Mix-Max: A Content-Aware Operator for Real-Time Texture Transitions

Romain Fournier, Basile Sauvage

DOI: 10.1111/cgf.15193

COMPUTER GRAPHICS *forum*
Volume 0 (2024), number 0, e15193

Mix-Max: A Content-Aware Operator for Real-Time Texture Transitions

Romain Fournier and Basile Sauvage

ICube UMR 7357, University of Strasbourg, CNRS, Strasbourg, France
{fournierr, sauvage}@unistra.fr

Abstract

Mixing textures is a basic and ubiquitous operation in data-driven algorithms for real-time texture generation and rendering. It is usually performed either by linear blending, or by cutting. We propose a new mixing operator which encompasses and extends both, creating more complex transitions that adapt to the texture's contents. Our mixing operator takes as input two or more textures along with two or more priority maps, which encode how the texture patterns should interact. The resulting mixed texture is defined pixel-wise by selecting the maximum of both priorities. We show that it integrates smoothly into two widespread applications: transition between two different textures, and texture synthesis that mixes pieces of the same texture. We provide constant-time and parallel evaluation of the resulting mix over square footprints of MIP-maps, making our operator suitable for real-time rendering. We also develop a micro-priority model, inspired by micro-geometry models in rendering, which represents sub-pixel priorities by a statistical distribution, and which allows for tuning between sharp cuts and smooth blend.

Keywords: mixing operator, filtering, image and video processing, antialiasing, real-time rendering, texture synthesis

CCS Concepts: • Computing methodologies → Texturing; Rendering; Antialiasing; Graphics processors

1. Introduction

In the field of computer graphics, textures enable the rendering of visually appealing and realistic environments. Mapping properties such as colours or normals on a surface enriches and refines its appearance. As materials are often not monolithic and exhibit spatial variations, transition between different materials are needed, as demonstrated in Figure 1. This paper investigates the local mix between multiple textures to allow for rich and realistic transitions.

In the context of real-time texturing, this problem is solved by either of two techniques. The cut technique superimposes the textures and optimizes globally for a sharp cut. It is efficient for structured materials like leaves or pebbles, but it may produce visible seams. On the other hand, the blending technique computes a progressive interpolation as a weighted average of the textures (Figure 2, middle). It produces nice fuzzy transitions for translucent materials like sand or dust scattered over a surface, but it is prone to ghosting artefacts for opaque materials. We argue that these two behaviours actually emerge from the same phenomenon, but looked at different scales: Looking closely, the elementary patterns (*e.g.* leaves or sand grains) overlap each others and the cut is sharp; looking from afar, a single pixel covers many elements from both textures and appears as an average. With this in mind, we build a mixing operator that is based on texture overlapping, and behaves coherently at all scales.

Another application of mixing operators is example-based texture synthesis. A family of real-time algorithms called 'tiling and blending' takes as input a small texture example, and generates a large resembling output by assembling pieces of the input. To assemble the pieces, called tiles, it is needed to transition between twice the same material. In this use case, a new challenge is to produce a stationary mix, *i.e.* which has the same statistical properties everywhere. Otherwise tiles' boundaries create a grid artefacts (Figure 4).

Aiming for these two applications (transition and synthesis) in real-time, a mixing operator must have three properties. (i) It must be *content-dependent*, *i.e.* the way the two textures are mixed must depend on the intensity values of the mixed pixels. The absence of this property accounts for most of the ghosting artefacts of 'content blind' techniques such as linear blending. (ii) It must come along with a constant-time algorithm for *low-pass filtering*. This is required to maintain high frame-rate while avoiding minification artefacts when zooming out. (iii) It must *preserve the stationarity* in

Figure 1: *Our texture mixing operator creates complex content-dependent transitions between textures. It easily replaces linear blending in real-time algorithms for transition, generation and rendering. Resolution of the mixed texture:* 10240×2048.

order to be used in texture synthesis algorithms. At the very least, the mean of the mix must equal the mean of the mixed textures.

Our first contribution (Section 3) is a mixing operator that meets the three above properties. It relies on *priority maps* which correlate to the texture's content (see greyscale Figure 2, top row). We call it Mix-Max operator because it selects at every texel the texture with maximum priority. It implements the intuition of textures that overlap in a complex and intertwined manner (Figure 2, bottom). We provide a filtering technique (Section 5) which approximates a MIP-map of the mix. We adapt our mixing operator to stationary texture synthesis (Section 4) and prove the invariance in space, not only of the mean but also of the whole distribution of intensities. We generalize our operator to more than two textures (Section 7), which makes it versatile.

Our second contribution (Section 6) is a statistical representation of micro-priority at the sub-pixel scale. The idea is similar to the micro-facet theory for rendering, which represents the micro-geometry by a statistical distribution. Within each texel, we represent variations of the priority maps by a Gaussian distribution characterized by a 'base variance'. It introduces point-wise blending between texels, and allows for a continuous tuning between sharp and fuzzy transitions. This micro-scale is coherent with the macro-scale, and the filtering formulas combine naturally.

We discuss additional results in Section 8 and show the flexibility of the model. We also demonstrate its ease of integration in the graphics pipeline and provide a parallel GPU implementation and we finally sketch out avenues for future research.

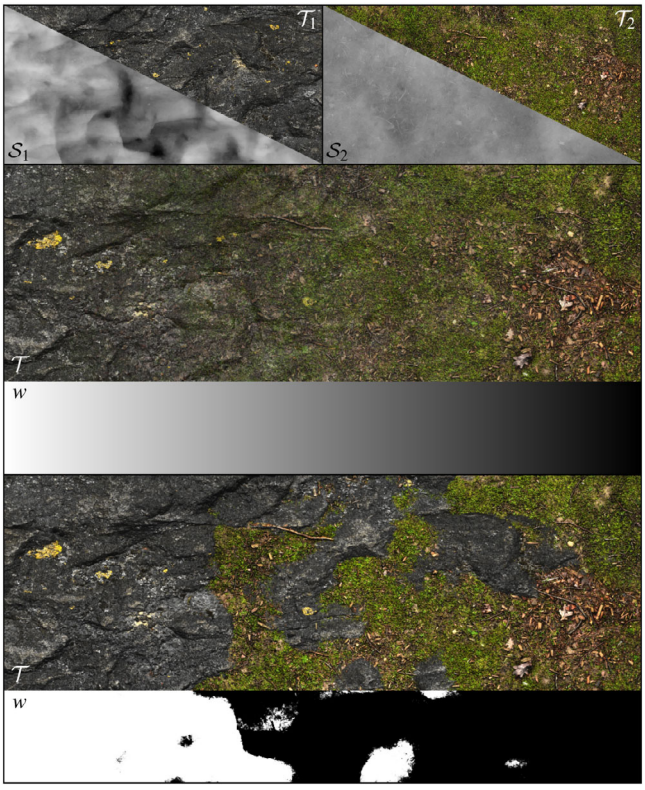
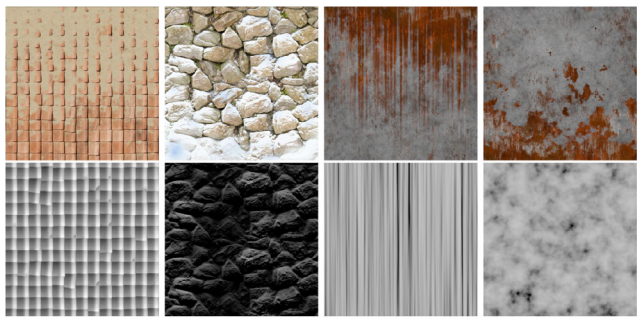
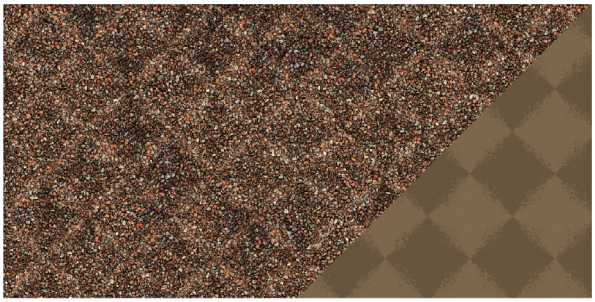


