Comparative study of DXT1 texture encoding techniques

Jizhen Ye[1], Jian Wei[2], Yan Huang[1*], Jingliang Peng[1]

[1]*School of Computer Science and Technology, Shandong University, Jinan, China*

[2]*Qualcomm Inc., San Diego, U.S.A.*

Received 1 March 2014, www.tsi.lv

Abstract

In this paper, we make a comprehensive survey of many different methods to implement DXT1 (a widely used lossy texture compression algorithm). Besides that, we propose two new methods that aim for computing speed and image quality, respectively to implement DXT1 texture compression algorithm. For computing speed, we propose a new method called Lsq3d fit which achieves a very fast speed to encode texture images while keeping acceptable image quality. For image quality, we propose a new method called kmeans iteration fit and make a combination of it and the cluster fit from libsquish (an open source lib for DXTC). Kmeans iteration fit performs competitively in the quality of compressed texture images compared with the state-of-the-art DXT1 encoders, and we achieve different levels of quality by controlling the times of iteration. Finally, we test all the methods on Kodak Lossless True Color Image Suite, and CSIQ (Computational Perception and Image Quality Lab) image dataset. Our proposed methods have competitive results of speed and quality in both image datasets. The combination of cluster fit and kmeans iteration fit defeats all other methods in the quality of compressed images.

Keywords: Texture compression, DXTC, DXT1, S3TC, k-means clustering

1 Introduction

Textures play an important role in computer graphics. They are used to increase the realism of the rendered scenes by adding visual details to geometric models. However, textures can not only consume large amounts of system and video memory, but also take up a lot of bandwidth which usually limits the performance of modern rasterizer architectures for computer graphic system [1].

To solve these problems, Knittel et al. [2] and Beers et al. [3] proposed texture compression whose main idea is to conduct lossy compression on texture images, and store the compressed version of the textures. On one hand, as textures have been compressed before they are transferred to video memory by bus, we can save both bandwidth and memory. On the other hand, when accessing the compressed textures during rendering, the compressed textures should be decompressed on-the-fly in real time and support random access. Therefore texture compression is not the same as general image compression. It has its own properties to satisfy peculiar application requirements. We will introduce the main differences between texture compression and image compression in Section 2.1. Most of today's graphics cards allow textures to be stored in a variety of compressed formats that are decompressed in real-time during rasterization [1]. One such format which is supported by most graphics cards is S3TC, also known as DXT compression [4, 5].

The family of DXT compression formats is made up of DXT1, DXT2, DXT3, DXT4 and DXT5. They are different in the way they handle the alpha channel. In this paper, we focus on DXT1 which is the base of other DXT formats. DXT1 [6] is simple, whose basic idea is to divide a texture image into many 4×4 pixel blocks and encode each block independently. Every encoded block is composed of two parts. The first part is used to store two 16-bit RGB565 colours c_0 and c_1. The second part is used to store 16 2-bit colour indices. The structure of encoded DXT1 block is shown in Figure 1. If the first base colour c_0 as a 16-bit unsigned integer is greater than c_1, two other colours c_2 and c_3 are calculated as follows: $c_2 = (2c_0 + c_1)/3$ and $c_3 = (c_0 + 2c_1)/3$. Otherwise, $c_2 = (c_0 + c_1)/2$ and c_3 is transparent black. The indices are used to determine the colour value for each pixel. The base colours c_0 and c_1 are the most important to determine the colour quality of each bock. How to choose two base colours that can best represent the 4×4 block has been the main focus of DXT1.

In this paper, we make a comprehensive survey of many different encoding techniques conforming to the DXT1 texture compression standard, and test these methods on two widely used image datasets (CSIQ and Kodak). We also propose two new DXT1 texture encoding algorithms that aim for computing speed and image quality, respectively. Experimental results on Kodak image dataset and CSIQ image data set indicate that our methods have outstanding performance in speed or quality.

