GPU-Friendly Laplacian Texture Blending
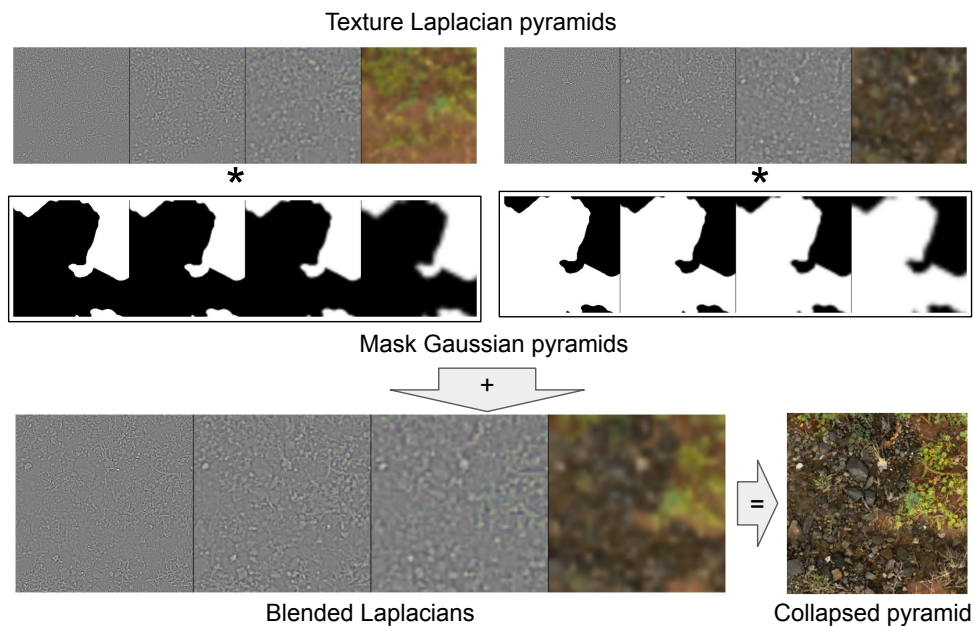
Bartlomiej Wronski
NVIDIA, USA

Figure 1. Overview: Instead of blending material textures with a fixed blending radius, we propose to blend different Laplacian pyramid levels with different mask sharpness proportional to Laplacian feature size. This ensures contrast and detail preservation as well as smooth perceptual transition. Laplacian levels are constructed in place from traditional texture mipmaps.

Abstract

Texture and material blending is one of the leading methods for adding variety to rendered virtual worlds, creating composite materials, and generating procedural content. When done naively, it can introduce either visible seams or contrast loss, leading to an unnatural look not representative of blended textures. Earlier work proposed addressing this problem through careful manual parameter tuning, lengthy per-texture statistics precomputation, look-up tables, or training deep neural networks. In this work, we propose an alternative approach based on insights from image processing and Laplacian pyramid blending. Our approach does not

require any precomputation or increased memory usage (other than the presence of a regular, non-Laplacian, texture mipmap chain), does not produce ghosting, preserves sharp local features, and can run in real time on the GPU at the cost of a few additional lower mipmap texture taps.

1. Introduction

Texture and material creation is one of the most time-consuming aspects of 3D content creation and defines the final appearance of the rendered objects. In physically based shading, the artist defines the surface reflectance and other physical properties of the BRDF [Burley 2012]. To help artists author physically based materials, a common industry practice is creating a hierarchy of progressively more complex materials that get blended through masking [Neubelt and Pettineo 2013]. Masking is typically manually tweaked by artists but can be extended to the procedural generation of infinite materials. To reduce visible repetitiveness and material tiling, Heitz and Neyret [2018] proposed hexagonal macro-tiling of random rotations of material textures. They analyzed a common problem of naively blending textures—either sharp, unnatural transitions or contrast loss and ghosting—and proposed a solution based on local histogram correction based on precomputation. We propose a different solution to the problem of texture and material blending that can also be applied to interpolate different textures. Our solution blends local Laplacians to reduce variance loss and provide a perceptually natural transition of differently sized features. Laplacian pyramid construction is approximated inline in the final shader using traditional texture mipmaps (used for regular trilinear filtering) and requires no costly precomputation. Our approach is designed to run in real time on the GPU and requires defining only a single parameter: the number of the Laplacian levels that affect the maximum size of the features that get blended.

2. Related Work

Texture blending is common in practice and is part of texture and material creation, but relatively little of it has been explored by the literature. Artists are assumed to tweak blending masks and source materials until the desired look is achieved. The state-of-the-art procedural texturing tool Adobe Substance Designer uses various simple pointwise blending modes to facilitate that process [Adobe 2023]. Mikkelsen [2022] proposed to use simple pointwise blending with a manually designed, content-dependent weight curve for natural-looking transitions.

While artists can manually adjust masks and the appearance of materials in traditional workflows, procedural texture synthesis aims to automate this process. Pure pointwise operations often fail to produce reliable and consistent procedural blending results, as a single pixel does not inform about neighbors, image patterns, or struc-

tures. Most procedural texture synthesis publications rely on global or local neighborhood approaches during runtime or as a precomputation step.

One of the earliest practical works was noise by example [Galerne et al. 2012]. Those early methods were costly and required offline optimization procedures, such as basis pursuit. Subsequent works targeted performance optimizations and reduction of the precomputation needed. Yu et al. [2010] proposed global variance-based normalization for blending fluid textures. Heitz and Neyret [2018] identified this approach's quality shortcomings and instead proposed adjusting the local texture values based on offline precomputation of optimal transport of histograms. In later work, Burley [2019] proposed simplifying that process through 1D precomputation along with improvements to reduce visual artifacts by taking clipping into account. Recently, Fournier and Sauvage [2024] introduced a novel pointwise operator combined with fast precomputation techniques that guarantee consistent minification, antialiasing upon magnification, and stationarity of the resulting blended textures.

