Analytic computation of line-drawn objects in computer generated holography

DAVID BLINDER,1,2,* TAKASHI NISHITSUJI,3 TAKASHI KAKUE,4 TOMOYOSHI SHIMOBABA,4 TOMOYOSHI ITO,4 AND PETER SCHELKENS1,2

1Department of Electronics and Informatics (ETRO), Vrije Universiteit Brussel (VUB), Pleinlaan 2, B-1050 Brussel, Belgium
2IMEC, Kapeldreef 75, B-3001 Leuven, Belgium
3Faculty of Systems Design, Tokyo Metropolitan University, 6-6 Asahigaoka, Hino, Tokyo, Japan
4Graduate School of Engineering, Chiba University, 1-33 Yayoi-cho, Inage-ku, Chiba, Chiba, Japan
* david.blinder@vub.be

Abstract: Digital holography is a promising display technology that can account for all human visual cues, with many potential applications i.a. in AR and VR. However, one of the main challenges in computer generated holography (CGH) needed for driving these displays are the high computational requirements. In this work, we propose a new CGH technique for the efficient analytical computation of lines and arc primitives. We express the solutions analytically by means of incomplete cylindrical functions, and devise an efficiently computable approximation suitable for massively parallel computing architectures. We implement the algorithm on a GPU (with CUDA), provide an error analysis and report real-time frame rates for CGH of complex 3D scenes of line-drawn objects, and validate the algorithm in an optical setup.

1. Introduction

Computer generated holography (CGH) consists of numerical diffraction algorithms for efficiently calculating interference for various applications in holography, such as display technology [1], beam shaping [2] and interferometry [3], among many others. One of the main computational challenges is that every (virtual) scene element can potentially affect all hologram pixels. This problem is exacerbated for holographic video displays, requiring high-resolution hologram generation at video rates.

Holographic displays are often cited as the ultimate 3D display technology [4]. In principle, holograms can display virtual objects which are indistinguishable from real objects since they account for all human visual cues. This makes them promising candidates for augmented and virtual reality applications, since they do not suffer from the drawbacks of head-mounted displays with conventional screens, as they typically cannot account for eye-focusing and exhibit accommodation-vergence conflicts.

There are many kinds of CGH algorithms [4,5], whose relative advantages and drawbacks are reflected in attributes such as computational speed, supported graphical effects and scene geometry, numerical precision and visual realism. One major way of classifying CGH algorithms is by their used elementary constituents: point cloud methods [6–11], layer-based methods [12–14], polygon methods [15–17] and ray-based methods [18–20].

In this work, we investigate the problem of diffracting line-drawn segments. We target applications such as head-mounted displays, near-eye displays, navigation systems; but this algorithm could be used as a building block for CGH of more complex virtual scenes too. This may also have broader applications in pure diffraction theory beyond displays, e.g. the efficient computation of straight or curved line apertures.
This problem was recently addressed in [21], which we designate as the reference “CG line” (computer graphics) CGH method. It subdivides the scene into depth planes, utilizing a pre-computed one-dimensional signal per plane encoding a converged light distribution. For every curve, the buffer is added on the hologram plane for each of a series of equidistant points along the curve with a period equal to the pixel pitch. This buffer is swept along the curve, drawn perpendicularly to the curve’s tangent at every point. The algorithm was successful in creating line-drawn object CGH at multiple depths in real-time.

However, the current version of that method has a few drawbacks. It is primarily designed for shorter distances to the hologram plane and operates within a fixed set of depth planes. Furthermore, noticeable errors are present at the curve’s edges, especially at lower resolutions. To address this, we introduce a novel way for computing the hologram of line-drawn objects. We propose an analytical technique for directly computing the value of diffracted line and circle arc apertures in any point (e.g., pixel). This eliminates the error present at edges, and imposes no constraints on the number of depth levels, up to one per line if needed. Moreover, this analytical formulation makes a parallel implementation more straightforward: we achieve video-rate CGH for driving a HD spatial light modulator (SLM) with complex 3D line-drawn content on a GPU.

The paper is structured as follows: in sections 2 and 3 we derive analytical expressions for the diffraction of curve apertures. We begin with the straight line case, and proceed with the derivation of general circular arc segments. We derive an efficiently computable numerical approximation and provide an error analysis. Then, in section 4, we compare the calculation speed and accuracy with multiple CGH algorithms, including the reference point-based CGH and the CG line CGH algorithms. We examine the visual quality in both simulation environment and validate it in a holographic display setup. Finally, we conclude in section 5.

2. Line apertures

The expression for a single point-spread function $P$, describing the diffraction pattern induced by a luminous point at coordinates $(\delta, \epsilon, \zeta)$, is given by

$$P(x,y) = a \cdot \exp\left(\frac{\pi i}{\lambda \zeta} \left( (x-\delta)^2 + (y-\epsilon)^2 \right) \right)$$  \hspace{1cm} (1)

where $\zeta$ is the distance to the hologram plane, $i$ is the imaginary unit, $\lambda$ is the wavelength and $a \in \mathbb{C}$ is the point amplitude. This expression will be generalized for diffracting line-drawn segments of two kinds: straight line segments and arbitrary circle segment apertures, or arcs.

