Fast compression of computer-generated holographic images based on a GPU-accelerated skip-dimension vector quantization method

Y. K. Lam*, W. C. Situ, and P. W. M. Tsang

Department of Electronic Engineering, City University of Hong Kong, 83 Tat Chee Avenue, Kowloon, Hong Kong, China

**Corresponding author: kinglam4@student.cityu.edu.hk*

Received December 25, 2012; accepted January 27, 2013; posted online April 19, 2013

A method for fast and low bit-rate compression of digital holograms based on a new vector quantization (VQ) method known as the skip-dimension VQ (SDVQ) is proposed. Briefly, a complex hologram is converted into a real off-axis hologram, and partitioned into a set of image vectors. The image vectors are passed into a graphic processing unit (GPU), and compressed through SDVQ into a set of code indices considerably smaller in data size than the source hologram. Experimental evaluation reveals that our scheme is capable of compressing a digital hologram to a compression ratio of over 500 times, in approximately 20–22 ms.

OCIS codes: 090.0090, 100.0100, 200.0200.

doi: 10.3788/COL201311.050901.

In the past two decades, investigation on computer-generated holography (CGH)[1] has become an area of immense interest. The approach enables holograms of real objects and synthetic computer graphic (CG) models to be implemented without the expensive and precision setup of optical equipments. Fueled by the rapid advancement of computing technology, it has been anticipated that CGH can become a novel technological advancement for three-dimensional (3D) televisions[2]. In practice, a digital Fresnel CGH can be generated using the point light concept, which, as explained in Ref. [3], is a numerical realization of the zone-plate approach in holography. Mathematically, given a set of self-illuminating 3D object points $O(x,y,z) = [o_0(x_0, y_0, z_0), o_1(x_1, y_1, z_1), \cdots, o_{N-1}(x_{N-1}, y_{N-1}, z_{N-1})]$, the digital Fresnel hologram $H(x,y)$ can be generated by

$$H(x,y) = \sum_{j=0}^{N-1} \frac{a_j}{r_j} \exp\left(\mathrm{i} w_n r_j\right), \tag{1}$$

where $w_n = 2\pi/\lambda$ is the wave number of the laser light, with λ is the wavelength of the laser, and a_j represents the amplitude of the jth object point. The term $r_j = \sqrt{(x_j - x)^2 + (y_j - y)^2 + z_j^2}$ is the distance between an object point at (x_j, y_j, z_j) and a point (x,y) on the hologram. Without loss of generality, the hologram is positioned on the $x-y$ plane (i.e., on the z= 0 plane), and the object point is at an axial distance z_j away from the hologram. Subsequently, the source 3D scene can be reconstructed by displaying it on a high resolution display, e.g., the liquid crystal on silicon (LCOS) device. One major concern associated with CGH is the enormous amount of data required to record a hologram. In the past, a number of solutions have been proposed to provide moderate hologram compression, including, but are not limited to, the early works of Sasaki *et al.*[4], and more recently, the use of Fresnelets[5], companding[6], histogram approach[7], and vector quantization (VQ)[8–10]. Among these methods, the VQ scheme[9,10] has particularly attractive considerable attention. Briefly, a complex hologram is converted into a real off-axis (ROA) hologram, and decomposed into a set of image vectors. VQ is applied to determine the index of the nearest codevector (based on the Euclidean distance) from a given collection of vectors known as the codebook for each image vector. The list of code indices is either stored as a compressed file, or transmitted to the receiving end via certain distribution channel(s). The corresponding codevector is fetched from the codebook for each index and taken to approximate the image vector to recover the hologram from the compressed data. Compared with the existing methods, the adoption of VQ in holographic compression has the advantages of high compression ratio, low complexity decompression, and high immunity to corruption of the encoded data.

Despite these favorable outcomes, the method suffers from the shortcoming of a computational intensive encoding process. With a typical personal computer (PC), the time for compressing a moderate-size hologram through central processing unit (CPU) is nearly one second, which is too lengthy for real time application. In this letter, we propose a solution to overcome this problem using a VQ variation known as the "skip-dimension vector quantization" (SDVQ), such that the time taken to determine the nearest codevector for each image codevector can be reduced. Moreover, we encapsulate the encoding process to a form that can be executed in a parallel fashion, based on the fragment shader, in a graphic processing unit (GPU). Experimental evaluation reveals that our proposed method is capable of compressing a 2 048×2 048 (pixel), medium-size hologram in around 20–22 ms.

