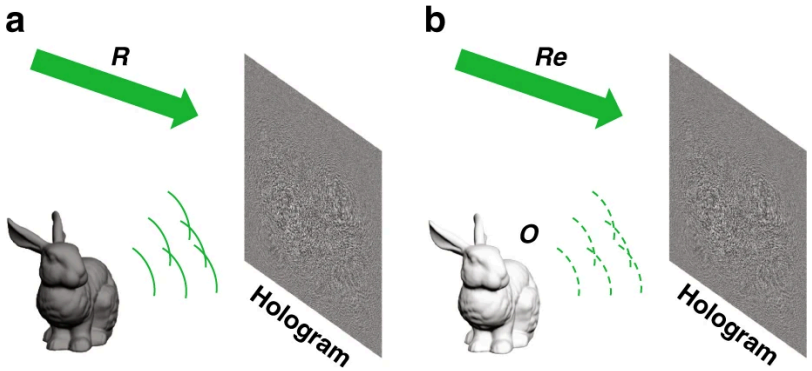
Lab 4

1. Abstract

This lab explores the fundamentals of holography, focusing on the recording and reconstruction of three-dimensional scenes through interference and diffraction principles. By analyzing interference patterns between object and reference waves, holography can capture and later reconstruct the amplitude and phase information of complex wavefronts. The experiments examine different interference scenarios, including plane waves, spherical waves, and vortex waves, demonstrating how these interactions are recorded and subsequently analyzed for image reconstruction. Techniques such as Fourier transform filtering and phase unwrapping are utilized to handle the spatial frequency content and reconstruct the original object wave. Additionally, this study introduces the Sparsity of Gradient (SoG) criterion for holographic autofocusing, using edge sparsity metrics to identify optimal focus planes. The methods demonstrated in this lab lay a foundation for understanding digital holography and its applications in capturing fine details of 3D structures.`

2. Holography

Holography is a technique to record and reconstruct all the physical information of a 3D scene based on interference and diffraction theory. In recording process, the object light O and the reference light R interfere in the hologram plane and the interference fringe I is recorded on the hologram. In reconstruction process, the hologram is illuminated by the reconstruction light Re.



$$I = |O + R|^2 = |O|^2 + |R|^2 + OR^* + O^*R \tag{1}$$

$$U = |O|^2 R + |R|^2 R + O|R|^2 + O^* R^2 \tag{2}$$

If the reconstruction light Re is chosen as the reference light R, the third term of the diffraction light U is, up to a multiplicative constant, an exact duplication of the original object light O, which indicates that the original object can be reconstructed successfully.

In the following section, we see the interference of different waves, and reconstruction of the field.

1. Two plane waves
2. Plane wave + Spherical wave
3. Plane wave + Vortex
4. Spherical wave + Vortex

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import imageio.v3 as io
from skimage import color
from skimage.transform import rescale, resize, downscale_local_mean
```

In [2]:
```
# Helper functions
def ft2(x):
    return np.fft.fftshift(np.fft.fft2(np.fft.ifftshift(x))) # Fourier transform

def ift2(x):
    return np.fft.fftshift(np.fft.ifft2(np.fft.ifftshift(x))) # Inverse Fourier transform
```

In [3]:
```
N = 512 # Grid size
p = 5e-6 # Pixel size

x0 = np.linspace(-N/2,  N/2, N, endpoint = True)
x, y = np.meshgrid(x0, x0)
x = x*p
y = y*p

f0 = x0/N
fx, fy = np.meshgrid(f0, f0)
```

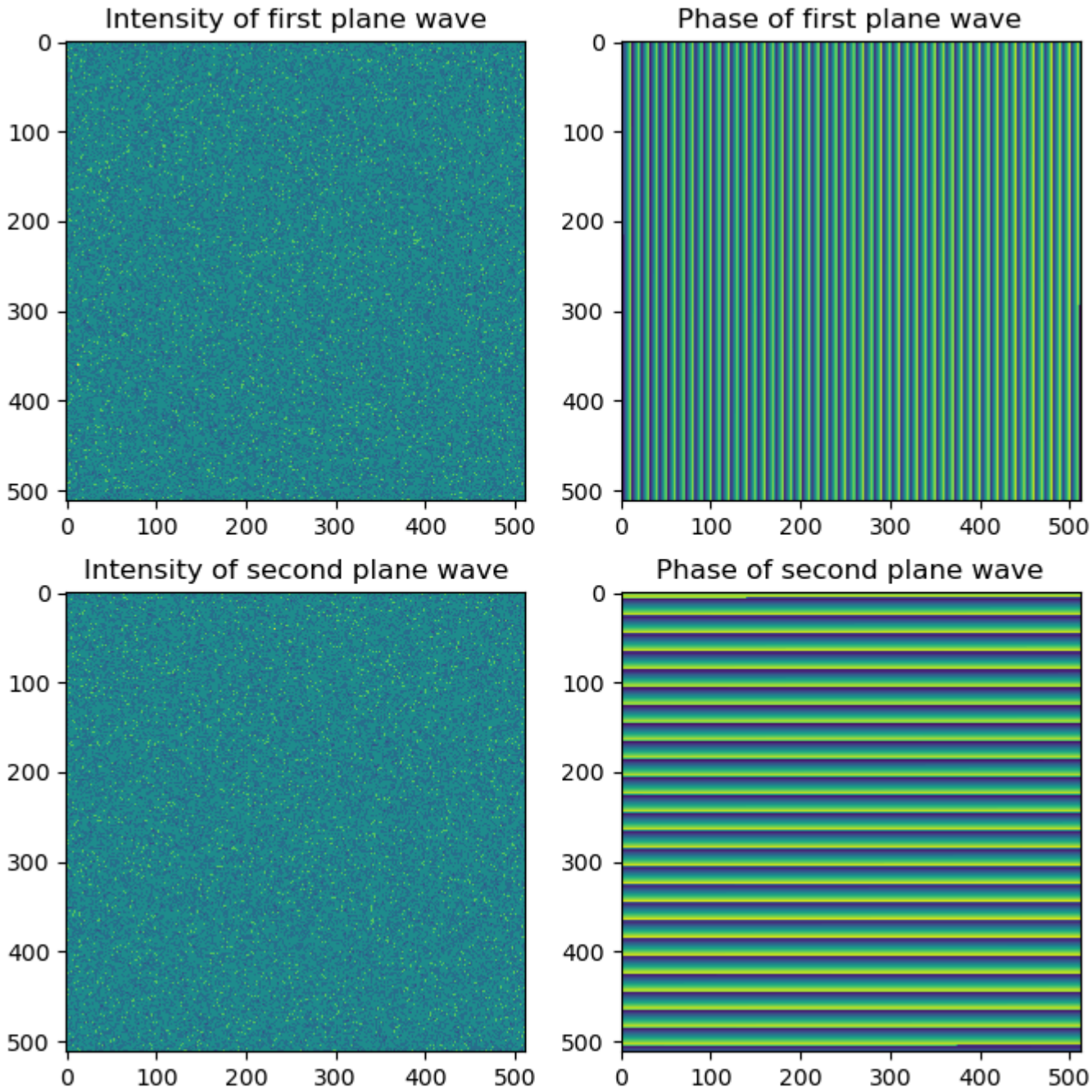
```
fx = fx/p
fy = fy/p

wavelength = 0.5e-6
k = 2*np.pi/wavelength

alpha = np.sqrt(k**2 - 4*np.pi**2*(fx**2 + fy**2))

# Define a function for propagation of field
def propagation(field, z):
    # Low pass filter
    lp_f = (1/wavelength) * 1 / (np.sqrt(1 + (2 * z / (N * p))**2))
    filtr = np.zeros((N, N))
    filtr[(fx**2 + fy**2 < lp_f**2)] = 1
    # Propagation function
    H = np.exp(1j*alpha*z)
    field_fft = ft2(field)
    field_prop = field_fft*H*filtr
    return ift2(field_prop)
```

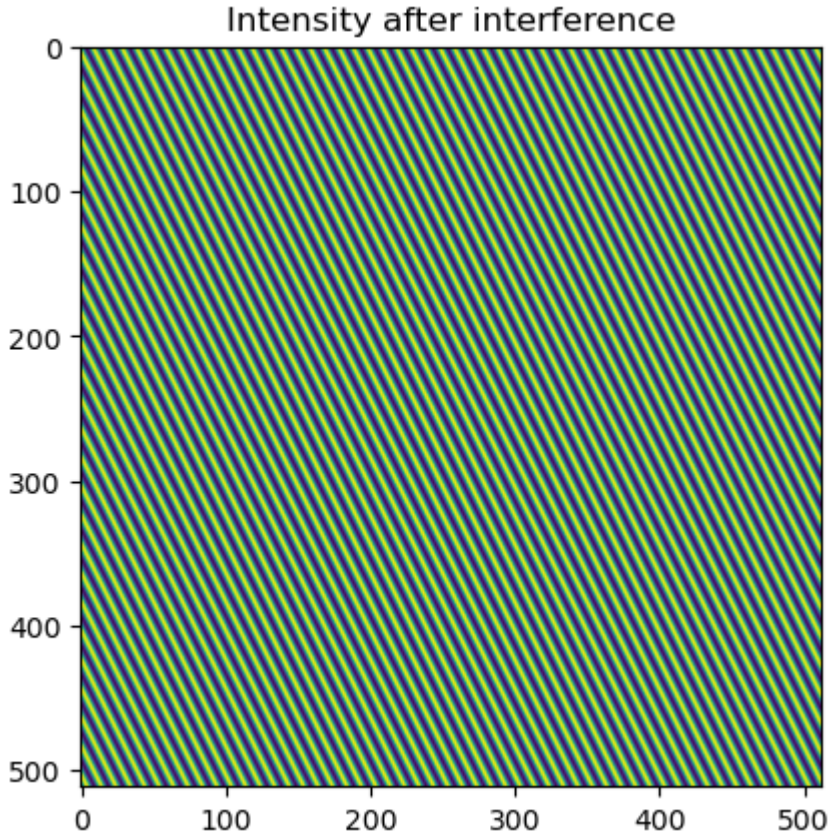
2.1. Plane wave + Plane wave

Interference of two plane waves

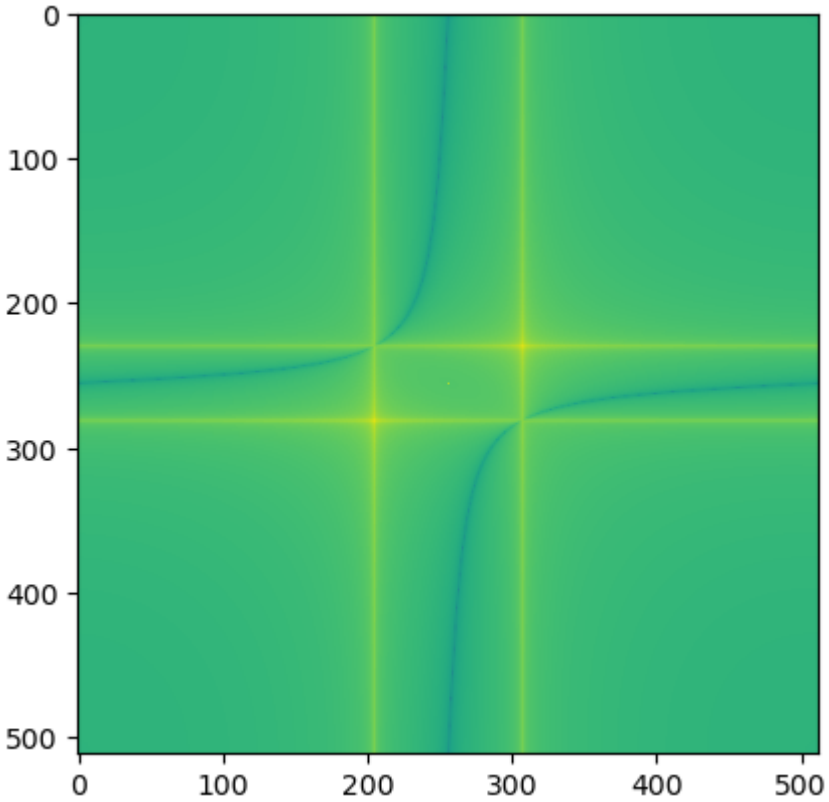
In [4]:
```
fx1, fy1 = 20000, 1
E1 = np.exp(1j*2*np.pi*(x*fx1 + y*fy1)) # Plane wave 1
I1 = np.abs(E1)**2

fx2, fy2 = 1, 10000
E2 = np.exp(1j*2*np.pi*(x*fx2 + y*fy2)) # Plane wave 2

E12 = E1 + E2 # Interference
I12 = np.abs(E12)**2

plt.figure(figsize = (8,8))
plt.subplot(2,2,1)
plt.imshow(np.abs(E1)**2)
plt.title("Intensity of first plane wave")
```

