Published in Image Processing On Line on 2011–09–23.
Submitted on 2011–00–00, accepted on 2011–00–00.
ISSN 2105–1232 © 2011 IPOL & the authors CC–BY–NC–SA
This article is available online with supplementary materials,
software, datasets and online demo at
http://dx.doi.org/10.5201/ipol.2011.ggm_rpn

Micro-Texture Synthesis by Phase Randomization

Bruno Galerne[1],Yann Gousseau[2], Jean-Michel Morel[3]

[1] CMLA, ENS Cachan, CNRS, UniverSud, France (galerne@cmla.ens-cachan.fr)
[2] LTCI CNRS, Télécom ParisTech, France (gousseau@enst.fr)
[3] CMLA, ENS Cachan, CNRS, UniverSud, France (morel@cmla.ens-cachan.fr)

Abstract

This contribution is concerned with texture synthesis by example, the process of generating new texture images from a given sample. The Random Phase Noise algorithm presented here synthesizes a texture from an original image by simply randomizing its Fourier phase. It is able to reproduce textures which are characterized by their Fourier modulus, namely the random phase textures (or micro-textures).

Source Code

The source code (ANSI C), its documentation, and the online demo are accessible at the IPOL web page of this article[1].

Keywords: texture; synthesis; random Fourier phase

1 Introduction

1.1 Context

This contribution is concerned with *texture synthesis by example*, the process of generating new texture images from a given sample. This field has witnessed an important number of key contributions over the last 15 years, see e.g. [2], [3], [4], [5]. In [2], the texture is reconstructed by a Markov Random field model directly inspired from Shannon's famous English sentence reconstruction. Each incomplete patch of the image under construction is compared to all patches of the sample image. The closest patches in the sample permit to predict a new pixel value in the synthetic image. The iterative algorithm grows in that way a new image from a random seed, or even from the sample itself by extending its boundaries. The method of [3] is a significant technical improvement of the same basic ideas, where the shape of the incomplete patch is no more variable. These patch-based algorithms suffer, however, from several drawbacks pointed out by the authors themselves: the resulting image is sometimes strikingly good, but sometimes also its statistics do not match those of

[1] http://dx.doi.org/10.5201/ipol.2011.ggm_rpn

the input and the algorithm may even diverge toward poorly structured results. Unwanted periodicities can also occur, the algorithm having a strong tendency to produce verbatim copy of the input. The algorithms [4] and [5], inspired from models of human texture perception (e.g. [6]) and from wavelet theory, learn the wavelet coefficients statistics from the sample and enforce them in the reconstructed image. In [4] this reconstruction deals only with wavelet coefficient histograms while in [5] the cross-correlation and autocorrelation statistics of the wavelet channels are also enforced. The final results, particularly those of [5], are absolutely striking. The process is nonetheless quite complex, its convergence not guaranteed, and color artifacts may also appear. In short, in spite of considerable progress, no present algorithm delivers a full reproduction of all textures, and it is still important to explore other ways, well adapted to this or that kind of texture.

1.2 Contribution

The *Random Phase Noise* (*RPN*) algorithm presented here, and first introduced in [1], synthesizes a texture from an original image by simply randomizing its Fourier phase. It is able to reproduce textures which are characterized by their Fourier modulus, namely the random phase textures (or micro-textures). It is also able to create a random texture from any input image. It is in spirit quite close to the *noise* generators from computer graphics, see e.g. [7] and [8].

The presented algorithm deals with color images. It is able to synthesize output textures with arbitrary sizes. The algorithm is definitely limited to a certain class of textures, but it has the following good properties, that are particularly useful for graphic applications and are not shared by, e.g., exemplar-based methods.

- It is fast, since it basically only needs the computation of two FFTs.
- It is perceptually stable: all the textures synthesized from the same input image are visually similar. In particular, it will never *grow garbage*.
- It will never yield verbatim copy of the original.
- It has no convergence issues.

2 Algorithm

2.1 Basic RPN

By definition, the *RPN* of an image h is the random image obtained by adding a random phase θ to the Fourier phase of the image. By a random phase we mean a white noise image uniformly distributed over $[-\pi, \pi]$ and that is constrained to be symmetric. Then the basic *RPN* algorithm consists in:

1. Computing a realization θ of a random phase.
2. Computing the discrete Fourier transform (DFT) $\hat{h}$ of h.
3. Adding the random phase
$$\hat{h}(\xi_1, \xi_2) \leftarrow \hat{h}(\xi_1, \xi_2) e^{i\theta(\xi_1, \xi_2)}.$$
4. Returning the inverse DFT of $\hat{h}$.

2.2 Extension to Color Images

The *RPN* of an RGB color image $h = (h_R, h_G, h_B)$ is obtained by **adding the same random phase** to the DFT of each color channel. More precisely, step 3 of the above *basic RPN* algorithm is replaced by

$$\left(\hat{h}_R(\xi_1,\xi_2), \hat{h}_G(\xi_1,\xi_2), \hat{h}_B(\xi_1,\xi_2)\right) \leftarrow \left(\hat{h}_R(\xi_1,\xi_2), \hat{h}_G(\xi_1,\xi_2), \hat{h}_B(\xi_1,\xi_2)\right) e^{i\theta(\xi_1,\xi_2)}.$$

Adding the same random phase to the original phases of each color channel preserves the phase displacements between channels. This is important as it permits to create new textures without creating false colors [1].

2.3 Avoiding Artifacts Due to Non Periodicity

The *RPN* algorithm is based on the FFT and consequently the periodicity of the input image is a critical requirement. Indeed, as experiments show, randomizing the phase of an image which is not periodic creates strong artifacts consisting in highly contrasted horizontal and vertical waves. To avoid these artifacts the input image h is replaced by its periodic component p as defined by L. Moisan in [9].

The periodic component p is the unique solution of the problem

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

where Δ is the usual discrete periodic Laplacian (each point has four neighbors) and Δ_i is the discrete Laplacian in the interior of the image domain (points at the border of the image have only three or two neighbors).

The periodic component p of h is computed using the classic FFT-based Poisson solver:

1. Compute the discrete Laplacian $\Delta_i h$ of h.
2. Compute the DFT $\widehat{\Delta_i h}$ of $\Delta_i h$.
3. Compute the DFT $\hat{p}$ of p by inversing the discrete periodic Laplacian:

$$\begin{cases} \hat{p}(\xi_1,\xi_2) &= \left(4 - 2\cos\left(\frac{2\xi_1\pi}{M}\right) - 2\cos\left(\frac{2\xi_2\pi}{N}\right)\right)^{-1} \widehat{\Delta_i h}(\xi_1,\xi_2), \\ \hat{p}(0) &= \sum_{(x,y)} h(x,y). \end{cases}$$

4. Compute p by inverse DFT.

Note that this procedure permits to compute the DFT $\hat{p}$ of p using only one call to the FFT algorithm. Let us also precise that computing the periodic component of a color image simply consists in computing the periodic component of each color channel.

2.4 Spot Extension Technique

An important issue in texture synthesis is to synthesize textures with arbitrary large size from a given sample.

A practical method solves this problem for *RPN* textures. This is done by extending the original texture sample into an equivalent spot having a larger size, and then by applying the *basic RPN* algorithm to this larger equivalent spot, as illustrated in the example in Figure 1.

