LightPipes for Matlab

Beam Propagation Toolbox Manual

version 1.3

Toolbox Routines:
Gleb Vdovin,
OKO Technologies
Reinier de Graafweg 300
2625 DJ Delft
The Netherlands
fax: +31-15-256-9361
Email: gleb@okotech.com
WWW: http://www.okotech.com
(c)1993--1996, Gleb Vdovin

Matlab mex files:
Fred van Goor,
University of Twente
Department of Applied Physics
Quantum electronics group
P.O. Box 217, 7500 AE Enschede
The Netherlands
Email: F.A.vanGoor@tn.utwente.nl
fax: +31-53-4891102
(c)1998, Fred van Goor

1 Introduction

LightPipes for Matlab is a set of functions written in C available to Matlab. It is designed to model coherent optical devices when the diffraction is essential. The toolbox consists of a number of functions. Each function represents an optical element or a step in the light propagation. There are apertures, intensity filters, beam-splitters, lenses and models of free space diffraction in LightPipes. There are also more advanced tools for manipulating the phase and amplitude of the light. Also problems involving polarization can be simulated. The program operates on a large data structure, containing square two-dimensional arrays of complex amplitudes of the optical field of the propagating light beam.
The LightPipes for Matlab routines are modifications of the LightPipes C routines written by Gleb Vdovin for Unix, Linux, DOS and OS2 workstations or PC. There is also a version for Mathcad available. The Matlab (and the Mathcad) version of LightPipes has a number of advantages:

- The graphics-, animation- and other features of Matlab can be combined with the LightPipes commands.
- You can use variable arguments in the function calls and handle complex data structures in a very simple way.
- Enhanced flexibility and fast execution.

Most of the commands of LightPipes for Matlab are the same as the Unix/DOS version. Only the commands handling the in- and output of the results to disk and the plot commands have been skipped as one can use the build-in commands of Matlab for disk communications and graphics with more advantage. The names of the commands are proceeded with the letters "LP" in order to prevent the confusion with existing Matlab or other functions (m-files). The arguments are tested during execution of the command and error messages appear in the case of bad arguments. Also on-line help is available by simply typing the command without arguments or through the Matlab help facilities.

2 Warranty

3 Availability

The LightPipes for Matlab package for beam propagation is copyright (c)(1993--1996) of Gleb Vdovin and (c)(1998) of Fred van Goor. This document (also (c) of Gleb Vdovin and Fred van Goor) may be freely distributed together with the demo executables. No part of this document can be reproduced without written permission of the authors. A demo version of LightPipes for Matlab with a limited functionality of 64x64 grid dimension and this manual are freely available from:
WWW: http://www.okotech.com
LightPipes for Matlab with unlimited grid dimension (limited by your computer memory) is available from:

OKO Technologies,
Reinier de Graafweg 300,
2625 DJ Delft,
The Netherlands.
fax: +31-15-256-9361
Email: gleb@okotech.com
WWW: http://www.okotech.com

4 Installation of LightPipes for Matlab

4.1 System requirements.

LightPipes for Matlab must be installed on a system with the following hardware and software:

Hardware:

1. An 80486 or Pentium based IBM or compatible computer.
2. At least 16 megabytes of memory.
3. A hard disk with at least 10 megabytes of free space.

Because the calculations require large arrays of data we recommend 32 megabytes of memory and a fast processor like a 150MHz Pentium or better.

Software:

Microsoft Windows 95.
Matlab version 5.1.

4.2 Installation

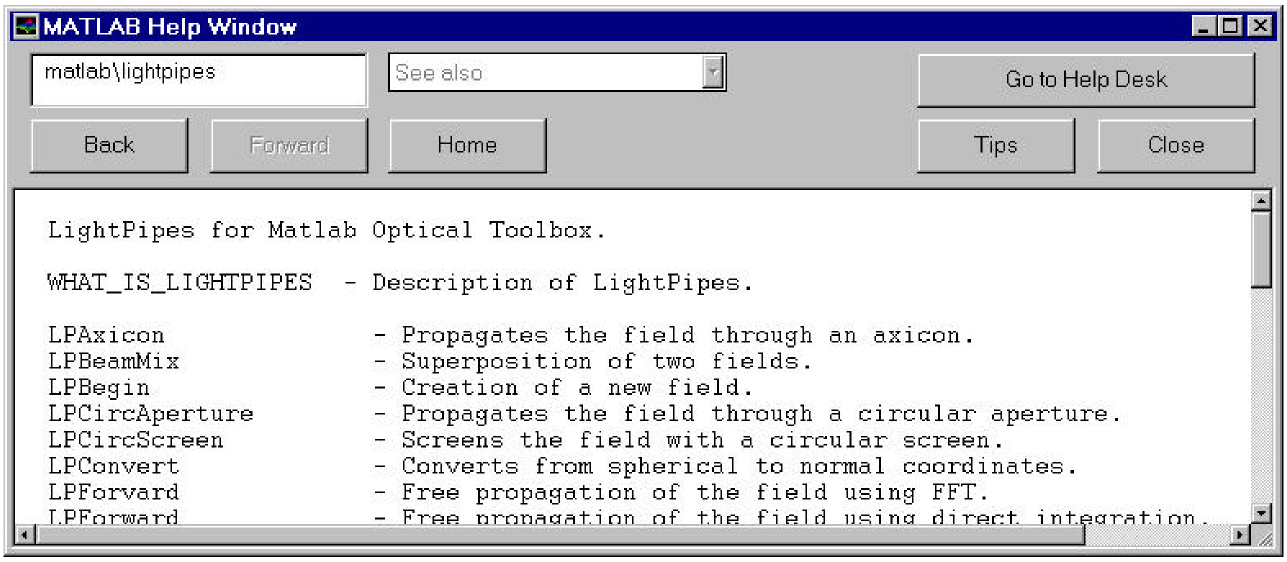
LightPipes for Matlab can be installed on your system by copying the file 'MatlabDLL.zip' with the dll's (mex-files) to a directory of your choice. Unpack this file to extract the mex-files. Within Matlab, run File, Set Path..., Add to Path... to add the directory containing the LightPipes mex-files to the Matlab's path. Save the settings by clicking Save Settings and leave Matlab and start it again. From now on the LightPipes commands and the help facilities are available to the user. Test the installation by typing 'help WHAT_IS_LIGHTPIPES' in the command window. If you are not in the directory where you put the LightPipes dll's in you can test the on-line help by clicking: Help, Help Window. The MATLAB Help window appears. The item 'LightPipes for Matlab Optical Toolbox' should be visible in the list. Double-click on this item and a list of all the LightPipes commands should appear. Double-click on one of the commands to see a short description of the command. (If the LightPipes directory is the current directory you get the message "Topic '******\LightPipes' was not found". Chance to another directory and try again).
You can remove LightPipes for Matlab from your sytem by simply deleting the LightPipes directory you created and its contents.

5 LightPipes for Matlab description

5.1 First steps

5.1.1 Help

Help can be obtained by clicking on Help, Help Window in the command window of Matlab. Then the MATLAB Help Window appears on which you can double-click on the 'LighPipes for Matlab Optical Toolbox' item. A list of all the LightPipes for Matlab commands (Beginning with the letters 'LP') with a short description will appear:

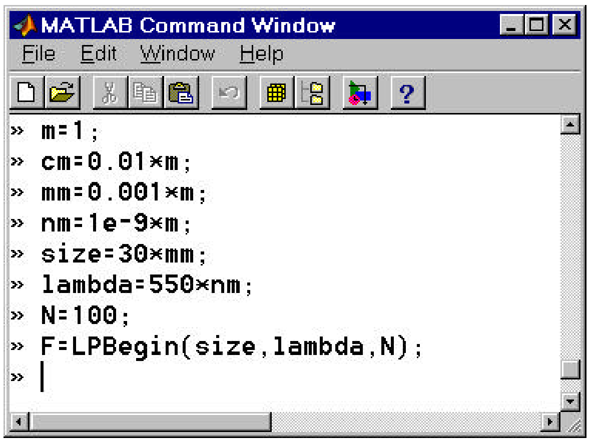


```
LightPipes for Matlab Optical Toolbox.

WHAT_IS_LIGHTPIPES  - Description of LightPipes.

LPAxicon             - Propagates the field through an axicon.
LPBeamMix            - Superposition of two fields.
LPBegin              - Creation of a new field.
LPCircAperture       - Propagates the field through a circular aperture.
LPCircScreen         - Screens the field with a circular screen.
LPConvert            - Converts from spherical to normal coordinates.
LPForvard            - Free propagation of the field using FFT.
LPForward            - Free propagation of the field using direct integration.
```

Figure 1 The Matlab help window showing a list of the LightPipes commands.

5.1.2 Starting the calculations

All the calculations must start with the LPBegin function. This function defines the size of the square grid, the grid dimension and the wavelength of the field. You can type the commands in the command window of Matlab after the prompt:



```
» m=1;
» cm=0.01*m;
» mm=0.001*m;
» nm=1e-9*m;
» size=30*mm;
» lambda=550*nm;
» N=100;
» F=LPBegin(size,lambda,N);
» 
```

Figure 2 The use of LPBegin to define a uniform field with amplitude 1 and phase 0.

A two dimensional array will be defined containing the complex numbers with the real and imaginary parts equal to one and zero respectively. The minimal dimension of the grid must be 8x8 and the maximum will be determined by your computer memory (or 64x64 if you have the demo version of LightPipes for Matlab). The grid dimension must be an even number. An extra column is added to the field array to carry information, such as the grid dimension, grid size and wavelength, with the field.

5.1.3 m-files

A better alternative is to write m-files or scripts in stead of typing the commands in the command window. See the Matlab documentation for details to write m-files. In this manual we have indicated the m-files of the examples by gray-shading them:

```
% This is an example of an m-file.
command(1);
command(2);
  .
  .
  .
command(n);
```

5.1.4 The dimensions of structures

The field structures in LightPipes for Matlab are two-dimensional arrays of complex numbers. For example a grid with 256x256 points asks about 1Mb of memory. 512x512 points ask more then 4Mb and 1024x1024 16Mb. Some commands, however need more memory because internal arrays are necessary. LightPipes for Matlab works fast and without disk swapping with arrays up to 256x256 points for a Pentium PC with 32Mb RAM memory and running under Windows'95.

5.1.5 Apertures and screens

The simplest component to model is an aperture. There are four different types of apertures:

- The circular aperture: LPCircAperture(R, x_s, y_s, Field)

- The rectangular aperture: LPRectAperture(w_x, w_y, x_s, y_s, φ, Field)
- The Gaussian diaphragm: LPGaussAperture(R, x_s, y_s, T, Field)
- The Super Gaussian Aperture: LPSuperGaussAperture(R,n,x_s,y_s,T,Field)

Where R=radius, x_s and y_s are the shift in x and y direction respectively, φ is a rotation and T is the centre reflection of the (super)Gaussian aperture. n= the power of the super Gaussian. In addition, there are three commands describing screens: LPCircScreen (inversion of the circular aperture), LPRectScreen, (inversion of the rectangular aperture) LPGaussScreen (inversion of the gauss aperture). and LPSuperGaussScreen (inversion of the super-gauss aperture). Figure 3 shows an example of the usage of a circular aperture.

```
%man0001.m
%
%LightPipes for Matlab
%Circular aperture
clear;
m=1;
mm=0.001*m;
nm=1e-9*m;
size=10*mm;
lambda=500*nm;
N=128;
R=2.5*mm;
xs=1*mm;
ys=-1*mm;
Field=LPBegin(size,lambda,N);
Field=LPCircAperture(R,xs,ys,Field);
Intensity=LPIntensity(1,Field);
figure(3);
imshow(Intensity);
flnm=sprintf('(man0001.emf)');flnm=strcat('\it',flnm);
title('Insertion of a circular aperture in the field');
text(N-20,N+30,flnm);
print -dmeta '..\figures\man0001';
```

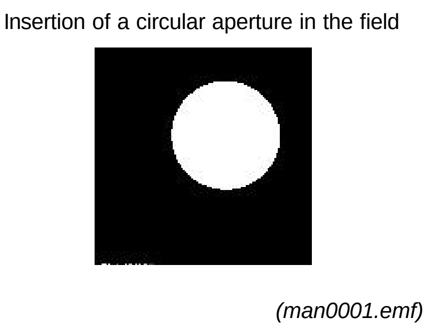


Figure 3 Example of the application of a circular aperture.

```
%man0002.m
%
%LightPipes for Matlab
%Demonstration of the usage of screens and apertures.
clear;
m=1;
mm=0.001*m;
nm=1e-9*m;
size=10*mm;
lambda=1000*nm;
N=250;
Field=LPBegin(size,lambda,N);
Field=LPRectScreen(0.001,0.001,-0.0015,-0.002,0,Field);
Field=LPCircScreen(0.0007,0.001,0.0015,Field);
Field=LPGaussAperture(0.004,0,0,1,Field);
Field=LPRectScreen(0.001,0.0035,-0.002,0.0025,30,Field);
Intensity=LPIntensity(1,Field);

figure;
imshow(Intensity);
flnm=sprintf('(man0002.emf)');flnm=strcat('\it',flnm);
title('various screens and apertures');
text(N-20,N+30,flnm);
print -dmeta '..\figures\man0002';
```

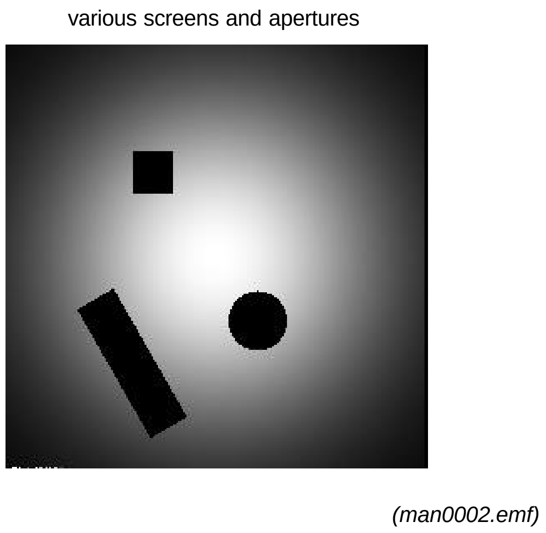


Figure 4 The use of screens and apertures.

All kinds of combinations of circular, rectangular and Gaussian apertures and screens can be made as illustrated in Figure 4. The use of units in the simulations is recommended because it is very convenient and it reduces the chance on errors.

5.1.6 Graphing and visualisation.

In Figure 3 and Figure 4 the intensity of the field is calculated after aperturing 'Field' with the LPIntensity command using option 1, which means that a bitmap with gray values (0...1) is produced which can be displayed with the 'imshow' command of Matlab. Other powerful graphic features of Matlab can be applied for displaying the intensity or phase distributions including coloured contour plots, surface plots, etc.. With the command LPInterpolate you can reduce the grid dimension of the field speeding up subsequent plotting considerably. This is illustrated in The use of interpolation will be explained in more detail in section 5.2.6. In Figure 5 we also plotted the cross section of the beam in a two-dimensional XY plot. For this we have to define an integer, i, ranging from 1 to the grid dimension. This integer must be used to define the element of the (square) array, 'Intensity', which is used for the vertical axis of the XY plot. You can also use it to define a vector x for the horizontal axis. Refer to the Matlab documentation for more useful plot commands.

```
%man0003.m
%
%LightPipes for Matlab
%cross section and mesh plots.
%usage of LPInterpol to reduce the grid dimension.
m=1;
mm=0.001*m;
nm=1e-9*m;
size=10*mm;
lambda=500*nm;
N=128;
R=2.5*mm;
xs=0*mm;
ys=0*mm;
T=0.8;
Field=LPBegin(size,lambda,N);
Field=LPGaussAperture(R,xs,ys,T,Field);
Intensity=LPIntensity(1,Field);
i=[1:N];
x(i)=-size/2+i*size/N;
figure(5);
subplot(1,2,1);
plot(x(i)/mm,Intensity(N/2,i),'k');
xlabel('x [mm]');
ylabel('Intensity');
axis([-size/1.8/mm, size/1.8/mm, 0, 1.2]);
axis square;
Field=LPInterpol(size,N/4,0,0,0,1,Field);
Intensity=LPIntensity(1,Field);
```

```
subplot(1,2,2);
mesh(Intensity);
axis square;
axis off;
flnm=sprintf('(man0003.emf)');flnm=strcat('\it',flnm);
title('Graphing and Visualization');
text(-N/4,-N/4,flnm);
print -dbitmap '..\figures\man0003';
```

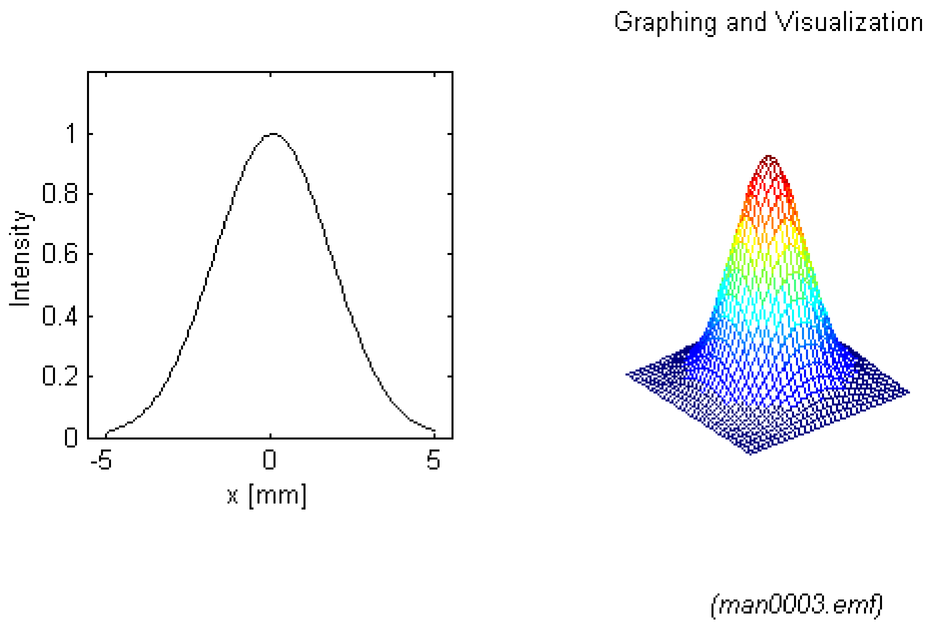


Figure 5 The use of XY-plots and surface plots to present your simulation results. Note the usage of LPInterpol to reduce the grid dimension for a faster and nicer surface plot.

5.2 Free space propagation

There are five different possibilities for modelling the light propagation in LightPipes for Matlab.

5.2.1 FFT propagation (spectral method).

Let us consider the wave function U in two planes: U(x,y,0) and U(x,y,z) . Suppose then that U(x,y,z) is the result of propagation of U(x,y,0) to the distance z, with the Fourier transforms of these two (initial and propagated) wave functions given by A(a,b,0) and A(a,b,z) correspondingly. In the Fresnel approximation, the Fourier transform of the diffracted wave function is related to the Fourier transform of the initial function via the frequency transfer characteristic of the free space H(a,b,z) , given by [1, 2]:

$$H = \frac{A(a,b,z)}{A(a,b,0)} = \exp\left\{-ikz(1-a^2-b^2)^{½}\right\} \qquad (5.1)$$

where:

$$A(a,b,0) = \int\int_{-\infty}^{\infty} U(x,y,0)\exp\left\{-ik(ax+by)\right\}dxd \qquad (5.2)$$

$$(1)\;\; A(a,b,z) = \int\int_{-\infty}^{\infty} U(x,y,z)\exp\left\{-ik(ax+by)\right\}dxdy \qquad (5.3)$$

Expressions provide a symmetrical relation between the initial and diffracted wave functions in the Fresnel approximation. Applied in the order (5.2) → (5.1) → (5.3) they result in the diffracted wave function, while being applied in the reversed order they allow for reconstruction of the initial wave function from the result of diffraction. We shall denote the forward and the reversed propagation operations defined by expressions (5.1), (5.2), (5.3) with operators L+ and L- respectively.
The described algorithm can be implemented numerically using Fast Fourier Transform (FFT) [2, 3] on a finite rectangular grid with periodic border conditions. It results in a model of beam propagation inside a rectangular wave guide with reflective walls. To approximate a free-space propagation, wide empty guard bands have to be formed around the wave function defined on a grid. To eliminate the influence of the finite rectangular data window, Gaussian amplitude windowing in the frequency domain should be applied-see [2, 3] for extensive analysis of these computational aspects.
The simplest and fastest LightPipes command for propagation is LPForvard. (The 'v' is a type error made on purpose!) It implements the spectral method described by (5.1), (5.2), (5.3). The syntax is simple, for example if you want to filter your field through a 1cm aperture and then propagate the beam 1m forward, you type the commands listed below:

```
%man0004.m
%
%LightPipes for Matlab
%Propagation of a field after filtering through a 1cm aperture.
m=1;
mm=0.001*m;
nm=1e-9*m;
size=20*mm;
lambda=1000*nm;
N=256;
R=5*mm;
z=0.8*m;

Field=LPBegin(size,lambda,N);
Field=LPCircAperture(R,0,0,Field);
Field=LPForvard(z,Field);
Intensity=LPIntensity(1,Field);

i=[1:N];
```

```
x(i)=-size/2 + i*size/N;
figure(6);
subplot(1,2,1);
imshow(Intensity);
subplot(1,2,2);
plot(x(i)/mm,Intensity(i,N/2));
xlabel('x [mm]');
ylabel('Intensity');
axis square;
flnm=sprintf('(man0004.emf)');flnm=strcat('\it',flnm);
text(5,-0.3,flnm);
print -dmeta '..\figures\man0004';
```

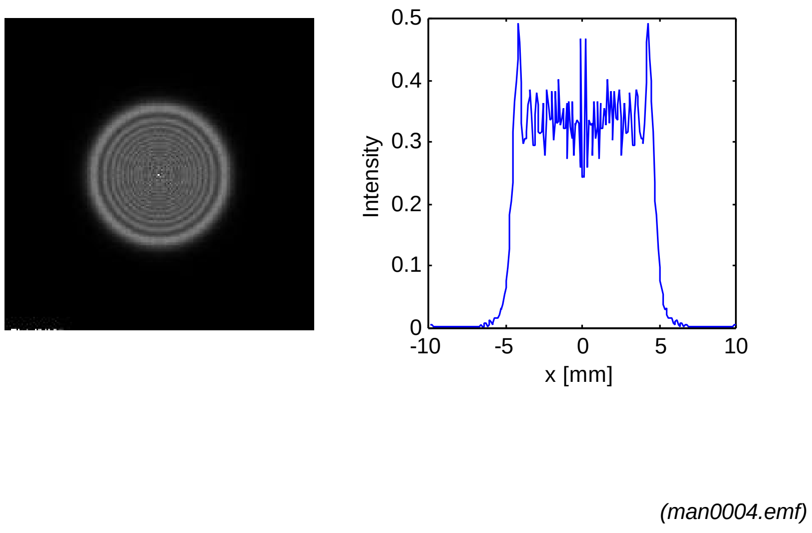


Figure 6 The result of the propagation: density and cross section intensity plots.

