Optics Communications 573 (2024) 131021

Contents lists available at ScienceDirect

Optics Communications

journal homepage: www.elsevier.com/locate/optcom

Performance evaluation of polygon-based holograms in terms of software, hardware and algorithms

Anuj Gupta [a], Fan Wang [b,*], Bhargab Das [a], Raj Kumar [a], David Blinder [b,c,d], Tomoyoshi Ito [b], Tomoyoshi Shimobaba [b]

[a] CSIR-Central Scientific Instruments Organisation, Chandigarh, 160030, India
[b] Graduate School of Engineering, Chiba University, 1-33 Yayoi-cho, Inage-ku, Chiba, 263-8522, Japan
[c] Department of Electronics and Informatics (ETRO), Vrije Universiteit Brussel (VUB), Pleinlaan 2, B-1050, Brussel, Belgium
[d] IMEC, Kapeldreef 75, B-3001, Leuven, Belgium

ARTICLE INFO

Keywords:
Computer-generated hologram
Polygon-based method
Wavefront recording plane method
Graphics processing unit
Python
MATLAB

ABSTRACT

Computational holography involves creating complex holographic patterns, which is both fundamental and computationally intensive. This process presents significant challenges, particularly in achieving real-time hologram generation. This study presents a thorough comparison and analysis of computational efficiency for computing polygon-based computer-generated holograms (CGH) in terms of programming language (Python and MATLAB), execution hardware (CPU and GPU) and algorithms (interpolation-based and analytical-based). We open-sourced all the codes used for polygonal CGH executed in both MATLAB and Python, offering valuable insights into the performance suitability of different algorithms and languages. Basically, MATLAB demonstrates superior performance over Python, especially for CPU calculations, whereas it performs similarly when utilizing a graphics processing unit (GPU) and an accelerated algorithm like the wavefront recording plane (WRP) method. Analytical-based method and interpolation-based method are not consistently superior; the former performs well when addressing small matrices (e.g., using WRP), while the latter performs well when addressing large matrices.

1. Introduction

Holography is a technology that captures both the amplitude and phase information of objects, enabling the creation of real 3D displays with depth perception [1,2]. Holography has found various applications in art and display, security features on documents, medical imaging for 3D visualization, non-destructive testing, high-density data storage, holographic optical elements in optical systems, scientific visualization for complex structures, educational tools for interactive learning and many more [3–5]. However, the requirement of the presence of a physical object during recording and its static nature hinder its use in real-time dynamic display scenarios. Computer-generated holography overcomes these limitations [6]. Computational holography involves numerical calculation of a hologram by computing the full complex wavefront of a virtual object, based on diffraction theory [6,7]. This digitally created diffractive pattern is named the computer-generated hologram (CGH), and it can be created without a physical object and can adapt to dynamic changes in the displayed object or scene. Subsequently, the holographic reconstruction can be achieved using methods such as spatial light modulators (SLMs) [8], holographic printers [9], or laser lithography system-based fabrication [10]. Despite challenges like the small space bandwidth product (SBP) of the display and low computational efficiency [11], CGH has found applications in various fields [12]. It is used in holographic displays [13–15], beam shaping [16,17], optical metrology of aspheric surfaces [18,19], and advanced holographic projections [20,21], demonstrating its versatility and impact.

CGH methods typically calculate optical field of 3D objects using point-, polygon-, and layer-based algorithms [22]. To accelerate CGH computation, several algorithms have been proposed. These include look-up table (LUT) [23,24] and wavefront recording plane (WRP) [25,26] for point-based holograms. For the polygon-based holograms, interpolation spectra [27,28] and full-analytical spectra [29–32] are relevant methods. Additionally, the fraction method works for layer-based holograms [33–35]. While these algorithms offer various trade-offs between imaging quality and SBP, computational rates still

* Corresponding author.
E-mail address: wangfan@chiba-u.jp (F. Wang).

https://doi.org/10.1016/j.optcom.2024.131021
Received 9 June 2024; Received in revised form 7 August 2024; Accepted 24 August 2024
Available online 28 August 2024

fall short of achieving real-time video CGH [22]. Hardware implementations, such as HOlographic ReconstructioN (HORN) systems [36–43], have also been developed to achieve high-speed CGH calculations. Despite the significant breakthroughs in efficiency and quality achieved by neural holography algorithms in recent years [44,45], they still encounter limitations regarding requirements of large depth of field and field of view [46]. The present study is exclusively focused on the polygon-based approach for generating CGH. It incorporates accelerated algorithms, specifically interpolation spectra and full-analytical spectra. Additionally, a WRP-like method [47] is implemented to further accelerate the computation of CGH in these accelerated algorithms of the polygon-based approach.

While previous works focused on implementing these methods exclusively in MATLAB [32,47,48], the current study emphasizes their implementation in Python and compares performance of the two languages. Python is included in this study because, like MATLAB, it is a powerful and well-known computational tool. Additionally, Python is freely distributed as an open-source package, making it accessible to a wide range of users. This comparison helps CGH learners with a clear understanding of the performance differences in different computational environments and methodologies. Additionally, we have compared the performance evaluation of these two programming languages for a layer-based approach in one of our previous works [49]. In this study, we concentrate on the polygon-based method for CGH generation. This method involves dividing a 3D object into several polygons and then propagating optical field from the object plane to the hologram plane. Compared to the point-based method, where optical fields of all object points are propagated, the polygon-based method is faster because there are fewer polygons than points in an object, resulting in shorter propagation times. Moreover, the polygon-based method provides better resolution than the layer-based method, where optical field of the entire object layer is propagated simultaneously. Since several polygons exist in one layer, the object is better resolved in the polygon-based method. The choice of the polygon-based method is driven by its balanced approach between resolution and computational speed, aligning with the objectives of this study.

Due to their effectiveness, Python and MATLAB are among the most popular coding platforms used for calculating CGH. This study focuses on these programming languages to assess their efficiency and performance in CGH calculations. Python is renowned for its versatility and extensive libraries, particularly in scientific computing and image processing, which contribute to its popularity. On the other hand, MATLAB, a proprietary software package, continues to be a fundamental tool in various scientific and engineering fields. Both platforms have distinct advantages and disadvantages, making these appealing for CGH applications. This study delves into the performance, scalability, and ease of use of Python and MATLAB in calculating CGH matrices. It thoroughly explores the comparative performance of both programming languages in generating holograms. The importance of this comparative analysis lies in its ability to uncover the strengths and limitations of each programming environment when faced with the demanding task of rapid CGH computation. The selection of the polygon-based method forms the foundation for the following comparative analysis between Python and MATLAB in CGH generation. Section 2 discusses the methods or algorithms implemented in this study and outlines the methodology used to formulate the comparative analysis. Section 3 presents the results of comparison of the execution time of the codes, and Section 4 concludes the study.

