Eurographics Symposium on Rendering 2014
Wojciech Jarosz and Pieter Peers
(Guest Editors)

Volume 33 (2014), Number 4

Interactive Parameter Retrieval for Two-Tone Procedural Textures

L. Gieseke[1,2], S. Koch[1], J.-U. Hahn[2] and M. Fuchs[1]

[1]University of Stuttgart, Stuttgart, Germany
[2]Stuttgart Media University, Stuttgart, Germany

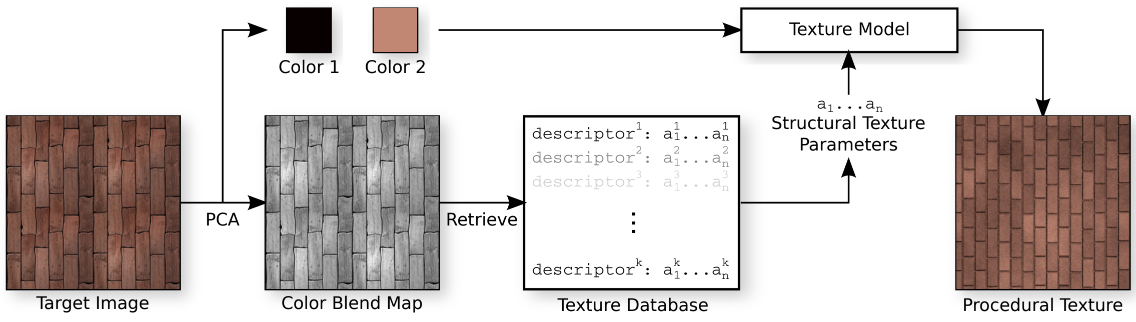

Color 1
Color 2
Texture Model
$a_1...a_n$
Structural Texture Parameters
descriptor^1: $a_1^1...a_n^1$
descriptor^2: $a_1^2...a_n^2$
descriptor^3: $a_1^3...a_n^3$
descriptor^k: $a_1^k...a_n^k$
PCA
Retrieve
Target Image
Color Blend Map
Texture Database
Procedural Texture

Figure 1: *Principal Component Analysis reveals the constituent colors of a two-tone image and a corresponding blend map. By identifying the most similar image from a database of images generated by a procedural texture model according to a texture descriptor, we can retrieve structural parameters. Together with the colors, they produce an image closely matching the input.*

Abstract

The choice of parameters for procedural textures to achieve a desired appearance poses a challenging problem even for experienced artists. We propose a method to automatically determine such parameters to reproduce the appearance of input images. Addressing two-tone textures, we separate the estimation of color and structure information and interpret the problem as image retrieval task from the space of procedural outputs. Applying a perceptually motivated image metric based on a texture descriptor enables us to precompute a comprehensive collection of possible parameter sets and yet achieve interactive retrieval performance. Our method supports a large variety of procedural texture models with a unified approach.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Color, shading, shadowing and texture I.3.m [Computer Graphics]: Miscellaneous—Image-based modeling

1. Introduction

Visually appealing models and high quality renderings are required for the construction of virtual worlds. However, manual modeling of details, especially regarding surface textures, becomes increasingly intractable as quality demands rise. Historically, two principal avenues have been explored to face this challenge: image-based modeling on the one hand, where natural images are used for highly realistic renderings at the cost of expensive recording and storage and limitations for editing, and procedural modeling on the other, which has most compact storage requirements but suffers at times from unwieldy parameter control. In this article, we investigate a fusion of the two principles, choosing the parameters of procedural models automatically so as to match the appearance of a natural input picture.

For an artist, the initial exploration of the parameters of a procedural model is tedious and often the parameter space appears unmanageable as the parameters might behave nonlinearly with overlapping effects. We intend to support this process by offering an artist to start with an automatically retrieved feasible parameter set for a procedural model, enabling the artist to fully focus on the creative task of finalizing the look. In designing our system, we have real-world production scenarios in mind, which require intuitive and real-time interaction mechanisms and robust setups. We show that our pipeline reasonably matches production textures (see Acknowledgments for image sources). Preliminary discussions with members of the movie industry have revealed interest in our work, for instance as a technique to support pre-visualization in production.

Our proposed technique, outlined in Figure 1, makes the following contributions: we match the input pictures to structural parameters while employing a perceptually motivated image distance metric calibrated to the end user of the technique. Aside from this calibration, our approach is configuration free and works robustly for very different texture model classes, which we demonstrate for noise textures, regular grids, and special-purpose texture models for tiled and wooden surfaces. Interpreting the problem as an image retrieval task permits to precompute databases of texture descriptors, which in turn enables interactive performance.

2. Related Work

In order to distinguish the different example-based texture synthesis techniques, we are briefly going to discuss texture categories and their procedural representations. We then relate our work to other example-based texture synthesis publications and review the different steps of our interactive parameter retrieval pipeline.

Lin et al. [LHW*06] define a texture spectrum that ranges from regular deterministic textures with distinguishable texture elements recurrently placed, to irregular placements, ending in purely stochastic textures. Most publications about example-based texture synthesis focus on one specific texture type, for example on stochastic texture models. In our pipeline there are no general restrictions on the structural spectrum of the matched textures.