[*] ***Corresponding author*** e-mail: yan.h@sdu.edu.cn

2 Related work

This section introduces the main differences between general image compression and texture compression.

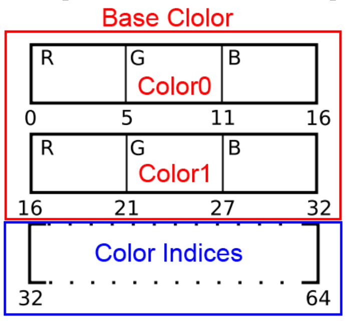

FIGURE 1 The structure of DXT1 encoded block. The red part is two base colours and the blue part is each pixel's colour indices to four colours c_0, c_1, c_2 and c_3

2.1 GENERAL IMAGE COMPRESSION VS TEXTURE COMPRESSION

Texture compression is a kind of special compression method which has its own properties to satisfy specific application requirements. Beers et al. [3] list four factors that should be considered when evaluating a texture compression scheme, as described below.

Decoding speed: The accessing of texture data is a critical part in the texturing operations. So it is highly desirable to be able to render directly from the compressed texture data. The decoding algorithm of texture compression should be relatively simple to reduce the cost of hardware and should not impact rendering performance.

Random access: As objects may be oriented and obscured arbitrarily, it is therefore difficult to predict the order that a renderer accesses texels. As such, any texture compression scheme must allow fast random access to compressed texture data.

Encoding speed: In most applications that are related to textures, the majority of textures are compressed well in advance of the rendering, which we often call off-line encoding. It is therefore feasible to employ a scheme where the encoding is considerably slower than the decoding.

Compression rate and visual quality: Because the most important issue in texturing is the quality of the rendered scene rather than the quality of individual textures themselves, some loss of fidelity in the compressed texture is more tolerable than image compression.

From the above descriptions, we know that most general image compression schemes, e.g. JPEG, cannot support direct random access of pixel data within the compressed format because the per-pixel storage rate varies throughout. As such, many texture compression schemes we introduced later, which employ 'fixed rate' encoding.

2.2 HISTORY OF DXTC

In 1979, Delp and Mitchell [7] developed a simple scheme, called block truncation coding (BTC) for image compression. BTC compressed grey scale images in blocks of 4x4 pixels. For each block, two representative 8-bit grey scale values are chosen and each pixel within the block is quantized to either of these two values. This resulted in 2 bits per pixel (2bpp).

Campbell et al. [8] presented colour cell compression (CCC), which is often seen as a simple extension of BTC. CCC stores two colour indices to a palette in a 4x4 block instead of 2 grey scale values in BTC. By using a 256-wide colour palette, the colours can be represented with eight bits each. Thus CCC can encode colour images at 2 bpp. However, the limitation of only two colours in a 4x4 block gives rise to banding artifacts and CCC requires a memory lookup in the palette. Knittel et al. [2] suggested that CCC be implemented in hardware and used in a texturing system.

The S3TC texture compression method by Iourcha et al. [9] is a further adaption of the BTC/CCC method by improving colour data encoding. S3TC (a.k.a DXTC) is the de facto standard in texture compression method. Unlike CCC, it stores two base colours in R5G6B5 format and a 2-bit index for each pixel in a 4x4 block. Each pixel can have four colours to choose, two base colours and two additional colours in-between the base colours.

2.3 EXISTING IMPLEMENTATIONS OF DXT1 ENCODER

From the above descriptions, we know that all DXTC's colours in a block lie on a line in colour space of RGB. To choose two base colours that can best represent all the colours of each block is the main work of DXTC encoder. There are several good DXT compressors available. Such as the ATI Compressonator [10] and the nVidia DXT Library [11], squish library [12], crunch lib [13], LSDxt DXT Compressor [14], Jason Dorie's image library: ImageLib, Mesa S3TC compression library: libtxc_dxtn [16] and so on.

All the above encoders produce different levels of quality to DXT compressed texture images, and some of them are not open source. So it is very meaningful to give a comparative study on these different methods.