Figure 2: *Comparison between a linear interpolation (middle) and our mix-max operator (bottom); the greyscale bands represent the weight* w_1. *The top row shows the two input pairs (texture* $\mathcal{T}_i$, *priority* $\mathcal{S}_i$*).*

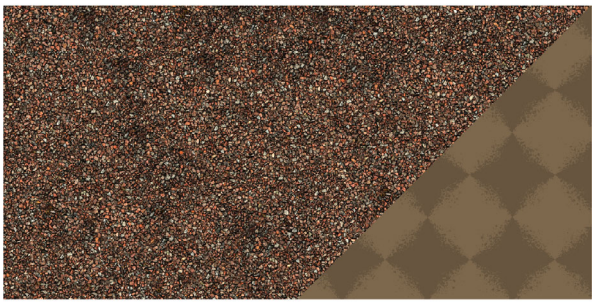


(a) *Height map* **(b)** *Normal map* **(c)** *User set* **(d)** *User set*

Figure 3: *Investigation of different priority maps (bottom) to model various transitions phenomena (top).*



(a) *same $\mathcal{S}_1$ and $\mathcal{S}_2$, the synthesis is not stationary*



(b) *opposite $\mathcal{S}_1$ and $\mathcal{S}_2$, the synthesis is stationary*

Figure 4: *Stationary texture synthesis using dual tiling and blending [LSD23]. (a) All tiles have the same priority map, a bright grid artefact appears. (b) Every second tile has an opposite priority map, the artefact disappears.*

2. Related Work

Composite images or textures. Refers to a specific type of spatial variations: The texture is composed of several sub-textures that partition the texture plane. The most current generation technique consists in synthesizing the sub-textures separately and computing transitions at the boundaries. The transition problem has already been well addressed [PD84, ZFCVG05, DSB*12, WDK*16, HVCB21] but remains a challenge in the context of real-time generation [LSA*16, GSDC17]. For real-time, the most popular transition method remains linear blending, as it is easy to compute in parallel and to filter. However, as it is content blind, it is prone to ghosting artefacts.

Content-dependent mixing has been proposed by Hardy and Mc Roberts [HMR06] in the context of texture transitions for terrains, in order to reduce ghosting artefacts. Our 'priority maps' are inspired from their 'blend maps' but our mixing formula is different. We overcome two limitations of their work: a biased filtering (Section 5) and a lack of stationarity (Section 4).

Texture interpolation. Is a problem close to but different from transition. It consists in defining a homogeneous appearance between two textures: two grasses with different tints, or pebbles with different sizes. Xia *et al.* [XFPA14] propose interpolation for unstructured Gaussian textures. Structured patterns are more difficult to handle. Different techniques rely on optimization of patches or of texture coordinates [RSK10, DSB*12, PBK13], or on neural networks [VDKCC20, HVCB21], which are not designed for real-time applications. These approaches can be used to compute transitions, but they produce progressive morphs of the local patterns [RLW*09, VDKCC20], which is different from our goal of intertwining and overlapping of two materials.

Example-based texture synthesis. Refers to a family of algorithms that take as input a (small) texture example, and produce as output a larger and similar texture. Among the many techniques [WLKT09], we are concerned by those that re-arrange and assemble pieces of the input.

Linear blending of small pieces of texture has been used for data-driven texture synthesis [KEBK05, KNL*15] and texture interpolation [DSB*12] using global optimizations, which are offline techniques. Parallel optimization has been used to synthesize non-homogeneous textures [PBK13], reaching interactive computation times for medium size textures. In comparison, our technique reaches real-time for very-high resolution textures, because it is fully parallel and the filtering is constant-time. In our results, we show real-time rendering of a landscape with millimetre-scale texturing.

Computing optimal cuts between large pieces of textures is an other approach [EF01, KSE*03], which has been used successfully to pre-compute tiles [CSHD03] and for offline synthesis. It is possible to reach real-time synthesis by pre-computing the cuts [VSLD13, KCD15, LSLD19], but the cuts are more visible. In comparison, our approach computes much more complex transitions (we are not limited to a single cut), is compliant with local blending (see, for example, Figure 9) and is real-time.

Closer to our real-time interests are the tiling and blending techniques [HN18, DH19, LSD23], which assemble regular tiles, copied in the texture example at a random position. Neighbouring tiles overlap and are blended together. It has been first designed for Gaussian textures, which is a pretty narrow range of unstructured textures. An additional histogram correction improves the method for non Gaussian textures, but ghosting artefacts appear because the blending does not depend on the texture's content. Burley reduced ghosting by narrowing the blending zone [BS19], but a boundary between tiles become visible. This motivated Mikkelsen [Mik22] to propose a content-aware mixing operator, and to bypass the histogram correction. He uses the luminance as a priority criterion for blending, but the synthesis is not stationary anymore, as we show in Section 4, and the filtering becomes biased. Similarly, Schuster *et al.* [STSK20] use the height map as a priority criterion. This approach is also subject to stationarity problems and filtering bias.

Our method is inspired by these works [HMR06, STSK20, Mik22]. However, by clearly separating the priority criterion and the mixing formula, our approach is more versatile, it preserves the stationarity, and unbiased filtering is possible.

3. Mix-Max Operator

3.1. Notations

Let $\mathcal{T}$ be a texture

$$\mathcal{T} : \mathbf{u} = \begin{bmatrix} u \\ v \end{bmatrix} \mapsto \mathbf{t} \tag{1}$$

that associates a value $\mathbf{t}$ to every position $\mathbf{u}$. The value $\mathbf{t}$ may be for instance an intensity, a colour or a normal.

A priority map

$$\mathcal{S} : \mathbf{u} \mapsto s \in [-1; 1] \tag{2}$$

associates to every position $\mathbf{u}$ a scalar value s, which will encode to what extent the texel has priority in a mixing process.

The problem of transitioning between two textures $\mathcal{T}_1$ and $\mathcal{T}_2$ is stated as follows. We are given two interpolation fields $v_1(\mathbf{u})$ and $v_2(\mathbf{u})$ with values in [0; 1] and $v_1 + v_2 = 1$. We aim to define a texture $\mathcal{T}$ which equals $\mathcal{T}_1$ where $v_1 = 1$, equals $\mathcal{T}_2$ where $v_2 = 1$, and has an in-between appearance for other (v_1, v_2) pairs. It is illustrated Figure 2 with trivial linear functions v_1 and v_2.

3.2. Mixing

We define the resulting texture

$$\mathcal{T}(\mathbf{u}) = w_1(\mathbf{u})\mathcal{T}_1(\mathbf{u}) + w_2(\mathbf{u})\mathcal{T}_2(\mathbf{u}) \tag{3}$$

by a linear combination, weighted by two fields w_1 and w_2. The very common blending fits this formula with $w_i = v_i$. This operator is 'blind' in the sense that it is independent of the texture content. The result is shown in Figure 2 middle: the impression of uniform transparency is not realistic for textures that represent opaque materials.

