Composite Texture Synthesis

A. Zalesny[1], V. Ferrari[1], G. Caenen[2] and L. Van Gool[1,2]

[1]D-ITET/BIWI, ETH Zurich, Switzerland, lastname@vision.ee.ethz.ch

[2]ESAT/PSI Visics, Univ. of Leuven, Belgium, lastname@esat.kuleuven.ac.be

Abstract

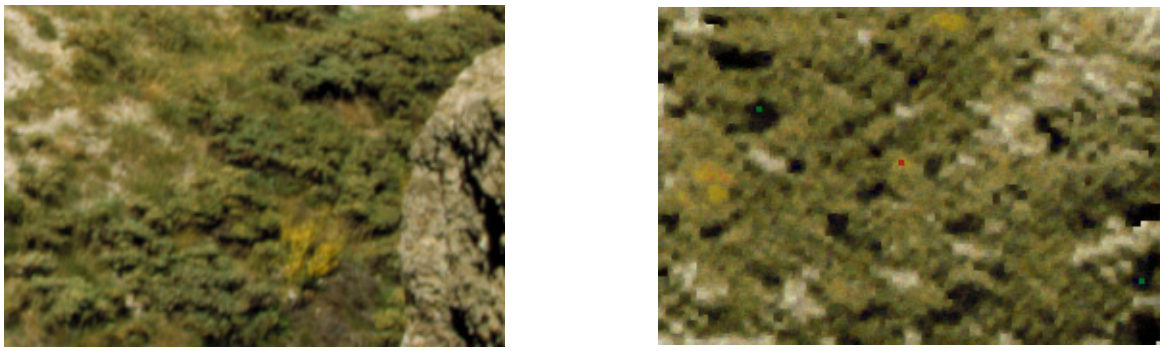
Many textures require complex models to describe their intricate structures. Their modeling can be simplified if they are considered composites of simpler subtextures. After an initial, unsupervised segmentation of the composite texture into the subtextures, it can be described at two levels. One is a label map texture, which captures the layout of the different subtextures. The other consists of the different subtextures. This scheme has to be refined to also include mutual influences between textures, mainly found near their boundaries. The proposed composite texture model also includes these. The paper describes an improved implementation of this idea. Whereas in a previous implementation subtextures and their interactions were synthesized sequentially, this paper proposes a parallel implementation, which yields results of higher quality.

Index Terms – texture synthesis, texture analysis, statistical texture modeling, composite textures, hierarchical texture model.

1. Introduction

Many textures are so complex that for their analysis and synthesis they can better be considered a composition of simpler subtextures. A good case in point is landscape textures. Open pastures can be mixed with patches of forest and rock. The direct synthesis of the overall texture would defy existing methods, which is in compliance with observations in [Gousseau] stating that the textures are as a rule intermediate objects between homogeneous fields and complicated scenes, and the analysis/synthesis of the totally averaged behavior can fail in reproducing important texture features. The whole only appears to be one texture at a very coarse scale. In terms of intensity, colors, and simple filter outputs such scene can not be considered "homogeneous". The homogeneity rather exists in terms of the regularity (in a structural or stochastic sense) in the layout of simpler subtextures as well as in the properties of the subtextures themselves. We propose such hierarchical approach to texture synthesis. We show that this approach can be used to synthesize intricate textures and even complete scenes.

Fig. 1 shows an example texture image. A model for this texture was extracted using a method in [Zalesny 2001]. This method will be referred to as the "basic method", and such model as a "basic model".



(a) original (b) synthesis from basic model

Fig. 1. The image on the left shows a complex landscape texture; the image on the right shows the result of attempting to synthesize similar texture from its basic model that considers the original as one, single texture

Fig. 1 also shows a texture that has been synthesized on the basis of this model. As can be seen, the result is not entirely convincing. The problem is that the pattern in the example image is too complicated to be dealt with as a single texture. In cases like this a more sophisticated texture model is needed. As mentioned, the idea explored in this paper is that a prior decomposition of such textures into their subtextures (e.g. grass, sand, bush, rock, etc. in the example image) is useful. As mentioned in [Paget], distinguishing between subtextures, i.e. decomposing or segmenting, is in general easier than texture synthesis. This allows to separate both procedures and use much more complex iterative modeling/generating algorithms only for analysis/synthesis. Despite the fact, that simple pairwise pixel statistics are used for modeling, their optimal combination yields a powerful generative model of textures, whereas for segmentation it is enough to use a preselected filter bank.

The layout of these subtextures can be described as a "label map", where pixels are given integer labels corresponding to the subtexture they belong to. This label map can be considered as a texture in its own right, which can be modeled using an approach for simple textures such as our basic modeling scheme and of which more can then be synthesized. Also the subtextures can be modeled using their basic model and these subtextures can be filled in at the places prescribed by the synthesized label map.

The creation of a composite texture model starts with one or more example images of that texture as the only input. A first step is the decomposition of the texture into its subtextures. This is the subject of Section 3. We propose an unsupervised segmentation scheme, which calculates pixel similarity scores on the basis of color and local image structure and which uses these to group pixels through efficient clique partitioning. Once this decomposition has been achieved, the hierarchical texture model can be extracted. This process is described in Section 2.
Based on such model, texture synthesis amounts to first synthesizing a label map, and then synthesizing the subtextures at the corresponding places. Results are shown in Section 4. Section 5 concludes the paper.

This paper presents an improved version of an earlier composite texture approach that we presented in [Zalesny 2002]. An idea similar to our composite texture approach has been propounded independently in [Hertzmann], but their "texture by numbers" scheme (based on smart copying from the example [Efros, Wei]) did not include the automated extraction or synthesis of the label maps (they were hand drawn).

2. Composite texture modeling

This section focuses on the construction of the composite texture model, on the basis of an example image and its segmentation. We start with a short description of the "basic model", used for the description of simple textures. Then, interdependencies between subtextures are noted to be an issue. After these introductory sections the actual composite texture modeling process is described. Finally, it is explained how the model is used for the synthesis of composite textures.

2.1. The basic texture model

Before explaining the composite texture algorithm, we concisely describe our basic texture model for single textures, in order to make this paper more self-contained and to introduce some of the concepts that will also play an important role with the composite texture scheme. The point of departure of the basic model is the co-occurrence principle. Simple statistics about the colors at pixel pairs are extracted, where the pixels take on carefully selected, relative positions. The approach differs in this selectivity from more broad-brush co-occurrence methods [Gagalowicz, Gimel'farb], where all possible pairs are considered. Every different type of pair - i.e. every different relative position - is referred at as a clique type. The notion "clique" is meant in a graph-theoretical sense where pixels are nodes, and arcs connect those

pixels into pairs whose statistics are used in the texture model. This way one obtains the so-called neighborhood graph where pairs are second-order cliques. Fig. 2 exemplifies clique types assuming the translation invariance scheme.

Fig. 2. Clique type assignment for the translation invariant scheme

The statistics gathered for these cliques are the histograms of the intensity differences between the head and tail pixels of the pairs, and this for all three color bands R, G, and B. Clique types are selected mutually dependent, one-by-one, each time adding the type with statistics computed from the current synthesis deviating most from those of the target texture. The initial set of clique types is restricted only by the maximal clique length, which is proportional to the size of the image under consideration. After the clique selection process is over, all clique types have statistics similar to those of the target texture, but only a small fraction of the types needed to be included in the model, which therefore is very compact. Hence, the basic model consists of a selection of cliques (the so-called "neighborhood system") and their color statistics (the so-called "statistical parameter set"). A more detailed explanation about these basic models and how they are used for texture synthesis is given in [Zalesny 2001].

2.2. Subtexture interactions

A straightforward implementation for composite texture synthesis would use the basic method first to synthesize a novel label map, after which it would be applied to each of the subtextures separately, in order to fill them in at the appropriate places. In reality, subtextures are not stationary within their patch boundaries. Typically there are natural processes at work (geological, biological, ...) that cause interactions between the subtextures. There are transition zones around some of the subtexture boundaries. Fig. 3 illustrates such transition effect. The image on the left is an original image of zebra fur. The image in the middle is the result of taking the left image label map (consisting of the black and white stripes) and filling in the black and white subtextures. The boundaries between the two look unnatural. The image on the right has been synthesized taking the subtexture interactions into account, using the algorithm proposed in this paper. The texture looks much better now.