2. Polygon-based method for computer-generated-hologram

Polygon-based methods represent a significant class of CGH generation techniques that have been developed over the past two decades [31,50]. Numerous advanced algorithms have been proposed to accelerate the execution speed of CGH computation [28,48,51,52] and enhance the realism [53–58] of reconstructed image. As already mentioned, a virtual 3D object is partitioned into distinct polygons, where each polygon represents a segment of the 3D object. Once all object polygons are acquired, spectrum of each polygon is propagated from the object plane to the hologram plane using Eq. (1).

$$F\left(f_x,f_y\right)=\iint_{\Delta} u_0(x,y)\cdot\exp\left[-j2\pi\left(f_x x+f_y y\right)\right]\mathrm{d}x\mathrm{d}y\cdot\exp\left(-j2\pi f_z z_0\right) \quad (1)$$

where $u_0(x,y)$ represents the initial wave field on the polygon surface that is chosen to be triangle in our case because a triangular surface simplifies the implementation while maintaining a good balance between accuracy and computational efficiency, $F\left(f_x,f_y\right)$ represents the spectral distribution of triangle Δ on the hologram plane, $\left(f_x,f_y\right)$ denotes the spatial frequency coordinates corresponding to the global system (x,y,z), $f_z=\sqrt{\frac{1}{\lambda^2}-f_x^2-f_y^2}$, where λ denotes wavelength of light, z_0 denotes the distance between the object plane and the hologram plane, and $\mathrm{d}x,\mathrm{d}y$ signify the sampling intervals in the horizontal and vertical directions of the object plane, respectively.

In the realm of polygon-based methods, the calculation of angular spectrum diffraction Eq. (1) can be performed using either numerical or analytical techniques. As shown in Fig. 1, a spatial triangle (ΔABC) in the global system is tilted to the hologram plane. A fixed right triangle in the local system is defined in Fig. 1(a), which is always related to ΔABC by a 3D affine transformation. Therefore, analytical-based technique can give an analytical spectral equation for ΔABC based on the spectrum of the fixed right triangle. However, numerical-based technique first yields the spectrum of the rasterized triangle of the local system by fast Fourier transform (FFT), and interpolates the spectrum based on the rotated coordinates, as shown in Fig. 1(b). Both methods aim to solve for the spectral distribution of ΔABC in the hologram plane, as given in Eq. (1). Then, the complex hologram pattern $H(x_0,y_0)$ can be obtained by an inverse Fourier transformation as follows:

$$H(x_0,y_0)=\iint F\left(f_x,f_y\right)\exp\left[j2\pi\left(f_x x_0+f_y y_0\right)\right]\mathrm{d}f_x\,\mathrm{d}f_y \quad (2)$$

where (x_0,y_0) denotes the pixel in hologram plane and the wavefield $H(x_0,y_0)$ shows a 3D diffusion in Fig. 1.

These algorithms have been implemented and validated by various researchers such as Matsushima [10,27,28,57], Zhang [52,54,56,59], Dong [55], Wang [48,50,60] using different computational platforms and programming languages. In order to comprehensively assess their performance, we have chosen several classical polygon-based methods for CGH generation and performed the comparative analysis focusing on algorithms, hardware environment, and software environment.

2.1. Algorithm

The algorithm used for calculation plays a pivotal role in determining the performance of MATLAB and Python in CGH generation. This subsection briefly discusses two methods: fully analytical and interpolation methods.

By leveraging the geometric properties of triangles, an arbitrary spatial triangle can be transformed into a fixed triangle through rotation or affine transformation [29,30], as depicted in Fig. 1(a). This transformation enables obtaining spectral distributions along the propagation direction using an analytical function. Further advancements allow for incorporating texture [53,54,56] and lighting effects [32,55,58] using analytical functions. The full analytical method provides a spectral equation with smoothly shaded triangles by inputting the vertex coordinates and vertex normals [32]. This method employs a novel smooth shading approach for polygon-based holograms, which offers a fully analytical spectral formula based on the Blinn-Phong reflection model [61,62].

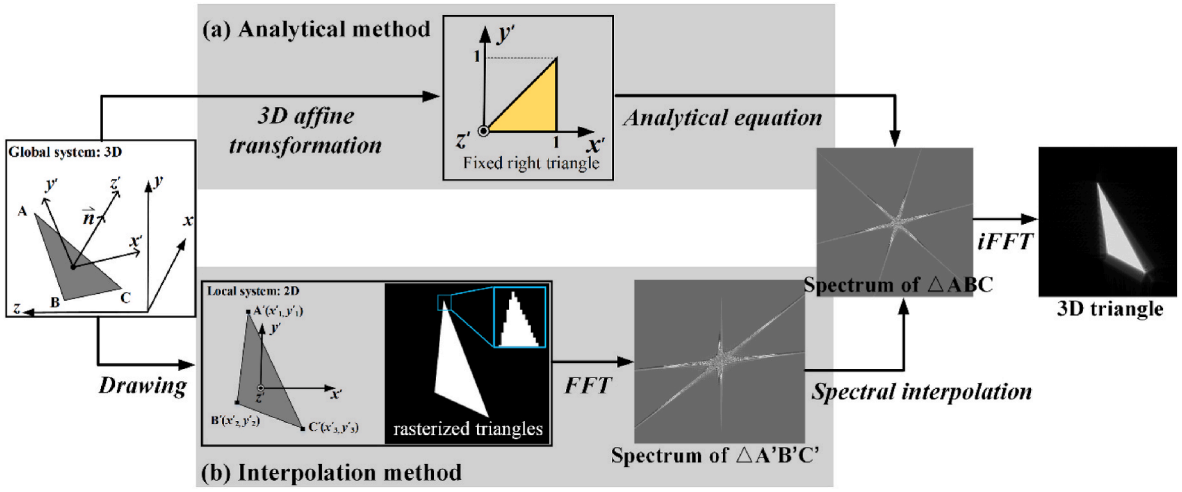

Fig. 1. Illustration showing two distinct approaches employed in polygon-based method for CGH generation.