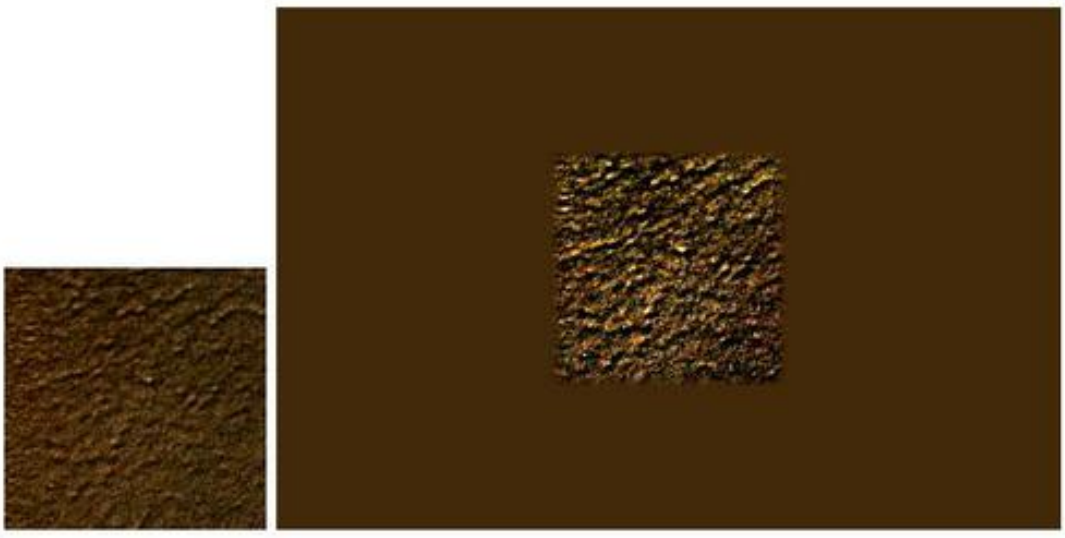


Figure 1: A texture image (left) and its corresponding extended spot (right).

Let us specify how the extended spot is computed. First, the periodic component of the original texture sample is computed and pasted in a large constant image equal to the mean of the sample, with a previous variance normalization. The obtained image is then multiplied by a smooth transition function in order to attenuate the discontinuities along the inner frame of the spot. On the interval $[0,1]$, the smooth transition function φ_α is defined as follows: On $[0,\alpha]$ the function varies as the primitive of the standard $\mathcal{C}^\infty$ function

$$t \mapsto \exp\left(-1/\left(1-(2t/\alpha-1)^2\right)\right),$$

it is symmetrically defined on the interval $[1-\alpha,1]$ and it is constant to 1 on the inner interval $[\alpha,1-\alpha]$. In Figure 2 are both a cross section and a gray-level representation of the smooth transition function used to attenuate the spot along the border of the image.

In addition, in order to preserve the variance of the initial spot, the smooth function is normalized so that its L^2-norm equals 1. The value of the parameter α is not a sensitive parameter. For all experiments as well as for the online demo the parameter α has been fixed to α=0.1. However α is an optional parameter of the downloadable source code.

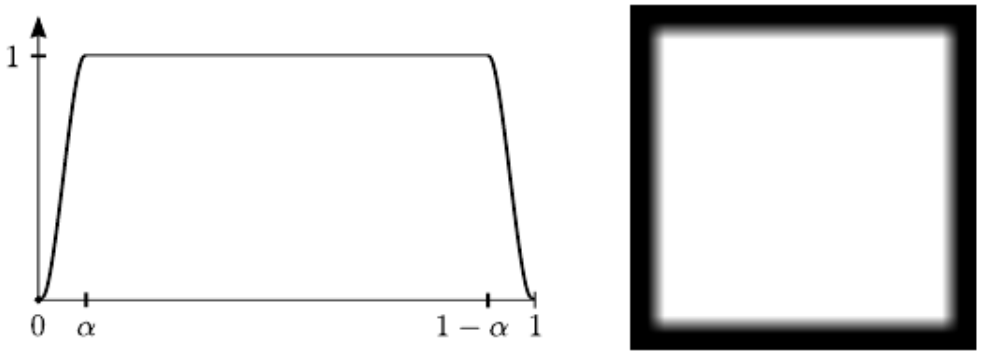


Figure 2: Smooth transition function used to attenuate the extended spot along the border of the image.

3 Implementation

Our implementation consists in two distinct procedures: same size *RPN* and increased size *RPN*.

3.1 Same Size *RPN*

The whole algorithm for synthesizing a *RPN* texture having the same size as the original image h consists in the following steps:

1. Compute the DFT $\hat{p}$ of the periodic component p of the input image h.
2. Compute a realization θ of a random phase.
3. Add the random phase θ to each color channel of $\hat{p}$.
4. Return the inverse DFT of the phase randomized $\hat{p}$.

Note that this algorithm only uses two calls to the FFT algorithm.

3.2 Increased Size *RPN*

1. Compute the periodic component p of the input image h.
2. Extend the periodic component p into an equivalent spot of larger size using the spot extension technique described above.
3. Compute a random phase θ having the same size as the extended spot.
4. Compute the *RPN* by adding the random phase θ to the DFT of each color channel.

This procedure is computationally more expensive than for the "same size *RPN*". Indeed, it makes four calls of the FFT algorithm: two with the size of the original spot h for the computation of the periodic component p, and two with the larger output size for the phase randomization.

4 Micro-Textures

4.1 What are Micro-Textures

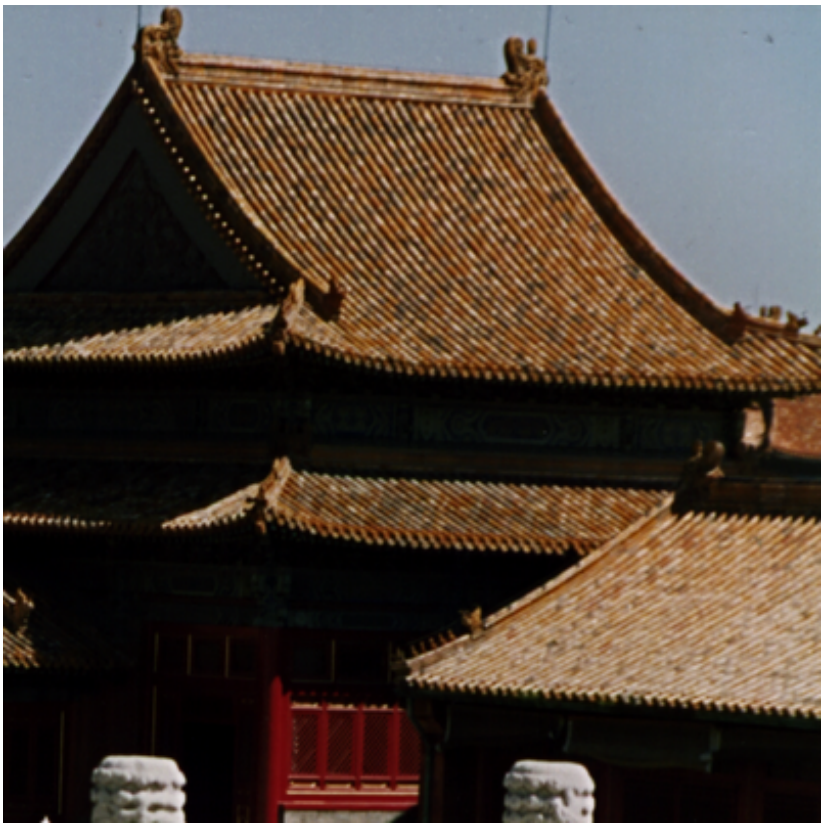
When photographed, remote homogeneous areas made of thin, small, or semitransparent objects create homogeneous regions in images. The geometric features and colors of the constituents of the observed area are mixed, due to the blur inherent to image formation. The resulting image region is a *micro-texture*. Most homogeneous regions in any image should be micro-textures. Figure 3 shows an example. Five rectangles belonging to various homogeneous regions were picked in a high resolution landscape. These textures are displayed in pairs where the left is the original sub-image, and the right a simulation obtained by the *RPN* algorithm. With the exception of the clouds rectangle (which is obviously non-stationary), these samples and their simulated copies are usually considered perceptually equivalent by observers. Here pebbles, wet sand, and various types of waves are correctly simulated.

Yet, not all homogeneous regions of an image are micro-textures. See many examples in Section 4.2, and the failure catalog (Section 5.1) as well.

Figure 3: Some emulated textures taken from a high resolution landscape. See text for details.

