Object tracking mask-based NLUT on GPUs for real-time generation of holographic videos of three-dimensional scenes

Min-Woo Kwon[1], Seung-Cheol Kim[1], Sung-Eui Yoon[2], Yo-Sung Ho[3], Eun-Soo Kim[1,*]

[1]HoloDigilog Human Media Research Center (HoloDigilog), 3D Display Research Center (3DRC), Kwangwoon University, 447-1Wolge-Dong, Nowon-Gu, Seoul 139-701, South Korea
[2] Department of Computer Science, Korea Advanced Institute of Science and Technology (KAIST), 291 Daehak-ro (373-1 Guseong-dong), Yuseong-gu, Daejeon 305-701, South Korea
[3]The School of Information and Communications, Gwangju Institute of Science and Technology (GIST),123 Cheomdangwagi-ro, Buk-gu, Gwangju 500-712, South Korea
[*]eskim@kw.ac.kr

Abstract: A new object tracking mask-based novel-look-up-table (OTM-NLUT) method is proposed and implemented on graphics processing units (GPUs) for real-time generation of holographic videos of three-dimensional (3-D) scenes. Since the proposed method is designed to be matched with software and memory structures of the GPU, the number of compute-unified-device-architecture (CUDA) kernel function calls and the computer-generated hologram (CGH) buffer size of the proposed method have been significantly reduced. It therefore results in a great increase for the computational speed of the proposed method and enables real-time generation of CGH patterns of 3-D scenes. Experimental results show that the proposed method can generate 31.1 frames of Fresnel CGH patterns with 1,920×1,080 pixels per second, on average, for three test 3-D video scenarios with 12,666 object points on three GPU boards of NVIDIA GTX TITAN, and confirm the feasibility of the proposed method in the practical application of electro-holographic 3-D displays.

OCIS codes: (090.0090) Holography; (090.1760) Computer holography; (100.6890) Three dimensional image processing; (090.5694) Real-time holography.

References and links

1. C. J. Kuo and M. H. Tsai, Three-Dimensional Holographic Imaging (John Wiley & Sons, 2002).
2. T.-C Poon, Digital Holography and Three-Dimensional Display (Springer Verlag, 2007).
3. R. Oi, K. Yamamoto, and M. Okui, "Electronic generation of holograms by using depth maps of real scenes," Proc. SPIE **6912**, 69120M (2008).
4. M. Lucente, "Interactive computation of holograms using a look-up table," J. Electron. Imag. **2**, 28-34 (1993).
5. T. Yamaguchi and H. Yoshikawa, "Computer-generated image hologram," Chin. Opt. Lett. **9**, 120006 (2011).
6. K. Matsushima, and M. Takai, "Recurrence formulas for fast creation of synthetic three-dimensional holograms," Appl. Opt. **39**, 6587–6594 (2000).
7. Koki Murano, Tomoyoshi Shimobaba, Atsushi Sugiyama, Naoki Takada, Takashi Kakue, Minoru Oikawa, and Tomoyoshi Ito., "Fast computation of computer-generated hologram using Xeon Phi coprocessor", Comput. Phys. Commun. 185, 2742-2757 (2014)
8. J. Weng, T. Shimobaba, N. Okada, H. Nakayama, M. Oikawa, N. Masuda, and T. Ito, "Generation of real-time large computer generated hologram using wavefront recording method," Opt. Express **20**, 4018-4023 (2012). http://www.opticsinfobase.org/oe/abstract.cfm?URI=oe-20-4-4018
9. N. Okada, T. Shimobaba, Y. Ichihashi, R. Oi, K. Yamamoto, M. Oikawa, T. Kakue, N. Masuda, and T. Ito, "Band-limited double-step Fresnel diffraction and its application to computer-generated holograms," Opt. Express **21**, 9192-9197 (2013). http://www.opticsinfobase.org/oe/abstract.cfm?uri=oe-21-7-9192
10. T. Shimobaba, T. Kakue, and T. Ito, "Acceleration of color computer-generated hologram from three-dimensional scenes with texture and depth information," Proc. SPIE **9117**, 91170B (2014).

11. T. Senoh, K. Wakunami, Y. Ichihashi, H. Sasaki, R. Oi, and K. Yamamoto, “Multiview image and depth map coding for holographic TV system,” Opt. Eng. **53**, 112302 (2014).
12. Xuewu Xu, Yuechao Pan, Phyu Phyu Mar Yi Lwin, and Xinan Liang, "3D Holographic Display and Its Data Transmission Requirement," Information Photonics and Optical Communications (IPOC), 2011 International Conference on. IEEE, 1-4 (2011).
13. E. Darakis and T. J. Naughton, “Compression of digital hologram sequences using MPEG-4,” Proc. SPIE **7358**, 735811 (2009).
14. H. Yoshikawa and J. Tamai, “Holographic image compression by motion picture coding,” Proc. SPIE **2652**, 2–9 (1996).
15. T. Ito, N. Masuda, K. Yoshimura, A. Shiraki, T. Shimobaba, and T. Sugie, “Special-purpose computer HORN-5 for a real-time electroholography,” Opt. Express **13**, 1923-1932 (2005).
16. Yuechao Pan, Xuewu Xu, Sanjeev Solanki, Xinan Liang, Ridwan Bin Adrian Tanjung, Chiwei Tan, and Tow-Chong Chong, “Fast CGH computation using S-LUT on GPU”, Opt. Express **17**, 18543-18555 (2009). http://www.opticsinfobase.org/oe/abstract.cfm?uri=oe-17-21-18543
17. S.-C. Kim and E.-S. Kim, “Effective generation of digital holograms of 3D objects using a novel lookup table method,” Appl. Opt. **47**, D55-D62 (2008).
18. S.-C. Kim, J.-H. Yoon and E.-S. Kim, “Fast generation of 3-D video holograms by combined use of data compression and look-up table techniques,” Appl. Opt. **47**, 5986-5995 (2008).
19. S.-C. Kim, X.-B. Dong, M.-W. Kwon and E.-S. Kim, “Fast generation of video holograms of three-dimensional moving objects using a motion compensation-based novel look-up table,” Opt. Express **21**, 11568-11584 (2013). http://www.opticsinfobase.org/oe/abstract.cfm?uri=oe-22-14-16925
20. X.-B. Dong, S.-C Kim, and E.-S Kim, “MPEG-based novel look-up table for rapid generation of video holograms of fast-moving three-dimensional objects,” Opt. Express **22**, 8047-8067 (2014). http://www.opticsinfobase.org/oe/abstract.cfm?uri=oe-22-7-8047
21. D.-W. Kwon, S.-C. Kim, and E.-S. Kim, “Hardware implementation of N-LUT method using field programmable gate array technology,”Proc. SPIE **7957**, 79571C (2011).
22. M.-W. Kwon, S.-C. Kim, and E.-S. Kim, “Graphics processing unit-based implementation of a one-dimensional novel-look-up-table for real-time computation of Fresnel hologram patterns of three-dimensional objects”, Opt. Eng. **53**, 035103 (2014).
23. Raj Talluri, Karen Oehler, Thomas Bannon, Jonathan D. Courtney, Arnab Das, and Judy Liao, “A robust, scalable, object-based video compression technique for very low bit-rate coding.” IEEE Trans. Circuits Syst. Video Technol. **7**, 221-233 (1997).
24. Yiwei Wang, John F. Doherty, Robert E. Van Dyck, “Moving object tracking in video”, Proceedings of the 29th workshop of Applied Imagery Pattern Recognition, 95–101 (2000).
25. C. Kim and J.-N. Hwang, “Fast and automatic video object segmentation and tracking for content-based applications”, IEEE Trans. Circuits Syst. Video Technol. **12**, 122-129 (2002).
26. Hualu Wang and Shih-Fu Chang, “A Highly Efficient System for Automatic Face Region Detection in MPEG Video”, IEEE Trans. Circuits Syst. Video Technol. **7**, 615-628 (1997).
27. NVIDIA GeForce series GTX TITAN, http://www.nvidia.com/object/geforce_family.html.
28. S.-C. Kim, J.-M. Kim, and E.-S. Kim, “Effective memory reduction of the novel look-up table with one-dimensional sub-principle fringe patterns in computer-generated holograms,” Opt. Express 20, 12021–12034 (2012).
29. NVIDIA, CUDA C PROGRAMMING GUIDE (ver. 6.5), 2014

1. Introduction

Thus far, computer-generated hologram (CGH) has attracted much attention in the field of electro-holographic three-dimensional (3-D) display, since it can correctly record and reconstruct the light waves of 3-D scenes [1, 2]. The CGH-based electro-holographic display system, however, has a challenging issue of an enormous computational time involved in the generation of CGH patterns for 3-D video images.