In terms of computational efficiency, the fully analytical method stands out by requiring the computation of only full analytical formulas. The method is further accelerated in CGH calculation by implementing two approximations corresponding to specular reflection and diffused reflection [60]. With this method, reconstructed images exhibit a highly realistic illuminated scene and smooth surface even with a small number of triangles. On the contrary, the interpolation-based method relies on purely numerical calculations to generate holograms. This method involves several steps: first, a rasterized triangle is drawn on the local system; next, a FFT is performed to obtain the spectrum distributed in the local system; and finally, the local spectrum is rotated to the global system by spectral interpolation [10,27,28,51,57], as depicted in Fig. 1 (b). The unrestricted nature of the drawing step allows for arbitrary amplitudes of the rasterized triangles, facilitating the easy rendering of texture and lighting information. While the operations of FFT and interpolation are known to be computationally expensive, the interpolation-based method may still offer advantages, especially for large matrix operations. This is because linear interpolation, which is used in this method, involves simple polynomial operations (e.g., multiplication and addition). In contrast, analytic equations often require more complex operations such as division and exponentiation, which can be more computationally intensive, as can be observed in the results.

2.2. Recording plane

Optical diffraction is a process where optical field of a single unit (such as a point or polygon) affects multiple pixels within a region determined by the diffraction angle. The farther the light source is from the recording plane, the larger the area affected, leading to significant computational effort. To reduce this computational cost, an effective approach is to narrow down the region of interest (ROI) by introducing an intermediary plane called the wavefront recording plane (WRP) near the source unit (polygon), as shown in Fig. 2(a). In contrast, the hologram plane is considered the recording plane, known as the image hologram (IH) [63] as shown in Fig. 2(b). Here, operations are performed on each triangle using the same matrix as the hologram. However, the WRP operates on smaller matrices. Since the ROI varies for each triangle, the WRP needs to resample in the frequency domain and convert each to the spatial domain using an inverse FFT. Subsequently, the wavefront is spliced on the WRP. Therefore, the WRP requires N_t more operations of FFT compared to HRP, where N_t is the number of triangles [47].

2.3. Hardware configuration

The hardware configuration plays a crucial role in determining the performance of MATLAB and Python for CGH generation, especially when comparing Central Processing Unit (CPU) and Graphics Processing Unit (GPU) utilization. CPUs and GPUs are fundamental components of a

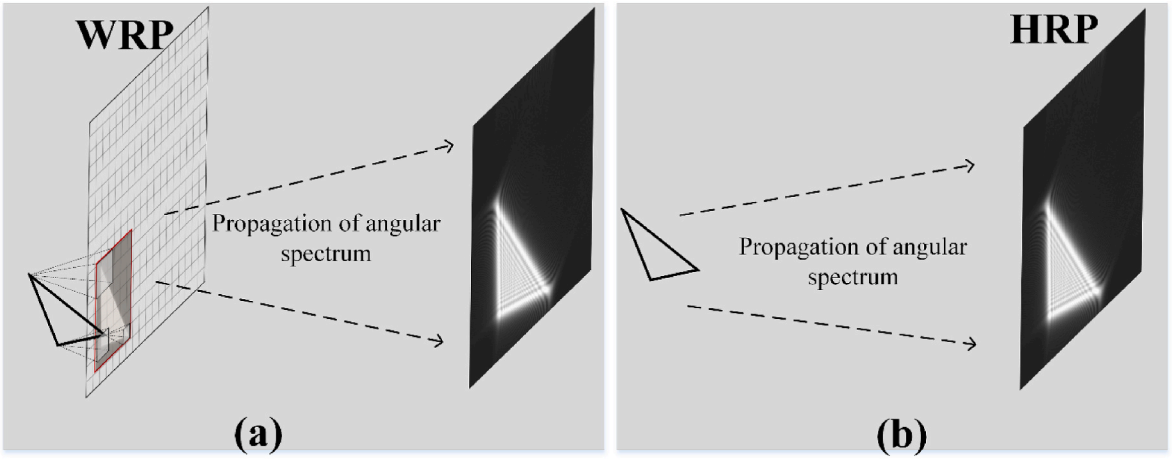

Fig. 2. Illustration of the (a) wavefront recording plane (WRP) and (b) image hologram (IH).

computer, but their architecture and function differ. CPUs are designed for general-purpose computing tasks and are characterized by a few powerful cores. They excel at handling complex tasks that require a high level of control and synchronization, making them ideal for running MATLAB and Python codes that involve intricate computations. However, these CPU cores are optimized for sequential processing. Sequential programming involves a single core executing a set of instructions while all other cores in the CPU remain idle. This idle state of resources can lead to inefficiencies and increased processing time, causing suboptimal performance. Thus, CPUs may struggle with parallel processing tasks, which can limit their performance in specific scenarios.

On the other hand, GPUs are specifically designed for parallel processing, with hundreds or even thousands of smaller cores that can handle multiple tasks simultaneously. The parallel programming gave output in much less time by using all the available cores simultaneously. This architecture makes GPUs highly efficient for tasks that can be parallelized, such as graphics rendering. In the context of CGH generation, utilizing the GPU can significantly accelerate the computation process, especially for algorithms that can be parallelized effectively. The comparison between CPU and GPU performance for MATLAB and Python in CGH generation showcases the benefits of leveraging GPU's parallel processing capabilities. While CPUs remain crucial for handling complex, sequential tasks, GPUs offer a compelling solution for accelerating computations that can be parallelized, leading to significant performance gains in CGH generation and other computational tasks.

The polygon-based approach is suitable for implementation in Python and MATLAB, forming the basis for our comparative analysis of CGH generation between the two languages. Therefore, we developed an algorithm for CGH generation sequentially using a polygon-based technique. The CGH calculation was performed in series, utilizing only one CPU thread. Throughout the process of CGH generation, we ensured consistency in the programming environment, implementation of mathematical operations and logic, and the hardware used for CGH generation. The same target object files were used as input, and the same values of holographic parameters were provided for both Python and MATLAB implementations. The implementation of underlying physics is cross-verified through the numerical reconstruction of the output CGH, thus confirming the consistency of CGHs synthesized by different algorithms and different languages. Numerical reconstruction involves simulating the path of light waves backward from the CGH plane to the object plane. This process is crucial as it allows for the recovery of information about the object's spatial structure, aiding in accurately recreating the original three-dimensional scene captured in the hologram. In optical reconstruction, the backward propagation of the CGH is passed through a 4-F system used to filter the DC light and then the reconstructed images are captured by a camera. This completely restores the propagation process in the optical reconstruction, as shown in the schematic of optical setup shown in Fig. 3.