4.2 Micro-Textures and Macro-Textures

Many images or image parts usually termed textures do not fit to the micro-texture requisites. Typically, periodic patterns with big visible elements, such as brick walls, are not micro-textures. More generally, textures whose building elements are spatially organized, such as the branches of a tree, are not micro-textures. Yet, each textured object has a critical distance at which it becomes a micro-texture. For instance, tiles at a close distance are a *macro-texture*, and are not amenable to phase randomization. The smaller tiles on roofs photographed at some distance can instead be emulated (see Figure 4).



Original image

Samples RPN simulation

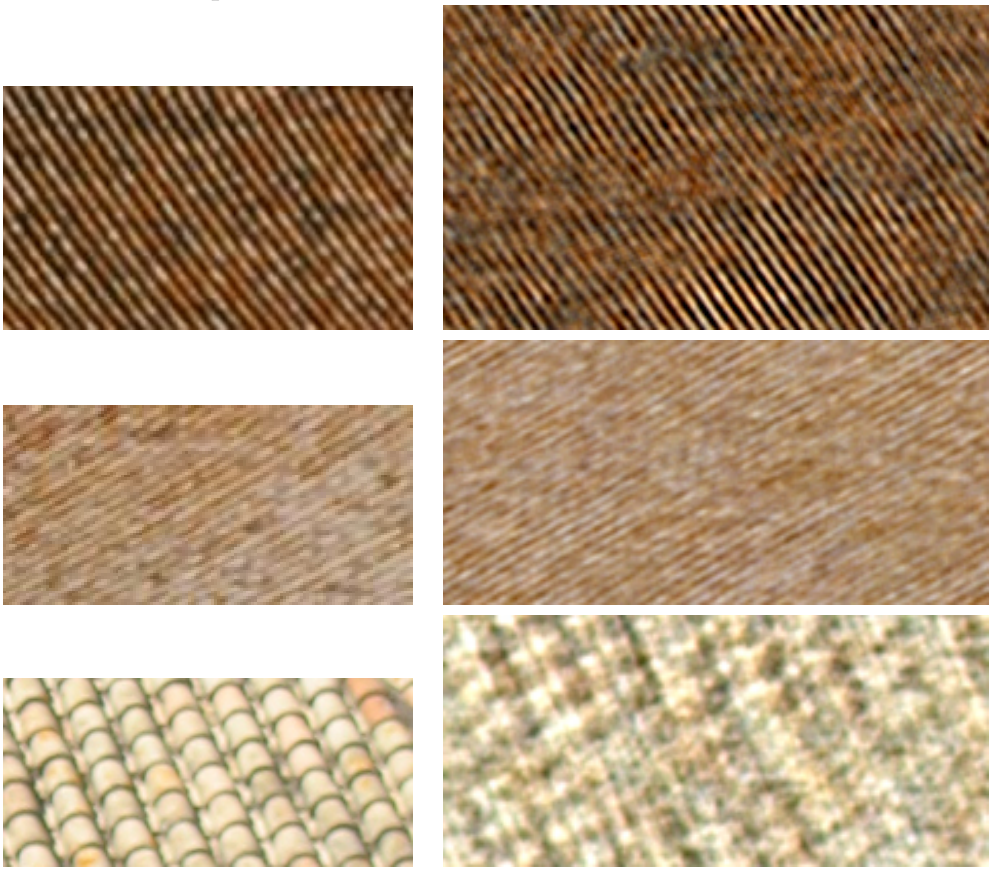


Figure 4: Examples of micro and macro textures. Top, Microtextures: tiled roofs. Bottom, left column: samples from original image; right colum: simulations with RPN. Remark that RPN fails to simulate the macro-texture in the last row (tiles at short range).

5 Examples

Figures 5 to 14 show some examples of satisfyingly well reproduced textures of wood, wall, fabric, and paint. See also the failure catalog (Section 5.1) for examples which are not well reproduced.

Figure 5: Wood examples. Samples (left) and RPN simulations (right). Wood samples must be homogeneous in direction to be correctly emulated by RPN. Wood samples with knots or other conspicuous patterns fall logically in the failure catalog (Section 5.1).

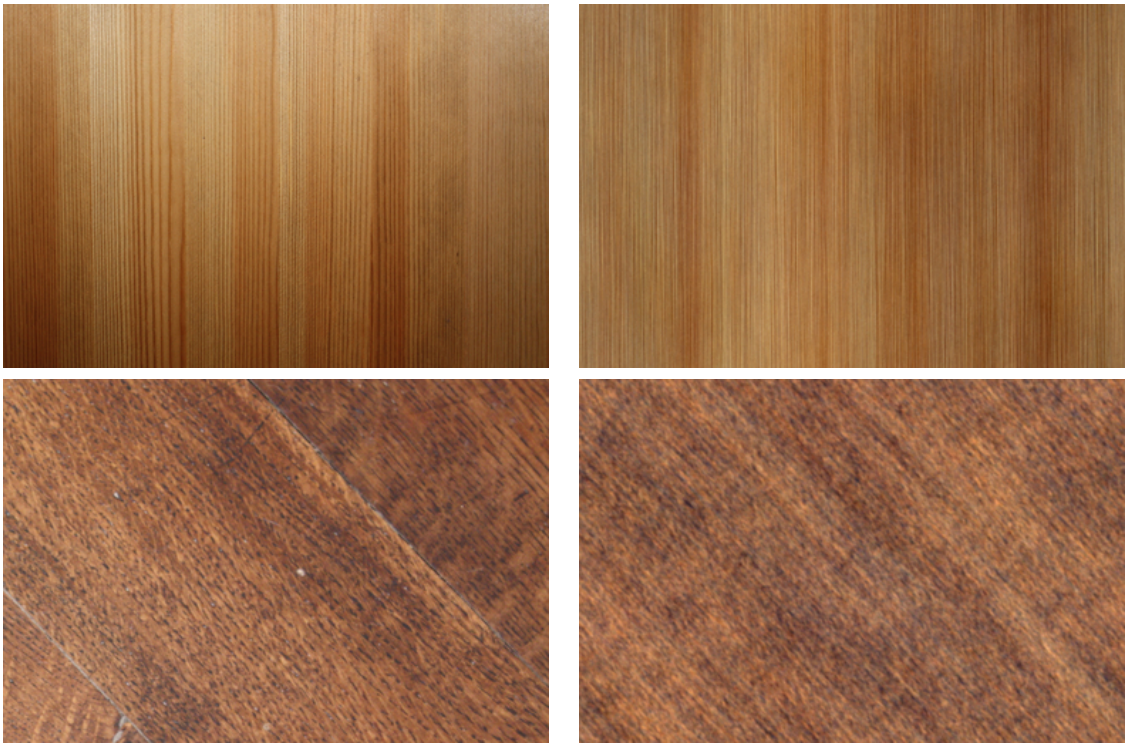


Figure 6: More wood examples. Samples (left) and RPN simulation (right).