A number of CGH algorithms for accelerating the computational speed has been proposed. Some of them include ray-tracing [1-3], look-up-table (LUT) [4], image hologram [5], recurrence relation [6, 7], wave-front recording plane (WRP) [8], double-step Fresnel diffraction (DSF) [9], and polygon methods [10]. In addition, various holographic video compression methods have also been proposed for electro-holographic television systems [11-14]. Moreover, several attempts to implement those CGH algorithms on field-programmable-gate-arrays (FPGAs) or graphics processing units (GPUs) toward their real-time applications have been made [7, 8, 15, 16].

Recently, a novel-look-up-table (NLUT) method was also proposed as an alternative approach for fast generation of CGH patterns [17]. Here, it must be noted that the NLUT,

contrary to other methods, generates the CGH patterns of 3-D scenes based on a two-step processing: pre- and main-processing [18-20]. In the pre-processing step, the number of object points to be calculated has been minimized by removing as much redundant object data between the consecutive 3-D video frames as possible by using motion estimation and compensation-based data compression algorithms. In the following main-processing step, the CGH patterns only for those compressed object data obtained from the pre-processing are computed by using the NLUT based on simple shifting and addition operations of the principal fringe patterns (PFPs), which were pre-calculated and stored [17]. This is the unique CGH calculation process carried out in the NLUT method by taking advantage of its shift-invariance property [19, 20].

Several types of NLUTs employing the pre-processing steps for eliminating the temporal redundancy between the consecutive 3-D video frames have been proposed until now. Some of them include temporal redundancy-based NLUT (TR-NLUT) [18], motion compensation-based NLUT (MC-NLUT) [19], and MPEG-based NLUT (MPEG-NLUT) [20] methods.

In practice, these NLUTs would be eventually implemented on FPGAs or GPUs for their practical application to real-time CGH generation of 3-D scenes [21, 22]. For efficient implementation of NLUTs on GPU boards, they must be compatible with the software and memory structures of the GPU boards. This implies that the NLUT algorithms must be well tailored to be compatible with the specifications of the GPU boards.

Recently, MPEG-based video communication and television systems have employed a concept of object tracking mask (OTM) for obtaining a high video coding-efficiency [23-25]. Here, the primary objective of OTM is to support the coding of video sequences, which are pre-segmented based on video contents, and to allow separate and flexible reconstruction and manipulation of contents at the decoder in MPEG [24, 25]. In video calls, CCTVs, NEWS, and most broadcasting, cameras to capture videos are fixed and contain only a small number of objects and people with motion in a large stationary background. Under these circumstances, a high coding-efficiency can be achieved by employing the OTM-based video compression method [23].

This OTM method can be directly applied to the NLUT-based holographic 3-D video communication system, since the NLUT has a unique property of shift-invariance unlike other CGH generation algorithms. At the same time, the OTM method has the simplest algorithmic and memory structure to be practically implemented on the commercial GPU boards among those NLUT methods mentioned above. Even though the conventional TR-NLUT, MC-NLUT and MPEG-NLUT methods show good performances in the compression of 3-D video data, these methods have several drawbacks, when they are evaluated to be implemented on GPU boards.

In the TR-NLUT method, only the difference images between two consecutive 3-D video frames are involved in CGH calculation. However, for CGH calculation of the current frame, two-step calculation processes are required, increasing its algorithmic complexity. That is, in the CGH calculation process of the current frame, the CGH pattern of the previous frame must be saved on the temporary buffer. If the size of the temporary buffer becomes larger than that of the on-chip memory, the global memory of the GPU is used to store this buffer instead. As a result, the number of accesses to the global memory increases, causing a deterioration of the computational performance of the GPU.

In the MC-NLUT and MPEG-NLUT methods, object-based and block-based motion estimation and compensation processes are additionally performed, respectively, before the difference images between two consecutive video frames are extracted using the TR-NLUT. The algorithmic complexities of the MC-NLUT and MPEG-NLUT methods therefore increase much more than that of the TR-NLUT.

In this paper, a new object tracking mask-based NLUT (OTM-NLUT) method, which has simple algorithmic and memory structures, is proposed and implemented on GPU boards for real-time CGH computation of 3-D video frames. In the proposed method, the current video frame is divided into fixed and moving object points by using a simple OTM method.

The CGH pattern for the fixed object points of the previous frame is then reused to generate the CGH pattern for those of the current video frame, while the CGH pattern for the moving object points are calculated using the NLUT. Therefore, the total number of calculated object points of the current video frame can be significantly reduced by the reuse of the CGH pattern of the fixed object points in the previous frame. Furthermore, contrary to the two-step calculation processes of the conventional NLUT methods, the proposed method has only a single calculation process for the moving object points.

To confirm the feasibility of the proposed method, experiments with three kinds of test 3-D video scenarios are carried out on three GPU boards, and the results are compared to those of the conventional NLUT, TR-NLUT, MC-NLUT, and MPEG-NLUT methods in terms of the numbers of calculated object points and CUDA kernel function calls, the CGH buffer size, the total calculation time, and the frames per second (FPS).

2. CGH generation of a 3-D object using the NLUT

In the NLUT method, a three dimensional object is approximated as a set of discretely sliced object planes that have different depth. Only the fringe patterns for the center-located object points on each object plane, called principal fringe patterns (PFPs), are pre-calculated, and stored. This method, in turn, achieves a significant increase of the computational speed as well as a massive reduction of the memory capacity [17]. Figure 1 shows geometry for generation of the Fresnel CGH pattern of a 3-D object.

The location coordinate of the p^{th} object point is specified by (x_p, y_p, z_p), and each object point is assumed to have an associated real-valued magnitude and phase, a_p *and* φ_p. Additionally, we assume that the CGH pattern is positioned on the plane of $z = 0$.

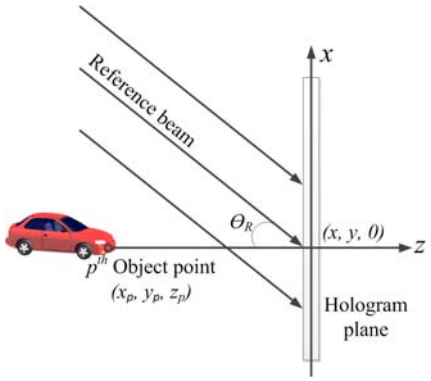

Fig. 1 Geometry for generating the Fresnel CGH hologram pattern of a 3-D object

As mentioned above, a 3-D object can be treated as a set of 2-D object planes discretely sliced along the depth direction of z, and each object plane is approximated as a collection of self-luminous object points of light. Thus, the unity-magnitude PFP for the object point (0, 0, z_p) positioned at the center of an object plane with a depth of z_p, $T(x, y; z_p)$ can be defined by the following equation [2]:

$$T(x,y;z_p) \equiv \frac{1}{r_p}\cos\left[kr_p + kx\sin\theta_R + \phi_p\right] \tag{1}$$

Here, the wave number k is defined as $k = 2\pi/\lambda$, where λ and θ_R represent the free-space wavelength of the light and the incident angle of the reference beam, respectively. The oblique distance r_p between the p^{th} object point of (x_p, y_p, z_p) and the point on the hologram plane of (x, y, 0) is given by:

$$r_p = \sqrt{(x - x_p)^2 + (y - y_p)^2 + z_p^2}. \tag{2}$$

The fringe patterns of other object points on the same object plane can be, then, simply obtained by shifting this pre-calculated PFP [17]. Therefore, fringe patterns for all object points located on the same object plane can be generated by adding these shifted PFP versions. The final CGH pattern of a 3-D object can be obtained by summing all shifted PFP versions together, which are generated on each object plane.

