Image Group Compression using Texture Databases

Matthias Kramm

Munich University of Technology (TUM), Institute for Computer Science, Boltzmannstr. 3, 85748 Garching, Germany

ABSTRACT

An image compression approach capable of exploiting redundancies in groups of images is introduced. The approach is based on image segmentation, texture analysis and texture synthesis. The proposed algorithm extracts textured regions from an image and merges them with similar texture data from other images, in order to take advantage of textural re-occurrences between the images. The texture extraction is done by taking overlapping rectangular texture parameter samples from the input image(s), and using a clustering algorithm to merge them into spatially connected regions, resulting in a polygonal texture map. The textures of that map are henceforth analysed by extracting various features from the texture regions. Using a metric defined on these features, the textures are then merged with entries from a central database, which consists of all the textures in all the images of the image collection, so that for each image, only a polygonal segmentation map and references into this texture database need to be stored. Decoding (decompression) works by extracting the polygonal texture map followed by filling the map regions with patterns generated using texture synthesis based on the texture feature vectors from the database.

Keywords: Karhunen Loeve Compression, KLT, Image Clustering, Image Similarity, Multi-Image Compression

1. INTRODUCTION

When dealing with large scale image archive systems, efficient data compression is crucial for the economic storage of data. Currently, most image compression algorithms only work on a per-picture basis — however most image databases (both private and commercial) contain a large number of similarities between images, especially when a lot of images of the same objects, persons, locations, or made with the same camera, exist. In order to exploit those correlations, it's desirable to apply image compression not only to individual images, but also to groups of images, in order to gain better compression rates by exploiting inter-image redundancies.

Extending image compression to image groups has not attracted much research so far. The only exceptions are the areas of hyperspectral compression [1–3] and, of course, video compression [4], which both handle the special case of compressing highly correlated images of exactly the same size.

Concerning generalized image group compression, we recently researched an algorithm which works by building an eigenimage library for extracting principal component based similarities in groups of images. While the algorithm presented in [5] is quite fast, and manages to merge low-scale redundancy from multiple images, it fails to detect more global scale redundancies (for example, similar image parts which are both translated and scaled, or contain similar textures), and also has the problem of becoming "saturated" quite fast (i.e., the more images in a group, the worse the additional compression rate of the individual images), which limits the size of possible image groups.

We also researched a multi-image PIFS-based fractal based compression method [6], which manages to detect similar image parts, but which suffers from the same problem of not being able to handle image groups beyond a given size, due to the quadratically growing computing time $C \cdot n^2$ when compressing n images.

In this paper, we propose a novel algorithm for image groups, which is based on the so called "second generation" approach [7]. The idea behind this method is to dissect an image into homogeneous regions using a segmentation

Further author information:
Matthias Kramm, eMail: kramm@in.tum.de, Telephone: +49 89 289-18680

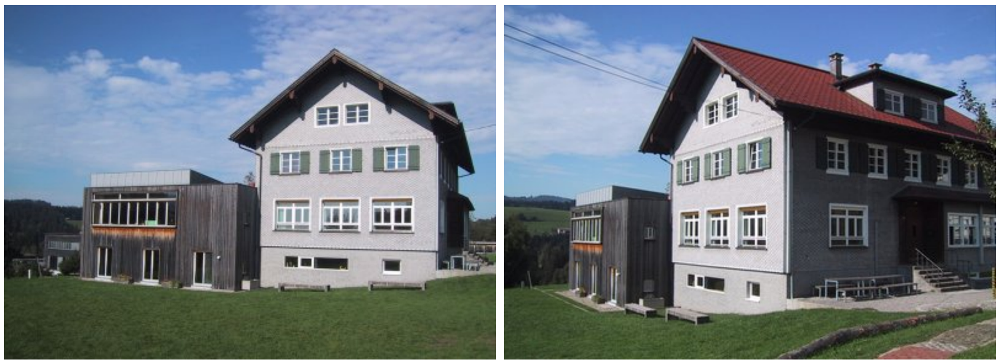

Figure 1. *Two images, with a number of similar textures: The grass, the sky, the different wall types, as well as the forest in the background have almost identical texture characteristics.*

algorithm, and to store the contents of these (textured) regions as well as the segmentation map (or polygon model) separately.

While each of the necessary components (image segmentation, image polygonalization, texture extraction, texture analysis, and texture synthesis) have been, and still are being, thoroughly researched, only few scientific works put a complete compressor model to discussion which actually combines these building blocks into an actual compressor [8] [9] [10] [11].

While this paper does detail such a complete algorithm which can also be used to compress single images, the primary focus is to establish a model for compressing groups of images. This is in the same context as our previous work for extending compression algorithms to work on image groups, in order to exploit inter-image redundancies.

Extending compressions algorithms so that they work on image groups instead of individual images takes advantage of repeatedly occurring elements between images. So while in [5], the statistical features were the particular KLT eigenimages of a number of images, and in [6], redundancies were exploited by finding shifted or scaled image regions (with different contrast and luminance), what we seek to exploit now with the methods in this paper are images which contain highly similar textured regions. (See Fig. 1)

Hence, unlike previous “second generation” compression models, we seek to compute a central texture database which is shared between images and needs to be stored only once, globally. In the following, we will first describe the segmentation approach used for splitting an image into homogeneous textured regions, and afterwards we detail the texture model used for the analysis part of the algorithm, and the synthesis part used for recreating (decompressing) an image.

2. SEGMENTATION

Segmentation and texture analysis have to be closely entwined in this particular application. The segmentation needs to provide the texture analysis algorithm with regions homogeneous enough so that a coherent texture model can be built from them (and also needs to exclude regions for which no such model can be created), and likewise the texture analysis parameters have to be constructed in such a way that the segmentation is able to dissect an image into regions with sufficiently high granularity.

During image segmentation, one is always walking on a fine line between the requirement to take context into approach (by using larger averaging filters and histogram regions), and the requirement that edges between regions should be recognized on the single pixel level. A number of authors have hence tried to combine filter or texton histograms with edge detection algorithms, and feedbacking either one into the other iteratively. [12] [13]

While the methods used in these works work well for visual-based image segmentation, a different approach is needed for texture segmentation. For example, visual edges as such are perfectly valid to appear inside a connected texture patch, as long as the edges are evenly, and randomly, distributed. Analogously, regions of slowly varying color or texture, which in previous works have explicitly been treated as connected [14] need to be segmented into different textured regions if a static texture model is used, in order to preserve lightening and/or perspective properties of the image.

In our segmentation algorithm, we hence took a different direction. Instead of taking contex into account implicitly, by applying filter banks to each pixel neighborhood, and then afterwards removing the "wrong" contexts with an edge detector, we take context into account explicitly, by relying on a model which, for each pixel, specifies the contexts it is part of. For this, we divide the image into a number of rectangular, overlapping $n \times n$ blocks (for our implementation, $n = 8$ was chosen), and, for each patch, measure a number of texture features. The statistics we measure are inspired by our texture model (see section 4). For each $n \times n$ region, we measure the following attributes:

1. Mean, Variance, Skew and Kurtosis of the pixel grey values
2. Complex modulus of the local Fourier Transform of the localized patch (power spectrum)
3. Minimum and Maximum value of the pixel grey values
4. Mean of the U and V components of a YUV color space representation of the image.

We afterwards quantise each of these statistics to the range $0 - 127$ (for implementation purposes, we use a fast SSE2 implementation of k-Means which processes 16 pixels in parallel), and store them as a vector field.

We then use said k-Means clustering algorithm in order to derive the homogeneous image regions. Each resulting cluster corresponds to a texture. Unlike conventional k-Means, however, in the final step we only assign a pixel to a cluster if it's within some minimal distance to the cluster center (6, in our implementation). This is to keep textures homogeneous, i.e., prevent texture features from being too distant from each other at different image positions.

Once the image blocks have been clustered into a discrete number of textures, all that is left is to assign a specific texture to each image pixel. As for each pixel, $n \times n$ blocks will overlap, a single pixel can potentially belong to n^2 different texture regions (one for each block encompassing the pixel, see Fig. 2), one of which has to be chosen. In our implementation, we chose the texture which occurs most often in the surrounding region (or no texture, if also no texture was assigned to the major number of the blocks). Other selection strategies are possible, however. In particular, we noticed that edge boundaries between textured and non-textured regions are approximated somewhat more precisely if we always take the rightmost, bottommost texture in the current pixels neighborhood. However, this also introduced jagged edges, while the strategy of choosing the most-occurring texture produces much smoother edges.

As not all pixels will be near a cluster center, some parts of the image may have no texture assigned at all. These are usually image regions which are too complex to be described with textures, and which will hence be left alone.

3. TEXTURE EXTRACTION

Once the image has been segmented into regions considered to be homogeneous, and synthesizeable by the texture algorithm, the actual texture data can be sampled. In order to extract data usable for texture synthesis from the regions, we need to do a more fine-grained parameter sampling.

As it's inconvenient (and, in some cases, even impossible) to do texture sampling on an irregular region, especially when it comes to analyzing frequency components, we rely on finding a square area within the detected texture, of maximum size (see Fig. 3).

We found that it's possible to do this in $O(w \cdot h \cdot (w + h))$ time, where w and h are the width and height of the texture map, respectively, using a simple algorithm: For each position, we first count the number of texture pixels belonging to the current texture below the current pixel (or zero, if the current pixel itself doesn't belong to it), which requires $O(w \cdot h \cdot h)$ time. Afterwards, for each pixel, we count to the right over the newly created array, stopping once the minimal value encountered so far is lower than the relative x position. Using this approach, it's possible to maximum sampling rectangles for each texture in quadratic time.

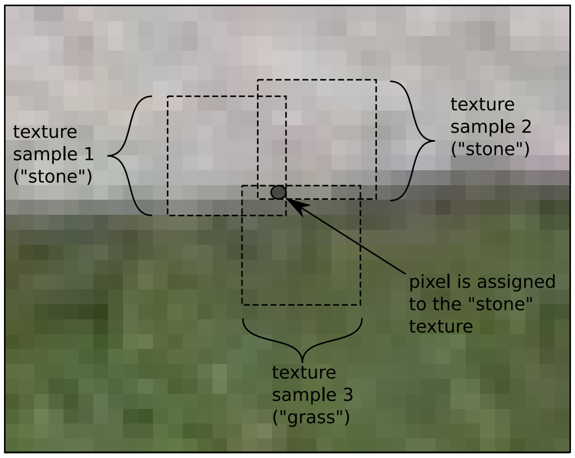

Figure 2. *Assignment of pixels to textures, based on the order of occurrence of surrounding texture samples.*

4. TEXTURE MODEL

In order to analyze and synthesize a quadratic texture patch in a way that's compatible with our compression approach, the texture model needs to have a few particular properties:

Mergeability It needs to be able to generate a "common" texture model from two different textures. This is necessary so that for images sharing a texture, an intermediate texture description can be generated.

Tileability The generated textures need to be tileable, even if the image patches the textures are extracted from are not, in order to be able to fill irregular partitions with them. It should be possible to generate tiles larger than the original image patch (extrapolation), so that larger regions don't look repetitious.

Color The texture model should incorporate color.

Inpainting The synthesis algorithm needs to be able to fill holes in a bitmap with the texture in such a way that the boundary between texture and bitmap is as visually undetectable as possible.

We chose to roll our own model-based algorithm which fulfills these requirements, by loosely adapting the model proposed in [15]. In particular, we define a number of texture features, which are extracted from the different textures, and stored in the database for later use by the texture synthesis part of the decompressor. We don't require any bitmaps of textures, as such, in the database, unlike texture synthesis algorithms like GraphCut [16]. We require only texture statistics, which makes it easier to fulfil the need of Mergeability, as statistics can be easily combined.

As the texture algorithm will only be asked to synthesize textures as complex as what the segmentation algorithm declares to be "homogeneous", we can get along with only a subset of the texture features defined in [15]. We found that for our application, the following features are sufficient:

1. Skew and Kurtosis across scales
2. Mean, Variance, Minimum and Maximum value of the pixel grey values
3. Power spectrum, i.e. the Modulus $|F(x,y)|^2$ of the Fourier Transform $F = \mathcal{F}(I)$ of the greyscale image.
4. Mean of the U and V components of a YUV color space representation of the image.

Using these features, we were able to describe almost all textures of our example image set, see Fig. 4. While adding additional features is possible, and will enhance the number of texture types that can be correctly resampled, it will also enhance the size of the texture in the database, so we choose to instead make due with a segmentation algorithm which detects these regions, see Fig. 5.

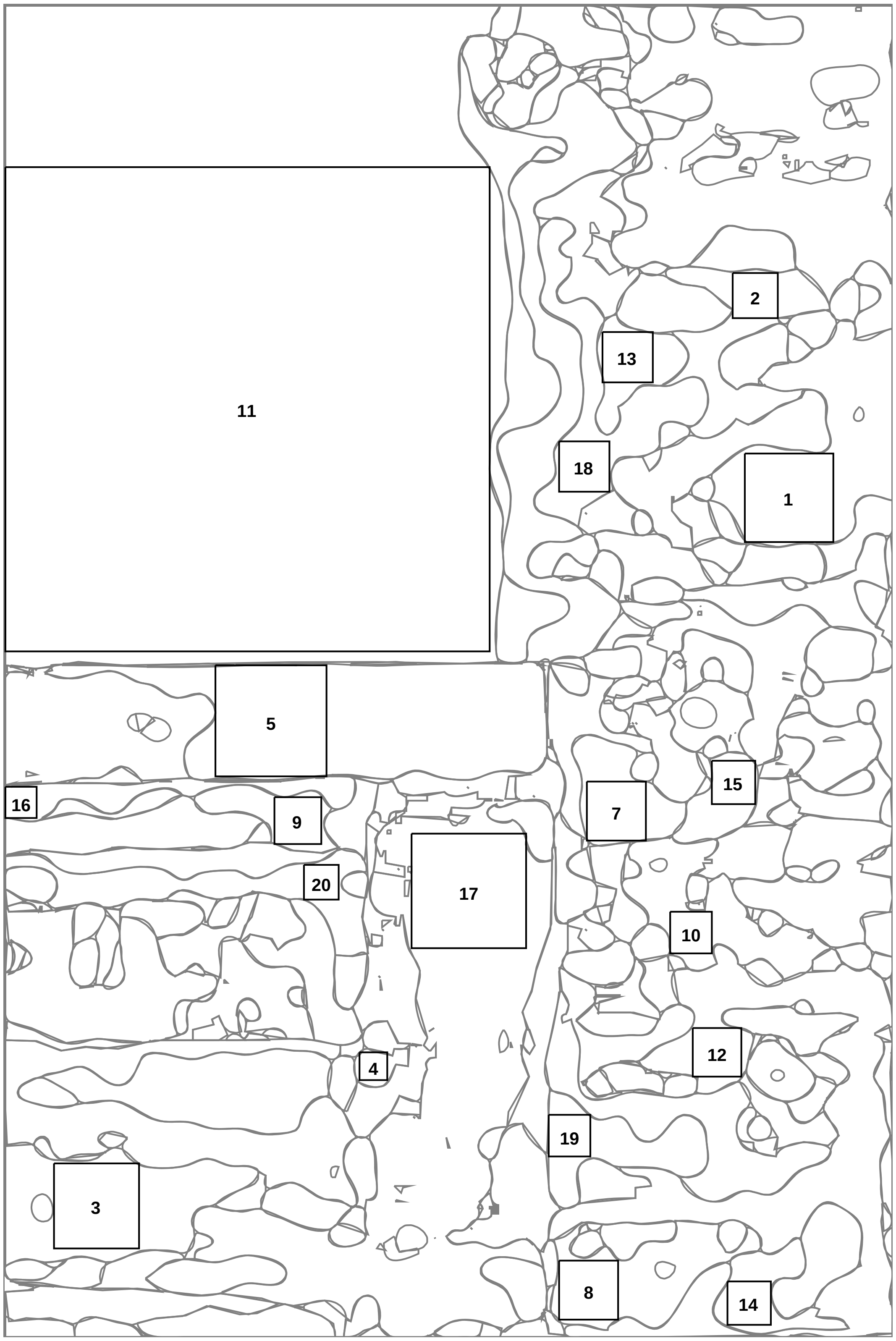

Figure 3. *Extraction of textures from image regions by fitting rectangles of maximum size in each textured region.*

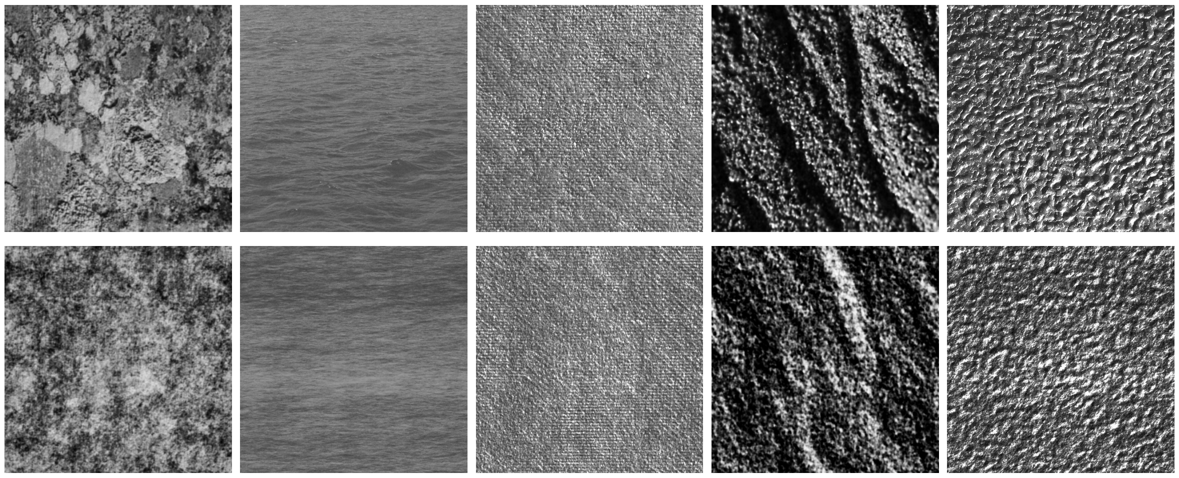

Figure 4. *A number of textures from the VisTex database (top row) which were resynthesized (bottom row) using our algorithm.*

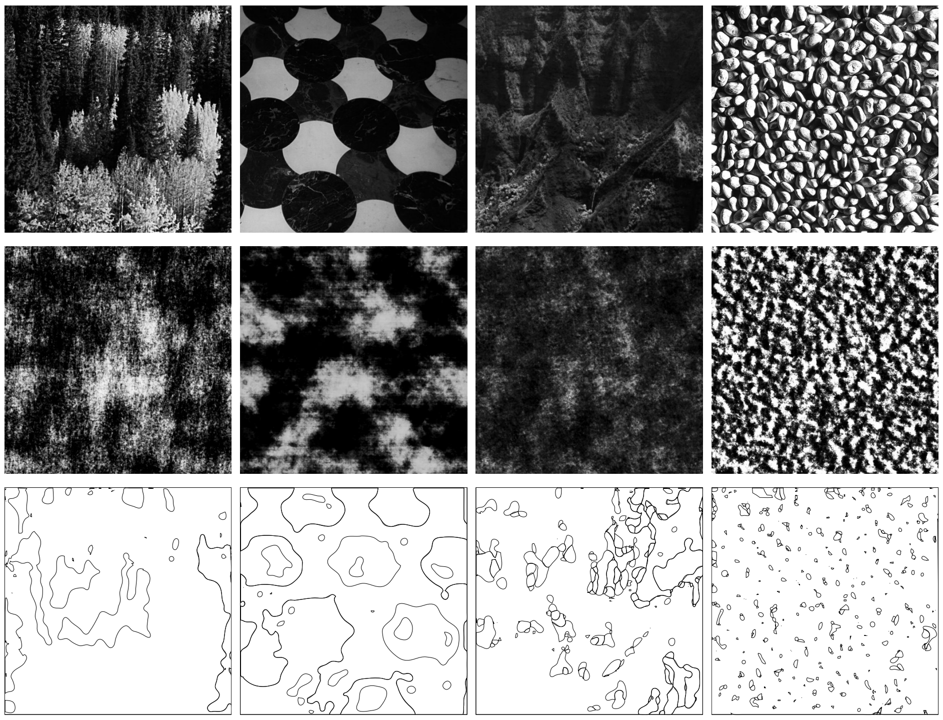

Figure 5. *A number of textures from the VisTex database (top row) where our synthesis algorithm failed (center row). These textures are treated as a number of disjoint textures by the segmentation algorithm. (bottom row)*

5. ENCODING

In order to now encode the texture-modelled images, both the textures as well as the mapping of textures to image pixels needs to be stored. The textures are stored in a central database, while the mapping is stored for each image.

Textures which are to be stored in the database are first checked against the current entries already stored, so that similar textures can be merged. In order to be able to do fast comparisons, we store the k-Means clustering center vector (in other words, the “simplified” texture model used during segmentation) for each texture of the compressed image, against which the k-Means clustering center of the new texture is compared. Textures within some minimal distance to each other are then merged by taking the weighted (by means of texture occurrence) mean of all texture features.

It should be noted that unlike our previous multi-image compression related research [5] [6], with the texture compression approach, it’s easily possible to add images to an existing set of compressed images. The textures of the “new” image will then be merged with existing textures, and “new” textures added to the database.

Encoding the texture is straightforward. We store each texture attribute as a single byte, hence requiring 6 bytes for the marginal stats, 2 bytes for each scale for the 3rd and 4th order moment, and $n \times n$ bytes for the power spectrum (with $n = 64$, for our implementation).

Finding a good encoding for the texture map is more complicated. While previous authors have promoted the use of polygonal-based storage [10], we found that the error introduced to the edges of the image description adds too much visual error. We hence decided to compress the texture map directly, using a pixel prediction algorithm adapted from PNG as well as the lossless zlib compressor. This allows up to 256 different textures per image, which is more than sufficient for most images.

6. DECOMPRESSION

Decompression consists of two parts:

1. Extraction of the polygonalized texture map, to find out which pixel belongs to which texture (or which color it has, if it doesn’t belong to a texture). This is straightforward.
2. Synthesis of all texture regions with the texture models from the database.

For the synthesis part, we create textures analogous to [15] from the model data by iterative projection:
Starting with an image consisting of Gaussian white noise, we adjust one statistical feature at a time by a projection along the gradient of the feature distance. Iterations are repeated until all the features are close enough to their respective value.

We found that it’s possible to soften boundaries between textures by running synthesis steps alternatively: For two textures sharing a boundary, we alternatingly compute synthesis steps for both textures (inside an image region where the image pixels of the second texture are “visible”), which causes both textures to adapt to their common boundary, see Fig. 6.
A disadvantage of this method, however, is that it’s computationally much more intensive than synthesizing a (wrappable) texture of the needed size and pattern-filling the image with it. We hence for further experiments used the “single synthesis” method.

7. RESULTS

In order to test the algorithm, we used a random subset of 100 images from the Berkeley segmentation database, which consists of 300 images of the dimensions 481×381.

We adjusted the texture distances so that each texture was used in average in 2 images. We furthermore only extracted and synthesized textures which were at least of the dimensions 24×24 (or more precisely, the texture patch contained a rectangular region of at least this size). By increasing the former or decreasing the latter parameter, one can produce lower bitrates at a cost of image quality.

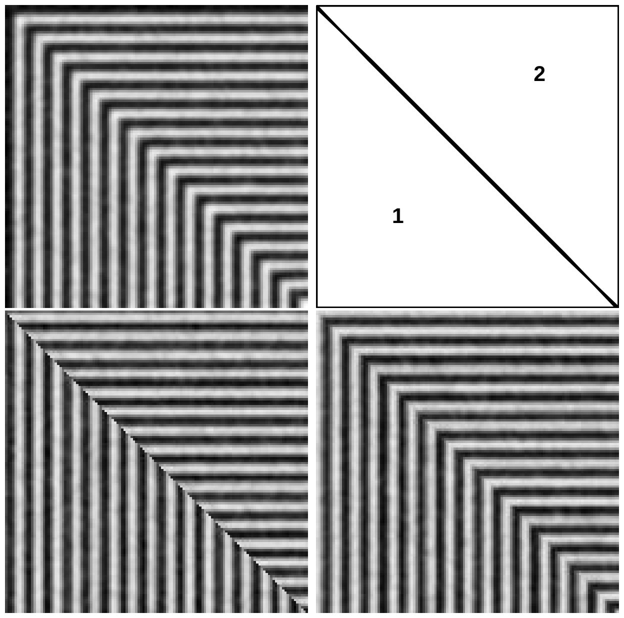

Figure 6. *Motivation for using interleaved texture synthesis: Upper left: Original image. Upper right: Segmentation of original image. Lower left: Reconstruction using independent texture synthesis. Lower right: Reconstruction using interleaved texture synthesis.*

Using these parameters, of the 300 images, roughly 26% were reconstructed in an acceptable way, 39% showed small segmentation or texture artifacts, and 35% were utterly unrecognizable.

Most of the acceptable images were landscape images. Images of objects or persons were generally more distorted. The most problematic artifacts came from segmentation errors, and the introduction of hard edges between different textures in regions of smoothly varying color or texture (like different sky segments), giving some images a "cartoon" look.

Also, images of fabrics or ramifications (like in knotted fisher ropes, or tree branches) seem to be difficult, because they tend to have a lot of locally smooth textures, which however are entwined with each other (e.g., each smaller tree branch in the "small branches" texture is connected to exactly one larger tree branch in the "large branches" texture, etc.), a fact not recognized by the synthesis algorithm.

On the other hand, re-use of textures between images didn't seem to cause any noticeable degradations in image quality, as long as the maximum allowed texture distance (in parameter space) was kept below a certain level.

8. FURTHER RESEARCH

We'd like to refine both our segmentation algorithm and our texture synthesis algorithm, in order to research how much influence either of these has on the final image quality as well as the compression rate.

Also, it will be interesting to study the effects the substituted texture regions have on the human observer — in particular, how sensitive observers are to re-occurrences of identical textures in different images, as well as the substitution of larger image regions against artificial smooth textures, and what this means to the overall impression of image quality.

Furthermore, we also plan to develop a number of other image group compression algorithms using different approaches, also extending into the field of lossless image compression.

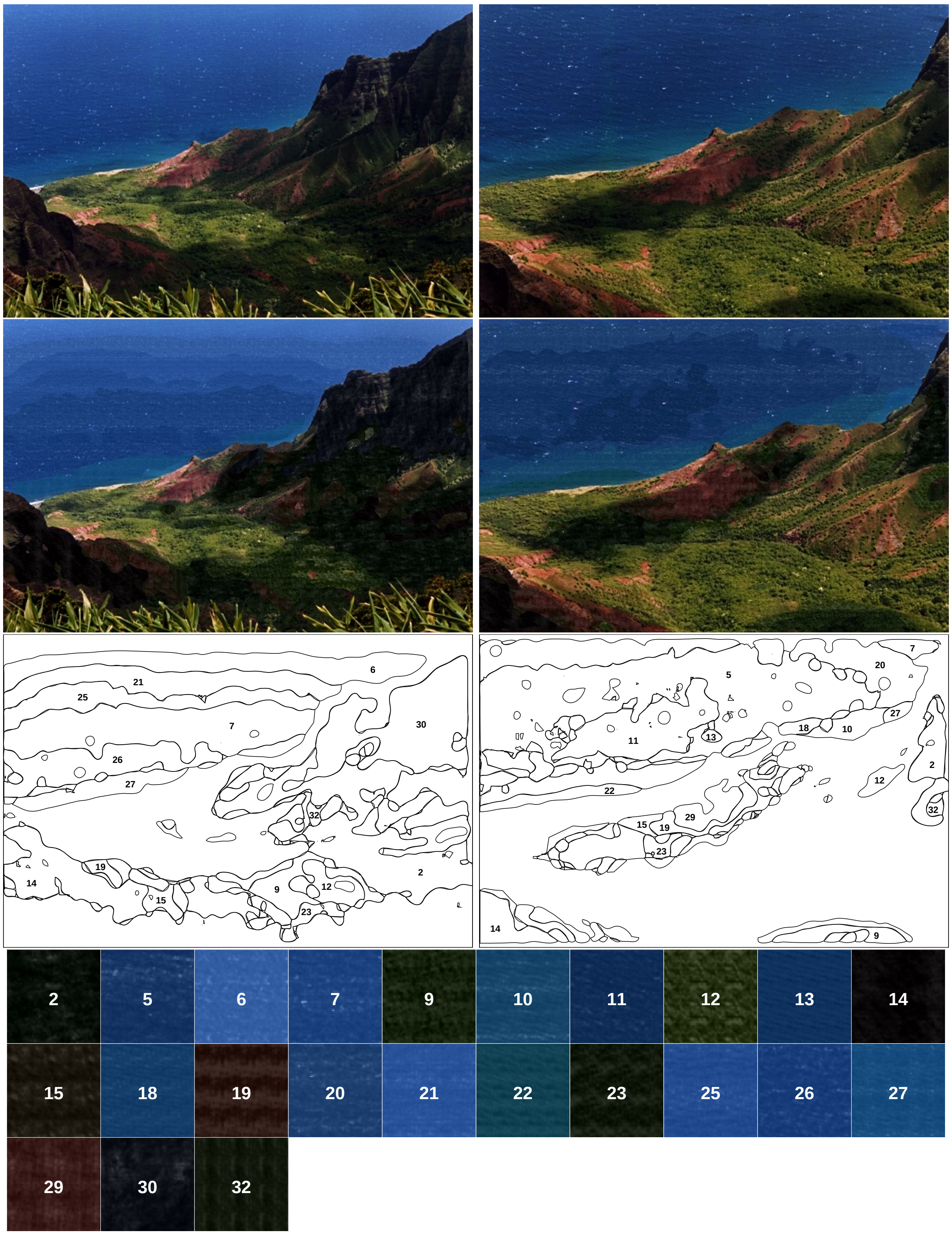

Figure 7. *Example of two images sharing a number of textures. First row: Original images. Second row: "Compressed" images (where textures were substituted), Third row: Mapping of textures to image regions (regions with no number in them were stored as "non-texture"). Fourth row: The texture database used in these images.*

Acknowledgment

This work is part of the IntegraTUM project, which was realised under partial funding by the German Research Foundation (DFG) from 2004 to 2009, and also funded (in the same amount) in the context of TUM's InnovaTUM program, with further substantial contributions by the permanent staff of the Technical University of Munich (TUM) and the Leibniz Supercomputing Centre (LRZ).

REFERENCES

1. J. Saghri, A. Tescher, and J. Reagan, "Practical transform coding of multispectral imagery," *Signal Processing Magazine, IEEE* , pp. 32–43, 2005.
2. J. Lee, "Optimized quadtree for karhunen-loeve transform in multispectral image coding," *Image Processing, IEEE Transactions on* **8**, pp. 453–461, 1999.
3. Q. Du and C.-I. Chang, "Linear mixture analysis-based compression for hyperspectral image analysis," *Geoscience and Remote Sensing, IEEE Transactions on* **42**, pp. 875–891, 2004.
4. L. Torres and E. Delp, "New trends in image and video compression," *Proceedings of the European Signal Processing Conference (EUSIPCO)* , pp. 5–8.
5. M. Kramm, "Compression of image clusters using Karhunen Loeve transformations," in *Electronic Imaging, Human Vision*, **XII**(6492), pp. 101–106, 2007.
6. M. Kramm, "Image Cluster Compression using Partitioned Iterated Function Systems and efficient Inter-Image Similarity Features," in *SITIS 2007*,
7. M. Kunt, A. Ikonomopoulos, and M. Kocher, "Second-generation image-coding techniques," *Proceedings of the IEEE* **73**(4), pp. 549–574, 1985.
8. L. Shen and R. Rangayyan, "A segmentation-based lossless image coding method for high-resolution medical image compression," *Medical Imaging, IEEE Transactions on* **16**(3), pp. 301–307, 1997.
9. A. Jain, "Image data compression: A review," *Proceedings of the IEEE* **69**(3), pp. 349–389, 1981.
10. T. Ryan, L. Sanders, H. Fisher, and A. Iverson, "Image compression by texture modeling in the wavelet domain," *Image Processing, IEEE Transactions on* **5**(1), pp. 26–36, 1996.
11. M. Wakin, J. Romberg, H. Choi, and R. Baraniuk, "Image compression using an efficient edge cartoon + texture model," *dcc* **00**, p. 0043, 2002.
12. J. Malik, S. Belongie, T. Leung, and J. Shi, "Contour and Texture Analysis for Image Segmentation," *International Journal of Computer Vision* **43**(1), pp. 7–27, 2001.
13. J. Chen, T. Pappas, A. Mojsilovic, and B. Rogowitz, "Adaptive image segmentation based on color and texture," *Image Processing. 2002. Proceedings. 2002 International Conference on* **3**, 2002.
14. T. Pappas, "An adaptive clustering algorithm for image segmentation," *Signal Processing, IEEE Transactions on [see also Acoustics, Speech, and Signal Processing, IEEE Transactions on]* **40**(4), pp. 901–914, 1992.
15. J. Portilla and E. P. Simoncelli, "A parametric texture model based on joint statistics of complex wavelet coefficients," *Int. J. Comput. Vision* **40**(1), pp. 49–70, 2000.
16. V. Kwatra, A. Schoedl, I. Essa, G. Turk, and A. Bobick, "Graphcut textures: image and video synthesis using graph cuts," *ACM Transactions on Graphics* **22**(3), pp. 277–286, 2003.