One approach for example-based texture synthesis is data-driven, producing its output as an array of pixel data [HB95, EL99, WL00]. Wei et al. [WLKT09] present a comprehensive summary of such data-driven techniques, focusing on neighborhood-based texture synthesis applications, also in regard to dynamic and solid texture synthesis. Recently, Vanhoey et al. [VSLD13] contribute with an on-the-fly generation of non-periodic infinite texture data and height maps using multi-scale texture tiles. As all these techniques produce pixel data, they are fundamentally different to our approach of retrieving parameter sets for procedural textures: compact representations which can be evaluated independently per texel permitting parameteric control. The term procedural texture has been used with different meanings in a variety of texture synthesis approaches. We refer to Ebert et al.'s [EMP*02] characterization and understand procedural textures as mathematical texture models. Ebert et al. [EMP*02] give a valuable survey for such function based texture programs. Among other techniques, they discuss modelling based on noise functions, an approach thoroughly discussed by Lagae et al. [LLC*10]. They can either be used as texture models on their own or as a basis for further pattern generation. Perlin noise [Per85], for example, is a common basis function for procedural textures and Perlin's texture renderings demonstrate that many natural phenomena can easily be described in such a compact fashion.

Related work on example-based procedural texture synthesis techniques is distinguishable regarding the underlying texture models, which distance metrics are employed and how its parameters are controlled. For the class of stochastic texture models, the generating parameters can be observed by computational analysis of the query images. This has been used as a parameter control strategy for stochastic textures in a variety of techniques. Lagae at al. [LVLD10] compute weights for the different noise bands of a multi-resolution noise to match isotropic stochastic procedural textures. Galerne et al. [GLLD12] automatically adjust the parameters of bandwidth-quantified Gabor noise. Gilet et al. [GDG12] present a multiple kernel noise which they designed by defining the power spectral density. Successively decomposing and matching the noise frequencies enabled the creation of visually appealing procedural textures. All techniques above yield convincing results but, being based on purely stochastic textures, their expressiveness is limited.

Lefebvre and Poulin [LP00] transfer measured properties of images to corresponding parameters for procedural brick and wood textures. The algorithm takes a reference image, a binary mask, and the texture class as input and produces persuasive results for these two structural texture types.

Gilet and Dischler [GD10] apply a more general optimization strategy for choosing the parameters of a procedure. They minimize an image distance metric computed with a multi-resolution Gabor filter bank and a windowed Fourier transform with gradient descent. While their approach is limited to a specific class of stochastic textures, they can create volumetric representations from 2D input images. Bourque and Dudek [BD04] also employ distance metrics and non-linear optimization while allowing for the whole procedural texture spectrum when matching. Our work differs from theirs by reducing the search space by several dimensions: we are able to support more complex structural designs as we process a texture's color information independently, an idea loosely based on previously implemented color space transformations [HB95] [VSLD13]. Furthermore our work is distinguished by presenting a robust and generalized pipeline for any structural texture model. In Bourque and Dudek's work the user needs to select the distance metric and optimization strategy for each fitting task individually. Finally, we present a larger variety of results with average matching times in less than one second during run-time due to our retrieval technique while Bourque and Dudek's non-linear optimization cost around 12 minutes at that time.

Wu et al. [WDR13] also implement a retrieval-based core for their inverse bi-scale material design pipeline. They split the appearance design task into a search in pre-computed small scale geometry and material libraries, employing an overall non-uniformly weighted euclidean distance of the BRDF representations. We base our distance metric on a more abstract appearance feature vector by implementing a Gabor distance metric. Field [Fie87] show that the responses of a Gabor filter bank, with filters that differ in orientation, frequency and resolution, relate to the responses of the human visual system to image elements. Manjunath and Ma [MM96] introduce a robust image browsing and retrieval application based on a Gabor filter bank and a distance measure from the accumulation of the filter responses. They compared their implementation with other classification algorithms and concluded that Gabor filters show slightly better performance and retrieval accuracy.

3. Texture Model

We model a three-channel color texture

$$T_{\vec{a},\vec{c}_1,\vec{c}_2} : \mathbb{R}^2 \to \mathbb{R}^3, \quad \vec{x} \mapsto (r,g,b)^T \tag{1}$$

as convex combination of two RGB colors $\vec{c}_1, \vec{c}_2 \in [0,1]^3$. Each pixel position $\vec{x}$ may be expressed as

$$T_{\vec{a},\vec{c}_1,\vec{c}_2}(\vec{x}) = (1 - s_{\vec{a}}(\vec{x})) \cdot \vec{c}_1 + s_{\vec{a}}(\vec{x}) \cdot \vec{c}_2. \tag{2}$$

The structure function

$$s_{\vec{a}} : \mathbb{R}^2 \to [0,1] \subset \mathbb{R} \tag{3}$$

controls the blend between the two colors dependent on a parameter vector

$$\vec{a} \in \left([a_{\min,1}, a_{\max,1}], \ldots, [a_{\min,n}, a_{\max,n}]\right) \subset \mathbb{R}^n \tag{4}$$

and is implemented as a procedural texture. Thus, T is defined by the discrete choice of a structural function (manually selected by the user or automatically chosen by the algorithm), two color tones, $\vec{c}_1, \vec{c}_2$, and a structural parameter vector $\vec{a}$ depending on the concrete choice of the structural function. This formulation expresses colors using RGB values, which we found favorable for the intended place deep in the digital asset production pipeline. While perceptual uniformly scaled color spaces would conceivably improve predictive capabilities of perception, they would be most helpful only after lighting and color grading has already taken place.