VQ[11] is a classical compression method that can be interpreted as a mapping $G : R^k \rightarrow C$ converting each vector in a k-dimensional Euclidean space R^k to the nearest counterpart in a finite set $C = \left\{c_i \in R^k : i = [0, N-1]\right\}$ of vectors. The finite set is recorded into a database commonly referred to as the

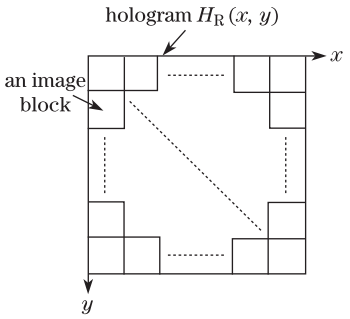

Fig. 1. Partitioning of the integrated hologram plane into non-overlapping blocks.

Table 1. LBG Algorithm in Codebook Generation[12]

Step Operation
1 Generate an Initial Codebook $C = [c_0, c_1, \cdots, c_{N-1}]$ from Randomly Selected Vectors in T.
2 For each Vector in T, Apply Eq. (2) to Locate the Centroid in the Codebook Nearest to It.
3 Update Each Centroid by Averaging the Vectors Associated with It.
4 Determine the Total Error between the Centroids Obtained in Step 3 and the Vectors Associated with Each of Them.
5 If the Error is Below a Pre-defined Threshold, Terminate the Process and Output the Codebook C.
6 Go to Step 2.

codebook. c_i, i, and N are referred to as the codevector, index (i.e., the transmission label), and codebook size, respectively. In the compression process, each vector s_j in a set of source data $S = \{s_0, s_1, \cdots, s_{M-1}\}_{s_j \in R^k|_{0 \leqslant j < M}}$ is mapped to the nearest codevector c_q in C according to the minimum Euclidean distance criteria, i.e., $d(s_j, c_q) \leqslant d(s_j, c_i)$ for $i = [0, N-1]$. The term $d(s_j, c_q)$ represents the Euclidean distance (sometimes referred to as the error) between the codevector and the image vector, given by

$$d(s_j, c_q) = \sqrt{\sum_{m=0}^{k-1} \left(s_j^m - c_q^m\right)^2}, \tag{2}$$

where s_j^m and c_q^m represent the mth dimension of vectors s_j and c_q, respectively.

After compression with VQ, each of the k-dimensional vector s_j in the set S is replaced by an integer index i. The index i is taken to extract the corresponding vector c_q in the codebook to approximate the input vector s_j to recover the source data from the compressed data. Suppose the value of the input vector is quantized with Q bits and there are N vectors in the codebook (where N is a power of 2), the compression ratio (CR) is then given by

$$\mathrm{CR} = \frac{k \times Q}{\log_2 N}. \tag{3}$$

Compressing a set of vectors S with VQ requires a codebook C. This process is usually conducted with the Linde-Buzo-Gray (LBG) algorithm[12] by clustering a set of vectors $T = \{t_0, t_1, \cdots, t_{N-1}\}$ into N codevectors (also referred to as the centroids). The details have been reported in Ref. [12]; thus, this study will only show the essential steps, as listed in Table 1. If C is generated from the source data to be compressed (i.e., $T = S$), the codebook is termed in-training set. Otherwise, it will be referred to as out-training set.

The complex hologram is denoted by $H(x, y)$, where x and y are the horizontal and vertical axis. Reducing the data size of the hologram, $H(x, y)$ is first converted into a ROA hologram by mixing with a inclined reference plane wave $B(y)$, and preserving the real part of the result as

$$H_{\mathrm{R}}(x, y) = \mathrm{Re}\left[B(y)\, H(x, y)\right], \tag{4}$$

where Re[·] represents the real part of the quantity being bracketed, and $H_{\mathrm{R}}(x, y)$ is a real hologram plane that is only half the data size of the complex hologram. When $H_{\mathrm{R}}(x, y)$ is illuminated with an off-axis planar beam, a virtual and a real twin images are reconstructed. However, these two images are separated from each other by an angle due to the incorporation of the off-axis reference beam. $H_{\mathrm{R}}(x, y)$ is thereby subsequently compressed with VQ.

Next we describe the application of classical VQ in compressing the ROA hologram obtained above. Initially, $H_{\mathrm{R}}(x, y)$ is partitioned into non-overlapping square image blocks of size $b \times b$, as shown in Fig. 1. An enlarged view of the jth image block in the hologram is shown in Fig. 2, illustrating the labeling of each pixel from first element $p_{j;0}$ at the top-left corner to the last element $p_{j;b^2-1}$ at the bottom-right corner. The pixel sequence in the image block is represented with a k-dimensional vector s_j, where $s_j^m = p_{j;m}\big|_{0 \leqslant m < b^2}$ and $k = b^2$. Given the horizontal and vertical extents of the hologram as X and Y, respectively, there will be Y/b blocks in each row and X/b blocks in each column, constituting to a total of $M = XY/b^2$ vectors. A collection of vectors from the M image blocks forms the data set $S = \{s_0, s_1, \cdots, s_{M-1}\}_{s_j \in R^k|_{0 \leqslant j < M}}$. If $H_{\mathrm{R}}(x, y)$ is utilized to build a codebook, its set of vectors S is trained by the LBG algorithm depicted in Table 1 to generate the codebook C.

If $H_{\mathrm{R}}(x, y)$ is a hologram to be compressed, the error between each vector in S and all the codevectors in C are computed using Eq. (2). Subsequently, each vector

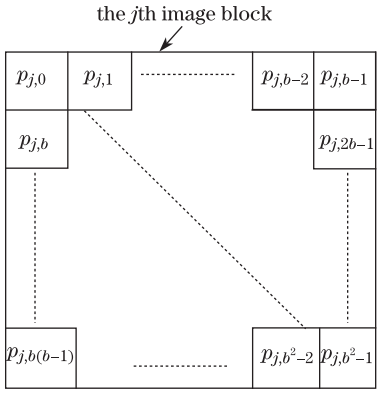

Fig. 2. Enlarged view of an image block in $H_{\mathrm{R}}(x, y)$.

Table 2. Texture of a Single k-dimensional Image Vector

Texel	0	1	—	$(k/4)-1$
Image Vector	$p_{j;0}, p_{j;1}, p_{j;2}, p_{j;3}$	$p_{j;4}, p_{j;5}, p_{j;6}, p_{j;7}$	—	$p_{j;k-4}, p_{j;k-3}, p_{j;k-2}, p_{j;k-1}$

is labeled by the index of codevector C, which is nearest in terms of the distance among all the members in the codebook. This process results in a sequence of integer labels $L = \{l_0, l_1, \cdots, l_{M-1}\}$, each corresponding to a unique block vector in S. Each index l_j will be taken to extract the corresponding vector in the codebook C to recover the hologram from the compressed data. The retrieved vectors approximate, with certain amount of distortion, the vectors in the original data set S. The decoding process only requires negligible amount of computation and memory requirement, constituting one of the major advantages of VQ compression.

Although the VQ decoder is near computation free, the compression process is computationally intensive because Eq. (2) has to be repetitively applied to determine the nearest codevector for each image vector. In the evaluation of the distance between each pair of vectors, the square of the difference for each dimension (from $m = 0$ to $k-1$) has to be computed. With a typical PC, the encoding time for a $2\,048 \times 2\,048$ (pixel) hologram is nearly one second, which is too lengthy for real time video application. Therefore, we propose two measures to accelerate the compression process. Firstly, we have developed a new VQ scheme known as SDVQ to lower the average number of dimensions of arithmetic operations in Eq. (2). Secondly, the SDVQ compression process is encapsulated as a texture rendering process that can be executed with the fragment shader in GPU. As will be explained later, the nearest centroids for all the image vectors can be conducted concurrently, resulting in a significant reduction in the computation time. The concept of conducting the SDVQ compression in the GPU is illustrated in Fig. 3.

Essentially, the codevectors are preloaded into the GPU and converted into the first graphic texture (texture 1). Subsequently, the Fresnel hologram to be compressed is decomposed into a set of image vectors, each representing a non-overlapping square block of the hologram. The image vectors are then passed to the GPU, and converted into the second graphic texture (texture 2). In the GPU

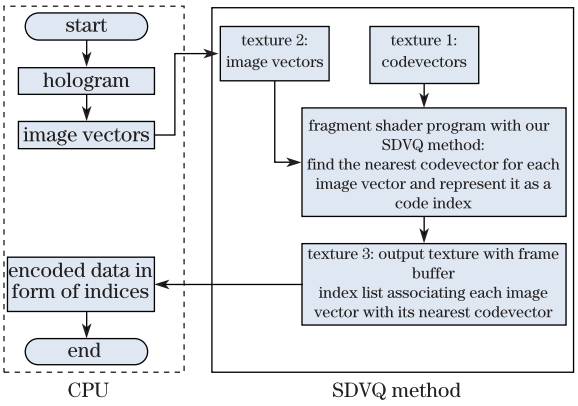

Fig. 3. Illustration of the SDVQ compression process into a texture rendering form conducted in the GPU.

memory, a texture is organized as a sequence of texels, each recording four dimensions of an image vector or a codevector. As an example, the texture of a single image vector is shown in Table 2. The pair of textures is input to the fragment shader, where the nearest codevector for each image vector is computed using our skip-dimension method. Finally, the output of the fragment shader is stored in the output texture (texture 3), and returned to the CPU as a list of indices associating each image vector with its nearest codevector, thereby forming the encoding data.

A fragment shader is a program, written in the Cg language, which computes the error between each image vector with all the codevectors in the codebook. The codevector index with the smallest error will be used to represent the image vector. In contrast to a program that runs on a CPU, the codevectors and image vectors are stored as textures. Thus, the GPU can evaluate all the image vectors concurrently, leading to significant increase in computation speed. The implementation of the SDVQ compression method is outlined as follows. As explained previously, other image vectors will be evaluated simultaneously using the same methods. Initially, the error between the image vector and the first codevector c_0 in the codebook, is computed. The error is recorded in a variable 'best-error', and the code index is set to 0 (corresponding to the first codevector). Subsequently, the accumulated error AE_t (where $t = [0, k-1]$) is deduced in a recursive manner for each of the remaining codevectors, as given by

$$\mathrm{AE}_{t+1} = \mathrm{AE}_t + \left(s_j^t - c_q^t\right)^2, \tag{5}$$

where AE_0=0. The recursive process in Eq. (5) is allowed to progress from the current dimension t to the next dimension $(t+1)$ if the accumulated error AE_{t+1} is smaller than the best-error; otherwise the operation will be "pre-terminated" and evaluation on the remaining dimensions will be skipped. If Eq. (5) has been applied to all the k dimensions and the total accumulated error AE_{t+1} is smaller than the best-error, the latter and the code index will be replaced by AE_{t+1} as well as the index of the corresponding centroid, respectively. After all the centroids have been evaluated, the code index pointing to the codevector nearest the image vector will be returned. The method is faster than classical VQ, as the recursive evaluation of some dimensions of codevectors will be skipped in our proposed method using Eq. (5). In view of the latter mechanism, we have named our proposed method as the SDVQ.

The pair of test images, "lenna's eyes" and "baboon's eyes" shown in Figs. 4(a) and (b), together with the random image in Fig. 4(c) for generating the codebook, are employed to demonstrate the effectiveness of our proposed method. Both test images are planar and located at an axial distance of 0.4 m from the hologram. Equation (1) is applied to generate the ROA holograms for these two images based on the optical settings in

Table 3. The numerical reconstructed images of the pair of holograms at the focal plane are shown in Figs. 5(a) and (b). We observed that the reconstructed images are similar to the original ones in Figs. 4(a) and (b). Subsequently, classical VQ is applied and executed on a typical PC to compress each ROA hologram with a block size of 16×16 (i.e., $b = 16$), based on an out-training-set codebook comprising of 256 code vectors. The codebook is generated using the LBG algorithm in Table 1, applied to the ROA hologram of the random image (which is also positioned at 0.4 m from the hologram) in Fig. 4(c). The numerical reconstructed images of the compressed hologram are shown in Figs 6(a) and (b). Apart from some distortion, they are similar to the ones obtained from the original ROA holograms. The time taken to compress each hologram, with the compression process executed in the CPU of a typical PC, is about one second. Comparing with the original complex hologram, a compression ratio of 512 times is attained.

