P441 & P442: Investigating Fourier Optics and Spatial Filtering

Jyotirmaya Shivottam*
National Institute of Science Education and Research, Bhubaneswar, India
(Dated: February 05, 2021)

In this experiment, we have investigated the far-field or Fraunhofer regime of diffraction and its natural connection with "optical Fourier Transform" or spatial Fourier modes and how one could exploit those to create simple optical image processors. In particular, we have constructed a *4f*-Imaging System and used it to test the implications of "optical Fourier transform" with several basic objects and filters. We have also extended this to some applications, such as Optical Image Filtration, Character Recognition, Phase-Contrast Imaging and Dark-Field Illumination. We have presented data for all the studied objects and as a comparative numerical model, we have written Python codes to simulate the *4f*-image processor. We have analyzed our observations and results and compared those to the numerical and theoretical predictions. Finally, we have examined the various issues that one may be face with the current experimental setup & offered potential solutions for the same; and the future directions that one could take this experiment in.

CONTENTS

I. INTRODUCTION

A. Overview

The diffraction of light can be studied in various ways, assuming the wave-like nature of light. One can apply the Huygens-Fresnel principle [2] or make use of the extensive work done by Sommerfeld and Kirchhoff [3] to recover the physical results of interference and diffraction. Sommerfeld and Kirchhoff's work, in particular, has proven extremely convenient to model the propagation of light across various configurations, either analytically or via numerical modelling. It has also paved the way for *Fourier Optics*, that is the study of wave phenomena using Fourier Transforms, where the complex waveforms are considered to be a superposition of plane waves.

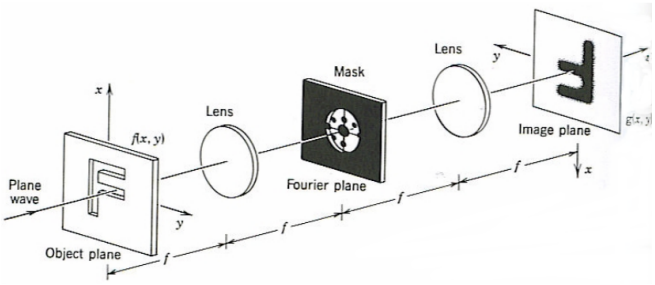


FIG. 1: *4f*-Imaging System [1]

In this experiment, we have investigated Fourier Optics in the far-field limit i.e., Fraunhofer Diffraction regime. We have first derived the relationship between the 2D Fourier transform and Fraunhofer Diffraction and then validated the expected waveforms through an experimental setup, consisting of a *4f*-Imaging System (Fig. 1). We have also presented a schematic for the setup and several observations, taken using the setup. The *4f*-Imaging System is a well-studied optical image processor (for reference, see [1] and [4]). In this experiment, we have explored some aspects of it and created a numerical model of the setup, under paraxial approximation. We have found that the physics of the Fraunhofer regime can be satisfactorily modeled by Fourier Optics. We have also extended our investigation to some of the practical use-cases of this setup

* jyotirmaya.shivottam@niser.ac.in

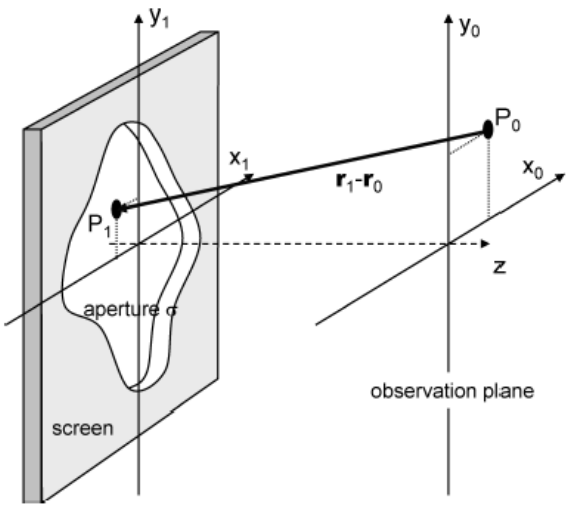


FIG. 2: Transmission through an aperture [5]

and Fourier Optics in general. Specifically, we have considered Spatial Filtering (Optical Image Filtration), Character Recognition, Phase-Contrast Imaging and Dark-Field Illumination.

In the next section, we expand on the theory of Fourier Optics and briefly explain the aforementioned use-cases. Then, in Section - II, we detail the apparatus and procedure for this experiment and in Section - III, we present our observations. Ultimately, in Section - IV, we discuss our results as well as the experimental setup. We have also presented some extensions to this experiment in the same section. The appendices, containing the code for the numerical model for each case-study, can be found near the end of this document.

B. Theory

1. *Fraunhofer Diffraction Formula*

For a monochromatic wave incident on an aperture, the Rayleigh-Sommerfeld diffraction formula can be given as follows:

$$\boxed{U(P_0) = \int\int_{\sigma} h(P_0, P_1) U(P_1) \mathrm{d}s_1} \tag{1}$$

where,

$$h(P_0, P_1) = \frac{-1}{i\lambda} \cos(\boldsymbol{n}, \boldsymbol{r_1} - \boldsymbol{r_0}) \frac{e^{-ik|\boldsymbol{r_1} - \boldsymbol{r_0}|}}{|\boldsymbol{r_1} - \boldsymbol{r_0}|}. \tag{2}$$

$h(P_0, P_1)$ can be thought of as a weight, that is applied to the field, $U(P_1)$ to get the field $U(P_0)$.

Now, consider a setup as shown in Fig. 2. Then, we can write the field at P_0 as:

$$U(P_0) = \int\int_{\sigma} h(x_0, y_0, x_1, y_1) U(x_1, y_1) \mathrm{d}x_1 \mathrm{d}y_1 \tag{3}$$

with

$$h(x_0, y_0, x_1, y_1) = \frac{-1}{i\lambda} \cos(\boldsymbol{n}, \boldsymbol{r_1} - \boldsymbol{r_0}) \frac{e^{-ik|\boldsymbol{r_1} - \boldsymbol{r_0}|}}{|\boldsymbol{r_1} - \boldsymbol{r_0}|}. \tag{4}$$

Also, $|\boldsymbol{r_1} - \boldsymbol{r_0}| = \sqrt{(x_0 - x_1)^2 + (y_0 - y_1)^2 + z^2}$. If we assume the axial distance (along z) to be much larger than transverse dimensions, then we also have $\cos(\boldsymbol{n}, \boldsymbol{r_1} - \boldsymbol{r_0}) \approxeq 1$. We can also expand the square root in a binomial expansion. Retaining terms till order 2, we get:

$$\begin{aligned} |\boldsymbol{r_1} - \boldsymbol{r_0}| &= \sqrt{(x_0 - x_1)^2 + (y_0 - y_1)^2 + z^2} \\ &\approxeq z \left[1 + \frac{1}{2} \left(\frac{x_1 - x_0}{2} \right)^2 + \frac{1}{2} \left(\frac{y_1 - y_0}{2} \right)^2 \right] \end{aligned} \tag{5}$$

Substituting back into Eq. (4), we get:

$$h(x_0, y_0, x_1, y_1) \approxeq \frac{-e^{-ikz}}{i\lambda z} e^{-\frac{ik}{2z}[(x_1 - x_0)^2 + (y_1 - y_0)^2]}. \tag{6}$$

Finally, we can replace integration over the aperture (σ) by integration over the entire plane by defining $U(x_1, y_1) = 0$ outside σ. Using this condition and re-arranging, we finally obtain:

$$\boxed{U(x_0, y_0) = \frac{-e^{-ikz}}{i\lambda z} e^{-\frac{ik}{2z}[x_0^2 + y_0^2]} \int\int_{-\infty}^{+\infty} U(x_1, y_1) e^{-\frac{ik}{2z}[x_1^2 + y_1^2]} e^{\frac{i2\pi}{\lambda z}[x_0 x_1 + y_0 y_1]} \mathrm{d}x_1 \mathrm{d}y_1}. \tag{7}$$

Eq. (7) is known as the *Fresnel Diffraction integral* and from its form, one can clearly see that the field $U(x_0, y_0)$, in the observation plane, is a 2D Fourier transform of the field in the object plane, $U(x_1, y_1) e^{\frac{i2\pi}{\lambda z}[x_0 x_1 + y_0 y_1]}$, where the spatial frequencies are defined by: $f_x = -\frac{x_0}{\lambda z}$ and $f_y = -\frac{y_0}{\lambda z}$. This result is valid close to the aperture or in the *near-field regime*. Far from the aperture, we can add another assumption, as given by the following equation:

$$z >> \frac{k(x_1^2 + y_1^2)_{max}}{2} \tag{8}$$

Using this, we can neglect the quadratic phase term, giving us the *Fraunhofer Diffraction formula*:

$$\boxed{U(x_0, y_0) = \frac{-e^{-ikz}}{i\lambda z} e^{-\frac{ik}{2z}[x_0^2+y_0^2]} \int\int_{-\infty}^{+\infty} U(x_1, y_1) e^{\frac{i2\pi}{\lambda z}[x_0 x_1 + y_0 y_1]} \mathrm{d}x_1 \mathrm{d}y_1}. \tag{9}$$

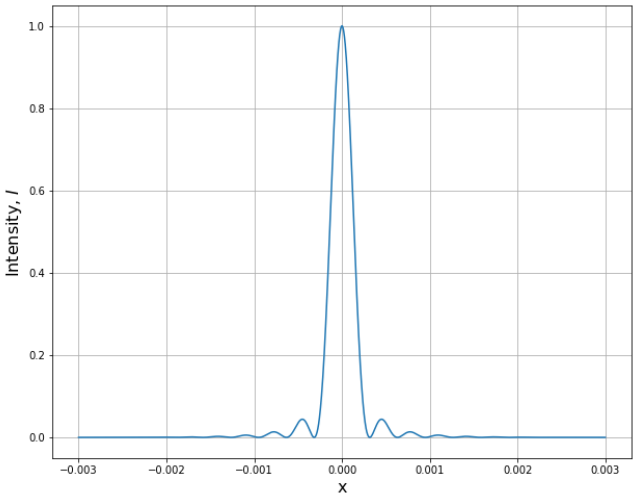


FIG. 3: Intensity profile for a rectangular aperture (along x) (e.g. single slit)

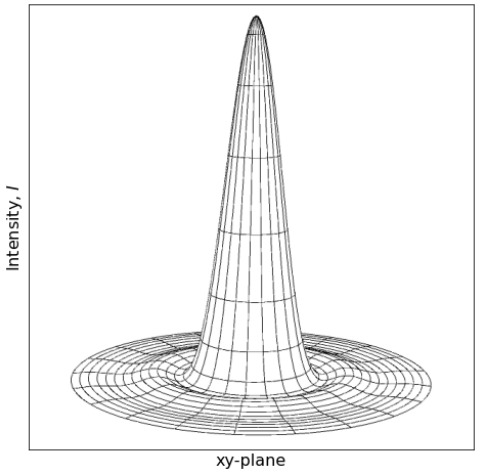


FIG. 4: Intensity profile for a circular aperture (in the $xy-$plane) (e.g. pin hole) (Source)

This is clearly a Fourier transform of the aperture field, with spatial frequencies set to f_x and f_y. To visualize this result clearly, we have discussed some theoretical cases below.

a. Rectangular aperture For a rectangular aperture, the incident field (assuming a plane, monochromatic wave) can be written as:

$$U(x_1, y_1) = \mathrm{rect}\left(\frac{x_1}{\ell_x}\right) \mathrm{rect}\left(\frac{y_1}{\ell_y}\right)$$

Plugging this into Eq. (9), we obtain the expression for Intensity profile (Modulus squared of $U(x_0, y_0)$) as:

$$I_{\mathrm{rect}}(x_0, y_0) = \left(\frac{\ell_x \ell_y}{\lambda z}\right)^2 \mathrm{sinc}\left(\frac{\pi \ell_x x_0}{\lambda z}\right) \mathrm{sinc}\left(\frac{\pi \ell_y y_0}{\lambda z}\right)$$

We have plotted this function to visualize the intensity at the observation screen, as presented in Fig. 3. Note that, since the form of I_{rect} is symmetric in x and y, we should obtain a similar variation with respect to y, with exact values dependent on the constants in the two dimensions.

b. Circular aperture For a circular aperture, with similar assumptions, the incident field can be written as:

$$U(x_1, y_1) = \mathrm{circ}\left(\frac{2r_1}{\ell}\right)$$

Plugging this into Eq. (9), we obtain the expression for Intensity profile as:

$$I_{\mathrm{circ}}(r_0) = \left(\frac{k\ell^2}{i8z}\right)^2 \left[\frac{2J_1 \frac{k\ell^2}{2z}}{\frac{k\ell^2}{2z}}\right]^2$$

where $J(r_0)$ denotes the Bessel function. This distribution is also known as the *Airy Diffraction pattern* (See Fig. 4).

2. *Fresnel Number*

Now that we have expressions for diffraction in near and far-field regimes, it would be convenient to have a method of knowing which regime is applicable to a particular problem. To this end, we can define the *Fresnel Number* or *F-Number* as follows:

$$F = \frac{a^2}{L\lambda}, \tag{10}$$

where, a is the characteristic size for the aperture (e.g. aperture radius for a lens), L is the distance to the observation screen (or focal length, f, for a lens) and λ is the wavelength of the incident light. $F >> 1$ implies near-field diffraction, while $F \approxeq 1$ implies far-field diffraction. In our case, since we are using multiple lenses to make it possible to work in the far-field regime; in effect, we are taking measures to reduce F.

3. Spatial Filtering & Convolution Theorem

Since a monochromatic wave, passing through an aperture, results in a Fourier transform in the far-field regime, with spatial modes or frequencies, one can design "filters" to block some of those modes (or in other words, block some of the information in either position or phase space). Then, if an inverse Fourier transform of the 'filtered' wave is taken, due to information loss, the resulting image will not have those spatial frequencies. This is known as *Spatial Filtering*. Two simple applications of spatial filtering are *Low* and *High*-pass image filters. Low-pass filters block higher frequencies, leading to an image, that consists of only low level features, such as simple curves. On the other hand, high-pass filters block lower frequencies, leading to an image with high level features, such as distinct & complex curves, but with no low level information. These kind of filters can be used to obtain a 'wireframe' image of an object (via high-pass filtering) or a 'blurred' image (via low-pass filtering). The former is connected to the concept of *Dark-Field Illumination* [6], wherein the zero diffraction order is filtered out by a small disk of appropriate size.

Spatial filtering also affects the phase information and thus, filters can be designed to alter this information, for example to separate out two overlapped or composite images. This forms the basis of *Phase-Contrast Imaging* [7].

Another use-case comes in the form of encoding images. Since Fourier transform of any function in position space, gives a unique encoding of the function, in phase space, we can use this property to encode images of say, letters or numbers, which can then be used alongside an Optical Character Reader to perform *Character Recognition*. The specific term associated with this kind of encoding is *Fourier Descriptors* [8].

For all of the use-cases discussed above, the mathematical foundation lies in the *Convolution Theorem*, which states that the Fourier transform of a convolution of two functions, in the position space, is the *pointwise product* of their Fourier transforms in the frequency domain or phase space:

$$\boxed{h(x) = \{f * g\}(x) = \mathcal{F}^{-1}\{F \cdot G\}} \tag{11}$$

Here, the convolution operation is understood to be between the filter function (f) and the incident wave (g). As per the Convolution theorem, the output image (in position space) is simply an inverse Fourier transform of the pointwise product of the Fourier transforms of f and g in the phase space. In other words, a spatial filter results in a computation of the 'overlap' between the incident wave and the filter, leading to a modified diffraction pattern.

4. Coherence of Incident Light

Since we require well-defined and stationary phase space information, the incident beam or light source must be *coherent*. In particular, the beam should be *spatially* and *temporally coherent*. The former gives stationary spatial modes, while the latter implies a monochromatic wave. For coherent waves, the aforementioned assumptions hold and thus the wave can be Fourier-decomposed, as per Eq. (9). If the wave is incoherent, there is no fixed phase relation between two spatial points. While we can still take Fourier transforms, the result is no longer a Fourier-decomposition into fixed-frequency plane waves, but rather a superposition of several monochromatic, coherent contributions, resulting in a uniform illumination at the observation screen, instead of a diffraction pattern.

5. Fast Fourier Transform

As part of this experiment, we have used the ImageJ software [9] to calculate Fourier and Inverse Fourier transforms of observed images. We have also written Python code to simulate a $4f$-Imaging Processor, under paraxial and far-field approximations. We have based our code on a Python library called *diffractio* [10]. Our Python code can be found in several Jupyter Notebooks in a GitHub repository. Both ImageJ and *diffractio* make use of *Fast Fourier Transform* (FFT), which is a *divide-and-conquer* algorithm, first devised by James W. Cooley and John W. Tukey [11], to make Fourier coefficient computations efficient. More information on FFT, including a bare-bones implementation can be found in [12].

II. METHODOLOGY

A. Apparatus

We have made use of the following equipment for this experiment:

1. For $4f$-Imaging System:
   (a) 2 Optical Benches (length = 1 m each) placed end-to-end
   (b) He-Ne LASER ($\leq 4\,mW$), with $\lambda = 633\,nm$
   (c) Beam Expander, with $f = 50\,mm$
   (d) Collimating Spherical Lens, with $f = 300\,mm$
   (e) 2 Fourier Transform Spherical Lenses, with $f = 300\ mm$ and aperture diameter, $d \approxeq 50\,mm$
   (f) Observation Screen
   (g) 2 XY-stages and holders, for placing transparency or object and filter
2. For creating or modifying objects and filters:

(a) Set of Metrologic transparencies
(b) Adjustable Slit
(c) Scissors
(d) Scalpel
(e) Electrical Tape
(f) Permanent Marker or Pen
(g) Multiple, transparent A4 sheets
(h) Laser Printer

3. Mobile Phone camera to record diffraction patterns and reconstructed images.

4. Some additional equipment, that may prove useful:
   (a) Measuring tape or ruler to measure distances
   (b) Small card, e.g. a business card, that can be used to follow and view the light beam at any location along the setup
   (c) Multiple screens to avoid interference with background light from other setups

B. Experimental Setup

The experimental setup consists of a *4f*-Imaging System, whose schematic has been presented in Fig. 5. The actual setup has been displayed in Fig. 7 and Fig. 8. '*4f*' in the name comes from the fact, that the objective, Fourier transform lenses and the observation screen span a total of 4 focal lengths. In our case, $f_1 = f_2$, but in general, f_1 need not be the same as f_2.

After the light from the LASER is expanded and collimated, it is shined on a transparency or object and then passed through $L1$, which produces the Fourier-decomposition or diffraction pattern at the focus of $L1$. This plane is called the *Fourier plane*. Filters are placed on the Fourier plane to selectively obfuscate some spatial modes. Subsequently, $L2$ produces another Fourier transform (or by asymmetry, negative of Inverse Fourier transform), which results in an inverted, microscopic image of the object at the focus of $L2$. We can place the observation screen, S at either this focus or we can place a beam expander (which is a lens) to produce a magnified (and erect) image of the final output, as in Fig. 6. Note that, if a beam-expanding lens is used, it will result in a Fourier decomposition in the far field limit. So, the observation screen must be placed sufficiently close to the beam-expanding lens to observe the magnified image, instead of its diffraction pattern. Since for many objects, the diffraction patterns at the Fourier plane are microscopic, we can also use a beam expander there to enlarge the image. For most of the observations, presented in the next section, beam expanders have been used, both at the Fourier plane and the Observation plane, in order to capture sharper images.

The benefit of a *4f*-imaging system is that it allows us to study Spatial Filtering in a rather compact setup. To demonstrate this, let us consider diffraction at a slit, $0.5\ mm$ wide, without any of the Fourier Transform lenses. Recalling our discussion on *Fresnel Number*, if we substitute $a = 0.5\ mm$, $\lambda = 633\ nm$ and $F = 1$ into Eq. (10), we get $L \approx 39.49\ cm$. Thus, the far-field approximations are valid only for $L > 39.49\ cm$ and so, we cannot place the observation screen any closer than L, in order to observe Fraunhofer diffraction. However, with a lens, we can obtain the Fourier Transform of the slit function, which is equivalent to the Fraunhofer diffraction pattern, at the focus, thereby greatly reducing the setup footprint.

C. Procedure

At first, we set up the experimental bench to observe the Fraunhofer diffraction pattern for several objects. Then, we set up the *4f*-imaging system and recorded the diffraction pattern & final outputs for the following objects:

1. Slit
2. Pinhole
3. Grating at an angle

After comparing and validating the observed diffraction patterns with theoretical results, we explored Spatial Filtering using a Mesh,as the object, and some filters, such as a pinhole and an adjustable slit.

We also explored these applications of Spatial Filtering:

1. Low Pass Filtering a half-tone image
2. High Pass Filtering a half-tone image
3. *Dark-Field Illumination* or Edge Enhancement in a half-tone image
4. Study of Fourier encoding of alphabets, 'E' and 'F' (*Character Recognition*)
5. Separating overlapping or composite images, using phase information (*Phase Contrast Imaging*)

The low and high pass effects were observed with several objects, but we have mainly focused on the half-tone image, as the object. Moreover, in the case of *Character Recognition*, we have experimented with inverted transparency to see, if its Fourier pattern is different in any meaningful way. As an aside, we have also explored the effects of an incoherent light source on the diffraction pattern and the final image. Data for all

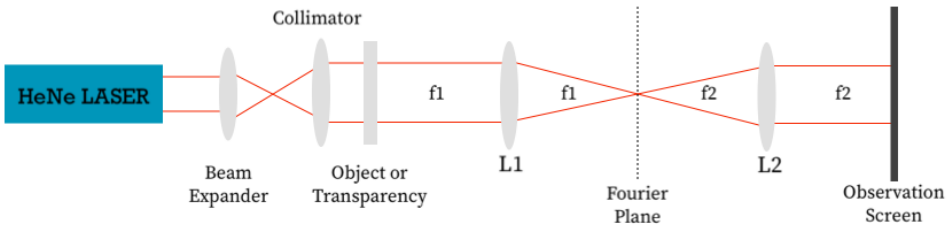


FIG. 5: Schematic of the Experimental Setup

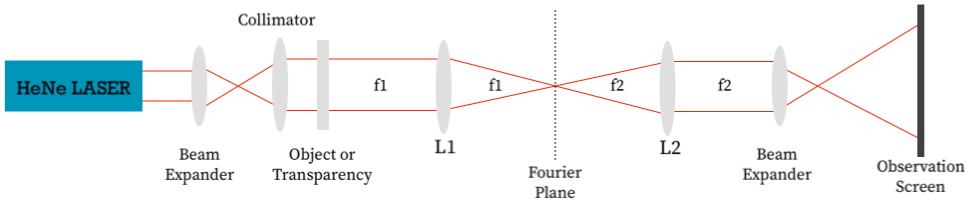


FIG. 6: Schematic, with an additional Beam Expander

these (sub-)experiments can be found in the next section.

As mentioned before, all image analysis (mainly Fourier Transforms) was done using ImageJ. We have also written Python code to simulate the $4f$-imaging system, under paraxial approximation. These can be found in the appendices at the end of the report. Furthermore, we investigated the calculation of correlation coefficients between experimental observations and images, produced using our Python code. In particular, we looked into *Average Hash* and *pHash* [13] algorithms to calculate a normalized correlation between source and target images. However, there is no standard way of relating different correlation values, across datasets. Hence, we have not included these values in the report.

III. OBSERVATIONS

As a large number of pictures were taken as part of the experiment, the experimental data has been presented in Appendix - IV C. Note that, since a handheld mobile phone camera was used to take the pictures, we have magnified, cropped and leveled some of the pictures. The ISO settings on the phone camera varied from 100 - 400, while the exposure time was always $< 1/60\ s$. Also, in some cases, such as for the half-tone image and alphabets, the image at the Fourier plane was microscopic. So, we have used a beam expander at the Fourier plane to magnify the diffraction pattern and record it. This also helped with the creation of custom filters. In the next section, we discuss and interpret these results.

IV. DISCUSSION

The parallel and instantaneous nature of optical systems can be elegantly and simplistically seen with a imaging processor, such as the $4f$-imaging system. In this mostly qualitative analysis of Fourier Optics and Spatial Filtering, for all of the cases under study, our results match with the theoretical predictions, e.g. the intensity patterns for a rectangular and a circular aperture (Compare Fig. 3 with Fig. 12 and Fig. 4 with Fig. 13). Also compare our results with that of [4]. The fact, that lenses are optical Fourier transform computers, is also verified based on the analysis of the observed intensity profiles. This lens property is also the core reason behind such a compact setup, which is crucial for the experiment. Apart from these, we

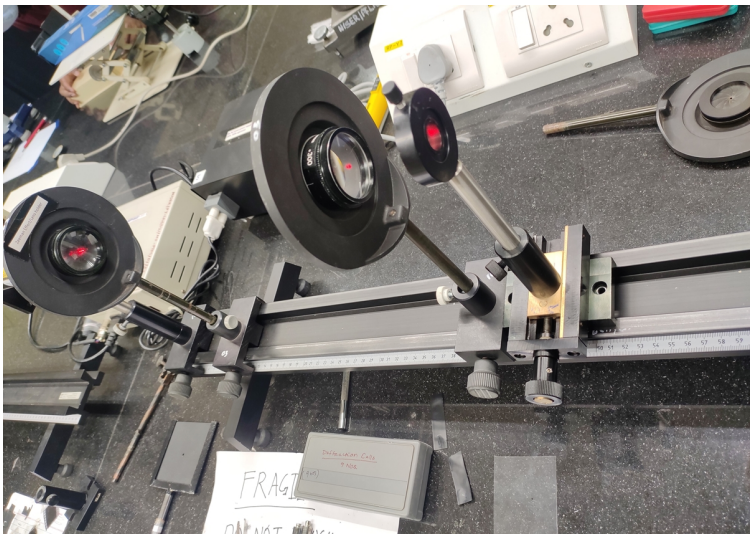


FIG. 7: Experimental Setup: HeNe LASER, Beam Expander, Collimator and Object or Transparency. At the bottom, a box containing metrologic transparencies is also visible.

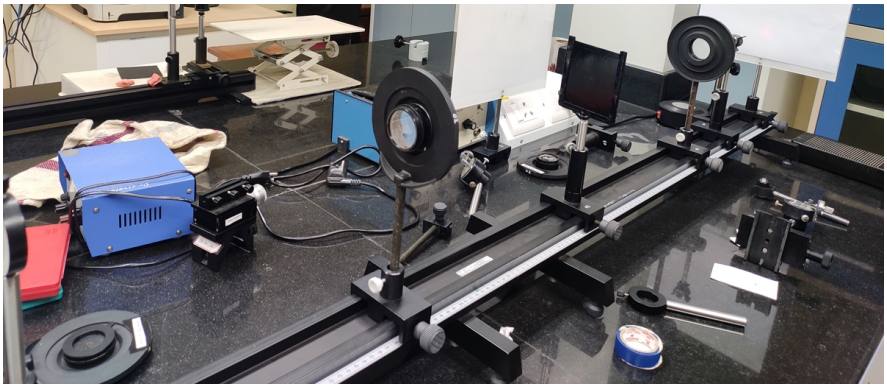


FIG. 8: Experimental Setup: The *4f*-Imaging System, consisting of two Fourier Transform lenses and an Observation Screen. A pinhole spatial filter can also be seen at the Fourier plane.

have also established the importance of spatial and temporal coherence of light, by using a diffused, white light source, that gave a uniform illumination with no discernible diffraction pattern at the Fourier Plane, as can be seen in Fig. 20. Now, let us analyze the results application-wise:

a. Spatial Filtering The concept of spatial filtering is at the heart of all the other applications here. The results from our examination of filtering with a simple mesh are sufficient to explain why. (See Fig. 15, Fig. 16 and Fig. 17)

Without any filter, we find that the *4f*-imaging system reliably reconstructs the Mesh at the observation plane. Moreover, we see that letting only the central maxima pass through results in uniform illumination, which signifies a point source, which agrees with the waveform at the Fourier Plane. Also, the Fourier transform of the final image gives back the "convolved" or modified beam (in terms of intensity), at the filter or the Fourier Plane. We also observe that blocking horizontal modes leads to a vertical pattern and vice-versa. This is in line with the homogeneity argument from single-slit Fraunhofer diffraction, wherein a vertical slit produces a horizontal pattern and vice-versa. This is also in accordance with the Fourier transform of a delta function being a spread out distribution in the transverse dimension. For a visual understanding of spatial filtering, please consult the code in Appendix - IV C.

b. Low and High-pass Filtering We have verified the low and high pass filter effects using a Half-tone image. As can be seen in the sub-figures in Fig. 19, allowing the central (low) frequency makes the setup behave as a low-pass filter, producing a blurry outline of the source image, while blocking the same makes the setup behave as a high-pass filter, that yields an image with sharper edges, than the source image. This is as expected, as fast-oscillating terms define the high-level

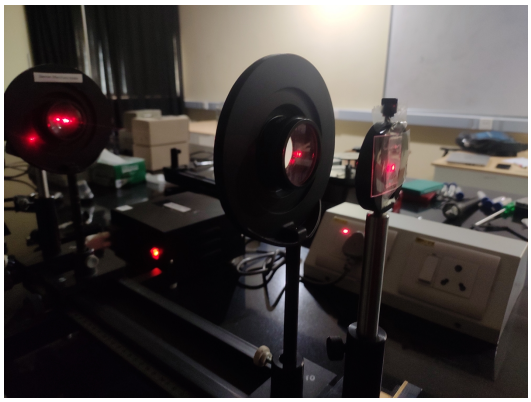


FIG. 9: In this image, back-reflection can be clearly seen on the the body of the beam expander.

features. This is also known as *Dark-field Illumination*. We have explored this in-depth in the Python code, that can be found in Appendix - IV C.

c. Character Recognition As with the other cases, we find that, here too, the results match well with the numerical outputs (Fig. 18) from ImageJ or the Python code (IV C). However, this case is unique, in that, we observe that the position space information or the diffraction pattern alone is insufficient to distinguish between the alphabets, as the patterns seem identical. However, the phase information is markedly different and can be used to encode the alphabet into a character recognition system, such as a Machine Learning-based classifier. We also note that the diffraction pattern of the inverted transparency (for letter, 'E') is not too different either. So, we can only rely on phase-space information here. In our code, we have presented more such experiments and observations.

d. Phase Contrast Imaging While we have explored and verified multiple aspects of Fourier Optics and Spatial Filtering, as well as investigated several practical use-cases, there are certain issues, that did pop up over the course of the experiment. The segment that was worst-affected was *Phase Contrast Imaging*, as data collection became impractical due to certain peculiarities of the setup. We discuss these issues and potential solutions in the next subsection. We would like to note that the Python code, written to simulate *Phase Contrast Imaging* did produce expected results. This code can be found in Appendix - IV C.

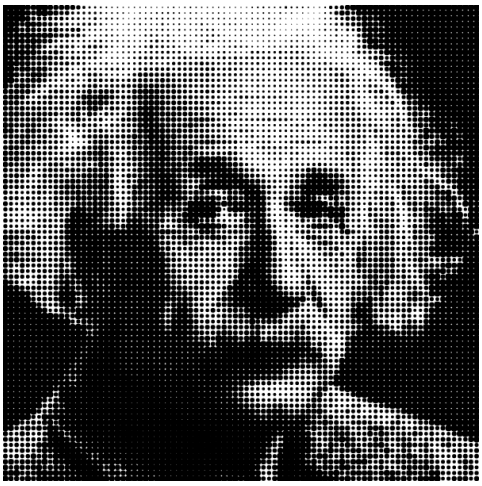
A. Setup: Issues & Improvements

a. Sustained coherence of Light A primary requirement for this experiment to be functional, is that the light source should remain coherent over the dimensions of the setup. While we confirmed that the LASER itself remained coherent and showed negligible beam deviation, the optical equipment used in the setup did affect the coherence or collimation length. Since all the lenses used in the setup were spherical, we observed varying degrees of back-reflection and optical cavity effect between different lenses. For example, see Fig. 9. It was most prominent for the large-aperture collimator and Fourier transform lenses. This poses a danger to the coherence of the beam over the setup length. While we avoided this somewhat by reorienting the lenses, a better solution would be to either use *Aspheric* lenses or add an *Aperture Stop*, which would also mitigate the LASER speckle effect. Moreover, as a test, we recommend removing all equipment except the LASER, beam expander, collimator and the observation screen (placed at the far end) and trying to decompose the observed laser beam image into Fourier modes. If Fourier modes are absent, the light beam is certainly not coherent.

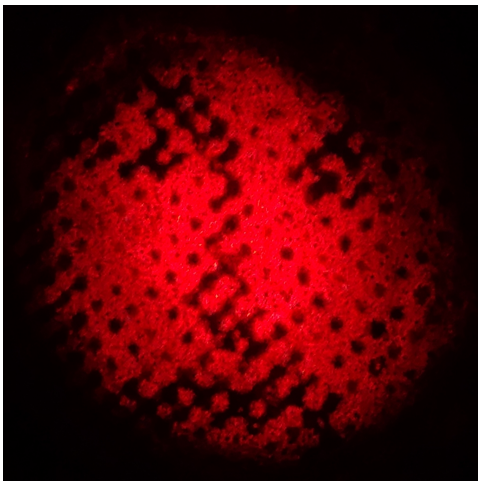
b. High Beam Intensity This issue plagued the experiment throughout, as taking data became nearly impossible with the mobile phone camera, until the ISO and exposure settings were tweaked. This caused issues, as we could not maintain a standard camera setting across images, thereby making background-subtraction a futile exercise. We did try to reduce the intensity using multiple transparencies, and even ground glass (as a sub-experiment), but we were not too successful in that endeavour. A lower intensity LASER would certainly help make the experiment and the $4f$-imaging system better.

c. Narrow Beam The beam expanders, that were available for the experiment, did not produce a light beam of sufficient width, and this became a major issue towards the end, as experiments, involving images (half-tone images or overlapped images) or anything substantially larger, could not be performed effectively. For instance, see Fig. 10, wherein only the face could be seen at the observation plane, as it was the lone portion of the image, that was being illuminated at the source. This, combined with the high beam intensity, made data collection for *Phase Contrast Imaging* impossible (Fig. 11). We tried images, other than half-tone, such as wireframe and line-composites, but neither could help us. Even with smaller objectives, the diffraction pattern and the final image were tiny and we had to resort to using an additional beam expander. A narrow beam also makes it difficult to locate the Fourier modes, that should be filtered. More often than not, we had to magnify the pictures by several orders.

A potential solution to this problem, apart from changing the beam expander or collimator, is to use a higher wavelength source. Since the beam width is inversely proportional to Rayleigh length, which in turn is inversely proportional to the wavelength, a higher wavelength beam should result in larger diffraction patterns and final images. A downside to this is shorter coherence length, but that can be mitigated by high magnification lenses, which decrease the setup footprint.



(a) Original Image



(b) At the Observation plane

FIG. 10: Due to a narrow beam, only the face can be reconstructed at the Observation plane. Combined with high intensity, it makes reliable data collection difficult.

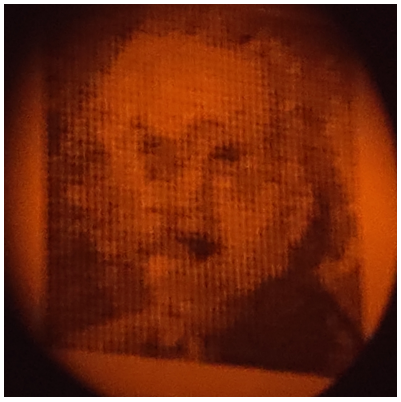


FIG. 11: Phase Contrast Imaging: Blurred reconstructed image for 2 overlapped images. Due to high beam intensity and LASER speckle, capturing sharper images was made difficult.

While these issues do necessitate some modifications to the setup, we can also enhance the experiment itself with some interesting extensions. We discuss these in the next subsection.

B. Future Extensions

This experiment can be taken forward in a plethora of ways. Some of those are:

a. Machine Learning We have seen that the Fourier decomposition for each character or alphabet is unique, either in position or phase space. This property can be utilized in the field of Machine Learning to better encode input data. This can also have speed advantages as most Machine or Deep Learning algorithms are in general, far less efficient than taking a Fast Fourier Transform. Apart from Character Recognition, these *Fourier Descriptors* can also be applied to image classification problems [14].

b. Transfer Function Formalism While we studied and understood the concepts of Fourier Optics via the work of Fresnel and Sommerfeld, the same can also be understood via signal processing techniques. In particular, Transfer function formalism can be used to study Fourier Optics in a concise manner. It can also help in understating operations, such as convolution at an aperture, better, than with the Rayleigh-Sommerfeld integral formulation. Moreover, since the complex phase space information is rolled into the definition of the transfer function, one can acquire an intuitive understanding of the problem. This, in turn, can be extended to the study of filters, such as the *VanderLugt Filter* [15], matched filters [16] and phase-only filters [17].

c. Information Theoretic-Approach Another extension to this experiment can be application of Information Theory principles to quantify the transfer and loss of information in optical image processors (See [18] and [19]). This can prove useful in maximizing information content, so as to allow for better reconstructions of distorted or modified source images.

C. Precautions

We have listed some of the precautions taken during the experiment below:

1. The coherence of the beam should be ensured over the setup dimensions.

2. It should be confirmed, that the far-field approximations are valid throughout. One may use the *Fresnel Number* as a quick check of the regime, one should work in, for a particular problem.

3. While taking pictures using a handheld camera, camera filters should be kept off and image distortions should be minimized.

4. If the pictures are taken at an angle, the Fourier transforms may pick up an additional phase contribution. It can be ignored, if no numerical work is being done (e.g., in this experiment). Otherwise, it must be accounted for.

ACKNOWLEDGMENTS

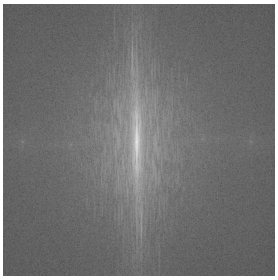
We would like to express our gratitude towards Dr. Sudakshina Prusty and Dr. Ritwik Das for insightful discussions on various aspects of this experiment.

[1] L. Y. Lin, Ch: 4, Fourier Optics and Diffraction; Introduction to Photonics, EE485, Winter 2004.
[2] C. Huygens, *Traité de la Lumière* (1690).
[3] G. Kirchhoff, Zur Theorie der Lichtstrahlen, Annalen der Physik **254**, 663 (1882).
[4] A. Eisenkraft, A closer look at diffraction: Experiments in spatial filtering, The Physics Teacher **15**, 199 (1977), https://doi.org/10.1119/1.2339599.
[5] P. R. Group, Fourier Optics: Scalar Diffraction Theory, .
[6] F. T. Chambers W. and D. MW, Dark-field Illumination, MicroscopyU, Nikon.
[7] O. R. S. S. Murphy, DB and D. MW, Phase-Contrast Imaging, MicroscopyU, Nikon.
[8] M. Shridhar and A. Badreldin, High accuracy character recognition algorithm using fourier and topological descriptors, Pattern Recognition **17**, 515 (1984).
[9] C. T. Rueden, J. Schindelin, M. C. Hiner, B. E. DeZonia, A. E. Walter, E. T. Arena, and K. W. Eliceiri, Imagej2: Imagej for the next generation of scientific image data, BMC Bioinformatics **18**, 529 (2017).
[10] L. M. S. Brea, `diffractio`: Python Diffraction-Interference module, Documentation, `diffractio`.
[11] J. W. Cooley and J. W. Tukey, An algorithm for the machine calculation of complex fourier series, Mathematics of Computation **19**, 297 (1965).
[12] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes 3rd Edition: The Art of Scientific Computing*, 3rd ed. (Cambridge University Press, USA, 2007).
[13] C. Zauner, Implementation and benchmarking of perceptual image hash functions (2010).
[14] *Auto-generated*, Fourier Descriptor, ScienceDirect.
[15] A. V. Lugt, Signal detection by complex spatial filtering, IEEE Transactions on Information Theory **10**, 139 (1964).
[16] G. Turin, An introduction to matched filters, IRE Transactions on Information Theory **6**, 311 (1960).
[17] J. L. Horner and P. D. Gianino, Phase-only matched filtering, Appl. Opt. **23**, 812 (1984).
[18] E. H. Linfoot, Information theory and optical images, J. Opt. Soc. Am. **45**, 808 (1955).
[19] J. A. O'Sullivan, R. E. Blahut, and D. L. Snyder, Information-theoretic image formation, IEEE Transactions on Information Theory **44**, 2094 (1998).

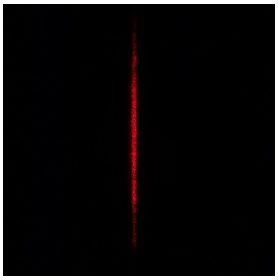
APPENDIX: OBSERVATIONS

We have taken a short video of the "assembly" of the final image from Fourier modes for the alphabet, 'E', that can be found in our GitHub repository.

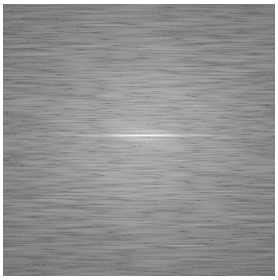
(a) At Fourier Plane



(b) FFT of Diffraction Pattern

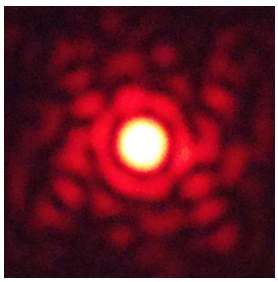


(c) At Observation Plane

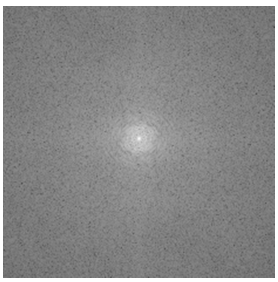


(d) FFT of Observed Image

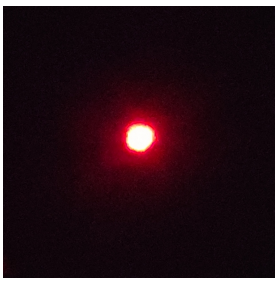
FIG. 12: Fraunhofer Diffraction Pattern for Single Slit



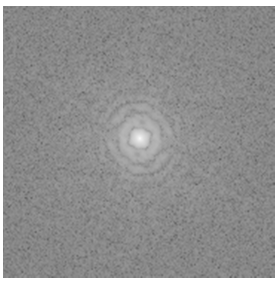
(a) At Fourier Plane



(b) FFT of Diffraction Pattern

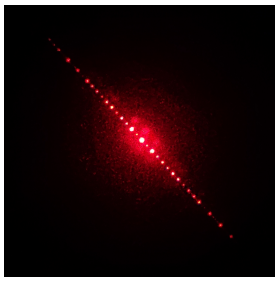


(c) At Observation Plane

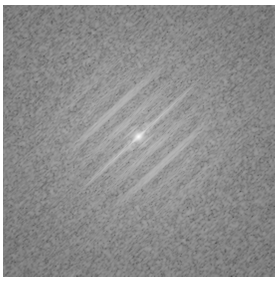


(d) FFT of Observed Image

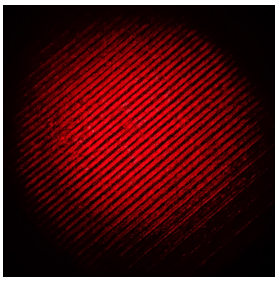
FIG. 13: Fraunhofer Diffraction Pattern for Pinhole



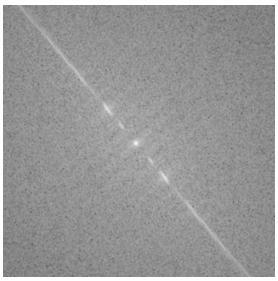
(a) At Fourier Plane



(b) FFT of Diffraction Pattern

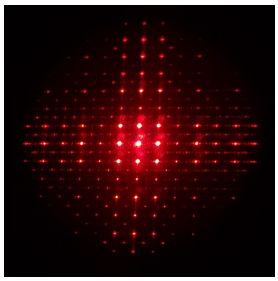


(c) At Observation Plane

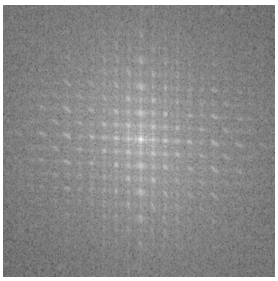


(d) FFT of Observed Image

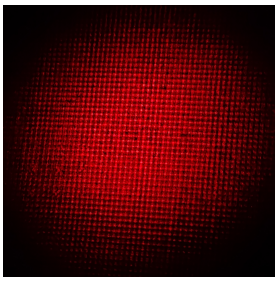
FIG. 14: Fraunhofer Diffraction Pattern for Grating, rotated at an angle



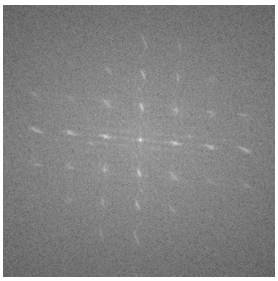
(a) At Fourier Plane



(b) FFT of Diffraction Pattern

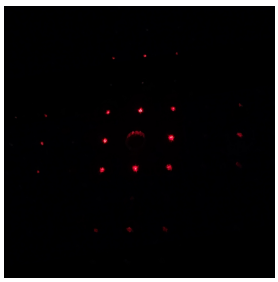


(c) At Observation Plane

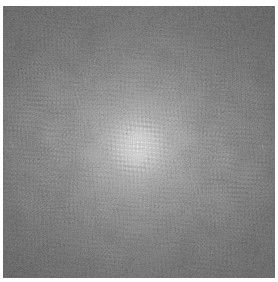


(d) FFT of Observed Image

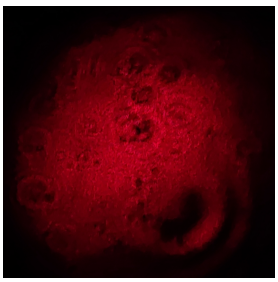
FIG. 15: Fraunhofer Diffraction Pattern for Mesh (No filter)



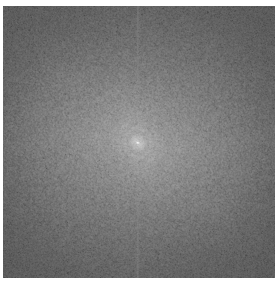
(a) At Fourier Plane



(b) FFT of Diffraction Pattern

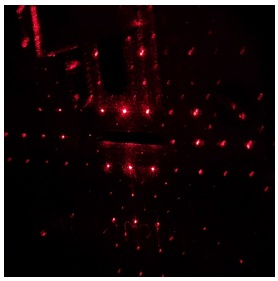


(c) At Observation Plane

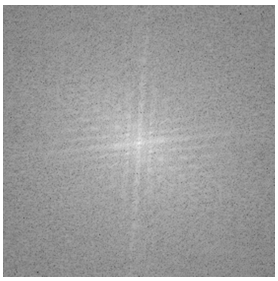


(d) FFT of Observed Image

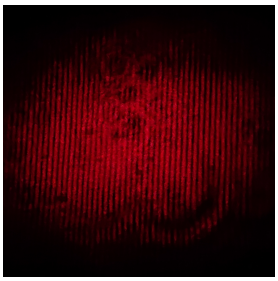
FIG. 16: Spatial Filtering: Mesh (Object) + Pinhole (A low-pass filter)