We see the diffraction effects, the intensity distribution is not uniform anymore. The algorithm is very fast in comparison with direct calculation of diffraction integrals. Features to be taken into account:

- The algorithm realises a model of light beam propagation inside a square wave guide with reflecting walls positioned at the grid edges. To approximate a free space propagation, the intensity near the walls must be negligible small. Thus the grid edges must be far enough from the propagating beam. Neglecting these conditions will cause interference of the propagating beam with waves reflected from the wave guide walls.

- As a consequence of the previous feature, we must be extremely careful propagating the plane wave to a distance comparable with D^2/λ where D is the diameter (or a characteristic size) of the beam, and λ is the wavelength. To propagate the beam to the far field (or just far enough) we have to choose the size of our grid much larger than the beam itself, in other words we define the field in a grid filled mainly with zeros. The grid must be even larger when the beam is aberrated the divergent beams sooner reach the region border.

Due to these two reasons the commands:

```
Field=LPBegin(0.02,10-6,256)
Field=LPRectAperture(0.02,0.02,0,0,0,Field)
Field=LPForvard(1,Field)
I=LPIntensity(2,Field)
```

make no sense (zero intensity). The cross section of the beam (argument of LPRectAperture) equals to the section of the grid (the first argument of LPBegin), so we have a model of light propagation in a wave guide but not in a free space. One has to put:

```
Field=LPBegin(0.04,10-6,256)
Field=LPRectAperture(0.02,0.02,0,0,0,Field)
Field=LPForvard(1,Field)
I=LPIntensity(2,Field)
```

for propagation in the near field, and may be:

```
Field=LPBegin(0.2,10-6,512)
Field=LPRectAperture(0.02,0.02,0,0,0,Field)
Field=LPForvard(400,Field)
I=LPIntensity(2,Field)
```

for far field propagation.

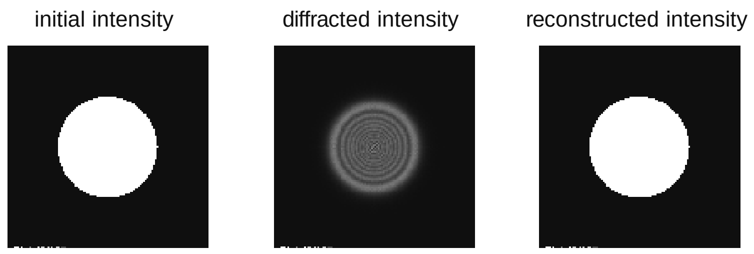
If we compare the result of the previous example with the result of:

```
Field=LPBegin(0.06,10-6,512)
Field=LPRectAperture(0.02,0.02,0,0,0,Field)
Field=LPForvard(400,Field)
I=LPIntensity(2,Field)
```

we'll see the difference.
We have discussed briefly the drawbacks of the FFT algorithm. The good thing is that it is very fast, works pretty well if properly used, is simple in implementation and does not require the allocation of extra memory. In LightPipes.1.1 and later a negative argument may be supplied to LPForvard. It means that the program will perform "propagation back" or in other words it will reconstruct the initial field from the one diffracted. For example:

```
%man0005.m
%
```

```
%LightPipes for Matlab
%Propagation and back propagation of a field
m=1;
mm=0.001*m;
nm=1e-9*m;

size=100*mm;
lambda=1000*nm;
N=256;
R=25*mm;
z=30*m;

Field=LPBegin(size,lambda,N);
Field=LPCircAperture(R,0,0,Field);
Iinitial=LPIntensity(1,Field);
Field=LPForvard(z,Field);
Idiffracted=LPIntensity(1,Field);
Field=LPForvard(-z,Field);
Ireconstructed=LPIntensity(1,Field);

figure(7);
subplot(1,3,1);
imshow(Iinitial);
title('initial intensity');
subplot(1,3,2);
imshow(Idiffracted);
title('diffracted intensity');
subplot(1,3,3);
imshow(Ireconstructed);
title('reconstructed intensity');
flnm=sprintf('(man0005.emf)');flnm=strcat('\it',flnm);
text(150,550,flnm);
print -dmeta '..\figures\man0005';
```



(man0005.emf)

Figure 7 The initial filed, the field propagated to the near field and the field propagated back.

5.2.2 Direct integration as a convolution: FFT approach

Another possibility of a fast computer implementation of the operator L+ is free from many of the drawbacks of the described spectral algorithm. The operator L+ may be numerically implemented with direct summation of the Fresnel-Kirchoff diffraction integral:

$$U(x_1, y_1, z) = \frac{k}{2\pi iz}\iint U(x, y, 0)\exp\left\{ik\frac{(x-x_1)^2+(y-y_1)^2}{2z}\right\}dxdy \qquad (5.4)$$

with functions U(x,y,0) and U(x,y,z) defined on rectangular grids. This integral may be converted into a convolution form which can be efficiently computed using FFT [4, 5]. This method is free from many drawbacks of the spectral method given by the sequence (5.2)$\rightarrow$(5.1)$\rightarrow$(5.3) although it is still very fast due to its use of FFT for computing of the integral sums.
Filter LPFresnel, defined starting from version LightPipes.1.1, implements this algorithm using the trapezoidal rule. It is almost as fast as LPForvard (from 2 to 5 times slower), it uses 8 times more memory than LPForvard and it allows for “more honest” calculation of near and far-field diffraction. As it does not require any protection bands at the edges of the region, the model may be built in a smaller grid, therefore the resources consumed and time of execution are comparable or even better than that of LPForvard. LPFresnel does not accept a negative propagation distance. When possible LPFresnel has to be used as the main computational engine within LightPipes for Matlab.

5.2.3 Direct integration

Direct calculation of the Fresnel-Kirchoff integrals is very inefficient in two-dimensional grids. The number of operations is proportional to N^4 , where N is the grid sampling. With direct integration we do not have any reflection at the grid boundary, so the size of the grid can just match the cross section of field distribution. LightPipes include a program LPForward realising direct integration. LPForward has the following features:

- arbitrary sampling and size of square grid at the input plane

- arbitrary sampling and size of square grid at the output plane, it means we can propagate field from a grid containing for example 52x52 points corresponding to 4.9x4.9cm to a grid containing 42x42 points and corresponding let’s say 8.75x8.75 cm.

5.2.4 Finite difference method.

It can be shown that the propagation of the field U in a medium with complex refractive coefficient A, is described by the differential equation:

$$\frac{\P^2 U}{\P x^2} + \frac{\P^2 U}{\P y^2} + 2ik\frac{dU}{d\,z} + A(x,y,z)U = 0 \tag{5.5}$$

To solve this equation, we re-write it as a system of finite difference equations:

$$\frac{U_{i+1,j}^{k+1} - 2U_{i,j}^{k+1} + U_{i-1,j}^{k+1}}{\Delta x^2} + \frac{U_{i,j+1}^{k} - 2U_{i,j}^{k} + U_{i,j-1}^{k}}{\Delta y^2} + 2ik\frac{U_{i,j}^{k+1} - 2U_{i,j}^{k}}{\Delta z} + A_{i,j}^{k+1}U_{i,j}^{k+1} = 0 \tag{5.6}$$

Collecting terms we obtain the standard three-diagonal system of linear equations, solution of which describes the complex amplitude of the light field in the layer *z+Dz* as a function of the field defined in the layer *z*:

$$-a_i U_{i-1,j}^{k+1} + c_i U_{i,j}^{k+1} - b_i U_{i+1.j}^{k+1} = f_i \tag{5.7}$$

where (we put $\Delta x = \Delta y = \Delta$)

$$a_i = b_i = -\frac{1}{\Delta^2} \tag{5.8}$$

$$c_i = A_{i,j}^{k+1} - \frac{2}{\Delta^2} + \frac{2ik}{\Delta z} \tag{5.9}$$

$$f_i = \frac{2ik}{\Delta z}U_{i,j}^{k} - \frac{U_{i,j+1}^{k} - 2U_{i,j}^{k} + U_{i,j-1}^{k}}{\Delta^2} \tag{5.10}$$

The three-diagonal system of linear equations (5.7) is solved by the standard elimination (double sweep) method, described for example in [6]. This scheme is absolutely stable (this variant is explicit with respect to the index *i* and implicit with respect to the index *j*). One step of propagation is divided into two sub-steps: the first sub-step applies the described procedure to all rows of the matrix, the second sub-step changes the direction of elimination and the procedure is applied to all columns of the matrix.
The main advantage of this approach is the possibility to take into account uniformly diffraction, absorption (amplification) and refraction. For example, the model of a waveguide with complex

three-dimensional distribution of refraction index and absorption coefficient (both are defined as real and imaginary components of the (three-dimensional in general) matrix $A^k_{i,j}$) can be built easily.
It works also much faster than all described previously algorithms on one step of propagation, though to obtain a good result at a considerable distance, many steps should be done. As the scheme is absolutely stable (at least for free-space propagation), there is no stability limitation on the step size in the direction *Z*. Large steps cause high-frequency errors, therefore the number of steps should be determined by trial (increase the number of steps in a probe model till the result stabilizes), especially for strong variations of refraction and absorption inside the propagation path.
Zero amplitude boundary conditions are commonly used for the described system. This, again, creates the problem of the wave reflection at the grid boundary. The influence of these reflections in many cases can be reduced by introducing an additional absorbing layer in the proximity of the boundary, with the absorption smoothly (to reduce the reflection at the absorption gradient) increasing towards the boundary.
In LightPipes version 1.2 the refraction term is not included into the propagation formulas, instead the phase of the field is modified at each step according to the distribution of the refractive coefficient. This “zero-order” approximation happened to be much more stable numerically than the direct inclusion of refraction terms into propagation formulas. It does not take into account the change of the wavelength in the medium, it does not model backscattering and reflections back on interfaces between different media. Perhaps there are other details to be mentioned. The described algorithm is implemented in a filter LPSteps.

```
% man0006.m
%
% LightPipes for Matlab
% calculation of the intensity near the
% focus of a lens with LPSteps.

clear;

m=1;
nm=1e-9*m;
mm=1e-3*m;
cm=1e-2*m;

lambda=632.8*nm;
size=4*mm;
N=100;
R=1.5*mm;
dz=10*mm;
f=50*cm;
n=(1+0.1*i)*ones(N,N);

F=LPBegin(size,lambda,N);
F=LPCircAperture(R,0,0,F);
F=LPLens(f,0,0,F);
```

```
for i=1:100
   F=LPSteps(dz,1,n,F);
   I=LPIntensity(0,F);
   for k=1:N
      Icross(i,k)=I(N/2,k);
   end
end
figure;
colormap('gray'); surfl(Icross); shading interp; axis off;
flnm=sprintf('(man0006.emf)');flnm=strcat('\it',flnm);
text(40,-50,flnm);
print -dmeta '..\figures\man0006';
```

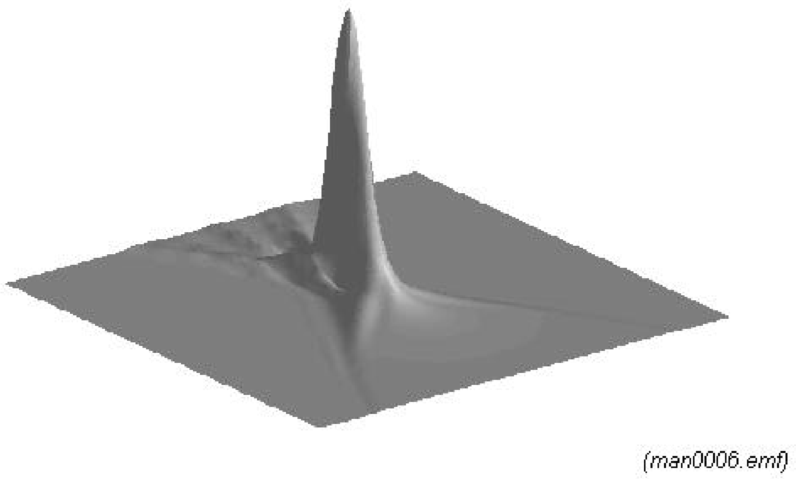


Figure 8. The use of LPSteps to calculate the intensity distribution in the focus of a lens.

LPSteps has a built-in absorption layer along the grid boundaries (to prevent reflections), occupying 10% of grid from each side. LPSteps is the only filter in LightPipes for Matlab allowing for modeling of (three-dimensional) waveguide devices.
Like LPForvard, LPSteps can inversely propagate the field, for example the sequence …..LPSteps(0.1, 1, n, Field) LPSteps(-0.1, 1, n, Field)…… doesn't change anything in the field distribution. The author has tested this reversibility also for propagation in absorptive/refractive media, examples will follow.
LPSteps implements scalar approximation, it is not applicable for modeling of waveguide devices in the vector approximation, where two components of the field should be taken into account.

5.2.5 Splitting and mixing beams

There are two commands in LightPipes which are useful for modelling of interferometers. With LPIntAttenuator we can split the field structure (amplitude division) - The two obtained fields could be processed separately and then mixed again with the routine LPBeamMix. In this script we have formed two beams each containing one “shifted” hole. After mixing these two beams we have a screen with two holes: a Young’s interferometer.

```
% man0007.m
%
% LightPipes for Matlab
% Two Holes Interferometer.
clear;

m=1;
nm=1e-9*m;
mm=1e-3*m;
cm=1e-2*m;

lambda=550*nm;
size=5*mm;
N=150;
R=0.12*mm;
d=0.5*mm;
z=5*cm;

F=LPBegin(size,lambda,N);
F1=LPCircAperture(R,d,0,F);
F2=LPCircAperture(R,-d,0,F);
F=LPBeamMix(F1,F2);
clear F1;
clear F2;
figure(1);
for i=1:10
   F=LPForvard(z,F);
   I=LPIntensity(1,F);
   subplot(2,5,i);
   subimage(I);
   Str=sprintf('z=%4.1f cm',i*z/cm)
   title(Str);
   axis off;
end
clear F;
flnm=sprintf('(man0007.emf)');flnm=strcat('\it',flnm);
text(0,300,flnm);
print -dmeta '..\figures\man0007'
```

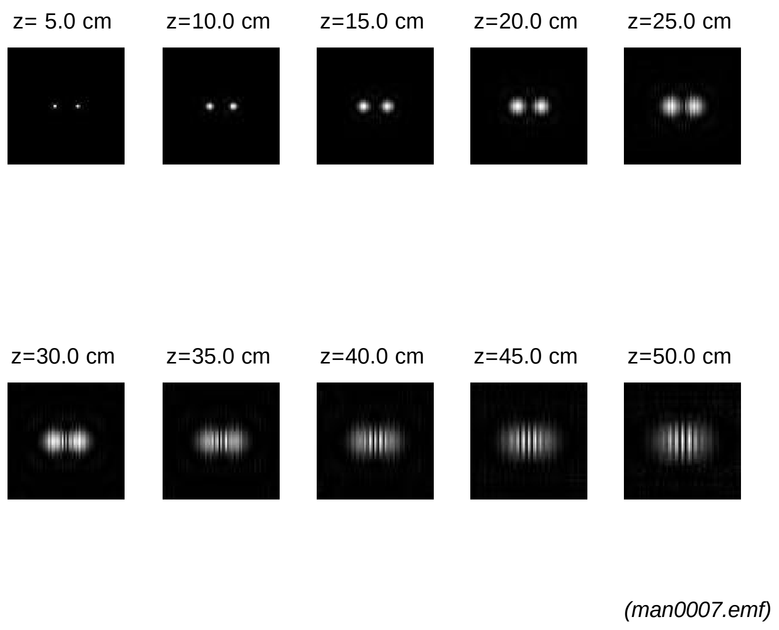


Figure 9 Young's interferometer.

Having the model of the interferometer we can "play" with it, moving the pinholes and changing their sizes. The following models the result of the interference of a plane wave diffracted at three round apertures:

```
% man0008.m
%
% LightPipes for Matlab
% Interference from three holes
clear;

m=1;
nm=1e-9*m;
mm=1e-3*m;
cm=1e-2*m;

lambda=550*nm;
size=5*mm;
N=150;
R=0.12*mm;
x=0.5*mm;
y=0.25*mm;
z=5*cm;

F=LPBegin(size,lambda,N);
F1=LPCircAperture(R,-x,-y,F);
F2=LPCircAperture(R,x,-y,F);
```

```
F3=LPCircAperture(R,0,y,F);
F=LPBeamMix(LPBeamMix(F1,F2),F3);
clear F1;
clear F2;
clear F3;
figure(1);
for i=1:10
   F=LPForvard(z,F);
   I=LPIntensity(1,F);
   subplot(2,5,i);
   subimage(I);
   Str=sprintf('z=%4.1f cm',i*z/cm)
   title(Str);
   axis off;
end
clear F;
flnm=sprintf('(man0008.emf)');flnm=strcat('\it',flnm);
text(0,300,flnm);
print -dmeta '..\figures\man0008';
```

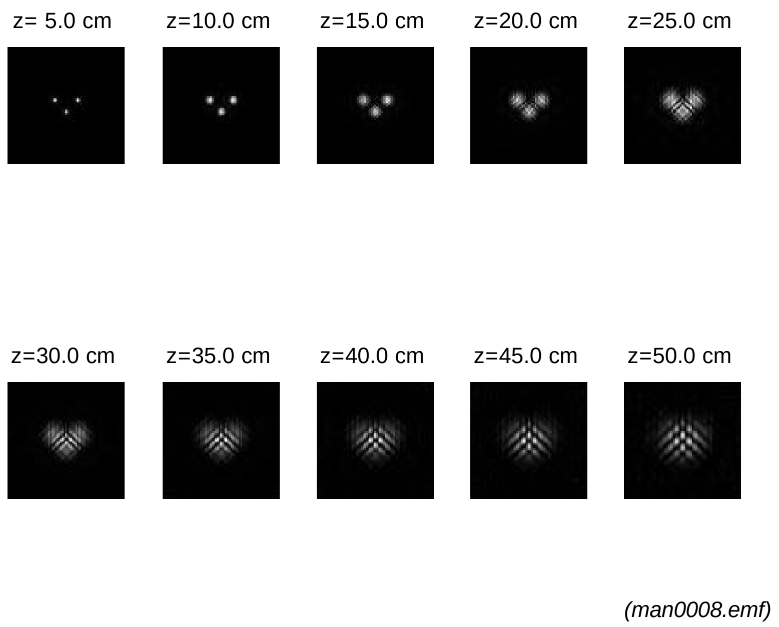
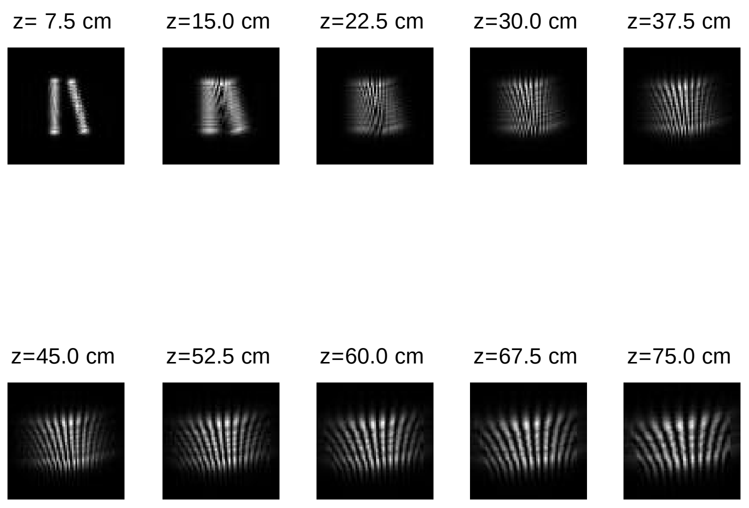


Figure 10 Intensity distributions behind the screen with three holes at several distances.

