GPU-Based Hierarchical Texture Decompression

J. Stachera and P. Rokita

Institute of Computer Science, Warsaw University of Technology, Poland

Abstract

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 HiTCg is based on block-wise approach, where each block is subject to local fractal transform.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Color, shading, shadowing, and texture

1. 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. [BAC96] and our findings [SR06] 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.

2. Previous work.

All the existing texture compression algorithms can be divided into three major groups: block truncation coding (*BTC)* [DM79] and local palettes [INH99][Fen03][SA05], vector quantization (*VQ*) [BAC96][KPK00], transform coding [TK96][CL02] and hierarchical coding [Per99][SR06]. In the case of the *GPU* implementation there is no clear division and we can observe algorithms being a mixture of different compression methods.

2.1. GPU texture compression/decompression.

Kraus and Ertl [KE02] introduced an algorithm for adaptively representing the texture map based on uniform grid index method which allows for random access and decompression on the GPU. Similar method was introduced by Tang et al. [TZQB05]. The difference was that in Tang method the texture blocks were of fixed size. Thus, those methods allowed for lossless compression of critical texture parts and were primarily useful for sparse textures. Candusi et al. [CDH05] used discrete Haar wavelet transform to compress textures. They used two structures, one to represent the wavelet coefficient tree (*WCT*) and another to store the index data to reference the tree nodes. Schneider proposes to use vector quantization to three levels of Laplacian pyramid which represented the volumetric texture [Sch03]. That representation required to store two codebooks for first two levels and explicit values of third level.

2.2. Problems.

The general problem with *BTC* based methods is that they do not address the problem of mip-mapping. 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 [CDH05], thus requiring intensive memory communications. It is not clear how to reduce the insignificant coefficients of wavelet transform to obtain the random access without severely reducing the compression ratio as for example in hierarchical approach presented by Pereberin [Per99]. The completely different hierarchical approach based on local fractal transform was proposed in [SR06] with higher compression ratios and less complex decoding scheme. In the case of *VQ* methods, the problem is with the indirect data access and codebook handling. Each reference to the texel requires at least two memory transactions. The problem is especially magnified in the methods which use more than one codebook [KPK00] [Sch03][TZQB05].

3. The GPU Hierarchical Texture Compression.

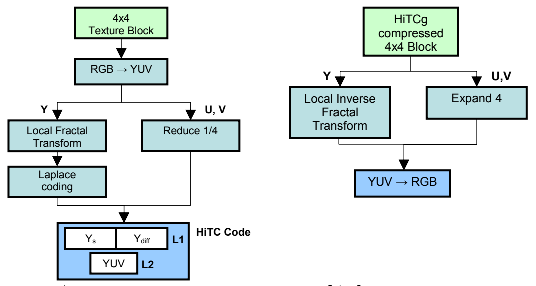
Taking into account the limitations of the nowadays GPU's we are proposing a new version of hierarchical compression algorithm introduced earlier in [SR06] we call *HiTCg*.

3.1. *HiTCg* Compression.

As opposed to HiTC [SR06] algorithm the proposed new *HiTCg* compression algorithm for GPU does not apply the local fractal transform to chrominance texture data. This allows for fast decompression since we do not have to decode two chrominance texture channels (*U, V*). In the process of the compression the chrominance values are only sub-sampled. This simplification was possible due to the fact that the reduction of the chrominance data has little impact on final image quality as opposed to the luminance. Thus, in the GPU decompression step we only need to apply local fractal transform to luminance channel. We observed that this led to three time faster decompression as opposed to the base scheme implementation [SR06].

Generally, the compression phase is composed of the following steps (Figure 1a):

1. Conversion of the texture to *YUV* color space (as in [SE02] – floating point version).
2. For *Y* component apply:
   a. Partitioning of the texture to *4x4* blocks.
   b. For each *4x4* texture block apply local fractal transform [SR06].
   c. The resulting coefficients code by one level of Laplacian pyramid.
3. For *U* and *V* component apply:
   a. Low-pass filtering operation to reduce the component size to ¼.



a) compression *b) decompression*

Figure 1: *HiTCg algorithm.*

and the decompression phase is composed of (Figure 1b):

1. For referenced 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.

3.2. Block structure.

We decided to use rectangular *RGBA* texture format with *8bit* texel depth (Figure 2).

mip level	**L2**			**L1**							
Parameters	y_d	u_d	v_d	s_0	s_1	s_2	s_3	$diff_0$	$diff_1$	$diff_2$	$diff_3$
bit res.	8	8	8	2	2	2	2	8	8	8	8
texture components	R	G	B	A				R	G	B	A

Figure 2: *HiTCg Block structure*.

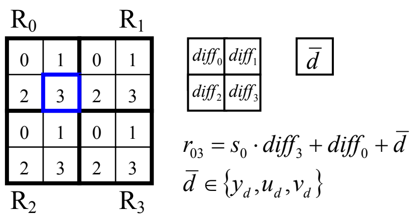

R_0
R_1
R_2
R_3
0 1 0 1
2 3 2 3
0 1 0 1
2 3 2 3
diff_0 diff_1
diff_2 diff_3
d̄
$r_{03} = s_0 \cdot diff_3 + diff_0 + \bar{d}$
$\bar{d} \in \{y_d, u_d, v_d\}$

Figure 3: *HiTCg Block addressing. Decoding of texel r_{03}.*

We use the uniform quantization for scaling parameters $\{s_i\}_{i=0}^{3}$ in the range [0.0, 1.5] [Ni97][SR06], which are represented by *2bits* $(s = \{0, 0.375, 0.75, 1.125\})$. The difference terms $\{diff\}_{i=0}^{3}$ are stored on *8bits* [SR06].

The normalization of difference term is performed to effectively exploit the *8bit* texture values and to avoid negative values. It is done prior to packing by first computing bias and scale parameters:

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

then the normalized values are computed by:

$$diff_{norm} = \frac{diff - bias}{scale} \qquad (2)$$

The block structure is represented in total by *64bits*, which corresponds to two *RGBA* texture values and additionally for each texture we need to store bias and scale values as texture header parameters. This allocation scheme is not optimal for the compression, since we do not take into account the difference terms $diff_{norm}$ quantization. They are characterized by low variance [BA83] and thus they can be quantized coarsely. We experimented with other allocations schemes [SR06] which can significantly increase the compression ratios and are perfectly suited for hardware implementations. But in the case of the GPU implementation, 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. Nowadays *GPU* are not equipped with integer processing units and all bit-wise operations are done on the floating point numbers and therefore require extra computations.

3.3. *HiTCg* Decompression.

The rectangular texture sampler object, texture coordinates and scale/offset are the input data to our algorithm (Figure 4). The output constitutes the linearly interpolated texel values. The decompression algorithm is executed in the pixel shader units.

Generally, the *HiTCg* decompression algorithm can be divided into following steps:

1) *For given texture coordinates find the HiTCg address and compute the mip-level.*
The *HiTCg* address is composed of two parts:
$$\langle block, index\rangle_{hitc} \rightarrow \{\langle h.x, h.y\rangle_{block}, (i,j)_{index}\}$$
   a. $\langle h.x, h.y\rangle_{block}$ - *HiTCg* block coordinates are used to access the *HiTCg* block data in the compressed texture,
   b. $(i,j)_{index}$ - indices are used to access the difference terms and scaling parameters in the fetched *HiTCg* block (Figure 2):

The mip-level is computed in the pixel shader (Figure 5):
$$\langle low, frac\rangle_{MipLevel} = MipLevel(texRCoord)$$
The *low* value indicates the lower mip level and *frac* value is the fractional part which is used to interpolate the values between mip levels in the *3* level mip-map mode.

2) *Fetch the HiTCg block data from the texture.*
3) *Unpack the parameters from the HiTCg block.*
Since, all the texture values which are accessed in the pixel shader are normalized, we need to perform additional step to unpack the scaling parameters and difference terms. In the case of scaling parameters it is required, to unpack the data from the byte and multiply it by the quantization step:
$$s_{deq} = s_{quant} \cdot Q_{step} \qquad (3)$$
$$diff = diff_{norm} \cdot scale + bias$$
then decode the texel value on the basis of mip level, and convert the *YUV* texel to *RGB* color space [SE02].
4) *Apply filtering.*
We have implemented two simple filtering modes: nearest neighbor and *3* level mip-mapping due to the limitations of GPU architectures. In nearest neighbor mode we omit the step related to computation of mip-map level and after unpacking the parameters we directly compute the texel value (Figure 4). In *3* level mip-mapping on the basis of the mip-map level we apply the linear interpolation to two values, which are accessed from two consecutive mip levels. It corresponds to *GL_NEAREST_MIPMAP_LINEAR* mode in *OpenGL* for the first three mip-map levels.

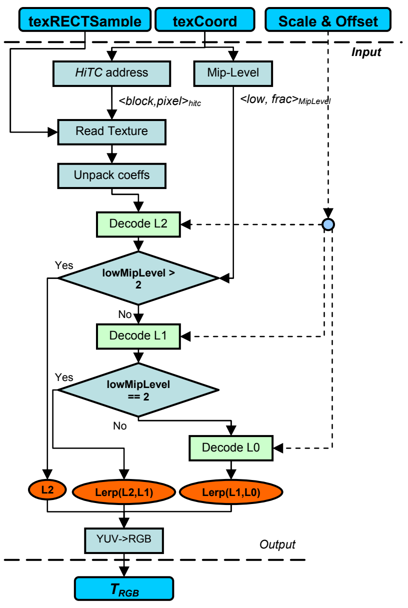


Figure 4: *HiTCg decompression on the GPU, pixel shader program.*

```
float2 MipLevel(float2 texRCoord) {
        float2 fw= max(abs(ddx(texRCoord)),
                   abs(ddy(texRCoord)));
        float m = log2(max(fw.x, fw.y));
        return float2(floor(m), frac(m)); }
```

Figure 5: *Mip-level computations on the basis of texture coordinates (texRCoord).*

4. Experimental results.

4.1. GPU decompression.

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

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

1) nearest neighbor

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

2) 3 level mip-mapping

***fps** – frames per second (peak performance)

Table 1,2: Decompression performance.

The values in the table represent the peak performance of our algorithm. The problem with the *GPU* implementation is that the pixel shader units are *SIMD* processors and in our algorithm we need to index the unpacked block data. It has obvious performance influence since we use conditional instruction. If the condition is differently evaluated by some processors then all the computations must be repeated. It can be noticed in our algorithm with decompression speed oscillations in the range of 25%.

4.2. Reconstruction results.

We have done many experiments to validate our new solution. The reconstruction quality of our algorithm was evaluated on the set of the standard images used in image compression field (Figure 6). More tests, results and examples are provided on the web page [WWW06]. For *HiTCg* block, the compression ratio is $C_R \cong 9:1$ (measured for three levels of mip-map: 4x4, 2x2, 1x1). The compared compression algorithms are characterized by compression ratios $C_R = 6:1$ (*DXTC*[INH99], *FXT1*[3df99], *PVRTC*[Fen03]), the only difference is PVRTC2 which has $C_R = 12:1$. 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*. The reconstruction quality is better than *PVRTC2* which gives compression ratio $C_R = 12:1$.

5. Conclusion.

We have presented a new algorithm for texture compression 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,
- low computational complexity,
- 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 *HiTCg* block represents three levels of mip-map pyramid. Thus, in terms of the number of accesses in trilinear filtering modes, the *HiTCg* method reduces it by one third as compared to state of the art block texture compression methods such as *S3TC* [INH99], *PVRTC* [Fen03]. Thus, it constitutes better alternative to *S3TC* in hardware architectures where the filtering and the memory bandwidth are of paramount importance. The texture quality requires further research.

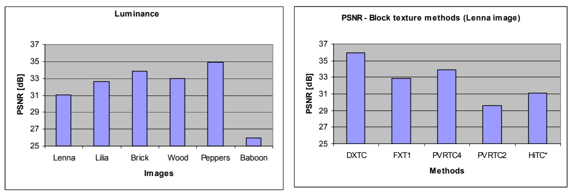
When compared to hierarchical methods based on wavelets, the texture access in our method does not require any tree walk procedures. The computational complexity was reduced to minimum with the aim of real-time application. Moreover, all the requirements outlined earlier in section 1 are fulfilled, thus making it superior for high performance rendering architectures (Table 3).

Our *GPU* implementation is strictly based on one structure in the form of the *HiTCg* compressed texture and we do not use any external structures, such as lookup tables in the form of additional textures. Thus, our method can preserve the memory bandwidth as opposed to previously proposed GPU-based methods [KE02][Sch03][TZQB05][CDH05], which require intensive memory communication.

We have evaluated two decompression modes on the *GPU* with nearest neighbor filtering and 3 level mip-mapping. The raw decompression speed of our algorithms meets the real-time requirements for *GPU* implementations. Current GPU hardware designs, as can be seen in section 3.2 - are not adjusted to the implementation of filtering modes. We hope that our work will also suggest changes in GPU designs.

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**

Table 3: *Texture compression methods comparison*



a) b)

Figure 5: *Reconstruction error, PSNR for luminance component [SE02], a) HiTCg, b) TC methods comparison*

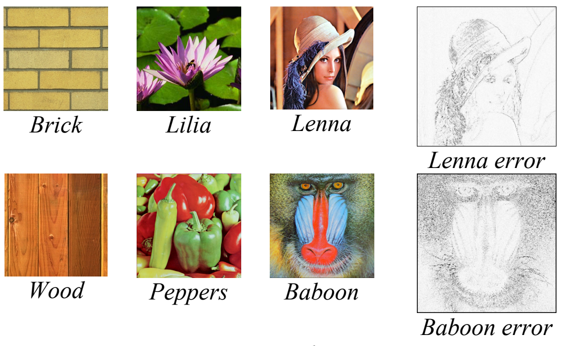


Figure 6: *Test and error images.*

References

[BA83] Burt P.J., Adelson E.H.: The Laplacian pyramid as a compact image code, IEEE TOC, 1983
[BAC96] Beers A. C., Agrawala M., Chaddha N.: Rendering from compressed textures, Siggraph 1996, July 1996.
[CDH05] 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
[CL02] Chen C.-H., Lee C.-Y.: A JPEG-like texture compression with adaptive quantization for 3D graphics application. The Visual Computer, v. 18, 2002
[DM79] Delp J., Mitchell R.: Image Compression Using Block Truncation Coding, IEEE TOC, v. 27, n. 9, 1979
[Fen03] Fenney S.: Texture Compression using Low-Frequency Signal Modulation, Graphics Hardware, ACM Press, 2003
[INH99] 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
[KE02] Kraus M., Ertl T.: Adaptive texture maps, In Proc. Siggraph/EG Graphics Hardware Workshop'02, 2002
[KPK00] Kwon Young-Su, Park In-Cheol, Kyung Chong-Min: Pyramid Texture Compression and Decompression Using Interpolative Vector Quantization, In Proc. of 2000 International Conference on Image Processing, v. 2, Sep. 10-13, 2000.
[Nin97] Ning Lu: Fractal Imaging, Academic Press, 1997
[Per99] Pereberin A.V.: Hierarchical Approach for Texture Compression, In Proc. of GraphiCon'99, 1999
[SA05] Ström J. and Akenine-Möller T.: iPACKMAN: High-Quality, Low-Complexity Texture Compression for Mobile Phones, Graphics Hardware 2005, 2005
[Sch03] Schneider J.: Kompressions- und Darstellungsmethoden für hochaufgelöste Volumendaten, Diploma Thesis (in English), RWTH Aachen, Germany, 2003
[SE02] G. Sullivan, S. Estrop.: Video Rendering with 8-Bit YUV Formats, Microsoft Developer Network
[SR06] Stachera J., Rokita P.: Hierarchical Texture Compression, In Proc. of WSCG'06, 2006
[TK96] Torborg J., Kajiya J.: Talisman: Commodity Realtime 3D Graphics for the PC, Siggraph 96, 1996
[TZQB05] Tang Ying, Zhang Hongxin, Qing Wang, Bao Hujun: Importance-Driven Texture Encoding based on Samples, In Proc. of Computer Graphics International 2005, New York, June, 2005
[Wil83] Williams L.: Pyramidal Parametrics. Computer Graphics, In Proc. of Siggraph'83, July, 1983
[WWW06] Hierarchical Texture Compression: http://staff.elka.pw.edu.pl/~jstacher/