A Comparison of Color Spaces for Texture Compression

Chirantan Ekbote
ekbote@seas.harvard.edu

Michael S. Kester
kester@eecs.harvard.edu

December 11, 2012

1 Background and Problem Statement

The use of texture mapping in modern 3D computer graphics has exploded since their introduction by Catmull [3]. Texture maps provide information like colors, normals, and alpha values for the objects in a 3D scene. Using texture maps reduces computation time during rendering by storing pre-computed information in a readily accessible format freeing up valuable processor cycles for other more intensive calculations. Even the most simple games today often contain hundreds of textures and expect hardware support for fetching the texture values during rendering.

However, the large quantity of texture maps used in today's applications and the limited amount of memory available in graphics hardware places a severe constraint on the size and number of texture maps that can actually be stored in memory. Additionally, limited memory bandwidth can slow down rendering while the renderer waits for texture map data to be fetched.

To solve this problem, several compression methods have been proposed to reduce the memory footprint of each texture map and to reduce the memory accesses necessary to fetch any portion of the texture map. These compression schemes all require three key features:

1. Support fast decoding of the compressed data. Ideally, decompression can be implemented in graphics hardware.

2. Allow for random access to any part of the compressed data. Since we can not know how the renderer will access the data, it must be stored in a predictable layout.

3. Provide a good ratio between compression rate and visual quality. These compression schemes need to be lossy to achieve the desired rates and will introduce errors and artifacts in the decompressed textures. However, these methods must still provide a proper trade-off because in a real-time application, it is the quality of the scene that is most important, not of an individual texture.

Modern texture compression techniques achieve these properties by compressing fixed-size blocks of pixels together into compressed chunks laid out in a predetermined pattern. By reducing the inter-dependency between the chunks, they reduce the memory accesses necessary to reconstruct any part of the texture. However, most modern techniques still use the standard RGB (red-green-blue) channel format when storing color textures.

We propose to enhance these techniques by transforming the RGB channels into the YUV color space and using key properties of the YUV color space to improve the compression rate to visual quality ratio in the final reconstructed texture.

2 Related Work

Beers et al. [1] were the first to address key issues in texture compression algorithm design for real-time rendering, including fast decoding speed, random access, and a proper trade-off between compression rate and visual quality. They proposed a simple vector quantization (VQ) technique where each pixel is associated with an index into a lookup table that determines the final color of that pixel. The colors in the lookup table are generated using a Generalized Lloyd Algorithm (GLA) [7]. While VQ can provide up to a 35:1 compression ratio [15], it requires two memory accesses and is not a good option for applications where memory bandwidth is at a premium.

To overcome the deficiencies of VQ, we can instead used a block-based compression scheme to encode fixed-size blocks of pixels together. Block-based texture compression was initially proposed for grayscale images by Delp and Mitchell [5], and extended to color images by Campbell et al. [2]. The first block-based texture compression format to enjoy widespread hardware support was S3TC [8], also known as DXTC. S3TC/DXTC remains the de-facto standard on desktop graphics processors and Windows phones [14].

The S3TC/DXTC format stores two high-bit-depth color samples per block, and stores a low-bit-depth interpolation weight per pixel. S3TC/DXTC decompression obtains a rough estimate for the value of each recovered pixel by nearest-neighbor sampling the sparsely encoded high-bit-depth colors. To obtain the final color value for each recovered pixel, S3TC/DXTC decompression blends between the previously obtained high-bit-depth colors according to the pixel's corresponding interpolation weight.

The visual fidelity of S3TC/DXTC images depends heavily on the choice of high-bit-depth colors used to represent each block of pixels. Fortunately, optimal high-bit-depth colors can be determined analytically by applying local principle component analysis to each block [16]. However, using only two independent colors per block can introduce blocky artifacts and banding due to color discontinuities, especially in images with smooth gradients.

The PVRTC format [6] is conceptually similar to S3TC/DXTC, but explicitly addresses the problem of color discontinuities at block boundaries. As is the case in S3TC/DXTC decompression, PVRTC decompression recovers each

pixel by linearly interpolating between two high-bit-depth colors. However, in PVRTC decompression, the high-bit-depth colors used to recover each pixel come from bi-linearly blending among high-bit-depth color samples from the 4 nearest blocks. This is in contrast to S3TC/DXTC decompression, where the high-bit-depth colors used to recover each pixel come from nearest-neighbor sampling. This modification noticeably reduces color discontinuities at block boundaries but makes the problem of choosing optimal high-bit-depth color samples significantly more challenging. This is because the choice of each high-bit-depth color sample is (indirectly) coupled to every other high-bit-depth color sample and choosing the best high-bit-depth colors requires global optimization over the entire image.

The ETC1 texture format [12] is also conceptually similar to S3TC/DXTC, but divides each block into sub-blocks with one high-bit-depth color stored for each sub-block. ETC1 uses the low-bit-depth values to modify the luminance of the single color in each sub-block. The low-bit-depth values serve as indices into a codebook that is constant across all textures. ETC1 enjoys dedicated hardware support on some Android devices and has been extended in the ETC2 format [13]. ETC2 achieves higher quality by taking advantage of invalid bit sequences in ETC1 to specially handle cases where it performs poorly.

More recently, Wang et al. [17] proposed a method to render compressed High Dynamic Range (HDR) textures on programmable graphics hardware called DHTC. Their method first converts the RGB channels of an image into a custom LUVW color space, which they encode using S3TC/DXTC. Decoding the texture involves first decompressing the S3TC/DXTC texture and then using a shader program to convert the LUVW channels back to RGB color space. Sun et al. [15] extend this method by using the more standardized YUV color space and a dynamic modifier table to allow decompression even on low end non-programmable graphics hardware.

3 Experimental Setup

We test modified versions of PVRTC by using the techniques of [17], [15], [9], and [10]. PVRTC relies on linear interpolation in the RGB color space by choosing between 4 pre-determined interpolation weights to reconstruct a texture. Our experiments are designed to determine how well-suited the RGB color space is for linear interpolation and if we can achieve better results using a colorspace that takes advantage of the natural interdependence of the human perception of colors.

The YUV colorspace, Fig. 2, uses the Y channel for the luminance (brightness) and the UV channels for the chromaticity (colors) of each pixel. The Y, U and V channels are coordinates into a continuous color cube that more accurately represents the interdependence of the color channels in the human visual system [18]. We hypothesize that we can take advantage of key properties of the YUV colorspace to improve PVRTC compression by either improving the visual quality while keeping the compression rate static or by improving the

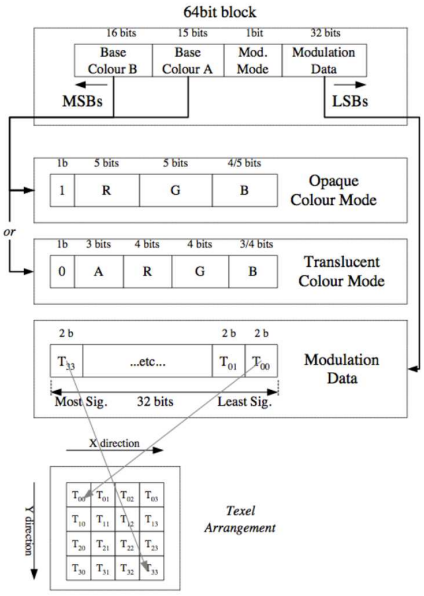


Figure 1: Layout of the compressed data in PVRTC. This figure is a reproduction of one that appears in [6].

compression rate while maintaining the same visual quality.

We test our hypothesis using three different block layouts for the compressed data. In the standard PVRTC layout we store two RGB colors using 5, 6, and 5 bits for each channel respectively. Additionally, we store 16 interpolation weights using 2 bits per pixel (Fig. 1).

3.1 Standard YUV Data Layout ("YUV")

The first data structure we test is a simple transformation from the RGB ("RGB") to YUV colorspace. We run the PVRTC algorithm on the transformed data unchanged and store the final compressed blocks using 6 bits for the L channel and 5 bits for the U and V channels each. Decoding involves first running the regular PVRTC decompression and then transforming the channels back into the RGB color space. Transformation between color spaces is accomplished by multiplying the color vector by a static matrix and can easily be implemented in hardware.

64 bits

L_a	U_a	V_a	L_b	U_b	V_b	Modulation Bits
6	5	5	6	5	5	32

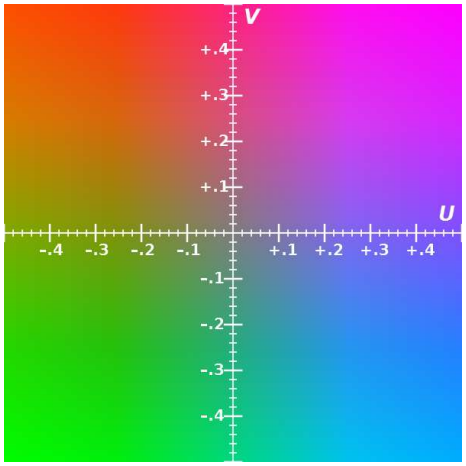


Figure 2: Example of a UV color plane with Y value = 0.5.

3.2 Max YUV Data Layout ("YUVm")

The human visual system is more sensitive to changes in brightness than to changes in color [4]. Taking advantage of this feature, we modify the compressed data structure seen in Fig. 1 to allocate more bits to the Y (brightness) channel by stealing them from the U and V channels, giving us the following layout for the compressed data:

64 bits

L_a	U_a	V_a	L_b	U_b	V_b	Modulation Bits
8	4	4	8	4	4	32

3.3 16-bit YUV Data Layout ("YUV16")

The approach in the previous section relies on the assumption that the encoded colors make a bigger difference to the visual quality of the decoded image than the interpolation weights. We determine if this is really the case by experimenting with increasing the number of weight choices from 4 to 8. By encoding only one U and V value and two Y values per block, we halve the number of bits required to encode the color information. We then use these extra bits to give us a smoother interpolation gradient.

64 bits

L_a	L_b	U	V	Modulation Bits
4	4	4	4	48

4 Trials and Results

We ran a series of tests to validate the effectiveness of our compression assumptions. The results of these tests are presented here and interpreted in section 5.

4.1 Setup

We ran each compression method on one hundred images and recorded the results. The images were collected from the web arbitrarily through a manual search. Many of the images come from the Natural Geographic website [11].

Each 512-by-512 image was chosen as a representation of some feature that might affect the impact of the compression in one way or another. Some were chosen for being monochromatic, others for subject matter—faces might produce artifacts displeasing to the eye while similar artifacts in a landscape could go unnoticed. We also chose a variety of file formats as input: JPEG (81%), BMP (17%), PNG (2%). Since we had no reason to believe that input file format would affect the calculations, the ratio of each format is simply a fallout of the selection method. While an attempt was made to vary the color, texture, and subject matter, some selection bias was likely introduced through the manual selection of images.

4.2 Results

The results on visual quality can be seen in the collections of images in figures 4 and 5. In both cases the error rates increase across the figures (b), (c), (d), and (e) producing a "greenening" but with minimal artifacts in even the worst performing method. We do however see significant artifacts introduced in some of the other images.

The graph in Figure 3 shows significant banding of the error rates. We can easily rank the compression schemes based on Root Mean Squared (RMS) error per pixel: RGB followed by YUV, YUVm and finally YUV16. This follows the ranking one would apply visually. Interestingly, the rankings are consistent for each image. That is to say, in terms of error, e, and visual quality, v, for an individual image it is true that[1]:

$$e_{rgb} < e_{yuv} < e_{yuvm} < e_{yuv16} \implies \tag{1}$$

$$v_{rgb} < v_{yuv} < v_{yuvm} < v_{yuv16} \tag{2}$$

It is not always true across images however:

$$e_{yuv16_a} < e_{yuv16_b} \not\implies v_{yuv16_a} < v_{yuv16_b} \tag{3}$$

In fact, the error rate does not seem to be a good indicator of artifacts specifically, though a high error rate does seem to imply a inferior image (images with high error and few artifacts display more color distortion).

[1] In the case of visual inspection the '<' means "looks better than"

	RGB	YUV	YUVm	YUV16
Best	5.1546	8.1071	14.574	27.698
Worst	21.007	25.662	32.363	50.49
Mean	9.4663	17.494	24.816	39.466

Table 1: Error Statistics Summary

The ranges of error rate varied significantly with the best RGB image suffering an error rate of only 5.15 while the worst performing RGB image had a rate of 21.01. Similar results were observed in the other schemes. A summary is provided in Table 1.

The error introduced produces a "greenening" of the image with minimal artifacts introduced until the YUV16 which frequently displays the most significant artifacts.

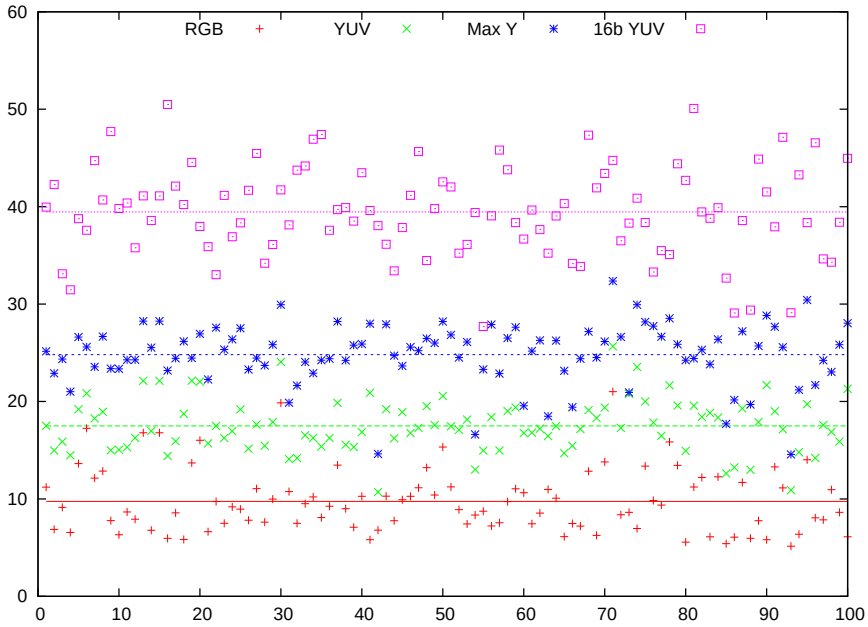


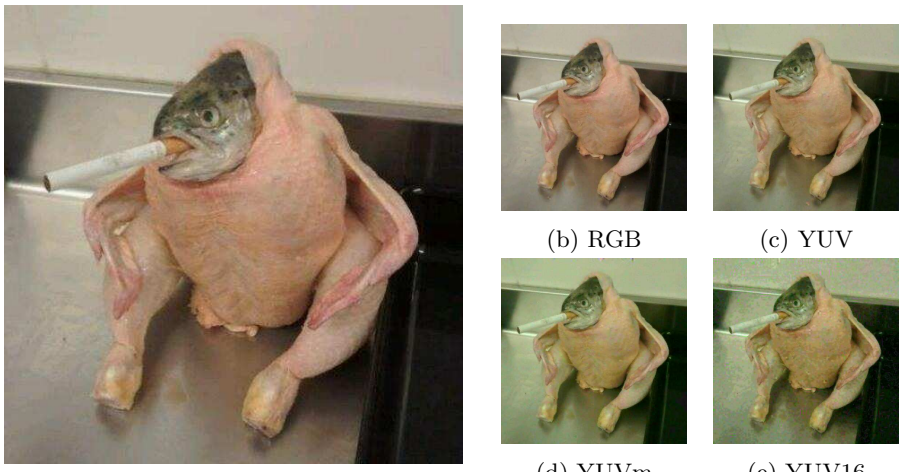
Figure 3: Comparison of RMS error for the 100 experimental images. The line is the mean for that compression scheme.

5 Conclusion

Our experiments showed that the RGB space is much better suited for PVRTC than the YUV colorspace. We originally hypothesized that using a color space

(a) Original Image (b) RGB (c) YUV (d) YUVm (e) YUV16

Figure 4: Comparison of visual artifacts and distortion (Lohan).



(a) Original Image (b) RGB (c) YUV (d) YUVm (e) YUV16

Figure 5: Comparison of visual artifacts and distortion (Chicken-Fish).

that matched the natural inter-dependence of the human visual system would produce more visually pleasing results. However, the independence of the RGB channels is actually an advantage for compression rather than a disadvantage.

In the RGB space, modifying a single channel only affects one of the final (RGB) colors of the pixel and any error is limited to that channel. In the YUV space however, changing the value of any of the channels modifies all the final (RGB) colors for that pixel. This interdependence makes it harder to find pairs of values that accurately bound all the pixels in a block and increases the bound on the minimum error that can be achieved.

