Fast Spatially Controllable Multi-Dimensional Exemplar-Based Texture Synthesis and Morphing

Felix Manke and Burkhard Wünsche

University of Auckland, Dept. of Computer Science, Graphics Group, Private Bag 92019, Auckland, New Zealand
fman020@aucklanduni.ac.nz, burkhard@cs.auckland.ac.nz
http://www.cs.auckland.ac.nz/~burkhard

Abstract. Texture synthesis and morphing are important techniques for efficiently creating realistic textures used in scientific and entertainment applications. In this paper we present a novel fast algorithm for multi-dimensional texture synthesis and morphing that is especially suitable for parallel architectures such as GPUs or direct volume rendering (DVR) hardware. Our proposed solution generalizes the synthesis process to support higher than three-dimensional synthesis and morphing.
We introduce several improvements to previous 2D synthesis algorithms, such as new appearance space attributes and an improved jitter function. We then modify the synthesis algorithm to use it for texture morphing which can be applied to arbitrary many 2D input textures and can be spatially controlled using weight maps. Our results suggest that the algorithm produces higher quality textures than alternative algorithms with similar speed. Compared to higher quality texture synthesis algorithms, our solution is considerably faster and allows the synthesis of additional channels, such as transparencies and displacement maps, without affecting the running time of the synthesis at all. The method is easily extended to allow fast 3D synthesis and we show several novel examples and applications for morphed solid 3D textures.
Overall the presented technique provides an excellent trade-off between speed and quality, is highly flexible, allows the use of arbitrary channels, can be extended to arbitrary dimensions, is suitable for a GPU-implementation, and can be effectively integrated into rendering frameworks such as DVR tools.

1 Introduction

Texture mapping is one of the most important techniques for increasing the realism of a 3D scene by providing fine surface details. Exemplar-based 2D texture synthesis is a powerful tool to generate large textures from a single input example. Texture morphing as an extension creates coherent transitions between entirely different materials with a quality and flexibility that cannot be achieved using simple blending techniques. The applications of texture morphing are manifold and include terrain rendering, scientific visualization, the creation of transitions in animal fur and between biomedical or geological materials, and the simulation of aging processes.

3D solid textures have the advantage that, in contrast to 2D textures, objects can be ”carved” out of a 3D material resulting in more realistic results. Since the acquisition of

3D textures is difficult, the synthesis and morphing of solid textures from 2D exemplars is very important. However, the task is extremely challenging and usually requires long computation times.

In this paper, we present a new fast algorithm for exemplar-based 2D texture morphing and higher-dimensional texture synthesis and morphing. After presenting the method for generating morphed 2D textures from 2D exemplars, we extend the algorithm to support 3D solid texture synthesis and morphing from the 2D input. A generalization to higher-dimensional texture morphing is followed by an evaluation of our method and comparison with Kopf et al.'s algorithm [10].

Our texture morphing algorithm is based on Lefebvre and Hoppe's pixel-based texture synthesis algorithm [13], a real-time approach implemented on the GPU by utilizing the parallel synthesis scheme proposed by L. Wei [26]. In a second publication the authors introduce the *appearance space*, a high-dimensional space that carries much more information per pixel than only color [14]. Its information richness allows us to perform a very robust texture morphing between exemplars of different nature.

Because of the close relationship we will give a brief summary of the original synthesis algorithm for only one input exemplar in section 3. We then discuss our extensions for texture morphing of arbitrary many input exemplars in 2D (section 4), 3D (section 5), and higher dimensions (section 6). In section 7 we discuss our results and conclude with an outlook on future research in section 8.

2 Related Work

Texture synthesis and texture morphing are closely related fields in which numerous different algorithms have been proposed. Procedural techniques for both 2D and 3D texture synthesis [19, 24, 28, 29] proofed to be hard to control and, compared to exemplar-based methods, limited in the variety of materials that can be modeled. Parametric exemplar-based methods, as proposed in [7, 3, 20, 2], rely on models of global statistical properties which serve as constraint function while matching statistics of the input and target texture. Though extensions for 3D synthesis have been made [7, 4], parametric models are usually only successful in synthesizing homogeneous and stochastic exemplars. Mixing properties of different textures is possible, but for texture morphing not enough spatial control is offered. Patch-based methods paste random patches of the exemplar into the output texture and optimize the transitions between overlapping patches [21, 5, 12]. While these methods could probably be extended to use 3D texture patches as input, there is no straightforward way to generate 3D textures from 2D input patches. In fact, we believe it is questionable whether such techniques can be used to create texture morphing of acceptable quality at all (though Kwatra et al. placed flower textures onto a grass texture and optimized the grass seams [12]).

In contrast, by processing one pixel at a time pixel-based methods [6, 27, 1, 13] offer a control that is fine enough to allow high-quality texture synthesis and morphing in 2D, 3D, and even higher dimensions. A successful 3D synthesis has been shown by L. Wei [25, 26]. Finally, optimization-based approaches use the local similarity measures of pixel neighborhoods to define a global texture energy function that is minimized [11]. Recently, Kopf et al. demonstrated that energy-based methods can be used for 3D synthesis [10], though the synthesis times of up to 90 minutes are rather slow and a GPU implementation is non-trivial. Another specialized solution for synthesizing 3D composite materials based on stereology theories (the analysis of 2D cross-sections of 3D volumes) has been proposed by Jagnow et al. [9].