We propose new weighting fields, which are content-aware, and which represent overlapping. We define

$$w_1(\mathbf{u}) = \begin{cases} 1 & \text{if } \mathcal{S}_1(\mathbf{u}) + v_1(\mathbf{u}) > \mathcal{S}_2(\mathbf{u}) + v_2(\mathbf{u}) \\ 0 & \text{else} \end{cases} \qquad w_2(\mathbf{u}) = 1 - w_1(\mathbf{u}) \tag{4}$$

where $\mathcal{S}_1$ and $\mathcal{S}_2$ are priority maps correlated to $\mathcal{T}_1$ and $\mathcal{T}_2$ (see Figure 2, top). The weights are binary, so that Equation (3) boils down to choosing between $\mathcal{T}_1(\mathbf{u})$ and $\mathcal{T}_2(\mathbf{u})$, the texel with the maximum $S_i + v_i$ value. This is why it appears as overlapping, not as transparency. As shown in Figure 2 bottom, this leads to a richer and more natural transition, provided that the priority maps are defined wisely. The intuition of Equation (4) is that, for each position $\mathbf{u}$, the interpolation values v_1 and v_2 represent target *probabilities* of getting $\mathcal{T}_1$ or $\mathcal{T}_2$, whereas in the blind blending, it represents the *proportions* of $\mathcal{T}_1$ and $\mathcal{T}_2$. The priority maps $\mathcal{S}_1$ and $\mathcal{S}_2$ bias the probabilities, making it possible to adapt to the texture's content. Without loss of generality, we consider $\mathcal{S}_1$ and $\mathcal{S}_2$ to be centred on 0, so that the mixing is globally not biased towards one of the exemplars.

3.3. Priority maps

By associating coherent priorities with the texture patterns, it is possible to overlap two textures while preserving the visual elements. For example, in Figure 1, the pebbles are delineated, and the sand infiltrates between bricks and under the grass. This enables the representation of layering arising from phenomena such as growth, weathering or scattering.

Today's popular PBR materials come with a multitude of maps to represent various properties of a material. The height map is a good candidate for defining the priority because it captures geometry: the higher the more probable to overlap (Figure 3a). The normal map can be used to model direction-dependent phenomena, like snow on upward-pointing surfaces (Figure 3b). The priorities can also be set by the user, so as to tune the transition: for instance, we emulated rust streaks (Figure 3c) or rust patches (Figure 3d) by using anisotropic (respectively, isotropic) noises as priority maps.

4. Texture Synthesis

In the previous section, we presented the Mix-Max operator for a transition between two *different* textures. We now investigate the case where $\mathcal{T}_1$ and $\mathcal{T}_2$ come from the *same* texture. This is useful for real-time texture synthesis by tiling and blending [HN18, LSD23]. In these algorithms, the texture space is covered by overlapping tiles. The content of the tiles is sampled on-demand at rendering, randomly drawn from an example texture, and at every position $\mathbf{u}$, two (or more) overlapping tiles are mixed.

The challenge in this context is to enforce stationarity: the generated texture $\mathcal{T}$ is regarded as a stochastic field, and $T = \mathcal{T}(\mathbf{u})$ is regarded as a random variable. It is mandatory for the field to be stationary, *i.e.* for the probability density function of T to be independent of the position $\mathbf{u}$, otherwise the tiling creates a grid artefact (see Figure 4, top). For simplicity, we explain for a scalar texture, *i.e.* $\mathbf{t}$ is a scalar intensity denoted as t. We denote the random variables $T_1 = \mathcal{T}_1(\mathbf{u})$ and $T_2 = \mathcal{T}_2(\mathbf{u})$. We have to enforce that the cumulative distribution functions (CDF)

$$Proba(T \leq t) = Proba(T_1 \leq t) = Proba(T_2 \leq t). \tag{5}$$

are equal and do not depend on $\mathbf{u}$. This is true for T_1 and T_2 because the tiles $\mathcal{T}_1$ and $\mathcal{T}_2$ are randomly sampled in a stationary example [LSD23]. However, because of the operator (4) and the correlation between $\mathcal{S}_i$ and $\mathcal{T}_i$, the first equality is not true in general: intensities correlated to high priorities are over-represented in the synthesis. We demonstrate, in the supplemental document, that this can be solved by setting $\mathcal{S}_2$ equal to the opposite of $\mathcal{S}_1$ (instead of equal to $\mathcal{S}_1$). The intuition is that this makes the conditional distribution of priorities symmetric with respect to 0. In practice, we do not modify the priority map itself, but replace the condition in the operator (4) by $\mathcal{S}_1(\mathbf{u}) + v_1(\mathbf{u}) > -\mathcal{S}_2(\mathbf{u}) + v_2(\mathbf{u})$.

Figure 4 shows the result of the dual tiling and blending [LSD23] using our mix-max operator. The grid artefact disappears and the stationarity is enforced. More precisely, the figure shows that *the mean* is stationary. Equation (5) states that the stochastic field is first-order stationary, *i.e. the whole distribution* of intensities is invariant in space. To validate this experimentally, we adopt the same strategy as Lutz *et al.* [LSD21]. They observe cyclic statistics by computing histograms of polyphase components, which estimate pointwise intensity distributions. In our case, the statistics are not cyclic, but they depend on the interpolation value v_1, so we compute histograms $h(T|v_1)$ for fixed v_1. We achieve this by computing a transition with a linear ramp v_1 from 0 to 1; this is done many times for the same example but different random seeds; for each v_1, we compute an histogram $h(T|v_1)$ by collecting the intensities T from all realizations. Figure 5 shows the results for the brick wall

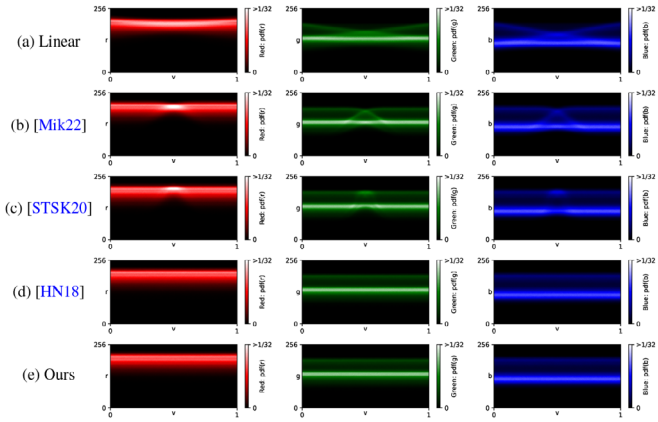


Figure 5: *Experimental validation of the first-order stationarity of the synthesis. For each RGB channel, the histograms $h(T|v_1)$ are plotted vertically: the abscissa is the interpolation value v_1; the ordinate is the intensity T. Interpretation: we expect $h(T|v_1)$ not to depend on v_1,* i.e. *the vertical slices must be equal. v_1 is sampled with 512 values from 0 to 1, and each histogram is build from* 102, 400 *realizations of synthesis on the brick wall example.*

example, with v_1 in abscissa and T in ordinate. We can observe on (e) that the vertical slices are identical, *i.e.* that the histograms $h(T|v_1)$ are identical for all v_1, so our model is first-order stationary. On the other hand, both linear blending (a) and the technique of Mikkelsen [Mik22] (b) vary across v_1. The variance-preserving technique of Heitz *et al.* [HN18] (d) needs special attention. Its result is stationary, but it operates only on Gaussian fields, so here it requires an extra histogram correction. This is pretty tough to implement in the shader and to filter [DH19], which motivates Mikkelsen to bypass the histogram correction. The method of Schuster *et al.* [STSK20] (c) vary across v_1 despite a global histogram correction, because a correlation between the textures and the blending weights introduces a bias.

We further discuss the results in Section 8 and Figure 14.

5. Real-Time Filtering

In this section, we present a real-time filtering method, which is required for our technique to be applied in real-time algorithms, whether for transitions (Section 3) or for synthesis (Section 4).

We rely on usual MIP-maps, as shown in Figure 6. Let $\mathbb{P}$ denote the footprint of a texel in the MIP-map, *i.e.* a square area with the side length equal to a power of two. The challenge consists in evaluating, in constant time, the average of $\mathcal{T}$ over any of these footprints.