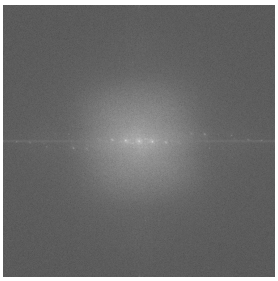
(a) At Fourier Plane



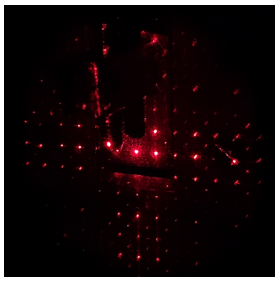
(b) FFT of Diffraction Pattern



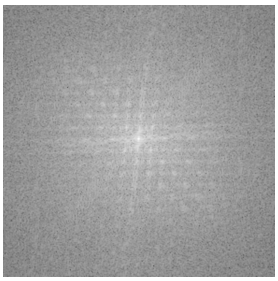
(c) At Observation Plane



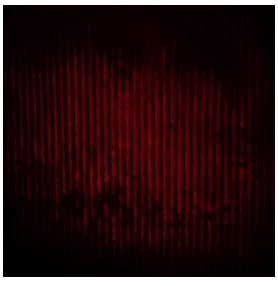
(d) FFT of Observed Image



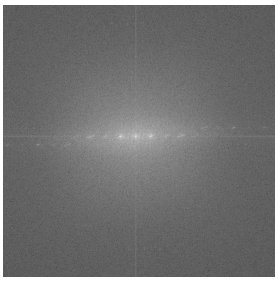
(e) At Fourier Plane



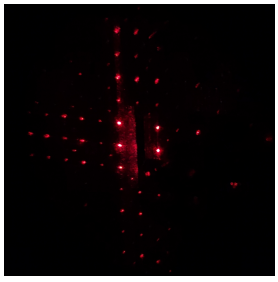
(f) FFT of Diffraction Pattern



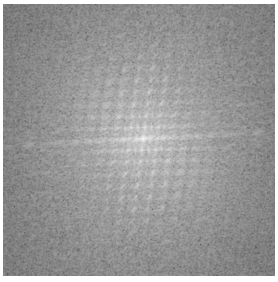
(g) At Observation Plane



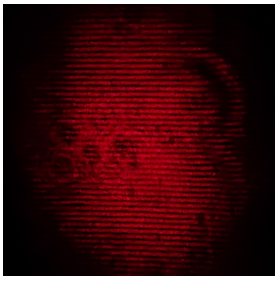
(h) FFT of Observed Image



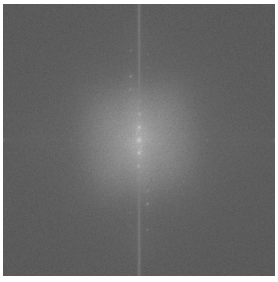
(i) At Fourier Plane



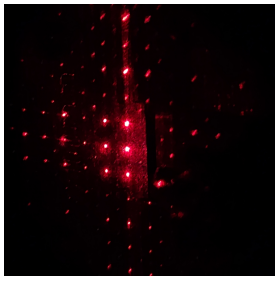
(j) FFT of Diffraction Pattern



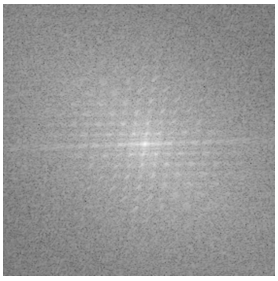
(k) At Observation Plane



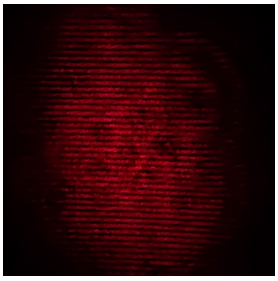
(l) FFT of Observed Image



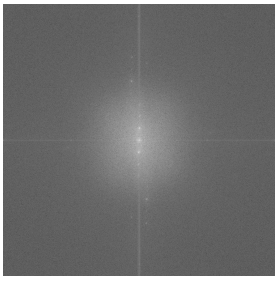
(m) At Fourier Plane



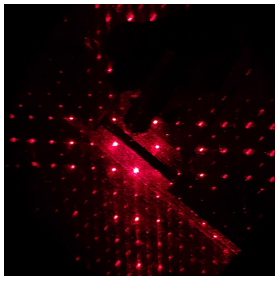
(n) FFT of Diffraction Pattern



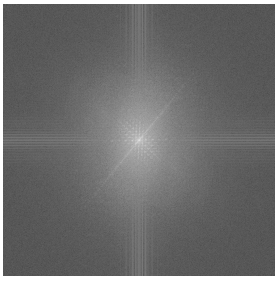
(o) At Observation Plane



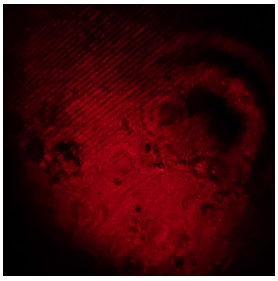
(p) FFT of Observed Image



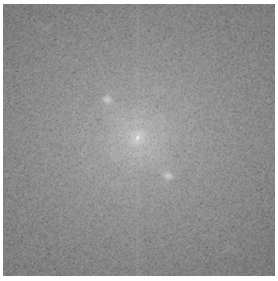
(q) At Fourier Plane



(r) FFT of Diffraction Pattern

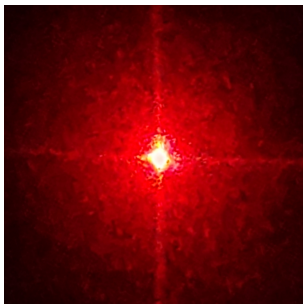


(s) At Observation Plane



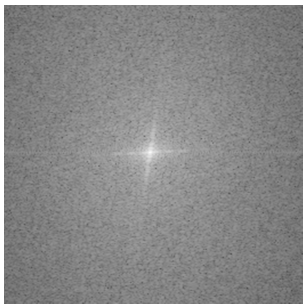
(t) FFT of Observed Image

FIG. 17: Spatial Filtering: Mesh (Object) + Adjustable Slit in various orientations

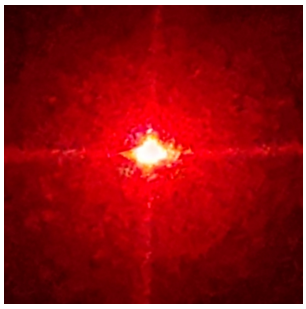


(a) At Fourier Plane

(b) At Observation Plane

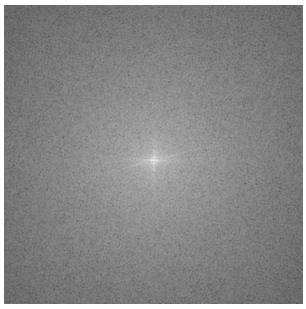


(c) FFT of Observed Image

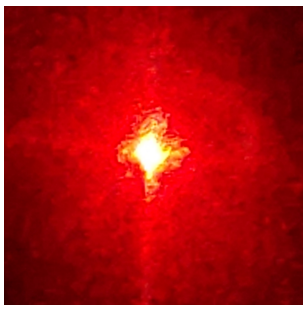


(d) At Fourier Plane

(e) At Observation Plane

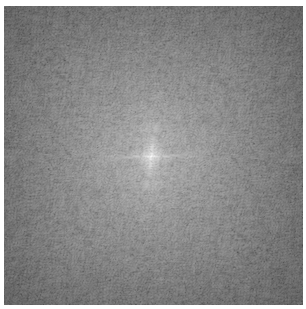


(f) FFT of Observed Image

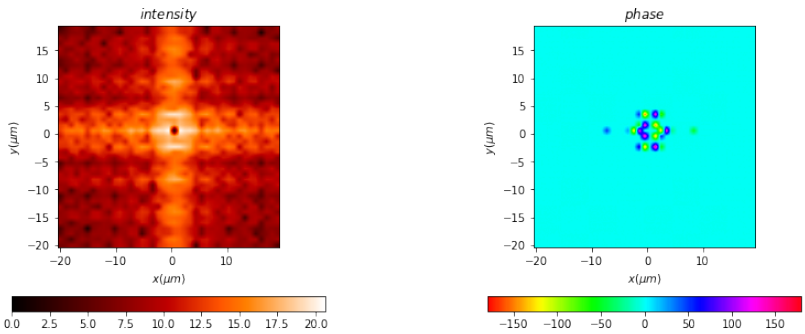


(g) At Fourier Plane

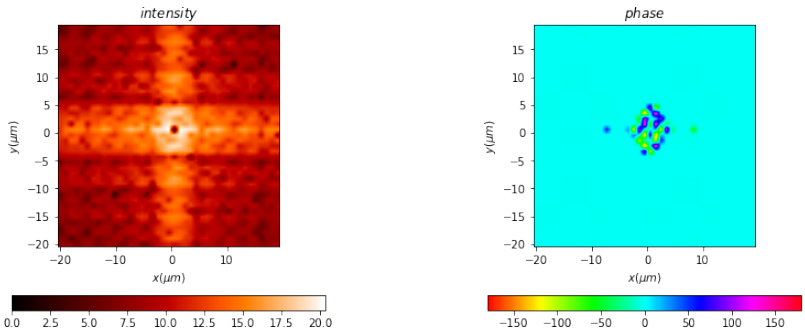
(h) At Observation Plane



(i) FFT of Observed Image

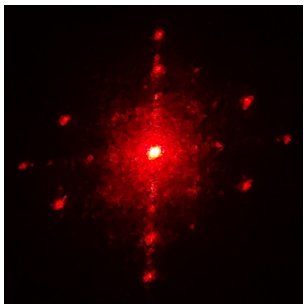


(j) Closer look at the diffracted field for 'E', using the Python code in Appendix - IV C

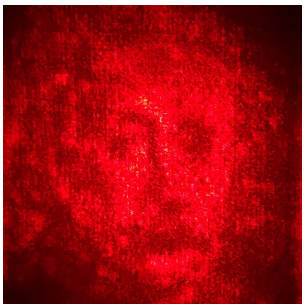


(k) Python Code: Closer look at the diffracted field for 'F', using the Python code in Appendix - IV C

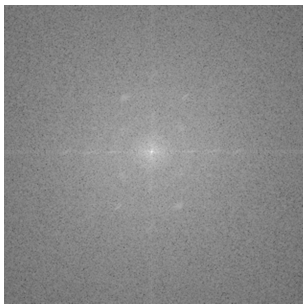
FIG. 18: Character Recognition: Alphabets 'E' and 'F'



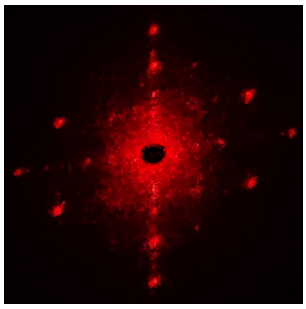
(a) At Fourier Plane, with No Filter



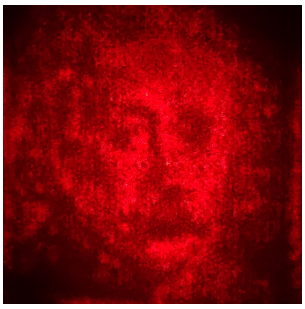
(b) At Observation Plane



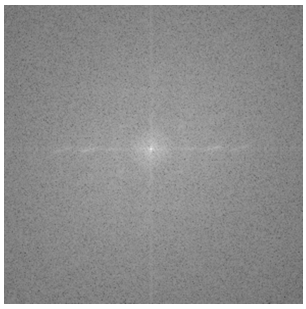
(c) FFT of Observed Image



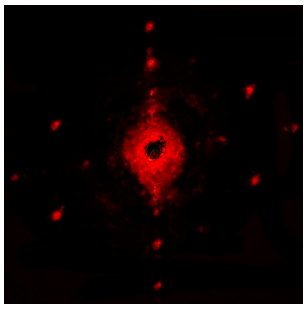
(d) At Fourier Plane, with a pinhole



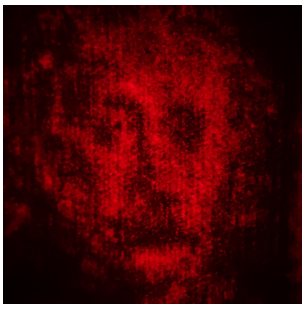
(e) At Observation Plane



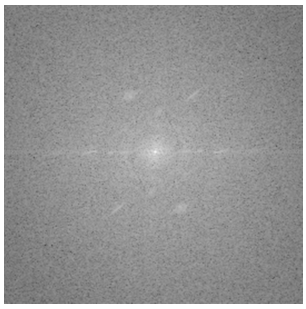
(f) FFT of Observed Image



(g) At Fourier Plane, with a black dot



(h) At Observation Plane



(i) FFT of Observed Image

FIG. 19: Spatial Filtering: Half-tone Image + Filters, showcasing Low and High Pass Filtering. Note that, the high pass filter also results in sharper edges (*Dark-Field Illumination*).

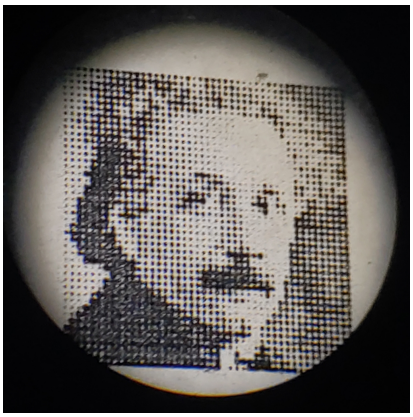


FIG. 20: Effect of an incoherent source (a flashlight) on the Diffraction Pattern. This image was taken at the Fourier plane of the setup. The diffraction pattern is nowhere to be seen and the picture is just an image of the source.

APPENDIX: PYTHON CODE

Basic Imports

In [1]:
```
import numpy as np
from diffractio import mm, um, degrees
from diffractio.scalar_sources_XY import Scalar_source_XY
from diffractio.scalar_masks_XY import Scalar_mask_XY


# Setting up
length = 1 * mm
num_data = 512
x0 = np.linspace(-length / 2, length / 2, num_data)
y0 = np.linspace(-length / 2, length / 2, num_data)
wavelength = 0.633 * um
```

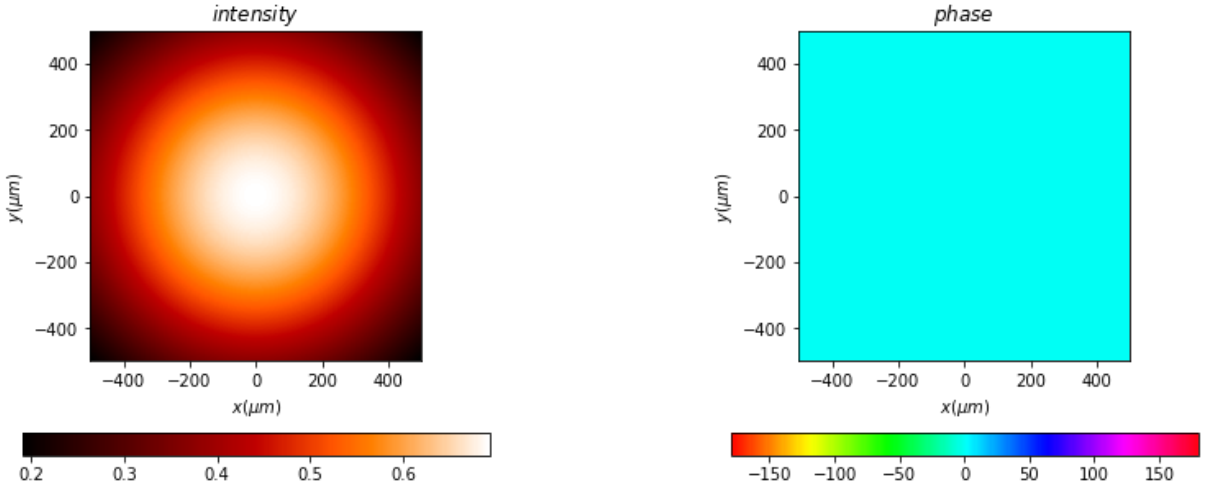
```
number of processors: 12
```

Setting up source

- Gaussian Beam (LASER)

In [2]:
```
# Gaussian Beam Source - like a LASER
u0 = Scalar_source_XY(x=x0, y=y0, wavelength=wavelength)
u0.gauss_beam(r0=(0, 0), w0=(800 * um, 800 * um), z0=0.0)
u0.draw(kind='field', logarithm=True)
```

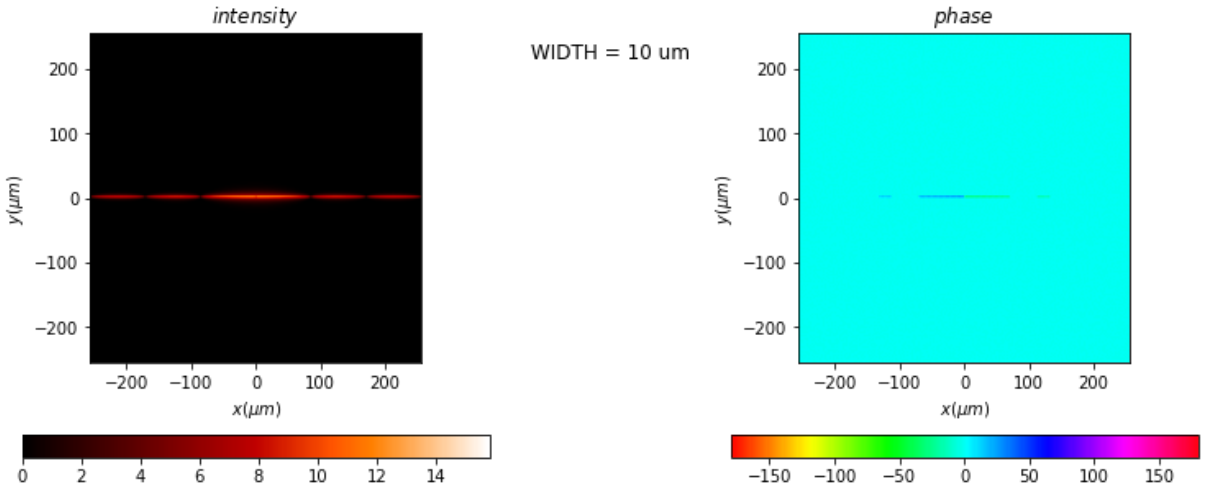
Out[2]:
```
((<matplotlib.image.AxesImage at 0x272b852da60>,
  <matplotlib.image.AxesImage at 0x272b87cc220>),
 None,
 None)
```

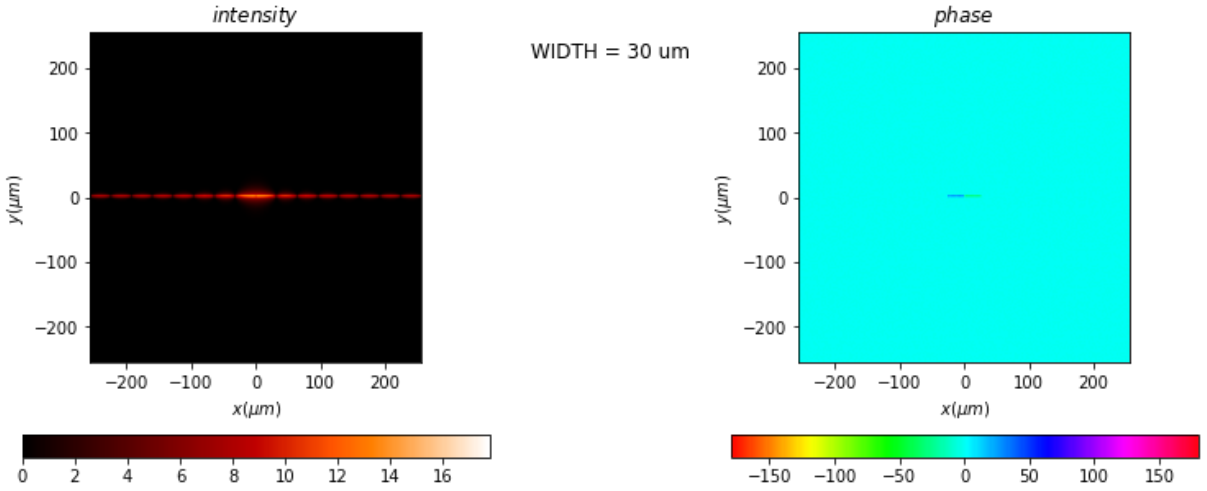


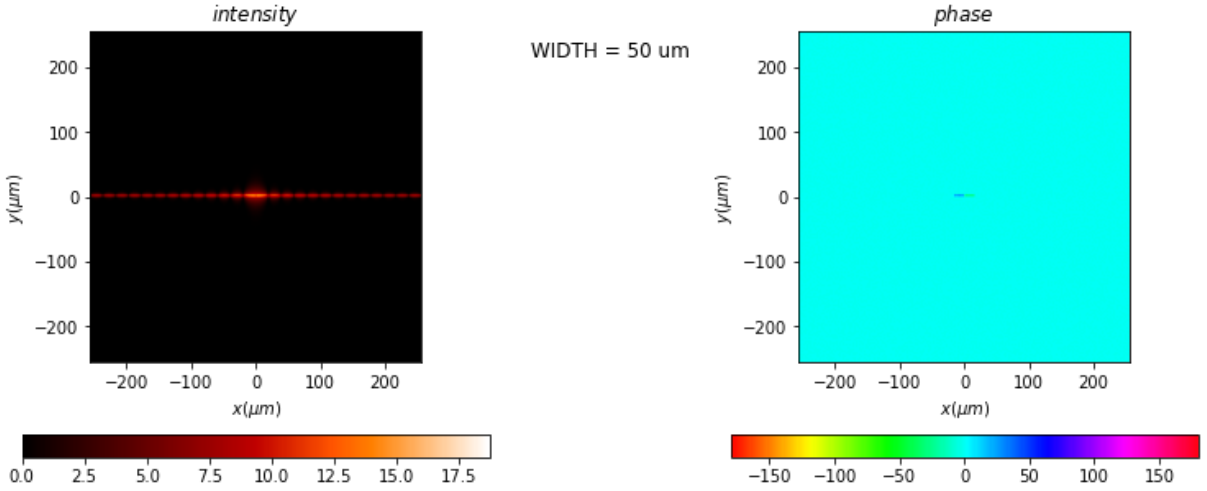
Slits

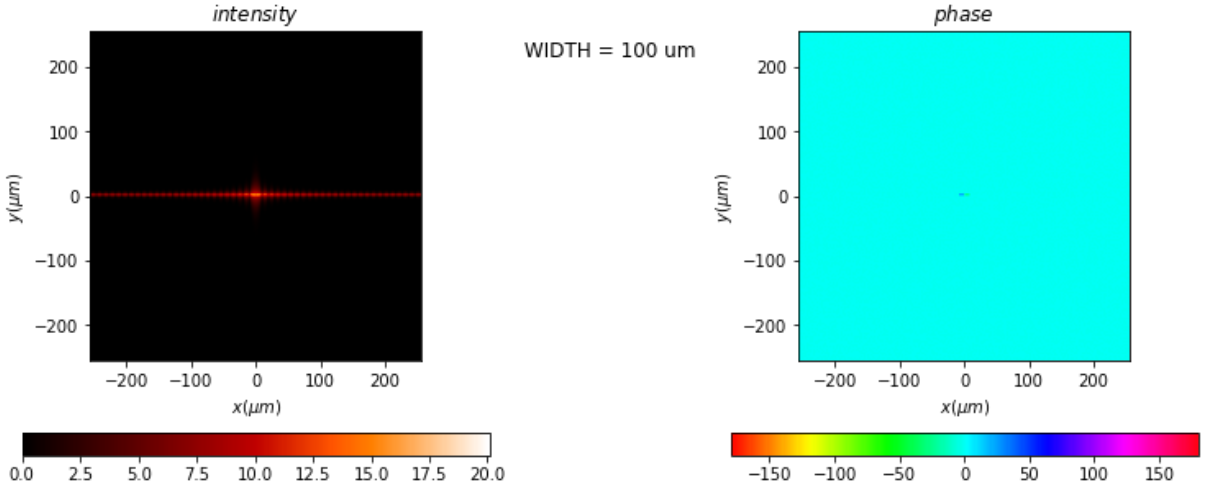
- Fraunhofer Diffraction Patterns for different slit widths

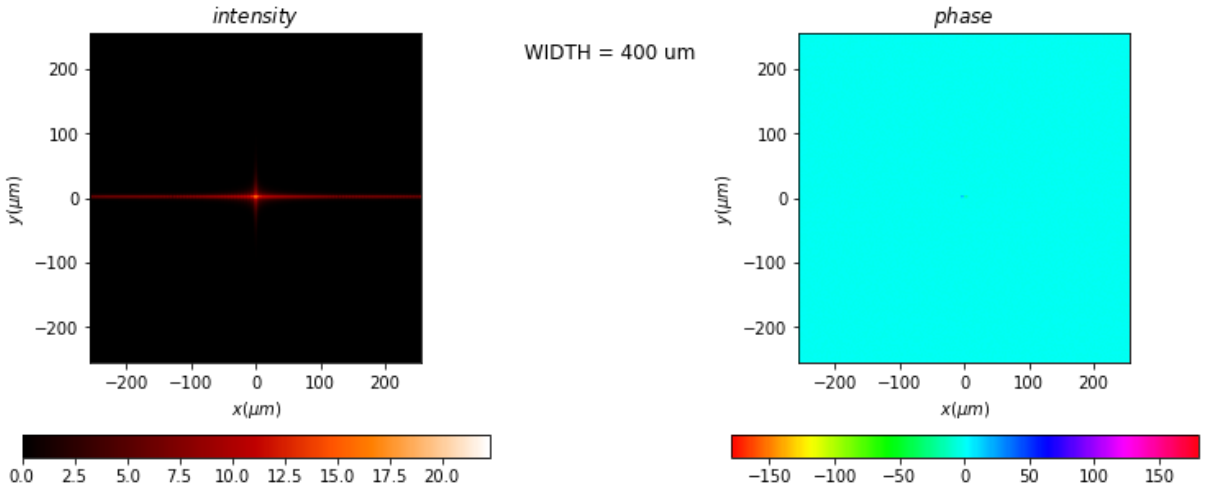
In [3]:
```
widths = [10, 30, 50, 100, 400]  # in um
for width in widths:
    varSlit = Scalar_mask_XY(x0, y0, wavelength)
    varSlit.slit(
        x0=0 * um,
        size=width * um
    )
#     varSlit.draw(kind='field', logarithm=True)
    a_varSlit = (u0 * varSlit).fft(z=1 * mm, new_field=True)
    a_varSlit.draw(title=f" WIDTH = {width} um ", kind='field', logarithm=True)
```


intensity
phase
WIDTH = 10 um
y(μm)
x(μm)


intensity
phase
WIDTH = 30 um
y(μm)
x(μm)


intensity
phase
WIDTH = 50 um
y(μm)
x(μm)


intensity
phase
WIDTH = 100 um
y(μm)
x(μm)



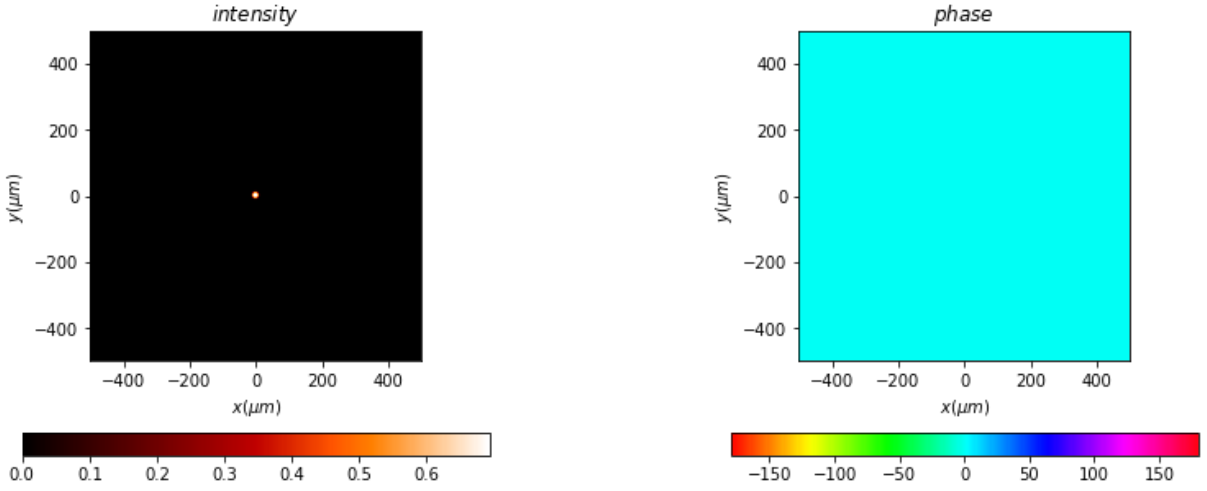
The intensity variation matches the expected profile.

Pinhole

- Fraunhofer Diffraction Patterns for different pinhole radii

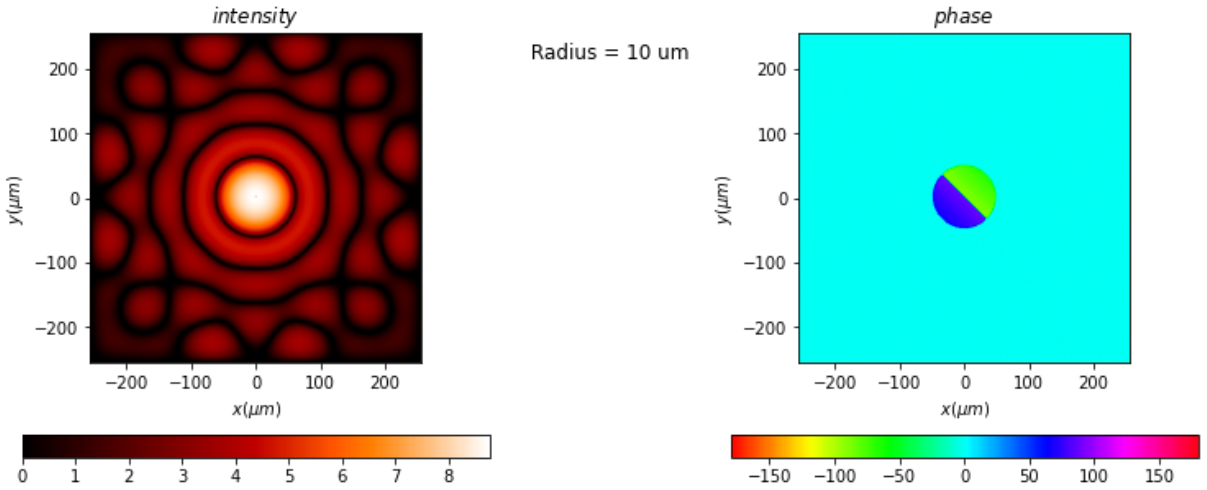
In [4]:
```
# Showcasing a pinhole
circ = Scalar_mask_XY(x0, y0, wavelength)
circ.circle(
    r0=(0 * um, 0 * um),
    radius=(10 * um, 10 * um),
    angle=0
)
circ.draw(kind='field', logarithm=True)
```

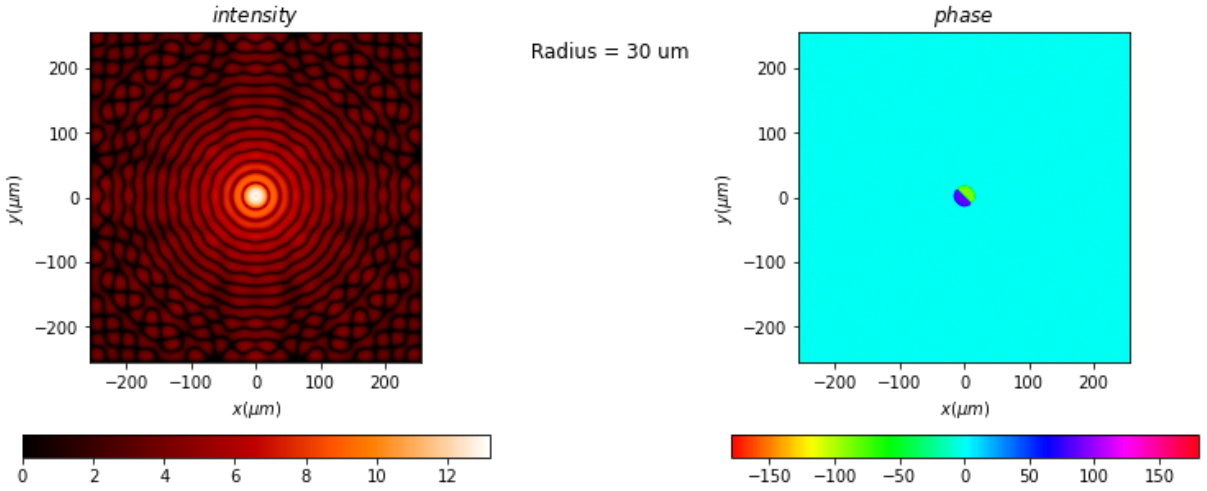
Out[4]:
```
((<matplotlib.image.AxesImage at 0x272bd337eb0>,
  <matplotlib.image.AxesImage at 0x272bd3aa6a0>),
 None,
 None)
```

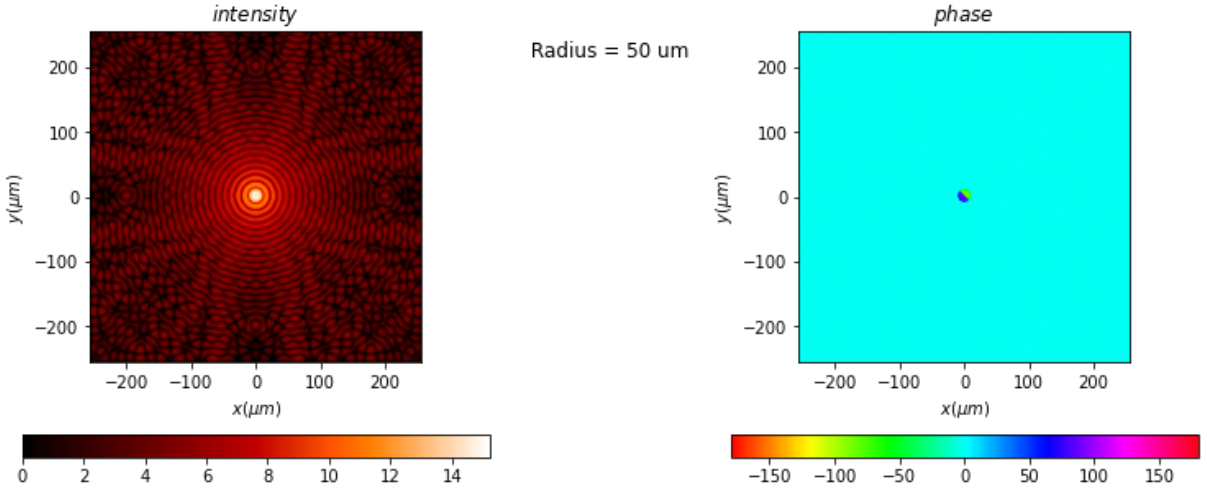


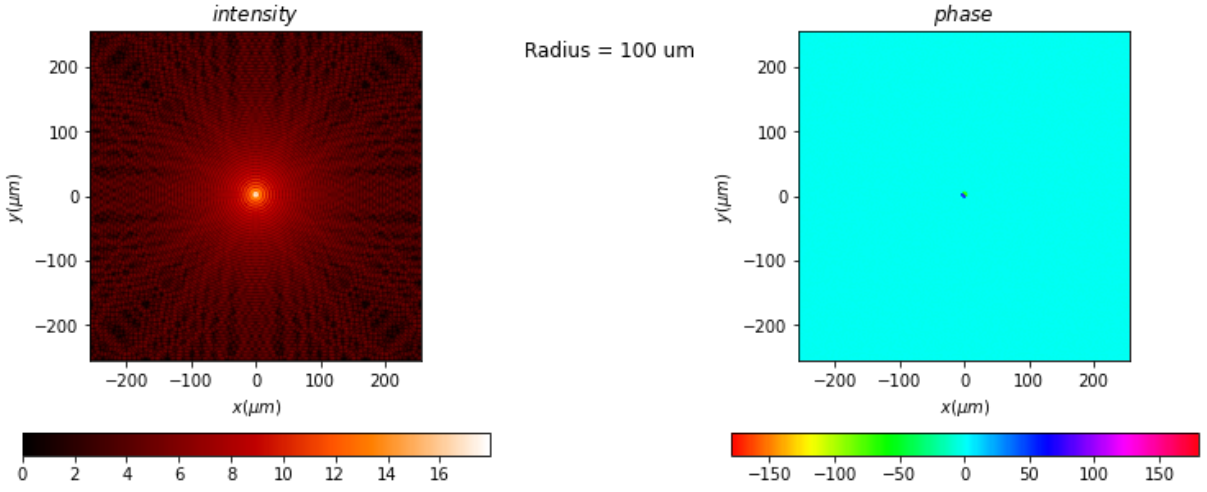
Diffraction pattern for multiple pinholes having different radius

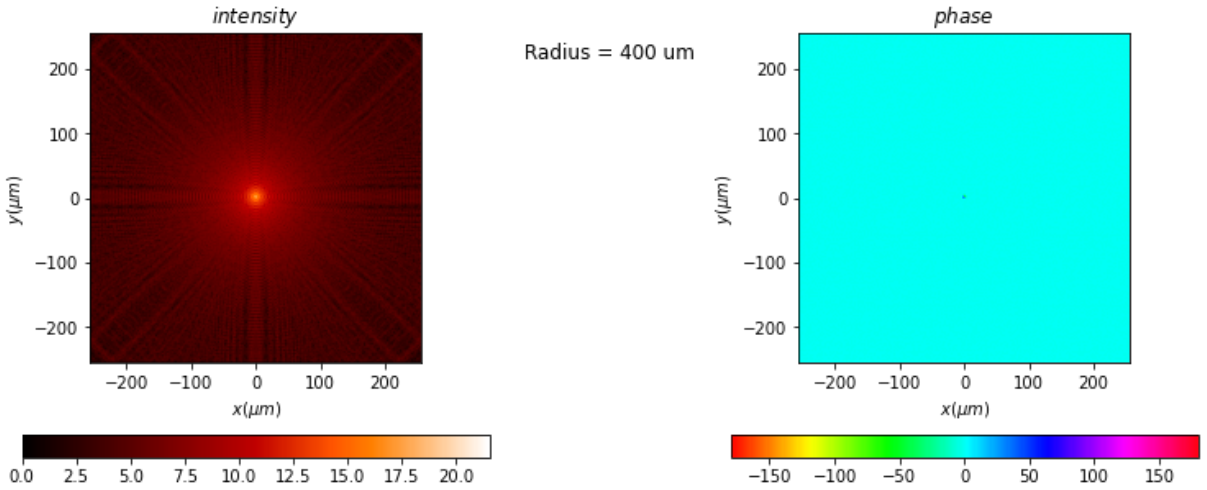
In [5]:
```
radii = [10, 30, 50, 100, 400]  # in um
for rad in radii:
    circ = Scalar_mask_XY(x0, y0, wavelength)
    circ.circle(
        r0=(0 * um, 0 * um),
        radius=(rad * um, rad * um),
        angle=0
    )

    a_circ = (u0 * circ).fft(z=1 * mm, new_field=True)
    a_circ.draw(title=f" Radius = {rad} um ", kind='field', logarithm=True)
```


intensity
phase
Radius = 10 um
x(μm)
y(μm)


intensity
phase
Radius = 30 um
x(μm)
y(μm)


intensity
phase
Radius = 50 um
x(μm)
y(μm)


intensity
phase
Radius = 100 um
x(μm)
y(μm)



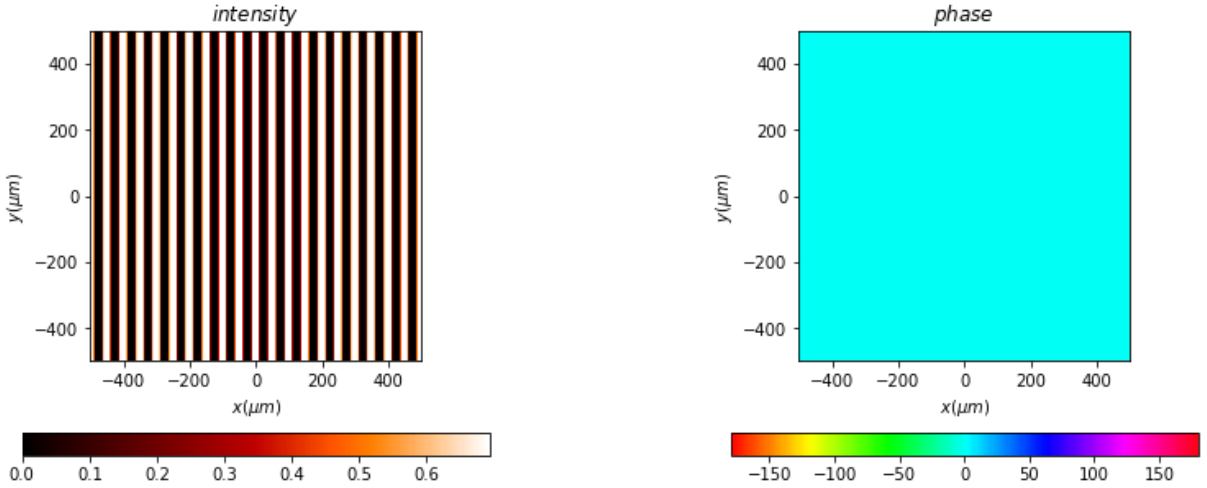
We observe the well-known Airy Disk in all cases, as theoretically expected.

Grating

- Fraunhofer Diffraction Patterns for grating

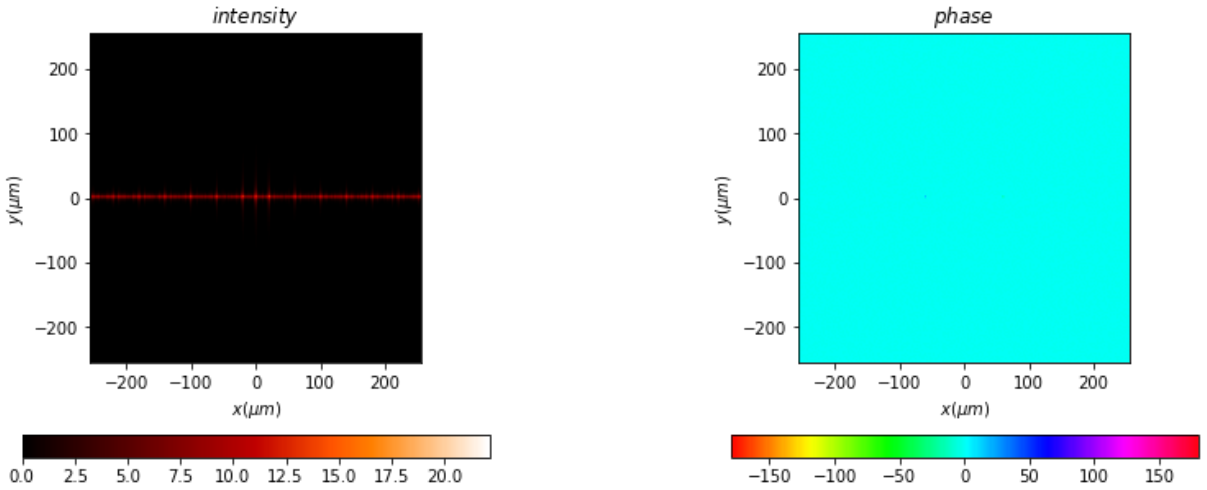
In [6]:
```
grating = Scalar_mask_XY(x0, y0, wavelength)
grating.ronchi_grating(
    period=50 * um,
)

grating.draw(kind='field', logarithm=True)
```

Out[6]:
```
((<matplotlib.image.AxesImage at 0x272bf6b80a0>,
  <matplotlib.image.AxesImage at 0x272bcbed820>),
 None,
 None)
```



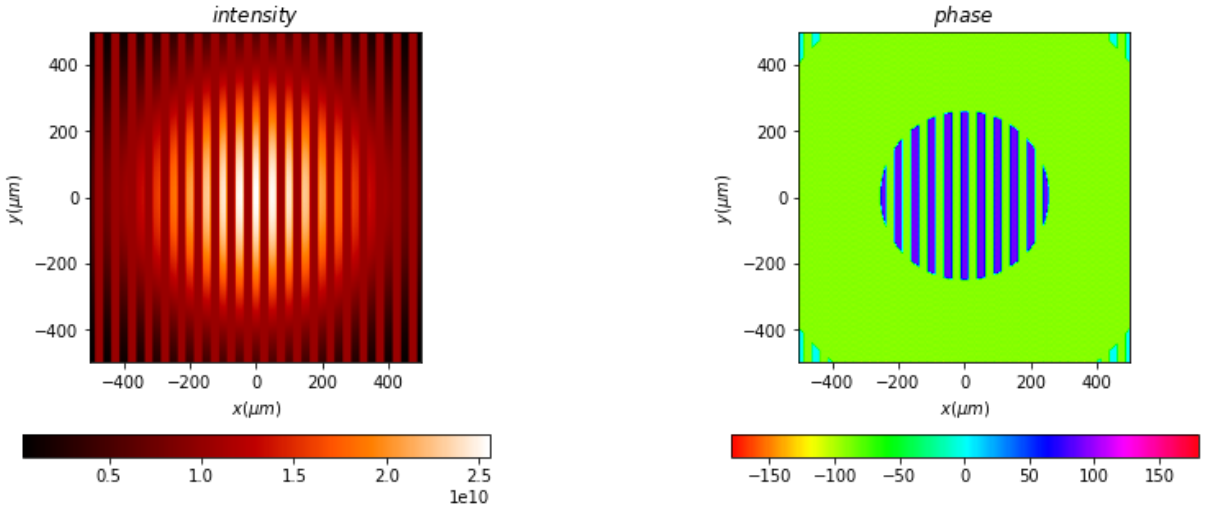
In [7]:
```
# Fourier Plane - No Filter
a_L1 = (u0 * grating).fft(z=1 * mm, new_field=True)
a_L1.draw(kind='field', logarithm=True)
```

Out[7]:
```
((<matplotlib.image.AxesImage at 0x272b95608e0>,
  <matplotlib.image.AxesImage at 0x272b954b0a0>),
 None,
 None)
```



In [8]:
```
a_L2 = a_L1.fft(z=1 * mm, shift=False, remove0=False, new_field=True)
a_L2.draw(kind='field', logarithm=False)
```

Out[8]:
```
((<matplotlib.image.AxesImage at 0x272b95ee6a0>,
  <matplotlib.image.AxesImage at 0x272b928b970>),
 None,
 None)
```



In [ ]:

Mesh

Compare results with figures in Eisenkraft (1977)

In [1]:
```
import numpy as np
from diffractio import mm, um, degrees
from diffractio.scalar_sources_XY import Scalar_source_XY
from diffractio.scalar_masks_XY import Scalar_mask_XY


# Setting up
length = 1 * mm
num_data = 512
x0 = np.linspace(-length / 2, length / 2, num_data)
y0 = np.linspace(-length / 2, length / 2, num_data)
wavelength = 0.633 * um
```

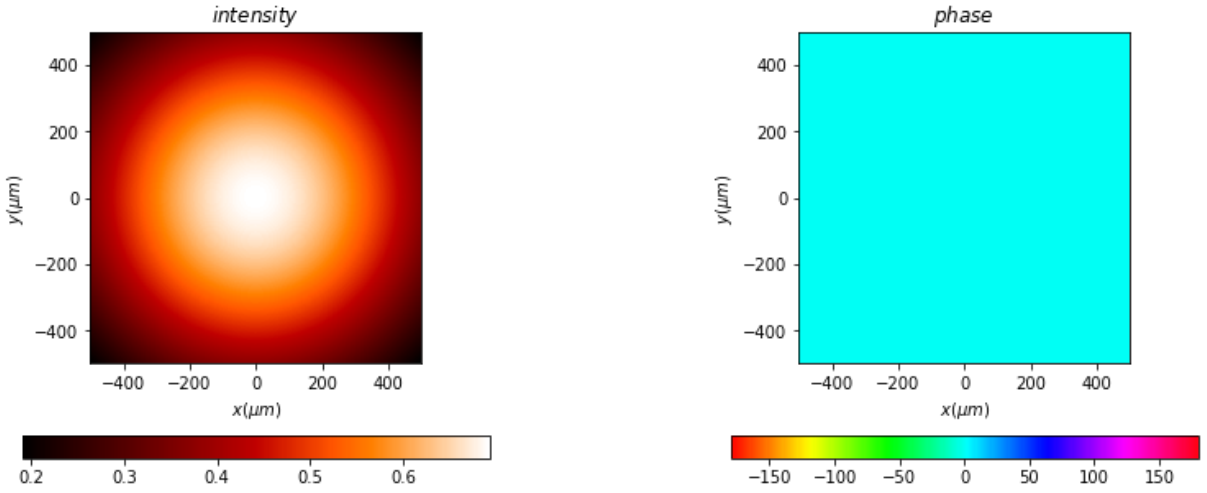
```
number of processors: 12
```

Setting up source(s)

