GPU-Accelerated Point-Based Holograms

Sascha Fricke[†] **Reinhard Caspary**[⋆] **Susana Castillo**[†] **Marcus Magnor**[†]

[†] *Institut für Computergraphik, TU Braunschweig, Mühlenpfordtstr. 23, 38106 Braunschweig, Germany*
[⋆] *Cluster of Excellence PhoenixD, Leibniz Universität Hannover, Welfengarten 1A, 30167 Hannover, Germany*

[†] {*fricke, castillo, magnor*}*@cg.cs.tu-bs.de* [⋆]*reinhard.caspary@phoenixd.uni-hannover.de*

Abstract: We present a novel fast method to compute point-based Computer Generated Holograms with occlusion on the GPU, using modern hardware capabilities. Our method offers a promising new route towards real-time calculation of high-resolution holograms.

1. Problem statement

Point-Based (PB) holography algorithms offer a flexible way of generating holograms from a point cloud with arbitrarily and sparsely distributed points in three dimensions. However, they can suffer from poor algorithmic scaling, i.e. $O(n \times m)$ with n object points and m hologram points, and while acceleration via thread scheduling on Graphics Processing Unit (GPU)'s is well suited to solve this limitation, doing it efficiently is a non-trivial problem [1]. In this work, we propose a new method vastly simplifying the combination of both approaches, enabling fast occlusion processing, and thus being able to generate PB holograms in interactive time.

In PB methods, an Object Point (OP) emits a spherical wave that intersects with the hologram plane, where the resulting wavefront is recorded. Superimposing these wavefronts from multiple points creates an interference pattern. One of the first algorithms to propose using the GPU for Computer Generated Holography (CGH), Petz and Magnor [2], used the hardware fixed-function pipeline in conjunction with hardware texturing to accelerate the interference computations. This method was restricted by the lack of floating point precision and programmable shaders provided by the hardware of the time. Later, Chen and Wilkinson [1] introduced the use of Compute Unified Device Architecture (CUDA) to accelerate this process further. The calculation time of PB methods scales with the propagation distance, since the radius of the point spread function rises with the propagation distance. Therefore, Shimobaba *et al.* [3] introduced the concept of the Wavefront Recording Plane (WRP), a virtual plane close to the object. The wavefront from the WRP to the desired hologram plane is then propagated using angular spectrum propagation. In these works, as in any CGH method, efficient occlusion detection and culling are crucial parts . Light field methods [4] are based on individual rays and therefore intrinsically take occlusion into account. Occlusion can be computed using either an approximation [1], raycasting [5], or a combination of raycasting with layer-based methods [6]. All of these techniques trade quality for a computational overhead. We improve both aspects by using modern hardware capabilities, leading to real-time calculation of high-resolution holograms, which is required by many applications like near-eye displays for augmented reality.

2. Method

Our algorithm follows the established PB method [1]. We store the phase distribution for a point of a certain distance similarly to [7]. The point spread function of each OP is limited to a circle with a radius derived from the grating equation. The pixel pitch $p = w_x/H_x = w_y/H_y$ of a hologram of size w_x, w_y and number of pixels H_x, H_y, leads to a maximum reconstruction angle at the wavelength λ: $\theta_{\max} = \arcsin\left(\frac{\lambda}{2p}\right)$. The cone defined by $\theta_{\max}$ projects a circle with radius $r = z_0 \tan\theta_{\max}$ onto the hologram plane limiting the influence of each OP (x_0, y_0, z_0).

Our algorithm splits the circular shape of the Point Spread Function (PSF) into a fan of triangles (Fig. 1(E)). For each segment, an occlusion ray is cast from its center to the respective OP using the programmable raytracing pipeline of the GPU. In case of occlusion, this segment is marked in a mask and thus ignored by later stages. Hence, the overall complexity of occlusion processing is independent of size and resolution of the hologram and depends only on the number of OPs and the fan subdivision. Since the occlusion computation prevents parts of the PSF circle from being rasterized, casting occlusion rays reduces the rasterization costs, leading to a net improvement on both quality and performance. Nominally, our method computes holograms of $300k$ OPs in 185 ms with occlusion vs. 179 ms without.