By using two-tone texture models we assume that the input picture shows a two-tone texture also. This implies that the distribution of its color values in the RGB cube follows a straight line. A principal component analysis (PCA) of the color values reveals the line's location as the first principal component (the one corresponding to the eigenvector of the covariance matrix with the largest eigenvalue). The extremal pixel values of the input picture define its two constituent tones $\vec{c}_1$ and $\vec{c}_2$. An affine transformation maps the color with the smaller luminance value to 0 and the greater one to 1. If the luminance comparison fails, we use the three channels in lexicographical order for comparison. This mapping transforms the input picture to a gray valued structural image s_{target} and we can proceed with matching this structural information. The determined tones $\vec{c}_1$ and $\vec{c}_2$ constitute the final colors for the texture program.

For input images that deviate from the two-tone assumption, the pixel values are projected onto the straight line $\overline{\vec{c}_1\vec{c}_2}$. As the projected values may fall outside the RGB unit cube, we chose the intersection points of the line with the RGB cube as the two constituent colors.

4. Texture Distance Metric

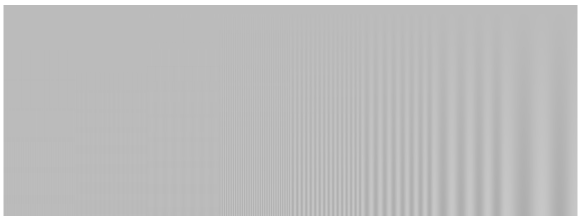
For our application, the choice of a suitable texture distance metric is crucial. A useful metric needs to abstract the texture input sufficiently while maintaining features relevant for a human observer, supporting fast evaluation. In mimicking human perception, both global statistics of an image, which relate to features such as overall brightness and contrast, as well as its frequencies and their distributions, which correspond to structural orientations, have been successfully used in the past [PCR11, MM96]. We formulate a metric incorporating both types into a descriptor vector which can be independently computed per input image and compactly stored.

As global statistical features of the structure functions in the observed domain, we employ the mean and standard deviations μ_{global} and σ_{global}. In order to model the distributions of frequency content in the structure maps, we employ a descriptor derived from a Gabor filter bank as used by Manjunath and Ma [MM96]. It is constructed from a family of harmonic, sinusoidal functions, weighted by a Gaussian distribution and extracts localized descriptions for differently oriented frequency bands. The response for one of the individual Gabor filters g_i is calculated by

$$R_i := \frac{\iint_{x,y \in A_i} |(I * g_i)(x,y)|\, dx\, dy}{|A_i| \cdot \iint_{x,y \in A_i} |g_i(x,y)| dx\, dy} \tag{5}$$

where A_i is the maximal circular inner image area within which the convolution with a filter on a given scale does not cross the image boundary. For fast evaluation, the convolutions are best performed in the frequency domain, as computed by the Discrete Fourier Transform [Smi97] [FJ05].

In our application, we employ sixteen different orientations and four different frequencies and an additional isotropic kernel on six filter scales each, except for the highest frequency on the lowest scale; this covers scales from the sampling limit up to the maximum filter size for which the kernels observe quasi-repetitive structures with the 256^2 textures we will use later on. This results in a total of 374 filter response images R_i.



0.60	0.96	0.99	0.99	1.00	0.99	0.95	0.94

Table 1: *The top 10% of the total test chart used to determine contrast weights, with the weights used reported below the corresponding sinusoids. The full chart is 4096 pixels wide and was used in actual size with scrolling.*

Intuitively, two images should have similar descriptor vectors if and only if the distributions of filter responses are similar. In order to achieve a compact representation, we model these histograms as Gaussian distributions, storing only arithmetical means μ_i and standard deviations σ_i.

For comparing the distributions we compute the W_2 Wasserstein distance, which in this case can be expressed compactly in closed form [GS84]. In the 1D case (not analyzing the covariance between different responses R_i, R_j), the comparison corresponds to evaluating a L_2 distance in the first two statistical moments [DL82] in a vector space with μ_i and σ_i in separate dimensions.

This motivates a m-dimensional descriptor vector for the image I to be assembled from the mean μ_i and standard deviations σ_i of the filter responses R_i interpreted as individual sets of numbers concatenated to the global values μ_{global} and σ_{global}, inducing a global L_2 distance metric

$$D(I_1, I_2) := \sqrt{w_{\text{global}}^2 \cdot d_{\text{global}}^2 + \sum_i w_i^2 \cdot d_i^2(I_1, I_2)} \qquad (6)$$

where d_{global} and d_i are the Wasserstein distances of the respective distributions (the square root being omitted in practice). D depends on weights which express the relative importance of the respective descriptor entries. These weights model the frequency-dependent contrast sensitivity of the human visual system [CR68]. As the specific weights depend on both the observer and on viewing conditions, they need to be calibrated individually in production. For our experiments, we have chosen w_i dependent on the scale of the Gabor filters proportional to the location of beginning contrast sensitivity as determined by a contrast test chart displayed on the monitor. Exact values and peaks vary between observers, but follow a similar distribution for different individuals. The weights used for all results in this article are reported in Table 1.

