Non-periodic Tiling of Procedural Noise Functions

ALEKSANDR KIRILLOV, Unity Technologies

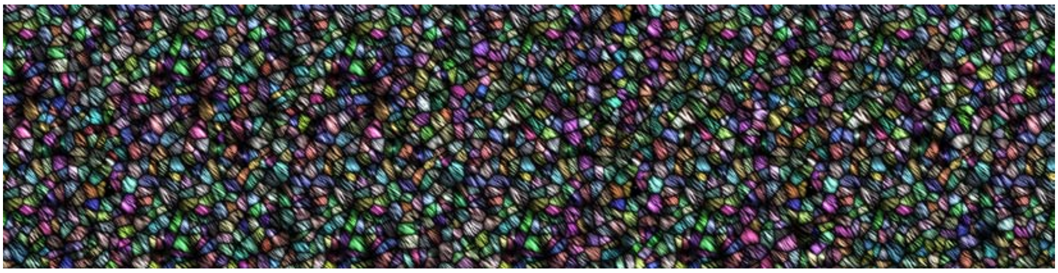


Fig. 1. An example image generated using our algorithm.

Procedural noise functions have many applications in computer graphics, ranging from texture synthesis to atmospheric effect simulation or to landscape geometry specification. Noise can either be precomputed and stored into a texture, or evaluated directly at application runtime. This choice offers a trade-off between image variance, memory consumption and performance.

Advanced tiling algorithms can be used to decrease visual repetition. Wang tiles allow a plane to be tiled in a non-periodic way, using a relatively small set of textures. Tiles can be arranged in a single texture map to enable the GPU to use hardware filtering.

In this paper, we present modifications to several popular procedural noise functions that directly produce texture maps containing the smallest complete Wang tile set. The findings presented in this paper enable non-periodic tiling of these noise functions and textures based on them, both at runtime and as a preprocessing step. These findings also allow decreasing repetition of noise-based effects in computer-generated images at a small performance cost, while maintaining or even reducing the memory consumption.

CCS Concepts: • **Computing methodologies → Rendering**; **Image processing**; **Texturing**;

Additional Key Words and Phrases: procedural texture, texture synthesis, noise, rendering, Wang tiles, corner tiles, tile-based texture mapping, non-periodic tiling, Perlin noise, Gabor noise, Worley noise, better gradient noise

ACM Reference Format:
Aleksandr Kirillov. 2018. Non-periodic Tiling of Procedural Noise Functions. *Proc. ACM Comput. Graph. Interact. Tech.* 1, 2, Article 32 (August 2018), 15 pages. https://doi.org/10.1145/3233306

1 INTRODUCTION

Procedural noise functions have been one of the key tools for adding visual fidelity in computer graphics since their introduction by Ken Perlin [1985]. Many industries employ them, from movie production to game development. Most frequently they are used in procedural texturing, allowing

Author's address: Aleksandr Kirillov, Unity Technologies, aleksandrk@unity3d.com.

2577-6193/2018/8-ART32 $15.00
https://doi.org/10.1145/3233306

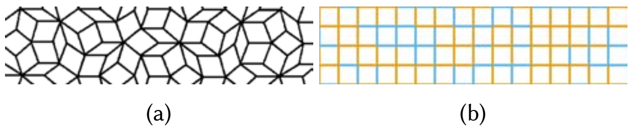


Fig. 2. Non-periodic tiling examples. (a) Penrose tiling. (b) Wang tiling.

us to carry out actions like simulating natural phenomena, generating landscape geometry, and creating textures containing surface properties or details.

Procedural methods often reduce the time spent on authoring content. They are becoming increasingly attractive both to game studios working on large projects (for example, Deguy et al. [2016] and Werle and Martinez [2017]) and to independent developers that cannot afford to spend resources on manual asset production. Because simulated environments tend to only increase in size and detail, these methods are likely to become a necessary component in the content creation process.

Applications often cannot afford to evaluate noise functions at runtime, so they use textures that store precomputed results instead. Many noise functions that are designed to be evaluated in a very efficient manner on modern hardware are periodic with a relatively small period. Because most applications only use basic texture tiling algorithms, these cases often lead to either repetition or loss of detail in the resulting image.

There are some common ways to address these problems. Decal textures can break the repetition, but require extra memory to be stored and development time to be spent on adding them to the scene. Higher texture resolution offers more detail, but leads to an increase in memory consumption.

Employing algorithms that produce non-periodic tilings (that is, tilings that lack translational symmetry) can help address both visual repetition and loss of detail. An additional benefit of using such tilings is the potential to reduce the amount of memory required to store the textures. The two most well-known examples of such algorithms are Penrose tiles [Penrose 1974] and Wang tiles [Wang 1961].

Penrose tiling (Fig. 2a) uses tiles constructed from shapes related to a pentagon. They are supplemented by matching rules in order to tile non-periodically.

Wang tiling (Fig. 2b) uses a set of rectangular tiles of the same size, with color-coded edges. A valid surface tiling is produced by any composition of tiles with two constraints: (a) tiles cannot be rotated or reflected, and (b) adjacent tiles must share an edge of the same color.

In this paper, we present modifications to several popular procedural noise functions to allow to directly produce a complete set of Wang tiles with two edge colors. The modifications can be used either in a preprocessing step or during the application runtime and can be generalized to three or more dimensions and to a higher edge color count. We also discuss filtering of the modified noise functions, and their mapping to surfaces.

Additionally, we present an improvement of the algorithm by Wei [2004] implementing Wang tiling on the GPU. We also show that our modifications enable non-periodic tiling for a large range of noise-based procedurally generated textures. Finally, we analyze the effect these modifications have on the key noise characteristics, as well as on the performance and their limitations.

2 RELATED WORK

2.1 Procedural Noise Functions

Noise is formally defined as a stationary and normal random process, with power spectrum control provided either directly or through the summation of a number of independent scaled instances of

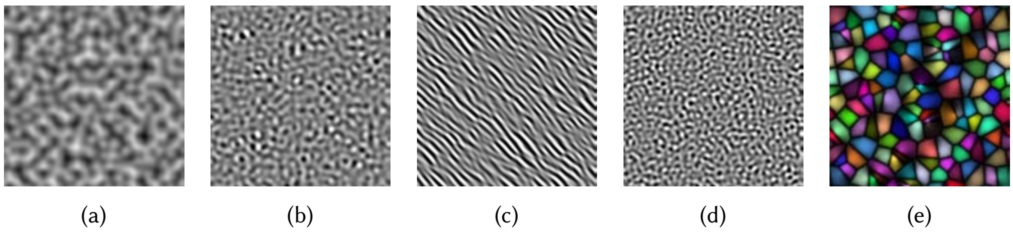


Fig. 3. Procedural noise function examples. (a) Perlin noise. (b) Better gradient noise. (c) Anisotropic Gabor noise. (d) Isotropic Gabor noise. (e) Worley noise.

noise. A procedural noise function is a procedural technique for simulating and evaluating noise [Lagae et al. 2010]. In the next subsections, we give a brief description of several popular procedural noise functions.

2.1.1 Perlin Noise. Perlin noise [1985; 2002] (Fig. 3a) is probably the most well-known procedural noise function. It is fast and simple, and is widely used to simulate natural phenomena, such as clouds, fire and smoke.

The original noise function assigns pseudo-random gradient vectors to each point of the integer lattice, and obtains the resulting value by using a cubic polynomial to smoothly interpolate between the closest gradient vectors. The pseudo-randomness is achieved by indexing a predefined gradient table using a successive hash of the lattice point coordinates with a permutation table.

There were several improvements of the original algorithm. Improved Perlin noise [Perlin 2002] reduced visual artifacts in the noise derivatives and improved overall noise appearance. Modified noise [Olano 2005] employed a hash function based on a pseudo-random number generator (PRNG), to make the noise function evaluation very efficient on the GPU. It also modified the set of gradient vectors to make the noise dimension-reducible. Better gradient noise [Kensler et al. 2008] (Fig. 3b) used separate permutation tables for each dimension, and a different hashing function to better decorrelate the hashed values. It also replaced the filter kernel to improve band-limitation, and introduced a projection method to improve the quality of solid noise on 2D surfaces.

A common property of these procedural noise functions is periodicity. The period is determined either by the permutation table size or by the hashing function.

2.1.2 Gabor Noise. Gabor noise [Lagae et al. 2009] (figures 3c and 3d) is a procedural noise function that offers intuitive and direct control over the power spectrum, and supports anisotropy. Gabor noise is constructed as a sparse convolution noise [Lewis 1989] that uses the Gabor kernel as kernel.

In order to increase the computational efficiency, Gabor noise is evaluated on a grid. The properties of the Gabor kernels are generated per cell on the fly using a PRNG. The cell size of the grid equals the radius of the Gabor kernels. This allows us to limit the evaluation of the noise function to the cell containing the point being evaluated, and its immediate neighbors.

Lagae et al. [2009] provide ways to produce both periodic and non-periodic noise. In order to obtain a noise with the period of N, the grid cells are enumerated in the row-major order with cell coordinates taken with modulo N. The seed for the PRNG is then calculated as a sum of the cell index and a global offset parameter. Non-periodic noise uses Morton order for cell enumeration.

2.1.3 Cellular Texture Basis Function. In 1996, Steven Worley introduced a new texture basis function to complement procedural noise functions [Worley 1996] (Fig. 3e). This function can be used to produce various textures like cobblestones, water and cellular structures. The algorithm randomly distributes feature points in space. Function F_n is defined as the distance between the input and the n-th closest feature point. The resulting value is determined by a function with F_1 ... F_n as parameters.

Although the cellular texture basis function is not a procedural noise function, it is very relevant to texture synthesis. Lagae et al. [2010] observed that its implementation is similar to that of sparse convolution noise.

2.2 Wang Tiles

Wang tiling is named after Hao Wang, a Chinese mathematician who assumed in 1961 that if a finite set of tiles with color-coded edges could tile a plane in a valid way, then a periodic tiling would exist as well [Wang 1961]. Later research [Berger 1966] has shown that a set of 20,426 Wang tiles exist that would tile a plane only non-periodically. A series of successor works [Culik 1996; Kari 1996] have since reduced this number, the latest one [Jeandel and Rao 2015] proving that 11 tiles that use four colors comprise the minimum set required for non-periodic tiling.

Wang tiles were first used for large non-repetitive texture synthesis by Stam [1997]. He described the construction of a limited set of patterns, mostly focusing on water surface textures and caustics. Neyret and Cani [1999] proposed to use triangular patches with color-coded edges instead of rectangles, described an algorithm to map those patches to surfaces, and provided techniques to generate them procedurally based on Perlin [1985] and Worley [1996] noises. Wei and Levoy [2000], as well as Efros and Freeman [2001], proposed methods to avoid repetition by using a small texture tile as an example. This example is used to create a larger non-repetitive texture that looks similar to the original tile. Toroidal boundary handling ensures that the resulting textures are seamlessly tileable. The results, however, induce additional computational and storage cost, and do not guarantee that the resulting texture has no internal seams. Cohen et al. [2003] adapted these methods to generate small sets of Wang tiles and introduced a stochastic process of laying down individual tiles to produce non-periodic tiling. Wei [2004] described a GPU-friendly layout for Wang tile sets in a single texture map, effectively enabling hardware filtering for such textures, and a hash-based algorithm to map the tiles on a plane.

Other research in graphics related to Wang tiles was mostly focused on object distribution [Lagae and Dutré 2005] and Poisson point set generation [Kopf et al. 2006; Lagae and Dutré 2005].

We observe, however, an insufficiency in modern Wang tile set synthesis methods. Many procedural noise functions are limited to producing only periodic images. Noise functions that allow the construction of non-periodic noise at runtime are usually computationally expensive, which makes them unusable by many real-time applications. We lift these limitations by combining Wang tiles and procedural noise functions. In the following section we present modifications to several noise synthesis algorithms which directly output a minimal complete set of Wang tiles consisting of 16 square tiles with 2 edge colors. The tiles are arranged in a single texture map in a GPU-friendly manner, as proposed by Wei [2004]. The resulting methods guarantee seamless tiling, can be extended to support a different number of edge colors or higher dimensions, and can be applied to the noise synthesis either at runtime or during the precomputation.

3 NON-PERIODIC TILING OF PROCEDURAL NOISE FUNCTIONS

Many procedural noise functions allow to use toroidal boundary handling to make the resulting textures seamlessly tileable. This is trivially achieved for integer lattice noise functions. For example, suppose that the noise function uses the function $T : \mathbb{R}^2 \rightarrow \mathbb{Z}^2$ to transform input coordinates to integer grid cell coordinates. For a non-tileable result, this function is a simple round down operation: $T_0(x, y) = (\lfloor \frac{x}{w_c} \rfloor, \lfloor \frac{y}{w_c} \rfloor)$, where $w_c \times w_c$ is the grid cell size. In order to obtain a noise

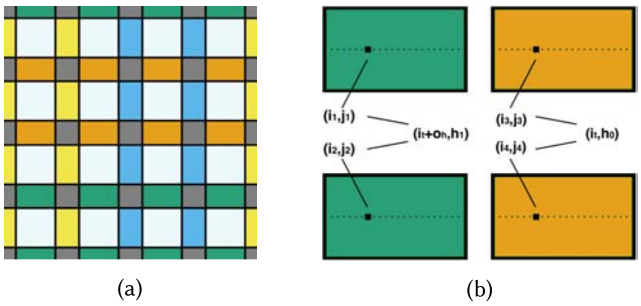


Fig. 4. A schematic representation of the $T_{PerlinW}(x, y)$ function. (a) Color-coded groups of lattice points forming a Wang tile set: C (grey), H_0 (orange), H_1 (green), V_0 (yellow), V_1 (light blue), inner points (white). The coordinates of the inner points are not modified. (b) The corresponding points within the tiles that belong to the same group are mapped to the same coordinate.

function with a period of N, the input grid cell coordinate is taken with modulo N. This ensures that the lattice coordinates are wrapped around N, provided that the function is either non-periodic or has a period greater than or equal to N.

We use a similar approach to construct the Wang tile set. For each of the selected noise functions, we define a function $T(x, y)$ that modifies the cell or lattice point coordinates in such a way that the noise function directly produces a Wang tile texture map. When paired with an algorithm to map the tiles to a surface, such as the one proposed by Wei [2004], the surface can then be tiled non-periodically.

3.1 Perlin Noise Tiling

Perlin noise is an integer lattice procedural noise function. We subdivide all lattice points into *boundary points*, which lie on Wang tile boundaries and *inner points*. The function $T(x, y)$ needs to modify only the coordinates of the boundary points. We further subdivide the boundary points into five groups: corner points; C; two groups of points (one per edge color of the Wang tiles) that belong to the vertical borders (V_0 and V_1); and two groups that belong to the horizontal borders (H_0 and H_1). In order to guarantee the correct construction of the Wang tile set, the function $T_{PerlinW}(x, y)$ has to ensure that all points within the same group at corresponding positions within a tile have the same coordinates (see Fig. 4).

Let the tile size be $w \times w$. Original lattice point coordinates are given by $(i, j) = T_{non\text{-}tileable}(x, y)$. Lattice point coordinates local to the tile can be calculated as $(i_t, j_t) = (i \bmod w, j \bmod w)$. We define the function $T_{PerlinW}(x, y)$ as a piecewise function,

$$T_{PerlinW}(x, y) = \begin{cases} (c_0, c_1), & (i, j) \in C \\ (i_t, h_0), & (i, j) \in H_0 \\ (i_t + o_h, h_1), & (i, j) \in H_1 \\ (v_0, j_t), & (i, j) \in V_0 \\ (v_1, j_t + o_v), & (i, j) \in V_1 \\ (i, j), & \text{otherwise} \end{cases} \tag{1}$$

The parameters c_0, c_1, h_0, h_1, v_0, v_1, o_h, and o_v control the locations of the lattice points that lie on the borders of Wang tiles. These parameters have no noticeable effect on the resulting noise characteristics as long as the resulting lattice regions do not intersect each other. In order to

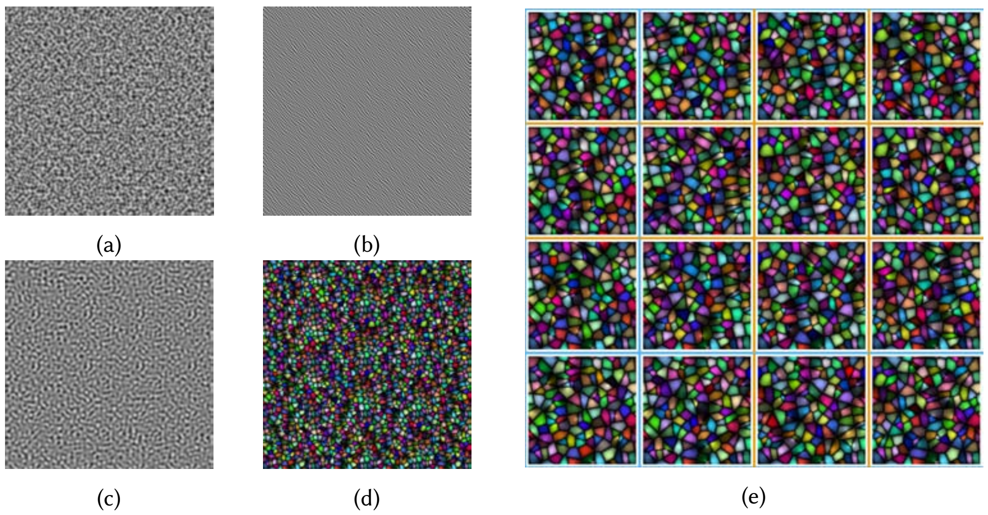


Fig. 5. Noise function Wang tile texture maps. (a) Perlin noise. (b) Anisotropic Gabor noise. (c) Better gradient noise. (d) Worley noise. (e) Worley noise with separated tiles. Note that the corresponding edges of the tiles that are marked with the same color have the same border.

simplify the definition and increase the performance of the function $T_{PerlinW}(x, y)$, we used 0 for all parameters except o_h and o_v, which were both set to w.

The algorithm that maps the tiles to a surface at runtime described by Wei [2004] provides a method to determine the edge colors of a tile given the input texture request coordinate. Using the lattice point coordinate local to the tile, (i_t, j_t), it is trivial to check to which group the current point belongs to, and modify the coordinate using the equation (1) accordingly. Our implementation of this function is given in the auxiliary material.

In order to obtain a Wang tile texture map using Perlin noise, we set the lattice size to be $(w \cdot 4 + 1) \times (w \cdot 4 + 1)$ points, where $w \times w$ is the size of a single tile. The parameter w is determined by the size of the resulting texture $w_t \times w_t$, and the pixel distance between two consecutive lattice points w_c: $w = \frac{w_t}{w_c \cdot 4}$. The resulting image (see Fig. 5a) is then obtained by sampling the original noise function at each pixel, with the function $T_{PerlinW}(x, y)$ used for determining the integer lattice coordinates.

The function $T_{PerlinW}(x, y)$ from the equation (1) can be used together with several other integer lattice noise functions without modification. We successfully applied it to modified noise and Perlin noise with the *xor* hash function introduced by Kensler et al. [2008].

3.2 Gabor Noise Tiling

The value of Gabor noise at the point (x, y) depends on the cell to which this point belongs, and its immediate neighbors. This means that the function $T_{PerlinW}(x, y)$ from the equation (1) cannot directly be used together with Gabor noise. We use a slightly modified version of this function to achieve the same result.

Similarly to the approach taken with Perlin noise, we subdivide all the grid cells into *boundary cells* and *inner cells*. The boundary cells are all the cells inside a tile that can potentially affect the value of the noise function at the tile edge. Lagae et al. [2009] discussed the truncation of the Gabor kernel to provide the possibility to evaluate the noise on the grid. The truncation limits the cells that influence the value at the tile edge to the cell this edge belongs to, and its immediate neighbors located at the tile edge.

We also chose a different grouping of the boundary cells for the Gabor noise. The cells form two groups: C_0 contains all the corner cells and all the border cells that comprise the tile borders of the first color, and C_1 contains all the border cells that belong to the edges of the second color. This subdivision simplifies the function definition and improves the performance.

Let the cell size be $w_c \times w_c$ and let the number of cells in a tile be $c_t \times c_t$. Let (i, j) be the coordinate of the grid cell which contains the point (x, y): $(i, j) = (\lfloor \frac{x}{w_c} \rfloor, \lfloor \frac{y}{w_c} \rfloor)$. Let (i_t, j_t) be the grid cell coordinate local to the tile: $(i_t, j_t) = (i \mod c_t, j \mod c_t)$. We define the function $T_{GaborW}(x, y)$ as a piecewise function,

$$T_{GaborW}(x, y) = \begin{cases} (i_t, j_t), & (i, j) \in C_0 \\ (i_t + c_t, j_t + c_t), & (i, j) \in C_1 \\ (i, j), & \text{otherwise} \end{cases} \tag{2}$$

As with the Perlin noise, during the application runtime a simple check determines which group the current cell belongs to. Our implementation of the function $T_{GaborW}(x, y)$ is given in the auxiliary material. An example of the resulting Wang tile texture map is shown in Fig. 5b.

During the preprocessing the size of the resulting texture $w_t \times w_t$, the amount of cells per Wang tile $n_c \times n_c$ and the cell size $w_c \times w_c$ are constrained by having to satisfy the following relationship: $w_t = n_c \cdot w_c \cdot 4$.

3.3 Tiling Worley and Better Gradient Noises

As mentioned in section 2.1.3, the implementation of Worley noise is very similar to that of Gabor noise. Both subdivide the space into a uniform grid, and a random number of positions are generated for each cell. The positions become the locations of feature points for Worley noise and the centers of Gabor kernels for Gabor noise.

The described similarities allow us to use the cell coordinate transformation function we defined for Gabor noise (equation 2) for Worley noise without modifications. An example Wang tile texture map is shown in Fig. 5d. The same texture broken down into individual tiles is displayed in Fig. 5e.

Better gradient noise [Kensler et al. 2008] represents a special class of integer lattice noises. The filter kernel used by this noise function covers 4×4 lattice points in 2D, while Perlin noise takes only the 4 closest lattice points into account. This property precludes us from using the function $T_{PerlinW}(x, y)$ with better gradient noise. At the same time, we observe a similarity between the definitions of the better gradient noise and the Gabor noise. Gabor noise is evaluated at the cell that contains the sample point and its immediate neighbors, and covers a 3×3 region.

This similarity allows us to use the cell coordinate transformation function we defined for Gabor noise (equation 2) for better gradient noise with a slight modification: the boundary points are comprised from two rows of lattice points instead of one.

4 TILED NOISE ON SURFACES

There are three methods that are commonly used to map a noise to a surface: explicit parameterization, solid noise sampling, and defining the noise directly on the surface [Lagae et al. 2010].

Explicit parametrization is most commonly used with regular textures. It allows fine-tuning, but requires additional memory and can introduce distortions and seams.

Solid noise is a 3D noise function that is sampled at each surface point. The objects textured using solid noise appear carved out of a block of matter. The memory cost remains low if the noise is not being precomputed.

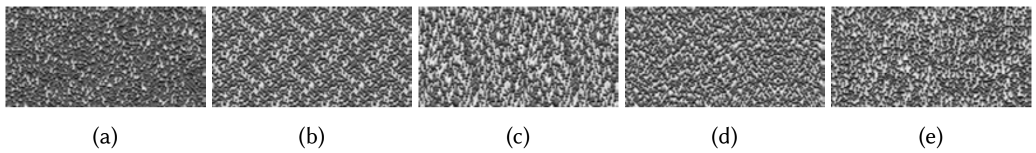


Fig. 6. Tile patterns produced by different tiling algorithms. Each image contains 128 × 64 color-coded Wang tiles. (a) Stochastic. (b) Permutation table, 32 entries. (c) Permutation table, 64 entries. (d) BBS hash, $K = 3, N = 201$. (e) MD5 hash.

Some noise functions allow us to define the noise directly on the surface, giving the appearance of the noise features following the curvature. Sparse convolution noises enable this by locally splatting the kernels onto the surface.

The definitions of the functions in equations (1) and (2) imply that the noise using them works with explicit surface parameterization and as solid noise, regardless of whether the noise is precomputed or evaluated at application runtime.

Surface noise obtained from sparse convolution noises is limited to runtime evaluation only. Obtaining surface noise from Gabor noise, for example, does not involve boundary handling, which makes our methods not applicable to surface noise.

When the noise is mapped to surfaces, special care must be taken to avoid aliasing on distant objects and on surfaces seen at an angle. While the precomputed noise can benefit from the existing texture filtering methods, noise functions that are evaluated at application runtime usually rely on certain properties of the noise to reduce aliasing.

Using mipmaps is one of the most common ways of filtering textures on distant surfaces in real-time applications. When dealing with textures containing tiles, we need to take into account the way the hardware performs the sampling while downscaling the images in order to minimize the discontinuities at the tile boundaries between two adjacent levels of detail.

Anisotropic filtering [McCormack et al. 1999] is another commonly used technique that increases the image quality on surfaces viewed at an oblique angle. It takes several samples in an elliptic area, producing a more accurate approximation of the pixel projection to the screen. When the filter area crosses the tile borders, it can produce incorrect values. All noises using the $T_{GaborW}(x, y)$ function introduce such errors only when the cell size is less than the size of the filter. The error is always present when sampling textures based on the $T_{PerlinW}(x, y)$ function. While some applications can neglect it, others may increase the width of the boundary region to remove the error.

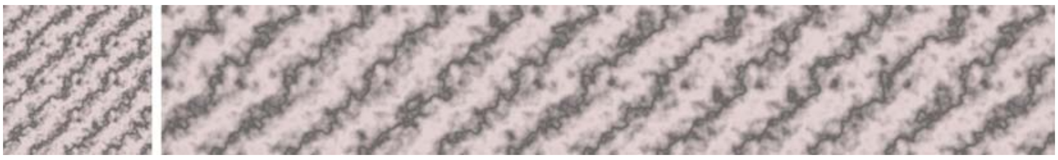
Our experiments show that anisotropic filtering hides the discontinuities introduced by general-purpose downscaling algorithms used to construct the mipmaps, even in high-contrast images, and that a combination of anisotropic filtering and mipmaps produces the best visual results.

The boundary handling functions described in section 3 do not modify the underlying noise function. It follows that filtering the procedural noise functions that use the presented modifications can be done using the methods that are applicable to the original noise functions. Read Lagae et al. [2010] for details on these techniques.

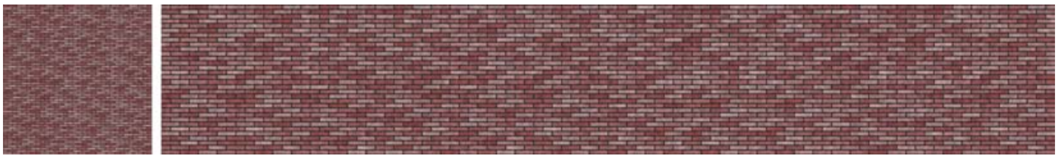
5 TILING IMPROVEMENTS

Wei [2004] proposed the use of toroidal boundary handling to compute the tile index from the input texture coordinate. The tile coordinates are then used as inputs to a hash function based on a small permutation table, which determines the edge colors of the resulting tile.

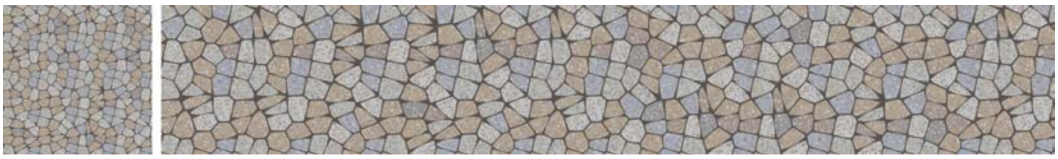
The permutation table size, however, either grows linearly together with the size of the output texture, or causes repetition of the tile pattern. While the amount of memory required for the



(a) A marble texture. Obtained by evaluating and color mapping $|\sin(K_1 + K_2 \cdot T)|$, where T is the result of blending several octaves of Perlin noise (turbulence).



(b) A brick wall texture. Obtained by multiplying a Wang tile brick wall texture map by a turbulence.



(c) A cobblestone texture. Obtained by color mapping two Worley noises with different parameters.

Fig. 7. Some interesting Wang tile texture maps produced using the presented techniques (left) and the corresponding tilings (right).

permutation table is very modest by modern standards, it has to be kept in very fast memory to provide efficient access. Our experiments show that using the same hash function as Olano [2005] produces acceptable results and is very fast to evaluate. The hash value is calculated by offsetting the input by a fixed number K, and applying the Blum Blum Shub [1986] PRNG to the result twice. Because modern GPUs have full support for integer operations, it is no longer necessary to constrain the PRNG parameters to avoid precision loss in floating-point numbers.

Applications that require higher periods of tile repetition and can afford a more expensive hash function would benefit from using cryptographic hashes. They offer statistical randomness of the result even if evaluated in parallel (as opposed to many classical PRNGs), and do not depend on the evaluation order. An example of such a hash function is the implementation of an MD5 hash for GPUs by Tzeng and Wei [2008].

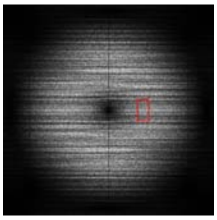
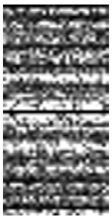
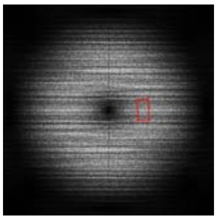
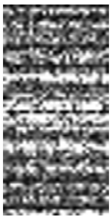
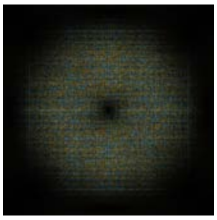
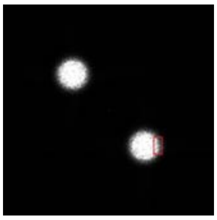
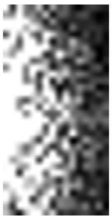
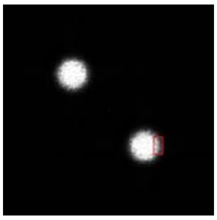
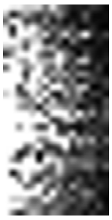
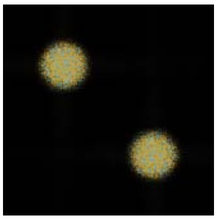
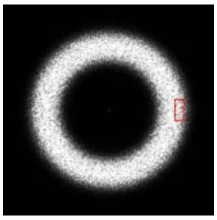
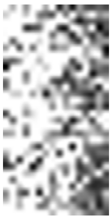
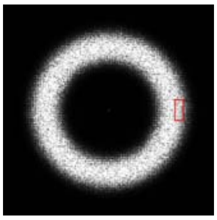
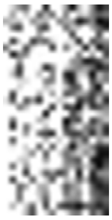
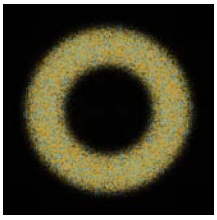
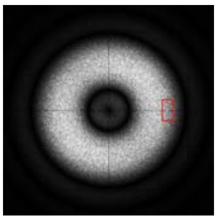
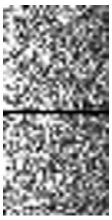
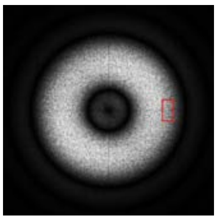
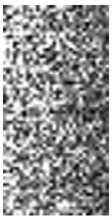
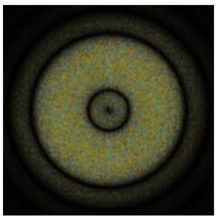
Fig. 6 shows the tile patterns produced by these tiling methods and the stochastic tiling algorithm proposed by Cohen et al. [2003].

6 RESULTS

Ken Perlin showed that his noise function can be used as a building block for creation of realistic-looking textures [Perlin 1985]. Similar results can be achieved with the noise functions modified as proposed in this paper (see Fig. 7a for an example).

Additionally, we would like to make an observation that once a function F is applied to a Wang tile texture, the result remains a Wang tile texture with the same layout if F depends on the pixel color, or if it depends on the pixel position and forms a Wang tile texture with the same layout. This class of functions includes, for example, linear combinations of Wang tile textures and periodic functions, with the period being a multiple of the Wang tile size. We show some textures generated with our application in Fig. 7.

Table 1. Periodograms of the noise functions. (a) No tiling. (b) Zoomed in part of (a). (c) Wang tile texture map. (d) Zoomed in part of (c). (e) Difference enhanced for better visibility. (1) Perlin noise. (2) Anisotropic Gabor noise. (3) Isotropic Gabor noise. (4) Better gradient noise.

(1)					
(2)					
(3)					
(4)					
	(a)	(b)	(c)	(d)	(e)

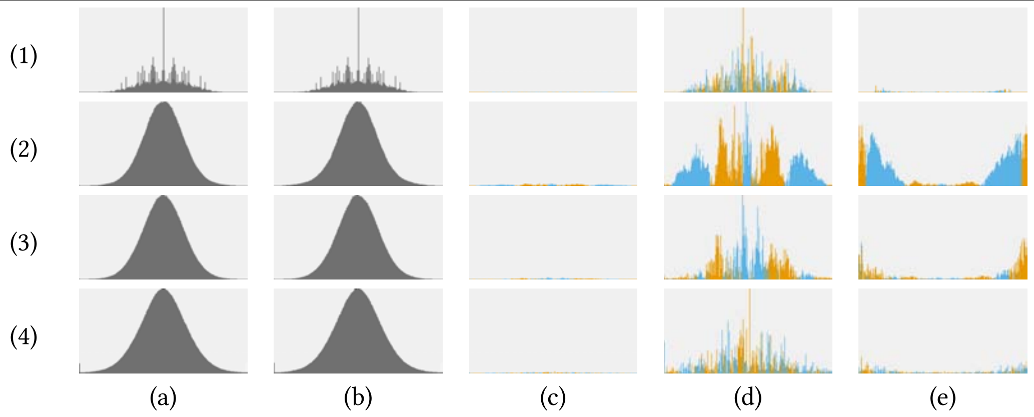
7 DISCUSSION

In this section we analyze the effect of the proposed boundary handling functions on the characteristics of Perlin noise, better gradient noise and isotropic and anisotropic Gabor noise, analyze the performance impacts on both CPU and GPU, and discuss the limitations of the presented methods.

7.1 Effect on Noise Characteristics

Power spectrum. We analyze the effect of employing the proposed boundary handling functions on the noise function power spectrum using periodograms. The periodograms are defined as the squared magnitude of the Fourier transform and are frequently used to estimate the power spectrum. To help visualize the differences between two periodograms, we show the linear difference between them. The sign of the difference controls whether its absolute value is encoded as orange color (positive difference) or blue color (negative difference) in the final image.

Table 2. Histograms of the noise functions. (a) No tiling. (b) Wang tile texture map. (c) Difference. (d) Normalized difference. (e) Difference normalized per bin. (1) Perlin noise. (2) Anisotropic Gabor noise. (3) Isotropic Gabor noise. (4) Better gradient noise.



The periodograms of the Wang tile texture maps are more clumped at the edges. This indicates that the noise in the Wang tile textures becomes less random. This is especially visible on Gabor noise and better gradient noise, because we duplicate the grid cells that belong to the border in order to achieve the required tiling characteristics. The effect is much less pronounced on Perlin noise, because pixels further away from the border are affected less due to interpolation.

The power spectrum shape stays the same for all three noise functions. It follows that the introduced methods do not alter the band-limitation characteristics of the original noise.

Amplitude distribution. We use histograms of the noise values to analyze the effect of the presented boundary handling functions on the noise amplitude distribution. In order to facilitate the amplitude distribution comparison, we provide difference images of the histograms, color-coded in the same way as the difference images of the periodograms. We provide three histogram difference images per noise function: one with the difference value divided by the largest histogram bin value, another with the difference normalized per bin, and one with the normalized difference value.

The strongest amplitude distribution distortion happens in the areas of the histogram that correspond to the amplitudes that are the rarest. The normalized difference values with the largest magnitudes appear closer to the middle of the histogram, which are at the same time the areas with the most frequently encountered amplitudes. The shape of the histograms is nearly unaffected by the proposed modifications. The effect is again more pronounced on Gabor noise and better gradient noise for the same reasons as described in the power spectrum analysis section.

While the differences in the power spectrum and in the amplitude distribution between the original and the noise with the proposed modifications are explicit, the noises are visually indistinguishable.

7.2 Performance analysis

We conducted a series of CPU and GPU tests to analyze the impact of the proposed modifications on the performance. The test environment description and tables with the detailed performance test results are available in the auxiliary material.

CPU performance. For the CPU performance test we provided two algorithms that compute the values of the Perlin noise and the better gradient noise. The default implementation samples the function as usual, both with non-tileable and Wang tiles boundary handling functions. An optimized implementation first precomputes the lattice gradients, then uses them to efficiently calculate the values of the final function.

The default implementations of both lattice-based noise functions with boundary handling that produce Wang tiles are several times slower than the original function. Still, the amount of time it takes to sample the functions once per pixel in a 2048×2048 texture is relatively small, indicating that they are very cheap to evaluate. The optimized implementations require additional memory, but make the performance of both the original and the modified function equal. The optimized Perlin noise implementation is also faster than the default implementation by more than 20%.

The evaluation of the anisotropic Gabor noise with zero Gabor kernels per cell closely estimates the performance cost of using the modified noise function. It is around 40% higher than that of the original function. The evaluation of additional Gabor kernels quickly reduces the weight of the boundary function evaluation to values close to zero in the overall function sampling duration.

GPU performance, precomputed textures. Sampling a Wang texture once per pixel carries very little overhead. In this case the performance drops insignificantly, by less than 2%. Sampling a texture 16 times per pixel makes the difference more noticeable, up to about 20%. Our tests showed that enabling trilinear and anisotropic filtering did not affect the performance.

GPU performance, direct noise evaluation. Finally, we conducted a performance comparison between the original and the modified versions of the noise functions, when the evaluation happens directly on the GPU at application runtime. The cost of sampling a modified noise function can be up to 10% higher than the cost of sampling the original function.

Overall, the results of the performance comparison tests indicate that the modifications presented in this paper can be adopted by many applications without having a major effect on the frame rate. Sampling a precomputed Wang tile texture is roughly equivalent in performance to evaluating Perlin noise with simple tiling in the shader, and is at least several times faster than evaluating noises that offer higher quality.

7.3 Limitations

As mentioned in section 4, special downscaling algorithms are required for Wang tile textures in order to minimize errors on tile borders between adjacent mip levels. The same applies to block texture compression methods. Anisotropic filtering, however, visually hides the discontinuities arising from texture compression as well.

Using a Wang tile texture within a texture atlas presents challenges very similar to those arising when using texture atlases with wrapping modes other than clamping. A detailed discussion is available in NVIDIA [2004]. If the Wang tile textures are of the same size, texture arrays can be utilized on the hardware that supports them.

Sampling a Wang tile texture requires using a version of the texture sampling function with explicitly provided gradients. Older or low-power hardware used in some mobile devices may lack support for gradient computation. Additionally, some old GPUs in mobile devices undergo a performance penalty when the texture coordinates used to sample a texture are modified in the fragment shader. This can be mitigated in some cases by adjusting the mesh UV coordinates in a way that would enforce all tile corners to correspond to mesh vertices, and by moving the texture coordinate calculation to the vertex shader.

The presented algorithms do not solve certain inherent problems of Wang tiles. For example, when the resulting tiles have large-scale distinct features, the tiling pattern becomes quite obvious.

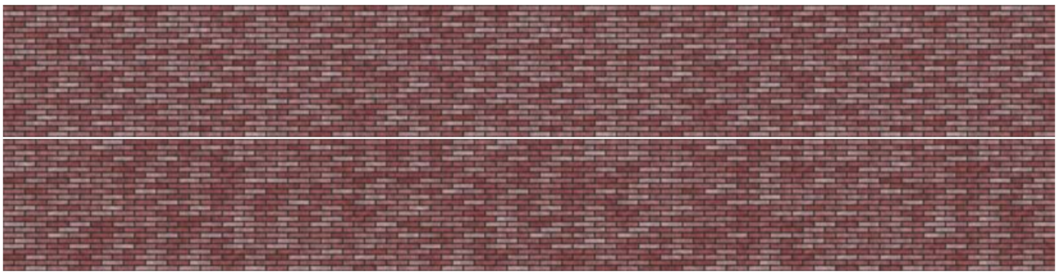


Fig. 8. A comparison between simple tiling of a 1024×1024 texture (top) and Wang tiling of a 512×512 (16 tiles, 128×128 pixels each) Wang tile texture map (bottom). The latter is non-periodic and requires 4 times less memory.

Hash-based evaluation of tile edge colors requires a complete set of Wang tiles to operate. The amount of tiles in the set grows very quickly when adding an edge color or increasing the number of dimensions. A full 2D set contains N^4 tiles, where N is the number of colors. This significantly increases the memory requirements for the precomputed textures. A tile index texture map can be employed to reduce the memory occupied by the tile set by including only those tiles that are actually used. This, however, introduces additional complexity to the tile packing step, and adds a texture read to the shader.

Another difficulty with Wang tiles is that they do not take into account tile corners, which may result in a discontinuity in the resulting image. Our boundary handling functions are defined in such a way that all the tile corners are the same, and thus are guaranteed to not produce any discontinuities. Alternatively, we can employ corner tiles [Lagae and Dutré 2006], which solve this problem by imposing restrictions on the diagonal tile neighbors in addition to the horizontal and vertical ones. Our methods are easily extensible to support corner tiles.

8 CONCLUSION

We presented a set of modifications to several popular procedural noise functions that directly produce texture maps containing the smallest complete set of Wang tiles. The proposed modifications can be used both at application runtime and during the preprocessing steps and can be generalized to higher dimensions and Wang tile sets with more edge colors.

We showed that the modified noise functions retain most of the key characteristics of the original functions. We discussed the effect of using the proposed modifications on noise function filtering and the mapping of noise functions to surfaces. Additionally, we presented an improvement of the tiling algorithm on the GPU for Wang tiles.

Our modifications enable non-periodic tiling for a large range of noise-based procedurally generated textures and effects. The presented techniques can be used to produce large non-repetitive and detailed terrain geometry, atmospheric effects and realistic-looking natural and artificial textures. The option to combine the Wang tile texture maps while maintaining tile layout increases the diversity of the possible results even further.

The performance tests indicate that the proposed techniques can be adopted even by high-performance interactive real-time applications. The presented methods offer a potential to decrease the memory consumption of precomputed noise-based textures, and enable precomputation for large noise-based textures that would not fit into the memory budget of an application otherwise, without sacrificing the final image variance (see Fig. 8). The Wang tile texture map sampling is also straightforward to implement. This guarantees smooth integration into both existing and new applications.

9 FUTURE WORK

There are several interesting directions for future work. One is to investigate the possibility of defining boundary handling functions similar to the ones presented in this paper for a wider range of procedural noise functions; namely wavelet noise [Cook and DeRose 2005] and anisotropic noise [Goldberg et al. 2008]. Adding the functionality to handle corner tiles and higher dimensions would broaden the possible applications of non-periodically tiled noise functions.

Introduction of an algorithm that minimizes the discontinuities on the tile borders between the adjacent levels of detail when downscaling the Wang tile textures would be beneficial for applications that strive to provide high-quality rendering results.

REFERENCES

Robert Berger. 1966. The undecidability of the domino problem. *Memoirs of the American Mathematical Society* 66 (1966), 1–72.

L. Blum, M. Blum, and M. Shub. 1986. A Simple Unpredictable Pseudo-Random Number Generator. *SIAM J. Comput.* 15, 2 (1986), 364–383. https://doi.org/10.1137/0215025

Michael F. Cohen, Jonathan Shade, Stefan Hiller, and Oliver Deussen. 2003. Wang Tiles for Image and Texture Generation. *ACM Trans. Graph.* 22, 3, 287–294. https://doi.org/10.1145/882262.882265

Robert L. Cook and Tony DeRose. 2005. Wavelet Noise. *ACM Trans. Graph.* 24, 3, 803–811. https://doi.org/10.1145/1073204.1073264

Karel Culik, II. 1996. An Aperiodic Set of 13 Wang Tiles. *Discrete Math.* 160, 1-3 (Nov. 1996), 245–251. https://doi.org/10.1016/S0012-365X(96)00118-5

Sebastien Deguy, Rogelio Olguin, and Brad Smith. 2016. Texturing Uncharted 4: a Matter of Substance. GDC Vault. https://www.gdcvault.com/play/1023488/Texturing-Uncharted-4-a-matter

Alexei A. Efros and William T. Freeman. 2001. Image Quilting for Texture Synthesis and Transfer. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '01)*. ACM, New York, NY, USA, 341–346. https://doi.org/10.1145/383259.383296

Alexander Goldberg, Matthias Zwicker, and Frédo Durand. 2008. Anisotropic Noise. *ACM Trans. Graph.* 27, 3, Article 54, 8 pages. https://doi.org/10.1145/1360612.1360653

Emmanuel Jeandel and Michael Rao. 2015. An aperiodic set of 11 Wang tiles. https://arxiv.org/pdf/1506.06492.pdf

Jarkko Kari. 1996. A Small Aperiodic Set of Wang Tiles. *Discrete Math.* 160, 1-3 (Nov. 1996), 259–264. https://doi.org/10.1016/0012-365X(95)00120-L

Andrew Kensler, Aaron Knoll, and Peter Shirley. 2008. Better Gradient Noise. SCI Institute Technical Report No. UUSCI-2008-001. https://www.cs.utah.edu/~aek/research/noise.pdf

Johannes Kopf, Daniel Cohen-Or, Oliver Deussen, and Dani Lischinski. 2006. Recursive Wang Tiles for Real-time Blue Noise. *ACM Trans. Graph.* 25, 3 (July 2006), 509–518. https://doi.org/10.1145/1141911.1141916

Ares Lagae and Philip Dutré. 2005. A Procedural Object Distribution Function. *ACM Transactions on Graphics* 24, 4 (October 2005), 1442–1461. https://doi.org/10.1145/1095878.1095888

Ares Lagae and Philip Dutré. 2006. An Alternative for Wang Tiles: Colored Edges Versus Colored Corners. *ACM Trans. Graph.* 25, 4 (Oct. 2006), 1442–1459. https://doi.org/10.1145/1183287.1183296

Ares Lagae, Sylvain Lefebvre, Rob Cook, Tony DeRose, George Drettakis, David S. Ebert, John P. Lewis, Ken Perlin, and Matthias Zwicker. 2010. State of the Art in Procedural Noise Functions. In *EG 2010 - State of the Art Reports*, Helwig Hauser and Erik Reinhard (Eds.). Eurographics, Eurographics Association, Norrkoping, Sweden.

Ares Lagae, Sylvain Lefebvre, George Drettakis, and Philip Dutré. 2009. Procedural Noise Using Sparse Gabor Convolution. *ACM Trans. Graph.* 28, 3, Article 54, 10 pages. https://doi.org/10.1145/1531326.1531360

J. P. Lewis. 1989. Algorithms for Solid Noise Synthesis. *SIGGRAPH Comput. Graph.* 23, 3, 263–270. https://doi.org/10.1145/74334.74360

Joel McCormack, Ronald Perry, Keith I. Farkas, and Norman P. Jouppi. 1999. Feline: Fast Elliptical Lines for Anisotropic Texture Mapping. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '99)*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 243–250.

Fabrice Neyret and Marie-Paule Cani. 1999. Pattern-based Texturing Revisited. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '99)*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 235–242. https://doi.org/10.1145/311535.311561

NVIDIA. 2004. Improve Batching Using Texture Atlases. NVSDK 7.0 Whitepaper. http://download.nvidia.com/developer/NVTextureSuite/Atlas_Tools/Texture_Atlas_Whitepaper.pdf

Marc Olano. 2005. Modified Noise for Evaluation on Graphics Hardware. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware (HWWS '05).* ACM, New York, NY, USA, 105–110. https://doi.org/10.1145/1071866.1071883

Roger Penrose. 1974. The role of aesthetics in pure and applied mathematical research. *Bulletin of the Institute of Mathematics and its Applications* 10 (1974).

Ken Perlin. 1985. An Image Synthesizer. *SIGGRAPH Comput. Graph.* 19, 3, 287–296. https://doi.org/10.1145/325165.325247

Ken Perlin. 2002. Improving Noise. *ACM Trans. Graph.* 21, 3, 681–682. https://doi.org/10.1145/566654.566636

Jos Stam. 1997. Aperiodic texture mapping. Tech. rep., R046. European Research Consortium for Informatics and Mathematics (ERCIM). http://www.dgp.toronto.edu/people/stam/reality/Research/pdf/R046.pdf

Stanley Tzeng and Li-Yi Wei. 2008. Parallel White Noise Generation on a GPU via Cryptographic Hash. In *Proceedings of the 2008 Symposium on Interactive 3D Graphics and Games (I3D '08).* ACM, New York, NY, USA, 79–87. https://doi.org/10.1145/1342250.1342263

Hao Wang. 1961. Proving theorems by pattern recognition II. *Bell Systems Technical Journal* 40 (1961), 1–42.

Li-Yi Wei. 2004. Tile-based Texture Mapping on Graphics Hardware. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware (HWWS '04).* ACM, New York, NY, USA, 55–63. https://doi.org/10.1145/1058129.1058138

Li-Yi Wei and Marc Levoy. 2000. Fast Texture Synthesis Using Tree-structured Vector Quantization. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '00).* ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 479–488. https://doi.org/10.1145/344779.345009

Guillaume Werle and Benoit Martinez. 2017. Ghost Recon Wildlands, Terrain Technology and Tools. GDC Vault. https://www.gdcvault.com/play/1024029/-Ghost-Recon-Wildlands-Terrain

Steven Worley. 1996. A Cellular Texture Basis Function. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '96).* ACM, New York, NY, USA, 291–294. https://doi.org/10.1145/237170.237267