DIGITAL PHASE CONJUGATION AND REAL-TIME 3D OBJECT GENERATION WITH MEMS PHASE LIGHT MODULATOR

by

Yefu Zhang

A Thesis Submitted to the Faculty of the

JAMES C. WYANT COLLEGE OF OPTICAL SCIENCES

In Partial Fulfillment of the Requirements

For the Degree of

MASTER OF SCIENCE

In the Graduate College

THE UNIVERSITY OF ARIZONA

2024

THE UNIVERSITY OF ARIZONA
GRADUATE COLLEGE

As members of the Master's Committee, we certify that we have read the thesis prepared by **Yefu Zhang**, titled ***Digital Phase Conjugation and Real-time 3D Object Generation with MEMS Phase Light Modulator,*** and recommend that it be accepted as fulfilling the dissertation requirement for the Master's Degree.

Yuzuru Takashima
Professor Yuzuru Takashima
Date: June 24, 2024

Tom D Milster
Professor Thomas D. Milster
Date: Jun 26, 2024

Yushi Kaneda
Professor Yushi Kaneda
Date: Jul 7, 2024

Final approval and acceptance of this thesis is contingent upon the candidate's submission of the final copies of the thesis to the Graduate College.

I hereby certify that I have read this thesis prepared under my direction and recommend that it be accepted as fulfilling the Master's requirement.

Yuzuru Takashima
Professor Yuzuru Takashima
Master's Thesis Committee Chair
Wyant College of Optical Sciences
Date: June 24, 2024

Signature: Tom D Milster (Jun 26, 2024 19:23 PDT)
Email: milster@optics.arizona.edu

Signature:
Email: ykaneda@arizona.edu

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

An ideal Near-to-Eye display (NED) requires high-resolution images, a large field of view (FOV) and depth cues. Sometimes, those performances are degraded due to optical aberrations of optics. In the first project, to correct for aberrations, digital phase conjugation (DPC) was demonstrated with a Texas Instruments phase light modulator (TI-PLM) for the first time to generate a tightly focused point image over an aberrated TIR/geometrical image guide.

We measured aberration of the image guide combiner by off-axis holography that captures the off-axis fringes using a camera sensor. Subsequently, image processing on the captured fringes, involving Fourier transform and cropping of +1st order, to extract the final field information while reducing low-frequency noise. Computer-Generated Hologram (CGH) was generated to negate the phase aberration, which is then displayed on the PLM.

Through phase conjugation, the spherical and un-aberrated wavefront was reconstructed over an aberrated optical medium, resulting in a series of point sources displayed at different depths, and producing a 3D point images. This method can be utilized to generate multiple point sources with different depths, contributing to the 3D image in Near-to-Eye display even via aberrated medium.

As a second part of the thesis, a real time pipe line towards holographic 3D head up display (HUD) for Advanced Driver Assistance System (ADAS) was pursued. A holographic 3D HUD provides an integrated environment for vehicle drivers by sensing the situation around the vehicle, by extracting significant and critical information from the sensor data, such as traffic sign, preceding traffics, and even invisible obstacles, and notifying the driver.

To achieve the pipeline for a 3D holographic Head-Up Display (HUD) we addressed three primary challenges. Firstly, image processing for road lane recognition was accomplished using

OpenCV from sensor inputs. Then, to ensure a latency-free display, CGH was calculated using GPU in CUDA mode. Lastly, based on the detected road lanes, the optical overlay of the 3D image and road lane was achieved through a first-order optical model. This approach allows the 3D image to be displayed over the road lane at the same size and depth, providing the driver with direction guidance based on navigation data. The speed of CGH calculations was compared across three platforms: CPU-based, GPU-based, and cloud-based computing.

The development of a 3D holographic Display using TI-PLM addresses significant challenges in NED and ADAS. By leveraging DPC and CGH, the thesis demonstrates the ability to correct optical aberrations and produce high-resolution, depth-cued 3D images. The pipeline developed for a holographic HUD integrates real-time sensor data, ensuring latency-free display and accurate overlay of road lane. Through GPU-based processing, the project achieves efficient CGH generation, overcoming computational challenges associated with complex 3D objects. The application of this technology in the automotive industry holds potential for enhancing driver safety and navigation by providing immediate and intuitive visual information. Overall, the integration of real-time CGH with HUD systems promises to revolutionize the conveyance of critical information, leading to safer and more efficient operations.

CHAPTER 1 | INTRODUCTION

1.1 3D Display Technology

Display technology is always keeping developing with the development of human community. Whether it is black and white display technology or today's LCD and projection technology, display technology is always being updated to achieve more realistic scenes. 3D display technology is a goal that people have been pursuing since our real world is three dimensional. Nowadays, more and more technologies are used for 3D display.

Binocular disparity is one of the traditional methods to achieve 3D display [2]. This technology is based on the different object location seen by the left and right eyes (Figure 1). Since there is no reconstruction of the object amplitude and phase information is employed, this kind of technologies are not 3D displays in the true sense. Binocular parallax display uses polarization, time-sharing or grating to input different images with certain visual differences to the observer's eyes. The two images are fused in the brain to form a certain depth perception. For observers, the 3D image is in front of or behind the screen, but the human eyes need to focus on the screen, which can easily cause fatigue and discomfort to the human eyes because of the vergence accommodation conflicts [1].

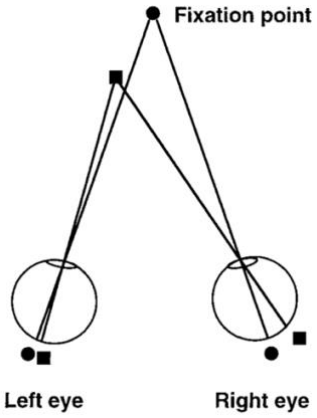


Figure 1. 3D display based on binocular disparity technology [2].

Volumetric display technology reconstructs a three-dimensional light field based on pixels in three-dimensional space. Currently, it generally scans one luminous point or luminous surface quickly, faster than the flicker frequency of human eye, 20Hz. Due to the persistence of vision of the human eye, 3D vision will be produced as show in Figure 2. The volume display has all the depth elements and can be viewed at 360°, but the current scanning speed is still difficult to achieve large-size, high-resolution 3D display. DepthCube solid-state 3D volumetric display is a typical represent for this technology. It is a solid state, rear projection, volumetric display that consists of two main components: a high-speed video projector, and a multiplanar optical element composed of an air-spaced stack of liquid crystal scattering shutters [3].

Figure 2. 3D object Reconstruction through volumetric display technology [3].

Integral Imaging is based on the principle of refraction and uses a two-dimensional microlens array to record and reconstruct 3D object. As shown in Figure 3, during the recording process, it records the three-dimensional information of the object through the microlens array. Note that, the barriers physically or optically between them are to avoid the overlapping between

neighboring elemental images. Since each lens element records a part of the scene information in different directions, each lens element generated the image elements correspond to the parallax information of different viewing angles. During reconstruction, the ray bundles produced by the pixels of the display intersect at the same region of the reconstruction space. After the intersection, the bundles continue to propagate toward the observer. Then what the observer receives is a diverging ray beam that is equivalent to the one produced by a real point source on the object [4].

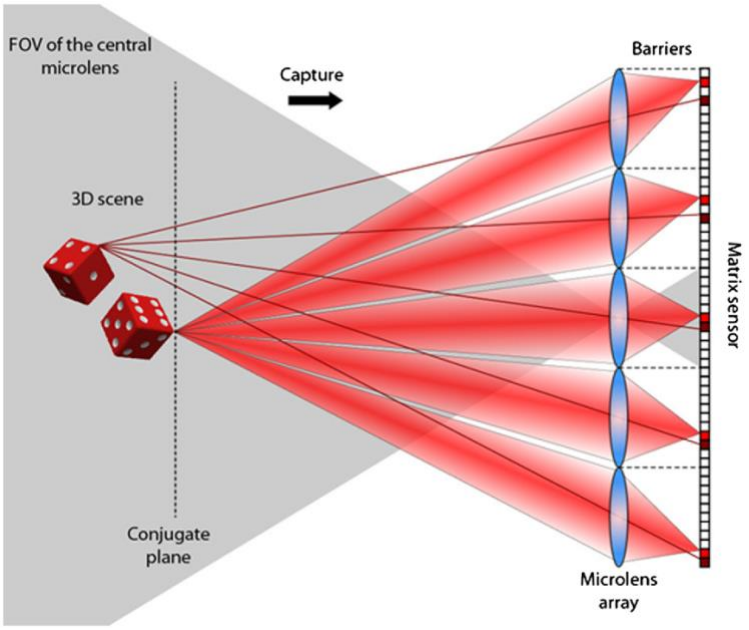


Figure 3. Recording and reconstruction with microlens array by integral imaging [4].

1.2 Holographic 3D Display

The information of light emitted by a 3D object mainly includes phase and amplitude. However, traditional flat-panel display technology can only reproduce the amplitude of the light field, but not the phase, losing the three-dimensional information of the object. Holographic technology includes both amplitude information and phase information of the object beam. The display based on holographic technology can completely restore all the information of the original object beam, provide all the elements required for human stereoscopic vision, and produce a

stereoscopic image reconstruction that is like coming from a natural scene. It is one of the most promising 3D display technologies.

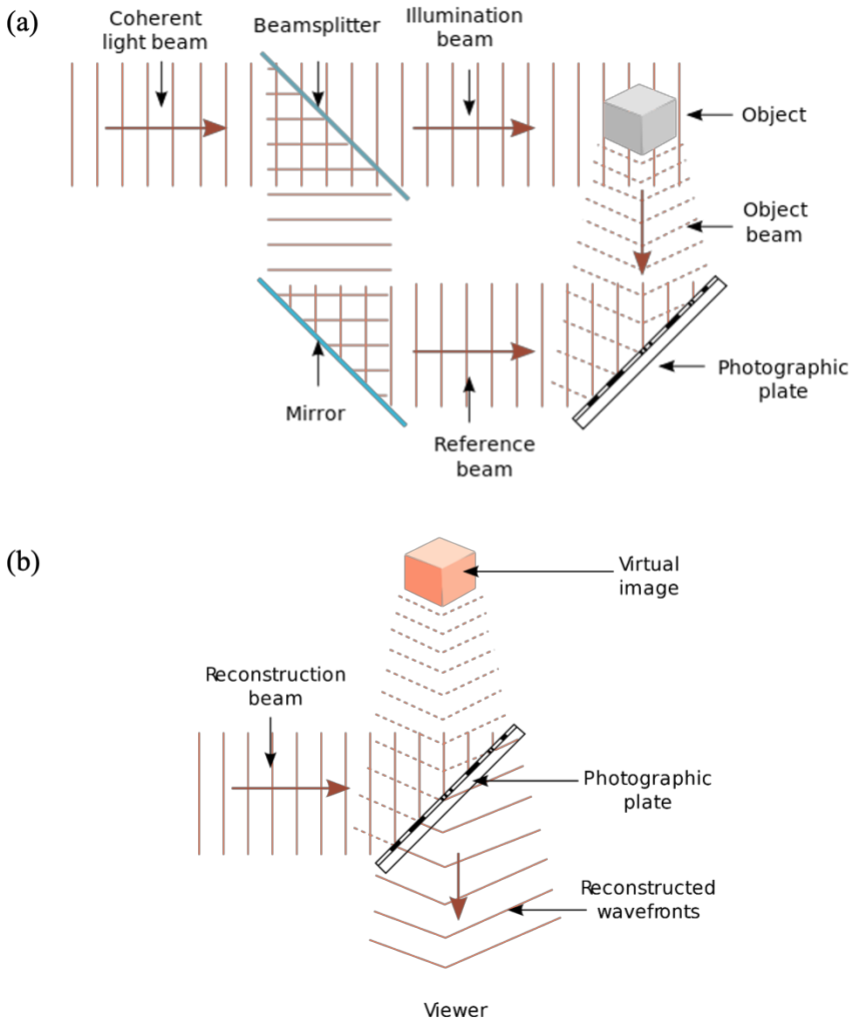


Figure 4. The principle of holography: (a) recording, (b) reconstruction

Early optical holography used light interference and diffraction respectively to record and reconstruct the amplitude and phase information of the original object beam (Figure 4). The recording of holograms is achieved through optical interference. For a 3D object, a coherent light source is used to illuminate the object to generate an object beam. A beam of light that is coherent with the object beam is used as a reference beam. The two beams of light intersect and superimpose on the recording material to produce interference fringes which contain the amplitude and phase

information of object beam. During reconstruction of object beam, the hologram is illuminated with the original reference light. The fringes of the hologram can be regarded as the superposition of multiple gratings. When the illumination light wave is incident on the recorded hologram, grating diffraction occurs, and then the original object beam is restored, so that 3D objects can be observed.

The complex amplitudes of the object beam and the reference beam are $O(x,y)$ and $R(x,y)$ respectively, as shown in Eq. (1), and Eq. (2).

$$O(x,y) = |o(x,y)| \exp\left(-j\varphi(x,y)\right) \tag{1}$$

$$R(x,y) = |r(x,y)| \exp\left(-j\phi(x,y)\right) \tag{2}$$

The light field intensity after the interference of the two beams is shown in Eq. (3).

$$I(x,y) = |O(x,y) + R(x,y)|^2$$
$$= |R(x,y)|^2 + |O(x,y)|^2 + O(x,y)^* R(x,y) + R(x,y)^* O(x,y) \tag{3}$$

The fourth term contains the complex amplitude information of the original object beam, and the third term represents the conjugate of the object beam. During reconstruction, the reference beam R(x, y) is used to illuminate the hologram, and the reconstruction beam is as shown in Eq. (4).

$$I(x,y)R(x,y) = (|R(x,y)|^2 + |O(x,y)|^2)R + O(x,y)^* R^2(x,y) + R(x,y)^* R(x,y) O(x,y) \tag{4}$$

The fourth term: $R(x,y)^* R(x,y) O(x,y) = |R(x,y)|^2 O(x,y)$. If the reference beam we use is collimated parallel light, $R(x,y) = C \exp\left(-j\phi(x,y)\right)$, where C is a constant value, then

the fourth term becomes $C^2O(x,y)$, which is exactly the perfect reconstruction of the original object beam.

D. Gabor invented holographic technology in 1947 [5], for which he was awarded the Nobel Prize in Physics. Initially, electron waves were used for recording, and then D. Gabor replaced the electron waves with a mercury (Hg) light source. However, the coherence of mercury light sources is still not good enough. Moreover, in-line holography was used at that time. The directions of the reference beam and the object beam are both on the optical axis, resulting in the inability to separate the original beam and its conjugate beam, so the application of early holography was limited. With the invention of laser light sources with high coherence, holographic technology has ushered in rapid progress. Soon, in 1962, shortly after the laser invention, off-axis holography [6] was discovered by Leith and Upatnieks. This method uses an inclined reference beam so that the original object beam image and the conjugate object beam image are on both sides of the optical axis during reconstruction. Nowadays, static holographic technology is relatively mature. Zebra Technologies can realize color holograms, whose size and field of view can be very large. Figure 5 is a 3D holographic map produced by the company [37]. However, the production time of a hologram of the company lasts from tens of minutes to several hours, and it cannot be used for dynamic display.

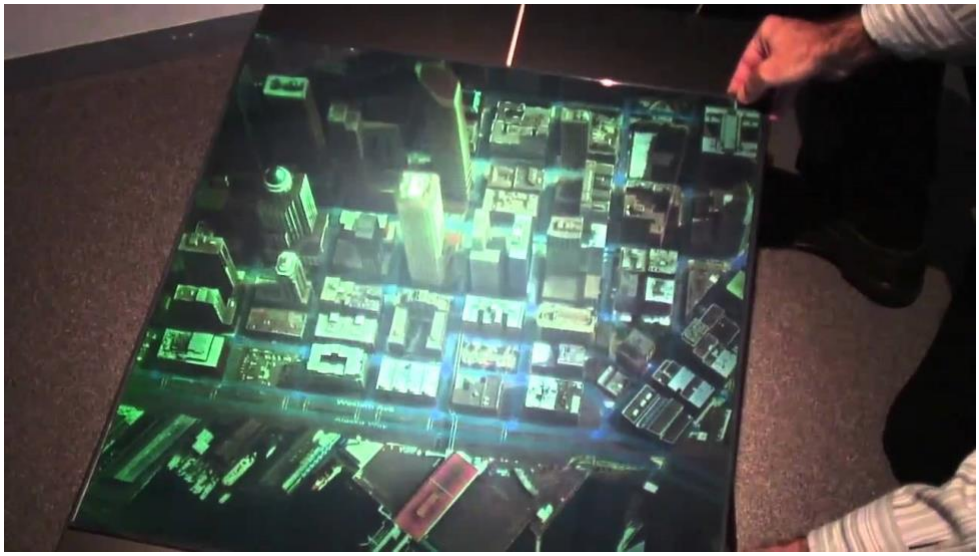


Figure 5. 3D holographic map produced by Zebra Technologies [37].

Holograms recorded by traditional methods can basically only realize the reconstruction of static images rather than dynamic images. To achieve dynamic display, digital holograms are generated through computer generated holography and input into Spatial Light Modulator (SLM), a refreshable light modulation device, thereby generating a dynamically refreshed light field [7]. Computational holography technology was first proposed by Lohmann et al. in 1967 [8]. Computer-generated hologram (CGH) is calculated by computer without the need for optical recording. SLM is a type of light modulation device that can be dynamically refreshed. It can modulate the incident light according to the input CGH and reconstruct the wavefront of the object. SLM can conveniently and quickly load CGHs and reproduce holographic 3D images to achieve dynamic holographic 3D display.

Compared with static holography, dynamic holographic 3D display still has many problems, such as the slow calculation speed of CGH, the relatively low resolution and size of SLM, etc. However, dynamic display is undoubtedly the future development direction of holography. It does not require a complicated optical recording process. It only requires computer calculations to obtain CGH of the 3D object, and the generated hologram is digital, which is very

convenient and fast, and the cost is relatively low. Both real objects and virtual objects can be reconstructed. Interaction between real images and viewers can be achieved. Therefore, dynamic holographic technology has broad prospects in many application fields and is an inevitable trend in future 3D display technology.

1.3 Computer-Generated Holography

Different from traditional optical holography, computer-generated holography does not require a strict and stable recording environment. Instead, it performs calculation of light field propagation and forming interference pattern in the computer, so it has a rich variety of recordable objects. The information of 3D objects is usually described by many points, lines, or surfaces. We regard these point-based, line-based, or surface-based objects as light sources. By superimposing the complex amplitudes of the light fields emitted by all units when they propagate to the hologram surface, the wavefront of the object beam of the 3D object on the hologram surface can be obtained.