Algorithms specifically for 2D texture morphing have also been developed. L. Wei used his pixel-based method to create transitions between two exemplars [26]. However, the synthesis quality within the transition area decreased significantly. Liu et al. proposed a pattern-based approach that uses ideas of image morphing in order to generate metamorphosis sequences [15]. Both Zhang et al. and Tonietto and Walter used texton maps to support a pixel-based texture morphing [30, 23]. Unfortunately, all three publications show results only for very similar input exemplars, which makes it difficult to assess the quality of the approaches. Matusik et al. utilized a simplical complex model to build a neighborhood graph of input exemplars [18]. Though examples with several input textures are given, the approach relies on a texture database and is explicitly designed for similar textures only. To our knowledge, texture morphing in three dimensions using exemplars of very different irregularly textured materials has not been shown by anyone before.

3 Lefebvre and Hoppe's 2D Texture Synthesis

As most pixel-based methods, the algorithm proposed in [13, 14] performs an iterative optimization to minimize the difference of the synthesis result to the original exemplar, where the distance is measured using the sum of squared differences (SSD) of local neighborhoods. A standard multi-resolution approach is pursued by computing a Gaussian pyramid E of the exemplar and creating an "empty" pyramid S for the synthesis result. The synthesis is performed from the coarsest to the finest resolution, first establishing low frequencies and then defining the fine details.

A key difference of the algorithm to other methods is that S does not store image colors, but *pixel coordinates* into the exemplar E, which facilitates a GPU implementation. To pass the synthesis result of a coarse level S_{i-1} to a finer level S_i an *upsampling* of the coordinates is performed that distributes the value of $S_{i-1}(P)$ (that is, a coordinate into E_i) to four child pixels in S_i. In the correction phase, the synthesis error is minimized by searching the pixel coordinate Q with the best-matching local neighborhood $N_{E_i}(Q)$ in the exemplar for the neighborhood $N_{S_i}(P)$ around $S_i(P)$. The correction phase is accelerated using sub-passes, each of which optimizes only selected pixels, and *k-coherence* search [22] based on pre-computed candidate sets.

The texture synthesis can greatly benefit from using an *appearance space* [14], where pixels encode texture characteristics in addition to color. The high-dimensional appearance vectors are projected into a low-dimensional space defined by the first n principal components obtained from a principal component analysis (PCA). In tests we found that usually more than 95% of the total variance of an exemplar is explained by the first 8 components.

4 Our 2D Texture Morphing Algorithm

When dealing with texture morphing we have to synthesize a texture based on several input exemplars. The result should reflect the nature of all exemplars, though the influence of each input can vary spatially. To control the spatial influence of each of the m exemplars, we use scalar *weight maps* of the size of the synthesized texture S. Each weight map W^j encodes the weight of the exemplar E^j per position $P \in S$. To ensure a correct morphing, we normalize the weight maps so that $\sum_{j=1}^{m} W^j(P) = 1$. Note that, when specifying only one input exemplar, our algorithm behaves like a standard texture synthesis algorithm.

In the following, we will present the extensions we made for every single step of the original synthesis algorithm. Because the coordinate upsampling remains unchanged and is performed on each S_i^j individually, we do not include it in the discussion. Note that the modifications necessary for texture morphing still allow an implementation on parallel architectures.

4.1 Initialization

The initialization of the exemplars themselves remains unchanged, because they are independent from each other. For every weight map W^j that is associated with each exemplar, we additionally compute a Gaussian pyramid. Because the algorithm is based on manipulating exemplar *coordinates* rather than colors (and coordinates cannot be averaged or merged), we need a separate synthesis pyramid S^j for every exemplar. Instead of initializing S_{-1}^j with zero coordinates, we find the following initialization more intuitive (where s_{ej} is the size of E^j):

$$S_{-1}(P) = P \bmod s_{ej}.$$

This better reflects how the algorithm proceeds, especially when only a few pyramid levels are used.

4.2 Coordinate Jitter

Similar to the upsampling the coordinate jitter is also independent for each exemplar and could be carried over unchanged. However, the plot of Lefebvre and Hoppe's function $J_i(P)$ for different combinations of $\mathfrak{H}$ and r (see figure 1 (left)) shows that $J_i(P)$ always returns zero for randomness $r < 0.5$. In addition, for $r = 1$ the probability of $J_i(P) = 0$ is twice as high as for $J_i(P) = -1$ and $J_i(P) = 1$. We therefore propose the following equation that behaves more intuitive (see also figure 1 (right)):

$$J_i(P) = \left\lfloor j + \begin{pmatrix} k_x \\ k_y \end{pmatrix} \right\rfloor, \text{ where}$$

$$\begin{aligned} j &= \mathfrak{H}(P) \cdot \mathrm{lerp}(0.5, 1, r_i), \\ k_{x|y} &= \begin{cases} \mathrm{lerp}(0.5, 2/3, r_i) & \text{if } j_{x|y} \geq 0 \\ 1 - \mathrm{lerp}(0.5, 2/3, r_i) & \text{otherwise,} \end{cases} \\ \mathrm{lerp}(a,b,t) &= a + t(b-a). \end{aligned}$$

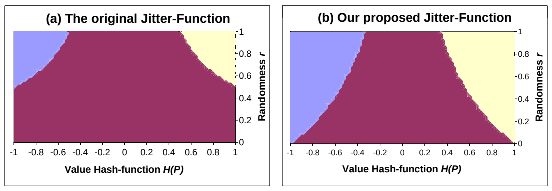

Fig. 1. Plots of the result of different jitter functions $J_i(P)$. Left: The original jitter function proposed in [13]. Right: Our improved jitter function. Color encoding: Blue $= -1$, Magenta $= 0$, Yellow $= 1$.

