Research Article Vol. 27, No. 3 | 4 Feb 2019 | OPTICS EXPRESS 2001

Optics EXPRESS

Coherent subhologram-based computer generated display hologram

PETR LOBAZ[1,2,*]

[1] *Department of Computer Science and Engineering, Faculty of Applied Sciences, University of West Bohemia, Univerzitní 8, 306 14 Plzeň, Czech Republic*
[2] *NTIS – New Technologies for the Information Society, Faculty of Applied Sciences, University of West Bohemia, Univerzitní 8, 306 14 Plzeň, Czech Republic*
[*] *lobaz@kiv.zcu.cz*

Abstract: A modification of the subhologram-based approach to the calculation of a computer generated display hologram of a 3-D scene is presented. The hologram area is split into subholograms. For each of them, a perspective image with depth coordinate is rendered and converted to a point cloud. These points are processed in a novel way and the optical field in the subhologram is calculated. The processing guarantees that the points used for the optical field calculation are the same for all subholograms (i.e., coherent). The coherence provides sharp hologram reconstruction. The parameters of the processing are designed to match the human vision requirements.

1. Introduction

There are many approaches to 3-D display construction [1]. One of them is based on holography [2,3]. Conventional optical holography can provide ultra-realistic images of existing objects [4]. Computer generated holography (CGH) can be used to calculate a synthetic hologram of a virtual 3-D scene. The calculated hologram, i.e. a diffractive pattern, is then displayed somehow. The most common ways are 1. displaying using an optical system involving electronic microdisplays (spatial light modulators, SLM), 2. "printing" using a lithography system and illuminating the "print" in the same way as a conventional optical hologram. A reader not familiar with CGH should learn the basics from, e.g., [5] or other introductory text before reading the rest of the paper. Please note that technologies that "print" holograms using light interference, such as Geola / Zebra Imaging system or dot-matrix machines [4,6] are not discussed here.

The calculation methods that target SLMs usually exploit SLM limitations – small space-bandwidth product (a few megapixels) and a small diffraction angle (a few degrees). This allows optimizations such as using large look-up tables that lead to real-time hologram generation necessary for video display (many holograms per second).

Calculation methods targeting "printed" holograms must be able to deal with large diffraction angles and high space-bandwidth product. For example, a "printed" hologram 65×65 mm^2 with pixel pitch 1 μm leads to a diffraction angle around 15° for visible light and 4.2 Gpixels in total. This paper targets primarily the latter case.

There are two important approaches that have been used for the production of large "printed" holograms. They mainly differ in solving the hidden surface elimination. We can call them "the polygonal approach" and "the subhologram approach".

The first one – the polygonal approach – was proposed by Matsushima and Kondoh in [7]. The algorithm is being constantly refined, for more recent version see, e.g., [8]. Here, the scene is composed of polygonal objects. First, the polygons are sorted by their distance from the hologram plane. The first polygon (the farthest one) is treated as an optical field defined in a plane. The optical field is propagated to the plane of the second polygon. The optical field there is replaced by a texture of the second polygon at points where the polygon is located. The optical

#352424 https://doi.org/10.1364/OE.27.002001
Journal © 2019 Received 23 Nov 2018; revised 20 Dec 2018; accepted 20 Dec 2018; published 23 Jan 2019
Corrected: 6 Feb 2019

field is then propagated to the plane of the third polygon, and so on. Finally, the optical field is propagated to the hologram plane.

The second one – the subhologram approach – is a general method that shares a lot of properties with holographic stereograms, see, for example, [9–11]. Here, the hologram area is decomposed to small areas – subholograms. A perspective image is calculated for each subhologram, the camera is usually located at the subhologram center. In the case of stereograms, each pixel of the perspective image is treated as light coming from a particular direction. Thus, the Fourier transform can be used to calculate the optical field in the subhologram. Another approach gets spatial coordinates of the point that is visible through a pixel of the perspective image and treats that point as a point light source. The optical field in the subhologram is then just a sum of optical fields from all visible point light sources, see, e.g., [12].

It is a well-known fact that the subhologram approach does not work well when the objects are located far away from the hologram plane – such objects get blurred. Wakunami and Yamaguchi designed a simple method to overcome that [13, 14]. The stereogram is calculated in a plane close to the objects (the authors call it "the ray-sampling plane") and the light field is then propagated to the hologram plane. The method was recently much improved by Igarashi et al. [15, 16].

Although the aforementioned methods proved their qualities, they also have several drawbacks.

The polygonal approach calculates the whole hologram area at once. Thus, large holograms require a vast amount of shared computer memory. It follows it is difficult to split the calculation between several computers or independent computational nodes. The polygonal approach is also very different from computer graphics, i.e., everything that has been done in computer graphics so far must be re-invented for CGH calculation – illumination model implementation, reflections, transparency, acceleration techniques, hardware support, etc.

The subhologram/"ray-sampling plane" approach also requires to keep the whole hologram in a shared memory (despite the fact that the method was substantially improved in this context in [16]). Contrary to the polygonal approach, the advantage of the subhologram approach is that it utilizes computer graphics for most tasks (illumination, hidden surface removal, etc.) and uses wave optics just in the last steps.

Both approaches also share a common drawback: the calculation precision is mostly driven by the resolution of the final hologram, not by the resolution of human vision.

This paper introduces a modification of the subhologram approach. It states that the critical problem of the method is the loss of coherence between subholograms. It describes a method how to keep the coherence – see Fig. 5 later in the paper for a self-explanatory example. It also describes a method that separates parameters of perspective images rendering (they are driven by the resolution of human vision) from parameters of the hologram calculation (they are driven by the wave properties of light).

The design objectives of the proposed methods were as follows. 1. Small memory footprint – no requirement of keeping the whole hologram in the memory. 2. Easy parallelization and distribution between many computers, minimum synchronization. 3. Easy scalability – the method must be capable of hologram calculation of any reasonable size.

The method was implemented as a proof of concept, i.e., a fast calculation was not the main goal. Please note that in the rest of the paper, the word "hologram" is not used in a conventional way. In fact, the complex optical fields are being calculated rather than holograms. A hologram should then encode the amplitude and phase of the optical field (the wavefront) somehow, for example by taking the intensity of its interference with a reference wave [3]. This encoding is omitted here; when talking about a hologram or a subhologram, we will talk about "a full complex hologram", i.e., about an unencoded optical field.

2. The method

2.1. The algorithm overview

The algorithm is very similar to the calculation of a holographic stereogram, a few additional steps introduce the features sketched in Section 1. The hologram area is split into subholograms. For each subhologram, a perspective image of a 3-D scene with additional depth coordinate is calculated. This image contains in fact locations of point light sources that are used for the subhologram calculation.

