Simulating Speckle with Python

Joseph W. Goodman

SPIE PRESS
Bellingham, Washington USA

Library of Congress Cataloging-in-Publication Data

Names: Goodman, Joseph W., author.
Title: Speckle simulation with Python / Joseph W. Goodman.
Description: Bellingham, Washington, USA : SPIE Press, [2024] | Includes bibliographical references.
Identifiers: LCCN 2023048231 | ISBN 9781510671362 (paperback) | ISBN 9781510671379 (pdf)
Subjects: LCSH: Speckle--Computer simulation. | Speckle--Statistical methods. | Python (Computer program language)
Classification: LCC QC367.3.S64 G66 2024 | DDC 537.5/6--dc23/eng/20231122
LC record available at https://lccn.loc.gov/2023048231

Published by
SPIE
P.O. Box 10
Bellingham, Washington 98227-0010 USA
Phone: +1 360.676.3290
Fax: +1 360.647.1445
Email: books@spie.org
Web: www.spie.org

The cover image was generated by DALL-E.

Printed in the United States of America.
First printing 2024.
For updates to this book, visit http://spie.org and type “PM369” in the search field.

Table of Contents

Chapter 1

Introduction

The speckle phenomenon is ubiquitous in many fields of science and technology. Speckle phenomena can be seen in many different coherent imaging modalities, including acoustical imaging (e.g., medical ultrasound) and microwave imaging (e.g., synthetic-aperture radar imaging). This book focuses on simulating *optical* speckle with Python, but the same methods used can in many cases be applied to other software packages.

This book is a revised version of the previously published book entitled *Simulating Speckle with Mathematica*® (Ref. [1]). The reader may wonder why Python has been chosen as the software package for this new book. There are several reasons for this choice. First, and most important, Python is easily available for download from the web and, unlike *Mathematica*, does not require a paid license. Second, many students in the sciences and engineering use Python as their software package of choice. The one disadvantage of Python when compared with *Mathematica* is that basic Python does not include all of the scientific functions one typically needs, but additional add-on libraries are available and can be incorporated in Python with appropriate instructions in the Python program. This book has been written entirely in Python using the integrated development environment (IDE) Jupyter Notebook, again, available from the web without a paid license. The Python files for all chapters can be found at the following URL: https://spie.org/Samples/Pressbook_Supplemental/PM369_sup.zip. This book is meant as a companion to the book *Speckle Phenomena in Optics: Theory and Applications, 2nd Edition*, published by SPIE Press (Ref. [2]). An extensive list of references can be found in that book.

1.1 Brief Python Background

We do not attempt to teach the reader how to code in Python here. For novices, an excellent book is Ref. [3], while for more advanced scientific programming, Ref. [4] is very helpful. Python version 3 and Jupyter Notebook 6.1.4 have been used here and are recommended to the reader. The Python environment consists of the basic Python core plus a multitude of add-on libraries. The primary libraries needed for

our computations are **numPy**, **sciPy**, **mpmath**, and **pyplot** from **matplotlib**, which can be loaded and given abbreviations with the commands below:

In [2]:
```
import numpy as np
import scipy as sci
import mpmath as mp
from matplotlib import pyplot as plt
```

Note the abbreviations **np**, **plt**, **mp**, and **sci**. Often we may import a library such as **numPy** multiple times in a single chapter so that portions of the code can be run without running the code for the entire chapter. To illustrate the method for calling a specific function, consider calculating the square root of 2 using **numPy**:

In [3]:
```
np.sqrt(2)
```

Out[3]:
```
1.4142135623730951
```

To execute a cell of code, you must press shift-return. To see all of the available commands in **numPy**, run the code **dir(np)**.

In Python, parentheses ( ) are used to enclose the arguments of functions, to group mathematics, and to define tuples, which are immutable lists. Square brackets [ ] are used to enclose lists and to enclose index numbers of elements in lists. Double quotes ”” or single quotes ’ ’ are used to define strings. Also useful to note at the start, comments in the code begin with a symbol #, and line breaks in the code are introduced by the symbol \.

From the coding examples presented in this book, the reader may gain a sufficient understanding of Python to at least get started. The author is not an expert in Python programming, but has gained enough knowledge to carry out the examples in this book. A Python expert might use simpler and/or more sophisticated coding than presented here. However, such coding may be more difficult to understand than the simpler coding used here.

1.2 Speckle Background

In optics, speckle arises when light is reflected from a rough surface or is transmitted through a diffuser that jumbles the phase at each object point by an unpredictable amount. The contributions from various scattering regions on the object then generate a multitude of complex wavelets that interfere to produce speckle. When the reflected or transmitted light propagates to an observation plane some distance away, complicated fluctuations of amplitude, phase, and intensity occur in that plane due to random interference. These fluctuations are what we refer

to as speckle. If the phase perturbations introduced by the object equal or exceed 2π radians, we say that the speckle is fully developed. If, on the other hand, the phase fluctuations introduced by the object are less than 2π radians, the resulting speckle is called partially developed. In some cases, one scattered wavelet may be much larger than the others, in which case the speckle is neither fully developed nor partially developed, but rather requires a special development to understand the statistics of the observed light amplitude or intensity.

It is important to remember that when we speak of the statistics of speckle, we are speaking of fluctuations over an ensemble of macroscopically similar but microscopically different rough surfaces or diffusers. The resulting perturbed wavefront is unchanging for any one surface or diffuser, but changes as different rough surfaces or different diffusers are introduced. Since we do not know the fine-scale structure of the surface fluctuations, the best we can do is specify statistics over an ensemble of possible surfaces. To experimentally discover statistical properties of the speckle, either many microscopically different reflecting or transmitting structures must be introduced sequentially, or, in the case of speckle that is spatially ergodic (i.e., statistically similar over a wide region of the speckle pattern), spatial averages should yield the same results as an ensemble average.

Note that the theoretical results for the probability density functions of amplitude or intensity, as found in Ref. [2], are based on the assumption of an infinite number of random phasor contributions. Obviously, we can not simulate an infinite number of random phasors on the computer, but we can choose a large finite number. Our results, then, will yield information on how well the theoretical predictions of the statistics of amplitude and intensity match the results based on a large but finite number of phasors.

