The Visual Computer
https://doi.org/10.1007/s00371-021-02212-4

ORIGINAL ARTICLE

Edge-based procedural textures

Hansoo Kim[1] · Jean-Michel Dischler[2] · Holly Rushmeier[3] · Bedrich Benes[1]

Accepted: 15 June 2021

Abstract
We introduce an edge-based procedural texture (EBPT), a procedural model for semi-stochastic texture generation. EBPT quickly generates large textures from a small input image. EBPT focuses on edges as the visually salient features extracted from the input image and organizes into groups with clearly established spatial properties. EBPT allows the users to interactively or automatically design new textures by utilizing the edge groups. The output texture can be significantly larger than the input, and EBPT does not need multiple textures to mimic the input. EBPT-based texture synthesis consists of two major steps, input analysis and texture synthesis. The input analysis stage extracts edges, builds the edge groups, and stores procedural properties. The texture synthesis stage distributes edge groups with affine transformation. This step can be done interactively or automatically using the procedural model. Then, it generates the output using edge group-based seamless image cloning. We demonstrate our method on various semi-stochastic inputs. With just a few input parameters defining the final structure, our method can analyze the input size of 512×512 in 0.7 s and synthesize the output texture of 2048×2048 pixels in 0.5 s.

Keywords Texture synthesis · Procedural modeling · Image analysis

1 Introduction

Texture synthesis has been used in various traditional computer graphics areas such as digital art, gaming, real-time applications, and visualization, as well as in scientific areas that aim at understanding the underlying principles of composition and appearance of materials and structures [6].

Texture synthesis algorithms aim to provide expressive yet straightforward methods that are easy-to-control and generate large expanses of texture quickly. Existing approaches can be classified into example-based, procedural, and simulation-based. Example-based methods use existing textures to create new ones. Procedural synthesis uses algorithms and control parameters to generate a texture. Simulation-based approaches mimic appearance producing processes observed in nature.

Despite progress in texture synthesis, many challenges remain unsolved. Example-based methods are limited to the provided examples and suffer from limited user control. Moreover, large quantities of data are needed to meet user requirements and train a deep neural network. They also do not always generalize across different examples, and combining examples can be problematic. Procedural methods are difficult to control because the input parameters do not have apparent interpretations, are nonlinear, and have intricate interdependencies. The user often uses a trial-and-error approach to generate the output. Simulations may require the user to have a good understanding of the modeled phenomena. They are usually slow, do not offer extensive output data, and do not allow the user to affect the intended output appearance directly.

Inverse procedural modeling [1] harnesses the power of procedural modeling by finding descriptions of processes that generate structures and their parameters. It is similar to generation by-example in that it uses existing structures, yet the goal is finding an underlying procedural representation of the input. Inverse procedural models have been successful in generating regular structures. Ran-

✉ Hansoo Kim
hansookim@google.com

Jean-Michel Dischler
dischler@unistra.fr

Holly Rushmeier
holly.rushmeier@yale.edu

Bedrich Benes
bbenes@purdue.edu

[1] Purdue University, West Lafayette, IN, USA

[2] University of Strasbourg, Strasbourg, France

[3] Yale University, New Haven, CT, USA

Published online: 14 July 2021

dom structures were addressed using noise or noise-similar basis functions [21], but the scope of these functions is limited.

Our key observation is that *texture synthesis can be expressed as a controllable inverse procedural problem*. An input image can be classified into visually important (salient) features (edges) [36] that can be stored as parameterized procedural building blocks. Our *edge-based procedural texture* (EBPT) bridges the gap between example-based texture synthesis and inverse procedural modeling. EBPT consists of groups of edges, and their mutual relationship can be thought of as terminal symbols of the procedural generation. The relationship is the parameter set of the EBPT. The input texture image is analyzed, and the edges are encoded into edge groups based on their spatial properties. The edge groups and the input texture are used to synthesize the output texture. The output texture is outlined by positioning the edge groups (manually or automatically). Then, seamless image cloning [38] completes the output pixels. By storing the salient features and encoding their properties, we provide a method for synthesizing structures similar to the input. A set of simple input parameters can efficiently and intuitively control the process.

Figure 1 shows our approach. The input exemplar is analyzed, and the EBPT is generated. The final image is synthesized using only a few representative edge groups and is completed by pixel filling. Several variants can be generated from a relatively small input sample. Our main contributions are: (1) a novel procedural model called EBPT for texture representation as a set of parameterized procedural building blocks called edge groups that are based on visually important features (edges), (2) encoding of edges and groups of edges, (3) an interactive algorithm for texture synthesis by utilizing the edge groups, and (4) a set of controls that allow the transformation and combination of multiple exemplars at a fine level to design new textures, rather than simply replicating them.

2 Related work

Our work is related to inhomogeneous textures computed from examples using a prior extraction of salient features, as well as to inverse procedural modeling.

Inhomogeneous textures A large body of work has appeared in the past two decades in the field of by-example texture synthesis—too many to be thoroughly enumerated here—and we refer the reader to reviews that describe previous work [3,46]. By consensus, optimization techniques, as first introduced in [30], provide the best state-of-the-art results, as long as textures are homogeneous. Inhomogeneous or highly structured textures still raise important issues. A prior analysis step is needed to provide either a smart initial guess [28] or guidance maps [54]. However, it is generally difficult to extract and segment "meaningful" structure-related information (e.g., a feature or distance map) fully automatically. The complexity of the corresponding objective function formulation also leads to time-consuming synthesis and allows only limited control compared to procedural texture synthesis.

Edges, ridges, and more generally curvilinear features provide major human visual cues [36] as opposed to cues from surfaces with small gradient changes [4]. This has been used by many "edge-aware" image processing techniques [15] including filtering [51] and denoising [9]. Not surprisingly, they have been also considered essential in texture synthesis: Several methods apply a prior edge detection, such as the Canny edge detector, to improve or guide synthesis [48,53] or to synthesize the edge strokes directly [2,26,34]. Portilla et al. use statistics on wavelets coefficients to generate textures [39], and Wu et al. [49] focus on curvilinear features in art patterns. Lukáč et al. [32] use edges to take into account feature orientation during synthesis and later uses features to paint textures in [33]. For all of these methods, the key is to jointly measure the similarity of color and shape features, between the input and a synthesized texture. Some types of natural patterns, for which edges are predominant, like cracks, have been addressed with specific approaches. Mould [37] applies

Fig. 1 A (small) input texture is analyzed, and edges are extracted and encoded into an edge-based procedural texture (EBPT). New textures are generated either automatically or by controlling the EBPT generation by the user

image filters and generates an image of a fractured surface from a line drawing. Glondu et al. [18] combine physical simulation with image statistics to guide the synthesis of fractures. Our approach differs from edge-aware approaches in that it first synthesizes the salient features using a procedural technique and then maps corresponding patterns. It is thus fast, intuitively controllable and applies to a large variety of structured textures.

Inverse procedural texture modeling has been used to determine parameters using measures applied to input exemplars, so as to recover a model using a generative method, that ensures a certain resemblance between the model and the input. While popular for 3D scenes [12] and 2D layouts [44], very little has been done for stochastic objects [45]. Inverse procedural modeling [1] has not yet received much attention in the field of texture synthesis. Mainly highly symmetric structures like building facades [47] and stochastic "noise"-like patterns have been tackled [14,16], thus limiting the scope of applications. Gilet et al. [17] improve noise by example by adding some types of structures for procedural synthesis. However, the band-pass filtering applied to extract the structural components blurs edges which we want to preserve. One reason for the lack of research in inverse procedural texture synthesis may be related to the extreme difficulty of this problem. There are not many procedural texture "shaders" in databases (even though this number is increasing constantly) compared to the huge variety of existing natural and man-made patterns. For a given natural image, it is unlikely that a shader program exists that could exactly match the image. To circumvent this issue, Ref. [21] models only the structural part of the texture using a noise—similar point process function. The resulting procedural pattern is then augmented with color details using example-based synthesis. The problem is the still limited scope of this function as well as difficult analysis (segmentation to isolate structure and noise parameter extraction). Our approach adds also color details to procedural representations, but avoids the difficulty raised by the complexity of noise functions. We consider *edges*, information which is easy to extract and analyze, instead of considering complicated structural patterns.

