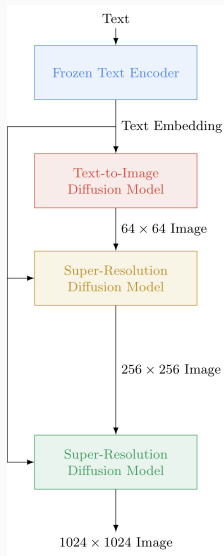
Input: A dog looking curiously in the mirror, seeing a cat.



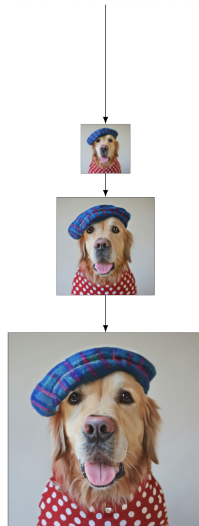


# Conditional generative models: Examples

## Imagen pipeline:



“A Golden Retriever dog wearing a blue checkered beret and red dotted turtleneck.”



(source: (Saharia et al., 2022))

# Conditional generative models: Examples

In August 2022, StableDiffusion was released:

- Based on the paper (Rombach et al., 2022)
- **Open source!**

futuristic tree house, hyper realistic,  
epic composition, cinematic, landscape  
vista photography by Carr Clifton &  
Galen Rowell, Landscape veduta photo  
by Dustin Lefevre & tdraw, detailed  
landscape painting by Ivan Shishkin,  
rendered in Enscape, Miyazaki, Nausicaa  
Ghibli, 4k detailed post processing,  
unreal engine  
Steps: 50, Sampler: PLMS, CFG scale:  
9, Seed: 2937258437



Diffusion models are considered mature models and have been used in a large variety of frameworks.

- Diffusion models **beyond image generation**: Text to video, motion generation, proteins, soft robots,...
- **Control of (latent) diffusion models**((Ruiz et al., 2023), (Zhang et al., 2023),...)
- **Diffusion models as priors for imaging inverse problems** ((Chung et al., 2023), (Song et al., 2023), lot of applications in medical imaging, etc.)

## DreamBooth: Fine Tuning Text-to-Image Diffusion Models for Subject-Driven Generation

Nataniel Ruiz\*,<sup>1,2</sup>

Yael Pritch<sup>1</sup>

Yuanzhen Li<sup>1</sup>

Michael Rubinstein<sup>1</sup>

Varun Jampani<sup>1</sup>

Kfir Aberman<sup>1</sup>

<sup>1</sup> Google Research    <sup>2</sup> Boston University



Figure 1. With just a few images (typically 3-5) of a subject (left), *DreamBooth*—our AI-powered photo booth—can generate a myriad of images of the subject in different contexts (right), using the guidance of a text prompt. The results exhibit natural interactions with the environment, as well as novel articulations and variation in lighting conditions, all while maintaining high fidelity to the key visual features of the subject.

(source: (Ruiz et al., 2023))

## Adding Conditional Control to Text-to-Image Diffusion Models

Lvmin Zhang, Anyi Rao, and Maneesh Agrawala  
Stanford University

{lvmin, anyirao, maneesh}@cs.stanford.edu



Figure 1: Controlling Stable Diffusion with learned conditions. ControlNet allows users to add conditions like Canny edges (top), human pose (bottom), *etc.*, to control the image generation of large pretrained diffusion models. The default results use the prompt “a high-quality, detailed, and professional image”. Users can optionally give prompts like the “chef in kitchen”.

(source: ControlNet (Zhang et al., 2023))

## Diffusion posterior sampling for general noisy inverse problems (Chung et al., 2023)

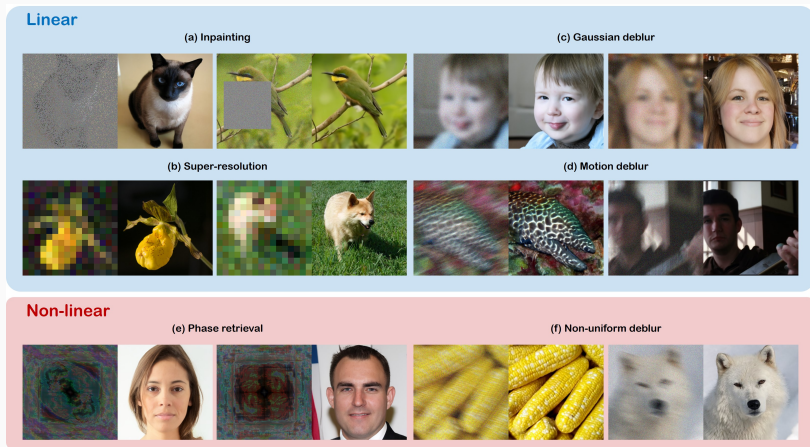


Figure 1: Solving noisy linear, and nonlinear inverse problems with diffusion models. Our reconstruction results (right) from the measurements (left) are shown.

(source: (Chung et al., 2023))

## DiffusionLight: Light Probes for Free by Painting a Chrome Ball

Pakkapon Phongthawee\*<sup>1</sup>

Worameth Chinchuthakun\*<sup>1,2</sup>

Nontaphat Sinsunthithet<sup>1</sup>

Amit Raj<sup>3</sup>

Varun Jampani<sup>4</sup>

Pramook Khungurn<sup>5</sup>

Supasorn Suwajanakorn<sup>1</sup>

<sup>1</sup>VISTEC

<sup>2</sup>Tokyo Tech

<sup>3</sup>Google Research

<sup>4</sup>Stability AI

<sup>5</sup>Pixiv

<https://diffusionlight.github.io/>

- Text-to-video generation.
- Image generation broadly available via conversational agent (ChatGPT, Le Chat by Mistral (based on Flux Pro),...).
- Diffusion models used as “world model”.



Figure 1. We leverage a pre-trained diffusion model (Stable Diffusion XL) for light estimation by rendering an HDR chrome ball. In each scene, we show our normally exposed chrome ball on top and our underexposed version, which reveals bright light sources, on the bottom.

(source: (Phongthawee et al., 2024))

Courses shared with **Arthur Leclaire (Telecom Paris)**.

## Warm up (today):

- Introduction to generative models for images (done)
- Texture synthesis as a simple problem for generative modeling

## Part I: Established generative models frameworks based on CNN

1. Generative Adversarial Networks (GANs)  
(**AL**, 2 courses)
  - Training of GANs
  - Link with optimal transport
2. Other generative models based on likelihood maximization  
(**BG**, 1 course)
  - Variational AutoEncoders (VAEs)
  - Normalizing Flows (NFs)
3. Application of generative models for imaging inverse problems  
(**AL**, 1 course)
  - Classical imaging problems: Super-Resolution, inpainting,...
  - Image-to-image translations



## Part II: Diffusion or Score-Based Generative Models (SGM) (BG, 2 courses)

1. Diffusion models in pixel space
  - Time reversal of stochastic processes
  - Denoising Diffusion Probabilistic Models (DDPM)
  - Denoising Implicit Diffusion Models (DDIM)
  - Application to imaging inverse problems
2. Latent Diffusion Models (LDM)
  - Stable diffusion pipeline
  - Text-to-image synthesis
  - Controllable diffusion

## Part III: Plug-and-Play methods for imaging inverse problems (AL, 2 courses)

1. Proximal splitting and PnP methods
2. Convergence of PnP methods via fixed point
  - Fixed point theorem for “averaged” operators
  - Convergence of PnP with an “averaged” denoiser
  - Learning a denoiser with Lipschitz constraint
3. Convergence of PnP methods via non-convex optimization
  - Convergence of proximal splitting in the non-convex case
  - Gradient-step regularization, and convergence of PnP
  - Applications to several inverse problems

## Validation (Master MVA)

- Un devoir maison intermédiaire (obligatoire) : Sous forme de complément de TP cette année
- Un projet final (étude d'un article en groupe de 2 étudiants) avec soutenance orale à Telecom Paris (plateau de Saclay).
- Merci de vous inscrire via le google form.
- Email: [generative.modeling.mva@gmail.com](mailto:generative.modeling.mva@gmail.com)

Introduction on generative models

Texture synthesis

Discrete Fourier transform and texture

Textures synthesis using wavelet statistics

Textures synthesis using CNN statistics

Discussion on variational texture synthesis

## Texture synthesis

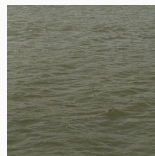
---

# What is a texture?

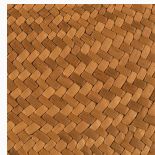
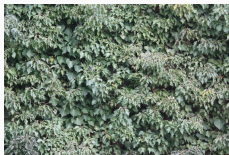
A minimal definition of a **texture** image is an “image containing repeated patterns” (Wei et al., 2009). The family of patterns reflects a certain amount of randomness, depending on the nature of the texture.