This leaves the balancing of local feature sensitivity vs. global brightness and contrast, as expressed by w_{global}. As

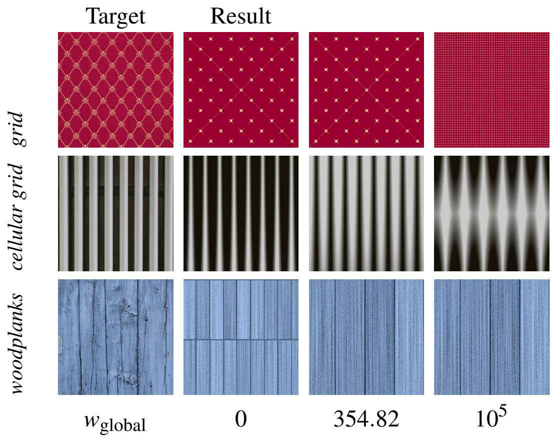


Figure 2: *Influence of w_{global}. Low values tend to underemphasize global statistics, large values put too little weight on local structure. For an intermediate value, a good compromise is achievable.*

Figure 2 shows, there is a trade-off to be made. For low values of w_{global}, local structure dominates the descriptor, and while orientations and edge densities are recovered well, average brightness is lost. For high values, the opposite effect occurs. Such a balancing of features also depends on production requirements. In order to provide a comparable evaluation, we used the same w_{global} for all texture classes. Following the intuition that global statistics and local structure are equally important, we chose $w_{\text{global}} := \sum_i w_i$, as the sum of weights for the descriptor elements representing local structure (354.82 in the reported results).

Note that, in contrast to full color texture descriptors, which would need at least three separate descriptor vector entries for each of the color channels – possibly more in order to resolve color correlation – our approach of explicitly mapping to a two-tone space remains 1-D and thus compact while being descriptive.

5. Retrieval

In the sense of the image distance defined above, the ideal choice of structural parameters $\vec{a}$ for a given input image s_{target} and a given texture model are simply those which result in the minimal distance, i.e. $\arg\min_{\vec{a}} D(s_{\text{target}}, s_{\vec{a}})$. The compactness of the Gabor texture descriptor vector and the pre-processing of the colors allow to fully pre-compute and store a map $\mathbb{R}^n \to \mathbb{R}^m$, encoding the descriptor vectors for a dense sampling of the parameter space $\mathbb{R}^n$. For moderately complex texture models, this sampling can even be exhaustive in practice. Where more than 10^6 entries are required, we instead apply random sampling of the parameter space.

As Gabor filter kernels for a given frequency are identical up to discretized rotation, this map also contains the de-

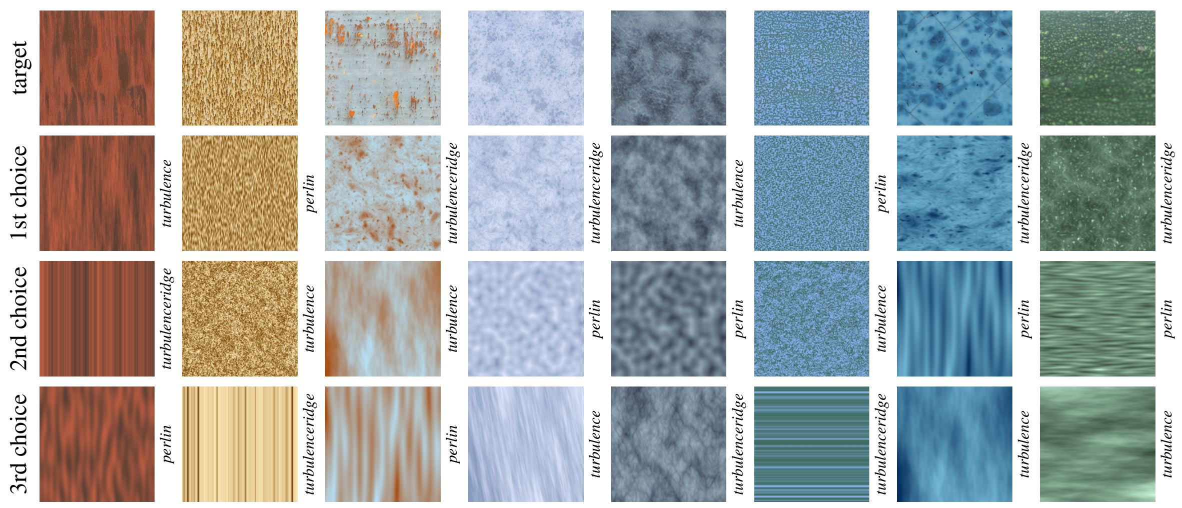


Figure 3: *Results with automatic model selection for noisy textures.*

scriptors of rotated versions of all structure images it stores descriptors for. Hence, while searching this map for the optimal match, we simultaneously search for rotated versions simply by permutating descriptor values during comparison, and we require no explicit sampling of texture rotations.

Most texture programs produce useful results when swapping the roles of the colors $\vec{c}_1$ and $\vec{c}_2$. Instead of encoding this behavior as a dimension of the structural parameter vector, we search for the descriptors of both s_{target} and $1 - s_{\text{target}}$, and, if the latter produces a solution with smaller distance, exchange $\vec{c}_1$ and $\vec{c}_2$, cutting storage requirements in half.

6. Results

We consider a usage scenario in which an artist is tasked with choosing a texture model and defining its procedural parameters for matching a reference picture (such as a photograph or drawing). Our approach intends to provide substantial support for this task by suggesting the best-matching parameters for several possible texture models from a preselected class. Within a class, the results are arranged in increasing distance D to the reference. The pre-selection of a model class is easy for the human artist to do and makes the descriptor more discriminative. As a consequence, its compact representation remains meaningful and comparable as the individual texture models constrain the search space.

