universität wien

MASTERARBEIT / MASTER'S THESIS

Titel der Masterarbeit / Title of the Master's Thesis

„Simulation of Electron-Light Interactions“

verfasst von / submitted by

Matthias Schneller BSc

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien, 2022 / Vienna, 2022

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	UA 066 876
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Masterstudium Physik
Betreut von / Supervisor:	Assoz. Prof. Dipl.-Ing. Dr. Thomas Juffmann

Acknowledgements

The creation of a thesis and the successful conclusion of a university study is a task that can not be done completely alone. Thus, I want to mention a few people here that have been very supportive to me in the last few years.

From a scientific point of view, I want to thank Thomas Juffmann for the great supervision as well as the possibility to be part of such an exciting project in the last few years. I also want to thank Stefan Nimmrichter for his patient explanations of everything concerning theory. Additionally, I want to thank my office mates Marius and Lucas for their support and readiness to help whenever I had a question.

But a successful completion of a study is not possible without the support of people outside the field of studies. There I want to thank my family for their continued support over the last few years and for the possibility to finish my degree. I also want to thank Felix for his support, great discussions and being a very good friend, Lisa for her council and nice evenings together, and I also want to thank Andrea for her proofreading of this thesis, fruitful discussions and her ability to always keep me on the ground.

Abstract

Since the discovery of electrons an interest exists in how electrons and light fields interact with each other, starting with the interactions like the photoelectric effect in materials or the interaction of free electrons with photons as described by the Kapitza-Dirac effect in 1933. This thesis gives a quick summary of the different possible interactions for free space electrons and photons, describes and derives the effect on electrons that are co-/counter propagating with the photons and shows possible use cases of this effect. In this thesis we discuss the possibility of creating electron lenses using a programmable light source, which will not be subject to Scherzer's theorem, is discussed and possible spatial light distributions that act as a convex and concave lenses are given. The given light intensity distributions are then tested using a Monte Carlo based ray tracing model and it could be verified that lensing is happening. With a quantum mechanical wave simulation, interference effects for certain laser intensity distributions are shown. Additionally, the advantages and drawbacks of the wave simulation are discussed and a guide is given on how to successfully conduct calculations on transfer function based solutions to the Helmholtz equation.

Kurzfassung

Seit der Entdeckung der Elektronen gibt es Interesse daran, Elektronen mit Licht zu beeinflussen. Die Manipulation wurde zuerst in Materialien erreicht, zum Beispiel beim photoelektrischen Effekt, jedoch wurde auch bald von Kapitza und Dirac (1933) eine Interaktion im Vakuum vorhergesagt. Diese Masterarbeit gibt eine kurze Zusammenfassung verschiedener Effekte welche eine Elektronen-Photonen Interaktion im Vakuum erlauben. Zusätzlich wird eine Herleitung formuliert für den Fall, dass die Elektronen und ein Laser Puls sich (anti-)parallel zueinander bewegen und mögliche Anwendungsgebiete dieses Effektes werden besprochen. Einer dieser Anwendungsmöglichkeiten ist die Erzeugung von Laser-Linsen für Elektronenstrahlen, welche nicht Scherzer's Theorem unterworfen sind und daher in der Theorie aberrationsfrei gestaltet werden können. Zudem wird die Möglichkeit beschrieben, nicht nur konvexe sondern auch konkave Elektronen-Linsen zu erzeugen. Die Anwendung dieser Linsen wird durch eine Monte Carlo basierte, sogenannte "Ray Tracing" simulation getestet und mit theoretischen perfekten Linsen verglichen. Mit einer, auf quantenmechanischen Grundsätzen basierenden, Wellenoptik simulation werden potentielle Anwendungsgebiete im Bereich der Materienwellenoptik simuliert.

Contents

1. Introduction

As the title "Simulation of Electron Light Interaction" already indicates, this thesis discusses the simulation of a new way to manipulate electrons in an electron microscope.

The idea to build an electron microscope came after the first appearance of quantum theory. The foundations were laid by the young Louis de Broglie, who, in his dissertation from the year 1925, combined the theory of the light quanta with special relativity and postulated the relation $h\nu_0 = m_0c^2$ for atoms [dB]. This relation can be rewritten to yield the de Broglie wavelength

$$\lambda = \frac{h}{p},$$

where h is the Planck constant and p is the momentum of the particle. This relation treats the atoms that were always thought of as particles as waves and opened the door to a whole new interpretation of physics. Not only that, it created the possibility to increase the resolution of microscopes drastically as the resolution of a microscope is proportional to the wavelength of the probing beam. Following this argument, the first electron microscope was built in 1931 by Ernst Ruska, with a magnification of only 17.4× [Rus]. Technical issues were overcome with time and the resolution of electron microscopes has beaten the resolution of optical microscopes by orders of magnitudes.

One big challenge in increasing the resolution of an electron microscope was the problem of spherical aberrations in the electron lenses. Otto Scherzer calculated in 1936 that it is not possible for a conventional electron lens to be free of spherical aberrations [Sch]. In his calculations, Scherzer assumed that there are no space charges on the optical axis, that the lens is cylindrically symmetric, and that there is no temporal dependency of the electric and magnetic fields. These assumptions left the loophole, that if it is possible to create a lens that does not fall under Scherzer's assumptions, it might be possible to construct better electron lenses. Examples of aberration correction are a complex sextupole lens setup developed in 1995 by Max Haider and a quadrupole with a sextuplet of octapoles in the setup of Ondrej Krivanek in 1997 [Haw]. Another idea to solve the issue of aberrations was proposed by Shiloh et al. where the aberration correction is realized using a sculpted thin film that is positioned in the electron beam [SRL$^+$].

These approaches demonstrated that it is possible to create a very good convex electron lens. The approach using sculpted thin films was also used to manipulate the electrons in more general ways [RSL$^+$] [SLLA] [UT] [MAA$^+$]. This thin film approach has the advantage that the shaping is arbitrary, as the thickness of the material film will directly influence the phase shift applied to the electron beam. A disadvantage of the scheme is that it is not programmable, so for every pattern a new film is needed.

Our group set out to realize a programmable wavefront shaping approach based on stimulated elastic Compton scattering, as was later also proposed by Javier García de Abajo and Andrea Konečná [dAK]. The idea is to shape a light field using a spatial light modulator (SLM), and to imprint a phase pattern onto the electron beam making use of the stimulated elastic Compton scattering [MWS$^+$]. This thesis describes the derivation of the phase shifts caused by stimulated Compton scattering, approaches to implement the electron-light interaction into numerical simulations and possible applications.

As the most fundamental point in [MWS$^+$] is the interaction between electrons and photons, we will dedicate the next sections to the description of the electron-light interaction and its historical development.

1.1. Electron - Light Interaction History

Before J. J. Thomson discovered electrons in 1897, they were known as cathode rays. Prior to that, it was already known since 1858 that they interact with the magnetic field [Plu] as well as, since 1890, that they interact with the electric field [Bak]. As Maxwell's equations had already been known, it was clear that there is the possibility of an interaction between light waves and electrons.

With the advent of quantum theory in the early 1900s, new ideas emerged that would describe the interaction between an electron and a photon. In the following sections we will show and describe a few of these interactions in detail. Starting with the simple picture of one electron and one photon interaction, we will show why this is not possible in general. Following up on this we will discuss Compton scattering and the Kapitza-Dirac effect, as well as the electron-light interaction near a surface. This chapter will be closed by a detailed description of the programmable electron wavefront shaping that was described prior.

1.1.1. Elastic Free Electron-Photon Interaction

We start by asking whether it is possible for a free-space electron to first absorb a photon and then spontaneously re-emit a photon elastically. The electron travels along the positive x-direction and interacts with a photon from an arbitrary direction. For a relativistic electron, we have the dispersion relation of

$$E_e = \sqrt{c^2 p_e^2 + m_e^2 c^4}, \tag{1.1}$$

where c is the speed of light, m_e is the electron mass and p_e is the absolute value of the electron momentum.

The energy and momentum of the photon are defined as

$$\begin{aligned} E_p &= \hbar\omega_p \qquad \text{and} \\ \boldsymbol{p}_p &= \hbar k_p \begin{pmatrix} \cos\theta \\ \sin\theta \end{pmatrix}, \end{aligned}$$

with ω_p being the angular frequency of the light, k_p being the wave number of the light, and θ denoting the propagation direction of the photon in the x-y plane.

After photon absorption the electron will have the energy $E = E_e + E_p$ and the momentum $\boldsymbol{p} = \boldsymbol{p}_e + \boldsymbol{p}_p$, which both obey the dispersion relation in equation (1.1). Calculating the electron dispersion after the interaction gives

$$\begin{aligned} E_e + \hbar\omega_p &= \sqrt{c^2(p_e + \hbar k_p \cos\theta)^2 + c^2\hbar^2 k_p^2 \sin^2\theta + m_e^2 c^4} \\ (E_e + \hbar\omega_p)^2 &= c^2 p_e^2 + 2c^2 p_e \hbar k_p \cos\theta + c^2\hbar^2 k_p^2 \underbrace{(\cos^2\theta + \sin^2\theta)}_{=1} + m_e^2 c^4 \\ \cos\theta &= \frac{\cancel{E_e^2} + 2E_e\hbar\omega_p + \hbar^2\omega_p^2 - \cancel{c^2 p_e^2} - c^2\hbar^2 k_p^2 - \cancel{m_e^2 c^4}}{2c^2 p_e \hbar k_p}. \end{aligned} \tag{1.2}$$

Further simplifying equation (1.2) using the dispersion relation of light $\omega_p = ck_p$ will result in

$$\cos\theta = \frac{\sqrt{p_e^2 + m_e^2 c^2}}{p_e} > 1, \tag{1.3}$$

which is a contradiction. Equation (1.3) states that it is not possible, should energy and momentum be conserved, that a free electron absorbs and then spontaneously re-emits a photon elastically.

1.1.2. Compton Scattering

We now consider inelastic Compton scattering of photons by free-space electrons. Compton scattering was first derived in 1923 by A. H. Compton [Com] in the electron rest frame.

The incoming photon can be characterized with the energy $E_\gamma = \hbar\omega_\gamma$ and the momentum $\boldsymbol{p}_\gamma$, while the scattered photon has the energy $E_{\gamma'} = \hbar\omega_{\gamma'}$ and the momentum $\boldsymbol{p}_{\gamma'}$. The electrons are defined similarly with the incoming ($E_e = m_e c^2$, $|\boldsymbol{p}_e| = 0$) and scattered ($E_{e'} = \sqrt{p_{e'}^2 c^2 + m_e^2 c^4}$, $\boldsymbol{p}_{e'}$) energies and momenta respectively.

Because of the conservation of energy the incoming and the scattered energies and momenta must be conserved:

$$\begin{aligned} E_\gamma + E_e &= E_{e'} + E_{\gamma'} \\ \boldsymbol{p}_\gamma &= \boldsymbol{p}_{e'} + \boldsymbol{p}_{\gamma'} \end{aligned}$$

Inserting the energy into the conservation law gives

$$\begin{aligned} &\hbar\omega + E_e = \hbar\omega' + \sqrt{p_{e'}^2 c^2 + m_e^2 c^4} \\ \implies \quad &c^2 p_{e'}^2 = \left(\hbar\omega - \hbar\omega' + E_e\right)^2 - m_e^2 c^4, \end{aligned} \tag{1.4}$$

where the only unknown is the $p_{e'}^2$ parameter. From the conservation of momentum, the missing $p_{e'}^2$ factor can be calculated to be

$$\begin{aligned} p_{e'}^2 =& (\boldsymbol{p}_\gamma - \boldsymbol{p}_{\gamma'}) \cdot (\boldsymbol{p}_\gamma - \boldsymbol{p}_{\gamma'}) \\ =& p_\gamma^2 + p_{\gamma'}^2 - 2\boldsymbol{p}_\gamma \cdot \boldsymbol{p}_{\gamma'} \\ =& p_\gamma^2 + p_{\gamma'}^2 - 2p_\gamma p_{\gamma'} \cos\theta, \end{aligned} \tag{1.5}$$

where θ is the scattering angle between the incoming photon and the scattered photon. Now $p_{e'}^2$ is inserted into equation (1.4) and with $E_\gamma = \hbar\omega_\gamma = cp_\gamma$ we get

$$\begin{aligned} & \hbar^2\omega_\gamma^2 + \hbar^2\omega_{\gamma'}^2 - 2\hbar^2\omega_\gamma\omega_{\gamma'}\cos\theta = \big(\hbar\omega - \hbar\omega_{\gamma'} + E_e\big)^2 - m_e^2c^4 \\ \Longrightarrow \quad & E_e\big(\omega - \omega_{\gamma'}\big) = \hbar\omega'\omega\big(1 + \cos\theta\big). \end{aligned}$$

Further reshuffling and replacing the angular frequency with the wavelength using the relation $\omega = 2\pi c/\lambda$ finally gives

$$\lambda_{\gamma'} - \lambda_\gamma = \frac{h}{m_e c}\big(1 + \cos\theta\big),$$

which is the usual representation of the Compton scattering.

1.1.3. Kapitza - Dirac Effect

We now consider photon absorption followed by stimulated emission of a photon of the same energy. This is possible at high light intensities, when both absorption and stimulated emission happen within a short time, such that energy-time uncertainty makes up for the energy and momentum conservation problems encountered in section 1.1.1 [Bat].

This was first proposed by KD in 1933 by Kapitza and Dirac and is now known as the Kapitza-Dirac (KD) effect [KD]. In their paper, they propose the deflection of electrons at a standing light wave, which can be achieved by reflecting a light wave from a mirror. This standing wave can also be viewed as two waves that are counter-propagating. According to Kapitza and Dirac, these counter-propagating waves will then give rise to so-called stimulated Compton scattering, which describes the process, where one photon is absorbed from one beam and the emission is stimulated by the other beam. They thus predict that the momentum transfer from the photon to the electron must be in multiples of $\pm 2\hbar k$, where $\hbar k$ is the photon momentum.

A classical description of this phenomenon is given by H. Batelaan in his review paper [Bat] using the ponderomotive potential. A standing wave along the z-axis can be defined as the vector potential $\boldsymbol{A} = A_0\hat{\boldsymbol{e}}_z \cos(kx)\sin(\omega t)$ with the wave-number k, the angular frequency ω and $\hat{\boldsymbol{e}}_z$ being the unit vector in z-direction. Assuming the gauge $\phi = 0$ the electric field can be written, in SI units, as

$$\begin{aligned} & \boldsymbol{E} = -\frac{\partial \boldsymbol{A}}{\partial t} \\ \Longrightarrow \quad & \boldsymbol{E} = -A_0\omega\hat{\boldsymbol{e}}_z \cos(kx)\cos(\omega t). \end{aligned}$$

The magnetic field can also be calculated from the magnetic vector potential using

$$\begin{aligned} & \boldsymbol{B} = \nabla \times \boldsymbol{A} \\ \implies \quad & \boldsymbol{B} = A_0 k \hat{\boldsymbol{e}}_y \sin(kx) \sin(\omega t). \end{aligned}$$

Comparing the two definitions of the electric field and the magnetic field, we will see that the electric and magnetic fields are oscillating $\pi/2$ out of phase. An electron that is placed in this oscillating field is affected by the electric field and will start to move according to the Lorentz force. If we would only consider the electric field, this force would average to zero over time, as the electric field changes signs during its oscillation. But due to the effect of the magnetic field, which couples to the electron when it starts to move, and due to the phase difference between the magnetic and electric field, the force stops to average to zero [Bat]. The effective ponderomotive potential can then be written as

$$V_P = \frac{e^2 A_0^2}{4 m_e} \cos^2(kx).$$

After deriving the ponderomotive potential we want to analyze the possible implications of such a potential. To achieve this we have a look at a Schrödinger equation with a periodic potential

$$i\hbar \frac{\partial}{\partial t}\psi = -\frac{\hbar^2}{2m_e}\frac{\partial^2}{\partial x^2}\psi + V_0 \cos^2(kx)\psi,$$

where V_0 is the potential amplitude. As an ansatz for the solution we choose a set of plane waves with multiples of the photon momentum $\hbar k$ and an amplitude c_n, where c_n^2 that describes the probability of the electron having such a momentum [Bat]

$$\psi(x,t) = \sum_n c_n(t) e^{inkx}.$$

Applying the trial wave function to each of the terms of our Schördinger equation gives us the time derivative

$$i\hbar \frac{\partial}{\partial t}\psi = \sum_n \frac{\mathrm{d}c_n(t)}{\mathrm{d}t} e^{inkx},$$

the kinetic term

$$-\frac{\hbar^2}{2m_e}\frac{\partial^2}{\partial x^2}\psi = \frac{\hbar^2 k^2}{2m_e}\sum_n n^2 c_n(t) e^{inkx}$$

and the potential term

$$\begin{aligned} V_0 \cos^2(kx)\psi &= \frac{V_0}{4}\left(e^{ikx} + e^{-ikx}\right)^2 \sum_n c_n(t) e^{inkx} \\ &= \frac{V_0}{4}\left(\sum_n c_n(t) e^{i(n+2)kx} + 2\sum_n c_n(t) e^{inkx} + \sum_n c_n(t) e^{i(n-2)kx}\right) \\ &= \frac{V_0}{4}\sum_n \left(2c_n(t) + c_{n-2}(t) + c_{n+2}(t)\right) e^{inkx}. \end{aligned}$$

Combining all three terms together yields differential equations for the amplitudes c_n [Bat]

$$i\frac{\mathrm{d}c_n}{\mathrm{d}t} = \left(\frac{\hbar k^2}{2m_e}n^2 + \frac{V_0}{2\hbar}\right)c_n + \frac{V_0}{4\hbar}\big(c_{n-2} + c_{n+2}\big). \tag{1.6}$$

In equation (1.6) it can clearly be seen that the electrons are only allowed to have a momentum transfer in multiples of $2\hbar k$, which is exactly what Kapitza and Dirac described in their paper from 1933 [KD]. For the solution of equation (1.6), it is beneficial to look at two limiting cases.

Diffraction regime

When the kinetic energy of the electron is much smaller than the potential energy $\hbar k^2 n^2/(2m) \ll V_0/(2\hbar)$, the wave-function will be restricted to a tiny area, which will lead to huge momentum uncertainty [Bat]. This solution is called diffraction regime as the electrons can diffract in many orders of $\pm 2\hbar k$. The solution to equation (1.6) then can be written as

$$d_m = i^m \exp\left(-\frac{i}{\hbar}V_0 t\right) J_m(V_0 t/\hbar),$$

where $d_m = c_{n/2}$ with $m = 0, \pm 1, \pm 2, \ldots$ and J_m being the Bessel functions [Bat]. The absolute square of $|d_m|^2$ describes the probability that an electron can be found in the m-th diffraction order.

Bragg regime

In the other extreme, which means $\hbar k^2 n^2/(2m) \gg V_0/(2\hbar)$, the $V_0/(2\hbar)$ term vanishes. In this so called Bragg regime, the electron is barely restricted in space which in turn allows only for a small momentum uncertainty. The solutions for the Bragg regime are [Bat]

$$c_1 = \exp\left(-i\frac{\hbar k^2}{2m}t\right)\cos\big(V_0 t/4\hbar\big)$$
$$c_{-1} = -i\exp\left(-i\frac{\hbar k^2}{2m}t\right)\sin\big(V_0 t/4\hbar\big).$$

Here, only two solutions exist and the probability of finding an electron in each of those solutions is oscillating over time.

The experimental verification of the Kapitza-Dirac effect was achieved by Freimund et. al. in 2001 [FAB] where the diffraction regime was oserved. These limiting cases show that a standing wave of light can act as a thin (diffraction regime) or thick (Bragg regime) potential grating for electrons. The goal of this thesis is to calculate ponderomotive potentials of arbitrarily shaped light fields. The respective theory will be given in chapter 2. The experimental setup used for demonstrating the effect of ponderomotive potentials is briefly introduced in section 1.2.

1.1.4. Bound Electron Photon Interaction

When the electrons that interact with light are close to a material surface, it is possible that they absorb single photons, even though the calculations in section 1.1.1 forbid that at first glance. The difference to section 1.1.1 is that the electron can couple to a plasmon in a nearby material. This coupling means that the electron can transfer momentum and energy to and from the material, which opens more possibilities for momentum and energy conservation momentum conservation from section 1.1.1.

One technique that is based on this form of interaction is the so-called Photon-induced near-field electron microscopy (PINEM) which was developed in the group of A. Zewail in 2009 [BFZ]. In the PINEM setup, the optical near field of a near field of a e.g. carbon nanotube [BFZ], is probed with the help of electrons, wherein an electron energy loss spectrometer the different absorption and emitting energies can be seen.

Similarly, an electron bound to an atom can absorb (emit) the energy of a photon, when (de)excited to higher (lower) energy state. Again, it is the presence of a third interaction partner, namely the atom, that facilitates energy and momentum conservation.

1.2. Transverse Electron Beam Shaping

As mentioned at the beginning of this chapter, our group realized almost arbitrary deflection of an electron beam using ponderomotive potentials [MWS$^+$]. The experimental setup can be seen in figure 1.1.

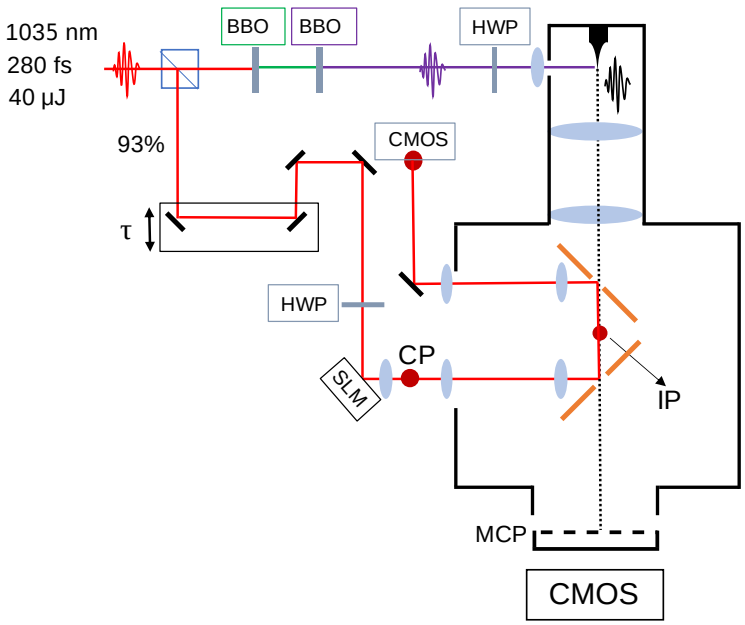


Figure 1.1.: This figure is taken from [MWS$^+$] and shows the laser and electron setup that was used to conduct the experiment. See text for details.

Because of the high laser intensities needed to achieve a relevant phase shift, a femtosecond laser with a pulse energy of up to 40μJ and a wavelength of 1035nm was used. In the experimental setup in 1.1 the input laser beam can be seen on the left-hand side and the scanning electron microscope column on the right-hand side. The laser beam enters a 93/7 beam-splitter and the less intense part traverses through two frequency doubling barium borate (BBO) crystals. The laser beam exits the second BBO with a wavelength of 258nm and thus in the ultra-violet (UV) frequency range. This UV laser beam is then used to trigger the electron gun to emit an electron pulse [MWS$^+$].

The 93% beam from the first beam-splitter enters a delay stage which is used to ensure that the electron and laser pulse are interacting at the focal point of the laser pulse. After the delay stage and a half-wave plate (HWP), the light field hits a spatial light modulator (SLM) which will imprint a programmable phase pattern onto the laser beam.

The phase pattern is chosen such that the desired light intensity pattern is realized in an interaction plane (IP). It is in this plane that the electron pulse and the light pulse overlap in space and time, such that a phase pattern can be imprinted onto the electron beam. The light pulse is then out-coupled from the vacuum chamber, where a CMOS camera allows for further analysis. To discuss the further propagation of the electron beam, we look at the zoomed-in sketch in figure 1.2. We see that the electron beam is focused to a point a small distance from the interaction plane and afterwards hits a multi-channel plate detector. The electron beam thus travels along the positive z-axis and the photon beam along the negative z-axis.

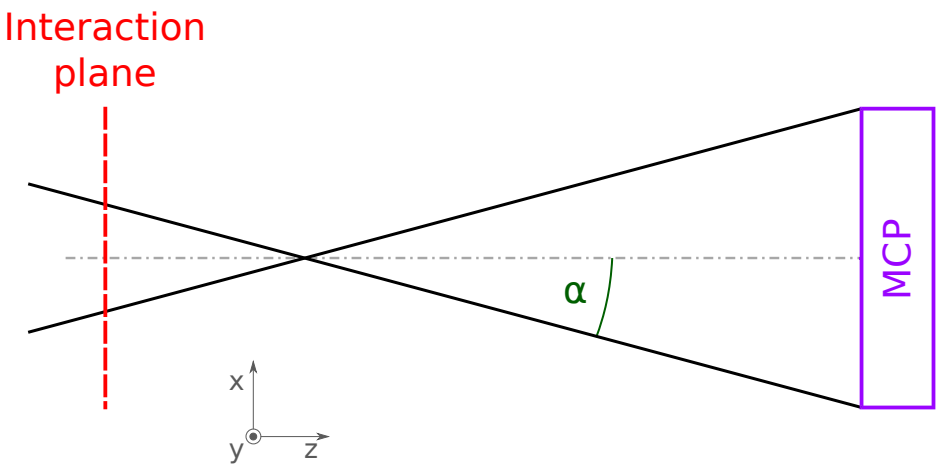


Figure 1.2.: The interaction plane indicates the position where the laser pulse and the electron pulse meet each other. The black lines with the angle α describe the electron cone and the divergence angle. The violet "MCP" description is symbolizing a multi-channel plate that is used to detect the electron distributions.

After the electron-light interaction takes place, the electrons traverse the cross-over point and propagate freely for around 50cm before they interact with the multi-channel plate which will detect the electron distribution.

With a phase retrieval algorithm, more accurately a modified Gerchberg-Saxton algorithm, it is now possible to calculate the phase imprint that is needed at the SLM plane to achieve a desired intensity distribution in the interaction plane [MWS$^+$].

With the described setup, it was possible to show lensing using a convex and even a concave laser lens [MWS$^+$]. Additionally, it was possible to show that the intensity distribution of electrons can be formed nearly arbitrarily. In figure 1.3 we can see the electron deflection for different aberrations in the focal point of the laser. The electron intensity distribution in plot (a) is from a laser with vertical trefoil, in (b) coma and in (c) astigmatism. Figure 1.3 (d) shows, that with the application of the before-mentioned modified Gerchberg-Saxton algorithm, the electron beam could be shaped in the form of a smiley [MWS$^+$].

Following up on this experimental data, in chapter 2 we will discuss the theory behind the electron-photon interaction and give an equation that describes the imprinted phase shift. Additionally, some thoughts are shared on the creation of electron lenses using light, first of all, the creation of the required phase shifts, and second the quality of those lenses and possible aberrations.

In chapter 3 a comprehensive classical treatment of the interaction is provided, with an

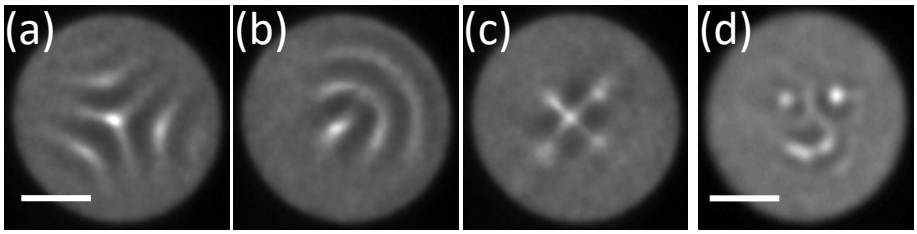


Figure 1.3.: This figure shows micrographs from [MWS$^+$], where the impact of the laser beam onto the electron beam can be observed. The images (a) to (c) (scale bar: 1.1mm) show different aberrated laser imprints, namely for (a) vertical trefoil, for (b) coma and for (c) astigmatism. In figure (d) a smiley can be seen with a scale bar of 1.2mm

additional framework of how to simulate the electron-light interaction in a purely classical way. Details of the implementation and used software are discussed and the chapter closes with an example and motivation for why one would use a program based on classical physics to simulate an electron microscope.

Chapter 4 describes a different simulation approach, namely a purely quantum mechanical description of the electron and matter-wave optics calculations. The simulation code is provided and discussed and possible use cases of the simulation are given with a comparison to the classical calculation.

2. Theory

In this chapter, we will describe the ponderomotive interaction. Starting from the Dirac equation and making use of the paraxial approximation, we derive an equation that describes the phase imprint that a laser pulse will add to an electron wave function. Additionally, the regime in which this equation is valid will be discussed.

We will see that the derived phase shift of the electrons is directly proportional to the transverse intensity distribution of the laser beam, enabling the imprint of arbitrary phase patterns onto the electron beam. Following up on this, the second part of the theoretical chapter discusses different ways of creating a concave or convex laser-based electron lens. Different properties of each of the lens generating intensity distributions are discussed as well as their advantages and caveats.

2.1. Ponderomotive Interaction

In this section, the interaction of an electron and a light field is derived. Other than in [Bat] and in [ACS$^+$], the ponderomotive potential will not be calculated for a standing light wave perpendicular to the electron trajectory, but it will be calculated for co- and counter propagation of the electron and laser beam. For a continuous wave laser, this has been done [dAK]. In this thesis, however, the calculation will be conducted for a pulsed laser field, analogous to [MWS$^+$].

We consider and electron traveling along the z-axis with the electron velocity $\boldsymbol{v} = v\hat{\boldsymbol{e}}_z$. As in a typical electron microscope the acceleration voltages are on the order of several tens of keV, so we treat the electrons relativistically. The relativistic energy is given by $E_e = \gamma m_e c^2$ with $\gamma = 1/\sqrt{1 - v^2/c^2}$ being the relativistic factor. With these preliminary steps out of the way, we can start with defining the electron wave function as

$$\psi(\boldsymbol{r}, t) = \psi_\perp(\boldsymbol{r}, t) \exp\left\{\frac{i}{\hbar}\gamma m_e(vz - c^2 t)\right\}.$$

Here we have separated the wave function into a slowly varying transverse amplitude and phase distribution $\psi_\perp(\boldsymbol{r}, t)$ and a rapidly oscillating phase term representing a plane wave component in z-direction ([dAK], [MWS$^+$]).

If we now want to calculate the time evolution of our electron wave function, the Dirac equation is needed

$$\left\{\beta_d m_e c^2 + c\boldsymbol{\alpha}_d \cdot [\hat{\boldsymbol{p}} + e\boldsymbol{A}(\boldsymbol{r}, t)] - e\phi(\boldsymbol{r}, t)\right\}\boldsymbol{\Psi}(\boldsymbol{r}, t) = i\hbar\frac{\partial \boldsymbol{\Psi}(\boldsymbol{r}, t)}{\partial t}, \tag{2.1}$$

where $\boldsymbol{A}$ and ϕ are the electro-magnetic potentials, e is the elemental charge, $\hat{\boldsymbol{p}}$ is the momentum operator and the Dirac matrices

$$\boldsymbol{\alpha}_d = \begin{pmatrix} 0 & \boldsymbol{\sigma} \\ \boldsymbol{\sigma} & 0 \end{pmatrix},$$
$$\beta_d = \begin{pmatrix} \mathcal{I}_2 & 0 \\ 0 & -\mathcal{I}_2 \end{pmatrix}.$$

In the definition of $\boldsymbol{\alpha}_d$ and β_d, $\mathcal{I}_2$ is the 2×2 identity matrix and $\boldsymbol{\sigma}$ the Pauli matrices.

Within the non-recoil approximation, which assumes that the energy change that comes from the light field is negligible compared to the electron energy E_e, the Dirac equation (2.1) can be rewritten to

$$(\partial_t + \boldsymbol{v} \cdot \nabla)\psi_\perp(\boldsymbol{r}, t) = -\frac{i}{\hbar} U(\boldsymbol{r}, t)\psi_\perp(\boldsymbol{r}, t), \tag{2.2}$$

which is an effective Schrödinger equation [dAK]. The $U(\boldsymbol{r}, t)$ in equation (2.2) is the effective interaction potential between the electrons and photons, which has the form of

$$U(\boldsymbol{r}, t) = \frac{e^2}{2m_e\gamma}\left[A_x^2(\boldsymbol{r}, t) + A_y^2(\boldsymbol{r}, t) + \frac{1}{\gamma^2}A_z^2(\boldsymbol{r}, t)\right] + e\boldsymbol{v} \cdot \boldsymbol{A}(\boldsymbol{r}, t) - e\phi(\boldsymbol{r}, t). \tag{2.3}$$

Using gauge freedom, the scalar potential can be set to zero $\phi = 0$.

In the following derivation we assume that the light intensity distribution does not change during the interaction, i.e. the interaction time is very short. The analytical solution of equation (2.2) can be obtained by simply integrating equation (2.2) between the times t_0 and t_1 [MWS$^+$]. This gives, as a solution for the transverse state propagation,

$$\psi_\perp(\boldsymbol{r} + \boldsymbol{v}t_1, t_1) = \psi_\perp(\boldsymbol{r} + \boldsymbol{v}t_0, t_0) \exp\left[-\frac{i}{\hbar}\int_{t_0}^{t_1} \mathrm{d}t' U(\boldsymbol{r} + \boldsymbol{v}t', t')\right] \tag{2.4}$$

for a change of state from time t_0 to time t_1. If now the time passed is chosen to be very large (larger than the electron - light interaction), i.e. $t_0 = -\infty$ and $t_1 = \infty$, equation (2.4) can be interpreted as a scattering transformation at an infinitesimal scatterer at time t [MWS$^+$]. This changes equation (2.4) to the simpler form of

$$\psi_\perp^{(\mathrm{out})}(\boldsymbol{r}, t) = \psi_\perp^{(\mathrm{in})}(\boldsymbol{r}, t) \exp\left[i \underbrace{\left(-\frac{1}{\hbar}\int_{-\infty}^{\infty} \mathrm{d}t' U(\boldsymbol{r} + \boldsymbol{v}t', t')\right)}_{\varphi_p(\boldsymbol{r})}\right] \tag{2.5}$$

in which φ_p is the ponderomotive phase shift induced by a light pulse.

Now we have a look at the $e\boldsymbol{v} \cdot \boldsymbol{A}$ term of the interaction potential (2.3). The effect of the $e\boldsymbol{v} \cdot \boldsymbol{A}$ term on the ponderomotive phase shift can be calculated by integrating the term according to equation (2.5).

Therefore a general plane wave

$$\boldsymbol{E}_{\boldsymbol{k}\sigma}(\boldsymbol{r}, t) = \mathfrak{Re}\left\{\hat{\boldsymbol{e}}_{\boldsymbol{k}\sigma} e^{i(\boldsymbol{k}\cdot\boldsymbol{r} - \omega t)}\right\} \tag{2.6}$$

with the polarization vector $\hat{\boldsymbol{e}}_{\boldsymbol{k}\sigma}$ is transformed into the vector potential

$$\begin{aligned}\boldsymbol{A}_{\boldsymbol{k}\sigma}(\boldsymbol{r},t) &= -\int \mathrm{d}t \boldsymbol{E}_{\boldsymbol{k}\sigma}(\boldsymbol{r},t) \\ &= -\mathfrak{Re}\{\hat{\boldsymbol{e}}_{\boldsymbol{k}\sigma} e^{i\boldsymbol{k}\cdot\boldsymbol{r}}\} \int \mathrm{d}t \cos(\omega t) \\ &= -\mathfrak{Re}\{\hat{\boldsymbol{e}}_{\boldsymbol{k}\sigma} e^{i\boldsymbol{k}\cdot\boldsymbol{r}}\} \frac{\sin(\omega t)}{\omega} \\ &= \frac{1}{\omega}\mathfrak{Im}\{\hat{\boldsymbol{e}}_{\boldsymbol{k}\sigma} e^{i(\boldsymbol{k}\cdot\boldsymbol{r}-\omega t)}\}\end{aligned}$$

using $-\partial_t \boldsymbol{A}(\boldsymbol{r},t) = \boldsymbol{E}(\boldsymbol{r},t)$ (gauge: $\phi = 0$). These plane wave components can then be put into superposition to construct arbitrary electric fields that are far away from any source [dAK].

In order to calculate the phase shift of each component, they have to be integrated according to equation (2.5). Substituting $\boldsymbol{r} = \boldsymbol{v}t$ and computing these calculations results in

$$-\frac{1}{\hbar}\int_{-\infty}^{\infty} \mathrm{d}t\, e\boldsymbol{v}\cdot\boldsymbol{A} = -\frac{1}{\hbar}\sum_{j=1}^{3} v_j \hat{e}_{k_j\sigma} \int_{-\infty}^{\infty} \mathrm{d}t e^{\pm i(v_j k_j - \omega)t'} = -\frac{1}{\hbar}\sum_{j=1}^{3} v_j \hat{e}_{k_j\sigma} 2\pi\delta(v_j k_j - \omega)$$

and thus can be neglected because the δ functions are only non-zero if one of the electron velocity components is equal to the speed of light [dAK].

For calculating the $\boldsymbol{A}^2$ term specific constraints are applied to the electric field. These are that the field is traveling along the $\pm z$ direction, is paraxial, is pulsed (envelope function condition: $u(t \to \pm\infty) = 0$), normalizable ($\int |\boldsymbol{E}| \mathrm{d}x\mathrm{d}y\mathrm{d}z < \infty$) and linearly polarized in x-direction. With these limitations, the electric field is defined as

$$\boldsymbol{E}(\boldsymbol{r},t) = \mathcal{E}_0 \hat{\boldsymbol{e}}_x g(x,y) u(t \mp z/c) e^{-i\omega_L(t\mp z/c)}, \tag{2.7}$$

where $\mathcal{E}_0$ is the field amplitude, $\hat{\boldsymbol{e}}_x$ is the unit vector in x-direction, $g(x,y)$ is the transverse mode profile and $u(t)$ is the temporal envelope. To satisfy normalizeability, the transverse mode profile $g(x,y)$ and the temporal envelope $u(t)$ have to be chosen accordingly. The vector potential can be calculated with $-\partial_t \boldsymbol{A}(\boldsymbol{r},t) = \mathfrak{Re}\{\boldsymbol{E}(\boldsymbol{r},t)\}$ [MWS$^+$]. First steps of solving this differential equation yields

$$\begin{aligned}\boldsymbol{A}(\boldsymbol{r},t) &= -\int \mathrm{d}t \mathfrak{Re}\big\{\boldsymbol{E}(\boldsymbol{r},t)\big\} \\ &= -\mathcal{E}_0 \hat{\boldsymbol{e}}_x g(x,y) \int \mathrm{d}t \cos\big[\omega_L(t\mp z/c)\big] u(t\mp z/c)\end{aligned}$$

under consideration that $\mathcal{E}_0$, $g(x,y)$ and $u(t)$ are all real-valued. Assuming that the temporal pulse length Δt is much larger than the optical oscillations $2\pi/\omega_L$, the envelope function can, in first order, be pulled out of the integral [MWS$^+$]. This leaves

$$\begin{aligned}&\boldsymbol{A}(\boldsymbol{r},t) \approx -\mathcal{E}_0 \hat{\boldsymbol{e}}_x g(x,y) u(t\mp z/c) \int \mathrm{d}t \cos\big[\omega_L(t\mp z/c)\big] \\ \Longrightarrow\quad &\boldsymbol{A}(\boldsymbol{r},t) \approx -\frac{\mathcal{E}_0 \hat{\boldsymbol{e}}_x}{\omega_L} g(x,y) u(t\mp z/c) \sin\big[\omega_L(t\mp z/c)\big]\end{aligned} \tag{2.8}$$

for the vector potential.

Inserting equation (2.8) into equation (2.5) gives an interaction phase shift described by

$$\varphi_p(\boldsymbol{r}) \approx -\frac{e^2\mathcal{E}_0^2}{2m_e\gamma\omega_L^2\hbar} g^2(x,y) \int_{-\infty}^{\infty} \mathrm{d}t' \sin^2\left[\omega_L\left(t' \mp \frac{z+vt'}{c}\right)\right] u^2\left(t' \mp \frac{z+vt'}{c}\right).$$

Now the substitution $t = t' \mp (z+vt')/c \rightarrow dt' = dt/(1\mp\beta)$ with $\beta = v/c$ is conducted. The $\mp$ sign denotes the relative traveling direction of the electrons compared to the light pulse, where $-$ is for co-propagating and $+$ for counter-propagating electrons and light pulses. Inserting this substitution and computing the integrals

$$\begin{aligned}
\varphi_p(x,y) &\approx -\frac{e^2\mathcal{E}_0^2}{2(1\mp\beta)m_e\gamma\omega_L^2\hbar} g^2(x,y) \int_{-\infty}^{\infty} \mathrm{d}t \underbrace{\sin^2(\omega_L t)}_{[1-\cos(2\omega_L t)]/2} u^2(t) \\
&\approx -\frac{e^2\mathcal{E}_0^2}{4(1\mp\beta)m_e\gamma\omega_L^2\hbar} g^2(x,y) \left[\int_{-\infty}^{\infty} \mathrm{d}t\, u^2(t) - \underbrace{\int_{-\infty}^{\infty} \mathrm{d}t \cos(2\omega_L t) u^2(t)}_{\approx 0}\right] \\
&\approx -\frac{e^2\mathcal{E}_0^2}{4(1\mp\beta)m_e\gamma\omega_L^2\hbar} g^2(x,y) \int_{-\infty}^{\infty} \mathrm{d}t\, u^2(t), \qquad (2.9)
\end{aligned}$$

gives an expression for the phase shift induced by the light field. Again in the second line of the calculation the slowly varying envelope compared to the electric field oscillation was used to compute the integral to be zero [MWS$^+$].

From an experimental point of view, equation (2.9) is not yet useful, as it contains the still abstract parameter of the electric field amplitude $\mathcal{E}_0$. The goal is to express the electric field amplitude with an experimental parameter, in this case, the laser pulse energy E_L.