The ground truth. Is the average over the footprint $\mathbb{P}$, given by the integral

$$\mathcal{T}(\mathbb{P}) = \frac{1}{area(\mathbb{P})} \int_{\mathbb{P}} \mathcal{T}(\mathbf{u}) \mathrm{d}\mathbf{u}. \tag{6}$$

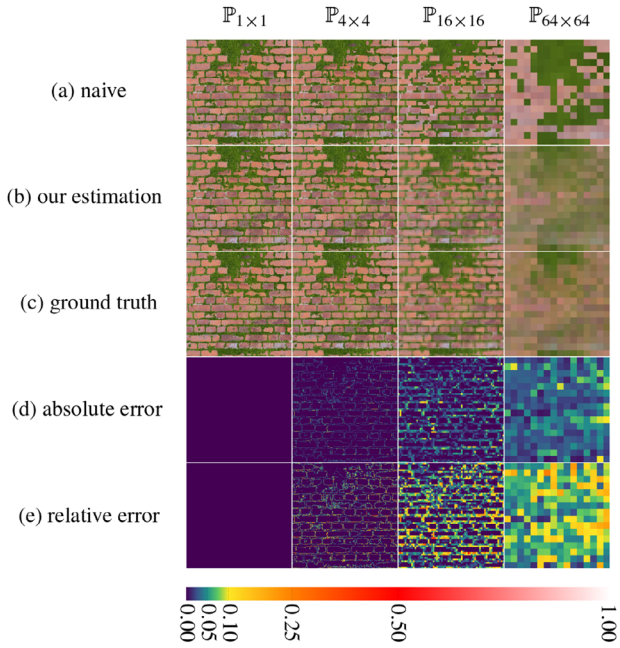


Figure 6: *Filtering over square footprints up to* 64×64 *texels. (a) Naive filtering (Equation 8). (b) Our estimation (Equation 10). (c) Ground truth (Equation 6). (d), (e) Absolute and relative error of our estimator versus the ground truth.*

The direct evaluation of this integral (a discrete sum over the $\mathbf{u} \in \mathbb{P}$) is impossible in real-time because its complexity grows linearly with the number of texels in $\mathbb{P}$.

To overcome this problem, we draw inspiration in previous work for Gaussian textures [DH19, GSDT22], and approximate

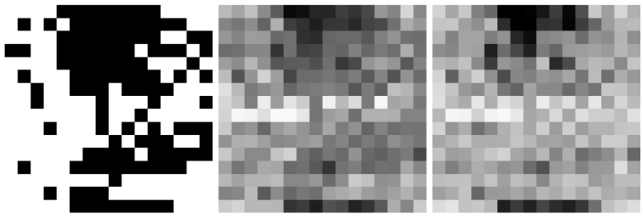
$$\mathcal{T}(\mathbb{P}) \approx w_1(\mathbb{P})\mathcal{T}_1(\mathbb{P}) + w_2(\mathbb{P})\mathcal{T}_2(\mathbb{P}) \tag{7}$$

as a weighted sum of $\mathcal{T}_i(\mathbb{P})$ which are pre-computed MIP-maps. This approximation assumes that w_i and $\mathcal{T}_i$ are not correlated: though this is not exact in our case, it is pretty effective, as shown below. The key now is to derive a formula for $w_i(\mathbb{P})$, which can be estimated in constant time.

A naive filtering. Consists in applying the same formulas

$$w_1(\mathbb{P}) = \begin{cases} 1 & \text{if } \mathcal{S}_1(\mathbb{P}) + v_1(\mathbb{P}) > \pm\mathcal{S}_2(\mathbb{P}) + v_2(\mathbb{P}) \\ 0 & \text{else} \end{cases}$$
$$w_2(\mathbb{P}) = 1 - w_1(\mathbb{P}) \tag{8}$$

directly on the footprints $\mathbb{P}$, using MIP-maps of $\mathcal{S}_i$ and v_i. For methods resorting to content blind blending, this approximation is often close to the ground truth [DH19, GSDT22] as the weighting fields w_i are quasi-constant over $\mathbb{P}$ and are independent of $\mathcal{T}_i$. Thus we compare ourselves against this naive filtering, considering it as the baseline. As can be seen in Figure 6 top, the result is far from the ground truth. This is due to our binary weights $w_i(\mathbf{u})$ which vary quickly within $\mathbb{P}$, and to the complex relationships with the priority maps.



(a) *naive Eq. (8)* **(b)** *ours Eq. (10)* **(c)** *ground truth*

Figure 7: *Comparison of the weight w_1 for filtering, corresponding to $\mathbb{P}_{64\times64}$ in Figure 6. The reference (c) is the actual average of w_1 over $\mathbb{P}$. Compared to naive binary weights (a), our constant time approximation (b) is nuanced, and close to the reference.*

Our constant time approximation. Is based on the idea that $w_1(\mathbb{P})$—and similarly for w_2—should equal the average of w_1 over $\mathbb{P}$, which we approximate by the probability

$$Proba(S_1(\mathbf{u}) + v_1(\mathbf{u}) > \pm S_2(\mathbf{u}) + v_2(\mathbf{u}) | \mathbf{u} \in \mathbb{P}) \tag{9}$$

that $\mathcal{T}_1$ is selected by the mix-max operator within $\mathbb{P}$. To estimate Equation (9), we assume that the priorities $\mathcal{S}_i$ have a Gaussian distribution over $\mathbb{P}$ and that the interpolation fields v_i are constant over $\mathbb{P}$. All that remains is to evaluate a probability of the form $P(X > Y)$. Downton [Dow73] provides formulas that we apply to estimate the probability of $X = S_1 + v_1$ to exceed $Y = S_2 + v_2$, and obtain

$$w_1(\mathbb{P}) = 1 - \Phi\left(\frac{(\mathcal{S}_2(\mathbb{P}) + v_2(\mathbb{P})) - (\mathcal{S}_1(\mathbb{P}) + v_1(\mathbb{P}))}{\sqrt{\sigma_1^2 + \sigma_2^2}}\right) \tag{10}$$
$$w_2(\mathbb{P}) = 1 - w_1(\mathbb{P})$$

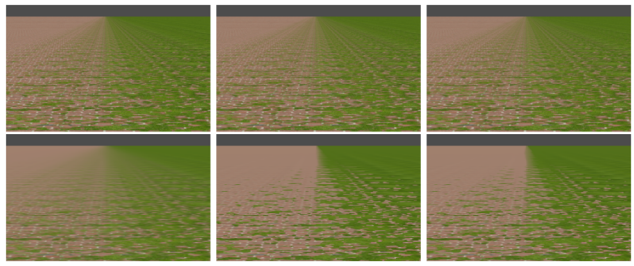
where Φ is the CDF of the normal standard distribution, and $\mathcal{S}_i(\mathbb{P})$ and σ_i^2 are, respectively, the mean and variance of $\mathcal{S}_i$ over $\mathbb{P}$.

