RAY TRACING FOR HOLOVIZIO LIGHT FIELD DISPLAYS

Oleksii Doronin, Attila Barsi

Holografika
Budapest, Hungary
{o.doronin, a.barsi}@holografika.com

Peter A. Kara, Maria G. Martini

WMN Research Group
Kingston University, London, UK
{p.kara, m.martini}@kingston.ac.uk

ABSTRACT

Ray tracing is a well-known method of virtual 3D scene visualization. Nowadays it is mostly used for the rendering of photo-realistic static pictures for conventional 2D displays. In this paper, we describe the implementation of a ray tracing algorithm for a specific type of 3D rendering system, namely for the HoloVizio horizontally parallax light field display. We discuss the difficulties of the implementation of ray tracing for such systems, together with their solutions, as well as the benefits that could not apply to conventional 2D rendering systems.

Index Terms— ray tracing, light field, light field display.

1. INTRODUCTION

Ray tracing can provide visual realism by simulating the natural behavior of light rays as they reflect from the objects in a virtual scene. The basic ray tracing technique was introduced in 1979 by Whitted [1]. This technique was immediately recognized in the field of computer graphics as one that can provide the rendering of quite complex visual effects, such as reflections and refractions.

However, at the time of its introduction, this approach required too much computational power, thus making ray tracing an ineligible method for real-time rendering. In spite of being an effective method for rendering photo-realistic images, ray tracing has been considered as a so-called offline-only solution for decades. Over the years, it has been constantly improved in terms of the quality of the resulting image, but the offline-only attitude towards ray tracing is slowly being changed as well, due to the huge variety of both algorithmic and hardware improvements that are continuously being introduced.

In the conventional 2D framework, ray tracing refers to the generation of rays coming from the camera through each pixel of the screen (*primary rays*), and checking where these rays intersect the scene geometry. If the surface of the geometry has reflective or refractive properties, additional rays (*secondary* rays) are generated, originating from the hitting positions of the primary rays towards the corresponding directions. If any of the aforementioned rays intersect the scene geometry at any point, additional *shadow rays* are generated from the hit point towards each light source, in order to check whether the surface of the hit point is occluded or not.

Beside ray tracing, many alternative solutions for simulating the light behavior (e.g., global illumination, ambient occlusion, bidirectional path tracing, etc.) have been introduced as well. They do contribute to the aimed visual realism, but the scope of this paper is solely set on ray tracing.

In this paper, we describe the application of the ray tracing algorithm for the a real-life industrial light field visualization system, namely the HoloVizio system. The paper details the way of generating the primary rays for such system. We also prove that the application of the ray tracing approach is beneficial for the HoloVizio light field system (compared to the rasterization methods), as it can handle the non-linear distortion of the geometric primitives in the output images.

The remainder of the paper is structured as follows. Section 2 discusses the related work in this research area. An overview of the light field system used in our research is given in Section 3, with the details of implementation described in Section 4. The results of the testing of our system are provided in Section 5. The paper is concluded in Section 6.

2. RELATED WORK

The work of Wald *et al.* [2] can be considered as a good survey of different methods to improve the basic algorithm. Keller *et al.* [3] summarizes recent advancements in the field of ray tracing in form of an academic course, discussing several low- and high-level frameworks (Embree, OSPRay, FireRender, FireRays, NVIDIA OptiX).

In order to increase the running speed of ray tracing, several *acceleration structures* have been introduced. The bounding volume hierarchy (BVH) [4, 5, 6, 7, 8] is perhaps the most commonly used one. However, other options (e.g., kd-tree [9], bounding interval hierarchy [10], uniform grid [11]) are also possible. Other directions of work on ray tracing include different methods for the improvement of the quality of the resulting image (e.g., multi-sampling and anti-aliasing [12], defocus effect and motion blur [13, 14]), which can necessitate changes in the algorithm itself.

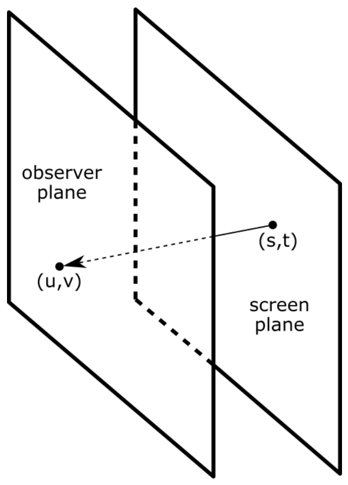


Fig. 1: Two-plane parameterization of light field.