Therefore, the CGH pattern for a 3-D object $I(x, y)$ in the NLUT method can be expressed in terms of shifted versions of pre-calculated PFPs of Eq. (1) as shown in Eq. (3).

$$I(x,y)=\sum_{p=1}^{N} a_p T\left(x-x_p, y-y_p; z_p\right), \tag{3}$$

where N is the number of object points. Eq. (3) shows that the CGH pattern of a 3-D object can be obtained simply by shifting the PFPs as a function of the displaced values of the object points from the center object points on each object plane and by then getting their sum.

3. Proposed OTM-NLUT method

In this section we explain main components of our method and its compatibility with GPUs.

3.1. Object tracking mask for data compression of 3-D video frames

The MPEG-4 standard suggests the use of an object tracking mask (OTM) in video compression [24], and now video object segmentation and tracking becomes an important issue for successful usage of MPEG-4, MPEG-7, and MPEG-21 algorithms. Segmentation by object-tracking allows much useful functionality for multimedia applications such as easy access to bit streams of individual objects, object-based manipulation of bit streams, various interactions with objects, and the reuse of content information by scene composition [25].

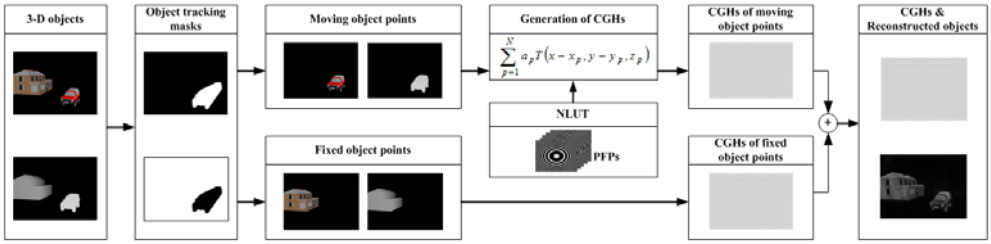

Fig. 2 Overall block-diagram of the proposed method for CGH generation of 3-D video images

Here in this paper, this OTM-based video compression technique is employed for reduction of the number of object points to be calculated in CGH generation with the NLUT method. Figure 2 shows an overall block-diagram of the proposed OTM-NLUT method to generate the CGH patterns for 3-D video images. The proposed method is composed of three steps. First, intensity and depth data of the current frame of 3-D video images are extracted and divided into fixed and moving object points using the OTM. Second, the CGH patterns for the moving object points are calculated with the NLUT. Third, the CGH pattern of the current frame is generated by adding the CGH pattern generated for the fixed object points of the previous frame to that for the moving object points of the current frame, as both the current and previous frames share the same fixed object points. The CGH pattern of the current frame is then transmitted to the CGH video output.

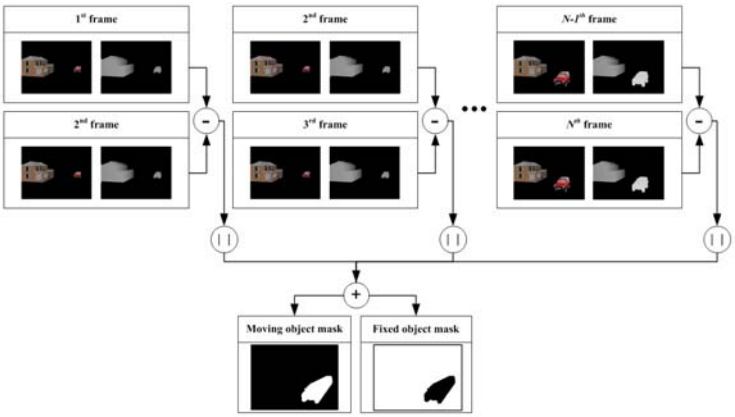

Fig. 3 Flowchart for generation of the OTM

Figure 3 shows the flowchart for generation of the OTMs over N frames. It consists of four steps, in which '| |' denotes the absolute value. First, absolute values of the differences between the video frames are accumulated in order to find the moving object points. Second, when the accumulated value of the object point is '0', which means this object point has no difference, it becomes the fixed point. Thus, the mask values of the fixed and moving object points are set to be '1' and '0', respectively. Otherwise, the mask values of the moving and fixed object points are set to be '1' and '0', respectively. Finally, two masks for each of the moving and fixed object points, which are called $MASK_{\text{moving}}$ and $MASK_{\text{fixed}}$, are obtained. Here, by using these $MASK_{\text{moving}}$ and $MASK_{\text{fixed}}$, corresponding moving and fixed object points of the current video frame can be extracted.

In most MPEG-based video communication and television systems, the number of frames considered for computing a one set of OTMs is usually set to be from 12 to 15, when the frames per second (FPS) are given by 25 to 30 [26].

3.2. Compatibility of the NLUT with the GPU board

Generally, the GPU board can calculate the CGH patterns of 3-D scenes in parallel with a high computational speed by using the compute-unified-device-architecture (CUDA), where a CUDA kernel function is executed by the massive number of threads of the GPU board. That is, video data to be processed by these threads are transferred from the host memory, i.e., main memory, to the device global memory, and the threads then access their portion of data from the global memory. Here, the global memory, which is typically implemented with dynamic random access memories (DRAMs), tends to have long access latencies (e.g., hundreds of clock cycles) and finite access bandwidth. Moreover, even though many threads are available for parallel execution in the GPU board, traffic congestion in the global memory may degrade the calculation performance of the GPU significantly. Thus, degrees of algorithmic complexity and global memory utilization is one of the most important factors in implementation of an efficient parallel program [29].

Here, the conventional TR-NLUT, MC-NLUT, MPEG-NLUT, and proposed OTM-NLUT methods are comparatively analyzed in terms of degrees of algorithmic complexity and global memory utilization. Figure 4(a) shows the software structure of the TR-NLUT, MC-NLUT, and MPEG-NLUT methods. Those methods mainly consist of three steps. First, the preprocessing mechanisms of the TR-NLUT, MC-NLUT, and MPEG-NLUT methods, which can reduce the number of calculated object points for CGH generation [18-20], are performed in the 'A' part of Fig. 4(a). After extracting intensity and depth data of the 3-D object of the current frame, the difference image between two object images of the previous and the current frames is calculated in the TR-NLUT. On the other hand, in the MC-NLUT, motion vectors of the 3-D object between the previous and current frames are extracted, and the motion-compensated object image is obtained with these motion vectors. Then, the difference image

between the motion-compensated object image of the previous frame and the object image of the current frame is calculated.

Moreover, in the MPEG-NLUT, motion vectors of the segmented image blocks between the previous and current frames are extracted, and the motion-compensated image blocks are obtained with these motion vectors. The difference image blocks between the motion-compensated image blocks of the previous frame and the image blocks of the current frame are then calculated.

Second, the CGH patterns for the changed parts in both previous and current frames are calculated by using the NLUT in each of the TR-NLUT, MC-NLUT, and MPEG-NLUT methods. The CGH pattern of the current frame is then generated by subtracting the CGH patterns for the disappeared object points from the CGH pattern of the previous frames and by adding the CGH pattern for the newly appeared object points to the CGH pattern of the previous frame. The lastly calculated CGH pattern is transmitted to the CGH video output as well as stored in the previous frame buffer of the CGH [18-20].

Figure 4(b) shows the software structure of the proposed OTM-NLUT method. From Fig. 4(b), the number of base CUDA kernel function calls and the total CGH buffer size of the proposed method can be obtained. Three-step processing for CGH generation in the proposed method has already been explained in Chapter 3.1 with Fig. 2.

In this paper, the NLUT method is implemented as the base CUDA kernel function. Since the CGH can be generated with PFPs on each depth in the NLUT method [17,22], the base CUDA kernel function may be called on each depth. Actually, the computational performance of the GPU depends on the number of base CUDA kernel function calls, and the conventional NLUT methods have different numbers of base CUDA kernel function calls. In addition, as seen in the ‘B’ part of Fig. 4(a), the CGH pattern of the current frame is generated by compensating the CGH pattern of the previous frame, and thus the CGH pattern of the previous frame must be saved in the temporary buffer. Since this temporary buffer is saved in the global memory of the GPU board. As the total CGH buffer size increases, the corresponding degree of global memory utilization of the GPU board also grows. It results in a decrease of the GPU’s computational performance.

