EUROGRAPHICS 2026 / B. Masia and J. Thies
(Guest Editors)

COMPUTER GRAPHICS *forum*
Volume 45 (2026), Number 2

Real-time by-example texture synthesis and filtering using local statistics exchange

Nicolas Lutz[1] and Guillaume Gilet[2]

[1]LIRMM, Université de Montpellier, CNRS, France
[2]Université de Sherbrooke, Canada

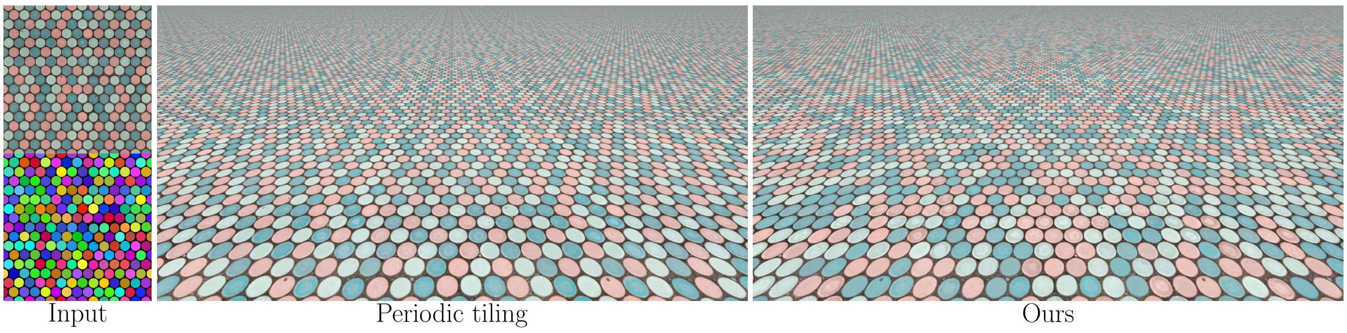

Figure 1: *Exemplar (left, top) with its corresponding region map (left, bottom) tiled on the surface (middle) and synthesized with our method (right). Our new by-example texture synthesis algorithm made for exemplars with distinct regions takes as input a periodic exemplar and its region map, and textures an unbounded, filterable, non-repetitive surface in real time.*

Abstract

Real-time by-example texture synthesis is used in interactive virtual worlds to generate the appearance of an unbounded surface from an exemplar texture with as few repetitions as possible. Currently, leading real-time methods rely on a tiling and blending scheme which is known to synthesize well texture patterns with little spatial organization (such as random scratches or noise) or patterns with periodic spatial organization (such as brick walls or regular tilings).
However, attempting to synthesize texture patterns composed of distinct, non-periodic regions with such methods remains a challenge and can lead to visual artifacts. In this paper, we propose a novel texture synthesis method to address this issue. The key of our technique relies on the observation that exchanging the appearance of periodically tiled regions is sufficient to hide repetition artifacts. We therefore present a synthesis scheme that relies on the real-time exchange of local statistics, including means, covariance matrices, or histograms.
Since our target texture is evaluated on the fly, naive filtering schemes relying on precomputation, such as direct MIP-mapping, do not provide accurate results. Therefore, we propose an adequate real-time filtering approximation and show that our method produces high-quality results with little artifacts and a GPU-friendly implementation.

CCS Concepts
• ***Computing methodologies*** $\rightarrow$ ***Rendering; Texturing;***

1. Introduction

In online virtual world generation and rendering, texture synthesis is used to generate textures, which configure the appearance of surfaces. Offline texture synthesis is used to generate a bounded texture from a set of parameters, while online/real-time texture synthesis generates textures on an unbounded surface, on demand, and in real-time. By-example synthesis is a form of either offline or online synthesis that generates a texture from a sample texture called the exemplar. In this work, we are exclusively interested in online by-example synthesis. Such algorithms are subjected to many constraints in terms of performance and quality: they must be executed in parallel, on a per-pixel demand, and as quickly as possible; they must take as little memory consumption as possible; they must limit the amount of tiling artifacts, i.e. visual repetition and alignment; finally, the generated texture must be filtered on the fly to avoid under-sampling artifacts. Popular real-time algorithms are usually carried out by reorganizing tiles of content taken from the exemplar [Wei04, VSLD13], blendings [GLM17], or, more recently, a combination of tiling and blending [HN18, DH18, Bur19]. They are often designed for a class of stochastic textures that exhibit little global organization. Modifications of these algorithms enable the synthesis of patterns exhibiting a global periodic organization [LSD21], or increase the preservation of internal correlations [LSD23], but fail to reproduce complex patterns composed of irregularly-shaped regions, yielding important visual artifacts (see Figure 2).

In this work, we propose a novel texture synthesis method to deal with such complex texture patterns using local statistics exchanges of the content of these regions. The key insight of our method is that most visual repetition artifacts can be avoided without degrading quality by exchanging the local statistics of each region of a perfectly periodic tiling. Our method takes as input the exemplar texture and its corresponding region map, used to identify each individual region. During a pre-computation step, local statistics (local means, local covariance matrices and local histograms) of each region are computed and stored. During rendering, the exemplar is periodically tiled, but the local statistics of each region of the output are exchanged together. While the structure of the exemplar remains strictly periodic, most repetition artifacts are avoided, as shown in Figure 2. We propose three different recoloring schemes of increasing complexity. On-the-fly filtering of the output, however, is not a trivial task as our recoloring scheme precludes direct MIP-mapping of the exemplar. To make our synthesis viable for high-quality rendering, we propose a fast approximation of the filtering process.

2. Related works

2.1. Real-time texture synthesis

Texture synthesis can be split into two families: offline texture synthesis, mostly used for discrete texture authoring as a pre-computation step, and online or real-time texture synthesis, tailored for real-time rendering. These algorithms must be as fast as possible (often measured in number of Texture Accesses Per Texel of the output, which we abbreviate *TAPT* here), and must consume as little memory as possible.

Early tilings. Wei [Wei04] showed that Wang tiling techniques [CSHD03] can run in parallel on graphics hardware once a set of matching squared texture tiles is precomputed. This method is extremely efficient, as it only requires 1 TAPT. The main issue is that the variety of the output is tied to the amount of pre-computed tiles. In practice, Wang tiling always exhibits aligned repetitions, because tiles are limited in number due to their memory cost, look similar, and are tiled on a regular grid.

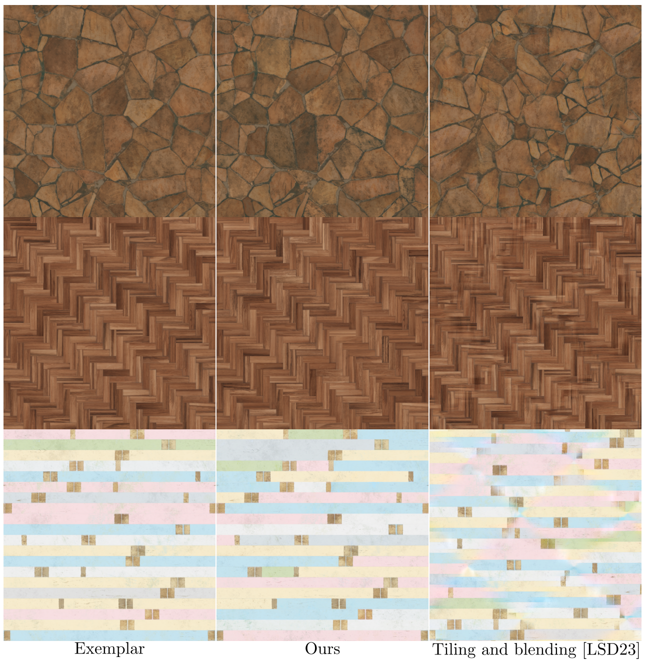

Figure 2: *In each view : Exemplar (left) synthesized with our algorithm (middle) and dual square tiling and blending [LSD23] (right). Tiling and blending creates ghosting artifacts such as stones blending into each others and seams such as planks abruptly transitioning from one color to another, while ours does not.*

The content exchange algorithm [VSLD13] works with a periodic region map similar to ours. In content exchange, the exemplar is periodically repeated, but an alternative content for each region can be sampled from pre-determined offsets in the exemplar, which removes the need for extra content storage. However, reaching a high variety requires a high amount of different contents, which increases the risk of visible seams between regions. On top of this, its only known filtering method is expensive to compute and requires explicitly storing the alternative contents and its MIP-maps [LSLD19], thus tying memory consumption and variety, similarly to Wang tiling.