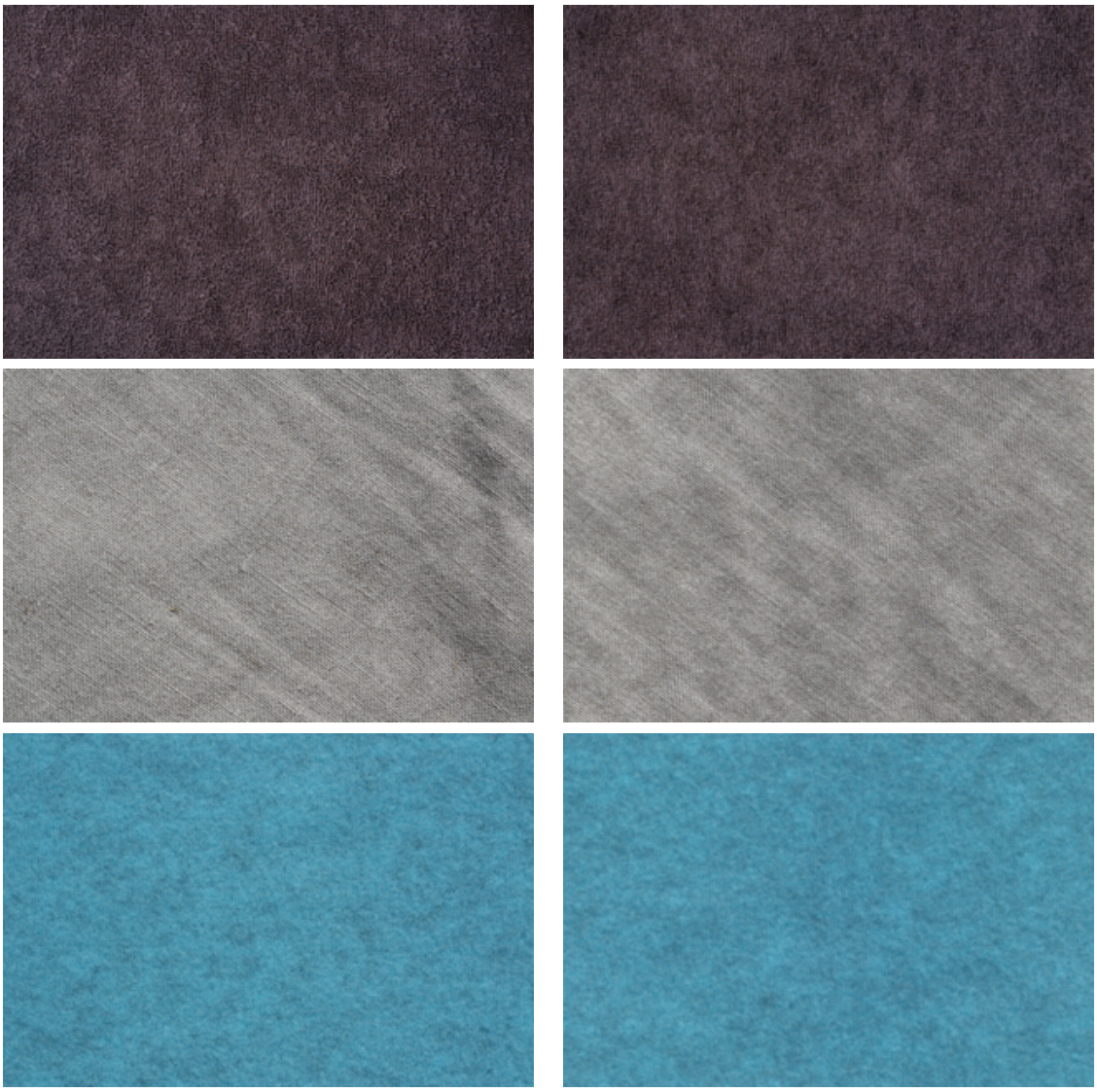


Figure 7: Fabric examples. Samples (left) and RPN simulations (right).

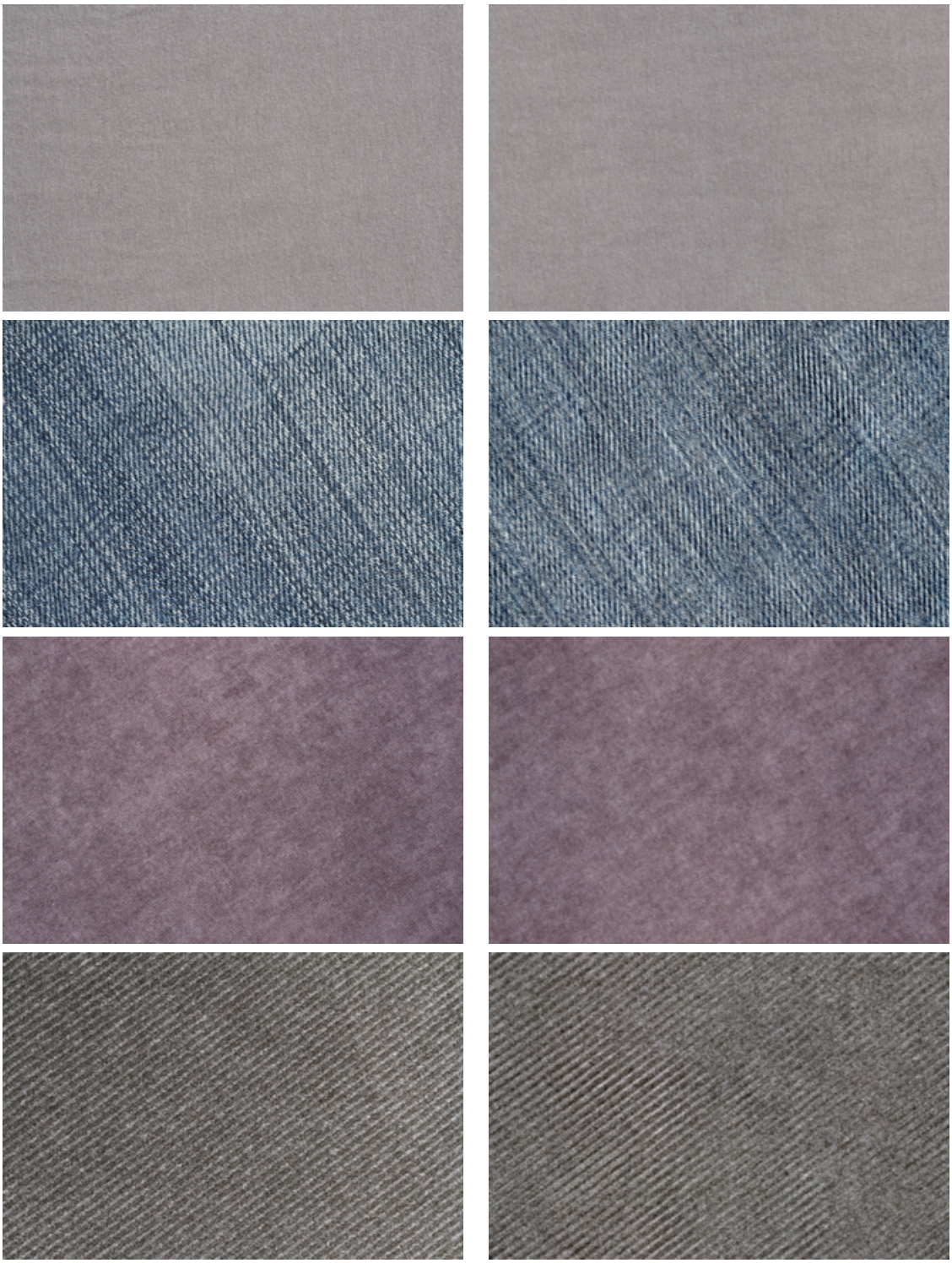


Figure 8: More fabric examples. Samples (left) and RPN simulations (right).

Figure 9: More fabric examples. Samples (left) and RPN simulations (right).

