1999

Design of a diffractive optical element and evaluation of its performance for vision system

Michael Teissier
Lehigh University

Teissier, Michael

Design of a Diffractive Optical Element and Evaluation of its Performance for Vision System

January 2000

DESIGN OF A DIFFRACTIVE OPTICAL ELEMENT AND EVALUATION OF ITS PERFORMANCE FOR VISION SYSTEM

by

Michael Teissier

A Thesis

Presented to the Graduate and Research Committee

of Lehigh University

in Candidacy for the Degree of

Master of Science

in

Electrical Engineering

Lehigh University

November 1999

This thesis is accepted in partial fulfillment of the requirements for the degree of Master of Science.

11/17/99
(Date)

Prof. McAulay
(Thesis Advisor)

Prof.
(Chairman of Department)

Acknowledgments

I would like to express my deep thanks to Dr.Alastair McAulay for his guidance during my thesis research. I would also like to thank Alexandra Grandpierre for her help. Special thanks go to my parents for their support during my graduate studies.

.....

Table of Contents

List of Figures

Abstract

A diffractive optical element in the shape of a zone plate is considered for directing light from part of an image to selected cells in a photodetector array for a vision application. A single cell diffractive optical element is investigated analytically and numerically both on axis and off-axis using a computer algebra program. The results are compared with a lens for efficiency. A Gerchberg Saxton algorithm was implemented in C++ and used to demonstrate the computation of phase for a diffractive element in order to give a specified intensity output.

Chapter 1

Introduction

Nowadays, the examination of a scene on a screen is going faster and faster for defense purpose. Although some vision systems use electronic devices to read the whole image, the diffractive optical element allows faster reading and processing video data than conventional cameras because only a fraction of the image is needed. The DOE consists essentially of a microstructure with local variation of grating period and profile and it is constructed by using zone plates. Consequently, one can obtain a focus of the light off-axis. All the equations required are derived for the zone plates and are used to find the output intensity.

One cell for a diffractive optical element is investigated numerically and analytically in chapter 2 for the on-axis and off-axis cases. A lens is analyzed in chapter 3. The efficiency of the diffractive optical element and the lens are computed and

compared in chapter 4. In Chapter 5 a Gerchberg Saxton algorithm is described and implemented in C++ for designing a diffractive optical element to generate a specified output intensity.

Chapter 2

One cell analysis for the diffractive optical element

2.1 General equations

To find the intensity of the field at the output plane, we have to use Fresnel diffraction. It can be written as,[6],[1],

$$g(x_0, y_0, z_0) = \frac{-j}{\lambda z_0} \int_x \int_y f(x, y) \exp\left\{ j \frac{k}{2z_0} \left[(x - x_0)^2 + (y - y_0)^2 \right] \right\} dxdy \qquad (2.1)$$

For circular symmetry

$$\rho = \sqrt{x^2 + y^2}$$

So equation (2.1) is reduced to

$$g(\rho_0, z_0) = \frac{-j2\pi}{\lambda z_0} e^{jkr_0} \int_0^a \rho f(\rho) e^{j\frac{k\rho^2}{2z_0}} J_o(\frac{k\rho\rho_0}{z_0}) d\rho \tag{2.2}$$

where

$$r_0 = z_0 + \frac{(x_0^2 + y_0^2)}{2z_0} \tag{2.3}$$

The transfer function for the zone plate is

$$T(\rho) = U\left[\cos\left(\frac{\rho^2}{2\sigma^2}\right)\right] \tag{2.4}$$

where

$$U(x) = 1 \text{ if x>=0}$$

$$0 \text{ if x } < 0$$

So, if we insert equation (2.4) into (2.2) in place of $f(\rho)$ and change variables using $w = \frac{k\rho^2}{2z_0}$, the result appears in equation (2.5).

$$g(\rho_0, z_0) = -je^{jkr_0} \int_0^{\frac{ka^2}{2z_0}} e^{jw} U\left[\cos\left(\frac{wz_0}{f}\right)\right] J_0\left[\rho_0\left(\frac{2kw}{z_0}\right)^{\frac{1}{2}}\right] dw \tag{2.5}$$

For n rings, equation (2.5) rmcomes

$$g(\rho_0, z_0) = -je^{jkr_0}[\int_0^{\frac{\pi}{2}} e^{jw} J_0\left[\rho_0\left(\frac{2kw}{z_0}\right)^{\frac{1}{2}}\right] dw + \int_{\frac{3\pi}{2}}^{\frac{5\pi}{2}} e^{jw} J_0\left[\rho_0\left(\frac{2kw}{z_0}\right)^{\frac{1}{2}}\right] dw + \ldots \tag{2.6}$$

$$\int_{\pi(n-3)/2}^{\pi(n-1)/2} e^{jw} J_0\left[\rho_0\left(\frac{2kw}{z_0}\right)^{\frac{1}{2}}\right]] dw$$

2.2 Equation and shape for on axis zone plate output intensity.

2.2.1 Numerically

If we take the series approximation of the Bessel function around the origin, we have:

$$BesselJ(0, a\sqrt{w}) = 1 - \frac{1}{4}a^2w + \frac{1}{64}a^4w^2 - \frac{1}{2304}a^6w^3 + ... \tag{2.7}$$

where

$$a = \rho_0 \left(\frac{4\pi}{\lambda f}\right)^{\frac{1}{2}} \tag{2.8}$$

where λ is the wavelength, f the focal distance.

Substituting (2.7) into (2.6) and using the numbers below, in two dimensions the shape is shown in figure 2.1

In three dimensions, it is shown in figure 2.2.

The parameters used were:

The number of ring equals 13 (n=13)

The DOE diameter equals $3.6e^{-3}$ m (a=b= $3.6e^{-3}$ m)

the wavelength equals 10^{-6} m(λ =10^{-6} m)

the focal length equals 1 meter (f_c = 1 m).

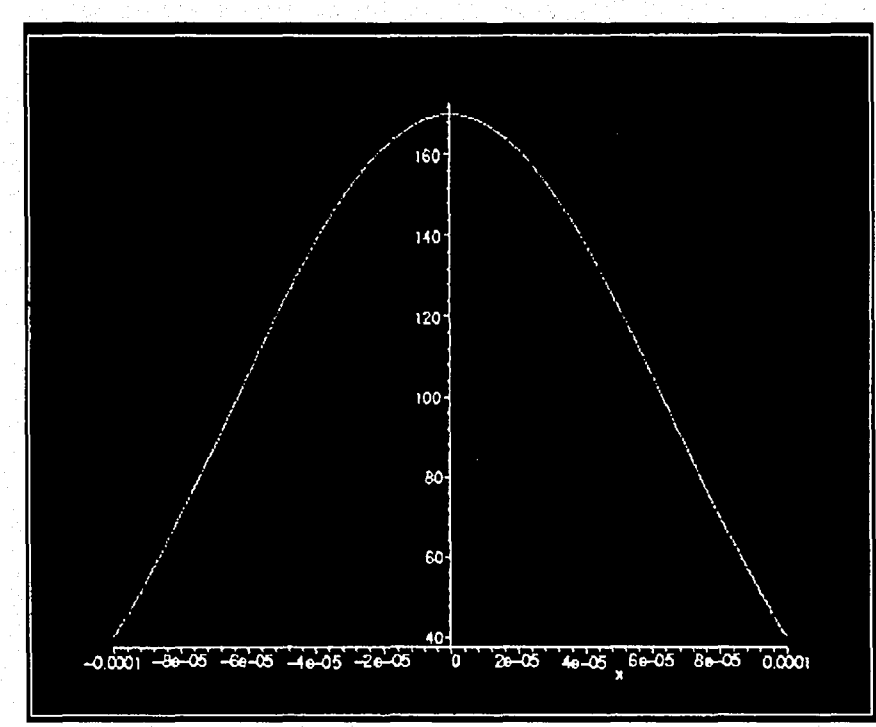

Figure 2.1: The diffractive optical element intensity

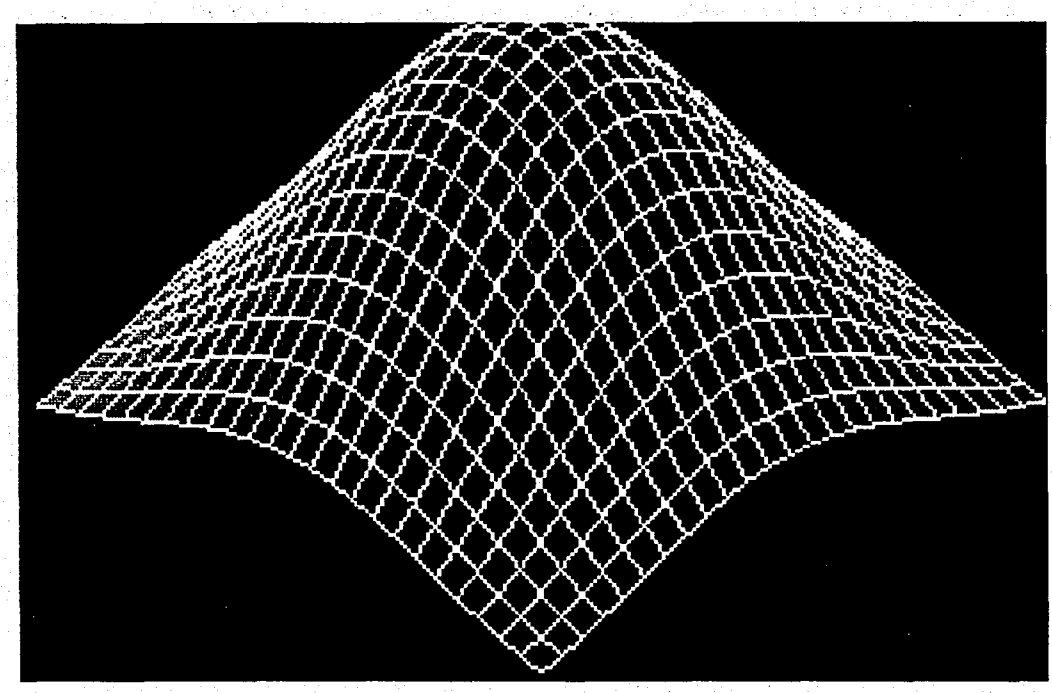

Figure 2.2: The DOE intensity in 3D

Graphically, the intensity at x=y=0 is approximatively 170 times the input intensity.