- Gaussian Beam (LASER)

In [2]:
```
# Gaussian Beam Source - like a LASER
u0 = Scalar_source_XY(x=x0, y=y0, wavelength=wavelength)
u0.gauss_beam(r0=(0, 0), w0=(800 * um, 800 * um), z0=0.0)
u0.draw(kind='field', logarithm=True)
```

Out[2]:
```
((<matplotlib.image.AxesImage at 0x249d2c1ea60>,
  <matplotlib.image.AxesImage at 0x249d2ebc220>),
 None,
 None)
```



Mesh

In [3]:
```
mesh = Scalar_mask_XY(x0, y0, wavelength)
mesh.grating_2D(
    period=50 * um,
    fill_factor=0.5,
    angle=0 * degrees
)

mesh.draw(kind='field', logarithm=True)
```

Out[3]:
```
((<matplotlib.image.AxesImage at 0x249d36f11c0>,
  <matplotlib.image.AxesImage at 0x249d3960940>),
 None,
 None)
```

Basic Imports

In [1]:
```
import numpy as np
from diffractio import mm, um, degrees
from diffractio.scalar_sources_XY import Scalar_source_XY
from diffractio.scalar_masks_XY import Scalar_mask_XY


# Setting up
length = 1 * mm
num_data = 512
x0 = np.linspace(-length / 2, length / 2, num_data)
y0 = np.linspace(-length / 2, length / 2, num_data)
wavelength = 0.633 * um
```

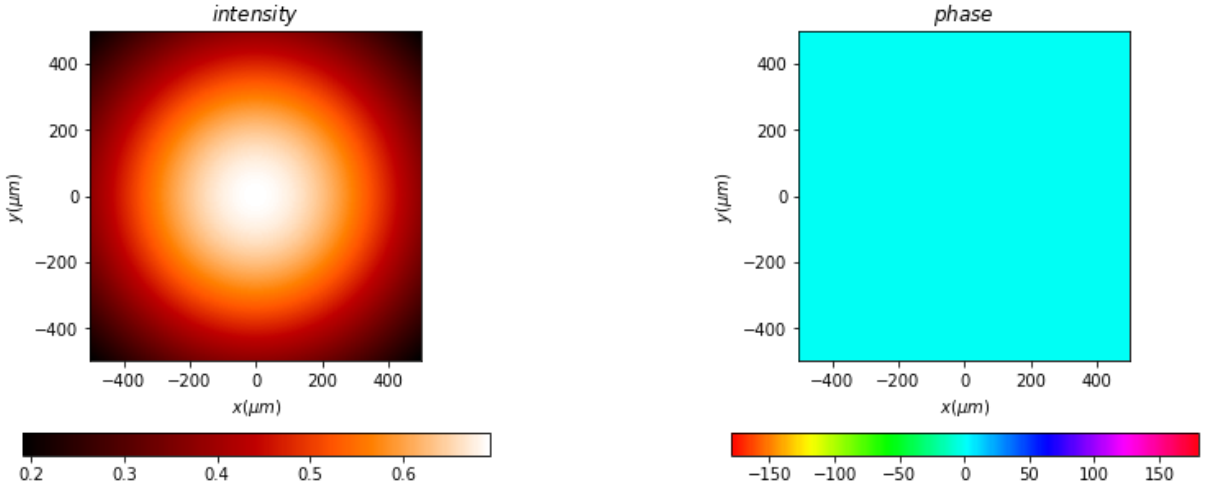
```
number of processors: 12
```

Setting up source

- Gaussian Beam (LASER)

In [2]:
```
# Gaussian Beam Source - like a LASER
u0 = Scalar_source_XY(x=x0, y=y0, wavelength=wavelength)
u0.gauss_beam(r0=(0, 0), w0=(800 * um, 800 * um), z0=0.0)
u0.draw(kind='field', logarithm=True)
```

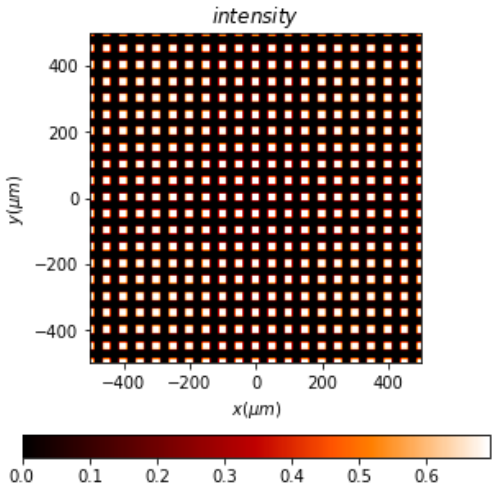
Out[2]:
```
((<matplotlib.image.AxesImage at 0x272b852da60>,
  <matplotlib.image.AxesImage at 0x272b87cc220>),
 None,
 None)
```

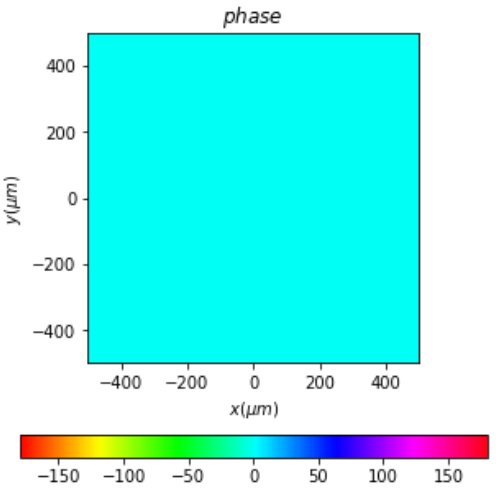


Slits

- Fraunhofer Diffraction Patterns for different slit widths

In [3]:
```
widths = [10, 30, 50, 100, 400]  # in um
for width in widths:
    varSlit = Scalar_mask_XY(x0, y0, wavelength)
    varSlit.slit(
        x0=0 * um,
        size=width * um
    )
#     varSlit.draw(kind='field', logarithm=True)
    a_varSlit = (u0 * varSlit).fft(z=1 * mm, new_field=True)
    a_varSlit.draw(title=f" WIDTH = {width} um ", kind='field', logarithm=True)
```

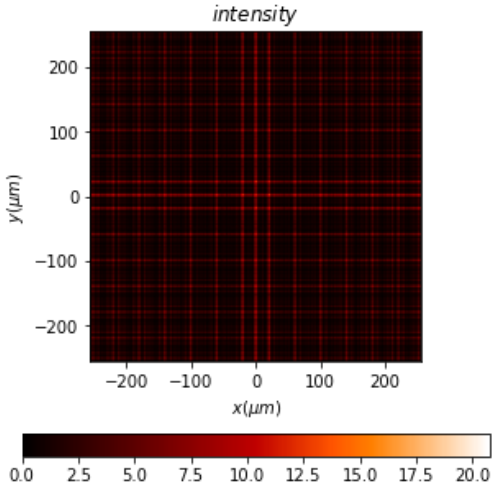


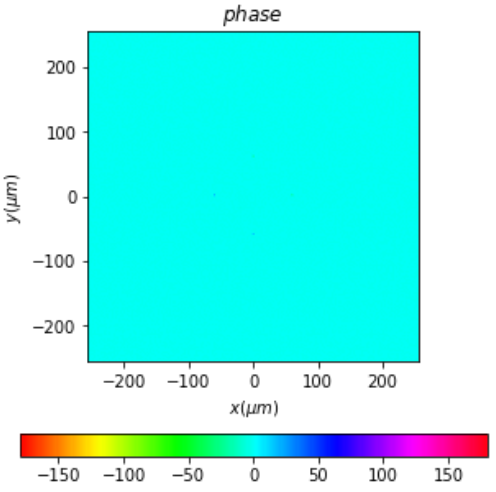


In [4]:
```
# angledMesh = Scalar_mask_XY(x0, y0, wavelength)
# angledMesh.grating_2D(
#     period=50 * um,
#     fill_factor=0.5,
#     angle=45 * degrees
# )

# angledMesh.draw(kind='field', logarithm=True)
```

In [5]:
```
# Fourier Plane - No Filter
a_L1 = (u0 * mesh).fft(z=1 * mm, new_field=True)
a_L1.draw(kind='field', logarithm=True)
```

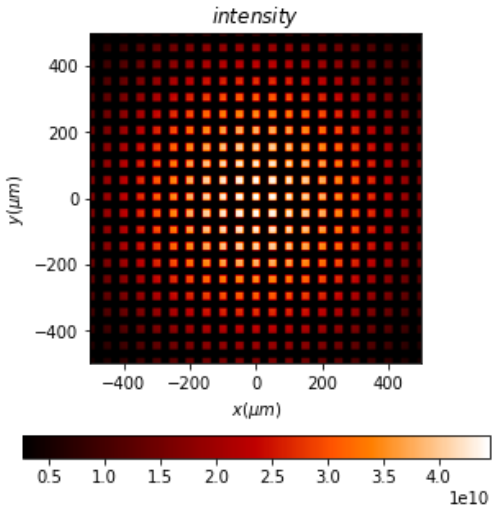
Out[5]:
```
((<matplotlib.image.AxesImage at 0x249d356fe80>,
  <matplotlib.image.AxesImage at 0x249d3607670>),
 None,
 None)
```

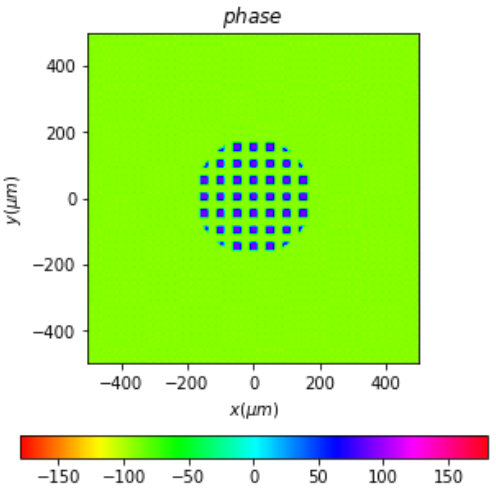




In [6]:
```
# This is how, it'd look without any filter, at the Observation screen
a_L2 = a_L1.fft(z=1 * mm, shift=False, remove0=False, new_field=True)
a_L2.draw(kind='field', logarithm=False)
```

Out[6]:
```
((<matplotlib.image.AxesImage at 0x249d3deaee0>,
  <matplotlib.image.AxesImage at 0x249d6a856d0>),
 None,
 None)
```





Masks / Filters

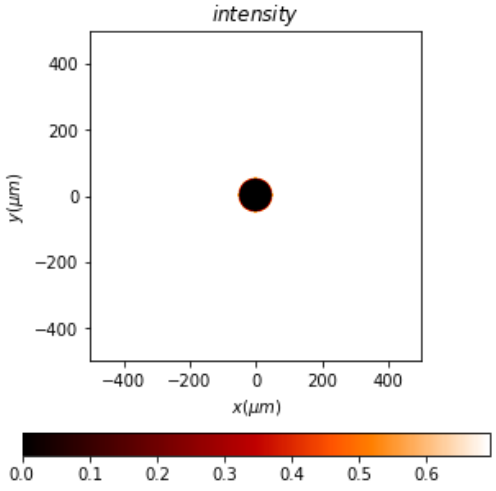
- Center Dot - High Pass
- Square - Low Pass
- Square + Center Dot

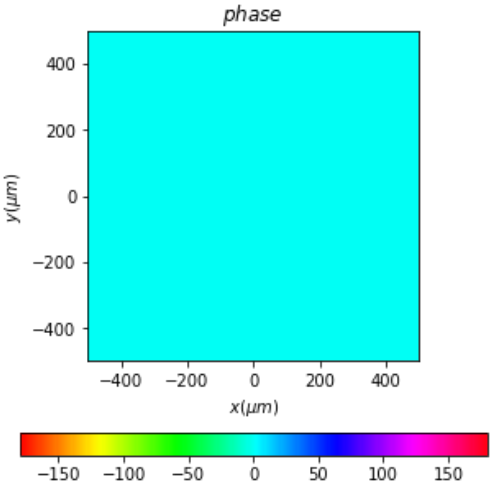
- Vertical Slit

- Horizontal Slit
- Angled Slit

In [7]:
```
cDot = Scalar_mask_XY(x0, y0, wavelength)
cDot.ring(
    r0=(0 * um, 0 * um),
    radius1=(50 * um, 50 * um),
    radius2=(1000 * um, 1000 * um)
)
cDot.draw(kind='field', logarithm=True)
```

Out[7]:
```
((<matplotlib.image.AxesImage at 0x249d6b5e460>,
  <matplotlib.image.AxesImage at 0x249d6bcdbe0>),
 None,
 None)
```

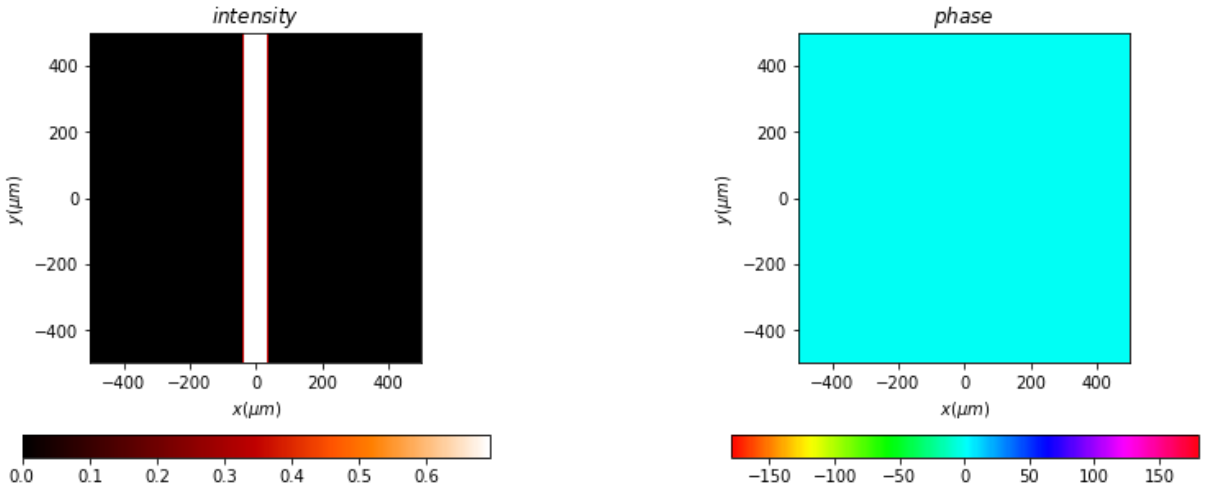




In [8]:
```
vert = Scalar_mask_XY(x0, y0, wavelength)
vert.slit(
    x0=0 * um,
    size=75 * um
)
vert.draw(kind='field', logarithm=True)
```

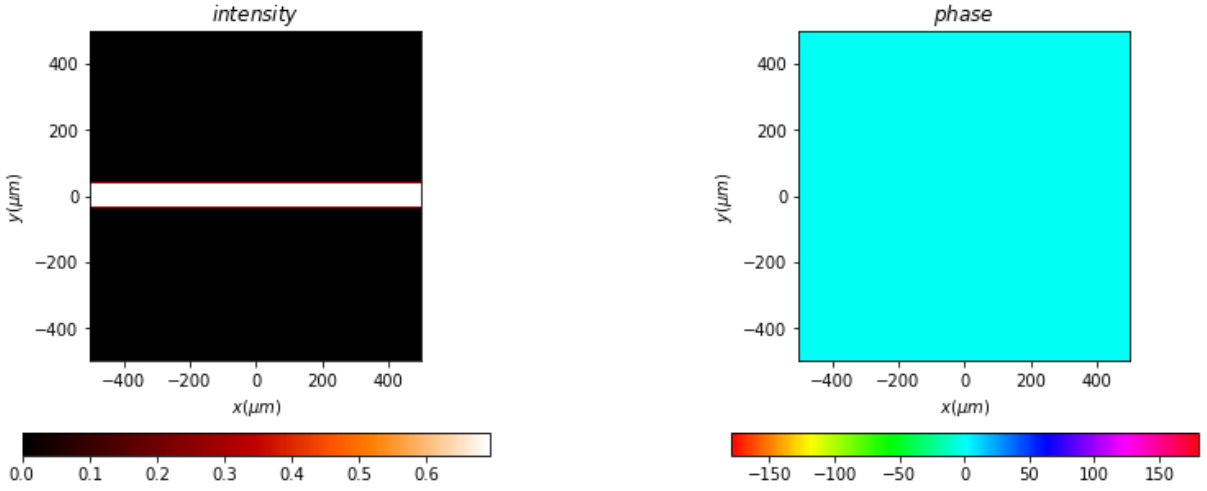
Out[8]:
```
((<matplotlib.image.AxesImage at 0x249d3695460>,
```

```
 <matplotlib.image.AxesImage at 0x249d2e91ac0>),
None,
None)
```



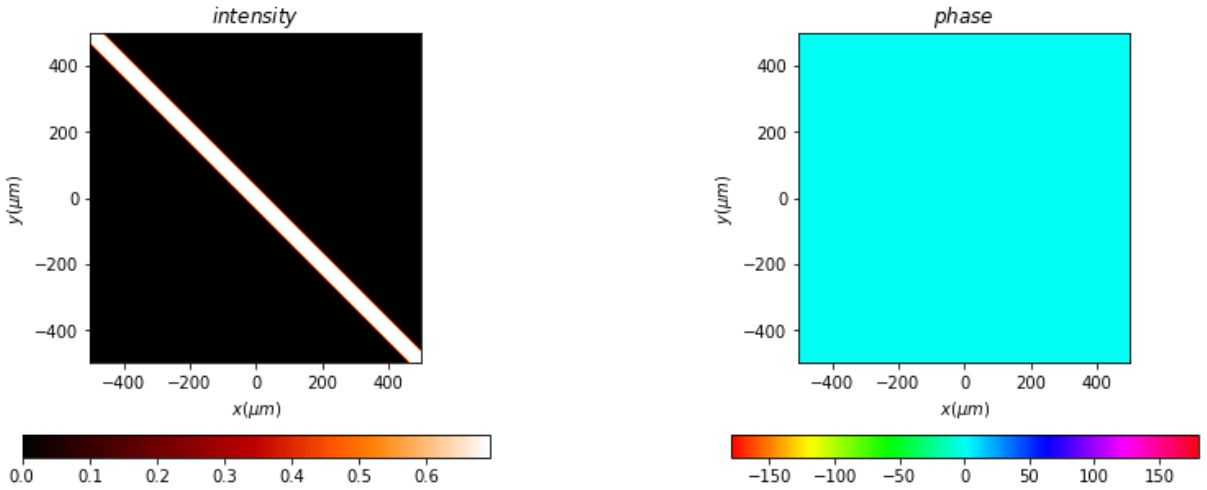
In [9]:
```
horiz = Scalar_mask_XY(x0, y0, wavelength)
horiz.slit(
    x0=0 * um,
    size=75 * um,
    angle=np.pi / 2
)
horiz.draw(kind='field', logarithm=True)
```

Out[9]:
```
((<matplotlib.image.AxesImage at 0x249d8190130>,
  <matplotlib.image.AxesImage at 0x249d81e08e0>),
 None,
 None)
```



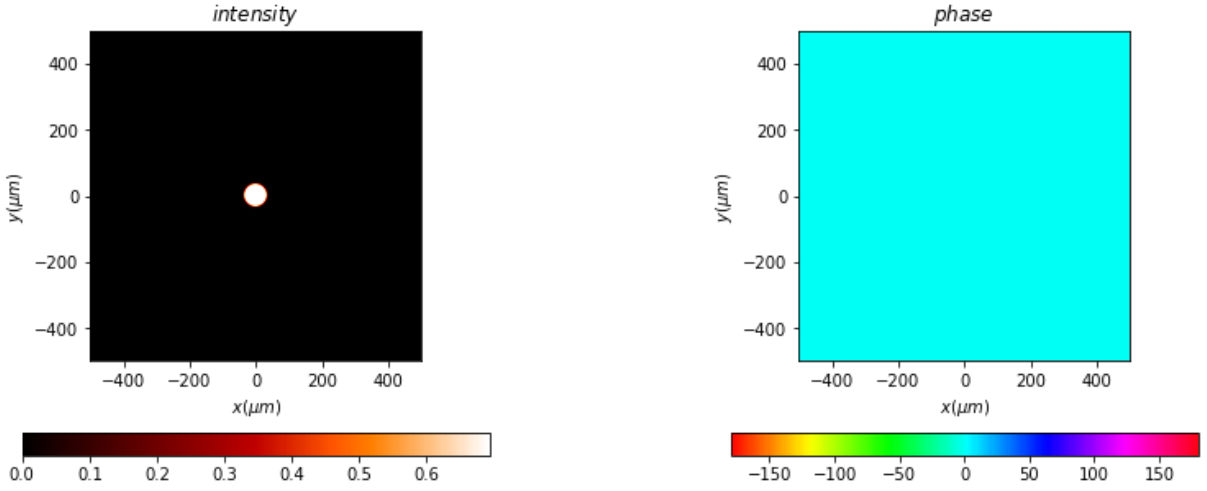
In [10]:
```
angled = Scalar_mask_XY(x0, y0, wavelength)
angled.slit(
    x0=0 * um,
    size=50 * um,
    angle=np.pi / 4
)
angled.draw(kind='field', logarithm=True)
```

Out[10]:
```
((<matplotlib.image.AxesImage at 0x249d8f35a00>,
  <matplotlib.image.AxesImage at 0x249d8f90220>),
 None,
 None)
```



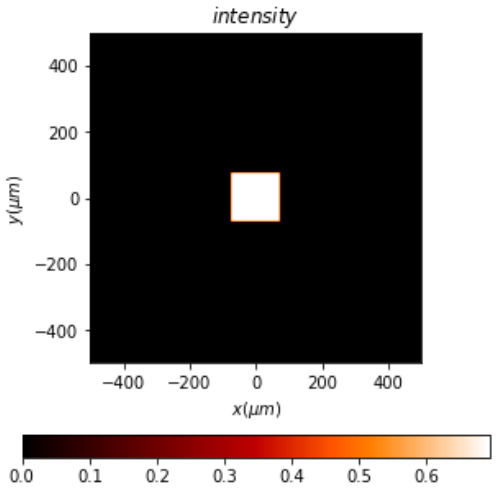
In [11]:
```
circ = Scalar_mask_XY(x0, y0, wavelength)
circ.circle(
    r0=(0 * um, 0 * um),
    radius=(35 * um, 35 * um),
    angle=0
)
circ.draw(kind='field', logarithm=True)
```

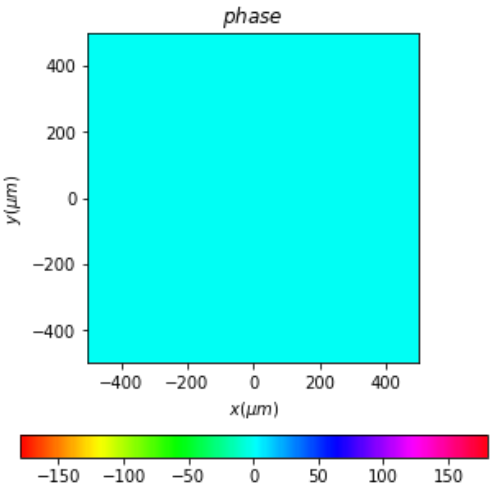
Out[11]:
```
((<matplotlib.image.AxesImage at 0x249d90575e0>,
  <matplotlib.image.AxesImage at 0x249d90d0910>),
 None,
 None)
```



In [12]:
```
sq = Scalar_mask_XY(x0, y0, wavelength)
sq.square(
    r0=(0 * um, 0 * um),
    size=(150 * um, 150 * um),
    angle=0,
)
sq.draw(kind='field', logarithm=True)
```

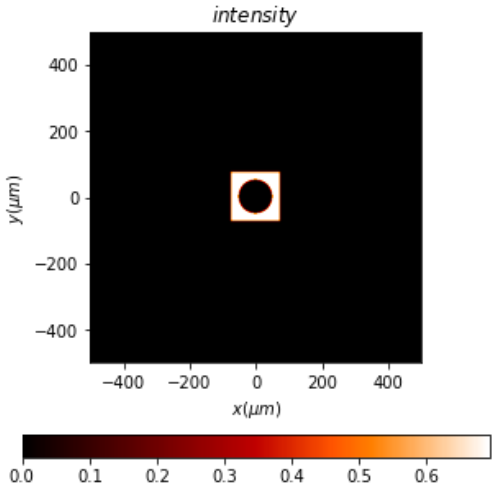
Out[12]:
```
((<matplotlib.image.AxesImage at 0x249da61ddc0>,
  <matplotlib.image.AxesImage at 0x249da698070>),
 None,
 None)
```

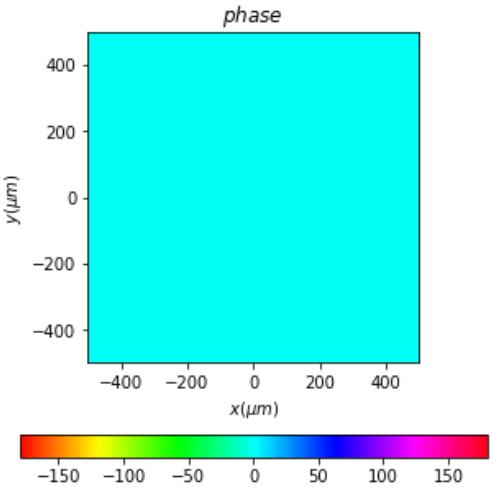




In [13]:
```
# Adding two filters
sqCirc = sq * cDot
sqCirc.draw(kind="field", logarithm=True)
```

Out[13]:
```
((<matplotlib.image.AxesImage at 0x249da6da160>,
  <matplotlib.image.AxesImage at 0x249d8146f70>),
 None,
 None)
```





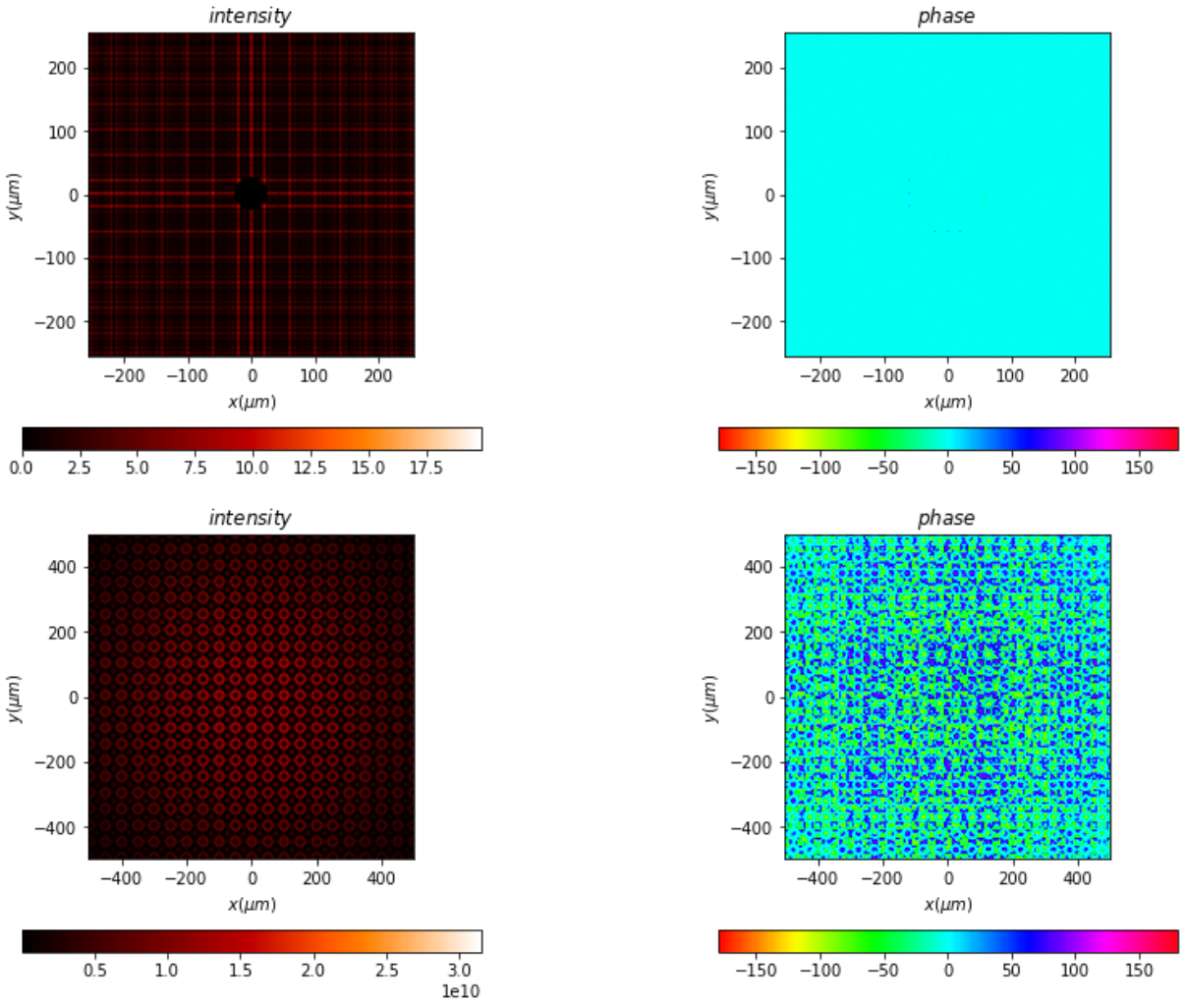
Returns wave after encountering filter and then L2

In [14]:
```
def sim(a_L1, mask):
    """
    a_L1: Wave after passing through Lens 1
    mask: Mask or Filter to apply

    """
    a_L1_Mask = a_L1 * mask
    a_L1_Mask_L2 = a_L1_Mask.fft(z=1 * mm, shift=False, remove0=False, new_field=True)

    return a_L1_Mask, a_L1_Mask_L2
```

Center Dot - High Pass - EDGE DETECTION

In [15]:
```
a_L1_cD, a_L1_cD_L2 = sim(a_L1, cDot)
a_L1_cD.draw(kind='field', logarithm=True)
a_L1_cD_L2.draw(kind='field', logarithm=False)
```

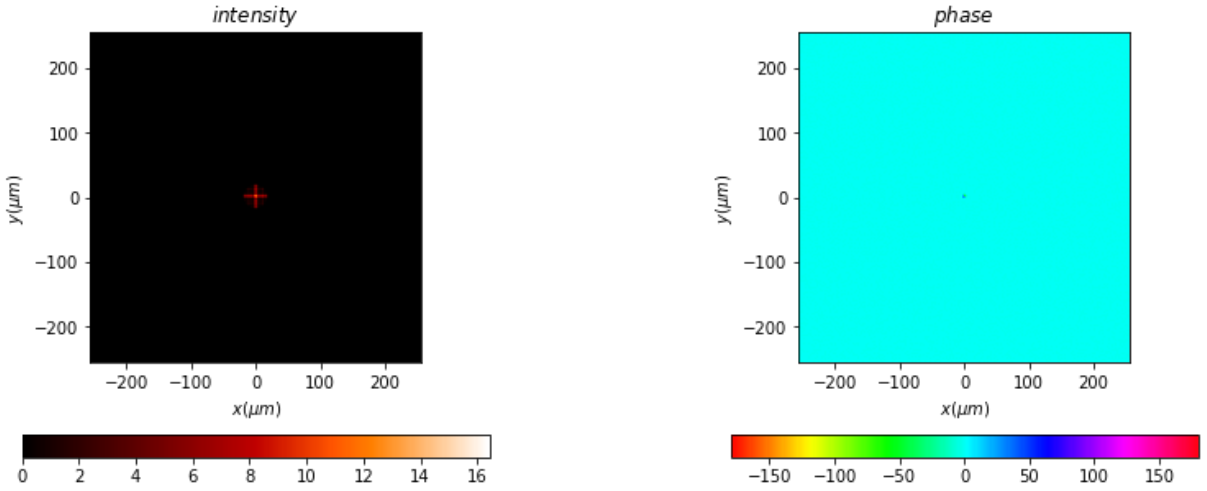
Out[15]:
```
((<matplotlib.image.AxesImage at 0x249db350ee0>,
  <matplotlib.image.AxesImage at 0x249dbe436d0>),
 None,
 None)
```

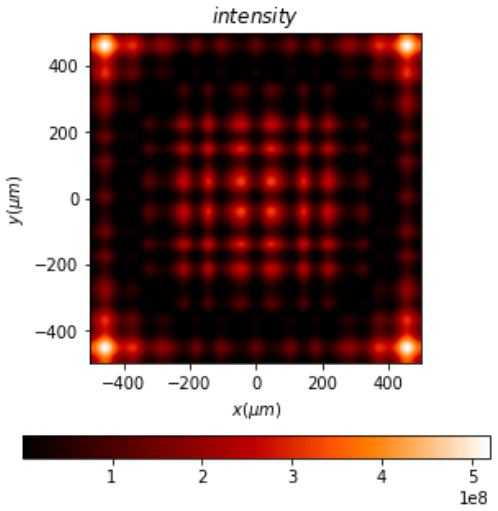


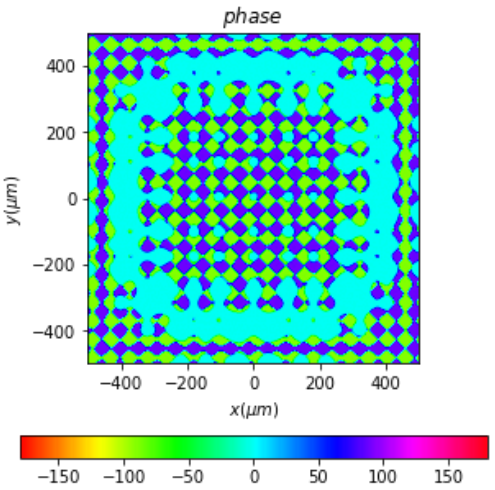
Circle - Low Pass - Blurred Output

In [16]:
```
a_L1_Circ, a_L1_Circ_L2 = sim(a_L1, circ)
a_L1_Circ.draw(kind='field', logarithm=True)
a_L1_Circ_L2.draw(kind='field', logarithm=False)
```

Out[16]:
```
((<matplotlib.image.AxesImage at 0x249dcee2a00>,
  <matplotlib.image.AxesImage at 0x249dcf4dd00>),
 None,
 None)
```



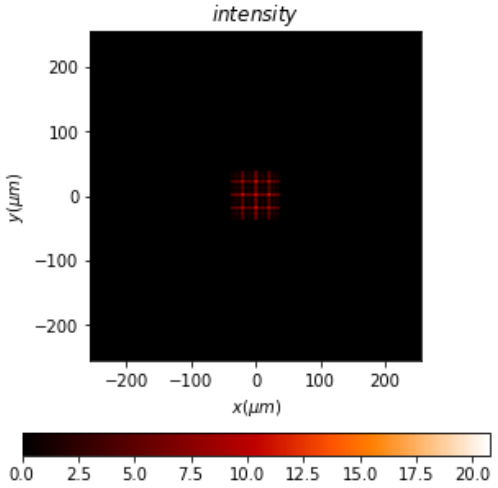


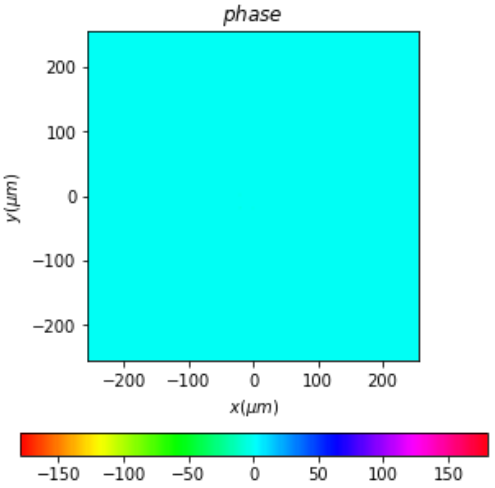


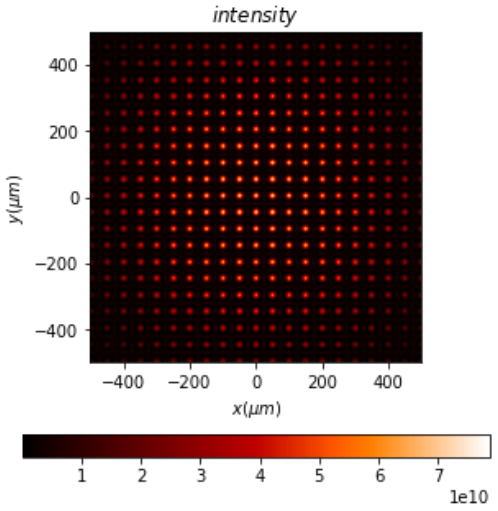
Square

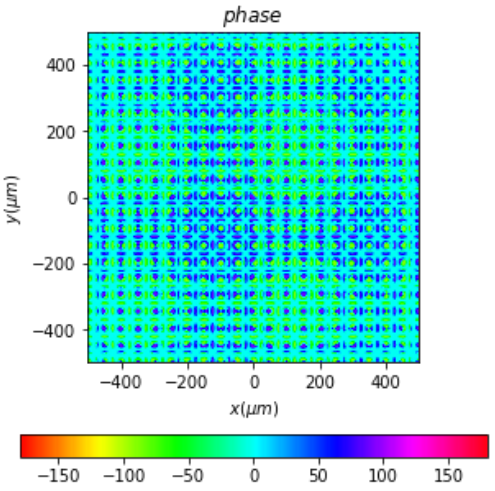
In [17]:
```
a_L1_Sq, a_L1_Sq_L2 = sim(a_L1, sq)
a_L1_Sq.draw(kind='field', logarithm=True)
a_L1_Sq_L2.draw(kind='field', logarithm=False)
```

Out[17]:
```
((<matplotlib.image.AxesImage at 0x249d8f9f0d0>,
  <matplotlib.image.AxesImage at 0x249dcea9220>),
 None,
 None)
```





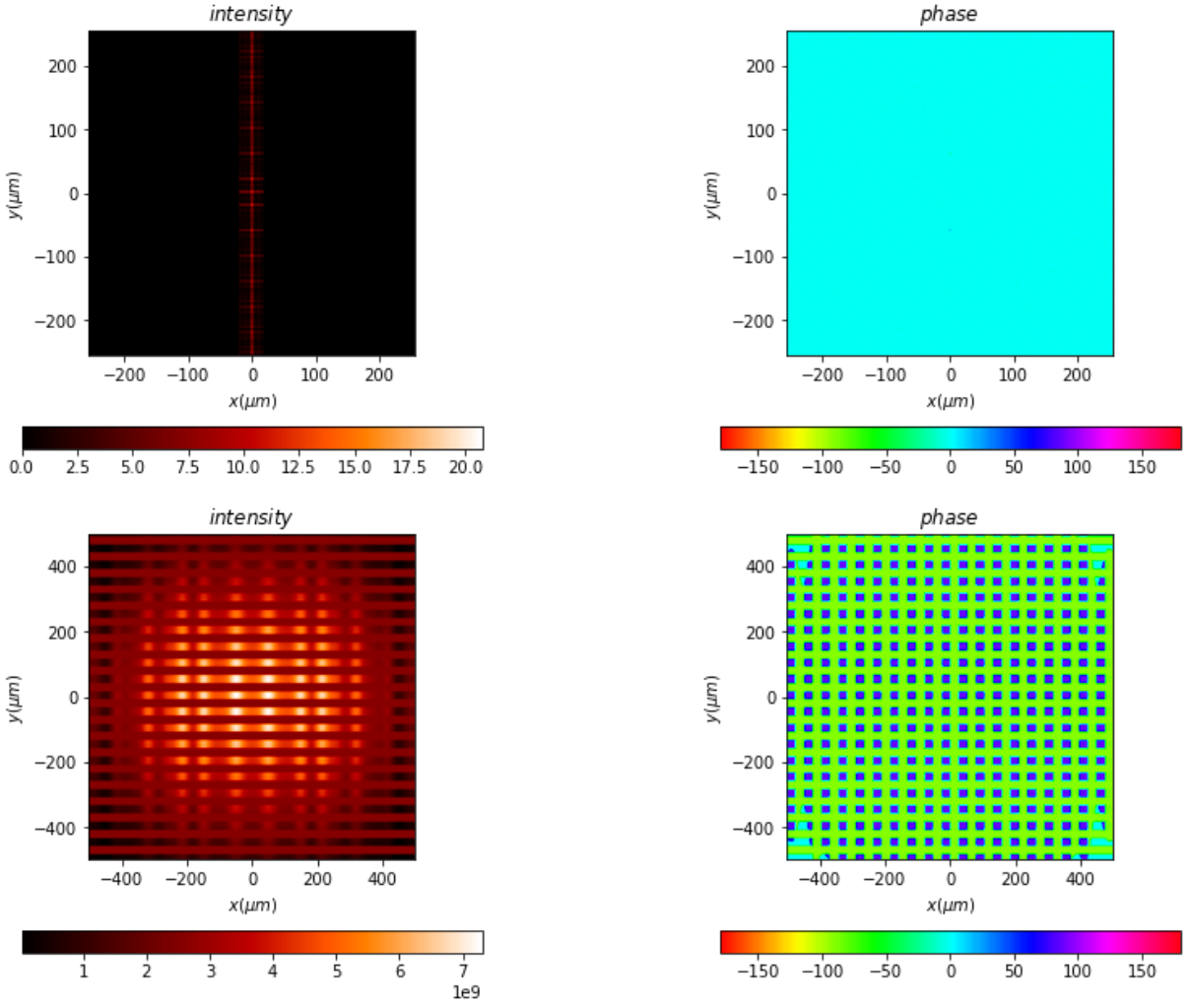




Vertical Slit

In [18]:
```
a_L1_Vert, a_L1_Vert_L2 = sim(a_L1, vert)
a_L1_Vert.draw(kind='field', logarithm=True)
a_L1_Vert_L2.draw(kind='field', logarithm=False)
```

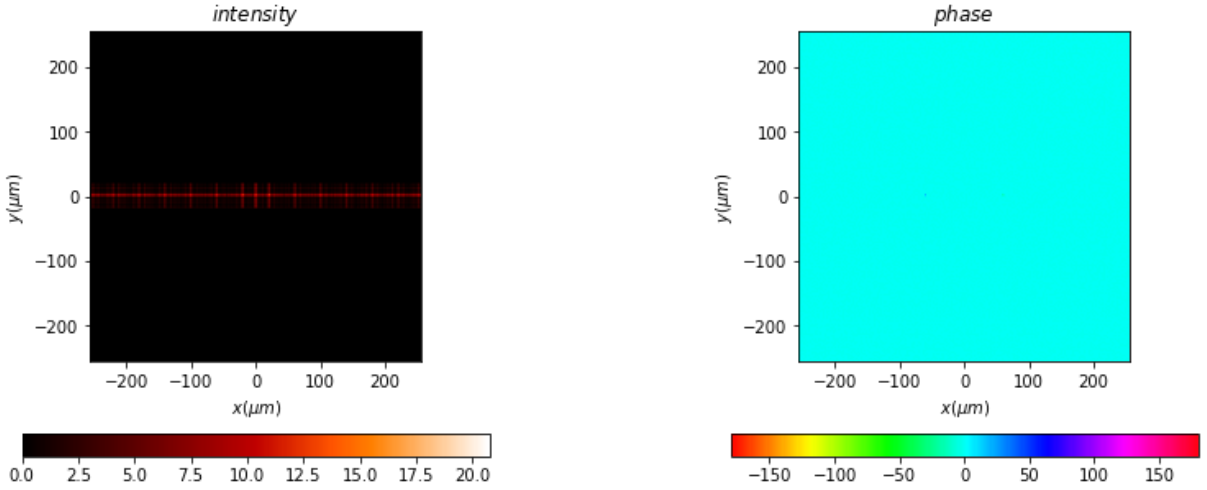
Out[18]:
```
((<matplotlib.image.AxesImage at 0x249de599910>,
  <matplotlib.image.AxesImage at 0x249dbfe10d0>),
 None,
 None)
```

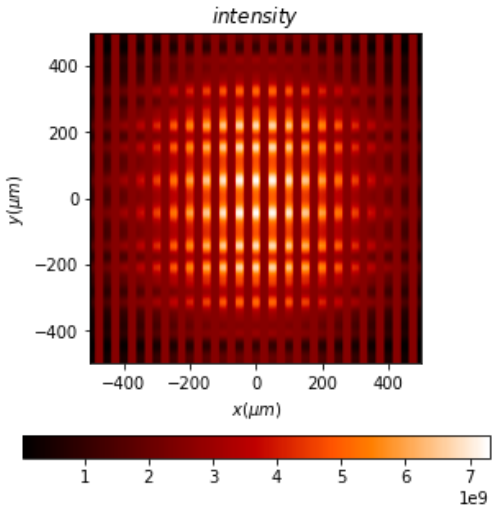


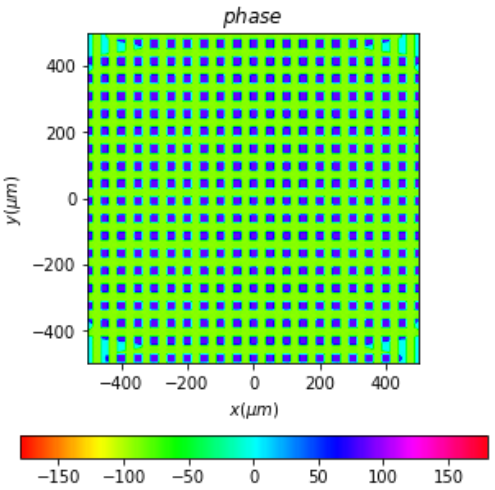
Horizontal Slit

In [19]:
```
a_L1_Horiz, a_L1_Horiz_L2 = sim(a_L1, horiz)
a_L1_Horiz.draw(kind='field', logarithm=True)
a_L1_Horiz_L2.draw(kind='field', logarithm=False)
```

Out[19]:
```
((<matplotlib.image.AxesImage at 0x249df20fd30>,
  <matplotlib.image.AxesImage at 0x249df27bfa0>),
 None,
 None)
```



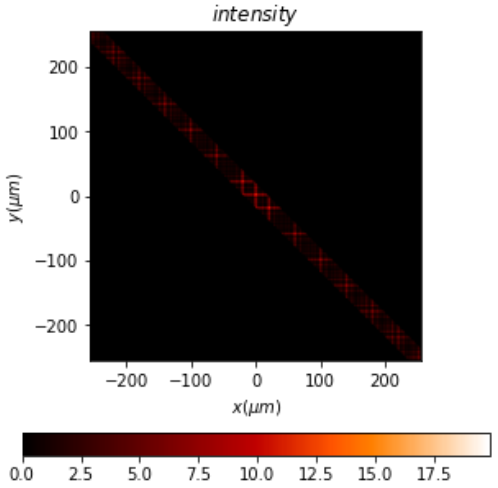


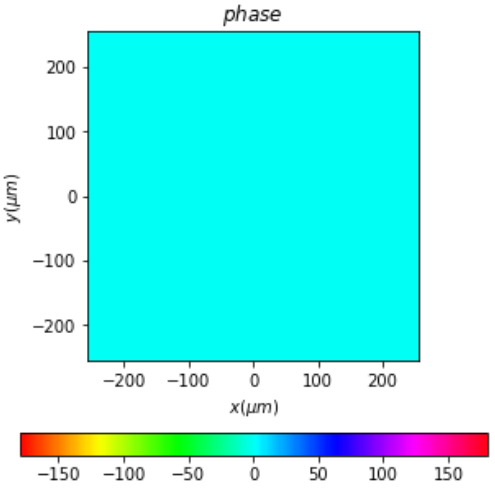


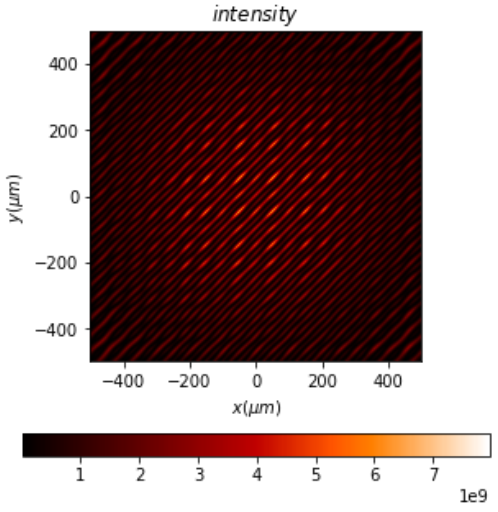
Angled Slit

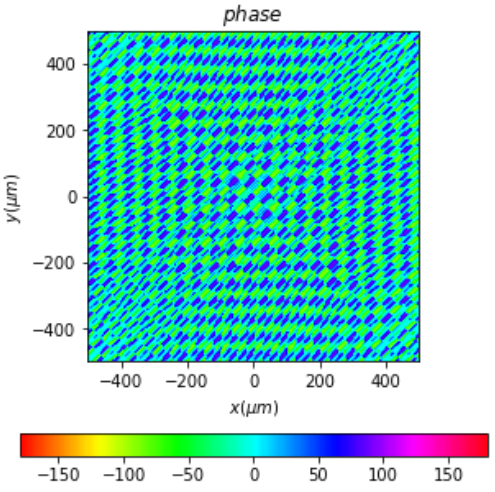
In [20]:
```
a_L1_Angled, a_L1_Angled_L2 = sim(a_L1, angled)
a_L1_Angled.draw(kind='field', logarithm=True)
a_L1_Angled_L2.draw(kind='field', logarithm=False)
```