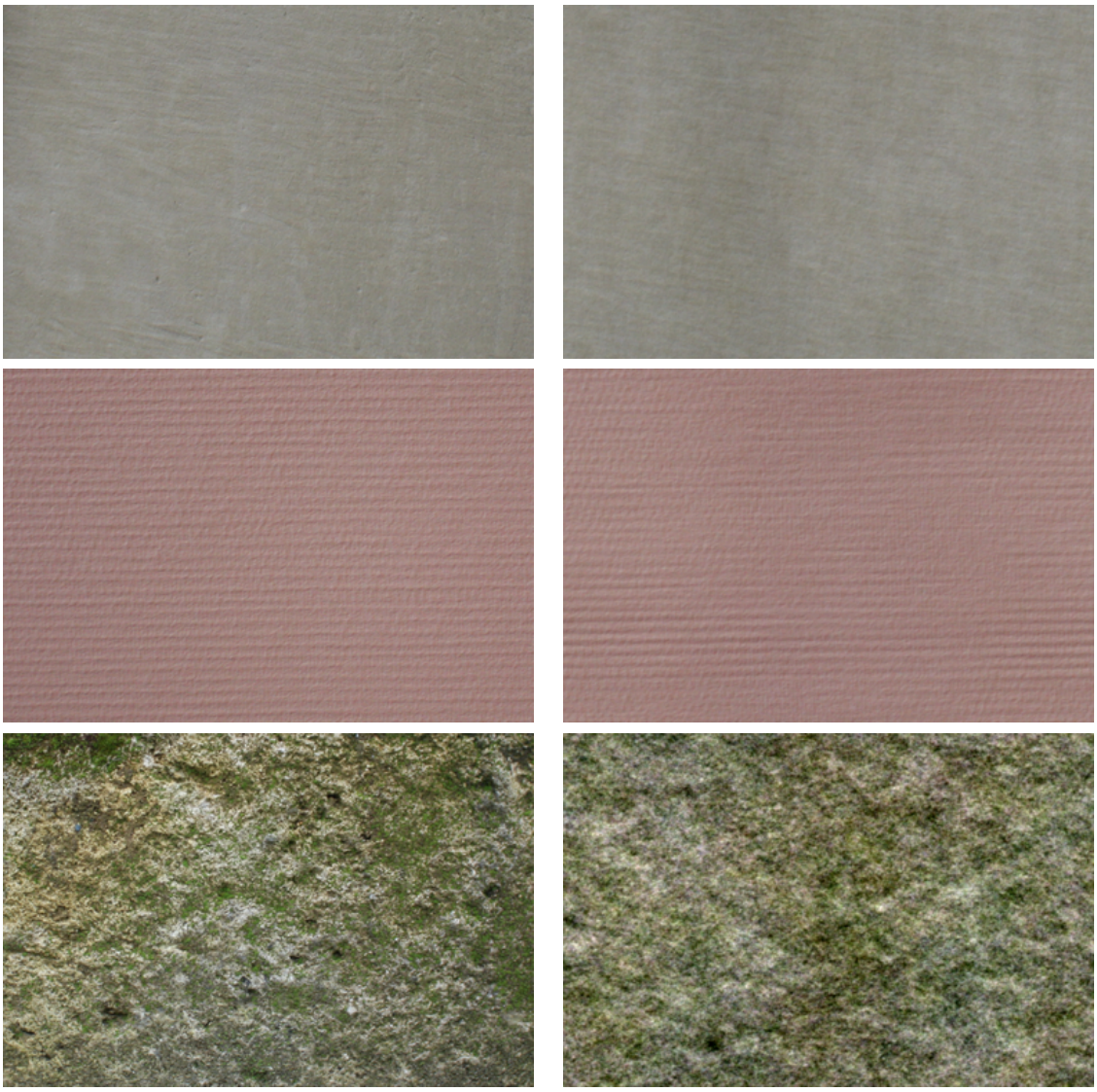


Figure 10: Wall examples. Samples (left) and RPN simulations (right).

Figure 11: More wall examples. Samples (left) and RPN simulations (right).

Figure 12: More wall examples. Samples (left) and RPN simulations (right).

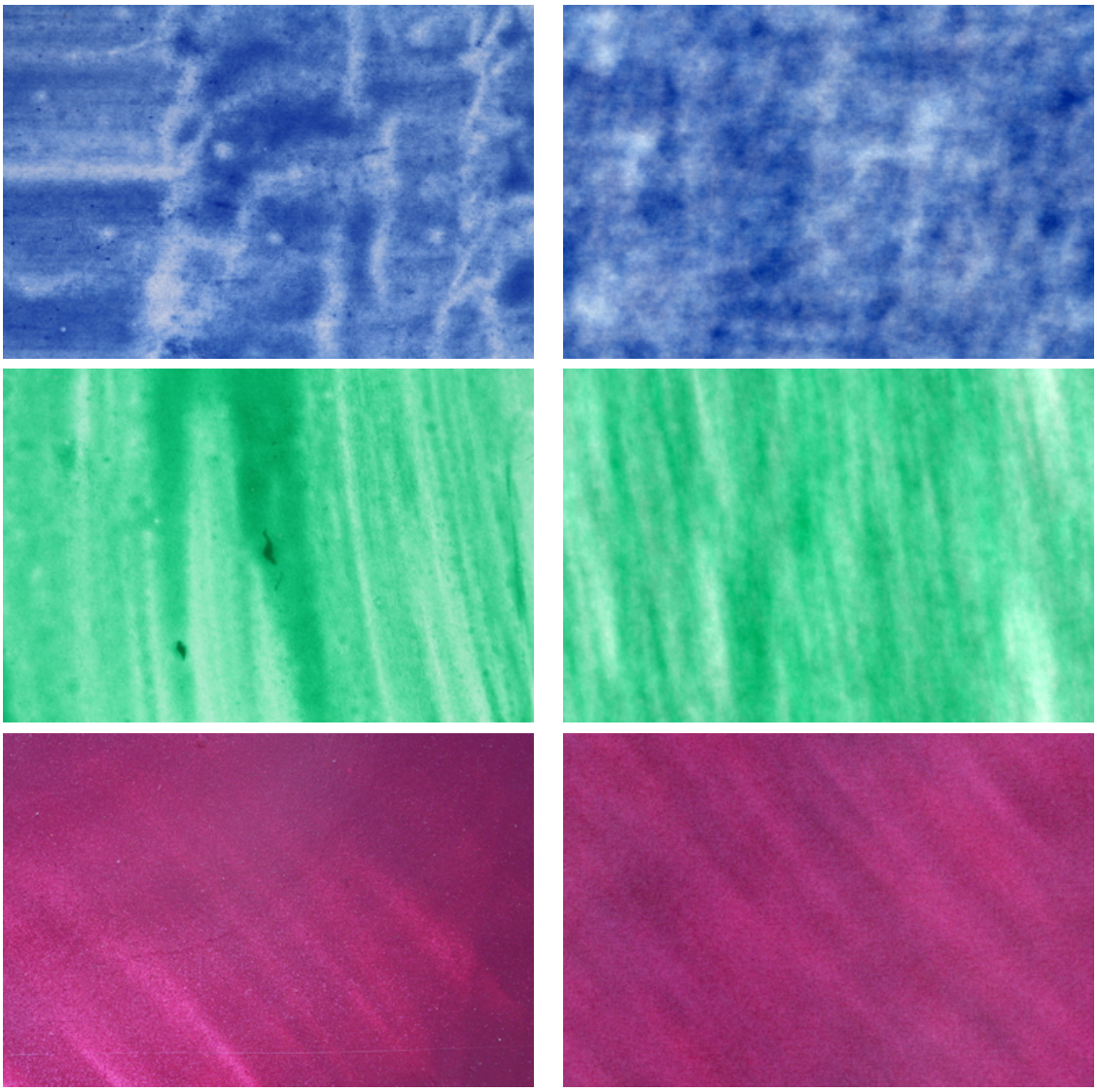


Figure 13: Paint examples. Samples (left) and RPN simulations (right).