1.3 Methods for Simulating Speckle

In Chapters 2 and 3, we simulate the first-order statistics (i.e., the speckle at a single point in space or time) by summing a large number of complex phasors. Assumptions are made in various sections about the statistics of the phase of the phasors or about the number of phasors. Histograms of the various results are computed and compared with the theoretical results valid for an infinite number of phasors. The ideas behind these simulations are quite straightforward. We simply sum a finite number of complex phasors and examine the statistics of amplitude (Chapter 2) or intensity (Chapter 3) by calculating histograms of the results from a large number of independent trials.

Chapter 2

First-Order Statistics of Speckle Amplitude

By first-order statistics of speckle we mean the statistics observed at a point in space or a point in time, with the statistics being over an ensemble of rough surfaces or rough diffusers. First we consider the statistics of speckle complex amplitude, relevant when using ultrasound or microwave illumination of surfaces that are rough on the scale of their individual wavelengths. For such imaging modalities, it is possible to measure both the magnitude and the phase of the wavefields. In Chapter 3, we turn to the statistics of speckle intensity, which is the most relevant quantity for the optical region of the spectrum, where a detector can measure only intensity.

2.1 Speckle as the Sum of Many Independent Random Phasors

The statistics of speckle at a point are the same as the statistics of a sum of complex phasors with independent amplitudes and phases. Let the symbol c_k represent the k^{th} element in an array of N different complex phasors of the following form:

$$c_k = a_k \exp(j\phi_k), \qquad k = 1, \ldots, N, \tag{2-1}$$

where a_k is a non-negative amplitude and ϕ_k is a phase. Each phasor represents an independent contribution to the complex value of the speckle field at a point in space or time. We assume that a_k and ϕ_k are random variables drawn from a statistical ensemble, and that they are statistically independent of each other and statistically independent of all other random variables occurring in the array of phasors. We then form the normalized sum,

$$\begin{aligned} S &= \frac{1}{\sqrt{N}} \sum_{k=1}^{N} c_k = \frac{1}{\sqrt{N}} \sum_{k=1}^{N} a_k \exp(j\phi_k) \\ &= \frac{1}{\sqrt{N}} a_k \cos\phi_k + j\frac{1}{\sqrt{N}} \sum_{k=1}^{N} a_k \sin\phi_k, \end{aligned} \tag{2-2}$$

Chapter 3

First-Order Statistics of Speckle Intensity

In the optical region of the spectrum, detectors are unable to follow the ultra-fast cycles of the optical field amplitude, but rather, they respond to incident power or intensity, averaged over some response time of the detector and some finite area of the detector element. For that reason, in studying speckle in the optical region of the spectrum, the intensity statistics of random phasor sums are of much greater interest than amplitude statistics. We now turn our attention to simulating the intensity statistics of random phasor sums. The Python code used in the previous chapter for various amplitude cases can be reused with small changes. Instead of calculating the length of the random phasor sums, we must calculate the squared length of those sums, for the squared length corresponds to intensity. Accordingly, we modify the previous code to calculate intensity statistics. We will compare the analytical results with histograms in the various cases we consider.

3.1 Intensity Statistics of the Sum of Many Random Phasors with Unit Lengths and Random Phases

The first case considered will be one for which the lengths of all phasors in the sum are unity (aside from the $1/\sqrt{N}$ normalization) and the individual phases are uniformly distributed on the interval $(-\pi,\pi)$. The appropriate sections of Ref. [2] for this case are 3.1 and 3.2. Our goal is to calculate discrete approximations to the probability density functions of the real part, the imaginary part, and the squared magnitude of S. Note that the squared magnitude of S is the same as the squared length of the resultant phasor, and corresponds to optical intensity. In this section we consider only a large number of phasors $(N \geq 10)$.

Two parameters need to be chosen at the start: the first is N, the number of independent phasors in the sum defining S; the second is M, which is the number of times the simulation is run with independent phases for the individual phasors contributing to the sum. This multitude of runs collects the statistical data we desire. The value $M = 100,000$ is used in this case. If you wish to obtain more-

Chapter 4

Simulation of Speckle in Optical Imaging

In this chapter we examine methods for simulating speckle as it occurs in optical imaging. Unlike the 1D examples in previous chapters, the simulations in this chapter will be two-dimensional, since images are 2D entities. Discrete models must be used in the simulations. Thus, the field complex amplitudes must be represented by discrete arrays, and therefore sampling is inherent in the simulation. An important question concerns the sampling density that must be used in the various cases, with the density chosen high enough to avoid aliasing. According to the sampling theorem, the required density depends on the bandwidth of the complex amplitude representation of the light leaving the rough surface or diffuser.

In our previous 1D simulations of speckle, we have used independent diffuser phase samples, i.e., samples with no local correlations. A Fourier transform of a diffuser model with no phase correlations exhibits no bandwidth limitation and therefore results in aliasing. The aliasing is not always harmful. Aliasing of the circular complex Gaussian field samples results in another circularly symmetric complex Gaussian field, so the statistics of the field are not changed. Nonetheless, in some applications, particularly when a diffuser is sandwiched with an object that has its own structure, it is useful or even necessary to minimize the aliasing by introducing correlations between the diffuser phase samples, thereby reducing the diffuser bandwidth. Accordingly, we will introduce correlations between adjacent phase samples, which will reduce the bandwidth of the diffuser. The farther the correlations extend in the discrete diffuser phase array the smaller the resulting bandwidth of that diffuser.

It should be noted that even creating a completely bandlimited phase array (i.e., one with a sharp cutoff in spatial frequency space) will not create a perfectly bandlimited wave generated by the diffuser. This is because a perfectly bandlimited phase ϕ does not generate a perfectly bandlimited $\exp(j\phi)$ unless $\phi \ll 2\pi$. The nonlinearity of the complex exponential function generates frequency components that were not present in the phase sequence. However, bandlimiting the phase sequence does narrow the spectrum of the complex exponential, as we shall see. Note also that bandlimiting the complex exponential function (rather than the

The image of the diffuser is now a field of speckle with uniform statistics appropriate for fully developed speckle.