```
plt.subplot(2,2,2)
plt.imshow(np.angle(E1))
plt.title("Phase of first plane wave")

plt.subplot(2,2,3)
plt.imshow(np.abs(E2)**2)
plt.title("Intensity of second plane wave")

plt.subplot(2,2,4)
plt.imshow(np.angle(E2))
plt.title("Phase of second plane wave")

plt.show()

plt.figure()
plt.imshow(I12)
plt.title("Intensity after interference")
plt.show()
```


Intensity of first plane wave
Phase of first plane wave
Intensity of second plane wave
Phase of second plane wave
0
100
200
300
400
500



In [5]:
```
i12 = ft2(I12)
plt.figure()
plt.imshow(np.log(np.abs(i12 +1e-10)))
plt.show()
```

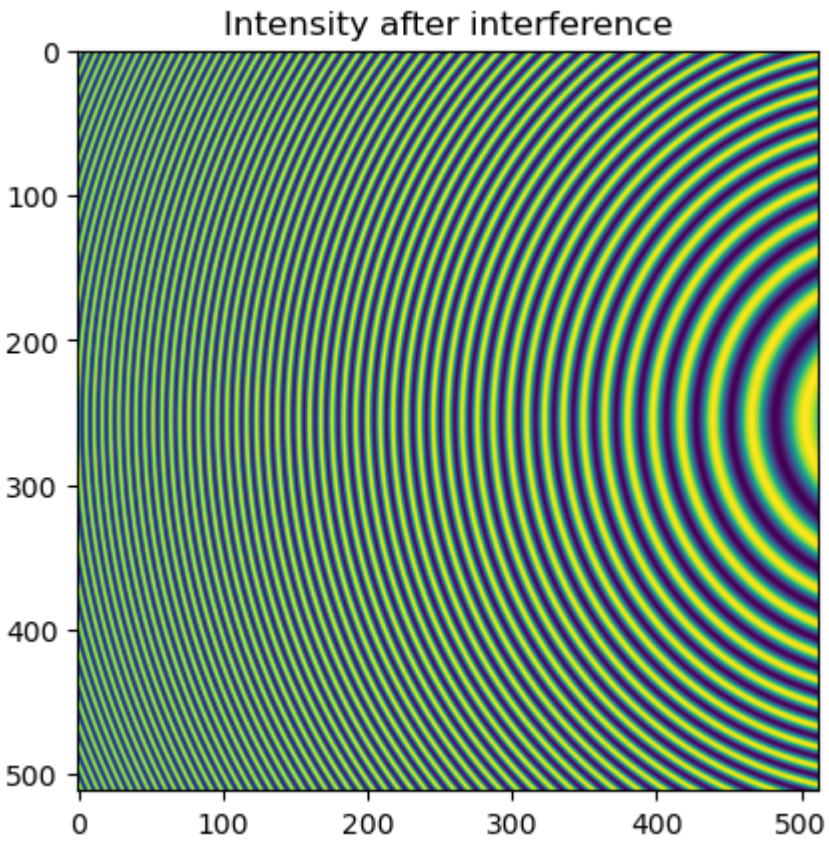


2.2. Plane wave + Spherical wave

Interference of plane wave and spherical wave using plane wave as a reference wave.

In [6]:
```
alpha = 40e6
E3 = np.exp(1j*alpha*(x**2 + y**2)) # Spherical wave
I3 = np.abs(E3)**2

E13 = E1 + E3 # Interference with plane wave
I13 = np.abs(E13)**2

plt.figure()
plt.imshow(np.abs(E13)**2)
```

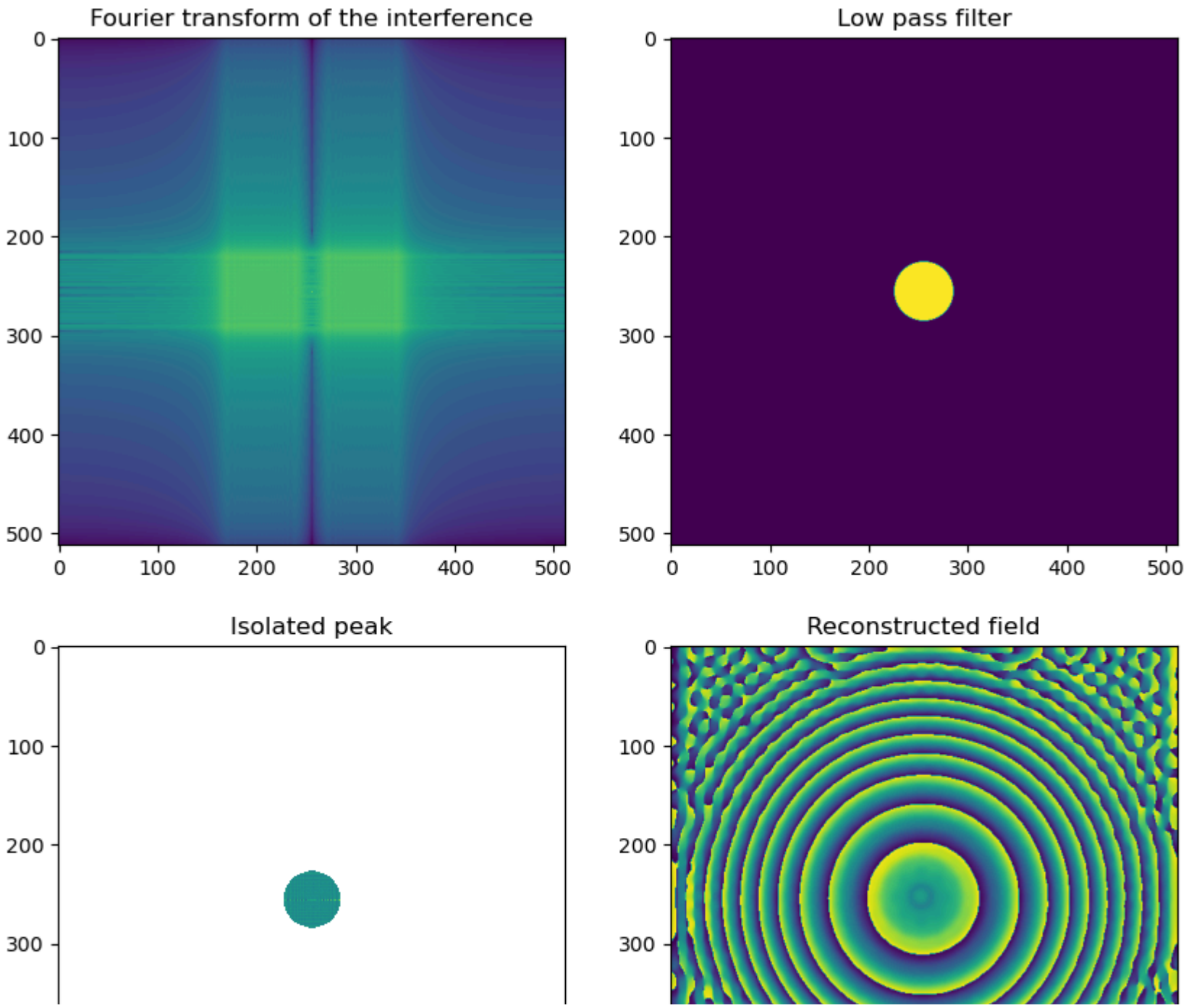
```
plt.title("Intensity after interference")
plt.show()
```

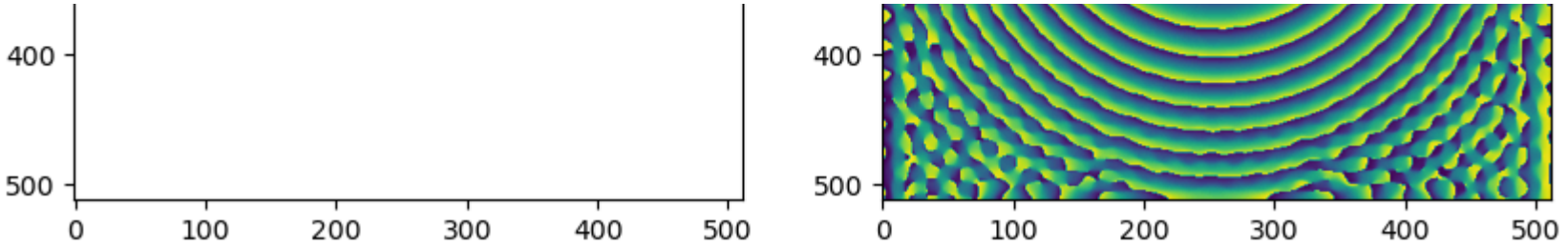


In [7]:
```
i13 = ft2(I13)
I113 = E1 * I13
i113 = ft2(I113)

low = np.zeros((N,N))
low[x**2 + y**2 < (30*p)**2] = 1

i = i113*low
I = ift2(i)

plt.figure(figsize = (10,10))
```

```
plt.subplot(2,2,1)
plt.imshow(np.log(np.abs(i13)))
plt.title("Fourier transform of the interference")
plt.subplot(2,2,2)
plt.imshow((np.abs(low)))
plt.title("Low pass filter")
plt.subplot(2,2,3)
plt.imshow(np.log(np.abs(i)))
plt.title("Isolated peak")
plt.subplot(2,2,4)
plt.imshow(np.angle(I))
plt.title("Reconstructed field")
plt.show()
```