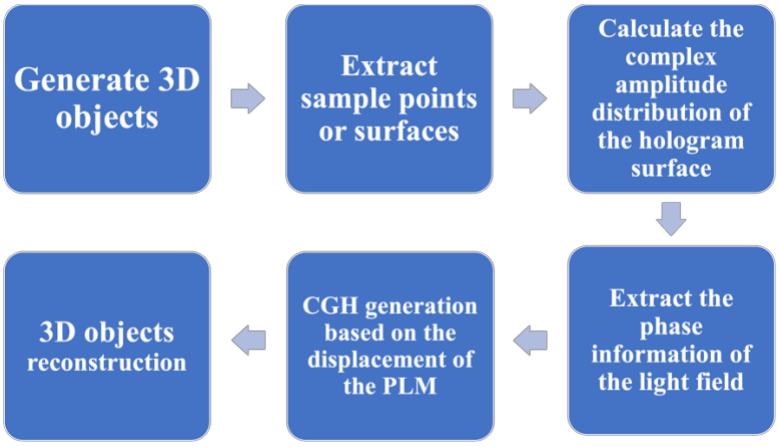


Figure 6. Flow chart of computer-generated holography

The overall process of making CGH is shown in

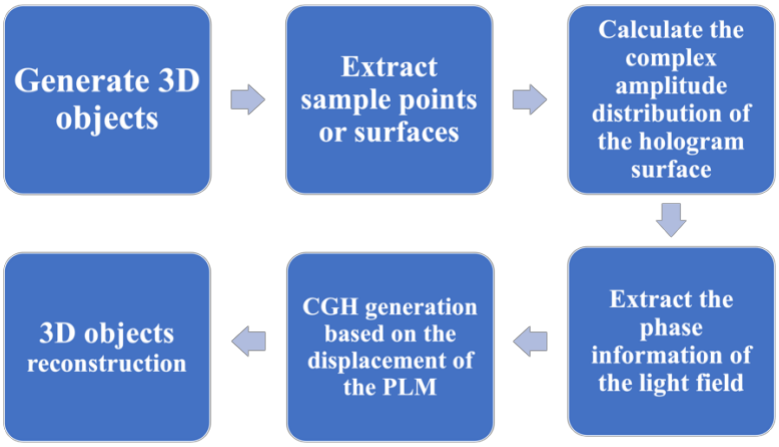


Figure 6. The specific calculation steps are as follows:

a. Generate virtual objects. For objects that exist in reality, information can be obtained through cameras, LIDAR system, etc. and then input into the computer. For virtual objects that do not exist, information can be input directly in the form of functions or through computer graphics.

b. Extract sample points or surfaces. By sampling continuous objects, a discrete sample point distribution is obtained that is easy for computers to process.

c. Calculate the complex amplitude distribution of the hologram surface. Simulate light field propagation in the computer to obtain the complex amplitude distribution on the hologram surface.

d. Extract the phase information of the light field. Phase information can be extract through Python Numpy angle() function.

e. CGH generation based on the displacement of the PLM. For the TI-PLM used in these experiments, a total of 16 displacements are considered. The phase must be aligned with these displacements to generate the CGH.

f. 3D objects reconstruction. Upload generated CGH on PLM to achieve 3D object reconstruction.

Point-based hologram method is shown in Figure 7. The 3D object is composed of N luminous points, and each luminous point is an ideal point light source. The wavefront of the object at the hologram surface is formed by the superposition of the spherical wavefronts emitted by all luminous points. This method is flexible and is currently the most widely used method [10, 11]. However, as the number of sampling points increases, the calculation amount of this method increases linearly, and the calculation time becomes very large.

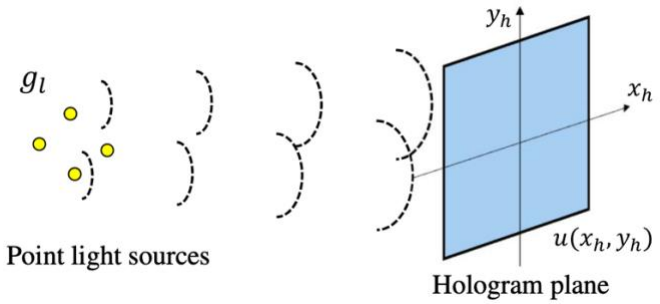


Figure 7. Point-based hologram generation [9]

The layer-based or multiplane method [12-14] discretizes 3D objects into luminous object planes, and each object plane emits an optical field to the hologram plane. The complex amplitude distribution at the hologram plane is superimposed by the optical fields propagating from the object plane to the hologram plane. For the multiplane method, the calculation of the complex amplitude of the hologram plane can be accelerated using the Fast Fourier Transform (FFT), which can greatly improve the calculation speed.

In addition, according to the modulation method of light, CGH can be divided into amplitude-type CGH and phase-type CGH, which are used in amplitude-type and phase-type SLM respectively. Because phase modulation devices are more energy efficient, phase-type CGH are more widely used.

CGH is more convenient and faster than traditional optical recording methods and is a key technology for dynamic holographic display. However, there are still some challenges. On the one hand, the calculation speed of holograms is still not fast enough, and new algorithms and hardware need to be developed to further accelerate hologram calculations. On the other hand, phase-type CGH lose part of the amplitude information of the object wavefront, making the reconstructed image quality is not high, and there are problems such as distortion and noise.

1.4 High-speed Calculation of CGH

The calculation load of CGH is very large, and the computing speed of current computers cannot meet the huge calculation load of dynamic holography. For example, for a 3D object, if 1024 points are taken using the point-based method and the number of CGH sampling points is 1920×1024, the time required will be 59s with Intel Core 2 Quad Q6600 computer CPU [11].

In order to achieve fast calculation of holograms, many researchers have studied this and proposed various solutions. From software to hardware, improve computing speed.

In terms of software and algorithms, one of the more widely used methods is the look-up table (LUT) method [15]. This method discretizes 3D space into points, and any 3D object can be composed of some of these points. The wavefronts of the spherical waves emitted by all points at the hologram plane are calculated in advance and then stored in the memory. When calculating a 3D object, first determine the points that make up the 3D object, then search the wavefronts of these points in the memory and add them together to obtain the complex amplitude of the object beam, thereby obtaining the CGH. The CGH calculation process of this method is greatly simplified and the calculation speed is also greatly improved. However, the preparation of this method is relatively arduous, because it needs to store wavefront data of many points, which requires large memory space of the computing device. Some improved algorithms based on the

LUT method, such as N-LUT [16], C-LUT [17], and S-LUT [18], can reduce storage space and further increase calculation speed.

CGH computing is basically parallel computing, and a graphics processing unit (GPU) can be used for parallel computing to accelerate CGH computing. Based on GPU devices, the calculation of CGH can be accelerated by more than 1000 times under parallel computing [11].

1.5 Application of Holographic 3D Display in Augmented Reality

Augmented reality (AR) display technology can superimpose computer-generated virtual images with real environment scene images [19], allowing users to observe the real world and continuously obtain information output by the computer, thus assisting users in understanding the real world [20-22].

With the improvement of computer technology, AR has gradually entered the public vision and industry application from the laboratory theory, and can be applied in many fields. For example, in the medical field, it can help doctors better analyze and judge the condition and assist in surgery [23]. In the cultural field, it can help restore damaged scenic spots and historic sites. In addition, in the commercial field, games and entertainment and many other fields have very broad application prospects. In recent years, AR has become increasingly mature. Currently well-known products include Google's Google glass, ODG, Epson's Moverrio (Figure 8), Meta glass, and Microsoft's Hololens.

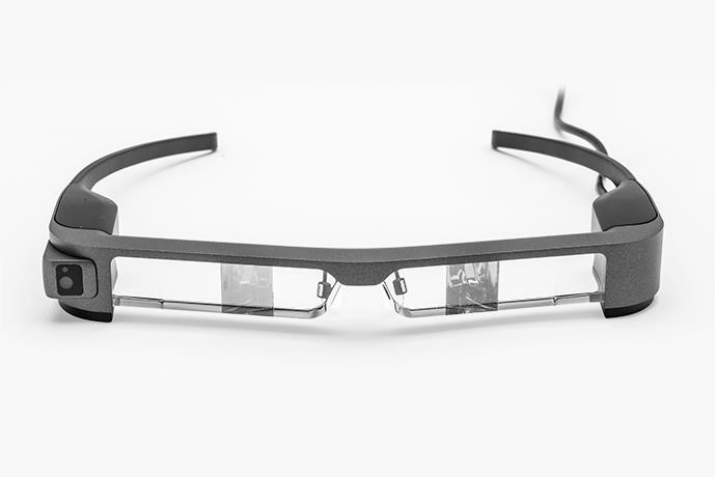


Figure 8. Epson's Moverio.

Google Glass was released in 2012. It has the structure of traditional glasses, is lighter in weight, and can be equipped with lenses for nearsightedness and farsightedness. The mechanism diagram of the glasses is shown in Figure 9 below. The semi-transparent mirror on the front of the glasses can deflect the internally projected image onto the retina of the human eye without affecting the entry of external light into the human eye, thereby realizing the function of augmented reality display. Google Glass has a short battery life, is expensive, and has a viewing angle of only 14.8 degrees, so it has not really opened the consumer market.

Figure 9. Google glass.

Current AR products either do not have 3D display capabilities and only display 2D images, such as Google glass, or they use binocular parallax technology for 3D imaging, such as Microsoft's HoloLens. For AR products that use binocular disparity technology, and long-term wear by observers will cause visual fatigue. To solve this problem, researchers proposed to combine true 3D display technology with AR technology. The true 3D display technologies used include integral imaging, volumetric display, holography, etc. Holographic 3D display is a 3D image that achieves continuous parallax. In theory, it can perfectly reconstruct real 3D scenes. When combined with AR technology, it can produce very good display effects.

Holographic AR display [24-26] loads the CGH generated by the computer onto the SLM, generates a 3D image under coherent light illumination, and then introduces it into the human eye through an optical coupling device that does not block ambient light, producing a holographic 3D image and AR effect with ambient light overlay.

1.6 The Research Work and Structure of Thesis

Holographic 3D display technology is theoretically the most advanced 3D display technology. By recording and reconstructing the phase and amplitude information of object beam, it fully reconstructs all the depth information and 3D models perceived by the human eye.

In this thesis, computer-generated holography is utilized for two distinct applications. The first application addresses aberration correction. With the advancement of AR/VR displays, aberrations introduced by optical elements can significantly affect display image quality. Although optical design processes consider aberrations, they may not efficiently correct aberrations in 3D displays. This thesis presents a method called digital phase conjugation for aberration correction. This method records aberrations at different image planes. By using the interference pattern of the object beam and reference beam, the wavefront information containing the aberration of the object

beam can be extracted through image processing. The same method is used to obtain aberrations at different image planes. To achieve a 3D near-eye display, reconstructions can be obtained at different depth cues with aberration compensation by applying the aberration phase information to the original CGH.

The second application focuses on holographic 3D displays for head-up displays. In this project, real-time CGH generation was achieved using GPU. Since point-based CGH calculations are parallel, they are suitable for GPU acceleration. Road lane recognition algorithms were used for this application. Based on the detected road lanes, point-based and line-based 3D objects are generated. The time consumption of CPU-based, GPU-based, and cloud-based methods is compared. The results show the overlap of the detected road lane and the 3D arrow image generated through point-light method (PLM).

The main structure of this thesis is as follows:

CHAPTER 1 | INTRODUCTION summarizes the research background of holographic 3D display and AR technology, introduces the development history of holographic display technology, and analyzes current problems and technical bottlenecks.

CHAPTER 2 | DIGITAL PHASE CONJUGATION presents research on the application of digital phase conjugation through CGH and PLM to achieve aberration correction introduced by AR display optical media.

CHAPTER 3 | REAL-TIME 3D OBJECT GENERATION focuses on real-time CGH generation for head-up displays. Different CGH calculation methods and road lane recognition algorithms are introduced. A comparison of different methods based on object type and calculation environment is also presented.

CHAPTER 4 | DISCUSSION and CHAPTER 5 | CONCLUSION provide the discussion and conclusion of this thesis.

CHAPTER 2 | DIGITAL PHASE CONJUGATION

2.1 Background and Motivation

A near-to-eye display necessitates images with high resolution, a large field of view, and effective depth cues. Often, quality of the displayed image is degraded due to optical aberrations of the image transfer medium. Such optical aberration affects the quality of the final image formed by incoherently, i.e., display panel illuminated by light emitting diode as a point spread function with aberration introduced. For coherently formed images, the quality of the image severely suffers from degradation, due to the deviation in the optical path assumed during calculation of CGH.

To solve this problem, we employ digital phase conjugation (DPC) technique that compensates for the phase error of image transfer device. The DPC process consists of measurement of the optical aberration of the image transfer medium with camera sensor, followed by displaying CGH on a PLM while considering the aberration measurement.

In this chapter, the methodology of off-axis holography to measure the phase aberration is presented. The off-axis interference pattern is then used for image processing to extract the on-axis phase information based on the theory of Fourier Transform [27]. The on-axis holography method was used first. However, there was no reconstruction achieved with the CGH produced by this method. With off-axis holography, the low frequency noise and the nonuniformity of the illumination can be easily removed. Using the extracted phase information, a CGH is generated and then displaced onto the PLM. The dynamic modulation capabilities of the PLM allow correction of the medium induced by subtracting the measured aberration term from the CGH. With this experiment, any aberration introduced by the optical medium can be easily removed and then improve the display quality.

Based on the final result of the experiment, most of the aberration generated by the optical medium (Moverio Image Guide) was successfully removed. The phase information of the fringes was calculated using the intensity detected by the camera sensor, which may introduce some error due to the sensor's limitations. To mitigate this error, intensity normalization was employed, as detailed in part 3 of this chapter. For further improvement, a wavefront sensor could be used to directly detect the phase information instead of inferring it from the intensity of the interference pattern.

2.2 System Setup for Capturing Phase Aberration and Phase Conjugate Reconstruction

The recording of phase aberration information is accomplished through off-axis holography. Figure 10 illustrates the configuration of the recording process. The flip mirror in the system, represented by a dashed line, indicates that the mirror is flipped off, allowing the light beam to pass through it. A 532nm laser and a Texas Instruments phase light modulator (TI-PLM) designed for visible wavelengths are employed.

Initially, the laser emits a 532nm light beam, which then passes through the half-wave plate (HWP). A beam expander is used to enlarge the beam size while keeping it collimated. After expansion, the laser beam is divided into object and reference beams by the polarizing beam splitter (PBS). The HWP adjusts the laser's polarization, and based on the PBS's polarizing direction, different polarizations result in different intensities for the object and reference beams. The intensity adjustment ensures that the interference pattern's contrast approaches 1 (perfect fringe contrast), reducing errors from the camera sensor, which may not have sufficient resolution if the maximum and minimum intensities are too close.

In Figure 10, the object beam first passes through a lens, focusing on its focal point located in the foreground of the Moverio Image Guide. During reconstruction, a virtual point image appears in front of the human eye to achieve the near-to-eye display. The image guide acts as the optical medium and introduces aberration. The purpose of introducing the lens after beam splitter (BS1) is reducing the object beam size to capture more frequency information. Another beam splitter (BS2) is positioned just before the PLM and camera sensor which make the aberrated object beam interferes with the off-axis reference beam. Mirror 1 (M1) is not placed at a 45-degree angle, generating the off-axis reference beam by applying a linear phase to the original collimated reference beam. The aperture after M1 crops the reference beam, producing fringes matching the dimensions of the PLM plane, ensuring the CGH size matches the PLM.

The spacing between the camera sensor and the BS2 is the same as the spacing between the PLM and the BS2, ensuring the CGH displayed through the PLM matches the on-axis fringes after image processing of the off-axis fringes captured by the camera sensor.

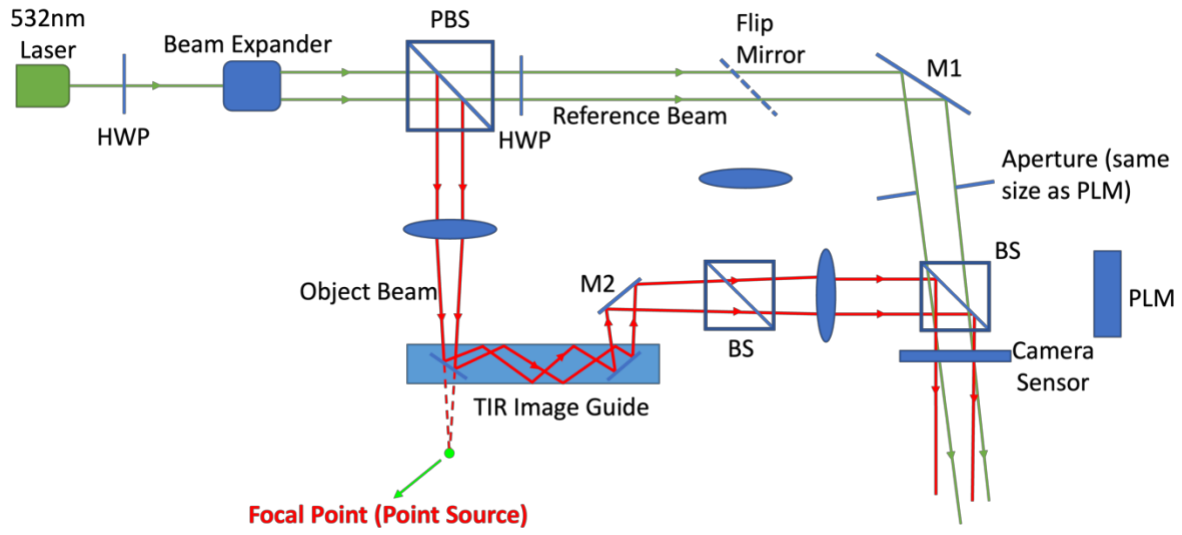


Figure 10. The schematic diagram of the off-axis interference pattern recording on the camera.

The off-axis holography offering the advantage of incorporating more frequency information while eliminating low-frequency noise through image processing, in contrast to the on-axis holography.

After the recording process, the off-axis fringes captured by the camera sensor undergo image processing. This processing step is crucial for extracting the desired on-axis phase information from the recorded fringes.

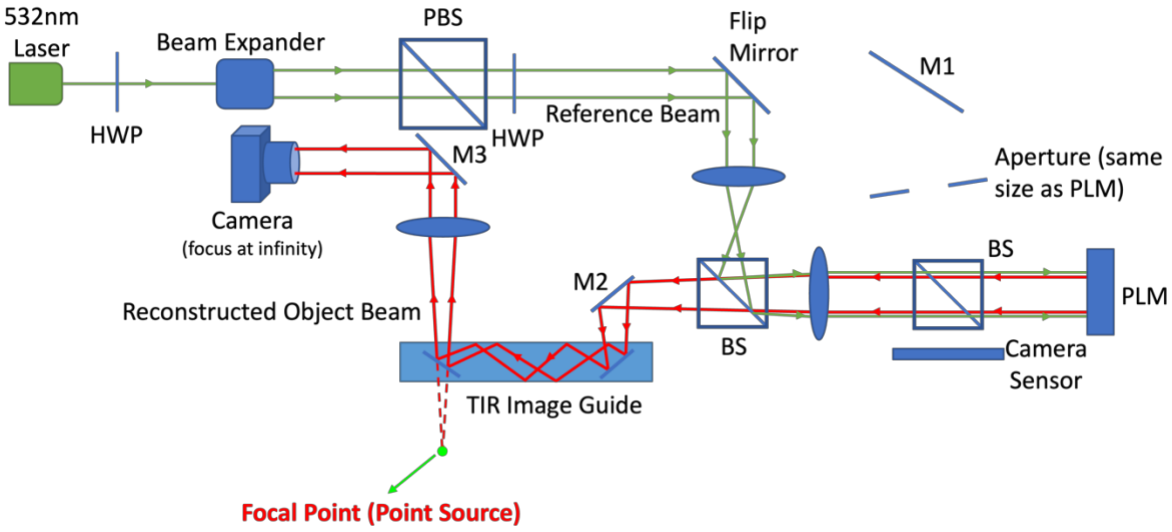


Figure 11. The schematic diagram of the holographic reconstruction with phase conjugation using PLM.

