Optical Engineering

SPIEDigitalLibrary.org/oe

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

Min-Woo Kwon
Seung-Cheol Kim
Eun-Soo Kim

SPIE

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

Min-Woo Kwon, Seung-Cheol Kim, and Eun-Soo Kim*
Kwangwoon University, HoloDigilog Human Media Research Center, 3D Display Research Center, Department of Electronic Engineering, 447-1 Wolgye-Dong, Nowon-Gu, Seoul 139-701, Republic of Korea

Abstract. A one-dimensional novel-look-up-table (1-D N-LUT) has been implemented on the graphics processing unit of GTX 690 for the real-time computation of Fresnel hologram patterns of three-dimensional (3-D) objects. For that, three types of optimization techniques have been employed, which include the packing technique of input 3-D object data and the managing techniques of on-chip shared memory and registers. Experimental results show that the average hologram calculation time for one object point of the proposed system has been found to be 0.046 ms, which confirms that the proposed system can generate almost 3 frames of Fresnel holograms with 1920×1080 pixels per second for a 3-D object with 8000 object points. [DOI: 10.1117/1.OE.53.3.035103]

Keywords: computer-generated hologram; novel-look-up-table; sub-principal fringe patterns; graphics processing unit.

Paper 131591 received Oct. 18, 2013; revised manuscript received Jan. 23, 2014; accepted for publication Feb. 12, 2014; published online Mar. 17, 2014.

1 Introduction

Thus far, a number of approaches to accelerate the computational time toward the real-time generation of computer-generated holograms (CGHs) have been proposed.[1–16] One of them is the look-up-table (LUT) method.[1] In this method, a significant increase of the computational speed has been obtained by precalculating all fringe patterns corresponding to point-source contributions from each of the possible locations in the object volume, which are called elemental fringe patterns (EFPs), and by storing them in the LUT. The greatest drawback of this method, however, is the enormous memory capacity for the LUT.[1]

Recently, a novel-LUT (N-LUT) to dramatically reduce the required memory capacity along with maintaining the fast computational speed was proposed.[2] In this approach, a three-dimensional (3-D) object is approximated as a set of discretely sliced object planes having different depths. Then, only the fringe patterns for the center-located object points on each object plane, which are called two-dimensional principal fringe patterns (2-D PFPs), are precalculated and stored in the N-LUT, so that the memory size of the N-LUT could be reduced down to the order of gigabytes (GB) from the order of terabytes (TB) of the conventional LUT for the moderate 3-D object volumes.[2]

Basically, the memory capacity and the computational speed are the most challenging issues in the conventional N-LUT method. Therefore, several attempts to reduce the memory capacity or to accelerate the computation time of the N-LUT have been made for its practical application.[3–8]

For further reduction of the memory, a new type of N-LUT based on one-dimensional (1-D) sub-PFPs decomposed from the conventional 2-D PFPs, which is called here 1-D N-LUT, has been proposed.[3] In this 1-D N-LUT method, the GB memory of the conventional 2-D PFPs-based N-LUT, which is called here 2-D N-LUT, could be dropped down to the order of megabyte (MB) memory.[3]

In addition, for further reduction of the computational time of the N-LUT, several software and hardware methods have been proposed.[4–8] That is, in those software approaches, including the spatial redundancy-based N-LUT,[4] line redundancy-based N-LUT,[5] block redundancy-based N-LUT,[6] as well as the temporal redundancy-based N-LUT,[7] 3-D object data to be calculated have been effectively reduced by removing the spatial or temporal redundancy of the 3-D objects by using various image compression algorithms.

Moreover, a couple of attempts to implement the conventional 2-D N-LUT on a field programmable gate array or a graphics processing unit (GPU) has been done.[9,10] But those approaches were just trials to test the possibility of hardware implementation of the N-LUT, so they could not show any improvement in the computational speed.

Until now, the commercial GPU boards with highly parallel-computing architectures have been employed for accelerating the computational speed of several CGH generation algorithms toward real-time applications.[11–16]

For the N-LUT algorithm, simultaneous loading of a set of PFPs as well as the input 3-D object data onto the internal memory of the GPU is needed for the calculation of Fresnel hologram patterns, so that the memory and computing structure of the GPU must be carefully evaluated for efficient implementation of the N-LUT on them. In this regard, the conventional 2-D N-LUT requiring more than GB memory for storing the 2-D PFPs may not be compatible with the GPU having low-memory capacities. Furthermore, the memory and computing structure of the GPU need to be

*Address all correspondence to: Eun-Soo Kim, E-mail: eskim@kw.ac.kr

optimized for the fast algorithmic processing of the N-LUT because of its limited memory capacity and bandwidth.

Accordingly, in this article, we implement the 1-D N-LUT, maintaining a low-memory usage of megabytes, on the commercial GTX 690 GPU board for the real-time computation of Fresnel hologram patterns of 3-D objects. For fast loading and accessing the sub-PFPs data of the 1-D N-LUT and the 3-D data of the input object on the GPU board, three types of optimization techniques are employed, which 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.

Additionally, experiments with three kinds of test 3-D objects are carried out with the proposed GPU-based 1-D N-LUT system, and the results are comparatively analyzed with those of the conventional central processing unit (CPU)-based 1-D N-LUT system in terms of the computational speed.

2 Analysis of the 1-D and 2-D N-LUT Methods

2.1 *Two-Dimensional N-LUT*

As mentioned above, in the 2-D N-LUT method, the number of fringe patterns to be stored in the N-LUT has been dramatically reduced by employing a new concept of 2-D PFP. Geometry for generating the Fresnel hologram pattern of a 3-D object is shown in Fig. 1 (Ref. 2).

Here, 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 of a_p, φ_p, respectively. Also, the CGH pattern is assumed to be positioned on the plane of $z = 0$.

Actually, a 3-D object can be treated as a set of 2-D object planes discretely sliced along the depth direction of z. Each object plane is approximated as a collection of self-luminous object points of light. In this 2-D N-LUT method, only the fringe patterns of the center points on each image plane are precalculated, which are called 2-D PFPs, and stored in contrary to the LUT method in which the fringe patterns for all possible object points, called 2-D EFPs, are precalculated and stored. Therefore, the unity-magnitude 2-D PFP for the object point $(0, 0, z_p)$ positioned on the center of an object plane with a depth of z_p, $T(x, y; z_p)$, can be defined as follows:[2]

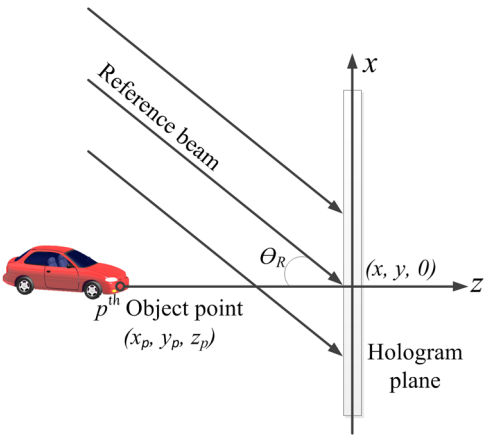


Fig. 1 Geometry for generating the Fresnel hologram pattern of a three-dimensional (3-D) object.

$$T(x, y; z_p) \equiv \frac{1}{r_p} \cos[k r_p + kx \sin \theta_R + \varphi_p]. \tag{1}$$

Here, the wave number k is defined as $k = 2\pi/\lambda$, in which 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

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

Then, the fringe patterns of other object points on same object plane can be obtained by simply shifting this precalculated 2-D PFP.[2] Therefore, fringe patterns of all object points located on the same object plane can be generated by adding these shifted versions of the 2-D PFP, and the final CGH pattern for a 3-D object can be obtained by overlapping all shifted versions of 2-D PFPs that are generated on each depth-dependent object plane.