```
C:\Users\nitin negi\AppData\Local\Temp\ipykernel_13152\2144613613.py:19: RuntimeWarning: divide by zero encountered in log
  plt.imshow(np.log(np.abs(i)))
```

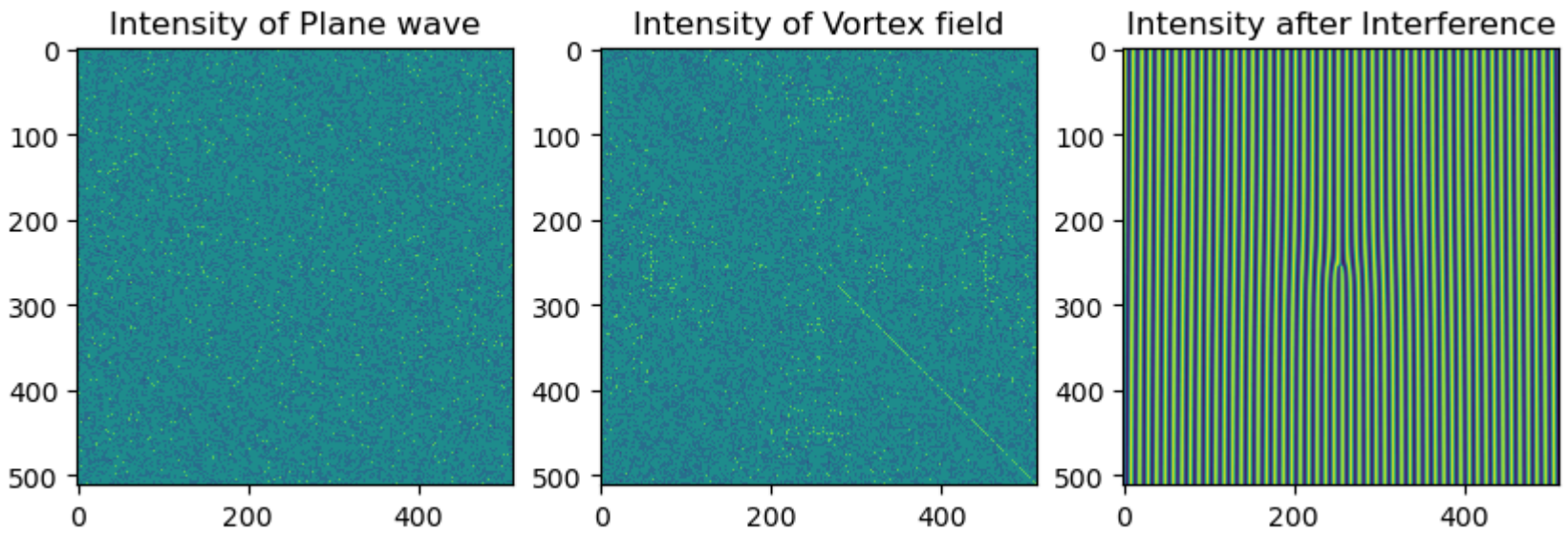

Fourier transform of the interference
Low pass filter
Isolated peak
Reconstructed field
0
100
200
300
400
500



2.3. Plane Wave + Vortex

Interference of plane wave and vortex wave with plane wave as the reference.

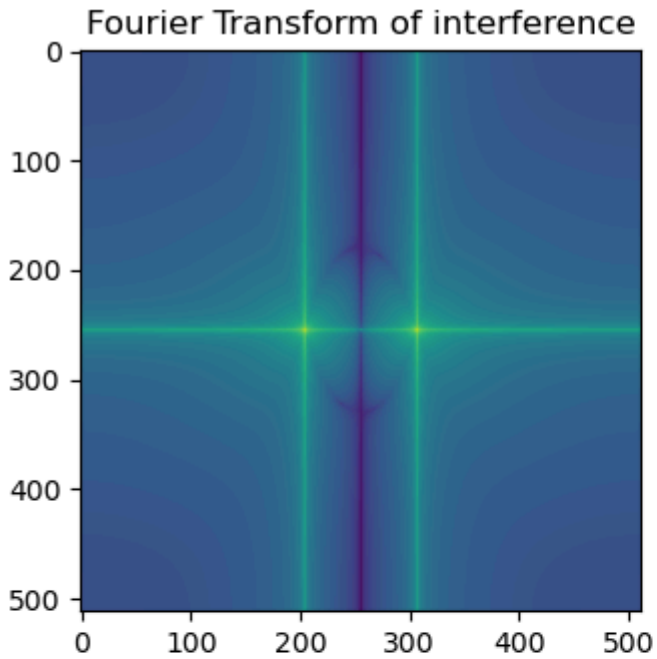
In [8]:
```
theta = np.arctan2(y, x)
E4 = np.exp(1j *theta) # Vortex wave
E14 = E1 + E4 # Interference
I4 = np.abs(E4)**2
I14 =  np.abs(E14)**2

plt.figure(figsize = (10, 8))
plt.subplot(1,3,1)
plt.imshow(I1)
plt.title("Intensity of Plane wave")

plt.subplot(1,3,2)
plt.imshow(I4)
plt.title("Intensity of Vortex field")

plt.subplot(1,3,3)
plt.imshow(I14)
plt.title("Intensity after Interference")

plt.show()
```

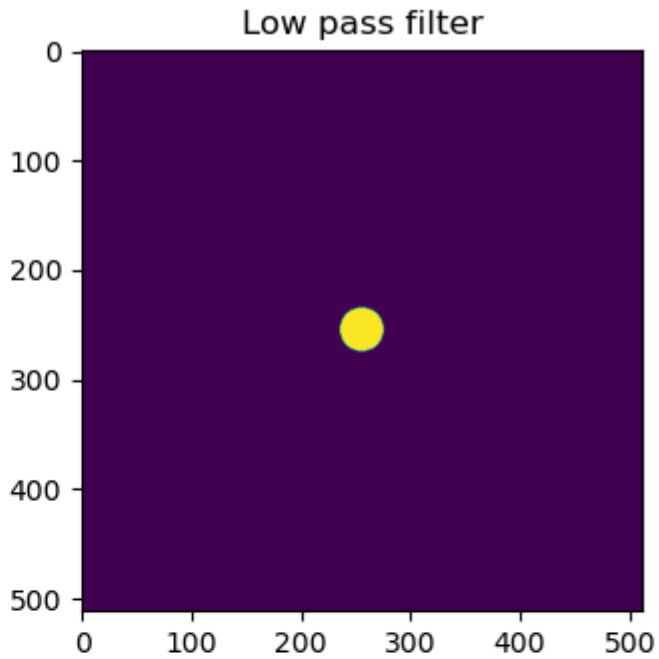


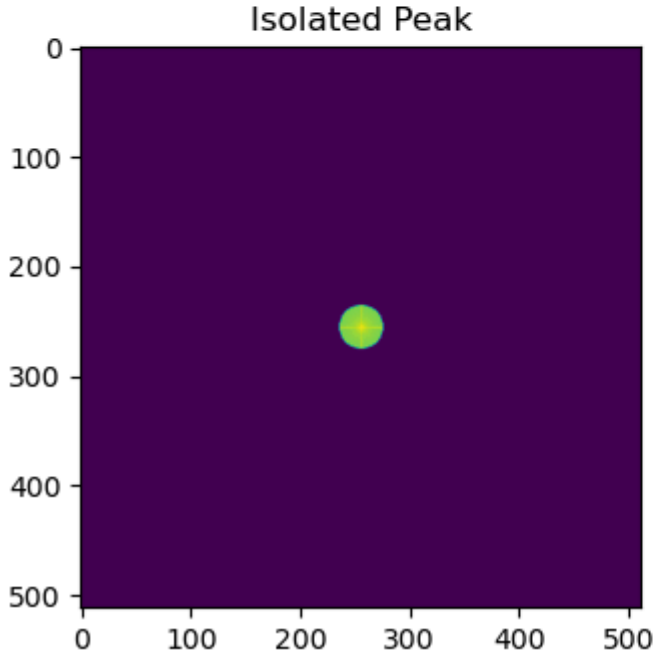
In [9]:

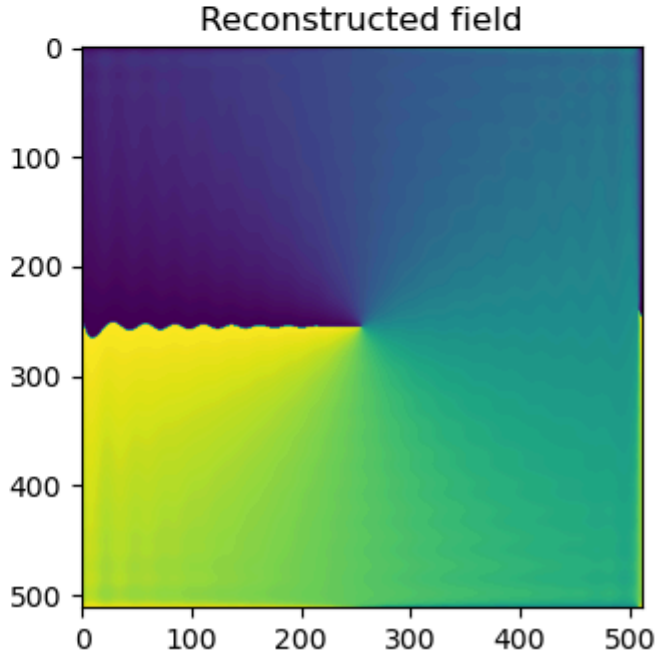
```
i14 = ft2(I14)
I114 = E1 * I14
i114 = ft2(I114)

low = np.zeros((N,N))
low[x**2 + y**2 <= (20*p)**2] = 1

i = i114 * low
I = ift2(i)

plt.figure(figsize = (10, 8))

plt.subplot(2,2,1)
plt.imshow(np.log(np.abs(i14)))
plt.title("Fourier Transform of interference")

plt.subplot(2,2,2)
plt.imshow(np.log(np.abs(low) + 1e-10))
plt.title("Low pass filter")

plt.subplot(2,2,3)
plt.imshow(np.log(np.abs(i) + 1e-10))
```

```
plt.title("Isolated Peak")

plt.subplot(2,2,4)
plt.imshow(np.angle(I))
plt.title('Reconstructed field')

plt.show()
```


Fourier Transform of interference
0
100
200
300
400
500
0
100
200
300
400
500


Low pass filter
0
100
200
300
400
500
0
100
200
300
400
500

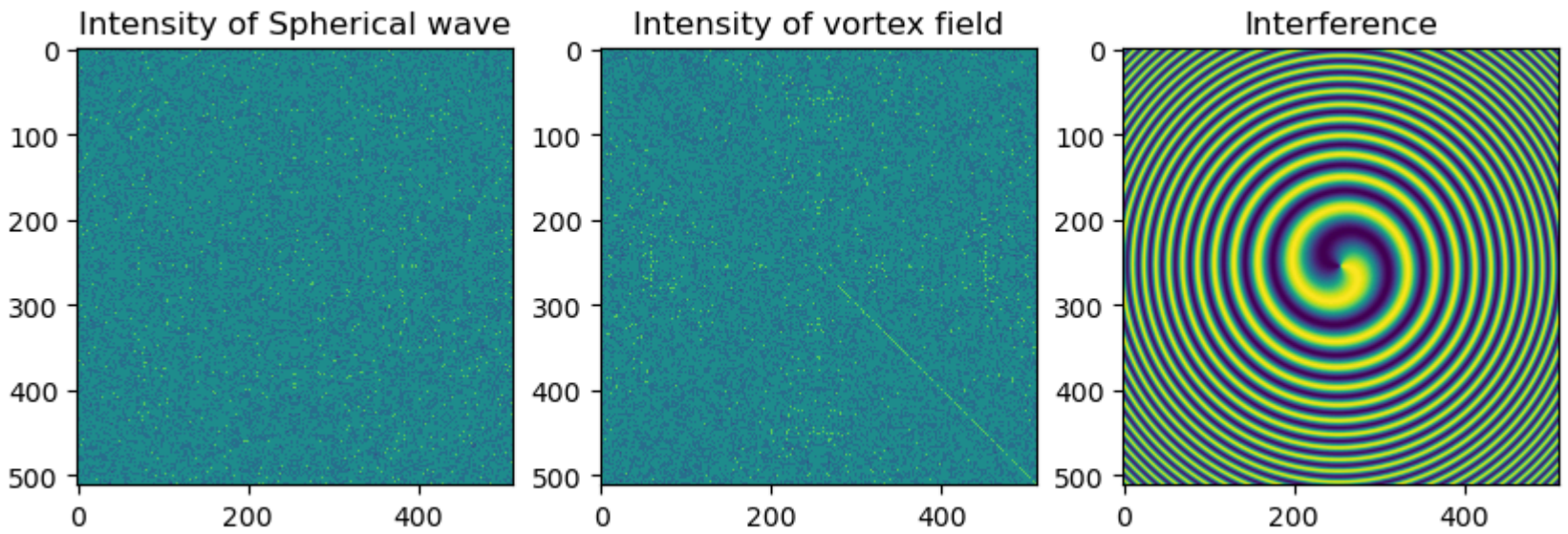

Isolated Peak
0
100
200
300
400
500
0
100
200
300
400
500


Reconstructed field
0
100
200
300
400
500
0
100
200
300
400
500

2.4. Spherical wave + Vortex beam

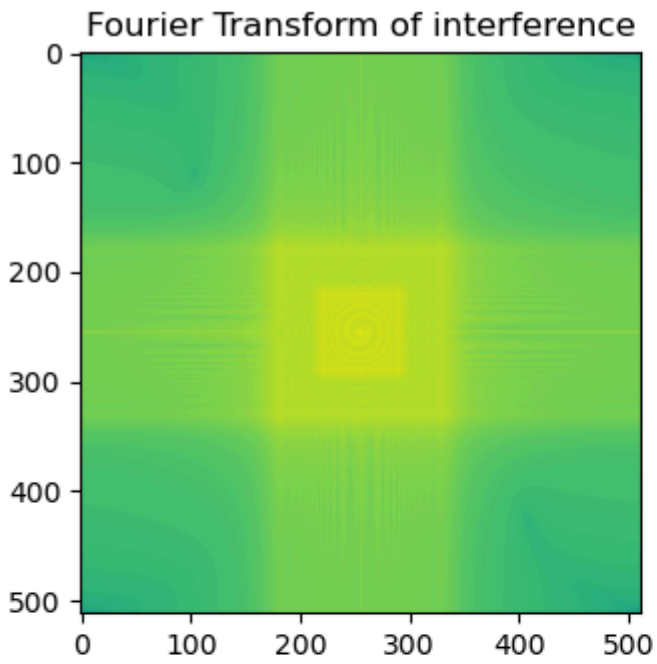
Interference between spherical wave and vortex wave using spherical wave as reference.

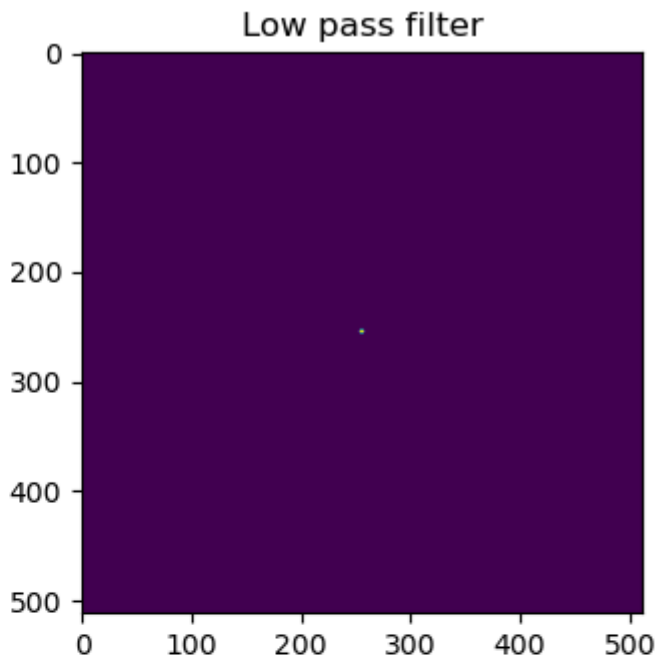
In [10]:
```
E34 = E3 + E4 # Interference
I34 = np.abs(E34)**2