In practice to evaluate the variance σ_i^2, we use a method similar to that of Olano and Baker [OB10] or Grenier *et al.* [GSDT22]. We store the priorities $\mathcal{S}_i(\mathbf{u})$ and the priorities squared $\mathcal{S}_i^2(\mathbf{u})$ in two textures. We compute MIP-maps $\mathcal{S}_i(\mathbb{P})$ and $\mathcal{S}_i^2(\mathbb{P})$ which store the first two moments for any footprint $\mathbb{P}$, and use it to compute the variance

$$\sigma_i^2 = \mathcal{S}_i^2(\mathbb{P}) - \mathcal{S}_i(\mathbb{P})^2. \tag{11}$$

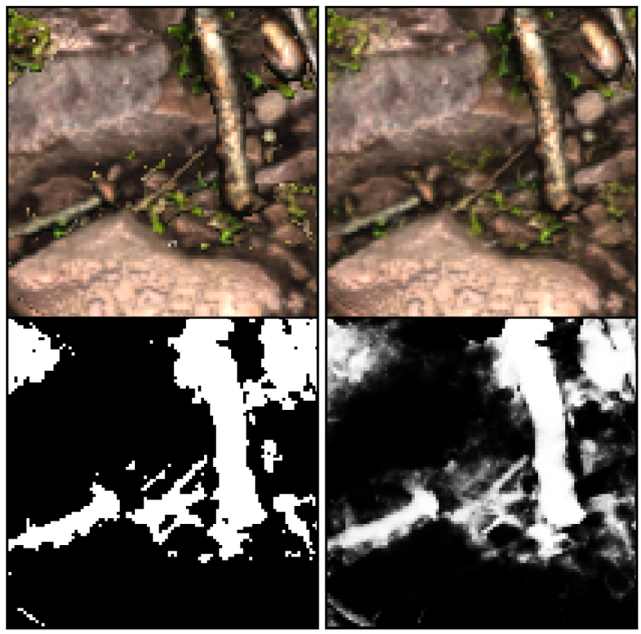
Figure 7 shows that our non-binary weights (10) are close to the actual average of binary weights. The result, shown in the second row Figure 6, is much more faithful to the ground truth than the naive filtering. In practice, we measure low though not negligible errors (bottom rows Figure 6). More importantly, the estimation is stable as the camera moves, thus avoiding flickering artefacts.

We now compare, in Figure 8, our technique to the blend maps of Hardy and Mc Roberts [HMR06] and to Schuster *et al.* [STSK20]. They faced the same transparency artefacts as we do, and solved the problem in a similar way than the above naive filtering: they use the averages $\mathcal{S}_i(\mathbb{P})$ but neglect the variations within $\mathbb{P}$ that we capture in the σ_i. Their filtering (Figures 8b and 8c) succeeds in avoiding transparency artefacts, however it is biased for large footprints and large transitions. The transitions (bottom row) are too sharp compared to their ground truth (top row), while our technique produces more faithful mixes.



(a) *Ours* **(b)** *[HMR06]* **(c)** *[STSK20]*

Figure 8: *Comparison between (a) our filtering, (b) the blend maps [HMR06] and (c) the max-blending [STSK20]. Top row: ground truths generated by super-sampling. Bottom row: real-time approximation.*



(a) *Without micro-priority* **(b)** *With* $\lambda_1 = \lambda_2 = 0.014$

Figure 9: *Micro-priority: result $\mathcal{T}$ (top row) and weight w_1 (bottom row). Introducing a sub-pixel 'base variance' λ_i of the priorities (b—Equation 12) softens the sharp cuts (a—Equation 4) due to binary weights.*

6. Micro-Priority and Sub-Pixel Blending

In the previous section, we addressed minification artefacts. We now address magnification artefacts. Because our weights w_i are binary, close-ups exhibit sharp cuts (Figure 9 left) at the boundary between $\mathcal{T}_1$ and $\mathcal{T}_2$, which may look unnatural. We fix this by coherently extending our model to a statistical representation of micro-priority at sub-pixel scale. Our approach is very similar to the micro-facets in rendering, which represents the micro-geometry by a probability distribution at very fine scales. The values (normal, colour, *etc.*) of every single texel emerge from (unknown) sub-pixel variations, which can be represented in a richer way than only its average value.

We model sub-pixel variations of the priority as a probability distribution. We chose a normal distribution: $\mathcal{S}_i(\mathbf{u})$ is the mean micro-priority within $\mathbf{u}$, and we add a variance λ_i^2. This 'base variance' is similar to the roughness in micro-facets theory. Then Equation (9) can be applied to a single texel, and Equation (4) becomes

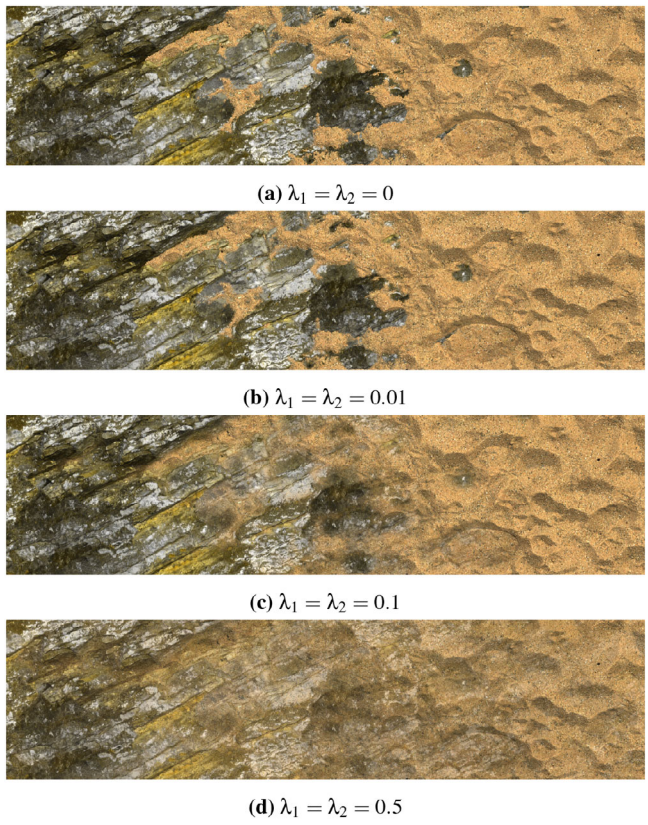
$$w_1(\mathbf{u}) = 1 - \Phi\left(\frac{(\mathcal{S}_2(\mathbf{u}) + v_2(\mathbf{u})) - (\mathcal{S}_1(\mathbf{u}) + v_1(\mathbf{u}))}{\sqrt{\lambda_1^2 + \lambda_2^2}}\right) \quad (12)$$

which combines naturally with Equation (10) into

$$w_1(\mathbb{P}) = 1 - \Phi\left(\frac{(\mathcal{S}_2(\mathbb{P}) + v_2(\mathbb{P})) - (\mathcal{S}_1(\mathbb{P}) + v_1(\mathbb{P}))}{\sqrt{\sigma_1^2 + \sigma_2^2 + \lambda_1^2 + \lambda_2^2}}\right). \quad (13)$$

We observe the coherence of these equations: Equation (12) is a special case of Equation (13) for a single texel, so $\sigma_i = 0$; Equation (4) is the limit of Equation (13) as the base variances $\lambda_i \to 0$.

The right column in Figure 9 shows the ability of the micro-priority to soften the sharp cuts, while preserving the complex intertwining of the patterns. We observe that it does not act as a uniform blur: the transition becomes soft at some places and remains sharp at some other places. This is an interesting property which relies on the fact that the mix does not depend on a distance to the cut in space $\mathbf{u}$, but on a distance between $(\mathcal{S}_1 + v_1)$ and $(\mathcal{S}_2 + v_2)$ at a fixed $\mathbf{u}$. Figure 10 shows how the base variance can be used to continuously tune the transition from sharp to fuzzy. In our experiments, $0.01 \leq \lambda_i \leq 0.1$ give visually pleasing results on most cases. One can observe the complementary roles of v_i (effect on the width of transition), $\mathcal{S}_i$ (effect on the shape of the cuts) and λ_i (effect on fuzzyness), though the interactions between these three parameters are complex.



(a) $\lambda_1 = \lambda_2 = 0$

(b) $\lambda_1 = \lambda_2 = 0.01$

(c) $\lambda_1 = \lambda_2 = 0.1$

(d) $\lambda_1 = \lambda_2 = 0.5$

Figure 10: *Micro-priority. The appearance of the transition is tuned by the base variances λ_i. Low variances (top) generate sharp cuts with complex shapes. High variances (bottom) generate fuzzy transitions, while still being content-dependent.*

7. N-Way Mixing

In this section, we generalize the operator to $N > 2$ textures, which is useful in practical applications. For example, the original tiling and blending [HN18] mixes three tiles. One may also want to transition between several textures as in Figure 11. Lastly, synthesis *and* transition can be combined to produce richer appearances, as for terrains in Figure 16.