4.3 The Autocorrelation Function of Speckle Intensity

We turn now to a calculation of the autocorrelation of the speckle intensity, or more correctly stated, the autocovariance, since we will eventually subtract estimates of the mean intensity. The result will also be normalized to unity at the origin, thus yielding the normalized autocovariance of intensity. For circular complex Gaussian field statistics, the correlation function of intensity is given by

$$\Gamma_I(\Delta x, \Delta y) = \bar{I}^2(1 + |\mu(\Delta x, \Delta y)|^2), \tag{4-1}$$

where $\bar{I}$ is the average intensity across the image plane and μ is the normalized autocovariance function of the complex field, given by the van Cittert−Zernike theorem as a normalized Fourier transform of the intensity distribution I_p across the focal-plane aperture,

$$\mu(\Delta x, \Delta y) = \frac{\int\int I_p(\alpha,\beta) \exp(-j\frac{2\pi}{\lambda f}(\Delta x\alpha + \Delta y\beta)\, d\alpha\, d\beta}{\int\int I_p(\alpha,\beta)\, d\alpha\, d\beta}. \tag{4-2}$$

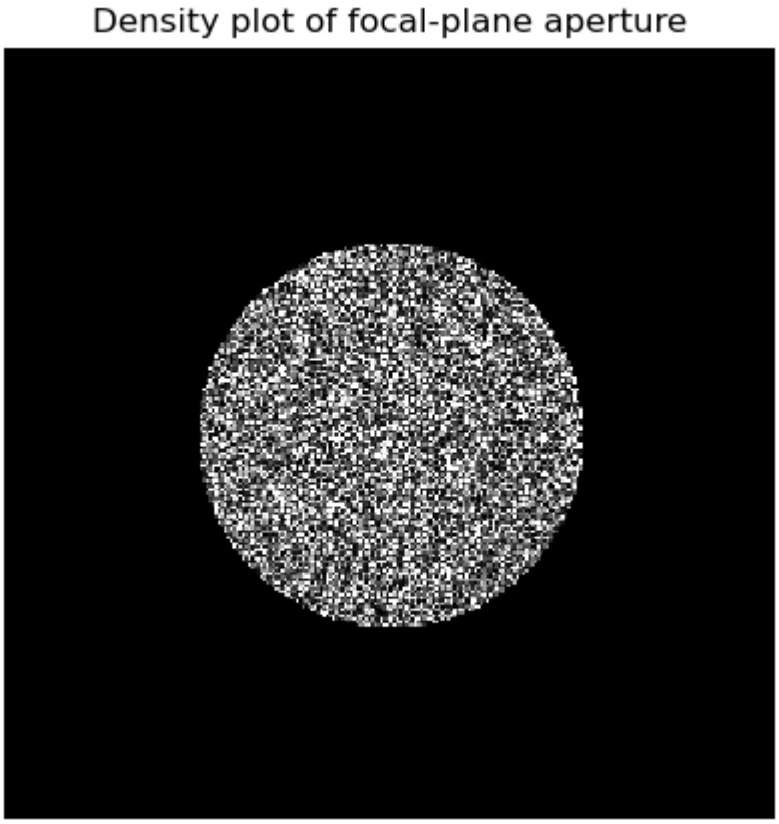
Here, the integration is over the entire (α, β) focal-plane coordinates, f is the focal length of the lens, and λ is the wavelength. The coordinates in the image autocovariance plane are the difference coordinates $(\Delta x = x_1 - x_2, \Delta y = y_1 - y_2)$, while (x, y) are coordinates in the image plane. For the ideal continuous case and a circular focal-plane aperture stop of diameter D, we have (see Ref. [2], Eq. (5-20))

$$\mu(\Delta x, \Delta y) = 2\frac{J_1\left(\pi\frac{D}{\lambda f}\sqrt{\Delta x^2 + \Delta y^2}\right)}{\pi\frac{D}{\lambda f}\sqrt{\Delta x^2 + \Delta y^2}}. \tag{4-3}$$

In the case of interest here, the intensity transmitted by the focal-plane stop is limited by a circle of diameter D but contains speckle, as seen in the density plot below, which is limited to a 256×256 subarray of the focal-plane array.

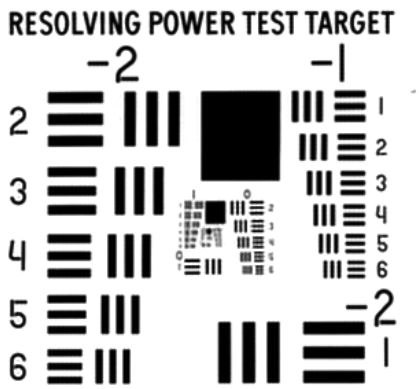
In [9]:
```
# Create a density image of the intensity transmitted by the
# focal-plane stop.
from matplotlib import pyplot as plt
import numpy as np
```

```
# Set the plotting parameters.
x = np.linspace(-1,1,256)
y = np.linspace(-1,1,256)
X, Y = np.meshgrid(x,y)

# Calculate the transmitted intensity in a subarray.
ip = abs(transmitted_field[896:1152,896:1152])**2
Z = ip/np.max(ip)

# Create a density plot of the intensity subarray.
plt.rcParams["figure.figsize"] = [5,5]
plt.title('Density plot of focal-plane aperture')
plt.axis('off')
plt.imshow(Z,vmin = 0,vmax = 0.2,cmap = 'gray', interpolation = 'nearest')

plt.show()
```

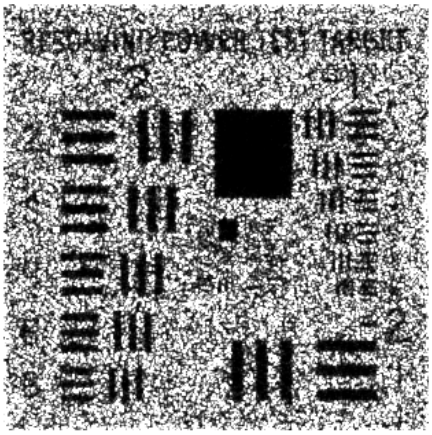


Our goal here is to perform a calculation that will yield a discrete approximation to the intensity covariance function in the presence of speckle. The code below performs this calculation, with the simplification that we assume that $\lambda f = 1$. According to Eq. (4.2) above, we can Fourier transform the intensity passed by the focal-plane stop and, after normalizing the result by the sum of those intensities, we obtain the field autocovariance μ.