However, the perspective image is not used directly to calculate the subhologram. The computer graphics part and the wave calculation part are connected through a data structure called “a global buffer”. The global buffer allows storing location, phase, and amplitude of point light sources that are visible from the hologram plane. In the beginning, the global buffer is empty. In each subhologram calculation, new point light sources can be added according to the corresponding perspective image of the scene. Implementation details of the global buffer will be given in Sec. 2.2.

The optical field in the subhologram is actually calculated from point light sources stored in a simple data structure called “a local buffer”; a simple list or an array is sufficient. It stores locations, amplitudes, and phases of point light sources. The complex amplitude U of light at a point $\mathbf{x}$ is simply given by

$$U(\mathbf{x}) = \sum_{i=1}^{N} \frac{A_i}{|\mathbf{x} - \mathbf{x}_i|} \exp\left[j(2\pi|\mathbf{x} - \mathbf{x}_i|/\lambda + \phi_i\right], \tag{1}$$

where N is the count of point light sources in the local buffer, A_i, ϕ_i and $\mathbf{x}_i$ are the amplitude, the phase and the location of the i-th point light source, λ is the wavelength and $j^2 = -1$.

The algorithm proceeds as follows:

1. Make an empty global buffer and an empty local buffer.
2. Determine subholograms count and their positions in the hologram.
3. For each subhologram:
   (a) Clean the local buffer, mark all lights in the global buffer as “unused”.
   (b) Set the perspective camera location, calculate the perspective image with depth coordinates.
   (c) For each non-empty pixel p of the perspective image:
      i. Calculate location $\mathbf{x}'_p$ of the point light source (from the pixel indices, the camera coordinates, and the depth value).
      ii. Try to add the light source at $\mathbf{x}'_p$ to the global buffer:
         - If the global buffer already contains a point light source located at $\mathbf{x}_p$ close to $\mathbf{x}'_p$, mark it as “used”. Rewrite its amplitude to a value corresponding to the intensity of the pixel p.
         - Otherwise, generate a random phase ϕ_p, determine the amplitude A_p and add $(\mathbf{x}'_p, A_p, \phi_p)$ to the global buffer and mark it as “used”. Note that the global buffer stores quantized locations, i.e., the location actually stored differs slightly from $\mathbf{x}'_p$. Also note that while the location and the phase are truly global and never change, the amplitude can be rewritten for each subhologram. This allows mimicking directional radiance distribution.
   (d) Scan the lights in the global buffer and remove the suspicions ones, e.g., the ones that make “thick surfaces” (see Fig. 3(b) and Sec. 2.5 for details).

(e) Copy all "used" lights from the global buffer to the local buffer.

(f) Calculate the subhologram using Eq. (1) and lights in the local buffer.

We will assume that the hologram is located in the plane $z = hologramZ$, the scene is located in the half-space $z < hologramZ$ and the observer in $z > hologramZ$. Note that both loops (items 3 and 3c) are the conventional ones, i.e., the body of the loop must by executed one by one (not in parallel). The order of execution in not specified here; we will discuss it in Sec. 2.4.

It is worth emphasizing that the points present in the global buffer approximate the scene by a point cloud. Their presence in the global buffer guarantees that the same location and phase will be used for the whole hologram (thus, it provides coherence). The incremental building of the point cloud guarantees that we work just with points that can affect the hologram. Finally, well-chosen parameters of the global buffer and perspective images guarantee that the precision of the point cloud matches human vision requirements; see Sec. 3 for details.

2.2. The global buffer

The data structure that implements the global buffer must be able to quickly find a point in the vicinity of a searched one. It must be able to store the points with the precision necessary for good human perception, but not necessarily higher.

In the proof-of-concept implementation, I have used "an extended perspective image" for the global buffer, see Fig. 1(a). The global buffer is, in fact, a 2-D array, where each element stores a list of points. We can imagine it as a screen (located in the plane $z = gBufferZ$) of a perspective camera located somewhere in the plane $z = gBufferEyeZ$. Each element of the global buffer then stores a list of points (point light sources) lying on the ray from "the global buffer camera" through the center of that element. Note that an element of the global buffer contains just points of the scene that can be seen from some point of the hologram.

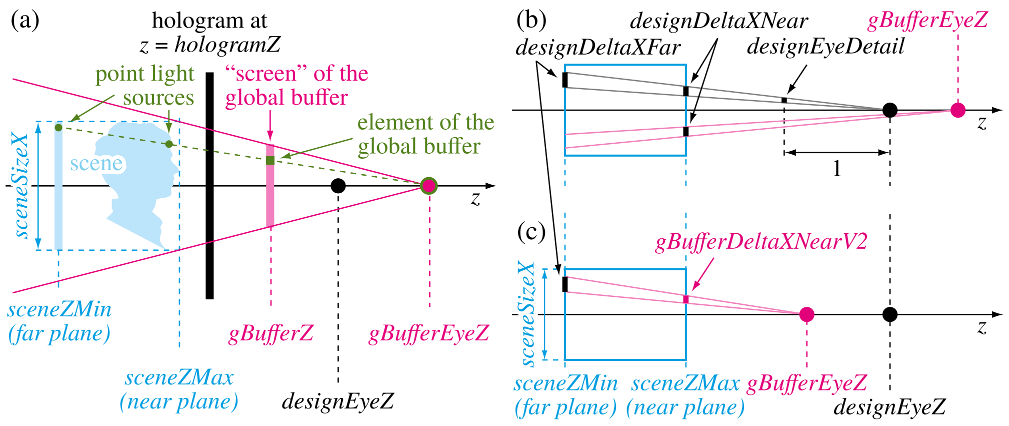

Fig. 1. (a) Global buffer and the scene geometry. (b) Position of the global buffer camera for $gBufferEyeZ \geq designEyeZ$. (c) For $gBufferEyeZ \leq designEyeZ$.

When searching a point in the global buffer, we can easily determine the element of the global buffer that should contain it. In this element, we must then look for a point with a similar depth coordinate. For simplicity, we can use just tolerance $|z_B - z_0| < \Delta_Z$, where z_B is the depth coordinate of the point already present in the global buffer, z_0 is the depth coordinate of the point we are looking for, Δ_Z is a constant set with respect to human vision, e.g. 0.1 mm.

If we cannot find a point in "the correct" element of the global buffer, we can try to scan the elements in the close neighbourhood for reasons explained in Sec. 4.