Therefore, the CGH pattern for a 3-D object $I(x, y)$ in this 2-D N-LUT method can be expressed in terms of the shifted versions of precalculated 2-D PFPs of Eq. (1), as shown in Eq. (3).

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

where N is the number of object points. Equation (3) shows that the CGH pattern of a 3-D object can be obtained by just shifting the 2-D PFPs depending on the displaced values of the object points from the center object points on each object plane and adding up them all.

Here, since the memory capacity of the 2-D N-LUT is calculated to be on the order of GB even for the moderate 3-D object volume with image points of $320 \times 240 \times 256$, in which 320, 240, and 256 denote the number of pixels in the horizontal, vertical, and depth directions, respectively, this 2-D N-LUT may not be well matched with the commercial GPU boards in terms of the computing and memory structures, which means that we could not expect an enhancement in the computational speed from this GPU-based 2-D N-LUT system.

2.2 *One-Dimensional N-LUT*

In the 1-D N-LUT method, 2-D PFPs of the conventional N-LUT can be decomposed into a pair of 1-D sub-PFPs by using a simple trigonometric relation. Therefore, these decomposed 1-D sub-PFPs, instead of the 2-D PFPs, are stored in the 1-D N-LUT for the CGH calculation of the 3-D object, from which a remarkable memory reduction of the 1-D N-LUT down to the order of MB could be obtained.[3]

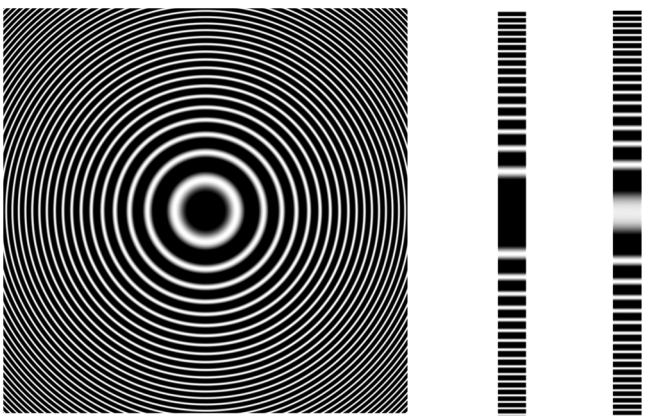
$$
\begin{aligned}
T(x,y;z_p) &= \cos\left[\frac{k}{z_p}(\Delta x^2+\Delta y^2+z_p^2)\right] \\
&= \cos\left[k\left(\frac{\Delta x^2}{z_p}+\frac{\Delta y^2}{z_p}+\frac{z_p^2}{2}+\frac{z_p^2}{2}\right)\right] \\
&= \cos\left[k\left(\frac{\Delta x^2}{z_p}+\frac{z_p^2}{2}\right)+k\left(\frac{\Delta y^2}{z_p}+\frac{z_p^2}{2}\right)\right] \\
&= \cos\left[k\left(\frac{\Delta x^2}{z_p}+\frac{z_p^2}{2}\right)\right]\cos\left[k\left(\frac{\Delta y^2}{z_p}+\frac{z_p^2}{2}\right)\right] \\
&\quad -\sin\left[k\left(\frac{\Delta x^2}{z_p}+\frac{z_p^2}{2}\right)\right]\sin\left[k\left(\frac{\Delta y^2}{z_p}+\frac{z_p^2}{2}\right)\right], \qquad (4)
\end{aligned}
$$

where $\Delta x = x - x_p$ and $\Delta y = y - y_p$. As you can see in Eq. (4), a 2-D PFP can be simply calculated by using a pair of 1-D sub-PFPs according to the trigonometric relation.

Figures 2(a) and 2(b) show a 2-D PFP and its 1-D version of a pair of sub-PFPs for a specific depth plane. For 256 depth layers, 256 1-D sub-PFPs are stored in the 1-D N-LUT, whereas the same number of 2-D PFPs is stored in the 2-D N-LUT.

For memory comparison, here we assume a 3-D object composed of plane images sliced with 256 levels along the depth direction, in which each plane image has a resolution of 320×240. In addition, the resolution of the CGH pattern to be generated is assumed to be 1920×1080 pixels, in which each pixel size is $10 \times 10\ \mu m^2$. Also, it has been well known that the human visual system usually sees as continuous two points that are separated by 3 mrad of arc. Therefore, if the viewing distance is assumed to be 100 mm from the image plane, then the separation between two object points becomes 30 μm ($100\ mm \times 0.003 = 30\ \mu m$).[1] Thus, the pixel size of hologram pattern of 10 μm corresponds to the shift of 3 pixels. In other words, to display the two neighboring points to be seen as continuous, the PFP should be shifted by 3 pixels.[2]

Hence, to fully display the CGH pattern, the 2-D PFP must be shifted by 960 ($320 \times 3 = 960$) and 720 ($240 \times 3 = 720$) pixels along the horizontal and vertical directions, respectively, which means that the resolution of a 2-D PFP becomes 2880×1800 pixels, as seen in Eq. (5).



(a) (b)

Fig. 2 Two-dimensional (2-D) PFP and its one-dimensional (1-D) version of sub-PFPs: (a) 2-D PFP, and (b) 1-D sine and cosine sub-PFPs for a specific depth plane.

$$
\begin{aligned}
&\text{Horizontal resolution of the 2-D PFP}(2880) \\
&\quad = \text{horizontal resolution of the 3-D object}(320) \\
&\qquad \times \text{intervals of the reconstructed image point}(30)/ \\
&\qquad\ \text{pixel pitch of the hologram}(10) \\
&\qquad + \text{horizontal resolution of the CGH}(1920) \\
&\text{Vertical resolution of the 2-D PFP}(1800) \\
&\quad = \text{vertical resolution of the 3-D object}(240) \\
&\qquad \times \text{intervals of the reconstructed image point}(30)/ \\
&\qquad\ \text{pixel pitch of the hologram}(10) \\
&\qquad + \text{vertical resolution of the CGH}(1080). \qquad (5)
\end{aligned}
$$

Therefore, the total memory capacity of the 2-D N-LUT is calculated to be 1.2 GB, as shown in Eq. (6).

$$
\begin{aligned}
&\text{Total memory capacity of the 2-D N-LUT}(1.2\ \text{GB}) \\
&\quad = \text{horizontal resolution of the 2-D PFP}(2880) \\
&\qquad \times \text{vertical resolution of the 2-D PFP}(1800) \\
&\qquad \times \text{depth resolution of the 3-D object}(256). \qquad (6)
\end{aligned}
$$

On the other hand, in the 1-D N-LUT method, to fully display the sub-PFP for the 3-D object volume having a resolution of $320 \times 240 \times 256$ pixels, the 1-D PFP must be shifted by 960 ($320 \times 3 = 960$) and 720 ($240 \times 3 = 720$) pixels along the horizontal and vertical directions, respectively. Thus, the resolution of a 1-D sub-PFP becomes 2880 ($960 + 1920 = 2880$) pixels, and as a result, the total memory capacity of the 1-D N-LUT is calculated to be 1.4 MB, as seen in Eq. (7).

$$
\begin{aligned}
&\text{Total memory capacity of the 1-D N-LUT}(1.4\ \text{MB}) \\
&\quad = \text{resolution of the 1-D PFP}(2880) \\
&\qquad \times \text{the number of 1-D PFPs}(2) \\
&\qquad \times \text{depth resolution of the 3-D object}(256). \qquad (7)
\end{aligned}
$$

Accordingly, the memory size of the 1-D N-LUT has found to be almost 10^3-fold smaller than that of the conventional 2-D N-LUT.

3 Compatibility of the 1-D N-LUT with the GPU

3.1 *Memory Capacity*

In the N-LUT-based CGH generation method, a set of PFPs for the 3-D object must be precalculated and stored for calculation of the CGH patterns. That is, the N-LUT method needs a simultaneous loading of a set of PFPs as well as the input object data on the internal memory of the GPU. Basically, the memory size for the PFPs would be much larger than that for the object data, which means that the memory structure of the commercial GPU board must be carefully evaluated for efficient implementation of the N-LUT on it. Since a new type of N-LUT called 1-D N-LUT, which can maintain a low-memory usage of megabytes as well as operate

Table 1 Global memory specifications of various commercial GPUs.