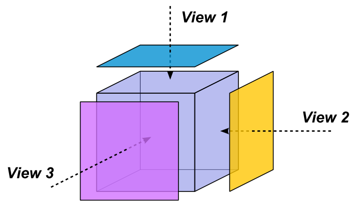

Fig. 2. Several 2D exemplars define different views of the solid texture being synthesized. Adopted from [25].

4.3 Correction Phase

In the correction phase the best-matching pixel for a synthesis pixel P is searched in the exemplar. However, now $S_i(P)$ must represent *all* input exemplars, weighted according to the weights $W_i^j(P)$. For the original 2D synthesis algorithm the neighborhood of P in the synthesis level S_i can be defined as the set:

$$N_{S_i}(P) = \{E_i(Q) \mid Q = S_i(P + \Delta_N)\}, \;\; \Delta_N \in \mathcal{N}, \tag{1}$$

where Δ_N takes the values of offset vectors from the neighborhood's center to all pixels in the neighborhood as defined by the set $\mathcal{N}$ of all offsets:

$$\mathcal{N} = \{\delta \mid (P + \delta) \in N(P)\}.$$

For the texture morphing of multiple exemplars, we extend the neighborhood gathering as follows:

$$\begin{aligned} N_{S_i}(P) &= \{C(i, P + \Delta_N)\}, \;\; \Delta_N \in \mathcal{N}, \\ C(i,X) &= \sum_{j=1}^{m} W_i^j(X) \cdot E_i^j(Q), \;\; Q = S_i^j(X). \end{aligned} \tag{2}$$

$C(i,X)$ is the synthesized color in level i at location X and is‘ the weighted average of all exemplar levels. Given the synthesis neighborhood $N_{S_i}(P)$ the best-matching pixel is searched for in each of the exemplars E_i^j and $S_i^j(P)$ where $N_{S_i}(P)$ is the same for all m synthesis levels $S_i^j(P)$:

$$S_i(P) = \underset{Q \in E_i}{\operatorname{argmin}} \; SSD(N_{S_i}(P), N_{E_i}(Q)). \tag{3}$$

5 3D Texture Synthesis and Morphing

In the following, we will discuss how to generate textures with an additional spatial dimension and how to morph between different materials. As illustrated in figure 2, several input exemplars are considered as being different *views* of the solid texture cube that is to be synthesized [25, 10]. During the synthesis, the algorithm tries to generate a 3D texture that reflects the characteristics of all views. The close relationship to 2D texture morphing in terms of the use of multiple exemplars was already mentioned by L. Wei [25]. To create smooth transitions between entirely different materials within the solid texture cube, exemplar views need to be specified for each material. For 3D morphing the weight maps are used as in the 2D morphing to define the spatial influence of the materials. Our algorithm gives a unified tool for supporting 3D synthesis or morphing at the same time. It is not always necessary or appropriate to specify all three exemplars. Sometimes, for example when a material exhibits dominant directional features, it is better to define only two views [25].

The jitter step is not affected by our modifications, because the coordinates stored in a synthesis level $S_i^j(P)$ are still defined in $\mathbb{R}^2$. A minor difference is that the jitter function $J_i(P)$ and the hash function $\mathfrak{H}(P)$ now take 3D coordinates as input argument.

5.1 Initialization

In contrast to the 2D morphing, the synthesis pyramids S^j and the weight maps W^j are solid texture cubes. In S^j each voxel stores a 2D coordinate into the corresponding exemplar E^j. Because E^j represents only one particular view onto the solid target texture, the initialization of S_{-1}^j is modified in the following way:

$$S_{-1}^j(P) = P_{u,v} \mod s_{e^j},$$

where u and v denote the two components of $P \in \mathbb{R}^3$ to which E^j is parallel.

5.2 Coordinate Upsampling

The coordinate upsampling cannot simply be extended by an additional dimension, because the synthesis pyramids still store 2D coordinates. Thus, we apply the 2D upsampling for every second slice that is oriented parallel to the exemplar view and duplicate the result for each subsequent slice:

$$\begin{aligned} S_i^j(2 \cdot P_{u,v|w} + \Delta_{u,v}) = S_i^j(2 \cdot P_{u,v|(w+1)} + \Delta_{u,v}) = \\ (2 \cdot S_{i-1}(P) + \Delta_U) \mod S_{e^j}, \end{aligned} \tag{4}$$

where w is even ($w \mod 2 = 0$) and depicts the component of P that is orthogonal to the view. $\Delta_{u,v}$ describes a 3D vector with the same value in the u- and v-component as Δ_U and $w = 0$.

5.3 Correction Phase

When performing neighborhood-matching during the correction we have to compare neighborhoods around voxels in a 3D synthesis pyramid with neighborhoods of pixels of a 2D input exemplar. We exploit that the exemplars are oriented orthogonally to one of the principal axes of the solid cube that is synthesized. Since each exemplar represents only the view in this direction, the synthesis pyramid for an exemplar only needs to reflect the exemplar in that direction. Thus, we can align the 2D neighborhoods $N_{S_i}(P)$ to stand parallel to the exemplar, as it has also been proposed in [25, 10].