Procedural noise. Procedural noises form a set of algorithms simulating stochastic fields (usually stationary Gaussian fields), to generate textures [LLC+10]. The algorithms are often based on some kind of theoretical sparse convolution process involving the convolution of a kernel in the spatial or frequency domain, such as asymptotic discrete spot noise or random phase noise [GGM11]. Among instances of these algorithms, Gabor noise [GLLD12], local random phase noise [GSV+14], and texton noise [GLM17] are by-example parallelizable variants, but they often take many texture or memory accesses to converge (authors of texton noise mention 30 TAPT). Later, Heitz et Neyret [HN18] presented a faster alternative to classic procedural noise called tiling and blending, relying on the blendings of texture tiles randomly chosen on the fly in the exemplar. This technique only requires 3 TAPT, though Lutz et al. [LSD23] proposed a faster version requiring only 2 TAPT. To increase the range of possible patterns, Guehl et al. [GAD+20] proposed a generic form of sparse convolution allowing the generation of a wide variety of different shapes. However, choosing the basis functions of the generation process for a given targeted pattern is a difficult task, and filling these shapes with complex details in real-time remains an open issue.

Non-stationary patterns. All of the aforementioned synthesis simulate stationary fields, designed for disorganized patterns. Lutz et al. have shown that tiling and blending can be used to synthesize patterns with a periodic global organization, using cyclostationarity [LSD21], and patterns with specific inner correlations [LSD23]. However, these techniques are not well-suited for synthesizing patterns with distinct regions exhibiting different local appearances, which we tackle in this work without tiling and blending. To achieve synthesis of non-stationary complex patterns, Guingo et al. [GSDC17] proposed to decompose a given texture into a structural layer made of n regions and a common noise layer. However, this technique is inherently limited to a few different regions as the number of stored masks and spectra linearly increases with the number of regions. Furthermore, properly filtering the combination of structural and noise layers is not trivial and remains an open issue. Although our method cannot change the structure of the exemplar, it is faster, filterable, and not bounded by the number of different regions.

2.2. Statistics Exchange

Coloring/Whitening. Coloring is the process of associating a covariance matrix to a set of observations with an identity matrix, while whitening is its inverse (associating an identity covariance matrix to a set of observations). Both coloring and whitening can be encoded in a matrix. It is often used for style transfer [Chi19]. PCA Whitening and coloring, in particular, have often been used in texture synthesis to temporarily remove unwanted correlations between texture channels, allowing texture synthesis processes that would otherwise break these correlations [HB95, Eom00, GSDC17]. It was notably used by Deliot and Heitz [DH18] to decorrelate color channels, during a pre-computation step, and perform tiling and blending and histogram transfer in a decorrelated color space. Our real-time covariance exchange of section 4.3 changes local appearances using a recoloring scheme inspired by these works.

Histogram specification. Histogram specification is the process of mapping the histogram of an image to another. It is typically done in image processing by equalizing the histogram in a source image and associating it with a target histogram. In texture synthesis, it can be useful for reestablishing a histogram lost after a histogram-destructing texture synthesis process. Galerne et al. [GLR17] used it for exemplars synthesized with a texture synthesis algorithm simulating a Gaussian process, which produces outputs with Gaussian statistics. Heitz and Neyret [HN18] adapted this solution for real-time synthesis by using a lookup table. Deliot and Heitz [DH18] improved this method by using a principal component analysis (PCA) to decorrelate the n channels of the lookup table. It enables the execution of n 1D histogram exchanges instead of one nD histogram exchange, thus saving storage and computation capabilities. Furthermore, they showed how to correctly filter a texture subjected to a histogram specification by convolving it with a Gaussian kernel of resolution-varying size. Our real-time local histogram exchange of section 4.4 is inspired by these recent propositions.

2.3. Procedural texture filtering

Texture filtering of discrete textures is usually done by MIP-mapping the texture, and, in real time, interpolating between the closest resolutions of the MIP-map. However, in real-time texture synthesis, the actual texture on the target surface is not known in advance, often requiring one different filtering technique per synthesis algorithm. For Wang tiling, Wei [Wei04] used a MIP-mapping of a content-aware packing of all possible contents, leading to a satisfactory enough filtering if the contents are squared. Filtering the content exchange algorithm [VSLD13] is more difficult, as contents have irregular shapes and are defined as an indirection into the exemplar. Lutz et al. [LSLD19] correctly filtered it by explicitly storing contents and their MIP-mapped versions in an atlas, then assembling them in real-time using a special MIP-mapping of the region map to reconstruct the output at the required resolution. In theory, we actually could use this filtering technique, but its memory footprint and maximum rendering cost scale with the amount of different contents, which, in our case, is equal to the number of regions rather than a fixed small number, thus making it very expensive. Our filtering model (presented in section 5) relies instead on an approximation and does not require storing any extra content nor does it require iterating over an unknown amount of overlapping contents, at the cost of accuracy for lower resolutions.

3. Mathematical background

In this section, we mathematically express notions of textures, coloring and whitening, histogram specification, and filtering used for this work.

3.1. Textures

We define a texture as a function $I : X \to S$, where X is the index set representing the indices of texels, and S is the d-dimensional value set representing their values. In this work, we consider only 2D textures, therefore X is considered to be $\mathbb{R}^2$ throughout this paper. Textures explicitly defined on indices in $[0,1]^2$ can be implicitly extended on $\mathbb{R}^2$ by periodic tiling. We also consider the set of values S to be $\mathbb{R}^d$ throughout this paper, as is typical for texture values stored on a GPU. For instance, S can be $\mathbb{R}^3$ if the targeted texture only includes albedo, or $\mathbb{R}^4$ if it also includes a roughness map.

3.2. Coloring and whitening

Covariance. In this work, we are interested in the covariance between different dimensions of the same vector-valued random variable $Y \in \mathbb{R}^d$, i.e.

$$\mathrm{Cov}(Y,i,j) = \mathbb{E}\left[\left(Y_i - \mathbb{E}\left[Y_i\right]\right)\left(Y_j - \mathbb{E}\left[Y_j\right]\right)\right], \qquad (1)$$

where i and j correspond to different indexed dimensions of Y. The covariance matrix Σ_Y of a random variable Y is the symmetrical matrix storing the Cov for each possible pair (i, j).

An estimate of the covariance function $\widehat{\text{Cov}}$ can also be straightforwardly computed on a set of observations deemed to be of the same random variable. In our case, an estimation of local covariance matrices is used for computing local whitenings and colorings in section 4.3.

Whitening. A whitening matrix is a matrix W that encodes a set of mean-centered observations $\hat{Y}$ in S such that the covariance matrix of $\hat{Y}W$ is the identity matrix. There are an infinite number of functions W that satisfy this constraint. However, in most cases, only invertible linear functions are useful, among which the *Principal Components Analysis* (PCA) and *Zero Components Analysis* (ZCA) whitenings are some of the most commonly used [Chi19].

If we express the Eigen decomposition of a covariance matrix $\Sigma = U \Lambda U^T$, where U is the matrix of eigenvectors and Λ the diagonal matrix of eigenvalues, then the PCA (resp. ZCA) whitening matrices are

$$W_{PCA} = \Lambda^{-\frac{1}{2}} U^T \quad , \quad W_{ZCA} = U \Lambda^{-\frac{1}{2}} U^T . \tag{2}$$

PCA whitening is well-known in texture synthesis and projects the observations in an orthogonal basis aligned with the principal component. ZCA whitening further rotates the observations so that they are aligned with the orthonormal axis. The matrix W_{ZCA} also has the benefit of being symmetrical, which reduces the amount of coefficients that needs to be stored and read in our case.

Coloring. Coloring is the inverse of whitening. A coloring matrix W^{-1} decodes a set of whitened observations, i.e., applies a covariance matrix embedded in W^{-1} to a set of centered observations with an identity covariance matrix. In other terms, if W^{-1} embeds a covariance matrix Σ and we are given a set of observations $\hat{Y}$ that have an identity covariance matrix, then $\hat{Y}W^{-1}$ takes Σ as its covariance matrix. The PCA (resp. ZCA) coloring matrices are

$$W_{PCA}^{-1} = U \Lambda^{\frac{1}{2}} \quad , \quad W_{ZCA}^{-1} = U \Lambda^{\frac{1}{2}} U^T . \tag{3}$$

Similarly to the ZCA whitening W_{ZCA}, the ZCA coloring matrix W_{ZCA}^{-1} is also symmetrical.

3.3. Histogram specification

Histogram specification is the process of mapping the histogram of a set of observations onto another. It can be achieved for vector-valued observations by computing an optimal transport plan. However, due to the complexity of such a specification, we focus on dimension-wise histogram specification with the help of the decorrelation allowed by the whitening of section 3.2.