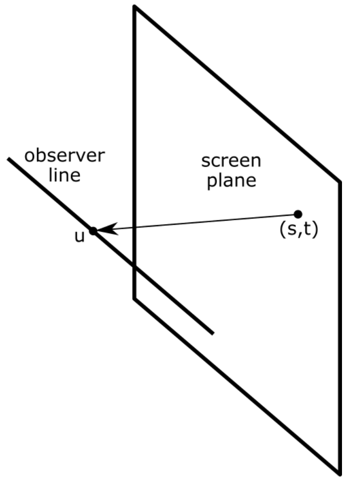


Fig. 2: Plane-line parameterization of light field.

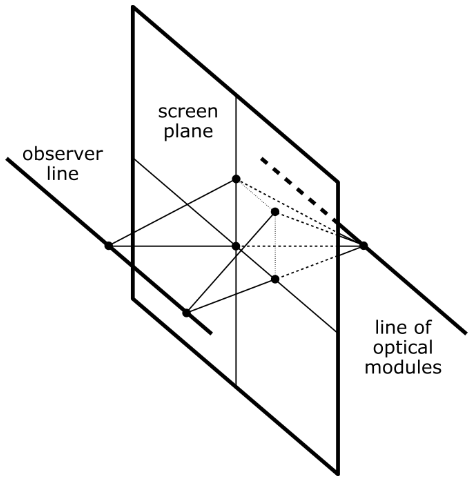


Fig. 3: Plane-line parameterization in HoloVizio.

The fundamental application of ray tracing to 3D visualization has already been described by several researchers. A method for fast generation of pictures from different viewer positions was introduced by Levoy and Hanrahan [15]. The application of this approach to real-life lumigraph image-based rendering systems is introduced by Isaksen *et al.* [16]. Research in the field of virtual reality is introduced by Wald *et al.* [17], and for augmented reality by Scheer *et al.* [18].

According to the knowledge of the authors, more recent advancements in the field of ray tracing mostly focus on the improvement of well-known algorithms (or certain parts these algorithms), or optimizing them for specific platforms, rather than inventing new methods. For example, Catalano *et al.* [19] introduced a framework for the global illumination algorithm that uses point clouds and optimizes the work of shaders. Benthin *et al.* [20] presented a method for constructing a two-level BVH, which can be considered as a good trade-off between the overall BVH quality and its construction speed. Binder and Keller [21] developed a method to ray trace bicubic Bezier patches together with conventional primitives. Wald *et al.* [22] introduced a specific acceleration structure (the P-k-d tree) that is optimal for ray tracing a large collection of spherical primitives (e.g., particles, molecules, celestial bodies). Benthin *et al.* [23] presented a method for the adaptive tesselation of primitives. However, we could not find any sources in the scientific literature that would describe the application of the ray tracing algorithm to a real-life light field display. Therefore, we introduce an implementation of ray tracing algorithm for the HoloVizio [24, 25] light field display system.

In computer graphics, there is a common assumption that the characteristics of light do not change within the same ray of light. Under this assumption, it is suitable to use the *two-plane parameterization* [26] of light field (see Figure 1). It defines every possible ray of light in the light field by determining the coordinates of intersections of the ray with two parallel planes. This parameterization represents a 4D space: the first two dimensions are the plane coordinates of the first plane, and last two are the plane coordinates of the second plane.

3. OVERVIEW OF THE HOLOVIZIO SYSTEM

The two-plane 4D parameterization (see Figure 1) is probably the most frequently used for parameterizing the full-parallax light field. One plane in this parameterization can be considered as the *screen plane* (where the light sources can be placed), and the another as the *observer plane* (where the viewers can perceive light field).

For the horizontally-parallax light field systems (like HoloVizio), Balázs *et al.* [24] suggested to use the 3D plane-line parameterization (see Figure 2). In this case, the observer plane can be substituted with the *observer line*.

The HoloVizio system inherently uses the described plane-

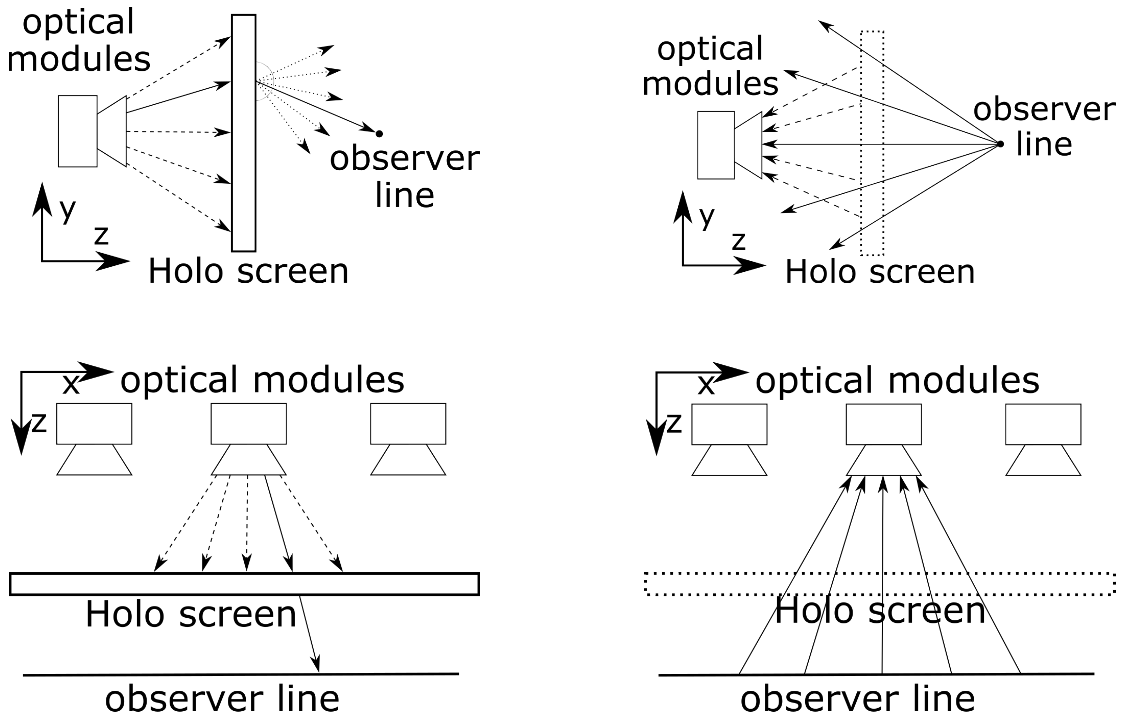


Fig. 4: Rays of light emitted by the optical modules (left), and the primary rays in ray tracing (right).

line parameterization. In its most simple form, this system consists of a semi-transparent screen with special optical properties, and a row of optical modules placed behind it (see Figure 2). When an individual ray of light from an optical module passes through the screen, it gets diffused in all directions, but with a different amount of scattering. As vertical scattering is very strong, we assume that the rays are scattered uniformly in all vertical directions. Among vertically scattered rays, we consider only those ones that will eventually hit the observer line (see Figures 3 and 4). The horizontal scattering is quite weak, and we can neglect it. Therefore, for the purpose of parameterizing the initial ray tracing setup, we assume that each ray of light that originates from the optical module, is then refracted at the screen plane in order to hit the observer line while keeping its initial horizontal direction.

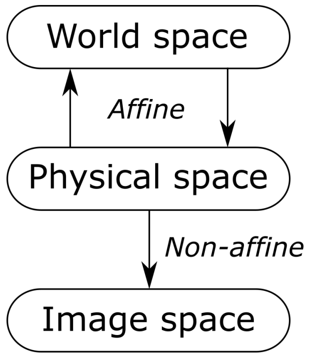


Fig. 5: Relationship between the spaces in HoloVizio.

The input information for each optical module in the HoloVizio system is internally represented as a 2D texture. At each pixel, this texture contains the color of the appropriate ray of light coming from the optical module (see Figure 4). For convenience, we differentiate between the three different spaces: the *world space* (in which the coordinates of the virtual scene are stored), the *physical space* (where the transformations of rays are computed), and the *image space* (coordinates of a texture for each individual optical module). The affine transformation from the world space to the physical space can be defined as the transformation between two matching rectangular parallelepipeds, the coordinates of which are expressed in the corresponding spaces. The mapping from the physical space to the image space is calculated taking into consideration the position and direction of the optical module and the screen curvature, together with its diffusion through the screen surface. Therefore, it is neither affine nor homogeneous. Figure 5 shows the relationship between the aforementioned spaces.

4. IMPLEMENTATION

According to the knowledge of the authors, the research presented in this paper is the first successful attempt to implement ray tracing on an actual light field display. In this section, we introduce our implementation of the algorithm, and in the next section, we detail the results obtained for different 3D models and scenes.

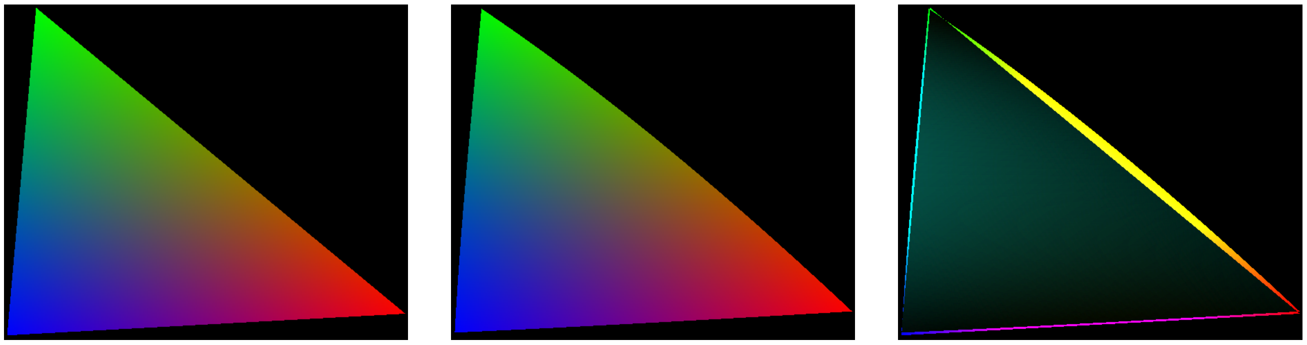

(a) Rasterization (linear interpolation) (b) Ray tracing (c) Difference (400% amplification)

Fig. 6: Difference between a conventional rasterization algorithm and ray tracing for the optical module image.

4.1. Generation of Primary Rays

The primary rays for ray tracing in the HoloVizio system are generated in a way that they should be inverse to the rays that hit the observer line from the optical module (see Figure 4 and Algorithm 1). Therefore, all primary rays start at a certain position on the observer line, and go through the corresponding point on the screen plane. The starting positions and directions of the primary rays are computed per pixel of the optical image. These computations are done in the physical space. Generally speaking, no pair of the primary rays should share the same starting position or direction – in contrast to the conventional 2D ray tracing. It is also important to state that the starting positions and directions of the primary rays do not change in time while the HoloVizio system operates; the transformation from the physical space to each particular image space is constant in time. To keep all necessary information about the primary rays during the operation of the system, we use two 2D textures for each optical module: one for the starting positions (*origins texture*), and another for the directions (*directions texture*).

Algorithm 1 Generation of primary rays.

```
pixel ← textural $(x, y)$ coordinates of current pixel;
optiRay ← emitRayFromOpticalModule(pixel);
screenRay ← diffuseRayThroughScreen(optiRay);
primRay.origin ← hitObserverLine(screenRay);
primRay.dir ← − screenRay.dir;
```

Algorithm 1 calculates the origins and directions of the primary ray in the physical space for a given (x, y) position in the image space. This implicitly defines the mapping from the physical space to the image space. In this mapping, one point of the image space corresponds to a line (aligned with a primary ray) in the physical space.

The affine transformation from the world space to the physical space can be defined much easier. In the world space, let us encapsulate the volume that we would like to display by a rectangular parallelepiped B_W. In the physical space, let us encapsulate the volume that we can actually display, by a rectangular parallelepiped B_P. The affine transformation, that corresponds to the linear bijective mapping from B_W to B_P, is the transformation from the world space to the physical space that we are looking for.

4.2. Implemented Features

For testing purposes, we limit ourselves to static scenes only. We choose to use the BVH [7] with axis-aligned bounding boxes (AABB) as the acceleration structure. Our version of BVH is constructed using the top-bottom SAH partitioning [8] with 32 bins. This process is implemented as a single-thread C++ code. Each BVH node is represented by a 32-byte structure. After construction, the array of BVH nodes is uploaded into the GPU memory using appropriate OpenGL buffer.

The rendering phase of our algorithm is implemented as two OpenGL compute shaders: the *traversal shader* and the *drawing shader*. The traversal shader takes two textures (ray origins and ray positions) and two OpenGL buffers (array of BVH nodes and array of geometric primitives) as the input. As the output, it fills the *intersection buffer* (OpenGL buffer) with all necessary information about the intersections of all primary and secondary rays with the scene geometry, and the *index texture* that contains the indices of the intersection buffer. The drawing shader takes the intersection buffer, index texture and the scene geometry as the input, together with all necessary textures for material rendering in OpenGL bindless texture format. After the application of lighting and shading, it renders the final texture that is further used by the HoloVizio optical module. We use a slightly improved version of the stack-less approach of Hapala *et al.* [5] as the traversal algorithm.

5. RESULTS

5.1. Correction of the Output Image

As it was explained in Section 3, the mapping from the physical space to the image space is not homogeneous and not invertible. Therefore, when some geometric primitive (e.g., triangle) is rendered, it is not correct to apply the linear interpolation to get the correct positions of the points inside this primitive. This implies that the application of conventional rasterizer-type algorithms (e.g., OpenGL vertex/fragment shader pair) is not always suitable for the HoloVizio light field display. An acceptable solution for rasterized rendering is the tesselation of the geometry in order to keep the interpolation error small.

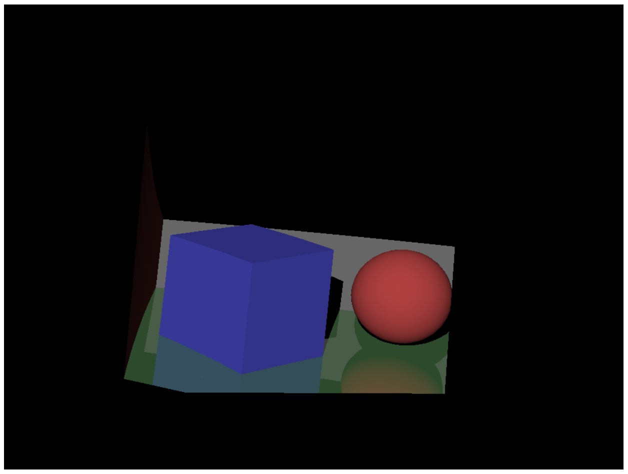
In contrast, the primary rays in the introduced ray tracing algorithm are generated in a way to simulate the actual behavior of light coming from each optical module. This means that the application of ray tracing algorithm will result in getting the correct position information for all visible points of the virtual scene.

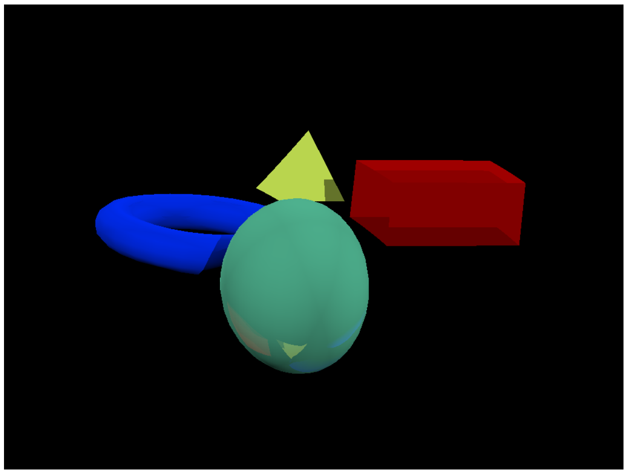
Figure 6 shows an optical module texture with a rendered primitive of a large size on it. We choose to render a triangle with world coordinates of its vertices $(1, 0, 0)$, $(0, 1, 0)$, $(0, 0, 1)$. The RGB values of this image are equal to the appropriate world coordinates of the triangle (R for x, G for y, B for z). The left part of the figure (a) is rendered with the help of conventional rasterizer with linear interpolation between vertices. The middle part (b) is rendered with the introduced ray tracing algorithm. The right part (c) is the difference image between the left and central, obtained with the help of Compressonator tools [27], with difference being magnified by 400%.

One may argue that in most scenes the size of the geometrical primitives is small, and thus, the visual difference between the optical module images obtained with conventional rasterizer and ray tracing can hardly be perceived. This argument is true in case there is no post-processing step, which would actually require the correct position information. In different scenarios (i.e., if the screen-space ambient occlusion algorithm [28] is applied), the quality of the resulting image could be severely degraded. At the end of the day, it is perceived quality that determines the success and the actual performance of the solution at the consumer's side, thus such visual degradation should be avoided.

5.2. Performance

We measured the performance of the ray tracing algorithm on a simulation of the HoloVizio environment, running on a machine with Intel i7-5820K 3.30GHz CPU and GeForce GTX 960 video card. The simulated system included only one optical module with the image resolution of 1024×768 pixels. We tested three scenes (see Figure 7): the Cornell box with reflecting floor (1110 triangles, no textures), a set of geomet-


(a) Cornell


(b) Shapes

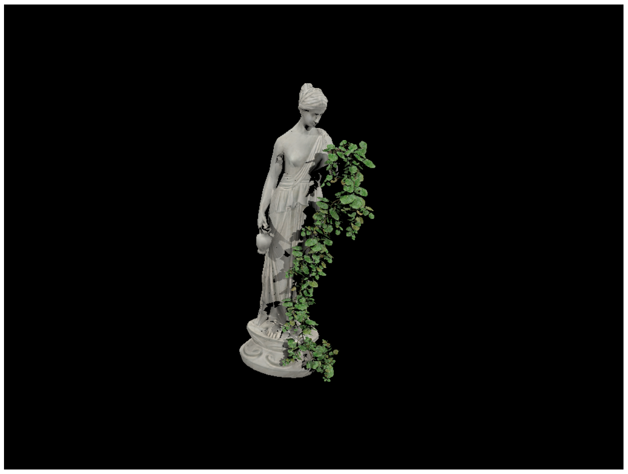

(c) Statue

Fig. 7: Pictures for one optical module.

rical shapes with two refracting objects (1556 triangles, no textures), and a model of a statue (5011 triangles, 3 textures). We set the maximum depth of ray tracing as 5, and used 4 directional light sources with different directions. For each scene, we rendered 100 frames and measured the average time to render a single frame (see Table 1).

	ms/fr	fps
Cornell	146.05	6.85
Shapes	205.43	4.87
Statue	1084.64	0.922

Table 1: Measured average performance in milliseconds per frame (first column) and frames per second (second column).

We acknowledge that the achieved performance results are insufficient for modern-day real-time computer graphics. However, we proved that it is possible to adjust the ray tracing related algorithms for real light field displays, and there can be a lot of possibilities to make further research on performance improvement. For example, one can consider the implementation of packetized traversal [7]. Although this technique is capable to produce a significant speedup for the conventional ray tracing, there are at least two problems for optimizing it for the HoloVizio system. First, the primary rays are significantly less aligned with each other on the HoloVizio system (compared to a conventional system), which will make it harder to define an optimal bounding volume (frustum) for them. Second, the conventional definition of a frustum as a four-sided pyramid may be not an optimal choice for the bunch of primary rays on the HoloVizio system. Therefore, the in-depth performance analysis with real solutions on performance improvement stays outside the scope of this paper.

Figure 8 contains the footage of the statue rendered and displayed on a HoloVizio 640RC light field system. By the time of this paper, the mentioned HoloVizio system is not available in the market, and the closest available analogue is the HoloVizio 722RC system [29].

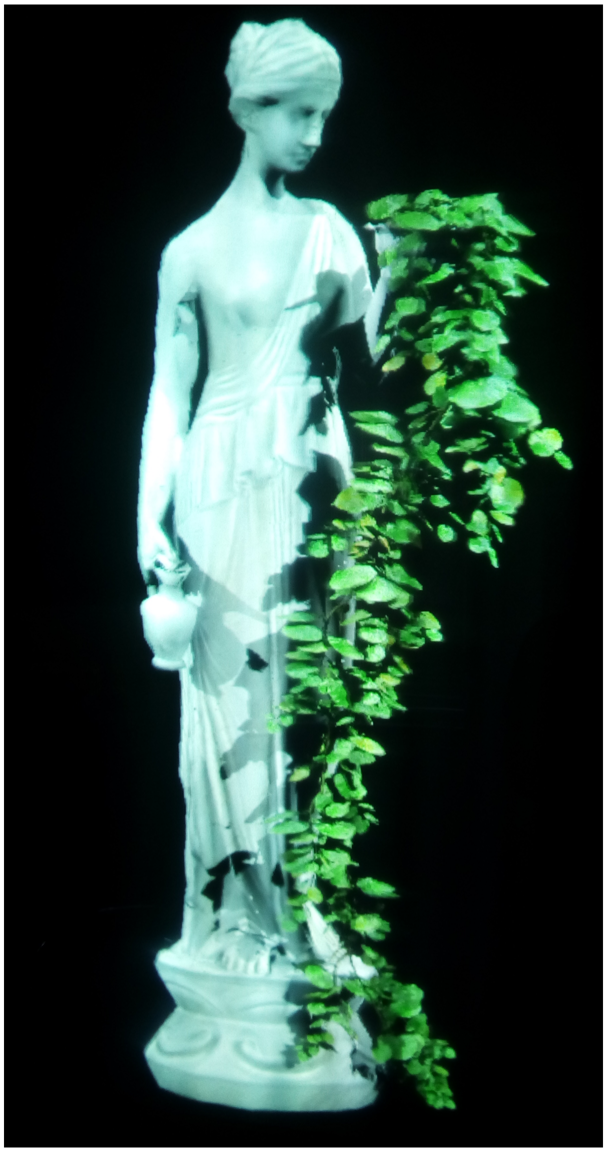


Fig. 8: Photo of the HoloVizio 640RC system displaying a ray-traced scene.

6. CONCLUSIONS

In this paper, we discussed the possibility of the implementation of ray tracing on the HoloVizio light field display system, with its benefits and pitfalls. The research shows that the conventional ray tracing algorithm can be implemented on this system, with the only difference in the generation of the primary rays that is specific for this type of displays. The application of ray tracing is beneficial for the HoloVizio system, compared to the conventional rasterization algorithms, in terms of more precise construction of the resulting image for the optical modules. We also conclude that a visualization algorithm based on ray tracing is a better choice for such system, if additional post-processing steps are required.

7. ACKNOWLEDGEMENTS

The research in this paper was done as a part of and was funded from the European Unions Horizon 2020 research and innovation program under the Marie Sklodowska-Curie grant agreement No 676401, ETN-FPI, and No 643072, Network QoE-Net.

8. REFERENCES

[1] Turner Whitted. An Improved Illumination Model for Shaded Display. volume 23, pages 343–349, New York, NY, USA, June 1980. ACM.

[2] Ingo Wald, William R. Mark, Johannes Günther, Solomon Boulos, Thiago Ize, Warren Hunt, Steven G. Parker, and Peter Shirley. State of the Art in Ray Tracing Animated Scenes. In *Computer Graphics Forum*, volume 28, pages 1691–1722. Wiley Online Library, 2009.

[3] Alexander Keller, Ingo Wald, Takahiro Harada, Dmitry Kozlov, Ralf Karrenberg, Luke Peterson, and Tobias Hector. The Quest for the Ray Tracing API. In *ACM SIGGRAPH 2016 Courses*, SIGGRAPH '16, pages 25:1–25:1, New York, NY, USA, 2016. ACM.

[4] Yan Gu, Yong He, Kayvon Fatahalian, and Guy Blelloch. Efficient BVH Construction via Approximate Agglomerative Clustering. In *Proceedings of the 5th High-Performance Graphics Conference*, HPG '13, pages 81–88, New York, NY, USA, 2013. ACM.

[5] Michal Hapala, Tomáš Davidovič, Ingo Wald, Vlastimil Havran, and Philipp Slusallek. Efficient Stack-less BVH Traversal for Ray Tracing. In *Proceedings of the 27th Spring Conference on Computer Graphics*, SCCG '11, pages 7–12, New York, NY, USA, 2013. ACM.

[6] Daniel Kopta, Thiago Ize, Josef Spjut, Erik Brunvand, Al Davis, and Andrew Kensler. Fast, Effective BVH Updates for Animated Scenes. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, I3D '12, pages 197–204, New York, NY, USA, 2012. ACM.

[7] Ingo Wald, Solomon Boulos, and Peter Shirley. Ray Tracing Deformable Scenes using Dynamic Bounding Volume Hierarchies. *ACM Trans. Graph.*, 26(1), January 2007.

[8] Ingo Wald. On fast Construction of SAH-based Bounding Volume Hierarchies. In *Interactive Ray Tracing, 2007. RT'07. IEEE Symposium on*, pages 33–40. IEEE, 2007.

[9] Ingo Wald and Vlastimil Havran. On building fast kd-Trees for Ray Tracing, and on doing that in O(N log N). In *Interactive Ray Tracing 2006, IEEE Symposium on*, pages 61–69. IEEE, 2006.

[10] Carsten Wächter and Alexander Keller. Instant Ray Tracing: The Bounding Interval Hierarchy. In *Proceedings of the 17th Eurographics Conference on Rendering Techniques*, EGSR '06, pages 139–149, Aire-la-Ville, Switzerland, Switzerland, 2006. Eurographics Association.

[11] John Amanatides and Andrew Woo. A Fast Voxel Traversal Algorithm for Ray Tracing. In *Eurographics*, volume 87, pages 3–10, 1987.

[12] Maxim Shevtsov, Mikhail Letavin, and Alexey Rukhlinskiy. Low Cost Adaptive Anti-Aliasing for Real-Time Ray-Tracing. In *Proceedings of GraphiCon2010*, pages 45–48. St.Petersburg State University, 2010.

[13] Solomon Boulos, Dave Edwards, J. Dylan Lacewell, Joe Kniss, Jan Kautz, Peter Shirley, and Ingo Wald. Interactive Distribution Ray Tracing. *SCI Institute, University of Utah, Technical Report*, 2006.

[14] Qiming Hou, Hao Qin, Wenyao Li, Baining Guo, and Kun Zhou. Micropolygon Ray Tracing with Defocus and Motion Blur. *ACM Trans. Graph.*, 29(4):64:1–64:10, July 2010.

[15] Marc Levoy and Pat Hanrahan. Light Field Rendering. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '96, pages 31–42, New York, NY, USA, 1996. ACM.

[16] Aaron Isaksen, Leonard McMillan, and Steven J. Gortler. Dynamically Reparameterized Light Fields. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '00, pages 297–306, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[17] Ingo Wald, Andreas Dietrich, Carsten Benthin, Alexander Efremov, Tim Dahmen, Johannes Gunther, Vlastimil Havran, Hans-Peter Seidel, and Philipp Slusallek. Applying Ray Tracing for Virtual Reality and Industrial Design. In *Interactive Ray Tracing 2006, IEEE Symposium on*, pages 177–185. IEEE, 2006.

[18] Fabian Scheer, Oliver Abert, and Stefan Müller. Towards Using Realistic Ray Tracing in Augmented Reality Applications with Natural Lighting. In *GI Workshop ARVR*, volume 7, 2007.

[19] Enzo Catalano, Rajko Yasui-Schöffel, Ken Dahm, Nikolaus Binder, and Alex Keller. GI Next: Global Illumination for Production Rendering on GPUs. In *ACM SIGGRAPH 2016 Talks*, SIGGRAPH '16, pages 69:1–69:2, New York, NY, USA, 2016. ACM.

[20] Carsten Benthin, Sven Woop, Ingo Wald, and Attila T. Áfra. Improved Two-level BVHs Using Partial Rebraiding. In *Proceedings of High Performance Graphics*, HPG '17, pages 7:1–7:8, New York, NY, USA, 2017. ACM.

[21] Nikolaus Binder and Alexander Keller. Stackless Ray Tracing of Patches from Feature-adaptive Subdivision on GPUs. In *ACM SIGGRAPH 2015 Talks*, SIGGRAPH '15, pages 22:1–22:1, New York, NY, USA, 2015. ACM.

[22] Ingo Wald, Aaron Knoll, Gregory P. Johnson, Will Usher, Valerio Pascucci, and Michael E. Papka. CPU Ray Tracing Large Particle Data with Balanced P-k-d Trees. In *IEEE ScientificVisualization Conference 2015 (SciVis)*, pages 57–64. IEEE, 2015.

[23] Carsten Benthin, Sven Woop, Matthias Nießner, Kai Selgrad, and Ingo Wald. Efficient Ray Tracing of Subdivision Surfaces Using Tessellation Caching. In *Proceedings of the 7th Conference on High-Performance Graphics*, HPG '15, pages 5–12, New York, NY, USA, 2015. ACM.

[24] Ákos Balázs, Attila Barsi, Péter T. Kovács, and Tibor Balogh. Towards mixed reality applications on light-field displays. In *3DTV-Conference: The True Vision-Capture, Transmission and Display of 3D Video (3DTV-CON), 2014*, pages 1–4. IEEE, 2014.

[25] Tibor Balogh, Péter T. Kovács, and Zoltán Megyesi. HoloVizio 3D Display System. In *Proceedings of the First International Conference on Immersive Telecommunications*, ImmersCom '07, pages 19:1–19:5, ICST, Brussels, Belgium, Belgium, 2007. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[26] Xianfeng Gu, Steven J. Gortler, and Michael F. Cohen. Polyhedral Geometry and the Two-Plane Parameterization. In *Proceedings of the Eurographics Workshop on Rendering Techniques '97*, pages 1–12. Springer-Verlag, London, UK, UK, 1997.

[27] Compressonator.
http://gpuopen.com/gaming-product/compressonator/
(retrieved July 2017).

[28] Oleksii Doronin, Peter A. Kara, Attila Barsi, and Maria G. Martini. Screen-Space Ambient Occlusion for Light Field Displays. In *25th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG2017)*, 2017.

[29] Holovizio 722RC light field display.
http://www.holografika.com/Products/
HoloVizio-722RC.html (retrieved July 2017).