For this, the field energy has to be calculated, which can be done with the Poynting vector

$$\boldsymbol{S} = \mathfrak{Re}\{\boldsymbol{E}\} \times \mathfrak{Re}\{\boldsymbol{H}\}, \qquad (2.10)$$

where $\boldsymbol{H} = c\varepsilon_0 \boldsymbol{k} \times \boldsymbol{E}$ is the magnetic field strength [PW]. In the paraxial approximation, this can be reduced to

$$\boldsymbol{H}(\boldsymbol{r},t) \approx c\varepsilon_0 \hat{\boldsymbol{e}}_z \times \boldsymbol{E}(\boldsymbol{r},t).$$

Plugging in the definition of the electric field (2.7) results in a magnetic field strength of

$$\boldsymbol{H}(\boldsymbol{r},t) \approx \varepsilon_0 c \mathcal{E}_0 \hat{\boldsymbol{e}}_y g(x,y) u(t \mp z/c) e^{-i\omega_L(t\mp z/c)}$$

and consequently, when inserting the magnetic field strength $\boldsymbol{H}$ and the electric field $\boldsymbol{E}$ into the definition of the Poynting vector, (2.10) gives

$$\boldsymbol{S}(\boldsymbol{r},t) \approx \varepsilon_0 c \hat{\boldsymbol{e}}_z \mathcal{E}_0^2 g^2(x,y) u^2(t) \mathfrak{Re}\{e^{-i\omega_L(t)}\}^2.$$

Note that we set $z = 0$ because we are interested in the intensity at the interaction plane. The energy per pulse E_L is then defined to be the integral over the absolute value of the Poynting vector in the spatial and temporal directions

$$\begin{aligned} E_L &= \varepsilon_0 c \mathcal{E}_0^2 \int_{-\infty}^{\infty} \mathrm{d}x\mathrm{d}y\, g^2(x,y) \int_{-\infty}^{\infty} \mathrm{d}t\, u^2(t) \underbrace{\cos(\omega_L t)}_{(1+\cos(2\omega_L t))/2} \\ &= \frac{\varepsilon_0 c \mathcal{E}_0^2}{2} \int_{-\infty}^{\infty} \mathrm{d}x\mathrm{d}y\, g^2(x,y) \left[\int_{-\infty}^{\infty} \mathrm{d}t\, u^2(t) + \underbrace{\int_{-\infty}^{\infty} \mathrm{d}t\, u^2(t) \cos(2\omega_L t)}_{\approx 0} \right] \\ &\approx \frac{\varepsilon_0 c \mathcal{E}_0^2}{2} \int_{-\infty}^{\infty} \mathrm{d}x\mathrm{d}y\, g^2(x,y) \int_{-\infty}^{\infty} \mathrm{d}t\, u^2(t), \end{aligned} \tag{2.11}$$

where in the second line the slowly varying envelope was used again [MWS+].

Now the energy per pulse (2.11) has to be reshuffled to express the electric field amplitude $\mathcal{E}_0$ in terms of the pulse energy E_L. This calculation then results in

$$\mathcal{E}_0^2 = \frac{2E_L}{\varepsilon_0 c \int_{-\infty}^{\infty} \mathrm{d}x\mathrm{d}y\, g^2(x,y) \int_{-\infty}^{\infty} \mathrm{d}t\, u^2(t)}. \tag{2.12}$$

In order to compute the phase imprint for a known intensity distribution $g^2(x,y)$ and a known pulse energy E_L, equation (2.12) must be inserted into equation (2.9). Doing this, yields

$$\varphi_p(x,y) \approx -\frac{\alpha}{2\pi(1 \mp \beta)} \frac{E_L}{E_e} \frac{\lambda_L^2 g^2(x,y)}{\int g^2(x,y)\mathrm{d}x\mathrm{d}y} \tag{2.13}$$

an equation of the phase imprint that only has easy-to-obtain experimental values in it [MWS+]. In equation (2.13) the fine structure constant $\alpha = e^2/(4\pi\varepsilon_0\hbar c)$ was used.

We see that, in the approximation of a sufficiently short interaction time, the temporal pulse shape is not important for the electron interaction. The only relevant parameters are the pulse energy E_L, light wavelength λ_L and the electron energy E_e. With the transverse laser intensity distribution $g^2(x,y)$ the electron wavefront can be shaped directly.

2.2. Lens Imprint

An obvious usage for a phase imprint would be the correction of aberrations in electron lenses. Otto Scherzer showed in 1936 that it is not possible to construct aberration-free electron lenses that follow certain assumptions. These are that the electromagnetic fields are radially symmetric and static and that there are no space charges in the optical axis [Sch]. As most state-of-the-art electron lenses are subject to Scherzer's theorem, they are also subject to the inevitable aberrations.

Due to the fact that the light field is never static, and we are not limited to rotational symmetric light intensity distributions, the limitations from Scherzer's theorem can be

overcome. Additionally to the correction of existing electron lenses, it is also possible to construct electron lenses with a laser beam interaction. As Scherzer's assumptions do not apply to a laser lens, it would be conceivable to construct a perfect laser lens.

In the following, we will discuss how Laguerre-Gaussian modes can be used to realize almost ideal convex and concave ponderomotive lenses. The latter case is especially interesting because energy-preserving concave lenses are impossible to realize with static, cylindrically-symmetric electromagnetic fields. For a perfect electron lens with a focal length f, the applied phase shift is

$$\varphi_l(\rho) = -\frac{\pi}{f\lambda_e}\rho^2, \tag{2.14}$$

where ρ is the distance from the optical axis and $\lambda_e = h/(\gamma m_e v)$ is the electron wavelength with h being the Planck constant and v being the electron velocity. For a convex lens, the focal length f will be positive and for a concave lens it will be negative, so the functional dependency will change depending on the type of lens that will be created.

2.2.1. Convex lens

In a convex lens, the focal length f is positive, which means that the phase imprint needs to be $\propto -\rho^2$. As the phase imprint in equation (2.13) is proportional to the negative intensity distribution, the laser intensity distribution has to be proportional to ρ^2

$$I(\rho) \propto \rho^2.$$

In general the addition of a constant intensity background will also yield the same result because of the negligibility of a constant phase offset.

The obvious thing to do would be to assume that the light intensity distribution is

$$I(\rho) = \begin{cases} A\rho^2 & \text{if } \rho < \rho_0 \\ 0 & \text{else} \end{cases} \tag{2.15}$$

with A being a constant and ρ_0 being the size of the lens. This ansatz has two major issues. The first one is a fundamental physical problem and the second is a technical problem. The first problem arises from the fact that equation (2.15) is not continuous at the position ρ_0, which is impossible for a light field, due to it being a solution of the wave equation. In reality, the discontinuity will be smeared out over the minimal resolution of the beam. The second, reason is that creating arbitrary intensity distributions using wavefront shaping techniques often comes with technical challenges, such as loss and unwanted speckle patterns.

These problems do not arise if a fundamental solution to the Helmholtz equation with circular aperture can be used to realize the desired intensity patterns. As we show in the

following, this can be done with Laguerre-Gaussian beams. These are defined as

$$E(\rho,\phi,z) = E_0\sqrt{\frac{2p!}{\pi(p+|l|)!}}\frac{w_0}{w(z)}\left(\frac{\rho\sqrt{2}}{w(z)}\right)^{|l|}\exp\left(-\frac{\rho^2}{w^2(z)}\right)\times \\ L_p^{|l|}\left(\frac{2\rho^2}{w^2(z)}\right)\exp\left(-ik\frac{\rho^2}{2R(z)}\right)\exp\left(-il\phi\right)\exp\left(i\psi(z)\right)\exp(-ikz) \tag{2.16}$$

with

$$w(z) = w_0\sqrt{1+\left(\frac{z}{z_R}\right)^2} \\ R(z) = z\left[1+\left(\frac{z_R}{z}\right)^2\right] \\ \psi(z) = (|l|+2p+1)\arctan\left(\frac{z}{z_R}\right).$$

In equation (2.16) E_0 is the electric field amplitude, w_0 is the beam waist in the focal spot, $L_p^{|l|}$ are the generalized Laguerre polynomials, $k = 2\pi/\lambda_L$ the wave vector with λ_L the wavelength and $z_R = \pi w_0^2/\lambda_L$ the Rayleigh length.

In this calculation we assume a very short interaction time and with that approximate that the complete interaction takes place in the focal spot of the light field. This means that only the $z = 0$ plane is relevant for the intensity, and simplifies (2.16) to

$$E(\rho,\phi,z) = E_0\sqrt{\frac{2p!}{\pi(p+|l|)!}}\left(\frac{\rho\sqrt{2}}{\omega_0}\right)^{|l|}\exp\left(-\frac{\rho^2}{\omega_0^2}\right)L_p^{|l|}\left(\frac{2\rho^2}{\omega_0^2}\right)\exp(-il\phi), \tag{2.17}$$

Because the acquired phase shift is proportional to the intensity distribution, we are not interested in the field but in the intensity $I(\rho) = \varepsilon_0 c|E(\rho,\phi,0)|^2/2$ [PW]. Inserting equation (2.17) into the intensity definition gives

$$I(\rho) = \frac{\varepsilon_0 c I_0}{2}\left(\frac{\rho\sqrt{2}}{w_0}\right)^{2|l|}\exp\left(-\frac{2\rho^2}{w_0}\right)L_p^{|l|}\left(\frac{2\rho^2}{w_0^2}\right)^2, \tag{2.18}$$

with the first few Laguerre polynomials

$$L_0^{|l|}(x) = 1 \\ L_1^{|l|}(x) = 1+|l|-x \\ L_2^{|l|}(x) = \frac{1}{2}\left[x^2-(2|l|+2)+(|l|+1)(|l|+2)\right] \\ L_3^{|l|}(x) = \frac{1}{6}\left[-x^3+3(|l|+3)x^2-3(|l|+2)(|l|+3)x+(|l|+1)(|l|+2)(|l|+3)\right].$$

If we neglect the exponential term in (2.18) we see that $I(\rho) \propto \rho^2$ for $l = 1$ and $p = 0$. We will therefore use the LG10 mode to get a first approximation of an ideal lens.

LG10 Mode

The first mode that we are interested in is the LG10 mode. This means our intensity (2.18) simplifies to

$$I_{10}(\rho) = \frac{\varepsilon_0 c I_0}{2} \left(\frac{\rho\sqrt{2}}{w_0} \right)^2 \exp\left(-\frac{2\rho^2}{w_0^2} \right). \tag{2.19}$$

The Taylor expansion around $\rho = 0$ of equation (2.19) yields

$$I_{10}(\rho) = \frac{\varepsilon_0 c I_0}{2} \left(\frac{2}{w_0^2}\rho^2 - \frac{4}{w_0^4}\rho^4 + \frac{4}{w_0^6}\rho^6 - \frac{8}{3w_0^8}\rho^8 \right) + \mathcal{O}(\rho^9), \tag{2.20}$$

which has the desired quadratic phase shift. This means that in close proximity to the center of the LG10 mode we get the same functional dependency as for a perfect lens.

To get an estimate of how big the radius is, in which the desired phase shift is approximated well, we compare the relative error between the first term of the Taylor expansion, representing an ideal lens, in (2.20) and the LG10 mode in (2.19). The relative error ϵ is thus defined as

$$\begin{aligned} \epsilon &= \left| \frac{\frac{2}{w_0^2}\rho^2 - \left(\frac{\rho\sqrt{2}}{w_0} \right)^2 \exp\left(-\frac{2\rho^2}{w_0^2} \right)}{\frac{2}{w_0^2}\rho^2} \right| \\ &= \left| 1 - \exp\left(-\frac{2\rho^2}{w_0^2} \right) \right|. \end{aligned}$$

Reshuffling the relative error to

$$\rho = \sqrt{-\frac{\log(1-\epsilon)}{2}} \tag{2.21}$$

(note that the relative error ϵ can only be between 0 and 1) then yields the effective size of the lens, depending on the desired maximum deviation from a perfect lens.

In figure 2.1, the LG10 intensity distribution is shown as well as the cross-section of the beam, which is compared to the first order of the Taylor expansion, corresponding to an ideal lens. Also the effective size of the lens can be seen assuming an error of $\epsilon = 0.1$ which yields an effective lens radius of $\rho_{\text{eff}} = 0.2295 w_0$.

An LG10 mode thus yields a reasonable approximation of an ideal lens. Going one step further, we will now investigate higher order LG modes, to see whether they can be used to achieve an even better approximation.

LG20 Mode

Using the LG20 mode of equation (2.18) yields

$$I_{20}(\rho) = \frac{\varepsilon_0 c I_0}{2} \left(\frac{\rho\sqrt{2}}{w_0} \right)^4 \exp\left(-\frac{2\rho^2}{w_0^2} \right)$$

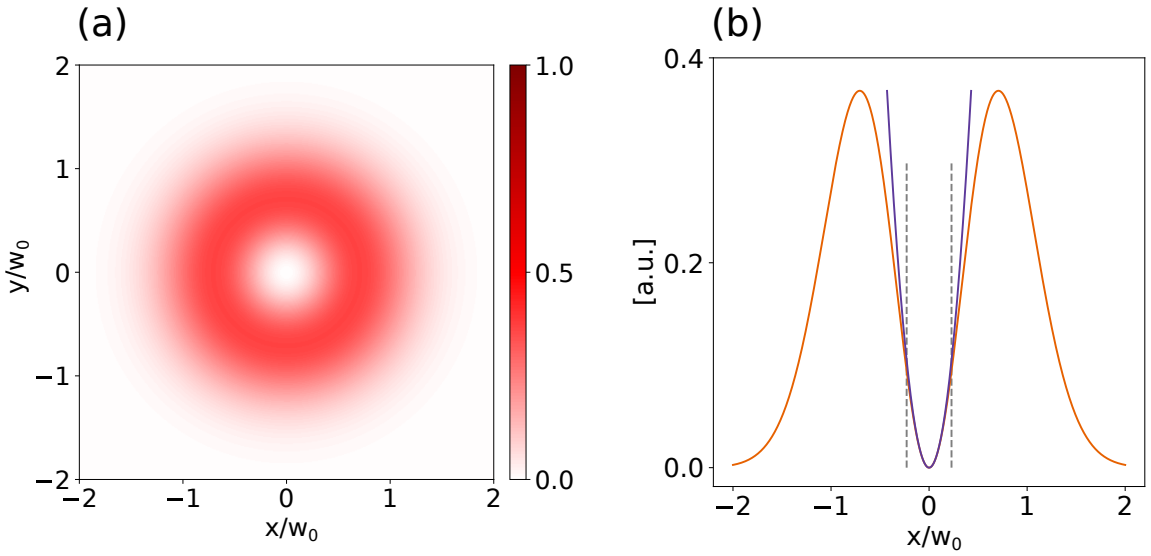


Figure 2.1.: This figure visualizes the LG10 mode in units of the beam waist w_0. In (a) we can see the intensity distribution of a perfect LG10 beam. In (b) we can see the cross-section of the beam in orange, and the first order of the Taylor expansion in equation (2.20) which represents a perfect lens, as well as the effective lens radius $\rho_{\text{eff}} = 0.2295 w_0$ assuming an acceptable error of $\epsilon = 10\%$ as dashed gray vertical lines.

with the Taylor expansion

$$I_{20}(\rho) = \frac{\varepsilon_0 c I_0}{2}\left(\frac{4}{w_0^4}\rho^4 - \frac{8}{w_0^6}\rho^6 + \frac{8}{w_0^8}\rho^8\right) + \mathcal{O}(\rho^{10}). \tag{2.22}$$

Here we can see that the lowest order is $\propto \rho^4$, which is too high for a lens, so it cannot be used directly to our advantage, but it can be used to enhance the lensing capabilities of the LG10 mode.

Sum of LG10 and LG20 modes

Here we use a superposition of the LG10 and LG20 mode to greatly increase the effective size of the lens. Even though the experimental realization of such an addition of Intensities is not easily achieved, we will consider this superposition, because it can in theory be used to greatly increase the effective radius of the lens. We define our $I_{10+20}(\rho)$ mode as

$$I_{10+20}(\rho) = \frac{\varepsilon_0 c I_0}{2}\left(\frac{\rho\sqrt{2}}{w_0}\right)^2\left[1 + \left(\frac{\rho\sqrt{2}}{w_0}\right)^2\right]\exp\left(-\frac{2\rho^2}{w_0^2}\right), \tag{2.23}$$

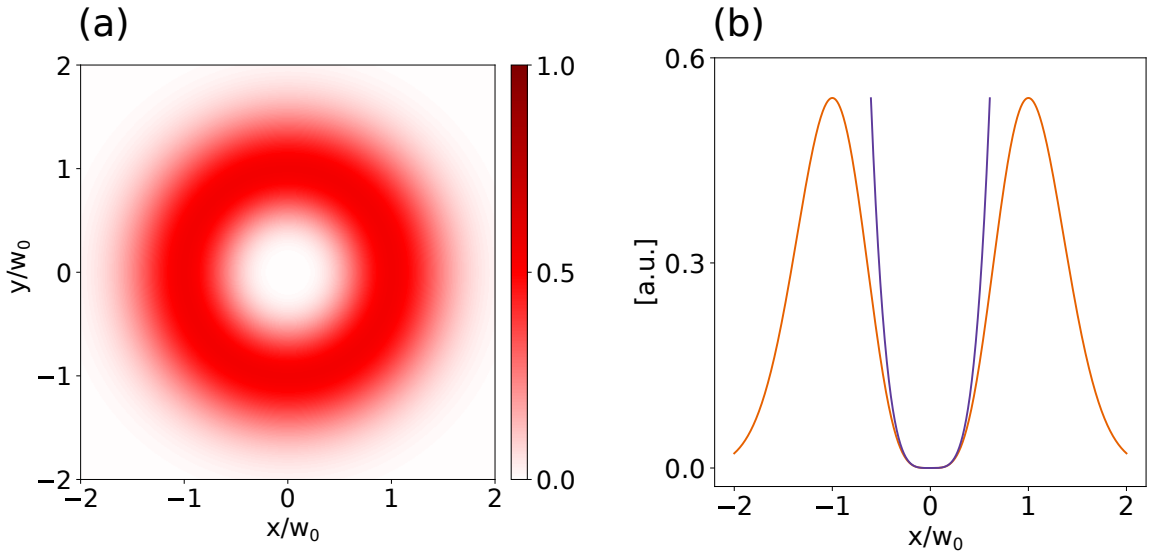


Figure 2.2.: This figure visualizes the LG20 mode. In (a) we can see the intensity distribution of a perfect LG20 beam in units of the beam waist. In (b) we can see the cross-section of the beam in orange, and the first order of the Taylor expansion in equation (2.22) in purple.

which then has the corresponding Taylor expansion

$$I_{10+20}(\rho) = \frac{\varepsilon_0 c I_0}{2}\left(\frac{2}{w_0^2}\rho^2 - \frac{4}{w_0^6}\rho^6 + \frac{16}{3w_0^8}\rho^8\right) + \mathcal{O}(\rho^9). \tag{2.24}$$

We can see that if we compare equation (2.20) with equation (2.24) we don't have a dependency on ρ^4 anymore in the Taylor expansion, which will give a larger effective lens radius compared to the LG10 mode.

To get the effective radius the same method is used as in section 2.2.1, which means defining the relative error in dependence of the radius ρ_{eff}. For the relative error ϵ we get

$$\begin{aligned}\epsilon &= \frac{\left|\frac{2}{w_0^2}\rho_{\text{eff}}^2 - \frac{2}{w_0^2}\rho_{\text{eff}}^2\left(1+\frac{2}{w_0^2}\rho_{\text{eff}}^2\right)\exp\left(-\frac{2\rho_{\text{eff}}^2}{w_0^2}\right)\right|}{\frac{2}{w_0^2}\rho_{\text{eff}}^2} \\ &= \left|1 - \left(1+\frac{2}{w_0^2}\rho_{\text{eff}}^2\right)\exp\left(-\frac{2\rho_{\text{eff}}^2}{w_0^2}\right)\right|,\end{aligned}$$

which can be solved either numerically or using the Lambert W function. For a relative error of $\epsilon = 0.1$, we get an effective lens radius of $\rho_{\text{eff}} = 0.515661 w_0$. When compared with the effective radius in section 2.2.1, we can see that we get an improvement of a factor of 2.25 compared to the plain LG10 mode.

In figure 2.3, we can see the intensity distribution of the LG10+LG20 beam, the cross-section of the beam as well as the effective radius ρ_{eff}.

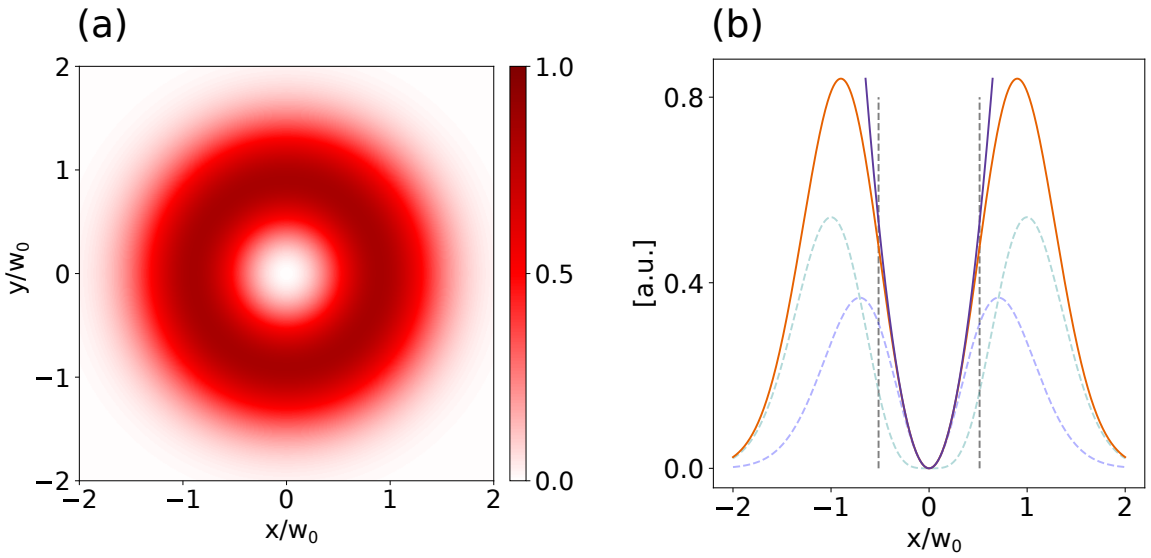


Figure 2.3.: This figure visualizes the sum of the LG10 and LG20 modes. In (a) we can see the intensity distribution of a perfect LG10+LG20 normalized to a maximum value of one in units of the beam width. In (b) we can see the cross-section of the beam in orange, and the first order of the Taylor expansion in equation (2.24) in purple, the LG10 mode as dashed blue, the LG20 mode as dashed cyan and as dashed gray the boundaries in which the sum of modes has a relative error of less then 10% compared to a perfect parabola. This effective radius is $\rho_{\text{eff}} = 0.515661 w_0$.

Focal length

LG10 mode To get out the focal length of the ponderomotive lens, we can compare the definition of a perfect lens in equation (2.14) with the phase shift that an electron experiences according to equation (2.13).

We first calculate the denominator in equation (2.13)

$$\int_{-\infty}^{\infty} \mathrm{d}x \int_{-\infty}^{\infty} \mathrm{d}y\, I_{10}(x,y) = 2\pi \int_{0}^{\infty} \mathrm{d}\rho\, \rho I_{10}(\rho) = \frac{\varepsilon_0 c I_0}{2} \frac{\pi w_0^2}{2}.$$

Now we can insert the first order of the Taylor expansion in (2.20) into equation (2.13), which results in a phase shift of

$$\varphi_p(\rho) \approx -\frac{2\alpha}{\pi^2(1 \mp \beta)} \frac{E_L}{E_e} \frac{\lambda_L^2}{w_0^4} \rho^2. \tag{2.25}$$

A comparison of coefficients can be conducted between equation (2.14) and equation (2.25) yielding an expression for the focal length

$$f_{10}(E_L) \approx (1 \mp \beta) \frac{\pi^3 w_0^4}{2\alpha \lambda_L^2 \lambda_e} \frac{E_e}{E_L}. \tag{2.26}$$

The derivation of the focal length can also be found in [MWS$^+$].

Sum of LG10 and LG20 modes The focal length of the sum of the modes can be calculated in the same way as the focal length for the LG10 mode. The only difference is that the integral over the whole area is significantly larger than compared to the LG10 mode. This can clearly be seen when looking at figure 2.3 (b) where the area under the orange curve is significantly larger than the area under the dashed blue curve. This change in the normalization means, compared to (2.13), that the applied phase shift will get smaller, and we have a direct trade-off between the strength of the lens and the effective radius.

To quantify the difference, the integral of I_{10+20} has to be evaluated

$$\int_{-\infty}^{\infty} \mathrm{d}x \int_{-\infty}^{\infty} \mathrm{d}y \, I_{10+20}(x, y) = 2\pi \int_0^{\infty} \mathrm{d}\rho \, \rho I_{10+20}(\rho) = \frac{\varepsilon_0 c I_0}{2} \frac{3\pi w_0^2}{2}$$

and plugged into equation (2.13), to get

$$\varphi_p(\rho) \approx -\frac{2\alpha}{\pi^2 (1 \mp \beta)} \frac{E_L}{E_e} \frac{\lambda_L^2}{w_0^4} \rho^2, \tag{2.27}$$

which then, again using a comparison of coefficients, can be used to calculate the focal length of the lens

$$f_{10+20}(E_L) \approx (1 \mp \beta) \frac{3\pi^3 w_0^4}{2\alpha \lambda_L^2 \lambda_e} \frac{E_e}{E_L}. \tag{2.28}$$

We see that the focal length of LG10 mode (2.26) is shorter by a factor of 3 compared to the focal length of the LG10 + LG20 mode (2.28), which makes the LG10 lens more efficient.

2.2.2. Concave lens

Contrary to the convex lens, in a concave lens, we have a negative focal length. When we compare that sign change with equation (2.14), we see that a phase shift proportional to ρ^2 is needed. Because of equation (2.13), it can directly be seen that the intensity distribution of the lens-inducing light field also has to be of the form

$$I(\rho) \propto \rho^2.$$

Gaussian beam (LG00 Mode)

A general Gaussian beam can be described as follows:

$$E_{00}(\rho, z) = E_0 \frac{w_0}{w(z)} \exp\left(-\frac{\rho^2}{w^2(z)}\right) \exp\left(-ik\frac{\rho^2}{2R(z)}\right) \exp(i\psi(z)) \exp(-ikz). \quad (2.29)$$

Following the arguments for the convex lens (section 2.2.1), only the case of $z = 0$ will be considered. This reduces equation (2.29) to

$$E_{00}(\rho, 0) = E_0 \exp\left(-\frac{\rho^2}{w_0^2}\right) \exp(-ikz). \quad (2.30)$$

Again the relevant parameter is not the field but the intensity

$$I_{00}(\rho) = \frac{\varepsilon_0 c I_0}{2} \exp\left(-\frac{2\rho^2}{w_0^2}\right). \quad (2.31)$$

Now we will check how well the chosen intensity distribution matches the requirements. For doing so, a Taylor expansion of equation (2.31) is conducted:

$$I_{00}(\rho) = \frac{\varepsilon_0 c I_0}{2}\left(1 - \frac{2}{w_0^2}\rho^2 + \frac{2}{w_0^4}\rho^4 - \frac{4}{3w_0^6}\rho^6 + \frac{2}{3w_0^8}\rho^8\right) + \mathcal{O}(\rho^9). \quad (2.32)$$

The quadratic term in the Taylor expansion gives the "perfect" lens, while the terms of higher order are aberrations from the perfect lens. To get a measure of how large the effective lens radius is, we take the same approach as before, namely, we have a look at the relative error function and search for a radius ρ_{eff}, where the lens has an error smaller than ϵ. The relative error function takes the form

$$\epsilon = \frac{\left|1 - \frac{2}{w_0^2}\rho_{\text{eff}}^2 - \exp\left(-\frac{2\rho_{\text{eff}}^2}{w_0^2}\right)\right|}{1 - \frac{2}{w_0^2}\rho_{\text{eff}}^2},$$

which yields, under the assumption that the acceptable lens error is $\epsilon = 0.1$, an effective radius of $\rho_{\text{eff}} = 0.4333 w_0$.

In figure 2.4 a Gaussian beam is displayed, as well as the cross-section of it and the comparison with the first order of the Taylor expansion.

Modified Gaussian mode

To further improve the size of the effective lens, the higher order terms in the Taylor expansion of the Gaussian (2.32) have to be suppressed. To achieve this, a modified Gaussian is chosen that has the form of

$$I_{00\text{mod}}(\rho) = I_{00}(\rho^2/w_0^2) = \exp\left(-\frac{2\rho^4}{w_0^4}\right), \quad (2.33)$$

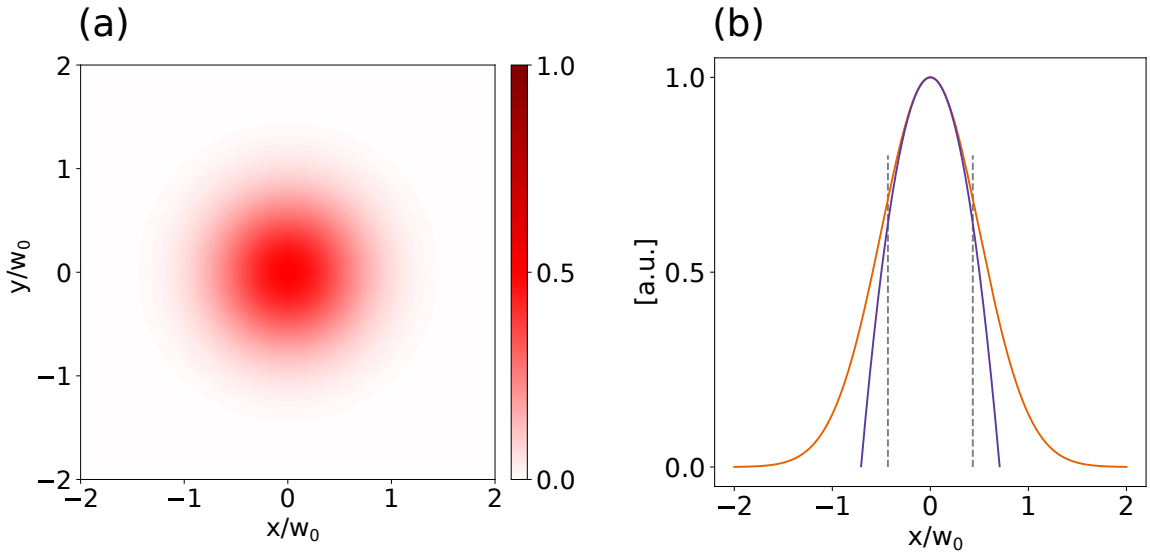


Figure 2.4.: This figure shows a Gaussian beam corresponding to a concave lens and the relevant features. In (a) a Gaussian beam, as defined in equation (2.31), is visualized, where the x and the y axis are given in units of the beam waist. In (b) the cross section of (a) is plotted in orange, and the first order of the Taylor expansion (equation (2.32)) is plotted in purple. The dashed vertical lines in (b) represent the effective lens radius $\rho_{\text{eff}} = 0.4333 w_0$, for a maximum 10% deviation from a perfect lens.

where I_{00} is the definition of the Gaussian intensity in equation (2.31). As this mode is not a solution of the Helmholtz equation it is not easily constructed in an experimental setting. One possibility to construct such an intensity distribution would be the use of the Gerchberg Saxton Algorithm as described in [MWS+]. If now the Taylor series of equation (2.33)

$$I_{00\text{mod}}(\rho) = \frac{\varepsilon_0 c I_0}{2}\left(1 - \frac{2}{w_0^4}\rho^4 + \frac{2}{w_0^8}\rho^8 - \frac{4}{3w_0^{12}}\rho^{12}\right) + \mathcal{O}(\rho^{13}) \tag{2.34}$$

is taken, and compared to the Taylor expansion of the Gaussian (2.32), it can be seen that the modified Gaussian can be used to enhance the lensing properties of the Gaussian beam.

In figure 2.5 the modified Gaussian, as well as its cross-section is visualized.

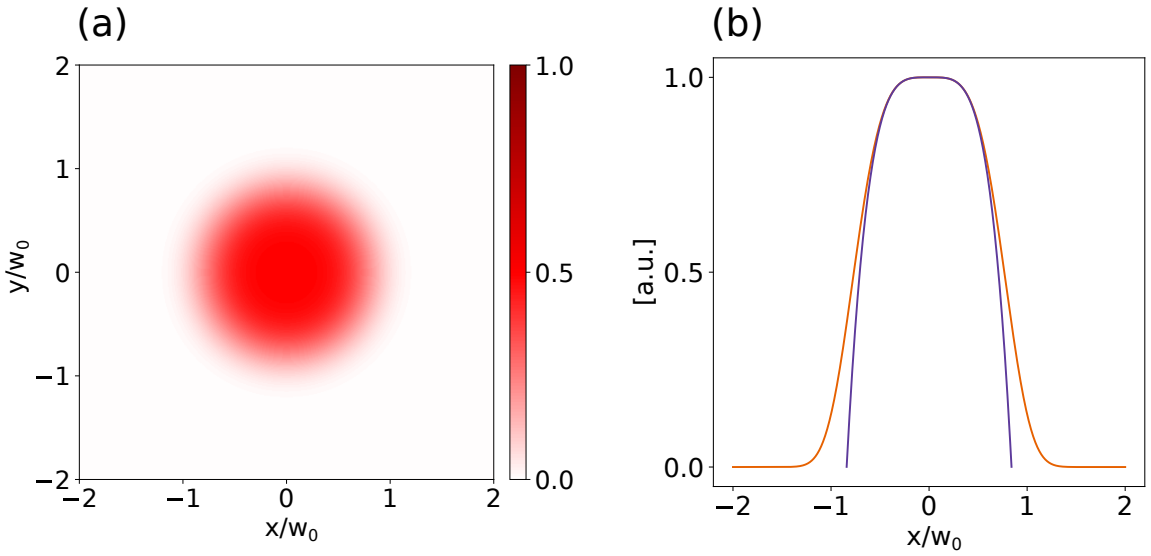


Figure 2.5.: In (a) the modified Gaussian defined in equation (2.33) is shown, and in (b) the cross section (orange) as well as the first order of the Taylor expansion (2.34, purple) is shown. The respective axes are defined in units relative to the beam waist.

Sum of Gaussian and Modified Gaussian Modes

To improve the size of the effective lens, we take the sum of the Gaussian mode and the modified Gaussian mode

$$I_{00+00\mathrm{mod}}(\rho) = I_{00}(\rho) + I_{00\mathrm{mod}}(\rho) = \exp\left(-\frac{2\rho^2}{w_0^2}\right) + \exp\left(-\frac{2\rho^4}{w_0^4}\right). \tag{2.35}$$

To calculate the Taylor expansion of (2.35), we just take the sum of the two Taylor expansions (2.32) and (2.34). This yields

$$I_{00+00\mathrm{mod}}(\rho) = \frac{\varepsilon_0 c I_0}{2}\left(2 - \frac{2}{w_0^2}\rho^2 - \frac{4}{3w_0^6}\rho^6\right) + \mathcal{O}(\rho^8), \tag{2.36}$$

where it can immediately be seen that the ρ^4 term dropped out. This means that the Taylor expansion is more accurate over a larger radius ρ_{eff} compared to the normal Gaussian lens.

To quantify the enhancement the effective size of the lens we again calculated ρ_{eff} using the relative error function

$$\epsilon = \frac{\left|2 - \frac{2}{w_0^2}\rho_{\mathrm{eff}}^2 - \exp\left(-\frac{2\rho_{\mathrm{eff}}^2}{w_0^2}\right) - \exp\left(-\frac{2\rho_{\mathrm{eff}}^4}{w_0^4}\right)\right|}{2 - \frac{2}{w_0^2}\rho_{\mathrm{eff}}^2}.$$

Evaluating this expression with an assumed acceptable error of $\epsilon = 0.1$, the effective beam radius is $\rho_{\text{eff}} = 0.8677 w_0$, and thus a factor two larger compared to the simple Gaussian beam previously.

Figure 2.6 shows the intensity distribution and the cross-section of the sum of the Gaussian with the modified Gaussian, as well as the cross-section and the different components that add up to the final distribution.

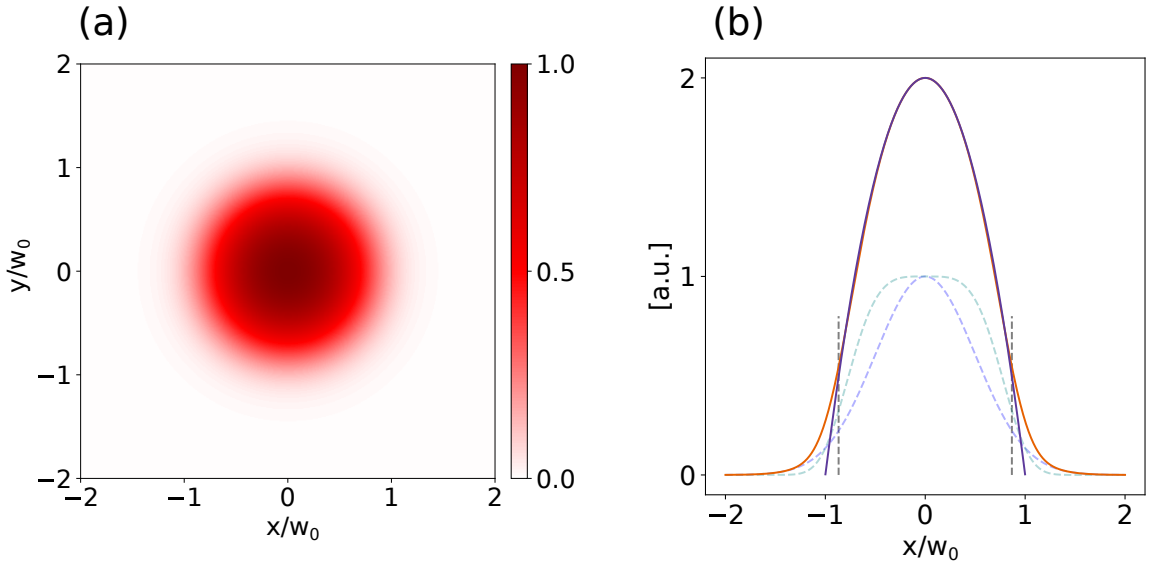


Figure 2.6.: This figure shows the sum of the Gaussian beam and the modified Gaussian beam. In (a) the intensity distribution of equation (2.35) is shown. In (b) the cross section of (a) is in orange, the first order Taylor expansion (2.36) in purple, the Gaussian distribution (2.31) cross section is visualized in dashed blue, and the modified Gaussian distribution (2.33) cross-section is shown in dashed teal. Additionally, the size of the effective lens radius $\rho_{\text{eff}} = 0.8677 w_0$ is depicted in dashed black.

Focal length

Gaussian The focal length of the Gaussian mode can be obtained by comparing the phase shift that would apply for a perfect lens (2.14) and the phase shift that the Gaussian mode induces. This derivation is conducted in the appendix of [MWS$^+$]. Note that the focal length for a concave lens is defined with a negative sign.

To get the phase shift, which is induced from the intensity in equation (2.31), we first calculate:

$$\int_{-\infty}^{\infty} \mathrm{d}x \int_{-\infty}^{\infty} \mathrm{d}y \, I_{00}(x,y) = 2\pi \int_{0}^{\infty} \mathrm{d}\rho \, \rho I_{00}(\rho) = \frac{\varepsilon_0 c I_0}{2} \frac{\pi w_0^2}{2}.$$

The normalization and the second order of the Taylor expanded Gaussian mode can then be plugged into equation (2.13) to yield

$$\varphi_p(\rho) \approx -\frac{\alpha}{\pi^2(1 \mp \beta)} \frac{E_L}{E_e} \frac{\lambda_L^2}{w_0^2} \left(1 - \frac{2}{w_0^2}\rho^2\right). \tag{2.37}$$

As a constant phase factor can be ignored, only the term with the ρ^2 in equation (2.37) is relevant for the focal length calculation. The focal length itself can be obtained by conducting a comparison of coefficients between equations (2.14) and (2.37)

$$f_{00}(E_L) \approx (1 \mp \beta) \frac{\pi^3 w_0^4}{2\alpha\lambda_e\lambda_L^2} \frac{E_e}{E_L}. \tag{2.38}$$

Sum of Gaussian and modified Gaussian To obtain the focal length of the lens with the improved effective radius, the same scheme as above can be applied. The normalization of equation (2.35) is

$$\int_{-\infty}^{\infty} \mathrm{d}x \int_{-\infty}^{\infty} \mathrm{d}y \, I_{00+00\mathrm{mod}}(x,y) = 2\pi \int_0^{\infty} \mathrm{d}\rho \, \rho I_{00+00\mathrm{mod}}(\rho) = \frac{\varepsilon_0 c I_0}{2} \frac{\pi}{4} (2 + \sqrt{2}\pi) w_0^2$$

and following that, the phase imprint, using the Taylor expansion from equation (2.36), is calculated to be

$$\varphi_p(\rho) \approx -\frac{4\alpha}{\pi^2(1 \mp \beta)(2 + \sqrt{2}\pi)} \frac{E_L}{E_e} \frac{\lambda_L}{w_0^2} \left(1 - \frac{1}{w_0^2}\rho^2\right).$$

Again using a comparison of coefficients, the focal length can be evaluated to be

$$f_{00+00\mathrm{mod}}(E_L) \approx -(1 \mp \beta)(2 + \sqrt{2}\pi) \frac{\pi^3 w_0^4}{4\alpha\lambda_e\lambda_L^2} \frac{E_e}{E_L}. \tag{2.39}$$

Comparing the focal length of the Gaussian in equation (2.38) and the sum of the Gaussian's in equation (2.39), it can be seen that the focal length of the Gaussian is smaller by a factor of $(2 + \sqrt{2}\pi)/2 \approx 3.22$ than the focal length of the summation. This means that for the same energy in the light pulse E_L the lensing effect is smaller for the sum of Gaussian's, but the effective radius ρ_{eff} is getting larger and vice versa.

In conclusion we have shown in this chapter a derivation for ponderomotive interaction between light and electrons, as well as possible new electron optics based on this phenomenon. These electron lenses are theoretical because in a realistic setting the intensity distribution is not perfect. In appendix A.1 we show the treatment of a realistic electron lens based on [MWS+].

3. Ray Tracing

In this chapter, we will introduce a classical ray tracing description of electron optics. We will use it to describe the propagation of electrons in an arbitrary setup and to test the proposed ponderomotive lenses from section 2.2.