	General-purpose GPU				Professional GPU	
	GTX 690	GTX 580	GTS 450	GT 640	Quadro 6000	TeslaK10
GDDR5 memory (GB)	2	1.5	1	2	6	8

on the 1-D data structure, is well matched with those of the commercial GPUs, here in this article, the 1-D N-LUT is employed for its implementation on the GPU board.

Table 1 shows the GDDR5 (graphics double data rate, version 5) memory specifications in various commercial GPUs.[17–19] Here, the GDDR5 SDRAM is a type of high-performance DRAM graphics memory card designed for the computer applications requiring a high bandwidth, which is the largest memory in the GPU called "global memory." Therefore, in the 1-D N-LUT method, sub-PFPs and object data may be stored in this global memory of GDDR5 SDRAM.

For example, the general-purpose GPU of GTX 690 and the professional GPU of Quadro 6000 have the global memories of 2 and 6 GB, respectively, as seen in Table 1.

Moreover, Fig. 3 shows the memory size dependence on the resolution of the input object in both the 1-D and 2-D N-LUT methods, in which the resolution of the hologram to be generated is assumed to be 1920×1080.

For the 3-D object volume having image points of $300 \times 300 \times 256$, the total memories required in the 1-D and 2-D N-LUTs are calculated to be 1.38 MB and 1.33 GB, respectively. Moreover, they become 3.87 MB and 13.37 GB for each case of the 1-D and 2-D N-LUTs for the 3-D object having the image points of $2000 \times 2000 \times 256$. These results reveal that as the resolution of the input object increases, the corresponding memory size for the 2-D N-LUT gets sharply increased according to the square-law on the GB range, whereas it may linearly increase on the MB range in the 1-D N-LUT.

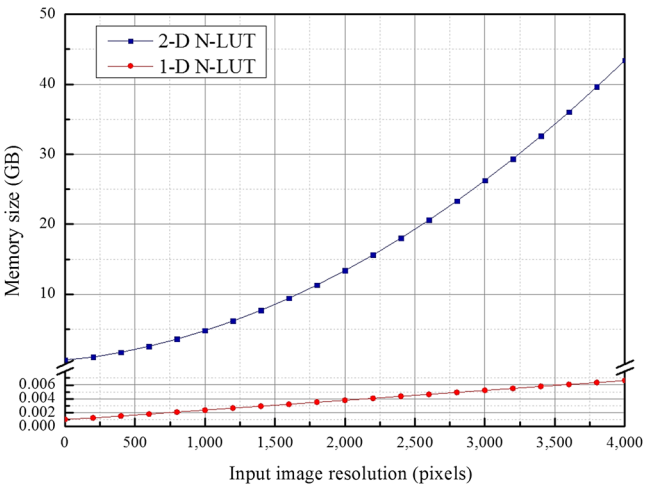


Fig. 3 Memory size dependence on the resolution of the input object in both the 1-D and 2-D N-LUTs.

Even for the 3-D object volume having image points of $4000 \times 4000 \times 256$, only 6.8 MB memory is required for the 1-D N-LUT, which is the sufficient amount of memory to be fully stored in the global memories of the GTX GPU series with the maximum memory size of 2 GB. In other words, the 1-D N-LUT looks highly compatible with the commercial GTX GPU series, so that one of the general-purpose GPU, the GTX 690 GPU, has been chosen here for implementation of the 1-D N-LUT on it.

3.2 *Memory Structure*

In fact, the 1-D N-LUT has been developed for its implementation on the commercial GPU boards.[3] That is, contrary to the conventional 2-D N-LUT, the memory and computing structures of the 1-D N-LUT are well matched with those of the GTX 690 GPU.[17]

Generally, a GPU board supports several types of memories that can be used by programmers to achieve a high performance and thus a high-execution speed in their kernels. Figure 4 shows the memory structure and compute-unified-device-architecture (CUDA) thread organization of the GTX 690 GPU.[17] Here, in the CUDA thread organization, the thread characterizes the fundamental means of parallel execution in CUDA, and each block is composed of a group of threads. Moreover, the global memory shown at the bottom of Fig. 4 is written and read by the host by calling application programming interface functions, but it has problems of long latency and limited bandwidth.

Hence, the GTX 690 GPU also supports the on-chip short-latency shared memory and registers. As shown in Fig. 4, variables that reside on those memories can be accessed at the very high speed in a highly parallel manner.

Although these on-chip shared memory and registers can be effectively used for the reduction of the access number to the global memory, one must be careful not to exceed the capacity of these memories. Each GPU device offers a limited amount of GPU memory, which may limit the number of threads that can be simultaneously resided in multiprocessor for a given application.[20]

For the GTX 690 GPU board employed in this article, the total amount of shared memory per block (SMPB) and the total number of registers available per block are 49,152 bytes (48 kB) and 65,536 bytes (64 kB), respectively.[17]

4 GPU-Based Implementation of the 1-D N-LUT

4.1 *Ordinary Implementation*

Figure 5 shows a basic software structure and pseudocode for implementation of the 1-D N-LUT on the GTX 690 GPU board using the global memory, which is called here "Ordinary GPU-based 1-D N-LUT system." It largely consists of three parts: Input, Calculation, and Output parts. That is, in the Input part, x, y, z coordinates and intensity data extracted from the input 3-D object and sub-PFPs of the 1-D N-LUT are stored in the global memory of the GPU board. In the Calculation part, GPU kernel functions are invoked to load x, y coordinates and intensity data and sub-PFPs for specific z depth from the global memory and to calculate the CGH pattern. Furthermore, in the Output part, the calculated CGH pattern is saved and the GPU kernel cumulates the calculated CGH patterns to