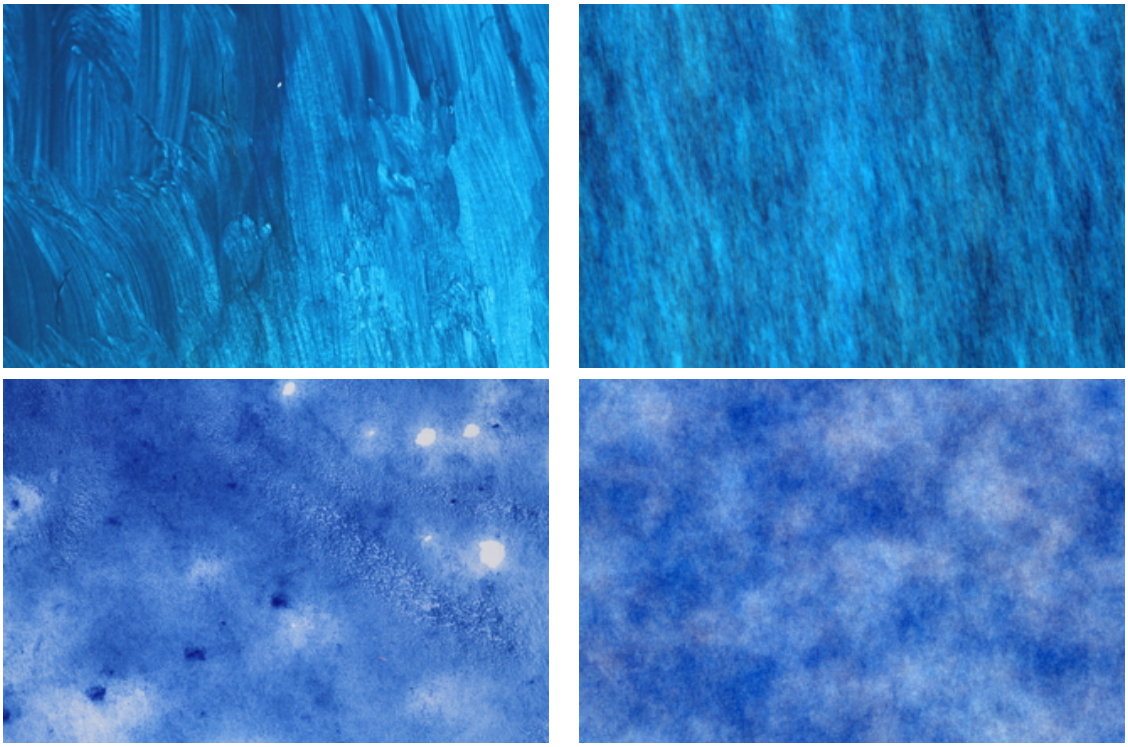


Figure 14: More paint examples. Samples (left) and RPN simulations (right).

5.1 Failure Catalog

Most failures are **macro**-textures. For instance:

- textures containing periodic geometric patterns with large period,
- textures containing strong edges, such as cracks in bark,
- textures containing definite shapes, such as knots in wood or fruit or visible leaves in foliage,
- strictly periodic patterns, even with small period, where phase shifts cause aliasing effects,
- failure also occurs when the sample texture contains different dominant directions in different areas. Then these directions are mixed by the random sampler.

Some failure examples are shown in Figures 15 to 20.

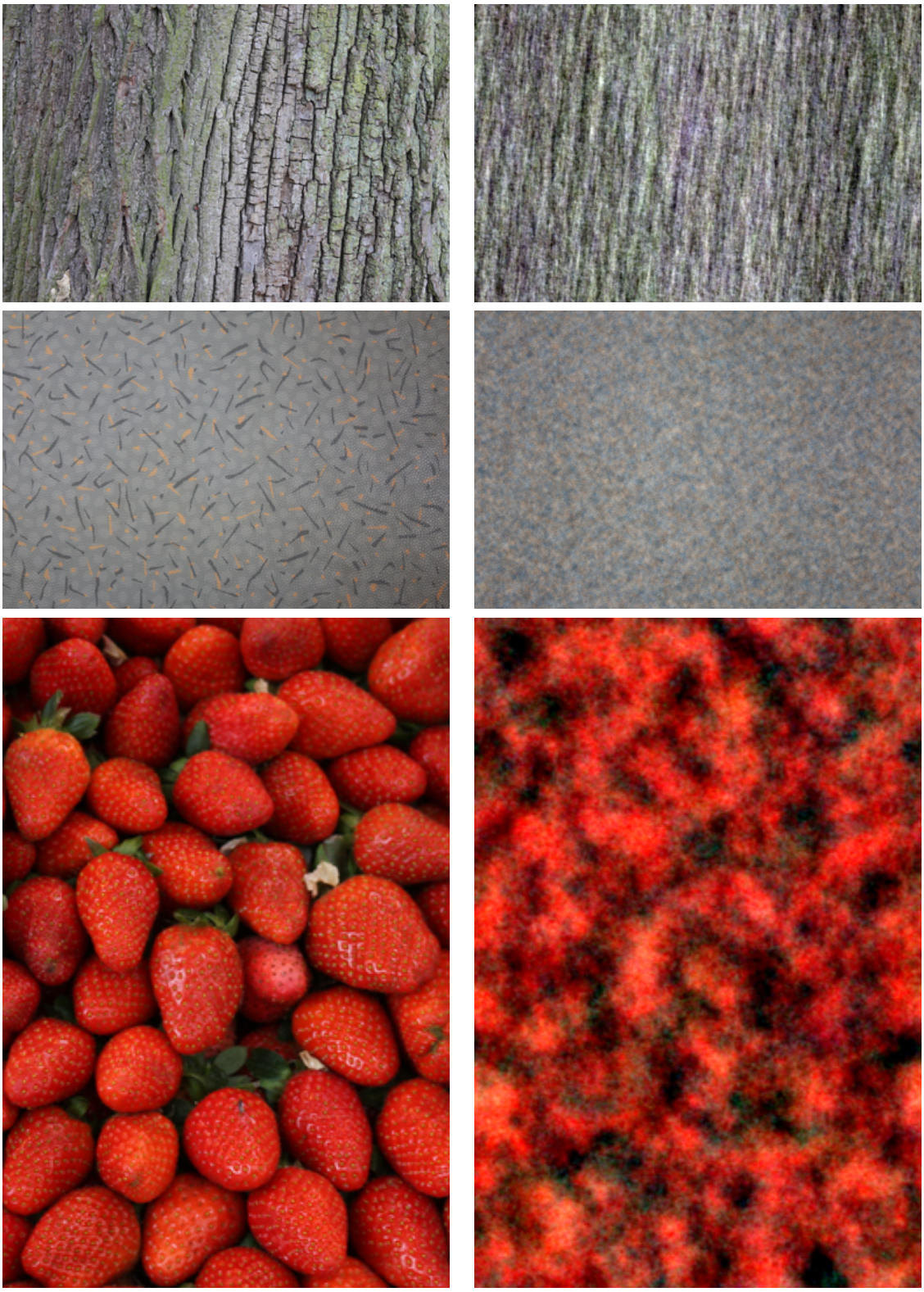


Figure 15: Failure examples. Samples (macro-textures) (left) and RPN simulations (right).