Our results represent several common texture classes and the procedures follow standard texture models available in commercial modeling packages, such as Autodesk's Maya. In addition to six scalars for the two color tones, our implementations have one parameter rotation and additionally between one and seven structural parameters. We consider this representative, both in line with tools used in content production such as the count of structural parameters of Autodesk's Maya texture nodes and in research [LLD12a] [LLD12b]. Exemplary source code and a detailed description of parameter counts, value ranges, sampling densities and exemplary visualizations for each model are provided as supplementals. We have implemented texture models from these classes:

- **Noise** textures, including classic Perlin noise with added contrast control (*perlin*), *turbulence*, and turbulence with a ridge (*turbulenceridge*) [Per85]. They all support anisotropic stretch and global scaling, and are suitable for a large variety of surface materials, including concrete, wood, rust, clouds or even vegetable skin (Figure 3).
- **Grids** of regular structures, with models covering grids of variable spacing, edge sizes and edge softness (*grid*, *cellular grid*) (Figure 4).
- **Tiled** textures, comprising a semi-shaded turbulent surface type (such as occurs in a *brick* wall) and a texture mimicking hardwood floor (*woodplanks*) (Figure 5).
- Cut and polished **Wood** surfaces, implemented with varying base structures – *woodlines*, *woodplanks* and a one parameter *woodstreaks* model (Figure 6).
- **Rings** with turbulent lines of variable width, as in some wood and marble-like structures (Figure 7, *rings*).

Noisy queries (Figure 3) yield good results for each of the texture models in many cases. *Perlin* does not support isolated peaks, and *turbulenceridge* does not support very high frequencies. In these cases, anisotropic stretch is chosen as compromise, and the results are automatically downranked and better results from other models preferred.

As grid queries (Figure 4), we have used a mixture of targets intended to illustrate the behavior when abstraction of shapes is required. Neither of our grid texture implementations support diamond shapes; rotated rectangular grids are

