Opto-Electronic Advances

Supplementary information

January 2024, Vol. 7, No. 1

DOI: 10.29026/oea.2024.230108

Dynamic interactive bitwise meta-holography with ultra-high computational and display frame rates

Yuncheng Liu[1†], Ke Xu[1†], Xuhao Fan[1], Xinger Wang[1], Xuan Yu[1], Wei Xiong[1,2*] and Hui Gao[1,2*]

[1]Wuhan National Laboratory for Optoelectronics and School of Optical and Electronic Information, Huazhong University of Science and Technology, Wuhan 430074, China; [2]Optics Valley Laboratory, Wuhan 430074, China.

[†]These authors contributed equally to this work.

*Correspondence: W Xiong, E-mail: weixiong@hust.edu.cn; H Gao, E-mail: gaohui_wnlo@hust.edu.cn

This file includes:

Supplementary information for this paper is available at https://doi.org/10.29026/oea.2024.230108

Section 1: Benchmark measurements of the computational frame rates

Modern software, including the software we developed for this study, incorporates many memory allocations, IO communications, and more, alongside core computations. These programs are often asynchronous, allowing parallel execution of numerous steps, thereby increasing the complexity of the execution process and complicating direct measurements on the actual working software. Therefore, we developed specific benchmark codes for measurement purposes. These codes only incorporate the essential implementations for assessing the metrics of interest. The benchmark codes were compiled under the same platform, using the same compiler and parameters as the working software described in the main text.

The Raspberry Pi micro-controller, compiler, and GUI framework used are as previously described. For the compiler, we used the "-O3" option for maximal execution speed optimization and enabled three additional options: "-fno-exceptions," "-fvisibility-inlines-hidden," and "-fomit-frame-pointer."

To ensure accurate timing from the measurement to the critical step, we determined the necessary preparations, such as relevant storage structures, before starting each measurement. Resource releases were executed post-measurement. Memory for storing the start and end time points of each measurement was pre-allocated to circumvent disturbances during the measurement process due to automatic memory allocation mechanisms. Once all measurements were complete, the metrics were calculated, evaluated, and output.

We constructed the same 144-bit values as outlined in the main text to indicate the state, which were initialized at the outset as a set of random bit values.

For our bitwise operations, we defined the process of measuring a round as follows:

1. All bits are inverted to simulate a complete update each time

2. An array of square squares, each side being 50 pixels, is generated in the center of a 1920×1080 image. This array is compactly arranged as 12×12, and the corresponding squares are filled with white or black, depending on the bit state.

We maintained consistency with the above process for the Fourier transform-based scheme and generally computed the two-dimensional (2D) real-to-complex Fast Fourier transform once in each round of the process. A randomly generated 2D matrix was used as computational input before the measurement. Memory allocation and release were performed before and after the measurement starts to minimize additional overhead caused by system calls. The implementation of the Fast Fourier Transform utilized the basic API provided by the FFTW3 library.

In addition, considering that Fourier transform schemes often require the computation of the entire image, we separately measured for 1920×1080 images as well as for 600×600 images (consistent with the size of the area that would actually be updated by our scheme).

The measurement we used was to count the total time of multiple rounds of computation and subsequently calculate the time of each round. We executed 1000000 cycles for our scheme and 1000000 cycles for the Fourier scheme, and calculated their average execution times, obtaining the following measurements:

Table S1 | Measurement results of our bitwise scheme and traditional Fourier scheme.

	Bitwise	FFT-2D (600×600)	FFT-2D (1920×1080)
Iterations	1000000	1000	1000
Frame generation time (ms)	0.001256	36.631	213.324
Frame rates (kHz)	796.178	0.027	0.005

We should emphasize that we did not utilize any advanced vectorization instructions or special engineering techniques to further accelerate the computation in this particular context. The results presented above represent only a quantitative outcome on our current computing platform. They are used to characterize the metrics to a certain extent and do not represent a generic, absolute metric quantity.