2.2.2 Analytically

The intensity for the DOE at the origin can be written (appendix B)

$$I_{DOE} = \left(\frac{a^2}{\lambda f}\right)^2 \tag{2.9}$$

With the parameters above, equation (2.6) is 169 times the input intensity which agrees with the numerica 1computation in section 2.2.1.

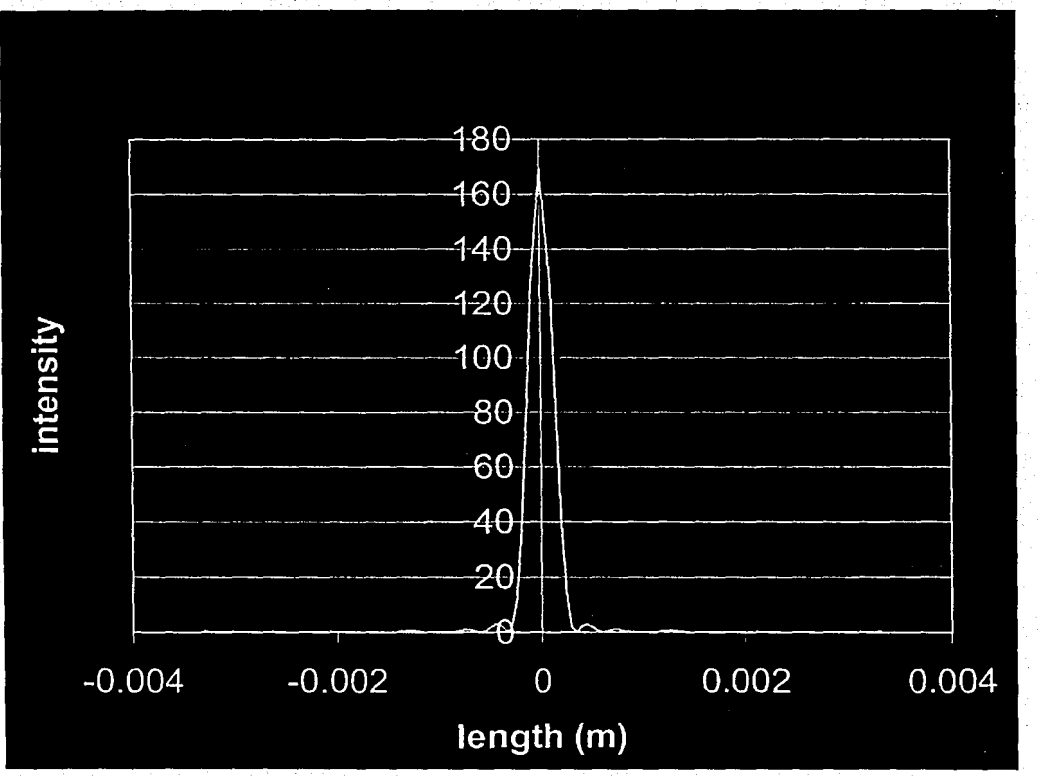

Figure 2.3: discontinuous plot

2.3 Shape for off axis zone plate.

This is not very easy to plot the shape using equation (2.6). For this purpose, we have to use a C++ program in order to compute the Bessel function for all points wanted and then plot the result. The program appears in appendix C. The result is shown figure 2.3.

Chapter 3

Analysis for the lens

3.1 At the origin

For the lens, [7], the field across the focal plane ($z_0 = f$) can be written as (see appendix A)

$$g(\rho_0, z_0) = -jbe^{jkr_0}\frac{J_1\left(\frac{bk\rho_0}{f}\right)}{\rho_0} \tag{3.1}$$

where J_1 is the first order Bessel function, f is the focal length, $k=\frac{2\pi}{\lambda}$, and b is the radius of the plane. r_0 is defined in equation (2.3).

At the origin (see appendix D), the intensity is

$$I_{lens} = \left(\frac{\pi b^2}{\lambda f}\right)^2 \tag{3.2}$$

The shape is shown in figure 3.1.

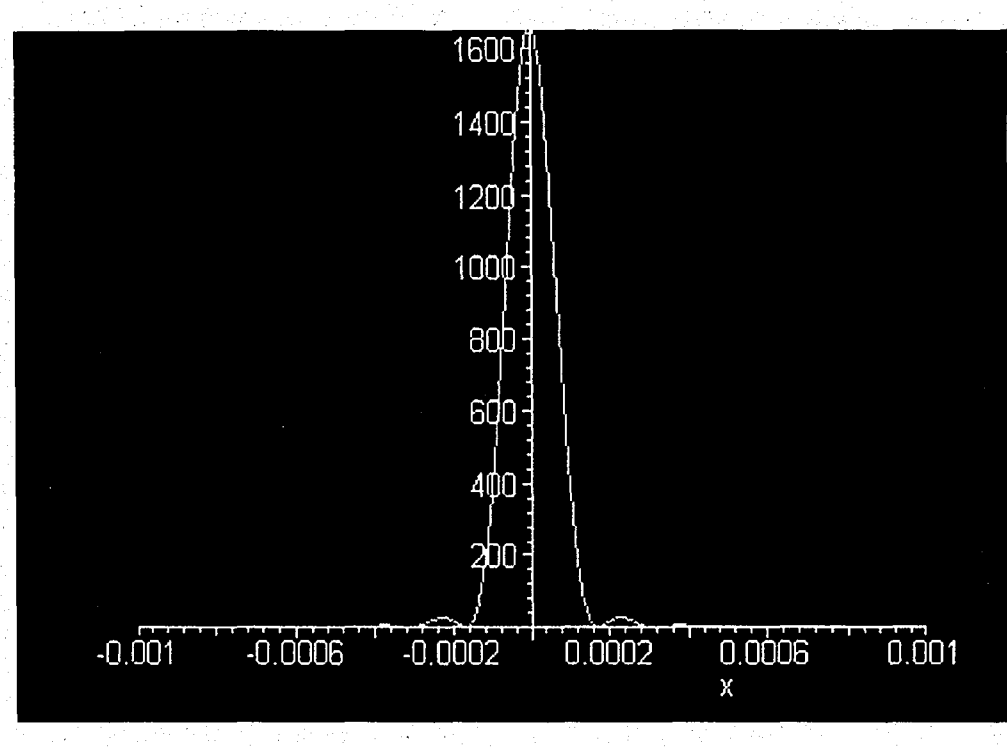

Figure 3.1: The lens intensity

The intensity at the origin is π^2 times larger than the DOE which agrees with the numerical computation. Using (3.2) and (2.9), and set a=b, one has

$$\frac{I_{lens}}{I_{DOE}} = \frac{\left(\frac{\pi b^2}{\lambda f}\right)^2}{\left(\frac{a^2}{\lambda f}\right)^2} = \pi^2$$

Chapter 4

The efficiency

In most DOEs, only the first diffraction order contains the desired light distribution. All others are unwanted. Thus the ratio between the useful and the incoming light intensity is called the efficiency. It can be written[2],

$$E = \frac{\int\int_{q} I(x,y)dxdy}{\int\int_{f} |W(u,v)|^2 dudv} \tag{4.1}$$

where I(x,y) is the intensity distribution in the focal plane and W(u,v) represents the laser beam complex amplitude. "q" is the focal domain and f is the aperture. For our example, the aperture is a 0.16*0.16 meter square and the focal plane is a 0.0002*0.0002 meter square.

If we assume $|W(u,v)| = 1$, equation (4.1) becomes

$$E = \frac{\int\int_{q} I(x,y)dxdy}{\int\int_{f} dudv} \tag{4.2}$$

4.1 The diffractive optical element

Using the relation above, we find with the first seven terms of equation (2.6):

$$E_{DOE} = 4.76\%$$

We can increase the efficiency by increasing n in equation (2.6), decrease the DOE aperture, Increase the focal plane,[3].

4.2 The lens

For the same values

$$E_{Lens} = 46.2\%$$

This example shows the ratio between the DOE efficiency and the lens equals π^2.

The computation is in appendix E.

Figure 4.1 shows the two shapes together.

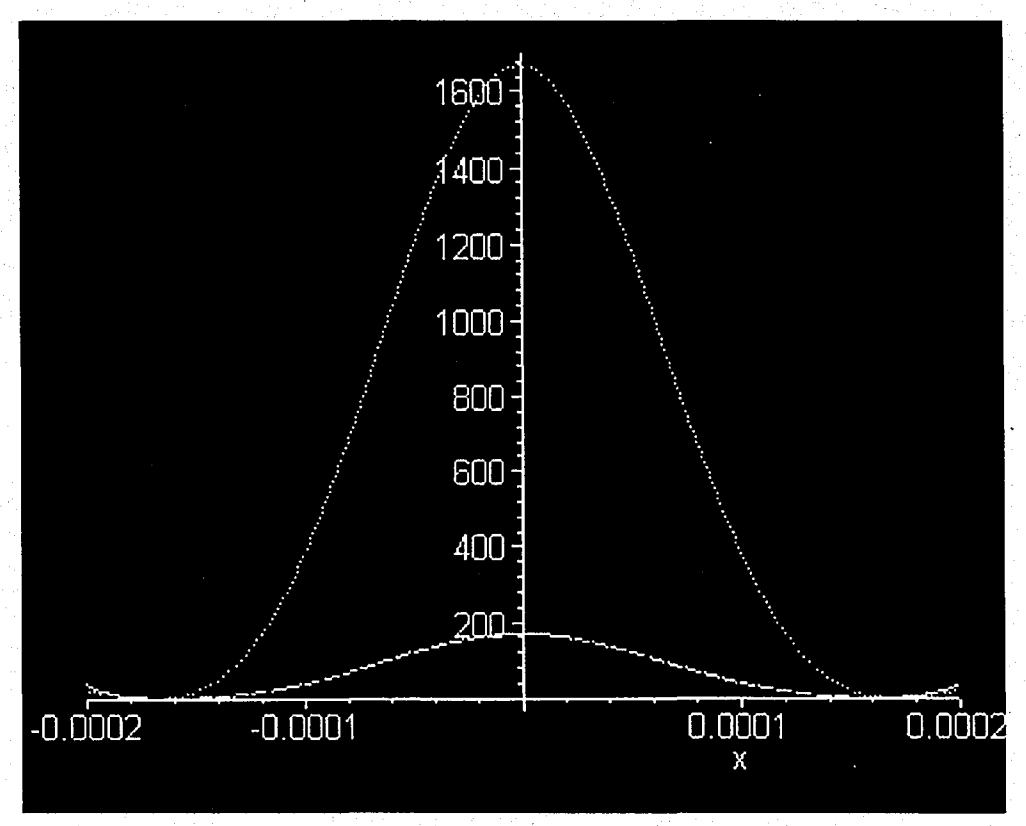

Figure 4.1: The efficiency

Chapter 5

The Gerchberg-Saxton algorithm

A C++ computer program was written to implement the Gerchberg Saxton algorithm and is listed in appendix F.