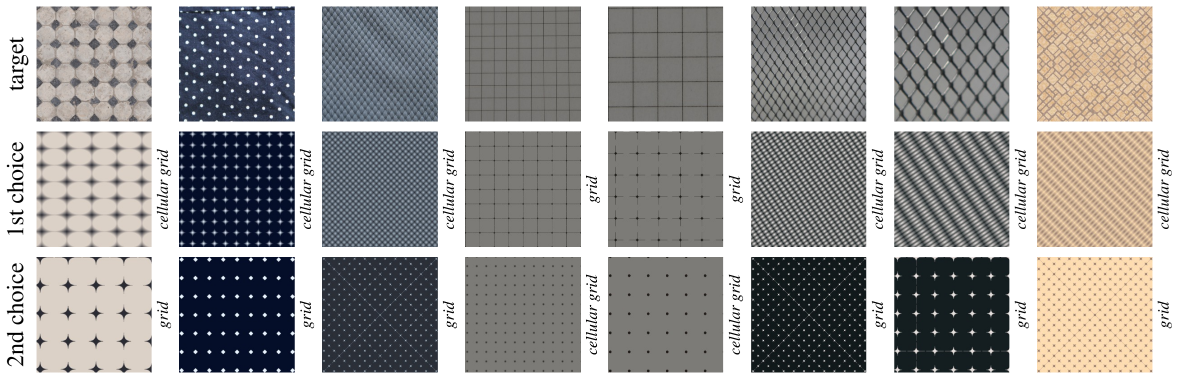


Figure 4: *Results with automatic model selection for grids.*

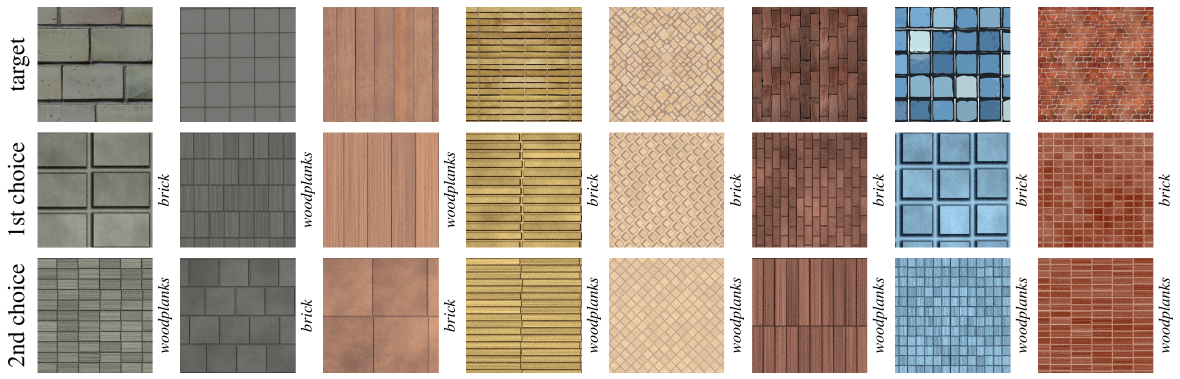


Figure 5: *Results with automatic model selection for tiles.*

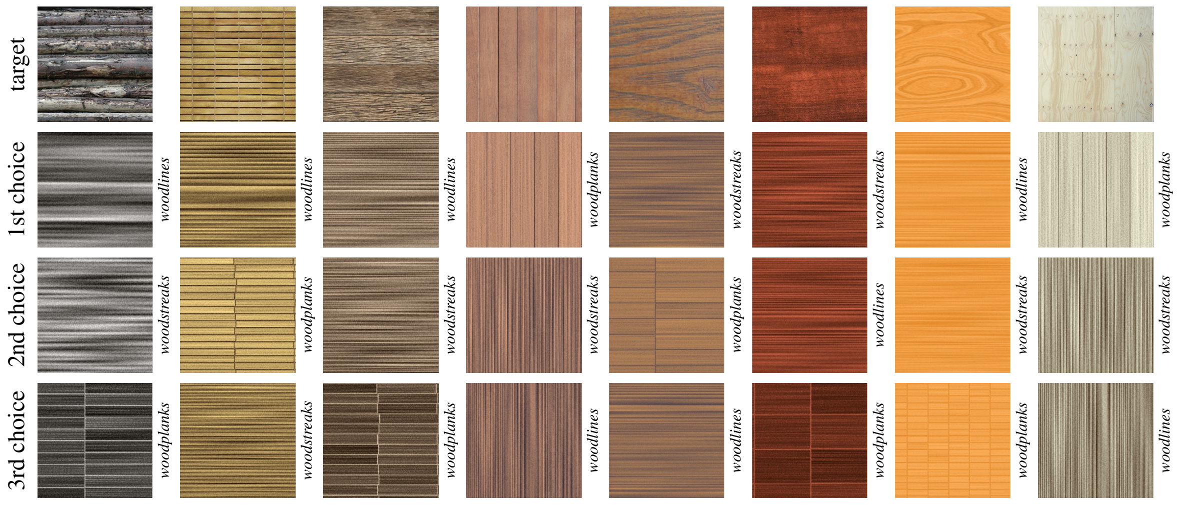


Figure 6: *Results with automatic model selection for wooden surfaces.*

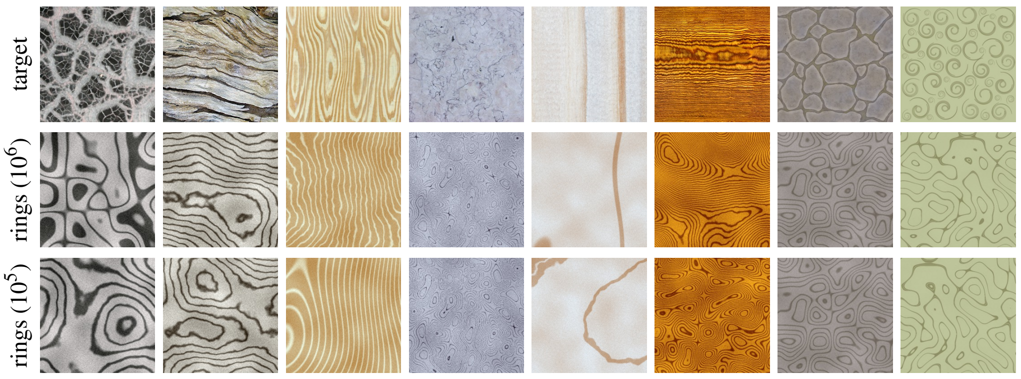


Figure 7: *Results for the texture class rings for different numbers of random samples in the database.*

chosen as best-effort match. Both the *grid* and *cellular grid* implementations have a large underlying parameter space to search in; *cellular grid* illustrates the behavior for stronger quantization of scales, *grid* demonstrates random sampling of the larger space; as a result, exact reproductions of scales may not be possible; however, the resulting scales are always similar to the scales in the target image, and would only require minimal modifications by an artist to finalize.

The tiled textures (Figure 5) *woodplanks* and *brick* implement models where spacing parameters need to be selected. In the presence of hard-coded design choices – not uncommon in production textures – a selection needs to be made between wood-like and stone-like surface appearance. Excepting the blue cartoon tiles, which are outside the appearance space of the texture models, grid spacings are robustly found. The optional phase-shift between rows and columns plays a less important role in the texture descriptor.

For the wooden examples (Figure 6), we have tested implementations with small parameter space volumes in addition to the wooden planks from the tiled textures. Even though *woodlines* has only two, and *woodstreaks* only one parameter which needs to be sampled, targets which are close to the appearance space of the textures are approximated well, and in those where additional structures are present (such as the fence or wooden planks), the orientation and scale of estimated structures provide plausible matches.

The *rings* texture (Figure 7) illustrates results for a complex texture, the parameter of space of which cannot be exhaustively precomputed, requiring random sampling. As can be seen, the results with 10^6 randomly sampled database entries match the structures of the target pictures closely. For an interactive application, a smaller sample with 10^5 entries results in fast approximations an artist could use interactively.

In order to test extreme mismatches between target query and texture model appearance space, we implemented a *symbol* texture model arranging symbols from the Wingdings font, and a *star* model showing regular arrangements of stars of variable number of tips. As Figure 8 shows, structures are still recognizable even when the appearance space of the model and the target have no overlap. Matching marguerites, we even get a lucky result of an arrangement of flowers.

In total, our results show that our technique is successful: for each requested target picture, the preferred match is a feasible approximation of the input, requiring at most minimal fine-tuning by an artist for optimal results (note that all results shown are the direct, automatic result of the algorithms as-is, though). Also, the selected texture model approximates the input best from the models in the same class.

Sizes and Timings

Table 2 shows the size of the precomputed database for the texture models we tested and the associated retrieval times. While implementing the texture models, we set limits for the scales so as not to generate structures which are either finer than the sampling limit or too large for multiple repetitions in the 256^2 pixel texture window we experimented with. We chose this size as we expect it to sufficiently cover procedural models with any practical number of parameters: higher resolutions would only be useful to define procedural models with both very fine (pixel scale) features and very large features (in excess of half of the image size), but are certainly possible, increasing the number of descriptors approximately linearly with the larger image dimension.

We only roughly distributed the parameter space perceptually uniformly, for instance, using exponential scales for structure sizes. An automatic scaling, such as investigated by Lasram et al. [LLD12b], is outside the scope of this article. We discretized the parameter space so as to create visible differences between stops on each parameter scale.

The precomputation time of the sampling ranges from one minute single thread CPU time for an exhaustive search on simple models to 1377 hours for a 10^6 random sampling. While we parallelized the precomputation – an easy optimization on industrial render farms – performance would

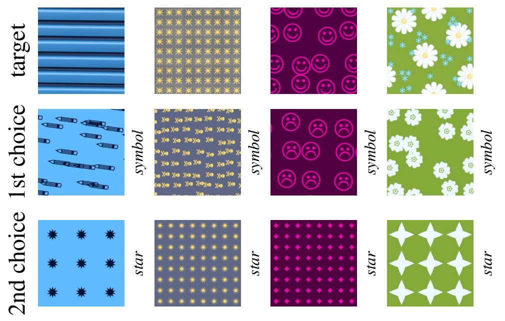


Figure 8: *Results for mismatches between query texture and supported model appearance.*

texture model	# entries	db size	time / s
brick	224 422	645.5 MB	0.80
cellular grid	101 871	293.0 MB	0.80
grid	10^6*	2876.3 MB	7.79
grid	10^5*	287.6 MB	0.8
perlin	12 221	35.1 MB	0.11
star	44	0.1 MB	< 0.01
symbol	16 830	48.3 MB	0.14
turbulence	23 331	67.0 MB	0.22
turbulenceridge	4 851	13.9 MB	0.07
woodlines	121	0.3 MB	< 0.01
woodplanks	20 402	58.6 MB	0.2
woodstreaks	11	< 1 MB	< 0.01
rings	10^6*	2887.7 MB	7.97
rings	10^5*	288.8 MB	0.80

Table 2: *Database dimension, size and retrieval times, single precision floating point values. Retrieval time excludes the computation of the target texture descriptor (about 0.6 s) and is reported for an Intel® Core™ i7-2600K CPU @ 3.40GHz. * denotes random sampling.*

further benefit from a GPU implementation. We do not consider these timings practical concerns, but rather one-time investments. The resulting databases fit in-core at runtime and a retrieval time of well below 1s for an exhaustive search of the entire parameter space for most textures is achieved.

7. Limitations and Future Work

Our sample-and-retrieve approach is fundamentally different from continuous methods. They may apply non-linear optimization and interpolation methods [BD04], and thus precisely find a local optimum in the parameter space. However, the numerical evaluation of the gradient of a cost function alone, involving several computations of an image difference function, may take more time than our entire retrieval step. Additionally a continuous approach needs at least locally continuous maps from the parameter into the distance function space. In choosing a discrete, sampled model, we believe to have found a compelling alternative.

When designing our texture descriptor we focused on maintaining relevant texture features while enabling fast evaluation. As a consequence, it does not reach the full expressivity which could be expected from state-of-the art texture analysis but follows our intent to construct a compactly storable descriptor vector. Modeling the distributions as Gaussians, disregarding phase sensitivity and covariances between the individual responses, may seem as a coarse oversimplification. Procedural models are, however, usually constrained in the effect of their parameters: a brick model, for instance, may offer expressive control on brick size, placement, and dimensions of mortar, but will not permit independent phase shifting of low and high frequencies at its edges. A representation of correlations between features on different scales and orientations would require scaling the descriptor vector length quadratically in the filter bank size, but could be expected to improve the automatic selection of texture models across the different texture classes.

A more general challenge in aligning automatic texture analysis with perceptual expectations of a human observer lies in semantic understanding: searching for a picture containing a happy smiley, we obtained the result shown in Figure 8: objectively, a good match for the input in terms of density, line structure, global and local contrast – but a bad match taking into account semantic associations of a human observer. Nonetheless, we believe to have found a useful balance of accuracy and efficiency.

Following the idea of pre-processing the color matching within our pipeline, it could be promising to estimate a continuous rotation before employing a structural texture descriptor. This might result in more accurate matches and faster retrieval performance.

Individually implemented texture metrics for cluster of texture types, combined in a common pipeline could enable the application to determine a texture class automatically. This would be useful for computing the layering of different texture models into one rendering, determining the influence of each texture model to match a target.

Many textures, which cover an application-relevant space, are representable with two-tone models – almost two-thirds of our target images come from online resources of the industry. Without general restrictions for the structural design of a texture model, our presented texture classes exemplify the large variety of classes we support. Nevertheless, an extension to the texture model could be investigated which still allows for the separation of color and structure, but also spans a colorful appearance space beyond two-tone textures.

Finally, it would be certainly worthwhile to integrate our technique into commercial rendering packages – Figure 9 shows a proof of concept that such an integration is feasible.

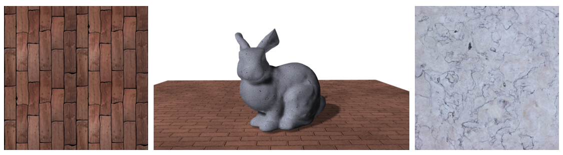


Figure 9: *3D scene rendered in Maya with procedural textures controlled by input pictures.*

8. Conclusion

In this work we have introduced an interactive parameter retrieval technique for parameter sets of two-tone procedural textures. Based on our achievements in improving the retrieval performance we believe to have come one step closer towards the actual programming of procedural textures by example, a goal we would like to pursue further.