The next interferometer is more interesting:

```
% man0009.m
%
% LightPipes for Matlab
% Interference from two slits
```

```
clear;

m=1;
nm=1e-9*m;
mm=1e-3*m;
cm=1e-2*m;
rad=1;

lambda=550*nm;
size=5*mm;
N=150;
w=0.1*mm;
h=2.5*mm;
x=0.5*mm;
phi=15*rad;
dz=7.5*cm;
figure(1);

for i=1:10
   z=i*dz;
   F=LPBegin(size,lambda,N);
            F1=LPRectAperture(w,h,-x,0,0,F);
            F2=LPRectAperture(w,h,x,0,phi,F);
            F=LPBeamMix(F1,F2);
            clear F1;
            clear F2;
   F=LPFresnel(z,F);
   I=LPIntensity(1,F);
   subplot(2,5,i);
   subimage(I);
   Str=sprintf('z=%4.1f cm',z/cm)
   title(Str);
   axis off;
end
clear F;
flnm=sprintf('(man0009.emf)');flnm=strcat('\it',flnm);
text(0,300,flnm);
print -dmeta '..\figures\man0009';
```



(man0009.emf)

Figure 11 Intensity distributions behind a screen with two slits at several distances.

In the last example the intensity distribution is modulated by the wave, reflected from the grid edge, nevertheless it gives a good impression about the general character of the interference pattern. To obtain a better result, the calculations should be conducted in a larger grid or other numerical method should be used. The following example uses a direct integration algorithm (the input and output are in different scales and have different samplings):

```
% man0010.m
%
% LightPipes for Matlab
% Interference from two slits with direct integration

clear;

m=1;
nm=1e-9*m;
mm=1e-3*m;
cm=1e-2*m;
rad=1;

lambda=550*nm;
size=2.6*mm;
SizeNew=5*mm;
N=64;
w=0.1*mm;
h=2.5*mm;
x=0.5*mm;
```

```
phi=15*rad;
z=75*cm;
figure;

  F=LPBegin(size,lambda,N);
        F1=LPRectAperture(w,h,-x,0,0,F);
        F2=LPRectAperture(w,h,x,0,phi,F);
        F=LPBeamMix(F1,F2);
        clear F1;
        clear F2;
  F=LPForward(z,SizeNew,N,F);
  I=LPIntensity(1,F);
  subimage(I);
  Str=sprintf('z=%4.1f cm',z/cm)
  title(Str);
  axis off;
clear F;
flnm=sprintf('(man0010.emf)');flnm=strcat('\it',flnm);
text(N-5,N+5,flnm);

print -dmeta '..\figures\man0010';
```

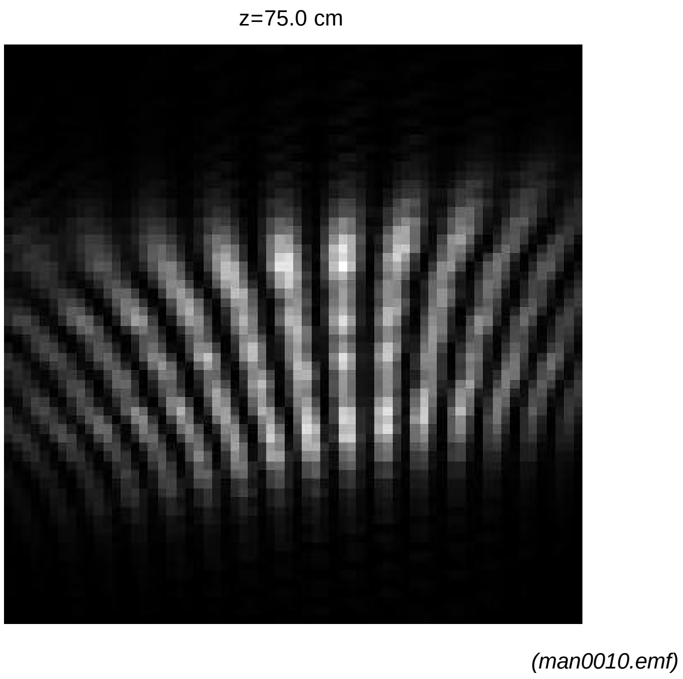


Figure 12 Intensity distributions in plane of the screen and 75 cm after the screen, note that input and output have different scales, input grid is 2.5x2.5mm, output is 5x5mm

This example uses approximately 25 times less memory than the previous FFT. The calculation may take from minutes to tens of minutes, depending on the speed of the computer.

5.2.6 Interpolation

The program LPInterpol is the tool for manipulating the size and the dimension of the grid and for changing the shift, rotation and the scale of the field distribution. It accepts six command line arguments, the first is the new size of the grid, second and other are optional arguments. The second argument gives the new number of points, the third gives the value of transverse shift in X direction, the fourth gives the shift in Y direction, the fifth gives the field rotation (first shift and then rotation). The last sixth argument determines the magnification, its action is equivalent to passing the beam through a focal system with magnification M (without diffraction, but preserving the integral intensity). For example if the field was propagated with FFT algorithm LPForvard, then the grid contains empty borders, which is not necessary if we want to propagate the field further with LPForward. Other way around, after LPForward we have to add some empty borders to continue with LPForvard. LPInterpol is useful for interpolating into a grid with different size and number of points. Of course it is not too wise to interpolate from a grid of 512x512 points into a grid of 8x8, and then back because all information about the field will be lost. The same is true for interpolating the grid of 1x1m to 1x1mm and back. When interpolating into a grid with larger size, for example from 1x1 to 2x2, the program puts zeros into the new added regions. Figure 13 illustrates the usage of LPInterpol for the transition from a fine grid used by LPForvard (near field) to a coarse grid used by LPForward (far field).

```
% man0011.m
%
% LightPipes for Matlab
% Calculation of the far field with interpolation and
% direct integration.
clear;

m=1;
nm=1e-9*m;
mm=1e-3*m;
cm=1e-2*m;

lambda=1000*nm;
size=10*mm;
NewSize=7.5*mm;
NewerSize=400*mm;
N=256;
NewN=32;

w=5*mm;
h=5*mm;
z=20*cm;
zfar=400*m;

F=LPBegin(size,lambda,N);
F=LPRectAperture(w,h,0,0,0,F);
F=LPForvard(z,F);
```

```
I0=LPIntensity(1,F);
F=LPInterpol(NewSize,NewN,0,0,0,1,F);
F=LPForward(zfar,NewerSize,NewN,F);
I1=LPIntensity(1,F);

figure;
subplot(1,2,1);
imshow(I0);
axis off;
subplot(1,2,2);
mesh(I1);
axis off;
clear F;
flnm=sprintf('(man0011.bmp)');flnm=strcat('\it',flnm);
text(10,-10,flnm);

print -dbitmap '..\figures\man0011';
```

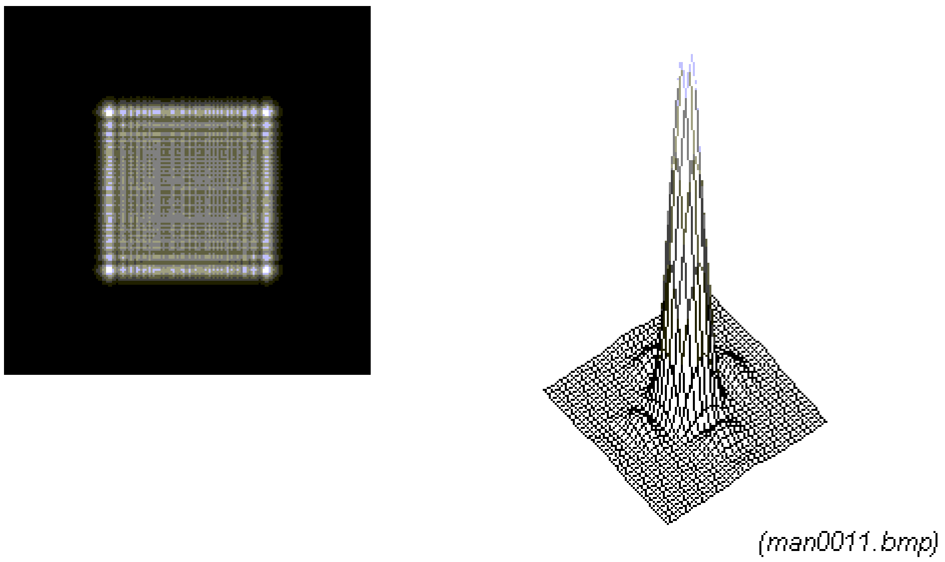


Figure 13 Illustration of the usage of LPInterpol for the transition from a fine grid used by LPForvard (near field) to a coarse grid used by LPForward (far field).

5.2.7 Phase and intensity filters

There are four kinds of phase filters available in LightPipes -wave front tilt, the quadratic phase corrector called lens, a general aberration in the form of a Zernike polynomial, and a user defined filter. To illustrate the usage of these filters let’s consider the following examples:

```
% man0012.m
%
% LightPipes for Matlab
% propagation through a lens.
clear;

m=1;
nm=1e-9*m;
mm=1e-3*m;
cm=1e-2*m;

lambda=1000*nm;
size=40*mm;
N=256;

w=20*mm;
f=8*m;
z=4*m;

F=LPBegin(size,lambda,N);
F=LPRectAperture(w,w,0,0,0,F);
F=LPLens(f,0,0,F);
I0=LPIntensity(1,F);
Phi0=LPPhase(F);
F=LPForvard(z,F);
I1=LPIntensity(1,F);

figure;
subplot(2,3,1);
i=[1:N];
x(i)=-size/2+i*size/N;
plot(x/mm,Phi0(N/2,i));xlabel('x [mm]');ylabel('phase [rad]');
title('Phase distribution');

axis square;
subplot(2,3,4);
imshow(I0);
str=sprintf('Intensity in\nthe plane of the lens.');
title(str);

subplot(2,3,6);
```

```
imshow(I1);
str=sprintf('Intensity at half\nthe distance to the focus.');
title(str);
flnm=sprintf('(man0012.bmp)');flnm=strcat('\it',flnm);
text(150,350,flnm);

print -dbitmap '..\figures\man0012';
```

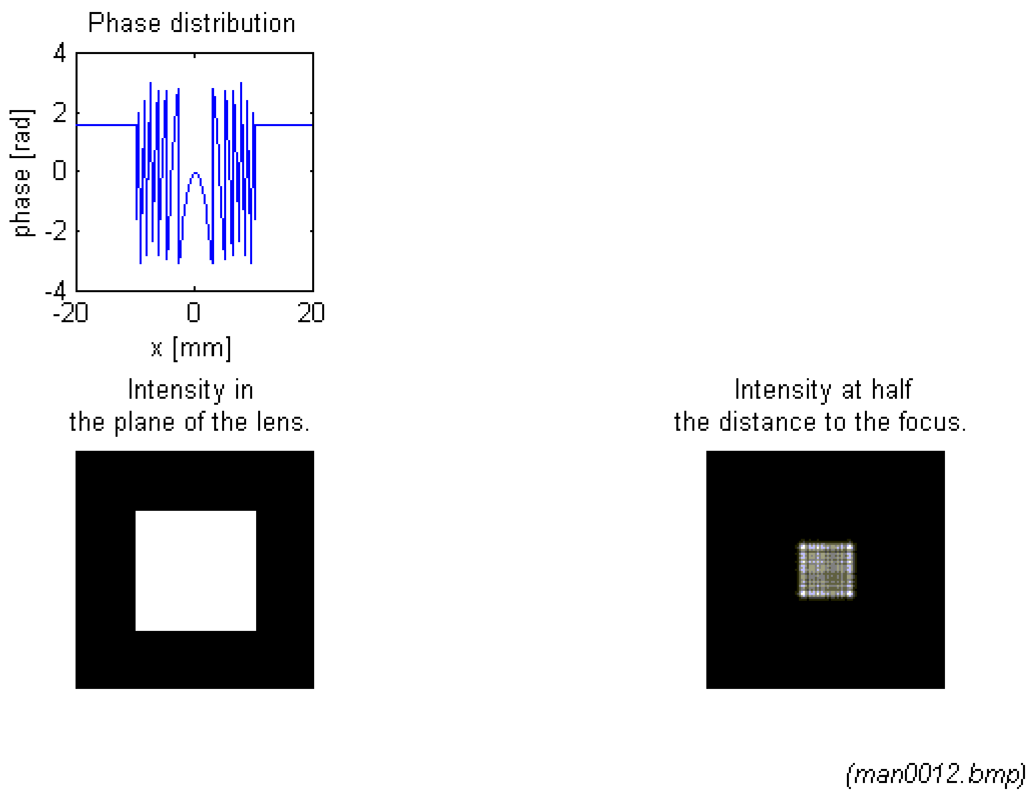


Figure 14 The phase distribution after passing the lens, intensity in the plane of the lens and at a distance equal to the half of the focal distance. (from left to right)

The first sequence of operators forms the initial structure, filters the field through the rectangular aperture and then filters the field through a positive lens with optical power of 0.125D (the focal distance of 1/0.125=8m). With the second command we propagate the field 4m forward. As 4m is exactly the half of the focal distance, the cross section of the beam must be reduces twice.
We have to be very careful propagating the field to the distance which is close to the focal distance of a positive lens- the near—focal intensity and phase distributions are localised in the central region of the grid occupying only a few grid points. This leads to the major loss of information about the field distribution. The problem is solved by applying the co-ordinate system that is tied to the divergent or convergent light beam, the tools to do this will be described later.
The lens may be decentered, LPLens(8, 0.01, 0.01, Field) produces the lens with a focal length of 1/0.125 shifted by 0.01 in X and Y directions. Note, when the lens is shifted, the aperture of the

lens is not shifted, the light beam is not shifted also, only the phase mask corresponding to the lens is shifted.
The wave front tilt is illustrated by following example:

```
% man0013.m
%
% LightPipes for Matlab
% Demonstration of tilt.
clear;

m=1;
nm=1e-9*m;
mm=1e-3*m;
cm=1e-2*m;
rad=1;
mrad=1e-3*rad;

lambda=1000*nm;
size=40*mm;
N=256;

w=20*mm;
z=4*m;
tx=0.1*mrad;
ty=-0.1*mrad;

F=LPBegin(size,lambda,N);
F=LPRectAperture(w,w,0,0,0,F);
F=LPTilt(tx,ty,F);
F=LPForvard(z,F);
I=LPIntensity(0,F);
Phi=LPPhase(F);
i=[1:N];
x(i)=-size/2+i*size/N;

figure;
subplot(1,2,1);
plot(x/mm,I(N/2,i));
xlabel('x [mm]');
ylabel('Intensity');
axis square;

subplot(1,2,2);
plot(x/mm,Phi(N/2,i));
xlabel('x [mm]');
ylabel('phase [rad]');
```

```
axis square;
flnm=sprintf('(man0013.bmp)');flnm=strcat('\it',flnm);
text(5,-8,flnm);

print -dbitmap '..\figures\man0013';
```

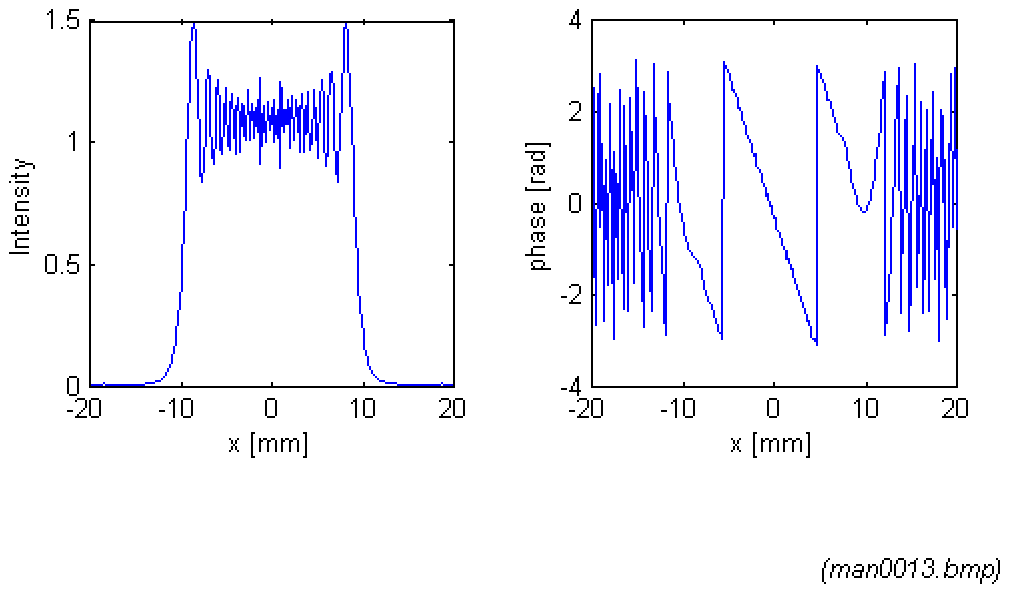


Figure 15 The intensity and the phase after tilting the wave front by 0.1 mrad and propagating it 8 m forward. Note the transversal shift of the intensity distribution and the phase tilt.

In this example the wave front was tilted by α=0.1 mrad in X and Y directions, then propagated it to the distance Z=8m, so in the output distribution we observe the transversal shift of the whole intensity distribution by αZ=0.8mm.

5.2.8 Zernike polynomials

Any aberration in a circle can be decomposed over a sum of Zernike polynomials. Formulas given in [7] have been directly implemented in LightPipes. The program is called LPZernike and accepts four command line arguments:

1. The radial order n (first column in Table 13.2 [8] p. 465).
2. The azimuthal order m, |m|≤n, polynomials with negative n are rotated 90° relative to the polynomials with positive n. For example LPZernike(5,3,1,1,Field) gives the same aberration as LPZernike(5,-3,1,1,Field), but the last is rotated 90°. This index corresponds to the n-2m given in the third column of Table 13.2 in [8], p. 465.

3. The radius, R
4. The amplitude of aberration in radians reached at R

We can uniformly introduce LPLens and LPTilt with LPZernike, the difference is that we pass the amplitude of the aberration to LPZernike. LPLens and LPTilt accept conventional meters and radians.

5.2.9 Spherical co-ordinates

The principle of beam propagation in the "floating" co-ordinate system (for the case of a lens wave guide) is shown in Figure 16.

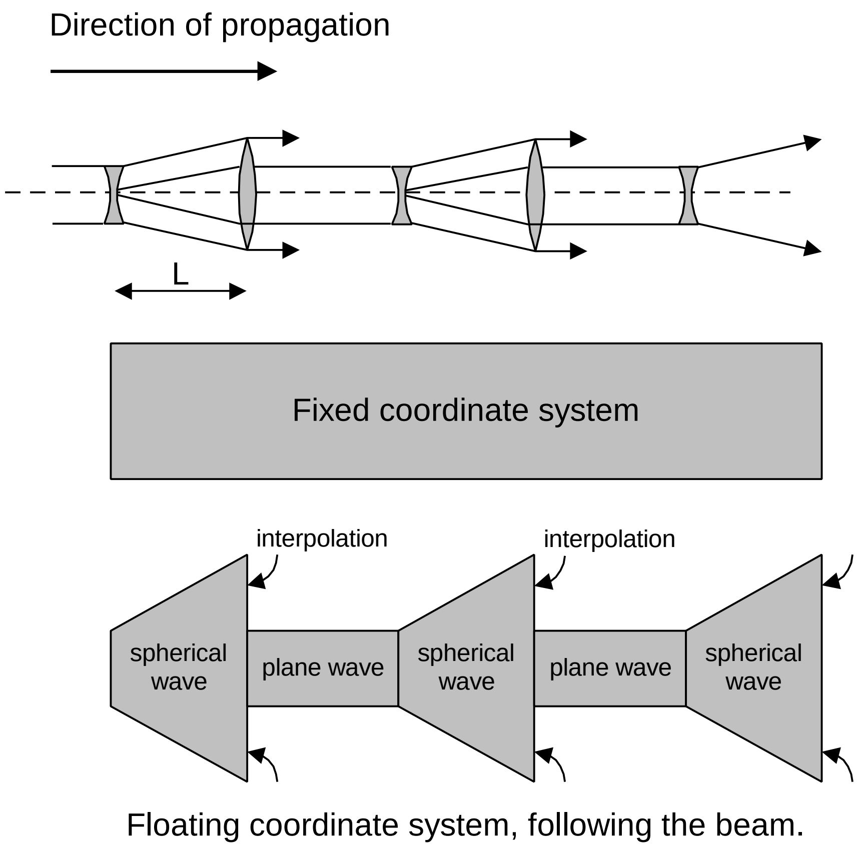


Figure 16. Illustration for the light propagation in lens wave guides with fixed and floating co-ordinate systems.