Given a set of scalar observations $\hat{y}_i \in \hat{Y}$ with identity covariance matrix, let F be its cumulative distribution function (CDF) and G an arbitrary target CDF. The histogram specification is therefore defined through the mapping

$$\hat{y}_i \mapsto G^{-1}(F(\hat{y}_i)). \tag{4}$$

Histogram equalization is a specific instance of histogram specification where G is the uniform distribution, which boils down to removing G^{-1} in equation 4. In real-time computer graphics, F and G^{-1} are often discretized and stored as lookup tables. We make use of histogram specification to exchange local histograms in section 4.4.

3.4. Filtering

In rendering, filtering refers to computing the footprint of a screen pixel over a textured surface and averaging its content. Using the notation of Grenier et al. [GSDT22], it consists in solving exactly or approximately

$$\bar{I}(\mathcal{P}) = \int_{\mathcal{P}} I(x)\, dx \tag{5}$$

for any texture I and footprint $\mathcal{P}$. Solving this equation in real time usually involves pre-integration, such as MIP-maps for explicitly stored textures. However, real-time texture synthesis often involves some kind of indirection, which makes the use of MIP-maps more challenging [LSLD19, GSDT22]. Since our synthesis is also subjected to such difficulty, we propose an approximate solution in section 5.

4. Texture synthesis model

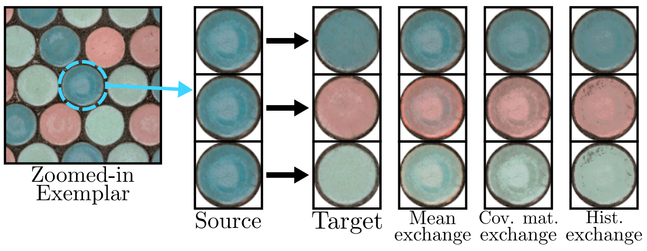

Figure 3: *Illustration of the three synthesis algorithms we present in section 4. During rendering, the statistics of each source region are mapped to a target region, using one of three algorithms of increasing complexity: mean exchange (section 4.2), covariance matrix exchange (section 4.3) or histogram exchange (section 4.4). Mean exchange is simple, but can lead to subtle false colors (especially visible on the borders). Covariance exchange (here with a PCA whitening/coloring matrix) yields better colors. Histogram exchange is more expensive and can lead to noise in the output, but values are better kept within the computed histogram of the target.*

In this section, we present our texture synthesis model. It takes as input a periodic exemplar texture as well as its corresponding periodic region map, which serves to identify each local region as described in section 4.1. Our synthesis roughly consists in first precomputing and storing local statistics for each of the regions of the exemplar; then, during rendering, these regions are periodically tiled over a target surface, but the local statistics of each region are randomly exchanged.

Several statistics can be exchanged and produce different visual results: this process is described in sections 4.2 for local means, 4.3 for covariance matrices and 4.4 for histograms. These three processes are illustrated in Figure 3.

4.1. Region map

We define the region map as a function $R : X \to \mathbb{N}$ which acts as a map to label each region of the exemplar texture E. The region map can be manually authored from the exemplar, semi-automatically computed by a segmentation algorithm or model, such as SAM2 [RGH+24], or directly extracted from offline texture creation tools, such as Substance Designer or Material Maker.

For convenience, we define $R(x) = 0$ as the *background region*, which is a region that may or may not be present in R and that we do not subject to a statistics exchange. We use the notation X_k to denote all *indices of the k-th region*, i.e. the set $\{x \,|\, R(x) = k\}$. Similarly, we use the notation E_k to denote the *content of the k-th region*, i.e. the set $\{E(x) \,|\, x \in X_k\}$.

Finally, we define $\tau : \mathbb{N}, X \to \mathbb{N}$ as a function that yields a potentially different, often pseudo-random and valid region number to each continuous region of R, with the exception of the background region, which is subject to $\tau(0,x) = 0$ and $\tau(k,x) \neq 0, \forall k > 0$. This function is used during rendering to exchange local statistics.

4.2. Local mean exchange

Local mean exchange consists in replacing the spatial mean of a local source region with that of a target region.

The spatial mean μ is the average value of a set of samples. The local spatial mean μ_k of an exemplar texture E and a region map R describing X_k is

$$\mu_k = \frac{1}{|X_k|} \int_{X_k} E(x)\, dx. \tag{6}$$

In real time, we can therefore exchange local means by subtracting the local source mean and offsetting with a local target mean chosen at random, i.e.

$$I(x) = E(x) - \mu_{R(x)} + \mu_{\tau(R(x),x)}\,. \tag{7}$$

Local mean exchange is the cheapest solution we propose. However, the mean is a very approximate representation of the local first-order statistics of a region, and exchanging it often yields false colors, as can be seen in Figure 3 next to the border of the regions. To alleviate this issue, we propose to exchange local covariance matrices in the next section 4.3.

4.3. Local covariance matrix exchange

Local covariance exchange consists in replacing both the spatial mean and the covariance matrix of a local source region with those of a target region.

We first fill a local covariance matrix Σ_k where the i-th line and j-th column are estimated as $\widehat{\mathrm{Cov}}(E_k, i, j)$ from equation 1, considering the region content E_k as the input set of observations. From all Σ_k, we compute a local whitening matrix W_k and a local coloring matrix W_k^{-1} using either the PCA or ZCA method of equations 2 and 3.

In real time, we can therefore exchange local covariance matrices by coloring the whitened version of E with a different random target coloring matrix and offset it with the mean of the same target region, i.e.

$$I(x) = W^{-1}_{\tau(R(x),x)} \cdot E_W(x) + \mu_{\tau(R(x),x)}\,, \tag{8}$$

where

$$E_W(x) = W_k \cdot \Big(E(x) - \mu_{R(x)}\Big) \tag{9}$$

is the centered, whitened version of E, which can either be computed on the fly or pre-computed.

Note that for PCA whitening, it is possible for the texel values of $I(x)$ to reflect locally around the mean (e.g. light colors turn dark and vice versa) due to different whitening/coloring pairs rotating texel values in antagonistic directions. To solve this issue, we ensure that each of the eigen vector of Σ_k has the same direction as its closest orthonormal axis (i.e. the axis corresponding to the largest absolute dimension of the eigen vector); otherwise, we reflect that eigen vector, still yielding a valid whitening/coloring pair.

Whether to use PCA or ZCA covariance exchange is difficult to determine. They both produce similar-looking results that differ in close-up details in a way that is highly dependent on the exemplar. We discuss this further in one of our supplemental material.

Local covariance exchange is a little more expensive in memory and computation time than mean exchange, as it requires the minimum additional storage of n coloring matrices (and n whitening matrices if E_W is not pre-computed), one of which is read per texel. It generally produces believable colors that match the target, compared to the simpler mean exchange. However, this does not guarantee a perfect match with the target distribution, which is why we also propose the histogram exchange in the next section 4.4.

4.4. Local histogram transfer

Local histogram exchange consists in mapping the full histogram of the content of a region with that of a target region using histogram specification. As stated in section 3.3, we exploit the properties of the whitening process to ensure that different value dimensions of the exemplar are decorrelated. This requires us not to pre-compute the CDF of E, but rather that of its whitened version E_W of equation 9, as well as its inverse.

In real time, we can therefore transfer local histograms by specifying the equalized version of E using a different random CDF, coloring it and offsetting it with the color matrix and mean of that same random region :

$$I(x) = W^{-1}_{\tau(R(x),x)} \cdot F^{-1}_{\tau(R(x),x)}\left(E_F(x)\right) + \mu_{\tau(R(x),x)}\,, \tag{10}$$

where

$$E_F(x) = F_{R(x)}(E_W(x)) \tag{11}$$

is the centered, whitened and then equalized version of E, which can either be computed on the fly or pre-computed; F_k and F_k^{-1} are respectively the CDF and the inverse CDF for the region k. Note that in most implementations, care should be taken to normalize $E_W(x)$ between 0 and 1, and to de-normalize $F^{-1}_{\tau(R(x),x)}(E_F(x))$. This can be done in real time by storing the minimum and maximum value of each whitened region.

Local histogram exchange is significantly more expensive in memory and computation time than covariance exchange, as it requires storing n quantized inverse CDFs (and n quantized forward CDFs if $E_F(x)$ is not pre-computed), one of which is read in real-time per texel. It generally provides a better match to the target histogram than the local covariance transfer, but often leads to abrupt transitions between values in the output. This is further discussed in section 6.2.