Figure 11 illustrates the configuration of the reconstruction process. Once the on-axis CGH is acquired, the flip mirror returns to its original position, providing an on-axis reference beam for the reconstruction process. At this point, the object beam is blocked to ensure that only the reference beam interacts with the PLM. The collimated reference beam then passes through the PLM, generating the reconstructed object beam loaded with the aberration information.

This reconstructed object beam, now carrying the corrected phase information, passes through the Total Internal Reflection (TIR) image guide to compensate the aberration. In theory, after passing through the image guide, the wavefront of the object beam should be a perfect

spherical wave originating from the virtual point source. This spherical wave then becomes a plane wave after passing through the subsequent lens.

To verify the accuracy of the reconstruction, additional components are introduced into the system. Mirror 3 (M3) directs the beam toward a camera. During the experiment, a piece of paper is placed at the location of the virtual point source plane. When the camera is focused at infinity, it images the superposition of the paper and the point source onto the camera sensor. This confirms that the aberration has been successfully compensated and that the reconstruction process is accurate. By ensuring that the reconstructed wavefront is an ideal spherical wave, and subsequently a plane wave, the system validates the effectiveness of the aberration correction. This process demonstrates the potential for holographic display technologies, particularly in applications requiring near-to-eye displays with a presence of optical aberration.

2.3 Image Processing with Fourier Transform

Image processing is crucial for obtaining the on-axis hologram necessary for reconstruction. This methodology is based on the work of Takeda and Ina [27], utilizing off-axis holography to obtain phase information through the Fourier Transform. Eq. (5) shows the form of the interference fringe pattern:

$$g(x,y) = a(x,y) + b(x,y)\cos\left[2\pi f_0 x + \phi(x,y)\right] \tag{5}$$

In this equation, $\phi(x,y)$ contains the desired phase information, while $a(x,y)$ and $b(x,y)$ represent unwanted irradiance variations caused by nonuniform light reflection or transmission by the test object. Additionally, $a(x,y)$ may contain low-frequency noise introduced by dust on the surfaces of optical elements. The term f_0 is the coefficient of a linear phase added to the original

phase information. Typically, the linear phase or tilted wavefronts are eliminated by setting the tilt to zero, resulting in the fringe pattern described in Eq. (6).

$$g(x,y) = a(x,y) + b(x,y)\cos\left[\phi(x,y)\right] \tag{6}$$

However, $a(x,y)$ and $b(x,y)$ still exist in the fringe pattern and are difficult to eliminate. Therefore, the Fourier Transform is introduced to address this problem.

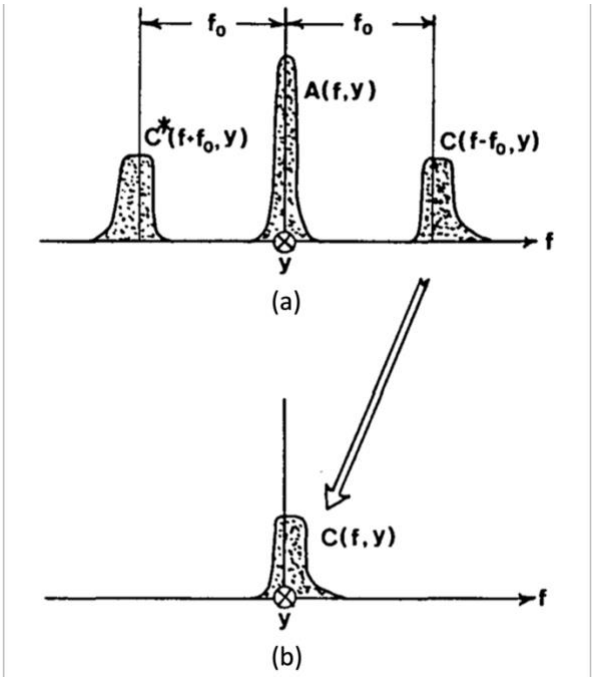


Figure 12. (a) The Fourier transformed interference fringe pattern. (b) Shifted +1st order to eliminate the f_0 fact [27].

Firstly, Eq. (5) is rewritten as Eq. (7). The cosine function is represented by two exponential functions using Euler's formula to simplify the Fourier Transform calculation:

$$g(x,y) = a(x,y) + c(x,y)\exp\left(2\pi i f_0 x\right) + c^*(\mathrm{x,y})\exp\left(-2\pi i f_0 x\right) \tag{7}$$

with

$$c(x,y) = \frac{1}{2}b(x,y)\exp\left[i\phi(x,y)\right] \tag{8}$$

Then, a Fourier Transform is applied with respect to x using a Fast Fourier Transform (FFT) algorithm, resulting in Eq. (9):

$$G(f,y) = A(f,y) + C(f - f_0, y) + C(f + f_0, y)^* \tag{9}$$

As shown in Figure 12a, the spectra in Eq. (9) are separated by f_0 in the frequency domain. Therefore, by cropping $C(f - f_0, y)$ or $C(f + f_0, y)^*$ and shifting it to the origin (0^{th} order), $C(f - f_0, y)$ will become $C(f, y)$ (Figure 12b). Again, when applying FFT to it, the phase information can be obtained without any linear phase introduced. This on-axis phase information can then be used for reconstruction.

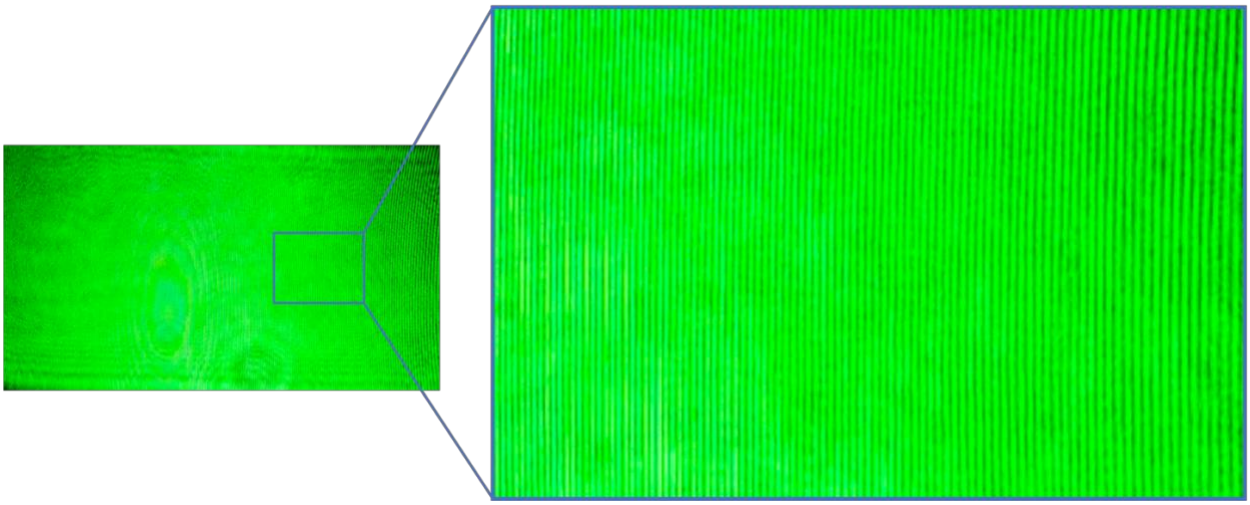


Figure 13. Off-axis interference pattern captured by camera sensor.

After discussing the theory, the experimental process will be detailed. During the recording process, the reference beam is set to off-axis mode, and the camera sensor captures the interference pattern. Starting with the off-axis interference pattern (Figure 13), FFT is applied to transform it into the frequency domain. In the frequency domain (Figure 14), a result similar to Figure 12a is obtained. Due to the linear phase (tilted reference beam), the 0th order, which contains low-frequency noise and the nonuniformity of the illumination, is successfully separated. In Figure 15,

the -1st order is subsequently cropped and shifted back to the origin (0th order) to derive the frequency domain of the on-axis hologram.

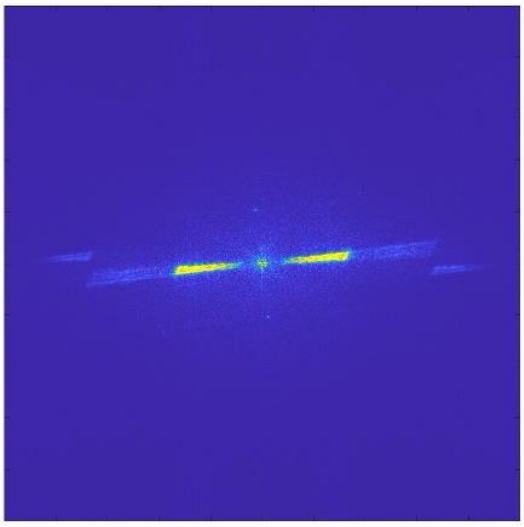


Figure 14. Frequency domain of the interference patter.

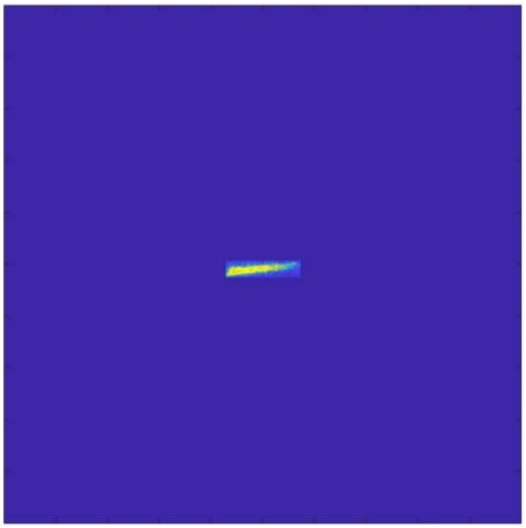


Figure 15. Shifted -1 order in frequency domain.

An Inverse Fast Fourier Transform (IFFT) algorithm is then employed to generate the on-axis hologram (Figure 16). The phase information corresponds to a range from 0 to 2π. There are in total 16 displacements of the PLM micromirror. Therefore, the continuously distributed phase must be mapped to the closest corresponding displacement of the micromirror. The generated CGH will then be uploaded onto the PLM to achieve the reconstruction.

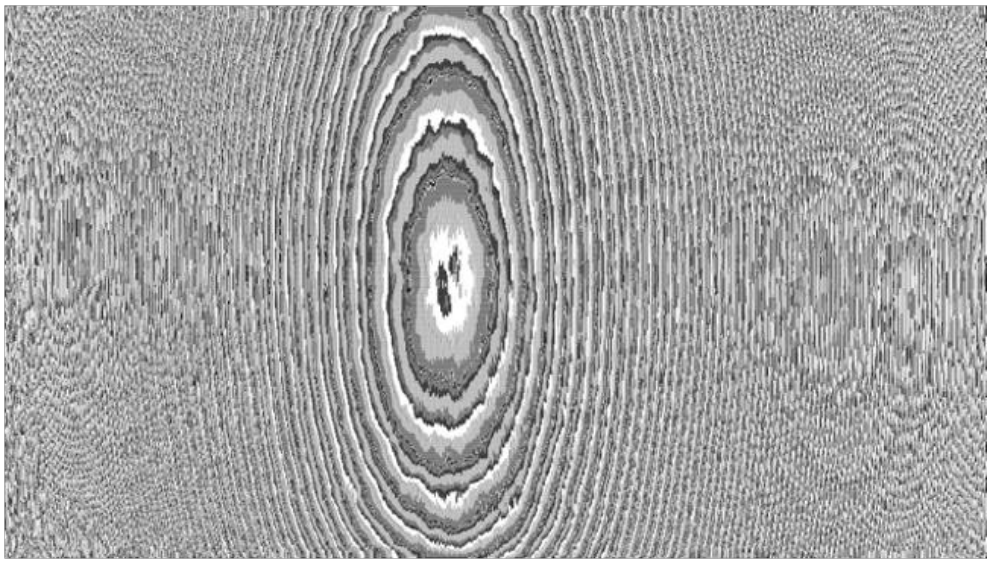


Figure 16. On-axis CGH obtained by inverse Fourier Transform.

2.4 Experiment Result

An image guide was adopted from commercially available AR glasses (Epson's Moverio BT200) which acts as the optical channel for image delivery and the images are formed at the foreground of the image guide. In general, the image guide is designed with projection optics and a Liquid Crystal on Silicon (LCoS) panel acting as the Spatial Light Modulator (SLM) (Figure 17a). However, this projection optics was removed for this experiment and used only the image guide to demonstrate optical aberration correction. This projection optics was used to compensate the astigmatism of the image guide. On doing so, we observed a prominent astigmatism of the image guide. To address this issue, phase conjugation technique was employed for aberration correction.

Figure 18a and Figure 18b show holographic projection of a point image by TI-PLM with image guide depicted in Figure 17b. TI-PLM is a microelectromechanical system comprising an array of micromirrors. These micromirrors move up and down in a piston motion to modulate the optical path length and so the phase. This phase array is specifically designed to diffract incident

light, making it ideal for holography applications [28]. By adjusting the voltage on the electrode, the PLM offers 16 levels and a maximum of 2π phase shift at 532 nm [29].

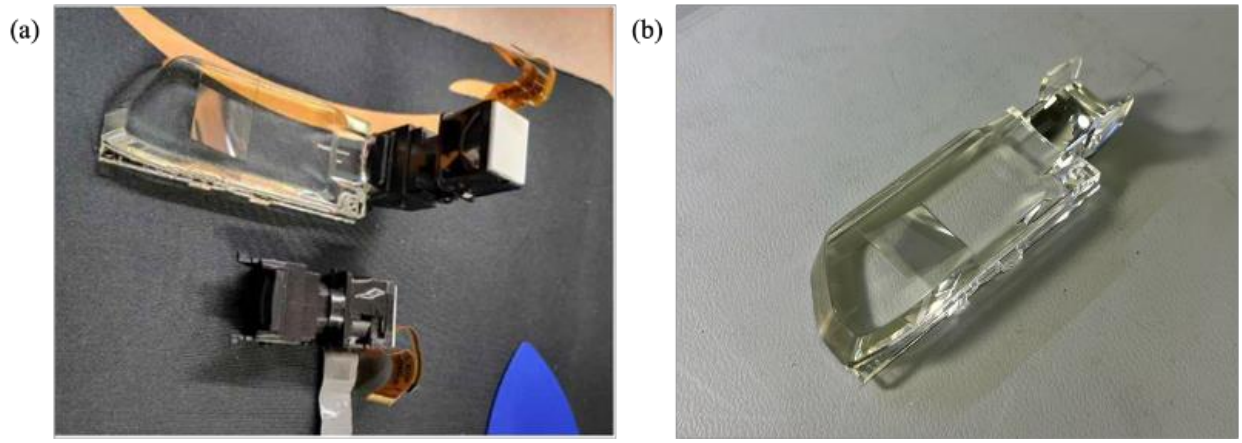


Figure 17. (a) Moverio Image guide disassembled with projection system attached and at the bottom is the projection system with LCoS panel. (b) Image guide without the projection optical system.

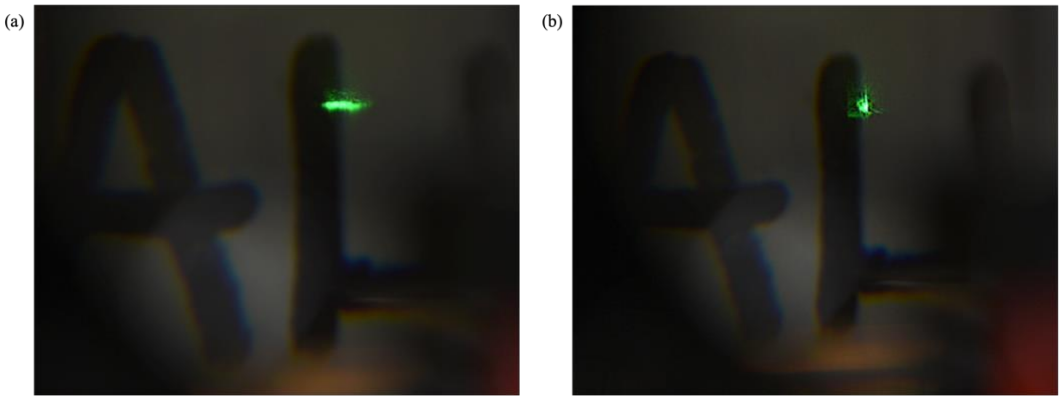


Figure 18. (a) Aberration introduced by image guide when PLM is turned off. (b) Aberration correction achieved through phase conjugation when PLM is turned on.

The astigmatism introduced by the image guide is apparent when plane wave is employed, or when PLM is off (Figure 18a). Upon activating PLM and uploading the CGH which loaded with the aberration of image guide and generated through image processing, the reconstruction of a single point source becomes evident (Figure 18b), indicating successful aberration compensation through phase conjugation. The background a word 'UA' on a cardboard serves as a reference plane for different focus positions. The letters ‘UA’ is positioned at the point source plane in front of the image guide (Figure 18).

Additionally, to achieve the display with depth cue, the simultaneous reconstruction of two point-sources is achieved by combining two distinct CGHs. As shown in Figure 10, the position of the point source can be easily relocated by shifting the lens after the PBS in the object beam arm. Therefore, we can generate two point-sources which are in front of and behind the image guide respectively. In Figure 19, two reference planes, denoted as 'U' and 'A,' host the reconstructions of the two distinct CGHs. The letter 'A' plane located in front of the image guide, whereas the letter 'U' plane positioned behind the image guide. The reconstructions of a single point source at different locations are presented in Figure 20. When the camera is focused at distinct planes ('U' and 'A') and the corresponding CGH is uploaded onto the PLM, it becomes evident that the single point source is reconstructed precisely at the selected plane.