In section 3.1 we will discuss a classical ray-optics description of electrons. In the ray optics picture, the ponderomotive phase shifts of equation (2.13) are replaced by a light induced momentum kick. While this model cannot predict interference effects, it can still be used to predict electron probability distributions where the spatial resolution is limited. Section 3.2 gives an overview of the implementation of the model described in 3.1. In the last section of this chapter 3.3 we present a simulation that shows the effect of the of the lenses proposed in section 2.2. It can be seen that lensing is achieved and that lens quality is sufficiently good for small aperture diameters.

3.1. Modeling

In this section, the modeling of the different parts of the simulation is described. As the primary method used here is ray-tracing, the framework of ABCD-matrices is used extensively. The section starts with the description of a single electron in free space, based on which an electron beam is defined via random sampling of initial conditions. After the electron description, the components of the optical setup are described. These include simple free propagation over a certain distance, a lens, and the interaction with a light field.

3.1.1. Electron Description

The electrons in this simulation are assumed to be classical, described by their position (x, y and z) and momentum (p_x, p_y and p_z). The main propagation direction of each electron is assumed to be the positive z-direction, and the starting point for each electron is at $z = 0$. Note that this is different to the wave optics description in chapter 2. Additionally, we assume the paraxial approximation (p_x, $p_y \ll p_z$) and a monochromatic electron beam (every electron has the same initial momentum $|\boldsymbol{p}|$). The momentum in z-direction p_z can be replaced by one global input velocity times the electron mass $m_e v$. Using the conservation of energy and momentum, the electron trajectories are predetermined by their starting conditions. This reduces the free variables per electron from six to four (x, y, p_x, p_y) plus one, for all electrons constant, input velocity v.

The transverse momenta p_x and p_y can be mapped to the angles θ_x and θ_y. These angles correspond to the angle between the z-axis and the momentum component in the

respective direction, and thus can be written as $\theta_{x,y} = \arcsin(p_{x,y}/|\boldsymbol{p}|)$. A visualization of the starting conditions of the electron beam is shown in figure 3.1. The classical state of each electron is thus described by a vector:

$$\psi = \begin{pmatrix} x \\ y \\ \theta_x \\ \theta_y \end{pmatrix}$$

Using this representation for the electron state and the aforementioned paraxial approximation clears the path to using the ABCD-Matrix formulation (Chapter 9 of [PW]). In this framework, optical elements can be described by a matrix multiplication between the state vector ψ and a specific matrix that describes the optical element. If a state is propagating through multiple optical elements at a time, the matrices can just be multiplied, and then applied to the incoming state to get the outgoing state. The following sections will make use of a 2D version of the ABCD-Matrix formulation to efficiently calculate the electron trajectories. Considering an optical setup where an incoming electron beam ψ^{in} propagates in free space (described by the matrix M_{free1}), encounters a lens (M_{lens}) and propagates again freely (M_{free2}) the outgoing electron state ψ^{out} can be calculated as

$$\psi^{\text{out}} = M_{\text{free2}} M_{\text{lens}} M_{\text{free1}} \psi^{\text{in}}.$$

3.1.2. Electron Beam

To generate an image more than one electron needs to be simulated, and with that, the question arises of how to generate the electron beam.

To avoid creating artifacts the initial electron positions x and y will be randomly chosen. This is also in agreement with the experimental setup [MWS$^+$], where, on average, only 0.1 electrons per pulse enter the aperture at a random position. This means that the random initial position is not only good to reduce artifacts, but is also closer to the experimental reality than a regularly spaced grid of electrons. Figure 3.1 represents the random initial conditions for the electrons with the described properties.

To encode the input electron beam intensity distribution either the random number generator for the electron positioning must be adapted to the desired input, or a weighting factor is added to each electron. In this implementation, a weighting factor is added, which has the consequence that the output will not represent single electrons, but will represent the probability of an electron hitting a certain pixel.

3.1.3. Component description

In the setup described in figure 1.2, we can see that only three optical elements are needed to simulate it properly. The first one is the free propagation, the second one is a lens, and the third one is the ponderomotive interaction between the electrons and the light field.

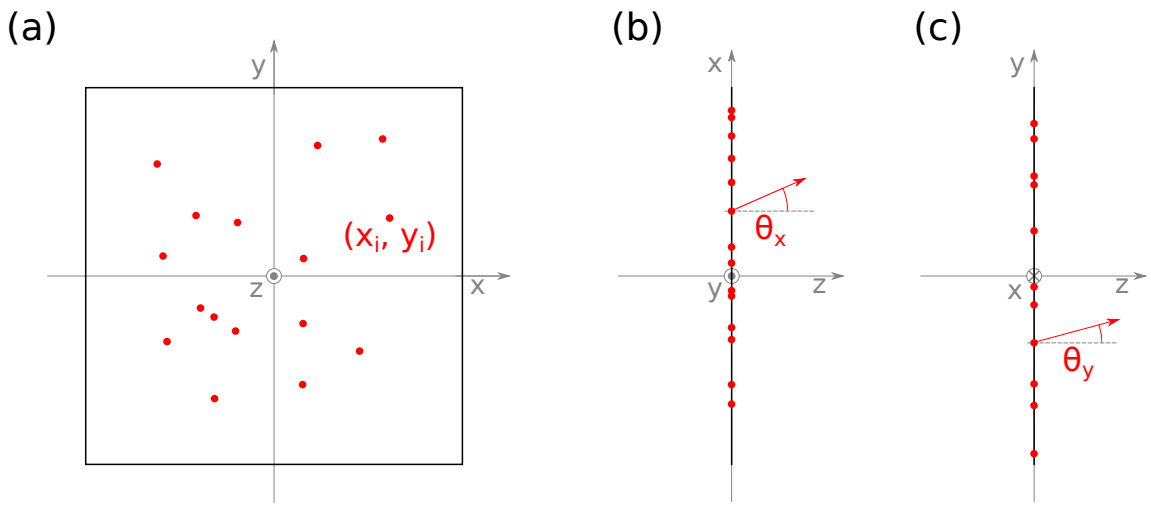


Figure 3.1.: Visualization of the initial state where the black rectangle represents the field of view in the x-y plane at $z = 0$. In (a) the random positions in the x-y plane are shown. In (b) the definition of the θ_x value can be seen, and in (c) the definition of the θ_y value. It should be noted that the shown angles are exaggerated.

Free Propagation

The free propagation can be implemented using an ABCD-Matrix

$$M_{\text{free}} = \begin{pmatrix} 1 & 0 & d & 0 \\ 0 & 1 & 0 & d \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \tag{3.1}$$

where d is the propagation distance. To calculate a new state ψ^{out} from an initial state ψ^{in} we apply the following operation:

$$\psi^{\text{out}} = M_{\text{free}} \psi^{\text{in}}$$

Lens

To calculate the effect that a lens has on the electron beams we can also use an ABCD matrix. For a given focal length f (if the focal length is positive we have a convex lens, if

it is negative we have a concave one) the matrix has the form of:

$$M_{\text{lens}} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -\frac{1}{f} & 0 & 1 & 0 \\ 0 & -\frac{1}{f} & 0 & 1 \end{pmatrix}. \tag{3.2}$$

Ponderomotive Interaction

To calculate the effect of the light field on the electron state, we need to calculate the induced momentum kick and translate it into a change of θ_x and θ_y.

Classical Momentum Kick For ray-tracing a classical ansatz is used to calculate the laser-electron interaction. In contrast to the quantum mechanical derivation in section 2.1, where the quantum mechanical photon-electron interaction potential was used, the classical Hamiltonian for a relativistic charged particle in an electromagnetic field is used. This Hamiltonian can be written as [HK]

$$\mathcal{H}(\boldsymbol{r}, t) = c\sqrt{m_e^2 c^2 + \big(\boldsymbol{p} - e\boldsymbol{A}(\boldsymbol{r}, t)\big)^2} + e\phi, \tag{3.3}$$

where $\boldsymbol{p}$ is the canonical momentum, $\boldsymbol{A}$ is the magnetic vector potential and ϕ is the electric potential.

The Hamilton equations of motion

$$\frac{\mathrm{d}q_i}{\mathrm{d}t} = \frac{\partial \mathcal{H}}{\partial p_i}, \qquad \frac{\mathrm{d}p_i}{\mathrm{d}t} = -\frac{\partial \mathcal{H}}{\partial q_i} \tag{3.4}$$

can be used to calculate the temporal change of position $\boldsymbol{q}$ and momentum $\boldsymbol{p}$. For the derivation of the ponderomotive deflection we are only interested in the change of momentum $\mathrm{d}\boldsymbol{p}$ from the second equation. As we assume that the interaction takes place instantly, the change in position $\mathrm{d}\boldsymbol{q}$ can be ignored. Thus we can write:

$$\mathrm{d}p_i = -\frac{\partial \mathcal{H}}{\partial q_i}\mathrm{d}t. \tag{3.5}$$

When we assume the gauge $\phi = 0$, the only thing that is dependent on the position $\boldsymbol{q}$ is the vector potential $\boldsymbol{A}$. The derivative of the Hamiltonian with respect to the position can in general be calculated to

$$\frac{\partial \mathcal{H}}{\partial q_i} = \frac{c\left(-ep_i\dfrac{\partial A_i}{\partial q_i} + \dfrac{e^2}{2}\dfrac{\partial A_i^2}{\partial q_i}\right)}{\sqrt{m_e^2 c^2 + \big(\boldsymbol{p} - e\boldsymbol{A}\big)^2}}. \tag{3.6}$$

For further calculations, a definition of the magnetic vector potential is needed. For consistency reasons, the same vector potential will be chosen as in section 2.1, where the

potential is defined in equation (2.8). In this calculation, we will consider the electron rest frame, so the potential is

$$\boldsymbol{A}(\boldsymbol{r},t) = -\frac{\mathcal{E}_0'}{\omega_L'}\hat{\boldsymbol{e}}_x g(x,y)u(t)\sin\left(\omega_L' t\right), \tag{3.7}$$

where ω_L' is the angular frequency of the laser in the rest frame and $\mathcal{E}_0'$ is the vector field amplitude. The conversion from the electron rest frame to the lab frame can be calculated using the Lorentz transformation [Nol] where $\omega_L' = \omega_L\sqrt{1\mp\beta}/\sqrt{1\pm\beta}$ with $\beta = v/c$. The upper sign is used for co- propagation and the lower sign for counter-propagation of the electrons and the laser beam. Similar to equation (2.8) the transverse field amplitude distribution is given by $g(x,y)$ and the temporal envelope function is $u(t)$. The linear polarization is chosen, without loss of generality, to be in the x-direction, which is indicated by the polarization unit vector $\hat{\boldsymbol{e}}_x$. Inserting the vector potential (3.7) into (3.6) and then into the Hamilton equation (3.5) gives the momentum change induced by an electromagnetic field

$$\Delta\boldsymbol{p}_i = \frac{ec\mathcal{E}_0'}{\omega_L'}\,p_i\hat{e}_x\frac{\partial g(\boldsymbol{q})}{\partial q_i}\underbrace{\int_{-\infty}^{\infty}\frac{u(t)\sin(\omega_L' t)}{\sqrt{m_e^2c^2+\left(\boldsymbol{p}-e\boldsymbol{A}\right)^2}}\mathrm{d}t}_{\approx 0}$$
$$-\frac{e^2c\mathcal{E}_0'^2}{2\omega_L'^2}\frac{\partial g^2(\boldsymbol{q})}{\partial q_i}\int_{-\infty}^{\infty}\frac{u^2(t)\sin^2(\omega_L' t)}{\sqrt{m_e^2c^2+\left(\boldsymbol{p}-e\boldsymbol{A}\right)^2}}\mathrm{d}t.$$

It was assumed that the envelope function $u(t)$ contains many oscillations of the $\sin(\omega_L' t)$. The first integral in the above equation thus yields zero and can be neglected. In the second therm we replace $\sin^2(x)$ with $(1-\cos(2x))/2$ in the second integral give

$$\Delta p_i \approx -\frac{e^2c\mathcal{E}_0'^2}{4\omega_L'^2}\frac{\partial g^2(x,y)}{\partial q_i}\left[\int_{-\infty}^{\infty}\frac{u^2(t)}{c\sqrt{m_e^2c^2+\left(\boldsymbol{p}-e\boldsymbol{A}\right)^2}}\mathrm{d}t + \underbrace{\int_{-\infty}^{\infty}\frac{u^2(t)\cos(2\omega_L' t)}{\sqrt{m_e^2c^2+\left(\boldsymbol{p}-e\boldsymbol{A}\right)^2}}\mathrm{d}t}_{\approx 0}\right],$$

where we use the same argument as before to cancel out the integral over the product of the cosine with the envelope function. To solve the resulting integral we can use a Taylor expansion of the denominator

$$\sqrt{m_e^2c^2+\left(\boldsymbol{p}-e\boldsymbol{A}\right)^2} = m_ec\sqrt{1+\frac{\left(\boldsymbol{p}-e\boldsymbol{A}\right)^2}{m_e^2c^2}} = m_ec + \frac{\left(\boldsymbol{p}-e\boldsymbol{A}\right)^2}{2m_e^2c^2} - \ldots,$$

which is valid for $(\boldsymbol{p}-e\boldsymbol{A})^2 \ll m_e^2c^2$. Multiplying both sides with c^2 we can see that the condition can be interpreted as that the kinetic energy of the electron must be a lot smaller than the rest energy ($T \ll E_0$). Inserting the Taylor expansion in the momentum formula and rewriting it in vector notation gives

$$\Delta\boldsymbol{p}(x,y) \approx -\frac{e^2\mathcal{E}_0'^2}{4m_e\omega_L'^2}\nabla g^2(x,y)\int_{-\infty}^{\infty}u^2(t)\,\mathrm{d}t. \tag{3.8}$$

We see that the momentum change is proportional to the spatial gradient of the total energy in the light pulse and inversely proportional to the squared laser frequency in the rest frame of the electron. In the following, we will express $\mathcal{E}_0'^2$ with the energy of the light pulse as measurable in the lab frame. For this, the total energy of each laser pulse has to be calculated, which can be done by integrating over the absolute value of the Poynting vector. This calculation is similar to the one in section 2.1, but now performed in the rest frame of the electron. The Poynting vector is defined as [PW]

$$\boldsymbol{S} = \boldsymbol{E} \times \frac{\boldsymbol{B}}{\mu_0}$$

and integrating it over time and space

$$E_L' = \iint_{-\infty}^{\infty} |\boldsymbol{S}| \, \mathrm{d}x\mathrm{d}y\mathrm{d}t \tag{3.9}$$

gives the pulse energy E_L'. To evaluate the Poynting vector, first, a relation between the electric and magnetic field and the vector potential has to be found. Using the gauge condition $\phi = 0$ the electric field can be written as

$$\boldsymbol{E} = -\frac{\partial \boldsymbol{A}}{\partial t}.$$

According to Maxwell's equations, the magnetic field is coupled to the electric field by Faraday's law [PW] and can be written as

$$\boldsymbol{B} = \frac{1}{\omega_L'} \boldsymbol{k}' \times \boldsymbol{E} = -\frac{1}{\omega} \boldsymbol{k}' \times \frac{\partial \boldsymbol{A}}{\partial t},$$

where $\boldsymbol{k}$ is the wave vector of the light field. Inserting the electric and magnetic field into the Poynting vector gives

$$\boldsymbol{S} = \frac{c^2 \varepsilon_0}{\omega_L'} \left[\frac{\partial \boldsymbol{A}}{\partial t} \times \left(\boldsymbol{k}' \times \frac{\partial \boldsymbol{A}}{\partial t} \right) \right], \tag{3.10}$$

where the relation $c^{-2} = \mu_0 \varepsilon_0$ was used.

Setting the propagation direction of the laser field to be the z-direction reduces the wave vector to $\boldsymbol{k}' = \hat{\boldsymbol{e}}_z k'$ where $\hat{\boldsymbol{e}}_z$ is a unit vector in z-direction. Inserting the vector potential (3.7) into equation (3.10) gives

$$\boldsymbol{S} = c^2 \varepsilon_0 \mathcal{E}_0'^2 \frac{k'}{\omega_L'} \left[\hat{\boldsymbol{e}}_x \times \left(\hat{\boldsymbol{e}}_z \times \hat{\boldsymbol{e}}_x \right) \right] g^2(x,y) u^2(t) \cos^2(\omega_L' t),$$

where it was assumed that the envelope function $u(t)$ is changing in larger timescales than the field oscillations $2\pi/\omega_L'$. Using that $c = \omega_L'/k'$ the Poynting vector can be written as

$$\boldsymbol{S} = c \varepsilon_0 \mathcal{E}_0'^2 \hat{\boldsymbol{e}}_z g^2(x,y) u^2(t) \cos^2(\omega_L' t).$$

Inserting the Poynting vector in the light pulse energy definition in equation (3.9) gives

$$E'_L = \iint_{-\infty}^{\infty} c\varepsilon_0 \mathcal{E}_0'^2 g^2(x,y) u^2(t) \cos^2(\omega'_L t) \mathrm{d}x\mathrm{d}y\mathrm{d}t$$
$$\implies \quad \mathcal{E}_0'^2 = \frac{2E'_L}{c\varepsilon_0} \left[\int_{-\infty}^{\infty} g^2(x,y)\mathrm{d}x\mathrm{d}y \int_{-\infty}^{\infty} u^2(t)\mathrm{d}t \right]^{-1}.$$

Replacing the electric field amplitude $\mathcal{E}_0'^2$ in equation (3.8) gives:

$$\Delta \boldsymbol{p} \approx -\frac{e^2 E'_L}{2m_e c\omega'^2_L \varepsilon_0} \frac{\nabla g^2(x,y)}{\int_{-\infty}^{\infty} g^2(x,y)\mathrm{d}x\mathrm{d}y}.$$

Using the definition of the fine structure constant $\alpha = e^2/(4\pi\varepsilon_0\hbar c)$, the relation $\omega'_L = 2\pi c/\lambda'_L$ and the definition of the electron rest energy $E'_e = m_e c^2$ this equation can be rewritten to

$$\Delta \boldsymbol{p} \approx -\frac{\alpha}{2\pi} \frac{E'_L}{E'_e} \frac{\lambda'^2_L}{\int_{-\infty}^{\infty} g^2(x,y)\mathrm{d}x\mathrm{d}y} \hbar \nabla g^2(x,y). \tag{3.11}$$

To transform equation (3.11) from the electron rest frame to the lab frame, we make use of the Lorentz transformation. The electron energy changes according to $E_e = \gamma E'_e$, while for the light field we have the relations [Nol]

$$E'_L = E_L \sqrt{\frac{1 \mp \beta}{1 \pm \beta}} \quad \text{and}$$
$$\lambda'_L = \lambda_L \sqrt{\frac{1 \pm \beta}{1 \mp \beta}}.$$

In order to rewrite equation (3.11) in the lab frame, the primed quantities have to be replaced with the un-primed quantities defined prior. The momentum kick in the lab frame can thus be written as

$$\Delta \boldsymbol{p} \approx -\frac{\alpha}{2\pi(1 \mp \beta)} \frac{E_L}{E_e} \frac{\lambda_L^2 \hbar \nabla g^2(x,y)}{\int_{-\infty}^{\infty} g^2(x,y)\mathrm{d}x\mathrm{d}y}. \tag{3.12}$$

Considering the relation between the force and the potential used above $\boldsymbol{F} = -\nabla V$, the connection between momentum kick and force $\Delta \boldsymbol{p} = \int \boldsymbol{F}\mathrm{d}t$ as well as $\phi = 1/\hbar \int V \mathrm{d}t$ a relation between the classical momentum kick and quantum mechanical phase $\Delta \boldsymbol{p} = \hbar \nabla \phi$ can be constructed. Applying this formula to equation (2.13) will then directly yield the classically derived equation (3.12).

Electron Deflection The electron beam angles of the incoming beam can be written as θ_x^{in} and θ_y^{in}. These angles can be used to define the momentum components of the electron beam using trigonometric functions:

$$p_x = p_z \tan\theta_x^{\text{in}}$$
$$p_y = p_z \tan\theta_y^{\text{in}}$$

The longitudinal component can then be derived from the total momentum and the angles from each electron trajectory

$$\begin{aligned} p^{\text{in}} &= \sqrt{p_x^2 + p_y^2 + p_z^2} \\ &= p_z \underbrace{\sqrt{1 + \tan^2\theta_x^{\text{in}} + \tan^2\theta_y^{\text{in}}}}_{\mathcal{C}}. \end{aligned} \tag{3.13}$$

The momentum after the ponderomotive interaction can be defined as

$$p_x^{\text{out}} = p_x + \Delta p_x$$
$$p_y^{\text{out}} = p_y + \Delta p_y$$
$$p_z^{\text{out}} = p_z + \Delta p_z$$

with which the total outgoing momentum can be defined as

$$p^{\text{out}} = \sqrt{(p_x + \Delta p_x)^2 + (p_y + \Delta p_y)^2 + (p_z + \Delta p_z)^2}.$$

Because there is no absorption of the involved photons, this must be an elastic interaction, which means that the total momentum must be conserved ($p^{\text{in}} = p^{\text{out}}$), which then gives us

$$p_z\mathcal{C} = \sqrt{(p_x + \Delta p_x)^2 + (p_y + \Delta p_y)^2 + (p_z + \Delta p_z)^2}, \tag{3.14}$$

where the transverse momentum kick (Δp_x and Δp_y) is defined in equation (3.12), and Δp_z is the longitudinal momentum change. We can now rewrite equation (3.14) to yield the total momentum in the z-direction after the electron-light interaction.

$$p_z + \Delta p_z = \sqrt{p_z^2\mathcal{C}^2 - (p_x + \Delta p_x)^2 - (p_y + \Delta p_y)^2}$$

With the outgoing momentum $p_z + \Delta p_z$ we now have everything to define the new outgoing angles θ_x^{out} and θ_y^{out} as

$$\tan\theta_x^{\text{out}} = \frac{p_x + \Delta p_x}{p_z + \Delta p_z}$$
$$\tan\theta_y^{\text{out}} = \frac{p_y + \Delta p_y}{p_z + \Delta p_z}$$

which yields

$$\theta_x^{\text{out}} = \arctan\left(\frac{p_x + \Delta p_x}{\sqrt{p_z^2\mathcal{C}^2 - (p_x + \Delta p_x)^2 - (p_y + \Delta p_y)^2}}\right) \tag{3.15}$$

$$\theta_y^{\text{out}} = \arctan\left(\frac{p_y + \Delta p_y}{\sqrt{p_z^2\mathcal{C}^2 - (p_x + \Delta p_x)^2 - (p_y + \Delta p_y)^2}}\right). \tag{3.16}$$

State Change In the case that $p_z \gg p_x, p_y$, which for a simulation in an electron microscope is given, the denominator on equations (3.15) and (3.16) can be written to be $p_z\mathcal{C}$. Using a Taylor expansion for the definition of $\mathcal{C}$ we can linearize it to be $\mathcal{C} \approx 1$.

Under the assumption that after the interaction the momentum component is still much larger than the transverse components, the tan can be dropped and the new angles can be written as

$$\theta_x^{\text{out}} = \frac{p_x + \Delta p_x}{p_z}$$
$$\theta_y^{\text{out}} = \frac{p_y + \Delta p_y}{p_z}.$$

As the momentum change is directly proportional to the light intensity distribution, which as a function of x and y, is in general not linear, it is not possible to construct an ABCD-Matrix for the general case. This means that in order to simulate the state change simply the old angles $\theta_{x,y}^{\text{in}}$ will be replaced with the new angles $\theta_{x,y}^{\text{out}}$.

3.2. Implementation

For the implementation of the ray tracing the programming language Julia was chosen [BEKS]. The multiple dispatch paradigm makes a modular implementation of the different optical elements, described in section 3.1, straightforward to implement. This allows the code to not only run the setup described in figure 1.2, but also an arbitrary combination of these elements.

The main idea is that the simulated image is formed by electron rays that are randomly distributed in a sample region. Random sampling is used to avoid getting squared patterns in the image due to the equidistant spacing in the incoming plane. After the different electron rays are generated they will be subjected to the optical elements that were described in the section before. In the end, an image is formed that will correspond to the ray approximation of the electron distribution at that plane.

A thorough description of how the calculation is implemented in the program follows in the next subsections. The complete code is in appendix A.2.1 and also on `https://github.com/JuffmannLab/Raytracing` available.

3.2.1. Ray generation

As discussed in section 3.1.1, we will describe each electron with a vector consisting of the two position arguments x and y and the two angles θ_x and θ_y. To realistically describe

the electron beam we will add the `intensity` which describes the electron intensity in the input plane. With this `intensity` component the out-coming picture will not be a simulation of how many counts are at which position, but a metric for how probable it is that an electron will hit the pixel at which it is shown.

index	ψ	`intensity`
1	$(x_1, y_1, \theta_{x,1}, \theta_{y,1})$	I_1
2	$(x_2, y_2, \theta_{x,2}, \theta_{y,2})$	I_2
$\vdots$	$\vdots$	$\vdots$
$n-1$	$(x_{n-1}, y_{n-1}, \theta_{x,n-1}, \theta_{y,n-1})$	I_{n-1}
n	$(x_n, y_n, \theta_{x,n}, \theta_{y,n})$	I_n

Table 3.1.: This table shows the information that is saved for each of the n electrons. Each electron is identified with the index and has in the position and travelling direction encoded in the ψ parameter. The `intensity` parameter encodes the probability of this trajectory.

In table 3.1 the logical implementation is illustrated, where the index represents the different electrons, ψ the state of the electron, and `intensity` the input intensity attributed to the position x_i and y_i.

```
mutable struct Electron <: AbstractElectron
    ψ::Vector{Array}
    intensity::Vector{<:Real}
    v::Real
    x::Vector{<:Real}
    y::Vector{<:Real}
end
```

Listing 1: The `Electron` struct has all the information that defines the electrons.

The structure of table 3.1 is implemented in Julia which is shown in listing 1. Here a `struct` called `Electron` is created that saves all the relevant information about the electrons. Additionally the incoming electron velocity `v` and the coordinate axis (`x`, `y`) are saved. In this implementation, we assume that the velocity of the electrons does not change after the ponderomotive interaction which is strictly speaking not true, but works out for all settings where $p_z \gg \Delta p_x, \Delta p_y$. In the initialization of the `Electron` struct we will assign each electron random input positions x and y as well as input angles equal to zero. This means that the standard electron beam is always collimated, but can be brought to a diverging or converging state by applying a lens before the propagation. After the generation of the electron beam, the `Electron` struct is created and returned to the user. The `Electron` struct is essential for each calculation step, as it saves the current state of the simulation.

The initialization code for the `Electron` struct is shown in appendix A.2.1 in the "electron.jl" file.

3.2.2. Light Field description

As we want to simulate the interaction between electrons and a light field, we also need to implement the light field itself. Following the derivation in 2.1 and 3.1.3, the light field can be described as a two-dimensional intensity distribution in the plane $z = 0$. This can easily be achieved by a matrix `intensity` with the coordinate system `x` and `y`.

```
struct LightField <: AbstractLight
    intensity::Matrix{<:Real}
    norm::Real
    x::Vector{<:Real}
    y::Vector{<:Real}
    λ::Real
    E::Real
end
```

Listing 2: The `LightField` struct saves all the relevant information about a laser pulse for the ponderomotive interaction.

In listing 2 these three main components of the light field can be seen implemented in the `LightField` struct. Additionally, the wavelength λ and the energy per pulse `E` are implemented.

3.2.3. Components

One goal of the implementation is that the program is as flexible and modular as possible. This can be achieved by conducting the calculation for each optical component (free propagation, lens, and ponderomotive deflection) in one specific function which we call `calculate!`.

In listing 3 we collect each component in order of their appearance (line 5) to the `Setup` struct (line 1), and use it to calculate the propagation through the setup (line 7). Because of the multiple dispatch paradigm in Julia, the correct method for the function will be determined by the type of the input parameters [BEKS], we can calculate the propagation of the electrons by just iterating over all components and calling `calculate!` (line 8).

```
1 struct Setup
2     setup::Vector{<:Component}
3 end
4
5 Setup(comps::Component...) = Setup(collect(comps))
6
```

```
7  function propagation!(rays::AbstractElectron, setup::Setup)
8      for component in setup.setup
9          calculate!(rays, component)
10     end
11 end
```

Listing 3: The constructor for the setup type can be seen in line 5. The `propagation!` function will then iterate over the setup and apply the calculations to the electron rays.

In the following we will describe all the components with obligatory struct and `calculate!` functions.

Free propagation

To calculate the free propagation of the electron, the ABCD-Matrix for the free propagation (3.1) is multiplied to each electron state ψ. In order for the program to work, two things are essential: The `Electron` struct and the desired propagation distance `d`. First a `Free` struct is created that saves the propagation matrix. The second part, the calculation of the matrix product, will take place when the whole setup is prepared and ready. As can be seen in listing 4, the only thing that is calculated is the matrix product between the state ψ and the propagation matrix that is saved in `Free`.

```
1  struct Free <: Component
2      mat::Matrix{<:Real}
3  end
4
5  Free(d::Real)::Free = Free([1. 0. d 0.; 0. 1. 0. d; 0. 0. 1. 0.; 0. 0. 0. 1.])
6
7  function calculate!(rays::Electron, free::Free)
8      for ray in rays.ψ
9          ray[:] = free.mat * ray
10     end
11 end
```

Listing 4: The creation of the `Free` struct can be seen in line 5, and the application of the propagation matrix for each electron starting in line 8.

Lens calculation

The calculation of a lens is very similar to free propagation. The main difference is that the propagation matrix is equation (3.2) and that instead of the propagation distance, the focal length `f` of the lens has to be given.

The code that is responsible for handling the lenses can be seen in listing 5.

```
1 struct Lens <: Component
2     mat::Matrix{<:Real}
3 end
4
5 Lens(f::Real)::Lens = Lens([1. 0. 0. 0.; 0. 1. 0. 0.; -1/f 0. 1. 0.; 0. -1/f 0.
  ↪  1.])
6
7 function calculate!(rays::Electron, lens::Lens)
8     for ray in rays.ψ
9         ray[:] = lens.mat * ray
10    end
11 end
```

Listing 5: This listing describes and shows how a lens is calculated. The `Lens` struct is created in line 5 and the lens matrix is applied to each electron starting in line 8.

Ponderomotive Deflection

As described in section 3.1.3, the case for the ponderomotive interaction is more difficult than just applying an ABCD-Matrix to the electron state. For each electron, the propagation angles θ_x and θ_y have to be replaced with the new propagation angles that are calculated in equation (3.15) and equation (3.16), respectively.

```
1 struct PondInteraction <: Component
2     lf::AbstractLight
3 end
4
5 function calculate!(ray::Electron, pond::PondInteraction)
6     # Calculation of each constant is left out for simplicity
7     lf = pond.lf
8     constant = - 1 / (1 + β) * αhbar / (2 * π) * lf.E / E_e * lf.λ^2 / lf.norm
9     gradx, grady = _gradient(lf.intensity, lf.x, lf.y)
10
11    for i in eachindex(ray.ψ)
12        Δpx = constant * _interpolation(gradx, lf.x, lf.y, ray.ψ[i][1],
          ↪  ray.ψ[i][2])
13        Δpy = constant * _interpolation(grady, lf.x, lf.y, ray.ψ[i][1],
          ↪  ray.ψ[i][2])
14
15        tan_α = tan(ray.ψ[i][3])
16        tan_β = tan(ray.ψ[i][4])
```

```
17          A = sqrt(1 + tan_α^2 + tan_β^2)
18
19          pz = p / A
20          px = pz * tan_α
21          py = pz * tan_β
22
23          ray.ψ[i][3] = atan((px + Δpx) / real(sqrt(complex(pz^2 * A^2 - (px +
            ↪  Δpx)^2 - (py + Δpy)^2))))
24          ray.ψ[i][4] = atan((py + Δpy) / real(sqrt(complex(pz^2 * A^2 - (px +
            ↪  Δpx)^2 - (py + Δpy)^2))))
25      end
26  end
```

Listing 6: This listing describes `PondInteraction` struct and the `calculate!` function to simulate the deflection of the electrons. It should be noted that to make it easier to read, definitions of physical constants are not shown here.

In listing 6 we show a simplified version of the code that is responsible for the calculation of the deflection angle, the full code is shown in the appendix A.2.1 in "pondInteraction.jl". In line 1 we define the `PondInteraction` struct, which requires a `LightField` struct. The important part of the calculation starts in line 9, where the gradients in x and in y-directions are calculated. Following line 11 each electron will get a momentum kick that will be interpolated to the current position of the electron. This momentum kick then is computed into the new angles θ_x and θ_y that are defined in lines 23 and 24 according to equations (3.15) and (3.16).

3.2.4. Image Acquisition

After the propagation through the setup, the interpretation of the simulation is the next key point. As the output data is just a vector of electron states and corresponding intensities, it is not possible for a human to directly interpret the data and we need a way to visualize it. This is done by mapping all the incoming beams to a given matrix and adding up the intensities in each pixel of the matrix.

```
1  function getintensity(ray::Electron, x::Vector{<:Real},
   ↪  y::Vector{<:Real})::Matrix{<:Real}
2      intensity = zeros(Float64, size(x, 1), size(y, 1))
3      Δx = abs(x[1]-x[2])
4      Δy = abs(y[1]-y[2])
5
6      for i in eachindex(ray.ψ)
7          n = round(Int, (ray.ψ[i][1]-x[1]) / Δx, RoundDown) + 1
8          m = round(Int, (ray.ψ[i][2]-y[1]) / Δy, RoundDown) + 1
9
10         if 0 < n < size(intensity, 1) + 1 && 0 < m < size(intensity, 2) + 1
```

```
11              intensity[n, m] += ray.intensity[i]
12          else
13          end
14      end
15
16      return intensity
17  end
```

Listing 7: This function is the implementation of the electron mapping onto a given field of view. Electrons that happen to be outside of the field of view will be discarded and do not contribute to the image.

In listing 7, the mapping of the electrons in `ray` is mapped onto an image that has the field of view that is described by `x` and `y`. Line 6 iterates over each electron and in lines 7 and 8 for each electron the position in comparison to `x` and `y` is calculated. If these positions `n` and `m` are inside the field of view (line 10), the intensity is added to the `intensity` matrix.

To reflect the physical reality, which in [MWS$^+$] is the measurement with a multichannel plate (MCP), the point spread function (PSF) of the MCP has to be taken into account. To get a realistic image, the `intensity` matrix has to be convolved with the point spread function.

```
1 function mcp(intensity::Matrix{<:Real}, psf::Matrix{<:Real})::Matrix{<:Real}
2     INTENSITY = fft(intensity)
3     PSF = fft(psf)
4     PSF .*= INTENSITY
5     return real.(ifftshift(ifft(PSF)))
6 end
```

Listing 8: This code snippet shows the convolution of the `intensity` and the point spread function `psf` using the convolution theorem.

After the point spread function is applied to the `intensity` matrix, it can be compared to a measurement with an instrument that has the same point spread function.

3.3. Results

3.3.1. Convex Lenses

In this section, we will test the capabilities of the previously described laser-induced convex lens in the ray optics picture. The laser intensity distributions responsible for these imprints can be seen in equations (2.19) and (2.23).

A sketch of the simulated setup is shown in figure 3.2. It consists of a collimated input electron beam, then an amplitude sample. After the sample the beam will hit the convex lens (Note that the beam is still collimated when reaching the lens, due to the ballistic description of the electrons.) and is magnified onto the detector. The focal length f is chosen to be $f = 1\text{cm}$ for all lenses, and the distance between the lens and the detector is 50cm. Note that the image at the detector will always be a mirror image of the probed sample. To avoid confusion, the result of the simulations are mirrored when presented in this thesis.

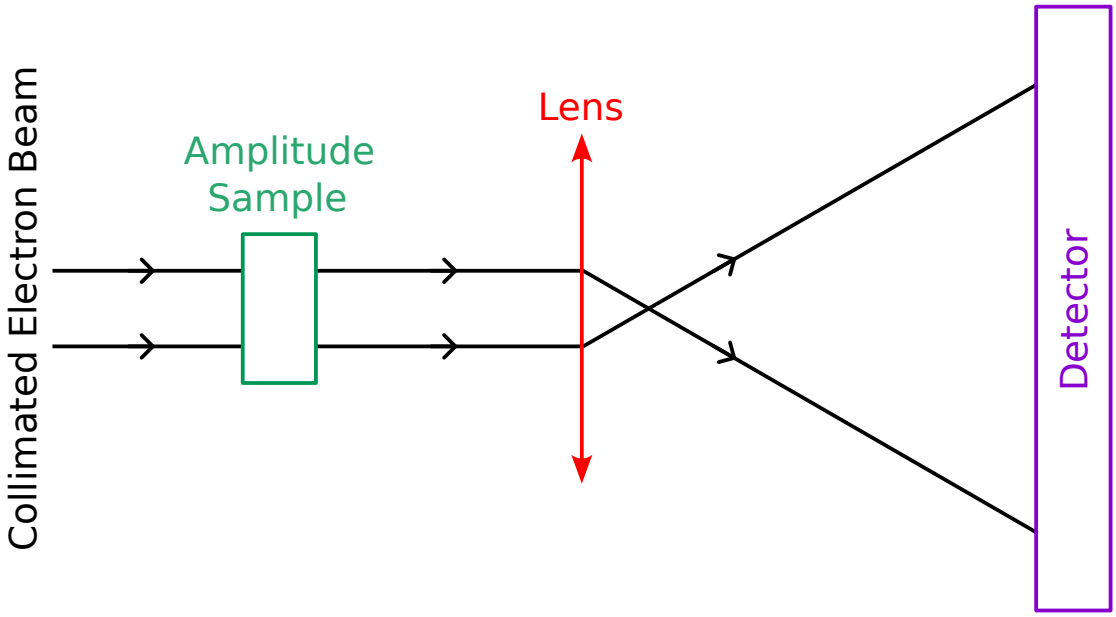


Figure 3.2.: This sketch represents the simulated situation where a collimated electron beam interacts with a sample that only alters the amplitude and not the phase. Afterward, an objective lens is used to enlarge the image. The lens will be simulated as the interaction of the electrons with a specifically shaped light pulse. After the propagation from the lens to the detector, the intensity distribution of the electron beam is evaluated.

In figure 3.3 (a) the input intensity distribution after the interaction with the sample can be seen. The diameter of the electron beam is 15μm and the acceleration voltage is 30keV. In (b) of figure 3.3 a simulation is shown with a perfect electron lens at the detection region. This simulation uses, same as all the following simulations, 50 million electrons to shape the image. This image is a perfect image without any aberrations or other distortions and will be a benchmark against the ponderomotive lenses.

Figure 3.3 (c) shows the intensity distribution using the LG10 lens from section 2.2.1. The configuration of the lens is chosen such that the focal length is $f = 1\text{cm}$, and the total pulse energy is $E_L = 1.28\text{mJ}$. With these constraints and the equation for the focal length of the LG10 lens (2.26), the beam width can be calculated to be

$$w_0 \approx \left[\frac{2\alpha\lambda_L^2\lambda_e f E_L}{(1+\beta)\pi^3 E_e}\right]^{\frac{1}{4}} = 25\mu\text{m},$$

where $\beta = v/c$ with v being the electron velocity, α is the fine structure constant, λ_L and λ_e are the laser and electron wavelength, respectively, and E_e is the relativistic electron energy. In plot 3.3 (c) we obtain an image that deviates significantly from the ideal image. This is expected, since the effective lens size (see section 2.2) is $\rho_{\text{eff}} = 0.2295 w_0 = 5.73\mu\text{m}$, which is smaller than the radius of the incoming beam.

In plot (d) of figure 3.3, the electron distribution at the detector for a modified LG10 lens is shown. The modified version is a superposition of the LG10 beam with an LG20 beam, which increases the effective lens radius. The laser intensity profile that is needed to induce the lens phase shift is written in equation (2.23). The beam radius for a given focal length and laser pulse energy can then be derived from equation (2.28)

$$w_0 \approx \left[\frac{2\alpha\lambda_L^2\lambda_e f E_L}{3((1+\beta)\pi^3 E_e}\right]^{\frac{1}{4}} = 19\mu\text{m}.$$

For the modified LG lens the effective lens size $\rho_{\text{eff}} = 0.5157 w_0 = 9.8\mu\text{m}$, so for a electron beam radius of $7.5\mu\text{m}$ the image should be relatively aberration free. When looking at figure 3.3 (d) we can clearly see a better image compared to (c), despite using the same laser pulse energy. The electron pattern is still aberrated, which can be seen both in the size difference between images (b) and (d) and also in the electron accumulation and slight distortions at the edge of (d).

In summary, this simulation shows that the modified LG lens is much better suited for practical application than the simple LG10 lens. Even though the LG10 lens needs only a third of the pulse energy to form the same focal length as the modified lens, the larger effective lens radius makes it more achievable to construct ponderomotive lenses in the size of typical electron apertures. The jupyter notebook in which the simulation was conducted is provided in appendix A.2.2.