plt.figure(figsize = (10, 8))

plt.subplot(1,3,1)
plt.imshow(I3)
plt.title("Intensity of Spherical wave")

plt.subplot(1,3,2)
plt.imshow(I4)
plt.title('Intensity of vortex field')

plt.subplot(1,3,3)
plt.imshow(I34)
plt.title("Interference")

plt.show()
```

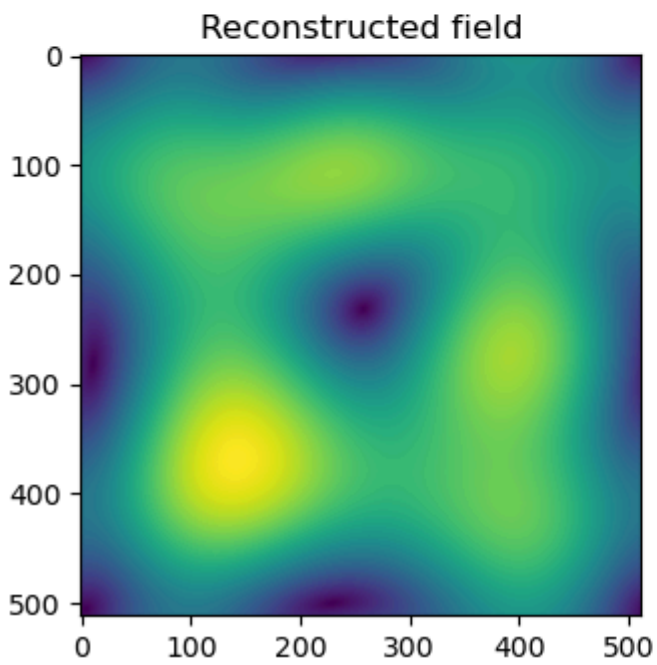


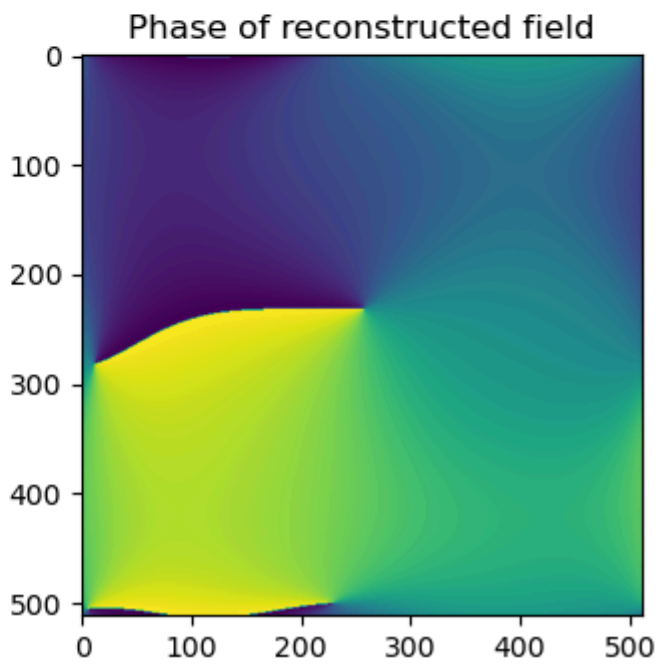
In [11]:
```
i34 = ft2(I34)
I134 = E3 * I34
i134 = ft2(I134)

low = np.zeros((512, 512))
low[x**2 + y**2 <= (1.6*p)**2] = 1

i = i134 * low
I = ift2(i)

plt.figure(figsize = (10, 8))

plt.subplot(2,2,1)
plt.imshow(np.log(np.abs(i134)))
plt.title("Fourier Transform of interference")

plt.subplot(2,2,2)
plt.imshow((np.abs(low)))
plt.title("Low pass filter")

plt.subplot(2,2,3)
plt.imshow((np.abs(I)))
```

```
plt.title("Reconstructed field")

plt.subplot(2,2,4)
plt.imshow(np.angle(I))
plt.title("Phase of reconstructed field")

plt.show()
```


Fourier Transform of interference
0
100
200
300
400
500
0
100
200
300
400
500


Low pass filter
0
100
200
300
400
500
0
100
200
300
400
500


Reconstructed field
0
100
200
300
400
500
0
100
200
300
400
500


Phase of reconstructed field
0
100
200
300
400
500
0
100
200
300
400
500

3. Phase of hologram if we dont know the reference wave frequency

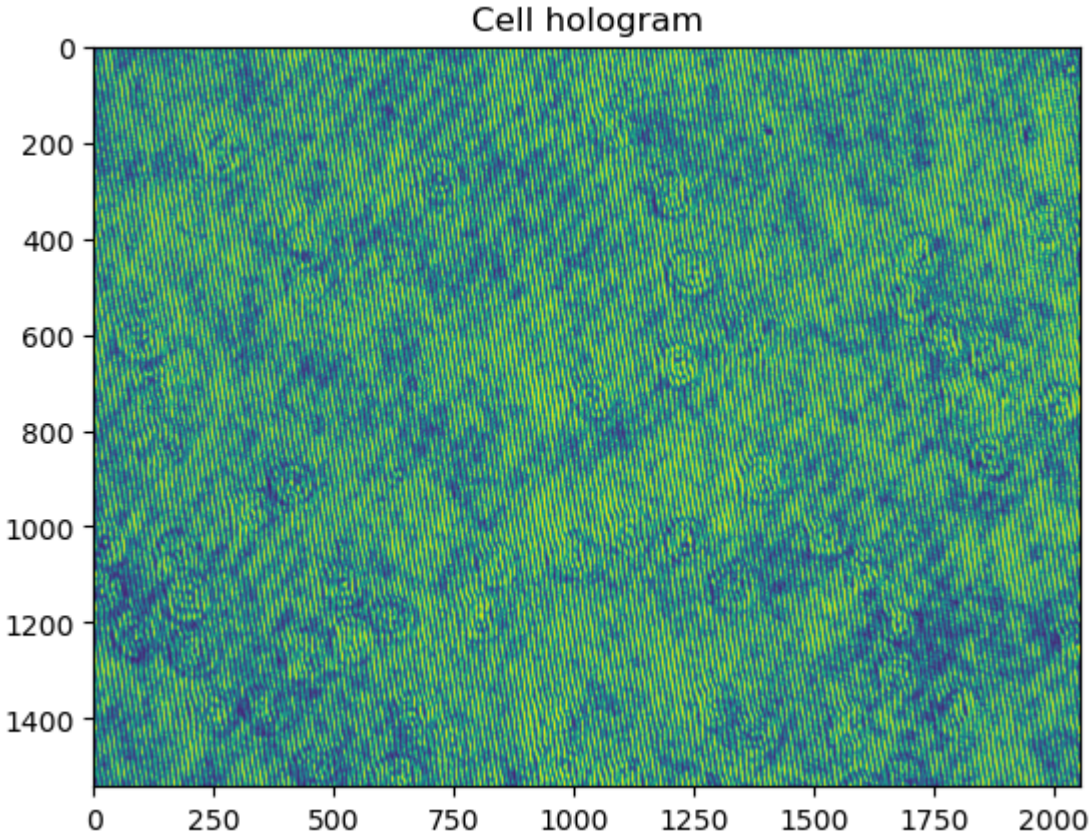
When we don't know the reference wave frequency in a hologram, we can use Fourier analysis to identify it based on the spatial frequency content of the hologram. The recorded hologram I(x,y) contains not only the object and reference wave intensities but also cross terms between the object and reference waves. When we inspect the Fourier transform F(I), we see three main peaks:

1. A central peak at the origin, corresponding to the zero frequency terms.
2. Two off-center peaks at locations (f_x, f_y) and $(-f_x, -f_y)$, which correspond to the spatial frequencies of the reference wave.

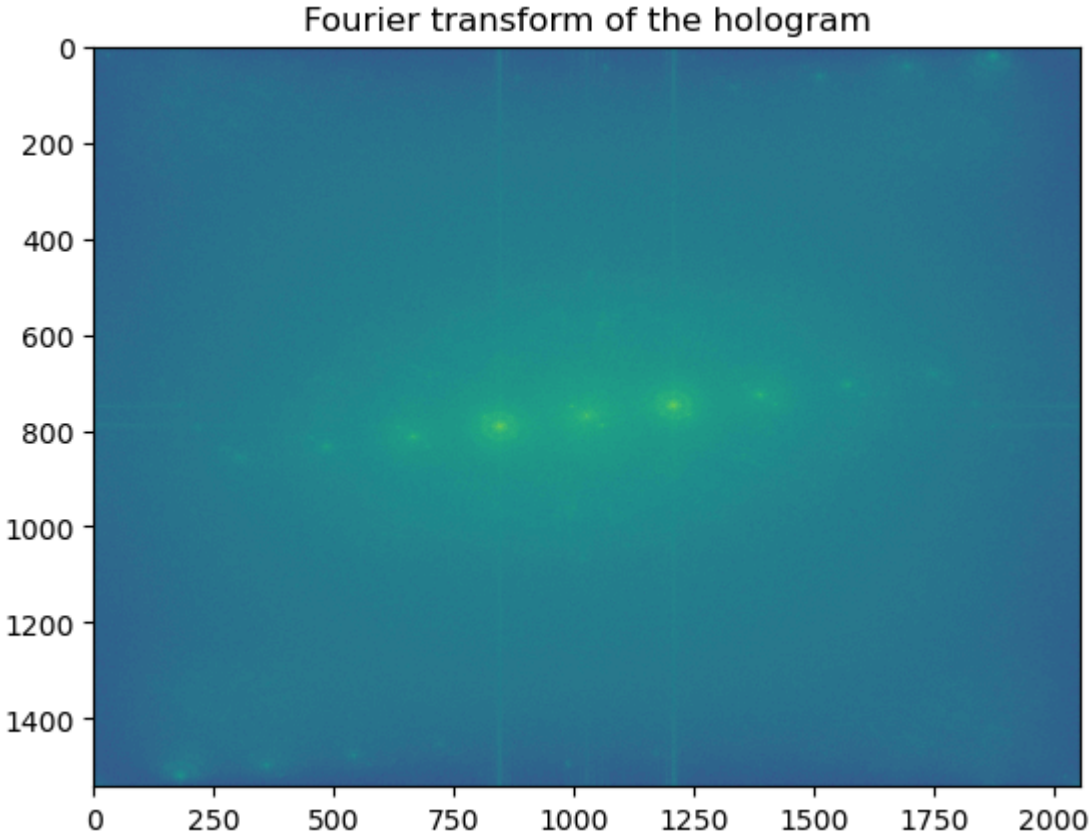
The locations of these off-center peaks directly give us the frequency of the reference wave. By measuring the distance of these peaks from the origin in the Fourier domain, we can deduce the spatial frequencies f_x and f_y of the unknown reference wave.

Once we know the reference wave frequency, we can isolate the terms that correspond to the object wave in the Fourier domain by filtering around the higher-order peak. By shifting and applying an inverse Fourier transform, we can reconstruct the object wave's amplitude and phase information.

In [12]:
```
im = io.imread("Cell_hologram.png")
im = color.rgba2rgb(im)
im = color.rgb2gray(im)