Out[20]:
```
((<matplotlib.image.AxesImage at 0x249e117e220>,
  <matplotlib.image.AxesImage at 0x249e2a88490>),
 None,
 None)
```





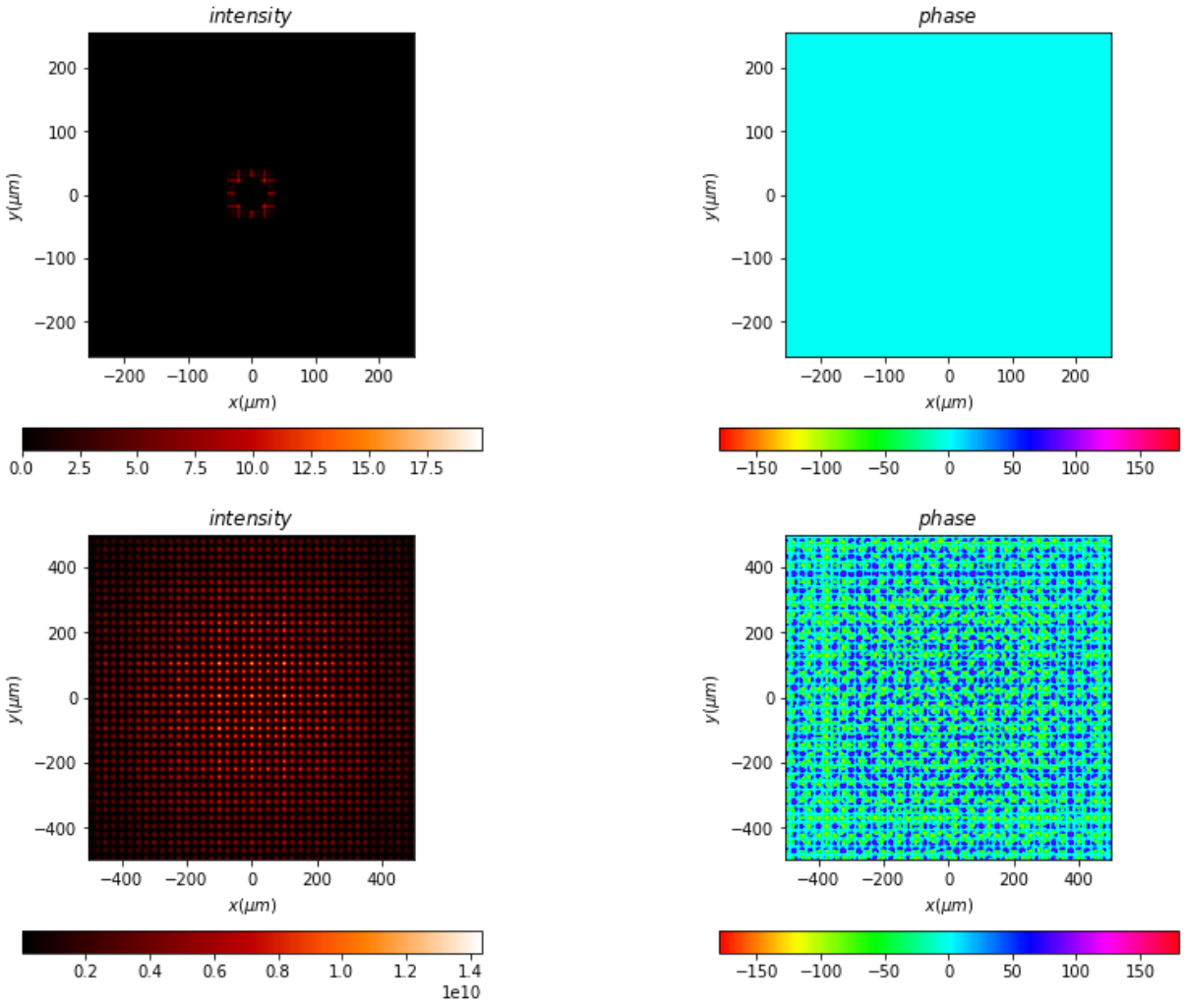




Square + Circle

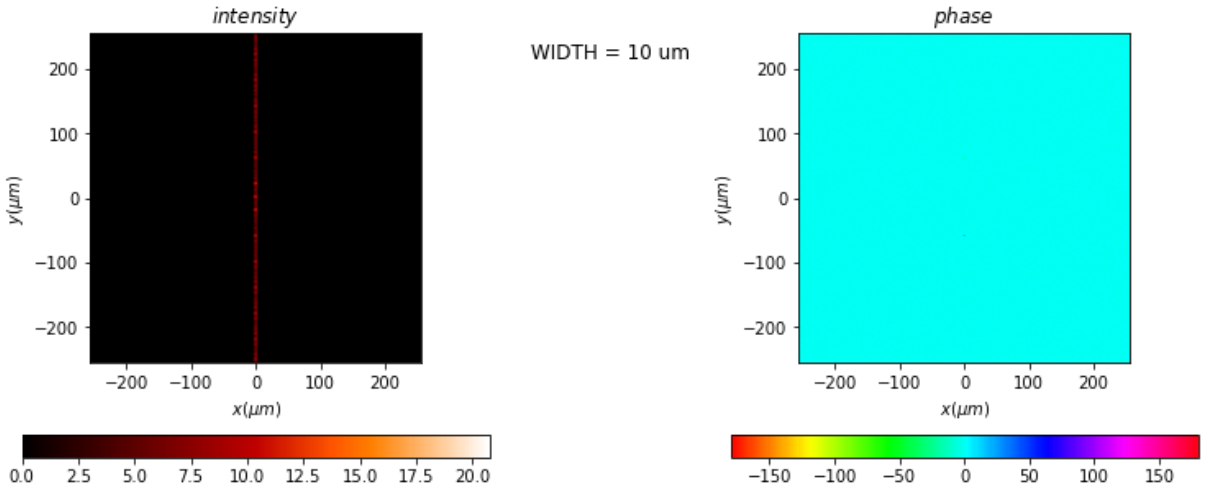
In [21]:
```
a_L1_SqCirc, a_L1_SqCirc_L2 = sim(a_L1, sqCirc)
a_L1_SqCirc.draw(kind='field', logarithm=True)
a_L1_SqCirc_L2.draw(kind='field', logarithm=False)
```

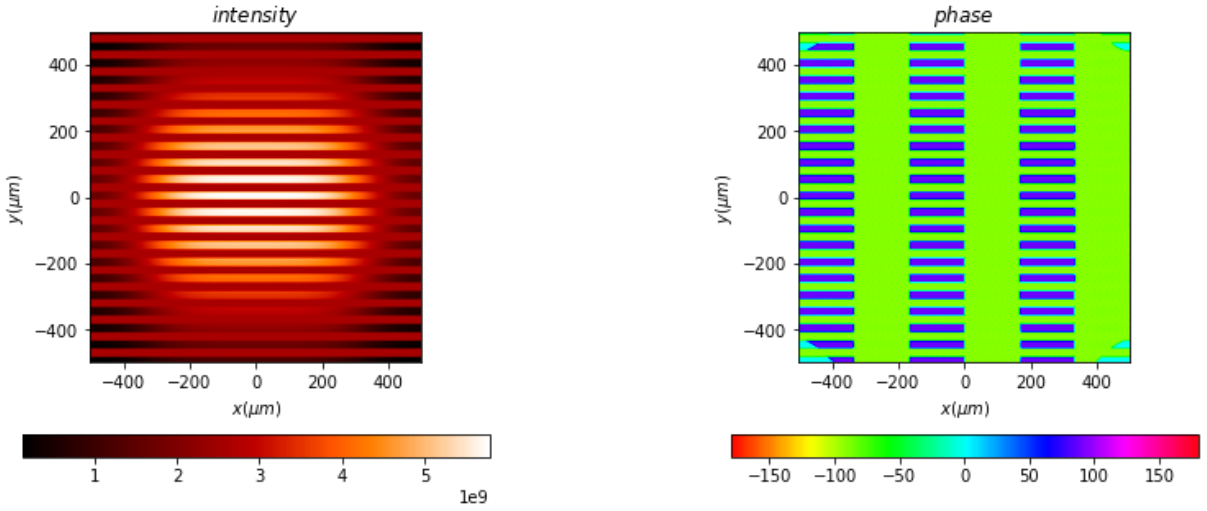
Out[21]:
```
((<matplotlib.image.AxesImage at 0x249dcfbb4f0>,
  <matplotlib.image.AxesImage at 0x249dcfc3580>),
 None,
 None)
```

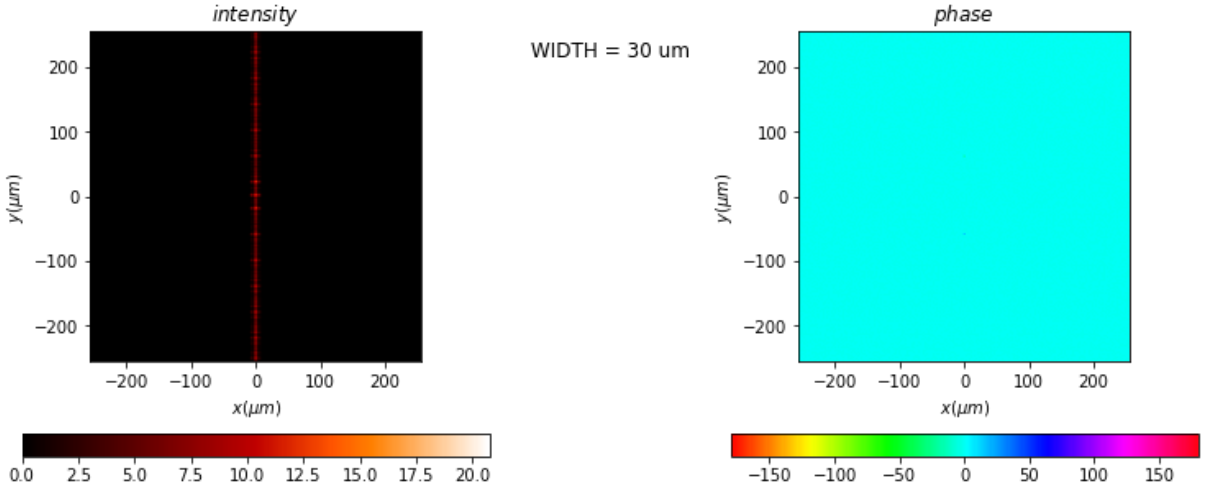


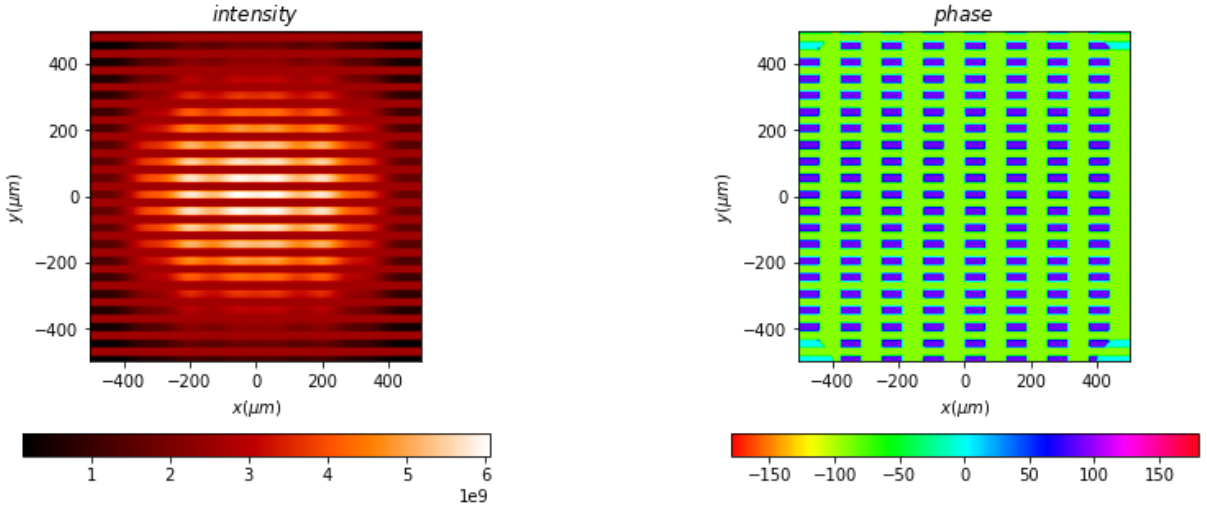
Variable Slit Widths

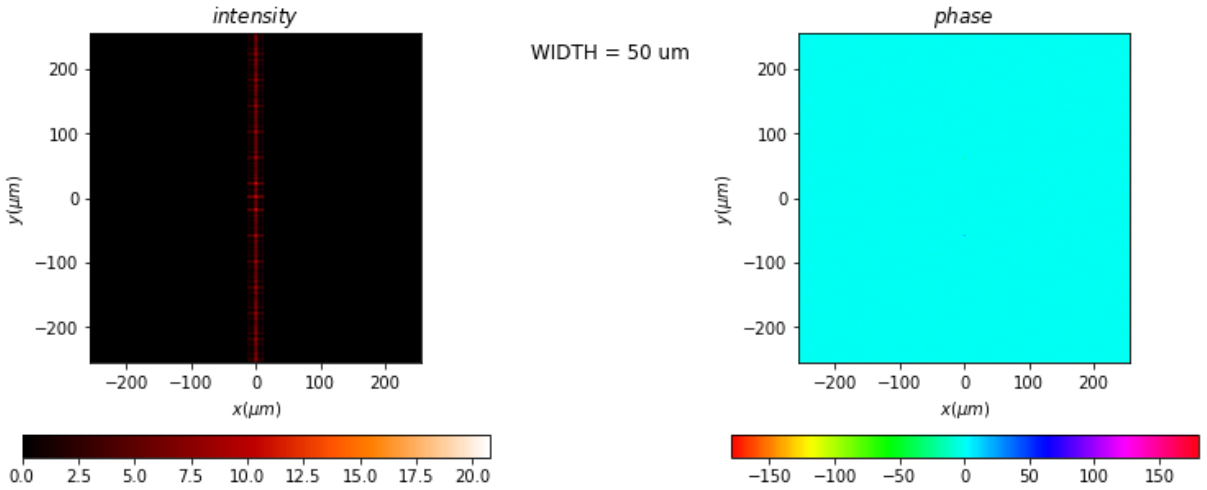
In [22]:
```
widths = [10, 30, 50, 76.3209395, 100, 400]  # in um
for width in widths:
    varSlit = Scalar_mask_XY(x0, y0, wavelength)
    varSlit.slit(
        x0=0 * um,
        size=width * um
    )
#     varSlit.draw(kind='field', logarithm=True)

    a_L1_VarSlit, a_L1_VarSlit_L2 = sim(a_L1, varSlit)
    a_L1_VarSlit.draw(title=f" WIDTH = {width} um ", kind='field', logarithm=True)
    a_L1_VarSlit_L2.draw(kind='field', logarithm=False)
```

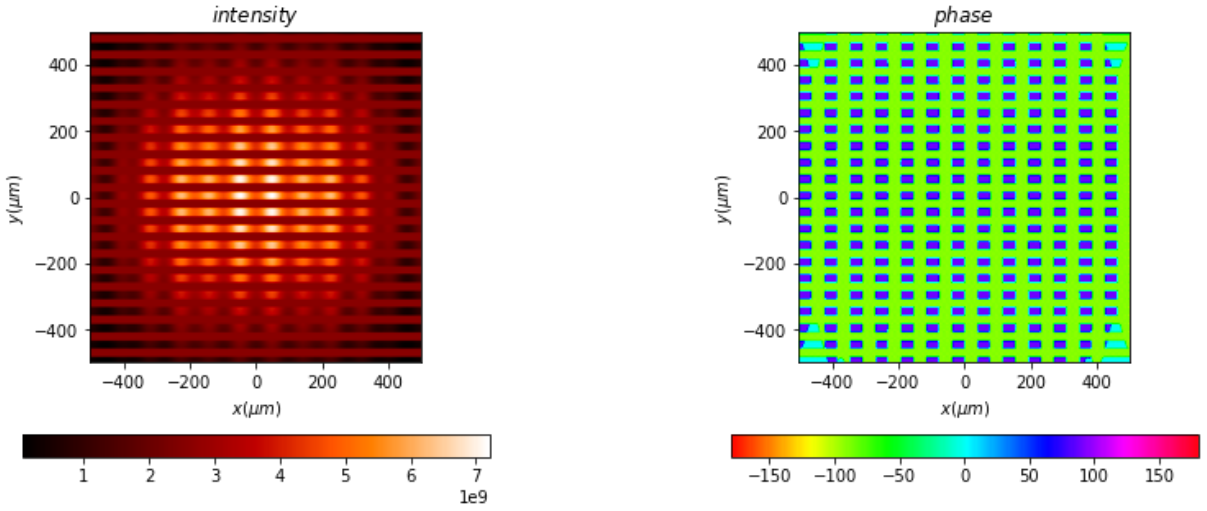

intensity
phase
WIDTH = 10 um
x(μm)
y(μm)
1e9

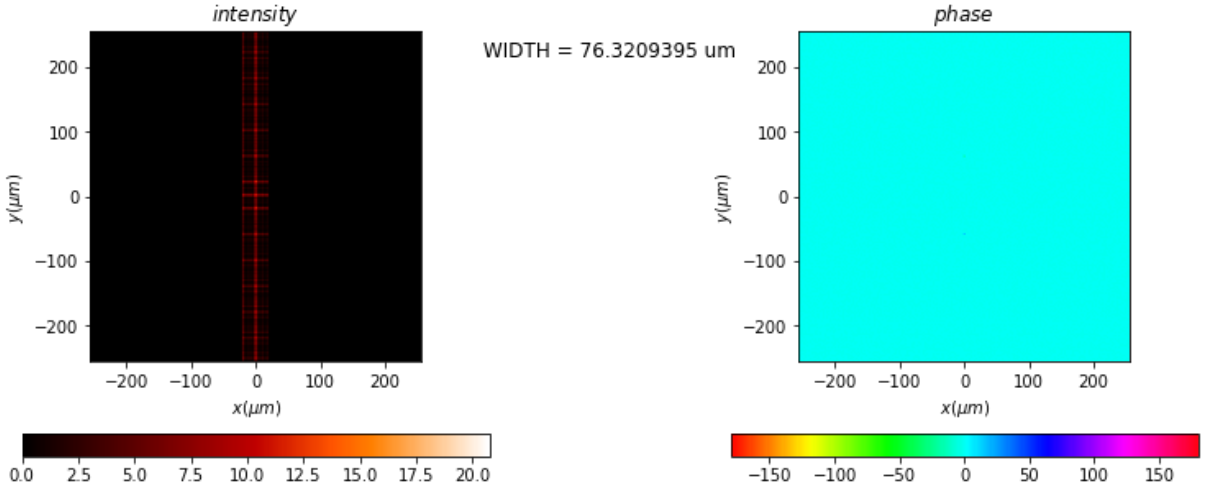

intensity
phase
x(μm)
y(μm)
1e9

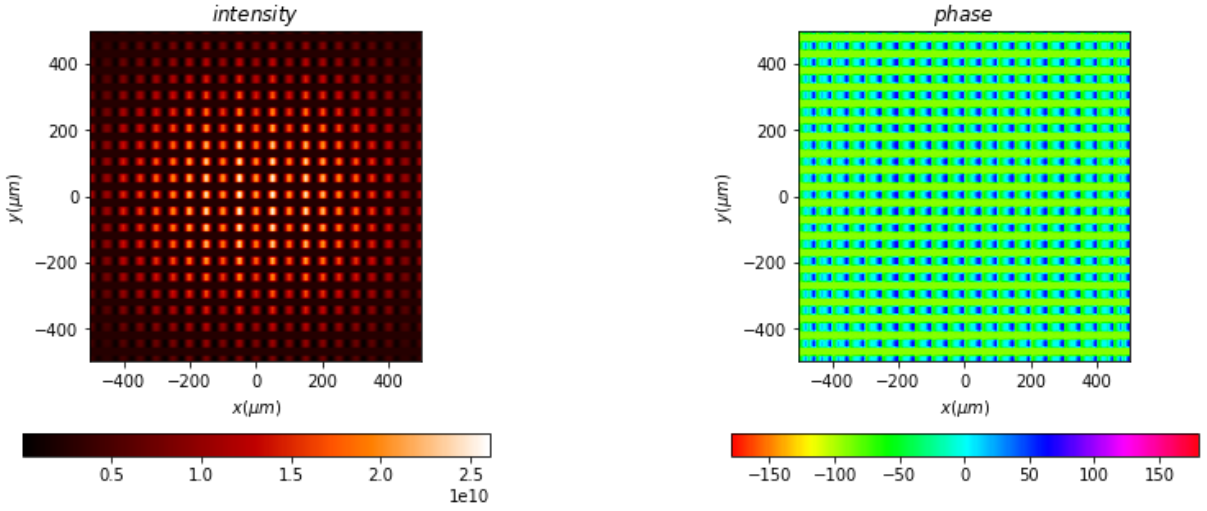

intensity
phase
WIDTH = 30 um
x(μm)
y(μm)
1e9

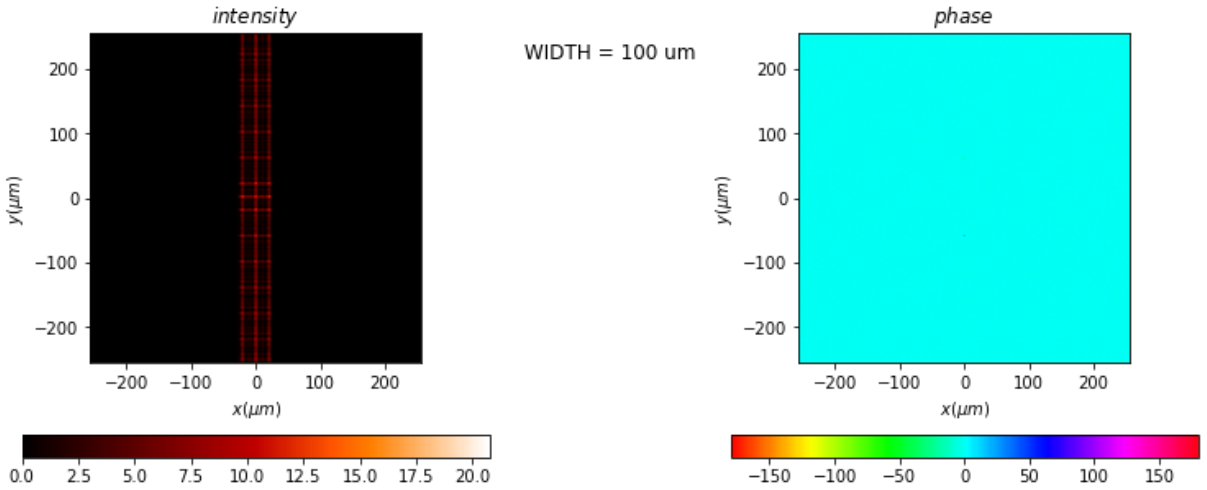

intensity
phase
x(μm)
y(μm)
1e9

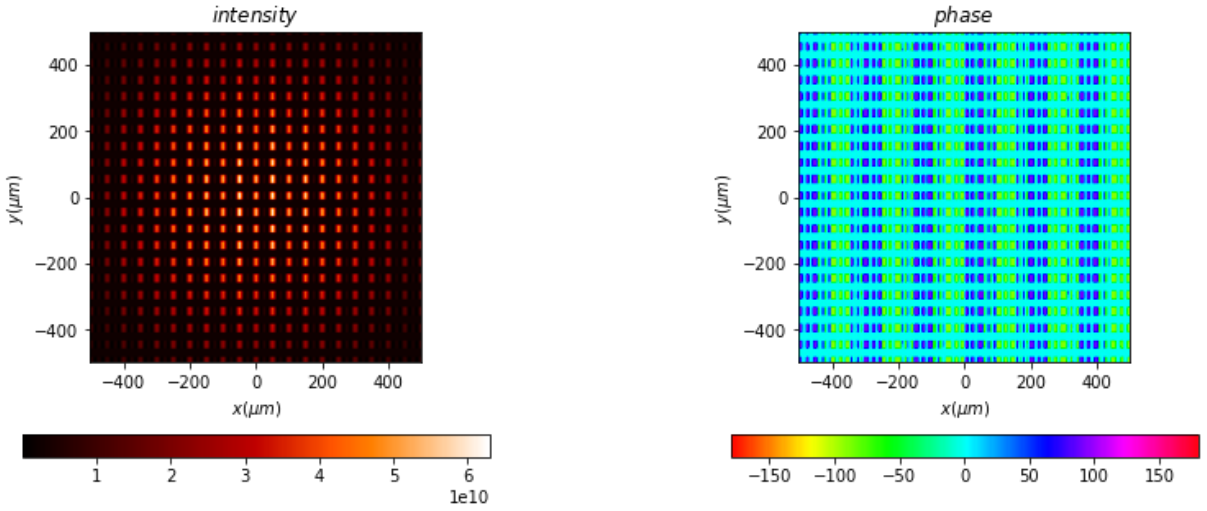

intensity
phase
WIDTH = 50 um
y(μm)
x(μm)

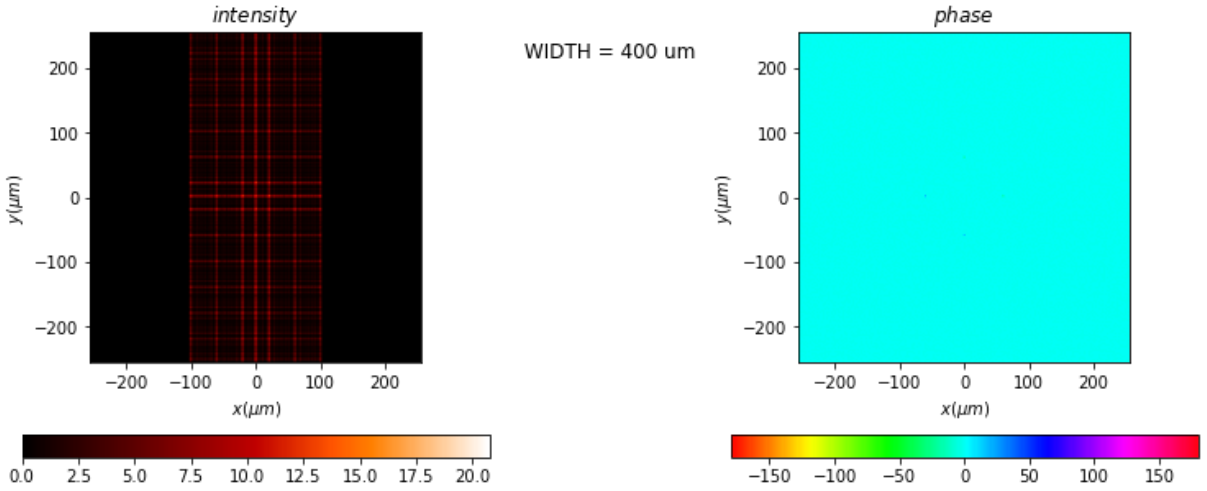

intensity
phase
y(μm)
x(μm)
1e9

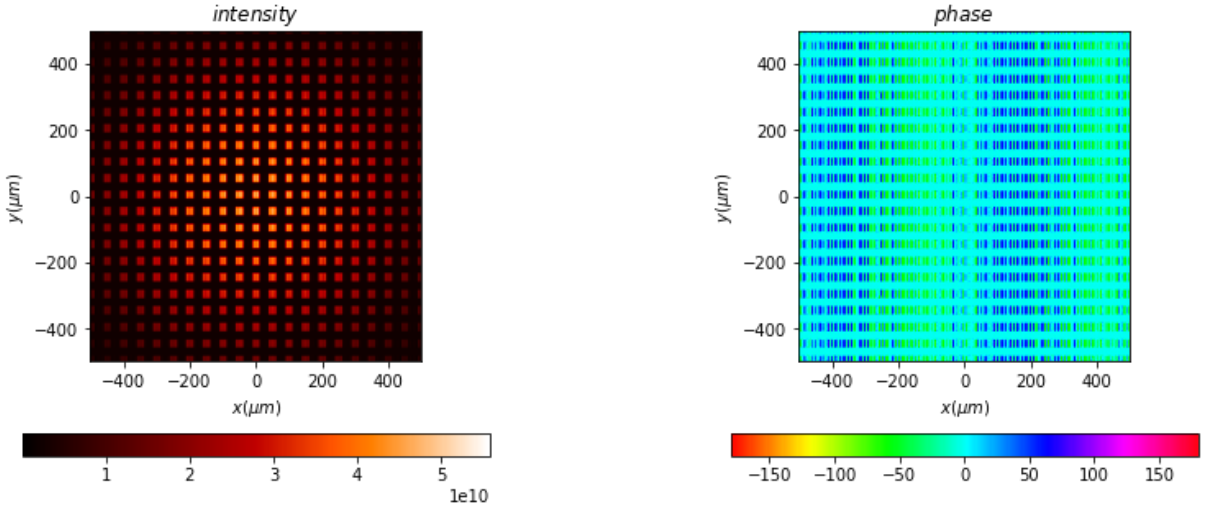

intensity
phase
WIDTH = 76.3209395 um
y(μm)
x(μm)


intensity
phase
y(μm)
x(μm)
1e10


intensity
phase
WIDTH = 100 um
y(μm)
x(μm)
1e10


intensity
phase
y(μm)
x(μm)
1e10


intensity
phase
WIDTH = 400 um
y(μm)
x(μm)
1e10


intensity
phase
y(μm)
x(μm)
1e10

Half-tone Image - Low & High-pass Filtering and Edge Enhancement / Dark-Field Illumination

Compare results with figures in Eisenkraft (1977)

In [1]:
```
import numpy as np
from diffractio import mm, um, degrees
from diffractio.scalar_sources_XY import Scalar_source_XY
from diffractio.scalar_masks_XY import Scalar_mask_XY


# Setting up
length = 1 * mm
num_data = 512
x0 = np.linspace(-length / 2, length / 2, num_data)
y0 = np.linspace(-length / 2, length / 2, num_data)
wavelength = 0.633 * um
```

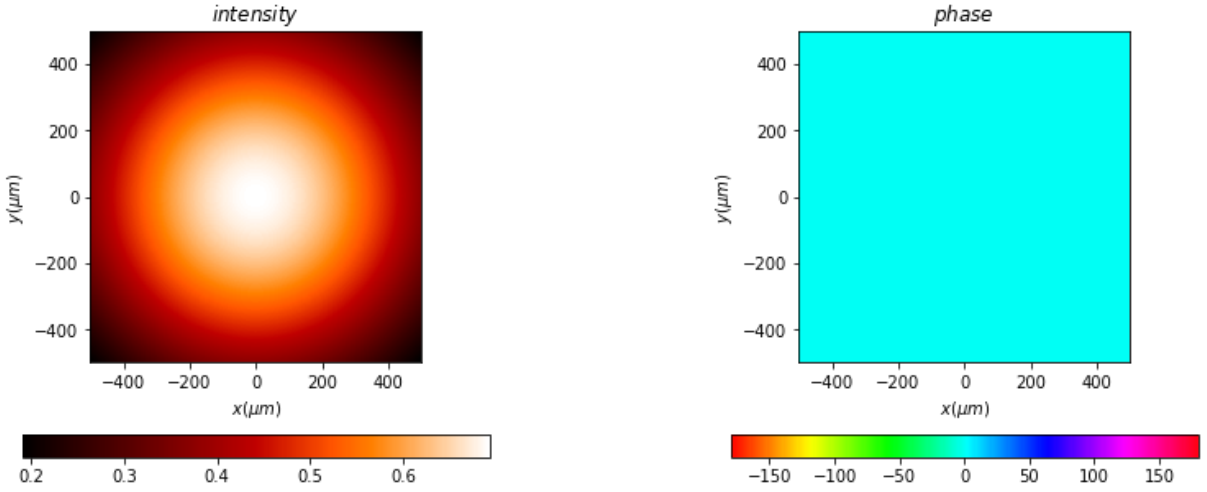
```
number of processors: 12
```

Setting up source(s)

- Gaussian Beam (LASER)
- Experiments (Either only requires modification in filter shapes)
  - Plane Wave
  - Spherical Wave

In [2]:
```
# Gaussian Beam Source - Like a LASER
u0 = Scalar_source_XY(x=x0, y=y0, wavelength=wavelength)

u0.gauss_beam(r0=(0, 0), w0=(800 * um, 800 * um), z0=0.0)
u0.draw(kind='field', logarithm=True)
```

Out[2]:
```
((<matplotlib.image.AxesImage at 0x2c0a2a9daf0>,
  <matplotlib.image.AxesImage at 0x2c0a2d3d2b0>),
 None,
 None)
```



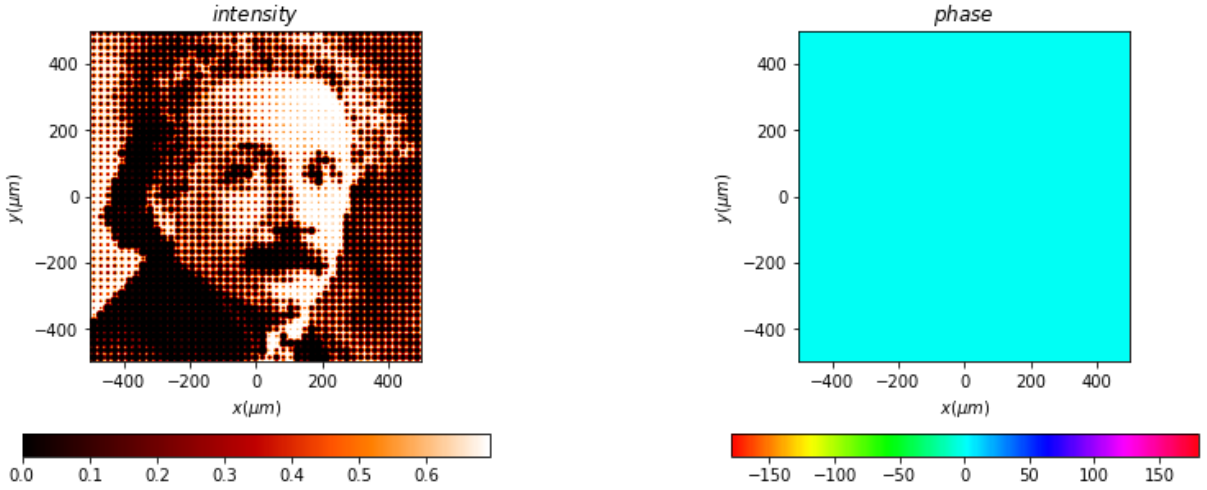
In [3]:
```
# # Plane Wave Source
# u1 = Scalar_source_XY(x=x0, y=y0, wavelength=wavelength)
# u1.plane_wave()
# u1.draw(kind='field', logarithm=True)
```

In [4]:
```
# # Spherical Wave Source
# u2 = Scalar_source_XY(x=x0, y=y0, wavelength=wavelength)
# u2.spherical_wave(z0=1, mask=False)
# u2.draw(kind='field', logarithm=True)
```

Image

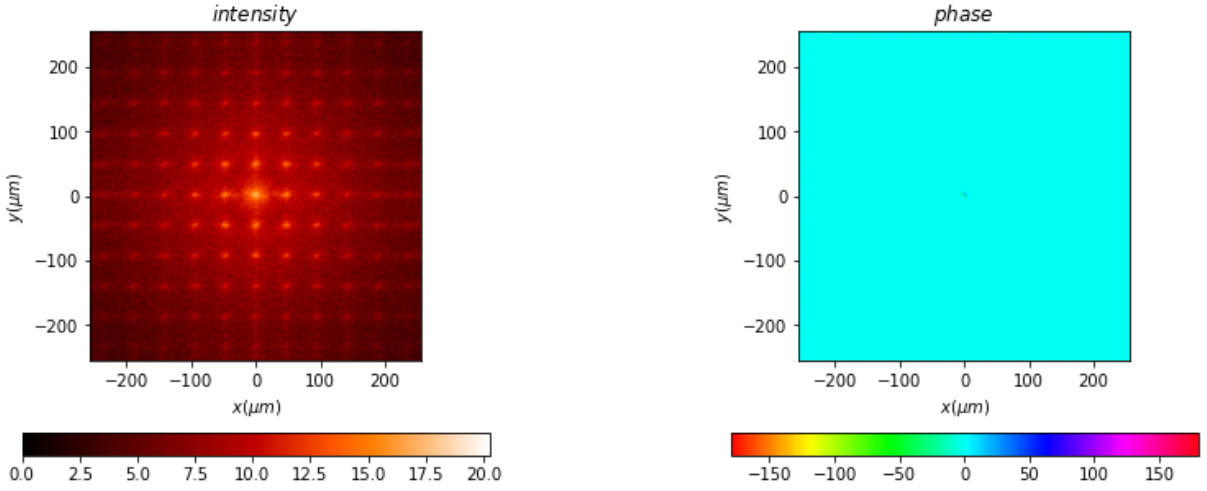
In [5]:
```
y15 = Scalar_mask_XY(x0, y0, wavelength)
y15.image(
    filename="y-15-HT.png",
    normalize=True,
    canal=2,
)

y15.draw(kind='field', logarithm=True)
```

Out[5]:
```
((<matplotlib.image.AxesImage at 0x2c0a313e520>,
  <matplotlib.image.AxesImage at 0x2c0a37bfca0>),
 None,
 None)
```



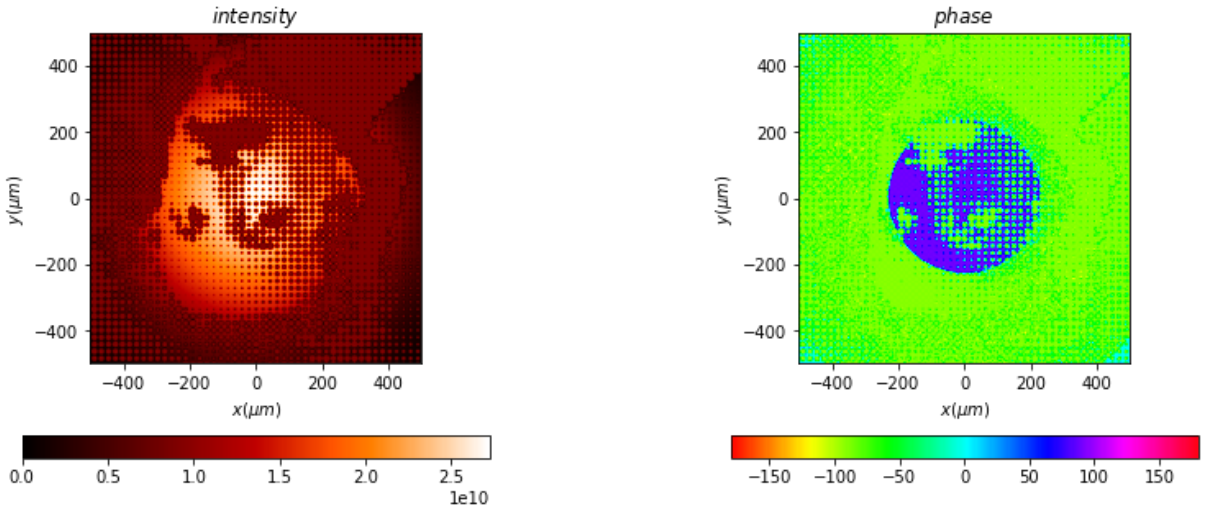
In [6]:
```
# Fourier Plane - No Filter
a_L1 = (u0 * y15).fft(z=1 * mm, new_field=True)
a_L1.draw(kind='field', logarithm=True)
```

Out[6]:
```
((<matplotlib.image.AxesImage at 0x2c0a384fc40>,
  <matplotlib.image.AxesImage at 0x2c0a3922430>),
 None,
 None)
```



In [7]:
```
# This is how, it'd look without any filter, at the Observation screen
a_L2 = a_L1.fft(z=1 * mm, shift=False, remove0=False, new_field=True)
a_L2.draw(kind='field', logarithm=False)
```

Out[7]:
```
((<matplotlib.image.AxesImage at 0x2c0a3e44310>,
  <matplotlib.image.AxesImage at 0x2c0a3e93ac0>),
 None,
 None)
```



Masks / Filters

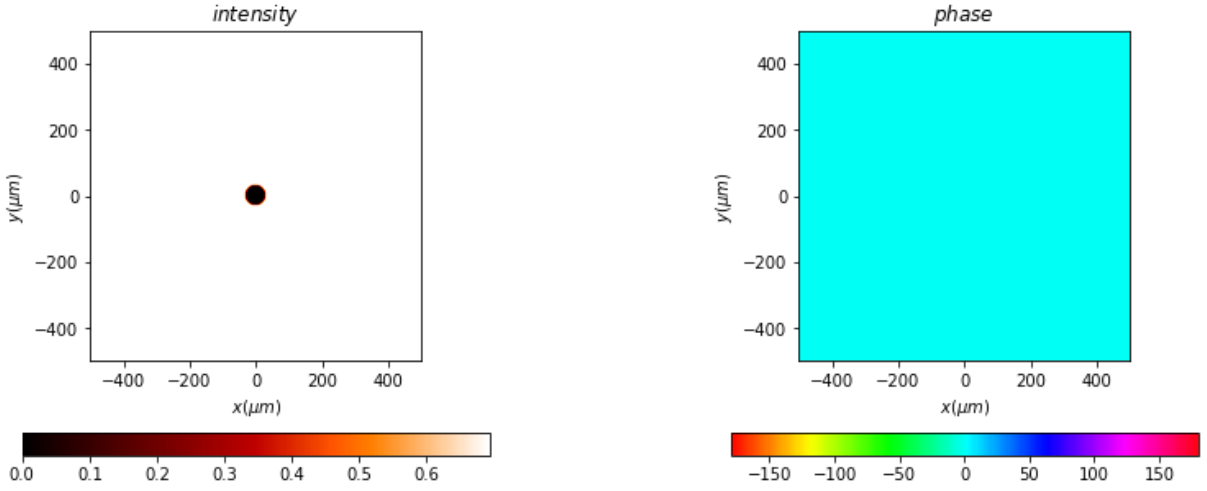
- Center Dot - High Pass
- Square - Low Pass
- Square + Center Dot -

- Vertical Slit

- Horizontal Slit
- Angled Slit

In [8]:
```
cDot = Scalar_mask_XY(x0, y0, wavelength)
cDot.ring(
    r0=(0 * um, 0 * um),
    radius1=(30 * um, 30 * um),
    radius2=(1000 * um, 1000 * um)
)
cDot.draw(kind='field', logarithm=True)
```

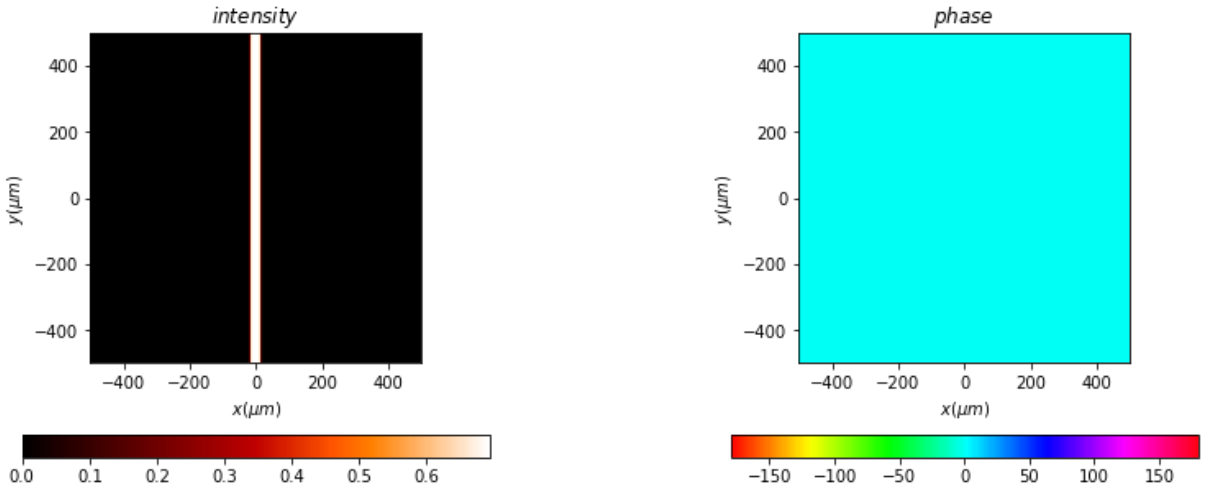
Out[8]:
```
((<matplotlib.image.AxesImage at 0x2c0a7d6dc40>,
  <matplotlib.image.AxesImage at 0x2c0a7e04430>),
 None,
 None)
```



In [9]:
```
vert = Scalar_mask_XY(x0, y0, wavelength)
vert.slit(
    x0=0 * um,
    size=30 * um
)
vert.draw(kind='field', logarithm=True)
```

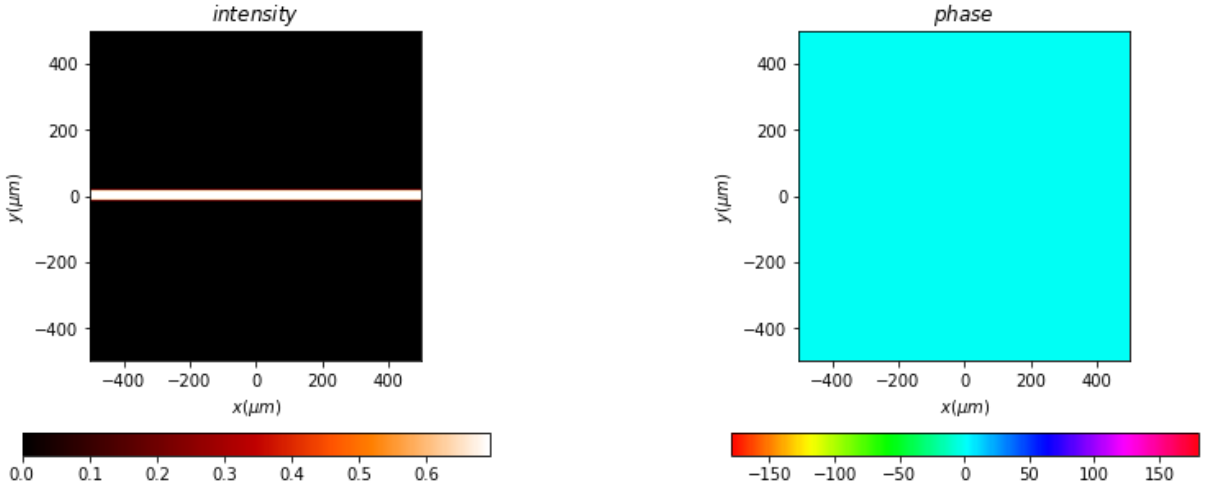
Out[9]:
```
((<matplotlib.image.AxesImage at 0x2c0a7d2fb80>,
```

```
 <matplotlib.image.AxesImage at 0x2c0a30a7970>),
None,
None)
```