9. Acknowledgements

This work was partially funded by the graduate program Digital Media, granted by the state Baden-Württemberg, Germany. Target images courtesy of (indices starting from the left) www.cgtextures.com (Figure 2 top and bottom, Figure 3 1.-4., 6., Figure 4 1., Figure 5 3.-4., Figure 6 1.-4., 6.-8., Figure 7 1.-4., 6., Figure 8 1.-2., Figure 9 2.), Toon Texture sets from www.3dtotal.com (Figure 1, Figure 4 8., Figure 5 5.-8., Figure 7 7., Figure 9 1.) and www.turbosquid.com (Figure 7 8., Figure 8 4.) or are photographs taken by the authors. Lastly, we would like to thank Andreas Schilling, Douglas Cunningham and the reviewers for valuable discussions.

References

[BD04] Bourque E., Dudek G.: Procedural texture matching and transformation. *Comput. Graph. Forum 23*, 3 (2004), 461–468. 2, 8

[CR68] Campbell F. W., Robson J. G.: Application of fourier analysis to the visibility of gratings. *J. Physiol. 197*, 3 (Aug. 1968), 551–566. 4

[DL82] Dowson D., Landau B.: The fréchet distance between multivariate normal distributions. *J. Multivar. Anal. 12*, 3 (1982), 450–455. 4

[EL99] Efros A. A., Leung T. K.: Texture synthesis by non-parametric sampling. In *Proc. IEEE ICCV* (1999), pp. 1033–1038. 2

[EMP*02] Ebert D. S., Musgrave F., Peachey D., Perlin K., Worley S.: *Texturing and Modeling: A Procedural Approach*, 3rd ed. Morgan Kaufmann Publishers Inc., 2002. 2

[Fie87] Field D. J.: Relations between the statistics of natural images and the response properties of cortical cells. *J. Opt. Soc. Am. 4*, 12 (1987), 2379–2394. 3

[FJ05] Frigo M., Johnson S.: The design and implementation of FFTW3. *Proc. IEEE 93*, 2 (2005), 216–231. 3

[GD10] Gilet G., Dischler J.-M.: An image-based approach for stochastic volumetric and procedural details. *Comput. Graph. Forum 29*, 4 (2010), 1411–1419. 2

[GDG12] Gilet G., Dischler J.-M., Ghazanfarpour D.: Multiple kernels noise for improved procedural texturing. *The Visual Comput. 28*, 6-8 (2012), 679–689. 2

[GLLD12] Galerne B., Lagae A., Lefebvre S., Drettakis G.: Gabor noise by example. *ACM TOG 31*, 4 (2012), 73:1–73:9. 2

[GS84] Givens C. R., Shortt R. M.: A class of Wasserstein metrics for probability distributions. *Michigan Math. J. 31*, 2 (1984), 231–240. 4

[HB95] Heeger D. J., Bergen J. R.: Pyramid-based texture analysis/synthesis. In *Proc. SIGGRAPH '95* (1995), Annual Conference Series, pp. 229–238. 2

[LHW*06] Lin W.-C., Hays J., Wu C., Liu Y., Kwatra V.: Quantitative evaluation of near regular texture synthesis algorithms. In *IEEE CVPR* (2006), pp. 427–434. 2

[LLC*10] Lagae A., Lefebvre S., Cook R., DeRose T., Drettakis G., Ebert D. S., Lewis J. P., Perlin K., Zwicker M.: State of the art in procedural noise functions. In *Eurographics 2010 - State of the Art Reports* (2010). 2

[LLD12a] Lasram A., Lefebvre S., Damez C.: Procedural Texture Preview. *Comput. Graph. Forum 31*, 2 (2012), 413–420. 5

[LLD12b] Lasram A., Lefebvre S., Damez C.: Scented sliders for procedural textures. *Proc. Eurographics* (2012). (Short). 5, 7

[LP00] Lefebvre L., Poulin P.: Analysis and synthesis of structural textures. In *Proc. Graphics Interface* (2000), pp. 77–86. 2

[LVLD10] Lagae A., Vangorp P., Lenaerts T., Dutré P.: Procedural isotropic stochastic textures by example. *Comp. & Graph. 34*, 4 (2010), 312–321. 2

[MM96] Manjunath B. S., Ma W. Y.: Texture features for browsing and retrieval of image data. *IEEE Trans. Pattern Anal. 18*, 8 (1996), 837–842. 3

[PCR11] Pouli T., Cunningham D. W., Reinhard E.: A survey of image statistics relevant to computer graphics. *Comput. Graph. Forum 30*, 6 (2011), 1761–1788. 3

[Per85] Perlin K.: An image synthesizer. *Comput. Graph. 19*, 3 (1985), 287–296. 2, 5

[Smi97] Smith S. W.: *The Scientist and Engineer's Guide to Digital Signal Processing*. California Technical Publishing, 1997. 3

[VSLD13] Vanhoey K., Sauvage B., Larue F., Dischler J.-M.: On-the-fly multi-scale infinite texturing from example. *ACM TOG 32*, 6 (2013), 208:1–208:10. 2

[WDR13] Wu H., Dorsey J., Rushmeier H.: Inverse bi-scale material design. *ACM TOG 32*, 6 (2013), 163:1–163:10. 3

[WL00] Wei L.-Y., Levoy M.: Fast texture synthesis using tree-structured vector quantization. In *Proc. SIGGRAPH '00* (2000), Annual Conference Series, pp. 479–488. 2

[WLKT09] Wei L.-Y., Lefebvre S., Kwatra V., Turk G.: State of the art in example-based texture synthesis. In *Eurographics 2009 - State of the Art Reports* (2009), pp. 93–117. 2