3. Laplacian Pyramid Blending

Blending natural and photographic images is a common operation in image processing literature, focusing significantly on perceptual effects on image color, details, and discontinuities. Pérez et al. [2003] have approached the problem of blending images in the gradient domain by swapping image gradients and solving a screened Poisson equation. One of the key insights of their method is that local image gradient discontinuities don't lead to a perceived image discontinuity.

In parallel and in a similar manner, Brown and Lowe [2003] proposed to blend panorama photographs using a simple, two-level frequency decomposition of the image—a *detail* layer with high frequencies that are blended locally, and a *global* layer that is blended with a large radius to prevent visible seams or discontinuity. Efros [2005] expanded this idea to a multi-level Laplacian decomposition and the blending of images using Laplacian pyramids in his influential course on image processing.

This technique is commonly used in perceptual image processing: for example, in the Exposure Fusion [Mertens et al. 2007] algorithm used for high-dynamic-range (HDR) fusion of multiple low-dynamic-range images or to locally tone-map an HDR image in the HDR+ pipeline [Hasinoff et al. 2016]. Earlier HDR fusion approaches would, for example, apply the tone mapping only to the bilaterally filtered image while adding the non-tone-mapped *detail* layer back later, which often resulted in an unnatural, exaggerated, and over-detailed look. Exposure Fusion solved this problem with a much smoother, layer-dependent blending radius from Gaussian-blurring blending masks. We analyze why this method preserves contrast from the image statistics and signal-processing perspectives and propose blending materials and their textures using Laplacian pyramids.
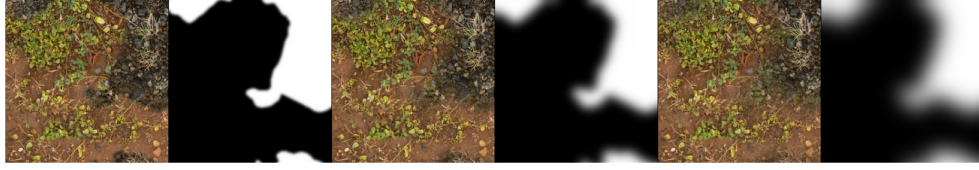
Figure 2. Different blending radii can affect the visual look of blended tiled textures. We can observe either unnatural, harsh transitions (left) or significant blurriness and contrast loss (middle). Furthermore, the rightmost example shows significant ghosting and overlap of distinct details.

4. Method

When directly linearly blending different textures, the transition radius and mask smoothness affect the final image look (Figure 2). The rightmost example shows the smoothest transition but has less contrast and detail than the two others, with smaller blending radii. Furthermore, various small visual features overlap, producing an unnatural *ghosting* effect. Addressing the shortcomings of such blending was one of the contributions of the method of Heitz and Neyret [2018], which was one of the inspirations for our work.

We propose a different and straightforward method, based on the ideas present in the image processing work: blending Laplacians with different radii [Brown and Lowe 2003; Efros 2005; Mertens et al. 2007]. We take images x and y and decompose them with Laplacian operators:

$$x = l_{x0} + L_{x1} + L_{x2} + \cdots + G_{xn}, \tag{1}$$

$$y = L_{y0} + L_{y1} + L_{y2} + \cdots + G_{yn}, \tag{2}$$

where L_{xm} is a Laplacian pyramid level of the image x and G_{xn} is the final Gaussian level of the signal. Similarly, we create multiple *Gaussian* levels of the mask image m: $G_{m0}, \ldots, G_{mn}$. The radii of Gaussian blurring of the consequent Gaussian levels are assumed (but not required) to be the same as the radii of Gaussian blurs during the construction of Laplacian pyramids for images x and y.

Given this notation, the proposed blending operation is as follows:

$$Blend(x, y, m) = G_{xn} \cdot G_{mn} + G_{yn} \cdot (1 - G_{mn}) + \sum_{i=0}^{n-1} L_{xi} \cdot G_{mi} + L_{yi} \cdot (1 - G_{mi}). \tag{3}$$

A visual example of our method's appearance is presented in Figure 3.

This method preserves sharpness and contrast in the transition area while blending the textures over a large area. This operation generalizes to blending more than just two textures. In such cases, we replace the m and $(1 - m)$ with different masks $m_0, \ldots, m_k$ and add the weighted Laplacians linearly. The number of added Laplacian levels defines the sharpness of the transition. Before showing the impact of the
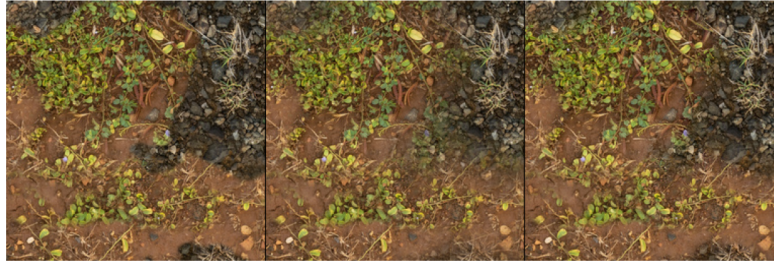
Figure 3. Left: Direct texture blending with a small radius. Middle: Direct texture blending with a large radius. Right: Laplacian texture blending using three Laplacian levels with a wide radius blending low-frequency details and a narrow radius blending high-frequency details.

Laplacian count, the pyramid construction filters, and presenting a practical implementation, we analyze why the method works well for preserving the visual appearance and contrast of the blended textures.

4.1. Perceptual Impact of Frequency-Dependent Blending Radius

Efros [2005] noted in his course how differently sized features require different blending radii to *look natural*. We demonstrate this effect on noise textures in Figure 4. When the transition radius is mismatched with the frequency content of the noise texture, it results in a discontinuous look or a visible blurry *stripe* between the two textures. A perceptually optimal blending radius is wide for low-frequency noise and medium for medium-frequency noise, and there is almost no transition for high-frequency noise.

