Vision, Video, and Graphics (2003)
P. Hall, P. Willis (Editors)

Extending Natural Textures with Multi-scale Synthesis

O. Stahlhut

Institute of Communication Theory and Signal Processing, University of Hanover, Germany
[stahlhut@tnt.uni-hannover.de, http://www.tnt.uni-hannover.de/∼stahlhut]

Abstract

This paper presents a texture synthesis algorithm that was designed for the tile-less generation of large images of arbitrary size from small sample images. The synthesised texture shows features that are visually similar to the sample over a wide frequency range. The development of the algorithm aimed at achieving high quality results for a large range of natural textures, incorporation of the original samples in the synthesis product, ease of use and good texturing speed even with input sample data two magnitudes larger than used by previous techniques. Like other algorithms we utilise an implicit texture model by copying arbitrary shaped texture patches from the sample to the destination over a multi-scale image pyramid. Our method combines the advantages of different previous techniques with respect to quality. A mixture of exhaustive searching, massive parallel computing and the well-known LBG-algorithm ensures a good balance between texturing quality and speed.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation

1. Introduction

Texture processing plays an important role in computer graphics and vision. It involves analysis, "understanding", modelling, synthesising and segmentation of textures. All these terms are related to each other, e.g. texture synthesis usually demands a preceding analysis and texture model. Segmentation and synthesis can be interchangeable, respectively can be derived from the same basic algorithm. There is a manifold of sources for obtaining textures: digital cameras, interactive digital image creation, scans of photos or hand-drawn pictures. All these sources share the common disadvantage that the texture size is usually limited, inapplicable for a certain task and can't be extended without tiling artefacts and visible seams. On the other hand these limited texture samples can be used to derive a texture model for a texture synthesis process. Texture synthesis uses a small sample to produce a texture of arbitrary size with the same visual appearance as the sample texture. Depending on the type of algorithm these automatic generated textures can also be tileable for fast generation of even larger textures. Other fields of application are image denoising or restoration, together with texture mapping synthesis is used to produce realistic impressions of the environment within virtual worlds.

The next section will give a brief overview on previous work in the field of texture synthesis.

2. Previous Work

There is a long history of previous works on automatic texture synthesis techniques, for example procedural approaches [7, 11, 17] which are well suited for specific classes of materials (e.g., wood, marble, clouds and fire), simulation of physical processes for generation of biological patterns [19, 21, 22] or corrosion [5, 6] and other natural phenomena. We want to focus on more general texture synthesis methods related to our work, which are sample-image based. Most of the sample-image based methods can be divided into four categories:

- Feature Matching
- Markov Random Field (MRF), Gibbs Sampling with explicit model
- MRF with implicit model
- Texture patching

The feature matching methods adapt the features of a sample and output texture in an iterative process. Impressive results were first shown by [12] who matched the histograms

of two steerable image pyramids of sample and destination. This work was further improved by consideration of the cross-scale dependencies within the pyramids [2] and matching of joint statistics [18]. The feature matching methods have the disadvantage of being iterative and computational expensive. The texturing speed decreases with the number of different matched features which are necessary to provide a good texturing quality. MRF methods characterise a texture by determining significant dependencies between a central pixel and its spatial neighbours. These neighbourhood cliques are modelled by Markov Random Fields which are used for probability sampling for an iterative texture synthesis process [9, 10, 24]. Even though MRF provide a very general texture description and many good results were shown in the mentioned publications, the drawback again is computational speed. MRF methods with an implicit model avoid the construction of an explicit model [1, 20]. Local similarity of the synthesised texture and texture sample is achieved by continuous comparison of the synthesis pixel spatial neighbourhood with all given spatial neighbourhoods of the texture sample. Describing the pixel neighbourhoods as a vector involves searching in a high-dimensional vectorspace. This also leads to heavy computational effort but fortunately can be improved by a lot of different measures. Texture patching, e.g. Image Quilting [8] or Patch-Based Sampling [15], rearrange small texture blocks or arbitrary shaped texture patches of the sample texture to form a new visually similar texture. These methods spend most of the process time on optimising the transition between neighbouring patches. It involves calculating least cost paths through boundary error surfaces and post-processing, e.g. smoothing of the patch borders.

3. Motivation

Looking at the different algorithms for texture synthesis and comparing their impressive results for a collection of textures from *VisTex* [13] and Brodatz [3], we were interested in designing a texture synthesis algorithm which includes the most advantageous aspects of recent approaches to achieve a good balance between synthesis quality and time. We considered **constrained synthesis** as necessary - the algorithm should be able to extend or fill user-masked areas of the sample textures. The synthesised texture should also be tileable in horizontal *and* vertical direction. Texture structures should be well reproduced over a wide frequency range. The last three points require multi-scale operation using an image pyramid [2, 12, 18, 20]. We combine an **implicit MRF model**, which promises an especially good quality for a variety of sample textures [1, 20], with **texture patching**, which reduces the cost of extensive searching [8, 15]. Not single pixels but small blocks, are copied from the sample texture. Block transfer is especially suited to preserve the local structure of the texture. If we define a texture block as **tex**ture **el**ement, which of course is different from the general texel definition (pixel of a texture map), one can say that the application of the implicit MRF model is shifted from the pixel to the texel domain. Together with the multi-scale approach it ensures an equally good reproduction of local *and* global texture features. At last a resolution-dependent, **variable search strategy** is used to identify the optimum synthesis block.

4. Algorithm

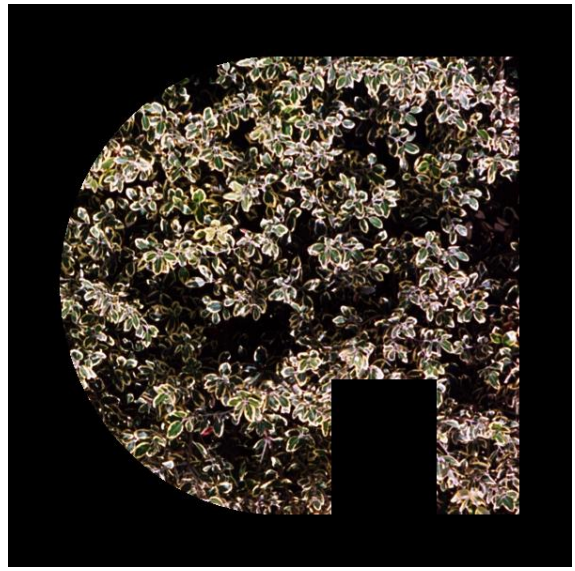

Figure 1: *Incomplete sample texture.*

First we want to give an overview on how the algorithm works on a single resolution level l. Figure 2 shows a part of the fourth level ($l = 3$) of an image pyramid I_l constructed from the texture sample I_0 seen in figure 1. For construction of Gaussian image pyramids the reader is referred to [4]. The black area of I_l is undetermined and is going to be filled during the texture synthesis. The image of the current level I_3 is divided into non-overlapping blocks B_i in scanline ordering. In practical application the user-defined, odd blocksize is $B^{size} \in [3,5,7,9,11]$, single pixel synthesis ($B^{size} = 1$) is also possible. A list $L\{B_u\}$ of all blocks with (partly) undetermined texture information is generated. The green blocks surrounded by the red grids in figure 2 exemplarily show four locations of undetermined texture information. The red grid defines the symmetric, respectively square shaped, pixel neighbourhood of the enclosed block. These neighbourhoods contain defined as well as undefined (black) pixels. The list $L\{B_u\}$ is sorted in ascending order according to the number of undefined pixels in the symmetric block neighbourhood $p^{undef}(N(B_u))$.

In the next step the reference information for the texture synthesis has to be identified. Only blocks B_r with neighbourhood $N^{ref}(B_r)$ that is completely *inside* the sample texture area are used. The position of these blocks is handled with pixel accuracy. In figure 2 the green blocks with

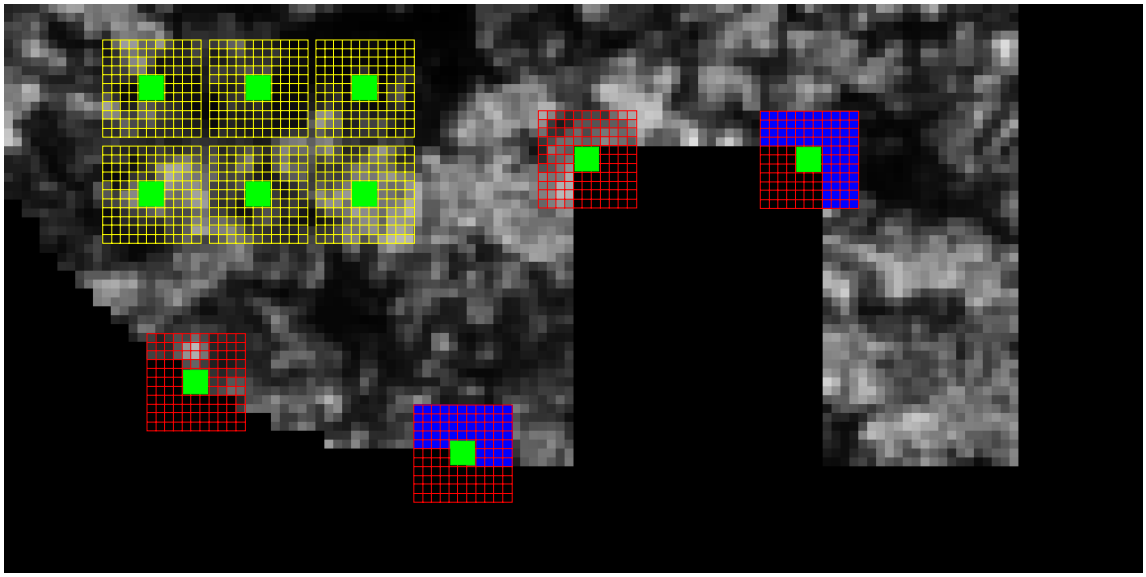

Figure 2: *Symmetric neighbourhoods. yellow/green: causal reference neighbourhoods with reference texture blocks, red/green: non-causal search neighbourhoods with (partly) undefined texture blocks, blue: variable search masks for ensuring causality of the texture synthesis.*

yellow neighbourhood N^{ref} depict such well-defined references. For the synthesis process each reference block B_r is represented by its symmetric reference neighbourhood N_r^{ref}. A N_r^{ref} can be expressed as a linear vector $\vec{N}_r^{ref}$. This means that all neighbourhood pixels are collected in scanline ordering (from left to right and top to bottom) and are stored in a contiguous memory block. All $\vec{N}_r^{ref}$ together form a neighbourhood vectorspace $V\{N^{ref}\}$, which is calculated once and stored in memory.

In the synthesis loop the neighbourhood N of the first block $B_{u=0}$ from the block list $L\{B_u\}$ is converted to a neighbourhood search vector $\vec{N}_u^{search}$. This vector is compared to all reference vectors in $V\{N^{ref}\}$ (for more details on distance metrics see section 7). The corresponding reference block of the vector $\vec{N}_r^{ref}$ that is closest to $\vec{N}_u^{search}$ is copied to the position of the block B_u. Only undefined pixels of the block B_u which are marked by the variable mask are transferred at the copying stage. The block B_u is removed from the list $L\{B_u\}$, the list position of all adjacent blocks to B_u is updated. The latter is necessary because the number of undefined pixel in the neighbourhoods of the adjacent blocks to B_u may have changed after the copy action. The loop continues until all blocks are removed from the list $L\{B_u\}$ and hence all undefined areas of I_l are filled.

Keeping the blocklist $L\{B_u\}$ sorted by $p^{undef}(N(B_u))$ ensures that those blocks which show the best-defined neighbourhood are synthesised first. In figure 2 the two blocks in the upper corners of the missing rectangle would be top of the sorted list. The synthesis "grows" the missing rectangle and afterwards progresses over the edgy border seen on the left.

For acceleration the list $L\{B_u\}$ can be transformed into a hash table with $p^{undef}(N(B_u))$ as hash key. Furthermore a spatial lookup table indexing the hash elements allows direct identification of adjacent blocks for fast list update.

5. Boundary handling

As one can see in figure 2 the comparison of a search vector $\vec{N}^{search}$ with a reference vector $\vec{N}^{ref}$ of $V\{N^{ref}\}$ must take into account that the the search vector is partially undefined. To ensure causality of the texture synthesis a masking operation on both vectors is performed before the actual comparison takes place. Example masks for two search neighbourhoods are highlighted blue in figure 2.

To fulfil the requirement of tileable synthesis results, the surface is treated toroidally, i.e. the image coordinates are always addressed with a modulo operation: $(x,y) = (x \bmod I_l^{width}, y \bmod I_l^{height})$. This, together with the image pyramid, enables tileability.

6. Weighted multi-scale synthesis

All MRF approaches have in common that they can only reproduce those structures correctly that match the size of the analysis window. This is also true for the neighbourhood searching described in 4. To synthesise global texture structures one has to enlarge the neighbourhood window to the

size of the largest texture feature which increases the computational cost dramatically. Another approach is to handle the different texture frequencies separately in an image pyramid and to refine the texture while progressing from low to high resolution levels.

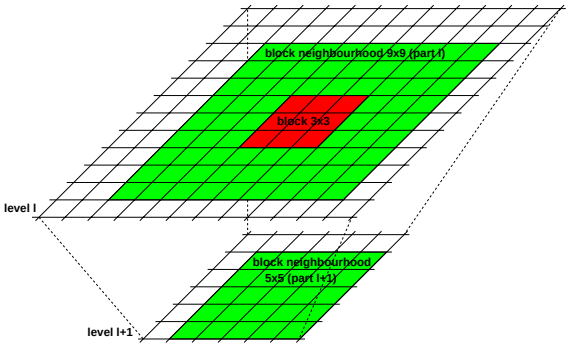

Figure 3: *Symmetric multiresolution neighbourhood.*

For our algorithm that means, that after completion of the pyramid level with the lowest resolution as described in 4 the next levels are processed in the same way, except that the symmetric neighbourhood is extended to take into account the results of the previous level to propagate and maintain the low frequency structures. Figure 3 shows a typical symmetric multiresolution neighbourhood. Both parts of the neighbourhood are always odd in size. The size of the low resolution part is adjusted to cover at least the texture area of the higher resolution level. During the comparison of $\vec{N}^{search}$ with all $\vec{N}^{ref}$ the two neighbourhood parts are treated separately. The weighting ratio between the two parts can be adjusted by the user. It's possible to put more emphasis on exact reproduction of global features which results in more artefacts in the local structure, i.e. sharp gradients at the border of the created texture patches. On the other hand reducing the weight of the low resolution part of the neighbourhood optimises the reproduction of local structures, especially the transition between neighbouring patches, with global coherence disappearing at the same time. Fortunately artefacts at the texture patch borders are compensated as the algorithm progresses through the pyramid levels.

The multi-scale approach is essential for single-pass constrained texture synthesis. On the lowest resolution level the neighbourhood window must be able to bridge the missing parts of the texture for achieving an optimum synthesis result. In other words, the size of the neighbourhood window in the lowest level determines the distance over which two different parts of the texture are equalised. Figure 4 shows the synthesis of three successive pyramid levels together with their copy map. The copy map codes the coordinates of the origin of the copied reference texture block and visualises the patching behaviour of the synthesis algorithm. The gradients left and bottom of the final pyramid level in figure 4 show the coordinate coding of the copy map. The red channel corresponds to the x coordinate, the green channel to the y coordinate.

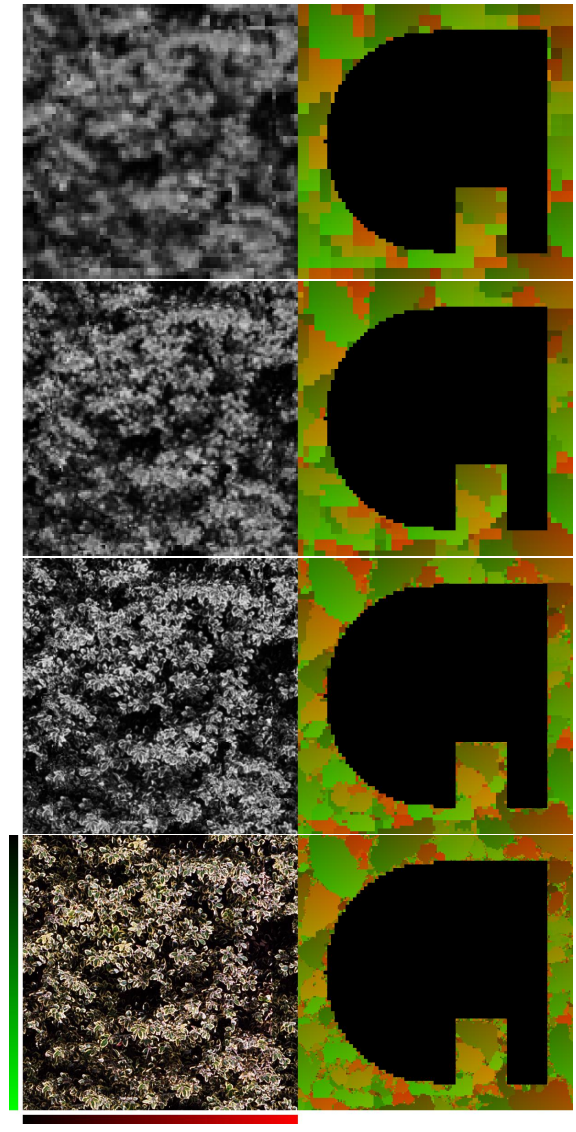

Figure 4: *Constraint multiresolution synthesis.*

7. Vector comparison

In section 4 the algorithm was outlined, except for details on search strategies and - most important - the comparison scheme for two neighbourhood vectors. In [20] the L_2 norm was used for measuring the distance between two vectors. As pointed out by [1] the L_2 norm is not a good measure for perceptual similarity. Metrics that are better adapted to the human visual system exist [14], but are computationally expensive.

Coming from video coding we decided to separate the luminance and chrominance information. Only the luminance information is processed in the core algorithm. Knowing that there is a strong correlation between structure and colour for certain classes of textures, we use a colour post processing in our first approach, see section 9. As synthesis speed is one of our primary objectives, we implemented the vector comparison as a sum of absolute differences of the vector components.

$$D(\vec{N}^{search}, \vec{N}^{ref}) = \sum_i \mid N_i^{search} - N_i^{ref} \mid \qquad (1)$$

This L_1 norm is also non-optimal for determination of perceptual similarity, but very common with multimedia applications and therefore available as hardware instruction in the MMX™ (MMX is a trademark of Intel Corporation) command set of x86-CPUs. Migration from L_2 norm calculation to L_1 norm with MMX™ increased the execution speed by a factor of 3. Only slight differences are visible in the synthesis results - L_2 norm gives a smoother transition between neighbouring patches, but also tends to blur the images a little more than L_1, see also [1].

8. Search strategy

Though computing the vector distance with MMX™-L_1 norm accelerates the algorithm, the exhaustive searching in the neighbourhood vector space $V\{N^{ref}\}$ is still very slow even on fast workstations. This is fatal if we think of huge output textures the size of several megabytes. To accelerate the search process [20] introduced tree structured vector quantisation (TSVQ) of the search space. Our experiments showed, that partitioning of the high-dimensional, thin populated search space with TSVQ only delivered good results with generous backtracking of the tree branches. The necessary degree of backtracking varied with the kind of texture that was processed. Also [1] pointed out, that TSVQ had a strong blurring influence on the synthesis. So we investigated other techniques and tried hypercube searching [16] and the LBG-algorithm [23].

Hypercube searching involves resorting, respectively hashing, of the vectorspace dimensions and is also very memory intensive. As the sorting has to happen only once at the beginning of the synthesis loop, followed by millions of search operations, we considered that sorting time can be neglected. Hypercube searching converges quickly if the hypercube size ε can be set to a small value and many vectors can be sorted out in a single iteration. We derived a value for ε from an exhaustive search result of our texture synthesis which was large (≈ 60) with respect to the absolute grey value range $[0, 255]$. Our final hypercube algorithm with dynamic ε adjustment increased texturing speed by a factor of 3. Unfortunately hypercube searching can't be combined with MMX™ and is a dead end for further acceleration. So we decided to keep the MMX™ advantage and concentrate on vector quantisation with the LBG algorithm.

With LBG [23] a number of seed vectors $\vec{N}_s^{seed}$ is placed in the vectorspace $V\{N^{ref}\}$ (step 1). In practice these vectors are chosen randomly from the existing vectors $\vec{N}_r^{ref}$. In step 2 each vector $\vec{N}_r^{ref}$ is assigned to the closest vector $\vec{N}_s^{seed}$. In step 3 each vector of $\vec{N}_s^{seed}$ is moved to the geometric centre of the group of assigned vectors. Steps 2 and 3 are repeated until the relative change of the sum of the absolute displacements of the seed vectors $\vec{N}_s^{seed}$ is below a given threshold, for example 5%. After convergence the seed vectors represent the centres of evenly distributed partitions of the vector space $V\{N^{ref}\}$. As with hypercube searching the preparation time for the partitioning is short compared to the actual search process. Within the synthesis process a search vector $\vec{N}^{search}$ is compared to all seed vectors $\vec{N}_s^{seed}$. The partition of the most similar seed vector is searched for the final reference vector $\vec{N}_r^{ref}$. The search speed is optimal if the number of seed vectors is chosen as square root of the number of vectors in $V\{N^{ref}\}$. For example instead of exhaustively searching 1000000 reference vectors, with LBG only 2 times 1000 vectors have to be compared. In addition LBG can be combined with MMX™-L1 norm calculation which makes a total increase of texturing speed by a factor 1500 for this example.

As with all quantisation schemes it is not guaranteed that LBG delivers the optimum vector. Compared to TSVQ the error rate was much lower at identical runtime and we achieved a better visual quality of the synthesis results. Even so, if we synthesise over a pyramid, errors in lower levels are propagated to higher resolutions. To avoid error accumulation we decided to use a mixed search strategy which can be adjusted to user preferences. Looking at the search process in general, we notice that the computational effort increases by a factor 16 from each pyramid level to the next. The reason for this is the quadratic enlargement of the reference texture (factor 4) multiplied by the quadratic enlargement of the surface to be textured (again factor 4). In practice that means that the lower pyramid layers can be computed very fast and exhaustive searching with the advantage of optimal search results should be favoured.

Therefore we created a multithreaded search procedure that is able to take advantage of symmetric multiprocessing architectures (SMP) by searching subpartitions of $V\{N^{ref}\}$ in parallel running threads. In the next step we enhanced this "brute force" approach by implementing standalone network search clients which can be controlled by the master synthesis process. The master checks for available resources and launches the clients remotely on suitable machines. The clients are calibrated with a dummy search procedure to get a figure for the local system performance. Based on these results the vector space $V\{N^{ref}\}$ is divided into partitions, which are distributed to the clients. At request a search vector is transmitted to the clients which respond with an in-

dex on the optimum vector in their associated subspace. The master checks the subspace results for the global optimum vector and executes the texture transfer. If a client fails or becomes busy the cluster is dynamically recalibrated and the vector space is repartitioned.

The user defines which search method (SMP, cluster, LBG) is used on which pyramid level. For most experiments we utilise SMP for all lowlevel layers and LBG for the pyramid layer with the highest resolution.

9. Colour handling

Figure 5: *Colour assignment error at texture patch borders. Left: full synthesis result gives a good overall impression, right: a detail view of the texture shows the colour artefacts.*

In section 7 we explained that the current version of our core synthesis algorithm relies only on the luminance information of the texture sample for acceleration reasons. Colour information may be introduced by two different approaches. First, the calculation of the sum of the absolute differences of two vectors can easily be extended to colour handling by including each of the RGB-channels separately into the sum. This would triple the memory usage of the vector space and reduce execution speed by a factor of three. That's why we chose to reconstruct the colour in the highest pyramid level. When synthesising the final resolution of the pyramid, we use the information of the copy map to rearrange the original colour input texture according to the texture patch layout seen in the copy map. This is done in parallel to the synthesis of the greyscale texture.

Running tests on the *VisTex* library we realized that most of the images shown there have a strong correlation between structure and colour. Our colour handling fails, if the same structures have different colours, see figure 5. At the borders of the connected texture patches colour artefacts are visible, which can be improved slightly by blurring the colour information.

10. Results

10.1. Full texture synthesis

Our algorithm described in the previous sections was designed for single-pass constrained synthesis, which will be discussed in 10.2. For comparison of our method with previous techniques, we had to alter the algorithm to simple scanline operation with a fixed variable mask that changed the neighbourhood to L-shape, see [20]. Also the synthesis surface had to be initialised with random noise drawn from the texture sample, to give the synthesis some information to start with. Results for a full synthesis along with textures created by [1] and [20] are shown in figure 6.

The results from Wei and Levoy where taken from their webpage (http://www.graphics.stanford.edu/projects/texture/demo/), the textures generated by Ashikhmin are taken from [1]. Our results were processed using a three level pyramid, a 9*x*9 neighbourhood, 3*x*3 or 5*x*5 blocks and a mix of exhaustive and LBG-searching. The global and local structures are well reproduced due to the multi-scale synthesis. As the input sample is very limited in size and information, the global variation of our results is low. Careful consideration shows that especially for the grass and knots texture only few global features where selected by the synthesis process. Although the results from Wei and Levoy were produced with a similar method (exhaustive searching over an image pyramid, but copying pixels) they contain a lot of blur and artefacts in the global structure. The reason for this is not totally clear to us, but could be an implementation issue. The local structure of Ashikhmin's results is the same quality as ours. The global coherence on the other hand is limited to the capability of the applied texture prefetch model which assumes that most of the transferred texture patches are adjacent to each other. Certain textures with more complicated global structure like the knots need a true multi-scale analysis for optimal reproduction. Figure 7 shows more examples for full synthesis with our algorithm.

Algorithm	Training [*s*]	Synthesis [*s*]
a) Wei, Levoy (exhaustive)	-	503
b) Wei, Levoy (TSVQ)	12	12
c) Ashikhmin	-	0.2
d) Stahlhut 1 (exhaustive)	-	72
e) Stahlhut 1 (exh./LBG)	1.8	11
f) Stahlhut 3 (exhaustive)	-	6.5
g) Stahlhut 3 (exh./LBG)	1.6	1.2

Table 1: *Running times for synthesising a* 192*x*192 *texture from a* 64*x*64 *sample. a)b)d)e)f)g) use a* 9*x*9 *neighbourhood, c)* 5*x*5 *neighbourhood, a)b)c)d)e) pixelwise synthesis, f)g)* 3*x*3*-blockwise synthesis, times for d)e)f)g) have been multiplied by 6.6 for direct comparison*

Due to the MMX™ utilisation our algorithm is bound to x86-architecture (Linux PC platform) and relating the texturing rate of our approach to former methods running on MIPS R10000 CPUs needs some estimates. Benchmarking integer, memory and floating point performances of a 2*GHz* Pentium 4 and a 195*MHz* MIPS R10000 gives the ratios

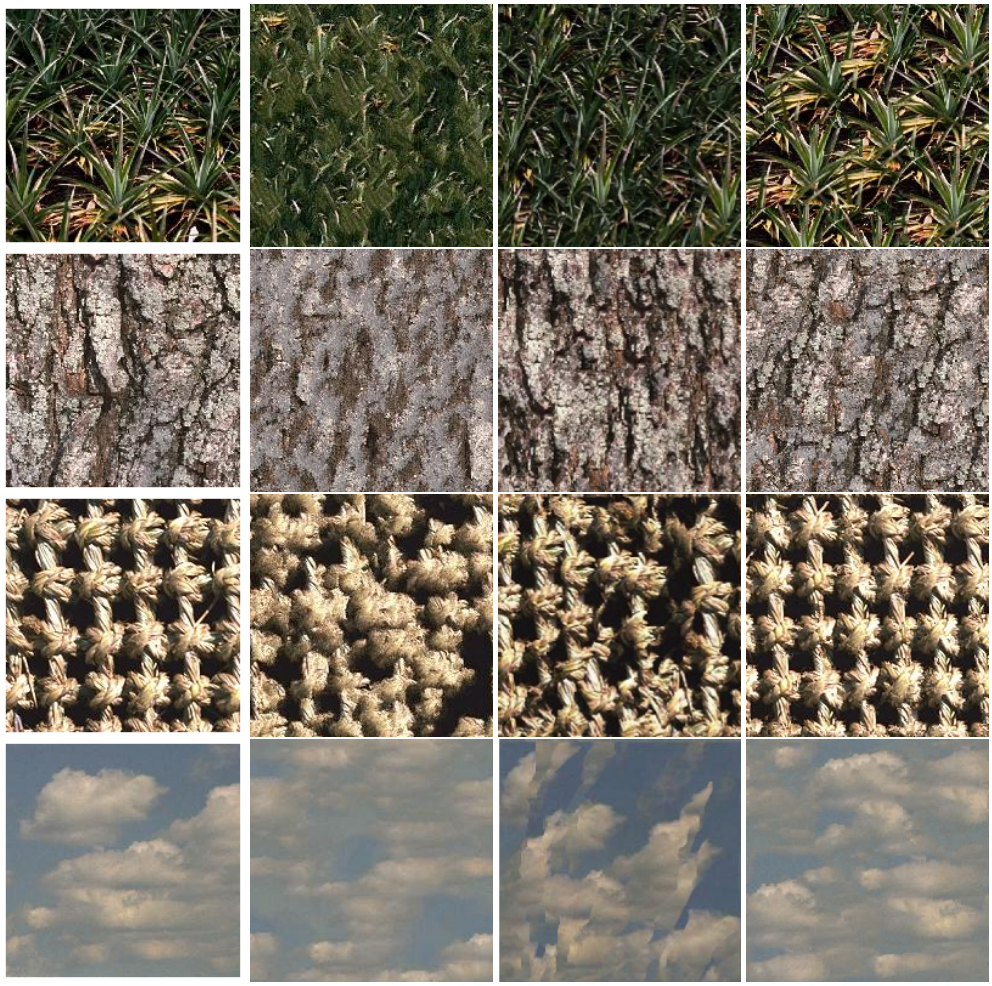

Figure 6: *Full texture synthesis results. From left to right: input texture sample* 192*x*192 *from VisTex* [13] *library, Wei Levoy results, Ashikhmin results, our results. All results are* 200*x*200.

$(6.9, 6.1, 4.0)$. As far as we know floating point operations aren't used in any of the algorithms, so we suggest an inter-architecture conversion factor of 6.6. Table 1 shows figures for full texture synthesis.

LBG preparation is faster than of TSVQ. The synthesis times of e) and g) in table 1 result from a combination of two pyramid layers exhaustive search and LBG for the top level. The difference between a) and d) is due to MMX™ operation and colour post-processing of our algorithm. Compared to Wei and Levoy we gain from implementation issues, compared to Ashikhmin who employs a texture prefetch model we are still half a magnitude slower for full synthesis.

10.2. Single-pass constrained synthesis

As our algorithm was developed for single-pass constrained synthesis, the advantages become fully visible when synthesising with blocks instead of pixels in combination with the image pyramid. In typical applications, such as texturing of landscape scenes, we use much larger reference textures (size 500*x*500 to 2000*x*2000 and above) which give a good overview about the global features. For example, synthesis of a 1000*x*1000 texture using a 500*x*500 texture as input with a blocksize of 5*x*5 and a 11*x*11 neighbourhood needs 154 seconds. Setting this into relation of 192*x*192 from 64*x*64 would result in a runtime of 0.09 seconds, which is close to the measurements of [1].

Figure 8 shows some results produced with our single-pass constrained synthesis. Texture reference for the synthesis process is always the input image itself, no further texture information was used to obtain the missing parts.

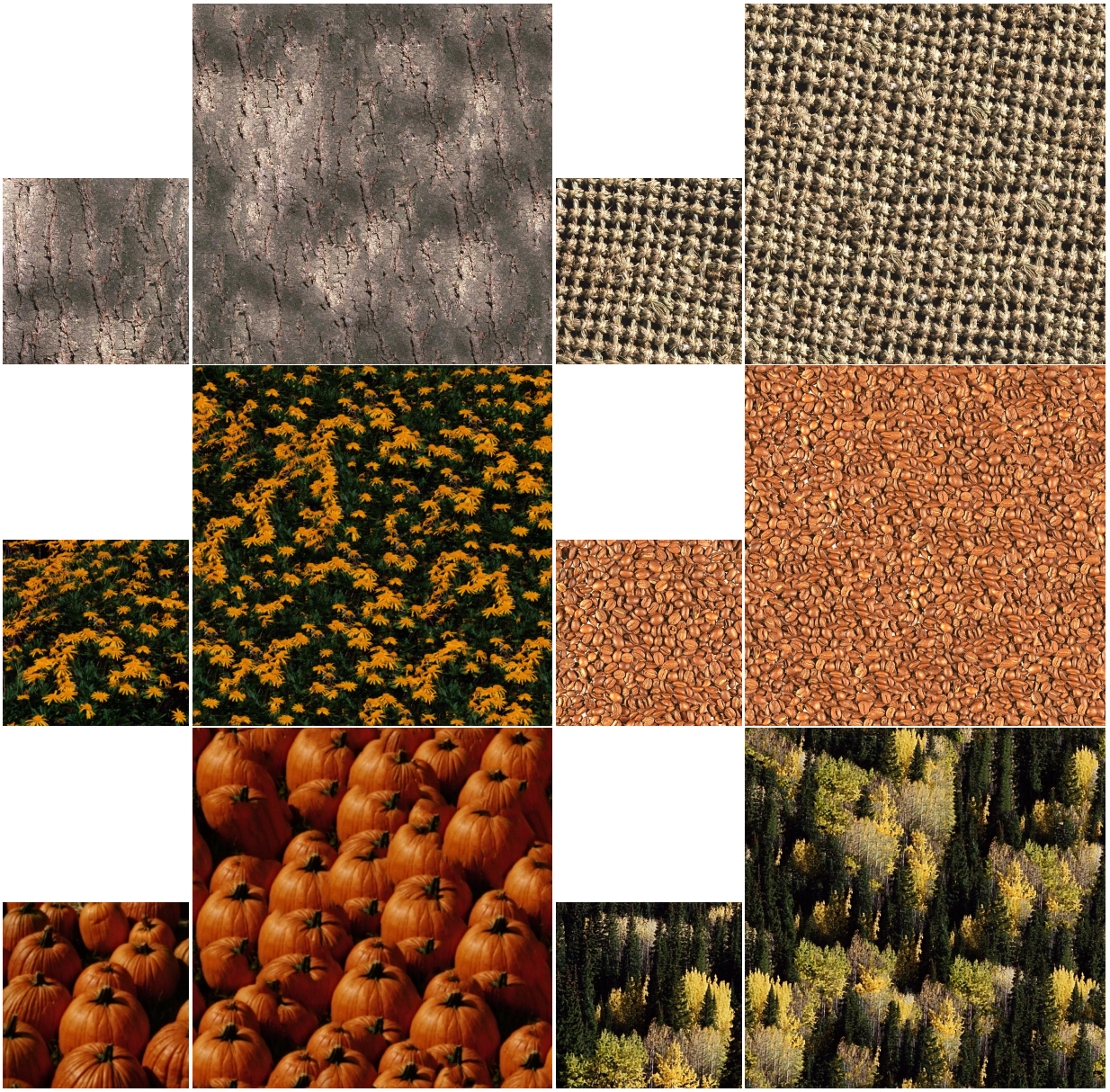

Figure 7: *Full texture synthesis. Left: input texture sample* 512*x*512 *from VisTex* [13] *library, right: synthesis results* 1024*x*1024.

10.3. Texture mixing

The pyramid offers another interesting aspect of synthesis: mixing of global and local features. We down-sample a large piece of texture and use it as input for texturing a low resolution pyramid layer. Doing so, the global structure is well preserved. In a next step representative parts of the local structure of the original texture are put together to a high resolution collage. This collage is used to refine the global information while progressing through the pyramid, see figure 9.

The idea behind this approach is, that the local texture structure is often redundant and processing of large input textures with previous techniques involved handling all this redundant information at the same time, which reduces the texturing speed significantly. By optimising the input for the different pyramid layers, the texturing rate can be increased by magnitudes without loosing visible quality.

Texture mixing also allows interactive user control by modifying the global structure in a lower pyramid layer before the high-frequency information is synthesised. The text

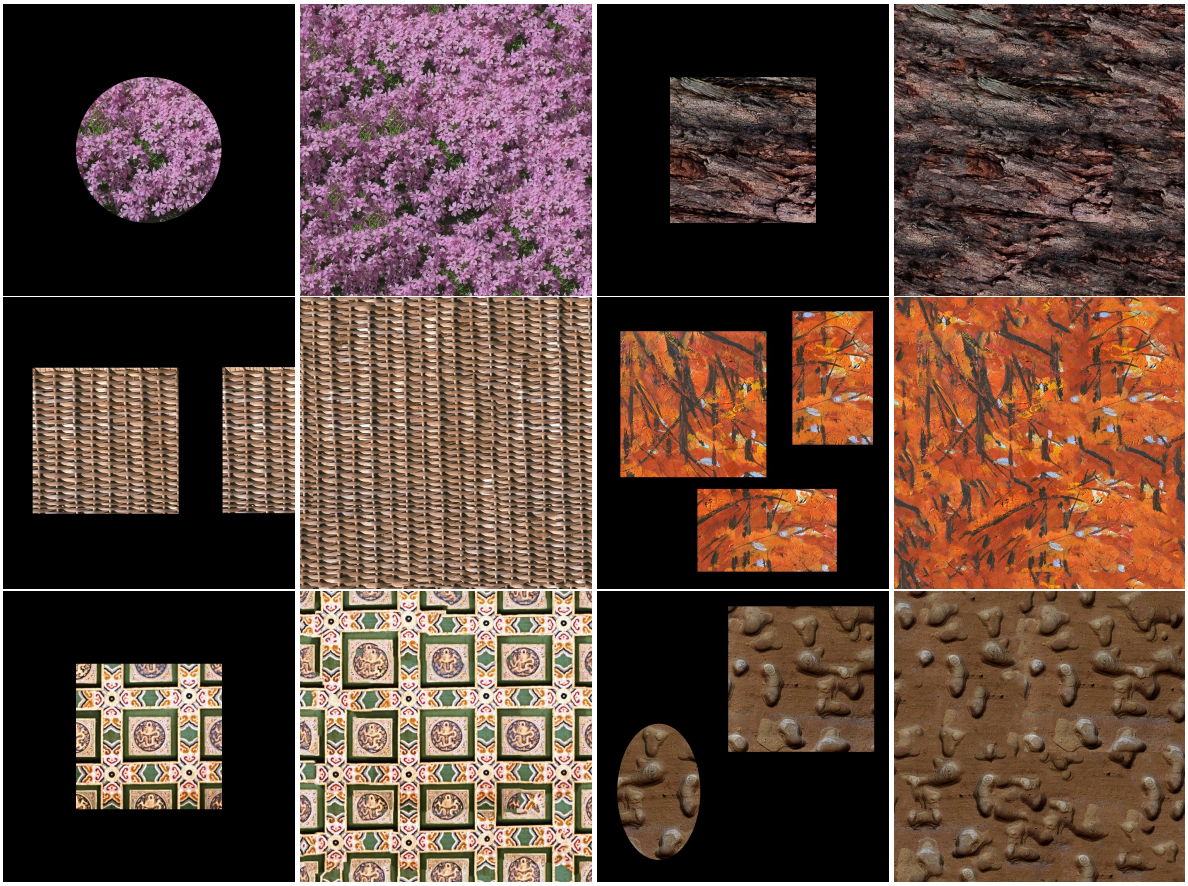

Figure 8: *Constrained texture synthesis. Left: incomplete input texture, right: synthesis results. All images from VisTex library* [13].

in figure 9 was created that way and integrates smoothly into the knot structure.

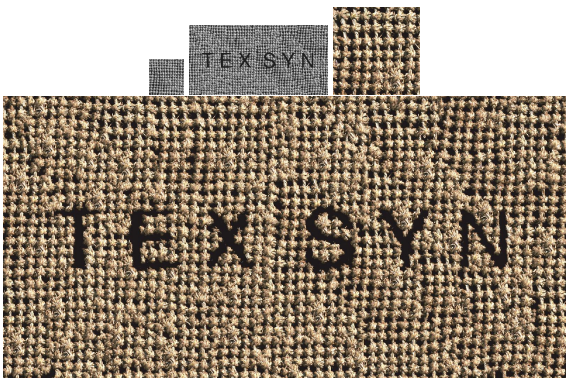

Figure 9: *Texture mixing. Top row: input and output texture of global structures, local input. Bottom row: output texture.*

11. Conclusion and Outlook

In this paper we presented an algorithm for constraint texture synthesis which uses an image pyramid for correct reproduction of local and global features of an input texture. The synthesis rearranges blocks of the input texture according to neighbourhood constraints to fill the output texture. The block selection is achieved by continuous comparison of the block neighbourhoods of output and reference texture. It involves heavy searching, which was accelerated by application of the LBG-algorithm. The algorithm works especially well for textures which show repeating, similiar elements on the local scale with an arbitrary global arrangement. Texture quality and speed have been improved compared to previous approaches.

Currently we are investigating new methods for colour handling by colour compression in conjuction with fast implementations of metrics for improved perceptual similarity. The algorithm is going to be extendend to work on a three-dimensional surface description.

References

1. Michael Ashikhmin. Synthesizing natural textures. In *Proceedings of the 2001 symposium on Interactive 3D graphics*, pages 217–226. ACM Press, 2001.

2. Jeremy S. De Bonet. Multiresolution sampling procedure for analysis and synthesis of texture images. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 361–368. ACM Press/Addison-Wesley Publishing Co., 1997.

3. P. Brodatz. *Textures: A Photographic Album for Artists and Designers*. Dover Publications, New York, 1966.

4. Peter J. Burt and Edward H. Adelson. The laplacian pyramid as a compact image code. *IEEE Transactions on Communications*, COM-31,4:532–540, 1983.

5. Julie Dorsey, Alan Edelman, Justin Legakis, Henrik Wann Jensen, and Hans Kohling Pedersen. Modeling and rendering of weathered stone. In *Siggraph 1999, Computer Graphics Proceedings*, pages 225–234, Los Angeles, 1999. Addison Wesley Longman.

6. Julie Dorsey and Pat Hanrahan. Modeling and rendering of metallic patinas. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 387–396. ACM Press, 1996.

7. David S. Ebert, F. Kenton Musgrave, Darwyn Peachey, Ken Perlin, and Steven Worley. *Texturing and modeling: a procedural approach*. Academic Press Professional, Inc., 1994.

8. Alexei A. Efros and William T. Freeman. Image quilting for texture synthesis and transfer. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 341–346. ACM Press, 2001.

9. G. L. Gimel'farb. Texture modeling by multiple pairwise pixel interactions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(11):1110–1114, 1996.

10. Georgy L. Gimel'farb. *Image Textures and Gibbs Random Fields*. Kluwer Academic Publishers, 1999.

11. John C. Hart, Nate Carr, Masaki Kameya, Stephen A. Tibbitts, and Terrance J. Coleman. Antialiased parameterized solid texturing simplified for consumer-level hardware implementation. In *Proceedings of the 1999 Eurographics/SIGGRAPH workshop on Graphics hardware*, pages 45–53. ACM Press, 1999.

12. David J. Heeger and James R. Bergen. Pyramid-based texture analysis/synthesis. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 229–238. ACM Press, 1995.

13. MIT Media LAB. *Vision texture library*. http://www-white.media.mit.edu/vismod/imagery/VisionTexture., 1995.

14. C. Lambrecht and J. Farrell. Perceptual quality metric for digitally coded color images. In *Proceedings of the European Signal Processing Conference*, pages 1175–1178, 1996.

15. Lin Liang, Ce Liu, Ying-Qing Xu, Baining Guo, and Heung-Yeung Shum. Real-time texture synthesis by patch-based sampling. *ACM Transactions on Graphics (TOG)*, 20(3):127–150, 2001.

16. Sameer A. Nene and Shree K. Nayar. A simple algorithm for nearest neighbor search in high dimensions. *IEEETPAMI: IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19, 1997.

17. Ken Perlin. An image synthesizer. In *Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, pages 287–296. ACM Press, 1985.

18. Javier Portilla and Eero P. Simoncelli. A parametric texture model based on joint statistics of complex wavelet coefficients. *International Journal of Computer Vision*, 40(1):49–70, 2000.

19. Greg Turk. Generating textures on arbitrary surfaces using reaction-diffusion. In *Proceedings of the 18th annual conference on Computer graphics and interactive techniques*, pages 289–298. ACM Press, 1991.

20. Li-Yi Wei and Marc Levoy. Fast texture synthesis using tree-structured vector quantization. In *Siggraph 2000, Computer Graphics Proceedings*, pages 479–488. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000.

21. Andrew Witkin and Michael Kass. Reaction-diffusion textures. In *Proceedings of the 18th annual conference on Computer graphics and interactive techniques*, pages 299–308. ACM Press, 1991.

22. Steven Worley. A cellular texture basis function. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 291–294. ACM Press, 1996.

23. A. Buzo Y. Linde and R.M. Gray. An algorithm for vector quantizer design. *IEEE Transactions on Communications*, COM-28:84–95, 1980.

24. Song Chun Zhu, Yingnian Wu, and David Mumford. Filters, random fields and maximum entropy (frame): Towards a unified theory for texture modeling. *International Journal of Computer Vision*, 27(2):107–126, 1998.