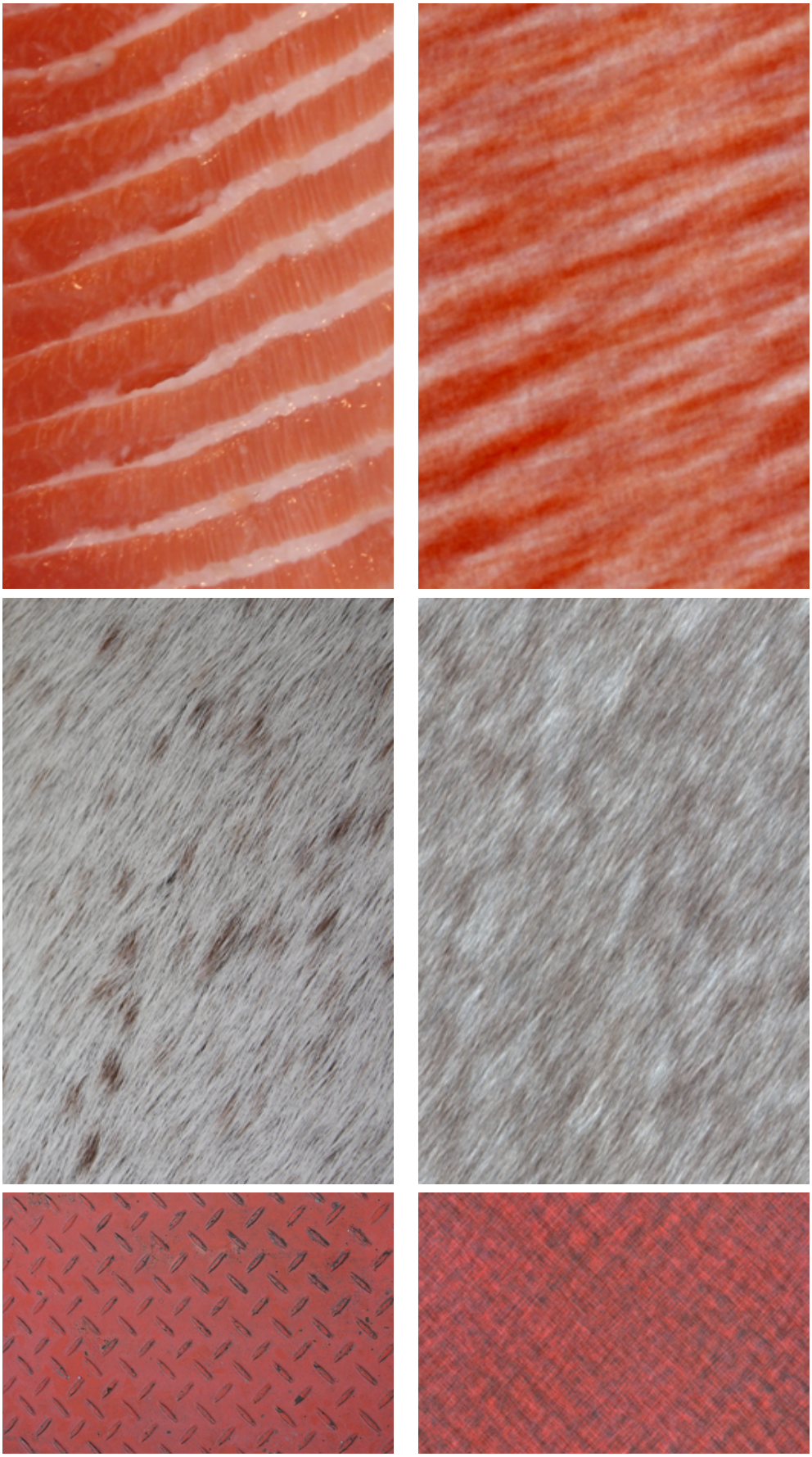


Figure 16: More failure examples. Samples (left) and RPN simulations (right).

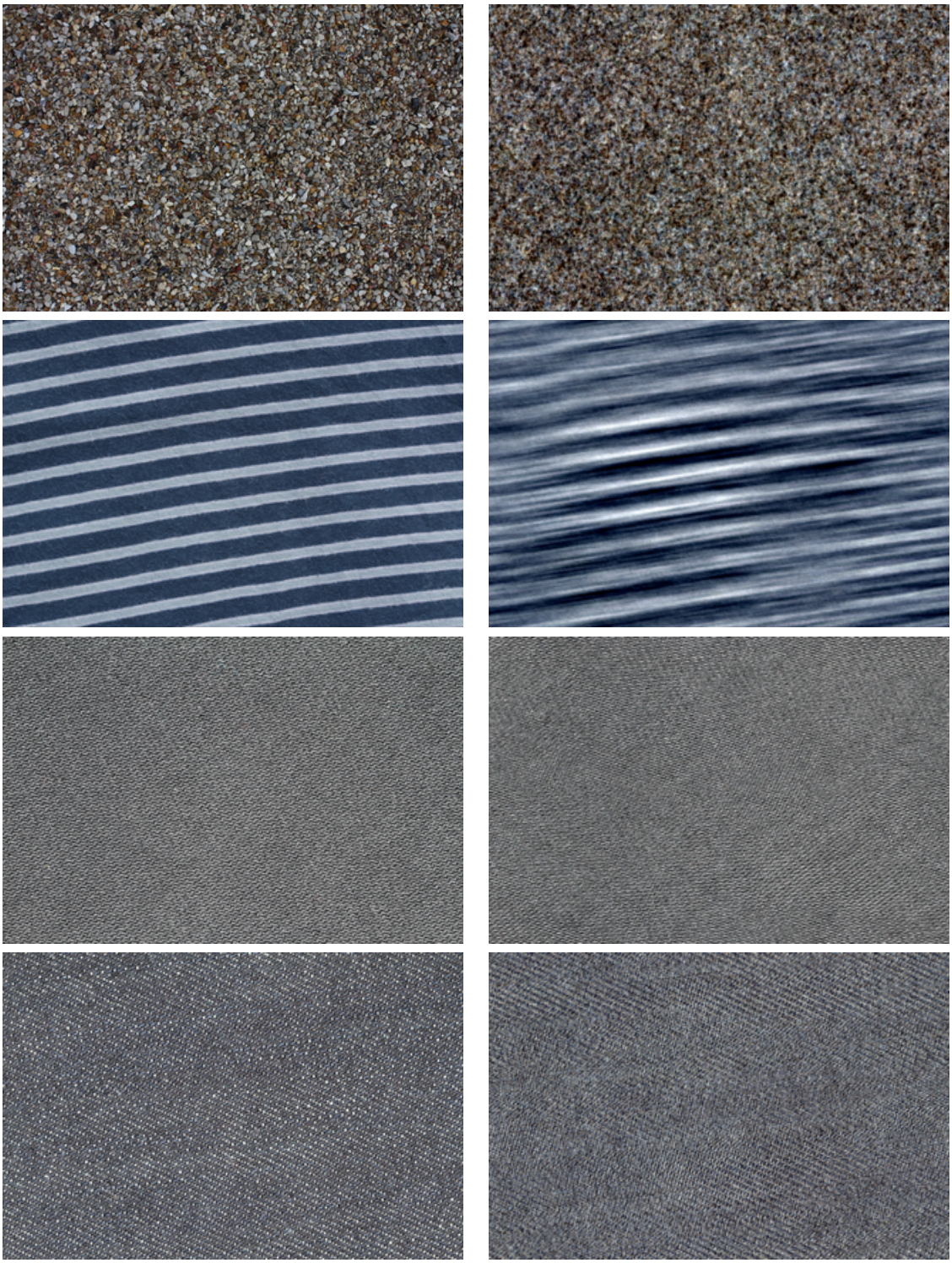


Figure 17: More failure examples. Samples (left) and RPN simulations (right).

