Západočeská univerzita v Plzni
Fakulta aplikovaných věd

# POČÍTAČEM GENEROVANÁ OBRAZOVÁ HOLOGRAFIE

Ing. Petr Lobaz

Disertační práce
k získání akademického titulu doktor
v oboru Informatika a výpočetní technika

Školitel: prof. Ing. Václav Skala, CSc.
Katedra: Katedra informatiky a výpočetní techniky

Plzeň 2017

University of West Bohemia
Faculty of Applied Sciences

# COMPUTER GENERATED DISPLAY HOLOGRAPHY

**Ing. Petr Lobaz**

Doctoral thesis
submitted in partial fulfillment of the requirements
for a degree of Doctor of Philosophy
in Computer Science and Engineering

Supervisor: prof. Ing. Václav Skala, CSc.
Department: Department of Computer Science and Engineering

Pilsen 2017

## Prohlášení

Prohlašuji, že jsem disertační práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne: . . . . . . . . . . . . . . . . . .                    Podpis: . . . . . . . . . . . . . . . . . . . . . . . . .

## Abstrakt

Holografie je speciální zobrazovací metoda umožňující záznam vysoce realistického 3-D obrazu. Navíc má nespočet technických aplikací, například v mikroskopii nebo nedestruktivním testování.

Holografie je dvoustupňovým procesem. Skládá se ze záznamu hologramu a jeho rekonstrukce. Digitální holografie usiluje o využití digitálních prostředků, jako jsou elektronické obrazové snímače nebo počítače, v jednom kroku nebo v obou krocích. Počítačem generovaná obrazová holografie je odvětvím digitální holografie. Jejím cílem je tvorba digitálního hologramu syntetické scény výpočetními prostředky. Může být chápána jako rozšíření počítačové grafiky, která ze syntetické scény tvoří digitální obraz.

Tato práce pojednává o jednom z nejdůležitějších problémů digitální holografie, o výpočtu šíření koherentního světla volným prostorem. Konkrétně analyzuje dvě široce používané metody výpočtu šíření světla mezi dvěma rovnoběžnými rovinami – konvoluční metodu a metodu rozkladu do úhlového spektra. Nejdůležitější část práce analyzuje jejich korektní diskretizaci. Navíc důkladně analyzuje konvoluční metodu, ukazuje její omezení a navrhuje způsoby, jak se jim vyhnout.

Práce také pojednává o vlivu zaokrouhlovacích chyb při výpočtu šíření světla. Dále navrhuje metodu založenou na vyhledávacích tabulkách, která urychluje výpočet šíření světla mezi rovnoběžnými rovinami nebo mezi množinou bodových zdrojů světla a rovinou. Tato metoda navíc značně omezuje diskutovaný vliv zaokrouhlovacích chyb. Metoda najde využití především v počítačem generované obrazové holografii.

## Abstract

Holography is a lensless imaging method capable of producing ultra-realistic 3-D images. In addition, it has many technical applications, for example in microscopy or non-destructive testing.

Holography is a two-step process. It consists of hologram recording and hologram reconstruction. Digital holography tries to utilize digital devices such as electronic sensors or computers in one or both steps. Computer generated display holography is a special branch of digital holography. It tries to make a digital hologram of a synthetic scene computationally. It can be thought as an extension of computer graphics, which tries to make a digital image of a synthetic scene.

This thesis discusses one of the most fundamental part of digital holography – calculation of coherent light propagation in free space. In particular, it analyses two widely used methods for calculation of light propagation between parallel planes – the convolution method and the angular spectrum decomposition method. The most important part of the thesis analyses their proper discretisation. In addition, it analyses the convolution method, shows its limitations and suggests a way how to overcome them.

Additionally, the thesis discusses influence of rounding errors in light propagation calculations. It also suggests a look-up table based method that accelerates calculation of light propagation between parallel planes or between a point cloud and a plane. Moreover, the method also reduces rounding errors significantly. The method is especially applicable in computer generated display holography.

## Acknowledgements

I would like to express my sincere gratitude to my advisor prof. Václav Skala. First of all, he invited me to join the project “3DTV: Integrated Three-Dimensional Television – Capture, Transmission and Display” and introduced me to holography. Second, he has been more than patient and let me slowly but steadily gain the experience.

Besides my advisor, I would like to thank prof. Ivana Kolingerová, staff of the Department of computer science and engineering, and staff of gcc for their support.

# Contents

**1 Introduction 13**

1.1 Areas of holography . . . . . . . . . . . . . . . . . . . . . . . . . 13

1.2 Objectives of the thesis . . . . . . . . . . . . . . . . . . . . . . . . 16

**2 3-D displays and holography 21**

2.1 “Holography” around us . . . . . . . . . . . . . . . . . . . . . . . . 21

2.2 Principle of holography . . . . . . . . . . . . . . . . . . . . . . . . 26

**3 Reference calculation of light propagation 35**

3.1 Preliminaries . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 35

3.2 Light propagation between parallel planes . . . . . . . . . . . . . 37

3.3 Reference calculation of light propagation between parallel planes of different sizes and sampling rates . . . . . . . . . . . . . . . . 42

**4 Filtering in light propagation calculations – the convolution method 51**

**5 Filtering in light propagation calculations – the angular spectrum method 69**

**6 Propagation calculations in a limited precision environment 97**

**7 Double look-up table for light propagation calculations 109**

**8 Conclusions and future work 125**

**9 Activities 129**

9.1 List of author’s publications . . . . . . . . . . . . . . . . . . . . . 129

9.2 Selected talks . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 131

9.3 Projects . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 132

9.4 Other activities related to holography . . . . . . . . . . . . . . . 133

**A Double lookup table method for fast light propagation calculations (conference article) 135**

**B Derivation of the DLUT method details** **141**

**C Time measurement of the DLUT method performance** **167**

**Bibliography** **177**

# Chapter 1

# Introduction

## 1.1 Areas of holography

Holography is a lensless imaging method introduced independently by Dennis Gabor [1, 2] in the USA and Yuri Nikolaevich Denisyuk [3] in the USSR. It became practical after invention of laser in early 1960's; and became a hot research topic when Emmett Norman Leith and Juris Upatnieks demonstrated first three-dimensional holographic images [4], see Figure 1.1.

Holography gained its popularity thanks to ultimate quality of three-dimensional images it produced. However, it became soon obvious that holography

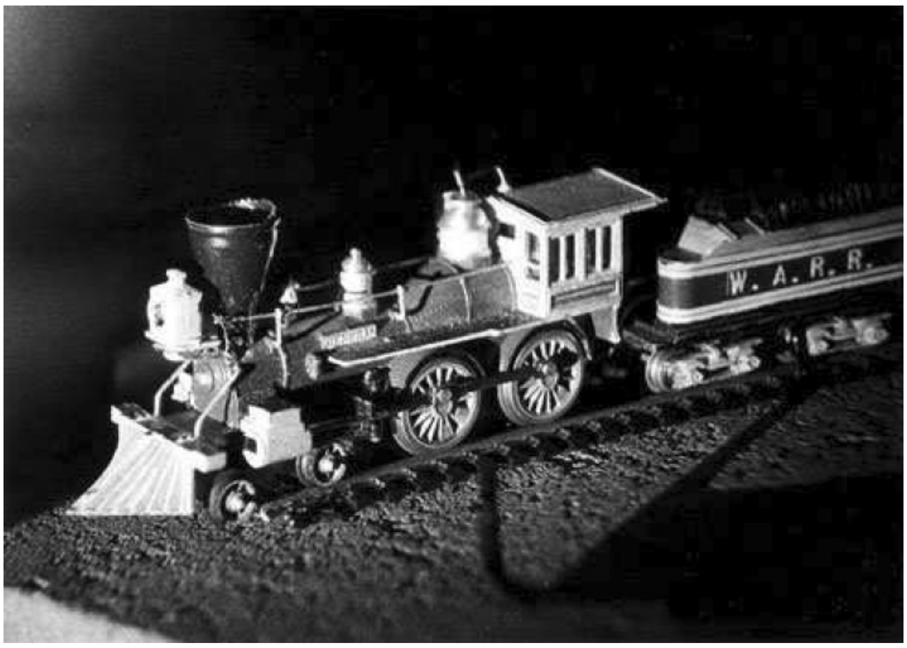


Figure 1.1: Toy Train, *Emmett N. Leith and Juris Upatnieks, March 1964, one of several variants of the first widely seen hologram (Upatnieks collection). Taken from [5, page 114].*

has many applications, not just plain imaging ones. In holographic microscopy, a hologram (i.e. a product of holography) of a specimen is made first, and the hologram is inspected under a microscope instead of the original specimen; this allows, e.g., to study short-living biologic specimens, particles in aerosol, etc. Holography can be used for imaging in presence of distorting medium, such as the Earth's atmosphere. Holographic interferometry and vibrometry allows to inspect, e.g., small movements of real objects or to inspect quality of surface shape. Thanks to high resistance of holograms to damage it is possible to use them for information storage; moreover, holograms allow to store large amount of data into a tiny piece of a recording material. It is also possible to encrypt data by optical means. Holography is closely connected to optical computing, e.g., it is possible to calculate the two-dimensional Fourier transform by optical means and to store it as a hologram. Holograms can be used as optical elements – they can emulate conventional optical elements, they can be used to correct their aberrations or can alter light in a way hardly possible by conventional optical elements. Holograms can be used as security elements thanks to their distinctive look and their comparatively difficult production. In short, holography is a general method with wide range of applications; this was acknowledged with the Nobel prize in physics to D. Gabor in 1971. For more details on applications and history of holography see, e.g., [5, 6, 7, 8].

Classical holography shares the same inconvenience with classical photography. In order to make a photograph (or a hologram), it is usually necessary to chemically process a recording medium such as a silver halide photographic film. This lead in the end of the 20th century to the biggest revolution in photography since its invention – photography became digital. This means that a photosensitive recording material (a film) in a camera was replaced by a photosensitive electronic element (a sensor) that captures incoming light. Data produced by a sensor are then processed and viewed using a computer; even making photographic prints relies on principles different from classical photographic ones.

Holography, especially in technical (non-imaging) applications, tries to undergo the same process. Digital holography (DH) also replaces a photosensitive recording material (a film) by a photosensitive electronic element (a sensor). The recorded pattern (a digital hologram, see Figure 1.2) is then computationally processed. In classical holography, it is necessary to illuminate a hologram in a special way in order to see the recorded image. In digital holography, this process is usually simulated using a computer. Calculations are performed with the digital hologram, i.e. data produced by the sensor, in order to obtain an image that would be seen as if a classical hologram was properly illuminated. Such a calculated image (called *reconstructed image*) can be then displayed on a common electronic display.

It is also possible to evaluate the reconstructed image computationally, for

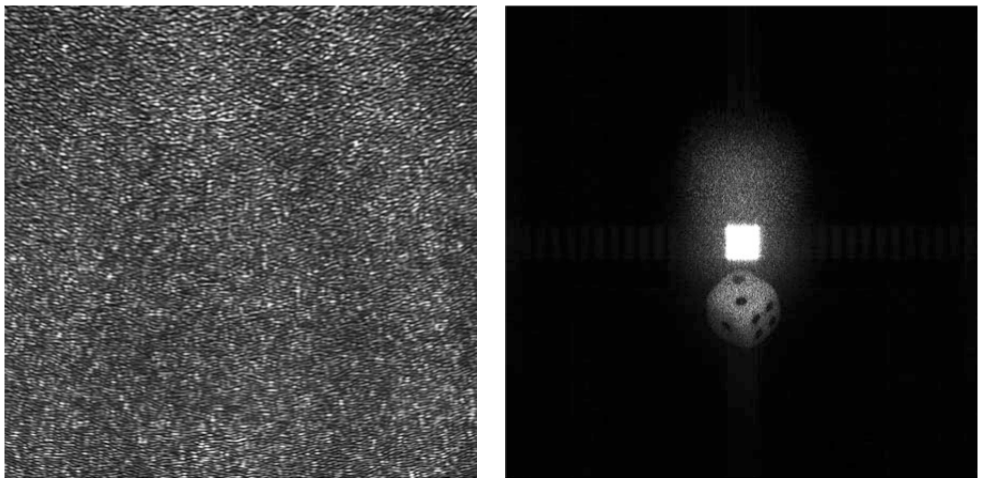


Figure 1.2: *A digital hologram (left) and its numerical reconstruction (right). Taken from [9, pages 46–47].*

example to count particles that were captured by the hologram. It is possible to process the reconstructed image, for example to denoise it or to enhance its contrast. Moreover, it is possible to evaluate and process the digital hologram itself, not just the reconstructed image. Computational processing of digital holograms then allows things impossible in classical holography (or at least very difficult). Finally, it is also possible to make a "hardcopy" of the digital hologram – to "print" it so that the hardcopy behaves in the same way as a properly captured classical hologram. For more details on digital holography and its applications see, e.g., [9, 10, 11, 12, 13].

Digital holography still requires a physical object to be recorded, a sensor, light sources, etc., just like classical holography. However, sometimes it is impossible or inconvenient to make a hologram – for example if the physical object does not exist.

The same problem also exists in common imaging – sometimes it is necessary to make an image of a thing that does not exist. There are many examples: visualisations of CAD/CAM models, drawing maps, production of motion picture special visual effects and so on. Computer graphics is often employed to synthesize such images – its input is a mathematical description, its output are image data that are in essence the same as data produced by a sensor of a digital camera. If desired, computer graphics can create photorealistic images, i.e. images indistinguishable from the images captured in the real world.

It follows that the same could be done in holography as well; we talk about computer generated holography (CGH). Its goal is to produce a digital hologram from a mathematical input (whereas computer graphics creates an image from the same input).

There are many applications of computer generated holography. For instance, it is possible to prepare a computer generated hologram that is subsequently used to test shape of an aspheric lens; computer generated hologram can be used as an optical element such as a diffuser; computer generated holograms can enhance anti-counterfeiting properties of security holograms; and so on.

Finally, it is possible to make a computer generated hologram for display purposes; then we talk about computer generated display holography (CGDH). The ultimate goal of computer generated display holography is to produce a hologram that, when properly displayed, fools an observer so that he or she has sense of reality.

Computer graphics reached this level – in specific scenarios – in the end of the 20th century. For example in motion picture production, computer graphics is used whenever it is cheaper than capturing real subjects; and it is taken for granted that the result will look real.

Classical display holography (i.e. making holograms of real objects using photosensitive recording materials) reached this level – in specific scenarios – in the beginning of the 21st century. Modern ultra-fine panchromatic recording materials are able to create a breathtaking illusion of reality, see, e.g., [14, 15] and Figure 1.3.

Computer generated display holography is very far from this level. Many algorithms that are taken for granted in computer graphics, such as shading or hidden surface elimination, still do not have "standard" counterpart in computer generated display holography. Output devices for computer graphics, such as electronic displays or printers, reached such a high level of fidelity that there is hardly any reason to improve them. Computer generated display holography still does not have any reasonably affordable and acceptable quality output devices. In short, it can be said that computer generated display holography looks like computer graphics in 1960's – some basic building blocks have been developed, but a lot of hard work is still waiting to be done.

## 1.2 Objectives of the thesis

This thesis deals with ideas and algorithms applicable in digital holography or computer generated display holography (DH / CGDH). Strictly speaking, it has no single goal, it does not offer any solution to any specific problem. Instead, it is composed of several loosely connected articles that discuss some problems that I feel were not well understood. Let me explain that.

Holography is a method of lensless image capture briefly introduced in Chapter 2. It is essentially composed of two interconnect parts – hologram capture and hologram reconstruction. DH / CGDH tries to mimic one or the other part computationally. One of the most important tasks in DH / CGDH is calculation of coherent light propagation in free space. There are many ways how to calculate it, see for example [6, 12, 16, 17]. Each method has its pros and cons; each of them works best in specific conditions and fails in another. Un-

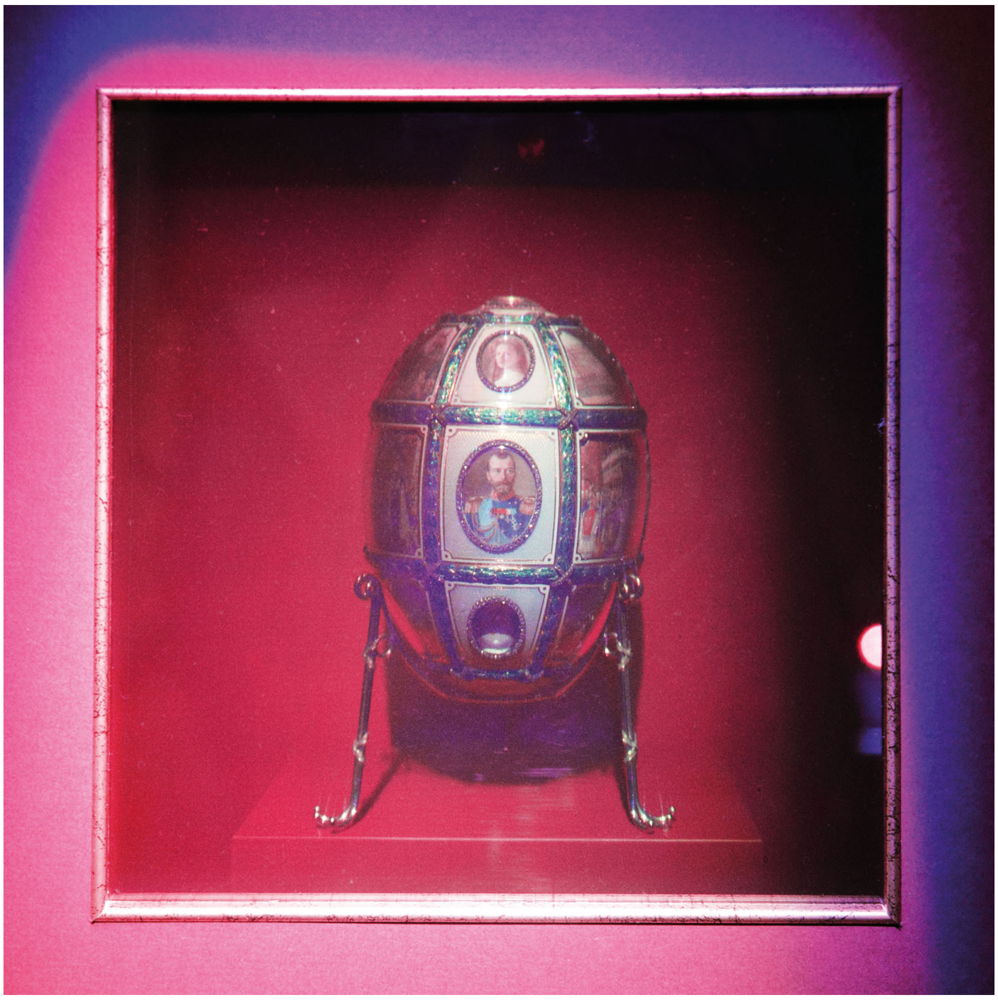


Figure 1.3: *A full colour Denisyuk hologram of the* 15th anniversary Fabergé Easter egg *by Andreas Sarakinos, Hellenic Institute of Holography, Greece, 2015. Recorded in-situ on BB-panchromatic glass plate, size approximately* $25 \times 25\ \text{cm}^2$.

fortunately, it is quite difficult to tell if a result of a calculation is correct in absence of a reference result. Thus, the thesis mostly discusses modifications of some standard methods so that their result can be then taken as "the reference result".

"Reference methods", however, tend to be very slow or memory intensive. Therefore it is often difficult to compare a result of a "fast method" with the reference result, because the reference result would take too long to calculate. Thus, I developed methods that can provide reference results within reasonable time and memory restrictions.

I restricted my research to calculations based on scalar theory of monochromatic coherent light [6, Chapter 3], because DH / CGDH usually does not require more elaborate mathematical model of light. As a reference mathematical description of light propagation in free space, I picked the Rayleigh-Sommerfeld diffraction formula of the first kind,

$$\begin{aligned} U(x,y,z_0) &= -\frac{1}{2\pi}\iint_\Omega U(\xi,\eta,0)\left(\mathbf{j}\,k-\frac{1}{r}\right)\frac{\exp(\mathbf{j}\,kr)}{r}\frac{z_0}{r}\,\mathrm{d}\xi\mathrm{d}\eta \\ r &= \sqrt{(x-\xi)^2+(y-\eta)^2+z_0^2}. \end{aligned} \tag{1.1}$$

see [18, Eq. 3.2-78] or [19, Eq. 2]. The meaning of the variables in Eq. (1.1) is not important now, the formula is shown just to get an idea how difficult is the calculation. Anyway, $U$ is a complex value describing the electromagnetic field at a specific point, $x$, $y$, $z_0$, $\xi$, $\eta$ are the spatial coordinates, $k$ is the wave number, $\Omega$ is an area in the plane $z = 0$ and $\mathbf{j}^2 = -1$. The formula calculates propagation of coherent light from the area $\Omega$ to the point $(x, y, z_0)$, see Figure 1.4.

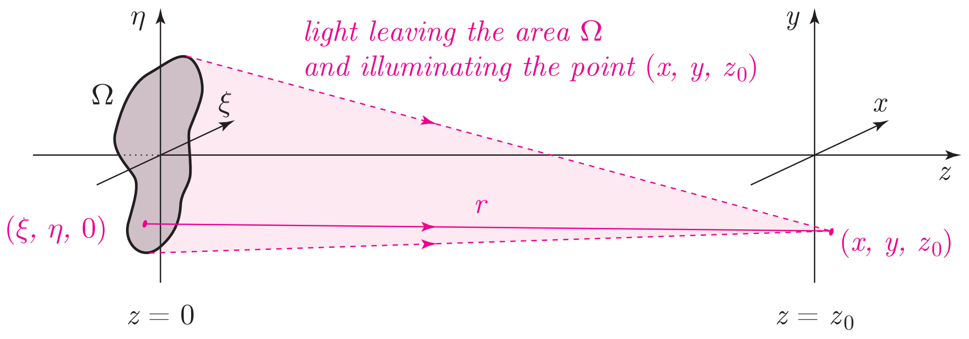


Figure 1.4: *The Rayleigh-Sommerfeld diffraction formula* (1.1) *calculates propagation of coherent light from the area* $\Omega$ *to the point* $(x, y, z_0)$.

First of all, Eq. (1.1) is directly derived from the Maxwell's equations describing monochromatic coherent light in free space [6, Chapter 3], thus it can be regarded as physically correct. Second, it is mathematically consistent, unlike for example the Fresnel-Kirchhoff diffraction formula [6, Chapter 3.5]. Third, it is mathematically equivalent to the method "angular spectrum decomposition" [19], which is used in DH / CGDH very often. Fourth, other methods of light propagation used in DH / CGDH, such as the Fresnel approximation or the Fraunhofer approximation, can be easily derived from the Rayleigh-Sommerfeld formula [6, Chapter 4]. Finally, as the Rayleigh-Sommerfeld formula is just a definite double integral in the complex domain, it can be easily calculated numerically.

The definite double integral in the Rayleigh-Sommerfeld formula can be approximated by double summation. However, as it is usually necessary to evaluate it at many points $(x, y, z_0)$, a straightforward calculation at every point would be very slow. However, Eq. (1.1) can be rewritten as a convolution and efficiently calculated using the fast Fourier transform.

"The convolution method" imposes some restrictions on the calculation. It can be often found in literature that the spatial coordinates $x$ and $\xi$ ($y$ and $\eta$, respectively) have to be discretised equally; that the area spanned by the coordinates $x$, $y$ has to be the same as the rectangular area $\Omega$ spanned by the coordinates $\xi$, $\eta$; and that the calculation requires certain amount of additional memory ("zero padding"). Sadly, actual details, such as the amount of additional memory, are given differently by different authors. Therefore, I rigorously examined the convolution method, derived its actual requirements and designed a method how to use it in various circumstances that were difficult before. The method is described in detail in Chapter 3.

There are other ways how to evaluate the Rayleigh-Sommerfeld formula than "the convolution method". As I mentioned, Eq. (1.1) is mathematically equivalent with "the angular spectrum decomposition" that expresses the formula using the Fourier transform. It is therefore tempting to use the discrete Fourier transform (especially the fast Fourier transform) to evaluate it numerically. Surprisingly, although the methods are equivalent in the continuous domain, their numerical evaluation differs a lot. The conventional wisdom says that the discrete convolution methods can be used just for long propagation distances ($z_0$ large in Eq. (1.1)), while methods based on the Fourier transform, such as the angular spectrum decomposition, should be used for short propagation distances, see for example [16]. There are some ad hoc tricks that, e.g., tweak the angular spectrum decomposition method so that it can be used for large propagation distances as well, see for example [20, 21]. However, the authors of such methods do not discuss physical meaning of the trick and its implications. Moreover, if two mathematically equivalent formulas lead to two numerical algorithms with completely different properties, it indicates hidden assumptions in the discretisation process. I closely examined the discretisation of both "the convolution method" and "the angular spectrum decomposition" and found the hidden problem. Then I created a method how to discretise both methods so that they are still equivalent and can be used for any propagation distance. The methods are described in Chapters 4 and 5. I consider them my biggest theoretical contribution to DH / CGDH.

In numerical calculations, it is necessary to use limited precision numbers, such as single precision or double precision IEEE 754 floating point numbers. The conventional wisdom says that double precision calculations are preferred in light propagation calculations. However, algorithms of computer generated display holography often utilize special hardware such as a GPU (graphics processing unit, "a graphics card") that favours single precision calculations. After all, hardware for single precision calculations is simpler and consumes less power; and in most situations, single precision calculations are good enough. I therefore examined influence of limited precision in light propagation calculation; the results are given in Chapter 6. I do not consider this result a fundamental one; on the other hand, I think it is very practical.

I have found that methods introduced in Chapters 4 and 5 tend to be slow in extreme situations. I have also found that many methods of light propagation calculation start to be unreliable for long propagation distances in a limited precision environment. Therefore, I designed look-up tables that solve both problems. The method is described in Chapter 7, additional details are given in Appendices A, B and C.

Chapter 8 concludes the thesis and describes my current and future work in computer generated display holography. Finally, Chapter 9 summarizes my activities in holography.

As the thesis is composed of several individual journal or conference articles, it does not contain any general, comprehensive “state of the art”. I have written and presented one as a part of my state doctoral examination. It is available upon request at the Department of Computer Science and Engineering, University of West Bohemia, Pilsen, Czech Republic.

# Chapter 2

# 3-D displays and holography

## 2.1 “Holography” around us

A realistic 3-D illusion is often called “a hologram”, stunning displays are often called “holographic”. However, if we search for the hologram on the Internet, the results are quite confusing, see Figure 2.1.

First of all, most found images are fakes, i.e. they are just a result of image retouching, see Figure 2.2. Not only they are not photographs of real devices – current physics just does not have tools to make such displays. Most remaining images are real photographs of devices that have nothing to do with holography. It just happened that the term “hologram” is so fancy that it is often used in

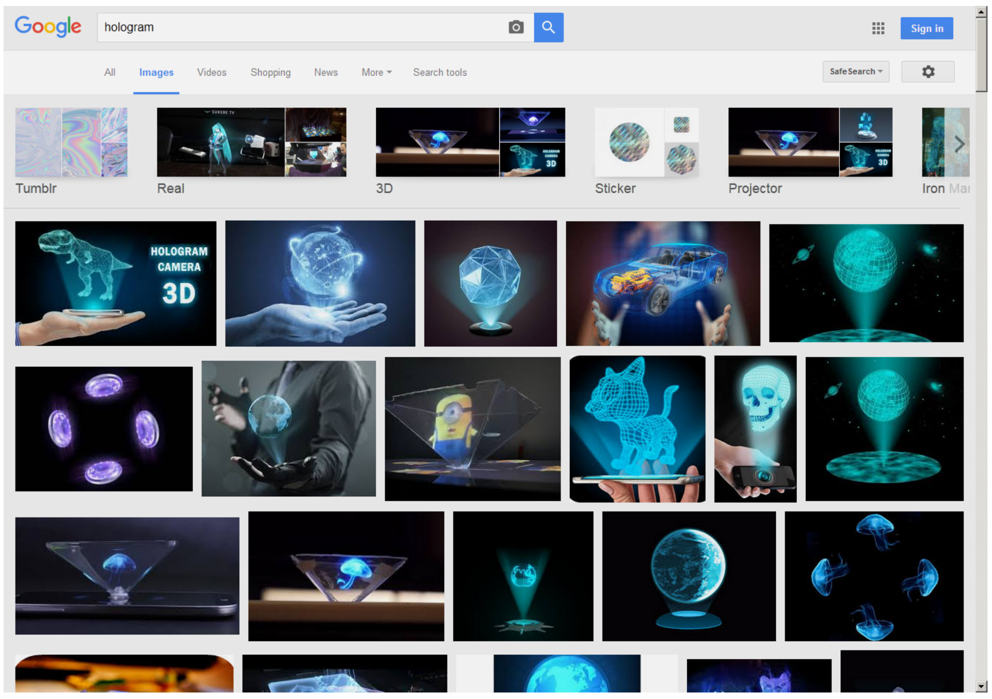


Figure 2.1: *Google search results for the word “hologram”, 2016/10/26.*

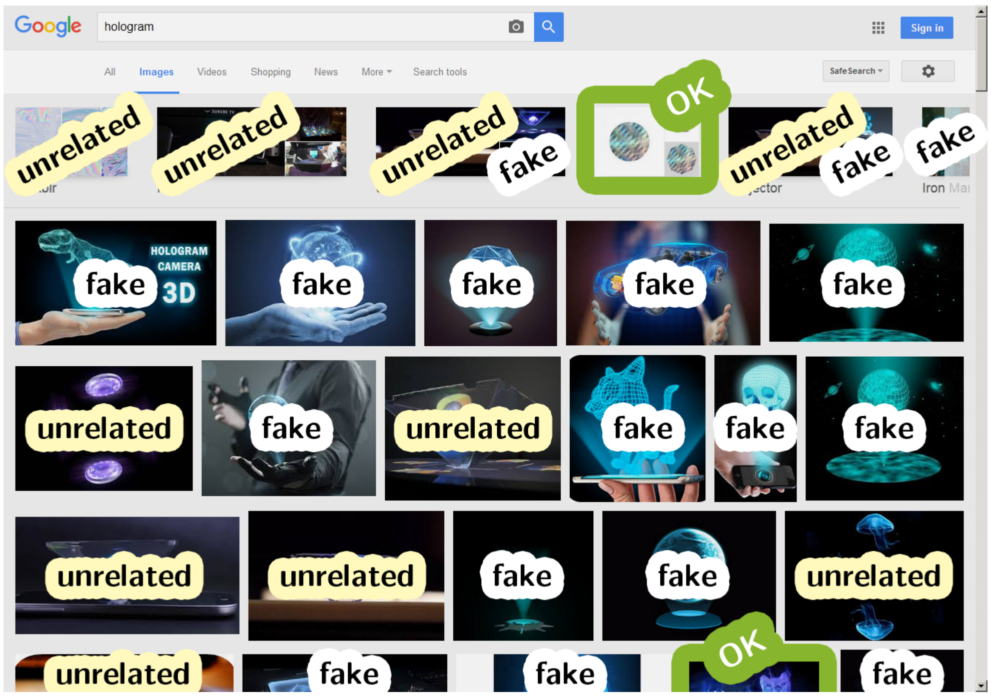


Figure 2.2: *Description of the search results from Figure 2.1. Just two results, depicted by green border, are related to holography.*

advertising new display technologies even if they utilize completely different principles.

In this particular search, just two images are related to holography. The one in the upper part of the screen is a sticker with a diffractive structure; this particular one is, strictly speaking, not a hologram too. So the only real hologram on this screen is partly displayed in the last row of images, see Figure 2.2.

The aim of this chapter is thus to clarify what is a hologram. Besides that, we will show examples of technologies often inappropriately called “holographic”. In the next section, we will introduce principle of holography.

If we ask “what is a hologram”, most people recall sci-fi movies and imagine something like “futuristic display”. However, illusions such as the one in Figure 2.3 cannot be produced with current knowledge of physics.

A lot of manufacturers advertise their products as “holographic”, for example Microsoft HoloLens. Campaigns are often accompanied by pictures such as Figure 2.4. Again, it is just a fake image. In particular, HoloLens is an augmented reality system that requires a user to wear special goggles. Images such as this one just give an impression what the user sees. Indeed, anyone without the goggles does not see the illusion.

Strictly speaking, HoloLens technology uses optical elements (waveguides) that have something to do with holography. As “hologram” sounds better than “waveguide”, the name HoloLens emerged.

Figure 2.3: *Star Wars: A New Hope (directed by G. Lucas, 1977).*

Figure 2.4: *Microsoft HoloLens: visualization of augmented reality. Taken from [22].*

Then there is a broad variety of stage illusions – the whole stadium sees that "a hologram" appeared on the stage, see Figure 2.5. Here, common technologies such as rear projection to a transparent screen, or projection to a screen and its reflection from a semi-transparent mirror are utilized. It follows that these illusions are just 2-D images that appear to be present on the stage. For details, see for example [23].

Exactly the same principle, often called (a bit imprecisely) "Pepper's ghost" [26], is utilized in small to medium sized displays that show an object seemingly floating in the air. Such displays, often in a shape of a pyramid, just reflect

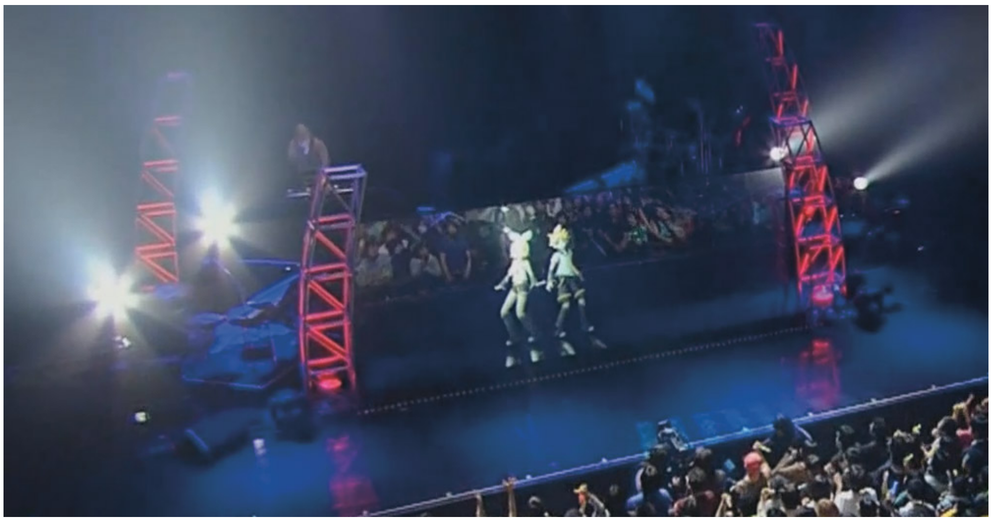


Figure 2.5: *Kagamine Rin & Len at a Hatsune Miku concert, 2010/3/9. The technology was developed by Crypton Future Media. Taken from [24].*

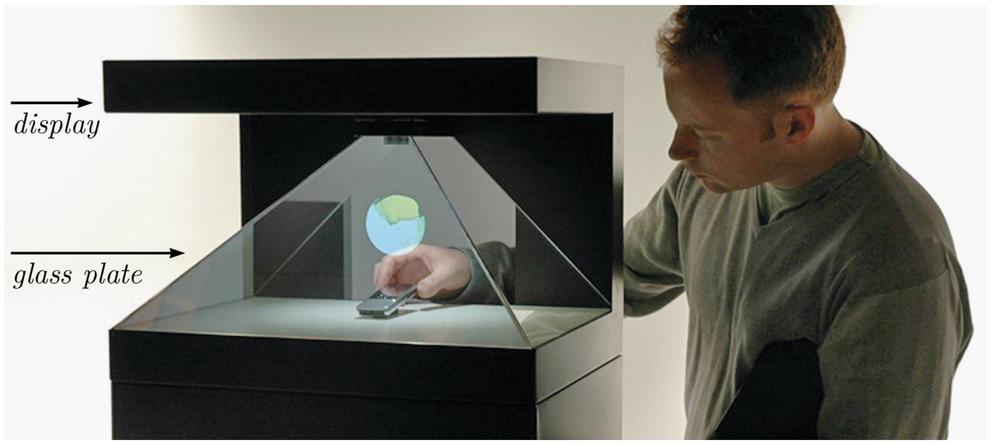


Figure 2.6: *Dreamoc$^{TM}$ HD3 display by Realfiction$^{TM}$ [25].*

an image produced by an ordinary 2-D display by a semi-transparent mirror, see Figure 2.6. However, if the object displayed rotates, changes, etc., the observer is fooled enough to believe he or she sees a perfect 3-D illusion. For details, see for example [27].

There are, of course, technologies that provide true 3-D illusion, such as the rotating display [28], see Figure 2.7. Despite that authors of such displays often do not claim they are holographic (as they are not), they are coined as such in newspapers, on TV, etc.

One of the most promising technologies for 3-D display is based on G. Lippmann's idea of "integral display", see for example [29, 30]. It is often called "light field display" as well, although light field is just a theoretical concept not related to any display technology. Similarly to a holographic display, it is a flat

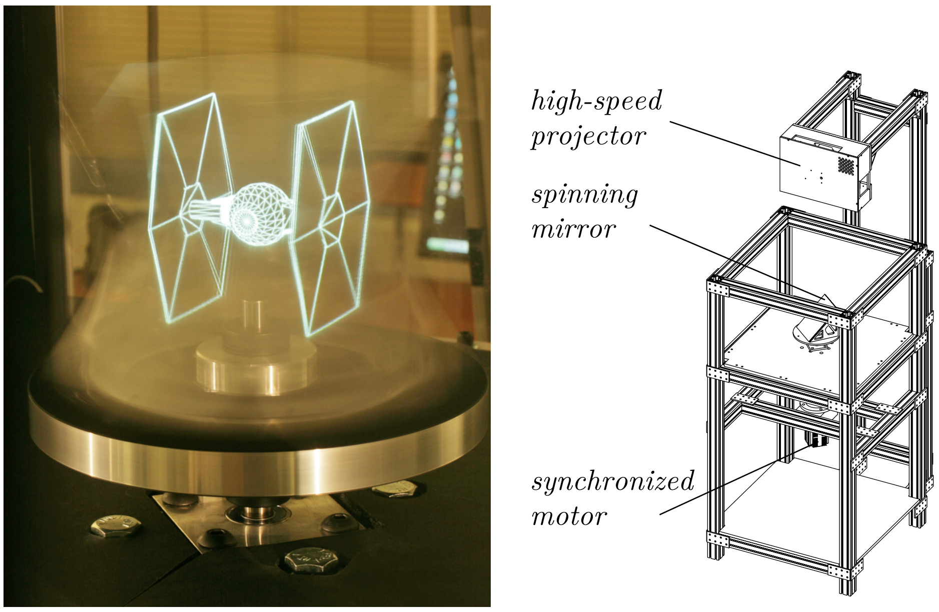


Figure 2.7: 360° *rotating 3-D display. Taken from [28].*

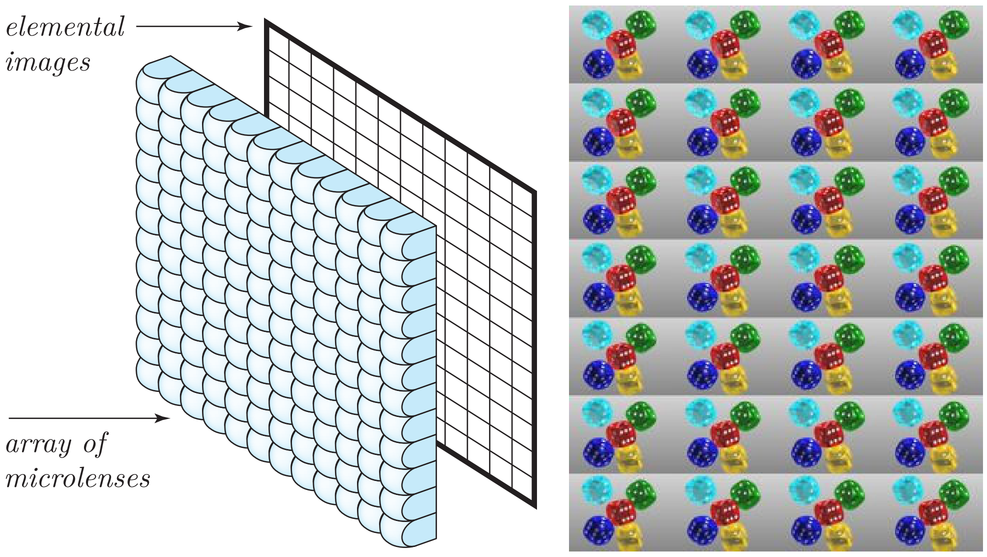


Figure 2.8: *Sketch of an integral display (left) and an example of elemental images (right). Please note that in reality, there is usually no gap between the microlenses and elemental images. The elemental images are taken from* `http://web.media.mit.edu/~gordonw/SyntheticLightFields`.

display that can provide goggles-free 3-D illusion. Contrary to a holographic display, it does not rely on wave optics principles and it seems that affordable integral displays could be built with current technology.

Basic idea of integral display is simple. A display can be thought as a window to another world. The window can be imaginarily split to many elementary areas.

Each area must provide a view to the world behind; if a viewer looks through that area, he or she “looks through a keyhole”. This is technically achieved by combination of “an elemental image” – a photograph virtually captured from the centre of the elementary area – and a microlens that virtually moves the elementary image far away and magnifies it, see Figure 2.8. An eye located close to the microlens then sees the same image as “looking through a keyhole”. A viewer located far away from the array of elemental image-microlens pairs then sees through many keyholes at once, exactly as if he or she looks through an ordinary window.

Finally, we should not forget attempts to display a 3-D illusion that really floats in space, can be observed from anywhere and are closest to “sci-fi holograms”. There are several attempts how to make such an illusion; one of them uses lasers to make points in the air glow, see Figure 2.9 or for example [31]. However, as the glowing points can be seen from anywhere, such displays cannot show opaque objects as every surface is always visible.

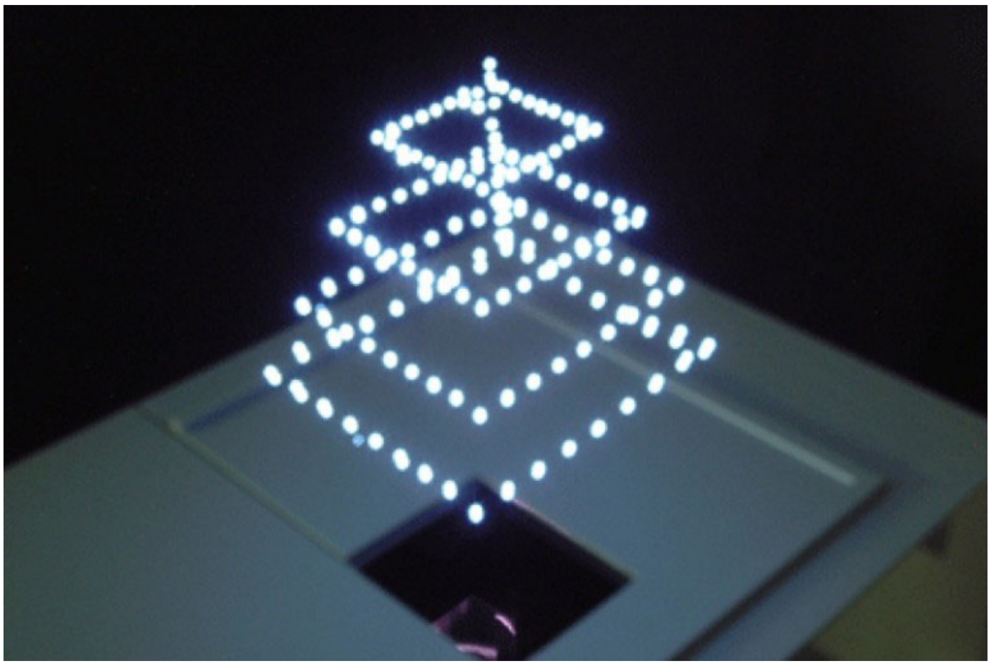


Figure 2.9: *Burton Inc. 3-D plasma volumetric display, 2011/11/15. Taken from [32]*

## 2.2 Principle of holography

Let us think for a while how to provide a perfect illusion, visually indistinguishable from the real world.

Recall that our eyes respond just to light that enters their pupils. Thus, in order to see an object, some light must be reflected off its surface, and some

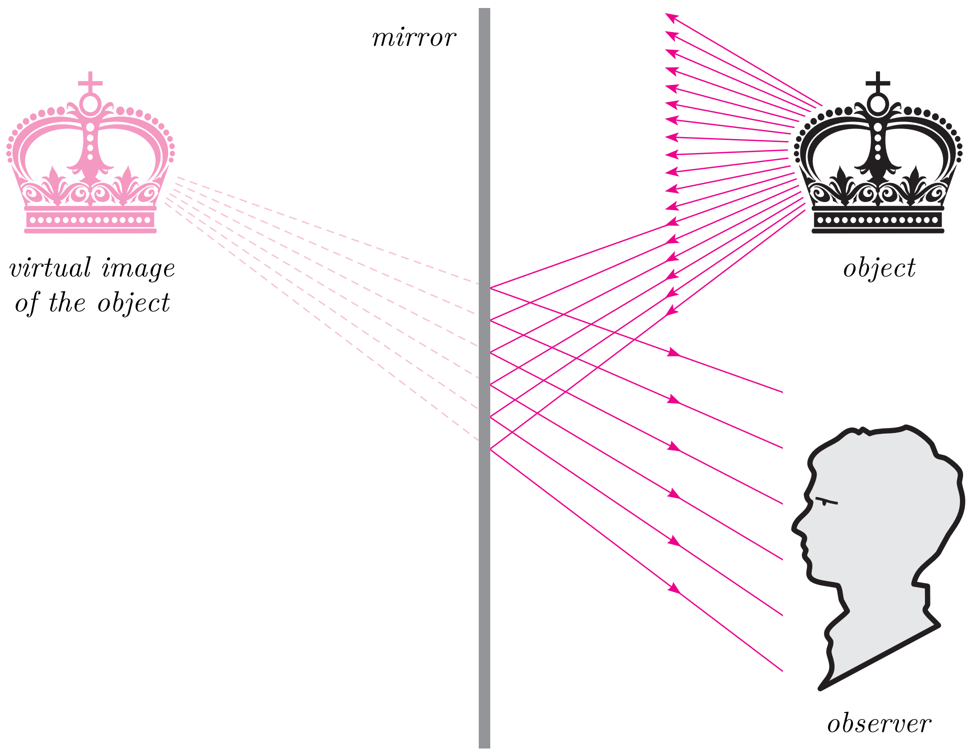


Figure 2.10: *An observer sees an object thanks to rays of light leaving the object and entering the observer's eyes. Apparent position of the object is given by light rays, not the object itself.*

light rays must find their way to the retina.

A light ray need not travel in a straight line – for example a mirror changes its direction abruptly. A mirror does not bend light rays arbitrarily. It bends them so that they appear to originate from the "mirror image" of their source. Thus, an observer looking towards the mirror sees the original object behind the mirror surface, see Figure 2.10. Strictly speaking, the observer sees the virtual image of the original object. If the mirror is perfect, it is hard to tell if we are looking at the original object or its virtual image. Anyone who ever visited a mirror maze can confirm that.

In order to create a perfect illusion, it is necessary to "freeze light" somehow – to make a hypothetical display, a surface that emits exactly those light rays that were leaving the mirror in Figure 2.10. As the light rays are exactly the same, the observer cannot tell if he or she watches the original object or its virtual image.

The points of the hypothetical display could emit light themselves such as CRT or OLED displays. Or, such as in LCD displays, some backlight could be provided. In this case, the task of the hypothetical display is to bend light rays from the light source so that light rays produced are the same as the light rays

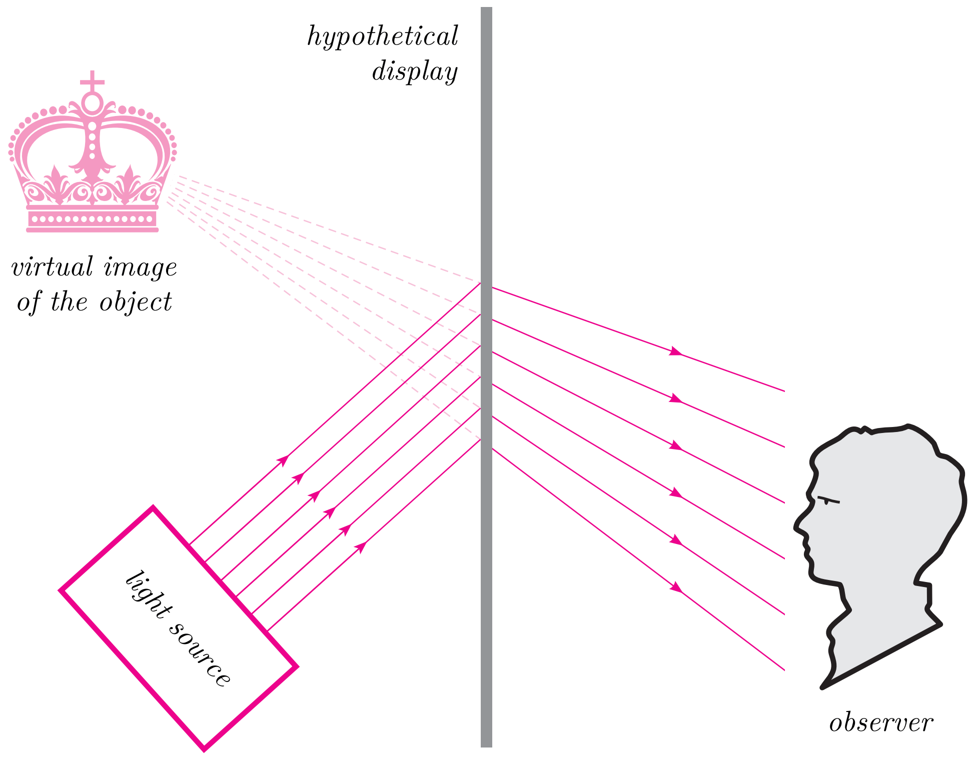


Figure 2.11: *An observer watching "a hypothetical display" sees a perfect illusion, such as when watching the object in a mirror.*

formed by the original object, see Figure 2.11.

In order to make such a hypothetical display, two questions have to be answered: how to capture complete information about light rays leaving the object, and how to replicate them. Both questions were first answered by D. Gabor in his seminal papers [1, 2], where he proposed a new method for lensless image capture. He called the image formed by the process *the hologram* and the process *the wavefront reconstruction*; the process was later renamed to *holography*.

Recall that the hypothetical display has to bend rays coming from the backlight. Light diffraction does exactly this. A light ray passing through a fine locally periodic structure of stripes splits to several new rays called diffracted rays, see Figure 2.12. Their direction depends on stripes' distance $d$, wavelength of light $\lambda_2$ (the presence of the subscript will be useful later) and direction of the original ray. If a structure, called a diffractive structure, forms several rays at once (it depends on stripes' properties), they are numbered by integer $m$, and they are called diffraction orders. Diffraction order $m = 0$ is just the directly transmitted ray. In particular, angles $\theta_{in}$ of the incoming ray and $\theta_{out,\,m}$ of the $m$-th diffracted ray are related in *the grating equation*, see for example [7]:

$$\sin\theta_{out,\,m} = m\frac{\lambda_2}{d} + \sin\theta_{in}. \tag{2.1}$$

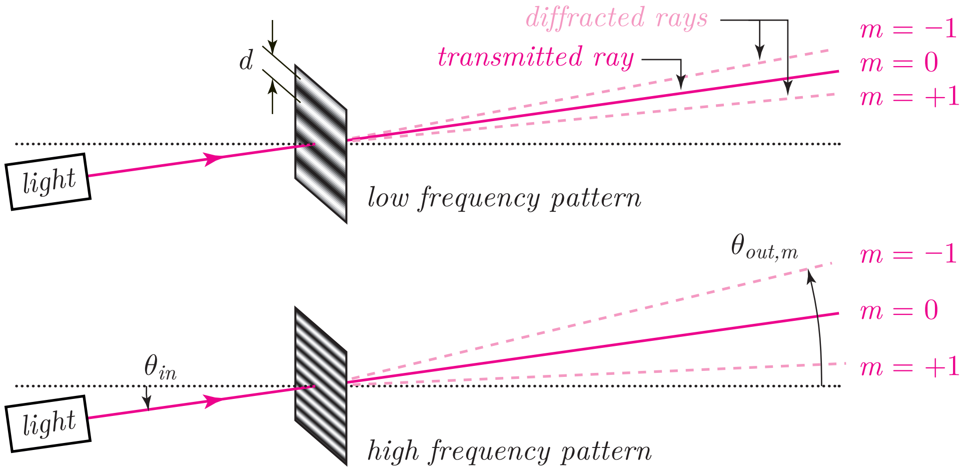


Figure 2.12: *Diffraction of light ray at different gratings.*

Please note that all rays lie in a single plane. For simplicity, we assume that the stripes of the diffractive structure are perpendicular to the incoming ray.

For now, let us just remember the most important fact: fine stripes bend a ray a lot, rough stripes bend a ray a little.

Thus, to make an illusion of a point floating behind the display surface, the diffraction pattern has to vary its properties. Somewhere it is necessary to bend the backlight a lot, thus the structure has to be fine. Somewhere else the backlight rays are almost in the right direction, therefore the structure can be rough, see Figure 2.13. It is easy to see that the structure has to look like concentric circles and their density grows from their common centre.

Now it is time to substitute some real-world numbers to the grating equation (2.1). Wavelength of visible light is between 0.4 and 0.7 $\mu$m; let us take

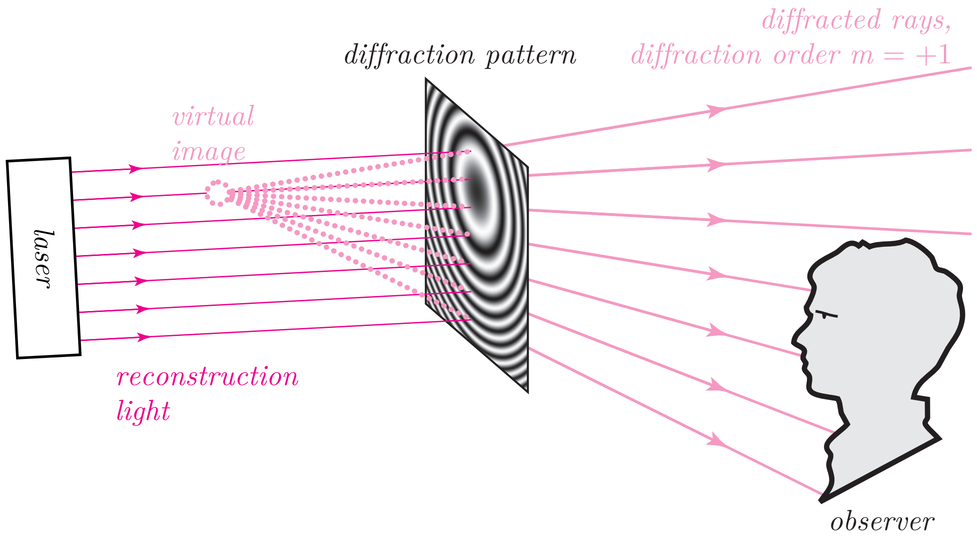


Figure 2.13: *Diffraction at a specific structure creates an illusion of a point.*

$\lambda_2 = 0.5\ \mu$m (green-cyan) as an example. If the stripes are just 10 $\mu$m apart (that is, one cycle from opaque through transparent to opaque is 10 $\mu$m), then the ray deflection is only 2.87° for diffraction order $m = 1$ (we are usually interested in this diffraction order). This is not too impressive – when watching a common display, the field of view is about 30°. Such a big deflection angle would require stripe width about 1 $\mu$m.

Besides the original two questions, a new one appears: how to fabricate such a fine pattern? Luckily, there is a phenomenon that gives an answer: light interference. If two lights are mutually coherent (for example coherence of laser light is quite good), they interfere and create a pattern of bright and dark stripes that can be recorded by a fine photographic film. For example, let us imagine a wide beam of laser light illuminating a screen (or a photographic film) and a dust particle in between. Light (we call it *reference light*) scatters on the dust particle and the particle starts to behave as a point light source. The screen is thus illuminated by two light sources – from the laser unit and from the dust particle. These two lights interfere and create exactly the pattern we want, see Figure 2.14.

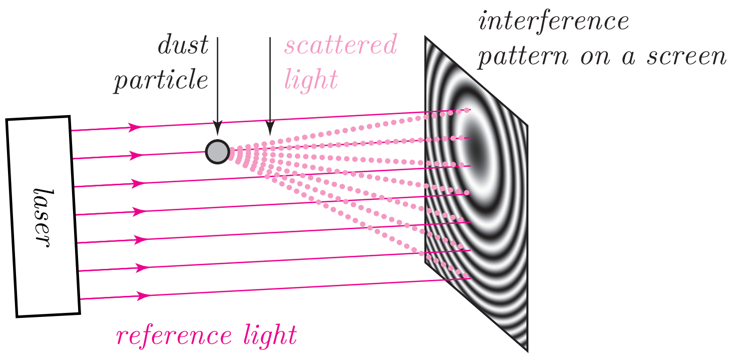


Figure 2.14: *Laser light illuminates a dust particle. The scattered light interferes with the laser light and creates an interference pattern on a screen.*

To explain why, we need to understand light interference quantitatively. If two "coherent" light beams illuminate a screen, they form an interference pattern that locally looks like a set of straight stripes. Their width $d$ depends on angles of the forming light beams and their common wavelength $\lambda_1$ (the presence of the subscript will be useful in a while). It is described by *the interference equation* (2.2), see also Figure 2.15. Please note that for simplicity, we assume that both beams lie in a single plane. In this case, the stripes of the interference pattern are perpendicular to that plane.

$$d = \frac{\lambda_1}{\sin\theta_A - \sin\theta_B}. \tag{2.2}$$

For example, if two green-cyan ($\lambda_1 = 0.5\ \mu$m) mutually coherent beams meet at angles $\theta_A = 45°$, $\theta_B = -45°$, they form the interference pattern where stripes

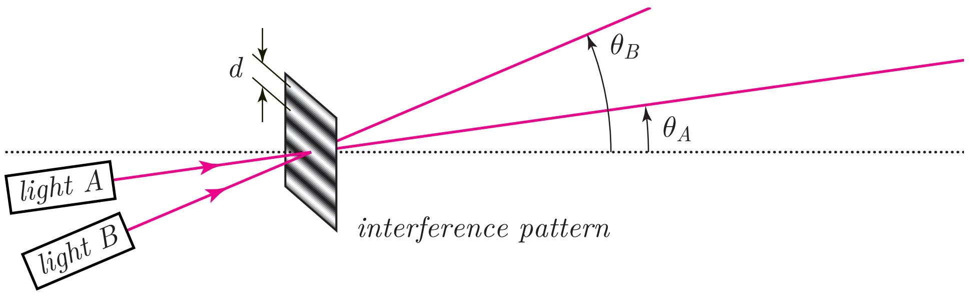


Figure 2.15: *Interference of two mutually coherent light beams.*

are $d = 0.35$ $\mu$m apart.

It it worth mentioning that it is quite tricky to define coherence. For now, let us just say that if lights interfere, they are coherent, and vice versa. This is not a very useful definition, indeed, but we will not need rigorous theory of coherence for our purposes.

Now it is straightforward to combine principles of interference and diffraction. We know that two lights $A$, $B$ of wavelength $\lambda_1$ interfere and make a pattern of certain density described by the interference equation (2.2). We also know that light "*in*" of wavelength $\lambda_2$ passing through a pattern of certain density is diffracted; this is described by the grating equation (2.1). We can combine both equations together and get *the sine-theta equation* (the name is the same as in [7]):

$$\sin\theta_{out,\,m} = m\frac{\lambda_2}{\lambda_1}\big(\sin\theta_A - \sin\theta_B\big) + \sin\theta_{in}. \tag{2.3}$$

Again, for simplicity we assume that all light rays involved in the equation lie in a single plane.

Now, imagine a simple experiment. Let us illuminate a photographic film by two beams of the wavelength $\lambda_2 = \lambda$. Let their angles of incidence be $\theta_A$ and $\theta_B$. An interference pattern is created and recorded by the film, see again Figure 2.15. Then, let us illuminate the film by a light ray of the same wavelength $\lambda_1 = \lambda$ at an angle $\theta_B$, i.e. $\theta_{in} = \theta_B$. Substitution to the sine-theta equation reveals that the first order diffracted ray ($m = 1$) leaves the film at an angle $\theta_A$, see Figure 2.16. That is, the pattern encoded the ray direction somehow, and we are able to reconstruct it.

It is now very easy to describe holography; in fact, we have described it right now. Let us call light leaving an object as *the object wave* (the term wave is used because interference and diffraction rely on wave nature of light). Let us put a photographic film somewhere so that it is illuminated by the object wave. Let us illuminate the film with additional light called *the reference wave.* If the reference wave and the object wave are coherent, the interference pattern is formed are recorded by the film. This recording is called *the hologram.*

The hologram is subsequently illuminated by *an illumination wave.* If the

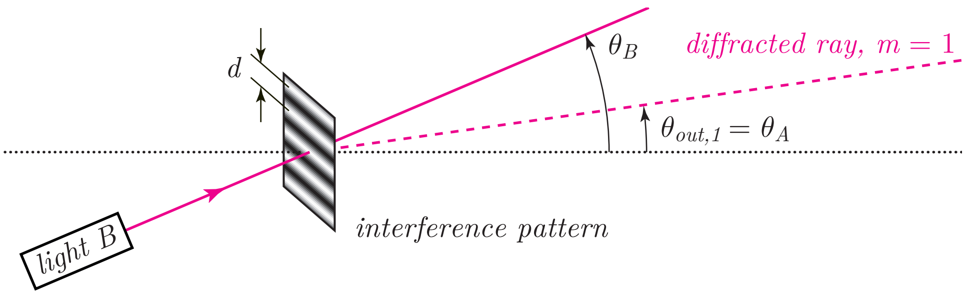


Figure 2.16: *Diffraction of light B at the structure recorded in Figure 2.15.*

illumination wave is a replica of the reference wave, diffracted light perfectly reconstructs the object wave. Thus, holography is sometimes called *the wavefront reconstruction.*

To see why holography works, let us examine a simple case. Let the object wave be light leaving a point in the space. Also for simplicity, let us assume that reference rays are mutually parallel. Both lights share the same wavelength $\lambda_{ref}$. We can imagine that the hologram is split to tiny elementary areas so that each area is illuminated by a single ray from the object and a single reference ray. Each elementary area thus records a simple interference pattern. Its density changes across the hologram area because the angle between the object and the reference ray varies slightly, see Figure 2.17a. Indeed, the angles of rays should be called $\theta_{obj}$ and $\theta_{ref}$ now instead of $\theta_A$ and $\theta_B$ in the sine-theta equation (2.3).