plt.figure()
plt.imshow(im)
plt.title("Cell hologram")
plt.show()
```



In [13]:
```
im_fft = ft2(im)

plt.figure()
plt.imshow(np.log(np.abs(im_fft)))
plt.title("Fourier transform of the hologram")
plt.show()
```

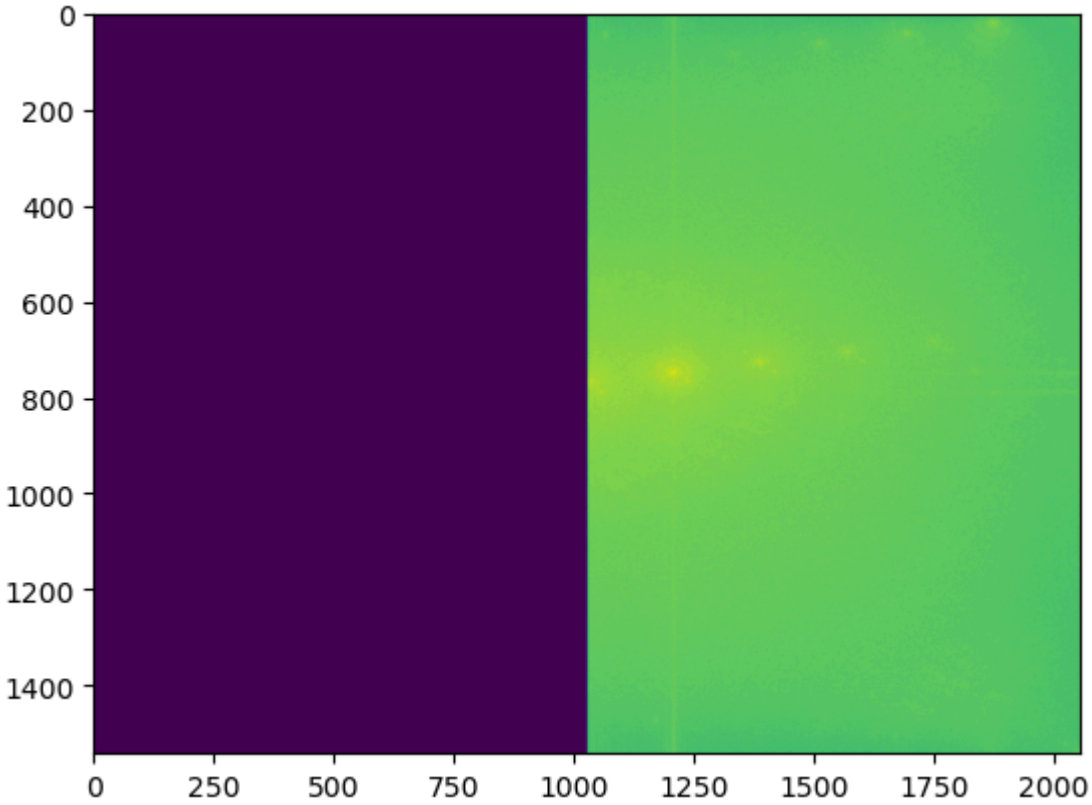

Fourier transform of the hologram
0
200
400
600
800
1000
1200
1400
0
250
500
750
1000
1250
1500
1750
2000

In [14]:
```
N0 = im.shape[1]
N1 = im.shape[0]
p = 5e-6
x0 = np.linspace(-N/2,  N/2, N0, endpoint = True)
y0 = np.linspace(-N/2, N/2, N1, endpoint = True)
x, y = np.meshgrid(x0, y0)
x = x*p
y = y*p

fx0 = x0/N0
fy0 = y0/N1
fx, fy = np.meshgrid(fx0, fy0)
fx = fx/p
fy = fy/p
```

In [15]:
```
mask = np.zeros((N1, N0))
mask[:, 1030:N0] = 1

masked = mask * im_fft
peak = np.unravel_index(np.argmax(masked), im_fft.shape)
print(peak)

plt.imshow(np.log(np.abs(masked) + 1e-10))
```

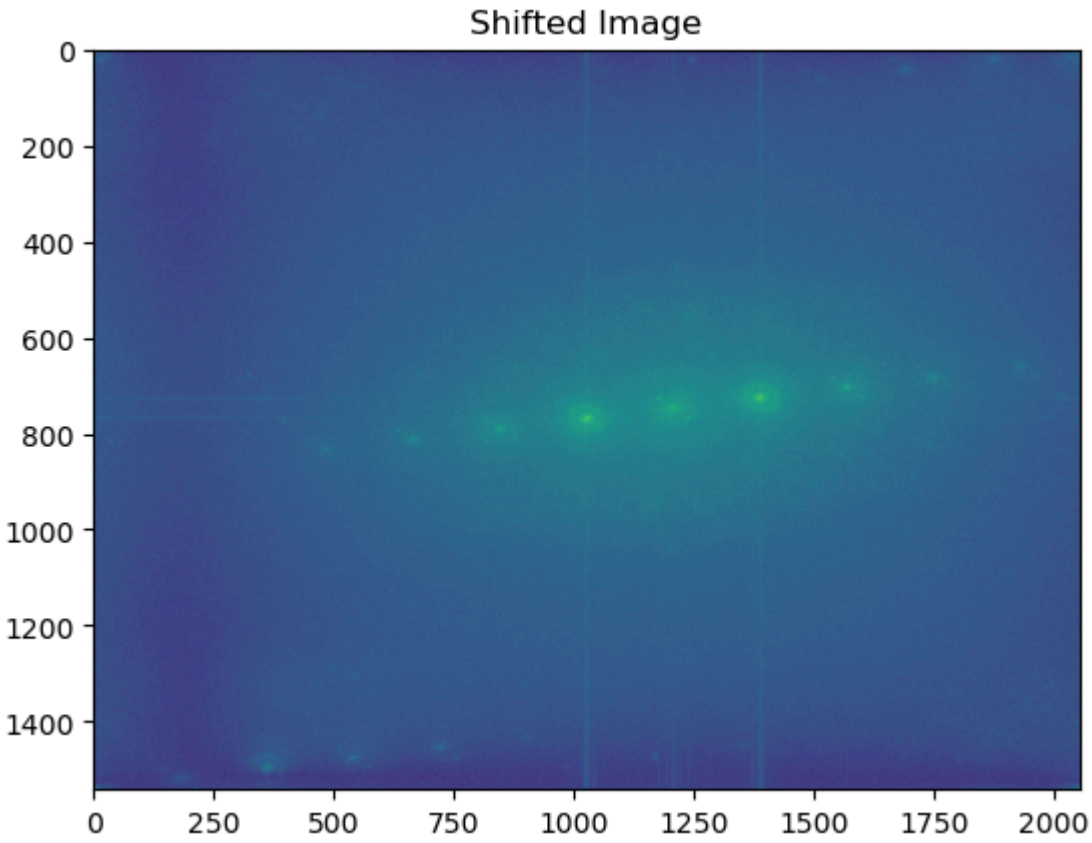
```
(749, 1209)
```

Out[15]: <matplotlib.image.AxesImage at 0x244611511d0>

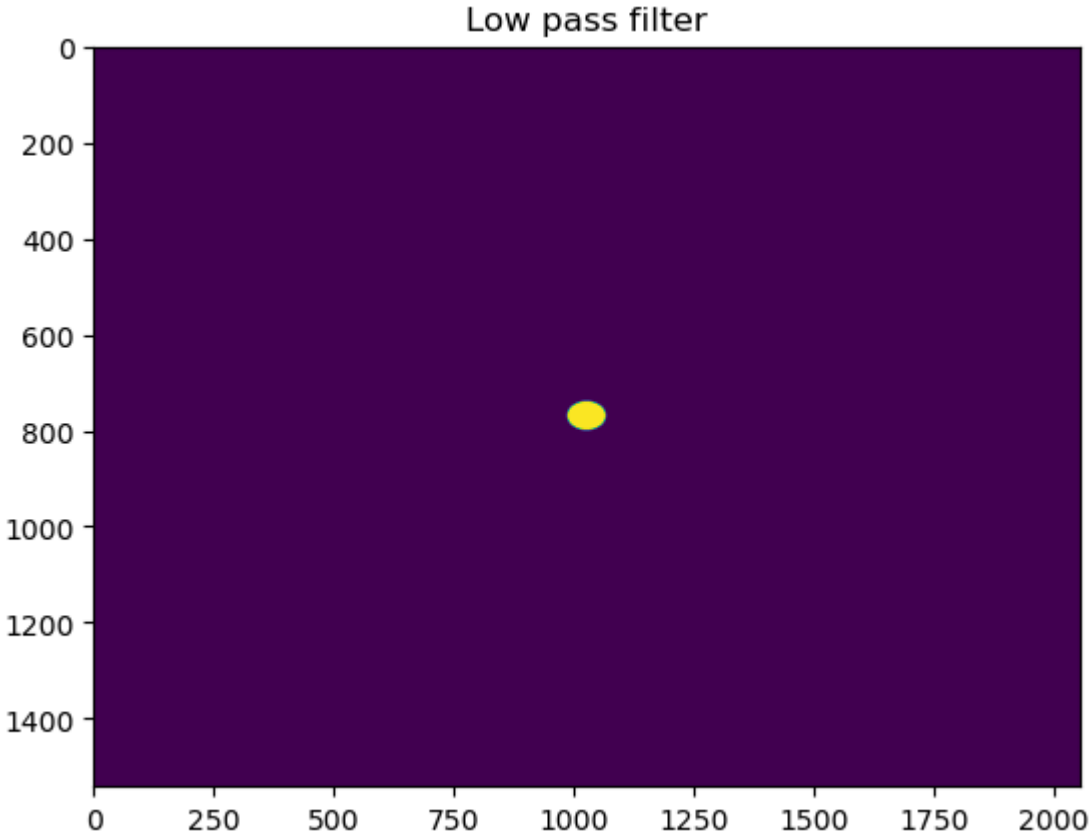


We mask half of the image to get the first order peak, since the central peak just corresponds to the zero frequency terms.

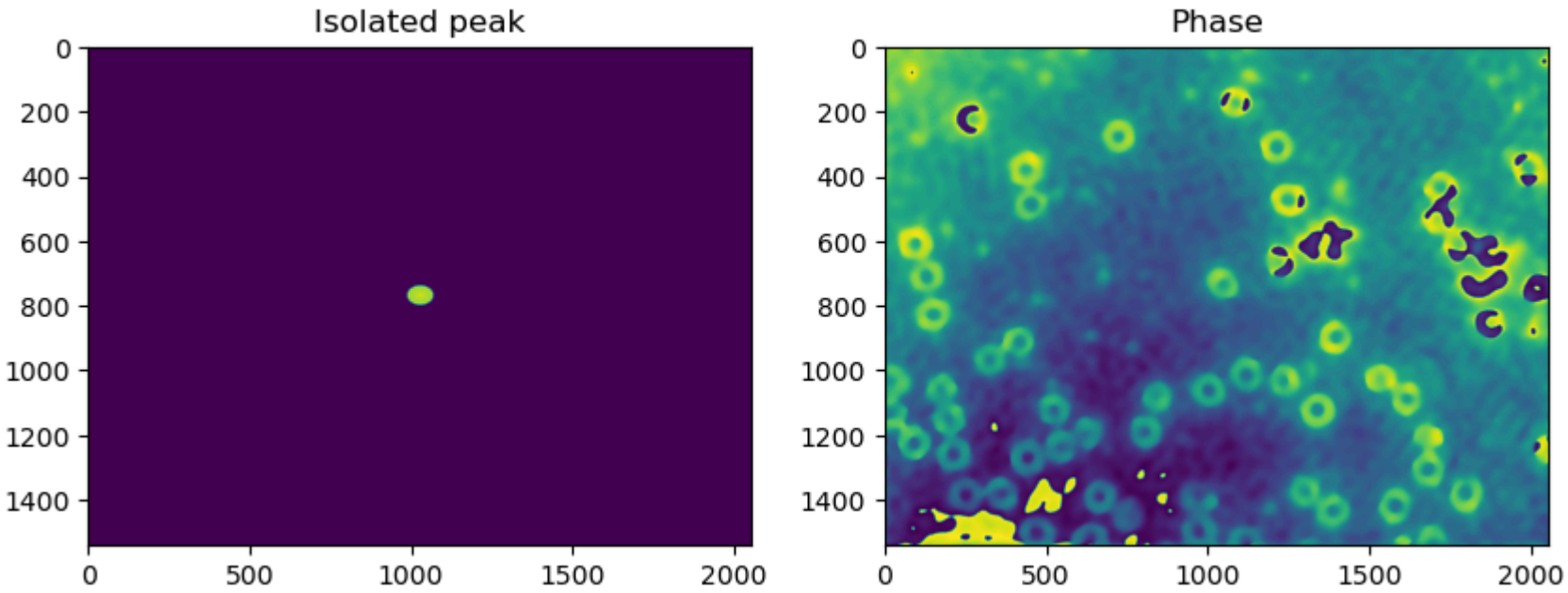
In [16]:
```
# Shifting the higher order peak to center
im_shift = np.roll(im_fft, -22, axis = 0)
im_shift = np.roll(im_shift, 181, axis = 1)

