Graphics Hardware (2003)
M. Doggett, W. Heidrich, W. Mark, A. Schilling (Editors)

Texture Compression using Low-Frequency Signal Modulation

Simon Fenney†

Imagination Technologies Ltd., United Kingdom

Abstract

This paper presents a new, lossy texture compression technique that is suited to implementation on low-cost, low-bandwidth devices as well as more powerful rendering systems. It uses a representation that is based on the blending of two (or more) 'low frequency' signals using a high frequency but low precision modulation signal. Continuity of the low frequency signals helps to avoid block artefacts. Decompression costs are kept low through use of fixed-rate encoding and by eliminating indirect data access, as needed with Vector Quantisation schemes. Good quality reproduction of (A)RGB textures is achieved with a choice of 4bpp or 2bpp representations.

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

1. Introduction

Since its introduction by Catmull[1], texture mapping has become ubiquitous in computer graphics. Today, even low-cost consumer devices, such as games consoles, are expected to support real-time texturing in hardware. In any such system, two related problems must be addressed. The first is simply the cost of storing these textures in memory. Despite the decreasing cost of RAM, consumer 3D systems still only have a relatively small amount of memory available for the storage of textures and this resource can rapidly become filled. This is especially aggravated by the use of true colour textures which typically have 24 or 32 bits per texel – eight bits for each of the Red, Green, Blue, and optionally Alpha (translucency) components.

The second, and more critical problem, is that of data transfer. During the rendering of the 3D scene, a considerable amount of texture data must be accessed and, in a real-time system, the memory bus can soon become a significant performance bottleneck. Texture filtering can increase the demand on texel fetches and, although a texel cache can eliminate some of the external fetches, it clearly has to be of finite size and capability.

One approach to alleviate both of these problems is to use some form of image compression. Several such systems are employed in image processing and transmission but, as Beers et al.[2] point out, few are suited to real-time 3D computer graphics texturing. They list four factors that should be considered when evaluating a texture compression scheme and these are worth reiterating:

> *Decoding speed*: The accessing of texture data is a critical path in the texturing operation and so the speed of decode is of paramount importance. The decode algorithm should also be relatively simple to keep down the cost of hardware.
>
> *Random Access*: As objects may be oriented and obscured arbitrarily, it is therefore a requirement to be able to quickly access any random texel.
>
> *Compression Rate and Visual quality*: Because the important issue is the quality of the rendered scene rather than the quality of individual textures, some loss of fidelity in the compressed texture is tolerable. Lossy compression schemes offer higher levels of compression than their lossless counterparts, and so the former are generally preferred.
>
> *Encoding speed*: With the exception of, say, dy-

† simon.fenney@powervr.com

namic environment maps, the majority of textures are prepared well in advance of the rendering. It is therefore feasible to employ a scheme where encoding is considerably slower than the decoding.

The main problem with the general image compression schemes, e.g. JPEG, is that they do not permit direct random access of pixel data within the compressed format because the per-pixel storage rate varies throughout. As a result, many of the texture compression schemes that have been proposed and/or implemented employ 'fixed rate' encoding.

Ideally, decompression should also be fast and inexpensive to implement in hardware – this usually eliminates many of the 'transform coding' systems such as the discrete cosine transform, DCT. When quantised, such transforms occasionally exhibit compression artefacts, such as 'ringing', around sharp edges that occur, particularly, in textures containing lines or text.

2. Previous Work on Fixed Rate Texture Encoding

Perhaps the most commonly used class of texture compression system is that based on colour palettes, which are a form of vector quantisation, or VQ. Colour palettes were originally used as a means of reducing the memory and bandwidth costs of video frame buffers[3] and so can address the same issues in texturing. This method has been used in many applications ranging from flight simulators down to computer games consoles. In such schemes, each texel is replaced by a small number of bits, typically 4 or 8, which is an index into a table of colours, or palette, with 16 or 256 entries respectively. Numerous methods for converting an original 'true colour' image to this palettised format exist, including Heckbert's[4] and Wu's[5].

There are, however, several drawbacks to textures compressed with a colour palette system. The first is the indirection involved in decoding each texel – competition for the memory bus and the latency of access on today's graphics accelerators can be relatively high and chaining two accesses only exacerbates the problem. Solutions that reduce the time delay and bandwidth cost of the double read include bringing the colour table 'on chip' or using dedicated RAM, but these too incur additional penalties. For example, each time a new texture is accessed, a dedicated colour table must be reloaded. Alternatively, a global palette could be used for all textures but this would compromise the quality of the compressed images. Perhaps more damaging though, are the facts that the storage and bandwidth savings for 8bpp are not outstanding, and that the quality of 4bpp textures can sometimes be poor.

Further cost complications arise due to texture filtering. If we just consider the case of texture magnification, a filter must be applied to several sampled texels to avoid the texture appearing 'blocky'. Even restricting this to a simple bilinear filter still requires a weighted sum of a 2x2 group of neighbouring texels, which need to be supplied in parallel for maximum performance. In a system incorporating colour palette textures, this would involve fetching the indices for all four texels and then reading each texel's corresponding colour in the table. It can be appreciated that, unless multi-ported (i.e. more expensive), the palette/colour table RAM could easily become a bottleneck. This situation only becomes worse when more sophisticated filtering, e.g. trilinear MIP mapping[6] is employed.

The low compression rate of palettised textures was effectively addressed by Beers et al.[2] by using a more sophisticated form of vector quantisation. They simulated a system that, depending on MIP map level, replaced blocks of 4x4, 2x2, or 1x1 texels with indices into the corresponding codebook. This achieved significant reductions in the texture footprint, i.e. 1bpp or 2bpp textures, with an acceptable loss in quality. Inspired by their research, a simpler VQ system, without the MIP map level sharing, was co-developed by the author and implemented in the Sega Dreamcast™ games console hardware.

Although these forms of VQ do offer high levels of compression at reasonable quality, they still suffer from needing two memory accesses. Furthermore, the size of the look-up table is much greater than that of palettised textures and so any internal storage or caching of the table is correspondingly more expensive. However, unlike the palettised texture system, supporting bilinear filtering with 2x2 VQ is actually not unduly costly as the codebook can be split so that each texel in each 2x2 code can be addressed independently.

Block Truncation Coding, or BTC, as presented by Delp and Mitchell[7] is an alternative compression system that avoids the indirection of VQ. In BTC, a grey-scale image is subdivided into non-overlapping rectangular blocks, say, 4x4 pixels in size, and each block is then processed independently. Two representative values, e.g. each 8 bits, are chosen per block and each pixel within the block is quantised to either of these two values. The storage cost for each block in the example is therefore 32 bits, thus giving an overall rate of 2bpp. Because the blocks are independent, this simplifies the compression and decompression algorithms, however it can potentially lead to artefacts at block boundaries.

Campbell et al.[8] developed Color Cell Compression, CCC, which replaces the block representatives in BTC with indices into a palette, thus encoding colour images at 2bpp. Despite the need for a colour palette and some example images showing evidence of colour banding, Knittel et al.[9] suggested using CCC in a texturing system.

Iourcha et al.[10] further adapt the BTC/CCC methods to improve colour data encoding. Their system, known in the industry as S3TC™ (or DXTC™ within Microsoft's DirectX 3D interface), also uses 4x4 texel blocks. As with BTC, each block is completely independent of every other block. Each pixel in the 64-bit block is encoded with two bits

which selects one of four colours. Two of these colours are stored in the block as 16-bit RGB values while the remaining pair are derived directly from the stored colours. These additional colours are usually 1:2 and 2:1 linear blends of the main representatives but, in some blocks, one of the indices can be used to indicate a fully transparent, black pixel for so called 'punch-through' textures. The decoding is simple enough that it can be done 'on-the-fly' without the need to retain decompressed texels.

The quality of the S3TC system is generally much higher than that given by CCC and it has the advantage of eliminating the separate colour palette but these gains are achieved at the cost of doubling the storage to 4bpp. The S3TC system works well because the colour space of each 4x4 block often has a very dominant axis which can be approximated by the linearly-arranged, representative colours. Furthermore, the signal to noise ratio in each local area will typically stay constant, since large errors in the representation will usually only occur when there are correspondingly large changes in the image. With some textures, however, S3TC can exhibit artefacts at block boundaries or where the colours change dramatically. Nevertheless, S3TC has become a leading texture compression method.

If we now consider the bilinear filtering of an S3TC compressed texture, as shown in Figure 1, we see that in just over half the cases, the required 2x2 source texels can be sourced entirely from a single S3TC block. For the remainder, either two or four texture blocks must be accessed. A real-time system that guarantees production of one bilinearly filtered screen pixel 'per clock' should thus be able to access and decode pixels from four blocks in parallel.

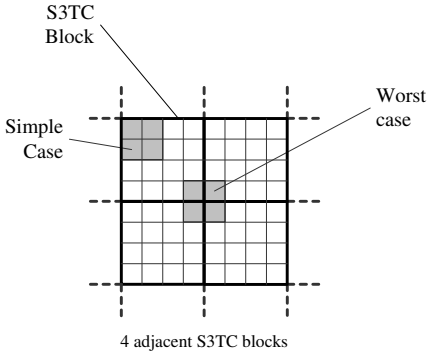


Figure 1: *Block requirements for various bilinear filtering cases.*

To address the situations where the texels in a block do not map conveniently to a line segment, Levkovich-Maslyuk et al.[11] allow colours to be chosen from an RGB tetrahedron. To allow the greater selection of colours, without the size of the indices 'ballooning', they partition pixels within the block and create sub-palettes for each partition.

One recent system that has instead used a transform coding approach is that of Pereberin[12], which again uses blocks of 4x4 texels. By assuming that box filtering is used in the production of MIP map levels, it also encodes texels from three adjacent levels simultaneously. Each block is mapped into the YUV colour space and then two passes of a 2D Haar wavelet transform are applied. All of the low and mid-frequency coefficients are stored for the Y, U, and V, channels, but only the five most significant of Y's highest-frequency coefficients are kept while the remainder and all of those for U and V, are assumed to be zero. In total, 96 bits are used for each block which therefore gives an overall compression rate of $\approx$ 4.6bpp. Although the choice of YUV should work well for natural images, there *may* be some problems with high saturation graphics. The separate encoding of the components also fails to capitalise on the frequent correlation of the channels (as can be seen by the effectiveness of S3TC).

Finally, one scheme that mixes a block-based system with a palette-like approach has been presented by Ivanov and Kuzmin[13]. Each block stores at least one base colour but a larger local palette is implied by allowing access to a certain set of the neighbouring blocks' colours. For example, the local palette for a particular block may have access to the base colours from an additional three neighbours, say, from the the block to right, the one below, and the one to the 'below and right'. In this example, each texel would thus be replaced by a two bit index accessing one of the four available base colours.

3. Research Aims, Constraints, and Observations

The main aim of the research was to develop a new texture compression technique that would provide 'good quality' compression of (A)RGB images with data rates of around 4 or 2bpp. The target platforms included very low cost devices, potentially PDAs and mobile phones, and so the scheme also had to be inexpensive to implement. (Although less significant, cost is still important in the 'desktop' environment where the increased gate budget is offset by the requirement to support multiple pipelines, multiple texture layers, and complex texture filtering.)

It is well known that low-pass filtered signals are often good approximations of the original signal, and so some initial experiments using wavelets[14,15] were conducted. In particular, (bi)linear and (bi)cubic wavelets were tested and showed some promise as a means of representing the textures. The difficulty lay with an efficient method of reducing the wavelet coefficients for the high frequency information. A Huffman-based encoding with truncation of insignificant values within blocks was tried, but it proved difficult to achieve suitable compression within a small pixel block 'window', e.g. 4x4 texels. Furthermore, for some texture types, such as those containing lines or text, performing such transformations often made compression *more* difficult.

For such images, the simplicity of the direct encoding used in BTC/S3TC compression schemes was more appealing.

It occurred to the author that, although the BTC-based encoding schemes benefit from the similarity of colours in localised regions, they do not take advantage any correlation of actual texel position and colour value. Furthermore, most of these methods treat every block independently which, although simplifying compression, fail to capitalise on the frequent similarity of adjacent blocks, and can also lead to discontinuities between blocks.

The requirement of texture filtering was also a prime concern. Since bilinear filtering is often the basic building block of more advanced schemes such as trilinear and some implementations of anisotropic filtering, another aim of the design was to have the chosen decompression process produce a set of 2x2 neighbouring texels 'almost' as cheaply as an individual texel.

3.1. Initial trials

Given the resemblance of low-pass filtered signals to their source textures, research focused on using continuous 'low frequency' signals as the basis for the stored representation rather than a set of constant per-block colours. The aim was to use two (or potentially more) low-resolution images, A and B, generally between $\frac{1}{4}$ and $\frac{1}{8}$ of the dimensions of the target texture, along with a full resolution 2bpp modulation signal, M.

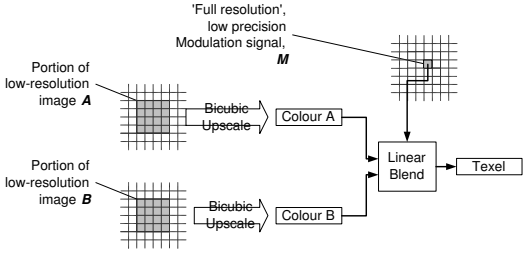


Figure 2: *Decompression steps for an individual texel.*

The decompression process for the initial test system is shown in Figure 2. To produce an individual texel, two corresponding sets of 4x4 source texels were identified and read from each of the A and B low resolution images, These two sets were then bicubicly up-scaled to produce two base colours, *ColourA* and *ColourB*, for the target texel. The modulation value for the target was read from M, and was then used to linearly interpolate *ColourA* and *ColourB* to produce the decompressed texel. In the initial design, the low-frequency data was stored 'separately' from the modulation information allowing the separate data elements to be read in bursts and cached independently. Figure 3 shows the process applied to actual data.

It is interesting to note that S3TC can also be viewed as a variant of this scheme in which a step function is used to upscale the A and B images.

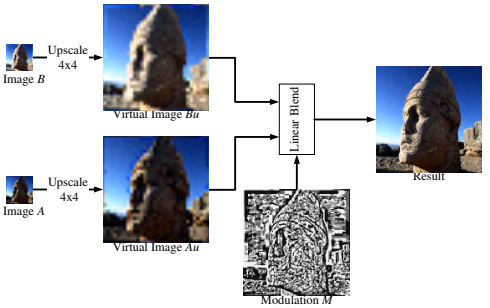


Figure 3: *Example decompression with new scheme.*

Although the initial results (targeting 2.5∼4bpp) were very promising, from a hardware standpoint the costs of bicubic interpolation and of independent colour and modulation caches were considered too great for low-budget hardware. The scheme was thus simplified to use bilinear upscaling which then allowed the low-resolution colour data to be interspersed with the modulation information.

4. 4bpp Texture Mode: Decompressor

As with VQ and S3TC, it is much easier to understand the system by describing the decompressor, which is designed to produce the 2x2 set of texels required for subsequent bilinear filtering. The 4bpp mode is also slightly simpler than the 2bpp, and so will be described first.

The data for the compressed texture consists of a $\frac{P}{4}$ by $\frac{Q}{4}$ set of 64-bit blocks, where P and Q are the dimension of the texture. (The 64-bit block size was chosen to be 'affordable' in a low-cost solution.) To minimise page breaks, the $\frac{PQ}{16}$ blocks of the texture are arranged in Morton order.

Each block contains two colour values, corresponding to a pixel from each of the A and B low frequency signals, one mode bit, and modulation data for a set of 4x4 texels. This is shown in Figure 4

Although this arrangement bears some similarity to that used in the BTC-based schemes and, in particular, S3TC, there are some differences. Unlike S3TC, each of the A and B base colour entries can be independently set to be either fully opaque or to have variable levels of alpha. Although colour precision is sacrificed in the latter case, this should be less critical when alpha blending is used.

4.1. Optimising the Bilinear Up-scale

The block's base colours, along with selected neighbouring colours, must be up-scaled by a factor of four in each dimension to produce per-texel A and B colours. To increase hardware re-use, the two low precision colour modes are initially

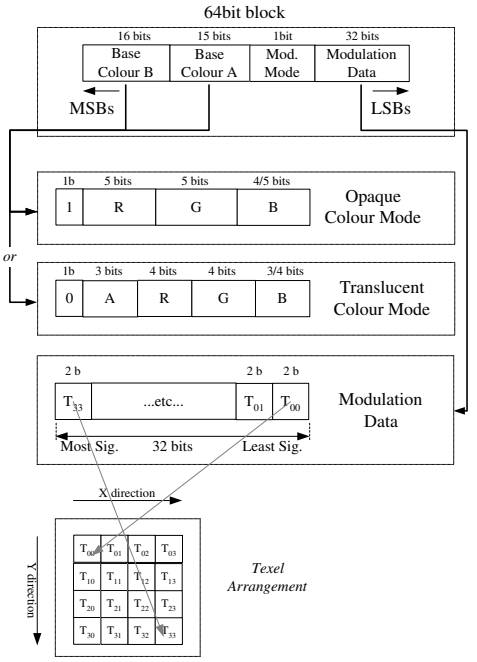


Figure 4: *Structure of 4bpp texture blocks.*

expanded to a uniform ARGB 4555 format as follows: For the red, green, and blue channels, the standard approach of replicating the MSBs is taken. For alpha, the value is either set to 0xF for the opaque case or, when translucent, a zero bit is appended to the stored, 3-bit value.

To further minimise the hardware costs, rather than converting the 4555 colours to their 8-bit equivalents (again via replication of the MSBs) and then bilinearly up-scaling, the decompressor reverses the order of operations. This is valid since the conversion is equivalent to a multiplication by a constant (i.e., $\frac{17}{16}$ or $\frac{33}{32}$) and this factor can be moved outside of the linear filter. The bilinear filtering is thus applied to the 4- or 5-bit values and the results (including the resulting additional fractional bits) are finally 'converted' to '8-bit' by multiplying by the appropriate factor.

Additional savings in the hardware are made by treating each of the bilinear up-scales as a bilinear surface patch. Repeated binary subdivision, for example of the four *A* source values, will produce the four *A* colours needed for a 2x2 block of texels. In all, this requires only a handful of multiplexers and eight 'add' units per colour channel.

The 'phase' of the bilinear up-scale is also quite important for both quality and decode efficiency. For compression symmetry, it is desirable for the base colours to be representative of the centres of each block, i.e. the bilinear upscale would 'produce' a base colour at the middle of the corresponding block. For decompression efficiency, it is better to have the base colour 'land' on the centre of a texel, as this will keep the weighting factors small. As a compromise, the bilinear up-scale evaluates to the base colours at the texels immediately to the right and below of the block centre.

Another immediate benefit of this choice is shown in Figure 5.

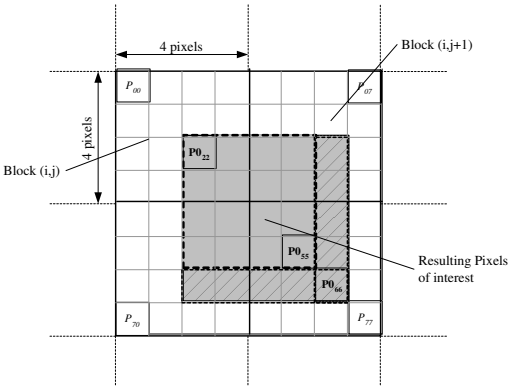


Figure 5: *Region of texels that require data from exactly the blocks (i,j) thru (i+1, j+1).*

Within the grey region of texels, it can be noted that any set of 2x2 texels, as needed for a bilinear texture filtering operation, requires data from exactly 2x2 data blocks. This corresponds to the worst-case situation for schemes such as S3TC and, since all combinations of 2x2 texels fall into one such grouping, this therefore imposes no extra cost to decompression. At the borders of the texture, the programmers/artists can elect whether to 'wrap' the bilinear filtering, or to repeat the representative values.

4.2. Modulation Modes

The modulation data in each block is interpreted in one of two ways depending on the value of the corresponding 'modulation mode' flag. In the standard setting, '0', each texel's 2-bit modulation value implies the following choice of blend value:

Bit Pattern	Modulation Value
00	$0/8$
01	$3/8$
10	$5/8$
11	$8/8$

The reasons for this set of values are twofold. Firstly, the numerators are small and so the blending operations are inexpensive to implement in hardware. These values can be contrasted to the $\frac{1}{3}$ and $\frac{2}{3}$ values specified for S3TC, although Iourcha et al. do suggest approximating these blend values. Secondly, the distribution of modulation values was

often found to follow the shape of a normal distribution, and so it seemed advantageous to bias the representatives slightly towards the centre.

If the mode flag is set to 1, then a form of 'punch-through alpha' texturing is enabled for the block of pixels. Despite the numerous drawbacks of such texture formats, this was included for compatibility with older applications. The modulation values are then interpreted as:

Bit Pattern	Modulation Value
00	0/8
01	4/8
10	4/8♣
11	8/8

♣For the special '10' case, the alpha channel is forced to be zero. Although it is common to set the colour of punch-through texels to black (eg. S3TC), this tends to result in unpleasant halos appearing around the edges of objects when filtering is applied. For this reason, the average of the A and B values is chosen instead.

5. 4-bpp Texture Mode: Compressor

Although the decompression process is relatively straightforward, that of the compression is far more involved. The current algorithm will be described but this is an area of ongoing research.

The process begins with the generation of initial values for the A and B signals, and then performs refinement iterations on these values.

5.1. Generating Initial Values

The first step applies a low-pass filter of the data. This is based on a linear wavelet transform[15], retaining only the low frequency coefficients, followed by the 4x4 bilinear upscale. The initial wavelet transform is modified so that it 'centres' the representative values to match the bilinear upscale.

The delta image, i.e. the difference between the original texture and the filtered data, is computed and then an 'axis image' for the deltas is generated. Several approaches to generate this data have been evaluated, and two will be described here:

The first method finds the principal axis of sets of overlapping regions of (delta) texels centred on each texel. As sometimes used in vector quantisation[5] or the S3TC compressor[10], this is done by computing the covariance matrix of each region's texels and from that deriving the principal eigenvector[16] of the matrix. The second, and far simpler approach is to optionally 'flip' the direction of each delta vector so that it more closely matches the orientation of its neighbours.

Note that this axis image is different to the principal axes required for VQ or S3TC, in that the vectors are computed from the delta signal rather than to the original texels and, for want of a better term, are 'perpendicular' to the original texels.

After the axis image has been filtered, it is used to generate initial full-resolution prototypes for the low-frequency A and B representatives. One simple approach is to take the per-texel dot product of the delta value with its corresponding axis direction. Depending on the sign of the result, the corresponding texels in the prototype A and B images are alternatively set to the original texel colour and the filtered image value minus the delta. This gives two images which 'bound' the original filtered signal. These prototypes are then themselves filtered to generate the initial A and B candidates.

5.2. Iterative improvement

The compressor then begins an iterative refinement process. In each pass, the appropriate, quantised modulation values for the texture are computed from the current A and B candidates. These, in turn, are used to produce newer 'optimum' A and B signals by solving for the minimum least squares error using Singular Value Decomposition, SVD[16]. Because this algorithm is too expensive (i.e. $O(n^3)$) to run on the entire texture in a single pass, the compressor steps through overlapping windows of pixels, each approximately 3x3 blocks wide, that are centred around 2x2 blocks. This is shown in Figure 6.

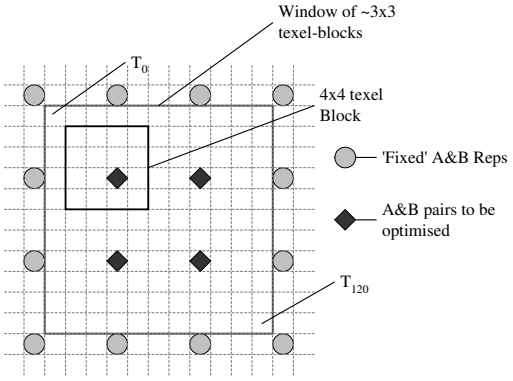


Figure 6: *Optimising four sets of representative colours for a window of 121 texels.*

This region of 121 texels is affected by exactly 16 pairs of A and B colour values. The outer set of 12 are assumed to be held fixed while the refinement step finds the optimum values for the middle four pairs. A matrix, M_W, is constructed with 121 rows, corresponding to the texels, by 8 columns, corresponding to the 4 pairs of A and B values to be optimised, i.e.,

$$M_W = \begin{bmatrix} w_0^{A_{TL}} & w_0^{B_{TL}} & w_0^{A_{TR}} & w_0^{B_{TR}} & \dots & w_0^{B_{BR}} \\ w_1^{A_{TL}} & \dots & & & & \\ \vdots & & & & & \vdots \\ w_{120}^{A_{TL}} & \dots & & & \dots & w_{120}^{A_{BR}} \end{bmatrix}$$

Each entry in M_W contains the 'weight', w_n^{val}, which would be applied to the corresponding A and B values when decoding texel T_n – this takes into account the relative position of the texel and the chosen modulation value, and so...

$$M_W . \begin{bmatrix} A_{TL} \\ B_{TL} \\ A_{TR} \\ \vdots \\ B_{BR} \end{bmatrix} = \begin{bmatrix} P_0 \\ P_1 \\ \vdots \\ P_{120} \end{bmatrix}$$

...where the P_n values are the colours of the source image texels adjusted to remove any contribution due to the fixed 12 pairs.

The SVD algorithm, *in effect*, computes M_W^{-1} such that multiplication by the P_n vector produces the optimum values for the eight centre representatives. These new values are then used to update the existing A and B texels before repeating the iterative process. The current compressor usually applies 2 to 4 of these passes.

6. 2BPP Mode

The 2bpp mode is similar, in principle, to the 4bpp except that the low resolution A and B images are scaled by a factor of eight instead of four in the horizontal dimension, and that the modulation data for 8x4 texels is packed into the space that was originally occupied by only 16 texels.

While the scaling of the A and B data is not usually too critical to the image quality, a reduction in modulation accuracy can have a dramatic impact. To address this, the 2bpp mode also uses two modulation modes. The first is a direct 1bpp encoding which is very similar to CCC's. This is useful for regions containing hard edges, such as those with text.

The second mode uses 2-bit modulation values for every second texel, arranged in a chequerboard pattern. The remaining texel modulation values are then obtained, according to one of three sub-modes, by averaging the neighbouring two vertical, two horizontal, or all four modulation values.

7. Comparative Results

Figure 10 shows the behaviour (including flaws in the current compression tool) of the new method on a 'typical' spread of texture types. Results from S3TC (using the ATI™ 'compressenator' with default settings) are also shown for comparison purposes. Some remarks on these results are given below:

stormsky	This image, similar to some used in popular games, is well suited to the new technique whereas S3TC can display some blockiness.
lorikeet	The multiple colours in each region are handled quite well by the 4bpp mode, but some loss of fidelity is evident in the 2bpp variant.
cottage	This image illustrates some serious flaws with the current compressor especially with the 2bpp mode. Inappropriate modulation modes have been chosen, leading to the crenellated pattern around the window frames. An example (2bpp-mod) where that particular mode has been forcibly disabled in the compressor shows a marked improvement in that region.

Note that this particular set of images was chosen/generated because of potential copyright concerns with some other popular images.

Despite RMS error not being an ideal measure of perceived quality, Table 1 gives numeric compression errors for 'Lena', the first five images from the Kodak™ PhotoCD PCD0992[17], and the images in Figure 10. Results from the 2bpp VQ mode used in the Dreamcast console are also included. For compatibility with that compressor, the Kodak images have been cropped to the top left 512x512 pixels.

Image	S3TC	DC-VQ	4bpp	2bpp
lena	7.91	12.6	7.11	11.29
kodim01	8.31	13.5	8.98	19.40
kodim02	6.60	9.5	6.20	11.46
kodim03	5.70	10.9	5.61	11.04
kodim04	5.97	11.2	5.76	10.86
kodim05	10.50	19.7	10.59	21.92
stormsky	6.85	11.2	5.79	9.08
lorikeet	9.80	16.6	8.08	12.11
rust	11.35	17.1	10.45	18.82
nemrut	10.08	15.9	10.31	20.72
cottage	11.11	15.4	14.25	27.57

Table 1: *Comparative RMS Errors*

As mentioned, there are a number of flaws in the current compressor. It will often get trapped in local minima with poor choices of modulation values; for example, on transitions between areas of flat colour it will often 'overshoot'. A possible solution to this may be to detect areas of constant colour and force the texels to use the same modulation value.

It is 'slow' ($\approx$ 1min/image) which is why the number of iterations is limited to only a few passes even though more will decrease the error. This is partly due to it being inefficient in that it re-processes texel data which may already be represented adequately instead of concentrating on areas where the error is high.

Finally, the compressor also assumes that the textures 'tile seamlessly' as it mimics the initial, simplified hardware implementation. Clearly not all textures fall into this category and so some images will have greater than necessary errors

near the boundaries. For example, Reflecting the 'nemrut' image about the x and y axes results in a compressed RMS error of 9.98 for the 4bpp mode.

8. Future Work

With any compression scheme, the quality of the results is gated by the quality of the compressor and, as can be seen, the current compressor is in need of improvement.

It is also planned to extend the scheme to support 3 dimensional textures which, due to their size, will definitely benefit from some form of compression. The interpolation scheme naturally extends to multiple dimensions and should automatically take advantage of the increasing density of representative values that will occur with a 3D data set.

Similarly, reducing the number of colour dimensions could allow higher precision for certain texture types, such as perturbed UV maps or light maps. Other options may be to investigate some combination of the new technique with either transform coding or VQ, say, for the modulation information.

9. Conclusions

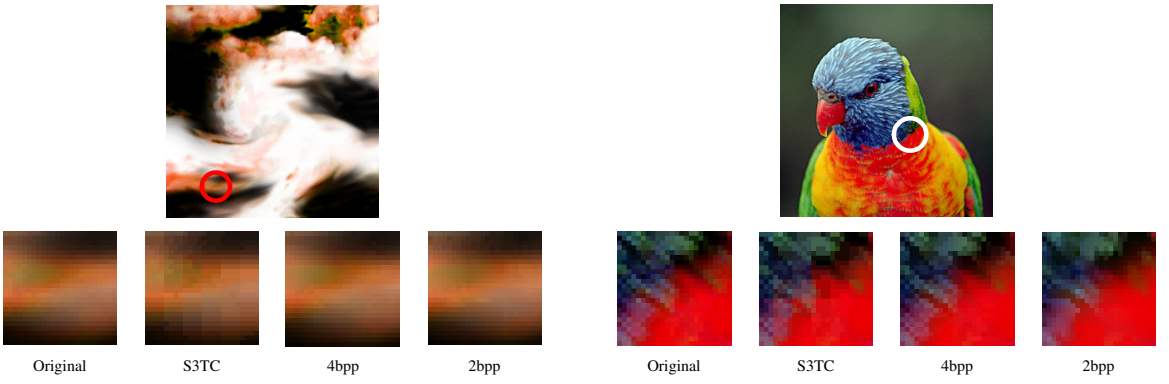
This paper has presented a new method of compressing textures that is both relatively inexpensive to implement in hardware and has the potential to produce good quality images with both 4bpp and 2bpp compression rates. The decompression scheme is optimised for texture filtering (using bilinear as the basic building block). The compression tool, however, needs more research.

10. Acknowledgements

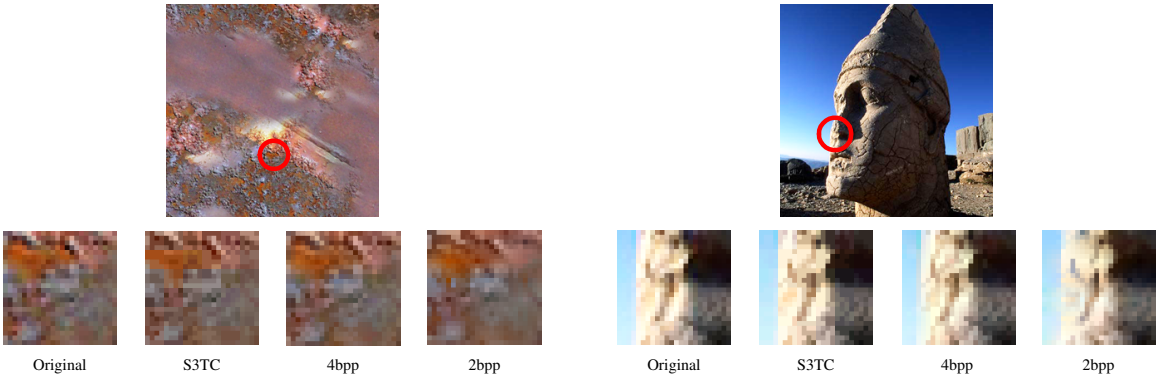
The author would like to thank several of his colleagues who gave support during the research and preparation of this report. In particular I'd like to thank Piers Barber for telling me that "it's using too many gates" and pushing me to improve the method, Carlos Sarria, Nicolas Thibieroz, and the rest of PowerVR developer relations team for testing the compressor, and Steve Morphet, Mark Butler, and Aaron Burton for kindly proof reading this paper at very short notice. Finally, the author would like to thank the reviewers for their feedback and suggestions.

References

1. Edwin Catmull. "Computer Display of Curved Surfaces", *Proc. IEEE Conf. Computer Graphics, Pattern Recognition, and Data Structures* May 1975, 11–17.
2. Andrew C. Beers, Maneesh Agrawala, and Navin Chadda. "Rendering from Compressed Textures", *Computer Graphics, Proc. SIGGRAPH* **7**(3), Aug.1996, 373–378
3. James Kajiya, Ivan Sutherland, and Edward Cheadle. "A Random-Access Video Frame Buffer", *Proc. IEEE Computer Graphics, Pattern Recognition, and Data Structures* May 1975, 1–6.
4. Paul Heckbert. "Color Image Quantisation for Frame Buffer Display", *Computer Graphics* **16**(3), July 1982, 297–307.
5. Xiaolin Wu. "Color Quantization by Dynamic Programming and Principal Analysis", *ACM Transactions on Graphics* **11**(4), Oct. 1992, 348–372.
6. Lance Williams. "Pyramidal Parametrics", *Computer Graphics* **7**(3), July 1983, 1–11.
7. E. Delp and O. Mitchell. "Image Compression Using Block Truncation Coding", *IEEE Trans. on Communications.* Vol.COM-2 (9), Sept. 1979, 1335–1342.
8. Graham Campbell et al. "Two bit/pixel full color encoding", *Computer Graphics* **20**(4), August 1986, 215–223.
9. G. Knittel, A. Schilling, A. Kugler, and W. Strasser. "Hardware for Superior Texture Performance", *Proceedings of the 10th Eurographics Workshop on Graphics Hardware* 1995, 33–39.
10. 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*.
11. L. Levkovich-Maslyuk, P. G. Kalyuzhny, and A. Zhirkov. "Texture Compression with Adaptive Block Partitions", *ACM Multimedia 2000* Nov. 2000.
12. A.V. Pereberin. "Hierarchical Approach for Texture Compression", *Proceedings of GraphiCon '99* 1999,195–199.
13. Denis Ivanov and Yevgeniy Kuzmin. "Color Distribution - A New Approach to Texture Compression", *Computer Graphics Forum* **19**(3), August 2000, 283–289.
14. Eric Stollnitz, Tony DeRose, and David Salesin. "Wavelets for Computer Graphics. Theory and Applications", *ISBN 1-55860-375-1*
15. Wim Sweldens and Peter Shroeder. "Building your own Wavelets at Home", *Journal* Technical Report 1995:5, Industrial Mathematics Initiative, Department of Mathematics, University of South Carolina
16. Press, Teukolsky, Vettering, and Flannery. "Numerical Recipes in C", *ISBN 0-521-43108-5*
17. http://sqez.home.att.net/thumbs/Thumbnails.html



(a) stormsky (b) lorikeet



(c) rust (d) nemrut

(e) cottage

Figure 10: Enlargements of S3TC, 4bpp, and 2bpp modes