After benchmarking by numerical reconstruction, the results of the execution times for both implementations were recorded after executing the code separately in MATLAB and Python. The roadmap given in Table 1 is followed for the systematic comparison of the performance of both the programming languages for the polygon-based method in CGH,

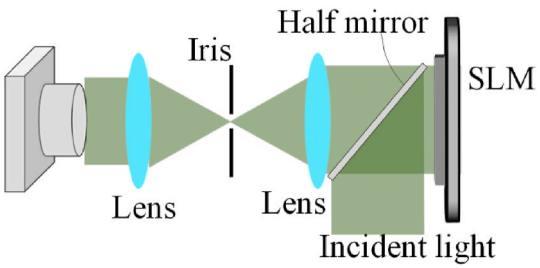

Fig. 3. Experimental arrangement for optical reconstruction of CGH.

Table 1
Roadmap for the comparative analysis.

S. No.	Algorithm (A)	Recording plane (B)	Hardware (C)	Programming Language (D)
1	Analytical-based (A1)	Holographic recording plane (B1)	CPU (C1)	MATLAB
2	Interpolation-based (A2)	Wavefront recording plane (B2)	GPU (C2)	Python

considering different algorithms, recording planes, and hardware configurations.

The polygon-based method is employed in both Python and MATLAB for two algorithms: analytical calculation-based (A1) and interpolation-based (A2). The code is designed to record the diffraction pattern directly onto the holographic recording plane (B1), with calculations executed sequentially on the CPU (C1). Each simulation model is named based on the algorithm, recording plane, and hardware used. For example, the A1+B1+C1 model represents CGH computation on the CPU using the analytical calculation-based algorithm and recording directly onto the hologram plane. The initialization time and computation time are recorded for each case to analyze the performance of both programming languages for the polygon-based method. This process is repeated for six out of eight cases, including both algorithms (A1/A2) on both recording planes (B1/B2) using the CPU (C1), and the A1 algorithm on the GPU (C2). However, MATLAB and Python embedded interpolation functions running on the GPU do not support extrapolation for out-of-bounds inputs., so the interpolation-based algorithm is not implemented on the GPU. The results of all simulations are discussed in the following section.

3. Results

This comparative study can offer valuable insights into the suitability, feasibility, and efficiency of Python and MATLAB for CGH generation using polygon-based methods, and objectively evaluate the performance of multiple algorithms in the same computational environment. Professionals and researchers working in fields like optics, holography, and computational imaging stand to benefit, as they may gain a deeper understanding of the trade-offs and advantages of different polygon-based algorithms running in each programming languages. This understanding will enable them to make informed decisions when choosing algorithms and programming environments for their specific CGH applications. For this analysis, we utilized an object file containing the coordinates of a 3D object created using Blender software. The three objects viz. "teapot", "hand" and "manbody" are chosen. These objects consist of 1032, 4605 and 12044 faces respectively that are calculated in the algorithm after the back-face culling. We have run the simulations for these three objects and generated their CGHs using respective.obj files using the polygon-based method.

The pixel pitch was set at 3.74 μm, the laser's wavelength at 532.0 nm, and the diffraction distance between the object plane and the CGH plane at 30 mm. The resolution of the object or hologram was set to 1024 × 1024. It is worth noting that the physical dimensions of the object or hologram can be determined by multiplying the pixel pitch of the hologram by its resolution, resulting in approximate physical dimensions of 6.5 mm × 6.5 mm. It is crucial to specify the hardware platform, as the execution time of the CGH codes is dependent on it. The system employed for this study includes an Intel CoreTMi9-12900 processor, 16 cores, 24 threads, and 32 GB of RAM. It is complemented by an nVIDIA Quadro T1000 graphics card featuring 896 CUDA cores. The operating system used is Windows 11. The holographic image of the object is numerically reconstructed using the angular spectrum method.

Fig. 4(a)–(b) depicts the amplitude part of the computed complex CGH using the analytical method and the corresponding numerically