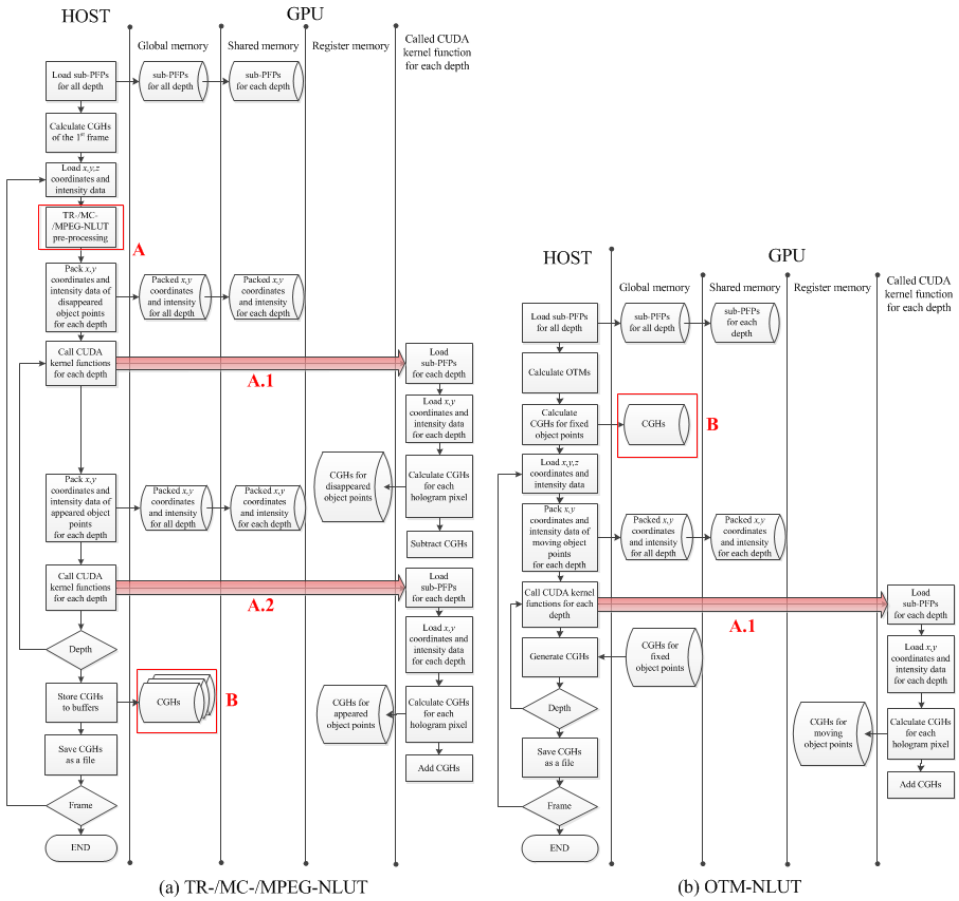

Fig. 4 These flow charts show software structures of (a) TR-NLUT, MC-NLUT, and MPEG-NLUT, and (b) OTM-NLUT methods

Table 1 shows the comparison results on compatibilities of the conventional and proposed NLUT methods with the GPU board. As shown in 'A.1' and 'A.2' parts of Fig. 4(a), the number of base CUDA kernel function calls for each frame of the conventional TR-NLUT method is given by $N_{DP\text{-}AOP}+N_{DP\text{-}DOP}$, in which $N_{DP\text{-}AOP}$ and $N_{DP\text{-}DOP}$ represent the numbers of depth planes with appeared object points and disappeared object points, respectively.

In the MC-NLUT and MPEG-NLUT methods [19, 20], object-based and block-based motion compensation processes, respectively are carried out for CGH generation, in which each frame consists of several objects and blocks. Therefore, the final CGH pattern of each frame of those methods can be generated by summing hologram patterns of each object and block. Because these hologram patterns of each object and block are calculated by using PFPs, the numbers of base CUDA kernel function calls for each frame is given by summation of those $N_{DP\text{-}AOP}$ and $N_{DP\text{-}DOP}$ for all moving objects in the MC-NLUT method and for all moving blocks in the MPEG-NLUT method, respectively as shown in Table 1. Thus, as the numbers of objects and blocks increase, the corresponding numbers of CUDA kernel function increase as well.

Table 1 Compatibilities of the conventional and proposed NLUT methods with the GPU board

	TR-NLUT	MC-NLUT	MPEG-NLUT	Proposed method

Number of CUDA kernel function calls	$N_{DP-AOP} + N_{DP-DOP}$	$\sum_{m=1}^{N_{object}} (N_{DP_m-AOP} + N_{DP_m-DOP})$	$\sum_{m=1}^{N_{block}} (N_{DP_m-AOP} + N_{DP_m-DOP})$	N_{DP-MOP}
CGH buffer size	*CGH size*	*CGH size* × (N_{object}+1)	*CGH size* × N_{block}	*CGH size*

On the other hand, as seen in 'A.1' of Fig. 4(b), the proposed method has only a one-step calculation process contrary to the conventional methods, which have two-step calculation processes. That is, in the proposed method, the number of base CUDA kernel function calls of each frame is given by $N_{DP\text{-}MOP}$, in which $N_{DP\text{-}MOP}$ represents the number of depth planes with moving object points, which is similar to $N_{DP\text{-}AOP}$. Because the proposed method has no compensation process, only the appeared object points representing the moving object points are calculated to generate the CGH pattern for each frame. Thus, the total number of base CUDA kernel function calls can be highly reduced, when it is compared to those of the conventional TR-NLUT, MC-NLUT, and MPEG-NLUT methods. The reduced number of base CUDA kernel function calls in turn can result in a great reduction of the CGH computation time of the proposed method on the GPU boards.

Table 1 also shows the total CGH buffer sizes of the conventional TR-NLUT, MC-NLUT, MPEG-NLUT, and proposed methods. As seen in the 'B' part of Fig. 4(a), in all conventional methods, the CGH patterns of the current frame are generated by compensating the CGH patterns of the previous frames, and thus the CGH patterns of the previous frames must be stored in the temporary buffers. In the TR-NLUT method, just the CGH pattern of the previous frame is saved in the temporary buffer, so that the total CGH buffer size of the TR-NLUT is given by the '*CGH size*' itself. On the other hand, in the MC-NLUT and MPEG-NLUT methods, CGH patterns for each object and block of the current frame are generated by compensating the CGH patterns for each object and block of the previous frame [19, 20], therefore the CGH patterns for all objects and blocks of the previous frame have to be maintained in the temporary buffer. Consequently, the total CGH buffer sizes of the MC-NLUT and MPEG-NLUT methods increase up to '*CGH size* × (N_{object}+1)' and '*CGH size* × N_{block}', respectively. Here, the CGHs for the fixed objects are saved together in one buffer, thus one more CGH buffer for those fixed objects is additionally needed [19]. Since these temporary buffers are saved in the global memory of the GPU board, degrees of global memory utilization of the GPU board is increased much more than that of the TR-NLUT, which can cause the computational performance of the GPU to be deteriorated.

However, in the proposed method, as seen in the 'B' part of Fig. 4(b), the CGH pattern of the current frame is generated simply by adding the CGH pattern for the fixed object points of the previous frame to that for the moving object points of the current frame. Therefore, only the CGH pattern for the fixed object points of the previous frame is saved in the temporary buffer. That is, the total size of the CGH buffer of the proposed method equals to that of the '*CGH size*' itself. The proposed method, therefore, turns out to have the least CGH buffer size. It means that the computational performance of the proposed method can be enhanced.

3.3. Numbers of calculated object points for CGH generation

In the conventional TR-NLUT, MC-NLUT, and MPEG-NLUT methods, the numbers of calculated object points can be greatly reduced, since all those methods employ pre-processing steps for removal of redundant object data between the consecutive video frames [18-20]. Equation (4) shows the total numbers of calculated object points in the conventional TR-NLUT, MC-NLUT, and MPEG-NLUT methods.

$$N_{TCOP} = N_{DOP} + N_{AOP} - 2 \times N_{ROP} \tag{4}$$

Here, N_{TCOP} indicates the total number of calculated object points required for CGH generation for each frame. N_{DOP} and N_{AOP} denote the number of disappeared object points in the previous frame and the number of appeared object points in the current frame, respectively.