In consequence, we need to introduce several synthesis neighborhoods $N_{S_i|u,v}(P)$, one for each possible orientation of exemplars. Note that, as in the 2D morphing algorithm, $N_{S_i|u,v}(P)$ is a merged neighborhood that needs to represent all exemplars. Our definition in equation 2 is also valid for the 3D synthesis, except that $P \in S_i^j$ is now defined in $\mathbb{R}^3$. We therefore modify the definition to support oriented neighborhood gathering in 3D solid textures:

$$N_{S_i|u,v}(P) = \{C(i, P + \Delta_{N|u,v})\}, \;\; \Delta_{N|u,v} \in \mathcal{N}_{u,v}, \tag{5}$$

where $\Delta_{N|u,v}$ gives the neighborhood offsets parallel to the current view (the w-component of $\Delta_{N|u,v}$ is set to 0). Notice the similarity to the upsampling step, which also depends on the view's orientation.

Interleaved correction for an improved convergence using sub-passes is still possible in 3D synthesis and morphing. However, because of the additional dimension we now have to define eight sub-passes as a pattern in a 2^3 cube.

5.4 A New Neighborhood

As discussed in section 3, the synthesis in the reduced appearance space makes it possible to use a very compact neighborhood consisting of only 4 corner pixels. However, 3D texture synthesis is a much more challenging problem. The algorithm generally has to deal with little information that is available for generating a solid texture from the 2D exemplars. We found that the reduced neighborhood is not capable of preserving the features of the input exemplars. Much better results can be achieved using a full 5×5 or even 7×7 neighborhood — of course at the expense of speed.

In order to improve the synthesis while keeping the computation time low we propose a new "half-reduced" neighborhood. The layout is shown in figure 3 (top). We still average several pixel values to compute the values for the individual neighborhood values (shown on the bottom of the figure). Note that the new neighborhood, consisting of 9 points, is a superset of the neighborhood proposed in [14]. Figure 4 shows a comparison of the results using different neighborhoods for 3D synthesis. Our new neighborhood is much better capable of preserving feature coherence than the original four-pixel neighborhood, which fails to produce acceptable results. The new neighborhood can also be used for 2D texture synthesis and morphing. We found that 2D morphing results are improved significantly for exemplars with large semantic features [16].

5.5 Synthesis and Morphing of Additional Channels

The synthesis and morphing based on texture coordinates makes it possible to restore the exact location in the original exemplar for each pixel in the synthesized texture.

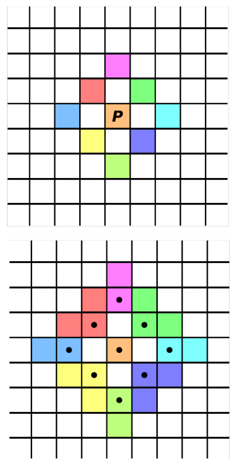

Fig. 3. Pixels used in our half-reduced neighborhood (top) and pixels for computing averaged values of the neighborhood (bottom).

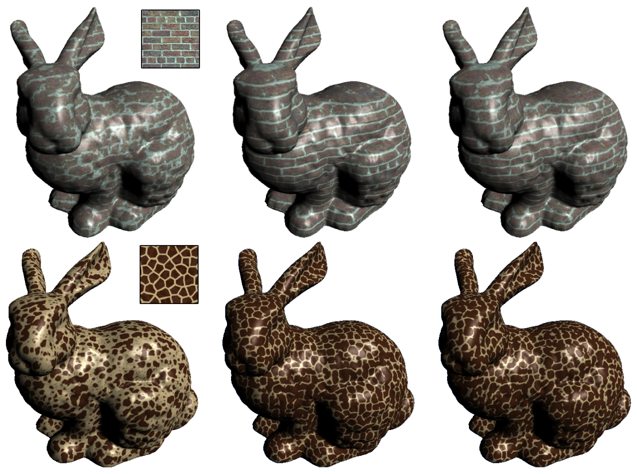

Fig. 4. Comparison of different neighborhoods used during the correction phase of the 3D texture synthesis. First column: Reduced four-pixel neighborhood. Second column: Full 5×5 neighborhood consisting of 25 pixels. Third column: Our new half-reduced neighborhood consisting of 9 pixels.

Instead of a color image, the result of the synthesis is a map of texture coordinates that is used to sample the exemplars and output the final image.

Using this map of texture coordinates, we are able to sample arbitrary input images, and not only the color exemplar. Hence, additional channels — like alpha channels, displacement maps, specularity maps, etc. — can be synthesized without affecting the performance of the synthesis/morphing at all. This is an advantage over other methods that do not keep track of the original pixel locations in the input exemplars.

6 Higher-dimensional Texture Synthesis

Our algorithm can theoretically be extended to support texture synthesis and morphing in arbitrary dimensions using also higher-dimensional input exemplars. Though it might already be hard to define what 4D texture morphing exactly means, a possible application could be the synthesis of material properties that change spatially over time, for example aging wood or rusted metal.

The idea for higher-dimensional synthesis generalizes the modifications we made in the previous section. For a target dimension N and M-dimensional exemplars, each exemplar defines a view onto an M-dimensional "face" of the synthesis texture, the $N-M$ remaining dimensions are not specified by this view.