plt.figure()
plt.imshow(np.abs(np.log(im_shift)))
plt.title("Shifted Image")
plt.show()

shifted = np.unravel_index(np.argmax(im_shift), im_fft.shape)
```



In [17]:
```
# Isolating the higher order peak
a = np.zeros(im.shape) # low pass filter
a[(x)**2 + (y)**2 < (10*p)**2] = 1

plt.figure()
plt.imshow(a)
plt.title("Low pass filter")
plt.show()
```



In [18]:
```
carrier_fft = im_shift*a
carrier = ift2(carrier_fft)
plt.figure(figsize = (10,10))
plt.subplot(1,2,1)
plt.imshow(np.log(np.abs(carrier_fft + 1e-10)))
plt.title("Isolated peak")
plt.subplot(1,2,2)
plt.imshow(np.angle(carrier))
plt.title("Phase")
plt.show()
```



4. Phase Unwrapping: Transport of Intensity equation

There are many digital signal-processing techniques that use the four quadrant arctangent function to calculate the phase of a signal. The amplitude of the phase that is calculated can take any value and typically exceeds the range [-π,π] that is returned by the arctangent function. In cases where the phase exceeds this range of values, it will be wrapped so that it stays within the normal range [-π,π]. In such cases the wrapped phase will contain one, or more 2π jumps.

The 2π jumps that are present in the wrapped phase signal must be removed in order to return the phase signal $xw(n)$ to a continuous form and hence make the phase usable in any analysis or further processing. This process is called phase unwrapping and has the effect of returning a wrapped phase signal to a continuous phase signal that is free from 2π jumps.

TIE (Transport of Intensity Equation) describes the relationship between the unwrapped phase (true phase) Θ_{UW} and the intensity in the form of a differential equation as follows:

$$\nabla(I\nabla\Theta_{UW}) = \frac{-2\pi}{\lambda}\frac{\partial I}{\partial z} \tag{2}$$

where ∇ is the transverse gradient, I is the intensity of signal field, and λ is the wavelength.

TIE offers the advantage of simplicity in phase calculation owing to its closed-form solution and its non-interferometric nature, eliminating the need for phase unwrapping. Consider the case of a pure-phase object, i.e., the disturbance only influences the phase of light, I(x,y,0) is constant. Thus, the left-hand side (LHS) of above equation is simplified to a Laplacian operator.

$$\Theta(x,y) = \nabla^{-2}_{perpendicular}[-\frac{2\pi}{\lambda}\frac{\partial I}{\partial z}] \tag{3}$$

To solve this equation, the first step is to simplify the expression on the right-hand side by replacing the intensity derivative with difference:

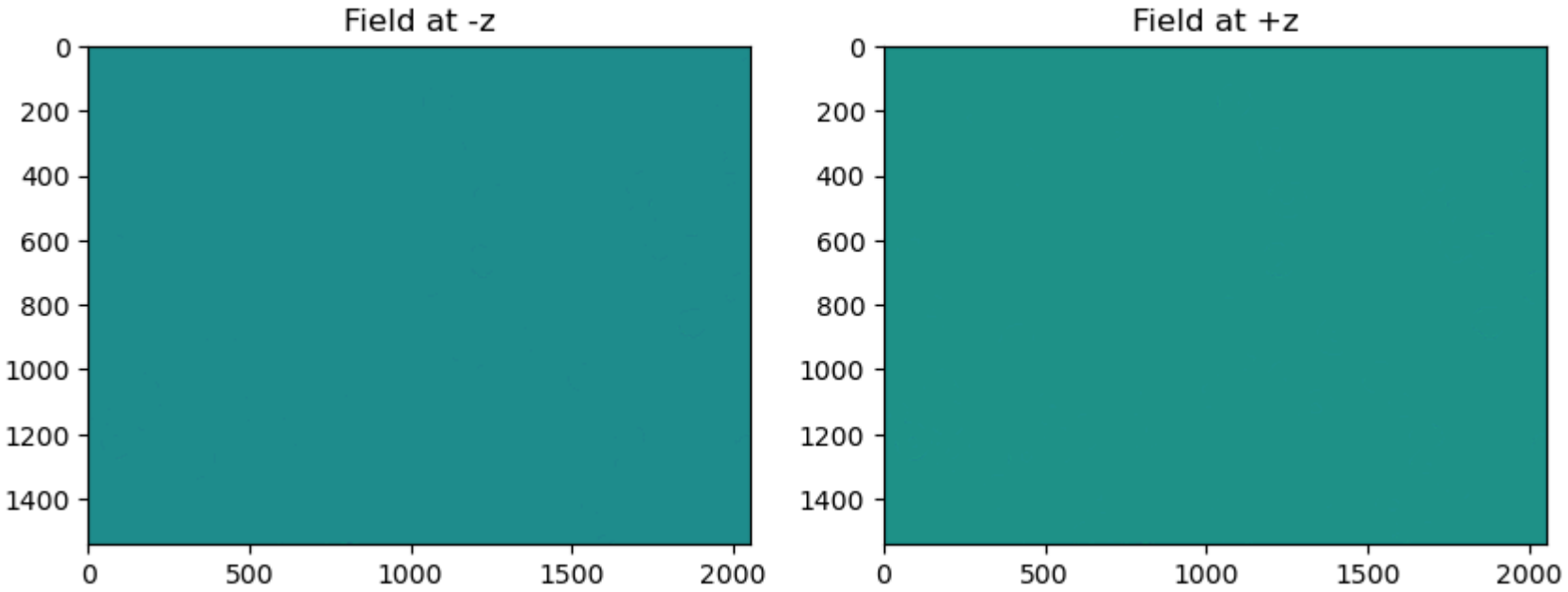
$$\frac{\partial I}{\partial z} = \frac{|u(x,y,\Delta z)|^2 - |u(x,y,-\Delta z)|^2}{2\delta z} \tag{4}$$

where δz is the plane separation. The wave propagation is emulated by the Rayleigh-Sommerfeld (RS) diffraction integral, and numerical calculations are conducted using an FFT-based Angular Spectrum (AS) method.

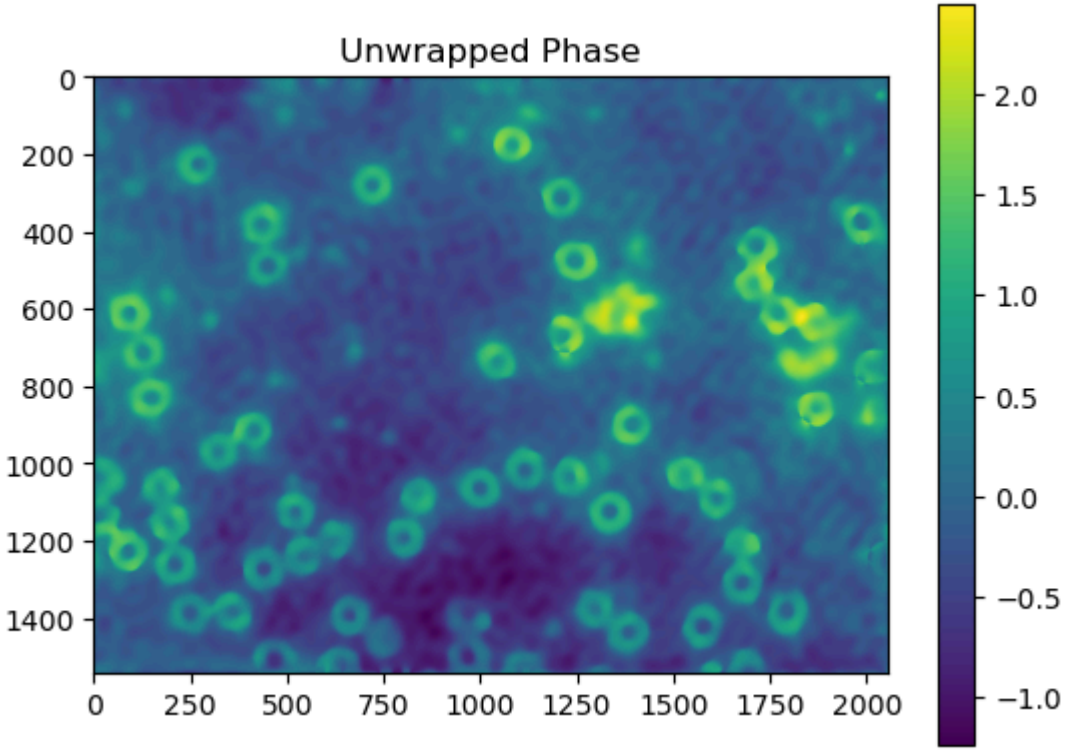
$$\nabla^{-2}g(x,y) = \int\int df_x df_y G(f_x,f_y) exp(i2\pi(f_x x + f_y + y)) \frac{[-4\pi^2(f_x^2+f_y^2)]}{[-4\pi^2(f_x^2+f_y^2)]^2 + \epsilon^2} \tag{5}$$

In [19]:
```
wavelength = 0.5e-6
k = 2*np.pi/wavelength

alpha = np.sqrt(k**2 - 4*np.pi**2*(fx**2 + fy**2))

def propagation(field, z):
    # Low pass filter
    lp_f1 = (1/wavelength) * 1 / (np.sqrt(1 + (2 * z / (N1 * p))**2))
    lp_f2 = (1/wavelength) * 1 / (np.sqrt(1 + (2 * z / (N0 * p))**2))
    filtr = np.zeros((N1, N0))
    filtr[(fx**2 + fy**2 < lp_f1*lp_f2)] = 1
    H = np.exp(1j*alpha*z)
    field_fft = ft2(field)
    field_prop = field_fft*H
    return ift2(field_prop)
```

In [20]:
```
Field = np.exp(1j * np.angle(carrier))
z1 = -2*p
z2 = 2*p

