Hierarchical texture compression

Jerzy Stachera
Institute of Computer Science
Warsaw University of Technology
Ul. Nowowiejska 15/19
00-665 Warszawa, POLAND
J.Stachera@ii.pw.edu.pl

Przemyslaw Rokita
Institute of Computer Science
Warsaw University of Technology
Ul. Nowowiejska 15/19
00-665 Warszawa, POLAND
P.Rokita@ii.pw.edu.pl

ABSTRACT

Texture mapping is a technique for adding visual realism to the computer generated images. As the level of realism increases with the number and the resolution of textures, we are faced with the problem of limited texture memory space. Moreover, in order to alleviate the aliasing artefacts many graphics systems use the mip-mapping technique which needs to store additionally the texture pyramid. We propose an algorithm for texture compression which is characterized by low computational complexity, random access to compressed data and the hierarchical texture representation. The proposed hierarchical texture compression algorithm (HiTC) is based on a block-wise approach, where each block is subject to the modified fractal compression method and is partly represented by Laplacian pyramid. This allows us to incorporate the mip-map structure into the compressed texture and perfectly suits for real-time computer graphics applications.

Keywords
Texture compression, fractal compression, Laplacian pyramid, mip-mapping.

1. INTRODUCTION

The computer generated scenery composed of wire-frame models was the first step into the virtual world. The models could only reflect the 3D structure of objects and were devoid of any other form of visual information. That was changed by the introduction of texture mapping techniques. Textures in their basic form are images which are applied upon 3D wire-frame models. When mapped onto an object surface, the color of the object surface is modified by the corresponding color from the texture. In general, the process of texture mapping takes several steps. Since textures are represented by an array of discrete samples, a continuous image must first be reconstructed from these samples. In the next step, the image must be warped to match any distortion and filtered to remove high-frequency aliasing artefacts. In most cases, the filtering is done by mip-mapping [Wil83], which introduces additional memory cost in the form of texture pyramid.

Current applications of textures are much wider and textures are used to control most of the visual parameters of the object surfaces such as transparency, reflectivity, bumpiness, roughness. This greatly increases the realism of the virtual objects and pushes the hardware resources to the limits. To simulate the real world at the interactive frame rate, it is necessary to have fast and random access to a large number of high resolution detailed textures. Thus, the bandwidth and storage requirement for textures introduces the need to use more efficient texture compression methods.

2. PROBLEM DEFINITION

Although a texture can be regarded as a digital image, most of classical image compression methods cannot be applied to textures. There is a strong need for efficient, highly specialized texture compression algorithms.

According to Beers [Bee96] and our findings when designing a texture compression algorithm the following aspects must be taken into consideration:

Decoding speed. It is the essential feature which allows for rendering directly from the compressed textures. As texture compression is mostly used in real time computer graphics the decompression algorithm must be designed to allow high frame rates.

Random access. Since the mapping process introduces discontinuous texture access in the texture space, it is difficult to know in advance how the texture will be accessed. Thus, the methods which may be considered for fast random access are based on fixed length codes.

Hierarchical representation. To deal efficiently with level of detail and mip-map texture representations this requirement must be fulfilled. Hierarchical structure allows for rendering directly from the compressed texture represented on number of different resolution levels.

Compression rate and visual quality. The difference between textures and images is that the images are viewed on their own and they are presented in the static content, while the textures are the part of the scene which usually changes dynamically. Thus, the loss of information in texture compression is more acceptable and the compression ratio is more important issue.

Encoding speed. Texture compression is an asymmetric process, in which the decompression speed is crucial and the encoding speed is useful but not essential.

3. PREVIOUS WORK

All existing texture compression algorithms can be divided into three major groups: block truncation coding and local palettes, vector quantization, transform coding.

3.1 Block truncation coding and local palettes.

The block truncation coding (BTC) was introduced for image compression by Delph and Mitchel [Del79]. The method represents image 4×4 block by two 8-bit gray scale values and index map. Each pixel in the block references one of the gray scale values by 2 bit index from the index map. This corresponds to 2 bits per pixel (bpp) compression. Although, the primary application of BTC was not the texture compression, many other proposed texture compression methods are based on it.

