Real-Time Holographic Solution for True 3D-Display

A. Kaczorowski, S.J. Senanayake, R. Pechhacker, T. Durrant, M. Kaminski and D. F. Milne
VividQ Research and Development Division
Cambridge, UK, CB3 0AX
darran.milne@vivid-q.com

Abstract— Holographic display technology has a been a topic of intense academic research for some time but has only recently seen significant commercial interest. The uptake has been hindered by the complexity of computation and sheer volume of the resulting holographic data meaning it takes up to several seconds minutes to compute even a single frame of holographic video, rendering it largely useless for anything but static displays. These issues have slowed the development of true holographic displays. In response, several easier to achieve, yet incomplete 3D-like technologies have arisen to fill the gap in the market. These alternatives, such as 3D glasses, head-mounted-displays or concert projections are partial solutions to the 3D problem, but are intrinsically limited in the content they can display and the level of realism they can achieve.

Here we present VividQ's Holographic Solutions, a software package containing a set of proprietary state-of-the-art algorithms that compute holograms in milliseconds on standard computing hardware. This allows three-dimensional holographic images to be generated in real-time. Now users can view and interact with moving holograms, have holographic video calls and play fully immersive holographic-mixed reality games. VividQ's Solutions are a vital component for Industry 4.0, enabling IOT with 3D holographic imaging. The software architecture is built around interoperability with leading head-mounted-display and head-up-display manufacturers as well as universal APIs for CAD, 3D gaming engines and Windows based engineering software. In this way, VividQ software will become the new benchmark in 3D enhanced worker/system interaction with unrivalled 3D imaging, representation and interactivity.

Keywords— Digital Holography, GPU, Augmented Reality, Mixed Reality, Optical Systems, Display Technology

I. Introduction

Owing to the recent increased interest in 3D display, multiple technologies have emerged to deliver a convincing 3D experience [1-4]. These largely rely on multi-view or stereoscopic representations designed to "trick" the eye into providing correct depth cues to make the projections appear three-dimensional. However, these depth cues are often limited and in some cases can cause accommodation-vergence conflicts leading to nausea and headaches for the user. Holographic display, on the other hand, aims to precisely recreate the wave-front created from a 3D object or scene, creating a true 3D image of the input scene with all the correct depth cues intact. This makes holographic display an ideal candidate for augmented/mixed reality applications, as it provides 3D virtual objects that appear in focus with their surroundings.

With advances in 3D sensors together with dramatic increases in computational power, Digital Holography (DH) has become a topic of particular interest. In DH, holograms are calculated from point cloud objects, extracted from 3D data sources such as 3D cameras, game engines or 3D design tools, by simulating optical wave propagation [5][6][7]. This simulation can take multiple forms depending on the desired quality of the recovered image and whether the holograms are chosen to be in the far or near field. The resulting hologram may then be loaded onto a suitable digital display device with associated optical set-up for viewing.

A conceptually simple approach for hologram generation is the ray-tracing method [8][9][10], in which the paths from each point on the object to each hologram pixel are computed and aggregated to produce the hologram representation. While the ray-tracing method is physically intuitive, it is highly computationally expensive. To address this issue, many modifications [11-14], and alternatives solutions such as the polygon [14-18] and image-based methods [19][20] have been proposed. In the paper, we describe a real-time holographic display system, that uses a different algorithmic approach based on a Layered Fourier Transform (LFT) scheme [21][22]. We demonstrate how data may be extracted from a 3D data source i.e. the Unity engine, streamed to a holographic generation engine, containing the LFT algorithms. The LFT algorithms are highly parallelized and optimized to run via CUDA kernels on NVidia Graphics Processing Units (GPUs). The resulting holograms are then output to a suitable display device, in this case a Ferroelectic LCoS Spatial Light Modulator (FLCOS SLM).

In the following we describe the various components of the real-time holographic display architecture. In section II, we discuss the streaming of game data from the Unity engine and the standardization of Unity data to a generic 3D format. In Section III, we present the real-time Hologram Generation Engine (HGE) before going into the display setup and driver in Section IV. We discuss results and future work in Section V.

II. DATA CAPURE AND STANARDIZATION

To stream date to the Hologram Generation Engine, we must first extract 3D data from a suitable source. In this case, we choose the Unity engine. Unity is a well-known gaming platform that may be used to create entire 3D scenes using pre-rendered assets.

A. 3D Data Streaming from Unity

Key to the performance of the real-time process is that 3D content rendered within Unity is passed to the holographic generation engine without having to copy memory from the CPU to GPU and back again. The process is summarized in *Fig.1*. Unity uses a concept of *Shaders* to create objects known as Textures that describe virtual scenes. The standard Unity Shaders create Textures as colour maps that specify colours RGB in a 2D grid. While this is suitable for rendering to a standard 2D display, such as a monitor or stereoscopic device, this is insufficient to capture depth information about a scene as required for 3D Holography. Instead, a custom Shader was implemented that renders a colour map with depth (Z) to create a four-channel Colour-Depth-Map (CDM) with channels RGBZ . Each CDM is rendered at the resolution of the Spatial Light Modulator (in this case the rendering is actually performed at half the SLM resolution due the binary nature of the device giving rise to twin images). Unity renders into the CDM texture within an OpenGL context. This allows it to pass the CDM texture object direct to the HGE. Within the HGE, the CUDA-OpenGL-Interop library is utilized to make the data available to the HGE's custom kernel functions, contained in a set of C++ DLLs. This way, Unity is able to render the 3D scene and the information is passed straight to the hologram algorithms without multiple memory copies between the CPU and GPU. In this sense, the OpenGL context acts as a translator between the two steps, allowing us to pass a pointer to the texture directly to the DLLs holding the HGE algorithms. While this implementation is based on OpenGL, one could consider alternative approaches using Direct3D or Vulkan. Direct3D is widely use in the game industry and represents a natural next step in the evolution of the streaming solution as it contains libraries similar to the CUDA-OpenGL-Interop. For Vulkan there is currently no such support, but it is likely that there will be in the near future.

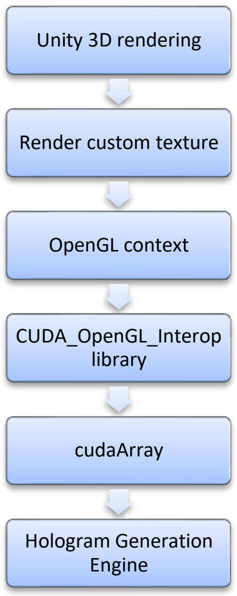

Fig. 1. Unity Streaming Process: Unity renders a 3D scene, the shader creates a custom texture which is passed to an array on the GPU (cudaArray) where the hologram will be calculated.

B. Data Standardization

While Unity is a useful tool for gaming applications and technology demonstrations, for Holography to be available for more general applications, one should define a process to stream and work on data from arbitrary 3D data sources.

Three-dimensional data is present in many forms across multiple software and hardware platforms. To compute holograms, fundamentally we require data in a point-cloud format. A point cloud can be thought of simply as a list of 3D coordinates, specifying the geometry of the object, along with a set of attributes of the cloud e.g. colours (the CDM texture from Unity can be thought of a flattened point cloud with each point of the grid specifying (x,y)-coordinates and the depth channel providing the z). While point clouds are a common and intuitive data type, so far no standard point cloud format has emerged that is compatible with the majority of 3D source data systems. To overcome this issue in holographic applications, and allow arbitrary sources to be streamed to the HGE, we present a new Point Cloud class structure that incorporates the essential features of 3D data required for holographic computation.

C. Point Cloud Class

The point cloud class, **PointCloud**, provides a common framework for data passing through the real-time holographic display system. This allows 3D data to be passed around in memory rather than in file format for fast processing. **PointCloud** is an abstract base class that allows derivative classes to specify specific point cloud representations. In the holographic generation case, we are interested in two particular types of point cloud representation: 3D and 2.5D point clouds. The 3D case refers to a PC that contains complete geometric information of a given object while the 2.5D case occurs when using a PC viewed from a particular perspective. In this case (assuming the object is not transparent), one may neglect points that are occluded.

The base class and inheritance structure of **PointCloud** is designed to be generic and easily extensible so one may define further derivative classes for higher dimensional PCs or PCs with attributes specific to the chosen application or data source. The base class contains generic file reading and writing methods but there is no embedded algorithmic functionality. Instead, all parts of the holographic system architecture may accept an instance of these types and run algorithms using the data contained in them.

With data streamed via the Unity process or through the generic point cloud class we may now compute the holographic representation of the data to be displayed on the SLM for viewing. In the next section we discuss the theory behind the hologram generation process, outline the algorithm in the HGE and describe the expected outputs.

III. REAL-TIME HOLOGRAM GENERATION

A physically intuitive generation method for the calculation of digital holograms is a direct simulation of the physical holographic recording process. In this model, objects are represented by a point cloud where points in the cloud are assumed to emit identical spherical light waves that propagate towards a fixed 2D "holo-plane" offset from the cloud. The resulting interference pattern is calculated on the surface of the holo-plane to yield the digital hologram. While this method is conceptually simple and can produce high quality holograms, it is computationally intensive and time consuming to implement. To reduce the computational load of hologram generation we make use of a layer-based Fourier algorithm. This method partitions the point cloud into parallel, two-dimensional layers by choosing a discretization along one axis of the object. Points that do not intersect one of the discrete layers are simply shifted along the axis of discretization to the closest layer. To construct the hologram a Discrete Fourier Transform (DFT) is applied to each of the layers. The DFT is implemented by the Fast Fourier Transform (FFT) algorithm. To account for the varying depths, a simulated effective lens correction is calculated and applied to each layer. The transformed and depth corrected layers are summed to yield the final hologram. So for a hologram, H, with holo-plane coordinates (α,β), the construction is described by:

$$H(\alpha,\beta)=\sum_i e^{iz_i(\alpha^2+\beta^2)}FT[A_i(x,y)].$$

Where $A_i(x,y)$ is the i^{th} object layer, z_i is the depth parameter for the i^{th} layer. The sum is defined over all the layers in the discretization.

The implementation of the LFT method in the HGE is complicated by two issues. First, three coloured holograms (RGB) must be created to achieve full colour holographic images. This is achieved in this case by including a loop over the colours and essentially running the algorithm three times. The resulting holographic images can then be overlaid in the hologram replay field to give the final full colour holographic image. Note that the three coloured holograms will not yield the same size of image in the replay field due to the different wavelengths, diffracting at different rates on the display. To account for this the input point cloud or CDM for each colour channel must be scaled to ensure the images overlap exactly.

The second issue is that the output display element – in this case a FLCoS SLM – is a binary phase device. Hence, the hologram $H(\alpha,\beta)$, with which in general takes complex values, representing both amplitude and phase, must be quantized to just two phase values i.e. 1-bit per pixel. This causes a severe reduction in quality in the resulting hologram and noise reduction methods must be applied to discern a viewable image. In general, even non-FLCoS devices, such as Nematic SLMs, cannot represent full phase and must quantize to some finite number of phase levels (usually 256-levels i.e. 8-bits per pixel). While an individual binary hologram may give a very poor reconstruction quality, it is possible to use time-averaging to produce high quality images with low noise variance. Such a scheme is implemented in the HGE to account for the limitations of the output display device.

A. *Algorithm Structure*

Given the that we require three-colour holograms composed of multiple object layers and noise reduction must be applied as part of the process, the HGE algorithm proceeds as follows:

1. The CDM Texture is passed from Unity to the C++ DLL that wraps the underlying CUDA kernels.
2. For each colour the CDM is scaled in size to account for the variable rates of diffraction of the three laser fields.
3. FFTs are applied to the CDM data using the cuFFT CUDA Library to give full-phase holograms.
4. The full-phase holograms are quantized to give binary phase holograms.
5. The time-averaging algorithm is applied to eliminate noise in the replay field image.
6. The holograms are stored in memory as a 24-bit Bitmap.

The output of this procedure is a 24-bit Bitmap that can be streamed directly to the FLCoS SLM.

The majority of the algorithmic work is handled by several custom CUDA kernels, which are responsible for handling the CDM object, creating layers, preparing them for FFT and

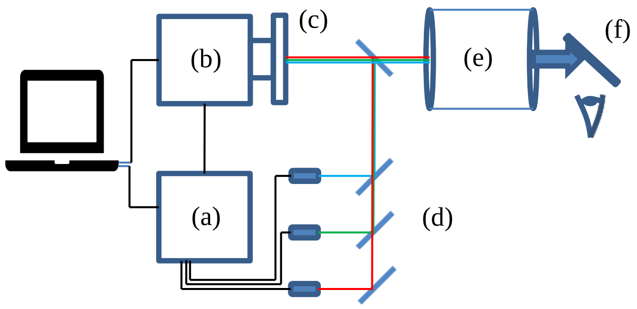

Fig. 2. The holographic display apparatus: The FLCoS SLM is synchronized to three laser diodes (RGB) via a custom micro-controller. The holographic image created by the reflected light, is enlarged by an optical set up and viewed via a beam-splitter eye-piece. (a) Micro-controller, (b) SLM Driver, (c) SLM, (d) Laser-diode array (RGB). (e) Image enlarging optics, (f) Eye-piece

merging to create the holograms. These kernels are called via C++ wrapper functions that expose the functionality without the need to interact directly with the low level CUDA C code.

IV. Device Drivers and Hologram Display

With the holograms computed, all that remains is to display on a suitable output device. Holographic images of this type, cannot be viewed on a typical display such as LED or OLED as these do not allow for phase modulation of light. Instead we use a reflective phase-only Spatial Light Modulator. These devices allow us to modulate the phase of incoming coherent light to generate desired interference patterns in the reflected wave-front to create the holographic image.

The device used here is a ForthDD Ferroelectric Liquid Crystal on Silicon (FLCoS) SLM with 2048 x 1536 pixels of pitch 8.2μm. The device comes equipped with a control unit and drivers to allow developers to alter setting such as refresh rate and colour sequencing. To create the holographic images, RBG lasers are collimated and directed to the SLM surface that is displaying the holograms. The prototype display uses off-the-shelf optical components from ThorLabs and is designed to replicate an augmented reality style display. In this scheme, the holograms are reflected back to a beam-splitter which acts as an eye-piece to achieve the augmented reality effect (Fig. 2).

For this implementation, we create three colour holograms (RG and B) which must be displayed sequentially in a time-multiplexed fashion. To achieve this, a custom Arduino microcontroller was developed that synchronizes the RGB frames with three laser diodes. These frames are shown at high frequency to ensure that the images are time-averaged with respect to the viewer's eye to give a single full-colour image (Fig. 3).

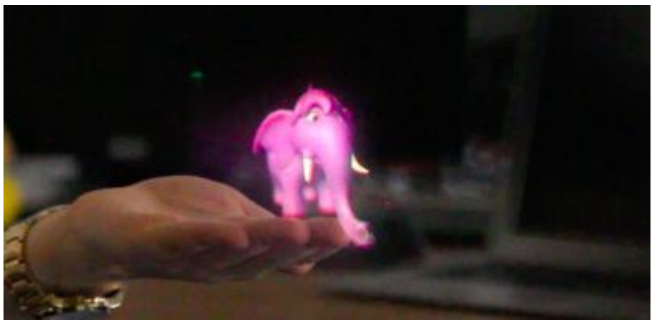

Fig. 3. Augmented Reality Holographic Elephant Image. Photographed directly through the eye-piece with DSLR camera. Image is constructed by overlaying RBG holographic elephants in the replay field to create single full-colour elephant.

V. Results and Discussion

The real-time holographic generation and display process has been tested on a NVidia GTX 1070 – a mid-range gaming GPU. Running a 3D Unity game, with a depth resolution of between 32 and 64 depth layers (number of layers computed depends on the content in the scene and is determined at run-time) the GTX 1070 yields a framerate of 52-55 Hz. This creates a smooth gaming experience assuming a static display as tested here, but a framerate of 90-120 Hz would be required to achieve a seamless mixed reality display system. Increasing the memory available to the GPU would be a first step to allow more holograms to be computed in parallel. Indeed the new generation Volta architecture from NVidia make use of half-float calculations would give a significant speed up to HGE and is projected to allow for >90Hz in the system. Moving to a dedicated ASIC would improve this further by running at lower power in a more compact package, suitable for portable, untethered devices. As 80% of the compute time in the HGE is spent performing FFT, a dedicated embedded solution would provide significant speed up over the current generic approach.

In this optical setup, the image size and quality is constrained by several physical factors. The eye-box and field of view are very small due to the dimensions of the SLM and the optics used to expand the image size. The holographic images also pick up noise due to speckle effects from the laser diodes and there is also some residual noise in the replay field due to the quantization errors created by the binary SLM. These issues can be addressed primarily through higher quality SLMs with smaller pixel pitch and higher resolution. Nematic type, 8-bit SLMs with 4k x2K resolutions and pixel pitch of 3.74 μm are currently available with 8K devices likely to emerge within the near future. The higher resolution and smaller pitch of these devices allow for wider fields of view and finer detail in the holographic images. Additionally, one can consider waveguide solutions combined with eye-tracking for accurate eye-box mapping to ensure the viewer never loses sight of the holographic image. Such schemes are the subject of current research and development.

VI. Conclusion

Here we have presented an end-to-end holographic generation and display system that allows 3D data to be extracted directly from a Unity game, full 3D holograms to be computed and then streamed to an augmented reality holographic display. The hologram generation algorithms achieve a depth resolution between 32 and 64 while maintaining a framerate >50 Hz on a 2k x 1.5k SLM. While the hardware required to view the holographic images is in a nascent state, such an advance in the algorithmic side will enable the development of high-quality, fully interactive holographic display systems that are suitable for mass adoption.

Acknowledgment

The authors would like to thank E. Bundyra, M. Robinson and M. Ippolito for providing funding and business development support in this project. We would also like to thank Prof T. Wilkinson and CAPE at the University of Cambridge for supporting the project in its early stages.

References

[1] J. Park, D. Nam, S. Y. Choi, J. H. Lee, D. S. Park, and C. Y. Kim, "Light field rendering of multi-view contents for high density light field 3D display," in SID International Symposium, pp. 667–670 (2013).

[2] G. Wetzstein, D. Lanman, M. Hirsch, and R. Raskar, "Tensor displays: compressive light field synthesis using multilayer displays with directional backlighting," ACM Trans. Graph. 31, 1–11 (2012).

[3] T. Balogh, P. T. Kovács, and Z. Megyesi, "Holovizio 3D display system," in Proceedings of the First International Conference on Immersive Telecommunications (ICST) (2007).

[4] X. Xia, X. Liu, H. Li, Z. Zheng, H. Wang, Y. Peng, and W. Shen, "A 360-degree floating 3D display based on light field regeneration," Opt. Express 21, 11237–11247 (2013).

[5] R. H.-Y. Chen and T. D. Wilkinson, "Computer generated hologram from point cloud using graphics processor," Appl. Opt. 48(36), 6841 (2009).

[6] S.-C. Kim and E.-S. Kim, "Effective generation of digital holograms of three-dimensional objects using a novel look-up table method," Appl. Opt. 47(19), D55–D62 (2008).

[7] T. Shimobaba, N. Masuda, and T. Ito, "Simple and fast calculation algorithm for computer-generated hologram with wavefront recording plane," Opt. Lett. 34(20), 3133–3135 (2009).

[8] D. Leseberg, "Computer-generated three-dimensional image holograms," Appl. Opt. 31, 223–229 (1992).

[9] K. Matsushima, "Computer-generated holograms for three dimensional surface objects with shade and texture," Appl. Opt. 44, 4607–4614 (2005).

[10] R. Ziegler, S. Croci, and M. Gross, "Lighting and occlusion in a wave-based framework," Comput. Graph. Forum 27, 211–220 (2008).

[11] M. E. Lucente, "Interactive computation of holograms using a look-up table," J. Electron. Imaging 2, 28–34 (1993).

[12] Y. Pan, X. Xu, S. Solanki, X. Liang, R. B. Tanjung, C. Tan, and T. C. Chong, "Fast CGH computation using S-LUT on GPU," Opt. Express 17, 18543–18555 (2009).

[13] P. Tsang, W.-K. Cheung, T.-C. Poon, and C. Zhou, "Holographic video at 40 frames per second for 4-million object points," Opt. Express 19, 15205–15211 (2011).

[14] J. Weng, T. Shimobaba, N. Okada, H. Nakayama, M. Oikawa, N. Masuda, and T. Ito, "Generation of real-time large computer generated hologram using wavefront recording method," Opt. Express 20, 4018–4023 (2012).

[15] K. Matsushima, "Wave-field rendering in computational holography: the polygon-based method for full-parallax high-definition CGHs," in IEEE/ACIS 9th International Conference on Computer and Information Science (ICIS) (2010).

[16] D. Im, E. Moon, Y. Park, D. Lee, J. Hahn, and H. Kim, "Phase regularized polygon computer-generated holograms," Opt. Lett. 39, 3642–3645 (2014).

[17] K. Matsushima and A. Kondoh, "A wave-optical algorithm for hidden surface removal in digitally synthetic full parallax holograms for threedimensional objects," Proc. SPIE 5290, 90–97 (2004).

[18] H. Nishi, K. Matsushima, and S. Nakahara, "Advanced rendering techniques for producing specular smooth surfaces in polygon-based high-definition computer holography," Proc. SPIE 8281, 828110 (2012).

[19] H. Shum and S. B. Kang, "Review of image-based rendering techniques," Proc. SPIE 4067, 2–13 (2000).

[20] F. Remondino and S. El-Hakim, "Image-based 3D modelling: a review," Photog. Rec. 21, 269–291 (2006).

[21] J.-S. Chen, D. Chu, and Q. Smithwick, "Rapid hologram generation utilizing layer-based approach and graphic rendering for realistic three-dimensional image reconstruction by angular tiling," J. Electron. Imaging 23, 023016 (2014).

[22] J. S. Chen and D. P. Chu, "Improved layer-based method for rapid hologram generation and real-time interactive holographic display applications," Opt. Express 23, 18143–18155 (2015).