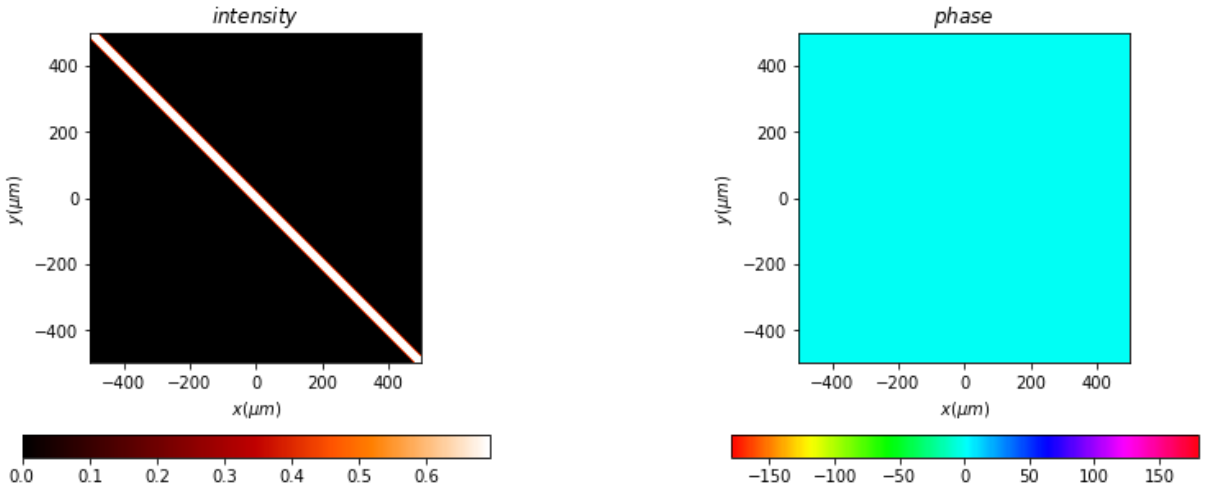
In [10]:
```
horiz = Scalar_mask_XY(x0, y0, wavelength)
horiz.slit(
    x0=0 * um,
    size=30 * um,
    angle=np.pi / 2
)
horiz.draw(kind='field', logarithm=True)
```

Out[10]:
```
((<matplotlib.image.AxesImage at 0x2c0a8de2730>,
  <matplotlib.image.AxesImage at 0x2c0a8e6ceb0>),
None,
None)
```



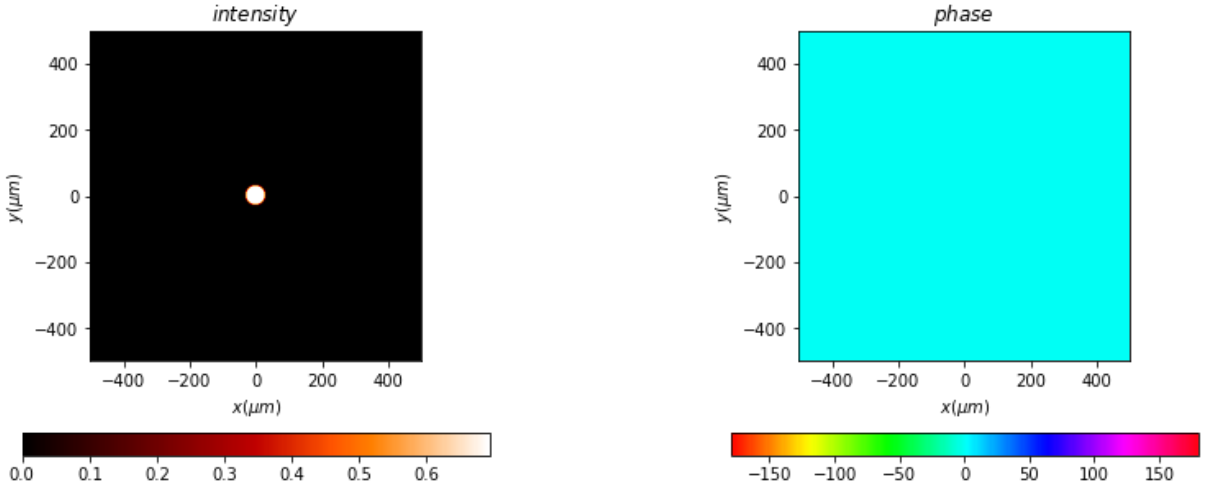
In [11]:
```
angled = Scalar_mask_XY(x0, y0, wavelength)
angled.slit(
    x0=0 * um,
    size=30 * um,
    angle=np.pi / 4
)
angled.draw(kind='field', logarithm=True)
```

Out[11]:
```
((<matplotlib.image.AxesImage at 0x2c0aa1de160>,
  <matplotlib.image.AxesImage at 0x2c0aa22e940>),
None,
None)
```



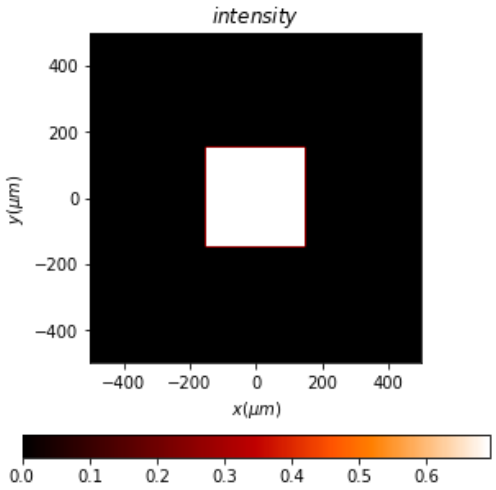
In [12]:
```
circ = Scalar_mask_XY(x0, y0, wavelength)
circ.circle(
    r0=(0 * um, 0 * um),
    radius=(30 * um, 30 * um),
    angle=0
)
circ.draw(kind='field', logarithm=True)
```

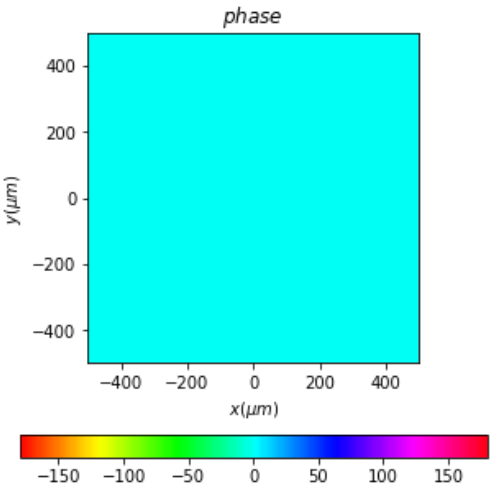
Out[12]:
```
((<matplotlib.image.AxesImage at 0x2c0aa304d00>,
  <matplotlib.image.AxesImage at 0x2c0aa372880>),
 None,
 None)
```



In [13]:
```
sq = Scalar_mask_XY(x0, y0, wavelength)
sq.square(
    r0=(0 * um, 0 * um),
    size=(300 * um, 300 * um),
    angle=0,
)
sq.draw(kind='field', logarithm=True)
```

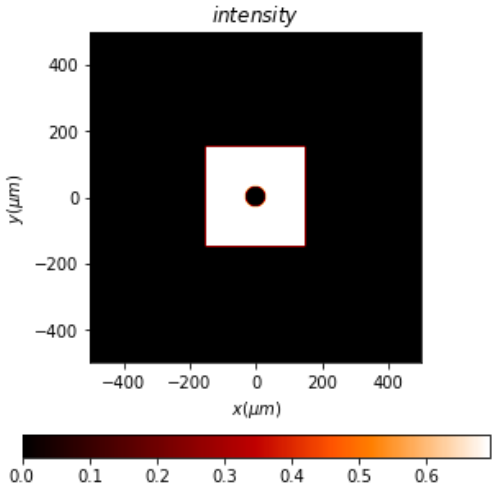
Out[13]:
```
((<matplotlib.image.AxesImage at 0x2c0aae98c10>,
  <matplotlib.image.AxesImage at 0x2c0aaefde80>),
 None,
 None)
```

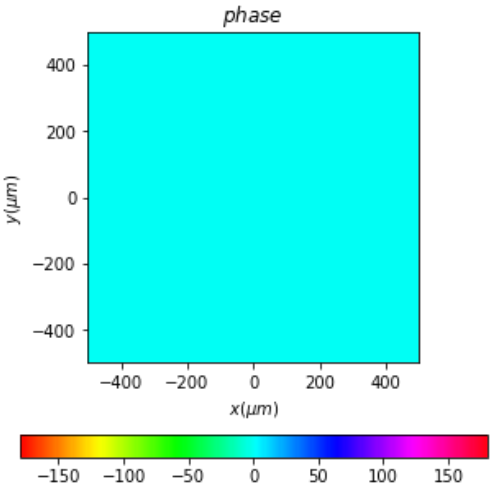




In [14]:
```
# Adding two filters
sqCirc = sq * cDot
sqCirc.draw(kind="field", logarithm=True)
```

Out[14]:
```
((<matplotlib.image.AxesImage at 0x2c0aae94a60>,
  <matplotlib.image.AxesImage at 0x2c0a8dc96a0>),
 None,
 None)
```





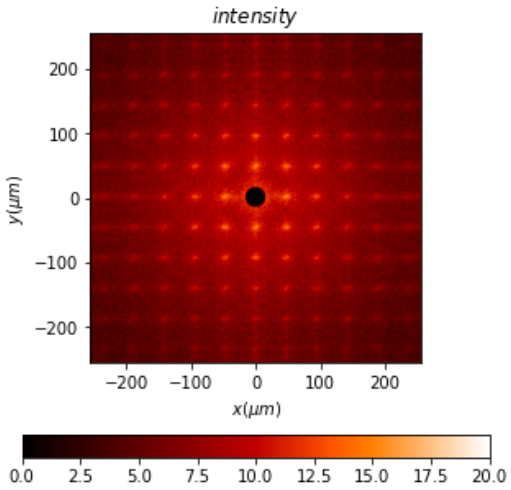
Returns wave after encountering filter and then L2

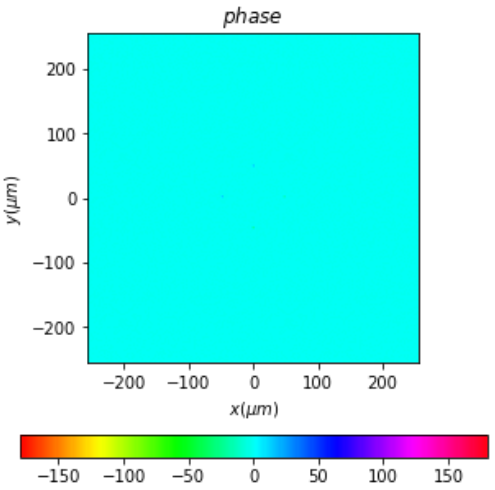
In [15]:
```
def sim(a_L1, mask):
    """
    a_L1: Wave after passing through Lens 1
    mask: Mask or Filter to apply

    """
    a_L1_Mask = a_L1 * mask
    a_L1_Mask_L2 = a_L1_Mask.fft(z=1 * mm, shift=False, remove0=False, new_field=True)

    return a_L1_Mask, a_L1_Mask_L2
```

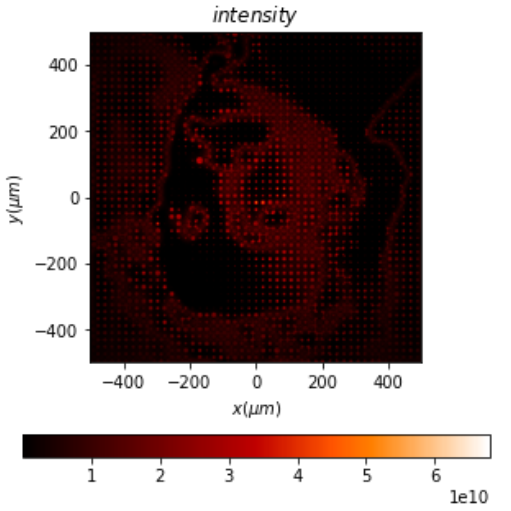
Center Dot - High Pass - HT Structure Visible - EDGE DETECTION

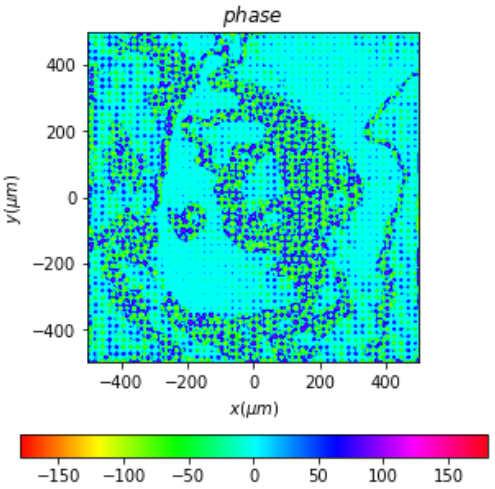
In [16]:
```
a_L1_cD, a_L1_cD_L2 = sim(a_L1, cDot)
a_L1_cD.draw(kind='field', logarithm=True)
a_L1_cD_L2.draw(kind='field', logarithm=False)
```

Out[16]:
```
((<matplotlib.image.AxesImage at 0x2c0ac627f10>,
  <matplotlib.image.AxesImage at 0x2c0ac69d700>),
 None,
 None)
```





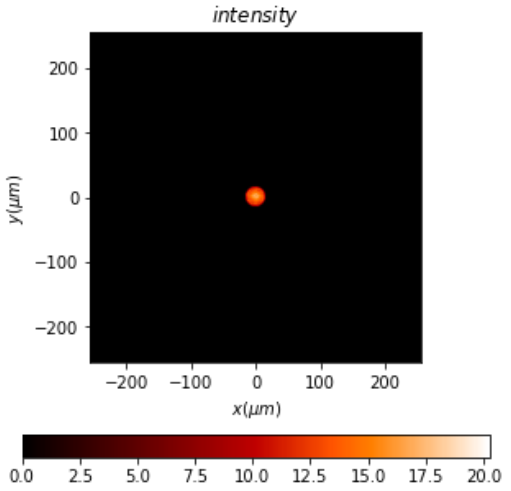


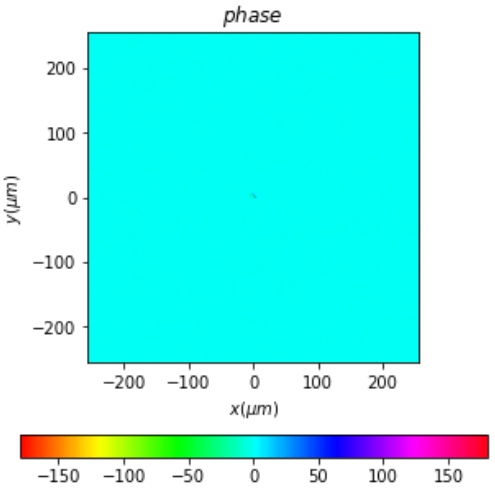


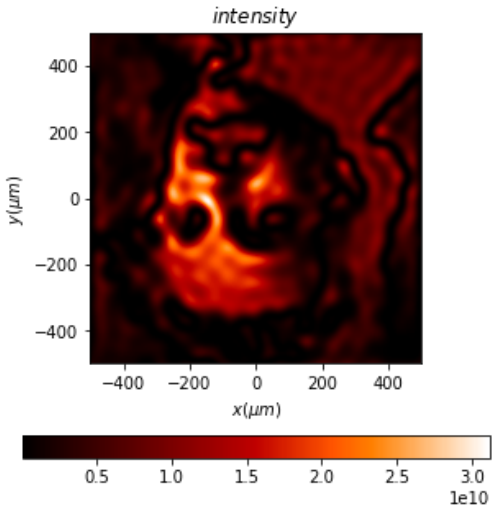
Circle - Low Pass - Blurred Output

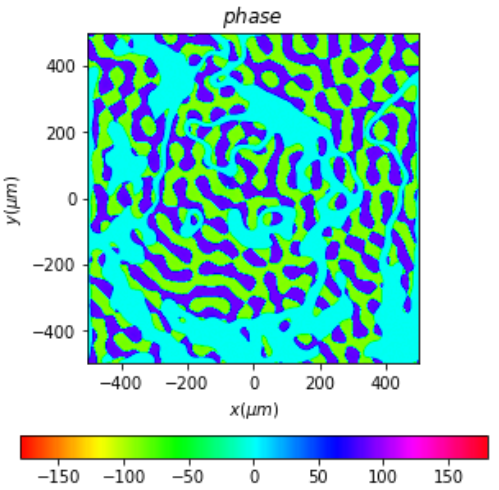
In [17]:
```
a_L1_Circ, a_L1_Circ_L2 = sim(a_L1, circ)
a_L1_Circ.draw(kind='field', logarithm=True)
a_L1_Circ_L2.draw(kind='field', logarithm=False)
```

Out[17]:
```
((<matplotlib.image.AxesImage at 0x2c0ae2eedf0>,
  <matplotlib.image.AxesImage at 0x2c0b1ef8130>),
 None,
 None)
```





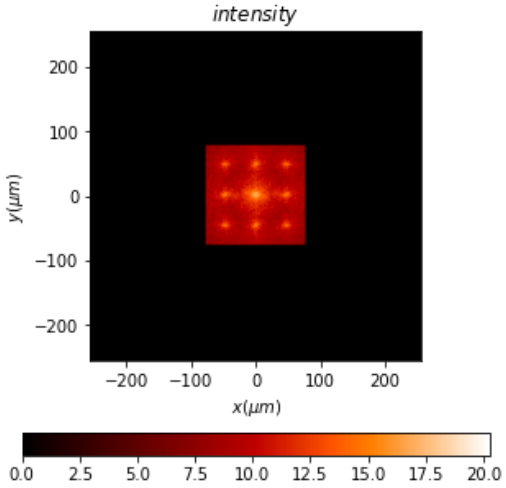


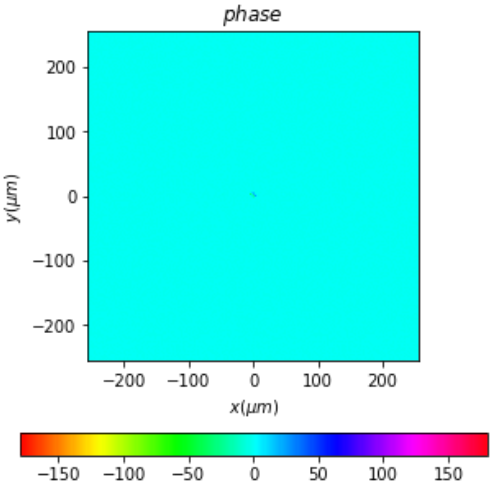


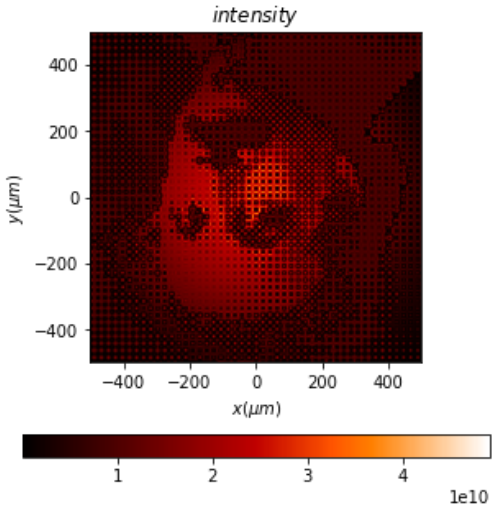
Square

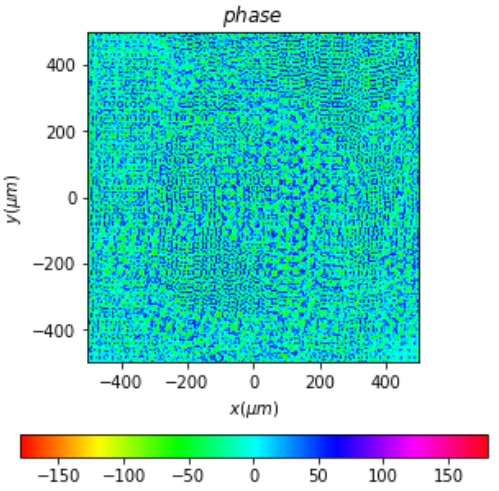
In [18]:
```
a_L1_Sq, a_L1_Sq_L2 = sim(a_L1, sq)
a_L1_Sq.draw(kind='field', logarithm=True)
a_L1_Sq_L2.draw(kind='field', logarithm=False)
```

Out[18]:
```
((<matplotlib.image.AxesImage at 0x2c0b1c24a90>,
  <matplotlib.image.AxesImage at 0x2c0b1a80fd0>),
 None,
 None)
```





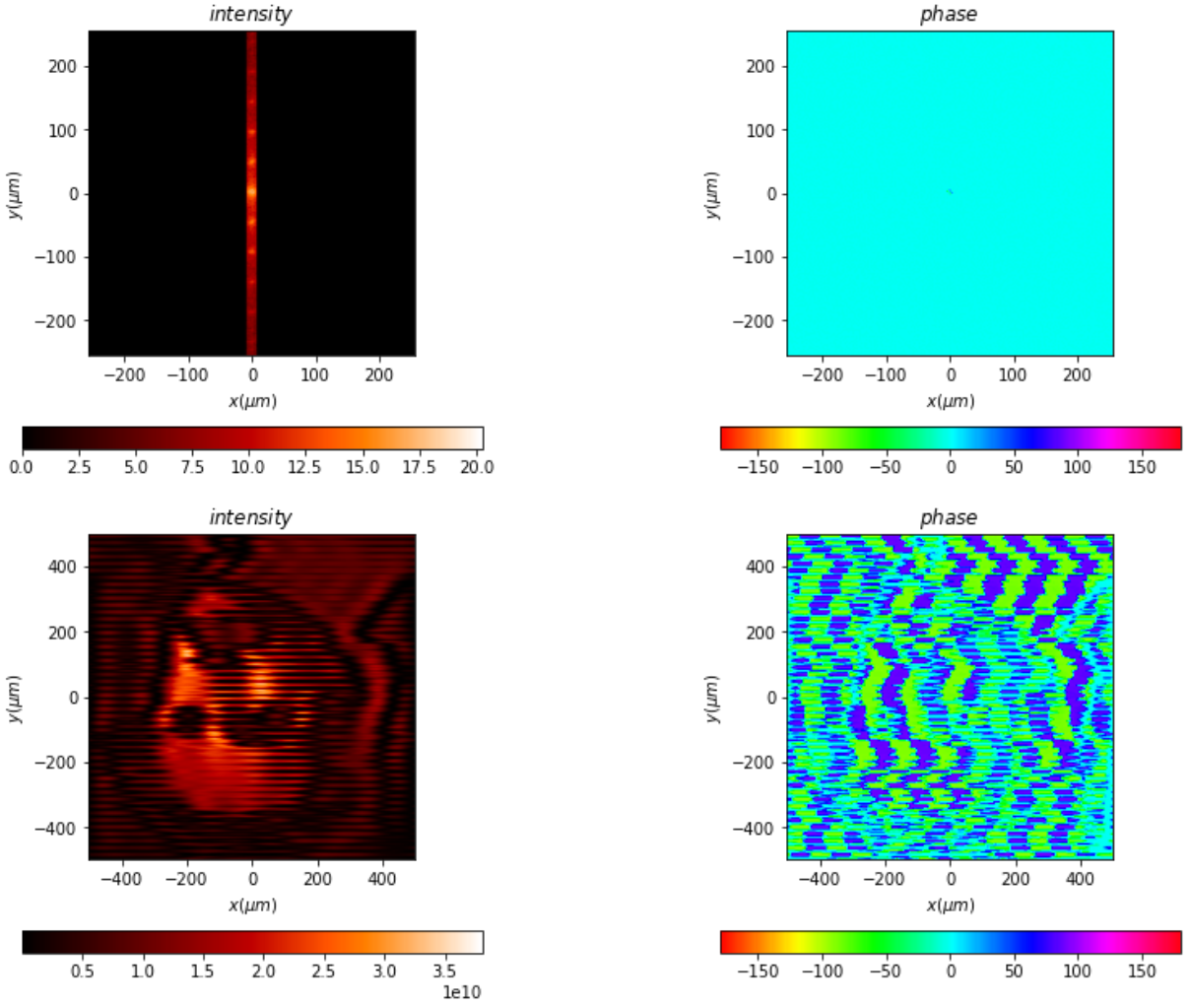




Vertical Slit

In [19]:
```
a_L1_Vert, a_L1_Vert_L2 = sim(a_L1, vert)
a_L1_Vert.draw(kind='field', logarithm=True)
a_L1_Vert_L2.draw(kind='field', logarithm=False)
```

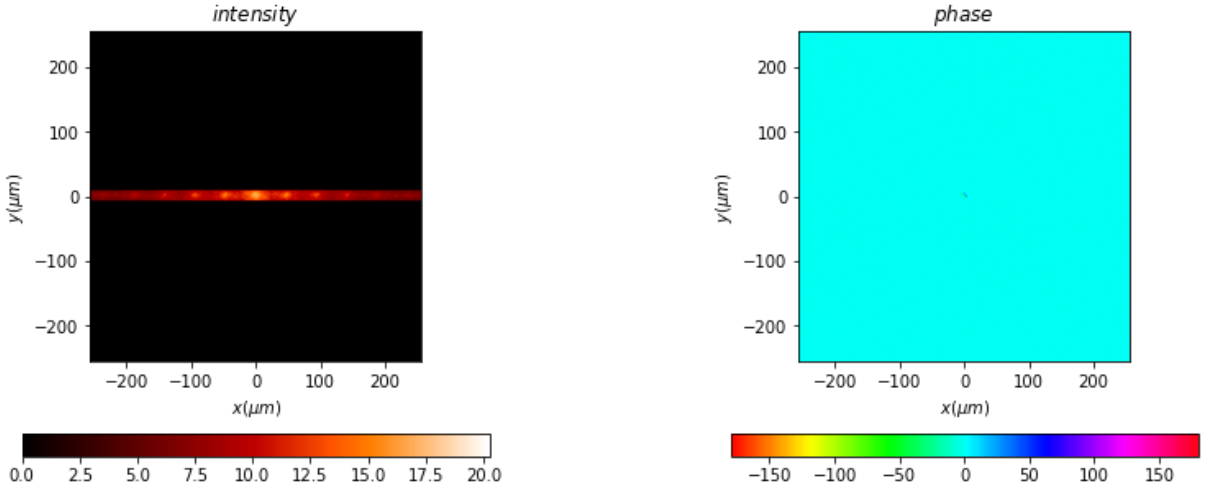
Out[19]:
```
((<matplotlib.image.AxesImage at 0x2c0ac560be0>,
  <matplotlib.image.AxesImage at 0x2c0ae248880>),
 None,
 None)
```

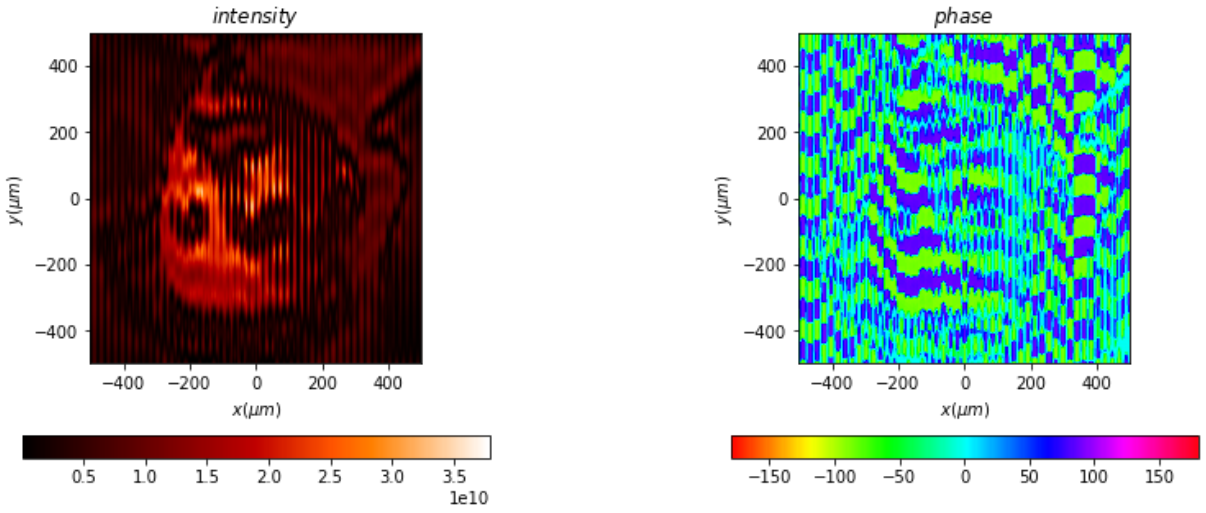


Horizontal Slit

In [20]:
```
a_L1_Horiz, a_L1_Horiz_L2 = sim(a_L1, horiz)
a_L1_Horiz.draw(kind='field', logarithm=True)
a_L1_Horiz_L2.draw(kind='field', logarithm=False)
```

Out[20]:
```
((<matplotlib.image.AxesImage at 0x2c0b48461f0>,
  <matplotlib.image.AxesImage at 0x2c0b48af460>),
 None,
 None)
```

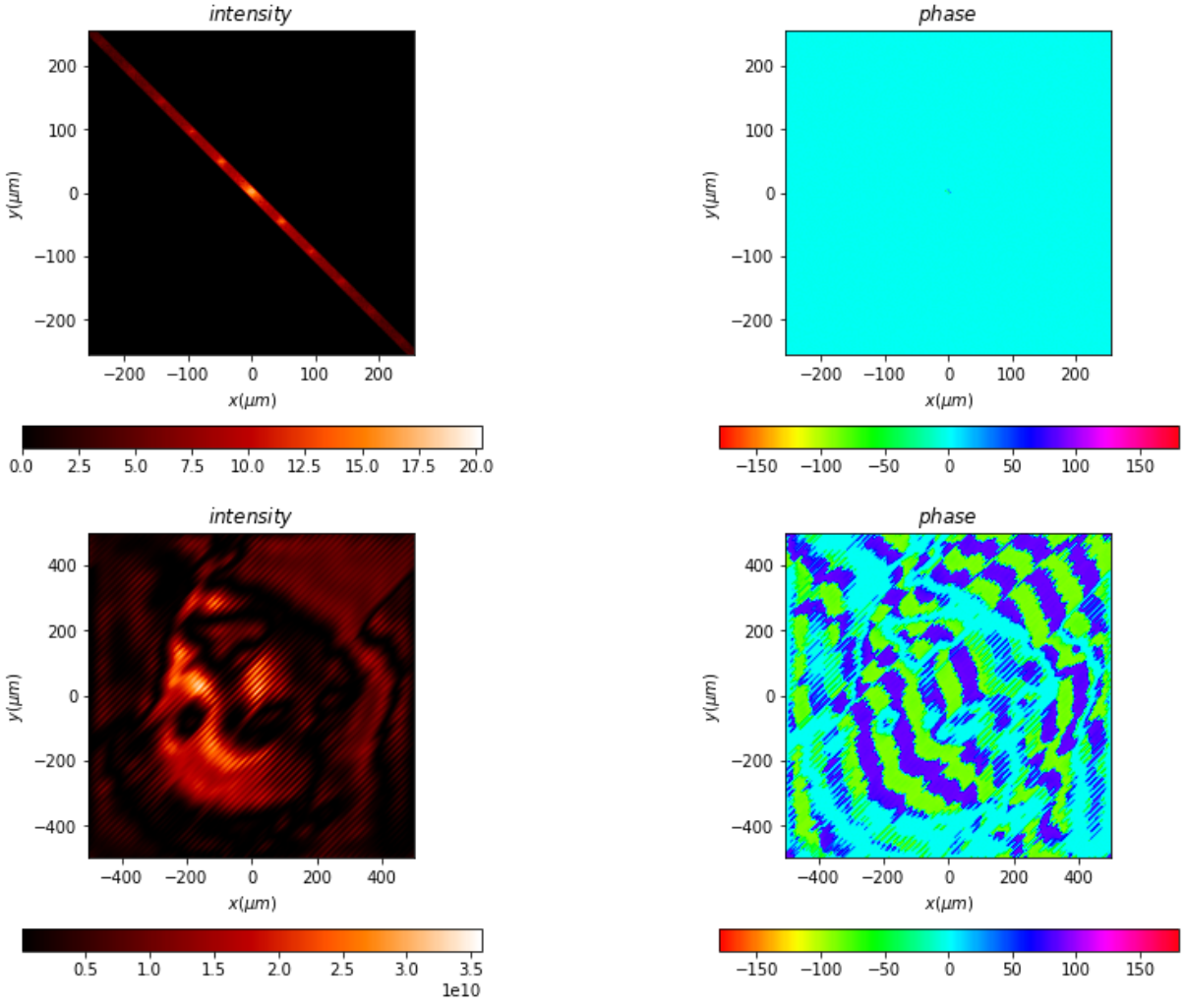




Angled Slit

In [21]:
```
a_L1_Angled, a_L1_Angled_L2 = sim(a_L1, angled)
a_L1_Angled.draw(kind='field', logarithm=True)
a_L1_Angled_L2.draw(kind='field', logarithm=False)
```

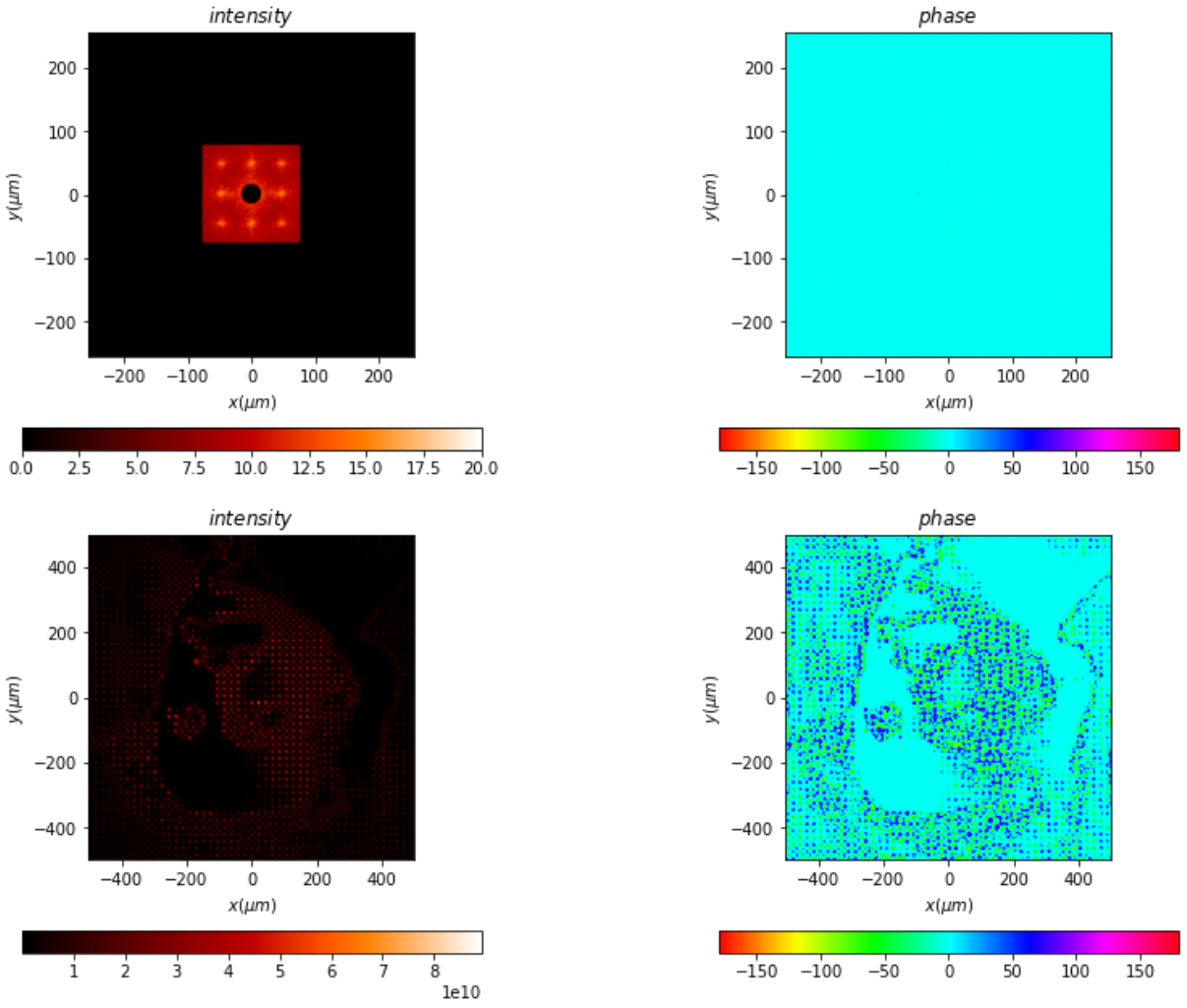
Out[21]:
```
((<matplotlib.image.AxesImage at 0x2c0b4f0e6a0>,
  <matplotlib.image.AxesImage at 0x2c0b7a1a910>),
 None,
 None)
```



Square + Circle - Dark-Field Illumination / Edge Enhancement

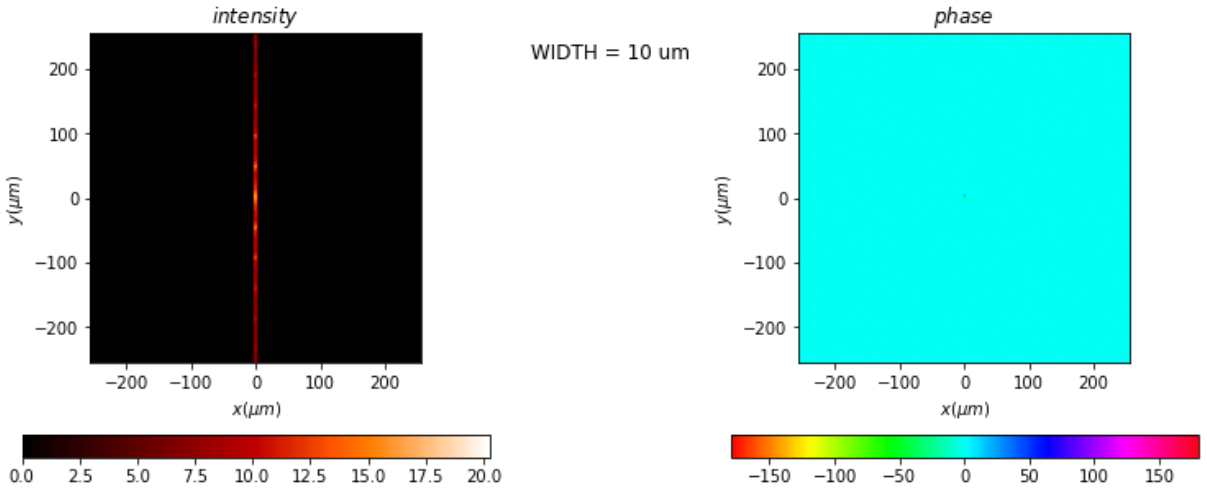
In [22]:
```
a_L1_SqCirc, a_L1_SqCirc_L2 = sim(a_L1, sqCirc)
a_L1_SqCirc.draw(kind='field', logarithm=True)
a_L1_SqCirc_L2.draw(kind='field', logarithm=False)
```

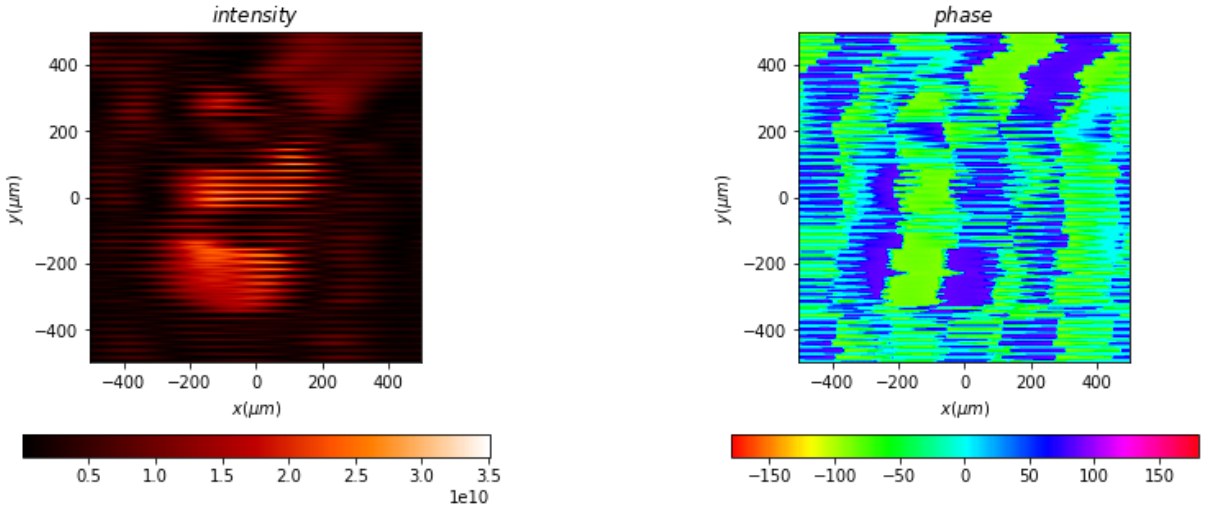
Out[22]:
```
((<matplotlib.image.AxesImage at 0x2c0ac53f4f0>,
  <matplotlib.image.AxesImage at 0x2c0b1a87fa0>),
 None,
 None)
```

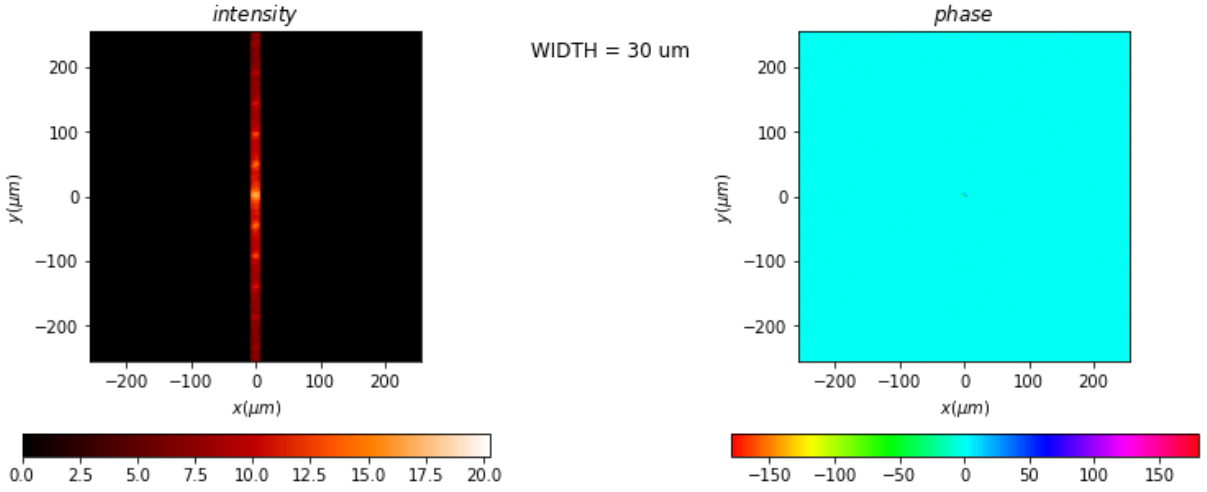


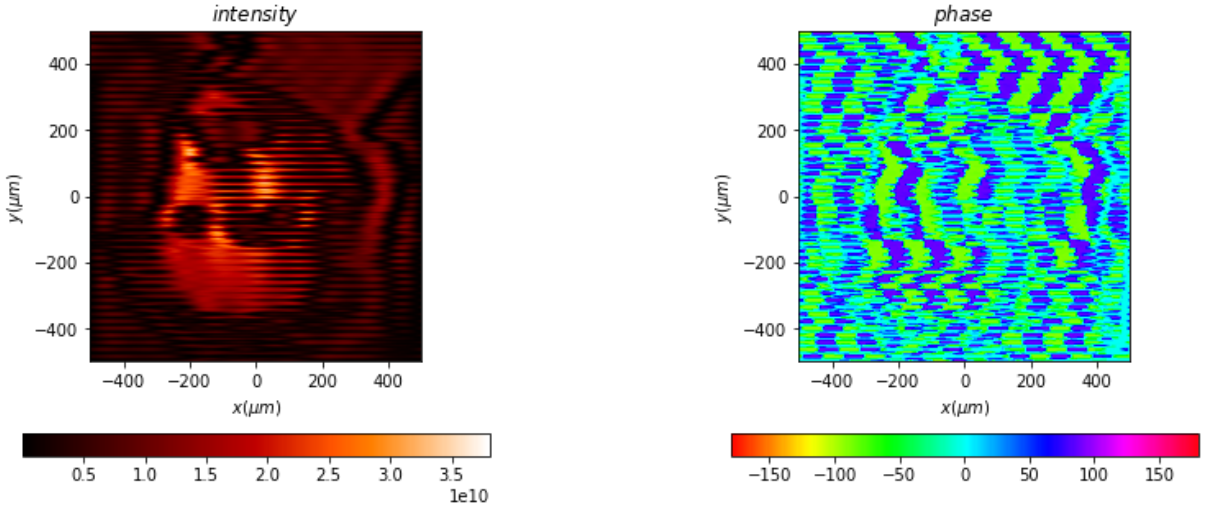
Variable Slit Widths

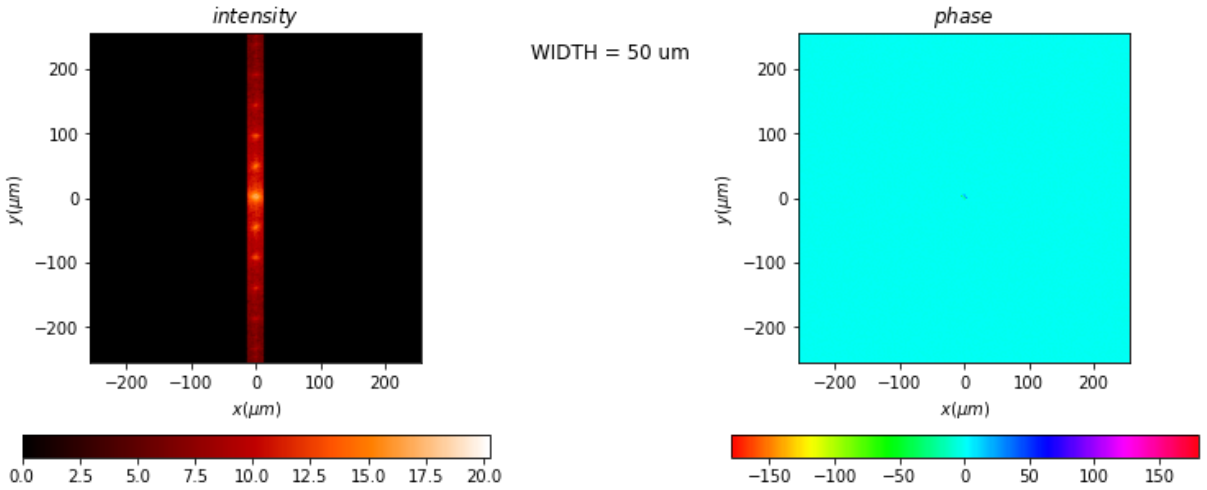
In [23]:
```
widths = [10, 30, 50, 100, 400]  # in um
for width in widths:
    varSlit = Scalar_mask_XY(x0, y0, wavelength)
    varSlit.slit(
        x0=0 * um,
        size=width * um
    )
#     varSlit.draw(kind='field', logarithm=True)

    a_L1_VarSlit, a_L1_VarSlit_L2 = sim(a_L1, varSlit)
    a_L1_VarSlit.draw(title=f" WIDTH = {width} um ", kind='field', logarithm=True)
    a_L1_VarSlit_L2.draw(kind='field', logarithm=False)
```