Also, N_{ROP} represents the number of redundant object points between the disappeared and appeared object points. In the TR-NLUT method, the value of N_{ROP} becomes small, since only the differences between the consecutive video frames are extracted. However, in the MC-NLUT and MPEG-NLUT methods, it can be increased by employed motion estimation and compensation processes, which results in a reduction of the total number of calculated object points. On the other hand, in the proposed OTM-NLUT method, the CGH patterns of only the moving object points are calculated for each frame. Thus, the total number of calculated object points of the proposed method can be given by Eq. (5).

$$N_{TCOP} = N_{MOP,} \tag{5}$$

where N_{MOP} denotes the number of moving object points obtained with the $MASK_{\text{moving}}$.

3.4. OTM-NLUT method on GPUs

Figure 5 shows a conceptual diagram for implementing the proposed OTM-NLUT method on three GPU boards. Here, the CGH pattern with a resolution of 1920 × 1080 pixels is calculated by three GPU boards (Model: NVIDIA GTX TITAN) [27]. As shown in Fig. 5, the same input video image data is simultaneously fed to the three GPU boards, while the CGH pattern to be calculated is divided into two parts, where the resolution of each divided CGH pattern is given by 1920 × 540 pixels. Each GPU board then calculates its own CGH pattern in parallel.

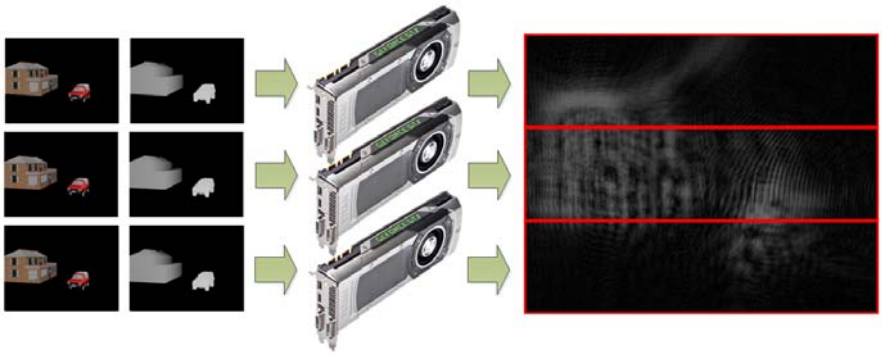

Fig. 5 Conceptual diagram for calculation of the CGH patterns in parallel on three GPU boards

Figure 6 shows a block-diagram of the software structure of the proposed OTM-NLUT method implemented on three GPU boards. The CPU code is written in C-language and OpenMP, which only controls the system flow. Main CGH computation is carried out on three NVIDIA GTX TITAN GPU boards. The CPU core-1 controls the main system flow, which loads the PFPs for all depths, calculates the OTMs, loads the input object data, saves the CGH pattern as the file, and controls the GPU board-1. Each of the CPU core-2 and -3 controls only its corresponding GPU board-2 and -3, respectively. Since the CPU core-1, -2 and -3 work in parallel with the OpenMP code, these can work in parallel. Thus, GPU board-1, -2 and -3 simultaneously calculate their allocated parts of the CGH pattern as shown as Fig. 5.

In the GPU board-1, -2 and -3, in order to maintain low-memory usage of megabytes, one-dimensional (1-D) NLUT is used for generating the CGH patterns [28]. For fast loading and accessing the sub-PFPs data of the 1-D NLUT and the 3-D data of the input object on the GPU boards, three types of optimization techniques are also employed. These include the packing technique of the input 3-D object data for efficient storing in the on-chip shared memory, and the managing techniques of the on-chip shared memory for fast computation of the CGH, and the on-chip registers for quick storing of the calculated hologram data [22].

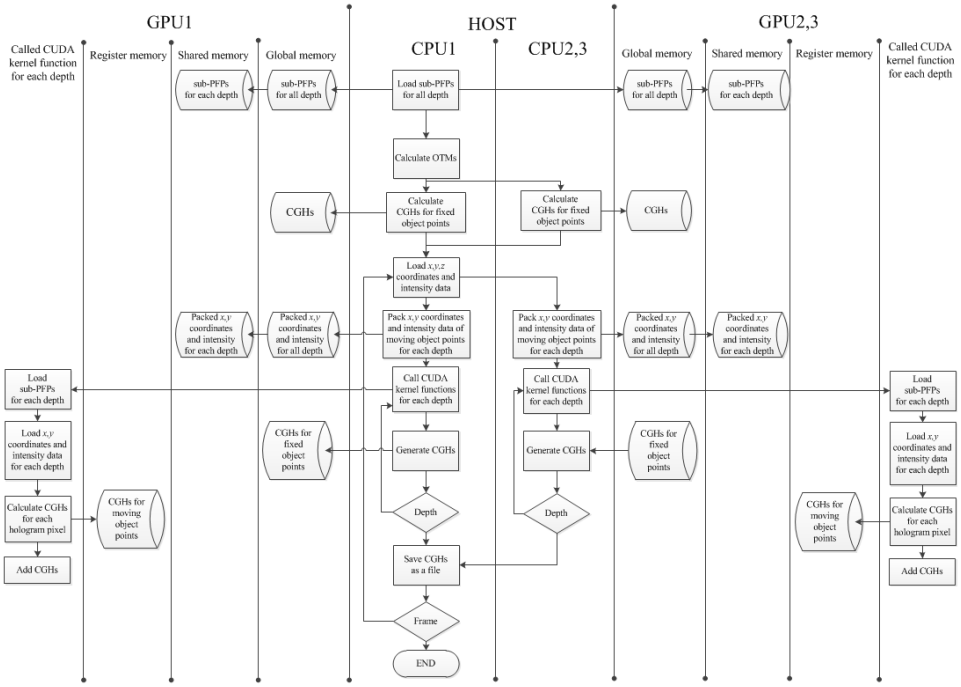

Fig. 6 Software structure of the proposed OTM-NLUT on three GPU boards

4. Experiments and the results

4.1. Test 3-D video scenarios

To confirm the benefits of the proposed method and to comparatively analyze the computational performance of the proposed method with those of the conventional methods, experiments are performed. As test 3-D videos, three kinds of scenarios, which commonly occur in CCTVs, TV dramas, and video calls, are computationally generated. Here, each 3-D video scenario consists of 50 frames, each of which has a resolution of 320×240 pixels.

Figure 7 shows intensity and depth images of the 1st, 30th and 60th frames for each of the scenarios. As seen in Fig. 7, ‘Scenario I’ shows the sequential front views of a 3-D video of ‘A car passing by the house’, which looks like the scene captured by the CCTV camera. This video is mostly composed of a small part of a moving car image and a large part of a fixed house. In addition, ‘Scenario II’ and ‘Scenario III’ also show the sequential front views of 3-D videos of ‘A man walking nearby the bed’ and ‘A man holding out his hand’, respectively. Here, these video images look similar with scenes captured from the TV drama and video call, which are also composed of small parts of a moving man, and large parts of fixed points just like the case of ‘Scenario I’.

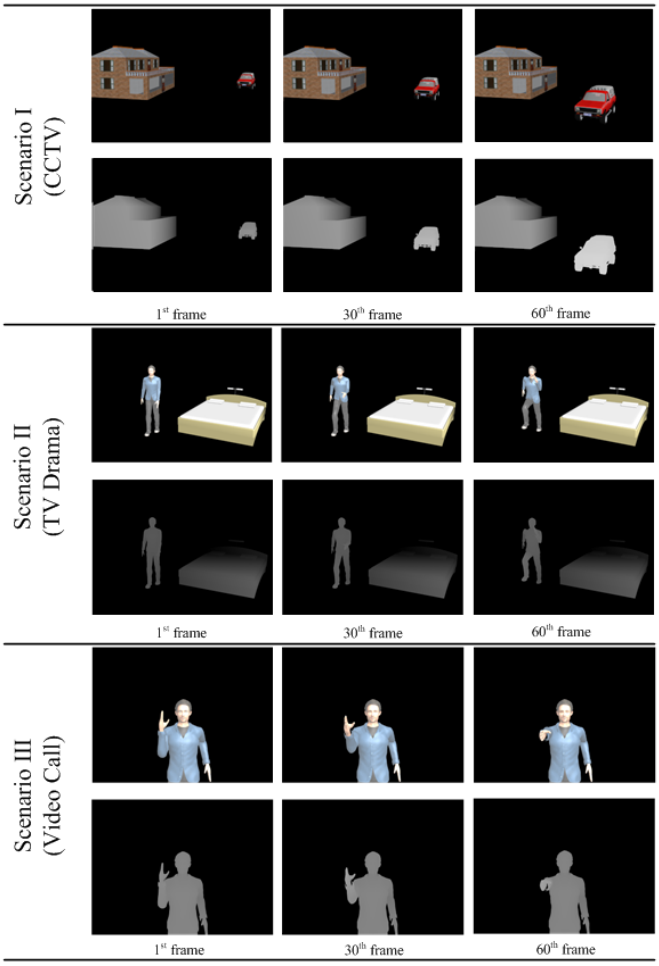

Fig. 7 Intensity and depth images of the 1st, 30th, and 60th frames for each of the 'Scenario I, II and III'