The spherical co-ordinates follow the geometrical section of the divergent or convergent light beam. Propagation in spherical co-ordinates is implemented with programs LPLensForvard and LPLensFresnel. Both filters accept two parameters: the focal distance of the lens, and the distance of propagation. When LPLensForvard or LPLensFresnel is called, it "bends" the co-ordinate system so, that it follows the divergent or convergent spherical wave front, and then propagates the field to the distance Z in the transformed co-ordinates. Filter LPConvert should be used to convert the field back to the rectangular co-ordinate system. Some LightPipes filters can not be applied to the field in spherical co-ordinates. As the co-ordinates follow the geometrical section of the light beam, operator LPLensForvard(10,10,Field) will produce floating exception because the calculations can not be conducted in a grid with zero size (that is so in the geometrical approximation of a focal point). On the other hand, diffraction to the focus is equivalent to the diffraction to the far field (infinity), thus the FFT convolution algorithm will not work properly anyway. To model the diffraction into the focal point, a more complicated trick should be used:

```
% man0014.m
%
% LightPipes for Matlab
% Use of spherical coordinates.
clear;

m=1;
nm=1e-9*m;
mm=1e-3*m;
cm=1e-2*m;

lambda=1000*nm;
size=10*mm;
N=256;

w=5*mm;
f1=10*m;
f2=1.111111*m;
z=1*m;

F=LPBegin(size,lambda,N);
F=LPRectAperture(w,w,0,0,0,F);
F=LPLens(f1,0,0,F);
F=LPLensFresnel(f2,z,F);
F=LPConvert(F);
I=LPIntensity(0,F);
Phi=LPPhase(F);
i=[1:N];
x(i)=-size/2+i*size/N;

figure;
subplot(1,2,1);
plot(x/mm,Phi(N/2,i));
xlabel('x [mm]');
ylabel('phase [rad]');
axis square;

subplot(1,2,2);
plot(x/mm,I(N/2,i));
xlabel('x [mm]');
ylabel('Intensity');
```

```
axis square;
flnm=sprintf('(man0014.bmp)');flnm=strcat('\it',flnm);
text(2,-450,flnm);

print -dbitmap '..\figures\man0014';
```

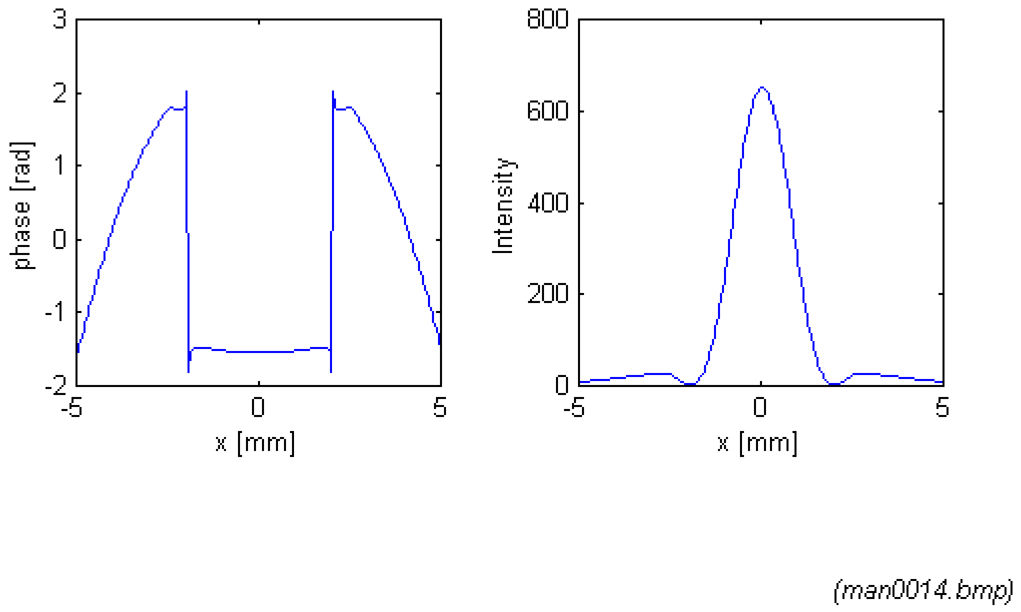


Figure 17. Diffraction to the focus of a lens using spherical co-ordinates.

In Figure 17 we calculate the diffraction to the focus of a lens with a focal distance of 1m. It is represented as a combination of a weak phase mask LPLens(f_1,F) and a “strong” geometrical co-ordinate transform LPLensFresnel(f_2,z,F). The grid after propagation is 10 times narrower than in the input plane. The focal intensity is 650 times higher than the input intensity and the wave front is plain as expected.

5.2.10 User defined phase and intensity filters

The phase and intensity of the light beam can be manipulated in several ways. The phase and intensity distributions may be produced within Matlab as shown in the next examples:

```
% man0015.m
%
% LightPipes for Matlab
% Substitution of intensity- and phase masks into the field.
clear;
```

```
m=1;
nm=1e-9*m;
mm=1e-3*m;
cm=1e-2*m;

lambda=1000*nm;
size=30*mm;
N=256;

R=10*mm;
for i=1:N
  for j=1:N
                Int(i,j)=abs(sin(i/10)*cos(j/5));
    Phase(i,j)=cos(i/10)*sin(j/5);
  end
end


F=LPBegin(size,lambda,N);
F=LPSubIntensity(Int,F);
F=LPSubPhase(Phase,F);
F=LPCircAperture(R,0,0,F);
I=LPIntensity(1,F);
Phi=LPPhase(F);

figure;
subplot(1,2,1);
imshow(I);

subplot(1,2,2);
imshow(Phi);
flnm=sprintf('(man0015.emf)');flnm=strcat('\it',flnm);
text(100,400,flnm);

print -dbitmap '..\figures\man0015';
```

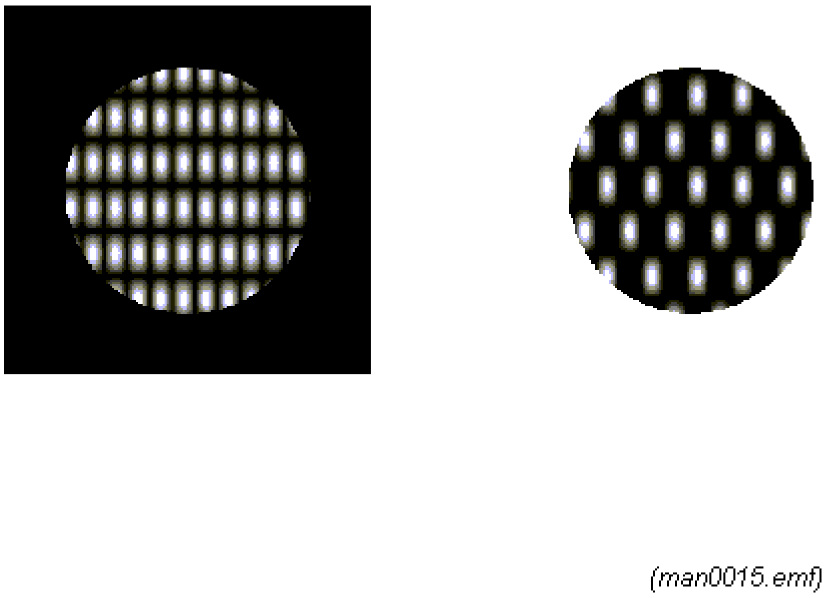


Figure 18 An arbitrary intensity- and phase distribution.

You can also create your own mask with a program like Microsoft Windows95 Paint. In the next example we made an arrow using Paint and stored it as a 200 pixels width x 200 pixels height, black-and-white, monochrome bitmap file (for example: arrow.bmp). This file can be read into Matlab using Matlab’s “double(imread(‘arrow’,’bmp’))” command. Next the arrow can be used as a filter:

```
% man0016.m
%
% LightPipes for Matlab
% Importing a mask from disk.
clear;

m=1;
nm=1e-9*m;
mm=1e-3*m;
cm=1e-2*m;

lambda=1000*nm;
size=25*mm;

R=6*mm;
Xshift=2*mm;
```

```
Yshift=0*mm;

arrow=double(imread('arrow','bmp'));
[N,N]=size(arrow);
Field=LPBegin(size,lambda,N);
Field=LPGaussAperture(R,Xshift,Yshift,1,Field);
Field=LPMultIntensity(arrow,Field);
I=LPIntensity(1,Field);
figure;
subplot(1,2,1),imshow(arrow);
subplot(1,2,2),imshow(I);
str1=sprintf('Imported bitmap');
str2=sprintf('Intensity distribution\nafter the Gaussian aperture\nand the arrow-
mask');
subplot(1,2,1),title(str1);
subplot(1,2,2),title(str2);
flnm=sprintf('(man0016.emf)');flnm=strcat('\it',flnm);
text(200,400,flnm);

print -dmeta '..\figures\man0016';
```

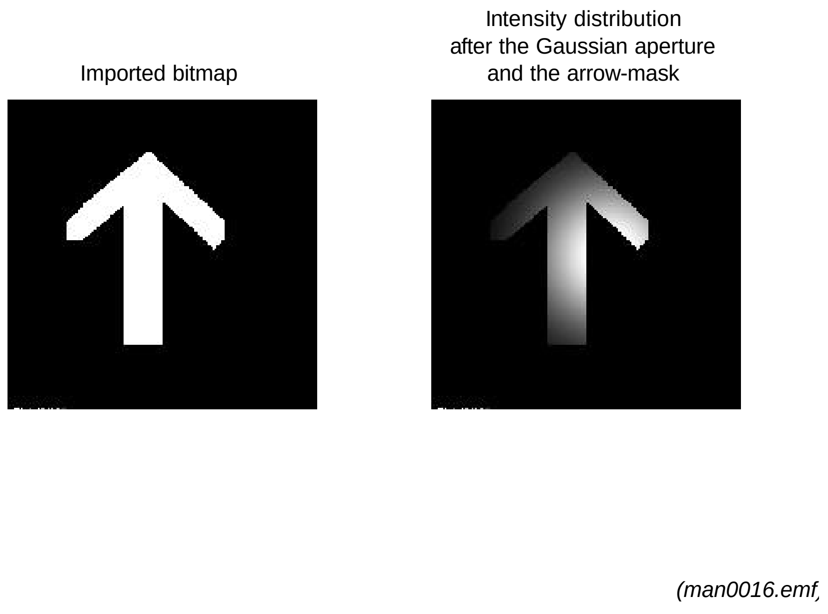


Figure 19 Illustration of importing a bit map from disk

5.2.11 Random filters

There are two random filters to introduce a random intensity or a random phase distribution in the field. The commands LPRandomIntensity and LPRandomPhase need a seed to initiate the random number generator. Use has been made of the standard C function rand. The LPRandomPhase command needs a maximum phase value (in radians). The LPRandomIntensity command yields a normalised random intensity distribution. The LPRandomIntensity and the LPRandomPhase commands leave the phase and the intensity unchanged respectively.

5.2.12 FFT and spatial filters

LightPipes for Matlab provides a possibility to perform arbitrary filtering in the Fourier space. There is an operator, performing the Fourier transform of the whole data structure: LPPipFFT.

```
% man0017.m
%
% LightPipes for Matlab
% The use of Fourier transform.
clear;

m=1;
nm=1e-9*m;
mm=1e-3*m;
cm=1e-2*m;

lambda=1000*nm;
size=15*mm;
N=150;

Rf=1.5e-3;
R=3*mm;
z=1*m;
seed =7;
MaxPhase=1.5;
Xshift=2*mm;
Yshift=0*mm;

Field=LPBegin(size,lambda,N);
Field=LPCircAperture(R,0,0,Field);
Field=LPRandomPhase(seed,MaxPhase,Field);
Field=LPFresnel(z,Field);
I0=LPIntensity(1,Field);
Field=LPPipFFT(1,Field);
Field=LPCircAperture(Rf,0,0,Field);
Field=LPPipFFT(-1,Field);
I1=LPIntensity(1,Field);
figure;
subplot(1,2,1),imshow(I0);
subplot(1,2,2),imshow(I1);
str1=sprintf('Unfiltered Intensity');
str2=sprintf('Filtered Intensity');
```

```
subplot(1,2,1),title(str1);
subplot(1,2,2),title(str2);
flnm=sprintf('(man0017.emf)');flnm=strcat('\it',flnm);
text(100,220,flnm);

print -dmeta '..\figures\man0017';
```

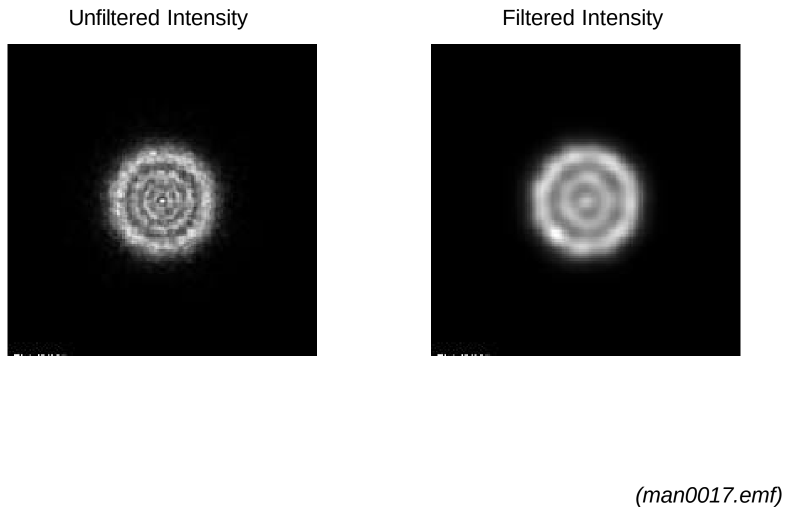


Figure 20. Intensity distributions before and after applying a spatial filter.

Note, we still can apply all the intensity and phase filters in the Fourier-space. The whole grid size in the angular frequency domain corresponds to 2πλ/Δx, where Δx is the grid step. One step in the frequency domain corresponds to 2πλ/x, where x is the total size of the grid. LightPipes for Matlab filters do not know about all these transformations, so the user should take care about setting the proper size (using the relations mentioned) of the filter (still in linear units) in the frequency domain.

5.2.13 Laser amplifier

The LPGain command introduces a simple single-layer model of a laser amplifier. The output field is given by:

$$F_{out}(x,y)=\sqrt{\exp\left[\frac{G_0\,L}{1+\dfrac{I(x,y)}{I_{sat}}}\right]}F_{in}(x,y) \tag{5.11}$$

where G_0 is the small signal gain, L is the length of the gain medium, $I(x,y)$ is the intensity and I_{sat} is the saturation intensity. The next example illustrates the usage of LPGain for the simulation of the amplification of a Gaussian laser beam. Note the effect of gain-saturation.

```
% man0018.m
%
% LightPipes for Matlab
% Saturable laser gain.
clear;

m=1;
nm=1e-9*m;
mm=1e-3*m;
cm=1e-2*m;

lambda=1000*nm;
size=30*mm;
N=50;

R=5*mm;
Gain0=3/m;
Isat=0.1;
Lgain=100*cm;
Field=LPBegin(size,lambda,N);
Field=LPGaussAperture(R,0,0,1,Field);
Iin=LPIntensity(0,Field);
Field=LPGain(Isat,Gain0,Lgain,Field);
Iout=LPIntensity(0,Field);

figure;
subplot(1,2,1),mesh(Iin);axis([0 N 0 N 0 1.5]);
subplot(1,2,2),mesh(Iout);axis([0 N 0 N 0 1.5]);
str1=sprintf('Input beam');
str2=sprintf('Amplified beam');
subplot(1,2,1),title(str1); axis square;
subplot(1,2,2),title(str2); axis square;
flnm=sprintf('(man0018.bmp)');flnm=strcat('\it',flnm);
text(20,-10,-1.,flnm);

print -dbitmap '..\figures\man0018';
```

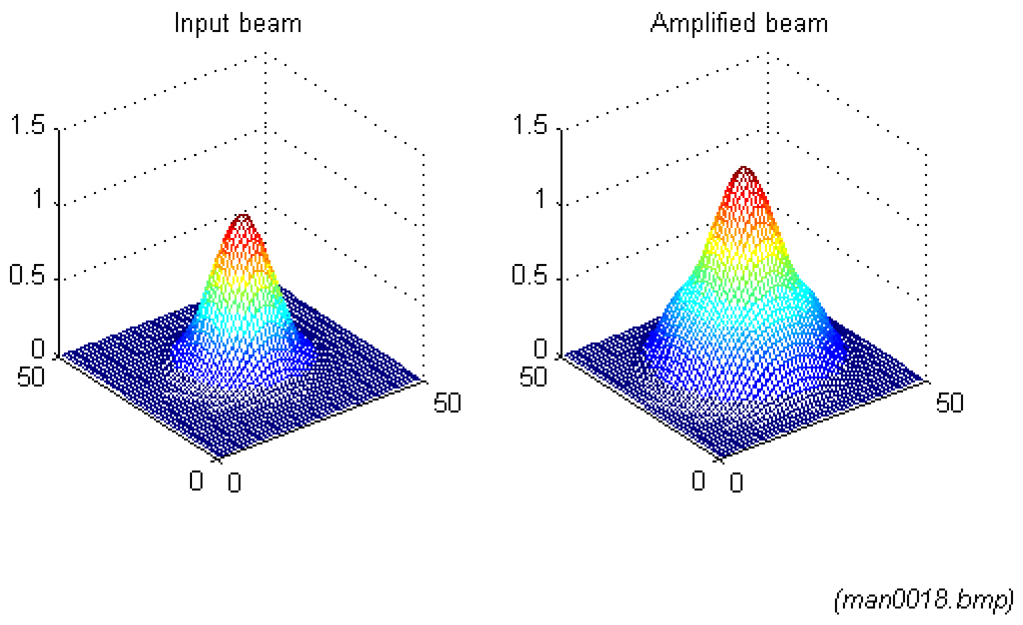


Figure 21. The effect of a saturable laser gain-section on the field.

5.2.14 Diagnostics: Strehl ratio, beam power

The LPStrehl command calculates the Strehl ratio of the field, defined as:

$$StrehlRatio = \frac{\left(\iint \mathrm{Re}(F_{in}(x,y)dxdy\right)^2 + \left(\iint \mathrm{Im}(F_{in}(x,y)dxdy\right)^2}{\left(\iint |F_{in}(x,y)| dxdy\right)^2} \qquad (5.12)$$

The next example calculates the Strehl ratio of a field with increasing random phase fluctuations:

```
% man0019.m
%
% LightPipes for Matlab
% Beam diagnostics
clear;

m=1;
nm=1e-9*m;
mm=1e-3*m;
cm=1e-2*m;

lambda=1000*nm;
```

```
size=30*mm;
N=200;

w=25*mm;

F=LPBegin(size,lambda,N);
F=LPRectAperture(w,w,0,0,0,F);
for i=1:21
        seed=rand(1);
        MaxPhase(i)=i-1;
  Field=LPRandomPhase(seed,MaxPhase(i),F);
  Strehl(i)=LPStrehl(Field);
  Phase=LPPhase(Field);
  for j=1:N
    Phi(i,j)=Phase(N/2,j);
  end
end
j=[1:N];
i=[1:21];
x(j)=(-size/2+j*size/N)/mm;
figure;
xstr='x [mm]';
ystr='phase';
subplot(2,3,1),plot(x(j),Phi(1,j)),axis([-20 20 -4
4]),title('MaxPhase=0'),xlabel(xstr),ylabel(ystr);axis square;
subplot(2,3,2),plot(x(j),Phi(3,j)),axis([-20 20 -4
4]),title('MaxPhase=2'),xlabel(xstr),ylabel(ystr);axis square;
subplot(2,3,3),plot(x(j),Phi(20,j)),axis([-20 20 -4
4]),title('MaxPhase=20'),xlabel(xstr),ylabel(ystr);axis square;
subplot(2,3,4),plot(MaxPhase(i),Strehl(i));
str2=sprintf('Strehl-Ratio of\na beam with random phase.');
subplot(2,3,4),title(str2),xlabel('MaxPhase'),ylabel('Strehl-ratio');axis square;
flnm=sprintf('(man0019.bmp)');flnm=strcat('\it',flnm);
text(60,-0.2,flnm);

print -dbitmap '..\figures\man0019';
```

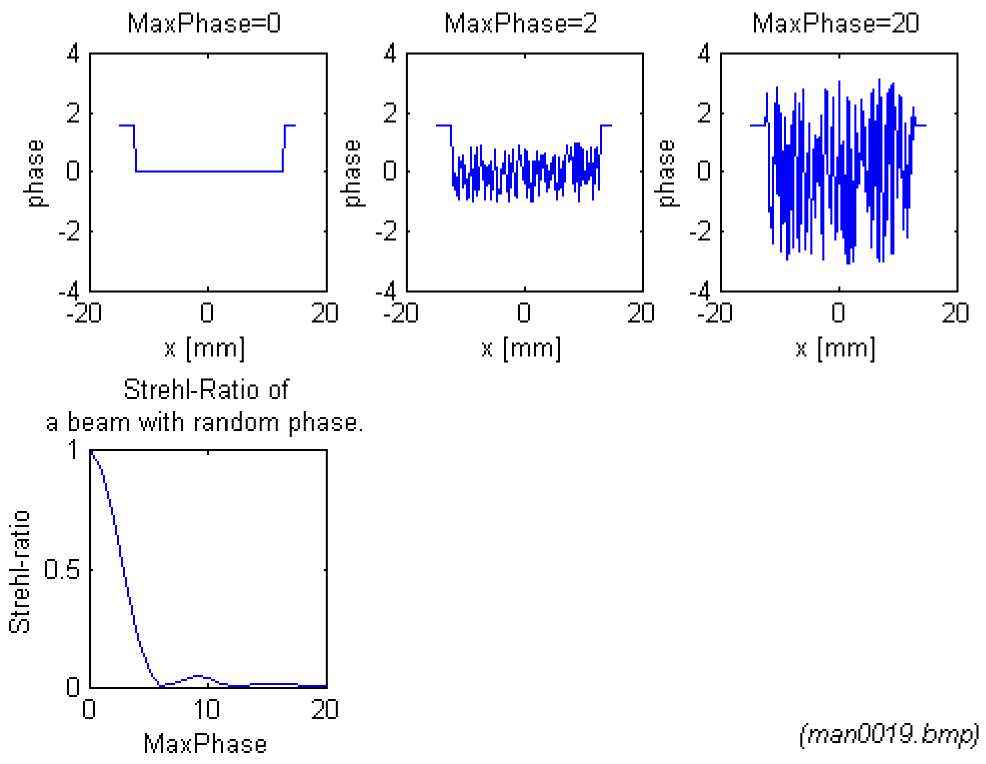


Figure 22. Demonstration of the use of the LPStrehl function to calculate the Strehl ratio (beam quality).

