ftc — Floating Precision Texture Compression

Philipp Klaus Krause

Albert-Ludwigs-Universität Freiburg
Email: krauseph@informatik.uni-freiburg.de

Abstract

This paper presents new fixed-rate texture compression systems for RGB and RGBA images, which use variable precision differential encoding for the color codewords, resulting in reduced compression artifacts, in particular for smooth color variations. Experiments using hundreds of textures show that they achieve higher quality than the commonly used S3TC at the same compression ratio or (for RGBA images) similar quality to S3TC at twice the compression ratio. In hardware implementations a large part of decompression hardware can be shared with existing common systems, minimizing the amount of additional hardware needed.

1 Introduction

In computer graphics achieving high visual quality typically requires high-resolution textures. However the desire for increasing texture resolution conflicts with the limited amount of graphics memory and bandwidth available.

Memory bandwidth is the most important aspect of graphics system performance today [1]. Most of it is consumed by texture accesses; the common trilinear texture filtering accesses 8 texels during each texture read. Increasing memory bandwidth is expensive. Increasing memory clock frequency requires more expensive and power-consuming memory chips. Increasing the number of data lines often results in more memory chips increasing cost and power consumption, too. Especially for embedded systems this may not be an option.

Texture compression can help to achieve higher graphics quality with given memory and bandwidth or reduce memory and bandwidth consumption without degrading quality too much.

Texture compression systems must allow fast random access to texels. A fixed compression ratio eases address computations. Since computer graphics systems are typically highly pipelined it is desirable to keep the number of indirections during texture lookup low. A texture compression system should preserve the locality of reference to work well with texture caches. Most texture compression systems achieve this by splitting the texture into blocks of equal size, which are compressed independent of each other. The decompression algorithm should be simple, fast, and easy to implement in hardware. Generic image compression systems like JPEG or PNG do not fulfill these requirements.

In many texture compression systems the compressed texture contains color codewords, from which the colors of individual texels are derived. Often the quality of compressed textures is limited by the precision of the color codewords if those codewords are near to each other, but limited by other aspects of the compression system when the color codewords contain colors that are far from each other in color space.

The approach presented herein, ftc, improves image quality by using variable precision in the color codewords, sacrificing precision when the codewords are far apart to increase precision when the coded colors are near: One color codeword is stored directly, while the other one is stored as a signed difference to the first. When the color codewords are near to each other more bits are used for the first one, while the difference is stored with less bits and sign-extended before being added to the first codeword to retrieve the second.

The new texture compression systems have been designed to share most of the decompression hardware with S3TC implementations. Since S3TC is the standard texture compression system today it will have to be supported for the foreseeable future. Thus the additional hardware needed to support the new texture compression systems in graphics hardware is minimal.

Compared to the de-facto standard S3TC ftc provides superior image quality at the same compression ratio or comparable quality at twice the compression ratio.

2 Related research

The earliest approaches were adaptions of generic lossy compression systems to texture compression: Indexed color and vector quantization, presented in section 2.1. Since they suffered from low compression quality and an indirection during memory access later texture compression systems that compress blocks of texels independently were developed; these systems are presented in section 2.2. High dynamic range (HDR) texture compression developed nearly a decade after the introduction of high dynamic range digital imaging by Debevec [10], [9]; these systems can be found in section 2.3. Though presented in an extra section these are block-based, too.

Lossless texture compression systems [17], which do not have a fixed compression ratio have not found widespread use since they they require relatively complex hardware to implement.

2.1 Indexed color and vector quantization

Probably the earliest image compression system was indexed color. Research into color quantization algorithms for RGB images was pioneered by Heckbert [16]. A per-texture color palette is chosen and texels are approximated by using the nearest color from the palette.

Indexed color can be considered a special case of vector quantization, first proposed for texture compression by Beers et al. in [5]. For textures that contain many repeating patterns vector quantization is superior to texture compression systems discovered later; see Wei [35] for modern research into this topic.

2.2 Block-based compression systems

Block-based compression systems achieve locality of reference by compressing blocks of texels independently from each other.

An early block-based image compression system is Block truncation coding (BTC), proposed by Delp and Mitchell [11] for grayscale images. It has been used for color images by encoding each color-channel separately. Color Cell compression (CCC), a generalization of BTC for color images, was proposed by Campbell at al. [8]. It was the first approach to be considered for hardware implementation as a texture compression system by Knittel et al. [18].

The most common texture compression system today is S3TC [6]. For each block of 4×4 pixels two color codewords are chosen; colors of the compressed texture lie at discrete points on the line connecting the color codewords. A variation of S3TC using a smaller block size has been presented by Akenine-Möller and Ström [2].