If the point we are looking for is not present in the global buffer, we can quantize its x, y coordinates in such a way that "the global buffer camera", the center of the global buffer element and the quantized point are collinear. The quantized coordinates, the point phase, and amplitude are stored in the list of an appropriate global buffer element. Note that coordinate quantization is actually not necessary, but it leads to somewhat simpler implementation. Also, note that the point amplitude is stored just temporarily, it can be replaced any time so that it represents point intensity as seen from a different viewpoint (see the step 3(c)ii in the algorithm in Sec. 2.1). Finally, each light stored in the global buffer also stores a boolean value that tells if that point should be used for the calculation of a current subhologram.

We should naturally ask: how to set *gBufferZ*, *gBufferEyeZ*, the global buffer screen size and its resolution? The answer will be given in Sec. 3.

2.3. Rendering of perspective images

In theory, a subhologram should be affected by all points of the scene that illuminate it. In practice, we can restrict the points to the inside of a pyramid. Its apex angle 2α is usually given by the pixel pitch Δ_H of the hologram, as the maximum angle of light incidence w.r.t. normal that can be sampled is given by $\alpha = \operatorname{asin}(\lambda/2\Delta_H)$. Even in the case of sub-micron pixel pitch, it is worth limiting the apex angle – the diffraction efficiency of a hologram is usually low for large angles anyway.

The viewing angle of the perspective camera can be thus set to 2α. The camera is then placed behind the subhologram in such a way the subhologram forms the projection screen of the camera, see Fig. 2(a). Please note that in the case of holographic stereograms, the camera is usually placed to the center of the subhologram. This means that such a camera cannot see points that are very close to the hologram and the subhologram boundary, outside the camera viewing angle. However, holographic stereograms assume that the scene is located far from the hologram plane, which means that this glitch is not important there.

In the proof-of-concept implementation, I assumed the scene is located behind the hologram. Extension to both sides of the hologram should not be difficult.

The only remaining question is how to set the pixel count of the perspective image. It is answered in Sec. 3.

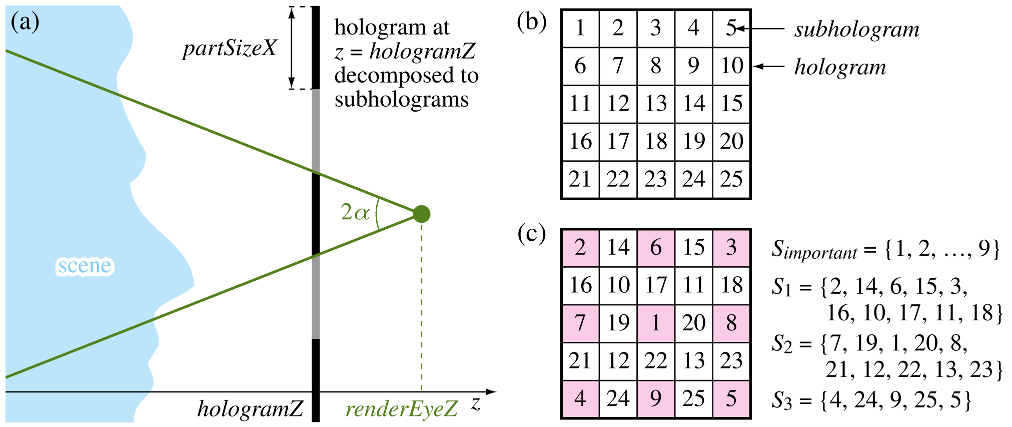

Fig. 2. (a) Rendering of the perspective image. (b) Simple order of subholograms calculation. (c) Diamond-square order of subholograms calculation.

2.4. Loop execution order

The current description of the algorithm does not meet the design objectives: the loop that iterates over all subholograms (item 3 in Sec. 2.1) cannot be run in parallel. It is thus difficult to distribute the calculation between independent computational nodes. The problem is that once some points are added to the global buffer, all following subhologram calculations must take them into account.

This is especially inconvenient if we traverse the subholograms in a simple order (see Fig. 2(b), a hologram is split to 5×5 subholograms). When calculating the first subhologram, the global buffer is filled with points that approximate the scene as seen from the subhologram 1. When calculating subsequent subholograms, only points that were not visible from the previous ones are added to the global buffer. As the subholograms are next to each other, only a few points are usually added in each subhologram.

A better order of subholograms traversal should add the points to the global buffer as fast as possible. We can use "the diamond-square algorithm" [17], see Fig. 2(c). Here, the first few subholograms are located far from each other, thus the scene is rendered from very different viewpoints first. It is very likely that the point approximation of the scene in the global buffer becomes mostly complete after traversal of a few percent of the subholograms. Let us denote the set these "important subholograms" as $S_{important}$.

If we split the calculation between several computational nodes, we usually assign a few rows of subholograms to each node. Let us denote the set of subholograms assigned to the node N as S_N (Fig. 2(c) decomposes the calculation to 3 nodes). The computational node N traverses all subholograms of the hologram in the diamond-square order. If the subhologram belongs to $S_{important}$, the points of the scene are added to the global buffer. If the subhologram belongs to S_N, the optical field in the subhologram is calculated. All subholograms that do not belong to $S_{important}$ or S_N are skipped at all.

This way guarantees that all computational nodes work with the same global buffer. As the number of subholograms in $S_{important}$ is much smaller than the number of all subholograms, the number of subholograms that are traversed "unnecessarily" (outside S_N) is acceptably small.

The subholograms in $S_{important}$ are depicted by red squares in Fig. 2(c). It is an open question how the structure and the size of $S_{important}$ affects the visual quality of the hologram. A reasonable assumption is that if the distance between the closest "red squares" is about 2 mm, i.e., the diameter of a human eye pupil, nothing visually important is missed. It is definitely possible to construct an artificial scene that breaks this assumption; only testing of many practical scenes will tell if the assumption is acceptable in practice.

2.5. Global buffer filtering

The locations of points we are working with stem from perspective images. Most rendering algorithms introduce quantization to pixels, some of them also quantize the depth coordinate in order to use the z-buffer algorithm. It follows that some errors appear further in the process.

The most simple case is shown in Fig. 3(a). A flat patch in $z = z_0$ is rendered from viewpoints $renderEye_A$ and $renderEye_B$. Clearly, the rendered points are not the same, i.e., the coherence between subholograms is lost. Note that pure holographic stereogram based algorithms such as [10, 11] cannot fix it by tweaking the Fourier transform of the rendered images – the images simply do not display the same scene points. Also note that the problem does not appear when the scene is given as a point cloud. On the other hand, point cloud representation of a 3-D scene has its own problems and is not discussed here.