The diffraction pattern for an arc aperture $A$, cf. Fig. 1(a), can be described by the sum of the set of all $P(x,y)$ points lying on the arc with radius $\rho$ and spanning an angle $\alpha$, and is given by

$$A(x,y) = a\rho \int_0^\alpha \exp\left(\frac{\pi i}{\lambda \zeta} \left( (x-\rho \cos \phi)^2 + (y-\rho \sin \phi)^2 \right) \right) d\phi. \hspace{1cm} (2)$$

The integral is multiplied by $\rho$ to get a constant intensity per unit arc length; e.g. when $\rho$ doubles, the integral “sweeps” twice as fast over the arc, requiring doubling the integral to exactly compensate for the energy loss. For notational simplicity, we assume the arc to be centered at the origin and anchored in the $x$-axis. The solution at the end can be easily adapted for arbitrary arcs later by a change of coordinates. Similarly, we consider a straight line segment parallel lying on the $x$-axis, cf. Fig. 1(b), beginning at the origin with length $\ell$:

$$L(x,y) = a \int_0^\ell \exp\left(\frac{\pi i}{\lambda \zeta} \left( (x-u)^2 + y^2 \right) \right) du. \hspace{1cm} (3)$$

The goal is to solve these integrals and find an efficiently computable analytical expression for the real-time CGH calculation of line-drawn objects. Any curve shape can be approximated to arbitrary precision with lines and arcs, such as arc splines [22]. We will begin with the special case of a straight line, whose solution will serve as a basis for the computation of arbitrary arcs.
Fig. 1. Diagram of luminous line drawn apertures with its parameters. The example arc (a) is centered at the origin and anchored at the x-axis, with radius $\rho$ and spanning an angle $\alpha$. The exemplary straight line (b) lies on the x-axis, begins at the origin and has length $\ell$.

2.1. Straight line apertures

We rewrite (3) as

$$
L(x, y) = a \exp \left( \frac{\pi i}{\lambda \zeta} y^2 \right) \int_0^\ell \exp \left( \frac{\pi i}{\lambda \zeta} (x - u)^2 \right) du
$$

$$
= \frac{1}{2} a \sqrt{2 \lambda \zeta} \exp \left( \frac{\pi i}{\lambda \zeta} y^2 \right) \left[ \text{erf} \left( \sqrt{\frac{\pi}{2 \lambda \zeta}} (1 - i)x \right) + \text{erf} \left( \sqrt{\frac{\pi}{2 \lambda \zeta}} (1 - i)(\ell - x) \right) \right]
$$

(4)

where erf($c$) is the error function. Note that the input argument $c$ to erf($c$) is complex-valued, so we cannot rely on standard built-in functions and algorithms. Although $c$ is complex, we only have to consider input values along the main diagonals of the complex plane, since $c = (1 - i)s$, for some $s \in \mathbb{R}$. This function is therefore closely related to the Fresnel integrals, for which many numerical solutions exist such as table-based calculations. We will use the same approximation function $E(t)$, $t \in \mathbb{R}$ as derived in [11]:

$$
(i - 1) \sqrt{\frac{\pi}{2}} \text{erf}(\frac{1-i}{\sqrt{2}} t) = E(t) \approx \begin{cases} 
E_S(t), & \text{if } |t| < 1.609 \\
E_L(t), & \text{otherwise.} 
\end{cases}
$$

(5)

where $E_S(t)$ and $E_L(t)$ are valid for small and large values of $|t|$. $E_S(t)$ was derived using a Taylor expansion, in separate real ($\mathbb{R}$) and imaginary ($\mathbb{I}$) parts:

$$
\mathbb{R}\{E_S(t)\} = 2t - \frac{1}{5} t^5 + \frac{1}{108} t^9
$$

(6)

$$
\mathbb{I}\{E_S(t)\} = -\frac{3}{4} t^3 + \frac{1}{21} t^7 - \frac{1}{306} t^{11}
$$

(7)

while $E_L(t)$ was obtained by truncating an asymptotic expansion:

$$
E_L(t) = (1 - i) \sqrt{\frac{\pi}{2}} \cdot \text{sgn}(t) + \frac{i \exp \left( -it^2 \right)}{t}
$$

(8)

where $\text{sgn}(t)$ is the sign function, equal to 1 when $t$ is positive and equal to $-1$ when $t$ is negative.

Using $E(t)$, we can rewrite (4) and obtain the final expression

$$
L(x, y) = -\frac{ia}{2\gamma} \exp \left( i\gamma^2 y^2 \right) \left[ E(\gamma x) + E(\gamma(\ell - x)) \right]
$$

(9)

for straight line segment diffraction, where $\gamma = \sqrt{\frac{\pi \zeta}{\lambda}}$. 
3. Arc apertures

We convert expression (2) to polar coordinates \((x, y) = (r \cos \theta, r \sin \theta)\):

\[
A(r, \theta) = a \rho \int_0^\alpha \exp \left( \frac{\pi i}{\lambda \zeta} \left[ (r \cos \theta - \rho \cos \phi)^2 + (r \sin \theta - \rho \sin \phi)^2 \right] \right) d\phi
\]

\[
= a \rho \int_0^\alpha \exp \left( \frac{\pi i}{2} \left[ \rho^2 - 2 \rho \cdot (\cos \phi \cos \theta + \sin \phi \sin \theta) \right] \right) d\phi
\]

\[
= a \rho \exp \left( \frac{\pi i}{2} (r^2 + \rho^2) \right) \int_0^\alpha \exp \left( -i \frac{2 \pi r \rho}{\lambda \zeta} (\cos \phi \cos \theta + \sin \phi \sin \theta) \right) d\phi
\]

which can be simplified further using the identity \(\cos \phi \cos \theta + \sin \phi \sin \theta = \cos (\phi - \theta)\):

\[
A(r, \theta) = a \rho \exp \left( \frac{\pi i}{2} (r^2 + \rho^2) \right) \int_0^\alpha \exp \left( -i \frac{2 \pi r \rho}{\lambda \zeta} \cos \phi \right) d\phi.
\]