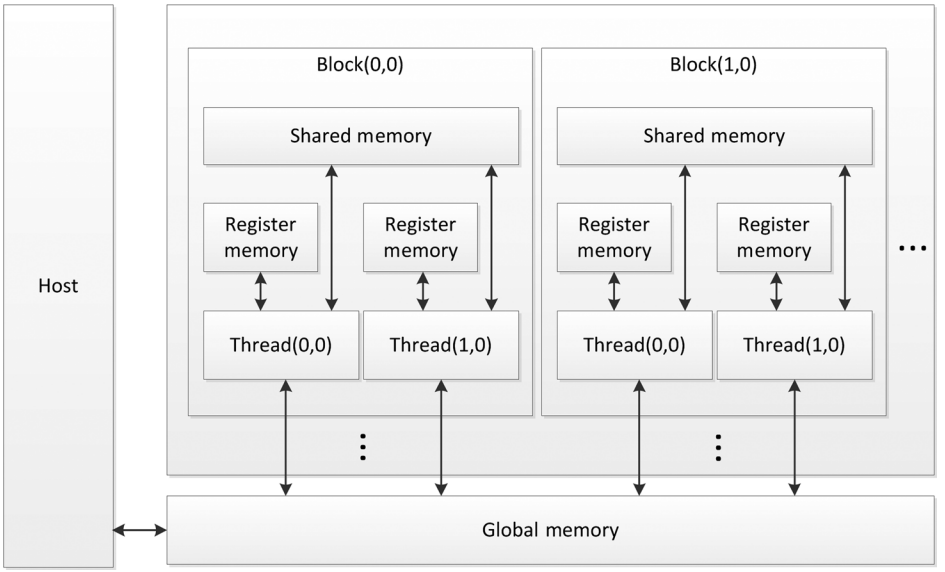


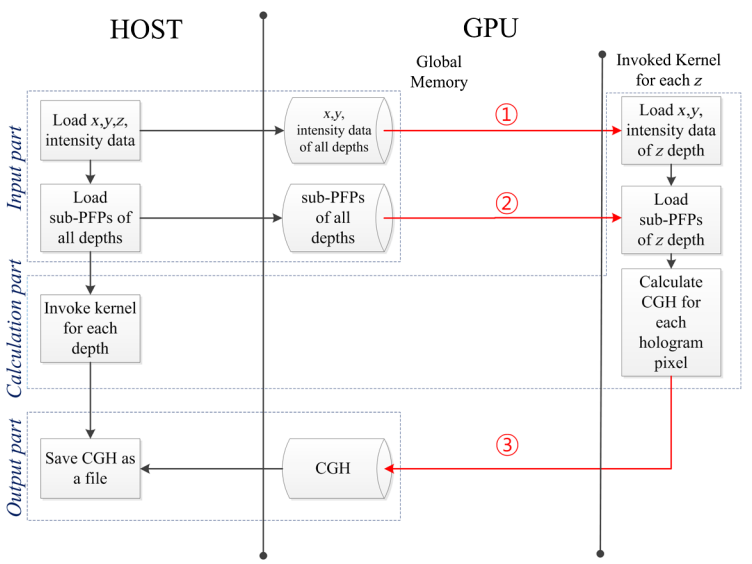
Fig. 4 Memory structure of the GTX 690 GPU board.

the global memory. Right after these CGH calculations are completed for all depth layers, the host computer loads the cumulated CGHs from the global memory and then the main program may end.

Actually, this basic software structure based on the global memory of the GPU board has been employed in a couple of conventional GPU-based CGH generation methods.[11,12] However, in this software structure of the ordinary GPU-based system shown in Fig. 5, the fast-accessible on-chip shared memory and registers of the GTX 690 GPU board may not be well managed for accelerated CGH calculations; so, an optimization process for efficient utilization of them is needed. Figure 5 shows the potential bottlenecks (①, ②, and ③) of this ordinary GPU-based system. That is, if the GPU program is implemented only with the global memory, then it may experience a traffic jam in loading and accessing the memory data, which may result in a degradation of the system performance.

In particular, since a large sub-PFPs data as well as the small object data are simultaneously stored in the global memory of the GPU board from the host in the ordinary GPU-based system as shown in ① and ② of Fig. 5, a kind of traffic jams between the global memory and the threads in the GPU may happen whenever the data are accessed, because the global memory has the long-access latency as well as the limited memory bandwidth. Moreover, whenever the threads store the calculated results as shown in ③ of Fig. 5, multiple write operations may occur between the threads and the global memory. It also causes the degradation of the system performance.

Accordingly, if we can alleviate those bottlenecks mentioned above, then the computational speed of the ordinary GPU-based system expects to be much enhanced. Therefore, in this article three types of optimization techniques for data loading and accessing in the GPU board are proposed. These will be discussed in detail later.




```
Main function
  Load x,y,z and intensity data of 3-D object
  Arrange x,y and intensity data for each z depth
  Store x,y and intensity data of all depths in global memory
  Load sub-PFPs
  Store sub-PFPs in global memory
  For each z depth
    Invoke kernel function
  End for loop
  Load CGH from global memory
End function

Kernel function
  For each x,y and intensity data in specific depth
    Load x,y and intensity data and sub-PFPs of specific depth
    Calculate CGH
    Cumulate CGH to global memory
  End for loop
End function
```

Fig. 5 Basic software structure and pseudocode of the ordinary GPU-based 1-D N-LUT system.

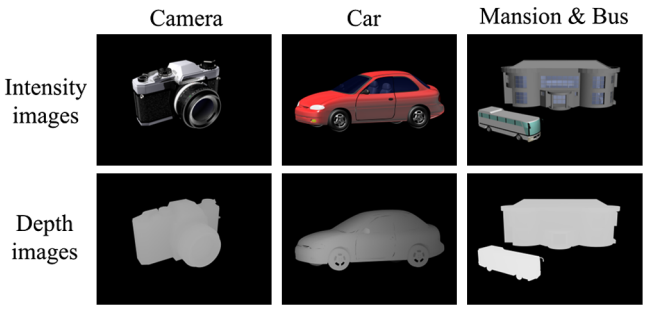


Fig. 6 Intensity and depth images of the test objects of "Camera," "Car," and "Mansion & Bus."

Here, the ordinary GPU-based 1-D N-LUT system employing the basic software structure of Fig. 5 is first implemented, and experiments are carried out for comparison. Intensity and depth images of three test 3-D objects are shown in Fig. 6, which are used as the input object data for the generation of its CGH pattern on the implemented ordinary GPU-based system. In the experiments, three test objects of "Camera," "Car," and "Mansion & Bus" with the same 3-D object volume of $320 \times 240 \times 256$ pixels are assumed to have object points of 3021, 9944, and 19,984, respectively. Moreover, the CGH pattern to be generated is assumed to have a resolution of 1920×1080 pixels, in which each pixel size is 10×10 μm^2, and the horizontal and vertical discretization steps of less than 30 μm are chosen since the viewing distance is fixed to be 100 mm in the experiment.

With these test object data and corresponding sub-PFPs, CGH patterns are calculated by using the implemented ordinary GPU-based 1-D N-LUT system. For comparison, the same calculations are also performed with the conventional CPU-based 1-D N-LUT system. Here, the computer system used in the experiment is composed of an Intel i7 3770 CPU, an 8 GB RAM, and a GTX 690 GPU of NVIDIA, and it works on the CentOS 6.3 Linux platform. Basically, even though the GTX 690 GPU board is composed of two GTX 680 GPU boards, only one GTX 680 GPU board is employed in our experiment for just verifying the compatibility of the 1-D N-LUT with the GPU. Moreover, we use only one CPU thread, in which this CPU was not used for CGH calculations but for controlling the software structure such as loading the input images and N-LUT, controlling the GPU kernels, and saving the output holograms.

Experimental results show that the average calculation times for one object point in the implemented ordinary GPU-based and the conventional CPU-based systems are estimated to be 0.143 and 11.956 ms, respectively. Here, the average calculation time for one object point of the ordinary GPU-based system has been found to be 83-fold improved compared with that of the conventional CPU-based system, but these results may say that the implemented ordinary GPU-based system still needs a further improvement in the computational speed for the real-time applications.

4.2 *Proposed Implementation*

Figure 7 shows an overall software structure and pseudocode of the proposed GPU-based 1-D N-LUT system, which is called here "Proposed GPU-based 1-D N-LUT system." In the proposed system, three types of optimization techniques to efficiently manage the on-chip shared memory and registers of the GPU board are suggested for solving the bottlenecks of the ordinary GPU-based system. Here, the optimization techniques may depend on the employed CGH generation algorithms,[16,21,22] so, in this article, new types of memory managing techniques properly optimized to the 1-D N-LUT are proposed.

As shown in Fig. 7, in the Input part, the 3-D object data are packed together for efficient storing as much object data as possible in the on-chip shared memory. In the Calculation and Output parts, two memory managing techniques for utilizing both the fast-accessible on-chip shared memory and registers are used. These optimization techniques may improve the performance of the proposed GPU-based 1-D N-LUT system.