No diffuser



With a diffuser



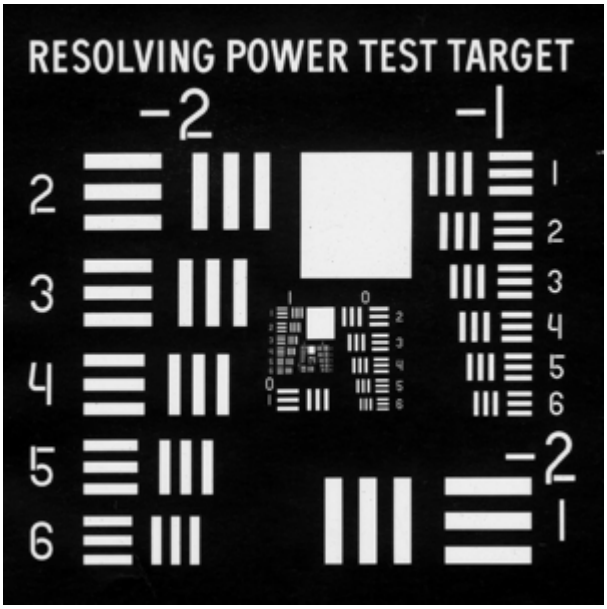
In both images above, we have chosen the grayscale extent (through the choice of **vmax**) to optimize our view of the bar-triplets in the resolution chart. On the left is the image when no diffuser is present and on the right is the image obtained with a diffuser. When no diffuser is present, we can resolve most of the bar-triplets in the group with label "0." However, when the diffuser is present, we cannot resolve any of the bar-triplets in that group. Thus, the presence of speckle has certainly reduced our ability to resolve small structures for this object and this choice of a stop diameter.

White Structure on an Opaque Background

We now consider the reverse case in which the structure to be resolved is white and the background is black. Is there a difference between the effects of speckle just shown and the effects of speckle in this case? First we define the new object to be imaged.

In [29]:
```
# Define the inverted image and display a small version of it.
newim3 = ImageOps.invert(newim1)
thumbimage3 =  newim3.resize((300,300));thumbimage3
```

Out[29]:



We again calculate an image with no diffuser present.

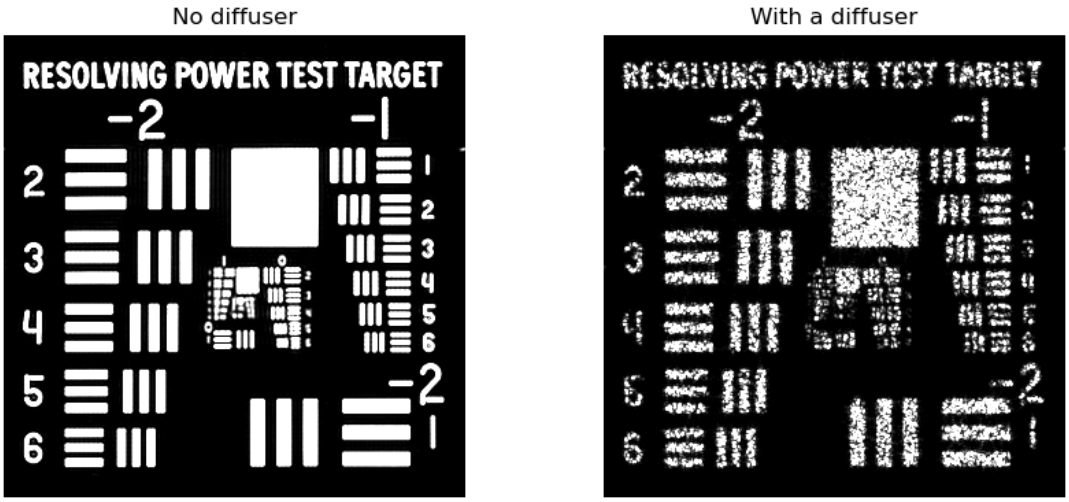
In [30]:
```
# Pass the field incident on the focal plane through the aperture with
# no diffuser present.
incidentfield3 = sfft.fftshift(sfft.fft2(newim3))
transmittedfield3 = incidentfield3 * stop

# Fourier transform the field transmitted through the focal plane
# to reach the image plane.
imagefield3 = sfft.ifft2(transmittedfield3)

# Calculate the image intensity.
imageintensity3 = np.abs(imagefield3)**2

# Convert the np.array to "image" format.
normalizedimage3 = (imageintensity3/np.max(imageintensity3))*255
img3 = Image.fromarray(normalizedimage3.astype(np.uint8))
```

Now we include the diffuser in the object plane.

In [31]:
```
# Pass the field incident on the focal plane through the
# focal-plane aperture.
# Include the diffuser.
incidentfield4 = sfft.fftshift(sfft.fft2(newim3*diffuser))
transmittedfield4 = incidentfield4 * stop

# Fourier transform the field transmitted through the focal plane
# to reach the image plane.
imagefield4 = sfft.ifft2(transmittedfield4)
imageintensity4 = np.abs(imagefield4)**2

# Convert the image to "image" format.
normalizedimage4 = (imageintensity4/np.max(imageintensity4))*255
img4 = Image.fromarray(normalizedimage4.astype(np.uint8))
```

We plot the two images side-by-side, on the left with no diffuser present and on the right with a diffuser present.

In [32]:
```
# Set the plotting parameters.
from matplotlib import pyplot as plt
from PIL import Image
plt.rcParams["figure.figsize"] = [10.0, 10.0]
plt.subplots(1,2)
plt.subplots_adjust(wspace = 0.3,hspace = 0.1)

# Plot the two images side-by-side. We set the parameters vmax in
# both images to yield the best results.
plt.subplot(1,2,1)
plt.axis("off")
plt.title('No diffuser')
plt.imshow(img3,vmin = 0,vmax = 20,cmap = 'binary_r')

plt.subplot(1,2,2)
plt.axis("off")
plt.title('With a diffuser')
plt.imshow(img4,vmin = 0,vmax = 10,cmap = 'binary_r')

plt.show()
```

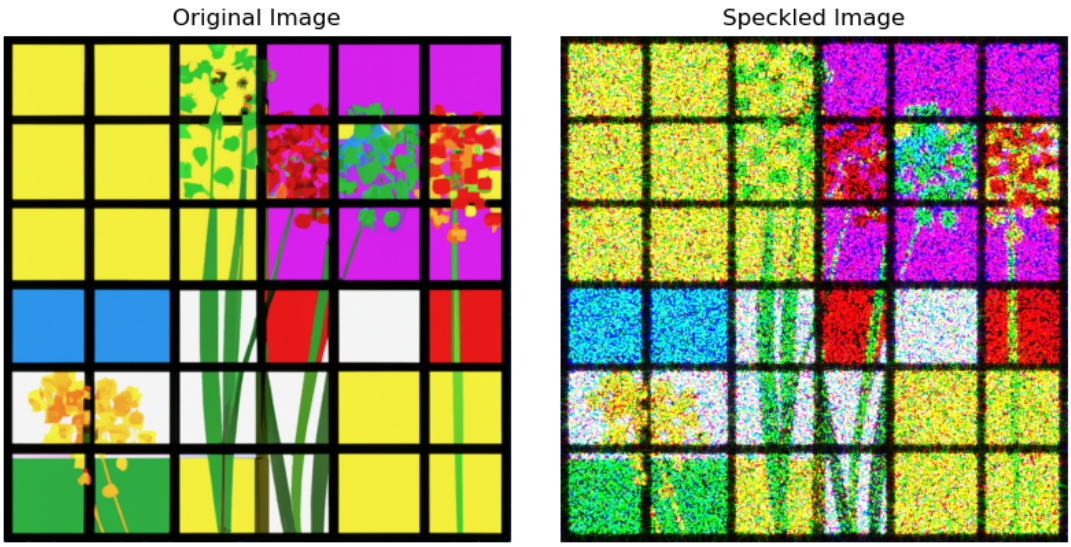


When no diffuser is present, we can resolve the top two sets of vertical and horizontal bars in group "0." When the diffuser is present, it is very difficult to resolve those same sets of bars. However, if we compare the transparent-background object with the opaque-background object, resolution is worse in the transparent-background case. There is some advantage to having a transparent structure on an opaque background, rather than the reverse case.

```
# Merge the arrays into a color image.
color_array = np.dstack((Rimageintensity, Gimageintensity, \
                         Bimageintensity));
```

In [81]:
```
# Set the plotting parameters.
plt.rcParams["figure.figsize"] = [10.0, 10.0]
plt.subplots(1,2)
plt.subplots_adjust(wspace = 0.1,hspace = 0.1)

# Plot the two images side-by-side.
plt.subplot(1,2,1)
plt.axis("off")
plt.title('Original Image')
plt.imshow(RGBcolor/np.max(RGBcolor))

plt.subplot(1,2,2)
plt.axis("off")
plt.title('Speckled Image')
plt.imshow(50*color_array/np.max(color_array))

plt.show()
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
```



The warning message arises because we have exceeded the range $(0, 1)$ in the image color array components in order to make the speckled image brighter and more visible. The multiplication factor 50 brings the speckled image into an appropriate range for **imshow( )**.

A close examination of the colored specked image reveals some interesting facts. First, the red speckle lobes do seem to be slightly wider than the green or blue speckle lobes, as they should be. Second, because the wavelength separations are wide enough to generate independent speckle patterns in the three colors, one can

distinguish what mixtures of colors were used for various regions of the original object. For example, in the yellow portion of the upper left of the speckled image, we see a mixture of mostly red and green speckle, with overlapping speckles of red and green producing yellow speckle. In the white region, we see speckles of all colors. The magenta squares in the upper right have a mixture of red and blue speckles. It's also interesting that in all of the squares, the RGB speckles overlap enough to maintain a rendering of the original colors.

Chapter 5

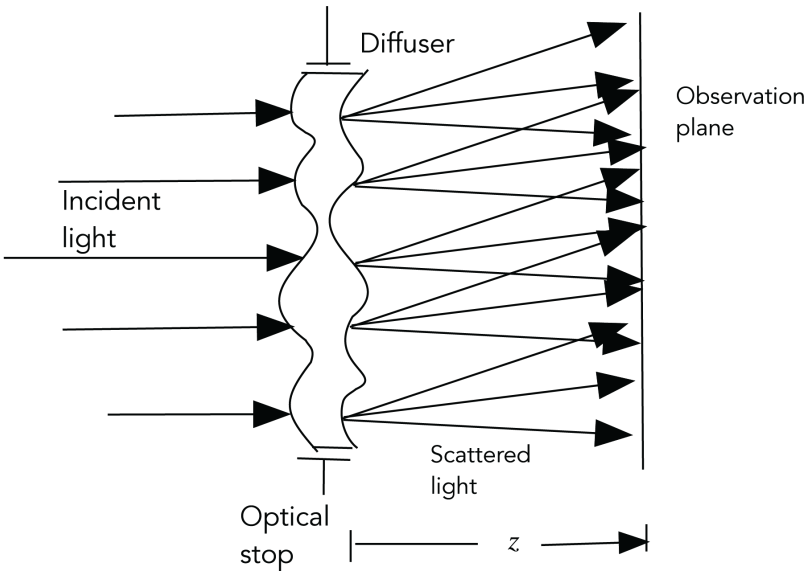
Simulation of Speckle in Free-Space Propagation

Calculating the diffraction patterns of clear apertures of various shapes is a common task. However, similar calculations for apertures with a complex internal structure are less common. In the case studied here, the internal structure of a rectangular aperture is a random pattern of phase representing a diffuser. To perform the simulation of propagation, choices must be made for the number of samples within the aperture (the parameter M in what follows) and the total number of samples in the simulation (the parameter N in what follows), as well as the method of calculation. Guidelines for selecting these parameters and methods for clear apertures do not apply in this more complex case.

The geometry assumed for these simulations is shown in the figure below.

In [1]:
```
# Import library.
from IPython.display import Image

# Load image from local storage.
Image(filename = "freespace.png", width = 400, height = 100)
```

Out[1]:



A normally incident plane wave from a highly coherent source, indicated by the arrows on the left, is incident on a diffuser, and light is scattered in many directions.