The LPNormal command normalises the field according to:

$$F_{out}(x,y) = \frac{F_{in}(x,y)}{\sqrt{P}} \tag{5.13}$$

$$P = \iint \left|F_{in}(x,y)\right|^2 dxdy \tag{5.14}$$

where P is the total beam power. Use the command: [Fout,NC]=LPNormal(Fin) where NC is the calculated normalization coefficient.

6 Example models

6.1 Shearing interferometer

We can easily model a shearing interferometer using the operators introduced in the previous sections. Suppose we have to model a device shown in .

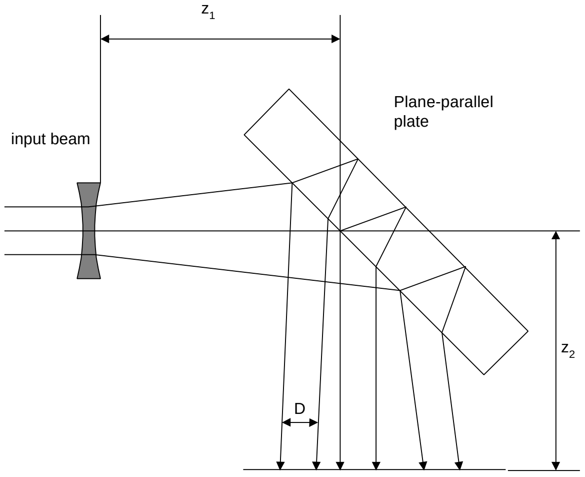


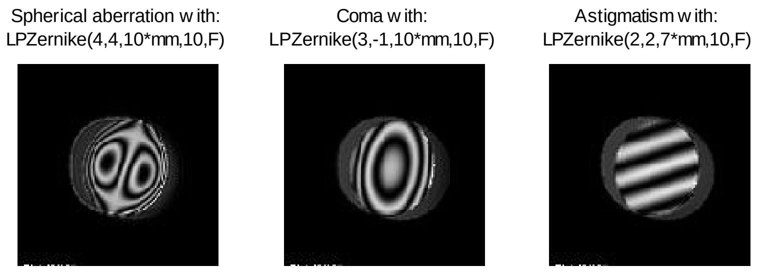
Figure 23 Shearing interferometer

Let z_1 = 0.5m, z_2 = 0.5m, λ=500 nm, the focal length of the lens is -20m, the beam is divided 1:1 and one of the two beams is shifted by D=3mm in the x-direction and D_1=1mm in the Y-direction after reflection on the back surface of the plane-parallel plate.We can put a source of arbitrary aberration on the place of the lens. Changing the amplitude and the order of aberration we can obtain all the shearing interferograms shown in the chapter about shearing interferometers of [7]. Figure 24 shows the use of the LPZernike command to obtain these aberrations.

```
% man0020.m
%
% LightPipes for Matlab
% Shearing interferometer with aberrated wavefront.
clear;

m=1;
cm=1e-2*m;
mm=1e-3*m;
nm=1e-9*m;

size=4*cm;
lambda=500*nm;
N=128;
R=1*cm;
f=-20*m;
z1=50*cm;
z2=50*cm;
```

```
D=3*mm;
D1=1*mm;
Rplate=0.5;
figure(1);

%1) Spherical aberration:
F=LPBegin(size,lambda,N);
F=LPCircAperture(R,0,0,F);
F=LPZernike(4,0,10*mm,10,F);
F=LPForvard(z1,F);
F1=LPIntAttenuator(Rplate,F);
F2=LPIntAttenuator(1-Rplate,F);
F2=LPInterpol(size,N,D,D1,0,1,F2);
F=LPBeamMix(F1,F2);
I=LPIntensity(1,F);
str1=sprintf('Spherical aberration with:\nLPZernike(4,4,10*mm,10,F)');
subplot(1,3,1),imshow(I),title(str1,'FontSize',8);

%2) Coma:
F=LPBegin(size,lambda,N);
F=LPCircAperture(R,0,0,F);
F=LPZernike(3,-1,10*mm,10,F);
F=LPForvard(z1,F);
F1=LPIntAttenuator(Rplate,F);
F2=LPIntAttenuator(1-Rplate,F);
F2=LPInterpol(size,N,D,D1,0,1,F2);
F=LPBeamMix(F1,F2);
I=LPIntensity(1,F);
str1=sprintf('Coma with:\nLPZernike(3,-1,10*mm,10,F)');
subplot(1,3,2),imshow(I),title(str1,'FontSize',8);

%3) Astigmatism:
F=LPBegin(size,lambda,N);
F=LPCircAperture(R,0,0,F);
F=LPZernike(2,2,7*mm,10,F);
F=LPForvard(z1,F);
F1=LPIntAttenuator(Rplate,F);
F2=LPIntAttenuator(1-Rplate,F);
F2=LPInterpol(size,N,D,D1,0,1,F2);
F=LPBeamMix(F1,F2);
I=LPIntensity(1,F);
str1=sprintf('Astigmatism with:\nLPZernike(2,2,7*mm,10,F)');
subplot(1,3,3),imshow(I),title(str1,'FontSize',8);
flnm=sprintf('(man0020.emf)');flnm=strcat('\it',flnm);
text(50,275,flnm);
```

```
print -dmeta '..\figures\man0020';
```



(man0020.emf)

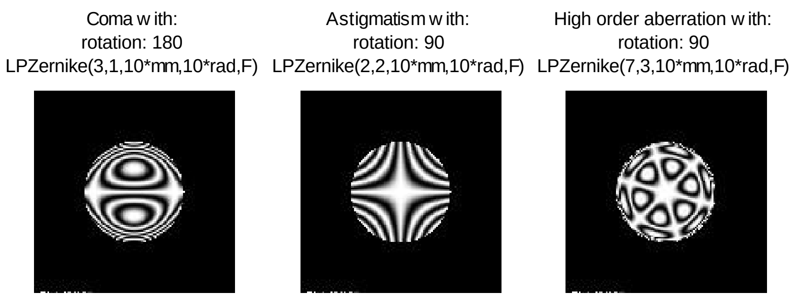
Figure 24 Shearing interferograms of spherical aberration, coma and astigmatism.

6.2 Rotational shearing interferometer

In the rotational shear interferometer the beam interferes with a copy of itself rotated by an angle α around the optical axis. This interferometer is useful for detecting asymmetrical aberrations. We shall only consider a bare-bone (no propagation and diffraction) model of such an interferometer. The results of aberration calculations for coma, astigmatism and high order Zernike aberration are shown in Figure 25.

```
% man0021.m
%
% LightPipes for Matlab
% Rotational Shear interferometer with aberrated wavefront.

clear;

m=1;
cm=1e-2*m;
mm=1e-3*m;
nm=1e-9*m;
rad=1;

size=4*cm;
lambda=500*nm;
N=256;
Rp=1*cm;
Rplate=0.5;

figure;

% 1) Coma:
PhiRot=180;
nZer=3;
mZer=1;
RZer=10*mm;
AZer=10*rad;
F=LPBegin(size,lambda,N);
F=LPCircAperture(Rp,0,0,F);
F=LPZernike(nZer,mZer,RZer,AZer,F);
F1=LPIntAttenuator(Rplate,F);
F2=LPIntAttenuator(1-Rplate,F);
F2=LPInterpol(size,N,0,0,PhiRot,1,F2);
F=LPBeamMix(F1,F2);
I=LPIntensity(1,F);
str1=sprintf('Coma with:\nrotation:
%d\nLPZernike(%d,%d,%d*mm,%d*rad,F)',PhiRot,nZer,mZer,RZer/mm,AZer);
subplot(1,3,1),imshow(I), title(str1,'FontSize',8);
```

```
% 2) Astigmatism:
PhiRot=90;
nZer=2;
mZer=2;
RZer=10*mm;
AZer=10*rad;
subplot(1,3,2),imshow(I),title(str1);
F=LPBegin(size,lambda,N);
F=LPCircAperture(Rp,0,0,F);
F=LPZernike(nZer,mZer,RZer,AZer,F);
F1=LPIntAttenuator(Rplate,F);
F2=LPIntAttenuator(1-Rplate,F);
F2=LPInterpol(size,N,0,0,PhiRot,1,F2);
F=LPBeamMix(F1,F2);
I=LPIntensity(1,F);
str1=sprintf('Astigmatism with:\nrotation:
%d\nLPZernike(%d,%d,%d*mm,%d*rad,F)',PhiRot,nZer,mZer,RZer/mm,AZer);
subplot(1,3,2),imshow(I),title(str1,'FontSize',8);

% 3) High order aberration:
PhiRot=90;
nZer=7;
mZer=3;
RZer=10*mm;
AZer=10*rad;
F=LPBegin(size,lambda,N);
F=LPCircAperture(Rp,0,0,F);
F=LPZernike(nZer,mZer,RZer,AZer,F);
F1=LPIntAttenuator(Rplate,F);
F2=LPIntAttenuator(1-Rplate,F);
F2=LPInterpol(size,N,0,0,PhiRot,1,F2);
F=LPBeamMix(F1,F2);
I=LPIntensity(1,F);
str1=sprintf('High order aberration with:\nrotation:
%d\nLPZernike(%d,%d,%d*mm,%d*rad,F)',PhiRot,nZer,mZer,RZer/mm,AZer);
subplot(1,3,3),imshow(I),title(str1,'FontSize',8);
flnm=sprintf('(man0021.emf)');flnm=strcat('\it',flnm);
text(100,500,flnm);

print -dmeta '..\figures\man0021';
```


Coma w ith:
rotation: 180
LPZernike(3,1,10*mm,10*rad,F)
Astigmatism w ith:
rotation: 90
LPZernike(2,2,10*mm,10*rad,F)
High order aberration w ith:
rotation: 90
LPZernike(7,3,10*mm,10*rad,F)

(man0021.emf)

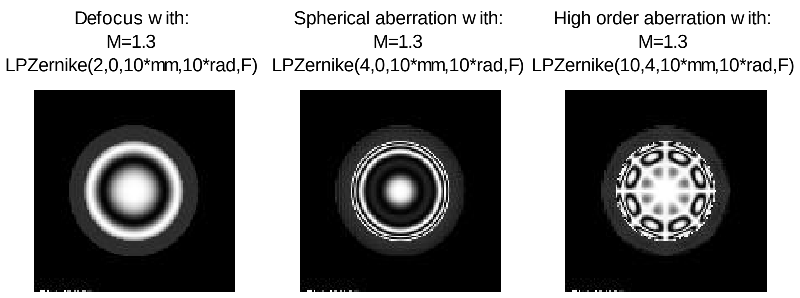
Figure 25 Rotational interferograms of coma, astigmatism and high order Zernike aberration.

6.3 Radial shear interferometer

In the radial shearing interferometer the beam interferes with a copy of itself magnified by a factor M. This interferometer is useful for detecting axisymetrical aberrations. As for the previous case, we shall consider only a bare-bone (no propagation and diffraction) model of such an interferometer:

```
% man0022.m
%
% LightPipes for Matlab
% Radial Shear interferometer.

clear;

m=1;
cm=1e-2*m;
mm=1e-3*m;
nm=1e-9*m;
rad=1;

size=4*cm;
lambda=500*nm;
N=256;
Rp=1*cm;
Rplate=0.5;
figure(1);

M=1.3;
nZer=2;
mZer=0;
RZer=10*mm;
AZer=10*rad;
F=LPBegin(size,lambda,N);
F=LPCircAperture(Rp,0,0,F);
F=LPZernike(nZer,mZer,RZer,AZer,F);
F1=LPIntAttenuator(Rplate,F);
F2=LPIntAttenuator(1-Rplate,F);
F1=LPInterpol(size,N,0,0,0,M,F1);
F=LPBeamMix(F1,F2);
I=LPIntensity(1,F);
str1=sprintf('Defocus
with:\nM=%3.1f\nLPZernike(%d,%d,%d*mm,%d*rad,F)',M,nZer,mZer,RZer/m
m,AZer);
subplot(1,3,1),imshow(I),title(str1,'fontsize',8);

M=1.3;
```

```
nZer=4;
mZer=0;
RZer=10*mm;
AZer=10*rad;
F=LPBegin(size,lambda,N);
F=LPCircAperture(Rp,0,0,F);
F=LPZernike(nZer,mZer,RZer,AZer,F);
F1=LPIntAttenuator(Rplate,F);
F2=LPIntAttenuator(1-Rplate,F);
F1=LPInterpol(size,N,0,0,0,M,F1);
F=LPBeamMix(F1,F2);
I=LPIntensity(1,F);
str1=sprintf('Spherical aberration
with:\nM=%3.1f\nLPZernike(%d,%d,%d*mm,%d*rad,F)',M,nZer,mZer,RZer/m
m,AZer);
subplot(1,3,2),imshow(I),title(str1,'fontsize',8);

M=1.3;
nZer=10;
mZer=4;
RZer=10*mm;
AZer=10*rad;
F=LPBegin(size,lambda,N);
F=LPCircAperture(Rp,0,0,F);
F=LPZernike(nZer,mZer,RZer,AZer,F);
F1=LPIntAttenuator(Rplate,F);
F2=LPIntAttenuator(1-Rplate,F);
F1=LPInterpol(size,N,0,0,0,M,F1);
F=LPBeamMix(F1,F2);
I=LPIntensity(1,F);
str1=sprintf('High order aberration
with:\nM=%3.1f\nLPZernike(%d,%d,%d*mm,%d*rad,F)',M,nZer,mZer,RZer/m
m,AZer);
subplot(1,3,3),imshow(I),title(str1,'fontsize',8);
flnm=sprintf('(man0022.emf)');flnm=strcat('\it',flnm);
text(100,500,flnm);

print -dmeta '..\figures\man0022';
```


Defocus w ith:
M=1.3
LPZernike(2,0,10*mm,10*rad,F)
Spherical aberration w ith:
M=1.3
LPZernike(4,0,10*mm,10*rad,F)
High order aberration w ith:
M=1.3
LPZernike(10,4,10*mm,10*rad,F)

(man0022.emf)

Figure 26 Radial shearing interferometer. Interferograms of defocus, spherical aberration and higher order aberration.

6.4 Twyman-Green interferometer

An example of the Twyman-Green interferometer is shown in Figure 27. Mirror 1 is plane and mirror 2 is aberrated (coma) with an aberration amplitude of 1μm. The beam splitter is ideal and divides the beam 3:7.

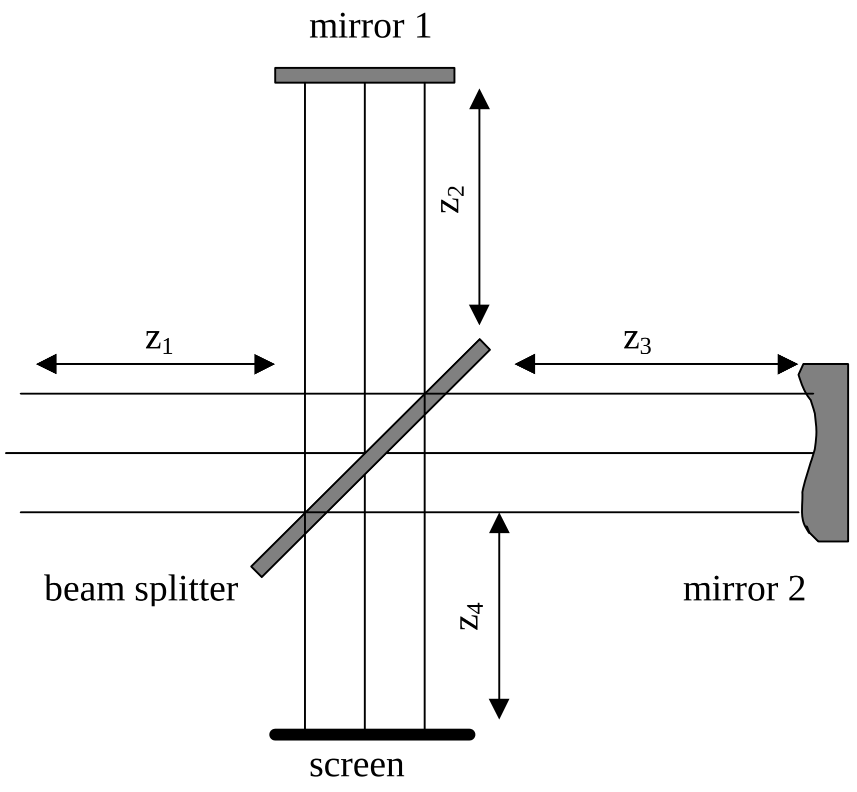


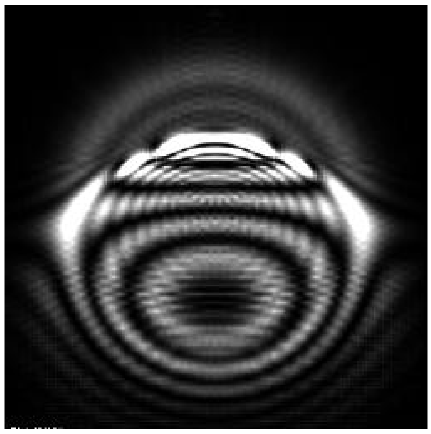
Figure 27 Twyman-Green interferometer.

The Twyman-Green interferometer can be simulated with the next m-file:

```
% man0023.m
%
% LightPipes for Matlab
% Twyman Green interferometer.
% The Twyman green interferometer consists of a beamsplitter
% and two mirrors. One of the mirrors
% is perfect, the other can be investigated for aberrations.

clear;

m=1;
cm=1e-2*m;
mm=1e-3*m;
nm=1e-9*m;
```

```
size=30*mm;
lambda=500*nm;
R=5*mm;
N=250;
z1=50*cm;
z2=40*cm;
z3=40*cm;
z4=100*cm;
RBS=0.3;
nz=3;
mz=1;
Rz=0.005;
Az=25;

F=LPBegin(size,lambda,N); F=LPCircAperture(R,0,0,F);
F=LPForvard(z1,F);
F1=LPIntAttenuator(RBS,F); F2=LPIntAttenuator(1-RBS,F);
F1=LPForvard(2*z2,F1); F2=LPForvard(z3,F2);
F2=LPZernike(nz,mz,Rz,Az,F2);
F2=LPForvard(z3,F2);
F1=LPIntAttenuator(1-RBS,F1); F2=LPIntAttenuator(RBS,F2);
F=LPBeamMix(F1,F2);
F=LPForvard(z4,F); F=LPInterpol(0.012,250,0,0,0,1,F);
I=LPIntensity(1,F)*4;

figure;
imshow(I);
title('Twyman Green interferometer with Zernike aberration');
flnm=sprintf('(man0023.emf)');flnm=strcat('\it',flnm);
text(200,270,flnm);
print -dmeta '..\figures\man0023';
```

Twyman Green interferometer with Zernike aberration



(man0023.emf)

Figure 28 Twyman-Green interferometer.