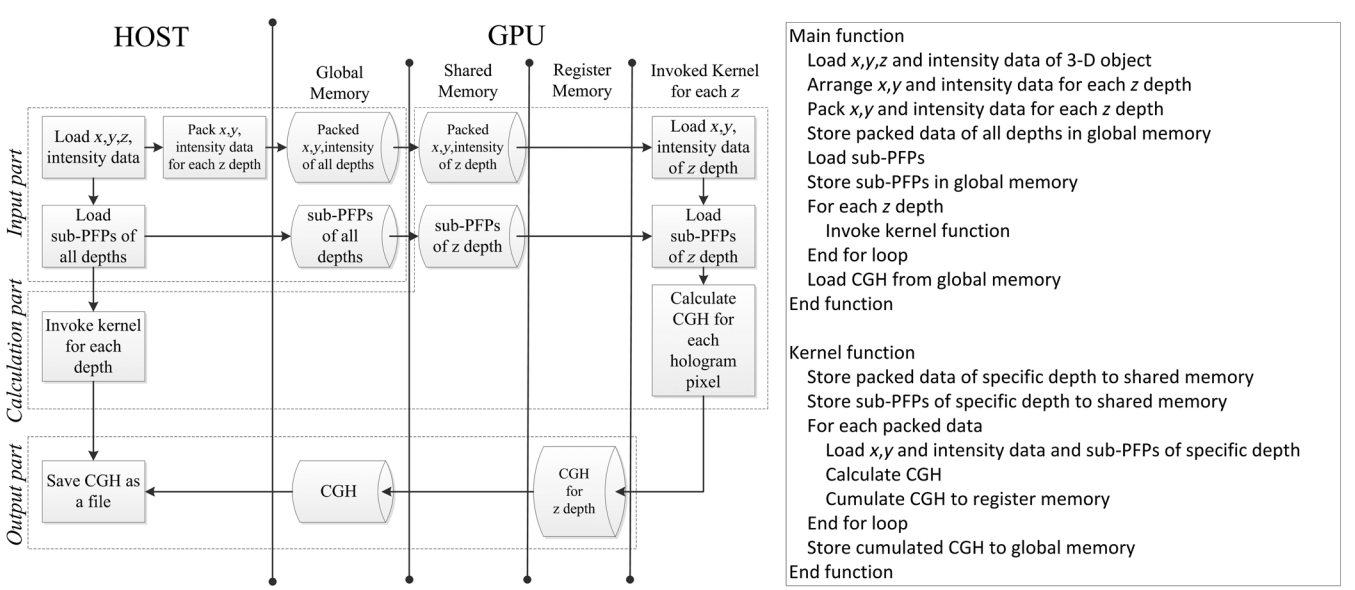


Fig. 7 Overall software structure and pseudocode of the proposed GPU-based 1-D N-LUT system.

4.2.1 *Operational design of the proposed system*

In this article, the GPU board calculates the CGH at a high speed in parallel by using the CUDA. The CUDA is a parallel-computing platform and programming model invented by the NVIDIA. It enables a dramatic enhancement of computing performance by harnessing the power of the GPU. In the CUDA, the computing performance of the GPU depends on the occupancy of the GPU cores. In other words, it may depend on how efficiently the GPU cores in parallel are used.

In this article, each thread of the GPU board calculates each pixel of the CGH. That is, a parallel-processing algorithm can be implemented based on each pixel of the CGH. Figure 8 shows how to perform the parallel processing based on each pixel of the CGH, in which "Hologram cols" and "Hologram rows" represent the column and row sizes of the CGH pattern, respectively. That is, one thread is allocated to one pixel of the CGH and several threads are grouped by a block.

Here, occupancy of the GPU cores is calculated by using the "CUDA GPU Occupancy Calculator (CGOC)."[23] Input parameters of the CGOC include "threads per block (TPB)," "registers per thread (RPT)," and "shared memory per block (SMPB)." In the CGOC, occupancy of the GPU is evaluated by the ratio between the active number of "warps per multiprocessor (WPM)" and the maximum number of WPM, in which the maximum number of WPM is 64 and the active number of WPM is obtained by multiplication of the number of "warps per block (WPB)" to the number of "active block (AB)." Here, the WPB is evaluated by dividing the TPB with the "threads per warp (TPW)," in which the TPW is 32.

To get 100% occupancy of the GPU cores, the TPB, RPT, and SMPB are determined to be 512, 19, and 12,288, respectively. Here, the AB can be calculated with each of the TPB, RPT, and SMPB.[23] That is, if the TPB is 512, the AB is calculated to be $4(64/16 = 4)$ through division of the maximum number of WPM by the number of WPB. For the case if the RPT is 19, AB is then calculated to be $5(84/16 = 5.25)$ by dividing the "registers per multiprocessor (RPM)" with the "registers per block (RPB)." Moreover, if the SMPB is 12,288, then the AB is also calculated to be $4(49{,}152/12{,}288 = 4)$ through division of the "shared memory per multiprocessor (SMPM)" by the SMPB, in which the SMPM is 49,152.

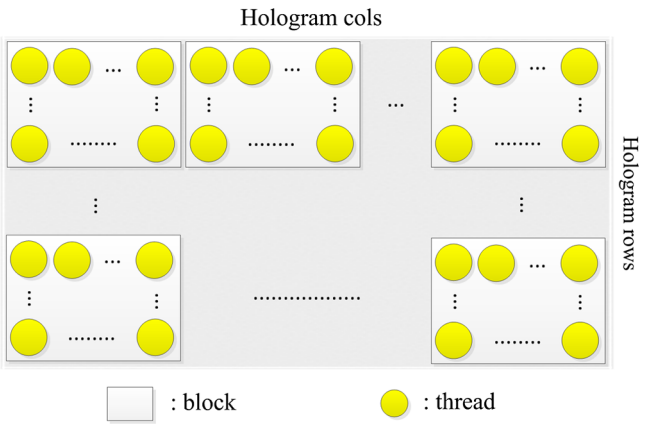


Fig. 8 Block thread-based parallel processing of the CGH.

Since the CGOC may select the smallest value of the AB among three cases derived above, the AB is finally determined to be 4. Therefore, the value of active number of WPM is calculated to be $64(16 \times 4 = 64)$ through multiplication of this AB value to the WPB. Then, the active and the maximum numbers of WPM become exactly the same, and this result confirms 100% $(64/64 \times 100 = 100)$ occupancy of the GPU. These calculations explained above are automatically performed by using the CGOC (CUDA GPU OC).[23] In short, the proposed GPU-based 1-D N-LUT system has been designed to work with 100% occupancy of the GPU cores as well as to secure the maximum space of the on-chip shared memory for simultaneous loading of x, y coordinates and intensity data extracted from the input 3-D object and the sub-PFPs data for specific z depth.

4.2.2 *Optimization of the input part*

In the ordinary GPU-based 1-D N-LUT system, x, y, z coordinates and intensity data extracted from the input 3-D object as well as the sub-PFPs data are stored in the global memory of the GTX 690 GPU. On the other hand, in the proposed system as shown in Fig. 7, both of them are stored in the global memory temporary and in the on-chip shared memory, whenever the CGH calculation operation starts.

Here, in order to maintain the designed 100% occupancy of the GPU cores, the SMPB must be kept to not larger than 12,288 bytes $(49{,}152/4 = 12{,}288)$ as discussed above. For the 3-D object volume with object points of $320 \times 240 \times 256$ and the CGH pattern with a resolution of 1920×1080 pixels, the resultant memory size of a pair of sub-PFPs for specific z depth is calculated to be 5760 bytes $(2880 \times 2 = 5760)$. Then, for loading these sub-PFPs data of 5760 bytes onto the on-chip shared memory by using 512 threads in parallel, 6144 bytes of the shared memory, which is the multiple numbers of 512, must be needed. Therefore, the remaining portion of the shared memory of $6144(12{,}288 - 6144 = 6144)$ bytes can be allocated for loading x, y coordinates and intensity data extracted from the input 3-D object for specific z depth.

Accordingly, in the proposed system, to store as many of these data as possible in this limited on-chip shared memory, combined packing scheme of 3-D object data is proposed. In the 1-D N-LUT method, a set of sub-PFPs corresponding to each depth layer is provided for the calculation of the CGHs of the 3-D object. The GPU kernel function may be invoked in relation to the depth layer, so that the x, y coordinates and intensity values can be arranged together as tables for each depth layer having their z coordinate values.