To generalize our operator to an arbitrary number N of textures, we want to apply it recursively, like a reduction operator on a list of textures. For example, $MixMax(A, B, C) = MixMax(MixMax(A, B), C)$ for $N = 3$ textures. To do so, we need the operator to have the same inputs and outputs:

$$\begin{bmatrix} T(\mathbb{P}) \\ \mu \\ \omega^2 \end{bmatrix} = MixMax\left(\begin{bmatrix} T_1(\mathbb{P}) \\ \mu_1 \\ \omega_1^2 \end{bmatrix}, \begin{bmatrix} T_2(\mathbb{P}) \\ \mu_2 \\ \omega_2^2 \end{bmatrix}\right) \quad (14)$$

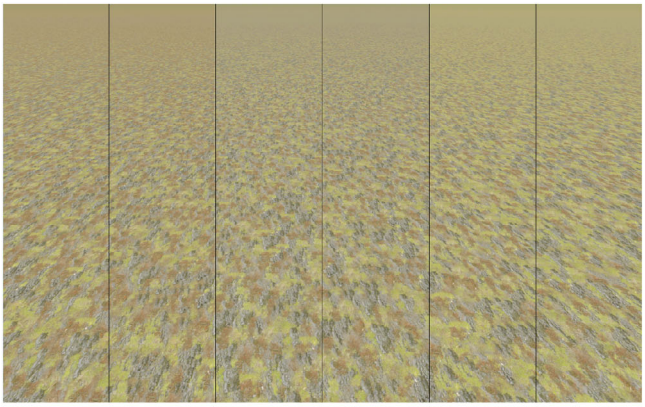
where $\mu_i = \mathcal{S}_i(\mathbb{P}) + v_i(\mathbb{P})$ is the biased priority mean, and $\omega_i^2 = \sigma_i^2 + \lambda_i^2$ is the priority variance. For two textures, the only output we needed was $\mathcal{T}(\mathbb{P})$, which is defined by Equations (7) and (13). Let us now define the outputs μ and ω^2.

A straightforward approach would be to linearly interpolate μ and ω^2, *i.e.* replacing $\mathcal{T}$ by μ and ω^2 in Equation (7). However, we assumed that w_i and $\mathcal{T}_i$ were uncorrelated, which is not true for w_i and

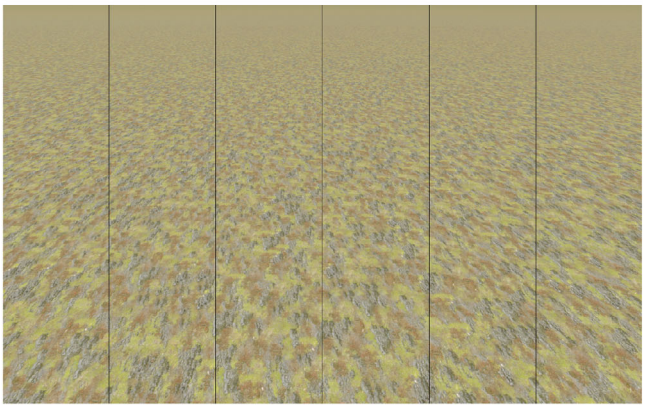
Figure 11: *Transition between three textures.*

$\mathcal{S}_i$. As a consequence, it would introduce a strong bias, illustrated Figure 12(a) near the horizon: the colour is dependent on the mixing order. This is because the mean priority of the first mix is underestimated, leading to an over-representation of the third texture.

To overcome this problem, we leverage our Gaussian approximation of the distribution of priorities. Our problem boils down to deriving the PDF of the max of two Gaussian variables $\mathcal{S}_i + v_i \sim \mathcal{N}(\mu_i, \omega_i^2)$. Nadarajah and Kotz [NK08] show that the mean and



(a) *Naive linear interpolation*



(b) *Equations* (15) *and* (16)

Figure 12: *Recursive Mix-Max applied to $N = 3$ textures: A (Grass), B (Stone) and C (Dirt), with equal interpolation fields $v_i = \frac{1}{3}$. From left to right, we mix in the order ABC, BAC, ACB, CAB, BCA, CBA. (a) A naive linear interpolation of the priority mean μ and variance ω^2 biases the mean downwards. So the last texture to be mixed is over-represented (see the green, grey and brown bias near the horizon). (b) Equations (15) and (16) provide consistent results.*

variance of the max are given by

$$\mu = \mu_1 \Phi\left(\frac{\mu_1 - \mu_2}{\widetilde{\omega}}\right) + \mu_2 \Phi\left(\frac{\mu_2 - \mu_1}{\widetilde{\omega}}\right) + \widetilde{\omega}\phi\left(\frac{\mu_1 - \mu_2}{\widetilde{\omega}}\right) \tag{15}$$

and

$$\begin{aligned}\omega^2 = {} & \left(\omega_1^2 + \mu_1^2\right)\Phi\left(\frac{\mu_1 - \mu_2}{\widetilde{\omega}}\right) + \left(\omega_2^2 + \mu_2^2\right)\Phi\left(\frac{\mu_2 - \mu_1}{\widetilde{\omega}}\right) \\ & + (\mu_1 + \mu_2)\widetilde{\omega}\phi\left(\frac{\mu_1 - \mu_2}{\widetilde{\omega}}\right) - \mu^2,\end{aligned} \tag{16}$$

where ϕ and Φ are, respectively, the PDF and the CDF of the standard normal distribution, and $\widetilde{\omega}^2 = \omega_1^2 + \omega_2^2$. The third term in Equation (15) is an upward shift of the mean when taking the max, which corrects errors previously identified, as illustrated in Figure 12(b).