6.5 Propagation in a lens-like / absorptive medium.

In this example we model the propagation of a Gaussian beam in a lens-like waveguide. The profile of the refractive index is chosen such that the beam preserves approximately its diameter in the waveguide (we use the fundamental mode). We'll consider the propagation of an axial mode, tilted with respect to the waveguide axis and a non-axial mode. In the next simulation a quadratic index profile of the absorptive lens-like medium is made. In Figure 29 we use the approximation for the profile of the refractive coefficient in the form: $n^2 = n_0{}^2 - n_0 n_1 r^2$. It is a well-known fact [9] that the half-width of the fundamental Gaussian mode of a lens-like waveguide is defined as: $w_0{}^2 = \dfrac{2}{k(n_0 n_1)^{\frac{1}{2}}}$. For a waveguide of 1x1mm, n_0=1.5, n_1=400 and λ=1μm, the Gaussian mode has a diameter of 226μm. In Figure 29 we demonstrate the propagation of a (two) Gaussian beam(s) through the waveguide.

```
% man0024.m
%
% LightPipes for Matlab
% Propagation in a lens-like / absorptive medium.

clear;

m=1;
nm=1e-9*m;
microns=1e-6*m;
mm=1e-3*m;
cm=1e-2*m;
rad=1; mrad=1e-3*rad;

lambda=1*microns;
size=1*mm;
N=100;
n0=1.5; n1=400/m/m;
R=113*microns;
tilt=1*mrad; shift=0.2*mm;
dz=1*mm;
kmax=100;

%definition of the real and imaginary parts of the refractive index distribution:
p=[1:N]; q=[1:N];
x(p)=-size/2+p*size/N; y(q)=-size/2+q*size/N;
for ii=1:N
  for jj=1:N
    f(ii,jj)=sqrt(n0*n0-n0*n1*(x(ii)*x(ii)+y(jj)*y(jj)));
    n(ii,jj)=f(ii,jj)*(1-i);
  end
```

```
end
figure(1);subplot(1,2,1); mesh(real(n)); axis off;
title('real part of the refractive index');
subplot(1,2,2); mesh(imag(n)); axis off;
title('imaginary part of the refractive index');
flnm=sprintf('(man0024-0.bmp)');flnm=strcat('\it',flnm);
text(10,-40,-1.5,flnm);
print -dbitmap '..\figures\man0024_0';

%Problem #1: tilted Gaussian beam, incident
%           in the centre of the waveguide:
Field=LPBegin(size,lambda,N);
Field=LPGaussAperture(R,0,0,1,Field);
Field=LPTilt(tilt,0,Field);

%Problem #2: Gaussian beam shifted from
%           and parallel to the waveguide axis:
%Field=LPBegin(size,lambda,N);
%Field=LPGaussAperture(R,shift,0,1,Field);

%Problem #3: Two Gaussian beams incident parallel to
%           the waveguide axis:
%Field=LPBegin(size,lambda,N);
%Field1=LPGaussAperture(R,shift,0,1,Field);
%Field2=LPGaussAperture(R,-shift,0,1,Field);
%Field=LPBeamMix(Field1,Field2);

%Propagation through the wave guide:
for k=1:kmax
        progress(k,kmax,2);
   Field=LPSteps(dz,10,n,Field);
   Int=LPIntensity(0,Field);
        figure(3); axis off;
        subplot(10,10,k);
        imshow(Int);
        drawnow;
   Ix(p)=Int(p,N/2);
   Iy(q)=Int(N/2,q);
   Mx(k,p)=Ix(p);
   My(k,q)=Iy(q);
end
figure(4);
colormap(gray);
subplot(1,2,1); surfl(Mx);shading interp; axis off; view(39,44);
title('Radial distribution in x-direction');
subplot(1,2,2); surfl(My);shading interp; axis off; view(39,44);
```

```
title('Radial distribution in y-direction');

%Output problem #1:
flnm=sprintf('(man0024-1.emf)');flnm=strcat('\it',flnm);
text(100,0,-0.2,flnm);
print -dmeta '..\figures\man0024_1';
figure(3);
flnm=sprintf('(man0024-1-1.emf)');flnm=strcat('\it',flnm);
text(-100,N+80,flnm);
print -dmeta '..\figures\man0024_1_1';

%Output problem #2:
%flnm=sprintf('(man0024-2.emf)');flnm=strcat('\it',flnm);
%text(100,0,-0.1,flnm);
%print -dmeta '..\figures\man0024_2';
%figure(3);
%flnm=sprintf('(man0024-2-1.emf)');flnm=strcat('\it',flnm);
%text(-100,N+80,flnm);
%print -dmeta '..\figures\man0024_2_1';

%Output problem #3:
%flnm=sprintf('(man0024-3.emf)');flnm=strcat('\it',flnm);
%text(100,0,-0.8,flnm);
%print -dmeta '..\figures\man0024_3';
%figure(3);
%flnm=sprintf('(man0024-3-1.emf)');flnm=strcat('\it',flnm);
%text(-100,N+80,flnm);
%print -dmeta '..\figures\man0024_3_1';
```

Figure 29** **Problem #1: Propagation of a Gaussian beam, tilted with respect to the waveguide axis, in a lens-like absorptive medium .
Problem #2: Propagation of a Gaussian beam, shifted from the centre and parallel to the waveguide axis
Problem #3: Propagation of two shifted Gaussian beams incident parallel to the waveguide axis.
The scripts for problems #2 and #3 have been commented away with % characters.

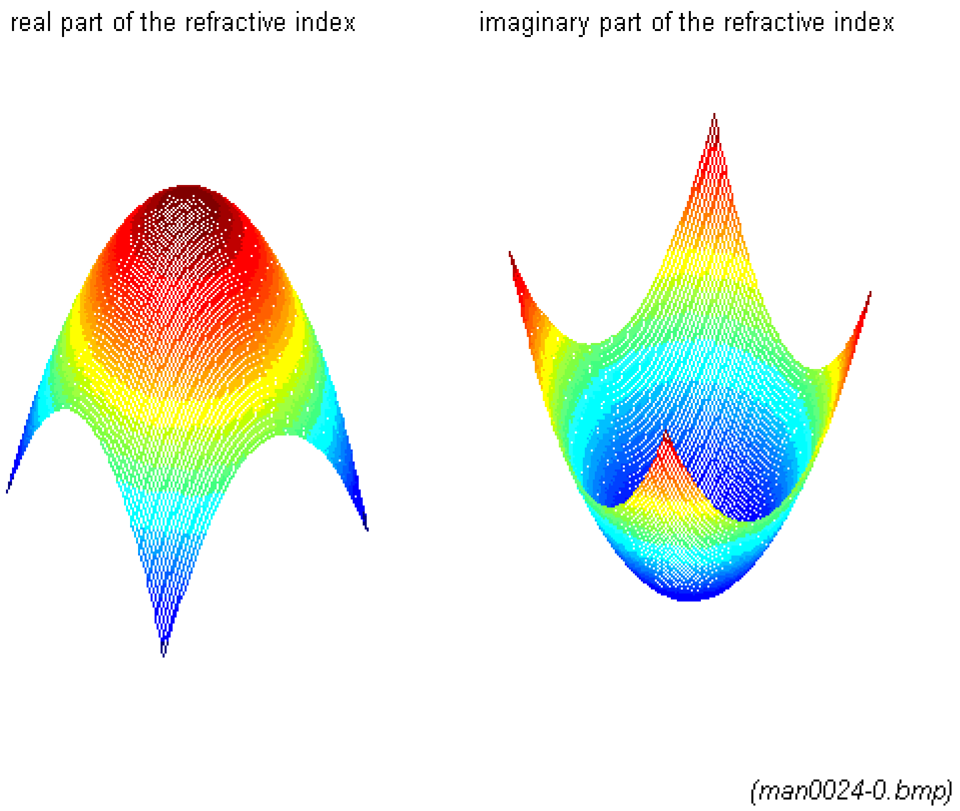


Figure 30 Real (left) and imaginary (right) parts of the refractive index profile.

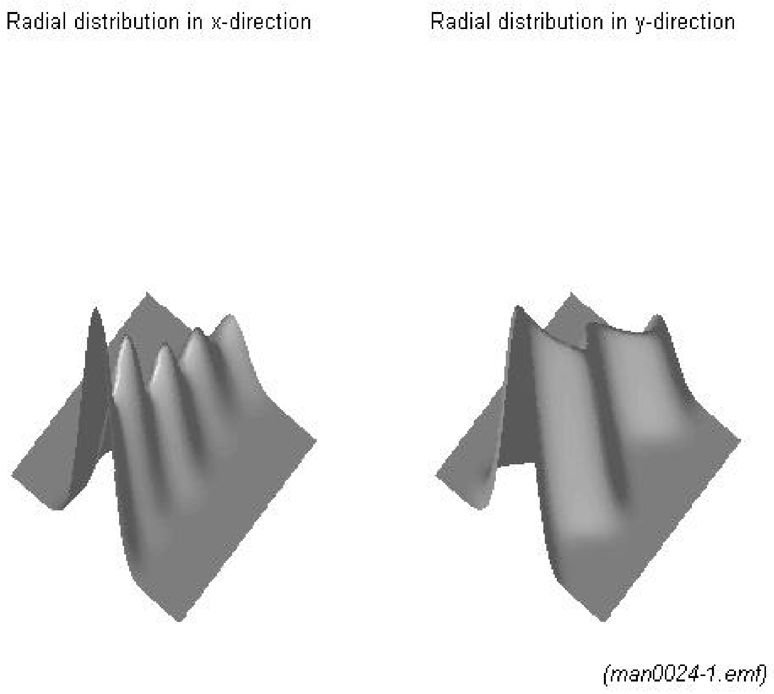

Radial distribution in x-direction
Radial distribution in y-direction
(man0024-1.emf)

Figure 31 Radial distributions of the energy distributions in the x- and y-direction.

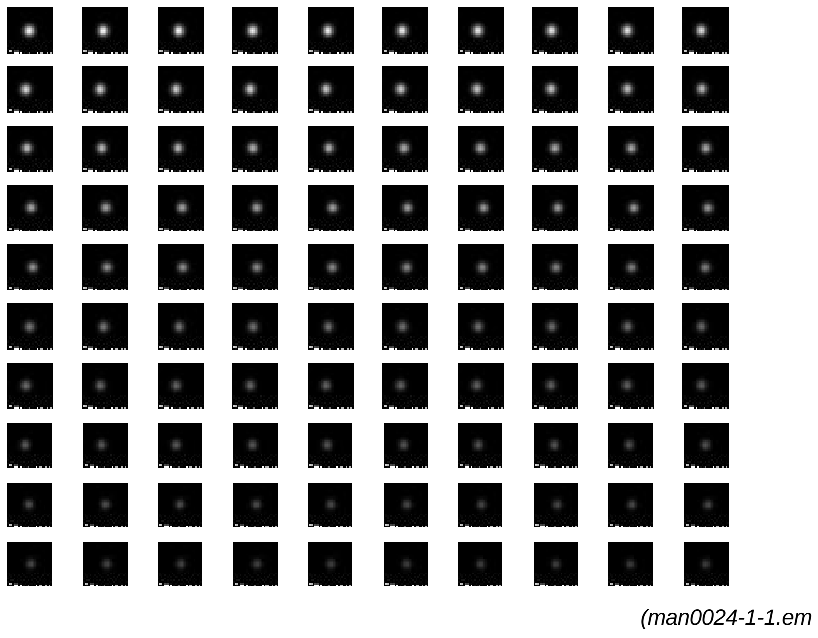


Figure 32 As Figure 31 Radial distributions of the energy distributions in the x- and y-direction. (from upper left to lower right).

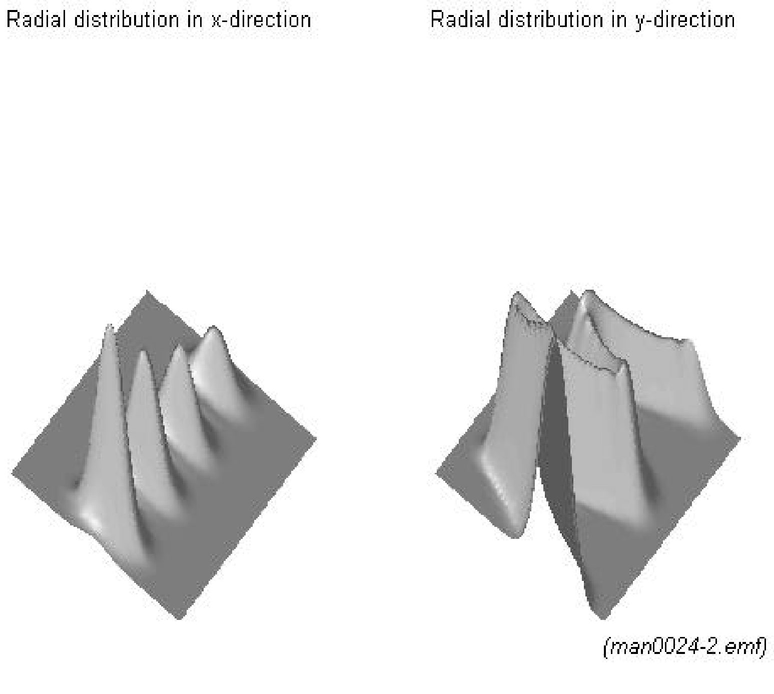

Radial distribution in x-direction
Radial distribution in y-direction
(man0024-2.emf)

Figure 33 Radial energy distributions of a Gaussian beam incident shifted and parallel to the waveguide axis.

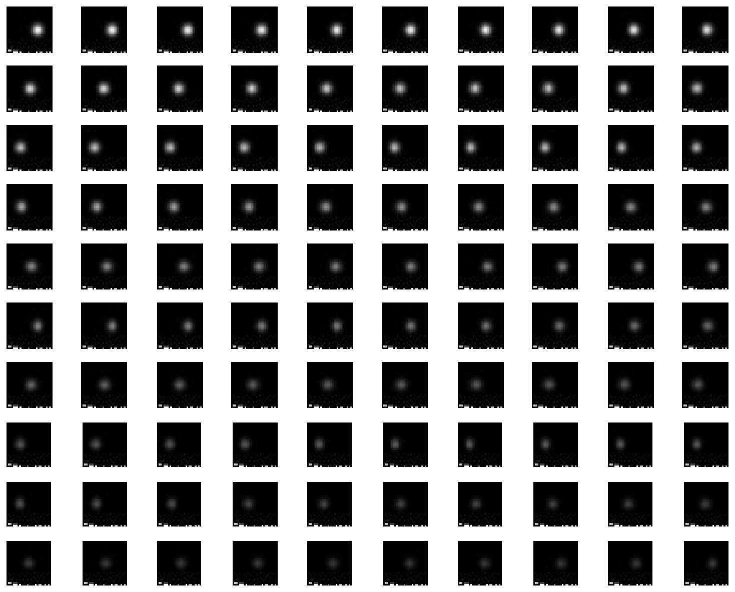


Figure 34 As Figure 33 Radial energy distributions of a Gaussian beam incident shifted and parallel to the waveguide axis. (from upper left to lower right).

In Figure 35 the results of the simulation of two Gaussian beams incident parallel to the waveguide axis are shown. For this the simulation was modified with the following fragment:

```
Field1=LPGaussAperture(R,0.2*mm,0,1,Field);
Field2=LPGaussAperture(R,-0.2*mm,0,1,Field);
Field=LPBeamMix(Field1,Field2);
```

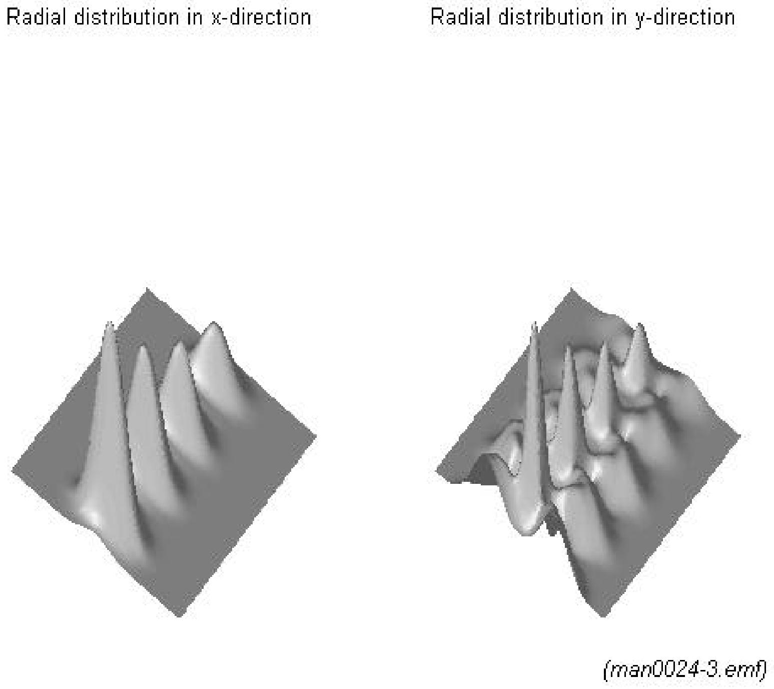

Radial distribution in x-direction
Radial distribution in y-direction
(man0024-3.emf)

Figure 35 Propagation of two Gaussian beams incident parallel to the waveguide axis.

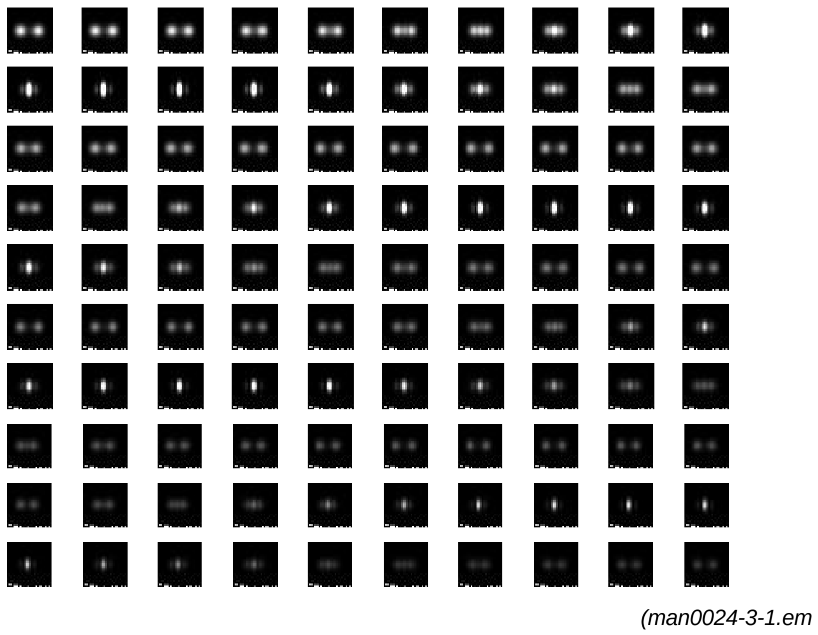


Figure 36 As Figure 35 Propagation of two Gaussian beams incident parallel to the waveguide axis. (from upper left to lower right).

6.6 Michelson interferometer

In this example, Figure 37, we simulate a Michelson interferometer. This interferometer consists of a positive

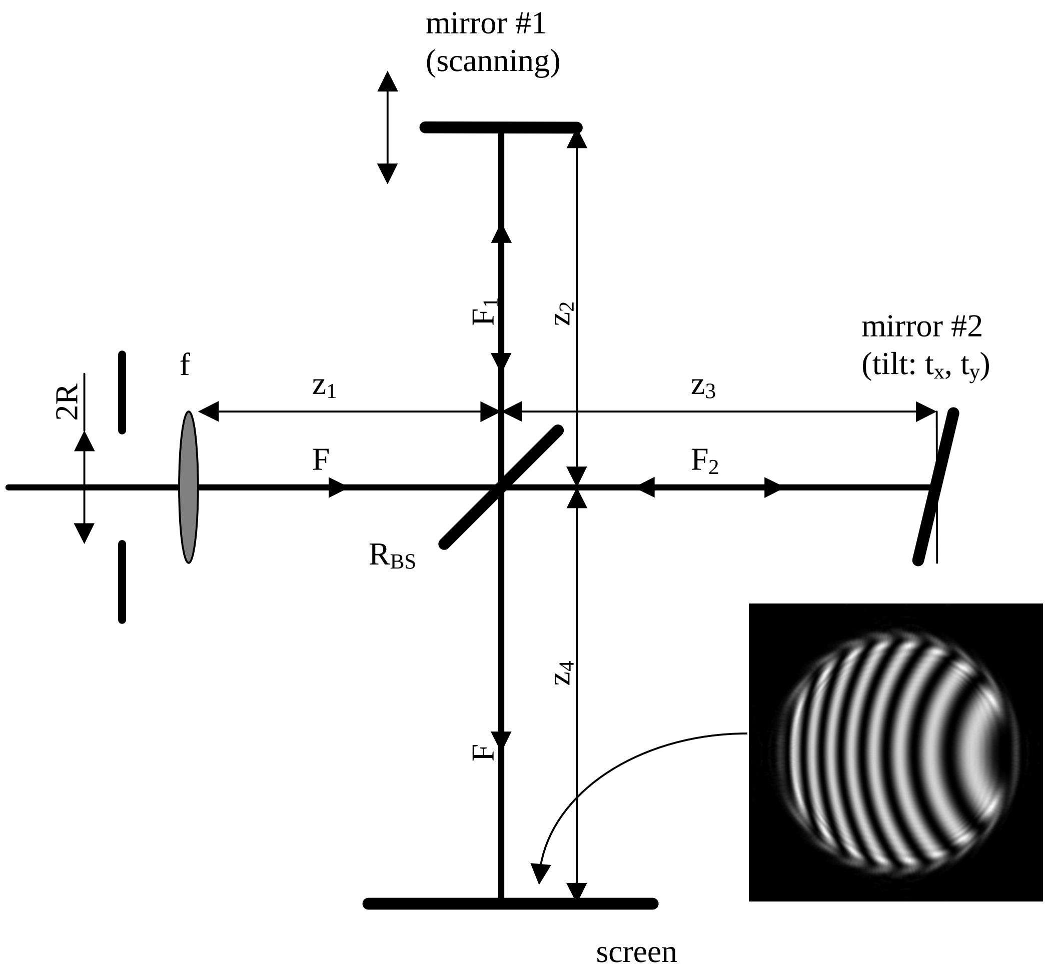


Figure 37 Michelson interferometer with one mirror tilted.

lens with focal length f, a beamsplitter, R_{BS}, and two mirrors with 100% reflectivity. One of the mirrors can be tilted and the other can be moved. The focal length of the lens is chosen such that an image of the aperture is formed behind mirror #2. When the light has a sufficient large coherence length fringes can be observed on the screen when the distances from the beamsplitter to the two mirrors are different. The following m-file simulates the Michelson interferometer and makes a movie of the fringe pattern on the screen.

```
% man0025.m
%
% LightPipes for Matlab
% Simulation of a Michelson interferometer
```

```
clear;

m=1; nm=1e-9*m; mm=1e-3*m; cm=1e-2*m;
rad=1; mrad=1e-3*rad;

lambda=500*nm;
size=25*mm;
N=250;
R=12*mm;
z1=10*cm;
z4=10*cm;
RBS=0.5;
ty=0.0*mrad;
tx=0.2*mrad;
f=600*cm;
z3=30*cm;
z20=50*cm;

%A weak converging beam using a weak positive lens:
F=LPBegin(size,lambda,N);
F=LPCircAperture(R,0,0,F);
F=LPLens(f,0,0,F);

%Propagation to the beamsplitter:
F=LPForvard(z1,F);

%Splitting the beam and propagation to mirror #2:
F2=LPIntAttenuator(1-RBS,F);
F2=LPForvard(z3,F2);

%Introducing tilt and propagating back to the beamsplitter:
F2=LPTilt(tx,ty,F2);
F2=LPForvard(z3,F2);
F2=LPIntAttenuator(RBS,F2);

%Splitting off the second beam:
F10=LPIntAttenuator(RBS,F);

%Initializing the screen and the array for storage of the movie:
figure(1);
I=ones(N);
imshow(I);
M=moviein(32);

%Scanning mirror #1:
for FRAME=1:32
```

```
        z2=z20+(lambda/32)*(FRAME-1);
        F1=LPForvard(z2*2,F10);
  F1=LPIntAttenuator(1-RBS,F1);
  %Recombining the two beams and propagation to the screen:
        F=LPBeamMix(F1,F2);
        F=LPForvard(z4,F);
  I=LPIntensity(1,F);
  %plot the intensity on the screen (bitmap)
  %and fill the movie-matrix:
  imshow(I);
  M(:,FRAME)=getframe;
end;

%save the results for later use:
save '..\mat-files\michelson';

%save the last picture:
% if tx=0.2*mrad:
print -dmeta '..\figures\man0025_1';

% if tx=0:
% print -dmeta '..\figures\man0025_2';

%play the movie:
movie(M,5);
```

Figure 38 Simulation of a Michelson interferometer with a tilted mirror.