An extension of BTC to represent color images, called color cell compression (CCC) was proposed by Campbell [Cam86]. The two 8-bit values were used for storing the index into a color palette. That representation allowed compressing the color images to 2bpp. Even though, the additional cost of storing a color palette for every texture and indirect data access requiring two memory transactions were prohibitive for some real time application, it was suggested by the Knittel to implement it in the texturing hardware [Kni96]. In terms of image quality both BTC and CCC are characterized by the block effect. This effect is a result of two contrary factors, namely independent block compression and limited number of colors used for the block representation.

The further extension of those two described methods was the S3TC texture compression method introduced by Iourcha [Iou99]. The S3TC represents a 4×4 block by four 16 bits (RGB565) color values and the index map with 2 bits per index. Two base colors are stored explicitly in the compressed block and the others are linearly interpolated from those two during the decompression process. Thus, the final size of the block is equal to 64 bits which gives 4bpp. The FXT1 texture compression method developed by 3DFX added a few more modes to S3TC [3DF99]. The first mode is the same as S3TC and the other compress 4×8 blocks with local palette with four and eight colors interpolated depending on the mode from two or three base colors. Another S3TC like method was a part of POOMA texturing system for low-cost devices [Ake03]. It represented a 3×2 block by two base colors and used block overlapping instead of a texture caching to reduce the memory access.

The texture quality of S3TC based methods is significantly better than CCC and BTC, but it still suffers from the block effect partly introduced by linear interpolation of colors. A number of solutions were proposed to tackle that problem. The color distribution approach proposed by Ivanov and Kuzmin allowed to share colors between neighbour blocks [IVA00], thus representing a block by a larger number of unique colors. Levkovich-Maslyuk et al. allow colors to be chosen from an RGB tetrahedron, and partitioned the block into sub-pallets for better approximation of block original color distribution [Lev00].

Completely different approach was taken by Fenney [Fen03]. On the basis of the fact that low-pass filtered signals are often a good approximations of original signal, his method used two low resolution images and full-resolution low precision modulation signal to represent a texture. Another approach was proposed by Ström and Akenine-Möller in PACKMAN system and its extension iPACKMAN [Str04][Str05]. Proposed methods represent a block by a single base color and an index map with indices into a codebook which values modulate the pixel luminance.

Although texture compression methods based on BTC are currently the mainstream in computer graphics hardware, they do not address the problem of hierarchical representation. Thus, the main features that characterize them, namely - decoding speed and fast random access, may come at great expense when applied to level of detail and mip-map texture representations.

3.2 Vector quantization.

Beers et al. proposed a method for texture compression based on vector quantization (VQ) with mip-map texture representations [Bee96]. The first mip-map (original texture) was represented by 4×4 blocks from the codebook. On the basis of the first mip-map codebook the second and the third were created by averaging each 4×4 codeword respectively to 2×2 and 1×1 codewords. The final extended codebook was created by concatenating the 4×4 codeword with the corresponding sub-sampled 2×2 and 1×1 codewords. This representation achieved significant compression of 1bpp at the cost of lower reconstruction quality due to codeword sub-sampling.

Method proposed by Kwon and Chong used Interpolative Vector Quantization to represent a texture pyramid used for mip-mapping [Kwo00]. This allowed reducing correlation between mip-map levels. Additionally, the method needed to store two codebooks corresponding to low and high frequency texture terms and to interpolate the low frequency part of the block during the decompression.

Tang and Zhang in their texture compression method address the problem of texture regions visual importance [Tan05]. The codebook in this method is constructed taking into account such issues as visual importance and texture mapping distortion.

Generally, the VQ based methods suffer from two major problems: indirect data access and codebook handling. To retrieve single texture element we need two memory accesses. The codebook size can be too expensive for implementation in hardware and impose additional cost on texture caching when used with mip-maping.

3.3 Transform coding.

The most common transform method used in image compression which was applied to textures was Discrete Cosine Transformation (DCT). Talisman texturing systems (TREC) solved the problem of variable length coding of DCT by preserving the DC components without DPCM [Mic97]. Moreover, TREC used index table and link list to address each block, resulting in block random access. More elaborated texture compression scheme based on DCT was proposed by Chen and Lee [Che02]. They use adaptive quantization of 8×8 blocks, by assigning quantizer scale factor to each block. Thus, each block could be encoded to fixed-length code. Although those methods resulted in compression ratio higher than BTC methods and comparable to VQ methods with better image quality, they were too expensive for hardware implementation due to large block size, high complexity of inverse transform and lack of random access on texture pixel level.