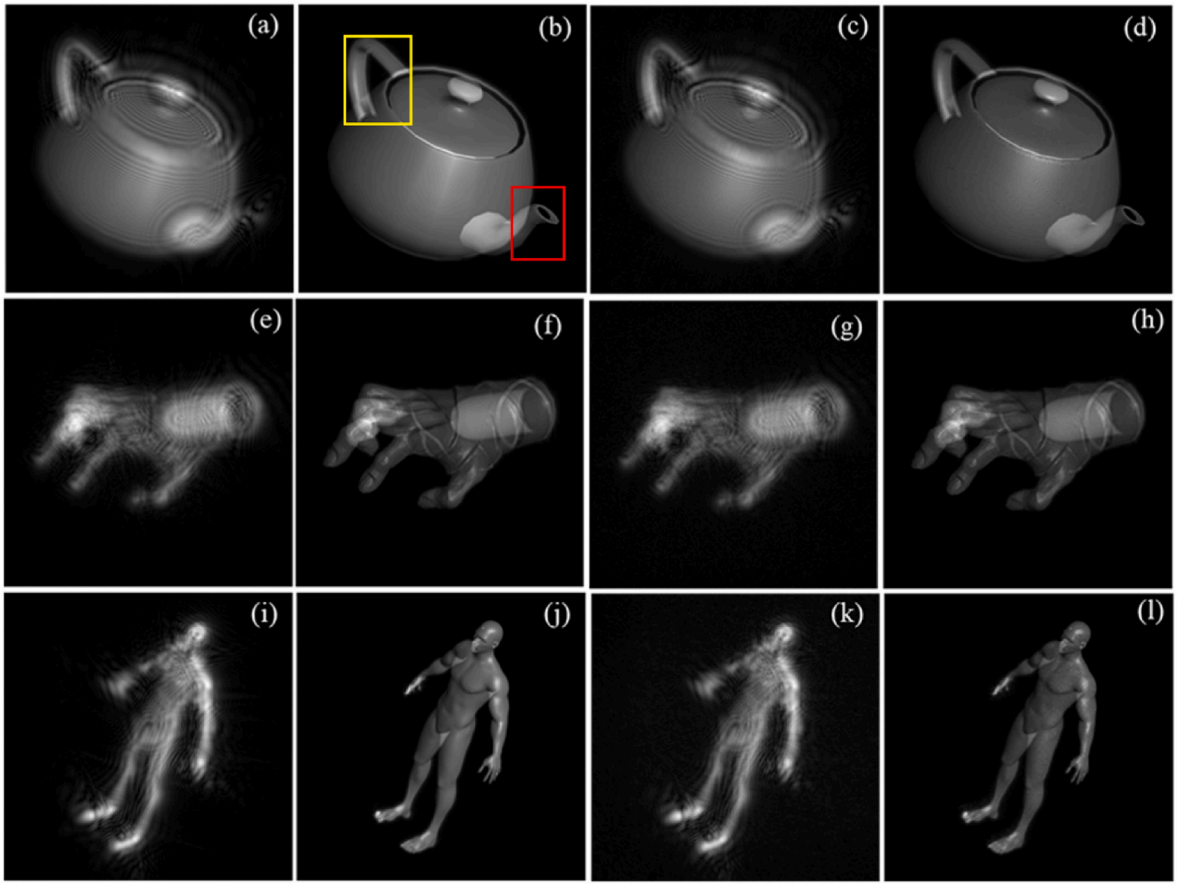

Fig. 4. (a, e, i) Amplitude of the complex hologram generated using analytical method for object 1 – teapot, object 2 – hand and object 3 – manbody respectively (b, f, j) Numerically reconstructed object from the CGH created using analytical method for objects 1, 2 and 3 respectively (c, g, k) Amplitude of the complex hologram generated using interpolation method for objects 1, 2 and 3 respectively (d, h, l) Numerically reconstructed objects from the CGH created using interpolation method for objects 1, 2 and 3 respectively.

reconstructed result, where the front end of the teapot is in focus, respectively. Similarly, Fig. 4(c)–(d) shows the amplitude part of the computed complex CGH using the interpolation method and the corresponding numerically reconstructed result, where focus is on the front end of the teapot, respectively. The teapot we used for computing has a depth range of 30 mm–33.2 mm, i.e., the depth is only 3.2 mm, resulting in less noticeable de-focus and in-focus. However, the enlarged portions of the front part and the rear part are shown in Fig. 5, which proves that the focused front part is clear and the out-of-focus rear part is blurry. The

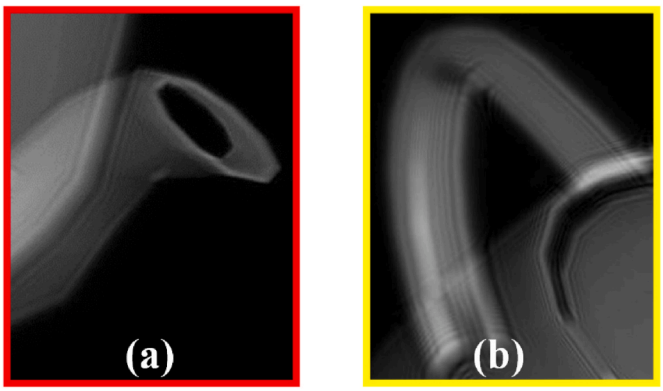

Fig. 5. (a) Focused front part and (b) blurry rear part of the numerically reconstructed teapot object, illustrating the difference in focus due to the object's limited depth range.

results from all the different simulation models are in complete agreement with each other. Notably, the results remain consistent across the use of GPU, the incorporation of the WRP method, and both programming languages.

We analyzed the performance differences between Python and MATLAB, considering the impact of hardware (CPU vs. GPU), algorithms (interpolation-based vs. analytical-based), and the recording plane (HRP vs. WRP) on computation time. The analysis is done for three different objects viz. "teapot", "hand", and "manbody". As already mentioned, these objects consist of 1032, 4605 and 12044 faces i.e. no of polygons, respectively that are calculated in the algorithm after the back-face culling. We carefully observed the data initialization and execution times for calculating CGH using different methods implemented in the two programming languages. The initialization time is the time it takes for the code to be set up before the main calculation loop. The variation in initialization time could be due to factors like system load, caching effects, or the complexity of the initialization tasks. The presented initialization time encompasses tasks such as importing necessary libraries, declaring functions and variables, etc. It also includes preprocessing steps for the 3D data, such as reading coordinates from the object file, rotating and resizing all data according to the global coordinate system, and culling the back faces from the hologram plane. On the other hand, the execution time is the time it takes for the code to complete the main calculation loop. It signifies the actual duration spent on computing the CGH. The variation in execution time could be due to the complexity of the calculations, data size, or other factors related to the computation itself. Thus, it includes the time consumed in the diffraction spectrum calculation for each of the vertices and faces of the

object. The times are measured by running the code five times and averaging them. A comparative analysis of the results obtained for different models and different objects is detailed in Table 2.

Table 2
Initialization and computation times for different models[a].

Object 1 – Teapot (No. of Polygons Calculated – 1032)					
		Initialization Time (in seconds)		Computation Time (in seconds)	
Sr. No.	Method	MATLAB (t_{Mi})	Python (t_{pi})	MATLAB (t_{Me})	Python (t_{pe})
1.	Analytical on CPU (A1+B1+C1)	0.022	0.641	204.45	1176.0
2.	Analytical on GPU (A1+B1+C2)	2.76	1.404	69.95	92.18
3.	Analytical + WRP on CPU (A1+B2+C1)	0.021	0.075	0.752	1.59
4.	Analytical + WRP on GPU (A1+B2+C2)	0.185	0.075	1.96	1.44
5.	Interpolation on CPU (A2+B1+C1)	0.021	0.118	73.5	938.9
6.	Interpolation + WRP on CPU (A2+B2+C1)	0.023	0.093	4.12	4.0
Object 2 – Hand (No. of Polygons Calculated – 4605)					
		Initialization Time (in seconds)		Computation Time (in seconds)	
Sr. No.	Method	MATLAB (t_{Mi})	Python (t_{pi})	MATLAB (t_{Me})	Python (t_{pe})
1.	Analytical on CPU (A1+B1+C1)	0.0485	0.625	950.634	5291.477s
2.	Analytical on GPU (A1+B1+C2)	0.0592	0.8108	291.707	413.079
3.	Analytical + WRP on CPU (A1+B2+C1)	0.042	0.573	1.859	3.0469
4.	Analytical + WRP on GPU (A1+B2+C2)	0.0328	0.6824	9.337	6.9404
5.	Interpolation on CPU (A2+B1+C1)	0.0612	0.436	332.224	3312.91
6.	Interpolation + WRP on CPU (A2+B2+C1)	0.0412	0.3991	8.1112	7.5289
Object 3 – Man Body (No. of Polygons Calculated – 12044)					
		Initialization Time (in seconds)		Computation Time (in seconds)	
Sr. No.	Method	MATLAB (t_{Mi})	Python (t_{pi})	MATLAB (t_{Me})	Python (t_{pe})
1.	Analytical on CPU (A1+B1+C1)	0.0905	1.575	2459.65	13711.684
2.	Analytical on GPU (A1+B1+C2)	0.638	1.781	844.243	1084.796
3.	Analytical + WRP on CPU (A1+B2+C1)	0.151	1.538	4.413	5.732
4.	Analytical + WRP on GPU (A1+B2+C2)	0.075	1.489	19.530	16.320
5.	Interpolation on CPU (A2+B1+C1)	0.1215	1.133	830.624	7665.472
6.	Interpolation + WRP on CPU (A2+B2+C1)	0.086	1.081	13.981	13.139

[a] All the source codes associated with these six models for three different objects are included in the supplementary data with the filenames corresponding to the names listed in parentheses in Column 1 of Table 2.