Subsequently, we illuminate the hologram by the illumination wave of wavelength $\lambda_{ill}$; in the sine-theta equation, we should rename $\theta_{in}$ to $\theta_{ill}$ to make it

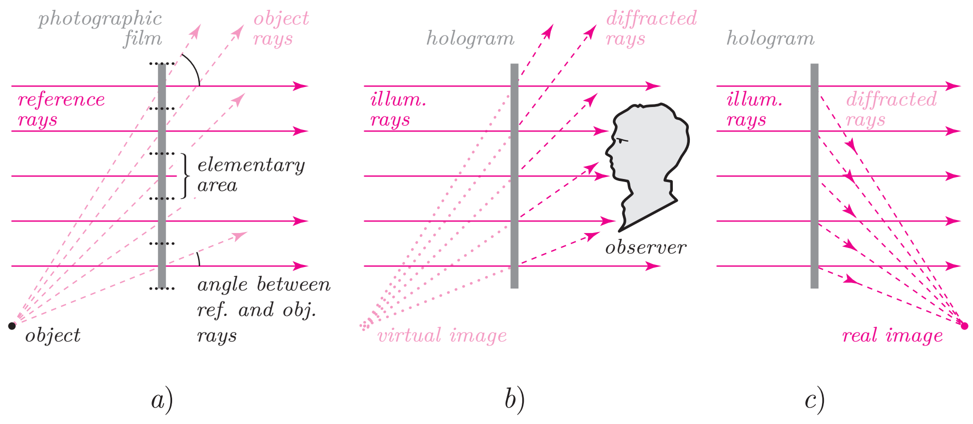


Figure 2.17: *Principle of holography. a) hologram recording, b) rays of order $m = 1$ diffracted by the hologram form the virtual image, b) in this case, rays of order $m = -1$ form the real image.*

clear. Thus, the sine-theta equation, after renaming the symbols, becomes

$$\sin\theta_{out,\,m} = m\frac{\lambda_{ill}}{\lambda_{ref}}\big(\sin\theta_{obj} - \sin\theta_{ref}\big) + \sin\theta_{ill}. \tag{2.4}$$

If the illumination wave is a replica of the reference wave ($\theta_{ref} = \theta_{ill}$, $\lambda_{ref} = \lambda_{ill}$), the object wave is perfectly reconstructed for diffraction order $m = 1$, as

$$\sin\theta_{out,\,1} = \frac{\lambda_{ref}}{\lambda_{ref}}\big(\sin\theta_{obj} - \sin\theta_{ref}\big) + \sin\theta_{ref} = \sin\theta_{obj}.$$

That is, if an observer looks towards the hologram, he or she sees the virtual image of the original point, see Figure 2.17b. As a general object can be decomposed to many point light sources, the same would apply for a general object.

We can also notice an additional detail. If we use the reference light at normal incidence to the hologram ($\theta_{ref} = \theta_{ill} = 0$) and substitute $m = -1$ to the sine-theta equation, we get $\theta_{out,\,-1} = -\theta_{obj}$. That is, if we place a sheet of paper to a certain distance from the hologram, a bright point appears. At this place, a real image of the original point was formed, see Figure 2.17c. This means that holography is able to make an illusion both in front or behind the hologram surface.

Let us conclude this short introduction. In order to watch a hologram, it is necessary to have a diffractive structure and to illuminate it in a proper way. The diffractive structure (the hologram) can be made by illuminating a photographic film by light reflected off an object and auxiliary (reference) light.

Computer generated display holography tries to make the hologram computationally, for example by simulating the hologram recording process. Digital holography tries to reconstruct images encoded by the hologram, for example by simulating the hologram illumination process. In either case, it is absolutely necessary to understand propagation of light including phenomena such as interference or diffraction.

# Chapter 3

# Reference calculation of light propagation

The article *Reference calculation of light propagation between parallel planes of different sizes and sampling rates* [33] published in Optics Express, included in this chapter from page 43, discusses enhancements of “the convolution method” of light propagation calculation. To make the article shorter, it was necessary to omit some details, especially in its first sections. As this chapter starts the technical part of the thesis, it is convenient to include them right here, before the article itself.

## 3.1 Preliminaries

Light can be described as the electromagnetic field. The idea of electromagnetic field declares that each point $\mathbf{x}$ in the space possesses two time-dependent vectorial properties: the electric field $\mathbf{E}(\mathbf{x}, t)$ and the magnetic field $\mathbf{H}(\mathbf{x}, t)$. These fields are not independent; their dependency is described by the Maxwell’s equations.

The Maxwell’s equations is a set of four vectorial differential equations. Their solution can be very difficult; therefore, simplifications are usually employed. In digital holography / computer generated display holography, we usually assume simple free space, i.e. there is no free electric charge, there is no electric current or external magnetic force and the environment is linear, isotropic, homogeneous, nondispersive and nonmagnetic.

In such environment, it can be found [6, Section 3.2] that each component $E_x$, $E_y$, $E_z$ of the electric field $\mathbf{E}$ (we assume the Cartesian coordinate system) and $H_x$, $H_y$, $H_z$ of the magnetic field $\mathbf{H}$ must satisfy the same differential equation

$$\frac{\partial^2}{\partial x^2}u(\mathbf{x}, t) + \frac{\partial^2}{\partial y^2}u(\mathbf{x}, t) + \frac{\partial^2}{\partial z^2}u(\mathbf{x}, t) = \frac{n^2}{c^2}\frac{\partial^2}{\partial t^2}u(\mathbf{x}, t)$$

(the wave equation), where $t$ is time, $c$ is the velocity of light in vacuum, $n$ is the refractive index of the medium and $u$ stands for any of $E_x$, $E_y$, $E_z$, $H_x$, $H_y$, $H_z$. We usually solve the wave equation for the symbol $u$ and do not care what physical quantity it stands for. This simplification is called *the scalar theory of light*

There are many solutions of the wave equation. The solution called *the monochromatic wave* is

$$u(\mathbf{x}, t) = A(\mathbf{x}) \cos\left[\phi(\mathbf{x}) - 2\pi\nu t\right],$$

where $A(\mathbf{x})$ and $\phi(\mathbf{x})$ are the amplitude and phase at a point $\mathbf{x}$ and $\nu$ is the time frequency of the field oscillation. Now, the problem is to find functions $A$ and $\phi$.

The monochromatic wave can be compactly described as

$$u(\mathbf{x}, t) = \Re\left[U(\mathbf{x}) \exp(-\mathbf{j}\, 2\pi\nu t)\right], \tag{3.1}$$

where $\mathbf{j}^2 = -1$, $\Re(\cdot)$ is the real part of a complex number and the complex function $U(\mathbf{x}) = A(\mathbf{x}) \exp[\mathbf{j}\, \phi(\mathbf{x})]$ is called *the complex amplitude* or *the phasor.*

Substitution to the wave equation reveals that the complex amplitude $U$ must satisfy *the Helmholtz equation*

$$\frac{\partial^2}{\partial x^2} U(\mathbf{x}) + \frac{\partial^2}{\partial y^2} U(\mathbf{x}) + \frac{\partial^2}{\partial z^2} U(\mathbf{x}) + k^2 U(\mathbf{x}) = 0,$$

where

$$k = 2\pi \frac{\nu}{c/n} = \frac{2\pi}{\lambda}.$$

The number $k$ is called *the wave number*, $\lambda = c/(n\nu)$ is *the wavelength.*

In DH / CGDH, we are usually interested in perfect monochromatic waves of prescribed wavelength $\lambda$. Thus, we are looking for the phasor $U$ from the Helmholtz equation. Please note that the phasor has no direct physical meaning. To get it, it is necessary to multiply it with the time-varying term and extract the real part, see Eq. (3.1). However, the value of $|U|^2$ is equal, up to a multiplicative factor, to the light intensity, see [6, Section 4.1.1].

The Rayleigh hypothesis (see for example [34]) states that once $U(\mathbf{x})$ is known in a single plane, say $z = 0$, its value is given in the whole space. In fact, the same statement is given by the Rayleigh-Sommerfeld diffraction formula of the first kind (1.1), repeated here for convenience:

$$U(x, y, z_0) = -\frac{1}{2\pi} \iint_{\Omega} U(\xi, \eta, 0) \left(\mathbf{j}\, k - \frac{1}{r}\right) \frac{\exp(\mathbf{j}\, kr)}{r} \frac{z_0}{r} \,\mathrm{d}\xi \mathrm{d}\eta$$

$$r = \sqrt{(x - \xi)^2 + (y - \eta)^2 + z_0^2}.$$

We assume that the phasor $U(\xi, \eta, 0)$ is zero outside the area $\Omega$ lying in the plane $z = 0$ and that $z_0 > 0$. If we define $U(\xi, \eta, 0) = 0$ outside $\Omega$, we can rewrite the formula to

$$\begin{aligned} U(x,y,z_0) &= -\frac{1}{2\pi}\iint_{-\infty}^{\infty} U(\xi,\eta,0)\left(\mathbf{j}\,k-\frac{1}{r}\right)\frac{\exp(\mathbf{j}\,kr)}{r}\frac{z_0}{r}\,\mathrm{d}\xi\mathrm{d}\eta \\ r &= \sqrt{(x-\xi)^2+(y-\eta)^2+z_0^2}. \end{aligned} \tag{3.2}$$

If we define the Fourier transform of a function $g(x,y)$ as

$$\mathcal{F}\{g(x,y)\} = G(f_x,f_y) = \iint_{-\infty}^{\infty} g(x,y)\exp\left[-\mathbf{j}\,2\pi(xf_x+yf_y)\right]\mathrm{d}x\mathrm{d}y,$$

where $f_x$ and $f_y$ are *the spatial frequencies*, and its inverse as

$$\mathcal{F}^{-1}\{G(f_x,f_y)\} = g(x,y) = \iint_{-\infty}^{\infty} G(f_x,f_y)\exp\left[\mathbf{j}\,2\pi(xf_x+yf_y)\right]\mathrm{d}f_x\mathrm{d}f_y,$$

it is possible to rewrite (3.2) to

$$U(x,y,z_0) = \mathcal{F}^{-1}\Big\{\mathcal{F}\{U(x,y,0)\}\exp\Big(\mathbf{j}\,kz_0\sqrt{1-\lambda^2f_x^2-\lambda^2f_y^2}\Big)\Big\}, \tag{3.3}$$

which is valid for every $z_0$, see [19] for the proof of equivalence.

Equation (3.2) leads to *the convolution method* of light propagation calculation, equation (3.3) leads to *the angular spectrum decomposition* and others.

## 3.2 Light propagation between parallel planes

Basic task of digital holography is a digital hologram reconstruction. This, for example, means simulation of the hologram illumination by coherent light and subsequent simulation of light propagation to a virtual screen. Virtual screen captures, indeed, the real image produced by the hologram.

First part of the process is easy. A digital hologram is usually given as an ordinary digital image, i.e., it is given as a 2-D signal (array of samples) $h[m,n]$ of the function $h(x,y)$,

$$h[m,n] = h(m\Delta_x, n\Delta_y), \quad m,n\in\mathbb{Z}, \quad \Delta_x>0,\ \Delta_y>0.$$

The function $h(x,y)$ usually represents transmittance (real or complex) of the hologram at a particular point. Any light passing through the hologram is then locally attenuated, which is mathematically expressed as multiplication. Specifically, if the hologram lying in the plane $z=0$ is illuminated from behind (from "negative $z$ half-space") with light described by the phasor $L(x,y,z)$, then light phasor just after passing the hologram is given as

$$U(x,y,0) = L(x,y,0)h(x,y).$$

If the planes of the hologram and the screen are parallel to each other, it is possible to use either Equation (3.2) or (3.3) to calculate the light phasor in the plane $z=z_0$.

As as explained in Section 1.2, it is useful to have a realiable reference algorithm that calculates light propagation numerically. It is wiser to prefer the Rayleigh-Sommerfeld formula (3.2) over the angular spectrum decomposition (3.3), as the latter one requires more elaborate mathematics. While it is possible to approximate the Fourier transform by the discrete Fourier transform, the consequences of the approximation are not obvious. Specifically, if we change the continuous transforms in (3.3) to the discrete approximations, we get some result. If we were able to calculate the result in the continuous domain and discretise (sample) the result, it is not clear if it was the same as the result of the discrete calculation.

The Rayleigh-Sommerfeld diffraction formula (3.2) can be rewritten using convolution. Recall that convolution of functions $f$, $g$ is the function $c$

$$c(x) = f(x) \otimes g(x) = \int_{-\infty}^{\infty} f(\xi) g(x-\xi) \mathrm{d}\xi,$$

or in two dimensions

$$c(x,y) = f(x,y) \otimes g(x,y) = \int_{-\infty}^{\infty} f(\xi,\eta) g(x-\xi, y-\eta) \mathrm{d}\xi \mathrm{d}\eta.$$

Thus, Equation (3.2) leads to

$$U(x,y,z_0) = U(x,y,0) \otimes K_{RS}(x,y,z_0),$$

where

$$\begin{aligned} K_{RS}(x,y,z_0) &= -\frac{1}{2\pi}\left(\mathbf{j}\,k - \frac{1}{r}\right)\frac{\exp(\mathbf{j}\,kr)}{r}\frac{z_0}{r} \\ r &= \sqrt{x^2+y^2+z_0^2}. \end{aligned} \tag{3.4}$$

Convolution has a very useful property – it can be calculated using the Fourier transform:

$$c(x) = f(x) \otimes g(x) = \mathcal{F}^{-1}\Big\{\mathcal{F}\{f(x)\}\mathcal{F}\{g(x)\}\Big\},$$

and similarly in higher dimensions. We will utilize it later.

In the discrete domain, we can define *the discrete convolution*:

$$c[\,] = f[\,] \otimes_d g[\,],$$

where

$$c[n] = \sum_{m=-\infty}^{\infty} f[m] g[n-m],$$

and similarly in higher dimensions. Here, signals $f[\,]$, $g[\,]$ and $c[\,]$ contain samples of functions $f(x)$, $g(x)$ and $c(x)$, that is $f[m] = f(m\Delta)$, $g[m] = g(m\Delta)$, $c[n] = c(n\Delta)$, where $m$, $n$ are integer and $\Delta$ is a common sampling distance. Please note that an infinite number of samples is assumed here. Also note that we assume correct sampling according to the Nyquist-Shannon sampling theorem.

It is important to emphasise that the discrete convolution is a good counterpart of the continuous convolution: if we calculate the continuous convolution $c(x) = f(x) \otimes g(x)$ and sample $c(x)$, we get the same result as if we sampled functions $f$ and $g$ first and calculated $c[\,] = f[\,] \otimes_d g[\,]$.

Unfortunately, the discrete convolution requires signals of infinite extent. On the other hand, there is *the discrete cyclic convolution* that works with finite extent signals:

$$c'[\,] = f[\,] \otimes g[\,],$$

where

$$c'[n] = \sum_{m=0}^{C-1} f[m] g[(n-m) \bmod C].$$

Here we assume that signals $f[\,]$, $g[\,]$ and $c'[\,]$ contain $C$ samples each and that they are numbered from 0 to $C-1$. Please note that we are using the same symbol $\otimes$ for both the continuous and the discrete cyclic convolution. As they use continuous or discrete operands, respectively, it is clear which one is actually used. The discrete convolution $\otimes_d$ is used mostly in proofs.

The discrete cyclic convolution has the same useful property as the continuous convolution:

$$c'[\,] = f[\,] \otimes [\,] = \mathbf{IDFT}\Big\{\mathbf{DFT}\{f[\,]\} \odot \mathbf{DFT}\{g[\,]\}\Big\}, \tag{3.5}$$

where $\mathbf{DFT}\{\cdot\}$ and $\mathbf{IDFT}\{\cdot\}$ stand for *the discrete Fourier transform* and its inverse, respectively, and $\odot$ is the Hadamard (element-wise) product. The discrete transforms are defined by equations

$$\begin{aligned} \mathbf{DFT}\{g[\,]\} &= G[\,] \\ G[m] &= \sum_{n=0}^{C-1} g[n] \exp\left(-\mathbf{j}\, 2\pi \frac{mn}{C}\right), \\ m &\in \{0, 1, 2, \ldots, C-1\}, \end{aligned}$$

and

$$\begin{aligned} \mathbf{IDFT}\{G[\,]\} &= g[\,], \\ g[n] &= \frac{1}{C} \sum_{m=0}^{C-1} G[m] \exp\left(\mathbf{j}\, 2\pi \frac{mn}{C}\right), \\ n &\in \{0, 1, 2, \ldots, C-1\}, \end{aligned}$$

and similarly in higher dimensions. Calculating the cyclic convolution using the discrete Fourier transform has a unique property: the discrete Fourier transform can be calculated for all indices as once with computational complexity

$\mathcal{O}(C \log C)$. The algorithm for fast evaluation of the discrete Fourier transform is usually called *the fast Fourier transform.*

While the discrete convolution $\otimes_d$ is a good counterpart of the continuous convolution, there is no reason to expect the same from the discrete cyclic convolution. In the following paragraphs, we will explore how to utilize the discrete cyclic convolution (as it can be calculated fast) to approximate the continuous convolution (as this is what we need).

Here it is worth emphasising an important fact: we will be talking about convolutions only. The fact that the discrete cyclic convolution can be calculated fast will not be important. We will not expect *anything* from the discrete Fourier transform – we are not interested if it is a good approximation of the continuous Fourier transform, or if it is not. The discrete transform is just “a black box” that makes our lives easier, are we are not interested in how it works.

## Utilizing the discrete cyclic transform

Let us review our task for clarity. We know light phasor $U(x, y, 0)$ in the rectangular area $\Omega$ in the plane $z = 0$; we assume $U(x, y, 0) = 0$ outside $\Omega$. We want to calculate light propagation to the plane $z = z_0$; in particular, we will be interested in $U(x, y, z_0)$ in a rectangular area $\Omega'$ in $z = z_0$ only.

For simplicity, let us work in 2-D, i.e. with coordinates $x$, $z$ only; generalization to 3-D is straightforward. Let us denote light phasor in $z = 0$ as $s(x)$; the letter $s$ stands for *the source.* We are interested in light phasor in $z = z_0$; let us denote it $t(x)$, the letter $t$ stands for *the target.* Clearly,

$$t(x) = s(x) \otimes p(x), \tag{3.6}$$

where $p(x)$ is some function describing light propagation, for example the function $K_{RS}$ from Equation (3.4).

It is possible to use the discrete convolution $\otimes_d$ to approximate the result at $x = n\Delta$:

$$t(n\Delta) = t[n] = \sum_{m=-\infty}^{\infty} s[m]\, p[n-m], \tag{3.7}$$

where again $s[m] = s(m\Delta)$, $p[n-m] = p((n-m)\Delta)$.

As we assume $s(x) = 0$ outside $\Omega$, the sum need not be infinite. Without loss of generality, let us assume that $s(x) = 0$ for $x < 0$ and for $x > (M-1)\Delta$, i.e., the only $s[m] \neq 0$ must satisfy $0 \leq m \leq M-1$. Thus,

$$t[n] = \sum_{m=0}^{M-1} s[m]\, p[n-m].$$

Without loss of generality, let us assume that we want to calculate $t(x)$ in an area $\Omega'$: $0 \leq x \leq (N-1)\Delta$, i.e., we want to calculate samples $t[n]$ for $0 \leq n \leq N-1$. Clearly, if we want to calculate $t[0]$, we need values $p[-(M-1)], \ldots, p[-1]$,

$p[0]$. It is also clear that calculation of $t[N-1]$ requires values $p[(N-1)-(M-1)]$, $\ldots$, $p[N-2]$, $p[N-1]$. Thus, to calculate $t[n]$ for every $n$, we need just values $p[-(M-1)]$, $\ldots$, $p[N-1]$, that is $(M+N-1)$ samples of $p(x)$.

It follows we can define signal $p'[\,]$ with $C$ samples numbered from 0 to $C-1$:

$$p'[m] = \begin{cases} p[m] & \text{for } 0 \le m \le N-1, \\ p[m-C] & \text{for } C-(M-1) \le m \le C-1, \\ \text{arbitrary} & \text{elsewhere.} \end{cases} \tag{3.8}$$

As the cases must be disjunctive, it follows $C \ge M+N-1$. There is no other restriction on $C$ value. For example, for $M=3$, $N=4$ and $C=8$ we have

$$p'[m] = \begin{cases} p[m] & \text{for } 0 \le m \le 3, \\ p[m-8] & \text{for } 6 \le m \le 7, \\ 0 & \text{elsewhere} \end{cases}$$

that is

| $m$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| $p'[m]$ | $p[0]$ | $p[1]$ | $p[2]$ | $p[3]$ | 0 | 0 | $p[-2]$ | $p[-1]$ |

We can also define signal $s'[\,]$ with $C$ samples by zero-padding the signal $s[\,]$:

$$s'[m] = \begin{cases} s[m] & \text{for } 0 \le m \le M-1, \\ 0 & \text{for } M \le m \le C-1, \end{cases}$$

that is for our example numbers $M=3$, $N=4$ and $C=8$

| $m$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| $s'[m]$ | $s[0]$ | $s[1]$ | $s[2]$ | 0 | 0 | 0 | 0 | 0 |

Now it is possible to calculate the discrete cyclic convolution

$$t'[n] = \sum_{m=0}^{C-1} s'[m]\, p'[(n-m) \bmod C]. \tag{3.9}$$

If we subsitute $n=0$, we get

$$\begin{aligned} t'[0] &= \sum_{m=0}^{C-1} s'[m]\, p'[(-m) \bmod C] \\ &= \sum_{m=0}^{M-1} s[m]\, p[-m] \\ &= t[0]. \end{aligned}$$

(it is helpful to use the example $s'[\,]$ and $p'[\,]$ tables above to see that). Close inspection of other values of $n$ in (3.9) reveals that $t'[n] = t[n]$ for all $0 \le n \le N-1$.

We can conclude that to calculate $N$ samples of the target $t(x)$ of light propagating from $M$ samples of the source $s(x)$, it is necessary

1. to choose any value $C \geq M + N - 1$,
2. to prepare the signal $s'[\,]$ by zero-padding the signal $s[\,]$ to $C$ samples,
3. to pre-calculate $C$ samples of the signal $p'[\,]$,
4. to calculate the discrete cyclic convolution $t'[\,] = s'[\,] \otimes p'[\,]$, for example by using the fast Fourier transform, see Equation 3.5,
5. to extract $N$ samples of $t'[\,]$, i.e. to define the result $t[n] = t'[n]$ for $0 \leq n \leq N - 1$.

## 3.3 Reference calculation of light propagation between parallel planes of different sizes and sampling rates

The article *Reference calculation of light propagation between parallel planes of different sizes and sampling rates* [33], which is reprinted in this section, extends ideas introduced in Section 3.2.

First of all, it discusses a method that allows (within some constraints) different sampling distances in the source and the target areas. Second, it discusses how to make the calculation less memory intensive. Recall that for the calculation of propagation from the source of size $M \times M$ samples to the target of size $M \times M$ samples, it is necessary to use auxiliary signals of sizes $2M \times 2M$ samples, i.e., four times larger than the source and the target themselves. Third, it mentions that a proper choice of the convolution kernel ($p(x)$ in Equation (3.6)) allows to calculate off-axis propagation. Finally, it mentions that size of the auxiliary signals (number $C$ from Equation (3.8)) can be chosen such that the calculation of the fast Fourier transform is effective. In conclusion, it describes the method that can be used as “the reference” under variety of situations.

The article does not discuss some details that should be emphasised here.

First, the convolution kernel $p[\,]$ introduced in Equation (3.7) was supposed to be “full”, i.e. we supposed that all of its $M + N - 1$ samples are non-zero. This, however, need not be true. In practice, the function $p(x)$ can have small support, i.e. just most samples of $p[\,]$ are zero. If we take this into account, the calculation can be more effective.

Second, the method assumes correct sampling of the continuous functions involved, especially of the convolution kernel $p(x)$. This assumption is thoroughly discussed in Chapter 4. An alternative interpretation of the discretisation process is provided in Chapter 5.

# Reference calculation of light propagation between parallel planes of different sizes and sampling rates

**Petr Lobaz***

*Department of Computer Science and Engineering, University of West Bohemia, Univerzitní 8, 306 14 Plzeň, Czech Republic*
**lobaz@kiv.zcu.cz*

**Abstract:** The article deals with a method of calculation of off-axis light propagation between parallel planes using discretization of the Rayleigh-Sommerfeld integral and its implementation by fast convolution. It analyses zero-padding in case of different plane sizes. In case of memory restrictions, it suggests splitting the calculation into tiles and shows that splitting leads to a faster calculation when plane sizes are a lot different. Next, it suggests how to calculate propagation in case of different sampling rates by splitting planes into interleaved tiles and shows this to be faster than zero-padding and direct calculation. Neither the speedup nor memory-saving method decreases accuracy; the aim of the proposed method is to provide reference data that can be compared to the results of faster and less precise methods.

**OCIS codes:** (070.2025) Discrete optical signal processing; (070.7345) Wave propagation.

## References and links

1. J. W. Goodman, *Introduction to Fourier Optics* (Roberts & Company Publishers, 2004), 3rd ed.
2. E. Lalor, "Conditions for the validity of the angular spectrum of plane waves," J. Opt. Soc. Am. **58**(9), 1235–1237 (1968).
3. L. Onural, "Exact analysis of the effects of sampling of the scalar diffraction field," J. Opt. Soc. Am. A **24**(2), 359–367 (2007).
4. L. Onural, A. Gotchev, H. Ozaktas, and E. Stoykova, "A survey of signal processing problems and tools in holographic three-dimensional television," IEEE Trans. Circ. Syst. Video Tech. **17**(11), 1631–1646 (2007).
5. V. Katkovnik, J. Astola, and K. Egiazarian, "Discrete diffraction transform for propagation, reconstruction, and design of wavefield distributions," Appl. Opt. **47**(19), 3481–3493 (2008).
6. N. Delen, and B. Hooker, "Free-space beam propagation between arbitrarily oriented planes based on full diffraction theory: a fast Fourier transform approach," J. Opt. Soc. Am. A **15**(4), 857–867 (1998).
7. J.-L. Kaiser, E. Quertemont, and R. Chevallier, "Light propagation in the pseudo-paraxial fresnel approximation," Opt. Commun. **233**(4-6), 261–269 (2004).
8. E. Sziklas, and A. Siegman, "Diffraction calculations using fast Fourier transform methods," Proc. IEEE **62**(3), 410–412 (1974).
9. R. P. Muffoletto, J. M. Tyler, and J. E. Tohline, "Shifted Fresnel diffraction for computational holography," Opt. Express **15**(9), 5631–5640 (2007).
10. F. Zhang, G. Pedrini, and W. Osten, "Reconstruction algorithm for high-numerical-aperture holograms with diffraction-limited resolution," Opt. Lett. **31**(11), 1633–1635 (2006).
11. K. Matsushima, and S. Nakahara, "Extremely high-definition full-parallax computer-generated hologram created by the polygon-based method," Appl. Opt. **48**(34), H54–H63 (2009).
12. M. Frigo, and S. G. Johnson, "The design and implementation of FFTW3," Proc. IEEE **93**(2), 216–231 (2005) (Special issue on "Program Generation, Optimization, and Platform Adaptation").

## 1. Introduction

A fundamental tool of digital holography, or computer generated holography, is a numerical simulation of coherent light propagating in free space. We will use, as usual, scalar approximation of the vectorial nature of light, and will not consider time-dependent behavior of the light [1]. One of the most usual tasks is to calculate light propagation between two

#137599 - $15.00 USD Received 2 Nov 2010; revised 10 Dec 2010; accepted 12 Dec 2010; published 20 Dec 2010

parallel planes; the Rayleigh-Sommerfeld integral [1], its mathematical equivalent angular spectrum [1,2] or their various approximations are used most often.

Those equations need to be discretized for numerical calculation, i.e. one has to both sample and spatially restrict optical fields involved. The discretization itself leads to various errors well described in literature [3,4]. The calculation tries to avoid those errors while trying to work as fast as possible.

It is, however, often hard to decide what is an error of discretization itself, what is an inherent error of a given fast method and what is just an error of a particular implementation. The goal of this article is to describe the reference numerical calculation of light propagation between parallel planes that has just one "error", the discretization. Any other method can be then compared to this one. As we will assume fine sampling that does not lead to aliasing errors, we have to deal with a large amount of data. We will try to handle it as fast as possible while retaining the accuracy of the calculation.

The proposed method focuses on off-axis light propagation between two rectangular areas that share neither size nor sampling, or, between a spatial light modulator (SLM) and a camera sensor. Reference calculation is described, e.g., in [5], off-axis propagation, e.g., in [6,7]; different samplings are treated in [8] by coordinate system change, in [9] by shifted convolution kernel, in [10] by scaled Fourier transform; different sizes of SLM and sensor is solved in [10,11] using tiling while decreasing memory demands. This article solves all the requirements in a unified way by using the convolution approach and tiling; in contrast to references given, it deals with optimization too.

## 2. One-dimensional case

We are going to show all the principles in a one-dimensional case before showing the full 2D version. This means that we will calculate light propagation between two line segments instead of two rectangular areas. We will call them *Source* and *Target*.

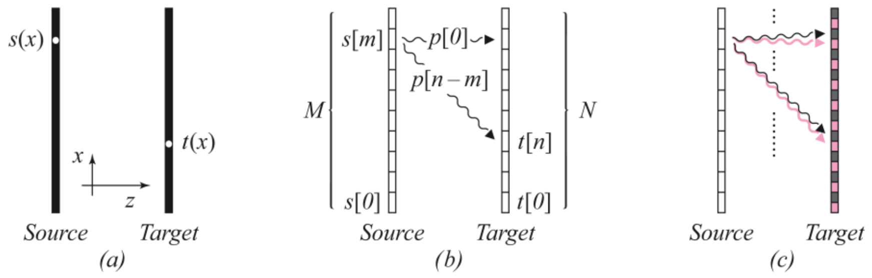


Fig. 1. One-dimensional light propagation. Description of (a), (b), (c) is given in the text below.

In linear optics it is assumed that light sources do not influence each other and that every single point $t(x)$ of *Target* (e.g. a camera sensor) is affected by the shining of all points $s(x)$ of *Source* (e.g. a SLM, see Fig. 1a). As mentioned in the introduction, we will assume scalar approximation of coherent light, i.e. the light source can be completely described by amplitude $A$ and phase $\phi$, or complex amplitude $A\exp(\mathrm{j}\phi)$, where $\mathrm{j}^2 = -1$.

Light changes both amplitude and phase by propagation. This change can by described by multiplication with some complex number $p$. Light propagation is space invariant; therefore, it is just the mutual position of points on *Source* and *Target* that matters, not their absolute positions. It follows that the calculation will have a convolution form:

$$t(x) = \int_{Source} s(\xi)\, p(x-\xi)\,\mathrm{d}\xi \qquad (1)$$

In the equation there is no distance along $z$ axis between *Source* and *Target* because it is a constant and can be a part of the function $p$.

We can discretize the equation by equidistant splitting of *Source* and *Target* into $M$ and $N$ basic elements $s[m]$ ($0 \le m \le M-1$) and $t[n]$ ($0 \le n \le N-1$), i.e. by uniform sampling. Therefore, the element $t[n]$ receives from the element $s[m]$ light with complex amplitude $s[m]p[n-m]$, see Fig. 1b. Equation (1) can be written in discrete form:

$$t[n] = \sum_{m=0}^{M-1} s[m]p[n-m] \qquad \text{for } 0 \le m \le M-1, \quad 0 \le n \le N-1. \tag{2}$$

To compute all elements $t[n]$, we have to know numbers $p[n-m]$ for $-(M-1) \le n-m \le N-1$, i.e. $M+N-1$ different values. We can obtain them in different ways, some of which will be mentioned in section 3. For now, let us assume we know them.

The calculation of all elements $t[n]$ is most often done by rewriting Eq. (2) as a cyclic convolution and subsequent use of the discrete Fourier transform. The cyclic convolution has a form:

$$t[n] = \sum_{m=0}^{C-1} s[m]p[(n-m) \bmod C] \qquad \text{for } 0 \le m \le C-1, \quad 0 \le n \le C-1, \tag{3}$$

where $C \ge M+N-1$ and $s[m]=0$ for $M \le m \le C-1$ (i.e., the $s$ is zero-padded to the size $C$). Notice that the important values of $t[n]$ are those for $0 \le n \le C-M$; the others are damaged by the cyclic behavior of indices in arithmetic $(\bmod C)$. It is also worth mentioning that in cyclic convolution it is usually assumed that $M = N$. The proof of validity for the case $M \ne N$ consists just in the expansion of equations for $t[n]$. Finally, $C$ is an arbitrary number bigger than or equal to $M+N-1$. By choosing a suitable $C$, we can speed up the computation significantly, because

$$t = \mathbf{IDFT}(\mathbf{DFT}(s) \times \mathbf{DFT}(p)), \tag{4}$$

where $t$, $s$ and $p$ are $C$-dimensional vectors (arrays) of complex numbers, $\mathbf{DFT}$ is a discrete Fourier transform of a vector, $\mathbf{IDFT}$ is an inverse discrete Fourier transform of a vector and $\times$ is the Hadamard product (element by element product). The speedup is expected due to the fact that the calculation of $\mathbf{DFT}$, or $\mathbf{IDFT}$, can be done in time $O(C \log C)$ [12]. Choosing a suitable $C$ is important because the actual calculation time is highly sensitive to its character.

Let us assume that the sampling rate of *Target* is twice as fine as the sampling rate of *Source*. Then the subset of even samples from *Target* has the same sampling rate as *Source*. Consequently, we can easily calculate the propagation of *Source* to even samples of *Target*. Obviously, we can do the same with odd samples. It means we can split the calculation of light propagation into two calculations; they are denoted in Fig. 1c by black and magenta arrows. It follows that the same principle can be applied when the sampling rate of *Target* is $\tau$-times finer than the sampling rate of *Source*, where $\tau$ is a natural number; we have to split *Target* into $\tau$ "*interleaved tiles*", i.e. the calculation has to be split into $\tau$ calculations.

The same situation appears when *Source* has $\sigma$-times finer sampling than *Target*. The idea can be generalized: if the sampling rates of *Source* and *Target* are in a ratio $\sigma : \tau$, where $\sigma$ and $\tau$ are coprime natural numbers (i.e. $\sigma$ elements of *Source* have the same size as $\tau$ elements of *Target*), we can split the calculation into $\sigma \times \tau$ independent calculations. More precisely, we have to $\sigma$-times calculate the propagation for the sampling rate ratio $1 : \tau$ and sum the results. Usually we do not care about the exact value of the sampling rate; therefore, we can choose such $\sigma$ and $\tau$ that approximate the desired sampling fairly well.

It follows from Eq. (3) that the vectors $s$ and $t$ have to be padded to size $C$. This means that for $M = N$, approximately 50% of elements are held in memory uselessly; in 2D convolution, it is as much as 75%. Therefore, for big $M$ and $N$ we can expect a lack of memory very soon, especially when using special hardware for **DFT** calculation, e.g. GPU. For example a naive approach to propagation of a microscopic *Source* to an extended detector *Target* could require hundreds of gigabytes.

We need two memory spaces of size $C \geq M + N - 1$ for the calculation of Eq. (4); in practice a restriction may appear: that just two spaces of size $P < C$ are available. Let us assume, for example, that $M = 512$, $N = 1024$ and $P = 1024$. We cannot make the calculation directly because $C \geq 1535$; but we can split *Target* in the middle into two parts (let us call them "*common tiles*") with $N' = 512$ elements and make two calculations. For them, we need only two spaces of size at least $M + N' - 1 = 1023$, and therefore we are not limited by $P = 1024$. The same idea would apply if $M = 1024$, $N = 512$ and $P = 1024$: we would calculate the light propagation of both small parts of *Source* to *Target* and sum the results. As in the "different sampling" case, even this idea can be generalized: *Source* can be split into $S$ parts, *Target* into $T$ parts and then we have to calculate $S \times T$ propagations.

## 3. Two-dimensional case

The extension of equations from section 2 to the 2D case is straightforward: instead of sums we just put double sums there. For calculation of light propagation of a part of *Source* to a part of *Target*, it is necessary to carefully calculate the convolution kernel, i.e. 2D array $p$. A practical aid is the fact that its element $p[0,0]$ describes light propagation from element $s[0,0]$ of a particular part of *Source* to element $t[0,0]$ of a particular part of *Target*. The equations for sampling rates, samples counts and offsets are simple but technically demanding, so we will not show them here.

We should, however, mention the calculation of convolution kernel values. The Rayleigh-Sommerfeld equation [2] for the light propagation from the plane $z = 0$ to point $[x, y, z]$ is:

$$U(x,y,z) = \frac{-1}{2\pi} \iint_{-\infty}^{\infty} U(\xi,\eta,0) \frac{\partial}{\partial z} \frac{\exp(\mathrm{j}kr)}{r} \mathrm{d}\xi \mathrm{d}\eta$$

where $U(x,y,z)$ is a complex amplitude of light in a point $[x,y,z]$, $k = 2\pi/\lambda$ is a wavenumber ($\lambda$ is a wavelength), and $r$ is the distance between points $[x,y,z]$ and $[\xi,\eta,0]$. This equation can be written in a convolution form:

$$U(x,y,z) = U(x,y,0) \otimes \left[ \frac{-z}{2\pi} \left( \mathrm{j}k - \frac{1}{\sqrt{x^2+y^2+z^2}} \right) \frac{\exp(\mathrm{j}k\sqrt{x^2+y^2+z^2})}{x^2+y^2+z^2} \right] \quad (5)$$

where $\otimes$ is the 2D convolution operator, $(f \otimes g)(x,y) = \iint_{-\infty}^{\infty} f(\xi,\eta) g(x-\xi, y-\eta) \mathrm{d}\xi \mathrm{d}\eta$.

The simplest method of convolution kernel discretization (the expression on the right side of the convolution operator in Eq. (5)) is a plain sampling, i.e. its calculation for a particular $x$, $y$ ($z$ is a constant). Alternatively, we can assume that a sample of *Source* in fact expresses – using some pixel spread function – the shining of a particular non-zero-area element of *Source* and alter the kernel accordingly. If we take non-zero area of *Target* (sensor) elements into account, we can pre-filter the kernel. If *Source* is a mathematical model of a real display, we can even measure the kernel. The proposed method therefore does not depend on particular features of the kernel; its only assumption is the description of light propagation as a convolution. In our implementation, we did not deal with advanced methods of kernel calculation, however, and we have chosen plain sampling.

**4. Theoretical time of computation**

The calculation works with arrays of size $C = C_x \times C_y$ samples. It consists of three **DFT**'s (more precisely, two forward and one backward), calculation of the convolution kernel $p$ with complexity $O(C)$ and the Hadamard product of the same complexity. For a large $C$, only times spent on **DFT**'s matter. The time of calculation using the fast Fourier transform is therefore proportional [12] to

$$(\mathbf{DFT}\text{'s count})C\log C \tag{6}$$

In the following analysis, we will assume light propagation from a square area of size $M \times M$ samples to a square area $N \times N$ samples. For the convolution calculation, we will assume memory space $C = (M+N)\times(M+N)$. Time of calculation is then given by:

$$t_{\text{basic}}(M,N) = 3(M+N)^2 \log(M+N)^2 \equiv 6(M+N)^2 \log(M+N)$$

Let us begin with the case that *Source* and *Target* share the sampling. Then we can also split *Target* into $T \times T$ common tiles of size $(N/T)\times(N/T)$. It follows that we calculate $\mathbf{DFT}(\mathit{Source})$ once and calculate $\mathbf{DFT}(p)$ and $\mathbf{IDFT}(\mathbf{DFT}(\mathit{Source})\otimes\mathbf{DFT}(p))$ $T \times T$-times, while we assume the array sizes for **DFT** as $C = (M+N/T)\times(M+N/T)$. Using Eq. (6), we get time of calculation

$$t_{\text{common tiles}}(M,N,T) = \left(1+2T^2\right)2\left(M+\frac{N}{T}\right)^2 \log\left(M+\frac{N}{T}\right)$$

Let us assume that $M = \omega N$ and watch the speedup

$$s_1(\omega,N,T) = \frac{t_{\text{basic}}(\omega N,N)}{t_{\text{common tiles}}(\omega N,N,T)}$$

of tiled calculation versus direct calculation. The graph of $s_1(\omega,N,T)$ shows that tiled calculation starts to be faster for $T=2$ and $\omega < 1/4$, while for bigger $T$ it is faster for even smaller $\omega$. Figure 2a shows the graph for $N = 1024$; for different $N$ it does not change very much.

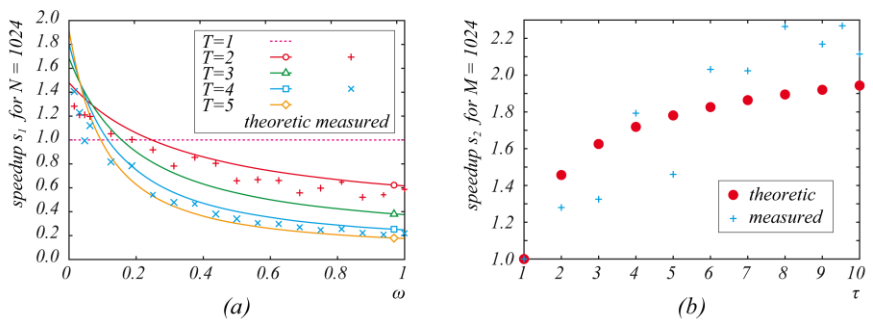


Fig. 2. The ratio of direct calculation time to (a) tiled calculation for given ratio $\omega$ of *Source* to *Target* side sizes (for *Target* side size $N$ = 1024), and (b) tiled calculation for given ratio $\tau$ of *Target* to *Source* and sampling rates (for *Source* side size $M$ = 1024). FFTW library was used for time measurement.

A more interesting result arises when *Source* and *Target* have different sampling rates. Let us assume that *Target* has $\tau$ -times finer sampling than *Source*, i.e. $N = \tau M$. For direct calculation, *Source* must have the same sampling rate as *Target*, i.e. it must have $\tau M \times \tau M$ samples too. We can increase number of samples by interpolation or just by interleaving current ones by a suitable amount of zeros. By substituting $C = (\tau M + \tau M) \times (\tau M + \tau M)$ into Eq. (6), we get time of calculation

$$t_{\text{upscaled}}(M,\tau) = 6(\tau M + \tau M)^2 \log(\tau M + \tau M) \equiv 24\tau^2 M^2 \log(2\tau M)$$

If we split *Target* into $\tau \times \tau$ interleaved tiles, we have to calculate $\tau \times \tau$ light propagations from *Source* to a part of *Target* of size $M \times M$ samples. By substituting $C = (M + M) \times (M + M)$ into Eq. (6), we get time of calculation

$$t_{\text{interleaved tiles}}(M,\tau) = \left(1+2\tau^2\right) 2\left(M+M\right)^2 \log\left(M+M\right) \equiv \left(1+2\tau^2\right) 8M^2 \log\left(2M\right)$$

The speedup

$$s_2(M,\tau) = \frac{t_{\text{upscaled}}(M,\tau)}{t_{\text{interleaved tiles}}(M,\tau)} \equiv \frac{3\tau^2 \log(2\tau M)}{\left(1+2\tau^2\right)\log\left(2M\right)}$$

is bigger than 1 for $\tau > 1$, i. e., tiled calculation is always faster; a graph of $s_2(M,\tau)$ dependent on $\tau$ for $M = 1024$ is in Fig. 2b. Again, a different $M$ leads to a similar graph.

One can see in Fig. 2 that measured speedups are scattered around theoretical curves. A discussion of this fact is contained in the following section.

Until now we have assumed that *Target* has more samples than *Source*. That led to saving nearly one-third of all **DFT**'s. In the opposite situation this does not hold any more; however, the results of the analysis are similar, although not so outstanding. The analysis of general situations, where the ratio of sampling rates or sizes is a general fraction, leads to the following results: speedup is greater when the nominator and denominator of the sampling rate ratio are big too; splitting of *Source* and *Target* of the same sampling rate into common tiles in a complicated ratio leads rather to slowdown if we do not mind memory restrictions.

## 5. Implementation notes

The implementation of the proposed method is simple, although due to a number of variables quite demanding. The first step is to estimate the ratio of *Source* and *Target* sampling rates, and to split them into interleaved tiles of common sampling. In case we have enough memory to calculate light propagations between these tiles, we can calculate them (but see later). In another case, we are facing a still unresolved problem.

Let us assume that two memory spaces of size $C$ are available. The interleaved tiles created by splitting *Source* and *Target* due to the requirement of common sampling have to be split into $S_x \times S_y$ common tiles of size $M_x \times M_y$, and $T_x \times T_y$ common tiles of size $N_x \times N_y$ (due to integer division one row and one column may be smaller) so that:

$$(M_x + N_x - 1)(M_y + N_y - 1) \le C$$

while the time of the calculation has to be as short as possible. We cannot trust the theoretical time complexity of **DFT** $O(C \log C)$ when searching for optimum; algorithms of **DFT** for special array sizes (e.g. power of 2) are usually faster than for comparable (e.g. prime) array sizes. For example, FFTW library used in our implementation calculates **DFT** of special array sizes up to 6 × faster than for comparable prime sizes. This is the first expected reason for scattered look of the measured data in Fig. 2. One can see in Fig. 2b that measured speedups are bigger than theoretical ones. This behavior starts to appear for $M > 512$ in our implementation. We expect this behavior appears due to a following fact. When calculating a propagation without any tiling, we have to increase *Source* side size $\tau$-times, so **DFT** has to work with a big array. Using interleaved tiling, **DFT** works with smaller arrays that fit into cache memory more easily, and therefore bigger speedup can be expected.

Implemented heuristic suggests splitting in this way. First, we have measured calculation times $time[i]$ of one-dimensional **DFT**'s of array sizes $i$. From these times we have picked "friendly-size" ones that satisfy condition $time[\mathit{friendly\text{-}size}] < time[i]$ for all measured $i > \mathit{friendly\text{-}size}$. Two-dimensional **DFT** is separable, so it is expected that a two-dimensional array of *friendly-size* side sizes will be "friendly" too. Next, we have measured calculation times for those arrays. The propagation itself is then calculated in 2D "friendly-size" arrays. In case we need to split into common tiles due to memory restrictions, we choose tiles of maximum width (tiles are then in fact horizontal stripes); height of *Source* and *Target* tiles is chosen to be approximately equal and as big as possible. We have compared this heuristic to the optimal solution found using brute force; it suggests at most approximately $1.7\times$ worse tiling.

A topic that has not been discussed yet is the precision of the proposed algorithm compared to the precision of direct calculation without any tiling. We have tried to propagate the *Source* to the *Target* of the same size (for several different sizes) using different tiling schemes. We have found that the relative difference of the calculated complex amplitudes is in the order of $10^{-10}$ or smaller; the difference was calculated as the absolute value of complex amplitudes differences. This difference is so small that we have not tried to find where it comes from.

## 6. Conclusion

A method for reference calculation of light propagation between two rectangular areas of parallel planes has been presented. These areas do not have to have either the same sampling rate or the same size, see Fig. 3. The calculation can be split into tiles to meet memory restrictions given; on the other hand, saving memory often leads to worse calculation times. We have shown that in the case of a simple sampling rates ratio, the proposed method actually speeds up the calculation up to $2\times$ by splitting the areas into interleaved tiles compared to zero-padding and direct calculation. We have shown as well that if one area is much bigger than the other, it is faster to split the bigger one into common tiles. A still unresolved problem is how to split the calculation into common tiles (due to memory restrictions) to get the fastest calculation. We have, however, proposed suboptimal heuristic to address this problem.

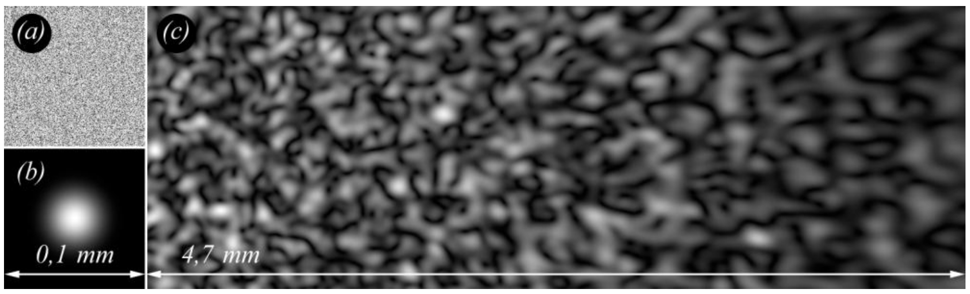


Fig. 3. Speckle simulation: an example of algorithm output. A patch of size 0.1 × 0.1 mm$^2$ with a random phase (a) and a Gaussian intensity (b) was sampled to a 1024 × 1024 array of complex amplitudes. Intensity of the off-axis propagation to the distance 5 mm is shown in the subimage (c), size of the subimage is 4.7 × 1.6 mm$^2$. A naive approach to the convolution would require approx. 28 GiB of memory while proposed algorithm can work on a common PC.

**Acknowledgments**

This work has been supported by the Ministry of Education, Youth and Sports of the Czech Republic under the research program LC-06008 (Center for Computer Graphics). The author would like to thank Ivo Hanák, Václav Skala and the reviewers for their valuable comments.

# Chapter 4

# Filtering in light propagation calculations – the convolution method

This chapter presents the article *Memory-efficient reference calculation of light propagation using the convolution method* [35] published in Optics Express; it starts on page 57. The motivation for this article is a strange behaviour of the convolution method of light propagation calculation. Let us demonstrate it first.

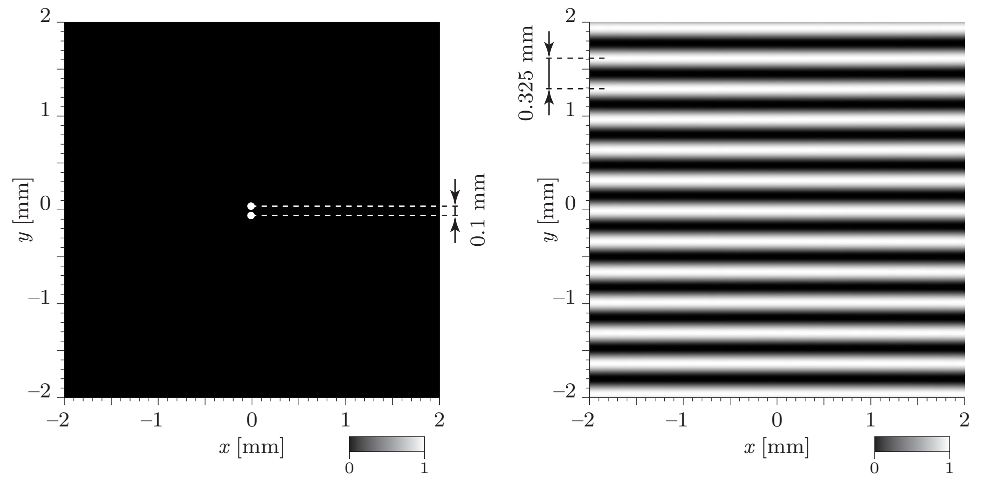
Imagine there are two coherent point light sources in the plane $z = 0$ separated 0.1 mm apart, let us assume their wavelength $\lambda = 650$ nm. They should create an interference pattern. The interference equation (2.2) predicts that in the plane $z = 50$ mm, the fringes should be 0.325 mm apart:

$$d = \frac{\lambda}{\sin\theta_A - \sin\theta_B} = \frac{650 \text{ nm}}{\sin(0.05/50) - \sin(-0.05/50)} \approx 0.325 \text{ mm}.$$

To verify the result, let us prepare *the source* area in the plane $z = 0$. Let us choose its size, for instance $4 \times 4$ mm$^2$, and sampling distances, for instance 10 $\mu$m in both $x$ and $y$ directions. Most samples of the source area will be zero; just two of them, representing two point light sources, will be nonzero, for instance one. The result, calculated by the method presented in Chapter 3, is shown in in Figure 4.1. It confirms the theoretical calculation.

Please note that the source area was sampled by $401 \times 401$ samples. We have chosen the target area of the same size, i.e., it is also sampled by $401 \times 401$ samples.

It is indeed possible to choose different sampling distance to verify the result of the interference equation. Any smaller sampling distance should improve accuracy of discretisation. If we choose, for instance, 2 $\mu$m (i.e. both the source and the target are sampled by $2001 \times 2001$ samples), the calculated target area looks virtually the same. Thus, everything makes us believe that the calculation is correct.



*the source* ($z = 0$ mm) *the target* ($z = 50$ mm)

Figure 4.1: *Interference pattern created by two point light sources separated* 0.1 mm *apart,* $\lambda = 650$ nm*. The interference equation predicted fringes period* 0.325 mm*, which is confirmed. Please note that the points in the source plane have been emphasized to make them clearly visible in this figure.*

As a second experiment, let us try to simulate light diffraction on sinusoidal amplitude diffraction grating illuminated at normal incidence with a plane wave, $\lambda = 650$ nm again. The linear amplitude grating should create diffraction maxima; angles of diffracted light rays are given by the grating equation (2.1) from page 28, repeated here for convenience:

$$\sin\theta_{out,\,m} = m\frac{\lambda}{d} + \sin\theta_{in},$$

where $d$ is the period of the grating.

Thus, the $m$-th diffracted ray should cross the plane $z = z_0$ at a distance $D = z_0 \tan\theta_{out,\,m}$ from the undiffracted ray.

Let us choose $d = 30$ $\mu$m, $\theta_{in} = 0$ (normal incidence) and $m = \pm 1$. It follows that

$$D = z_0 \tan \operatorname{asin} m\frac{\lambda}{d} + \sin\theta_{in} = \pm 50 \times 10^{-3} \tan \operatorname{asin} \frac{650 \times 10^{-9}}{30 \times 10^{-6}} \approx \pm 1.08 \text{ mm}.$$

Let us choose size of the diffraction grating $1 \times 1$ mm$^2$. If we choose the sampling distance 10 $\mu$m again, it is perfectly possible to correctly sample the diffraction grating profile, see Figure 4.2. Please note that the source area is $4 \times 4$ mm$^2$ again, as we have discovered that in such case, propagation to the target area of size $4 \times 4$ mm$^2$ in the plane $z = 50$ mm should be correct.

The result of the propagation calculation is in Figure 4.4. We notice that the plus-minus first diffraction orders are at expected positions.

Theoretical analysis of sinusoidal gratings predicts that the plus-minus first diffraction orders should have just one quarter of the intensity of undiffracted

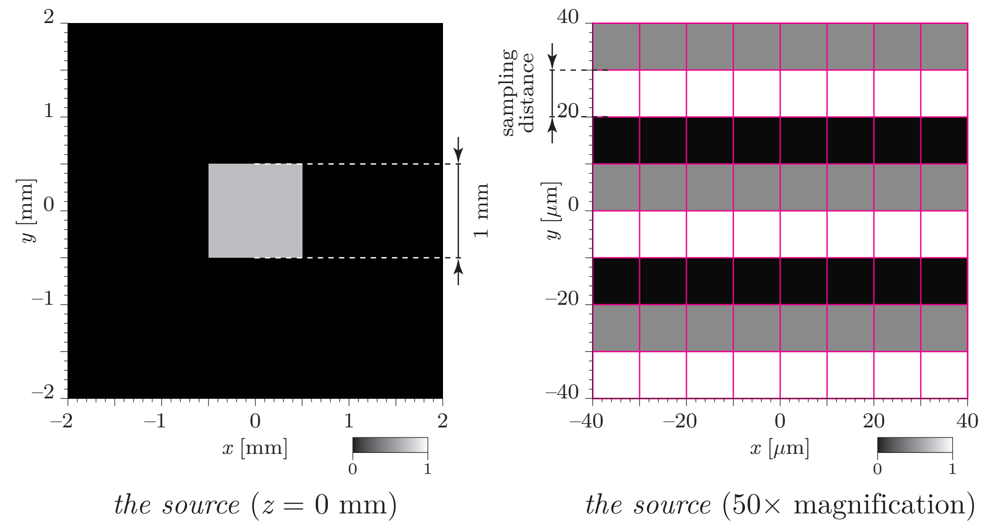


Figure 4.2: *The source area with* $1 \times 1$ mm$^2$ *sinusoidal amplitude diffraction grating of period* $d = 30$ $\mu$m*. The magenta grid depicts individual samples, sampling distance* $\Delta = 10$ $\mu$m *. Note that the the Nyquist-Shannon limit requires at least two samples per period; we have three.*

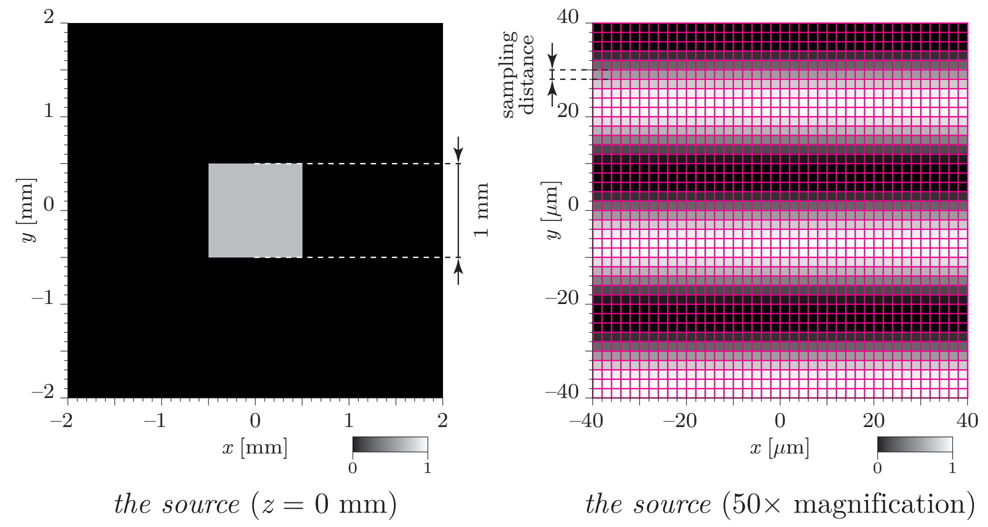
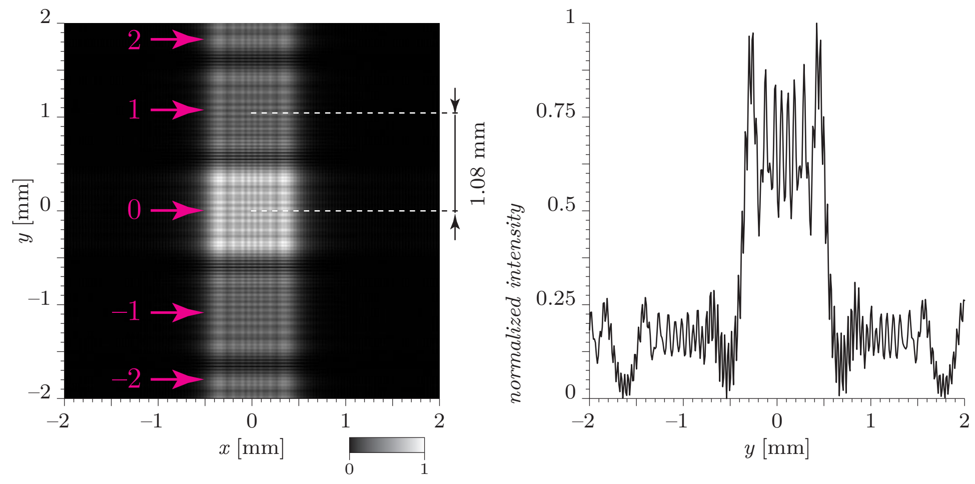


Figure 4.3: *The source area with* $1 \times 1$ mm$^2$ *sinusoidal amplitude diffraction grating of period* $d = 30$ $\mu$m*. The magenta grid depicts individual samples. Note that the sampling frequency* $\Delta = 2$ $\mu$m *allows to capture the sinusoidal profile more precisely.*

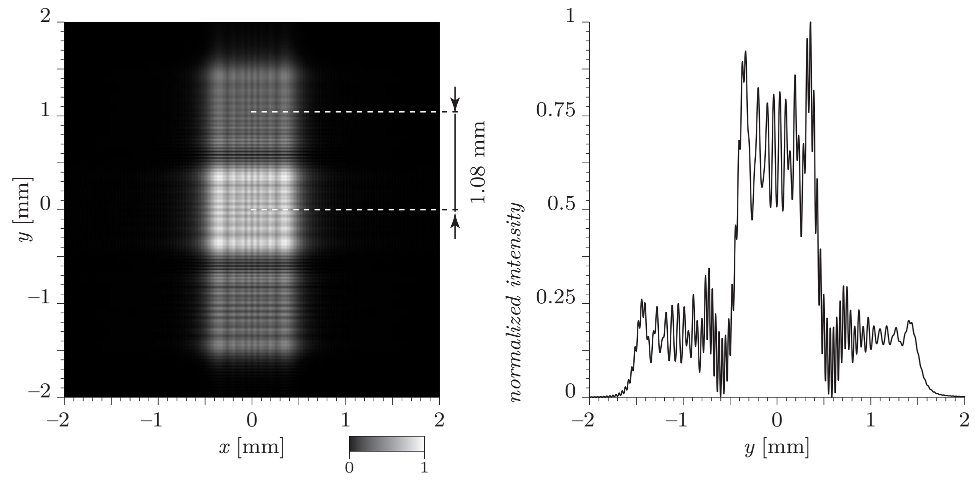
light, see for example [6, Section 4.4.3]. Figure 4.4 reveals that it approximately holds; undiffracted light is the central square. However, the analysis also predicts there should be no higher diffraction orders, but we can clearly see the plus-minus second ones near the target boundary.