Following the DCT more attention in TC is gaining the discrete wavelet transform (DWT). This is the result of the multi-resolution representation. Pereberin introduced the compression method based on Haar wavelet to encode three adjacent levels of mip-map pyramid [Per99]. Mapping the texture to YUV color space and reducing the insignificant coefficients allowed him to achieve average 4bpp compression. Representing a texture as a Wavelet Coefficient Tree (WCT) in the form of coefficient texture and the index texture was proposed by Candusi and DiVerdi [Can05]. Though, the DWT poses multi-dimensional feature, which makes it superior for hierarchical texture representation, it is not obvious how to reduce the insignificant coefficients to obtain the random access without severely reducing the compression ratio.

4. HIERARCHICAL TEXTURE COMPRESSION.

As the level of detail representation and filtering based on mip-mapping technique is ubiquitous in real-time applications, it is needed for textures to have a form of hierarchical structure.

It can be seen in image processing field that hierarchical approach gives an effective solution for compressing images. A good illustration being wavelet decomposition [Tau02] or Laplacian pyramid [Bur83]. These methods store the hierarchy explicitly compared to other class of methods based on fractal theory. Fractals are characterized by super-resolution property and while they can be represented on different resolutions levels, the hierarchical structure is not explicitly stored.

Even though, the hierarchical and multi-resolution feature of wavelets and fractals seems promising for texture compression applications, they are generally unsuitable. On one hand, wavelets methods need tree-walk procedures which require multiple accesses to memory. On the other hand, fractals during decompression need information from different parts of the texture. Moreover, they both are characterized by variable length code which makes them inferior to widely used block truncation coding methods.

Taking into consideration the requirements of texture compression we propose a method (HiTC) that combines the hierarchical representation with block-wise approach. This allows us to join the advantages of both approaches.

5. THE ALGORITHM DESCRIPTION.

The proposed algorithm is based on fractal and wavelet theory. Since the final structure of algorithm was subject to number of simplifications, the next part of the paragraph will explain step by step the process that was carried to derive it.

5.1 Introduction.

One of the most important factors in BTC texture compression methods is the size of the compressed texture block, which is in most cases equal to 4×4. It is the result of a balance between the image quality and compression ratio. Moreover, in texture compression applications it is considered as optimal for hardware implementation. But if we take into account the Fractal compression methods, the block that is subject to compression constitutes the whole image. The fact that the base fractal scheme does not address the problem of local compression [Fis95] makes the process of decompression computationally expensive. The work on the local fractal transform was carried on by the author resulting in the local fractal compression algorithm [Sta05]. That algorithm is based on quadtree structure. It allows for local decoding and random access on block level. The block size is restricted by the image quality and is not lower than 32×32. Although the local fractal compression algorithm achieves high compression ratios and close to real-time decompression, the quadtree structure makes it difficult for hardware implementation and it does not allow for random access on texture pixel level.

In our new method (HiTC) we take advantage of the local fractal compression algorithm for 4×4 blocks compressed independently. To overcome the problem of quadtree structure we use regular partitioning of compressed block to 2×2 range blocks R. The local domain pool for compressed block consists of only one domain block being the whole compressed block $D = B$. The resulting fractal code is a subject to further compression using Haar wavelet or Laplacian Pyramid.

5.2 Compression.

The process of texture compression in our algorithm (HiTC) consists of the following steps:

1. The texture is partitioned into 4×4 blocks.
2. Each block is subject to the modified local fractal compression.
3. The fractal code of each block is further compressed by Haar wavelet decomposition or is represented by Laplacian pyramid.