4. Discussions

4.1. On the aspect of software

The comparative analysis reveals nuanced performance differences between MATLAB and Python for CGH generation. While MATLAB generally demonstrates superior performance, particularly in initialization and computation times for most methods, Python shows competitive performance in certain scenarios, such as GPU utilization and specific WRP implementations. When using the analytical method on CPU (A1+B1+C1), i.e., No. 1, MATLAB outperforms Python in both initialization and computation times. The initialization time is significantly lower in MATLAB (0.022 s) compared to Python (0.641 s) in case of teapot. The negligible initialization time taken by MATLAB indicates its more pronounced superiority during the initialization phase. Similarly, MATLAB's computation time is much lower (204.45 s) than Python's (1176.0 s). Thus, Python's execution time is more than five times the execution time MATLAB takes. Similarly, for the interpolation method on CPU (A2+B1+C1), i.e, No. 5, Python has a slightly higher initialization time compared to MATLAB. However, Python's computation time is significantly higher than MATLAB's. The difference in computation time is particularly notable, with MATLAB taking 73.5 s compared to Python's 938.9 s. Python's execution time is more than twelve times the execution time taken by MATLAB. The similar relative performance is observed in other two objects i.e the calculation time increases in the same manner as the number of polygons increases.

The large discrepancy in execution times between Python and MATLAB shown in results of No.1 (A1+B1+C1) and No. 5 (A2+B1+C1) indicates that Python's CPU performance is generally poorer than MATLAB, which can be attributed to several factors, including numerical computation libraries, code compilation, and memory management. MATLAB's optimized numerical computation libraries are highly efficient for matrix operations and linear algebra calculations, contributing to its faster execution time. In particular, MATLAB is superior when dealing with intensive computations such as large matrices, while the gap between the two narrows significantly when dealing with small matrices, as show in results of No. 3 (A1+B2+C1) and No. 6 (A2+B2+C1) when using WRP. Further, the JIT (just-in-time) compiler in MATLAB further enhances performance by optimizing code at runtime. In contrast, Python lacks a comparable JIT compiler, leading to slower execution times, particularly for computationally intensive tasks like CGH generation. However, Python can overcome this limitation with the use of Numba, a JIT compiler that compiles code at native machine code speed, improving performance for parts of the program. It is worth noting that not all codes can be optimized using Numba. MATLAB's memory management is also optimized for numerical computations, enabling efficient handling of large datasets. Python's memory management, while flexible, may introduce overhead that can impact performance, especially for large-scale computations.

The comparative analysis for all the three objects in six different models presented in Table 2 demonstrates MATLAB's consistent advantage in execution times, particularly on the CPU. However, this advantage diminishes for WRP or GPU models, indicating Python's competitive performance in specific scenarios. Specifically, Python shows comparable or slightly better initialization and computation times than MATLAB in cases such as analytical with WRP on GPU and interpolation with WRP on CPU. When utilizing a GPU in the analytical method, the performance gap between MATLAB and Python narrows, with MATLAB still performing better, albeit less significantly. Notably, this study did not include the GPU implementation of the interpolation-based algorithm.

4.2. On the aspect of algorithms

The comparison of model No. 1 and model No. 5 in Table 2 shows that the analytical method takes more computation time than the

interpolation-based method. The time difference between the analytical and interpolation-based methods in CGH generation can be attributed to their underlying principles and computational complexity. The analytical method calculates the hologram by directly applying diffraction theory to each polygon in the object, requiring intensive mathematical computations. The analytical spectral equation typically involves more mathematical operations, e.g., many multiplication, division and exponential operations, leading to longer computation times. On the other hand, the linear interpolation simplifies the computation by approximating the hologram values based on the surrounding points, and its operations are only the addition and multiplication of polynomials. This reduces the number of mathematical operations required and makes the process faster. However, the comparison between No. 3 and No. 6 in Table 2, shows the analytical-based and interpolation-based performance is reversed due to the use of a WRP method. This indicates that the reduction of kernel size using WRP is more favorable for the analytical-based method, but not so much for the interpolation-based method because of the different mathematical operations.

The cause is analyzed as follows. The simple operations used in the interpolation method calculate each pixel value faster than the complex operations used in the analytical method. However, the interpolation method first requires searching for the values of nearby points, which would be an additional overhead relative to the analytical method. For a large-scale matrix, the additional overhead to search nearby points in the interpolation method is negligible compared to operations itself, and the computation speed depends on the complexity of operations, thus the interpolation-based is superior to the analytical-based method, as shown in results of No. 1 and No. 5. For the small-scale matrix, the additional overhead of searching in the interpolation method is no longer economical relative to the operation itself, thus the analytical-based is superior to the interpolation-based method, as shown in results of No. 3 and No. 6. It is important to mention here that the bilinear interpolation method is generally used to estimate the hologram values based on nearby points. Therefore, this method sacrifices some precision for faster computation, resulting in some minor noise on the object surface, as shown in Fig. 4(d). The same type of artifacts is present for the "hand" (Object 2) and the "manbody" (Object 3). For Object 2, these artifacts are noticeable in Fig. 4(h), where the minor noise from the interpolation method appears on the object's surface. In contrast, the analytical method used in Fig. 4(f) does not exhibit such noise. Similarly, for Object 3, the artifacts are visible in Fig. 4(l) due to the interpolation method, while the results from the analytical method in Fig. 4(j) are free of such artifacts. However, this noise can be made negligible by adding a random phase to each triangle during computation.

Moreover, the use of WRP-like method affects the performance between different algorithms (interpolation-based and analytical-based) in addition to the differences between software languages. For example, No. 1 (A1+B1+C1) using the HRP shows that MATLAB is 5 times more efficient than Python, while No. 3 (A1+B2+C1) using the WRP significantly reduce the difference between both languages. Also, No. 5 (A2+B1+C1) and No. 6 (A2+B2+C1) further confirm this fact. This is because the use of WRP and HRP determines the size of computational matrix, and two languages address large and small matrices differently in efficiency due to computational libraries, as mentioned in Section 4.1.