(a) original

(b) subtexture synthesis without interactions

(c) subtexture synthesis with interactions

Fig. 3. Composite texture synthesis of zebra fur with and without subtexture interactions demonstrates the importance of the latter

In [Zalesny 2002] we have proposed a scheme that orders the subtextures by complexity, and then embarks on a sequential synthesis, starting with the simplest. Only interactions with subtextures that have been synthesized already are taken into account. In this paper we propose an alternative, parallel approach, where all subtextures and their interactions are taken care of simultaneously, both during modeling and synthesis. The sequential aspect that remains, is that first the label texture is synthesized and only then the subtextures.

2.3. A parallel composite texture scheme

The parallel composite texture scheme is a generalization of the basic scheme in [Zalesny 2001]. It is also based on the careful selection of cliques and the statistics of their head-tail intensity differences. Yet, for composite texture a distinction will be made between "intra-label" and "inter-label" cliques. Intra-label cliques have both their head and tail pixels within the same subtexture. Inter-label cliques have their head and tail pixels within different subtextures.

The parallel composite texture modeling scheme takes the following steps:

1. Segment the example image of the composite texture. The image and the resulting label map are the input for the modeling procedure. Let K be the number of subtextures and B the number of image color bands.
2. Calculate the intensity difference histograms for all inter-label and intra-label clique types that occur in the example image, up to a maximal, user-specified head-tail distance (the clique length). They will be referred to as reference histograms. After this step the example image is no longer needed.
3. Construct an initial composite texture model containing the $K\times B$ intensity histograms for each subtexture and each color band and $2K\times B$ clique types and their statistics: for each subtexture/band the shortest horizontal and shortest vertical cliques are added to the model. The head and tail pixels are direct neighbors.

Loop:

4. Synthesize a texture using the input label map and the current composite texture model, as discussed further on.
5. Calculate the intensity difference histograms for all inter-label and intra-label clique types from the image synthesized in step 4, up to a maximal, user-specified clique length. They will be referred to as current histograms.
6. Measure the histogram distances – a weighted (see below) Euclidean distance between the reference and current histograms.
7. If the maximal histogram distance is less than a threshold go to the step 9.
8. Add the following $2K$ histograms to the composite model: (a) K intra-label ones, for each subtexture the one with the largest histogram distance; (b) K inter-label ones, those with the largest histogram distance for pairs of subtextures (k,n) for all n and one fixed k at a time.

Loop end.

9. Model the label map as a normal non-composite texture (i.e. using the basic model), except that instead of intensity difference histograms co-occurrence matrices are used.

Stop.

The number of image bands could vary in general, including for example multispectral images or even additional heterogeneous image properties like filter responses etc. In the latter case the intensity differences can be substituted by other statistics including the complete or quantized co-occurrence.

The statistical data kept for the cliques normally consist of intensity histograms, but for the label map texture model a complete label co-occurrence matrix is stored. Indeed, the label map generation is driven by label co-

occurrences and not differences because the latter are meaningless in that case. Also, there are only a few labels in a typical segmentation and, hence, taking the full co-occurrence matrix rather than only difference histograms into account comes at an affordable cost. We now concisely describe how texture synthesis is carried out, and how the histogram distances are calculated.

2.4. Composite texture synthesis

Texture synthesis is organized as an iterative procedure that generates an image sequence, where histogram distances for the clique types in the model decrease with respect to the corresponding reference histograms. This evolution is based on non-stationary, stochastic relaxation, underpinned by Markov Random Field theory. Non-stationary means that the control parameters of the synthesizer (in our case these are so-called Gibbs parameters of the random field) are changed based on the comparison of the reference and current histograms. A more detailed account is given in [Zalesny 2001].

A separate note on the synthesis of subtexture interactions is in order here. Even if the modeling procedure selects quite a few inter-label clique types, they still represent a very sparse sample from all possible such clique types, as there are of the order of K^2 subtexture pairs, as opposed to K subtextures, for which just as many clique types were selected. Thus, many subtexture pairs do not interact according to the composite texture model, i.e. there are no cliques in the model corresponding to the label pair under consideration. For such pairs, subtexture knitting - a predefined type of interaction - is used. During knitting neighboring pixels outside the subtexture's area are nevertheless treated as if they lay within, and this for all clique types of that subtexture. The intensity difference is calculated and its entry in the histogram for the given subtexture is used. Knitting produces smooth transitions between subtextures. In case clique types describing the interaction between a subtexture pair have been included in the model, their statistical data are used instead and knitting is turned off for that pair. During normal synthesis, the composite texture model is available from the start and all subtexture pairs without modeled interactions are known beforehand. Hence, knitting is always applied to the same subtexture pairs, i.e., it is static. During the texture modeling stage the set of selected clique types will be constantly updated and the knitting is adaptive. Knitting will be automatically switched off for subtexture pairs with a selected inter-label clique type. This will also happen for subtexture pairs that do not interact, e.g. a texture that simply occludes another one. This is because during the composite texture modeling process knitting is turned on for all pairs which are left without an inter-label clique type. Knitting will not give good results for independent pairs though, as it blends the textures near their border. Hence, a clique type will be selected for such pairs, as the statistics near the border are being driven away from reality under the influence of knitting. The selection of this clique type turns off further knitting, and will itself prescribe statistics that are in line with the subtextures' independence. This process may not be very elegant in the case of independent subtextures, but it works.

2.5. Histogram distances

The texture modeling algorithm heavily relies on histogram distances. For intra-label cliques, these are weighted Euclidean distances, where the weight is calculated as follows:

$$\text{weight}(k,k,type) = \frac{N(k,k,type)}{N_{\max}}, \tag{1}$$

where the clique count $N(\cdot)$ is the number of cliques of this type having both the head and the tail inside the label class k, and $N_{\max}$ is the maximum clique count reached over all clique types. The rationale behind this weighing is

that types with low clique counts must not dominate the model, as the corresponding statistical relevance will be wanting. This weight also reduces the influence of long clique types, which tend to have lower clique counts. For the inter-label cliques, this effect is achieved by making the weights dependent on clique length explicitly:

$$\text{weight}(k,n,type) = \left(1 - \frac{l^2(type)}{(l_{\max}+1)^2}\right)\frac{N_{\max}(k,n)}{N_{\max}}, \tag{2}$$

where $N_{\max}(k,n)$ is the maximal clique count among the types for the given subtexture pair, $l(type)$ is the clique length, and $l_{\max}$ is the maximal clique length taken over all types present in the example texture. Such weighing again increases the statistical stability and gives preference to shorter cliques, which seems natural as the mutual influence of the subtextures can be expected to be stronger near their boundary.

2.6. Parallel vs. sequential approach

As mentioned before, prior to this work we have proposed a sequential composite texture scheme [Zalesny 2001]. The main advantage of the parallel approach discussed here is that all bidirectional, pairwise subtexture interactions can be taken into account. This, in general, results in better quality and a more compact model. Additionally, there is a disturbing asymmetry between the model extraction and subsequent texture synthesis procedures with the sequential approach. During modeling the surrounding subtextures are ideal, i.e. taken from the reference image. This is not the case during the synthesis stage, where the sequential method has to build further on the basis of previously synthesized subtextures. There is a risk that the sequentially generated subtextures will be of lower and lower quality, due to error accumulation. The parallel approach, in contrast, is free from these drawbacks, as both the modeling and synthesis stages operate under similar conditions. The advantage of the sequential modeling step (but not the synthesis one!) is that every subtexture can be modeled simultaneously, distributed over different computers. But as speed is more crucial during synthesis, this advantage is limited in practice. During model extraction, the foremost problem is the clique type selection. This problem is more complicated in the parallel case, as there are many more clique types to choose from. At every iteration of the modeling algorithm a choice can be made between all inter- and intra-label cliques.

3. Texture decomposition

In order for the texture modeling and synthesis approach to work, the decomposition of the texture into its subtextures needs to be available. We propose an unsupervised segmentation scheme, which calculates pixel similarity scores on the basis of color and local image structure and which uses these to group pixels through efficient clique partitioning. Once this decomposition has been achieved, the hierarchical texture model can be extracted. The approach is unsupervised in the sense that neither the number nor the sizes of subtextures are given to the system.