5. Filtering model

In this section, we present our anisotropic filtering model for filtering an output texture generated with the synthesis model of section 4. Since our synthesis model relies on an indirection through a region map R, filtering cannot be trivially done by MIP-mapping the exemplar. An accurate model should instead compute an indirection for all of the different regions found in a given footprint in real time. This was done by Lutz et al. [LSLD19], who reconstruct a virtual MIP-map of I in real time using pre-filtered texture contents. However, this technique comes at the expense of memory space for storing pre-filtered contents, and performance for reconstructing low resolutions.

We therefore propose an approximate filtering model based on the principle of reconstructing a virtual MIP-map of I, but with a fixed number of texture accesses at every level and without storing pre-filtered contents. It relies on a combination of three MIP-maps : the exemplar, the region map R and a contribution map C. The region map is MIP-mapped so that each texel encodes the *majority region* for its corresponding footprint, i.e. the region that takes up the most space in that footprint, as described in section 5.1. The contribution map C is computed and MIP-mapped so that each texel gives the contribution ratios of three kind of regions under a given footprint: the *majority region*, the *background*, and the *rest*, as described in section 5.2. These two structures allow us to perform at worst 2 indirections and approximate other indirections with a *substitute*, as described in section 5.3.

In the following section, we detail this filtering model: how the region R is MIP-mapped, how the contribution map C is computed and MIP-mapped, and how they are used together with the MIP-map of the exemplar to create an approximate anisotropic filtering model.

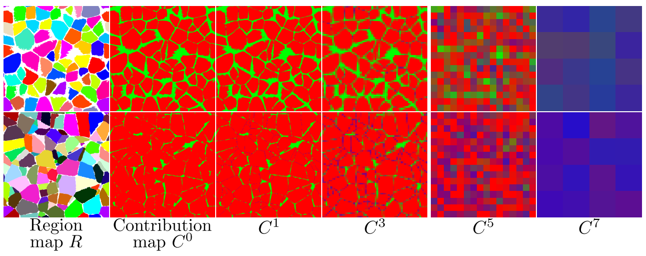

Figure 4: *Contribution maps $C = C^0$ computed from a region map R, along with instances of C^l for MIP-map levels $l = 1$, $l = 3$, $l = 5$ and $l = 7$. C_{maj} is color-coded in red, C_{fix} in green and C_{rest} in blue. C_{rest} appears as several regions subjected to a statistics exchange merge, before covering most of the contribution map in low resolutions.*

5.1. MIP-mapping the region map

An ideal MIP-mapping of the region map R should represent all regions that fall under the corresponding footprint of a texel of the MIP-map. Lutz et al. [LSLD19] use a bit mask representation and create MIP-map levels by using a 4 ways OR operator instead of the usual 4 ways interpolation. By iterating over the different regions, a MIP-map of the synthesized texture can be reconstructed in real time. However, this technique requires a variable and potentially high amount of texture accesses, especially in low resolutions, where the number of overlapping regions is the highest. We instead choose to store in each texel of the MIP-map of R the index of the *majority region*: the predominant index in the footprint $\mathcal{P}$ of the texel in the base level of R. A MIP-map level R^l is computed from R as

$$R^l(\mathcal{P}) = \arg\max_r |\{x \in \mathcal{P} \,|\, R(x) = r\}|\,, \tag{12}$$

5.2. MIP-mapping the contributions map

The contribution map C represents the number of texels of the highest resolution that belong to a given kind of region, for all texels in the footprint. Similarly to the region map R, the contribution map C cannot be trivially MIP-mapped.

Each texel $C^l(x)$ in each resolution l takes three kinds of value that sum up to 1, depending on the contribution of each kind of region :

- the contribution of the majority region, if it is subjected to a statistics exchange, denoted as $C_{maj}(x)$,
- the contribution of the background region which is not subjected to a statistics exchange, denoted as $C_{fix}(x)$,
- the contribution of the rest of the regions subjected to a statistics exchange, denoted as $C_{rest}(x)$.

Each contribution is computed from the highest resolution by computing the amount of texels within the footprint of each resolution that respect these conditions. Using equation 12, we have

$$C^l_{maj}(\mathcal{P}) = \frac{\left|\left\{x \in \mathcal{P} \,|\, R(x) = R^l(\mathcal{P})\right\}\right|}{\mathcal{F}(l)} \tag{13}$$

Recalling that we defined the background to follow $R(x) = 0$, this yield

$$C^l_{fix}(\mathcal{P}) = \frac{|\{x \in \mathcal{P} \,|\, R(x) = 0\}|}{\mathcal{F}(l)}, \tag{14}$$

where $\mathcal{F}(l)$ is the size of this footprint, which is 2^{2l} for squared and power of two textures. Since the contributions sum up to 1, we have

$$C^l_{rest}(\mathcal{P}) = 1 - C^l_{maj}(\mathcal{P}) - C^l_{fix}(\mathcal{P}), \tag{15}$$

Hence, storing C^l_{rest} in memory is not required. The rest C^0_{rest} is equal to 0 everywhere for $l = 0$, emerges at lower resolutions where several regions overlap, and covers most of the contribution map at the lowest resolutions, as seen in Figure 4.

5.3. Real-time MIP-map reconstruction

Our filtering relies on an approximate reconstruction of the MIP-map of the output texture I, which we call I^l for each MIP level l. Using both R^l and C^l along with a trivial MIP-mapping of the exemplar E^l (in a space that depends on the technique used in section 4), each texel of I^l is computed in real time as the per-texel product

$$I^l = O^l C^l_{maj} + E^l C^l_{fix} + S^l C^l_{rest}, \tag{16}$$

where O^l is the output of any statistics exchange model (equations 7, 8, or 10), but with the region map R replaced with its MIP-mapping R^l (equation 12) and the exemplar E replaced with its MIP-mapping E^l. S^l is the substitute function described in the following section 5.3.

The term S^l in equation 16 is an approximate substitute meant to plausibly replace the integral of all non-majority regions that fall under a given footprint. We propose three variants of increasing complexity to define S^l, each with their pros and cons. Their effects are shown in Figure 5.

Mean as substitute. The mean of all regions subjected to a statistics exchange may be precomputed and used as a substitute. It has the disadvantage of blurring the textured surface when viewing from far away or with a grazing angle, which is especially visible when anisotropic filtering is enabled (as opposed to only using trilinear filtering).

Exemplar as substitute. The periodically-tiled exemplar may also be used as a substitute. It has the disadvantage of reproducing alignment artifacts when viewing from far away or with a grazing angle. Using this substitute also requires either storing the exemplar or computing an identity statistics exchange on the pre-computed whitened or equalized version of E, i.e., replacing the statistics of each region with its own.

Real-time fast stationary synthesis as substitute. A different real-time synthesis method can also be used as a substitute. Such a substitute does not exhibit alignment artifacts, similarly to the global statistics exchange, while closely reproducing the frequency content of the filtered exemplar, thus avoiding blurriness. However, it has the disadvantage of creating an aspect that does not match between different resolutions, which can be visible when the content of different regions have extreme variations. Like using the exemplar as a substitute, it also requires either storing the exemplar or computing an identity statistics exchange on the pre-computed whitened or equalized version of E. Any real-time texture synthesis may be used as long as it is trivially filterable using a MIP-map of the exemplar, and the substitute inherits the properties of the used texture synthesis algorithm. In our experiment, and due to its performance, we used the 2 texture access tiling and blending of Lutz et al. [LSD23], using two overlapping square tiles for each texel.

5.4. GPU implementation

Texture maps All maps required for our synthesis and this filtering scheme can be precomputed on a CPU before being sent to the GPU. Simple statistics such as local means, whitening matrices and coloring matrices can be stored in a lookup table. For histogram exchange, a quantized CDF and inverse CDF for each region can be stored in a single 2D texture. Additionally, properly filtering the histogram exchange method of section 4.4 requires pre-filtering the quantized CDF and inverse CDF per region and per level of detail using a known filtering scheme for non-linear transfer functions [HNPN13, HN18]. However, the benefit of properly filtering histogram exchange is limited for two reasons: firstly, most real-life exemplars feature little variations in color within a single region, making local CDF and its filtered versions often close. Secondly, with our filtering scheme, the weight of histogram exchange rapidly fades compared to the weight of the substitute of section 5.3, thus not justifying the increased memory cost.

Manual filtering. A limitation of our GPU implementation lies with the texture indirections required for the computation of the value of each texel. As such, simple hardware filtering cannot be relied on despite being generally more efficient. In other words, bilinear, trilinear and anisotropic filtering all have to be implemented manually, using a statistics exchange call in place of the regular texel fetch of bilinear interpolation.

