Fractal-based hierarchical mip-pyramid texture compression

J. Stachera and P. Rokita
Institute of Computer Science, Warsaw University of Technology, Poland

October 16, 2006

Abstract. As the level of realism of computer generated images increases with the number and resolution of textures, we are faced with the problem of limited hardware resources. Additionally, the filtering methods require multiple accesses and extra memory space for the texture representation, thus severely reducing the memory space and bandwidth, the most common example being mip-mapping technique. We propose a hierarchical texture compression algorithm for real-time decompression on the GPU. Our algorithm is characterized by low computational complexity, random access and hierarchical structure which allow us to access first three levels of encoded mip-map pyramid. The hierarchical texture compression algorithm HiTC is based on block-wise approach, where each block is subject to local fractal transform and further effectively coded by one level of Laplacian Pyramid.

Key words: 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 [3], 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 the classical image compression algorithms can not be applied to textures. There is a strong need for efficient, highly specialized texture compression algorithms. According to Beers et al. [9] and our findings [30] when designing a texture compression algorithm the following aspects must be taken into consideration:
Decoding speed. It is the most important feature which allows for rendering directly from the compressed texture. As the texture compression is mainly used in real time computer graphics the decompression algorithm must be designed to allow high frame-rate.
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 the existing texture compression algorithms can be divided into three major groups: block truncation coding (BTC) and local palettes, vector quantization (VQ) and transform coding.

3.1. Block Truncation Coding and Local Palettes

Introduced by Delph and Mitchel [1], Block Truncation Coding (BTC) has become the standard for the many real-time texture compression methods implemented in hardware [13][24][28]. The method gained its popularity due to simple decoding algorithm. In BTC the image is divided into uniform grid of $4x4$ blocks where each block is represented by fixed number of greyscale values and index map to reference the values for block pixels. An extension of BTC (called Color Cell Compression CCC) to color images was proposed by Campbell et al. [4]. The block was represented by index map but the color values

were stored in a color palette which was shared by all image blocks. The drawbacks of CCC were indirect data access and color banding. The CCC problems were effectively addressed by S3TC, which is based on the concept that most of the natural image block colors lie close to the line in RGB color space. Although the texture quality of S3TC is significantly better than CCC and BTC, it shares the problem of block artefacts typical to BTC methods and in some cases false color due to linear color interpolation. A number of solutions were proposed to tackle those problems, a good examples being: color distribution [19], sub-palettes [21], low frequency signal modulation [24] and others [23][28].

3.2. Vector Quantization

The pioneered work of Beers et al. which gives fundamentals for texture image compression, was also the first which used vector quantization for image compression and stored texture representation in the hierarchical structure [9]. This representation was able to represent first three levels of mip-map pyramid at 1bpp. Based on the remarks made by Beers in his paper [9], Kwoon applied the Interpolative Vector Quantization (IVQ) to compress whole mip-map pyramid [20]. The resulted code was comprised of two codebooks for image AC and DC component and requires additional memory accesses. Completely different approach was taken by Tang et al. where the codebook was represented by image blocks chosen by taking into account such issues as visual importance and texture mapping distortions [29].

3.3. Transform coding

The most common transform method used in image compression which was applied to textures was Discrete Cosine Transformation (DCT). It was used in Talisman Texturing System (TREC) [10] and in Image Processing Unit (IPU) of Playstation [15]. In the case of TREC random access to compressed data required index tables and linked lists. More elaborate scheme was proposed by Chen and Lee [17]. They encoded DCT $8x8$ blocks to fixed length code using a form of adaptive quantization. That allowed for random access without the use of dynamic structures. Pereberin proposed a method based on Haar wavelet transform [14] to represent three levels of mip-map pyramid. Mapping the 4x4 texture blocks to YUV color space and reducing the insignificant coefficients allows him to achieve moderate compression ratio of 4bpp and random access to texture data.

3.4. Problems

The general problem with BTC based methods is that they do not address the problem of mip-mapping (Figure 5). It results in additional decompression units when applied to hierarchical structures such as mip-maps. The problem is partially solved in transform methods, where wavelet transform is used. But the compact representation of wavelets

which ensure high compression ratios is based on the variable length code. Moreover, the decompression process needs tree walk procedures [27], thus requiring intensive memory communications. It is not clear how to reduce the insignificant coefficient of wavelet transform to obtain the random access without severely reducing the compression ratio [14]. In the case of VQ methods which are also the basis of some of the GPU compression methods, the problem is with the indirect data access and codebook handling. Each reference to the texel requires at least two memory transactions (one to read index and another to read the data). The problem is especially magnified in the methods which use more than one codebook [20][26][29].