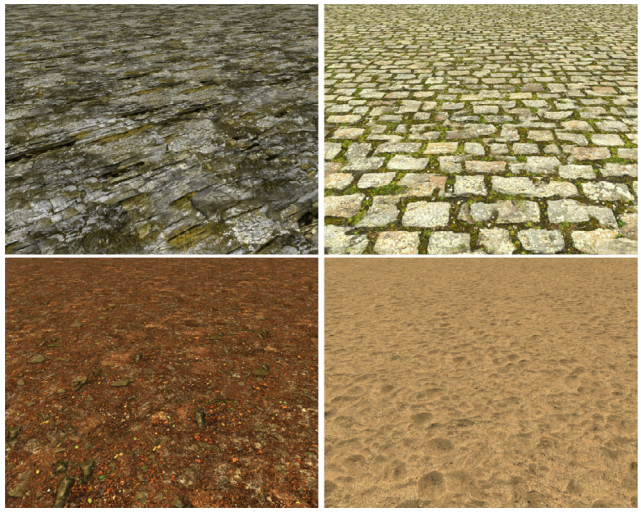


Figure 13: *Synthesis, high resolution results.*

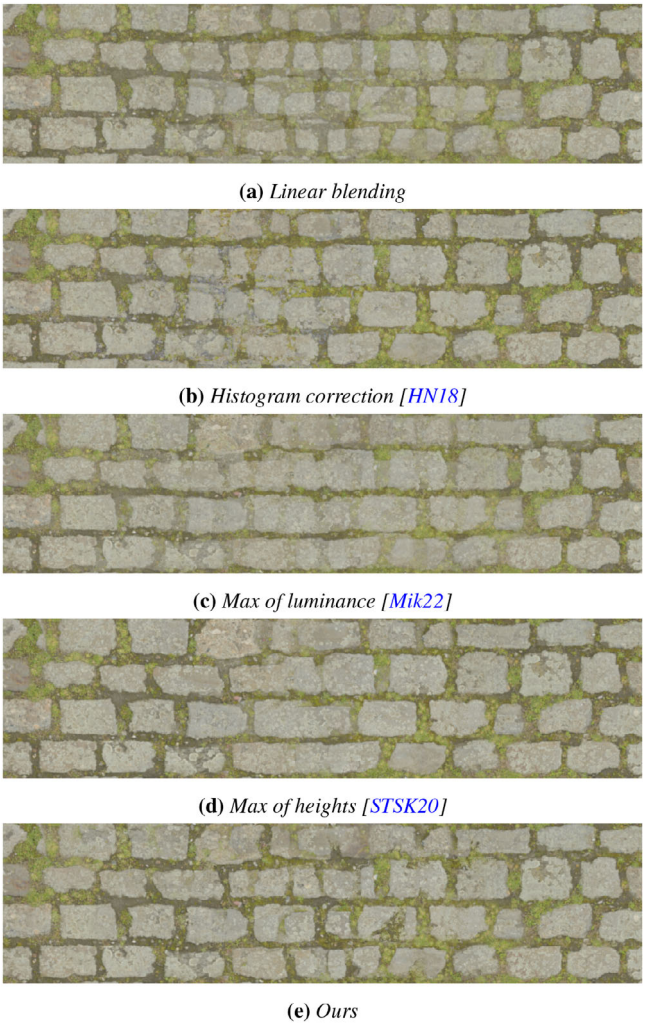
Note that the max has a bell-shaped distribution but it is not a Gaussian [NK08]. So, when applying the operator recursively, we may accumulate small errors due to our Gaussian hypothesis. In practice, we did not observe noticeable errors.

8. Discussion

Implementation details. Our technique is straightforward to implement, and easy to integrate into the graphical pipeline. It consists in Equations (7), (11), (13), (15) and (16). $\mathcal{T}_i(\mathbb{P})$, $\mathcal{S}_i(\mathbb{P})$ and $\mathcal{S}_i^2(\mathbb{P})$ are fetched in their respective MIP-maps. $v_i(\mathbb{P})$ is approximated by the value at the centre of $\mathbb{P}$ [GSDT22]. λ_i are set by the user. The denominators in Equations (13), (15) and (16) are clamped at a small positive value to avoid divisions by zero; at the same time, it makes $1 - \Phi$ tend to a Heaviside step function, which emulates Equation (4) for single texels with no micro-priority. As Φ is not packaged in the shader language, we used $\frac{1}{2} + \frac{1}{2}\tanh(0.85x)$ as an approximation.

We implemented our technique both in WebGL and Vulkan. As a use case for performance testing, with Vulkan implementation and an Nvidia RTX3060 GPU, the demo of Figure 16 runs above 60 frames per second at a 5120 × 2880 ultra HD resolution. Each of the three examples has 2048^2 texels and five 8-bits channels (three colours and two priorities), which amounts to 60-MB storage, plus the MIP-map overhead. No texture compression is considered here.

Results and limitations. Figure 13 shows synthesis results with different textures. The stationarity is assessed numerically in supplemental document 2. Compared to previous techniques [HN18, Mik22, STSK20], there is no ghosting artefact and the result is stationary, as shown Figure 14. A close investigation of this example shows that our model is pushed to its limits: some patterns are outlined, due to the interaction between the priority maps on the border of the patterns. As a future work, we would like to investigate the use of pattern morphing [RLW*09] to reduce misalignment problems.



(a) *Linear blending*

(b) *Histogram correction [HN18]*

(c) *Max of luminance [Mik22]*

(d) *Max of heights [STSK20]*

(e) *Ours*

Figure 14: *Comparison with state of the art mixing formulas. Linear blending (a) and blending with histogram correction (b) produce ghosting artefacts. Content-dependent mixes based on luminance (c) and height (d) are not stationary. Our method (e) has no ghosting and is stationary. However, misalignment of patterns can still be visible.*

Figure 15: *Transitions, high-resolution results. The textures are repeated periodically (no tiling and blending).*

Several additional results of transitions are presented in the supplemental document 2, along with variations of the micro-priority and filtering error measurements. Figure 15 shows transitions on PBR textures, rendered in real-time at very high resolutions. Figure 16 combines the synthesis of and the transitions between three textures. A landscape is covered with grass, stone and dirt with spatial distribution controlled by the terrain's topography.

Demos. We provide a number of WebGL demos matching our figures that we also present in supplemental document 3:

- Synthesis only (Figure 13): https://igg.unistra.fr/people/fournier/mixmax/synthesis/
- Transitions only on periodic textures (Figure 15): https://igg.unistra.fr/people/fournier/mixmax/blend_demo/
- Synthesis and transitions combined on a terrain (Figure 16): https://igg.unistra.fr/people/fournier/mixmax/terrain/
- Richer light effects with a point light that follows the mouse cursor: https://igg.unistra.fr/people/fournier/mixmax/light_demo/

Future work. The design of the priority maps would deserve further investigation. We used maps packaged with PBR textures, such as heights, normals or luminance. However the final result depends on the interaction between two priority maps, which is hard to predict, so specific design could be beneficial. In particular, it might be possible to design maps that do not require the inversion

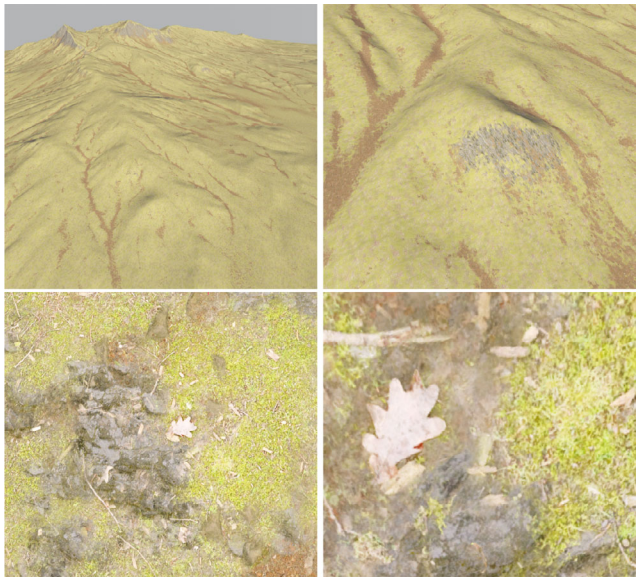


Figure 16: *Three textures (grass, rock and dirt) are synthesized and applied on a terrain. Realistic transitions are produced from millimetre to kilometre scale, and rendered in real-time at ultra-high resolution. The interpolation fields v_i are generated automatically (rocks on steep slopes, dirt where water flows) to control spatial variations at a large scale. Each of the three textures is synthesized by blending three tiles [HN18], resulting in up to a nine-way blending.*

of Section 4, which would reduce the remaining contour artefacts (Figure 14, bottom).

Our micro-priority model (Section 6) represents sub-pixel variations. We defined constant λ_i to introduce blending and transparency effects. Making λ_i vary across the texture could introduce more variety in the transitions, for instance by combining opaque and translucent materials. We also would like to explore further the behaviour of complex PBR material, and the combination of our model with micro-facets theory.

9. Conclusions

In this work, we introduced an operator for mixing textures, based on the intuition of intertwining and overlapping materials. Priority maps are correlated to the textures, and the max of priorities guides the mix. When applied to texture transitions, it produces complex transitions that well preserve sharp micro patterns. When applied to texture synthesis by tiling and blending, it generates stationary textures and locally preserve the histogram of the example. We provide constant-time filtering formulas that allow for very-high resolution rendering in real-time. We also extend our technique to sub-pixel variations, by introducing a statistical micro-priority representation, which unifies the model from micro-scale to macro-scale.

A first avenue for future work deals with the design of the priority maps and their usage by artists. A second avenue deals with the combination with physically based material models and rendering.

Acknowledgements

We would like to thank Thery Sylvain for his help in implementing the method.

Conflicts of Interest

The authors declare that the research was conducted in the absence of any commercial or financial relationship that could be construed as a potential conflict of interest.

References

[BS19] Burley B., Studios W. D. A.: On histogram-preserving blending for randomized texture tiling. *Journal of Computer Graphics Techniques 8*, 4 (2019), 31–53.

[CSHD03] Cohen M. F., Shade J., Hiller S., Deussen O.: Wang tiles for image and texture generation. *ACM Transactions on Graphics 22*, 3 (July 2003), 287–294. doi: https://doi.org/10.1145/882262.882265.

[DH19] Deliot T., Heitz E.: Procedural stochastic textures by tiling and blending. In *GPU Zen 2: Advanced Rendering Techniques*. Black Cat Publishing Inc., Encinitas, CA (2019).

[Dow73] Downton F.: The estimation of pr (y < x) in the normal case. *Technometrics 15*, 3 (1973), 551–558.

[DSB*12] Darabi S., Shechtman E., Barnes C., Goldman D. B., Sen P.: Image melding: Combining inconsistent images using patch-based synthesis. *ACM Transactions on Graphics 31*, 4 (July 2012), 82:1–82:10. doi: https://doi.org/10.1145/2185520.2185578.

[EF01] Efros A. A., Freeman W. T.: Image quilting for texture synthesis and transfer. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques* (2001), pp. 341–346.

[GSDC17] Guingo G., Sauvage B., Dischler J.-M., Cani M.-P.: Bi-layer textures: A model for synthesis and deformation of composite textures. *Computer Graphics Forum 36*, 4 (2017), 111–122. doi: https://doi.org/10.1111/cgf.13229.

[GSDT22] Grenier C., Sauvage B., Dischler J.-M., Thery S.: Color-mapped noise vector fields for generating procedural micro-patterns. *Computer Graphics Forum 41* (2022), 477–487.

[HMR06] Hardy A., Mc Roberts D. A. K.: Blend maps: Enhanced terrain texturing. In *Proceedings of the 2006 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists on IT Research in Developing Countries* (2006), pp. 61–70.

[HN18] Heitz E., Neyret F.: High-performance by-example noise using a histogram-preserving blending operator. In *Proceedings of the ACM on Computer Graphics and Interactive Techniques 1*, 2 (2018), 1–25.

[HVCB21] Heitz E., Vanhoey K., Chambon T., Belcour L.: A sliced Wasserstein loss for neural texture synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2021), pp. 9412–9420.

[KCD15] Kolář M., Chalmers A., Debattista K.: Repeatable texture sampling with interchangeable patches. *The Visual Computer* (2015), 1–10. doi: https://doi.org/10.1007/s00371-015-1161-4.

[KEBK05] Kwatra V., Essa I., Bobick A., Kwatra N.: Texture optimization for example-based synthesis. *ACM Transactions on Graphics 24*, 3 (July 2005), 795–802. doi: https://doi.org/10.1145/1073204.1073263.

[KNL*15] Kaspar A., Neubert B., Lischinski D., Pauly M., Kopf J.: Self tuning texture optimization. *Computer Graphics Forum 34*, 2 (2015), 349–359. doi: https://doi.org/10.1111/cgf.12565.

[KSE*03] Kwatra V., Schödl A., Essa I., Turk G., Bobick A.: Graphcut textures: Image and video synthesis using graph cuts. *ACM Transactions on Graphics (TOG) 22*, 3 (2003), 277–286.

[LSA*16] Lockerman Y., Sauvage B., Allègre R., Dischler J.-M., Dorsey J., Rushmeier H.: Multi-scale label-map extraction for texture synthesis. *ACM Transactions on Graphics 35*, 4 (July 2016), 140:1–140:12. (Proceedings of Siggraph'16). doi: https://doi.org/10.1145/2897824.2925964.

[LSD21] Lutz N., Sauvage B., Dischler J.-M.: Cyclostationary Gaussian noise: Theory and synthesis. *Computer Graphics Forum (Proceedings Eurographics) 40*, 2 (2021), 239–250.

[LSD23] Lutz N., Sauvage B., Dischler J.-M.: Preserving the autocovariance of texture tilings using importance sampling. *Computer Graphics Forum (Proceedings Eurographics) 42*, 2 (May 2023), 347–358. doi: https://doi.org/10.1111/cgf.14766.

[LSLD19] Lutz N., Sauvage B., Larue F., Dischler J.-M.: Anisotropic filtering for on-the-fly patch-based texturing. In *Eurographics 2019, Short Papers* (May 2019), Eurographics Association.

[Mik22] Mikkelsen M. S.: Practical real-time hex-tiling. *Journal of Computer Graphics Techniques 11*, 2 (2022), 77–94.

[NK08] Nadarajah S., Kotz S.: Exact distribution of the max/min of two Gaussian random variables. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems 16*, 2 (2008), 210–212.

[OB10] Olano M., Baker D.: Lean mapping. In *Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (2010), pp. 181–188.

[PBK13] Park H., Byun H., Kim C.: Multi-exemplar inhomogeneous texture synthesis. *Computers & Graphics 37*, 1-2 (2013), 54–64.

[PD84] Porter T., Duff T.: Compositing digital images. In *Proceedings of the 11th Annual Conference on Computer Graphics and Interactive Techniques* (1984), pp. 253–259.

[RLW*09] Ray N., Lévy B., Wang H., Turk G., Vallet B.: Material space texturing. *Computer Graphics Forum 28*, 6 (2009), 1659–1669. doi: https://doi.org/10.1111/j.1467-8659.2009.01423.x.

[RSK10] Ruiters R., Schnabel R., Klein R.: Patch-based texture interpolation. *Computer Graphics Forum 29*, 4 (2010), 1421–1429. doi: https://doi.org/10.1111/j.1467-8659.2010.01739.x.

[STSK20] Schuster K., Trettner P., Schmitz P., Kobbelt L.: A three-level approach to texture mapping and synthesis on 3D surfaces. *Proceedings of the ACM on Computer Graphics and Interactive Techniques 3*, 2 (2020), 1–19.

[VDKCC20] Vacher J., Davila A., Kohn A., Coen-Cagli R.: Texture interpolation for probing visual perception. *Advances in Neural Information Processing Systems 33* (2020), 22146–22157.

[VSLD13] Vanhoey K., Sauvage B., Larue F., Dischler J.-M.: On-the-fly multi-scale infinite texturing from example. *ACM Transactions on Graphics 32*, 6 (Nov. 2013), 208:1–208:10. (Proceedings of Siggraph Asia'13). doi: https://doi.org/10.1145/2508363.2508383.

[WDK*16] Wu F., Dong W., Kong Y., Mei X., Yan D.-M., Zhang X., Paul J.-C.: Feature-aware natural texture synthesis. *The Visual Computer 32* (2016), 43–55. doi: https://doi.org/10.1007/s00371-014-1054-y.

[WLKT09] Wei L.-Y., Lefebvre S., Kwatra V., Turk G.: State of the art in example-based texture synthesis. In *Eurographics 2009, State of the Art Report, EG-STAR* (2009), Eurographics Association, pp. 93–117.

[XFPA14] Xia G.-S., Ferradans S., Peyré G., Aujol J.-F.: Synthesizing and mixing stationary gaussian texture models. *SIAM Journal on Imaging Sciences 7*, 1 (2014), 476–508. doi: https://doi.org/10.1137/130918010.

[ZFCVG05] Zalesny A., Ferrari V., Caenen G., Van Gool L.: Composite texture synthesis. *International Journal of Computer Vision 62*, 1-2 (Apr. 2005), 161–176. doi: https://doi.org/10.1007/s11263-005-4640-7.

Supporting Information

Additional supporting information may be found online in the Supporting Information section at the end of the article.

Supporting Information