Machine learning (ML) has been used for generating synthetic images and parametric models for image synthesis. However, generating feasible images with a good control is still challenging. A sampling approach [29] and a probability-based diffusion model [43] were proposed to extract visual characteristics from training sets. However, the output often suffers from blurry artifacts. Generative adversarial network (GAN) [19] was introduced as a learning and inference method, but generated images are often noisy and may have oscillatory artifacts due to a small training data. Deep convolutional GAN (DCGAN) [40] expands GAN by using vector arithmetic for visual concepts that enables arbitrary feature transfer from input to output, but is limited to faces and human-made objects. One of the commonly used approaches is the `pix2pix` [27] that has been applied in a wide variety of applications including text to image [52], style transfer [25], terrain synthesis [22], multi-domain image synthesis [7,56], guided synthesis [41,50], and for generation of non-stationary textures [20,55]. While ML methods provide user control and generate realistic output, they often require an expensive learning process with large amount of (labeled) data, they need to be supervised, or they suffer from noise.

Our procedural representation of edge groups extends formulations that have been limited to element arrangements based on point groups [35]. We are thus able to produce more complex structured color patterns, such as irregular-shaped stone blocks, that were not possible with previous formulations.

3 Method overview

Figure 2 shows an overview of our method. The input is a 2D texture image, and the output is its procedural representation as EBPT with a set of parameters $\overrightarrow{p}$. A new texture can be synthesized either fully automatically or by manually placing the edge groups. The overall appearance is controlled by varying $\overrightarrow{p}$.

The underlying idea of our approach is to find visually salient features, i.e., edges, and encode their mutual positions and orientations. Edges are essential for the perceived shape of the structures in the image, and humans perceive them as the significant visual features [36] as opposed to low gradient changes [4].

Our algorithm (see an overview in Fig. 2) first *analyzes* the input image (Sect. 4). The edges are extracted and grouped based on their position, orientation, and length. Then, we outline important regions in the input using super pixels [31]. The regions will work as the blueprint of edge groups. The user provides the desired number of groups, and the algorithm calculates clusters of edges based on each edge's length and orientation. Each group's center is used to calculate local statistics that characterize the group. The information includes the distribution, orientation, and shape (style) of the edges in the group. The result of the texture analysis is a set of edge groups that share similar properties. The edge groups, together with the distribution of the edges inside each group and the input image texture, form the EBPT of the input image that can reconstruct it and generate new variations.

During output texture *synthesis* (Sect. 5), a new texture is generated similar to the input. Our algorithm supports two modes: First, a new texture can be generated fully automatically by sampling the parameters of the EBPT. Second, the user can also generate the output interactively either by selecting the edge groups and placing them manually in the output or by varying the parameters $\overrightarrow{p}$ of the EBPT. Once

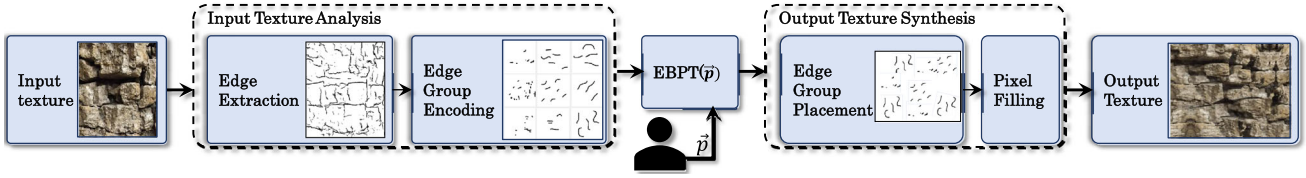

Fig. 2 System overview. The input texture analysis results in the edge-based procedural model for the give input texture image. The input texture is analyzed by extracting and organizing edges into edge groups. Edge groups form the EBPT and are basic building blocks that can be parameterized for either interactive or automatic texture generation. The texture synthesis generates new exemplars by first placing the edges groups and then filling the missing pixels from the input image

the EBPT is executed, it generates edges with similar distribution, orientation, and style based on the input statistics. Multiple edge groups can be placed in, and they can be arbitrarily blended.

After the edges are placed in the output image, our method fills the pixels between edges by importing and blending corresponding pixels from the input image using seamless cloning [38] (Sect. 5.2).

4 Texture analysis

The input is a 2D image representing a texture. The output is EBPT: a set of edge groups with information about edge distributions based on local statistics and additional information about edge group distributions. We develop a novel encoding method for the edges and edge groups.

4.1 Edge extraction and encoding

We first find edges by using the structured forest edge extraction algorithm [10,11]. Then, we encode the edges as chain code [13] that simplifies their further processing. We use the Moore neighborhood (8 pixels) for chain code symbols.

To facilitate edge analysis and comparison, we encode their spatial distributions by generating a feature vector (Fig. 3) denoted by $\mathcal{F}_e$ for each edge e from the input image

$$\mathcal{F}_e = [l_e, c_e, \alpha_e, w_e], \tag{1}$$

where l_e denotes the length of each edge in pixels, c_e is the center of the edge bounding box, α_e is its angle, and w_e is the average edge width. The angle α_e of the edge is the angle between the first eigenvector of the PCA and the x-axis of the coordinate system. The center point c_e is found by calculating edge moments using chain codes [23] (i.e., chain code screen space coordinates).

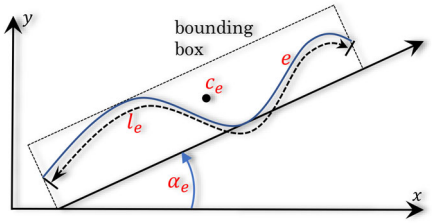

Fig. 3 Detected edge and the features used for further classification: angle α, length l, and center of its bounding box c. We also store the average edge width w

4.2 Edge groups and EBPT

We utilize superpixels [31] as the blueprint to build edge groups. For each region, we find all edges whose c_e belongs to the area defined by a superpixel. We then filter edges based on two spatial properties with strong visual saliency: the edge length l_e and its angle α_e. The user provides the number of bins (usually 3×3), and the edges are quantized into them. We experimented with uniform quantization that generates more edges in a bin if there is a prevailing direction or length in the input. We also implemented an adaptive quantization that guarantees a similar number of edges in a bin that provides better control. Both options are available in our implementation.

An example in Fig. 4 shows the set of extracted edges from an image (left) and its corresponding separation into 3×3 bins. For example, the lower right corner bin includes all edges between $2/3$ of the maximum edge length to the maximum (normalized) length with the angle between zero and $\pi/3$ degrees. The input image can be reconstructed by overlapping the edges from all the bins.

The reasoning behind the filtering is that longer edges are more critical in the input image than the shorter ones. The angle of the edges is also vital for the overall look-and-feel of the output. Using the bins, the users can exclude visually unimportant edges (e.g., short edges) from the synthesis or use edges with prevailing direction.