If we choose finer sampling distance, for instance 2 $\mu$m again, the result is



*the target* ($z = 50$ mm) *intensity of the target,* $x = 0$

Figure 4.4: *Propagation of the diffraction grating from Figure 4.2 (sampling distance* 10 $\mu$m*) to* $z = 50$ mm*. The magenta numbers depict diffraction orders.*



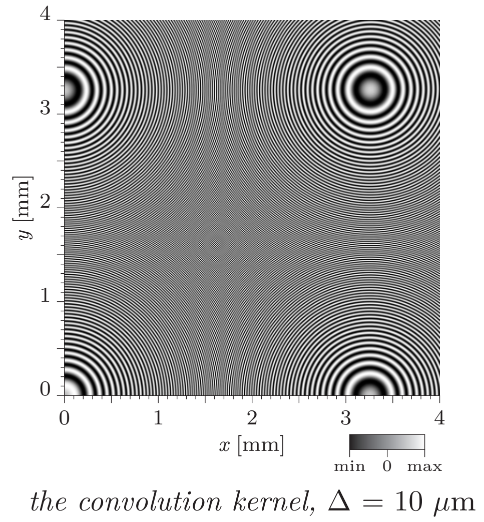
*the target* ($z = 50$ mm) *intensity of the target,* $x = 0$

Figure 4.5: *Propagation of the diffraction grating from Figure 4.3 (sampling distance* 2 $\mu$m*) to* $z = 50$ mm*. Note that there are no higher diffraction orders.*

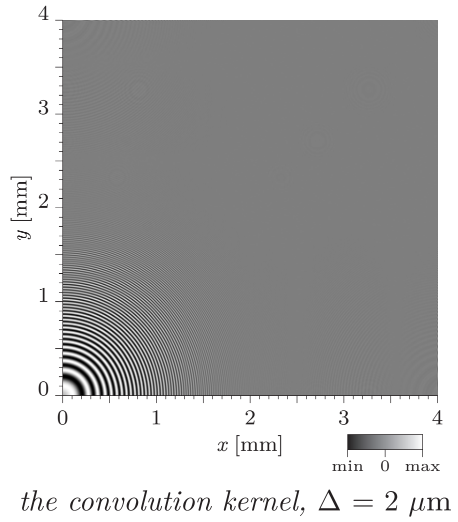
much closer to the theoretical prediction, see Figures 4.3 and 4.5.

The conventional wisdom attributes the wrong result to aliasing (undersampling) of the convolution kernel, see for example [16, 20, 21]. Indeed, if we compare the convolution kernels used for sampling distances $\Delta = 10$ $\mu$m and $\Delta = 2$ $\mu$m, see Figure 4.6, we notice that the one for $\Delta = 10$ $\mu$m shows aliasing artifacts – the convolution kernel, in this simple case, should resemble concentric circles.

Most authors thus analyse frequency content of the functions involved in the calculation (such as the convolution kernel), and low-pass filter them if necessary.

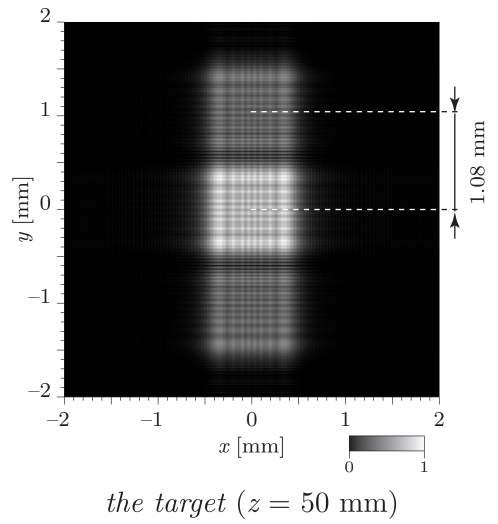


*the convolution kernel,* $\Delta = 10\ \mu$m

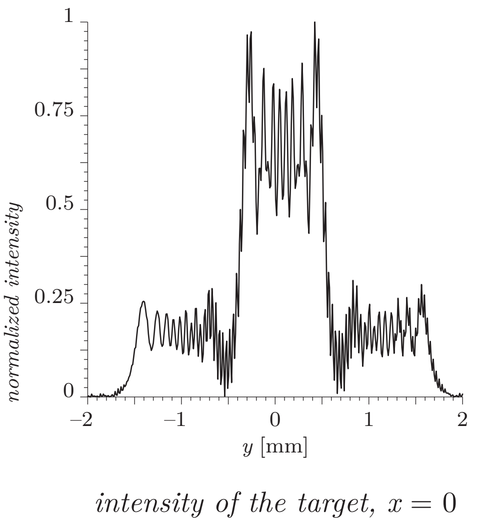


*the convolution kernel,* $\Delta = 2\ \mu$m

Figure 4.6: *Convolution kernels for on-axis propagation of the source to the target at a distance* 50 mm *for two different sampling distances. Common size of the source and target areas is* $4 \times 4$ mm$^2$*. Please note that just one quadrant of the symmetrical kernel is shown; the others are mirror images.*



*the target* ($z = 50$ mm)

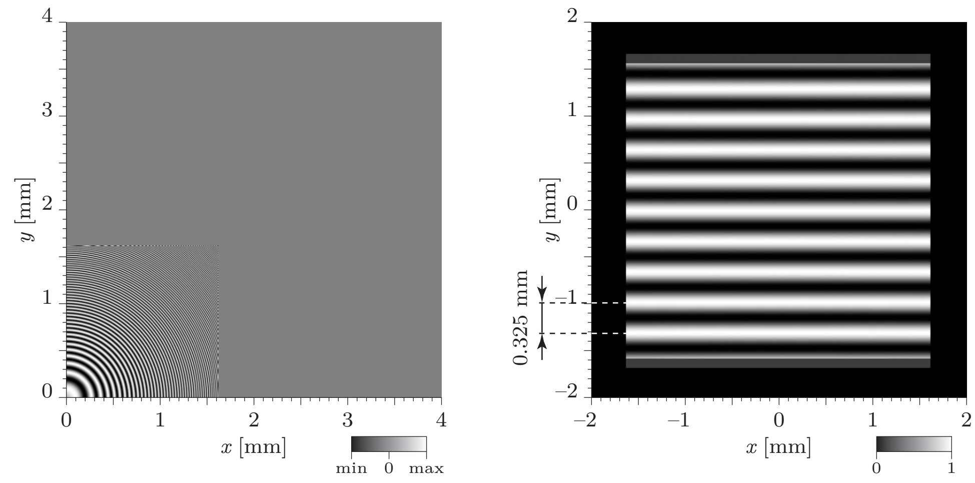


*intensity of the target,* $x = 0$

Figure 4.7: *Propagation of the diffraction grating from Figure 4.2 (sampling distance* 10 $\mu$m*) to* $z = 50$ mm*. Simple low-pass filtering of the convolution kernel was used. Please note that the higher diffraction orders are missing. The convolution kernel is shown in Figure 4.8.*

This process usually gives very good results, see Figure 4.7.

However, there are unanswered questions. First of all: low-pass filtering eliminates an artifact of discretisation. But, what is the physical meaning of this operation? Second: why was the result of calculation of point light sources interference correct for both sampling distances, and why it is not correct in one



*the convolution kernel,* $\Delta = 10\ \mu\text{m}$ *the target* ($z = 50$ mm)

Figure 4.8: *Interference pattern created by two point light sources separated* 0.1 mm *apart,* $\lambda = 650$ nm*. The interference equation predicted fringes period* 0.325 mm*, which is confirmed. However, there is no reason for the black border around the pattern. Please note that just one quadrant of the symmetrical convolution kernel is shown.*

case of the diffraction calculation?

These are not just academic questions. For example, if we use low-pass filtering of the convolution kernel to recalculate the interference pattern of two point light sources (see Figure 4.1), we get rather strange result, see Figure 4.8. Recall that point light sources illuminate the whole space; there is no reason for the interference pattern to be limited by a square. Thus, it seems that low-pass filtering eliminates some artifact, but introduces another.

These questions are answered in the following article.

# Memory-efficient reference calculation of light propagation using the convolution method

**Petr Lobaz**
*Department of Computer Science and Engineering, University of West Bohemia, Univerzitni 8, 306 14 Plzen, Czech Republic*
lobaz@kiv.zcu.cz

**Abstract:** In computational Fourier optics, computer generated holography, etc., coherent light propagation calculation between parallel planes is the essential task. A proper calculation discretization in the off-axis case leads to big memory demands in order to avoid aliasing errors. The proposed method typically cuts down the memory demands one hundred times. The principle of the method is based on the observation that there is a close correspondence between the reconstruction process (opposite of the sampling process) and prefiltering of the convolution kernel.

**OCIS codes:** (070.2025) Discrete optical signal processing; (070.7345) Wave propagation.

## References and links

1. J. W. Goodman, *Introduction to Fourier Optics* (Roberts & Company Publishers, 2004), 3rd ed.
2. D. G. Voelz, *Computational Fourier Optics: A Matlab Tutorial*, Tutorial texts in optical engineering (SPIE Press, 2011).
3. U. Schnars and W. Jueptner, *Digital Holography: Digital Hologram Recording, Numerical Reconstruction, and Related Techniques* (Springer, 2005).
4. K. Matsushima, "Computer-generated holograms for three-dimensional surface objects with shade and texture," Applied Optics **44**, 4607–4614 (2005).
5. I. Hanák, M. Janda, and V. Skala, "Detail-driven digital hologram generation," Visual Computer **26**, 83–96 (2010).
6. Y. Sakamoto, M. Takase, and Y. Aoki, "Hidden surface removal using z-buffer for computer-generated hologram," Practical Holography XVII and Holographic Materials IX **5005**, 276–283 (2003).
7. M. Yamaguchi, "Ray-based and wavefront-based holographic displays for high-density light-field reproduction," Three-Dimensional Imaging, Visualization, and Display 2011 **8043**, 804306 (2011).
8. P. W. M. Tsang, J. P. Liu, K. W. K. Cheung, and T. C. Poon, "Modern methods for fast generation of digital holograms," 3D Research **1**, 11–18–18 (2010).
9. N. Delen and B. Hooker, "Free-space beam propagation between arbitrarily oriented planes based on full diffraction theory: a fast fourier transform approach," J. Opt. Soc. Am. A **15**, 857–867 (1998).
10. K. Matsushima, H. Schimmel, and F. Wyrowski, "Fast calculation method for optical diffraction on tilted planes by use of the angular spectrum of plane waves," J. Opt. Soc. Am. A **20**, 1755–1762 (2003).
11. L. Onural, "Exact solution for scalar diffraction between tilted and translated planes using impulse functions over a surface," J. Opt. Soc. Am. A **28**, 290–295 (2011).
12. E. Lalor, "Conditions for the validity of the angular spectrum of plane waves," J. Opt. Soc. Am. **58**, 1235–1237 (1968).
13. H. M. Ozaktas, S. O. Arik, and T. Coşkun, "Fundamental structure of fresnel diffraction: natural sampling grid and the fractional fourier transform," Opt. Lett. **36**, 2524–2526 (2011).
14. K. Matsushima, "Shifted angular spectrum method for off-axis numerical propagation," Opt. Express **18**, 18453–18463 (2010).

#180735 - $15.00 USD Received 29 Nov 2012; revised 11 Jan 2013; accepted 11 Jan 2013; published 29 Jan 2013

15. L. Onural, “Exact analysis of the effects of sampling of the scalar diffraction field,” J. Opt. Soc. Am. A **24**, 359–367 (2007).
16. L. Onural, A. Gotchev, H. Ozaktas, and E. Stoykova, “A survey of signal processing problems and tools in holographic three-dimensional television,” Circuits and Systems for Video Technology, IEEE Transactions on **17**, 1631 –1646 (2007).
17. P. Lobaz, “Reference calculation of light propagation between parallel planes of different sizes and sampling rates,” Opt. Express **19**, 32–39 (2011).
18. V. Katkovnik, J. Astola, and K. Egiazarian, “Discrete diffraction transform for propagation, reconstruction, and design of wavefield distributions,” Appl. Opt. **47**, 3481–3493 (2008).
19. E. Steward, *Fourier Optics: An Introduction*, Ellis Horwood Series in Physics (Dover Publications, 2004), 2nd ed.
20. K. Turkowski, “Filters for common resampling tasks,” in “Graphics gems,” , A. S. Glassner, ed. (Academic Press Professional, Inc., San Diego, CA, USA, 1990).
21. D. P. Mitchell and A. N. Netravali, “Reconstruction filters in computer-graphics,” SIGGRAPH Comput. Graph. **22**, 221–228 (1988).

## 1. Introduction

Calculation of coherent light propagation in a free space is a fundamental tool in Fourier optics [1, 2], digital holography [3], computer generated holography (e.g. [4–8]) and other areas of optics. A very important and common task is the calculation of light propagation between two parallel planes; however, the general case of light propagation between tilted planes is also important in applications [9–11].

The problem is often given in this way: there is an area $\Sigma$ in a plane $z = 0$ containing an image called the *source* that is lit by a coherent light, mostly by a plane wave. The task is to calculate the light field in a plane $z = z_0$ in an area called the *target*.

In this task, we usually assume validity of the scalar approximation of the light [1]. A good approximation of the correct solution is then given by, e.g., a Rayleigh-Sommerfeld integral of the first kind. In this article, let us assume this approximation as the reference one.

The Rayleigh-Sommerfeld solution cannot usually be used in an analytic calculation due to its complexity. Sometimes it is possible to get some results by using its mathematically equivalent form, the angular spectrum decomposition [12]. It is, however, most usual to restrict the calculation to the paraxial approximation in either the near (Fresnel) or far (Fraunhofer) region.

It is possible to evaluate the “reference” Rayleigh-Sommerfeld integral numerically using computers; thanks to its form of convolution, the calculation leads to the use of three fast Fourier transforms (FFT). The reason for the use of the aforementioned forms or approximations lies in the number of FFT’s: the angular spectrum decomposition leads to two FFT’s; the Fresnel or Fraunhofer approximation lead to just one FFT or fast fractional Fourier transform [13].

The implementations of these faster algorithms are unfortunately not straightforward, as the discretization of their equations leads to various problems. Correct implementation of the angular spectrum decomposition is especially tricky [14]; even Fresnel and Fraunhofer approximations have to be discretized carefully [2, 15, 16].

It is therefore wise to verify fast algorithms by comparing them with a reference method based on the carefully discretized Rayleigh-Sommerfeld integral [17, 18]. The discretization process must consider both correct sampling of the *source* and sampling of the illumination light field and the Rayleigh-Sommerfeld convolution kernel. It is also necessary to consider the inverse operation to the sampling, i.e. the reconstruction. All of these considerations often lead to sampling distances smaller than the wavelength of light, and therefore to huge memory demands. This article studies the discretization process and suggests a method to avoid huge memory demands and consequent time demands of large arrays FFT’s.

The structure of the article is as follows. Section 2 shows a naive method of discretization; the example given will show an illuminated amplitude diffraction grating with a sine transmittance profile. We will show that the naive discretization leads to “wrong” results; we will explain the “wrong” result in a physical way and show that a fine sampling leads to the correct result (and to huge memory demands). In Section 3, we will show how to lower memory demands by a simple 1D example. In Section 4 we will remove certain simplifications introduced in Section 3, and in Section 5 we will discuss the general 2D algorithm. Finally, in Section 6 we will present the time and memory demands of the algorithm and in Section 7 we will give conclusions.

In the rest of the article we will assume SI units. Absolute value of a complex amplitude is electric field amplitude, unit volt/m. For conversion of a complex amplitude to an intensity value, see e.g. [1, 2]. Please also note that 1D examples should not be interpreted physically as propagation integrals were derived for 3D space; they just demonstrate main ideas of the final algorithm.

## 2. Effect of naive discretization

Both theoretical analysis and experiments show that an amplitude diffraction grating with a sine transmittance profile illuminated by a plane wave (let us call it the *source*) creates just three diffraction maxima in the far field – the directly transmitted wave and the plus-minus first diffraction order [1]. Sampling of the *source* is easy in this case; it is necessary to use a sampling frequency at least $2\times$ higher than the frequency of the pattern. Let us choose the perpendicular illumination and set its complex amplitude at $z = 0$ to be 1. Let us choose a sampling whose samples coincide with maxima and minima of the transmittance of the grating, i.e. the samples will be progressively $\ldots, 0, 1, 0, 1, 0, 1, \ldots$ (this is exactly at the Nyquist limit, see also the end of the section), and let us calculate the diffraction pattern using the Rayleigh-Sommerfeld integral. It is given as

$$U(x,y,z_0) = \frac{-1}{2\pi} \iint_{\Sigma} U(\xi,\eta,0) \frac{\partial}{\partial z} \frac{\exp(\mathrm{j}kr)}{r} \mathrm{d}\xi \mathrm{d}\eta \tag{1}$$

where $U(x,y,z_0)$ is the calculated complex amplitude at a point $[x,y,z_0]$ of the *target*, $U(\xi,\eta,0)$ is the complex amplitude at a point $[\xi,\eta,0]$ of the *source* (i.e. the product of the complex amplitude of incoming light and the transmittance of the grating), $\Sigma$ is the extent of the *source*, $\mathrm{j}^2 = -1$, $k = 2\pi/\lambda$ is the wave number and $r = ((x-\xi)^2 + (y-\eta)^2 + z_0^2)^{-1/2}$ is the distance between points $[x,y,z_0]$ and $[\xi,\eta,0]$.

Let us discretize the calculation by replacing the double integral with a double sum. The result shown in Fig. 1(a) differs a lot from the theoretical result. Where is the problem? (Note that Fig. 1 displays diffraction at sine grating of finite rectangular area in a finite distance, therefore the diffraction maxima have rectangular shape.)

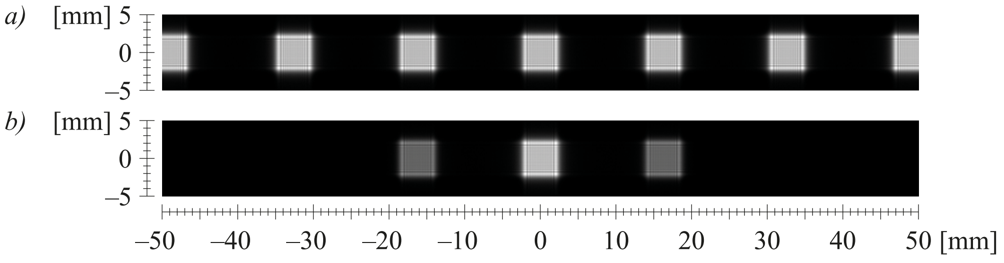


Fig. 1. Light diffraction at sine grating $5 \times 5$ mm$^2$ of period 50 cycles/mm illuminated perpendicularly by a plane wave with $\lambda = 650$ nm at a distance $z_0 = 0.5$ m. a) Discretization using $\Delta = 10$ $\mu$m. b) Discretization using $\Delta = 10/12$ $\mu$m $= 0.83$ $\mu$m.

When talking about discretization, let us discuss the direction of varying transmittance only (i.e. perpendicular to the grating stripes). The other direction is not important in this case.

The replacement of the integral by the sum means, in fact, that the original continuous function $U(\xi,\eta,0)$ is replaced by an array of Dirac pulses; in other words, by ideal point light sources arranged in a periodic lattice with spacing $\Delta$. Light originating from a lattice of sources of equal complex amplitude interferes and creates $m$-th diffraction maximum in angle $\theta_m = \arcsin m\lambda/\Delta$ (see e.g. [19]), where $m$ is an integer. In our case, every second sample is zero; that is, we are working in fact with a lattice with spacing $2\Delta$. This lattice creates diffraction maxima of equal intensities for all $m$, and the first diffraction maximum of this lattice coincides with the first diffraction maximum of the original sine grating, because its period is, thanks to sampling distance, equal to $2\Delta$. The result presented in Fig. 1a is therefore physically correct – although not for a sine grating, but for another experiment. The problem is that the discretization process did not take into account how to reconstruct the original continuous function $U(\xi,\eta,0)$ from the samples of the *source*, i.e. if there is zero transmittance between samples, if the samples represent a sine profile, a rectangular profile, etc. If we took this into account, the diffraction maxima created by the lattice would be attenuated somehow and the result would correspond with the original continuous situation.

A simple solution is easy. In the continuous situation, two close enough point light sources of the same complex amplitude do not create any interference pattern, i.e. at least one destructive interference. In the discretized situation, two point light sources of the same complex amplitude in adjacent samples can interfere destructively if the sampling distance is too big. Therefore, the sampling of the *source* has to represent the *source* correctly, and, moreover, the effect of the discretization has to be hidden – it must be such that the first-order destructive interference of adjacent point light sources created by the discretization (assume they have the same complex amplitude for now) has to lie out of the *target* area; note that in numerical calculations we are dealing with finite areas only. Mathematically, for any point $T$ of the *target* and any adjacent samples $S_1$, $S_2$ of the *source*, the inequality $|T-S_1|-|T-S_2| < \lambda/2$ must hold as the density of the samples has to resemble continuous nature of the *source*. The same idea could be expressed in terms of correct sampling of the propagation integral kernel (e.g. [2, 14, 18]), but this explanation based on point light sources is perhaps more intuitive. It is interesting to note that this simple explanation based on point light source model has not been explicitly published yet (as far as I know). Also note that advanced solutions exist that do not need finer sampling, e.g. [13, 15]; the purpose of this paragraph is to provide simple insight and a starting point for the following sections.

The result of the simple solution is presented in Fig. 1b. It was necessary to work with $12\times$ finer sampling, i.e. with an array $12^2 = 144\times$ bigger. This results in noticeably higher memory and time demands. In the following text we will present a way to discretize correctly without increasing memory demands.

At the end of the section it is worth noting that the sampling of the sine grating in the example above was not done "correctly", as the Nyquist limit (in its simplest form) requires a sampling frequency higher than double the maximum frequency contained in a signal. We have used exactly double the maximum frequency, and moreover, we did not consider that the *source* is spatially limited. However, had we used the mathematically precise method, the result would be the same and the discussion would not be as clear.

## 3. Simplified solution

In search of a memory-efficient algorithm, let us start with the solution presented in the last section, i.e. the sampling finer than requested by the Nyquist limit. For the sake of clarity, let us introduce two simplifications: let the *source* and the *target* be one-dimensional objects in the *xz* plane (this will be relaxed in Section 5) and let us suppose they are unbounded (this will be relaxed in Section 4).

Let the object *source* represented by samples *source*$[i]$, $i \in \mathbf{Z}$ (let the sample *source*$[0]$ be located at the point $[0,0]$) be perpendicularly illuminated by a plane wave of wavelength $\lambda$ (see Fig. 2). We will consider transmittance of the *source* to be complex, i.e. any *source* illuminated by any light can be converted to this scenario.

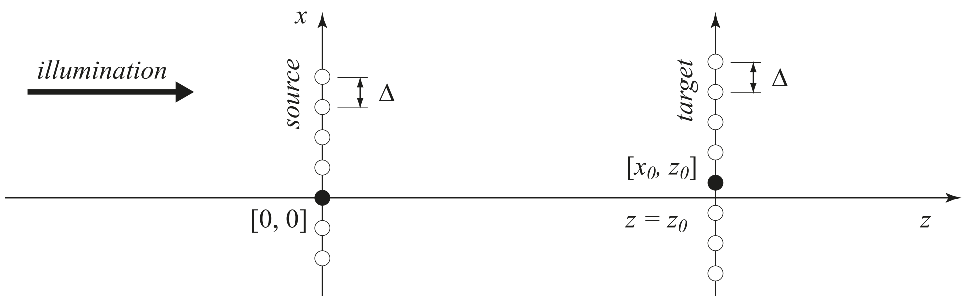


Fig. 2. Geometry of the simplified 1D case. Samples with index 0 are depicted as full circles, the others as empty circles.

Let us calculate complex amplitudes of propagated light in the "plane" $z = z_0 > 0$ in the unbounded area *target* represented by samples *target*$[j]$, $j \in \mathbf{Z}$ (let the sample *target*$[0]$ be located at the point $[x_0, z_0]$). The samples *source*$[i]$ are samples of the complex function $U(x,0)$; the samples *target*$[j]$ represent the function $U(x,z_0)$ (see Eq. (1)). If the sampling distance is equal in both the *source* and the *target*, let us call it $\Delta$, the following holds:

$$\begin{aligned} target[j] = U(j\Delta + x_0, z_0) &= \Delta \sum_{i=-\infty}^{\infty} U(i\Delta, 0)\, h\big((j-i)\Delta + x_0, 0, z_0\big) = \\ &= \Delta \sum_{i=-\infty}^{\infty} source[i]\, h_{x_0,z_0,1}[j-i] = \\ &= \Delta\big(source[\,] \otimes h_{x_0,z_0,1}[\,]\big)[j] \end{aligned} \tag{2}$$

where $\otimes$ is the discrete convolution and the array $h_{x_0,z_0,1}[\,]$ represents the Rayleigh-Sommerfeld convolution kernel (impulse response) defined as

$$h_{x_0,z_0,ups}[j] = h(j\Delta/ups + x_0, 0, z_0) \tag{3}$$

and according to (1)

$$h(x,y,z) = -\frac{1}{2\pi}\frac{\partial}{\partial z}\frac{\exp(\mathrm{j}kr)}{r} = \frac{-z}{2\pi}\left(\mathrm{j}k - \frac{1}{r}\right)\frac{\exp(\mathrm{j}kr)}{r^2}, \quad r = \sqrt{x^2+y^2+z^2} \tag{4}$$

Equation (2) is a discretization of the integral (1), i.e. we have changed the integral to the sum and the differential to the difference $\Delta$. As we will be interested just in the structure of the *target*, we will omit the constant term $\Delta$.

Let us suppose that the *source* is sampled correctly, such that the structure of $U(x,0)$ is fully acquired, but not fine enough for the propagation calculation. Please note that this "correct sampling" is not the same as sampling that obeys the sampling theorem; for example, piecewise constant signal cannot be sampled correctly according to the basic formulation of the sampling theorem as it has infinite frequency extent. However, if we know that the signal is piecewise constant with "steps" at known locations, then we can express the signal precisely using just one sample per constant part of the signal.

For the propagation calculation, we have to use a *ups*-times finer sampling, $ups \in \mathbf{Z}$, $ups \geq 1$. Let us call the preliminary upsampled array $source_{ups}[\,]$. To obtain this array, let us put $(ups-1)$ zero samples between every two original samples of the *source* (see Fig. 3 top):

$$source_{ups}[j] = \begin{cases} source[j/ups] & \text{if } (j/ups) \in \mathbf{Z} \\ 0 & \text{otherwise} \end{cases}$$

The final upsampled array $source^{fin}_{ups}[\,]$ is calculated using convolution with a kernel $filter[\,]$, i.e. $source^{fin}_{ups}[\,] = source_{ups}[\,] \otimes filter[\,]$. The convolution kernel $filter[\,]$ has to be chosen according to the nature of the function $U(x,0)$: a rectangular kernel provides a piecewise constant interpolation (which is suitable if $U(x,0)$ represents a pixelated spatial light modulator), a windowed sinc kernel provides a good interpolation in terms of frequency content (which is suitable if $U(x,0)$ is a general continuous function), a triangular kernel provides a piecewise linear interpolation (which is faster than a windowed sinc kernel and can provide acceptable results), etc. (see Fig. 3). Examples of kernel implementations will be shown at the end of Section 4.

The main idea of the method to be derived exploits the fact that the length of the $filter[\,]$ array is much smaller than the length of the array $source[\,]$ and $target[\,]$. For simplicity, let us assume the length of the $filter[\,]$ array to be odd. Let us write it as $2 \times fwh + 1$, where $fwh \in \mathbf{Z}$, $fwh \geq 0$.

Thanks to associativity of the convolution, the following holds:

$$\begin{aligned} target_{ups}[\,] &= \left(source_{ups}[\,] \otimes filter[\,]\right) \otimes h_{x_0,z_0,ups}[\,] = source_{ups}[\,] \otimes \left(filter[\,] \otimes h_{x_0,z_0,ups}[\,]\right) = \\ &= source_{ups}[\,] \otimes h^{fin}_{x_0,z_0,ups}[\,] \end{aligned}$$

where $target_{ups}[\,]$ represents the *target* sampled with a period $\Delta/ups$ and $h^{fin}_{x_0,z_0,ups}[\,]$ is the propagation kernel convolved by the array $filter[\,]$. Specifically,

$$\begin{aligned} target_{ups}[j] &= \sum_{i=-\infty}^{\infty} source_{ups}[j-i]\; h^{fin}_{x_0,z_0,ups}[i] = \\ &= \sum_{i=-\infty}^{\infty} source_{ups}[j-i] \sum_{k=-fwh}^{fwh} filter[k]\; h_{x_0,z_0,ups}[i-k] \end{aligned}$$

However, we need the final result *ups*-times downsampled, i.e. $target[j] = target_{ups}[ups \times j]$. Moreover, the sample $source_{ups}[i]$ is zero for $i/ups \notin \mathbf{Z}$. We can therefore omit these samples from the sum. It follows that

$$\begin{aligned} target[j] = target_{ups}[ups \times j] &= \sum_{i=-\infty}^{\infty} source_{ups}[ups \times (j-i)]\; h^{fin}_{x_0,z_0,ups}[ups \times i] = \\ &= \sum_{i=-\infty}^{\infty} source[j-i]\; h^{fin}_{x_0,z_0,1}[i] \end{aligned} \tag{5}$$

where $h^{fin}_{x_0,z_0,1}[\,]$ is a filtered propagation kernel:

$$h^{fin}_{x_0,z_0,1}[i] = \sum_{k=-fwh}^{fwh} filter[k]\; h_{x_0,z_0,ups}[ups \times i - k] \tag{6}$$

The propagation calculation leads to two discrete convolutions: in the first one (5), we work with sampling period $\Delta$, in the second one (6), with sampling period $\Delta/ups$.

The aforementioned equations worked with an infinite extent of the indices in order to avoid array boundary effects. In the following section, we will adjust the indices extent but the structure of the result will remain the same. The section will also explain the advantage of the presented method, which can be briefly described as follows.

The discrete convolution can be calculated indirectly using FFT or directly using the definition Eq. (2). The first way is advantageous if the convolution kernel is large; we will use it, therefore, for Eq. (5). The second way is better in the opposite case; we will therefore use it for Eq. (6). Here we will also use a nice property of direct calculation: the samples $h^{fin}_{x_0,z_0,1}[i]$ can be calculated with minimum memory demands.

## 4. Practical 1D solution

Let us assume that the *source* is sampled using $M$ samples and the *target* is sampled using $N$ samples. If we want to use FFT for the *target*[] calculation, we have to work with cyclic convolution. This means that the arrays *source*[] and *target*[] have to be zero-padded to $C$ samples, $C \geq M+N-1$ (see [17]). Then:

$$target[j] = \sum_{i=0}^{C-1} source[i \bmod C]\; h^{fin}_{x_0,z_0,1}[(j-i) \bmod C]$$

The array *source*[] contains correct values for sample indices $0,1,\ldots,M-1$, the array *target*[] for indices $0,1,\ldots,C-M$. The array $h^{fin}_{x_0,z_0,1}[\,]$ has to be then calculated as:

$$h^{fin}_{x_0,z_0,1}[i] = \begin{cases} \sum\limits_{k=-fwh}^{fwh} filter[k]\; h_{x_0,z_0,ups}[ups \times i - k] & \text{if } 0 \leq i < N \\ \sum\limits_{k=-fwh}^{fwh} filter[k]\; h_{x_0,z_0,ups}[ups \times (i-C) - k] & \text{if } N \leq i < C \end{cases}$$

It is worth noting that in the calculation of the sample $h^{fin}_{x_0,z_0,1}[i]$, it is possible to use the samples $h_{x_0,z_0,ups}[\,]$ calculated before, specifically for $h^{fin}_{x_0,z_0,1}[i-1]$. It is therefore convenient to save the samples $h_{x_0,z_0,ups}[\,]$ in a temporary buffer of size $2 \times fwh + 1$ samples, and to replace part of them in the calculation of $h^{fin}_{x_0,z_0,1}[i]$ with new values. The number of these new samples depends on the *filter* type.

To calculate the *filter*[] array, we have to choose the number of samples of the *source*[] array that contribute to the calculation of the interpolated sample in the $source^{fin}[\,]$ array, or in other words, in the $h^{fin}_{x_0,z_0,1}[i]$ array. This user-defined number specifies all the parameters needed: the size of the *filter*[] array, the interpolation type, and the number of the samples shared in the calculation of the neighbouring samples of the $h^{fin}_{x_0,z_0,1}[\,]$ array.

It is practical to use separable kernels when working with 2D arrays. We can discuss them right now, when working with 1D arrays. For clarity, some examples are given in Fig. 3. To make things simpler, we will show pure interpolation kernels in both the Fig. 3 and the following text, i.e. they do not preserve signal energy. For propagation calculation it is, however, appropriate to normalize the filter, i.e. the sum of its coefficients equals 1.

**Piecewise constant interpolation.** It is suitable if the *source* is split to rectangular pixels of non-zero area. In this case it is appropriate for *ups* to be odd, due to symmetry. Then the interpolation process adds an even number of samples between every two samples of the *source*[] array. Therefore, $fwh = (ups-1)/2$ and the kernel is given as: $filter[i] = 1$ for $-fwh \leq i \leq fwh$.

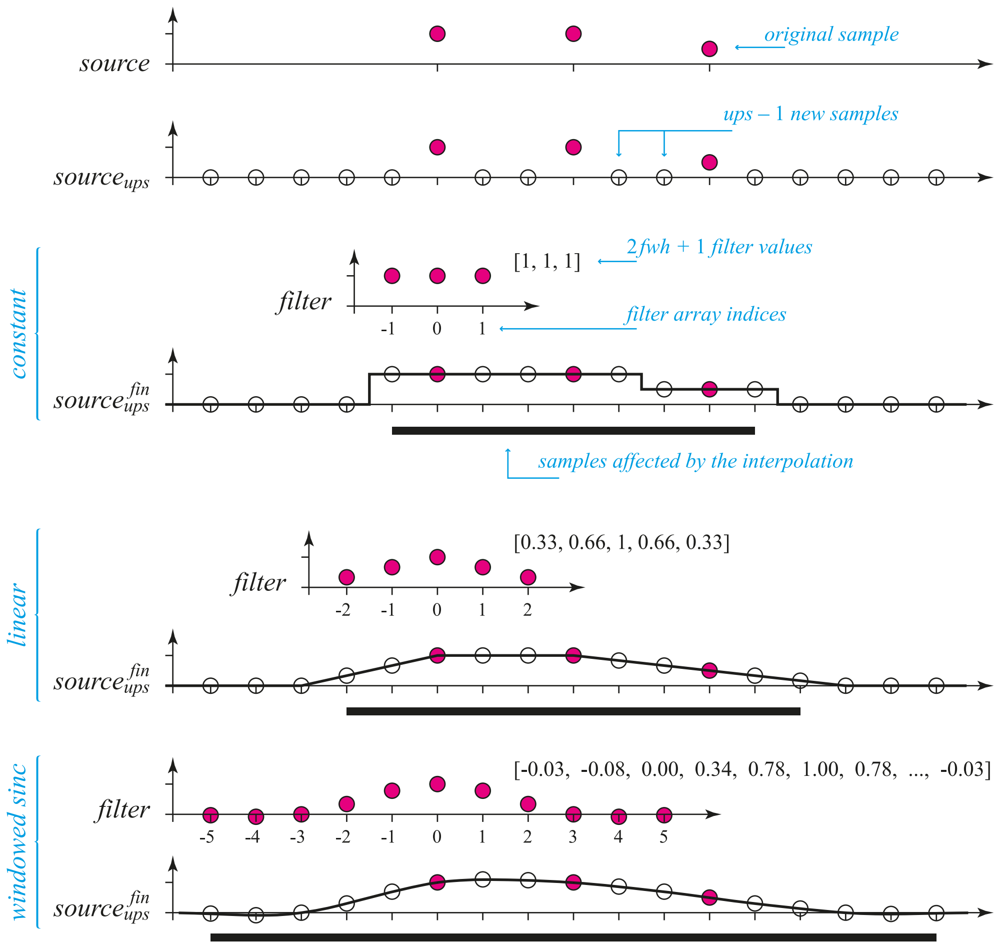


Fig. 3. Examples of interpolation convolution kernels (filters) for $ups = 3$. The windowed sinc filter shown is the normalized Lanczos filter for $a = 2$.

**Piecewise linear interpolation.** It is suitable if the *source* represents a continuous function and it does not contain fine details. We need two neighbouring original samples for the calculation of the interpolated one, i.e. $fwh = ups - 1$, $filter[i] = 1 - |i|/(fwh+1)$ for $-fwh \leq i \leq fwh$.

**Windowed sinc interpolation.** It is suitable if the *source* represents a continuous function and we care about good replication of its frequency content. Choice of the number of the original *source*[] samples has to be a compromise. The more samples are included, the better is the frequency content replication; on the other hand, too large kernels perform badly in the spatial domain. A Lanczos filter is considered a reasonable compromise. It considers $2a$ neighbouring samples, where usually $a = 2$ or $a = 3$ [20]. Then $fwh = a \times ups - 1$ and a preliminary kernel is defined as $filter_{prel}[i] = \text{lanczos}(a \times i/(fwh+1), a)$, where $\text{lanczos}(x,a) = a\sin(\pi x)\sin(\pi x/a)/(\pi^2 x^2)$ for $-fwh \leq i \leq fwh$. The final kernel has to be adjusted before use: the coefficients contributing to the same sample calculation, i.e. the coefficients *ups* samples apart, have to sum to 1 [21]. Mathematically, $filter[i] = filter_{prel}[i]/\sum_k filter_{prel}[i + k \times ups]$ for all allowed values of $k$.

## 5. Final 2D solution

Generalization of the aforementioned ideas is straightforward; instead of 1D cyclic convolutions (or FFT's), we have to use 2D versions. For simplicity, let us assume the sampling periods in both $x$ and $y$ directions to be the same, let us call them $\Delta$.

To propagate the *source* sampled by $M_x \times M_y$ samples to the *target* sampled by $N_x \times N_y$ samples, let us create the arrays $source[,]$ and $target[,]$ of size $C_x \times C_y$, $C_x \geq M_x + N_x - 1$, $C_y \geq M_y + N_y - 1$. It is convenient to choose the numbers $C_x$ and $C_y$ so that the FFT of these arrays runs fast, e.g. powers of 2. The samples of the *source* must be stored in the array $source[,]$ at indices from $[0,0]$ to $[M_x - 1, M_y - 1]$. After the calculation, the correct samples are located in the array $target[,]$ at indices from $[0,0]$ to $[N_x - 1, N_y - 1]$.

As the next step, we have to choose the parameter *ups*. The way to do it is as follows. Let us assume for a while that we work with the original lattice, i.e. $ups = 1$. In the propagation calculation, we have to calculate (4) for every vector $T - S$, where $S$ is a 3D position of a sample in the *source* and $T$ is a 3D position of a sample in the *target*. In (4) we have to use $r = |T - S|$ (see its application in (2)). Let $S_1$ and $S_2$ be positions of adjacent samples in the *source* and these samples have the same value; they represent two point light sources of the same complex amplitude. Let us calculate $r_1 = |T - S_1|$ and $r_2 = |T - S_2|$. If $|r_1 - r_2| = \lambda/2$, then the contributions from $S_1$ and $S_2$ cancel each other at the point $T$, i.e. in this direction there is the first diffraction minimum. As we have shown in Section 2, we need to exclude the first diffraction minimum from the *target*. To make this happen, we have to refine the sampling, i.e. increase the parameter *ups* until $|r_1 - r_2| < \lambda/2$. If a more precise result is needed, we can refine further. Numerical experiments have shown that higher *ups* than those leading to $|r_1 - r_2| < \lambda/5$ did not have any significant impact. The adjacent points $S_1$, $S_2$ and the point $T$ have to be chosen as "the worst case", i.e. the angle between $T - S_1$ (or $T - S_2$) and the $z$ axis has to be as big as possible.

As the next step, we have to calculate the array $h^{fin}_{x_0,y_0,z_0,1}[,]$, where $[x_0, y_0, z_0]$ is the position of the sample $target[0,0]$. We assume the position of the sample $source[0,0]$ to be $[0,0,0]$. For the calculation, we need the numbers $h_{x_0,y_0,z_0,ups}[i,j] = h(i\Delta/ups + x_0, j\Delta/ups + y_0, z_0)$ (see (3) for comparison). Thanks to separability of the filters, we can calculate them for one upsampled row only, convolve them with $filter[]$ and downsample, i.e. to use the procedure described in Section 4. We need to calculate $2 \times fwh + 1$ of such rows, convolve them by columns and downsample; this procedure leads to one row of the array $h^{fin}_{x_0,y_0,z_0,1}[,]$. The other rows are calculated in the same way.

Finally, we can calculate the propagation itself:

$$target[,] = \mathbf{IFFT}\big(\mathbf{FFT}(source[,]) \odot \mathbf{FFT}(h^{fin}_{x_0,y_0,z_0,1}[,])\big)$$

where **FFT** and **IFFT** are fast Fourier transform and inverse fast Fourier transform, respectively, and $\odot$ is the Hadamard product (element-wise product). The propagation calculation is complete now. Examples of the propagations are shown in Fig. 4.

It is worth adding three remarks:

- The same result can be obtained using the basic method, i.e. using upsampling, interpolation of the *source*, and convolution with a common Rayleigh-Sommerfeld propagation kernel. The upsampled array $source_{ups}[,]$ would have $ups - 1$ zero samples between original samples in every row (column), and additionally *fwh* zero samples to the sides in order to correctly calculate the convolution with an interpolation kernel $filter[,]$ of size $(2 \times fwh + 1) \times (2 \times fwh + 1)$ samples. The size of the array $source_{ups}$ would then be $\big(2 \times fwh + 1 + ups \times (M_x - 1)\big) \times \big(2 \times fwh + 1 + ups \times (M_y - 1)\big)$ samples, sampling period $\Delta/ups$. This means that the physical size of the *source* increases a bit for $fwh \geq 1$;

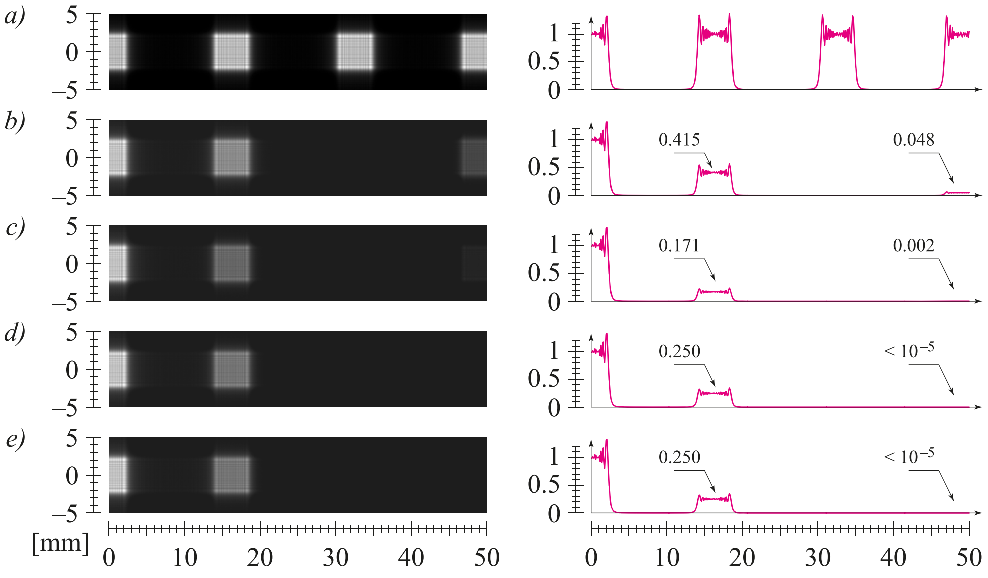


Fig. 4. Examples of diffraction by grating with vertical strips; grating size $5 \times 5$ mm$^2$, sampling period 10 $\mu$m, samples in each row $1, 0, 1, 0, \ldots$ (i.e. 50 slits/mm). Propagation distance 500 mm, illumination at normal incidence, $\lambda = 650$ nm. The left images show right halves of the diffraction patterns (compare with Fig. 1); the graphs to the right show the intensity relative to the central intensity. The interpolation used is a) none, b) rectangular filter, c) triangular filter, d) Lanczos filter, $a = 2$, e) Lanczos filter, $a = 3$.

this is the reason why the article never mentions the exact physical sizes of the *source* and the *target*. The additional borders are an interpolation artifact. However, their effect is negligible for big arrays.

- The proposed method with propagation kernel filtering is nothing else than a calculation rearrangement. The article [17] that describes the propagation calculation with large *source* and *target* or with different sampling periods of *source* and *target* is therefore fully compatible with the proposed method; is is sufficient to replace the calculation of propagation kernels.

- The calculation rearrangement leads to different rounding errors in numerical calculation and thus to differences between the basic and the proposed method. The differences are negligible, though. As it is hard to tell which method gives a more precise result, we will not discuss the numerical aspects of the proposed method.

## 6. Time and memory requirements

The motivation to create the proposed method was to calculate reference propagation using a small amount of memory. Let us compare its memory and time demands with the basic method. In this section, we will assume square arrays for simplicity, i.e. $M_x = M_y = M$, $N_x = N_y = N$.

The basic method, as we have shown in the last section, upsamples the array *source*$[,]$ to the array *source*$_{ups}$ with $(2 \times fwh + 1 + ups \times (M-1))^2$ samples and calculates propagation to the array *target*$_{ups}$. There is no need to introduce additional zero borders due to interpolation, i.e. the *target* will have $(1 + ups \times (N-1))^2$ samples. The propagation will be calculated in the arrays with $\left(2\times fwh+1+ups\times(M+N-2)\right)^2$ samples, which gives the memory requirements of the basic method.

The proposed method uses the original arrays *source*[,] and *target*[,], so the propagation will be calculated in arrays with $(M+N-1)^2$ samples. The memory demands are, however, a bit higher, as we have to take account of temporary memory used in the calculation of the filtered propagation kernel $h^{fin}_{x_0,y_0,z_0,1}[,]$. It is $2\times fwh+1$ samples for "row convolution" and $2\times fwh+1$ rows with $M+N-1$ samples for "column convolution"; together $(2\times fwh+1)(M+N)$ samples. Typically, $fwh=a\times ups$, where $a$ is small (in our examples at most 3), and *ups* is much smaller than $M+N$. Therefore, it is possible to ignore this amount in further discussion.

By comparing the memory demands, we conclude that the propagation calculation using the proposed method takes approximately $\left(2\times fwh+1+ups\times(M_x+N_x-2)\right)^2/(M+N-1)^2\approx ups^2$-times less memory. Common experiments in computer generated holography with centimetre-sized fields using a sampling period of about 10 $\mu$m off-axis propagated to a distance in the order of tens of centimetres require the *ups* parameter up to 20. We can therefore say that the proposed method has about 100-times less memory demands than the basic method.

On the other hand, the time of the computation does not fall so quickly. This is because the calculation consists not just of a convolution calculation using the FFT, which is very fast in the proposed method, but of a convolution kernel calculation as well that is approximately as slow as in the basic method. More precisely, the FFT works with arrays approximately $ups^2$-times smaller than in the basic method, which means it is approximately $ups^2$-times faster. The convolution kernel calculation requires calculating the array $h_{x_0,y_0,z_0,ups}[,]$ (the same as in the basic method), filtering it by rows and by columns using a filter of width $2\times fwh+1$, where again $fwh=a\times ups$, and downsampling the result. The analysis shows that the convolution kernel calculation in the proposed method is approximately $4a^2$-times slower than in the basic method, where $a$ is again a small number. It is not worth making a precise analysis, as the speed of the FFT, the Rayleigh-Sommerfeld convolution kernel and its filtering are hard to compare theoretically. It is more useful to measure the calculation times (see Fig. 5). The graph to the right shows that the speedup of the proposed method is not as big as its memory savings; this is mainly because the calculation of the convolution kernel dominates in the total time of the calculation.

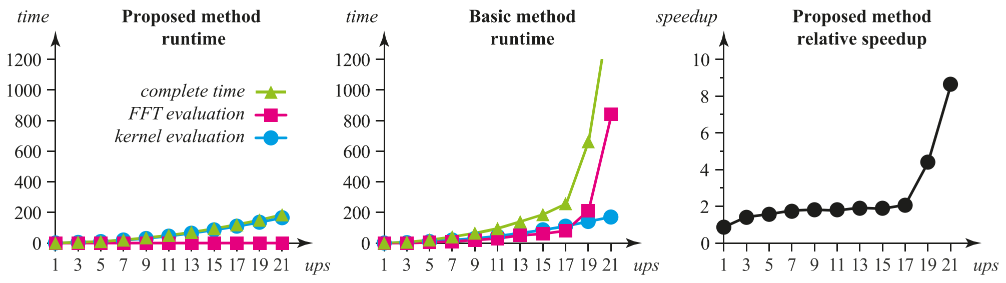


Fig. 5. Time of the calculation comparison. The graphs to the left and in the middle show the dependency of the time of propagation calculation for $N=M=500$ on the upsampling factor *ups*; the vertical scale used defines the time of the basic method for $ups=1$ to be 1. Besides the complete time of the calculation, the times of the FFTs and the propagation kernel calculation times are shown. The rightmost graph shows the ratio of the proposed and the basic method calculation times; e.g. a value of 4 means that the proposed method is 4× faster for a given *ups*.

## 7. Conclusions

In Section 2 we have shown and explained in terms of physics that in the discretization of the light propagation between parallel planes it is necessary to take into account both the correct sampling of the *source* and the opposite procedure, the reconstruction. We have shown that the correct result can be obtained using upsampling; we have shown in Section 5 how fine this upsampling should be. We have demonstrated in Section 6 that in typical tasks in computer generated holography the upsampling can be approximately $10\times$, which leads to $100\times$ slower calculation and $100\times$ bigger memory demands.

In Sections 3 to 5 we have derived a procedure based on filtration of the propagation kernel by the “interpolation” kernel that is used for interpolated upsampling of the *source*. Thanks to the properties of the interpolation kernel (small support, separability), the proposed method is faster and cuts the memory demands to approximately those values which would be needed if no upsampling was used. It can thus be said that the proposed method has approximately $100\times$ smaller memory demands than the basic method of the same precision. As the method just rearranges the calculation, the result is mathematically the same.

## Acknowledgments

This work has been supported by the Ministry of Education, Youth and Sports of the Czech Republic projects LC06008 and LH12181. The author would like to thank Václav Skala for discussions concerning the subject.

# Chapter 5

# Filtering in light propagation calculations – the angular spectrum method

This chapter presents the article *Discrete calculation of the off-axis angular spectrum based light propagation* [36] presented at the 9th International Symposium on Display Holography (ISDH 2012), 25–29 June 2012, MIT Media Lab, Cambridge Massachusetts USA; it starts on page 88.

Until now, we have discussed just the convolution method of light propagation calculation between parallel planes. We have proved that careful discretisation of the formulation in the continuous domain leads to correct results for any propagation distance. However, calculation of the convolution kernel for very small propagation distances is time consuming as it is necessary to properly filter highly oscillatory function $K_{RS}(x, y, z_0)$, see below. On the other hand, long propagation distances do not require full precision of the Rayleigh-Sommerfeld diffraction formula (1.1). Approximations such as the Fresnel approximation (see [6, Section 4.2]) often provide good enough results, and can be calculated more efficiently, see for example [12].

Thus, *the angular spectrum decomposition* method is usually preferred for small propagation distances.

For convenience, let us repeat that the convolution method describes light propagation as convolution (see Sections 3.1 and 3.2)

$$\begin{aligned} U(x, y, z_0) &= U(x, y, 0) \otimes K_{RS}(x, y, z_0) \\ &= \mathcal{F}^{-1}\Big\{\mathcal{F}\{U(x, y, 0)\}\mathcal{F}\{K_{RS}(x, y, z_0)\}\Big\}, \end{aligned}$$

where

$$\begin{aligned} K_{RS}(x, y, z_0) &= -\frac{1}{2\pi}\left(\mathbf{j}\,k - \frac{1}{r}\right)\frac{\exp(\mathbf{j}\,kr)}{r}\frac{z_0}{r} \\ r &= \sqrt{x^2 + y^2 + z_0^2}. \end{aligned}$$

The angular spectrum decomposition method is based on the fact that the Fourier transform of the function $K_{RS}(x, y, z_0)$ is known in the closed form:

$$\mathcal{F}\{K_{RS}(x, y, z_0)\} = H_{RS}(f_x, f_y, z_0) = \exp\left(\mathbf{j}\, k z_0 \sqrt{1 - \lambda^2 f_x^2 - \lambda^2 f_y^2}\right). \tag{5.1}$$

Please note that in signal processing, the function $K_{RS}$ would be called *the impulse response* and the function $H_{RS}$ *the transfer function.*

It is then tempting to discretise the calculation to

$$\mathit{target}[\,] = \mathbf{IDFT}\Big\{\mathbf{DFT}\{\mathit{source}[\,]\} \odot H_{RS}[\,]\Big\}, \tag{5.2}$$

where $\odot$ is the Haramard (element-wise) product, *source*[ ] is a discrete signal representing a finite area in the plane $z = 0$

$$\mathit{source}[m, n] = U(m\Delta_x, n\Delta_y, 0),$$

for $0 \le m \le M - 1$, $0 \le n \le N - 1$, $\Delta_x$ and $\Delta_y$ are the sampling distances in the spatial domain. The discrete signal $H_{RS}[\,]$ contains samples of the transfer function $H_{RS}(f_x, f_y, z_0)$, i.e.

$$H_{RS}[m, n] = H_{RS}(m\Delta_{f_x}, n\Delta_{f_y}, z_0)$$

for suitable range of indices $m$, $n$, where $\Delta_{f_x}$ and $\Delta_{f_y}$ are appropriately chosen sampling distances in the frequency domain. The conventional wisdom says that the discrete signal *target* contains samples of light field in the plane $z = z_0$, i.e.

$$\mathit{target}[m, n] \overset{?}{=} U(m\Delta_x, n\Delta_y, z_0).$$

We will refer the method of light propagation described by Equation (5.2) as *the plain angular spectrum decomposition method.* Let us inspect it closer.

First of all, we should notice that the transfer function $H_{RS}(\,)$ is radially symmetric and oscillatory – the higher the propagation distance $z_0$, the faster oscillations. In other words, a small propagation distance leads to a function that oscillates slowly, i.e. it is less prone to aliasing. This makes us believe that the discrete calculation should behave well for small propagation distances.

On the other hand, we have seen in Chapter 4 that the discrete calculation should be described as

$$\mathit{target}_{ext}[\,] = \mathbf{IDFT}\Big\{\mathbf{DFT}\{\mathit{source}_{pad}[\,]\} \odot \mathbf{DFT}\{K_{RS}^{fil}[\,]\}\Big\},$$

where $\mathit{source}_{pad}[\,]$ is a properly zero-padded discrete signal *source*[ ], $K_{RS}^{fil}[\,]$ contains samples of a properly filtered impulse response $K_{RS}(x, y, z_0)$ and $\mathit{target}_{ext}[\,]$ is a discrete signal, whose part contains samples of the function $U(x, y, z_0)$.

We should therefore ask: should we use zero padding in the plain angular spectrum decomposition method as well? How many zero samples should we add? Should we assume

$$\mathbf{DFT}\{K_{RS}^{fil}[\,]\} \overset{?}{=} H_{RS}[\,]\ ?$$

What shall we do if there are aliasing artifacts in the signal $H_{RS}[\,]$?

The article *Discrete calculation of the off-axis angular spectrum based light propagation*, included in this section from page 88, analyses the angular spectrum decomposition method thoroughly and gives answers.

As the article is not easy to follow (because the reasoning is tricky), I included the supplementary material to the article – an interactive tutorial. There I provided many examples of light propagation calculation – intensity images of propagation results, convolution kernels in both intensity and phasor visualizations (in both spatial and frequency domains). In total, it contains 10 short descriptions (about a paragraph each) and about 150 images. The reader is encouraged to download it from the publisher's website (see `https://dx.doi.org/10.1088/1742-6596/415/1/012040`). For convenience, I selected the most instructive images and included them on following pages.

Please follow the texts below Figures 5.1–5.15. They should illustrate problems that are discussed in the article starting on page 88.

### Selected images from the supplementary tutorial, part one

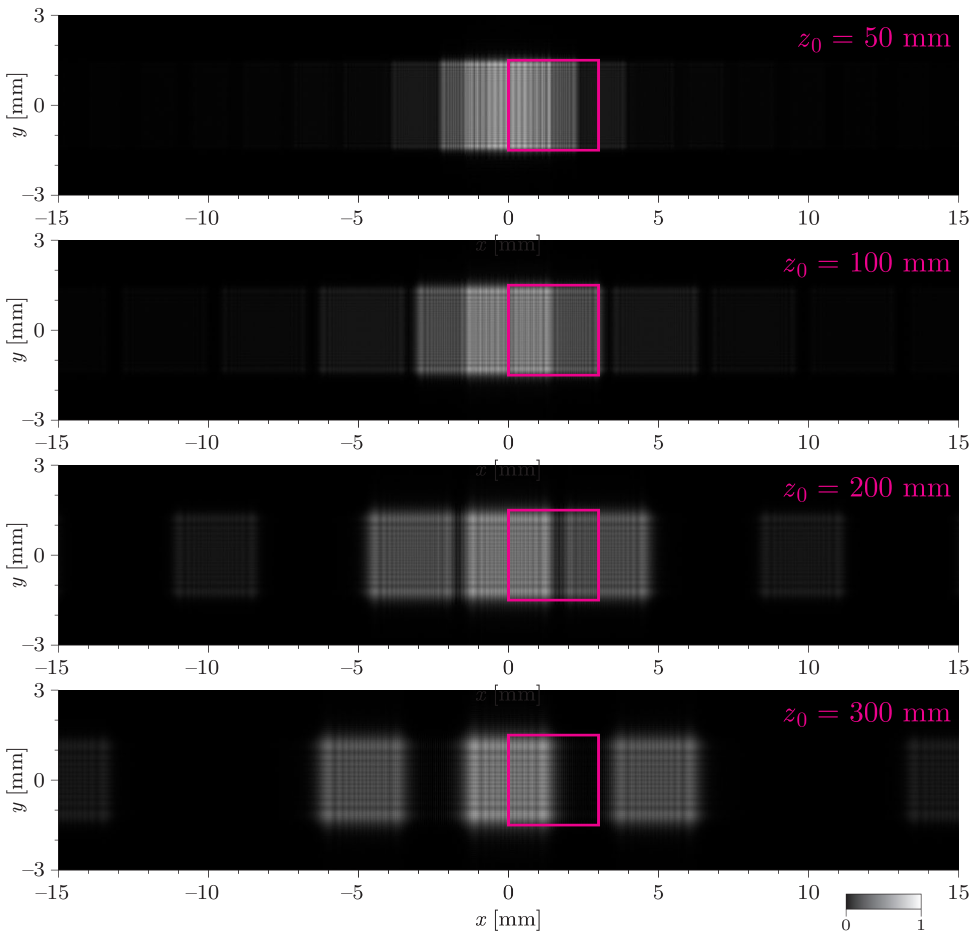
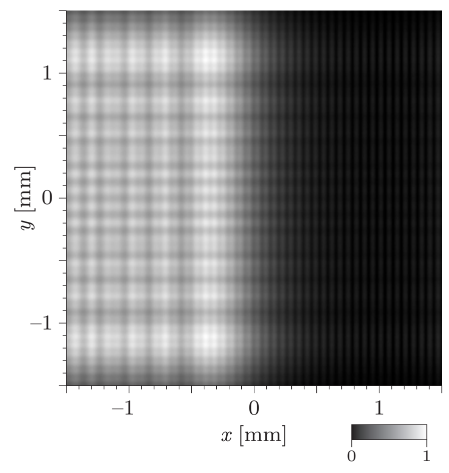


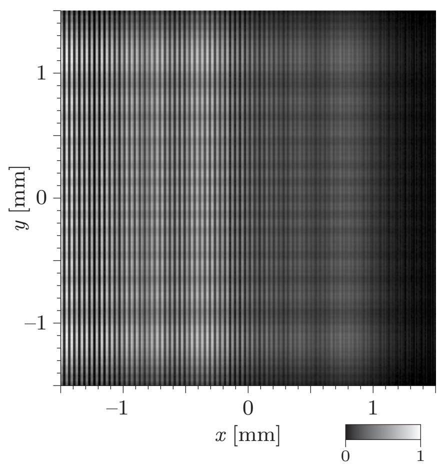
Figure 5.1: *Propagation to various distances (normalized intensity images).*

In the tutorial, we will study light (plane wave, $\lambda = 650$ nm) diffracted at the amplitude diffraction grating ($3 \times 3$ mm$^2$) composed of opaque and transparent vertical stripes 20 $\mu$m wide (i.e period is 40 $\mu$m). The diffraction patterns at distances 50 mm, 100 mm, 200 mm and 300 mm distance should look like these images. We will be interested in just a small area ($3 \times 3$ mm$^2$) depicted with a magenta box.

In the first part of the tutorial, we will study propagation to the distance 300 mm only. In the second part, we will study propagation to the distance 50 mm only. To make the images of the second part easier to analyse, the period of the diffraction grating was changed to 20 $\mu$m. Thus, the propagation looks approximately like the image above for $z_0 = 100$ mm.