The main remaining hurdle is to solve an integral of the form

\[
E(s, w) = \int_0^w \exp(i s t) dt
\]

where \(w\) and \(s\) are arbitrary real-valued parameters of a two-dimensional function we want to be able to efficiently compute. Unfortunately, this integral cannot be expressed in terms of elementary functions. They are closely related to the non-homogeneous Bessel’s differential equations. We start from the expressions [23]

\[
J_\nu(s) = \frac{s^\nu}{2^{\nu-1} \Gamma(\nu + \frac{1}{2}) \sqrt{\pi}} \int_0^{\frac{\pi}{2}} \cos (s \cos t) \sin^{2\nu} (t) dt
\]

\[
H_\nu(s) = \frac{s^\nu}{2^{\nu-1} \Gamma(\nu + \frac{1}{2}) \sqrt{\pi}} \int_0^{\frac{\pi}{2}} \sin (s \cos t) \sin^{2\nu} (t) dt
\]

where \(J_\nu(s)\) is the Bessel function of the first kind of order \(\nu\), \(H_\nu(s)\) is the Struve function of order \(\nu\) and \(\Gamma\) is the gamma function. Using Euler’s formula \(\exp(i s) = \cos(s) + i \sin(s)\) and taking the order \(\nu = 0\), we obtain for the fixed integration bounds \(w = \frac{\pi}{2}, \pi\)

\[
E(s, \pi) = \pi \cdot J_0(s);
\]

\[
E(s, \frac{\pi}{2}) = \frac{\pi}{2} \left( J_0(s) + iH_0(s) \right)
\]

To obtain a solution for all \(w\), we will need to further analyze \(E(s, w)\), which we will call an “incomplete cylindrical function”, following the naming convention in [24].

Although \(E(s, w)\) can be evaluated \(\forall s, w \in \mathbb{R}\), we can reduce the space of values under consideration by leveraging the following symmetries:

\[
E(-s, w) = \overline{E(s, w)};
\]

\[
E(s, w + \pi) = \frac{\pi}{2} J_0(s) + E(s, w);
\]

\[
E(s, -w) = -E(s, w).
\]

where the overbar stands for complex conjugation. Thus we will only consider \(s > 0\) and \(0 \leq w < \frac{\pi}{2}\) for the numerical model, since values for \(s\) and \(w\) outside of these bounds can always be reduced using the symmetries of (16). To efficiently numerically approximate \(E(s, w)\), we will consider two cases: large and small values of \(w\sqrt{s}\), which will be expounded on in the subsections below.
3.1. Asymptotic approximation for large parameters

Substituting \( u = \cos w \) in \( E(s, w) \) and using (15), we obtain

\[
E(s, w) = \int_{\cos w}^{1} \frac{\exp(isu)}{\sqrt{1 - u^2}} du = \frac{\pi}{2} (J_0(s) + iH_0(s)) - \int_{0}^{\cos w} \frac{\exp(isu)}{\sqrt{1 - u^2}} du. \tag{17}
\]

The latter integral can be calculated \([25]\) by using the Fourier series expansion of \((1 - u^2)^{-\frac{1}{2}}\):

\[
\int_{0}^{\cos w} \frac{\exp(isu)}{\sqrt{1 - u^2}} du = \int_{0}^{\cos w} \exp(isu) \left( \frac{\pi}{2} + \pi \sum_{m=1}^{\infty} J_0(m\pi) \cos(m\pi u) \right) du =
\]

\[
\frac{i\pi}{2s} (1 - e^{i\cos w}) + \sum_{m=1}^{\infty} \frac{\pi J_0(m\pi)}{m^2\pi^2 - s^2} \left( e^{i\cos w} \left[ is \cos(m\pi \cos w) + m\pi \sin(m\pi \cos w) \right] - is \right). \tag{18}
\]

Although this series can be used to compute \( E(s, w) \) to arbitrary precision for large \( s \), the convergence is too slow for our intended application, requiring too many calculations. Instead, we propose a different approach. We use the following equality:

\[
\frac{\partial}{\partial u} \frac{-i \exp(isu)}{s\sqrt{1-u^2}} = \frac{\exp(isu)}{\sqrt{1-u^2}} - \frac{iu \exp(isu)}{s(1-u^2)^{\frac{3}{2}}} \tag{19}
\]

showing that the left-hand side of (19) is close to an elementary function solution of the sought integral (18), except for the second term in the right-hand side. But because that term has a \( s \) in the denominator, its contribution will be small for large \( s \). We thus have

\[
\int_{0}^{\cos w} \frac{\exp(isu)}{\sqrt{1-u^2}} du \approx -\frac{i \exp(i s \cos w)}{s \sin w} + \frac{i}{s}. \tag{20}
\]

Finally, we would like to find a way to compute \( H_0(s) \); unlike Bessel functions, there are no standard C++/CUDA implementations of Struve functions at the time of writing. We therefore use the following asymptotic form for Struve functions:

\[
H_\nu(s) - Y_\nu(s) = \left( \frac{i}{s} \right)^{\nu-1} \frac{\nu-1}{\sqrt{\pi \Gamma \left( \nu + \frac{1}{2} \right)}} + O\left( \left( \frac{i}{s} \right)^{\nu-3} \right) \tag{21}
\]

where \( Y_\nu(s) \) is the Neumann function of order \( \nu \), also known as a Bessel function of the second kind, which is easier to calculate. Substituting this for \( \nu = 0 \) gives us:

\[
H_0(s) = Y_0(s) + \frac{2}{\pi s} + O\left( \frac{s^{-3}}{8} \right) \tag{22}
\]

this first error term will exactly compensate the \( \frac{s}{4} \) term derived in (20), resulting in the final expression

\[
E(s, w) \approx E_L(s, w) = \frac{\pi}{2} H_0(s) - \frac{i \exp(is \cos w)}{s \sin w} \tag{23}
\]

valid for larger values of \( s \) and \( \sin w \), where \( H_0(s) = J_0(s) + iY_0(s) \) is the Hankel function of order 0.
3.2. Series approximation for small parameters

For small values of \(w \sqrt{s}\), we need a different method. A natural first approach is to take the Taylor expansion in \(s\) of the expression within the integral of \(E(s, w)\),

\[
\exp(is \cos t) = 1 + is \cos t - \frac{1}{2} s^2 \cos^2 t - \frac{i}{6} is^3 \cos t^3 + O(s^4)
\]  

(24)

which can be integrated term by term to obtain

\[
\int_0^w \exp(is \cos t)dt = w + is \sin w - \frac{1}{2} s^2 (w + \cos w \sin w) - \frac{i}{18} s^3 \sin w (2 + \cos w^2) + O(s^4).
\]  

(25)

Unfortunately, this expression is only useful for very small values of \(|s| < 10^{-2}\); (25) cannot capture the oscillatory behavior of \(E(s, w)\) with a small number of terms.

We propose a different approach by taking the Taylor expansion of the exponent instead:

\[
\exp(is \cos t) = \exp \left( is \left( 1 - \frac{1}{2} t^2 + O(t^4) \right) \right)
\]  

(26)

valid for small values of \(w \approx \sin w\), after integration

\[
\int_0^w \exp(is \cos t)dt \approx \int_0^w \exp \left( is \left( 1 - \frac{1}{2} t^2 \right) \right) dt = \frac{\pi}{2} \exp \left( i \left( s - \frac{\pi}{4} \right) \right) \text{erf} \left( \frac{\frac{\pi}{2} w \sqrt{s}}{\sqrt{2}} \right).
\]  

(27)

Note that this is exactly the same kind of restricted domain error function we approximated with (5) for straight line apertures. Using this function, we get the final expression for \(E_S(s, w)\), only valid for small values of \(w \sqrt{s}\):

\[
E(s, w) = E_S(s, w) = \frac{1}{\sqrt{2\pi}} \exp(is)E \left( \frac{w \sqrt{2\delta}}{2} \right)
\]  

(28)

3.3. Error analysis

To know whether to choose \(E_S(s, w)\) or \(E_L(s, w)\) for any given \((s, w)\), we have to quantify which one has the smallest error. For this we use the following measure of relative precision:

\[
\text{RP} (a, \hat{a}) = -\log_{10} \left| \frac{a - \hat{a}}{a} \right| = \log_{10} |a| - \log_{10} |a - \hat{a}|
\]  

(29)

where \(a\) is the reference value and \(\hat{a}\) is the approximation. The relative precision of \(E_S\) and \(E_L\) w.r.t. \(E\) are shown in Figs. 2(a)–2(b) and their combined maximum in Fig. 2(c). In Fig. 2(d), we evaluate whether \(\text{RP} (E, E_L) > \text{RP} (E, E_S)\) or not for all tested pairs of \((s, w)\). There is a clear separation in two regions, so it suffices to know on what side of the divide a point \((s, w)\) lies, for which we want to find a simple expression. Using linear regression with the curve \(w \sqrt{s} = c\) (where \(c\) is the parameter to be fitted), we get that

\[
\left[ \text{RP} (E(s, w), E_L(s, w)) = \text{RP} (E(s, w), E_S(s, w)) \right] \approx \left( w \sqrt{s} = 2.26 \right)
\]  

(30)

with a goodness-of-fit \(R^2 = 0.9985\) when tested in the region \(s \in [0, \, 10^5]\).

Finally, we have to consider the case \(s \approx 0\), which will cause numerical problems since \(s\) appears in the denominator for expressions in both \(E_S\) and \(E_L\). For this we can use the (truncated) Taylor expansion in (24); in conclusion we get

\[
E(s, w) \approx \begin{cases} 
  w + is \sin(w), & \text{if } s < 10^{-2} \\
  E_S(s, w), & \text{else if } w \sqrt{s} < 2.26 \\
  E_L(s, w), & \text{otherwise}.
\end{cases}
\]  

(31)
Fig. 2. Error maps showing the relative precision of (a) $E_S(s, w)$ and (b) $E_L(s, w)$ w.r.t. $E(r, w)$. The color scale effectively shows the number of correct digits of computed values, clipped between 0 and 5 for better contrast. Note how the high and low precision regions are opposed in the respective maps, shown by taking the maximum of both maps in (c), which will be the effective precision of the algorithm. (d) the transition map shows for a given point $(s, w)$ whether $E_S$ has higher precision (black) or $E_L$ has higher precision (gray); it can be accurately modelled by the curve $(w\sqrt{s} = 2.26)$, shown by the red dashed line.

3.4. Evaluating arc aperture expressions

With the help of the incomplete cylindrical function $E(s, w)$, we can rewrite (11) as

$$A(r, \theta) = a\rho \exp\left(i\gamma^2(r^2 + \rho^2)\right) \left[E(s, \alpha + \theta) - E(s, \theta)\right]$$

(32)

where $s = -2\gamma^2 r \rho$ and $\gamma = \sqrt{\frac{2}{\lambda\kappa}}$ as defined for (9). However, we approximated $E(s, w)$ only for $s \geq 0$ and $0 \leq w \leq \frac{\pi}{2}$, so we need to extend the domain of $E(s, w)$ using the symmetries in (16). Extending for $s$ is more straightforward: the sign of $s$ only depends on $\gamma^2$, whose sign is equal to that of the depth position $\zeta$. Typically, the sign of $\zeta$ for all elements in a scene will be the same, so we can fill in $|s|$ instead of $s$ and take the complex conjugate of the final hologram if $s$ is negative. But this can be done on a per-line-segment basis as well if needed.
The case for $w$ is more tricky, as the input can be many multiples of $\frac{\pi}{2}$, including negative ones. Recursively applying the two last symmetries of (16) would be computationally sub-optimal, as this would introduce branching divergence across computation threads, which is unfavorable for parallel computation implementations. Instead, we combine all cases for $w$ using the following expression:

$$E(s, w) = m\pi J_0(s) + \text{bitsgn}(n)\Re\{E(s, |w - m\pi|]\} + i\text{bitsgn}\left(\left\lfloor \frac{w}{2} \right\rfloor\right)\Im\{E(s, |w - m\pi|]\} \quad (33)$$

where $\left\lfloor \cdot \right\rfloor$ is the floor operator (rounding down), with

$$n = \left\lfloor \frac{2w}{\pi} \right\rfloor, \quad m = \left\lfloor \frac{n + 1}{2} \right\rfloor, \quad \text{bitsgn}(t) = \begin{cases} +1, & \text{if } t \equiv 0 \pmod{2} \\ -1, & \text{otherwise.} \end{cases} \quad (34)$$

where $\pmod{\cdot}$ stands for the modulo operation. Thus, “bitsgn($t$)” is a type of sign function returning a value based on the least significant bit of an integer. In practice, this means we compute $E(s, |w - m\pi|)$ once using e.g. floating point numbers, and we xor the sign bits of the real and imaginary part with bits 0 and 1 of integer $n$, respectively. The value of $\pi J_0(s)$ only has to be computed once per evaluated value of $A(r, \theta)$ and can be re-used for both evaluations of $E(s, w)$ as well as in the approximated case $E_L(s, w)$.

4. Experiments

The experiments are partitioned into two parts. First, we perform numerical experiments to compare the calculation time and accuracy of the proposed algorithm with other references. Then, we test the algorithm in an optical setup and compare visual quality differences.

4.1. Numerical calculation experiments

We utilized two line segment data sets for our experiments: (1) the “Simple shapes” data set (Fig. 3), consisting of 40 straight lines and 6 circular segments, forming simple geometric shapes placed at multiple depth planes; (2) the “Seigaiha” data set (Fig. 4) made out of 144 arc segments, which is a traditional Japanese motif of superposed wave patterns.

![Fig. 3. “Simple shapes” data set, seen from the front (a) and the side (b), color-coded according to the depth plane.](image)

We implemented multiple algorithms for computing line holograms.

- **Point-based**: this is the exact reference algorithm, where we decompose all lines into points with a period equal to the pixel pitch. We sum over all point-spread functions according to (1).

- **FFT-based**: this will subdivide the scene into $D$ depth planes depending on the data set, directly draw the lines in their respective depth planes, followed by an FFT-based
Fig. 4. Diagram of the “Seigaiha” data set, seen from the front (a) and the side (b), color-coded according to the depth plane.

convolution for computing the Fresnel diffraction. This requires \((D+1)\) FFT calculations: \(D\) for each plane and a final extra (I)FFT to obtain the hologram after adding all the Fourier transforms together.

- **CG line method**: this is the method proposed in [21], using a 1D line buffer which are drawn over the hologram plane along the curve.

- **Analytical method**: the proposed algorithm used in this paper.

The algorithms may be implemented using multi-threading on a CPU, or using massively parallel processing on a GPU. The multi-threaded CPU implementations ran on a machine with an Intel Core-i7 8850H 2.60GHz with 16GB of RAM, using either Visual Studio’s “concurrency::parallel_for” or OpenMP with maximal CPU core utilization. The GPU versions of the algorithm were run on a machine with an Intel Xeon E5-2687W v4 processor 3Ghz, 256 GB of RAM, a NVIDIA TITAN RTX GPU with CUDA 11.0, enabling CUDA compute capability 7.5. All versions of the algorithm were implemented in C++17 running on Windows 10 OS, using 32-bit floating point precision.

The calculation times for all tested algorithms applied on both data sets are summarized in Table 1. The reference point-based algorithm is embarrassingly parallel, and thus easily implemented on a GPU; it will have calculation times proportional to the number of points, which will depend in turn on the number of line segments and their lengths. This explains why the “Seigaiha” with more line elements takes about half a second, which is 3 times longer than the “Simple shapes”.

### Table 1. Run times and numerical accuracy of the different algorithms for both tested data sets.

<table>
<thead>
<tr>
<th>Algorithm version</th>
<th>Simple shapes</th>
<th></th>
<th>Seigaiha</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>time (ms)</td>
<td>PSNR (dB)</td>
<td>time (ms)</td>
<td>PSNR (dB)</td>
</tr>
<tr>
<td>Point-based (GPU)</td>
<td>157</td>
<td>(reference)</td>
<td>484</td>
<td>(reference)</td>
</tr>
<tr>
<td>FFT-based (GPU)</td>
<td>277</td>
<td>38.1</td>
<td>285</td>
<td>35.8</td>
</tr>
<tr>
<td>CG method (CPU)</td>
<td>106</td>
<td>39.4</td>
<td>159</td>
<td>36.0</td>
</tr>
<tr>
<td>Analytical method (CPU)</td>
<td>1144</td>
<td>73.7</td>
<td>14048</td>
<td>51.5</td>
</tr>
<tr>
<td>Analytical method (GPU)</td>
<td>2</td>
<td>73.7</td>
<td>21</td>
<td>51.5</td>
</tr>
</tbody>
</table>

On the other hand, the FFT-based algorithm’s time will mostly depend on the number of required FFT operations, proportional to the number of depth planes. Unlike all other tested algorithms, this one does not scale linearly with the hologram resolution.
The CG method was implemented in a multi-threaded CPU environment. Direct implementation of this algorithm on GPU is not straightforward, as the drawn line buffers are self-intersecting in the hologram plane introducing dependencies, and requiring more complex memory access patterns. A successful GPU implementation may yield further significant speedups.

The proposed analytical method can compute pixel values independently, making it highly suitable for GPU: all threads can execute the same operations with built-in trigonometric and Bessel function operations. A functional, but non-optimized implementation on CPU is also provided for reference. The calculation time is proportional to the number of line segments, yet independent of their parameters. We report calculation times of 2ms and 21ms for “Simple shapes” and “Seigaiha” respectively, which ranges from 20x to over 50x faster than the point-based CGH reference algorithm.

Furthermore, we want to quantify and compare the numerical accuracy of the different algorithms for the tested data sets. We utilize two quality metrics: the peak signal-to-noise ratio (PSNR) and the structural similarity index (SSIM). The used quality metrics require a reference signal $H$, measuring the degree by which an approximated signal $\hat{H}$ differs from the reference. The general definition of the PSNR for a real- / complex-valued holographic signal is

$$\text{PSNR}(H, \hat{H}) := 10 \log_{10} \left( \frac{\max(\|H\|)}{\|H - \hat{H}\|_2} \right)$$

where $\|\cdot\|_2$ is the Euclidean norm. The PSNR was directly applied on the complex-valued holographic signal and the results are reported in Table 1.

The SSIM uses the definition found in [26], which has been shown to significantly outperform PSNR for measuring perceptual image quality. Thus we also report both metrics on the numerically backpropagated hologram amplitude at multiple different depth planes where some of the objects are in focus. This is done for the proposed analytical CGH algorithm, the FFT-based and the CG-based CGH algorithms, using the Point-based CGH as a reference. The results for the “Simple shapes” are reported in Table 2 and for “Seigaiha” are reported in Table 3.

<table>
<thead>
<tr>
<th>depth (m)</th>
<th>Analytical PSNR</th>
<th>SSIM</th>
<th>FFT-based PSNR</th>
<th>SSIM</th>
<th>CG-based PSNR</th>
<th>SSIM</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.24</td>
<td>23.31</td>
<td>0.61</td>
<td>22.07</td>
<td>0.38</td>
<td>21.81</td>
<td>0.46</td>
</tr>
<tr>
<td>0.28</td>
<td>24.22</td>
<td>0.65</td>
<td>23.25</td>
<td>0.41</td>
<td>22.22</td>
<td>0.47</td>
</tr>
<tr>
<td>0.32</td>
<td>25.24</td>
<td>0.68</td>
<td>24.19</td>
<td>0.45</td>
<td>23.71</td>
<td>0.52</td>
</tr>
<tr>
<td>0.36</td>
<td>24.36</td>
<td>0.69</td>
<td>23.21</td>
<td>0.42</td>
<td>22.49</td>
<td>0.49</td>
</tr>
<tr>
<td>0.40</td>
<td>25.21</td>
<td>0.73</td>
<td>23.77</td>
<td>0.44</td>
<td>23.38</td>
<td>0.51</td>
</tr>
<tr>
<td>0.44</td>
<td>26.99</td>
<td>0.78</td>
<td>25.15</td>
<td>0.50</td>
<td>24.15</td>
<td>0.54</td>
</tr>
<tr>
<td>0.48</td>
<td>28.00</td>
<td>0.80</td>
<td>25.44</td>
<td>0.55</td>
<td>22.49</td>
<td>0.47</td>
</tr>
<tr>
<td>0.52</td>
<td>25.97</td>
<td>0.76</td>
<td>23.09</td>
<td>0.46</td>
<td>23.31</td>
<td>0.49</td>
</tr>
<tr>
<td>0.56</td>
<td>27.13</td>
<td>0.77</td>
<td>23.01</td>
<td>0.43</td>
<td>23.62</td>
<td>0.47</td>
</tr>
<tr>
<td>0.60</td>
<td>29.70</td>
<td>0.81</td>
<td>25.39</td>
<td>0.53</td>
<td>21.93</td>
<td>0.37</td>
</tr>
<tr>
<td>0.64</td>
<td>30.70</td>
<td>0.80</td>
<td>25.02</td>
<td>0.50</td>
<td>25.34</td>
<td>0.47</td>
</tr>
<tr>
<td>0.68</td>
<td>28.17</td>
<td>0.74</td>
<td>23.47</td>
<td>0.41</td>
<td>23.79</td>
<td>0.42</td>
</tr>
<tr>
<td>average</td>
<td>26.58</td>
<td>0.74</td>
<td>23.92</td>
<td>0.46</td>
<td>23.19</td>
<td>0.47</td>
</tr>
</tbody>
</table>
Table 3. Error measurements for the CGH rendered images from the “Seigaiha” data set, for different algorithms w.r.t. the reference Point-based CGH algorithm, numerically backpropagated at different depths.