4.2. Generation of CGH patterns with the proposed method

In the experiments, CGH patterns with a resolution of 1,920×1,080 pixels are generated with intensity and depth data of the test 3-D video scenarios. Each pixel size of the CGH pattern is given by $10\mu m \times 10\mu m$. Moreover, the viewing distance is set to be $500mm$, and the horizontal and vertical discretization steps are set to be $150\mu m$, respectively, which means that the amount of the pixel shift is given by 15 pixels in the OTM-NLUT method [17]. In this configuration, to fully display the fringe patterns for the first and end image points located on each image plane, the PFP must be shifted by 2,250 (150×15) pixels horizontally and vertically. Thus, the total resolution of the PFP should become 2,750 (500+2,250) × 2,750 (500+2,250) pixels [17]. The PFPs for the center image points located on each plane can be calculated. Here, the computer system used in the experiment is composed of an Intel i7 3770 CPU with an 8GB RAM and three NVIDIA GTX TITAN GPU, and it works on the CentOS 6.3 Linux platform.

Figure 8 shows intensity and depth images of the 60th frame, two masks of $MASK_{\text{fixed}}$ and $MASK_{\text{moving}}$, and extracted fixed and moving objects with corresponding masks for each case of the 'Scenario I, II and III'. With these image data, the CGH patterns are calculated

using the proposed OTM-NLUT method. Here, the mask sizes are the same with those of input images, which are given by 320×240 pixels.

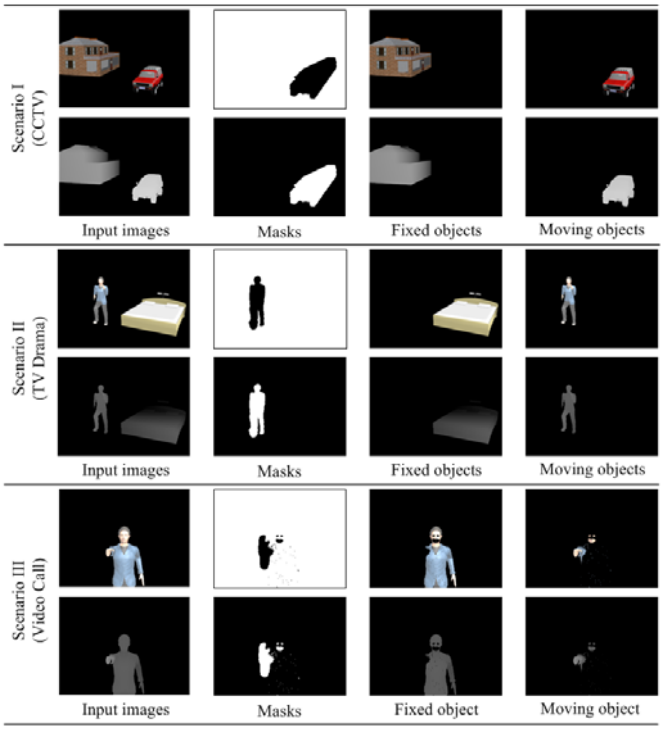

Fig. 8 Intensity and depth images of the 60th frame, masks and extracted fixed and moving objects with corresponding masks for each case of the 'Scenario I, II and III'

Figure 9 shows the calculated CGH patterns with the proposed method for fixed, moving and total object images of the 60th frame for each case of the 'Scenario I, II and III'. Here, in the 60th frame, the CGH patterns for only the moving objects are calculated, since the CGH patterns for the fixed objects over several frames have been calculated at a time.

For calculation of the CGH patterns for the moving object points, as seen in Fig. 6, the CPU core-1 controls the main system flow, which loads the 1-D NLUT, calculates the OTMs, loads input object data, obtains the moving object points using the $MASK_{\text{moving}}$, packs those moving object points for each depth, and controls the GPU board-1. CPU core-2 and -3 control only the GPU board-2 and -3, respectively. Then, CPU core-1, -2 and -3 transfer the 1-D NLUT and packed moving object points to each of the GPU board-1, -2 and -3. As seen in Fig. 5, the same moving object points and 1-D NLUT are simultaneously fed to all of the GPU board-1, -2 and -3, while the CGH pattern to be calculated is divided into three parts. Then, each GPU board calculated its own CGH pattern in parallel. The CGH pattern for the moving object points can be obtained just by shifting the PFPs depending on the displaced values of the object points from the center object points on each object plane and by adding them all up [17].

For this calculation, the CUDA kernel functions are called for each depth. If the moving object points are located in many depth layers, the number of CUDA kernel function calls is increased. After the hologram patterns for the moving object points are calculated in each of the GPU board-1, -2 and -3, the total CGH pattern of the 60th frame can be finally calculated simply by adding these hologram patterns for the moving object points and that for the fixed object points already calculated.

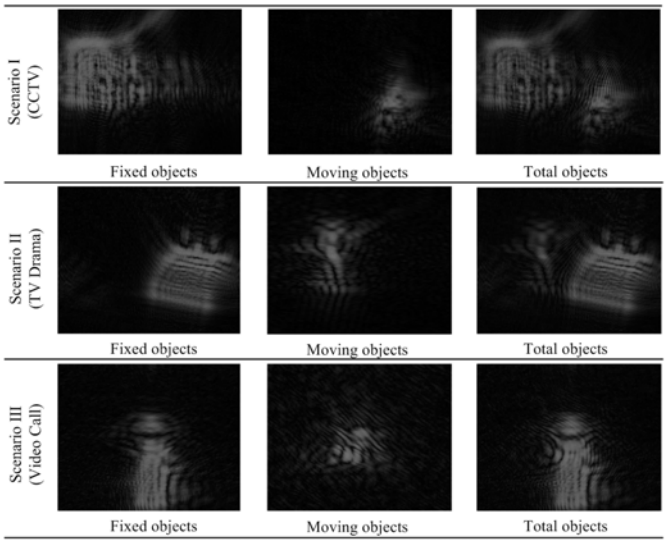

Fig. 9 Calculated CGH patterns of the 60th frames with the proposed method for the fixed, moving and total objects images for each case of the 'Scenario I, II and III'

Figure 10 shows the results of the reconstructed object images of the 1st, 30th and 60th frames from the CGH patterns calculated with the proposed method for three kinds of 3-D video scenarios. In the experiments, off-axis reference beams were used for reconstruction of object images without having direct and conjugate beams. As seen in Fig. 10, all object images have been successfully reconstructed and are visually correct.

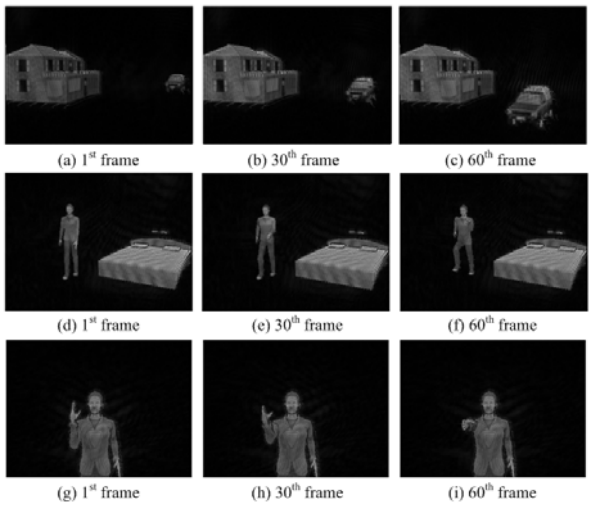

Fig. 10 Reconstructed 3-D video images of the 1st, 30th and 60th: (a)-(c) 'Scenario I', (d)-(f) 'Scenario II', (g)-(i) 'Scenario III'

4.3. Performance analysis of the proposed method

Figure 11 shows the comparison results on the performances of the conventional and proposed methods in terms of the numbers of calculated object points, CUDA kernel function calls, and the calculation time for each case of the 'Scenario I, II and III'. Table 2 also shows the average numbers of calculated object points, CUDA kernel function calls, and the average calculation time for one frame as well as the CGH buffer sizes and the numbers of video

frames per second (FPS) of the conventional and proposed methods for each case of the 'Scenario I, II and III'.