A block diagram for the Gerchbmrg-Saxton algorithm is shown in figure 5.1.

The algorithm is supposed to find the best phase distribution at the input C for constant magnitude given the desired magnitude at output B..

The program shown in appendix F allows us to start with a n x n complex matrix and it performs loops until the RMS error at point A doesn't change. I entered the following 8x8 matrix at point B in figure 5.1. The shape of the magnitude of the matrix appears in figure 5.4.

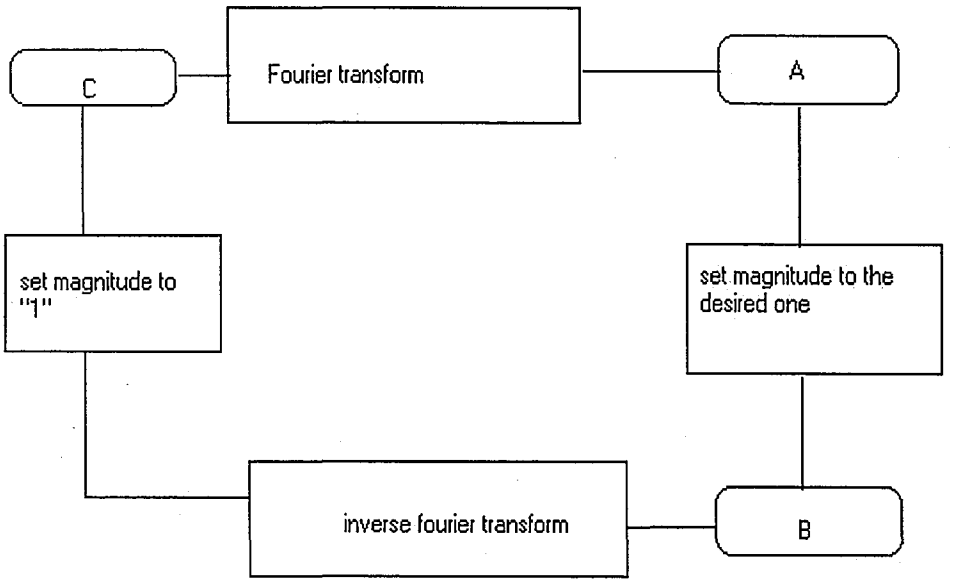

Figure 5.1: Block diagram of the Gerchberg Saxton algorithm

(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)
(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)
(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)
(0,0)	(0,0)	(0,0)	(1,0.01)	(1,0.01)	(0,0)	(0,0)	(0,0)
(0,0)	(0,0)	(0,0)	(1,0.01)	(1,0.01)	(0,0)	(0,0)	(0,0)
(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)
(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)
(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)

The RMS values at point A is shown in figure 5.2.

After 18 loops, we see at point A the Shape in figure 5.3. It looks like the shape at point B with some scaling.

As the matrix at point B is not symmetric, the phase has not a $\frac{\sin x}{x}$ shape. The matrix symmetry appears to be along the fifth column because the matrix wraps around due to the fourier transform. With a non symmetry, the fourier tranform of this matrix is multiplied by an exponential (cos+j sin) and since the result of the multiplication is the same for cos and sin, there is a 45 degree symmetry. We can see that in figures 5.5 to 5.12 but it is easier if we look at the output phase matrix at point C.

$$
\mathrm{C}=\begin{array}{cccccccc}
-0.02 & -2.70 & 0.91 & -1.67 & 2.36 & 1.25 & -1.04 & 2.63 \\
-2.70 & 0.74 & -2.10 & 1.26 & -2.10 & 0 & 3.05 & 0.14 \\
0.91 & -2.10 & 1.13 & -1.98 & 1.05 & -2.25 & 0.79 & -2.32 \\
-1.67 & 1.26 & -1.98 & 1.19 & -1.83 & 1.44 & -1.59 & 1.57 \\
2.36 & -2.10 & 1.05 & -1.83 & 1.61 & -1.21 & 2.23 & -0.66 \\
1.25 & 0 & -2.25 & 1.44 & -1.21 & 2.40 & -0.24 & -2.81 \\
-1.04 & 3.05 & 0.79 & -1.59 & 2.23 & -0.24 & -2.69 & 1.21 \\
2.63 & 0.14 & -2.32 & 1.57 & -0.66 & -2.81 & 1.21 & -1.17
\end{array}
$$

and $\mathrm{C}^T = C$.