Here, the input object data are packed in the 32-bit memory, in which x, y coordinate values and the intensity value are allocated to 24-bit and 8-bit of the memory, respectively. By using this packing scheme, the input image data for one object point could be reduced down to 4 bytes from $12(4 \times 3 = 12)$ bytes because x, y coordinate values and the intensity value can be carried only by one 32-bit (4 bytes) memory.

4.2.3 *Optimization of the calculation part*

Compared with the ordinary system, in which x, y, z coordinates and intensity data extracted from the input 3-D object and 1-D sub-PFPs are stored on the global memory as shown in Fig. 5, the proposed system utilizes the on-chip shared

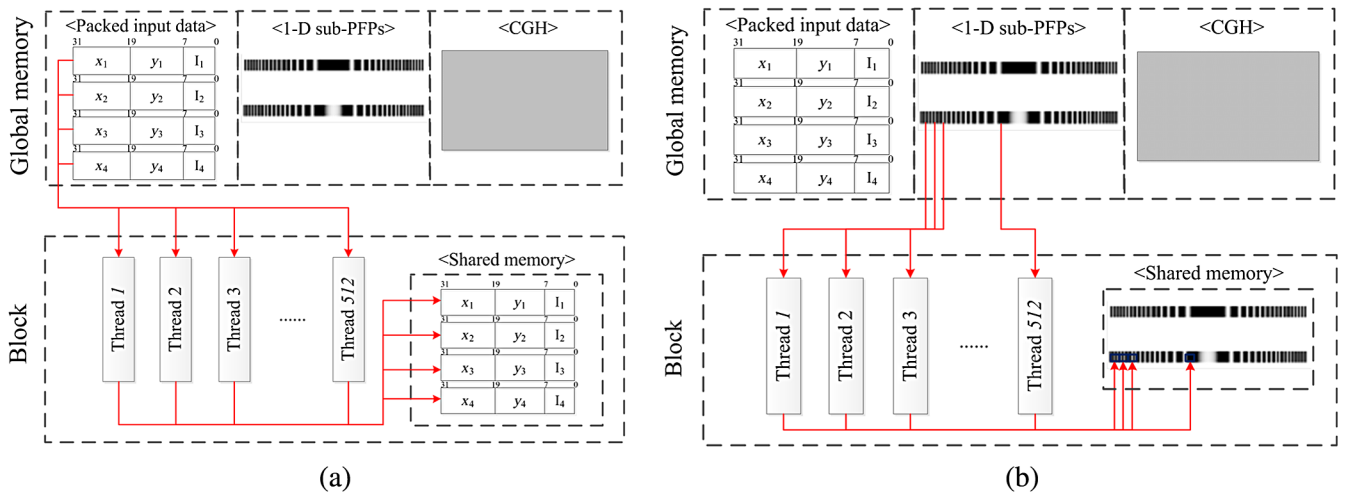


Fig. 9 Parallel storing of the sub-PFPs and the packed input object data to the on-chip shared memory: (a) storing of sub-PFPs, and (b) storing of the packed input object data.

memory to fast access these data to the threads for accelerated computation of the CGH in the Calculation part, as shown in Fig. 7. That is, the threads load the packed x, y coordinates and intensity data extracted from the input 3-D object and the sub-PFPs for specific z depth from the global memory and save them onto the shared memory in parallel. Figure 9 shows the parallel loading process of these data from the global memory and saving them onto the shared memory.

Figure 10 shows the implemented code for the calculation of the CGH pattern with the 1-D N-LUT. Here, "PACKED_DATA" means the packed object data loaded from the shared memory, which consist of x, y coordinates and intensity values packed as one integer variable. Using a bit operation of the C-language, these three values can be obtained as shown in Fig. 10(b). "PFP_X" and "PFP_Y," representing the x, y coordinate values of the 1-D sub-PFP, respectively, can be obtained by using the "MAGNIFICATION," which is 3 pixels, and "OBJECT_X" and "OBJECT_Y" denote the x, y coordinate values of the input object, as shown in Fig. 10(b).

Then, "cos _X," "cos _Y," "sin _X," and "sin _Y" representing four terms of the last line in Fig. 10(a), are obtained by using "COS_subPFPs," and "SIN_subPFPs," which are a pair of 1-D sub-PFPs loaded from the shared memory.

Based on the equation of Fig. 10(a), "HOLOGRAM" denoting the CGH can be finally obtained, in which "HOLOGRAM_INDEX" means the index of the CGH.

4.2.4 *Optimization of the output part*

Since in the ordinary GPU-based system, each thread calculates one pixel of the hologram and directly cumulates its result on the global memory, a tremendous number of read-write-operations may occur between the threads and the global memory. Therefore, in the proposed system, for accelerated accumulation of the calculated CGH pattern, the on-chip registers are used for storing the calculated CGH, as shown in Fig. 7.

That is, each thread calculates the CGH with x, y coordinates and intensity data extracted from the input 3-D object and the sub-PFPs for specific z depth and cumulates the results on the registers. In case of calculations for one depth layer end, each thread saves the cumulated results onto the global memory. Therefore, the number of read-write-operations between the threads and the global memory can be significantly reduced.

In the ordinary GPU-based system mentioned above, the number of read-write-operations between the threads and the global memory to cumulate the calculated CGH pattern is given by Eq. (8).

$$T(x,y;z_p)=\cos\left[\frac{k}{z_p}\left(\Delta x^2+\Delta y^2+z_p^2\right)\right]$$
$$=\cos\left[k\left(\frac{\Delta x^2}{z_p}+\frac{\Delta y^2}{z_p}+\frac{z_p^2}{2}+\frac{z_p^2}{2}\right)\right]=\cos\left[k\left(\frac{\Delta x^2}{z_p}+\frac{z_p^2}{2}\right)+k\left(\frac{\Delta y^2}{z_p}+\frac{z_p^2}{2}\right)\right]$$
$$=\cos\left[k\left(\frac{\Delta x^2}{z_p}+\frac{z_p^2}{2}\right)\right]\cos\left[k\left(\frac{\Delta y^2}{z_p}+\frac{z_p^2}{2}\right)\right]-\sin\left[k\left(\frac{\Delta x^2}{z_p}+\frac{z_p^2}{2}\right)\right]\sin\left[k\left(\frac{\Delta y^2}{z_p}+\frac{z_p^2}{2}\right)\right]$$

(a)

```
intensity = (PACKED_DATA&0xFF);
OBJECT_X = ((PACKED_DATA>>8)&0xFFF);
OBJECT_Y = ((PACKED_DATA>>20)&0xFFF);

PFP_X = MAGNIFICATION * OBJECT_X + HOLOGRAM_X;
PFP_Y = MAGNIFICATION * OBJECT_Y + HOLOGRAM_Y;

cos_X = COS_subPFPs[PFP_X];
cos_Y = COS_subPFPs[PFP_Y];
sin_X = SIN_subPFPs[PFP_X];
sin_Y = SIN_subPFPs[PFP_Y];

HOLOGRAM[HOLOGRAM_INDEX] += ((cosX*cosY-sinX*sinY)*intensity);
```

(b)

Fig. 10 Implemented code for calculating the CGH with the 1-D N-LUT: (a) equation of 2-D PFP by using a pair of 1-D sub-PFPs, and (b) implemented code.

$$N_{\text{Gmem\_ordinary\_system}} = N_{\text{Object}} \times N_{\text{Depth}}, \tag{8}$$

where N_{Object} and N_{Depth} mean the numbers of input object points and depth layers, respectively. On the other hand, in the proposed GPU-based system, the number of read-write-operations between the threads and the global memory to cumulate the calculated CGH pattern can be given by Eq. (9).

$$N_{\text{Gmem\_optimized\_system}} = N_{\text{Depth}}. \tag{9}$$

Therefore, it has been found from Eqs. (8) and (9) that the number of read-write-operations between the threads and the global memory to cumulate the calculated CGH pattern in the proposed method only depends on the number of depth layers regardless of the number of input object points, whereas it may depend on both the number of depth layers and the number of input object points in the ordinary method.

5 Experimental Results and Discussions

5.1 *CGH Calculation Time*

Figure 11 shows the comparison results of the reconstructed test object images from the CGH patterns generated by each of the conventional CPU-based, the ordinary GPU-based, and proposed GPU-based 1-D N-LUT methods for three test objects of "Camera," "Car," and "Mansion & Bus." Here in the experiments, off-axis reference lights were used for reconstruction of the object images without having direct and conjugate beams. As you can see in Fig. 11, all object images have been successfully reconstructed and visually look very well.

Table 2 shows the results of the average total CGH calculation times and the average CGH calculation times for one object point for each method. As seen in Table 2, the average CGH calculation times for one object point are estimated to be 11.956, 0.143, and 0.046 ms, respectively, for each case of the conventional CPU-based, the ordinary GPU-based, and the proposed GPU-based systems. In other words, the proposed GPU-based system has obtained 260 and threefold improvement of the average CGH calculation time, respectively, compared with those of the conventional CPU-based and the ordinary GPU-based systems.

Here, the total CGH calculation time for the test 3-D object of "Car" with 9944 object points is calculated to be 446 ms in the proposed system, which means that the proposed system can generate 2.24 frames of the CGH patterns with 1920×1080 pixels per second.

These results may confirm the feasibility of the proposed GPU-based system in the practical applications. Moreover, the performance of the proposed system could be further improved through massive reduction of the input 3-D object data by employing a concept of spatial and temporal redundancies as well as by multi-GPU-based implementation.

Figure 12 shows the total CGH calculation time dependence on the number of object points for each case of the conventional CPU-based, the ordinary GPU-based, and the proposed GPU-based 1-D N-LUT methods, in which the CGH resolution is set to be 1920×1080 and the number of object points is changed from 1 to 25,000.

For an input object volume with 10,000 object points, the total CGH calculation time of the proposed system has been improved by 99.64% and 68.53% compared with those of the conventional CPU-based and the ordinary GPU-based systems, respectively. The results on the total calculation times for three test objects of "Camera," "Car," and "Mansion & Bus," as shown in Table 2, are also included in Fig. 12. As seen in Fig. 12, the results of Table 2 have been well matched with those of Fig. 12.

From Fig. 12, the total CGH calculation time for a 3-D object volume having 8000 object points has been estimated to be about 0.36, 1.10, and 94.71 s in the proposed, the ordinary, and the conventional CPU-based systems, respectively,

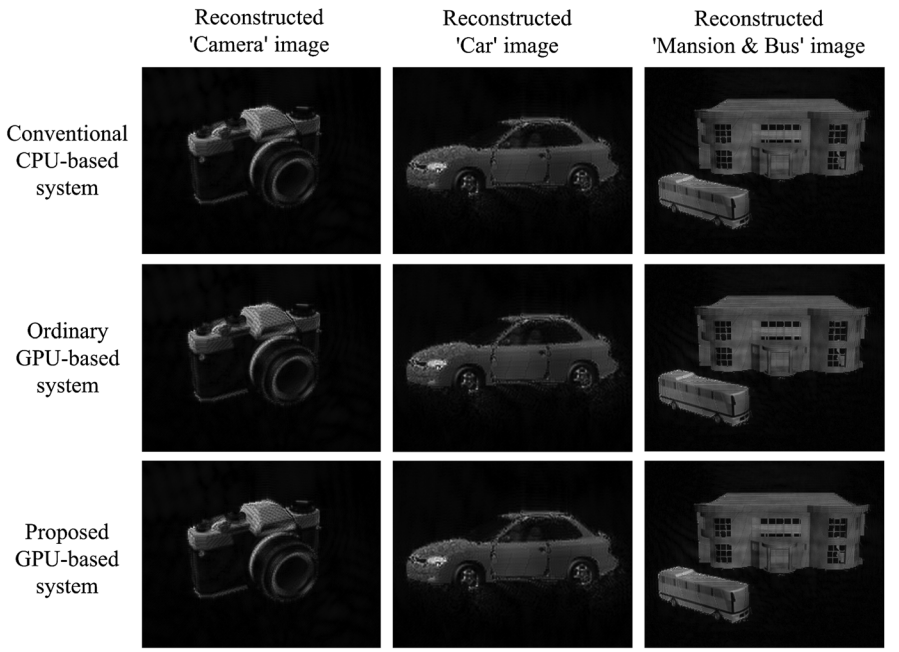


Fig. 11 Comparison of input images and results on the reconstructed test object images from the conventional CPU-based, the ordinary GPU-based, and the proposed GPU-based 1-D N-LUT methods.

Table 2 Comparison results on the total CGH computation times and the average calculation times for one object point of each system for three test objects.

Implementation method		Test object (number of object points)	Total calculation time (ms)	Calculation time for one point (ms)
Conventional CPU-based		Camera (3021)	36,113 (100%)	11.954 (100%)
GPU-based	Ordinary		457 (1.27%)	0.151 (1.27%)
	Proposed		146 (0.40%)	0.048 (0.40%)
Conventional CPU-based		Car (9944)	118,861 (100%)	11.953 (100%)
GPU-based	Ordinary		1384 (1.16%)	0.139 (1.16%)
	Proposed		446 (0.38%)	0.045 (0.38%)
Conventional CPU-based		Mansion & bus (19,984)	239,056 (100%)	11.962 (100%)
GPU-based	Ordinary		2751 (1.15%)	0.138 (1.15%)
	Proposed		871 (0.36%)	0.044 (0.36%)

which means that the proposed system can generate 3 frames of the CGH patterns with 1920×1080 per second, whereas the ordinary GPU-based and the conventional CPU-based systems can generate 1 and 0.01 frames of the CGH patterns per second, respectively.

5.2 *Achieved Occupancy of the GPU Cores*

The NVIDIA Visual Profiler can be used for showing the real occupancy value of the multiprocessor, which is called here "achieved occupancy (AO)." The AO of the proposed GPU-based system has been found to be 90.6%, which means that the proposed system may operate with 90.6% of the GPU cores. As discussed above, the proposed system has been designed to work with 100% occupancy of the GPU cores, but its real value in the implemented system has been evaluated to be a little bit lower than that. But this value of the AO may be generally acceptable in the real-application fields.[24]

5.3 *Occupancy of the GPU Cores Depending on the Shared Memory*

Figure 13 shows the occupancy of the GPU cores dependence on the shared memory size. As seen in Fig. 13, if the size of the shared memory for x, y coordinates and intensity data extracted from the input 3-D object and sub-PFPs data for specific z depth increases in the proposed system, then the occupancy of the GPU cores tends to be sharply decreased even though the available shared memory space for loading these data increases.

That is, if the shared memory size to be used for these data gets increased up to 16,384, 24,576, and 30,000 bytes, exceeding its upper limit of 12,288 bytes for maintaining the designed 100% occupancy of the GPU cores, then the corresponding AB values decrease down to $3(49{,}152/16{,}384{=}3)$, $2(49{,}152/24{,}579 = 2)$, and $1(49{,}152/30{,}000 = 1.64)$, respectively. Then, their active numbers of WPM values also reduce to $48(16 \times 3 = 48)$, $32(16 \times 2 = 32)$, and $16(16 \times 1 = 16)$, respectively, which finally results in

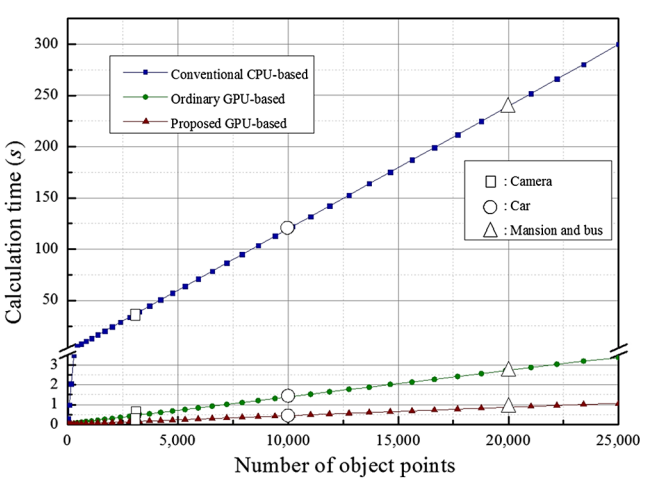


Fig. 12 Results of calculation performance according to the number of point and input objects.