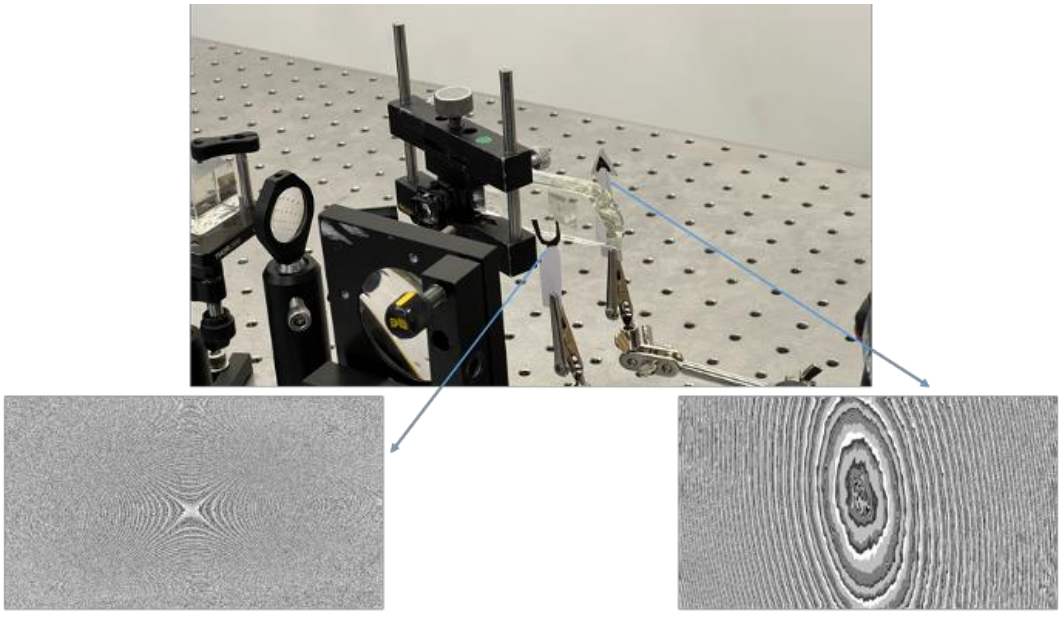


Figure 19. Two CGHs with constructions at the ‘U’ and ‘A’ planes.

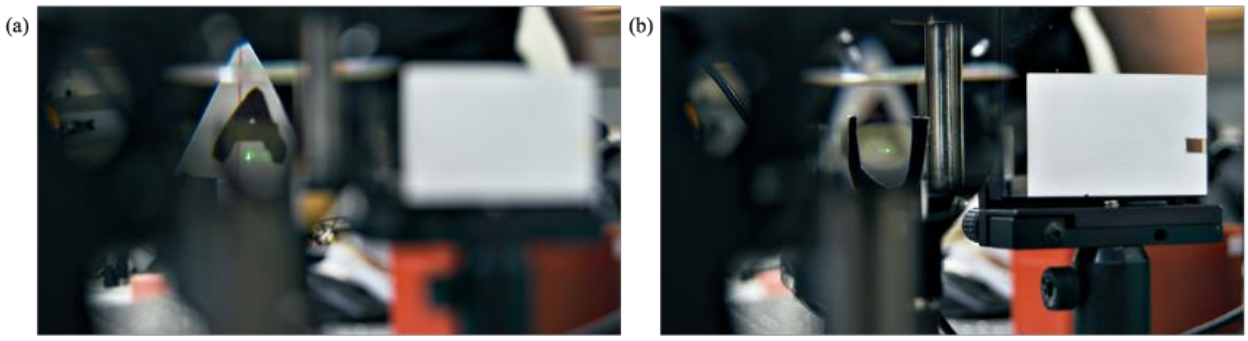


Figure 20. (a) Reconstruction of single point source at letter ‘A’ plane. (b) Reconstruction of single point source at letter ‘U’ plane.

By combining the optical fields of the two CGHs focused at the 'A' and 'U' planes as illustrated in Figure 19, a combined CGH is created, as depicted in Figure 21. The reconstruction of two points at the 'U' and 'A' planes is then accomplished by uploading the combined CGH onto the PLM (Figure 22). When the camera is focused at letter ‘A’ plane (Figure 22a), it results in the focused rendering of one point source, while the other becomes blurred. Subsequently, when the camera shifts its focus to the 'U' plane (Figure 22b), the scenario reverses, with one point source becoming blurred and the other coming into sharp focus.

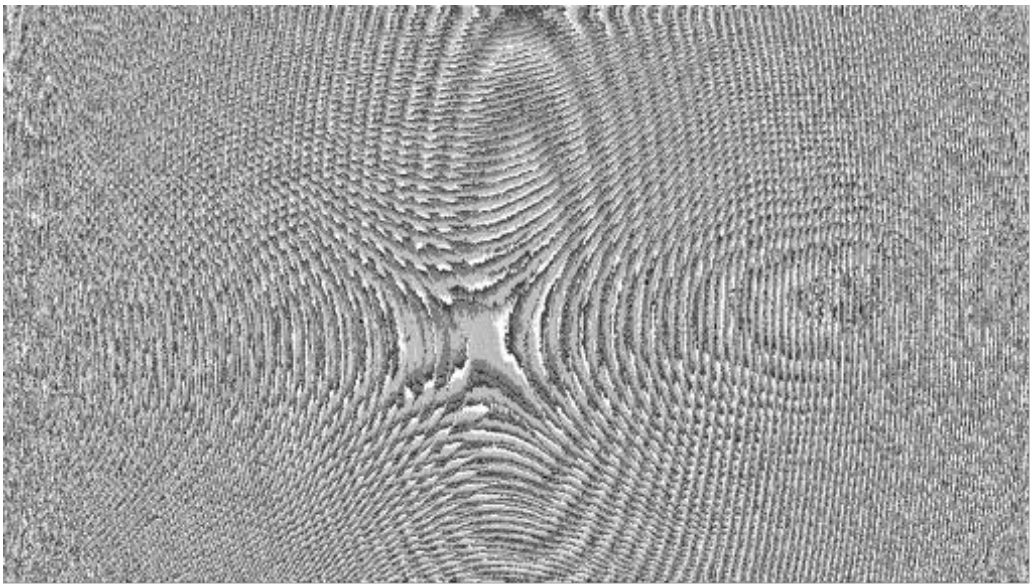


Figure 21. Combined CGH.

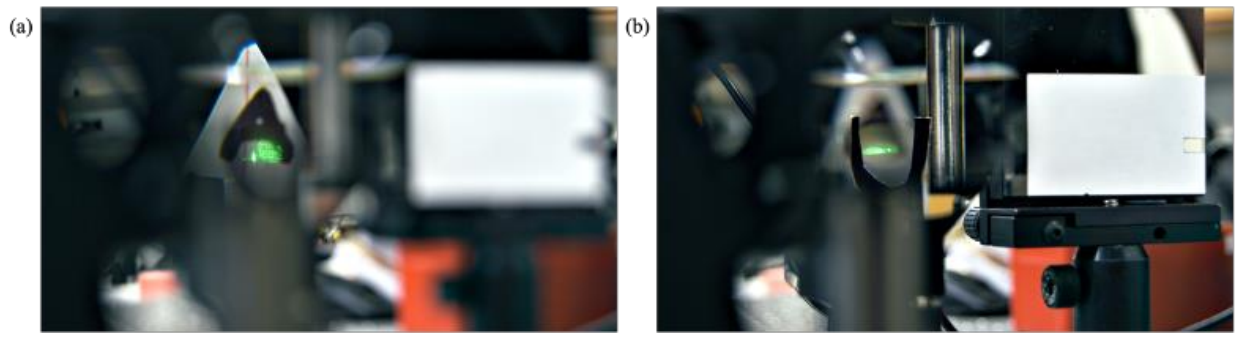


Figure 22. (a) The reconstruction of a point at the letter ‘A’ plane with a blurred image of another point source at the letter ‘U’. (b) The reconstruction of a point source at the letter ‘U’ plane with a blurred image of another point source at the letter ‘A’ verifying the reconstruction of two different depths.

2.5 Conclusion

Phase conjugation is demonstrated, and point images are reconstructed successfully over aberrated image transfer medium in this chapter. The process involves recording off-axis interference patterns with a camera sensor, followed by image processing using Fourier Transform. Subsequently, an on-axis CGH is generated and uploaded onto the Texas Instruments Phase Light Modulator (TI-PLM). We achieve distinct depth reconstructions of a single point source. A comparison between the CGH displayed on the TI-PLM and the TI-PLM in a flat state reveals that the aberrations introduced by the image guide object lens are corrected, leading to a sharp focus point. The aberration correction with digital phase conjugation is employed by using the TI-PLM as it offers high resolution, fast response time, and lower power consumption, making it suitable for NED devices. PLM offers multilevel phase diffraction patterns, achieving higher efficiency. Furthermore, we extend this methodology to simultaneously reconstruct two point-sources through the combination of CGHs. The promising potential next step involves combining multiple CGHs with varying reconstruction depths to generate 3D point cloud images.

CHAPTER 3 | REAL-TIME 3D OBJECT GENERATION

3.1 Background and Motivation

Advanced Driver Assistance System (ADAS) provides an integrated environment to driver of vehicle by sensing situation around the vehicle, extracting significant information from the sensor data, and notifying the significant information to the driver. Sensor for ADAS includes cameras, lidars, ultrasounds proximity sensor, and radar. Along with warning light and sounds signals, Head-up Display (HUD) displays information while superimposing the warning information on top of driver's see-through view. HUD usually employs flat panel displays with magnifying optics that produces 2-dimensional information. Displaying information by muti focal planes employing flat panel while switching optical path by scanning optics is proposed [30].

Holographic 3D image is an alternative to such 2D panel-based display for automotive HUDs [31]. In holographic HUD, Computer Generated Holograms (CGH) displayed on a Phase Light Modulator is commonly employed along with various CGH calculation algorithms [31]. In particular, Holographic HUD for ADAS requires displaying information in a real-time without latency, based on sensor input.

Recently a Micro Electro Mechanical System (MEMS) based Phase Light Modulator (PLM) that operates with kHz of refresh rate is available [32]. In this paper, we demonstrate pipeline of holographic HUD by employing MEMS PLM by displaying 3D image of extracted feature of the road by Open CV based on camera input, and calculation and displaying CGH based on camera input. In particular, we compared CGH calculation speed for three cases, CPU-based, GPU-based, and Cloud-based computing.

3.2 Image Processing of Road Lane Recognition

Figure 23 schematically shows schematic of the road lane recognition process and the generation of the 3D model based on the detected road lane. This algorithm is a common method used for the road lane recognition [33]. Based on the input video, the algorithm will process each frame and provide the output image overlap with the detected road lane. The code is based on Python and OpenCV package [34] is widely used for image processing.

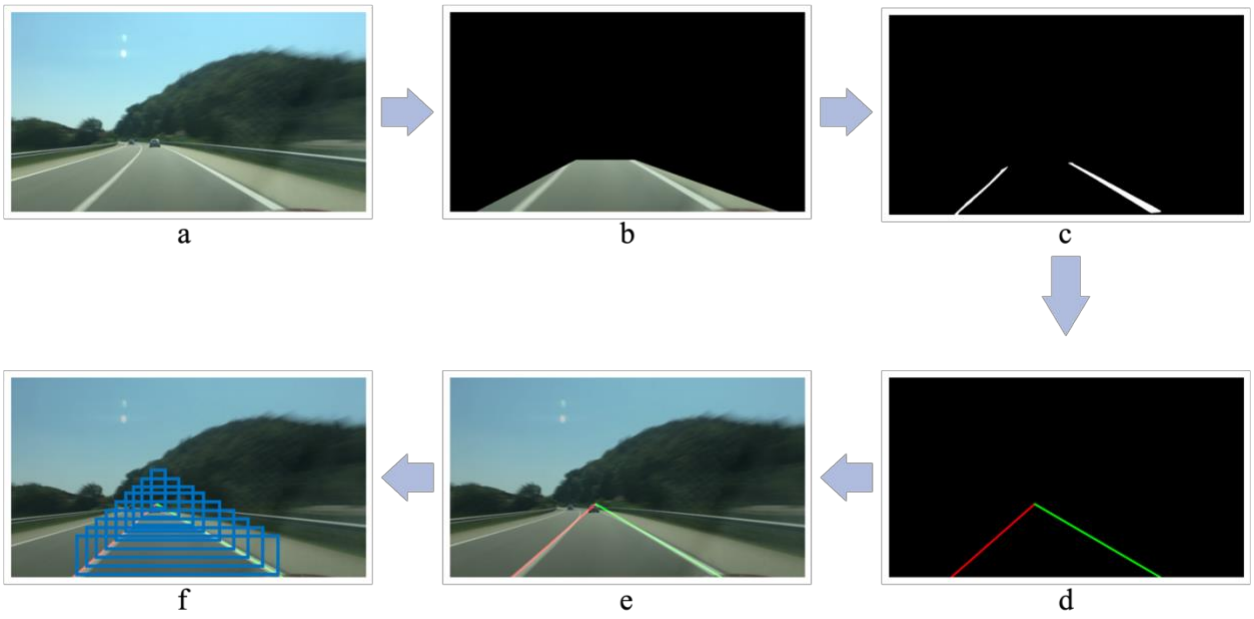


Figure 23. Schematic of road lane recognition process.

Beginning with one frame of the video (Figure 23a), the code defines the Region of Interest (ROI). According to the defined ROI, the image is cropped (Figure 23b). The cropped image is then converted from the RGB (Red, Green, Blue) color space to the HSV (Hue, Saturation, Value) color space. In the HSV space, the white and yellow regions are identified, and a color mask is applied (Figure 23c), as most road lanes are either white or yellow. After extracting the road lane, a bilateral filter is employed to smooth the image and reduce noise while preserving edges. Additionally, the Canny Edge Detection algorithm is used to detect the edges of the road lane. Based on the detected edges, the Hough Line Transform extracts the lines in the image and outputs

the start and end points of the lines (Figure 23d). These lines are then used to draw the road lanes, overlapping them with the original image (Figure 23e).

From the feature lines, the 3D coordinates of the center and edge lines are extracted based on the first order optical model, considering the focal length of the geometrical image combiner. By assuming that the camera in front of the car is focused at infinity and using its focal length, the actual size and distance of the lane can be determined by applying the Gaussian lens formula. The 3D image consists of elemental rectangular objects (Figure 23f) that are displayed at different depths, aligned with the center and edge lines. According to the rectangle elemental object, point cloud-based object and line-based object are generated.

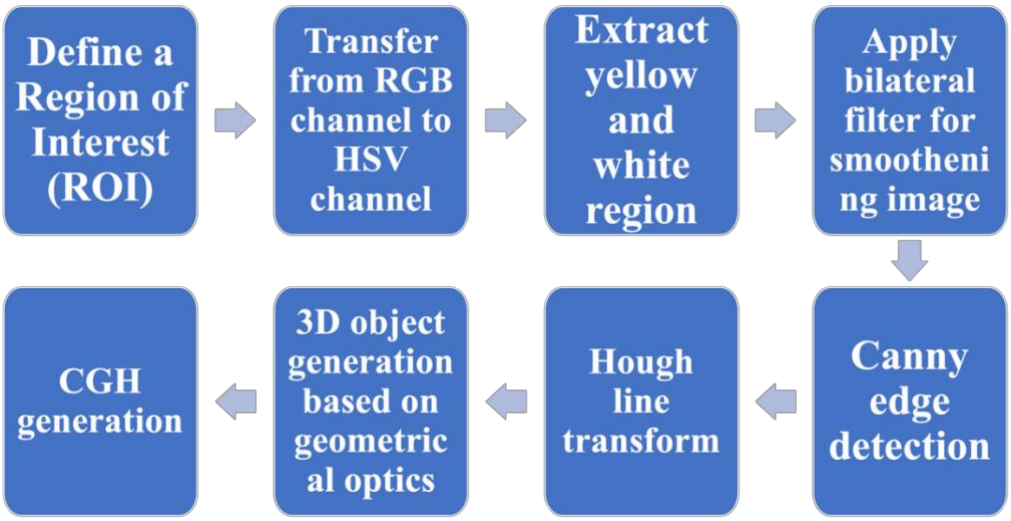


Figure 24. Block diagram of road lane recognition and CGH calculation

Figure 24 shows the more detailed steps of the road lane recognition as well as the CGH generation. The 3D point cloud and line-based CGH is calculated in real-time by local CPU and GPU (Intel CORE i7 13th gen and NVIDIA RTX 4060), as well as cloud based computing with Google Colaboratory (NVIDIA T4 GPU and High RAM mode). Detail of the 3D object is tabulated in **Error! Reference source not found.**.

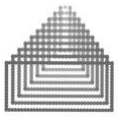
	Number of points and lines	Object pattern
3D point cloud	700	
Line based	40	

Table 1. 3D Object pattern.

Even though this method is simple and robust, it cannot accurately represent the actual road lane direction. The Hough Line Transform, used to extract the final road lane information, produces two straight lines. However, in reality, road lanes are often curved. To address this issue, another common method called curved lane detection can be used. Similar to the first method, the input image is first cropped, and the road is extracted by applying a color mask. Next, perspective warp in OpenCV is employed to transform the perspective from the camera view to a top-down view of the lanes. Based on this top-down image, road lanes can be detected using curve fitting. The data points for curve fitting are obtained through a Histogram Peak Detection algorithm, which generates a histogram of the image with respect to the x-axis. Each part of the histogram shows the number of white pixels in each column of the image. The algorithm then identifies the highest peaks on each side of the image, corresponding to each lane line.

3.3 CGH Calculation

Once the road lanes' function is acquired, generating the 3D object and CGH based on geometric optics becomes straightforward. The 3D point cloud is composed of N luminous points, and each luminous point is an ideal point light source. The wavefront of the object at the hologram surface is formed by the superposition of the spherical wavefronts emitted by all luminous points.

As the TI-PLM operates as a phase-only MEMS-SLM, the phase of the CGH in the PLM plane is determined by Eq. (10).

$$\widetilde{U}_P(x_P, y_P) = \arg\left(\sum_{k=1}^{n} \frac{A}{r_k} \exp\left(j\frac{2\pi}{\lambda} r_k\right)\right) \tag{10}$$

where (x_P, y_P) is pixel position of the PLM and $r_k = \sqrt{(x_k - x_P)^2 + (y_k - y_P)^2 + z_P^2}$ is the distance between the PLM pixel and the object point. λ represents the wavelength utilized, corresponding to the displacement of the micromirror in PLM.

CGH of line-based object is calculated by integral of each line [35]. The line has no depth information which means that it is located at single focal plane which is suitable for this rectangle elemental object. By defining the start and end points of the line, the optical field of a line object can be obtained.

Let's say the start point is (x_1, y_1, z_0), and the end point is (x_2, y_2, z_0). The integral can be presented by Eq. (11).

$$\widetilde{U}_P(x_P, y_P) = \int_{y_1}^{y_2} \int_{x_1}^{x_2} \frac{1}{r} exp\left(i\frac{2\pi}{\lambda} r\right) dxdy \tag{11}$$

where $r = \sqrt{(x - x_P)^2 + (y - y_P)^2 + z_0^2}$. To simplify this integral, we can assume that $(x - x_P)^2 + (y - y_P)^2 = \rho^2 \ll z_0^2$. By employing Taylor expansion, the integral can be simplified like Eq. (12).

$$\widetilde{U}_P(x_P, y_P) \approx \frac{1}{z_0} \exp(ikz_0) \int_{y_1}^{y_2} \int_{x_1}^{x_2} \exp\left(\frac{ik\rho^2}{2z_0}\right) dxdy \tag{12}$$

After getting this equation, we can set $C_1 = \frac{1}{z_0}\exp(ikz_0)$, $C_2 = \frac{k}{2z_0} = \frac{\pi}{\lambda z_0}$. Since the lines contained in the rectangle elemental object is parallel to x or y-axis, we can assume $y_1 = y_2$ or $x_1 = x_2$ during the calculation. Therefore, Eq. (13) can be obtained.

$$\widetilde{U}_P(x_P, y_P) = C_1 \exp\left(iC_2(y_1 - y_P)^2\right) \int_{x_1 - x_P}^{x_2 - x_P} \frac{1}{r} \exp\left(iC_2 x^2\right) dx \tag{13}$$

Since there is no closed-form expression for the integral, error function (Eq. (14)) is introduced to solve the problem.

$$\mathrm{erf(z)} = \frac{2}{\sqrt{\pi}} \int_0^z e^{-t^2} dt \tag{14}$$

Then the integral can be expressed by error function. If $x_1 - x_P < 0$ and $x_2 - x_P > 0$, it will be Eq. (15).

$$\widetilde{U}_P(x_P, y_P) = \frac{1}{4}(1+i)\sqrt{2\lambda z_0} C_1 \, exp(iC_2(y_1 - y_P)^2) * \left[erf\left(\sqrt{\frac{\pi}{2\lambda z_0}}(1-i)|x_1 - x_P|\right) + erf\left(\sqrt{\frac{\pi}{2\lambda z_0}}(1-i)|x_2 - x_P|\right)\right] \tag{15}$$

If $x_1 - x_P$, $x_2 - x_P > 0$ or < 0, it will be Eq. (16).

$$\widetilde{U}_P(x_P, y_P) = \frac{1}{4}(1+i)\sqrt{2\lambda z_0} C_1 \, exp(iC_2(y_1 - y_P)^2) * \left[erf\left(\sqrt{\frac{\pi}{2\lambda z_0}}(1-i)|x_2 - x_P|\right) - erf\left(\sqrt{\frac{\pi}{2\lambda z_0}}(1-i)|x_1 - x_P|\right)\right] \tag{16}$$

The equation is now straightforward and can be calculated when the start and end points are defined. However, there are complex numbers in the error function and there is no package in Python can support this. Therefore, we must simplify the error function.

$$\mathrm{erf}\left(\sqrt{\frac{\pi}{2\lambda z_0}}(1-i)(x_1-x_P)\right)=\mathrm{erf}\left(\frac{1-i}{\sqrt{2}}t\right)=\frac{1}{i-1}\sqrt{\frac{2}{\pi}}\mathcal{E}(t) \tag{17}$$

where $\mathrm{t}=\sqrt{\frac{\pi}{\lambda z_0}}(x_1-x_P)$.

Eq. (17) shows a way to simply the expression. The function $\mathcal{E}(t)$ is expressed in Eq. (18), Eq. (19) and Eq. (20). $\mathcal{E}_S(t)$ and $\mathcal{E}_L(t)$ are valid for small and large values of $|t|$. $\mathcal{E}_S(t)$ is derived using a Taylor expansion, in separate real and imaginary parts. $\mathcal{E}_L(t)$ is derived by truncating an asymptotic expansion.

$$\mathcal{E}(t)\approx\begin{cases}\mathcal{E}_S(t), & \text{if } |t|<1.609\\ \mathcal{E}_L(t), & \text{otherwise}\end{cases} \tag{18}$$

$$\begin{cases}Re\{\mathcal{E}_S(t)\}=2t-\frac{1}{5}t^5+\frac{1}{108}t^9\\ Im\{\mathcal{E}_S(t)\}=-\frac{2}{3}t^3+\frac{1}{21}t^7-\frac{1}{660}t^{11}\end{cases} \tag{19}$$

$$\mathcal{E}_L(t)=(1-i)\sqrt{\frac{\pi}{2}}\mathrm{sgn}(t)+\frac{iexp(-it^2)}{t} \tag{20}$$

where $\mathrm{sgn}(t)$ is sign function. With these methods, the code is finished based on Python environment (APPENDIX D | PYTHON CODE FOR ROAD LANE RECOGNITION AND REAL TIME CGH GENERATION).

Since the calculation process of each point or line is parallel, the optical field can be quickly calculated by GPU which is well-suited for multithreaded tasks [36]. NUMBA package in Python is used for speeding up the code with GPU. The functions responsible for optical field generation are defined in CUDA mode. Variables are transferred from CPU to GPU for optical field calculation and CGH generation.

3.4 Benchmarking Results

Error! Reference source not found. illustrates the time consumption (benchmarking results) across various modes. The time is significantly reduced when transferring from CPU mode to GPU mode, as the optical field of each point or line is calculated parallelly in GPU mode. In the CPU mode, the calculation time for point-based objects' optical fields is less than that for line-based objects. Notably, the processing time in Google Colaboratory is even faster than that of the local GPU mode. Additionally, the time required for optical field generation using the line integral method is shorter compared to the 3D point-based method in both GPU and Colab modes. Although the computational process for the optical field of line-based objects is more complex than that for point-based objects, the number of lines is much fewer than the number of points, which significantly reduces the calculation time in GPU mode.

Object	CPU	GPU	Colab
3D point cloud (700 points)	9.345s	0.133s	0.031s
3D line object (40 lines)	49.09s	0.080s	0.011s

Table 2. Benchmarking results.

3.5 First-order Optical Model

To accurately align the 3D arrow with the road lane, a first-order optical model is essential for calculating magnification and projection distance. The critical aspects to consider are size and depth matching. The objective is to determine the actual road depth using the known focal length of the camera in front of the vehicle and the actual width of the road. Subsequently, the size of the 3D arrow can be computed based on the determined road depth. In the United States, Interstate Highway standards specify a standard lane width of 12 feet (3.7 meters). The focal length of the camera is denoted as f_{cam}. Given the camera sensor size, the image size on the sensor can be easily calculated. The camera is set to focus at infinity. Consequently, the road depth can be computed using the road width in the image. The road width in the image, W_{road}, varies with the depth from the road to the camera. The depth can be calculated using Equation (20):

$$Road\ depth = 3.7 \times \frac{f_{cam}}{W_{road}}\ (meters) \tag{20}$$

After deriving the size and depth information of the road, the superposition of the 3D image and the real world can be achieved by determining the 3D arrow's size and projection distance.

First, the focal length of the image guide, as shown in Figure 27, must be determined based on the field of view (FOV) and the display chip size of the image guide. The display chip measures 11 mm by 6 mm, resulting in a diagonal dimension of 12.53 mm. According to the manufacturer, Epson, the FOV is 23 degrees. Using these parameters, the focal length can be calculated using Equation (21):

$$f_{image\ guide} = \frac{Diagonal\ Dimension}{2 \times \tan\left(FOV/2\right)} = \frac{12.53mm}{2 \times \tan\left(23°/2\right)} = 30.79mm \tag{21}$$

After deriving the focal length of the image guide, image projection can be analyzed using a first-order optical model. In Figure 25, the image guide is depicted as a thin lens. The observer behind the image guide sees the overlap of a 3D arrow and the road lane. The blue object at the focal point indicates the in-coupler's location. The other three solid objects are placed before the focal point to generate virtual images through the image guide. Since the 3D arrow consists of many 2D rectangles at different depths, the virtual image position and size can be adjusted by calculating the object and image distances using geometric optical relationships and the exact magnification at each position. The virtual images of the three objects (dashed arrows in Figure 25) are equally spaced and have the same height, even though the objects themselves are non-linearly spaced and have different heights. By applying the first-order optical model and based on the detected road lane depth, the position and size of the 3D arrow can be precisely adjusted to match the depth and size of the road lane.

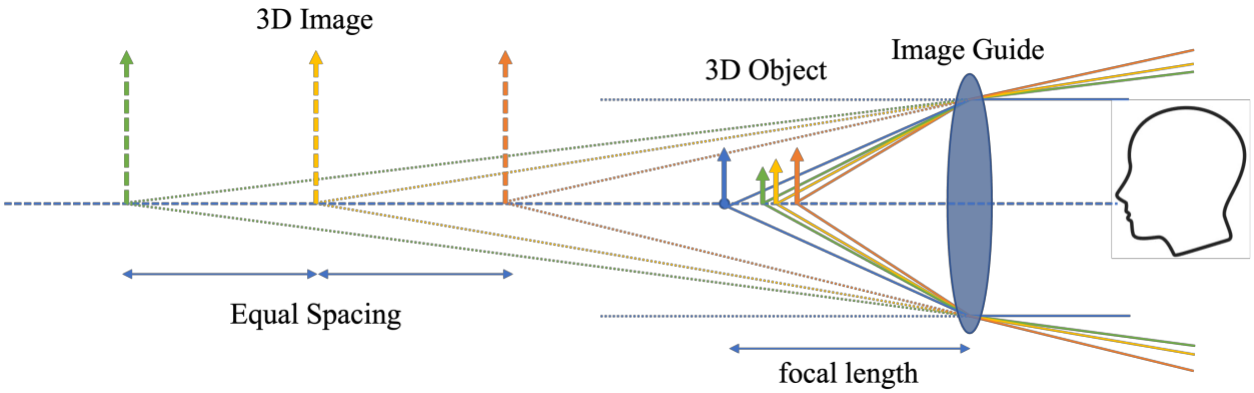


Figure 25. First-order optical model.

Following the explanation of how to overlap the real road lane with the 3D arrow, the experimental setup will also be discussed. Figure 26 shows the experimental setup of the system, illustrating how the 3D arrow overlaps the road lane on the screen instead of a real road lane.

Although it is an overlay between a 3D arrow and a 2D lane, size matching can be achieved through the first-order optical model depicted in Figure 25.

In Figure 27, the road lane displayed on the laptop screen is placed in front of the lens. The lens will generate a virtual image of the screen. The 3D arrow image, generated with the PLM, is displayed around the in-coupler of the image guide, resulting in the virtual image of the 3D arrow being positioned near the screen's virtual image. For the experiment, the 3D arrow's virtual image is set between 3 and 4 meters away from the image guide's principal image plane, which can be achieved by adjusting the object position. Then, by adjusting the position of the screen, its virtual image can be shifted to align with the 3D arrow. Consequently, the distance between the screen and the lens (z_o) can be scaled and used for calculating magnification. The focal length of the lens in front of the image guide is 350 mm. Therefore, the magnification of the screen should be:

$$mag_{screen} = \frac{-f}{z_o - f} = \frac{-350}{z_o - 350} \tag{22}$$

With the calculated magnification and the size of the road lane on the screen, the actual size of the road lane image can be calculated. To match the size between the road lane image and the 3D arrow image, the magnification of the arrow should be considered. Figure 26 shows the plot of the relationship between object and image position with the image guide. The three points on the plot indicate the locations of the rectangles, forming a 3D arrow. According to the plot, the three images are equally spaced between 3 meters and 9 meters, and the corresponding object distances and magnifications are listed in Table 3. Based on the derived magnification, the size of the 3D arrow generated by PLM can be determined. Thus, based on the road lane width on the screen, the precise adjustment of the 3D arrow width can be achieved according to the magnifications.

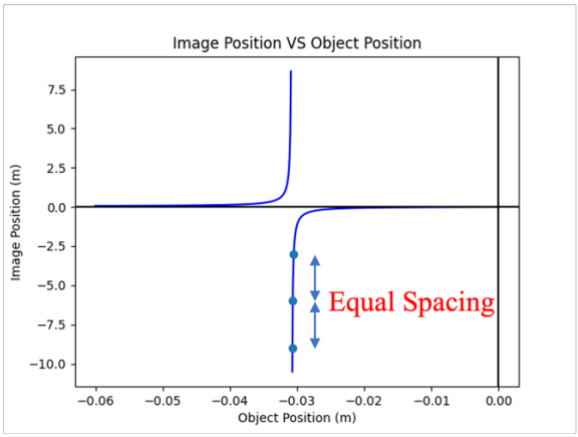


Figure 26. Object position versus image position.

Object Position (m)	Image Position (m)	Magnification
-0.0304772	-3	98.43423937
-0.0306328	-6	195.8684808
-0.03068502	-9	293.3027256

Table 3. Object position, image position and magnification.

3.6 System Setup and Experimental Results

Figure 27 shows the experimental setup. The system consists of an input video of the road displayed on a PC, a MEMS-PLM (Texas Instruments 0.47-inch PLM), a beam splitter, and a geometrical image guide combiner (image guide taken from the commercial product, Moverio BT200 by Epson) that overlays see-through images and 3D objects. A 532 nm laser passes through a beam expander first. After phase modulation by the PLM, the optical field reconstructs the real image of a 3D arrow at the in-coupler position of the image guide. The image guide makes the 3D image focus at infinity. To overlay the 3D image on the PC display screen, a lens is placed in front of the image guide to make the screen appear at infinity as well. The camera behind the image guide then captures the overlap of the 3D arrow and the input image.

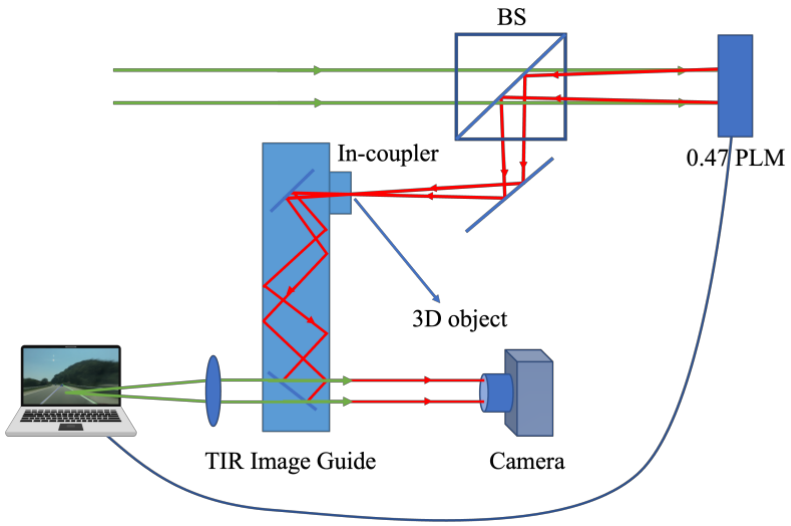


Figure 27. Experimental setup.

Figure 28 shows a frame capture of displayed image taken by a camera (f=50mm, F/1.8, Nikon D700). The display frame rate is 60Hz that is currently limited by the HDMI data rate of the 0.47-inch Texas Instruments PLM.

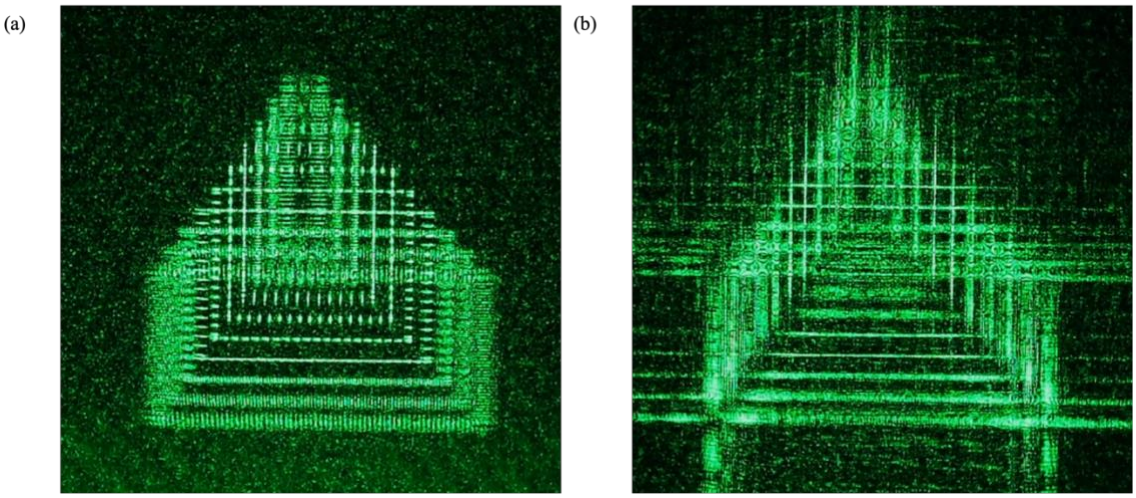


Figure 28. A screenshot of multi-depth rectangle objects consists of (a) 3D point cloud and (b) line images.

Figure 29 shows a frame capture of the displayed road video overlaid with the 3D point-based arrow, as taken by the camera. The 3D arrow matches the size and direction of the detected road lane.

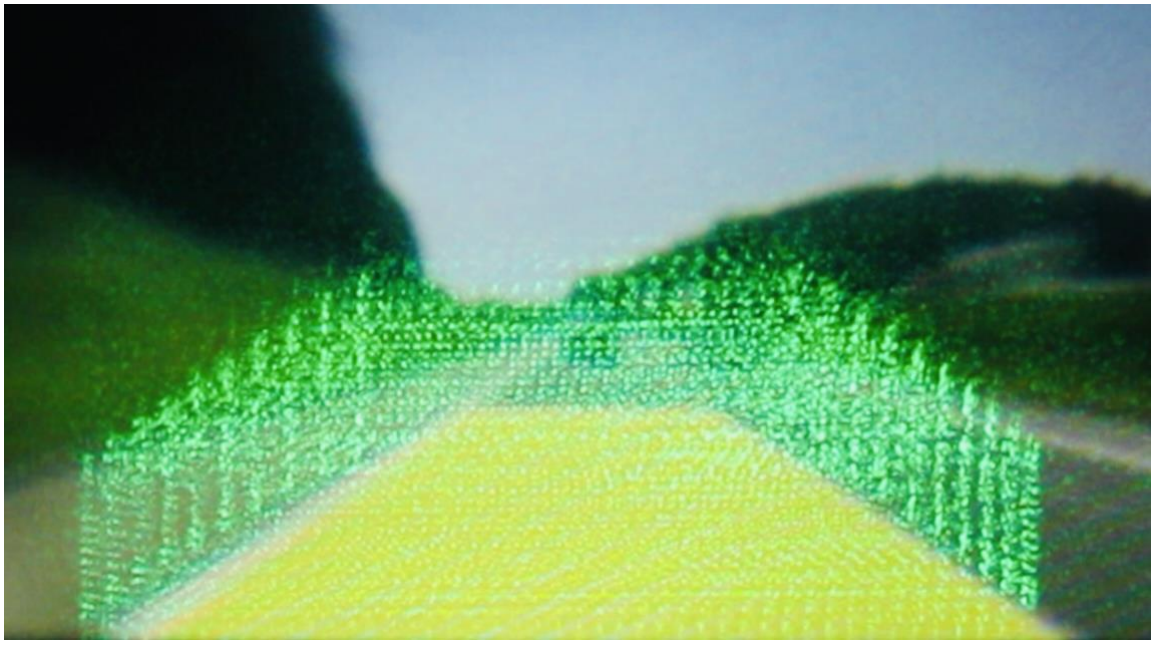


Figure 29. Overlap of the 3D point-based arrow and the detected road on the PC screen.

3.7 Conclusion

In this chapter, two distinct methods were employed for CGH generation. The line integral method exhibited a faster calculation time but resulted in inferior image quality compared to the point cloud method. The reduced image quality may stem from the calculation process, where multiple simplification methods are used for convenience. For example, the spherical wavefront is assumed to be a quadratic wavefront. Taylor expansion and asymptotic expansion are also used, which may introduce some error compared to the real optical field. Additionally, aliasing from frequency overlap may contribute to the poorer image quality. Notably, the time consumption between the two methods is relatively similar. Given the importance of image quality for HUD applications, we conclude that the point cloud method is more feasible.

For the development of a 3D holographic head-up display for an advanced driver assistance system, we benchmarked the speed of CGH calculation for two kinds of objects and across three computational platforms. We demonstrated real-time 3D object generation based on video input,

aligned, and superimposed on a see-through image via an image combiner, with a frame rate of 60Hz if the cloud-based calculation method (Google Colab) is used for CGH calculation. This frame rate is also limited by the HDMI interface cable, but we aim to improve it to the sub-kHz range.

CHAPTER 4 | DISCUSSION

With the application of CGH and TI-PLM, digital phase conjugation (DPC) and real-time 3D object generation for HUD are achieved. TI-PLM shows great promise for 3D holographic displays due to its high resolution, fast response time, and low power consumption.

Off-axis holography was used during DPC to capture more phase information and eliminate low-frequency noise and uneven illumination. The interference pattern was captured through a camera sensor, where the phase was determined by the brightness of the fringes. The precision of the camera sensor might influence phase recording, with overexposure or dark interference patterns affecting the accuracy of the results. A better way to improve the recording process may be using a wavefront sensor to capture more accurate phase information.

The image processing method using off-axis holography and Fourier Transform shows excellent results in correcting optical medium aberrations. By shifting the phase information in the Fourier domain, on-axis phase information can be obtained for reconstruction. Another method, called Zernike polynomials, can also be introduced in image processing. With the obtained on-axis phase information, the coefficients of the Zernike polynomials can be calculated. Since this sequence of polynomials is orthogonal, more accurate phase information can be obtained. Additionally, with these coefficients, different kinds of aberrations can be analyzed. As a result, single-point source and two-point sources at different depths reconstructions were achieved. This method shows great potential for correcting aberrations during AR display applications.

Real-time CGH generation is based on the NVIDIA GPU and NUMBA package in Python. This method takes advantage of the parallel calculation of CGH. For both point-based and line-based objects, the calculation of the optical field at the hologram plane is the superposition of different points or lines' fields. Even though this method greatly increases the calculation speed

compared to CPU calculation, it cannot achieve a high frame rate. The high frame rate is constrained by the HDMI interface cable and the CGH calculation process. Another method, called the look-up table (LUT), was introduced in Chapter 1. The wavefronts of the spherical waves emitted by all points at the hologram plane are calculated in advance and then stored in memory. Therefore, the optical field of a 3D object can be quickly calculated by searching the stored single-point fields and adding them together. Besides the time-consuming CGH calculation, road lane recognition also takes time to detect the road lane. The methods used in this application, including color masking, line detection, and curve fitting, are traditional ways to achieve this goal. Nowadays, neural network or deep learning-based approaches play a critical role in road lane recognition. These methods are not only faster but also yield more accurate results. Therefore, to meet the requirement of a high frame rate, we must improve the calculation speed of CGH and road lane detection.

CHAPTER 5 | CONCLUSION

This thesis presents an introduction and discussion of holographic 3D displays and their applications. Holographic 3D display technology is regarded as one of the most promising 3D display technologies, but its technical implementation is also one of the most difficult ones. Currently, holographic technology is immature in both the software algorithms for hologram calculation and the hardware of display equipment, making practical holographic 3D displays unfeasible. In recent years, the rise of near-eye display technologies such as AR/VR has brought new application scenarios for holographic technology but has also introduced new requirements for its portability. In response to these problems and challenges, this thesis explores several applications of holographic 3D displays.

In this thesis, computer-generated holography (CGH) is used for both Digital Phase Conjugation (DPC) and real-time 3D object generation. For DPC applications, the aberration introduced by the image guide was corrected using off-axis holography. Image processing through Fourier Transform was employed for on-axis CGH generation to eliminate low-frequency noise and uneven illumination. This method achieved the reconstruction of a single point source, corrected by Phase-Locked Loop (PLM). Additionally, the superposition of CGHs from two distinct point sources resulted in the reconstruction of point sources at different depths. This method effectively removes any aberrations introduced by the optical medium, thereby improving display quality.

For real-time 3D object generation, the CUDA mode of NVIDIA GPUs was used to accelerate CGH calculations. The calculation time for the optical field of a 700-point-based 3D object improved from 9.345 seconds to 0.133 seconds (70 times faster than in CPU mode). Similarly, the calculation time for the optical field of a 40-line-based 3D object improved from

49.09 seconds to 0.08 seconds (613 times faster than in CPU mode). With cloud-based calculation using Google Colab, the process can be even faster compared to local GPUs. The calculation time was reduced to 0.031 seconds for point-based objects and 0.011 seconds for line-based objects. With the acceleration of CGH generation, the application for Head-Up Displays (HUD) is considered. The aim is to generate a 3D arrow image overlaid on the road lane to inform the driver while superimposing the real scene in the driver's see-through view. Therefore, a road lane recognition algorithm is introduced. Based on the detected road lane, the CGH will be quickly calculated and displayed onto the PLM to generate a 3D arrow. Displayed road video overlaid with the 3D points, which match the size and direction of the detected road lane arrow, was achieved using image guidance from a camera.

The holographic 3D display applications for DPC and real-time 3D arrow display are achieved based on TI-PLM and CGH generation. However, due to limitations in experimental conditions and time, many issues remain unresolved and need further development in future research. For real-time CGH generation, methods like Look-Up Tables (LUT) could be introduced. Additionally, neural network or deep learning-based approaches could provide faster and more accurate road lane recognition than traditional methods. With these improvements, more reliable and faster real-time 3D arrow generation could be achieved and used for HUD.

APPENDIX A | PUBLICATIONS

Conference Publication

Yefu Zhang, Rajesh Shrestha, Emil Rajan Varghese, John Bass, Ted Liang-tai Lee, Jeff Ching-wen Chan, Xianyue Deng, Yushi Kaneda, Florian Willomitzer, Yuzuru Takashima, "Digital phase conjugation by Texas Instruments PLM for near-to-eye display," Proc. SPIE 12900, Emerging Digital Micromirror Device Based Systems and Applications XVI, 129000F (13 March 2024)

https://doi.org/10.1117/12.3001156

Yefu Zhang, Tianyao Zhang, Xianyue Deng, Rajesh Shrestha, Emil Rajan Varghese, Gregory Michael Nero, Yushi Kaneda, Yuzuru Takashima, “Real-time 3D Objects Generation by MEMS Phase Light Modulator based on Camera Input for ADAS Applications”, ODF 2024. (2024).

Yuzuru Takashima, Yexin Pei, Rajesh Shrestha, Lily Anne McKenna, Yefu Zhang, Emil Rajan Varghese, Xianyue Deng, Gregory Michael Nero, Jeff Chan, Jeff Ching-Wen Chang, Ted Lian-Tai Lee, Parker Liu, Tianyao Zheng, Yushi Kaneda, “Time to Angle Conversion by MEMS Spatial Light Modulator for Wide Field of View AR Projection Engine and Solid-state Scanning Lidar”submitted to SPIE Optics and Photonics, ODS 2024. (2024).

Rajesh Shrestha, Lily Anne McKenna, Yefu Zhang, Emil Rajan Varghese, Xianyue Deng, Jeff Ching-Wen Chang, Yushi Kaneda, Yuzuru Takashima,” Diffractive Beam Steering for Time-of-flight Lidar by MEMS Spatial Light Modulator” ODF 2024. (2024).

APPENDIX B | MATLAB CODE FOR DIGITAL PHASE CONJUGATION

```
% Read image
img = imread("DSC_2592.jpg");
img_grey = rgb2gray(img);
[numRows,numCols] = size(img_grey);

% Create the spatial grid
[Xa,Ya] = meshgrid(1:numCols,1:numRows);

% Create the vector of the spatial frequencies
Sfreq_x = (-1/2:1/numCols:1/2-1/numCols);
Sfreq_y = (-1/2:1/numRows:1/2-1/numRows);

% Frequency grid
[Sx,Sy] = meshgrid(Sfreq_x,Sfreq_y);

% Get FFT
img_fft = fftshift(fft2(img_grey));

% Plot FFT
figure;
imagesc(Sfreq_x,Sfreq_y,abs(img_fft)); axis square;
caxis([0 500000]);

% pick -1 order
roi = drawrectangle();
freq_x = roi.Position(1)+roi.Position(3)/2;
freq_y = roi.Position(2)+roi.Position(4)/2;
width_x = roi.Position(3); % Double the width since we will be using a window
width_y = roi.Position(4); % Double the width since we will be using a window

% keep only -1 order
% x1 = -0.180395;
% x2 = -0.0359693;
% y1 = -0.000976562;
% y2 = 0.0253906;
% freq_x = (x1+x2)/2;
% width_x = x2-x1;
% freq_y = (y1+y2)/2;
% width_y = y2-y1;

% add filter to freq domain
Mask1 = (Sx > freq_x-width_x/2).*(Sx < freq_x+width_x/2);
Mask2 = (Sy > freq_y-width_y/2).*(Sy < freq_y+width_y/2);
img_fft2 = img_fft.*Mask1.*Mask2;

% shift around zero
Mask3 = (Sx > -width_x/2).*(Sx < width_x/2);
Mask4 = (Sy > -width_y/2).*(Sy < width_y/2);% window centered around 0 of width 'width'
I1 = find(Mask1);
I2 = find(Mask2);
I3 = find(Mask3);
I4 = find(Mask4);

% Deal with issues when resolution issues causes masks to have difference
% of one column
[numRows1,numCols1] = size(I1);
[numRows2,numCols2] = size(I2);
[numRows3,numCols3] = size(I3);
[numRows4,numCols4] = size(I4);

if numRows1 > numRows3
    I1 = I1(1:numRows3,1);
else
    I3 = I3(1:numRows1,1);
end

if numRows2 > numRows4
    I2 = I2(1:numRows4,1);
else
    I4 = I4(1:numRows2,1);
end

img_fft3 = zeros(size(img_fft));
img_fft4 = zeros(size(img_fft));
img_fft3(I3) = img_fft2(I1); % shift in x direction
img_fft4(I4) = img_fft3(I2); %shift in y direction
figure,imagesc(Sfreq_x,Sfreq_y,abs(img_fft3)); axis square;
caxis([0 500000]);
```

```
% shifted
tempIFT = ifft2(ifftshift(img_fft4));
finalField = tempIFT(1:numRows,1:numCols);
fringe_phase = angle(finalField);
for i=1:numRows
    for j=1:numCols
        if fringe_phase(i,j)<0
            fringe_phase(i,j) = fringe_phase(i,j)+2*pi;
        end
    end
end

figure,
subplot 121
imagesc(img_grey); axis square;  colormap(hsv); axis off; title('Initial phase')
subplot 122
imagesc(fringe_phase); axis square; caxis([0 2*pi]); colormap(hsv); axis off;
title('Reconstructed phase')


% define PLM displacement matrix
p_0 = [0 0;1 1];
p_1 = [0 0;0 1];
p_2 = [0 0;0 0];
p_3 = [1 0;1 1];
p_4 = [0 0;1 0];
p_5 = [1 0;0 1];
p_6 = [1 0;1 0];
p_7 = [1 0;0 0];
p_8 = [0 1;1 1];
p_9 = [0 1;0 1];
p_10 = [0 1;1 0];
p_11 = [0 1;0 0];
p_12 = [1 1;1 1];
p_13 = [1 1;0 1];
p_14 = [1 1;1 0];
p_15 = [1 1;0 0];

% generate finges CGH
magnificationFactor = 540/numRows;
fringe_phase2 = imresize(fringe_phase,magnificationFactor);


for i=1:540
    for j=1:960
        n=fringe_phase2(i,j);
        if n <= 0.0107*pi*2
            GRT{i,j} =p_1;
        elseif n <= 0.0219*pi*2
                GRT{i,j} =p_2;
        elseif n <= 0.045*pi*2
                GRT{i,j} =p_3;
        elseif n <= 0.0598*pi*2
                GRT{i,j} =p_4;
        elseif n <= 0.0775*pi*2
                GRT{i,j} =p_5;
        elseif n <= 0.1206*pi*2
                GRT{i,j} =p_6;
        elseif n <= 0.185*pi*2
                GRT{i,j} =p_7;
        elseif n <= 0.3655*pi*2
                GRT{i,j} =p_8;
        elseif n <= 0.3955*pi*2
                GRT{i,j} =p_9;
        elseif n <= 0.451*pi*2
                GRT{i,j} =p_10;
        elseif n <= 0.5244*pi*2
                GRT{i,j} =p_11;
        elseif n <= 0.6393*pi*2
                GRT{i,j} =p_12;
        elseif n <= 0.7116*pi*2
                GRT{i,j} =p_13;
        elseif n <= 0.8502*pi*2
                GRT{i,j} =p_14;
        else
            GRT{i,j} =p_15;
        end
    end
end

CGHlevel = cell2mat(GRT);
imwrite(mat2gray(CGHlevel), sprintf('fringe_2_CGH_offaxis_June21.bmp'));
```

APPENDIX C | PYTHON CODE FOR 3D OBJECT CGH GENERATION

```
import numpy as np
import matplotlib.pyplot as plt
import cv2
import math
from PIL import Image as im
from timeit import default_timer as timer
import cmath
import time
import open3d as o3d
from numba import jit, njit, vectorize, cuda, uint32, f8, uint8, prange


@cuda.jit
def CGH_gene(u_final, GRT):
    i,j = cuda.grid(2)

    if i < u_final.shape[0] and j < u_final.shape[1]:

        if u_final[i][j] <= 0.54:
                GRT[i*2,   j*2] = 0
                GRT[i*2,   j*2+1] = 0
                GRT[i*2+1, j*2] = 255
                GRT[i*2+1, j*2+1] = 255

        elif u_final[i][j] <= 1.63:
                GRT[i*2,   j*2] = 0
                GRT[i*2,   j*2+1] = 0
                GRT[i*2+1, j*2] = 0
                GRT[i*2+1, j*2+1] = 255

        elif u_final[i][j] <= 3.35:
                GRT[i*2,   j*2] = 0
                GRT[i*2,   j*2+1] = 0
                GRT[i*2+1, j*2] = 0
                GRT[i*2+1, j*2+1] = 0

        elif u_final[i][j] <= 5.24:
                GRT[i*2,   j*2] = 255
                GRT[i*2,   j*2+1] = 0
                GRT[i*2+1, j*2] = 255
                GRT[i*2+1, j*2+1] = 255

        elif u_final[i][j] <= 6.87:
                GRT[i*2,   j*2] = 0
                GRT[i*2,   j*2+1] = 0
                GRT[i*2+1, j*2] = 255
                GRT[i*2+1, j*2+1] = 0

        elif u_final[i][j] <= 9.91:
                GRT[i*2,   j*2] = 255
                GRT[i*2,   j*2+1] = 0
                GRT[i*2+1, j*2] = 0
                GRT[i*2+1, j*2+1] = 255

        elif u_final[i][j] <= 15.28:
                GRT[i*2,   j*2] = 255
                GRT[i*2,   j*2+1] = 0
                GRT[i*2+1, j*2] = 255
                GRT[i*2+1, j*2+1] = 0

        elif u_final[i][j] <= 27.53:
                GRT[i*2,   j*2] = 255
                GRT[i*2,   j*2+1] = 0
                GRT[i*2+1, j*2] = 0
                GRT[i*2+1, j*2+1] = 0

        elif u_final[i][j] <= 38.05:
                GRT[i*2,   j*2] = 0
                GRT[i*2,   j*2+1] = 255
                GRT[i*2+1, j*2] = 255
                GRT[i*2+1, j*2+1] = 255

        elif u_final[i][j] <= 42.33:
                GRT[i*2,   j*2] = 0
                GRT[i*2,   j*2+1] = 255
                GRT[i*2+1, j*2] = 0
                GRT[i*2+1, j*2+1] = 255
```

```
        elif u_final[i][j] <= 48.77:
                GRT[i*2,   j*2] = 0
                GRT[i*2,   j*2+1] = 255
                GRT[i*2+1, j*2] = 255
                GRT[i*2+1, j*2+1] = 0

        elif u_final[i][j] <= 58.19:
                GRT[i*2,   j*2] = 0
                GRT[i*2,   j*2+1] = 255
                GRT[i*2+1, j*2] = 0
                GRT[i*2+1, j*2+1] = 0

        elif u_final[i][j] <= 67.55:
                GRT[i*2,   j*2] = 255
                GRT[i*2,   j*2+1] = 255
                GRT[i*2+1, j*2] = 255
                GRT[i*2+1, j*2+1] = 255

        elif u_final[i][j] <= 79.7:
                GRT[i*2,   j*2] = 255
                GRT[i*2,   j*2+1] = 255
                GRT[i*2+1, j*2] = 0
                GRT[i*2+1, j*2+1] = 255

        elif u_final[i][j] <= 92.51:
                GRT[i*2,   j*2] = 255
                GRT[i*2,   j*2+1] = 255
                GRT[i*2+1, j*2] = 255
                GRT[i*2+1, j*2+1] = 0

        else:
                GRT[i*2,   j*2] = 255
                GRT[i*2,   j*2+1] = 255
                GRT[i*2+1, j*2] = 0
                GRT[i*2+1, j*2+1] = 0


@cuda.jit
def opti_field(X, Y, z, u_final):
    a,b = cuda.grid(2)
    h = 540
    w = 960
    PS = 0.0108
    PI = 3.14159265358979323846

    if a < h and b < w:
        r = math.sqrt((X-(a-h/2)*PS)**2 + (Y-(b-w/2)*PS)**2 + z ** 2)
        u_final[a][b] += 1 / r * cmath.exp(-1j * 2 * PI / 0.000532 * r)


@cuda.jit
def rotation(X, Y, Z, rotate_angle):
    a = cuda.grid(1)
    if a < X.shape[0]:
        x = X[a] * np.cos(rotate_angle) + Z[a] * np.sin(rotate_angle)
        Y[a] = Y[a]
        z = -1 * X[a] * np.sin(rotate_angle) + Z[a] * np.cos(rotate_angle)
        X[a] = x
        Z[a] = z



bunny = o3d.data.BunnyMesh()
mesh = o3d.io.read_triangle_mesh(bunny.path)
mesh.compute_vertex_normals()
# o3d.visualization.draw_geometries([mesh])
pcd = mesh.sample_points_uniformly(number_of_points=3000)
# o3d.visualization.draw_geometries([pcd])
points = np.asarray(pcd.points)

points[:,0] = points[:,0]-(points[:,0].max()+points[:,0].min())/2
points[:,1] = points[:,1]-(points[:,1].max()+points[:,1].min())/2
points[:,2] = points[:,2]-(points[:,2].max()+points[:,2].min())/2
points = points*30
```

```
h = 540
w = 960
u_final = np.zeros((h, w), dtype=np.complex128)

TPB = 16
threadsperblock = (TPB, TPB)
blockspergrid_x = math.ceil(u_final.shape[0]/threadsperblock[0])
blockspergrid_y = math.ceil(u_final.shape[1]/threadsperblock[1])
blockspergrid = (blockspergrid_x, blockspergrid_y)


start = time.process_time()
rotate_angle = 0
points_gpu = cuda.to_device(points)
while rotate_angle < 2 * np.pi:

    rotation[blockspergrid, threadsperblock](points_gpu[:, 0], points_gpu[:, 1],
points_gpu[:, 2], np.pi/25)
    points_cpu = points_gpu.copy_to_host()
    points_cpu[:, 2] += 100

    u_final = np.zeros((h, w), dtype=np.complex128)
    u_final_gpu = cuda.to_device(u_final)

    for a in range(points_cpu[:, 0].shape[0]):
        opti_field[blockspergrid, threadsperblock](points_cpu[a, 0], points_cpu[a, 1],
points_cpu[a, 2], u_final_gpu)

    GRT = np.zeros((h * 2, w * 2), dtype=np.uint8)
    GRT_gpu = cuda.to_device(GRT)

    u_final_cpu = u_final_gpu.copy_to_host()
    u_final_cpu = ((np.angle(u_final_cpu) + np.pi) / (2 * np.pi) * 100).real
    u_final_gpu = cuda.to_device(u_final_cpu)

    CGH_gene[blockspergrid, threadsperblock](u_final_gpu, GRT_gpu)

    GRT_cpu = GRT_gpu.copy_to_host()

    cv2.imshow('cgh', GRT_cpu)

    if cv2.waitKey(25) & 0xFF == ord('q'):
        break

    rotate_angle += np.pi/25

print(time.process_time()-start)

# Closes all the frames
cv2.destroyAllWindows()
```