The proposed algorithm solves the problem by storing the rendered points into the global buffer and removing those that are too close to the ones already stored. Despite the effort, problems can appear for surfaces nearly parallel to the "rays" of the global buffer camera. The light blue lines in Fig. 3(b) are contour lines (isolines) of a surface that is somewhere almost parallel to the plane

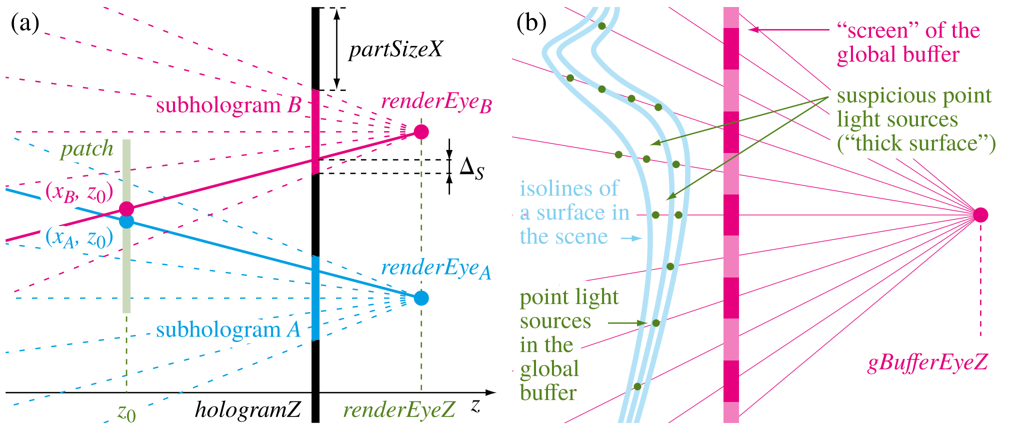
(a)
partSizeX
subhologram B
renderEye_B
patch
Δ_S
(x_B, z_0)
(x_A, z_0)
renderEye_A
subhologram A
z_0
hologramZ
renderEyeZ
z
(b)
"screen" of the global buffer
suspicious point light sources ("thick surface")
isolines of a surface in the scene
point light sources in the global buffer
gBufferEyeZ

Fig. 3. (a) Rendering of a flat patch for subholograms A and B. (b) Errors in the approximation of the surface.

containing the fan of the purple global buffer rays. It follows that points that are going to be stored to the global buffer strongly depend on the location of the perspective image camera. Points of that surface, as seen from various viewpoints, can then make "a thick surface" – the surface is represented by a thick point cloud rather than by a simple skin. Note that the points stored in the global buffer in Fig. 3(b) are quantized, i.e., they are located on the purple lines. Avoiding quantization does not solve the problem – it is necessary to guarantee that all subholograms work with the same points. Thus, it is necessary to to detect such areas and clean them.

In the current implementation, a simple algorithm is used. If an element of a global buffer contains just one point, it will be kept. If it contains points $\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \dots$ at $z_0 > z_1 > z_2 > \cdots$, the frontmost point ($\mathbf{x}_0$) will be kept. The point $\mathbf{x}_i$, $i > 0$, will be kept in case $z_{i-1} - z_i$ is large enough, for example 0.2 mm. In the opposite case, i.e., z_i is close to z_{i-1}, it is checked whether at least one surrounding global buffer element contains a point approximately at z_i. Moreover, for the global buffer elements to the left or below the current one, it is furthermore requested that a point approximately at z_i is not marked "to be deleted". If there is such a neighbouring global buffer element, the point $\mathbf{x}_i$ is marked "to be deleted". All points marked "to be deleted" are deleted from the global buffer at the end of the filtering step.

The algorithm still results in some artifacts, see Fig. 8, and should be refined. Anyway, for the proof-of-concept implementation it gives "good enough" results.

3. Parameters of the global buffer and the perspective images

As we are making a display hologram, the starting point in the derivation of optimal parameters should be visual acuity of the human vision system. Let us assume that an average eye can see details of size *designEyeDetail* from the unit distance. For 6/6 vision, it is about 0.3 mm from 1 m, i.e. the angular resolution is about 1 arc minute [3, Chapter 19].

Let us assume that the hologram is located in the plane $z = \mathit{hologramZ}$ and the reference observer watches it from the plane $z = \mathit{designEyeZ}$, see Fig. 1(a). Let us assume for simplicity that the scene is bounded by the far plane $z = \mathit{sceneZMin}$ and the near plane $z = \mathit{sceneZMax}$ and its lateral size is $\mathit{sceneSizeX} \times \mathit{sceneSizeY}$. Moreover, we assume $\mathit{sceneZMin} < \mathit{sceneZMax} \leq \mathit{hologramZ} < \mathit{designEyeZ}$.

Let us restrict the reasoning to the xz plane, the results for the yz plane are analogous. In the near and far planes, see Fig. 1(b), the eye of the reference observer sees details of size

$$designDeltaXNear = (designEyeZ - sceneZMax) \times designEyeDetail, \tag{2}$$

$$designDeltaXFar = (designEyeZ - sceneZMin) \times designEyeDetail. \tag{3}$$

3.1. Parameters of the global buffer

"The screen" of the global buffer is in the plane $z = gBufferZ$, the camera in the plane $z = gBufferEyeZ$. Without loss of generality, let us assume $gBufferZ = hologramZ$. The parameters we are looking for are thus $gBufferEyeZ$, the size of "the screen" and its resolution.

If we assume that the scene is bounded by a box, then "the screen" size is given by the dimension $sceneSizeX$ in the near plane, see Fig. 1(a).

The resolution of the global buffer must be at least as fine as the resolution of the reference observer. In the case $gBufferEyeZ \geq designEyeZ$, the number of samples (elements) of the global buffer is given by the details visible in the near plane, i.e.

$$gBufferSamplesXPreV1 = \frac{sceneSizeX}{designDeltaXNear}, \tag{4}$$

as the camera of the global buffer then sees details finer than $designDeltaXFar$ in the far plane, see Fig. 1(b). The number of samples (elements) is thus independent of $gBufferEyeZ$.

In the opposite case, i.e. $gBufferEyeZ < designEyeZ$, we should begin with details that the reference observer sees in the far plane, see Fig. 1(c). Such details, as seen by the eye of the global buffer, correspond to details

$$gBufferDeltaXNearV2 = designDeltaXFar \times \frac{gBufferEyeZ - sceneZMax}{gBufferEyeZ - sceneZMin} \tag{5}$$

in the near plane. This leads to the number of samples of the global buffer

$$gBufferSamplesXPreV2 = \frac{sceneSizeX}{gBufferDeltaXNearV2}\ . \tag{6}$$

It holds $gBufferSamplesXPreV1 < gBufferSamplesXPreV2$ for $gBufferEyeZ < designEyeZ$, i.e. the latter case leads to higher number of the global buffer samples (elements). In particular, the details in the near plane can get unnecessarily small for $gBufferEyeZ \ll designEyeZ$. This leads to conclusion that just the case $gBufferEyeZ \geq designEyeZ$ is practical.