*filtered convolution (correct)*



*plain angular spectrum decomp.*

Figure 5.2: *Propagation to* $z_0 = 300$ mm *(normalized intensity images).*

Angular spectrum decomposition is mathematically equivalent to the convolution with the Rayleigh-Sommerfeld convolution kernel. However, a straightforward implementation (we call it *the plain angular spectrum decomposition* here) provides incorrect results, especially in larger distances. Please note that while the reference (correct) image above is described as *the filtered convolution*, see Chapter 4, there was no filtering involved as the convolution kernel was well sampled. To see why are the results different, we should inspect the convolution kernels and the transfer functions, see Figures 5.3 and 5.4.

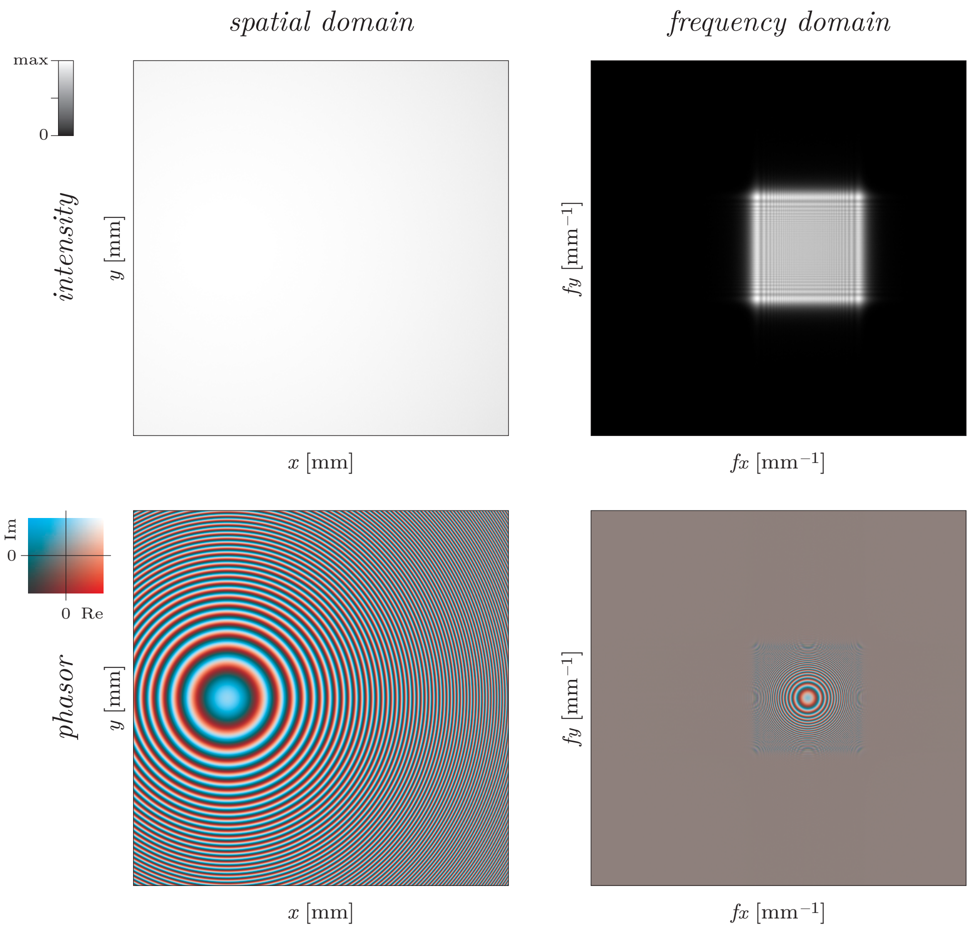


Figure 5.3: *Kernel of the filtered convolution method (reference),* $z_0 = 300$ mm.

These images show the convolution kernel used for the Figure 5.2 left, i.e. for the filtered convolution method. For convenience, it is shown as both intensity (top left) and complex phasor (bottom left) image. The right column shows the discrete Fourier transform of the kernel, both as squared absolute value ("intensity") and complex phasor visualisations. Please note there are no values on the axes as they are not important here.

First of all, we should note that the kernel is well sampled in the spatial domain (bottom left) as we do not see any aliasing artifacts. This is verified in the visualisation of the Fourier transform – the top right image shows that the kernel is frequency limited.

As the convolution kernel is correct, any other method of propagation calculation should work with similar functions, either in the spatial or the frequency domain. Please also note that the intensity of the convolution kernel in the spatial domain (top left) is nearly flat. This is expected: if a point light source located at the origin illuminates an area $3 \times 3$ mm$^2$ at a distance 300 mm, the intensity should be nearly constant.

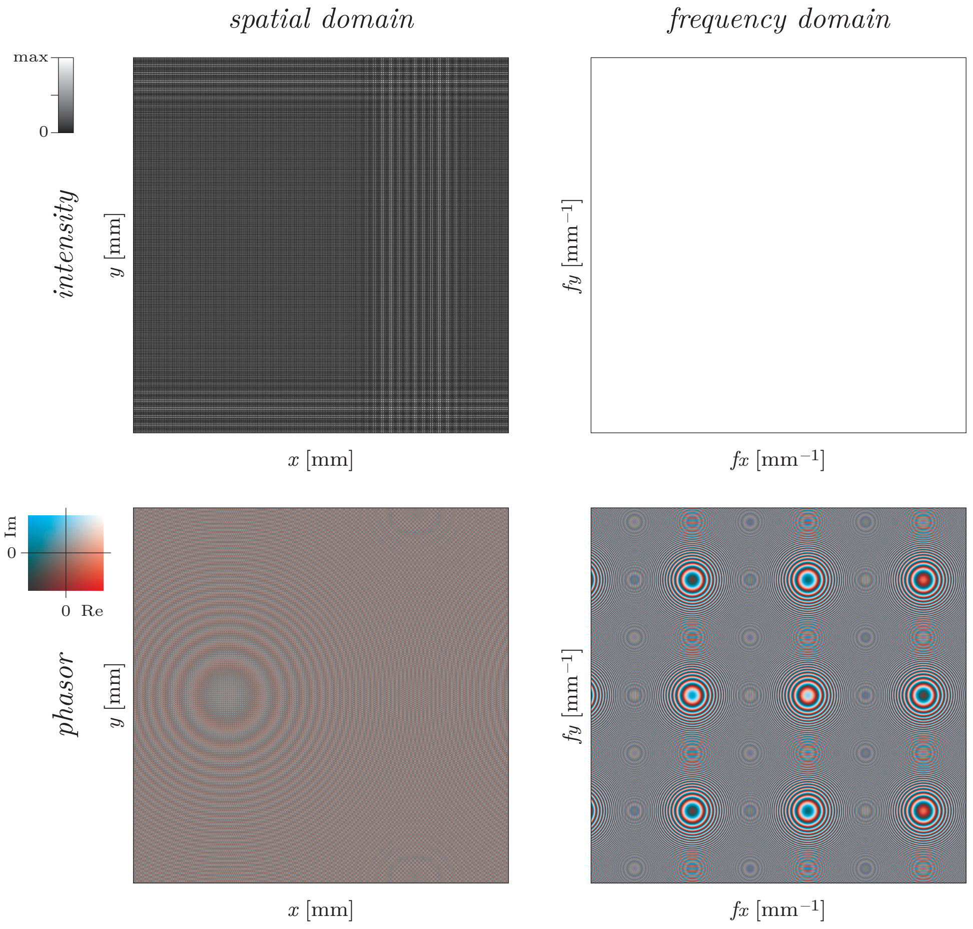


Figure 5.4: *Kernel of the plain angular spectrum method,* $z_0 = 300$ mm.

These images show the functions used for the Figure 5.2 right, i.e. for the plain angular spectrum decomposition method. Recall that in the angular spectrum decomposition methods we calculate the transfer function directly, i.e. the Fourier transform of the impulse response ("the convolution kernel in the spatial domain").

First of all, we should note that the transfer function (right column) is not well sampled as we see aliasing artifacts, see bottom right image – recall that the transfer function should look like a set of concentric circles. The top right image is perfectly flat, indeed, as $|H_{RS}(f_x, f_y, z_0)| = 1$ for $\lambda^2(f_x^2 + f_y^2) \leq 1$, which is satisfied here.

It is then no surprise that the inverse discrete Fourier transform of the transfer function (the left column) looks completely different from the correct convolution kernel shown in Figure 5.3.

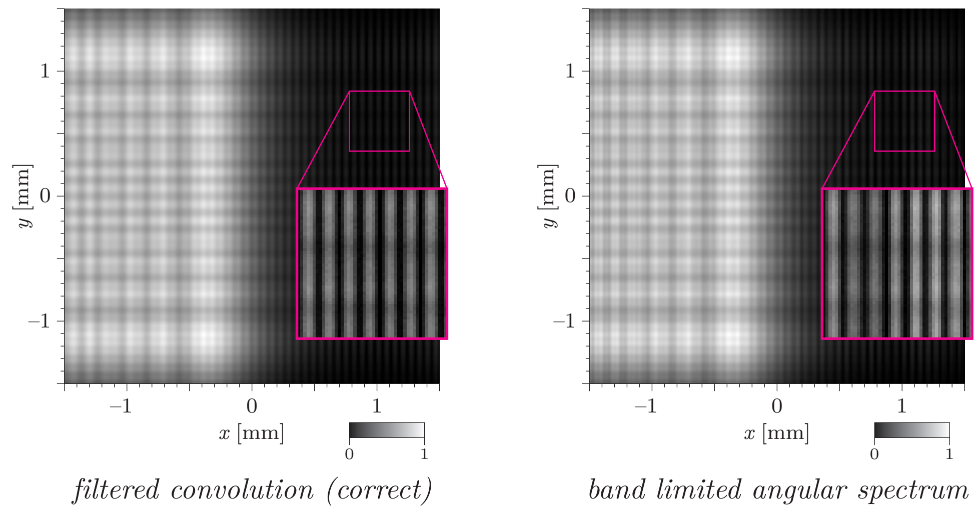


Figure 5.5: *Propagation to* $z_0 = 300$ mm *(normalized intensity images).*

A popular improvement of the plain angular spectrum metod, called *the band-limited angular spectrum method*, was proposed by Matsushima and Shimobaba [20]. It simply evaluates the local frequency of the transfer function and makes the transfer function zero if the local frequency exceeds the sampling frequency, see Figure 5.6.

The methods usually gives surprisingly good results. These two images are virtually the same, the only slight difference is hardly visible near the right edge (please note that contrast of the insets was adjusted). We should, however, ask: what are consequences of band limitation?

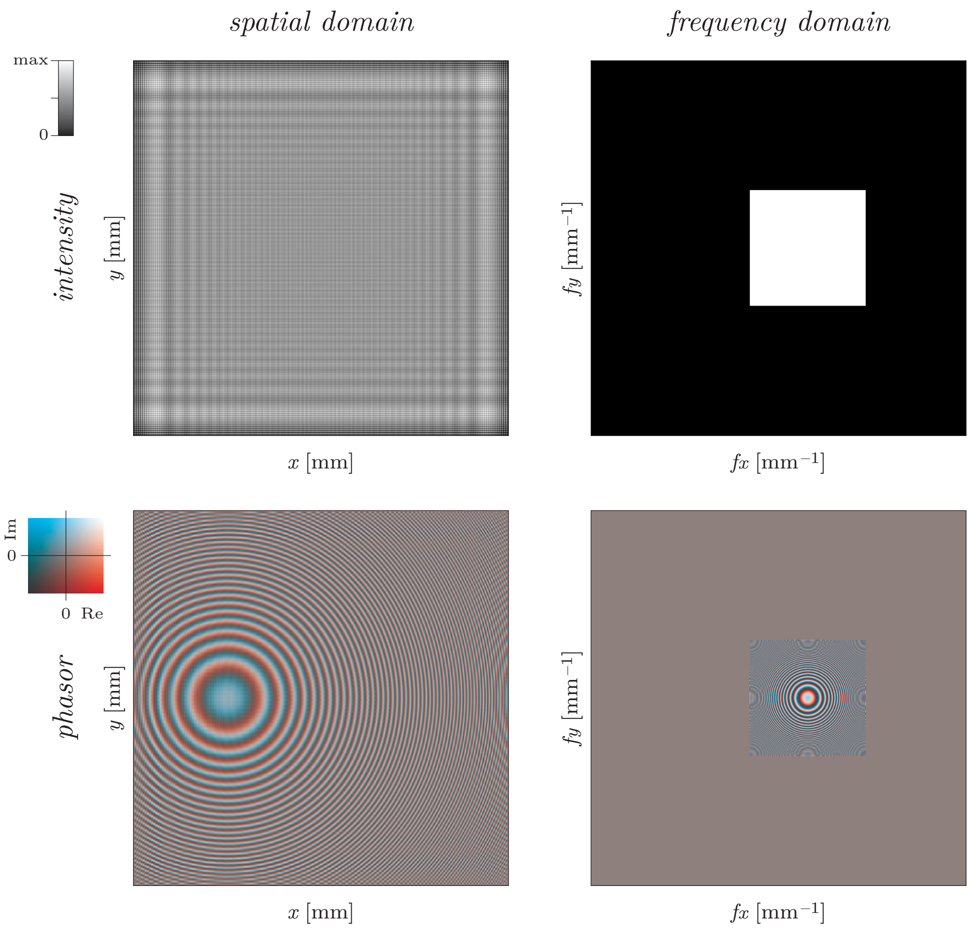


Figure 5.6: *Kernel of the band-limited angular spectrum method,* $z_0 = 300$ mm*.*

These images show the functions used for the Figure 5.5 right, i.e. for the band-limited angular spectrum method. The right column (the transfer functions) shows how the band limitation works. At every point, the local frequency is evaluated. If it is too high, the altered transfer function is set to zero.

We say that a function $g(x)$ is spatially limited in the spatial domain if there exists $a \in \mathbb{R}$ such that $g(x) = 0$ for $|x| > a$. We say that the function $g(x)$ is frequency limited if its Fourier transform is spatially limited in the frequency domain, i.e. there exists $A \in \mathbb{R}$ such that $\mathcal{F}\{g(x)\} = G(f_x) = 0$ for $|f_x| > A$. Similarly, we say that the function $G(f_x)$ is frequency limited if its inverse Fourier transform is spatially limited in the spatial domain.

The analysis in the article that follows shows that the correct convolution kernel should be spatially limited in the spatial domain, not in the frequency domain, which is the case of the band-limited angular spectrum method. Compare these images with Figure 5.3 to see how the correct convolution kernel should look like. As the kernels in the spatial domain look very different (check the intensity image!), it is clear that it is possible to construct such an input field so that the propagation result will be visibly wrong.

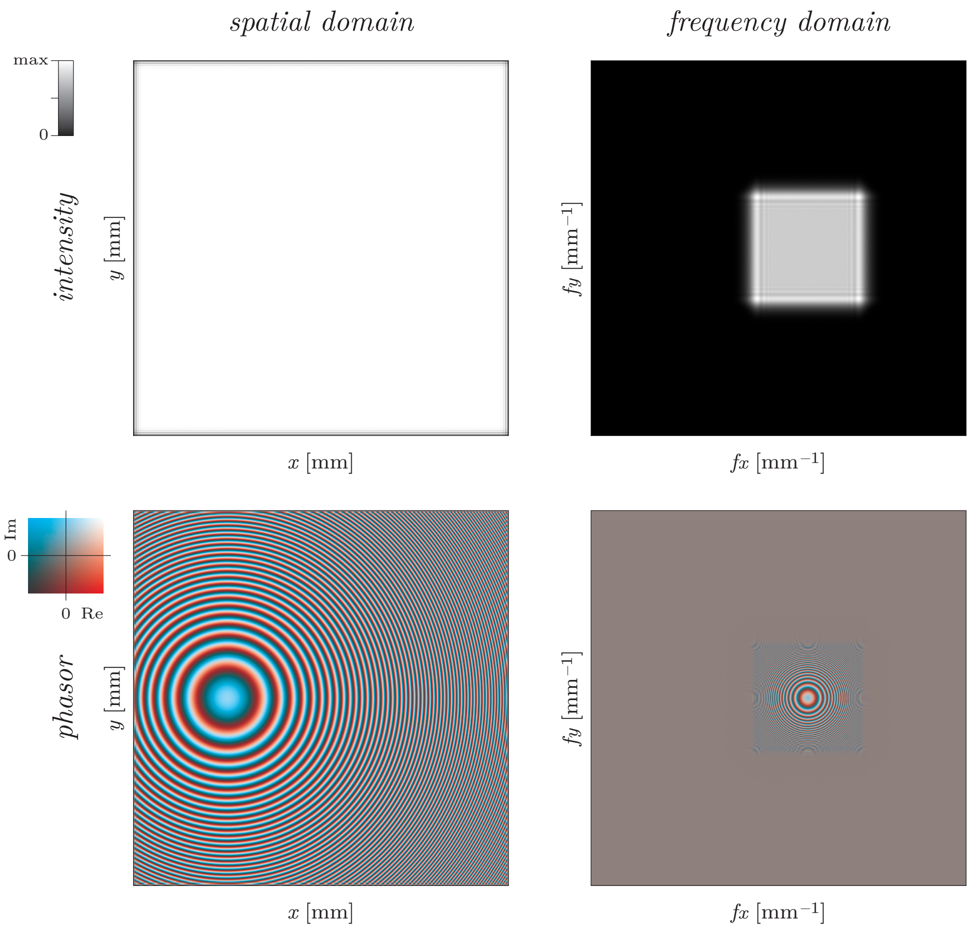


Figure 5.7: *Kernel of the filtered (101-tap windowed sinc) angular spectrum method,* $z_0 = 300$ mm*.*

To calculate the precise kernel in the frequency domain, it is necessary to incorporate its spatial limitation in the spatial domain somehow, as discussed in the following article. This can be done using digital low-pass filtering (frequency limiting in the frequency domain means spatial limiting in the spatial domain). Compare these images with Figures 5.3 and 5.6. The kernel in the frequency domain looks almost correct now! It is not perfect, of course, as the digital filter used is not perfect as well. (And moreover, the filter in this method is a 101-tap windowed sinc finite impulse response filter, which is quite slow.)

Compare also (intensity) pictures of the kernel in the spatial domain to the reference method in Figure 5.3. It should be clearly seen how the spatial limitation works: in the reference method, the kernel is sharply limited by the image boundary, in the proposed method the spatial limitation manifests itself as the decay near the image boundary.

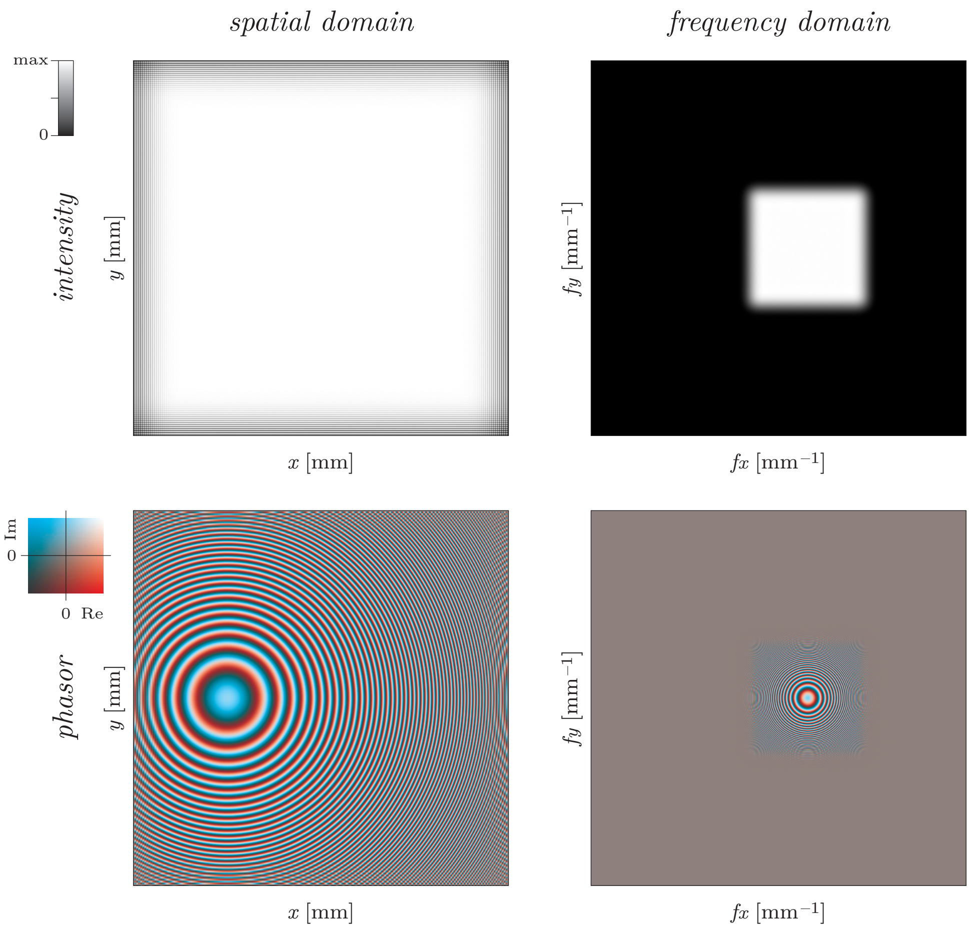


Figure 5.8: *Kernel of the filtered (21-tap windowed sinc) angular spectrum method,* $z_0 = 300$ mm.

It is also instructive to see what what happens if we use shorter (21-tap finite impulse response) digital low-pass filter to calculate the kernel in the frequency domain. As expected, the boundary of non-zero area in the frequency domain is not as wavy as in the reference (Figure 5.3) or 101-tap method (Figure 5.7). This weaker frequency limitation manifests itself as more gradual spatial limitation, see the intensity picture of the kernel in the spatial domain (top left). However, the inside of the image looks good. This can be especially seen in the complex phasor picture (bottom left). It follows that to avoid boundary errors, it is possible to extend the arrays used in the calculation somehow so that the boundary errors will not be visible in area of the final result.

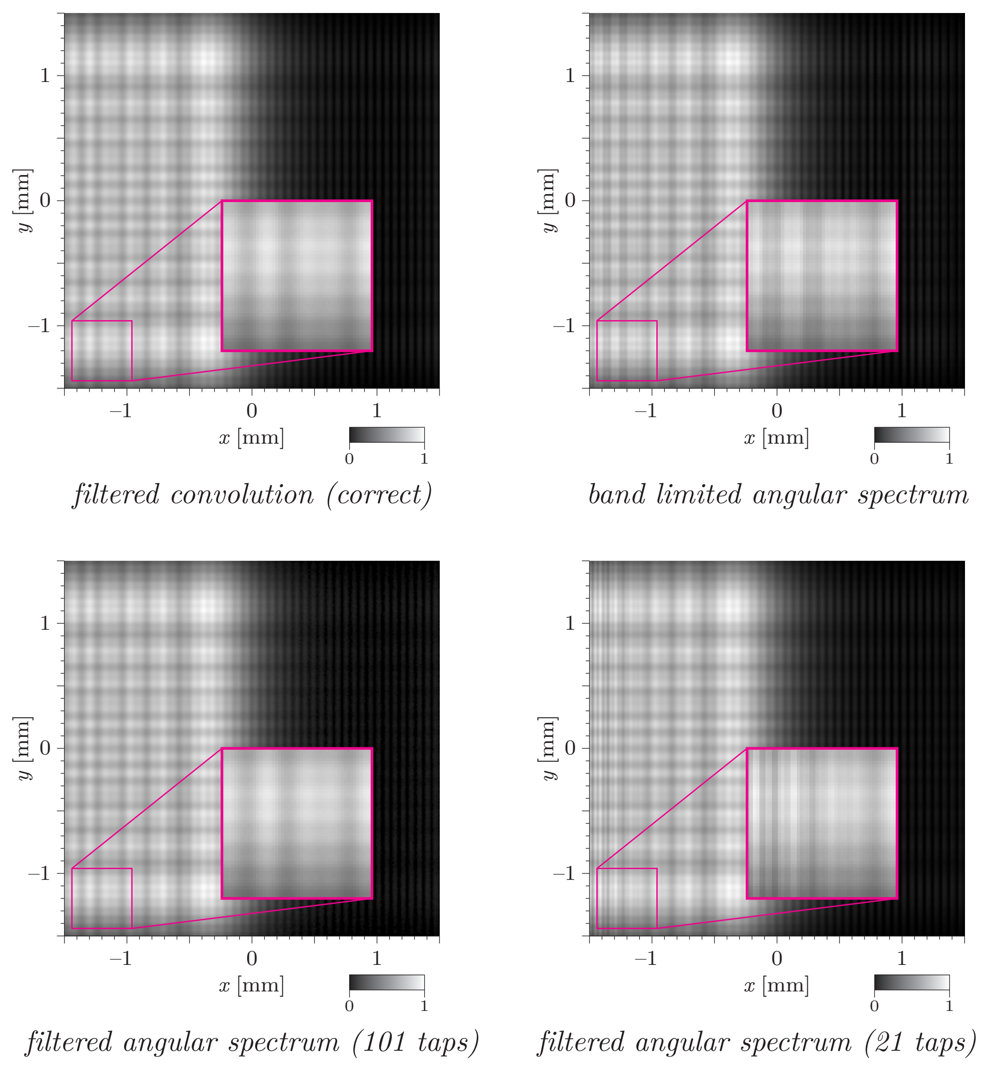
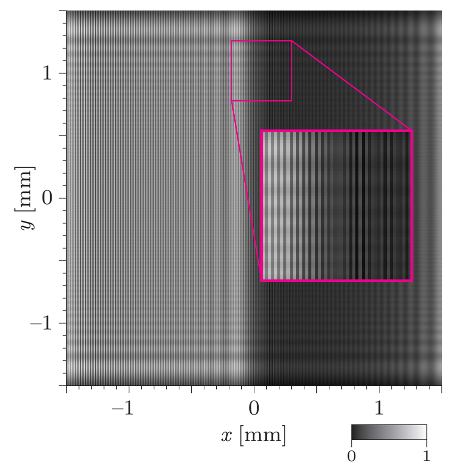


Figure 5.9: *Propagation to* $z_0 = 300$ mm *(normalized intensity images).*

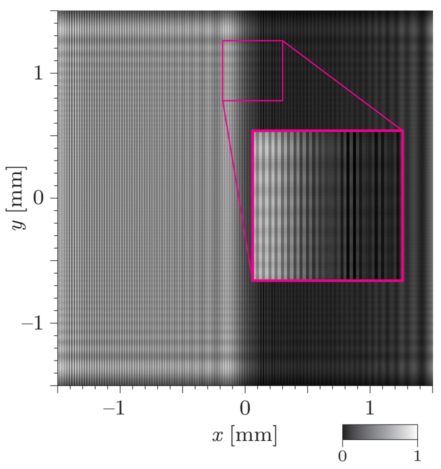
These pictures compare the results obtained by different calculation methods. The top row was already presented in Figure 5.5 and the images are repeated for convenience. The bottom row shows results obtained with kernels presented in Figures 5.7 and 5.8.

An artifact of the transfer function frequency limitation can be slightly seen in the bottom right image near the left edge. However, if we take into account damaged boundary of the kernel in the spatial domain (see Figure 5.8), it can be avoided by slightly larger zero padding.

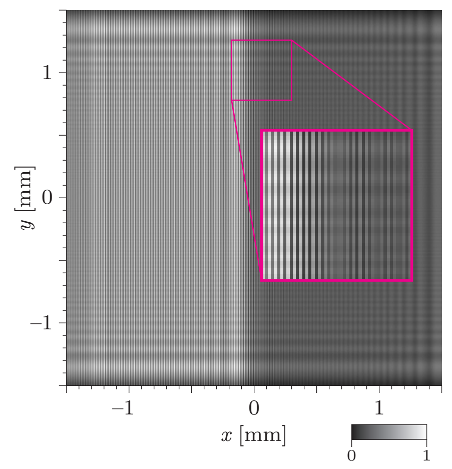
### Selected images from the supplementary tutorial, part two



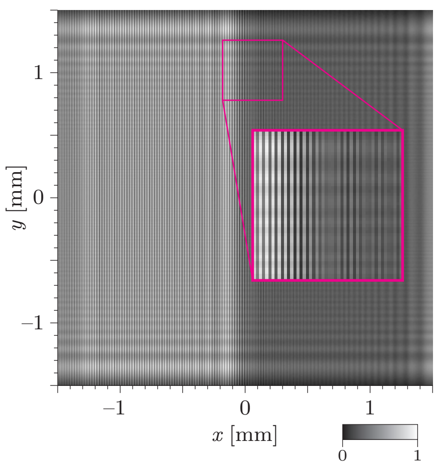
*filtered angular spectrum (21 taps)*



*band limited angular spectrum*



*filtered convolution (correct)*



*proposed method*

Figure 5.10: *Propagation to* $z_0 = 50$ mm *(normalized intensity images).*

Let us switch to a small propagation distance, $z_0 = 50$ mm. We have seen that some problems with angular spectrum were solved by applying digital low-pass filtering. But what happens if the frequencies in the transfer function are so small that the filtering does not filter out anything?

In following figures, we will discuss differences between the methods. Please note that “filtered angular spectrum” (top left) is the final method explained in the first part of this tutorial. As it still differes from the reference image, it is necessary to enhance it; the enhanced method is shown in the bottom right picture.

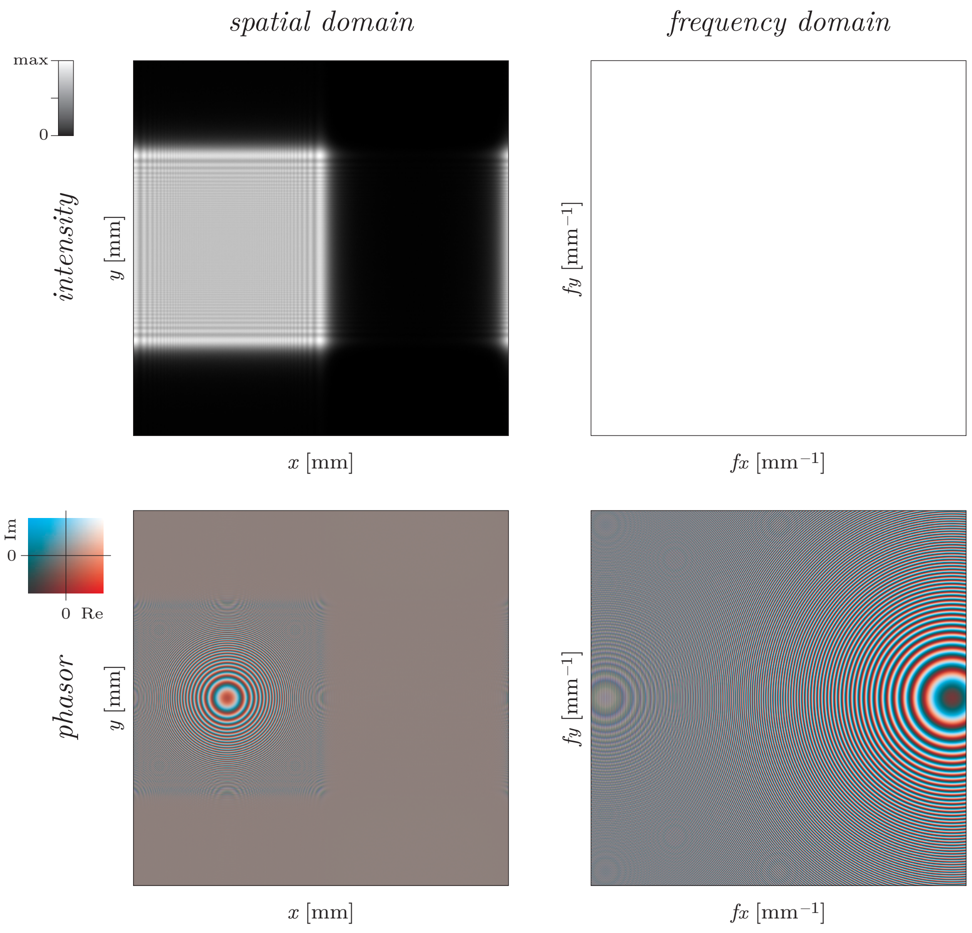


Figure 5.11: *Kernel of the plain angular spectrum method,* $z_0 = 50$ mm.

Let us begin with the method that is actually not present in Figure 5.10 – the plain angular spectrum method shown also in Figure 5.4. We can see that the transfer function (right column) is almost well sampled, only the left edge of the bottom right image seems to be aliased a bit. Images in the spatial domain confirm that. It is thus reasonable to use some low-pass filtering, although we should expect that the convolution kernel will look almost the same.

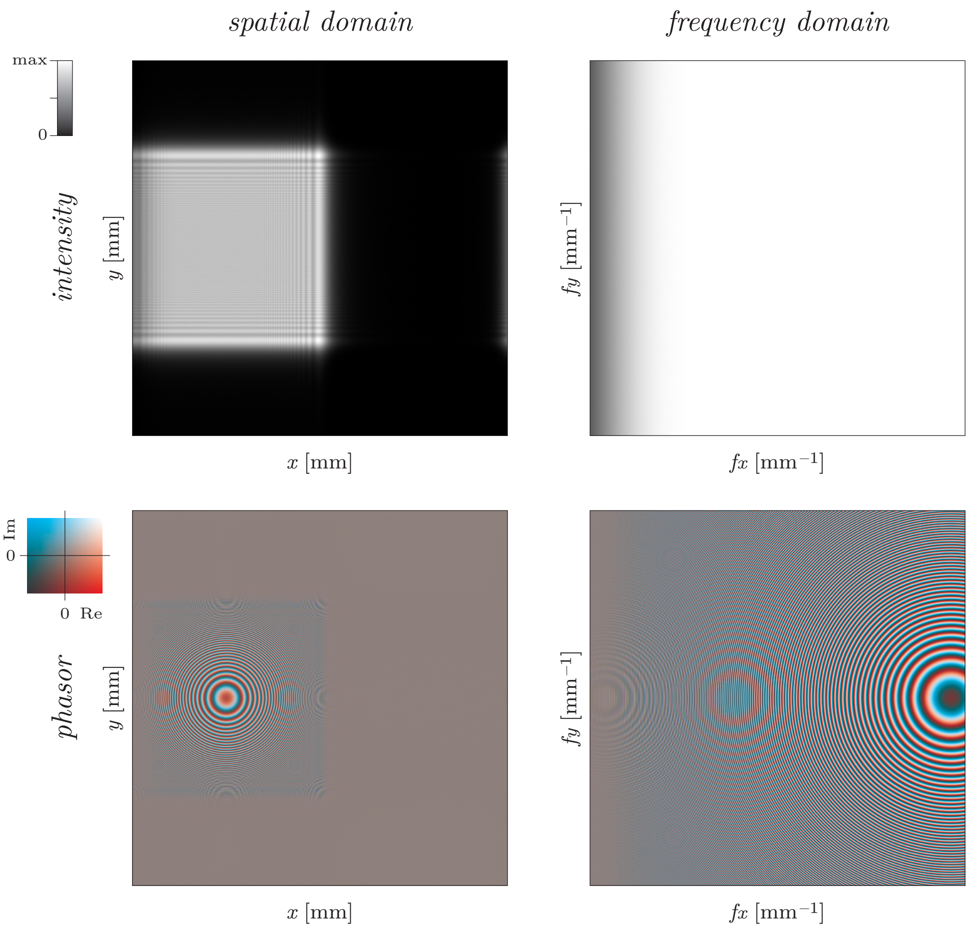


Figure 5.12: *Kernel of the filtered (21-tap windowed sinc) angular spectrum method,* $z_0 = 50$ mm.

These images show the convolution kernel used in the top left result in Figure 5.10 – the filtered angular spectrum method introduced in Figures 5.7 and 5.8. Please note that the transfer function (right column) is low-pass filtetred, which is clearly seen as the decay in the top right image.

From the signal processing perspective, everything looks well – with a small catch. The impulse response (left column) should represent light radiating from a single point (or "a sample", to be precise) to a distance 50 mm. There is, however, no clear reason why it should be limited to a square with wavy intensity near the edges.

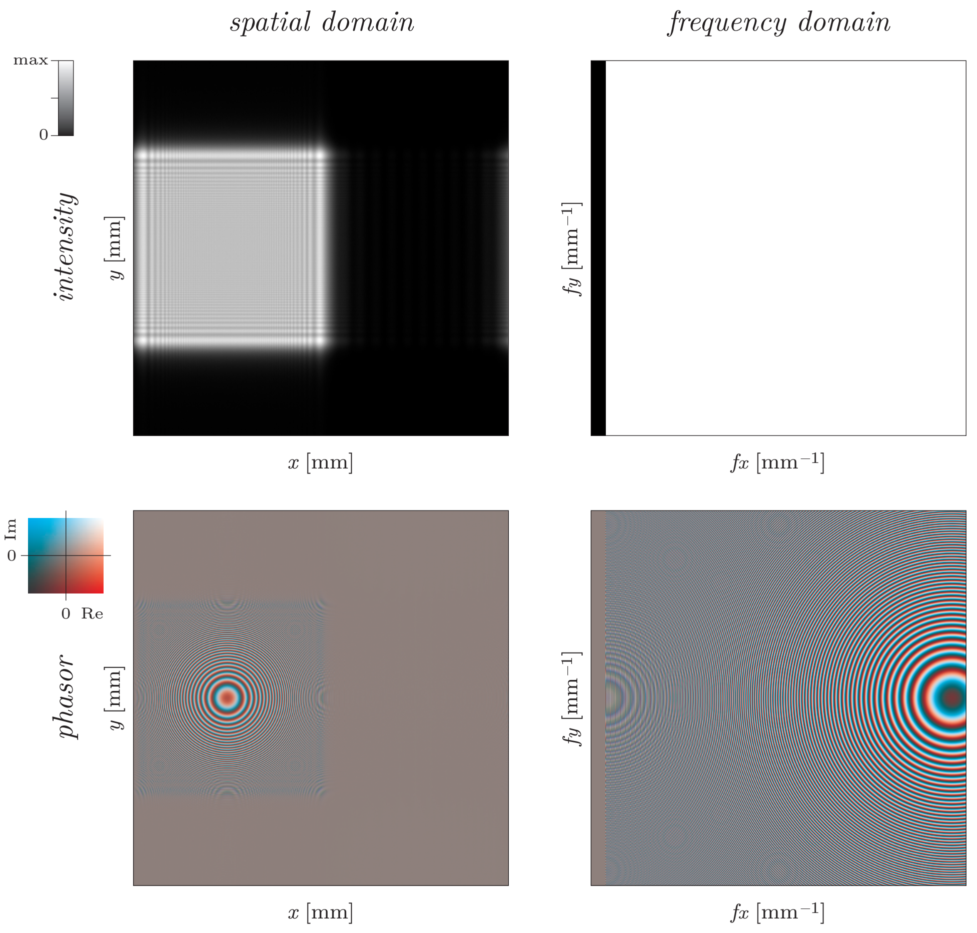


Figure 5.13: *Kernel of the band-limited angular spectrum method,* $z_0 = 50$ mm.

The band-limited angular spectrum method (top right in Figure 5.10) shows virtually the same behaviour as the filtered angular spectrum in Figure 5.12 with the same small catch.

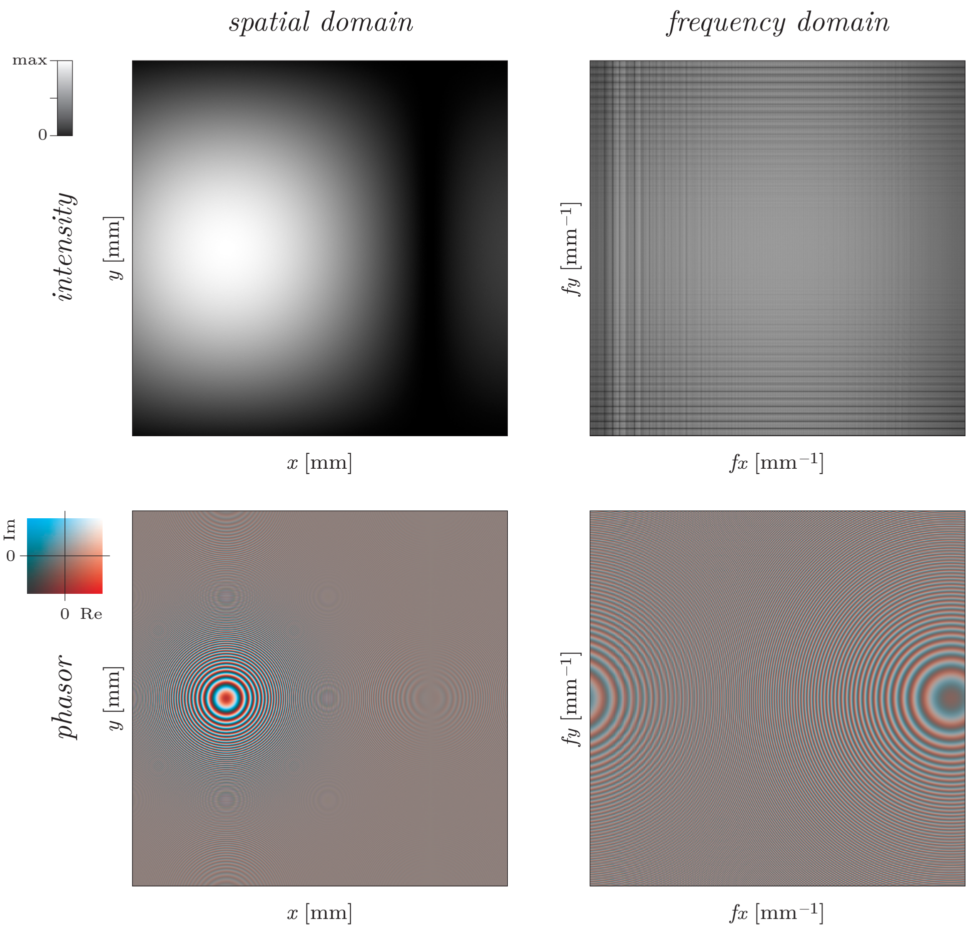


Figure 5.14: *Kernel of the filtered convolution method (correct),* $z_0 = 50$ mm.

These pictures show correct convolution kernel used for the bottom left image in Figure 5.10 – the filtered convolution method. Let us discuss why it is so much different from Figures 5.12 and 5.13.

The discrete calculation replaces the continuous light field with discrete point light sources. This change is barely visible in a far distance but makes a huge difference in a close distance. To overcome it, we have to assume that one sample of the input light field behaves as an elementary light area. In the calculation, we have to convolve the kernel in the spatial domain with a filter that describes the elementary light area. This filter acts in fact as an additional low-pass filter that was not incorporated in the filtered angular spectrum method.

Now it is clear that the impulse response should look like light radiating from the elementary area. As we calculate light diffraction on a grating with square transmittance profile, we have chosen very small rectangle ($10 \times 10$ $\mu$m) as the elementary area. The propagation distance $z_0 = 50$ mm is very big compared to the elementary area size. Therefore, we should expect that the impulse response should look like a far field propagation – which is the case.

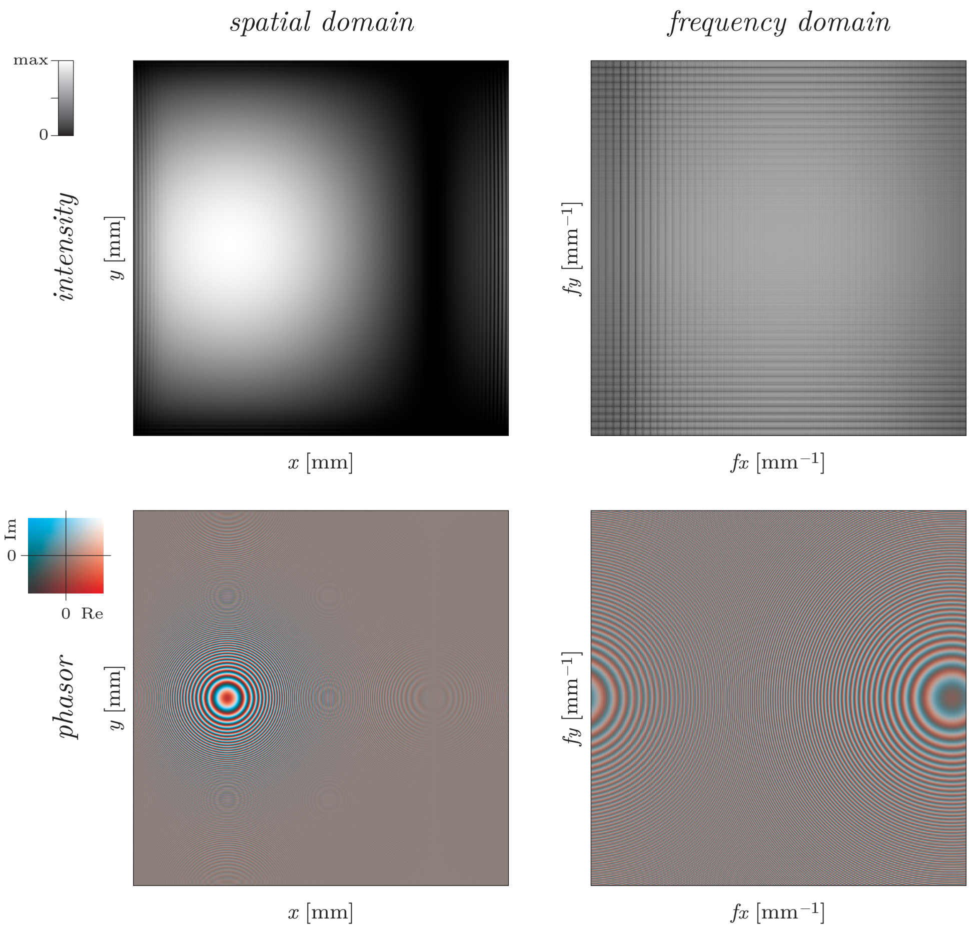


Figure 5.15: *Kernel of the proposed method,* $z_0 = 50$ mm*.*

These pictures finalize explanation of the proposed method that was used in the bottom right image in Figure 5.10.

We know that the filtered angular spectrum method in fact spatially limits somehow the kernel in the frequency domain. As the propagation distance gets smaller, this boundary gets larger until it is bigger than the frequency range we operate in. But this "frequency range" limitation is something artificial that stems from the numerical calculation only – it is not present in the continuous formulas. It means we should consider even the values of the kernel outside the frequency range we operate in.

As explained in the following article, the filtered convolution calculation can be done in two steps: first, calculate the propagation with finer sampling period in the spatial domain (i.e. bigger frequency range), and second, downsample the result. (These steps can be performed at once, there is no need to use bigger arrays in the calculation.) The downsampling in the spatial domain folds several frequency bands into one – and it is exactly what we should do. Please note that frequency band folding is in fact aliasing. Thus, while the conventional wisdom recommends to avoid aliasing, we are intentionally adding it.

Comparison of the kernels shown in Figures 5.11–5.15 confirms that this unexpected step is correct. The propagation result (bottom right image in Figure 5.10) reveals that the method is still not perfect but it is far more precise than the other angular spectrum based methods.

One may generally ask if aliasing is a friend or a foe. It depends. If we are interested in the real and the imaginary part of the final result, than alias is definitely bad. However, low-pass filtered (i.e. alias free) version does not contain high frequencies at all so it is questionable what version is better. The only correct solution is to use finer sampling in the spatial domain. On the other hand, if we are interested in the intensity only (and we often are), we can ignore aliasing of the real and imaginary parts if we are sure that the intensity picture is not aliased.

For example, if the complex signal is $[1\ \mathbf{j}\ \ 1\ \mathbf{j}\ \ 1\ \mathbf{j}\ \ 1\ \mathbf{j}\ \ \ldots]$, then its twice downsampled (i.e. aliased) version is, e.g., $[1\ \ 1\ \ 1\ \ 1\ \ \ldots]$. The squared absolute values ("intensities") of both signals are, however, the same – all ones. On the other hand, "proper" downsampling should be preceded by a low-pass filtering. Its result should look like $[(1+\mathbf{j})/2\ \ (1+\mathbf{j})/2\ \ (1+\mathbf{j})/2\ \ \ldots]$; now we can see that the intensities are damaged.

A more mathematical discussion is contained in the following article, as well as formal introduction of the method.

9th International Symposium on Display Holography (ISDH 2012) IOP Publishing
Journal of Physics: Conference Series **415** (2013) 012040 doi:10.1088/1742-6596/415/1/012040

# Discrete calculation of the off-axis angular spectrum based light propagation

**P Lobaz**

Department of Computer Science and Engineering, University of West Bohemia, Univerzitni 8, 306 14 Plzen, Czech Republic

E-mail: `lobaz@kiv.zcu.cz`

**Abstract.** Light propagation in a free space is a common computational task in many computer generated holography algorithms. A solution based on the angular spectrum decomposition is used frequently. However, its correct off-axis numerical implementation is not straightforward. It is shown that for long distance propagation it is necessary to use digital low-pass filtering for transfer function calculation in order to restrict source area illumination to a finite area. It is also shown that for short distance propagation it is necessary to introduce frequency bands folding in transfer function calculation in order to simulate finite source area propagation. In both cases it is necessary to define properly interpolation filters that reconstruct continuous nature of the source area out of its sampled representation. It is also necessary to zero-pad properly source area sampling in order to avoid artifacts that stem from the periodic nature of the fast Fourier transform.

## 1. Introduction

To calculate coherent light propagation in a free space, scalar approximation is used frequently. A common task is such a calculation where complex amplitudes of light are given in the area *source* in a plane $z = 0$ and we look for complex amplitudes in the area *target* in a plane $z = z_0$, $z_0 > 0$. A common procedure leads to the Rayleigh-Sommerfeld integral of the first kind [1], or to its mathematically equivalent form, the angular spectrum decomposition [2]. Approximations of these formulas are used frequently, namely Fresnel and Fraunhofer approximations. However, these approximations are used in paraxial regime while in computer generated holography an off-axis solution is often needed (e. g. [3]); therefore we will not discuss them.

The problem has to be solved numerically in computer generated holography. This leads to discretization of signals. Discretization of the Rayleigh-Sommerfeld solution is not straightforward [4, 5] and discretization of the angular spectrum decomposition is tricky [6, 7]. Onural [6] describes the discretization process in general and shows that it leads to formation of signal copies in spatial domain that can be filtered out. However, he does not discuss implementation issues. Matsushima [7] deals with the implementation and solves a troublesome aliasing problem by local frequency estimation; however, he does not analyse the effect of hard frequency clipping.

This article focuses on reference calculation of light propagation between parallel planes using angular spectrum decomposition. We will point out what makes the discretization difficult

9th International Symposium on Display Holography (ISDH 2012) IOP Publishing
Journal of Physics: Conference Series **415** (2013) 012040 doi:10.1088/1742-6596/415/1/012040

and how to overcome the problems so that the method provides the same results as the Rayleigh-Sommerfeld method. We will deal with both large propagation distances discussed by Matsushima [7] and small propagation distances that were not discussed in literature yet. We will explain the meaning of "large" and "small" distance later.

Structure of the article is as follows. At first we will precisely show how to discretize the propagation calculation based on convolution (i. e. based on Rayleigh-Sommerfeld integral). We will show that it is necessary to introduce spatial limitation of a convolution kernel for successful discretization. This leads to frequency limitation of the transfer function used in the angular spectrum decomposition method. We will show that the frequency limited transfer function can be calculated using digital signal processing methods. At last we will deal with the case where this frequency limitation is actually useless due to small propagation distance. We will show that, in this case, we have to deal with exact nature of sampling and reconstruction of signals involved. We will show that the choice of reconstruction leads to introduction of artificial alias.

## 2. How to read the article

The mathematical explanation presented may be unpleasant to follow. Readers are therefore encouraged to go through the presentation packed as a "multimedia" attachment to this article. It shows pictures containing various problems that appear when calculating the propagation numerically. I have decided to attach these pictures as a separate media for two reasons. The first one is: the pictures show mainly problems with aliasing. It is therefore needed to control the display of these images precisely which is not possible in a PDF reader or in printed media. The second one is: separate media gives the opportunity to show much more images than any printed media allows. I should note that the presentation does not contain any information not covered by the article.

## 3. Convolution discretization

Let us assume that we know complex amplitudes $u(x, y, 0)$ of monochromatic coherent light of wavelength $\lambda$ in the area *source* ($x_{s\min} \le x < x_{s\max}$, $y_{s\min} \le y < y_{s\max}$, $z = 0$) and we want to calculate complex amplitudes $u(x, y, z_0)$ in the area *target* ($x_{t\min} \le x < x_{t\max}$, $y_{t\min} \le y < y_{t\max}$, $z = z_0 > 0$). The solution is given by the Rayleigh-Sommerfeld integral of the first kind:

$$u(x, y, z_0) = \frac{-1}{2\pi} \iint\limits_{-\infty}^{\infty} u(\xi, \eta, 0) \frac{\partial}{\partial z} \frac{\exp(\mathrm{j}kr)}{r} \,\mathrm{d}\xi \,\mathrm{d}\eta \tag{1}$$

where $r = \sqrt{(x-\xi)^2 + (y-\eta)^2 + z_0^2}$, $k = 2\pi/\lambda$, $\mathrm{j}^2 = -1$ and $u(\xi, \eta, 0) = 0$ for $[\xi, \eta, 0] \notin$ *source*. The second term of the multiplication inside the integral depends on $(x - \xi)$ and $(y - \eta)$ only, which means that the integral can be rewritten as the convolution with the Rayleigh-Sommerfeld kernel $h(x, y, z)$:

$$u(x, y, z_0) = u(x, y, 0) \otimes h(x, y, z_0) = \iint\limits_{-\infty}^{\infty} u(\xi, \eta, 0) h(x - \xi, y - \eta, z_0) \,\mathrm{d}\xi \,\mathrm{d}\eta \tag{2}$$

$$h(x, y, z) = \frac{-1}{2\pi} \frac{\partial}{\partial z} \frac{\exp(\mathrm{j}kr)}{r} = \frac{-z}{2\pi} \left(\mathrm{j}k - \frac{1}{r}\right) \frac{\exp(\mathrm{j}kr)}{r^2}$$
$$r = \sqrt{x^2 + y^2 + z^2}$$

The kernel $h(x, y, z)$ can be interpreted easily: the value $h(x_d, y_d, z_d)$ describes the change of the complex amplitude of light travelling from the point $[x_s, y_s, z_s]$ to the point $[x_s + x_d, y_s + y_d, z_s + z_d]$.

9th International Symposium on Display Holography (ISDH 2012) IOP Publishing
Journal of Physics: Conference Series **415** (2013) 012040 doi:10.1088/1742-6596/415/1/012040

To calculate $u(x, y, z_0)$ in the area *target* using (1) we have to know the kernel $h(x, y, z_0)$ for $x_{t\,\mathrm{min}} - x_{s\,\mathrm{max}} \le x < x_{t\,\mathrm{max}} - x_{s\,\mathrm{min}}$, similarly for $y$. The value $h(x, y, z_0)$ is not important for other $x$, $y$ because in this case $u(x, y, 0) = 0$. We will use this fact in a while.

We discretize (1) easily by changing integrals to sums and differentials to differences. The sums will have finite extent of the indices thanks to (in fact) finite domain of the integration. Therefore their calculation will be easy, although the computational complexity will be high.

To reduce computational complexity, let us rewrite the equation (2):

$$u(x, y, z_0) = u(x, y, 0) \otimes h(x, y, z_0) = \mathcal{F}^{-1}\Big\{ \mathcal{F}\{u(x, y, 0)\} \cdot \mathcal{F}\{h(x, y, z_0)\} \Big\} \tag{3}$$

where $\mathcal{F}$ and $\mathcal{F}^{-1}$ are 2-D Fourier and inverse Fourier transform respectively. We will try to use the discrete Fourier transform (DFT) implemented as the fast Fourier transform (FFT) after discretization.

The discrete Fourier transform can be defined if the original continuous functions are periodic before discretization (sampling); then we take into account one period of functions $u_p(x, y, z_0)$, $u_p(x, y, 0)$ and $h_p(x, y, z_0)$ derived from functions $u(x, y, z_0)$, $u(x, y, 0)$ and $h(x, y, z_0)$. We can define the DFT another way if the functions to be transformed are spatially limited; in this case we can assume just one period of the functions $u_p(x, y, z_0)$, $u_p(x, y, 0)$ a $h_p(x, y, z_0)$. Both ways lead to the same results. This means that the results of the DFT can be interpreted in both ways. We will choose the way that will be more suitable in a particular situation.

Let us briefly give a hint what is the meaning of the functions $u_p(x, y, 0)$, $u_p(x, y, z_0)$ and $h_p(x, y, z_0)$ before we define them precisely. Their period in $x$ direction is equal to the sum of widths of the *source* and the *target* (similarly in $y$). One period of the function $u_p(x, y, 0)$ is composed of values of $u(x, y, 0)$ in the way that one corner of the *source* is translated to the origin. The meaning of the function $u_p(x, y, z_0)$ is similar. The $h_p(x, y, z_0)$ is restricted and shifted version of the function $h(x, y, z_0)$. The meaning of the value $h_p(0, 0, z_0)$ is the change of the complex amplitude of light travelling from the point $[x_{s\mathrm{min}}, y_{s\mathrm{min}}, 0]$ to the point $[x_{t\mathrm{min}}, y_{t\mathrm{min}}, z_0]$, see fig. 1.

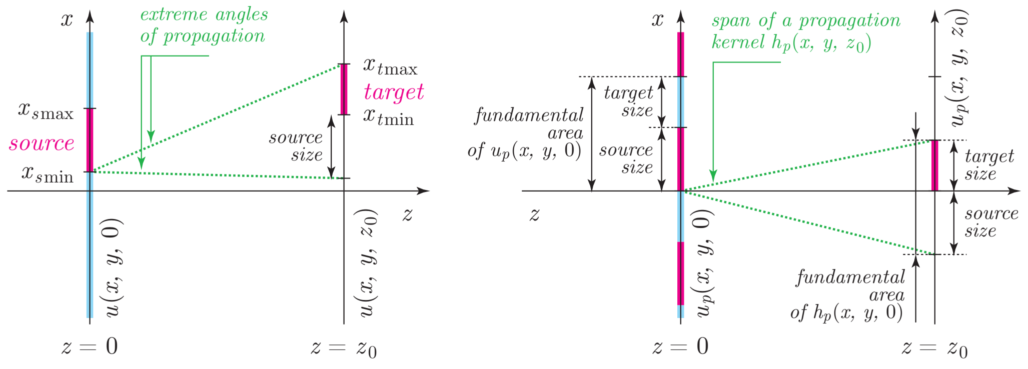


**Figure 1.** Geometry of the problem in $xz$ slice. *Left image:* original setup. Red line in the source area denotes the complex amplitudes to be propagated, blue line in the same plane defines zero value. It should be clear that we need to know the values of a propagation kernel contained in a green wedge only. *Right image:* setup prepared for discretization. *Source* and *target* are shifted to the $z$ axis, blue and red lines in the *source* plane show both zero padding and periodicity. The green wedge shows one period (fundamental area) of the function $h_p(x, y, z_0)$.

9th International Symposium on Display Holography (ISDH 2012) IOP Publishing
Journal of Physics: Conference Series **415** (2013) 012040 doi:10.1088/1742-6596/415/1/012040

Let us define the functions precisely. The function $h_p(x, y, z_0)$ is defined in "the fundamental area" $x_{s\min} - x_{s\max} \leq x < x_{t\max} - x_{t\min}$ (similarly for $y$) as:

$$h_p(x, y, z_0) = h(x + x_{t\min} - x_{s\min}, y + y_{t\min} - y_{s\min}, z_0) \tag{4}$$

Definition outside the fundamental area depends on situation: it can be either zero or the function can be made periodic. However, as we will see, the values outside the fundamental area are not important, which means that the periodic nature of the function is not harmful.

Let us define the function $u_p(x, y, 0)$ in the fundamental area $0 \leq x < (x_{t\max} - x_{t\min}) + (x_{s\max} - x_{s\min})$ (similarly for $y$; notice that the size of the area is the same as for $h_p(x, y, z_0)$) as:

$$u_p(x, y, 0) = \begin{cases} u(x + x_{s\min}, y + y_{s\min}, 0) & \text{for } 0 \leq x < x_{s\max} - x_{s\min}, \\ & \phantom{\text{for }} 0 \leq y < y_{s\max} - y_{s\min} \\ 0 & \text{elsewhere in the fundamental area} \end{cases}$$

and again let us make it periodic.

Let us calculate $u_p(x, y, z_0) = u_p(x, y, 0) \otimes h_p(x, y, z_0)$. If we consider the function $u_p(x, y, 0)$ to be periodic and the function $h_p(x, y, z_0)$ to be zero outside the fundamental area, we can easily prove that $u_p(x, y, z_0)$ will contain correct result of the propagation from the *source* to the *target* in the area $0 \leq x < x_{t\max} - x_{t\min}$ (similarly for $y$); the values of $h_p(x, y, z_0)$ are meaningless for other $x$, $y$ as they are damaged by periodicity of the function $u_p(x, y, 0)$. The proof easily follows from the geometry of the problem.

If we consider the function $u_p(x, y, 0)$ to be spatially limited and the function $h_p(x, y, z_0)$ to be periodic, we get the same result. If we consider both functions to be periodic, the meaning of the result remains the same; however, we are in fact in the world of discrete Fourier transform.

The form of the discrete calculation follows from the aforementioned ideas. The area *source* is discretized by $M_x \times M_y$ samples, the area *target* by $N_x \times N_y$ samples. For simplicity let us assume such parameters so that the sampling period $\Delta$ is the same in both directions $x$ and $y$ and in both *source* and *target* areas. Its value is then e. g. $\Delta = (x_{s\max} - x_{s\min})/M_x$.