If we consider the texture block B, the modified local fractal compression is derived from fractal block-based method [Fis95] [Woh99] and is done by:

1. The 4×4 block B is divided into four non overlapping range blocks (4 squares of size 2×2).
$$\{R_j\},\ j = 1,2,3,4$$
2. Each range block R is matched to the domain block D (in this case $D = B$) by computing the optimal coefficients of the transformation:
$$\hat{R} = s \cdot \varphi(D) + o \tag{1}$$
which minimize the error $d(\hat{R}, R)$ in this case the root mean square error:
$$rms = \sum_{i=1}^{4} \left(s \cdot \varphi(d_i) + o - r_i\right)^2 \qquad r_i \in R_j,\ d_i \in D \tag{2}$$

where, $\varphi(\cdot)$ is the spatial contraction function which averages four adjacent domain block elements and then maps the averaged values onto range block applying one of eight isometries (sym). Next, the resulting range block values are scaled by s (scaling) and added to o (offset) coefficient [Fis95] [Nin97].

Thus, after compression each range block R_j is represented by a triple $\{s_j, o_j, sym_j\}$.

Pi et al. in the context of image coding proposed to replace the offset coefficient o (equation 1) by range block mean $\bar{r}$ which led to transformation of the form [Pi03]:
$$\hat{R} = s \cdot \varphi(D - \bar{d}) + \bar{r} \tag{3}$$

where $\bar{d}$ is the average domain value, $\bar{r}$ represents range block DC component and s (scaling) in this context is related to range block AC component.

They proved that the range block mean values are good approximation of image DC component. Consequently, starting from the initial image equal to the range-average image (DC image) could lead to faster convergence. Moreover, further iteration on DC image change only AC component, thus reducing the iterations on average to two.

Even though, the convergence of the algorithm proposed by Pi is faster we still need to compute the domain block average at each iteration. In our method, we solve this problem by making the domain block average as one of the compressed block coefficients. This is only cost-effective due to block size restriction to 4×4.

Finally, we obtain the following compressed block structure which is represented by four range blocks coefficients $R_j = \{s_j, \bar{r}_j, sym_j\}$, $j = 1,2,3,4$ and domain block average $\bar{d}$. As can be seen from the figure 1 two levels of mip-map are the part of our fractal code. Thus, we can further apply Haar transform or Laplacian pyramid to increase the compression ratio.

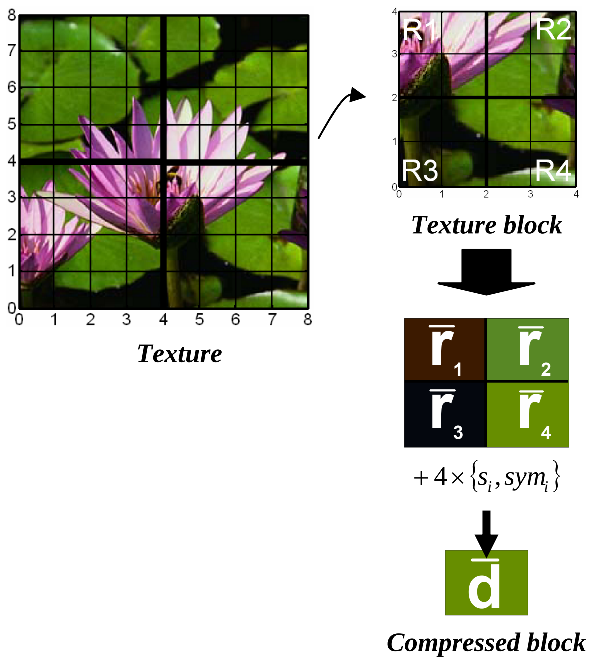

Figure 1. Hierarchical texture compression (HiTC).

A simple example reveals the effectiveness of our method. For Lenna image if we store only the mip-map pyramid and set all the rest coefficients to default values ($s_j = 0.75$, $sym_j = 0$, one to one mapping) we achieve $psnr = 30{,}65dB$, $rmse = 7{,}47$ and $C_R = 4.2:1$. In this case, we obtained compression ratio of BTC methods at the same time storing explicitly two levels of mip-map pyramid. Moreover, the same hierarchical structure represented by wavelets could not be compressed without wavelet coefficient quantization. Thus, each memory access for wavelets require a de-quantization step [Per99], while in our method it is done directly.

5.3 Coefficients allocation.

The next problem that must be solved is the coefficients allocation. Namely, given the compressed texture block representation we search for optimal coefficient bit allocation with respect to compression ratio and image quality. Therefore, each of the coefficients was independently subject to uniform and non-uniform quantization with default coefficients values set to: scaling 7-bits, offset 5-bits and symmetry 3-bits [Fis95].