There are two important special cases. For $gBufferEyeZ = \infty$, the global buffer actually uses the parallel projection instead of the perspective one. The global buffer then resolves smaller details in the far plane than necessary. For $gBufferEyeZ = designEyeZ$, the global buffer resolves exactly the same details as the reference observer throughout the scene. Let us pick the case $gBufferEyeZ = designEyeZ$. Please note that in case of scene bounding volume other than a box, the analysis might give different results.

Number of samples must be a natural number; we must round $gBufferSamplesXPreV1$ and add 1. In practice, slight errors can appear near the global buffer boundary. To minimize them, it is worth enlarging "the screen" of the global buffer a bit, for example by one sample on each side. The final number of samples is then

$$gBufferSamplesX = \text{ceiling}(gBufferSamplesXPreV1) + 3, \tag{7}$$

where ceiling(x) is the nearest bigger or equal integer than x.

Thus, the global buffer resolves details on the near and the far planes:

$$globalDeltaXNear = \frac{sceneSizeX}{gBufferSamplesX - 3}\ , \tag{8}$$

$$globalDeltaXFar = globalDeltaXNear \times \frac{gBufferEyeZ - sceneZMin}{gBufferEyeZ - sceneZMax}\ . \tag{9}$$

3.2. Parameters of perspective images

The size (the distance from the first sample to the last sample) of the subhologram is

$$partSizeX = \Delta_H \times (partSamplesX - 1), \tag{10}$$

where Δ_H is the hologram pixel pitch and *partSamplesX* is the number of columns of the subhologram, see Fig. 2(a).

The viewing angle of the perspective camera must cover the subhologram. Its z coordinate must be then

$$renderEyeZ = \frac{partSizeX}{2\tan\left(maxDiffractionAngleX\right)} + hologramZ, \tag{11}$$

where *maxDiffractionAngleX* is the maximum angle of incident light (w.r.t. hologram normal) we are willing to capture. It must naturally hold

$$maxDiffractionAngleX \le \operatorname{asin}\left[\min\left(1, \frac{\lambda}{2\Delta_H}\right)\right]. \tag{12}$$

As noted in Sec. 2.3, *maxDiffractionAngleX* is usually limited to a reasonably small value for small hologram pixel pitch (e.g. 100 nm), for example to 30°. It is worth emphasizing the limitation is not mandatory; on the other hand, it would be impractical to use the plain perspective projection for *maxDiffractionAngleX* close to 90°. Other projections such as "the fisheye lens" simulation could help to keep the number of image samples low.

Let us assume that $gBufferEyeZ > renderEyeZ$. The camera resolves the smallest details in the near plane, the resolvable detail in the far plane is the biggest. Resolvable details in the far plane thus determine the perspective image resolution.

The visible part of the far plane has size

$$renderSizeFarX = partSizeX \times \frac{renderEyeZ - sceneZMin}{renderEyeZ - hologramZ} \tag{13}$$

It must be sampled in such a way that the sampling distance matches the resolution of the global buffer in the far plane, i.e.,

$$renderSamplesX = \operatorname{ceiling}\left(\frac{renderSizeFarX}{globalDeltaXFar}\right) + 1 \tag{14}$$

3.3. Examples of parameters values

Let us calculate parameters of the global buffer and the perspective images rendering for a hologram of size 65×65 mm^2 displaying a scene of size $60 \times 60 \times 100$ mm^3 located 100 mm behind the hologram plane, hologram pixel pitch 1 μm (other parameters are given in Tab. 1). Equations of Sec. 3.1 and 3.2 give

$$gBufferSamplesX = 575 \qquad renderSamplesX = 798 \tag{15}$$

If we change the hologram pixel pitch to 0.25 μm and limit *holoDiffractionAngleX* to 30°, we get

$$gBufferSamplesX = 575 \qquad renderSamplesX = 1716 \tag{16}$$

That is, the number of samples is perfectly manageable compared to hologram pixel count. Note that in the first case, the hologram itself is composed of 4.2 Gpixels, in the latter case 67.6 Gpixels.

Given the hologram size, the analysis shows that the number of samples of the global buffer depends mostly on the near plane location *sceneZMax*, the number of samples of the

Table 1. Parameters used for the examples in Sec. 3.3

sceneZMin	−200 mm	*sceneZMax*	−100 mm
designEyeDetail	0.3 mm/m	*designEyeZ*	250 mm
sceneSizeX	60 mm	*gBufferEyeZ*	250 mm
holoSizeX	65 mm	λ	532 nm
holoSamplingDistanceX	1 µm	*holoDiffractionAngleX*	15°
partSamplesX	0.25 mm/*holoSamplingDistanceX*		

perspective image depends mostly on the far plane location *sceneZMin* and the diffraction angle *maxDiffractionAngleX*. Graphs of these two values are shown in Fig. 4. They clearly show that the memory requirements for the global buffer and the perspective images are reasonable for practical purposes.

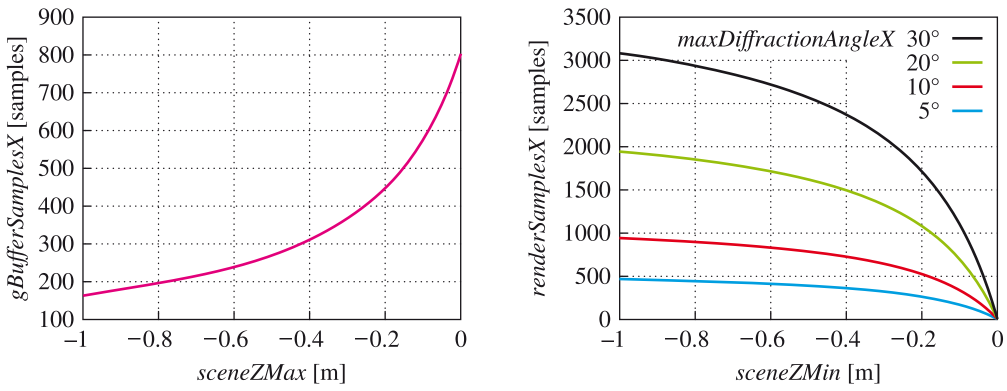

Fig. 4. Numbers of samples (elements) of the global buffer and the perspective images.

3.4. Analysis of the parameters

If we assume *designEyeZ* = *gBufferEyeZ* and approximate Eq. (7) by neglecting rounding and the final addition of two samples, we get a simplified formula

$$gBufferSamplesX \approx \frac{sceneSizeX}{designEyeDetail \times (gBufferEyeZ - sceneZMax)}. \tag{17}$$

Thus, the number of samples of the global buffer is proportional to the lateral dimension of the scene. It is inversely proportional to the distance of the near plane from the design eye location and to the visual acuity.