6. Results and discussion

We have implemented the pre-computation steps in C++/Qt and rendered a prototype of our synthesis in Godot [LMC14] using gdshader, an overlay of GLSL. More results, with their corresponding region maps and filtering comparison, along with a prototype implementation are available in the supplemental materials. Complete source code will also be released upon publication.

6.1. Comparison with stationary tiling and blending

We compare our synthesis with periodic tiling and the dual-square tiling and blending algorithm used in Lutz et al. [LSD23] in Figure 6. Overall, our synthesis avoids both disadvantages of tiling and

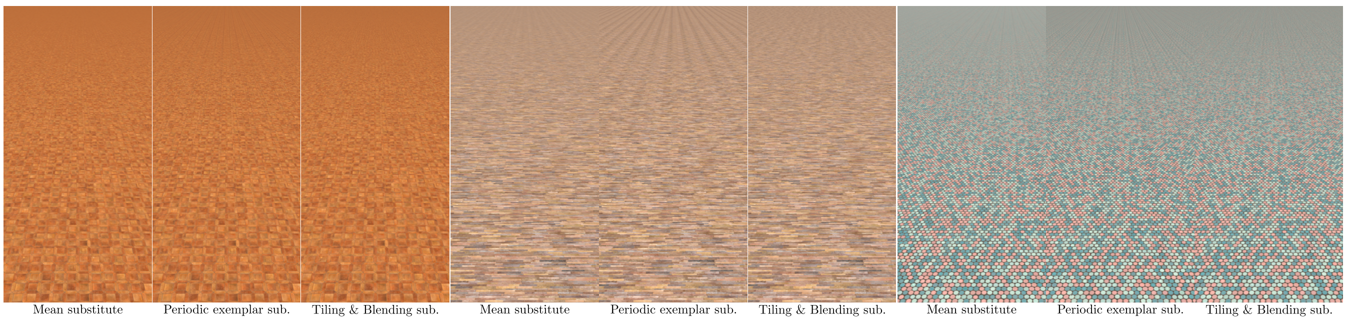

Figure 5: *Anisotropic filtering of PCA covariance exchange with three substitutes for filtering overlapping exchangeable regions proposed in section 5.3 (mean, periodic exemplar and tiling & blending). When viewing from far away or with a grazing angle, using the mean blurs the output; using the periodic exemplar creates obvious alignments; using tiling and blending neither blurs or create obvious alignments.*

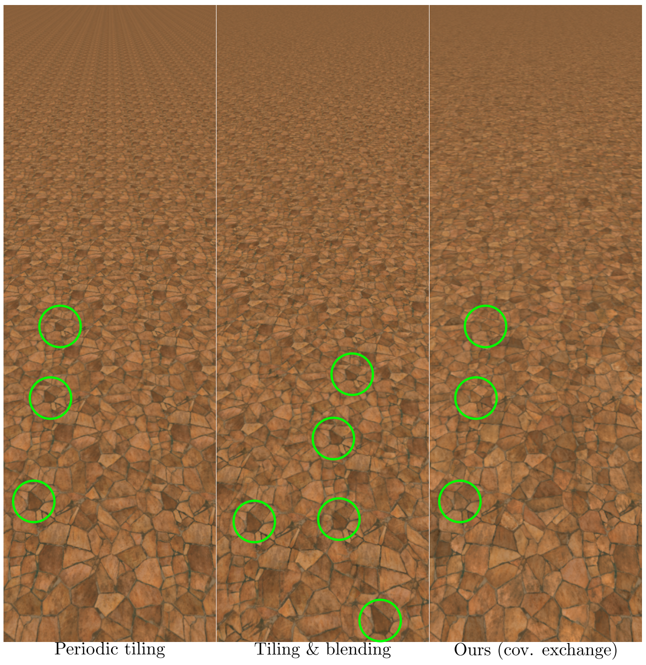

Figure 6: *Our synthesis using PCA covariance exchange (right) compared to the tiling and blending of Lutz et al. [LSD23] (middle) and periodic tiling (left), taking an exemplar with irregularly shaped regions. Tiling & blending does not align features (circled in green), but blends and repeats them randomly. Ours aligns features, but changes their values, avoiding visible repetitions.*

blending: cuts along features are avoided by repeating the structure, and repetitions are avoided by exchanging the values of features. This is valid as long as the background does not contain too many salient features (which we address in section 6.8 and there is enough variety in the different regions to avoid similar-looking regions (which we address in section 6.3).

6.2. Comparison between exchange methods

Our synthesis methods are ranked by increasing order of complexity, memory footprint and computation time. All methods use the same filtering scheme, and require one periodic access to the exemplar E, two additional random accesses to E, and one periodic access to the contribution map C and region map R (noting that C can be embedded as 2 additional channels of R).

We give comparisons for albedo, assuming 3 channels total. All exchange methods are efficient at removing repetition artifacts. We compare the results of each method in Figure 8 and highlight specific visual details over several exemplars. To provide a better understanding of each exchange, we provide a toy example to highlight the consequences of each method in Figure 7.

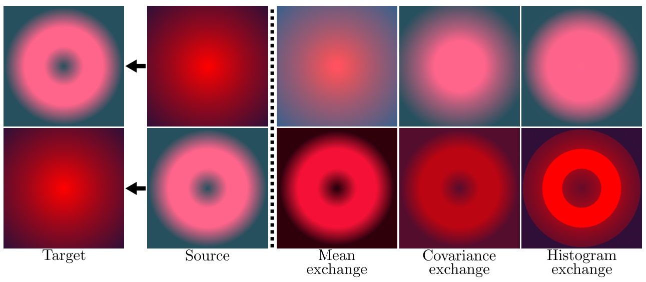

Figure 7: *Our synthesis applied on a synthetic case featuring 2 regions with different distributions: a bi-lobe whose brightest pink and darkest blue each take 25% of the region (top target), and a single lobe centered around a red hue (bottom target). Exchanging means produces false colors; exchanging covariances creates believable colors, though they do not quite match the target distribution; exchanging histograms creates visible cuts when mapping the single lobe onto the bi-lobe.*

Mean exchange is very fast, requiring an additional two accesses to a LUT storing the local means. However, it produces little variety when the distributions of different regions are too similar and false colors when they are too different, as the mean is too crude a representation of local first-order distributions.