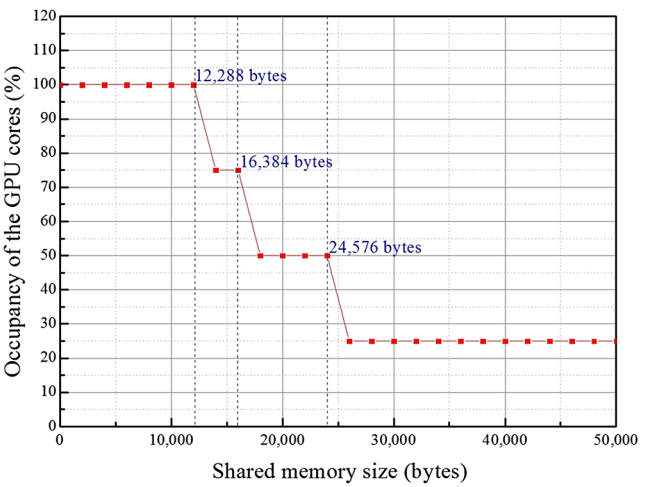


Fig. 13 Occupancy of the GPU cores dependence on the shared memory size.

occupancy reduction of 75% $(48/64 \times 100 = 75)$, 50% $(32/64 \times 100 = 50)$, and 25% $(16/26 \times 100 = 25)$, respectively.[23,24]

That is, the more we use the on-chip shared memory of the GPU board, the more we can store the input object and sub-PFPs data in this on-chip shared memory. On the other hand, the corresponding occupancy of the GPU cores may sharply decrease according to the size of the shared memory used for the input object and sub-PFPs data, which means that there might be a tradeoff between them. Accordingly, here in this article, the maximum space of the shared memory for storing the input object and sub-PFPs data has been limited to 12,288 bytes for maintaining the designed 100% occupancy of the GPU cores.

6 Conclusions

In this article, we have implemented 1-D N-LUT on the GTX 690 GPU board by using three types of memory managing techniques. Experiments with test 3-D objects show that the average CGH calculation time for one object point of the implemented system has been calculated to be 0.046 ms, which means that it could generate almost 3 frames of CGHs with 1920×1080 pixels per second for the 3-D object with 8000 object points. Experimental results finally confirm the feasibility of the proposed system in the practical applications.

Acknowledgments

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIP) (No. 2013-067321). This work was partly supported by the IT R&D program of MSIP/MOTIE/KEIT (10039169, Development of Core Technologies for Digital Holographic 3-D Display and Printing System). The present Research has been conducted by the Research Grant of Kwangwoon University in 2014.

References

1. M. Lucente, "Interactive computation of holograms using a look-up table," *J. Electron. Imaging* **2**(1), 28–34 (1993).
2. S.-C. Kim and E.-S. Kim, "Effective generation of digital holograms of 3-D objects using a novel look-up table method," *Appl. Opt.* **47**, D55–D62 (2008).
3. 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).
4. S.-C. Kim and E.-S. Kim, "Fast computation of hologram patterns of a 3-D object using run-length encoding and novel look-up table methods," *Appl. Opt.* **48**, 1030–1041 (2009).
5. S.-C. Kim, W.-Y. Choe, and E.-S. Kim, "Accelerated computation of hologram patterns by use of interline redundancy of 3-D object images," *Opt. Eng.* **50**(9), 091305 (2011).
6. S.-C. Kim, K.-D. Na, and E.-S. Kim, "Accelerated computation of computer-generated holograms of a 3-D object with $N \times N$-point principle fringe patterns in the novel look-up table method," *Opt. Laser Eng.* **51**, 185–196 (2013).
7. 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).
8. S.-C. Kim et al., "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).
9. 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).
10. Z. Ali et al., "Simplified novel look-up table method using compute unified device architecture," *3D Res.* **2**, 1–5 (2011).
11. X. Xu et al., "Computer-generated holography for dynamic display of 3D objects with full parallax," *Int. J. Virtual Reality* **8**, 33–38 (2009).
12. Y. Pan et al., "Fast CGH computation using S-LUT on GPU," *Opt. Express* **17**, 18543–18555 (2009).
13. Y. Liu et al., "A fast analytical algorithm for generating CGH of 3D scene," *Proc. SPIE* **7619**, 76190N (2010).
14. Y.-Z. Liu et al., "High-speed full analytical holographic computations for true-life scenes," *Opt. Express* **18**, 3345–3351 (2010).
15. T. Shimobaba et al., "Rapid calculation algorithm of Fresnel computer-generated-hologram using look-up table and wavefront-recording plane methods for three-dimensional display," *Opt. Express* **18**, 19504–19509 (2010).
16. J. Weng et al., "Generation of real-time large computer generated hologram using wavefront recording method," *Opt. Express* **20**, 4018–4023 (2012).
17. NVIDIA GeForce series GTX690, GTX580, GTS450, GT 640, http://www.nvidia.com/object/geforce_family.html.
18. NVIDIA Quadro6000, http://www.nvidia.com/object/quadro.html.
19. NVIDIA Tesla K10, http://www.nvidia.com/object/tesla-supercomputing-solutions.html.
20. W. Hwu and D. Kirk, "CUDA memories," Chapter 5 in *Programming Massively Parallel Processors: A Hands-On Approach*, pp. 77–93, Morgan Kaufmann, San Francisco (2010).
21. T. Shimobada et al., "Fast calculation of computer-generated-hologram on AMD HD5000 series GPU and OpenCL," *Opt. Express* **18**, 9955–9960 (2010).
22. N. Takada et al., "Fast high-resolution computer-generated hologram computation using multiple graphics processing unit cluster system," *Appl. Opt.* **51**, 7303–7307 (2012).
23. NVIDIA CUDA Occupancy Calculator, http://developer.download.nvidia.com/compute/cuda/3_1/sdk/docs/CUDA_Occupancy_calculator.xls.
24. Private communications with a CUDA specialist (Dr. H.-G. Ryu) working at NVIDIA, Korea.

Min-Woo Kwon received his BS degree from Kwangwoon University, Seoul, Republic of Korea, in 2006, and his MS degree from the Department of Electronics Convergence Engineering of the Graduate School of Kwangwoon University in 2008. Since 2012, he has been a PhD of the 3DRC (3D Display Research Center) and HoloDigilog (Holodigilog Human Media Research Center) of Kwangwoon University. His research interests include 3D imaging and display, holography, and optical information processing.

Seung-Cheol Kim received his BS degree from Kwangwoon University, Seoul, Republic of Korea, in 2002, and his MS and PhD degrees in electronic engineering from the Graduate School of Kwangwoon University, in 2004 and 2007, respectively. Since 2007, he has been a research professor of the 3DRC (3D Display Research Center) and HoloDigilog (Holodigilog Human Media Research Center) of Kwangwoon University. His research interests include 3D imaging and display, holography, and optical information processing.

Eun-Soo Kim is a professor in the Department of Electronics Engineering at Kwangwoon University in Seoul, Republic of Korea, and the director of the 3D Display Research Center and the director of the HoloDigilog Human Media Research Center. In 1984, he received his PhD degree in electronics from Yonsei University, Seoul. He was a visiting professor at California Institute of Technology from 1987 to1988. His research interests include 3-D imaging and displays, 3-D fusion technologies, and their applications.