Important to mention is that, in contrast with traditional segmentation schemes, we envisage a clustering that is not necessarily semantically correct. Our goal is to reduce the complexity of the texture in terms of structure and color properties.

3.1. Pixel similarity scores

For the description of the subtextures, both color and structural information is taken into account. Local statistics of the (L,a,b) color coordinates and response energies of a set of Gabor filters $(f_1,\ldots,f_n)$ are chosen, but another

wavelet family or filter bank could be used to optimize the system.

The initial (L,a,b)-color and $(f_1,\ldots,f_n)$-structural feature vector of an image pixel i are both referred to as $\mathbf{x}_i$, for the sake of simplicity. The local statistics of the vectors $\mathbf{x}_j$ near the pixel i are captured by a local histogram p_i.

To avoid problems with sparse high-dimensional histograms, we first cluster both feature spaces separately. For the structural features this processing is done in the same vein as the texton analysis in [Malik]. The cluster centers are obtained using the k-means algorithm and will serve as bin centers for the local histograms.

Instead of assigning a pixel to a single bin, each pixel is assigned a vector of weights that express its affinity to the different bins. The weights are based on the Euclidean distances to the bin centers: if $d_{ik} = \|\mathbf{x}_i - \mathbf{b}_k\|$ is the distance between a feature value $\mathbf{x}_i$ and the k-th bin center, we compute the corresponding weight as

$$w_{ik} = e^{-d_{ik}^2/2\sigma^2}. \tag{3}$$

The resulting local *weighted histogram* p_i of pixel i is obtained by averaging the weights over a region R_i:

$$p_i(k) = \frac{1}{|R_i|}\sum_{j\in R_i} w_{jk}. \tag{4}$$

In our experiments, R_i was chosen a fixed shape: a disc with a radius of 8 pixels. The values $p_i(k)$ can therefore be computed for all pixels at once (denoted $P(k)$), using the convolution:

$$P(k) = W_k * \chi_D \text{ with } W_k(i) = w_{ik} \text{ and } \chi_D(i) = \begin{cases} 1 \text{ if } i \in D, \\ 0 \text{ if } i \notin D, \end{cases} \tag{5}$$

where $D = \{i \mid \|i\| < 8\}$.

The resulting weighted histogram can be considered a smooth version of the traditional histogram. The weighting causes small changes in the feature vectors (e.g. due to non-uniform illumination) to result in small changes in the histogram. In traditional histograms this is often not the case as pixels may suddenly jump to another bin. Fig. 4 illustrates this by computing histograms of two rectangular patches from a single Brodatz texture. These patches have similar texture, only the illumination is different. Clearly the weighted histograms (right) are less sensitive to this difference. This is reflected in a higher Bhattacharyya score (6).

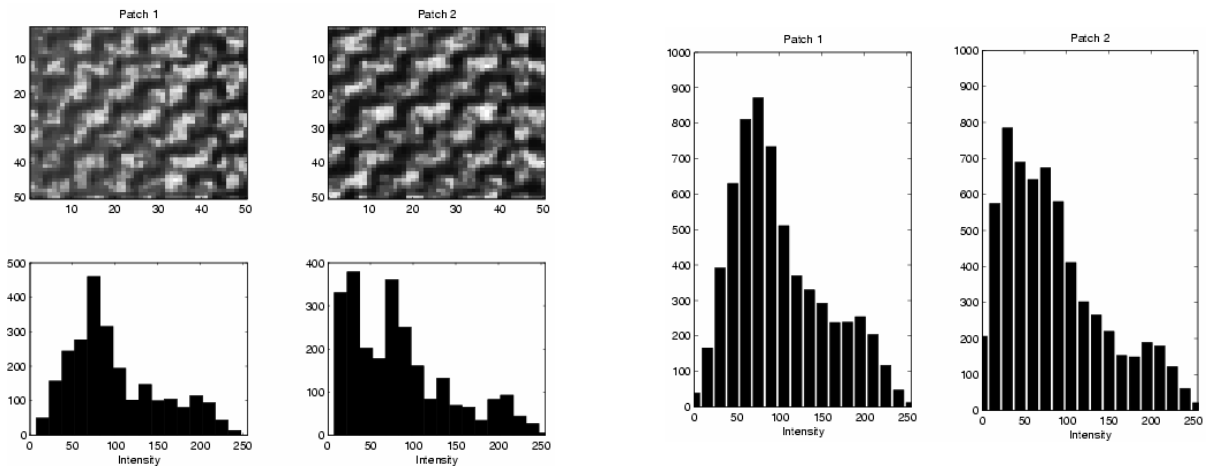


Fig. 4. (top left) patches with identical texture and different illumination; (bottom left) traditional intensity histograms of the patches; (right) weighted histograms of the patches

Color and structural histograms are computed separately. In a final stage, the color and structure histograms are simply concatenated into a single, longer histogram and the $p_i(k)$ are scaled to ensure the sum to 1.

In order to compare the feature histograms, we have used the Bhattacharyya coefficient ρ. Its definition for two frequency histograms $p=(p_1,\ldots,p_n)$ and $q=(q_1,\ldots q_n)$ is:

$$\rho(p,q)=\sum_i \sqrt{p_i q_i}\ . \tag{6}$$

This coefficient is proven to be more robust [Aherne] in the case of zero count bins and non-uniform variance than the more popular chi-squared statistic (denoted χ^2). In fact, after a few manipulations one can show the following relation in case the histograms are sufficiently similar:

$$\rho(p,q)\approx 1-\frac{1}{8}\sum_i \frac{(q_i-p_i)^2}{p_i}=1-\frac{1}{8}\chi^2(p,q). \tag{7}$$

Another advantage of the Bhattacharyya coefficient over the χ^2-measure is that it is symmetric, which is more natural when similarity has to be expressed.

In order to evaluate the similarity between two pixels, their feature histograms are not simply compared. Rather, the comparison of the histogram for the first pixel is made with those of all pixels in a neighborhood of the second. The best possible score is taken as the similarity $S'(i,j)$ between the two pixels. This allows the system to assess similarity without having to collect histograms from large regions R_i. Additional advantages of such approach are that boundaries between subtextures are slightly better located and narrow subtextures can still be distinguished. Fig. 5 illustrates the behavior near texture boundaries and an example of an improved segmentation as well as a comparison with Normalized Cuts is shown in Fig. 6.

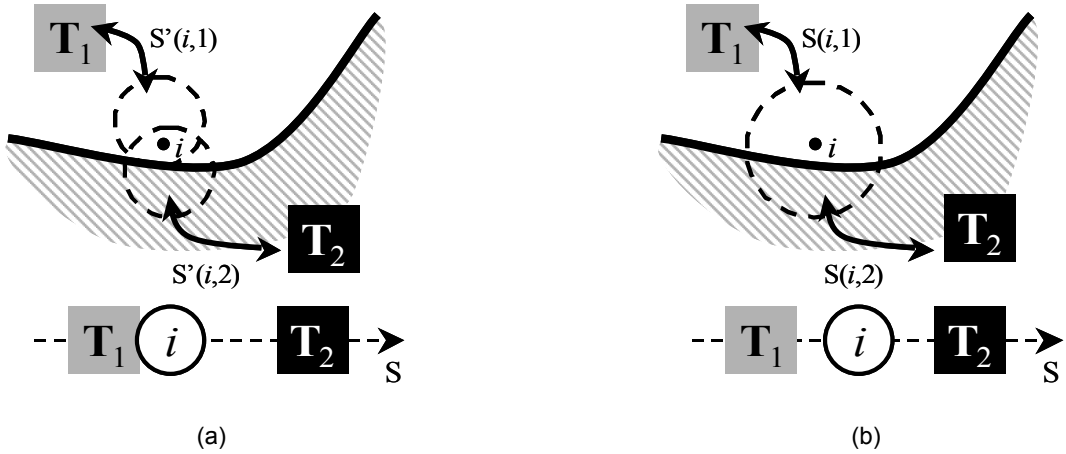


Fig. 5. **(a) comparison between two pixels using shifted matching; the dashed lines indicate the supports of the histograms that yield optimal similarities between i and subtextures $\mathbf{T}_1$ and $\mathbf{T}_2$; (b) comparison without shifts. The similarity scores $S'(i,1)$ and $S'(i,2)$ for (a), although *both* significantly higher than $S(i,1)$ and $S(i,2)$ in (b), are proportionally closer to the desired similarities, as is indicated schematically**