The initialization of S_{-1} can be defined as $S_{-1}(P) = P_{x_1,\cdots,x_M} \bmod s_{ej}$. Similarly, the coordinate upsampling needs to repeat the upsampled coordinates for $N-M$ "slices". The extensions of the jitter function to M dimensions are trivial. For optimizing the synthesis during the correction phase the neighborhoods $N_{S_i|x_1,\cdots,x_m}$ need to be aligned with the exemplar's orientation, which can be achieved by defining the offsets in $\mathcal{N}_{x_1,\cdots,x_m}$ according to an M-dimensional neighborhood. An interleaved correction scheme would specify a sub-pass pattern in a 2^N-dimensional cube.

7 Results

We implemented our algorithm using C++ and execute it on the CPU in order to facilitate experimentation and integration into existing biomedical visualization software. As exemplars we used 64×64 or 128×128 pixel textures. For the 2D morphing, the target size is 512×512. Our generated solid textures have 128^3 or 256^3 voxels. We used the appearance space attributes discussed in section 3 and projected the 150-dimensional vectors onto the first eight components using the PCA implementation from [8]. Several new appearance space attributes such as neighborhood variance and principal direction of texture features were extensively tested. We found that RGB color, signed feature distance, and gradient estimates are best with respect to visual quality and computational cost [16]. We performed two full correction passes per synthesis level. We used the reduced or our half-reduced neighborhood for the 2D outputs, and the half-reduced or a full 5×5 neighborhood for 3D synthesis and morphing.

Figure 5 shows some of our 2D morphing results with two exemplars to demonstrate how the transition between structures is generated. Note how the algorithm gradually defines a coherent transition of features and morphs between them, even if the exemplars are extremely different. Morphing examples with several exemplars and complex weight maps are shown in figure 6.

Examples of our 3D synthesis, including one with an additional channel for displacement mapping, are given in figures 7 and 8. As can be seen, the generated 3D solid textures coherently reflect the characteristics of the materials. However, a smoothing of the fine details can be observed — a problem that is common in solid texture synthesis algorithms (compare for example to [26, 10]).

The charts in figure 9 illustrate how our algorithm performs using the settings stated above, executed using one core of a 2.13 GHz Intel® Core™ 2 Duo CPU with 2 GB RAM. The 2D morphing using two exemplars and a reduced neighborhood takes less than 12 seconds on average. With our half-reduced neighborhood the timings are still below 17 seconds. Our examples with three and four exemplars needed less than 18 and 25 seconds respectively for the morphing with a reduced neighborhood.

For 3D synthesis, even with a full 7×7 neighborhood (which we never use in practice) our algorithm needs no more than 15 minutes to synthesize a 128^3 solid texture cube, and 3D morphing with twice as many exemplar views takes less than 30 minutes. The half-reduced neighborhood performs with little more than 5 minutes for 3D synthesis and about 12 minutes for 3D morphing very fast while producing high-quality results. For morphing solid textures with a resolution of 256^3 voxels, our algorithm needs between 110 minutes (half-reduced neighborhood) and 140 minutes (full 5×5 neighborhood). Note that doubling the target resolution leads to eight times as many voxels in the solid cube. With an implementation on the GPU we expect a significant performance boost, possibly by several orders of magnitude.

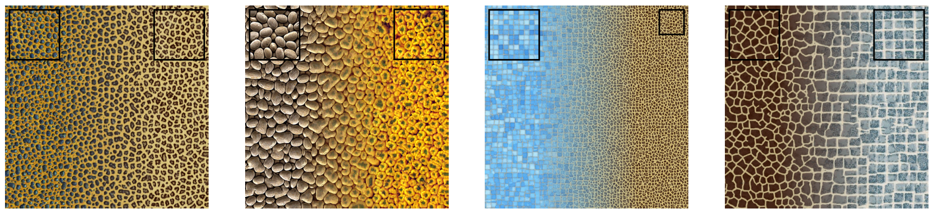

Fig. 5. 2D morphing results using two input exemplars and linear weight maps. The examples illustrate how the morphing algorithm finds coherent transitions between the features.

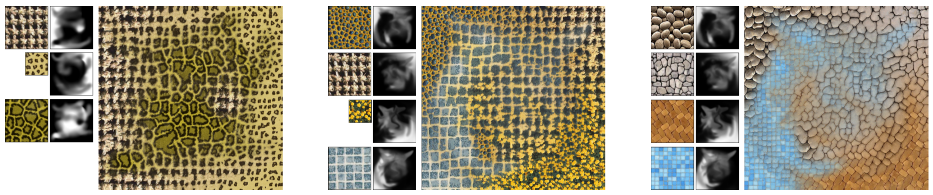

Fig. 6. 2D morphing results using three and four input exemplars and complex weight maps. Note that the weight maps are normalized and actually of the same size as the synthesized target texture.