We applied our proposed SDVQ method, executed with the fragment shader in GPU, to compress the pair of ROA holograms. The time taken to compress the ROA holograms representing the images in Figs. 4(a) and (b) are 20 and 22 ms, respectively, which is over 40 times faster than the use of the classical VQ compression executing with a CPU. Note that our proposed method only increases the speed of the compression process, and the quality of the compressed hologram is the same as that employing the classical VQ. Finally, we would like to compare the computation speed of our proposed method, based on identical CPU and GPU, with the existing method[13] that realizes classical VQ compression on GPU. The comparison is listed in Table 4. We observed that our method is about 8.5–9.5 times faster. The experimental results were worked out with the Intel Core i7 CPU 950 with 3-GB RAM in Windows XP. The PC is equipped with the graphics card of NVIDIA GeForce GTX 580. All the time measurements are the average of 20 runs in the experiment.

In conclusion, we propose a modified VQ compression method known as the SDVQ, and execute the algorithm with the fragment shader in GPU. Experimental evaluation reveals that a ROA hologram composing of 2 048 × 2 048 (pixel) can generally be compressed in around 20–22 ms using our proposed method. Compared with the existing state-of-the-art GPU utilization to realize the classical VQ, our proposed method is over 8 times faster. Further investigation can be conducted by applying our proposed method in other types of holographic data, e.g., the image hologram[14] and the fringe patterns of the virtual diffraction plane[15].

Fig. 4. Images of (a) "Lenna's eyes" and (b) "baboon's eyes"; (c) random image used to generate the hologram for training the codebook.

Fig. 5. (a) and (b) Numerical reconstructed images of the hologram representing the images in Figs. 4(a) and (b).

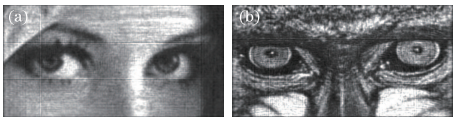

Fig. 6. (a) and (b) Numerical reconstructed images of the compressed hologram representing the images in Figs. 4(a) and (b).

Table 3. Settings of Hologram Generation Process

Parameter	Value
Hologram Size (pixel)	2 048 × 2 048
Pixel Size of the Hologram (μm^2)	7
Quantization of Pixel Intensity in the Hologram (bit)	16 (8 bit for the real part, and 8 bit for the imaginary part)
Wavelength of Light (nm)	650
Angle of the Off-axis Plane Reference Wave $B(y)$ (deg.)	1.2

Table 4. Comparison of Computation Speed between Our Proposed Method and Existing Approach Based on the Classical VQ Executing on the GPU

	Computation Time	
Source ROA Hologram	Classical VQ (ms)	Our Proposed SDVQ (ms)
Hologram Representing the Single Depth Image in Fig. 4(a)	185	20
Hologram Representing the Single Depth Image in Fig. 4(b)	185	22

References

1. T.-C. Poon, *Digital Holography and Three-Dimensional Display: Principles and Applications* (Springer, New York, 2006).
2. T.-C. Poon, J. Info. Disp. **3,** 12 (2002).
3. T.-C. Poon, Am. J. Phys. **76,** 738 (2007).
4. K. Sasaki, E. Tanji, and H. Yoshikawa, J. Inst. Tele. Eng. Japan **48,** 1238 (1994).
5. E. Darakis and J. J. Soraghan, IEEE Trans. Image Process **15,** 3804 (2006).
6. A. E. Shortt, T. J. Naughton, and B. Javidi, Opt. Express **14,** 5129 (2006).

7. A. E. Shortt, T. J. Naughton, and B. Javidi, IEEE Trans. Image Process **16,** 1548 (2007).
8. A. E. Shortt, T. J. Naughton, and B. Javidi, Proc. SPIE **5827,** 265 (2005).
9. P. W. Tsang, W. K. Cheung, and T.-C. Poon, Appl. Opt. **50,** H42 (2011).
10. P. W. Tsang, W. K. Cheung, and T.-C. Poon, in *Proceedings of Digital Holography and Three-Dimensional Imaging* DTuD1 (2011).
11. R. M. Gray, IEEE ASSP Magazine **1,** 4 (1984).
12. Y. Linde, A. Buzo, and R. M. Gray, IEEE Trans. Commun. **28,** 84 (1980).
13. H. Takizawa and H. Kobayashi, J. Supercomput. **36,** 219 (2006).
14. T. Yamaguchi and H. Yoshikawa, Chin. Opt. Lett. **9,** 120006 (2011).
15. P. W. M. Tsang and T.-C. Poon, Chin. Opt. Lett. **11,** 010902 (2013).