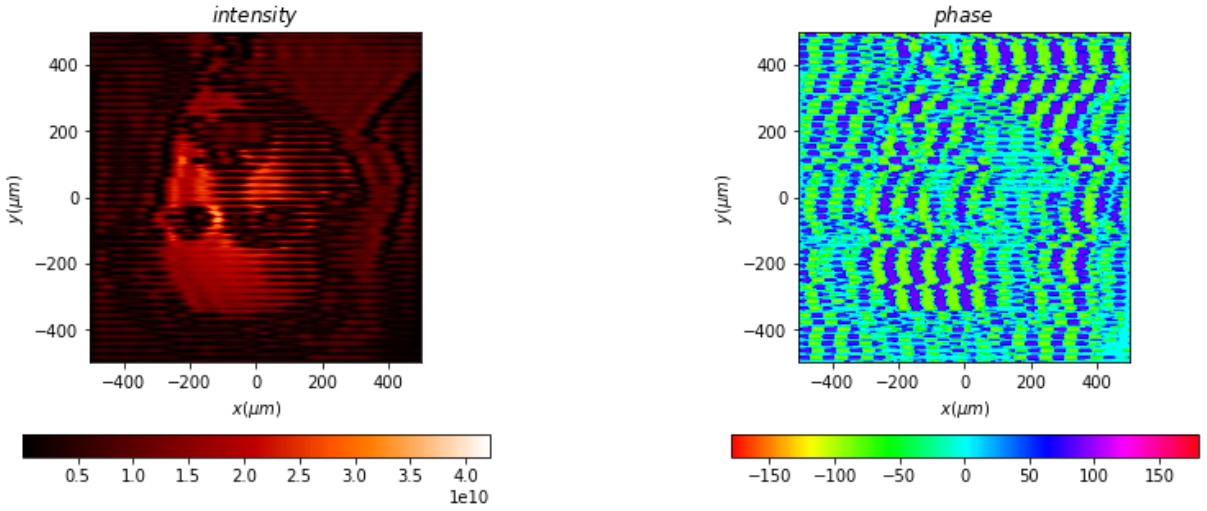

intensity
phase
WIDTH = 10 um
y(μm)
x(μm)
1e10

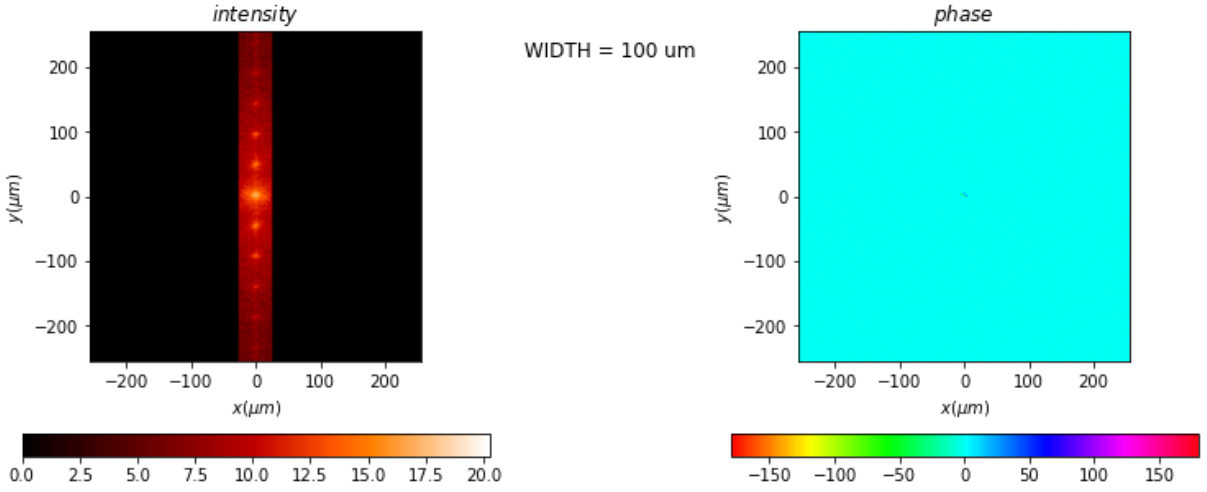

intensity
phase
y(μm)
x(μm)
1e10

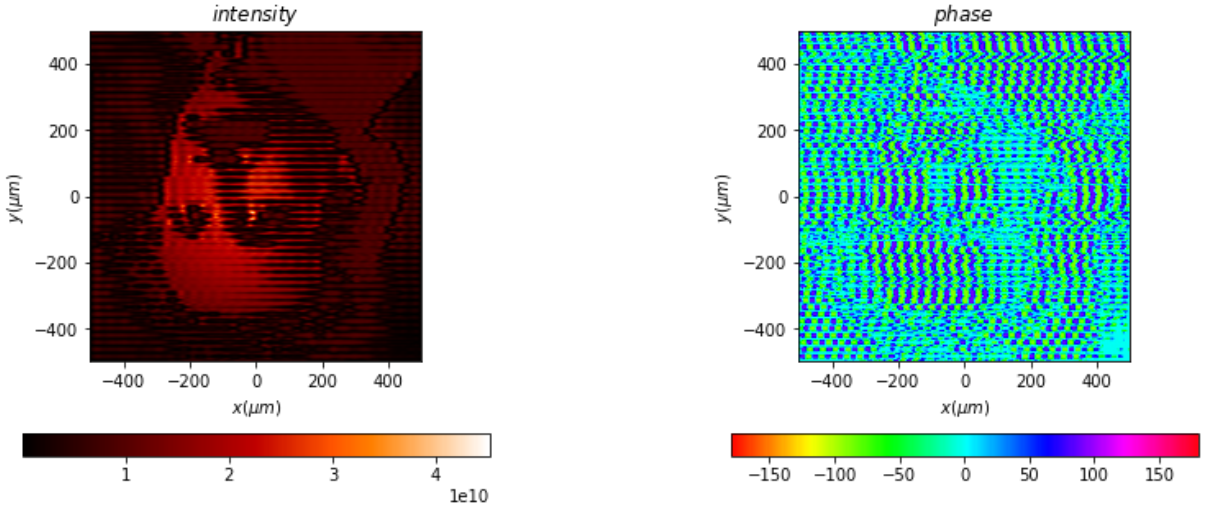

intensity
phase
WIDTH = 30 um
y(μm)
x(μm)
1e10

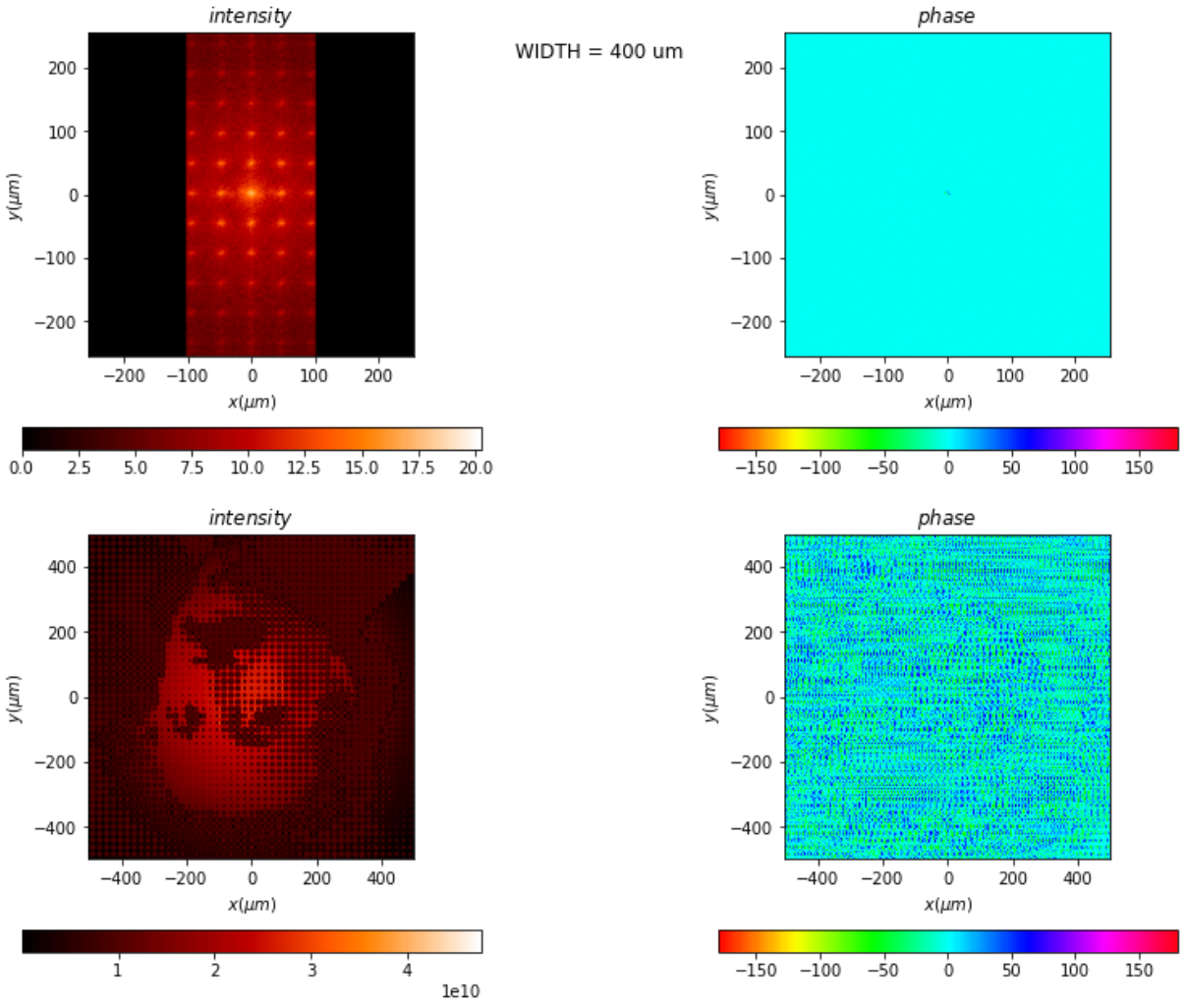

intensity
phase
y(μm)
x(μm)
1e10


intensity
phase
WIDTH = 50 um
y(μm)
x(μm)
1e10


intensity
phase
y(μm)
x(μm)
1e10


intensity
phase
WIDTH = 100 um
y(μm)
x(μm)
1e10


intensity
phase
y(μm)
x(μm)
1e10


intensity
WIDTH = 400 um
phase
200
100
0
−100
−200
y(μm)
−200 −100 0 100 200
x(μm)
0.0 2.5 5.0 7.5 10.0 12.5 15.0 17.5 20.0
−150 −100 −50 0 50 100 150
intensity
phase
400
200
0
−200
−400
−400 −200 0 200 400
1 2 3 4
1e10

In [ ]:

Alphabets - Spatial Filtering and Character Recognition

In [1]:
```
import numpy as np
from diffractio import mm, um, degrees
from diffractio.scalar_sources_XY import Scalar_source_XY
from diffractio.scalar_masks_XY import Scalar_mask_XY


# Setting up
length = 1 * mm
num_data = 512
x0 = np.linspace(-length / 2, length / 2, num_data)
y0 = np.linspace(-length / 2, length / 2, num_data)
wavelength = 0.633 * um
```

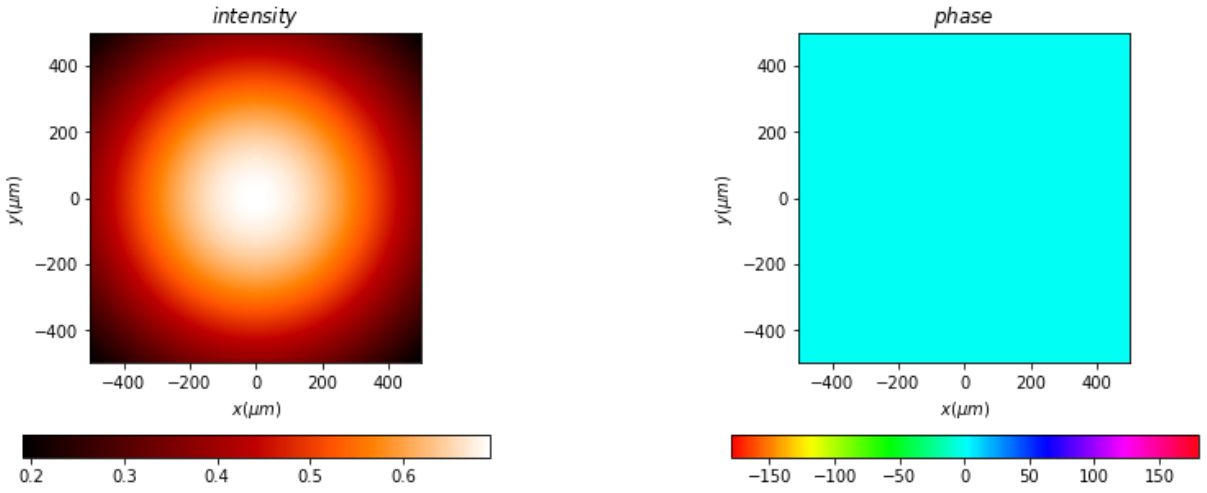
```
number of processors: 12
```

Setting up source

- Gaussian Beam (LASER)

In [2]:
```
# Gaussian Beam Source - Like a LASER
u0 = Scalar_source_XY(x=x0, y=y0, wavelength=wavelength)
u0.gauss_beam(r0=(0, 0), w0=(800 * um, 800 * um), z0=0.0)
u0.draw(kind='field', logarithm=True)
```

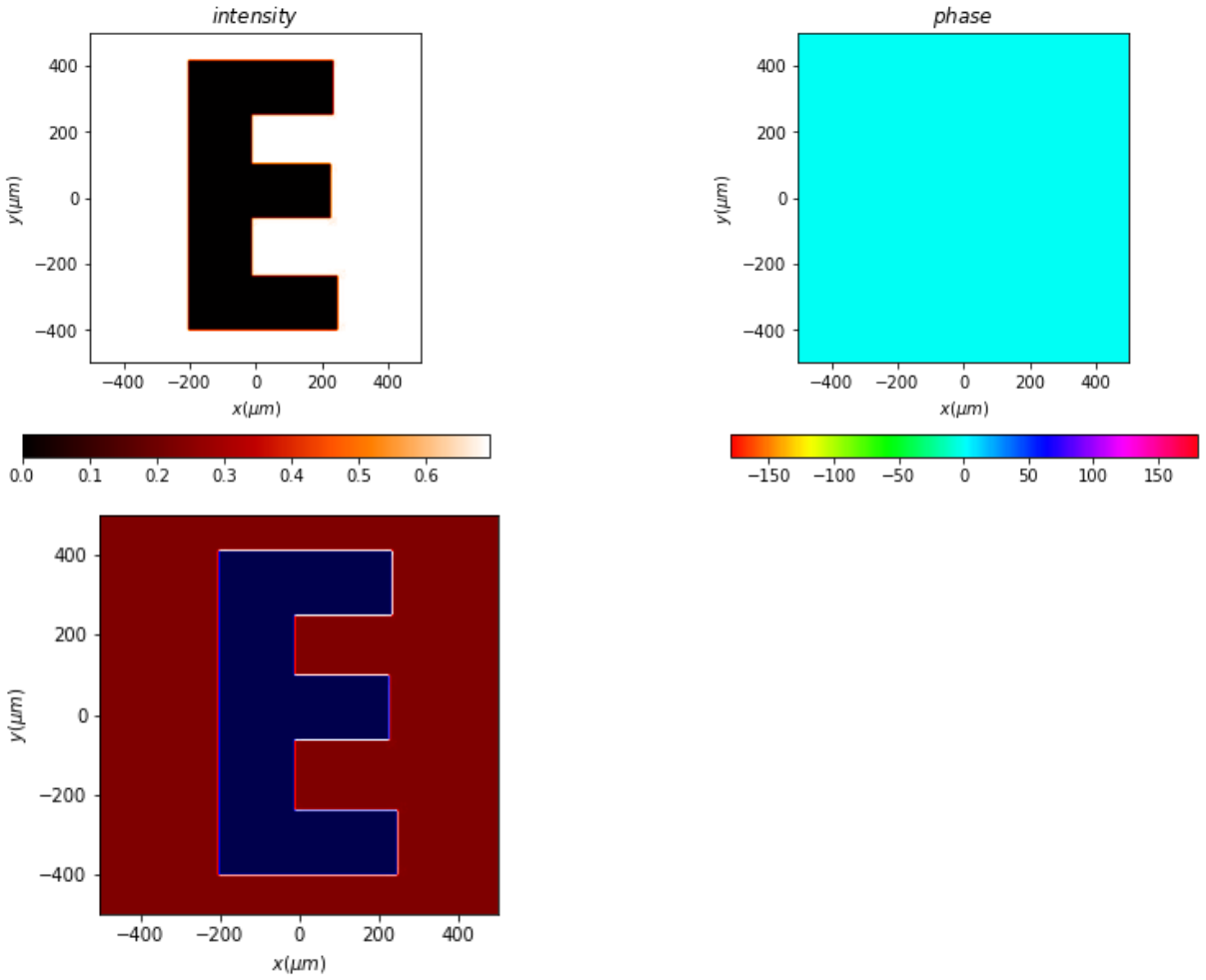
Out[2]:
```
((<matplotlib.image.AxesImage at 0x218cbf0da60>,
  <matplotlib.image.AxesImage at 0x218cc1ad220>),
 None,
 None)
```



E & F

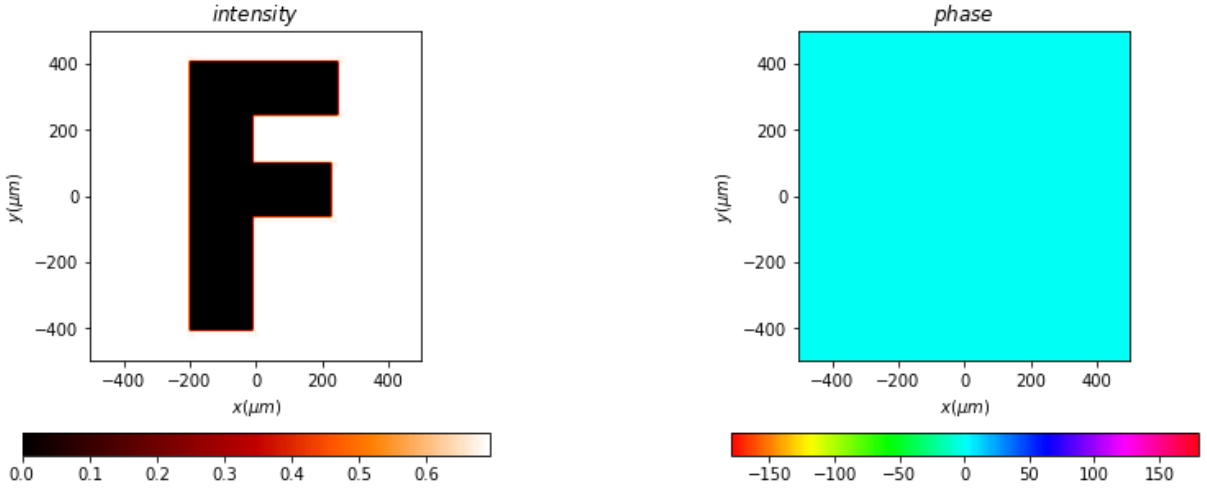
In [3]:
```
E = Scalar_mask_XY(x0, y0, wavelength)
E.image(
    filename="E.png",
    normalize=True,
    canal=0,
)

E.draw(kind='field', logarithm=True)
E.draw(kind='real_field', logarithm=True)
```

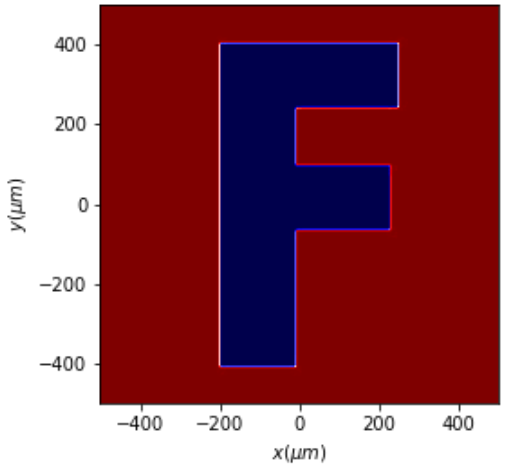
Out[3]:
```
(<Figure size 432x288 with 1 Axes>,
 <AxesSubplot:xlabel='$x  (\\mu m)$', ylabel='$y  (\\mu m)$'>,
 <matplotlib.image.AxesImage at 0x218cca87dc0>)
```



In [4]:
```
F = Scalar_mask_XY(x0, y0, wavelength)
F.image(
    filename="F.png",
    normalize=True,
    canal=0,
)

F.draw(kind='field', logarithm=True)
F.draw(kind='real_field', logarithm=True)
```

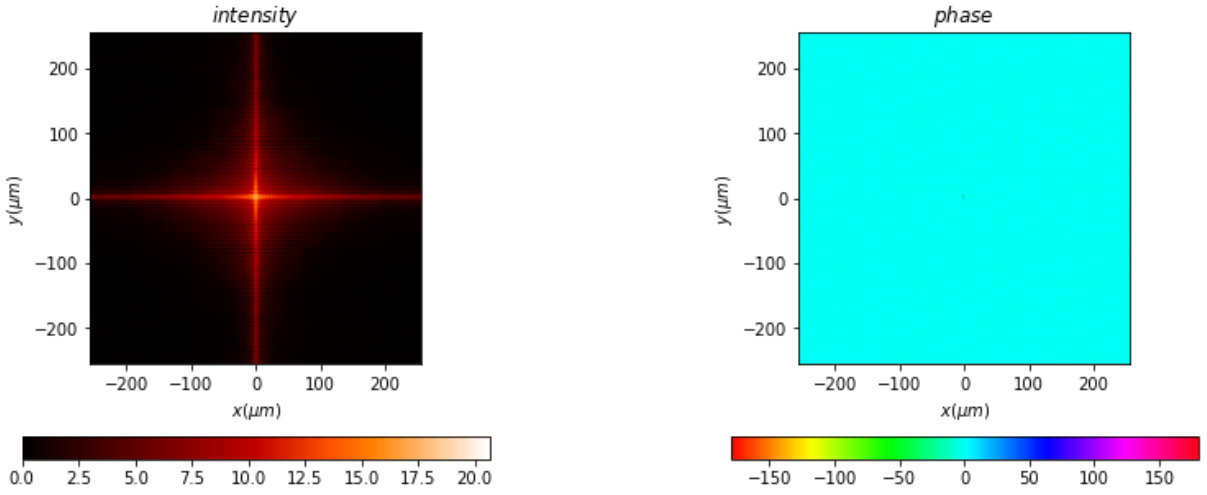
Out[4]:
```
(<Figure size 432x288 with 1 Axes>,
 <AxesSubplot:xlabel='$x  (\\mu m)$', ylabel='$y  (\\mu m)$'>,
 <matplotlib.image.AxesImage at 0x218cd0785b0>)
```





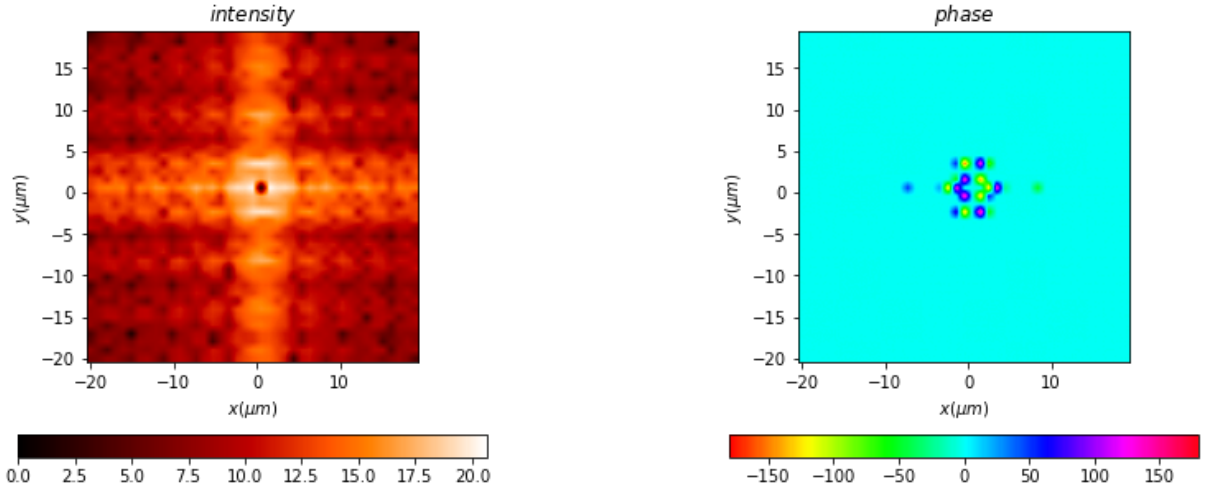
In [5]:
```
# Fourier Plane - No Filter
E_a_L1 = (u0 * E).fft(z=1 * mm, new_field=True)
E_a_L1.draw(kind='field', logarithm=True)
```

Out[5]:
```
((<matplotlib.image.AxesImage at 0x218d0d64190>,
  <matplotlib.image.AxesImage at 0x218d0e0e910>),
 None,
 None)
```



In [6]:
```
# A closer look
E_a_L1_Crop = E_a_L1.cut_resample(x_limits=(-20, 20), y_limits=(-20, 20), new_field=True)
E_a_L1_Crop.draw(kind='field', logarithm=True)
```

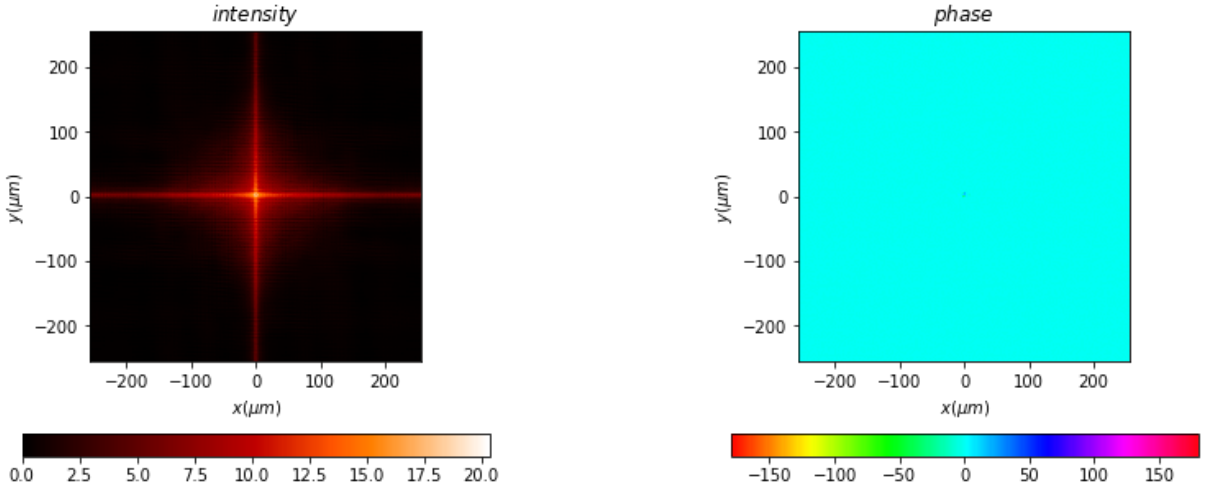
Out[6]:
```
((<matplotlib.image.AxesImage at 0x218d0d9ac10>,
  <matplotlib.image.AxesImage at 0x218cd043400>),
 None,
 None)
```



In [7]:
```
# Fourier Plane - No Filter
```

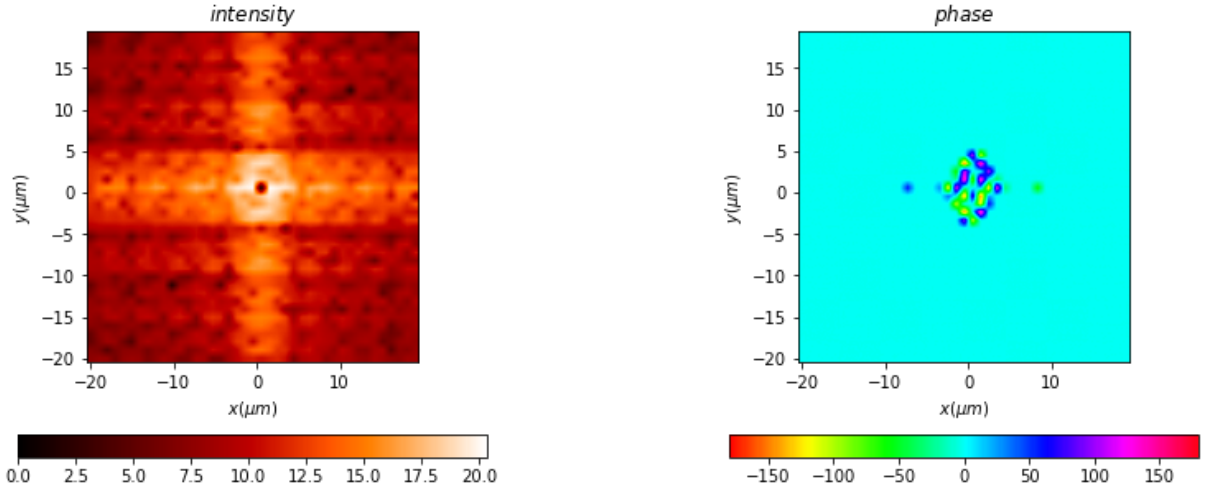
```
F_a_L1 = (u0 * F).fft(z=1 * mm, new_field=True)
F_a_L1.draw(kind='field', logarithm=True)
```

Out[7]:
```
((<matplotlib.image.AxesImage at 0x218d0e9a940>,
  <matplotlib.image.AxesImage at 0x218ccb75520>),
 None,
 None)
```



In [8]:
```
# A closer look
F_a_L1_Crop = F_a_L1.cut_resample(x_limits=(-20, 20), y_limits=(-20, 20), new_field=True)
F_a_L1_Crop.draw(kind='field', logarithm=True)
```

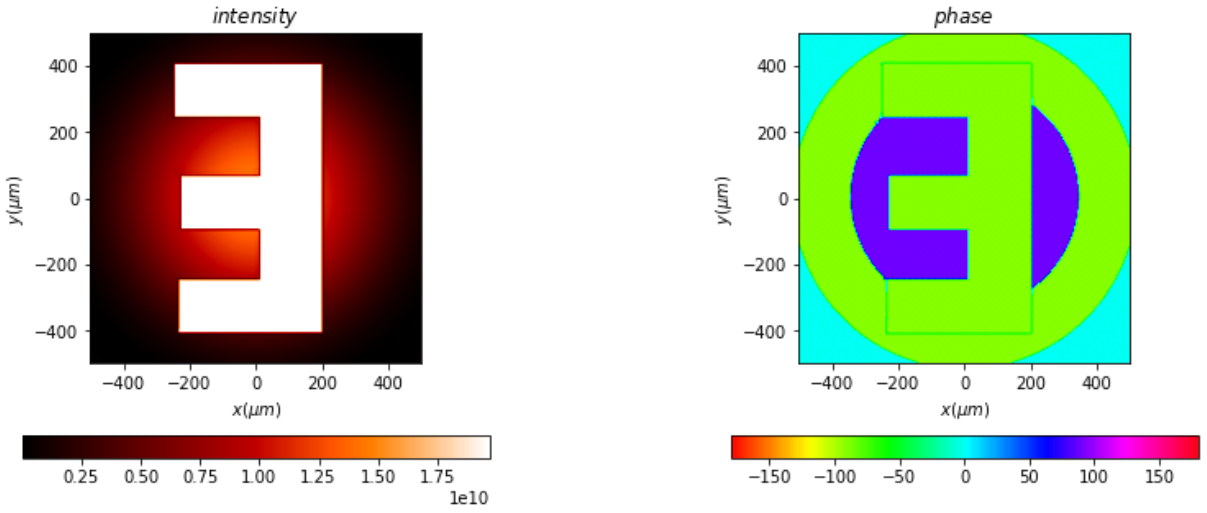
Out[8]:
```
((<matplotlib.image.AxesImage at 0x218ccbc7fa0>,
  <matplotlib.image.AxesImage at 0x218d0f32730>),
 None,
 None)
```



So, the difference in the Fourier encoding for two very similar characters or alphabets lies not so much in the position-space, as it does in the phase-space. This "hidden" phase-space information can be used to enable better machine encodings for Character Recognition software.

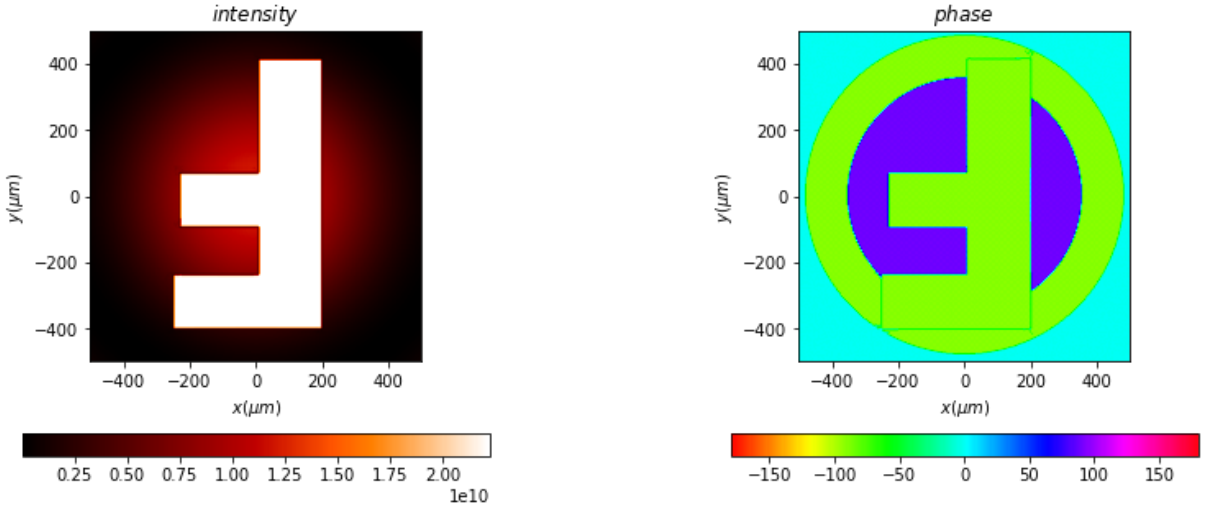
In [9]:
```
# This is how, it'd look without any filter, at the Observation screen
E_a_L2 = E_a_L1.fft(z=1 * mm, shift=False, remove0=False, new_field=True)
E_a_L2.draw(kind='field', logarithm=False)
```

Out[9]:
```
((<matplotlib.image.AxesImage at 0x218d0bf7130>,
  <matplotlib.image.AxesImage at 0x218d0c8c370>),
 None,
 None)
```



In [10]:
```
# This is how, it'd look without any filter, at the Observation screen
F_a_L2 = F_a_L1.fft(z=1 * mm, shift=False, remove0=False, new_field=True)
F_a_L2.draw(kind='field', logarithm=False)
```

Out[10]:
```
((<matplotlib.image.AxesImage at 0x218d15767f0>,
  <matplotlib.image.AxesImage at 0x218d1604a60>),
 None,
 None)
```



Some Masks / Filters

- Center Dot - High Pass
- Square - Low Pass
- Square + Center Dot -

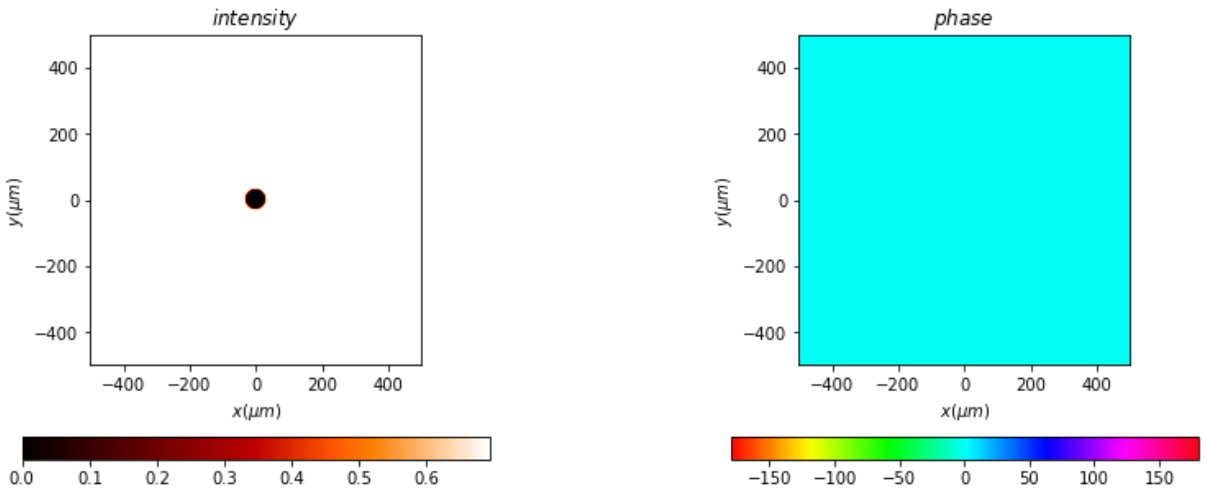
- Vertical Slit

- Horizontal Slit
- Angled Slit
- Variable Slits

- Mesh

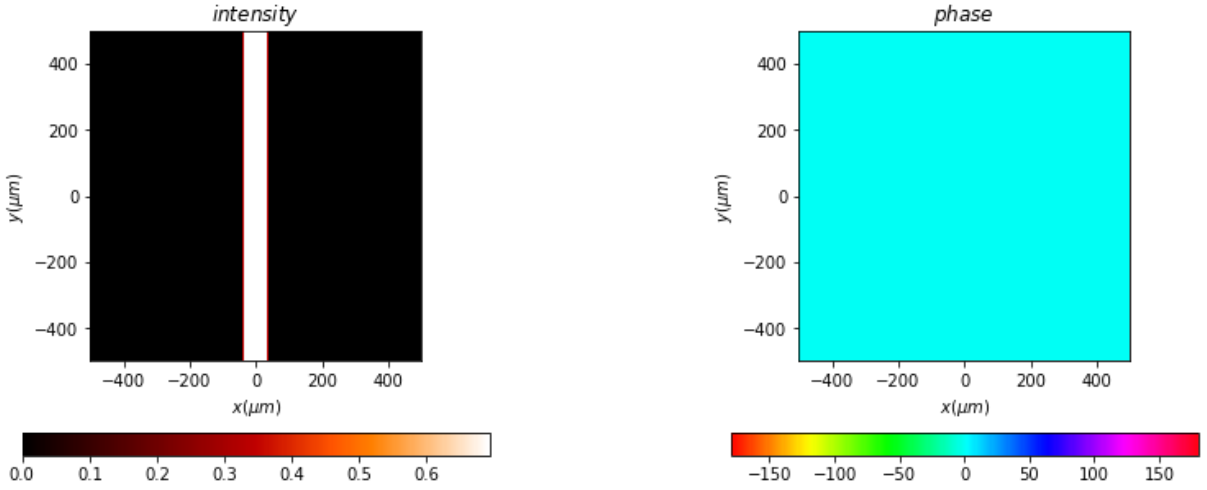
In [11]:
```
cDot = Scalar_mask_XY(x0, y0, wavelength)
cDot.ring(
    r0=(0 * um, 0 * um),
    radius1=(30 * um, 30 * um),
    radius2=(1000 * um, 1000 * um)
)
cDot.draw(kind='field', logarithm=True)
```

Out[11]:
```
((<matplotlib.image.AxesImage at 0x218d15a4190>,
  <matplotlib.image.AxesImage at 0x218d0dc1100>),
 None,
 None)
```



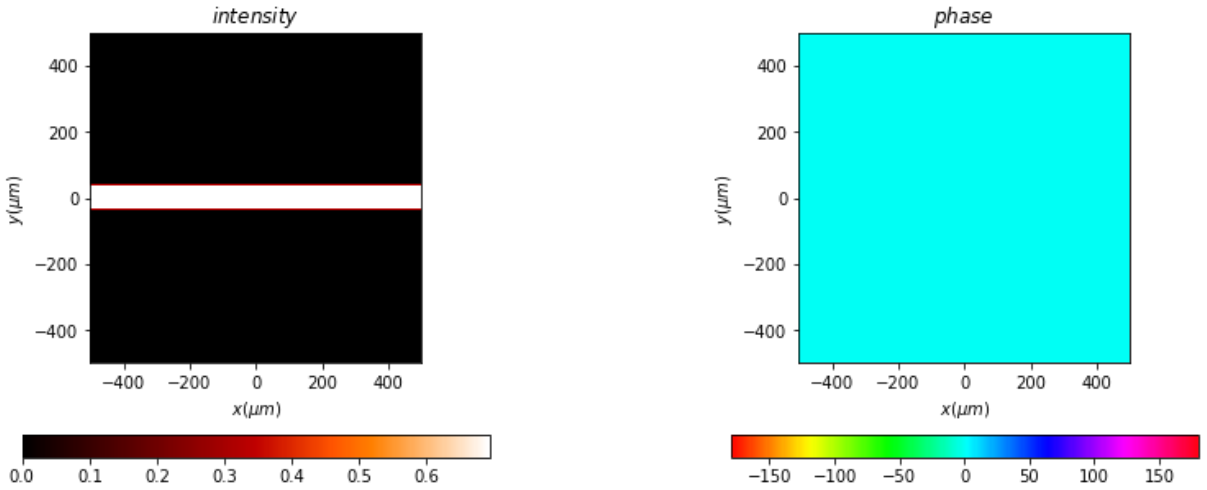
In [12]:
```
vert = Scalar_mask_XY(x0, y0, wavelength)
vert.slit(
    x0=0 * um,
    size=75 * um
)
vert.draw(kind='field', logarithm=True)
```

Out[12]:
```
((<matplotlib.image.AxesImage at 0x218d1fbb910>,
  <matplotlib.image.AxesImage at 0x218d20430d0>),
 None,
 None)
```



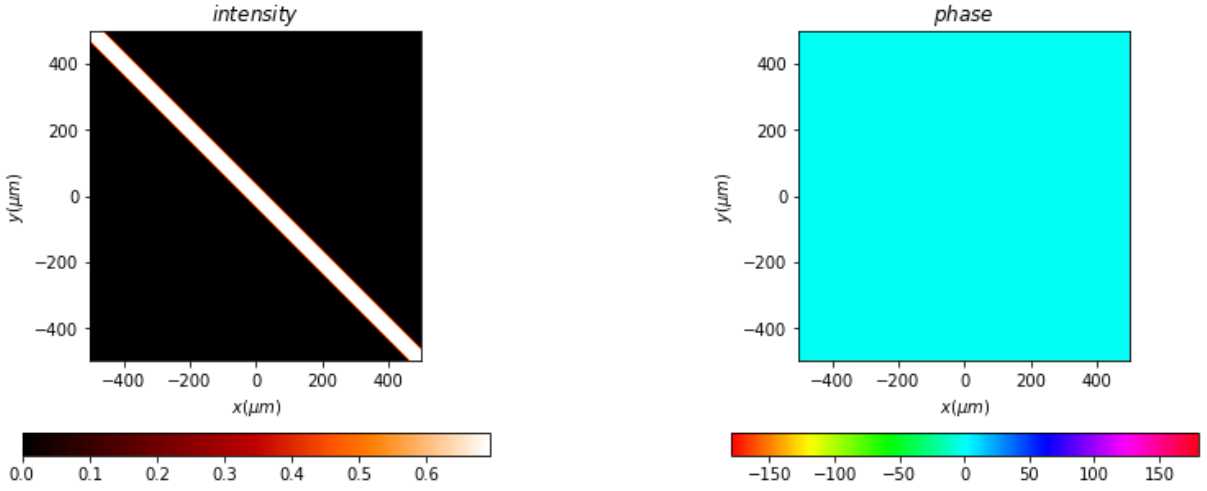
In [13]:
```
horiz = Scalar_mask_XY(x0, y0, wavelength)
horiz.slit(
    x0=0 * um,
    size=75 * um,
    angle=np.pi / 2
)
horiz.draw(kind='field', logarithm=True)
```

Out[13]:
```
((<matplotlib.image.AxesImage at 0x218d2108370>,
  <matplotlib.image.AxesImage at 0x218d2184b50>),
 None,
 None)
```



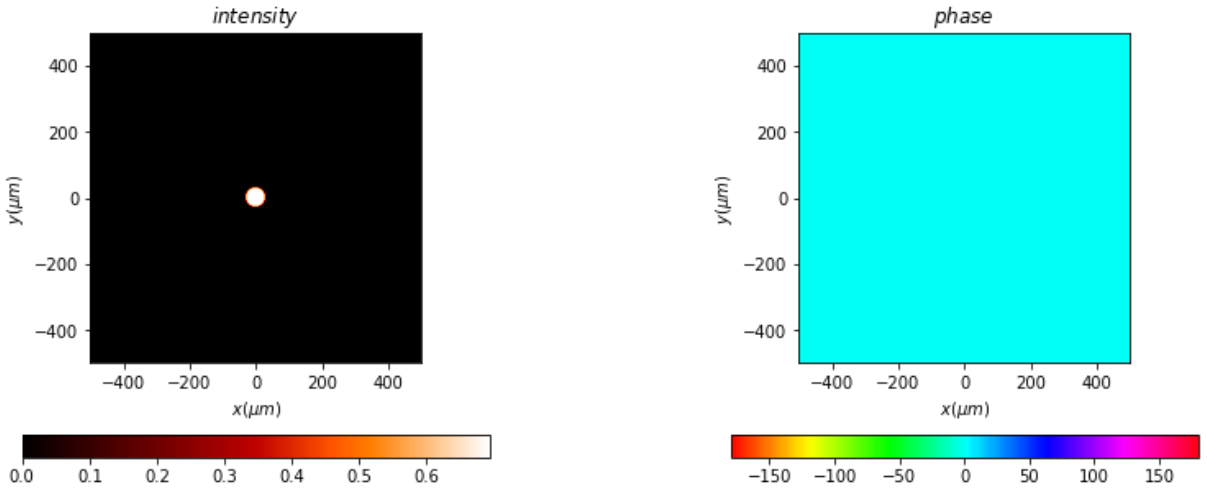
In [14]:
```
angled = Scalar_mask_XY(x0, y0, wavelength)
angled.slit(
    x0=0 * um,
    size=50 * um,
    angle=np.pi / 4
)
angled.draw(kind='field', logarithm=True)
```

Out[14]:
```
((<matplotlib.image.AxesImage at 0x218d30abdf0>,
  <matplotlib.image.AxesImage at 0x218d312d5e0>),
 None,
 None)
```



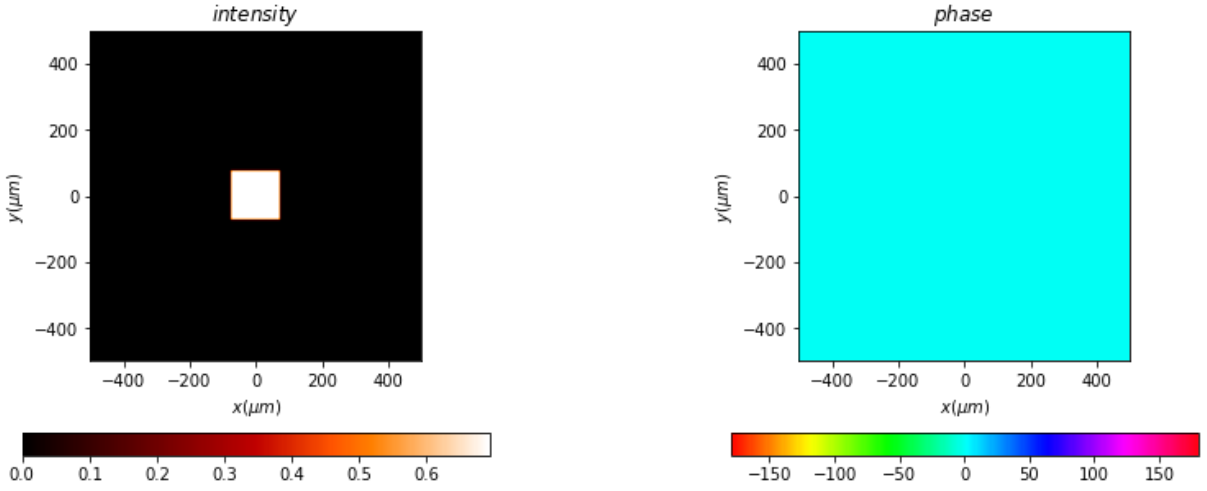
In [15]:
```
circ = Scalar_mask_XY(x0, y0, wavelength)
circ.circle(
    r0=(0 * um, 0 * um),
    radius=(30 * um, 30 * um),
    angle=0
)
circ.draw(kind='field', logarithm=True)
```