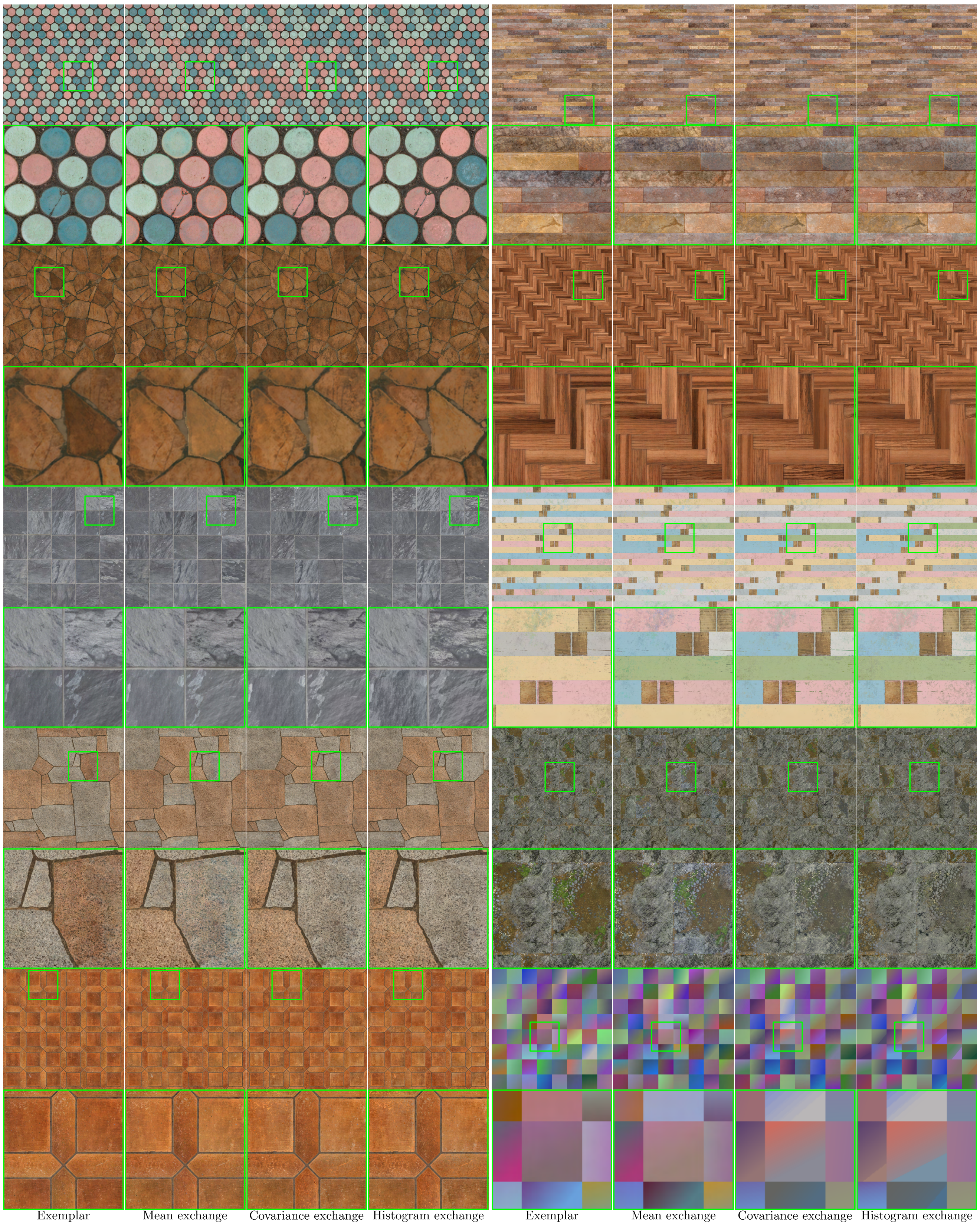

Figure 8: *Various results of the 3 versions of our synthesis with close-up views, using PCA whitening/coloring for covariance and histogram exchange.*

Covariance exchange is relatively fast, requiring an additional two accesses to a LUT storing the local whitening and coloring matrices. These accesses cost three texel fetches for PCA on albedo (to fetch a 3x3 matrix), and two texel fetches for ZCA on albedo (to fetch a 3x3 symmetrical matrix). Overall, covariance exchange generates plausible colors that match the target distribution at least in variance/covariance and mean. It still does not generally fit the distribution perfectly, meaning a recolored region can still feature colors that differ compared to the target.

Histogram exchange is the most expensive method we propose. It requires significantly more storage, requiring an additional two accesses to a LUT storing the forward and inverse equalizations, as well as an access to a LUT storing the minimum and maximum of each whitened region to normalize/de-normalize values before/after the specification. The specified region matches the first-order distribution of the target in theory. In practice, this does not necessarily lead to a better appearance or even a better overall match. In fact, large changes in the first-order distribution between the source and the target that do not take locality into account can lead to abrupt transitions between values in the output, as shown with a synthetic exemplar in Figure 7.

Overall and due to the flaws of histogram exchange, we consider PCA covariance exchange to be the best compromise between appearance and performance. ZCA is a viable alternative that is slightly less expensive and situationally yields better colors, but breaks easily should the distributions be unfavorably aligned with certain orthonormal axis.

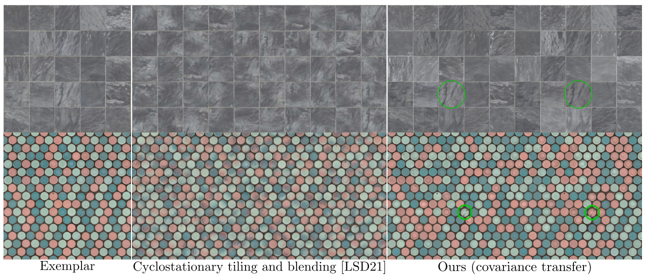
Exemplar
Cyclostationary tiling and blending [LSD21]
Ours (covariance transfer)

Figure 9: *Cyclostationary tiling and blending of Lutz et al. [LSD21] (middle) compared to ours (right). Cyclostationary tiling and blending does not align texture features such as local patterns (top) or texture flaws (bottom), circled in green in ours, but its underlying blendings can create undesired local variations.*

6.3. Comparison with cyclostationary tiling and blending

The cyclostationary tiling and blending algorithm of Lutz et al [LSD21] is capable of reproducing a periodic global organization in real-time for regularly shaped patterns, which form a sub-class of exemplars we are also able to synthesize. The main property of cyclostationary tiling and blending is that it naturally generates a high amount of variety thanks to the blending of randomly-chosen contents. This creates more variety than our algorithm, as the latter is bound to repeat features of the exemplar, including local patterns and texture flaws, as seen in Figure 9. The created variety can however be undesired, and tilings can break local patterns while blending operations can blend features together that do not match well, such as very different local colors.

6.4. Comparison with sliced Wasserstein loss synthesis

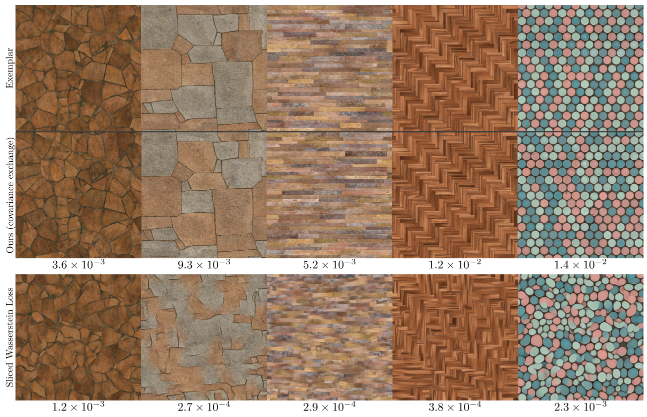

Figure 10: *Exemplars (top) synthesized with our method using PCA covariance exchange (middle) and the sliced Wasserstein loss texture synthesis network of Heitz et al. [HVCB21] (bottom), with loss value computed for each synthesized texture. Sliced Wasserstein loss synthesis consistently yields better loss than ours; however, regions bleed into each others.*

The Sliced Wasserstein Loss (SWL) texture synthesis of Heitz et al. [HVCB21] is an offline-only texture synthesis algorithm based on the optimization of a texture from a pre-trained network, using sliced Wasserstein distance as loss value. In Figure 10, we compared our synthesis to SWL synthesis, including the loss value computed from our own synthesis. While SWL synthesis optimizes textures with better loss values (from 2 to 50 times better), the different regions in the results often bleed into each other, creating unnatural textures. The difference in loss values could be explained by the fact that exchanging local statistics in our method does not guarantee an identical distribution of the statistics in the output, since region statistics that were present in the exemplar can disappear or be duplicated in the output. Regardless, SWL synthesis and other optimization methods are not able to synthesize textures in real time like ours.

6.5. Comparison between ground truth filtering and ours

Our filtering algorithm is efficient to filter any exemplar synthesized with our method in a real-time friendly way. Compared to Lutz et al. [LSLD19], it does not iterate over the different regions in a footprint. In terms of memory consumption, our method increases the memory footprint of the region map with two new channels instead of multiplying the entire memory consumption by 4.

We show the loss of accuracy, compared to a ground truth filtering, including the L2 error between the latter and our filtering, in Figure 11 and the supplementary material. There are two places where approximations occur in our filtering: where regions

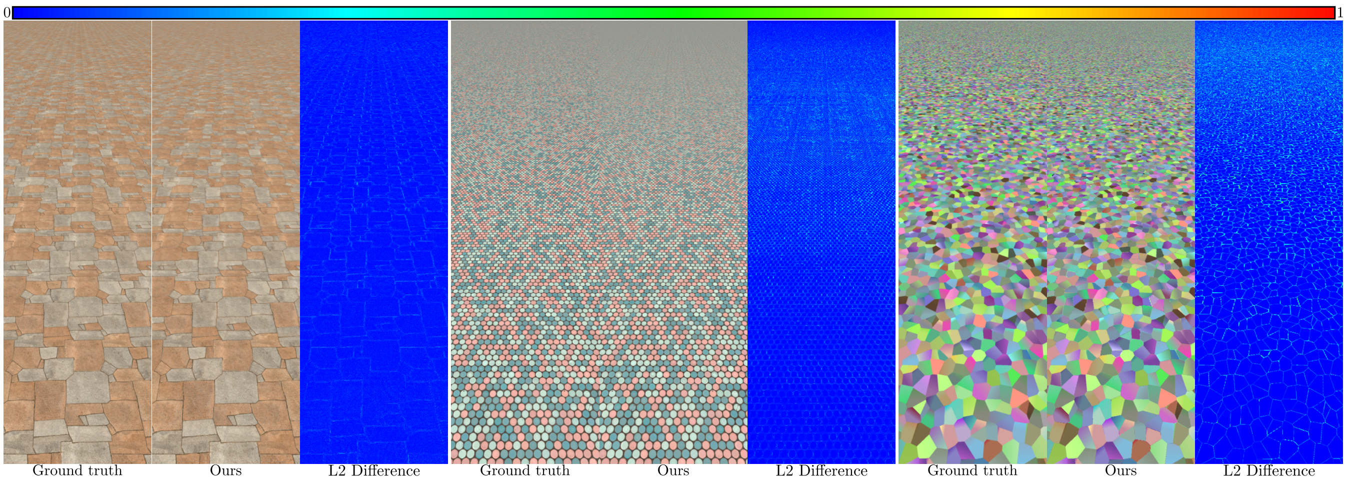

Figure 11: *Our synthesis using PCA covariance exchange, filtered with tiling and blending as substitute (see section 5.3) against a ground truth filtering. Exemplars used feature a small number of regions with a background, a high number of regions with a background, and a medium number of regions with no background. The approximation of our filtering near the background and when several regions blend together causes visible differences between the ground truth and ours. However, these differences remain low and fade at the horizon.*

are blended with the background, expressed as a mix between the periodic exemplar and a statistics exchange, and where regions are blended with each other, expressed as a completely different synthesis algorithm, the latter being the most incorrect. Due to this, errors appear essentially at the borders of each region, and are most important when there is little to no background and when there is a large amount of different colors. Despite these approximations, the L2 error remains low for most pixels.