Two main subclasses:

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



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



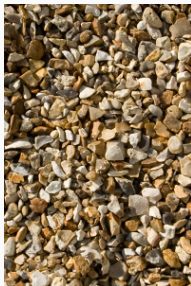
# Textures and scale of observation

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



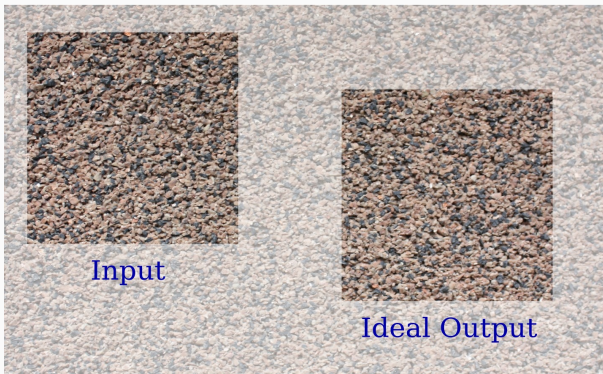
Macro-texture



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: 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:

## 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

## 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...

## **Discrete Fourier transform and texture**

---

- 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 with **periodic boundary**



# 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  $u$ .

## Computation:

- The Fast Fourier Transform (FFT) algorithm computes  $\hat{u}$  in  $\mathcal{O}(MN \log(MN))$  operations.

# Discrete Fourier transform of digital images

## Symmetry property:

- Since  $u$  is real-valued,  $\hat{u}(-s, -t) = \overline{\hat{u}(s, t)}$ .
- $|\hat{u}|$ : **Fourier modulus** of  $u$  is even.
- $\arg(\hat{u})$ : **Fourier phase** of  $u$  is odd.

## Visualization of the DFT:

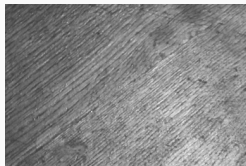
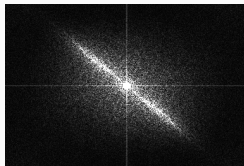
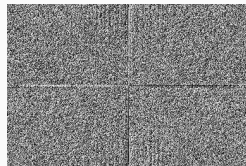


Image  $u$



Modulus  $|\hat{u}|$



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

- Beware of rectangular domains: Orthogonal directions of the waves when spectrum displayed in a square domain (rectangular pixels).

# Discrete Fourier transform of digital images

## Symmetry property:

- Since  $u$  is real-valued,  $\hat{u}(-s, -t) = \overline{\hat{u}(s, t)}$ .
- $|\hat{u}|$ : **Fourier modulus** of  $u$  is even.
- $\arg(\hat{u})$ : **Fourier phase** of  $u$  is odd.

## Visualization of the DFT:

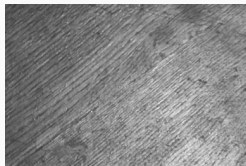
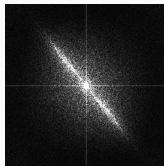


Image  $u$



Modulus  $|\hat{u}|$

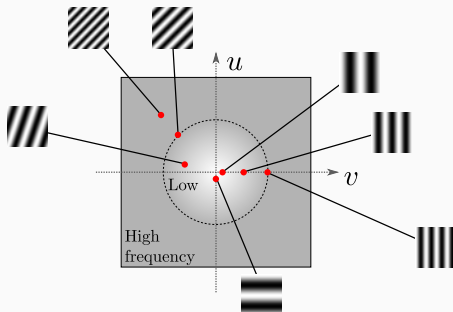


Phase  $\arg(\hat{u})$

- Beware of rectangular domains: Orthogonal directions of the waves when spectrum displayed in a square domain (rectangular pixels).

# Discrete Fourier transform of digital images

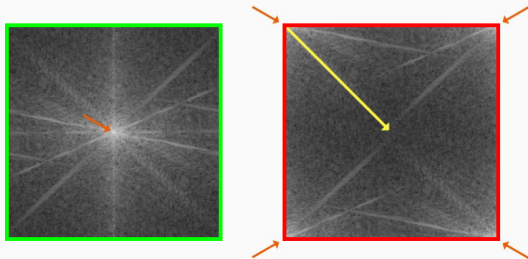
- Represent each Fourier coefficients on a 2d grid



- $|\hat{u}_{s,t}|$ : contribution of frequency  $\sqrt{s^2 + t^2}$  in the direction  $(s, t)$ .
- $\arg \hat{u}_{s,t}$ : phase shift of frequency  $\sqrt{s^2 + t^2}$  in the direction  $(s, t)$ .
- Center  $\equiv$  low frequencies
- Periphery  $\equiv$  high frequencies

# Discrete Fourier transform of digital images

## Recenter / Shift



- Option 1: place the zero-frequency in the middle
  - Good way to visualize it
- Option 2: place the zero-frequency at top left location
  - Good way to manipulate it, used by Python, Matlab,...
- In `numpy`:
  - Forward transform: `dtfu = np.fft.fftshift(np.fft.fft2(u))`
  - Inverse transform: `v = np.fft.ifft2(np.fft.ifftshift(dtfu))`

(source: C. Deledalle)

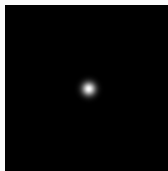
# Discrete Fourier transform of digital images: Convolution

**Convolution product:** Given two images  $f$  and  $g \in \mathbb{R}^{M \times N}$ , the convolution product between  $f$  and  $g$  is the image  $f * g$  defined by

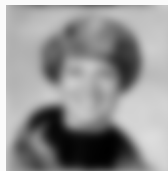
$$(f * g)(x) = \sum_{y \in \Omega} f(x - y)g(y), \quad x \in \Omega.$$



Image  $f$



Gaussian  
kernel  $g$



Blurred image  $f * g$

# Discrete Fourier transform of digital images: Convolution

**Convolution product:** Given two images  $f$  and  $g \in \mathbb{R}^{M \times N}$ , the convolution product between  $f$  and  $g$  is the image  $f * g$  defined by

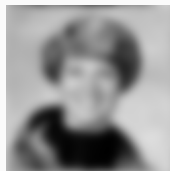
$$(f * g)(x) = \sum_{y \in \Omega} f(x - y)g(y), \quad x \in \Omega.$$



Image  $f$



Gaussian  
kernel  $g$



Blurred image  $f * g$

# Discrete Fourier transform of digital images: Convolution

**Convolution product:** Given two images  $f$  and  $g \in \mathbb{R}^{M \times N}$ , the convolution product between  $f$  and  $g$  is the image  $f * g$  defined by

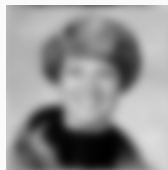
$$(f * g)(x) = \sum_{y \in \Omega} f(x - y)g(y), \quad x \in \Omega.$$



Image  $f$



Gaussian  
kernel  $g$



Blurred image  $f * g$

**Convolution theorem:**

$$\widehat{f * g}(\xi) = M N \hat{f}(\xi) \hat{g}(\xi), \quad \xi \in \hat{\Omega}.$$

- The operator “convolution by  $f$ ” is diagonalized by the Fourier transform.
- Computing the convolution between  $f$  and  $g$  has the cost of 3 FFT calls.
- Useful for large support only (not used for CNN).



# Modulus and phase of a digital image

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

Image 1



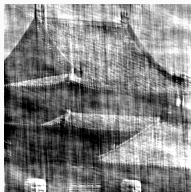
Image 2



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



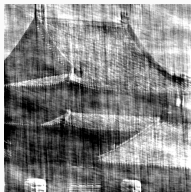
Image 2



Modulus of 1  
& phase of 2



Modulus of 2  
& phase of 1



- Geometric contours are mostly contained in the phase.