Chapter 6

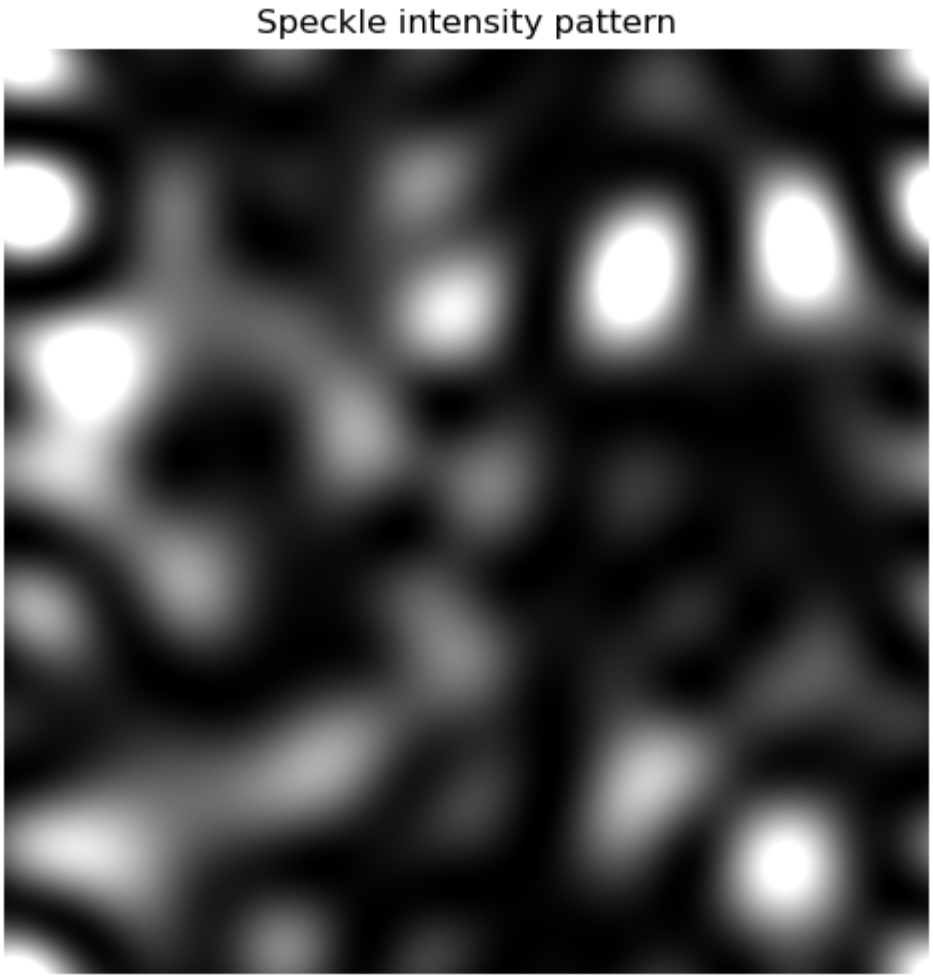
Speckle at Low Light Levels

In this chapter we explore the properties of speckle when detected at low light levels. The model used is semiclassical, based on the assumption that light of uniform intensity I incident on a detector pixel results in a Poisson distribution of photoelectrons with an average number of photoevents proportional to the integral of I. Thus, if $n(x, y)$ represents the number of photoevents occurring from the detector pixel centered at coordinates (x, y), the mean number of such photoevents is given by

$$\overline{n(x,y)} = \frac{\eta}{h\nu} \int\int_A I(x,y)\, dxdy, \tag{6-1}$$

where η is the quantum efficiency of the detector, h is Planck's constant, ν is the optical frequency, and A is the area of the photodetector pixel. The multiplier of the above integral represents the efficiency with which integrated intensity is converted into numbers of photoevents (see Ref. [2], Section 8.2 for a more detailed discussion of this subject). Thus, in simulations, an array of intensity values incident on an array of detector pixels generates an array of integer numbers representing photocounts. If the intensity varies from detector element to detector element, as it does when the incident light comprises a speckle pattern, the counts will have varying means at each detector element. The number of elements in the intensity array in general will be greater than the number of elements in the photodetector array, since detector pixels may integrate over several speckles. If the intensity across a given detector element is constant, the statistics of the number of photoevents will obey a Poisson distribution. On the other hand, if the intensity across a given detector element is not constant, as could be the case for an incident speckle pattern, the photoevent statistics are modified, as described below.

To illustrate the generation of Poisson-distributed counts from a detector array, for simplicity, we start with a uniform intensity over all elements of the array (i.e., no speckle present) and generate independent Poisson photocounts with identical means at all elements of the array. To do so requires partitioning the incident intensity array of $N \times N$ pixels into $N/Q \times N/Q$ super-pixels, each representing one detector element. Within one super-pixel there are $Q \times Q$ samples of the



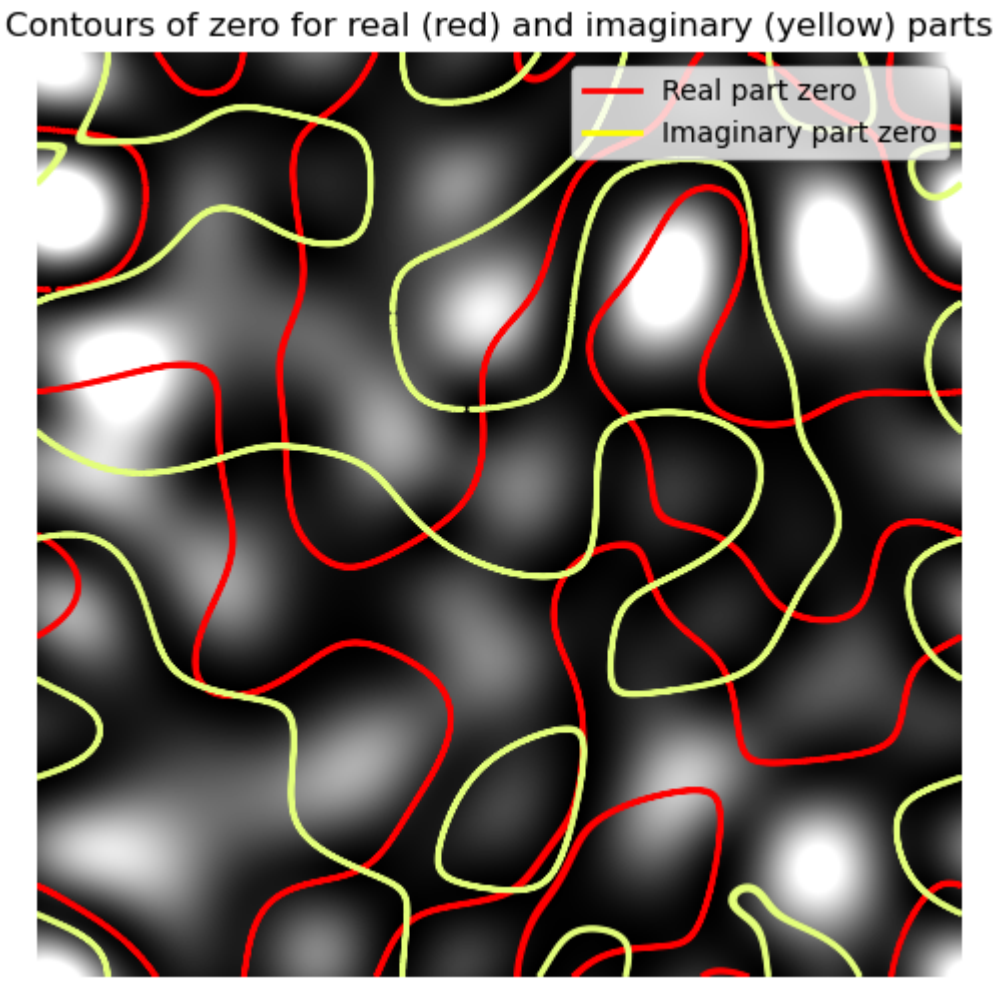
This is the speckle intensity pattern for which we will study the properties of the phase in the vicinity of the zeros of intensity.