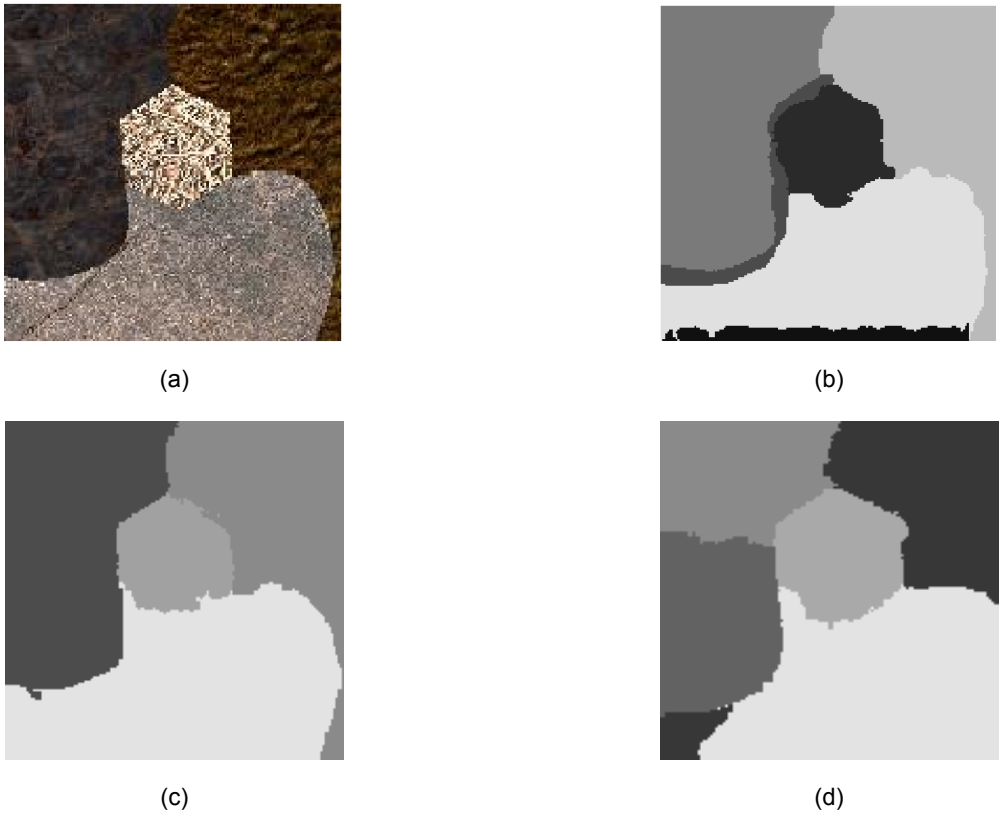


Fig. 6. **(a) original image (texture collage) and segmentations obtained: (b) with CP, larger regions, and no shifted matching; (c) using CP and shifted matching (smaller regions, mean shift optimization); and (d) using a version of Normalized Cuts along with its standard parameter set available at http://www.cs.berkeley.edu**

Using the shifting strategy has three major effects on the similarity scores, depending on the location of the pixels:

1. Pixels that lie *in the interior of a subtexture* only have strong similarities with this texture.

2. Pixels *near a texture border* (Fig. 5) attain an increased similarity score when compared to the particular subtexture they belong to. Comparisons made with the neighboring texture will also yield higher scores, yet less extreme ones.
3. Pixels *on a texture border* are very similar to both adjacent textures.

To fully exploit this shifted matching result we transform our similarity scores using $S'(i,j) \rightarrow S'(i,j)^n$, with $n = 10$ in our experiments. This causes the similarities established between pixels of type 2 and their neighbor texture to decrease significantly.

The search for the location with the best matching histogram close to the second pixel is based on the mean shift gradient to maximize the Bhattacharyya measure [Comaniciu]. This avoids having to perform an exhaustive search.

A final refinement is by defining a symmetric similarity measure S: $S(i,j) = S(j,i) = \max\{S'(i,j), S'(j,i)\}$.

As shifted matches cause neighboring pixels to have an exact match, the similarity scores are only computed for a subsample (a regular grid) of the image pixels, which also yields a computational advantage. Yet, after segmentation of this sample, a high-resolution segmentation map is obtained as follows. The histogram of each pixel is first compared to each entry in the list of neighboring sample histograms. The pixel is then assigned to the best matching class in the list.

Our particular segmentation algorithm requires a similarity matrix S with entries ≥ 0 indicating that pixels are likely to belong together and entries < 0 indicating the opposite. The absolute value of the entry is a measure of confidence. So far, all the similarities S have positive values. We subtract a constant value, which was determined experimentally and kept the same in all our experiments. With this fixed value images with different numbers of subtextures could be segmented successfully. Hence, the number of subtextures was not given to the system, as would e.g. be required in *k*-means clustering. Having this threshold in the system can be an advantage, as it allows the user to express what he or she considers being perceptually similar: the threshold determines the simplicity or homogeneity of each subtexture or in other words, the level of hierarchy.

3.2. Pixel grouping

In order to achieve the intended, unsupervised segmentation of the composite textures into simpler subtextures, pixels need to be grouped into disjoint classes, based on their pairwise similarity scores. Taken on their own, these similarities are too noisy to yield robust results. Pixels belonging to the same subtexture may e.g. have a negative score (false negative) and pixels of different subtextures may have positive scores (false positives). Nevertheless, taken altogether, the similarity scores carry quite reliable information about the correct grouping. The transitivity of subtexture membership is crucial: if pixels i and j are in the same class and j and k too, then i, j and k must belong to the same class. Even if one of the pairs gets a falsely negative score, the two others can override a decision to split. Next, we formulate the texture segmentation problem so as to exploit transitivity to detect and avoid false scores. We present a time-optimized adaptation of the grouping algorithm we first introduced in [Ferrari]. We construct a complete graph G where each vertex represents a pixel and where edges are weighted with the pairwise similarity scores. We partition G into completely connected disjoint subsets of vertices (usually also cliques but please note the different meaning in this context) through edge removal so as to maximize the total score on the remaining edges (Clique Partitioning, or CP). As to avoid confusion with a clique concept defined earlier, we will use the word "component" instead of clique

here. The transitivity property is ensured by the component constraint: every two vertices in a component are connected, and no two vertices from different components are connected. The CP formulation of texture segmentation is made possible by the presence of positive *and* negative weights: they naturally lead to the definition of a *best solution* without the need of knowing the number of components (subtextures) or the introduction of artificial stopping criteria as in other graph-based approaches based on strictly positive weights [Shi, Aslam]. On the other hand, our approach needs the parameter t_0 that determines the splitting point between positive and negative scores. But, as our experiments have shown, the same parameter value yields good results for a wide range of images. Moreover, the same value yields good results for examples with a variable number of subtextures. This is much better than having to specify this number, as would e.g. be necessary in a *k*-means clustering approach.

CP can be solved by Linear Programming [Graham] (LP). Let w_{ij} be the weight of the edge connecting (i,j), and $x_{ij} \in \{1,0\}$ indicate whether the edge exists in the solution ($0 = no,\ 1 = yes$). The following LP can be established:

$$\text{maximize} \sum_{1 \le i < j \le n} w_{ij} x_{ij} \text{ subject to } \begin{cases} x_{ij} + x_{jk} - x_{ik} \le 1, \forall 1 \le i < j < k \le n \\ x_{ij} - x_{jk} - x_{ik} \le 1, \forall 1 \le i < j < k \le n \\ -x_{ij} + x_{jk} + x_{ik} \le 1, \forall 1 \le i < j < k \le n \\ x_{ij} \in \{0,1\}, \forall 1 \le i < j \le n. \end{cases} \tag{8}$$

The inequalities express the transitivity constraints, while the objective function to be maximized corresponds to the sum of the intra-component edges.