<table>
<thead>
<tr>
<th>depth (m)</th>
<th>Analytical PSNR</th>
<th>SSIM</th>
<th>FFT-based PSNR</th>
<th>SSIM</th>
<th>CG-based PSNR</th>
<th>SSIM</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.10</td>
<td>28.22</td>
<td>0.86</td>
<td>19.16</td>
<td>0.20</td>
<td>18.09</td>
<td>0.11</td>
</tr>
<tr>
<td>0.20</td>
<td>30.40</td>
<td>0.90</td>
<td>20.42</td>
<td>0.27</td>
<td>18.30</td>
<td>0.15</td>
</tr>
<tr>
<td>0.30</td>
<td>34.97</td>
<td>0.93</td>
<td>21.38</td>
<td>0.31</td>
<td>18.53</td>
<td>0.22</td>
</tr>
<tr>
<td>0.40</td>
<td>36.72</td>
<td>0.94</td>
<td>21.43</td>
<td>0.31</td>
<td>18.47</td>
<td>0.20</td>
</tr>
<tr>
<td>0.50</td>
<td>37.13</td>
<td>0.94</td>
<td>21.39</td>
<td>0.30</td>
<td>18.36</td>
<td>0.21</td>
</tr>
<tr>
<td>0.60</td>
<td>36.96</td>
<td>0.94</td>
<td>21.43</td>
<td>0.29</td>
<td>18.08</td>
<td>0.20</td>
</tr>
<tr>
<td>0.70</td>
<td>36.66</td>
<td>0.93</td>
<td>20.84</td>
<td>0.26</td>
<td>18.14</td>
<td>0.18</td>
</tr>
<tr>
<td>0.80</td>
<td>34.30</td>
<td>0.90</td>
<td>20.35</td>
<td>0.23</td>
<td>18.49</td>
<td>0.17</td>
</tr>
<tr>
<td>average</td>
<td>34.42</td>
<td>0.92</td>
<td>20.80</td>
<td>0.27</td>
<td>18.31</td>
<td>0.18</td>
</tr>
</tbody>
</table>

Fig. 5. Graphs of the NRMSE for the CGH of a single arc line segment as a function of its radius. This was tested for both the Analytical methods and the CG line method, each placed at 0.1 m and at 0.8 m.

The proposed algorithm outperforms the CG line algorithm on average by 3.4 dB PSNR (and 0.27 SSIM units) for “Simple shapes” and by 16.1 dB PSNR (and 0.74 SSIM units) for “Seigaiha”. This gain difference is likely attributable to the stronger approximation of the CG method at line edges, which are more prominent in “Seigaiha”. The FFT-based algorithm performs slightly better than the CG line algorithm overall. However, please note that the current version of the FFT-based algorithm draws 1-pixel thick lines using nearest pixel neighbors. The quality may be improved considerably by using anti-aliasing. Moreover, note that the analytical method, unlike the other methods, requires a decomposition of curves into lines and arcs, so this may affect the resulting curve shape depending on the chosen approximation [22].

Finally, we want to confirm that the proposed analytical algorithm has consistently better quality for any arc radius, not just for specific cases. To do so we measure the approximation error using the the normalized mean-square error (NMSE)

\[
\text{NMSE}(H, \hat{H}) := \sqrt{\frac{\|H - \hat{H}\|_2}{\|H\|_2}}
\] (36)
testing the values by varying the arc radius from 160 up to 6000 pixels. The holograms were computed at 0.1 m and at 0.8 m for both the CG line and the analytical CGH versions and graphed in Fig. 5. The NMSE error curves for the analytical CGH are always strictly lower than the CG line CGH counterpart, on average by 0.04 units for the 0.1 m case and 0.15 units for the 0.8 m case.

4.2. Optical experiments

We also verify the proposed algorithms using optically reconstructed image of the tested holograms. We used a phase-modulation-type SLM (Holoeye Photonics AG, “PLUTO”) and a green laser with 532 nm wavelength (Showa Optronics, “J150GS”, Japan), cf. Figure 6.

![Diagram](image1)
![Photograph](image2)

**Fig. 6.** Images of the used holographic display setup. (a) shows a simplified diagram, (b) is an annotated photograph of the optical setup.