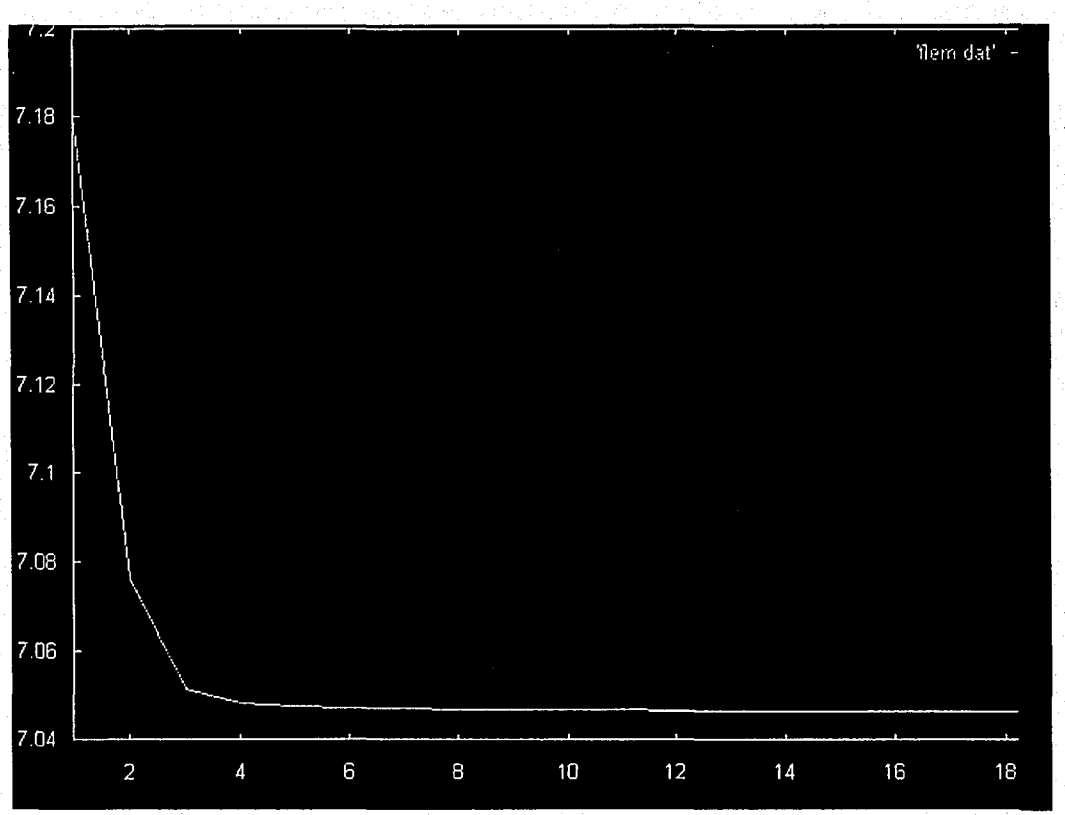

Figure 5.2: The RMS

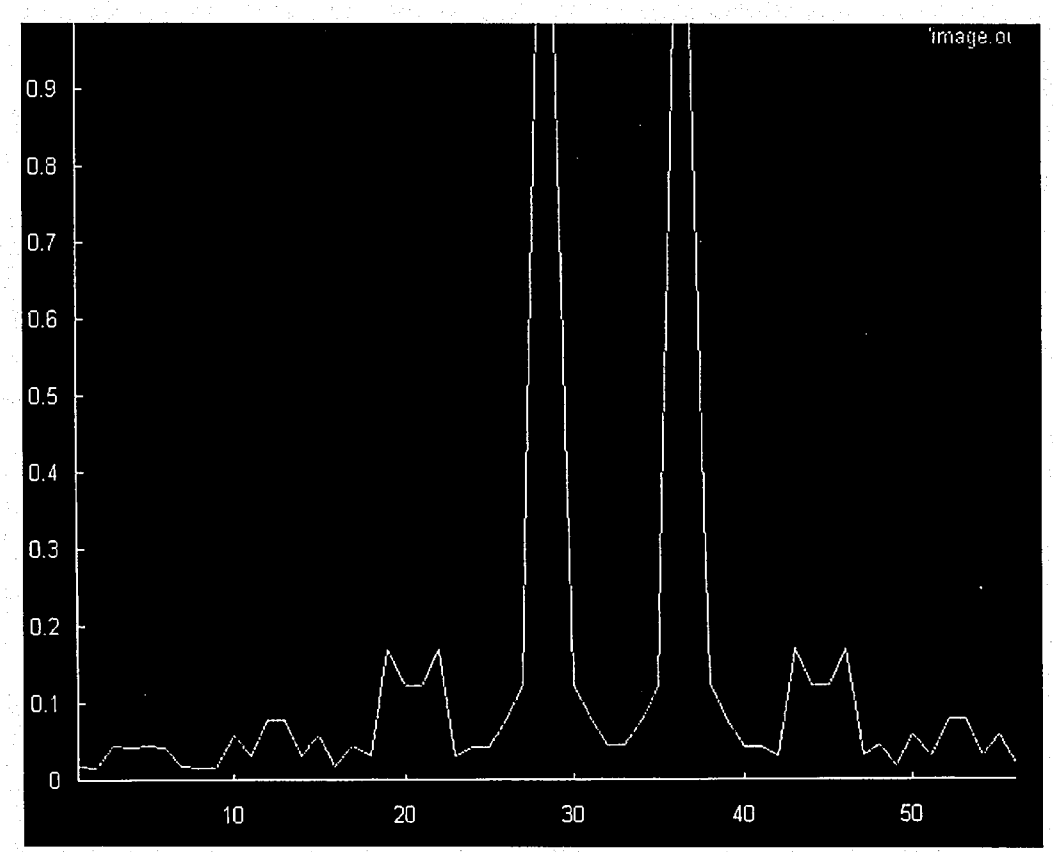

Figure 5.3: The amplitude for a 8x8 matrix at point A

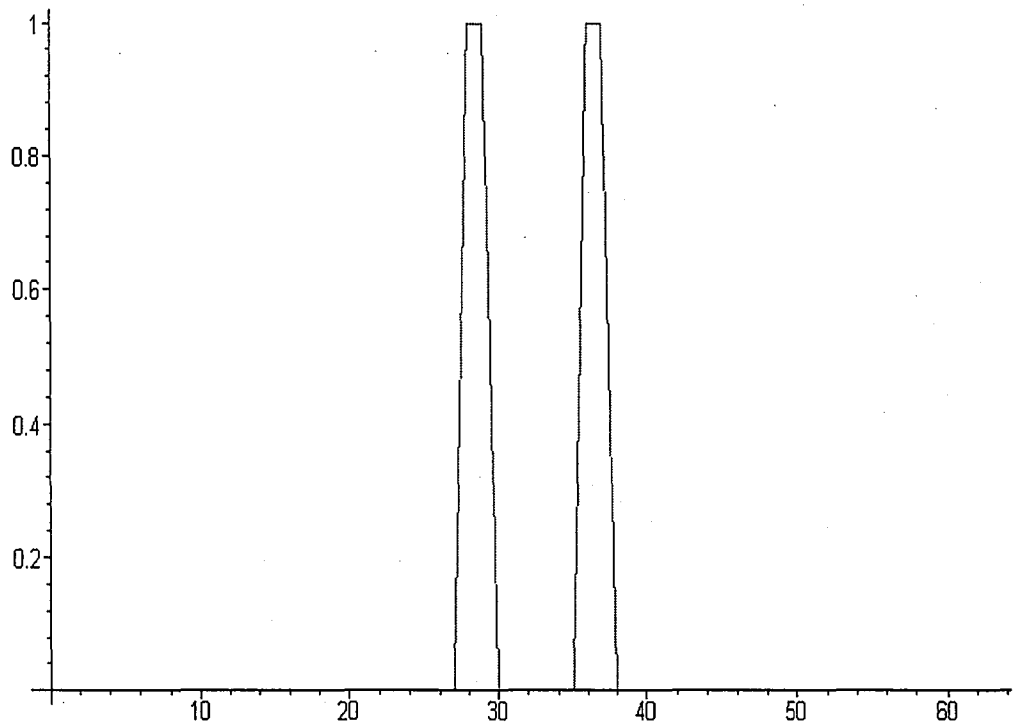

Figure 5.4: The amplitude for a 8x8 matrix at point B.