The area *source* is discretized by samples $u_0[m, n] = u_p(m\Delta, n\Delta, 0)$, the convolution kernel is discretized by samples $h[m, n] = h_p(m\Delta, n\Delta, z_0)$ for $m \in \{0, 1, \ldots, M_x + N_x - 2\}$ (similarly $n$, see [4]). We consider functions $u_p(x, y, 0)$ and $h_p(x, y, z_0)$ to be periodic. Then we can calculate

$$u_{z_0}[\,] = \mathbf{IDFT}\Big\{\mathbf{DFT}\{u_0[\,]\} \cdot \mathbf{DFT}\{h[\,]\}\Big\} \tag{5}$$

to get the array $u_{z_0}[\,]$ that contains complex amplitudes of the area *target* in the first $N_x \times N_y$ elements. In the equation (5), **DFT** and **IDFT** stands for forward and backward 2-D discrete Fourier transform, respectively, and $\cdot$ stands for elementwise product (Hadamard product).

We can summarize the results as follows. For the discrete calculation of light propagation, we have to use a convolution kernel spatially limited to a certain area. The *source* has to be zero-padded to span the same area. The result of their convolution contains correct values in the area of the size proportional to the size of the *target*.

## 4. Angular spectrum discretization for large propagation distances

The equation (1) can be written in the equivalent form called the angular spectrum decomposition [2]:

$$u(x, y, z_0) = \mathcal{F}^{-1}\{U(f_x, f_y, 0) \cdot H(f_x, f_y, z_0)\}$$

where

$$\begin{aligned} U(f_x, f_y, 0) &= \mathcal{F}\{u(x, y, 0)\} \\ H(f_x, f_y, z_0) &= \exp\Big(-\mathrm{j}\, 2\pi z_0 \sqrt{\lambda^{-2} - f_x^2 - f_y^2}\Big) \end{aligned}$$

9th International Symposium on Display Holography (ISDH 2012) IOP Publishing
Journal of Physics: Conference Series **415** (2013) 012040 doi:10.1088/1742-6596/415/1/012040

and $f_x$, $f_y$ are Fourier domain variables.

It follows from (3) that

$$u(x, y, z_0) = u(x, y, 0) \otimes h(x, y, z_0) = \mathcal{F}^{-1}\left\{U(f_x, f_y, 0) \cdot \mathcal{F}\{h(x, y, z_0)\}\right\},$$

that is $H(f_x, f_y, z_0) = \mathcal{F}\{h(x, y, z_0)\}$. It seems that the propagation calculation should be advantageous using the angular spectrum decomposition compared to convolution with the Rayleigh-Sommerfeld kernel because we have one Fourier transform less – the transform of the kernel is known in the analytic form.

In the following paragraphs, we will talk about various limitations of functions in both spatial and Fourier (frequency) domain. A function $f(x, y)$ defined in spatial domain is spatially limited if it is zero outside a bounded area in the plane $(x, y)$; it is frequency limited if $\mathcal{F}\{f(x, y)\}$ is zero outside a bounded area in the plane $(f_x, f_y)$. Similarly, a function $F(f_x, f_y)$ defined in frequency domain is spatially limited if it is zero outside a bounded area in the plane $(f_x, f_y)$; it is frequency limited if $\mathcal{F}^{-1}\{F(f_x, f_y)\}$ is zero outside a bounded area in the plane $(x, y)$.

We know from the previous section that we have to introduce certain functions to discretize the calculation. We have to define the periodic function $u_p(x, y, 0)$ based on the function $u(x, y, 0)$ and to calculate its (discrete) Fourier transform; we will follow this procedure exactly. We also need to spatially restrict the function $h(x, y, z_0)$, and to make its periodic form alternatively. However, we do not know the Fourier transform of the function $h_p(x, y, z_0)$ in the analytic form.

Fortunately we can use digital signal processing tools. If the signal is spatially limited in one domain (in our case spatial domain), it is frequency limited in the other domain (in our case frequency domain). We can calculate frequency limited signal using low-pass filter $l(f_x, f_y)$. Let us assume the function $h_p(x, y, z_0)$ to be spatially limited (i. e. not periodic). For propagation calculation, we have to use the transfer function $H_p(f_x, f_y, z_0) = H(f_x, f_y, z_0) \otimes l(f_x, f_y)$.

It follows from properties of the Fourier transform that the function $H_p(f_x, f_y, z_0)$ has to be spatially unlimited. This does not matter even in numerical calculation. Since the function $u_p(x, y, 0)$ is periodic, then $U_p(f_x, f_y, 0)$ is spatially limited; and we need to calculate the product $U_p(f_x, f_y, 0)H_p(f_x, f_y, 0)$. It also follows that the value of the function $H_p(f_x, f_y, z_0)$ can be arbitrary outside the important area, and therefore we can use its periodic form to introduce discrete calculation correctly.

We cannot limit the frequency content of the function $H(f_x, f_y, z_0)$ sharply using digital low-pass filtering; the frequency limitation is approximate. In the spatial domain it means that the transition between zero and non-zero part is gradual instead of sharp. This does not matter either. All we need to do is to enlarge the fundamental area of the functions $u_p()$ and $h_p()$, i. e. zero-padding of the array $u_p[\,]$ will be larger than defined in section 3, so that the gradual transition will not affect the *target* area.

We will face a problem in a practical implementation. The function $H(f_x, f_y, z_0)$ is frequency unlimited – the local frequency [1, 7] in the point $[f_x, f_y]$ grows without bound as this point approaches a circle of a radius $\lambda^{-1}$ centered in the origin. Moreover, if the propagation distance $z_0$ is large, then the local frequency will be large as well everywhere except in the origin. Therefore it is impossible to sample the function $H(f_x, f_y, z_0)$ correctly and then the low-pass filtering will not work properly.

However, we can use the same procedure as described by Matsushima [7] who realizes the aliasing problem. He evaluates local frequency when sampling the function $H(f_x, f_y, z_0)$, and if the local frequency is bigger than a half of the sampling frequency, he sets the function $H(f_x, f_y, z_0)$ to be zero. He solves the aliasing problem this way, on the other hand he introduces a spatial limitation of the function $H(f_x, f_y, z_0)$. It follows that the propagation kernel $\mathcal{F}^{-1}\{H_{\text{Matsushima}}(f_x, f_y, z_0)\}$ is then spatially unlimited which is not correct. It should be however emphasised that even though it gives remarkably good results.

9th International Symposium on Display Holography (ISDH 2012) IOP Publishing
Journal of Physics: Conference Series **415** (2013) 012040 doi:10.1088/1742-6596/415/1/012040

The solution is easy. We can sample the function $H(f_x, f_y, z_0)$ using higher sampling frequency than desired and use Matsushima's procedure to avoid alias. Then we can filter it using $l(f_x, f_y)$ and downsample the result to get a correct sampling frequency. As the cutoff frequency of the filter $l(f_x, f_y)$ is derived from the sizes of the arrays used in the calculation, it is guaranteed that no aliasing appears when downsampling.

It follows from practical experiments that it is sufficient to sample the function $H(f_x, f_y, z_0)$ using sampling frequency twice as high as desired, and to use sinc low-pass filter with Hamming window as $l(f_x, f_y)$. The size of the filter should be chosen carefully – a long filter filters high frequencies well, but takes long to evaluate. Evaluation with a short filter is faster but slow attenuation of high frequencies has to be compensated with bigger zero-padding of the array $u_0[\,]$.

## 5. Discretization for short propagation distances

To calculate the propagation numerically (either using convolution approach or angular spectrum approach), we have to calculate (discrete) Fourier transform of the function $u_p(x, y, 0)$. Its result is the function $U_p(f_x, f_y, 0)$ limited to the area $A = (-1/(2\Delta), 1/(2\Delta)) \times (-1/(2\Delta), 1/(2\Delta))$, or its periodic form. We also need to calculate the function $H_p(f_x, f_y, z_0)$ in the same area, and then to calculate $\mathcal{F}^{-1}\{U_p(f_x, f_y, 0)H_p(f_x, f_y, z_0)\}$.

We can naturally ask a question: what happens if the propagation distance $z_0$ is so small that the low-pass filtering of the function $H(f_x, f_y, z_0)$ will not have any significant effect inside the area $A$? It is not easy to find the answer. Let us start with one more look into the convolution based approach described by the equation (2).

The convolution kernel $h(x, y, z_0)$ is a spatially unlimited function that is in fact frequency limited if we ignore evanescent waves. We can easily show that its local frequency grows as the point $[x, y]$ moves away from the origin. The smaler is $z_0$ the faster is the growth. It can therefore easily happen that the function $h_p(x, y, z_0)$ cannot be properly sampled using sampling period $\Delta$ for small propagation distances.

Physical meaning of wrong sampling is easy. Discretization is based on change of integrals to sums in the equation (1). It means that we change a continuous light field in the *source* to a number of point light sources. This replacement cannot be observed from a big distance or in on-axis case but makes a big difference close enough or in off-axis case.

We have to assume (especially in small propagation distances) that one sample of the function $u(x, y, 0)$ represents behaviour of the light field in a small neighbouring area. Let us denote the result of the sampling of the function $u(x, y, 0)$ by the function $u_s(x, y, 0) = u(x, y, 0)\,\mathrm{comb}(x/\Delta)\,\mathrm{comb}(y/\Delta)$, where $\mathrm{comb}(x)$ is the sampling function with period of samples 1 (see [1]). Then we can describe the reconstruction of the continuos form using convolution with a reconstruction kernel $r(x, y)$:

$$u(x, y, 0) \approx u_r(x, y, 0) = u_s(x, y, 0) \otimes r(x, y)$$

where $u_r(x, y, 0)$ is a continuos function reconstructed from the discrete samples. The function $u_r(x, y, 0)$ is more or less similar to the function $u(x, y, 0)$ depending on the shape of a reconstruction kernel $r(x, y)$ and size of the sampling period $\Delta$.

We can therefore express the propagation as

$$u(x, y, z_0) \approx u_s(x, y, 0) \otimes \Big(r(x, y) \otimes h(x, y, z_0)\Big)$$

It is possible to put big parenthesses in the equation thanks to associativity of the convolution. It follows that we should not use the function $h(x, y, z_0)$ for discrete propagation calculation, but we shoud use its filtered version $r(x, y) \otimes h(x, y, z_0)$ instead.

9th International Symposium on Display Holography (ISDH 2012) IOP Publishing
Journal of Physics: Conference Series **415** (2013) 012040 doi:10.1088/1742-6596/415/1/012040

It is most common to demand for one sample of the function $u(x, y, 0)$ to influence its close neighbourhood only. For example, the kernel $r(x, y) = \mathrm{rect}(x/\Delta)\,\mathrm{rect}(y/\Delta)$, where $\mathrm{rect}(x)$ is a rectangular pulse of unity width and height centered in the origin, implies the function $u(x, y, 0)$ to be approximated with a piecewise constant function. We can construct other kernels as well that provide piecewise bilinear approximation, piecewise bicubic approximation and so on. In either case, the kernel $r(x, y)$ acts as a low-pass filter. If we use the filtered function $h(x, y, z_0) \otimes r(x, y)$ for construction of the function $h_p(x, y, z_0)$ in the equation (4), we get the result of the propagation calculation as precise as the function $u_r(x, y, 0)$ resembles the function $u(x, y, 0)$. Practical implementation of the procedure is described in [5].

We can repeat the analysis for the angular spectrum decomposition as well. We should not use the function $H(f_x, f_y, z_0)$ for construction of the function $H_p(f_x, f_y, z_0)$; we should use the Fourier transform of the filtered propagation kernel, the function $\mathcal{F}\{h(x, y, z_0\} \otimes r(x, y)\} = H(f_x, f_y, z_0)\,\mathcal{F}\{r(x, y)\}$. If we choose convenient kernel $r(x, y)$, we will know analytic form of its Fourier transform and calculation of the product will be simple. This step limits high frequencies that were caused by the discretization process.

It remains to solve the last, but important detail. In the beginning of the section we have stated that the calculations in frequency domain are done inside the area $A$. It is however possible that the support of the function $H_p(f_x, f_y, z_0)$ will not fit into the area $A$ even if it was filtered with both filters $l(f_x, f_y)$ and $r(x, y)$. If we reject the values of the function $H_p(f_x, f_y, z_0)$ outside of $A$ despite that fact, it means that the final function $u(x, y, z_0)$ was filtered by a third, still unjustified low-pass filter.

We can explain this final low-pass filter. It would be appropriate if the kernel $r(x, y)$ represents a perfect sinc low-pass filter limiting the frequency content of the function $u(x, y, z_0)$ to a range described by the area $A$. Then the kernel $r(x, y)$ has to be be spatially unlimited. If we do not care, the analysis is finished.

If we would rather keep the kernel $r(x, y)$ spatially limited (which is physically more natural), we have two choices to choose from. The first one is simple – we can use such a sampling period $\Delta/s$, $s \in \mathbf{N}$ for the calculation so that the area $A$ covers the support of $H_p(f_x, f_y, z_0)$ now. This leads to increase of time and memory demands of the calculation, of course. Moreover, if we demand sampling of the *target* to be $\Delta$, we have to downsample the result; we have to admit that a lot of values were calculated needlessly.

The second one is a bit strange at first sight. Signal downsampling in spatial domain can be described easily in frequency domain – the frequencies $f$ and $f + n/\Delta$ merge due to downsampling, where $f$ is frequency $f_x$ or $f_y$ from the range $(-1/(2\Delta), 1/(2\Delta))$ and $n \in \mathbf{Z}$. This effect is called aliasing. We want to avoid it usually; however, if we are decided to sample the *target* with an insufficient sampling period, we have to accept aliasing. It is worth to note that aliasing needs not be harmful. If we are interested in intensities $|u(x, y, z_0)|^2$ only, aliasing in the real or imaginary part of the function $u(x, y, z_0)$ may be harmless. For example, if $u(x, y, z_0) = \cos x + \mathrm{j} \sin x$, then the intensity is always 1 regardless sampling period used, while intensity of "correctly sampled" (i. e. low-pass filtered) version can be either 1 or 0 which is not correct.

The procedure in the second case is obvious: we will perform the downsampling process in frequency domain. Let us calculate the function $H_p(f_x, f_y, z_0)$ in the area $(-s/(2\Delta), s/(2\Delta)) \times (-s/(2\Delta), s/(2\Delta))$, assume it is zero outside this area, and calculate

$$H_a(f_x, f_y, z_0) = \sum_{n_x, n_y} H_p(f_x + \frac{n_x}{\Delta}, f_y + \frac{n_y}{\Delta}, z_0)$$

for $[f_x, f_y] \in A$ and all suitable $n_x$, $n_y$. To calculate the propagation, we have to use the function $H_a(f_x, f_y, z_0)$.

9th International Symposium on Display Holography (ISDH 2012) IOP Publishing
Journal of Physics: Conference Series **415** (2013) 012040 doi:10.1088/1742-6596/415/1/012040

## 6. Conclusion

The analysis shows how to numerically calculate the propagation of light using the angular spectrum decomposition method. Unlike the procedures described in the literature, it defines low-pass filters $l(f_x, f_y)$ and $r(x, y)$ that are needed to introduce for correct discretization. Then we can choose either more precise, slower calculation or faster, less precise by choosing their parameters. The analysis also shows that it is worth introducing aliasing of the transfer functions in certain situations. The results are shown in figures 2 and 3. The images may display wrong due to resampling; the reader is encouraged to look at the images in the multimedia attachment.

## References

[1] Goodman J W 2004 *Introduction to Fourier Optics* 3rd ed (Roberts & Company Publishers) ISBN 0974707724
[2] Lalor E 1968 *J. Opt. Soc. Am.* **58** 1235–1237
[3] Hanák I, Janda M and Skala V 2010 *Visual Computer* **26**(2) 83–96
[4] Lobaz P 2011 *Opt. Express* **19** 32–39
[5] Lobaz P 2012 Calculation of a coherent light propagation in a free space using filtered convolution Tech. rep. University of West Bohemia URL `http://www.kiv.zcu.cz/en/research/publications/`
[6] Onural L 2007 *J. Opt. Soc. Am.* A **24** 359–367
[7] Matsushima K 2010 *Opt. Express* **18** 18453–18463

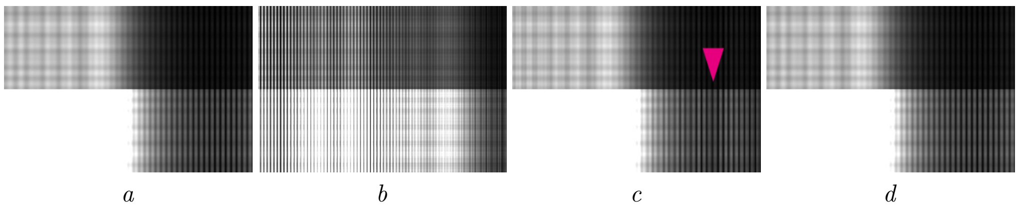


**Figure 2.** Light ($\lambda = 650$ nm) diffracted by a grating ($3 \times 3$ mm$^2$) composed of vertical slits (slits distance 40 $\mu$m). Off-axis propagation to a distance of 300 mm, size of each image $3 \times 3$ mm$^2$. Lower half of each image is overexposed to show the details. *a)* Reference Rayleigh-Sommerfeld calculation. *b)* Angular spectrum based calculation without any modification. Notice that aliasing errors destroy the image completely. *c)* Angular spectrum based calculation with Matsushima's kernel filtering. Notice different brightness of fine stripes compared to the reference image. *d)* Angular spectrum based calculation with $l(f_x, f_y)$ sinc kernel of length 50.

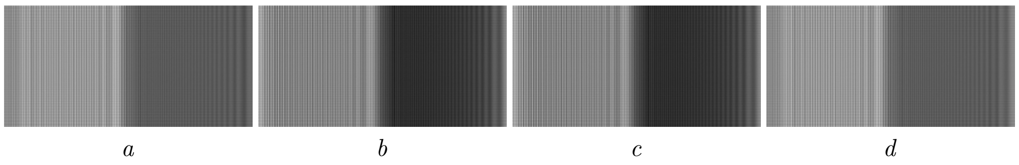


**Figure 3.** Light ($\lambda = 650$ nm) diffracted by a grating ($3 \times 3$ mm$^2$) composed of vertical slits (slits distance 20 $\mu$m). Off-axis propagation to a distance of 50 mm, size of each image $3 \times 1.5$ mm$^2$. *a)* Reference Rayleigh-Sommerfeld calculation. *b)* Angular spectrum based calculation without any modification. Notice that the right half is much darker than in the reference image. *c)* Angular spectrum based calculation with Matsushima's kernel filtering. Notice that the image is the same as without any modification because aliasing did not occur. *d)* Angular spectrum based calculation with introduced aliasing. Notice that the image is almost the same as the reference one.

# Chapter 6

# Propagation calculations in a limited precision environment

The article *Safe range of free space light propagation calculation in single precision* [37] published in Optics Express, included in this chapter from page 98, discusses another general problem of numerical light propagation calculation – limited precision of floating point numbers. It discusses influence of rounding errors, and, more importantly, when to expect problems.

Motivation to solve these problems came from [38], where the authors proposed a method to overcome errors induced by limited precision of real numbers in FPGA environment. Thus, the problem with limited precision exists. However, we have observed a contradiction in practice. On the one hand, the conventional wisdom recommends using IEEE 754 double precision numbers for light propagation calculations. On the other hand, many researchers working with GPU use single precision only and do not report any problem.

Therefore we decided to analyse influence of rounding errors rigorously. The following article reveals interesting facts:

- Calculation in single precision can provide wrong results in very realistic scenarios.
- There is a simple formula that shows if single (or any other) precision calculation is safe.
- Many researchers use the Fresnel approximation in light propagation calculation. This approximation behaves well even in limited precision calculations.

# Safe range of free space light propagation calculation in single precision

**Petr Lobaz**[1,2,*] **and Petr Vaněček**[1]

[1]*Department of Computer Science and Engineering, Faculty of Applied Sciences, University of West Bohemia, Univerzitni 8, 306 14 Plzen, Czech Republic*
[2]*NTIS – New Technologies for the Information Society, European Centre of Excellence, Faculty of Applied Sciences, University of West Bohemia, Univerzitni 8, 306 14 Plzen, Czech Republic*

*[*]lobaz@kiv.zcu.cz*

**Abstract:** Calculations of free space light propagation, such as those used in digital holography, deal with distances much longer than a wavelength. Computer representation of real numbers must therefore provide enough precision to handle this situation. We show that single precision must be used with the utmost care, which is especially important in GPU calculations. We also show that Fresnel approximation significantly improves single precision calculations for distances bigger than about one metre.

**OCIS codes:** (000.4430) Numerical approximation and analysis; (070.2025) Discrete optical signal processing; (090.1995) Digital holography; (070.7345) Wave propagation.

## References and links

1. D. Goldberg, "What every computer scientist should know about floating point arithmetic," ACM Computing Surveys **23**, 5–48 (1991).
2. Wikipedia, "List of Nvidia graphics processing units — Wikipedia, The Free Encyclopedia," (2014). [Online; accessed 25-July-2014]. http://en.wikipedia.org/w/index.php?title=List_of_Nvidia_graphics_processing_units
3. J. Song, J. Park, H. Park, and J.-I. Park, "Real-time generation of high-definition resolution digital holograms by using multiple graphic processing units," Opt. Eng. **52**, 015803 (2013).
4. A.-H. Phan, M. lan Piao, S.-K. Gil, and N. Kim, "Generation speed and reconstructed image quality enhancement of a long-depth object using double wavefront recording planes and a GPU," Appl. Opt. **53**, 4817–4824 (2014).
5. J.-M. Muller, N. Brisebarre, F. de Dinechin, C.-P. Jeannerod, V. Lefèvre, G. Melquiond, N. Revol, D. Stehlé, and S. Torres, *Handbook of Floating-Point Arithmetic* (Birkhäuser Boston, 2010).
6. J. W. Goodman, *Introduction to Fourier Optics* (Roberts & Company Publishers, 2004), 3rd ed.
7. I. Hanák, P. Zemčík, M. Žádník, and A. Herout, "Hologram synthesis accelerated in field programmable gate array by partial quadratic interpolation," Optical Engineering **8**, 1–7 (2009).
8. D. G. Voelz, *Computational Fourier Optics: A Matlab Tutorial*, Tutorial texts in optical engineering (SPIE Press, 2011).
9. P. Lobaz, "Memory-efficient reference calculation of light propagation using the convolution method," Opt. Express **21**, 2795–2806 (2013).
10. W. J. Dallas, "Phase quantization in holograms—a few illustrations," Appl. Opt. **10**, 674–676 (1971).

## 1. Introduction

It is a well known fact that double precision computer calculations usually provide more precise results than single precision ones [1]. It is also "conventional wisdom" that many calculations in wave optics (such as free space light propagation) are more reliable using double precision.

The question whether to use single or double precision numbers is quite unimportant in CPU calculations as they are often internally performed in higher than single precision and the final result is rounded to the requested one.

The question, however, is rather important in GPU calculations as current GPUs favour single precision calculations over double precision ones (see [2] or other list of GPU parameters). It is of utmost importance today as many researchers, especially in computer generated (digital) holography, utilize GPU calculations whenever possible (see e.g. [3, 4]). We will focus our attention to one of the most fundamental task in digital holography, Fourier optics and wave optics – calculation of light propagation in free space to a certain distance. As far as we know, no one analysed range of distances where it is safe to use single precision, where it is better to switch to double precision and what to do if we want to avoid double precision calculations.

In Sec. 2 we will show that single precision calculations can lead to significant problems even in realistic scenarios. In Sec. 3 we will point out the problem origin; as most readers are likely not computer scientists, an overview of important facts about floating-point representation is given there as well. Sec. 4 gives basic analysis of the problem and tells when it is safe to use single precision in on-axis calculations; Sec. 5 adds a few notes to an off-axis generalization. Sec. 6 shows that parabolic (Fresnel) approximations can save single-precision calculations in most cases. Finally, Sec. 7 supplements the discussion with reference to aliasing and phase quantization; Sec. 8 concludes the article.

## 2. The problem demonstration

Let us perform the most basic calculation in scalar wave optics: let us calculate an interference pattern on the plane $z = 0$ of two mutually coherent, monochromatic, equally bright point light sources. Simply written:

$$I(x,y,0) = \left| \frac{\exp(\mathrm{j}2\pi r_0/\lambda)}{r_0} + \frac{\exp(\mathrm{j}2\pi r_1/\lambda)}{r_1} \right|^2, \tag{1}$$

where $I(x,y,0)$ is the light intensity at a point $[x,y,0]$, $r_0$ and $r_1$ are the distances between the point $[x,y,0]$ and the particular point light source, $\mathrm{j}^2 = -1$ is the imaginary constant, and $\lambda$ is the wavelength.

Let us place the light sources (for instance) to the points $[\pm 0.001 z_0, 0, z_0]$, set $\lambda = 500$ nm and calculate the pattern for various $z_0$ in both single and double precision. Figure 1 shows the result. The patterns for $z_0 = 10$ mm are basically the same regardless precision used. It is clear that for $z_0 = 100$ mm, the pattern calculated in single precision contains noise introduced by limited precision, but its quality is generally acceptable. The pattern for $z_0 = 1000$ mm in single precision loosely reminds the correct one, but its quality is generally not acceptable. Finally, for $z_0 = 10000$ mm, the single precision result is completely useless.

This demonstration naturally leads to several questions. What is the source of such behaviour? Where is the boundary between acceptable and unacceptable calculations? What do we mean by "acceptable"? Is it possible to improve the calculation without switching to higher precision? Let us try to answer them all.

## 3. Origin of the problem

Let us begin by reviewing the basic facts about floating-point numbers. We decided to include this short section written in a form of tutorial as we do not know any suitable, short enough reference that would explain all the necessary concepts. Readers familiar with theory of floating point calculations and their limitations can skip this section.

A floating-point number $x$ is usually (according to the IEEE 754 standard for floating-point arithmetic) internally expressed as $x = s \times m \times 2^{e-p-1}$, where 1-bit number $s$ (*sign*) is equal

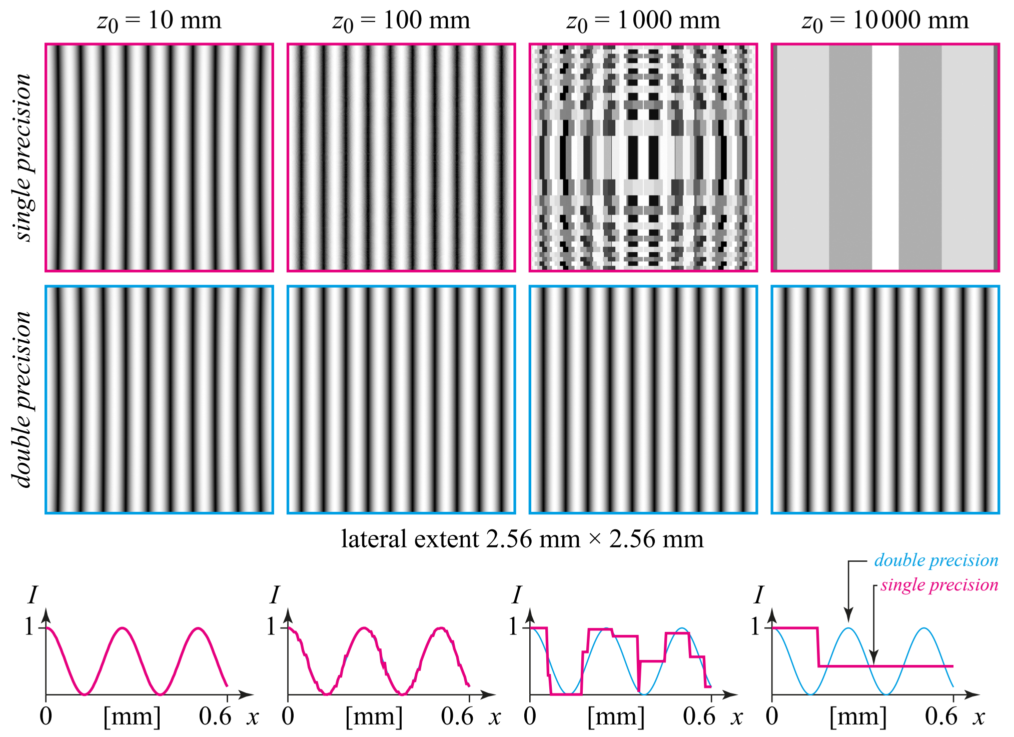


Fig. 1. Interference pattern (normalized intensity) of two point light sources located at $[\pm 0.001 z_0, 0, z_0]$ in the plane $z = 0$. The first row shows calculation in single precision, the second row in double precision. Each column stands for a particular $z_0$. The bottom row of graphs shows the normalized intensity values; for better clarity, only a segment of each image is shown.

to $+1$ or $-1$, $m$ (*mantissa* or *significand*) is a $p$-bit unsigned integer and $e$ (*exponent*) is a $q$-bit signed integer. Most often, significand $m \geq 2^{p-1}$ (number $x$ is then called a *normalized number*), or in other words, the most significant bit of $m$ is usually 1. The IEEE 754 standard defines $p = 24$ for single precision, $p = 53$ for double precision. As $\log_{10} 2^{24-1} \approx 6.9$, we can say single precision corresponds to about 7 significant decimal digits; double precision to about 16 decimal digits.

If the basic arithmetic operations ($+$, $-$, $\times$, $/$, mod, $\sqrt{\ }$) are performed according to the IEEE 754 standard, their results are the same as if they were calculated exactly and correctly rounded to the requested precision; the numerical error is therefore bounded [1]. As an example (we will recall it later in Sec. 4), let us see what happens in the addition operation.

When calculating (for instance) $25165824 + 1$ in single precision, we need to understand how the operands are internally expressed. Because $25165824 = 2^{24} + 2^{23}$, it cannot be directly represented by a 24-bit unsigned integer; instead, it is expressed as $m_1 \times 2^1$, where $m_1 = 2^{23} + 2^{22}$. The second operand can be also expressed in the normalized form as $1 = m_2 \times 2^{-23}$, where $m_2 = 2^{23}$. The exact sum is equal to $25165825 = 2^{24} + 2^{23} + 1$. Unfortunately, this number cannot be expressed by a 24-bit unsigned integer as it contains 25 significant bits. So the least significant bits (in this case one bit) must be cut off and the result must be altered in such a way the "rounding error" is the smallest. The altered 24-bit value is then used as a significand in the approximate floating-point value of the result, i.e. $(2^{23} + 2^{22}) \times 2^1$. It is clear that size of the rounding error is comparable with the unit at the least significant binary digit of the result.

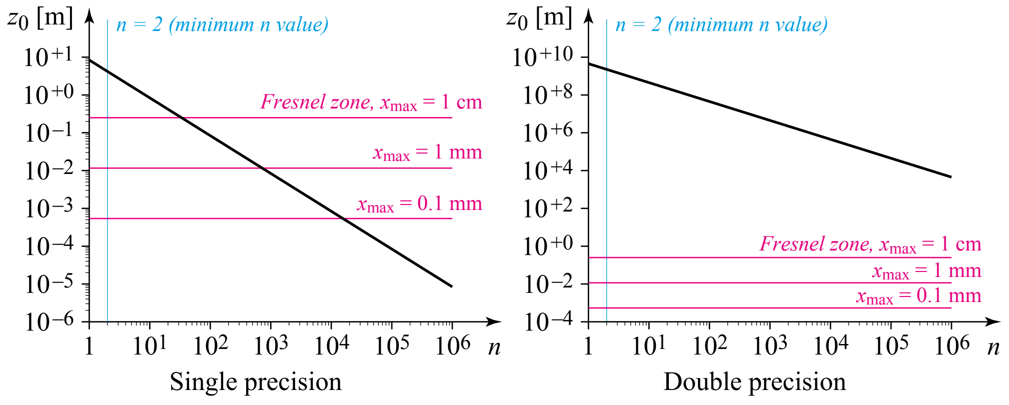


Fig. 2. Maximum $z_0$ according to inequality (2) as a function of $n$ for $\lambda = 500$ nm, i.e. the maximum distance for which $(z_0^2 + (\lambda/n)^2)^{1/2} \neq z_0$ holds. For example, if we want to calculate in single precision (left graph) and to resolve about 100 values in the first fringe, $z_0$ should be at most about 0.1 m. "Safe region" is then *below* the black line. For convenience, Fresnel zones for different $x_{\max}$ are shown as magenta horizontal lines, see Sec. 6 for details. The Fresnel approximation is valid near the $z$ axis ($|x| < x_{\max}$, $|y| < x_{\max}$) for $z_0$ *bigger* than the value depicted.

The real numerical problem appears when operands of floating-point operations are results of other floating-point operations, i.e. they are rounded versions of their "true values". Operation by operation, the numerical error accumulates without any bound.

The most numerically sensitive operations are subtraction and modulo. It is easy to see why. For example, both $t_1 = 8192 \times 8192 = 2^{13} \times 2^{13} = 2^{26}$ and $t_2 = 8191 \times 8193 = (2^{13} - 1) \times (2^{13} + 1) = 2^{26} - 1$ are evaluated as $2^{26} = 67108864$ in single precision, because significand is only 24 bits long; hence difference of their floating-point values is 0. The true difference value is of course $t_1 - t_2 = 67108864 - 67108863 = 1$. Generally, if the "true value" $t_1$ is approximated by a floating-point number $x_1 = t_1 + \varepsilon_1$ ($\varepsilon_1$ is the rounding error), other "true value" $t_2$ is approximated by $x_2 = t_2 + \varepsilon_2$, then subtraction of floating-point approximations is $x_1 - x_2 = t_1 - t_2 + \varepsilon_1 - \varepsilon_2 + \varepsilon_3$, where $\varepsilon_3$ is the rounding error of the subtraction. Naturally, if $t_1 \approx t_2$, then the subtraction result is strongly influenced by rounding errors, i.e. the relative error of the result is big.

The same problem (often called *cancellation* [1]) appears in the modulo operation, as $a \bmod b = a - \lfloor a/b \rfloor b$. The "hidden subtraction" inside the modulo operation damages the result, especially if $a$ is just a floating-point approximation of a certain true value and $b \ll a$; in this case, $\lfloor a/b \rfloor \approx a/b$, therefore $\lfloor a/b \rfloor b \approx a$ and cancellation appears.

A short look at Eq. (1) reveals origin of strange behaviour in single precision calculations. The complex exponential can be written as $\exp(\mathrm{j}2\pi r/\lambda) = \cos(2\pi r/\lambda) + \mathrm{j}\sin(2\pi r/\lambda)$, so it is necessary to evaluate trigonometric functions of a very large argument, because usually $r \gg \lambda$. The trigonometric function evaluation begins with *range reduction* [5] of its argument, which is basically (mod $2\pi$) operation; therefore it is very sensitive to cancellation due to rounding errors of the argument. Let us analyse when to expect strong cancellation problems.

## 4. On-axis analysis

The function $\exp(\mathrm{j}2\pi r/\lambda)$ is a periodic function with respect to $r$. We know that a periodic function evaluation starts with range reduction operation, in this case by $(2\pi r/\lambda \bmod 2\pi)$. It follows that when $2\pi r/\lambda \gg 2\pi$, this operation evaluated in floating-point arithmetic cuts many

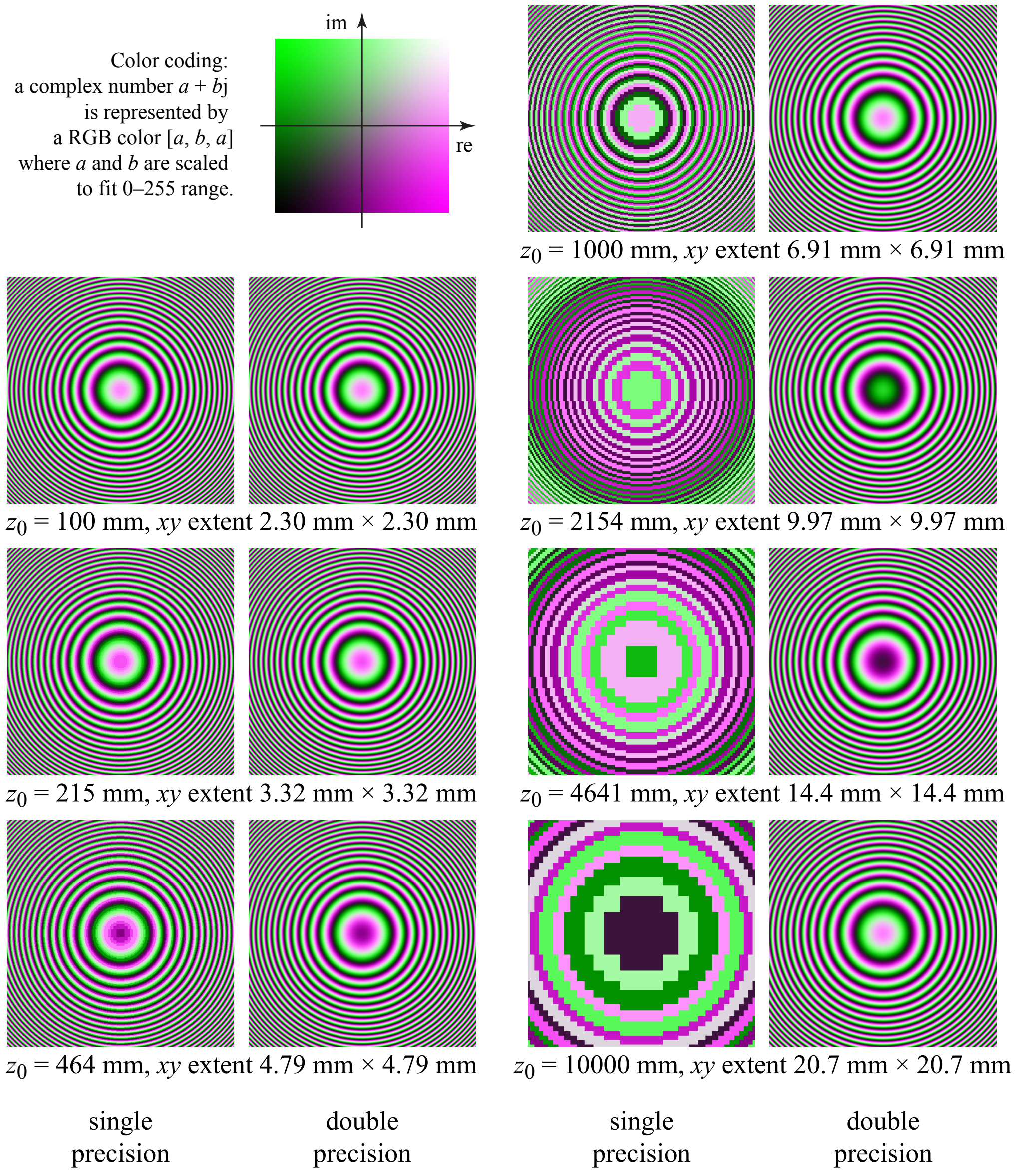


Fig. 3. Visualization of the phasor $\exp(\mathrm{j}2\pi r/\lambda)/r$ for various distances $z_0$ calculated in single and double precision. The lateral extent of each pair (single/double precision) is the same; it was chosen so that all images contain approximately the same number of fringes. It is not a fault that all double precision images look similar to each other; they really do.

significant digits of $r$. An extreme cancellation appears when $2\pi r/\lambda$ and e.g. $2\pi r/\lambda + \pi$ are expressed by the same floating point number; in this case, sine or cosine of both numbers is the same.

In order to capture fine details of $\exp(\mathrm{j}2\pi r/\lambda)$, the argument must be evaluated in floating-point arithmetic in such a way that its value differs for $r_0$ and $r_1 = r_0 + \lambda/n$ where $n \geq 2$. In other words, we need at least two different results per period in the same way as alias-free sampling requires at least two samples per period.

For simplicity, let us place the first point of evaluation to $[0,0,z_0]$, $z_0 > 0$, therefore $r_0 = z_0$. Let us place the second point of evaluation to $[x_1,0,z_0]$ in such a way that $r_1 = (x_1^2 + z_0^2)^{1/2} = r_0 + \lambda/n = z_0 + \lambda/n$. It is easy to see that $x_1^2 = 2z_0\lambda/n + (\lambda/n)^2$.

Let us continue. We need to evaluate $r_1 = (x_1^2 + z_0^2)^{1/2}$. It is absolutely necessary to hold inequality $x_1^2 + z_0^2 \neq z_0^2$ in floating-point evaluation; we have seen in Sec. 3 that floating-point summation may violate it under certain circumstances. If we assume $x_1 < z_0$, then the inequality will hold if

$$x_1^2 > 2^{-p+1} z_0^2,$$

where $p$ is again number of significand (mantissa) bits, in case of single precision $p = 24$. After substitution,

$$2z_0\lambda/n + (\lambda/n)^2 > 2^{-p+1} z_0^2.$$

It is easy to solve the quadratic inequality for $z_0$; the square root in the result can be then approximated by a Taylor series (assuming $2^{-p} \ll 1$) and we get final simple result:

$$z_0 < \lambda 2^p/n. \tag{2}$$

For $\lambda = 500$ nm, $p = 24$ (single precision) and $n = 2$ we get approximately $z_0 < 4.19$ m. For $p = 53$ (double precision) we get approximately $z_0 < 2.25 \times 10^9$ m.

It is worth explaining meaning of this value. First of all, by setting $n = 2$ we calculated upper limit of $z_0$; beyond it, the evaluated value of $\exp(\mathrm{j}2\pi r/\lambda)$ has no significant digits. We also explored just one source of rounding error, the range reduction operation; other operations also contribute to the overall error, however their contribution is very small. We have found that the upper limit is slightly less than the calculated one, about $0.95 \times \lambda 2^p/n$. See Sec. 7 for further details.

Second, we are usually interested in evaluation of $\exp(\mathrm{j}2\pi r/\lambda)$ in a non-zero area, but this analysis is valid for points only on $z$ axis. We should therefore generalize the analysis to an off-axis case; see Sec. 5. It would be definitely possible to show just off-axis analysis as it is more practical; however, its derivation is more complicated and the idea can be seen in on-axis analysis more clearly.

Finally, although evaluation near the upper limit theoretically captures the correct structure of the function $\exp(\mathrm{j}2\pi r/\lambda)$, the error is usually unacceptably big as the result has about one significant digit. For practical calculations, we should set $n \gg 2$, see Fig. 2 for a dependency of $z_0$ on $n$ and Fig. 3 for actual evaluation of the function $\exp(\mathrm{j}2\pi r/\lambda)$. See also Sec. 7 for further details.

## 5. Off-axis analysis

We are usually interested in evaluation of the function $\exp(\mathrm{j}2\pi r/\lambda)$ in a non-zero area. Without loss of generality, we can analyse the function evaluation on the $xz$ plane only.

Let us again place the first point of evaluation to $[x_0,0,z_0]$, i.e. $r_0 = (x_0^2 + z_0^2)^{1/2}$, and the second point of evaluation to $[x_1,0,z_0]$ such that $r_1 = (x_1^2 + z_0^2)^{1/2} = r_0 + \lambda/n$. By setting $a = x_0/z_0$ and applying the same procedure as in Sec. 4, we get simple rule:

$$z_0 < \lambda \frac{2^p}{n\sqrt{1+a^2}}. \tag{3}$$

The estimation of the upper $z_0$ seems to be reliable, see Fig. 4 for the visualization of the case $a = 1$ (i.e. 45° inclination), $\lambda = 500$ nm; inequality (3) predicts the calculation to be right for approximately $z_0 < 2.1$ m.

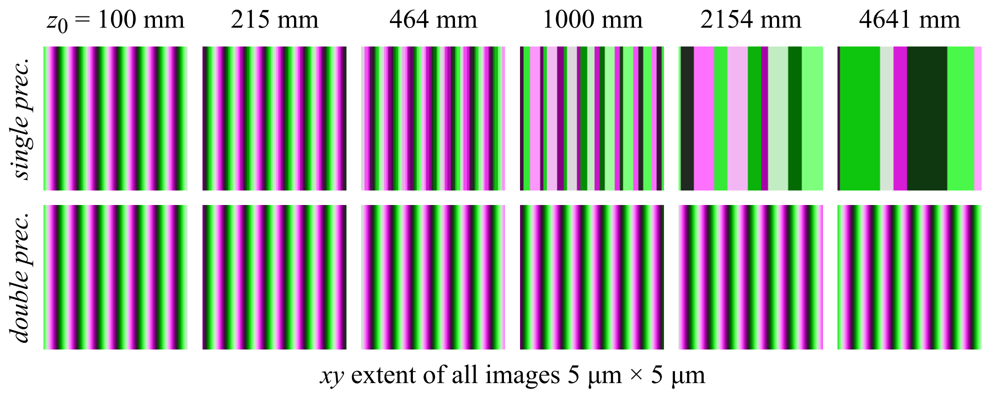


Fig. 4. Visualization of the phasor of off-axis calculation $\exp(\mathrm{j}2\pi r/\lambda)/r$ for various distances $z_0$ in single and double precision. The center of each image is set to $[z_0, 0, z_0]$, i.e. $a = 1$ (see inequality (3)). The color coding is the same as in Fig. 3.

For paraxial calculations where $x_0/z_0 \ll 1$, the result of inequality (2) is virtually the same as of inequality (3). In practical estimations, we can use inequality (2) even for off-axis calculations provided that we ask for “high enough” $n$; one should, however, ask if the scalar approximation of light is reliable in highly off-axis cases.

## 6. Fresnel approximation

The analysis and examples given clearly show that single precision calculations should be used with the utmost care; on the other hand, double precision calculations are good enough for most practical cases. Still, single precision calculation are attractive in massively parallel systems such as GPUs as single precision circuits naturally occupy less space on a chip, consume less power etc. It would be then desirable to avoid single precision problems without going to double precision.

A simplistic approach would be to use double precision for the calculation of $r$ and the range reduction. Unfortunately, switching between single and double precision tends to be slow as GPU architectures are not optimized for such a weird task. Moreover, we still need double precision for a substantial part of the calculation.

We have seen that the precision problem arises when $z_0$ is big. On the other hand, in a big distance, we can use paraxial parabolic (Fresnel) approximation

$$\exp(\mathrm{j}2\pi\sqrt{x^2+y^2+z_0^2}/\lambda) \approx \exp(\mathrm{j}2\pi z_0/\lambda)\exp\left(\mathrm{j}\pi\frac{x^2+y^2}{z_0\lambda}\right) \tag{4}$$

provided that $x, y \ll z_0$, or more precisely (according to [6]) $z_0 \gg [\pi(x^2+y^2)^2_{\max}/4\lambda]^{1/3}$.

It is worth noting that there are no cancellation problems in Eq. (4). First of all, there is no summation of a large $z_0$ and small $x$, $y$ in the argument of exp() function; recall that the most serious error appears when floating-point value of $x^2+y^2+z_0^2$ is almost the same as floating-point value of $z_0^2$. In Eq. (4), arguments dependent on $z_0$ and $x$, $y$ are separated to separate exponential functions, i.e. their arguments are range reduced independently. There is no other modulo or subtraction operation prone to cancellation in Eq. (4). Visualization of the calculation in Fig. 5 therefore shows no significant sign of numerical problems except, of course, slightly different pattern due to $z_0$ quantization in single precision (i.e. $z_0$ in single precision $\neq z_0$ in

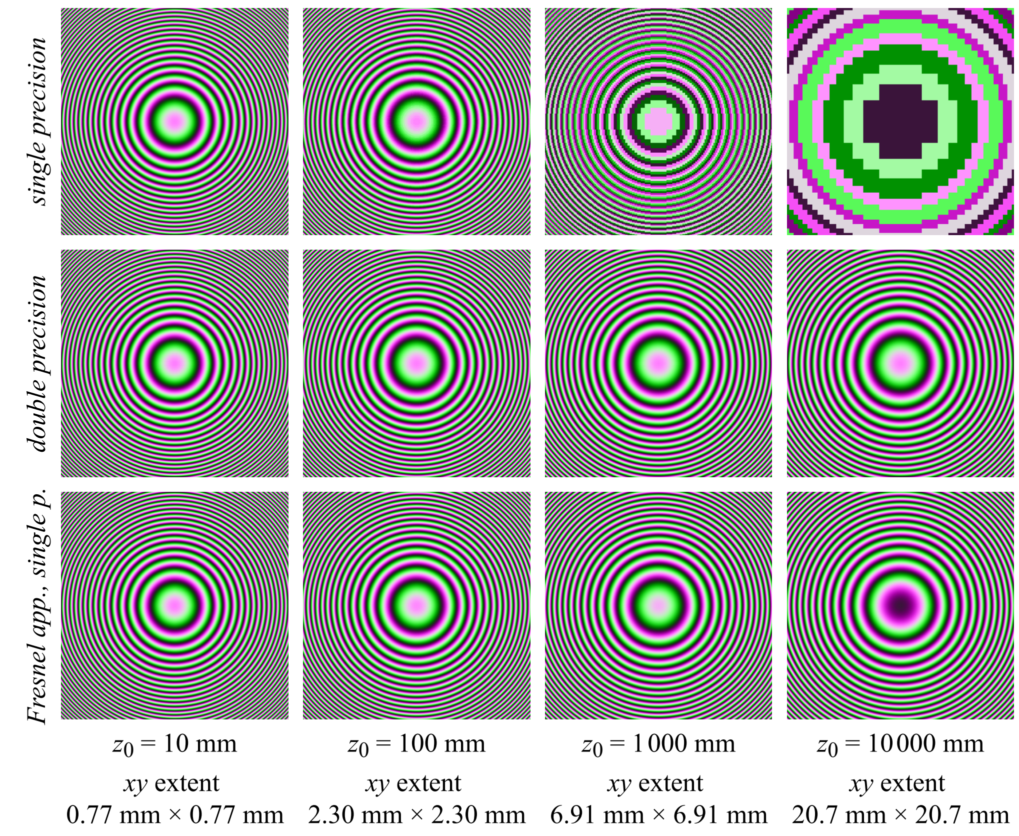


Fig. 5. Comparison of the phasor $\exp(\mathrm{j}2\pi r/\lambda)/r$ for various distances $z_0$ in single and double precision and in the Fresnel approximation in single precision. The lateral extent was chosen so that the images contain approximately the same number of fringes. The color coding is the same as in Fig. 3. The phase in the center of the image for Fresnel approximation, $z_0 = 10000$mm is different from double precision calculation because such a big $2\pi z_0$ cannot be represented in single precision well enough.

double precision). For convenience, Fig. 2 also shows various “Fresnel approximation safe” zones.

In an off-axis case, where the area of interest is located around a point $[x_0, y_0, z_0]$, it is possible to use parabasal parabolic approximation that approximates

$$\sqrt{(x+x_0)^2+(y+y_0)^2+z_0^2} \approx r_0 + \frac{xx_0+yy_0}{r_0} + \frac{x^2+y^2}{2r_0},$$

where $r_0 = (x_0^2+y_0^2+z_0^2)^{1/2}$, provided that $x, y \ll r_0$.

The only problem remains if it is necessary to evaluate $\exp(\mathrm{j}2\pi r/\lambda)$ over a large area, i.e. we cannot use parabolic approximation. It should be possible to use partial quadratic approximation of the square root [7], but we did not verify it.

## 7. Additional notes

We have mentioned in Sec. 4 that setting $n = 2$ in inequality (2) leads to an approximate value of the upper limit of $z_0$; beyond that limit, information about structure of the function $\exp(\mathrm{j}2\pi r/\lambda)$

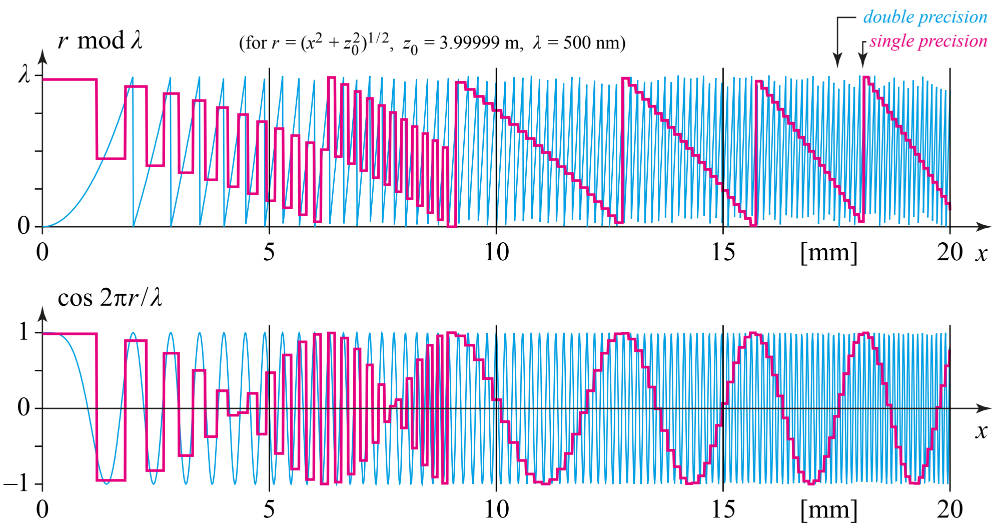


Fig. 6. A closer look at calculation $\exp(j2\pi r/\lambda)/r$ for $z_0 = 3.99999$ in single and double precision. The upper graph shows "range reduction" of the argument, i.e. $(2\pi r/\lambda)$ mod $2\pi$, which is basically the same as $2\pi(r \bmod \lambda)$. The lower graph shows the real part of $\exp(j2\pi r/\lambda)$. The distance $z_0$ was chosen so that the transition between "correct" and "wrong" calculations can be easily seen; in this case, the transition appears near $x = 9$ mm.

is lost. Let us try to find experimentally an example of precise $z_0$ where the effect of "information loss" can be seen.

Figure 6 shows evaluation of the function for $z_0 = 3.99999$ m. The upper part shows the result of range reduction in both single and double precision. It is clear that for $x < 9$ mm the evaluation performs "well" – at least in a way that single precision calculation returns two distinct values in a single cycle. In other words, the value of $r$ is quantized to just two levels in each cycle.

The lower part of the figure shows the result of $\cos(2\pi r/\lambda)$. It is no surprise that for $x < 9$ mm, the single precision result contains the same frequency as the double precision one. The effect for $x > 9$ mm would be normally called *aliasing* [8] as a high frequency content is misinterpreted as a low frequency content. In this case, *aliasing is a direct consequence of quantization of floating point numbers*.

The sampling theorem [6] states that the signal can be recovered if we have more than two samples per cycle. It is then worth asking if e.g. calculation for $z = 2.2$ m (see Fig. 7) is good enough. Let us then try to calculate (in single precision) a complex optical field on the plane $z = z_0$ of a single point source located at the origin and calculate the back-propagation (in double precision) to the plane $z = 0$. Figure 7 shows the results for various $z_0$; we have used the filtered Rayleigh-Sommerfeld convolution [9] for propagation calculation. It is clearly seen that even optical fields heavily damaged by errors of evaluation in single precision, but in the "safe zone of single precision", perform very well. Naturally, optical fields for $z_0$ beyond its upper limit are irreparably damaged.

This result is by no means surprising. Floating-point calculations lead to quantization of the argument of $\exp(j2\pi r/\lambda)$, and therefore to phase quantization of the optical field. The effects of phase quantization are shown in e.g. [10]; it is known that even coarsely quantized phase does not completely damage information "stored" in an optical field.

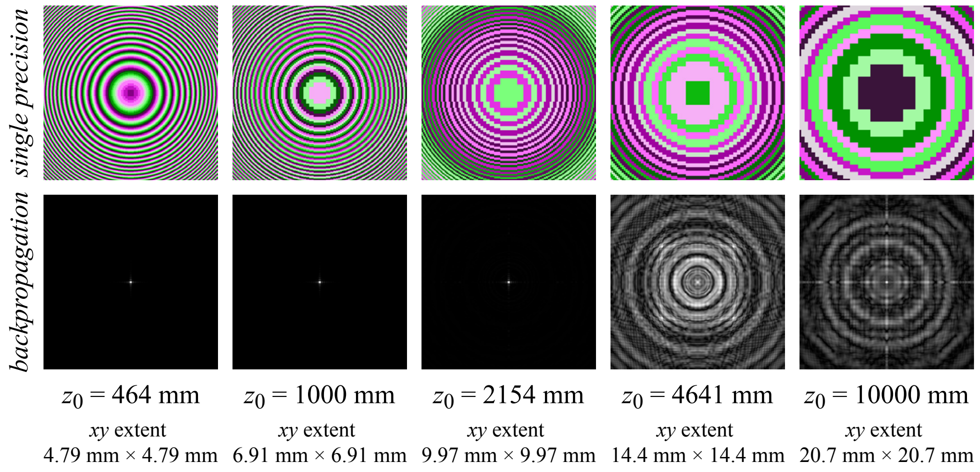


Fig. 7. Upper row: phasor of the optical field in the plane $z = z_0$ of a single point light source located at the origin. The lateral extent was chosen so that the images contain approximately the same number of fringes. Please note that fringes are damaged for $z_0 = 4641$ mm and $z_0 = 10000$ mm due to single precision calculation, not due to incorrect sampling rate. Lower row: normalized intensity of the light field backpropagated to the plane $z_0 = 0$. The images should contain just a single bright dot. Naturally, the last two images are damaged as the optical field was calculated incorrectly.

## 8. Conclusions

We have shown that unwary use of single precision floating-point numbers in calculations of free space light propagation can lead to significant errors or even completely pointless results, such as those presented in Sec. 2. The errors stem from the calculation of $(r \bmod \lambda)$, where $r$ is a distance; serious precision problems can appear for approximately $r > 10^6\lambda$, the results are pointless for approximately $r > 10^7\lambda$. We have also shown that in certain cases, Fresnel approximation or similar one can significantly improve range where single precision calculations remain valid. For example, wrong results shown in Sec. 2 can be fixed using Fresnel approximation or at least predicted using analysis in Sec. 5. We have also shown that double precision calculations are safe in most practical situations.

While the analysis of computer arithmetic may seem to be rather technical and unrelated to optics, we believe the opposite is true. There are many approximations that are used in optics, e.g. scalar approximation of light, Fresnel approximation of light propagation, etc. Applicability of such "classical" approximations are widely studied using "classical" tools such as mathematical analysis; results are obvious for practical optics calculations. As most calculations today are performed with computer approximation of arithmetic, we believe that analysis of computer arithmetic issues should belong to standard mathematical toolbox of an optician. This is especially true because GPU calculations are so popular today and limited precision numbers are usually necessary in massively parallel calculations.

## Acknowledgements

This work was supported by the European Regional Development Fund (ERDF), project "NTIS – New Technologies for the Information Society", European Centre of Excellence, CZ.1.05/1.1.00/02.0090, and by Ministry of Education, Youth, and Sport of Czech Republic – University spec. research – 1311.

# Chapter 7

# Double look-up table for light propagation calculations

Chapters 3, 4 and 5 described advanced methods for light propagation calculations. Main contributions of the chapters and the articles within are, however, not the methods themselves, but the reasoning behind them. Although the methods are correct, they tend to be slow.

Chapter 6 described problems with light propagation calculations in a limited precision environment. It showed that serious problems can be expected in certain (realistic) cases with single precision calculations, which are favoured on GPU's.

To overcome both problems, I have proposed a method that pre-calculates two look-up tables in the double precision (or higher), and the subsequent light propagation calculations do not require high precision. The method takes advantage of the radial symmetry of the functions used in light propagation calculations. The proposed *double look-up table method* was designed for two scenarios, both widely used in digital holography / computer generated display holography.

The first one is the same as discussed in Chapters 3, 4 and 5 – propagation of light between parallel planes. There, the lengthy calculation of the convolution kernel is accelerated by utilizing its symmetry. It should be, however, noted that the double-lookup method does not directly provide exactly the same result as the reference method described in Chapter 4 as the filtered kernel is not radially symmetric, see for example left column of Figure 5.14 on page 85. The almost rectangular shape of the convolution kernel in the spatial domain is caused by frequency filtering of some function using a separable filter. If we had used a nonseparable low-pass filter, it would be possible to get a perfectly radially symmetric function in the spatial domain. However, the difference between results of the reference method and the double look-up table method is acceptable in practice.

The second scenario for the double look-up table method is calculation of light propagation from a cloud of point light sources to a plane. This, in fact, is a starting point of many methods of computer generated holography.

The method was presented at the 10th International Symposium on Display Holography (ISDH 2015), 28 June–3 July 2015, Russian Academy of Sciences, St. Petersburg, Russia. The organizer surprisingly requested just a four pages long summary [39] to the proceedings (for the first time in history of this conference). The short summary is included here for convenience as Appendix A. In this chapter, I included the full version of the article that I intended to publish there.

The full version of the article had to be distributed with supplementary material – full derivation of some import formulas, time measurement of the method and some additional scripts. The derivation is included here as Appendix B, the measurement as Appendix C.

# Double lookup table method for fast light propagation calculations

Petr Lobaz

July 28, 2015

## Abstract

A method of rapid and robust calculation of radially symmetric functions is presented. It can be used for fast computer generated hologram calculation, light propagation calculation, etc. It is suitable for CPU, GPU or hardware implementation.

## 1 Introduction

In computer generated holography and digital holography, it is often necessary to evaluate radially symmetric functions of two variables. They include most notably convolution kernels for free space light propagation calculation, such as Rayleigh-Sommerfeld or Fresnel convolution kernels, free space transfer function in angular spectrum propagation method or lens phase shift function [3]. Many researchers try to accelerate their evaluation, as it takes significant time in the whole calculation of e.g. a computer generated hologram of a 3-D scene or in light propagation simulations.

There are several approaches to the acceleration of such evaluation. First of all, a moderately complicated formula, such as the Rayleigh-Sommerfeld convolution kernel

$$\begin{gathered} K_{\mathrm{RS}}(x,y;z_0) = -\frac{1}{2\pi}\left(\mathrm{j}k - \frac{1}{r}\right)\frac{\exp(\mathrm{j}kr)}{r}\frac{z_0}{r} \\ r = \sqrt{x^2+y^2+z_0^2} \end{gathered} \tag{1}$$

can be approximated by a much simpler formula, such as the Fresnel approximation [3]

$$K_{\mathrm{Fresnel}}(x,y;z_0) = \exp\left(\mathrm{j}k\frac{x^2+y^2}{2z_0}\right)\frac{\exp(\mathrm{j}kz_0)}{\mathrm{j}\lambda z_0}. \tag{2}$$

These convolution kernels are used to calculate monochromatic light propagation from a plane $z = 0$ to a plane $z = z_0$; $\lambda$ stands for a wavelength, $k = 2\pi/\lambda$ is a wave number, $x$ and $y$ are transverse spatial coordinates and j is the imaginary constant ($\mathrm{j}^2 = -1$). This approach not only accelerates the function evaluation, but moreover it can improve its numerical behaviour [10]. On the other hand, of course, any approximation brings an approximation error, and must be used with caution.

A completely different acceleration method just precalculates the function for every $x$, $y$ and $z_0$ that will be used in subsequent calculations and stores the values in a 3-D look-up table (LUT); due to radial symmetry in $xy$ coordinates, only one quadrant or even octant must be actually saved in the look-up table [5, 1]. Despite of this enhancement, memory requirements of the method are still excessive as it is often necessary to precalculate the function for many $z_0$ values and discretization of $x$ and $y$ coordinates must be usually very fine. This is due to the fact that the evaluated function usually contains high spatial frequencies; some researchers reduce the look-up table size by taking into account that the highest spatial frequency usually depends on $z_0$ [15]. Naturally, the method also does not specify how to precalculate the look-up table efficiently.

One way how to reduce memory requirements is based on separability of certain functions. For example, the Fresnel convolution kernel (2) can be rewritten as

$$K_{\text{Fresnel}}(x, y; z_0) = \left[\exp\left(\frac{\text{j}kx^2}{2z_0}\right)\sqrt{\frac{\exp(\text{j}kz_0)}{\text{j}\lambda z_0}}\right] \times \left[\exp\left(\frac{\text{j}ky^2}{2z_0}\right)\sqrt{\frac{\exp(\text{j}kz_0)}{\text{j}\lambda z_0}}\right].$$

It is easy to see that one does not need a 3-D look-up table; it is sufficient to create one 2-D look-up table for each of the two factors in the equation above, as each factor depends on just two variables [14, 2, 6]. Moreover, if the extents and the sampling distances are the same for $x$ and $y$ coordinates, just one 2-D look-up table is necessary. Naturally, this technique can be used only if the function to be evaluated is separable.

Some researchers try to simplify function evaluation by using recurrence formulas, e.g. [16, 11], i.e. they look for simple formulas how to calculate $K(x_0+\epsilon_x, y_0 + \epsilon_y; z_0)$ using value $K(x_0, y_0; z_0)$ for small $\epsilon_x$ and $\epsilon_y$. It should be noted that influence of computer arithmetic rounding errors is usually poorly analysed and that recurrence formulas tend to produce sequential computer code rather than parallel one.

An original approach to evaluation of radially symmetric function $K$ is based on computer graphics algorithm for circle rasterization [13]; this method creates a 2-D array of values $K(m\Delta_{xy}, n\Delta_{xy}; z_0)$, where $\Delta_{xy}$ is a sampling distance in transverse coordinates and integer indices $m$, $n$ span the area of interest. The method calculates $K(m\Delta_{xy}, 0; z_0)$ in a common way and this value is subsequently "copied" to samples at the same distance from the point $(0, 0; z_0)$. The biggest drawback of the method is its complicated memory access, thus memory caching cannot be used efficiently. Recently, authors proposed a method that overcomes this difficulty using recurrence formulas [12].

Parts of the method we are going to analyse in following sections were independently described by other authors [12, 4, 7]. We will discuss these references after we describe the basic idea of the method in Sec. 2; in short, the method is also based on a pair of 2-D look-up tables and it can be easily used for any function that is radially symmetric in the $xy$ plane. Main contributions of this article are introduction of a new type of a look-up table (we will call it "rhoLUT") and a detailed analysis of interpolation in look-up tables to further reduce their size; it will be discussed mainly in Sec. 3 and 4. Sec. 5 adds details about the second type of look-up table ("waveLUT") used by the method. In Sec. 6 we discuss results of the method and Sec. 7 concludes the article. Some details of the derivations, results analysis etc. were omitted in this text; they can be found in the supplementary material.

## 2 Principle of the method

We explain the method on a basic task in digital holography: monochromatic light propagation in free space. Let us calculate propagation of light from the plane $z = 0$ to the plane $z = z_0$ using Rayleigh-Sommerfeld convolution, i.e.

$$U(x, y; z_0) = U(x, y; 0) \otimes K_{\mathrm{RS}}(x, y; z_0),$$

where $U(x, y, z)$ is a complex amplitude (phasor) of light at a point $(x, y, z)$, $\otimes$ stands for convolution and $K_{\mathrm{RS}}(x, y; z_0)$ is the convolution kernel defined by Eq. (1). In discrete calculation, $x$ and $y$ coordinates have to be sampled with sampling period $\Delta_{xy}$ in finite areas of planes $z = 0$ and $z = z_0$. Let us assume for simplicity that both areas in $z = 0$ and $z = z_0$ share the same square shape of the size $M\Delta_{xy} \times M\Delta_{xy}$, where $M$ is an integer number of samples. If we want to employ fast Fourier transform to calculate the discrete convolution, the kernel must be sampled by at least $(2M - 1) \times (2M - 1)$ samples with the same sampling distance $\Delta_{xy}$ [8]. Let us further assume that the centres of the square areas in $z = 0$ and $z = z_0$ are located at $x = 0$, $y = 0$. Then we need to evaluate $K_{\mathrm{RS}}(m\Delta_{xy}, n\Delta_{xy}; z_0)$ for all integers $m \in [1 - M, M - 1]$, $n \in [1 - M, M - 1]$.

It is easy to see that $K_{\mathrm{RS}}(x, y; z_0)$ depends in fact on $r$ and $z_0$, where $z_0$ is constant and $r = (x^2 + y^2 + z_0^2)^{1/2}$ in this example. Moreover, we can define $\rho(x, y) = (x^2 + y^2)^{1/2}$ and write $r = (\rho^2 + z_0^2)^{1/2}$.

To calculate $K_{\mathrm{RS}}(m\Delta_{xy}, n\Delta_{xy}; z_0)$, it is then necessary to evaluate $\rho = \Delta_{xy}(m^2 + n^2)^{1/2}$ and $K_{\mathrm{RS}}(\rho; z_0)$. It is possible to calculate $\rho$ directly or using a 2-D look-up table; we call this look-up table "rhoLUT"; its construction is simple, as $x$ and $y$ are sampled uniformly. In this particular case, it is even possible to store just one quarter of necessary values, i.e. just for $m \in [0, M-1]$, $n \in [0, M - 1]$.

A simple look-up table cannot be constructed for $K_{\mathrm{RS}}(\rho; z_0)$, as the set of discrete values of $\rho$ does not have uniform structure. Instead, we must rely on some interpolation scheme. If we assume that interpolation is sufficiently precise, we can indeed build a look-up table with values $K_{\mathrm{RS}}(q\Delta_w; z_0)$ where $\Delta_w$ is sufficiently small to capture the structure of $K_{\mathrm{RS}}$ and integer index $q$ spans all possible values of $\rho$, i.e. $q \in [0, \lceil M\sqrt{2}\Delta_{xy}/\Delta_w \rceil]$, where $\lceil \cdot \rceil$ is the "round up" (ceil) operation. We denote this look-up table "waveLUT" because it actually captures wave structure of light.

Even if we calculate $K_{\mathrm{RS}}(m\Delta_{xy}, n\Delta_{xy}; z_0)$ for single $z_0$ and all $m$, $n$, the above mentioned procedure is advantageous. The calculation of $\rho$ is usually simple and must be done for all $m$, $n$ anyway, i.e. for $(2M - 1)^2$ samples. The calculation of $K_{\mathrm{RS}}(m\Delta_{xy}, n\Delta_{xy}; z_0)$ is much more involved, but due to waveLUT it is necessary to evaluate it just for $M\sqrt{2}\Delta_{xy}/\Delta_w$ samples of $\rho$, which is usually a much smaller number. We assume that look-up operations and interpolations are fast, which is usually the case.

It is also indeed possible to incorporate some interpolation to rhoLUT, i.e. to precalculate just values $\rho(r,s) = \Delta_\rho(r^2+s^2)^{1/2}$ for $r$, $s$ in $[0, \lceil M\Delta_{xy}/\Delta_\rho\rceil]$, where $\Delta_\rho \geq \Delta_{xy}$ is sufficiently small. In exchange of one more interpolation we get much smaller rhoLUT table. The scheme of calculation is depicted in Fig. 1.

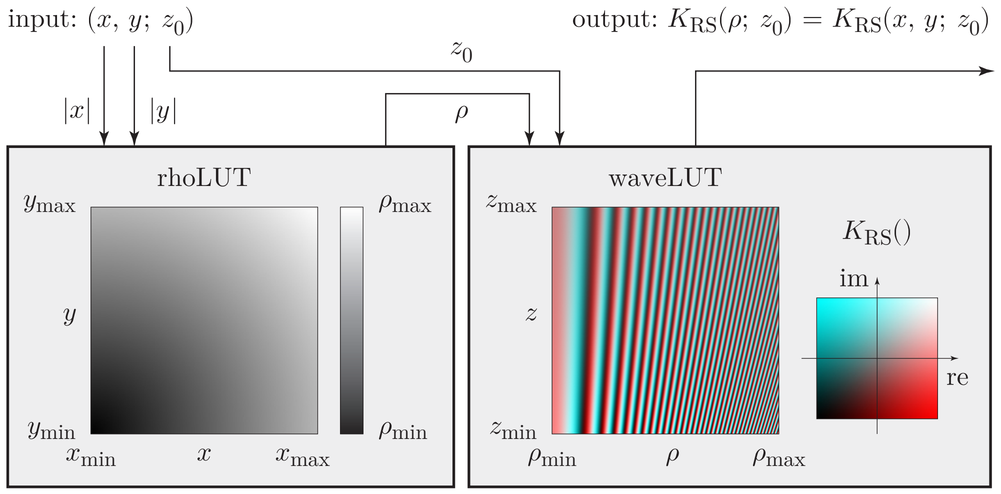


Figure 1: Calculation of radially symmetric function $K_{\mathrm{RS}}(x,y;z_0)$ using 2-D look-up tables rhoLUT and waveLUT. Complex value $(a+b\mathrm{j})$ in the waveLUT is displayed as a RGB color $[a,b,b]$, where $a$ and $b$ are scaled to fit 0–255 range.

The authors of [7] used a very similar method. Instead of rhoLUT, they calculated a look-up table directly for $r = (x^2+y^2+z_0^2)$, so they could not reuse it for any other $z_0$ coordinate as in the case of rhoLUT. Moreover, sampling distance of this table was set to $\Delta_{xy}$, and thus keeping this table for every $z_0$ would be memory inefficient.

The authors of [4] use the table we call waveLUT here (they call it BPP_phase) and although they in fact use rhoLUT as well (due to Matlab style of matrix manipulation, see PFT_distance if Fig. 4 of [4]), they do not discuss it explicitly, nor do they investigate its influence.

The authors of [12] use waveLUT as well, but instead of rhoLUT, they use recurrence formulas to find values of $\rho$. They also do not discuss the influence of interpolation on calculation precision.

## 3 Introduction of the rhoLUT

The aim of the rhoLUT is to replace the calculation of $\rho(x,y) = (x^2+y^2)^{1/2}$, $x \in [0, M\Delta_{xy}]$, $y \in [0, M\Delta_{xy}]$, by a look-up operation, optionally followed by interpolations. Let us define it as a 2-D array

$$\mathrm{rhoLUT}[m,n] = \Delta_\rho\sqrt{m^2+n^2}$$

for integer indices $m$, $n$ in $[0, \lceil M\sqrt{2}\Delta_{xy}/\Delta_\rho\rceil]$. It is naturally possible to generalize the rhoLUT for other index or $xy$ range, but we want to keep the analysis simple.

First of all, we should ask if such a look-up table is practical. In a general computational environment, such as a modern CPU, it depends on other circumstances. Floating point operations are provided here with sufficient precision (usually IEEE 754 double precision or better) and are quite fast; a large look-up table requiring random access to the main memory may suffer from high traffic on the bus. On the other hand, clever caching and small look-up table can easily outperform direct calculation, as especially square root operation is significantly slower than memory access.

Once we move towards more special architectures, such as GPUs or even specially designed FPGAs, reasons to use look-up table become stronger. Architecutures designed for massively parallell computing tend to keep computational cores as simple as possible. This leads to both limited instruction set and limited precision. Direct calculation forces every core to have addition, multiplication and square root operations; look-up table approach requires just memory access plus addition and multiplication for interpolations. Limited precision is even more severe – it is easy to see that calculation of $\sqrt{x^2}$ leads to loss of about half of mantissa bits. Single precision calculation (mantissa length 24 bits) is thus at the edge of applicability for wave optics calculations [10]; lower precision arithmetic such as half precision is out of the question. Contrary to direct $\rho$ calculation, a look-up table precalculated in a high precision environment can be easily used in limited precision enviromnemt.

Finally, as rhoLUT implementation is very easy, it is often possible to implement both direct calculation and look-up table and to choose the faster approach either in advance or at runtime. In conclusion, it is advantageous to implement rhoLUT.

Let us now examine the rhoLUT details – its sampling (or size) and the interpolation of values.

In the simplest case, sampling distance of the rhoLUT equals sampling distance in $x$, $y$ coordinates, i.e. $\Delta_\rho = \Delta_{xy}$, and $\rho$ calculation for $x = m\Delta_{xy}$, $y = n\Delta_{xy}$ is trivial:

$$\begin{aligned}\rho_C(x,y) &= \text{rhoLUT}\big[\,\text{round}(|x|/\Delta_\rho), \text{round}(|y|/\Delta_\rho)\big]\\ &= \text{rhoLUT}\big[|m|, |n|\big].\end{aligned}$$

We can indeed use the same equation even if $\Delta_\rho \neq \Delta_{xy}$; now, $\rho(x,y)$ is approximated by a piecewise constant ("staircase") function $\rho_C(x,y)$. It is easy to see that approximation error is at most $\Delta_\rho\sqrt{2}/2$ and the structure of the error is the same in the whole $xy$ plane. This observation is important: in the next step, we are going to use $\rho$ as an index to the waveLUT. However, local frequency of the function sampled by the waveLUT, e.g. $K_{\text{RS}}(\rho; z_0)$, usually grows as $\rho \to \infty$; a small error in $\rho$ will be thus magnified for big $x$, $y$. Piecewise constant approximation must be therefore used with the utmost caution.

As $\rho(x,0) = |x|$, it follows that the linear interpolation between rhoLUT values gives exact results for $y = 0$. We should then expect that in a general case, bilinear interpolation should give quite precise results:

$$\begin{aligned} \rho_B(x,y) &= (1-i_x)(1-i_y)\rho_{00} + (1-i_x)i_y\rho_{01} + i_x(1-i_y)\rho_{10} + i_x i_y \rho_{11}, \\ i_x &= \frac{|x|}{\Delta_\rho} - \left\lfloor \frac{|x|}{\Delta_\rho} \right\rfloor, \qquad i_y = \frac{|y|}{\Delta_\rho} - \left\lfloor \frac{|y|}{\Delta_\rho} \right\rfloor, \\ \rho_{st} &= \text{rhoLUT}\big[\lfloor |x|/\Delta_\rho \rfloor + s, \lfloor |y|/\Delta_\rho \rfloor + t\big]. \end{aligned}$$

It can be easily found that the approximation error $\rho_B(x,y) - \rho(x,y)$ vanishes for $x \to \infty$, $y \to \infty$, and its maximum is $\Delta_\rho(2-\sqrt{2})/4 \approx 0.146\Delta_\rho$; the maximum error is located at $(x,y) = (\Delta_\rho/2, \Delta_\rho/2)$, see Fig. 2.

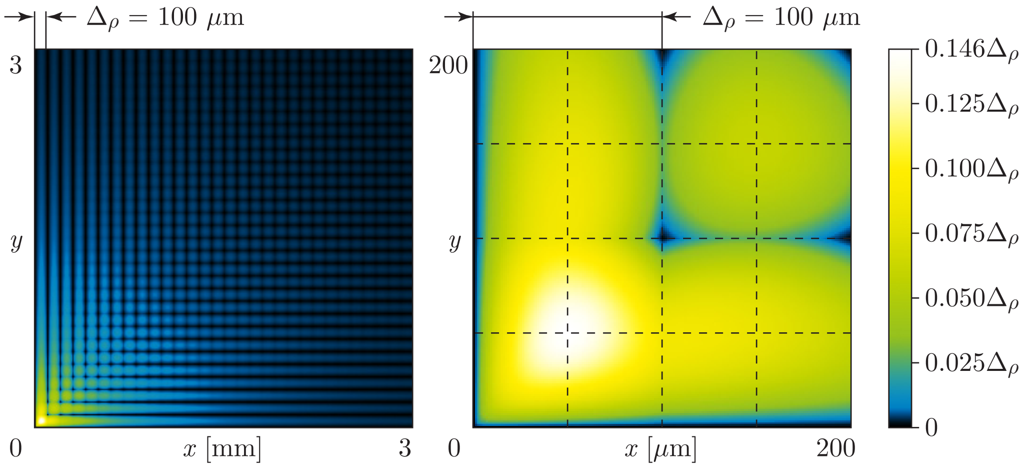


Figure 2: Visualization of $\rho_B(x,y) - \rho(x,y)$, i.e. the error introduced by the rhoLUT with bilinear interpolation. Left graph shows error vanishing, right graph shows that maximum error is located at $(x,y) = (\Delta_\rho/2, \Delta_\rho/2)$. Notice there is indeed no error for $x$, $y$ being integer multiples of $\Delta_\rho$.

## 4 Sampling of the rhoLUT

To select a small enough sampling distance $\Delta_\rho$, it is not sufficient to examine error $\rho_B(x,y) - \rho(x,y)$ alone – we need to take into account that approximate value $\rho_B$ is used in subsequent calculations. Let us assume that we are going to calculate $K_{\text{RS}}(\rho_B; z_0)$ directly, without waveLUT. Effects of waveLUT approximation error will be discussed afterwards.

There are at least two ways how to analyze effect of error in $\rho$ approximation. The first one assumes that the error of $\rho$ is multiplied by the local frequency of $K_{\text{RS}}(\rho; z_0)$; let us explain why. Local frequency $\text{lf}_{\text{RS}}(\rho; z_0)$ is defined as a first derivative of the argument of the high frequency component of $K_{\text{RS}}(\rho; z_0)$ [3]:

$$\text{lf}_{\text{RS}}(\rho_0; z_0) = \frac{\partial}{\partial\rho}\frac{r}{\lambda}\bigg|_{\rho=\rho_0} = \frac{\partial}{\partial\rho}\frac{\sqrt{\rho^2+z_0^2}}{\lambda}\Bigg|_{\rho=\rho_0} = \frac{\rho_0}{\lambda\sqrt{\rho_0^2+z_0^2}}. \tag{3}$$

It tells us how many cycles per unit length can be found in the vicinity of $\rho_0$. Thus, unit error in $\rho$ results in phase change $\mathrm{lf}_{\mathrm{RS}}(\rho; z_0)$ of $K_{\mathrm{RS}}(\rho; z_0)$. It is therefore necessary to analyze the function $[\rho_B(x, y) - \rho(x, y)] \times \mathrm{lf}_{\mathrm{RS}}(\rho(x, y); z_0)$.

The second way leads to the same results and is perhaps more intuitive; let us derive optimal $\Delta_\rho$ in this way. Here, only the main steps of the derivation are presented; see the supplementary material for the full derivation and reasoning.

The evaluation of $K_{\mathrm{RS}}$ as defined in Eq. (1) depends on $r(\rho, z_0) = (\rho^2 + z_0^2)^{1/2}$. If we compare $r$ evaluated with exact $\rho$ and with its approximation $\rho_B$ (using bilinear approximation), we can observe following facts (see Fig. 3):

- The difference $r\big(\rho_B(x, y), z_0\big) - r\big(\rho(x, y), z_0\big)$ approaches 0 as $\rho(x, y) \to \infty$. However, the error decreases quite slowly for $\rho \approx 0$. As the most sensitive term of $K_{\mathrm{RS}}$ is $\exp(\mathrm{j}2\pi r/\lambda)$, we can conclude that the overall error of $K_{\mathrm{RS}}$ also vanishes.
- The biggest error can be found in the vicinity of $(x, y) = (0, 0)$. Moreover, maximum error for $0 < x < \Delta_\rho$, $0 < y < \Delta_\rho$ is either the global maximum of the error, or it is very close to the maximum.
- Although the error of $\rho$ is maximal at $(x, y) = (\Delta_\rho/2, \Delta_\rho/2)$, the error of $r$ is maximal at a different point. However, the error of $r$ at $(x, y) = (\Delta_\rho/2, \Delta_\rho/2)$ is a rough approximate ($\approx 85\%$) of the maximum error.

We can thus define estimate of the maximum error of $r$ when using the rhoLUT with bilinear interpolation as

$$\mathrm{maxRErr}(\Delta_\rho, z_0) = 1.2 \left\{ r\left[\rho_B\left(\frac{\Delta_\rho}{2}, \frac{\Delta_\rho}{2}\right), z_0\right] - r\left[\rho\left(\frac{\Delta_\rho}{2}, \frac{\Delta_\rho}{2}\right), z_0\right] \right\}, \quad (4)$$

where factor 1.2 reflects the observation that the error for $(x, y) = (\Delta_\rho/2, \Delta_\rho/2)$ is approximately 85% of the maximum. Although it is possible to analyze the error more rigorously, in numerical testing we have found that this approximation works well (see Sec. 6) and the factor 1.2 being slightly higher than 1/0.85 leads to a slightly pessimistic error estimate.

It can be easily seen in Fig. 4 that in the log-log graph, maxRErr can be approximated by a line for a wide range of useful errors and $\Delta_\rho$. Please note that $r$ is divided by $\lambda$ in Eq. (1), and thus an acceptable error must be much lower than $\lambda$. For convenience, there is a thick dashed horizontal line in Fig. 4 at maxRErr = 1 $\mu$m, i.e. any acceptable error for visible light calculations must be well below this line.

The linear approximation in a log-log graph is defined as $\log(\mathrm{maxRErr}) \approx \kappa \log(\Delta_\rho) + \xi$, where $\kappa$ is a slope and $\xi$ is an intercept. It is easy to measure that $\kappa \approx 1.9998$ for a wide range of common $z_0$ and acceptable maxRErr. It is also clear that $\xi$ depends on $z_0$ and this dependence is also linear in the log-log graph: $\xi = \xi_0 + \xi_1 \log(z_0)$, where $\xi_0$ is a value for $z_0 = 1$ and $\xi_1$ is the dependency factor. It can be easily measured from the graph that $\xi_0 \approx -1.9886$ and $\xi_1 \approx -0.9999$.

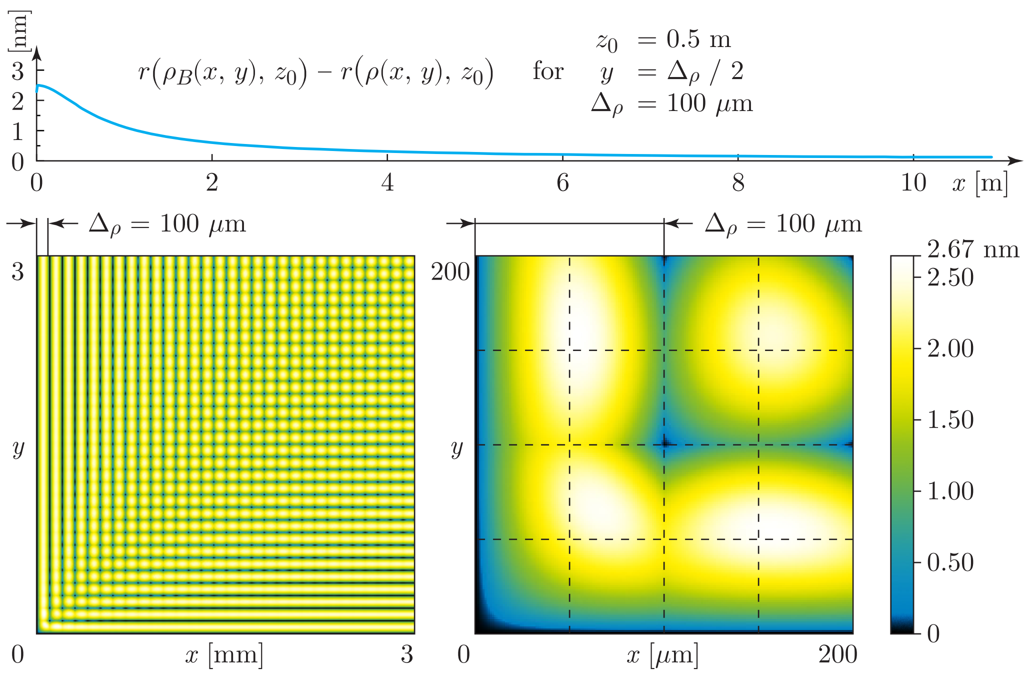


Figure 3: Visualization of $r(\rho_B(x,y),z_0)-r(\rho(x,y),z_0)$, i.e. the error introduced in evaluation of $r = (\rho^2 + z_0^2)^{1/2}$ by a rhoLUT with bilinear interpolation. The top graph shows error vanishing, the bottom left shows the error structure, the bottom right shows that error at $(x,y) = (\Delta_\rho/2, \Delta_\rho/2)$ is a rough approximation of the maximum error. Notice that each graph has different scale .

The linear approximation allows us to find optimal $\Delta_{\rho\ \mathrm{opt}}$ for a chosen maxRErr:

$$\Delta_{\rho\ \mathrm{opt}}(\mathrm{maxRErr}, z_0) \approx \exp\left(\frac{1}{\kappa}\log\left[\frac{\mathrm{maxRErr}}{(z_0)^{\xi_1}\exp(\xi_0)}\right]\right) \quad \text{for} \quad \begin{aligned}\kappa &\approx +1.9998\\ \xi_0 &\approx -1.9886\\ \xi_1 &\approx -0.9999\end{aligned} \tag{5}$$

Example values of $\Delta_{\rho\ \mathrm{opt}}$ for $\lambda = 500$ nm can be found in Fig. 5. For example, propagation to $z_0 = 1$ m can be calculated with rhoLUT prepared with $\Delta_{\rho\ \mathrm{opt}} = 0.19$ mm if we allow error in $r$ calculation $\lambda/100$, which is usually acceptable. If the sizes of the areas in $z = 0$ and $z = z_0$ are $50 \times 50$ mm, the rhoLUT size should be $264 \times 264$ samples. For maximum error $\lambda/10$, we set $\Delta_{\rho\ \mathrm{opt}} = 0.6$ mm and the rhoLUT size is just $84 \times 84$ samples.

We can approximate further to get some insight to Eq. (5). As $\kappa \approx 2$, $\xi_0 \approx -2$ and $\xi_1 \approx -1$, we can approximate Eq. (5) by

$$\Delta_{\rho\ \mathrm{opt}}(\mathrm{maxRErr}, z_0) \approx 2.72\sqrt{z_0\,\mathrm{maxRErr}}. \tag{6}$$

We know that for the fixed extent of $\rho$, the rhoLUT size is proportional to $1/\Delta_\rho^2$ (it is a 2-D look-up table). Therefore Eq. (5) actually tells that the rhoLUT size is inversely related to maxRErr and $z_0$. It is also worth noting that

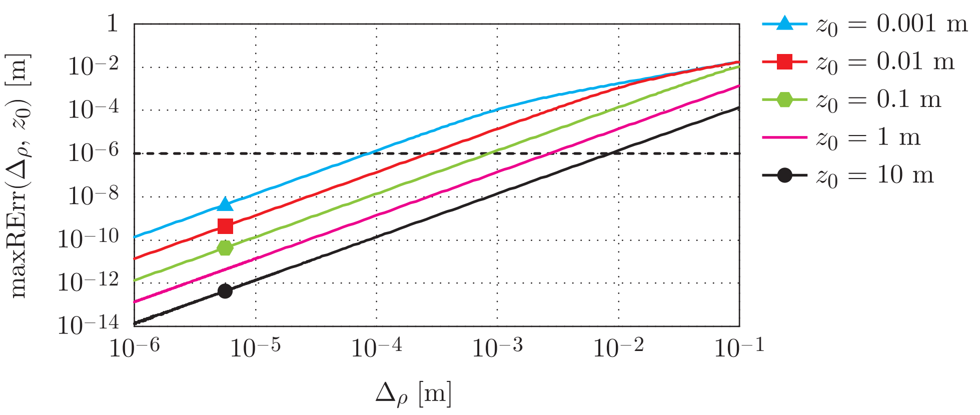


Figure 4: Approximate value of maximum error in calculation of $r$ using rhoLUT with bilinear approximation. Useful part of the curves is below the thick horizontal line for visible light. Please note it is a log-log graph and both axes are in meters.

Eq. (6) can be easily derived from the Taylor expansion of Eq. (4) with respect to $\Delta_\rho$; in this case, the constant factor changes to $\approx 2.70$.

## 5 The waveLUT

Once we have the approximate $\rho$ value, we can calculate $K_{\mathrm{RS}}(\rho; z_0)$ using the waveLUT. We should set small enough $\Delta_w$ and define

$$\mathrm{waveLUT}[q; z_0] = K_{\mathrm{RS}}(q\Delta_w; z_0).$$

where $q$ is an integer index. Again, for a particular value $\rho_0$ we can estimate the value of $K_{\mathrm{RS}}(\rho_0; z_0)$ using piecewise constant approximation

$$K_{\mathrm{RS\ C}}(\rho_0; z_0) = \mathrm{waveLUT}[\mathrm{round}(\rho_0/\Delta_w); z_0] \tag{7}$$

or e.g. piecewise linear approximation

$$\begin{aligned} K_{\mathrm{RS\ L}}(\rho_0; z_0) &= (1-i)\mathrm{waveLUT}\big[\lfloor \rho_0/\Delta_w \rfloor; z_0\big] + i\,\mathrm{waveLUT}\big[\lfloor \rho_0/\Delta_w \rfloor + 1; z_0\big], \\ i &= \frac{\rho_0}{\Delta_w} - \left\lfloor \frac{\rho_0}{\Delta_w} \right\rfloor. \end{aligned} \tag{8}$$

We should discuss following topics: how to deal with interpolation of $z_0$, how to set $\Delta_w$, and what interpolation to use in the $\rho$ direction.

In many applications, only a few values of $z_0$ are necessary or the value of $z_0$ is not critical. For example, in computer generated holography, it is often necessary to calculate the propagation of light from a point light source located at $(x_0, y_0, z_0)$ to a plane $z = 0$, and it is acceptable to quantize $z_0$ quite coarsely. In that case, it is possible to choose a suitable quantization step $\Delta_z$ and to precalculate the waveLUT for every integer multiple of $\Delta_z$ in a given range $[z_{\min}, z_{\max}]$.

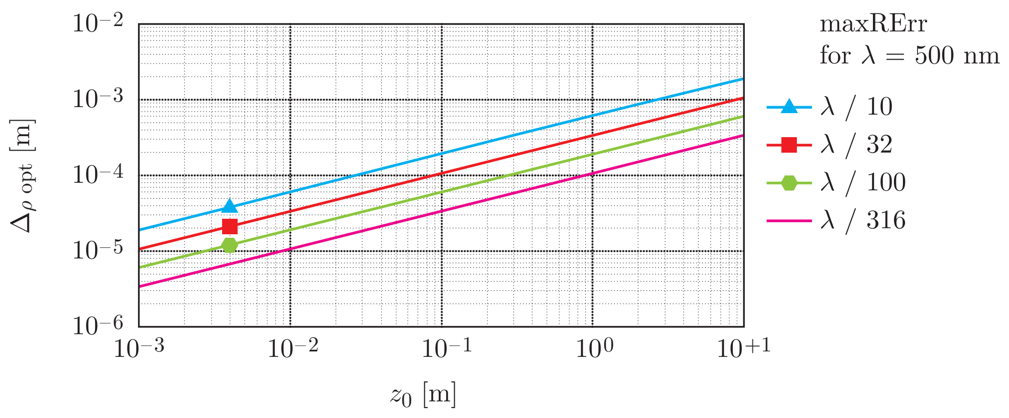


Figure 5: Optimal sampling distance $\Delta_{\rho\ \mathrm{opt}}$ for the rhoLUT calculated using Eq. (5) for $\lambda = 500$ nm. Please note it is a log-log graph and both axes are in meters.

It is also possible to set $\Delta_z$ as an integer multiple of $\lambda$. Now, the waveLUT looks like its visualization in Fig. 1 – notice that the structure in the vertical ($z$) direction is very simple. It should be possible to introduce some form of interpolation in this direction, but we will not discuss it here.

The extent of the waveLUT in the $\rho$ direction is easy to set. Maximum and minimum values of $\rho$ are given by the rhoLUT (or the geometry of the problem). For a given maximum value $\rho_{\max}$, we can calculate the local frequency $\mathrm{lf}_{\mathrm{RS}}(\rho_{\max}; z_0)$ using Eq. (3) and set $\Delta_w < 1/[2\,\mathrm{lf}_{\mathrm{RS}}(\rho_{\max}; z_0)]$ to have at least two samples per cycle of $K_{\mathrm{RS}}(\rho; z_0)$.

Our experiments show that good results are obtained with 8 samples per cycle, i.e.

$$\Delta_{w\ \mathrm{opt}} = 1/[8\,\mathrm{lf}_{\mathrm{RS}}(\rho_{\max}; z_0)] \tag{9}$$

and linear interpolation according to Eq. (8). Piecewise constant approximation (7) leads to much higher phase quantization and subsequently increases noise in the optical field. However, provided that $\rho$ is precise enough, phase quantization itself does not destroy properties of the optical field [10]. We have also tested other interpolation schemes (cubic and various windowed sinc), but experiments show that for our purposes (computer generated holography), slightly higher accuracy of the result does not justify slower calculation.

## 6 Results

We have implemented several tests to measure look-up tables performance, both in speed and in precision. Here we present results of CPU tests; details on GPU implementation are in preparation and will be published elsewhere. Please note that this section presents just summary of the results; the details are presented in the supplementary material to this article.

As shown in [10], direct calculation of highly oscillatory functions such as $K_{\mathrm{RS}}$ is prone to numerical error in single precision calculations. In short, problems appear in calculation of $\cos[2\pi(x^2 + y^2 + z_0^2)^{1/2}/\lambda]$ for visible light and $z_0 \approx 1$ m or bigger. No problems appeared when using a rhoLUT and a waveLUT precalculated in double precision. Direct calculation of $\rho$ (i.e. omission of a rhoLUT) is possible for on-axis propagation calculations; in off-axis cases, calculation of $\rho = (x^2 + y^2)^{1/2}$ in single precision is prone to numerical errors as well. See Fig. 6 for an example of a correct and incorrect kernel calculation.

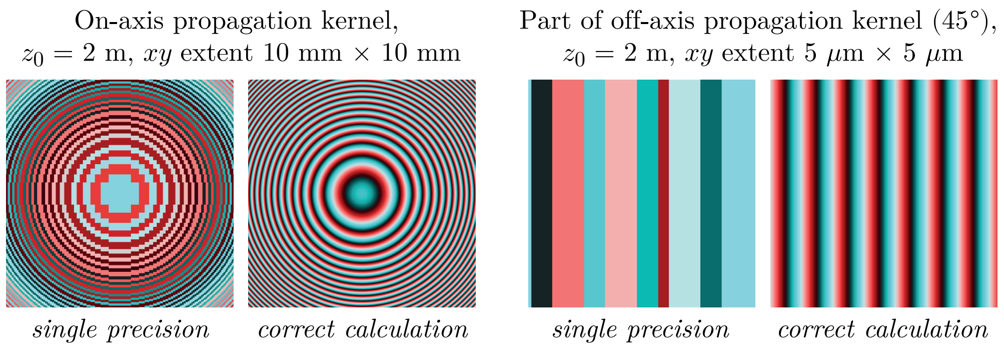


Figure 6: Examples of problems in calculation of $K_{\mathrm{RS}}$. "Single precision" shows influence of rounding errors in IEEE 754 single precision arithmetic, "correct calculation" shows a correct result calculated in double precision. Calculation using look-up tables leads to the same correct results. Color coding of complex values is the same as in Fig. 1.

Tests of $\Delta_\rho$ and $\Delta_w$ selection according to Eqs. (5) and (9) confirmed theoretical analysis. For example, if we decide to accept error maxRErr $= \lambda/100$ in calculation of $r$, $\Delta_\rho$ calculated using Eq. (5) and bilinear interpolation in the rhoLUT leads to an actual error at most $\lambda/103$, i.e. 97% of the desired value. Other values of maxRErr lead to similar numbers.

We have also measured actual error of calculation of $K_{\mathrm{RS}}$. It was calculated as the maximum or the average value of $|K - K'|/K_{\max}$, where $K$ is the value of $K_{\mathrm{RS}}$ calculated precisely, $K'$ is the value of $K_{\mathrm{RS}}$ calculated using look-up tables and $K_{\max}$ is the maximum of $K_{\mathrm{RS}}$ in the whole calculated area; the error was evaluated for both real and imaginary parts and the higher (worse) value is presented here.

Choosing 8 samples per fringe in selection of $\Delta_w$ according to Eq. (9) and linear interpolation in the waveLUT leads to maximum error 7.0% (1.1% on average) in the final propagation kernel; 16 samples per fringe to 1.9% (0.3% on average); 32 samples per fringe to $\approx 0.46\%$ (0.07% on average). While these numbers may seem high, it should be noted that the maximum error appears in the finest fringes and does not affect their frequency; thus, this error has a negligible impact on the optical field properties. Combination of the rhoLUT and the waveLUT naturally further increases the error; typical values of maxRhoErr $= \lambda/100$ and 8 samples per fringe lead to the maximum error 8.8% (2.8% on average), which is perfectly acceptable for our purposes.

Calculation time was tested in realistic geometric scenarios for propagated areas sampled by $64 \times 64$ samples to $2048 \times 2048$ samples. We have prepared two test cases – one for complicated filtered propagation kernels (see [9]) where look-up tables should clearly win over direct calculation, and one for simplified Rayleigh-Sommerfeld kernel (without $-1/r$ term) where even direct calculation is quite fast. Naturally, larger kernels benefit from look-up tables as their preparation takes comparatively smaller part of calculation time.

Complicated filtered propagation kernel calculation is accelerated mainly by utilizing waveLUT; $10\times$ or $100\times$ faster calculation is easily achieved, depending on the complexity of the kernel. Using waveLUT in simple kernel calculation leads to about $1.7\times$ faster calculation. These numbers include the waveLUT calculation, which takes about 1% of the overall time.

Introducing rhoLUT enhances numerical behaviour in single precision environment; in double precision environment (CPU), this advantage is not important, as $\rho$ can be easily evaluated directly. Unoptimized implementation of the rhoLUT can actually double calculation time compared to direct $\rho$ calculation and the waveLUT. A slightly optimized rhoLUT implementation is approximately as fast as direct $\rho$ calculation – this optimization exploits the fact that some values used in the rhoLUT interpolation are constant within a single row (or column) of the rhoLUT. On the other hand, careful rhoLUT implementation leads to further 20% to 40% speed-up compared to to direct $\rho$ calculation and the waveLUT. This optimization sets $\Delta_\rho$ to an integer multiple of $\Delta_{xy}$ and uses integer arithmetic whenever possible.

## 7 Conclusion

We have introduced a method of calculation of arbitrary radially symmetric functions using a pair of look-up tables, a rhoLUT and a waveLUT. While using a waveLUT is always advantageous, using a rhoLUT has its pros and cons. The rhoLUT enhances numerical behaviour in a limited precision environment (such as GPU); in a high precision environment, it must be carefully implemented to improve the speed of calculation. We have also analyzed selection of look-up tables parameters and their influence on the calculation precision.

## Acknowledgments

This work was supported by the European Regional Development Fund (ERDF), project NTIS New Technologies for the Information Society, European Centre of Excellence, CZ.1.05/1.1.00/02.0090, and by Ministry of Education, Youth, and Sport of Czech Republic University spec. research 1311. I would like to thank Petr Vaněček for collaboration on implementation and testing and Libor Váša for valuable comments and help with manuscript preparation.

## References

[1] Sungjin Cho, Byeong-Kwon Ju, Nam-Young Kim, and Min-Chul Park. One-eighth look-up table method for effectively generating computer-generated hologram patterns. *Optical Engineering*, 53(5):054108, 2014.

[2] G. Bora Esmer. Fast computation of fresnel diffraction field of a three-dimensional object for a pixelated optical device. *Appl. Opt.*, 52(1):A18–A25, Jan 2013.

[3] Joseph W. Goodman. *Introduction to Fourier Optics*. Roberts & Company Publishers, 3 edition, December 2004.

[4] Yingqing Huang, Kai Zhao, Xingpeng Yan, Chuang Pei, and Xiaoyu Jiang. Phase-searching look-up table method for computer-generated holograms. *Journal of Electronic Imaging*, 23(4):043013, 2014.

[5] Seung-Cheol Kim and Eun-Soo Kim. Effective generation of digital holograms of three-dimensional objects using a novel look-up table method. *Appl. Opt.*, 47(19):D55–D62, Jul 2008.

[6] Seung-Cheol Kim, Jae-Man Kim, and Eun-Soo Kim. Effective memory reduction of the novel look-up table with one-dimensional sub-principle fringe patterns in computer-generated holograms. *Opt. Express*, 20(11):12021–12034, May 2012.

[7] S. Lee, H. C. Wey, D. K. Nam, D. S. Park, and C. Y. Kim. Fast hologram pattern generation by radial symmetric interpolation, 2012.

[8] Petr Lobaz. Reference calculation of light propagation between parallel planes of different sizes and sampling rates. *Opt. Express*, 19(1):32–39, Jan 2011.

[9] Petr Lobaz. Memory-efficient reference calculation of light propagation using the convolution method. *Opt. Express*, 21(3):2795–2806, Feb 2013.

[10] Petr Lobaz and Petr Vaněček. Safe range of free space light propagation calculation in single precision. *Opt. Express*, 23(3):3260–3269, Feb 2015.

[11] Kyoji Matsushima and Masahiro Takai. Recurrence formulas for fast creation of synthetic three-dimensional holograms. *Appl. Opt.*, 39(35):6587–6594, Dec 2000.

[12] Takashi Nishitsuji, Tomoyoshi Shimobaba, Takashi Kakue, and Tomoyoshi Ito. Fast calculation of computer-generated hologram using run-length encoding based recurrence relation. *Opt. Express*, 23(8):9852–9857, Apr 2015.

[13] Takashi Nishitsuji, Tomoyoshi Shimobaba, Takashi Kakue, Nobuyuki Masuda, and Tomoyoshi Ito. Fast calculation of computer-generated hologram using the circular symmetry of zone plates. *Opt. Express*, 20(25):27496–27502, Dec 2012.

[14] Yuechao Pan, Xuewu Xu, Sanjeev Solanki, Xinan Liang, Ridwan Bin Adrian Tanjung, Chiwei Tan, and Tow-Chong Chong. Fast cgh computation using s-lut on gpu. *Opt. Express*, 17(21):18543–18555, Oct 2009.

[15] Tomoyoshi Shimobaba, Hirotaka Nakayama, Nobuyuki Masuda, and Tomoyoshi Ito. Rapid calculation algorithm of fresnel computer-generated-hologram using look-up table and wavefront-recording plane methods for three-dimensional display. *Opt. Express*, 18(19):19504–19509, Sep 2010.

[16] Hiroshi Yoshikawa, Susumu Iwase, and Tadashi Oneda. Fast computation of fresnel holograms employing difference, 2000.

# Chapter 8

# Conclusions and future work

This thesis presented several ideas applicable in monochromatic coherent light propagation calculations. I consider some of them theoretically important, the others are more practically oriented.

In calculation of light propagation, a weird result is obtained sometimes. It is usually attributed to aliasing of the convolution kernel, either in the spatial domain (in the convolution based methods) or in the frequency domain (in the angular spectrum decomposition based methods). I have shown in Chapters 4 and 5 several theoretically important facts.

- Aliasing is not bad per se. The results that are considered wrong are often physically perfectly correct, although the input data have to be interpreted differently. For example, sometimes the result should be interpreted as interference pattern of several point light sources rather than propagation of a continuous light field.

  Moreover, if we are interested in light intensity only, aliased complex amplitudes of the calculated light field can provide superb result. On the other hand, anti-aliasing techniques can damage the intensity information significantly.

- Two methods of calculation of light propagation between parallel planes, the convolution method and the angular spectrum decomposition method, are equivalent in the continuous domain. They are, however, not considered equivalent in the discrete domain, i.e. in numerical calculations. The convolution method is preferred for large propagation distances, while the angular spectrum decomposition is preferred for short propagation distances. I have shown that this discrepancy stems from improper discretisation of the methods; they are equivalent in the discrete domain when discretised properly.

  However, using the convolution method for short propagation distances leads to excessively long calculation times, and the opposite holds for the angular spectrum decomposition method. Thus, the common preferences "angular spectrum for short" and "convolution for long" are still valid.

- While using the angular spectrum decomposition method for short propagation distances is considered safe, I have shown there are still some limitations. They can be easily explained as follows. Numerical calculation of the angular spectrum method samples the transfer function inside a rectangular area of the frequency plane only. It thus assumes it is spatially limited (has finite support) in the frequency domain. However, a function spatially limited in the frequency domain is spatially unlimited in the spatial domain (has infinite support there). At the same time, the input light field has to be considered periodic due to nature of the discrete Fourier transform. These two fact combine and say that any sample of the result is influenced by periodic copies of the input light field.

  It should be, however, emphasised that the described influence is usually negligible, thus for most practical situations the angular spectrum decomposition can be used as usual.

- Many methods of light propagation based on the angular spectrum decomposition method use techniques to suppress aliasing of the transfer function. They, however, do not discuss physical meaning of the aliasing suppression. I have shown that a proper discrete calculation requires spatially limited convolution kernel in the spatial domain, i.e. the impulse response must have finite support. It means that the transfer function has to be frequency limited. Thus, suppression of aliasing of the transfer function can be easily interpreted in terms of limiting support of the impulse response.

Chapter 3 further discussed the convolution method of calculation of light propagation between parallel planes. I have shown its limits, have analysed the zero-padding technique thoroughly and have shown a way how to limit its memory consumption. Finally, I have described a method how to utilize the convolution method when the sampling distances of the input and output light fields are not the same.

It should be noted that there are other methods of light propagation calculations that allow to use different sampling distances of the input and output light fields, that try to be memory efficient, etc. The enhancements to the convolution method I proposed aim to create a reliable, reference method that can be used to test results of other methods.

Chapter 6 described influence of floating point arithmetic in discussed methods of light propagation. I have shown that calculating the term $\exp(\mathbf{j}\,kr)$, that is very common in light propagation calculations, is very sensitive to rounding errors of floating point arithmetic. (Here, $r$ is a distance between points in space, $k$ is the wave number and $\mathbf{j}^2 = -1$). Serious problems have to be expected when $r$ is in order of meters. Moreover, I have shown that the problems often disappears when $r$ is calculated using the first order (Fresnel) approximation.

Finally, I have proposed a method in Chapter 7 that both accelerates the calculation of light propagation between parallel planes and limits influence of rounding errors significantly. Moreover, the method can be used for calculation of light propagation between a cloud of point light sources and a plane, which is a base of many methods of computer generated display holography.

### Future work

The future work can be divided to three areas: 1. what can be done, 2. what should be done and 3. what am I currently doing.

1. I have described in Chapter 3 that calculation of light propagation between parallel planes can be partitioned to smaller tasks; splitting leads to smaller memory consumption and sometimes to faster calculation. It is, however, not known how to choose the best partition.

   The article presented in Chapter 3 does not address the situation where the convolution kernel has smaller support than defined by the method. Such a situation appears for example when propagation distance is small and the convolution kernel was properly filtered. Taking this into account opens new possibilities in accelerating the method. For example, overlap-and-add convolution might result in fast algorithm for very small propagation distances.

2. There are methods of light propagation calculation based on single Fourier transform, especially the methods based on the Fresnel approximation. Their discretisation should be revisited with tools described in Chapters 4 and 5. Then there are methods that use more sophisticated reasoning such as the band-limited double-step Fresnel diffraction [40]. Their discretisation should be revisited as well.

   Other widely used algorithm in digital holography / computer generated display holography is the optical field rotation, see for example [41, 42, 43, 44]. Their reasoning is completely based on the continuous formulation. As effects of discretisation are not discussed there, it would be more than useful to explore them.

3. The double look-up method presented in Chapter 7 was implemented on GPU by Petr Vaněček (co-author of [37] presented in Chapter 6). We are going to evaluate it and utilize it in light propagation calculations. I have also derived equations that allow to use the double look-up table method in calculations based on the angular spectrum decomposition. It should be well tested before publishing the results.

I am also working on an algorithm that actually produces a computer generated display hologram of a synthetic scene. Almost everything from this thesis is utilized there.

# Chapter 9

# Activities

## 9.1 List of author's publications

**Conference papers (in order of importance)**

- Lobaz, P. "Discrete calculation of the off-axis angular spectrum based light propagation". In: *Journal of Physics: Conference Series.* Vol. 415. 1. 2013, p. 012040
- Lobaz, P. "Double lookup table method for fast light propagation calculations". In: *Proceedings of the 10th International Symposium on Display Holography.* 2015, in press

**Journal papers (in order of importance)**

- Lobaz, P. "Memory-efficient reference calculation of light propagation using the convolution method". In: *Opt. Express* 21.3 (Feb. 2013), pp. 2795–2806. DOI: `10.1364/OE.21.002795`. Total citations (February 2017): 4 (including 1 self citation).
- Lobaz, P. "Reference calculation of light propagation between parallel planes of different sizes and sampling rates". In: *Opt. Express* 19.1 (Jan. 2011), pp. 32–39. DOI: `10.1364/OE.19.000032`. Total citations (February 2017): 9 (including 3 self citations).
- Lobaz, P. and Vaněček, P. "Safe range of free space light propagation calculation in single precision". In: *Opt. Express* 23.3 (Feb. 2015), pp. 3260–3269. DOI: `10.1364/OE.23.003260`
- Lobaz, P. and Kovář, L. "Binarizace počítačem generovaného hologramu pomocí ditheringu". In: *Jemná mechanika a optika* 56.10, 11-12 (2011), pp. 290, 303–305. ISSN: 0447-6441

### Conference tutorials (in order of year)

- *Computer generated display holography.* [In English] A tutorial accepted to Eurographics 2017, April 24–28, Lyon, France.
- *Computer generated holography for computer graphics.* [In English] A tutorial presented at the 3DTV Conference 2011, May 15, Antalya, Turkey. DOI: `10.1109/3DTV.2011.5877153`.

### Details on journals and conferences

Digital holography / computer generated display holography is an interdisciplinary subject that requires both physics (physical optics) and computer science (computer graphics and digital signal processing). Despite that, most results are usually published in journals or at conferences categorized as “Optics” by the Web of Science.

**Optics Express** is an open-access, online journal published by The Optical Society (OSA), ISSN 1094-4087

- Impact Factor: 3.488 (ranked 10th out of 86 journals)
- 5 Year Impact Factor: 3.499
- EigenFactor Score: 0.24990 (ranked 1st out of 86 journals)
- Total Citations: 81,379 (ranked 2nd out of 86 journals) (According to Optics category rankings in the 2015 JCR, Thomson Reuters, 2015)
- One of the most important journals publishing results in digital holography / computer generated display holography (together with Applied Optics and Optics Letters)

**Jemná mechanika a optika** is a journal published by Fyzikální ústav AV ČR (Institute of Physics of the Czech Academy of Sciences), ISSN: 0447-6441, peer reviews since 2002

**International Symposium on Display Holography** is one out of five conferences specialized in holography, since 1982, indexed in the Scopus database. (The other conferences are Digital holography and 3-D imaging, organized by OSA – The Optical Society since 2007, Practical holography, organized by SPIE since 1986, The Holography Conference/Holo-pack.Holoprint, organized by Reconnaissance International since 1990, HoloExpo, organized by a group of Russian institutions since 2004)

**3DTV Conference** is a conference whose objective is to bring together researchers and developers to discuss the development of next generation 3-D immersive and interactive technologies, applications and services. It originated from the project FP6 3DTV Network of Excellence in 2007.

**Eurographics** is a major computer graphics conference organized by the Eurographics Association. It covers the wide field of computer graphics.

## 9.2 Selected talks

### Selected technical talks (in order of year)

- *Fourier transform and associated transforms in optics.* [In Czech] Institute of Mathematics of the Academy of Sciences of the Czech Republic, 2015.
- *Holography and computer science.* [In Czech] Czech Technical University, Prague, Czech Republic, 2015.
- *Computer generated holography: 3D vision and beyond.* [In Czech] Series of six lectures. University of West Bohemia, Pilsen, Czech Republic, 2013.
- *Computer generated holography: 3D vision and beyond.* [In Czech] Comenius University in Bratislava, Bratislava, Slovakia, 2013.
- *Computer generated holography.* [In English] Czech Technical University in Prague, Prague, Czech Republic, 2012.
- *Computer generated holography for computer graphics.* [In English] University of Maribor, Maribor, Slovenia, 2011.
- *Computer generated holography: 3D vision and beyond.* [In English] University of Minho, Braga, Portugal, 2011.
- *Computer generated holography.* [In Czech] The Palacký University, Olomouc, Czech Republic, 2010.

### Selected popular talks and exhibitions (in order of year)

- *Hologramy [Holograms].* [In Czech] A workshop for grammar school teachers on using holography in physics education. At 13th International conference “Dílny Heuréky – Náchod 2014”, Czech Republic.
- *Holograms.* An exhibition of art holograms (concepts by students of Intermedia art at the University of West Bohemia, holographer P. Lobaz) at Galerie Září, Prague, Czech Republic, 2014.
- *Holografie pro střední školy [Holography for grammar schools].* [In Czech]. A seminar and a hands-on course for grammar school teachers held in Hazuka hotel, Pilsen, Czech Republic, 2014.
- *Holography.* [In Czech] 3D Film Fest Prague, Prague, Czech Republic, 2013.

- *Holography.* An exhibition and hands-on courses at Dny vědy a techniky (Days of Science and Technology), Pilsen, Czech Republic, 2013.
- *Kouzlo? Ne. Holografie [Magic? No. Holography].* Half technical, half artistic exhibition of holography. Muzeum jižního Plzeňska v Blovicích, Blovice, Czech Republic, 2013.

## 9.3 Projects

- NECPA – Vývoj algoritmů počítačové grafiky a pro CAD/CAM systémy, LH12181 (1.3.2012–31.12.2015, ZČU/MŠMT a Shandong University, Jinan, People's Republic of China)
- CPG – Centrum počítačové grafiky, LC06008 a LC06008-prodloužený (1.3.2006–31.12.2011, NPV II MŠMT)
- 3DTV – Integrated Three-Dimensional Television – Capture, Transmission and Display, FP6-PLT-511568 (1.9.2004–31.8.2008, EUROPEAN COMMISSION)
- INTUITION – Network of Excellence on Virtual Reality and Virtual Environments Applications for Future Workspaces, FP6-IST-507248-2 (1.9.2004 – 31.10.2008, EUROPEAN COMMISSION)
- Sdílené virtuální světy, CES 128/2004 (12.1.2005–12.1.2007, Cesnet, z.s.p.o.)
- Inovace výuky tvorby počítačových her a multimédií, F0823/2010/F1a (1.1.2010–31.12.2010, FRVŠ MŠMT)
- Dovybavení laboratoře pro výuku kognitivních informatických technik, F0070/2011/Aa (1.1.2011–31.12.2011, FRVŠ MŠMT)
- Pokročilé výpočetní a informační systémy, SGS-2013-029 a SGS-2016-013 (1.3.2013–31.12.2018, ZČU)
- Zkvalitnění výuky speciálních grafických technik, ilustrace a fotografie, VS-15-022 (1.4.2015–31.12.2015, ZČU)
- NTIS – Nové technologie pro informační společnost, CZ.1.05/1.1.00/02.0090 (1.12.2010–31.12.2014, MŠMT OP VaVpI Evropská centra excelence)
- PUNTIS – Podpora udržitelnosti centra NTIS – Nové technologie pro informační společnost, LO1506 (1.7.2015–30.6.2020, MŠMT LO)

## 9.4 Other activities related to holography

- Member of OSA – The Optical Society
- Member of SPIE
- Active member of various online groups specialized in holography
  - Holographyforum.org
    (`www.holographyforum.org/forum`)
  - FB Holography
    (`www.facebook.com/groups/Holography`)
  - FB Holography Forum
    (`www.facebook.com/groups/holographyforum/`)
  - FB Голография [Holography]
    (`www.facebook.com/groups/1734692300124949/`)
  - FB OSA Holography and Diffractive Optics Technical Group
    (`www.facebook.com/groups/OSAholography/`)
- Consulting in both academy (e.g. Brno university of technology, Czech Republic; University of Minho, Portugal; Yıldız Teknik Üniversitesi, Turkey; University of Technology, Iraq) and industry (Optaglio, Czech Republic)

# Appendix A

# Double lookup table method for fast light propagation calculations (conference article)

# Double lookup table method for fast light propagation calculations

**Petr Lobaz**
Dept. of Computer Science and Engineering, University of West Bohemia, Czech Republic
E-mail: `lobaz@kiv.zcu.cz`

**Abstract.** A method of rapid and robust calculation of radially symmetric functions is presented. It is based on observation that $f(x, y, z) = f'(\rho, z)$, where $\rho = (x^2 + y^2)^{1/2}$. The method stores values of $f'$ and $\rho$ in look-up tables. It can be used e.g for fast computer generated hologram calculation. It is suitable for CPU, GPU or hardware implementation.

## 1. Introduction

In computer generated holography and digital holography, it is often necessary to evaluate radially symmetric functions of two variables such as convolution kernels for free space light propagation calculation, e.g. Rayleigh-Sommerfeld or Fresnel kernels [1]. There are several approaches how to accelerate their evaluation. First of all, a moderately complicated formula, such as the Rayleigh-Sommerfeld convolution kernel

$$K_{\mathrm{RS}}(x, y; z_0) = -\frac{1}{2\pi}\left(\mathrm{j}k - \frac{1}{r}\right)\frac{\exp(\mathrm{j}kr)}{r}\frac{z_0}{r}, \qquad \text{where } r = \sqrt{x^2 + y^2 + z_0^2} \tag{1}$$

can be approximated by a simpler formula, such as the Fresnel approximation. Such kernels are used to calculate light propagation from a plane $z = 0$ to a plane $z = z_0$; $\lambda$ stands for a wavelength, $k = 2\pi/\lambda$ is a wave number, $x$ and $y$ are transverse spatial coordinates and j is the imaginary constant. Among acceleration, a simpler formula can have better numerical properties [2]. On the other hand, the approximation error must be taken into account.

Other acceleration method pre-calculates the function for every necessary $x$, $y$ and $z_0$ and stores the values in a 3-D look-up table (LUT) [3, 4]. Some researchers do not calculate LUT at points where the function is not properly sampled, thus reduce its size [5]. Certain functions, e.g. the Fresnel convolution kernel, are separable, i.e. it holds $K(x, y; z_0) = K_x(x; z_0)K_y(y; z_0)$. In this case, it is sufficient to create one 2-D look-up table for each of the two factors $K_x$, $K_y$ [6, 7, 8]. Indeed, this approach is not applicable for non-separable functions.

Some researchers try to accelerate function evaluation by using recurrence formulas, e.g. [9, 10]. It should be noted that influence of computer arithmetic rounding errors is usually poorly analysed and that recurrence formulas tend to produce a sequential computer code rather than a parallel one. An original approach to evaluation of radially symmetric function $K$ is based on computer graphics algorithm for circle rasterisation [11]. Its biggest drawback is its complicated memory access, thus memory caching cannot be used efficiently. Recently, authors proposed a method that overcomes this difficulty using recurrence formulas [12].

Parts of the method we are going to analyse in following sections were independently described by other authors [12, 13, 14]. We will discuss these references after we describe the basic idea of the proposed method in Sec. 2. Detailed analysis is given in Sec. 3 and 4, results in Sec. 5.

## 2. Principle of the method

For light propagation calculation, $K_{\mathrm{RS}}(x, y; z_0)$ must be evaluated for a specific $z_0$ in a finite area of the $xy$ plane. This area should be sampled using sampling distance $\Delta_{xy}$. It is easy to see that $K_{\mathrm{RS}}(x, y; z_0)$ depends in fact on $r$ and $z_0$, where $z_0$ is constant. Moreover, we can define $\rho(x, y) = (x^2 + y^2)^{1/2}$ and write $r = (\rho^2 + z_0^2)^{1/2}$ and $K_{\mathrm{RS}}(x, y; z_0) \equiv K_{\mathrm{RS}}(\rho; z_0)$.

To calculate $K_{\mathrm{RS}}(m\Delta_{xy}, n\Delta_{xy}; z_0)$, where $m$, $n$ are integer indices, it is necessary to evaluate $\rho = \Delta_{xy}(m^2 + n^2)^{1/2}$ and $K_{\mathrm{RS}}(\rho; z_0)$. It is possible to calculate $\rho$ directly or using a 2-D look-up table; we call this look-up table "rhoLUT". We can also build a look-up table with values $K_{\mathrm{RS}}(q\Delta_w; z_0)$ where $\Delta_w$ is sufficiently small and integer index $q$ spans all possible values of $\rho$. We denote this look-up table "waveLUT" because it actually captures wave structure of light.

The waveLUT improves the calculation speed as the evaluation of $K_{\mathrm{RS}}$ or more complicated functions is usually slow. As any error in calculation of $\rho$ can significantly affect the precision of the result, the pre-calculated rhoLUT improves numerical behaviour of the calculation.