In the next step, the *spatial distribution* of each (filtered) edge group is characterized. We first calculate the center of

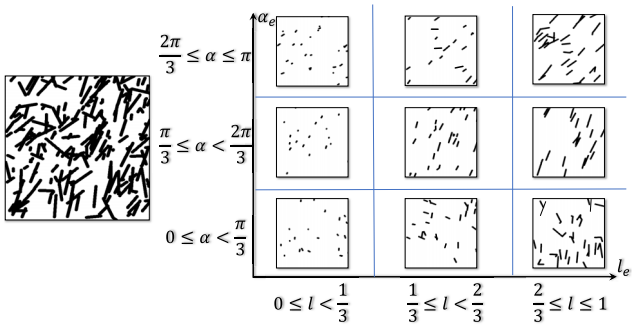

Fig. 4 The extracted edges (left) are divided into 3×3 bins that quantize them according to their length and angle

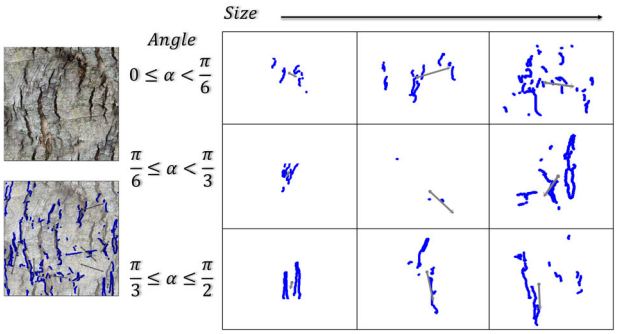

Fig. 6 From the input (left top), we extract edges and group them (left bottom). Then, all edge groups are encoded based on their angles and sizes. The user can control the number of bins

gravity of eg as the average of the edge centers, we denote it c_{eg}, and we call it the *edge group center*:

$$c_{eg} = \frac{1}{|eg|} \sum_{\forall e \in eg} c_e. \tag{2}$$

Then, we use a voting scheme to assign the edges to the spatial grid in polar coordinates centered around the c_{eg} (Fig. 5). The edge group is overlaid by a discrete grid of resolution $n \times m$ (12×30 in our implementation) in polar coordinates centered on c_{eg}. The relative number of edges that have their centers c_{e_i} in them is calculated for each bin.

The edge group is then represented by the non-empty cells and the edges that belong to them, the edge group local statistics, and ranking. An example in Fig. 5 shows a grid of 3×4.

In the next step, edge groups are encoded similarly to how the edges were encoded (Fig. 6). We store the group center, the distances from the center to each edge, and the group angle. Group angle is calculated in the same way with individual edges except that we use each edge's center as the PCA input.

As the edge groups are usually separated in the image, we do not encode edge groups into bins. In this way, the procedural representation is a two-level hierarchy. We encode edges into groups and edge groups into one meta-group.

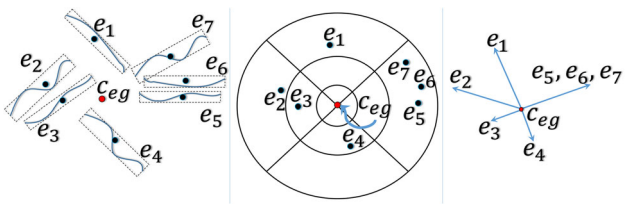

Fig. 5 Edge group representation. Each edge group has its center c_{eg} and a set of edges $(e_1, e_2, \ldots, e_7)$ (left). The edge centers are discretized into a grid in polar coordinates centered around c_{eg} (middle). The edge group is represented by the non-empty cells (right), and each cell stores its distance from c_{eg}, the angle, and the list of corresponding edges

The **EBPT** is then the two-level edge group hierarchy with the encoded edges and their properties. We utilize **EBPT** to characterize the input, represent procedural symbols, and synthesize the output.

5 Texture synthesis

The texture synthesis uses the EBPT (i.e., the edge groups, pixel from the input image, and EBPT parameters) to generate a new image from the input (Fig. 2). The placement of *edge groups* defines the overall look of the texture. It can be done either interactively or fully automatically by replicating the edges' distribution from the input. Pixel filling completes the output by copying corresponding pixels from the input for the edge groups.

5.1 Edge group placement

The input image can be reconstructed by placing the edge groups at the corresponding centers' location and applying the pixel filling step. The edge placement supports can be either interactive or automatic.

In the **interactive** synthesis mode, the user selects edge groups and manually places them on the "canvas" that will be the base of the final output (Fig. 7).

The texture can also be generated **automatically**. Instead of placing the edge groups manually, the edge groups are placed automatically, for example, by replicating the distribution of the edges from the input or by using only edges with a specific range of angles as shown in Fig. 15. Another example in Fig. 8 shows an example of procedural synthesis. Based on the parameters (Table 1), the output that exhibits varying details from the top one-third to the bottom two-thirds is automatically generated.

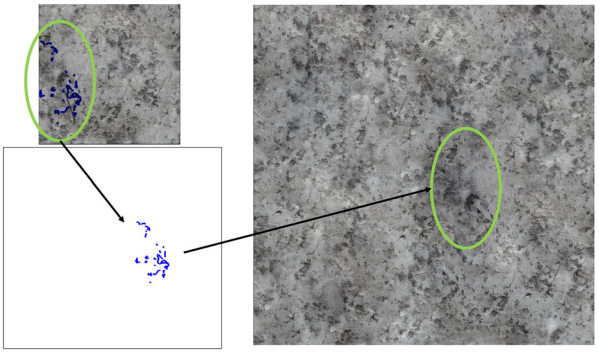

Fig. 7 The user selects an edge group (top left) and places it on the canvas (bottom left). The selected group appears in the output (right)

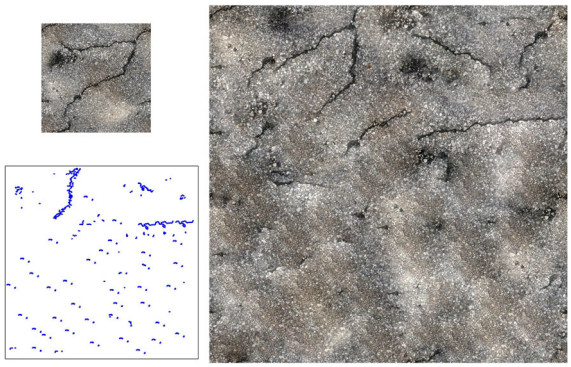

Fig. 8 An automatic synthesis using a simple procedural model. From the input (top left), extracted edge groups are automatically distributed (bottom left) based on the procedural representation. The bottom one-third of the output (right) does not have the crack patterns, whereas the rest shows mixed structures

Table 1 A simple procedural representation of edge groups

Region	Structure	Parameters	Rotation
Top one-third	Random	Random	[0.0, 40.0]
The rest	Granular	Size $\leq$ 32 px	0.0

5.2 Pixel filling

Once the edge groups have been positioned, the missing pixels in the output image are filled. We use seamless image cloning [38] based on the distributed edge groups (Fig. 9). The process consists of three steps. First, we find the region of interest (ROI) using the edge group's convex hull. The ROI serves as the input mask to extract a patch. Using the patch, we apply affine transformations (e.g., rotations) if needed. Finally, we execute the cloning process to generate the output. Although current methods such as image melding [8] can produce more detailed results, we decided to use seamless image cloning that enables an interactive synthesis.

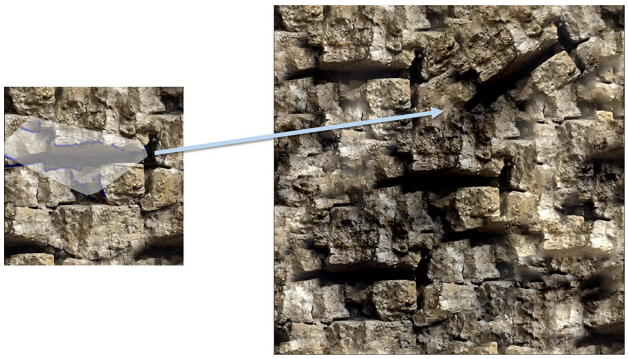

Fig. 9 The patch is defined based on an edge group (left). Transformations are applied before the seamless image cloning process (right)

Table 2 Timing of texture analysis and edge encoding steps

Image resolution	Edge extraction (s)
256 × 256	0.285
512 × 512	0.689
1024 × 1024	2.382
2048 × 2048	7.955
Number of edges	**Edge encoding (s)**
641	0.280
986	0.299
1366	0.350

6 Implementation and results

Our algorithm was implemented in `C++` with `OpenCV`. We tested our results on a workstation with Intel Xeon E5-1630 clocked at 3.7 GHz, with 12 GB of DDR4 RAM, and NVIDIA GeForce GTX TITAN X.

6.1 Performance

The time required by our application depends on the image resolution and the number of processed edges. Table 2 shows timing details for individual steps of texture analysis, and Table 3 shows timings for the output synthesis.

During the *texture analysis* (Sect. 4), the edges are extracted using the structured forest edge algorithm [10,11]. The edge encoding into chain codes takes up to three seconds for largest images (2048 × 2048), and it has complexity $\mathcal{O}(n \times m)$, where n is the number of edges and m is the number of processed pixels.

The *texture synthesis* (Sect. 5) has the same algorithmic complexity $\mathcal{O}(n \times m)$ as the texture analysis. The structure building is done by distributing the edge groups in linear time, and the pixel filling depends only on the number of participating edge groups (i.e., total number of edges).

Table 3 Timing of the texture synthesis step for texture sizes 1024^2 and 2048^2

Number of edges	1024×1024 (ms)	2048×2048 (ms)
120	167	210
246	288	357
360	329	454

Table 4 Timing comparison between image melding and our method using texture sizes 256^2, 512^2, and 1024^2. 157 edges participated in the synthesis process

Image size	Image melding (s)	Our method (s)
256×256	229.481	0.081
512×512	312.267	0.115
1024×1024	668.740	0.186

We compared the performance of our implementation with image melding [8], and the results are shown in Table 4. Since our method does not require any exponential algorithm or heavy repetitions, it is suitable for an interactive application.

6.2 Results

Below we show several results generated by our implementation of the proposed algorithm. The EBPT can be either called directly with the set of parameters, or interactively, where the user manipulates edges groups, rotates them, and changes the parameters, and the output texture is generated. The user modifies intuitively the location of the edge groups to control the appearance of the output.

6.2.1 User-assisted procedural synthesis

Figure 10 shows a texture of concrete with moss and scratches that has been generated from a small example by successively adding and rotating edges groups (shown in blue). When only one edge group is added, the overall shape is similar to the input, but it includes repetitions. The shape gets more random, with more details as more edge groups are added.

The user can control the new texture by placing edge groups and defining their parameters. The results in Figs. 8 and 9 show the effect of the edge group placement. A small input texture is analyzed, and the edge group is repeatedly placed and rotated. The edge group placement controls the overall shape.

Figures 11 and 12 show examples of large texture synthesis from small image examples. The left column shows the input images of tree bark and marble with the edge groups and the generated output edge groups. The edge groups were located manually, and the right column shows the output image. Each example took just a few seconds to generate.

We can also combine two EBPT from **different inputs** as shown in Fig. 13, where two images (a) and (b) with enhanced edge groups are used. The edge groups from each EBPT were placed at different locations in the output (c) resulting in the composite image (d).

Another example of using multiple input images is in Fig. 14 showing a transition between significantly different structures. The angular structures that appeared in (c) were generated by rotating the selected edge group in (a).

A **fully automatic** procedurally generated texture is shown in Fig. 15. A red marble texture was analyzed, and edge groups were extracted. Then, the EBPT generated a random texture with the same distribution as the input image (b), then the edge groups were rotated by the same angle by modifying one parameter of the EBPT, and the generation

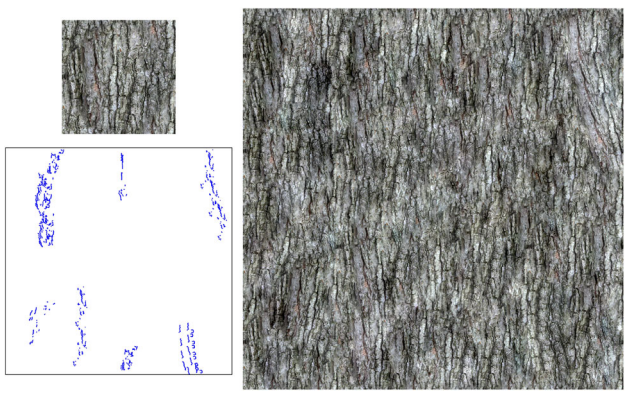

Fig. 11 A small input texture of bark (left up) was used to generate larger texture (right) by placing and randomizing edge groups (left bottom)

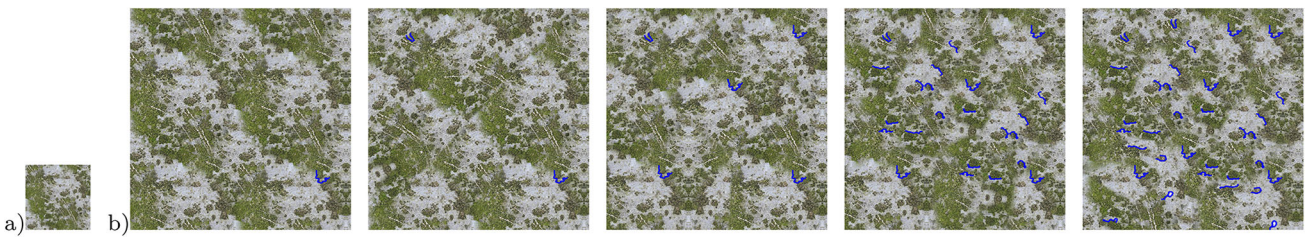

Fig. 10 A small input texture (**a**) is used to generate new ones by successive adding and rotating edge groups (blue) (**b**)

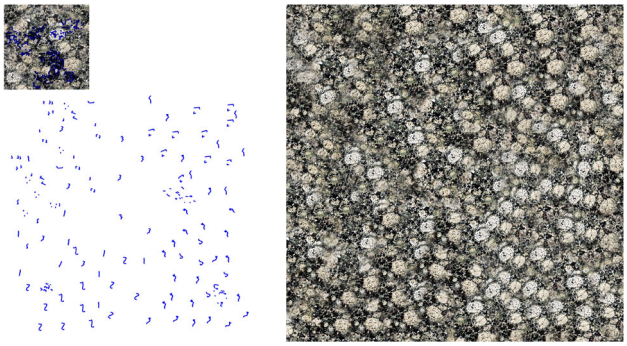

Fig. 12 A marble example (left up) generates larger texture (right) by placing and randomizing edge groups (left bottom)