It is known that texture compression does not require real-time encoding integral, but in some particular applications real-time compression is also important. A good DXT encoder should provide two different choices for users to choose. One is to compress a texture image very fast, while the quality of the texture can have more error tolerance. The other is to compress a texture with more time and produce high quality of DXT compressed texture images.

3 Encoding approaches

This section includes two parts. Part one introduces some encoding techniques in the open source squish lib. Part two describes two of our novel methods and a combination of cluster fit and kmeans iteration fit which has the best quality of all methods.

3.1 METHODS OF SQUISH LIBRARY

The squish library (abbreviated to libsquish) is an open source DXT compression library that was originally written by Simon Brown et al. Range fit and Cluster fit are based on a concept called principal component [15].

3.1.1 Range fit

For range fit method, it takes the minimum and maximum point along the principal axis as the endpoints directly. Although this method is quite simple and there may be better colour endpoints that are not part of the original point set, it can find the base colour points very fast and keep acceptable quality of compressed texture images. So range fit is a good choice for those applications that require very fast encoding speed, also known as real-time compression applications.

3.1.2 Cluster fit

Range fit takes endpoints from the original colour set, this is not the best choice in most cases. Cluster fit method is under the assumption that: If we assume that the principal axis is very similar to the direction of the line through optimal endpoints, we can also assume that a total ordering of the original colour set in these directions is also very similar. So cluster fit uses the principal axis to define a total ordering on the original colour set. It then tests all possible ways of clustering the original points that preserve this total ordering, and fit endpoints to each generated index set using least squares.

The cluster fit algorithm in squish now forms the core DXT compression algorithm for the NVIDIA Texture Tools.

3.2 OUR NOVEL METHODS AND COMBINATION METHOD

In this section, we will present our novel methods in detail.

3.2.1 Lsq3d fit

It is known that the method of Least Squares is a procedure to determine the best fit line to data.

Line fitting in computational mathematics often fits lines in 2D space with least square method. The fitted equation is $y = ax + b$. However, we want to fit lines in three-dimensional RGB space. We cannot use the linear least square method directly to determine the linear equation of 3D line. We refer to a math paper: Fitting of the Straight Line Equation in Space [17] to fit space line equation. Its base idea is to convert space line equation to plane line equation.

With the space line, Lsq3d fit takes two endpoints that have the maximum span in the line. Unfortunately, sometimes space line cannot be represented by the above equations. We choose range fit to encode those corresponding blocks. This Lsq3d fit method is proposed for high computing efficiency.

3.2.2 Kmeans iteration fit

Kmeans iteration fit method is based on k-means clustering algorithm which aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster. For each DXT block, we partition 16 colour points into 4 clusters firstly. Then we minimize Equation (1) that calculates the Euclidean distance between the DXT base colour points and cluster centre points.

$$f(x_0, y_0, z_0, x_1, y_1, z_1) = \sum_{i \in clusters[j]} \sum_{j=0}^{3} w_i (Observations[i] - Centers[j])^2 , \quad (1)$$

where clusters is the result of kmeans clustering, *Centers* is the centre of each cluster, *Observations* is the cluster member of each cluster, and w is the weight of each cluster, it is calculated by *clusters*[*i*].number/16.0.

To minimize $f(x_0, y_0, z_0, x_1, y_1, z_1)$, we calculate its partial derivative of each parameter.

Solving this linear equations, we can obtain values of x_0, y_0, z_0, x_1, y_1 and z_1, which are the coordinates of the DXT base colours.

Kmeans iteration fit can iterate for a user-specified number of times. The $c_0(x_0, y_0, z_0)$ and $c_1(x_1, y_1, z_1)$ are the base values. We feed c_0, c_1, c_2 and c_3 as the unit points to the next round of kmeans clustering. c_2 and c_3 are the interpolated values of c_0 and c_1.