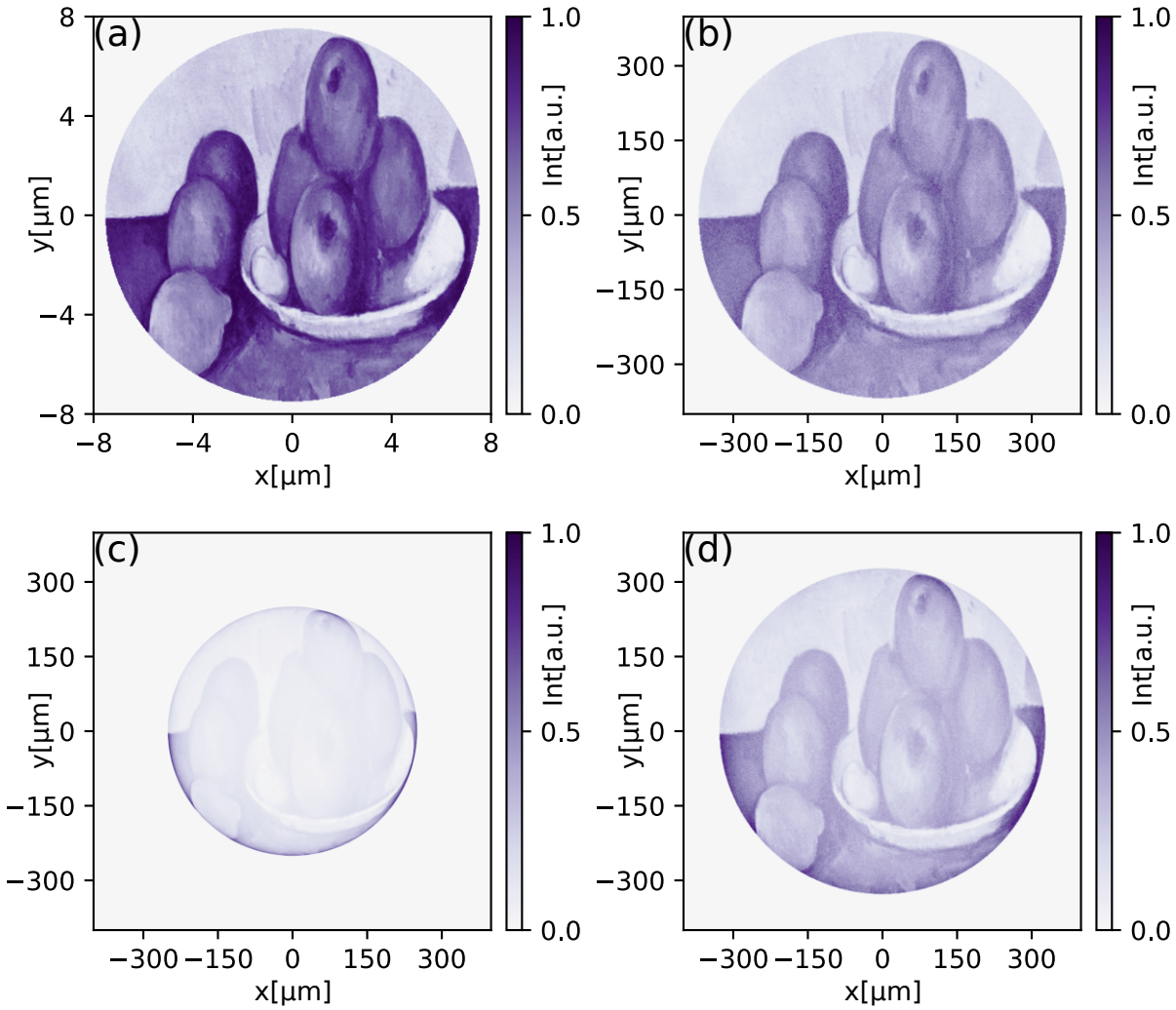


Figure 3.3.: Plot (a) shows the electron probability distribution after a sample that only induces amplitude variations. In plot (b), the image of the sample is shown where the objective lens is perfect. In (c) we can see the image where an LG10 laser beam was used to create the phase shift needed for a lens, with a pulse energy of $E \approx 1.28\text{mJ}$. Aberrations are clearly visible in the image. In (d) the modified LG mode is used to generate the lens, with the same pulse energy of $E \approx 1.28\text{mJ}$, and it can be seen that the aberrations are drastically reduced compared to (c).

4. Wave Optics

In the previous chapter, the modeling of the electrons was purely classical, and thus quantum mechanical effects such as interference are not accounted for. In this chapter, we will present a quantum mechanical approach to the simulation of electron propagation.

For these simulations, we will make use of propagation equations that are typically used to calculate light optics but can also be used for matter-wave propagation. In section 4.1 we will show and elaborate on algorithms that are used to calculate the propagation of waves and discuss the applicability for electron simulations. Following the theory section, we will give a detailed description of the implementation using the Julia programming language 4.2. Lastly, in the results section 4.3 we will show some example wave optics simulations that are then compared with the corresponding ray-tracing simulation.

4.1. Theory

The main difference between a classical and a quantum mechanical description of electrons is that the quantum mechanical electrons are matter waves and not ballistic particles. This means that, compared to the four parameters in section 3.1.1, a wave function $\boldsymbol{\Psi}_{\boldsymbol{p},s}$ is needed for the description. Also, the temporal evolution of the wave function is not governed by Newton's mechanics but by the Dirac equation which makes the calculation considerably more difficult.

In the following sections, we will start with a plane wave solution to the Dirac equation (2.1), and show that with a few approximations it can be rewritten to the scalar Helmholtz equation. Assuming a paraxial electron beam we will introduce the Fresnel approximation which is the starting point for the numerical calculations. We consider three different approaches to discretize the calculation and will discuss the advantages, drawbacks, and sampling conditions of each of these.

After the free propagation calculation, we show the model of optical elements. Especially the lens element is of importance and we will show how to model a lens as well as the sampling conditions for an artifact-free simulation. As the ponderomotive phase imprint on an electron that is discussed in section 2.1 is already quantum mechanical, no further derivation is needed for it.

4.1.1. Free electron propagation

As we are dealing with relativistic electrons, the proper way of describing them is the Dirac equation (2.1). The plane wave solution for the Dirac equation can be written

as [Mes] (Chapter 20)

$$\boldsymbol{\Psi}_{\boldsymbol{p},s}(\boldsymbol{r},t) = \frac{1}{\sqrt{\mathcal{N}}} e^{i(\boldsymbol{p}\cdot\boldsymbol{r}-E_p t)/\hbar} \begin{pmatrix} |s\rangle \\ \frac{c}{E_p+m_e c^2}\boldsymbol{p}\cdot\hat{\boldsymbol{\sigma}}|s\rangle \end{pmatrix}, \tag{4.1}$$

with

$$\begin{aligned} \mathcal{N} &= \sqrt{\frac{E_p + m_e c^2}{2E_p}} \\ E_p &= \sqrt{p^2c^2 + m_e c^4} \\ \hat{\boldsymbol{\sigma}} &= \begin{pmatrix} \hat{\sigma}_x \\ \hat{\sigma}_y \\ \hat{\sigma}_z \end{pmatrix}. \end{aligned}$$

Furthermore $\hat{\boldsymbol{\sigma}}$ are the Pauli matrices, $\boldsymbol{p}$ the electron momentum and $|s\rangle$ the spin-1/2 state.

The Dirac equation would also allow for positronic fields, which would correspond to a negative energy E_p, but those cases will not be considered here, so we restrict ourselves to positive energies $E_p > 0$. Also, as we are describing the free propagation of the electron, the spin state will be conserved. These two facts lead to the possibility of describing the spatial wave packet as a scalar field [MWS$^+$]

$$\boldsymbol{\Psi}_s(\boldsymbol{r},t) = \int \mathrm{d}^3 p c(\boldsymbol{p}) \boldsymbol{\Psi}_{\boldsymbol{p},s}(\boldsymbol{r},t) = \begin{pmatrix} \psi(\boldsymbol{r},t)|s\rangle \\ -i\hat{\boldsymbol{\sigma}}\cdot\nabla\chi(\boldsymbol{r},t)|s\rangle \end{pmatrix},$$

with χ being the positronic and ψ the electronic field component. Because the electronic wave function ψ solves the Dirac equation, it will also solve the Klein-Gordon equation [MWS$^+$]

$$\left(\hbar^2\partial_t^2 - \hbar^2 c^2\nabla^2 + m_e^2 c^4\right)\psi(\boldsymbol{r},t) = 0. \tag{4.2}$$

Considering the electron energy $E_p = E_e = \gamma m_e c^2$ and the quantum mechanical energy operator $\hat{E} = i\hbar\partial_t$, we can replace $\hbar^2\partial_t^2$ with $-\gamma^2 m_e^2 c^4$ in (4.2). With that, the Klein-Gordon equation can be rewritten to

$$\left(\nabla^2 - \frac{m_e^2 c^2(1-\gamma^2)}{\hbar^2}\right)\psi(\boldsymbol{r},t) = 0. \tag{4.3}$$

Conducting a product ansatz for the electron wave function $\psi(\boldsymbol{r},t) = \phi_\psi(\boldsymbol{r})\theta_\psi(t)$ allows us to eliminate the temporal dependency. We now define the time-independent electron wave function to be $\psi(\boldsymbol{r}) = \phi_\psi(\boldsymbol{r})$. Revisiting and simplifying the second term in (4.3) gives

$$\frac{m_e^2 c^2}{\hbar^2}\left(1-\gamma^2\right) = \frac{m_e^2 c^2}{\hbar^2}\left(1 - \frac{1}{1-\beta^2}\right) = \frac{2m_e^2 c^2}{\hbar^2}\beta^2\gamma^2 = \frac{\gamma^2 m_e^2 v^2}{\hbar^2} = k^2,$$

where $k = 2\pi/\lambda$ and $\lambda = h/(\gamma m_e v)$ was used [MWS$^+$].

Applying all this to equation (4.3) gives

$$\left(\nabla^2 - k^2\right)\psi(\boldsymbol{r}) = 0, \tag{4.4}$$

which is the scalar Helmholtz equation.

An analytical solution to the scalar Helmholtz equation (4.4), found by Gustav Kirchhoff in 1887 and called the Fresnel-Kirchhoff integral, has the form of [PW]

$$\psi(x,y,z) = -\frac{i}{\lambda}\iint_{\mathcal{A}} \mathrm{d}x'\mathrm{d}y'\,\psi_0(x',y')\frac{e^{ikR}}{R}\left[\frac{1+\cos\left(\boldsymbol{R},\,\hat{\boldsymbol{e}}_z\right)}{2}\right]$$

with

$$\boldsymbol{R} = \begin{pmatrix} x-x' \\ y-y' \\ z \end{pmatrix},$$

where $\mathcal{A}$ describes the aperture, $R = |\boldsymbol{R}|$ and $\hat{\boldsymbol{e}}_z$ is the unit vector in z-direction. Even for the easiest cases, this integral is difficult to solve analytically and thus approximations come in handy. One of these approximations is the so-called Fresnel approximation and it consists of two parts.

In the first step, we restrict ourselves to small angles with respect to the z-axis. From this consideration we can make the approximation that $\cos(\boldsymbol{R}, \hat{\boldsymbol{e}}_z) \approx 1$. After this step, the Fresnel-Kirchhoff integral is reduced to

$$\psi(x,y,z) \approx -\frac{i}{\lambda}\iint_{\mathcal{A}} \mathrm{d}x'\mathrm{d}y'\,\psi_0(x',y')\frac{e^{ikR}}{R}, \tag{4.5}$$

which is not yet sufficient for analytical considerations [PW].

The second step of the Fresnel approximation is that the r in the denominator of (4.5) is estimated to be $R \approx z$, where we also make use of the limitations to small angles [PW]. In the exponential part, we have to be careful, because a small deviation in R can result in larger changes in e^{ikR}. To resolve this issue a Taylor expansion of R is conducted

$$R = z\sqrt{1 + \frac{(x-x')^2 + (y-y')^2}{z^2}} \approx z\left[1 + \frac{(x-x')^2 + (y-y')^2}{2z^2} - \dots\right]$$

and the first order Taylor expansion is used to replace the R in the exponential term [PW].

Adding all these considerations together gives

$$\psi(x,y,z) \approx \frac{e^{ikz}}{i\lambda z}\iint_{\mathcal{A}} \mathrm{d}x'\mathrm{d}y'\,\psi_0(x',y')\exp\left\{i\frac{k}{2z}\left[(x-x')^2 + (y-y')^2\right]\right\}, \tag{4.6}$$

which is the Fresnel approximation.

4.1.2. Impulse response approach

For a spatially confined input wave function $\psi_0(x, y)$, equation (4.6) can be interpreted as a two-dimensional convolution of the input wave function with the impulse response $h(x, y) = \frac{e^{ikz}}{i\lambda z} \exp\left[\frac{ik}{2z}(x^2 + y^2)\right]$ [VR]:

$$\psi(x, y, z) \approx \big(\psi_0 * h\big)(x, y).$$

Using the convolution theorem the convolution can be rewritten as multiplication in Fourier space

$$\psi(x, y, z) \approx \mathfrak{F}^{-1}\big\{\mathfrak{F}\{\psi_0(x, y)\}\mathfrak{F}\{h(x, y)\}\big\}, \tag{4.7}$$

where $\mathfrak{F}$ and $\mathfrak{F}^{-1}$ represent the two-dimensional Fourier- and inverse Fourier transform operators. The convolution theorem requires that we have a circular convolution. A convolution is called a circular convolution when the two input functions are both periodic with the same periodicity [OSB]. As the two functions ψ_0 and h are not periodic, we would, strictly speaking, not have a circular convolution, and could not have used the convolution theorem in equation (4.7). As the Fourier transform approach inherently assumes that we have periodic input functions, i.e. at certain distances there exist periodic copies, we have to deal with them. We use zero-padding to increase the distance between the periodic copies, and when the distance is large enough, which means that the convolution has no periodic artifacts, the circular convolution is equivalent to the normal convolution. The representation with Fourier transforms has the advantage that, due to the fast Fourier transform algorithm, the numerical calculation of the convolution is now feasible for larger input intensity matrices.

The impulse transfer approach reduces the number of computations that are required for wavefront propagation. While a brute force numeric computation of a discretized version of equation (4.6) is of complexity $\mathcal{O}\big(n^2\big)$, a discrete fast Fourier transform reduce the complexity to $\mathcal{O}\big(3n \log_2(n)\big)$, where n is the number of sample points. The impulse transfer approach can be used to efficiently calculate the diffraction pattern of a light beam, as well as the probability density of an electron.

Sampling Conditions

Up to this point, it is assumed that we operate in a continuous regime. In order to solve the impulse response approach numerically, equation (4.7) and all its components need to be discretized. For the coordinates x and y, this means

$$x = (n_x - 1)\Delta x - \frac{L_x}{2} \quad \text{with} \quad n_x \in \big\{1, 2, \ldots, N_x - 1, N_x\big\} \tag{4.8}$$

$$y = (n_y - 1)\Delta y - \frac{L_y}{2} \quad \text{with} \quad n_y \in \big\{1, 2, \ldots, N_y - 1, N_y\big\}, \tag{4.9}$$

with L_x and L_y being the side lengths, N_x and N_y being the sample points and $\Delta x = L_x/N_x$ and $\Delta y = L_y/N_y$ being the sampling.

With these discrete parameters, it is important that all the features in the impulse response (this function can also be called a chirp function) can be resolved in real and in Fourier space. Writing down the impulse response

$$h(x,y) = \frac{e^{ikz}}{i\lambda z} \exp\left[i\frac{k}{2z}\left(x^2+y^2\right)\right], \tag{4.10}$$

we can see it contains a quadratic phase term in transverse directions. The x and y coordinates have to be chosen in a way that the impulse response can be represented in a non-aliased way in Fourier space. First, we write down the phase term in question

$$\phi_h(x,y) = \frac{k}{2z}\left(x^2+y^2\right). \tag{4.11}$$

According to [VR], the criterion for an unaliased phase in the x-direction can be written as

$$\Delta x \left|\frac{\partial \phi_h}{\partial x}\right|_{\max} \leq \pi$$

with an analog solution for the y-direction. This condition is assuming that the sampling is uniform and states that the phase difference between two neighboring samples has to be smaller than π. From this point on, only the calculation for the x-direction will be shown, as the calculation in the y-direction is analogous. Computing the derivative gives

$$\Delta x \left|\frac{k}{z}x\right|_{\max} = \frac{k\Delta x}{z}|x|_{\max} \leq \pi,$$

where in the second step, it was used that k and z are experimental parameters and thus the only value that can be maximized is x [VR]. Considering the definition of the discrete x values (4.8), we can immediately see that $|x|_{\max} = L_x/2$. Also taking into account the definition of the wave-number $k = 2\pi/\lambda$, the sampling condition can be rewritten to be [VR]

$$\Delta x \leq \frac{\lambda z}{L_x} \tag{4.12}$$

and for the y-direction

$$\Delta y \leq \frac{\lambda z}{L_y}.$$

Up to this point, only the sampling of the impulse response function $h(x,y)$ and not of the input wave function ψ_0 was considered. Note that in equation (4.10) also the input wave has to be sampled correctly. In the case that the sampling conditions are not being followed (which means that we are under-sampled), the impulse response will be aliased, meaning that there will be artifacts in the calculation [VR].

4.1.3. Transfer function approach

When looking at the impulse response approach in equation (4.7) and the impulse response itself in equation (4.10), it can be seen that it is also possible to compute the Fourier transform of the impulse response analytically and insert it into equation (4.7). This is then called the transfer function approach [VR]:

$$\psi(x,y,z) \approx \mathfrak{F}^{-1}\big\{\mathfrak{F}\{\psi_0(x,y)\}H(f_x,f_y)\big\} \tag{4.13}$$

with the transfer function

$$H(f_x,f_y) = \mathfrak{F}\{h(x,y)\} = e^{ikz}\exp\bigg[-i\frac{2\pi^2 z}{k}\big(f_x^2+f_y^2\big)\bigg], \tag{4.14}$$

where the frequency coordinates f_x and f_y are defined as [VR]

$$f_x = (n_x-1)\frac{1}{L_x} - \frac{1}{2\Delta x} \quad \text{with} \quad n_x \in \big\{1,2,\ldots,N_x-1,N_x\big\} \quad \text{and} \tag{4.15}$$

$$f_y = (n_y-1)\frac{1}{L_y} - \frac{1}{2\Delta y} \quad \text{with} \quad n_y \in \big\{1,2,\ldots,N_y-1,N_y\big\}. \tag{4.16}$$

One difference between the transfer function and the impulse response function is that the transfer function uses one less Fourier transform, and thus can be considered slightly faster than the impulse response approach. The second main difference will be visible when we have a look at the sampling considerations.

Sampling Conditions

The sampling considerations in the transfer function differ from the impulse response because here the transfer function is defined in the Fourier space and the sampling must be maintained in the real space.

The sampling in the Fourier space depends on the phase of the transfer function (4.14), where the constant phase term is omitted

$$\phi_H(f_x,f_y) = -\pi\lambda z\big(f_x^2+f_y^2\big).$$

Similar to the case of the impulse response, only the x-direction will be considered, as the y-direction is completely analogous to it. The sampling criterion also can be written in the same way [VR]

$$\Delta f_x \left|\frac{\partial\phi_H}{\partial f_x}\right|_{\max} \leq \pi,$$

where the frequency sampling is $\Delta f_x = 1/L_x$. Calculating the derivation and substituting for Δf_x yields

$$\frac{1}{L_x}\big|-2\pi\lambda z f_x\big|_{\max} = \frac{2\pi\lambda z}{L_x}\big|f_x\big|_{\max} \leq \pi$$

and with the fact that $|f_x|_{\max} = 1/(2\Delta x)$, we can get the sampling conditions

$$\Delta x \geq \frac{\lambda z}{L_x} \quad \text{and} \qquad \Delta y \geq \frac{\lambda z}{L_y}. \tag{4.17}$$

As we can see, the sampling condition of the transfer function approach (4.17) and the sampling condition for the impulse response approach (4.12) are complementary to each other. So for every situation that we can possibly encounter we can either choose one of those two algorithms or adjust the sampling such that either the transfer function or the impulse response algorithm can be used.

4.1.4. Single FFT Approach

Both of the convolution-based algorithms, that were described in the last two sections, have in common that the incoming coordinate system (x' and y') will also be the outgoing coordinate system (x and y). One way to lift this constraint is the single FFT approach that will be described in this section.

Computing all the binomials in the Fresnel approximation (4.6) gives

$$\psi(x,y,z) \approx \frac{e^{ikz}}{i\lambda z} \iint_{\mathcal{A}} \mathrm{d}x'\mathrm{d}y'\, \psi_0(x',y') \exp\left[i\frac{k}{2z}\left(x^2 - 2xx' + x'^2 + y^2 - 2yy' + y'^2\right)\right],$$

which, after separating the exponentials, can be rewritten as

$$\begin{aligned}\psi(x,y,z) \approx -\frac{i}{\lambda z} e^{ikz} \exp\left[i\frac{k}{2z}\left(x^2+y^2\right)\right] \times \\ \iint_{\mathcal{A}} \mathrm{d}x'\mathrm{d}y'\, \psi_0(x',y') \exp\left[i\frac{k}{2z}\left(x'^2+y'^2\right)\right] \exp\left[-i\frac{k}{z}\left(xx'+yy'\right)\right].\end{aligned}$$

The new representation of the Fresnel approximation can be interpreted as a Fourier transform of the input wave function multiplied with an inner chirp function. After the Fourier transforms, a second, outer chirp, function is multiplied. This interpretation can be seen in

$$\psi(x,y,z) \approx F_0\, \mathfrak{F}\left\{\psi_0(x',y')\, F(x',y')\right\} \tag{4.18}$$

with

$$F_0(x,y) = -\frac{i}{\lambda z} e^{ikz} \exp\left[i\frac{k}{2z}\left(x^2+y^2\right)\right] \quad \text{and}$$
$$F(x',y') = \exp\left[i\frac{k}{2z}\left(x'^2+y'^2\right)\right].$$

In equation (4.18), we can see that F_0 consists of an additional phase factor and an constant value, which we can neglect, if we are only interested in the probability density at the z position.

As only one Fourier transform is needed to calculate the output wave function, this approach is the fastest of the ones considered here. Also, as described before, it is not necessarily the case that the input and the output coordinate system will be the same. The coordinate transformation will change depending on how the sampling is chosen and will be discussed in the next section. But first, it should be mentioned that an arbitrary coordinate transformation is possible using the chirp-z-transform. With the Bluestein algorithm, the chirp-z-transform can be expressed as Fourier transforms and thus take advantage of the calculation speed increase ([HWW$^+$], [LL]).

Sampling Conditions

The discrete x and y coordinates are defined by equations (4.8) and (4.9). For the transformed coordinates we assume the side lengths L'_x and L'_y as well as the samplings $\Delta x'$ and $\Delta y'$. Because of the nature of the Fourier transform, the sampling number $N'_x = N_x$ and $N'_y = N_y$ stay the same. The x' and y' coordinates are then also defined by (4.8) and (4.9) with their sampling parameters, respectively.

As the coordinate transformation of the Fourier transform is known, we can thus make a connection between the sampling parameters at the incident plane and the outgoing plane [VR]

$$\Delta x' = \frac{\lambda z}{L_x} \quad \text{and} \tag{4.19}$$

$$L'_x = \frac{\lambda z}{\Delta x}. \tag{4.20}$$

As can be seen in equation (4.18), there are two chirp functions (F_0 and F) with this method.

First, we have a look at the F_0 chirp [VR], where the added phase is

$$\phi_{F_0} = \frac{k}{2z}(x^2 + y^2).$$

As the chirp phase is equivalent to the chirp phase for the impulse response approach in equation (4.11), the sampling condition is the same, namely

$$\Delta x \leq \frac{\lambda z}{L_x}.$$

Using the previous relations between the incoming and outgoing coordinates, the sampling condition can be rewritten to

$$\Delta x' \geq \frac{\lambda z}{L'_x}. \tag{4.21}$$

For the second chirp function F we have a familiar chirp phase

$$\phi_F(x', y') = \frac{k}{2z}(x'^2 + y'^2),$$

which will lead to the same sampling condition, but in the incoming instead of the outgoing coordinate system

$$\Delta x' \leq \frac{\lambda z}{L'_x}. \tag{4.22}$$

Comparing the two sampling conditions for the direct FFT approach (4.21) and (4.22) we can see that the only way both conditions are satisfied is, when [VR]

$$\Delta x' = \frac{\lambda z}{L'_x}.$$

Considering the sampling condition and the coordinate transformations (4.19) and (4.20), we can immediately see that the incoming and outgoing lengths $L_x = L'_x$ as well as the sampling $\Delta x = \Delta x'$ have the same values, and thus $x = x'$ must be true [VR].

If, on the other hand, only the probability density in the outgoing plane is important, the outer chirp F_0 can be ignored, and following that, also the sampling condition for the outer chirp (4.21). In this regime, coordinate changes are possible as long as the sampling condition for the inner chirp (4.22) is not violated [VR].

4.1.5. Lens imprint

The effect that a lens has on an electron wave is already mentioned in chapter 2.2. Here we only consider a perfect lens that is described by a phase shift in equation (2.14). For convenience the equation will be displayed again:

$$\varphi(\rho) = -\frac{\pi}{f\lambda}\rho^2 \tag{4.23}$$

In equation (4.23), f is the focal length of the electron lens, λ is the electron wavelength and ρ are radial coordinates.

As already mentioned before, due to the limitations described by Scherzer [Sch], it is, under certain common assumptions, not possible to construct a perfect electron lens. For the first-order spherical aberrations, a phase deviation from (4.23) can be written as

$$\chi(\theta) = \frac{\pi}{\lambda}\left(\frac{C_s}{2}\theta^4 - \Delta f\theta^2\right), \tag{4.24}$$

where C_s is the spherical aberration coefficient, θ is the semi-angle to the optical axis. In electron microscopy a defocus Δf is often use to partially compensate for spherical aberrations. For example, the Scherzer defocus $\Delta f_S = 0.5 C_s \alpha^2$, with α being the convergence semi-angle, minimizes the aberration at the edge of the aperture [SRL$^+$].

In figure 4.1, a sketch of a lens is shown, where all the before-mentioned parameters are included. The defocus Δf will affect the focal length of the lens and is used mainly in high-resolution transmission electron microscopy to enhance contrast [Kir].

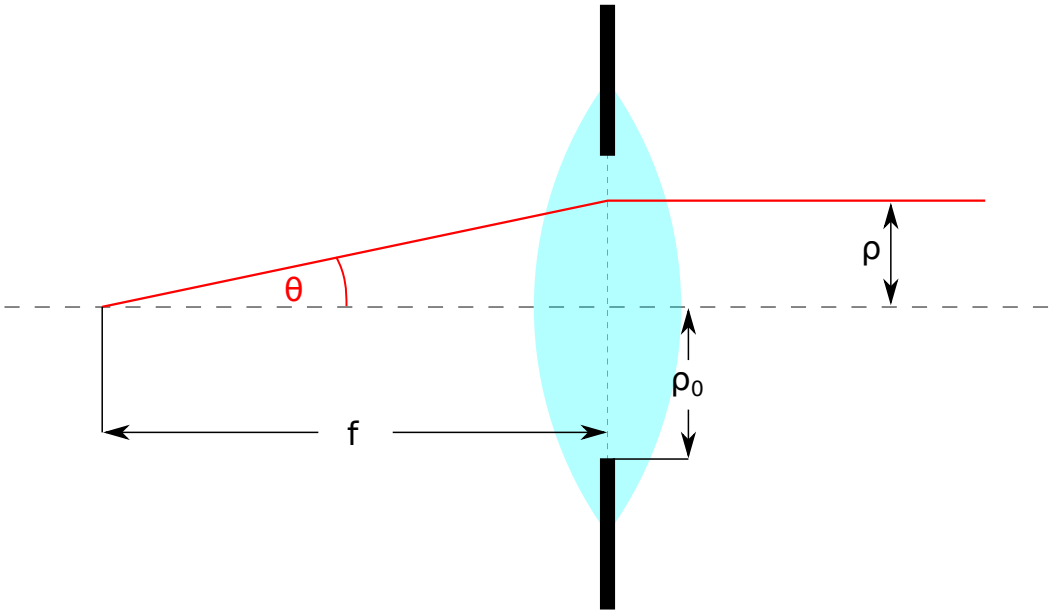


Figure 4.1.: Schematics of the lens, which is displayed as the turquoise blob with a focal length of f. The black bars in the lens are the aperture with the aperture radius ρ_0. The numerical implementation of the lens assumes that all the effects are happening in an instant at the center of the lens, which can be seen as a dashed gray line. From these schematics, the correspondence between the semi-angle θ and the distance to the optical axis ρ can be seen.

From 4.1 we can see that the semi-angle θ can be calculated as $\tan\theta = \rho/f$, which then can be simplified, assuming small angles, to $\theta = \rho/f$. Considering this substitution and creating a new, aberrated phase shift gives

$$\varphi_a(\rho) = -\frac{\pi}{f\lambda}\left(\rho^2 + \frac{\Delta f}{f^2}\rho^2 - \frac{C_s}{2f^4}\rho^4\right). \tag{4.25}$$

For a realistic lens, also the finite size has to be taken into account. In our model, we emulate this fact by multiplying an aperture function

$$A(\rho) = \begin{cases} 1 & \text{if } \rho \leq \rho_0 \\ 0 & \text{else} \end{cases}, \tag{4.26}$$

where ρ_0 is the size of the aperture.

To simulate the effect on an electron beam, we have to multiply the incoming electron wave ψ_{in} with the phase of equation (4.25):

$$\psi_{\text{out}}(\boldsymbol{r}, t) = \psi_{\text{in}}(\boldsymbol{r}, t) A(\rho) e^{i\varphi_a(\rho)}.$$

Sampling condition

Changing from the continuous to the discrete regime will also have to be considered in the case of the simulation of a lens. The main problem is that the phase shift that will be applied is very large, because the electron wavelength λ_e is normally very small (an acceleration voltage of 30keV corresponds to an electron wavelength of $\approx 7 \cdot 10^{-12}$m).

Similar to the free propagation sampling conditions, we will only have a look at the x-direction, as the y-direction is completely analogous. The condition for avoiding aliasing from [VR] gives

$$\Delta x \left| \frac{\partial \varphi_a}{\partial x} \right|_{\max} \leq \pi.$$

Because the lens and the aberration function both are rotationally symmetric, we will change to a one-dimensional Cartesian picture for the sampling, where $x = \rho$. The point $x_{\max}$ describes the point where the phase shift is maximal. To calculate $x_{\max}$ differential calculus was used

$$\begin{aligned} &\frac{\partial}{\partial x} \left| \frac{\partial \varphi_a(x_{1,2})}{\partial x} \right| \equiv 0 \\ \Longrightarrow \quad &x_{1,2} = \pm \sqrt{\frac{f^4}{3C_s} \left(1 + \frac{\Delta f}{f^2} \right)}, \end{aligned}$$

where it was assumed that $1 + \Delta f / f^2 > C_s / f^4 x^2$. This will hold for low NA lenses such as electron lenses. As we are dealing with radially symmetric phases only the positive solution x_1 can be considered. As the aperture radius $x_0 = \rho_0$ is a natural cutoff for the phase, we have to consider the possibility that x_1 is larger than x_0. In this case, the maximum slope of the phase is at the point of the aperture. Thus we can write for $x_{\max}$

$$x_{\max} = \begin{cases} x_1 & \text{if } x_1 < x_0 \\ x_0 & \text{else} \end{cases}.$$

This means that the anti-aliasing condition changes to

$$\begin{aligned} &\Delta x \left| \frac{\partial \varphi_a}{\partial x} \right|_{\max} = \frac{2\pi \Delta x}{f\lambda} \left(x_{\max} + \frac{\Delta f}{f^2} x_{\max} - \frac{C_s}{f^4} x_{\max}^3 \right) \leq \pi \\ \Longrightarrow \quad &\Delta x \leq \frac{f\lambda}{2x_{\max} \left(1 + \frac{\Delta f}{f^2} - \frac{C_s}{f^4} x_{\max}^2 \right)}. \end{aligned} \tag{4.27}$$

4.2. Implementation

The implementation of the wave approach was made using the Julia programming language [BEKS], with the goal to keep the program as modular as possible. As all sampling conditions must match, this computational implementation is more complex than the ray tracing approach. For this reason, the sampling will be tasked to the user of this program, as the program is assuming that everything is sampled correctly.

4.2.1. Electron description

According to the calculations in section 4.1.1, an electron can be described by a complex scalar field. The numerical analog to that would be the construction of a complex matrix ψ. For each matrix, we also need the corresponding coordinate system, which is then described by the vectors `x` and `y`. Again it is assumed that the propagation direction is the positive z-direction, the only unknown is the wavelength of the electron λ. The electron velocity `v` is not directly needed, as the velocity information is also encoded in λ, but it is easier to calculate both the wavelength and the velocity from the acceleration voltage in the constructor of the `ElectronBeam` type.

```
mutable struct ElectronBeam <: Wave
    ψ::Matrix{<:Complex}
    λ::Real
    v::Real
    x::Vector{<:Real}
    y::Vector{<:Real}
end
```

Listing 9: The `ElectronBeam` struct saves the relevant information for the electron beam.

In listing 9 the numerical description of the electrons is shown, where each of the before-mentioned components is saved.

Before the definition of the `ElectronBeam` sampling has to be implemented in the correct way. To help with the sampling condition and the calculation of the de Broglie wavelength the function `deBroglieWavelength` is provided, which automatically will calculate the relativistic electron wavelength for a given acceleration voltage.

4.2.2. Description of light

Based on the considerations made in section 2.1, the relevant properties of the light field is the intensity distribution in the interaction plane and the total pulse energy E and the wavelength λ. The laser beam is thus described by a matrix `I`, which is the interaction plane intensity distribution. Again the coordinate system is described by `x` and `y`.

```
mutable struct LaserBeam
    I::Matrix{<:Real}
    λ::Real
    E::Real
    x::Vector{<:Real}
    y::Vector{<:Real}
end
```

Listing 10: The `LaserBeam` struct saves the information that is needed to describe the ponderomotive interaction.

This means that for our purpose the laser beam can be described by the struct that is written in listing 10. It should be noted that the coordinate system of the laser beam can be independently chosen from the coordinate system of the electron beam, but the sampling should be similar or better compared to the electron sampling to ensure that the phase imprint is accurate.

4.2.3. Components

Similar to the Ray tracing description in chapter 3, each component of the simulation will have a `calculate!` function that is there to alter the `ElectronBeam` struct to represent its state after each component.

```
1 struct Setup
2     setup::Array{<:Component}
3 end
4
5 Setup(comps::Component...) = Setup(collect(comps))
6
7 function propagation!(wave::ElectronBeam, setup::Setup)
8     for comp in setup.setup
9         calculate!(wave, comp)
10    end
11 end
```

Listing 11: Here the `propagation!` function is shown with the corresponding `Setup` type. The for loop in line 8 applies each component onto `wave`.

In listing 11 the `Setup` type is defined. Inside the `Setup` struct an array of `Component` will be saved. The `Component` type is an abstract type that is the supertype of all components. In line 5 of listing 11 a constructor of the `Setup` struct is shown, which will create an array out of all the components that are given when `Setup` is called.

The propagation through the simulated setup is handled in line 8, where each `calculate!` method of the components is executed. After this for-loop the `ElectronBeam` will represent the electron state at the end of the setup.

Free propagation

For the free propagation, the transfer function approach was chosen, because it is easier to sample than the direct FFT approach, and faster than the impulse response approach. As a reminder, the transfer function approach is defined as (4.14)

$$\psi(x, y, z) \approx \mathfrak{F}^{-1}\{\mathfrak{F}\{\psi_0(x, y)\}H(f_x, f_y)\},$$

with the transfer function

$$H(f_x, f_y) = e^{ikz} \exp\left[-i\frac{2\pi^2 z}{k}\left(f_x^2 + f_y^2\right)\right].$$

The e^{ikz} term in the transfer function is a constant global phase, which we can neglect, since we are only interested in the intensities at the detector.

```
1  struct Free <: Component
2      transferfunction::Matrix{<:Complex}
3  end
4
5  function Free(wave::ElectronBeam, distance::Real)::Free
6      x = wave.x
7      y = wave.y
8      λ = wave.λ
9      dx = abs(x[1] - x[2])
10     dy = abs(y[1] - y[2])
11
12     fx = Vector{Float64}(range(-1/(2*dx), 1/(2*dx), length=size(x, 1)))
13     fy = Vector{Float64}(range(-1/(2*dy), 1/(2*dy), length=size(y, 1)))
14
15     transferfunction = @. exp(-1im * π * λ * distance * (fx^2 + fy'^2))
16     return Free(fftshift(transferfunction))
17 end
18
19 function calculate!(wave::ElectronBeam, free::Free)
20     Ψ = fft(fftshift(wave.ψ))
21     Ψ .*= free.transferfunction
22     wave.ψ = ifftshift(ifft(Ψ))
23 end
```

Listing 12: Definition of the `Free` type, which governs the calculation of the free propagation. In line 5 is the constructor, and in the `calculate!` function starting in line 19 the convolution is calculated.

In listing 12, the calculation of the free propagation is shown. Line 1 describes the `Free` type, where the transfer function is saved in.

In line 5 the constructor for the `Free` type is shown, with the most notable lines being 12-13 where the Fourier space coordinates are defined, and line 15 where the transfer function is calculated. The `@.` in line 15 calls a macro that will automatically change the function calls (`exp`, `*`, `^`, ...) to be vector based and element wise. The transfer function will then be shifted in line 16, which is needed to perform the Fourier transform properly afterward.

The `calculate!` function in line 19 then calculates the effect of the free propagation on the electron beam, and changes the `ElectronBeam` struct accordingly. This is achieved by

multiplying the wave function with the transfer function in reciprocal space. Afterwards, the inverse Fourier transform is applied in line 22.

For this algorithm to give good results, it is important that the sampling criterion described in equation (4.17) is obeyed [VR]. Furthermore, the usage of the Fourier transforms to calculate the convolution for the impulse response in equation (4.7) assumes that input functions, in this case, the input wave function and the impulse response function, are periodic. As the periodicity is not satisfied in our input functions, the Fourier transform calculation is a mere approximation that only is correct when there is enough zero padding around the input intensity distribution [OSB].

Lens

A lens without aberrations can be implemented with a phase shift that is given in equation (4.23). When implementing the lens, we have to think about the sampling again. As calculated before (4.27) and under the assumption that the defocus and the spherical aberration are not contributing, the sampling condition has the form of

$$\Delta x \leq \frac{f\lambda}{2x_{\max}},$$

where $x_{\max} = \sqrt{f\pi/(3C_s\lambda)}$. This sampling condition ensures that the lens phase imprint is not aliased and the implementation is consistent with the physical reality.

```
1 struct Lens <: Component
2     f::Real
3 end
4
5 function calculate!(wave::ElectronBeam, lens::Lens)
6     if lens.f == 0
7         return
8     else
9         wave.ψ .*= @. exp(-1im * π / wave.λ / lens.f * (wave.x^2 + wave.y'^2))
10    end
11 end
```

Listing 13: Here the code is shown that defines the `Lens` type, as well as the calculation of the effect of an ideal lens.

In listing 13 the lens type is defined, containing the focal length f. In the `calculate!` function, in line 5, it will be checked if the focal length is zero, and if so, nothing is done. This is to intercept a divide by zero error in the definition of the phase shift, as it is inversely proportional to the focal length. In line 9 the phase shift is imparted onto the wave function, and thus the lens is emulated.

Additionally to the phase shift, the lens also consists of an aperture. For re-usability as a separate optical element, we implement the `Aperture` type separately.

```
struct Aperture <: Component
    d::Real
end

function calculate!(wave::ElectronBeam, aperture::Aperture)
    for i = 1:size(wave.x, 1), j = 1:size(wave.y, 1)
        if wave.x[i]^2+wave.y[j]^2 <= (aperture.d/2)^2
        else
            wave.ψ[i, j] = 0
        end
    end
end
```

Listing 14: The definition of the `Aperture` type, as well as the corresponding `calculate!` function.

In listing 14 the `Aperture` type is defined. In it, the diameter of the aperture is saved. In the `calculate!` function the aperture is applied to the incoming wave function. This is done by setting every value to zero that is outside of the defined diameter `d`.

Phase imprint

As a reminder, the ponderomotive phase imprint (2.13) has the form of

$$\varphi_p(x,y) \approx -\frac{\alpha}{2\pi(1+\beta)}\frac{E_L}{E_e}\frac{\lambda_L^2 g^2(x,y)}{\int g^2(x,y)\mathrm{d}x\mathrm{d}y},$$

where the $g^2(x,y)$ function describes the spatial light intensity distribution. The $\mp$ from equation (2.13) has been replaced by a $+$ because in the use case of the implementation only counter-propagation was considered.

```
struct PhaseImprint <: Component
    lb::LaserBeam
end

function calculate!(wave::ElectronBeam, imprint::PhaseImprint)
    Δx = abs(imprint.lb.x[1] - imprint.lb.x[2])
    Δy = abs(imprint.lb.y[1] - imprint.lb.y[2])

    β = wave.v / c
    γ = 1 /sqrt(1-β^2)

    Ee = γ * m_e * c^2
    α = q^2 / (hbar * c) / (4 * π * ε_0)

```

```
15      norm = sum(imprint.lb.I) * Δx * Δy
16
17      prefactor = - α / (2*π * (1 + β)) * imprint.lb.E / Ee * imprint.lb.λ^2 /
        ↪  norm
18
19      for j in eachindex(wave.y), i in eachindex(wave.x)
20          wave.ψ[i, j] *= exp(1im * prefactor * interpolation(imprint.lb.I,
            ↪  imprint.lb.x, imprint.lb.y, wave.x[i], wave.y[j]))
21      end
22  end
```

Listing 15: The implementation of the `PhaseImprint` type, and the corresponding `calculate!` function.

In listing 15 the `PhaseImprint` type is defined where a `LaserBeam` type is saved.

The `calculate!` function will then use the `LaserBeam` type to calculate the ponderomotive phase shift, and imprint it onto the electrons. In the first few lines, the relevant constants are defined and in line 15 the light fields are normalized in a discretized version of equation (2.13). It is assumed that the intensity distribution is sampled well enough such as that the sum is an adequately good approximation to the integral and also that the intensity outside of the field of view is zero. In line 20 the phase shifts are imprinted onto the electron wave. To do so, the light intensities are interpolated at the grid points of the electron wave using a bi-linear interpolation function.

4.3. Results

While, large aperture and short focal length lenses are difficult to simulate due to the sampling conditions discussed in 4.1.5, the effect of light distributions at lower intensity can be simulated across a larger area. Here, we discuss the phase imprint and propagation on a collimated electron beam. The propagation length after the interaction is $z = 50\text{cm}$ and the acceleration voltage of the electrons is $U = 30\text{keV}$.

These simulations hint at possible experiments that show clear wave-optical interference effects, that cannot be explained by the ray optics introduced in chapter 3.

The code to simulate the following figures can be found in appendix A.3.2.

4.3.1. Gaussian beam

The first use case is a plane wave electron beam that is cut by a diaphragm with the diameter of $d = 30\mu\text{m}$ and interacts with a Gaussian laser pulse

$$g^2(x, y) = \exp\left(-\frac{x^2 + y^2}{w^2}\right)$$

with a beam diameter $w = 5\mu\text{m}$. The energy of this pulse is set such that the maximum phase shift is π. The required energy per pulse can be calculated, with equation (2.13)

and using the fact that $|g^2(x,y)|_{\max} = 1$, to be $E = 22.8\text{nJ}$. Following the interaction with the laser pulse, we simulate the electron propagation for a distance of $z = 50\text{cm}$ and afterwards calculate the electron probability distribution. Furthermore, it is assumed that the aperture is not completely sharp, but it is smeared out over a Gaussian $\exp\left(-r^2/\sigma^2\right)$ with $\sigma = 1\mu\text{m}$.