7.2 Finding the Zeros of Intensity

When the real and imaginary parts of the complex amplitude are simultaneously zero, the intensity is zero. Below we generate two contours, one (red) where the real part of the complex amplitude is zero or close to zero, and one (yellow) where the imaginary part is zero or close to zero.

In [11]:
```
# Plot in the same figure the Fourier intensity and the zero contours
# of the real and imaginary parts of the Fourier amplitudes.

fig, ax = plt.subplots()

# Set the figure size.
plt.rcParams["figure.figsize"] = [6.0, 6.0]
```

```
# Show the speckle intensity.
im = ax.imshow(Fourierintensity,vmin = 0,vmax = 250,cmap = 'gray')

# Contours are defined when the absolute value of the real and imaginary
# parts are within 0.03 of zero.
contourR = ax.contour(abs(ZR),levels = [.04],cmap = 'autumn',
  linewidths = 2)
contourI = ax.contour(abs(ZI),levels = [.04],cmap = 'Wistia',
  linewidths = 2)

plt.axis('off')
plt.title('Contours of zero for real (red) and imaginary (yellow) parts')

# Include a legend in the plot.
legend_elements = [
    plt.Line2D([0], [0], color = 'red', lw = 2, label = 'Real part zero'),
    plt.Line2D([0], [0], color = 'yellow', lw = 2,
          label = 'Imaginary part zero')]
ax.legend(handles = legend_elements, loc = 'upper right')

plt.show()
```



When the real and imaginary parts are simultaneously zero, the intensity is zero. Thus, the points where the red and yellow contours cross are zeros of the intensity.

7.3 The Phase Behavior in the Vicinity of the Zeros of Intensity

To explore the speckle phase in regions close to the zeros of intensity, we create a contour plot of the phase for eight discrete values of phase between –3π/4 and π. The quantity **mylevels** is a list containing the eight values of phase for which we desire contours. In order to interpret the contour plot, we superimpose that plot and the contours of zero of the real and imaginary parts so that we can easily identify where the zeros of intensity occur.

In [12]:
```
# Extract the phase of the Fourier amplitude. The phase values lie
# between -π and π.
phase = np.angle(Fourieramplitude)
```

Now we generate a shaded contour plot of the phase, also superimposing a plot of the contours of the zero real and zero imaginary parts. Calculating the phase contours takes some time, so be patient.

In [13]:
```
import numpy as np
import matplotlib.pyplot as plt

# Prepare to calculate and superimpose the images.
fig, ax = plt.subplots()

# Define the levels of phase for which we want contours.
mylevels = [-np.pi,-3*np.pi/4,-2*np.pi/4,-np.pi/4,0,np.pi/4,2*np.pi/4,
  3*np.pi/4,np.pi]

x = np.linspace(0,N,N)
y = np.linspace(0,N,N)
X, Y = np.meshgrid(x,y)
Z = phase

# Calculate the contour map of the phase.
cax = plt.contourf(X,Y,Z,levels = mylevels,vmin = -np.pi,vmax = np.pi,
  cmap = 'binary')

# Again, calculate the contours of the zeros of the real and imaginary
# parts of the amplitude.
ax.contour(abs(ZR),levels = [.03],cmap = 'autumn',linewidths = 2)
ax.contour(abs(ZI),levels = [.03],cmap = 'Wistia',linewidths = 2)

plt.axis('off')
plt.title('Phase contour plot')
```

Chapter 8

Polarization Speckle

The discussions of speckle in the previous chapters have implicitly assumed that the light in the speckle pattern is perfectly polarized. In many practical applications, especially when multiple scattering is present, the light scattered by a rough object or a diffuser may be partially or totally depolarized. In fact, just as the intensity of the light varies across the speckle pattern, so too the state of polarization may vary. This random fluctuation of polarization has become known as *polarization speckle*. For a more detailed discussion of this topic, see Ref. [5].

8.1 The Polarization Ellipse and the Degree of Polarization

A useful description of the state of polarization is the polarization ellipse, which is a graphical representation of the path followed by the tip of the electric vector through one oscillation cycle. The ellipse is characterized by the parameter a representing the major semi-axis, b representing the minor semi-axis, and ψ representing the angle of the major axis with respect to the x axis. See the example figure below and the code that generates it. In this case, $\psi = \pi/6$, $a = 0.8$, and $b = 0.4$.