The authors of [14] used a very similar method. Instead of rhoLUT, they calculated a look-up table directly for $r = (x^2 + y^2 + z_0^2)$, so they could not reuse it for other $z_0$ coordinate as in the case of rhoLUT. Moreover, sampling distance of this table was set to $\Delta_{xy}$, and thus keeping this table for every $z_0$ would be memory inefficient. The authors of [13] use the table we call waveLUT here and although they in fact use rhoLUT as well (due to Matlab style of matrix manipulation), they do not discuss it explicitly, nor do they investigate its influence. The authors of [12] use waveLUT as well, but instead of rhoLUT, they use recurrence formulas to find values of $\rho$. They also do not discuss the influence of interpolation on calculation precision.

## 3. The rhoLUT analysis

The aim of the rhoLUT is to replace the calculation of $\rho(x, y) = (x^2 + y^2)^{1/2}$, $x \in [0, M\Delta_{xy}]$, $y \in [0, M\Delta_{xy}]$, by a look-up operation, optionally followed by an interpolation. Let us define it as a 2-D array rhoLUT$[m, n] = \Delta_\rho\sqrt{m^2 + n^2}$ for integer indices $0 \leq m, n < \lceil M\Delta_{xy}/\Delta_\rho \rceil$.

No interpolation is necessary for $\Delta_\rho = \Delta_{xy}$. For $\Delta_\rho \neq \Delta_{xy}$, either piecewise constant or piecewise bilinear interpolation can be used. It can be shown that the approximation error caused by the piecewise constant interpolation is not generally acceptable, as the approximate value of $\rho$ is used for the $K_{\mathrm{RS}}$ calculation, where any error is greatly amplified. Piecewise bilinear interpolation gives much better results. If we denote $\rho_B(x, y)$ as a result of the rhoLUT look-up followed by the piecewise bilinear interpolation, it can be found that the approximation error $\rho_B(x, y) - \rho(x, y)$ vanishes for $x \to \infty$, $y \to \infty$, and its maximum is $\Delta_\rho(2 - \sqrt{2})/4 \approx 0.146\Delta_\rho$ located at $(x, y) = (\Delta_\rho/2, \Delta_\rho/2)$.

To select a small enough sampling distance $\Delta_\rho$, it is not sufficient to examine error $\rho_B(x, y) - \rho(x, y)$ alone, as we need to take into account that approximate value $\rho_B$ is used in subsequent calculation of $K_{\mathrm{RS}}$. The most sensitive part of $K_{\mathrm{RS}}$ is $\exp(\mathrm{j}2\pi r/\lambda)$, because even a slight error in $r$ is greatly amplified. It is thus necessary to analyse error $r(\rho_B, z_0) - r(\rho, z_0)$. Analysis reveals that this error is the biggest for $\rho$ close to 0. As we know that $\rho_B(x, y) - \rho(x, y)$ is the biggest at $(x, y) = (\Delta_\rho/2, \Delta_\rho/2)$, we can guess that $r(\rho_B, z_0) - r(\rho, z_0)$ at this point estimates maximum error of $r$. We can thus define estimate of the maximum error of $r$ as

$$\mathrm{maxRErr}(\Delta_\rho, z_0) = 1.2\left\{r\left[\rho_B\left(\frac{\Delta_\rho}{2}, \frac{\Delta_\rho}{2}\right), z_0\right] - r\left[\rho\left(\frac{\Delta_\rho}{2}, \frac{\Delta_\rho}{2}\right), z_0\right]\right\}, \tag{2}$$

where $r(\rho, z_0) = (\rho^2 + z_0^2)^{1/2}$ and factor 1.2 reflects the observation that the error of $r$ for $(x, y) = (\Delta_\rho/2, \Delta_\rho/2)$ is approximately 85% of the maximum.

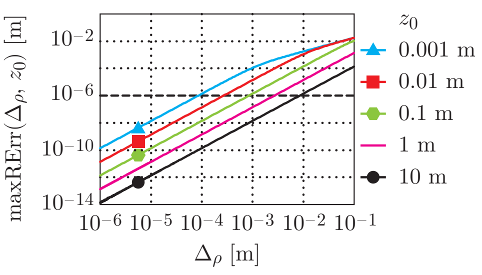


**Figure 1.** Approximate value of maximum error in calculation of $r$.

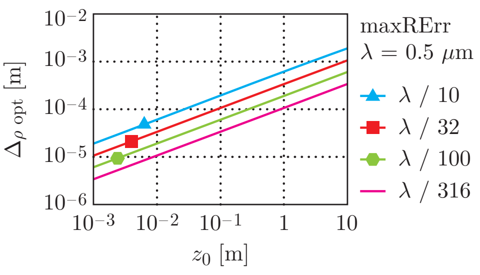


**Figure 2.** Optimal sampling distance $\Delta_{\rho\ \mathrm{opt}}$ calculated using Eq. (3).

It can be seen in Fig. 1 that in the log-log graph, maxRErr can be approximated by a line for a wide range of useful errors and $\Delta_\rho$. Recall that for visible light, error of $r$ must be well below 1 $\mu$m that is depicted by a thick dashed horizontal line.

A set of lines in a log-log graph is defined as $\log(\text{maxRErr}) = \kappa \log(\Delta_\rho) + \xi_0 + \xi_1 \log(z_0)$, where $\kappa$ is the slope and $\xi_0$ and $\xi_1$ define the intercept. We can measure in the graph that for a wide range of common $z_0$ and acceptable maxRErr, $\kappa \approx 2$, $\xi_0 \approx -2$ and $\xi_1 \approx -1$.

The linear approximation allows us to find optimal $\Delta_{\rho\ \mathrm{opt}}$ for a chosen maxRErr:

$$\Delta_{\rho\ \mathrm{opt}}(\text{maxRErr}, z_0) = \exp\left(\frac{1}{\kappa}\log\left[\frac{\text{maxRErr}}{(z_0)^{\xi_1}\exp(\xi_0)}\right]\right) \approx 2.72\sqrt{z_0\,\text{maxRErr}}. \tag{3}$$

Example values of $\Delta_{\rho\ \mathrm{opt}}$ for $\lambda = 500$ nm can be found in Fig. 2. Note they are much larger than $\lambda$, thus rhoLUT size is usually small.

## 4. The waveLUT analysis

Once we have the approximate $\rho$ value, we can calculate $K_{\mathrm{RS}}(\rho; z_0)$ using the waveLUT. We should set small enough $\Delta_w$ and define waveLUT$[q; z_0] = K_{\mathrm{RS}}(q\Delta_w; z_0)$, where $q$ is an integer index. Again, for a particular value $\rho_0$ we can estimate the value of $K_{\mathrm{RS}}(\rho_0; z_0)$ using piecewise constant approximation or piecewise linear approximation.

The extent of the waveLUT in the $\rho$ direction is given by the rhoLUT. For a given maximum value $\rho_{\max}$, we can calculate the local frequency $\mathrm{lf}_{\mathrm{RS}}(\rho_{\max}; z_0)$ and set $\Delta_w < 1/[2\,\mathrm{lf}_{\mathrm{RS}}(\rho_{\max}; z_0)]$ to have at least two samples per cycle of $K_{\mathrm{RS}}(\rho; z_0)$. Our experiments show that good results are obtained with 8 samples per cycle and a piecewise linear interpolation.

## 5. Results

As shown in [2], direct calculation of highly oscillatory functions such as $K_{\mathrm{RS}}$ is prone to numerical error in single precision calculations. No problems appeared in single precision environment when using a rhoLUT and a waveLUT pre-calculated in double precision. See Fig. 3 for an example of both correct and incorrect $K_{\mathrm{RS}}$ calculation. Tests also show that selection of $\Delta_\rho$ and $\Delta_w$ according to Sections 3 and 4 works as expected.

We have also measured actual error of calculation of $K_{\mathrm{RS}}$. Choosing 8 samples per fringe in selection of $\Delta_w$ and linear interpolation in the waveLUT leads to maximum error 7.0% (1.1% on average). While this number may seem high, it should be noted that the maximum error appears in the finest fringes and has a negligible impact on the optical field properties. Combination of the rhoLUT and the waveLUT further increases the error; typical value of maxRhoErr $= \lambda/100$ leads to the maximum error 8.8% (2.8% on average), which is still acceptable for our purposes.

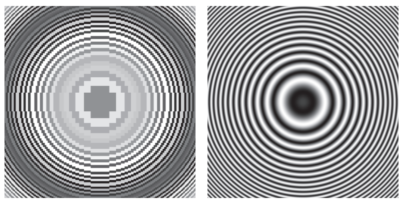


**Figure 3.** Real part of $K_{\mathrm{RS}}$ calculated in IEEE 754 single precision arithmetic and in double precision arithmetic ("correct calculation"). Calculation using look-up tables leads to the same correct result.

Calculation time was tested in realistic geometric scenarios. We have prepared two test cases – one for complicated filtered propagation kernels (see [15]), and one for the Rayleigh-Sommerfeld kernel. Complicated filtered propagation kernel calculation is accelerated mainly by utilizing waveLUT; 10× or 100× faster calculation is easily achieved, depending on the complexity of the kernel. Using waveLUT in simple kernel calculation leads to about 1.7× faster calculation. These numbers include the waveLUT calculation, which takes about 1% of the overall time.

Introducing rhoLUT enhances numerical behaviour in single precision environment; in double precision environment, this advantage is not important, as $\rho$ can be easily evaluated directly. Unoptimized implementation of the rhoLUT can actually double calculation time compared to direct $\rho$ calculation and the waveLUT. On the other hand, careful rhoLUT implementation that uses integer arithmetic whenever possible leads to further 20% to 40% speed-up compared to direct $\rho$ calculation and the waveLUT.

## 6. Conclusion

We have introduced a method of calculation of arbitrary radially symmetric functions using a pair of look-up tables, a rhoLUT and a waveLUT. While using a waveLUT is always advantageous, using a rhoLUT has its pros and cons. The rhoLUT enhances numerical behaviour in a limited precision environment, but it must be carefully implemented to improve the speed of calculation.

**Acknowledgments**

This work was supported by the European Regional Development Fund (ERDF), project "NTIS – New Technologies for the Information Society", European Centre of Excellence, CZ.1.05/1.1.00/02.0090, and by Ministry of Education, Youth, and Sport of Czech Republic – University spec. research – 1311. I would like to thank P. Vaněček for collaboration on implementation and testing and L. Váša for valuable comments.

## References

[1] Goodman J W 2004 *Introduction to Fourier Optics* 3rd ed (Roberts & Company Publishers) ISBN 0974707724
[2] Lobaz P and Vaněček P 2015 *Opt. Express* **23** 3260–3269
[3] Kim S C and Kim E S 2008 *Appl. Opt.* **47** D55–D62
[4] Cho S, Ju B K, Kim N Y and Park M C 2014 *Optical Engineering* **53** 054108
[5] Shimobaba T, Nakayama H, Masuda N and Ito T 2010 *Opt. Express* **18** 19504–19509
[6] Pan Y, Xu X, Solanki S, Liang X, Tanjung R B A, Tan C and Chong T C 2009 *Opt. Express* **17** 18543–18555
[7] Esmer G B 2013 *Appl. Opt.* **52** A18–A25
[8] Kim S C, Kim J M and Kim E S 2012 *Opt. Express* **20** 12021–12034
[9] Yoshikawa H, Iwase S and Oneda T 2000 *Proc. SPIE* **3956** 48–55
[10] Matsushima K and Takai M 2000 *Appl. Opt.* **39** 6587–6594
[11] Nishitsuji T, Shimobaba T, Kakue T, Masuda N and Ito T 2012 *Opt. Express* **20** 27496–27502
[12] Nishitsuji T, Shimobaba T, Kakue T and Ito T 2015 *Opt. Express* **23** 9852–9857
[13] Huang Y, Zhao K, Yan X, Pei C and Jiang X 2014 *Journal of Electronic Imaging* **23** 043013
[14] Lee S, Wey H C, Nam D K, Park D S and Kim C Y 2012 *Proc. SPIE* **8498** 84980O
[15] Lobaz P 2013 *Opt. Express* **21** 2795–2806

# Appendix B

# Derivation of the DLUT method details

# Double lookup table method for fast light propagation calculations

*Supplementary material to the article presented at 10th International Symposium on Display Holography,* *http://isdh2015.ifmo.ru*

*Petr Lobaz, Univerisity of West Bohemia, Czech Republic, 2015.*

This text analyses errors in propagation calculation using the DoubleLUT acceleration method. Here, LUT stands for a look-up table, “double” stands for using `waveLUT` and optionally `rhoLUT`, see below. The analysis is made for the Rayleigh-Sommerfeld convolution propagation method only. The analysis is thus also valid for filtered Rayleigh-Sommerfeld kernels. Moreover, it can be quite easily adjusted for other methods.

## 1 Introduction

In a propagation calculation, it is necessary to evaluate a propagation convolution kernel, for example the Rayleigh-Sommerfeld kernel:

$$K(x, y, z_0) = K(r) = -\frac{1}{2\pi}\left(\mathbf{j}\,k - \frac{1}{r}\right)\frac{\exp(\mathbf{j}\,kr)}{r}\frac{z_0}{r},$$

where $x$, $y$ are the transverse coordinates, $z_0$ is a propagation distance, $\mathbf{j}$ is the imaginary constant, $k$ is the wave number equal to $2\pi/\lambda$, where $\lambda$ is the wavelength, and $r$ is a distance between particular points in the source plane $z = 0$ and the target plane $z = z_0$.

Here, the value of $r$ is defined as

$$r(x, y, z_0) = \sqrt{x^2 + y^2 + z_0^2}$$

and can be rewritten as

$$\begin{aligned} r(\rho, z_0) &= \sqrt{\rho^2 + z_0^2} \\ \rho(x, y) &= \sqrt{x^2 + y^2}. \end{aligned}$$

It is then indeed possible to evaluate

$$K(x, y, z_0) = \texttt{waveLUT}\left(\texttt{rhoLUT}(x, y), z_0\right)$$

where

$$\texttt{rhoLUT}(x, y) = \sqrt{x^2 + y^2}$$

evaluates the value $\rho$ and

$$\texttt{waveLUT}(\rho, z_0) = K\left(\sqrt{\rho^2 + z_0^2}\right)$$

evaluates the final value. It also follows that $K(x, y, z_0)$ is radially symmetric in the plane $xy$ for constant $z_0$.

The heart of the DoubleLUT method is the observation that evaluation of `rhoLUT`$(x, y)$ is quite easy, while evaluation of `waveLUT`$(\rho, z_0)$ is much harder due to complexity of $K(x, y, z_0)$.

It is therefore possible to prepare a one dimensional look-up table `waveLUT` (single value $z_0$ is assumed) with pre-calculated values for different $\rho$ values. The value of $\rho$ can be evaluated directly, or using a two-dimensional look-up table. As two-dimensional tables occupy a lot of memory, it is desirable to sub-sample `rhoLUT` information and to interpolate desired value of $\rho$ from the closest pre-calculated values.

This analysis compares direct evaluation of $\rho(x, y)$ and interpolated calculation of $\rho$ using sub-sampled `rhoLUT` and bilinear interpolation. It also compares evaluation of $r(\rho, z_0)$ for $\rho$ calculated in either way.

Here, general properties of errors are analysed. Detailed analysis of errors for particular propagation parameters is implemented in the Octave/Matlab script `analyzeBiError_RS.m`, the other supplementary material to the article.

The script `analyzeBiError_RS.m` shows dependence of errors on $x$ and $y$. It can be easily seen there that the maximum error is found in the vicinity of $(0, 0)$, or more precisely in the area $0 \leq x \leq \Delta$, $0 \leq y \leq \Delta$, where $\Delta$ is the sampling distance of the `rhoLUT`. (Please note that this parameter is depicted as $\Delta_\rho$ in the article and in the script `analyzeBiError_RS.m`). Therefore we will be mostly interested in errors in this area. However, we are going to analyse limits for $x \to \infty$ and $y \to \infty$ as well.

Following sections are actually composed of input and output fields of the symbolic algebra software Maxima (`http://maxima.sourceforge.net/`). The reader is encouraged to use the Maxima software and change parameters to see different graphs, etc.

The structure of the text is as follows. In Sec. 2, only some assumptions and definitions are given. Sec. 3 analyses error in $\rho$ induced by bilinear interpolation in the `rhoLUT`. Sec. 4 analyses how the error of $\rho$ affects the error of $r = (\rho^2 + z_0^2)^{1/2}$. Its main result is an estimate of sampling distance $\Delta$ of the `rhoLUT`. Sec. 5 adds a few remarks to the derivations in Sec. 4. However, the main result is not altered.

## 2 Maxima assumptions and some definitions

We will need a particular value of a wavelength sometimes. Set some reference value. Please note that all dimensions here are in meters.

(%i1) `lambda : 500*10^(-9);`

(%o1) $\frac{1}{2000000}$

Precise $\rho(x, y)$ and $r(\rho, z)$ calculation:

(%i2) `rho(x,y) := sqrt(x^2+y^2);`

(%o2) $\rho\left(x, y\right) := \sqrt{x^2 + y^2}$

(%i3) `r(rho,z) := sqrt(rho^2+z^2);`

(%o3) $\mathrm{r}\left(\rho, z\right) := \sqrt{\rho^2 + z^2}$

In further derivations, it is wise to state some assumptions. Propagation distance should be positive:

(%i4) `assume(z>0);`

(%o4) $[z > 0]$

Bilinear interpolation in `rhoLUT` is done inside “a cell” with “lower left” corner $(x_0, y_0)$ and “upper right” corner $(x_0+\Delta, y_0+\Delta)$. Normalized coordinates `ix`, `iy` in the cell are assumed to be $0 <$ `ix` $< 1$, $0 <$ `iy` $< 1$. Sometimes, we will call them $a$, $b$.

(%i5) `assume(a>0);`

(%o5) $[a > 0]$

(%i6) `assume(b>0);`

(%o6) $[b > 0]$

(%i7) `assume(Delta>0);`

(%o7) $[\Delta > 0]$

Some graphs will be larger than Maxima default. Set the large graph size.

(%i8) `extsize:[800,300];`

(%o8) $[800, 300]$

## 3 Error of rho using bilinear interpolation

When calculating approximate value of $\rho$, we will use bilinear interpolation between precise values $\rho(x_0, y_0)$, $\rho(x_0+\Delta, y_0)$, $\rho(x_0, y_0+\Delta)$ and $\rho(x_0+\Delta, y_0+\Delta)$. We call this area "the cell $[x_0, y_0]$" (note the square brackets).

First, define normalized coordinates inside the cell.

```
(%i9)   ix(x, x0, Delta) := (x - x0) / Delta$
```

```
(%i10)  iy(y, y0, Delta) := (y-y0) / Delta$
```

Define bilinear interpolation at $(x, y)$ inside a cell $[x_0, y_0]$:

```
(%i11)  rhoB(x,y,x0,y0,Delta) :=
          rho(x0,y0)*(1-ix(x,x0,Delta))*(1-iy(y,y0,Delta))
          + rho(x0,y0+Delta)*(1-ix(x,x0,Delta))*iy(y,y0,Delta)
          + rho(x0+Delta,y0)*ix(x,x0,Delta)*(1-iy(y,y0,Delta))
          + rho(x0+Delta,y0+Delta)*ix(x,x0,Delta)*iy(y,y0,Delta)$
```

Define error as a difference between the approximate and the precise values.

```
(%i12)  rhoErrorB(x,y,x0,y0,Delta) := rhoB(x,y,x0,y0,Delta) - rho(x,y)$
```

Let us show that `rhoErrorB` is zero at infinity:

```
(%i13)  limit(rhoErrorB(x0+Delta/2,a*x0+Delta/2,x0,a*x0,Delta), x0, inf);
```

(%o13) $0$

```
(%i14)  limit(rhoErrorB(0+Delta/2, y0+Delta/2, 0, y0, Delta), y0, inf);
```

(%o14) $0$

```
(%i15)  limit(rhoErrorB(x0+Delta/2, x0+Delta/2, x0, x0, Delta), x0, inf);
```

(%o15) $0$

For $x = 0$, it holds $\rho(x, y) = |y|$. It is therefore clear that `rhoLUT` should be aligned with axes $x$, $y$, i.e. it should include $\rho(0, 0)$. Otherwise, the error near the origin will be big:

```
(%i16)  radcan(rhoErrorB(0,0,-Delta/2,-Delta/2,Delta));
```

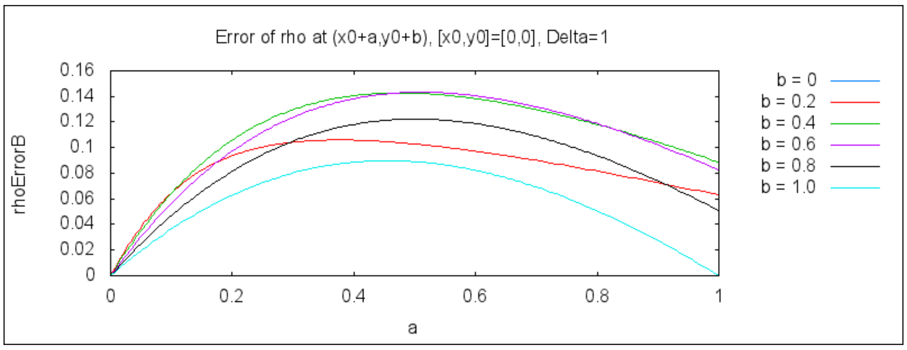
(%o16) $\frac{\Delta}{\sqrt{2}}$

```
(%i17)  float(%);
```

(%o17) $0.70710678118655\,\Delta$

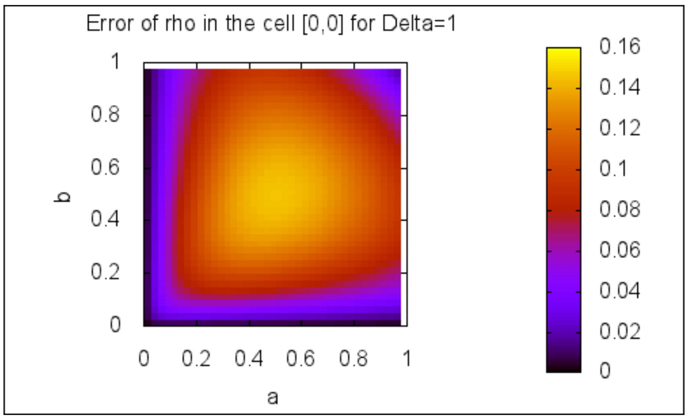
We will show that `rhoLUT` containing the cell $[0, 0]$ leads to much smaller $\rho$ error; we are going to show that the maximum error is found at $(\Delta/2, \Delta/2)$. First, let us display the `rhoErrorB` inside the cell $[0, 0]$:

```
(%i18) wxplot2d(makelist(rhoErrorB(0+a,0+b,0,0,1), b, bList),
         [a,0,1], [ylabel, "rhoErrorB"],
         [legend, makelist(simplode(["b = ", b]), b, bList)],
         [gnuplot_preamble,
         "set title 'Error of rho at (x0+a,y0+b), [x0,y0]=[0,0], Delta=1';
          set key outside"]),
       bList=makelist(b*0.2, b, 0, 5), wxplot_size=extsize;
```



(%t18)

```
(%i19) wxplot3d(rhoErrorB(0+a,0+b,0,0,1), [a,0,1], [b,0,1],
       [mesh_lines_color, false],
       [grid, 40, 40],
       [legend, ""],
       [gnuplot_preamble, simplode(["set view map; set size ratio -1;"
         "set size 1.2;",
         "set origin -0.2,-0.06;",
         "set colorbox user origin 0.8, 0.1 size 0.05, 0.8;",
         "set title 'Error of rho in the cell [0,0] for Delta=1'"])]);
```



(%t19)

We can easily show that the maximum can be found at $(\Delta/2, \Delta/2)$:

```
(%i20) rhoErrorBdiffX(x, y, Delta) :=
         ''(diff(rhoErrorB(x, y, 0, 0, Delta), x, 1))$

(%i21) rhoErrorBdiffY(x, y, Delta) :=
         ''(diff(rhoErrorB(x, y, 0, 0, Delta), y, 1))$
```

```
(%i22) rhoErrorBdiffX(Delta/2, Delta/2, Delta);
```

(%o22) $0$

```
(%i23) rhoErrorBdiffY(Delta/2, Delta/2, Delta);
```

(%o23) $0$

Now we can see that the error at $(\Delta/2, \Delta/2)$ is much smaller than in situation when `rhoLUT` did not contain the cell $(0, 0)$:

```
(%i24) radcan(rhoErrorB(0+Delta/2,0+Delta/2,0,0,Delta));
```

(%o24) $-\frac{\left(\sqrt{2}-2\right)\Delta}{4}$

```
(%i25) float(%);
```

(%o25) $0.14644660940673\,\Delta$

While error of $\rho$ is important, in future we are going to explore the error of $f(\rho')$, where $f$ is an arbitrary function and $\rho'$ is an approximate value of $\rho$. We will assume that the function $f$ is "simple".

Let us make a reasonable assumption: we know that there is no error in $\rho$ at the corners of the interpolation cell, thus $f(\rho')$ is exact there as well. As the function `rhoErrorB` is quite "simple", we can guess that the error of $f(\rho')$ in the center of the cell is a representative of the maximum error of $f(\rho')$ in the whole cell.

We don't claim the maximum error of $f(\rho')$ is in the cell center, it definitely depends on the function $f$; we claim that the error in the center is an estimate of the maximum error of the cell. Therefore we define an estimate of the error at $(x, y)$ as the error of the center of the cell $[x-\Delta/2, y-\Delta/2]$. Such an estimation neglects the alignment of the `rhoLUT` with the origin. On the other hand, such function is smooth and much easier to analyse.

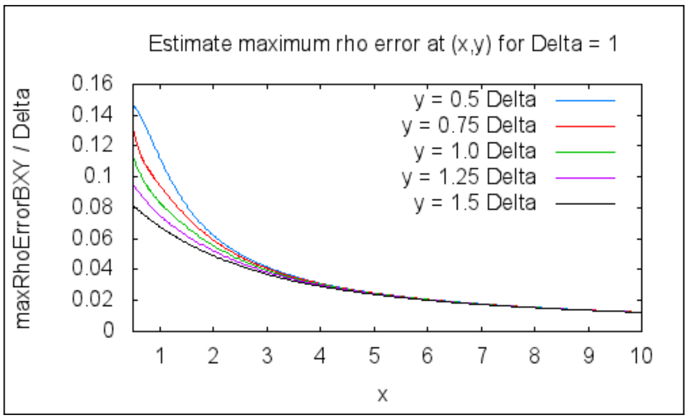
```
(%i26) maxRhoErrorBXY(x, y, Delta) :=
         rhoErrorB(x, y, x-0.5*Delta, y-0.5*Delta, Delta);
```

(%o26) $\texttt{maxRhoErrorBXY}\,(x, y, \Delta) := \texttt{rhoErrorB}\,(x, y, x-0.5\Delta, y-0.5\Delta, \Delta)$

As an example, let us draw the estimate of maximum error for various $y$ close to 0. Note that both $x$ and $y$ should not be smaller than $\Delta/2$, as the corner $(x_0, y_0)$ of the cell must be larger than 0.

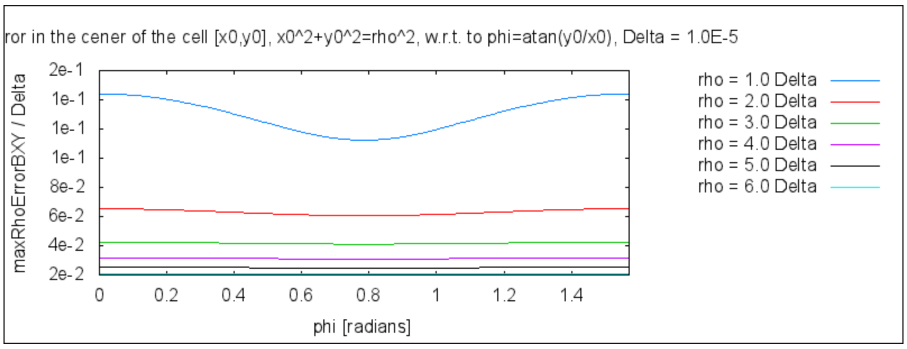
```
(%i27) wxplot2d(makelist(maxRhoErrorBXY(x, y*Delta, Delta)/Delta, y, ylist),
        [x, 0.5*Delta, 10*Delta],
        [legend, makelist(simplode(["y = ", y, " Delta"]), y, ylist)],
        [ylabel, "maxRhoErrorBXY / Delta"],
        [gnuplot_preamble,
          simplode(["set title 'Estimate maximum rho error at (x,y)"
          " for Delta = ",  Delta, "';"])]),
       ylist=[0.5, 0.75, 1.0, 1.25, 1.5], Delta = 1;
```

(%t27)



Let us simplify the analysis even more: let us explore if the error that we named `maxRhoErrorBXY` is radially symmetric. If this was true, it would allow us to reduce the dimensionality of the problem from two dimensions $(x, y)$ to one dimension ($\rho = (x^2 + y^2)^{1/2}$). The graph below draws `maxRhoErrorBXY` for various $x^2 + y^2 = const.$, where horizontal axis is equal to $\text{atan}(y/x)$. It is clear that the function is not perfectly radially symmetric – otherwise the lines would be perfectly horizontal.

```
(%i28) wxplot2d(makelist(maxRhoErrorBXY(cos(phi)*rho, sin(phi)*rho, Delta)
                                                 / Delta, rho, rhoList),
        [phi, 0, %pi/2], [xlabel, "phi [radians]"],
        [ylabel, "maxRhoErrorBXY / Delta"],
        [legend,
          makelist(simplode(["rho = ", rho/Delta, " Delta"]), rho, rhoList)],
        [gnuplot_preamble,
          simplode(["set title 'Error in the cener of the cell [x0,y0],",
          " x0^2+y0^2=rho^2, w.r.t. to phi=atan(y0/x0), Delta = ", Delta, "';",
          "set format y '%.0te%+1T'; set key outside"])]),
       rhoList=makelist(rho*1e-5, rho, 1, 6), Delta=1e-5, fpprintprec=2,
         wxplot_size=extsize;
```

(%t28)


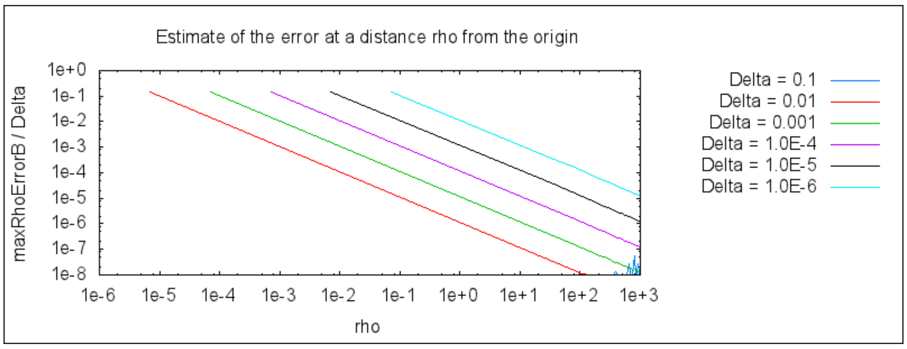
However, it is clear from the graph that the error is the biggest for $\phi = 0$ or $\phi = \pi/2$, i.e. for the cells $[x_0, 0]$ or $[0, y_0]$. If we are looking for an error estimate at a distance $\rho$ from the origin, we know it will not be larger than the maximum error in the cell $[\rho - \epsilon, 0]$, where $\epsilon$ is such that the center of this cell is located at a distance $\rho$ from the origin, i.e. at a point $((\rho^2 - \{\Delta/2\}^2)^{1/2}, \Delta/2)$. Let us define such an estimate:

```
(%i29) maxRhoErrorB(rho, Delta) :=
         maxRhoErrorBXY(sqrt(rho^2-(0.5*Delta)^2), 0.5*Delta, Delta);
```

(%o29) $$\mathtt{maxRhoErrorB}\left(\rho, \Delta\right) := \mathtt{maxRhoErrorBXY}\left(\sqrt{\rho^2 - (0.5\,\Delta)^2}, 0.5\,\Delta, \Delta\right)$$

The graph of `maxRhoErrorB` clearly shows that the error of $\rho$ drops as $\rho \to \infty$. It is also clear that the maximum error is at $(x, y) = (\Delta/2, \Delta/2)$, i.e. at the center of the cell $[0, 0]$. Note that all the lines begin at $\rho = \Delta\sqrt{2}/2$, because smaller $\rho$ would lead to cells $[x_0, y_0]$ with negative $x_0, y_0$, and our error estimation functions assume they are positive.

```
(%i30) wxplot2d(makelist([parametric, trho,
         maxRhoErrorB(trho, Delta)/Delta, [trho, Delta*sqrt(2)/2, 1000]],
        Delta, Deltalist),
       [ylabel, "maxRhoErrorB / Delta"], [xlabel, "rho"],
       [x, 1e-6, 1000], [y, 1e-8, 1], [logx], [logy],
       [legend,
         reverse(makelist(simplode(["Delta = ", Delta]), Delta, Deltalist))],
       [gnuplot_preamble,
         simplode(["set key outside; set format xy '%.0te%+1T';",
         "set title 'Estimate of the error at a distance rho"
         " from the origin'"])]),
       Deltalist=[1e-6, 1e-5, 1e-4, 1e-3, 1e-2, 1e-1], wxplot_size=[800,300],
       fpprintprec=2;
```

(%t30) 

In conclusion, bilinear interpolation in `rhoLUT` works well. If we need to limit error by $e$, it is possible to set $\Delta = e/0.146 = 7e$, see Eq. `(%o25)`. However, error is equal to $e$ just in the vicinity of the origin and vanishes quickly for large $x$, $y$.

## 4 Effect of rho error on error of r

Now we are going to analyse effect how the error in $\rho$ influences calculation of $r = (z^2 + \rho^2)^{1/2}$. Let us assume first that the error of $\rho$ (denoted as $e$) is constant. It is true for constant interpolation in the `rhoLUT`, but too restrictive for bilinear interpolation in the `rhoLUT`, as then the error $e$ vanishes as $\rho \to \infty$.

```
(%i31) rError(rho, z, e) := ''(r(rho+e, z) - r(rho, z));
```

(%o31) $$\texttt{rError}\,(\rho, z, e) := \sqrt{z^2 + (\rho + e)^2} - \sqrt{z^2 + \rho^2}$$

Even with constant $e$, the error vanishes for large $x$, $y$:

```
(%i32) limit(rError(0, z, e), z, inf);
```

(%o32) $0$

```
(%i33) limit(rError(rho,z,e), z, inf);
```

(%o33) $0$

However, these situations assumed “on axis” case. In the “off-axis” case, i.e. for $\rho = \alpha z$, $z \to \infty$ for some constant $\alpha$, the error does not vanish. But even now the overall error is not larger than $e$:

```
(%i34) limit(rError(alpha*z, z, e), z, inf);
```

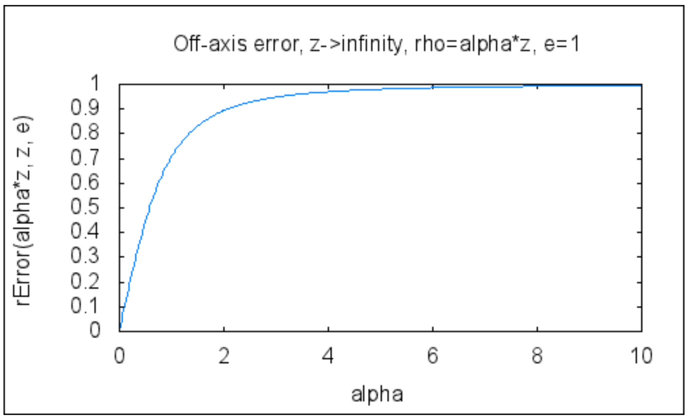
(%o34) $$\frac{\alpha\, e}{\sqrt{\alpha^2 + 1}}$$

```
(%i35) limit(%, alpha, inf);
```

(%o35) $e$

```
(%i36) wxplot2d([limit(rError(alpha*z, z, e), z, inf)],
         [alpha, 0, 10], [ylabel, "rError(alpha*z, z, e)"],
         [gnuplot_preamble,
         "set title 'Off-axis error, z->infinity, rho=alpha*z, e=1';"]),
       e=1;
```

(%t36)



It would be obviously better if the error vanished. This can be accomplished if $\rho$ is calculated using bilinear interpolation. Then the error $e$ is no longer constant, but can be estimated by `rErrorEstB` function.

```
(%i37) rErrorEstB(rho, z, Delta) := rError(rho, z, maxRhoErrorB(rho, Delta));
```

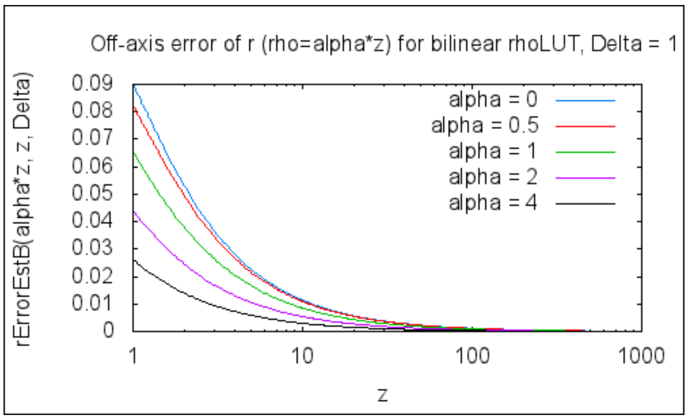
(%o37) $\mathtt{rErrorEstB}\left(\rho, z, \Delta\right) := \mathtt{rError}\left(\rho, z, \mathtt{maxRhoErrorB}\left(\rho, \Delta\right)\right)$

Now we can see that the error vanishes in both on-axis and off-axis cases.

```
(%i38) limit(rErrorEstB(rho, z, Delta), z, inf), ratprint=false;
```

(%o38) $0$

```
(%i39) wxplot2d(makelist(rErrorEstB(alpha*z + Delta*sqrt(2)/2, z, Delta),
           alpha, alphaList),
        [z, Delta, 1e3],
        [ylabel, "rErrorEstB(alpha*z, z, Delta)"],
        [legend, makelist(simplode(["alpha = ", alpha]), alpha, alphaList)],
        [logx],
        [gnuplot_preamble,
          simplode(["set title 'Off-axis error of r (rho=alpha*z)",
            " for bilinear rhoLUT, Delta = ", Delta, "';"])]),
       Delta=1, alphaList=[0, 0.5, 1, 2, 4];
```

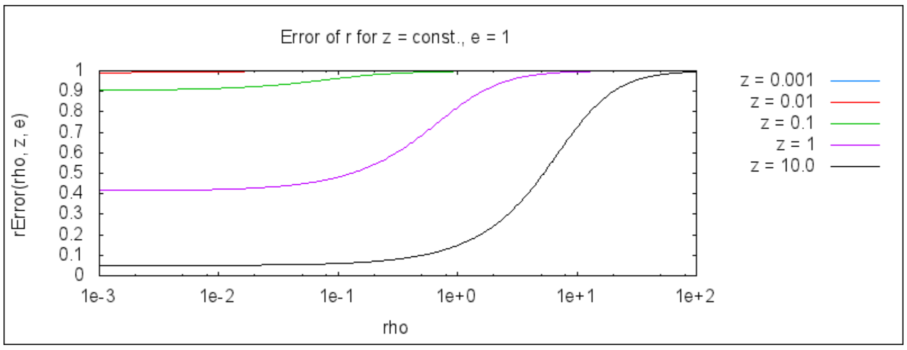
(%t39) 

We are mostly interested in a different error analysis. In most cases, $z =$ *const.* and we are interested how does the error change with $\rho$. First, let us again assume that the error of $\rho$ is equal to constant $e$, which is the case for constant `rhoLUT` interpolation. We can see that the error approaches to $e$. For small $z$, it is equal to $e$ instantly, for large $z$ it approaches slowly.

```
(%i40) limit(rError(rho, z, e), rho, inf);
```

(%o40) $e$

```
(%i41) wxplot2d(makelist(rError(rho, z, e), z, zlist),
         [rho, 1e-3, 100],
         [legend, makelist(simplode(["z = ", z]), z, zlist)],
         [ylabel, "rError(rho, z, e)"], [logx],
         [gnuplot_preamble,
           simplode(["set title 'Error of r for z = const., e = ", e, "';",
           "set key outside; set format x '%.0te%+1T'"])]),
       zlist=[1e-3, 1e-2, 1e-1, 1, 1e1], e=1, wxplot_size=extsize;
```

(%t41) 

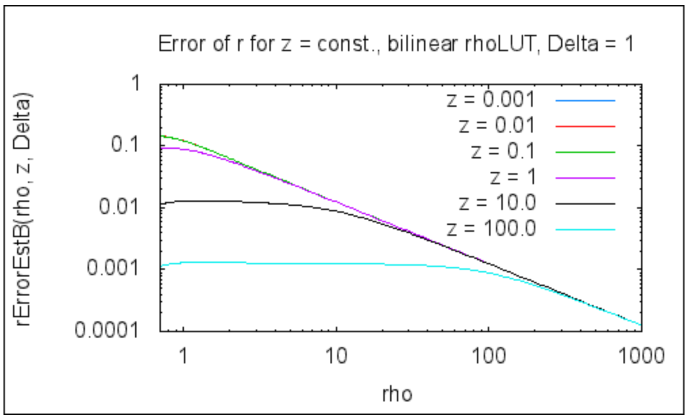
But again we should invesigate influence of bilinear interpolation in the `rhoLUT`, i.e. the error of $\rho$ vanishes for $\rho \to \infty$. Please note that again $\rho$ must be larger than $\Delta\sqrt{2}/2$, as this $\rho$ leads to the estimate of the error in the cell $[0, 0]$. We can see the overall error vanishes, which is obviously good.

```
(%i42) limit(rErrorEstB(rho, z, Delta), rho, inf), ratprint=false;
```

(%o42) 0

```
(%i43) wxplot2d(makelist(rErrorEstB(rho, z, Delta), z, zList),
         [rho, Delta*sqrt(2)/2, 1000],
         [legend, makelist(simplode(["z = ", z]), z, zList)],
         [ylabel, "rErrorEstB(rho, z, Delta)"], [logx], [logy],
         [gnuplot_preamble,
           simplode(["set title 'Error of r for z = const., bilinear rhoLUT,"
           " Delta =", Delta, "';"])]),
       zList=[1e-3, 1e-2, 1e-1, 1, 1e1, 1e2], Delta=1;
```

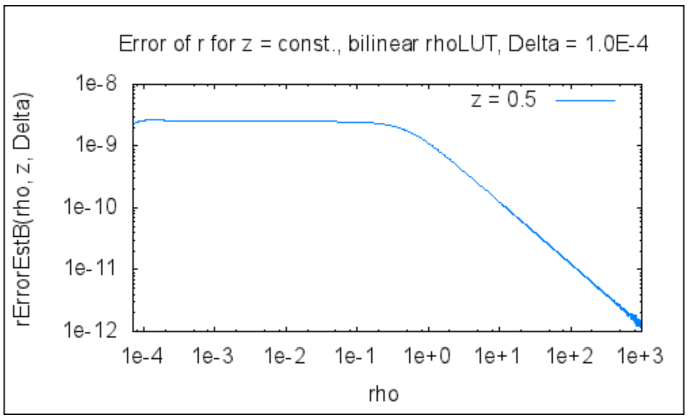
(%t43)



Just to check: Fig. 3 of the article shows that error of $r$ for $\Delta = 100\ \mu$m, $z = 0.5$ m is at most about 2.67 nm. Maximum of the graph below is the same, which indicates our error analysis is correct.

```
(%i44) wxplot2d(makelist(rErrorEstB(rho, z, Delta), z, zList),
         [rho, Delta*sqrt(2)/2, 1000],
         [legend, makelist(simplode(["z = ", z]), z, zList)],
         [ylabel, "rErrorEstB(rho, z, Delta)"], [logx], [logy],
         [gnuplot_preamble,
           simplode(["set format xy '%.0te%+1T';",
           "set title 'Error of r for z = const., bilinear rhoLUT,",
           " Delta = ", Delta, "';"])]),
       zList=[0.5], Delta=100e-6;
```

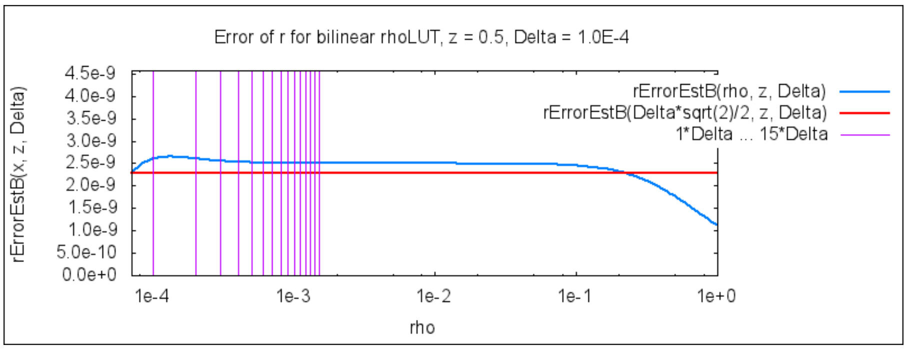
(%t44)



The graphs above show that the error of $r$ for $\rho = \Delta\sqrt{2}/2$ is a rough approximate of the maximum error. However, it is just an approximation, slightly larger $\rho$ leads to a bigger error. Let us draw a single graph for the parameters of Fig. 3 of the article, i.e. $z = 0.5$ m, $\Delta = 100$ $\mu$m. The red line is the error estimate for $\rho = \Delta\sqrt{2}/2$. The blue line is the estimate for variable $\rho$. Purple vertical lines are located at first integer multiples of $\Delta$. We can see that the error is actually larger than the red estimate in many `rhoLUT` cells near the origin.

```
(%i45) wxplot2d(append(
           [rErrorEstB(x, z, Delta), rErrorEstB(Delta*sqrt(2)/2, z, Delta)],
           makelist([parametric, n*Delta, t, [t, 0, 1e-7]], n, 1, DeltaLines)),
         [x, Delta*sqrt(2)/2, 10000*Delta], [xlabel, "rho"], [logx],
         append([legend,
          "rErrorEstB(rho, z, Delta)", "rErrorEstB(Delta*sqrt(2)/2, z, Delta)",
          simplode(["1*Delta ... ", DeltaLines, "*Delta"])],
          makelist("", n, 1, DeltaLines-1)),
         append([style, [lines, 2, 1], [lines, 2, 2]],
           makelist([lines, 1, 4], col, DeltaLines)),
         [ylabel, "rErrorEstB(x, z, Delta)"],
         [y,0,2*rErrorEstB(Delta*sqrt(2)/2, z, Delta)],
         [gnuplot_preamble,
           simplode(["set format x '%.0te%+1T'; set format y '%.1te%+1T'; ",
           "set key outside;",
           "set title 'Error of r for bilinear rhoLUT, z = ",
           z, ", Delta = ", Delta, "'"])]),
       z=0.5, Delta=100e-6, DeltaLines=15, wxplot_size=extsize;
```

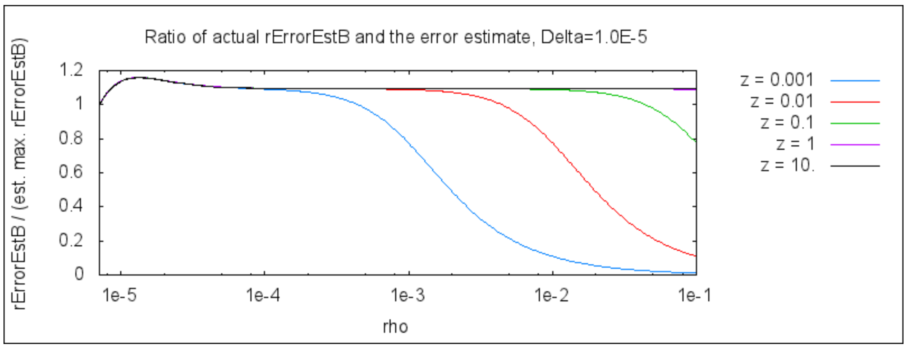
(%t45)



Following two graphs show error behaviour for $\Delta = 10$ $\mu$m and $\Delta = 1$ cm. Graphs for intermediate $\Delta$ are similar. The graphs show ratio of actual error and the error for $\rho = \Delta\sqrt{2}/2$. It is clear that the ratio is limited approximately by 1.2.

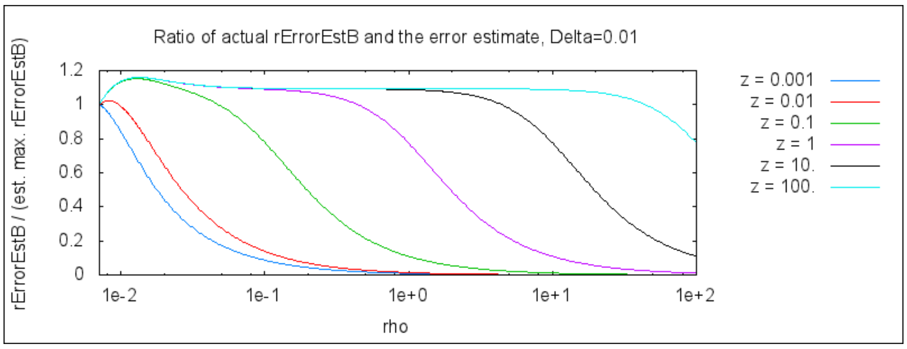
```
(%i46) wxplot2d(makelist(
           rErrorEstB(rho, z, Delta) /
           rErrorEstB(Delta*sqrt(2)/2, z, Delta), z, zlist),
         [rho, Delta*sqrt(2)/2, 10000*Delta], [logx],
         [legend, makelist(simplode(["z = ", z]), z, zlist)],
         [ylabel, "rErrorEstB / (est. max. rErrorEstB)"],
         [gnuplot_preamble,
          simplode(["set key outside; set format x '%.0te%+1T';",
          "set title 'Ratio of actual rErrorEstB and the error estimate, ",
          "Delta=", Delta, "';"])]),
       zlist=[1e-3, 1e-2, 1e-1, 1, 1e1], Delta=10e-6,
       wxplot_size=extsize, fpprintprec=2;
```

(%t46)



```
(%i47) wxplot2d(makelist(
           rErrorEstB(rho, z, Delta) /
           rErrorEstB(Delta*sqrt(2)/2, z, Delta), z, zlist),
         [rho, Delta*sqrt(2)/2, 10000*Delta], [logx],
         [legend, makelist(simplode(["z = ", z]), z, zlist)],
         [ylabel, "rErrorEstB / (est. max. rErrorEstB)"],
         [gnuplot_preamble,
           simplode(["set key outside; set format x '%.0te%+1T';",
           "set title 'Ratio of actual rErrorEstB and the error estimate, ",
           "Delta=", Delta, "';"])]),
       zlist=[1e-3, 1e-2, 1e-1, 1, 1e1, 1e2], Delta=1e-2,
       wxplot_size=extsize, fpprintprec=2;
```

(%t47)



In conclusion: maximum error of $r$ can be estimated as

$$1.2 \times \texttt{rError}(\rho, z, \texttt{maxRhoErrorB}(\rho, \Delta))$$

for $\rho = \Delta\sqrt{2}/2$.

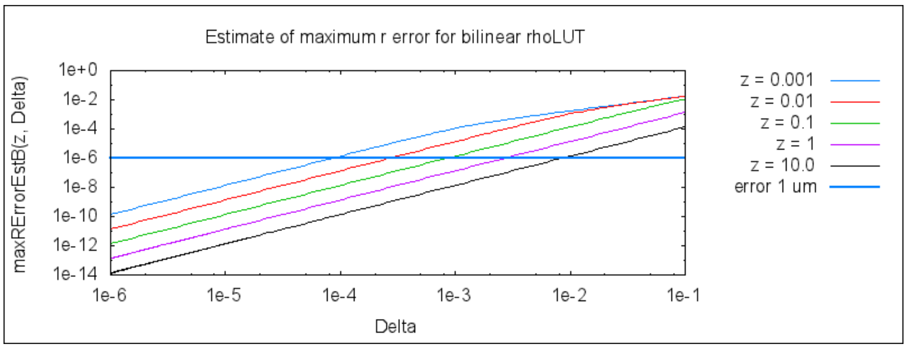
```
(%i48) maxRErrorEstB(z, Delta) := 1.2*rErrorEstB(Delta*sqrt(2)/2, z, Delta);
```

(%o48) $$\texttt{maxRErrorEstB}\,(z, \Delta) := 1.2\,\texttt{rErrorEstB}\left(\frac{\Delta\,\sqrt{2}}{2}, z, \Delta\right)$$

The graph below shows that `maxRErrorEstB` looks like a linear function in the log-log graph for wide range of $z$ and $\Delta$. Note that the value $r$ is used in the Rayleigh-Sommerfeld convolution kernel, where we can find the term $\exp(\mathbf{j}\,2\pi r/\lambda)$. The error of $r$ must be therefore much smaller than $\lambda$. For convenience, horizontal line in the graph shows error 1 $\mu$m, i.e. we are interested just in the errors well below this line.

This graph is referenced below in the text as *Estimate of maximum $r$ error for bilinear `rhoLUT`*.

```
(%i49) wxplot2d(append(makelist(maxRErrorEstB(z, Delta), z, zlist), [1e-6]),
         [Delta, 1e-6, 1e-1], [logx],
         [legend,
           append(makelist(simplode(["z = ", z]), z, zlist), ["error 1 um"])],
         [ylabel, "maxRErrorEstB(z, Delta)"], [logy],
         append([style],
           makelist([lines, 1, col], col, 1, length(zlist)), [[lines, 2, 1]]),
         [gnuplot_preamble,
           simplode(["set key outside; set format xy '%.0te%+1T';",
           "set title 'Estimate of maximum r error for bilinear rhoLUT';"])]),
       zlist=[1e-3, 1e-2, 1e-1, 1, 1e1], wxplot_size=extsize;
```

(%t49)


Linear approximation in the log-log graph is written as

$$\log(\texttt{maxRErrorEstB}) = \kappa \log(\Delta) + \xi(z)$$

where $\kappa$ is the slope and $\xi(z)$ the intercept. It also means

$$\texttt{maxRErrorEstB} = \exp(\kappa \log(\Delta) + \xi(z)) = \Delta^{\kappa} \exp(\xi(z))$$

Let us calculate $\kappa$, $\xi$ for $z = 1$.

We can estimate $\kappa$ for $10^{-6} < \Delta < 10^{-1}$ and $z = 1$:

```
(%i50) kappa : (log(float(maxRErrorEstB(z, DeltaMin)))
                - log(float(maxRErrorEstB(z, DeltaMax))))
          / (log(DeltaMin) - log(DeltaMax)),
       z=1, DeltaMin=1e-6, DeltaMax=1e-1;
```

(%o50) 1.999844918722706

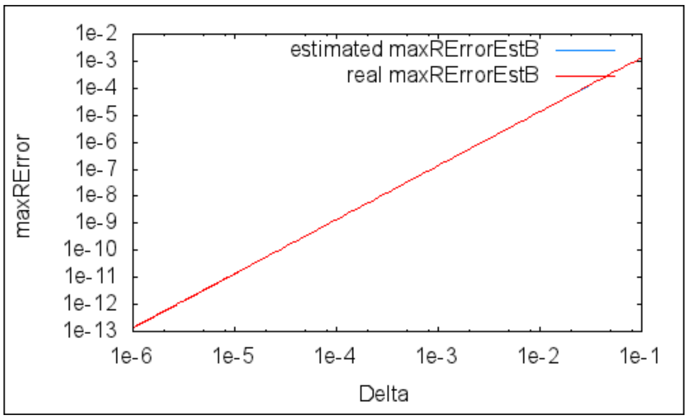
Just as an example, we can estimate $\xi(z)$ for the reference value $\Delta = 10^{-5}$ m:

```
(%i51) xi(z) := log(float(maxRErrorEstB(z, 10^(-5)))) - kappa*log(10^(-5));
```

(%o51) $\xi\left(z\right) := \log\left(\mathrm{float}\left(\mathtt{maxRErrorEstB}\left(z, 10^{-5}\right)\right)\right) - \kappa \log\left(10^{-5}\right)$

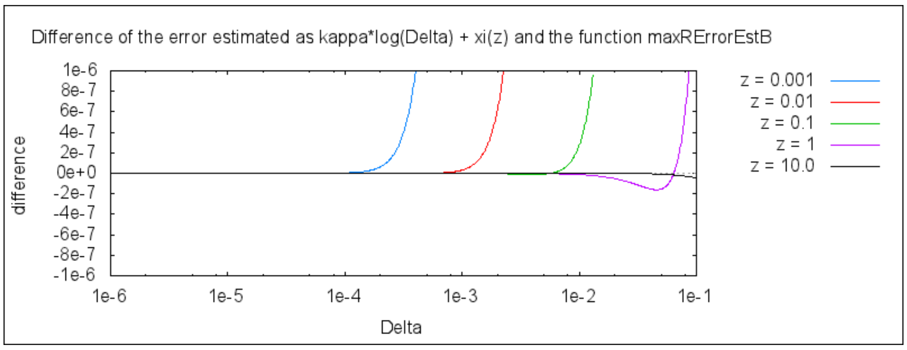
In the graph below we can see the values `maxRErrorEstB` estimated by a linear function are very close to the real ones.

```
(%i52) wxplot2d([Delta^kappa * exp(xi(z)), maxRErrorEstB(z, Delta)],
         [Delta, 1e-6, 1e-1],
         [legend, "estimated maxRErrorEstB", "real maxRErrorEstB"],
         [ylabel, "maxRError"],
         [logx], [logy], [gnuplot_preamble, "set format xy '%.0te%+1T'"]), z=1;
```

(%t52)


For better clarity, let us plot the difference between the linear estimate of the error and actual `maxRErrorEstB`. Clearly, difference 0 means the values are the same. As soon as the difference is much smaller than the wavelength, it can be regarded as precise enough. For visible light, the difference should be much smaller than 1 $\mu$m.

```
(%i53) wxplot2d(
         makelist(Delta^kappa * exp(xi(z)) - maxRErrorEstB(z,Delta), z, zlist),
         [Delta, 1e-6, 1e-1],
         [legend, makelist(simplode(["z = ", z]), z, zlist)],
         [ylabel, "difference"], [logx], [y, -1e-6, 1e-6],
         [gnuplot_preamble,
           simplode(["set format xy '%.0te%+1T'; set key outside;",
           "set title 'Difference of the error estimated as ",
           "kappa*log(Delta) + xi(z) and the function maxRErrorEstB';"])]),
         zlist=[1e-3, 1e-2, 1e-1, 1, 1e1], wxplot_size=extsize;
```

(%t53)


Function $\xi(z)$, i.e. the intercept of the line, seems to depend on $\log(z)$, see the graph (%t49) *Estimate of maximum $r$ error for bilinear* `rhoLUT` – we can see that the lines are equally spaced when $z$ follows a geometric sequence. Therefore we can estimate

$$\xi(z) = \xi_0 + \xi_1 \log(z),$$

where

$$\xi_0 = \xi(1)$$

Let us estimate $\xi_1$ for a wide range of $z$ values:

```
(%i54) xi_0 : float(xi(1));
```

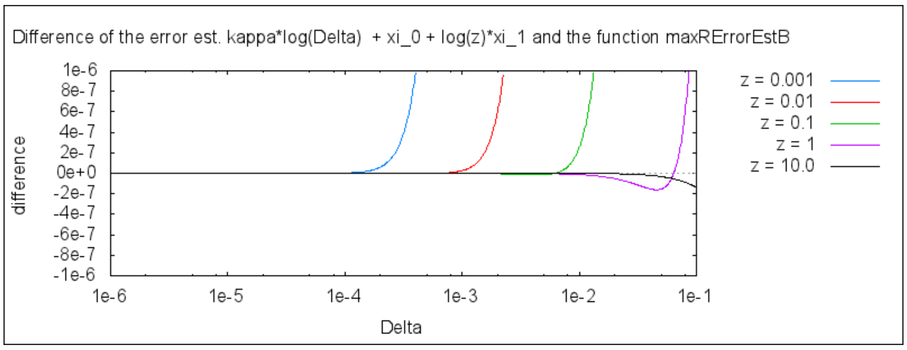
(%o54) $-1.988492520317706$

```
(%i55) xi_1 : float((xi(zMin) - xi(zMax)) / (log(zMin) - log(zMax))),
         zMin = 1e-3, zMax = 1e1;
```

(%o55) $-0.99988206363115$

Again, let us check if the estimate is good by graphing difference between linear approximation and the function `maxRErrorEstB`. Again, for visible light, the difference should be much smaller than 1 $\mu$m.

```
(%i56) wxplot2d(makelist(
         Delta^kappa * exp(xi_0) * z^xi_1 - maxRErrorEstB(z, Delta), z, zlist),
        [Delta, 1e-6, 1e-1],
        [legend, makelist(simplode(["z = ", z]), z, zlist)],
        [ylabel, "difference"], [logx], [y, -1e-6, 1e-6],
        [gnuplot_preamble,
          simplode(["set format xy '%.0te%+1T'; set key outside;",
         "set title 'Difference of the error est. ",
         "kappa*log(Delta) + xi_0 + log(z)*xi_1",
         " and the function maxRErrorEstB';"])]),
       zlist=[1e-3, 1e-2, 1e-1, 1, 1e1], wxplot_size=extsize;
```

(%t56)


In conclusion, for reasonable range of $z$ and $\Delta$ we can say that the maximum error of $r$ is

$$\begin{aligned}\log(\texttt{maxRErrorEstB}) &= \kappa\log(\Delta) + \xi(z)\\ &= \kappa\log(\Delta) + \xi_0 + \xi_1\log(z),\end{aligned}$$

which is equivalent to

$$\texttt{maxErrorR} = \Delta^\kappa \exp(\xi_0) z^{\xi_1}.$$

If we want to limit `maxErrorR` (by, e.g., $\lambda/100$), we must solve the equation for $\Delta$:

```
(%i57) DeltaEst(z, maxErrorRPossible) :=
         (maxErrorRPossible * z^(-xi_1) * exp(-xi_0))^(1/kappa);
```

(%o57) $$\texttt{DeltaEst}(z, \texttt{maxErrorRPossible}) := \left(\texttt{maxErrorRPossible}\, z^{-\xi_1}\exp(-\xi_0)\right)^{1/\kappa}$$

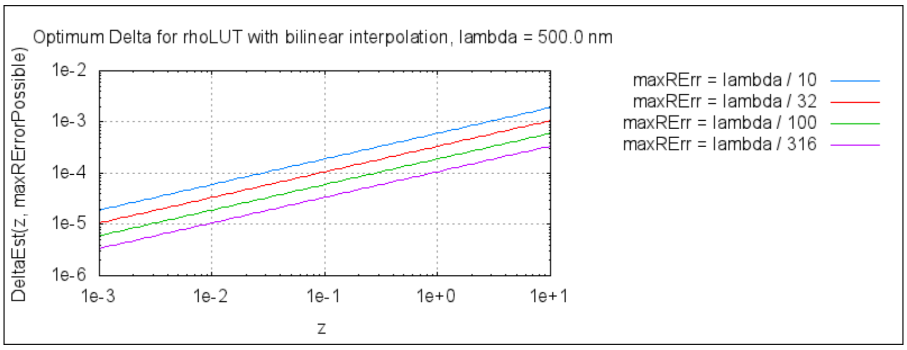
This equation should be calculated for following $\kappa$, $\xi_0$ and $\xi_1$:

```
(%i58) [kappa, xi_0, xi_1];
```

(%o58) $[1.999844918722706, -1.988492520317706, -0.99988206363115]$

The graph below shows various optimal $\Delta$ for `rhoLUT` construction depending on maximum error of $r$ we are willing to accept.

```
(%i59) wxplot2d(makelist(DeltaEst(z, lambda/maxerr), maxerr, maxRErrList),
         [z, 1e-3, 1e1], [logx], [logy],
         [ylabel, "DeltaEst(z, maxRErrorPossible)"],
         [legend, makelist(
           simplode(["maxRErr = lambda / ", maxerr]), maxerr, maxRErrList)],
         [gnuplot_preamble,
          simplode(["set title 'Optimum Delta for rhoLUT with bilinear",
          " interpolation, lambda = ", lambda*1e9, " nm';",
          "set format xy '%.0te%+1T'; set key outside; set grid"])]),
       lambda=500e-9, maxRErrList=[10, 32, 100, 316], wxplot_size=extsize;
```

(%t59)


Example values of optimal $\Delta$ for various propagation distances $z$ and $\lambda =$ 500 nm:

```
(%i60) makelist(float(DeltaEst(z, lambda/100)), z, [1e-3, 1e-2, 1e-1, 1]),
       lambda=500e-9;
```

(%o60) $$[6.0402218590023497 \times 10^{-6}, 1.9099970269534125 \times 10^{-5}, 6.0396600127092595 \times 10^{-5}, 1.9098193638187734 \times 10^{-4}]$$

## 5 Alternative analysis of the r error near the origin

(Some steps in this section are based on ideas of Libor Váša. Thank you!)