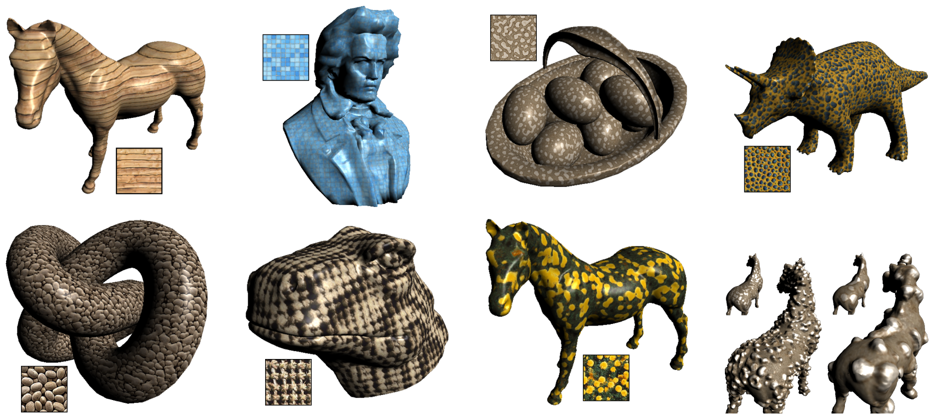

Fig. 7. 3D texture synthesis results. The generated solid textures have been used to render different 3D geometries. Bottom-right: Intensity values used as additional channel for displacement mapping.

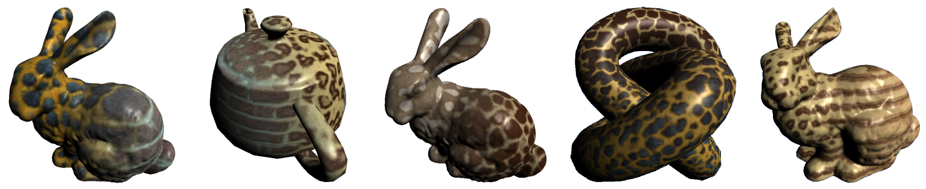

Fig. 8. 3D morphing results using two exemplars and linear weight maps. The last two examples use a morphed texture of size 256^3.

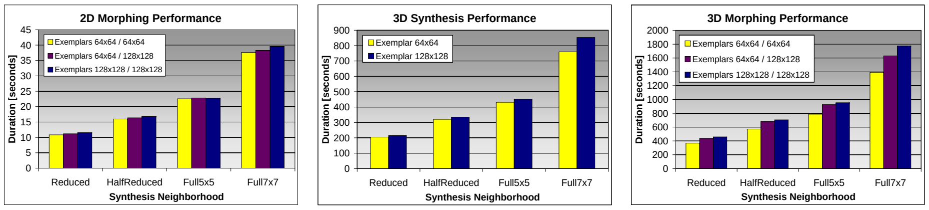

Fig. 9. Performance charts for our algorithm with different neighborhoods (without initialization times). Left: 2D texture morphing with two exemplars; Middle: 3D texture synthesis with one exemplar; Right: 3D texture morphing with two exemplars. The target texture size for the 2D tests was 512×512 and for the 3D tests 128^3.

Our algorithm has some limitations. As most pixel-based approaches the algorithm has difficulties to synthesize or morph textures with large features or where the features have a semantic meaning to humans. Problems also occur with near stochastic textures such as clouds and slightly crumbled paper. Textures with features of very different scale represent a particular problem for the morphing, because no common structures can be found that could be morphed into each other. Currently, we are working on texture morphing supported by *frequency-dependent feature scaling* (FDFS), a spatially varying exemplar scaling based on the dominant frequency of the textures. Depending on the frequencies of the exemplars and their weights $W^j(P)$ we compute a scaling factor $s(P)$ for each exemplar in order to locally match the feature sizes to each other.

7.1 Texture-enhanced Direct Volume Rendering

To further demonstrate the flexibility of the proposed morphing algorithm, we present examples of a new methodology termed *texture-enhanced DVR* [17]. In traditional DVR features of interest in a data set are classified by transfer functions (TF) that map data values to colors and opacities. Texture-enhanced DVR enriches this pipeline by supporting textures for encoding materials and conveying additional information. A new type of TF, called *texture transfer function*, maps from data values to weighting curves for the input textures, and therewith provides the weights for a subsequent 3D texture morphing step, which coherently morphs materials in transition zones. Figure 10 shows two examples of texture-enhanced DVR. On the left we show how to exploit the synthesis of additional channels (section 5.5) for screen-door transparency effects, the example on the right illustrates the use of texture morphing on a single layer.

7.2 Comparison with Kopf et al.'s Algorithm

In figure 11, we compare the synthesis quality of our proposed algorithm with results of the algorithm based on energy minimization recently presented by [10]. Disregarding the different illumination settings, the results for the first two exemplars (”woodwall” and ”animalskin”) appear to be of very similar quality. Our result for the ”woodwall” texture seems to have more structure than Kopf et al.'s result, which looks rather smooth. Although our result for ”animalskin” shows more variance in the size of the features, it

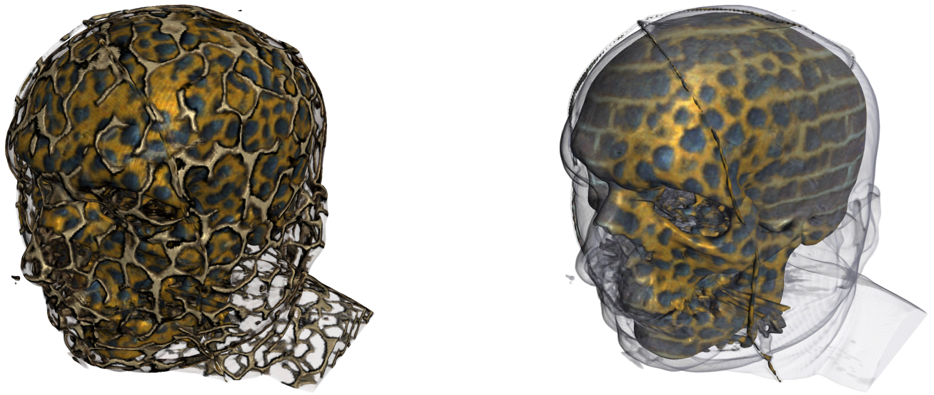

Fig. 10. Texture-enhanced Direct Volume Rendering uses textures to encode different materials and convey additional information. Left: Screen-door transparency effects; Right: Encoding imaginary materials using textures.

better reflects the structures within the blue spots. On the other hand, the boundaries between texture features look sharper in Kopf et al.'s solid texture, which can also be seen in the right hand image, where features (the tomatoes) are more distinct and the green leaves are not suppressed as much. However, our technique is significantly faster. Including the initialization, we need about 6 minutes when using the half-reduced neighborhood and less than 9 minutes using the full 5×5 neighborhood. In contrast, Kopf et al. reported up to 90 minutes required for the synthesis. Another advantage of our method is that additional channels can be synthesized without affecting the running time of the synthesis at all. In contrast, the cost of Kopf et al.'s method directly depends on the number of channels in the exemplar. Furthermore, Kopf et al.'s algorithm does not allow a straightforward implementation on the GPU, because a continuous update of the histogram is required.

8 Conclusion and Future Work

We presented a new and fast exemplar-based texture morphing algorithm for two, three and theoretically also higher dimensions as an extension of the pure 2D synthesis algorithm proposed in [13, 14]. Because our modifications obey the design principles of the original algorithm, our new contribution still allows an implementation on parallel stream-processing hardware. Even without hardware acceleration our current CPU-based implementation is already faster than comparable 3D synthesis methods.

The steps of the original algorithm have been generalized to support morphing with arbitrary many exemplars and higher-dimensional synthesis. A more intuitive jitter function and a new compact neighborhood suitable for fast 3D synthesis have been introduced and its performance evaluated.

In the future we want to further improve the synthesis quality. Spatially varying scaling based on the dominant frequency of the exemplars could support the morphing

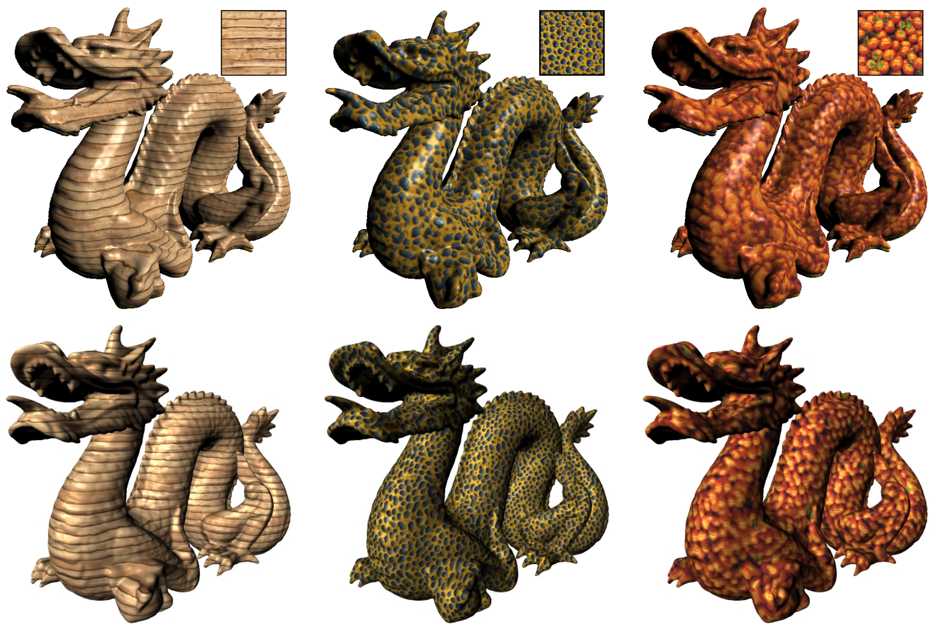

Fig. 11. Comparison of our 3D texture synthesis (top row) using the half-reduced neighborhood (left, right) and a full 5×5 neighborhood (center) with Kopf et al.'s (bottom row) algorithm using the same input exemplars and target resolution (from `http://www.johanneskopf.de/publications/solid/results/index.html`.)

to create better transitions between exemplars with structures of very different scale. Another interesting feature is the synthesis and morphing along time-varying vector and tensor fields.

References

1. ASHIKHMIN, M. Synthesizing natural textures. In *Proceedings of I3D '01* (2001), ACM Press, pp. 217–226.
2. BAR-JOSEPH, Z., EL-YANIV, R., LISCHINSKI, D., AND WERMAN, M. Texture mixing and texture movie synthesis using statistical learning. *IEEE Transactions on Visualization and Computer Graphics 7*, 2 (2001), 120–135.
3. DE BONET, J. S. Multiresolution sampling procedure for analysis and synthesis of texture images. In *Proceedings of SIGGRAPH '97* (1997), ACM Press, pp. 361–368.
4. DISCHLER, J.-M., GHAZANFARPOUR, D., AND FREYDIER, R. Anisotropic solid texture synthesis using orthogonal 2d views. *Computer Graphics Forum 17*, 3 (1998), 87–95.
5. EFROS, A. A., AND FREEMAN, W. T. Image quilting for texture synthesis and transfer. In *Proceedings of SIGGRAPH '01* (2001), ACM Press, pp. 341–346.
6. EFROS, A. A., AND LEUNG, T. K. Texture synthesis by non-parametric sampling. In *Proceedings of ICCV '99* (1999), IEEE Computer Society, pp. 1033–1038.
7. HEEGER, D. J., AND BERGEN, J. R. Pyramid-based texture analysis/synthesis. In *Proceedings of SIGGRAPH '95* (1995), ACM Press, pp. 229–238.