Another approach that has found some use is ETC, discovered by Ström et al. [29], an improvement of their earlier proposal [31]. Each 4×4 texel block is divided into two subblocks, with a color codewords stored for each subblock. Colors in the compressed subblock lie at discrete points on a line in the $(1, 1, 1)$ direction through the subblock's color codeword, allowing for more luminance than chrominance variation. A further improved, but more complex variant has been proposed by Ström et al. [30].

Pereberin's [26] proposal compresses individual blocks in a JPEG-like way.

There have been some efforts to optimize texture compression systems for normal maps used in bump mapping, which contain two components. The most widespread approach is ATI's 3Dc [3], where all compressed normals lie in an axis-aligned rectangle. Munkberg et al. [21] improved this by modifying the compression system to allow rotations of this rectangle and by introducing a new mode. It was further improved by Munkberg et al. [23].

Fenney [13] proposed a texture compression system, that stores two color codewords and modulation data for each 4×4 block of texels. During decompression the color codewords of 4 neighbouring blocks are mixed using bilinear interpolation. The results are then used similar to color codewords in S3TC.

2.3 HDR texture compression

Recently there has been research into compression of high dynamic range (HDR) textures. Munkberg et al. [22] proposed a complex high-quality texture compression system for HDR textures. In the same year Roimela et al. [27] proposed a simpler system, easier to implement in hardware but yielding lower image quality. Roimela et al. [28] tried to combine the advantages of both systems. The approach proposed by Wang et al. [33] stores HDR and LDR parts separately in S3TC(DXT5) textures. More recent approaches based on S3TC were proposed by Banterle et al. [4] and Sun et al. [32]. In DirectX 11 Microsoft will introduce new texture compression systems, including a HDR one [15].

3 Error measures

Error measures quantify the difference between images, and thus give the compression error as difference between the compressed and uncompressed texture.

The classical error measures are the mean absolute error (MAE) and the mean squared error (MSE) [14] along with the root mean squared error (RMSE). Let x and y be images with N pixels x_i and y_i.

$$\begin{aligned} \text{MAE}(x,y) &= \frac{1}{N}\sum_{i=0}^{N-1}|x_i - y_i|. \\ \text{MSE}(x,y) &= \frac{1}{N}\sum_{i=0}^{N-1}(x_i - y_i)^2. \\ \text{RMSE}(x,y) &= \sqrt{\text{MSE}(x,y)}. \end{aligned}$$

Structural similarity (SSIM) corresponds well to perceived visual quality [34]. Luminance l, contrast c and structure s are defined as functions of the mean values μ_x, μ_y, standard deviations σ_x, σ_y and correlation coefficient σ_{xy}:

$$l(x,y)=\frac{1\mu_x\mu_y+C_1}{\mu_x^2+\mu_y^2+C_1}, c(x,y)=\frac{2\sigma_x\sigma_y+C_2}{\sigma_x^2+\sigma_y^2+C_2}, s(x,y)=\frac{\sigma_{xy}+C_3}{\sigma_x\sigma_y+C_3},$$

$$\text{SSIM}(x,y) = l(x,y)^\alpha c(x,y)^\beta s(x,y)^\gamma.$$

The exponents $\alpha, \beta, \gamma > 0$ are used to adjust the impact of each measurement on SSIM, the C_i improve numerical stability.

Image dissimilarity (DSSIM) keeps the advantages of SSIM, but is more similar to distance measures (nonegative, reflexive, symmetric) [20]:

$$D(x,y) = \frac{1}{\text{SSIM(x,y)}} - 1.$$

DSSIM, like SSIM is defined for single-channel (e. g. grayscale) images only. The minimum over all channels has been used when calculating SSIM for multichannel images (e. g. [24]); here the maximum of the individual channel's DSSIM values is used as DSSIM for multichannel images. While [20] sets the C_i to 0 this causes numerical instability, so the values from the SSIM reference implementation have been used in [19].

4 ftc

Most existing texture compression systems use a small number of color codewords for each block, generate more colors from these and then choose one of the generated colors on a per-texel basis. Often the image quality is limited by the process that generates more colors (and their limited number) when the color codewords are far away from each other in color space, while it is limited by the precision of the color codewords, when these are near to each other in color space. The approaches to texture compression in this section increase precision in the color codewords when they are near to each other in color space by sacrificing some precision when they are far from each other.

Floating precision texture compression refers to texture compression systems that use variable precision in color codewords, depending on how near these color codewords are to each other in color space: Instead of storing the color codewords independently with fixed precision only one is stored directly, and the other is stored as a difference from the first one. When the color codewords are near to each other more bits are used for the first one, while the difference is stored with less bits.