If we assume small angle approximation ($\sin\phi \approx \phi \approx \tan\phi$), neglect rounding operations and without loss of generality set *hologramZ* = 0, we can simplify Eq. (14) to

$$renderSamplesX \approx \frac{\lambda \times sceneZMin - holoSamplingDistanceX^2 \times partSamplesX}{holoSamplingDistanceX \times designEyeDetail \times (sceneZMin - designEyeZ)}. \tag{18}$$

Here we assume that the maximum diffraction angle is given by the wavelength λ and the hologram pixel pitch *holoSamplingDistanceX*.

It can be shown that the parameter *partSamplesX* does not affect the result too much provided that the subholograms are small and the hologram pixel pitch is small. If we omit this term from the numerator and approximate $\mathit{maxDiffractionAngleX} \approx \lambda/(2\,\mathit{holoSamplingDistanceX})$, we get

$$\mathit{renderSamplesX} \approx \frac{2\,\mathit{maxDiffractionAngleX} \times \mathit{sceneZMin}}{\mathit{designEyeDetail} \times (\mathit{sceneZMin} - \mathit{designEyeZ})}. \tag{19}$$

Now it is easy to see that the number of samples of the perspective image is proportional to the maximum diffraction angle and inversely proportional to the visual acuity. The graph of *renderSamplesX* w.r.t. *sceneZMin* (i.e. the far plane location) is a hyperbola that approaches zero for flat scenes. For deep scenes, is limits to

$$\lim_{\mathit{sceneZMin} \to -\infty} \mathit{renderSamplesX} \approx 2\,\mathit{maxDiffractionAngleX}/\mathit{designEyeDetail}, \tag{20}$$

which means that a finite number of samples is sufficient even for deep scenes.

4. Analysis of the global buffer precision

Let us assume that the scene contains a flat patch in the plane $z = z_0$, $\mathit{sceneZMin} \le z_0 \le \mathit{sceneZMax}$. During the calculation, it is approximated by a point cloud stored in the global buffer. As the key to the success of the algorithm is the coherence between subholograms, there is a fundamental question that affects the whole algorithm: If we approximate the patch by a point cloud in the calculation of a subhologram A, do we get the same point cloud in the calculation of the subhologram B?

During the calculation of the subhologram A, a point (x_A, z_0) is represented by a pixel of the perspective image, see Fig. 3(a). If we know z_0 and the perspective camera parameters, it is easy to calculate x_A from the pixel position.

During the calculation of the subhologram B, the point (x_A, z_0) is also represented by a pixel of the perspective image. Due to quantization to the integer pixel positions, we estimate that the point is located at (x_B, z_0), $x_A \neq x_B$.

If we store the points (x_A, z_0) and (x_B, z_0) to the global buffer, we need to determine the global buffer elements ("pixels") where to store them. For the indices g_A, g_B of the elements, it holds

$$g_A - g_B = \frac{\Delta_S \epsilon_1 (\mathit{gBufferZ} - \mathit{gBufferEyeZ})(z_0 - \mathit{renderEyeZ})}{\Delta_G (\mathit{hologramZ} - \mathit{renderEyeZ})(z - \mathit{gBufferEyeZ})} + \epsilon_2 \tag{21}$$

where Δ_G and Δ_S are the sampling distances of the "global buffer screen" and the "perspective camera screen". Quantization errors are taken into account as numbers $\epsilon_1, \epsilon_2 \in [-1, 1]$. In the derivation of the equation, it was assumed that $\mathrm{round}(x) + \mathrm{round}(y) = (x + \epsilon_\alpha) + (y + \epsilon_\beta) = (x + y) + \epsilon_\gamma$, where $\epsilon_\alpha, \epsilon_\beta \in [-0.5, 0.5]$, $\epsilon_\gamma \in [-1, 1]$.

The difference $g_A - g_B$ is clearly the biggest for $\epsilon_1 = \epsilon_2 = 1$.

If we substitute the parameters of the global buffer and the image rendering derived in Sec. 3.1 and Sec. 3.2 to Eq. (21), and neglect the ceiling operation, we get a simple formula

$$g_A - g_B \approx \frac{(\mathit{designEyeZ} - \mathit{sceneZMin})\,(\mathit{renderEyeZ} - z_0)}{(\mathit{renderEyeZ} - \mathit{sceneZMin})\,(\mathit{designEyeZ} - z_0)} + 1. \tag{22}$$

The difference is the biggest for $z_0 = \mathit{sceneZMin}$; the expression is then equal to 2.

Thus, if the point (x_A, z_0) is stored in the element g_A of the global buffer and we need to look for a point (x_B, z_0), we should search elements $g_B \pm 2$ of the global buffer.

The conclusion is: it is possible to detect identical points in the global buffer (i.e. to ensure coherence) despite the fact that the input perspective images contain quantization errors. It is worth noting that the analysis assumed a patch in the plane parallel to the hologram plane. The result can be worse for non-parallel planes, but the practical experiments show that even then the algorithm gives acceptable results.

5. Implementation and results

The algorithm was implemented in ANSI C. OpenGL was used for the perspective image rendering. Calculation of the optical field according to Eq. (1) was implemented as a custom OpenGL pixel shader. All calculations used the wavelength $\lambda = 532$ nm.

The calculated holograms were numerically reconstructed using algorithms described in [18,19] that efficiently implement the diffraction calculation using the Rayleigh-Sommerfeld convolution.

In the case of the real image reconstruction, a direct propagation from the hologram plane to the plane of observation was used. In case of virtual image reconstruction, a camera simulation was utilized: the optical field was propagated from the hologram plane to the lens plane, a lens function was applied (see, e.g., [2, Chapter 5]), and the modified optical field was propagated to the sensor plane.

Fig. 5 shows results for the scene composed of a single point located at $(0, 0, -5$ mm$)$. If the global buffer is not used, the estimated location of the point differs in each subhologram by as much as 3.2 μm. Such incoherence leads to discontinuities at the subhologram borders and results in a damaged real image reconstructed from the hologram. Note that the damage is not too big as the hologram is intentionally very small in order to show its structure. Larger hologram leads to more blurry reconstruction.

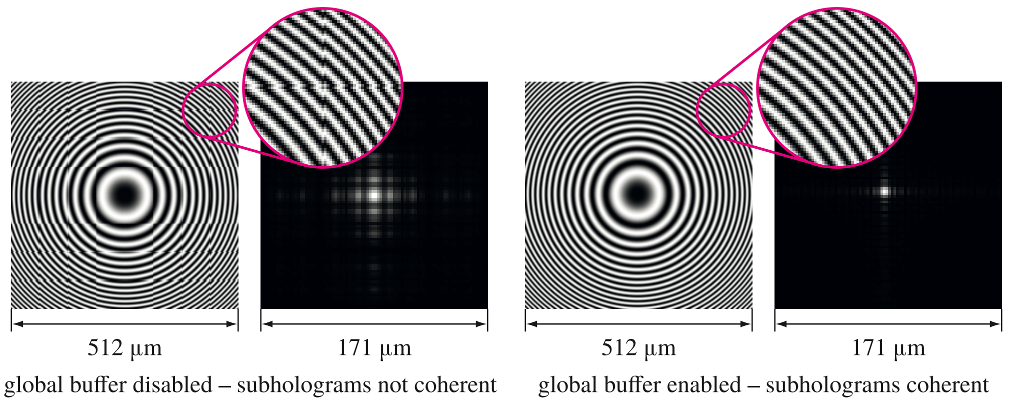
512 µm
171 µm
512 µm
171 µm
global buffer disabled – subholograms not coherent
global buffer enabled – subholograms coherent

Fig. 5. The complex hologram of a single point (real part) and its real image reconstruction.

In the second experiment, three USAF resolution targets of size 5×5 mm^2 are located 100, 120 and 140 mm behind the hologram of size 7.7×4.6 mm^2 (3840×2304 pixels, pixel pitch 2 μm, subhologram size 256×256 pixels). The checkerboard in the background is located 200 mm behind the hologram. The virtual camera is located 200 mm in front of the hologram and is focused to the plane $z = -120$ mm, the real image was reconstructed in the plane $z = -120$ mm. Fig. 6 clearly shows that the virtual image is much sharper when the global buffer is enabled. Note that the results obtained with the global buffer sharply resolve the points of the USAF target. Details visible in the image are proportional to the visual acuity used in the calculation, in this case 0.3 mm/m. Higher visual acuity leads to a more detailed result, see Fig. 7.

In the third experiment, a hologram of a general 3-D scene was calculated. The object size (lion head in "a cage") was 15 mm, its center it was located 150 mm behind the hologram, the background was located 165 mm behind the hologram. Hologram size was 20.5×20.5 mm^2, pixel pitch 1 μm. Two holograms were calculated: with subhologram size 256×256 pixels and 1024×1024 pixels. No significant difference was found, see Fig. 8 and Visualization 1 and Visualization 2. Obviously, a hologram with bigger subholograms can be calculated faster as fewer perspective images must be calculated and processed.

The result is not perfect: there are gaps on surfaces that are almost perpendicular to the hologram plane, see Fig. 8. Examination of the global buffer content showed that these parts of

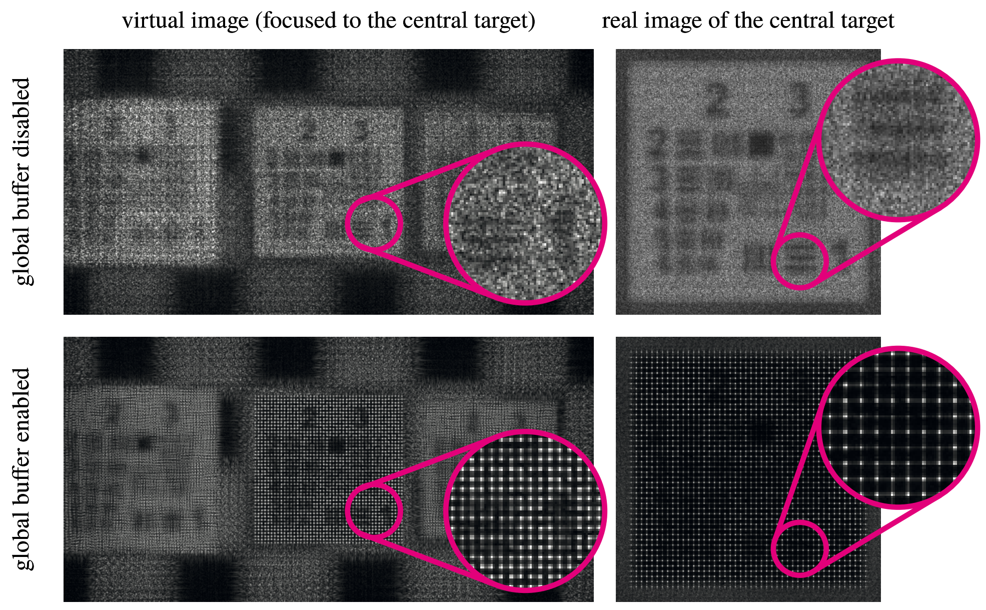
virtual image (focused to the central target)
real image of the central target
global buffer disabled
global buffer enabled

Fig. 6. Reconstructions of holograms of resolution targets, *designEyeDetail* = 0.3 mm/m.

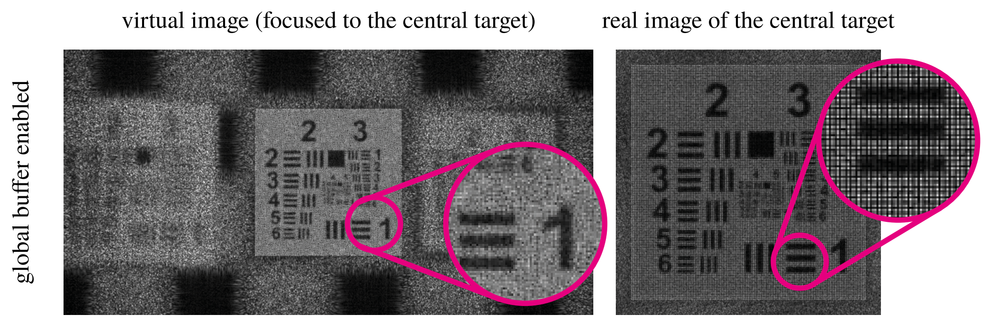
virtual image (focused to the central target)
real image of the central target
global buffer enabled

Fig. 7. Reconstructions of holograms of resolution targets, *designEyeDetail* = 0.1 mm/m.

the scene were not sampled well – the filtering operation removed some point that should be left. It is a future work to improve the filtering.

Note that larger subholograms lead to coarser approximation of the exact hidden surface removal. The subhologram size 0.256 mm was chosen because such subholograms are fine enough for the design viewing distance 250 mm. Thus, the hidden surface elimination was done dense enough. The result for the subhologram size 1.024 mm indicates that there is no need for such density of the hidden surface elimination – the result is visually the same. Is is an open question what is the maximum subhologram size that gives visually acceptable results.