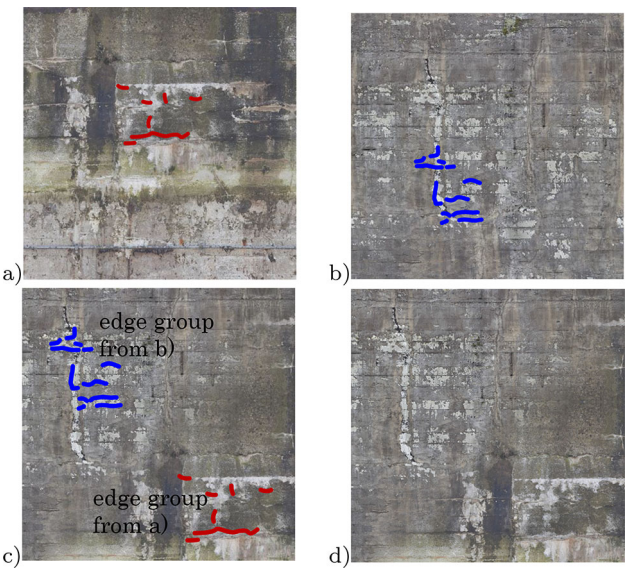

Fig. 13 Two input images (**a**, **b**) are analyzed, and EBPT was extracted. By combining two selected edge groups from the input images (**c**), a new texture is synthesized (**d**)

from the previous step was repeated (c). In the last example (d), we randomized EBPT parameters based on the regions.

6.3 Evaluation

We compare EBPT to several state-of-the-art algorithms. Figure 19 shows the input image and a visual comparison with self-tuning texture optimization [28]. Self-tuning texture optimization produces a visually similar output automatically (the example took 121 min) but provides no user control. The user cannot design a new variant of the texture. The EBPT allows for a simple generation of a variety of textures by controlling the orientation (Fig. 19, top) and the distributions (bottom) of edges. Moreover, the synthesis is three orders of magnitude faster and allows for interactive editing. The blending may, however, introduce artifacts,

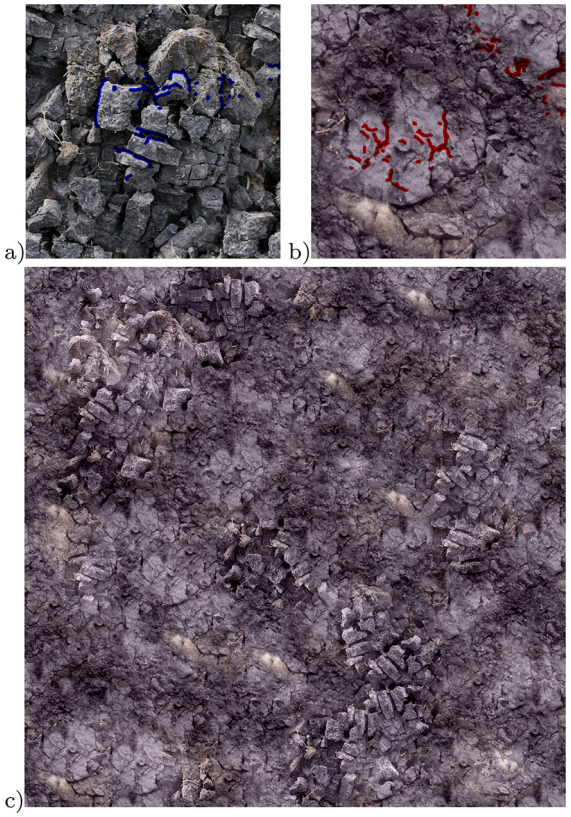

Fig. 14 Combining two EBPTs allows generation of textures from different input. Here, the edge groups from two textures were combined into a new image. Our interactive method grants users direct controls on features of interest. The result (**c**) consists of background (**b**) and user-placed features (**a**)

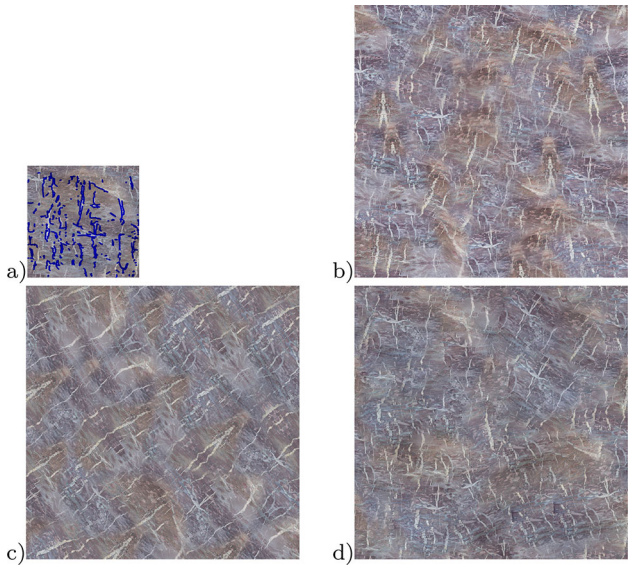

Fig. 15 A large texture can be generated by fully automatic execution of the EBPT. The sample texture using (**a**), randomized output with a similar distribution of the edge groups (**b**), edge groups rotated by a fixed angle (**c**), and region-specific angle constraints (**d**)

Fig. 16 Failure case: Our method tends to blur the image for a small number of edge groups

which is a limitation of our approach (see Sect. 6.4 for more limitations) (Fig. 16).

We have also compared our approach to the procedural generation from Gilet et al. [17]. The result in the leftmost column of Fig. 17 shows a noise synthesized by using only the power spectrum as initially suggested in the work [14]. It is well known that using the power spectrum only (Gaussian texture) fails to capture the input image's structure. The second column was generated by applying an energy threshold of 25% for the fixed-phases part, which tends to over blur edges. The last column shows textures generated with our approach.

Figure 18 represents comparisons between our method and the semi-procedural approach by Guehl et al. [20]. Out method allows an interactive synthesis using arbitrary stationary features and achieved 60% faster synthesis speed using only CPU.

6.4 Limitations and failure cases

A limitation of our method is that the pixel filling algorithm generates undesired artifacts if a small number of edge groups are used. Figures 16 and 17 (bottom right) show such cases. Moreover, the result may show symmetries and repetitions (Fig. 10a) because the method focuses on semi-structured appearances, which are difficult to clearly define on the border of visual features (i.e., chunks of pixels). Thus, the underlying idea of the pixel filling process is a smooth transition between visually salient features. Our method does not aim at pixel-level patch matching and stitching.

Another failure case is for very small example textures with strong structural properties where our method produces highly repetitive results. Highly structured textures amplify the artifacts between visually salient features (Fig. 19, top right). In addition, our method may fail to capture visually salient features in textures without clear edges (Fig. 19, bottom right). However, it can be expected because of the type of texture we focus on (i.e., unclear structures or patterns which are not entirely stochastic).

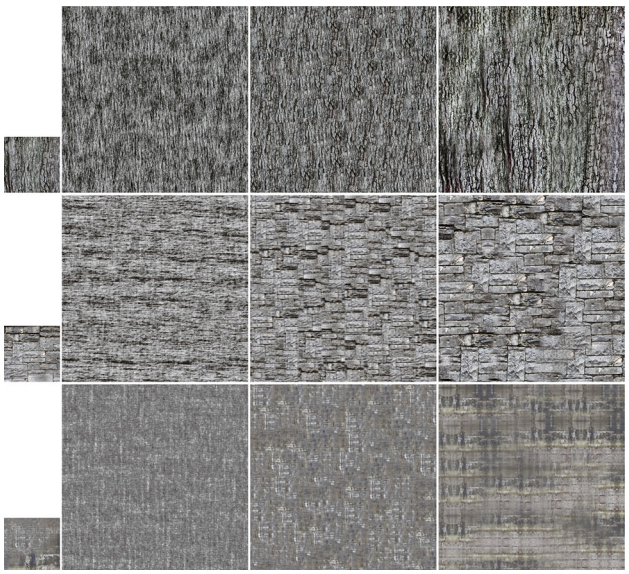

Fig. 17 The first column shows the sample texture. The two middle columns are generated by the algorithm from [17], and the last column shows results produced by our approach

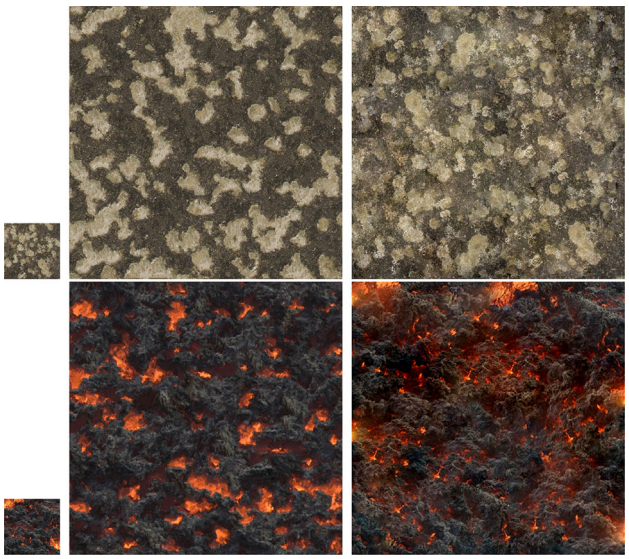

Fig. 18 The left column shows the sample texture. Images in the middle column are generated by the algorithm from [20], and the right column shows our results

7 Conclusions

We have presented a novel approach that generates large textures from a single image. Edge groups are the basic building blocks that are extracted from the input. Their structural properties are encoded as local statistics and, together with the control parameters, wrapped into the novel edge-based texture synthesis method. The EBPT can be used either interactively or automatically to generate larger outputs from a

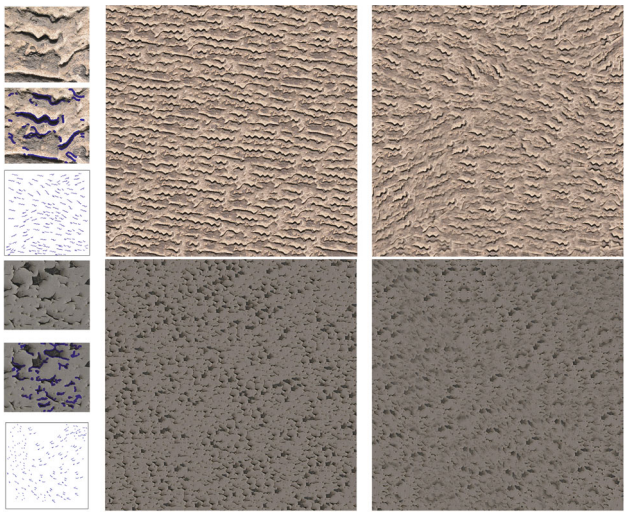

Fig. 19 Comparison of our results (right most column) with [28] (middle column). All images are generated from the examples on the left; the edge groups are also shown

small-sized exemplar. Moreover, multiple EBPT can be used to generate structures from different inputs.

There are many options for possible **future work**. The edge coverage of the input provides a partitioning of the output, and it would be interesting to see whether other methods could complete the pixels, such as graph cuts [5], image melding [8], or PatchNet [24].

Our paper considers only edges with no branching. It would be interesting to extend our method for branching structures such as the work [42]. Another avenue for future work is exploiting other basic building elements such as saliency maps or other visually important features.

Moreover, our method uses only one level of the hierarchy of the procedural model. We could fully integrate it with HSLIC [31] where the edge groups would be combined hierarchically to handle visually salient features more effectively. For example, if a long edge crosses several superpixel regions, our method requires user input to effectively subdivide and define important features (e.g., texels). Multi-level edge groups using HSLIC could address the issue.

Finally, while our method supports user input, its implementation is experimental without a robust GUI that would allow us to run a comprehensive user study left as future work.

Acknowledgements This research was funded in part by National Science Foundation Grant No. 10001387, *Functional Proceduralization of 3D Geometric Models*, and National Science Foundation Grant No. 1608762, *Inverse Procedural Material Modeling for Battery Design*. We thank Dr. Darrell Schulze for his unconditional support and help through this project.

Funding This research was funded by National Science Foundation Grant No. 10001387, Functional Proceduralization of 3D Geometric Models, and National Science Foundation Grant No. 1608762, *Inverse Procedural Material Modeling for Battery Design*.

Declarations

Conflict of interest All authors declare that they have no conflict of interest.

References

1. Aliaga, D.G., Demir, I., Benes, B., Wand, M.: Inverse procedural modeling of 3d models for virtual worlds. In: ACM SIGGRAPH 2016 Courses, SIGGRAPH '16, pp. 16:1–16:316. ACM, New York, NY, USA (2016). https://doi.org/10.1145/2897826.2927323
2. Barla, P., Breslav, S., Thollot, J., Sillion, F., Markosian, L.: Stroke pattern analysis and synthesis. In: Computer Graphics Forum, vol. 25, pp. 663–671. Wiley Online Library (2006)
3. Barnes, C., Zhang, F.L.: A survey of the state-of-the-art in patch-based synthesis. Comput. Vis. Media **3**(1), 3–20 (2017). https://doi.org/10.1007/s41095-016-0064-2
4. Biederman, I., Ju, G.: Surface versus edge-based determinants of visual recognition. Cogn. Psychol. **20**(1), 38–64 (1988)
5. Boykov, Y., Funka-Lea, G.: Graph cuts and efficient nd image segmentation. Int. J. Comput. Vis. **70**(2), 109–131 (2006)
6. Chiu, S.N., Stoyan, D., Kendall, W.S., Mecke, J.: Stochastic Geometry and Its Applications. Wiley, London (2013)
7. Choi, Y., Choi, M., Kim, M., Ha, J.W., Kim, S., Choo, J.: StarGAN: Unified Generative Adversarial Networks for Multi-domain Image-to-Image Translation. arXiv e-prints arXiv:1711.09020 (2017)
8. Darabi, S., Shechtman, E., Barnes, C., Goldman, D.B., Sen, P.: Image melding: Combining inconsistent images using patch-based synthesis. ACM Trans. Graph. **31**(4), 82:1–82:10 (2012). https://doi.org/10.1145/2185520.2185578
9. Deng, G.: Guided wavelet shrinkage for edge-aware smoothing. IEEE Trans. Image Process. **26**(2), 900–914 (2017). https://doi.org/10.1109/TIP.2016.2633941
10. Dollár, P., Zitnick, C.L.: Structured forests for fast edge detection. In: 2013 IEEE International Conference on Computer Vision (ICCV), pp. 1841–1848 (2013). https://doi.org/10.1109/ICCV.2013.231
11. Dollár, P., Zitnick, C.L.: Fast edge detection using structured forests. IEEE Trans. Pattern Anal. Mach. Intell. **37**(8), 1558–1570 (2015). https://doi.org/10.1109/TPAMI.2014.2377715
12. Emilien, A., Vimont, U., Cani, M.P., Poulin, P., Benes, B.: Worldbrush: Interactive example-based synthesis of procedural virtual worlds. ACM Trans. Graph. **34**(4), 106:1–106:11 (2015). https://doi.org/10.1145/2766975
13. Freeman, H.: On the encoding of arbitrary geometric configurations. IRE Trans. Electron. Comput. **EC-10**(2), 260–268 (1961). https://doi.org/10.1109/TEC.1961.5219197
14. Galerne, B., Lagae, A., Lefebvre, S., Drettakis, G.: Gabor noise by example. ACM Trans. Graph. **31**(4), 73:1–73:9 (2012). https://doi.org/10.1145/2185520.2185569
15. Gastal, E.S.L., Oliveira, M.M.: Domain transform for edge-aware image and video processing. In: ACM SIGGRAPH 2011 Papers, SIGGRAPH '11, pp. 69:1–69:12. ACM, New York, NY, USA (2011). https://doi.org/10.1145/1964921.1964964
16. Gilet, G., Dischler, J.M.: An image-based approach for stochastic volumetric and procedural details. In: Proceedings of the 21st Eurographics Conference on Rendering, EGSR'10, pp. 1411–1419. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland (2010). https://doi.org/10.1111/j.1467-8659.2010.01738.x

17. Gilet, G., Sauvage, B., Vanhoey, K., Dischler, J.M., Ghazanfarpour, D.: Local random-phase noise for procedural texturing. ACM Trans. Graph. **33**(6), 195:1–195:11 (2014). https://doi.org/10.1145/2661229.2661249
18. Glondu, L., Muguercia, L., Marchal, M., Bosch, C., Rushmeier, H., Dumont, G., Drettakis, G.: Example-based fractured appearance. Comput. Graph Forum **31**(4), 1547–1556 (2012). https://doi.org/10.1111/j.1467-8659.2012.03151.x
19. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial nets. In: Z. Ghahramani, M. Welling, C. Cortes, N.D. Lawrence, K.Q. Weinberger (eds.) Advances in Neural Information Processing Systems 27, pp. 2672–2680. Curran Associates, Inc. (2014). http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf
20. Guehl, P., Allegre, R., Dischler, J.M., Benes, B., Galin, E.: Semi-procedural textures using point process texture basis functions. Comput. Graph. Forum **39**(4), 159–171 (2020). https://doi.org/10.1111/cgf.14061 (Honorable mention from the Best Papers Committee)
21. Guehl, P., Allegre, R., Dischler, J.M., Benes, B., Galin, E.: Semi-procedural textures using point process texture basis functions. Comput. Graph. Forum **39**(4), 159–171 (2020). https://doi.org/10.1111/cgf.14061
22. Guérin, E., Digne, J., Galin, E., Peytavie, A., Wolf, C., Benes, B., Martinez, B.: Interactive example-based terrain authoring with conditional generative adversarial networks. ACM Trans. Graph. **36**(6), 228:1–228:13 (2017). https://doi.org/10.1145/3130800.3130804
23. Hu, M.K.: Visual pattern recognition by moment invariants. IRE Trans. Inform. Theory **8**(2), 179–187 (1962)
24. Hu, S.M., Zhang, F.L., Wang, M., Martin, R.R., Wang, J.: Patchnet: A patch-based image representation for interactive library-driven image editing. ACM Trans. Graph. **32**(6), 196:1–196:12 (2013). https://doi.org/10.1145/2508363.2508381
25. Huang, X., Belongie, S.: Arbitrary style transfer in real-time with adaptive instance normalization. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 1501–1510 (2017)
26. Hurtut, T., Landes, P.E., Thollot, J., Gousseau, Y., Drouillhet, R., Coeurjolly, J.F.: Appearance-guided synthesis of element arrangements by example. In: Proceedings of the 7th International Symposium on Non-photorealistic Animation and Rendering, NPAR '09, pp. 51–60. ACM, New York, NY, USA (2009). https://doi.org/10.1145/1572614.1572623
27. Isola, P., Zhu, J.Y., Zhou, T., Efros, A.A.: Image-to-image translation with conditional adversarial networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1125–1134 (2017)
28. Kaspar, A., Neubert, B., Lischinski, D., Pauly, M., Kopf, J.: Self tuning texture optimization. Comput. Graph. Forum **34**(2), 349–359 (2015). https://doi.org/10.1111/cgf.12565
29. Kingma, D.P., Welling, M.: Auto-Encoding Variational Bayes. arXiv e-prints arXiv:1312.6114 (2013)
30. Kwatra, V., Essa, I., Bobick, A., Kwatra, N.: Texture optimization for example-based synthesis. In: ACM SIGGRAPH 2005 Papers, SIGGRAPH '05, pp. 795–802. ACM, New York, NY, USA (2005). https://doi.org/10.1145/1186822.1073263
31. Lockerman, Y.D., Sauvage, B., Allègre, R., Dischler, J.M., Dorsey, J., Rushmeier, H.: Multi-scale label-map extraction for texture synthesis. ACM Trans. Graph. **35**(4), 140:1–140:12 (2016). https://doi.org/10.1145/2897824.2925964
32. Lukáč, M., Fišer, J., Asente, P., Lu, J., Shechtman, E., Sýkora, D.: Brushables: Example-based edge-aware directional texture painting. Comput. Graph. Forum **34**(7), 257–267 (2015). https://doi.org/10.1111/cgf.12764
33. Lukáč, M., Fišer, J., Bazin, J.C., Jamriška, O., Sorkine-Hornung, A., Sýkora, D.: Painting by feature: Texture boundaries for example-based image creation. ACM Transactions on Graphics (Proceedings of SIGGRAPH 2013) **32**(4), 116 (2013)
34. Ma, C., Wei, L.Y., Tong, X.: Discrete element textures. ACM Trans. Graph. **30**(4), 62:1–62:10 (2011). https://doi.org/10.1145/2010324.1964957
35. Ma, C., Wei, L.Y., Tong, X.: Discrete element textures. In: ACM SIGGRAPH 2011 Papers, SIGGRAPH '11, pp. 62:1–62:10. ACM, New York, NY, USA (2011).https://doi.org/10.1145/1964921.1964957
36. Marr, D., Hildreth, E.: Theory of edge detection. Proc. R. Soc. Lond. B Biol. Sci. **207**(1167), 187–217 (1980)
37. Mould, D.: Image-guided fracture. In: Proceedings of Graphics Interface 2005, GI '05, pp. 219–226. Canadian Human-Computer Communications Society, School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada (2005). http://dl.acm.org/citation.cfm?id=1089508.1089545
38. Pérez, P., Gangnet, M., Blake, A.: Poisson image editing. ACM Trans. Graph. **22**(3), 313–318 (2003). https://doi.org/10.1145/882262.882269
39. Portilla, J., Simoncelli, E.P.: A parametric texture model based on joint statistics of complex wavelet coefficients. Int. J. Comput. Vis. **40**(1), 49–70 (2000). https://doi.org/10.1023/A:1026553619983
40. Radford, A., Metz, L., Chintala, S.: Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. arXiv e-prints arXiv:1511.06434 (2015)
41. Sangkloy, P., Lu, J., Fang, C., Yu, F., Hays, J.: Scribbler: Controlling deep image synthesis with sketch and color. (2016). arXiv:1612.00835
42. Sibbing, D., Pavić, D., Kobbelt, L.: Image synthesis for branching structures. In: Computer Graphics Forum, vol. 29, pp. 2135–2144. Wiley Online Library (2010)
43. Sohl-Dickstein, J., Weiss, E.A., Maheswaranathan, N., Ganguli, S.: Deep unsupervised learning using nonequilibrium thermodynamics. arXiv e-prints arXiv:1503.03585 (2015)
44. Štěva, O., Benes, B., Měch, R., Aliaga, D.G., Krištof, P.: Inverse procedural modeling by automatic generation of l-systems. Comput. Graph. Forum **29**(2), 665–674 (2010). https://doi.org/10.1111/j.1467-8659.2009.01636.x
45. Štěva, O., Pirk, S., Kratt, J., Chen, B., Měch, R., Deussen, O., Benes, B.: Inverse procedural modelling of trees. Comput. Graph. Forum **33**(6), 118–131 (2014). https://doi.org/10.1111/cgf.12282
46. Wei, L.Y., Lefebvre, S., Kwatra, V., Turk, G.: State of the art in example-based texture synthesis. In: Eurographics 2009, State of the Art Report, EG-STAR, pp. 93–117. Eurographics Association (2009)
47. Wu, F., Yan, D.M., Dong, W., Zhang, X., Wonka, P.: Inverse procedural modeling of facade layouts. ACM Trans. Graph. **33**(4), 121:1–121:10 (2014). https://doi.org/10.1145/2601097.2601162
48. Wu, Q., Yu, Y.: Feature matching and deformation for texture synthesis. In: ACM SIGGRAPH 2004 Papers, SIGGRAPH '04, pp. 364–367. ACM, New York, NY, USA (2004). https://doi.org/10.1145/1186562.1015730
49. Wu, R., Wang, W., Yu, Y.: Optimized synthesis of art patterns and layered textures. IEEE Trans. Visual Comput. Graph. **20**(3), 436–446 (2014). https://doi.org/10.1109/TVCG.2013.113
50. Xian, W., Sangkloy, P., Lu, J., Fang, C., Yu, F., Hays, J.: Texturegan: Controlling deep image synthesis with texture patches. (2017). arXiv:1706.02823
51. Xu, L., Ren, J.S.J., Yan, Q., Liao, R., Jia, J.: Deep edge-aware filters. In: Proceedings of the 32nd International Conference on International Conference on Machine Learning—Volume 37, ICML'15, pp. 1669–1678. JMLR.org (2015). http://dl.acm.org/citation.cfm?id=3045118.3045296
52. Zhang, H., Xu, T., Li, H., Zhang, S., Wang, X., Huang, X., Metaxas, D.N.: Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks. In: Proceedings of

the IEEE International Conference on Computer Vision, pp. 5907–5915 (2017)
53. Zhou, H., Sun, J., Turk, G., Rehg, J.M.: Terrain synthesis from digital elevation models. IEEE Trans. Vis. Comput. Graph. **13**(4), 834–848 (2007)
54. Zhou, Y., Shi, H., Lischinski, D., Gong, M., Kopf, J., Huang, H.: Analysis and controlled synthesis of inhomogeneous textures. Computer Graphics Forum (Proc. of Eurographics 2017) **36**(2) (2017)
55. Zhou, Y., Zhu, Z., Bai, X., Lischinski, D., Cohen-Or, D., Huang, H.: Non-stationary texture synthesis by adversarial expansion. ACM Trans. Graph. **37**(4), 49:1–49:13 (2018). https://doi.org/10.1145/3197517.3201285
56. Zhu, J., Zhang, R., Pathak, D., Darrell, T., Efros, A.A., Wang, O., Shechtman, E.: Toward multimodal image-to-image translation. (2017). arXiv:1711.11586

Hansoo Kim is a software engineer at Google. He is currently a member of Google Photos team and is working on computational photography projects. He received his B.S. and M.S. degrees in multimedia engineering from Konkuk University in Republic of Korea and Ph.D. in computer graphics technology from Purdue University in Indiana, USA. His main research interests include texture synthesis, computational photography, and procedural modeling.

Jean-Michel Dischler holds the position of full Professor in Computer Science at Strasbourg University. He is currently the joint director of the 3D Computer Graphics Group, in the ICUBE laboratory, and leading the rendering and visualization team. His main research interests include texture acquisition synthesis, rendering, high-performance graphics, and simulation of natural phenomena.

Holly Rushmeier is a professor of Computer Science at Yale University. She received the B.S., M.S., and Ph.D. degrees in mechanical engineering from Cornell University in 1977, 1986, and 1988 respectively. Between receiving the Ph.D. and arriving at Yale, she held positions at Georgia Tech, NIST, and IBM Watson research. Her current research interests include acquiring and modeling material appearance, applications of human perception to realistic rendering, and applications of computer graphics in cultural heritage. She is a fellow of the Eurographics Association, an ACM Distinguished Engineer and the recipient of the 2013 ACM SIGGRAPH Computer Graphics Achievement Award.

Bedrich Benes is George McNelly professor of Technology and professor of Computer Science at Purdue University. His area of research is in procedural and inverse procedural modeling and simulation of natural phenomena and he has published over 150 research papers in the field.

Terms and Conditions

Springer Nature journal content, brought to you courtesy of Springer Nature Customer Service Center GmbH (“Springer Nature”).
Springer Nature supports a reasonable amount of sharing of research papers by authors, subscribers and authorised users (“Users”), for small-scale personal, non-commercial use provided that all copyright, trade and service marks and other proprietary notices are maintained. By accessing, sharing, receiving or otherwise using the Springer Nature journal content you agree to these terms of use (“Terms”). For these purposes, Springer Nature considers academic use (by researchers and students) to be non-commercial.
These Terms are supplementary and will apply in addition to any applicable website terms and conditions, a relevant site licence or a personal subscription. These Terms will prevail over any conflict or ambiguity with regards to the relevant terms, a site licence or a personal subscription (to the extent of the conflict or ambiguity only). For Creative Commons-licensed articles, the terms of the Creative Commons license used will apply.
We collect and use personal data to provide access to the Springer Nature journal content. We may also use these personal data internally within ResearchGate and Springer Nature and as agreed share it, in an anonymised way, for purposes of tracking, analysis and reporting. We will not otherwise disclose your personal data outside the ResearchGate or the Springer Nature group of companies unless we have your permission as detailed in the Privacy Policy.
While Users may use the Springer Nature journal content for small scale, personal non-commercial use, it is important to note that Users may not:

1. use such content for the purpose of providing other users with access on a regular or large scale basis or as a means to circumvent access control;
2. use such content where to do so would be considered a criminal or statutory offence in any jurisdiction, or gives rise to civil liability, or is otherwise unlawful;
3. falsely or misleadingly imply or suggest endorsement, approval , sponsorship, or association unless explicitly agreed to by Springer Nature in writing;
4. use bots or other automated methods to access the content or redirect messages
5. override any security feature or exclusionary protocol; or
6. share the content in order to create substitute for Springer Nature products or services or a systematic database of Springer Nature journal content.

In line with the restriction against commercial use, Springer Nature does not permit the creation of a product or service that creates revenue, royalties, rent or income from our content or its inclusion as part of a paid for service or for other commercial gain. Springer Nature journal content cannot be used for inter-library loans and librarians may not upload Springer Nature journal content on a large scale into their, or any other, institutional repository.
These terms of use are reviewed regularly and may be amended at any time. Springer Nature is not obligated to publish any information or content on this website and may remove it or features or functionality at our sole discretion, at any time with or without notice. Springer Nature may revoke this licence to you at any time and remove access to any copies of the Springer Nature journal content which have been saved.
To the fullest extent permitted by law, Springer Nature makes no warranties, representations or guarantees to Users, either express or implied with respect to the Springer nature journal content and all parties disclaim and waive any implied warranties or warranties imposed by law, including merchantability or fitness for any particular purpose.
Please note that these rights do not automatically extend to content, data or other material published by Springer Nature that may be licensed from third parties.
If you would like to use or distribute our Springer Nature journal content to a wider audience or on a regular basis or in any other manner not expressly permitted by these Terms, please contact Springer Nature at

onlineservice@springernature.com