6.6. Computation times

We have tested the computation times of our work using Godot 4.5 visual profiler [LMC14]. Table 6.6 shows average timings of our synthesis and filtering algorithm. As is common in texture synthesis, performance is linearly dependent on target resolution as most computation takes place in a per-fragment basis. We note that different substitutes do not significantly change computation time, as all of them only add one or two texture accesses. Using the ZCA whitening/coloring of section 3.2 is a little less expensive than using a PCA because the whitening and coloring matrices are less expensive to fetch. While covariance exchange is on average 30% to 35% more expensive than mean exchange (from about 25% to 40% in our data, increasing with the number of regions), histogram exchange is on average 3 times more expensive (from about 2 to 5 times in our data, increasing with the number of regions) due to the additional accesses in the LUT storing the forward and inverse local CDFs, making it not worth using it in most cases. Furthermore, we observe that the computational complexity and performance of all algorithms scale with the number of different regions. This may possibly be due to more frequent cache misses as the number of random accesses in different LUT increases.

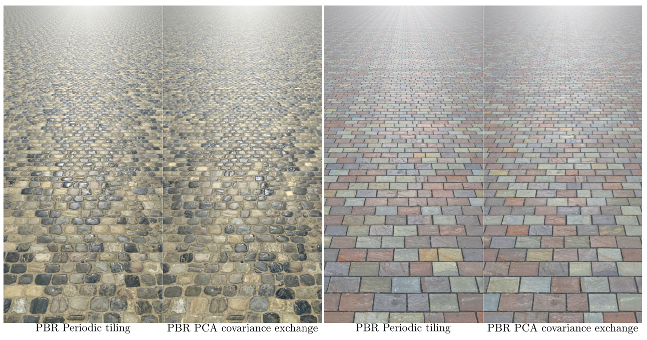

Figure 12: *Periodic tiling and our PCA covariance exchange using PBR maps. Both texture sets are made of albedo, normals, rugosity, height, and ambient occlusion, all of which are synthesized. Our synthesis allows to introduce variety for PBR maps which can be cut into individual regions.*

6.7. PBR textures

PBR textures are texture sets that include data for physically-based rendering, such as albedo, roughness, metallicity, normals, ambient occlusion, or height, which sometimes include variations of statistics between different regions. We have tested mean/covariance exchange for our synthesis. They are compatible with PBR textures, but with some drawbacks. Preserving covariances is expensive, as it would require to store whitening/coloring matrix pairs for a total of n^2 floats, where n is the cumulated number of texture canals. This issue can be circumvented by storing only local 3×3, such as one for albedo, normals, and rugosity+metallicity+ambient occlusion, or even using only one float for each scalar map; however,

Mean as substitute	**Mean exchange**	**PCA cov. exch.**	**ZCA cov. exch.**	**PCA hist. exch.**	**ZCA hist. exch.**
	0.95 ms	1.3 ms	1.25 ms	3.25 ms	3.00 ms
Exemplar as substitute	**Mean exchange**	**PCA cov. exch.**	**ZCA cov. exch.**	**PCA hist. exch.**	**ZCA hist. exch.**
	1.00 ms	1.35 ms	1.3 ms	3.25 ms	3.00 ms
Tiling & Blending as substitute	**Mean exchange**	**PCA cov. exch.**	**ZCA cov. exch.**	**PCA hist. exch.**	**ZCA hist. exch.**
	1.00 ms	1.35 ms	1.3 ms	3.25 ms	3.00 ms
Periodic tiling	0.25 ms				
Tiling and blending	0.30 ms				

Table 1: *Average rendering timings of all possible variants of our synthesis and filtering algorithm, on a 1920 $\times$ 1080 window and an RTX 4080 Super GPU, without pre-computing the whitened or equalized version of E. We used* $16\times$ *anisotropic filtering; histogram exchange variants use LUTs that are quantized to 256 values; exemplars have a size of* 1024^2*, and 8 different textures were counted in the results. Timings are rounded to the nearest* 0.05 *ms. While our synthesis and filtering algorithm is more expensive than tiling and blending, it produces better results for a new class of textures and its timings still remain real-time friendly.*

this does not enable to encompass full covariances. In theory, covariances are only perfectly correct when there are no differences of correlations between texture channels from one region to another. In practice, the mean carries most of the information anyway (e.g. in Figure 12, left, black pebbles always have low rugosity, which is carried by the mean from one region to another). The normal map in particular is especially difficult to perfectly synthesize: alongside a height map, normals are not generally correlated to heights, but rather to the gradients of the heights, which are not captured. Additionally, filtering the normal map requires LEAN-mapping [OB10]. Therefore, whitening and coloring should be done on the LEAN map instead, which is also correlated to the gradient of the height map rather than the height map itself.

Despite correlations issues, our method allows to create variety in order to reduce repetition artifacts when using PBR maps compatible with our synthesis, as shown in Figure 12.

6.8. Limits and perspectives

Alignment artifacts. The main flaw of our synthesis is that it only changes the statistics of the content of each local region, and not their shape, nor the content itself. In such a model, alignment artifacts are inevitable, but the actual amount of perceived alignments varies from one texture to another. Although it is difficult and beyond the scope of this paper to quantify the actual amount of artifacts perceived on a surface, in our experiments, some of the conditions that favor alignment artifacts are salient local patterns, especially those in the background; a high variety in the shapes of the regions; and a low variety of content values within different regions.

In Figure 13, we propose an experiment to reduce artifacts related to salient local patterns that consist in randomly reflecting half of the region contents in whitened space around the origin to double the variety of local patterns. However, it can generate an appearance that does not make sense when compared to the exemplar, and does not solve the two other issues causing alignment artifacts to appear. Our method could benefit from more advanced structural synthesis to add variations to the shape of each region, such as a generic sparse convolution structural process, as presented by Guehl et al. [GAD+20] to generate arbitrary shapes in real-time.

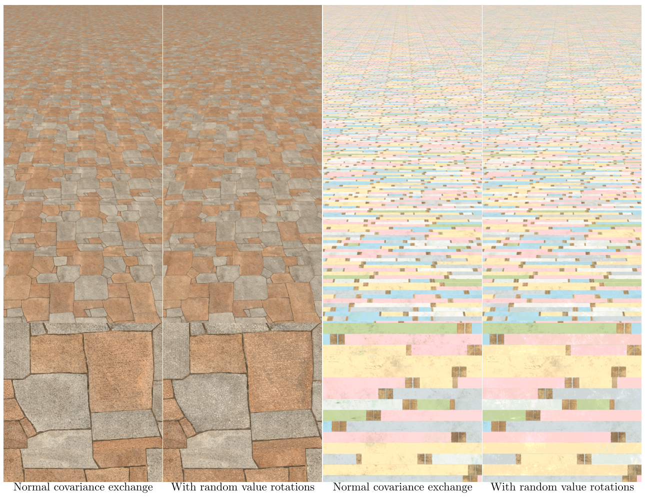
Normal covariance exchange
With random value rotations
Normal covariance exchange
With random value rotations

Figure 13: *In each view: covariance exchange (left) and covariance exchange with the improvement proposed in section 6.8. Each view shows a surface seen from afar (top) and a zoom up close (bottom). On the left view, white lines can be seen on the surface, while they are hidden with our improvement. On the right view however, alignment artifacts caused by salient elements in the background are still visible. On top of that, zooming in reveals potentially unwanted features, such as black pebbles and stains becoming white.*

Computation times. The computation times of our synthesis and filtering are adequate for real-time texturing scenarios, but still higher than tiling and blending methods. However, some processing steps may not always be mandatory: for instance, tiling and blending is only marginally used at high resolutions, so it could sometimes be cut out of the process. Mean exchange looks almost identical to the other methods when viewing the surface from far away, making more expensive methods less useful at low resolutions. With the same idea, the secondary components of the PCA could be discarded, especially at low resolutions, where their impact is lower. These improvements are beyond the scope of this paper, as it is always difficult to balance the global perceptual quality and the computational gains in an applicative context.