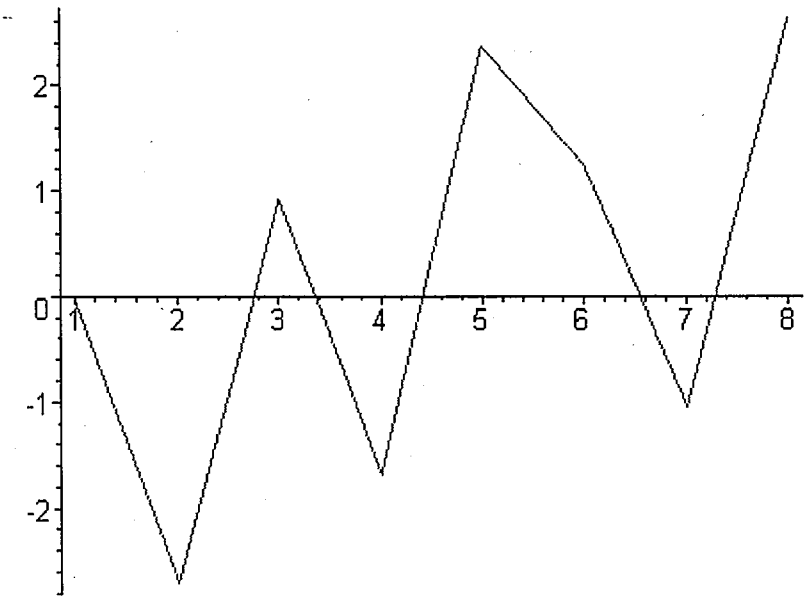

Figure 5.5: phase for the 1st row

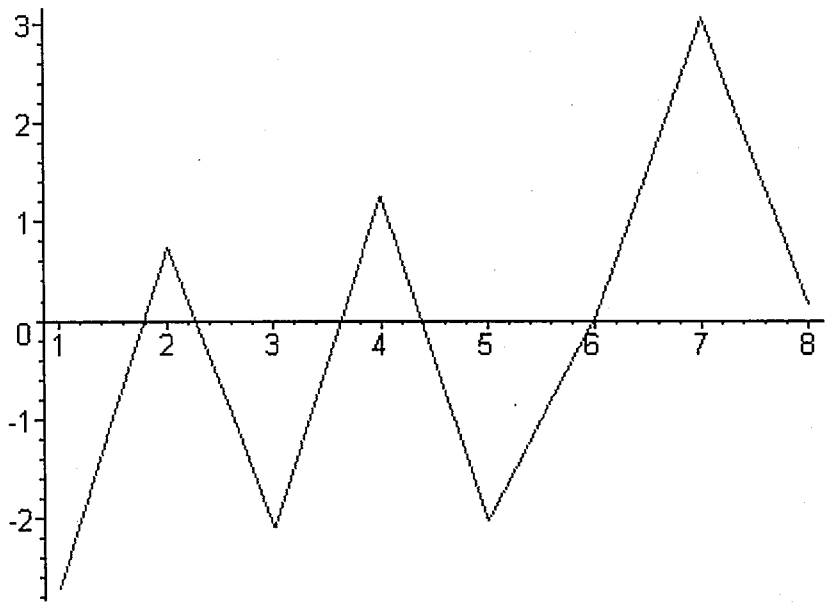

Figure 5.6: phase for the 2nd row

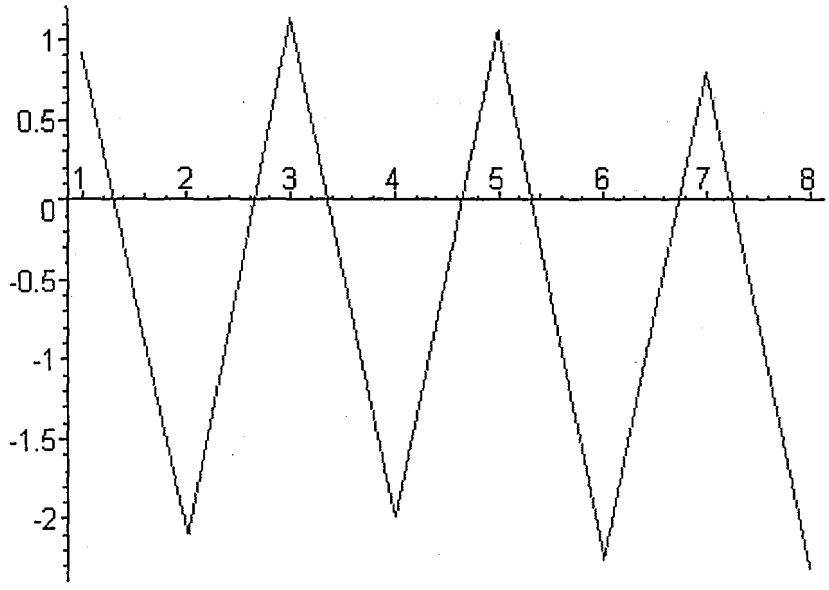

Figure 5.7: phase for the 3rd row

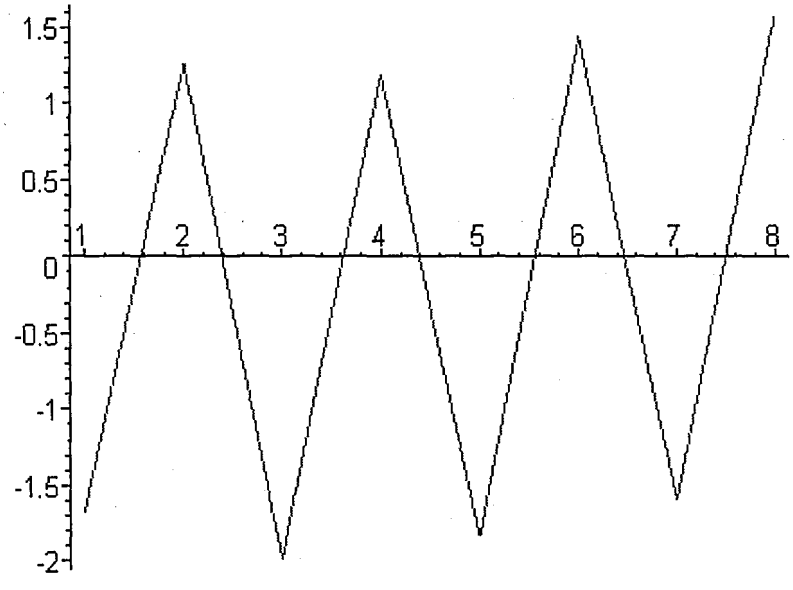

Figure 5.8: phase for the 4th row

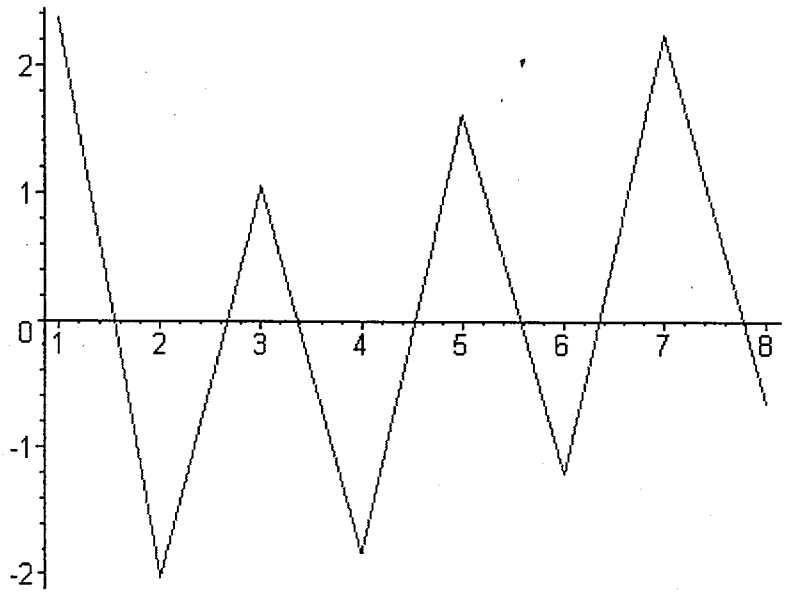

Figure 5.9: phase for the 5th row

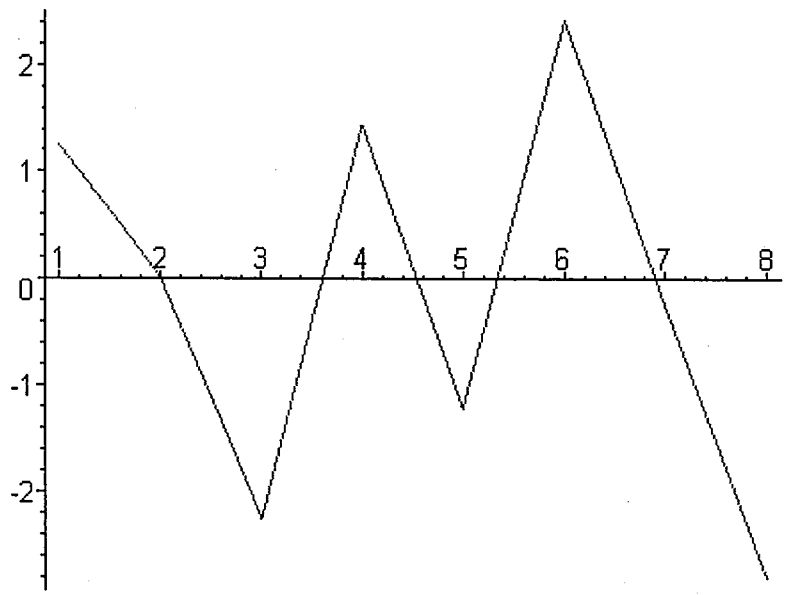

Figure 5.10: phase for the 6th row

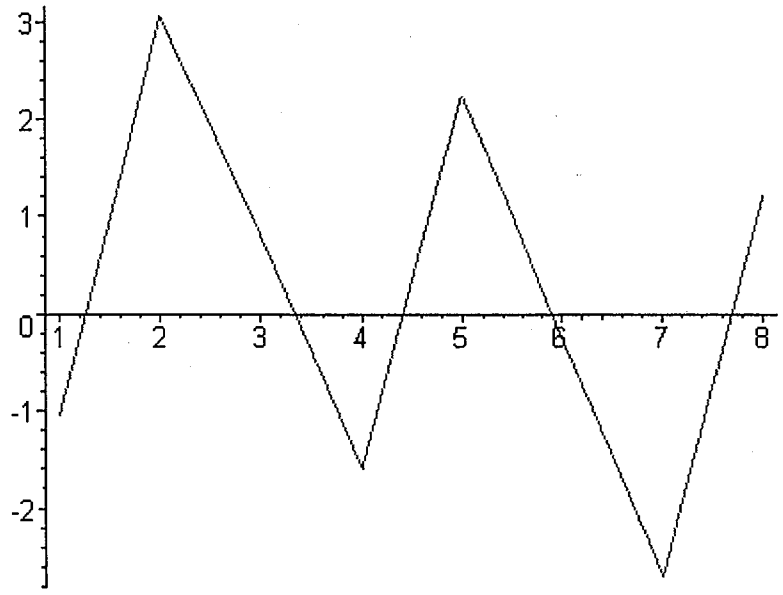

Figure 5.11: phase for the 7th row

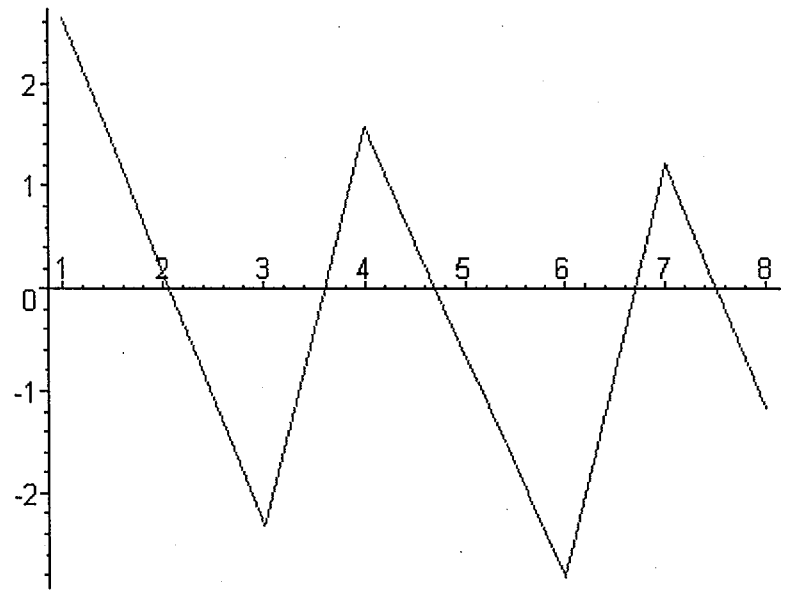

Figure 5.12: phase for the 8th row

Chapter 6

Conclusion

Analytic and numerical investigation of a lens and of a diffractive optical element in the shape of a zone plate showed that both will focus light to a small region in a similar manner. A computer algebra program was used in the analysis. A study of efficiency shows that the zone plate is less efficient but allows off axis control more readily than with a lens. For example, with a lens used for off axis focussing the spot in the output is elongated in one direction. Crosstalk between cells can be reduced for a diffractive element relative to an array of lenses because the regions between the lenses are properly accounted for. One method of designing diffractive elements with multiple zone plates is by means of the Gerchberg Saxton algorithm. A Gerchberg Saxton algorithm was implemented in C++ for computing phase for a diffractive element to generate a specific intensity output. The algorithm is used

to demonstrate the computation of phase for a given output intensity.

Bibliography

[1] McAulay Alastair D. "Diffractive optical element for multiresolution preprocessing for computer vision", SPIE, vol 3720, 1999.

[2] Doskolovich, L.L. " A method for estimating the DOE's energy efficiency" Optics and laser technology, vol 27, No 4, 1995, pps 219-221.

[3] Streib Norbert. " diffractive optical elements for optoelectronic interconnection" SPIE, vol 1574, pps 34-47.

[4] Hutley, M.C. Diffraction grating, 1982.

[5] Walter, A. The ray and wave theory of lenses, 1995, pps 364-375.

[6] Goodman Joseph W. Introduction to Fourier optics, 2nd edition, McGraw-hill, New York, 1996.

[7] Nishihara, H, "Micro Fresnel lenses", progress in optics, vol XXIV, pps 3-10.

Appendix A: Proof of expression equation 2.1

At first, the Fresnel Diffraction can be written as

$$g(x_0, y_0, z_0) = -\frac{j}{\lambda z_0} e^{jkz_0} \int_x \int_y f(x,y) \exp\left[j\frac{k}{2z_0}\left[(x-x_0)^2 + (y-y_0)^2\right]\right] dxdy \quad (1)$$

In circular, we have $\rho = \sqrt{x^2+y^2}$ so much so that the equation above gives:

$$g(\rho_0, z_0) = -\frac{j2\pi}{\lambda z_0} e^{jkr_0} \int_0^a \rho f(\rho) e^{\frac{jk\rho^2}{2z_0}} J_0(\frac{k\rho\rho_0}{z_0}) d\rho \quad (2)$$

where $r_0 = z_0 + \frac{(x_0^2+y_0^2)}{2z_0}$.

For the lens we use

$$f(\rho) = e^{-\frac{jk\rho^2}{2f}} \quad (3)$$

to cancel the exponent on equation (A.1) and we assume that $z_0 = f$.

Thus we have to compute

$$\int_0^a \rho J_0(\frac{k\rho\rho_0}{z_0}) d\rho \quad (4)$$

let $\rho^{'} = \frac{k\rho\rho_0}{f}$ then $\rho = \frac{f\rho^{'}}{k\rho_0}$ and $d\rho = \frac{f}{k\rho_0}d\rho^{'}$

$\rho^{'}$ goes from 0 to $\frac{ka\rho_0}{f}$

then (A.3) becomes

$$\int_0^a \frac{f\rho^{'}}{k\rho_0} J_0(\rho^{'}) \frac{f}{k\rho_0} d\rho^{'}$$

or

$$\int_0^a \left(\frac{f}{k\rho_0}\right)^2 \rho^{'} J_0(\rho^{'}) d\rho^{'}$$

which equals

$$\left(\frac{f}{k\rho_0}\right)^2 \frac{ka\rho_0}{f} J_1\left(\frac{ka\rho_0}{f}\right) \tag{5}$$

Then (A.1) looks like:

$$g(\rho_0, z_0) = -\frac{jk}{f} e^{jkr_0} \left(\frac{f}{k\rho_0}\right)^2 \frac{ka\rho_0}{f} J_1\left(\frac{ka\rho_0}{f}\right) \tag{6}$$

(A.6) equals

$$-\frac{jk}{f} e^{jkr_0} \left(\frac{f}{k\rho_0}\right) a J_1\left(\frac{ka\rho_0}{f}\right) \tag{7}$$

$$= -je^{jkr_0} \frac{1}{\rho_0} a J_1\left(\frac{ka\rho_0}{f}\right) \tag{8}$$

Finally

$$g(\rho_0, z_0) = -je^{jkr_0} \frac{1}{\rho_0} a J_1\left(\frac{ka\rho_0}{f}\right) \tag{9}$$