In figure 4.2 the result of this simulation is shown. In (a) the phase shift that is applied to the electron wave function can be seen. In (b) the wave optics result of the simulation can be seen, where the electron probability is enhanced in the center. (c) and (d) of figure 4.2 show a ray-tracing simulation with 50 million electrons to compare the results. While figure 4.2(c) shows the original result of the simulation, figure 4.2(d) shows the same data smeared out by a Gaussian $\exp\left(-r^2/w_s^2\right)$ with $w_s = 200\text{nm}$ and renormalized. This reduces the noise due to finite number of simulated trajectories and therefore makes the results more comparable to the wave-optical calculation.

In figure 4.3, the radial distributions of the electron wave simulation (in orange) and the ray-tracing (in violet) simulation are plotted. The noise in the ray-tracing radial part stems from the finite number of electrons that are simulated in each run. It can be seen that, to about a radius of $12\mu\text{m}$, the two simulations give practically identical results but have a slight miss-match at the edge of the electron beam likely due to interference fringes from the aperture. Jupyter notebooks where the simulation can be seen are provided in the appendix A.3.2.

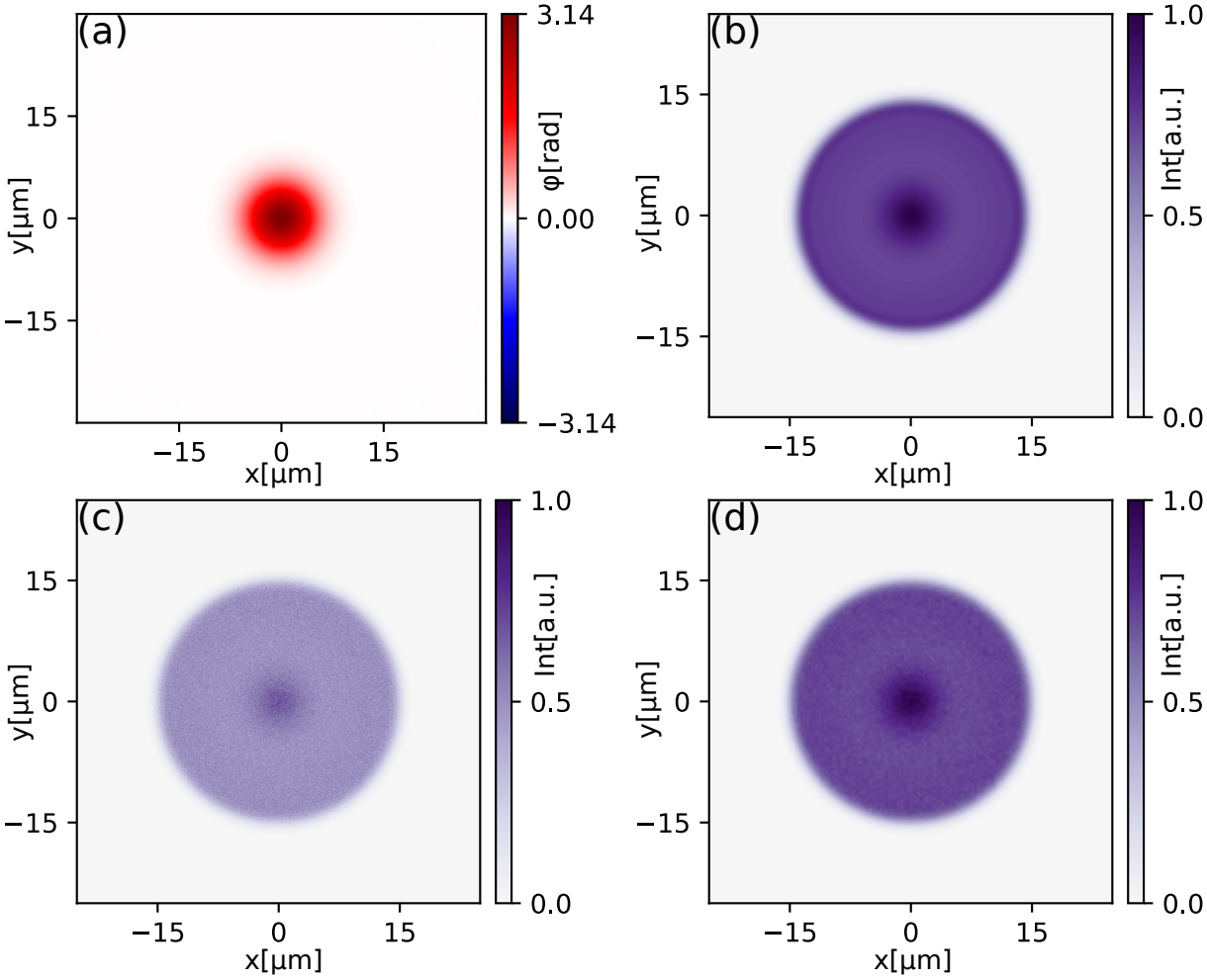


Figure 4.2.: (a) Ponderomotive phaseshift due to a Gaussian laser pulse with an energy of 22.8 nJ. The beam width of the Gaussian is $w = 5\mu$m. In (b) the the result of the wave optics simulation of the electron probability distribution is shown. In the lower row, the result of a ray-tracing simulation is shown with the same initial condition and 50 million electrons. In (c) the raw electron incidents are shown, and in (d) the electron incidents are smeared out with a Gaussian of $w_s = 200$nm and renormalized to make it comparable.

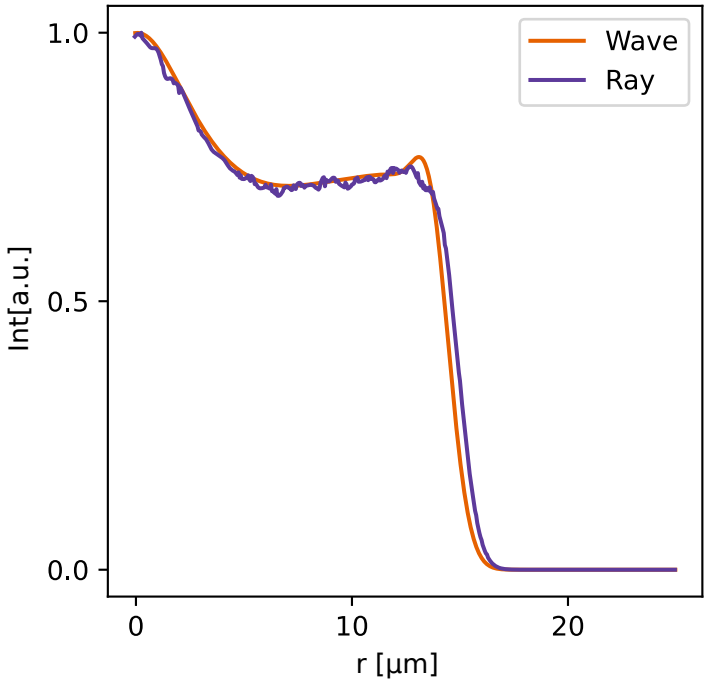


Figure 4.3.: Here the difference between the wave simulation in orange and the ray-tracing simulation in violet is shown. As a foundation for this figure, plots (b) and (d) from figure 4.2 are used, then the radial integral is computed and shown in this plot.

4.3.2. Two Gaussians

In this section we simulate the effect of two Gaussian beams on a plane wave electron beam. We will see that this situation will lead to interference effects, leading to different results for the wave- and ray-optics simulation. Again, the incoming electron beam is cut by a 30μm aperture. The free electron propagation after the interaction is $z = 50$cm and the acceleration voltage is $U = 30$keV.

In figure 4.4 the laser intensity distribution can be seen in (a) as well as the induced phase shift for a pulse energy of $E = 60$nJ and a wavelength of $\lambda_L = 1035$nm (b). The FWHM of the Gaussians is 4.3μm and the separation of them is 6μm. Figure 4.5 shows the electron probability distribution and its cross section along the x-axis for different energies. In 4.5 (a) and 4.5 (b) we simulate a light pulse energy of 50nJ and we see minima and maxima due to interference of the waves scattered from the two ponderomotive potentials. For 4.5 (c) and 4.5 (d) we change the light pulse energy to 60nJ and see that the central peak intensity decreases while the number of visible fringes increases. In 4.5 (e) and 4.5 (f) the pulse energy is 70nJ which leads to a further decrease of the central peak and another increase in the number of interference fringes. For finite detector resolution, and

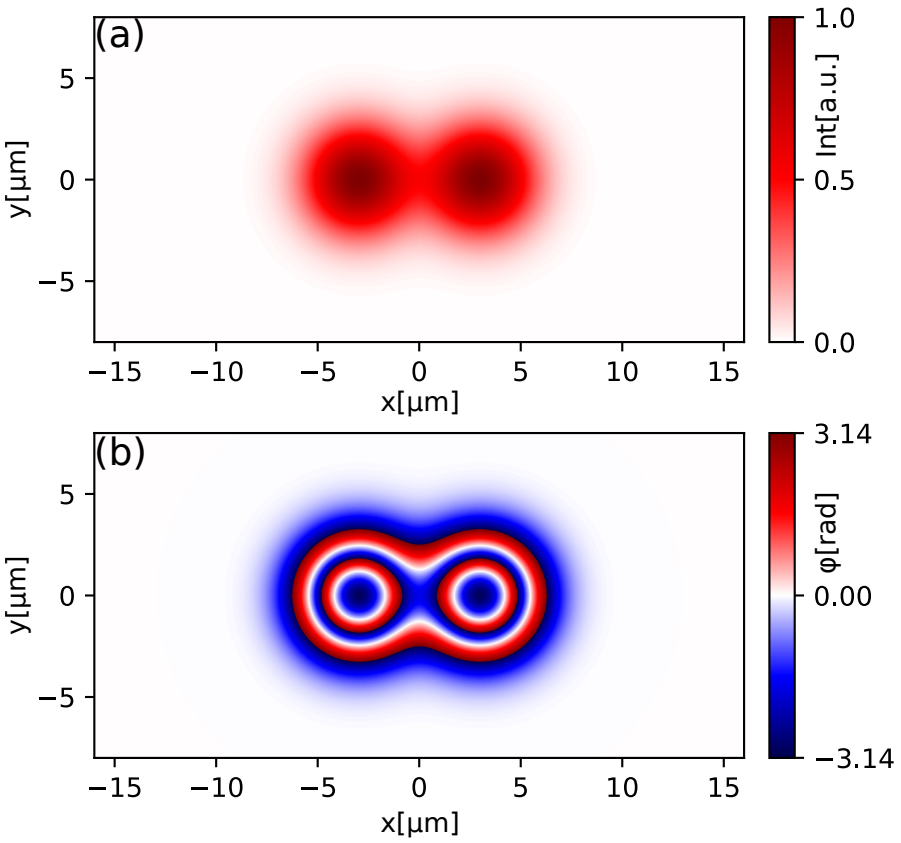


Figure 4.4.: Laser intensity distribution of two Gaussian's with FWHM of 4.3μm and separation of 6μm. The phase shift in (b) corresponds to a total pulse energy of 60nJ and a wavelength of 1035nm.

in the limit of very high pulse energy, the quantum mechanical result will resemble the classical one. This is the regime we present in our first publication [MWS$^+$].

Figure 4.6 is a ray tracing simulation that is directly comparable to figure 4.5. Contrary to the wave optics simulation, the electron distribution pattern (left column of 4.6) and cross section (right column of 4.6) in the ray tracing simulation have to be simulated separately. This is needed because, if we take the cross section from the electron distribution, the finite number of electrons will make the cross section very noisy, and an increase of simulated electrons is very inefficient as the electrons will most likely not contribute to the cross section. We simulate the electron distribution with a sample of 10 million electrons, and the cross section, after a reduction of the field of view to 8μm, with 20 million electrons. As every detector for electrons has a point spread function that is not a delta peak, we smeared out each of the electron incidents with a Gaussian $\exp(-\rho^2/w_s^2)$ with $w_s = 150$nm width to get the image and the cross-section. The noise that can be seen in the cross-section plots stems from the finite amount of electrons used, and can only be further reduced with a considerable numerical effort that will not contribute much

to the deeper understanding of this simulation. For the different laser pulse energies, in 4.6 (a) and 4.6 (b) we have 50nJ, in 4.6 (c) and 4.6 (d) 60nJ, and 4.6 (c) and 4.6 (d) 70nJ, we see that the ray tracing approach does not have interference fringes as we expected.

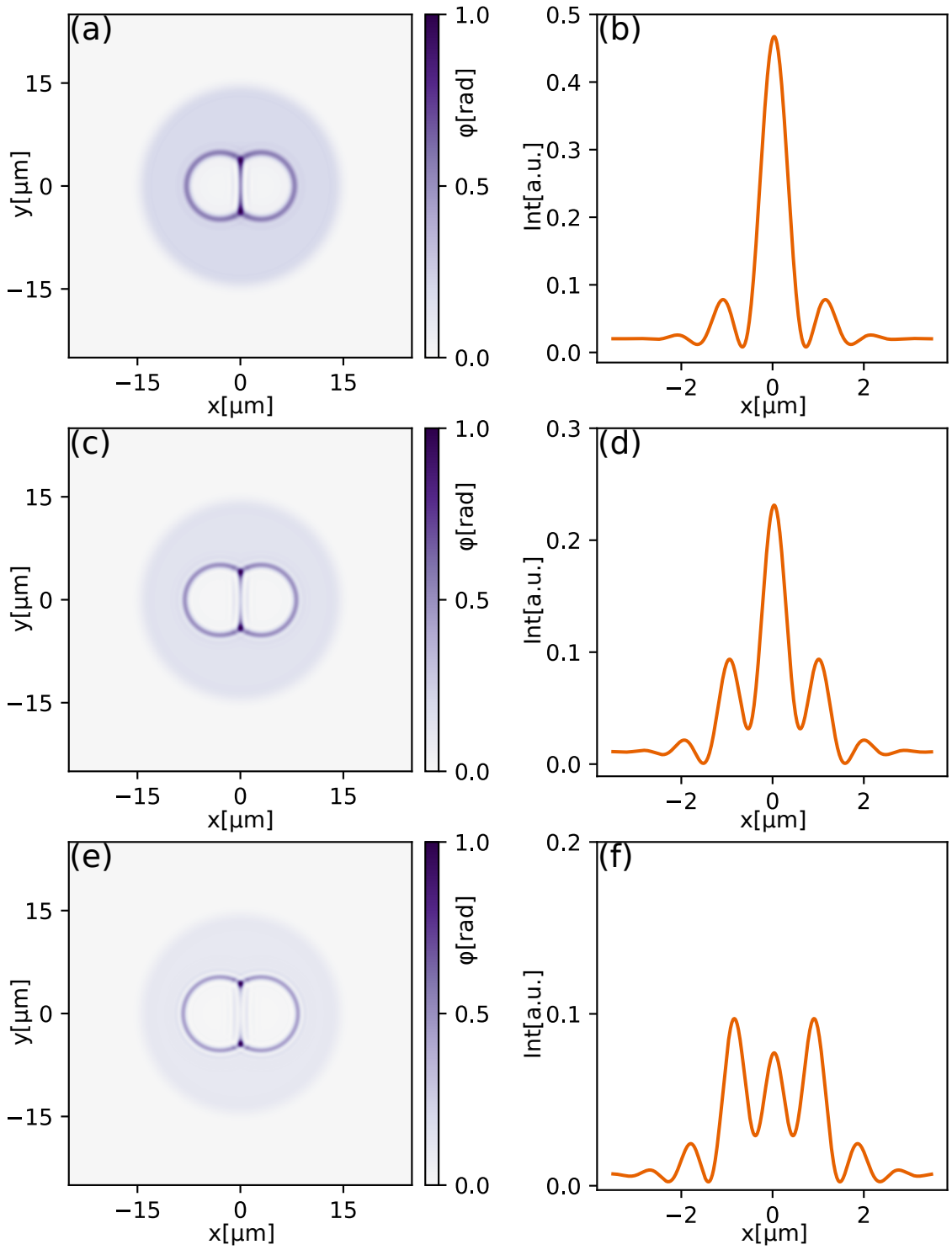


Figure 4.5.: The simulated electron probability distribution can be seen on the left side and the corresponding cross-section at $y = 0$ along the x-axis. The images (a) and (b) are for a laser pulse energy of 50nJ, (c) and (d) for 60nJ and (e) and (f) for 70nJ, respectively.

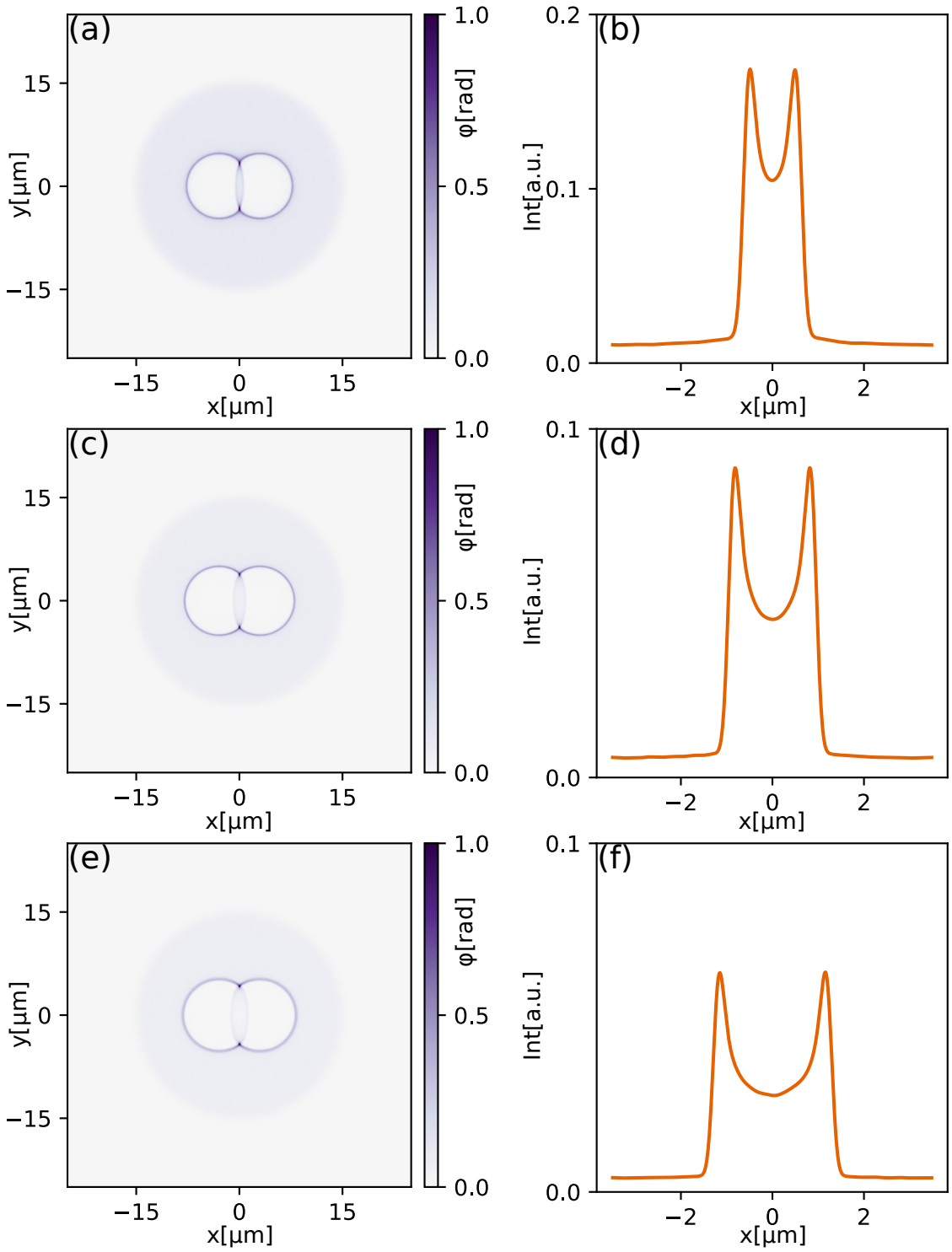


Figure 4.6.: This ray-tracing simulation has the same settings as the wave simulations in figure 4.5. In (a) and (b) the pulse energy is 50nJ, in (c) and (d) it is 60nJ and in (e) and (f) it is 70nJ. It can clearly be seen that there are no interference fringes.

4.3.3. Two Gaussians with Different Phase Shifts

In the next use case, a small intensity difference between the two laser beams is considered, leading to phase differences in the two waves scattered from the two potentials. Varying the intensity difference should therefore lead to a shift of the interference maxima and minima. The total beam energy is $E = 60\text{nJ}$ per pulse, and the FWHM of each Gaussian peak is $4.3\mu\text{m}$.

In figure 4.7 (a) the laser beam distribution is shown where the centers of the Gaussians are $d = 6\mu\text{m}$ apart from each other. In (b) the phase imprint on an electron is shown. We see that the amplitudes of the two Gaussians are chosen such that the phase shift difference at the respective centers is π. The transverse mode profile squared is written as

$$g^2(x,y) = Af(x - d/2, y) + Bf(x + d/2, y),$$

with

$$f(x,y) = \exp\left(-\frac{x^2+y^2}{2\sigma^2}\right),$$

where $\sigma = \text{FWHM}/2.355$ is the beam width. The parameters A and B, denote the amplitude of the mode profile. The amplitudes are the degrees of freedom that can be changed to achieve a phase shift $\Delta\varphi$ between the two Gaussians. To derive the values for A and B, that are required for a certain phase shift difference, we rewrite the laser phase imprint in equation (2.13) to

$$\varphi(x,y) \approx -K\frac{E_L}{\int \mathrm{d}x\mathrm{d}y\, g^2(x,y)} g^2(x,y),$$

where $K = \alpha\lambda_L^2/\left[2\pi(1+\beta)E_e\right]$ is the constant pre-factor and E_L is the laser pulse energy. Note that, with these definitions $g^2(x,y)$ is unit-less. Inserting the mode profile definition into the phase shift formula, we get

$$\varphi(x,y) \approx -K\frac{E_L}{(A+B)N}\left[Af(x - d/2, y) + Bf(x + d/2, y)\right],$$

where $N = \int \mathrm{d}x\mathrm{d}y\, f(x,y) = 2\pi\sigma^2$ is the normalization factor for each individual Gaussian. Under the assumption that $f(d,0) \ll f(0,0)$, which holds for the given parameters $(0.0045 \ll 1)$, and that $f(0,0) = 1$ the phase shift in the center of each Gaussian can be written as

$$\varphi(-d/2, 0) \approx -K\frac{E_L}{(A+B)N}A$$
$$\varphi(d/2, 0) \approx -K\frac{E_L}{(A+B)N}B.$$

The difference in phase between the two Gaussian peaks is $\Delta\varphi = \varphi(-d/2, 0) - \varphi(d/2, 0)$. It can be calculated if we assume, without loss of generality, that $A = 1$. Rewriting everything to calculate the second amplitude we get to

$$B = \frac{1 - \frac{N\Delta\varphi}{KE_L}}{1 + \frac{N\Delta\varphi}{KE_L}},$$

where we are now able to construct intensity distributions with arbitrary phase shifts between their peaks.

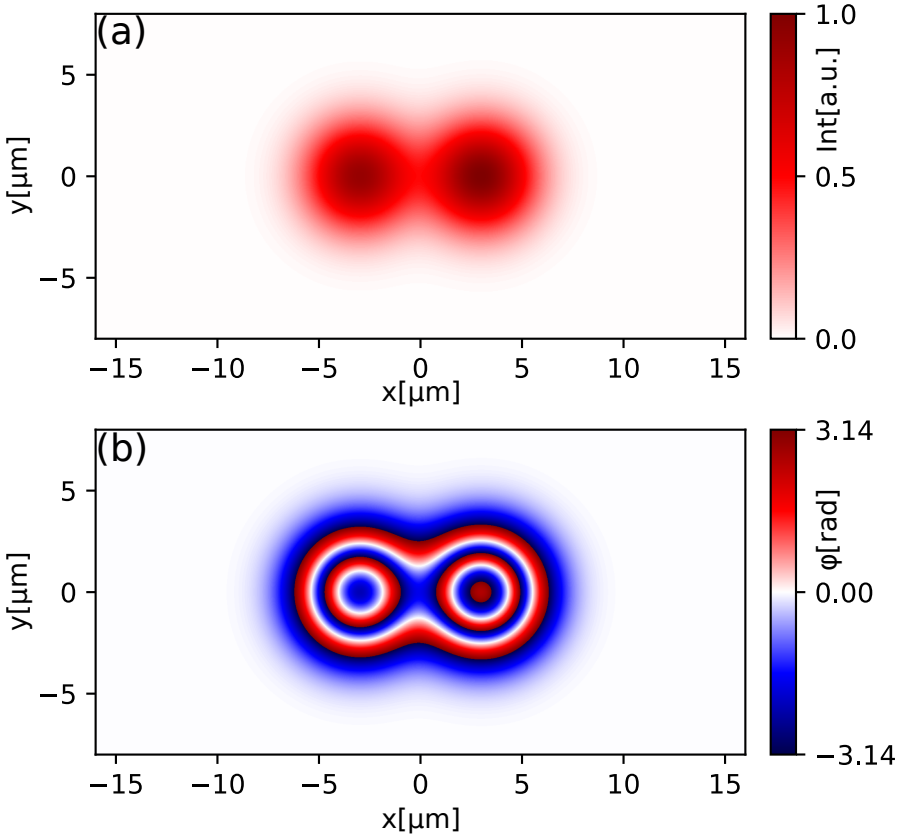


Figure 4.7.: In (a) the laser intensity distribution is shown for a total pulse energy of $E = 60\text{nJ}$. The FWHM for both beams is the same at $4.3\mu\text{m}$ and the intensity distribution is chosen such that the difference between phase shifts at the maximum is exactly π. In (b) the applied phase shift is shown.

In figure 4.8 we simulate collimated electrons with an acceleration voltage of 30keV and an electron diameter of $30\mu\text{m}$. The laser pulse energy was set to $E = 60\text{nJ}$, but the difference in phase between the two peaks was changed for every calculation. In the left column of figure 4.8, the electron distribution is shown, and in the right column, a

cross-section at $y = 0$ around the center is shown. In the first row of figure 4.8, plots (a) and (b), the phase difference $\Delta\varphi$ was chosen to be $\Delta\varphi = \pi/2$. It can be seen that there is not a big change compared to figure 4.5 (c) and (d). In the second row, images (c) and (d) of figure 4.8, the phase difference was chosen to be π. Again, both Gaussians deplete the center due to the repulsive nature of the ponderomotive interaction. The effect is stronger for the pulse with higher intensity, leading to increased contrast in the diffraction pattern. The higher ponderomotive phase shift also leads to an overall shift of the interference pattern in the negative x-direction. In the third row, corresponding to plots (e) and (f) in figure 4.8, the phase difference is changed to 2π. This increases the effect seen in (c) and (d) by quite some margin, and a clear difference compared to the plots with the same amplitude is visible.

Figure 4.9 shows a ray tracing simulation with the same parameters as described before. It is clearly visible that the interference fringes that are visible in figure 4.8 are no longer visible in figure 4.9, as is expected.

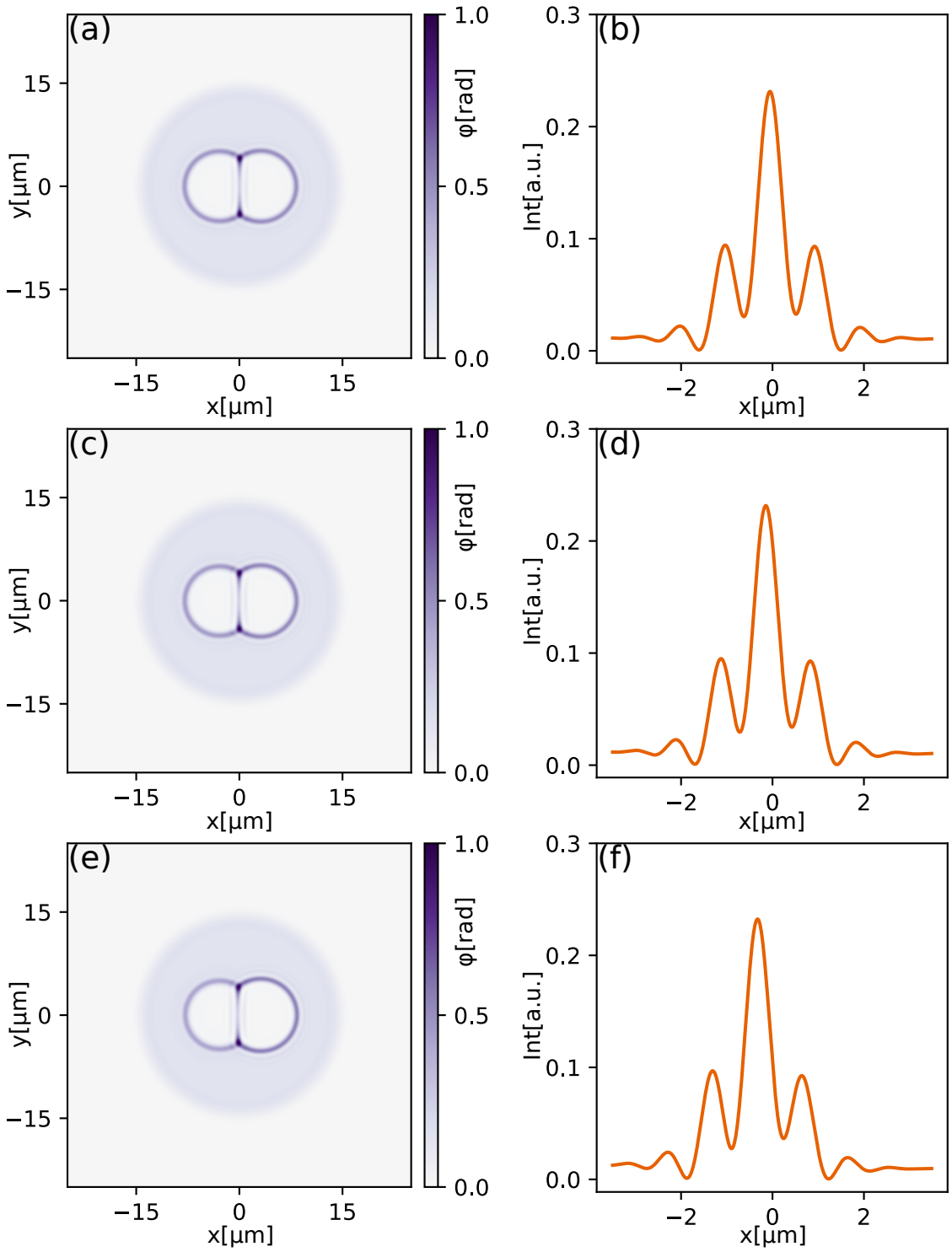


Figure 4.8.: The simulated electron probability distribution is shown in the left column, a cross-cut at $y = 0$ around the middle point is plotted in the right column. The laser pulse energy is $E = 60$nJ, where the right Gaussian is tweaked to induce a higher phase shift than the left Gaussian. In (a) and (b) the difference is $\pi/2$, in (c) and (d) it is π and in (e) and (f) it is 2π.

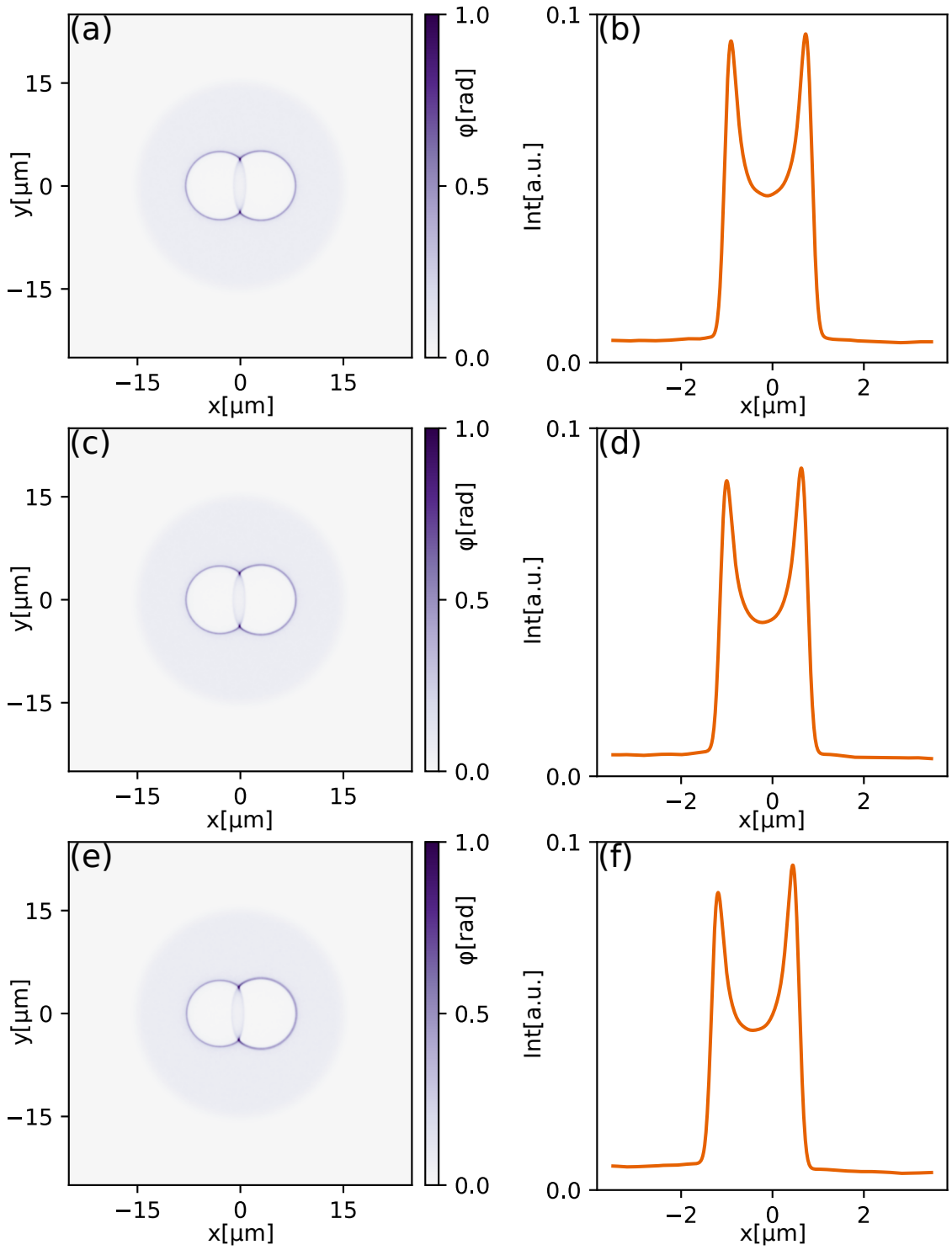


Figure 4.9.: This figure represents the ray-tracing version of figure 4.8 with the same experimental parameters. The difference in phase shift between the two Gaussians is in (a) and (b) $\pi/2$, in (c) and (d) π and in (e) and (f) 2π.

5. Outlook & Conclusion

In this section, we will review the results of this thesis and talk about possible implications.

5.1. Electron - Photon Interaction

In this thesis, we have derived the effect of stimulated Compton scattering of an electron in two different ways. We first introduced a quantum mechanical description in section 2.1. It is based on the interaction Hamiltonian in an electromagnetic field in free space. The interaction Hamiltonian, shown in equation (2.3), has a first order term that vanishes and a second order term that gives rise to the ponderomotive phase shift. We show that for our setup, the ponderomotive phase shift is proportional to the transverse laser intensity distribution.

We then introduced a classical ray-optics description of the interaction and the following electron propagation in section 3.1.3. The electrons are assumed to be classical particles propagating along straight trajectories. Electron beam simulations are performed using a Monte-Carlo approach, which is very efficient. A downside of this approach is that quantum-mechanical interference effects are not taken into account.

These can be simulated using the wave-optics approach introduced in chapter 4. However, due to sampling constraints, this approach is computationally expensive, and cannot be pursued in all practically relevant situations.

For the experiments in [MWS$^+$], we show that the quantum mechanical description in section 2.1 and the ray-optics description in section 3.1.3 are equivalent. This enables us to either do the simulations in a wave-optics or in a ray-tracing regime and the simulations will give the correct results in the appropriate situation.

5.2. Ponderomotive Lenses

In section 2.2 different possibilities are described of how a ponderomotive electron lens can be realized.

We find that both convex and concave lenses can be realized using electron-light interactions. The latter case is particularly interesting, considering that it is difficult to construct a concave electron lens (see section 2.2). Provided that the laser intensity distribution can be created sufficiently well and there is enough laser energy, it is easy to use the electron-light interaction to construct a ponderomotive lens. In [MWS$^+$] we realized convex and concave ponderomotive lenses experimentally. We could realize focal length down to 1.6mm with an electron beam radius of $R \approx 1.6\mu$m. Increasing the beam radius would require higher laser pulse energy. Our calculations show that, at constant

focal length, the laser pulse energy will grow with $E_L \propto R^4$ and thus the creation of larger electron lenses will get increasingly difficult. To experimentally realize a laser lens in an ultrafast transmission electron microscope, the simulations in section 3.3 indicate that for larger electron beam diameters (in this simulation 15μm) laser pulse energies E_L in the order of mJ are required to form the lens.

Far smaller laser pulse energy is required if the ponderomotive phase shifts are used to correct the aberrations of an already existing, electro- or magnetostatic lens. In this case, only 4.6μJ would be required to arbitrarily shape a disc with a diameter of 100μm, assuming perfect control of the laser distribution.

5.3. Simulation Approaches

The simulation of our experiments [MWS$^+$] was the main goal of this thesis. I wrote both a ray-optics and a wave-optics simulation to achieve this goal. The ray-optics approach is described in chapter 3, and we show the code in appendix A.2.1 as well as on GitHub `https://github.com/JuffmannLab/Raytracing`. There, the electrons are assumed to be single particles that travel in straight rays when in free space. We describe the second approach, the so-called wave optics calculation, in chapter 4. In appendix A.3.1 as well as on GitHub `https://github.com/JuffmannLab/ElectronPropagation` we show the source code for the wave-optics simulation. The wave optics calculation is based on the principle that the electron is a matter wave and will simulate the propagation of said wave.

When comparing ray tracing with the wave optics approach, it is apparent that ray tracing can only describe the classical limit. The wave optics approach, on the other hand, is fully quantum mechanical and can also describe interference effects of the electron wave-function. It can thus be said that the more exact approach is wave optics.

The big advantage of the ray-tracing code is, that it is, contrary to the wave-optics, not dependent on sampling conditions. It is thus possible to construct and simulate arbitrary electron-optical setups. This becomes apparent when simulating an electron lens. The sampling condition is given in equation (4.27), and plugging in numbers, we see that the matrices required for a wave-optics simulation are getting very large.

For this reason, it is very difficult to simulate a ponderomotive lens, which is a conventional electro- or magneto-static lens, with the wave optics approach. The ray tracing method has no limitations to sampling, it has an enormous advantage over the wave optics method in computational resource usage and is, as such, better suited for simulations, in which interference effects have no effect.

5.4. Outlook

An interesting application of the provided code is to see if our experimental setup allows for detection of interference effects. Similarly to the double Gaussian setup in section 4.3.2 and with the point spread function defined in [MWS$^+$], we can simulate the effects either classically or quantum mechanically. If it is possible to find parameters where the

difference in visibility of the central peak for the two simulations is large enough, we could experimentally show matter wave interference in the current setup. Because the spatial resolution is limited, the main difficulty of this task is to find parameters where a quantum mechanical effect can be made visible in our experimental setting [MWS$^+$]. An increased spatial resolution might in the future be achieved experimentally using magnification optics below the interaction plane, or by using high-resolution scintillator screens.

Ponderomotive electron beam shaping opens up the possibility to create Zernike phase contrast using a transmission electron microscope. To achieve Zernike phase contrast, the unperturbed electron needs to be phase-shifted, which was already shown to be possible using a laser phase-plate [SAC$^+$]. The goal is to find a laser intensity distribution that acts as a phase-plate as well as an aberration corrector. The framework that is provided in this thesis will be able to simulate such an electron microscope and estimate an outcome.

Bibliography

[ACS+] Jeremy J. Axelrod, Sara L. Campbell, Osip Schwartz, Carter Turnbaugh, Robert M. Glaeser, and Holger Mueller. Observation of the relativistic reversal of the ponderomotive potential. 124(17):174801.

[Bak] Bakerian lecture.—The discharge of electricity through gases. (Preliminary communication.).

[Bat] H. Batelaan. *Colloquium* : Illuminating the Kapitza-Dirac effect with electron matter optics. 79(3):929–941.

[BEKS] Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B. Shah. Julia: A Fresh Approach to Numerical Computing. 59(1):65–98.

[BFZ] Brett Barwick, David J. Flannigan, and Ahmed H. Zewail. Photon-induced near-field electron microscopy. 462(7275):902–906.

[Com] Arthur H. Compton. A Quantum Theory of the Scattering of X-rays by Light Elements. 21(5):483–502.

[dAK] F. Javier García de Abajo and Andrea Konečná. Optical Modulation of Electron Beams in Free Space. 126(12):123901.

[dB] Louis de Broglie. Recherches sur la théorie des quanta. page 121.

[FAB] Daniel L. Freimund, Kayvan Aflatooni, and Herman Batelaan. Observation of the Kapitza–Dirac effect. 413(6852):142–143.

[Haw] P. W. Hawkes. The correction of electron lens aberrations. 156:A1–A64.

[HK] Peter W Hawkes and Erwin Kasper. *Principles of Electron Optics: Wave Optics*, volume 3. Academic press.

[HWW+] Yanlei Hu, Zhongyu Wang, Xuewen Wang, Shengyun Ji, Chenchu Zhang, Jiawen Li, Wulin Zhu, Dong Wu, and Jiaru Chu. Efficient full-path optical calculation of scalar and vector diffraction using the Bluestein method. 9(1):119.

[KD] P. L. Kapitza and P. a. M. Dirac. The reflection of electrons from standing light waves. 29(2):297–300.

[Kir] Earl J. Kirkland. *Advanced Computing in Electron Microscopy*. Springer US.

[LL] Marcel Leutenegger and Epfl Sti Ioa Lob. Fast focus field calculations. page 15.

[MAA+] Benjamin J. McMorran, Amit Agrawal, Ian M. Anderson, Andrew A. Herzing, Henri J. Lezec, Jabez J. McClelland, and John Unguris. Electron Vortex Beams with High Quanta of Orbital Angular Momentum. 331(6014):192–195.

