ADDITIONS TO THE IMAGING CAPABILITY OF SAMPLE

by

Mark D. Prouty

Memorandum No. UCB/ERL M84/111

10 December 1984

ADDITIONS TO THE IMAGING CAPABILITY OF SAMPLE

by

Mark D. Prouty

Memorandum No. UCB/ERL M84/111

10 December 1984

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Table of Contents

List of Figures

Additions to the Imaging Capability of SAMPLE

Mark D. Prouty

1. Introduction

As feature sizes decrease in optical lithography, becoming smaller than the ratio of wavelength to numerical aperture, the faithfulness of imaging is significantly reduced. This makes computer simulations of optical lithography more difficult, first of all because more general techniques, such as phase shift masks, are being used to improve the image and secondly because the differences between 1 dimensional and 2 dimensional patterns, such as line end shortening and rounding of square apertures, are more important. Therefore, the capabilities of imaging phase shifting masks and 2 dimensional masks have been added to the processing simulation program SAMPLE.

The second chapter of this paper outlines the details of the changes this author made in the program to calculate images of phase shift masks. The next chapter outlines some results obtained using the new program. The fourth chapter outlines the use of the 2-D code written by Shankar Subramanian. Since this work was never documented or implemented in the SAMPLE program this chapter will serve as a user's manual for 2-D imaging.

2. Calculating Images of Phase Shift Masks.

2.1. Introduction

The resolution in optical lithography is limited by proximity effects, that is, the spillover of diffracted light between adjacent features. In conventional lithography, light from each feature arrives in phase between them, causing a slightly lightened area where a dark one is desired. However, if the light coming from one of the features is delayed by a coating so that it arrives 180° out of phase, the two diffracted beams will cancel, and the desired dark area will be obtained.

A mask using this technique, first proposed by Levenson, et al.(1), has been dubbed a phase shift mask. Levenson, using the basic case of a 180° phase shift, predicted theoretically and verified experimentally a much increased contrast. The success of this phase mask approach has raised many conceptual and practical questions. Therefore, we decided to enhance the capabilites of the SAMPLE program.

The SAMPLE program calculates images using Hopkins(2) theory of partially coherent imaging. Briefly, the image intensity is given as the convolution of the Fourier transform of the mask intensity distribution with a function known as the transmission cross-coefficient (TCC). The TCC is in turn the convolution of two functions, one over the condenser aperture and the other over the objective pupil. More details of this theory will be given later.

Previously, the SAMPLE program calculated images of real masks which only required calculating the real part of the TCC. We have modified the program to calculate complex coefficients for the Fourier series of the mask and to also calculate the imaginary part of the TCC. This is a more complicated procedure, but many symmetries could be exploited to shorten the numerical integration. In addition, the program will accept masks with up to 33 different regions with arbitrary amplitude transmission and phase shift. This method requires slightly more CPU time than the previous SAMPLE version, up to twice as long. This time is from 2 to 12 CPU seconds on a VAX 11/780 with UNIX, depending on the size of the image. For an image period of 2 microns, for example, 4 seconds are required.

2.2. Mathematical Methods

Kintner(3) gives details of the partial coherence theory, and Subramanian(4) shows how the theory is used in the SAMPLE program. There it is shown that the transform of the image intensity may be given as

$$I(f)=\sum_{i=1}^{\infty}\left\{a_n a_o^* T(nf_p,0)+\sum_{m=1}^{\infty} a_{n+m}a_m^* T((n+m)f_p,mf)+a_{n-m}a_n^* T((n-m)f_p,-mf_p)\right\}\delta(nf_p-f)$$

where f_p equals the reciprocal of the period.

$T(f',f'')$ is known as the transmission cross coefficient and is given by

$$T(f',f'')=\int\int J(f,g)\,K(f+f',g)\,K^*(f+f'',g)\,df\,dg \qquad (2)$$

where J, the Fourier transform of the mutual intensity function at the object is

$$J=\begin{Bmatrix}1 & f^2+g^2<s\\ 0 & f^2+g^2\geq s\end{Bmatrix} \qquad (3)$$

where s is the partial coherence factor and K, the objective pupil function is

$$K=\begin{Bmatrix}\exp-2\pi i\frac{\mu}{4}(f^2+g^2) & f^2+g^2<1\\ 0 & f^2+g^2\geq 1\end{Bmatrix} \qquad (4)$$

Here, μ is related to the defocus in microns by

$$\mu=d\frac{NA^2}{\lambda} \qquad (5)$$

The a_n 's are the fourier coefficients of the mask transmission, so that $F(x)$, the intensity and phase of the light at the mask is

$$F(x)=a_o+\sum_{n=1}^{\infty}a_n\cos 2\pi n f_p x \qquad (6)$$

For masks without phase shifts the a_n 's are real. Here, however, they may be complex.

These equations lead to a more specific expression for the TCC

$$T(f',f'')=\int\int_{\Omega}\exp-2\pi\mu i\left[f'^2-f''^2+2f(f'-f'')\right]df\,dg \qquad (7)$$

where Ω is the region of intersection of 3 circles, C, C', and C'' . C is the region where $J(f,g)$ is non-zero, *viz.* a circle of radius s centered at the origin. C' is the region where $K(f+f', g)$ is non-zero, *viz.* a circle of radius 1 centered at $(-f', 0)$ and C'' is the region where $K^*(f+f'', g)$ is non-zero, *viz.* a circle of radius 1 centered at $(-f'', 0)$. Several cases of these intersections are shown in figures 1-4.

2.3. Implementation

The specific formulas used in the program to integrate equation (7) are dependent upon the geometry of the circles C, C', C''. For the case shown in Figure 1

$$
\begin{aligned}
\mathrm{Re}T(f', f'') &= \int\int \cos -2\pi\mu\left\{f'^2 - f''^2 + 2f\,(f' - f'')\right\} df\, dg \\
&= \int_{-1-f'}^{f_1} \sqrt{1-(f+f')^2}\, \cos -2\pi\mu\left\{f'^2 - f''^2 + 2f\,(f' - f'')\right\} df \\
&+ \int_{f_1}^{f_2} \sqrt{s^2 - f^2}\, \cos -2\pi\mu\left\{f'^2 - f''^2 + 2f\,(f' - f'')\right\} df \\
&+ \int_{f_2}^{1-f''} \sqrt{1-(f+f'')^2}\, \cos -2\pi\mu\left\{f'^2 - f''^2 + 2f\,(f' - f'')\right\} df
\end{aligned}
\tag{8}
$$

ImT is identical, except cos is replaced by sin.

These integrands all contain radicals unsuitable for Gaussian quadrature. Therefore, a transformation of the form $f = s \cos x$ or $f + f' = \cos x$ is made. Then the first integral becomes

$$
\int_{\cos^{-1}(f_1+f')}^{\pi} \sin^2 x \cos -2\pi\mu\left\{f'^2 - f''^2 + 2(\cos x - f')(f' - f'')\right\} dx \tag{9}
$$

With the further changes $\sin^2 x \to (1-\cos 2x)/2$, $f' \to -f$, and $\cos x \to \cos(\pi - x)$ (which just changes the limits of integration), these integrals are encoded in the program as functions gt1 and gt1i (gt1 calculates the real part and gt1i calculates the imaginary part). The second integral becomes functions gt2 and gt2i. The functions gt1 and gt1i are also used to integrate the third expression.

When the geometry is as in figure 2, gc1 and gc1i are used to integrate (7). Further, for the case where s > 1, ft2(i) and ft3(i) are used for cases shown in figure 3 and 4, respectively.

3. Results Using the Enhanced Program

3.1. Introduction

Here, we explore a number of the phase mask imaging issues raised by Levenson's work. The first question raised is how much the contrast of periodic features depends upon defocus and the amount of phase shift used, and how this dependence changes with feature size. The second issue is what size defects in the phase shifting material may be tolerated. We also explore the possibility of making clear field transitions between phases without causing printable defects. The final issue is whether unprintable proximity features may be used to enhance the image of isolated features.

3.2. Results

Image calculations of the new extension to SAMPLE indicate the marked improvement in image quality possible in using phase-shift masks. Figure 5 shows the intensity profile for a periodic series of lines and spaces 0.75 μm in width (in all simulations throughout this chapter the wavelength used is 0.4358 μm, the lens has a numerical aperture of 0.28 and the defocus is 1.5 μm). Curve B is the image obtained with no phase shifting, and curve A is that obtained with every other space phase shifted 180°. The large improvement in contrast is due both to a decrease in the minimum intensity, as well as to an increase in the maximum intensity. This increase is due to the extra path difference involved in the light from both images. The beams that arrived out of phase in conventional lithography now arrive in phase.

This improvement is quantified more fully in figures 6 through 9. In figure 6 we plot contrast ($(I_{max}-I_{min})/(I_{max}+I_{min})$) vs. defocus for a partial coherence factor (sigma) of 0.7. Figure 7 shows the same thing for a sigma of 0.3. In both cases contrast is improved. The improvement is much greater for the lower sigma values, which is what we would expect from the mutual coherence function (MCF). This function gives the degree of correlation of the incident light as a function of distance across the mask. The width of its central lobe is equal to $1.22\lambda/NA\sigma$. Since phase changes are significant only for phase related beams, the larger the central lobe of the MCF (caused by a smaller sigma), the greater we would expect to be the effect of phase shifts.

Examining figure 7 more closely, we see how the phase shifting not only increases the level of contrast, but also makes the contrast more focus tolerant. This graph clearly shows the steady improvement possible by increasing the phase shift up to 180°. However, we shall see later that we must pay a price for this.

In figures 8 and 9 we now plot contrast vs. size, still for periodic sequences of lines and spaces. These curves show that for widths of less than 1 μm, steady contrast improvement may be obtained by increasing the phase shift. Above 1 μm proximity effects are lessened, and the two features are further apart than the diameter of the MCF. Therefore, little improvement is possible for features larger than 1 μm.

Improvement in image quality, unfortunately, comes at the price of an improvement in defect printability. In figure 10 we plot the image intensity for 0.2 μm wide 180° and 90° phase shift material defects on clear areas. This produces surprisingly large ripples in the intensity, which may leave resist remaining on the wafer in what was supposed to be a clear area. For features this small, however, the assumption of uniform intensity across the opening in the mask, made by the image calculation algorithm, is not exactly true and these results only give an approximate indication of the problem for a 1X system.

Defect printability may be reduced by using phase shifts of less than 180°. This is shown in figures 11 and 12 where we plot, for sigmas of 0.7 and 0.3 respectively, image intensity minimum vs. defect size. Thus, by lowering the phase angle, defect printability may also be lowered. For comparison, the contrast of an opaque defect is also shown in these figures.

The tradeoff between defect printability and contrast improvement is summarized in figure 13. There we have plotted, vs. defect size, the phase which gives an intensity minimum of 0.5, which is the minimum tolerable intensity. We have also plotted the phase angle which gives periodic features a contrast of 0.85, which we have taken as the minimum allowable feature contrast. As an example, if we wanted periodic features 0.75 μm wide, we need at least a phase shift of 90°. Moving to the left until we hit the defect line, then dropping down we see that the largest allowable defect at that phase angle is 0.2 μm.

In a 2-dimensional mask, it would be desirable for all adjacent features to be out of phase with respect to each other. This is similar to the problem of coloring a map with only 2 or 3 colors. Thus, it might be desirable to switch the phase of different parts of the same opening, that is, to color parts of the same country different colors. This type of phase transition, however, will cause a printable glitch as seen in figure 14. This figure plots the phase shift dependence of the image intensity minimum of a large clear region next to a large, clear, phase shifted region. We see profiles for a 180° and 90° phase transition in figure 15. Obviously, this type of phase shift will cause a defect to print.

We may reduce the contrast of this type of transition be making a smooth change in phase over a small length. Figure 16 shows profiles for a 180° phase shift made smoothly (by making 12 steps of 15° each) over a length of 0.6, 0.9, and 1.2 μm. This reduces the contrast, but separating the 0° and 180° regions by more than 1.0 μm defeats the purpose of having the phase transition. Thus even this smooth phase transition method is no good.

The discussion of printing features up to now has involved periodic sequences of lines and spaces. Another use of the phase mask is to add non-printable features next to an isolated feature to enhance its image. Figure 17 shows an attempt to improve the image of a .62 μm wide isolated line (this size was chosen because the intensity peak of the unenhanced image drops to one half the clear field value). Curves A through C are for the line with a 0.3, 0.2, and 0.1 μm wide 180° phase shifted proximity feature on each side, respectively. Curve D is for the line by itself. The center to center distance between the line and the proximity feature is 1.15 μm in all cases. The maximum intensity of the line increases as the proximity linewidth increases, and the peak becomes slightly narrower, making this a potentially useful technique.

3.3. Conclusion

We see, then, that phase masks may improve both periodic and isolated features. The size of periodic features may be reduced from 1.2 μm to 0.5 μm and still have a contrast of 0.85 by using phase shifts. This improvement is robust with respect to phase shift angle; 120° gives nearly as good results as 180°. Below 120°, however, improvement does drop off. The peak intensity of isolated features may be increased 30 percent by using unprintable proximity features. This improvement decreases for phase shifts of less than 120 °.

We have also seen that phase transitions above 90° will produce defects. Thus, clear field transitions are not viable even when the transition is relatively smooth. Another key problem is the printability of small defects. We have shown that a 90° defect prints about the same as an opaque defect, and shifts of 120° and 180° cause defects twice as small to print. With the limitations of no clear field transitions and proper defect control very significant improvements in aerial image quality are possible with phase masks.

4. 2-D User's Manual

4.1. Introduction

Not included within the SAMPLE program, but related to it, are a set of programs written by Shankar Subramanian for calculating 2 dimensional images. Programs for plotting contour plots and profiles of the images, written by this author, are also included. This chapter will outline the use of these programs.

4.2. Running the Image Program

Two versions of Shankar's imaging program exist. The program **image.out** calculates images with arbitrary defocus, while **nodef.out** calculates images with no defocus. Both programs assume $\lambda = .436\ \mu m$, $NA = 0.28$, and have the same I/O format.

Each program reads 6 parameters, *xs,xl,ys,yl,s,def*, in free format from a file named **lines**. The first four parameters specify the mask intensity in the following way. The mask intensity is assumed to be a separable function of x and y, i.e.

$$I(x,y) = X(x)Y(y) \tag{10}$$

$X(x)$ has period $xs+xl$, and $Y(y)$ has period $ys+yl$. The program defines $X(x)$ in the first period as follows

$$X(x) = \begin{cases} 0 & -(xs+xl)/2 \le x \le -xs/2 \\ 1 & -xs/2 \le x \le xs/2 \\ 0 & xs/2 \le x \le (xs+xl)/2 \end{cases} \tag{11}$$

$Y(x)$ is defined similarly. The parameter s is the partial coherence factor, and def is the defocus in μm. The program **nodef.out** ignores def.

The output of the programs is written to an unformatted file called **rint**, which contains a 100 by 100 array of intensities, followed by the two parameters

dx and dy. These two parameters are the spacings between x and y points, respectively, in units of λ / NA

$$dx = \left[\frac{NA}{\lambda}\right]\left[\frac{xs + xl}{198}\right] \qquad (12)$$

rint contains intensities for one quadrant of one period of the image, which, because of the x and y symmetry, contains all the information of the entire image. *rint*(1,1) is the intensity at the origin, *rint*(1,100) is the intensity at (0, $99dy\lambda / NA$) . *rint*(100,1) is the intensity at ($99dy\lambda / NA$, 0). The interactive programs **profile** and **contour** sets up a file of points called **f77punch7** suitable for plotting with the SAMPLE plotting programs.

4.3. How the Program Works

The 2-D program uses the same partial coherence formulas as the 1-D case, except, of course, in two dimensions. This makes the geometry more complicated, since the circles C', and C' are no longer centered on the x axis.

The main program calculates the Fourier coefficients of the mask. Currently, the method used is a specific one for the separable function of x and y assumed. The rest of the program, however, does not require that simplification. Therefore, a more general routine for calculating Fourier coefficients could be added. The only requirement is that the intensity transmission be real, and periodic and symmetric in both x and y directions.

After computing the Fourier coefficients, main calls spect, the subroutine which calculates the transform of the image intensity, performing many summations similar to the 1-D case. The transmission cross-coefficients, now functions of four variables, are calculated by the function t, two versions of which exist. The file **cross1.f** contains the version for the no defocus case, while the general case is located in **cross2o.f**, each of which requires several functions. The required modules are **main.f**, **cross1.f** (**cross2o.f**), arc.f, rsect.f, **inside.f**, **int.f**, and **spec1.f**.

The run time for these programs varies roughly proportionally to the area of the image period. The program **image.out** requires about 20 CPU seconds on a VAX 11/780 with UNIX per square micron, while **nodef.out** requires about half that.

5. Acknowledgement

This work was supported by the National Science Foundation under grant ECS-8106234 and industrial contributions under the MICRO program (83-14).

Appendix 1

Comparison of old and new SAMPLE images.

The following table shows a comparison of the output for a periodic series of 0.75 μm lines and spaces calculated three different ways. First, using SAMPLE 1.5b and input A, second using SAMPLE 1.5c and input A, and third using SAMPLE 1.5c and input B. All images are identical, as expected.

distance	intensity		
μm	1.5b input A	1.5c input A	1.5c inputB
0.000	.220	.220	.220
0.031	.221	.221	.221
0.061	.225	.225	.225
0.092	.231	.231	.231
0.122	.240	.240	.240
0.153	.251	.251	.251
0.184	.264	.264	.264
0.214	.280	.280	.280
0.245	.298	.298	.298
0.276	.319	.319	.319
0.306	.342	.342	.342
0.337	.367	.367	.367
0.367	.393	.393	.393
0.398	.420	.420	.420
0.429	.449	.449	.449
0.459	.477	.477	.477
0.490	.505	.505	.505
0.520	.533	.533	.533
0.551	.558	.558	.558
0.582	.582	.582	.582
0.612	.602	.602	.602
0.643	.619	.619	.619
0.673	.633	.633	.633
0.704	.642	.642	.642
0.750	.647	.647	.647

input A:

```
proj .6
lambda .5
trial 3 1 0 1
linespace .75 .75
run 1
end
```

input B:

```
proj .6
lambda .5
trial 3 1 0 1
trial 19 0 0 .75 1 0 .75
run 1
end
```

Invoking the new SAMPLE routines

The new routines are invoked through the 'trial 19' statement. Its form is:

trial 19 (amp1, phase1, dist1), ... , (ampN, phaseN, distN)

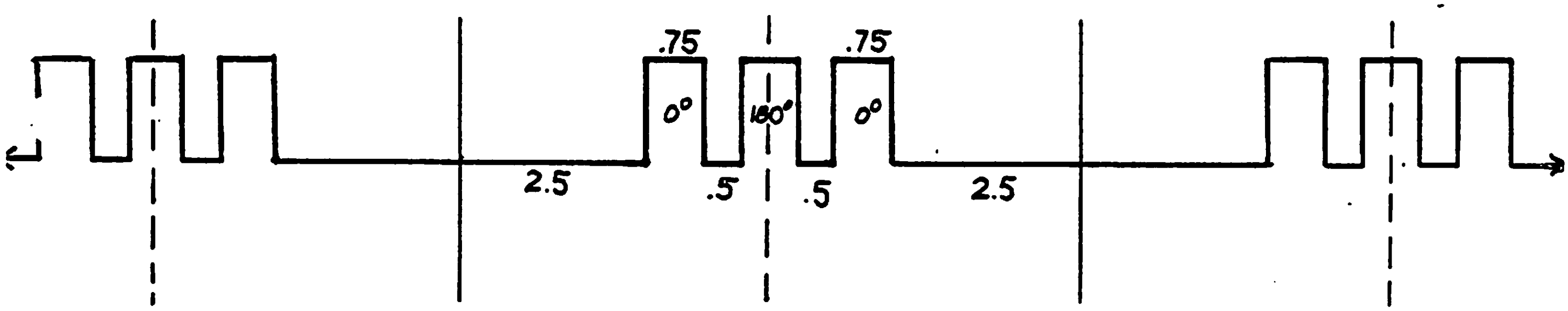
where amp, phase, and dist specify the amplitude and phase (in degrees) transmission for a region of length dist. Up to 33 regions may be specified.

The program then makes an even, periodic extension of this function It does so by first halving the length of the first and last regions. This now forms one half-period of the function. The other half-period is the mirror image of the first.

For example, the statement,

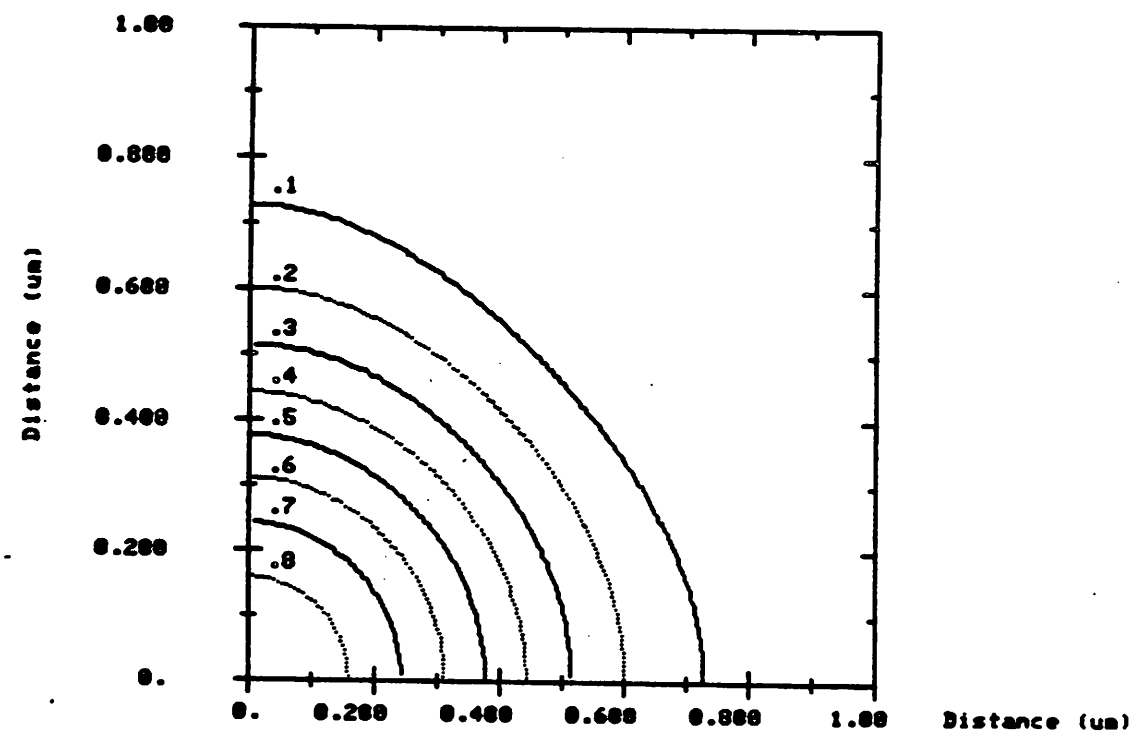
trial 19 1 180 .75 0 0 .5 1 0 .75 0 0 5

produces:



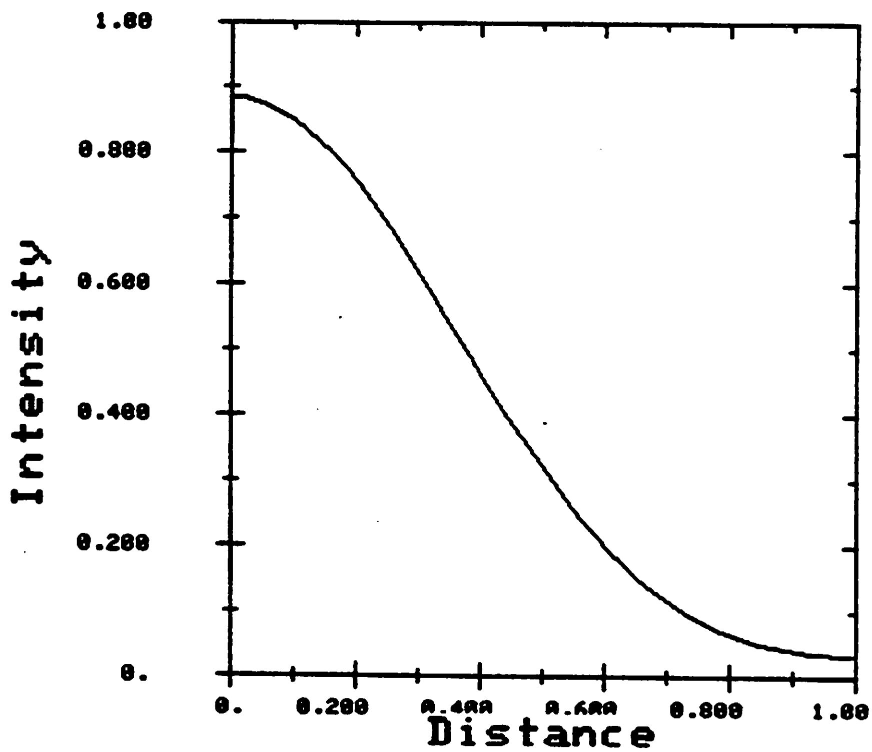
Appendix 2

The following is an input example for **image.out**. The file **lines** contains the one line: **1.0 1.0 1.0 1.0 .3 1.5** . Running **image.out**, then **contour**, and plotting the file **f77punch7** produces:



These routines are currently located in the directory, **/od/sample/prouty/Twodim**.

Running **profile**, and plotting **f77punch7** produces:



CPU time for calculating this image was 20 seconds.

Appendix 3

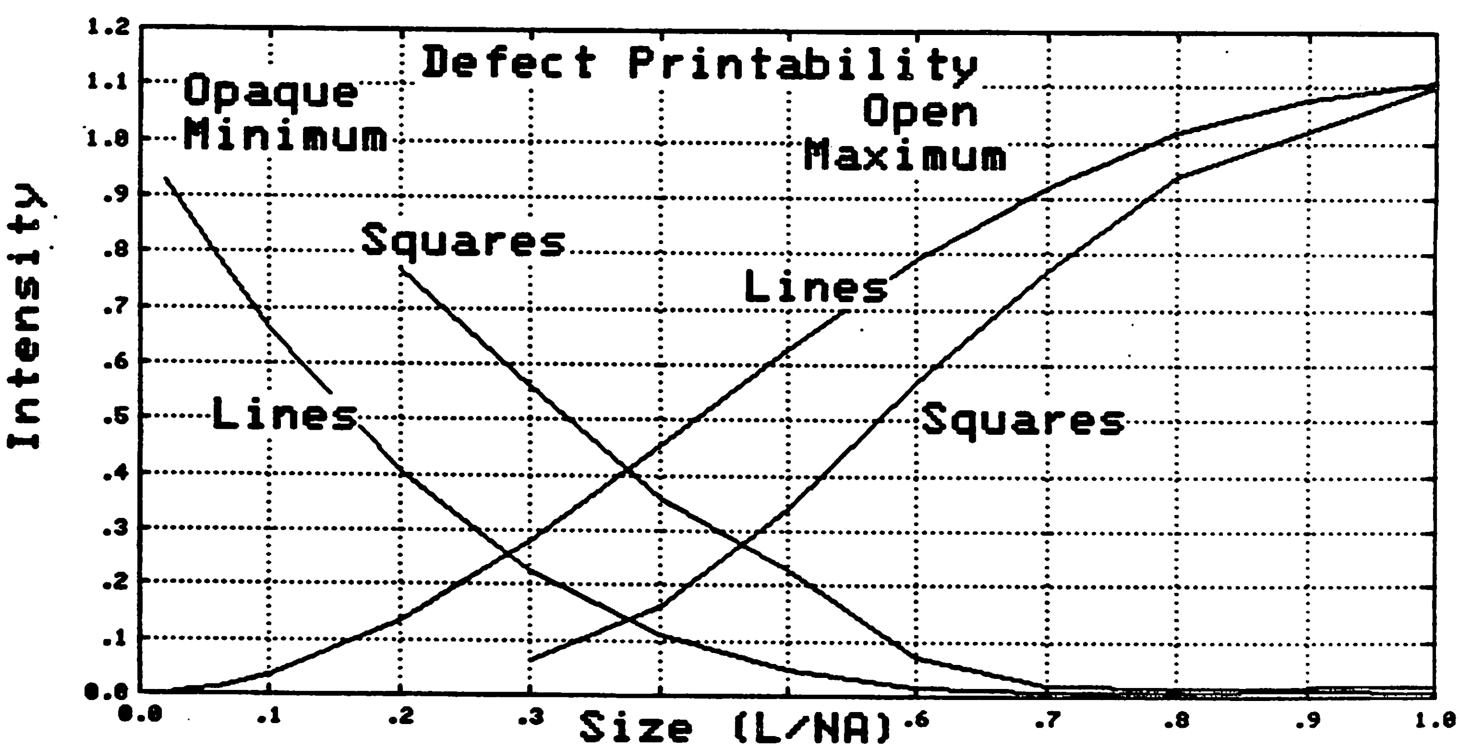
Comparison of Goodman and Subramanian

The following table compares the intensity at the center of small isolated squares of different sizes. Goodman's program was run by this author while working at IBM Watson Research Center. As shown, Subramanian's program gives identical results.

size λ / NA	Goodman	Subramanian
1.0	1.10 †	1.165
0.8	.94 †	.954
0.7	.77 †	.762
0.6	.57 †	.547
0.5	.34 †	.338
0.4	.16 †	.171
0.3	.064	.064

† These numbers were extrapolated from a graph and are accurate only to ±0.02.

The following figure plots the intensity at the center of small opaque and small open squares and, for comparison, lines.



References

[1] M.D. Levenson, N.S. Viswanathan, R.A. Simpson, "Improving Resolution in Photolithography with a Phase-Shifting Mask," IEEE Trans. Elect. Dev., vol ED-29 No. 12, p1828, 1982.

[2] H.H. Hopkins, Proc. R. Soc. London Ser A, vol 217, p408, 1953.

[3] E.C. Kintner, "Method for the Calculation of Partially Coherent Imagery," Appl. Opt., vol 17, p2747, 1978.

[4] S. Subramanian, "Rapid Calculation of Defocused Partially Coherent Images," Appl. Opt., vol 20, p1854, 1981.

[5] M. Prouty, A. Neureuther, "Optical Imaging With Phase Shift Masks," SPIE conference on Microlithography, 470-26, March 1984.

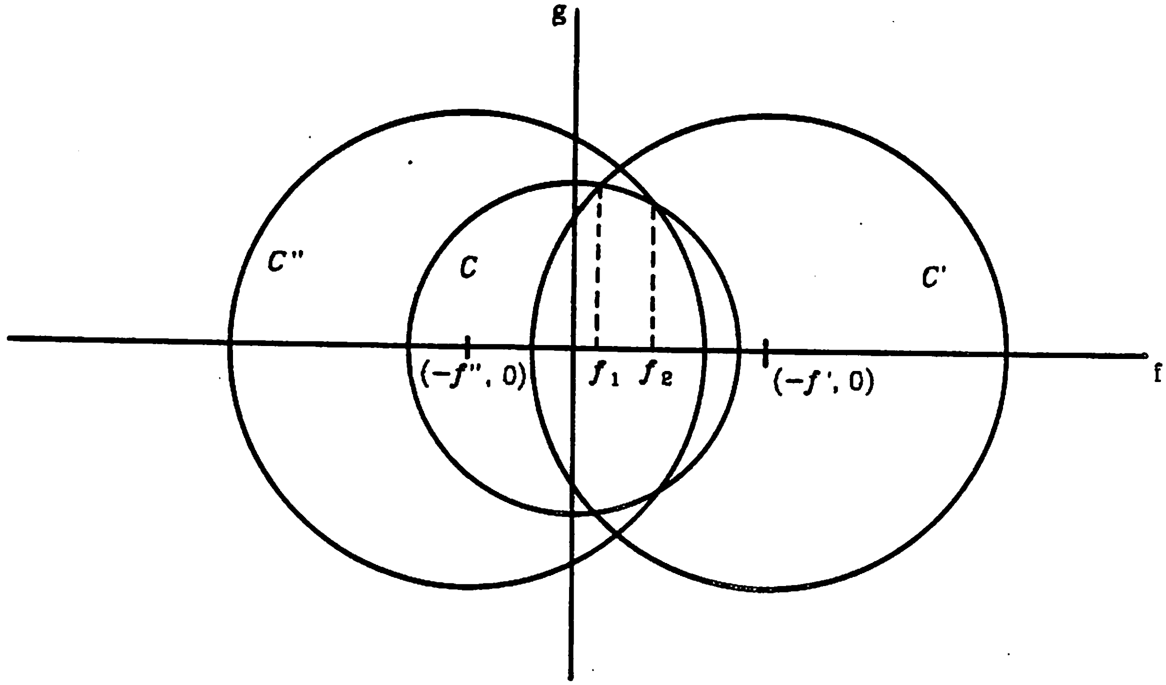


Figure 1. Region of integration for equation (7)

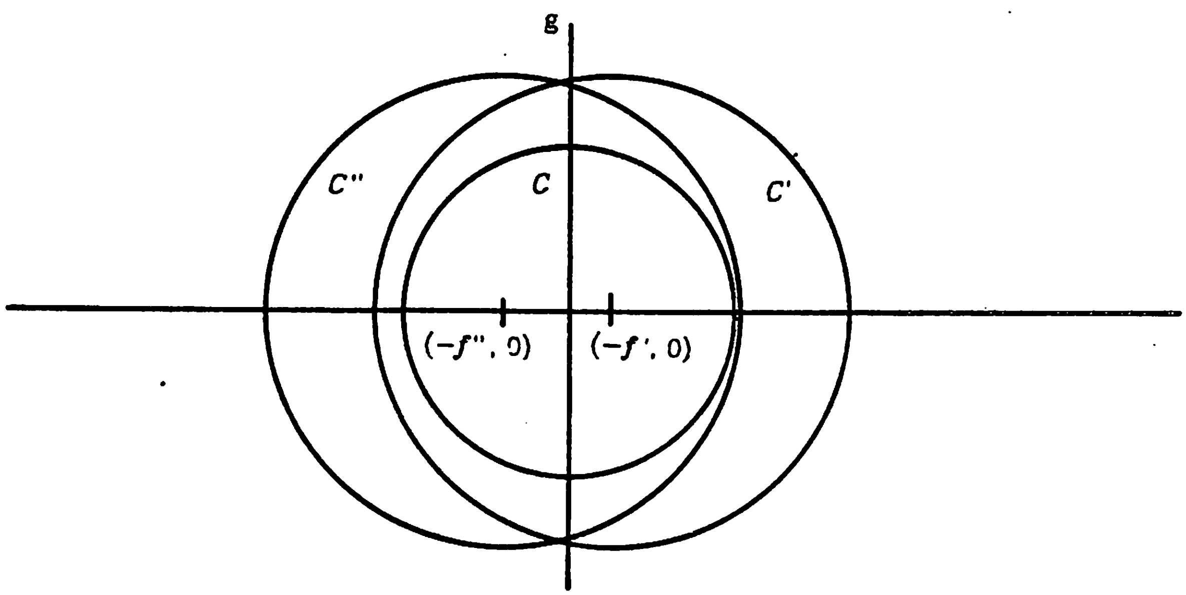


Figure 2. Region of integration for equation (7)

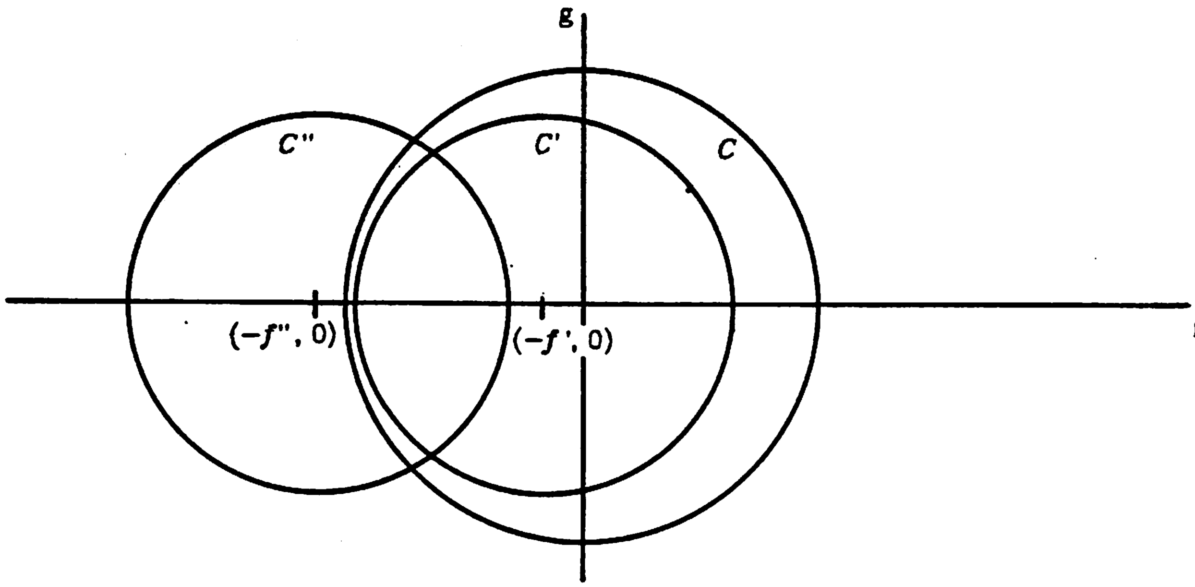


Figure 3. Region of integration for equation (7)

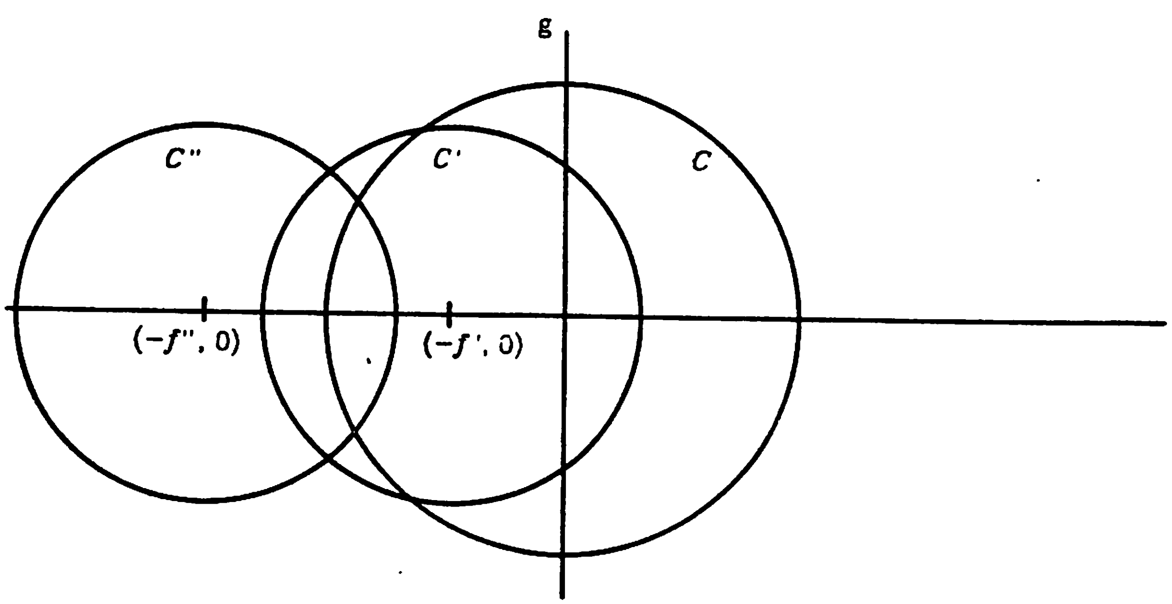


Figure 4. Region of integration for equation (7)

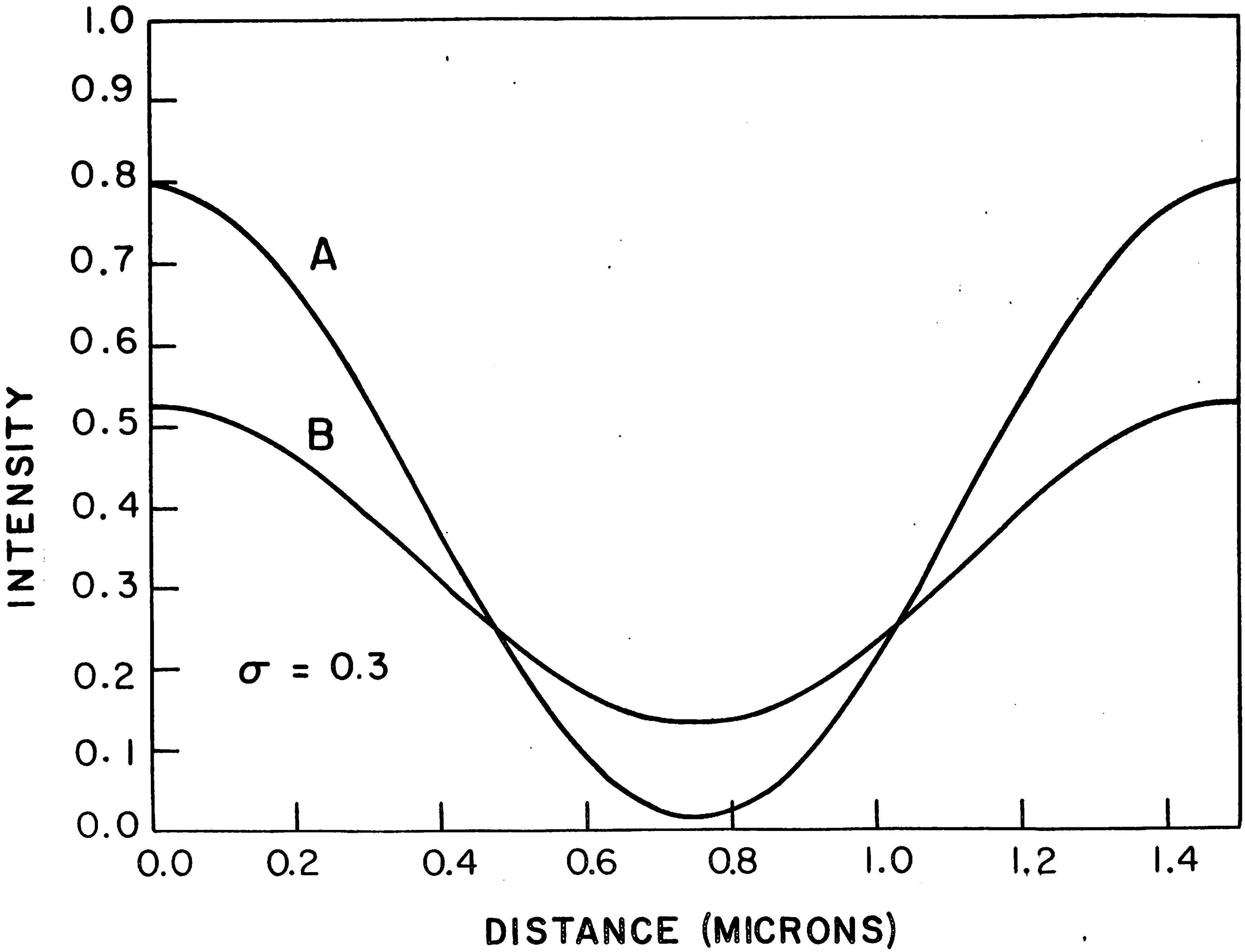


Figure 5. Image intensity for a periodic series of lines and spaces 0.75 μm wide, for no phase shift (B) and 180° phase shift (A).

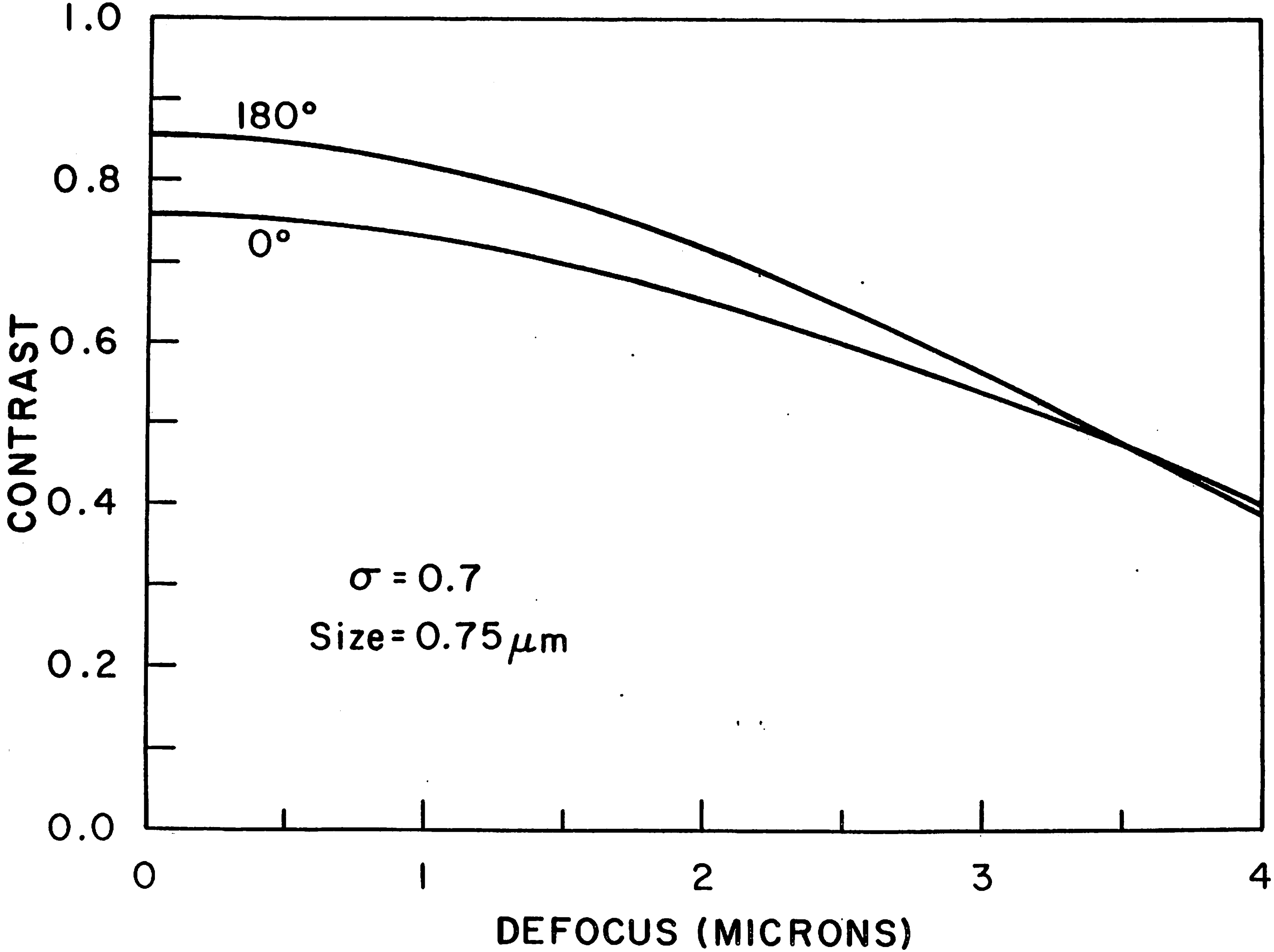


Figure 6. Contrast vs. Defocus for series of lines and spaces.

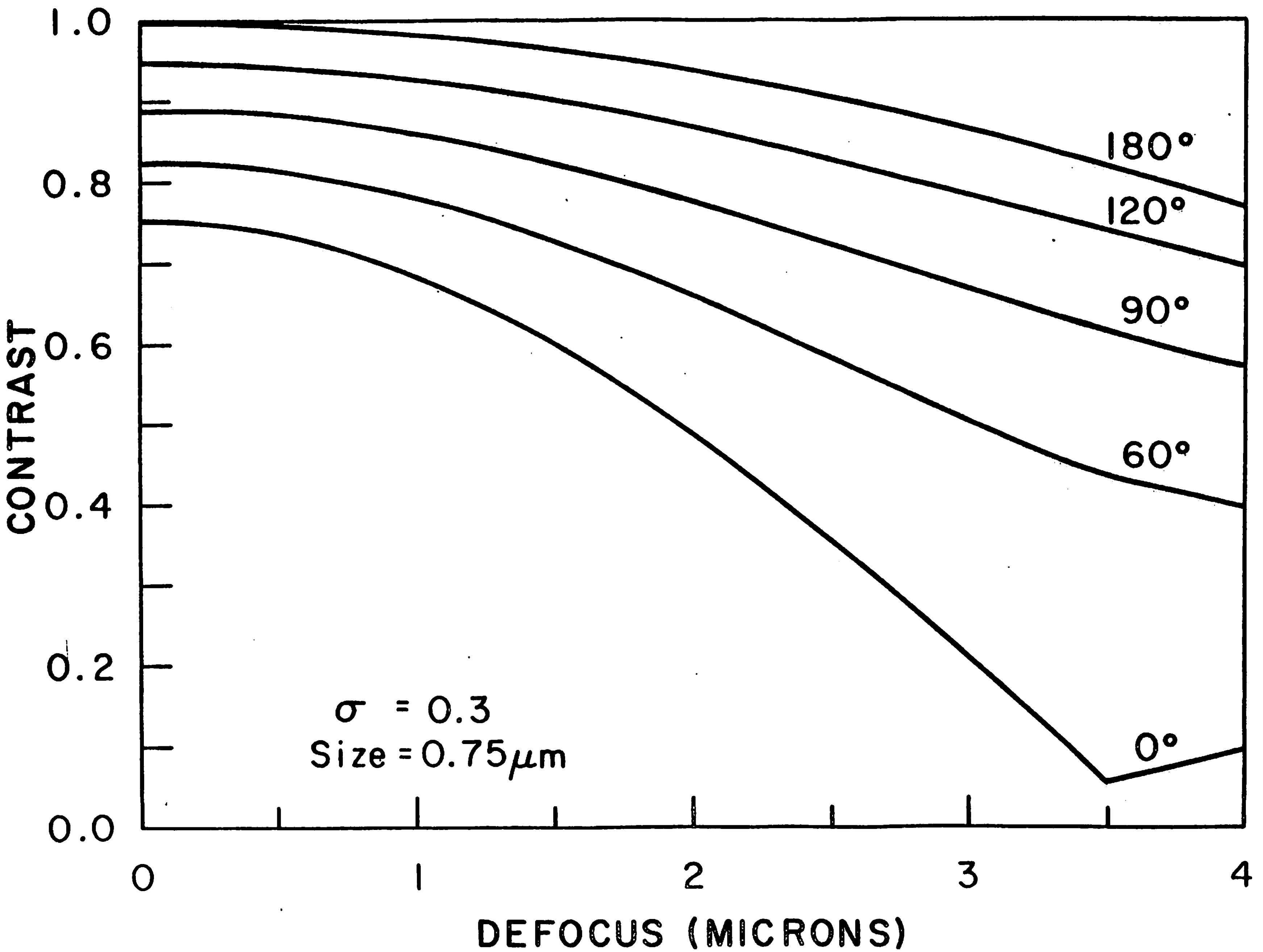
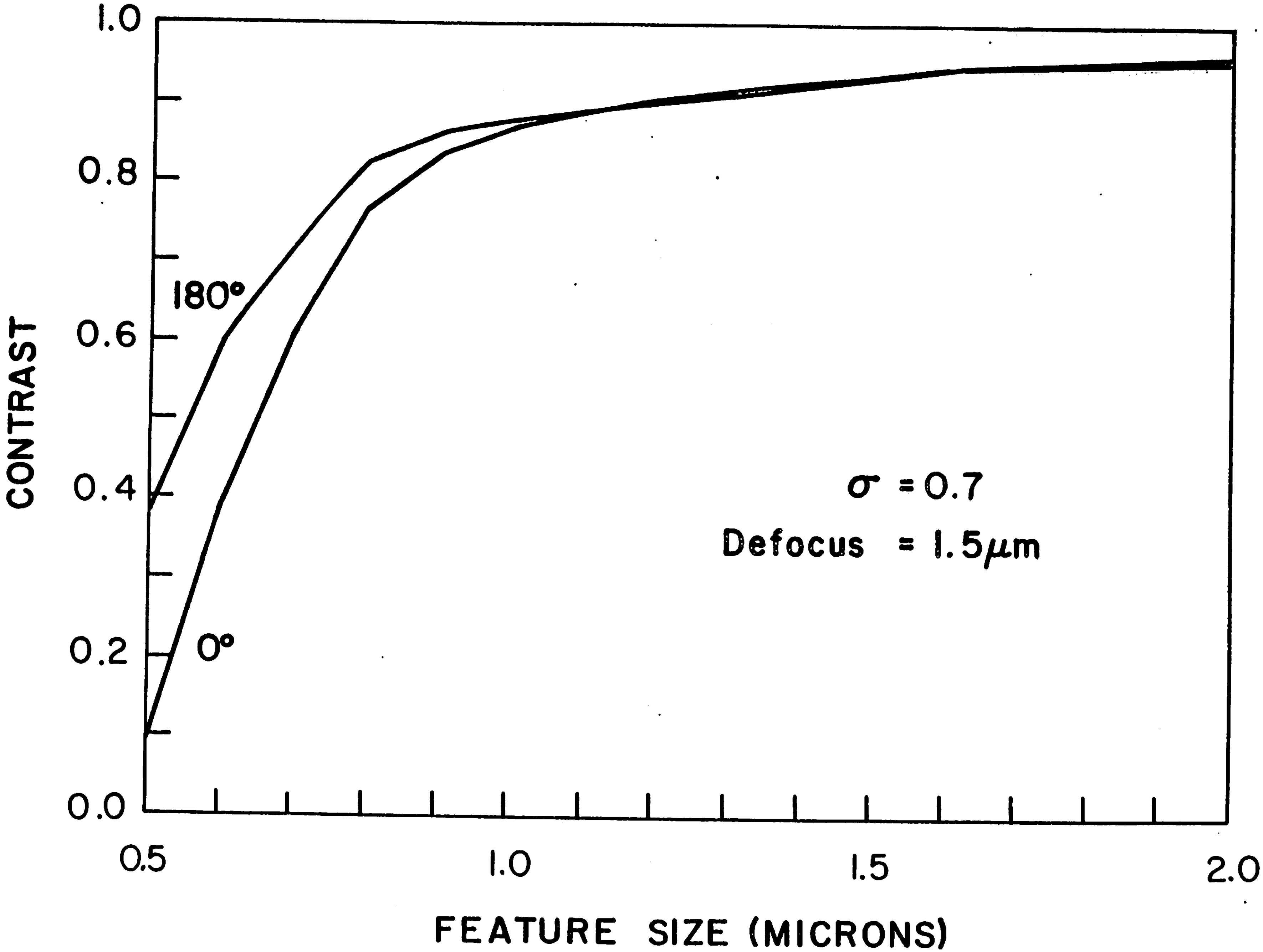


Figure 7. Contrast vs. Defocus for series of lines and spaces.

Figure 8. Contrast vs. Feature size for series of lines and spaces.



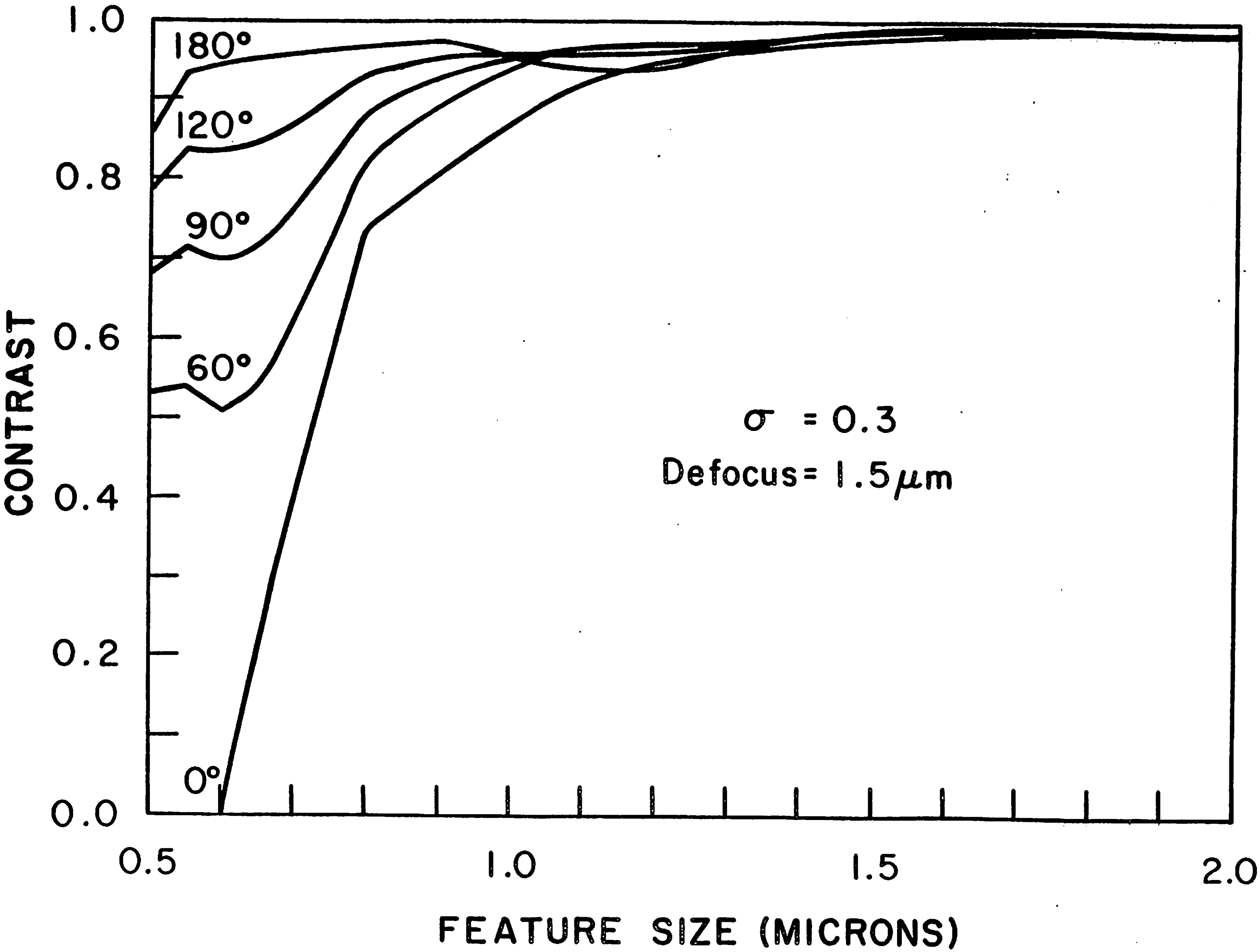


Figure 9. Contrast vs. Feature size for series of lines and spaces.

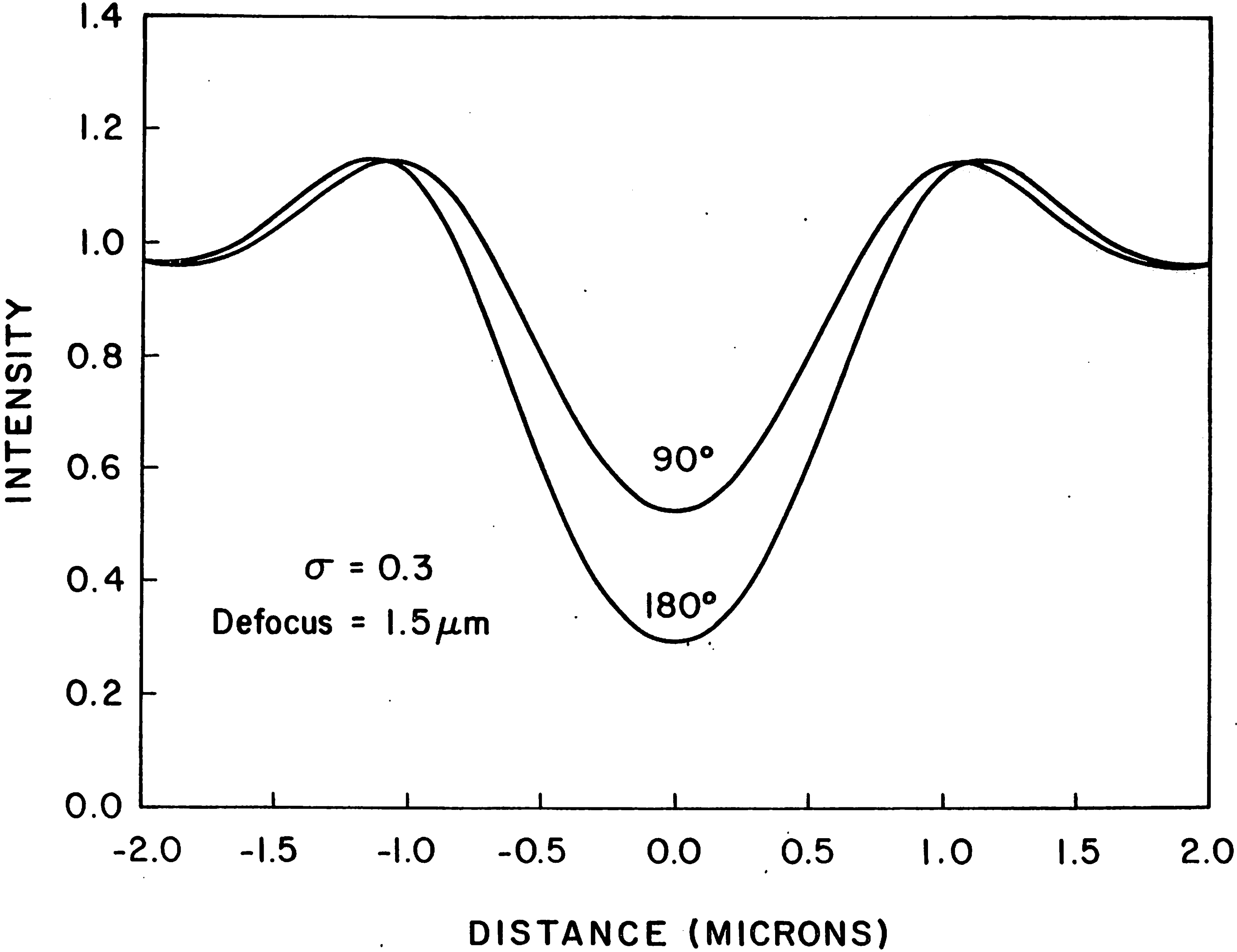


Figure 10. Intensity profile for 0.2 μm wide phase shift line defect.

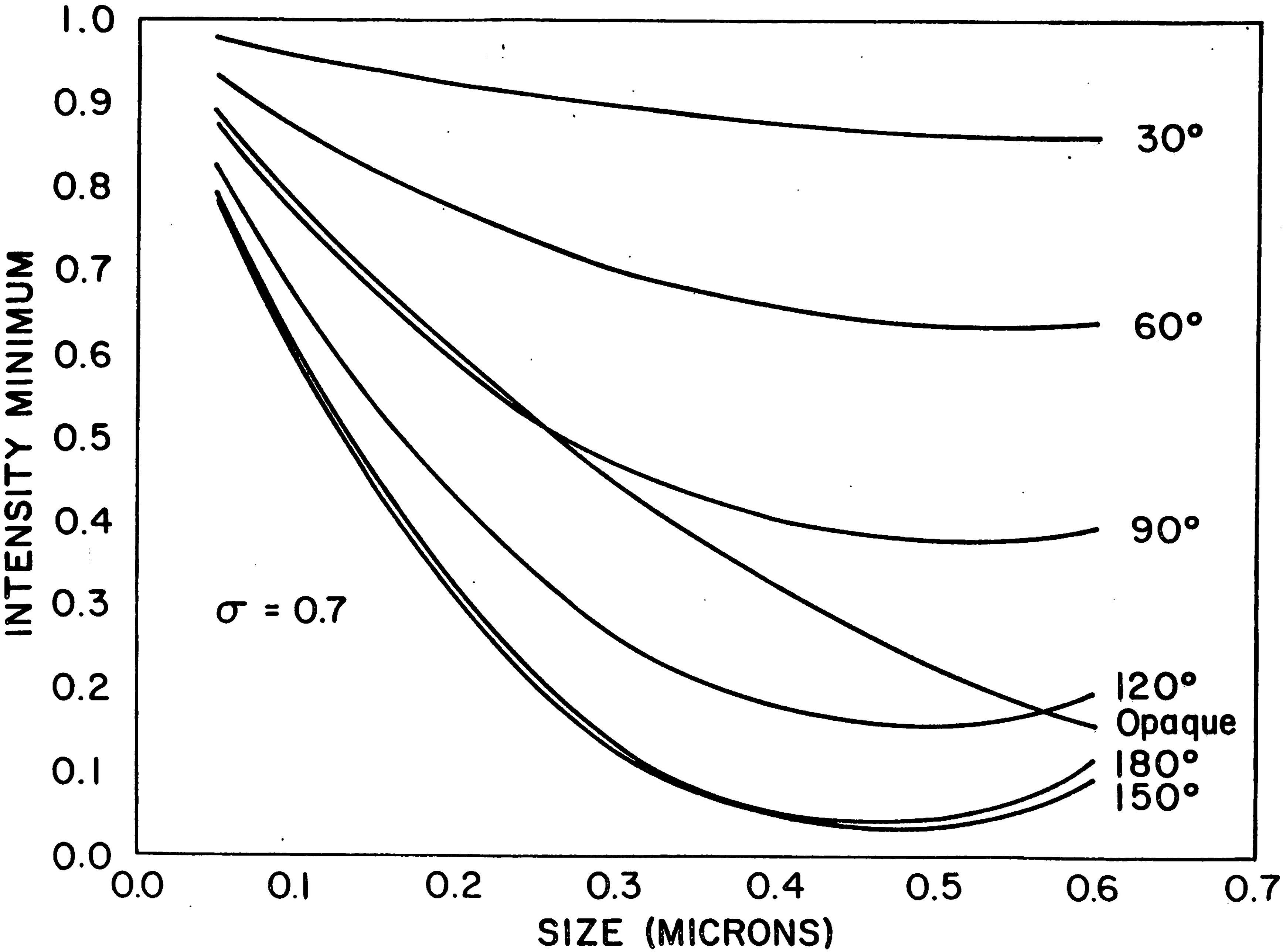


Figure 11. Intensity minimum vs. size for phase shift line defects.

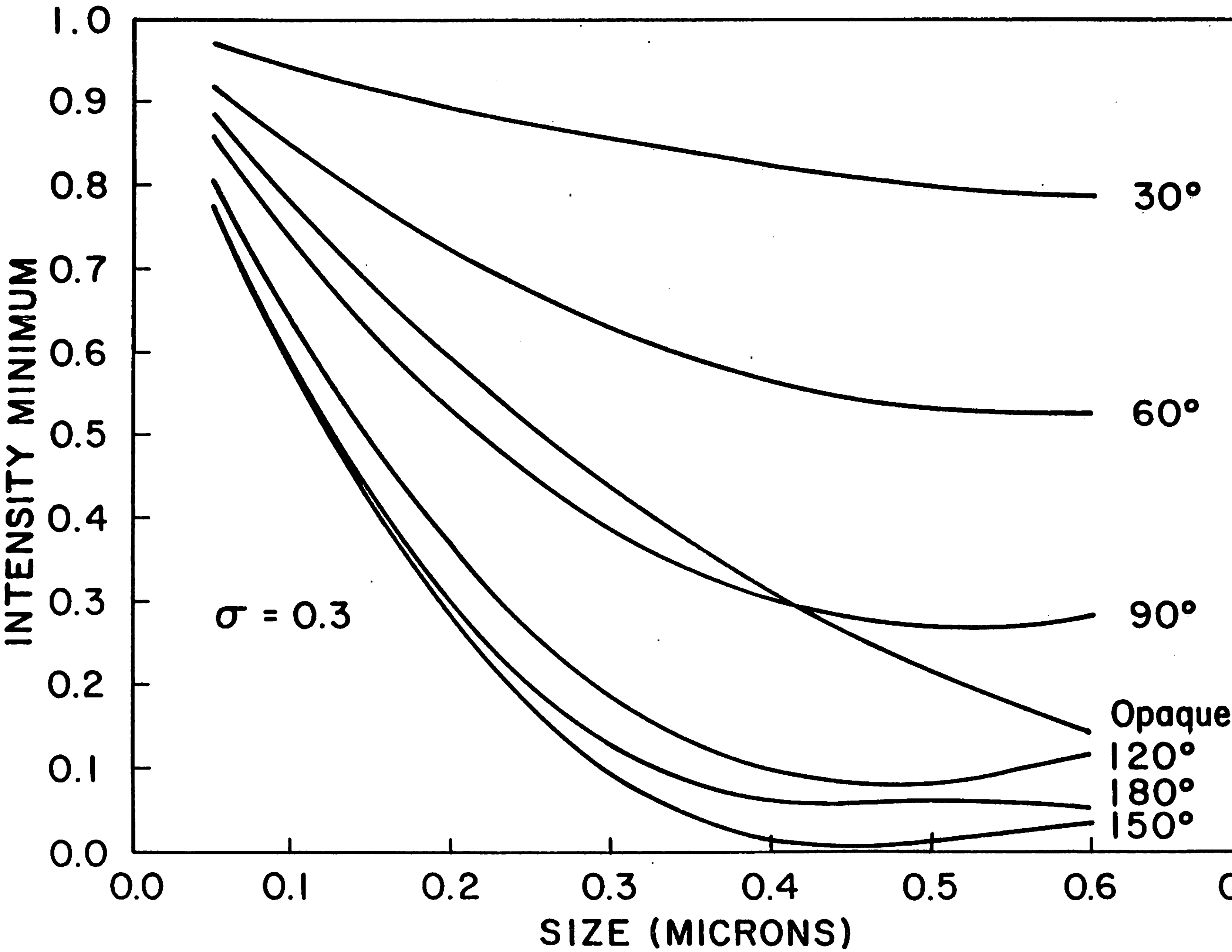


Figure 12. Intensity minimum vs size for phase shift line defects.

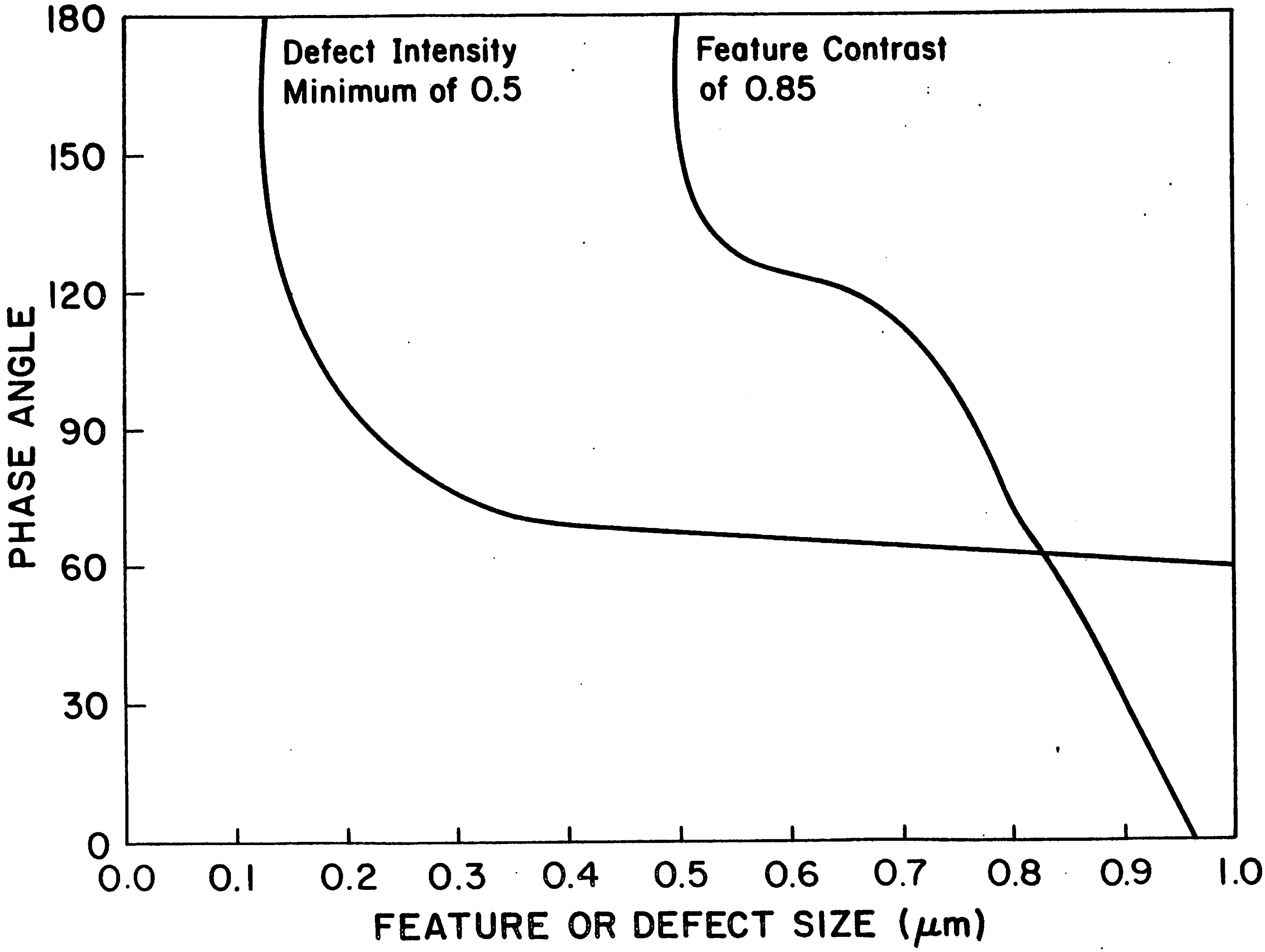


Figure 13. Phase angle for necessary feature contrast and defect minimum as a function of size. $\sigma = 0.3$.