# Defocused image
field1 = propagation(Field, z1)
field2 = propagation(Field, z2)

del_Iz = (np.abs(field2)**2 - np.abs(field1)**2)/(2 * (z2 - z1))

plt.figure(figsize = (10, 8))
plt.subplot(1,2,1)
plt.imshow(np.abs(field1))
plt.title("Field at -z")
plt.subplot(1,2,2)
plt.imshow(np.abs(field2))
plt.title("Field at +z")
plt.show()
```



In [21]:
```
epsilon = 1e-10
g = -(2*np.pi/wavelength)*(del_Iz)
G = ft2(g)
kernel = G * (-4*np.pi**2*(fx**2 + fy**2))/((-4*np.pi**2*(fx**2 + fy**2))**2 + epsilon**2)
unwrapped = ift2(kernel)

plt.figure()
plt.imshow(np.real(unwrapped))
plt.title("Unwrapped Phase")
plt.colorbar()
plt.show()
```



5. Holographic Autofocusing : Edge Sparsity Criterion

The SoG method is based on the edge sparsity of the object's in-focus image. The underlying principle is that when a hologram is correctly focused, the number of sharp edges in the reconstructed image is minimized, as objects typically have flat regions with well-defined boundaries. By analyzing the sparsity of these edges, SoG identifies the optimal focus distance. The two specific measures used are:

1. Gini of the Gradient (GoG):

The Gini index (GI) measures sparsity in the distribution of values in I∇UI. A high Gini index indicates that most of the energy is concentrated in a few sharp edges, characteristic of an in-focus image. For a gradient image N pixels, sorted in ascending order a(k), the Gini index is defined as:

$$GI(C) = 1 - \frac{2}{N}\sum_{k=1}^{N}\frac{a_{[k]}}{sum(C)}\left(\frac{N-k+0.5}{N}\right) \tag{6}$$

2. Tamura Coefficient of the Gradient (ToG):

The Tamura coefficient (TC) measures the sharpness and texture based on the standard deviation of the gradient normalized by its mean. For the gradient image C = I∇UI, the Tamura coefficient is given by:

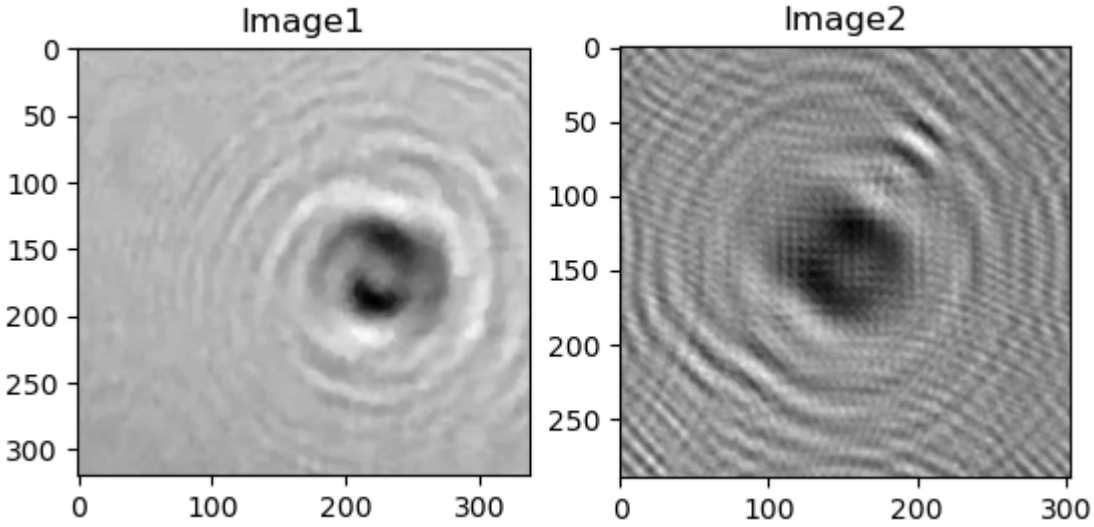
$$TC(C) = \sqrt{\frac{\sigma(C)}{< C >}} \tag{7}$$

The SoG method finds the optimal focus plane by digitally refocusing the hologram across various depths z z and evaluating SoG at each depth. The z z-plane that maximizes SoG (i.e., where edge sparsity is highest) is selected as the optimal focus distance.

The SoG criterion, using GoG and ToG, provides a unified, robust, and accurate method for autofocusing in digital holography. By focusing on the sparsity of edges in the gradient of the complex wavefront, SoG avoids the polarity issues faced by traditional criteria and achieves high accuracy across amplitude-only, phase-only, and mixed contrast samples.

In [22]:
```
test1 = io.imread("test_image1.jpg")
test2 = io.imread("test_imag2.jpg")

plt.figure()
plt.subplot(1,2,1)
plt.imshow(test1, cmap = "gray")
plt.title("Image1")
```

```
plt.subplot(1,2,2)
plt.imshow(test2, cmap = "gray")
plt.title("Image2")
plt.show()
```



5.1. Tamura of the Gradient (TOG)

5.1.1. Image 1

In [23]:
```
Nx = test1.shape[1]
Ny = test1.shape[0]
p = 1.4e-6
lambda_ = 650e-9
k = 2*np.pi/lambda_

x0 = np.linspace(-Nx/2, Nx/2, Nx, endpoint = True)
y0 = np.linspace(-Ny/2, Ny/2, Ny, endpoint = True)
x, y = np.meshgrid(x0, y0)
x, y = x*p, y*p

fx0, fy0 = x0/Nx, y0/Ny
```

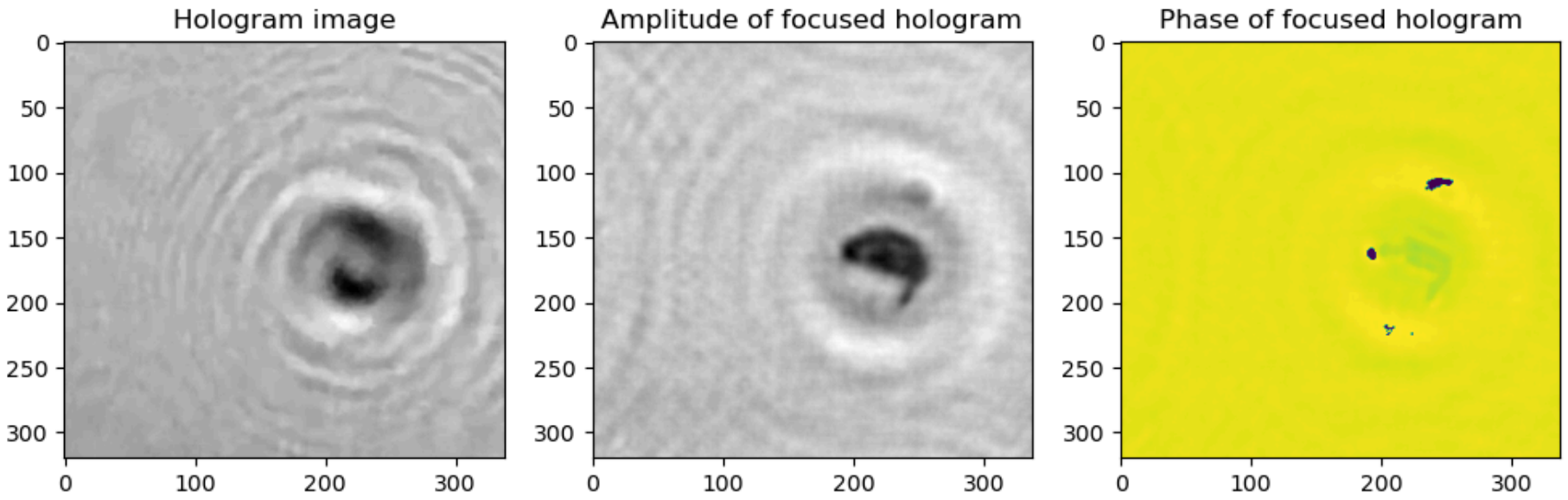
```
fx, fy = np.meshgrid(fx0, fy0)
fx, fy = fx/p, fy/p
```

In [24]:
```
def ft2(x):
    return np.fft.fftshift(np.fft.fft2(np.fft.ifftshift(x)))

def ift2(x):
    return np.fft.fftshift(np.fft.ifft2(np.fft.ifftshift(x)))

def propagation(field, z):
    alpha = np.sqrt(k**2 - 4*np.pi**2*(fx**2 + fy**2))
    H = np.exp(1j*alpha*z)
    field_fft = ft2(field)
    field_prop = field_fft*H
    return ift2(field_prop)
```

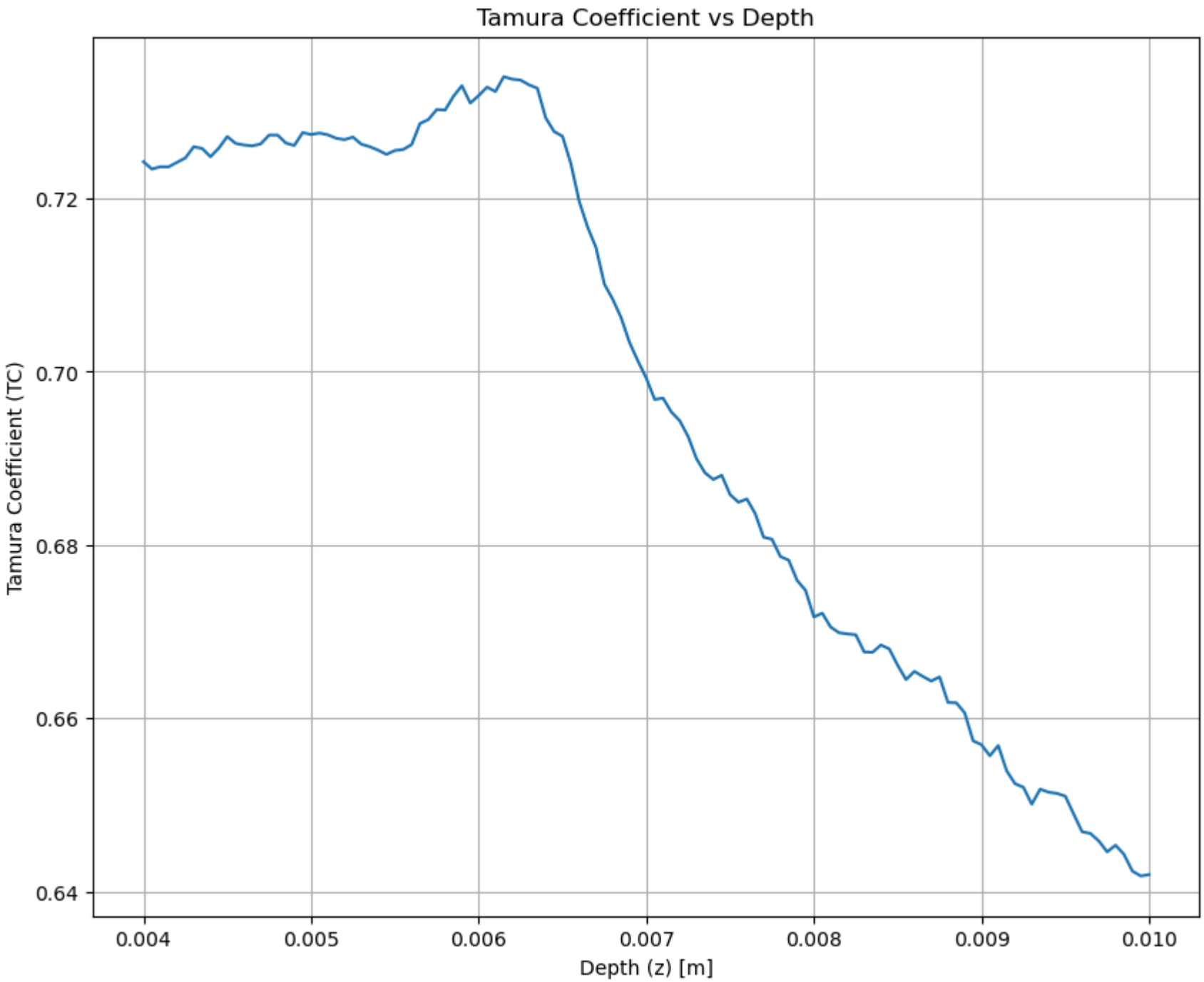
In [25]:
```
z_values = np.linspace(4e-3, 10e-3, 121, endpoint = True) # Range of the distance
TC_list = []

# Scan the image for the range and compute TC
for z in z_values:
    uz = propagation(test1, z)
    ux, uy = np.gradient(uz)
    C = np.sqrt(np.abs(ux)**2 + np.abs(uy)**2)
    TC = np.sqrt(np.std(C)/np.mean(C))
    TC_list.append(TC)
```

In [26]:
```
focus = z_values[np.argmax(TC_list)] # z value for which TC is highest

focused_image = propagation(test1, -focus) # Back propagation of the image to get the focused image

plt.figure(figsize =(12,10))

plt.subplot(1,3,1)
plt.imshow(test1, cmap = "gray")
plt.title("Hologram image")

plt.subplot(1,3,2)
plt.imshow(np.abs(focused_image), cmap = "gray")
plt.title("Amplitude of focused hologram")
```

```
plt.subplot(1,3,3)
plt.imshow(np.angle(focused_image))
plt.title("Phase of focused hologram")

plt.show()
```



In [27]:
```
# Plot the TC vs Depth
plt.figure(figsize = (10, 8))
plt.plot(z_values, TC_list)
plt.xlabel("Depth (z) [m]")
plt.ylabel("Tamura Coefficient (TC)")
plt.title("Tamura Coefficient vs Depth")
plt.grid(True)
plt.show()

print(f'''Focus Distance : {focus*1e3}mm
Tamura Coefficient : {np.max(TC_list)}''')
```


Tamura Coefficient vs Depth
Tamura Coefficient (TC)
0.72
0.70
0.68
0.66
0.64
0.004
0.005
0.006
0.007
0.008
0.009
0.010
Depth (z) [m]

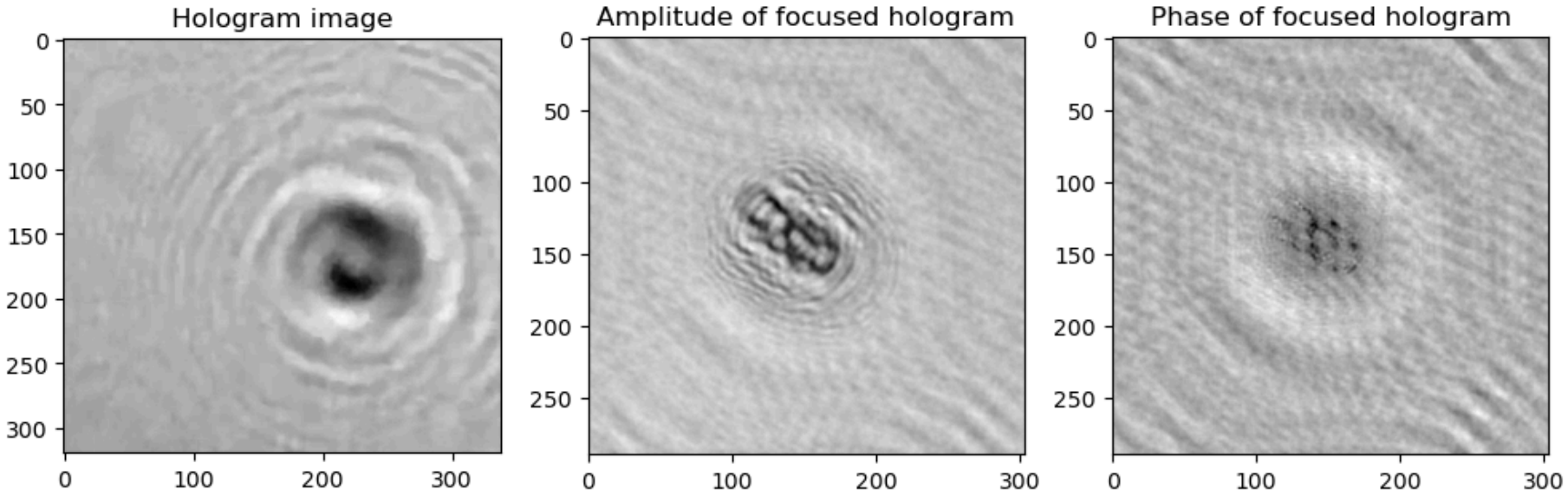
```
Focus Distance : 6.15mm
Tamura Coefficient : 0.7340535621806802
```

5.1.2. Image 2

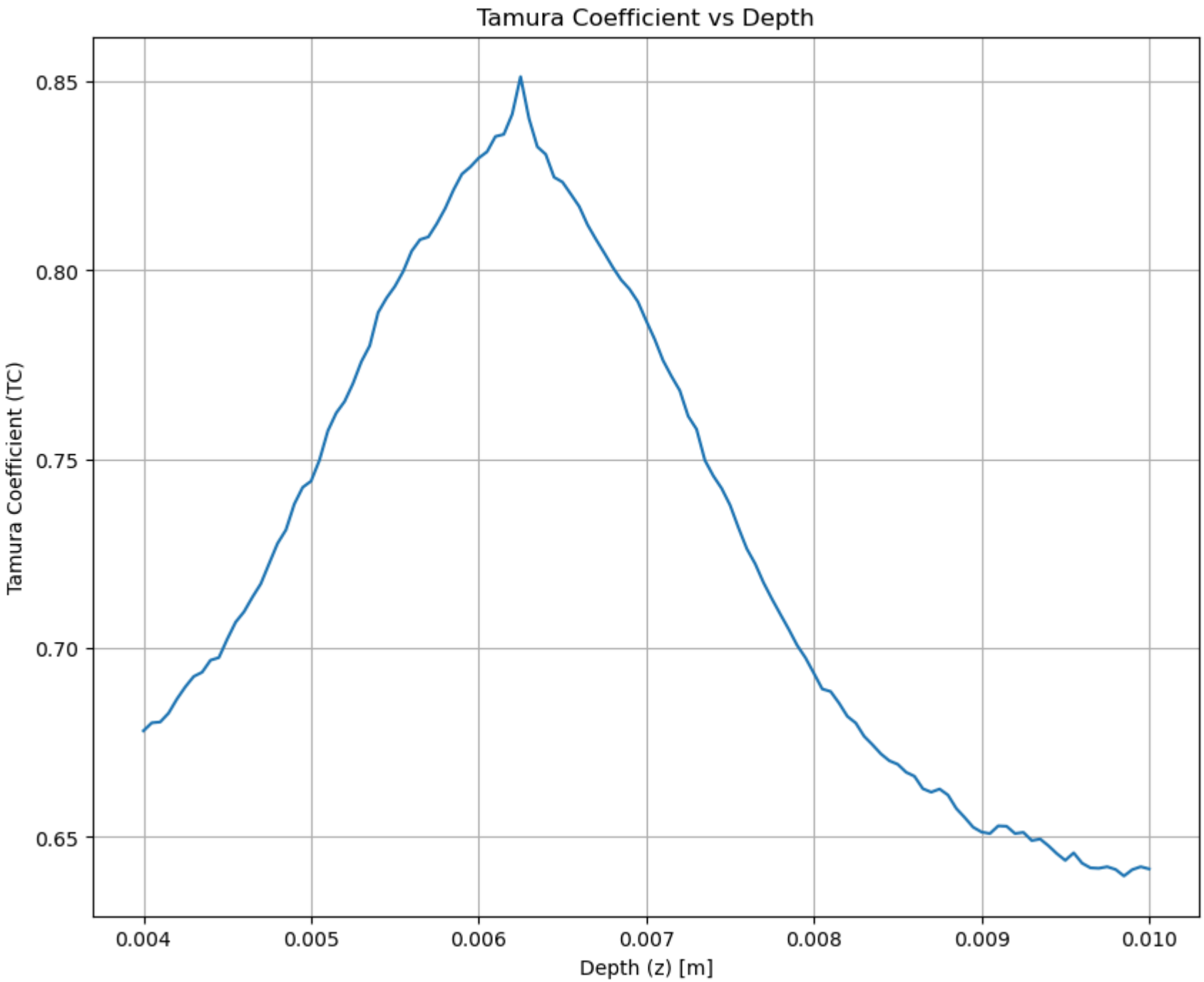
In [28]:

```python
Nx = test2.shape[1]
Ny = test2.shape[0]
p = 1.4e-6
lambda_ = 650e-9
k = 2*np.pi/lambda_

x0 = np.linspace(-Nx/2, Nx/2, Nx, endpoint = True)
y0 = np.linspace(-Ny/2, Ny/2, Ny, endpoint = True)
x, y = np.meshgrid(x0, y0)
x, y = x*p, y*p

fx0, fy0 = x0/Nx, y0/Ny
fx, fy = np.meshgrid(fx0, fy0)
fx, fy = fx/p, fy/p

z_values = np.linspace(4e-3, 10e-3, 121, endpoint = True)
TC_list = []

for z in z_values:
    uz = propagation(test2, z)
    ux, uy = np.gradient(uz)
    C = np.sqrt(np.abs(ux)**2 + np.abs(uy)**2)
    TC = np.sqrt(np.std(C)/np.mean(C))
    TC_list.append(TC)

focus = z_values[np.argmax(TC_list)]

focused_image = propagation(test2, -focus)

plt.figure(figsize =(12,10))

plt.subplot(1,3,1)
plt.imshow(test1, cmap = "gray")
plt.title("Hologram image")
```

```
plt.subplot(1,3,2)
plt.imshow(np.abs(focused_image), cmap = "gray")
plt.title("Amplitude of focused hologram")

plt.subplot(1,3,3)
plt.imshow(np.angle(focused_image), cmap = "gray")
plt.title("Phase of focused hologram")

plt.show()
```



In [29]:
```
plt.figure(figsize = (10, 8))
plt.plot(z_values, TC_list)
plt.xlabel("Depth (z) [m]")
plt.ylabel("Tamura Coefficient (TC)")
plt.title("Tamura Coefficient vs Depth")
plt.grid(True)
plt.show()

print(f'''Focus Distance : {focus*1e3}mm
Tamura Coefficient : {np.max(TC_list)}''')
```


Tamura Coefficient vs Depth
Tamura Coefficient (TC)
0.85
0.80
0.75
0.70
0.65
0.004
0.005
0.006
0.007
0.008
0.009
0.010
Depth (z) [m]

```
Focus Distance : 6.25mm
Tamura Coefficient : 0.8512631212885063
```

6. Conclusion

This lab provides a comprehensive overview of holographic principles and techniques for capturing and reconstructing wavefronts. Through various interference patterns, we demonstrated the ability to record complex 3D information and reconstruct it using reference wave techniques. Key methodologies, including Fourier analysis and SoG-based autofocusing, proved effective in refining the reconstructed image's clarity.

In [ ]: