Practical Perceptually Adaptive Color Texture Map Compression for 3D Video Games

Alexander Wong
a28wong@engmail.uwaterloo.ca

William Bishop
wdbishop@uwaterloo.ca

Department of Electrical and Computer Engineering
University of Waterloo
Waterloo, Ontario, Canada, N2L 3G1
Phone Number: 519-888-4567 ext. 37159

ABSTRACT

3D video games use texture maps to improve the realism and the visual detail of graphical objects without significantly increasing rendering complexity. The general trend in the gaming industry towards the use of large texture maps has lead to the use of texture compression techniques. Current color texture compression techniques treat all texture content uniformly which can lead to reduced visual quality.

This paper addresses this issue by introducing a perceptually adaptive approach to color texture map compression that is based on the human vision system. The proposed method attempts to improve visual detail and compression performance without adding computational complexity. Experimental results show improved visual quality and compression performance over existing techniques. The proposed method is well suited for real-time applications such as 3D video games.

Keywords: texture maps, adaptive compression, 3D video games

1. INTRODUCTION

One of the essential graphics techniques used in modern 3D video games is texture mapping [1]. In texture mapping, an image, referred to as a texture, is mapped to the surface of a simpler graphical object to give the appearance of fine details. Figure 1 illustrates the use of texture mapping on a head model. The face texture used gives the simple head model the appearance of having eyes, nose, lips and hair, despite the fact that the underlying model does not contain such details. This technique is used to improve the realism of graphical objects without the increased computational complexity of rendering complex geometry. This makes the technique suitable for real-time 3D applications such as video games.

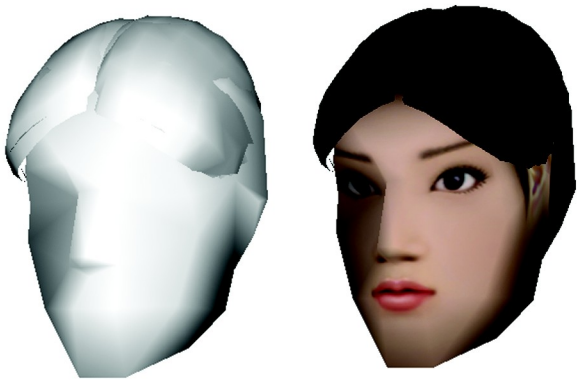

Figure 1: Left: Simple head model
Right: Head model with texture mapping

The major drawback to texture mapping is that textures need to be stored in memory and therefore the trend towards larger textures have increased the memory requirements of 3D video games. This problem is compounded by the fact that the amount of video memory available on a consumer 3D

video card is limited. Therefore, it is often the case that textures need to be fetched from slower sources such as secondary storage devices (e.g., hard drives and optical drives). To reduce the storage and data transfer requirements of 3D video games, texture map compression is often used. Texture maps are also used by more advanced techniques such as bump-mapping [2] and displacement mapping [3].

Many different techniques have been proposed for compressing images to reduce storage requirements. The most popular techniques are transform-based techniques such as JPEG [4] and JPEG2000 [5], which have high compression rates while maintaining a high level of visual quality. However, such image compression techniques are not well suited for handling texture maps in a real-time 3D video game environment. First, transform-based techniques require that entire blocks of pixels be decompressed. Furthermore, these algorithms are variable rate techniques that do not allow for the address calculation of a specific pixel within a texture map compressed using these algorithms. Therefore, these compression schemes do not allow for fast random access to individual pixels in a texture map. Finally, these techniques can be computationally expensive to perform, thus making real-time processing challenging in the context of a 3D video game.

More efficient techniques have been used to compress texture maps in 3D video games. DXTC (DirectX Texture Compression) is a family of texture compression techniques based on the Color Cell Compression (CCC) method [6]. The DXT-1 format can be used to compress a RGBA texture at a ratio of 6:1 and the DXT-5 format can be used to compress a RGBA texture at a ratio of 4:1. One of the major benefits of DXTC is the fact that it is part of the Direct3D standard and therefore hardware acceleration is typically available on modern consumer 3D video cards. DXTC has become the leading

technique for compressing color textures in current generation 3D video games. Recently, ATI Technologies introduced 3Dc, a compression algorithm based on BTC (Block Truncation Coding) [7] to address the shortcomings of DXTC when used for normal map compression. Other texture compression algorithms include texture compression using low-frequency signal modulation [8] and PACKMAN / *i*PACKMAN [9].

One of the major disadvantages of DXTC is the fact that it treats all texture content uniformly. Therefore, the same amount of data is used to represent regions lacking details as the amount of data used to represent highly detailed regions in the texture map. DXTC establishes a fixed tradeoff between visual quality and texture map compression rates, regardless of the amount of detail in the texture map. The adverse effect of this approach is that regions containing perceptually important details such as edges may appear to be significantly degraded. This is particularly a problem for textures that can be viewed up close by the user in a 3D video game environment, such as floor and wall textures. An example of this problem is illustrated in Figure 2, where a wall texture is compressed using DXTC. It can be easily seen that fine details are significantly distorted by the compression process. Insufficient data is used to represent the details of the wall texture. One approach to addressing this problem is to develop a texture compression algorithm that adapts based on the perceptual characteristics of the human vision system.

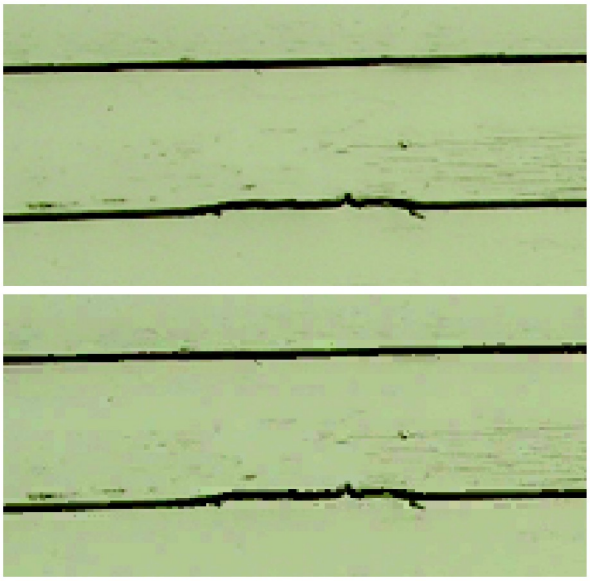

Figure 2: Texture compression using DXTC
Top: Original wall texture
Bottom: Wall texture compressed using DXTC
Notice the fine details in the texture, such as edges appear significantly degraded.

The main contribution of this paper is a practical adaptive approach to texture compression for use in 3D video games. This method utilizes perceptually important characteristics such as edge activity and texture activity to improve visual quality while achieving high compression rates. This allows for 3D video games to retain the visual detail as envisioned by the texture artist. In this paper, the underlying theory behind adaptive texture map compression is described in Section 2. A CCC-based implementation of the proposed algorithm is presented in Section 3. The testing methods and test data are outlined in Section 4. Finally, experimental results comparing an implementation of the proposed approach with the original DXTC algorithm are discussed in Section 5, and conclusions are drawn in Section 6.

2. THEORY

Before outlining an implementation of the proposed texture map compression algorithm, it is important to discuss the theory behind the key components of the algorithm. First, the underlying theory behind the CCC compression scheme is presented. Second, a number of perceptual characteristics that are important to the human vision system are presented along with practical empirical metrics used for quantifying these characteristics. Finally, the theory behind perceptually adaptive texture map compression is described in detail.

2.1. Color Cell Compression (CCC)

CCC is a lossy texture compression algorithm designed for the compression of color texture maps. In CCC, the image is divided into a set of smaller blocks of size 4×4. For each block, two 8-bit color values are selected to represent the base colors of the block. The selection is based on the local statistics of the block. Finally, a 16-bit value is stored to indicate which of the two base colors are used to represent the pixel, resulting in a total storage requirement of 2 bits per pixel (BPP). The most popular extension to CCC is DXTC, also referred to as S3TC (S3 Texture Compression).

DXTC is a family of lossy texture compression methods that is widely used in 3D video games and is typically supported in hardware by consumer 3D video cards. As an industry standard for texture

compression in 3D video games, it is important to understand the improvements DXTC makes to the basic CCC concept. There are a total of five versions of the basic DXTC compression methodology, aptly named DXT1 through DXT5. For the purpose of explaining the basic DXTC compression methodology, the DXT1 compression method is described. In DXT1, a RGB image is divided into a set of smaller blocks of size 4 x 4. For each block, two 16-bit R5G6B5 color values C_1 and C_2 are chosen based on the local color statistics to represent the pixel values in the block. Two intermediate color values (C_3 and C_4) are calculated as follows:

If $C_1 > C_2$:

$$C_3 = \frac{2}{3}C_1 + \frac{1}{3}C_2 \qquad (1)$$

$$C_4 = \frac{1}{3}C_1 + \frac{2}{3}C_2 \qquad (2)$$

If $C_1 \leq C_2$:

$$C_3 = \frac{1}{2}C_1 + \frac{1}{2}C_2 \qquad (3)$$

$$C_4 = \{R = 0, G = 0, B = 0\} \qquad (4)$$

The color value used to represent each pixel in the block is then determined by finding the Euclidean distance between the actual color value of a pixel and each of the four color values. This information is then stored as a 32-bit value, where 2 bits are used per pixel. This information serves as a

color index. For example, if the actual color value of a pixel is closest to C_1, then the pixel is given a value of 0. The color index and the two color values C_1 and C_2 are then stored for each block.

To illustrate the compression performance of DXTC, consider the compression of a RGB texture map using DXT1. Assume that each pixel in the original texture map is represented by a 24-bit RGB color value. DXT1 divides the image into blocks of size 4×4 and represents the color value of a pixel using 16-bit R5G6B5 values. Since two 16-bit color values and one 32-bit color index are stored for each block, a total of 64 bits is needed per block. As the original RGB texture map requires $16 \times 24 = 384$ bits of data per block, DXTC compression results to a compression ratio of 6:1.

The DXTC texture compression technique inherits a number of important benefits from CCC that are useful in 3D video games. First, it facilitates random access to individual pixels within a texture map without the need to decompress large segments of a texture map. This is an important factor for real-time 3D applications such as 3D video games, where pixels may be accessed from different blocks within a texture. Second, it is computationally efficient, particularly in modern 3D video cards that provide full hardware acceleration for this technique. However, DXTC also suffers from the fact that it is a fixed rate texture compression algorithm that treats all texture content equally. Since highly detailed texture content is stored using the same limited amount of data as simple content, important details in the texture appear noticeably degraded, particularly due to the sensitivity of the human vision system to such details. This problem can be illustrated using a simple example. Consider a texture map consisting of a highly detailed region surrounded by a constant red background. While the blocks in the constant red background regions can be represented by a single 16-bit color value, a total of 64 bits is spent on the

block. The blocks in the highly detailed region require fine color granularity to be represented properly but are limited to the 64 bits available per block, resulting in visible degradation. Therefore, a method to adaptively allocate data in a more effective manner is highly desired to address this problem.

2.2. Perceptually Important Metrics

One approach to allocating data resources for texture representation in a more effective manner is to adjust the data allocation scheme depending on the visual importance of texture content. Fixed texture compression algorithms such as CCC and its variants treat all texture content equally and do not take into account the perception of the human vision system. Characteristics that are perceptually important to the human vision system can be used to enhance the overall visual quality of a texture while achieving high compression rates and reasonable performance. In the proposed texture compression algorithm, the following perceptually important characteristics are chosen for consideration:

1. Edge activity
2. Texture activity
3. Brightness

These characteristics can be evaluated quantitatively using computationally efficient metrics. For the proposed algorithm, the metrics are evaluated in a block-based fashion.

2.2.1 Edge Activity

To evaluate the edge characteristics of the texture, the texture is processed using an edge detection algorithm such as a Sobel edge detector or a Canny edge detector to create a binary edge map (E), where edge pixels are represented by a value of 1 and non-edge pixels are represented by a value of 0. An edge rating (ER) for a texture block is determined using the following metric:

$$ER = \sum_{x} \sum_{y} (E_{\text{block}}(x, y)) \tag{5}$$

where $E_{\text{block}}(x,y)$ is the edge value of the pixel at coordinates (x,y) within the texture block.

The human vision system is heavily reliant on edges for visual recognition [10]. As such, the human vision system is very sensitive to edge degradation in visual content. Therefore, it is important that edge information be preserved to improve the overall perceived quality of a texture. As such, more data bits should be allocated for regions with high edge activity.

2.2.2 Texture Activity

To evaluate the texture activity of a texture block, a texture rating (TR) is determined using the following metric:

$$TR = s^2_{\text{block}} \qquad (6)$$

where s^2_{block} is the spatial variance of pixel intensities within the texture block. A high texture rating indicates high texture activity. The human vision system is less sensitive to visual degradation in regions with high texture activity than those containing high edge activity [10]. Therefore, these regions with high texture activity can be represented with less data than regions with high edge activity. Furthermore, regions with low texture activity such as smooth constant regions can be represented using fewer data bits than regions with high texture activity without experiencing a noticeable degradation in visual quality. The spatial variance is a good metric for evaluating texture activity and has the added benefit of having relatively low computational complexity.

2.2.3 Brightness

The overall brightness rating (*BR*) of a texture block is determined using the following metric:

$$BR = \mu_{\text{block}} \qquad (7)$$

where μ_{block} is the sample mean of the pixel intensities within the texture block. The human vision system is less sensitive to visual degradation in dark regions. Therefore, regions with a low brightness rating can be represented with fewer data bits without noticeably degrading visual quality. The sample mean is used as a measurement of brightness because of its low computational complexity.

2.3. Perceptually Adaptive Texture Map Compression

Using the perceptual evaluation metrics described in Section 2.2, it is possible to improve visual quality by allocating more bits to regions within a texture map that improve perceptually from the additional bits. Furthermore, it is also possible to improve compression rates by allocating fewer bits to regions that can be represented with fewer bits without noticeable visual quality degradation. A practical approach to performing adaptive data allocation is to divide the texture map into smaller blocks and classify each block based on its perceptual characteristics. Based on its classification, a block is compressed using different fixed rate texture compression techniques. As such, all blocks within a particular classification have the same compression rate. However, many compression rates may exist within a single texture map. This is the approach taken by the proposed algorithm.

There are a number of important benefits to the proposed approach when applied to texture compression. First, since the bit rate for each block is known based on its classification, efficient random access to individual pixels within a texture map is possible. Second, the use of different classes allows different texture content to be represented with the amount of data bits needed to provide good visual quality. Finally, the block classification process is performed during the production stage of game development when the texture map is compressed and therefore does not add additional overhead during the decompression process.

To achieve high compression rates while preserving visual quality, a number of different

compression techniques are used based on the perceptual characteristics of texture content. The first technique is the use of variable texture block sizes. For algorithms such as CCC, variable block sizes allow for the same base colors to be used for a larger area. While this is unsuitable for regions with a high level of fine detail, such as regions with high edge activity, it is useful for regions with fewer fine details without causing noticeable visual degradation. The second technique is the use of a reduced number of base colors used to represent a block. This technique is well suited for regions where the pixel color values are very similar to each other. An example illustrating this is a uniform region where all pixels are the same color. In such a case, all the pixels within the region can be represented by a single base color value. The final technique is the use of subsampling, where a block is decimated to a smaller size spatially during compression. Random access to an individual pixel within a block can still be achieved by dividing its relative coordinates by the subsampling factor. To illustrate this technique, consider a texture map that was subsampled by a factor of 2 in each dimension and compressed using CCC. To access the color value of a pixel in the texture map at the original coordinates $(x,y) = (5,5)$, the x and y-coordinates are divided by 2 and then rounded to the nearest integer. Therefore, the pixel color value is obtained at coordinates $(x,y) = (3,3)$ in the subsampled texture map. This technique can be performed on a block level. This technique is suitable for smooth regions, as little perceivable detail is lost during the subsampling process for such regions.

3. IMPLEMENTATION

3.1 Perceptual Classification

To demonstrate the effectiveness of perceptually adaptive texture compression, an implementation of the concept was developed based on the CCC compression algorithm. For the purpose of our implementation, the source textures are RGB raster images, where each pixel of an image is represented by 24-bits (8-bits for red, 8-bits for green, and 8-bits for blue). Texture maps are divided into 8×8 macroblocks. The macroblocks are then classified into one of four macroblock classes (LOWEST, LOW, MEDIUM, and HIGH) based on the perceptual metrics described in Section 2.2. The following classification algorithm was used for each macroblock (based on the 8-bit luminance component of each macroblock):

IF ER > t_{edge} THEN

Class = HIGH

ELSE IF TR > $t_{texture\_medium}$ THEN

Class = MEDIUM

ELSE IF TR > $t_{texture\_low}$ AND BR > $t_{brightness}$ THEN

Class = LOW

ELSE

Class = LOWEST

where t_{edge} is the edge activity threshold, $t_{texture\_medium}$ and $t_{texture\_low}$ are the two texture activity thresholds, and $t_{brightness}$ is the brightness threshold. Based on test results with a number of different types of textures, the threshold coefficients that yield the desired balance between compression performance and visual

quality were found to be the following: t_{edge}=1, $t_{texture\_medium}$=5, $t_{texture\_low}$=2, and $t_{brightness}$=10. These parameters can be adjusted by the texture artist to optimize the balance between compression performance and visual quality for a specific texture. The class information is stored as bit-mask with 2 bits being used to identify the macroblock class. This bit-mask allows for the efficient address calculation of each block and facilitates fast random access to individual pixels in a texture map. Once the macroblocks have been classified, they are compressed using the technique chosen for the macroblock class.

3.2 'LOWEST' Macroblock

If a macroblock is classified as LOWEST, the only information stored is a 15-bit R5G5B5 color value representing the mean color value of the block. Including the 2 bits used to identify the macroblock class, a total of 17 bits is required to represent the 8 × 8 macroblock. This results in a bit rate of 0.266 BPP.

3.3 'LOW' Macroblock

If a macroblock is classified as LOW, the block is subsampled by a factor of two in both the horizontal and vertical directions to yield a 4 × 4 block. Two 15-bit color values are stored as base colors. Based on the stored base colors, two 15-bit color values are calculated using linear interpolation. These color values are used as intermediate base colors. Finally, one 32-bit color index is used to indicate which of the four base colors represents the color of a particular pixel. Including the 2 bits used

to identify the macroblock class, a total of 64 bits is required to represent the 8x 8 macroblock. This results in a bit rate of 1 BPP.

3.4 'MEDIUM' Macroblock

If a macroblock is classified as MEDIUM, it is divided into two 8x 4 sub-blocks. For each sub-block, two 15-bit stored color values and two 15-bit interpolated intermediate color values are used as base colors. Finally, one 64-bit color index is used for each sub-block to indicate which of the four base colors represents the color of a particular pixel. Including the 2 bits used to identify the macroblock class, a total of 190 bits is required to represent the 8x 8 macroblock. This results in a bit rate of 3 BPP.

3.5 'HIGH' Macroblock

If a macroblock is classified as HIGH, it is divided into four 4x 4 sub-blocks. For each sub-block, two 15-bit stored color values and six 15-bit interpolated intermediate color values are used as base colors. Finally, one 48-bit color index is used for each sub-block to indicate which of the eight base colors represents the color of a particular pixel. Including the 2 bits used to identify the macroblock class, a total of 314 bits is required to represent the 8x 8 macroblock. This results in a bit rate of 5 BPP.

3.6 'High Quality' Implementation

A higher quality (HQ) variant of the proposed implementation is possible for 3D video games where visual quality is more important than compression performance. This is accomplished by increasing the number of interpolated color values to six for macroblocks classified as 'MEDIUM' and 14 for those classified as 'HIGH'. This results in a total of 378 bits per 'HIGH' macroblock (6 BPP) and 254 bits per 'MEDIUM' macroblock (4 BPP).

4. TESTING METHODS

To test the effectiveness of the proposed implementation, seven texture maps of various sizes and content types were compressed using both the normal variant (NORMAL) and high quality variant (HQ) of the method. The texture maps were also compressed using DXTC for comparison purposes. The chosen textures are typical of those used in 3D video games. The test textures are summarized below.

- **WOOD:** A 512x 512 wood-tiled wall texture obtained from [11].
- **BRICK:** A 256x 256 brick wall texture obtained from [12].
- **CRATE:** A 512x 512 crate box texture obtained from [12].
- **GIRL:** A modified 512x 512 texture of a girl's head based on HL2 Korin model from [13].
- **GIRL2:** A 1024x 1024 texture of a girl's head obtained from HL2 Charlie Vigor model from [14].
- **ROOF:** A 256x 256 texture of a rectangular panel roof obtained from [12].

- **FLOOR:** A 512x 512 wooden floor texture obtained from [12].
- **DOOR:** A 512x 512 door texture obtained from [12].

The thresholds described in Section 3.1 were used for all tests. However, in practical situations, the texture artist can adjust the thresholds to achieve better compression performance or better visual quality. To evaluate the visual quality of the compressed textures in a quantitative manner, the PSNR of the luminance component was measured.

5. EXPERIMENTAL RESULTS

The compression performance results are shown in Table I. Based on the results, it can be seen that the normal implementation of the proposed technique performed noticeably better than DXTC in seven of the eight test cases, and slightly better in the remaining BRICK test case. The reason that better compression was not achieved in the BRICK test case is due to the large number of significant edges within the texture, as edges are preserved with more data than other characteristics. The high quality implementation was able to outperform DXTC noticeably in six of the eight test cases, while performing worse than DXTC in the BRICK and ROOF cases.

The PSNR results providing a quantitative measure of visual quality are shown in Table II. It can be observed that the normal implementation performed noticeably better than DXTC in all test cases. Furthermore, the high quality implementation performed significantly better than both DXTC and the normal implementation. Two of the test cases (GIRL and ROOF) are shown using the three compression methods as illustrated in Figure 3, Figure 4, and Figure 5. The visual quality of the normal and high quality variants is noticeably superior to that produced by the DXTC method. The fine details are significantly more refined in the textures produced using the proposed implementations, with the high quality implementation comparable to the original uncompressed textures. This is particularly noticeable at the ends of the shingles in the ROOF case and around the eyebrow and eye regions in the GIRL case. Therefore, it is clear that the proposed algorithm can be effective at providing high quality texture compression while retaining overall compression performance.

TABLE I
COMPRESSION PERFORMANCE

Texture	**Compression Ratio**		**Improvement Over DXTC**	
	NORMAL	**HQ**	**NORMAL**	**HQ**
WOOD	12.19:1	9.979:1	103.1%	66.31%
BRICK	6.092:1	4.859:1	1.537%	-19.02%
CRATE	11.7:1	9.538:1	95.04%	58.97%
GIRL	10.18:1	7.978:1	69.75%	32.97%
GIRL2	27.85:1	23.6:1	364.2%	293.3%
ROOF	6.7:1	5.239:1	11.67%	-12.68%
FLOOR	12.11:1	10.12:1	101.9%	68.59%
DOOR	7.875:1	6.314:1	31.24%	5.236%
Average	**11.84:1**	**9.70:1**	**97.33%**	**61.66%**

TABLE II
Y-PSNR AFTER COMPRESSION

Texture	DXTC (dB)	Proposed (dB)		PSNR Gain (dB)	
		NORMAL	HQ	NORMAL	HQ
WOOD	37.9444	41.15	44.32	3.209	6.379
BRICK	33.2397	35.64	38.63	2.403	5.386
CRATE	37.7446	40.23	43.23	2.488	5.483
GIRL	42.4887	43.62	46.92	1.129	4.429
GIRL2	48.7520	48.94	49.8	0.188	1.044
ROOF	36.5583	38.4	43.87	1.839	7.315
FLOOR	40.2802	43.94	46.14	3.663	5.86
DOOR	35.6481	39.55	43.94	3.901	8.289
Average	**39.082**	**41.43**	**44.61**	**2.353**	**5.523**

Figure 3: ROOF test case
Top-left: Uncompressed; Top-right: DXTC;
Bottom-left: Normal Implementation;

Bottom-right: High Quality Implementation

Figure 4: GIRL test case

Top-left: Uncompressed; Top-right: DXTC;
Bottom-left: Normal Implementation;
Bottom-right: High Quality Implementation

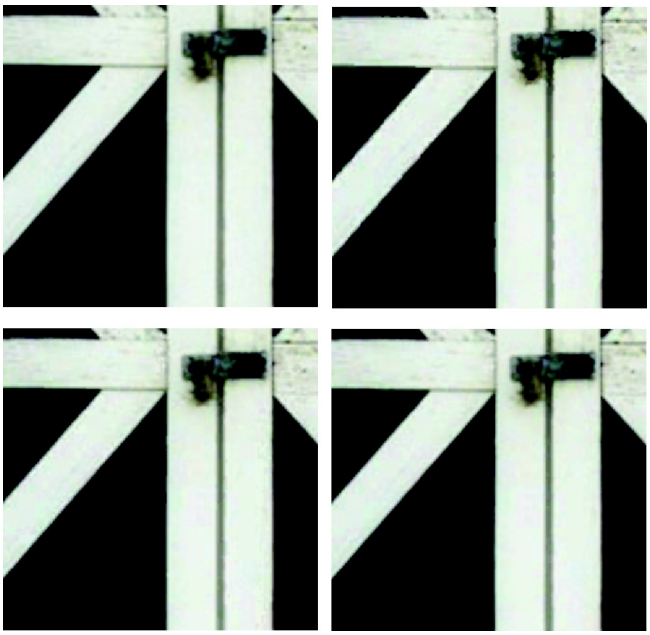

Figure 5: DOOR test case
Top-left: Uncompressed; Top-right: DXTC;
Bottom-left: Normal Implementation;
Bottom-right: High Quality Implementation

6. CONCLUSIONS AND FUTURE WORK

In this paper, we have introduced a practical adaptive method for compressing texture maps based on the human vision system for use in 3D video games. Experimental results demonstrate superior compression performance and visual quality when compared to DXTC. Therefore, this technique can be used to improve visual quality in future 3D video games. Future work includes developing texture compression hardware based on the proposed technique.

ACKNOWLEDGEMENTS

The authors would like to thank Epson Canada and the Natural Sciences and Engineering Research Council of Canada. Special thanks also go to texture author Slap for the HL2 Charlie Vigor textures.

REFERENCES

[1] E. Catmull, "Computer Display of Curved Surfaces," in *Proceedings of IEEE Conference on Computer Graphics, Pattern Recognition, and Data Structures*, pages 11-17, 1975.

[2] J. Blinn, "Simulation of Wrinkled Surfaces," in *Proceedings of the 5th Annual Conference on Computer Graphics and Interactive Techniques*, Vol. 12, No. 3, pages 286-292, 1978.

[3] R. Cook, "Shade Trees," in *Proceedings of the 11th Annual Conference on Computer Graphics and Interactive Techniques*, pages 223-231, 1984.

[4] G. Wallace, "The JPEG Still Picture Compression Standard," *Communications of the ACM*, 34(4), pages 30-34, 1991.

[5] M. Boliek, C. Christopoulos, and E. Majani, "JPEG 2000 Part I Final Committee Draft Version 1.0," http://www.jpeg.org/public/fcd15444-1.pdf, 2000.

[6] G. Campbell, T. DeFanti, J. Frederiksen, S. Joyce, and L. Leske, "Two Bit / Pixel Full Color Encoding," in *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques*, pages 215-223, 1986.

[7] E. Delp and O. Mitchell, "Image Compression Using Block Truncation Coding," in *IEEE Transactions on Communications*, 27(9), pages 1335-1342, 1979.

[8] S. Fenney, "Texture Compression Using Low-Frequency Signal Modulation," in *Proceedings of the ACM SIGGRAPH / EUROGRAPHICS Conference on Graphics Hardware*, pages 84-91, 2003.

[9] J. Ström and T. Akenine-Möller, “iPACKMAN: High-Quality, Low-Complexity Texture Compression for Mobile Phones,” in *Proceedings of the ACM SIGGRAPH / EUROGRAPHICS Conference on Graphics Hardware*, pages 63-70, 2005.

[10] X. Ran and N. Farvardin, “A Perceptually Motivated Three-Component Image Model – Part I: Description of the Model,” *IEEE Transactions on Image Processing*, 4(4): 401-415, 1995.

[11] 3D Excellence.com: Wood Textures, http://www.3dexcellence.com/textures.html, 2005.

[12] PhotoRealistic Texture Pack, http://berneyboy.planetquake.gamespy.com//textures.htm, 2003.

[13] The Model for HL2, http://www11.plala.or.jp/nkinta/hl2.htm, 2006.

[14] Half-Life 2 Charlie Vigor DeathMatch Model, http://halflife2.filefront.com/file/HalfLife_2_Charlie_Vigor_DeathMatch_Model;57119, 2006.