As mentioned in Sec. 1, the goal of the implementation was to check the idea, not to get a fast calculation. For example, calculation of the hologram of a lion head (20480 × 20480 pixels) was split between three common PCs (Intel Xeon E5-2630 v2 2.6 GHz, 16 GB RAM, Windows 10 64bit, NVIDIA GeForce GTX 780 GPU). The calculation took 37 minutes, 94 % of the time was spent on the optical field calculation, about 2 % was spent on the global buffer processing, the image rendering took just 0.05 %, transfers between GPU and the main memory took the rest. In the future, mainly the optical field calculation must be optimized.

Also note that the hologram parameters for the experiments were chosen to show features of

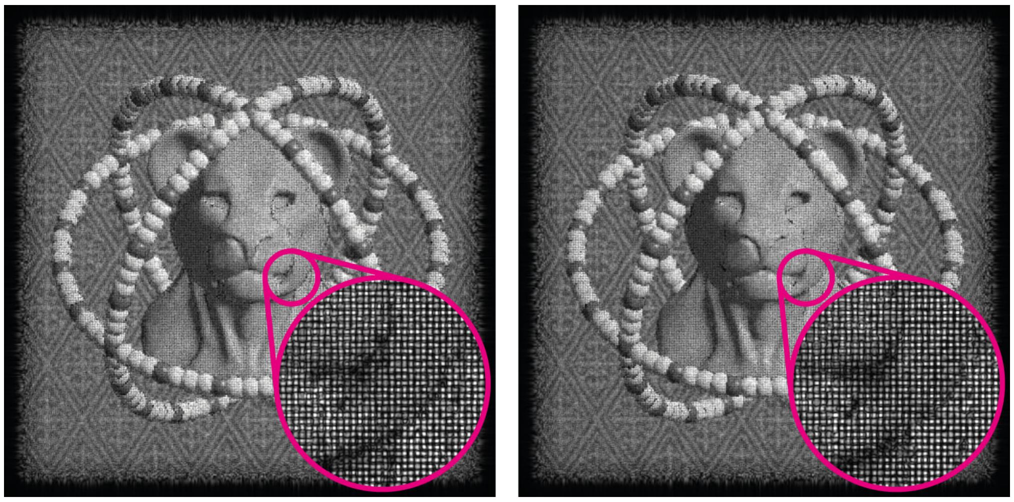

Fig. 8. Virtual image reconstruction. Left: subhologram size 256 × 256 μm, see also Visualization 1. Right: subhologram size 1024 × 1024 μm, see also Visualization 2. The blurred border of the image is the out-of-focus border of the hologram. Note there is no significant difference between the results.

the proposed algorithm. Thus, the distance between the scene and the hologram was chosen quite large to show the sharpness of a deep scene. On the other hand, hologram sizes were chosen rather small in order to keep calculation times of the hologram reconstructions manageable, especially the calculation of Visualization 1 and Visualization 2.

6. Conclusion

A modification of the subhologram-based approach of computer generated display hologram calculation was presented. The modification that guarantees the coherence between subholograms takes a negligible amount of time and provides superior image sharpness. The future works should connect it to a more efficient calculation of the optical field. The global buffer filtering should be refined to avoid the artifacts mentioned in Sec. 5.

Funding

This work was supported by the Ministry of Education, Youth and Sports of the Czech Republic, University specific research project SGS-2016-013 Advanced Graphical and Computing Systems, and project PUNTIS (LO1506) under the program NPU I.

References

1. J. Geng, “Three-dimensional display technologies,” Adv. Opt. Photon. **5**, 456–535 (2013).
2. J. W. Goodman, *Introduction to Fourier Optics* (Roberts & Co., 2004), 3rd ed.
3. S. A. Benton and V. M. Bove, Jr., *Holographic Imaging* (Wiley, 2008).
4. H. Bjelkhagen and D. Brotherton-Ratcliffe, *Ultra-Realistic Imaging: Advanced Techniques in Analogue and Digital Colour Holography* (Taylor & Francis, 2013).
5. P. Lobaz, “Computer generated display holography,” in *EG 2017 – Tutorials,* A. Bousseau and D. Gutierrez, eds. (The Eurographics Association, 2017).
6. G. Saxby and S. Zacharovas, *Practical Holography* (Taylor & Francis, 2015), 4th ed.
7. K. Matsushima and A. Kondoh, “A wave-optical algorithm for hidden-surface removal in digitally synthetic full-parallax holograms for three-dimensional objects,” Proc. SPIE **5290**, 90–97 (2004).
8. K. Matsushima and N. Sonobe, “Full-color digitized holography for large-scale holographic 3D imaging of physical and nonphysical objects,” Appl. Opt. **57**, A150–A156 (2018).

9. T. Yatagai, "Stereoscopic approach to 3-D display using computer-generated holograms," Appl. Opt. **15**, 2722–2729 (1976).
10. M. Yamaguchi, H. Hoshino, T. Honda, and N. Ohyama, "Phase-added stereogram: calculation of hologram using computer graphics technique," Proc. SPIE **1914**, 25–31 (1993).
11. H. Kang, T. Yamaguchi, and H. Yoshikawa, "Accurate phase-added stereogram to improve the coherent stereogram," Appl. Opt. **47**, D44–D54 (2008).
12. T. Ichikawa, K. Yamaguchi, and Y. Sakamoto, "Realistic expression for full-parallax computer-generated holograms with the ray-tracing method," Appl. Opt. **52**, A201–A209 (2013).
13. K. Wakunami and M. Yamaguchi, "Calculation of computer-generated hologram for 3D display using light-ray sampling plane," Proc. SPIE **7619**, 76190A (2010).
14. K. Wakunami, H. Yamashita, and M. Yamaguchi, "Occlusion culling for computer generated hologram based on ray-wavefront conversion," Opt. Express **21**, 21811–21822 (2013).
15. S. Igarashi, T. Nakamura, K. Matsushima, and M. Yamaguchi, "Efficient calculation method for realistic deep 3D scene hologram using orthographic projection," Proc. SPIE **9771**, 977100 (2016).
16. S. Igarashi, T. Nakamura, K. Matsushima, and M. Yamaguchi, "Efficient tiled calculation of over-10-gigapixel holograms using ray-wavefront conversion," Opt. Express **26**, 10773–10786 (2018).
17. A. Fournier, D. Fussell, and L. Carpenter, "Computer rendering of stochastic models," Commun. ACM **25**, 371–384 (1982).
18. P. Lobaz, "Reference calculation of light propagation between parallel planes of different sizes and sampling rates," Opt. Express **19**, 32–39 (2011).
19. P. Lobaz, "Memory-efficient reference calculation of light propagation using the convolution method," Opt. Express **21**, 2795–2806 (2013).