GPU-based Multi-Resolution Image Analysis for Synthesis of Tileable Textures

Gottfried Eibner[1], Anton Fuhrmann[1], and Werner Purgathofer[2]

[1] VRVis Forschungs GmbH
[2] Vienna University of Technology

Abstract. We propose a GPU-based algorithm for texture analysis and synthesis of nearly-regular patterns, in our case scanned textiles or similar manufactured surfaces. The method takes advantage of the highly parallel execution on the GPU to generate correlation maps from captured template images. In an analysis step a lattice encoding the periodicity of the texture is computed. This lattice is used to synthesize the smallest texture tile describing the underlying pattern. Compared to other approaches, our method analyzes and synthesizes a valid lattice model without any user interaction. It is robust against small distortions and fast compared to other, more general approaches.

1 Introduction and Related Work

Many manufactured surfaces contain regular patterns, which could efficiently be used as repetitive textures in a computer graphics. Capturing these patterns by camera or similar means leads in most cases - due to perspective or lens distortion, or due to the geometry of the underlying surface - to images which do not tile correctly. Converting these images into tileable texture maps that correctly represent the input pattern is the goal of this paper. We discuss the problem of finding pattern similarities, extracting a lattice model for repeating patterns and of generating tileable textures from such images. This allows us to render highly detailed surfaces where a base pattern repeats itself at a fixed period. The analysis and synthesis of stochastic or non-regular textures is out of scope of this paper. Input images to our algorithm should contain periodic patterns with at least one full period visible in both dimensions and deformations should be kept as small as possible. Even if those restrictions are not met, the algorithm is in many cases still able to create tileable patterns.

In the field of computer vision Lindeberg [1] introduced the concept of scale space. Lowe [2][3] introduced the idea of image descriptors which led to his SIFT implementation. His work has led to a variety of extensions and new methods to quickly find correspondences between different images even under strong perspective distortions, like ASIFT [4], MSER [5], SURF [6], to name a few of them.

In the area of symmetry detection many researchers like Kiryati and Gofman [7] made some progress, but their methods were very slow. Scognamillo et al. [8] take a different approach, but theirs needs a lot of user interaction, especially in preparing the input image for their detection algorithm. Loy and Eklundh [9] used the concept of scale-invariant features to quickly detect symmetry in a single picture.

Lin et al. [10] use autocorrelation functions, while Hays et al. [11] show that the analysis can be seen as a higher-order correspondence problem and make use of thin plate splines to compensate distortion and synthesize the texture.

Lin and Liu [12] showed an approach to track near-regular textures in motion. Park et al. [13] take advantage of mean-shift belief propagation to efficiently derive a lattice model. Dischler and Zara [14] uses mesh reconstruction to analyze the structure of a pattern.

The work from Liu et al. [15] picks up the essential result from another research area that only *five* possible lattices for 2D textures exist—a fact we can take advantage of for any texture analysis and synthesis. For our approach we will show that under certain circumstances these five possible lattices can be broken down to a single class of lattice, namely the rectangle lattice, which is a prerequisite for most rendering applications. As we will show, this can be done for almost all textures we encountered in our research.

A more detailed view of group theory and lattices can be found in another work of Liu et al. [16].

2 Proposed Method

The proposed algorithm is divided into three stages:

1. *Similarity Analysis*, in which positions of similar regions in the input image are calculated
2. *Lattice Generation*, in which the underlying lattice of the periodic texture is extracted, and
3. *Texture synthesis*, in which the results from the previous steps are used to synthesize a tiling pattern.

2.1 Similarity Analysis

Our algorithm takes a single image as input containing any repeating structure. The algorithm is based on a simple template matching algorithm. It randomly chooses a template which is matched against all pixels of the image. For each pixel a discrete convolution between the template and a corresponding region around the pixel is computed (see equation (1)). This simple approach can be compared to an exhaustive search which would definitely take to much time on a single CPU, but since we take advantage of the parallel execution on the GPU the computation is extremely fast. The per pixel calculation is performed within a fragment shader and each pixel in the new generated image represents the convolution, the sum of squared errors between the given sample region and the same sized region around each pixel. In the fragment shader for a gray scale image the following equation is computed where $i(x,y)$ denotes the intensity of the input image at pixel location x,y; and where x_0,y_0 is the center and w,h is half width and height of the template.

$$f(x,y) = \sum_{\xi=-w}^{w} \sum_{\eta=-h}^{h} \left(i(x+\xi,y+\eta) - i(x_0+\xi,y_0+\eta)\right)^2 . \tag{1}$$

By using a color image instead of a gray scale image we obtain better results in the matching algorithm. We use a heuristic metric which is 0 if the colors are visually different, and 1 if the colors are the same. We tried different functions in HSV and RGB space, but generally RGB performed adequately and resulted in faster code on the GPU. The metric is computed with equation (2), where $\boldsymbol{c}_i$ denotes the color's RGB vector, while c_i denotes the color's intensity value.

$$a(\boldsymbol{c}_0,\boldsymbol{c}_1) = 1 - \frac{\boldsymbol{c}_0}{\|\boldsymbol{c}_0\|} \cdot \frac{\boldsymbol{c}_1}{\|\boldsymbol{c}_1\|} * |c_0 - c_1| \,. \tag{2}$$

Using equation (1) and extending it with (2) we get equation (3) where $\boldsymbol{i}(x,y)$ denotes the RGB vector of the input image at the given position.

$$\tilde{f}(x,y) = \sum_{\xi=-w}^{w} \sum_{\eta=-h}^{h} \big(\boldsymbol{i}(x+\xi,y+\eta) - \boldsymbol{i}(x_0+\xi,y_0+\eta)\big)^2 * a\big(\boldsymbol{i}(x+\xi,y+\eta),\boldsymbol{i}(x_0+\xi,y_0+\eta)\big) \,. \tag{3}$$

Putting equation (3) into equation (4) we obtain a correlation value, where zero means not correlated to the template and one means a full match with the template.

$$p(x,y) = (1 - \alpha * \tilde{f}(x,y))^{\gamma} \;\; , \; \alpha = \frac{1}{w*h} \; , \; \gamma \geq 1 \,. \tag{4}$$

An exponent greater than one will enhance the response of matching regions and dampen the 'noise'. Choosing a high number for the exponent, will lead to a correlation map with high peaks. A higher exponent makes it easier in the post-process to extract the feature points since regions with high correlations are exaggerated. On the other hand it should be mentioned that by selecting a value that is too large some feature points could be missed.

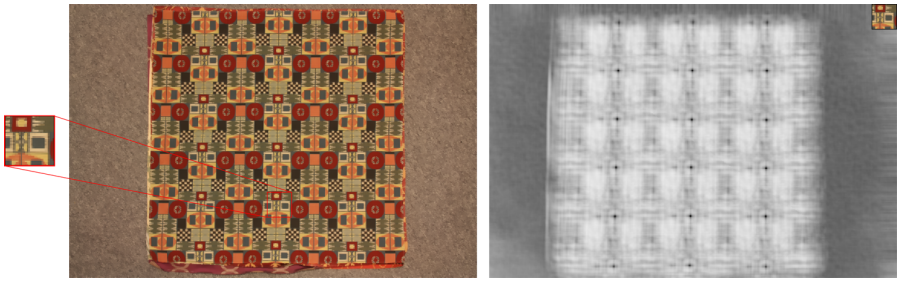


Fig. 1. A randomly chosen region (left) is used as comparison operator on the input image (middle) which leads to the correlation map (right) using equation (4) in a fragment shader.

If we perform a naive brute-force search, the computation time grows extremely fast if a bigger sample region is selected and a higher image resolution is chosen. We use a multi-resolution approach (see below) to avoid high computational costs.

As a post-process the correlation map is filtered. In a 6x6 pixel neighborhood only the highest correlation value is kept while all other pixels are set to zero to eliminate nearby matching 'echos' which are in the range of 2-3 pixels. After the filtering, all pixel values greater zero are stored with their correlation value and their image position as potential *feature points*.

2.2 Lattice Generation

All potential feature points from the correlation map are stored with their position and their correlation value. The correlation value represents the matching between the template and the region around the feature point (see figure 1).These correlation values are later used in the lattice model finding process.

The post-processing of the correlation map yields a list S_0 of feature points f_i. Each feature point stores the correlation value of the template and the region around the feature point's location. From these feature points we derive and verify a lattice model. As one can see in figure 2, the feature point candidates contain many false positives. To derive the correct lattice model, we examine various subsets of the feature points and use them to search for a valid lattice model. Each consecutive subset is generated by further raising the threshold level for the correlation values.

$$S_{t_{corr}} = \{f_i \in S_0 \mid value_{corr}(f_i) \geq t_{corr}\} \,. \tag{5}$$

In figure 2 four subsets are shown with a threshold of 0.0, 0.125, 0.25, and 0.5. By raising the correlation threshold, candidate feature points vanish until only those feature points remain which span the correct lattice of the texture pattern.

Lets assume we have N feature points extracted from the correlation map and stored in a list. While varying the correlation threshold a new subset from these points is generated and thus another possible lattice model can be found(see figure 2). We use the following method to find the lattice model:

- generate the subset for a given correlation threshold;
- for each feature point in the subset compute two displacement vectors from the two nearest neighbors; these two vectors must be linearly independent;
- cluster all displacement vector pairs with respect to their length and direction so that each cluster contains only vector pairs that go nearly in the same direction;
- take the largest cluster - the one containing the most items - as a starting base.

In this approach we assume that each feature point together with its two closest neighbors builds the correct lattice. To verify this assumption we have to check that

- the lattice model is connected,
- no outliers are taken into account, i.e. regions with a high correlation value but with a weak similarity to the template.

If neither a dominant cluster can be found, i.e. no cluster is found at all, or all clusters have fairly the same amount of vector pairs, nor one of the latter two criteria are met, we raise the correlation threshold until a valid lattice model is found. In figure 2 the lattice finding process is sketched. More and more feature points are removed until the

correct lattice model is found. In the first two pictures an unconnected lattice model is generated and thus rejected. In the third picture a lattice model was found which contains some outliers although all vector pairs fall into the same cluster due to their similarity in length and direction. The correct model is not found until the last picture, where all of the false-positive feature points have been removed.

The more feature points we use, the more possibilities exist to find a correct lattice, i.e. we could always use only the three most significant feature points (in respect to their correlation value) and extract at least one single displacement vector pair, but this leads to a suboptimal lattice and to one texture tile only. Therefore, we should make use of as many feature points as possible to verify the overall correctness of the lattice, compensate distortions, and remove outliers. This will be discussed in detail in the section 2.3.

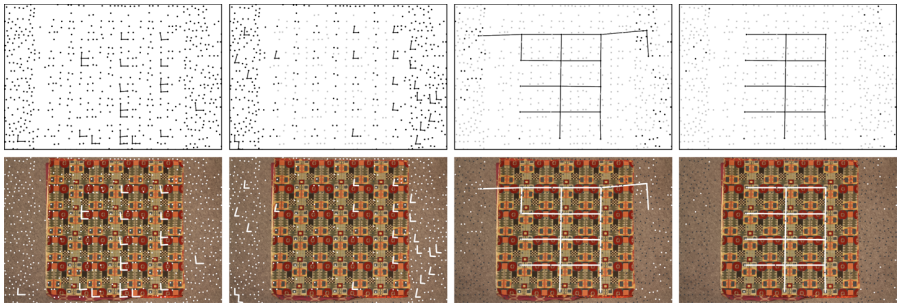


Fig. 2. Extracting a lattice model from left to right: increasing the correlation threshold removes unwanted feature points and leads to a valid lattice model. In the two left-most images a non-valid lattice model is found since it is not connected. In the third image two outliers lie within the lattice and the model is rejected (see top row for outliers). In the rightmost image the correct lattice is found.

Multi-Resolution Optimization. We have shown the details of our texture analysis approach in the previous sections. Now we want to improve the execution speed of the method, especially for high resolution images and high resolution sample regions. We use a pyramidal approach by constraining the search space for higher resolution using the results from lower ones. We start with the full resolution input image and successively sample it down to build a level-of-detail (LoD) pyramid. Each individual image of the pyramid is enhanced using a bilateral filter to reduce noise and attenuate color regions. The pyramid is constructed from the highest resolution available to a minimum resolution of at least 256 pixel width or height which ever is reached first.

For a full automatic extraction we decided to use a genetic algorithm approach since our search-space has several dimensions: the correlation threshold (1 dimension), sample size (1 dimension) and location (2 dimensions), and level of detail (1 dimension). So we end up in a 5 dimensional search space. The algorithm starts with the lowest resolution of the LoD-pyramid and a small template region of 16 by 16 pixels. The

template region's location is randomly chosen and altered until a valid lattice model is found or a specified amount of attempts is reached. If the maximum amount of attempts is exceeded and no lattice model is found we double the sample region and start again to find a valid lattice model. This is done until a valid lattice model is found or the size of the sample region covers more than a fourth of the LoD image. If we do not succeed in finding a lattice model by then we go to the next higher resolution in the LoD-pyramid and repeat the whole process starting again with a 16 by 16 pixel sample region.

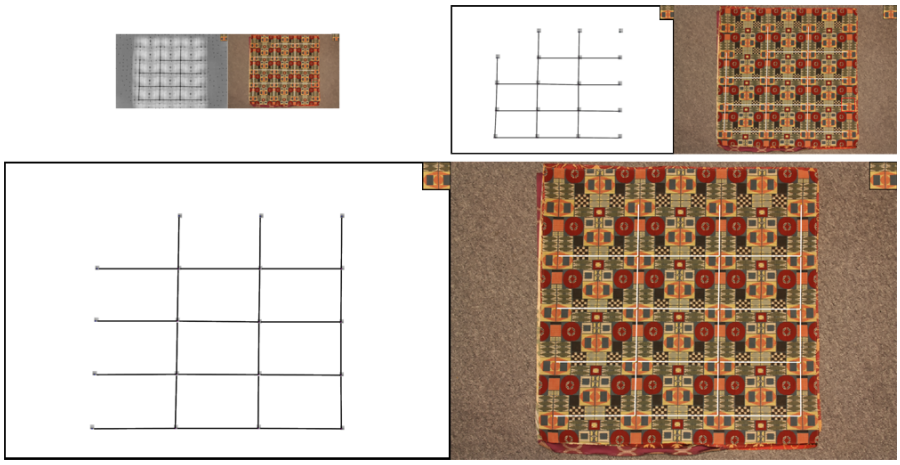


Fig. 3. Constraining the search space through multiple resolutions improves the performance of the algorithm and reduces the number of false-positive feature points at higher resolutions. From top to bottom, left to right. In the first image and lowest resolution all pixels are evaluated and a rough lattice is found. In the next steps only a small area around the feature points is evaluated and all other pixels remain blank.

If at any LoD level a valid lattice is found, we continue with the next higher resolution as follows: With the feature points from the lattice model we generate a stencil map. The fragment shader in combination with the stencil map computes the correlation values given by equation (4) only nearby those feature points. For all other pixels the fragment shader simply returns a value of zero. This saves time at higher resolutions and also reduces the chance of false-positive feature point extraction at higher resolutions since the maximum filter kernel is always kept at a size of 6 by 6 pixel.

Thus we narrow the search-space for higher resolutions and gain a speed-up of more than two orders of magnitude compared to an brute-force matching (see table 1).

The stenciled correlation map generation process is shown in figure 3. The top left image depicts the exhaustive search in the lowest resolution resulting where all pixels are evaluated. This gives a rough estimate of the lattice. The feature points' positions are used in the next higher resolution to constrain the search (top right image). The search is only performed in a small area around each feature point whereas in all other regions no matching is performed leading to blank pixels in the correlation map. As

we can see in this example, the direction of the displacement vectors may change from level to level, but this has no influence on the position of feature points or the resulting lattice and texture tiles.

Table 1. Time comparison table for the brute-force and multi-resolution search. We start with a brute-force search at the lowest resolution.

		Time [ms]	
Image res.	Sample size	multi-resolution	brute-force
267 x 178	16 x 16	10 *brute-force*	
534 x 356	32 x 32	3	680
1068 x 712	64 x 64	10	750
2135 x 1424	128 x 128	30	3000

2.3 Texture Synthesis

As soon as we have a valid lattice model and the displacement vectors, we can generate a texture tile. First we have to find two perpendicular displacement vectors. At this stage we know that the vectors are linearly independent (see section 2.2), but to extract a rectangular texture we need them to be perpendicular. We have to find the smallest possible rectangle fitting into the displacement vector model. In figure 4 a lattice with non-perpendicular displacement vectors v_0, v_1 is shown, and three highlighted feature points f_{i_0}, f_{i_1}, and f_{i_2}. Obviously the new displacement vectors should go from f_{i_0} to f_{i_1} and from f_{i_0} to f_{i_2}. Without the loss of generality we take v_0 as fixed and search a vector perpendicular to it. If we look at figure 4 again, one can see that

$$v_1^{'} = \lambda * v_1 + v_0. \tag{6}$$

$v_1^{'}$ is the new second displacement vector we are looking for and since the dot product of perpendicular vectors is always zero, we can write

$$v_1^{'} \cdot v_0 = 0 \quad \Rightarrow \quad (\lambda * v_1 + v_0) \cdot v_0 = 0. \tag{7}$$

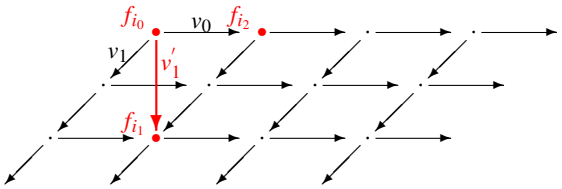
By rewriting equation (7) we get

$$\exists \lambda \in \mathbb{Z} \backslash \{0\} \quad , \quad \lambda = -\frac{v_0 \cdot v_0}{v_1 \cdot v_0}. \tag{8}$$

With equation (8) a new displacement vector pair and its new lattice are computed. A valid rectangular lattice can only be derived if λ is an integer. Each texture tile is extracted from the original image by taking the rectangle formed by each feature point and the two displacement vectors. If we have enough complete tiles available, we get a stack of valid textures which still may suffer from distortions due to bend and shear of the textile. Figure 5 shows a complete lattice model and the extracted texture tile stack.

In the figure the total error of the mean tile and the error of each single tile are shown. At this stage the error in the mean tile is high. To reduce the error we undistort each tile by simply morphing each tile in respect to a randomly choosen one and apply a median filter between all tiles to erase dirt and other artifacts (see figure 6 bottom image where the shadow on the textile is totally removed in the output).

Fig. 4. If the displacement vectors are not perpendicular we have to find a new vector basis that is perpendicular.



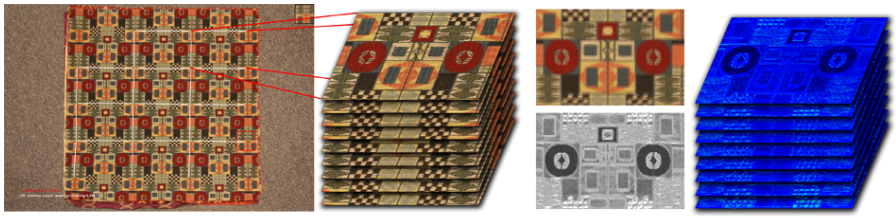


Fig. 5. A complete valid lattice (*left*) and the extracted texture tile stack on the (*left middle*). The mean tile and the summed error of all tiles (*right middle*) and error tiles for each single tile (*right*). Error coding: black no error, gray to white increasing error.

3 Results

The overall performance of the algorithm compared to previous approaches is of an order of magnitude faster. The approach of Hays et al. [11] takes some minutes up to half an hour to compute optimal lattice. Our approach takes less than a minute with comparable results.

The main goal of our algorithm was to find a lattice for repeating patterns on woven materials, but it may well be applied to a wide variety of other materials with an inherent priodicity. Its core algorithms are very simple and easy to implement on the GPU. Comparable algorithms can - due to their use of heavy math and their iterative nature - not as easily and efficiently be implemented on the GPU.

figure 6 shows some results for typical input photos.

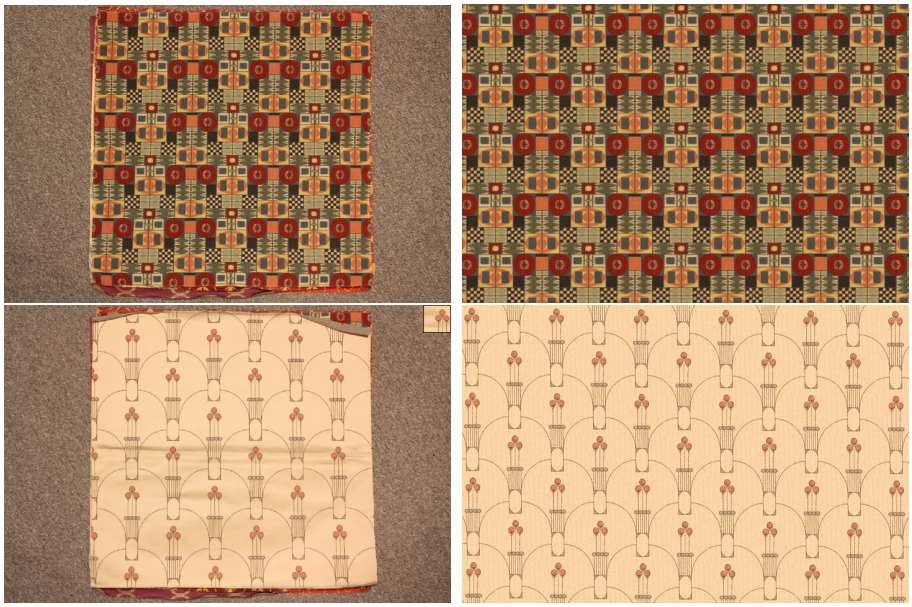


Fig. 6. Real photographs with a repeating pattern (*left*) and the extracted seamless repeating texture tiles (*right*).

4 Conclusion

We have shown a very simple and fast approach to analyze and synthesize textures from images of nearly regular periodic patterns. The overall performance is of an order of magnitude faster than previous approaches with comparable results.

The multi-resolution search step gives a high performance gain since at lower resolutions estimates about the lattice model are made and matching at higher resolutions will be performed only in regions around valid feature points from the found lattice.

Although the idea of template matching is very sophisticated in concept, it is so simple and easy to implement concurrently that it makes a perfect candidate for computation on the GPU.

5 Future Work

Our current work includes a re-implementation of the lattice finding algorithm on the GPU, which should result in even higher performance of our method. We are currently working on a lattice finding algorithm that almost entirely executes on the GPU which will reduce execution times even further.

Tests have shown that the simple template matching works well for extracting a lattice even for severely distorted patterns. The full lattice is currently not automatically found in these cases, since the lattice model is restricted and the displacement vectors variation is constrained. If we constrain the divergence of the displacement vectors only

locally, i.e. for nearby feature points, we could use methods describe by Hays [11] or Park [13] to extract the full lattice.

References

1. Lindeberg, T.: Scale-space theory: A basic tool for analysing structures at different scales. Journal of Applied Statistics (1994) 224–270
2. Lowe, D.G.: Object recognition from local scale-invariant features. In: International Conference on Computer Vision. (1999) 1150–1157
3. Lowe, D.G.: Distinctive image features from scale-invariant keypoints. International Journal of Computer Vision **60** (2004) 91–110
4. Morel, J.M., Yu, G.: On the consistency of the SIFT Method. Preprint, CMLA **26** (2008)
5. Matas, J., Chum, O., Urban, M., Pajdla, T.: Robust wide baseline stereo from. In: British Machine Vision Conference. (2002) 384–393
6. Bay, H., Tuytelaars, T., Gool, L.V.: Surf: Speeded up robust features. In: ECCV. (2006) 404–417
7. Kiryati, N., Gofman, Y.: Detecting symmetry in grey level images: The global optimization approach. In: Proceedings of the 13th International Conference on Pattern Recognition, volume I. (1996) 951–956
8. Scognamillo, R., Rhodes, G., Morrone, C., Burr, D.: A feature-based model of symmetry detection. In: Proceedings Biological sciences The Royal Society, volume 270. (2003) 1727–1733
9. Loy, G., Eklundh, J.O.: Detecting symmetry and symmetric constellations of features. In: ECCV. (2006) 508–521
10. Lin, H.C., Wang, L.L., Yang, S.N.: Extracting periodicity of a regular texture based on autocorrelation functions. Pattern Recogn. Lett. **18** (1997) 433–443
11. Hays, J., Leordeanu, M., Efros, A.A., Liu, Y.: Discovering texture regularity as a higher-order correspondence problem. In: ECCV, Springer (2006) 522–535
12. Lin, W.C., Liu, Y.: Tracking dynamic near-regular texture under occlusion and rapid movements. In Leonardis, A., Bischof, H., Pinz, A., eds.: Computer Vision ECCV 2006. Volume 3952 of Lecture Notes in Computer Science. (Springer Berlin - Heidelberg) 44–55
13. Park, M., Collins, R., Liu, Y.: Deformed lattice discovery via efficient mean-shift belief propagation. In Forsyth, D., Torr, P., Zisserman, A., eds.: Computer Vision ECCV 2008. Volume 5303 of Lecture Notes in Computer Science. (Springer Berlin - Heidelberg) 474–485
14. Dischler, J.M., Zara, F.: Real-time structured texture synthesis and editing using image-mesh analogies. The Visual Computer **22** (2006) 926–935
15. Liu, Y., , Liu, Y., Tsin, Y.: The promise and the perils of near-regular texture. International Journal of Computer Vision **62** (2002) 1–2
16. Liu, Y., Collins, R.T., Tsin, Y.: A computational model for periodic pattern perception based on frieze and wallpaper groups. IEEE Transactions on Pattern Analysis and Machine Intelligence **26** (2004) 354–371