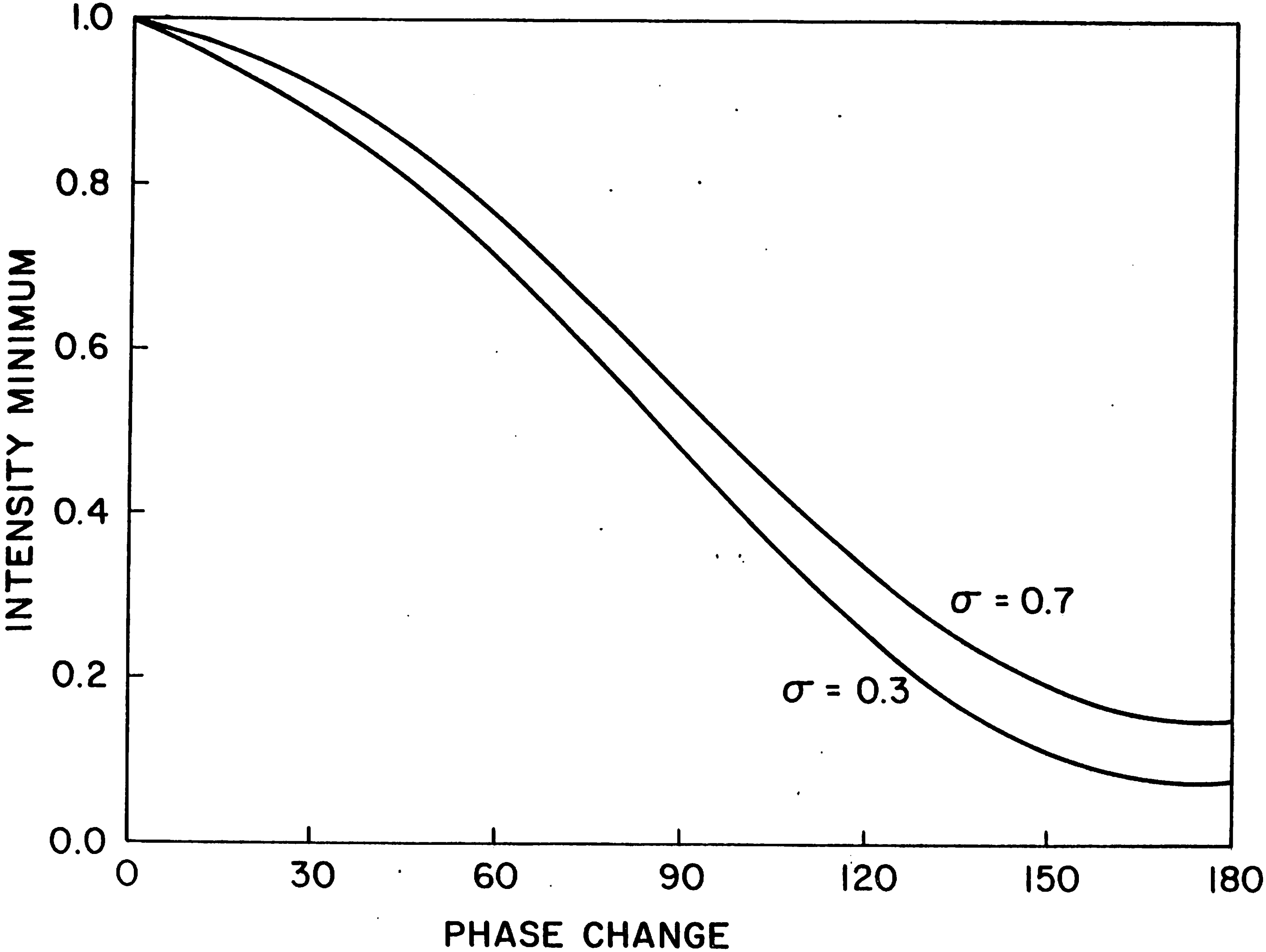


Figure 14. Minimum intensity for clear field phase transition.

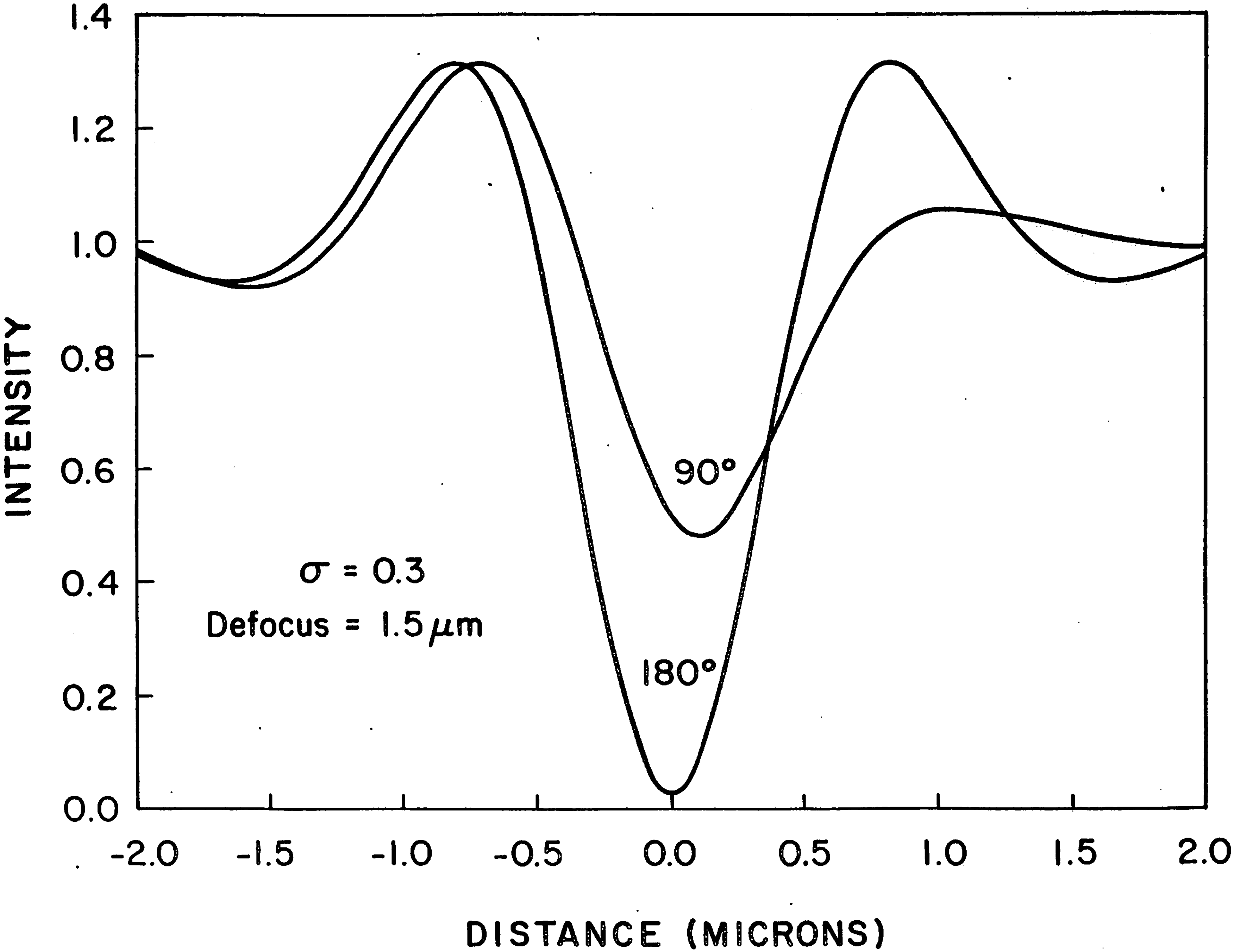


Figure 15. Image profile for clear field phase transition. Phase shifting material is on the right hand side.

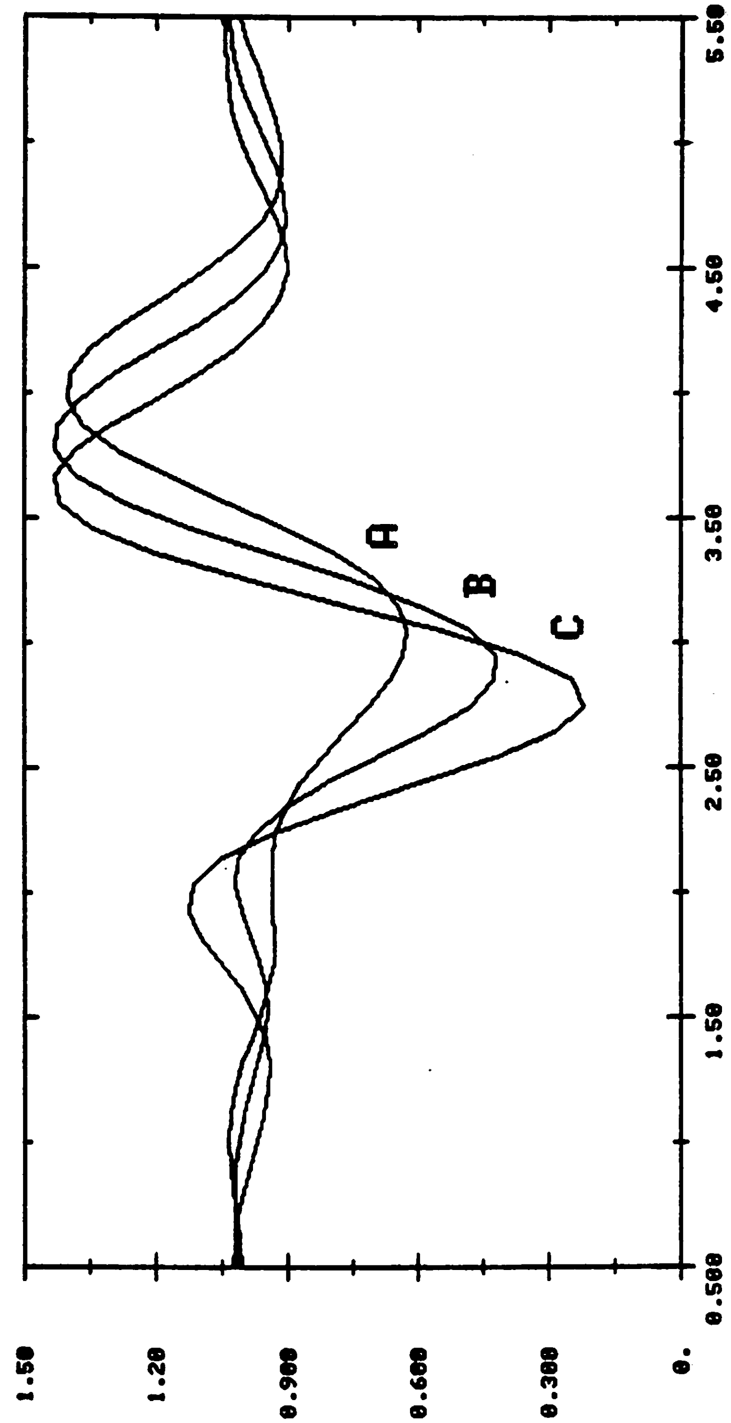


Figure 16. Image profiles for smooth 180° transition over a distance of 0.6 (A), 0.9 (B), and 1.2 (C) μm.

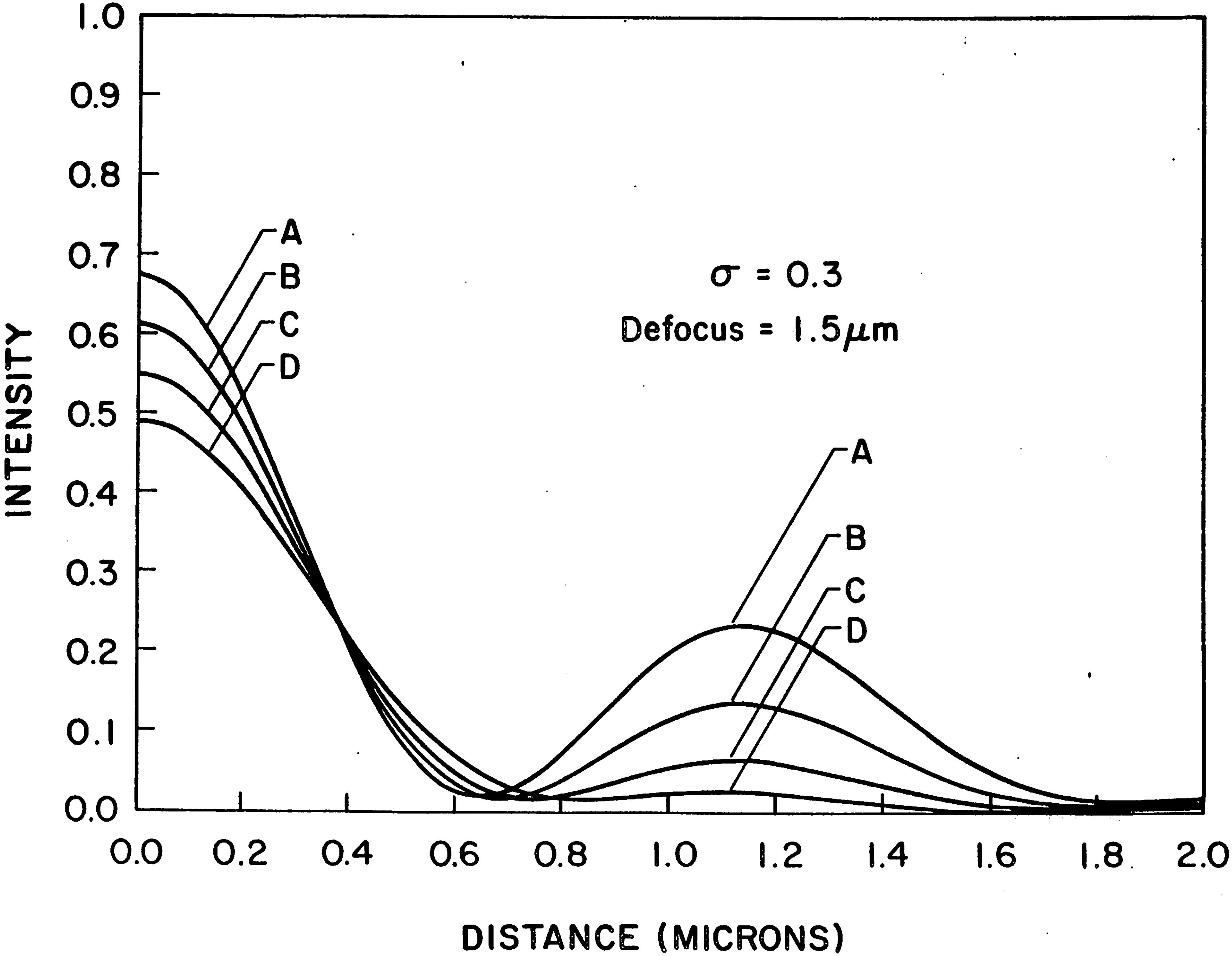


Figure 17. Image profiles for 0.82 μm line with 0.3 μm (A), 0.2 μm (B), 0.1 μm (C), and no (D) 180° phase shifted proximity feature. Center to center distance is 1.15 μm in all cases.

```
c
c
c
      subroutine clmtfi(ilmbd)
cs
c     clmtfi computes the diffraction limited fourier components for
c     the various cases.  a maximum of 41 frequencies is taken for the
c     linespace and 41 frequencies are taken for line and for space.
c     note - each horizontal point is the center of a cell.
c     (projection system only.)
c
      complex fsamsk(41),fsaimg(41)
      common /cbwind/ window, edge, wndorg
      common /fouser/ mxnmfr, nmfrcp, fsamsk(41), fsaimg(41)
      common /horimg/ deltx, mnhpts, nmhpts, horint(50)
      common /img2pr/ rna, mxnwtf, nmwtfc, rmtfwt(41)
      common /img3pr/ iincoh,iparco,ifulco,icoher,sigma,dfdist
      common /io1   / itermi,ibulk,iprout,iresv1,iin,iprint,ipunch
      common /objmsk/ mline, mspace, mlnspa, mirreg, maskty
      common /objms2/ rlw, rsw, rlw2, rsw2
      common /optic / curwl,v1,vmax,thorin(50)
      common /spectr/ mnlmbd, nmlmbd, rlambd(10), relint(10)
      common /cmask/ numreg,am(100),phi(100),dsize(100),zperid
c
c
c     /* set maximum frequency for coherent, partially coherent,
c     /* or incoherent case.
      if (icoher .eq. iincoh) vmax = 2.*rna/rlambd(ilmbd)
      if (icoher .eq. iparco) vmax = (rna/rlambd(ilmbd))*(1.+sigma)
      if (icoher .eq. ifulco) vmax = rna/rlambd(ilmbd)
c
      if (maskty .eq. mline) goto 100
      if (maskty .eq. mspace) goto 200
      if (maskty .eq. mlnspa) goto 300
      if (maskty .eq. mirreg) goto 400
      if (maskty .eq. 5     ) goto 500
      call imgmsg(2)
c
c     /* mask is a line
  100 deltx = window/float(nmhpts-1)
      wndorg = .5*rlw-edge
      nmfrcp = mxnmfr
      v1 = vmax/float(mxnmfr-1)
      rsw = 1./v1 - rlw
      call imgmsg(4)
c
c     /* calculate the fourier coifficents for the pattern
      call fourcf(.true.,rsw,rlw2,rsw2,rlw,fsamsk,mxnmfr,nmfrcp-1)
      goto 999
c
c     /* mask is a space
  200 deltx = window/float(nmhpts-1)
      wndorg = .5*rsw-edge
      nmfrcp = mxnmfr
      v1 = vmax/float(mxnmfr-1)
      rlw = 1./v1 - rsw
      call imgmsg(5)
c
c     /* calculate the fourier coeifficents for the pattern
      call fourcf(.false.,rsw,rlw2,rsw2,rlw,fsamsk,mxnmfr,nmfrcp-1)
      goto 999
```

```
c
c      /* mark is a linespace
  300 deltx = window/float(nmhpts-1)
      wndorg = .5*rlw - edge
c      /* find the fundamental frequency for the linespace case
      v1 = 1./(rlw + rsw)
c      /* find the number of frequency components that need to be taken
      itemp = int(vmax/v1)
      nmfrcp = itemp + 1
      if((vmax/v1-float(itemp)).ge..95) call imgmsg(9)
      if(nmfrcp.lt.mxnmfr) goto 350
      call imgmsg(3)
      nmfrcp = mxnmfr
c
c      /* calculate the fourier coifficents for the pattern
  350 call fourcf(.true.,rsw,rlw2,rsw2,rlw,fsamsk,mxnmfr,nmfrcp-1)
      goto 999
c
c      /* mark is a irregular pattern
  400 period = rsw+rlw+2.0*(rsw2+rlw2)
      deltx = window/float(nmhpts-1)
      wndorg = .5*rlw - edge
c      /* find the fundamental frequency for the linespace case
      v1 - 1./period
c      /* find the number of frequency componets that need to be taken
      itemp = int(vmax/v1)
      nmfrcp = itemp + 1
      if((vmax/v1-float(itemp)).ge..95) call imgmsg(7)
      if (nmfrcp.lt.mxnmfr) goto 450
      call imgmsg(3)
      nmfrcp = mxnmfr
c
c      /* calculate the fourier coeifficents for the pattern
  450 call fourcf(.true.,rsw,rlw2.rsw2,rlw,fsamsk,mxnmfr,nmfrcp-1)
      goto 999
c
  500 period = xperid
      deltx = window/float(nmhpts-1)
      wndorg = .5*dsize(1)-edge
c      /* find the fundamental frequency for the linespace case
      v1 - 1./period
c      /* find the number of frequency componets that need to be taken
      itemp = int(vmax/v1)
      nmfrcp = itemp + 1
      if((vmax/v1-float(itemp)).ge..95) call imgmsg(7)
      if (nmfrcp.lt.mxnmfr) goto 550
      call imgmsg(3)
      nmfrcp = mxnmfr
c
c      /* calculate the fourier coifficents for the pattern
  550 call cfour(fsamsk,nmfcrp-1)
c
  999 return
      end
```

```
      subroutine cfour(zcoeff,numcof)
      complex zcoeff(41)
      common /cmask/ numreg,am(100),phi(100),dsize(100),zperid
      dimension x(100),ang(100)
      x(numreg)=dsize(numreg)/2
      x(1)=dsize(1)/2
c
       do 10 i=2,numreg-1
 10   x(i)=dsize(i)
c
      p=zperid
      pi=3.1415926
      w=2*pi/p
c
c     convert degrees to radians
c
      do 20 i=1,numreg
20    ang(i)=phi(i)/360.*2*pi
c
      aimo=0.
      reo=0.
      do 30 i=1,numreg
      reo=reo+am(i)*x(i)*cos(ang(i))*2/p
30    aimo=aimo+am(i)*x(i)*sin(ang(i))*2/p
      zcoeff(1)=cmplx(reo,aimo)
c
c
      do 1000 n=1,numcof
      x1=0.
      x2=0.
      reo=0.
      aimo=0.
      do 40 i=1,numreg
      x2=x2+x(i)
      rel=2*am(i)*(sin(n*w*x2)-sin(n*w*x1))/(n*pi)
      reo=reo+rel*cos(ang(i))
      aimo=aimo+rel*sin(ang(i))
      x1=x2
40    continue
      zcoeff(n+1)=cmplx(reo,aimo)
c
1000  continue
      return
      end
c
```

```
c
c
      subroutine fourcf(lext,rs1,rl2,rs2,rl1,ai,n,numcof)
cs
c     the fourier coefficients for the mask amplitude transmission are
c     calculated.
c
c      rs1 = the first space specified for the pattern
c      rl2 = the second line in the pattern
c      rs2 = the second space in the pattern
c      rl1 = the last line specified for the pattern
c *** rs2 and rl2 are not need and should be zero for ordinary patterns
c *** if lext is specified true,then the pattern is centered around a
c     line instead around the space in resist,therefore the
c     Fourier coefficients are calculated with absolute zero the
c     center of rl1
c
      complex ai(n)
      logical lext
c
      pi=3.141593
      rper= rs1 + rl1 + 2.0 * (rs2 + rl2)
      ai(1) =  (rs1  + 2.0 * rs2) / rper
      if (lext)goto 150
          do 100 j = 1, numcof
            ai(j+1) = (2.0 / (float(j) * pi)) *
     &                (sin(float(j)*pi* rs1 / rper)  +
     &                 sin(float(j)*pi*(rs1 + 2.0*(rl2+rs2)) / rper) -
     &                 sin(float(j)*pi*(rs1 + 2.0*rl2) / rper))
 100      continue
          goto 999
 150  do 200 j = 1, numcof
          ai(j+1) =  (-2.0 / (float(j) * pi)) *
     &               (sin(float(j)*pi* rl1 / rper)  +
     &                sin(float(j)*pi*(rl1 + 2.0*(rl2+rs2)) / rper) -
     &                sin(float(j)*pi*(rl1 + 2.0*rs2) / rper))
 200  continue
 999  return
      end
c
c
```

```
c
c
      subroutine parco2(ilmbd)
cs.
c     this subr is a modified version of Mike O'Toole's subr parcoh.
c     It uses the values of rlw and rsw calculated in subr image
c     to calculate the Fourier coefficients for the mask amplitude
c     transmission (extended as an even periodic function if necessary)
c     and places them in the array ai(41). It then calls subr cross
c     which calculates the Fourier coefficients for the image intensity
c     pattern and leaves them in the array spec(81)
c     the common blocks /trans/ contains ai(41), spec(81), s (sigma),
c     n (the number of spatial frequency components (at most 40)),
c     and df (normalised defocus).
c
      complex fsamsk(41),fsaimg(41)
      complex ai(41),spec(81)
      common /cbwind/ window,edge,wndorg
      common /fouser/ mxnmfr, nmfrcp, fsamsk(41), fsaimg(41)
      common /horimg/ deltx, mnhpts, nmhpts, horint(50)
      common /imgflg/ imgfl(5)
      common /img2pr/ rna,mxnwtf,nmwtfc,rmtfwt(41)
      common /img3pr/ iincoh,iparco,ifulco,icoher,sigma,dfdist
      common /io1   / itermi,ibulk,iprout,iresv1,iin,iprint,ipunch
      common /optic / curwl,v1,vmax,thorin(50)
      common /spectr/ mnlmbd, nmlmbd, rlambd(10), relint(10)
      common /trans / s,n,p,df,ai(41),spec(81)
      common /objmsk/ mline, mspace, mlnspa, mirreg, maskty
      common /objms2/ rlw, rsw, rlw2, rsw2
      common /cmask/ numreg,am(100),phi(100),dsize(100),zperid
c
      s=sigma
      if (imgfl(4).eq.1) write (iprint, 1111)
 1111 format(///,9x, 43hno diagnostics are available for shankar-s ,
     *               27hpartial coherence routines. )
c
      pi = 3.14159265358
      rper = rsw+rlw+2.0*(rsw2+rlw2)
      if (maskty.eq.5) rper = zperid
      p = rlambd(ilmbd)/(rna*rper)
      df = 2.*pi*dfdist*rna*rna/rlambd(ilmbd)
      n = int((1.+s)/p)
      if (n .lt. 40) goto 100
        n=40
```

```
c
c      /* (why is there no check for n<=0?   march 25, 1981)
c      /* the Fourier coefficients for the mask amplitude transmission
c      /* are now calculated.
100    if (masktу.eq.5) call cfour(ai,n)
       if (maskty.eq.5) goto 101
       call fourcf(.false.,rsw,rlw2,rsw2,rlw,ai,41,n)
101    call cross
       origin=(-rsw/2 -edge)/rper
       if(maskty.eq.mspace)origin=(rsw/2.-edge)/rper
       if(maskty.eq.mirreg)origin=((rper+rlw)/2.0-edge)/rper
       if(maskty.eq.5)  origin = (dsize(1)/2-edge)/rper
       x=origin
       n2  =2*n
       delty=window/((nmhpts-1)*rper)
c
c
c
c
       do 150 j=1,n2
150    continue
c
c
c
       do 300 j=1,nmhpts
         thorin(j)=spec(1)
         do 200 k=1,n2
           zmag=real(spec(k+1))
           thorin(j)=thorin(j)+zmag*cos(2.*pi*k*x+zphas)
  200    continue
         x=x+delty
  300  continue
c      /* accumulate the intensity in horint.
       do 400 ihpt=1,nmhpts
         horint(ihpt) = horint(ihpt) + relint(ilmbd)*thorin(ihpt)
  400  continue
c
       return
       end
c
c
```

```
c
c
      subroutine cross
cs
      common /iapfl / iapert,isquar,icirc
      common /trans/s,n,p,df,ai(41),spec(81)
      common /fig/f,g
      complex ai(41),spec(81)
      complex fac
      external gc1,gt1,gt2,ft2,ft3
      external gc1i,gt1i,gt2i,ft2i,ft3i
c
c     /* set all elements of spec to zero
      do 1 j=1,81
        spec(j)=(0.0,0.0)
    1 continue
      spec(1)=ai(1)*conjg(ai(1))
      pi=3.14159265358
      pi2=pi/2.
c     /* circular and square pupils are handled separately
c??   /* the following line was present in a previous version.
c??   /* why? (march 31, 1981).
c??   if (n.eq.0) return
      if (iapert.eq.isquar) goto 66
      if (iapert.ne.icirc) return
c     /* set amax, the max. area of overlap
      amax = s*s*pi
      if (s.gt.1.) amax=pi
c     /* for circular pupils s>1 and s<1 are handled separately
      if (s .gt. 1.) goto 41
c     /* for circular pupils: C is a circle of radius s at the origin,
c     /* C' and C" are circles of unit radius. C' is centred at f. For
c     /* t C" is centred at O, for t1 at -g and for t2 at +g.  The
c     /* region of integration is the intersection of these circles.
      do 3 j = 1,n
      f=j*d
      t=0.
      ti=0.
c     /* for t: centre C" at O by setting g=0
      g=0.
c     /* set the limits of integration over C and over C'.
      a1 = f-1.
      a2 = ((s*s-1.)/f + f)/2.
      a3 = s
      if(f.gt.(1.-s))goto 21
c
c if C isn't inside C' goto 21. if it is inside integrate over C only
      call gauss(gc1,0.,pi2,t)
c     /* normalise t by dividing by amax
      t=2.*t/amax
      call gauss(gc1i,0.,pi2,ti)
      ti=2.*ti/amax
      goto 5
c the integrand has been transformed by u = cos(x). Transform the limits
   21 ac1 = acos(f-a1)
      ac2 = acos(f-a2)
      ac3 = acos(a3/s)
```

```
c??   /* why doesn't the stmt for ac3 have the bias of 0.0001 ?
      ac4 = acos(a2/(s+0.0001))
      call gauss(gt1,ac1,ac2,a1t)
      call gauss(gt2,ac3,ac4,a2t)
      t = (a1t +a2t)*2./amax
      call gauss(gt1i,ac1,ac2,a1ti)
      call gauss(gt2i,ac3,ac4,a2ti)
      ti = (a1ti +a2ti)*2./amax
    5 spec(i+j)=conjg(ai(1))*ai(j+1)*2.*cmplx(t,ti)
    3 continue
c     /* the nested do loops that follow make a single sum series out
c     /* of the double sum series
      do 4 j=1,n
      f=j*p
      do 4 k=1,j
      g=k*p
      t1 =0.
      t2 =0.
      t1i =0.
      t2i =0.
c     /* calculate t1 and t2. in general integrate over three
c     /* circles. in some cases the limits of integration are
c     /* are the same for an integral. this is because the
c     /* region of integration does not include that area.
c     /* this depends on the values of f and g.
c     /* first t1
      a1t1 =0.
      a2t1 =0.
      a3t1 =0.
      a1t1i =0.
      a2t1i =0.
      a3t1i =0.
c     /* if (f+g).ge 2 C' and C" don't intersect and t1=0.
      if((f+g).ge.2 )goto 25
c     /* set the limits for integrating over C',C and C" for t1
      a1 = f-1.
      a2 = ((s*s-1.)/f + f)/2.
      if(f.gt.(1.-s))goto 22
c     /* if f.le.(1-s) there's no integration over C',change limits and
c     /* out a1=a2 so that the integral over C' is zero
      a1 = -s
      a2 = -s
   22 a3 = -((s*s-1.)/g +g)/2.
      a4 = 1. -g
      if(g.gt.(1.-s))goto 23
c     /* if g.le.(1.-s) there's no integration over C",change limits
c     /* and put a3=a4 so that the integral over C" will be zero
      a3  = s
      a4 = s
c     /* if C is inside C' & C" integrate over C only.Goto 28 for that.
      if((f.le.(1.-s)).and.(g.le.(1.-s)))goto 28
   23 if((1.-f*g).gt.s*s)goto 24
c     /* if (1 -f*g).le.s*s the intersection of C' & C" is inside C.
c     /* No integration over C.  Set a2=a3 so that integral over C will
```

```
c      /* be zero
       a2 = (f-g)/2.
       a3 = a2
c      /* transform limits
   24  ac1 = acos(f-a1)
       ac2 = acos(f-a2)
       ac3 = acos(a3/s)
c      /* s is increased by 0.0001 to keep the argument to acos .le. 1 .
c??    /* (march 31, 1981) why does the above stmt for ac3 not have it?
       ac4 = acos(a2/(s+0.0001))
       ac5 = acos(a4 +g)
       ac6 = acos(a3 + g)
c      /* integrate over C',C & C"
c
c      gt1 calculates the integral over C'.  The real part of the
c           integral over C'' uses the same function, but the
c           imaginary part uses the negation.  Hence the switching
c           of the limits of integration in calculating the imaginary
c           part of the integral over C''.
c                                                         MDP
c
       call gauss(gt1,ac1,ac2,a1t1)
       call gauss(gt2,ac3,ac4,a2t1)
       call gauss(gt1,ac5,ac6,a3t1)
       t1 =(a1t1 + a2t1 + a3t1)/amax
       call gauss(gt1i,ac1,ac2,a1t1i)
       call gauss(gt2i,ac3,ac4,a2t1i)
       call gauss(gt1i,ac6,ac5,a3t1i)
       t1i =(a1t1i + a2t1i + a3t1i)/amax
       goto 25
   28  call gauss(gc1,0.,pi2,t1)
       t1 = t1/amax
       call gauss(gc1i,0.,pi2,t1i)
       t1i = t1i/amax
c      /* here calculate t2
c      /* if (f-g).ge 2 C' & C" don't intersect.
   25  if((f-g).ge.2 )goto 10
c      /* C' is centred at f, C" at g.  g .le. f so if f.le.(1.-s) C is
c      /* inside both C' & C".  In all cases there's no integral over C".
       if (f.le.(1.-s)) goto 29
c      /* if f.le.(1.-s) C is inside C' & C".
       a1t2=0.
       a2t2=0.
       a3t2=0.
       a1t2i=0.
       a2t2i=0.
       a3t2i=0.
       a1 = f-1.
       a2 = ((s*s-1.)/f + f)/2.
       if(f.gt.(1.-s))goto 26
       a1 = -s
       a2 = -s
```

```
 26 a3 =s
    ac1 = acos(f-a1)
    ac2 = acos(f-a2)
c?? /* (march 31, 1981) again why no bias of 0.0001 to s for ac3 ?
    ac3 =acos(a3/s)
    ac4 = acos(a2/(s+0.0001))
c   /* the same functions that were used as integrands for t1 are
c   /* used here. since C" is now centred at g and not at -g put g=-g
c   /* before integrating and change g back to it's previous value by
c   /* g=-g
    g=-g
    call gauss(gt1,ac1,ac2,a1t2)
    call gauss(gt2,ac3,ac4,a2t2)
    call gauss(gt1i,ac1,ac2,a1t2i)
    call gauss(gt2i,ac3,ac4,a2t2i)
    g=-g
    t2 = (a1t2 + a2t2)/amax
    t2i = (a1t2i + a2t2i)/amax
    goto 10
 27 g=-g
    call gauss(gc1,0.,pi2,t2)
    t2 = t2/amax
    call gauss(gc1i,0.,pi2,t2i)
    t2i = t2i/amax
    g=-g
 10 fac=ai(j+1)*conjg(ai(k+1))/2.
    t1 =2.*t1
    t2 = 2.*t2
    t1i =2.*t1i
    t2i = 2.*t2i
    m1=j+k
    m2=j-k
    if (m2.eq.0) goto 113
    t1=2.*t1
    t2=2.*t2
    t1i=2.*t1i
    t2i=2.*t2i
113 spec(1+m1)=spec(1+m1)+fac*cmplx(t1,t1i)
    spec(1+m2)=spec(1+m2)+fac*cmplx(t2,t2i)
  4 continue
    return
c   /* what follows is for a circular pupil with s>1
 41 do 2 j=1,n
    f=j*p
    g=0.
    t=0.
    ti=0.
    if(f.ge.2.)goto 2
c   /* integrate over C' int C"; C" centred at 0.
    ac1 =acos(f/2 )
    call gauss(gt1,0.,ac1,t)
    t=4.*t/pi
    call gauss(gt1i,0.,ac1,ti)
    ti=4.*ti/pi
    spec(1+j)=conjg(ai(1))*ai(j+1)*2.*cmplx(t,ti)
  2 continue
    do 14 j=1,n
    f=j*p
    do 14 k=1,j
    g=k*p
```

```
c       /* finding t1 and t2
        t1 =0.
        t2 =0.
        a1t2 =0.
        a2t2 =0.
        a3t2 =0.
        t1i =0.
        t2i =0.
        a1t2i =0.
        a2t2i =0.
        a3t2i =0.
        if ((f+g).ge.2. ) goto 11
c       /* checked to see if C' and C" intersect; if not goto 11 and
c       /* calculate t2
        ac1 = acos((f+g)/2. )
        call gauss(gt1,0.,ac1,t1)
        t1 = 2.*t1/pi
        t1i = 0.
c       /* calculate t2 here
    11  if((f-g).ge.2 )goto 100
        if((g+1.).gt.s)goto 12
c       /* int C' C" is inside C
        ac1=acos((f-g)/2.)
        call gauss(ft2,0.,ac1,t2)
        t2 = 2.*t2/pi
        t2i = 0.
        goto 100
c       /* int C' C" is not all inside C
    12  a2 = (f+g)/2.
        a3 = ((s*s-1.)/g +g)/2.
        if(a2 lt.a3)goto 13
        a2=((s*s-1.)/f +f)/2.
        a3 = a2
    13  ac1 = acos(f-a2)
        ac2 = acos(a2-g)
c??     (march 31, 1981) again why no bias of 0.0001 to s for ac3 ?
        ac3 = acos(a3-g)
        ac4 = acos(a3/(s+0.0001))
        call gauss(ft2,0.,ac1,a1t2)
        call gauss(ft2,ac3,ac2,a2t2)
        call gauss(ft3,0.,ac4,a3t2)
        t2 =(a1t2 + a2t2 + a3t2)/pi
        call gauss(ft2i,0.,ac1,a1t2i)
        call gauss(ft2i,ac2,ac3,a2t2i)
        call gauss(ft3i,0.,ac4,a3t2i)
        t2i =(a1t2i + a2t2i + a3t2i)/pi
   100  fac=ai(j+1)*conjg(ai(k+1))/2.
        t1 =2 *t1
        t2 = 2.*t2
        t1i =2.*t1i
        t2i = 2.*t2i
        m1=j+k
        m2=j-k
        if(m2 eq.0)goto 114
        t1=2.*t1
        t2=2.*t2
        t1i=2 *t1i
        t2i=2.*t2i
```

```
  114 spec(1+m1)=spec(1+m1)+fac*cmplx(t1,t1i)
      spec(1+m2)=spec(1+m2)+fac*cmplx(t2,t2i)
   14 continue
      return
c     /* the rest is for square pupils
   66 s2 =s
      if(s.gt.1.) s2 =1.
      amax= 4.*s2*s2
      do 82 j=1,n
      f=j*p
      t=0.
      if(f.ge.2.)goto 51
c     /* set the limits of integration a1 and a2
      a1=f-1.
      a2 = s2
      if((s.lt.1.).and.(f.lt.(1.-s)))a1=-s
c     /* if df is zero the integral for t is different.
      if(df.eq.0.)goto 50
      t = 2 *s2*(sin(df*f*(a2-f/2.)) -sin(df*f*(a1-f/2.)))/(df*f*amax)
      goto 51
   50 t = 2 *s2*(a2-a1)/amax
   51 spec(1+j)=ai(1)*ai(j+1)*2.*t
   82 continue
      do 84 j=1,n
      f=j*p
      do 84 k=1,j
      g=k*p
c     /* calculate t1 and t2
      t1 = 0.
      if((f+g).ge.2.)goto 52
c     /* at 52 find t2
      a1 = f-1.
      a2 =1.-g
      if (s.ge.1.) goto 53
      if (f.le.(1.-s)) a1=-s
      if (g.le.(1.-s)) a2= s
   53 if (df.ne.0.) t1 = 2.*s2*(sin(df*(f+g)*(2.*a2-f+g)/2.)
     *         -sin(df*(f+g)*(2.*a1-f+g)/2.))/(df*(f+g)*amax)
      if (df.eq.0.) t1 = 2.*s2*(a2-a1)/amax
c     /* calculate t2
   52 t2=0.
      if((f-g).ge.2.)goto 54
      a1 =f-1.
      if((f.lt.(1.-s)).and.(s.lt.1.))a1=-s
      a2 =g+1.
      if(a2 gt.s)a2 = s
      if(s.lt.1.)a2 = s
      if((df.ne.0.).and.(f.ne.g))t2=2.*s2*(sin(df*(f-g)*(2.*a2-f-g)/2.)
     *  -sin(df*(f-g)*(2.*a1-f-g)/2.))/(df*(f-g)*amax)
      if((df.eq.0.).or.(f.eq.g))t2 = 2.*s2*(a2-a1)/amax
      if((f+g).ge.2.)t1=0.
   54 fac=ai(j+1)*ai(k+1)/4.
      m1=j+k
      m2=j-k
      if(m2 eq.0)goto 143
      t1=2.*t1
      t2=2.*t2
```

```fortran
  143 spec(1+m1)=spec(1+m1)+fac*2.*t1
      spec(1+m2)=spec(1+m2)+fac*2.*t2
      t1=0.
      t2 = 0.
   84 continue
      return
      end
      function ft2(x)
cf
      common /fig/f,g
      common /trans/s,n,p,df,ai(41),spec(81)
      complex ai(41),spec(81)
c
      ft2 = 0.5*(1.-cos(2 *x))*cos(df*(f-g)*(2.*cos(x)-f+g)/2.)
      return
      end
      function ft3(x)
cf
      common /fig/f,g
      common /trans/s,n,p,df,ai(41),spec(81)
      complex ai(41),spec(81)
c
      ft3 = 0.5*s*s*(1.-cos(2.*x))*cos(df*(-f+g)*(2.*s*cos(x)-f-g)/2.)
      return
      end
      function gc1(x)
cf
      common /trans/s,n,p,df,ai(41),spec(81)
      common /fig/f,g
      complex ai(41),spec(81)
c
      gc1 = s*s*(1.-cos(2.*x))*cos(.5*df*(f*f-g*g))*cos(df*(f+g)*s*
     *            cos(x))
      return
      end
      function gt1(x)
cf
      common /trans/s,n,p,df,ai(41),spec(81)
      common /fig/f,g
      complex ai(41),spec(81)
c
      gt1 = 0.5*(1.-cos(2.*x))*cos(0.5*df*(f+g)*(2.*cos(x)-f-g))
      return
      end
```

```
      function gt2(x)
cf
      common /trans/s,n,p,df,ai(41),spec(81)
      common /fig/f,g
      complex ai(41),spec(81)
c
      gt2 = 0.5*s*s*(1.-cos(2.*x))*cos(0.5*df*(f+g)*(2.*s*cos(x)-f+g))
      return
      end
      function ft2i(x)
cf
      common /fig/f,g
      common /trans/s,n,p,df,ai(41),spec(81)
      complex ai(41),spec(81)
c
      ft2i = 0.5*(1.-cos(2.*x))*sin(df*(f-g)*(2.*cos(x)-f+g)/2.)
      return
      end
      function ft3i(x)
cf
      common /fig/f,g
      common /trans/s,n,p,df,ai(41),spec(81)
      complex ai(41),spec(81)
c
      ft3i = 0.5*s*s*(1.-cos(2.*x))*sin(df*(-f+g)*(2.*s*cos(x)-f-g)/2.)
      return
      end
      function gc1i(x)
cf
      common /trans/s,n,p,df,ai(41),spec(81)
      common /fig/f,g
      complex ai(41),spec(81)
c
      gc1i = s*s*(1.-cos(2.*x))*sin(.5*df*(f*f-g*g))*cos(df*(f+g)*s*
     *             cos(x))
      return
      end
      function gt1i(x)
cf
      common /trans/s,n,p,df,ai(41),spec(81)
      common /fig/f,g
      complex ai(41),spec(81)
c
      gt1i = 0.5*(1.-cos(2.*x))*sin(0.5*df*(f+g)*(2.*cos(x)-f-g))
      return
      end
      function gt2i(x)
cf
      common /trans/s,n,p,df,ai(41),spec(81)
      common /fig/f,g
      complex ai(41),spec(81)
c
      gt2i = 0.5*s*s*(1.-cos(2.*x))*sin(-0.5*df*(f+g)*(2.*s*cos(x)-f+g))
      return
      end
```