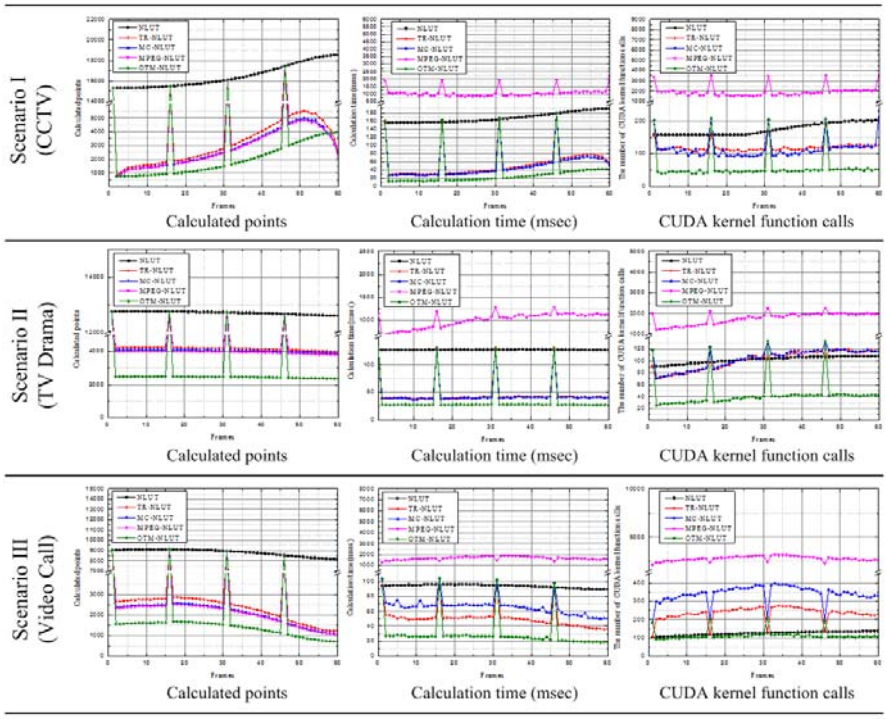

Fig. 11 Comparison results on the performances of the conventional and proposed methods for each case of the 'Scenario I, II and III'

Table 2 Average performance of the conventional and proposed methods for each case of the 'Scenario I, II and III'

		Number of calculated object points	Calculation time (*msec*)	Number of CUDA kernel function calls	CGH buffer size (MB)	Frames per second (FPS)
Scenario I	NLUT	16,470	168	174	0	6.0
	TR-NLUT	3,968	55	120	2	18.0
	MC-NLUT	3,679	52	116	4	19.1
	MPEG-NLUT	3,605	1,081	1,965	593	0.8
	OTM-NLUT	2,888	34	58	2	**29.8**
Scenario II	NLUT	12,705	127	102	0	7.9
	TR-NLUT	4,741	49	102	2	20.3
	MC-NLUT	4,624	48	104	4	20.8
	MPEG-NLUT	4,549	1,010	1,771	593	0.9
	OTM-NLUT	3,138	34	44	2	**29.1**
Scenario III	NLUT	8,824	94	125	0	10.7
	TR-NLUT	2,739	52	236	2	19.2
	MC-NLUT	2,524	66	349	10	15.2
	MPEG-NLUT	2,474	1,684	2,807	593	0.6
	OTM-NLUT	1,856	29	114	2	**34.3**

4.3.1. Analysis on the numbers of calculated object points

Figure 11 and Table 2 shows that the average numbers of calculated object points of the conventional NLUT, TR-NLUT, MC-NLUT, MPEG-NLUT, and proposed OTM-NLUT

methods. These numbers have been found to be 16,470, 3,968, 3,679, 3,605, 2,888 and 12,705, 4,741, 4,624, 4,549, 3,138 and 8,824, 2,739, 2,524, 2,474, 1,856, respectively, for each case of the 'Scenario I, II and III'. In other words, the average numbers of calculated object points of the proposed method have been reduced by 82.47%, 27.22%, 21.50%, 19.89% for 'Scenario I', and 75.30%, 33.81%, 32.14%, 31.02% for 'Scenario II' and 78.97%, 32.24%, 26.47%, 24.98% for 'Scenario III', respectively, when they are compared to each of the conventional NLUT, TR-NLUT, MC-NLUT and MPEG-NLUT methods.

As seen in Fig. 11 and Table 2, the proposed OTM-NLUT has the lowest numbers of calculated object points for all cases of 'Scenario I, II and III' among the tested methods. Interestingly, the conventional TR-NLUT, MC-NLUT, and MPEG-NLUT methods also show much reduced numbers of calculated object points than those of the original NLUT method for three cases. In fact, the great reductions in the numbers of calculated object points are resulted from their employed pre-processing steps for removing redundant object data between the consecutive video frames, which is contrary to the original NLUT method having no data reduction process [18-20].

In particular, the reason why the proposed method has the least numbers of calculated object points is that only the moving object points in the current frame are involved in CGH calculation unlike the conventional TR-NLUT, MC-NLUT, and MPEG-NLUT methods, where not only the disappeared object points in the previous frame, but also the newly appeared object points in the current frame are involved in CGH calculation together. That is, the total numbers of calculated object points of those conventional methods become almost doubled when they are compared to that of the proposed method.

As seen in Fig. 11, for the 1st frames, all object points get involved in CGH calculation in all methods, but from the 2nd frames, the numbers of calculated object points are sharply reduced with their data reduction processes between the two consecutive video frames. In fact, Table 2 shows the average numbers of calculated object points, so that the differences in the numbers of calculated object points on an arbitrary frame for each case of the conventional and proposed methods cannot be exactly seen. We also provide Table 3 for detailed analysis, which shows the reduced numbers of calculated object points on a frame for each case of the conventional and proposed methods, the total numbers of calculated object points of the current frame, the numbers of disappeared object points in the previous frame, the numbers of newly appeared object points in the current frame, and the numbers of redundant object points between the previous and current moving objects for the 30th frame.

Table 3 Results on the total numbers of calculated object points of the 30th for each case of the conventional and proposed NLUT methods

		Total number of calculated object points	Number of redundant object points	Number of disappeared object points	Number of appeared object points
Scenario I	NLUT	16,055	0	0	16,055
	TR-NLUT	2,806	97	1,498	1,502
	MC-NLUT	2,624	188	1,498	1,502
	MPEG-NLUT	2,474	263	1,498	1,502
	OTM-NLUT	1,502	0	0	1,502
Scenario II	NLUT	12,733	0	0	12,733
	TR-NLUT	4,274	371	2,502	2,514
	MC-NLUT	4,146	435	2,502	2,514
	MPEG-NLUT	4,062	477	2,502	2,514
	OTM-NLUT	2,514	0	0	2,514
Scenario III	NLUT	8,972	0	0	8,972
	TR-NLUT	2,604	235	1,544	1,530
	MC-NLUT	2,342	366	1,544	1,530
	MPEG-NLUT	2,294	390	1,544	1,530
	OTM-NLUT	1,530	0	0	1,530

As seen in Table 3, the total numbers of calculated object points of the 30th frame have been calculated to be 16,055, 2,806, 2,624, 2,474, 1,502 for 'Scenario I' and 12,733, 4,274, 4,146, 4,062, 2,514 for 'Scenario II' and 8,972, 2,604, 2,342, 2,294, 1,530 for 'Scenario III', respectively, for each case of the conventional NLUT, TR-NLUT, MC-NLUT, MPEG-NLUT and proposed methods. That is, the original NLUT method calculated all object points of the 30th frames for CGH generation, whereas the numbers of calculated object points have been massively reduced in other methods.