So

$$I = g * g^* = a^2 \left(\frac{J_1\left(\frac{ka\rho_0}{f}\right)}{\rho_0}\right)^2 \tag{10}$$

Appendix B: Proof of equation 1.9

If we define a parameter for the zone plate:

$$\sigma^2 = \frac{f}{k} \tag{B.1}$$

As

$$k = \frac{2\pi}{\lambda} \tag{B.2}$$

then (B.1) becomes:

$$f = \frac{2\pi\sigma^2}{\lambda} \tag{B.3}$$

Using (B.3) and if a is the radius of the nth ring, then:

$$n = \frac{a^2}{\lambda f} \tag{B.4}$$

Using appendix A equation A.6 and substituting for n, the intensity at the focal point on axis is given by

$$I_{DOE} = (\frac{a^2}{\lambda f})^2 \tag{B.5}$$

Appendix C: C++ program for equation 1.6

This is the C++ program:

```
#include<iostream.h>
#include<math.h>
float besselj0(float x);
float ax,z;
double xx,y,ans,ansl,ans2;
if ((ax=fabs(x))<8.0){
y=x*x; ansl=57568490574+y*(-13362590354.0+y*(651619640.7+
y*(-11214424.18 +Y*(77392.33017+y*(-184.9052456)))));
ans2=57568490411.0+ Y*(1029532985.0+y*(9494680.718+
y*(59272.64853 +y*(267.8532712+y*1.0))));
ans=ansl/ans2;
```

```
}
else
{z=8.0/ax;
y=z*z;
xx=ax-0.785398164;
ans1 =1.0+Y* (-0. 1098628627e-2+y* (0.2734510407e-4+
y*(-0.2073370639e-5 +Y*0.209388721 le-6)));
ans2=-0. 156449995e-1+y* (0. 1430488765e-3+
y* (-0.691114765le-5 +Y* (0.762109516e-6-y*O. 934935152e-7))),
ans=sqrt(0.636619772/ax)*(cos(xx)*ansl-z*sin(xx)*ans2)return ans;
}
void main()
{
float h,w,u,a,s,z;
for(int i=0;i<10;i++)
{
w=17.28;
cout < < " enter the a value" < < endl;
cin>>a;
Z=0;
```

```
do {
u=1809.15*a*sqrt(w);
z=z+besselj0(u);
w=w+0.03;
}
while(w<=20.42);
h=z/51; cout,
<<"the result is : "<<h<<endl;
s=h*h, cout,<<"the square is : "<<s<<endl;
}
```

Appendix D: Equations for intensity of focus for a lens

We now know from appendix A that the equation for the lens is given by:

$$g(\rho_0, z_0) = -jbe^{jkr_0}\frac{J_1(\frac{bk\rho_0}{f})}{\rho_0} \tag{D.1}$$

If one takes the series approximation for $J_1(\frac{bk\rho_0}{f})$ around 0 and divides by ρ_0, one finds, taking $c = \frac{bk\rho_0}{f}$,

$$y(c) = \frac{J_1(c)}{\rho_0} = \frac{1}{2}\frac{kb}{f} - \frac{1}{16}\frac{k^3b^3c^2}{f^3} + \frac{1}{384}\frac{k^5b^5c^4}{f^5} \tag{D.2}$$

So $y(0) = \frac{1}{2}\frac{kb}{f}$ where $k = \frac{2\pi}{\lambda}$.

So the intensity is

$$I_{lens} = (y(0))^2 * b^2 = \left(\frac{\pi b^2}{\lambda f}\right)^2 \tag{D.3}$$

Appendix E: Computer algebra program for computing efficiency for a diffractive optical element.

For the DOE, the efficiency is below.

```
l:=int(int((13.+.2213898395e29*x^2*y^6+.3320847591e29*x^4*y^4
+.2213898395e29*x^6*y^2+.2407390642e16*x^2*y^2
-.6113131525e36*x^4*y^6-216399710.4*x^2-216399710.4*y^2
+.1203695321e16*x^4-.3335093312e22*x^6+.1203695321e16*y^4
-.3335093313e22*y^6+.5534745985e28*x^8+.5534745985e28*y^8
-.6113131525e35*x^10-.6113131525e35*y^10-.3056565762e36*x^2*y^8
-.6113131525e36*x^6*y^4-.3056565762e36*x^8*y^2-.1000527994e23*x^2*y^4
-.1000527994e23*x^4*y^2)^2+(1.+.4093439823e28*x^4*y^4
+.1770703465e15*x^2*y^2-.9057677452e35*x^6*y^4+.2728959882e28*x^2*y^6
```

-.4528838728e35*x^2*y^8+.2728959882e28*x^6*y^2-.9849308356e21*x^2*y^4

-.9849308356e21*x^4*y^2-.4528838728e35*x^8*y^2-.9057677452e35*x^4*y^6

-10637327.10*y^2+.8853517321e14*y^4+.6822399704e27*y^8-10637327.10*x^2

-.3283102785e21*x^6+.8853517321e14*x^4-.9057677452e34*x^10

+.6822399703e27*x^8-.9057677451e34*y^10-.3283102786e21*y^6)^2

,x=-0.0002..0.0002),y=-0.0002..0.0002);

l := .00001218417971

o:=int(int(1,x=-0.008..0.008),y=-0.008..0.008);

o := .0002560000000

i:=l/o;

i := .04759445199

For the lens here is the efficiency:

y=series(BesselJ(1,(k*b/f)*c),c);

y(c) =(1/2*k*b/f)+(-1/16*k^3*b^3/(f^3))*c^2+(1/384*k^5*b^5/(f^5))*c^4;

f(c):=(1/2*k*b/f-1/16*k^3*b^3*c^2/(f^3)+1/384*k^5*b^5*c^4/(f^5))^2;

c:=sqrt(x^2+y^2);

f(c):=(1/2*k*b/f-1/16*k^3*b^3*c^2/(f^3)+1/384*k^5*b^5*c^4/(f^5))^2;

b:=0.007; lambda:=0.0000000384; f:=100;

k:=2*Pi/lambda;

l:=int(int(b^2*(1/2*k*b/f-1/16*k^3*b^3*(x^2+y^2)/(f^3)

```
+1/384*k^5*b^5*(x^2+y^2)^2/(f^5))^2,x=-0.0002..0.0002),

y=-0.0002..0.0002);

l := .0001182618413

o:=int(int(1,x=-0.008..0.008),y=-0.008..0.008);

o := .0002560000000

i:=l/o;

i := .4619603176
```

Appendix F: C++ program for the Gerchberg Saxton algorithm

list of functions:

writedata();

hello();

error();

openfiles(inFile, outFile);

getinfo(loops, tolerance, z);

closefiles();

diffractive(A,b,re);

convert(A,c,b,b);

convertback(A,c,b,b);

expo(A,B,b);

exponentielle(D,Y,A,b);

```
ampl(A,T,b);

arg(A,E,b);

RMS(Y,A,b,re);

adjust(A,Y,b);

division(c,toi);

init(v,ndim);

fresprop(fldfres,ndim,mdim,delz,delx,dely);

writetest(fldfres,ndim,mdim,1);

writebeam(fldfres,ndim,mdim);

fourn(c,v,2,-1);

#include <iostream.h> // I/O

#include <fstream.h> // file I/O

#include <stdlib.h> // included for the exit(1) commands

# include <math.h> // included for the FFT procedure

# include <complex.h>

#include "wire.hpp"

//define any constants

//#define test

#define Pi 22/7 // define Pi I think that I am going to use it later

//#define height 4 // this is the # of elements in the width of the image
```

```
//#define width 4 // this is the # of elements in the height of the image
//#define elements height*width // the 2 is because we have mag and phase
#define SWAP(a,b) tempr=(a);(a)=(b);(b)=tempr // needed for the FFT pro-
cedure
void writedata(ofstream &outFile, float data[],int pdim){
int i;
// Place a Header on the output file here
// outFile<<"This is the output file from..... It contains"
// <<"\nall of the data that was listed in the file hw10.dat with "
// <<"\nextra information appended onto the end of it"
// <<"\n\n\n";
for(i=0; i<2*pdim; i++)
{
outFile<<data[i]<<"\n";
}
return;
};
/*****************************************************************/
/*****************************[hello]*****************************/
/*****************************************************************/
```

```
// Tell the user of this program what its purpose and the method it uses
// to satisfy its purpose.
void hello(void){
// info:
cout<<"This program's purpose is to develop the required filter"
<<"\nnessasary to image a picture to a screen where different"
<<"\nresolutions are directed toward different matrix lines in"
<<"\nthe picture. Image data is taken from the file 'image.dat"'
<<"\nmanipulated and the resulting filter properties are written"
<<"\nto the file 'image.out"'
<<"\n\n NOTE: for information on the required format of the data "
<<"\n located in the 'image.dat' file read the Readme.txt "
<<"\n File."
<<"\n\n(Press any key to continue)";
cin.get(); // hold the screen so the user can read it if they want.
return;
}
/*****************************************************************/
/*****************************[error]*****************************/
/*****************************************************************/
```

```
// this function is called when a file was unable to be opened or closed
// and it just basically tells the user that their was a I/O problem.
// It takes a string as an argument that allows the program to add extra
// description as to what the error is/was.
void error(char info[40]){
cout<<"\n\nFatal Error: Unable to "<<info<<".\n"
<<" Check to make sure the drive is not full.\n"
<<" Press Return to terminate this program ";
cin.get(); // hold the screen before the program ends
exit(1); // terminate the program
return;
}
/*************************************************************/
/*************************[openfiles]*************************/
/*************************************************************/
// Open up all files.
// if a file fails to open call error which will tell the user the
// problem and terminate the program.
void openfiles(ifstream &inputfile, ofstream &outputfile){
inputfile.open("image.dat"); // try to open hw8.dat
```

```
if(inputfile.fail()){ // see if it opened or failed
error("open the file image.dat"); // if it failed tell the user and kill
}
outputfile.open("image.out"); // try to open hw8.out
if(outputfile.fail()){ // see if it opened or failed
error("open the file image.out"); // if it failed tell the user and kill
}
return;
};
/*****************************************************************/
/***************************[getinfo]*****************************/
/*****************************************************************/
void getinfo(int &loops, double &tolerance, double &z){
char choice = 'a';
while((choice != 'n') && (choice != 'y') && (choice != 'N') && (choice != 'Y')){
cout<<"\nWould your like to enter your own parameters or use default ones"
<<"\n(enter a 'y' or 'n') := ";
cin>>choice;
}
if((choice == 'y') || (choice == 'Y')){
```

```
cout<<"\n\nEnter new value for the maxumum number of"
<<"\nloops (Default is 100) := ";
cin>>loops;
cout<<"\nEnter the nuber for value for the tolerance "
<<"\n(Default is .01) := ";
cin>>tolerance;
cout<<"\nEnter the distance between the input plane and the"
<<"\noutput plane (Default is .10 meters)";
cin>>z;
}
return;
};
/*****************************************************************/
/*************************[closefiles]*************************/
/*****************************************************************/
// if either of the two close's below fails then call file fail which
// will tell the user the problem and then terminate the program.
void closefiles(ifstream &inputfile, ofstream &outputfile){
outputfile.close(); // try to close the output file "HW10.out"
if(outputfile.fail()){ // check to see if the close was successful
```

```
error("close the file image.out"); // if it failed tell the user and kill
}
inputfile.close(); // try to close the inputfile "HW10.dat"
if(inputfile.fail()){ // check to see if the close was successful
error("close the file image.dat"); // if it failed tell the user and kill
}
return;
}
/*****************************************************************/
/*************************[diffractive]***********************/
/*****************************************************************/
void diffractive(complex A[][b],int pdim, int dim){
float c=0.0;
int random(int num);
float u,p=0.0;
for (int i=1;i<pdim;i++){
for (int j=1;j<pdim;j++){
cout<<"entrez la valeur de A"<<i<<j<<"\n";
cin>>c;
if(c!=0)
```

```
{
int t=random(1000);
float w=0.001*t;
complex f(c,0.01);
A[i][j]=f;
}
else
{
complex t(0,0);
A[i][j]=t;
}
}
}
for (int i=1;i<pdim;i++)
for (int j=1;j<pdim;j++)
p=p+abs(A[i][j])*abs(A[i][j]);
for (int i=1;i<pdim;i++)
for (int j=1;j<pdim;j++)
A[i][j]=A[i][j]*sqrt(dim/p);
return;
```

```
}
/*****************************************************/
/*************************[convert]*******************/
/*****************************************************/
void convert(complex array[][b],float data[],int n,int m)
{
float *ptdata;
ptdata=&data[1];
for(int i=1;i<n;i++)
for(int j=1;j<m;j++)
{
*ptdata++=real(array[i][j]);
// cout<<*(ptdata-1)<<"\n";
*ptdata++=imag(array[i][j]);
// cout<<*(ptdata-1)<<"\n";
}
}
/*****************************************************/
/*************************[convertback]****************/
/*****************************************************/
```

```
void convertback(complex array[][b],float data[],int n,int m)
{
float *ptdata;
float *ptdata1;
ptdata=&data[1];
for(int i=1;i<n;i++)
for(int j=1;j<m;j++)
{
ptdata1=ptdata+1;
complex temp(*ptdata++,*ptdata1);
array[i][j]=temp;
ptdata++;
// cout<<"("<<real(array[i][j])<<","<<imag(array[i][j])<<")\n";
}
}
/*******************************************************/
/*************************[expo]*************/
/*******************************************************/
void expo(complex h[][b], complex g[][b],int pdim)
{
```

```
for(int i=1;i<pdim;i++){
for(int j=1;j<pdim;j++)
{
complex d(0,0);
if(h[i][j]==d)
{
complex u(1,0);
g[i][j]=u;
}
else
{
complex s(real(h[i][j])/abs(h[i][j]),imag(h[i][j])/abs(h[i][j]));
g[i][j]=s;
}
}
}
return;
}
/*****************************************************************/
/************************[exponentielle]**************************/
```

```
/*************************************************************/
void exponentielle(complex e[][b],complex h[][b],complex g[][b],int pdim)
{
for(int i=1;i<pdim;i++)
{
for(int j=1;j<pdim;j++)
{
float p=abs(h[i][j])*real(g[i][j])/abs(g[i][j]);
float q=abs(h[i][j])*imag(g[i][j])/abs(g[i][j]);
complex s(p,q);
e[i][j]=s;
}
}
return;
}
/*****************************[amplitude]*********************/
void ampl(complex R[][b],float P[],int pdim)
{
float *ptb;
ptb=&P[1];
```

```
for(int i=1;i<pdim;i++)
for(int j=1;j<pdim;j++)
*ptb++=abs(R[i][j]);
return;
}
/*****************************[argument]***********************/
void arg(complex R[][b],float P[],int pdim)
{
float *ptb;
ptb=&P[1];
for(int i=1;i<pdim;i++)
for(int j=1;j<pdim;j++)
*ptb++=arg(R[i][j]);
return;
}
/************************[RMS]************************/
int RMS(complex Ad[][b],complex A[][b],int pdim, int dim)
{
//int k=0;
float p=0,r,u;
```

```
float f=0;
//p=p+((norm(Ad[0])-norm(A[0]))*(norm(Ad[0])-norm(A[0])));
//f=f+(norm(Ad[0])-norm(A[0]));
//r=sqrt(p);
//u=sqrt(f);
// cout<<"\nla valeur de r est \n"<<r;
//cout<<"\nla valeur de u est \n"<<u;
for(int i=1;i<pdim;i++)
for(int j=1;j<pdim;j++)
{
// k=k+1;
p=p+((abs(Ad[i][j])-abs(A[i][j]))*(abs(Ad[i][j])-abs(A[i][j])));
f=f+(abs(Ad[i][j])*(abs(Ad[i][j])));
}
r=sqrt(p/dim);
u=sqrt(f/dim);
cout<<"\nla valeur de u est \n"<<u;
cout<<"\nla valeur de r est \n"<<r;
if(sqrt(p/pdim)<0.1)
return 1;
```

```
else
return 0;
}
/**************************[adjust]**************************/
void adjust(complex Ad[][b],complex A[][b],int pdim){
for (int i=1;i<pdim;i++)
for (int j=1;j<pdim;j++)
A[i][j]=Ad[i][j];
return;
}
/**************************[division by 16]************/
void division(float p[],int pdim)
{
for (int i=1;i<pdim;i++)
p[i]=p[i]/(pdim-1);
return;
}
/**************************[initialize nn[dim]]************/
void init(unsigned long nn[], int dim)
{
```

```
nn[1]=dim;
nn[2]=dim;
return;
}
/*********************************************************/
/*************************[fresprop]*************/
/*********************************************************/
void fresprop(complex fldfres[][mdim],int ndim, int mdim,
double delz, double delfx, double delfy)
{
//computes transfer function matrix for Fresnel propagation
int n,m,nr,mr,ndimh=ndim/2,mdimh=mdim/2;
//ndimh1=ndimh+1,mdimh1=mdimh+1;
double phfac = PI*lambda*delz;
complex exph;
//avoid center lines in n and m and 0 edges in n and m
for(n=1;n<ndimh;n++) {
nr = ndim-n;
for(m=1;m<mdimh;m++){
mr = mdim-m;
```

```
// cout<<"\n phfac= \n"<<phfac;
// cin.get();
complex phase(0.0,-phfac*(pow(n*distx/ndim,2)+ pow(m*disty/mdim,2)));
// cout<<"\n phase= \n"<<phase;
//cin.get();
complex s1(cos(-phfac*(pow(n*distx/ndim,2)+ pow(m*disty/mdim,2)))
,sin(-phfac*(pow(n*distx/ndim,2)+ pow(m*disty/mdim,2))));
// cout<<"\n -phfac*(pow(n*distx/ndim,2)+ pow(m*disty/mdim,2)) \n"
<<-phfac*(pow(n*distx/ndim,2)+ pow(m*disty/mdim,2));
// cout<<"\n cos(-phfac*(pow(n*distx/ndim,2)+ pow(m*disty/mdim,2))) \n"
<<cos(-phfac*(pow(n*distx/ndim,2)+ pow(m*disty/mdim,2)));
// cin.get();
fldfres[n][m] =s1 ;
// cout<<"\n s \n"<<s1;
// cin.get();
// cout<<"\n fldfres[n][m] \n"<<fldfres[n][m];
// cin.get();
complex s2(cos(-phfac*(pow(n*distx/ndim,2)+ pow(m*disty/mdim,2))),
sin(-phfac*(pow(n*distx/ndim,2)+ pow(m*disty/mdim,2))));
fldfres[nr][m] =s2;
```

```
complex s3(cos(-phfac*(pow(n*distx/ndim,2)+ pow(m*disty/mdim,2))),
sin(-phfac*(pow(n*distx/ndim,2)+ pow(m*disty/mdim,2))));
fldfres[n][mr] =s3;
complex s4(cos(-phfac*(pow(n*distx/ndim,2)+ pow(m*disty/mdim,2))),
sin(-phfac*(pow(n*distx/ndim,2)+ pow(m*disty/mdim,2))));
fldfres[nr][mr] =s4;
}
}
//for n = 0
for(m=1;m<mdimh;m++){
mr = mdim-m;
complex phase(0.0,-phfac*(pow(0*distx/ndim,2)+ pow(m*disty/mdim,2)));
complex s5(cos(-phfac*(pow(0*distx/ndim,2)+ pow(m*disty/mdim,2))),
sin(-phfac*(pow(0*distx/ndim,2)+ pow(m*disty/mdim,2))));
fldfres[0][m] =s5 ;
complex s6(cos(-phfac*(pow(0*distx/ndim,2)+ pow(m*disty/mdim,2))),
sin(-phfac*(pow(0*distx/ndim,2)+ pow(m*disty/mdim,2))));
fldfres[0][mr] =s6 ;
}
//for m = 0
```

```
for(n=1;n<ndimh;n++){
nr = ndim-n;
complex phase(0.0,-phfac*(pow(n*distx/ndim,2)+ pow(0*disty/mdim,2)));
complex s7(cos(-phfac*(pow(n*distx/ndim,2)+ pow(0*disty/mdim,2))),
sin(-phfac*(pow(n*distx/ndim,2)+ pow(0*disty/mdim,2))));;
fldfres[n][0] =s7;
complex s8(cos(-phfac*(pow(n*distx/ndim,2)+ pow(0*disty/mdim,2))),
sin(-phfac*(pow(n*distx/ndim,2)+ pow(0*disty/mdim,2))));
fldfres[nr][0] =s8 ;
}
//for n centerline, n = ndimh
for(m=1;m<mdimh;m++){
mr = mdim-m;
complex phase(0.0,-phfac*(pow(ndimh*distx/ndim,2)+
pow(m*disty/mdim,2)));
complex s9(cos(-phfac*(pow(ndimh*distx/ndim,2)+
pow(m*disty/mdim,2))),
sin(-phfac*(pow(ndimh*distx/ndim,2)+ pow(m*disty/mdim,2))));
fldfres[ndimh][m] =s9;
complex s10(cos(-phfac*(pow(ndimh*distx/ndim,2)+
```

```
pow(m*disty/mdim,2))),
sin(-phfac*(pow(ndimh*distx/ndim,2)+ pow(m*disty/mdim,2))));
fldfres[ndimh][mr] = s10;
}
// for m centerline, m = mdimh
for(n=1;n<ndimh;n++){
nr = ndim-n;
complex phase(0.0,-phfac*(pow(n*distx/ndim,2)+
pow(mdimh*disty/mdim,2)));
complex s11(cos(-phfac*(pow(n*distx/ndim,2)+
pow(mdimh*disty/mdim,2))),
sin(-phfac*(pow(n*distx/ndim,2)+ pow(mdimh*disty/mdim,2))));
fldfres[n][mdimh] =s11;
complex s12(cos(-phfac*(pow(n*distx/ndim,2)+
pow(mdimh*disty/mdim,2))),
sin(-phfac*(pow(n*distx/ndim,2)+ pow(mdimh*disty/mdim,2))));
fldfres[nr][mdimh] = s12;
}
//points left over ndimh,mdimh; 0,0; 0,mdimh; ndimh,0;
{ complex phase(0.0,-phfac*(pow(ndimh*distx/ndim,2) +
```

```
pow(mdimh*disty/mdim,2)));
complex s13(cos(-phfac*(pow(ndimh*distx/ndim,2) +
pow(mdimh*disty/mdim,2))),
sin(-phfac*(pow(ndimh*distx/ndim,2) +
pow(mdimh*disty/mdim,2))));
fldfres[ndimh][mdimh] =s13 ;
}
{
complex phase(0.0,-phfac*(pow(0*distx/ndim,2)+
pow(0*disty/mdim,2)));
complex s14(cos(-phfac*(pow(0*distx/ndim,2)+
pow(0*disty/mdim,2))),
sin(-phfac*(pow(0*distx/ndim,2)+ pow(0*disty/mdim,2))));
fldfres[0][0] =s14 ;
}
{
complex phase(0.0,-phfac*(pow(0*distx/ndim,2)+
pow(mdimh*disty/mdim,2)));
complex s15(cos(-phfac*(pow(0*distx/ndim,2)+
pow(mdimh*disty/mdim,2))),
```

```
sin(-phfac*(pow(0*distx/ndim,2)+ pow(mdimh*disty/mdim,2))));
fldfres[0][mdimh] =s15 ;
}
{
complex phase(0.0,-phfac*(pow(ndimh*distx/ndim,2)+
pow(0*disty/mdim,2)));
complex s16(cos(-phfac*(pow(ndimh*distx/ndim,2)+
pow(0*disty/mdim,2))),
sin(-phfac*(pow(ndimh*distx/ndim,2)+ pow(0*disty/mdim,2))));
fldfres[ndimh][0] = s16;
}
return;
}
/*********************************************************/
/************************[writetest]**********************/
/*********************************************************/
void writetest(complex fld[][mdim], int ndim,int mdim,int numb)
//writes out complex array at terminal
{
cout<<"write out array for testing "<<numb<<"th time\n";
```

```
for(int n=0;n<ndim;n++) {
cout<<"\n";
for(int m=0;m<mdim;m++) {
cout<<" ("<<n<<","<<m<<") = "<<fld[n][m]<<" ";
}
}
cout<<"\n";
return;
}
/*************************************************************/
/*************************[writetest1]************************/
/*************************************************************/
void writetest1(complex fld[][b], int pdim)
//writes out complex array at terminal
{
cout<<"write out array for testing "<<1<<"th time\n";
for(int i=1;i<pdim;i++) {
for(int j=1;j<pdim;j++) {
cout<<"\n";
cout<<" ("<<i<<j<<") = "<<fld[i][j]<<" ";
```

```
}
}
cout<<"\n";
return;
}
/*********************************************************************/
/*************************[writebeam]*****************/
/*********************************************************/
void writebeam(complex fld[][mdim],int ndim, int mdim)
//write out to gaussbeam.dat for propagation
{
int n,m;
ofstream out_file("fresprop.dat");
if(!out_file)
cerr<<"failed to open fresprop.dat. \n";
for(n=0;n<ndim;n++) {
out_file<<"\n";
for(m=0;m<mdim;m++) {
out_file<<real(fld[n][m])<<" "
<<imag(fld[n][m])<<" ";
```

```
}
}
out_file.close();
return;
}
/*********************************************************/
/*************************[ FFT procedure ]*****************/
/*********************************************************************/
void fourn(float data[], unsigned long nn[], int ndim, int isign)
{
int idim;
unsigned long i1,i2,i3,i2rev,i3rev,ip1,ip2,ip3,ifp1,ifp2;
unsigned long ibit,k1,k2,n,nprev,nrem,ntot;
float tempi,tempr;
double theta,wi,wpi,wpr,wr,wtemp;
for (ntot=1,idim=1;idim<=ndim;idim++)
ntot *= nn[idim];
nprev=1;
for (idim=ndim;idim>=1;idim-) {
n=nn[idim];
```

```
nrem=ntot/(n*nprev);
ip1=nprev << 1;
ip2=ip1*n;
ip3=ip2*nrem;
i2rev=1;
for (i2=1;i2<=ip2;i2+=ip1) {
if (i2 < i2rev) {
for (i1=i2;i1<=i2+ip1-2;i1+=2) {
for (i3=i1;i3<=ip3;i3+=ip2) {
i3rev=i2rev+i3-i2;
SWAP(data[i3],data[i3rev]);
SWAP(data[i3+1],data[i3rev+1]);
}
}
}
ibit=ip2 >> 1;
while (ibit >= ip1 && i2rev > ibit) {
i2rev -= ibit;
ibit >>= 1;
}
```

```
i2rev += ibit;
}
ifp1=ip1;
while (ifp1 < ip2) {
ifp2=ifp1 << 1;
theta=isign*6.28318530717959/(ifp2/ip1);
wtemp=sin(0.5*theta);
wpr = -2.0*wtemp*wtemp;
wpi=sin(theta);
wr=1.0;
wi=0.0;
for (i3=1;i3<=ifp1;i3+=ip1) {
for (i1=i3;i1<=i3+ip1-2;i1+=2) {
for (i2=i1;i2<=ip3;i2+=ifp2) {
k1=i2;
k2=k1+ifp1;
tempr=(float)wr*data[k2]-(float)wi*data[k2+1];
tempi=(float)wr*data[k2+1]+(float)wi*data[k2];
data[k2]=data[k1]-tempr;
data[k2+1]=data[k1+1]-tempi;
```

```
data[k1] += tempr;
data[k1+1] += tempi;
}
}
wr=(wtemp=wr)*wpr-wi*wpi+wr;
wi=wi*wpr+wtemp*wpi+wi;
}
ifp1=ifp2;
}
nprev *= n;
}
}

/*************************************************************/
/**************************[main]*******************/
/*************************************************************/
void main(){
double delx=0;
double delfy=disty/mdim;
double delz=1;
double delfx=distx/ndim;
```

```
double dely=0;
cout<<"\n ndim = "<<ndim<<" mdim = "<<mdim<<
"\n distx = "<<distx<<" disty = "<<disty<<" distz = "<<distz<<
"\n PI = "<<PI<< " sigma = "<<sigma<<" wavelength = "
<<lambda<<" layers = "<<layers<<
"\n delx = "<<delx<<" dely = "<<dely<<" delz = "<<delz<<
"\n delfx = "<<delfx<<" delfy = "<<delfy<<"\n";
complex (*fldfres)[mdim] = new complex[ndim][mdim];
if (fldfres == NULL) {
cerr<<"Failed to allocate memory for fldfres\n";
//exit(EXIT_FAILURE);
}
fresprop(fldfres,ndim,mdim,delz,delx,dely);
writetest(fldfres,ndim,mdim,1);
writebeam(fldfres,ndim,mdim); //write to file gaussbeam.dat
int w,q,loops = 1;
unsigned long v[2];
double tolerance = 0.01;
double z = .10;
float c[sa],T[re],E[re];
```

```
complex (*D)[b] = new complex[b][b];

complex (*Y)[b] = new complex[b][b];

complex (*A)[b] = new complex[b][b];

complex (*B)[b] = new complex[b][b];

ifstream inFile;

ofstream outFile;

openfiles(inFile, outFile);

getinfo(loops, tolerance, z);

init(v,ndim);

diffractive(A,b,re);

adjust(A,Y,b);

convert(A,c,b,b);

fourn(c,v,2,-1);

division(c,toi);

convertback(A,c,b,b);

writebeam5(A,b,b);

arg(A,E,b); //trou

expo(A,B,b);

convert(B,c,b,b);

fourn(c,v,2,1);
```

```
convertback(A,c,b,b);
ampl(A,T,b);
RMS(Y,A,b,re);
exponentielle(D,Y,A,b);
for(int p=1;p<20;p++)
{
adjust(D,A,b);
convert(A,c,b,b);
fourn(c,v,2,-1);
division(c,toi);
convertback(A,c,b,b);
arg(A,E,b);
expo(A,B,b);
convert(B,c,b,b);
fourn(c,v,2,1);
//writedata(outFile,c,50);
convertback(A,c,b,b);
ampl(A,T,b);
RMS(Y,A,b,re);
exponentielle(D,Y,A,b);
```

```
}

writedata(outFile,E,re);

cin>>w;

closefiles(inFile,outFile);

}
```

Vitae

Michael Teissier was born in Metz, France on september 8th 1975. He graduated from ESIGELEC, Rouen, France with a diploma in electrical Engineering in July 1999. He was admitted to Lehigh University in August 1998. Since then, he has been working towards his master degree.

END OF TITLE