Our scheme is theoretically of lower algorithmic complexity, and the computational efficiency can potentially be further enhanced considering modern processors' efficient support for bitwise operations, as well as the on-demand update of the binary control pattern described in the main text. The Fourier transform scheme we measured did not involve generating the final image used for dynamically refreshing the image on the device. Additionally, in practice, this

type of scheme often requires an iterative process to obtain the phase hologram, which further constrains the computational frame rate.

Section 2: Pattern distortion correction

According to the diffraction process, each pixel on the metasurface device can be considered as a spherical wave source. When we use a flat screen in a finite distance to receive the reconstructed pattern, distortion occurs as shown in Fig. S1(a). The point A of the ideal pattern is extended to point A', with distortion becoming more pronounced the closer it is to the pattern edge, which is similar to pincushion distortion in camera imaging. To mitigate this, we performed an inverse-distortion transformation on the target pattern so that the final distorted pattern would match what it would be without distortion.

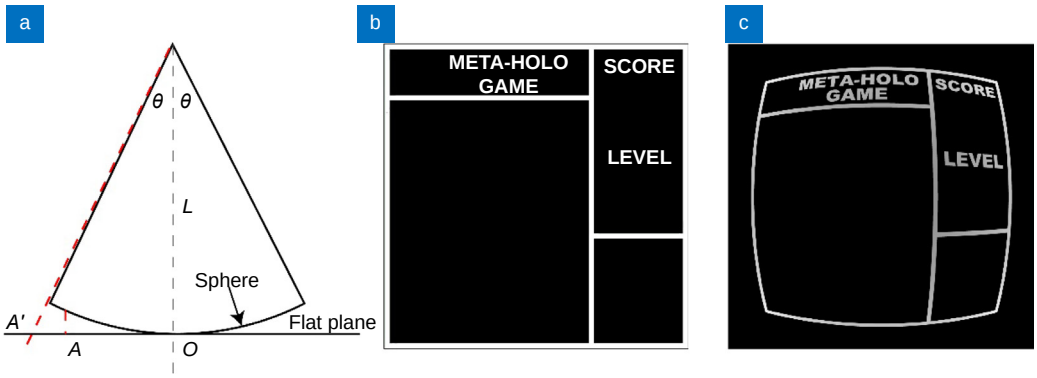

Fig. S1 | Schematic diagram of the distortion correction. (**a**) L is the distance from the point source to the screen, O is the center of the screen, and A and A' are the ideal point position and the distorted point position on the straight screen, respectively. (**b**) Target design pattern, only one pattern was selected for the demonstration. (**c**) The pattern after pre-forming reverse distortion transformation and intensity compensation.

For simplicity, we describe the pre-distortion process in a one-dimensional case. Each pixel on the device can be seen as a grating structure. According to the grating equation:

$$d \times \sin(\theta) = m \times \lambda \,, \tag{1}$$

where d is the grating stripe spacing, θ is the diffraction angle, m is the diffraction level, and λ is the wavelength. On the diffraction plane, $d = 2 \times p$, where p is size of the square pixel. For simplicity, only ± 1 level is interest, so we can have:

$$\sin(\theta) = \frac{\pm\lambda}{2 \times p} \,, \tag{2}$$

As depicted in Fig. S1(a), the sine function relationship does not hold because the point A is stretched to A'. The new relationship between OA' and the diffraction angle can be described using the tangent function.

The diffraction angle is used as an invariant in this process. First, we calculate the angle, then the position of the point before being stretched. From the source to the flat screen, the vertical distance be L (shown in Fig. S1(a)). When distortion occurs, the propagation distance from the point source becomes $\frac{L}{\cos(\theta)}$ times. Therefore, compensating the light intensity of some pixel values according to this relationship is achieved in the image by using different gray values.

For a 2D pattern, the position and intensity grayscale value of each non-distorted pixel is calculated. The new pattern is then used as an input to calculate the hologram. The target display pattern and the pattern with the pre-transformation and intensity compensation are shown in Fig. S1(b) and Fig. S1(c), respectively.