[Mes] Albert Messiah. *Quantum Mechanics*. Courier Corporation.

[MWS+] Marius Constantin Chirita Mihaila, Philipp Weber, Matthias Schneller, Lucas Grandits, Stefan Nimmrichter, and Thomas Juffmann. Transverse Electron Beam Shaping with Light.

[Nol] Wolfgang Nolting. *Grundkurs Theoretische Physik 4*. Springer-Lehrbuch. Springer Berlin Heidelberg.

[OSB] Alan V. Oppenheim, Ronald W. Schafer, and John R. Buck. *Discrete-Time Signal Processing*. Prentice Hall, 2nd ed edition.

[Plu] M. Pluecker. XLVI. *Observations on the electrical discharge through rarefied gases*. 16(109):408–418.

[PW] Justin Peatross and Michael Ware. *Physics of Light and Optics*.

[RSL+] Dolev Roitman, Roy Shiloh, Peng-Han Lu, Rafal E. Dunin-Borkowski, and Ady Arie. Shaping of Electron Beams Using Sculpted Thin Films. 8(12):3394–3405.

[Rus] Ernst Ruska. The development of the electron microscope and of electron microscopy. 59(3):627–638.

[SAC+] Osip Schwartz, Jeremy J. Axelrod, Sara L. Campbell, Carter Turnbaugh, Robert M. Glaeser, and Holger Müller. Laser phase plate for transmission electron microscopy. 16(10):1016–1020.

[Sch] O. Scherzer. Über einige fehler von elektronenlinsen. 101(9):593–603.

[SLLA] Roy Shiloh, Yossi Lereah, Yigal Lilach, and Ady Arie. Sculpturing the electron wave function using nanoscale phase masks. 144:26–31.

[SRL+] Roy Shiloh, Roei Remez, Peng-Han Lu, Lei Jin, Yossi Lereah, Amir H. Tavabi, Rafal E. Dunin-Borkowski, and Ady Arie. Spherical aberration correction in a scanning transmission electron microscope using a sculpted thin film. 189:46–53.

[UT] Masaya Uchida and Akira Tonomura. Generation of electron beams carrying orbital angular momentum. 464(7289):737–739.

[VR] David G. Voelz and Michael C. Roggemann. Digital simulation of scalar optical diffraction: Revisiting chirp function sampling criteria and consequences. 48(32):6132.

A. Appendix

A.1. Realistic Convex Lens

When considering realistic ponderomotive lenses, we will inevitable have imperfections in the laser intensity distribution. As we want to have simulations not only on perfect lenses but also on realistic lenses, we have to adapt the model for a ponderomotive lens to a realistic case. In our experiment we measured the laser intensity distribution of a convex lens [MWS$^+$]. In figure A.1 we can see the laser intensity distribution as well as a cross section of it.

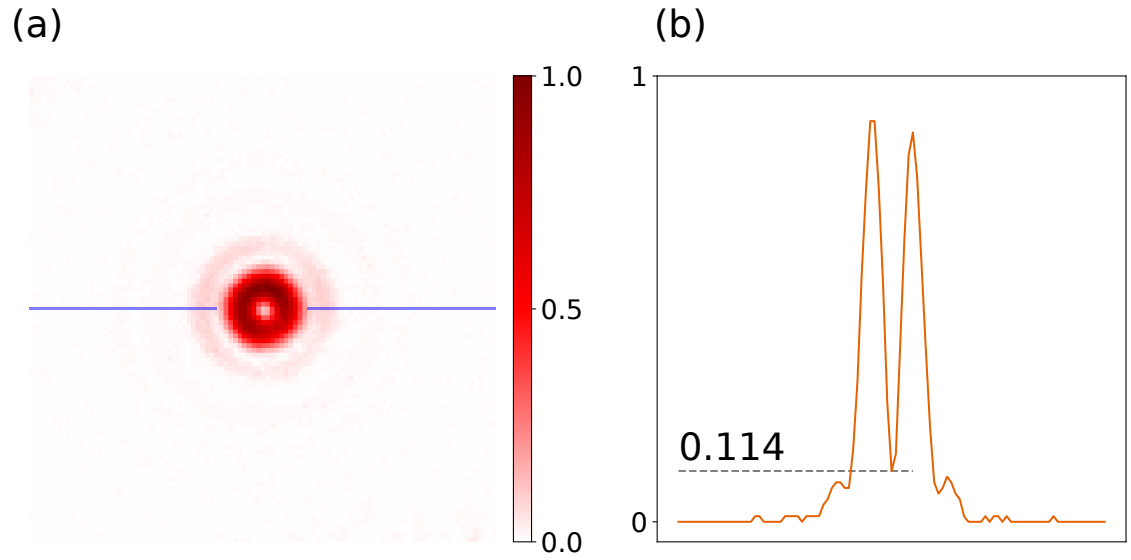


Figure A.1.: In (a) of this figure the experimental light field distribution of [MWS$^+$] can be seen. The distribution is normalized to 1, and the blue line marks the plane, where the cross section is taken. This cross section is shown in (b) with the dashed black line showing the minimal relative intensity of 0.114.

To now model this experimental data we take the sum of Gaussian with the LG10 mode as an ansatz for fitting. The idea is to add a function that goes sufficiently fast to zero to the LG10 beam and see if the intensity slope, in the first order, still is quadratic. A natural choice for a function that would fit the requirements is a Gaussian beam with a prefactor that is equivalent to the minimal value in the doughnut beam. Now we define

the corrected intensity distribution as

$$I(\rho) = \frac{\varepsilon_0 c I_0}{2}\left[A\exp\left(-\frac{2\rho^2}{B^2 w_0^2}\right) + \left(\frac{\rho\sqrt{2}}{w_0}\right)^2 \exp\left(-\frac{2\rho^2}{w_0^2}\right)\right], \tag{A.1}$$

where A and B are fitting parameters [MWS$^+$].

The Taylor expansion of equation (A.1)

$$I(\rho) = \frac{\varepsilon_0 c I_0}{2}\left[A - \frac{2}{w_0^2}\left(\frac{A}{B^2} - 1\right)\rho^2 + \frac{2}{w^4}\left(\frac{A}{B^2} - 2\right)\rho^4\right] + \mathcal{O}(\rho^6) \tag{A.2}$$

is given by equation (A.2), and it can be seen that the lowest, non-constant intensity is a quadratic term, which makes this intensity ansatz viable for a potential concave lens.

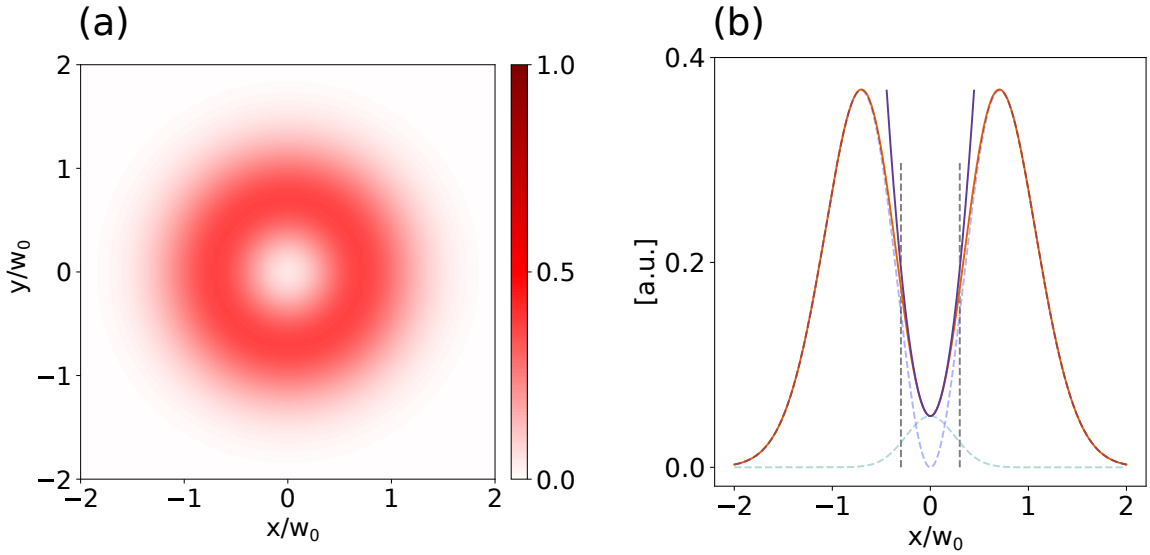


Figure A.2.: In (a) the corrected intensity distribution in equation (A.1) is shown with arbitrary fitting parameters $A = 0.05$ and $B = 0.5$. In (b) the cross section of (a) is shown in orange and the quadratic order of the Taylor expansion in purple. The Gaussian (dashed teal) and LG10 (dashed blue) mode are also visiualized. The vertical dashed black lines denote the effective lens size $\rho_{\text{eff}} = 0.3005 w_0$, where the relative error is $\epsilon = 0.1$.

To calculate the effective size of the lens ρ_{eff} the relative error between the intensity distribution (A.1) and the first order of the Taylor expansion (A.2) is calculated to be

$$\epsilon = \frac{\left|A - \frac{2}{w_0^2}\left(\frac{A}{B^2} - 1\right)\rho_{\text{eff}}^2 - A\exp\left(-\frac{2\rho_{\text{eff}}^2}{B^2 w_0^2}\right) - \left(\frac{\rho_{\text{eff}}\sqrt{2}}{w_0}\right)^2 \exp\left(-\frac{2\rho_{\text{eff}}^2}{w_0^2}\right)\right|}{A - \frac{2}{w_0^2}\left(\frac{A}{B^2} - 1\right)\rho_{\text{eff}}^2}.$$

Using the definition of the effective lens radius to be the size up until the relative error has reached $\epsilon = 0.1$, this formula can be reshuffled and the effective lens size calculated to be $\rho_{\text{eff}} = 0.3005 w_0$.

In figure A.2, the new intensity ansatz can be seen, as well as the comparison between the quadratic order of the Taylor expansion and the intensity distribution.

Focal length For the focal length calculation, the integral from equation (2.13) has to be evaluated. This means for a realistic concave lens

$$\int_{-\infty}^{\infty} \mathrm{d}x \int_{-\infty}^{\infty} \mathrm{d}y\, I(x,y) = 2\pi \int_0^{\infty} \mathrm{d}\rho\, \rho I(\rho) = \frac{\varepsilon_0 c I_0}{2} \frac{\pi}{2} w_0^2 (AB^2 + 1),$$

with which the total phase imprint for this intensity distribution can be calculated to be

$$\varphi_p(x,y) \approx -\frac{\alpha}{\pi^2(1 \mp \beta)} \frac{E_L}{E_e} \lambda_L^2 \frac{A - \dfrac{2}{w_0^2}\left(\dfrac{A}{B^2} - 1\right)\rho^2}{w_0^2(AB^2 + 1)}.$$

Conducting a comparison of coefficients between the phase imprint calculated here and the phase imprint of a perfect lens in equation (2.14) yields the focal length

$$f(E_L) \approx \pi^3 (1 \mp \beta) \frac{w^4 (AB^2 + 1) E_e}{2\lambda_l^2 \lambda_e \alpha (A/B^2 - 1) E_L}.$$

A.2. Raytracing

A.2.1. Code

In this section the code for the ray tracing simulation that is described in chapter 3 will be provided. The code provided here can be seen on the Github page of the Juffmann Lab `https://github.com/JuffmannLab/Raytracing`. As the code will be provided in a Package, there are files (the Manifest.toml and the Project.toml) that are generated by Julia itself and will not be provided here. The folder structure of the code can be seen in this directory tree

```
Raytracing/
├── LICENSE
├── Manifest.toml
├── Project.toml
├── README.md
└── src/
    ├── Raytracing.jl
    ├── electron.jl
    ├── helperFunctions.jl
    ├── lightField.jl
    ├── components.jl
    └── components/
        ├── free.jl
        ├── lens.jl
        └── pondInteraction.jl
```

where the LICENSE file contains the "European Union Public License 1.2" license text. The different files provided in the same order as they are shown in the directory tree.

Raytracing.jl

```
1  module Raytracing
2
3  # export the different functions and structs that can be used
4  export Free, Lens, PondInteraction, Setup, propagation!
5  export Electron, getintensity, mcp
6  export loadintensity, LightField
7
8  # define some nature constants
9  global const m_e = 9.1093837015e-31  # the electron mass
10 global const q = 1.602176634e-19     # electron charge
11 global const c = 299792458           # the speed of light
12 global const ε_0 = 8.8541878128e-12  # vacuum permitivity
13
```

```
# include all the code that is used
include("./helperFunctions.jl")
include("./electron.jl")
include("./lightField.jl")
include("./components.jl")

end # module
```

electron.jl

```
using FFTW

mutable struct Electron
    ψ::Vector{Array}
    intensity::Vector{<:Real}
    v::Real
    x::Vector{<:Real}
    y::Vector{<:Real}
end

"""
    Electron(x::Vector{<:Real}, y::Vector{<:Real},
    ↪ intensity::Matrix{<:Real},
             U::Real, n::Int64)::Electron2d

Return the Electron2d type.

Construct and return a 2D wave. The `x` and `y` vector correspond to
    ↪ the
x and y axis of the problem, and the `intensity` is the intensity
    ↪ corresponding
to the x and y axis given prior. The `U` parameter denotes the
    ↪ electron
energy in eV. The number of rays that should be simulated are
    ↪ denoted by the `n`
value.
"""
function Electron(x::Vector{<:Real}, y::Vector{<:Real},
    ↪ intensity::Matrix{<:Real},
                  U::Real, n::Int)::Electron

    # create the array with the different angles and positions
    ψ = Vector{Array}(undef, n)
```

```
28
29      # create a flattened intensity array
30      int_flat = similar(intensity, n)
31
32      # calculate the relativistic electron velocity
33      v = c * sqrt(1 - 1 / (1 + U * q / m_e / c^2)^2)
34
35      # calculate the span of x and y, aswell as Δx and Δy
36      span_x = abs(x[1]-x[end])
37      span_y = abs(y[1]-y[end])
38      Δx = abs(x[2]-x[1])
39      Δy = abs(y[2]-y[1])
40
41      # iterate over the ψ array, fill it with random coordinates
42      for i = 1:n
43          # generate random coordinates that are evenly distributed in
            ↪  x and y direction
44          coords_rand = rand(Float64, 2) .- 0.5
45          x_temp = coords_rand[1] * span_x
46          y_temp = coords_rand[2] * span_y
47
48          # fill the ψ, p_ges and flat_int vectors
49          ψ[i] = [x_temp, y_temp, 0., 0.]
50          int_flat[i] = _interpolation(intensity, x, y, x_temp,
            ↪  y_temp)
51      end
52
53      # the wanted struct
54      return Electron(ψ, int_flat, v, x, y)
55  end
56
57  """
58      getintensity(ray::Electron), x::Vector{<:Real},
    ↪  y::Vector{<:Real}::Matrix{<:Real}
59
60  Return the intensity.
61
62  Calculate the current intensity and return it. The `ray` value is
    ↪  the electron beam struct
63  from which the intensity should be returned, the `x` and `y` values
    ↪  are the coordinates
64  in which the intensity should be returned.
```

```
"""
function getintensity(ray::Electron, x::Vector{<:Real},
↪  y::Vector{<:Real})::Matrix{<:Real}

    # create the empty intensity array
    intensity = zeros(Float64, size(x, 1), size(y, 1))
    Δx = abs(x[1]-x[2])
    Δy = abs(y[1]-y[2])

    # loop over the rays
    for i in eachindex(ray.ψ)

        # calculate the position of the intensity in the new grid
        # the minus the first element "normalizes" the grid
        n = round(Int, (ray.ψ[i][1]-x[1]) / Δx, RoundDown) + 1
        m = round(Int, (ray.ψ[i][2]-y[1]) / Δy, RoundDown) + 1

        # add the intensity to the grid point
        if 0 < n < size(intensity, 1) + 1 && 0 < m < size(intensity,
        ↪  2) + 1
            intensity[n, m] += ray.intensity[i]
        else
        end
    end

    # return the intensity
    return intensity
end

"""
    mcp(intensity::Matrix{<:Real},
↪  psf::Matrix{<:Real})::Matrix{<:Real}

Return the mcp electron distribution.

The light that leaves the mcp is calculated here. It is done by
↪  taking the input electron
intensity distribution in the `intensity` matrix and the point
↪  spread funciton `psf` of the
measuring device.
"""
```

```
function mcp(intensity::Matrix{<:Real},
    ↪ psf::Matrix{<:Real})::Matrix{<:Real}
    INTENSITY = fft(intensity)
    PSF = fft(psf)
    PSF .*= INTENSITY
    return real.(ifftshift(ifft(PSF)))
end
```

helperFunctions.jl

```
"""
    _interpolation(A::Matrix{<:Real}, coords_x::Vector{<:Real},
                   coords_y::Vector{<:Real}, x::Real, y::Real)::Real
Return the bilinear interpolation.
Calculate and return the bilinear interpolation on the Matrix `A`
with the coordinates `coords_x` and `coords_y` at the point with
the position `x`, `y`.
"""
function _interpolation(A::Matrix{<:Real}, coords_x::Vector{<:Real},
                        coords_y::Vector{<:Real}, x::Real,
                        ↪ y::Real)::Real

    # if the value is not in the area of coords return 0
    if x < coords_x[1] || x > coords_x[end]
        return zero(eltype(A))
    elseif y < coords_y[1] || y > coords_y[end]
        return zero(eltype(A))
    end

    # calculate the Δx and Δy values
    Δx = abs(coords_x[1]-coords_x[2])
    Δy = abs(coords_y[1]-coords_y[2])

    # calculate the bins in which the x and y values are
    n = round(Int, (x-coords_x[1]) / Δx, RoundDown) + 1
    m = round(Int, (y-coords_y[1]) / Δy, RoundDown) + 1

    # interpolation in x direction on both y-points
    k1 = (A[n, m] - A[n+1, m]) / (coords_x[n] - coords_x[n+1])
    d1 = A[n, m] - k1 * coords_x[n]

    k2 = (A[n, m+1] - A[n+1, m+1]) / (coords_x[n] - coords_x[n+1])
    d2 = A[n, m+1] - k2 * coords_x[n]
```

```
33
34      A1 = k1 * x + d1
35      A2 = k2 * x + d2
36
37      # use the interpolated values on the x-axis for the
        ↪  interpolation in the y axis
38      k = (A1 - A2) / (coords_y[m] - coords_y[m+1])
39      d = A1 - k * coords_y[m]
40
41      # return the interpolated value
42      return k * y + d
43
44  end
45
46
47  """
48      _gradient(V::Matrix{<:AbstractFloat}, x::Vector{<:Real},
49                y::Vector{<:Real})::Tuple{Matrix{<:AbstractFloat},
    ↪  Matrix{<:AbstractFloat}}
50  Calculate the gradient.
51  This function calculates and returns the gradient of a 2D input
    ↪  potential `V`, and will
52  return the Gradient in the form of the tuple corresponding to `(Fx,
    ↪  Fy)`, where `Fx` and `Fy`
53  are Matrices with the same size and datatype as the `V` value. The
    ↪  `x` and `y` values
54  correspond to the coordinate system of the viewed Potentials/Forces.
55  """
56  function _gradient(V::Matrix{<:AbstractFloat}, x::Vector{<:Real},
57                     y::Vector{<:Real})::Tuple{Matrix{
                       ↪  <:AbstractFloat}, Matrix{<:AbstractFloat}}
58
59      # define the output gradient vector field
60      Fx = similar(V)
61      Fy = similar(V)
62
63      # set the boundaries
64      Fx[1, :] .= 0
65      Fy[1, :] .= 0
66      Fx[end, :] .= 0
67      Fy[end, :] .= 0
68      Fx[:, 1] .= 0
```

```
    Fy[:, 1] .= 0
    Fx[:, end] .= 0
    Fy[:, end] .= 0

    # calculate the Δx and Δy value
    Δx = abs(x[1]-x[2])
    Δy = abs(y[1]-y[2])

    # calculate the gradient
    for j = 2:size(V, 2)-1
        for i = 2:size(V, 1)-1
            Fx[i, j] = (V[i+1,j] - V[i-1,j]) / 2 / Δx
            Fy[i, j] = (V[i,j+1] - V[i,j-1]) / 2 / Δy
        end
    end

    return (Fx, Fy)
end
```

lightField.jl

```
# import the image toolbox
using Images

"""
    LightField(intensity::Matrix{<:Real}, norm::Real,
↪  x::Vector{<:Real},
                y::Vector{<:Real}, λ::Real, E::Real)::LightField

Return a Light Field struct.

Create and return a Light Field struct, where the intensity is given
↪  by the
`intensity` parameter, and the normalization factor (integral over
↪  the whole space)
`norm` of the intensity, the coordinates in the `x` and `y`
↪  direction The `λ` parameter
denotes the wavelength of the light field and the parameter `E` is
↪  the pulse energy.
"""
struct LightField
    intensity::Matrix{<:Real}
    norm::Real
```

```
    x::Vector{<:Real}
    y::Vector{<:Real}
    λ::Real
    E::Real
end


"""
    loadintensity(s::String)::Matrix{<:Real}

Return the intensity matrix.

Load the intensity image from the file at `s`. The image will then
be normalized, such that the values range from 0 to 1, and then
 ↪  returned as
a Matrix with the datatype being Float64.
"""
function loadintensity(s::String)::Matrix{<:Real}
    # load the image, save it in a matrix
    input = Float64.(Gray.(load(s)))

    # normalize the image between 0 and 1
    input .-= minimum(input)
    input ./= maximum(input)

    # return the wanted image
    return input
end
```

components.jl

```
abstract type Component end

include("./components/free.jl")
include("./components/lens.jl")
include("./components/pondInteraction.jl")

struct Setup
    setup::Vector{<:Component}
end

"""
    Setup(comps::Component...)::Setup
```

```
13
14  Return the Setup struct.
15
16  Create and return the setup struct. The varargs `comps` describe
17  the different components that are used in the setup.
18  """
19  Setup(comps::Component...) = Setup(collect(comps))
20
21  """
22      propagation!(rays::AbstractElectron, setup::Setup)
23
24  Propagate the electrons in the setup.
25
26  Take the electrons in `rays` and apply all the different alterations
27  that are carried out by the setup denoted by `setup`.
28  """
29  function propagation!(rays::Electron, setup::Setup)
30
31      # iterate over all the components, apply the calculate function
32      for component in setup.setup
33          calculate!(rays, component)
34      end
35  end
```

free.jl

```
1   # the free propagation struct and constructor
2   struct Free <: Component
3       mat::Matrix{<:Real}
4   end
5
6   """
7       Free(d::Real)::Free
8
9   Return the Free type.
10
11  Construct and return the Free type. The `d` value represents
12  the distance that the free propagation should be calculated.
13  """
14  Free(d::Real)::Free = Free([1. 0. d 0.; 0. 1. 0. d; 0. 0. 1. 0.; 0.
    ↪  0. 0. 1.])
15
16  """
```

```
    calculate!(rays::Electron1d, free::Free)

Apply the free propagation on the electrons.

The free propagation is applied to all the differen rays that are contained
in the `rays` object, and applies the effect that is saved in the `free` object.
"""
function calculate!(rays::Electron, free::Free)
    for ray in rays.ψ
        ray[:] = free.mat * ray
    end
end
```

lens.jl

```
struct Lens <: Component
    mat::Matrix{<:Real}
end

"""
    Lens(f::Real)::Lens

Return the Lens type.

Construct and return the lens type. The `f` denotes
the focal length of the lens.
"""
Lens(f::Real)::Lens = Lens([1. 0. 0. 0.; 0. 1. 0. 0.; -1/f 0. 1. 0.; 0. -1/f 0. 1.])


"""
    calculate!(rays::Electron, lens::Lens)

Apply the effect of the lens on the electrons.

This function takes the `Electron1d` object and the `lens` object,
and uses the information provided to apply the effect of the Lens on
the electron beam.
"""
function calculate!(rays::Electron, lens::Lens)
```

```
    for ray in rays.ψ
        ray[:] = lens.mat * ray
    end
end
```

pondInteraction.jl

```
struct PondInteraction <: Component
    lf::LightField
end

"""
    calculate!(ray::Electron, pond::PondInteraction)

Calculate the scattering angle.

Calculate the angle that the electron has after the interaction with the light field
and save it in the ray object. The input parameters are `pond` which is a light field
object, aswell as `ray` which is the electron beams object.
"""
function calculate!(ray::Electron, pond::PondInteraction)

    # get the light field out of the PondInteraction struct
    lf = pond.lf

    # define the electron Energy and momentum
    γ = 1 / sqrt(1 - (ray.v / c)^2)
    β = ray.v / c
    p = γ * m_e * ray.v
    αħ = q^2 / (4 * π * ε_0 * c)
    E_e = γ * m_e * c^2

    # calculate the constant
    constant = - 1 / (1 + β) * αħ / (2 * π) * lf.E / E_e * lf.λ^2 / lf.norm

    gradx, grady = _gradient(lf.intensity, lf.x, lf.y)

    # iterate over the all the electron beams
    for i in eachindex(ray.ψ)

```

```
34          # calculate the momentum change in x and y direction
35          Δpx = constant * _interpolation(gradx, lf.x, lf.y,
            ↪  ray.ψ[i][1], ray.ψ[i][2])
36          Δpy = constant * _interpolation(grady, lf.x, lf.y,
            ↪  ray.ψ[i][1], ray.ψ[i][2])
37
38          # calculate the tangens and a constant
39          tan_α = tan(ray.ψ[i][3])
40          tan_β = tan(ray.ψ[i][4])
41          A = sqrt(1 + tan_α^2 + tan_β^2)
42
43          # calculate the momentum
44          pz = p / A
45          px = pz * tan_α
46          py = pz * tan_β
47
48          # calculate the new angle
49          # the complex - real transformation is to prevent floating
            ↪  point errors
50          # due to floating point differences not being 0 when they
            ↪  should be
51          ray.ψ[i][3] = atan((px + Δpx) /
52                             real(sqrt(complex(pz^2 * A^2 - (px +
                               ↪  Δpx)^2 - (py + Δpy)^2))))
53          ray.ψ[i][4] = atan((py + Δpy) /
54                             real(sqrt(complex(pz^2 * A^2 - (px +
                               ↪  Δpx)^2 - (py + Δpy)^2))))
55      end
56  end
```

A.2.2. Application

In this section the jupyter notebook is presented that was used to conduct the simulation and plot the results in section 3.3. The code is also available on GitHub and can be found under `https://github.com/JuffmannLab/MasterSimulations`.

```
1 using Raytracing
2 using PyPlot
3 using Images
```

```
1 rcParams = PyPlot.PyDict(PyPlot.matplotlib."rcParams")
2 rcParams["font.size"] = 10
```

```
minval = 0.5
maxval = 1.0
dpi = 200
cmap = get_cmap("PuOr")
violett = PyPlot.matplotlib.colors.LinearSegmentedColormap.from_list("trunc("*cmap.name*", "*string(minval)*", "*string(maxval)*")", cmap(range(minval, maxval, length=cmap.N)))
cmap = get_cmap("seismic")
red = PyPlot.matplotlib.colors.LinearSegmentedColormap.from_list("trunc("*cmap.name*", "*string(minval)*", "*string(maxval)*")", cmap(range(minval, maxval, length=cmap.N)))
;
```

```
c = 299792458           # the speed of light
ħ = 1.054571817e-34     # the reduced planck constant
m_e = 9.1093837015e-31  # the electron mass
q = 1.602176634e-19     # electron charge
ε_0 = 8.8541878128e-12  # vacuum permitivity
;
```

```
xmin = -8e-6
xmax =  8e-6
ymin = -8e-6
ymax =  8e-6

nx = 501
ny = 501

debeam = 15e-6

x = Vector{Float64}(range(xmin, xmax, length=nx))
y = Vector{Float64}(range(ymin, ymax, length=ny))

aperture = [x[i]^2+y[j]^2 < debeam^2/4 ? 1. : 0. for i in eachindex(x), j in eachindex(y)]

```

```
img = Float64.(imresize(Gray.(-1 *
↪  load("./images/sample_image.png")), (501, 501))) .+ 1

inint = img .* aperture
inint ./= maximum(inint)

inputextent = [x[1], x[end], y[1], y[end]] .* 1e6

figsize = [7, 7] ./ 2.54

pygui(false)
fig, ax = subplots(figsize=figsize, dpi=dpi)

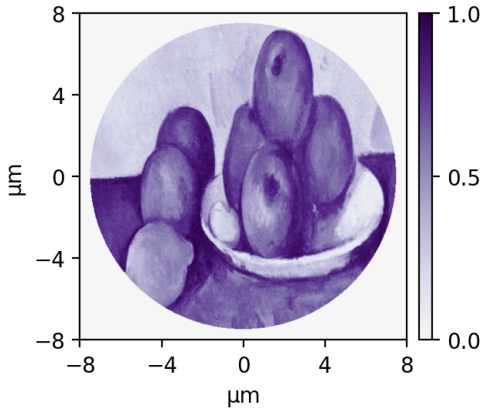
plt = ax.imshow(inint, cmap=violett, extent=inputextent)

cax = fig.add_axes([ax.get_position().x1+0.03,
↪  ax.get_position().y0,0.03,ax.get_position().height])
fig.colorbar(plt, ax=ax, shrink=0.8, ticks=[0, 0.5, 1], cax=cax)

ax.set_xlabel("μm")
ax.set_ylabel("μm")

ax.set_xticks([-8, -4, 0, 4, 8])
ax.set_yticks([-8, -4, 0, 4, 8])

savefig("./images/lens/input.svg", bbox_inches="tight", dpi=dpi)
```




```
U = 30e3
```

```
z = 0.5
λ_L = 1035e-9
N = 50e6

f = 0.01

v = c * sqrt(1 - 1 / (1 + q*U/m_e/c^2)^2)
α = 1 / (4 * π * ε_0) * q^2 / (ħ * c)
β = v / c
γ = 1 / sqrt(1 - β^2)
Ee = γ*m_e*c^2
λ_e = 2 * π * ħ / (γ * m_e * v)
;
```

```
# LG10 effective lens size: 0.2295 * w
lg10(ρ, w) = @. 2 * ρ^2 / w^2 * exp(-2*ρ^2/w^2)
e10(f, w) = (1+β) * π^3 * w^4 * Ee / (2 * α * λ_L^2 * λ_e) / f
w10(f, E) = ( 2 * α * λ_L^2 * λ_e * f * E / ( (1+β) * π^3 * Ee)
↪  )^(1/4)

# modified LG10 effective lens size: 0.5157 * w
modLG(ρ, w) = @. 2 * ρ^2 / w^2 * (1 + 2 * ρ^2 / w^2) *
↪  exp(-2*ρ^2/w^2)
emod(f, w) = (1+β) * 3 * π^3 * w^4 * Ee / (2 * α * λ_L^2 * λ_e)
↪  / f
wmod(f, E) = ( 2 * α * λ_L^2 * λ_e * f * E / ( 3*(1+β) * π^3 *
↪  Ee) )^(1/4)
;
```

```
fov = 120e-6
Δxin = 0.1e-6
Δyin = 0.1e-6

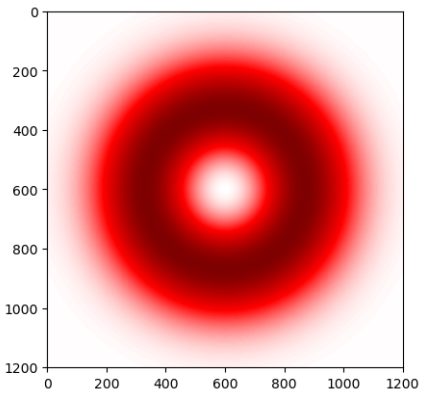
w = 30e-6

xin = Vector{Float64}(range(-fov/2, fov/2, step=Δxin))
yin = Vector{Float64}(range(-fov/2, fov/2, step=Δyin))

ρ = @. sqrt(xin^2 + yin'^2)

intextent = [xin[1], xin[end], yin[1], yin[end]] .* 1e6
```

```
13
14 imshow(modLG(ρ, w), cmap=red)
15 ;
```




```
1 # perfect lens simulation
2 eb = Electron(x, y, inint, U, Int(N))
3 lens = Lens(f)
4 free = Free(z)
5 setup = Setup(lens, free)
6 propagation!(eb, setup)
7 ;
```

```
1 fovimg = 0.8e-3
2 Δximg = 1e-6
3 ximg = Vector{Float64}(range(-fovimg/2, fovimg/2-Δximg,
  ↪  step=Δximg))
4 yimg = copy(ximg)
5
6 out_int = getintensity(eb, ximg, yimg)
7
8 out_int ./= maximum(out_int)
9
10 extent = [ximg[1], ximg[end], yimg[1], yimg[end]] .* 1e6
11
12 figsize = [7, 7] ./ 2.54
13
14 pygui(false)
15 fig, ax = subplots(figsize=figsize, dpi=dpi)
16
```

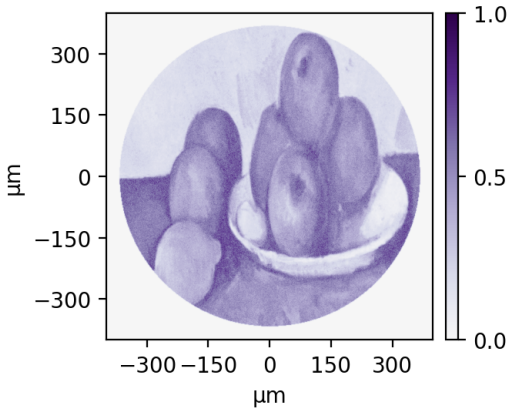
```
plt = ax.imshow(out_int[end:-1:1, end:-1:1], cmap=violett,
    extent=extent)

cax = fig.add_axes([ax.get_position().x1 + 0.03,
    ax.get_position().y0,0.03, ax.get_position().height])
fig.colorbar(plt, ax=ax, shrink=0.8, ticks=[0, 0.5, 1], cax=cax)

ax.set_xlabel("μm")
ax.set_ylabel("μm")

ax.set_xticks([-300, -150, 0, 150, 300])
ax.set_yticks([-300, -150, 0, 150, 300])

savefig("./images/lens/perfect.svg", bbox_inches="tight",
    dpi=dpi)
;
```




```
# lg10 lens simulation
E = 1.28e-3
w = 25e-6
intensity = lg10(ρ, w)
norm = sum(intensity) * Δxin * Δyin
E = e10(f, w)
lf = LightField(intensity, norm, xin, yin, λ_L, E)

eb = Electron(x, y, inint, U, Int(N))
lens = PondInteraction(lf)
free = Free(z)
```

```
setup = Setup(lens, free)
propagation!(eb, setup)
;
```

```
display(E*1e6)
out_int = getintensity(eb, ximg, yimg)

out_int ./= maximum(out_int)

figsize = [7, 7] ./ 2.54

pygui(false)
fig, ax = subplots(figsize=figsize, dpi=dpi)

plt = ax.imshow(out_int[end:-1:1, end:-1:1], cmap=violett,
↪  extent=extent)

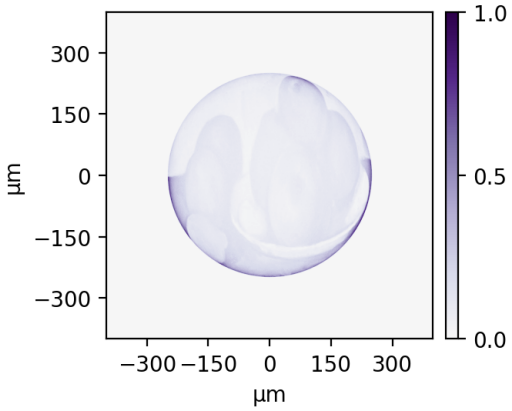
cax = fig.add_axes([ax.get_position().x1 + 0.03,
↪  ax.get_position().y0,0.03, ax.get_position().height])
fig.colorbar(plt, ax=ax, shrink=0.8, ticks=[0, 0.5, 1], cax=cax)

ax.set_xlabel("μm")
ax.set_ylabel("μm")

ax.set_xticks([-300, -150, 0, 150, 300])
ax.set_yticks([-300, -150, 0, 150, 300])

savefig("./images/lens/lg10.svg", bbox_inches="tight", dpi=dpi)
;
```

```
1278.095327823601
```




```
# lg10 lens simulation
w = 19e-6
E = 1.28e-3
intensity = modLG(ρ, w)
norm = sum(intensity) * Δxin * Δyin
E = emod(f, w)
lf = LightField(intensity, norm, xin, yin, λ_L, E)

eb = Electron(x, y, inint, U, Int(N))
lens = PondInteraction(lf)
free = Free(z)
setup = Setup(lens, free)
propagation!(eb, setup)
;
```

```
display(E*1e6)
out_int = getintensity(eb, ximg, yimg)

out_int ./= maximum(out_int)

figsize = [7, 7] ./ 2.54

pygui(false)
fig, ax = subplots(figsize=figsize, dpi=dpi)

plt = ax.imshow(out_int[end:-1:1, end:-1:1], cmap=violett,
↪  extent=extent)

```

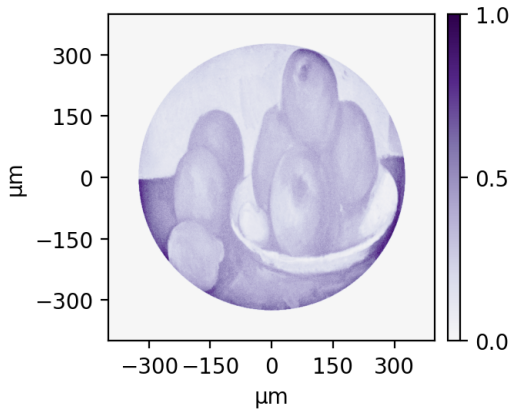
```
cax = fig.add_axes([ax.get_position().x1 + 0.03,
↪  ax.get_position().y0, 0.03,ax.get_position().height])
fig.colorbar(plt, ax=ax, shrink=0.8, ticks=[0, 0.5, 1], cax=cax)

ax.set_xlabel("μm")
ax.set_ylabel("μm")

ax.set_xticks([-300, -150, 0, 150, 300])
ax.set_yticks([-300, -150, 0, 150, 300])

savefig("./images/lens/modlg.svg", bbox_inches="tight", dpi=dpi)
;
```

```
1279.2012381488605
```



A.3. Waveoptics

A.3.1. Code

The package that was created based on the information given in chapter 4 is given in this section. Alternatively this code can be found on GitHub under `https://github.com/JuffmannLab/ElectronPropagation`. The folder structure can be seen in the following diagram, where the Manifest.toml and the Project.toml file are machine generated.

```
ElectronPropagation/
├── LICENSE
├── Manifest.toml
├── Project.toml
├── README.md
└── src/
    ├── ElectronPropagation.jl
    ├── electronBeam.jl
    ├── laserBeam.jl
    ├── components.jl
    └── components/
        ├── aperture.jl
        ├── lens.jl
        ├── freePropagation.jl
        └── phaseImprint.jl
```

ElectronPropagation.jl

```
module ElectronPropagation

# export the different kind of waves
export ElectronBeam, LaserBeam, deBroglieWavelength

# export the different kinds of setup possibilities
export Free, PhaseImprint, Aperture, Lens
export Setup, propagation!

# define some physical constants
global const c = 299792458            # the speed of light
global const ħ = 1.054571817e-34      # the reduced planck constant
global const m_e = 9.1093837015e-31   # the electron mass
global const q = 1.602176634e-19      # electron charge
global const ε_0 = 8.8541878128e-12   # vacuum permitivity

# include all the needed code
```

```
include("./electronBeam.jl")
include("./laserBeam.jl")
include("./components.jl")

# end module
end
```

electronBeam.jl

```
"""
    ElectronBeam(ψ::Matrix{<:Complex}, λ::Real, v::Real,
↪  x::Vector{<:Real}, y::Vector{<:Real})

Return the ElectronBeam type.

In the ElectronBeam type the wave function `ψ`, the de Broglie
↪  wavelength `λ`,
the electron velocity `v` as well as the coordinates `x` and `y` are
↪  saved.

# Example
```jldoctest
julia> ψ = ones(ComplexF64, 2, 2)
2×2 Matrix{ComplexF64}:
 1.0+0.0im  1.0+0.0im
 1.0+0.0im  1.0+0.0im

julia> ElectronBeam(ψ, 1e-12, 1e8, [1, 2], [1, 2])
ElectronBeam(ComplexF64[1.0 + 0.0im 1.0 + 0.0im; 1.0 + 0.0im 1.0 +
↪  0.0im], 1.0e-12, [1, 2], [1, 2])

```

See also: [`deBroglieWavelength`] [`LaserBeam`](@ref)
"""
mutable struct ElectronBeam
    ψ::Matrix{<:Complex}
    λ::Real
    v::Real
    x::Vector{<:Real}
    y::Vector{<:Real}
end

```

```
31
32 """
33   ElectronBeam(ψ::Matrix{<:Complex}, U::Real, x::Vector{<:Real},
    ↪   y::Vector{<:Real})
34
35 Return the ElectronBeam type.
36
37 Calculate the de Broglie wavelength and the velocity, and returning
    ↪   the corresponding
38 ElectronBeam type. The wavefunction is `ψ`, the coordinates are `x`
    ↪   and `y`.
39 `U` is the electron acceleration voltage.
40
41 # Example
42 ```jldoctest
43 julia> ψ = ones(ComplexF64, 2, 2)
44 2×2 Matrix{ComplexF64}:
45   1.0+0.0im 1.0+0.0im
46   1.0+0.0im 1.0+0.0im
47
48 julia> ElectronBeam(ψ, 30e3, [1, 2], [1, 2])
49 ElectronBeam(ComplexF64[1.0 + 0.0im 1.0 + 0.0im; 1.0 + 0.0im 1.0 +
    ↪   0.0im], 6.979081574270336e-12, 9.844470106002943e7, [1, 2], [1,
    ↪   2])
50
51 ```
52
53 See also: [`LaserBeam`](@ref)
54 """
55 function ElectronBeam(ψ::Matrix{<:Complex}, U::Real,
    ↪   x::Vector{<:Real}, y::Vector{<:Real})
56   λ = deBroglieWavelength(U)
57   v = c * sqrt(1 - 1 / (1 + q*U/m_e/c^2)^2)
58   return ElectronBeam(ψ, λ, v, x, y)
59 end
60
61
62 """
63   deBroglieWavelength(U::Real)
64
65 Return the de Broglie wavelength of an electron.
66
```

```
This function calculates the relativistic de Broglie wavelength of
an electron using the acceleration Voltage `U` as an input.
The output unit will be in meters.

# Example
```jldoctest
julia> deBroglieWavelength(30e3)
6.979081574270336e-12
```

See also: [`ElectronBeam`](@ref)
"""
function deBroglieWavelength(U::Real)::Real
    return 2 * π * ħ / sqrt(2 * U * q * m_e) / sqrt(1 + q * U / 2 /
    ↪ m_e / c^2)
end
```

laserBeam.jl

```
"""
    LaserBeam(I::Matrix{<Real}, λ::Real, E::Real, x::Vector{<:Real},
↪  y::Vector{<:Real})

Return the LaserBeam type.

Save the intensity distribution `I`, the wavelength `λ`, the pulse
↪  energy `E`
and the coordinates `x` and `y` in the LaserBeam struct that is
↪  returned.

# Example
```jldoctest
julia> I = rand(Float64, 2, 2)
2×2 Matrix{Float64}:
 0.956952  0.353413
 0.52275   0.874929

julia> LaserBeam(I, 1e-6, 1e-6, [1, 2], [1, 2])
LaserBeam([0.9569521001448673 0.35341262246269656;
↪  0.5227499716351234 0.8749291118118734], 1.0e-6, 1.0e-6, [1, 2],
↪  [1, 2])
```