The point spread functions of neighboring OPs overlap to a high degree. This introduces a race condition when the contributions of different OPs to a certain hologram pixel need to be stored for summation. Synchronization using atomic operations can have a big performance impact and increases implementation complexity. Therefore,

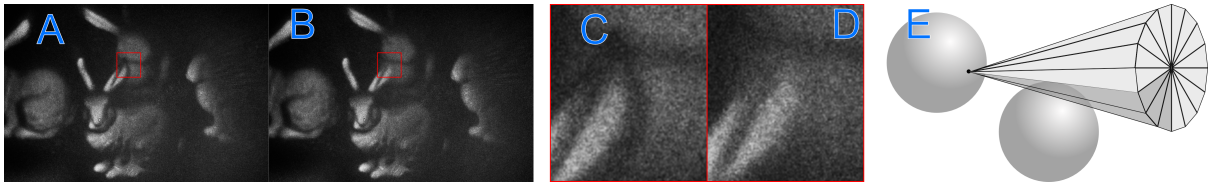

Fig. 1: Exemplary result of our method with occlusion (A, C) versus no occlusion (B, D). Occlusion method (E)

we opt for a hardware solution. Our algorithm uses the rasterization pipeline stage provided by modern Nvidia GPUs consisting of task shader, mesh shader, and the rasterizer.

Similar to compute shaders, both mesh and task shaders are scheduled by specifying a number of work groups consisting of a predetermined number of threads (in our case 32). A mesh shader work group outputs a single *meshlet* with an upper bound on the number of vertices and triangles. An optional task shader can determine the number of meshlets emitted by a task work group. In the first stage of the pipeline, we use the task shader to schedule groups of fan segments for processing by the mesh shader. Each task shader outputs at most 16 segments for a whole triangle fan. The occlusion mask computed in the raycasting stage is read here to cull occluded segments and determine the number of segments that need to be forwarded to the mesh shader. We set the maximum meshlet size to 17 vertices and 16 triangles. The output of the mesh shader is a vertex and index buffer that is directly forwarded to the rasterization stage. The hardware rasterizer splits each triangle into a set of pixel fragments and executes a fragment program on these to determine the fragment color. A separate configurable blending step combines these fragments to determine the final pixel color. We encode the complex valued phase and amplitude information from each OP as a 2D vector. The blending step is configured to add together the resulting values to create the final phase/amplitude distribution. This step implicitly handles the synchronization and eliminates any potential race condition. Therefore, the hardware rasterization pipeline is handling scheduling, synchronization, accumulation, and interference computation without the need for any manual optimization.

As can be seen in Fig. 1(A-D), our method improves sharpness in the corners of objects (C vs. D) where background points become visible due to the shifting perspective.

3. Conclusion

We presented an algorithm that uses modern hardware features to accelerate PB computer generated holography with approximate occlusion. Using the hardware rasterizer, we were able to tightly define the area on the hologram covered by the PSF that needs to be computed and thus more easily saturate the graphics hardware without the need for any manual thread and memory optimizations. The algorithm is fast enough to allow interactive performance even on consumer hardware from the last generation. Furthermore, we have shown that approximating occlusion with one occlusion ray per segment of a triangulated Fresnel zone plate, we could decouple the performance impact of the occlusion computation from the hologram resolution and minimize its cost by reducing the amount of fragments generated by the hardware rasterizer.

Acknowledgements

The authors gratefully acknowledge funding from the German Research Foundation (DFG) under Germany's Excellence Strategy within the Cluster of Excellence PhoenixD (EXC 2122, Project ID 390833453).

References

1. R. H.-Y. Chen and T. D. Wilkinson, "Computer generated hologram from point cloud using graphics processor," Appl. Opt. **48**, 6841–6850 (2009).
2. C. Petz and M. Magnor, "Fast hologram synthesis for 3D geometry models using graphics hardware," in *Practical Holography XVII and Holographic Materials IX,* , vol. 5005 (SPIE, 2003), pp. 266 – 275.
3. T. Shimobaba, N. Masuda, and T. Ito, "Simple and fast calculation algorithm for computer-generated hologram with wavefront recording plane," Opt. Lett. **34**, 3133–3135 (2009).
4. L. Shi, F.-C. Huang, W. Lopes, W. Matusik, and D. Luebke, "Near-eye light field holographic rendering with spherical waves for wide field of view interactive 3d computer graphics," ACM TOG **36**, 1–17 (2017).
5. D. Blinder, M. Chlipala, T. Kozacki, and P. Schelkens, "Photorealistic computer generated holography with global illumination and path tracing," Opt. Lett. **46**, 2188–2191 (2021).
6. A. Gilles, P. Gioia, R. Cozot, and L. Morin, "Hybrid approach for fast occlusion processing in computer-generated hologram calculation," Appl. Opt. **55**, 5459–5470 (2016).
7. S. Fricke, R. Caspary, S. Castillo, and M. Magnor, "Adaptive gaussian points for faster and better computer-generated holograms," in *Digital Holography and Three-Dimensional Imaging 2022,* (Optica Publishing Group, 2022), p. DW3A.4. To Appear.