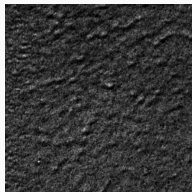
# Modulus and phase of a digital image

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

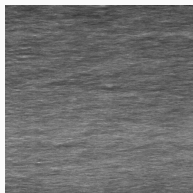
Image 1



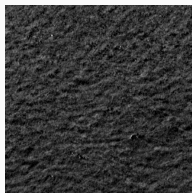
Image 2



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

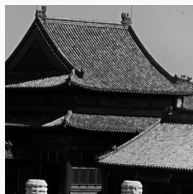
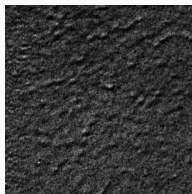
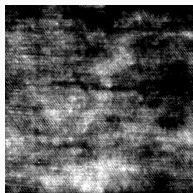


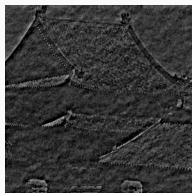
Image 2



Modulus of 1  
& phase of 2



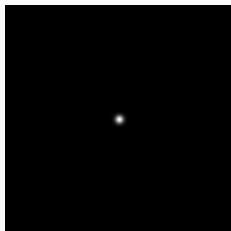
Modulus of 2  
& phase of 1



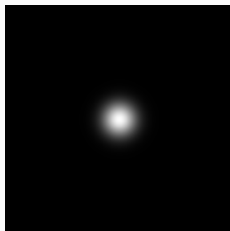
- Geometric contours are mostly contained in the phase.
- Textures are mostly contained in the modulus.

# Random Phase Noise (RPN)

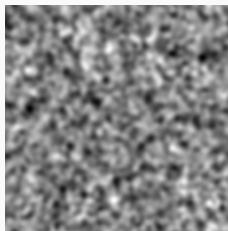
- Texture synthesis algorithm: **random phase noise (RPN)**: (van Wijk, 1991)
  1. Compute the DFT  $\hat{h}$  of the input  $h$
  2. Compute a random phase  $\theta$  using a pseudo-random number generator
  3. Set  $\hat{Z} = |\hat{h}| e^{i\theta}$  (or  $\hat{Z} = \hat{h}e^{i\theta}$ )
  4. Return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



Original image  $h$



Modulus  $|\hat{h}|$



RPN associated with  $h$

## Texture synthesis with a stationary Gaussian model: (Galerne et al., 2011)

- Stationary Gaussian distribution with similar mean and covariance:

$$G(z; \Theta) = \text{mean}(\mathbf{h}) + \frac{1}{\sqrt{MN}}(\mathbf{h} - \text{mean}(\mathbf{h})) * \mathbf{z}$$

where  $\mathbf{z}$  is a Gaussian white noise image (each pixel is iid with standard normal distribution).

**Closely related to RPN: Same covariance imposes same Fourier modulus (up to some multiplicative noise)**

Data



Spot  $\mathbf{h}$

Generated images



$G(\mathbf{z}_1; \Theta)$



$G(\mathbf{z}_2; \Theta)$



$G(\mathbf{z}_3; \Theta)$

## **Textures synthesis using wavelet statistics**

---

## References:

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

## Statistical constraints:

- Histogram of colors
- Histogram of “wavelet” coefficients at each scale, more precisely the steerable pyramid transform ([Simoncelli et al., 1992](#))

## Algorithm:

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

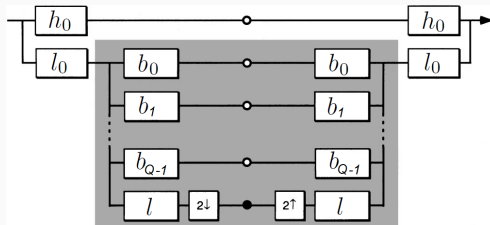
## Two main tools:

- Steerable pyramid decomposition and reconstruction
- Histogram matching



# Steerable pyramid decomposition



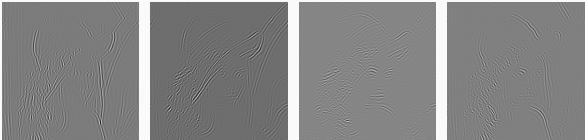
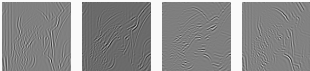

## Diagram for the steerable pyramid:



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


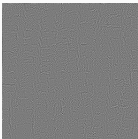
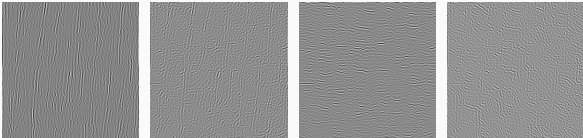
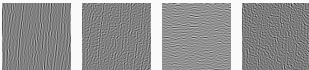

# Steerable pyramid decomposition

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

Original image	Associated steerable pyramid
	 <p>High frequency residual</p>  <p>1st scale of oriented subbands with angle <math>\theta = 0, \frac{\pi}{4}, \frac{\pi}{2}, \text{ and } \frac{3\pi}{4}</math></p>  <p>2nd scale of oriented subbands with angle <math>\theta = 0, \frac{\pi}{4}, \frac{\pi}{2}, \text{ and } \frac{3\pi}{4}</math></p> 


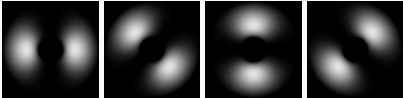
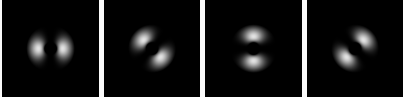





# Steerable pyramid decomposition of a texture

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

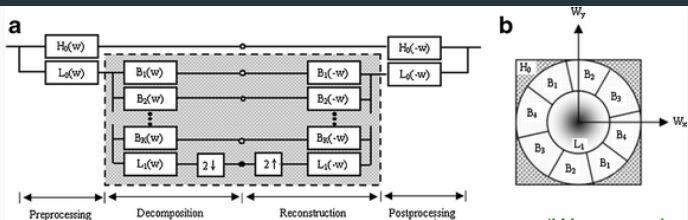
Original image	Associated steerable pyramid
	 High frequency residual  1st scale of oriented subbands with angle $\theta = 0, \frac{\pi}{4}, \frac{\pi}{2}, \text{ and } \frac{3\pi}{4}$  2nd scale of oriented subbands with angle $\theta = 0, \frac{\pi}{4}, \frac{\pi}{2}, \text{ and } \frac{3\pi}{4}$ 



# Steerable pyramid decomposition: Fourier domain

Fourier modulus of filters	Sum of squared moduli
 <p>High frequency filter <math>H_0</math></p>  <p>1st scale of oriented subband filters with angle <math>\theta = 0, \frac{\pi}{4}, \frac{\pi}{2}, \text{ and } \frac{3\pi}{4}</math></p>  <p>1st scale of oriented subband filters with angle <math>\theta = 0, \frac{\pi}{4}, \frac{\pi}{2}, \text{ and } \frac{3\pi}{4}</math></p> 	   

# Steerable pyramid reconstruction



(source: (Wang et al., 2014))

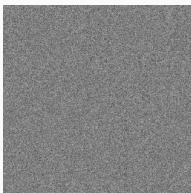
- The steerable filters are designed so that each stage of the diagram has a flat system response.

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

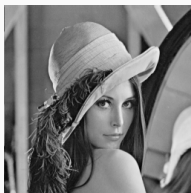
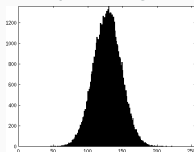
- Denote by  $A$  the matrix of the pyramid decomposition operator, then the matrix of the reconstruction operator is simply the transpose  $A^T$ .
- But the flat response ensures that  $A^T A = Id$ , so that  $A^T$  is also the pseudo-inverse of  $A$ :  $A^\dagger = (A^T A)^{-1} A^T = A^T$ .
- Important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

# Histogram matching

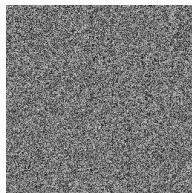
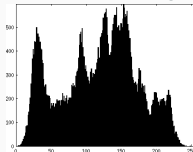
- Example of histogram matching: The histogram of a Gaussian white noise is matched with the histogram of the Lena image.



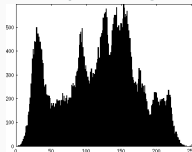
Input image



Reference image

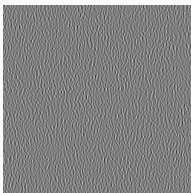


Output image

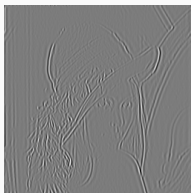
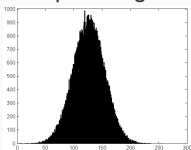


# Histogram matching

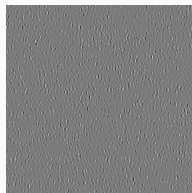
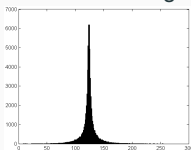
- Example of histogram matching of a pyramid image: The histogram of an oriented pyramid image of a Gaussian white noise is matched with the corresponding image in Lena's pyramid



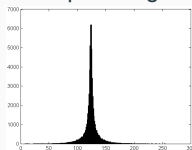
Input image



Reference image



Output image





---

## Algorithm 1: Histogram matching

---

**Input** : Input image  $u$ , reference image  $v$  (both images have size  $M \times N$ )

**Output**: Image  $u$  having the same histogram as  $v$  (the input  $u$  is lost)

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

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

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

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

---

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

---

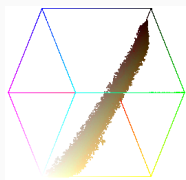
**Input** : Number of scales  $P$ , number of orientations  $Q$ , texture image  $u$  of size  $M \times N$  such that  $M$  and  $N$  are multiples of  $2^P$ , number of iterations  $N_{\text{iter}}$

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

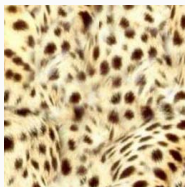
1. **Input analysis:**
  2. Compute and store the steerable pyramid with  $P$  scales and  $Q$  orientations of the input texture  $u$ .
  3. **Output synthesis:**
  4. Initialize  $v$  with a Gaussian white noise.
  5. Match the gray-level histogram of  $v$  with the gray-level histogram of  $u$ .
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

# Heeger and Bergen algorithm for color textures

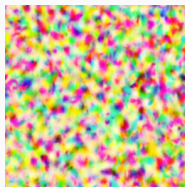
- The Heeger-Bergen algorithm is designed for grayscale images (one channel).
- The principal limitation comes from the histogram matching algorithm.
- Applying the algorithm to each channel of an RGB image does not work: Indeed the output color channels are independent !



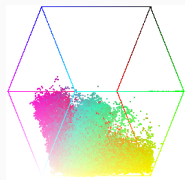
Input color cube



Input texture



Output texture

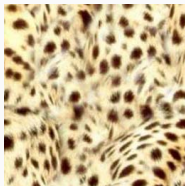


Output color cube

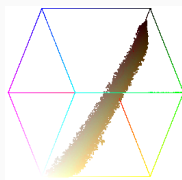
- RGB cube visualizatiton from the IPOL paper and demo (Lisani et al., 2011)

# RGB channels non independence

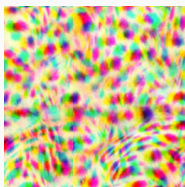
- Never treat the RGB color channels as being independent.



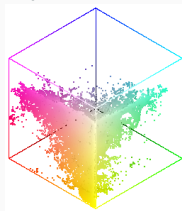
Input texture



Input color cube



Independent channel shift

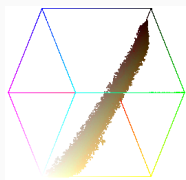


Color cube after random shift

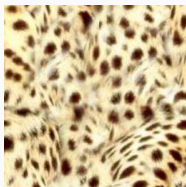
- RGB cube visualizatiton from the IPOL paper and demo (Lisani et al., 2011)

# Heeger and Bergen algorithm for color textures

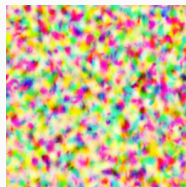
- The Heeger-Bergen algorithm is designed for grayscale images (one channel).
- The principal limitation comes from the histogram matching algorithm.
- Applying the algorithm to each channel of an RGB image does not work: Indeed the output color channels are independent !



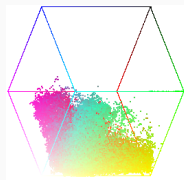
Input color cube



Input texture



Output texture



Output color cube

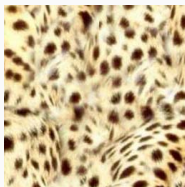
- **Solution:** Change the color space so that the independence of the color channels is acceptable.

# Heeger and Bergen algorithm for color textures

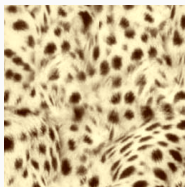
## PCA color space of an image:

- Compute the PCA of the colors seen as a point cloud in  $\mathbb{R}^3$ .
- This corresponds to find (one of) the orthonormal basis of  $\mathbb{R}^3$  that diagonalizes

$$C_h(0) = \frac{1}{MN} \sum_{t \in \Omega} \begin{pmatrix} h_r(t) - m_r \\ h_g(t) - m_g \\ h_b(t) - m_b \end{pmatrix} \begin{pmatrix} h_r(t) - m_r \\ h_g(t) - m_g \\ h_b(t) - m_b \end{pmatrix}^T \in \mathbb{R}^{3 \times 3}.$$



Color texture



1st PC



2nd PC

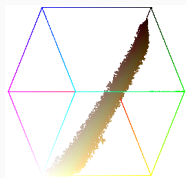


3rd PC

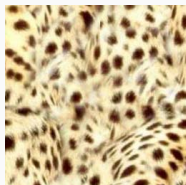
- In general, the dynamic of the texture is mainly contained in the first principal component.

# Heeger and Bergen algorithm for color textures

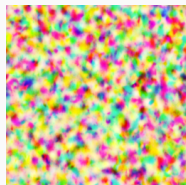
## Independent synthesis in RGB color space:



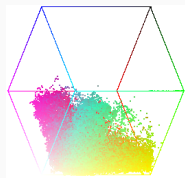
Input color cube



Input texture

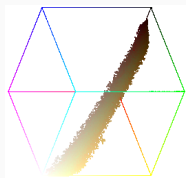


Output texture

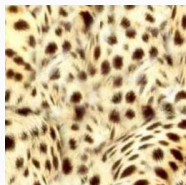


Output color cube

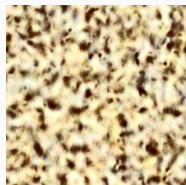
## Independent synthesis in PCA color space:



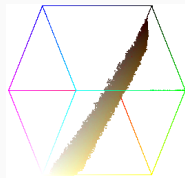
Input color cube



Input texture



Output texture



Output color cube

# Heeger and Bergen algorithm: Results





Several parameters:

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

# Heeger and Bergen algorithm: Parameters

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

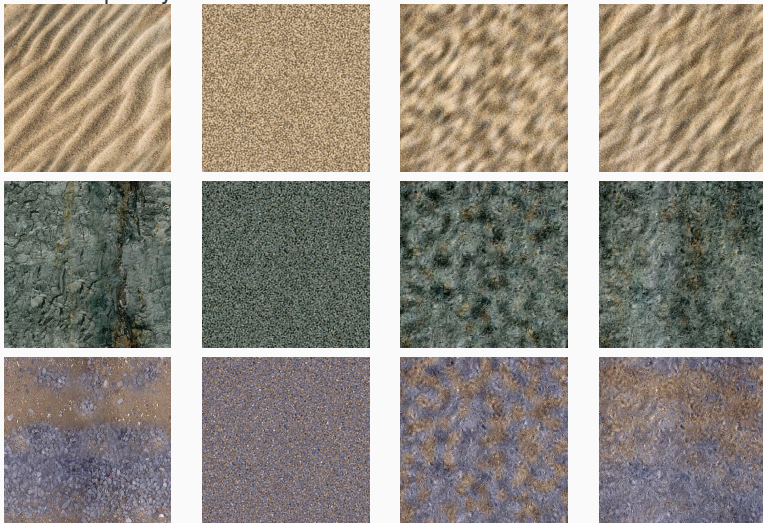
- Generally 5 iterations is enough for “visual convergence”



# Heeger and Bergen algorithm: Parameters

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

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



## References:

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

## A must read paper !


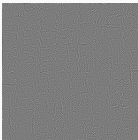
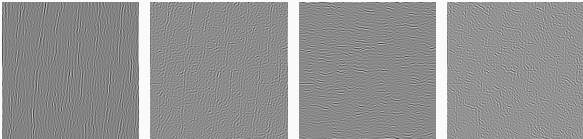
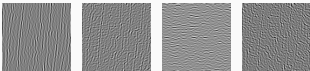

J. Vacher and T. Briand, *The Portilla-Simoncelli Texture Model: Towards the Understanding of the Early Visual Cortex*, **IPOL (with on-line demo)** (2021)

## General ideas:

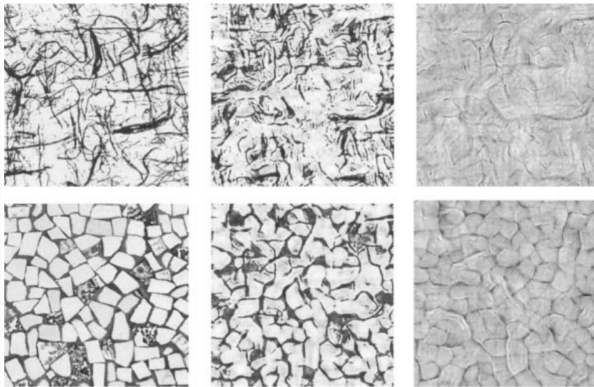
- In Heeger-Bergen the pyramid images are treated independently, but they are clearly highly correlated.
- Introduce joint statistics within neighborhood pixels, with neighborhood on spatial grid but also across scales.
- Select a small set of statistics (moments) and show that each of them is important.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

# Steerable pyramid decomposition of a texture

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

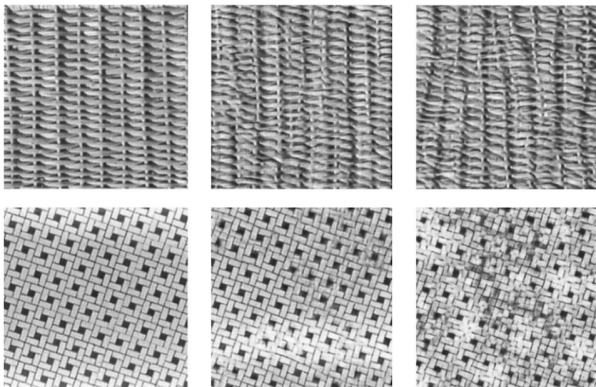
Original image	Associated steerable pyramid
	 High frequency residual  1st scale of oriented subbands with angle $\theta = 0, \frac{\pi}{4}, \frac{\pi}{2}, \text{ and } \frac{3\pi}{4}$  2nd scale of oriented subbands with angle $\theta = 0, \frac{\pi}{4}, \frac{\pi}{2}, \text{ and } \frac{3\pi}{4}$ 

From (Portilla and Simoncelli, 2000).



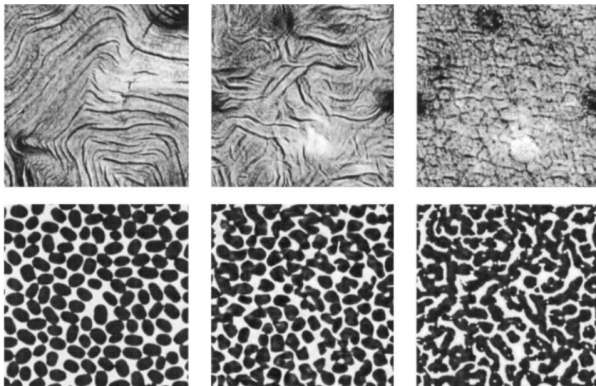
*Figure 3.* Necessity of marginal constraints. Left column: original texture images. Middle: Images synthesized using full constraint set. Right: Images synthesized using all but the marginal constraints.

From (Portilla and Simoncelli, 2000).



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

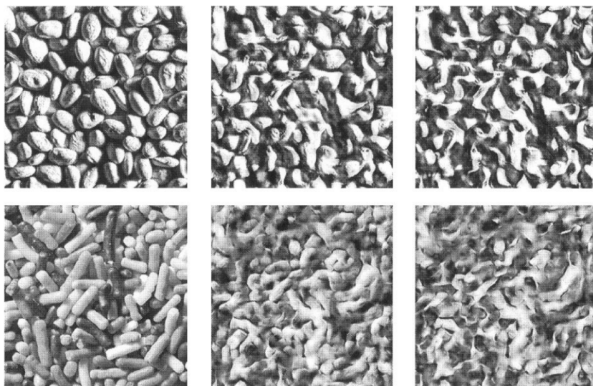
From (Portilla and Simoncelli, 2000).



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



From (Portilla and Simoncelli, 2000).

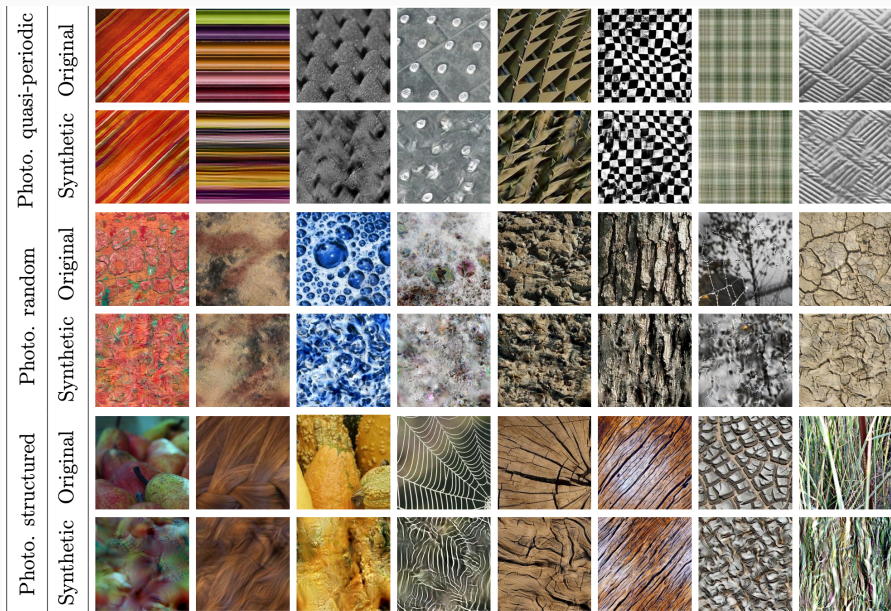


*Figure 8.* Necessity of cross-scale phase constraints. Left column: original texture images. Middle: Images synthesized using full constraint set. Right: Images synthesized using all but the cross-scale phase constraints.

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

- Very impressive results.
  - Stayed state of the art for statistical texture synthesis for a long time.
  - Patch-based methods were introduced at the same period.
  - Here the textures are synthesized with only 710 parameters.
- 
- Extended to color textures in a Matlab code (no associated publication).
  - Reproduced by J. Vacher and T. Briand (IPOL paper) ([Vacher and Briand, 2021](#))

# Portilla and Simoncelli: Color results



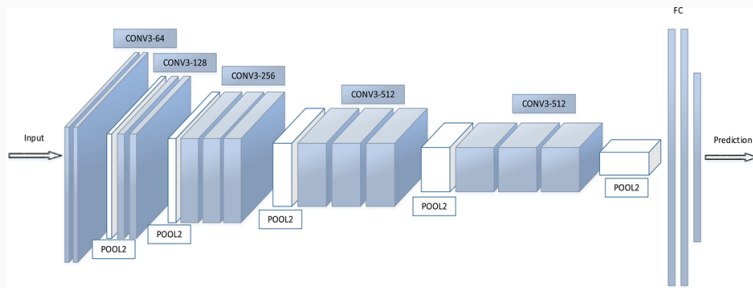
## **Textures synthesis using CNN statistics**

---

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

# Convolutional Neural Networks (CNN)

- Main idea: Use the feature layers of a trained deep CNN, namely VGG 19 (Simonyan and Zisserman, 2015), as statistics.
- VGG 19 was trained for image classification.
- It only uses  $3 \times 3$  convolution kernels followed by RELU (= positive part) and max-pooling.



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

Given an example image  $u$  and a random initialization  $x_0$ , one optimizes the loss function

$$E(x) = \sum_{\text{for selected layers } L} w_L \|G^L(x) - G^L(u)\|_F^2$$

where

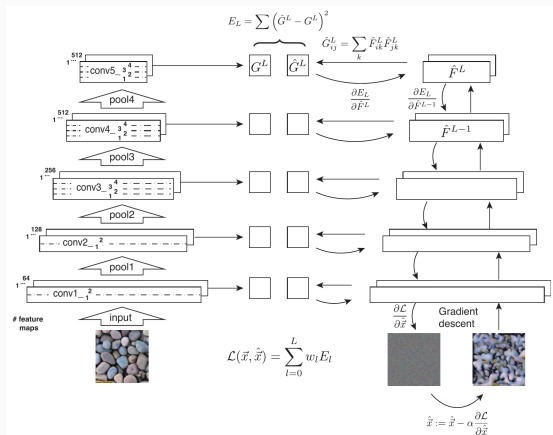
- $w_L$  is a weight parameter for each layer
- $\|\cdot\|_F$  is the Frobenius norm
- for an image  $y$  and a layer index  $L$ ,  $G^L(y)$  denotes the **Gram matrix** of the VGG-19 features at layer  $L$ : if  $V^L(y)$  is the feature response of  $y$  at layer  $L$  that has spatial size  $w \times h$  and  $n$  channels,

$$G^L(y) = \frac{1}{wh} \sum_{k \in \{0, \dots, w-1\} \times \{0, \dots, h-1\}} V^L(y)_k V^L(y)_k^T \in \mathbb{R}^{n \times n}.$$

The Gram matrix is a spatial statistics of order 2 that contains mean and covariance information.



# Gatys et al algorithm



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

Given an example image  $u$  and a random initialization  $x_0$ , one optimizes the loss function

$$E(x) = \sum_{\text{for selected layers } L} w_L \|G^L(x) - G^L(u)\|_F^2$$

## Pseudo-code:

**Require:** Input image  $u$ , set of selected VGG-19 layers  $\mathcal{L}$  and associated layer weights parameter  $\{w_L, L \in \mathcal{L}\}$

**Ensure:** Synthesized texture  $x$

Apply VGG-19 to  $u$  and extract the layers  $\{V_L(u), L \in \mathcal{L}\}$

Compute the target Gram matrices  $\{G_L(u) = \text{Gram}(V_L(u)), L \in \mathcal{L}\}$ .

Initialize  $x$  with some Gaussian white noise.

**for**  $N_{\text{it}}$  iterations **do**

    Apply VGG-19 to  $x$  and extract the layers  $\{V_L(x), L \in \mathcal{L}\}$ .

    Compute the current Gram matrices of  $x$ :  $\{G_L(x) = \text{Gram}(V_L(x)), L \in \mathcal{L}\}$

    Compute the loss  $E(x)$  and its gradient  $\nabla_x E(x)$  using backpropagation.

$x \leftarrow \text{L-BFGS-step}(x, E(x), \nabla_x E(x))$ .

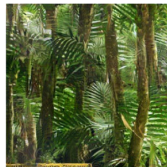
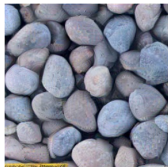
**end for**

# Gatys et al algorithm: Depth influence

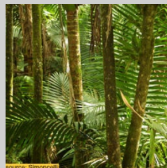
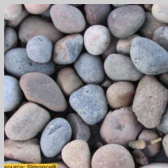


# Gatys et al algorithm: Results

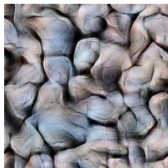
pool4



original

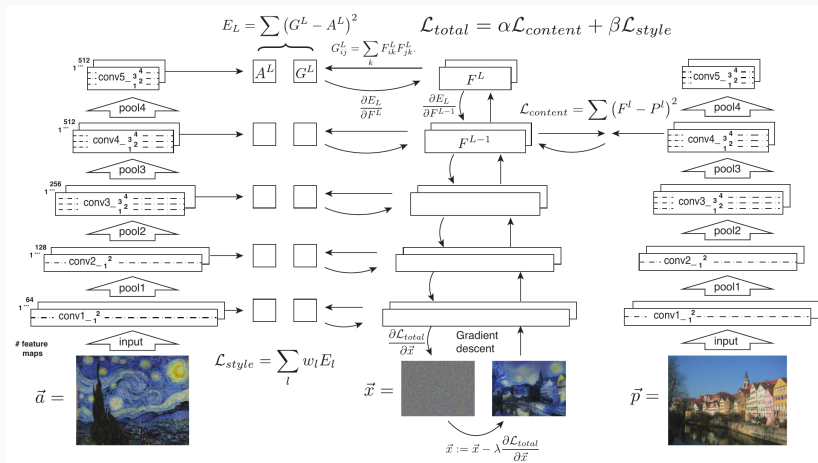


Portilla & Simoncelli



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

Reference: (Gatys et al., 2016)



# Gatys et al for style transfer

Reference: (Gatys et al., 2016)

A



B



C



D

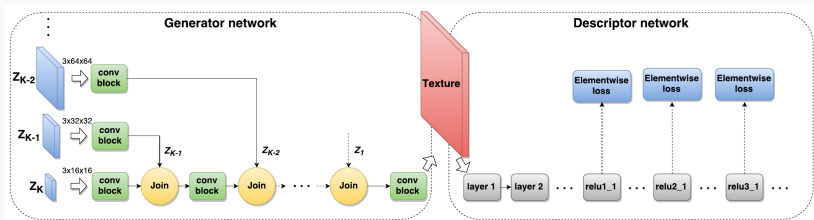


- Very nice and clean PyTorch implementation:  
`https://github.com/leongatys/PytorchNeuralStyleTransfer`
- Today: Practice session based in this code.
- **Very slow** on CPU and computationally demanding with high-end GPU (and memory consuming, e.g. 8 GB of memory for a  $1024 \times 1024$  image). See practice session.
- Regarding texture modeling, the **number of parameters is huge**: Textures are described by the Gram matrices and the number of elements in the Gram matrices totals 850k. That is 1000 times more than Portilla-Simoncelli !



# Generative networks for texture synthesis

- A workaround for speeding up synthesis is to train generative forward networks to mimic Gatys algorithm, as proposed by (Ulyanov et al., 2016) (coined Texture Networks).
- This is **an example of generative network**.
- The generator is trained to produce images with low Gatys loss (self-supervised training) from multi-scale white noise.
- Synthesis is fast thanks to the feedforward architecture.



# Generative networks for texture synthesis

## Texture Networks

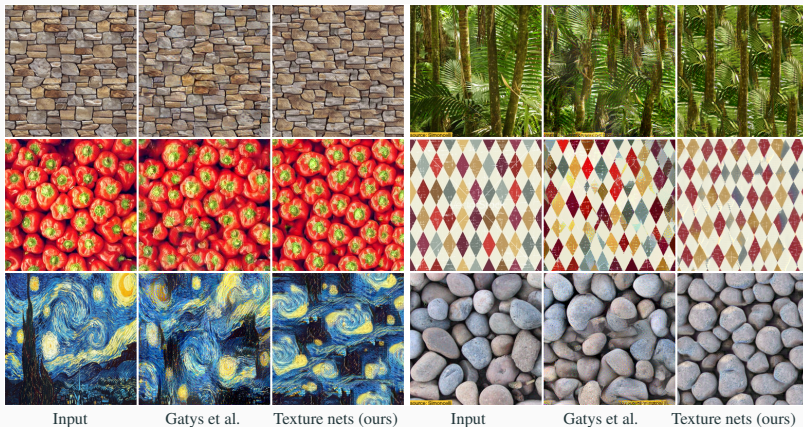
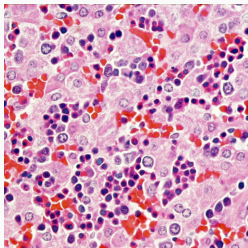


Figure 1. Texture networks proposed in this work are feed-forward architectures capable of learning to synthesize complex textures based on a single training example. The perceptual quality of the feed-forwardly generated textures is similar to the results of the closely related method suggested in (Gatys et al., 2015a), which use slow optimization process.

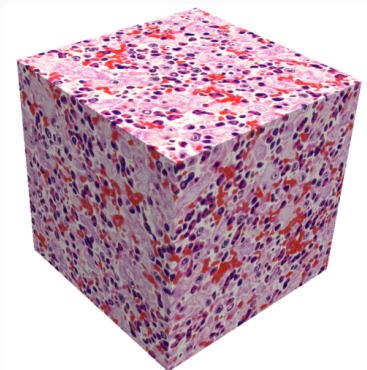
- However texture quality is not as good, and a network has to be trained for each new image.

# On demand solid texture synthesis using deep 3D networks

- Texture networks can be extended to 3D (Gutierrez et al., 2020) while the Gatys approach is infeasible.

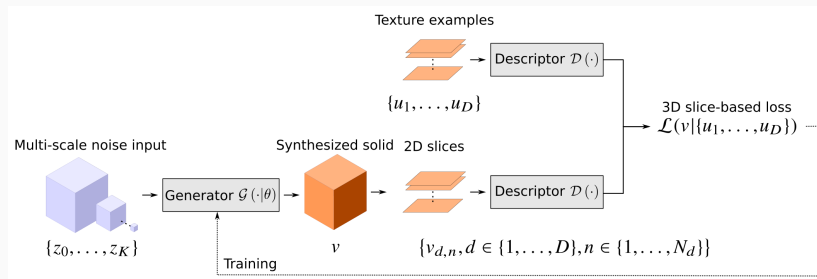


Input



Output

## Training framework for the proposed CNN Generator network:



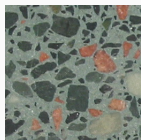
- The generator  $\mathcal{G}(\cdot|\theta)$  with parameters  $\theta$  processes a multi-scale noise input  $Z$  to produce a solid texture  $v$
- The loss  $\mathcal{L}$  compares, for each direction  $d$ , the feature statistics induced by the example  $u_d$  in the layers of the pre-trained Descriptor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



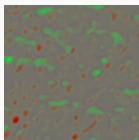
# On demand solid texture synthesis using deep 3D networks

Training: Find the parameters  $\theta$  for a given texture

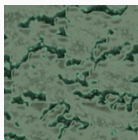
- Exploit invariance by translation to generate batches of width one voxel only (“single-slice training scheme”)
- Minimize Gatys’ loss for each slice, using 3000 iterations of Adam algorithm



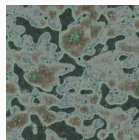
input



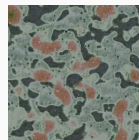
10



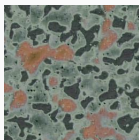
20



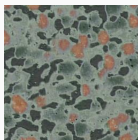
50



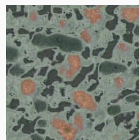
100



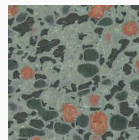
200



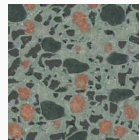
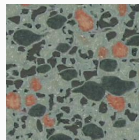
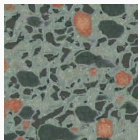
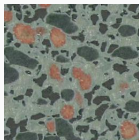
300



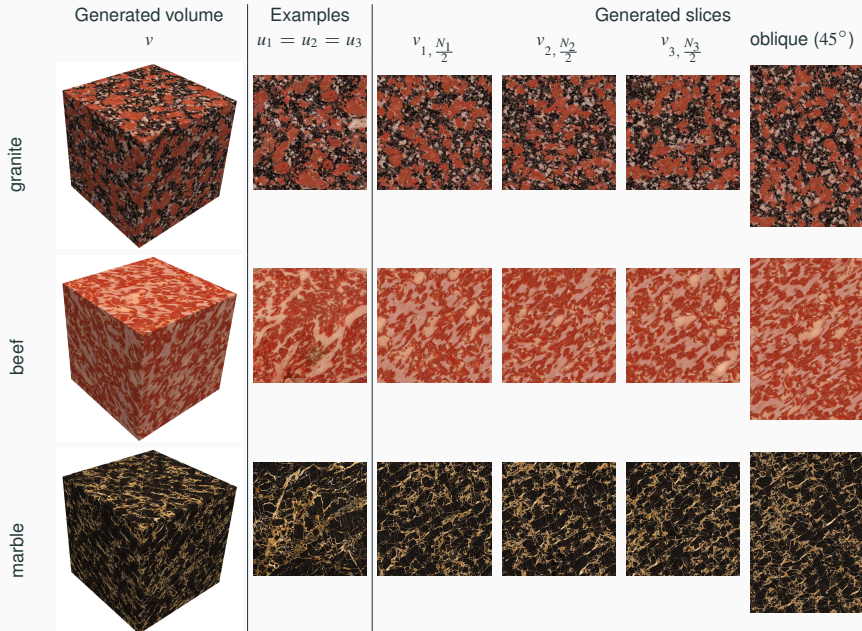
500



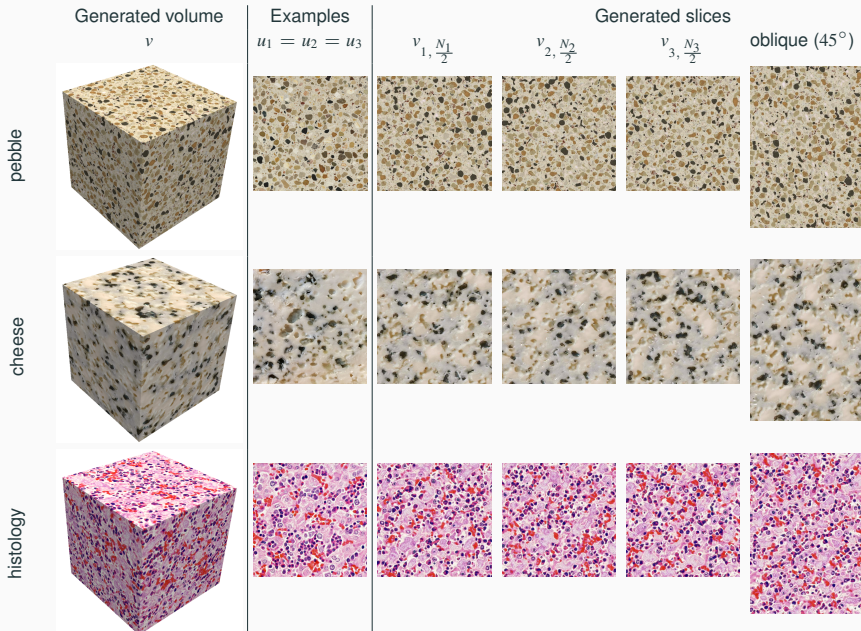
1000



# On demand solid texture synthesis using deep 3D networks



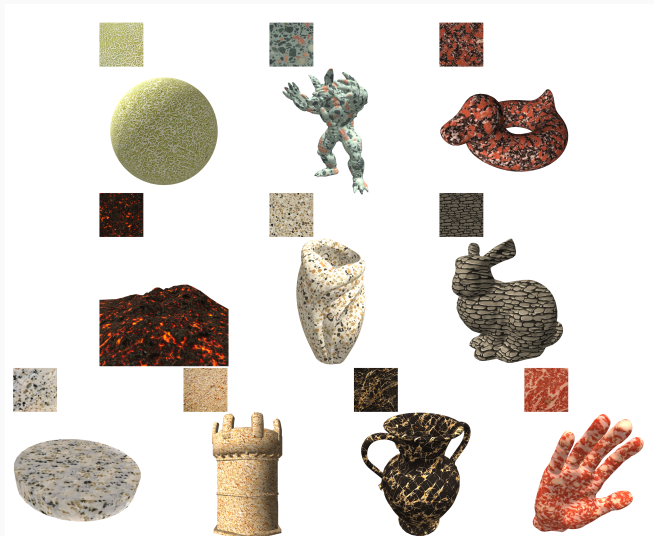
# On demand solid texture synthesis using deep 3D networks





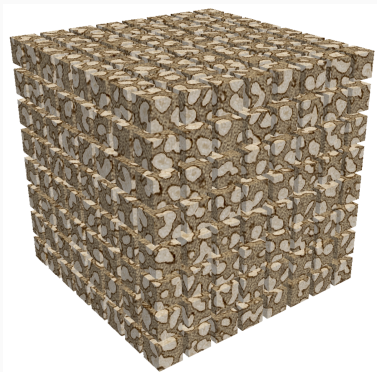
# On demand solid texture synthesis using deep 3D networks

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



# On demand solid texture synthesis using deep 3D networks

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



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

- Other contributions propose generative networks for **universal** style transfer/texture synthesis (e.g. (Li et al., 2017)).
- **Universal** means that a single network is adapted to all images.
- This still relies in approximating the Gatys procedure with some approximation for a faster style transfer: auto-encoders to invert VGG19 and imposing Gram matrices.
- Using Wasserstein distance between Gaussians in VGG19 feature space is efficient for texture mixing (Vacher et al., 2020) (see practice session).

## **Discussion on variational texture synthesis**

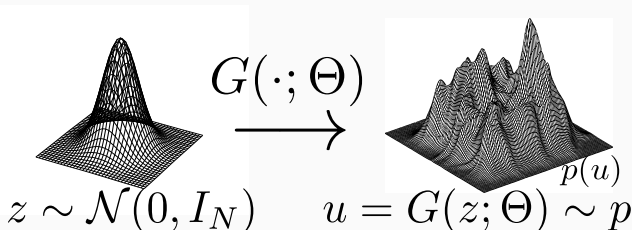
---

# Lack of probabilistic modeling

So far the discussed algorithms uses the following methodology.

1. Define statistics of interest  $F(x_0)$
2. Define a (non convex) energy  $E$  that represents the distance of an image  $x$  to the ones of the input  $x_0$ , e.g.  $E = \|F(x) - F(x_0)\|^2$ .
3. Minimize  $E$  with some descent algorithm **starting from a white noise image**

Several observations:



- Here the generator is an optimization algorithm.

# Lack of probabilistic modeling

So far the discussed algorithms uses the following methodology.

1. Define statistics of interest  $F(x_0)$
2. Define a (non convex) energy  $E$  that represents the distance of an image  $x$  to the ones of the input  $x_0$ , e.g.  $E = \|F(x) - F(x_0)\|^2$ .
3. Minimize  $E$  with some descent algorithm **starting from a white noise image**

Several observations:

Regarding optimization:

- One always have  $E(\text{input}) = 0$  that is the input is a global minimum.
- But for texture synthesis we do not want the output to be equal to the input !
- Things are OK in practice since we minimize a non convex energy starting from a white noise that is very far from the input.

# Lack of probabilistic modeling

So far the discussed algorithms uses the following methodology.

1. Define statistics of interest  $F(x_0)$
2. Define a (non convex) energy  $E$  that represents the distance of an image  $x$  to the ones of the input  $x_0$ , e.g.  $E = \|F(x) - F(x_0)\|^2$ .
3. Minimize  $E$  with some descent algorithm **starting from a white noise image**

## Several observations:

### Regarding probabilistic modeling:

- There is no randomness except for the initial white noise.
- By minimizing  $E$  one tries to correct the excess of disorder of the initial white noise image.
- There is no random field model, but the probable states are the local minima of the energy  $E$ , at least the ones one can reach starting from a white noise image... Nothing can be said about the implicit probability distribution, i.e. how the local minima are charged.

## Practice session:

Today we use the website:

<https://storimaging.github.io/notebooksImageGeneration/>

Texture synthesis using CNN statistics (link) (Gatys et al., 2015)



## References

---

- Briand, T., Vacher, J., Galerne, B., and Rabin, J. (2014). The heeger & bergsen pyramid based texture synthesis algorithm. *Image Processing On Line*, 4:276–299.
- Chung, H., Kim, J., McCann, M. T., Klasky, M. L., and Ye, J. C. (2023). Diffusion posterior sampling for general noisy inverse problems. In *The Eleventh International Conference on Learning Representations*.
- Galerne, B., Gousseau, Y., and Morel, J.-M. (2011). Random phase textures: Theory and synthesis. *IEEE Trans. Image Process.*, 20(1):257 – 267.
- Gatys, L., Ecker, A. S., and Bethge, M. (2015). Texture synthesis using convolutional neural networks. In *Advances in Neural Information Processing Systems 28*, pages 262 – 270.
- Gatys, L. A., Ecker, A. S., and Bethge, M. (2016). Image style transfer using convolutional neural networks. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2414–2423.

- Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N., and Weinberger, K., editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc.
- Gutierrez, J., Rabin, J., Galerne, B., and Hurtut, T. (2020). On demand solid texture synthesis using deep 3d networks. *Computer Graphics Forum*, 39(1):511–530.
- Ho, J., Jain, A., and Abbeel, P. (2020). Denoising diffusion probabilistic models. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33, pages 6840–6851. Curran Associates, Inc.
- Isola, P., Zhu, J.-Y., Zhou, T., and Efros, A. A. (2017). Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

- Karras, T., Laine, S., and Aila, T. (2019). A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.
- Li, Y., Fang, C., Yang, J., Wang, Z., Lu, X., and Yang, M.-H. (2017). Universal style transfer via feature transforms. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.
- Lisani, J.-L., Buades, A., and Morel, J.-M. (2011). Image Color Cube Dimensional Filtering and Visualization. *Image Processing On Line*, 1:57–69. <https://doi.org/10.5201/ipol.2011.blm-cdf>.
- Oppenheim, A. V. and Lim, J. S. (1981). The importance of phase in signals. In *Proceedings of the IEEE*, volume 69, pages 529–541.

- Phongthawee, P., Chinchuthakun, W., Sinsunthithet, N., Jampani, V., Raj, A., Khungurn, P., and Suwajanakorn, S. (2024). Diffusionlight: Light probes for free by painting a chrome ball. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 98–108.
- Portilla, J. and Simoncelli, E. P. (2000). A parametric texture model based on joint statistics of complex wavelet coefficients. *Int. J. Comp. Vis.*, 40(1):49–71.
- Rombach, R., Blattmann, A., Lorenz, D., Esser, P., and Ommer, B. (2022). High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10684–10695.
- Ruiz, N., Li, Y., Jampani, V., Pritch, Y., Rubinstein, M., and Aberman, K. (2023). Dreambooth: Fine tuning text-to-image diffusion models for subject-driven generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 22500–22510.

- Saharia, C., Chan, W., Saxena, S., Li, L., Whang, J., Denton, E., Ghasemipour, S. K. S., Ayan, B. K., Mahdavi, S. S., Lopes, R. G., Salimans, T., Ho, J., Fleet, D. J., and Norouzi, M. (2022). Photorealistic text-to-image diffusion models with deep language understanding.
- Simoncelli, E., Freeman, W., Adelson, E., and Heeger, D. (1992). Shiftable multiscale transforms. *IEEE Transactions on Information Theory*, 38(2):587–607.
- Simonyan, K. and Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. In Bengio, Y. and LeCun, Y., editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.
- Sohl-Dickstein, J., Weiss, E., Maheswaranathan, N., and Ganguli, S. (2015). Deep unsupervised learning using nonequilibrium thermodynamics. In Bach, F. and Blei, D., editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 2256–2265, Lille, France. PMLR.

- Song, J., Vahdat, A., Mardani, M., and Kautz, J. (2023). Pseudoinverse-guided diffusion models for inverse problems. In *International Conference on Learning Representations*.
- Ulyanov, D., Lebedev, V., Vedaldi, A., and Lempitsky, V. (2016). Texture networks: Feed-forward synthesis of textures and stylized images. In *ICML*, pages 1349–1357.
- Vacher, J. and Briand, T. (2021). The Portilla-Simoncelli Texture Model: towards Understanding the Early Visual Cortex. *Image Processing On Line*, 11:170–211. <https://doi.org/10.5201/ipol.2021.324>.
- Vacher, J., Davila, A., Kohn, A., and Coen-Cagli, R. (2020). Texture interpolation for probing visual perception. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33, pages 22146–22157. Curran Associates, Inc.

- 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.
- Wang, X., Zhang, B., and Yang, H. (2014). Content-based image retrieval by integrating color and texture features. *Multim. Tools Appl.*, 68(3):545–569.
- 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.
- Zhang, L., Rao, A., and Agrawala, M. (2023). Adding conditional control to text-to-image diffusion models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 3836–3847.