The numerical and optical reconstructions of the “Simple shapes” and “Seigaiha” data sets are shown in Tables 4 and 5, respectively. For “Simple shapes”, we showed cropped zoomed-in images of the holograms brought into focus at different depth planes. The numerical reconstructions look largely the same for all three methods, but there is some intensity loss at the line’s edges for the CG line method as previously observed in [21]. These optical reconstructions mirror the numerical reconstructions quite well. For “Seigaiha”, we show reconstructions of the full hologram refocused in the middle of the virtual depth volume at 0.5 m. The analytical method’s reconstructions resemble the reference Point-based CGH closer than the CG line method, whose edge darkening effect is visible just like in the other data set.

The visible noise in the optical reconstructions is due to several factors: the zero-order diffraction light, the lack of amplitude modulation in the SLM and the partial visibility of objects in other focal planes due to their long depth of fields. This effect could be mitigated by using i.a. a 4f optical system with a spatial filter, using SLMs with higher resolutions and modulation capabilities, time-multiplexing or rapidly changing the relative phase values of the different geometric shapes over time.
Table 4. Zoomed-in numerical and optical reconstructions of the “Simple shapes” data set CGH at various depths. The first three columns show numerical reconstructions, the latter three show the corresponding optical reconstructions.

<table>
<thead>
<tr>
<th>Depth</th>
<th>CG Method (numerical)</th>
<th>Analytical (numerical)</th>
<th>Reference (numerical)</th>
<th>CG Method (optical)</th>
<th>Analytical (optical)</th>
<th>Reference (optical)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.24</td>
<td><img src="image1.png" alt="Image" /></td>
<td><img src="image2.png" alt="Image" /></td>
<td><img src="image3.png" alt="Image" /></td>
<td><img src="image4.png" alt="Image" /></td>
<td><img src="image5.png" alt="Image" /></td>
<td><img src="image6.png" alt="Image" /></td>
</tr>
<tr>
<td>0.32</td>
<td><img src="image7.png" alt="Image" /></td>
<td><img src="image8.png" alt="Image" /></td>
<td><img src="image9.png" alt="Image" /></td>
<td><img src="image10.png" alt="Image" /></td>
<td><img src="image11.png" alt="Image" /></td>
<td><img src="image12.png" alt="Image" /></td>
</tr>
<tr>
<td>0.40</td>
<td><img src="image13.png" alt="Image" /></td>
<td><img src="image14.png" alt="Image" /></td>
<td><img src="image15.png" alt="Image" /></td>
<td><img src="image16.png" alt="Image" /></td>
<td><img src="image17.png" alt="Image" /></td>
<td><img src="image18.png" alt="Image" /></td>
</tr>
<tr>
<td>0.48</td>
<td><img src="image19.png" alt="Image" /></td>
<td><img src="image20.png" alt="Image" /></td>
<td><img src="image21.png" alt="Image" /></td>
<td><img src="image22.png" alt="Image" /></td>
<td><img src="image23.png" alt="Image" /></td>
<td><img src="image24.png" alt="Image" /></td>
</tr>
<tr>
<td>0.56</td>
<td><img src="image25.png" alt="Image" /></td>
<td><img src="image26.png" alt="Image" /></td>
<td><img src="image27.png" alt="Image" /></td>
<td><img src="image28.png" alt="Image" /></td>
<td><img src="image29.png" alt="Image" /></td>
<td><img src="image30.png" alt="Image" /></td>
</tr>
<tr>
<td>0.64</td>
<td><img src="image31.png" alt="Image" /></td>
<td><img src="image32.png" alt="Image" /></td>
<td><img src="image33.png" alt="Image" /></td>
<td><img src="image34.png" alt="Image" /></td>
<td><img src="image35.png" alt="Image" /></td>
<td><img src="image36.png" alt="Image" /></td>
</tr>
</tbody>
</table>

Table 5. Reconstructions of the “Seigaiha” data set CGH, refocused at 0.50m.

<table>
<thead>
<tr>
<th>Method</th>
<th>Numerical</th>
<th>Optical</th>
</tr>
</thead>
<tbody>
<tr>
<td>CG Method</td>
<td><img src="image37.png" alt="Image" /></td>
<td><img src="image38.png" alt="Image" /></td>
</tr>
<tr>
<td>Analytical</td>
<td><img src="image39.png" alt="Image" /></td>
<td><img src="image40.png" alt="Image" /></td>
</tr>
<tr>
<td>Reference</td>
<td><img src="image41.png" alt="Image" /></td>
<td><img src="image42.png" alt="Image" /></td>
</tr>
</tbody>
</table>
5. Conclusion

We propose a novel algorithm for the fast computation of CGH for line-drawn objects. We analytically derive the solution with incomplete cylindrical functions and simplify the expressions to an efficiently computable approximation suitable for GPU implementations. We report 1 to 2 orders of magnitude speedups and higher accuracy than previous solutions, and validate the rendered CGH in an holographic display setup. This algorithm enables the real-time generation of line-drawn CGH, opening the door to novel display applications in digital holography. Future work may include adding phase modulation to the line segments, different curve primitives and extensions to 3D curves.

Funding

Fonds Wetenschappelijk Onderzoek (12ZQ220N, VS07820N); Kenjiro Takayanagi Foundation; Inoue Foundation for Science; Japan Society for the Promotion of Science (20K19810).

Disclosures

The authors declare no conflicts of interest.

References