5.3.1 Scaling.

The scaling coefficient was subject to uniform quantization in the range $[-1.5, 1.5]$. The value $s_{\max} = 1.5$ was chosen experimentally on the basis of the reconstruction error. The results of scaling quantization revealed that choosing the quantization levels with precision higher than 1/32 has little impact on image quality as it was indicated by Ning Lu [Nin97] (fig. 2). Moreover, restricting the scaling values to positive resulted in $1dB$ lower reconstruction error, which may be acceptable in texture compression. Applying the non-uniform quantization showed little improvement. The most noticeable difference was for one quantization level where the reconstruction error was reduced by $0.5dB$. It corresponded to reconstruction values $s \approx \{0.26, 0.86\}$, as opposed to $s \approx \{0, 0.75\}$ for uniform quantization. The compression was also checked for constant scaling coefficient. The optimal value was equal to $s = 0.5$. But since the constant value introduce the artefacts to high frequency term of the image, it seems reasonable to use at least one level of quantization for scaling coefficient.

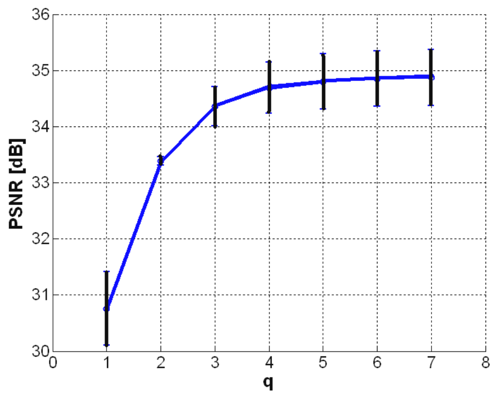

Figure 2. Scaling coefficient quantization for Lenna, with error bars indicating the difference between uniform and non-uniform quantization.

5.3.2 Symmetry.

The symmetry coefficient is responsible for mapping the domain block onto the range block [Fis95] (there are eight ways to map a square onto another). As it can be seen from the distribution of symmetry coefficient for positive scaling coefficient, the trivial mapping, which maps averaged domain values on the same positions in the range block is dominant. Additionally, which is not showed here for positive and negative scaling coefficient the second dominant mapping is the mapping which rotates the block by 180^0. Thus, it can be suggested to restrict the compression process to use only this trivial mapping and positive scaling coefficient. Moreover, if we remove the trivial mapping from the distribution diagram, the rest is uniformly distributed and therefore choosing any particular mapping gives no improvement. The only visual difference can be observed for at least four symmetries (fig. 4, q2 and q3)

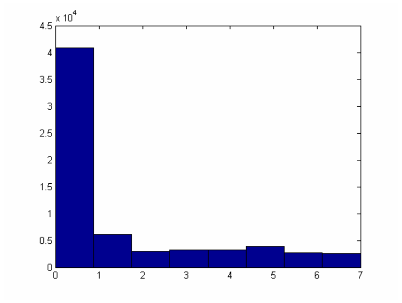

Figure 3. Symmetry coefficient distribution (0 – trivial mapping).

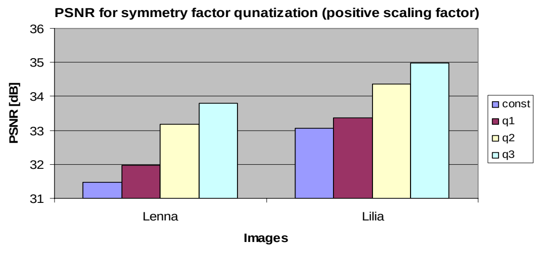

Figure 4. Symmetry coefficient quantization.

5.3.3 Mip-map.

The result of the texture compression process is the set of parameters, where the values $\{\bar{r}_j\}, j=1,2,3,4$ and $\bar{d}$ represent the second and the third (the lowest) level of the mip-map pyramid. Therefore, the next step is to apply compression to those parameters. We considered two approaches: the Haar decomposition [Per99] and Laplacian pyramid representation [Bur83]. We chose the Laplacian pyramid, since it fits more closely to decompression process and at the same time is less expensive computationally. It can be seen from equation 3, we can reduce the computation of domain difference $D-\bar{d}$ since it is a part of Laplacian pyramid representation. We represent the mip-map pyramid by storing the lowest level explicitly and the second level as the difference terms $d_j = \bar{r}_j - \bar{d}\,, i=1,2,3,4$. Moreover, we take advantage of color space with separated luminance and chrominance to coarsely approximate the chrominance data.

5.4 Block structure and decompression.

Generally, the structure of compressed texture block is represented by the triples $\theta = \{s, \bar{r}, sym\}$, where each corresponds to one of four range blocks. Taking into account the previous paragraph we can reduce this representation. The proposed bit allocation scheme is presented below.

The scaling coefficient can be represented by one bit and in the case of the uniform quantization the reconstruction values are $s \approx \{0, 0.75\}$. Since when using only one quantization level the value of the coefficient is on one third equal to zero, thus the process of decompression is usually reduced to retrieving the block mean value (equation 3).

The symmetry coefficient is dominated by one to one mapping, thus it can be safely removed from the block structure.

The mip-map representation in our case is compressed by using the Laplacian pyramid representation. But before that we take advantage of color space conversion to reduce the data. We chose the YC$_b$C$_r$ color space to compress the chrominance data. We allocate 7-bits for luminance difference terms and 3-bits for chrominance difference terms.

The final block bit allocation structure for the color components consists of:

- block average value $\bar{d}$ (8-bits),
- difference terms d_j, $j=1,2,3,4$ (7-bits – luminance and 3-bits chrominance),
- scaling coefficient s_j, $j=1,2,3,4$ (1-bit).

This allocation scheme gives 88-bits (40-bits for luminance and 2x24-bits for chrominance components) and compression ratio $C_R = 5.72:1$ (the compression of S3TC for the same set of mip-maps would be $C_R = 4.57:1$ [Per99]).

The decompression process for color component (Y, C$_b$ or C$_r$) simulating three level mip-map pyramid can be described in the following steps:

```
ColorComp getTexel(int x, int y, int mipLevel) {
     CBlock_x = x/4; CBlock_y = y/4;
     If (mipLevel >= 2) {
          getCoeff(CBlock_x, CBlock_y, &d);
          return d;
     }
     else if (mipLevel == 1) {
          getCoeff(CBlock_x,CBlock_y,&d,&d1);
          return d + d1;
     }
     //mip level 0
     getCoeff(CBlock_x,CBlock_y,&d,&s,&d1,&d2);
     if (s == 0)
          return  d + d1;         //r = d + d1
     else
          return s*(d2) + d + d1; //s*(di-d)+r;
}
//getCoeff() – unpacks the data from the block
```

6. RESULTS.

The presented method (HiTC) was compared with S3TC algorithm on a sequence of test images (fig. 5) and the rendered OpenGL scene (fig. 6). The error was measured by computing the peak signal to noise ratio on luminance component. The luminance component was chosen for the reason of its visual importance.

The HiTC shows better reconstruction in the regions of smooth color variance as opposed to the S3TC. The S3TC reconstruction error is uniformly distributed over whole image which can be easily seen in the form of the block effect. This effect is especially magnified on the block edges where the colors do not lie on a line in a RGB color space. The same effect in minor form can be observed for HiTC. It is reduced in our method by approximating the colors of the texture by the mip-map pyramid.

The final reconstruction errors for the textures should be taken with caution. Since they are part of the visualized scene, they are subject to mapping and filtering process. Although, the compression of single texture gives some differences when comparing with uncompressed texture, the final rendered scene is visually indistinguishable. Both methods achieved the peak signal to noise ratio for rendered scene higher than $40dB$.

The problem of decompression cannot be fully addressed without hardware implementation. However, since our method addresses the problem of storing three levels of mip-map in one block, we could expect the lowest complexity of accessing the texture pixel comparing to other methods.

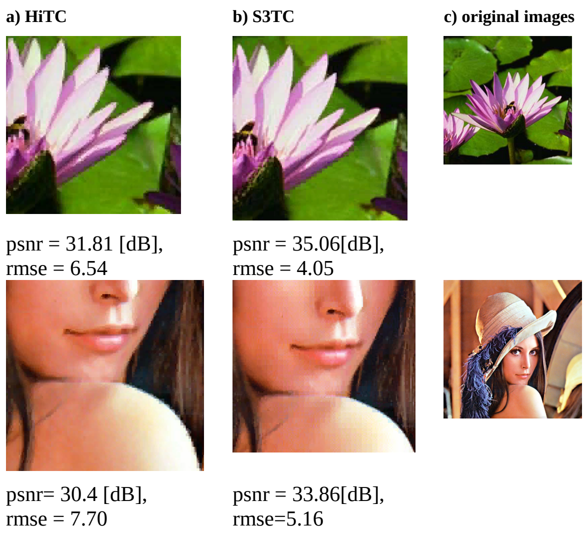
a) HiTC
b) S3TC
c) original images
psnr = 31.81 [dB], rmse = 6.54
psnr = 35.06[dB], rmse = 4.05
psnr= 30.4 [dB], rmse = 7.70
psnr = 33.86[dB], rmse=5.16

Figure 5. Texture reconstruction error.

7. CONCLUSION.

We have proposed a new approach for texture compression. The major advantage of our fractal block-based approach is a hierarchical representation, which allows for:

- direct decompression, which does not need any iteration since all the coefficients are stored explicitly in the compressed block structure,
- texture hierarchical representation, which is the result of our modified local fractal compression process,
- low computational complexity - to compute first level we need to perform only one multiplication and one addition, the second level needs only one addition and the third level is stored explicitly.

HiTC has the advantage on currently used BTC methods that the process of mip-mapping is addressed by the compressed texture block (table 1). This can be seen when used with trilinear filtering. Our method outperforms currently proposed solutions, since it can access three levels of mip-map directly. The hardware implementation is simple and does not require any external structures which for example needs vector quantization methods. There is no pre-processing step related to coefficient de-quantization which is common for wavelet compression. The computational complexity was reduced to minimum with the aim of real-time application. Moreover, all the requirements on texture compression method are fulfilled (table 1), thus making it superior for high performance rendering architectures.

8. REFERENCES

[3df99] 3dfx. FXT1: White paper. 3dfx Interactive. http://wwwdev.3dfx.com/fxt1/fxt1whitepaper.pdf, 1999.

[Ake03] Akenine-Möller T., Ström J. Graphics for the Masses: A Hardware Rasterization Architecture for Mobile Phones. ACM Transactions on Graphics, 22, 3 (2003), 801–808.

[Bee96] Beers A. C., Agrawala M., Chaddha N. Rendering from compressed textures. Siggraph 1996, pp. 373–378, July 1996.

[Bur83] Burt P.J. and Adelson E.H. The Laplacian pyramid as a compact image code. IEEE Transactions on Communications, 31:532–540, 1983.

[Cam86] Campbell G., Defanti T. A., Frederiksen J., Joyce S. A., Leske L. A., Lindberg J. A., Sandin D. J. Two Bit/Pixel Full Color Encoding. In Proceedings Of Siggraph (1986), Vol. 22, Pp. 215–223.

[Can05] Candussi N., DiVerdi S., Hollerer T. Real-time Rendering with Wavelet-Compressed Multi-Dimensional Textures on the GPU. Computer Science Technical Report 2005-05, University of California, Santa Barbara.

[Che02] Chen C.-H., Lee C.-Y. A JPEG-like texture compression with adaptive quantization for 3D graphics application. The Visual Computer, vol. 18, 2002, pp. 29-40.

[Fen03] Fenney S. Texture Compression using Low-Frequency Signal Modulation. In Graphics Hardware (2003), ACM Press, pp. 84–91.
[Fis95] Fisher Y. (Ed.). Fractal Image Compression: Theory and Application to Digital Images. Springer Verlag, New York, 1995.
[Iou99] Iourcha K., Nayak K., Hong Z. System and Method for Fixed-Rate Block-based Image Compression with Inferred Pixels Values. In US Patent 5,956,431 (1999).
[IVA00] Ivanov D., Kuzmin Y. Color Distribution – A New Approach to Texture Compression. In Proceedings of Eurographics (2000), vol. 19, pp. C283–C289.
[Kni96] Knittel G., Schilling A., Kugler A., Strasser W. Hardware for Superior Texture Performance. Computers & Graphics 20, 4 (July 1996), 475– 481.
[Kwo00] Kwon Young-Su, Park In-Cheol, and Kyung Chong-Min. Pyramid Texture Compression and Decompression Using Interpolative Vector Quantization. Proceedings of 2000 International Conference on Image Processing, vol. 2, pp.191-194, Sep. 10-13, 2000.
[Lev00] Levkovich-Maslyuk L., Kalyuzhny P. G., and Zhirkov A. Texture Compression with Adaptive Block Partitions. ACM Multimedia 2000, Nov.2000.
[Mic97] Microsoft, "Escalante hardware overview. Talisman". Graph Multimedia Syst, pp 89–106.
[Nin97] Ning Lu, Fractal Imaging, Academic Press, 1997.
[Per99] Pereberin A.V. Hierarchical Approach for Texture Compression. Proceedings of GraphiCon '99, 1999,195–199.
[Pi03] Pi M., Basu A., and Mandal M. A new decoding algorithm based on range block mean and contrast scaling. IEEE International Conference on Image Processing (ICIP), vol. 2, pp. 241-274, Barcelona, Spain, September 14-17, 2003.
[Sta05] Stachera J., Nikiel S. Large textures storage using fractal image compression. to be published in Computational Imaging And Vision book series, Kluwer 2005.
[Str04] Ström J., Akenine-Möller T. PACKMAN: Texture Compression for Mobile Phones. In Sketches program at SIGGRAPH (2004).
[Str05] Ström J. and Akenine-Möller T., iPACKMAN: High-Quality, Low-Complexity Texture Compression for Mobile Phones, Graphics Hardware 2005, pp. 63-70, 2005.
[Tan05] Tang Y., Zhang H., Wang Q., Bao H. Importance-Driven Texture Encoding Based on Samples. Computer Graphics International 2005.
[Tau02] Taubman D., Marcellin M. W. JPEG2000: Image Compression Fundamentals, Standards and Practice. Kluwer, Boston, 2002.
[Wil83] Williams L. Pyramidal Parametrics. Computer Graphics (SIGGRAPH'83 Proceedings), Pages 1-11, July, 1983.
[Woh99] Wohlberg B., Jager G.,"A Review of the Fractal Image Coding Literature",IEEE Transactions On Image Processing,Vol. 8, No. 12, December 1999

Method	**Random Access**	**Simple decoding**	**Simple hardware implementation**	**Hierarchical representation**	**Compression Ratio**	**Image quality**
BTC	Yes	Yes	Yes	No	Average	Average
VQ	Yes	Yes	No	No	High	Average
DCT	No	No	No	No	High	High
DWT	No	No	No	Yes	Highest	Highest
Fractal	No	Yes	No	Yes	Highest	High
HiTC	**Yes**	**Yes**	**Yes**	**Yes**	**Average**	**Average**

Table 1. Texture compression methods comparison

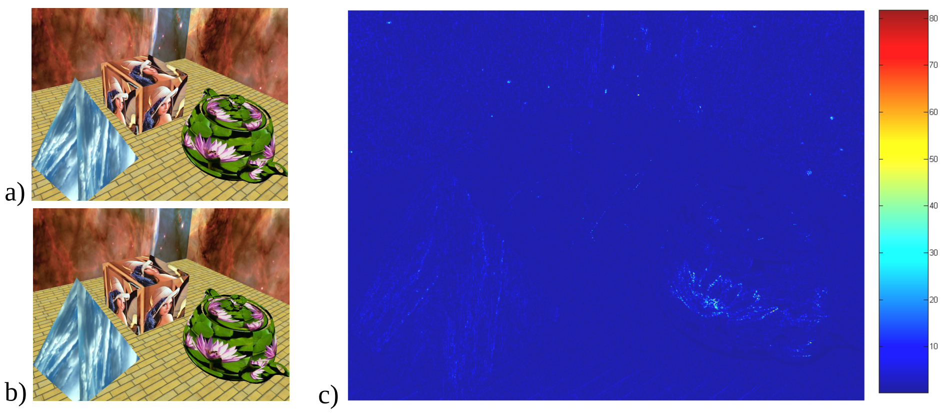

Figure 6. Scene rendered in OpenGL. a) normal view with uncompressed textures and with c) HiTC compressed textures. c) Scene error image for HiTC textures (psnr = 43dB, rmse = 1.82) .