In [1]:
```
import matplotlib.pyplot as plt
from matplotlib.patches import Ellipse

# Create a figure and axes.
fig, ax = plt.subplots()

# Set the figure size.
plt.rcParams["figure.figsize"] = [8.0, 8.0]

# Create an ellipse.
ellipse = Ellipse(xy = (0.5, 0.5), width = 0.8, height = 0.4, angle = 30,
  fill = False)

# Define the x and y coordinates of the major axis of the ellipse.
x1 = [0.16, 0.84]
y1 = [0.3,0.7]

# Define the x and y coordinates of the minor axis of the ellipse.
x2 = [0.6,0.38]
```

Chapter 9

Speckle Simulation for Metrology

In most imaging applications, speckle is a nuisance, and various methods are used to attempt to suppress it. However, in the field of metrology, speckle can be a friend rather than a foe, and many methods for using speckle in measurement have been devised. Here we will simulate only a few of such methods; the interested reader is referred to Ref. [2], (Chapter 9) for a more comprehensive discussion of this topic, together with references to the pertinent original publications.

9.1 Measurement of In-Plane Displacement

An early application of speckle to metrology was for measurement of lateral in-plane displacement. Assume that we are imaging a finite region on a rough object and we observe speckle in the image. The object then moves laterally, and we wish to use a second exposure to determine the amount of lateral movement. As the object translates, the speckle in its image translates as well, but with some change as new scatterers move into the fixed illuminated region and previous scatterers move out of the illuminated region. In this simulation, the diffuser represents a rough surface, the translation of which we aim to measure.

We assume that the optical imaging system is a $4f$ system of the same type we have used in other chapters. The rough object lies in the front focal plane of a positive lens with focal length f. In the rear focal plane of that lens there is an aperture, centered on the optical axis, that restricts the area through which light can pass. At one further focal length, a second positive lens, again with focal length f, captures the light passed by the aperture and passes it on to its rear focal plane where a filtered image of the rough surface is found.

First, we generate a diffuser representing the rough object that will be laterally translated. Again, we smooth the diffuser phase so that it has a finite phase correlation length. Next, we define a finite window corresponding to the finite illumination spot on the diffuser. This window remains fixed while the diffuser moves under it. The number of pixels in the simulation is $N \times N$, P represents the

Appendix A

Some Subtleties in Simulation With the *4f* Imaging System

A.1 Effects on the Speckle Contrast

As pointed out in Section 1.3, there are some subtleties associated with the generation of simulated speckle using the $4f$ imaging system that arise from the finite size of the arrays used in the simulation. The phenomenon involves a complicated relationship between the size of the arrays used and the diameter of the focal-plane stop. It is possible to choose these parameters in such a way that the contrast of the speckle (i.e., the standard deviation of intensity normalized by the mean intensity) never achieves its ideal value of unity.

Two effects can be identified. To understand these effects, it is helpful to consider how much of the diffuser contributes to the value of intensity at any single point in the image plane. With a finite focal-plane aperture diameter, there is a weighting function on the diffuser that averages over a finite region, adding complex-valued sample points that contribute to the image amplitude and intensity at a point. The smaller the focal-plane aperture the broader that weighting function on the diffuser becomes; and the larger the diameter of the aperture the narrower that weighting function becomes.

The sum of the complex phasors within that averaging region generates a new complex phasor having a length and phase determined by interference between the complex diffuser pixels lying within the averaging region. If the focal-plane aperture is one pixel in diameter, a single spectral sample passes the focal plane, and the intensity in the image plane is constant, with contrast equal to zero for any single diffuser. If the focal-plane aperture has a diameter $\sqrt{2}N$ or larger (this circle size covers the corners of the rectangular sample array), the extent of the weighting function on the diffuser will be one pixel, covering only one phase cell of the diffuser. The result is a pure phase image with contrast zero.

Thus, we see that in the limits of small or large focal-plane apertures, the speckle contrast in the image can be reduced, either due to the small number of phasors contributing to image intensity at an image point in the former case, or due to

Appendix B

Illustration of How to Group Pixels into Super-pixels

This appendix explains how to group pixels in a given 2D array into super-pixels, each containing multiple pixels, while maintaining the proper ordering of pixels. Creating super-pixels should group adjacent pixels in both the horizontal and vertical dimensions. If we grouped pixels in super-pixels only along the x axis row-by-row, this would be analogous to having a detector with elements that are long horizontally and thin vertically. Our goal is to properly group the pixels for square detector elements.

B.1 A Simple Example

To make the demonstration as clear as possible, we use small arrays for illustration. We start with an 8×8 array and reduce it to a 2×2 array of super-pixels, each containing 4×4 elements.

In [1]:
```
import numpy as np

# Size of original array is N x N.
N = 8

# Size of a super-pixel is 4 x 4.
Q = 4

lineararray = np.zeros(N**2)
squarearray = np.zeros([N,N])

# Create a flattened NumPy array with the original values.
lineararray = np.arange(N**2)
squarearray = lineararray.reshape(N,N)

# Show the square array.
squarearray
```

References

1. J.W. Goodman, *Simulating Speckle with Mathematica®*, SPIE Press, Bellingham, WA (2022).
2. J.W. Goodman, *Speckle Phenomena in Optics: Theory and Applications, Second Edition*, SPIE Press, Bellingham, WA (2020).
3. J.M. Kinder and P. Nelson, *A Student's Guide to Python for Physical Modeling, Second Edition*, Princeton University Press, Princeton, NJ (2015).
4. C. Hill, *Learning Scientific Programming with Python*, Cambridge University Press, Cambridge, UK (2015).
5. W. Wang, S.G. Hanson, and M. Takeda, "Statistics of polarization speckle: theory versus experiment," *Proc. SPIE* **7388**, 738803 (2009).
6. R.A. Chapman, W.-S.T. Lam, and G. Young, *Polarized Light and Optical Systems*, CRC Press, Boca Raton, FL (2019).
7. J.N. Butters, "Speckle pattern interferometry using video techniques," *Opt. Eng.* **10**(1), 100105 (1971).
8. K. Creath, "Phase-shifting speckle interferometry," *Appl. Opt.* **24**, 3053-3058 (1985).
9. S. Nakadate and H. Saito, "Fringe scanning speckle-pattern interferometry," *Appl. Opt.* **24**, 2172-2180 (1985).
10. J. Wyant, https://wp.optics.arizona.edu/jcwyant/wp-content/uploads/sites/13/2016/08/Phase-Shifting-Interferometry.nb_.pdf
11. D.C. Ghiglia and M.D. Pritt, *Two Dimensional Phase Unwrapping: Theory, Algorithms and Software*, Wiley Interscience, New York (1998).