8. INTEL$^{\circledR}$ CORPORATION. Open Source Computer Vision Library. URL: `http://sourceforge.net/projects/opencvlibrary/` [checked: 07/24/2009].
9. JAGNOW, R., DORSEY, J., AND RUSHMEIER, H. Stereological techniques for solid textures. *ACM Transactions on Graphics (Proceedings of SIGGRAPH '04) 23*, 3 (2004), 329–335.
10. KOPF, J., FU, C.-W., COHEN-OR, D., DEUSSEN, O., LISCHINSKI, D., AND WONG, T.-T. Solid texture synthesis from 2d exemplars. *ACM Transactions on Graphics (Proceedings of SIGGRAPH '07) 26*, 3 (2007), (2.1)–(2.9).
11. KWATRA, V., ESSA, I., BOBICK, A., AND KWATRA, N. Texture optimization for example-based synthesis. *ACM Transactions on Graphics (SIGGRAPH '05) 24*, 3 (2005), 795–802.
12. KWATRA, V., SCHÖDL, A., ESSA, I., TURK, G., AND BOBICK, A. Graphcut textures: image and video synthesis using graph cuts. *ACM Transactions on Graphics (Proceedings of SIGGRAPH '03) 22*, 3 (2003), 277–286.
13. LEFEBVRE, S., AND HOPPE, H. Parallel controllable texture synthesis. *ACM Transactions on Graphics (Proceedings of SIGGRAPH '05) 24*, 3 (2005), 777–786.
14. LEFEBVRE, S., AND HOPPE, H. Appearance-space texture synthesis. *ACM Transactions on Graphics (Proceedings of SIGGRAPH '06) 25*, 3 (2006), 541–548.
15. LIU, Z., LIU, C., SHUM, H.-Y., AND YU, Y. Pattern-based texture metamorphosis. In *Proceedings of Pacific Graphics '02* (2002), IEEE Computer Society, p. 184.
16. MANKE, F. Texture-enhanced direct volume rendering, July 2008. MSc thesis, Dept. of Computer Science, University of Auckland, New Zealand.
17. MANKE, F., AND WÜNSCHE, B. Texture-enhanced direct volume rendering. In *Proceedings of GRAPP '09* (Lisbon, Portugal, 2009), pp. 185–190.
18. MATUSIK, W., ZWICKER, M., AND DURAND, F. Texture design using a simplicial complex of morphable textures. *ACM Trans. on Graphics (SIGGRAPH '05) 24*, 3 (2005), 787–794.
19. PERLIN, K. An image synthesizer. In *Proc. of SIGGRAPH '85* (1985), ACM Press, pp. 287–296.
20. PORTILLA, J., AND SIMONCELLI, E. P. A parametric texture model based on joint statistics of complex wavelet coefficients. *Int. Journal of Computer Vision 40*, 1 (2000), 49–70.
21. PRAUN, E., FINKELSTEIN, A., AND HOPPE, H. Lapped textures. In *Proceedings of SIGGRAPH '00* (2000), ACM Press, pp. 465–470.
22. TONG, X., ZHANG, J., LIU, L., WANG, X., GUO, B., AND SHUM, H.-Y. Synthesis of bidirectional texture functions on arbitrary surfaces. In *Proceedings of SIGGRAPH '02* (2002), ACM Press, pp. 665–672.
23. TONIETTO, L., AND WALTER, M. Texture metamorphosis driven by texton masks. *Computers & Graphics 29*, 5 (2005), 697–703.
24. TURK, G. Generating textures on arbitrary surfaces using reaction-diffusion. In *Proceedings of SIGGRAPH '91* (1991), ACM Press, pp. 289–298.
25. WEI, L.-Y. *Texture Synthesis by Fixed Neighborhood Searching*. PhD thesis, Stanford University, 2002.
26. WEI, L.-Y. Texture synthesis from multiple sources. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Sketches & Applications* (2003), ACM Press, pp. 1–1.
27. WEI, L.-Y., AND LEVOY, M. Fast texture synthesis using tree-structured vector quantization. In *Proceedings of SIGGRAPH '00* (2000), ACM Press, pp. 479–488.
28. WITKIN, A., AND KASS, M. Reaction-diffusion textures. *SIGGRAPH Computer Graphics 25*, 4 (1991), 299–308.
29. WORLEY, S. A cellular texture basis function. In *Proceedings of SIGGRAPH '96* (1996), ACM Press, pp. 291–294.
30. ZHANG, J., ZHOU, K., VELHO, L., GUO, B., AND SHUM, H.-Y. Synthesis of progressively-variant textures on arbitrary surfaces. *ACM Transactions on Graphics (Proceedings of SIGGRAPH '03) 22*, 3 (2003), 295–302.