According to Eq. (4), the total numbers of calculated object points have been estimated to be 2,806, 2,624 and 2,474 for 'Scenario I' and 4,274, 4,146, and 4,062 for 'Scenario II' and 2,604, 2,342, and 2,294 for 'Scenario III', respectively, for each case of the TR-NLUT, MC-NLUT, and MPEG-NLUT methods. On the other hand, in the proposed method, the numbers of appeared object points in the 30th frames have been calculated to be 1,502, 2,514 and 1,530, respectively, for each case of the 'Scenario I, II and III'. These values also represent the total numbers of calculated object points of the 30th frame of the proposed method according to Eq. (5). Thus, the total number of calculated object points of the proposed method has been reduced down to 65.76% of those of the conventional methods on average. It is because the conventional TR-NLUT, MC-NLUT, and MPEG-NLUT methods require two-step calculation processes of subtraction and adding of CGH patterns in both 29th and 30th frames, whereas only a one-step calculation process in the 30th frame is needed in the proposed method as explained in Chapter 3.3.

4.3.2. Analysis on the CGH buffer sizes in the global memory of a GPU board

Table 1 shows that the TR-NLUT and proposed methods have the least CGH buffer sizes of 2*MB* (1,920 × 1,080) for all cases of 'Scenario I, II and III'. However, those sizes of the MC-NLUT method are increased to 4*MB* (1,920 × 1,080 × 2) for two cases of 'Scenario I and II' with one moving object, because two CGH buffers, one for the moving object and the other for the fixed objects, are needed here. For the case of 'Scenario III' with four moving parts of an object, which are arm, mouth and two eyes, five CGH buffers are thus needed. As a result, its total CGH buffer size is increased up to 10*MB* (1,920 × 1,080 × 5). On the other hand, in the MPEG-NLUT method, those sizes are sharply increased up to 593*MB* (1,920 × 1,080 × 300) for all cases of 'Scenario I, II and III' under the condition that the object image with a resolution of 320 × 240 pixels has been divided into 16 × 16 blocks.

These CGH buffers are then stored in the global memory of a GPU board. Here, the global memory tends to have long access latencies and finite access bandwidth. Moreover, even though many threads are available for parallel execution in the GPU board, traffic congestion in the global memory can degrade the calculation performance of the GPU. Thus, the CGH buffer size in the global memory of the GPU board as well as the number of CUDA kernel function calls plays a very important role in determination of the CGH calculation time. As shown in Table 2, the MPEG-NLUT method has been found to have the largest CGH buffer sizes as well as the largest numbers of CUDA kernel function calls and the longest calculation times in all scenarios. Moreover, as the number of blocks of the MPEG-NLUT method increases, its CGH calculation time increases as well. Thus, these results confirm that even though the MPEG-NLUT method shows a good performance in data reduction of 3-D video images, it may not be well matched with the algorithmic and memory structures of the GPU boards.

4.3.3. Analysis on the numbers of CUDA kernel function calls

As seen in Table 2, the proposed method shows the least numbers of CUDA kernel function calls of 58, 44 and 114, respectively, for the 'Scenario I, II and III'. On the other hand, those numbers have been found to be 120, 102, 236 in the TR-NLUT method and 116, 104, 349 in the MC-NLUT method and 1,965, 1,771, 2,807 in the MPEG-NLUT method, respectively, for the 'Scenario I, II and III'. In other words, the numbers of CUDA kernel function calls of the TR-NLUT, MC-NLUT and MPEG-NLUT methods have been increased by 206.90%, 200.00%, 3,387.93% and 231.82%, 236.36%, 4,025.00% and 207.02%, 306.14%, 2,462.28%,

respectively for each case of the 'Scenario I, II and III', when they are compared to those of the proposed method.

These results have stemmed from the fact that the conventional TR-NLUT, MC-NLUT and MPEG-NLUT methods require two-step calculation processes, as mentioned in Chapter 3.2. Therefore, the numbers of CUDA kernel function calls of the TR-NLUT method are given by the summation of $N_{DP\text{-}AOP}$ and $N_{DP\text{-}DOP}$. In addition, those of the MC-NLUT and MPEG-NLUT methods are given by summations of $N_{DP\text{-}AOP}$ and $N_{DP\text{-}DOP}$ for each of the moving objects and for each of the blocks with moving object points, respectively, as shown in Table 1. Specifically, as shown in Table 2, in the MPEG-NLUT method, as the number of blocks increases, its data compression ratio also increases accordingly, whereas the number of CUDA kernel function calls rapidly increases and it causes the total calculation time of the MPEG-NLUT method to be greatly increased on GPU boards.

On the other hand, the proposed OTM-NLUT method has only one-step calculation contrary to the conventional methods. Since the proposed method has no compensation process, only the moving object points of the current frame have been involved in CGH calculation. Thus, the total number of CUDA kernel function calls, which is given by $N_{DP\text{-}MOP}$ in Table 1, turns out to be the least value, when it is compared to those of the conventional TR-NLUT, MC-NLUT, and MPEG-NLUT methods. Then, it drastically reduces the total calculation time of the proposed OTM-NLUT method.

4.3.4. Analysis on the calculation times and the numbers of FPSs

As seen in Table 2, the proposed method has the least average calculation times and the largest average FPSs among those methods across the tested three scenarios. The average calculation times and the average FPSs of the conventional NLUT, TR-NLUT, MC-NLUT, MPEG-NLUT, and proposed OTM-NLUT methods have been found to be 168*msec*, 55*msec*, 52*msec*, 1,081*msec*, 34*msec* and 6.0, 18.0, 19.1, 0.8, 29.8 for 'Scenario I' and 127*msec*, 49*msec*, 48*msec*, 1,010*msec*, 34*msec* and 7.9, 20.3, 20.8, 0.9, 29.1 for 'Scenario II' and 94*msec*, 52*msec*, 66*msec*, 1,684*msec*, 29*msec* and 10.7, 19.2, 15.2, 0.6, 34.3 for 'Scenario III', respectively. In other words, the average calculation times of the proposed method for each of the 'Scenario I, II and III' have been reduced by 79.76%, 38.18%, 34.62%, 96.85% and 73.23%, 30.61%, 29.17%, 96.63% and 69.15%, 44.23%, 56.06%, 98.28%, respectively. In addition, the average FPS of the proposed method for each of the 'Scenario I, II and III' have been increased by 469.67%, 165.56%, 156.02%, 3,725.00% and 368.35%, 143.35%, 139.90%, 3,233.33% and 320.56%, 178.65%, 225.66%, 5,716.67%, respectively, when they are compared to each of the conventional NLUT, TR-NLUT, MC-NLUT and MPEG-NLUT methods.

Based on these results, the FPS of the proposed method has been found to be 31.1, while those of the NLUT, TR-NLUT, MC-NLUT, and MPEG methods have been calculated to be 8.2, 19.2, 18.4 and 0.8, respectively. That is, the proposed method has the biggest FPS value among them. As mentioned above, these experimental results have stemmed from the fact that the proposed OTM-NLUT methods has only a one-step calculation process, requiring much less numbers of CUDA kernel function calls and much less CGH buffer sizes in the global memory of a GPU board than those of the conventional methods.

Especially, the MPEG-NLUT method has been found to have the smallest FPS value of 0.8, which means that it is not well matched with algorithmic and memory structures of the GPU boards, therefore it may not be suitable for GPU-based implementation even though the MPEG-NLUT method, thus far, has shown the best performance in 3-D video compression.

Successful experimental results with three test scenarios having 12,666 object points finally show that the proposed method can generate 31.1 frames of Fresnel CGH patterns with 1,920 × 1,080 pixels per second on average on the NVIDIA GTX TITNAN GPU boards and confirm its feasibility in the practical application fields of most video calls, CCTVs, NEWS, and dramas in television broadcasting.

5. Conclusions

In this paper, a practical OTM-NLUT method, which was designed to be well matched with the software and memory structures of the commercial GPUs, has been proposed and implemented on GPU boards for real-time generation of the CGH patterns of 3-D videos. Experimental results reveal that the proposed method can generate 31.1 frames of Fresnel CGH patterns with 1,920 × 1,080 pixels per second on average for three cases of 3-D video scenarios with 12,666 object points on three NVIDIA GTX TITNAN GPU boards. These successful results confirm the feasibility of the proposed method in the practical application fields such as holographic video calls and television broadcasting systems.