You can play the movie in Matlab by loading the mat-file produced by the progarm and starting the movie with the following commands (it repeats the movie 5 times, see Matlab help for more details):

```
» load '..\mat-files\Michelson';
» imshow(I);
» movie(M,5);
```

If both mirrors are aligned (t_x=t_y=0) circular fringes are observed on the screen:

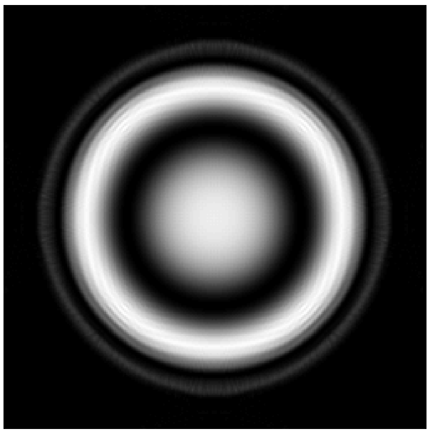


Figure 39 Circular fringes with aligned mirrors.

6.7 Unstable laser resonator

The unstable resonator can be simulated starting with an arbitrary field that circulates between the mirrors towards a steady state solution: the eigen-mode of the resonator. In this example we simulate a positive branch, confocal unstable resonator . In stead of mirrors, we use a lens-guide of alternating positive and negative lenses, separated a distance, L, the resonator length.

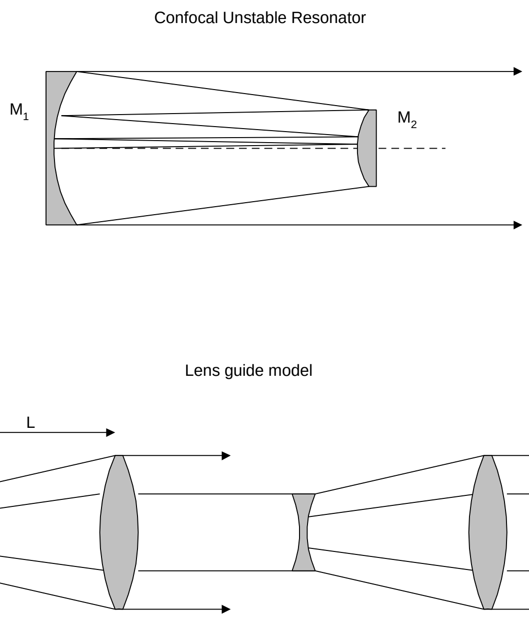


Figure 40 An unstable confocal resonator and its equivalent lens guide.

The following m-file calculates the resonator mode. Starting from random noise the field evolves to a steady-state.

```
% man0026.m
%
% LightPipes for Matlab
% Hard-edge confocal unstable resonator with gain.

clear;

m=1; nm=1e-9*m; mm=1e-3*m; cm=1e-2*m;

lambda=308*nm; size=14*mm; N=100; w=5.48*mm;
f1=-10*m; f2=20*m; L=10*m; Isat=1.0; alpha=1e-4; Lgain=1e4;

%Part one: Calculation of the evolution of the resonator mode from random
noise:
F=LPBegin(size,lambda,N); F=LPRandomIntensity(2,F);
F=LPRandomPhase(5,1,F);
for i=1:10
```

```
   F=LPRectAperture(w,w,0,0,0,F);   F=LPGain(Isat,alpha,Lgain,F);
   F=LPLensFresnel(f1,L,F);   F=LPGain(Isat,alpha,Lgain,F);
   F=LPLensFresnel(f2,L,F);
   SR(i)=LPStrehl(F);
   F=LPInterpol(size,N,0,0,0,1,F);
   fprintf('Round trip %d Strehl ratio= %f \n',i,SR(i));
   F2=LPRectScreen(w,w,0,0,0,F);
   I=LPIntensity(1,F2);
         str1=sprintf('%d',i);
   figure(1);   subplot(2,5,i);   imshow(I); title(str1);
end
print -dmeta '..\figures\man0026_1';

%Part two: Plot the Strehl ratio as a function of the # of roundtrips:
F2=LPConvert(F2);
I=LPIntensity(1,F2);
figure(2);
subplot(2,1,1); plot(SR); xlabel('Number of Roundtrips'); ylabel('Strehl ratio');
print -dbitmap '..\figures\man0026_2';

%Part three: Plot the intensity distribution in 3D:
figure(3); mesh(I); axis off;
title('Intensity distribution just behind the outcoupler');
print -dbitmap '..\figures\man0026_3';

%Part four: Far-field calculation:
z=1*m; f=40*m;
ff=z*f/(f-z);
F2=LPLens(f,0,0,F2);
F2=LPLensFresnel(ff,z,F2);
F2=LPConvert(F2);
I2=LPIntensity(1,F2);
figure(4);
mesh(I2); axis off;
title('Intensity distribution in the far field');
print -dbitmap '..\figures\man0026_4';
```

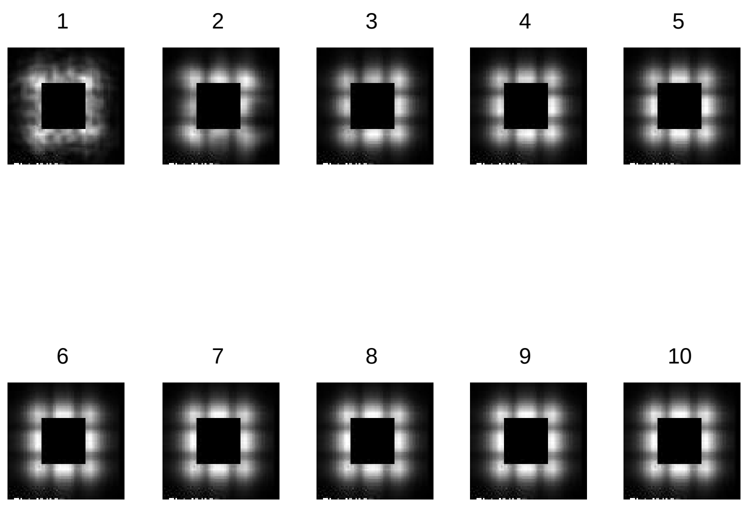


Figure 41 Evolution of the near-field intensity pattern from noise.

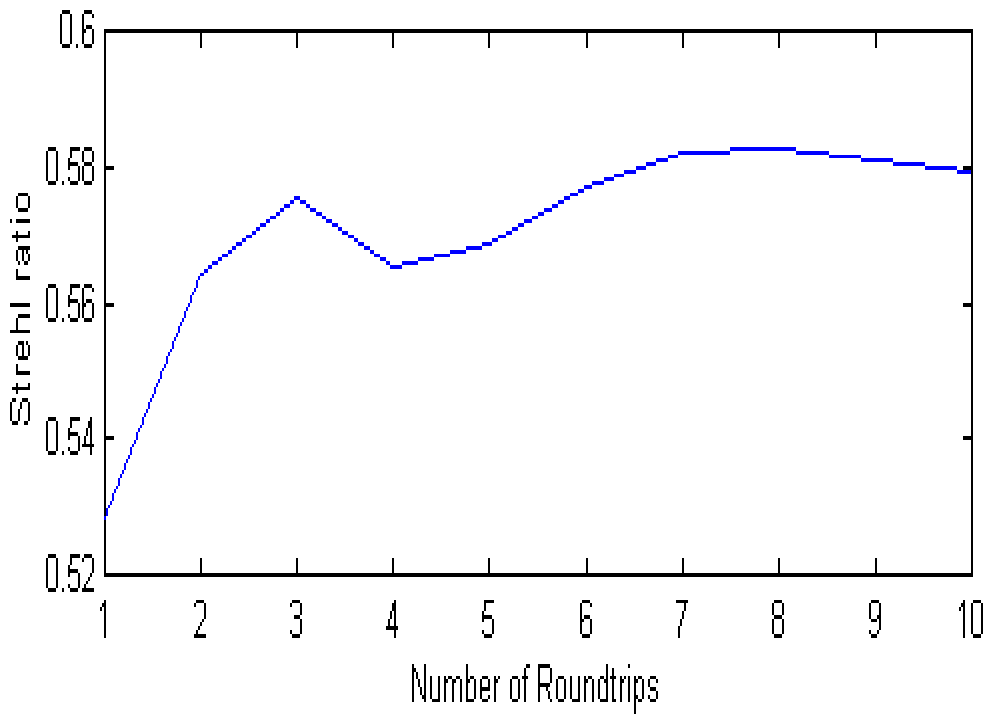


Figure 42 The Strehl ratio as a function of the number of roundtrips.

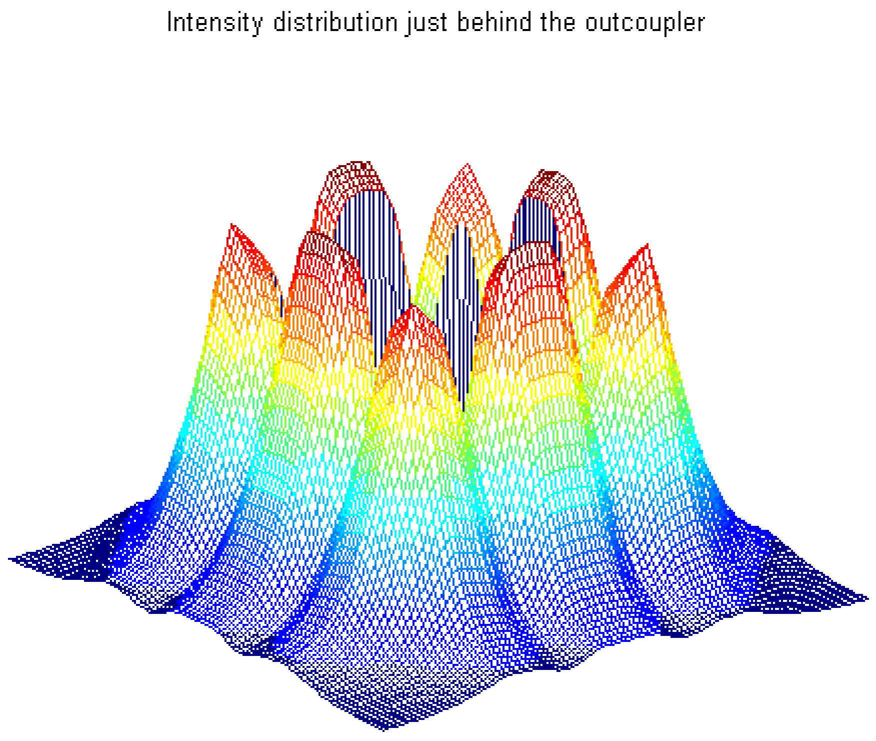


Figure 43 Outcoupled intensity distribution.

The far-field can be calculated by propagation of the field to the focus of a lens. Because the density of the grid soon becomes too low for a correct calculation we use spherical co-ordinates using the LPLensFresnel command. The LPLensFresnel(z,z) command, however, cannot be used for propagation to the focus, because the program tries to define a zero grid dimension and hence will generate a floating point error. To solve this problem we can use an extra weak positive lens with focal length f and propagate the field to a distance z using spherical co-ordinates with the LPLensFresnel(f_f,z) command. z is the focal length of the combined lenses:

or:

$$\frac{1}{z} = \frac{1}{f_f} + \frac{1}{f}$$

$$f_f = \frac{z f}{f - z}$$

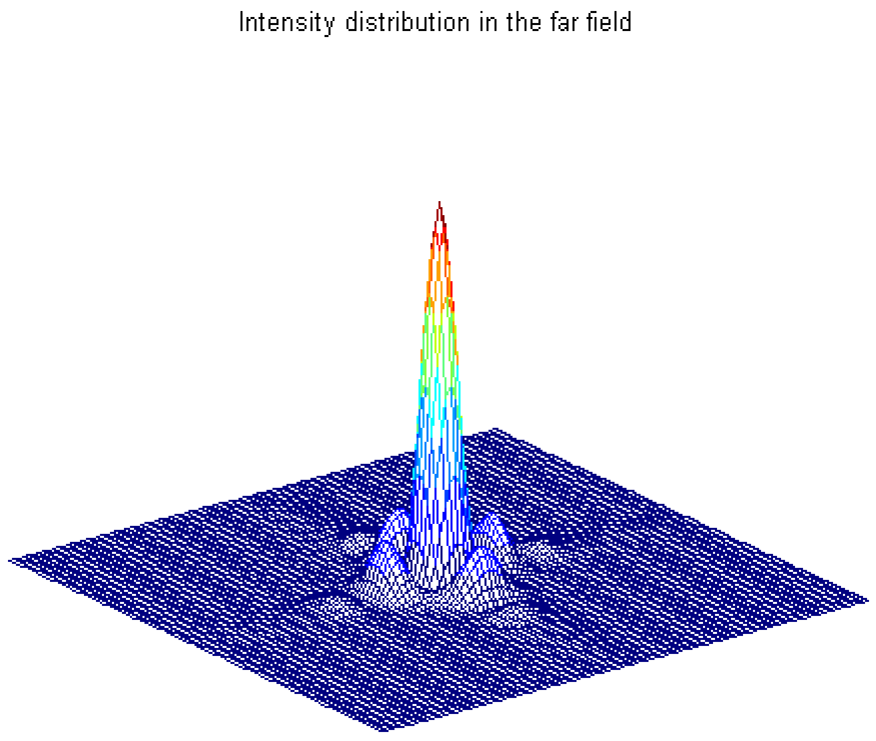


Figure 44 Far field intensity distribution.

6.8 Inverse problem: reconstruction the phase from measusred intensities.

Suppose we have measured two intensity distributions of a beam with for example a ccd camera: one in the near field and one after propagation of a distance z. The two intensity distributions have been stored on disk. The knowledge of these intensity patterns is sufficient to reconstruct the phase of the beam. To do this we repeatedly substitute the far field intensity in an arbitrary (we chose a uniform field) field, propagate this field back to the near field position, substitute the near field intensity and propagate to the far field and so on. After sufficient iterations the phase and the intensity of the beam become stable after each iteration and the phase distribution has been recovered. The next script demostrates the idea. The near and far field intensities are from a HeNe laser beam with an aberrated phase front. They have been measured at positions 2m from each other.

```
% man0027.m
%
% LightPipes for Matlab
% Phase reconstruction

clear;

m=1;
nm=1e-9*m;
mm=1e-3*m;
cm=1e-2*m;
rad=1;

lambda=632.8*nm;
size=11*mm;
N=128;
z=2*m;

%Read the experimental data of the near and the far field
%intensity patterns from disk:
fid=fopen('near.dat','rb');
Inear=fread(fid,[N,N],'double');
fclose(fid);
fid=fopen('far.dat','rb');
Ifar=fread(fid,[N,N],'double');
fclose(fid);

F=LPBegin(size,lambda,N);
FnearOrg=LPSubIntensity(Inear,F);
InearOrg=LPIntensity(1,FnearOrg);
FfarOrg=LPSubIntensity(Ifar,F);
IfarOrg=LPIntensity(1,FfarOrg);
```

```
n=100;
SizeNew=20*mm; Nnew=256;

%start the iteration:
F=LPBegin(size,lambda,N);
for k=1:n
        progress(k,n,1);
  F=LPSubIntensity(Ifar,F);
  F=LPInterpol(SizeNew,Nnew,0,0,0,1,F);
  F=LPForvard(-z,F);
  F=LPInterpol(size,N,0,0,0,1,F);
  F=LPSubIntensity(Inear,F);
  F=LPFresnel(z,F);
end
FnearRec=LPForvard(-z,F);
InearRec=LPIntensity(1,FnearRec);
FfarRec=LPFresnel(z,FnearRec);
IfarRec=LPIntensity(1,FfarRec);
figure(1); subplot(1,2,1); imshow(InearOrg);
subplot(1,2,2); imshow(InearRec);
print -dmeta '..\figures\man0027_1';

figure(2); subplot(1,2,1); imshow(IfarOrg);
subplot(1,2,2); imshow(IfarRec);
print -dmeta '..\figures\man0027_2';

PhiRec=LPPhase(FnearRec);
PhiRec=LPPhaseUnwrap(1,PhiRec);
figure(3);
surfl(PhiRec); colormap(gray); shading interp; axis off; view(-70,50);
print -dmeta '..\figures\man0027_3';
```

Figure 45 Reconstruction of the phase of a HeNe laser beam from two measured intensity patterns.

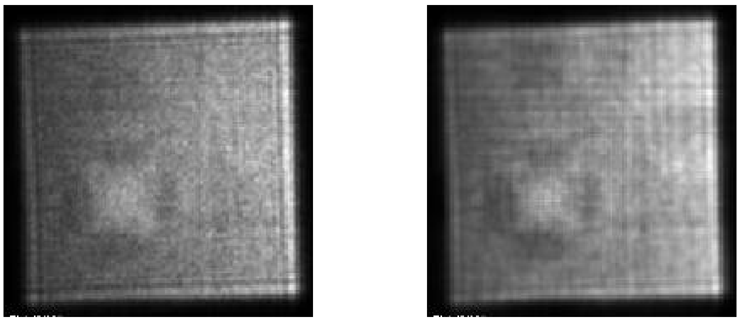


Figure 46 Original (left) and reconstructed (right) intensity patterns of the near field.

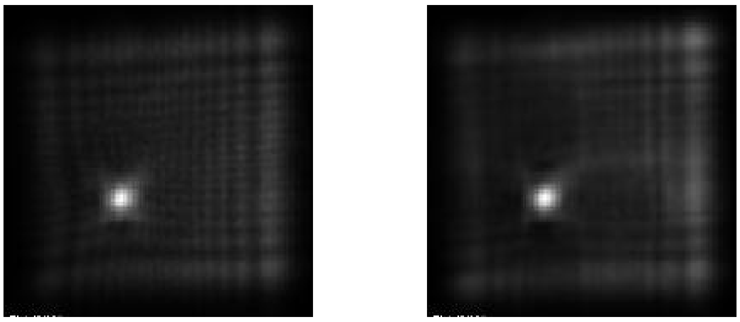


Figure 47 Original (left) and reconstructed (right) intensity patterns of the far field.