In the experimental section of the main text, we demonstrated our system's ability to operate at different wavelengths. In this study, our metasurface device was primarily designed for 532 nm, which causes some degree of aberration at both 473 nm and 633 nm. We chose patterns where the aberration is more pronounced during the experiments, as shown in Fig. S2. However, this can be solved by pattern distortion correction and intensity compensation for multiple wavelengths during the device design.

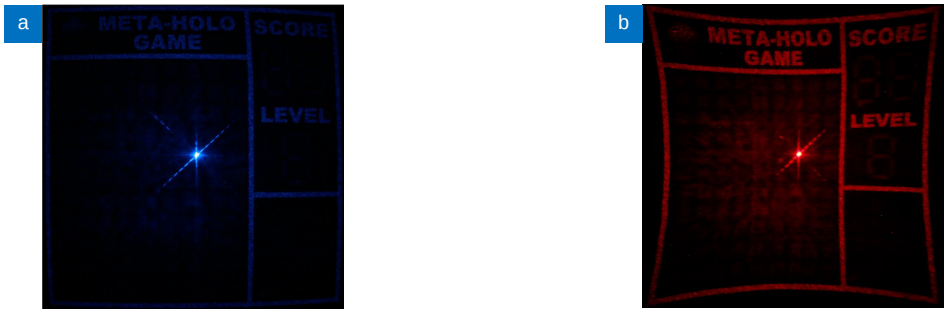

Fig. S2 | The experimental result plots with more obvious pattern distortion at (a) 473 nm and (b) 633 nm, respectively.

Section 3: Discussion on the channel number allocation

In the main manuscript, we allocated 144 channels specifically for the Tetris game. Determining the appropriate number of channels requires consideration of the intended design objectives. Typically, there are two primary factors to account for: the intricacy of the display pattern and the precision required for its trajectory and speed.

For intricate displays, the number of channels might vary significantly, even when showcasing similar content.

The intricacy and artistic style of the display content can be tuned by designing the sub-patterns from each channel and adjusting the number of channels. As illustrated in Fig. S3(a), the classic 7-segment digital tube can depict numbers from 0 to 9. Meanwhile, as demonstrated in Fig. S3(b), by increasing the number of channels to 23, we can showcase numbers from 0 to 9 in another styles. Typically, rendering intricate curve objects necessitates a greater number of channels.

Considering object motion, intricate 2D movements often necessitate additional channels to augment the set of basic sub-patterns. The channel count largely hinges on the desired movement's intricacy, precision of its trajectory, and the accuracy of speed control.

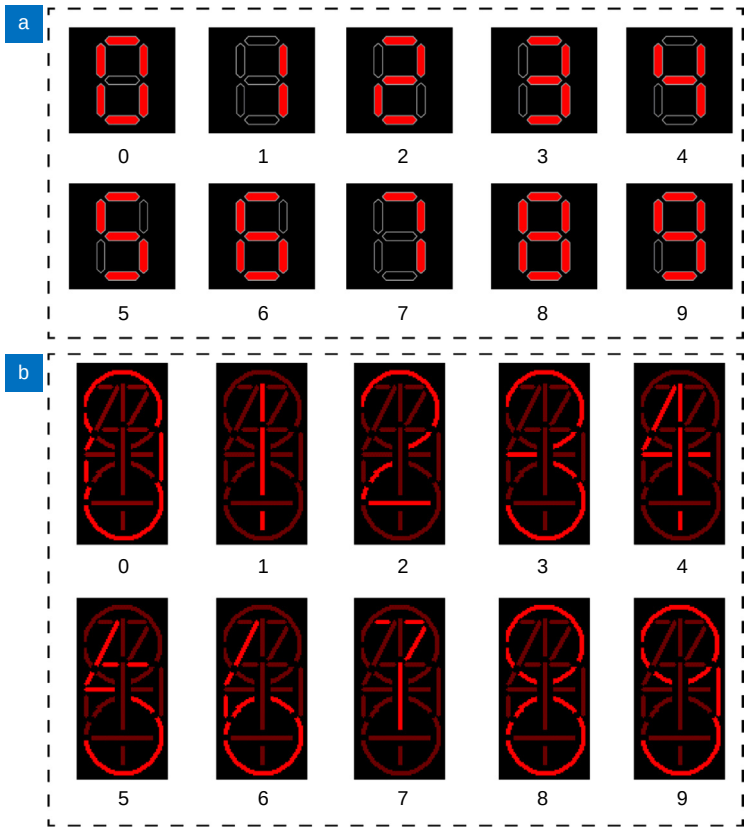

Fig. S3 | Schematic diagram of different digital tube designs to project numbers from 0 to 9. (**a**) A typical 7-segment digital tube. (**b**) 23-segment digital tube.

Using the Tetris game we designed as a foundational example: Four adjacently positioned channels project four immediately adjacent squares, each with a side length of d, as illustrated in Fig. S4(a). The positional accuracy stands at d. Only trajectories characterized by either horizontal or vertical one dimension (1D) movement exhibit smoothness.

Assuming that our system switches 2 frames at a time interval of T. This is influenced by the computational and display frame rates previously mentioned, which, for this discussion, is held constant. Thus, objects' minimum velocity is gauged at d/T.

When the reconstructed pattern overlap by half—while maintaining the total projection area—the number of channels required escalates to nine, as depicted in Fig. S4(b). These channels remain tightly organized adjacent to one another on the device, indicating an expansion in device surface area. Now, the positional accuracy refines to $d/2$, and objects' minimum velocity is adjusted to $d/(2T)$. This configuration avails more freedom in motion within the same projection space, rendering the trajectory more fluid, and it facilitates enhanced precision in speed control.

There exists a direct correlation between the overlap range of reconstructed sub-patterns and both the liberty in motion and precision in object speed. Moreover, when keeping the projection range consistent, the overlap extent correlates positively with the channel count. Yet, if the aim is to maintain the device's surface area while augmenting the channel count, it would be imperative to curtail the number of unit structures within each channel space. This adjustment could potentially compromise reconstructed pattern's display quality.

Consequently, practical scenarios might be more intricate than the example delineated here. Thus, determining the number of channels mandates a meticulous evaluation across multiple facets, inclusive of displayed pattern complexity, motion trajectories, and even the device design and associated manufacturing expenses.

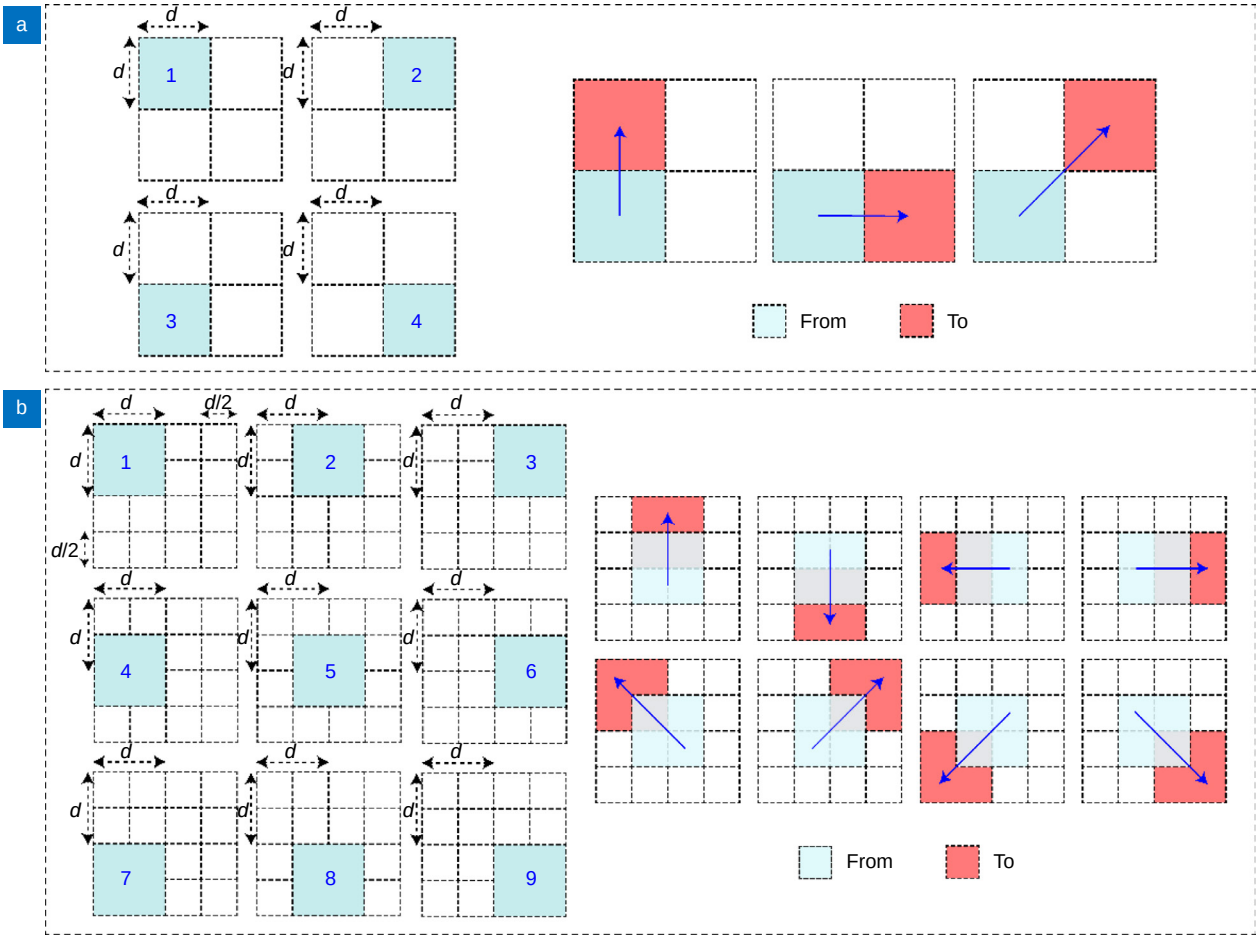

Fig. S4 | Schematic diagrams of different projection modes within the same projection range. All channels are tightly arranged, with only the reconstructed sub-patterns being designed for different situations (**a**) where the reconstructed patterns are tightly arranged and the motion diagram in this situation. (**b**) The reconstructed patterns overlap by half and indicate the direction of motion in this case.

Section 4: Software architecture and development

The control program we developed, which runs on the Raspberry Pi micro-controller mentioned in the main text, is based on the C++ programming language (ISO C++ 20 Standard Version).

We primarily developed the code under Windows 10, utilizing Visual Studio 2022 (Microsoft). The MSVC build tool on Windows served as a toolchain for development and debugging. On the Raspberry Pi platform (64-bit Raspbian OS, Arm Linux), we used the GNU toolchain (GCC 10.1) to build the program. CMake (Kitware, Inc.) was employed as a cross-platform build tool. For GUI development, we used the cross-platform GUI framework Qt (Qt company, Qt 5.15.2 Version).

The program's architecture is depicted in Fig. S5 below. The channel data is designed as a set of bit values, stored compactly in memory, and shared across all components (with data read/write protection in place). The program runs three main parts independently: the game server, the debug console, and the rendering server.

The game server only implements the game logic and encodes the game state data to the channel data, allowing for the implementation of other pixel games, such as Gluttony. The rendering server only generates the compact, underlying binary image data based on the channel data and rendering parameters for loading onto the DMD. This process is actually achieved by writing calculated raw data directly to the HDMI port and doesn't require processing by the graphics rendering module (such as OpenGL). The debug console is primarily used in experiments to adjust parameters such as channel size, rendering parameters, and the channel on/off state. All parameters can be saved to a configuration file and will automatically load at the beginning of the program start. Once debugging is complete, the debug console can be fully closed to further conserve computing resources.

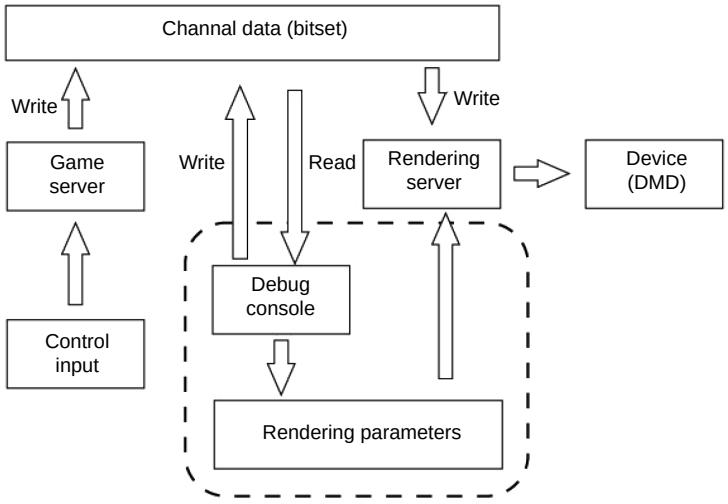

Fig. S5 | Schematic diagram of the architecture of the control software.

Section 5: The optical system for experiments

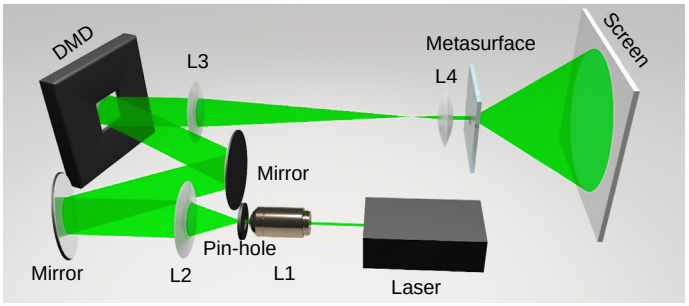

Fig. S6 | Schematic diagram of the optical system. The laser propagates through a spatial pin-hole filter and collimating lens L2 and becomes an expanded laser beam with suitable beam quality. Then, the expanded laser beam is modulated by a DMD at high speed. The coded beam propagates through the 4f system consisting of lens L3 and lens L4, and reconstructed holograms are subsequently acquired behind the sample.

Section 6: Elimination of zero-order diffracted light

In Fig. 5 in the main text, a bright spot is present in the center of the experimental result. The bright spot is unmodulated zero-order diffracted light. Since our study mainly focuses on high-speed dynamic holographic updating strategies, we only employed a basic iterative optimization algorithm to compute the holograms.

Some of the following solutions can be used to eliminate the effect of zero-order light on the display: designing off-axis holograms to keep the zero-order light out of the main viewing area[S1]; setting up optical filters in the optical path to filter out the zero- order light; designing devices using the polarization conversion properties of the anisotropic structure to control the unmodulated light field components finely[S2]; and improving the modulation efficiency of the metasurface through further optimization of the structure parameters[S3]. The core solution of our paper does not conflict with these works, and thus can be combined to achieve better display performance.

Section 7: Description of supplementary movies

File name: Movie S1

Description: The whole process of holographic Tetris game and its channel control patterns that ended in game failure.

File name: Movie S2

Description: The whole process of holographic Tetris game and its channel control patterns that ended in game clearance.

File name: Movie S3

Description: The actual interactive game process, showing the whole process of using the gamepad to play the game.

References

S1. Wen DD, Yue FY, Li GX et al. Helicity multiplexed broadband metasurface holograms. *Nat Commun* **6**, 8241 (2015).
S2. Fu R, Deng LG, Guan ZQ et al. Zero-order-free meta-holograms in a broadband visible range. *Photonics Res* **8**, 723–728 (2020).
S3. He Q, Sun SL, Xiao SY et al. High-efficiency metasurfaces: principles, realizations, and applications. *Adv Opt Mater* **6**, 1800415 (2018).