The reason for visible discontinuities when using a small blending radius to blend low-frequency features can be analyzed from a signal-processing perspective. Multiplying a texture by a mask in pixel space is the same as the convolution of two signals
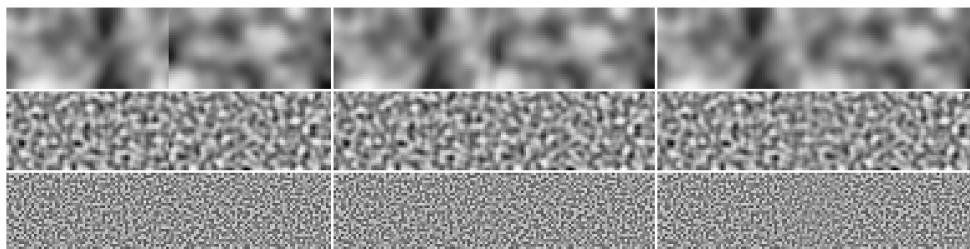

Figure 4. Top to bottom: Noise textures with a different frequency content, from low to high frequencies. Left to right: Different blending radii between two textures. Different frequencies of noise require different transition radii for the most natural appearance. A small blending radius produces visible discontinuities on low-frequency content, while a wide radius causes contrast and detail loss on high-frequency textures.

in the frequency space. A harsh transition ramp has rich frequency content and, after multiplication, causes new, high frequencies not present in either of the original images to appear.

It would seem that a wide radius—with a smooth falloff—would be preferred, but it's not the case for medium and high frequencies. We look at those cases from the perspective of variance loss.

4.2. Analysis: Variance Loss

We use the variance of the color variation in the texture as a simple-to-analyze proxy for the contrast. We look at the texels of blended textures x and y as random variables X and Y and analyze their blended variance $Var(a \cdot X + (1 - a) \cdot Y)$ after linear blending:

$$Var(a \cdot X + (1-a) \cdot Y) = a^2 \cdot Var(X) + (1-a)^2 \cdot Var(X) + a \cdot (1-a) \cdot Cov(X,Y). \quad (4)$$

The effect on variance is zero on the edges of the transition and the largest in the middle of the transition:

$$Var\left(\frac{X}{2} + \frac{Y}{2}\right) = \frac{Var(X)}{4} + \frac{Var(X)}{4} + \frac{Cov(X,Y)}{2}. \quad (5)$$

In the case of uncorrelated blended textures, blending reduces the variance by half, reducing visual contrast. If the textures are anti-correlated (which can happen when blending the same periodic texture with different phase offsets), it can lead to the complete zeroing of the texture variation and detail.

The blending radius defines the size of the area with a lowered variance, which for uncorrelated variables with the same variance on average is equal to

$$\int_0^1 a^2 \cdot Var(X) + (1-a)^2 \cdot Var(X) \mathrm{d}a = \frac{2}{3} Var(X). \quad (6)$$

Outside of the blending area, there is no variance loss. The wider the transition area, the more variance and contrast are reduced. While looking at variance only and restoring it is insufficient [Heitz and Neyret 2018], and better methods operate on full histograms, it leads naturally to the analysis of benefits of the Laplacian decomposition.

4.3. Laplacian Decomposition of Variance Loss

If the Laplacian levels are uncorrelated, we can write

$$Var(X) \approx Var(L_{X0}) + Var(L_{X1}) + Var(L_{X2}) + \cdots + Var(G_{Xn}). \quad (7)$$

In the case of a perfect Fourier decomposition, this equality is strict from Parseval's theorem.
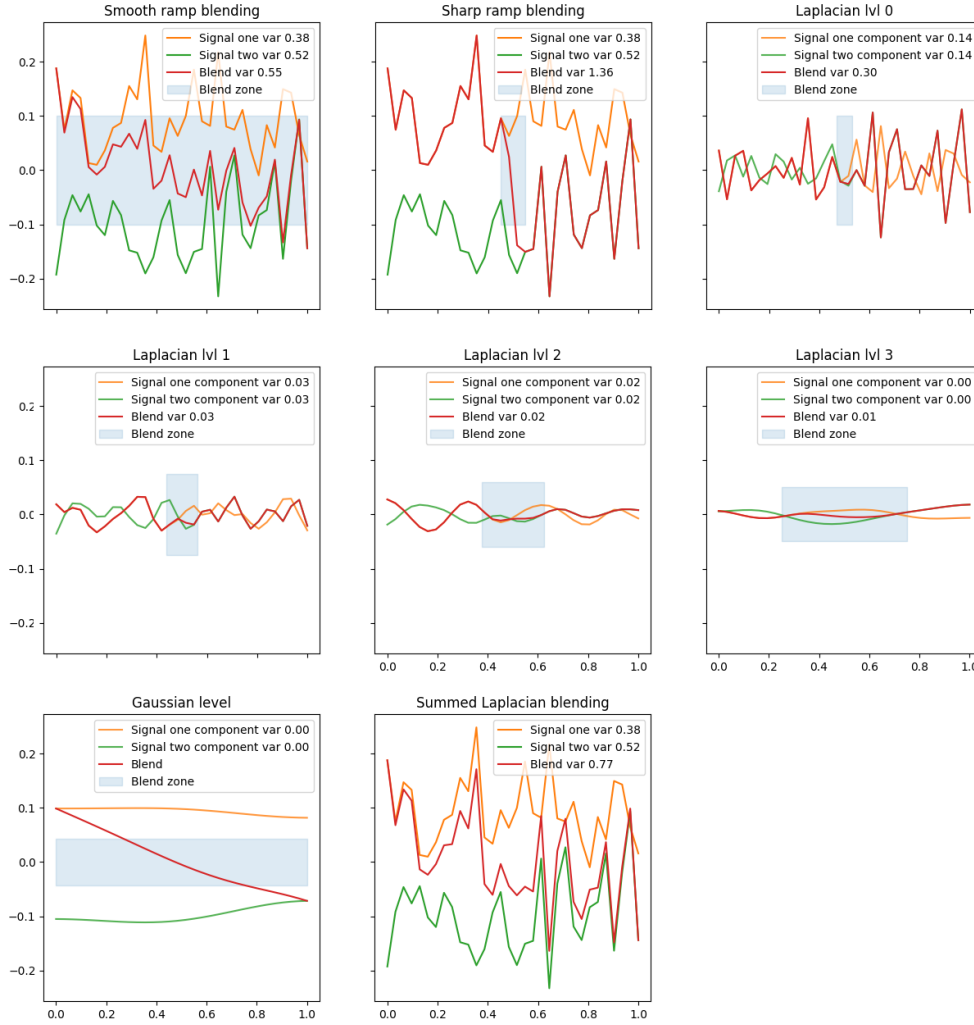
Figure 5. A toy 1D example demonstrating how, with Laplacian blending, regions of variance loss are distributed between different levels.

In practice, the correlation of the Laplacian levels depends on the quality of the used filters and the specific signals (for example, due to insufficient filtering and aliasing). We analyze the impact of the used filters in Section 5.2, but for now, we assume that the correlation is small:

$$Cov(L_{Xk}, L_{Xm}) \approx 0, \tag{8}$$

$$Cov(L_{Xk}, G_{Xn}) \approx 0. \tag{9}$$

Blending Laplacian pyramid levels with different radii, we distribute the variance reduction between different Laplacian levels and over different area sizes. We show a toy 1D linear blending example in Figure 5. Two different 1D signals are blended, and a sharp linear transition region causes a visible discontinuous "jump," while a smooth

linear transition reduces sharpness and the original signal features. Conversely, Laplacian blending distributes blending across different frequency content levels and region sizes. For instance, the highest signal frequencies get attenuated only over very small regions.

The exact variance and contrast loss depends on the spectral content of the blended images and is always between the sharpest (single texel) and the widest blending radii. This causes a smaller contrast or high-frequency detail loss than a wide blending radius while preserving its natural perceptual smoothness and lack of visible discontinuities.

5. Controlling the Behavior

The proposed method does not require per-texture parameter tuning for robust behavior, but a few parameters and the filtering kernel choice impact its visual appearance.

5.1. Laplacian Pyramid Level Count

The Laplacian pyramid level count is the most important parameter we propose to expose for artistic appearance control. It defines the effective radius of the transition. We suggest three to four levels as a practical default value. Above five levels, the high transition radius causes average colors of different textures to blend and some of their unique appearance identity to be lost (Figure 6). We note, however, that even such wide radius blending still doesn't show visible ghosting, and while the color contrast gets lower, the local contrast and high-frequency features are intact.

While we propose to use by default the same level of the mask Gaussian pyramid as the levels of the blended Laplacian pyramid, $0, \ldots, n$, it is possible to use biased further levels $k, \ldots, (n+k)$ for more aggressive blending with fewer levels. However, this can lead to minor ghosting, as it becomes similar to direct linear blending with a larger radius.

Figure 6. From left to right: Blending with three to seven Laplacian levels. A higher level count increases the smoothness and perceived continuity of the transition, but all preserve sharp contrast and high-frequency features without visible ghosting. The transitions above five Laplacian levels are very smooth, losing some of the distinct identity of the source textures. This can be both an advantage or undesirable from an artistic perspective depending on the use case.

5.2.   Prefiltering and Upsampling Kernel

There are many ways to construct a Laplacian pyramid. For example, it can be either decimated (where each next level is a lower resolution and often constructed through subtracting progressively more low-pass-filtered and decimated images) or undecimated using bandpass filters. We focus on the decimated case, as it allows for an efficient, practical, and low-memory-use implementation. Decimated Laplacian pyramid behavior and each level's contents depend on both the filter used before decimation and the level upsampling filter:

$$L_{xk} = G_{xk} - F_{up}\left((F_{down}(G_{xk})\downarrow)\uparrow\right),\qquad(10)$$

where F_{down} is a downsampling filter, F_{up} is an upsampling filter, $\downarrow$ symbolizes a decimation operation (decreasing the resolution by dropping every other pixel in each dimension), and $\uparrow$ symbolizes the resolution increase operation by a factor of two by zero-insertion in each dimension prior to application of an upsampling F_{up} filter. Later in this text we will use the notation $\uparrow_k$ for a resolution increase by a 2^k factor.

Recommending a good and efficient filter is beyond the scope of this work, but we will analyze two commonly used downsampling filters in computer graphics. The first is a box filter, often used for mipmap chain construction due to implementation simplicity, similar behavior to trilinear filtering, and the lowest possible cost—a single texture tap when using bilinear hardware samples. The second one is the Lanczos2 filter [Duchon 1979], a sinc windowed sinc filter with a very sharp frequency response given a relatively small spatial support (4×4 in the case of Lanczos2 2D downsampling). The Lanczos2 filter demonstrates good low-pass filtering but tends to produce perceptual sharpening and some ringing. Similarly to the downsampling filter, the upsampling filter choice can impact the visual results, but we focus on the most common bilinear upsampling filter due to its very small computational cost and hardware filtering support on the GPU.

Those two downsampling filters produce different Laplacian decompositions (Figure 7). The Lanczos filter separates filtered levels better, and thus, each Laplacian difference has more energy. The filter also improves the sharpness of the final Gaussian level but creates some over- and undershoots.

This difference contributes to different visual outcomes of Laplacian pyramid blending (Figure 8). Lanczos2 Laplacian pyramid blending is sharper but produces over-darkening and overshoots in some regions. While noting the difference and that one filter might be subjectively preferred over the other, we conclude that the proposed method works well with either method.
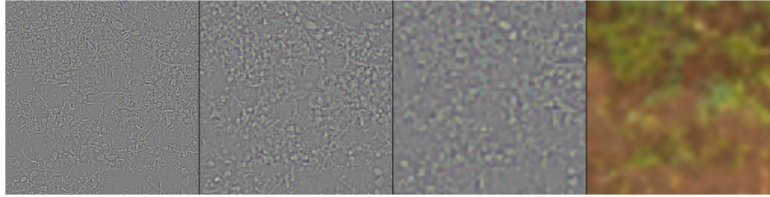
(a) *Box-filter Laplacian Pyramid*
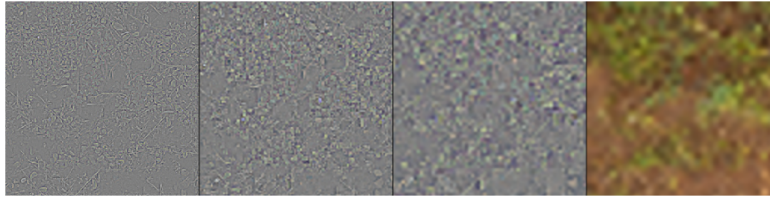
(b) *Lanczos4-filter Laplacian Pyramid*

Figure 7. The impact of used downsampling filters on Laplacian pyramid contents. The Lanczos filter preserves more high-frequency contents in lower levels, including the final Gaussian level.

(a) *Box-filter Laplacian Pyramid blending*

(b) *Lanczos4-filter Laplacian Pyramid blending*

Figure 8. Different downsampling filters and Laplacian pyramid creation methods produce a different visual outcome. Lanczos2 results are sharper but produce over-darkening and overshoots in some regions: an example in the right image is marked with a yellow oval.

6.   Practical GPU Implementation

Creating, blending, and filtering multiple image pyramids might seem costly, but we propose an efficient, GPU-friendly implementation through two simple modifications of the core algorithm. Instead of constructing a Gaussian pyramid, we use an existing texture mipmap chain. Instead of explicitly constructing a Laplacian pyramid, we propose an approximation using difference of Gaussians. We construct the full-resolution Laplacian level in place from two existing low-resolution texture mip levels:

$$L_{xk} \approx F_{up}\left((G_{xk})\uparrow_k\right) - F_{up}\left((G_{xk+1})\uparrow_{k+1}\right). \tag{11}$$

This approximation allows for the in-place construction of Laplacian levels in the final shader when a blended texture is read. Listing 1 presents an example application with

```
#define NUM_LEVELS 4
vec4 tex0_levels[NUM_LEVELS+1];
vec4 tex1_levels[NUM_LEVELS+1];
vec4 mask_levels[NUM_LEVELS+1];

for (int i = 0; i < NUM_LEVELS+1; i += 1) {
    tex0_levels[i] = texture2D(tex0, uv, float(i));
    tex1_levels[i] = texture2D(tex1, uv, float(i));
    mask_levels[i] = texture2D(mask, uv, float(i));
}

vec4 blended = vec4(0.0);
for (int i = 0; i < NUM_LEVELS; i += 1) {
    vec4 tex0_laplace = tex0_levels[i] - tex0_levels[i+1];
    vec4 tex1_laplace = tex1_levels[i] - tex1_levels[i+1];
    blended +=  tex0_laplace * (1.0 - mask_levels[i]) +
                tex1_laplace * mask_levels[i];
}
// Gaussian level.
vec4 tex0_gauss = tex0_levels[NUM_LEVELS];
vec4 tex1_gauss = tex1_levels[NUM_LEVELS];
blended +=  tex0_gauss * (1.0 - mask_levels[NUM_LEVELS]) +
            tex1_gauss * mask_levels[NUM_LEVELS];
```

Listing 1. Example implementation.

four levels. With the proposed implementation, there is no memory storage overhead or precomputation— assuming that textures already have mipmaps. This code scales to regular, pointwise linear blends—when NUM_LEVELS is zero.

To sample n Laplacian levels and an additional Gaussian level, we take just $n+1$ samples—where all the additional samples come from lower mipmaps (with a negligible bandwidth/cache cost). Other than mipmap sampling, the arithmetic cost of our method is just multiply-adds and multiplies: the same as regular blending, but multiplying it $n+1$ times and accumulating all of the blended levels.

The whole cost of the method is $n+1$ times more samples for a given desired level and n blends and adds. In practice, the cost of additional samples doesn't need to scale linearly and depends on the hardware architecture. The final GPU performance cost depends on various factors: the utilization of the texture unit, memory bandwidth, cache sizes, register usage, and arithmetic operations. Even when computing the Laplacian pyramid from all the mip levels, the maximum used memory bandwidth is 133% of the original cost (total mipchain pixel count).

The whole method is presented in Listing 1. It is worth noting that in GLSL the third parameter of the texture2D method is the level-of-detail bias, which we will use in Section 6.1.

Levels	1–3	4	5	6	7
Overhead (ms)	Not observed	0.087	0.113	0.125	0.134

Table 1. Performance overhead of the presented method in 4K resolution.

We report example timings in Table 1, measured by rendering in 3840×2160 pixels resolution, blending two textures on an NVIDIA RTX 4090 GPU, and repeating the blending 1000 times with changing UVs and averaging the overhead. The typical Laplacian level count that produces smooth but sharp blends is three to five, for which the proposed method has a minimal runtime performance impact.

6.1. Minification and Mipmapping

The description of our method so far assumes that texture blending happens at the full resolution of the textures (the finest mip level). This is sufficient for applications such as caching blending using virtual texturing, but would pose a problem on perspective-projected 3D assets requiring varying minification levels.

We can address this limitation with a simple modification of our algorithm. We begin by observing that the spectral contents of a mip level k are low-pass filtered image frequencies from the original full-resolution texture higher than its Nyquist levels. From Equations (10) and (11) we see that this is also equivalent to zeroing out the Laplacian levels $0, \ldots, k - 1$ while keeping the further and coarser Laplacian/-Gaussian pyramid levels.

This translates to two straightforward modifications of the code in Listing 1. First, the desired mip level k has to be queried using the `textureQueryLod` method and the fetched levels start at k instead of zero. Second, the number of blended Laplacian levels and the index of the selected Gaussian level (lines 12 and 20) are set to be equal to `max(NUM_LEVELS - k, 0)`. This is functionally equivalent to the code in Listing 1 when no minification is present, drops the Laplacian level blending when the texture is minified beyond the coarsest blending level, and partially blends Laplacians in between.

However, we note that the results of alpha-blending minified textures are not the same as those of minifying alpha-blended textures, irrespective of the use of our method. This is caused by the nonlinear nature of alpha mask multiplication and shading applied to material textures described by Pharr et al. [2024], and our method is compatible with their family of stochastic filtering techniques.

6.2. Optimization: Level Skipping

If the cost of the proposed method is too high (for instance, when blending multiple textures per material, or on mobile devices), one can use a further approximation for
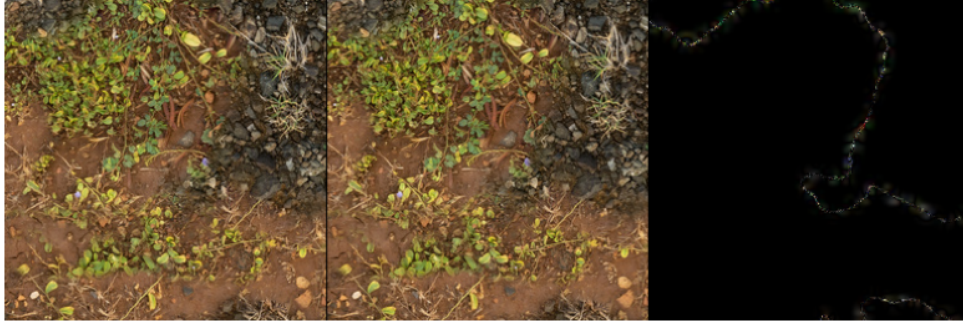
Figure 9. Left: Original algorithm, four levels. Middle: Level skipping (two levels computed). Right: Absolute difference amplified $5\times$.

Laplacian creation by skipping the mip levels: for example,

$$\widehat{L}_{xk} \approx G_{xk} - F_{up}((G_{xk+2}) \uparrow_{k+2}). \tag{12}$$

Using such a modified definition, only Laplacians and Gaussian $\widehat{L}_{x0}, \widehat{L}_{x2}, \widehat{L}_{x4}, \ldots$ need to be computed and blended. This reduces the cost overhead to $\frac{N}{2} + 1$ more samples and evaluations. It's worth noting, however, that this changes the visual appearance and leads to some minor quality loss (less preservation of sharp features and more ghosting) presented in Figure 9.

6.3. Dynamic Blend Mask Levels

We propose a second modification to our technique that eliminates the need to create a Gaussian pyramid of the mask texture. This modification can efficiently create dynamic and changing masks or remove the need to sample multiple mask texture mip levels. In this technique variation, we use smooth masks resembling alpha maps or distance fields—they can be either procedural or stored in textures. We obtain an approximation of the Gaussian level n through clamped remapping, such as rescaled

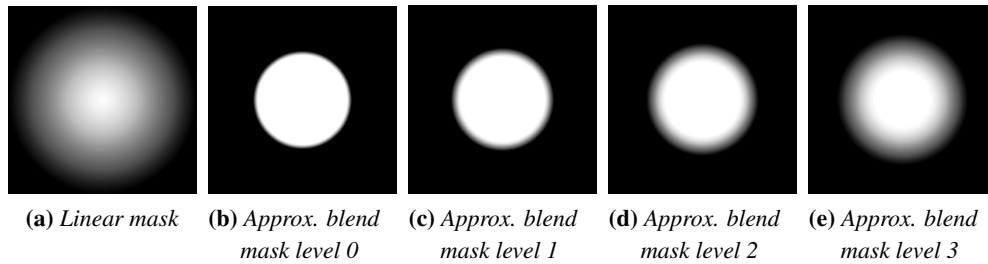| **(a)** *Linear mask* | **(b)** *Approx. blend mask level 0* | **(c)** *Approx. blend mask level 1* | **(d)** *Approx. blend mask level 2* | **(e)** *Approx. blend mask level 3* |

Figure 10. An example of creating different approximate Gaussian levels of blend masks dynamically from a source linear texture (a).

thresholding:

$$G_{mn} \approx \mathrm{clamp}\left(\frac{G_{m0} - t}{s2^n} + 0.5, 0, 1\right), \tag{13}$$

where t is the threshold of the transition center and s is a scale that depends on the values stored in the texture (for instance, texture resolution if the values are stored in the normalized $[0, 1]$ range). See the example in Figure 10.

7. Applications

The proposed method of Laplacian texture blending can be applied in different applications requiring smooth transitions between multiple textures in a real-time rendering context. We discuss two main applications.

7.1. Material Layering

The material authoring process often involves layering and blending multiple different textures of base materials [Neubelt and Pettineo 2013; Adobe 2023]. The proposed method can be included in the material creation toolset as one of the blend modes.

    The material authoring process can either create dynamic, real-time materials or bake them into textures, and our method is compatible with both workflows. The lack of precomputations makes it especially attractive for so-called *uber materials*, where a single material can use different parameters and texture sets, sometimes changed in real time. An example of dynamic adjustments can be a dynamic weather or season system in a rendering engine, when an animated mask progressively changes, revealing or covering a different material.

    Typically, materials comprise multiple textures of different BRDF properties and not just color information. Our method aims to preserve gradients present in the image. It makes no other assumptions about the color, its distribution, perceptual space, or the map semantics and thus works on any other type of property map. For example, harsh transitions of blended normal maps can produce visible surface and
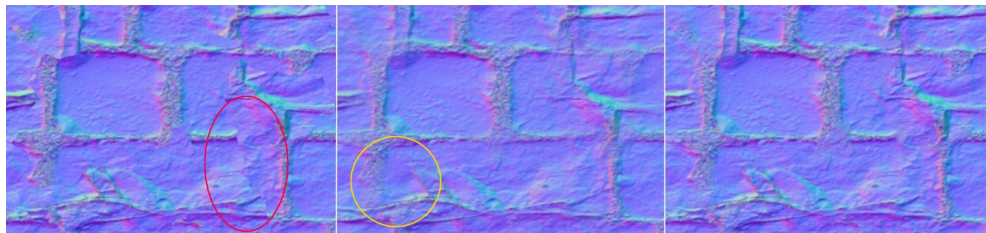

Figure 11. Blending two unrelated normal maps. Left: A narrow linear blend results in visible seams (red oval). Middle: A wide linear blend attenuates and blurs the blended normal map details toward the normal pointing up (yellow circle). Right: A Laplacian pyramid blend preserves normal map details while not producing surface discontinuities.
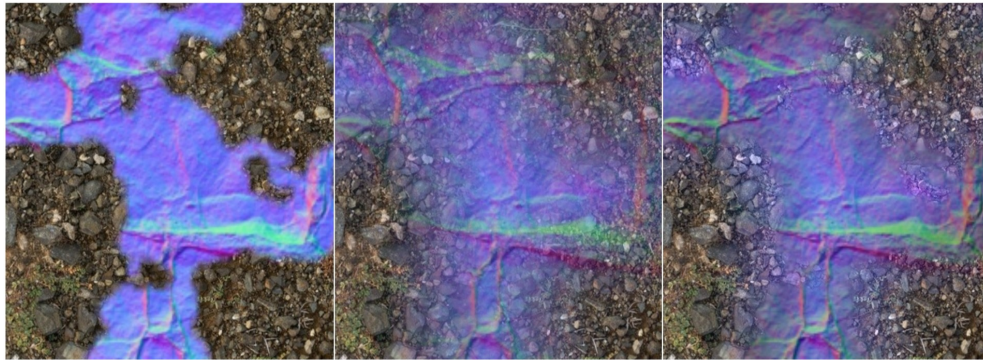
Figure 12. Extreme blending of unrelated content. Left: Narrow linear blend. Middle: Wide linear blend. Right: Laplacian pyramid blend, which creates color blends as wide as the wide linear blend while preserving contrast and avoiding feature overlap and ghosting.

lighting discontinuities. We show an example of normal map blending in Figure 11, where our method demonstrates the same perceptual smoothness of wide blends while preserving the fine details and edges.

As an additional experiment to showcase the strength of our method in this scenario, we demonstrate blending two completely unrelated textures in Figure 12. Our method blends colors similarly to wide blends while preserving local features without visible ghosting.

7.2. Procedural Texture Generation and Texture Tiling

Texture synthesis and hexagonal tiling literature [Burley 2019; Heitz and Neyret 2018; Mikkelsen 2022] inspired our work, and our method is designed to work in such a scenario. With hex tiling, every evaluated pixel blends three textures based on their distance from hexagon edges. Our method is compatible with such a setup and, similarly to the original hexagonal tiling work, doesn't require sampling hexagonal masks, as the blending weights can be determined analytically for every level. We present an example in Figure 13. The advantages of our method are high-quality results, no need for precomputations, and no need for empirical tuning. The biggest disadvantage is the increased cost. The base hex tiling method requires three samples for each material texture in the blended region; our method increases it by a level-count-dependent factor.

8. Limitations

8.1. Performance Cost

The main limitation of the proposed method is the increased runtime cost: additional samples from lower mipmaps and arithmetic operations. While additional samples

use lower mipmaps and are more likely to be localized in the L1/L2 cache without using more memory bandwidth, it is possible to saturate the texture unit with too many requests. We recommend profiling and evaluating the proposed method's cost, especially when blending multi-channel physically based rendering materials and more than two materials per pixel. For particularly complex but common scenarios like terrain material blending, an alternative can be using the proposed method to blend into a cached virtual texture [Chen 2015].

8.2. Possibility of Overshooting and Haloing

Blending Laplacians independently and at different rates can produce halos or overshooting (Gibbs phenomenon) and negative values. This was reported as a problem in HDR exposure fusion literature with proposed heuristics and workarounds [Hasinoff et al. 2016]. We have not observed those problems using a bilinear upsampling filter in the evaluated examples. Furthermore, using only a few Laplacian levels limits the maximum potential halo size. However, we recommend clamping the blended textures to the original texture value range $[0, 1]$.

8.3. Possibility of Visible Aliasing on Animated Content

If the material blending mask or blended textures are animated, imperfect filtering during the Laplacian construction can lead to aliasing. While high-quality filters would minimize this problem, it can occur when using a typical box filter and bilinear upsampling (Section 5.2). We note that it is similar to any dynamic mipmap creation, like for the bloom effect, and real-time rendering literature proposes using stronger low-pass filters while keeping the low-cost upsampling filters [Jimenez 2014].
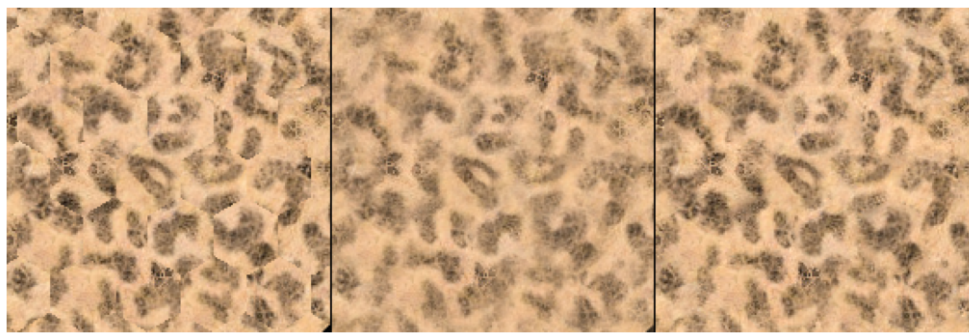

Figure 13. Left: Hard tiling without transition reveals an obvious tile structure. Middle: Soft blending causes ghosting and contrast loss. Right: Laplacian blending retains most contrast and appearance while not producing visible tiles.

9.   Conclusion

In this work, we proposed an efficient, GPU-friendly adaptation of Laplacian image blending for real-time rendering applications—semi-procedural materials, material texture layering, and example-based synthesis like hex tiling. The proposed method can be used both in a fully automated setting and presented to the artists as an additional tool for creating and blending textures and materials.

Perceptual characteristics of Laplacian pyramids have been used in image processing and computational photography literature for many years. Meanwhile, real-time graphics use image pyramids almost exclusively for performance acceleration and processing some effects at lower resolutions.

While different performance and memory storage requirements between different computer science domains require re-designing and approximating key components of any adapted algorithm, we believe that computer graphics can benefit from further adoption of computational photography techniques such as multi-level signal decomposition and nonlinear blending and filtering.

References

ADOBE. Substance 3D Designer: Blending modes description, 2023. URL: https://substance3d.adobe.com/documentation/sddoc/blending-modes-description-132120605.html. 22, 34

BROWN, M. AND LOWE, D. Recognising panoramas. In *Proceedings of the Ninth IEEE International Conference on Computer Vision*, volume 2, pages 1218–1225. IEEE Computer Society, 2003. URL: https://doi.org/https://doi.org/10.1109/ICCV.2003.1238630. 23, 24

BURLEY, B. Physically-based shading at disney. ACM SIGGRAPH course, 2012. URL: https://disneyanimation.com/publications/physically-based-shading-at-disney/. 22

BURLEY, B. On histogram-preserving blending for randomized texture tiling. *Journal of Computer Graphics Techniques (JCGT)*, 8(4):31–53, 2019. URL: https://jcgt.org/published/0008/04/02/. 23, 35

CHEN, K. Adaptive virtual texture rendering in Far Cry 4. Presented at Game Developers Conference, 2015. URL: https://gdcvault.com/play/1021761/Adaptive-Virtual-Texture-Rendering-in. 36

DUCHON, C. E. Lanczos filtering in one and two dimensions. *Journal of Applied Meteorology and Climatology*, 18(8):1016–1022, 1979. URL: https://doi.org/10.1175/1520-0450(1979)018<1016:LFIOAT>2.0.CO;2. 29

EFROS, A. Image pyramids and blending. Lecture from Computational Photography course, Carnegie Mellon University, 2005. URL: http://graphics.cs.cmu.edu/courses/15-463/2005_fall/www/Lectures/Pyramids.pdf. 23, 24, 25

FOURNIER, R. AND SAUVAGE, B. Mix-max: A content-aware operator for real-time texture transitions. *Computer Graphics Forum*, 43(6):e15193, 2024. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.15193. 23

GALERNE, B., LAGAE, A., LEFEBVRE, S., AND DRETTAKIS, G. Gabor noise by example. *ACM Transactions on Graphics*, 31(4):73:1–73:9, 2012. URL: https://doi.org/https://doi.org/10.1145/2185520.2185569. 23

HASINOFF, S. W., SHARLET, D., GEISS, R., ADAMS, A., BARRON, J. T., KAINZ, F., CHEN, J., AND LEVOY, M. Burst photography for high dynamic range and low-light imaging on mobile cameras. *ACM Transactions on Graphics*, 35(6):192:1–192:12, 2016. URL: https://doi.org/https://doi.org/10.1145/2980179.2980254. 23, 36

HEITZ, E. AND NEYRET, F. High-performance by-example noise using a histogram-preserving blending operator. *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, 1(2):31:1–31:25, 2018. URL: https://doi.org/https://doi.org/10.1145/3233304. 22, 23, 24, 26, 35

JIMENEZ, J. Next generation post-processing in Call of Duty: Advanced Warfare. SIGGRAPH course: Advances in Real-Time Rendering in Games, 2014. URL: https://www.iryoku.com/next-generation-post-processing-in-call-of-duty-advanced-warfare/. 36

MERTENS, T., KAUTZ, J., AND VAN REETH, F. Exposure fusion. In *15th Pacific Conference on Computer Graphics and Applications (PG'07)*, pages 382–390. IEEE, 2007. URL: https://doi.org/https://doi.org/10.1109/PG.2007.17. 23, 24

MIKKELSEN, M. S. Practical real-time hex-tiling. *Journal of Computer Graphics Techniques (JCGT)*, 11(3):77–94, August 2022. ISSN 2331-7418. URL: http://jcgt.org/published/0011/03/05/. 22, 35

NEUBELT, D. AND PETTINEO, M. Crafting a next-gen material pipeline for The Order: 1886, 2013. URL: https://blog.selfshadow.com/publications/s2013-shading-course/. 22, 34

PÉREZ, P., GANGNET, M., AND BLAKE, A. Poisson image editing. In *ACM SIG-GRAPH 2003 Papers*, pages 313–318. ACM, 2003. URL: https://doi.org/https://doi.org/10.1145/1201775.882269. 23

PHARR, M., WRONSKI, B., SALVI, M., AND FAJARDO, M. Filtering after shading with stochastic texture filtering. *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, 7(1):14:1–14:20, 2024. URL: https://doi.org/https://doi.org/10.1145/3651293. 32

YU, Q., NEYRET, F., BRUNETON, E., AND HOLZSCHUCH, N. Lagrangian texture advection: Preserving both spectrum and velocity field. *IEEE Transactions on Visualization and Computer Graphics*, 17(11):1612–1623, 2010. URL: https://doi.org/https://doi.org/10.1109/TVCG.2010.263. 23

Index of Supplemental Materials

A WebGL demo is provided as supplementary material.
Download:

- https://jcgt.org/published/0014/01/02/supplement_demo.zip

Run live:

- https://jcgt.org/published/0014/01/02/supplement_demo

Author Contact Information

Barlomiej Wronski
NVIDIA, USA
bwronski@nvidia.com