DOI: 10.1111/j.1467-8659.2010.01739.x

Eurographics Symposium on Rendering 2010
Jason Lawrence and Marc Stamminger
(Guest Editors)

Volume 29 (2010), Number 4

Patch-based Texture Interpolation

Roland Ruiters, Ruwen Schnabel and Reinhard Klein

Institute for Computer Science II, University of Bonn †

Abstract

In this paper, we present a novel exemplar-based technique for the interpolation between two textures that combines patch-based and statistical approaches. Motivated by the notion of texture as a largely local phenomenon, we warp and blend small image neighborhoods prior to patch-based texture synthesis. In addition, interpolating and enforcing characteristic image statistics faithfully handles high frequency detail. We are able to create both intermediate textures as well as continuous transitions. In contrast to previous techniques computing a global morphing transformation on the entire input exemplar images, our localized and patch-based approach allows us to successfully interpolate between textures with considerable differences in feature topology for which no smooth global warping field exists.

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

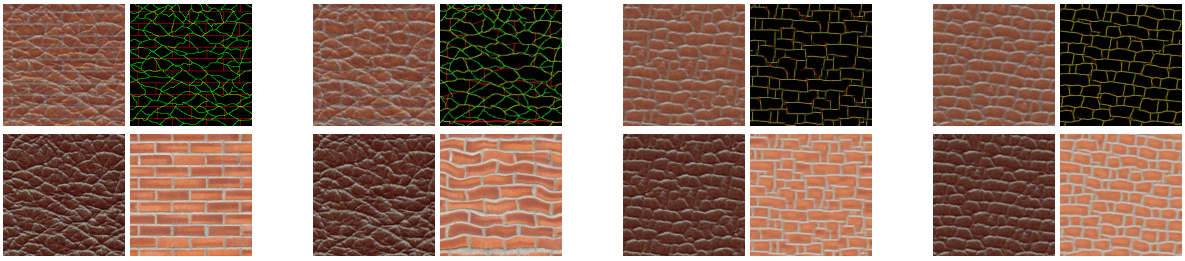
1. Introduction

Realistic textures are one of the most important factors determining the quality of a rendering. However, manually designing textures - procedurally or painted - is extremely time consuming, while relying on photographs limits the designer to available material samples. Moreover, data acquisition from real world samples often requires tedious manual postprocessing and editing to achieve the desired appearance. One approach to overcome these problems are data-driven techniques. The idea is to first acquire several textures from real-world samples, allowing the designer to create new textures by interpolating between the captured samples. This approach has already been successfully applied in many other areas, like the acquisition of animation data (eg. [KGP02]), the capture of BRDFs (eg. [MPBM03]) or the synthesis of faces (eg. [BV99]).

A second important application for texture interpolation is the creation of spatially varying textures. Many objects show continuous transitions between two textures, either naturally occurring, like for example natural variations in fur or leather, or caused by wear or weathering. To create such effects, it is not sufficient to only create an interpolated material, but a smooth transition between two materials is also necessary.

However, creating meaningful interpolations for textures is a difficult problem. Simple linear interpolation between textures usually does not give reasonable results, as is illustrated in Figure 1a. Blending only works satisfactory when features, like for example edges, ridges or cracks, in both textures are aligned to each other. In [MZD05], it was therefore proposed to solve this problem by first computing a global warping field, which aligns corresponding features in the two texture samples, and then interpolating between the warped images. However, this approach suffers from two problems: Firstly, the amount of deformation has to be kept low, as otherwise the appearance of the texture changes considerably and artifacts occur in strongly warped regions. Thus, a strong regularization must to be added to penalize strong warping. As a result of this, not all features can be brought into exact alignment. Secondly, this approach can only be used if a continuous warping field can be found. If there are large topological differences between the layout of features in the two images, this is not possible since the topology of the features cannot be changed by applying a continuous mapping. Figure 1b shows an example where

† e-mail: {ruiters, schnabel, rk}@cs.uni-bonn.de



(a) Linear Blending *(b)* Warping *(c)* Linear patch blending *(d)* Warped patches (our technique)

Figure 1: *Comparison of different texture interpolation approaches. Top left image is the final result, top right demonstrates the feature alignment obtained by the techniques. The bottom row shows input images after warping/synthesis but prior to blending (Note: for our technique these are only shown for demonstration purposes. The technique interpolates the neighborhoods prior to the synthesis and performs the synthesis with the blended patches, therefore these images are usually not created). The images correspond to an interpolation amount of 0.4.*

due to these reasons it was not possible to bring all features into alignment.

Based on the assumption that a texture can be modeled as a Markov-Random-Field (MRF), which is characterized by a collection of local image neighborhoods only, we propose to overcome these drawbacks by performing the interpolation directly on these local neighborhoods. We first find corresponding neighborhoods with similar feature topology, and then warp and blend these patches to create a novel set of neighborhoods defining the MRF of the interpolated texture. In the next step, we use texture optimization [KEBK05] to synthesize a new image from this interpolated MRF. In contrast to approaches finding one global warping transform, our local approach requires less deformation on the patch level, since corresponding patches are chosen to have a similar feature layout and topology. Moreover, topology changes are resolved during the texture optimization which reassembles different patches in a suitable manner. Finally, we combine the patch based synthesis technique with statistical texture synthesis [PS00] to preserve high frequency features, which would otherwise be lost because of the patch blending and resampling necessary for warping.

As shown in Figure 1d, using this approach, a texture with very exact feature alignment and an intuitive looking and consistent overall structure can be synthesized. Note, that it is not sufficient to only linearly blend patches with similar topology, as this way it is not possible to bring the features in good alignment (1c). Furthermore, the overall structure of the resulting texture is still dominated by the right angles in the brick texture, even though the interpolation amount is set to 0.4 and thus the result should be closer to the leather.

Using our approach, interpolation is thus possible for a large class of materials, even when dealing with considerable topological differences in the layout of the features in the two textures.

In summary, we make the following contributions:

- We propose a novel texture interpolation technique, based on warping small patches onto each other and then reassembling a texture from these patches using texture synthesis
- We introduce two improvements to the texture optimization algorithm [KEBK05] which are especially important in the case of texture interpolation, as they improve the synthesis performance when the patches available for synthesis are not very consistent.
- We combine patch based texture optimization [KEBK05] with a statistical texture synthesis algorithm [PS00] to conserve high frequency details otherwise lost by the warping.

2. Previous Work

2.1. Texture Interpolation

Existing approaches to texture interpolation can roughly be grouped in three classes.

One approach is to compute a warping field which maps one texture onto the other to align features in the two images and then blend between the two warped images. In [LLSY02], manually specified feature correspondences are used to create the warping field, whereas in [MZD05] the warping is computed, using an automatically extracted feature channel. In [LW07], a similar warping technique is used to create smooth transitions in synthesized video sequences of natural phenomenons like water, clouds and fire. However, computing one warping transformation for the whole image is only possible when the images are topologically similar and the features are distributed in such a way that unique correspondences and a continuous warping field can be found.

A second class of techniques is based on texture synthesis. In [Wei02], textures are interpolated by minimizing an energy function, which contains a weighted sum of the differences to the nearest patch in each of the input images. To improve the stability of the synthesis, an additional user defined feature channel is used. In [ZZV*03], a binary texton

map is used instead of the feature channel. To interpolate two textures, first an intermediate texton map is created. Then constrained texture synthesis is uesed to synthesize from the input textures two textures with aligned features which are then blended. The interpolated texton mask can be obtained via linear blending [ZZV*03, TW]. Recently, it was instead suggested [RLW*09] to use advection to create the interpolated mask. However, all of the approaches based on texton masks suffer from the problem, that the texton masks are interpolated globally. The alignment of the features is either – in the case of linear blending – not taken into account, or can only be corrected locally via the advection performed in [RLW*09].

Approaches which first warp an input image and then perform texture synthesis to restore details lost by the warping have also been proposed. In [FH07] a user specified warping is used, whereas in [LWX*09] a similar approach is used to create an image mixing the appearance of one texture with the feature distribution of a different texture.

A third class of techniques [HB, PS00, BJEYLW01] is based on the extraction and interpolation of image statistics. From these interpolated statistics a new image can then be synthesized. However, for many complex images, statistical synthesis alone does not provide very satisfactory results. These techniques can also be combined with other synthesis techniques to help to preserve image details, which would otherwise be lost [MZD05].

Function: TextureInterpolation(I_1, I_2, F_1, F_2, α)
Add distance channel to I_l and I_2
Initialize image I with noise
foreach *resolution r and neighborhood size s* **do**
  $\mathcal{N}'$ = InterpolateNeighborhoods(I_1, I_2, F_1, F_2, α,r,s)
  for $i = 1...N$ **do**
    Find nearest neighbors in $\mathcal{N}'$ for patches in I
    Update neighborhood centers via relaxation
    Update pixel colors in I via Mean-Shift
    Perform statistical synthesis
  end
  return I;
end

Function: InterpolateNeighborhoods(I_1, I_2, F_1, F_2, α, r, s)
foreach $l = \{1,2\}$ **do** $\mathcal{N}_l$ = ExtractNeighborhoods(I_l, r, s)
foreach $l = \{1,2\}$ **do**
  $\hat{l} = 3 - l$
  foreach $N_j^{(l)} \in \mathcal{N}_l$ **do**
    Find k nearest neighbors $N^{(\hat{l})}_{c^{(l)}(j,1)}, \cdots, N^{(\hat{l})}_{c^{(l)}(j,k)}$ of $N_j^{(l)}$
    foreach $i = 1 \cdots k$ **do**
      Find warping W between $N_j^{(l)}$ and $N^{(\hat{l})}_{c^{(l)}(j,i)}$
      Create interpolated patch N'
      $\mathcal{N}' = \mathcal{N}' \cup \{N'\}$
    end
  end
  return $\mathcal{N}'$
end

Pseudocode 1: *Our texture interpolation algorithm*

2.2. Texture synthesis

Given a small exemplar image, texture synthesis techniques create a larger texture with a similar visual appearance. In the last years, texture synthesis has received much attention and a considerable amount of work in this area has been published. Because of the limited space available, we will therefore only cite those works directly related to our technique and refer the reader to a recent state-of-the-art report [WLKT] for a more comprehensive overview.

Texture synthesis approaches can be classified into techniques based on image statistics and neighborhood-based approaches. The first neighborhood-based techniques [EL99, WL] synthesize the image by sequentially selecting the color for each pixel from the best corresponding neighborhood in the input images. An alternative approach is used by patch-based techniques (e.g. [PFH, LLX*01, EF01, KSE*03]), which work by copying larger patches into the synthesized texture. To ensure pattern continuity during patch assembly, in [WY04] the individual patches are warped by aligning a binary feature map via a thin plate spline transformation. This approach is similar to our warping technique, but only used during synthesis and not for texture interpolation. Texture optimization techniques [KEBK05, HZW*06, KFCO*07] combine pixel and patch based approaches by considering texture synthesis as an optimization problem, which is solved by minimizing an energy function. A different approach is used by techniques, which decompose the image into several subbands and then enforce statistics like color histograms [HB] and cross-correlation between subbands [PS00].

3. Overview

Given two images I_1 and I_2 which are samples of textures T_1 and T_2 respectively, we want to synthesize a new image I' of arbitrary dimensions sampled from a texture T' which perceptually lies between T_1 and T_2. This is controlled by a user specified interpolation amount $\alpha \in [0,1]$, where 0 corresponds to T_1 and 1 to T_2. Based on the common assumption that a texture can be modeled as a Markov-Random-Field, it is possible to synthesize a new texture only from a collection $\mathcal{N} = \{N_j\}_{j=1...M}$ of local neighborhoods extracted from the input image. Since these local neighborhoods characterize a texture completely, interpolation between textures can also be regarded as the creation of a new interpolated set of neighborhoods. To interpolate between textures T_1 and T_2, we thus first extract the neighborhood sets $\mathcal{N}_1$ and $\mathcal{N}_2$ from I_1 and I_2. Based on these, a new set of interpolated neigh-

borhoods $\mathcal{N}'$ is computed which is then used to synthesize I'.

In Pseudocode 1, an overview of the different steps of our algorithm is given. More details on the individual steps are then given in the following sections.

To obtain $\mathcal{N}'$, we search for each $N \in \mathcal{N}_l$, $l \in \{1,2\}$, k corresponding neighborhoods in the other set $\mathcal{N}_{\hat{l}}$, with $\hat{l} = 3 - l$. The interpolated neighborhoods are then created using a warping transformation to first align features between N and these k corresponding neighborhoods and then linearly blending the patches. For this, we use binary feature maps F_1 and F_2 marking the positions of relevant features in both input images which should be aligned during the interpolation.

Our texture synthesis is based on the texture optimization algorithm of Kwatra et al. [KEBK05]. The texture is synthesized in multiple steps from coarse to fine with successively increasing image resolution and decreasing patch size to allow the algorithm to synthesize textures with features at different scales. For each combination of resolution and patch size, we create the set of interpolated neighborhoods $\mathcal{N}'$ from the input textures. Compared to ordinary texture synthesis, the set $\mathcal{N}'$ is less consistent due to the warping of the individual patches. In particular, it is no longer guaranteed that for each patch neighboring patches which are identical in the region of overlap exist. To compensate for this, we therefore do not employ a regular grid of patches during the synthesis, but allow for a more irregular distribution of the pasted neighborhoods. In addition, a relaxation step moves the pasted neighborhoods in such a way that the error between overlapping neighborhoods is minimized. Finally, to preserve high frequency details, we perform an additional statistical synthesis step based on the technique described in [PS00] in each iteration of the algorithm.

4. Neighborhood interpolation

The patch interpolation creates an interpolated set $\mathcal{N}'$ of neighborhoods by finding corresponding neighborhoods in the two input textures I_1 and I_2 and then interpolating between these.

4.1. Correspondence search

First, the sets $\mathcal{N}_1$ and $\mathcal{N}_2$ of all possible neighborhoods of the desired size are extracted from I_1 and I_2. To find corresponding patches, we then search for each $N_j^{(l)} \in \mathcal{N}_{(l)}$, $l \in \{1,2\}$, the nearest k corresponding neighborhoods in the other set $\hat{l}$: $N^{(\hat{l})}_{c^{(l)}(j,1)}, \ldots, N^{(\hat{l})}_{c^{(l)}(j,k)} \in \mathcal{N}^{(\hat{l})}$, where $c^{(l)}(j,i)$ is the index of the i-th nearest neighbor of $N_j^{(l)}$ in $N^{(\hat{l})}$. For this, we use two binary feature maps F_1 and F_2. These maps could be generated automatically using a feature detector. However, depending on the input images very different elements might constitute features. We tried both the compass operator [RT99], which was suggested in [MZD05] for this purpose, as well as several edge detectors. Still, for many images the results were not very satisfactory and thus for this paper we use manually created feature maps.

Ideally, we would like to find corresponding neighborhoods which have features that can be aligned under minimal deformation. For instance, a good distance measure would probably be the warping error D_W given in Eq. (1). However, computing all pairwise warpings is prohibitively expensive and cannot be performed in reasonable time. A faster alternative is to compute the difference of the binary feature channels. However, this results in large errors even for only slightly misaligned features. Thus, as was suggested in [LH06], we perform a distance transform to add an additional distance channel to the neighborhoods, in which for each point the distance to the nearest feature point is stored. The L_2 error in this distance channel is a considerably better measure how well the features in the two images correspond to each other. We use the metric $\|W_C \bullet (N_i^{(1)} - N_j^{(2)})\|_2$ to compute the distance between two neighborhoods $N_i^{(1)}$ and $N_j^{(2)}$. Here, W_C specifies weights for the channels of the neighborhood. We thus include both the color and distance channels during the search to prefer warpings between patches with similar color and feature distribution. Using this distance measure, the k nearest patches can easily be found via PCA accelerated KD-tree search and thus a very fast lookup becomes possible.

It is important to perform these searches for the neighborhoods in both textures, first using $\mathcal{N}_1$ as input and searching in $\mathcal{N}_2$ and then the other way around. If the searches are not performed symmetrically it may be that many neighborhoods in the second image are not nearest neighbor to any of the neighborhoods from the first texture and would not be taken into account at all. This is a problem, since we want to ensure that the extracted patches are not only similar to the input images but also complete in the sense that everything visible in the two input images is also represented in the extracted patch set. In [WHZ*08] and [SCSI08], this problem is described in more detail in the context of the creation of image compactions.

4.2. Patch interpolation

Once two corresponding neighborhoods $N_j^{(l)}$ and $N^{(\hat{l})}_{c^{(l)}(j,i)}$ have been found, we perform a color adjustment. To this end, the mean and variance of the color channels in both input textures are computed, and then linearly interpolated according to α. The color values of the two neighborhoods are then shifted and scaled accordingly.

To compute the feature aligning warp, we search for a *thin-plate spline* (TPS) transform θ [Duc77] that maps the features in one neighborhood onto features in the other neighborhood. We create two point sets $P^{(1)}$ and $P^{(2)}$ from

the sections of the two feature maps F_1 and F_2 which correspond to the currently processed neighborhoods. For this, we store the coordinate of each non-zero pixel in the point sets $(P^{(i)} = \{(x,y)|F_i(x,y) \neq 0\})$. We then use the algorithm of Chui and Rangarajan [CR00], which uses deterministic annealing, to simultaneously find the correspondence between $P^{(1)}$ and $P^{(2)}$ and the aligning transform.

The algorithm iterates between updating the correspondences between points and calculating a new TPS according to the found correspondences. It is based on softassignments, captured in a matrix $\mathbf{M}$. Each entry m_{ij} of M gives the probability that the pair of points $P_i^{(1)}, P_j^{(2)}$ correspond to each other. The algorithm thus minimizes the error function

$$D_W(M,\theta) = \sum_i \sum_j m_{ij} \|\theta(P_i^{(1)}) - P_j^{(2)}\|^2 + R(\theta), \quad (1)$$

where R is a regularization term penalizing strong deformations. During the annealing, as the temperature parameter is decreased, the softassignments in $\mathbf{M}$ become successively more distinct until they finally converge against one-to-one correspondences. This technique allows to find the correspondences in a considerably more robust way than algorithms like *iterated closest points* [BM92], which are based on hard assignments only. To obtain the interpolated patches, we compute both TPS transformations, $\theta_{1,2}$ mapping P_1 to P_2 and $\theta_{2,1}$ mapping P_2 to P_1, during the point matching algorithm, using the same correspondences for both. The interpolated transforms are given by interpolating the identity transform and the TPS, i.e. $\theta'_{1,2} = (1-\alpha)\theta_{1,2} + \alpha\mathbf{I}$ and $\theta'_{2,1} = \alpha\theta_{2,1} + (1-\alpha)\mathbf{I}$. These two transformations are then used to warp one patch by α and the other patch by $1-\alpha$ to align the features and the following blended patch is added to $\mathcal{N}'$:

$$N'(x) = (1-\alpha)N_1(\theta'_{2,1}(x)) + \alpha N_2(\theta'_{1,2}(x))$$

Even though we only consider patches, especially for larger neighborhood sizes it is still possible that one patch has features that cannot be mapped onto any feature in the other patch. Thus, we need a very robust handling of outliers during the computation of the TPS. We slightly modified the original algorithm by assigning outlier points to a point at their own position during the last iteration for the computation of the TPS. This adds a certain inertia to the iteration, avoiding large deformations that align outliers with very distant features. In our experiments, this improved the robustness of the point matching.

We only want to use those neighborhoods during the synthesis which achieve a good alignment of the warped feature maps and for which the actual images fit well to each other. Thus, we compute the similarity between the two warped neighborhoods and only keep those where the similarity is above a certain threshold. For this similarity, we combine the difference of the two warped distance channels and the normalized cross correlation of the color channels. The relative weight of distance channel and color channels is a user selectable parameter controlling the importance of the feature channel in relation to the images themselves. Since it is rather difficult to decide in advance on a similarity threshold, we instead use a certain percentile of the similarities of all patches as threshold. To improve the quality of the results, it is possible to choose a high value of k, to create a large number of candidate neighborhoods, and then use a small percentile to keep only the best. However, we found that it is usually sufficient to use only one neighborhood ($k = 1$) and then keep the better 50% of the generated neighborhoods. All images shown in this paper were generated this way.

4.3. Texture Synthesis

4.4. Neighborhood based synthesis

Since we perform the actual interpolation of the image on the neighborhoods $\mathcal{N}_1$ and $\mathcal{N}_2$, any synthesis algorithm which works with a collection of neighborhoods $\mathcal{N}'$ as input could be used together with our interpolation technique. We decided to use texture optimization [KEBK05] since it achieves high quality results and iteratively optimizes the whole texture to avoid the accumulation of errors. This is especially important in our case, since the input might contain incoherent neighborhoods which cannot be combined with any other neighborhood in a sensible way. When no interpolation is performed, all possible neighborhoods are extracted from the input image. This means, that the input patches can be consistently overlapped. However, when interpolation is used, this is not necessarily the case. When two neighborhoods, which are overlapping in one image, are warped against different patches in the other image the resulting patches are no longer consistent. Therefore, a rather tolerant synthesis algorithm, which can compensate for these patches is needed.

The texture optimization algorithm iterates over different combinations (r,s) of image resolutions, obtained by downsampling the input images I_1 and I_2 by the factor r, and neighborhood sizes s to be able to synthesize features on different size scales. For each of these combinations, we independently create a new set of interpolated neighborhoods $\mathcal{N}'$. Depending on the size of features in the input images, different choices of r and s might be necessary. We usually perform the synthesis in three steps with $(r = 2, s = 33), (r = 2, s = 17), (r = 1, s = 17)$. Only for images with very large and regular features, we use an additional iteration with $(r = 2, s = 49)$.

For a given combination (r,s), an image is synthesized by repeatedly choosing matching neighborhoods from $\mathcal{N}'$ for a set of center points in the current synthesis result I. Then I is updated by pasting the matching patches into I while averaging their contributions in the respective regions of overlap. In

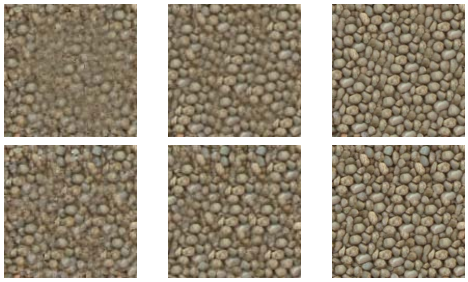


Figure 2: *Difference between neighborhoods aligned to a regular grid (top) and using irregular neighborhood centers (bottom) after each of the synthesis iterations.*

the original algorithm, the matching is performed for neighborhoods centered around points located in a regular sparse grid, in such a way that for each pixel several overlapping neighborhoods are available. However, using a regular grid does not give good results in our case. In traditional texture syntheses, all possible neighborhoods are extracted from the input image. This means, that input neighborhoods are available for all possible positions of a patch center. However, when using the interpolated neighborhoods, it might happen that a good patch is available for one center point, but not for the center directly adjacent to this point. When this is not taken into account during the synthesis, the synthesized images become blurred and only a small subset of the available neighborhoods is used, resulting in repetitive images. Thus, we do not search for neighborhoods centered around points on a regular grid, but instead use an irregular distribution. Within each cell of the original grid, we search the one neighborhood center C_i which results in the lowest error and then use this point instead. This way, we can ensure that each pixel is overlapped by enough neighborhoods, but on the other hand allow for a more flexible placement of the neighborhoods. Unfortunately, this slows down the synthesis, as a higher number of neighborhood searches is needed. Still, as can be seen in Figure 2, it improves the quality of the synthesized result considerably.

To improve the coherence between the available neighborhoods in $\mathcal{N}'$, we also looked into enforcing spatial coherence during the interpolation by assigning neighborhoods adjacent in one image to neighborhoods adjacent in the other image and using similar warpings for both. However, we did not notice a considerable difference in the resulting textures.

After the selection of the neighborhood centers C_i and the corresponding patches, we perform an additional relaxation step. Here, the matched neighborhood patches are shifted relative to each other in such a way, that the difference between overlapping parts is minimized. For this, for each neighborhood center the optimal position within a small search radius of a few pixels is searched and each neighborhood is shifted to this new position. To prevent the centers from moving too far apart, we add an additional spring term which penalizes deviations from the old position relative to

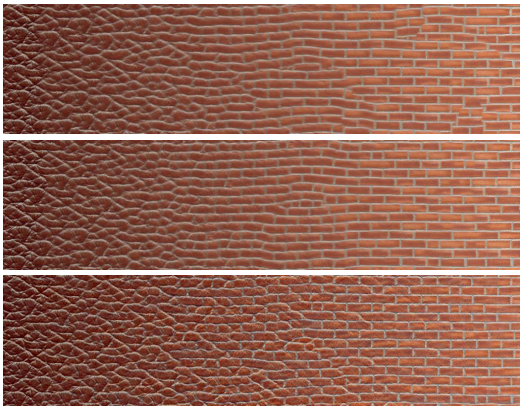


Figure 3: *Improvements obtained via relaxation (center) and additional statistical synthesis (bottom)*

the neighbor patches. This process is repeated until a local minimum is reached for all centers. This relaxation process improves the alignment of the patches, especially when there are regular structures in the texture (see Figure 3 central image). Though the relaxation step helps, the synthesis result still sometimes show discontinuities.

In the next step, the synthesized image is updated by averaging the pixel value of the overlapping neighborhoods. Here, we use iteratively reweighted least squares to obtain weights for the neighborhoods, as was suggested in [KEBK05]. Instead of averaging the colors, we perform an additional Mean Shift clustering [CM02] for each pixel to find the dominant mode and use the center of this mode as new color, as was suggested in [KFCO*07]. This helps to obtain sharper images and cope better with incoherent patches.

4.5. Statistical synthesis

Finally, we perform a statistical synthesis step, which helps to preserve high frequency details, which are otherwise easily lost during the patch interpolation. For this, we use the synthesis algorithm from [PS00] which works by enforcing image statistics and cross correlations between bands of a steerable pyramid decomposition of the input image. For each iteration of our synthesis loop, we perform one iteration of the statistical synthesis algorithm, using the implementation kindly made available by Portilla and Simoncelli on their homepage `http://www.cns.nyu.edu/~lcv/texture/`. We slightly extended the original implementation, which only works for greyscale images, by adding an additional cross correlation between the three color channels for each band of the pyramid. This correlation is then also enforced during the synthesis. Currently, we use linear interpolation to obtain the image statistics. A simple linear interpolation was already described in [PS00] and often resulted in rather unsatisfactory results when used directly for texture synthesis. Thus, further research how the

image statistics should be interpolated correctly is certainly necessary. Still, as shown in Figure 3, when used together with our neighborhood based synthesis result as initialization, the additional statistical synthesis step helps to preserve high frequency details in the interpolated textures.

To create spatially varying interpolations, we create interpolated patches for several different interpolation amounts α (we use 20 equidistantly spaced steps). We then encode the α values in an additional channel of the patches and use all patches together during the synthesis. By enforcing this channel during synthesis to resemble the user-supplied distribution mask, we then generate the spatially varying textures. The weight for this channel controls during synthesis whether the distribution mask is represented exactly, or whether smooth transitions and better synthesis results should be favored by the algorithm.

We currently perform the statistical synthesis independently for several equidistantly sampled values of α, each time only on the relevant parts of the image according to the distribution map. Then, the results are blended using α as blending weight. This approach is rather slow, but worked quite well in our experiments and did not require large changes to the original synthesis algorithm.

5. Results

In Figure 4 we show several examples of texture interpolation sequences obtained with our algorithm. The corresponding input images and feature maps are shown together with the results. As can be seen, the algorithm is able to create seamless transitions, which remain sharp and detailed during the whole sequence, for a wide range of different materials. Even when there are considerable differences in the topology and structure of the features, like for example in the transition between the leather and the brick wall, a continuous and plausible interpolation is obtained.

In the left three columns of Figure 5, we show several materials designed by interpolating between two samples of one material class. Here, the algorithm is able to create plausibly looking materials, which perceptually lie in between the input textures. Both the feature structure as well as the material details are interpolated.

In Figure 6, we directly compare our results with those reported in [RLW*09], a recently published texture interpolation technique based on texton-masks. Our algorithm is able to create smooth and detailed interpolation sequences, in which there is a continuous and seamless transition between the structures of the two materials. The advection algorithm from [RLW*09] interpolates the whole texton mask, which results in completely new structures not present in either of the input images. In contrast, our approach works on a more local level and creates a more direct transition with intermediate images which adhere considerably more to the input textures.

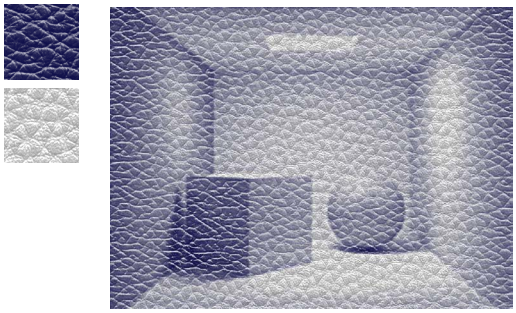


Figure 7: *Spatially-varying texture created by interpolating between the two shown material samples using a grayscale image of a Cornell Box as distribution map controlling the interpolation amount* α.

In Figure 7, we show an example for a spatially varying texture with a more complex distribution map. It can be seen, that not only the color and fine-structure of the material is varied spatially, but also the feature shape, which is interpolated from a irregular elongated pattern to a more regular triangular one. Furthermore, discontinuities are handled plausibly, with features continuing over the discontinuities.

Since we perform the warping and interpolation on the individual patches, our technique is mainly suited for materials with regular or semi-regular structures, which can be characterized by these patches and for which an alignment of corresponding features is possible. For materials with large scale structures or with purely stochastic behavior, different techniques are probably better suited. Furthermore, it must be possible to characterize the material via the feature mask. The interpolation between two wood samples in Figure 5 shows a case where the feature map does not characterize the material well and where the structure of the features is very different for the two samples.

A further limitation, visible in the left image of Figure 6 and more clearly in the top right image of Figure 5, is that we currently do not compensate for scale changes in the two textures. Therefore, the pebbles are not continuously enlarged until they reach the size of the larger green structures or the paving stones, but instead the larger structures are assembled by combining several smaller stones. Depending on the application, a continuous enlargement/shrinking of the features may be desirable, instead. This might, in the future, be implemented by adding an additional scaling step during the patch interpolation.

On a Q6600 system, the computation of one of the sequences in Figure 4 (in a resolution of 1024x256 pixels) requires about 3.5 hours, using 10 iterations for each resolution and patch size. About 20 minutes are needed for the patch interpolation, which is already optimized moderately, being implemented in C++ and parallelized on 4 cores. In contrast, large parts of the synthesis code are not yet parallelized and the statistical synthesis is implemented in MATLAB.

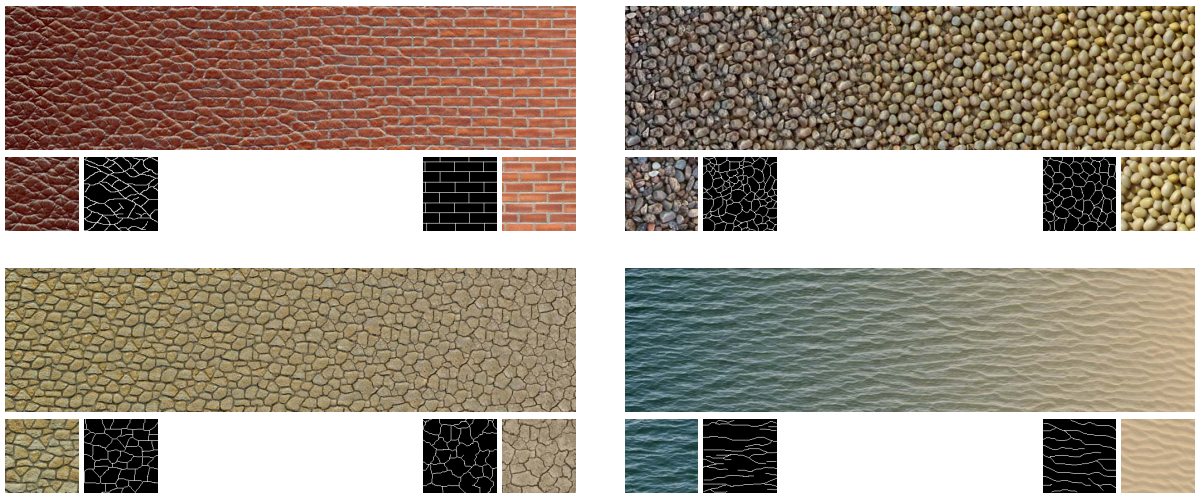


Figure 4: *Sample interpolation sequences, that were created with our technique. Input textures and feature maps are shown.*

Figure 5: *Several new materials designed by interpolating between two texture samples (All images are for* $\alpha = 0.5$*). Rightmost column shows failure-cases for materials with large differences in feature scale (top right) and materials which cannot be well characterized by line features (lower right).*

6. Conclusion and Future Work

In this work, we presented a novel texture interpolation algorithm, which locally warps and blends individual neighborhoods of the input textures and then reassembles the image using the texture optimization algorithm. In contrast to techniques based purely on warping, this way we can create consistent images, even when it is not possible to continuously warp one input image onto another. By combining this algorithm with statistical texture synthesis we are furthermore able to preserve high frequency details.

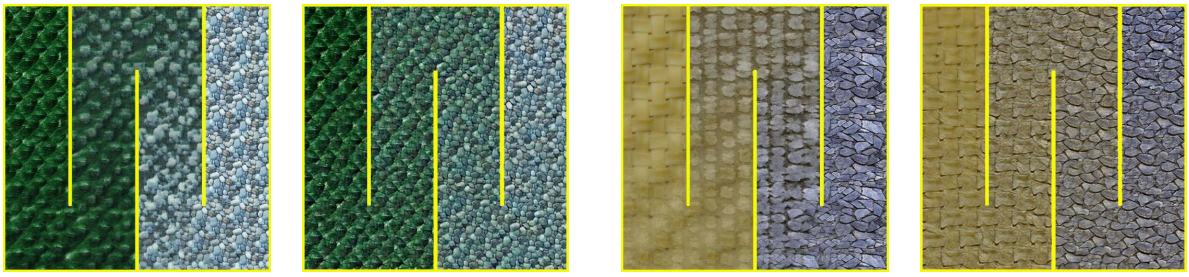
So far, we have only investigated interpolations between two input images. Especially for the synthesis of novel materials from a larger database it would be interesting to perform the interpolation between a larger number of materials. However, establishing the feature correspondences and creating intermediate patches becomes a considerably more complicated problem. A further interesting avenue of research would be a perceptual reparameterization of the resulting interpolation sequences. Often, the sequences do not seem to linearly interpolate between the two materials, but instead keep the characteristics of one of the two materials for a rather long time. A perceptual study investigating these effects might lead to techniques that reparameterize α in a way which allows for more linear transitions.

7. Acknowledgments

This work was supported by the German Science Foundation (DFG) under research grant KL 1142/4-1. We would like to thank Raoul Wessel for his suggestion to use the algorithm from [CR00] for the computation of the warp field and Michael Zschippig for his help creating the sample images. Furthermore, we would like to thank CGTextures (http://www.cgtextures.com) for allowing us to use their textures.

References

[BJEYLW01] BAR-JOSEPH Z., EL-YANIV R., LISCHINSKI D., WERMAN M.: Texture mixing and texture movie synthesis us-



[RLW*09] Our technique [RLW*09] Our technique

Figure 6: *Comparison of interpolation results from [RLW*09] with our results.*

ing statistical learning. *IEEE Transactions on Visualization and Computer Graphics 7* (2001), 120–135. 3

[BM92] BESL P., MCKAY N.: A method for registration of 3-d shapes. *PAMI 14* (1992), 239–256. 5

[BV99] BLANZ V., VETTER T.: A morphable model for the synthesis of 3d faces. In *SIGGRAPH '99* (1999), pp. 187–194. 1

[CM02] COMANICIU D., MEER P.: Mean shift: A robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence 24* (2002), 603–619. 6

[CR00] CHUI H., RANGARAJAN A.: A new algorithm for non-rigid point matching. *CVPR 2* (2000), 44 –51 vol.2. 5, 8

[Duc77] DUCHON J.: Splines minimizing rotation-invariant semi-norms in Sobolev spaces. In *Lecture Notes in Math., Vol. 571*. 1977, pp. 85–100. 4

[EF01] EFROS A. A., FREEMAN W. T.: Image quilting for texture synthesis and transfer. In *SIGGRAPH '01* (2001), pp. 341–346. 3

[EL99] EFROS A. A., LEUNG T. K.: Texture synthesis by non-parametric sampling. In *ICCV '99* (1999), vol. 2, p. 1033. 3

[FH07] FANG H., HART J. C.: Detail preserving shape deformation in image editing. *ACM Trans. Graph. 26*, 3 (2007), 12. 3

[HB] HEEGER D. J., BERGEN J. R.: Pyramid-based texture analysis/synthesis. In *SIGGRAPH '95*, pp. 229–238. 3

[HZW*06] HAN J., ZHOU K., WEI L.-Y., GONG M., BAO H., ZHANG X., GUO B.: Fast example-based surface texture synthesis via discrete optimization. *Vis. Comp. 22*, 9 (2006), 918–925. 3

[KEBK05] KWATRA V., ESSA I., BOBICK A., KWATRA N.: Texture optimization for example-based synthesis. *ACM Transactions on Graphics, SIGGRAPH 2005* (2005). 2, 3, 4, 5, 6

[KFCO*07] KOPF J., FU C.-W., COHEN-OR D., DEUSSEN O., LISCHINSKI D., WONG T.-T.: Solid texture synthesis from 2d exemplars. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2007) 26*, 3 (2007), 2:1–2:9. 3, 6

[KGP02] KOVAR L., GLEICHER M., PIGHIN F.: Motion graphs. *ACM Trans. Graph. 21*, 3 (2002), 473–482. 1

[KSE*03] KWATRA V., SCHÖDL A., ESSA I., TURK G., BOBICK A.: Graphcut textures: image and video synthesis using graph cuts. *ACM Trans. Graph. 22*, 3 (2003), 277–286. 3

[LH06] LEFEBVRE S., HOPPE H.: Appearance-space texture synthesis. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers* (New York, NY, USA, 2006), ACM, pp. 541–548. 4

[LLSY02] LIU Z., LIU C., SHUM H.-Y., YU Y.: Pattern-based texture metamorphosis. In *PG '02: Pacific Conference on Computer Graphics and Applications* (2002), p. 184. 2

[LLX*01] LIANG L., LIU C., XU Y.-Q., GUO B., SHUM H.-Y.: Real-time texture synthesis by patch-based sampling. *ACM Trans. Graph. 20*, 3 (2001), 127–150. 3

[LW07] LAI C.-H., WU J.-L.: Temporal texture synthesis by patch-based sampling and morphing interpolation. *Comput. Animat. Virtual Worlds 18*, 4-5 (2007), 415–428. 2

[LWX*09] LIU Y., WANG J., XUE S., TONG X., KANG S. B., GUO B.: Texture splicing. *Comput. Graph. Forum 28*, 7 (2009), 1907–1915. 3

[MPBM03] MATUSIK W., PFISTER H., BRAND M., MCMILLAN L.: A data-driven reflectance model. *ACM Transactions on Graphics 22*, 3 (July 2003), 759–769. 1

[MZD05] MATUSIK W., ZWICKER M., DURAND F.: Texture design using a simplicial complex of morphable textures. *ACM Trans. Graph. 24*, 3 (2005), 787–794. 1, 2, 3, 4

[PFH] PRAUN E., FINKELSTEIN A., HOPPE H.: Lapped textures. In *SIGGRAPH '00*, pp. 465–470. 3

[PS00] PORTILLA J., SIMONCELLI E. P.: A parametric texture model based on joint statistics of complex wavelet coefficients. *Int. J. Comput. Vision 40*, 1 (2000), 49–70. 2, 3, 4, 6

[RLW*09] RAY N., LÉVY B., WANG H., TURK G., VALLET B.: Material space texturing. *Comput. Graph. Forum 28*, 6 (2009), 1659–1669. 3, 7, 9

[RT99] RUZON M. A., TOMASI C.: Color edge detection with the compass operator. *CVPR 2* (1999), 2160. 4

[SCSI08] SIMAKOV D., CASPI Y., SHECHTMAN E., IRANI M.: Summarizing visual data using bidirectional similarity. *CVPR* (2008), 1–8. 4

[TW] TONIETTO L., WALTER M.: Morphing textures with texton masks. In *SIBGRAPI '04: XVII Brazilian Symposium on Computer Graphics and Image Processing*, pp. 348–353. 3

[Wei02] WEI L.-Y.: *Texture Synthesis by Fixed Neighborhood Searching*. PhD thesis, Stanford University, 2002. 2

[WHZ*08] WEI L.-Y., HAN J., ZHOU K., BAO H., GUO B., SHUM H.-Y.: Inverse texture synthesis. In *SIGGRAPH '08* (2008), pp. 1–9. 4

[WL] WEI L.-Y., LEVOY M.: Fast texture synthesis using tree-structured vector quantization. In *SIGGRAPH '00*, pp. 479–488. 3

[WLKT] WEI L.-Y., LEFEBVRE S., KWATRA V., TURK G.: State of the art in example-based texture synthesis. In *Eurographics 2009, EG-STAR*. 3

[WY04] WU Q., YU Y.: Feature matching and deformation for texture synthesis. *ACM Trans. Graph. 23*, 3 (2004), 364–367. 3

[ZZV*03] ZHANG J., ZHOU K., VELHO L., GUO B., SHUM H.-Y.: Synthesis of progressively-variant textures on arbitrary surfaces. *ACM Trans. Graph. 22*, 3 (2003), 295–302. 2, 3