This section shows two alternative ways how to derive the estimate value of $\Delta$. They are shorter than derivation presented in previous sections. However, their assumptions are less supported by evidence.

Let us analyse the error of $r$ with bilinear `rhoLUT` – the function `rErrorB` – again.

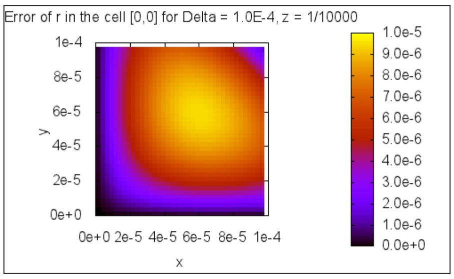
```
(%i61) rErrorB(x, y, x0, y0, Delta, z) :=
         r(rhoB(x, y, x0, y0, Delta), z) - r(rho(x, y), z);
```

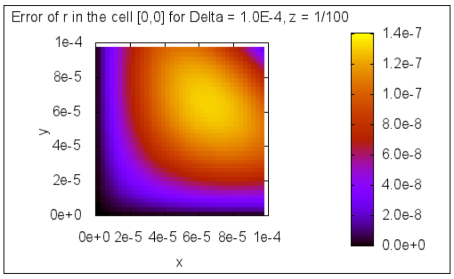
(%o61) $$\texttt{rErrorB}\left(x, y, x0, y0, \Delta, z\right) := r\left(\texttt{rhoB}\left(x, y, x0, y0, \Delta\right), z\right) - r\left(\rho\left(x, y\right), z\right)$$

In the previous text we have shown that the maximum error of $r$ can be found in the vicinity of the origin. In fact, we have estimated it as an multiple of the error in the center of the cell $[0, 0]$, i.e. at the point $(\Delta/2, \Delta/2)$. If we actually display the $r$ error in this cell, we clearly see the error is maximum at different point (we reflected this fact in previous text by factor 1.2)

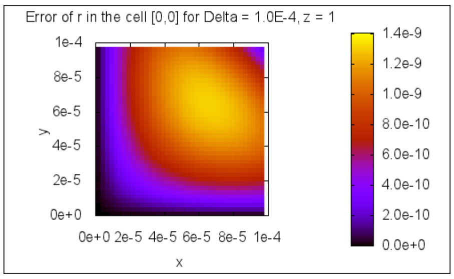
```
(%i62)  makelist(
        wxplot3d(rErrorB(x, y, x0, y0, Delta, z), [x,0,Delta], [y,0,Delta],
        [mesh_lines_color, false], [grid, 40, 40], [zlabel, "rErrorB"],
        [legend, false],
        [gnuplot_preamble,
          simplode(["set view map; set size ratio -1; set size 1.2;",
          " set origin -0.2,-0.06;",
          "set colorbox user origin 0.78, 0.1 size 0.05, 0.8;",
          " set format xy '%.0te%+1T';",
          "set format cb '%.1te%+1T'; set key top;",
          "set title 'Error of r in the cell [",
          x0, ",", y0, "] for Delta = ", Delta, ", z = ", z,
          "'"])]), z, zList),
        x0=0, y0=0, Delta=100e-6, zList=makelist(10^(2*x), x, -2, 1),
        fpprintprec=2$
```

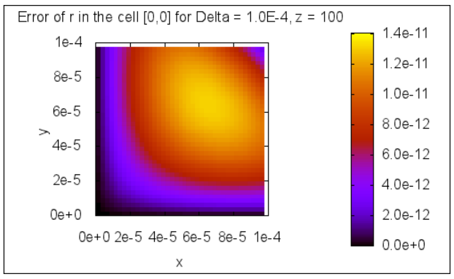
(%t62)
(%t63)





(%t64)
(%t65)





It is quite difficult to find the extremal point exactly. We can, however, take an approximate approach, which is not rigorous but leads to remarkably good results.

We are looking for the maximum difference between $r(\texttt{rhoB}(x, y, x_0, y_0, \Delta), z)$ and $r(\rho(x, y), z)$. Let us guess that this difference is maximal when difference of their squares is maximal:

```
(%i66)  rErrorSqBi00(x, y, Delta) :=
          ''(r(rhoB(x, y, 0, 0, Delta), z0)^2 - r(rho(x, y), z0)^2);
```

(%o66) $\texttt{rErrorSqBi00}\,(x, y, \Delta) :=$

$$\left(x\left(1-\frac{y}{\Delta}\right)+\left(1-\frac{x}{\Delta}\right)y+\frac{\sqrt{2}\,x\,y}{\Delta}\right)^2-y^2-x^2$$

Now we can easily find the extremal point. Note that we are taking into account just the first cell of the `rhoLUT` (corner $[0,0]$), so we are interested just in $0<x<\Delta$, $0<y<\Delta$:

```
(%i67)  solve([diff(rErrorSqBi00(x, y, Delta), x, 1)=0,
              diff(rErrorSqBi00(x, y, Delta), y, 1)=0], [x, y]);
```

(%o67) $$[[x=0,y=0],[x=0,y=\frac{\left(\sqrt{2}+2\right)\,\Delta}{2}],[x=\frac{\left(\sqrt{2}+2\right)\,\Delta}{2},y=0],$$
$$[x=\frac{\left(\left(7\,\sqrt{2}+10\right)\,\sqrt{17-3\,2^{\frac{5}{2}}\,\sqrt{5}}+3\,\sqrt{2}+6\right)\,\Delta}{4},$$
$$y=\frac{\left(7\,\sqrt{2}+10\right)\,\sqrt{17-3\,2^{\frac{5}{2}}\,\sqrt{5}}\,\Delta+\left(3\,\sqrt{2}+6\right)\,\Delta}{4}],$$
$$[x=-\frac{\left(\left(7\,\sqrt{2}+10\right)\,\sqrt{17-3\,2^{\frac{5}{2}}\,\sqrt{5}}-3\,\sqrt{2}-6\right)\,\Delta}{4},$$
$$y=-\frac{\left(7\,\sqrt{2}+10\right)\,\sqrt{17-3\,2^{\frac{5}{2}}\,\sqrt{5}}\,\Delta+\left(-3\,\sqrt{2}-6\right)\,\Delta}{4}]]$$

```
(%i68)  float(%);
```

(%o68) $$[[x=0.0,y=0.0],[x=0.0,y=1.707106781186548\,\Delta],$$
$$[x=1.707106781186548\,\Delta,y=0.0],$$
$$[x=4.469263575571515\,\Delta,y=4.469263575571515\,\Delta],$$
$$[x=0.65205676798813\,\Delta,y=0.65205676798813\,\Delta]]$$

Clearly, the last solution is the one we are looking for. Note that $x$ and $y$ are just multiples of $\Delta$, i.e. the maximum is found at the same relative position inside the cell $[0,0]$. The color visualisations of the $r$ error confirm this. Let us call the relative positions `xCritical` and `yCritical`:

```
(%i69)  xCritical : ''(part(part(part(''(%th(1))[5], 1), 2), 1));
```

(%o69) $0.65205676798813$

```
(%i70)  yCritical : ''(part(part(part(''(%th(2))[5], 1), 2), 1));
```

(%o70) $0.65205676798813$

It follows it would be better to approximate the maximum error by the error found at (`xCritical` $\times\,\Delta$, `yCritical` $\times\,\Delta$) and not at the point $(\Delta/2,\Delta/2)$ as before.

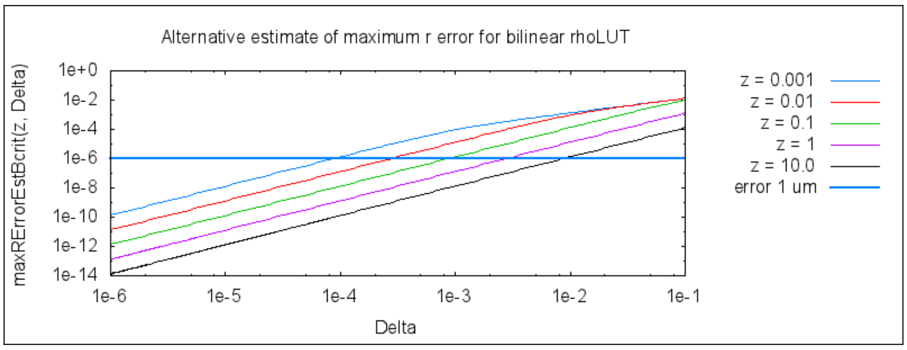
```
(%i71) maxRErrorEstBcrit(z, Delta) :=
         r(rhoB(xCritical*Delta, yCritical*Delta, 0, 0, Delta), z)
         - r(rho(xCritical*Delta, yCritical*Delta), z);
```

(%o71) $$\texttt{maxRErrorEstBcrit}\left(z,\Delta\right):=r\left(\texttt{rhoB}\left(\texttt{xCritical}\,\Delta,\texttt{yCritical}\,\Delta,0,0,\Delta\right),z\right)-r\left(\rho\left(\texttt{xCritical}\,\Delta,\texttt{yCritical}\,\Delta\right),z\right)$$

We can draw behaviour of this estimate in the same way as in the graph `(%t49)` *Estimate of maximum r error for bilinear **rhoLUT***. We can see it is very similar.

```
(%i72) wxplot2d(
         append(makelist(maxRErrorEstBcrit(z, Delta), z, zlist), [1e-6]),
         [Delta, 1e-6, 1e-1], [logx],
         [legend, append(makelist(simplode(["z = ", z]), z, zlist),
           ["error 1 um"])],
         [ylabel, "maxRErrorEstBcrit(z, Delta)"], [logy],
         append([style],
           makelist([lines, 1, col], col, 1, length(zlist)), [[lines, 2, 1]]),
         [gnuplot_preamble,
          simplode(["set title 'Alternative estimate of maximum r error",
          " for bilinear rhoLUT';",
          "set key outside; set format xy '%.0te%+1T'"])]),
       zlist=[1e-3, 1e-2, 1e-1, 1, 1e1], wxplot_size=extsize;
```

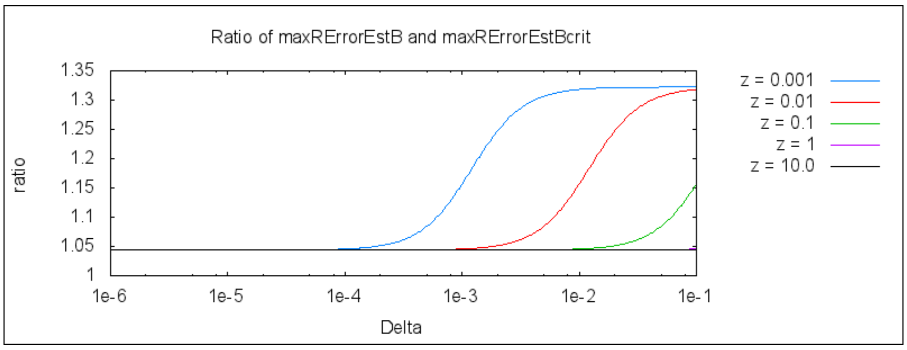
(%t72)



We can also show the ratio of these two error estimates: values near 1 indicate they are the same, which is mostly the case. As the values in the graph are bigger than 1, the former error estimate from the point $(\Delta/2, \Delta/2)$ is a bit pessimistic.

```
(%i73) wxplot2d(makelist(
           bfloat(maxRErrorEstB(z, Delta)/maxRErrorEstBcrit(z, Delta)),
           z, zlist),
         [Delta, 1e-6, 1e-1], [logx],
         [legend, makelist(simplode(["z = ", z]), z, zlist)],
         [ylabel, "ratio"],
         [gnuplot_preamble, simplode([
           "set title 'Ratio of maxRErrorEstB and maxRErrorEstBcrit';",
           "set key outside; set format x '%.0te%+1T'"])]),
       zlist=[1e-3, 1e-2, 1e-1, 1, 1e1], wxplot_size=extsize, fpprec=30;
```

(%t73)



Let us again try to find parameters $\kappa$, $\xi_0$ and $\xi_1$, now for the error estimated at $(\texttt{xCritical} \times \Delta, \texttt{yCritical} \times \Delta)$:

```
(%i74) kappaCrit : (log(float(maxRErrorEstBcrit(z, DeltaMin)))
                    - log(float(maxRErrorEstBcrit(z, DeltaMax))))
                   / (log(DeltaMin) - log(DeltaMax)),
       z=1, DeltaMin=1e-6, DeltaMax=1e-1;
```

(%o74) 1.999680826343935

```
(%i75) xiCrit(z) :=
         log(float(maxRErrorEstBcrit(z, 1e-5))) - kappaCrit*log(1e-5)$

(%i76) xiCrit_0 : float(xiCrit(1));
```

(%o76) $-2.033290245705118$

```
(%i77) xiCrit_1 :
         float((xiCrit(zMin) - xiCrit(zMax)) / (log(zMin) - log(zMax))),
       zMin = 1e-3, zMax = 1e1;
```

(%o77) $-1.000089302501926$

Compare them to the values found for error estimate at the point $(\Delta/2, \Delta/2)$ and the multiplication factor 1.2 – they are very similar:

```
(%i78) [kappa, xi_0, xi_1];
```

(%o78) $[1.999844918722706, -1.988492520317706, -0.99988206363115]$

At the end we will show that the equation `(%o57)` for `DeltaEst` can be also found using Taylor expansion of `rErrorB`. First, we take Taylor expansion at $(x, y) = (\texttt{xCritical} \times \Delta, \texttt{yCritical} \times \Delta)$

```
(%i79)  assume(maxRErr>0);
```

(%o79) $[\texttt{maxRErr} > 0]$

```
(%i80)  float(taylor(rErrorB(xCritical*Delta, yCritical*Delta, 0, 0, Delta, z),
          Delta, 0, 2))
        , ratprint: false;
```

(%o80) $$\frac{0.13138723649657\,\Delta^2}{z}$$

Now we can easily solve for $\Delta$:

```
(%i81)  sublist(float(solve(%=maxRErr, Delta)), lambda([x], rhs(x)>0)),
        ratprint:false;
```

(%o81) $[\Delta = 2.758820287226045\,\sqrt{z}\,\sqrt{\texttt{maxRErr}}]$

Just as a matter of interest, let us try to apply the same procedure to 1.2 multiple of the error at $(\Delta/2, \Delta/2)$. We can see that this estimate is more conservative, i.e. it chooses smaller Delta than the previous equation.

```
(%i82)  float(taylor(1.2*rErrorB(0.5*Delta, 0.5*Delta, 0, 0, Delta, z),
          Delta, 0, 2)),
        ratprint: false;
```

(%o82) $$\frac{0.13713203435596\,\Delta^2}{z}$$

```
(%i83)  sublist(float(solve(%=maxRErr, Delta)), lambda([x], rhs(x)>0)),
        ratprint:false;
```

(%o83) $[\Delta = 2.70041517962955\,\sqrt{z}\,\sqrt{\texttt{maxRErr}}]$

Please note that the equation `(%o57)`, repeated here for convenience

$$\texttt{DeltaEst}\,(z, \texttt{maxErrorRPossible}) := \left(\texttt{maxErrorRPossible}\, z^{-\xi_1} \exp(-\xi_0)\right)^{\frac{1}{\kappa}}$$

was supposed to work with values

$$\begin{aligned} \kappa &= 1.999844918722706 \approx 2 \\ \xi_0 &= -1.988492520317706 \\ \xi_1 &= -0.99988206363115 \approx -1 \end{aligned}$$

which approximately leads to

$$\begin{aligned}\texttt{DeltaEst}\,(z, \texttt{maxErrorRPossible}) &\approx \sqrt{\texttt{maxErrorRPossible}\, z \exp(-\xi_0)} \\ &\approx 2.703\sqrt{z}\sqrt{\texttt{maxErrorRPossible}},\end{aligned}$$

that is, we came to virtually the same estimate in a different way.

# Appendix C

# Time measurement of the DLUT method performance

**Double lookup table method for fast light propagation calculations**

In this test, **a single point light source at (0, 0, -3 mm)** is propagated to a plane z=0, sampling distance 1 um, lambda = 635 nm. XY size of the target area varies, see "screen side" column.
Execution time was measured for precise algorithm (no look-up tables, simple propagation kernel), algorithm with waveLUT only and algorithm with both rhoLUT and waveLUT (= double LUT or dLUT for short).
**The rhoLUT uses constant interpolation, i.e. it must be large. The implementation of the rhoLUT is not optimized.**
Table shows complete times as well as partial times used for look-up tables generation and propagation itself. Actually, the calculation ran 10 times and the total time is shown here; single calculation is therefore 1/10 of the times below.
The analysis (blue) shows speed-up of the particular method, where speed-up > 1 means that the method is faster than the reference. It also shows how much time was spent on tables generation etc in [%].
The summary (green) shows minimum, maximum and average values of the matching columns above.
For example, wLUT-only method is **1.67×** faster (on average) than precise calculation. The waveLUT generation took **4.86%** of the calculation. After introduction of the rhoLUT, the average speed-up is just **1.11×**. In comparison to waveLUT only, rhoLUT+waveLUT method is **0.66×** faster, i.e. **1/0.66 = 1.5×** slower.

| | precise | waveLUT | | | | dLUT (rhoLUT + waveLUT) | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| screen side [samples] | complete [s] | waveLUT size [samples] | complete [s] | propagation [s] | waveLUT gen. [s] | rhoLUT side size [samples] | waveLUT size [samples] | complete [s] | propagation [s] | rhoLUT gen. [s] | waveLUT gen. [s] |
| 64 | 0,00645292 | 358 | 0,003809 | 0,00325883 | 0,0005425 | 33 | 364 | 0,00558208 | 0,00483117 | 0,000190028 | 0,000556293 |
| 128 | 0,0220594 | 720 | 0,0140387 | 0,0129794 | 0,00105358 | 65 | 726 | 0,0209483 | 0,0191538 | 0,000729081 | 0,0010601 |
| 256 | 0,0884375 | 1444 | 0,0539347 | 0,0518391 | 0,00208801 | 129 | 1450 | 0,0814942 | 0,0763799 | 0,00297226 | 0,00213169 |
| 512 | 0,356299 | 2892 | 0,211467 | 0,2072 | 0,00425418 | 257 | 2898 | 0,320704 | 0,305198 | 0,0113109 | 0,00418522 |
| 1024 | 1,4319 | 5788 | 0,837171 | 0,828658 | 0,00848615 | 513 | 5794 | 1,27723 | 1,22145 | 0,0473665 | 0,00838998 |
| 2048 | 5,656 | 11581 | 3,29325 | 3,27634 | 0,0168773 | 1025 | 11587 | 5,0415 | 4,83528 | 0,189216 | 0,0169769 |

| screen side [samples] | | | comparison precise / waveLUT (>1 = good) | propagation [%] | waveLUT gen. [%] | | comparison waveLUT / dLUT (>1 = good) | comparison precise / rhoLUT (>1 = good) | propagation [%] | rhoLUT gen. [%] | waveLUT gen. [%] |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 64 | | | 1,69 | 85,56 | 14,24 | | 0,68 | 1,16 | 86,55 | 3,40 | 9,97 |
| 128 | | | 1,57 | 92,45 | 7,50 | | 0,67 | 1,05 | 91,43 | 3,48 | 5,06 |
| 256 | | | 1,64 | 96,11 | 3,87 | | 0,66 | 1,09 | 93,72 | 3,65 | 2,62 |
| 512 | | | 1,68 | 97,98 | 2,01 | | 0,66 | 1,11 | 95,17 | 3,53 | 1,31 |
| 1024 | | | 1,71 | 98,98 | 1,01 | | 0,66 | 1,12 | 95,63 | 3,71 | 0,66 |
| 2048 | | | 1,72 | 99,49 | 0,51 | | 0,65 | 1,12 | 95,91 | 3,75 | 0,34 |
| | | | | | | | | | | | |
| | | min | 1,57 | 85,56 | 0,51 | | 0,65 | 1,05 | 86,55 | 3,40 | 0,34 |
| | | avg | 1,67 | 95,10 | 4,86 | | 0,66 | 1,11 | 93,07 | 3,59 | 3,32 |
| | | max | 1,72 | 99,49 | 14,24 | | 0,68 | 1,16 | 95,91 | 3,75 | 9,97 |

**Double lookup table method for fast light propagation calculations**

In this test, **a single point light source at (0, 0, -3 mm)** is propagated to a plane z=0, sampling distance 1 um, lambda = 635 nm. XY size of the target area varies, see "screen side" column.
Execution time was measured for precise algorithm (no look-up tables, simple propagation kernel), algorithm with waveLUT only and algorithm with both rhoLUT and waveLUT (= double LUT or dLUT for short).
**The rhoLUT uses bilinear interpolation, i.e. it can be small. The implementation of the rhoLUT is not optimized.**
Table shows complete times as well as partial times used for look-up tables generation and propagation itself. Actually, the calculation ran 10 times and the total time is shown here; single calculation is therefore 1/10 of the times below.
The analysis (blue) shows speed-up of the particular method, where speed-up > 1 means that the method is faster than the reference. It also shows how much time was spent on tables generation etc in [%].
The summary (green) shows minimum, maximum and average values of the matching columns above.
For example, wLUT-only method is **1.63×** faster (on average) than precise calculation. The waveLUT generation took **4.92%** of the calculation. After introduction of the rhoLUT, the average speed-up is just **0.96×**. In comparison to waveLUT only, rhoLUT+waveLUT method is **0.59×** faster, i.e. **1/0.59 = 1.7×** slower.

| | precise | waveLUT | | | | dLUT (rhoLUT + waveLUT) | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| screen side [samples] | complete [s] | waveLUT size [samples] | complete [s] | propagation [s] | waveLUT gen. [s] | rhoLUT side size [samples] | waveLUT size [samples] | complete [s] | propagation [s] | rhoLUT gen. [s] | waveLUT gen. [s] |
| 64 | 0,00552653 | 358 | 0,00382624 | 0,00325922 | 0,000552845 | 4 | 533 | 0,00624987 | 0,00544646 | 1,53E-05 | 0,000783867 |
| 128 | 0,0221176 | 720 | 0,0140548 | 0,0129802 | 0,00106891 | 6 | 887 | 0,0234026 | 0,0220904 | 2,15E-05 | 0,00128499 |
| 256 | 0,0886628 | 1444 | 0,0539669 | 0,0518345 | 0,00212518 | 10 | 1596 | 0,0917469 | 0,0893984 | 3,29E-05 | 0,00230793 |
| 512 | 0,357005 | 2892 | 0,21146 | 0,207192 | 0,00425725 | 18 | 3014 | 0,361636 | 0,357223 | 7,85E-05 | 0,00432545 |
| 1024 | 1,43306 | 5788 | 0,837332 | 0,828673 | 0,00863365 | 34 | 5849 | 1,4405 | 1,43176 | 0,00027125 | 0,00843826 |
| 2048 | 5,656 | 11581 | 3,29342 | 3,27651 | 0,0168884 | 67 | 11696 | 5,68567 | 5,66782 | 0,000804939 | 0,0170179 |
| screen side [samples] | | | comparison precise / waveLUT (>1 = good) | propagation [%] | waveLUT gen. [%] | | comparison waveLUT / dLUT (>1 = good) | comparison precise / rhoLUT (>1 = good) | propagation [%] | rhoLUT gen. [%] | waveLUT gen. [%] |
| 64 | | | 1,44 | 85,18 | 14,45 | | 0,61 | 0,88 | 87,15 | 0,25 | 12,54 |
| 128 | | | 1,57 | 92,35 | 7,61 | | 0,60 | 0,95 | 94,39 | 0,09 | 5,49 |
| 256 | | | 1,64 | 96,05 | 3,94 | | 0,59 | 0,97 | 97,44 | 0,04 | 2,52 |
| 512 | | | 1,69 | 97,98 | 2,01 | | 0,58 | 0,99 | 98,78 | 0,02 | 1,20 |
| 1024 | | | 1,71 | 98,97 | 1,03 | | 0,58 | 0,99 | 99,39 | 0,02 | 0,59 |
| 2048 | | | 1,72 | 99,49 | 0,51 | | 0,58 | 0,99 | 99,69 | 0,01 | 0,30 |
| | | min | 1,44 | 85,18 | 0,51 | | 0,58 | 0,88 | 87,15 | 0,01 | 0,30 |
| | | avg | 1,63 | 95,00 | 4,92 | | 0,59 | 0,96 | 96,14 | 0,07 | 3,77 |
| | | max | 1,72 | 99,49 | 14,45 | | 0,61 | 0,99 | 99,69 | 0,25 | 12,54 |

## Double lookup table method for fast light propagation calculations

In this test, **a single point light source at (0, 0, -3 mm)** is propagated to a plane z=0, sampling distance 1 um, lambda = 635 nm. XY size of the target area varies, see "screen side" column.
Execution time was measured for precise algorithm (no look-up tables, simple propagation kernel), algorithm with waveLUT only and algorithm with both rhoLUT and waveLUT (= double LUT or dLUT for short).
**The rhoLUT uses constant interpolation, i.e. it must be large. The implementation of the rhoLUT is optimized and uses integer arithmetic whenever possible.**
Table shows complete times as well as partial times used for look-up tables generation and propagation itself. Actually, the calculation ran 10 times and the total time is shown here; single calculation is therefore 1/10 of the times below.
The analysis (blue) shows speed-up of the particular method, where speed-up > 1 means that the method is faster than the reference. It also shows how much time was spent on tables generation etc in [%].
The summary (green) shows minimum, maximum and average values of the matching columns above.
For example, wLUT-only method is **1.71×** faster (on average) than precise calculation. The waveLUT generation took 5**.16%** of the calculation. After introduction of the rhoLUT, the average speed-up is **2.23×**. In comparison to waveLUT only, rhoLUT+waveLUT method is **1.31×** faster.

| | precise | waveLUT | | | | dLUT (rhoLUT + waveLUT) | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| screen side<br>[samples] | complete<br>[s] | waveLUT size<br>[samples] | complete<br>[s] | propagation<br>[s] | waveLUT gen.<br>[s] | rhoLUT side size<br>[samples] | waveLUT size<br>[samples] | complete<br>[s] | propagation<br>[s] | rhoLUT gen.<br>[s] | waveLUT gen.<br>[s] |
| 64 | 0,00551389 | 358 | 0,00367146 | 0,00310367 | 0,000555527 | 33 | 364 | 0,00290866 | 0,00216962 | 0,000188496 | 0,000544799 |
| 128 | 0,0220211 | 720 | 0,0133556 | 0,0122909 | 0,00105703 | 65 | 726 | 0,0103558 | 0,00856124 | 0,000727548 | 0,00106086 |
| 256 | 0,0884816 | 1444 | 0,0511379 | 0,0489883 | 0,0021428 | 129 | 1450 | 0,0391991 | 0,0340718 | 0,00297801 | 0,00213935 |
| 512 | 0,356289 | 2892 | 0,201057 | 0,196715 | 0,00433043 | 257 | 2898 | 0,151425 | 0,136104 | 0,0110649 | 0,00424729 |
| 1024 | 1,43227 | 5788 | 0,795204 | 0,786762 | 0,00841757 | 513 | 5794 | 0,600556 | 0,545726 | 0,0463857 | 0,00841719 |
| 2048 | 5,65545 | 11581 | 3,1638 | 3,14669 | 0,0170807 | 1025 | 11587 | 2,38808 | 2,18478 | 0,186252 | 0,0170167 |

| screen side<br>[samples] | | | comparison<br>precise / waveLUT<br>(>1 = good) | propagation<br>[%] | waveLUT gen.<br>[%] | | comparison<br>waveLUT / dLUT<br>(>1 = good) | comparison<br>precise / rhoLUT<br>(>1 = good) | propagation<br>[%] | rhoLUT gen.<br>[%] | waveLUT gen.<br>[%] |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 64 | | | 1,50 | 84,54 | 15,13 | | 1,26 | 1,90 | 74,59 | 6,48 | 18,73 |
| 128 | | | 1,65 | 92,03 | 7,91 | | 1,29 | 2,13 | 82,67 | 7,03 | 10,24 |
| 256 | | | 1,73 | 95,80 | 4,19 | | 1,30 | 2,26 | 86,92 | 7,60 | 5,46 |
| 512 | | | 1,77 | 97,84 | 2,15 | | 1,33 | 2,35 | 89,88 | 7,31 | 2,80 |
| 1024 | | | 1,80 | 98,94 | 1,06 | | 1,32 | 2,38 | 90,87 | 7,72 | 1,40 |
| 2048 | | | 1,79 | 99,46 | 0,54 | | 1,32 | 2,37 | 91,49 | 7,80 | 0,71 |
| | | min | 1,50 | 84,54 | 0,54 | | 1,26 | 1,90 | 74,59 | 6,48 | 0,71 |
| | | avg | 1,71 | 94,77 | 5,16 | | 1,31 | 2,23 | 86,07 | 7,32 | 6,56 |
| | | max | 1,80 | 99,46 | 15,13 | | 1,33 | 2,38 | 91,49 | 7,80 | 18,73 |

**Double lookup table method for fast light propagation calculations**

In this test, **100 point light sources at z=-3 mm** are propagated to a plane z=0, sampling distance 1 um, lambda = 635 nm. XY size of the target area varies, see "screen side" column.
Execution time was measured for precise algorithm (no look-up tables, simple propagation kernel), algorithm with waveLUT only and algorithm with both rhoLUT and waveLUT (= double LUT or dLUT for short).
**The rhoLUT uses constant interpolation, i.e. it must be large. The implementation of the rhoLUT is not optimized.**
Table shows complete times as well as partial times used for look-up tables generation and propagation itself. Actually, the calculation ran 10 times and the total time is shown here; single calculation is therefore 1/10 of the times below.
The analysis (blue) shows speed-up of the particular method, where speed-up > 1 means that the method is faster than the reference. It also shows how much time was spent on tables generation etc in [%].
The summary (green) shows minimum, maximum and average values of the matching columns above.
For example, wLUT-only method is **1.71×** faster (on average) than precise calculation. The waveLUT generation took **0.13%** of the calculation. After introduction of the rhoLUT, the average speed-up is just **1.16×**. In comparison to waveLUT only, rhoLUT+waveLUT method is **0.68×** faster, i.e. **1/0.68 = 1.5×** slower.

| | precise | waveLUT | | | | dLUT (rhoLUT + waveLUT) | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| screen side [samples] | complete [s] | waveLUT size [samples] | complete [s] | propagation [s] | waveLUT gen. [s] | rhoLUT side size [samples] | waveLUT size [samples] | complete [s] | propagation [s] | rhoLUT gen. [s] | waveLUT gen. [s] |
| 64 | 0,554523 | 1173 | 0,32769 | 0,325939 | 0,00171562 | 133 | 1088 | 0,483627 | 0,481276 | 0,000745172 | 0,00157616 |
| 128 | 2,2135 | 1543 | 1,2998 | 1,29749 | 0,00226885 | 165 | 1410 | 1,91588 | 1,91183 | 0,00192365 | 0,0020907 |
| 256 | 8,85543 | 2275 | 5,18428 | 5,1808 | 0,00342434 | 229 | 2093 | 7,63252 | 7,6245 | 0,00490971 | 0,00305999 |
| 512 | 35,6243 | 3730 | 20,7198 | 20,7142 | 0,00555603 | 357 | 3509 | 30,4898 | 30,4692 | 0,0153866 | 0,0051028 |
| 1024 | 143,26 | 6631 | 83,0653 | 83,0554 | 0,00985083 | 613 | 6385 | 122,22 | 122,155 | 0,0554213 | 0,00945161 |
| 2048 | 564,63 | 12426 | 327,723 | 327,704 | 0,0183389 | 1125 | 12166 | 482,571 | 482,347 | 0,204995 | 0,018214 |
| screen side [samples] | | | comparison precise / waveLUT (>1 = good) | propagation [%] | waveLUT gen. [%] | | comparison waveLUT / dLUT (>1 = good) | comparison precise / rhoLUT (>1 = good) | propagation [%] | rhoLUT gen. [%] | waveLUT gen. [%] |
| 64 | | | 1,69 | 99,47 | 0,52 | | 0,68 | 1,15 | 99,51 | 0,15 | 0,33 |
| 128 | | | 1,70 | 99,82 | 0,17 | | 0,68 | 1,16 | 99,79 | 0,10 | 0,11 |
| 256 | | | 1,71 | 99,93 | 0,07 | | 0,68 | 1,16 | 99,89 | 0,06 | 0,04 |
| 512 | | | 1,72 | 99,97 | 0,03 | | 0,68 | 1,17 | 99,93 | 0,05 | 0,02 |
| 1024 | | | 1,72 | 99,99 | 0,01 | | 0,68 | 1,17 | 99,95 | 0,05 | 0,01 |
| 2048 | | | 1,72 | 99,99 | 0,01 | | 0,68 | 1,17 | 99,95 | 0,04 | 0,00 |
| | | min | 1,69 | 99,47 | 0,01 | | 0,68 | 1,15 | 99,51 | 0,04 | 0,00 |
| | | avg | 1,71 | 99,86 | 0,13 | | 0,68 | 1,16 | 99,84 | 0,08 | 0,08 |
| | | max | 1,72 | 99,99 | 0,52 | | 0,68 | 1,17 | 99,95 | 0,15 | 0,33 |

**Double lookup table method for fast light propagation calculations**

In this test, **100 point light sources at z=-3 mm** are propagated to a plane z=0, sampling distance 1 um, lambda = 635 nm. XY size of the target area varies, see "screen side" column.
Execution time was measured for precise algorithm (no look-up tables, simple propagation kernel), algorithm with waveLUT only and algorithm with both rhoLUT and waveLUT (= double LUT or dLUT for short).
**The rhoLUT uses bilinear interpolation, i.e. it can be small. The implementation of the rhoLUT is not optimized.**
Table shows complete times as well as partial times used for look-up tables generation and propagation itself. Actually, the calculation ran 10 times and the total time is shown here; single calculation is therefore 1/10 of the times below.
The analysis (blue) shows speed-up of the particular method, where speed-up > 1 means that the method is faster than the reference. It also shows how much time was spent on tables generation etc in [%].
The summary (green) shows minimum, maximum and average values of the matching columns above.
For example, wLUT-only method is **1.71×** faster (on average) than precise calculation. The waveLUT generation took **0.13%** of the calculation. After introduction of the rhoLUT, the average speed-up is just **1.00×**. In comparison to waveLUT only, rhoLUT+waveLUT method is **0.58×** faster, i.e. **1/0.58 = 1.7×** slower.

| | precise | waveLUT | | | | rhoLUT | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| screen side [samples] | complete [s] | waveLUT size [samples] | complete [s] | propagation [s] | waveLUT gen. [s] | rhoLUT side size [samples] | waveLUT size [samples] | complete [s] | propagation [s] | rhoLUT gen. [s] | waveLUT gen. [s] |
| 64 | 0,553963 | 1173 | 0,327471 | 0,325708 | 0,00172252 | 10 | 1190 | 0,559951 | 0,558171 | 2,61E-05 | 0,0017252 |
| 128 | 2,21205 | 1543 | 1,29978 | 1,29746 | 0,00228494 | 12 | 1515 | 2,21998 | 2,21771 | 3,87E-05 | 0,00219644 |
| 256 | 8,85363 | 2275 | 5,18283 | 5,17944 | 0,00334695 | 16 | 2193 | 8,86774 | 8,86445 | 7,05E-05 | 0,00317876 |
| 512 | 35,6129 | 3730 | 20,7035 | 20,698 | 0,00544914 | 24 | 3585 | 35,6259 | 35,6206 | 1,13E-04 | 0,00517483 |
| 1024 | 143,302 | 6631 | 82,9022 | 82,8926 | 0,00960563 | 41 | 6499 | 143,209 | 143,2 | 0,000280828 | 0,00939338 |
| 2048 | 564,587 | 12426 | 327,086 | 327,067 | 0,0181021 | 73 | 12239 | 566,008 | 565,989 | 0,00094746 | 0,0177592 |

| screen side [samples] | | | comparison precise / waveLUT (>1 = good) | propagation [%] | waveLUT gen. [%] | | comparison waveLUT / dLUT (>1 = good) | comparison precise / rhoLUT (>1 = good) | propagation [%] | rhoLUT gen. [%] | waveLUT gen. [%] |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 64 | | | 1,69 | 99,46 | 0,53 | | 0,58 | 0,99 | 99,68 | 0,00 | 0,31 |
| 128 | | | 1,70 | 99,82 | 0,18 | | 0,59 | 1,00 | 99,90 | 0,00 | 0,10 |
| 256 | | | 1,71 | 99,93 | 0,06 | | 0,58 | 1,00 | 99,96 | 0,00 | 0,04 |
| 512 | | | 1,72 | 99,97 | 0,03 | | 0,58 | 1,00 | 99,99 | 0,00 | 0,01 |
| 1024 | | | 1,73 | 99,99 | 0,01 | | 0,58 | 1,00 | 99,99 | 0,00 | 0,01 |
| 2048 | | | 1,73 | 99,99 | 0,01 | | 0,58 | 1,00 | 100,00 | 0,00 | 0,00 |
| | | min | 1,69 | 99,46 | 0,01 | | 0,58 | 0,99 | 99,68 | 0,00 | 0,00 |
| | | avg | 1,71 | 99,86 | 0,13 | | 0,58 | 1,00 | 99,92 | 0,00 | 0,08 |
| | | max | 1,73 | 99,99 | 0,53 | | 0,59 | 1,00 | 100,00 | 0,00 | 0,31 |

**Double lookup table method for fast light propagation calculations**

In this test, **100 point light sources at z=-3 mm** are propagated to a plane z=0, sampling distance 1 um, lambda = 635 nm. XY size of the target area varies, see "screen side" column.
Execution time was measured for precise algorithm (no look-up tables, simple propagation kernel), algorithm with waveLUT only and algorithm with both rhoLUT and waveLUT (= double LUT or dLUT for short).
**The rhoLUT uses constant interpolation, i.e. it must be large. The implementation of the rhoLUT is optimized and uses integer arithmetic whenever possible.**
Table shows complete times as well as partial times used for look-up tables generation and propagation itself. Actually, the calculation ran 10 times and the total time is shown here; single calculation is therefore 1/10 of the times below.
The analysis (blue) shows speed-up of the particular method, where speed-up > 1 means that the method is faster than the reference. It also shows how much time was spent on tables generation etc in [%].
The summary (green) shows minimum, maximum and average values of the matching columns above.
For example, wLUT-only method is **1.80×** faster (on average) than precise calculation. The waveLUT generation took **0.14%** of the calculation. After introduction of the rhoLUT, the average speed-up is **2.59×**. In comparison to waveLUT only, rhoLUT+waveLUT method is **1.44×** faster.

| | precise | waveLUT | | | | rhoLUT | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| screen side [samples] | complete [s] | waveLUT size [samples] | complete [s] | propagation [s] | waveLUT gen. [s] | rhoLUT side size [samples] | waveLUT size [samples] | complete [s] | propagation [s] | rhoLUT gen. [s] | waveLUT gen. [s] |
| 64 | 0,554504 | 1173 | 0,312518 | 0,310752 | 0,00172328 | 133 | 1088 | 0,218491 | 0,21614 | 0,000744406 | 0,00157616 |
| 128 | 2,21231 | 1543 | 1,23918 | 1,23688 | 0,0022585 | 165 | 1410 | 0,860027 | 0,855974 | 0,00191446 | 0,00210181 |
| 256 | 8,85652 | 2275 | 4,9303 | 4,92682 | 0,00342204 | 229 | 2093 | 3,41688 | 3,40887 | 0,00490319 | 0,00305846 |
| 512 | 35,6216 | 3730 | 19,6837 | 19,678 | 0,00554875 | 357 | 3509 | 13,6216 | 13,6011 | 0,0153551 | 0,00508479 |
| 1024 | 143,288 | 6631 | 78,7872 | 78,7773 | 0,00993588 | 613 | 6385 | 54,669 | 54,604 | 0,0556113 | 0,0093225 |
| 2048 | 564,855 | 12426 | 315,074 | 315,056 | 0,0181496 | 1125 | 12166 | 218,759 | 218,536 | 0,204876 | 0,0178945 |

| screen side [samples] | | | comparison precise / waveLUT (>1 = good) | propagation [%] | waveLUT gen. [%] | | comparison waveLUT / dLUT (>1 = good) | comparison precise / rhoLUT (>1 = good) | propagation [%] | rhoLUT gen. [%] | waveLUT gen. [%] |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 64 | | | 1,77 | 99,43 | 0,55 | | 1,43 | 2,54 | 98,92 | 0,34 | 0,72 |
| 128 | | | 1,79 | 99,81 | 0,18 | | 1,44 | 2,57 | 99,53 | 0,22 | 0,24 |
| 256 | | | 1,80 | 99,93 | 0,07 | | 1,44 | 2,59 | 99,77 | 0,14 | 0,09 |
| 512 | | | 1,81 | 99,97 | 0,03 | | 1,45 | 2,62 | 99,85 | 0,11 | 0,04 |
| 1024 | | | 1,82 | 99,99 | 0,01 | | 1,44 | 2,62 | 99,88 | 0,10 | 0,02 |
| 2048 | | | 1,79 | 99,99 | 0,01 | | 1,44 | 2,58 | 99,90 | 0,09 | 0,01 |
| | | min | 1,77 | 99,43 | 0,01 | | 1,43 | 2,54 | 98,92 | 0,09 | 0,01 |
| | | avg | 1,80 | 99,86 | 0,14 | | 1,44 | 2,59 | 99,64 | 0,17 | 0,19 |
| | | max | 1,82 | 99,99 | 0,55 | | 1,45 | 2,62 | 99,90 | 0,34 | 0,72 |

**Double lookup table method for fast light propagation calculations**

In this test, the source optical field at z=0 was propagated to the target of the same size at z=3 mm. Size of the source/target varies, see "side size" column. Sampling distance = 1 um, lambda = 635 nm.
Propagation was calculated using filtered Rayleigh-Sommerfeld kernel. Its calculation is very slow for high numerical apertures, i.e. when sizes of the source and target areas are big compared to their distances.
The propagation was calculated precisely (with lots of optimizations in the kernel calculations), using waveLUT and using waveLUT + rhoLUT (= double LUT or dLUT for short). **The rhoLUT uses bilinear interpolation, therefore can be small. The rhoLUT interpolation scheme is not optimized.**
The table below shows average times of calculation; actual times are listed in the sheet **DataA**, there are 10 iterations per every calculation. Besides complete calculation time, times for kernel calculation and convolution (= 3x FFT) are shown. The times include look-up tables generation.
The first analysis (blue) compares a look-up table method to the precise method; naturally, convolution time should be the same so that this column value is approximately one. It is clear that the speed-up is just in the kernel calculation and more complex kernel leads to higher speed-up. The second analysis (green) compares dLUT to waveLUT.
For example, for propagated areas 1024 × 1024 samples, the kernel calculation is **38×** faster with waveLUT, but just **21×** faster with dLUT. DLUT method is in fact **0.54×** faster (= **1/0.54** = **1.85×** slower) than waveLUT method.

| | Precise | | | waveLUT propagation: | | | dLUT (= waveLUT + rhoLUT) propagation: | | |
|---|---|---|---|---|---|---|---|---|---|
| side size [samples] | Complete time [s] | Kernel time [s] | Convolution time [s] | Complete time [s] | Kernel time [s] | Convolution time [s] | Complete time [s] | Kernel time [s] | Convolution time [s] |
| 64 | 0,004714772 | 0,003734365 | 0,00074793 | 0,002446922 | 0,001551988 | 0,000746628 | 0,00379191 | 0,002897091 | 0,000746781 |
| 128 | 0,02088881 | 0,01455173 | 0,005283633 | 0,01163779 | 0,005548179 | 0,005200764 | 0,01630796 | 0,01023375 | 0,0051895 |
| 256 | 0,1453614 | 0,1160581 | 0,02595585 | 0,05206576 | 0,0224063 | 0,0261545 | 0,07013853 | 0,04082331 | 0,02616427 |
| 512 | 0,9926633 | 0,847222 | 0,1329556 | 0,229052 | 0,08418382 | 0,1328215 | 0,3017198 | 0,1568112 | 0,1328727 |
| 1024 | 13,59913 | 12,92086 | 0,6286457 | 1,002117 | 0,3378521 | 0,616487 | 1,29216 | 0,6276428 | 0,6166865 |
| 2048 | 170,4906 | 166,3615 | 3,926716 | 5,391706 | 1,270854 | 3,929897 | 6,541871 | 2,431887 | 3,918442 |
| side size [samples] | | | | comparison precise / wLUT (>1 = good) | comparison precise / wLUT (>1 = good) | comparison precise / wLUT (>1 = good) | comparison precise / rhoLUT (>1 = good) | comparison precise / rhoLUT (>1 = good) | comparison precise / rhoLUT (>1 = good) |
| 64 | | | | 1,93 | 2,41 | 1,00 | 1,24 | 1,29 | 1,00 |
| 128 | | | | 1,79 | 2,62 | 1,02 | 1,28 | 1,42 | 1,02 |
| 256 | | | | 2,79 | 5,18 | 0,99 | 2,07 | 2,84 | 0,99 |
| 512 | | | | 4,33 | 10,06 | 1,00 | 3,29 | 5,40 | 1,00 |
| 1024 | | | | 13,57 | 38,24 | 1,02 | 10,52 | 20,59 | 1,02 |
| 2048 | | | | 31,62 | 130,91 | 1,00 | 26,06 | 68,41 | 1,00 |
| side size [samples] | | | | | | | comparison waveLUT / rhoLUT (>1 = good) | comparison waveLUT / rhoLUT (>1 = good) | comparison waveLUT / rhoLUT (>1 = good) |
| 64 | | | | | | | 0,65 | 0,54 | 1,00 |
| 128 | | | | | | | 0,71 | 0,54 | 1,00 |
| 256 | | | | | | | 0,74 | 0,55 | 1,00 |
| 512 | | | | | | | 0,76 | 0,54 | 1,00 |
| 1024 | | | | | | | 0,78 | 0,54 | 1,00 |
| 2048 | | | | | | | 0,82 | 0,52 | 1,00 |

**Double lookup table method for fast light propagation calculations**

In this test, the source optical field at z=0 was propagated to the target of the same size at z=3 mm. Size of the source/target varies, see "side size" column. Sampling distance = 1 um, lambda = 635 nm.
Propagation was calculated using filtered Rayleigh-Sommerfeld kernel. Its calculation is very slow for high numerical apertures, i.e. when sizes of the source and target areas are big compared to their distances.
The propagation was calculated precisely (with lots of optimizations in the kernel calculations), using waveLUT and using waveLUT + rhoLUT (= double LUT or dLUT for short). **The rhoLUT uses constant interpolation, therefore must be big. The rhoLUT implementation alsu uses integers whenever possible.**
The table below shows average times of calculation; actual times are listed in the sheet **DataB**, there are 10 iterations per every calculation. Besides complete calculation time, times for kernel calculation and convolution (= 3x FFT) are shown. The times include look-up tables generation.
The first analysis (blue) compares a look-up table method to the precise method; naturally, convolution time should be the same so that this column value is approximately one. It is clear that the speed-up is just in the kernel calculation and more complex kernel leads to higher speed-up. The second analysis (green) compares dLUT to waveLUT.
For example, for propagated areas 1024 × 1024 samples, the kernel calculation is **38×** faster with waveLUT, and just **65×** faster with dLUT. DLUT method is in fact **1.72×** faster than waveLUT method.

| | Precise | | | waveLUT propagation: | | | dLUT (= waveLUT + rhoLUT) propagation: | | |
|---|---|---|---|---|---|---|---|---|---|
| side size [samples] | Complete time [s] | Kernel time [s] | Convolution time [s] | Complete time [s] | Kernel time [s] | Convolution time [s] | Complete time [s] | Kernel time [s] | Convolution time [s] |
| 64 | 0,005505154 | 0,004396706 | 0,000805284 | 0,002451942 | 0,00155421 | 0,000747317 | 0,001909287 | 0,001010751 | 0,000748888 |
| 128 | 0,02130932 | 0,01493754 | 0,005303171 | 0,01166047 | 0,005549442 | 0,005210496 | 0,009428014 | 0,003307683 | 0,00521992 |
| 256 | 0,1480798 | 0,1187191 | 0,02602289 | 0,0517506 | 0,02238336 | 0,02622307 | 0,04284076 | 0,01348163 | 0,02619941 |
| 512 | 0,98854 | 0,8436283 | 0,132474 | 0,2288875 | 0,08419285 | 0,1325792 | 0,1936394 | 0,04876961 | 0,1327095 |
| 1024 | 13,44956 | 12,77212 | 0,6276179 | 1,004779 | 0,33784 | 0,6190459 | 0,8624188 | 0,1963881 | 0,6180069 |
| 2048 | 156,4966 | 152,3568 | 3,933948 | 5,386726 | 1,275866 | 3,919253 | 4,855197 | 0,744881 | 3,919511 |
| side size [samples] | | | | comparison precise / wLUT (>1 = good) | comparison precise / wLUT (>1 = good) | comparison precise / wLUT (>1 = good) | comparison precise / rhoLUT (>1 = good) | comparison precise / rhoLUT (>1 = good) | comparison precise / rhoLUT (>1 = good) |
| 64 | | | | 2,25 | 2,83 | 1,08 | 2,88 | 4,35 | 1,08 |
| 128 | | | | 1,83 | 2,69 | 1,02 | 2,26 | 4,52 | 1,02 |
| 256 | | | | 2,86 | 5,30 | 0,99 | 3,46 | 8,81 | 0,99 |
| 512 | | | | 4,32 | 10,02 | 1,00 | 5,11 | 17,30 | 1,00 |
| 1024 | | | | 13,39 | 37,81 | 1,01 | 15,60 | 65,04 | 1,02 |
| 2048 | | | | 29,05 | 119,41 | 1,00 | 32,23 | 204,54 | 1,00 |
| side size [samples] | | | | | | | comparison waveLUT / rhoLUT (>1 = good) | comparison waveLUT / rhoLUT (>1 = good) | comparison waveLUT / rhoLUT (>1 = good) |
| 64 | | | | | | | 1,28 | 1,54 | 1,00 |
| 128 | | | | | | | 1,24 | 1,68 | 1,00 |
| 256 | | | | | | | 1,21 | 1,66 | 1,00 |
| 512 | | | | | | | 1,18 | 1,73 | 1,00 |
| 1024 | | | | | | | 1,17 | 1,72 | 1,00 |
| 2048 | | | | | | | 1,11 | 1,71 | 1,00 |

# Bibliography

[1] Gabor, D. “A New Microscopic Principle”. In: *Nature* 161 (May 1948), pp. 777–778. DOI: 10.1038/161777a0.

[2] Gabor, D. “Microscopy by Reconstructed Wave-Fronts”. In: *Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences* 197.1051 (1949), pp. 454–487. DOI: 10.1098/rspa.1949.0075.

[3] Denisyuk, Y. N. “Photographic Reconstruction of the Optical Properties of an Object in Its Own Scattered Radiation Field”. In: *Soviet Physics Doklady* 7 (Dec. 1962), p. 543.

[4] Leith, E. N. and Upatnieks, J. “Wavefront Reconstruction with Diffused Illumination and Three-Dimensional Objects”. In: *J. Opt. Soc. Am.* 54.11 (Nov. 1964), pp. 1295–1301. DOI: 10.1364/JOSA.54.001295.

[5] Johnston, S. *Holographic Visions: A History of New Science*. Oxford University Press, Oxford, 2006. ISBN: 9780198571223.

[6] Goodman, J. W. *Introduction to Fourier Optics*. 3rd ed. Roberts & Company Publishers, Dec. 2004. ISBN: 0974707724.

[7] Benton, S. A. and Bove, Jr., V. M. *Holographic Imaging*. Wiley, 2008. ISBN: 9780470224120.

[8] Collier, R., Burckhardt, C. and Lin, L. *Optical holography*. Student editions. Academic Press, 1971. ISBN: 9780121810528.

[9] Schnars, U. et al. *Digital Holography and Wavefront Sensing: Principles, Techniques and Applications*. 2nd. Springer Publishing Company, Incorporated, 2014. ISBN: 3662446928, 9783662446928.

[10] Schnars, U. and Jueptner, W. *Digital holography: digital hologram recording, numerical reconstruction, and related techniques*. Springer, 2005. ISBN: 9783540219347.

[11] Yaroslavsky, L. *Digital Holography and Digital Image Processing: Principles, Methods, Algorithms*. Springer US, 2003. ISBN: 9781402076343.

[12] Picart, P. and Li, J.-C. *Digital Holography*. ISTE. Wiley, 2013. ISBN: 9781118563205.

[13] Poon, T.-C. and Liu, J.-P. *Introduction to Modern Digital Holography: With Matlab*. Cambridge University Press, 2014. ISBN: 9781107729117.

[14] Bjelkhagen, H. and Brotherton-Ratcliffe, D. *Ultra-Realistic Imaging: Advanced Techniques in Analogue and Digital Colour Holography*. Taylor & Francis, 2013. ISBN: 9781439827994.

[15] Bjelkhagen, H. I., Lembessis, A. and Sarakinos, A. “Ultra-realistic imaging and OptoClones”. In: *Proc. SPIE*. Vol. 9771. 2016, pp. 977105–977105–8. DOI: 10.1117/12.2207524.

[16] Voelz, D. G. *Computational Fourier Optics: A Matlab Tutorial*. Tutorial texts in optical engineering. SPIE Press, 2011. ISBN: 9780819482044.

[17] Ozaktas, H. M., Kutay, M. A. and Zalevsky, Z. *The Fractional Fourier Transform: With Applications in Optics and Signal Processing*. Wiley Series in Pure and Applied Optics. Wiley, 2001. ISBN: 9780471963462.

[18] Mandel, L. and Wolf, E. *Optical Coherence and Quantum Optics*. Cambridge University Press, 1995. ISBN: 9780521417112.

[19] Lalor, E. “Conditions for the Validity of the Angular Spectrum of Plane Waves”. In: *J. Opt. Soc. Am.* 58.9 (1968), pp. 1235–1237.

[20] Matsushima, K. and Shimobaba, T. “Band-Limited Angular Spectrum Method for Numerical Simulation of Free-Space Propagation in Far and Near Fields”. In: *Opt. Express* 17.22 (Oct. 2009), pp. 19662–19673. DOI: 10.1364/OE.17.019662.

[21] Falaggis, K., Kozacki, T. and Kujawinska, M. “Computation of highly off-axis diffracted fields using the band-limited angular spectrum method with suppressed Gibbs related artifacts”. In: *Appl. Opt.* 52.14 (May 2013), pp. 3288–3297. DOI: 10.1364/AO.52.003288.

[22] Kipman, A. *Meet the award recipients of the first Microsoft HoloLens academic research grants*. 2015. URL: https://blogs.windows.com/devices/2015/11/11/meet-the-award-recipients-of-the-first-microsoft-hololens-academic-research-grants (visited on 27/01/2017).

[23] Maass, U. *Device for displaying moving images in the background of a stage*. US Patent 5,865,519. Feb. 1999. URL: http://www.google.com/patents/US5865519.

[24] *Crypton Future Media – Hatsune Miku – World is Mine*. 2010. URL: https://youtu.be/pEaBqiLeCu0 (visited on 27/01/2017).

[25] Realfiction. *The remarkable dreamoc$^{TM}$ HD3*. 2015. URL: https://www.realfiction.com/products/dreamoc-hd3 (visited on 27/01/2017).

[26] Wikipedia. *Pepper's ghost — Wikipedia, The Free Encyclopedia.* [Online; accessed 2-March-2013]. 2013.

[27] Dyrholm, C. "Collapsible 3D display and a method of assembling said 3D display". Application EP2508933 A1. Oct. 2012.

[28] Jones, A. et al. "Rendering for an Interactive 360 Degree Light Field Display". In: *ACM SIGGRAPH conference proceedings.* San Diego, CA, Aug. 2007.

[29] Lueder, E. *3D Displays.* Wiley Series in Display Technology. Wiley, 2011. ISBN: 9781119963042.

[30] Okoshi, T. *Three-dimensional imaging techniques.* Academic Press, 1976. ISBN: 9780125252508.

[31] Ochiai, Y. et al. "Fairy Lights in Femtoseconds: Aerial and Volumetric Graphics Rendered by Focused Femtosecond Laser Combined with Computational Holographic Fields". In: *ACM Trans. Graph.* 35.2 (Feb. 2016), 17:1–17:14. ISSN: 0730-0301. DOI: `10.1145/2850414`.

[32] Jonsson, R. *Burton rolls out True 3D laser plasma display.* 2011. URL: `http://newatlas.com/burton-true-3d-laser-plasma-display/20499/` (visited on 27/01/2017).

[33] Lobaz, P. "Reference calculation of light propagation between parallel planes of different sizes and sampling rates". In: *Opt. Express* 19.1 (Jan. 2011), pp. 32–39. DOI: `10.1364/OE.19.000032`.

[34] Edee, K., Plumey, J. P. and Chandezon, J. "On the Rayleigh-Fourier method and the Chandezon method: Comparative study". In: *Optics Communications* 286 (Jan. 2013), pp. 34–41. DOI: `10.1016/j.optcom.2012.08.088`.

[35] Lobaz, P. "Memory-efficient reference calculation of light propagation using the convolution method". In: *Opt. Express* 21.3 (Feb. 2013), pp. 2795–2806. DOI: `10.1364/OE.21.002795`.

[36] Lobaz, P. "Discrete calculation of the off-axis angular spectrum based light propagation". In: *Journal of Physics: Conference Series.* Vol. 415. 1. 2013, p. 012040.

[37] Lobaz, P. and Vaněček, P. "Safe range of free space light propagation calculation in single precision". In: *Opt. Express* 23.3 (Feb. 2015), pp. 3260–3269. DOI: `10.1364/OE.23.003260`.

[38] Hanák, I. et al. "Hologram synthesis accelerated in field programmable gate array by partial quadratic interpolation". In: *Optical Engineering* 8.48 (2009), pp. 1–7. ISSN: 0091-3286.

[39] Lobaz, P. “Double lookup table method for fast light propagation calculations”. In: *Proceedings of the 10th International Symposium on Display Holography.* 2015, in press.

[40] Okada, N. et al. “Band-limited double-step Fresnel diffraction and its application to computer-generated holograms”. In: *Opt. Express* 21.7 (Apr. 2013), pp. 9192–9197. DOI: 10.1364/OE.21.009192.

[41] Tommasi, T. and Bianco, B. “Computer-generated holograms of tilted planes by a spatial frequency approach”. In: *J. Opt. Soc. Am. A* 10.2 (Feb. 1993), pp. 299–305. DOI: 10.1364/JOSAA.10.000299.

[42] Leseberg, D. “Computer-generated three-dimensional image holograms”. In: *Appl. Opt.* 31.2 (Jan. 1992), pp. 223–229. DOI: 10.1364/AO.31.000223.

[43] Matsushima, K. “Formulation of the rotational transformation of wave fields and their application to digital holography”. In: *Appl. Opt.* 47.19 (July 2008), pp. D110–D116. DOI: 10.1364/AO.47.00D110.

[44] Onural, L. “Exact solution for scalar diffraction between tilted and translated planes using impulse functions over a surface”. In: *J. Opt. Soc. Am. A* 28.3 (Mar. 2011), pp. 290–295. DOI: 10.1364/JOSAA.28.000290.