Multi-class statistics exchange. Our method exchanges statistics without considering different classes. For instance, in the rightmost exemplar of Figure 13, we did not exchange the small brown planks with other colored planks because they have unique sizes, hence exchanging them together would be too different from the exemplar; however, it would theoretically be possible to exchange the statistics of small brown planks together. This would lead to an increased variety for exemplars featuring similar cases.

Animated textures. In this study, we did not attempt to synthesize animated textures, which would be challenging. During the animation, if the size or content of regions change, the correct per-frame statistics exchange requires a pre-computation of its statistics again, requiring one set of pre-computed data per animation frame. Moreover, the result of statistics exchange would be different each frame, leading to a poor temporal stability.

Region map. The necessity of using a region map implies that the exemplar needs to be clearly cut into individual regions. However, this isn't always possible, as the border between regions may have a width of several texels. This issue is common in texture synthesis algorithms that segment the exemplar into regions [GSDC17]. In our case, this can lead to subtle border artifacts when the contrast of the contents between a source and target region is high. For histogram exchange in particular, including border texels in a region can lead to noise whose values come from these border texels, as can be seen in 3.

7. Conclusion

In this work, we have presented a novel real-time by example synthesis model which consists in exchanging local first-order statistics, including local means, local covariance matrices, and local histograms, using either PCA or ZCA whitening/coloring for the latter two. This method hides most repetition artifacts common in periodic tiling over large surfaces, while keeping good performance and high-quality renderings due to an efficient filtering scheme.

Acknowledgments

This work was funded by the University of Montpellier and the University of Sherbrooke. It was also partially funded by Ubisoft Divertissement in the context of a Mitacs Accelerate Fellowship. All textures are provided by either https://textures.com or https://ambientcg.com.

References

[Bur19] Brent Burley. On histogram-preserving blending for randomized texture tiling. *Journal of Computer Graphics Techniques (JCGT)*, 8(4):31–53, November 2019. 2

[Chi19] Tai-Yin Chiu. Understanding Generalized Whitening and Coloring Transform for Universal Style Transfer. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 4451–4459, Seoul, Korea (South), October 2019. IEEE. 3, 4

[CSHD03] Michael F. Cohen, Jonathan Shade, Stefan Hiller, and Oliver Deussen. Wang tiles for image and texture generation. *ACM Transactions on Graphics*, 22(3):287–294, 2003. 2

[DH18] Thomas Deliot and Eric Heitz. Procedural stochastic textures by tiling and blending. *GPU Zen 2: Advanced Rendering Techniques*, 2018. 2, 3

[Eom00] Kie B. Eom. Synthesis of Color Textures for Multimedia Applications. *Multimedia Tools and Applications*, 12(1):81–98, September 2000. 3

[GAD$^+$20] Pascal Guehl, Rémi Allegre, Jean-Michel Dischler, Bedrich Benes, and Eric Galin. Semi-procedural textures using point process texture basis functions. In *Computer Graphics Forum*, volume 39, pages 159–171. Wiley Online Library, 2020. 3, 12

[GGM11] Bruno Galerne, Yann Gousseau, and Jean-Michel Morel. Random phase textures: Theory and synthesis. *IEEE Transactions on Image Processing*, 20(1):257 – 267, 2011. 2

[GLLD12] Bruno Galerne, Ares Lagae, Sylvain Lefebvre, and George Drettakis. Gabor noise by example. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2012)*, 31(4):73:1–73:9, July 2012. 2

[GLM17] Bruno Galerne, Arthur Leclaire, and Lionel Moisan. Texton noise. *Computer Graphics Forum*, 36(8):205–218, 2017. 2

[GLR17] Bruno Galerne, Arthur Leclaire, and Julien Rabin. Semi-discrete optimal transport in patch space for enriching gaussian textures. In *Geometric Science of Information*, volume 10589 of *Lecture Notes in Computer Science*, Paris, France, November 2017. 3

[GSDC17] Geoffrey Guingo, Basile Sauvage, Jean-Michel Dischler, and Marie-Paule Cani. Bi-layer textures: a model for synthesis and deformation of composite textures. *Computer Graphics Forum*, 36(4):111–122, 2017. 3, 13

[GSDT22] Charline Grenier, Basile Sauvage, Jean-Michel Dischler, and Sylvain Thery. Color-mapped noise vector fields for generating procedural micro-patterns. In *Computer Graphics Forum*, volume 40, 2022. 4

[GSV$^+$14] Guillaume Gilet, Basile Sauvage, Kenneth Vanhoey, Jean-Michel Dischler, and Djamchid Ghazanfarpour. Local random-phase noise for procedural texturing. *ACM Transactions on Graphics*, 33(6):195:1–195:11, November 2014. 2

[HB95] David J Heeger and James R Bergen. Pyramid-Based Texture Analysis/Synthesis. *Proceedings., International Conference on Image Processing*, 3:648–651, 1995. 3

[HN18] Eric Heitz and Fabrice Neyret. High-performance by-example noise using a histogram-preserving blending operator. *Eurographics Symposium on High-Performance Graphics 2018*, 2018. 2, 3, 7

[HNPN13] Eric Heitz, Derek Nowrouzezahrai, Pierre Poulin, and Fabrice Neyret. Filtering non-linear transferfunctions on surfaces. *IEEE transactions on visualization and computer graphics*, 20(7):996–1008, 2013. 7

[HVCB21] Eric Heitz, Kenneth Vanhoey, Thomas Chambon, and Laurent Belcour. A sliced wasserstein loss for neural texture synthesis, 2021. 10

[LLC$^+$10] Ares Lagae, Sylvain Lefebvre, Rob Cook, Tony DeRose, George Drettakis, D.S. Ebert, J.P. Lewis, Ken Perlin, and Matthias Zwicker. State of the art in procedural noise functions. In *EG 2010 - State of the Art Reports*. Eurographics Association, May 2010. 2

[LMC14] Juan Linietsky, Ariel Manzur, and Godot Contributors. Godot engine, 2014. Open-source game engine under the MIT license. 7, 11

[LSD21] Nicolas Lutz, Basile Sauvage, and Jean-Michel Dischler. Cyclostationary Gaussian noise: theory and synthesis. In *Computer Graphics Forum*, Vienna, Austria, May 2021. (Proceedings of Eurographics 2021). 2, 3, 10

[LSD23] Nicolas Lutz, Basile Sauvage, and Jean-Michel Dischler. Preserving the autocovariance of texture tilings using importance sampling. In *Computer Graphics Forum*, volume 42, pages 347–358. Wiley Online Library, 2023. (Proceedings of Eurographics 2023). 2, 3, 7, 8

[LSLD19] Nicolas Lutz, Basile Sauvage, Frédéric Larue, and Jean-Michel Dischler. Anisotropic filtering for on-the-fly patch-based texturing. In *Eurographics 2019 Shorts*, Genoa, Italy, May 2019. 2, 3, 4, 6, 10

[OB10] Marc Olano and Dan Baker. Lean mapping. In *Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, I3D '10, page 181–188, New York, NY, USA, 2010. Association for Computing Machinery. 12

[RGH$^+$24] Nikhila Ravi, Valentin Gabeur, Yuan-Ting Hu, Ronghang Hu, Chaitanya Ryali, Tengyu Ma, Haitham Khedr, Roman Rädle, Chloe Rolland, Laura Gustafson, Eric Mintun, Junting Pan, Kalyan Vasudev Alwala, Nicolas Carion, Chao-Yuan Wu, Ross Girshick, Piotr Dollár, and Christoph Feichtenhofer. Sam 2: Segment anything in images and videos, 2024. 5

[VSLD13] Kenneth Vanhoey, Basile Sauvage, Frédéric Larue, and Jean-Michel Dischler. On-the-fly multi-scale infinite texturing from example. *Transactions on Graphics*, 32(6):208:1–208:10, 2013. (Proceedings of Siggraph Asia'13). 2, 3

[Wei04] Li-Yi Wei. Tile-based texture mapping on graphics hardware. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware*, HWWS '04, pages 55–63. ACM, 2004. 2, 3