Unfortunately CP is an NP-hard problem [Graham]: LP (8) has worst case exponential complexity in the number n of vertices (pixels), making it impractical for large n. The challenge is to find a practical way out of this complexity trap. The correct partitioning of the example in Fig. 7 is $\{\{1,3\},\{2,4,5\}\}$. A simple greedy strategy merging two vertices (i,j) if $w_{ij} > 0$ fails because it merges $(1,2)$ as its first move. Such an approach suffers from two problems: the generated solution depends on the *order* by which vertices are processed and it looks only at *local* information.

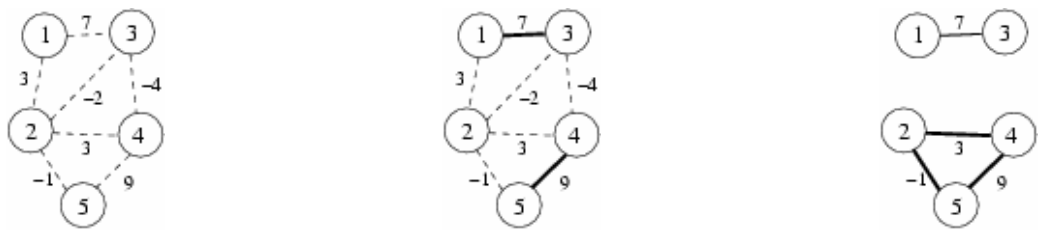


Fig. 7. An example graph and two iterations of our heuristic. Edges that are not displayed have zero weight

We propose the following iterative heuristic. The algorithm starts with the partition

$$\Phi = \{\{i\}\}_{1 \le i \le n} \tag{9}$$

composed of n singleton components each containing a different vertex. The function

$$m(c_1, c_2) = \sum_{i \in c_1, j \in c_2} w_{ij} \tag{10}$$

defines the cost of merging components c_1, c_2. We consider the functions

$$b(c) = \max_{t\in\Phi} m(c,t),$$
$$d(c) = \arg\max_{t\in\Phi} m(c,t), \tag{11}$$

representing, respectively, the score of the best merging choice for component c and the associated component to merge with. We merge components c_i, c_j if and only if the three following conditions are met simultaneously

$$d(c_i) = c_j,\ d(c_j) = c_i,\ b(c_i) = b(c_j) > 0\,. \tag{12}$$

In other words, two components are merged only if each one represents the best merging option for the other and if merging them increases the total score. At each iteration the functions $b(c)$, $d(c)$ are computed, and all pairs of components fulfilling the criteria are merged. The algorithm iterates until no two components can be merged. The function m can be progressively computed from its values in the previous iteration. The basic observation is that for any pair of merged components $c_k = c_i \cup c_j$, the function changes to $m(c_l, c_k) = m(c_l, c_i) + m(c_l, c_j)$ for all $c_l \notin \{c_i, c_j\}$. This strongly reduces the amount of operations needed to compute m and makes the algorithm much faster than in [Ferrari].

Fig. 7 shows an interesting case. In the first iteration $\{1\}$ is merged with $\{3\}$ and $\{4\}$ with $\{5\}$. Notice how $\{2\}$ is, correctly, not merged with $\{1\}$ even though $m(\{1\},\{2\}) = 3 > 0$. In the second iteration $\{2\}$ is correctly merged with $\{4,5\}$, resisting the (false) attraction of $\{1,3\}$ ($b(\{1,3\},\{2\}) = 1$, $d(\{1,3\}) = \{2\}$). The algorithm terminates after the third iteration because $m(\{1,3\},\{2,4,5\}) = -3 < 0$. The second iteration shows the power of CP. Vertex 2 is connected to unreliable edges (w_{12} is false positive, w_{25} is false negative. Given vertices $\{1,2,3\}$ only, it is not possible to derive the correct partitioning $\{\{1,3\},\{2\}\}$; but, as we add vertices $\{4,5\}$, the *global information* increases and CP arrives at the correct partitioning.

The proposed heuristic is *order independent*, takes a more global view than a direct greedy strategy, and resolves several ambiguous situations while maintaining polynomial complexity. Analysis reveals that the exact amount of operations depends on the structure of the data, but it is at most $4n^2$ in the average case. Moreover, the operations are simple: only comparisons and sums of real values (no multiplication or division is involved).

In the first iterations, being biased toward highly positive weights, the algorithm risks to take wrong merging decisions. Nevertheless our merging criterion ensures this risk to quickly diminish with the size of the components in the correct solution (number of pixels forming each subtexture) and at each iteration, as the components grow and increase their resistance against spurious weights.

4. Results

In this section we first present the results of experiments that test the effectiveness of the CP algorithm as a substitute for the much slower LP algorithm, as proposed in section 3. Once the viability of this approach for the creation of label maps has been established, a second section describes results of our parallel composite texture synthesis.

4.1. Performance of the CP approximation

The practical shortcut for the implementation of CP may raise some questions as to its performance. In particular, how much noise on the edge weights (i.e. uncertainty on the similarity scores) can it withstand? And, how well does the heuristic approximation approach the true solution of CP?

We tested both LP and the heuristic on random instances of the CP problem. Graphs with *a priori* known, correct partitioning were generated. Their sizes differed in that both the number of components and the total number of vertices (all components had the same size) were varied. Intra-component weights were uniformly distributed in $[-a, 9]$ with a real number, while inter-component weights were uniformly distributed in $[-9, a]$, yielding an ill-signed edge percentage of $a/(a+9)$. This *noise level* could be controlled by varying the parameter a. Let the *difference between two partitionings* be the minimum amount of vertices that should change their component membership in one partitioning to get the other. The quality of the produced partitionings is evaluated in terms of *average percentage of misclassified vertices*: the difference between the produced partitioning and the correct one, averaged over 100 instances and divided by the total number of vertices in a single instance.

Table 1 reports the performance of our approximation for larger problem sizes. Given 25% noise level, the average error already becomes negligible with component sizes between 10 and 20 (less than 0.5%). In problems of this size, or larger, the algorithm can withstand even higher noise levels, still producing high quality solutions. In the case of 1000 vertices and 10 components, even with 40% noise level (a = 6), the algorithm produces solutions, which are closer than 1% to the correct one. This case is of particular interest as its size is similar to the typical texture segmentation problems.

Table 1. Performance of the CP approximation algorithm on various problem sizes

Vertices	Components	Noise Level	Err% Approx
40	4	25	0.33
60	4	25	0.1
60	4	33	2.1
120	5	36	1.6
1000	10	40	0.7

Table 2 shows a comparison between our approximation to CP and the optimal solution computed by LP on various problem sizes, with constant noise level set to 25% ($a = 3$). In all cases the partitionings produced by the two algorithms are virtually identical: the average percentual difference is very small as shown in the third column of the table. Due to the very high computational demands posed by LP, the largest problem reported here has only 24 vertices. Beyond that point, computation times run into the hours, which we consider as too impractical. Note that the average percentage of misclassifications quickly drops with the size of the components.

Table 2. Comparison of LP and our approximation. The noise level is 25%. Diff % is the average percentual difference between the partitionings produced by the two algorithms. The two Err columns report the average percentage misclassified vertices for each algorithm

Vert.	Components	Diff%	Err% LP	Err% Approx
15	3	0.53	6.8	6.93
12	2	0.5	2.92	3.08
21	3	0.05	2.19	2.14
24	3	0.2	1.13	1.33

The proposed heuristic is fast: it completed these problems in less than 0.1 seconds, except for the 1000 vertices one, which took about 4 seconds on the average. The ability to deal with thousands of vertices is particularly important in our application, as every pixel to be clustered will correspond to a vertex. Fig. 8 shows the average error for a problem with 100 vertices and 5 components as a function of the noise level (a varies from 3 to 5.5). Although the error grows faster than linearly, and the problem has a relatively small size, the algorithm produces high quality solutions in situations with as much as 36% of noise.