APPENDIX D |
PYTHON CODE FOR ROAD LANE RECOGNITION AND REAL TIME CGH GENERATION

```
import numpy as np
from scipy.stats import linregress
import matplotlib.pyplot as plt
import cv2
# import torch
import math
from PIL import Image as im
from timeit import default_timer as timer
import cmath
import open3d as o3d
from numba import jit, njit, vectorize, cuda, uint32, f8, uint8, prange


@cuda.jit
def opti_field_line(point1, point2, z, u_final):
    h = 540
    w = 960
    wavelength = 0.000532
    PS = 0.0108
    a, b = cuda.grid(2)  # 2D grid
    PI = 3.14159265358979323846

    if point1[1] == point2[1]:
        if a < h and b < w:
            t1 = math.sqrt(PI/wavelength/z) * abs(point1[0]-(b-w/2)*PS)
            t2 = math.sqrt(PI/wavelength/z) * abs(point2[0]-(b-w/2)*PS)
            c5 = 1/4 * (1+1j) * math.sqrt(2*wavelength*z) * 1/z * cmath.exp(1j*2*PI/wavelength*z) *
cmath.exp(1j*PI/wavelength/z*(point1[1]-(a-h/2)*PS)**2)
            if t1 < 1.609 and t2 < 1.609:
                c1 = 1/(1j-1) * math.sqrt(2/PI) * ((2*t1-1/5*t1**5+1/108*t1**9) + 1j*
(-2/3*t1**3+1/21*t1**7+1/660*t1**11))
                c2 = 1/(1j-1) * math.sqrt(2/PI) * ((2*t2-1/5*t2**5+1/108*t2**9) + 1j*
(-2/3*t2**3+1/21*t2**7+1/660*t2**11))
                c = c2 - math.copysign(1, (point1[0]-(b-w/2)*PS) * (point2[0]-(b-w/2)*PS))*c1
                u_final[a][b] += c5 * c
            elif t1 < 1.609 and t2 > 1.609:
                c1 = 1/(1j-1) * math.sqrt(2/PI) * ((2*t1-1/5*t1**5+1/108*t1**9) + 1j*
(-2/3*t1**3+1/21*t1**7+1/660*t1**11))
                c4 = -1*math.copysign(1, t2)+ (1-1j)/2 * math.sqrt(2/PI) * cmath.exp(-1j*t2**2)/t2
                c = c4 - math.copysign(1, (point1[0]-(b-w/2)*PS) * (point2[0]-(b-w/2)*PS))*c1
                u_final[a][b] += c5 * c
            elif t1 > 1.609 and t2 < 1.609:
                c2 = 1/(1j-1) * math.sqrt(2/PI) * ((2*t2-1/5*t2**5+1/108*t2**9) + 1j*
(-2/3*t2**3+1/21*t2**7+1/660*t2**11))
                c3 = -1*math.copysign(1, t1)+ (1-1j)/2 * math.sqrt(2/PI) * cmath.exp(-1j*t1**2)/t1
                c = c2 - math.copysign(1, (point1[0]-(b-w/2)*PS) * (point2[0]-(b-w/2)*PS))*c3
                u_final[a][b] += c5 * c
            elif t1 > 1.609 and t2 > 1.609:
                c3 = -1*math.copysign(1, t1)+ (1-1j)/2 * math.sqrt(2/PI) * cmath.exp(-1j*t1**2)/t1
                c4 = -1*math.copysign(1, t2)+ (1-1j)/2 * math.sqrt(2/PI) * cmath.exp(-1j*t2**2)/t2
                c = c4 - math.copysign(1, (point1[0]-(b-w/2)*PS) * (point2[0]-(b-w/2)*PS))*c3
                u_final[a][b] += c5 * c
    else:
        if a < h and b < w:
            t1 = math.sqrt(PI/wavelength/z) * abs(point1[1]-(a-h/2)*PS)
            t2 = math.sqrt(PI/wavelength/z) * abs(point2[1]-(a-h/2)*PS)
            c5 = 1/4 * (1+1j) * math.sqrt(2*wavelength*z) * 1/z * cmath.exp(1j*2*PI/wavelength*z) *
cmath.exp(1j*PI/wavelength/z*(point1[0]-(b-w/2)*PS)**2)
            if t1 < 1.609 and t2 < 1.609:
                c1 = 1/(1j-1) * math.sqrt(2/PI) * ((2*t1-1/5*t1**5+1/108*t1**9) + 1j*
(-2/3*t1**3+1/21*t1**7+1/660*t1**11))
                c2 = 1/(1j-1) * math.sqrt(2/PI) * ((2*t2-1/5*t2**5+1/108*t2**9) + 1j*
(-2/3*t2**3+1/21*t2**7+1/660*t2**11))
                c = c2 - math.copysign(1, (point1[1]-(a-h/2)*PS) * (point2[1]-(a-h/2)*PS))*c1
                u_final[a][b] += c5 * c
            elif t1 < 1.609 and t2 > 1.609:
                c1 = 1/(1j-1) * math.sqrt(2/PI) * ((2*t1-1/5*t1**5+1/108*t1**9) + 1j*
(-2/3*t1**3+1/21*t1**7+1/660*t1**11))
                c4 = -1*math.copysign(1, t2)+ (1-1j)/2 * math.sqrt(2/PI) * cmath.exp(-1j*t2**2)/t2
                c = c4 - math.copysign(1, (point1[1]-(a-h/2)*PS) * (point2[1]-(a-h/2)*PS))*c1
                u_final[a][b] += c5 * c
            elif t1 > 1.609 and t2 < 1.609:
                c2 = 1/(1j-1) * math.sqrt(2/PI) * ((2*t2-1/5*t2**5+1/108*t2**9) + 1j*
(-2/3*t2**3+1/21*t2**7+1/660*t2**11))
                c3 = -1*math.copysign(1, t1)+ (1-1j)/2 * math.sqrt(2/PI) * cmath.exp(-1j*t1**2)/t1
                c = c2 - math.copysign(1, (point1[1]-(a-h/2)*PS) * (point2[1]-(a-h/2)*PS))*c3
                u_final[a][b] += c5 * c
            elif t1 > 1.609 and t2 > 1.609:
                c3 = -1*math.copysign(1, t1)+ (1-1j)/2 * math.sqrt(2/PI) * cmath.exp(-1j*t1**2)/t1
                c4 = -1*math.copysign(1, t2)+ (1-1j)/2 * math.sqrt(2/PI) * cmath.exp(-1j*t2**2)/t2
                c = c4 - math.copysign(1, (point1[1]-(a-h/2)*PS) * (point2[1]-(a-h/2)*PS))*c3
                u_final[a][b] += c5 * c

@cuda.jit
def CGH_gene(u_final, GRT):
    i, j = cuda.grid(2)

    if i < u_final.shape[0] and j < u_final.shape[1]:
```

```
        if u_final[i][j] <= 0.54:
            GRT[i * 2, j * 2] = 0
            GRT[i * 2, j * 2 + 1] = 0
            GRT[i * 2 + 1, j * 2] = 255
            GRT[i * 2 + 1, j * 2 + 1] = 255

        elif u_final[i][j] <= 1.63:
            GRT[i * 2, j * 2] = 0
            GRT[i * 2, j * 2 + 1] = 0
            GRT[i * 2 + 1, j * 2] = 0
            GRT[i * 2 + 1, j * 2 + 1] = 255

        elif u_final[i][j] <= 3.35:
            GRT[i * 2, j * 2] = 0
            GRT[i * 2, j * 2 + 1] = 0
            GRT[i * 2 + 1, j * 2] = 0
            GRT[i * 2 + 1, j * 2 + 1] = 0

        elif u_final[i][j] <= 5.24:
            GRT[i * 2, j * 2] = 255
            GRT[i * 2, j * 2 + 1] = 0
            GRT[i * 2 + 1, j * 2] = 255
            GRT[i * 2 + 1, j * 2 + 1] = 255

        elif u_final[i][j] <= 6.87:
            GRT[i * 2, j * 2] = 0
            GRT[i * 2, j * 2 + 1] = 0
            GRT[i * 2 + 1, j * 2] = 255
            GRT[i * 2 + 1, j * 2 + 1] = 0

        elif u_final[i][j] <= 9.91:
            GRT[i * 2, j * 2] = 255
            GRT[i * 2, j * 2 + 1] = 0
            GRT[i * 2 + 1, j * 2] = 0
            GRT[i * 2 + 1, j * 2 + 1] = 255

        elif u_final[i][j] <= 15.28:
            GRT[i * 2, j * 2] = 255
            GRT[i * 2, j * 2 + 1] = 0
            GRT[i * 2 + 1, j * 2] = 255
            GRT[i * 2 + 1, j * 2 + 1] = 0

        elif u_final[i][j] <= 27.53:
            GRT[i * 2, j * 2] = 255
            GRT[i * 2, j * 2 + 1] = 0
            GRT[i * 2 + 1, j * 2] = 0
            GRT[i * 2 + 1, j * 2 + 1] = 0

        elif u_final[i][j] <= 38.05:
            GRT[i * 2, j * 2] = 0
            GRT[i * 2, j * 2 + 1] = 255
            GRT[i * 2 + 1, j * 2] = 255
            GRT[i * 2 + 1, j * 2 + 1] = 255

        elif u_final[i][j] <= 42.33:
            GRT[i * 2, j * 2] = 0
            GRT[i * 2, j * 2 + 1] = 255
            GRT[i * 2 + 1, j * 2] = 0
            GRT[i * 2 + 1, j * 2 + 1] = 255

        elif u_final[i][j] <= 48.77:
            GRT[i * 2, j * 2] = 0
            GRT[i * 2, j * 2 + 1] = 255
            GRT[i * 2 + 1, j * 2] = 255
            GRT[i * 2 + 1, j * 2 + 1] = 0

        elif u_final[i][j] <= 58.19:
            GRT[i * 2, j * 2] = 0
            GRT[i * 2, j * 2 + 1] = 255
            GRT[i * 2 + 1, j * 2] = 0
            GRT[i * 2 + 1, j * 2 + 1] = 0

        elif u_final[i][j] <= 67.55:
            GRT[i * 2, j * 2] = 255
            GRT[i * 2, j * 2 + 1] = 255
            GRT[i * 2 + 1, j * 2] = 255
            GRT[i * 2 + 1, j * 2 + 1] = 255

        elif u_final[i][j] <= 79.7:
            GRT[i * 2, j * 2] = 255
            GRT[i * 2, j * 2 + 1] = 255
            GRT[i * 2 + 1, j * 2] = 0
            GRT[i * 2 + 1, j * 2 + 1] = 255

        elif u_final[i][j] <= 92.51:
            GRT[i * 2, j * 2] = 255
            GRT[i * 2, j * 2 + 1] = 255
            GRT[i * 2 + 1, j * 2] = 255
            GRT[i * 2 + 1, j * 2 + 1] = 0

        else:
            GRT[i * 2, j * 2] = 255
            GRT[i * 2, j * 2 + 1] = 255
            GRT[i * 2 + 1, j * 2] = 0
            GRT[i * 2 + 1, j * 2 + 1] = 0
```

```
@cuda.jit
def opti_field(X, Y, z, u_final):
    a, b = cuda.grid(2)
    h = 540
    w = 960
    PS = 0.0108
    PI = 3.14159265358979323846

    if a < h and b < w:
        r = math.sqrt((Y - (a - h / 2) * PS) ** 2 + (X - (b - w / 2) * PS) ** 2 + z ** 2)
        u_final[a][b] += 1 / r * cmath.exp(-1j * 2 * PI / 0.000532 * r)


@njit
def rectangular_4point(x, y, z, h, w):
    point_lu = np.zeros([w.size, 3])
    point_ld = np.zeros([w.size, 3])
    point_ru = np.zeros([w.size, 3])
    point_rd = np.zeros([w.size, 3])
    for num in range(w.size):
        point_lu[num, :] = np.array([x[num] - w[num] / 2, y[num] + h[num] / 2, z[num]])
        point_ld[num, :] = np.array([x[num] - w[num] / 2, y[num] - h[num] / 2, z[num]])
        point_ru[num, :] = np.array([x[num] + w[num] / 2, y[num] + h[num] / 2, z[num]])
        point_rd[num, :] = np.array([x[num] + w[num] / 2, y[num] - h[num] / 2, z[num]])

    return point_lu, point_ru, point_ld, point_rd


@njit
def road_lane_point(eq_left, eq_right, image_shape, rect_num, intersect_pt):
    width = np.zeros(rect_num)
    x = np.zeros(rect_num)
    spacing = (image_shape[0]-intersect_pt[1])/rect_num
    for i in range(width.size):
        x_left = (image_shape[0] - i * spacing - eq_left[1]) / eq_left[0]
        x_right = (image_shape[0] - i * spacing - eq_right[1]) / eq_right[0]
        x[i] = -((x_left + x_right)/2 - image_shape[1]/2) * 0.001
        width[i] = abs(x_left - x_right) * 0.0015
    z = np.linspace(50 + 0.01 * spacing * rect_num, 50, rect_num)
    y = np.linspace(0.5, -0.5, z.shape[0])
    h = np.zeros(z.shape) + 0.8
    point_lu, point_ru, point_ld, point_rd = rectangular_4point(x, y, z, h, width)

    return point_lu, point_ru, point_ld, point_rd


@njit
def image_pipeline(image, prev_left_ms=[], prev_left_bs=[], prev_right_ms=[],
                   prev_right_bs=[], bottom_offset=0, should_plot=False):
    height = image.shape[0]
    width = image.shape[1]

    image_cropped = crop(image, bottom_offset)
    image_blured = biliteral(image_cropped)
    image_unsharp = unsharp_mask(image_cropped, image_blured)
    image_clached = clache(image_unsharp)

    # ---------------------------- HOUGH ---------------------------
    left_lns, right_lns, median_left_f, median_right_f = detect(
        image_clached, 220, prev_left_ms, prev_left_bs, prev_right_ms, prev_right_bs)

    if median_left_f is None and median_right_f is None:
        return [image, None, None, None]

    intersect_pt = intersect(median_left_f, median_right_f, height)

    if intersect_pt is None:
        return [image, median_left_f, median_right_f, None]

    left_x = int((median_left_f - height).roots)
    right_x = int((median_right_f - height).roots)

    # ---------------------------- DRAW ---------------------------

    if should_plot:
        plot_start()
        plot_lines(left_lns, 'r')
        plot_lines(right_lns, 'b')
```

```
        if median_left_f is not None:
            plot_function(median_left_f, [left_x, intersect_pt[0]])

        if median_right_f is not None:
            plot_function(median_right_f, [intersect_pt[0], right_x])

    lanes = draw_lanes(height, width, left_x, right_x, intersect_pt)
    return [weighted_img(lanes, image), median_left_f, median_right_f, intersect_pt]


@njit
def weighted_img(img, initial_img, α=0.8, β=1., λ=0.):
    return cv2.addWeighted(initial_img, α, img, β, λ)


def stack_images(img_a, img_b):
    return np.hstack((img_a, img_b))


def show_image(img):
    plt.figure(figsize=(15, 10))
    plt.imshow(img)


def show_gray_image(img):
    plt.figure(figsize=(15, 10))
    plt.imshow(img, cmap='gray')


@njit
def plot_start():
    plt.figure(figsize=(15, 10))


@njit
def plot_lines(lines, color):
    for line in lines:
        for x1, y1, x2, y2 in line:
            plt.plot((x1, x2), (y1, y2), color)


@njit
def plot_function(f, xs):
    for x in range(xs[0], xs[len(xs) - 1]):
        plt.plot(x, f(x), 'g2')


@njit
def region_of_interest(img, vertices):
    mask = np.zeros_like(img)

    # defining a 3 channel or 1 channel color to fill the mask with depending on the input
image
    if len(img.shape) > 2:
        channel_count = img.shape[2]  # i.e. 3 or 4 depending on your image
        ignore_mask_color = (255,) * channel_count
    else:
        ignore_mask_color = 255

    cv2.fillPoly(mask, vertices, ignore_mask_color)
    masked_image = cv2.bitwise_and(img, mask)

    return masked_image


@njit
def crop_roi(image, top_left, top_right, bottom_right, bottom_left):
    roi = [np.array([top_left, top_right, bottom_right, bottom_left], dtype=np.int32)]
    return region_of_interest(image, roi)


@njit
def crop_by_ref(img, ref_width, ref_height, ref_top_x, ref_top_y, ref_bot_x, ref_bot_y):
    width = img.shape[1]
    image_height = img.shape[0]
    middle_x = int(width / 2)
    image_offset_bottom_x = int(width * ref_bot_x / ref_width)
    image_offset_bottom_y = int(image_height * ref_bot_y / ref_height)
    image_offset_top_x = int(width * ref_top_x / ref_width)
    image_offset_top_y = int(image_height * ref_top_y / ref_height)
```

```
    top_left = [middle_x - image_offset_top_x, image_offset_top_y]
    top_right = [middle_x + image_offset_top_x, image_offset_top_y]
    bottom_right = [width - image_offset_bottom_x, image_offset_bottom_y]
    bottom_left = [image_offset_bottom_x, image_offset_bottom_y]

    return crop_roi(img, top_left, top_right, bottom_right, bottom_left)


@njit
def crop(image, bottom_offset=0):
    ref_width = 960
    ref_height = 540
    ref_top_x = 50
    ref_top_y = 400
    ref_bottom_x = 70
    ref_bottom_y = 540 - bottom_offset

    return crop_by_ref(image, ref_width, ref_height, ref_top_x, ref_top_y, ref_bottom_x,
ref_bottom_y)


@njit
def binary_hsv_mask(img, color_range):
    lower = np.array(color_range[0])
    upper = np.array(color_range[1])

    return cv2.inRange(img, lower, upper)


@njit
def binary_gray_mask(img, color_range):
    lower = np.array(color_range[0])
    upper = np.array(color_range[1])

    return cv2.inRange(img, color_range[0][0], color_range[1][0])


@njit
def binary_mask_apply(img, binary_mask):
    masked_image = np.zeros_like(img)

    for i in range(3):
        masked_image[:, :, i] = binary_mask.copy()

    return masked_image


def binary_mask_apply_color(img, binary_mask):
    return cv2.bitwise_and(img, img, mask=binary_mask)


@njit
def draw_lanes(height, width, left_x, right_x, intersect_pt):
    line_img = np.zeros((height, width, 3), dtype=np.uint8)

    if left_x != 0 and intersect_pt != None:
        cv2.line(line_img, (left_x, height), (intersect_pt[0], intersect_pt[1]), [255, 0, 0],
5)

    if right_x != 0 and intersect_pt != None:
        cv2.line(line_img, (intersect_pt[0], intersect_pt[1]), (right_x, height), [0, 255,
0], 5)

    return line_img


@njit
def filter_by_color_ranges(img, color_ranges):
    result = np.zeros_like(img)
    gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    hsv_img = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)

    for color_range in color_ranges:
        color_bottom = color_range[0]
        color_top = color_range[1]

        if color_bottom[0] == color_bottom[1] == color_bottom[2] and color_top[0] ==
color_top[1] == color_top[2]:
            mask = binary_gray_mask(gray_img, color_range)
        else:
            mask = binary_hsv_mask(hsv_img, color_range)
```

```
        masked_img = binary_mask_apply(img, mask)
        result = cv2.addWeighted(masked_img, 1.0, result, 1.0, 0.0)

    return result


@njit
def color_threshold(img, white_value):
    white = [[white_value, white_value, white_value], [255, 255, 255]]
    yellow = [[80, 90, 90], [120, 255, 255]]

    return filter_by_color_ranges(img, [white, yellow])


@njit
def equalize_histogram(img):
    img_yuv = cv2.cvtColor(img, cv2.COLOR_BGR2YUV)
    img_yuv[:, :, 0] = cv2.equalizeHist(img_yuv[:, :, 0])

    return cv2.cvtColor(img_yuv, cv2.COLOR_YUV2BGR)


@njit
def clache(img):
    cl = cv2.createCLAHE(clipLimit=3.0, tileGridSize=(8, 8))

    img_yuv = cv2.cvtColor(img, cv2.COLOR_BGR2YUV)
    y, u, v = cv2.split(img_yuv)
    y_clache = cl.apply(y)
    img_yuv = cv2.merge((y_clache, u, v))

    return cv2.cvtColor(img_yuv, cv2.COLOR_YUV2BGR)


@njit
def canny(img, low_threshold, high_threshold):
    return cv2.Canny(img, low_threshold, high_threshold)


def gaussian_blur(img, kernel_size):
    return cv2.GaussianBlur(img, (kernel_size, kernel_size), 0)


@njit
def biliteral(img):
    return cv2.bilateralFilter(img, 13, 75, 75)


@njit
def unsharp_mask(image, blured):
    return cv2.addWeighted(blured, 1.5, blured, -0.5, 0, image)


@njit
def edges(img):
    gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    v = np.median(gray_img)
    sigma = 0.33
    lower = int(max(150, (1.0 - sigma) * v))
    upper = int(min(255, (1.0 + sigma) * v))

    return canny(gray_img, lower, upper)


@njit
def detect(image, white_value, prev_left_ms, prev_left_bs, prev_right_ms, prev_right_bs):
    houghed_lns = hough_lines(image, white_value)

    if houghed_lns is None:
        return [None, None, None, None]

    filtered_left_lns, filtered_right_lns = left_right_lines(houghed_lns)
    median_left_f = median(filtered_left_lns, prev_left_ms, prev_left_bs)
    median_right_f = median(filtered_right_lns, prev_right_ms, prev_right_bs)

    if median_left_f is None or median_right_f is None:
        return detect(image, white_value - 5, prev_left_ms, prev_left_bs, prev_right_ms,
prev_right_bs)
    else:
        return [filtered_left_lns, filtered_right_lns, median_left_f, median_right_f]
```

```
@njit
def hough_lines(image, white_value):
    if white_value < 150:
        return None

    image_masked = color_threshold(image, white_value)
    image_edges = edges(image_masked)
    houghed_lns = cv2.HoughLinesP(image_edges, 2, np.pi / 180, 50, np.array([]), 20, 100)

    if houghed_lns is None:
        return hough_lines(image, white_value - 5)

    return houghed_lns


@njit
def left_right_lines(lines):
    lines_all_left = []
    lines_all_right = []
    slopes_left = []
    slopes_right = []

    for line in lines:
        for x1, y1, x2, y2 in line:
            slope = (y2 - y1) / (x2 - x1)

            if slope > 0:
                lines_all_right.append(line)
                slopes_right.append(slope)
            else:
                lines_all_left.append(line)
                slopes_left.append(slope)

    filtered_left_lns = filter_lines_outliers(lines_all_left, slopes_left, True)
    filtered_right_lns = filter_lines_outliers(lines_all_right, slopes_right, False)

    return filtered_left_lns, filtered_right_lns


@njit
def filter_lines_outliers(lines, slopes, is_left, min_slope=0.5, max_slope=0.9):
    if len(lines) < 2:
        return lines

    lines_no_outliers = []
    slopes_no_outliers = []

    # filter the lines according to the slope of the lines (range and standard deviation)
    for i, line in enumerate(lines):
        slope = slopes[i]

        if min_slope < abs(slope) < max_slope:
            lines_no_outliers.append(line)
            slopes_no_outliers.append(slope)

    slope_median = np.median(slopes_no_outliers)
    slope_std_deviation = np.std(slopes_no_outliers)
    filtered_lines = []

    for i, line in enumerate(lines_no_outliers):
        slope = slopes_no_outliers[i]
        intercepts = np.median(line)

        if slope_median - 2 * slope_std_deviation < slope < slope_median + 2 *
slope_std_deviation:
            filtered_lines.append(line)

    return filtered_lines


@njit
def median(lines, prev_ms, prev_bs):
    if prev_ms is None:
        prev_ms = []
        prev_bs = []

    xs = []
    ys = []
    xs_med = []
    ys_med = []
    m = 0
    b = 0
```

```
    for line in lines:
        for x1, y1, x2, y2 in line:
            xs += [x1, x2]
            ys += [y1, y2]

    if len(xs) > 2 and len(ys) > 2:
        #         m, b = np.polyfit(xs, ys, 1)
        m, b, r_value_left, p_value_left, std_err = linregress(xs, ys)

        if len(prev_ms) > 0:
            prev_ms.append(m)
            prev_bs.append(b)
        else:
            return np.poly1d([m, b])

    if len(prev_ms) > 0:
        return np.poly1d([np.average(prev_ms), np.average(prev_bs)])
    else:
        return None


@njit
def intersect(f_a, f_b, bottom_y):
    if f_a is None or f_b is None:
        return None

    equation = f_a.coeffs - f_b.coeffs
    x = -equation[1] / equation[0]
    y = np.poly1d(f_a.coeffs)(x)
    x, y = map(int, [x, y])

    return [x, y]


h = 540
w = 960
u_final = np.zeros((h, w), dtype=np.complex128)

TPB = 16
threadsperblock = (TPB, TPB)
blockspergrid_x = math.ceil(u_final.shape[0] / threadsperblock[0])
blockspergrid_y = math.ceil(u_final.shape[1] / threadsperblock[1])
blockspergrid = (blockspergrid_x, blockspergrid_y)


cap = cv2.VideoCapture("road1.mp4")

# Read until video is completed
while (cap.isOpened()):

    # Capture frame-by-frame
    ret, frame = cap.read()
    if ret == True:
        # Display the resulting frame
        output_image, median_left_f, median_right_f, intersect_pt = image_pipeline(frame)

        if intersect_pt != None:
            equation_left = median_left_f.coeffs
            equation_right = median_right_f.coeffs
            point_lu, point_ru, point_ld, point_rd = road_lane_point(equation_left,
equation_right, output_image.shape, 10,
                                                                     intersect_pt)

            u_final = np.zeros((h, w), dtype=np.complex128)
            u_final_gpu = cuda.to_device(u_final)
            # u_final_gpu = cuda.to_device(u_final)

            for a in range(point_lu.shape[0]):
                opti_field_line[blockspergrid, threadsperblock](point_rd[a, :], point_ru[a,
:], point_rd[a, 2], u_final_gpu)
                opti_field_line[blockspergrid, threadsperblock](point_ld[a, :], point_lu[a,
:], point_rd[a, 2], u_final_gpu)
                opti_field_line[blockspergrid, threadsperblock](point_ld[a, :], point_rd[a,
:], point_rd[a, 2], u_final_gpu)
                opti_field_line[blockspergrid, threadsperblock](point_lu[a, :], point_ru[a,
:], point_rd[a, 2], u_final_gpu)

            GRT = np.zeros((h * 2, w * 2), dtype=np.uint8)
            GRT_gpu = cuda.to_device(GRT)

            # u_final_cpu = u_final_gpu.copy_to_host()
            u_final_cpu = u_final_gpu.copy_to_host()
            u_final_cpu = ((-np.angle(u_final_cpu) + np.pi) / (2 * np.pi) * 100)
```

```
            CGH_gene[blockspergrid, threadsperblock](u_final_cpu, GRT_gpu)

            GRT_cpu = GRT_gpu.copy_to_host()

            cv2.namedWindow('cgh', cv2.WINDOW_NORMAL)
            cv2.moveWindow('cgh', 1920, 0)
            cv2.setWindowProperty('cgh', cv2.WND_PROP_FULLSCREEN, cv2.WINDOW_FULLSCREEN)
            cv2.imshow('cgh', GRT_cpu)

            cv2.namedWindow('road lane', cv2.WINDOW_NORMAL)
            cv2.imshow('road lane', output_image)

            # Press Q on keyboard to exit
            if cv2.waitKey(25) & 0xFF == ord('q'):
                break

        else:
            cv2.namedWindow('cgh', cv2.WINDOW_NORMAL)
            cv2.moveWindow('cgh', 1920, 0)
            cv2.setWindowProperty('cgh', cv2.WND_PROP_FULLSCREEN, cv2.WINDOW_FULLSCREEN)
            cv2.imshow('cgh', np.zeros((h * 2, w * 2), dtype=np.uint8))

            cv2.namedWindow('road lane', cv2.WINDOW_NORMAL)
            cv2.imshow('road lane', output_image)

            # Press Q on keyboard to exit
            if cv2.waitKey(25) & 0xFF == ord('q'):
                break

    # Break the loop
    else:
        break

# When everything done, release
# the video capture object
cap.release()

# Closes all the frames
cv2.destroyAllWindows()
```