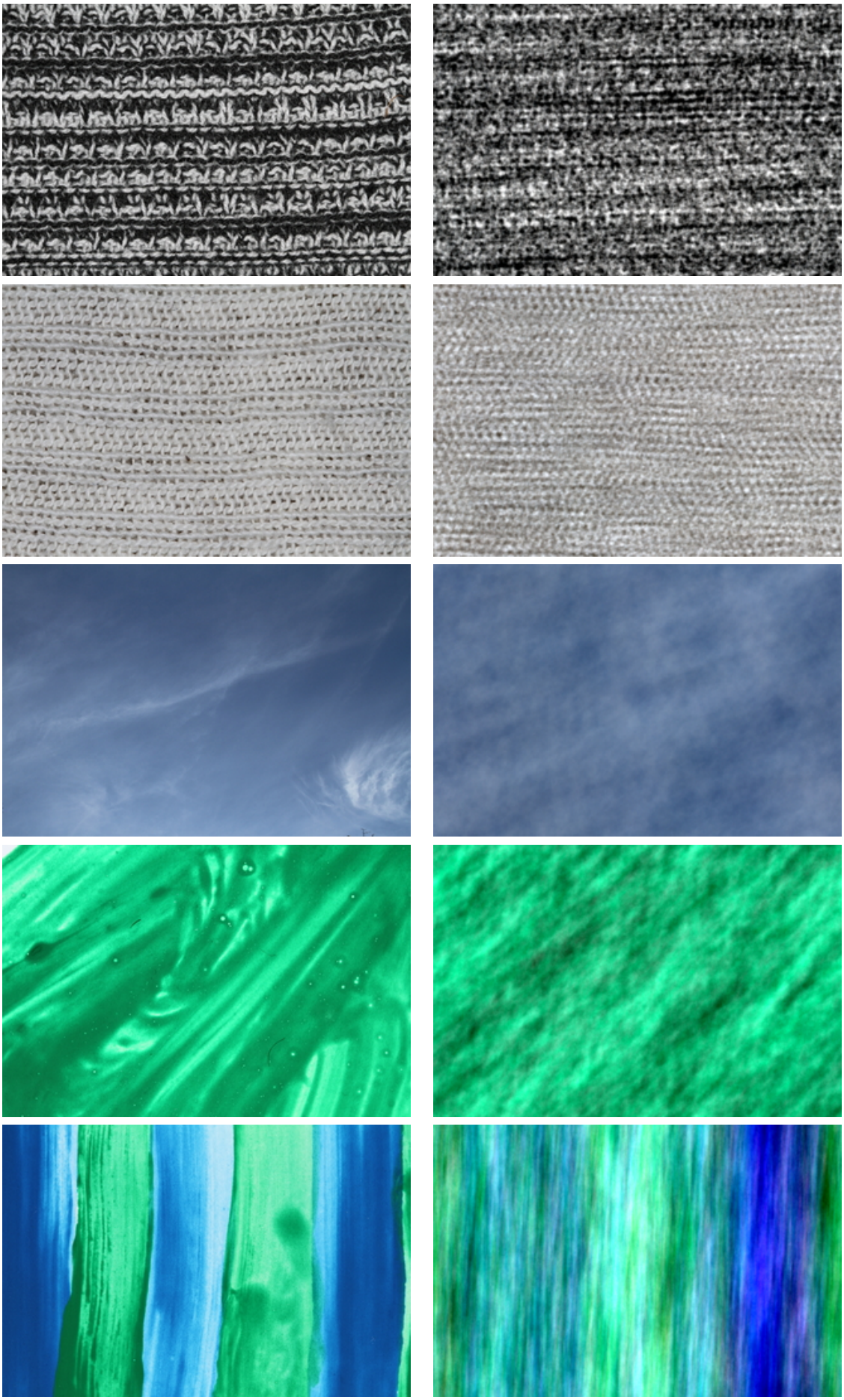


Figure 18: More failure examples. Samples (left) and RPN simulations (right).

Figure 19: More failure examples. Samples (left) and RPN simulations (right).

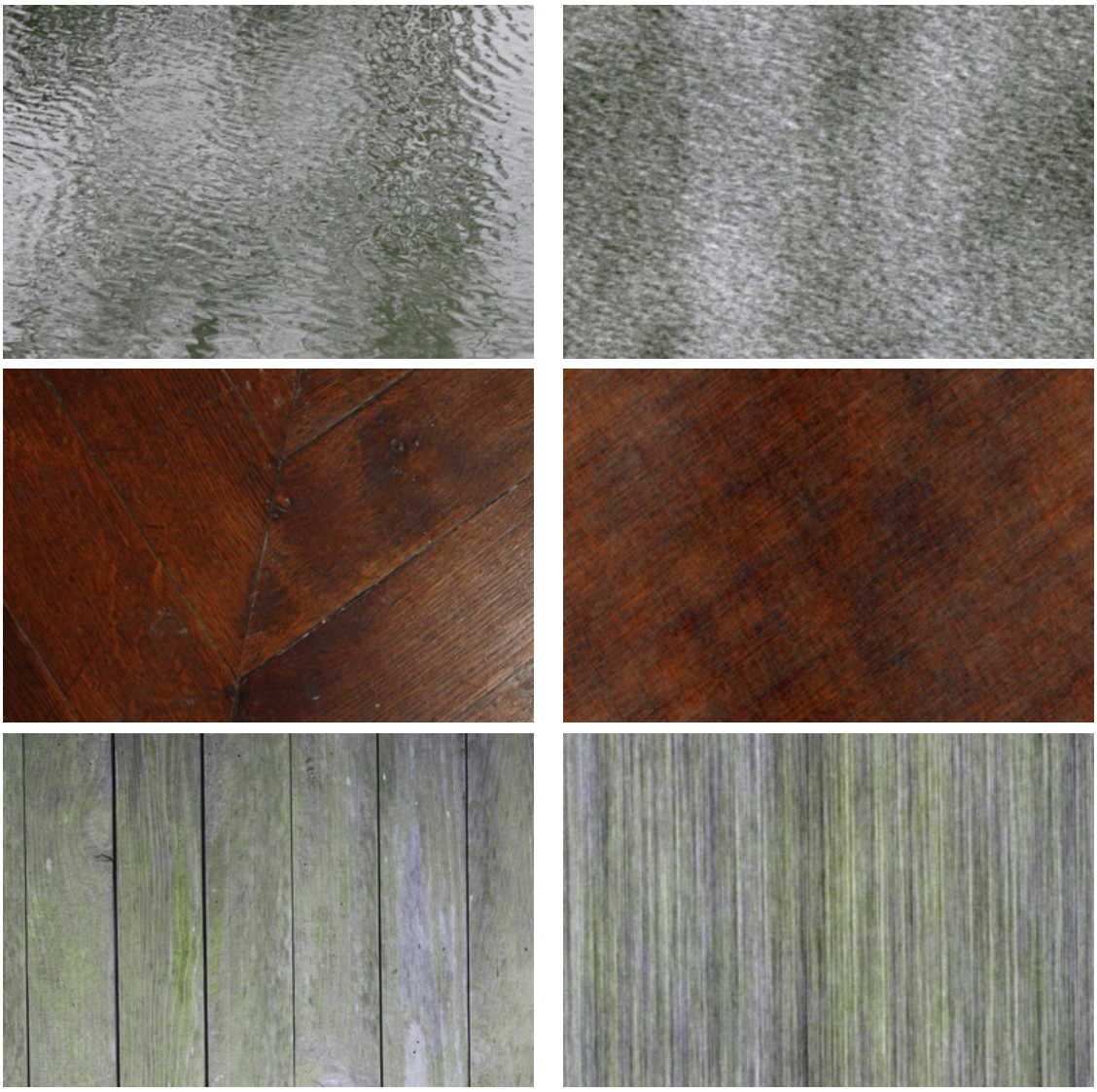
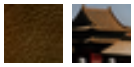


Figure 20: More failure examples. Samples (left) and RPN simulations (right).

Image Credits

 are images Sand.0002 and Tile.0006 of the VisTex database, copyright.
All other images are from the authors, CC-BY.

References

[1] B. Galerne, Y. Gousseau and J.-M. Morel, *Random Phase Textures: Theory and Synthesis*, IEEE Transactions on Image Processing, 2011. http://dx.doi.org/10.1109/TIP.2010.2052822.

[2] A. A. Efros and T. K. Leung, *Texture Synthesis by Non-parametric Sampling*, Proceedings of ICCV, 1999. http://dx.doi.org/10.1109/ICCV.1999.790383.

[3] L.-Y. Wei and M. Levoy, *Fast Texture Synthesis using Tree-structured Vector Quantization*, Proceedings of SIGGRAPH, 2000. http://dx.doi.org/10.1145/344779.345009.

[4] D. J. Heeger and J. R. Bergen, *Pyramid-based texture analysis/synthesis*, Proceedings of SIGGRAPH, 1995. http://dx.doi.org/10.1145/218380.218446.

[5] J. Portilla and E. P. Simoncelli, *A Parametric Texture Model Based on Joint Statistics of Complex Wavelet Coefficients*, International Journal of Computer Vision, 2000. http://dx.doi.org/10.1023/A:1026553619983.

[6] J. Malik and P. Perona, *Preattentive texture discrimination with early vision mechanisms*, Journal of the Optical Society of America A, 7, 923-932, 1990. http://dx.doi.org/10.1364/JOSAA.7.000923.

[7] K. Perlin, *An image synthesizer*, Proceedings of SIGGRAPH, 1985. http://dx.doi.org/10.1145/325165.325247.

[8] J. J. van Wijk, *Spot noise texture synthesis for data visualization*, Proceedings of SIGGRAPH, 1991. http://dx.doi.org/10.1145/127719.122751.

[9] L. Moisan, *Periodic plus Smooth Image Decomposition*, Journal of Mathematical Imaging and Vision, 2011. http://dx.doi.org/10.1007/s10851-010-0227-1.