Importance-Driven Texture Encoding Based on Samples

Ying Tang[1] Hongxin Zhang[2] Qing Wang[3] Hujun Bao[4*]

State Key Laboratory of CAD&CG Zhejiang University, Hangzhou, P. R. China, 310027

ABSTRACT

In this paper, we present an importance-driven texture encoding algorithm based on samples. Our algorithm determines a set of samples from source texture based on combined criteria which include compression ratio, visual attention and parameterization distortion. The sample set is used to encode the majority parts of the texture. The remaining regions are then encoded by traditional compression algorithm such as vector quantization. Our method can preserve details of important areas and be extended to dynamic textures. The decoding procedure is performed entirely in programmable graphics hardware, yielding real-time frame rates. Experimental results demonstrate the efficiency and performance of our algorithm.

CR Categories and Subject Descriptors: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism – Texture; I.4.2 [Image Processing and Computer Vision]: Compression (Coding)

Keywords: texture mapping, texture encoding, visual attention, parameterization

1 INTRODUCTION

Textures and texture mapping play significant roles in computer graphics to add visual richness without increasing scene geometric complexity. Most graphics hardware provides dedicated memory cache to store textures for real-time texturing. However, the cache is limited in size and easily filled by high-resolution or large-sized texture images, necessitating the highly efficient compression and on-the-fly decompression of textures.

One pioneer work by Beers *et al*. [1] proposes to compress texture images using Vector Quantization (VQ) algorithm, and accomplishes rendering directly from compressed texture. This approach achieves high compression rate with acceptable visual quality loss by adopting a software implementation of decoding process. Another remarkable compression strategy S3TC (or DXTC in Microsoft's DirectX 3D) [2] widely used in graphics hardware community compresses each 4×4 texel block of one texture independently by encoding each pixel with two-bits index to four representative colors. This technique provides 6 times compression ratio and has been integrated into standard graphics API OpenGL. Pereberin [3] introduces a hierarchical representation and block-wise scheme to support mip-mapping in texture compression. The approach presented in [4] partitions pixels within one block and creates sub-palettes for each partition to choose representative colors per block from an RGB tetrahedron. Ivanov [5] improves S3TC technique to allow sharing colors among multiple blocks, enriching the color choices in each block. Fenney [6] proposes an efficient representation to avoid block artifacts by blending multiple low frequency signals with a low-precision and high-frequency modulation signal.

Aforementioned approaches support appropriate compress ratios and reasonable decompress speed. However, they seldom take account of image contents and dispose each texture part equally. As a result, decompression errors are not distributed according to the content of the handled texture, neglecting the observation that important image region should have less error and vice versa.

Recently, Hertzmann *et al*. [7] present image analogies based on the texture synthesis techniques [8] [9]. It synthesizes a new filtered target image B^* from a pair of source images A, A^* and a target image B, which is relates to B in the same mode as A^* relates to A. By choosing the training pairs automatically, it suggests an interesting extension to image compression. Meanwhile, fractal image compression technique [10] utilizes the self-similarity implied in images to build a set of contractive maps for image compression. However, it needs to explicitly store a sequence of affine transformations and its iterative decompression scheme makes it difficult for time-critical applications.

As mentioned in [1], two major concerns are involved in texture compression, i.e., decoding speed and random access. Standard image compressing techniques such as JPEG are hardly applicable due to their expensive decompression computation cost and the variable rate code against random texel access. A deep view of the texture compression for real-time rendering gives us the following insights for the aspects to be considered compared with that of general image compression. First of all, decompression performance is very crucial, especially for interactive or time-critical applications. Second, texture mapping unavoidably introduces image distortions which are expected to be alleviated. Third, textures usually imply the visual perception of human beings to the appearances of mapped objects, and hence visual attentions have to be taken into account.

These tasks can be resolved by importance-driven texture encoding based on samples. In our algorithm, the handled texture is separated into two parts, one of which is encoded by these samples and the other one is dealt with traditional encoding algorithm. The samples are determined by visually emphasized importance. The contents of samples not only provide information to encode texture image, but also preserve important or interesting regions of the texture that are expected to be kept intact. In addition, compression errors can be adaptively distributed by carefully choosing samples. The compressed data structure produced by our algorithm facilitates real-time decompression in graphics hardware.

The rest of this paper is organized as follows. Section 2 describes the key idea and algorithm pipeline. Importance-driven

[1] ytang@cad.zju.edu.cn
[2] zhx@cad.zju.edu.cn
[3] qwang@cad.zju.edu.cn
[4] bao@cad.zju.edu.cn
[*] Corresponding Author

computation of samples is elaborated in section 3. Next, we present two encoding and decoding approaches for single image and image sequence respectively. Experimental results and discussions are given in section 5. Finally, we present conclusions and future work in section 6.

2 Algorithm Outline

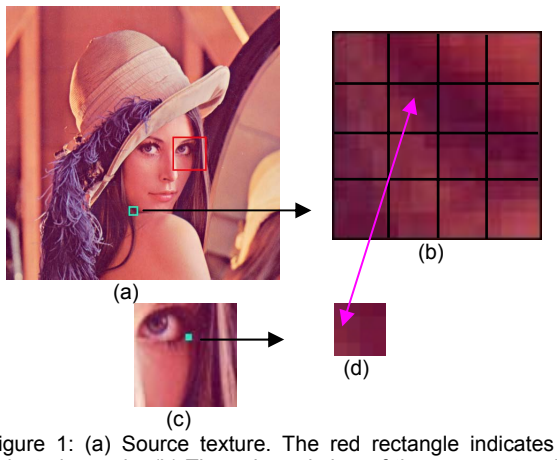


Figure 1: (a) Source texture. The red rectangle indicates the selected sample. (b) The enlarged view of the cyan rectangle in (a). The uniform grid covering the region is illustrated, with each cell being an image block. (c) The selected sample in (a). The cyan block in the sample shows the sample block matching one of the blocks in (b). (d) The enlarged view of the sample block represented by cyan block in the sample, with the magenta arrow indicating the block pair.

To make our explanation clear, we first introduce several basic concepts. For a given source texture, it can be divided into a uniform grid of square blocks, each of which is called an **image block**. The image blocks are non-overlapping. The **sample** is a rectangular region of the texture, in which all overlapping blocks of the same size with image blocks are called **sample blocks**. A series of samples with different sizes constructs a sample set. In the context of texture encoding, all sample blocks in a sample set provide the codebook to encode the handled texture. If an image block matches a sample block under some similarity metric, we encode this image block by mapping it to the sample block. We call them a block pair. Fig. 1 illustrates the basic idea. In the following, if we just use blocks, we mean both image and sample blocks.

Obviously, the more image blocks that can be encoded by one sample set, the better the sample set is. For a given texture and its block subdivision, there exists numerous possible sample sets. The determination of the best sample set can be regarded as a multi-target optimization procedure. On one hand, to encode more image blocks, more samples are required. On the other hand, the size of the sample set should be kept as small as possible to reduce redundancy and achieve higher compression ratio. Further, in order to visually emphasize important features of textures when they are used to paint 3D surfaces, several importance criteria are defined for the samples, yielding an importance-driven texture encoding scheme.

There are two level data representations in our algorithm. The first level, called index map, is defined on a uniform grid covering the texture and provides references to the texture data which are stored in the second level. The second level contains all actual texture data and is called codebook. The index map together with the codebook is stored as small-sized textures and can be used to recover the original texture on-the-fly during texture mapping. The power and flexibility of programmable graphics hardware make the whole decoding procedure real-time, even for a sequence composed of more than one images. Obviously, the size of the index map is determined by the block size. In our experiments, the size of the block is set to be 4×4. This uniform block size is in favor of random access during decoding process. Smaller block size would make the index data bulge, which lowers the compression ratio. Larger block size would make it difficult to find block pairs, which affects the sample's encoding performance.

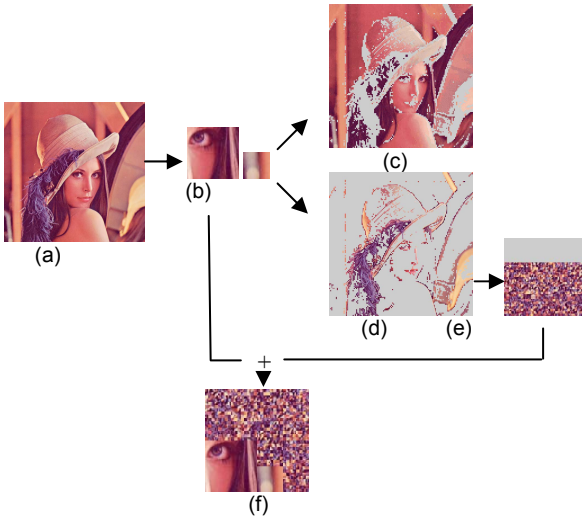


Figure 2: Illustration of importance-driven texure encoding based on samples. (a) Source texture. (b) The sample set consists of two samples. (c) Nearly 78 percent of original image blocks are encoded by the sample set. (d) The regions that can not be encoded by the sample set. (e) The codebook for the image blocks in (d). (f) The reformulated codebook of (b) and (e).

The whole algorithm pipeline is delineated in Fig. 2. Given a texture shown in Figure 2(a), a sample set is automatically extracted as initial codebook by determined criteria described later. In this example, two samples are chosen. The blocks that can be encoded by the sample set are illustrated in Figure 2(c). The blocks that can not be encoded are shown in Figure 2(d). We choose to encode them by the codebook (Figure 2(e)) created by traditional compressing technique. Finally, this codebook and the sample set are combined to reformulate the finial codebook (Figure 2(f)).

3 Importance-Driven Sample Determination

Importance as a rendering parameter has been widely applied in graphics fields. As texture encoding is concerned, we assign each sample an importance. The importance not only implies the performance of the sample set, but also contains important or interesting regions involved in textures. Therefore, any importance criterion can be introduced and the combination of all importance criterions is used to guide the determination of the sample set.

In this section, we introduce our importance criteria to determine the sample set from the source images.

3.1 Sample Reusability

The reusability of a sample describes how many image blocks it can encode compared with the size of the sample. For the given sample size, the sample representing most image blocks within the error tolerance is expected.

The samples are chosen directly from textures. Suppose the resolution of the texture image is $Ns \times Nt$ and the sample size is $Ss \times St$, there are totally $(Ns - Ss) \times (Nt - St)$ sample candidates. Note that, all overlapping blocks in one sample construct a searching space for the block pairs. The computation of the number of image blocks each possible sample can encode is described as follows:

（1）Loop the whole *image blocks* in row-first scanline order；

（2）For current *image block*, search the source image for a set of blocks which are within the error tolerance compared with it and indicate them as potential *sample blocks*；

（3）For each potential *sample block*, find the samples that contain this block and update the number of image blocks that these samples can encode. Repeat.

After all image blocks have been processed, we obtain the number of image blocks each possible sample can encode.

The above three stages imply an algorithm complexity $O(an^2)$, where n is the number of all overlapping blocks in the source image and a is inverse proportional to the block size. In the case of 4×4 block size, a is approximately 1/16. In the second step, we refine and optimize some calculations to further reduce a. Enumerating all-overlapping block in the search space is unnecessary due to the high correlation between two adjacent blocks. Instead, we search blocks which are two pixels apart to reduce computation cost. Since the distance between two blocks is a metric in high-dimensional space that costs much computation time, we pre-compute the average color for each block and compare the average colors before computing the distance between two blocks, eliminating many unqualified blocks. However, the computation time grows very fast with the increase of image resolutions. Under such circumstances, we use the multi-resolution representation of the texture and compute the corresponding results in low-resolution texture. The reusability information of samples obtained in this way is reasonably appropriate to be used to guide sample determination for original high-resolution image. The time consumption for computing the reusability information of a 64×64 sample in a 512×512 texture image is about 180 seconds. By using the searching approaches introduced in [11] [12], further acceleration can be achieved.

The size of samples is specified by the users. The users determine the sample size according to expected compression ratio which is closed related to the size of final codebook. For example, if the size of the codebook is 128×128, usually the resolution of the largest sample is 64×64. The sum of samples should not exceed 75 percent of the codebook. If multiple samples are to be obtained, we choose the one with the largest number of blocks covered by it. For the left blocks, we compute the second sample in the same mode. The determination procedure is continued till the sample number achieves some threshold. The error tolerance is set to be within 0.1 times the expected RMS error. The default expected RMS error is set to be 10.0 in our experiments. The user can adjust this value interactively.

Fig. 2 demonstrates two samples and corresponding encoded result by two samples which covers a majority part of original image. The sample set includes many texture features such as eyes, skin and hair.

3.2 Saliency-based Visual Attention

Human eyes always pay more attention to the parts of the image that are interesting and noticeable. It is reasonable that the important regions are compressed with low errors while the errors for the unimportant regions can be relatively high by adaptively compressing different parts in the image. Thus, the determined samples should extract the important information and keep them intact during encoding.

Saliency-based visual attention is a psychophysical concept and is significantly influenced by two general processes, namely, bottom-up (scene-dependent) and top-down (task-driven) controls. The bottom-up process is purely stimulus driven while the top-down process is volition-controlled and task-driven. For more general and automatic computation of visually important area, we adopt the bottom-up model proposed in [13]. This model is built on a biologically plausible architecture proposed in [14] and [15]. In this model, the input image is decomposed into feature maps for visually important channels such as intensity, color and orientation. Each feature map is computed by a set of linear center-surround operations akin to visual receptive fields which triggers response to changes between the center of the field and its surroundings. The center-surround effect makes the visual system particularly well-suited to detecting features which stands out from their surround. The feature maps are computed at multiple spatial scales and combined into a single topographical saliency map that quantifies visual attention.

Figure 3: An input image and its saliency map. The bright area corresponds to important area in visual attention.

Fig. 3 shows an image and its saliency map. The bright regions in the saliency map imply more visually important areas. In our experiments, it works fairly well for real world scenes, such as landscape and buildings. However, for the images incurring strong subject feelings and understanding, like portraits, the important regions computed by this model are not matched very well with our observations. This is because the model adopted here is completely bottom-up without considering any subjective guidance. In these situations, we propose to interactively paint the important region with brighter pixel intensities.

3.3 Distortion of Texture Mapping

Texture mapping normally introduces distortions with non-uniform sampling of texture images arising from object parameterization. Therefore, we introduce the mapping distortion as another visual importance for the sample determination.

The parameterization S of the surface maps a set of points in R^2 to a set of points in R^3 , which is expressed as $S(u,v) = (x(u,v), y(u,v), z(u,v))$, where (u,v) are points of R^2 and (x,y,z) are points in R^3 . The mapping distortion is

measured using the 3×2 Jacobian Matrix $[\partial S/\partial u, \partial S/\partial v]$ [16]. The largest and smallest singular values of the Jocabian represent the maximal and minimal parameterization distortions respectively. The surface defined by the polygonal mesh is piecewise linear. Hence, the parameterization of such surface is also piecewise linear in the texture domain. For each polygon in texture domain, we compute one set of singular values and obtain the singular values for each vertex by averaging values of incident faces. The piecewise continuous field in the texture domain is calculated by bilinear interpolating vertex values and is output as a 2D gray image where the pixel intensities corresponds to singular values. We call it the distortion map.

Maximum singular value of the Jocabian is adopted here because it reflects the largest stretch when mapping unit-length vectors from the texture domain to the surface. Larger value means the texture image is more stretched during parameterization. Such regions are expected to be enlarged when mapped to 3D surfaces and the compression errors within them are to be magnified. As a result, the texture parts with high stretch values in the distortion map should have low decompression errors.

3.4 Synthesis of Importance Maps

All consideration aforementioned can be represented by 2D maps. For visual attention and parameterization distortion, their importance information is output as saliency map and distortion maps respectively. The reusability information of sample can also be formulated by a 2D usability map by making the pixel intensity proportional to the image block number encoded by the sample. Finally, we get the guidance map by combining these maps with normalized weights. We tune the coefficients to balance the influences of these maps or to emphasize a specific map. The final map is used to guide the sample determination. Given the sample size, we choose the sample that has the greatest intensity sum.

4 Encoding and Decoding for Textures

In this section, we show how to encode textures with samples and decompress the encoded texture in programmable graphics hardware.

With the sample set, we can encode the source texture with the method described in section 2. Note that, not all image blocks can be encoded by the samples determined by the guidance map. For example, the lip and hat decorations in Fig. 3 can not be encoded by the sample set. To encode these regions, we propose two different approaches for single image and image sequence separately.

4.1 Vector Quantization Algorithm for Single Image

In order to optimally compress the left blocks and make full exploit of graphics hardware, we choose vector quantization (VQ) algorithm [17] to compress them for single image. VQ algorithm is an iterative clustering algorithm and yields a locally optimal codebook for a given set of blocks. The determined sample set and VQ codewords are stored together as the final codebook. Its size can be specified by the user in advance. The length of the VQ codewords is given by subtracting the samples from the codebook.

4.2 Incremental Encoding for Image Sequence

We further apply our compression method to image sequences composed of similar images. Image sequences are useful in sprite animation or stitching different textures onto a 3D model. It is not desirable to specify the size of the codebook in advance for image sequences. We prefer to update the codebook dynamically and incrementally for each new added image. When processing images in the sequences, this method incrementally expands the codebook for left blocks that can not be encoded by samples.

The method is quite simple. We first create a new empty codebook for the image blocks that can not been encoded by the sample set. The first unhandled block is directly added to the codebook. We then compare each of the unhandled blocks with the codewords in the codebook. For the block whose distance to some codeword is within the error tolerance, this codeword is used to represent it. Otherwise, this block is added to the codebook as a new codeword. The codebook computed in this way is combined with the sample set to form the final codebook, i.e., a small sized texture.

The incremental encoding approach is especially suitable for dynamically updated image sequence. When a new similar image is added, the algorithm adds only the information that can not be represented by existent codebook. Since the images in the sequence are similar, they can reuse the codebook with high probability and the add-in contents are controllable. On the other side, VQ algorithm is quite fixed since it pre-specifies the codebook size and has to perform iterative computation to re-converge to the codebook when new image is added. Compared with it, our incremental method is simple, efficient and flexible for dynamic image sequence.

4.3 Rendering from Compressed Textures

The compressed data including index map and codebook are stored in the texture memory. We implement the decoder in programmable graphics hardware. The decoder makes use of dependent texture lookup, through which the texture coordinates of the second texture access is derived from the first texture lookup. In our decoder, the data fetched from the index map provides the position where the texture data is to be fetched from codebook. Subsequently, the actual texture data are read from codebook and are used to render the corresponding parts. The whole simple decoding process achieves real–time frame rates, which has an advantage over slow iterative decompression scheme involved in fractal image compression that makes it difficult to be applied to time-critical applications.

5 Experimental Results and Discussions

In this section, we present the experimental results produced by proposed texture encoding algorithm for single images and image sequences. The decoder is implemented in Nvidia GeForce FX 5600 and is written in CG for openGL API and written in PS shder 2.0 for D3D API.

5.1 Texture Encoding for Single Images

For still images, the size of the codebook needs to be specified in advance. We assign different images with different codebook sizes and different codebook sizes lead to different compression ratios. For the examples in Fig. 4, Fig. 5 and Fig. 6, the size of the codebook is set to be one sixteenth of that of input image. The

size of the codebook is further reduced to be one sixty-forth of that of input image in Fig. 7. As for the size of index map, remember that the input image is divided into uniform grid with each cell being 4×4 pixels, so index map has one sixteenth of pixels in the input image. We limit the length of each pixel in index map to two bytes with each byte recording horizontal/vertical position of corresponding representing block in the codebook. Thus the largest size of the codebook is limited to be $2^8 \times 2^8$ and we find this is enough in our applications. The images to be compressed are RGB images with each pixel being three bytes long. So the overall compression ratio of input RGB image to the sum of codebook and index map is $1/(n + (1/16) \times (2/3))$, where n is the size of the codebook. When n is 1/16 in Fig. 4, Fig. 5 and Fig. 6, the compression ration is 9.6. The ratio is enhanced to 17.1 when n is 1/64 in Fig. 7.

In Fig. 4 we present the process of selecting one sample from the tower image of resolution 512×512. The codebook and the decompression result are also shown in this figure. The sample is set to be 64×64 pixels and is selected with the assistance of usability map which visualizes reusability information and the importance map which indicates the visual attention information. The RMS error between the decompressed and uncompressed images is 6.7. Without selecting samples, the result generated by pure VQ with the same codebook size has the RMS error of 6.5. Although our RMS error is a little higher than VQ's, we keep the top of tower which is the visual attention in the image losslessly encoded. So our result is more satisfying when taking visual importance into consideration.

For the girl face image of resolution 512×512 in Fig. 5, we present and compare the compression results of our method and VQ. In order to be fair, both codebook sizes are set to be the same (Fig. 5(d)). In VQ all blocks of original image are processed equally, so we observe some undesired compression artifacts in some important regions, such as the lip and eyes (Fig. 5(f)). Using our method, samples in such important regions are selected and packed in the codebook (Fig. 5(d)). So such regions are visual-pleasingly preserved and the errors are distributed to other unimportant regions, like the hair part (Fig. 5(e)). Although the overall RMS error of our result is a little higher compared with VQ's, the decompression result produced with our method looks better than VQ's for human visual effect.

In Fig. 6 we present 3D rendering results on the model of a cow head from the compressed data generated by our method and VQ separately. The texture image is of resolution 512 × 512. According to the parameterization distortion, the nose part of the tiger image is stretched highly and so needs to be preserved well. However, without such consideration, the blockwise artifacts are evident in the nose part when mapping the data produced by VQ (Fig. 6(d)) to the model. With our method the nose part is cut as a sample, so it is preserved well and the rendering result is more satisfying (Fig. 6(f)). The average RMS error across 60 frames animation between our result and uncompressed texture is 6.5 and such average RMS error between VQ and uncompressed texture is 6.4. The FPS (frame per second) of rendering speed is 40.0 when rendering from compressed data with our decoder.

In Fig. 7 we present another 3D result of mapping a large mountain image of resolution 2048×2048 on a terrain model. The visual quality of the rendering result from the encoded data is still satisfying with the reduced codebook size ratio. The average RMS error between our result and original rendering result from uncompressed texture is 11.0. The decoding is still real time with FPS being 38.0.

5.2 Encoding for Dynamic Textures

We implement a novel incremental encoding algorithm for animated image sequences. This method differs from VQ algorithm by dynamically updating its codebook with new input images from the sequence. This method does not produce separate codebook for each image. All images share the same codebook with their index maps.

We apply this method to the DolphinVS sample released by Microsoft DirectX 9.0 SDK. This sample shows an underwater scene with caustic effects on the dolphin and sea floor which is produced by an animated set of 32 textures whose resolutions are all 64×64. These 32 textures are compressed with sample-based incremental encoding method. In order to render from the compressed data, we modify the original texture sampling function by a pixel shader which decodes the texel value with dependent texture lookup.

The first texture is chosen as the sample and for the image blocks within the set of textures that can not be represented by this sample, they are added into the codebook. The sample and add-in blocks finally form the codebook whose resolution is 64×96 pixels. The rendering results of one frame are listed in Fig. 8. The compression rate is 11.3 and the average RMS error across 50 frames animation is 6.0. The FPS is 80 when rendering from the compressed data.

6 Conclusion and Future Work

In this paper we present a new algorithm for encoding textures with selected samples. It incorporates different ideas including texture synthesis, texture compression, visual attention and parameterizaion. Experimental results show that our approach can efficiently compress textures and render them from compressed data with high quality using programmable graphics hardware.

The proposed algorithm is flexible and scalable. More considerations can be incorporated to make more appropriate sample determination. We have performed some experiments on some aspects in visual attention and parameterization. In the future, we want to explore more sophisticated importance and utilize them to guide the compression process. One possible way is that the perceptual characteristics of human eyes can be taken into account. In addition, with graphics hardware that can support dependent texture lookup more than two levels, we can take multilevel compression into account to further compress the relatively unimportant regions in the sample set.

7 Acknowledgements

The project is supported in part by the 973 program of China (Grant No. 2002CB312102), National Natural Science Foundation of China (Grant No. 60021201) and Research Fund for Doctoral Program of Higher Education (Grant No.20030335083). Many thanks to Wei Chen for helping refine the structure and expression of this paper. We also thank the reviewers for their suggestions for improvements to this paper.

REFERERNCES

[1] Andrew C. Beers, Maneesh Agrawala, and Navin Chadda. Rendering From Compressed Textures. In *Computer Graphics (SIGGRAPH 1996 Conf. Proc.)*, Pages 373-378, 1996.

[2] Konstantine Iourcha, Krishna Nayak, and Zhou Hong. System and Method for Fixed-Rate Block-Based Image Compression with Inferred Pixel Values, US Patent 5,956,431.

[3] Anton V. Pereberin. Hierarchical Approach for Texture Compression. In *Proceedings of GraphiCon*, pages 195-199, 1999.

[4] L. Levkovich-Maslyuk, P.G.Kalyuzhny, and A. Zhirkov. Texture Compression with Adaptive Block Partitions. In *Proceedings of the eighth ACM international conference on Multimedia*, pages 401-403, 2000.

[5] Denis Ivanov and Yevgeniy Kuzmin. Color Distribution-A New Approach to Texture Compression. *Computer Graphics Forum 19(3)*, pages 283-289, 2000.

[6] Simon Fenney. Texture Compression Using low-frequency signal modulation. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 84-91, 2003.

[7] Aaron Hertzmann, Charles E. Jacobs, Nuria Oliver, Brian Curless, David H. Salesin. Image Analogies. In *Computer Graphics (SIGGRAPH 2001 Conf. Proc.)*, pages 327－340, 2001.

[8] L. Y. Wei and M. Levoy. Fast texture synthesis using tree structured vector quantization. In *Computer Graphics (SIGGRAPH 2000 Conf. Proc.)* pages 479-488, 2000.

[9] Michael Ashikhmin. Synthesizing Natural Textures. *2001 ACM Symposium on Interactive 3D Graphics*, pages 217-226, 2001.

[10] Yuval Fisher. *Fractal Image Compression: Theory and Application to Digital Images*. Springer Verlag, New York, 1995.

[11] B. Wohlberg and G. de Jager. A Review of the Fractal Image Coding Literature, In *IEEE Transactions on Image Processing, Volume 8, No. 12,* pages 1716-1729, 1999.

[12] Chong Sze Tong and Minghong Pi. Fast Fractal Image Encoding Based on Adaptive Search, In *IEEE Transactions on Image Processing, Volume 10, No. 9*, 2001.

[13] Laurent Itti, Christof Koch and Ernst Niebur. A Model of Saliency-Based Visual Attention for Rapid Scene Analysis, In *IEEE Transactions on Pattern Analysis and Machine Intelligence Volume 20, Issue 11*, pages 1254-1259, 1998.

[14] Christof Koch and S. Ullman. Shifts in Selective Visual Attention: Towards the Underlying Neural Circuitry. In *Human Neurobiology, Volume 4*, pages 219-227, 1985.

[15] Niebur, E. and Koch, C. Computational Architectures for Attention. In Parasuraman, R. editor, *The Attentive Brain*, pages 164-186, MIT Press, Cambridge, MA, 1998.

[16] Sander, P., Snyder, J., Gortler, S., and Hoppe, H. Texture mapping pregressive meshes. In *Computer Graphics (SIGGRAPH 2001 Conf. Proc.)*, pages.409-416, 2001.

[17] A. Gersho and R. M. Gray. *Vector Quantization and Signal Compression*. Kluwer Academic Publisher, 1991.

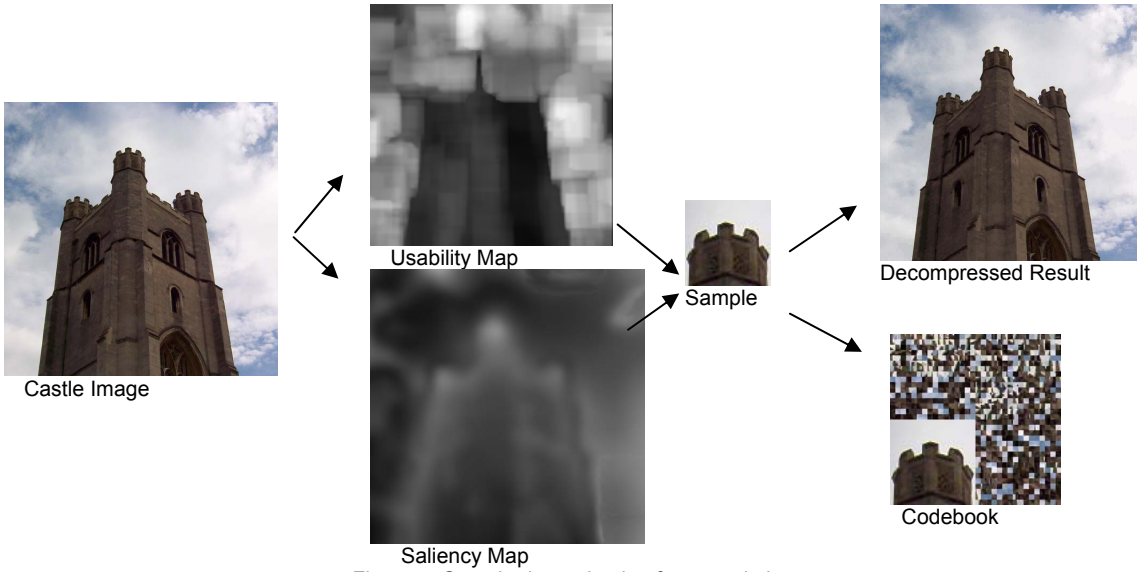


Figure 4: Sample determination for a castle image.

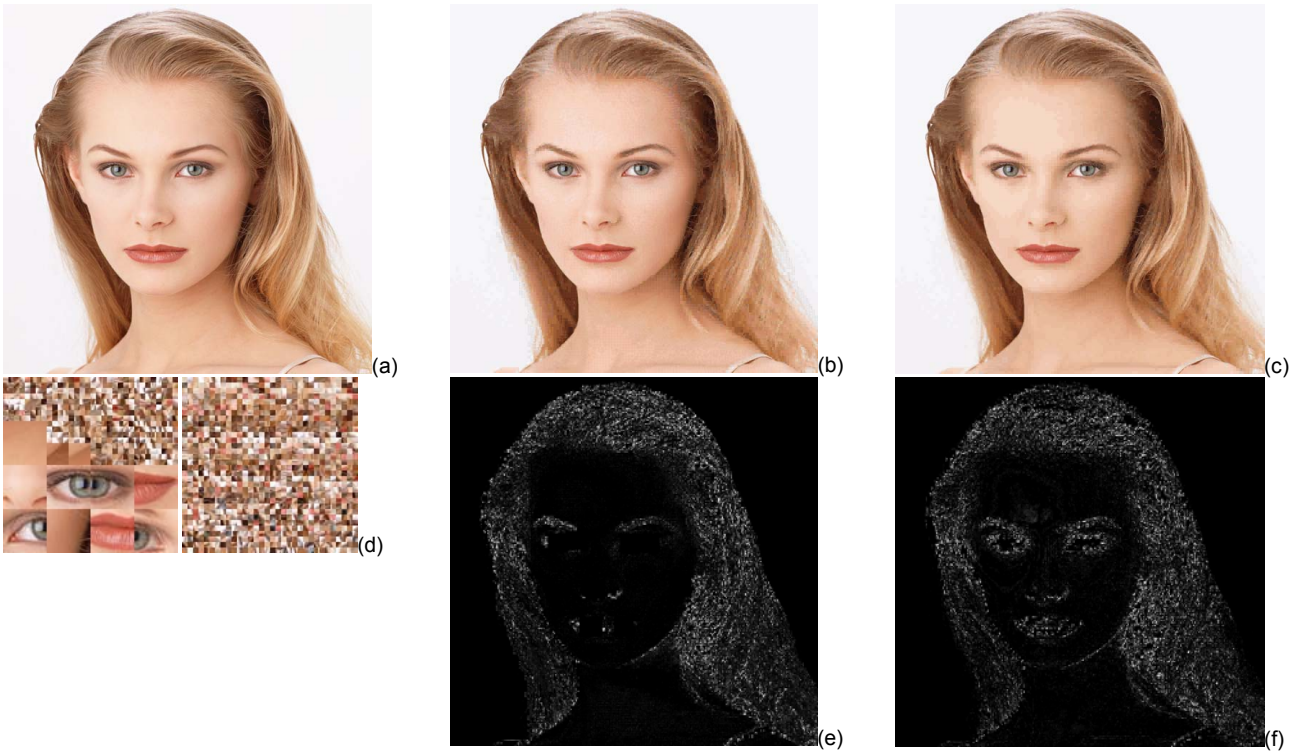


Figure 5: Comparison of the VQ algorithm and our method. (a) Source image. (b) Decompressed result by our method. The RMS error between this image and source image is 9.8. (c) Decompressed result by VQ algorithm. The RMS error between this image and source image is 9.2. (d) The codebook used in our method (the left one) and the codebook of VQ (the right one). (e) The difference map between our result and source image. (f) The difference map between the result of VQ algorithm and source image.

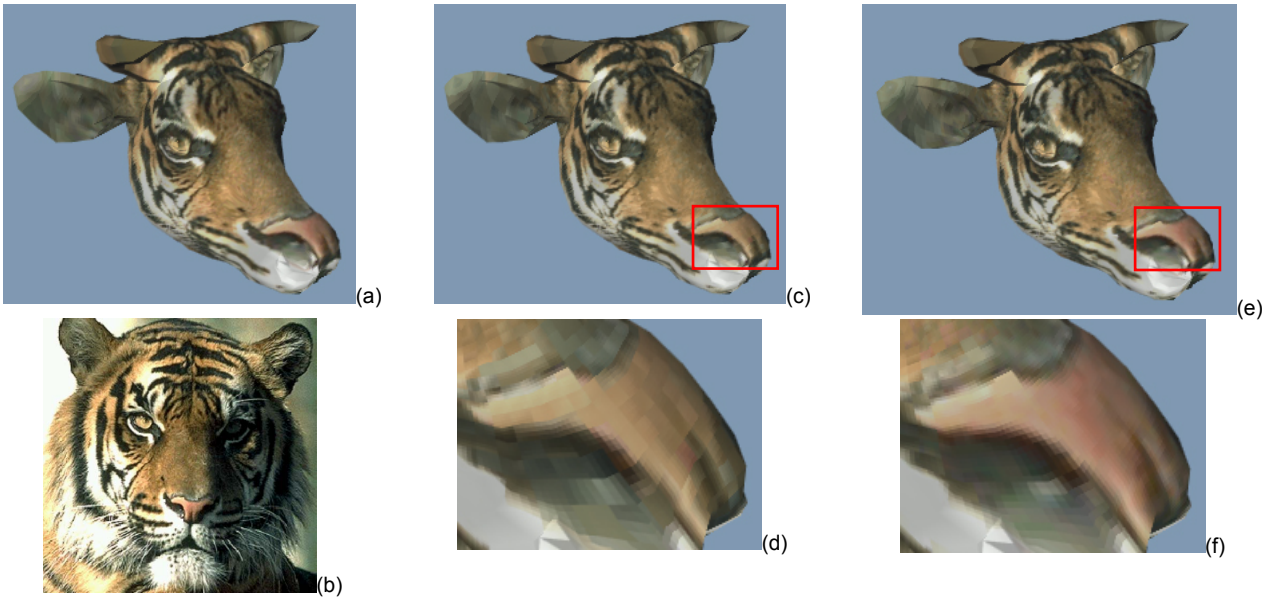
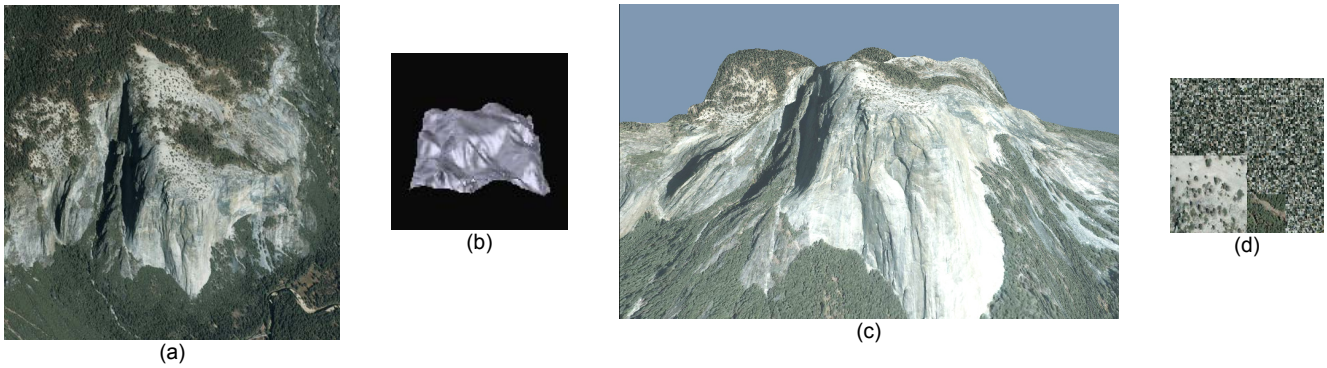
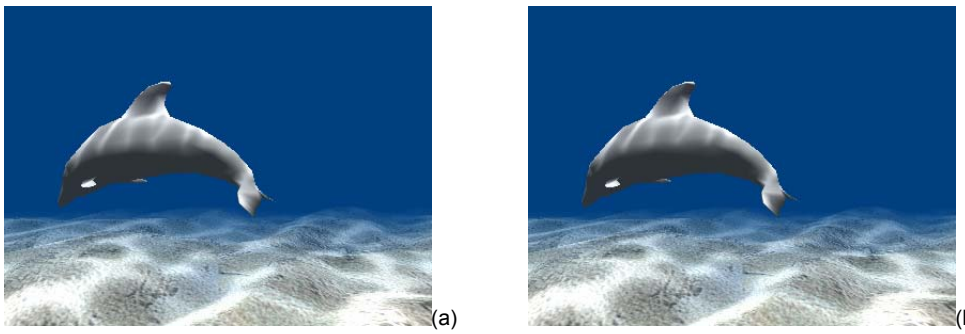


Figure 6: Texture mapping to a cow-head model. (a) Texture mapping of the source texture of (b). (b) Texture image. (c) Texture mapping of compressed data produced by VQ algorithm. (c) The enlarged part of the red region in (c). (e) Texture mapping of compressed data produced by our method. (f) The enlarged part the red region in (e).



(a) (b) (c) (d)

Figure 7: Texture mapping of a mountain image on a 3D terrain model. (a) The mountain image. (b) The 3D terrain model. (c) The rendering result from the compressed data. (d) The corresponding codebook used in our method.



(a) (b)

Figure 8: One frame chosen from the animation clip. (a) Result without compression. (b) Result with our algorithm.