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. We propose a new algorithm for hierarchical texture compression (HiTC) with the implementation on the GPU. The next section will briefly introduce the main concepts of fractal image compression related to our work (IFS, PIFS and no search fractal methods).

4.1. Fractal Compression

The main principle of fractal image compression is to represent an image by a contractive transformation which fixed point is close to that image. The first fully automatic fractal compression method was introduced by Jacquin as an extension of IFS [5], which was called PIFS (Partitioned Iterated Function System) [6]. The difference between IFS and PIFS is that it the case of PIFS the individual mapping operates on the subset of the image rather than on the entire image. Thus, to encode the image Jacquin defined two types of blocks: range blocks and domain blocks. Formally, let I be an encoded image. A set R of non-overlapping range blocks $\hat{R} = \{R_0, R_1, ..., R_n\}$, $R_i \cap R_j = \emptyset$ that tile I, $I = \bigcup_{i=0}^{n} R_i$ is called range pool. A set D of overlapping domain blocks $\hat{D} = \{D_0, D_1, ..., D_n\}$ is called domain pool $D_i \in I$. A set W is consisted of contractive transformation which for each range block R maps on its corresponding domain block D, $w_i : D_i \rightarrow R$. The most expensive operation in the fractal compression is the searching part where for each range block $R \in \hat{R}$ we seek to find the best domain block $D \in \hat{D}$, minimize the error $E(R, D)$. A number of papers were proposed to address this problem [16]. But, there was considerably less research done on the fast fractal decompression methods which are especially important in real-time texture mapping applications. Turner [22] examined the fractal properties in the process of the texture mapping. The decompression process in Turner algorithm is applied to the transformed texture in the image space and is based on the iterative decoding method. The use of

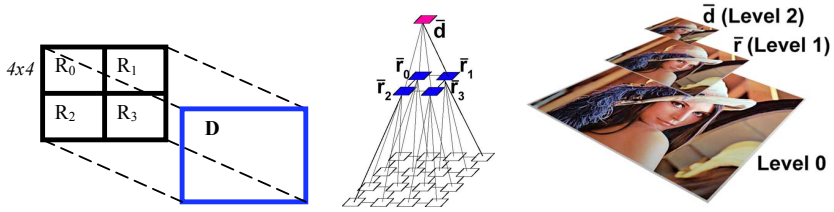
iterative method in texturing has potential disadvantages: the process needs buffering of intermediate texture images, the number of iteration is variable for different images, viewing only a part of the texture requires decoding full texture.

4.2. Design of Hierarchical Texture Compression algorithm (HiTC)

Taking into account the aspects of texture compression (section 2) we divided the design process of hierarchical texture compression algorithm into several steps, which comprised: partitioning scheme, parameters selection, parameters coding and color space selection.

4.2.1. Partitioning scheme

We chose the fixed size square blocks which can be represented by the fixed length code and thus allows for random access. The domain range relation which is similar to fractal coding based on IFS [5](Figure 1a). The difference is that we compress each texture block independently. Thus, it is IFS coding on the block basis [7] as opposed to Jacquin PIFS [6] where the domain block could be referenced from the whole image. In the proposed fixed partitioning scheme we restrict the domain block size to $4x4$ pixels (Figure 1c). The domain block size of $4x4$ pixels can achieve better reconstruction quality than bigger blocks used with fixed partitioning [7]. Moreover, the texture block can be represented more compactly, thus reducing the number of accesses for compressed texture block data. The domain range relation in our scheme is: $D = R_0 \cup R_1 \cup R_2 \cup R_3$ (1). Thus, the domain block is represented by the union of four range blocks.



a) Proposed b) Block c) Texture

Fig. 1. The domain range relation (a) and Code representation (b,c)

4.2.2. Parameters selection

Because of the strong correlation of contrast scaling s_i and luminance offset o_i in standard transform [6]: $\tilde{R} = w_i(D) = s_i\varphi(D) + o_i$ (2) it is difficult to find optimal bit allocation for those parameters. Oien and Lepsoy [8] and later on Pi et al. [25] advocated to replace the luminance offset o_i by range block mean $\bar{r}$. Thus, replacing o_i with $\bar{r} - s\bar{d}$ in standard affine transform (equation 2), resulted in: $\tilde{R} = w_i(D) = s_i\varphi(D - \bar{d}) + \bar{r}_i U$ (3) and in the error expression: $E(R, D) = \| R - s(D - \bar{d}U) - \bar{r}U \|^2 = \sum(r_{ij} - s\varphi(d_{ij} - \bar{d}) - \bar{r}_i)^2$ (4)

where the parameters s and $\bar{r}$ are pre-quantized. Thus, each range block $R \in \hat{R}$ is represented by: $\{s, \bar{r}, i_d, j_d\}$ (5). Due to the block size restriction, equal to $D = 2B$ and proposed partitioning scheme, the spatially contracted domain block is stored in our code in the form of range block means: $\varphi(D) = \bar{r}_0 \cup \bar{r}_1 \cup \bar{r}_2 \cup \bar{r}_3$ (6) (Figure 1b). Therefore, the only parameter in the decoding phase (equation 3) which needs iteration is domain average $\bar{d}$. In our algorithm, we store this average as a part of our code, making our algorithm non-iterative. Each texture compressed block in our scheme is represented by four range blocks affine parameters (Figure 1a): $W = \{\{w_i\}_{i=0}^{3}, \bar{d}\}, w_i = \{s, \bar{r}\}$ (7). It should be noted, that in our algorithm the compressed texture block constitutes the domain block and these names may be used interchangeably. The domain range position is fixed (Figure 1a) thus we do not store the domain position (equation 5, 7).

4.2.3. Parameters coding and Color space selection

As can be seen from equation 7, the compressed $4x4$ block code $W = \{w_i\}_{i=0}^{3}$ is partly represented by its reduced (averaged) versions: $2x2$ and $1x1$ which are in the form of $\{\bar{r}_i\}_{i=0}^{3}$ and $\bar{d}$ (Figures 1b, 1c). In terms of the compressed texture, those parameters represent first and second level of Gaussian Pyramid [2] (known in texture filtering as a Mip-Map pyramid [3]). In our algorithm, we use one level of Laplacian pyramid to reduce the correlation between parameters $\{\bar{r}_i\}_{i=0}^{3}$ and $\bar{d}$ (level 1 and 2). We use the simplest expand operation by zero order polynomial. Thus, the parameters $\{\bar{r}_i\}_{i=0}^{3}$ (level 1) are stored as the difference terms in the form: $diff = \{\bar{r}_i - \bar{d}\}_{i=0}^{3}$ (8) and the domain average values are stored explicitly (level 2). Thus, the block code is represented by: $W = \{\{w_i\}_{i=0}^{3}, \bar{d}\}, w_i = \{s, diff\}$ (9). In the case of textures, which are mostly represented in RGB color space, we make use of pre-processing phase in which the texture data are converted to YUV color space. This color space decorrelates the color data and even coarse approximation of chrominance channels (UV) leads to good approximation of the whole image.

5. Hierarchical Texture Compression - the algorithm

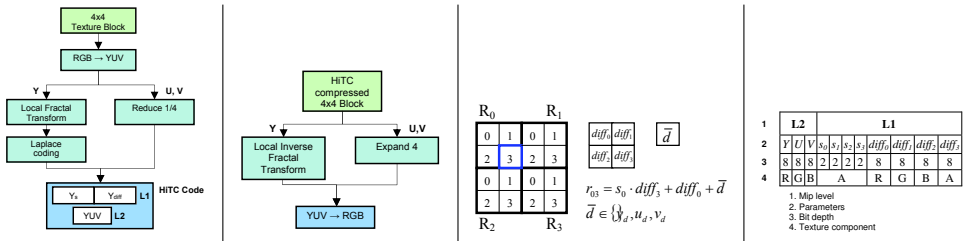
Taking into account the design phase of Hierarchical texture compression algorithm (HiTC), we propose an algorithm for hierarchical texture compression, in which compression phase is consisted of the following steps (Figure 2a):

1. Conversion of the texture to YUV color space.
2. For Y component apply:
   - Partitioning of the texture to $4x4$ blocks (Figure 1).
   - For each $4x4$ texture block apply local fractal transform.
   - The resulting coefficients code by one level of Laplacian pyramid.
3. For U and V component apply:

- Low-pass filtering operation to reduce the component size to $\frac{1}{4}$.

and the decompression phase is consisted of (Figure 2b):

1. For referenced texture texel compute the compressed texture block address.
2. Apply local inverse transform to Y component.
3. Expand the U and V values 4 times.
4. Convert the texel YUV values to RGB values.



a) Compressiom b) Decompression c) Inverse LF transform d) Block structure

Fig. 2. HiTC algorithm

5.1. Local fractal transform

The local fractal transform is computed by minimizing the equation 4 for each range block R (Figure 1a), with respect to the scaling parameter s. It means that for each range block R we need to compute the scaling parameter s: $s = \frac{\langle R-\bar{r}U, D'-\bar{d}U\rangle}{\|D'-\bar{d}U\|^2}$ and as a part of this process range block mean $\bar{r}$ and the domain mean $\bar{d}$ (Figure 1b). The result of applying the local fractal transform to the luminance values (Y) is the set of parameters: $W = \{\{w_i\}_{i=0}^{3}, \bar{d}\}, w_i = \{s, \bar{r}\}$. The range block means are next subject to one level of Laplace coding, where they are encoded as the difference terms. Finally, the luminance values are represented by: $W = \{\{w_i\}_{i=0}^{3}, \bar{d}\}, w_i = \{s, diff\}$

5.2. Inverse local fractal transform

The inverse local transform is computed as in equation 2: $\tilde{R} = w_i(D) = s_i\varphi(D - \bar{d}) + \bar{r}_i U$. The only difference is that we do not store the range means $\bar{r}$ but the difference terms $diff$ (equation 7), which also represent the normalized contracted domain values $\varphi(D - \bar{d})$. Thus, to reconstruct the range value we need to compute: $r_{ij} = w_i(D) = s_i diff_j + diff_i + \bar{d}$ (10) where $diff_i$ is used to reconstruct the range mean and $diff_j$ is contracted normalized domain value, which depends on the texel positions in the range block (Figure 2c).

The range blocks as the texels in the range blocks are arranged in Morton Order. We do not apply any isometry operation prior to mapping the domain block onto the range block as in Jacquin [6] since the one to one mapping is dominant in most cases [11][30].

6. GPU implementation

6.1. Block structure

We decided to use rectangular RGBA texture format with 8bit texel depth. The structure of the HiTC block is presented on the figure 2d. We use the uniform quantization for scaling parameters $\{s_i\}_{i=0}^{3}$ in the range $[0.0, 1.5]$ [11][30], which are represented by 2bits ($s = \{0, 0.375, 0.75, 1.125\}$). The difference terms $\{diff\}_{i=0}^{3}$ are stored on 8bits [30]. The normalization of difference term is performed to effectively exploit the 8bit texture values and to avoid negative values. Thus, they are normalized prior to packing the data to the texture. Therefore, it is needed to calculate the bias and scale factor:

$$\begin{aligned} bias &= min(diff) \\ scale &= \begin{cases} max(diff) - min(diff) & if\ max(diff) \neq min(diff) \\ 1 & else \end{cases} \end{aligned} \tag{11}$$

The normalized values are computed by: $diff_{norm} = \frac{diff - bias}{scale}$ (12). The block structure is represented totally by $64bits$, which corresponds to two RGBA texture values (Figure 2d). This allocation scheme is not optimal for the compression, since we do not take into account the difference terms quantization. They are characterized by low variance and thus they can be quantized coarsely. The reason, why we decided for this allocation is that it allows for simple decoding. We experimented with other allocations schemes [30][31] which can significantly increase the compression ratios. But, storing few different parameters (e.g. scaling and difference term) in one byte requires bit-wise operations and additionally applying non-uniform quantization to difference terms involve complex decoding and additionally nowadays GPU are not equipped with integer processing units.

6.2. Decoding

We implemented our algorithm in two modes with nearest neighbor and 3 mip-map level filtering. The block diagram on figure 3b represents the algorithm for CG language. The rectangular texture sampler object and texture coordinates are the input data to our algorithm, the output constitutes the linearly interpolated texel values. The decompression algorithm is executed in the pixel shader units. Generally, the HiTC decompression algorithm can be divided into following steps (Figure 3b):

1. For given texture coordinates find the HiTC address and compute the mip-level.
2. Fetch the HiTC block data from the texture.
3. Unpack the parameters from the HiTC block.

4. Apply filtering.

7. Experimental results

7.1. GPU decompression

We measured the raw performance of our algorithm for simple scene consisted from the textured rotating cube for two modes: nearest neighbor and 3 level mip-mapping on the NVIDIA GeForce 6800 in the resolution 1280x1024 with screen coverage equal to 75%. The values in the table represent the peak performance of our algorithm.

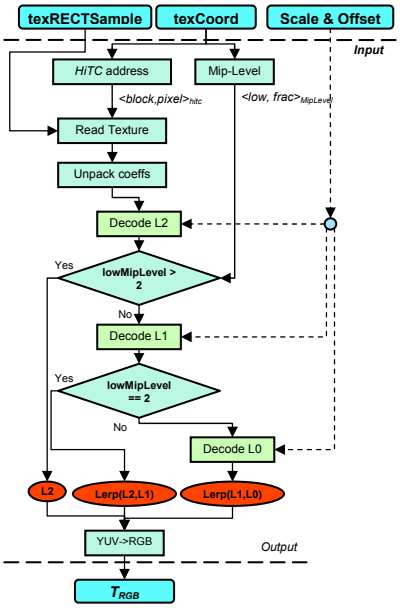
texture	size	fps*
Lilia	256	192
Baboon	512	192
Earth	2048	187

nearest neighbor

texture	size	fps*
Lilia	256	265
Baboon	512	265
Earth	2048	252

3 level mip-mapping

a) Decompression



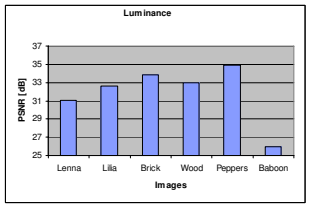
b) Pixel shader program

Fig. 3. HiTC GPU performance and implementation

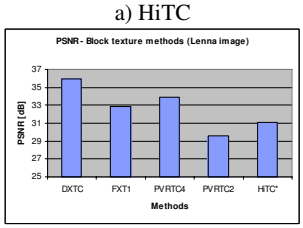
7.2. Reconstruction results

We used the peak signal to noise ratio PSNR to measure the reconstruction error for luminance image. The luminance component was chosen taking into account its visual importance and the HiTC compression algorithm, in which we compress only luminance, the chrominance components are only sub-sampled. In this case the reconstruction

quality of chrominance depends only on applied interpolation method. For three mip-map levels that are stored in HiTC block, it results in the compression ratio $C_R = 9 : 1$. The compared compression algorithms are characterized by compression ratios $C_R = 6 : 1$ (DXTC [13], FXT1 [12], PVRTC [24]), the only difference is PVRTC2 which has $C_R = 6 : 1$. The reconstruction results of our method are comparable to block compression methods. The advantage of our method can be seen in higher compression ratio as opposed to DXTC, FXT1 and PVRTC4 ($C_R = 6 : 1$). The reconstruction quality is better than PVRTC2 which gives compression ratio $C_R = 12 : 1$.



a) HiTC



b) TC methods comparison

a) Reconstruction error

b) Test and error images

Fig. 4. Reconstruction error, PSNR for luminance component [SE02]

8. Conclusion

We have presented a new algorithm for texture compression with the implementation on GPU. The major advantage of our fractal block-based approach is a hierarchical representation, which allows for:

- direct decompression of three levels of mip-map pyramid, easily extendable to all levels,
- low computational complexity, which allowed us to implement the method on the GPU,
- compact block representation, which allows for fast random access to texture data.

The overhead related to access the levels of mip-map pyramid is greatly reduced in our method, since HiTC block represents three levels of mip-map pyramid. Thus, in terms of the number of accesses in trilinear filtering modes, the HiTC method reduces it by one third as compared to state of the art block texture compression methods. When compared to hierarchical methods such as wavelets, the texture access in our method does not require any tree walk procedures. The computational complexity was reduced to minimum with the aim of real-time application. Moreover, all the requirements outlined earlier in section 2 are fulfilled, thus making it superior for high performance rendering architectures (Figure 5).

References

1979
[1] Delp J., Mitchell R.: Image Compression Using Block Truncation Coding, IEEE TOC, v. 27, n. 9, 1979.

1983
[2] Burt P.J., Adelson E.H.: The Laplacian pyramid as a compact image code. IEEE Transactions on Communications.
[3] Williams L.: Pyramidal Parametrics. Computer Graphics (SIGGRAPH'83 Proceedings), Pages 1-11.

1986
[4] 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.

1988
[5] Barnsley M., Sloan A., A better way to compress images, BYTE, Volume 13 , Issue 1 (January 1988), pp. 215-223.

1992
[6] Jacquin, A.E.: Image coding based on a fractal theory of iterated contractive image transformations, IEEE Transactions on Image Processing, Volume 1, Issue 1, Jan. 1992, pp. 18-30.

1995
[7] Dudbridge F.: Least Squares Block Coding by Fractal Functions, Fractal Image Compression: Theory and Application to Digital Images, Springer-Verlag, New York, pp. 231 -244.
[8] Oien G. E., Lepsoy S.: A class of fractal image coders with fast decoder convergence, Fractal image compression: theory and application, pp: 153 - 175.

1996
[9] Beers A. C., Agrawala M., Chaddha N.: Rendering from compressed textures, Siggraph 1996, pp. 373-378, July 1996.
[10] Torborg J., Kajiya J.: Talisman: Commodity Realtime 3D Graphics for the PC, Siggraph'96.

1997
[11] Ning Lu: Fractal Imaging, Academic Press.

1999
[12] 3dfx. FXT1: White paper. 3dfx Interactive. http://wwwdev.3dfx.com/fxt1/fxt1whitepaper.pdf
[13] 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
[14] Pereberin A.V.: Hierarchical Approach for Texture Compression, Proceedings of GraphiCon '99, 1999,195-199.
[15] Suzuoki, M. Kutaragi, K. Hiroi, T. Magoshi, H, et. al.: A microprocessor with a 128-bit CPU, ten floating point MAC's, four floating-point dividers, and an MPEG-2 decoder, IEEE Journal of Solid-State Circuits, November 1999, Vol 34, Issue 11
[16] Wohlberg B., Jager G., A review of the fractal image coding literature, IEEE Transactions on Image Processing, vol. 8, no. 12, pp. 1716–1729

2000
[17] Chen C.-H., Lee C.-Y.: A JPEG-like texture compression with adaptive quantization for 3D graphics application. The Visual Computer, vol. 18, pp. 29-40
[18] G. Sullivan, S. Estrop.: Video Rendering with 8-Bit YUV Formats, Microsoft Developer Network.
[19] Ivanov D., Kuzmin Y.: Color Distribution - A New Approach to Texture Compression. In Proceedings of Eurographics (2000), vol. 19, pp. C283-C289.
[20] 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.
[21] Levkovich-Maslyuk L., Kalyuzhny P. G., Zhirkov A. Texture Compression with Adaptive Block Partitions. ACM Multimedia 2000.
[22] Turner M.J.: Properties of Fractal Compression and their use within Texture Mapping, First IMA Conference on Fractal Geometry: Mathematical Methods, Algorithms and Applications.

2003
[23] Akenine-Mller T., Strm J.: Graphics for the Masses: A Hardware Rasterization Architecture for Mobile Phones. ACM Transactions on Graphics, 22, 3 (2003), 801-808.
[24] Fenney S.: Texture Compression using Low-Frequency Signal Modulation. In Graphics Hardware (2003), ACM Press.
[25] Pi M., Basu A., 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.
[26] Schneider J.: Kompressions- und Darstellungsmethoden fr hochaufgelste Volumendaten, Diploma Thesis (in English), RWTH Aachen, Germany.

2005
[27] 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.
[28] Strom J. Akenine-Moller T.: iPACKMAN: High-Quality, Low-Complexity Texture Compression for Mobile Phones, Graphics Hardware 2005, pp. 63-70.
[29] Tang Ying, Zhang Hongxin, Qing Wang, Bao Hujun: Importance-Driven Texture Encoding based on Samples, in Proceedings of Computer Graphics International 2005, N.Y.

2006
[30] Stachera J., Rokita P.: Hierarchical Texture Compression, WSCG Conference proceedings.
[31] Stachera J.: Hierarchical Texture Compression, http://staff.elka.pw.edu.pl/ jstacher/

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
HiTCg	**Yes**	**Yes**	**Yes**	**Yes**	**Average**	**Average**

Fig. 5. Texture compression methods comparison