4.3. On the aspect of hardware

Thanks to multi-thread processing, GPU can perform simple operations much more efficiently than CPU, as shown in the comparison between No. 1 (A1+B1+C1) and No. 2 (A1+B1+C2). Calling the GPU library in MATLAB and Python usually involves transferring data such as matrices to the GPU from the CPU and then assigning them to each thread in the GPU for parallel computation, so the workload includes data transfer in addition to computation. Therefore, based on the comparison between No. 1 and No. 2, MATLAB using a GPU with 896 CUDA cores is only 3 times faster than using a CPU, and Python using the GPU is 12 times faster than using the CPU. Additionally, due to the reduced matrix size using WRP, the comparison of No. 3 (A1+B2+C1) using CPU and No. 4 (A1+B2+C2) using GPU shows that MATLAB is even less efficient on GPU than on CPU, because the overhead of data transfer in computing small matrices accounts for a larger percentage such that the total time spent on the GPU is longer.

On the other hand, MATLAB is 5 times faster than Python in No. 1 using the CPU, while the gap narrows to 1.3 times in No. 2 using the GPU. Similarly, based on the use of WRP, No. 3 using the CPU shows MATLAB is faster than Python, while No. 4 using the GPU shows the opposite efficiency. This suggests that the performance of Python and MATLAB executed on the GPU converges, while the performance of the two languages on the CPU varies greatly. Since the interpolation-based method did not use a GPU implementation, it is not able to compare with No. 5 and No. 6 which use a CPU; however, we can foresee that the GPU implementation of the interpolation-based method will also be several times faster, but not equal to the number of parallel threads because of non-negligible overhead of data transfer.

5. Conclusion

This study aims to provide a deeper understanding of the advantages and disadvantages of different software, hardware and algorithms, assisting researchers working in computational holography in making informed decisions when choosing a programming environment for their specific CGH applications. Through numerical computations for different polygon-based CGH algorithms, we highlighted the trade-offs between Python and MATLAB, two widely used programming languages, and between CPU and GPU. Among the two main classes of polygon-based methods, interpolation-based is more efficient than analytical-based when computed an image hologram, whereas they show opposite performance when computed on the wavefront recording plane, no matter if they are computed in MATLAB and Python. This is because WRP changes the amount of computation by matric size, and the complex operations of the analytic equations are more favorable for small matrices, while the simple interpolation operations are more favorable for large matrices.

In order to ensure the relevance and practicality of the findings, the same algorithm is implemented while developing the code for two programming languages and for two hardware. Our findings revealed that MATLAB outperforms Python in terms of time, particularly for CPU calculations. However, both languages performed similarly when utilizing a GPU or implementing an accelerated algorithm like the wavefront recording plane method. Moreover, MATLAB and Python exhibited comparable image reconstruction accuracy. When choosing between Python and MATLAB for polygon-based CGH generation, it is essential to consider computational performance and other practical factors like ease of use, available libraries, and licensing costs. While MATLAB may offer better computational performance, Python's versatility, open-source nature, and extensive libraries make it a preferred choice for developers and researchers prioritizing flexibility and accessibility. It should also be mentioned here that the efficiency enhancements of the code are possible in both languages. In a nutshell, the choice between the two depends on the specific requirements of the task. If performance and efficiency are critical, especially for numerical computations, MATLAB may be the better choice. However, if flexibility, ease of use, and a larger ecosystem of libraries are important factors, Python remains a strong contender.

Funding

The authors express sincere appreciation to the Japan Society for the Promotion of Science (P22752, P23378, 22H03607, 19H01097) and IAAR Research Support Program, Chiba University for their invaluable support. The authors also acknowledge CSIR, India for their financial support under project number MLP2014 to carry out this research work.

Conflict of interests

The authors declare no conflict of interests.

CRediT authorship contribution statement

Anuj Gupta: Writing – original draft, Software, Methodology. **Fan Wang:** Writing – review & editing, Methodology. **Bhargab Das:** Writing – review & editing, Investigation. **Raj Kumar:** Writing – review & editing, Formal analysis. **David Blinder:** Writing – review & editing. **Tomoyoshi Ito:** Supervision, Project administration. **Tomoyoshi Shimobaba:** Writing – review & editing, Supervision, Project administration.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

I have shared the link to my code at the attach file step

Appendix A. Supplementary data

Supplementary data to this article can be found online at https://doi.org/10.1016/j.optcom.2024.131021.

References

[1] D. Gabor, Nature 161 (1948) 777–778.
[2] C. Slinger, C. Cameron, M. Stanley, Computer (Long. Beach. Calif) 38 (2005) 46–53.
[3] P. Hariharan, Optical Holography : Principles, Techniques, and Applications, Cambridge University Press, New York, NY, USA, 1996.
[4] B.J. Thompson, Reports Prog. Phys. 41 (1978) 633–674.
[5] R. Kumar, G. Dwivedi, Eng. Res. Express 5 (2023) 032005.
[6] G. Tricoles, Appl. Opt. 26 (1987) 4351.
[7] W.J. Dallas, in: T. Poon (Ed.), Digit. Hologr. Three-Dimensional Disp., Springer US, Boston, MA, 2006, pp. 1–49.
[8] N. Savage, Nat. Photonics 3 (2009) 170–172.
[9] A.V. Morozov, A.N. Putilin, S.S. Kopenkin, Y.P. Borodin, V.V. Druzhin, S. E. Dubynin, G.B. Dubinin, Opt Express 22 (2014) 2193.
[10] H. Nishi, K. Matsushima, S. Nakahara, Appl. Opt. 50 (2011) H245.
[11] L. Onural, F. Yaraş, Hoonjong kang, Proc. IEEE 99 (2011) 576–589.
[12] K. Murano, T. Shimobaba, A. Sugiyama, N. Takada, T. Kakue, M. Oikawa, T. Ito, Comput. Phys. Commun. 185 (2014) 2742–2757.
[13] P. St-Hilaire, S.A. Benton, M.E. Lucente, M. Lou Jepsen, J. Kollin, H. Yoshikawa, J. S. Underkoffler, in: S.A. Benton (Ed.), Pract. Hologr vol. IV, SPIE, 1990, pp. 174–182.
[14] J.Y. Son, H. Lee, B.R. Lee, K.H. Lee, Proc. IEEE 105 (2017) 789–804.
[15] C. Chang, K. Bang, G. Wetzstein, B. Lee, L. Gao, Optica 7 (2020) 1563.
[16] H. Hagino, S. Shimizu, H. Ando, H. Kikuta, Precis. Eng. 34 (2010) 446–452.
[17] T. Shimobaba, T. Kakue, Y. Endo, R. Hirayama, D. Hiyama, S. Hasegawa, Y. Nagahama, M. Sano, M. Oikawa, T. Sugie, T. Ito, Opt Express 23 (2015) 17269.
[18] F. Li, J. Zhao, R. Li, B. Zhang, L. Zheng, X. Zhang, in: Y. Zhang, J. Sasián, L. Xiang, S. To (Eds.), 2010, p. 765643.
[19] J. Burge, S. Ament, C. Zhao, in: Opt. Des. Fabr. Congr. 2023 (IODC, OFT), Optica Publishing Group, 2023, p. OW3B.5.
[20] E. Buckley, J. Disp. Technol. 7 (2011) 135–140.
[21] M. Makowski, I. Ducin, K. Kakarenko, J. Suszek, M. Sypek, A. Kolodziejczyk, Opt Express 20 (2012) 25130.
[22] E. Sahin, E. Stoykova, J. Mäkinen, A. Gotchev, ACM Comput. Surv. 53 (2021) 1–35.
[23] S.-C. Kim, E.-S. Kim, Appl. Opt. 47 (2008) D55.
[24] P.W.M. Tsang, T.-C. Poon, Y.M. Wu, Photonics Res 6 (2018) 837.
[25] T. Shimobaba, N. Masuda, T. Ito, Opt. Lett. 34 (2009) 3133.
[26] P.W.M. Tsang, T.-C. Poon, Chinese Opt. Lett. 11 (2013) 10902–10908.
[27] K. Matsushima, Appl. Opt. 44 (2005) 4607–4614.
[28] K. Matsushima, S. Nakahara, Appl. Opt. 48 (2009) H54.
[29] H. Kim, J. Hahn, B. Lee, Appl. Opt. 47 (2008) D117.
[30] L. Ahrenberg, P. Benzie, M. Magnor, J. Watson, Appl. Opt. 47 (2008) 1567–1574.
[31] Y. Zhang, H. Fan, F. Wang, X. Gu, X. Qian, T.-C. Poon, Appl. Opt. 61 (2022) B363.
[32] F. Wang, H. Shiomi, T. Ito, T. Kakue, T. Shimobaba, Opt. Lasers Eng. 160 (2023) 107235.
[33] J.-S. Chen, D.P. Chu, Opt Express 23 (2015) 18143.
[34] Y. Zhao, L. Cao, H. Zhang, D. Kong, G. Jin, Opt Express 23 (2015) 25440.
[35] Y. Yao, Y. Zhang, Q. Fu, J. Duan, B. Zhang, L. Cao, T. Poon, Opt. Lett. 49 (2024) 1481.
[36] T. Ito, T. Yabe, M. Okazaki, M. Yanagi, Comput. Phys. Commun. 82 (1994) 104–110.
[37] T. Ito, H. Eldeib, K. Yoshida, S. Takahashi, T. Yabe, T. Kunugi, Comput. Phys. Commun. 93 (1996) 13–20.
[38] T. Shimobaba, N. Masuda, T. Sugie, S. Hosono, S. Tsukui, T. Ito, Comput. Phys. Commun. 130 (2000) 75–82.
[39] T. Shimobaba, S. Hishinuma, T. Ito, Comput. Phys. Commun. 148 (2002) 160–170.
[40] N. Masuda, K. Yoshimura, A. Shiraki, T. Shimobaba, T. Sugie, 13 (2005) 1923–1932.
[41] Y. Ichihashi, H. Nakayama, T. Ito, N. Masuda, T. Shimobaba, A. Shiraki, T. Sugie, Opt Express 17 (2009) 13895.
[42] N. Okada, D. Hirai, Y. Ichihashi, A. Shiraki, T. Kakue, T. Shimobaba, N. Masuda, T. Ito, in: Int. Disp. Work., 2012.
[43] T. Nishitsuji, Y. Yamamoto, T. Sugie, T. Akamatsu, R. Hirayama, H. Nakayama, T. Kakue, T. Shimobaba, T. Ito, Opt Express 26 (2018) 26722.
[44] L. Shi, B. Li, C. Kim, P. Kellnhofer, W. Matusik, Nature 591 (2021) 234–239.
[45] J. Wu, K. Liu, X. Sui, L. Cao, Opt. Lett. 46 (2021) 2908.
[46] C. Edwards, Commun. ACM 64 (2021) 14–16.
[47] F. Wang, D. Blinder, T. Ito, T. Shimobaba, Opt Express 31 (2023) 1224.
[48] F. Wang, T. Ito, T. Shimobaba, Photonics Res 11 (2023) 313.
[49] A. Gupta, B. Das, R. Kumar, Performance evaluation of Python and MATLAB for CGH generation using layer-based approach, J. Opt. (2024), https://doi.org/10.1007/s12596-024-01735-y.
[50] F. Wang, in: Hardw. Accel. Comput. Hologr., Springer Nature Singapore, Singapore, 2023, pp. 207–226.
[51] F. Wang, T. Ito, Opt. Lett. 48 (2023) 3339–3342.
[52] H. Fan, B. Zhang, Y. Zhang, F. Wang, W. Qin, Q. Fu, T.-C. Poon, Appl. Sci. 12 (2022) 6873.
[53] Y.-M. Ji, H. Yeom, J.-H. Park, Opt Express 24 (2016) 28154.
[54] Q. Fu, Y. Zhang, B. Zhang, W. Qin, X. Gu, T.-C. Poon, Opt Express 31 (2023) 24537.
[55] J. Dong, B.-R. Yang, Z. Qin, Opt Express 31 (2023) 14821.
[56] W. Qin, Q. Fu, Y. Zhang, B. Zhang, P. Wang, T.-C. Poon, X. Gu, J. Opt. Soc. Am. A 41 (2024) A32.
[57] O. Kunieda, K. Matsushima, Appl. Opt. 58 (2019) G104.
[58] H.-J. Yeom, J.-H. Park, Opt Express 24 (2016) 19801.
[59] Y. Zhang, H. Fan, F. Wang, X. Gu, X. Qian, T.-C. Poon, Appl. Opt. 61 (2022) B363.
[60] F. Wang, H. Shiomi, T. Ito, T. Kakue, T. Shimobaba, Opt. Lasers Eng. 160 (2023) 107235.
[61] J.F. Blinn, ACM SIGGRAPH Comput. Graph. 11 (1977) 192–198.
[62] B.T. Phong, Commun. ACM 18 (1975) 311–317.
[63] T. Yamaguchi, G. Okabe, H. Yoshikawa, Opt. Eng. 46 (2007) 125801.