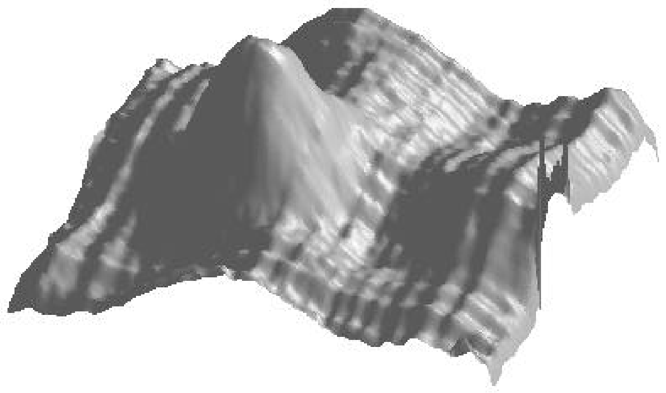
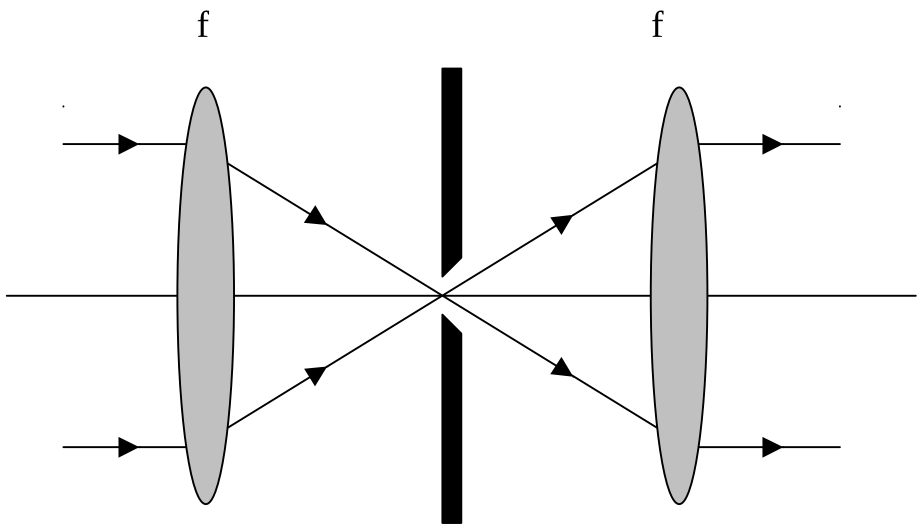


Figure 48 Reconstructed phase pattern of the beam in the near field.

6.9 Optical information processing.

LightPipes for Matlab can be used to model classical setups of Fourier-optics information processing. The next example models an optical computer performing Fourier-plane image filtering.



```
% man0028.m
%
% LightPipes for Matlab
% Optical filtering.
% In the focus of a lens is the Fourier transform of the input beam.
% Optical filtering can be achieved by placing an aperture
% or screen in that focus. A second lens performs an inverse Fourier transform.

clear;

m=1;
cm=1e-2*m;
mm=1e-3*m;
nm=1e-9*m;

size=10*mm;
lambda=500*nm;
f=1*m;
z=1*m;
R=1.5*mm;

% read the intensity mask from disk:
```

```
M=double(imread('dimesut','bmp'));
N=length(M);
F=LPBegin(size,lambda,N);

% place the mask in the beam:
F=LPSubIntensity(M,F);
I0=LPIntensity(0,F);

% Propagate through the first lens to the focus:
F=LPForvard(z,F); F=LPLens(f,0,0,F); F=LPForvard(f,F);
Ifour=LPIntensity(1,F);

% Filter the focus with an aperture or a screen to remove high- or low frequency
components
% and propagate through the second lens to the image plane:
Fhigh_freq=LPCircAperture(R,0,0,F); Fhigh_freq=LPForvard(f,Fhigh_freq);
Fhigh_freq=LPLens(f,0,0,Fhigh_freq); Fhigh_freq=LPForvard(z,Fhigh_freq);
Ihigh_freq=LPIntensity(1,Fhigh_freq);

Flow_freq=LPCircScreen(R,0,0,F);Flow_freq=LPForvard(f,Flow_freq);
Flow_freq=LPLens(f,0,0,Flow_freq); Flow_freq=LPForvard(z,Flow_freq);
Ilow_freq=LPIntensity(1,Flow_freq);

figure;
subplot(1,3,1); imshow(I0); axis off;
subplot(1,3,2); imshow(Ihigh_freq*2); axis off;
subplot(1,3,3); imshow(Ilow_freq*4); axis off;
print -dmeta '..\figures\man0028_1';

figure;
imshow(Ifour*500);
print -dmeta '..\figures\man0028_2';
```

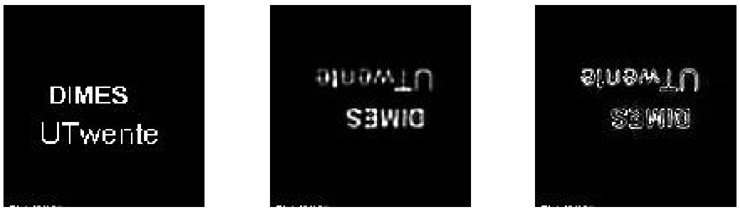


Figure 49 Original mask (left), filtered with high-frequency (middle), filtered with low-frequency (right).

The initial distribution and two filtered images are shown in Figure 49.

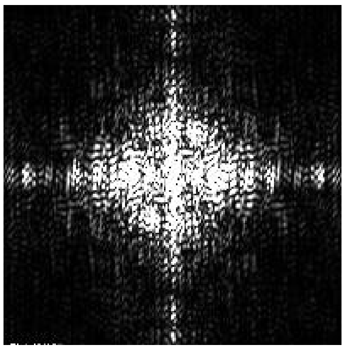


Figure 50 Intensity distribution in the focus.

In Figure 50 the intensity distribution in the focal plane of the two lenses is shown.

6.10 Generation and reconstruction of interferograms.

In the simplest case the interferogram (hologram) is generated by mixing of an aberrated beam, for example a perfect beam reflected by an imperfect mirror, with a plane wave:
In Figure 51 the phase distribution of an aberrated beam is shown. The beam has been made using the LPZernike command.

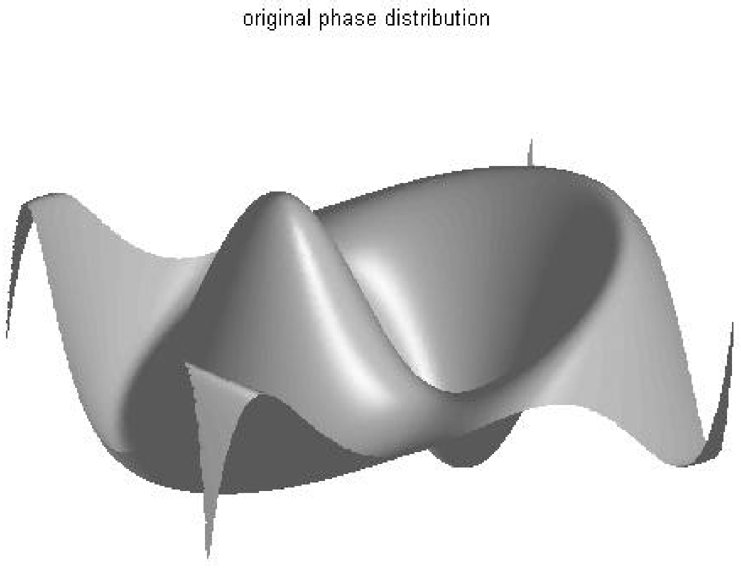


Figure 51 (Arbitrary) phase distribution made with the LPZernike command.

Mixing this beam with an ideal plane wave yields an interferferogram as shown in Figure 52.

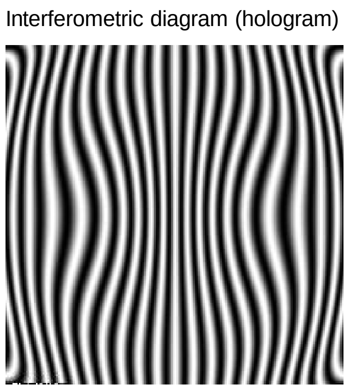


Figure 52 Interferometric diagram (hologram).

To reconstruct the phase from the obtained interferogram we shall build a model of the algorithm of phase reconstruction, using shift and filtering in the Fourier domain. Let the fringe pattern has the form:

$$g(x,y)=c(x,y)\exp(2pif_0x)+c^*(x,y)\exp(-2pif_0x) \qquad (6.1)$$

where:

$$c(x,y)=\frac{1}{2}\exp(iy\ \ (x,y)) \qquad (6.2)$$

where the phase *y (x,y)* contains information about the mirror shape and the term f_0x describes the wavefront tilt. Fourier transform of expression (6.1) gives:

$$G(f,y)=A(f,y)+C(f+f_0,y)+C^*(f-f_0,y) \qquad (6.3)$$

where the capitals *A* and *C* denote the Fourier spectra and *f* is the spatial frequency. We may take one of the two side-lobe spectra $C(f+f_0,y)$ or $C^*(f-f_0,y)$ and translate it to the origin with zero spatial frequency. Now we can take the inverse Fourier transform of the translated spectrum to obtain *c(x,y)* defined in the expressions (6.1) and (6.2). Calculating the complex logarithm of the expression (6.2) we obtain the phase *y (x,y)*:

$$iy\ \ (x,y)=\log[c(x,y)] \qquad (6.4)$$

The phase *y (x,y)* is obtained indeterminate to a factor 2π with its principal value localized in the range of -π…π. To obtain the continuous phase map a special unwrapping algorithm, removing discontinuities with an amplitude close to 2π is applied to the reconstructed phase map. The algorithm implementation includes the following steps shown in Figure 53:

1. After the substitution of the interferogram (hologram) into the field the spectrum of the initial intensity distribution is calculated with LPPipFFT. It consists of a central lobe and two side-lobes containing information about the phase.
2. Shift of the obtained spectrum to put one of the side lobes into the origin with the LPInterpol command.
3. Filtering, leaving only the side lobe containing the phase information. This operation is performed as a product with a rectangular spectral window LPRectAperture.
4. Reversed Fourier transform using LPPipFFT.
5. Calculation of the phase with LPPhase and unwrapping it with LPPhaseUnwrap.

```
% man0029.m
%
% LightPipes for Matlab
% Phase reconstruction from a hologram.

clear;

m=1;
cm=1e-2*m;
mm=1e-3*m;
nm=1e-9*m;
rad=1;
mrad=1e-3*rad;

size=10*mm;
```

```
lambda=1000*nm;
N=200;
nz=7; mz=-1;
R=7.2*mm; A=10;
theta=2*mrad;

% First we construct an aberrated beam using Zernike aberration:
Fab=LPBegin(size,lambda,N);
Fab=LPZernike(nz,mz,R,A,Fab);
Phase=LPPhase(Fab); Phase=LPPhaseUnwrap(1,Phase);

% Here we generate the reference beam, mix it with the aberrated beam
% to obtain the interferogram. We tilt the aberrated beam to get vertical fringes:
Fab=LPTilt(theta,0,Fab);

Fref=LPBegin(size,lambda,N); Fmixed=LPBeamMix(Fref,Fab);
Iinterferogram=LPIntensity(1,Fmixed);

% To reconstruct the phase of the aberrated beam we first substitude the
hologram in the (reference) beam:
Frec=LPBegin(size,lambda,N); Frec=LPSubIntensity(Iinterferogram,Frec);

% Then we Fourier transform the field and shift the spectral domain:
Frec=LPPipFFT(1,Frec); FrecShifted=LPInterpol(size,N,1*mm,0,0,1,Frec);

% Next we filter the beam in the Fourier domain and apply a reverse Fourier
transformation:
FrecShifted=LPRectAperture(1.5*mm,1,0,0,0,FrecShifted);
FrecShifted=LPPipFFT(-1,FrecShifted);

% Finally we calculate the phase and unwrap it
Phaserec=LPPhaseUnwrap(1,LPPhase(FrecShifted));

figure(1);
imshow(Iinterferogram);title('Interferometric diagram (hologram)');
print -dmeta '..\figures\man0029_1';

figure(2);
surfl(Phase); axis on; colormap(gray); shading interp; view(-21,36);title('original
phase distribution');axis off;
print -dmeta '..\figures\man0029_2';

figure(3);
surfl(Phaserec); axis on; colormap(gray); shading interp; view(-
21,36);title('reconstructed phase distribution');axis off;
print -dmeta '..\figures\man0029_3';
```

Figure 53 Algorithm to reconstruct the phase from the interferogram.

The result is shown in Figure 54.

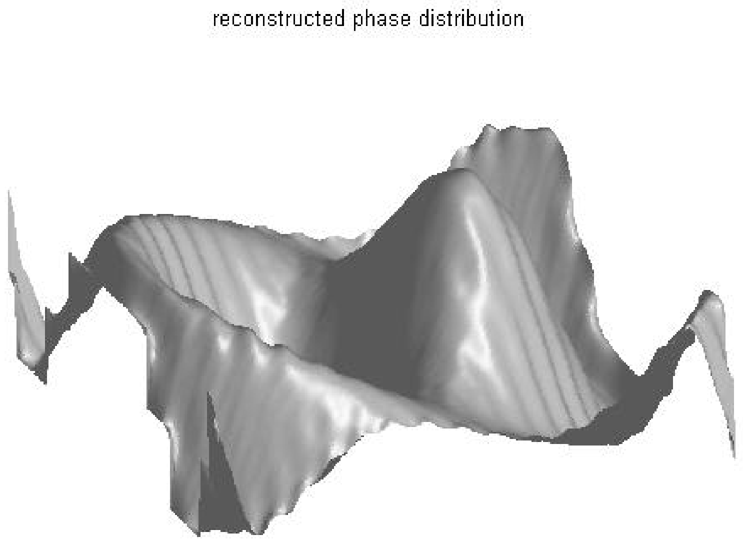


Figure 54 Reconstructed phase distribution.

The initial interferogram is even (symmetric) with respect to the vertical axis, but the phase is odd, thus the phase is reconstructed with inverted sign (this depends on the direction of the shift in the frequency domain). The amount of shift in the frequency domain can be calculated using relations, given in section 5.2.12, containing the description of the LPPipFFT command, it also can be determined experimentally by visualizing the spectrum and shifting it to position one of its side lobes at the origin. This can be done with the LPInterpol command. Visualization of the spectrum side lobes requires a high gamma of the (bitmap) picture (of the order of 10).

7 Command reference

The following commands are available in LightPipes for Matlab. All the routines have a 'LightPipes Field' as output and one or more 'LightPipes Field 's' as input. A 'LightPipes Field' is a complex square array containing the complex field amplitudes and an extra row containing information about the field (wavelength, array dimension, etc.). Exceptions are the commands: **LPNormal, LPStrehl, LPIntensity, LPPhase, and LPPhaseUnwrap** (see the command-descriptions for details).

LPAxicon(phi,n1,x_shift,y_shift,Field)
propagates the field through an axicon.
phi: top angle in radians;
n1: refractive index (must be real);
x_shift, y_shift: shift from the centre.

LPBeamMix(Field1,Field2)
Addition of Field1 and Field2. If the wavelengths are different, a new, average wavelength, will be calculated.

LPBegin(grid_size,lambda,grid_dimension)
initialises a uniform field with unity amplitude and zero phase.
grid size: the size of the grid in units you choose,
lambda: the wavelength of the field with the same units as the grid size,
grid dimension: the dimension of the grid. This must be an even number larger than 8.

LPCircAperture(R,x_shift,y_shift,Field)
circular aperture.
R=radius,
x_shift, y_shift = shift from centre.

LPCircScreen(R,x_shift,y_shift,Field)
circular screen.
R=radius,
x_shift, y_shift = shift from centre.

LPConvert(Field)
converts the field from a spherical grid to a normal grid.

LPForvard(z,Field)
propagates the field a distance z using FFT.

LPForward(z,new_size,new_number,Field)
propagates the field a distance z, using direct integration of the Kirchoff-Fresnel integral.

LPFresnel(z,Field)

propagates the field a distance z. A convolution method has been used to calculate the diffraction integral

LPGain(Isat,gain,length,Field)
Laser saturable gain sheet.
Isat=saturation intensity,
gain= small signal gain,
length=length of gain medium

LPGaussAperture(R,x_shift,y_shift,T,Field)
Gauss aperture.
R=radius,
x_shift, y_shift = shift from centre;
T= centre transmission

LPGaussHermite(n,m,A,w0,Field)
substitutes a TEMm,n mode with waist w0 and amplitude A into the field.

LPGaussScreen(R,x_shift,y_shift,T,Field)
Gauss screen.
R=radius,
x_shift, y_shift = shift from centre;
T= centre transmission

LPIntAttenuator(Att,Field)
Intensity Attenuator. Attenuates the field-intensity by a factor Att

LPIntensity(flag,Field)
calculates the intensity of the field. Output is a square array containing the (scaled) intensity distribution.
flag=0: no scaling;
flag=1: normalisation of the intensity;
flag=2: bitmap with gray scales

LPInterpol(new_size,new_n,xs,ys,phi,magn,Field)
interpolates the field to a new grid.
new_size: new grid size;
new_n: new grid dimension;
xs, ys : the shifts of the field in the new grid;
phi: angle of rotation (after the shifts);
magn: magnification (last applied)

LPLens(f,x_shift,y_shift,Field)
propagates the field through a lens.
f: focal length;
x_shift, y_shift: transverse shifts of the lens optical axis

LPLensForvard(f,z,Field)
propagates the field a distance z using a variable coordinate system.
f: focal length of the input lens:
z: distance to propagate

LPLensFresnel(f,z,Field)
propagates the field a distance z using a variable coordinate system.
f: focal length of the input lens;
z: distance to propagate

LPMultIntensity(Intensity,Field)
multiplies the field with an intensity profile stored in the array: Intensity

LPMultPhase(Phase,Field)
multiplies the field with a phase profile stored in the array: Phase

[FieldOut,NC]=LPNormal(Field)
normalizes the field. The normalization coefficient is stored in NC

LPPhase(Field)
calculates the phase of the field. Output is a square array containing the phase distribution of the field.

LPPhaseUnwrap(Phase)
unwraps phase (remove multiples of π). Input and Output are arrays contaning the (unwrapped) phase distribution.

LPPipFFT(Direction,Field)
Performs a Fourier transform to the Field.
Direction = 1: Forward transform;
Direction = -1: Inverse transform

LPPolarizer(Phi,Fx,Fy)
Polarizer. Ouputs a linear polarized field.
Phi=polarizer angle,
Fx, Fy=input fields of horizontal and vertical components.

LPRandomIntensity(seed,Field)
Random intensity mask.
'seed' is an arbitrary number to initiate the random number generator

LPRandomPhase(seed,max,Field)
Random phase mask.
'seed' is an arbitrary number to initiate the random number generator;
'max' is the maximum value of the phase.

LPRectAperture(sx,sy,x_shift,y_shift,phi,Field)
rectangular aperture.
sx, sy: dimensions of the aperture;
x_shift, y_shift: shift from centre;
phi: rotation angle.

LPRectScreen(sx,sy,x_shift,y_shift,phi,Field)
rectangular screen.
sx, sy: dimensions of the screen;
x_shift, y_shift: shift from centre;
phi: rotation angle.

LPReflectMultiLayer(P,N0,Ns,N,d,Th,Field)
MultiLayer Reflector. Reflection of the field by a multi-layer mirror.
P=0: s-Polarization; P=1: p-Polarization;
N0=refractive index of incident medium (must be real);
Ns=refractive index of substrate;
N= array with (complex) index of the layers;
d= array with the geometrical thicknesses of the layers;
Th= angle of incidence at first layer;
Field= the input field

LPSteps(z,N_steps,Refract,Field)
Propagates 'Field' a distance z in 'N_steps' steps in a medium with a complex refractive index stored in the NxN matrix 'Refract'. 'N' is the grid dimension.

SR=LPStrehl(Field)
calculates the Strehl ratio. The Strehl ratio is stored in: SR

LPSubIntensity(intensity,Field)
substitutes an intensity profile into the field. The profile must be stored in the square array: intensity

LPSubPhase(phase,Field)
substitutes a phase profile into the field.The phase profile must be stored in the square array: phase.

LPTilt(tx,ty,Field)
introduces tilt into the field distribution. tx, ty = tilt components in radians.

LPTransmitMultiLayer(P,N0,Ns,N,d,Th,Field)
MultiLayer Transmission. Transmission of the field through a multi-layer coated substrate.
P=0: s-Polarization; P=1: p-Polarization;
N0=refractive index of incident medium (must be real);

Ns=refractive index of substrate;
N= array with (complex) index of the layers;
d= array with the geometrical thicknesses of the layers;
Th= angle of incidence at first layer;
Field= the input field

LPZernike(n,m,R,A,Field)
Introduces arbitrary Zernike aberration into the field.
n and m are the integer orders, (See Born and Wolf, 6th edition p.465, Pergamon 1993).
R is the radius at which the phase amplitude reaches A (in radians)

8 References.

1 J.W. Goodman, Introduction to Fourier Optics, 49-56, McGraw-Hill (1968)
2 W.H. Southwell, J. Opt. Soc. Am., 71, 7-14 (1981)
3 A.E. Siegman, E.A. Sziklas, Mode calculations in unstable resonators with flowing saturable gain: 2. Fast Fourier transform method, Applied Optics 14, 1874-1889 (1975)
4 N.N. Elkin, A.P. Napartovich, in "Applied optics of Lasers", Moscow TsniiAtomInform, 66 (1989) (in Russian)
5 R. Bacarat, in "The Computer in Optical Research", ed. B.R. Frieden, Springer Verlag, 72-75 (1980)
6 A.A. Samarskii, E.S. Nikolaev, Numerical Methods for Grid Equations, V.1, Direct methods, pp. 61-65, Birkhäuser Verlag 1989.
7 M. Born, E. Wolf, Principles of Optics, Pergamon, 464, 767-772 (1993)
8 D. Malacara, Optical shop testing, J. Wiley & Sons, (1992)
9 D. Marcuse, Light Transmission Optics, Van Nostrand Reinhold, 267-280, 1972.