With the initial points, we can get new cluster results and new values of x_0, y_0, z_0, x_1, y_1 and z_1. If the distance error between previous endpoints and current endpoints is less than a specified threshold then the algorithm is terminated; otherwise the iteration process continues until the distance error is smaller than the threshold or the number of iteration reaches the maximum iteration number. The definition of the distance error is shown in Equation (2):

$$DisError = \frac{\sum_{i=0}^{3} (\text{curFitpoints[i]} - \text{preFtipoints[i]})^2}{MaxSpan} , \quad (2)$$

where curFitpoints[i] is the fitted results of this round and preFitpoints[i] is the results of previous round. MaxSpan is the longest distance of any two points in the block.

Figure 2 presents the whole algorithm flow of kmeans iteration fit.

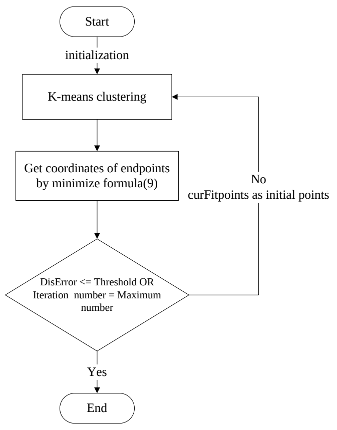
Start
initialization
K-means clustering
Get coordinates of endpoints by minimize formula(9)
No
curFitpoints as initial points
DisError <= Threshold OR Iteration number = Maximum number
Yes
End

FIGURE 2 The algorithm flow chart of kmeans iteration fit. Initialization specifies some parameters to execute k-means clustering

3.2.3. Combination

We observe many fitting results (as shown in Figure 3) of different encoding methods and find that our method can have a great fitting effectiveness when the colour set in a block has a fine linearity. In order to improve the encoding method, we propose a combination version.

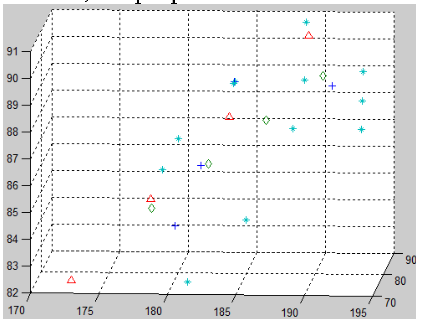

FIGURE 3 The fitting results of a block. Where the cyan stars represent the colour points, red triangles are fitting points of squish's cluster fit, blue add symbols represent cluster centres and the green diamonds are the fitting results of our kmeans iteration fit

For the combination method, we should make a judgment before we encode a block. If the colour points are of high linearity, then we choose kmeans iteration fit to encode. Else the colour points are encoded by cluster fit. We choose two factors to evaluate the linearity of colour points. One is the variance of the distances of cluster centres, and the other is the angle of centre lines. The specific formulas are shown in Equation (3):

$$Dis_{ij} = (center[j] - center[i])^2,$$
$$Variance = \frac{(Dis_{ij} - \overline{Dis})^2}{3}, \qquad (3)$$
$$\theta = \mathrm{Max}(\angle Dir_{01}Dir_{02}, \angle Dir_{32}Dir_{31}),$$

where $i = \{0, 1, 2\}$, $j = i + 1$, Dis_{ij} is the distance between *center*[*i*] and *center*[*j*]. Variance represents the variance of Dis_{01}, Dis_{12} and Dis_{23}.

We specify the value of *Variance* and θ by a lot of experiments. The experimental results indicate that this combination method has the best quality of compressed texture images, that is to say this method yields the highest PSNR values and the smallest *RmsError* values.

The flowchart of the combination method are shown in Figure 4.

4 Datasets and quality evaluating metrics

In this section, we introduce two most popular available image datasets that are used to test all DXT1 encoding methods. They are Kodak Image Dataset [18] and CSIQ (Computational Perception and Image Quality Lab) image dataset [19]. The quality evaluation metrics used in our experiment are introduced too.

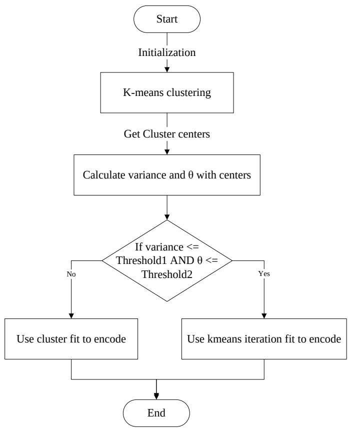
Start
Initialization
K-means clustering
Get Cluster centers
Calculate variance and θ with centers
If variance <= Threshold1 AND θ <= Threshold2
No
Yes
Use cluster fit to encode
Use kmeans iteration fit to encode
End

FIGURE 4 The flowchart of combination method. Initialization specifies some parameters to execute k-means clustering

Kodak Image Dataset [18] is released by the Kodak Corporation for unrestricted research usage. There are totally 25 uncompressed PNG true colour images of size 768×512 pixels in it. CSIQ (Computational Perception and Image Quality Lab) image dataset [19] was released by Computational Perception and Image Quality Lab. It consists of 30 original PNG images and six different types of distortions at four to five different levels of distortion. We only use the original images in our experiment. All 30 original images are of size 512×512 pixels and can be divided into five different types and 6 images for each type. They are animals, landscape, people, plants and urban.

In our experiment, we choose *RmsError* which is the square root of *MSE* (Mean Square Error) and *PSNR* (Peak signal-to-noise ratio) as quality evaluating metrics.

PSNR is most commonly used to measure the quality of reconstruction of loss compression codecs (e.g., for image compression). The signal in this case is the original data, and the noise is the error introduced by compression.

MSE is shown in Equation (4):

$$MSE = \frac{1}{3}\frac{1}{mn}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}[\mathrm{R}_{I(i,j)} - R_{C(i,j)}]^2 + [G_{I(i,j)} - G_{C(i,j)}]^2 + [\mathrm{B}_{I(i,j)} - B_{C(i,j)}]^2 , \tag{4}$$

where *I* is the original texture image and *C* is the compressed texture image. *m* is the width of the image and n is the height of the image. As we will evaluate the quality of RGB image, we should consider RGB three channels.

PSNR is calculated with Equation (5), where max is 255:

$$PSNR = 10\log_{10}\left(\frac{\mathrm{MAX}_I^2}{MSE}\right). \tag{5}$$

5 Experimental results

In this section, we use Kodak Image Dataset [18] and CSIQ dataset [19] to test all the DXT1 encoders we can find. We present the experimental results with tables and line charts.

Firstly, we will introduce the running environment of our experiment. All the programs run on Microsoft Visual Studio 2008 professional version. The computer used to conduct experiment is DELL OptiPlex 7010, its CPU is Intel® Core™ i5-3470 @3.20GHz, its Operation System is Windows 7 Ultimate Edition and its RAM is 4.00 GB.

Then, we present parameter settings of our own methods. For kmeans iteration fit, we set the DisError equal to 0.001. To balance the quality and speed, we set iteration times to be five. The result of kmeans iteration fit without iteration is also given to make a comparison. For combination method, the variance threshold is 1.0 and the cosθ threshold is 0.86. We determine these values by a lot of experiments and choose thresholds that have the best effects.

The numerical experimental results can be seen in Table 1. We measure *rmsError*, *PSNR*(db) and running time(second) for each image dataset, all these values are average values of corresponding datasets. For each numerical term, the left column is the results of Kodak dataset and the right column is CSIQ's results. As some methods only offer software or executable files, we cannot measure the running time accurately as we do in methods that have source code and the Compressonator of AMD [10] in quality no matter how many times it iterates. The combination methods (marked in blue) can have the best quality of all methods, but it needs longer time to encode. Improving the speed of this combination method is one of our future works.

TABLE 1 Experimental Results of all the methods on the two image datasets. "Can't measure accurately" means that we have to make compression operations interactively, so the total time used to compress is not accurate. The red row is Lsq3d fit, which is the fastest method to encode, while the quality is still acceptable and defeat range fit both in time and quality. The blue row is the combination method, which has the best quality of all methods

Methods	**Kodak**	**CSIQ**	**Kodak**	**CSIQ**	**Kodak**	**CSIQ**
	Mean RmsError		**Mean PSNR(db)**		**Running time(second)**	
Range fit	9.615789	11.991445	33.47758	31.713790	1.457000	1.215000
Lsq3d fit(Range fit for blocks can't express line)	8.994410	11.723098	34.057239	32.018587	0.747000	0.650000
Cluster fit + single color fit	6.982464	8.799585	36.26763	34.383929	201.629000	167.624000
LSDxt Engine by L. Spiro	7.621329	9.577089	35.51408	33.662967	Can't measure accurately	Can't measure accurately
AMD: The Compressonator	7.006528	8.860144	36.230736	34.312864	Can't measure accurately	Can't measure accurately
Crunch lib quality=0	19.036631	22.901982	27.683062	26.243883	Can't measure accurately	Can't measure accurately
Crunch lib quality=255	6.976277	8.866489	36.275241	34.317618	Can't measure accurately	Can't measure accurately
Kmeans iteration0 fit	7.673874	9.305032	35.414205	33.866736	23.667000	19.702000
Kmeans iteration5 fit	7.352763	9.110025	35.764885	34.049590	65.608000	54.208000
Combination(cluster fit and kmeans iteration fit)	6.855624	8.619251	36.431254	34.556220	235.573000	197.196000

Table 1 shows that Lsq3d fit (marked in red) can defeat range fit both in speed and quality. Kmeans iteration fit can have competitive performance compared with other methods, but it cannot be better than cluster fit.

In order to give a better visualization of all the methods' compression performance on every texture image, we present two line charts for Kodak [18] and CSIQ [19] respectively as shown in Figures 5 and 6.

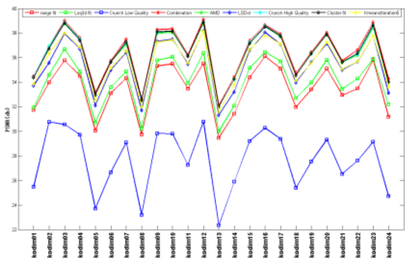

FIGURE 5 The line chart of Kodak dataset. X axis represents image names and Y axis represents the PSNR(db) value of each image

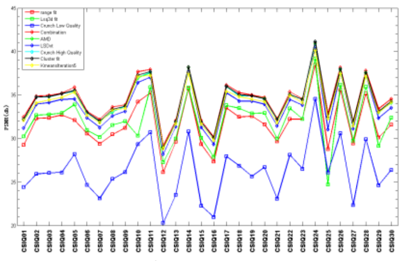

FIGURE 6. The line chart of CSIQ dataset. X axis represents image names and Y axis represents the PSNR(db) value of each image

In these two line charts, we present 9 different methods's *PSNR* values on each image. The corresponding methods are shown in the legend of the figure. The square symbols represent fast compression methods and the diamond symbols represent the high quality methods. To make it clearer, we use different colours for different methods. The line charts indicate that the combination method has the highest PSNR values for most images.

6 Conclusion and future work

DXTC has been a leading texture compression method for many years and is still widely used today. Correspondingly, many people focus on the implementation of DXT1 encoder. In this paper, we make a comparative study on all the encoders we can find and test them on two image datasets. Besides that, we propose our own method Lsq3d fit and k-means iteration fit aiming at speed and quality, respectively. The combination of our k-means iteration fit and cluster fit outperforms all the other methods in quality of compressed texture images.

We will extend our work to alpha channel encoding in the future. For textures with alpha channel, DXTC uses the same idea as for colour channels to encode alpha values. The most important thing is also to find the alpha endpoints that can best represent the original 4 x 4 block. New encoding method should be designed specifically for alpha channel in the future.

Acknowledgment

We thank Qualcomm Inc. for funding this project. We also thank Simon Brown who has written the squish lib and shared the source code.

References

[1] Aila T, Miettinen V, Nordlund P 2003 Delay Streams for Graphics Hardware *ACM Transactions on Graphics*, **22**(3) 792–800

[2] Knittel G, Schilling A, Kugler A, Strasser W 1996 Hardware for Superior Texture Performance *Computers & Graphics* **20**(4) July 475–81

[3] Beers A, Agrawala M, Chadda N 1996 Rendering from Compressed Textures *Proceedings of SIGGRAPH* 373–8

[4] S3 Texture Compression Pat Brown NVIDIA Corporation November 2001 Available Online: http://oss.Sgi.com/projects /oglsample/registry/EXT/texture_compression_s3tc.txt

[5] Compressed Texture Resources Microsoft Developer Network DirectX SDK, April 2006

[6] Brown P, Agopian M EXT texture compression dxt1. opengl extension registry. http://opengl.org/registry/specs/EXT/texture_compression_dxt1.txt

[7] Delp E, Mitchell O 1979 Image Compression using Block Truncation Coding *IEEE Transactions on Communications* **2**(9) 1335–42

[8] Campbell G, Defanti T A, Frederiksen J, Joyce S A, Leske L A, Lindberg J A, Sandin D J 1986 Two Bit/Pixel Full Color Encoding *Proceedings of SIGGRAPH* **22** 215–23

[9] Iourcha K, Nayak K, Hong Z 1999 System and Method for Fixed-Rate Block-based Image Compression with Inferred Pixels Values *US Patent* 5,956,431

[10] ATI Compressonator Library Seth Sowerby Daniel Killebrew ATI Technologies Inc The Compressonator version 1.27.1066

[11] NVidia DXT Library nVidia nVidia DDS Utilities April 2006 Available Online http://developer.nvidia.com/object/ nv_texture_tools.html

[12] Libsquish, open source DXT compression library writed by Simon Brown 2006 Available Online: http://code.google.com/p/libsquish/

[13] Crunch, advanced DXT texture compression and real-time transcoding library. Available Online: https://code.google.com/p/crunch/

[14] LSDxt DXT compressor by L Spiro October 11 2012. Available Online: http://lspiroengine.com/?p=516

[15] Pearson K, 1901 "On Lines and Planes of Closest Fit to Systems of Points in Space" Philosophical Magazine **2**(11) 559–72.

[16] Mesa S3TC Compression Library Roland Scheidegger libtxc_dxtn version 0.1 May 2006

[17] Huo X,2009 Fitting of the Straight Line Equation in Space *Journal Of Huaihua University* **28**(2) Feb 2009

[18] Kodak Image Dataset released by the Kodak Corporation for unrestricted research usage (Image source: http://r0k.us/graphics/Kodak)

[19] Larson E C, Chandler D M 2010 Most apparent distortion: full-reference image quality assessment and the role of strategy *J Electr Imaging* **19** 001006 1-21 2010

Authors

	Jizhen Ye, born in 1990, Fujian, China. **University studies:** Master student, School of Computer Science and Technology, Shandong University, Jinan, China
	Jian Wei, born in 1969, Xian, China. **Current position:** Researcher, Quallcomm multi-media R&D, San Diego, U.S.A. **Scientific interest:** computer vision, graphic technology.
	Yan Huang, born in 1974, Tongling, Anhui, China. **Current position, grades:** associate professor in the School of Computer Science and Technology, Shandong University. **Scientific interest:** Large scale 3D data visualization, intelligent multimedia data analysis.
	Jingliang Peng, born in 1974, Feixian, Shandong, China. **Current position, grades:** professor in the School of Computer Science and Technology, Shandong University. **Scientific interest:** digital geometry processing, content-based multi-media data retrieval, image analysis and understanding.