Section 4.1 presents ftc1, the ftc texture compression system for RGB images. The design goal of ftc was to preserve the strengths of DXT1 while improving image quality for textures where DXT1 does not perform so well. Section 4.2 proves that ftc will never perform much worse than DXT1 for a given texture. For real-world images and textures the situation is even better: The textures where ftc performs worse than DXT1 or ETC are few and the difference is never noticeable, while there are textures where ftc performs a lot better than DXT1 and ETC. These experimental results can be found in section 4.4. Section 4.3 presents a ftc1 compression algorithm.

Section 4.5 applies ftc to RGBA images, with the first two approaches being similar to the way S3TC compressed RGBA textures. The third presents a more novel approach, which uses ftc's strengths to achieve a 8:1 compression ratio (common texture compression systems for RGBA images achieve 4:1) while still providing competitive image quality. Corresponding experimental results can be found in section 4.6.

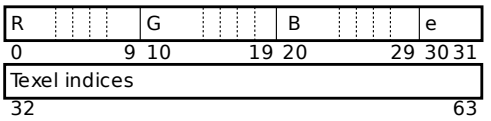

Fig. 1: ftc1 data layout

4.1 Compression of RGB images

The ftc texture compression system for RGB images, ftc1, has been designed to replace DXT1. It offers better image quality than DXT1 at the same compression ratio. In hardware implementations a large part of the decompression hardware can be shared with DXT1 decompression hardware. Compression algorithms can be implemented to provide sufficient speed for texture compression at runtime.

As in DXT1 two color codewords are stored, four colors are obtained by interpolation between these colors, for each texel one of the four colors is chosen. The main difference is in the way the color codewords are stored, with ftc1 offering higher precision when the color codewords are near to each other in color space.

The ftc1 data layout can be seen in Figure 1: There is a 2-bit exponent e and for each of the three colors 10 bits are stored. The first $(5+e)$ bits are the component of the primary color p. The other $(5-e)$ bits are a difference value d that gives the component of the secondary color relative to the primary color.

To obtain the secondary color the difference value d is sign-extended to $(5+e)$ bits and added to p (the wraparound in two's complement arithmetic avoids losing a bit of precision to the sign). Thus depending on the distance of the color codewords the effective precision can be anywhere from 5 to 8 bits. Both values are then expanded to the number of bits used per color channel by the implementation (typically 8).

Let (r_p, g_p, b_p) be the first color codeword obtained, (r_s, g_s, b_s) the second color codeword obtained. Since their role is symmetric one bit of information can be encoded by their order. This is used to choose the interpolation mode. Iff $r_s < r_p \vee (r_s = r_p \wedge g_s < g_p) \vee (r_s = r_p \wedge g_s = g_p \wedge b_s \leq b_p)$, then the alternative interpolation mode is used. Algorithm 1 shows the pseudocode of the method.

As can be seen in Figure 1 for each color channel primary color and difference value are stored next to each other. This results in fixed boundaries between the bits allocated to each color channel, easing hardware implementation and parallelization.

Input : A compressed block, 64 bit long, consisting of base color codeword (r_b, g_b, b_b), color distance (r_d, g_d, b_d), 2-bit exponent e, 2-bit texel indices $i_{0\ldots15}$
Output: A 4×4 block of RGB texels $t_{0\ldots15}$
$(r_p, r_s) := \text{DecodeColor}(r_b, r_d, e)$;
$(g_p, g_s) := \text{DecodeColor}(g_b, g_d, e)$;
$(b_p, b_s) := \text{DecodeColor}(b_b, b_d, e)$;
$c_0 := (r_p, g_p, b_p)$;
$c_1 := (r_s, g_s, b_s)$;
`// Color interpolation:`
if $r_s < r_p \vee (r_s = r_p \wedge g_s < g_p) \vee (r_s = r_p \wedge g_s = g_p \wedge b_s \leq b_p)$ **then**
  $c_2 := (c_0 + c_1)/2$;
  $c_3 := (0, 0, 0)$;
else
  $c_2 := (2c_0 + c_1)/3$;
  $c_3 := (c_0 + 2c_1)/3$;
for $j = 0 \ldots 15$ **do**
  $t_j := c_{i_j}$;

Algorithm 1: ftc1 decompression algorithm

Input : A $(5+e)$-bit base color b, a $(5-e)$-bit distance d and the 2-bit exponent e
Output: Two 8-bit colors p and s
`// Expand` b `to 8 bits to get` p`:`
$p := (b << (3-e))|(b >> (2+e*2))$;
`// Calculate` t `by adding` d `to` b`:`
$t := (b + \text{sign\_extend}(d))\&(0xff >> (3-e))$;
`// Expand` t `to 8 bits to get` s`:`
$s := (t << (3-e))|(t >> (2+e*2))$;

Algorithm 2: DecodeColor

In hardware implementations a stage that does sign extension and addition is needed. The other parts (interpolation, selection of texel color) can be shared with a DXT1 implementation.

Figure 2 illustrates that this approach achieves the design goal of improving compression quality for textures with small local chrominance variations: In contrast to other compression systems ftc1 causes no visible compression artifacts in the image gradient (while the example image features an image gradient created to show the differences in compression quality, image gradients are common e. g. in watercolor paintings or sunsets).

Fig. 2: Image gradient: Original, DXT1 (color bands, RMSE $\approx$ 1.55), ftc1 (nearly no visible artifacts, RMSE $\approx$ 0.94), ETC (color bands with jagged edges, RMSE $\approx$ 3.63)

4.2 Error bounds

Since S3TC is the most used texture compression system it is important for new texture compression systems to achieve high quality at textures where S3TC does.

The following result shows that ftc1's worst-case error is bounded by S3TC's error:

For a given RGB texture image with 8 bits per channel let D_{MAE} be the MAE of the ftc1-compressed texture minus the MAE of the DXT1-compressed texture. Let D_{RMSE} be the analogue for RMSE.

$$D_{\text{MAE}}, D_{\text{RMSE}} \leq \frac{255}{2(2^5-1)} \approx 4.11.$$

Proof: Since MAE and RMSE are metrics, by the triangle inequality D_{MAE} and D_{RMSE} are bounded by the maximum difference (in the respective metric) of a DXT1-compressed texture to the nearest ftc1-compressed texture. Let $\delta_0, \ldots, \delta_3$ be the differences in the 4 colors used in a block. The middle colors are obtained by interpolation from the color codewords, thus $\delta_1, \delta_2 \leq \max\{\delta_0, \delta_3\}$ holds. Since ftc1 is able to exactly represent the red and blue parts of color codewords used in DXT1 it is sufficient to look at the green channel to calculate these differences. There is only one case where ftc1 cannot represent DXT1's green values exactly: When DXT1 used a precision of 6 bits, while ftc1 uses 5 bits. Thus the maximum δ_0, δ_3 is half of the distance between ftc1's green values:

$$\delta_0, \delta_3 \leq \frac{255(2^5-1)^{-1}}{2}$$
$$\Rightarrow D_{\text{MAE}} \leq \frac{1}{N}\sum_{i=0}^{N-1} \max_j\{\delta_j\} \leq \max_j\{\delta_j\} \leq \frac{255}{2(2^5-1)},$$
$$D_{\text{RMSE}} \leq \sqrt{\frac{1}{N}\sum_{i=0}^{N-1} \max_j\{\delta_j^2\}} \leq \sqrt{\max_j\{\delta_j^2\}} \leq \frac{255}{2(2^5-1)}.$$

4.3 Compression algorithm

While the basic structure of this ftc1 compression Algorithm 3 is derived from the squish clusterfit algorithm [7] for DXT1 there are some important differences.

Input : A 4×4 block of uncompressed texels
Output: FTC1-compressed block
Determine the principal component;
Order the texel colors by their projection onto the principal component;
for *all clusterings of texel colors into three or four clusters preserving order* **do**
  Compute optimal endpoints;
  EncodeEndpoints() at standard precision;
  Keep solution if it results in the minimal error encountered so far;
  if *Previous EncodeEndpoints() suggested to try at reduced precision* **then**
    EncodeEndpoints() at reduced precision;
    Keep solution if it results in the minimal error encountered so far;
Use the kept solution to encode the block;

Algorithm 3: ftc1 compression algorithm

As in squish the colors are first ordered by their projection on the principal component, which can be found by a principal component analysis [25]. Then for all clusterings of colors into three or four clusters that preserve the ordering optimal line endpoints for placing clusters at the endpoints and along the line at the middle or one and two thirds are calculated and encoded. The clustering that results in the smallest (R)MSE along with the corresponding endpoints is chosen.

Some R8G8B8 values that cannot be exactly represented using the maximum number of possible bits can be represented better with less bits. Therefore if the exponent was > 0, encoding with a smaller exponent will be tried, too.

Since the compression algorithm operates on individual, independent blocks of 4×4 texels it can be easily parallelized (e. g. for execution on the graphics processing unit).

4.4 RGB image experiments

To compare ftc1 to existing texture compression systems images falling into three categories have been chosen and compressed with multiple texture compression systems, measuring the resulting image quality.

- Images typically used for this purpose in image compression, e. g. lena and lorikeet.
- Images from the strategy game *glest*, repre-

Input : Two RGB colors and a choice between standard and reduced precision
Output: Encoded colors and a suggestion whether to try again at reduced precision
for $e = 3 \ldots 0$ **do**
  TryEncode(e);
  if *TryEncode() succeeded* **then**
    if *precision = reduced* $\vee$ $e = 0$ **then**
      if $e > 0$ **then**
        Suggest to try at reduced precision, too.
      Return encoding given by TryEncode() above;
    else
      precision := standard;

Algorithm 4: EncodeEndpoints()

	MAE			RMSE			DSSIM		
Image	ftc1	DXT1	ETC	ftc1	DXT1	ETC	ftc1	DXT1	ETC
lena	8.22	8.63	9.87	6.93	7.09	8.04	0.074	0.083	0.121
lorikeet	10.06	10.36	11.98	8.44	8.54	9.90	0.131	0.143	0.162
glest avg	9.41	9.75	10.28	7.27	7.46	8.00	0.061	0.066	0.074
scourge avg	8.19	8.48	8.90	6.78	6.90	7.49	0.043	0.049	0.101
total avg	8.33	8.64	9.09	6.85	6.98	7.58	0.045	0.052	0.098

Tab. 1: Some experimental results for RGB images

senting textures that are used in applications that show a relatively large scene.

- Images from the role-playing game *scourge*, representing textures that are used in a closeup perspective.

Resulting in a total of 333 images.

ftc1 has been compared to two other texture compression systems, DXT1 and ETC. DXT1 has been chosen since it is still the most used compression system for RGB images, ETC as an example of a newer compression system.

For DXT1 squish has been used for compression, for ftc the algorithm from section 4.3 above. For ETC the program etcpack, available from Ericsson [12] has been used. For all three texture compression systems parameters have been set to minimize the (R)MSE.

Comparing ftc1 to DXT1 using RMSE or MAE for most images ftc1 is better than DXT1; there are few exceptions (82 out of 333 for MAE) and for these exceptions the difference between DXT1 and ftc1 is small (max. 0.75 MAE, typ. 0.1 MAE diff.). About as common as these exceptions are cases where ftc1 performs a lot better (up to 1.68 MAE, typ. 1.0 MAE diff.) than DXT1. The differences between ftc1 and ETC comparing RMSE and MAE are bigger, but the general tendency is the same.

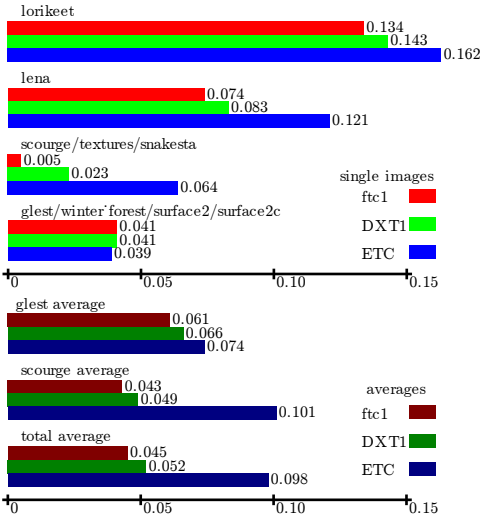

Fig. 3: DSSIM error for RGB images

Using DSSIM ftc1 yields better quality than DXT1 and ETC for nearly all images. Comparing ftc1 to DXT1 we see that only very rarely is ftc1's error bigger than DXT1's (9 out of 333 images). For seven of these images the difference in DSSIM is only 0.001, for the other two it's 0.002. On the other hand for images where ftc1 is better than DXT1 the difference goes as high as 0.036. Comparing ftc1 to ETC gives similar results: For 74 out of 333 images ETC yields better quality than ftc1. In the most extreme case the difference is 0.040. On the other hand for the images where ftc1 gives better quality than ETC the difference goes as high as 0.965.

Table 1 shows some experimental results. In the first two lines it can be seen that for the lena and lorikeet images ftc1 performs better than DXT1 and ETC. The next two lines show average values for the images from *glest* and *scourge*, while the last line gives an average over all 333 images. Figure 3 visualizes DSSIM results.

4.5 Compression of RGBA images

For transparency effects many images have an alpha channel in addition to their RGB channels. Three ftc variants for such images are discussed here.

A common use of the alpha channel is billboarding, where a complex object (e. g. trees, bushes) is

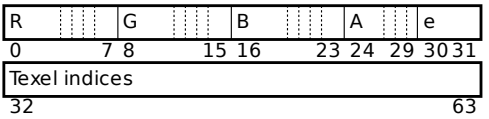

Fig. 4: ftc2 data layout

modelled using a small number of textured polygons. Each texel is either part of the object and opaque or not part of the object and fully transparent. Thus a single bit of precision in the alpha channel is sufficient. S3TC offers support for this in the DXT1 format through the transparent black texels in the alternative color interpolation mode.

ftc1 can support this in a similar way to DXT1: In the alternative color interpolation mode make the color c_3 (in Algorithm 1) transparent. In transparent texels a color used by nearby opaque texels is used: $c_3 = (c_0 + c_1)/2$. For premultiplied alpha a format that uses black for transparent texels instead could be designed.

For textures with a real alpha channel this approach is not suitable. One approach is to create a RGBA format from ftc1 in a similar way to how DXT5 has been created from DXT1: The ftc5 format consists of two 64-bit values per 4×4 texel block, the first value containing the RGB channels and to be decompressed like ftc1, the second value contains the alpha channel and is to be decompressed like the alpha channel in DXT5. The advantages of ftc1, Superior image quality and the reuse of S3TC texture decompression hardware hold.

In ftc2 the precision of color codewords is reduced to $(4 + e)$ bits per RGB channel (ftc1: $(5 + e)$). The six bits no longer used for RGB channels are used for the alpha channel resulting in a precision of $(3 + e)$ bits in the alpha channel. The resulting data layout can be seen in Figure 4, the decompression in Algorithm 5. For premultiplied alpha a format that sets the RGB channels of c_3 to black could be designed.

4.6 RGBA image experiments

For the RGBA variant of ftc1 and for ftc5 the ftc1 results from RGB images still hold. ftc2 has been compared to DXT5 and DXT3.

For the comparison 38 RGBA textures from *glest* and 11 diffuse maps from the *Doom 3* demo have been used. The experimental results can be seen in table 2: both DXT5 and DXT3 yield better image quality than ftc2. However the difference is not

Input : A compressed block, 64 bit long, consisting of base color codeword (r_b, g_b, b_b, a_b), color distance (r_d, g_d, b_d, a_d), 2-bit exponent e, 2-bit texel indices $i_{0\ldots15}$
Output: A 4×4 block of RGBA texels $t_{0\ldots15}$
$(r_p, r_s) := \text{DecodeColor2}(r_b, r_d, e)$;
$(g_p, g_s) := \text{DecodeColor2}(g_b, g_d, e)$;
$(b_p, b_s) := \text{DecodeColor2}(b_b, b_d, e)$;
$(a_p, a_s) := \text{DecodeAlpha}(a_b, a_d, e)$;
$c_0 := (r_p, g_p, b_p, a_p)$;
$c_1 := (r_s, g_s, b_s, a_s)$;
`// Color interpolation:`
if $r_s < r_p \vee r_s = r_p \wedge (g_s < g_p \vee g_s = g_p \wedge (b_s < b_p \vee b_s = b_p \wedge a_s \leq a_p))$ **then**
  `// Alternative color interpolation mode`
  $c_2 := (c_0 + c_1)/2$;
  $c_3 := (c_0 + c_1)/2$;
  $c_{3,3} := 0.0$;
else
  $c_2 := (2c_0 + c_1)/3$;
  $c_3 := (c_0 + 2c_1)/3$;
for $j = 0 \ldots 15$ **do**
  $t_j := c_{i_j}$;

Algorithm 5: ftc2 decompression algorithm

large and when measured by DSSIM, the error measure better matched to human perception, it is rather small. Considering that ftc2 has twice the compression ratio of DXT5/DXT3 these results are excellent, since for a given image size ftc2 will yield superior quality due to the higher resolution possible (Figure 6).

4.7 Limitations

ftc approximates all colors in the original image by colors on a line in color space. Thus quality will be

Input : A $(4 + e)$-bit base color b, a $(4 - e)$-bit distance d and the 2-bit exponent e
Output: Two 8-bit colors p and s
`// Expand` b `to 8 bits to get` p`:`
$p := (b << (4 - e))|(b >> (0 + e * 2))$;
`// Calculate` t `by adding` d `to` b`:`
$t := (b + \text{sign\_extend}(d))\&(0xff >> (4 - e))$;
`// Expand` t `to 8 bits to get` s`:`
$s := (t << (4 - e))|(t >> (0 + e * 2))$;

Algorithm 6: DecodeColor2

Input : A $(3+e)$-bit base alpha b, a $(3-e)$-bit distance d and the 2-bit exponent e

Output: Two 8-bit colors p and s

// Expand b to 8 bits to get p:
$p := (b << (5-e))|((b << 2) >> (e*2))|(b >> (1+e*3))$;
// Calculate t by adding d to b:
$t := (b + \text{sign\_extend}(d))\&(0xff >> (5-e))$;
// Expand t to 8 bits to get s:
$s := (t << (5-e))|((t << 2) >> (2+e*2))|(t >> (1+e*3))$;

Algorithm 7: DecodeAlpha

	MAE			RMSE			DSSIM		
Image	ftc1	DXT1	ETC	ftc1	DXT1	ETC	ftc1	DXT1	ETC
worker	9.86	8.57	8.70	9.89	8.35	8.38	0.037	0.032	0.032
sflgratetrans2	4.76	4.67	4.67	4.11	4.03	4.03	0.039	0.042	0.042
glest avg	12.41	10.14	10.14	11.33	9.05	9.19	0.060	0.043	0.044
doom3 avg	7.53	4.94	4.94	8.66	4.94	4.89	0.038	0.024	0.024
total avg	11.31	8.98	8.98	10.73	8.13	8.25	0.055	0.039	0.039

Tab. 2: Experimental results for RGBA images

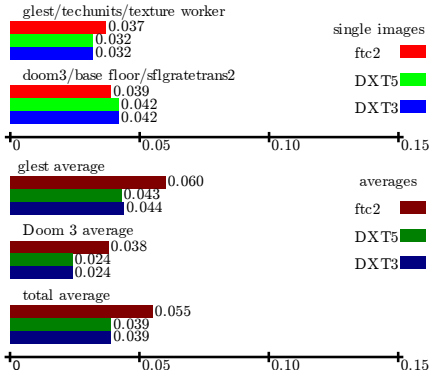

Fig. 5: DSSIM error for RGBA images

Fig. 6: Part of a RGBA texture from a computer game, rescaled to identical image size of 4 KB, transparent parts shown in grey: Uncompressed, DXT3, DXT5, ftc2.

Fig. 7: Problematic block: Original, DXT1 (RMSE $\approx$ 104.16), ftc1 (RMSE $\approx$ 104.21), ETC (RMSE $\approx$ 146.05)

low for 4×4-texel blocks with colors that cannot be well approximated by such a line. The most obvious example of such a situation is three different colors in a block, illustrated by Figure 7. However such blocks are highly problematic for other texture compression systems, too.

DXT3 and DXT5 can handle images with an alpha channel uncorrelated to the RGB channels better than ftc2 (at the cost of a lower compression ratio). Fortunately the alpha channel is often correlated to the RGB channels in real-world textures, e. g. transparent parts fading to black in images using premultiplied alpha.

5 Conclusion and Future Work

Texture compression schemes for both RGB and RGBA images have been presented. Compared to S3TC, the most common texture compression system today, they offer better quality at the same compression ratio or a higher compression ratio at comparable quality. Unlike other proposals the increase in decoding hardware complexity for implementing ftc is negligible.

The idea of floating precision texture compression can be applied to any texture compression system that uses color codewords. This work focused on creating systems similar to S3TC. It is likely that PVRTC [13], which needs more memory accesses than S3TC, but can yield better quality at the same compression ratio could be improved further using ftc. The new block-based texture compression systems to be introduced by Microsoft in DirectX 11 have modes that sacrifice precision in the color codewords to allow multiple lines in color space [15]. ftc could improve quality by lessening the impact of precision reduction. Other possibilities include creating a normal map compression system based on ftc or applying it to the various S3TC-based HDR texture compression systems.

Acknowledgements

Parts of this work were supported by the DFG-funded Graduierenkolleg Eingebettete Mikrosysteme. Thanks to PD Dr. Ilia Polian for discussions.

References

[1] Timo Aila, Ville Miettinen, and Petri Nordlund. Delay streams for graphics hardware. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers*, pages 792–800, 2003.

[2] Tomas Akenine-Möller and Jacob Ström. Graphics for the masses: a hardware rasterization architecture for mobile phones. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers*, pages 801–808, 2003.

[3] ATI Technologies. ATI Radeon X800 3Dc White Paper, 2004. `http://www.ati.com/products/radeonx800/3DcWhitePaper.pdf`.

[4] Francesco Banterle, Kurt Debattista, Patrick Ledda, and Alan Chalmers. A GPU-friendly method for high dynamic range texture compression using inverse tone mapping. In *Graphics Interface*, ACM International Conference Proceeding Series, pages 41–48, 2008.

[5] Andrew C. Beers, Maneesh Agrawala, and Navin Chaddha. Rendering from compressed textures. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 373–378, 1996.

[6] Pat Brown. EXT_texture_compression_s3tc, 2007. `http://opengl.org/registry/specs/EXT/texture_compression_s3tc.txt`.

[7] Simon Brown. DXT Compression Techniques. 2006. `http://www.sjbrown.co.uk/?article=dxt`.

[8] Graham Campbell, Thomas A. DeFanti, Jeff Frederiksen, Stephen A. Joyce, and Lawrence A. Leske. Two bit/pixel full color encoding. In *SIGGRAPH '86: Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, pages 215–223, 1986.

[9] Paul Debevec. Rendering synthetic objects into real scenes: bridging traditional and image-based graphics with global illumination and high dynamic range photography. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 189–198, 1998.

[10] Paul E. Debevec and Jitendra Malik. Recovering high dynamic range radiance maps from photographs. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 369–378, 1997.

[11] E. Delp and Robert Mitchell. Image compression using block truncation coding. *IEEE Transactions on Communications*, 27(9):1335–1342, 1979.

[12] Ericsson. Ericsson Texture Compression, 2007. `http://www.ericsson.com/mobilityworld/sub/open/technologies/texture_compression/tools/etcpack`.

[13] Simon Fenney. Texture compression using low-frequency signal modulation. In *HWWS '03: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 84–91, 2003.

[14] Carl Friedrich Gauss. Theoria combinationis observationum erroribus minimis obnoxiae. *Commentationes Societatis Regiae Scientiarum Gottingensis recentiores*, 5:33–90, 1823.

[15] Kevin Gee. Introduction to the Direct3D 11 Graphics Pipeline, 2008. `www.nvidia.com/content/nvision2008/tech_presentations/Game_Developer_Track/NVISION08-Direct3D_11_Overview.pdf`.

[16] Paul Heckbert. Color image quantization for frame buffer display. *SIGGRAPH Comput. Graph.*, 16(3):297–307, 1982.

[17] T. Inada and M. D. McCool. Compressed lossless texture representation and caching. In *GH '06: Proceedings of the 21st ACM SIGGRAPH/Eurographics symposium on Graphics hardware*, pages 111–120, 2006.

[18] G. Knittel, A. Schilling, A. Kugler, and W. Straßer. Hardware for Superior Texture Performance. In *Computers & Graphics 20*, pages 475–481, 1996.

[19] Philipp Klaus Krause. C++ implementation of DSSIM. `http://colecovision.eu/graphics/DSSIM/`.

[20] A. Loza, L. Mihaylova, N. Canagarajah, and D. Bull. Structural similarity-based object tracking in video sequences. In *9th International Conference on Information Fusion*, pages 1–6, 2006.

[21] Jacob Munkberg, Tomas Akenine-Möller, and Jacob Ström. High quality normal map compression. In *GH '06: Proceedings of the 21st ACM SIGGRAPH/Eurographics symposium on Graphics hardware*, pages 95–102, 2006.

[22] Jacob Munkberg, Petrik Clarberg, Jon Hasselgren, and Tomas Akenine-Möller. High dynamic range texture compression for graphics hardware. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers*, pages 698–706, 2006.

[23] Jacob Munkberg, Ola Olsson, Jacob Ström, and Tomas Akenine-Möller. Tight frame normal map compression. In *GH '07: Proceedings of the 22nd ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware*, pages 37–40, 2007.

[24] Damian Nowroth, Ilia Polian, and Bernd Becker. A study of cognitive resilience in a JPEG compressor. In *IEEE International Conference on Dependable Systems and Networks With FTCS and DCC*, pages 32–41, 2008.

[25] Karl Pearson. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh and Dublin Philosophical Magazine and Journal*, 6:559–572, 1901.

[26] Anton V. Pereberin. Hierarchical approach for texture compression. In *Proceedings of GraphiCon '99*, pages 195–199, 1999.

[27] Kimmo Roimela, Tomi Aarnio, and Joonas Itäranta. High dynamic range texture compression. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers*, pages 707–712, 2006.

[28] Kimmo Roimela, Tomi Aarnio, and Joonas Itäranta. Efficient high dynamic range texture compression. In *SI3D '08: Proceedings of the 2008 symposium on Interactive 3D graphics and games*, pages 207–214, 2008.

[29] Jacob Ström and Tomas Akenine-Möller. iPACKMAN: high-quality, low-complexity texture compression for mobile phones. In *HWWS '05: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 63–70, 2005.

[30] Jacob Ström and Martin Pettersson. ETC2: texture compression using invalid combinations. In *GH '07: Proceedings of the 22nd ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware*, pages 49–54, 2007.

[31] Jacob Ström and Tomas Akenine-Möller. Packman: Texture compression for mobile phones, 2004. `http://www.cs.lth.se/home/Tomas_Akenine_Moller/pubs/packman_sketch.pdf`.

[32] Wen Sun, Yan Lu, Feng Wu, and Shipeng Li. DHTC: an effective DXTC-based HDR texture compression scheme. In *GH '08: Proceedings of the 23rd ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware*, pages 85–94, 2008.

[33] Lvdi Wang, Xi Wang, Peter-Pike Sloan, Li-Yi Wei, Xin Tong, and Baining Guo. Rendering from compressed high dynamic range textures on programmable graphics hardware. In *I3D '07: Proceedings of the 2007 symposium on Interactive 3D graphics and games*, pages 17–24, 2007.

[34] Zhou Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *Image Processing, IEEE Transactions on*, 13(4):600–612, 2004.

[35] Li-Yi Wei. Tile-based texture mapping on graphics hardware. In *HWWS '04: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 55–63, 2004.