Recall that the YUV channels are coordinates into a continuous color cube (Fig 2). Since all the channels for a pixel use the same interpolation weight however, we cannot fully take advantage of the cube spanned by the two representative YUV coordinates per block (which act as the min and max coordinates of the cube) and are instead limited to only those values that fall on the diagonal between the two coordinates. This severely restricts the choice of available colors. To make full use of the properties of the YUV color space, we would have to encode different interpolation weights for each channel per pixel, which would significantly increase the size of the compressed data and defeat the purpose of the compression.

It is no surprise then, that our proposed YUVm and YUV16 data structures performed much worse than the standard RGB data structure. For the YUVm data structure, the additional two (lowest order) bits of precision given to the Y channel does not counteract the loss of one (higher order) bit of precision in each of the U and V channels. Similarly, the loss of information that comes from storing only one value for the U and V channels per block is not offset by a smoother interpolation gradient in the YUV16 data structure.

We have shown that independently encoding color channel information is ideal for block-based texture compression schemes and that simply transforming the colors into a different color space is not enough to yield any significant improvement in compression ratio or quality. To make color space transformations useful for texture compression, we must use a specially designed color space [17] or use a dynamic modifier table to account for the errors introduced by the interdependence of the channels [15].

References

[1] Andrew C. Beers, Maneesh Agrawala, and Navin Chadda. Rendering from compressed textures. In *Proceedings of SIGGRAPH*, pages 373–378, 1996.

[2] Graham Campbell, Thomas A. DeFanti, Jeff Frederiksen, Stephen A. Joyce, Lawrence A. Leske, J. A. Lindberg, and D. J. Sandin. Two bit/pixel full color encoding. In *Proceedings of SIGGRAPH*, volume 22, pages 215–223, 1986.

[3] Edwin Earl Catmull. *A subdivision algorithm for computer display of curved surfaces.* PhD thesis, The University of Utah, 1974. AAI7504786.

[4] David Corney, John-Dylan Haynes, Geraint Rees, and R. Beau Lotto. The brightness of colour. *PLoS ONE*, 4(3):e5091, 03 2009.

[5] E. Delp and O. Mitchell. Image compression using block truncation coding. *IEEE Transactions on Communications*, 2(9):1335–1342, 1979.

[6] Simon Fenney. Texture compression using low-frequency signal modulation. In *Graphics Hardware*, pages 84–91. ACM Press, 2003.

[7] Allen Gersho and Robert M. Gray. *Vector quantization and signal compression.* Kluwer Academic Publishers, Norwell, MA, USA, 1991.

[8] K. Iourcha, K. Nayak, and Z. Hong. System and method for fixed-rate block-based image compression with inferred pixels values. In *US Patent 5,956,431*, 1999.

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

[10] Kimmo Roimela, Tomi Aarnio, and Joonas Itäranta. Efficient high dynamic range texture compression. In *Proceedings of the 2008 symposium on Interactive 3D graphics and games*, I3D '08, pages 207–214, New York, NY, USA, 2008. ACM.

[11] National Geographic Society. http://www.nationalgeographic.com/.

[12] J. Ström and T. Akenine-Möller. ipackman: High-quality, low-complexity texture compression for mobile phones. In *Graphics Hardware*, 2005.

[13] J. Ström and M. Pettersson. Etc2: Texture compression using invalid combinations. In *Graphics Hardware*, 2007.

[14] J. Ström and P. Wennersten. Lossless compression of already compressed textures. In *High Performance Graphics*, 2011.

[15] Wen Sun, Yan Lu, Feng Wu, Shipeng Li, and J. Tardif. High-dynamic-range texture compression for rendering systems of different capacities. *Visualization and Computer Graphics, IEEE Transactions on*, 16(1):57 – 69, jan.-feb. 2010.

[16] J. van Waveren. Real-time texture streaming and decompression. Technical report, Id Software, 2006.

[17] 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 *Proceedings of the 2007 symposium on Interactive 3D graphics and games*, I3D '07, pages 17–24, New York, NY, USA, 2007. ACM.

[18] Thomas Young. The bakerian lecture: On the theory of light and colours. *Philosophical Transactions of the Royal Society of London*, 92:12–48, 1802.