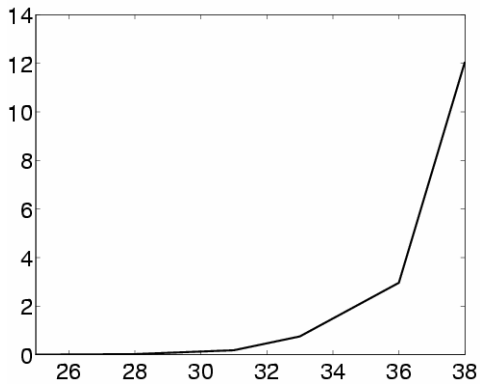


Fig. 8. Relationship between noise level and error, for a 100 vertices, 5 component problem. The average percentage of misclassified vertices (X-axis) is still low with as much as 36% noise level

These encouraging results show CP's robustness to noise and support our heuristic as a good approximation. Components in these experiments were only given the same size to simplify the discussion. The algorithm itself deals with differently sized components.

4.2. Composite texture synthesis results

This section presents some of the results obtained with the parallel composite texture synthesis as described in the paper. The various stages of our method will be systematically explained through an example. Afterwards we will focus on the different aspects that were touched upon in the paper using adequate examples.

The Complete Scheme – The landscape shown in Fig. 9 (top left) is clearly too complex to be synthesized when regarded as a single texture. Therefore, in keeping with the propounded composite texture approach, it is decomposed by analyzing the local color and structural properties. The CP-algorithm yields a label map based on the homogeneity of these properties, as shown in Fig. 9 (top right). Based on this label map and the example image, a model for the subtextures and their interactions is learned. In order to show the effectiveness of the texture synthesis, we also show

the same landscape layout, but with the label regions filled with textures generated on the basis of this model (Fig. 9 bottom). Of course, it is the very goal of the approach to go one step further and to create wholly new patterns. To that end, a *new* label map is generated, as shown in Fig. 10 (left), and this is filled with the corresponding subtextures (Fig. 10 right). The overall impression is quite realistic. The label map is capable to capture the main, systematic aspects of the layout. The sky is, e.g. created at the top, and also the different land cover types keep their natural, overall configuration.

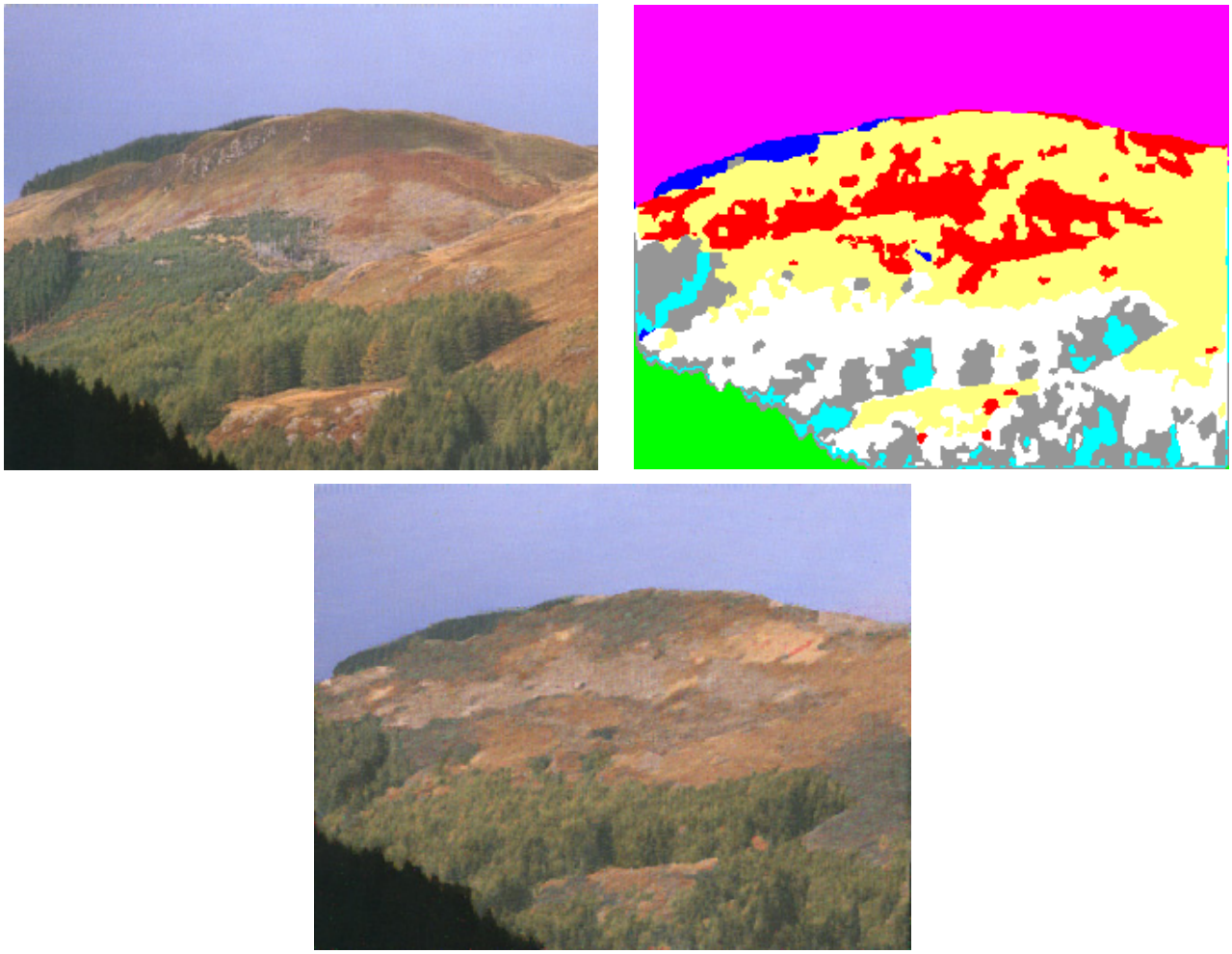


Fig. 9. Real landscape (top left) and label map (top right); synthetic landscape when keeping this label map (bottom)

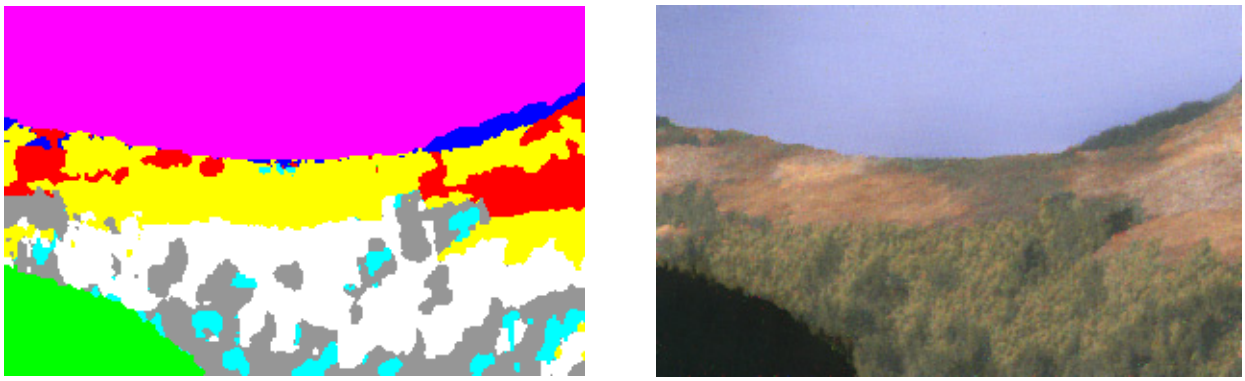


Fig. 10. Synthetic label map (left) and completely synthetic landscape (right)

Parallel vs. Sequential Approach – Fig. 11 shows three images of two landscapes. The ones on the left are the original images, used as the sole examples. The images in the middle show the result of our previous, sequential texture synthesis method [Zalesny 2002] applied to a synthesized label map. The images on the right show the same experiment, but now with textures synthesized by the parallel approach described in this paper. The overall results of the parallel method look better. In particular, unnaturally sharp transitions between the subtextures have been eliminated. This can e.g. be seen at the boundaries between the bush and grass textures of the top row. Also, the shadowing effects, learned as an interaction between bush and stone, added more reality to that result. In the sequential approach, only interactions with previously synthesized textures can be taken into account, not with those to come later in the process.

Dealing with Semantics – Fig. 12 shows a synthetic example of zebra fur. The label map for this example was synthesized only including information from the original image (Fig. 3). Apparently this did not suffice to capture sufficient statistics for the stripe layout (corresponding to the two subtextures in the label map). Without any further semantic knowledge, the system has generated one stripe, which is too wide, giving an unnatural impression. A larger texture sample should be provided as an example to resolve this. Fig. 13 shows another example where the model fails to pick up the underlying semantics. Despite the globally satisfying impression of the landscape, the model failed to "understand" the road separating the vineyard from the slope. Nevertheless the road was partially reconstructed as a transition effect when incorporating the subtexture interactions.

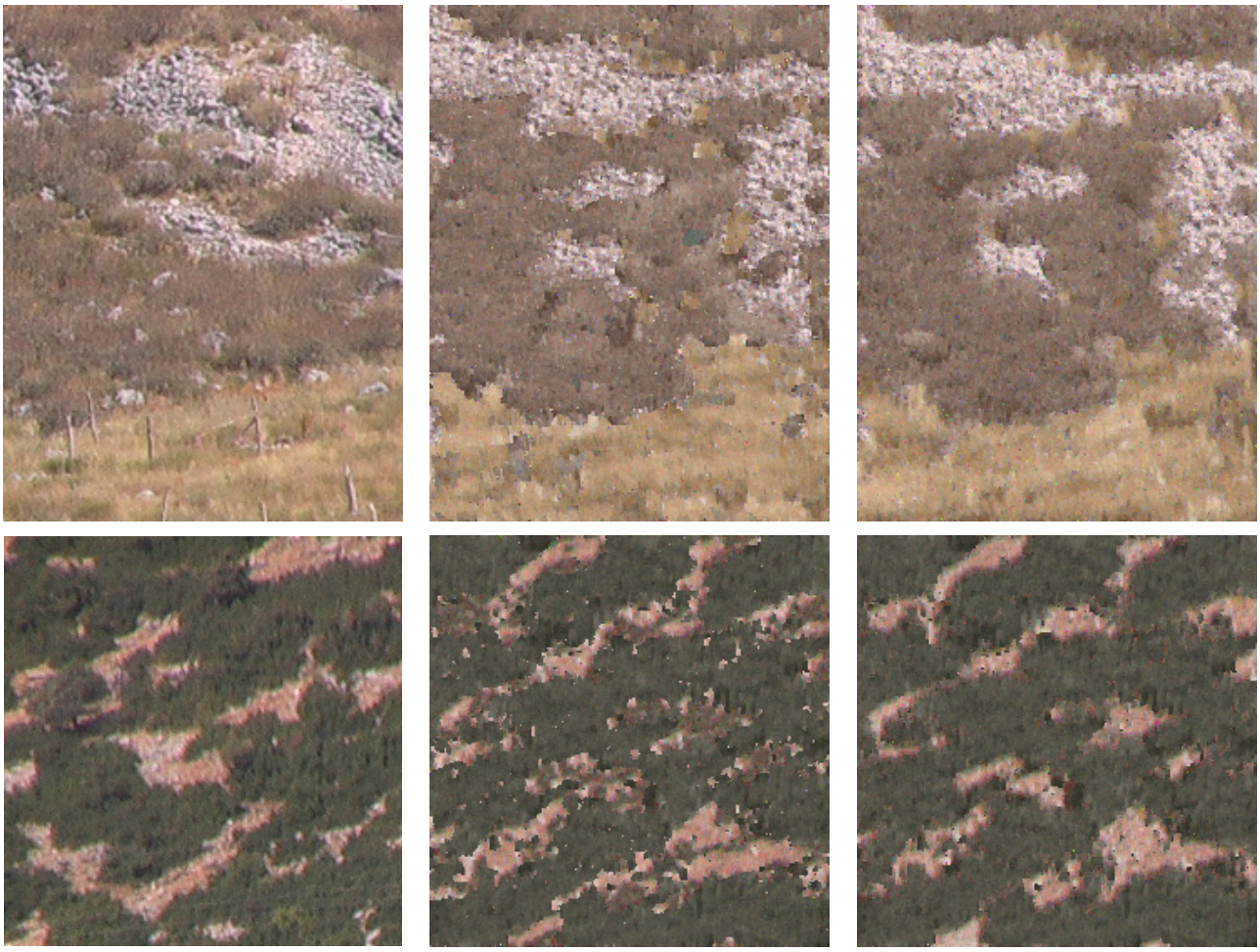
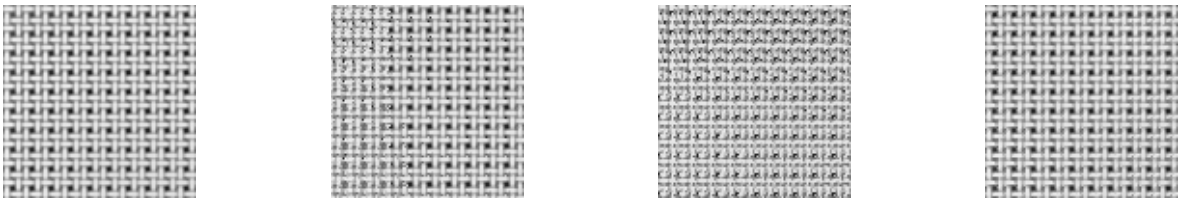


Fig. 11. Landscape texture synthesis. Left: original images with three different subtextures for the top landscape and two subtextures for the bottom landscape; middle: results with the older, sequential approach, right: parallel composite texture synthesis with better, more natural texture transitions

Fig. 12. Synthetic zebra fur based on a synthetic layout label map, demonstrates the importance of presenting a sufficiently large example image (cf. Fig. 3). One stripe is unnaturally wide

Fig. 13. Left: original aerial image of a landscape, right: image synthesized based on the original label map. The overall impression is satisfactory, but a semantic concept like road continuity was – of course – not picked up

Processing Simple Textures as Composite Textures – Fig. 14 illustrates that the composite texture approach even holds good promise for "simple" textures. Given the example on the left, a basic model was extracted and used to synthesize the two middle textures. For the texture (b) 59 cliques were selected and the synthesis was allowed to run over 3000 iterations. The image (c) shows the result if the number of iterations with the basic model is restricted to 500. Quality has clearly suffered. The image on the right is the result of a composite texture synthesis. Bright and dark regions were distinguished as two subtextures. Not only is this latter result better, it also took only 100 iterations, while the total number of cliques (for the two subtextures and their interactions) was still limited to 59. The computational complexity of the parallel approach is lower, because every pixel is involved in about only half as many cliques.



(a) original | (b) synthesis as a single texture, 3000 iterations | (c) synthesis as a single texture, 500 iterations | (d) synthesis as a composite texture

Fig. 14. Patterns that are traditionally considered as a single texture can benefit from the composite texture approach just the same. (a) original image; (b) synthesis using basic model with 3000 iterations; (c) synthesis using basic model with 500 iterations; (d) composite texture based synthesis (two subtextures)

Multiple Level Decomposition – We will now briefly illustrate the potential of increasing the number of layers in the composite texture description. So far, we have considered only two: the label map and directly beneath the subtextures. On the other hand, Fig. 14 has demonstrated that it may be useful to also subdivide the subtextures themselves. This is in agreement with the strategy as originally described, i.e., to decompose until a level of sufficient homogeneity in terms of simple properties is achieved. The sponge texture in the center of Fig. 16 (a) is quite intricate. The cavities show patterns that have to be captured quite precisely and simultaneously their structure varies over the texture, due to perspective effects and changing orientations. Fig. 15 shows a cutout of the sponge texture and a synthetic result, based on the basic model for this texture. The result more or less averages out the cavity variations in the example. The sponge is segmented out as a separate subtexture by an initial segmentation. Now, by lowering the

threshold introduced at the end of section 3.1 for this part, a further decomposition is achieved (Fig. 16 (b)). In Fig. 16 (c) the result of the synthesis based on this additional decomposition is shown. Clearly, the cavities have been recovered. This result is however preliminary as we don't have a systematic way of deciding where to stop the decomposition. This will be the subject of future research.

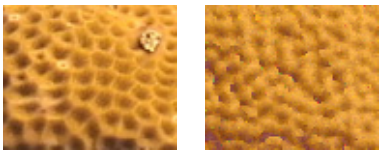


Fig. 15. The sea sponge texture with cavities cannot be synthesized as a single subtexture with the basic model

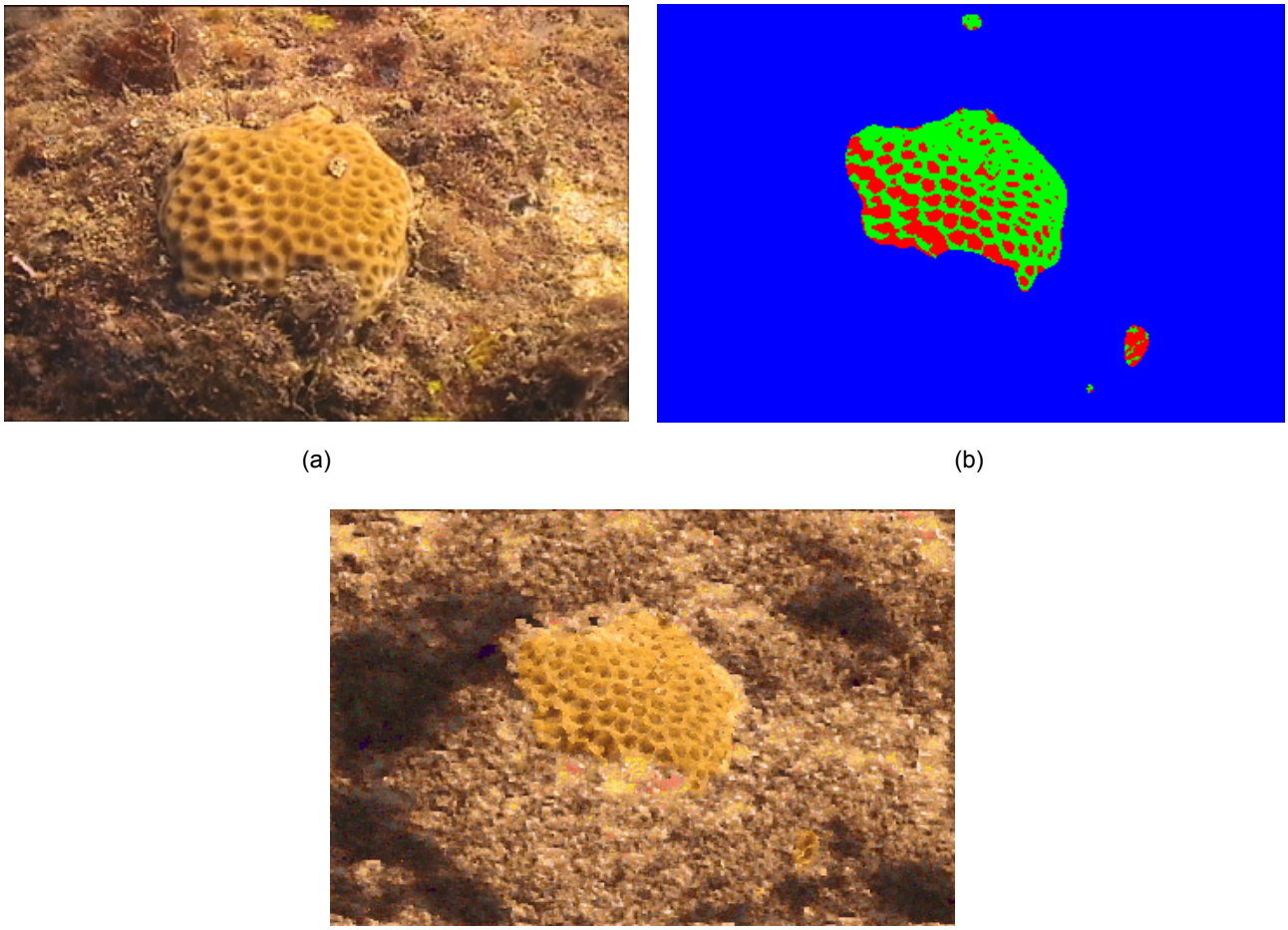


Fig. 16. (a) original image; (b) label map of the sponge texture after the extra decomposition into two subtextures; (c) synthesis based on the original label map using this extra level of decomposition. The cavities of the sponge are recovered

5. Conclusions

We have described a hierarchical texture synthesis approach, that considers textures as composites of simpler subtextures, that are studied in terms of their own statistics, that of their interactions, and that of their layout. The

approach supports the fully automated synthesis of complex textures from example images, without verbatim copying.

The following observation made this hierarchical approach possible: it is easier to distinguish textures than to synthesize them. It is in a full agreement with the complexity comparison between the segmenting and synthesizing stages. Segmentation uses fixed filters, which are texture- and mutually-independent, while the synthesis uses an optimal texture- and mutually-dependent pixel pair type selection obtained during the analysis-by-synthesis procedure. Despite a seaming simplicity of the pairwise statistics they, if taken together, represent much more intricate pixel interdependency compared to the segmenting filters.

In the current approach only one level of the hierarchy is thoroughly explored and a promising extension towards multiple levels is suggested. Future research will crack down on this problem of how to optimize the trade-off between the complexity of the label maps and the homogeneity of the subtextures they contain.

Acknowledgements

The authors gratefully acknowledge support by the European IST projects "3D-Murale" and "CogViSys".

References

Aherne, F., Thacker, N., and Rockett, P. 1998. The Bhattacharyya Metric as an Absolute Similarity Measure for Frequency Coded Data. Kybernetika 34, No. 4, pp.363-368.

Aslam, J., Leblanc, A., and Stein, C. 2000. A New Approach to Clustering. Workshop on Algorithm Engineering.

Comaniciu, D. and Meer, P. **2000**. Real-Time Tracking of Non-Rigid Objects using Mean Shift. In Proc. ICPR, Vol. 3, pp. 629-632

Efros, A. and Leung, T. **1999**. Texture Synthesis by Non-Parametric Sampling. In Proc. ICCV, Vol. 2, pp. 1033-1038.

Ferrari, V., Tuytelaars, T., and Van Gool, L. **2001**. Real-time affine region tracking and coplanar grouping. In Proc. CVPR, Vol. II, pp. 226-233.

Gagalowicz, A. and Ma, S.D. **1985**. Sequential Synthesis of Natural Textures. Computer Vision, Graphics, and Image Processing, Vol. 30, pp. 289-315.

Gimel'farb, G. **1999**. Image Textures and Gibbs Random Fields. Kluwer Academic Publishers: Dordrecht, 250 p.

Gousseau, Y., **2002**. Texture synthesis through level sets. In Proc. Texture 2002 workshop, pp. 53-57.

Graham, R., Groetschel, M., and Lovasz, L. (eds.). **1995**. Handbook of Combinatorics. Elsevier, Vol. 2, pp. 1890-1894.

Hertzmann, A., Jacobs, C., Oliver, N., Curless, B., and Salesin D. **2001**. Image Analogies. In Proc. SIGGRAPH, pp. 327-340.

Malik, J., Belongie, S., Leung, T., and Shi, J. **2000**. Contour and Texture Analysis for Image Segmentation. Perceptual Organization for Artificial Vision Systems. Boyer and Sarkar, eds. Kluwer.

Paget, R. **1999**. Nonparametric Markov Random Field Models for Natural Texture Images. PhD Thesis, University of Queensland, February 1999.

Puzicha, J., Hofmann, T., and Buhmann, J. **1999**. Histogram Clustering for Unsupervised Segmentation and Image Retrieval. Pattern Recognition Letters, 20(9), pp. 899-909.

Puzicha, J. and Belongie, S. **2000**. Model-based Halftoning for Color Image Segmentation.

Shi, J. and Malik, J. **1997**. Normalized Cuts and Image Segmentation. In Proc. CVPR, pp. 731-737.

Wei, L.-Y. and Levoy, M. 2000. Fast Texture Synthesis Using Tree-Structured Vector Quantization. In Proc. SIGGRAPH, pp. 479-488.

Zalesny, A. and Van Gool, L. **2001**. A Compact Model for Viewpoint Dependent Texture Synthesis. SMILE 2000, Workshop on 3D Structure from Images, Lecture Notes in Computer Science 2018, M. Pollefeys et al. (Eds.), pp. 124-143.

Zalesny, A., Ferrari, V., Caenen, G., Auf der Maur, D., and Van Gool, L. **2002**. Composite Texture Descriptions. In Proc. ECCV, pp. 180-194.