```

```
See also: [`ElectronBeam`](@ref)
"""
mutable struct LaserBeam
    I::Matrix{<:Real}
    λ::Real
    E::Real
    x::Vector{<:Real}
    y::Vector{<:Real}
end
```

components.jl

```
# create the abstract type of the component
abstract type Component end

# include all the components
include("./components/aperture.jl")
include("./components/freePropagation.jl")
include("./components/lens.jl")
include("./components/phaseImprint.jl")

# The setup struct
struct Setup
    setup::Array{<:Component}    # an array of components
end

"""
    Setup(comps::Component...)::Setup

Return the setup.

Create and return the setup type. This type will save the current
components in the order that they should be calculated. The
↪ components
are in the varargs `comps`.

See also: [`propagation!`](@ref)
"""
Setup(comps::Component...) = Setup(collect(comps))

"""
    propagation!(wave::ElectronBeam, setup::Setup)

```

```
Propagate the setup.

This function propagates the given wave `wave` through the given setup `setup`.
The wave will be altered according to the propagation.

See also: [`Setup`](@ref)
"""
function propagation!(wave::ElectronBeam, setup::Setup)
    for comp in setup.setup
        calculate!(wave, comp)
    end
end
```

aperture.jl

```
"""
    Aperture(d::Real)::Aperture

Return the Aperture type.

This struct simulates an aperture, where `d` is the
diameter of the aperture.

# Example
```jldoctest
julia> Aperture(1)
Aperture(1)
```

See also: [`Free`](@ref), [`PhaseImprint`](@ref), [`Lens`](@ref)
"""
struct Aperture <: Component
    d::Real
end

"""
    Calculate the aperture.

    Apply the aperture to a given wavefunction.
"""
function calculate!(wave::ElectronBeam, aperture::Aperture)
    for i = 1:size(wave.x, 1), j = 1:size(wave.y, 1)
```

```
28          if wave.x[i]^2+wave.y[j]^2 <= (aperture.d/2)^2
29          else
30              wave.ψ[i, j] = 0
31          end
32      end
33  end
```

lens.jl

```
1   """
2       Lens(f::Real)::Lens
3
4   Return a Lens type.
5
6   Here a lens type is returned. The `f` value is the focal length of
    ↪  the lens.
7
8   # Example
9   ```jldoctest
10  julia> Lens(1)
11  Lens(1)
12  ```
13
14  See also: [`Free`](@ref), [`PhaseImprint`](@ref), [`Aperture`](@ref)
15  """
16  struct Lens <: Component
17      f::Real    # focal length of the lense
18  end
19
20  """
21      calculate!(wave::Wave, lens::Lens)
22
23  Calculate the lense.
24
25  This function simulates the effect of a perfect lens on the
    ↪  wavefunction.
26  """
27  function calculate!(wave::ElectronBeam, lens::Lens)
28      if lens.f == 0
29          return
30      else
31          wave.ψ .*= @. exp(-1im * π / wave.λ / lens.f * (wave.x^2 +
            ↪  wave.y'^2))
```

```
    end
end
```

freePropagation.jl

```
# import statements
using FFTW

struct Free <: Component
    transferfunction::Matrix{<:Complex}
end

"""
    Free(wave::Wave, distance::Real)::Free

Return the Free type.

Create and return the PropTf struct that is needed to calculate the
wavefunction a given distance `distance` away. It also needs the
`wave`, which is an ElectronBeam type. Be aware that
for this function to calculate something meaningfull, sampling
must be fulfilled!

# Example
```jldoctest
julia> eb = ElectronBeam(Array{Float64}(1:2), Array{Float64}(1:2),
↪  1);

julia> Free(eb, 1)
Free(Complex{Float64}[1.0 - 1.926465401546721e-9im 1.0 -
↪  1.926465401546721e-9im;
1.0 - 1.926465401546721e-9im 1.0 - 1.926465401546721e-9im])
```

See also: [`Aperture`](@ref), [`PhaseImprint`](@ref), [`Lens`](@ref)
"""
function Free(wave::ElectronBeam, distance::Real)::Free

    # get the x axis out of the wave struct
    x = wave.x
    y = wave.y
    λ = wave.λ

```

```
37   # get some information out of the transverse coordinates
38   Δx = abs(x[1] - x[2])
39   Δy = abs(y[1] - y[2])
40
41   # define the frequency coordinates
42   fx = Vector{Float64}(range(-1/(2*Δx), 1/(2*Δx), length=size(x,
        ↪ 1)))
43   fy = Vector{Float64}(range(-1/(2*Δy), 1/(2*Δy), length=size(y,
        ↪ 1)))
44
45   # fill the transferfunction arrays with the proper values
46   transferfunction = @. exp(-1im * π * λ * distance * (fx^2 +
        ↪ fy'^2))
47
48   # shift the transferfunction and return the Free object
49   return Free(fftshift(transferfunction))
50 end
51
52
53 """
54   calculate!(wave::Wavem free::Free)
55
56 Calculate the Free.
57
58 This function calculates the light field in a given direction. After
    ↪ the
59 calculation the changed LightField object is returned.
60 """
61 function calculate!(wave::ElectronBeam, free::Free)
62
63   # Fouriertransform the input wave
64   Ψ = fft(fftshift(wave.ψ))
65
66   # apply the transferfunction
67   Ψ .*= free.transferfunction
68
69   # calculate the inverse fouriertransform
70   wave.ψ = ifftshift(ifft(Ψ))
71 end
```

phaseImprint.jl

```
1  using FFTW
```

```
2
3   """
4       PhaseImprint(lb::LaserBeam)::PhaseImprint
5
6   Return the PhaseImprint type.
7
8   The PhaseImprint type is generated and returned. The `lb`
9   value has to be a LaserBeam type, and represents the
10  laser beam that will make the phase imprint.
11
12  # Example
13  ```jldoctest
14  julia> lb = LaserBeam(Array(1:2), Array(1:2), Array(1:2), 1035e-9,
    ↪  15e-6, 40e-6, 280e-15);
15
16  julia> PhaseImprint(lb)
17  PhaseImprint(LaserBeam([0 0; 0 0], [1, 2], [1, 2], [1, 2],
    ↪  1.8199532051293268e15, 4.0e-5, 2.8e-13))
18  ```
19
20  See also: [`Free`](@ref), [`Aperture`](@ref), [`Lens`](@ref),
    ↪  [`ElectronBeam`](@ref)
21  """
22  struct PhaseImprint <: Component
23      lb::LaserBeam
24  end
25
26
27  """
28  Imprint the phase.
29
30  This function imprints a phase on the electron wave.
31  This function is only working if the coordinates of the wavefunction
32  are the same in x and y direction! This function only works if the
33  x and y components of the laser beam are the same as the x and y
    ↪  coordinates
34  of the wave object.
35  """
36  function calculate!(wave::ElectronBeam, imprint::PhaseImprint)
37      Δx = abs(imprint.lb.x[1] - imprint.lb.x[2])
38      Δy = abs(imprint.lb.y[1] - imprint.lb.y[2])
39
```

```
    β = wave.v / c
    γ = 1 /sqrt(1-β^2)

    Ee = γ * m_e * c^2
    α = q^2 / (ħ * c) / (4 * π * ε_0)

    norm = sum(imprint.lb.I) * Δx * Δy

    prefactor = - α / (2*π * (1 + β)) * imprint.lb.E / Ee *
    ↪  imprint.lb.λ^2 / norm

    for j in eachindex(wave.y), i in eachindex(wave.x)
        wave.ψ[i, j] *= exp(1im * prefactor *
        ↪  interpolation(imprint.lb.I, imprint.lb.x, imprint.lb.y,
        ↪  wave.x[i], wave.y[j]))
    end
end


"""
    interpolation(A::Matrix{<:Real}, coords_x::Vector{<:Real},
                  coords_y::Vector{<:Real}, x::Real, y::Real)::Real

Return the bilinear interpolation.
Calculate and return the bilinear interpolation on the Matrix `A`
with the coordinates `coords_x` and `coords_y` at the point with
the position `x`, `y`.
"""
function interpolation(A::Matrix{<:Real}, coords_x::Vector{<:Real},
                       coords_y::Vector{<:Real}, x::Real,
                       ↪  y::Real)::Real

    # if the value is not in the area of coords return 0
    if x <= coords_x[1]  x >= coords_x[end]
        return zero(eltype(A))
    elseif y <= coords_y[1]  y >= coords_y[end]
        return zero(eltype(A))
    end

    # calculate the Δx and Δy values
    Δx = abs(coords_x[1]-coords_x[2])
    Δy = abs(coords_y[1]-coords_y[2])
```

```
78
79      # calculate the bins in which the x and y values are
80      n = round(Int, (x-coords_x[1]) / Δx, RoundDown) + 1
81      m = round(Int, (y-coords_y[1]) / Δy, RoundDown) + 1
82
83      # interpolation in x direction on both y-points
84      k1 = (A[n, m] - A[n+1, m]) / (coords_x[n] - coords_x[n+1])
85      d1 = A[n, m] - k1 * coords_x[n]
86
87      k2 = (A[n, m+1] - A[n+1, m+1]) / (coords_x[n] - coords_x[n+1])
88      d2 = A[n, m+1] - k2 * coords_x[n]
89
90      A1 = k1 * x + d1
91      A2 = k2 * x + d2
92
93      # use the interpolated values on the x-axis for the
        ↪  interpolation in the y axis
94      k = (A1 - A2) / (coords_y[m] - coords_y[m+1])
95      d = A1 - k * coords_y[m]
96
97      # return the interpolated value
98      return k * y + d
99  end
```

A.3.2. Application

In this section the Notebooks of the wave optics simulations are provided. The code can also be seen on GitHub at `https://github.com/JuffmannLab/MasterSimulations`.

Gaussian beam

The calculations conducted in section 4.3.1 can be found in the following notebook.

```
1 using ElectronPropagation
2 using PyPlot
3 using FFTW
4 smoothe(A, x, y, σ) = ifftshift(ifft(fft(A) .* fft(@.
  ↪  exp(-(x^2+y'^2)/σ^2))))
5 ;
```

```
1 rcParams = PyPlot.PyDict(PyPlot.matplotlib."rcParams")
2 rcParams["font.size"] = 10
3 minval = 0.5
```

```
maxval = 1.0
dpi = 200
cmap = get_cmap("PuOr")
violett = PyPlot.matplotlib.colors.LinearSegmentedColormap.from_list(
            "trunc("*cmap.name*", "*string(minval)*", "*string(maxval)*")",
            cmap(range(minval, maxval, length=cmap.N)))
cmap = get_cmap("seismic")
red = PyPlot.matplotlib.colors.LinearSegmentedColormap.from_list(
            "trunc("*cmap.name*", "*string(minval)*", "*string(maxval)*")",
            cmap(range(minval, maxval, length=cmap.N)))
norm = PyPlot.matplotlib.colors.Normalize(vmin=0, vmax=1)
;
```

```
U = 30e3

L = 50e-6
z = 0.5
w = 15e-6  # electron beam radius

λ = deBroglieWavelength(U)

# sampling
Δx = λ * z / L / 4

x = Vector{Float64}(range(-L/2, L/2-Δx, step=Δx))
y = copy(x)
;
```

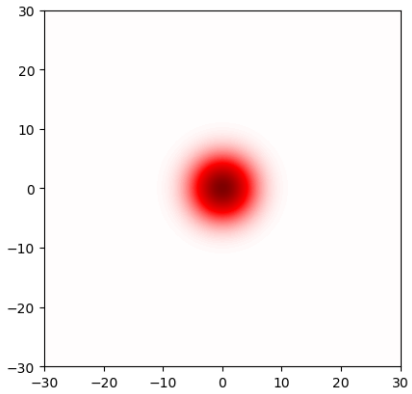
```
λ_l = 1035e-9
Δimg = 0.01e-6

wgauss = 5e-6

ximg = Vector{Float64}(range(-30e-6, 30e-6, step=Δimg))
yimg = ximg

I = @. exp(-(ximg^2+yimg'^2)/wgauss^2)
```

```
extent_l = [ximg[1], ximg[end], yimg[1], yimg[end]] .* 1e6
pygui(false)
imshow(I, extent=extent_l, cmap=red)
;
```




```
n, m = size(I)
I_cross = I[3001, :]
offset = 0.1
func(x) = x > 0.5
bounds = findall(func, I_cross)
display(ximg[bounds[end]]-ximg[bounds[1]])
```

```
8.319999999999998e-6
```

```
c = 299792458           # the speed of light
ħ = 1.054571817e-34     # the reduced planck constant
m_e = 9.1093837015e-31  # the electron mass
q = 1.602176634e-19     # electron charge
ε_0 = 8.8541878128e-12  # vacuum permitivity

v = c * sqrt(1 - 1 / (1 + q*U/m_e/c^2)^2)
α = 1 / (4 * π * ε_0) * q^2 / (ħ * c)
β = v / c
γ = 1 / sqrt(1 - β^2)
Ee = γ*m_e*c^2
;
```

```
# get the energy needed for the phase shift φ
φ = 1*π
E = -φ * 2 * π * (1 + β) / α / λ_l^2 * Ee * sum(I) * Δimg^2

display(E)

lb = LaserBeam(I, λ_l, E, ximg, yimg)

ψ = smoothe([i^2 + j^2 < w^2 ? complex(1.) : complex(0.) for i
↪  in x, j in y], x, y, 1e-6)
extent_e = [x[1], x[end], y[1], y[end]].*1e6

eb = ElectronBeam(copy(ψ), U, x, y)
ebfree = ElectronBeam(copy(ψ), U, x, y)

phase_imprint = PhaseImprint(lb)
free = Free(eb, z)

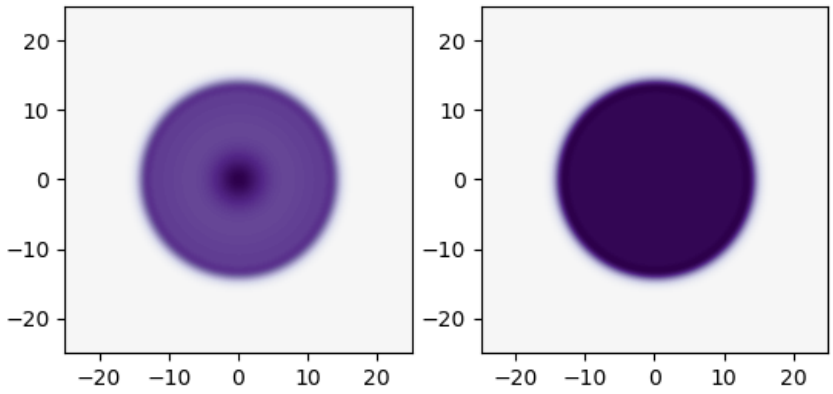
setup1 = Setup(phase_imprint, free)
setup2 = Setup(free)

propagation!(eb, setup1)
propagation!(ebfree, setup2)

out_int = abs2.(eb.ψ)

pygui(false)
fig, (ax1, ax2) = subplots(1, 2)
ax1.imshow(out_int, cmap=violett, extent=extent_e)
ax2.imshow(abs2.(ebfree.ψ), cmap=violett, extent=extent_e)
;
```

```
-2.283502477459125e-8
```




```
out = out_int ./ maximum(out_int)
out[1, 1] = 0

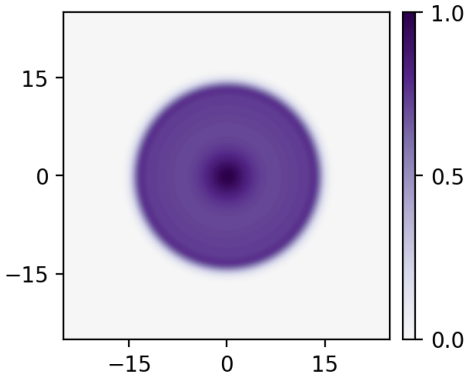
figsize = [7, 7] ./ 2.54

pygui(false)
fig, ax = subplots(figsize=figsize, dpi=dpi)
plt = ax.imshow(out, cmap=violett, extent=extent_e)

ax.set_xticks([-15, 0, 15])
ax.set_yticks([-15, 0, 15])

cax = fig.add_axes([ax.get_position().x1 + 0.03,
↪  ax.get_position().y0,0.03, ax.get_position().height])
fig.colorbar(plt, ax=ax, shrink=0.8, ticks=[0, 0.5, 1], cax=cax)

savefig("./images/pi_gauss/electron_intensity.svg",
↪  bbox_inches="tight", dpi=dpi)
;
```




```
int_plot = I
int_plot[1, 1] = 0
int_plot ./= maximum(int_plot)

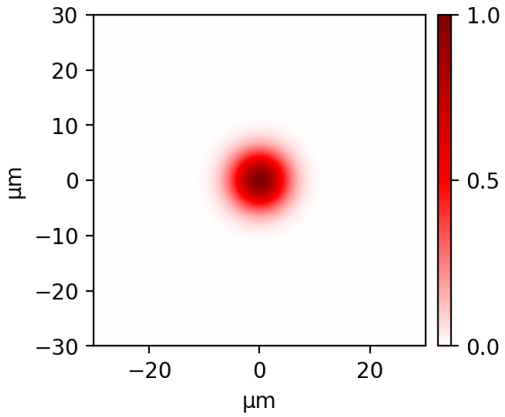
figsize = [7, 7] ./ 2.504
extent = [yimg[1], yimg[end], ximg[1], ximg[end]].*1e6

pygui(false)
fig, ax = subplots(figsize = figsize, dpi=dpi)
plt = ax.imshow(I, cmap=red, extent=extent)

ax.set_xlabel("μm")
ax.set_ylabel("μm")

cax = fig.add_axes([ax.get_position().x1 + 0.03,
↪  ax.get_position().y0,0.03, ax.get_position().height])
fig.colorbar(plt, ax=ax, shrink=0.8, ticks=[0, 0.5, 1], cax=cax)

savefig("./images/pi_gauss/gaussian_intensity.svg",
↪  bbox_inches="tight", dpi=dpi)
;
```




```
int_plot = I
int_plot[1, 1] = 0
int_plot ./= maximum(int_plot)

eb_plane = ElectronBeam(ones(ComplexF64, size(I)), U, ximg, yimg)
setup = Setup(PhaseImprint(lb))
propagation!(eb_plane, setup)

phase = angle.(eb_plane.ψ)

figsize = [7, 7] ./ 2.504
extent = [yimg[1], yimg[end], ximg[1], ximg[end]].*1e6

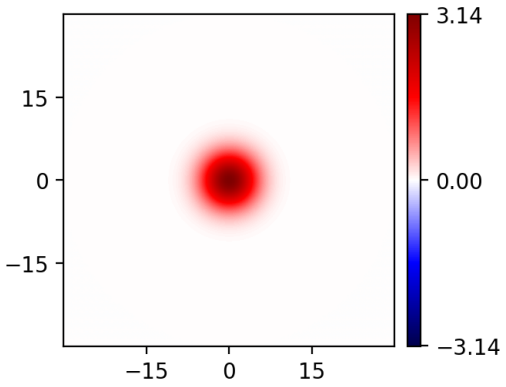
pygui(false)
norm = PyPlot.matplotlib.colors.Normalize(vmin=-maximum(abs.(phase)), vmax=maximum(abs.(phase)))
fig, ax = subplots(figsize = figsize, dpi=dpi)
plt = ax.imshow(angle.(eb_plane.ψ), cmap="seismic", extent=extent, norm=norm)

ax.set_xticks([-15, 0, 15])
ax.set_yticks([-15, 0, 15])


cax = fig.add_axes([ax.get_position().x1 + 0.03, ax.get_position().y0,0.03, ax.get_position().height])
```

```
fig.colorbar(plt, ax=ax, shrink=0.8, ticks=[-3.14, 0, 3.14], cax=cax)

savefig("./images/pi_gauss/gaussian_phase.svg", bbox_inches="tight", dpi=dpi)
;
```




```
using Array2file

ray_radx = readfile("./data/radx.txt", Float64)
ray_xcross = readfile("./data/xcross.txt", Float64) .* 1e6

ray_radx ./= maximum(ray_radx)

cross = out[round(Int, size(out_int, 1)/2), round(Int, size(out_int, 2)/2):end]
xcross = x[round(Int, size(out_int, 2)/2):end] .* 1e6

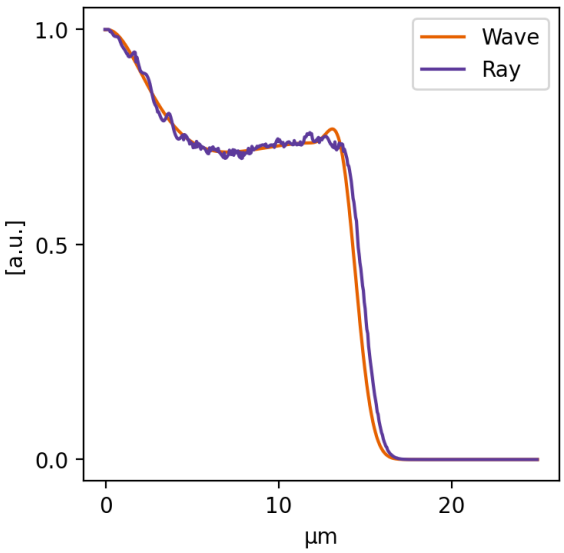
figsize = [10, 10] ./ 2.504
fig, ax = subplots(figsize=figsize, dpi=dpi)

ax.plot(xcross, cross, label="Wave", color="#E66100")
ax.plot(ray_xcross, ray_radx, label="Ray", color="#5D3A9B")

ax.set_xticks([0, 10, 20])
ax.set_yticks([0, 0.5, 1])

ax.set_xlabel("μm")
ax.set_ylabel("[a.u.]")
```

```
22
23 ax.legend()
24
25 savefig("./images/pi_gauss/radial_comparison.svg",
   ↪  bbox_inches="tight", dpi=dpi)
26
27 ;
```



The data that has been loaded in the last notebook box comes from the ray tracing cross check that was conducted. The notebook that is responsible for this calculation is the following:

```
1 using Raytracing
2 using PyPlot
3 using FFTW
4 using BenchmarkTools
5
6 smoothe(A, x, y, σ) = real.(ifftshift(ifft(fft(A) .* fft(@.
   ↪  exp(-(x^2+y'^2)/σ^2)))))
7 ;
```

```
1 rcParams = PyPlot.PyDict(PyPlot.matplotlib."rcParams")
2 rcParams["font.size"] = 10
3 minval = 0.5
4 maxval = 1.0
```

```
dpi=200
cmap = get_cmap("PuOr")
violett =
↪  PyPlot.matplotlib.colors.LinearSegmentedColormap.from_list(
            "trunc("*cmap.name*", "*string(minval)*",
            ↪  "*string(maxval)*")",
            cmap(range(minval, maxval, length=cmap.N)))
cmap = get_cmap("seismic")
red =
↪  PyPlot.matplotlib.colors.LinearSegmentedColormap.from_list(
            "trunc("*cmap.name*", "*string(minval)*",
            ↪  "*string(maxval)*")",
            cmap(range(minval, maxval, length=cmap.N)))
;
```

```
U = 30e3

L = 50e-6
z = 0.5
w = 15e-6

Δx = 50e-9

x = Vector{Float64}(range(-L/2, L/2-Δx, step=Δx))
y = copy(x)

extent_e = [x[1], x[end], y[1], y[end]] .* 1e6
;
```

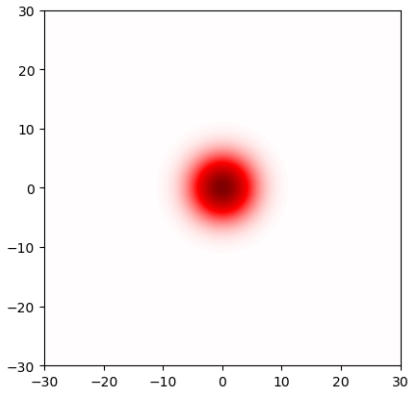
```
λ_l = 1035e-9
Δimg = 0.01e-6

wgauss = 5e-6

ximg = Vector{Float64}(range(-30e-6, 30e-6, step=Δimg))
yimg = ximg

I = @. exp(-(ximg^2+yimg'^2)/wgauss^2)
extent_l = [ximg[1], ximg[end], yimg[1], yimg[end]] .* 1e6
pygui(false)
imshow(I, extent=extent_l, cmap=red)
```

```
;
```




```
c = 299792458            # the speed of light
ħ = 1.054571817e-34      # the reduced planck constant
m_e = 9.1093837015e-31   # the electron mass
q = 1.602176634e-19      # electron charge
ε_0 = 8.8541878128e-12   # vacuum permitivity

v = c * sqrt(1 - 1 / (1 + q*U/m_e/c^2)^2)
α = 1 / (4 * π * ε_0) * q^2 / (ħ * c)
β = v / c
γ = 1 / sqrt(1 - β^2)
Ee = γ*m_e*c^2
;
```

```
# get the energy needed for the phase shift φ
φ = 1*π
norm = sum(I) * Δimg^2
E = -φ * 2 * π * (1 + β) / α / λ_l^2 * Ee * norm

ψ = smoothe([i^2 + j^2 < w^2 ? 1. : 0. for i in x, j in y], x, y,
↪  1e-6)

eb = Electron(x, y, ψ, U, Int(20e6))
lf = LightField(I, norm, ximg, yimg, λ_l, E)

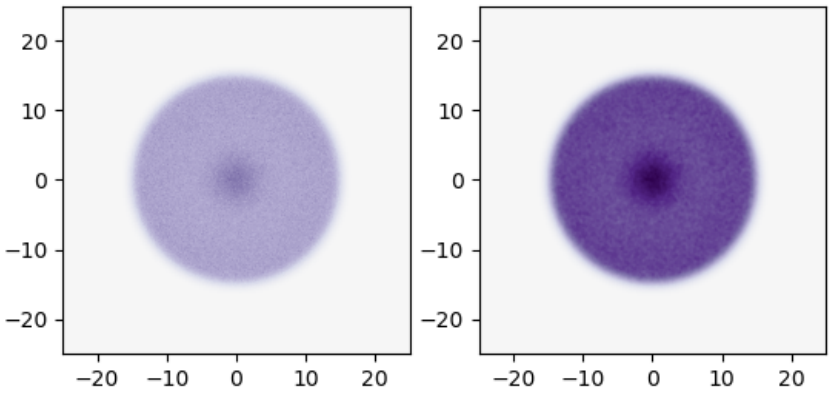
pond = PondInteraction(lf)
free = Free(z)

```

```
setup = Setup(pond, free)

propagation!(eb, setup)
;
```

```
pygui(false)
out_int = getintensity(eb, x, y)
out_smeared = smoothe(out_int, x, y, 200e-9)
fig, (ax1, ax2) = subplots(1, 2)
ax1.imshow(out_int, cmap=violett, extent=extent_e)
ax2.imshow(out_smeared, cmap=violett, extent=extent_e)
;
```




```
out = out_int ./ maximum(out_int)
out[1, 1] = 0

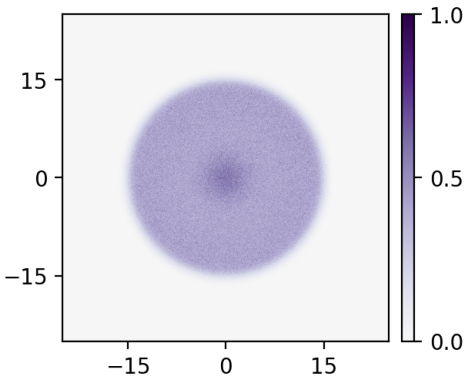
figsize = [7, 7] ./ 2.54

pygui(false)
fig, ax = subplots(figsize=figsize, dpi=dpi)
plt = ax.imshow(out, cmap=violett, extent=extent_e)

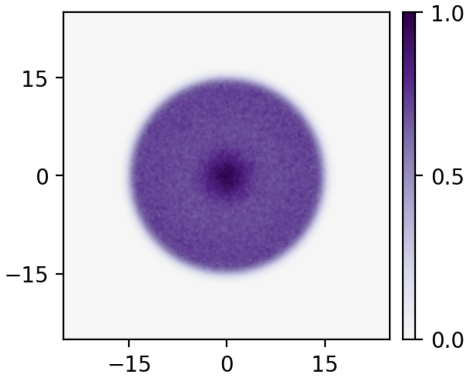
ax.set_xticks([-15, 0, 15])
ax.set_yticks([-15, 0, 15])

cax = fig.add_axes([ax.get_position().x1 + 0.03,
 ↪  ax.get_position().y0,0.03, ax.get_position().height])
fig.colorbar(plt, ax=ax, shrink=0.8, ticks=[0, 0.5, 1], cax=cax)
```

```
15
16 savefig("./images/electron_distribution.svg",
   ↪  bbox_inches="tight", dpi=dpi)
17 ;
```




```
1 out = out_smeared ./ maximum(out_smeared)
2 out[1, 1] = 0
3
4 figsize = [7, 7] ./ 2.54
5
6 pygui(false)
7 fig, ax = subplots(figsize=figsize, dpi=dpi)
8 plt = ax.imshow(out, cmap=violett, extent=extent_e)
9
10 ax.set_xticks([-15, 0, 15])
11 ax.set_yticks([-15, 0, 15])
12
13 cax = fig.add_axes([ax.get_position().x1 + 0.03,
   ↪  ax.get_position().y0,0.03, ax.get_position().height])
14 fig.colorbar(plt, ax=ax, shrink=0.8, ticks=[0, 0.5, 1], cax=cax)
15
16 savefig("./images/electron_distribution_smeared.svg",
   ↪  bbox_inches="tight", dpi=dpi)
17 ;
```

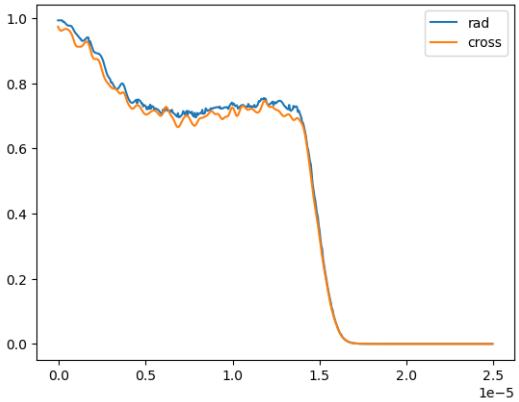



```julia
function radialint(A::Matrix{<:Real}, x::Vector{<:Real},
    y::Vector{<:Real})

    # define the distance matrix
    R = @. sqrt(x^2 + y'^2)

    # initialize the r axis and the radial distribution
    r = zeros(eltype(R), 0)
    a = zeros(eltype(A), 0)

    # create the peprmutation scheme for sorting the R and A
    perm = sortperm(vec(R))

    # make the matrix to vector, add permutation
    A_perm = vec(A)[perm]
    R_perm = vec(R)[perm]
    A_sum = 0
    count = 1
    rad_last = Inf

    @inbounds for (i, rad) in pairs(R_perm)

        # if we are in the first itaration, perpare the loop
        if i == 1
            A_sum = A_perm[1]
            rad_last = rad

        # if the current radius is the same as the one in the
        ↪  last iteration, raise the counter
        # and add the A value
```

```
        elseif rad_last == rad
            count += 1
            A_sum += A_perm[i]

        # if we have a new radius, save the old ones in
        ↪ appending the matrix
        # and reset the counter and the sum
        else
            append!(r, rad_last)
            append!(a, A_sum / count)
            count = 1
            A_sum = A_perm[i]
        end

        rad_last = rad
    end

    # normalize the radial distribution
    a ./= maximum(a)

    return r, a
end;
```

```
using Interpolations
using Array2file

r, rad = radialint(out, x, y)
xcross = x[round(Int, size(out_int, 2)/2):end]

maxind = findall(y->y==maximum(x), r)
r = r[1:maxind[1]]
rad = rad[1:maxind[1]]

interp = LinearInterpolation(r, rad, extrapolation_bc=Line())

radx = [interp(xcross[i]) for i in eachindex(xcross)]

writefile(radx, "radx.txt")
writefile(xcross, "xcross.txt")

cross = out[round(Int, size(out_int, 1)/2), round(Int,
↪ size(out_int, 2)/2):end]
```

```
plot(xcross, radx, label="rad")
plot(xcross, cross, label="cross")
legend()
;
```



Two Gaussian

The first simulation is the wave optics simulation from figure 4.5 and 4.4. The following notebook was used to conduct these simulations:

```
using ElectronPropagation
using PyPlot
using FFTW

smoothe(A, x, y, σ) = ifftshift(ifft(fft(A) .* fft(@.
↪  exp(-(x^2+y'^2)/σ^2))))
;
```

```
rcParams = PyPlot.PyDict(PyPlot.matplotlib."rcParams")
rcParams["font.size"] = 10
minval = 0.5
maxval = 1.0
cmap = get_cmap("PuOr")
violett =
↪  PyPlot.matplotlib.colors.LinearSegmentedColormap.from_list(
            "trunc("*cmap.name*", "*string(minval)*",
            ↪  "*string(maxval)*")",
            cmap(range(minval, maxval, length=cmap.N)))
cmap = get_cmap("seismic")
```

```
red =
    PyPlot.matplotlib.colors.LinearSegmentedColormap.from_list(
                "trunc("*cmap.name*", "*string(minval)*",
                 "*string(maxval)*")",
                cmap(range(minval, maxval, length=cmap.N)))
norm = PyPlot.matplotlib.colors.Normalize(vmin=0, vmax=1)
;
```

```
U = 30e3

L = 50e-6
z = 0.5
w = 15e-6  # electron beam radius

λ = deBroglieWavelength(U)

# sampling
Δx = λ * z / L / 10

display(Δx)

x = Vector{Float64}(range(-L/2, L/2-Δx, step=Δx))
y = copy(x)
;
```

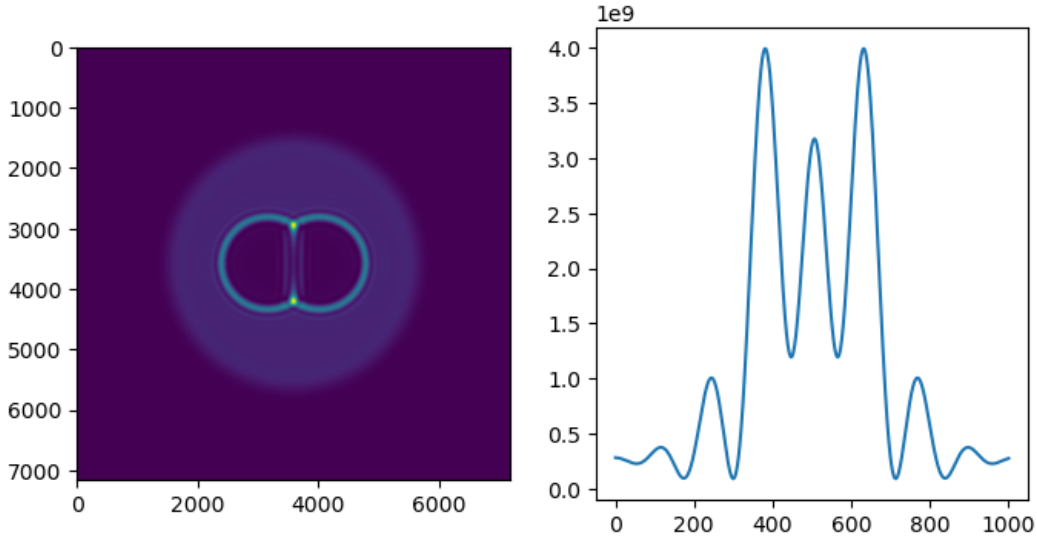
```
6.979081574270336e-9
```

```
FWHM = 4.3e-6
wgauss = FWHM / 2.355
#wgauss = 10e-6
gauss_sep = 6e-6

λ_l = 1035e-9
E = 70e-9

ximg = Vector{Float64}(range(-8e-6, 8e-6, step=0.02e-6))
yimg = Vector{Float64}(range(-16e-6, 16e-6, step=0.02e-6))

I = @. (exp(-ximg^2/2/wgauss^2 -
    (yimg'-gauss_sep/2)^2/2/wgauss^2 )
```

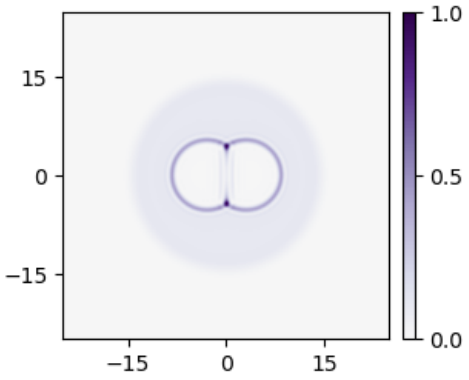
```
        + exp(-ximg^2/2/wgauss^2 -
        ↪  (yimg'+gauss_sep/2)^2/2/wgauss^2 ) )

#I = @. exp(-ximg^2/2/wgauss^2 - yimg'^2/2/wgauss^2 )

lb = LaserBeam(I, λ_l, E, ximg, yimg)

ψ = smoothe([i^2 + j^2 < w^2 ? complex(1.) : complex(0.) for i
↪  in x, j in y], x, y, 1e-6)
extent_e = [x[1], x[end], y[1], y[end]].*1e6

eb = ElectronBeam(ψ, U, x, y)
phase_imprint = PhaseImprint(lb)
free = Free(eb, z)
setup = Setup(phase_imprint, free)
propagation!(eb, setup)

out_int = abs2.(eb.ψ)
;
```

```
n, m = size(I)
I_cross = I[round(Int, n/2)+1, :]
ffset = 0.1
func(x) = x > 0.5
bounds = findall(func, I_cross)

#display(yimg[bounds[end]]-yimg[bounds[1]])
#plot(yimg, I_cross)
;
```

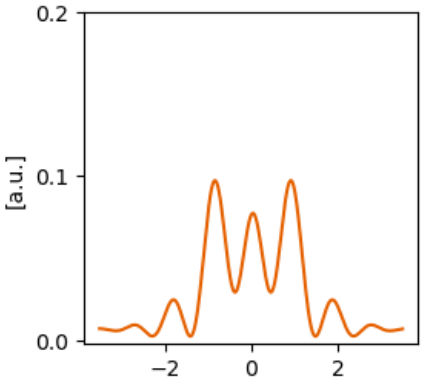
```
# plotting
cross = out_int[round(Int, size(out_int, 1)/2),
                round(Int, size(out_int, 2)/2)-500:round(Int,
                ↪  size(out_int, 2)/2)+500]

pygui(false)
figsize = [20, 10] / 2.504
fig, (ax1, ax2) = subplots(1, 2, figsize=figsize)
ax1.imshow(out_int)
ax2.plot(cross);
```



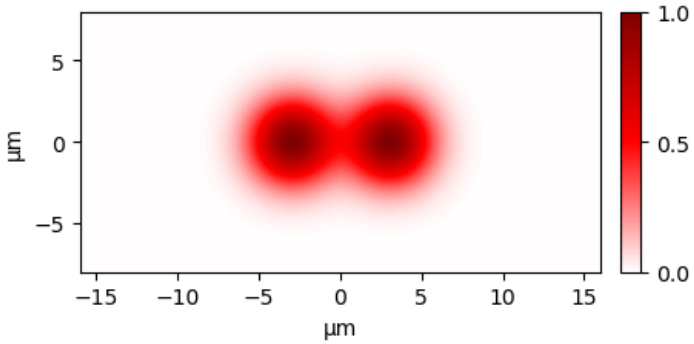

```
out = out_int ./ maximum(out_int)
out[1, 1] = 0

savestring = ("gauss"*string(round(Int, wgauss*1e6))
              *"sep"*string(round(Int, gauss_sep*1e6))
              *"E"*string(round(Int, E*1e9))*"_picture.svg")

figsize = [7, 7] ./ 2.54

pygui(false)
fig, ax = subplots(figsize=figsize)
plt = ax.imshow(out, cmap=violett, extent=extent_e)

#ax.set_xlabel("μm")
#ax.set_ylabel("μm")

ax.set_xticks([-15, 0, 15])
ax.set_yticks([-15, 0, 15])

cax = fig.add_axes([ax.get_position().x1 + 0.03,
 ↪  ax.get_position().y0, 0.03,ax.get_position().height])
fig.colorbar(plt, ax=ax, shrink=0.8, ticks=[0, 0.5, 1], cax=cax)
savefig("./images/two_gauss_sym/"*savestring,
 ↪  bbox_inches="tight", dpi=300)
;
```



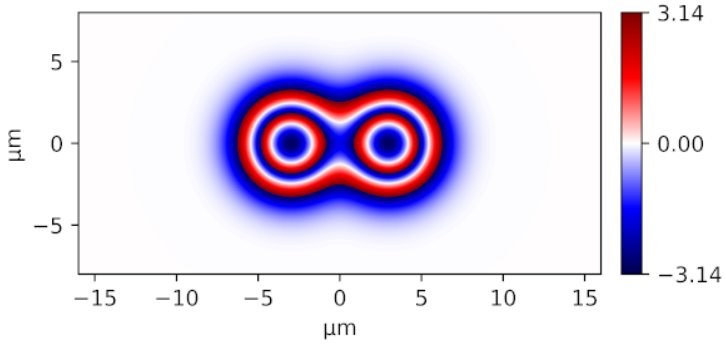

```
fov = 1000
cross = out[round(Int, size(out_int, 1)/2),
            round(Int, (size(out_int, 2)-fov)/2):round(Int,
             ↪  (size(out_int, 2)+fov)/2)]
xcross = x[round(Int, (size(out_int, 2)-fov)/2):round(Int,
 ↪  (size(out_int, 2)+fov)/2)] .* 1e6

savestring = ("gauss"*string(round(Int, wgauss*1e6))
              *"sep"*string(round(Int, gauss_sep*1e6))
              *"E"*string(round(Int, E*1e9))*"_cross.svg")

figsize = [7, 7] ./ 2.504

pygui(false)
fig, ax = subplots(figsize=figsize)
ax.plot(xcross, cross, color="#E66100")

ax.set_yticks([0, 0.1, 0.2])
ax.set_xticks([-2, 0, 2])

#ax.set_xlabel("μm")
ax.set_ylabel("[a.u.]")

savefig("./images/two_gauss_sym/"*savestring,
 ↪  bbox_inches="tight")
;
```




```
int_plot = I
int_plot[1, 1] = 0
int_plot ./= maximum(int_plot)

figsize = [11, 5.5] ./ 2.504
extent = [yimg[1], yimg[end], ximg[1], ximg[end]].*1e6

pygui(false)
fig, ax = subplots(figsize = figsize)
plt = ax.imshow(I, cmap=red, extent=extent)

ax.set_xlabel("μm")
ax.set_ylabel("μm")

cax = fig.add_axes([ax.get_position().x1 + 0.03,
↪  ax.get_position().y0, 0.03, ax.get_position().height])
fig.colorbar(plt, ax=ax, shrink=0.8, ticks=[0, 0.5, 1], cax=cax)

savefig("./images/two_gauss_sym/double_gaussian.svg",
↪  bbox_inches="tight", dpi=300)
;
```




```
int_plot = I
int_plot[1, 1] = 0
int_plot ./= maximum(int_plot)

eb_plane = ElectronBeam(ones(ComplexF64, size(I)), U, ximg, yimg)
setup = Setup(PhaseImprint(lb))
propagation!(eb_plane, setup)

figsize = [11, 5.5] ./ 2.504
extent = [yimg[1], yimg[end], ximg[1], ximg[end]].*1e6

pygui(false)
fig, ax = subplots(figsize = figsize, dpi=2000)
plt = ax.imshow(angle.(eb_plane.ψ), cmap="seismic", extent=extent)

ax.set_xlabel("μm")
ax.set_ylabel("μm")

cax = fig.add_axes([ax.get_position().x1 + 0.03, ax.get_position().y0,0.03, ax.get_position().height])
fig.colorbar(plt, ax=ax, shrink=0.8, ticks=[-3.14, 0, 3.14], cax=cax)

savefig("./images/two_gauss_sym/double_gaussian_phase.svg", bbox_inches="tight", dpi=2000)
;
```



The comparison with a ray tracing simulation is very similar the previous notebook, but using the ray tracing code instead of the wave optics one. This code was used to generate the figure 4.6:

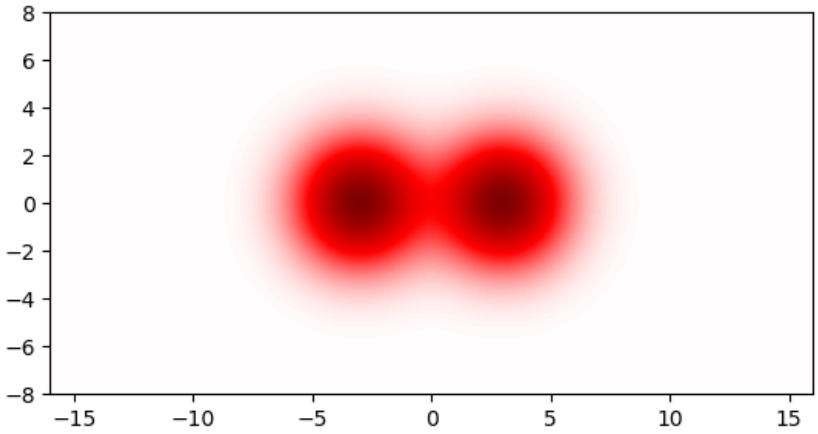
```
using Raytracing
using PyPlot
using FFTW

smoothe(A, x, y, σ) = real.(ifftshift(ifft(fft(A) .* fft(@.
↪  exp(-(x^2+y'^2)/σ^2)))))
;
```

```
rcParams = PyPlot.PyDict(PyPlot.matplotlib."rcParams")
rcParams["font.size"] = 10
minval = 0.5
maxval = 1.0
cmap = get_cmap("PuOr")
violett =
↪  PyPlot.matplotlib.colors.LinearSegmentedColormap.from_list(
            "trunc("*cmap.name*", "*string(minval)*",
            ↪  "*string(maxval)*")",
            cmap(range(minval, maxval, length=cmap.N)))
cmap = get_cmap("seismic")
red =
↪  PyPlot.matplotlib.colors.LinearSegmentedColormap.from_list(
            "trunc("*cmap.name*", "*string(minval)*",
            ↪  "*string(maxval)*")",
            cmap(range(minval, maxval, length=cmap.N)))
;
```

```
U = 30e3

L = 50e-6
z = 0.5
w = 15e-6

# same as for the wave simulation for easier image comparison
Δx = 6.97e-9

x = Vector{Float64}(range(-L/2, L/2-Δx, step=Δx))
y = copy(x)
display(size(x))

extent_e = [x[1], x[end], y[1], y[end]] .* 1e6
;
```

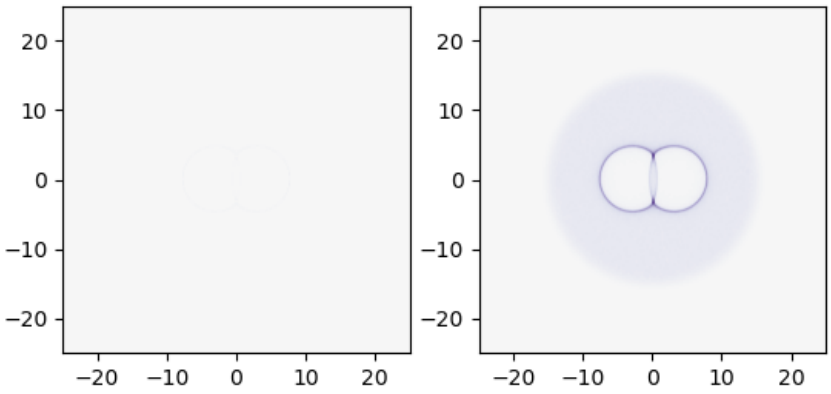
```
(7173,)
```

```
λ_l = 1035e-9
E = 50e-9
Δimg = 0.01e-6

FWHM = 4.3e-6
wgauss = FWHM / 2.355
gauss_sep = 6e-6

ximg = Vector{Float64}(range(-8e-6, 8e-6, step=Δimg))
yimg = Vector{Float64}(range(-16e-6, 16e-6, step=Δimg))

I = @. (exp(-ximg^2/2/wgauss^2 -
↪  (yimg'-gauss_sep/2)^2/2/wgauss^2 )
        + exp(-ximg^2/2/wgauss^2 -
        ↪  (yimg'+gauss_sep/2)^2/2/wgauss^2 ) )

norm = sum(I) * Δimg^2

extent = [yimg[1], yimg[end], ximg[1], ximg[end]].*1e6
pygui(false)
imshow(I, extent=extent, cmap=red)
;
```

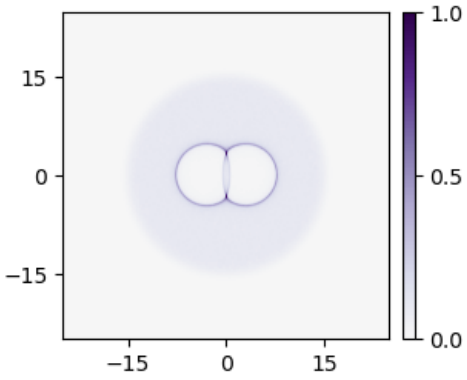



```
ψ = smoothe([i^2 + j^2 < w^2 ? 1. : 0. for i in x, j in y], x, y,
↪  1e-6)

eb = Electron(x, y, ψ, U, Int(10e6))
lf = LightField(I, norm, ximg, yimg, λ_1, E)

pond = PondInteraction(lf)
free = Free(z)

setup = Setup(pond, free)

propagation!(eb, setup)
;
```

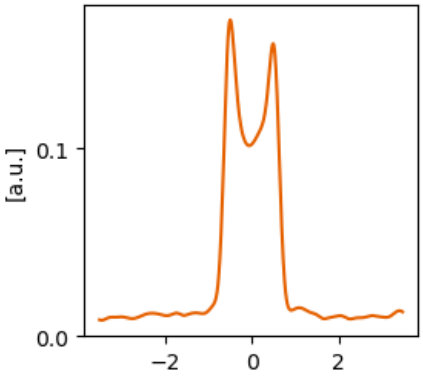
```
pygui(false)

out_int = getintensity(eb, x, y)
out_smeared = smoothe(out_int, x, y, 150e-9)

fig, (ax1, ax2) = subplots(1, 2)
ax1.imshow(out_int, cmap=violett, extent=extent_e)
ax2.imshow(out_smeared, cmap=violett, extent=extent_e)
;
```




```
out_plot = out_smeared ./ maximum(out_smeared)
out_plot[1, 1] = 0

figsize = [7, 7] ./ 2.54

savestring = ("ray_gauss"*string(round(Int, wgauss*1e6))
              *"sep"*string(round(Int, gauss_sep*1e6))
              *"E"*string(round(Int, E*1e9))*"_picture.svg")

pygui(false)
fig, ax = subplots(figsize=figsize)
plt = ax.imshow(out_plot, cmap=violett, extent=extent_e)

#ax.set_xlabel("μm")
#ax.set_ylabel("μm")

ax.set_xticks([-15, 0, 15])
ax.set_yticks([-15, 0, 15])

cax = fig.add_axes([ax.get_position().x1 + 0.03,
 ↪  ax.get_position().y0, 0.03, ax.get_position().height])
fig.colorbar(plt, ax=ax, shrink=0.8, ticks=[0, 0.5, 1], cax=cax)
savefig("./images/sym/"*savestring, bbox_inches="tight",
 ↪  dpi=300)
;
```



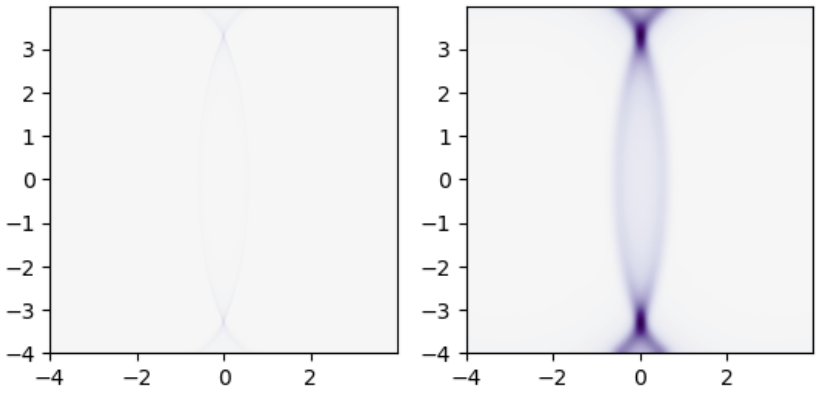

```
fov = 1000
cross = out_plot[round(Int, size(out_int, 1)/2),
            round(Int, (size(out_int, 2)-fov)/2):round(Int,
             ↪  (size(out_int, 2)+fov)/2)]
xcross = x[round(Int, (size(out_int, 2)-fov)/2):round(Int,
↪  (size(out_int, 2)+fov)/2)] .* 1e6

figsize = [7, 7] ./ 2.504

savestring = ("ray_gauss"*string(round(Int, wgauss*1e6))
              *"sep"*string(round(Int, gauss_sep*1e6))
              *"E"*string(round(Int, E*1e9))*"_cross.svg")

pygui(false)
fig, ax = subplots(figsize=figsize)
ax.plot(xcross, cross, color="#E66100")

ax.set_yticks([0, 0.1])
ax.set_xticks([-2, 0, 2])

#ax.set_xlabel("μm")
ax.set_ylabel("[a.u.]")

cross_max_big = maximum(cross)

#savefig("./images/"*savestring, bbox_inches="tight")
;
```




```
U = 30e3

L2 = 8e-6
z = 0.5
w = 15e-6

# same as for the wave simulation for easier image comparison
Δx = 6.97e-9

x2 = Vector{Float64}(range(-L2/2, L2/2-Δx, step=Δx))
y2 = copy(x2)

extent_e = [x2[1], x2[end], y2[1], y2[end]] .* 1e6

ψ = smoothe([i^2 + j^2 < w^2 ? 1. : 0. for i in x2, j in y2], x2,
↪  y2, 1e-6)

eb2 = Electron(x2, y2, ψ, U, Int(20e6))
lf = LightField(I, norm, ximg, yimg, λ_l, E)

pond = PondInteraction(lf)
free = Free(z)

setup = Setup(pond, free)

propagation!(eb2, setup)
```

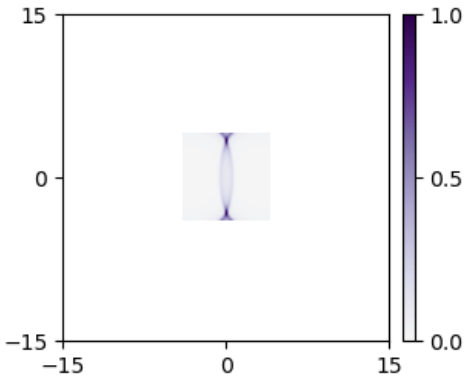
```
pygui(false)

```

```
out_int = getintensity(eb2, x2, y2)
out_smeared = smoothe(out_int, x2, y2, 150e-9)

fig, (ax1, ax2) = subplots(1, 2)
ax1.imshow(out_int, cmap=violett, extent=extent_e)
ax2.imshow(out_smeared, cmap=violett, extent=extent_e)
;
```



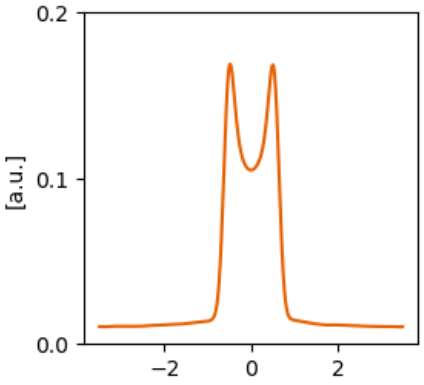

```
out_plot = out_smeared ./ maximum(out_smeared)
out_plot[1, 1] = 0

figsize = [7, 7] ./ 2.54

savestring = ("ray_gauss"*string(round(Int, wgauss*1e6))
              *"sep"*string(round(Int, gauss_sep*1e6))
              *"E"*string(round(Int, E*1e9))*"_picture.svg")

pygui(false)
fig, ax = subplots(figsize=figsize)
plt = ax.imshow(out_plot, cmap=violett, extent=extent_e)

#ax.set_xlabel("μm")
#ax.set_ylabel("μm")

ax.set_xticks([-15, 0, 15])
ax.set_yticks([-15, 0, 15])

cax = fig.add_axes([ax.get_position().x1 + 0.03,
↪  ax.get_position().y0,0.03, ax.get_position().height])
```

```
fig.colorbar(plt, ax=ax, shrink=0.8, ticks=[0, 0.5, 1], cax=cax)
;
```




```
fov = 1000
cross = out_plot[round(Int, size(out_int, 1)/2),
            round(Int, (size(out_int, 2)-fov)/2):round(Int,
             ↪  (size(out_int, 2)+fov)/2)]
xcross = x2[round(Int, (size(out_int, 2)-fov)/2):round(Int,
 ↪  (size(out_int, 2)+fov)/2)] .* 1e6

cross ./= maximum(cross)
cross .*= cross_max_big

figsize = [7, 7] ./ 2.504

savestring = ("ray_gauss"*string(round(Int, wgauss*1e6))
              *"sep"*string(round(Int, gauss_sep*1e6))
              *"E"*string(round(Int, E*1e9))*"_cross.svg")

pygui(false)
fig, ax = subplots(figsize=figsize)
ax.plot(xcross, cross, color="#E66100")

ax.set_yticks([0, 0.1, 0.2])
ax.set_xticks([-2, 0, 2])

#ax.set_xlabel("μm")
ax.set_ylabel("[a.u.]")
savefig("./images/sym/"*savestring, bbox_inches="tight")
```

```
;
```



Two Gaussian, different phase

The simulations in section 4.3.3 where the two Gaussian's have different intensities, is conducted in the following notebook:

```
using ElectronPropagation
using PyPlot
using FFTW

smoothe(A, x, y, σ) = ifftshift(ifft(fft(A) .* fft(@.
↪  exp(-(x^2+y'^2)/σ^2))))
;
```

```
rcParams = PyPlot.PyDict(PyPlot.matplotlib."rcParams")
rcParams["font.size"] = 10
minval = 0.5
maxval = 1.0
cmap = get_cmap("PuOr")
violett =
↪  PyPlot.matplotlib.colors.LinearSegmentedColormap.from_list(
            "trunc("*cmap.name*", "*string(minval)*",
            ↪  "*string(maxval)*")",
            cmap(range(minval, maxval, length=cmap.N)))
cmap = get_cmap("seismic")
red =
↪  PyPlot.matplotlib.colors.LinearSegmentedColormap.from_list(
```

```
            "trunc("*cmap.name*", "*string(minval)*",
            ↪  "*string(maxval)*")",
            cmap(range(minval, maxval, length=cmap.N)))
norm = PyPlot.matplotlib.colors.Normalize(vmin=0, vmax=1)
;
```

```
U = 30e3

L = 50e-6
z = 0.5
w = 15e-6  # electron beam radius

λ = deBroglieWavelength(U)

# sampling
Δx = λ * z / L / 10

x = Vector{Float64}(range(-L/2, L/2-Δx, step=Δx))
y = copy(x)
;
```

Derivation of the phase shift between the two peeks of $\Delta\varphi$ Phase shift from the laser pulse

$$\varphi = -\underbrace{\frac{\alpha}{2\pi(1+\beta)}\frac{\lambda_L^2}{E_e}}_{K}\frac{E}{\int I\,dxdy} = -K\frac{E}{\int I\,dxdy}I$$

Ansatz for the intensity distribution:

$$I = A\underbrace{f(x-x_1, y-y_1)}_{f_A} + B\underbrace{f(x-x_2, y-y_1)}_{f_B}$$

with

$$\max f = 1$$

and the norm

$$N = \int_{-\infty}^{\infty} dxdy\, f(x,y).$$

Rewriting the phase shift gives

$$\varphi = -K\frac{E}{(A+B)N}(Af_A + Bf_B).$$

The phase shift contribution of each peak gives

$$\varphi_A = -K\frac{E}{(A+B)N}Af_A\varphi_B = -K\frac{E}{(A+B)N}Bf_B,$$

which, under the assumption that $A = 1$, that the maximum phase shift is wanted ($f_A = f_B = 1$), that the phase difference between the maximum phase shifts is $\Delta\varphi$ and $\varphi_B = \varphi_A + \Delta\varphi$, can be rewritten to

$$\varphi_A = -K\frac{E}{(1+B)N}A\varphi_A = -K\frac{E}{(1+B)N}B - \Delta\varphi.$$

Eliminating φ_A and reshuffling everything gives

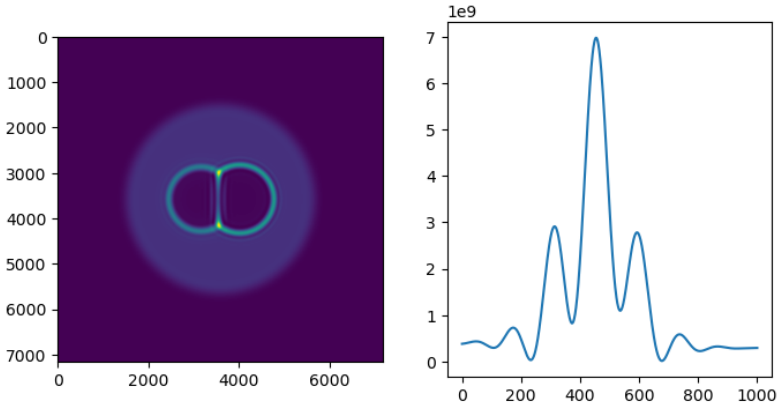
$$B = \frac{1 - \frac{N\Delta\varphi}{KE}}{1 + \frac{N\Delta\varphi}{KE}}$$

```
c = 299792458            # the speed of light
ħ = 1.054571817e-34      # the reduced planck constant
m_e = 9.1093837015e-31   # the electron mass
q = 1.602176634e-19      # electron charge
ε_0 = 8.8541878128e-12   # vacuum permitivity

v = c * sqrt(1 - 1 / (1 + q*U/m_e/c^2)^2)
α = 1 / (4 * π * ε_0) * q^2 / (ħ * c)
β = v / c
γ = 1 / sqrt(1 - β^2)
Ee = γ*m_e*c^2
;
```

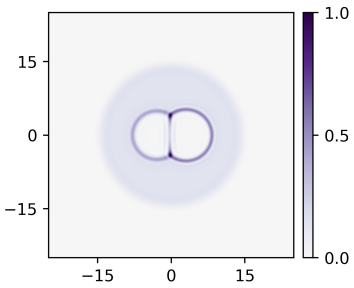
```
FWHM = 4.3e-6
wgauss = FWHM / 2.355
#wgauss = 10e-6
gauss_sep = 6e-6

# phase difference
Δφ = 2*π/1

Δimg = 0.02e-6

ximg = Vector{Float64}(range(-8e-6, 8e-6, step=Δimg))
yimg = Vector{Float64}(range(-16e-6, 16e-6, step=Δimg))

```

```
λ_l = 1035e-9
E = 60e-9

N = sum(@. exp(-(ximg^2 + yimg'^2)/wgauss^2)) * Δimg^2
K = α * λ_l^2 / (2 * π * (1 + β) * Ee)

B = (1 - N * Δφ / K / E) / (1 + N * Δφ / K / E)

I = @. (exp(-ximg^2/2/wgauss^2 -
 ↪  (yimg'-gauss_sep/2)^2/2/wgauss^2 )
        + B * exp(-ximg^2/2/wgauss^2 -
         ↪  (yimg'+gauss_sep/2)^2/2/wgauss^2 ) )

lb = LaserBeam(I, λ_l, E, ximg, yimg)

ψ = smoothe([i^2 + j^2 < w^2 ? complex(1.) : complex(0.) for i
 ↪  in x, j in y], x, y, 1e-6)
extent_e = [x[1], x[end], y[1], y[end]].*1e6

eb = ElectronBeam(ψ, U, x, y)
phase_imprint = PhaseImprint(lb)
free = Free(eb, z)
setup = Setup(phase_imprint, free)
propagation!(eb, setup)

out_int = abs2.(eb.ψ)
;
```

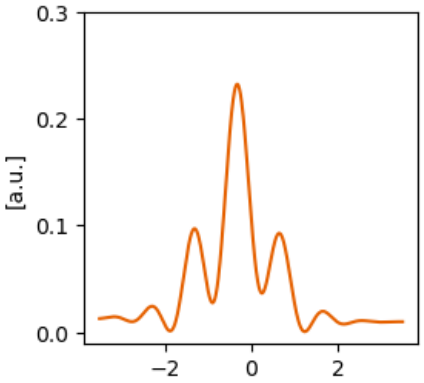
```
# plotting
cross = out_int[round(Int, size(out_int, 1)/2),
                round(Int, size(out_int, 2)/2)-500:round(Int,
                 ↪  size(out_int, 2)/2)+500]

pygui(false)
figsize = [20, 10] / 2.504
fig, (ax1, ax2) = subplots(1, 2, figsize=figsize)
ax1.imshow(out_int)
ax2.plot(cross);
```



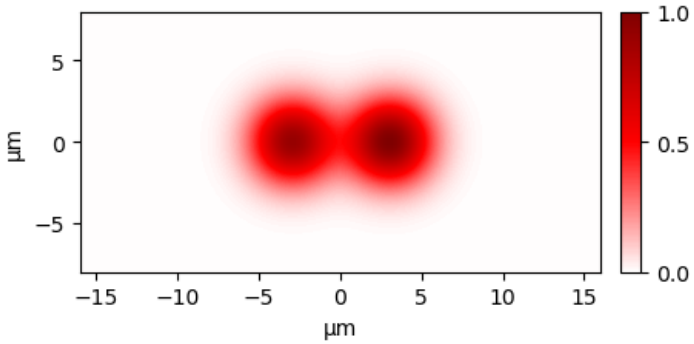

```
out = out_int ./ maximum(out_int)
out[1, 1] = 0

savestring = ("gauss"*string(round(Int, wgauss*1e6))
              *"sep"*string(round(Int, gauss_sep*1e6))
              *"dphi"*string(round(Int, Δφ))*"_picture.svg")

figsize = [7, 7] ./ 2.54

pygui(false)
fig, ax = subplots(figsize=figsize, dpi=500)
plt = ax.imshow(out, cmap=violett, extent=extent_e)

#ax.set_xlabel("μm")
#ax.set_ylabel("μm")

ax.set_xticks([-15, 0, 15])
ax.set_yticks([-15, 0, 15])

cax = fig.add_axes([ax.get_position().x1 + 0.03,
 ↪  ax.get_position().y0,0.03, ax.get_position().height])
fig.colorbar(plt, ax=ax, shrink=0.8, ticks=[0, 0.5, 1], cax=cax)
savefig("./images/two_gauss_asym/"*savestring,
 ↪  bbox_inches="tight", dpi=500)
;
```



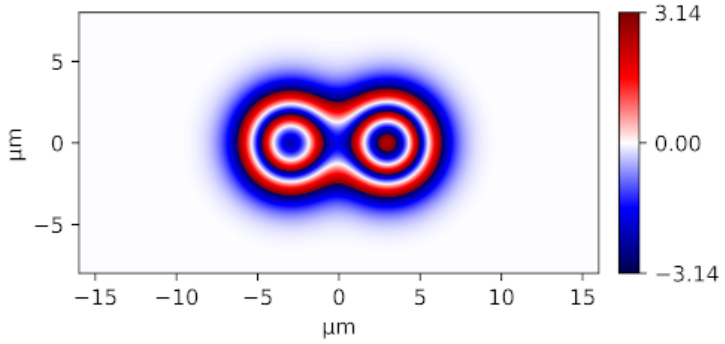

```
fov = 1000
cross = out[round(Int, size(out_int, 1)/2),
            round(Int, (size(out_int, 2)-fov)/2):round(Int,
             ↪  (size(out_int, 2)+fov)/2)]
xcross = x[round(Int, (size(out_int, 2)-fov)/2):round(Int,
 ↪  (size(out_int, 2)+fov)/2)] .* 1e6

savestring = ("gauss"*string(round(Int, wgauss*1e6))
              *"sep"*string(round(Int, gauss_sep*1e6))
              *"dphi"*string(round(Int, Δφ))*"_cross.svg")

figsize = [7, 7] ./ 2.504

pygui(false)
fig, ax = subplots(figsize=figsize)
ax.plot(xcross, cross, color="#E66100")

ax.set_yticks([0, 0.1, 0.2, 0.3])
ax.set_xticks([-2, 0, 2])

#ax.set_xlabel("μm")
ax.set_ylabel("[a.u.]")

savefig("./images/two_gauss_asym/"*savestring,
 ↪  bbox_inches="tight")
;
```




```
int_plot = I
int_plot[1, 1] = 0
int_plot ./= maximum(int_plot)

figsize = [11, 5.5] ./ 2.504
extent = [yimg[1], yimg[end], ximg[1], ximg[end]].*1e6

pygui(false)
fig, ax = subplots(figsize = figsize)
plt = ax.imshow(I, cmap=red, extent=extent)

ax.set_xlabel("μm")
ax.set_ylabel("μm")

cax = fig.add_axes([ax.get_position().x1 + 0.03,
↪  ax.get_position().y0,0.03, ax.get_position().height])
fig.colorbar(plt, ax=ax, shrink=0.8, ticks=[0, 0.5, 1], cax=cax)

savefig("./images/two_gauss_asym/double_gaussian_intensity.svg",
↪  bbox_inches="tight", dpi=300)
;
```




```
int_plot = I
int_plot[1, 1] = 0
int_plot ./= maximum(int_plot)

eb_plane = ElectronBeam(ones(ComplexF64, size(I)), U, ximg, yimg)
setup = Setup(PhaseImprint(lb))
propagation!(eb_plane, setup)

figsize = [11, 5.5] ./ 2.504
extent = [yimg[1], yimg[end], ximg[1], ximg[end]].*1e6

pygui(false)
fig, ax = subplots(figsize = figsize, dpi=2000)
plt = ax.imshow(angle.(eb_plane.ψ), cmap="seismic", extent=extent)

ax.set_xlabel("μm")
ax.set_ylabel("μm")

cax = fig.add_axes([ax.get_position().x1 + 0.03, ax.get_position().y0,0.03, ax.get_position().height])
fig.colorbar(plt, ax=ax, shrink=0.8, ticks=[-3.14, 0, 3.14], cax=cax)

savefig("./images/two_gauss_asym/double_gaussian_phase.svg", bbox_inches="tight", dpi=2000)
;
```



For the comparison between the wave optics and the ray tracing simulation of the asymmetric Gaussian beams, the following code was used:

```
using Raytracing
using PyPlot
using FFTW

smoothe(A, x, y, σ) = real.(ifftshift(ifft(fft(A) .* fft(@.
↪  exp(-(x^2+y'^2)/σ^2)))))
;
```

```
rcParams = PyPlot.PyDict(PyPlot.matplotlib."rcParams")
rcParams["font.size"] = 10
minval = 0.5
maxval = 1.0
cmap = get_cmap("PuOr")
violett =
↪  PyPlot.matplotlib.colors.LinearSegmentedColormap.from_list(
            "trunc("*cmap.name*", "*string(minval)*",
            ↪  "*string(maxval)*")",
            cmap(range(minval, maxval, length=cmap.N)))
cmap = get_cmap("seismic")
red =
↪  PyPlot.matplotlib.colors.LinearSegmentedColormap.from_list(
            "trunc("*cmap.name*", "*string(minval)*",
            ↪  "*string(maxval)*")",
            cmap(range(minval, maxval, length=cmap.N)))
;
```

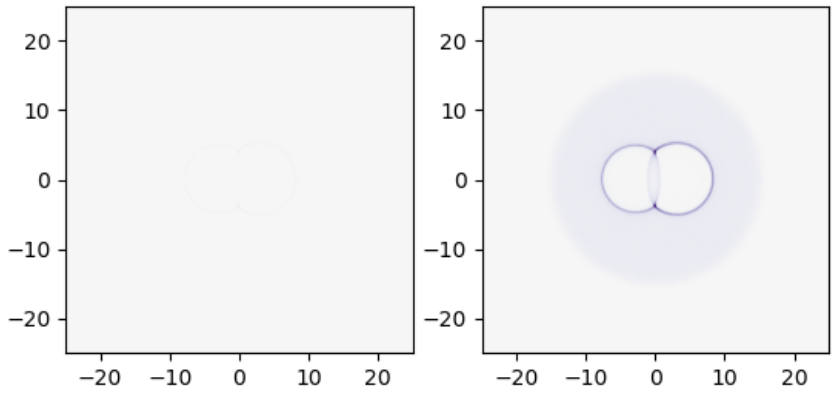
```
U = 30e3

L = 50e-6
z = 0.5
w = 15e-6

# same as for the wave simulation for easier image comparison
Δx = 6.97e-9

x = Vector{Float64}(range(-L/2, L/2-Δx, step=Δx))
y = copy(x)

extent_e = [x[1], x[end], y[1], y[end]] .* 1e6
;
```

```
c = 299792458           # the speed of light
ħ = 1.054571817e-34     # the reduced planck constant
m_e = 9.1093837015e-31  # the electron mass
q = 1.602176634e-19     # electron charge
ε_0 = 8.8541878128e-12  # vacuum permitivity

v = c * sqrt(1 - 1 / (1 + q*U/m_e/c^2)^2)
α = 1 / (4 * π * ε_0) * q^2 / (ħ * c)
β = v / c
γ = 1 / sqrt(1 - β^2)
Ee = γ*m_e*c^2
;
```

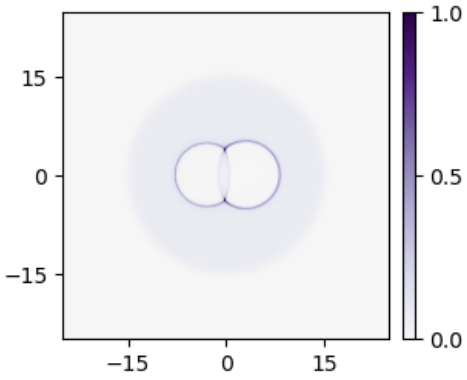
```
FWHM = 4.3e-6
wgauss = FWHM / 2.355
#wgauss = 10e-6
gauss_sep = 6e-6

# phase difference
Δφ = 2*π/1

Δimg = 0.02e-6

ximg = Vector{Float64}(range(-8e-6, 8e-6, step=Δimg))
yimg = Vector{Float64}(range(-16e-6, 16e-6, step=Δimg))

```

```
λ_l = 1035e-9
E = 60e-9

N = sum(@. exp(-(ximg^2 + yimg'^2)/wgauss^2)) * Δimg^2
K = α * λ_l^2 / (2 * π * (1 + β) * Ee)

B = (1 - N * Δφ / K / E) / (1 + N * Δφ / K / E)

I = @. (exp(-ximg^2/2/wgauss^2 -
    (yimg'-gauss_sep/2)^2/2/wgauss^2 )
        + B * exp(-ximg^2/2/wgauss^2 -
         (yimg'+gauss_sep/2)^2/2/wgauss^2 ) )

norm = sum(I) * Δimg^2

extent_e = [x[1], x[end], y[1], y[end]].*1e6
ψ = smoothe([i^2 + j^2 < w^2 ? 1. : 0. for i in x, j in y], x, y,
    1e-6)

eb = Electron(x, y, ψ, U, Int(10e6))
lf = LightField(I, norm, ximg, yimg, λ_l, E)

pond = PondInteraction(lf)
free = Free(z)

setup = Setup(pond, free)

propagation!(eb, setup)
```

```
pygui(false)

out_int = getintensity(eb, x, y)
out_smeared = smoothe(out_int, x, y, 150e-9)

fig, (ax1, ax2) = subplots(1, 2)
ax1.imshow(out_int, cmap=violett, extent=extent_e)
ax2.imshow(out_smeared, cmap=violett, extent=extent_e)
;
```

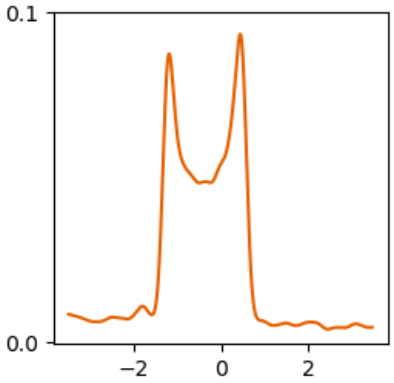



```
max_big = maximum(out_smeared)
out_plot = out_smeared ./ max_big
out_plot[1, 1] = 0

figsize = [7, 7] ./ 2.54

savestring = ("ray_gauss"*string(round(Int, wgauss*1e6))
              *"sep"*string(round(Int, gauss_sep*1e6))
              *"dphi"*string(round(Int, Δφ))*"_picture.svg")

pygui(false)
fig, ax = subplots(figsize=figsize)
plt = ax.imshow(out_plot, cmap=violett, extent=extent_e)

#ax.set_xlabel("μm")
#ax.set_ylabel("μm")

ax.set_xticks([-15, 0, 15])
ax.set_yticks([-15, 0, 15])

cax = fig.add_axes([ax.get_position().x1 + 0.03,
 ↪  ax.get_position().y0,0.03, ax.get_position().height])
fig.colorbar(plt, ax=ax, shrink=0.8, ticks=[0, 0.5, 1], cax=cax)
savefig("./images/asym/"*savestring, bbox_inches="tight",
 ↪  dpi=300)
;
```




```
fov = 1000
cross = out_plot[round(Int, size(out_int, 1)/2),
            round(Int, (size(out_int, 2)-fov)/2):round(Int,
             ↪  (size(out_int, 2)+fov)/2)]
xcross = x[round(Int, (size(out_int, 2)-fov)/2):round(Int,
 ↪  (size(out_int, 2)+fov)/2)] .* 1e6

figsize = [7, 7] ./ 2.504

savestring = ("ray_diff_gauss"*string(round(Int, wgauss*1e6))
              *"sep"*string(round(Int, gauss_sep*1e6))
              *"dphi"*string(round(Int, Δφ))*"_cross.svg")

pygui(false)
fig, ax = subplots(figsize=figsize)
ax.plot(xcross, cross, color="#E66100")

ax.set_yticks([0, 0.1])
ax.set_xticks([-2, 0, 2])
cross_max_big = maximum(cross)
display(cross_max_big)
;
```

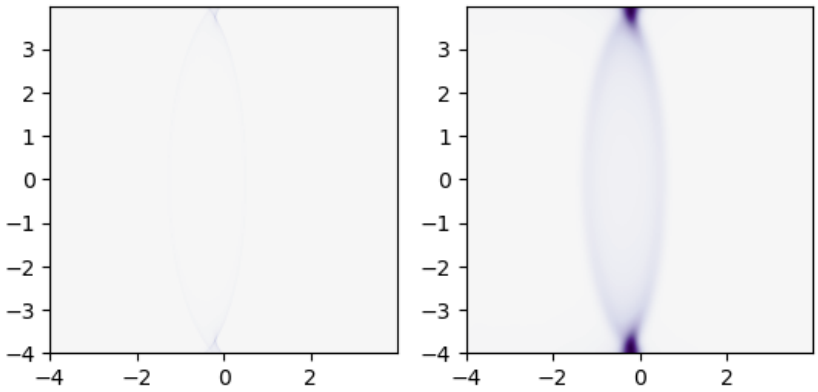
```
0.09334948642566751
```




```
U = 30e3

L2 = 8e-6
z = 0.5
w = 15e-6

# same as for the wave simulation for easier image comparison
Δx = 6.97e-9

x2 = Vector{Float64}(range(-L2/2, L2/2-Δx, step=Δx))
y2 = copy(x2)

extent_e = [x2[1], x2[end], y2[1], y2[end]] .* 1e6

ψ = smoothe([i^2 + j^2 < w^2 ? 1. : 0. for i in x2, j in y2], x2,
↪  y2, 1e-6)

eb2 = Electron(x2, y2, ψ, U, Int(20e6))
lf = LightField(I, norm, ximg, yimg, λ_l, E)

pond = PondInteraction(lf)
free = Free(z)

setup = Setup(pond, free)

propagation!(eb2, setup)
```

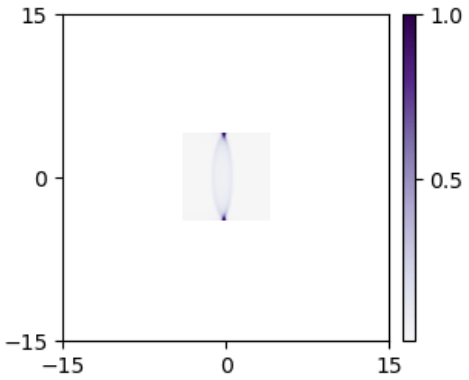
```
pygui(false)

```

```
out_int = getintensity(eb2, x2, y2)
out_smeared = smoothe(out_int, x2, y2, 150e-9)

fig, (ax1, ax2) = subplots(1, 2)
ax1.imshow(out_int, cmap=violett, extent=extent_e)
ax2.imshow(out_smeared, cmap=violett, extent=extent_e)
;
```

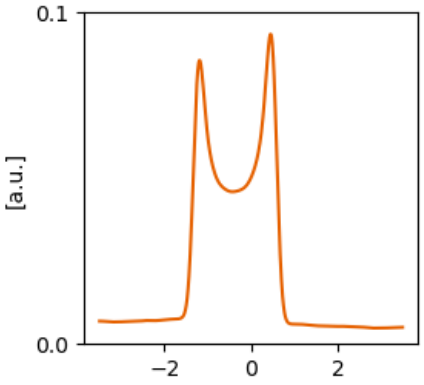



```
out_plot = out_smeared ./ maximum(out_smeared)

figsize = [7, 7] ./ 2.54

savestring = ("diff_ray_gauss"*string(round(Int, wgauss*1e6))
              *"sep"*string(round(Int, gauss_sep*1e6))
              *"E"*string(round(Int, E*1e9))*"_picture.svg")

pygui(false)
fig, ax = subplots(figsize=figsize)
plt = ax.imshow(out_plot, cmap=violett, extent=extent_e)

#ax.set_xlabel("μm")
#ax.set_ylabel("μm")

ax.set_xticks([-15, 0, 15])
ax.set_yticks([-15, 0, 15])

cax = fig.add_axes([ax.get_position().x1 + 0.03,
↪  ax.get_position().y0,0.03, ax.get_position().height])
fig.colorbar(plt, ax=ax, shrink=0.8, ticks=[0, 0.5, 1], cax=cax)
```

```
;
```




```
fov = 1000
cross = out_plot[round(Int, size(out_int, 1)/2),
            round(Int, (size(out_int, 2)-fov)/2):round(Int,
            ↪  (size(out_int, 2)+fov)/2)]
xcross = x2[round(Int, (size(out_int, 2)-fov)/2):round(Int,
↪  (size(out_int, 2)+fov)/2)] .* 1e6

cross ./= maximum(cross)
cross .*= cross_max_big

figsize = [7, 7] ./ 2.504

savestring = ("ray_gauss"*string(round(Int, wgauss*1e6))
              *"sep"*string(round(Int, gauss_sep*1e6))
              *"dphi"*string(round(Int, Δφ))*"_cross.svg")

pygui(false)
fig, ax = subplots(figsize=figsize)
ax.plot(xcross, cross, color="#E66100")

ax.set_yticks([0, 0.1])
ax.set_xticks([-2, 0, 2])

#ax.set_xlabel("μm")
ax.set_ylabel("[a.u.]")
savefig("./images/asym/"*savestring, bbox_inches="tight")
;
```


0.1
0.0
[a.u.]
−2
0
2