Out[15]:
```
((<matplotlib.image.AxesImage at 0x218d31d6ca0>,
  <matplotlib.image.AxesImage at 0x218d4aeef70>),
 None,
 None)
```



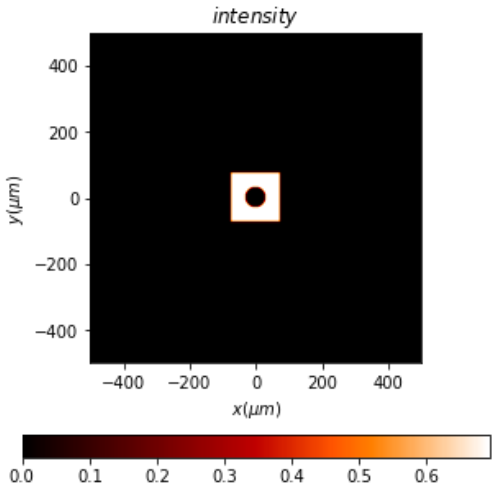
In [16]:
```
sq = Scalar_mask_XY(x0, y0, wavelength)
sq.square(
    r0=(0 * um, 0 * um),
    size=(150 * um, 150 * um),
    angle=0,
)
sq.draw(kind='field', logarithm=True)
```

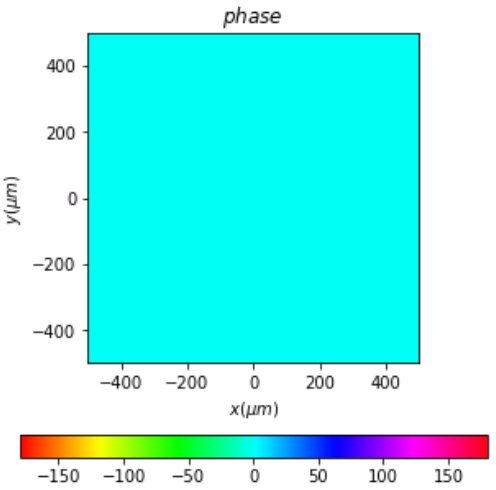
Out[16]:
```
((<matplotlib.image.AxesImage at 0x218d4ba1520>,
  <matplotlib.image.AxesImage at 0x218d4c357c0>),
 None,
 None)
```



In [17]:
```
# Adding two filters
sqCirc = sq * cDot
sqCirc.draw(kind="field", logarithm=True)
```

Out[17]:
```
((<matplotlib.image.AxesImage at 0x218d4bb70d0>,
  <matplotlib.image.AxesImage at 0x218d17567f0>),
 None,
 None)
```





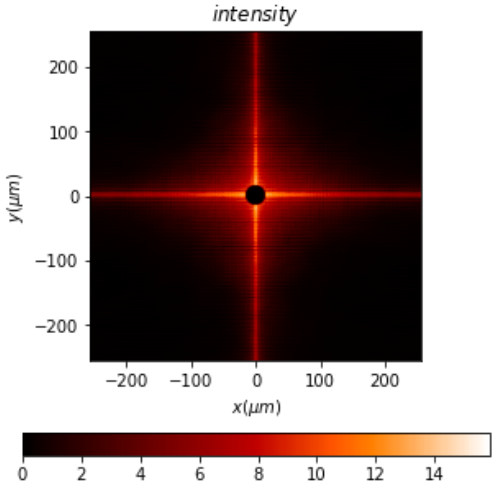
Returns wave after encountering filter and then L2

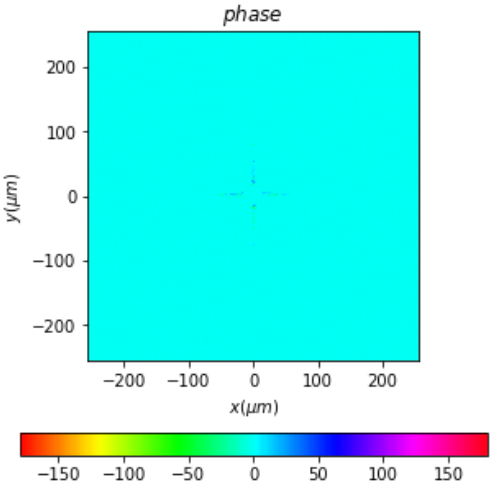
In [18]:
```
def sim(a_L1, mask):
    """
    a_L1: Wave after passing through Lens 1
    mask: Mask or Filter to apply

    """
    a_L1_Mask = a_L1 * mask
    a_L1_Mask_L2 = a_L1_Mask.fft(z=1 * mm, shift=False, remove0=False, new_field=True)

    return a_L1_Mask, a_L1_Mask_L2
```

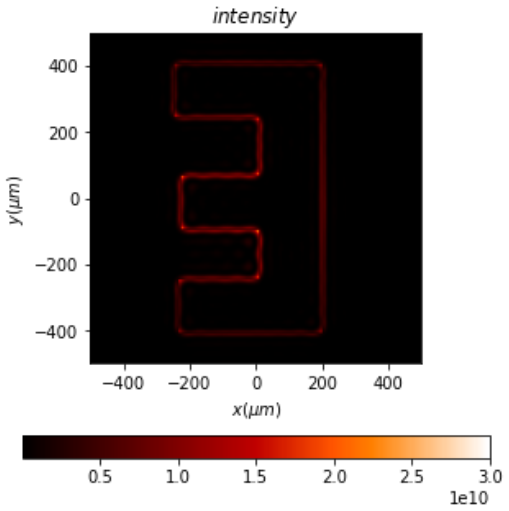
Center Dot - High Pass - HT Structure Visible - Dark-Field Illumination / Edge Enhancement

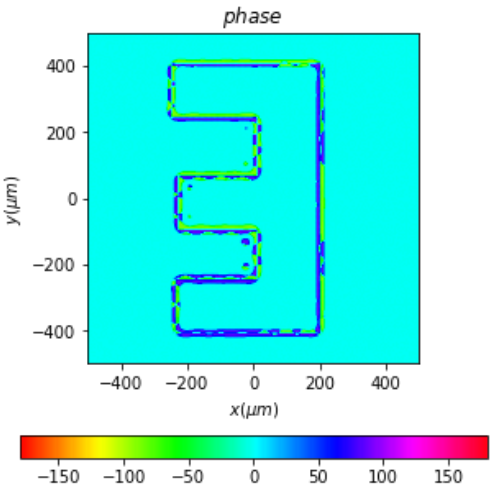
In [19]:
```
E_a_L1_cD, E_a_L1_cD_L2 = sim(E_a_L1, cDot)
E_a_L1_cD.draw(kind='field', logarithm=True)
E_a_L1_cD_L2.draw(kind='field', logarithm=False)
```

Out[19]:
```
((<matplotlib.image.AxesImage at 0x218d5f0c7c0>,
  <matplotlib.image.AxesImage at 0x218d7787f40>),
 None,
 None)
```





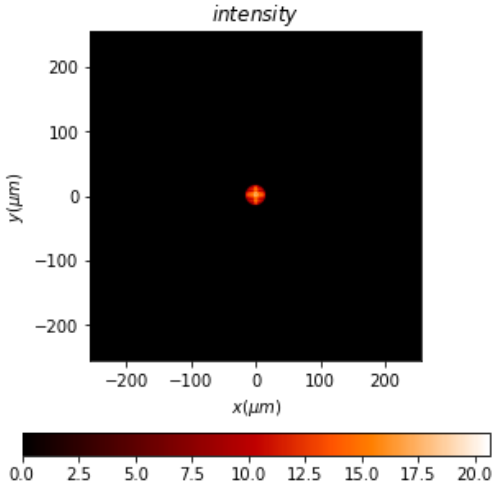


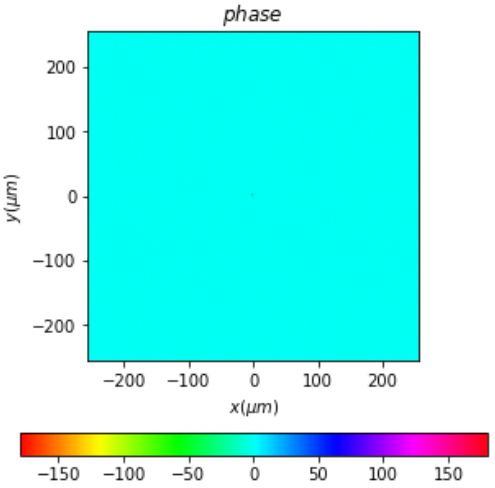


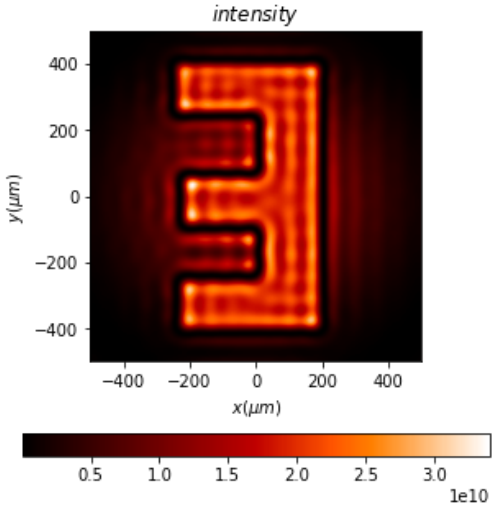
Circle - Low Pass - Blurred Output

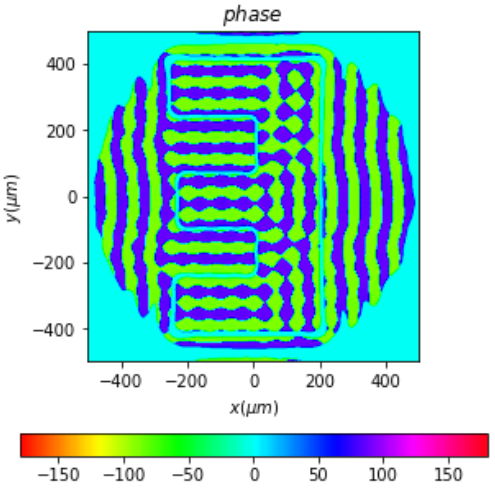
In [20]:
```
E_a_L1_Circ, E_a_L1_Circ_L2 = sim(E_a_L1, circ)
E_a_L1_Circ.draw(kind='field', logarithm=True)
E_a_L1_Circ_L2.draw(kind='field', logarithm=False)
```

Out[20]:
```
((<matplotlib.image.AxesImage at 0x218d8423370>,
  <matplotlib.image.AxesImage at 0x218d848e5e0>),
 None,
 None)
```





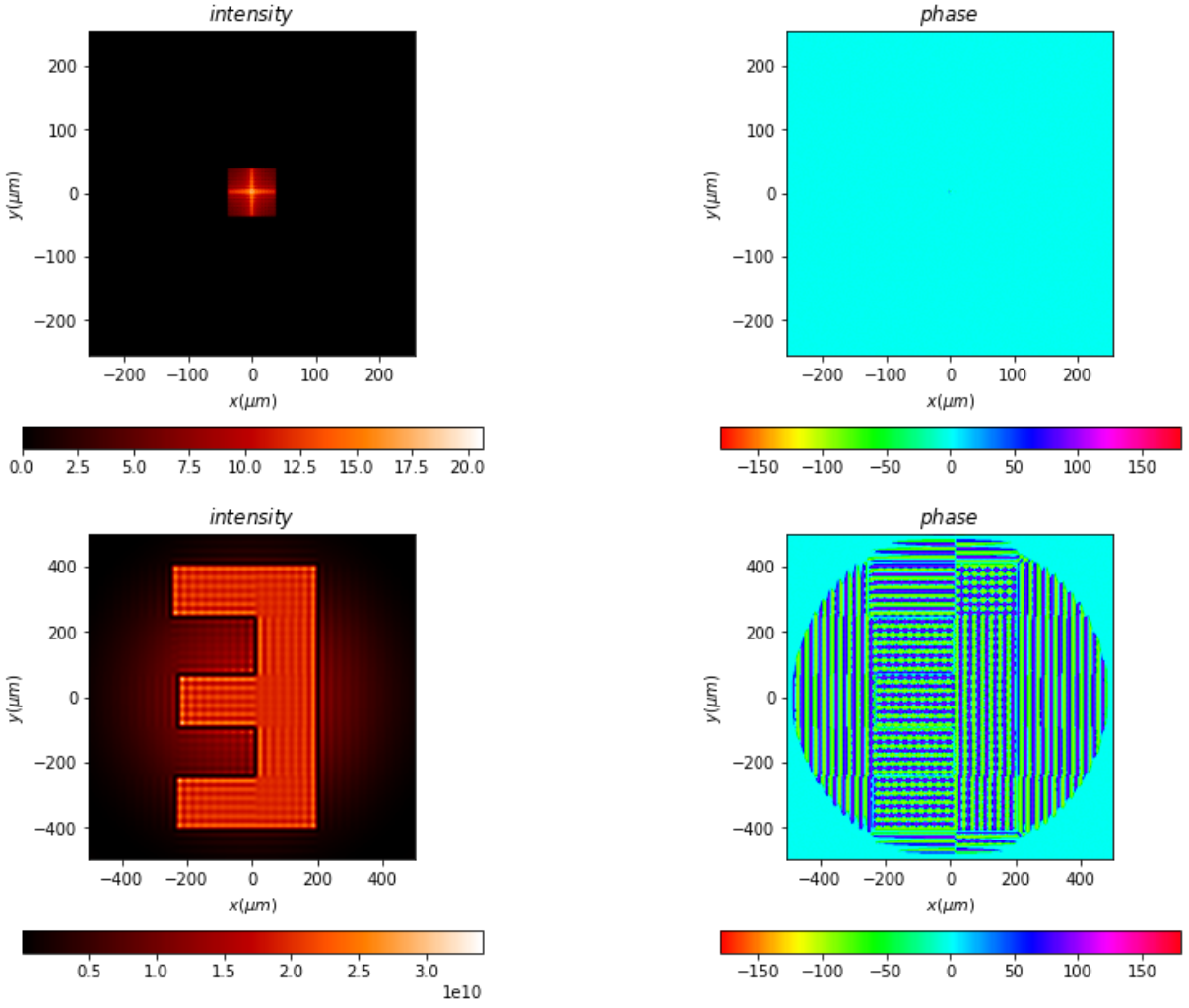




Square

In [21]:
```
E_a_L1_Sq, E_a_L1_Sq_L2 = sim(E_a_L1, sq)
E_a_L1_Sq.draw(kind='field', logarithm=True)
E_a_L1_Sq_L2.draw(kind='field', logarithm=False)
```

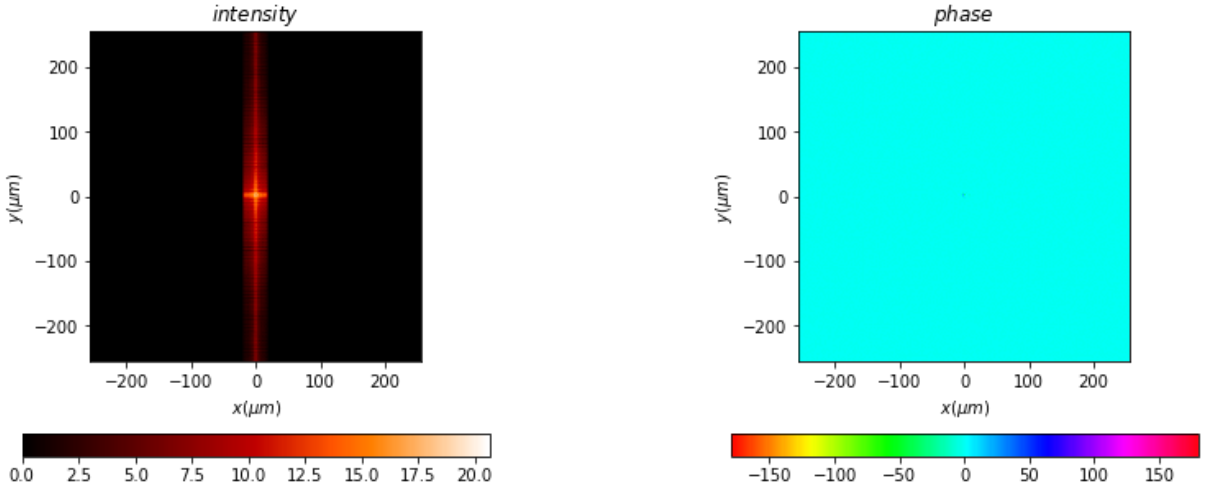
Out[21]:
```
((<matplotlib.image.AxesImage at 0x218d910f580>,
  <matplotlib.image.AxesImage at 0x218d91797f0>),
 None,
 None)
```

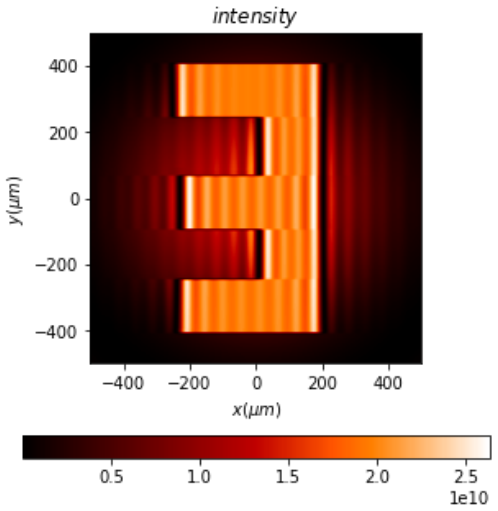


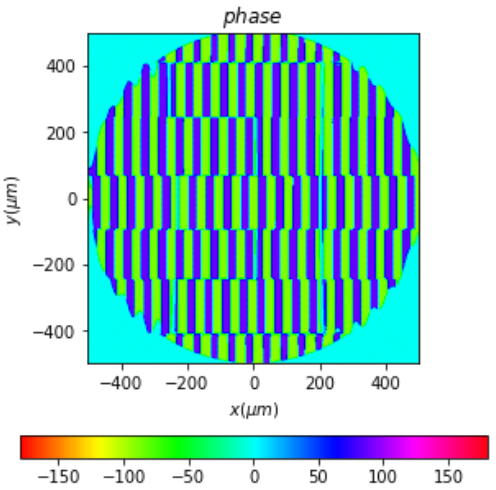
Vertical Slit

In [22]:
```
E_a_L1_Vert, E_a_L1_Vert_L2 = sim(E_a_L1, vert)
E_a_L1_Vert.draw(kind='field', logarithm=True)
E_a_L1_Vert_L2.draw(kind='field', logarithm=False)
```

Out[22]:
```
((<matplotlib.image.AxesImage at 0x218d4c9ba00>,
  <matplotlib.image.AxesImage at 0x218d83cf100>),
 None,
 None)
```



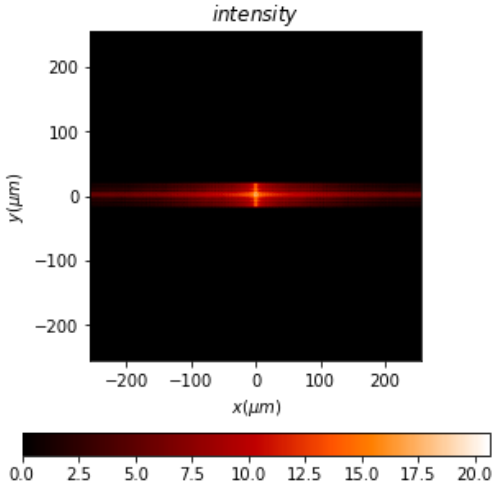


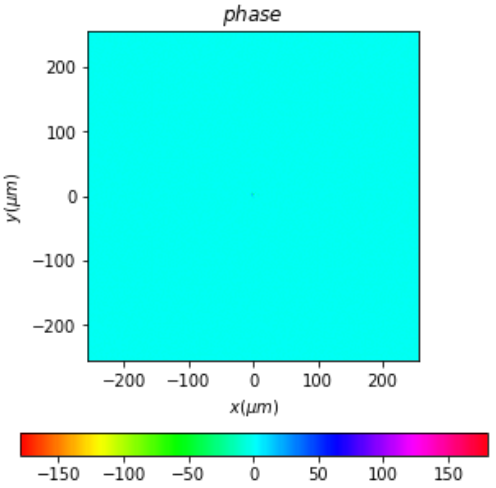


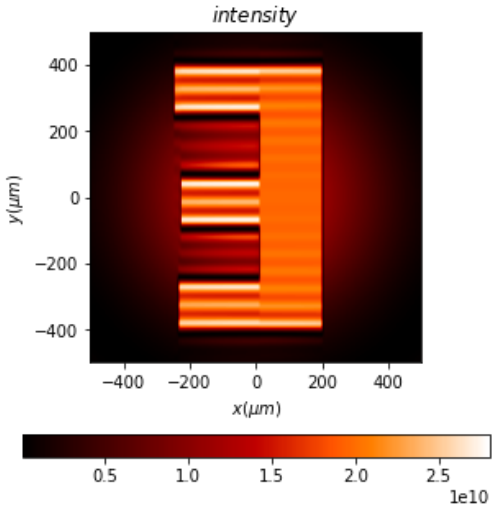
Horizontal Slit

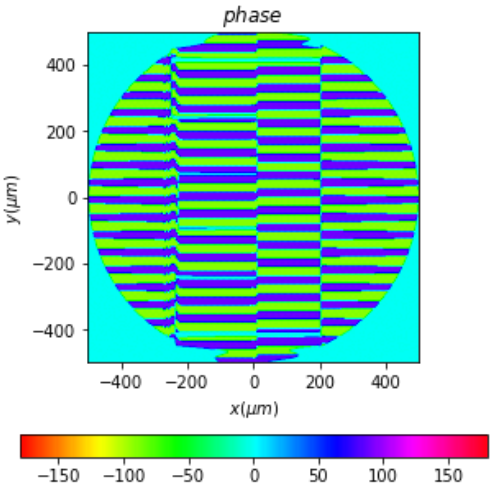
In [23]:
```
E_a_L1_Horiz, E_a_L1_Horiz_L2 = sim(E_a_L1, horiz)
E_a_L1_Horiz.draw(kind='field', logarithm=True)
E_a_L1_Horiz_L2.draw(kind='field', logarithm=False)
```

Out[23]:
```
((<matplotlib.image.AxesImage at 0x218d9d892b0>,
  <matplotlib.image.AxesImage at 0x218d9ded8e0>),
 None,
 None)
```





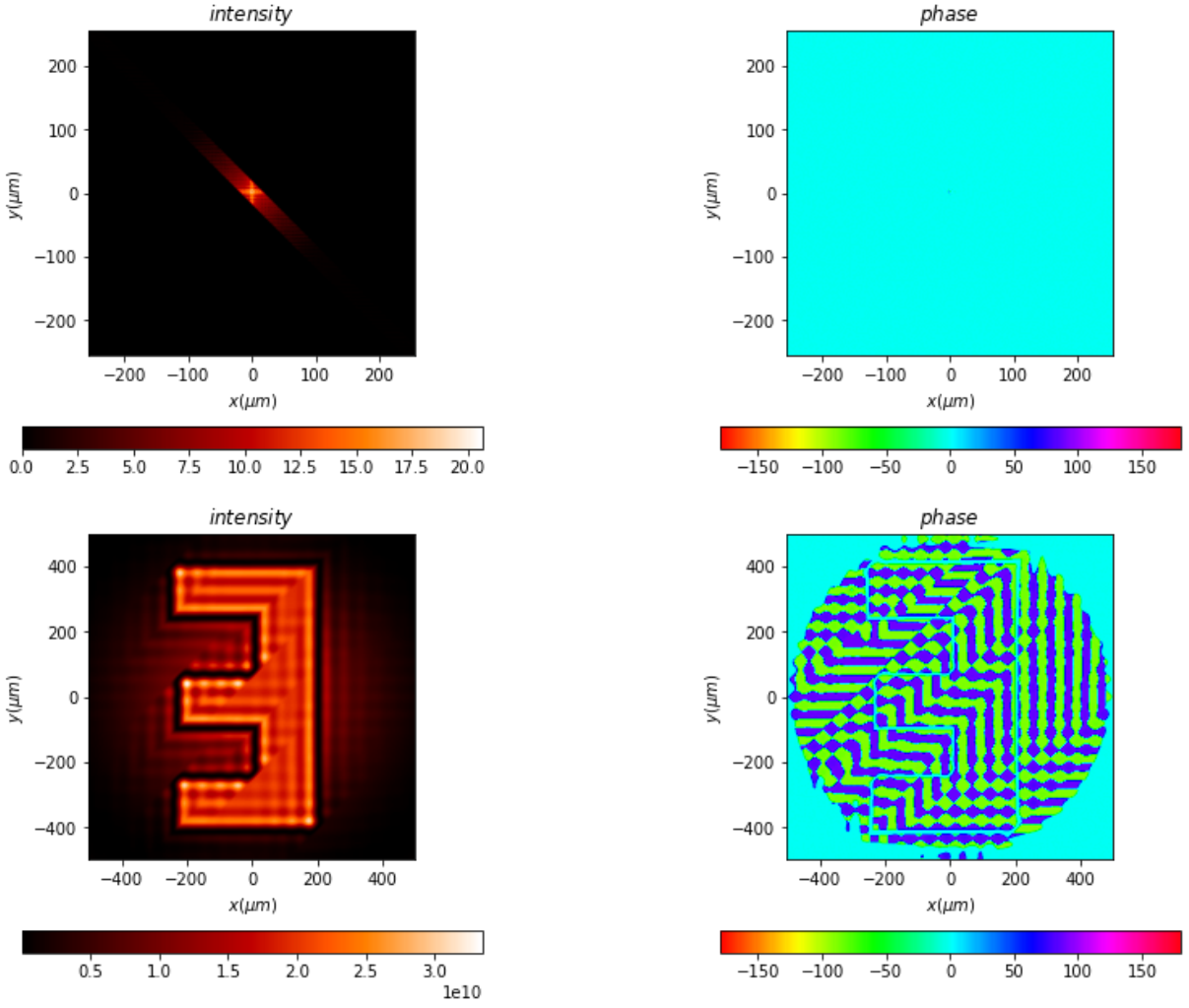




Angled Slit

In [24]:
```
E_a_L1_Angled, E_a_L1_Angled_L2 = sim(E_a_L1, angled)
E_a_L1_Angled.draw(kind='field', logarithm=True)
E_a_L1_Angled_L2.draw(kind='field', logarithm=False)
```

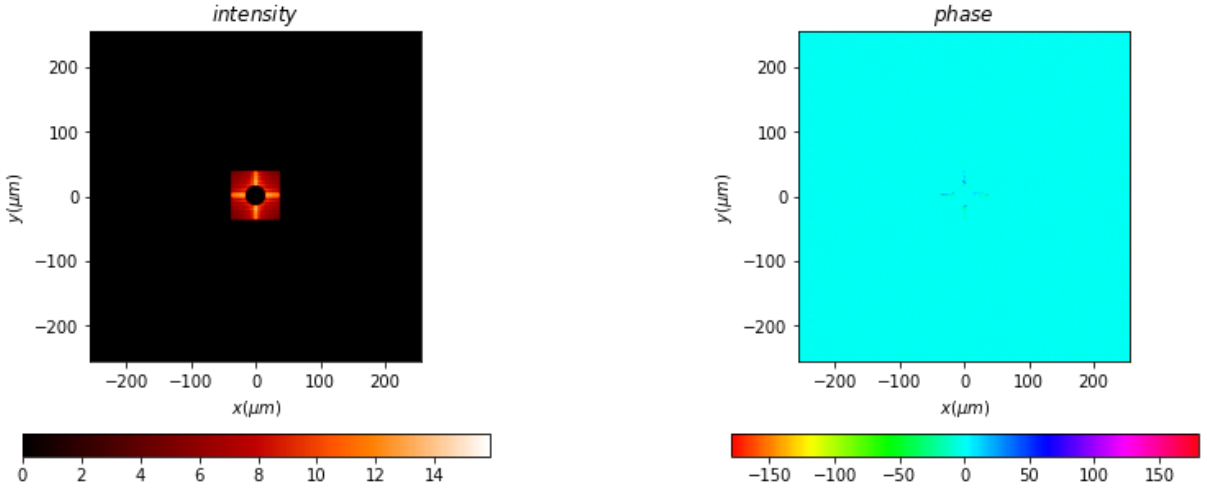
Out[24]:
```
((<matplotlib.image.AxesImage at 0x218da856a30>,
  <matplotlib.image.AxesImage at 0x218dc584ca0>),
 None,
 None)
```

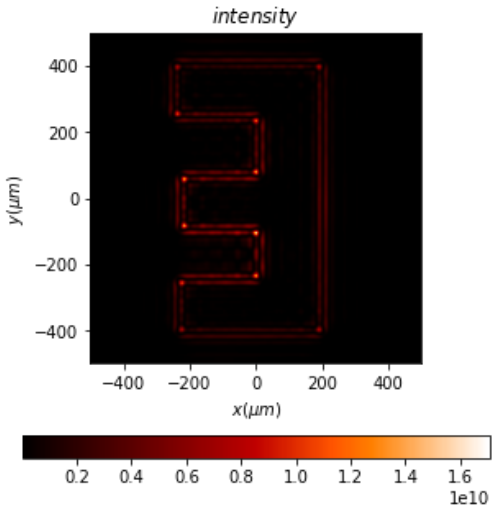


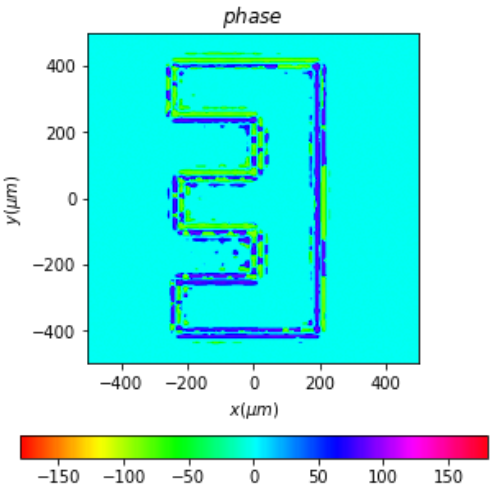
Square + Circle

In [25]:
```
E_a_L1_SqCirc, E_a_L1_SqCirc_L2 = sim(E_a_L1, sqCirc)
E_a_L1_SqCirc.draw(kind='field', logarithm=True)
E_a_L1_SqCirc_L2.draw(kind='field', logarithm=False)
```

Out[25]:
```
((<matplotlib.image.AxesImage at 0x218df855e80>,
  <matplotlib.image.AxesImage at 0x218dc5e2970>),
 None,
 None)
```

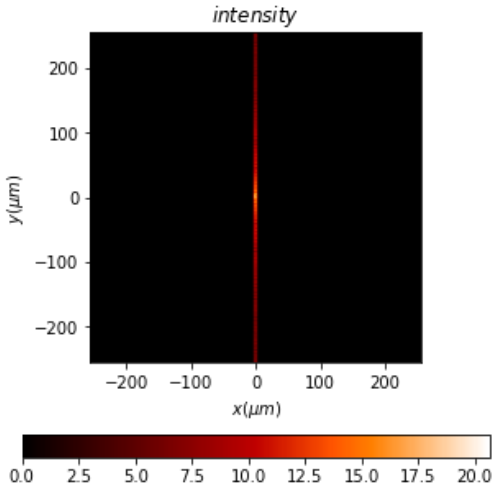




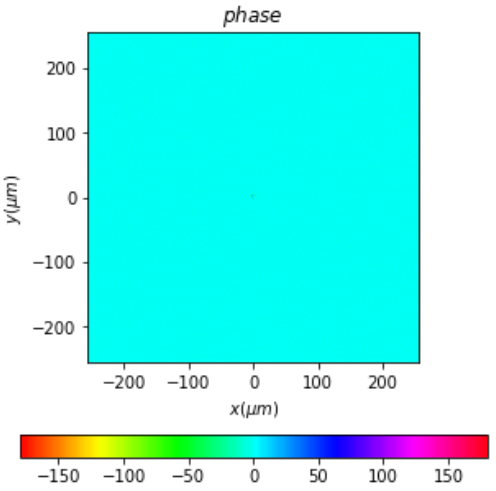


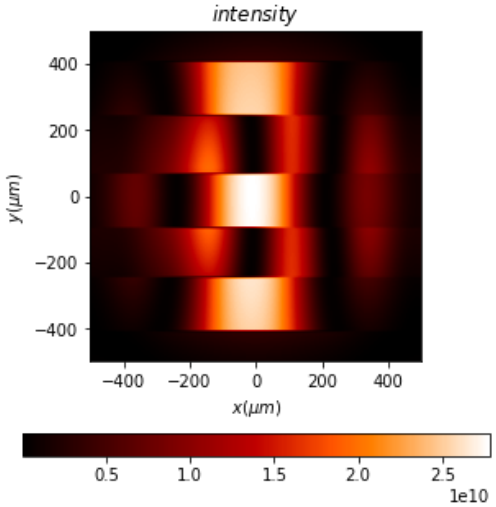
Variable Slit Widths

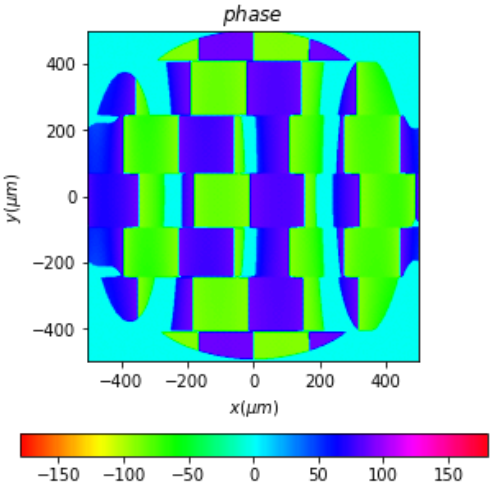
In [26]:
```
widths = [10, 30, 50, 76.3209395, 100, 400]  # in um
for width in widths:
    varSlit = Scalar_mask_XY(x0, y0, wavelength)
    varSlit.slit(
        x0=0 * um,
        size=width * um
    )

    E_a_L1_VarSlit, E_a_L1_VarSlit_L2 = sim(E_a_L1, varSlit)
    E_a_L1_VarSlit.draw(title=f" WIDTH = {width} um ", kind='field', logarithm=True)
    E_a_L1_VarSlit_L2.draw(kind='field', logarithm=False)
```

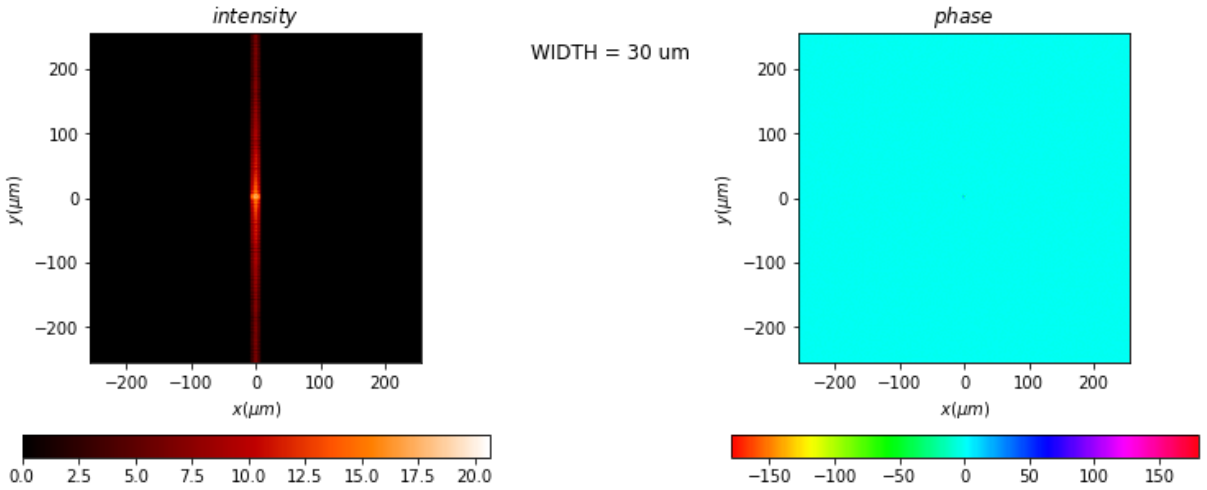


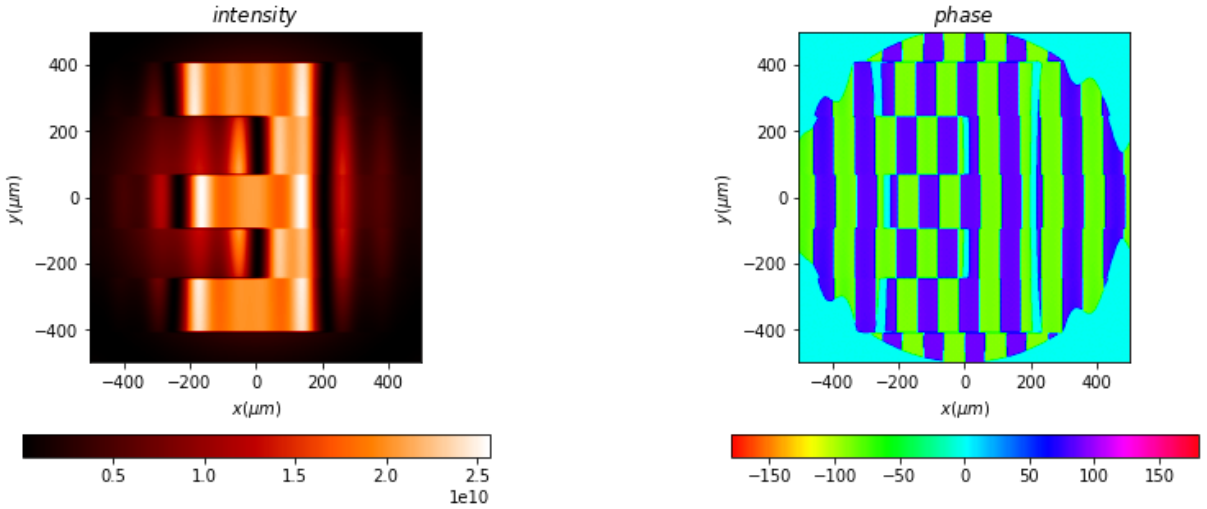
WIDTH = 10 um

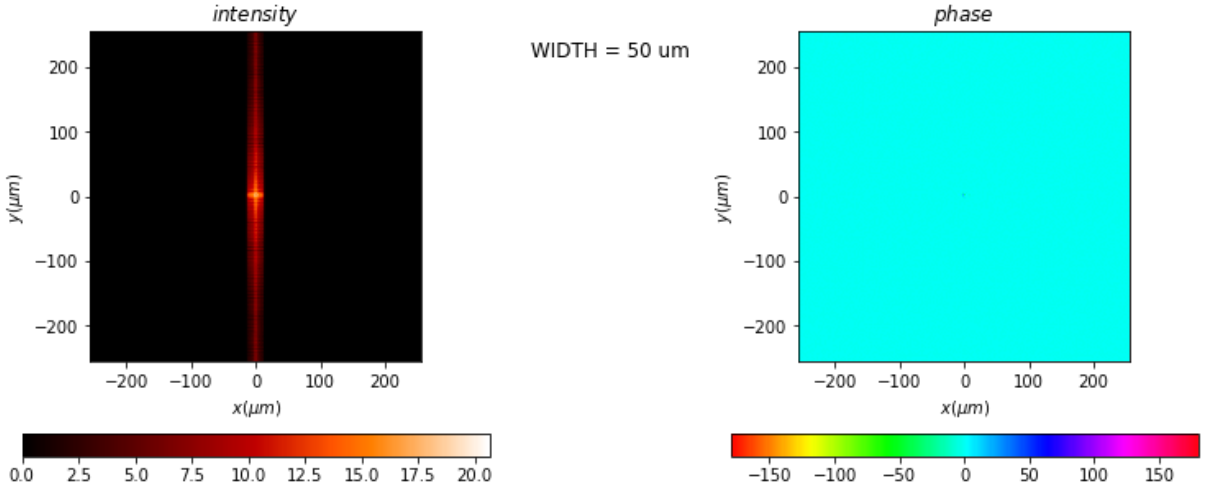


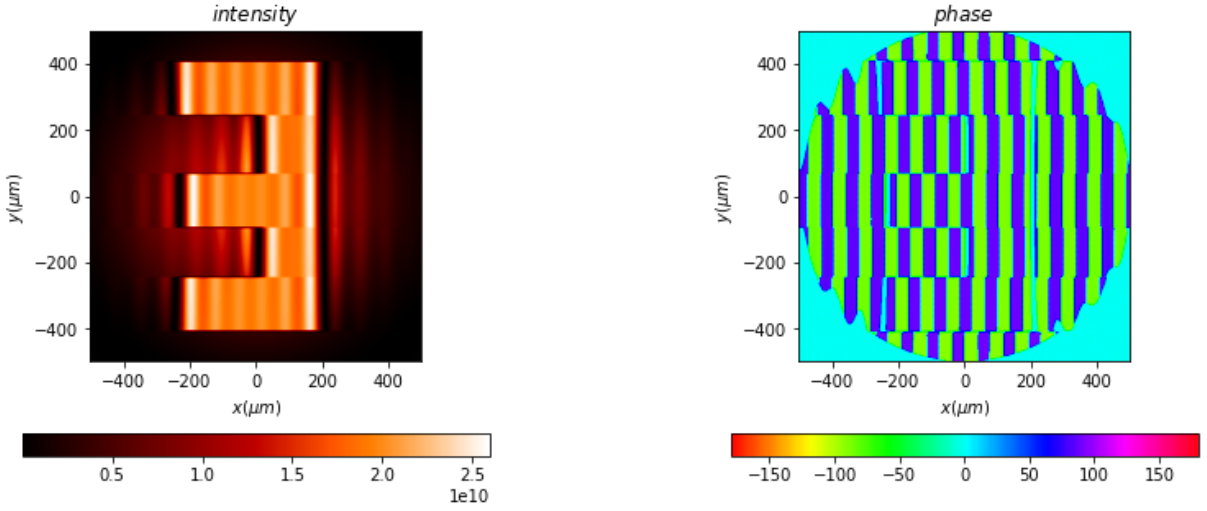


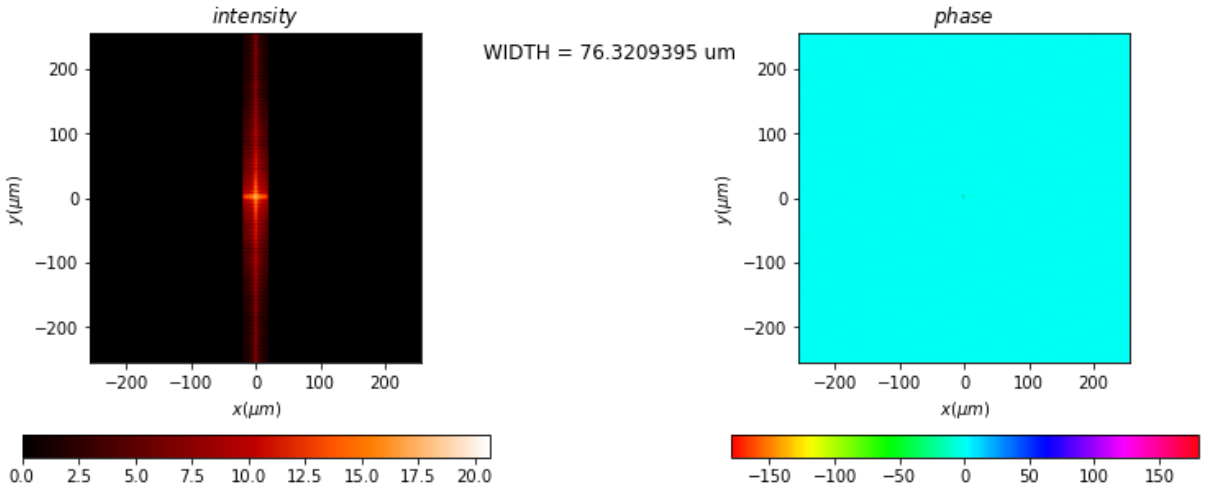


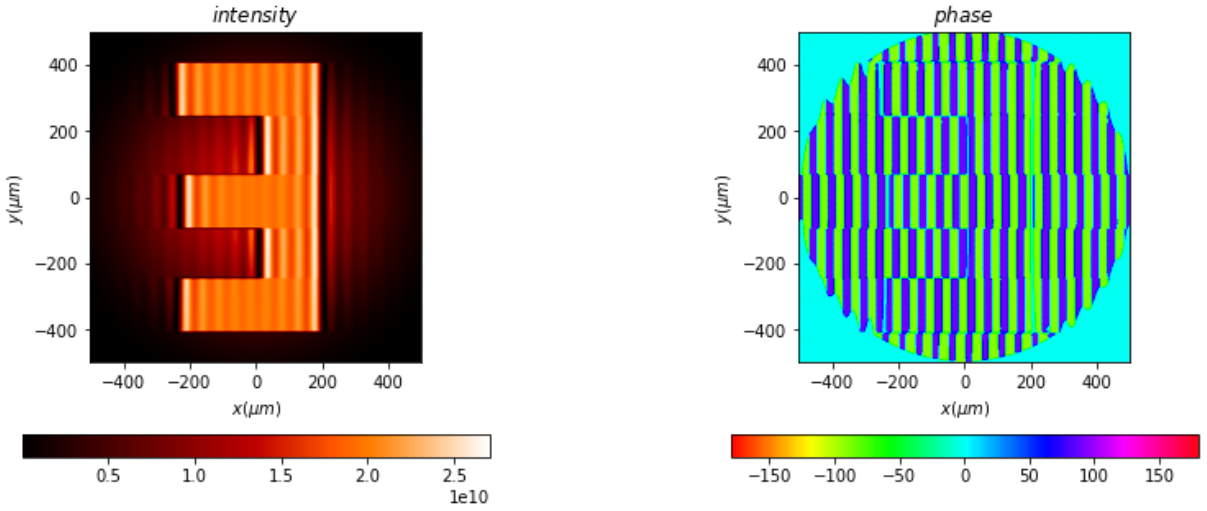

intensity
phase
WIDTH = 30 um
y(μm)
x(μm)
1e10

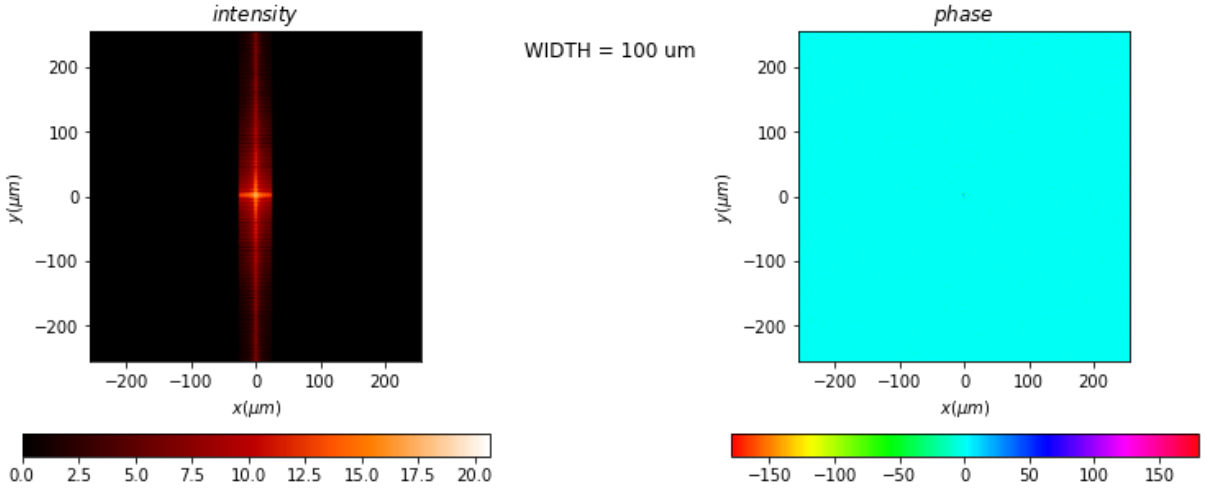

intensity
phase
y(μm)
x(μm)
1e10

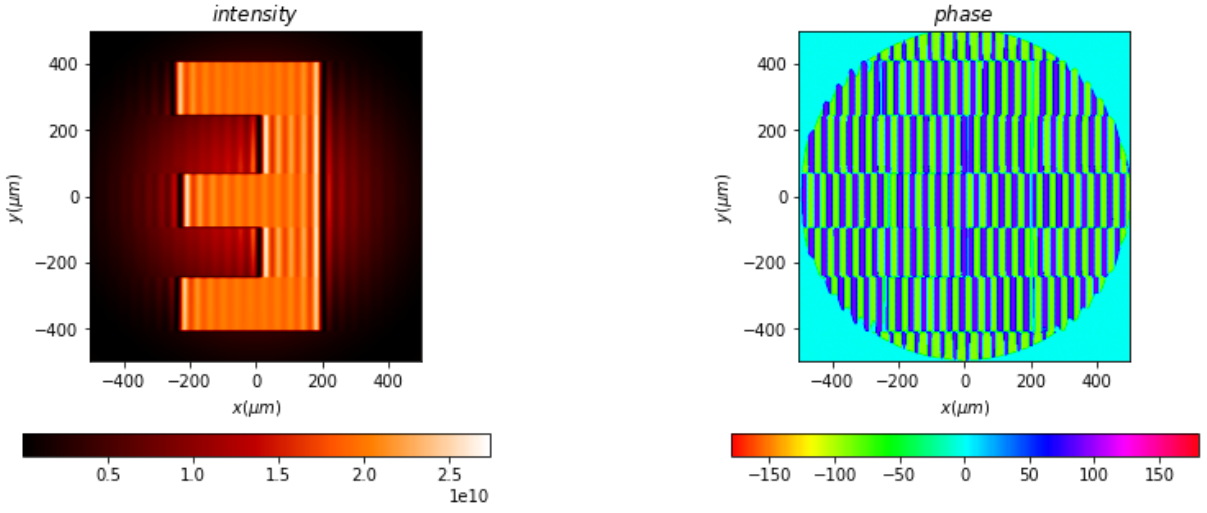

intensity
phase
WIDTH = 50 um
y(μm)
x(μm)
1e10

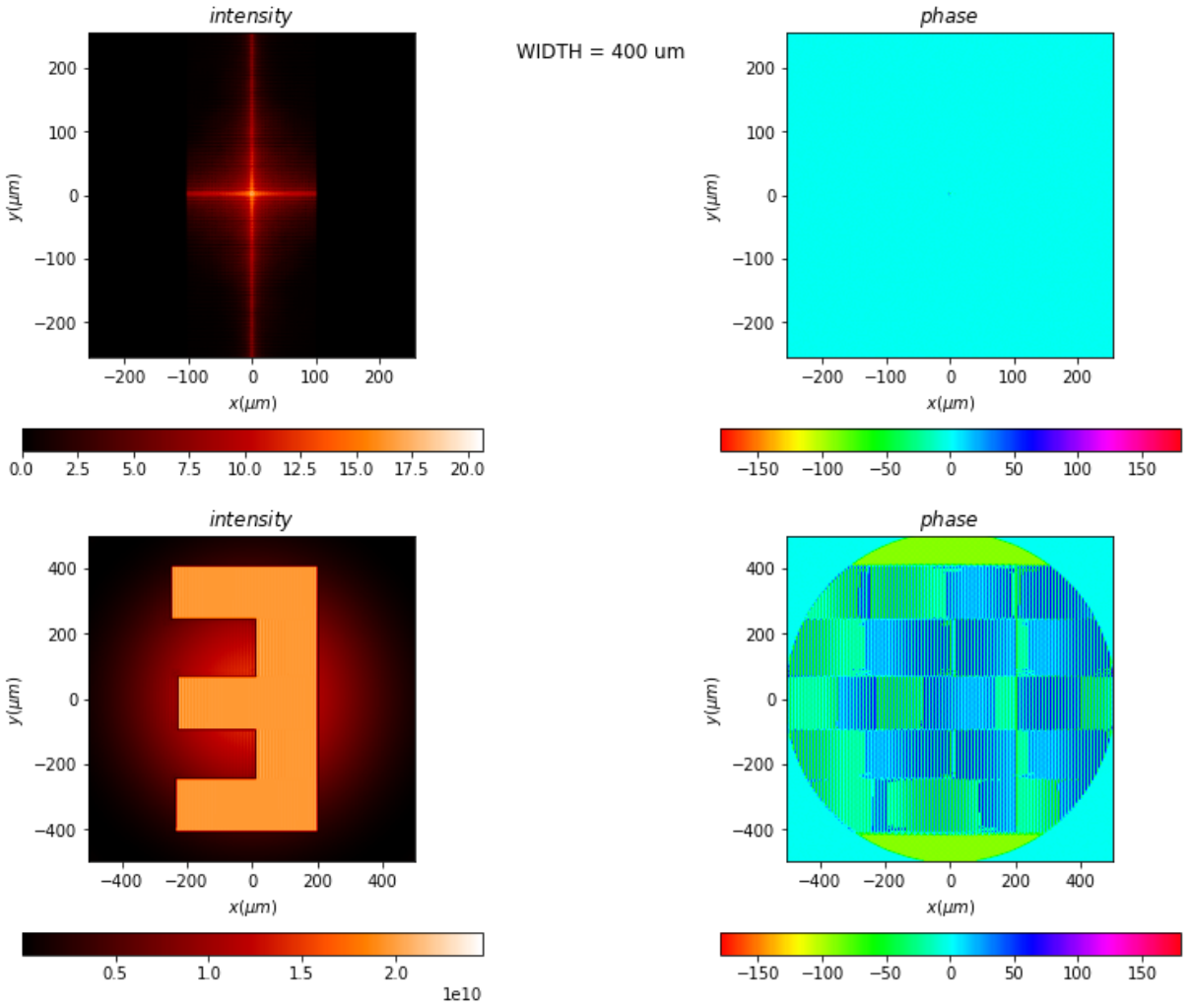

intensity
phase
y(μm)
x(μm)
1e10


intensity
phase
WIDTH = 76.3209395 um
y(μm)
x(μm)
0.0 2.5 5.0 7.5 10.0 12.5 15.0 17.5 20.0
−150 −100 −50 0 50 100 150