REFERENCES

1. Kooi F L, Toet A. Visual comfort of binocular and 3D displays[J]. Displays, 2004, 25(2-3): 99-108.
2. Qian N. Binocular disparity and the perception of depth[J]. Neuron, 1997, 18(3): 359-368.
3. Sullivan A. DepthCube solid-state 3D volumetric display[C]//Stereoscopic displays and virtual reality systems XI. SPIE, 2004, 5291: 279-284.
4. Xiao X, Javidi B, Martinez-Corral M, et al. Advances in three-dimensional integral imaging: sensing, display, and applications[J]. Applied optics, 2013, 52(4): 546-560.
5. Gabor D. A new microscopic principle[J]. 1948.
6. Leith E N, Upatnieks J. Reconstructed wavefronts and communication theory[J]. Josa, 1962, 52(10): 1123-1130.
7. Shimobaba T, Nakayama H, Masuda N, et al. Rapid calculation algorithm of Fresnel computer-generated-hologram using look-up table and wavefront-recording plane methods for three-dimensional display[J]. Optics Express, 2010, 18(19): 19504-19509.
8. Lohmann A W, Paris D P. Binary Fraunhofer holograms, generated by computer[J]. Applied optics, 1967, 6(10): 1739-1748.
9. Ogihara Y, Sakamoto Y. Fast calculation method of a CGH for a patch model using a point-based method[J]. Applied optics, 2015, 54(1): A76-A83.
10. Shimobaba T, Nakayama H, Masuda N, et al. Rapid calculation algorithm of Fresnel computer-generated-hologram using look-up table and wavefront-recording plane methods for three-dimensional display[J]. Optics Express, 2010, 18(19): 19504-19509.
11. Shimobaba T, Ito T, Masuda N, et al. Fast calculation of computer-generated-hologram on AMD HD5000 series GPU and OpenCL[J]. Optics express, 2010, 18(10): 9955-9960.
12. Ying C, Pang H, Fan C, et al. New method for the design of a phase-only computer hologram for multiplane reconstruction[J]. Optical Engineering, 2011, 50(5): 055802-055802-6.
13. Makowski M, Sypek M, Kolodziejczyk A, et al. Iterative design of multiplane holograms: experiments and applications[J]. Optical Engineering, 2007, 46(4): 045802-045802-6.
14. Michalkiewicz A, Kujawinska M, Krezel J, et al. Phase manipulation and optoelectronic reconstruction of digital holograms by means of LCOS spatial light modulator[C]//Eighth International Symposium on Laser Metrology. SPIE, 2005, 5776: 144-152.
15. Lucente M E. Interactive computation of holograms using a look-up table[J]. Journal of Electronic Imaging, 1993, 2(1): 28-34.
16. Kim S C, Kim E S. Effective generation of digital holograms of three-dimensional objects using a novel look-up table method[J]. Applied Optics, 2008, 47(19): D55-D62.
17. Jia J, Wang Y, Liu J, et al. Reducing the memory usage for effectivecomputer-generated hologram calculation using compressed look-up table in full-color holographic display[J]. Applied optics, 2013, 52(7): 1404-1412.
18. Pan Y, Xu X, Solanki S, et al. Fast cgh computation using s-lut on gpu[J]. Optics Express, 2009, 17(21): 18543-18555.
19. Azuma R T. A survey of augmented reality[J]. Presence: teleoperators & virtual environments, 1997, 6(4): 355-385.
20. Handbook of augmented reality[M]. Springer Science & Business Media, 2011.
21. Rolland J P, Hua H. Head-mounted display systems[J]. Encyclopedia of optical engineering, 2005, 2: 1-14.
22. Carmigniani J, Furht B, Anisetti M, et al. Augmented reality technologies, systems and applications[J]. Multimedia tools and applications, 2011, 51: 341-377.
23. Bichlmeier C, Wimmer F, Heining S M, et al. Contextual anatomic mimesis hybrid in-situ visualization method for improving multi-sensory depth perception in medical augmented reality[C]//2007 6th IEEE and ACM international symposium on mixed and augmented reality. IEEE, 2007: 129-138.
24. Yeom H J, Kim H J, Kim S B, et al. 3D holographic head mounted display using holographic optical elements with astigmatism aberration compensation[J]. Optics express, 2015, 23(25): 32025-32034.
25. Li G, Lee D, Jeong Y, et al. Holographic display for see-through augmented reality using mirror-lens holographic optical element[J]. Optics letters, 2016, 41(11): 2486-2489.
26. Moon E, Kim M, Roh J, et al. Holographic head-mounted display with RGB light emitting diode light source[J]. Optics express, 2014, 22(6): 6526-6534.
27. Mitsuo Takada, Hideki Ina, and Seiji Kobayashi. "Fourier-transform method of fringe-pattern analysis for computer-based topography and interferometry." JosA 72.1 (1982): 156-160.

28. Blanche, Pierre-Alexandre, and Remington S. Ketchum. "Texas instruments phase light modulator for holography." Digital Holography and Three-Dimensional Imaging. Optica Publishing Group, 2021.
29. Bartlett, Terry A., William C. McDonald, and James N. Hall. "Adapting Texas Instruments DLP technology to demonstrate a phase spatial light modulator." Emerging Digital Micromirror Device Based Systems and Applications XI. Vol. 10932. SPIE, 2019.
30. Fan C, Kong L, Yang B, et al. Design of dual-focal-plane AR-HUD optical system based on a single picture generation unit and two freeform mirrors[C]//Photonics. MDPI, 2023, 10(11): 1192.
31. Skirnewskaja J, Wilkinson T D. Automotive holographic head-up displays[J]. Advanced materials, 2022, 34(19): 2110463.
32. Bartlett T A, McDonald W C, Hall J N. Adapting Texas Instruments DLP technology to demonstrate a phase spatial light modulator[C]//Emerging Digital Micromirror Device Based Systems and Applications XI. SPIE, 2019, 10932: 161-173.
33. Low C Y, Zamzuri H, Mazlan S A. Simple robust road lane detection algorithm[C]//2014 5th international conference on intelligent and advanced systems (ICIAS). Ieee, 2014: 1-4.
34. Bradski G. The opencv library[J]. Dr. Dobb's Journal: Software Tools for the Professional Programmer, 2000, 25(11): 120-123.
35. Blinder D, Nishitsuji T, Kakue T, et al. Analytic computation of line-drawn objects in computer generated holography[J]. Optics Express, 2020, 28(21): 31226-31240.
36. Tang C I, Deng X, Takashima Y. Real-Time CGH Generation by CUDA-OpenGL Interoperability for Adaptive Beam Steering with a MEMS Phase SLM[J]. Micromachines, 2022, 13(9): 1527
37. Lucente M. The first 20 years of holographic video—and the next 20[C]//SMPTE 2nd annual international conference on stereoscopic 3D for media and entertainment. 2011: 21-23.