intensity
phase
y(μm)
x(μm)
0.5 1.0 1.5 2.0 2.5
1e10
−150 −100 −50 0 50 100 150


intensity
phase
WIDTH = 100 um
y(μm)
x(μm)
0.0 2.5 5.0 7.5 10.0 12.5 15.0 17.5 20.0
−150 −100 −50 0 50 100 150


intensity
phase
y(μm)
x(μm)
0.5 1.0 1.5 2.0 2.5
1e10
−150 −100 −50 0 50 100 150

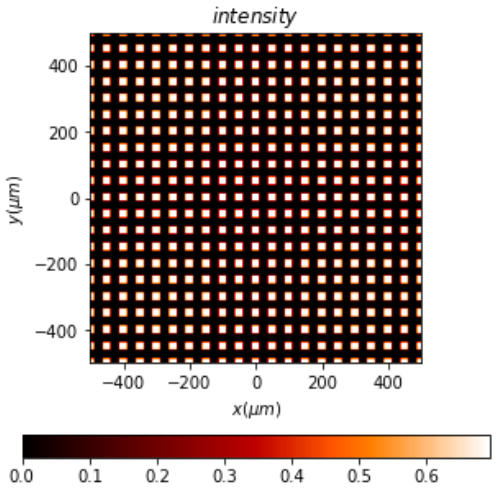


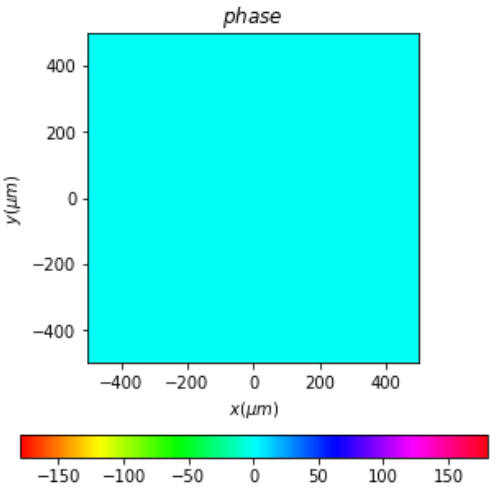
We can observe the "letter being assembled" from various modes, as higher frequencies are allowed to pass through.

Mesh

In [27]:
```
mesh = Scalar_mask_XY(x0, y0, wavelength)
mesh.grating_2D(
    period=50 * um,
    fill_factor=0.5,
    angle=0 * degrees
)

mesh.draw(kind='field', logarithm=True)
```

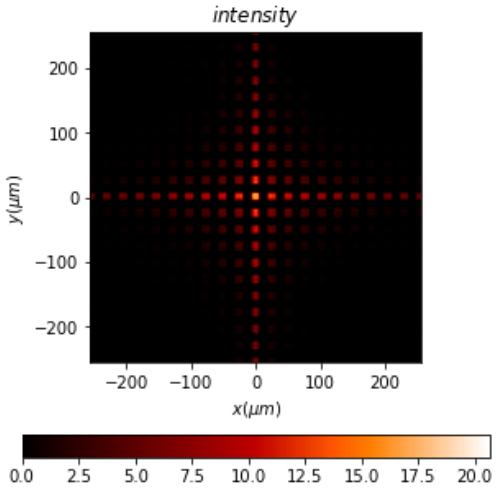
Out[27]:
```
((<matplotlib.image.AxesImage at 0x218e78d8fd0>,
  <matplotlib.image.AxesImage at 0x218e7b4e8b0>),
 None,
 None)
```

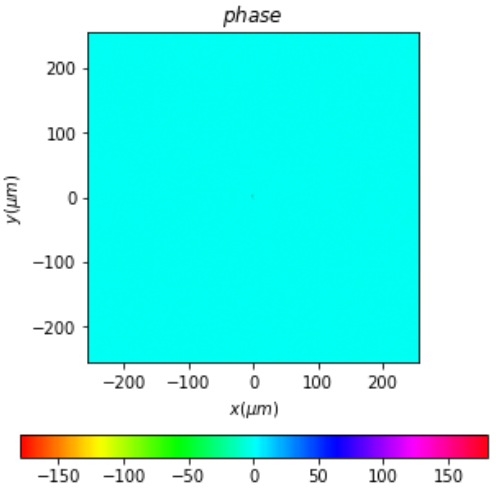


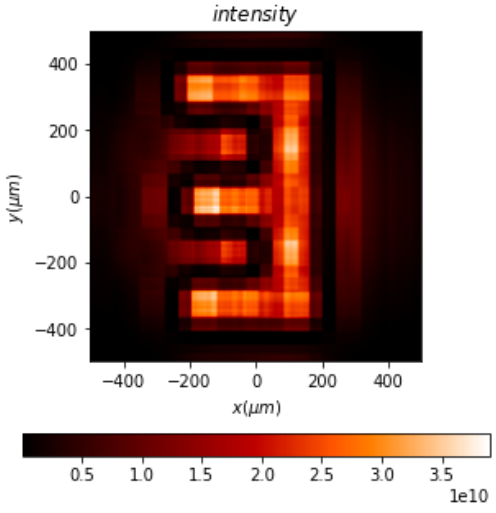


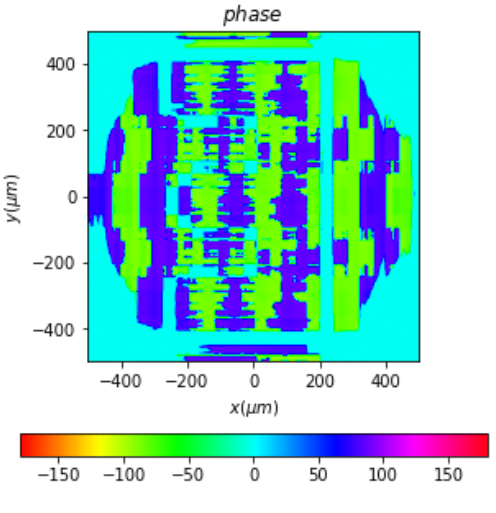
In [28]:
```
E_a_L1_Mesh, E_a_L1_Mesh_L2 = sim(E_a_L1, mesh)
E_a_L1_Mesh.draw(kind='field', logarithm=True)
E_a_L1_Mesh_L2.draw(kind='field', logarithm=False)
```

Out[28]:
```
((<matplotlib.image.AxesImage at 0x218e91c9c70>,
  <matplotlib.image.AxesImage at 0x218e9234ee0>),
 None,
 None)
```









One can obtain similar results with the "F" alphabet-mask.

Phase Constrast Imaging - Using phase information to filter overlapped images

In [1]:
```
import numpy as np
from diffractio import mm, um, degrees
from diffractio.scalar_sources_XY import Scalar_source_XY
from diffractio.scalar_masks_XY import Scalar_mask_XY


# Setting up
length = 1 * mm
num_data = 512
x0 = np.linspace(-length / 2, length / 2, num_data)
y0 = np.linspace(-length / 2, length / 2, num_data)
wavelength = 0.633 * um
```

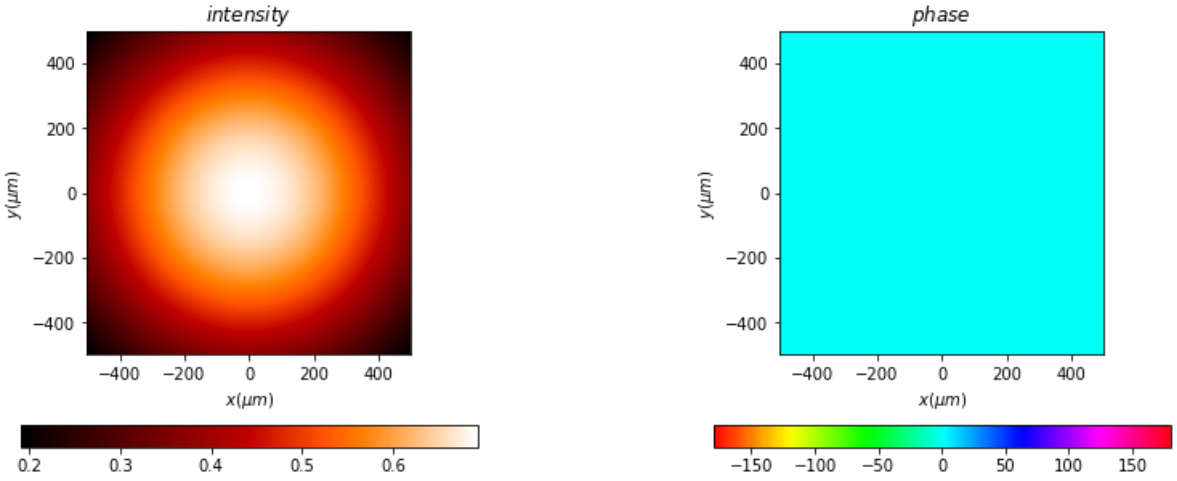
```
number of processors: 12
```

Setting up source

- Gaussian Beam (LASER)

In [2]:
```
# Gaussian Beam Source - like a LASER
u0 = Scalar_source_XY(x=x0, y=y0, wavelength=wavelength)
u0.gauss_beam(r0=(0, 0), w0=(800 * um, 800 * um), z0=0.0)
u0.draw(kind='field', logarithm=True)
```

Out[2]:
```
((<matplotlib.image.AxesImage at 0x14e0a72da60>,
  <matplotlib.image.AxesImage at 0x14e0a9cc220>),
 None,
 None)
```



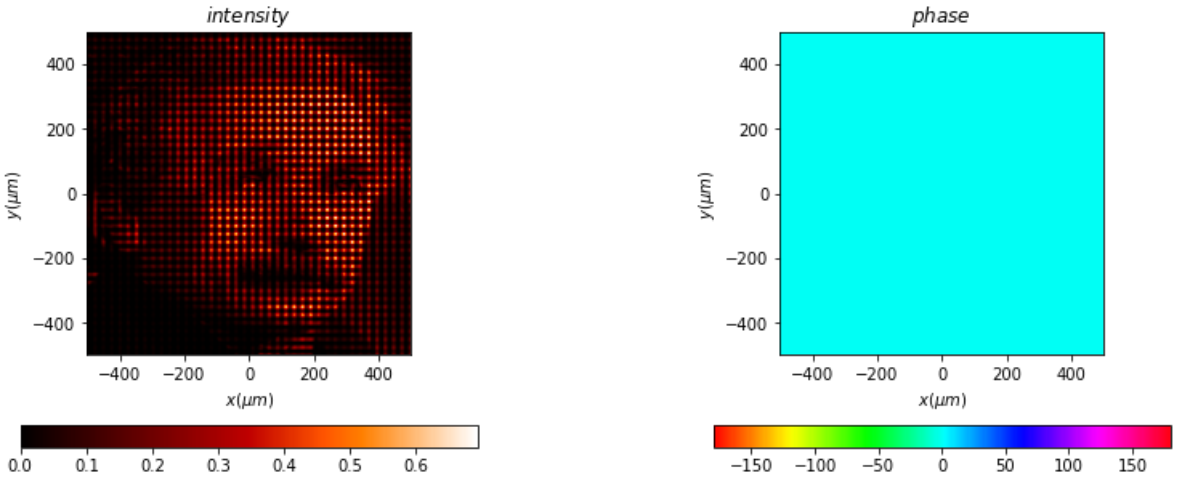
In [3]:
```
# Plane Wave Source - Just an experiment
# u1 = Scalar_source_XY(x=x0, y=y0, wavelength=wavelength)
# u1.plane_wave()
# u1.draw(kind='field', logarithm=True)
```

Image - Phase Contrast

Composition of Horizontal and Vertical Lines

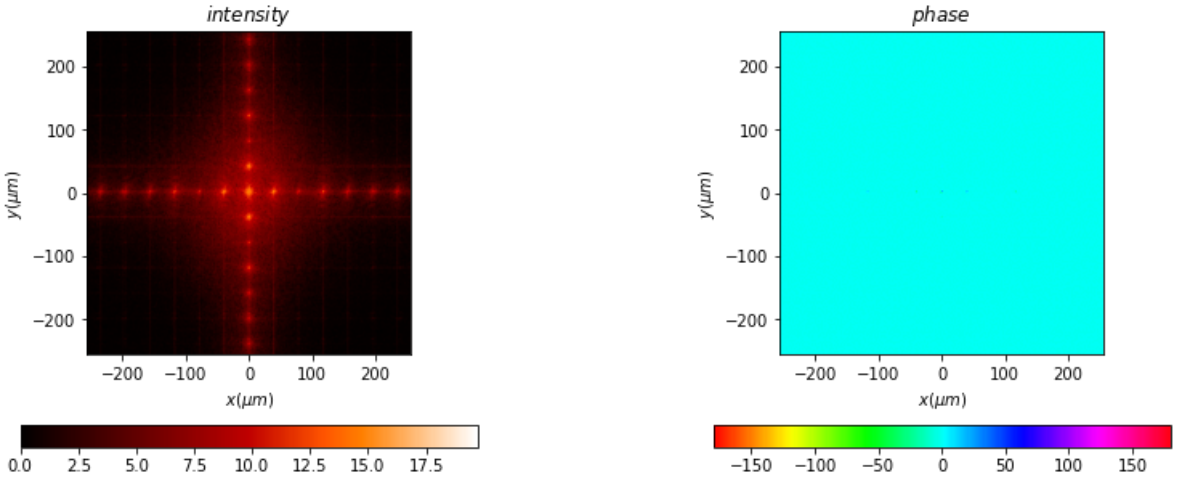
In [4]:
```
compLines = Scalar_mask_XY(x0, y0, wavelength)
compLines.image(
    filename="complines-4.png",
    normalize=True,
    canal=2,
)

compLines.draw(kind='field', logarithm=True)
# compLines.draw(kind='real_field', logarithm=True)
# compLines.draw(kind='amplitude', logarithm=True)
```

Out[4]:
```
((<matplotlib.image.AxesImage at 0x14e0b1f11c0>,
  <matplotlib.image.AxesImage at 0x14e0b452940>),
 None,
 None)
```



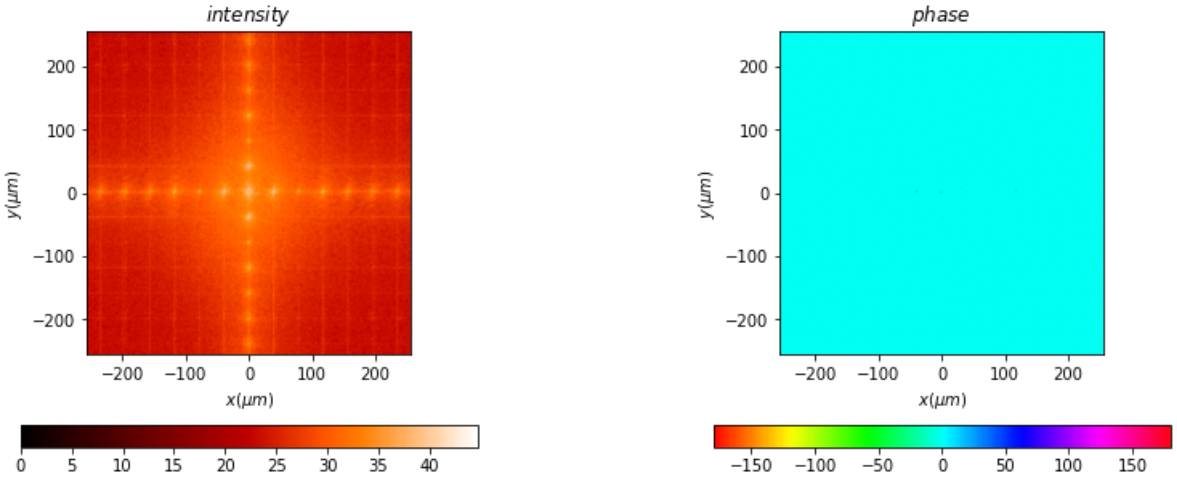
In [5]:
```
a_L1 = (u0 * compLines).fft(z=1 * mm, new_field=True)
a_L1.draw(kind='field', logarithm=True)
# a_L1.draw(kind='amplitude', logarithm=False)
```

Out[5]:
```
((<matplotlib.image.AxesImage at 0x14e0b06c880>,
  <matplotlib.image.AxesImage at 0x14e0b111070>),
 None,
 None)
```



In [6]:
```
a_L2_NF = (a_L1.fft(z=1 * mm, shift=False, remove0=False, new_field=True)).fft(z=1 * mm, shift=False, remove0=
a_L2_NF.draw(kind='field', logarithm=True)
# a_L2_NF.draw(kind='real_field', logarithm=True)
```

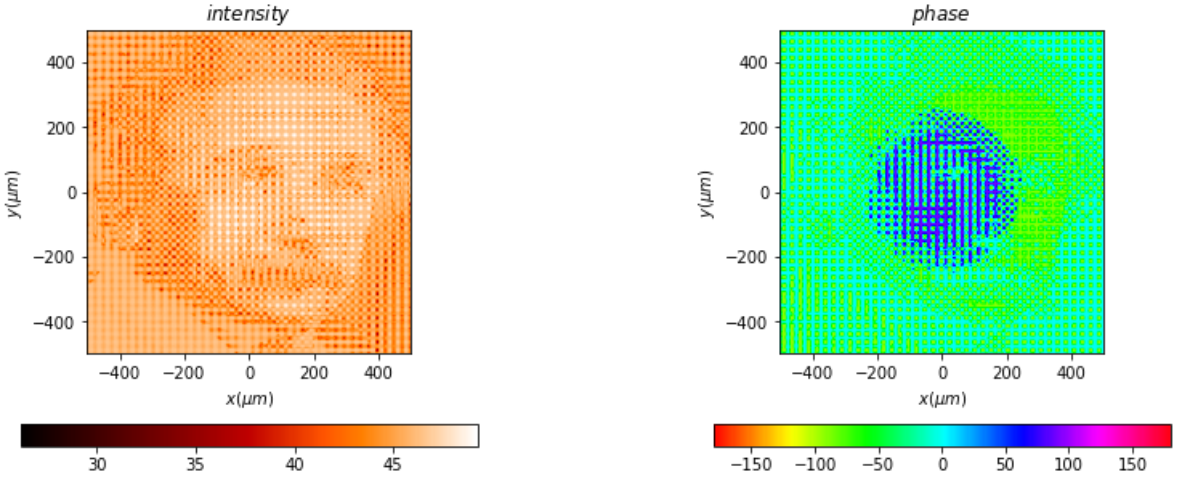
Out[6]:
```
((<matplotlib.image.AxesImage at 0x14e0b8b3c70>,
  <matplotlib.image.AxesImage at 0x14e0fd98460>),
 None,
 None)
```



In [7]:
```
IFFT_NF = a_L2_NF.fft(z=1 * mm, shift=False, remove0=False, new_field=True)
IFFT_NF.draw(kind='field', logarithm=True)
```

Out[7]:
```
((<matplotlib.image.AxesImage at 0x14e0fe49fd0>,
```

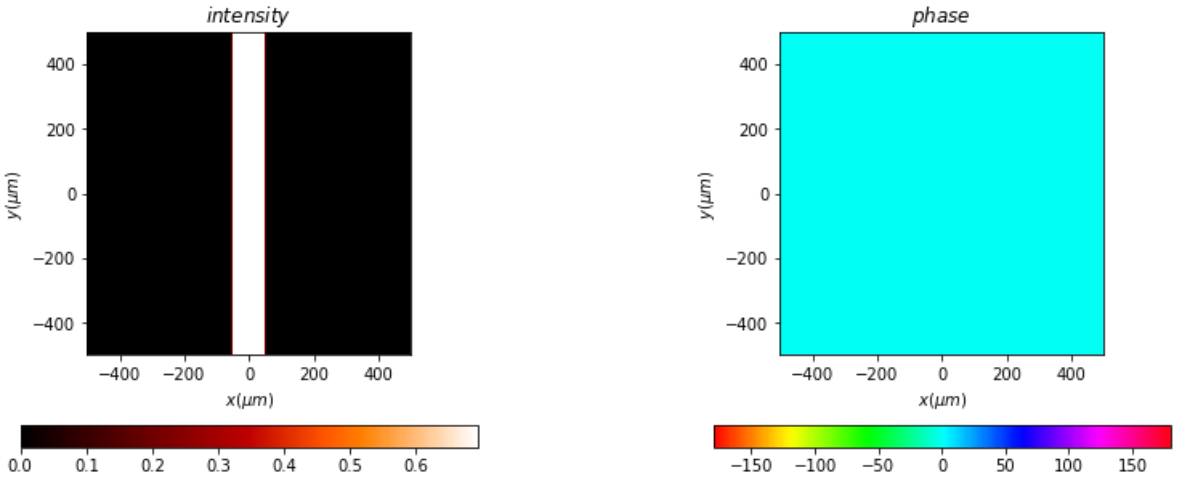
```
 <matplotlib.image.AxesImage at 0x14e10712790>),
None,
None)
```



In [8]:
```
csize = 100  # Size of slit
```

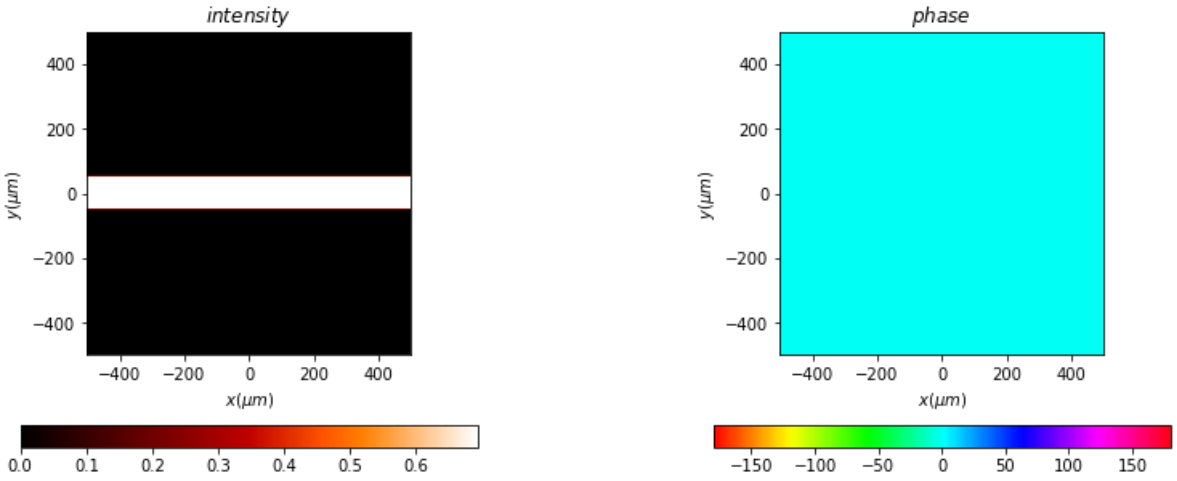
In [9]:
```
vertSlit = Scalar_mask_XY(x0, y0, wavelength)
vertSlit.slit(
    x0=0 * um,
    size=csize * um
)
vertSlit.draw(kind='field', logarithm=True)
```

Out[9]:
```
((<matplotlib.image.AxesImage at 0x14e0b45e340>,
  <matplotlib.image.AxesImage at 0x14e0b0aa970>),
None,
None)
```



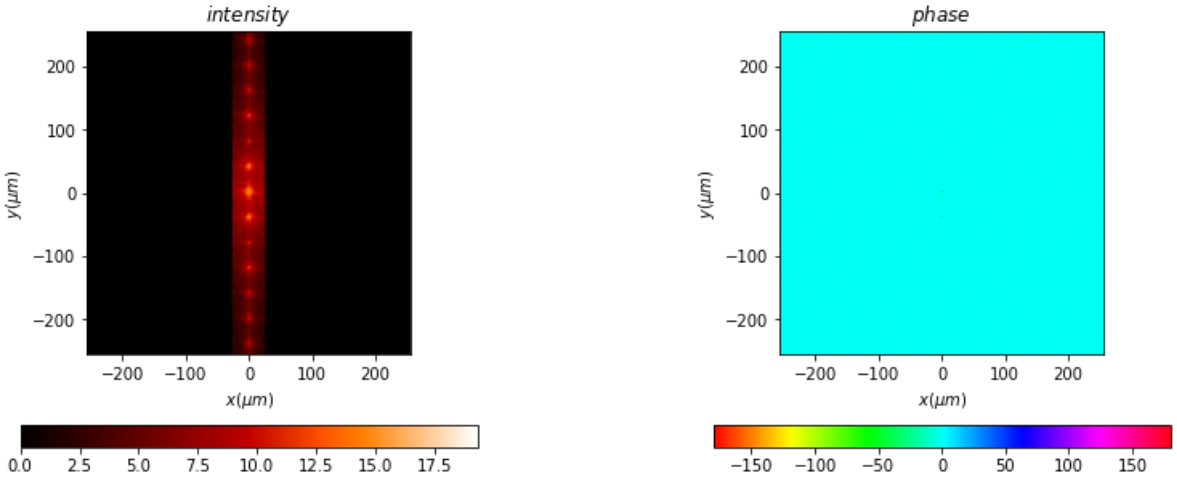
In [10]:
```
horizSlit = Scalar_mask_XY(x0, y0, wavelength)
horizSlit.slit(
    x0=0 * um,
    size=csize * um,
    angle=np.pi / 2
)
horizSlit.draw(kind='field', logarithm=True)
```

Out[10]:
```
((<matplotlib.image.AxesImage at 0x14e112b4eb0>,
  <matplotlib.image.AxesImage at 0x14e1136b670>),
None,
None)
```



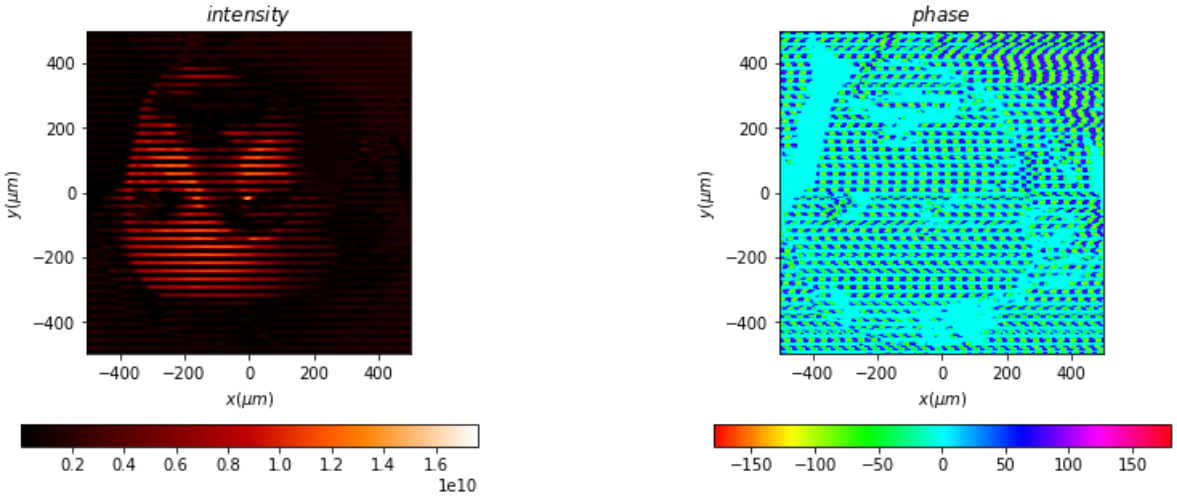
In [11]:
```
b_L2_a_vS = a_L1 * vertSlit
b_L2_a_vS.draw(kind='field', logarithm=True)
# b_L2_a_vS.draw(kind='amplitude', logarithm=True)
```

Out[11]:
```
((<matplotlib.image.AxesImage at 0x14e12921490>,
  <matplotlib.image.AxesImage at 0x14e12983c40>),
 None,
 None)
```



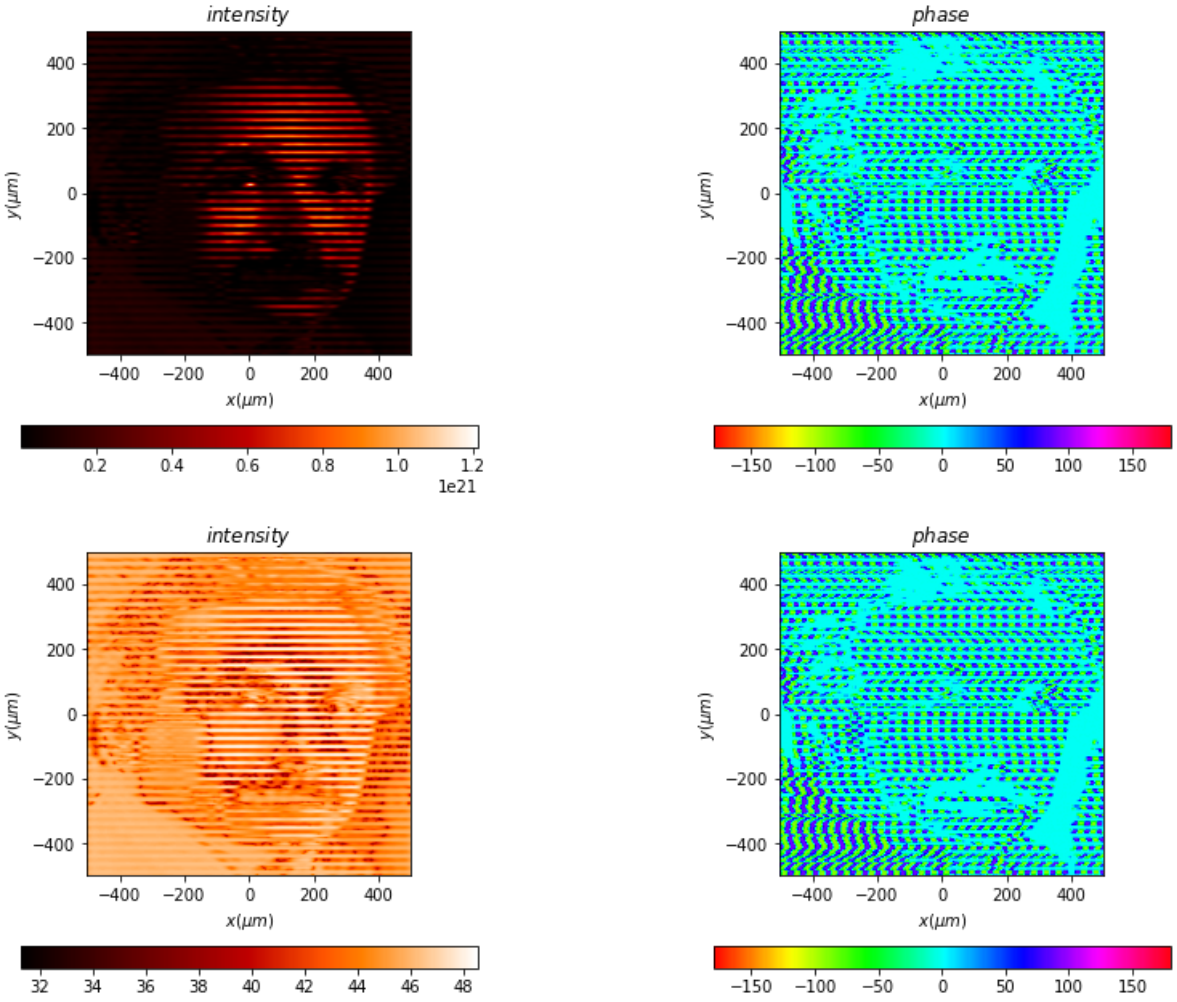
In [12]:
```
a_L2_vS = b_L2_a_vS.fft(z=1 * mm, shift=False, remove0=False, new_field=True)
a_L2_vS.draw(kind='field', logarithm=False)
# a_L2_vS.draw(kind='intensity', logarithm=True)
```

Out[12]:
```
((<matplotlib.image.AxesImage at 0x14e12a45760>,
  <matplotlib.image.AxesImage at 0x14e1331b7c0>),
 None,
 None)
```



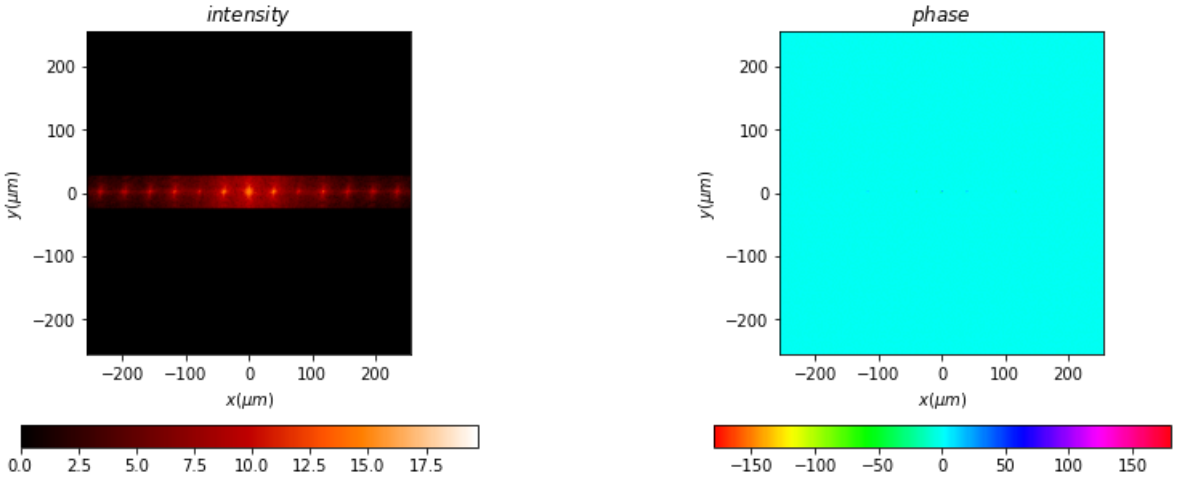
In [13]:
```
IFFT_vS = (a_L2_vS.fft(z=1 * mm, shift=False, remove0=False, new_field=True)).fft(z=1 * mm, shift=False, remov
IFFT_vS.draw(kind='field', logarithm=False)
IFFT_vS.draw(kind='field', logarithm=True)
```

Out[13]:
```
((<matplotlib.image.AxesImage at 0x14e0b0c38e0>,
  <matplotlib.image.AxesImage at 0x14e128b60d0>),
 None,
 None)
```



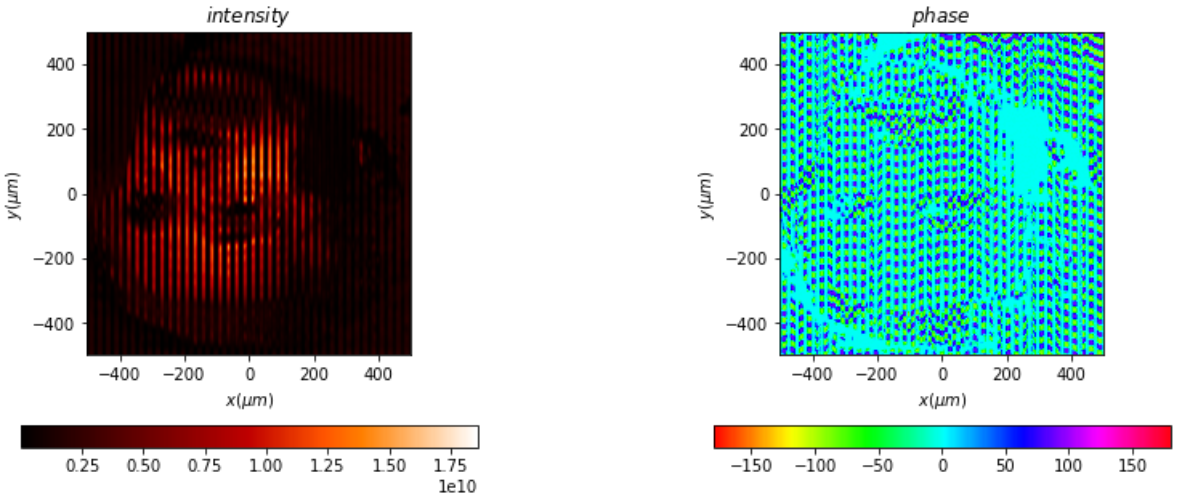
In [14]:
```
b_L2_a_hS = a_L1 * horizSlit
b_L2_a_hS.draw(kind='field', logarithm=True)
# b_L2_a_hS.draw(kind='amplitude', logarithm=True)
```

Out[14]:
```
((<matplotlib.image.AxesImage at 0x14e14874a00>,
  <matplotlib.image.AxesImage at 0x14e148ff220>),
 None,
 None)
```



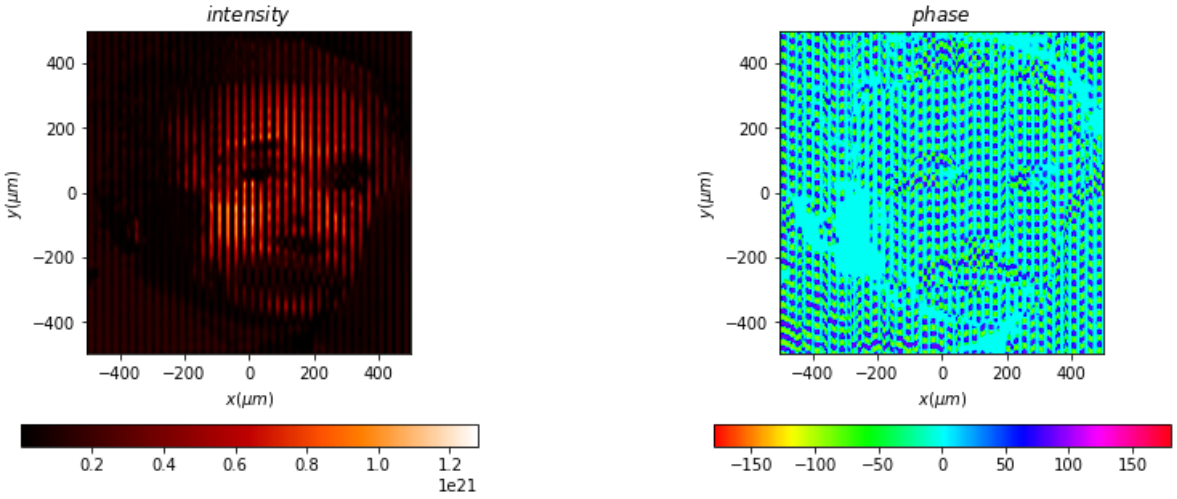
In [15]:
```
a_L2_hS = b_L2_a_hS.fft(z=1 * mm, shift=False, remove0=False, new_field=True)
a_L2_hS.draw(kind='field', logarithm=False)
# a_L2_hS.draw(kind='intensity', logarithm=True)
```

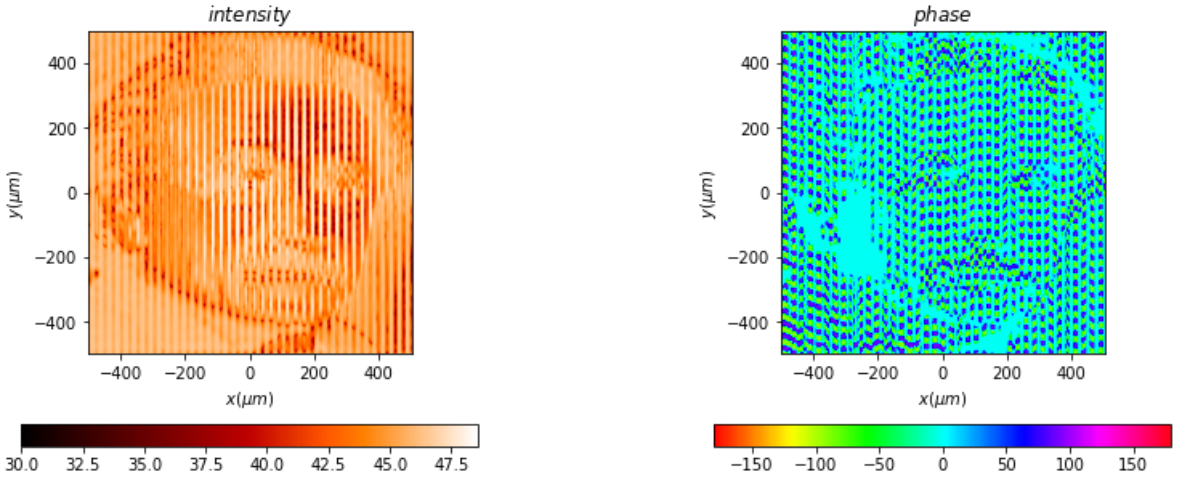
Out[15]:
```
((<matplotlib.image.AxesImage at 0x14e16c12c40>,
  <matplotlib.image.AxesImage at 0x14e16c98400>),
 None,
 None)
```



In [16]:
```
IFFT_hS = (a_L2_hS.fft(z=1 * mm, shift=False, remove0=False, new_field=True)).fft(z=1 * mm, shift=False, remov
IFFT_hS.draw(kind='field', logarithm=False)
IFFT_hS.draw(kind='field', logarithm=True)
```

Out[16]:
```
((<matplotlib.image.AxesImage at 0x14e182461f0>,
  <matplotlib.image.AxesImage at 0x14e182b04f0>),
 None,
 None)
```





In [ ]: