Modeling the *MOSES* Wavefront Error

Charles C. Kankelborg

July 16, 2009

Abstract

This paper describes a detailed model of the *MOSES* sounding rocket instrument using Fourier optics. The model is suitable for Monte Carlo modeling of system performance based on individual element specifications. The theory is developed from first principles.

1 Introduction

The *Multi-Order Solar Extreme Ultraviolet Spectrograph* (*MOSES*) is a rocket-borne instrument capable of simultaneous imaging and spectroscopy at high spatial resolution over a wide field, with high spectral resolution in a narrow band. This is accomplished by using a multilayer coated, concave diffraction grating to form images at three spectral orders, $m = -1, 0, +1$. Over the narrow multilayer passband, these spectral orders are widely separated. An oblong, multilayer coated flat mirror folds all three beams back to the corresponding detectors. A monochromatic, infinite-conjugate point source is imaged as a spot on each of the three detectors. The position of each spot depends on wavelength in a way that is proportional to m and to the grating dispersion. For an extended object such as the Sun, the image formed on each detector combines spatial and spectral information in a different way.

This study concerns the image forming properties of the *MOSES* optics. There is significant astigmatism in all three spectral orders, especially $m = \pm 1$. This is evident in the wavefront error and point spread functions for *MOSES*, which I have derived using a simple Fourier optics approach. The implementation of this model, in IDL, is capable of finding best focus, and can take into account known (real or hypothetical) errors in polishing the optics and ruling the grating. These features should make it possible ultimately to ingest interferogram data into the program and predict the best focus in EUV. The three detectors are mounted to a common structure, with shims to fix their relative positions. Since the central order has the same best focus regardless of wavelength, the focus can be accomplished by moving all the detectors in tandem.

The speed of the fast Fourier transform (FFT) makes the code suitable for Monte

Carlo modeling. The idea is to set optical specifications based on an assumed PSD for each component, produce many random realizations of the entire instrument based on the spec, and characterize the resulting performance statistically.

2 *MOSES* Optical Design

We begin with instrument optical coordinates (X, Y, Z) with the origin at the center of the surface of the *MOSES* $m = 0$ detector. All dimensions are in mm. The optic axis is along X toward the Sun. Grating dispersion is in the Y direction. The Z axis is perpendicular to the Lockheed Optical Table System (LOTS). In that coordinate system, the *MOSES* optical prescription is given by the Roger Thomas design memo of June 20, 2009. Table 1 differs from the June 20 memo in one respect: the naming of the plus and minus spectral orders has been reversed (m increases monotonically with y).

Next, we define new coordinates (x, y, z) for the purpose of this calculation, with the origin at the center of the grating surface and the optical system "unfolded" —without the fold flat, so that the detectors are now almost 5 m from the grating. The coordinate axes keep their same orientation with respect to the primary mirror. The details are worked out in Appendix A. GNU Octave source code implementing the coordinate transforms is given in Appendix A.1. Table 2 gives the locations of all optical elements in the new coordinates. The center of curvature of the spherical grating is at

$$\begin{aligned}(x_0, y_0, z_0) &= R \cdot [\cos(0.973^\circ), 0, -\sin(0.973^\circ)] \\ &= [9478.63, 0, 160.98].\end{aligned} \tag{1}$$

3 Ideal *MOSES*

We begin by modeling *MOSES* as designed. The design has aberrations, which we derive by calculating the wavefront error (WFE) directly. The performance of the system is characterized by calculating point spread and modulation transfer functions from the WFE.

3.1 WFE Calculation

Light is incident on the grating at point $\mathbf{r}$ from an infinite-conjugate source on the x axis (Figure 1). The grating surface—by which I mean the design figure, not the groove profile—is specified by the function $x(y, z)$. The WFE model for imaging at a field point $\mathbf{f}$ is developed by calculating the phase of ray paths connecting the source to $\mathbf{f}$ through $\mathbf{r} = [x(y, z), y, z]$. The phase, in radians, for spectral order m as a function of grating surface coordinates is

$$\phi_m(y, z) = \phi_I + \phi_G + \phi_O. \tag{2}$$

The three terms correspond to Incident, Grating, and Object:

$$\phi_I = -k\, x(y, z), \text{ propagation to } \mathbf{r}, \tag{3}$$

$$\phi_G = 2\pi \frac{my}{d}, \text{ diffraction grating}, \tag{4}$$

$$\phi_O = k\, |\mathbf{f} - \mathbf{r}|, \text{ from } \mathbf{r} \text{ to } \mathbf{f}, \tag{5}$$

Table 1: *MOSES* Design Table. Coordinates in mm.

Item	Surface	Width	Height	X	Y	Z	Pitch	Yaw
Pupil	Flat	94.0	87.0	2518.7*	0.0	78.5	0.000	0.000
Stop	Flat	80.0	80.0	118.7	0.0	78.5	+0.973	0.000
Grating	R=9480	90.0	90.0	118.7	0.0	78.5	+0.973	0.000
Folding	Flat	192.6	50.0	2428.6	0.0	0.0	+0.973	0.000
Det(-1)	Flat	28.0	14.0	3.0	-136.7	0.0	-0.011	+2.463
Det(0)	Flat	28.0	14.0	0.0	0.0	0.0	-0.004	0.000
Det(+1)	Flat	28.0	14.0	3.0	+136.7	0.0	-0.011	-2.463

* Pupil X-location is somewhat flexible, but cannot be much larger than the value given without vignetting, unless its size is increased which reduces the optical beam clearance.

Table 2: *MOSES* WFE Simulation Coordinates

Item	Surface	Width	Height	x	y	z
Grating	R=9480	90.0	90.0	0.0	0.0	0.0
Folding	Flat	192.6	50.0	2309.9	0.0	-78.5
Det(-1)	Flat	28.0	14.0	4734.1	-136.7	-160.88
Det(0)	Flat	28.0	14.0	4737.1	0.0	-160.99
Det(+1)	Flat	28.0	14.0	4734.1	+136.7	-160.88

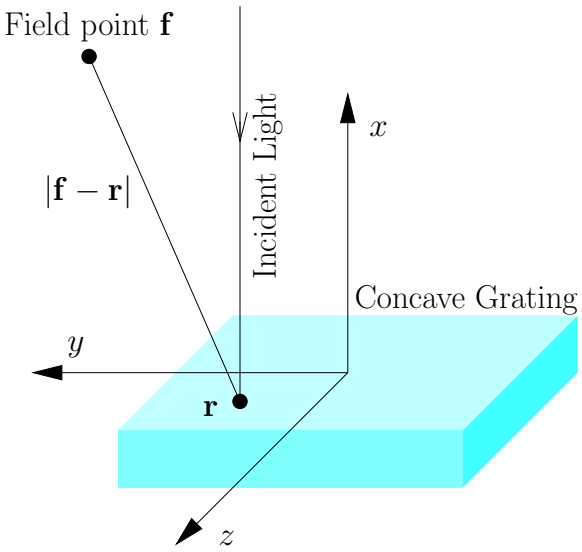

Figure 1: Calculating the phase of diffracted light in model coordinates (x, y, z). Light incident on the grating at point $\mathbf{r}$ is diffracted to point $\mathbf{f}$ (not to scale).

where $k = 2\pi/\lambda$ is the wavenumber of the incident light and d is the groove spacing of the grating. The grooves are assumed parallel to x and uniformly spaced in y. The field at point $\mathbf{f}$ is the sum of contributions over the entire mirror,

$$E(\mathbf{f}) \propto \int_M dy\, dz\, E_0(y,z) e^{i\phi_m(y,z)}, \quad (6)$$

where $E_0(y, z)$ is the amplitude of the incident light.

3.1.1 Physical Interpretation of ϕ_m

If $\phi_m(x, y)$ were constant over the whole aperture, then every ray path would add constructively at the assumed focus f. A nonzero constant offset of ϕ_m is called piston, and has neither physical significance nor mathematical effect, as it is only the phase of the wave at the focus. To the extent that ϕ_m varies, the system has wavefront error. The error can arise from something as simple as putting the focal point $\mathbf{f}$ at the wrong place (tip, tilt or defocus). The design itself, as described above, has aberration. In other words, there is no point $\mathbf{f}$ for which ϕ_m is constant. The "best focus" only minimizes the variance. Additional terms can be added to ϕ_m account for wavefront error arising from groove irregularities and imperfect figure of the grating or of the fold flat.

3.2 Fourier Optics Model

In this section we briefly derive a Fourier optics model of *MOSES* from first principles. In this context, the only thing that sets *MOSES* apart from any other imaging system is the specific model that determines the wavefront error (equations 2-5). In particular the grating term, ϕ_G, allows us to model diffraction in a specific spectral order m without resolving the groove pattern. Thus, the development that follows is generic to any imaging system. The development of the underlying diffraction theory is left to Appendix B.

We now consider the formation of a spot image in the vicinity of the putative focal point $\mathbf{f}$. This neighborhood is accessed by making small perturbations, $\mathbf{f}' = \mathbf{f} + \boldsymbol{\delta}$. The

phase from $\mathbf{r}$ to $\mathbf{f}'$ is

$$\begin{aligned}\phi'_O &= k\ |\mathbf{f}+\boldsymbol{\delta}-\mathbf{r}| \\ &= k\sqrt{|\mathbf{f}-\mathbf{r}|^2 - 2\boldsymbol{\delta}\cdot(\mathbf{f}-\mathbf{r})} \\ &\quad + O\left(\delta^2\right).\end{aligned} \tag{7}$$

The image is formed in a plane perpendicular to $\mathbf{f}$, which is the vector of the principal ray. We therefore set $\boldsymbol{\delta}\cdot\mathbf{f} = 0$. Neglecting terms of order δ^2 and higher,

$$\phi'_O = k\ |\mathbf{f}-\mathbf{r}| - \frac{k\,\boldsymbol{\delta}\cdot\mathbf{r}}{|\mathbf{f}-\mathbf{r}|}. \tag{8}$$

Note that $f_y, f_z \gg f_x$ and $|\mathbf{f}| \gg |\mathbf{r}|$. Keeping only the largest of the terms that depend on δ,

$$\phi'_O = \phi_O - k\,\frac{\delta_y y + \delta_z z}{f}, \tag{9}$$

where $f \equiv |\mathbf{f}|$. To simplest approximation, therefore, displacements in the focal plane merely add phase contributions proportional to the displacement angles δ_y/f and δ_z/f.

In Appendix B, equation 35 expresses the wave function in terms of a surface integral over an incident beam. The (non-normalized) wave function in the *MOSES* focal plane is

$$E = \int_M dy\,dz\,E_0\,e^{ik\phi_m}\,e^{-ik(\delta_y y+\delta_z z)/f}. \tag{10}$$

We can express this in terms of the two-dimensional Fourier transform of the phase exponential:

$$E(v,w) = \mathfrak{F}_{y,z}\left[E_0(y,z)\,e^{ik\phi_m(y,z)}\right]. \tag{11}$$

The variables v and w are the Fourier conjugates to y and z, respectively. They correspond directly to coordinates measured relative to point $\mathbf{f}$:

$$\begin{aligned}\delta_y &= \frac{fv}{k}, \\ \delta_z &= \frac{fw}{k}, \\ \delta_x &= -\frac{\delta_y f_y + \delta_z f_z}{f_x}\end{aligned} \tag{12}$$

Given the wavefunction E in the focal plane, it is trivial to calculate standard performance metrics such as the point spread function, line spread function, and 1D modulation transfer function:

$$P = |E|^2, \qquad \text{PSF;} \tag{13}$$

$$L(v) = \int P\,dw, \qquad \text{LSF;} \tag{14}$$

$$M = \left|\widetilde{L}\right|^2, \qquad \text{1D MTF.} \tag{15}$$

The practical motivation of the Fourier optics model is that it can be computed rapidly using the fast Fourier transform (FFT). The next two sections briefly describe an implementation of the *MOSES* Fourier optics model in IDL.

3.3 Implementation—General

The *WFEmodeling library* comprises the general-purpose Fourier optics codes listed in table 3. The source codes are all provided in Appendix C. As the codes are well-documented, they will not be discussed in detail here. However, some general comments are in order. The wavefront error (WFE) is represented numerically as a rectangular

Table 3: A suite of general-purpose Fourier optics codes in IDL. Source codes are in Appendix C. The first group is for generating and modifying WFE maps; the second group is for analyzing them and calculating performance metrics.

Name	Purpose
`psd2wfe`	Create WFE based on assumed PSD.
`wfe2psd`	Estimate PSD based on WFE.
`mirror`	WFE of circular or annular aperture based on PSD.
`spiders`	Create a Cassegrain aperture mask.
`fit_spherical_wave`	Subtract tip, tilt, piston and focus from WFE.
`defocus`	Modify WFE map to simulate a focus change.
`wfe2psf`	Convert WFE to PSF.
`psf2mtf`	Calculate 1D MTF (any orientation).
`psf2stats`	Calculate moments of PSF.
`wfe2strehl`	Calculate Strehl ratio from WFE.
`wfe2fringes`	Make a fringe image, as if by interferometry.

matrix. All of the routines support non-rectangular apertures by allowing the phase to be complex. Obscured areas of the rectangle are indicated by a large, positive real component. This effectively zeroes the complex exponential in the wave function. More generally, real values can be used to represent a specific distribution of incident light on the aperture ($E_0(y, z)$). In conformance with typical practice in the optics community, WFE is represented in waves rather than in radians as in the above mathematical development. Three test programs not listed in the table (`test1.pro`, `test2.pro`, `test3.pro`) are included as examples in the WFEmodeling directory.

I have not yet written a function to fit the WFE to Zernike polynomials. This would be a helpful way to characterize the aberration, even though Zernike polynomials are not orthogonal over the (square) *MOSES* aperture.

3.4 Implementation—*MOSES*

A set of codes for modeling and analyzing the *MOSES* instrument. The source codes are in Appendix D. The IDL routine `moses_wfe` (Appendix D.1) calculates the wavefront error of *MOSES*. There are options to optimize the focus position (which removes tip, tilt and defocus), and to input manufacturing wavefront errors. The IDL script `moses_ideal` (Appendix D.2) implements a demonstration of `moses_wfe` for the optics as designed. The results of `moses_ideal` are shown in table 4, and Figures 2-3.

The primary aberration in all spectral orders is astigmatism. This is consistent with a

spherical mirror used off-axis. In the nonzero spectral orders, the diffraction angle adds to the effective field angle, leading to large astigmatism. This is by design. The larger PSFs in the outboard orders reduce imaging performance, but also ensure a band-limited image so that they can be co-registered to the central order.

4 Test 1: Astigmatism

The ability of the model to incorporate figure error is tested by `moses_test1` (Appendix D.5). I chose an arbitrary grating figure error $\Delta x = Ayz$, where $A = 10^{-7}\,\mathrm{mm}^{-1}$. This is pure astigmatism. Using the `wfe_grating` keyword, I applied the corresponding wavefront error:

$$\phi_P = \frac{2}{pi}\,Ayz. \tag{16}$$

The results are shown in Table 5 and Figures 4-5. The resulting focus shift, compared to the ideal case (Table 4), is calculable but insignificant. The applied astigmatism is approximately double the value of the astigmatism inherent in the *MOSES* design for $m = \pm 1$. The WFE for the minus order, which has inherent astigmatism of the opposite sign, remains about the same, while the WFE in $m = +1$ is tripled. This leads to the highly asymmetric performance shown in Figure 5.

Table 4: Results of `moses_ideal`. Optimal focus position is given by $f + [dx, dy, dz]$.

m	RMS WFE	dx	dy	dz
-1	1.95	0.174	-0.074	-0.010
0	0.35	0.131	0.000	+0.033
$+1$	1.95	0.174	+0.074	-0.010

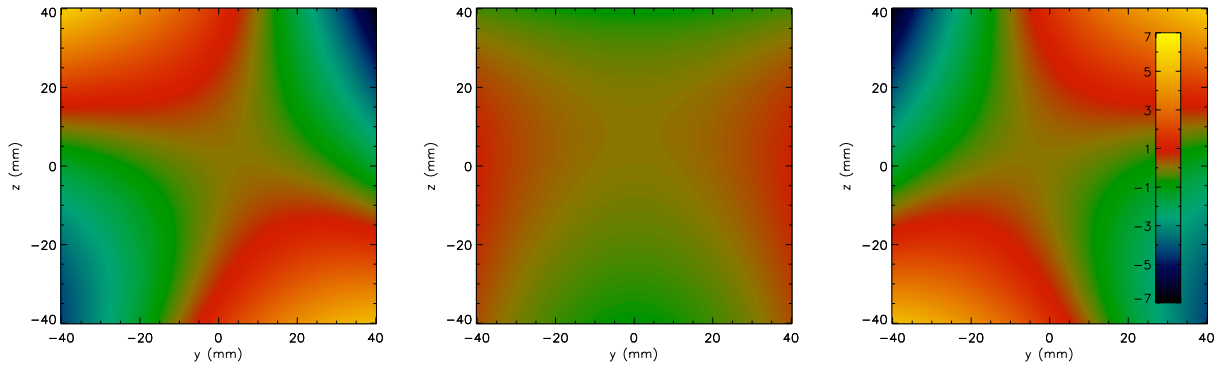

Figure 2: Ideal wavefront error (in waves, $\lambda = 304\,\text{Å}$) of *MOSES* for $m = -1, 0, +1$.

Table 5: Results of `moses_test1`. Optimal focus position is given by $f + [dx, dy, dz]$.

m	RMS WFE	dx	dy	dz
-1	1.83	0.169	-0.074	-0.010
0	3.54	0.131	0.000	0.033
$+1$	5.39	0.180	0.074	-0.010

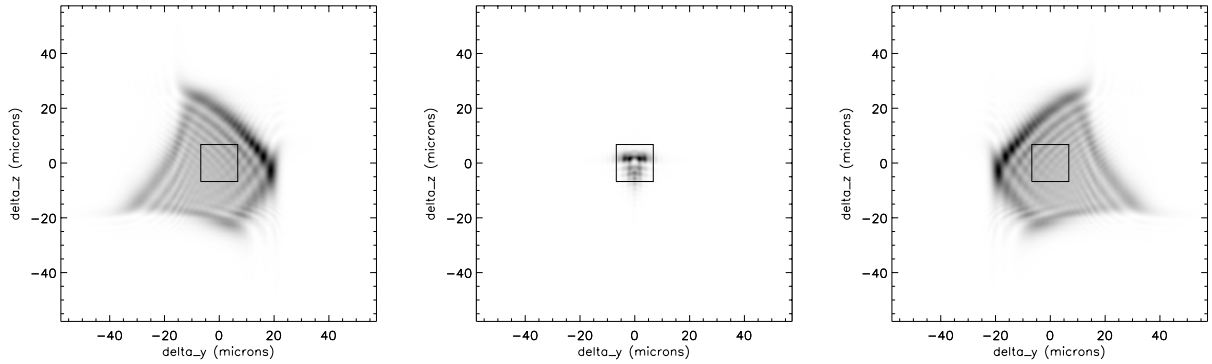

Figure 3: Ideal point spread function of *MOSES* for $m = -1, 0, +1$ (negative images). The 13.5 μm square centered in each plot represents one pixel.

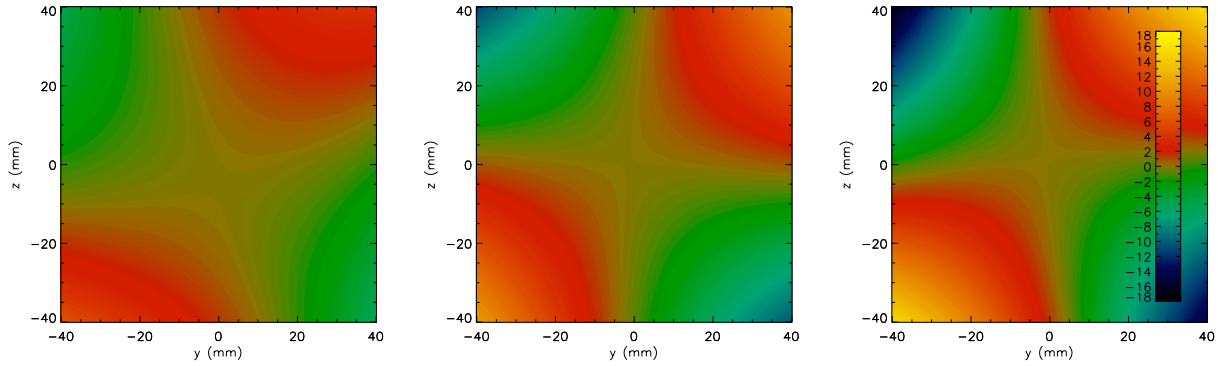

Figure 4: Wavefront error (in waves) for $m = -1, 0, +1$, including the astigmatism specified in equation 16.

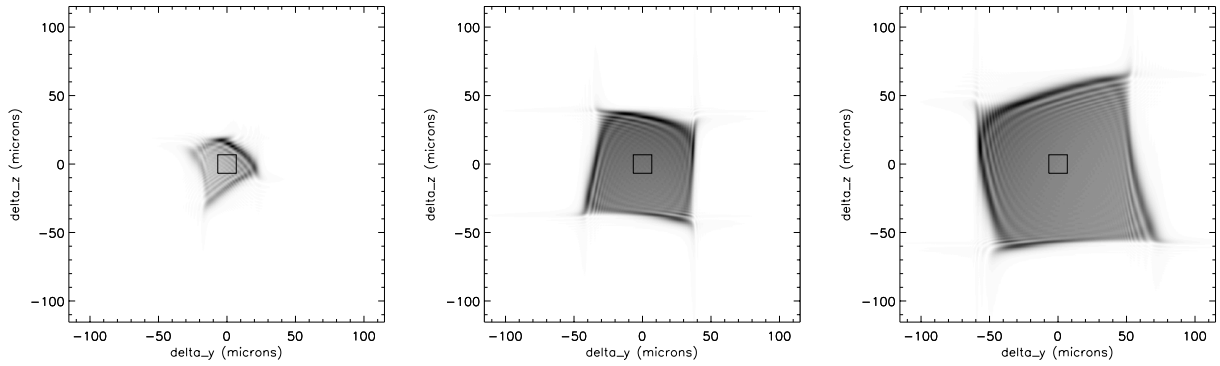

Figure 5: Point spread function for $m = -1, 0, +1$ (negative images). This includes the astigmatism specified in equation 16. The $13.5\,\mu$m square centered in each plot represents one pixel. Note the different scale compared to Figure 3.

5 Test 2: Focus Series

The function `moses_defocus` (Appendix D.3) simulates a focus series for the instrument. The calculation is done two ways:

1. By directly moving the image plane from ideal focus.
2. Using the `defocus` routine.

The latter method adds a paraboloid to the WFE, with a radius of curvature determined by the system f-ratio and the desired degree of defocus.

The results are spooled out to a rather large save file, which can then be interpreted by a script such as `mdef_analyze` (Appendix D.4). There is provision for a manufacturing wavefront error to be input to `moses_defocus`, but `moses_test2` provides an elementary example for an idealized *MOSES*. As of this writing, with 51 focus steps, `moses_test2` takes about 9 minutes on a MacBook Pro 2.16 GHz Core 2 Duo.

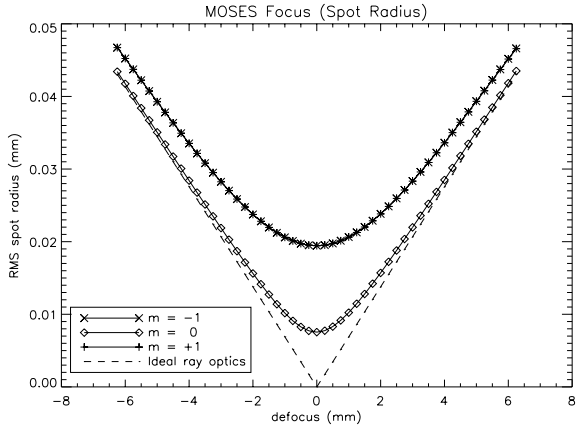

Figure 7: Summary results of the focus series.

Figure 6 is an embedded movie showing the results by both calculation methods. Note that they are identical. The astigmatism of all three spatial orders is evident in the focus series. Both methods of calculation yield

focus movie

Figure 6: *MOSES* focus series animation. The central frame corresponds to figure 3.

identical results, which validates the convenient and completely generic `defocus` program. The spot size is plotted *vs.* focus position in figure 7. The dashed line shows how the RMS spot radius would behave in the absence of diffraction and aberrations.

6 Test 3: One WFE Realization

What could lead to the PSFs observed in the *MOSES*-2006 flight data? Function `moses_test3` (Appendix D.7) explores this question in a preliminary way. A small amount of astigmatism is applied to the *MOSES* WFE, and then a focus series is generated.

A Coordinate transform

We wish to transform from design coordinates (X, Y, Z) to model coordinates (x, y, z). The axes keep the same orientation, but the origin is moved to the center of the concave grating. The coordinates of the detector of order m are placed at its virtual image in the fold flat. The geometry is summarized in Figure 8. By definition,

$$(x_G, y_G, z_G) = (0, 0, 0). \tag{17}$$

The coordinates of the fold flat are

$$x_F = X_F - X_G, \tag{18}$$
$$y_F = 0, \tag{19}$$
$$z_F = Z_F - Z_G. \tag{20}$$

For the detectors,

$$y_m = Y_m. \tag{21}$$

The rays extrapolated through the fold flat to the virtual image of the detector have slope

$$M \equiv \frac{Z_F - Z_G}{X_F - X_G} = \frac{z_m - z_F}{x_m - x_F}. \tag{22}$$

The distance to focus has to remain the same, so

$$(x_m - x_F)^2 + (z_m - z_F)^2 = (X_m - X_F)^2 + (Z_m - Z_F)^2 . \tag{23}$$

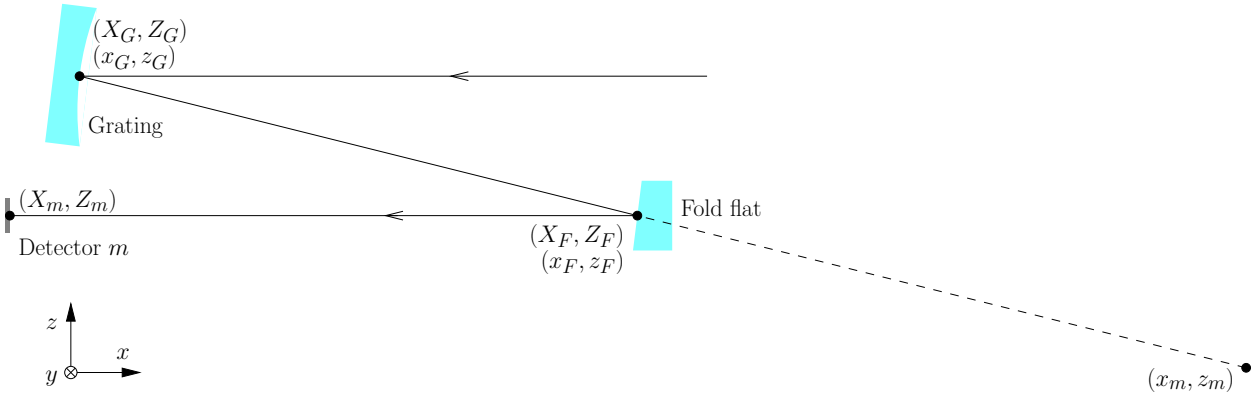

Figure 8: Design coordinates (X, Y, Z) and model coordinates (x, y, z) (not to scale).

Solving equation 22 for z_m and substituting into equation 23,

$$z_m = z_F + M(x_m - x_F), \quad \text{where} \tag{24}$$

$$x_m = x_F + \sqrt{\frac{(X_m - X_F)^2 + (Z_m - Z_F)^2}{1 + M^2}}. \tag{25}$$

A.1 Source Code: `coordinate_xform.m`

This simple GNU Octave program implements the change of coordinates derived in Appendix A.

```
% This is a script for converting MOSES design coordinates to WFE
% model coordinates.
% 2009-May-20  C. Kankelborg
clear % Erase any variables that have been defined previously.

O = '= = Design coordinates XYZ (mm) = = = O'
% Grating:
XG = 118.7
YG =   0.0
ZG =  78.5
% Fold flat:
XF = 2428.6
YF =    0.0
ZF =    0.0
% Spectral orders:
m = [-1, 0, +1]
% Detectors at each spectral order:
X = [   3.0, 0.0,   3.0]
Y = [-136.7, 0.0, 136.7]
Z = [   0.0, 0.0,   0.0]

O = '= = Model coordinates xyz (mm) = = = O'
% By definition:
xG = 0
yG = 0
zG = 0
```

```
% Translation of coordinates for the fold flat:
xF = XF - XG
yF = 0
zF = ZF - ZG

% Find new detector coordinates, unfolding the fold mirror:
y = Y
M = (ZF - ZG)/(XF - XG)
x = xF + sqrt( ((X-XF).^2 + (Z-ZF).^2) ./ (1+M^2) )
z = zF + M*(x - xF)
```

B Diffraction Theory

In free space, electromagnetic radiation propagates according to the wave equation,

$$c^2 \nabla^2 \mathbf{E} = \frac{\partial^2 \mathbf{E}}{\partial t^2}. \tag{26}$$

This equation for the electric field follows easily from Maxwell's equations, and is satisfied also by the magnetic field. We will make a series of simplifications, resulting in a straightforward foralism that represents the propagation of the wave through our instrument in terms of a wavefront entering a small aperture. Our approach will differ from traditional textbook presentations in that we will *not* assume that the aperture lies in a plane perpendicular to the direction of propagation. Instead, our formalism facilitates the treatment of a slow beam emerging from the surface of a steeply angled and/or strongly figured mirror or diffraction grating.

The general solution of the vector wave equation may be expressed in terms of plane waves. This solution, however, is not of much use because the wave equation itself does not encapsulate all of the physics. Only transverse wave solutions satisfy Maxwell's equations. To avoid the complexity introduced by the transversality requirement, we will instead treat waves in a scalar field,

$$c^2 \nabla^2 E = \frac{\partial^2 E}{\partial t^2}. \tag{27}$$

This may seem like an artificial choice, but if we confine ourselves to small f-ratio (as we will momentarily), transverse waves of each polarization state may be considered separately as scalar waves. In practice, the two polarizations may or may not behave differently enough to require separate treatment.

Consider monochromatic waves,

$$E = E(\mathbf{r})e^{i\omega t}, \quad \omega = kc.$$

This results in the Helmholtz equation,

$$(\nabla^2 + k^2)E = 0. \tag{28}$$

This time independent approach sacrifices no generality at all, because a fully time dependent solution can be synthesized by superimposing waves of different frequencies.

We will now solve for $E(\mathbf{f})$, where the "field point" $\mathbf{f}$ lies within a volume $\mathcal{V}$, a distance R from source point $\mathbf{r}$. We are given $E(\mathbf{r})$ on the entire enclosing surface $\partial\mathcal{V}$. The geometry is sketched in figure 9. We define the Green's function,

$$G(R) = \frac{e^{ikR}}{4\pi R}, \quad R \equiv |\mathbf{f} - \mathbf{r}|, \tag{29}$$

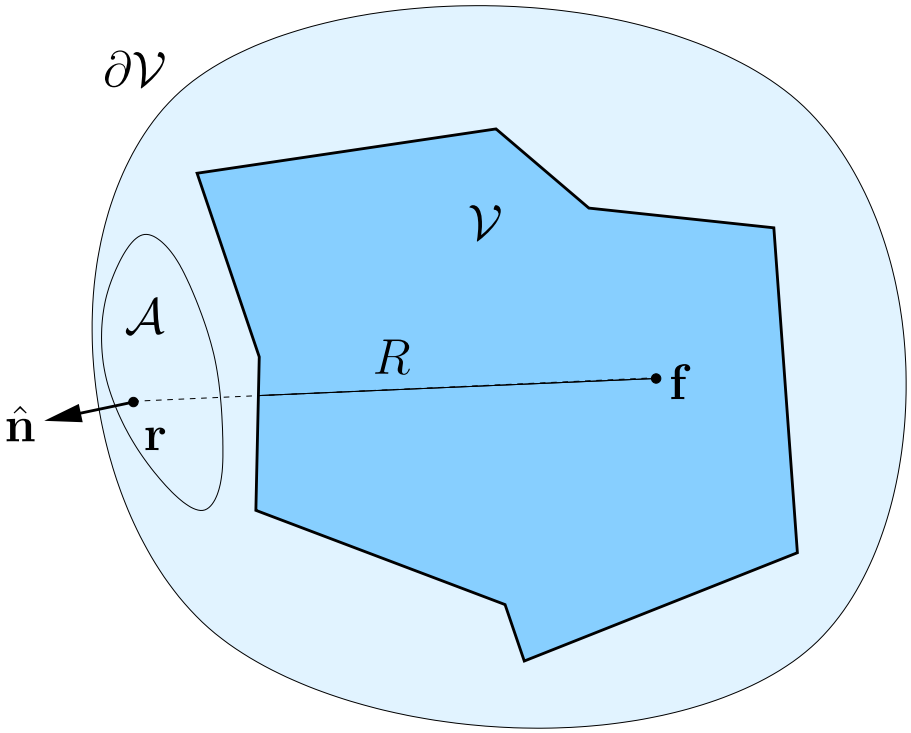

Figure 9: Sketch of the volume $\mathcal{V}$ on which the Helmholtz equation is to be solved.

which has the useful property

$$(\nabla^2 + k^2)G = \delta(\mathbf{f} - \mathbf{r}). \tag{30}$$

The solution is derived from Green's second identity,

$$\int_{\mathcal{V}} \left[E\nabla^2 G - G\nabla^2 E\right]\, d^3r = \oint_{\partial\mathcal{V}} \left[E\nabla G - G\nabla E\right] \cdot \mathbf{dS}. \tag{31}$$

Equations 28 and 30 simplify the left side. Using the normal derivative $\partial_n \equiv (\mathbf{dS} \cdot \nabla)/|\mathbf{dS}|$ on the right,

$$E(\mathbf{f}) = \oint_{\partial\mathcal{V}} \left[E\frac{\partial G}{\partial n} - G\frac{\partial E}{\partial n}\right]\, dS. \tag{32}$$

We imagine that light is entering $\mathcal{V}$ through an "aperture" $\mathcal{A}$. It is assumed that the electric field and its normal derivative are zero on the surface $\partial\mathcal{V}$ except within the aperture, so that the surface integral may be restricted to $\mathcal{A}$.[1] We further assume a narrow (slow) beam, aligned nearly parallel to the vector $\mathbf{f} - \mathbf{r}$. This entails, among other things, a large f-ratio ($R/\sqrt{\mathcal{A}} \gg 1$). Without any further assumptions, our x-axis is chosen to make small angles with the beam and with $\mathbf{f} - \mathbf{r}$. Therefore,

$$\frac{\partial}{\partial n} = \frac{\hat{\mathbf{n}} \cdot (\mathbf{r} - \mathbf{f})}{|\mathbf{r} - \mathbf{f}|}\frac{\partial}{\partial R} = -\frac{\hat{\mathbf{n}} \cdot (\mathbf{r} - \mathbf{f})}{|\mathbf{r} - \mathbf{f}|}\frac{\partial}{\partial x}. \tag{33}$$

The dot product can be eliminated by taking the integral over the variables y and z that are (approximately) orthogonal to the propagation direction. In other words,

$$dS = dy\, dz\, \frac{|\mathbf{r} - \mathbf{f}|}{\hat{\mathbf{n}} \cdot (\mathbf{r} - \mathbf{f})},$$

and so

$$E(\mathbf{f}) = \iint_{\mathcal{A}} \left[E\frac{\partial G}{\partial R} + G\frac{\partial E}{\partial x}\right]\, dy\, dz. \tag{34}$$

If the fractional variation in R is small and $kR \gg 1$ (i.e., the far field), then the expression for the wave function becomes exceedingly simple:

$$E(\mathbf{f}) = \frac{ik}{2\pi R} \iint_{\mathcal{A}} E(\mathbf{r}) e^{ikR}\, dy\, dz. \tag{35}$$

The factor of i only affects the phase. If we are only interested in calculating point spread functions, we may ignore the normalization factor in front of the integral. We have made

[1] This actually introduces a mathematical inconsistency. See, e.g., Born and Wolf.

a series of assumptions that restrict our attention to the far field of a slow optical system. However, we have left open the possibility that the surface $\mathcal{A}$ is strongly curved and angled with respect to the xy-plane. This formulation is not among the traditional diffraction approximations (e.g. Kirchoff, Fresnel, Fraunhofer). Nevertheless, it is compatible with a simple Fourier transform approach, as demonstrated in Section 3.2.

C WFEmodeling Library

C.1 Source Code: psd2wfe.pro

```
;NAME:
;   PSD2WFE
;PURPOSE:
;   Convert a power spectral density for polishing errors in a
;   reflective optic to a randomly generated wavefront error (WFE) map.
;   The WFE is generated from the power spectrum using random phases
;   and amplitudes. It is *not* periodic. Note from the documentation
;   below that the PSD runs from zero to the Nyquist frequency in units
;   of exactly *half* the fundamental. Note that the output WFE map is
;   complex (see keyword CIRCULAR). The imaginary part could be used
;   to represent apodization or nonuniform illumination of the aperture
;   if you like.
;CALLING SEQUENCE:
;   wfe = psd2wfe(psd [, rms=rms | pv=pv])
;INPUT PARAMETERS:
;   PSD = An N-element power spectral density for the polishing errors.
;       The period associated with the ith element of PSD is 2(N-1)/i
;       pixels in the ouptut N x N wafefront error map. One consequence
;       of this is that there is noplace to specify power in the frequency
;       interval between the 1D Nyquist frequency and the diagonal
;       Nyquist frequency. The PSD will be assumed flat in this range.
;OPTIONAL KEYWORD INPUTS:
;   RMS = The (scalar) RMS wavefront error. This is used to scale the
;       result, which eliminates any ambiguity about the normalization
;       of the Fourier transform (and hence of the PSD).
;   PV = Alternative to RMS, may specify peak-to-valley.
;   CIRCULAR = If set, then a circular aperture is inscribed in the square
;       WFE array. The region outside the circle is assigned the value
;       complex(0, 1e9). This results in a zero E-field later, which is
;       appropriate for regions outside the aperture.
;   VERBOSE =  If set, then print out some informative statistics.
;OUTPUTS:
;   WFE = a randomly generated, N x N wavefront error map that is consistent
;         with the input PSD and RMS or PV. If neither RMS nor PV are specified,
```

```
;         then the WFE will have a PV of 0.25 (a notional quarter-wave mirror).
;MODIFICATION HISTORUY:
;   2008-OCT-29  C. Kankelborg
;   2008-Nov-03  CCK Incorporated removal of best-fit spherical wave
;       before normalizing the WFE.
;   2008-Nov-07  CCK Added keyword option /CIRCULAR, and thereby
;       introduced the concept of a complex WFE map. The imaginary
;       part represents attenuation of the electric field amplitude.
;   2008-Nov-10  CCK fixed bug--- sqrt(PSD) to get amplitudes. Added
;       diagnostic printing of Strehl ratio. Added VERBOSE keyword
;       so that diagnostic messages are suppressed by default.

function psd2wfe, psd, rms=rms, pv=pv, circular=circular

const_i = sqrt(complex(-1)) ;square root of minus one

N = n_elements(psd)
frequencies = ( dist(2L*N) ) < N  ;2N x 2N array of frequencies

ft_power = interpolate(psd, frequencies, cubic=-0.5)>0
   ;threshold is to eliminate negatives that creep in from the interpolation.
ft_amplitudes = sqrt(ft_power)
   ;amplitude is the square root of power.
ft_phases = 2.0 * !pi * randomu(seed, 2L*N, 2L*N)
ft_complex = ft_amplitudes * exp(const_i * ft_phases)

wfe_big = float( fft(ft_complex, /inverse) )
wfe1 = wfe_big[0:N-1, 0:N-1] ;Cropping eliminates the periodic boundary conditions.

;Now subtract best-fit spherical wave
spherewave = fit_spherical_wave(wfe1)
wfe = complex(wfe1 - spherewave)
   ;Henceforth WFE will be complex. This has many uses, including
   ;the representation of apetures, apodization, and nonuniform
   ;illumination.
if keyword_set(verbose) then begin
    print,'Subtracted spherical component PV = ',max(spherewave) - min(spherewave)
    print,'Residual WFE before normalization is PV = ',max(wfe) - min(wfe)
```

```
endif

;Introduce aperture cropping, if any
if keyword_set(CIRCULAR) then begin
   aperture = where( shift(dist(N), N/2, N/2) lt N/2,  complement=crop )
   wfe[crop] = complex(0,1e9)
endif else aperture = findgen(N,N)



;Normalize wfe (any apodization or nonuniform aperture features should
;be inserted AFTER this step, since we would not want them to interfere
;with the scaling of the mirror surface figure)
wfe[aperture] -= mean(wfe[aperture]) ;set to mean of zero.
if keyword_set(rms) then begin
    wfe[aperture] *= rms / stdev(wfe[aperture])
      ;scale to desired RMS wavefront error
endif else begin
    if not keyword_set(pv) then pv = 0.25 ;quarter wave nominal
    range = max(wfe[aperture]) - min(wfe[aperture])
    wfe[aperture] *= pv/range
endelse

if keyword_set(verbose) then begin
    print,'WFE realized  PV = ', max(wfe[aperture]) - min(wfe[aperture])
    print,'WFE realized RMS = ',stdev(wfe[aperture])
    print,'Strehl = ',wfe2strehl(wfe)
endif
return, wfe

end
```

C.2 Source Code: wfe2psd.pro

```
;NAME:
;   WFE2PSD
;PURPOSE:
;   Produce an estimate of the power spectral density (PSD) of a wavefront
```

```
;   error map (WFE).
;CALLING SEQUENCE:
;   psd_est = wfe2psd(wfe)
;ALGORITHM:
;   A periodogram approach is used with a Hanning window. Two estimates
;   are produced and averaged: one using a vertical chord, and one a
;   horizontal chord. Compare Walsh et al. 1999, Appl. Opt. Vol. 38, No. 22,
;   p.4790.
;MODIFICATION HISTORY:
;   2008-Nov-10  C. Kankelborg
function wfe2psd, wfe

wfe_size=size(wfe)
N = wfe_size[1]
if wfe_size[2] ne N then message,'WFE not square. Exiting.'

windo = hanning(N)
    ;windowing function for estimate of PSD

chord1 = float(wfe[N/2,*])
chord2 = float(wfe[*,N/2])

c1fft = fft(chord1 * windo)
c2fft = fft(chord2 * windo)

psd = abs(c1fft)^2 + abs(c2fft)^2

;IDEA: ADD FEATURE TO FIT A POWER LAW TO THE PSD

return,psd[0:N/2-1]
end
```

C.3 Source Code: mirror.pro

```
;NAME:
;  mirror
;PURPOSE:
;  Use physical parameters to generate PSD and WFE for a mirror.
```

```
;  The polishing process is modeled by a power law PSD from one
;  period per diameter down to a cutoff.
;CALLING SEQUENCE:
;  wfe = mirror(diameter, rmswfe [, N=N] [, hole=hole] $
;     [, pindex=pindex] [, pcutoff=pcutoff] $
;     [, psd=psd] [, period=period]
;INPUT PARAMETERS:
;  diameter = diameter of the beam on the mirror (not the mirror
;     itself) in physical units (usually mm).
;OPTIONAL INPUT PARAMETERS:
;  rmswfe = RMS wavefront error (floating point number). If not supplied,
;     then the default behavior of psd2wfe will be used (1/4 wave PV).
;OUTPUT PARAMETERS:
;  wfe = wavefront error (WFE) for a particular, randomized
;     realization with the desired PSD.
;OPTIONAL KEYWORD INPUTS:
;  N = number of points (integer). The resulting PSD will have N
;     elements, and the WFE map will be NxN. Default=256.
;  circular = if set, use a circular aperture (passed straight
;     through to psd2wfe).
;  pindex = power-law index of the power spectral density (PSD)
;     of the polishing errors. PSD goes as
;           PSD = k^(-pindex).
;     Typical values are 2.1-3.0. Default=2.5.
;  pcutoff = The shortest period that will have nonzero power.
;     By default, 4mm. Use this keyword if you want a different
;     cutoff or if you want to use different units for the diameter.
;  hole = Diameter of concentric hole to be drilled in mirror. Uses same
;     units as diameter and pcutoff.
;  renormalize = If set, then renormalize to get the specified rmswfe over
;     the clear aperture, excluding the "hole" if specified. If not set,
;     then rmswfe will pertain to everything within the outer
;     boundary of the aperture.
;OPTIONAL KEYWORD OUTPUTS:
;  psd = The N-element PSD array. Note that the scale of the PSD
;     values is arbitrary.
;  period = N-element array of period values, in the same units
;     as diameter and pcutoff.
```

```
;  aperture = index array of WFE elements within the clear aperture.
;     Especially handy for annular mirrors (see CIRCULAR & HOLE keywords).
;MODIFICATION HISTORY:
;  2008-Nov-09  C. Kankelborg
;  2008-Nov-10  CCK Added HOLE and APERTURE keywords. Modified default
;       pindex to 2.5 based on SAO input. Improved documentation.
;  2008-Nov-17  CCK added RENORMALIZE keyword to correct rmswfe when HOLE
;       is specified.
function mirror, diameter, rmswfe, N=N, $
   pindex=pindex, pcutoff=pcutoff, psd=psd, circular=circular, $
   period=period, aperture=aperture, hole=hole, renormalize=renormalize

;Default keyword values
if not keyword_set(N) then N=256 ;default resolution of WFE map
if not keyword_set(pcutoff) then pcutoff = 4 ;mm minimum period
if not keyword_set(pindex) then pindex = 2.5

k = findgen(N) ;wavenumber scale, arbitrary units
period = 2.0*diameter/k ;period in same units as diameter

;NOTE:
;In keeping with psd2wfe, an N-element PSD leads to an NxN WFE.
;The following table may help to clarify the meaning of the
;frequency elements:
;
;  k        interpretation
;  ---------------------------------
;  0        DC.
;  1        1/2 period per diameter
;  2        1 period per diamater
;  ...
;  N-1      Nyquist frequency


;Implement power law PSD
psd = k^(-pindex)
psd[0:1] = 0
   ;first k with nonzero power is 1 per diameter
```

```
;Implement cutoff frequency
ss = where(period lt pcutoff)
if ss[0] ne -1 then psd[ss]=0

;Calculate and return WFE
wfe = psd2wfe(psd, rms=rmswfe, circular=circular)

;Drill hole if desired
if keyword_set(hole) then begin
    radius = round(N/2.0*hole/diameter) ;calculate radius in pixels
    radii = shift(dist(N), N/2, N/2)
    ss = where(radii lt radius) ;find where the hole lives
    wfe[ss] = complex(0, 1e9)   ;drill the hole
endif

aperture = where( imaginary(wfe) lt 1.0 )
if keyword_set(renormalize) then begin
   wfe[aperture] *= rmswfe/stdev(wfe[aperture])
endif

return, wfe


end
```

C.4 Source Code: spiders.pro

```
;NAME:
;   SPIDERS
;PURPOSE:
;   Model the spiders that support a Cassegrain secondary mirror for
;   purposes of WFE/PSF/MTF estimation. The resulting mask is to be
;   ADDED to a WFE map in order to block the wavefront. Note that
;   the mirror procedure is capable of setting up a circular aperture
;   with a central obstruction. Thus, MIRROR and SPIDERS together form
;   a comprehensive way of modeling the aperture of a Cassegrain telescope.
;CALLING SEQUENCE:
```

```
;   mask = spiders(N,widths,angles)
;EXAMPLE:
;   wfe += spiders(N,widths,angles) ;the mask is ADDED to the WFE.
;INPUT PARAMETERS:
;   N = size of square WFE array (N x N)
;   widths = Array of spider widths in pixels.
;   angles = Array of spider angles in degrees. Must have same number
;       of elements as the widths array. Angles are measured CCW
;       from a horizontal axis drawn from mask array center to right edge.
;OUTPUT PARAMETERS:
;   mask = An NxN complex array whose real part is 1.0 everywhere. The
;       imaginary part is 1e6 or greater on the spiders and 0.0 elsewhere. When
;       mask is multiplied by a WFE map, it has the effect of zeroing the
;       electric field on the spiders.
;MODIFICATION HISTORY:
;   2008-Nov-14  C. Kankelborg
;
function spiders, N, widths, angles

Nspiders = n_elements(widths)
if n_elements(angles) ne Nspiders then $
    message,'How many spiders do you really want? widths and angles do not have the same

;Initialize the mask.
mask = replicate(complex(0.0, 0.0), N, N)

;Create each spider and add it into the mask.
for i=0, Nspiders-1 do begin
    spider = complexarr(N, N)
    baseline = round((N-widths[i]-0.0)/2)
    spider[N/2:N-1, baseline:round(baseline+widths[i]-1)] = complex(0.0, 1e6)
    mask += rot(spider, -angles[i])
endfor

return, mask

end
```

C.5 Source Code: fit_spherical_wave.pro

```
;NAME:
;   fit_spherical_wave
;PURPOSE:
;   Find the best-fit spherical wave to a WFE map so that it can be subtracted
;   out. This is equivalent to removing defocus, tip and tilt. Actually, I have
;   settled for a paraboloid approximation of a sphere. The best fit is defined
;   by minimizing the sum of the absolute deviations (robust fitting).
;CALLING SEQUENCE:
;   swave = fit_spherical_wave(wfe)
;INPUT PARAMETERS:
;   WFE = 2D wavefront error map. The units can be phase, waves, nanometers,
;   or whatever. Compatible with complex WFE as introduced in psd2wfe.
;OUTPUTS:
;   SWAVE = Best fit spherical surface, in the same units as WFE. The WFE
;       pixels are assumed to be square.
;MODIFICATION HISTORY:
;   2008-Nov-03  C. Kankelborg
;   2008-Nov-10  CCK adapted to complex wfe (see psd2wfe.pro).
;   2009-May-28  CCK tweaked precision and max iterations for
;        use with moses_ideal.pro.

function paraboloid, prams
common fsw_block, wfe, Nx, Ny, xp, yp, aperture
z1 = prams[0] ;offset
K  = prams[1] ;curvature; K = 1/R if xy units equal z units.
x0 = prams[2] ;center
y0 = prams[3] ;center

return, (K/2.0)*( (xp-x0)^2 + (yp-y0)^2 ) + z1
end

function badness, prams
common fsw_block, wfe, Nx, Ny, xp, yp, aperture

sphere = paraboloid(prams)
result = total( abs(sphere[aperture] - wfe[aperture]) ) + 1.0
```

```
   ;The 1.0 aids in convergence!
; print,'[z1,K,x0,y0] = ',prams
; print,'badness = ',result
return,result
end



function fit_spherical_wave, wfe
common fsw_block, wfe_temporary, Nx, Ny, xp, yp, aperture
wfe_temporary = wfe ;Silly, memory-wasting workaround for the common block
aperture = where(imaginary(wfe) lt 1) ;adaptation to complex WFE

wfe_size = size(wfe)
Nx = wfe_size[1]
Ny = wfe_size[2]

;Create rectangular arrays for x and y coordinates.
xp = findgen(Nx) # replicate(1.0,Ny)
yp = replicate(1.0,Nx) # findgen(Ny)

initial_prams = [0,0,Nx/2.0,Ny/2.0]
scale_prams = [1.0, 1.0/((Nx/2)^2.0 + (Ny/2)^2.0), Nx, Ny]

prams_fit = amoeba(1e-5, FUNCTION_NAME = 'badness', $
    P0=initial_prams, scale=scale_prams, nmax=10000)

return, paraboloid(prams_fit)
end
```

C.6 Source Code: `defocus.pro`

```
;+
;NAME:
;  DEFOCUS
;PURPOSE:
```

```
;  Modify a wavefront error (WFE) image to simulate an increase (or decrease)
;  in distance to the detector. Resulting PSFs obtained with WFE2PSF will
;  show the focus changes, and can be used to create a focus series.
;CALLING SEQUENCE:
;  wfe_new = defocus(wfe_orig, disp, fratio, lambda)
;INPUT PARAMETERS:
;  wfe_orig -- 2D image of WFE, in waves, associated with the optical system.
;  disp -- displacement (change in distance from aperture to detector), in
;     same units as lambda.
;  fratio -- f-ratio of the optical system: effective focal length divided
;     by aperture diameter, where the aperture diameter equals the width of
;     the WFE image.
;  lambda -- wavelength of light, in same units as distance. If included,
;     then it is assumed that the wfe input and output will be in waves.
;     If it is not given, then lambda is set to 1 (equivalently, we may
;     say that the WFE is assumed to be in the same distance units as
;     disp, or that the displacement is given in waves).
;OUTPUT PARAMETERS:
;  wfe_new -- wavefront error with a paraboloid of revolution added, to
;     simulate the desired defocus (specified by distance).
;OPTIONAL OUTPUT KEYWORDS:
;  wfe_change -- change in WFE (wfe_new - wfe_orig).
;ALGORITHM:
;  We assume a large f-ratio, which results in a paraboloidal change in
;  wavefront error given by:
;     WFE_change =  -0.5*(disp/lambda) * (x^2 + y^2) / fratio^2,
;  where x and y are coordinates centered in the aperture and normalized
;  to its width. If fratio < 4, a warning is thrown.
;BUGS:
;  There may be some ambiguity as to what is meant by a positive wavefront
;  error. If the convention of this program differs from the convention
;  that generated wfe_orig, then the sign of disp would effectively be
;  reversed. Might want to include a keyword for that. By the way, this
;  version has been tested to agree with an alternate calculation in
;  moses_defocus.
;MODIFICATION HISTORY:
;  2009-Jun-10  C. Kankelborg
;  2009-Jun-11  CCK reversed sign of WFE change based on testing with
```

```
;      moses_defocus.
;-
function defocus, wfe_orig, disp, fratio, lambda, wfe_change=wfe_change

if n_elements(lambda) ne 1 then lambda = 1.0
if fratio lt 4.0 then message, $
   'WARNING: The approximation may be poor for fratio < 4.', /informational

;Create normalized aperture coordinates
wfesize = size(wfe_orig)
Nx = wfesize[1]
Ny = wfesize[2]
xlin = findgen(Nx)/(Nx-1.0) - 0.5
ylin = ( findgen(Ny) - 0.5*(Ny-1) )/(Nx-1.0)
x = xlin # replicate(1.0, Ny)
y = replicate(1.0, Nx) # ylin

;Calculate WFE
wfe_change = -0.5*(disp/lambda) * (x^2 + y^2) / fratio^2

return, wfe_orig + wfe_change
end
```

C.7 Source Code: wfe2psf.pro

```
;NAME:
;  wfe2psf
;PURPOSE:
;  Convert a wavefront error map to a point spread function.
;CALLING SEQUENCE;
;  psf = wfe2psf(wfe [,oversample=oversample])
;       ...see below for more keyword options!
;INPUT PARAMETERS:
;  wfe = wavefront error map, which may be complex as documented
;  in psd2wfe.pro. Must be a 2D square (NxN) array. Note that large
;  imaginary values essentially result in zero Efield (darkness). This
;  is how apertures are marked off. And by more subtle application,
;  apodization or nonuniform illumination effects could also be
```

```
;  implemented.
;OUTPUT PARAMETERS:
;  psf = (scalar, 2D array) point spread function, same dimensions
;     as the input wfe, and normalized to 100% response at DC.
;OPTIONAL KEYWORD INPUTS:
;  oversample = factor by which to oversample the PSF. Oversampling is
;     highly recommended. Default=4. Only integer values have been tested,
;     but maybe a non-integer would work. Do you feel lucky?
;  fratio = The f-ratio of the beam (focal length / diameter). Don't use
;     an effective f-ratio or T-ratio here, as this is solely for
;     geometric calculation of dx and xarr (see below).
;  lambda = Wavelength of the light. See dx and xarr below.
;OPTIONAL KEYWORD OUTPUTS:
;   dx = scale of pixels of the output image. Requires fratio and
;        wavelength keywords. The units are the same as the wavelength units.
;   xarr = array of axis values for the side of the PSF image. Requires
;        fratio and wavelength keywords. Units same as wavelength.
;MODIFICATION HISTORY:
;  2008-Nov-07  C. Kankelborg
;  2008-Nov-10  CCK added dx and xarr keywords.
;
function wfe2psf, wfe, oversample=oversample, fratio=fratio, lambda=lambda, $
    dx=dx, xarr=xarr

wfe_size = size(wfe)
N = wfe_size[1]

i = complex(0,1) ;square root of minus one

if not keyword_set(oversample) then oversample=4

bigmap = complexarr(oversample*N, oversample*N)
   ;padded to the degree specified by the oversample parameter.
bigmap[0:N-1,0:N-1] = exp(i * wfe * 2 * !pi)
bigmap = shift(bigmap, -N/2, -N/2)
Efield = fft( bigmap )
intensity =  abs(Efield)^2
intensity_small = ( shift(intensity, N/2, N/2) )[0:N-1, 0:N-1]
```

```
;Work out the scale of the PSF map.
;Without oversampling, one pixel would subtend an angle equal to the
;wavelength over the beam diameter.
if (n_elements(fratio) eq 1) and (n_elements(lambda) eq 1) then begin
    dx = lambda * fratio / oversample
    xarr = ( findgen(N) - N/2.0 ) * dx
endif
return, intensity_small/total(intensity_small)
end
```

C.8 Source Code: psf2mtf.pro

```
;NAME:
;  psf2mtf
;PURPOSE:
;  Convert a point spread function to a modulation transfer function
;  (and optionally a line spread function).
;CALLING SEQUENCE:
;  mtf = psf2mtf(psf [, theta=theta] [, lsf = lsf] [, dx=dx], [, karr=karr])
;INPUT PARAMETERS:
;  psf = point spread function (floating point 2D array, Nx x Ny). This
;     need not be properly normalized.
;OUTPUT PARAMETERS:
;  mtf = modulation transfer function, which is the amplitude response
;     as a function of scalar wavenumber. This is a 1D array with Nx/2
;     elements. Normalized to unit DC response.
;OPTIONAL KEYWORD INPUTS:
;  theta = CCW orientation of sinusoidal test pattern in degrees.
;     Default, theta=0, corresponds to vertical bars (horizontal
;     resolution).
;  dx = PSF pixel size, in whatever length units are desired for karr.
;OPTIONAL KEYWORD OUTPUTS:
;  lsf = linespread function (which is just PSF summed along the orientation
;     specified by theta).
;  karr = If dx is supplied, then karr is the frequency axis for the MTF
;     in periods per unit length.
;MODIFICATION HISTORY:
```

```
;  2008-Nov-07  C. Kankelborg
;  2008-Nov-10  CCK added dx and karr keywords.

function psf2mtf, psf, theta=theta, lsf=lsf, dx=dx, karr=karr
psf_size = size(psf)
Nx = psf_size(1)

if n_elements(theta) ne 1 then theta=0

psf_rot = rot(psf,theta)
lsf = total(psf_rot,2) ;linespread function
mtf = (abs(fft(lsf)))[0:Nx/2] ;leave out the redundant negative frequencies

if keyword_set(dx) then begin
    karr = findgen(Nx/2+1)/(Nx*dx) ;Max is Nyquist frequency, 1/(2*dx).
endif

return, mtf/mtf[0] ;properly normalized, unity DC response one hopes!

end
```

C.9 Source Code: psf2stats.pro

```
;NAME:
;  psf2stats
;PURPOSE:
;  Convert a point spread function to RMS spot radius (and optionally some
;  other statistics).
;CALLING SEQUENCE:
;  rmsrad = psf2stats(psf [, dx=dx])
;INPUT PARAMETERS:
;  psf = point spread function (floating point 2D array, Nx x Ny). This
;     need not be properly normalized.
;OPTIONAL KEYWORD INPUTS:
;  dx = PSF pixel size, in whatever length units are desired. Default=1.
;     Square pixels are assumed.
;OPTIONAL KEYWORD OUTPUTS:
;  I0 = zeroth moment (total fluence just total(psf)).
```

```
;  xc, yc = centroid coordinates (origin at image center).
;MODIFICATION HISTORY:
;  2009-Jun-23  C. Kankelborg
function psf2stats, psf, dx=dx, I0=I0, xc=xc, yc=yc

psf_size = size(psf)
Nx = psf_size[1]
Ny = psf_size[2]

;Coordinates (origin at image center):
x = (findgen(Nx) - (Nx-1)/2.0)*dx # replicate(1.0, Ny)
y = replicate(1.0, Nx) # (findgen(Ny) - (Ny-1)/2.0)*dx

;Zeroth moment (total fluence):
I0 = total(psf)

;First moments (centroid):
xc = total(x*psf)/I0
yc = total(y*psf)/I0

;Second moments:
r2 = (x-xc)^2 + (y-yc)^2 ;square of radius from spot center
rmsrad = sqrt( total(r2*psf)/I0 )

return, rmsrad
end
```

C.10 Source Code: `wfe2strehl.pro`

```
;NAME:
;  wfe2strehl
;PURPOSE:
;  Convert a wavefront error map to a Strehl ratio. This is defined
;  as the fraction of diffraction-limited intensity at the
;  diffraction-limited (gaussian) focus. For the result to be correct,
;  the best-fit sphere must have been subtracted from the WFE. All my
;  functions that generate WFE maps should do this automatically
;  (e.g. psd2wfe).
```

```
;CALLING SEQUENCE:
;  strehl = wfe2strehl(wfe)
;INPUT PARAMETERS:
;  wfe = wavefront error map in waves.
;OUTPUT PARAMETERS:
;  strehl = the strehl ratio. According to the Marechal criterion,
;     strehl > 0.8 is diffraction limited.
;ALGORITHM:
;  I use the exact form of the Strehl ratio given by Wyant ch.1,
;  eq. 64. See:
;     http://www.optics.arizona.edu/jcwyant/zernikes/Zernikes.pdf
;MODIFICATION HISTORY:
;  2008-Nov-09  C. Kankelborg
function wfe2strehl, wfe

i = complex(0,1) ;square root of minus one
Efield = exp(2*!pi*i*WFE)
strehl = (abs(total(Efield)) / total(abs(Efield)))^2

return, strehl
end
```

C.11 Source Code: wfe2fringes.pro

```
;NAME:
;   WFE2FRINGES
;PURPOSE:
;   Convert a map of wavefront error to an image of horizontal or vertical
;   fringes, as they would appear in an interferometer. It may be useless,
;   but it is cute.
;CALLING SEQUENCE:
;   fringes = wfe2fringes(wfe [, nfringes=nfringes] [, /vertical])
;INPUT PARAMETERS:
;   WFE = An Nx x Ny wavefront error map (in units of waves).
;OPTIONAL KEYWORD INPUTS:
;   NFRINGES = Number of fringes over the mirror surface. Default=4.
;   VERTICAL = If set, then make the fringes vertical (by default, they
;       are horizontal).
```

```
;OUTPUTS:
;   FRINGES = An image of the fringes you would see corresponding to the input WFE.
;MODIFICATION HISTORUY:
;   2008-OCT-30  C. Kankelborg
;   2008-Nov-07  CCK Made compatible with complex WFE maps (see psd2wfe.pro)
function wfe2fringes, wfe, nfringes=nfringes, vertical=vertical

if not keyword_set(nfringes) then nfringes=4

wfe_size = size(wfe)
Nx = wfe_size[1]
Ny = wfe_size[2]

;Construct perfect fringes, either vertical or horizontal.

if keyword_set(vertical) then begin ;vertical fringes
    fringes = nfringes*findgen(Ny,Nx)/( (Nx-1)*(Ny-1) )
    fringes = transpose(fringes)
endif else begin ;horizontal fringes
    fringes = nfringes*findgen(Nx,Ny)/( (Nx-1)*(Ny-1) )
endelse

fringes += wfe  ;Add wavefront error onto the perfect fringes

;Modulate and return.
i = complex(0,1) ;square root of minus one.
return, exp(2*!pi*i*fringes)

end
```

D *MOSES* Fourier Optics Codes

D.1 Source Code: moses_wfe.pro

```
;NAME:
;  moses_wfe
;PURPOSE:
```

```
;  Model the wavefront error (WFE) of the MOSES instrument. Dimensions
;  are in mm. WFE is in waves.
;CALLING SEQUENCE:
;  WFE = moses_wfe(m [, <various options>])
;INPUTS:
;  m = spectral order (normally 1, 0, or -1). Note that the input
;     value is converted to float on return.
;OPTIONAL KEYWORDS:
;  N = (input) size of mesh (NxN).
;  lambda = (input) wavelength (mm). Default 304e-7 (MOSES 06
;     nominal wavelength).
;  optimize = if set, then remove tip/tilt and defocus by optimizing
;     the vector focus displacement, df.
;  df = 3-element offset [dx, dy, dz] from design focal point. Normally
;     used as an input, but if /optimize is set, then df returns the
;     displacement corresponding to best focus.
;  RMSWFE = (output) RMS value of wavefront error.
;  wfe_grating = WFE of grating due to imperfect polishing of substrate. NxN.
;  wfe_rulings = WFE due to imperfect patterning of the rulings. NxN.
;  wfe_fold = WFE of fold flat due to imperfect polishing on the part
;     of the flat associated with spectral order m. NxN.
;MODIFICATION HISTORY:
;  2009-May-28  C. C. Kankelborg
;     Essentially restarted from scratch, beginning with design parameters
;     generated by the Octave program coordinate_xform.m and following
;     the quantitative development in moses_wfe.pdf (see documentation/).
;  2009-May-30  CCK: added optimization, wfe_grating, wfe_rulings,
;     wfe_fold, and N keywords.
;  2009-Jun-02  CCK: fixed bug where keyword N got lost under /optimize. This
;     keyword is now handled much wfe_grating/wfe_rulings/wfe_fold. These are
;     all a bit tricky since (1) they can't be passed as arguments or keywords
;     to mopt_badness and (2) moses_wfe is called by mopt_badness.

;=======================================;
;     F I G U R E   O F   M E R I T     ;
;=======================================;
function mopt_badness, df
common moses_opt, order, wavelength, WFE_MANUFACTURING, Nmesh
```

```
   ;parameters needed for moses_wfe, but not part of the optimization.
wfe = moses_wfe(order, lambda=wavelength, df=df, rmswfe=rmswfe, $
   wfe_grating = WFE_MANUFACTURING, N=Nmesh)
return, rmswfe
end

;=======================================;
;  F O C U S   O P T I M I Z A T I O N  ;
;=======================================;
pro moses_optimize, m, lambda=lambda, df_best=df_best, $
   wfe=wfe, rmswfe=rmswfe, prior_wfe=prior_wfe, N=N

common moses_opt, order, wavelength, WFE_MANUFACTURING, Nmesh
   ;parameters needed for moses_wfe, but not part of the optimization.
order = m
wavelength = lambda
WFE_MANUFACTURING = prior_wfe
Nmesh = N

;Optimization:
df0 = [0.0, 0.0, 0.0] ;initial guess
df_best = amoeba(1e-8, FUNCTION_NAME = 'mopt_badness', $
    P0=df0, scale=[1,1,1], nmax=10000) ;options chosen for < 1 um precision.
wfe = moses_wfe(m, df=df_best, lambda=lambda, rmswfe=rmswfe, $
   wfe_grating = WFE_MANUFACTURING, N=Nmesh)
end


;=======================================;
;        M A I N   P R O G R A M        ;
;=======================================;
function moses_wfe, m, df=df, lambda=lambda, $
   optimize=optimize, rmswfe = rmswfe, N=N, $
   wfe_grating=wfe_grating, wfe_rulings=wfe_rulings, wfe_fold=wfe_fold

;Simulation parameters
if not keyword_set(N) then N = 256 ;Simulated mirror will be N x N.
if not keyword_set(lambda) then lambda = 304e-7
```

```
if keyword_set(df) then begin
   if n_elements(df) ne 3 then message, 'Keyword df needs 3 elements!'
endif else df = [0, 0, 0]

prior_wfe = dblarr(N,N) ;zero unless wfe_* keywords set...
if keyword_set(wfe_grating) then prior_wfe += wfe_grating
if keyword_set(wfe_rulings) then prior_wfe += wfe_rulings
if keyword_set(wfe_fold)    then prior_wfe += wfe_fold


;-----------------------------------------------;
;           Optical Design parameters           ;
;  in "model coordinates" xyz, with the origin  ;
;         at the center of the grating.         ;
;-----------------------------------------------;
;Grating:
R =  9480.00d ;radius of curvature
x0 = 9478.63d ;center of curvature, x
y0 =    0.00d ;center of curvature, y
z0 = -160.98d ;center of curvature, z
xG = 0d
yG = 0d
zG = 0d
d = 1.0d/950 ;groove spacing
;Fold flat:
xF =  2309.9d
yF = 0d
zF = -78.500d
;Spectral orders:
orders =[-1, 0, 1]
;Detectors at each spectral order:
yd = [-136.7d,      0.0d,    136.7d ]
xd = [4734.1d,   4737.1d,   4734.1d ]
zd = [-160.88d,  -160.99d,  -160.88d]
;Aperture stop (square)
stop_width = 80.0d

;-----------------------------------------------;
```

```
;               WFE calculation                 ;
;-----------------------------------------------;
if keyword_set(optimize) then begin
   print, 'determining and removing tip/tilt and defocus...'
   ;wfe -= fit_spherical_wave(wfe)
   ;wfe -= fit_spherical_wave(wfe)

   moses_optimize, m, lambda=lambda, df_best=df, $
      rmswfe=rmswfe, wfe=WFE, prior_wfe=prior_wfe, N=N
   return, WFE
endif

;Process the input spectral order, m
ss = where(orders EQ m)
if ss[0] eq -1 then message,'Error: order m is not available.'
i = ss[0] ;The index of the correct order.
m = float(m) ;Let's not have any integer arithmetic.

;Focal position, including optional offset
f = [xd[i], yd[i], zd[i]] + df
focal_length = norm(f)

;Primary mirror (grating) figure, x(y,z)
ones = replicate(1.0, N)
yz_linear = stop_width * (findgen(N)/(N-1.0) - 0.5)
   ;A linear y or z axis (same since primary is square) with N elements.
y = yz_linear # ones
z = ones # yz_linear
x = x0 - sqrt( R^2 - (y-y0)^2 - (z-z0)^2 )

;WFE components (in waves)
WFE_I = -x/lambda  ;propagation to grating surface.
WFE_G = m*y/d      ;diffraction grating in spectral order m.
WFE_O = (sqrt( (f[0]-x)^2 + (f[1]-y)^2 + (f[2]-z)^2 ) - focal_length)/lambda
   ;Careful, that's letter O! Propagation from (x,y,z) to f.
   ;Note that I've subtracted out the focal length itself.
WFE = WFE_I + WFE_G + WFE_O + prior_wfe ;Total WFE
WFE -= mean(WFE)  ;remove piston, which is physically meaningless.
```

```
rmswfe = sqrt(mean( WFE^2 ) )

return, WFE

end
```

D.2 Source Code: moses_ideal.pro

```
;NAME:
;  moses_ideal
;PURPOSE:
;  Simple emonstration of moses_wfe for ideal MOSES (as designed).
;CALLING SEQUENCE:
;  .run moses_ideal
;MODIFICATION HISTORY:
;  2009-Jun-01  CCK
;  2009-Jun-02  CCK fixed WFE display bug.

N = 256 ;size of mesh
lambda = 303.8e-7 ;wavelength in mm (He II)

;Calculate system WFE at best focus for each order.
wfem = moses_wfe(-1, N=N, /optimize, lambda=lambda, rmswfe=rmswfem, df=dfm)
wfez = moses_wfe( 0, N=N, /optimize, lambda=lambda, rmswfe=rmswfez, df=dfz)
wfep = moses_wfe(+1, N=N, /optimize, lambda=lambda, rmswfe=rmswfep, df=dfp)

;Visualize WFE in all 3 spectral orders.
set_plot,'ps'
device, /encapsulated, filename='ideal_wfe.eps', /color, bits=8, $
   xsize=18, ysize=6 ;bounding box size in cm.
loadct, 4
;window, 0, xsize = 3*N, ysize = N
images = [[[wfem]], [[wfez]], [[wfep]]]
range=ceil(max(abs(images)))
   ;colorbar does not do well with uneven or non-integer ranges.
images = bytscl(images, min=-range, max=range)
!p.multi = [0,3,1] ;3 columns and 1 row of plots
plot_image, images[*,*,0], origin=[-40,-40], scale = 80.0/(N-1), charsize=1, $
```

```
   /noscale, xtitle='y (mm)', ytitle='z (mm)'
plot_image, images[*,*,1], origin=[-40,-40], scale = 80.0/(N-1), charsize=1, $
   /noscale, xtitle='y (mm)', ytitle='z (mm)'
plot_image, images[*,*,2], origin=[-40,-40], scale = 80.0/(N-1), charsize=1, $
   /noscale, xtitle='y (mm)', ytitle='z (mm)'
colorbar, range=[-range, range], color=0, /vertical, divisions = range, $
   position = [0.94, 0.20, 0.96, 0.85]
;write_png, 'ideal_wfe_maps.png', tvrd(/true)
device,/close
set_plot,'x'

print
print,'MOSES_IDEAL results'
print,'m         rmswfe        delta_x      delta_y      delta_z'
print,'-1:',rmswfem,dfm
print,' 0:',rmswfez,dfz
print,'+1:',rmswfep,dfp
print

;Calculate point spread functions
fratio = 9480.0/2.0/80.0 ;MOSES f-ratio (used for all 3 orders)
lambda *= 1e3 ;convert wavelength from mm to microns.
psfm = wfe2psf(wfem, fratio=fratio, lambda=lambda, xarr=xarr, dx=dx)
psfz = wfe2psf(wfez, fratio=fratio, lambda=lambda)
psfp = wfe2psf(wfep, fratio=fratio, lambda=lambda)

;3-panel EPS figure showing point spread functions
pix = 13.5  ;size of pixel in microns (Marconi/E2V spec)
set_plot,'ps'
device, /encapsulated, filename='ideal_psf.eps', /color, bits=8, $
   xsize=18, ysize=6 ;bounding box size in cm.
;window, 1, xsize=1440, ysize=512
loadct, 0
!p.multi = [0,3,1] ;3 columns and 1 row of plots
plot_image, -psfm, origin=[xarr[0],xarr[0]], scale = dx, charsize=1, $
   xtitle='delta_y (microns)', ytitle='delta_z (microns)'
oplot, (pix/2)*[-1,-1,1,1,-1], (pix/2)*[-1,1,1,-1,-1] ;pixel outline
plot_image, -psfz, origin=[xarr[0],xarr[0]], scale = dx, charsize=1, $
```

```
   xtitle='delta_y (microns)', ytitle='delta_z (microns)'
oplot, (pix/2)*[-1,-1,1,1,-1], (pix/2)*[-1,1,1,-1,-1] ;pixel outline
plot_image, -psfp, origin=[xarr[0],xarr[0]], scale = dx, charsize=1, $
   xtitle='delta_y (microns)', ytitle='delta_z (microns)'
oplot, (pix/2)*[-1,-1,1,1,-1], (pix/2)*[-1,1,1,-1,-1] ;pixel outline
device, /close
set_plot,'x'

end
```

D.3 Source Code: moses_defocus.pro

```
;NAME:
;  MOSES_DEFOCUS
;PURPOSE:
;  Simulate a MOSES focus series.
;CALLING SEQUENCE:
;  moses_defocus
;OPTIONAL KEYWORD INPUTS:
;  save_file -- name of save file containing all the results. The default
;     is 'moses_defocus.sav'.
;  N -- size of square mesh (default 384)
;  lambda -- wavelength in mm (default 303.8e-7, He II Ly alpha)
;  Nfsteps -- number of focus steps (default 51)
;  fstep -- focus step size in mm (default 0.25)
;  wfe_mfg -- WFE due to manufacturing & alignment (waves)
;  quiet -- if set, suppress prints & plots of useful information.
;ALGORITHM:
;  First, the design WFEs and focal positions are calculated using moses_wfe
;  with the /optimize option. Then, two independent methods are used to
;  generate the wavefront error corresponding to defocus:
;  (1) displacement of the focal position through keyword df to moses_wfe;
;  (2) application of the general defocus function to the initial WFEs.
;MODIFICATION HISTORY:
;  2009-Jun-10  C. Kankelborg
;  2009-Jun-12  CCK, save file is now gzipped.
;  2009-Jul-06  CCK, commented out the gzip. It was not worth the time.
```

```
pro moses_defocus, N=N, lambda=lambda, Nfsteps=Nfsteps, fstep=fstep, $
   wfe_mfg=wfe_mfg, save_file=save_file, quiet=quiet

;Simulation parameters
if not keyword_set(N) then N = 256+128 ;size of mesh
if not keyword_set(lambda) then lambda = 303.8e-7 ;wavelength in mm (He II)
if not keyword_set(Nfsteps) then Nfsteps = 51 ;number of focus steps
if not keyword_set(fstep) then fstep = 0.25 ;focus step size (mm)
if not keyword_set(save_file) then save_file = 'moses_defocus.sav'
aperture = 80 ;MOSES aperure diameter in mm

;Design detector positions [x,y,z] at each spectral order
;(based on moses_wfe.pro constants xd, yd, zd):
detm = [4734.1, -136.7, -160.88]
detz = [4737.1,    0.0, -160.99]
detp = [4734.1,  136.7, -160.88]
;Work out approximate unit vectors from grating center toward each focus.
;I could correct for exact focus positions using dfm/dfz/dfp, but
;it's not worth it.
uvm = detm / norm(detm) ;unit vector toward m = -1
uvz = detz / norm(detz) ;unit vector toward m =  0
uvp = detp / norm(detp) ;unit vector toward m = +1
fratio = norm(detz) / aperture ;fratio is needed later by wfe2psf.

;Work out the WFE & location of design best focus.
wfem = moses_wfe(-1, N=N, /optimize, lambda=lambda, rmswfe=rmswfem, $
   df=dfm, wfe_grating = wfe_mfg)
wfez = moses_wfe( 0, N=N, /optimize, lambda=lambda, rmswfe=rmswfez, $
   df=dfz, wfe_grating = wfe_mfg)
wfep = moses_wfe(+1, N=N, /optimize, lambda=lambda, rmswfe=rmswfep, $
   df=dfp, wfe_grating = wfe_mfg)

;---------------------------------------------------------------
;Allocate arrays for WFE and PSF results (3 orders, 2 methods):
;---------------------------------------------------------------
wfems1 = fltarr(N, N, Nfsteps) ;WFE m = -1 image stack, method 1
wfezs1 = fltarr(N, N, Nfsteps) ;WFE m =  0 image stack, method 1
wfeps1 = fltarr(N, N, Nfsteps) ;WFE m = +1 image stack, method 1
```

```
;-------------------------------------------------------------
psfms1 = fltarr(N, N, Nfsteps) ;PSF m = -1 image stack, method 1
psfzs1 = fltarr(N, N, Nfsteps) ;PSF m =  0 image stack, method 1
psfps1 = fltarr(N, N, Nfsteps) ;PSF m = +1 image stack, method 1
;-------------------------------------------------------------
wfems2 = fltarr(N, N, Nfsteps) ;WFE m = -1 image stack, method 2
wfezs2 = fltarr(N, N, Nfsteps) ;WFE m =  0 image stack, method 2
wfeps2 = fltarr(N, N, Nfsteps) ;WFE m = -+ image stack, method 2
;-------------------------------------------------------------
psfms2 = fltarr(N, N, Nfsteps) ;PSF m = -1 image stack, method 2
psfzs2 = fltarr(N, N, Nfsteps) ;PSF m =  0 image stack, method 2
psfps2 = fltarr(N, N, Nfsteps) ;PSF m = +1 image stack, method 2
;-------------------------------------------------------------
rmswfems1 = fltarr(Nfsteps) ;RMS WFEs, m = -1
rmswfezs1 = fltarr(Nfsteps) ;RMS WFEs, m =  0
rmsWFEps1 = fltarr(Nfsteps) ;RMS WFEs, m = +1
;-------------------------------------------------------------

;Calculate WFE and PSF for all focus positions
fpositions = fstep*( findgen(Nfsteps) - 0.5*(Nfsteps-1) ) ;focus positions
if not keyword_set(quiet) then window, 10, title='PSF', xsize=3*N, ysize=2*N
for i=0, Nfsteps-1 do begin
   ;Method 1
   ;m = -1
   wfems1[*,*,i] = moses_wfe(-1, N=N, lambda=lambda, rmswfe=rmswfe, $
      df=dfm+uvm*fpositions[i], wfe_grating = wfe_mfg)
   rmswfems1[i] = rmswfe ;can't do this directly in the keyword to moses_wfe,
      ;because IDL is stupid (doesn't pass array elements by reference)!!!
   psfms1[*,*,i] = wfe2psf(wfems1[*,*,i], fratio=fratio, lambda=lambda, $
      xarr=xarr, dx=dx)
   ;m = 0
   wfezs1[*,*,i] = moses_wfe( 0, N=N, lambda=lambda, rmswfe=rmswfe, $
      df=dfz+uvz*fpositions[i], wfe_grating = wfe_mfg)
   rmswfezs1[i] = rmswfe ;can't do this directly in the keyword to moses_wfe,
      ;because IDL is stupid (doesn't pass array elements by reference)!!!
   psfzs1[*,*,i] = wfe2psf(wfezs1[*,*,i], fratio=fratio, lambda=lambda, $
      xarr=xarr, dx=dx)
   ;m = +1
```

```
   wfeps1[*,*,i] = moses_wfe(+1, N=N, lambda=lambda, rmswfe=rmswfe, $
      df=dfp+uvp*fpositions[i], wfe_grating = wfe_mfg)
   rmswfeps1[i] = rmswfe ;can't do this directly in the keyword to moses_wfe,
      ;because IDL is stupid (doesn't pass array elements by reference)!!!
   psfps1[*,*,i] = wfe2psf(wfeps1[*,*,i], fratio=fratio, lambda=lambda, $
      xarr=xarr, dx=dx)

   ;Method 2 (much more streamlined!)
   ;m = -1
   wfems2[*,*,i] = defocus(wfem, fpositions[i], fratio, lambda)
   psfms2[*,*,i] = wfe2psf(wfems2[*,*,i], fratio=fratio, lambda=lambda)
   ;m =  0
   wfezs2[*,*,i] = defocus(wfez, fpositions[i], fratio, lambda)
   psfzs2[*,*,i] = wfe2psf(wfezs2[*,*,i], fratio=fratio, lambda=lambda)
   ;m = +1
   wfeps2[*,*,i] = defocus(wfep, fpositions[i], fratio, lambda)
   psfps2[*,*,i] = wfe2psf(wfeps2[*,*,i], fratio=fratio, lambda=lambda)

   if not keyword_set(quiet) then begin
      print, 'Completed focus position ',i,', defocus = ',fpositions[i],', mm.'
      print, 'RMS WFE (m,z,p) = ', rmswfems1[i], rmswfezs1[i], rmswfeps1[i]
      print, 'displaying PSFs...'
      tv, [[bytscl(psfms1[*,*,i]), bytscl(psfzs1[*,*,i]), bytscl(psfps1[*,*,i])],$
           [bytscl(psfms2[*,*,i]), bytscl(psfzs2[*,*,i]), bytscl(psfps2[*,*,i])]]
   endif

endfor


save, filename = save_file
;spawn, 'gzip -f ' + save_file
end
```

D.4 Source Code: mdef_analyze.pro

```
;NAME:
;  mdef_analyze
;PURPOSE:
```

```
;  Restore the results of moses_defocus (assumes default save filename), and
;  produce images and plots to document the results.
;CALLING SEQUENCE:
;  mdef_analyze
;MODIFICATION HISTORY:
;  2009-Jun-xx  C. Kankelborg
;  2009-Jul-06  CCK commented out the gzip stuff (as in moses_defocus.pro).

pro mdef_analyze
;message,'Unzipping the save file...',/informational
;spawn, 'gunzip -f moses_defocus.sav.gz'
message,'Restoring the save file...',/informational
restore, 'moses_defocus.sav'
;message,'Rezipping the save file...',/informational
;spawn, 'gzip -f moses_defocus.sav'

gamma = 0.5 ;contrast parameter
imfile = '/tmp/mdef' ;stem used for image filenames

;Work out rms spot radius (using method 2 results)
message,'Analyzing PSF images...',/informational
rmsradm = fltarr(Nfsteps)
rmsradz = fltarr(Nfsteps)
rmsradp = fltarr(Nfsteps)
for i=0, Nfsteps-1 do begin
   rmsradm[i] = psf2stats(psfms2[*,*,i], dx=dx)
   rmsradz[i] = psf2stats(psfzs2[*,*,i], dx=dx)
   rmsradp[i] = psf2stats(psfps2[*,*,i], dx=dx)
endfor

;RMS WFE PLOT
set_plot,'ps'
device, /encapsulated, filename='mdef_rmswfe.eps', /color, bits=8
plot, fpositions, rmswfems1, linestyle=0, psym=-7, title='MOSES Focus (WFE)', $
   ytitle='RMS wavefront error (waves)', xtitle='defocus (mm)'
oplot, fpositions, rmswfezs1, linestyle=0, psym=-4
oplot, fpositions, rmswfeps1, linestyle=0, psym=-1
legend,['m = -1','m =  0','m = +1'], linestyle=[0,0,0], psym=[-7,-4,-1]
```

```
device,/close
set_plot,'x'

;RMS Spot Radius PLOT
set_plot,'ps'
device, /encapsulated, filename='mdef_rmsrad.eps', /color, bits=8
plot, fpositions, rmsradm, linestyle=0, psym=-7, title='MOSES Focus (Spot Radius)', $
   ytitle='RMS spot radius (mm)', xtitle='defocus (mm)'
oplot, fpositions, rmsradz, linestyle=0, psym=-4
oplot, fpositions, rmsradp, linestyle=0, psym=-1
;For comparison, what is the RMS spot radius of a pyramidal beam?
rmsrad_1sq = 1.0/sqrt(6.0) ;rms radius of a unit square
rmsrad_pyramid = rmsrad_1sq * abs(fpositions)/fratio
   ;rms radius of ideal geometrical beam (square cross-section)
oplot, fpositions, rmsrad_pyramid, linestyle=2, psym=-3
legend,['m = -1','m =  0','m = +1','Ideal ray optics'], $
   linestyle=[0,0,0,2], psym=[-7,-4,-1,-3], /bottom
device,/close
set_plot,'x'

;FOCUS MOVIE
message,'Processing image frames...',/informational
set_plot,'z' ;do this all in the z-buffer.
device, set_resolution = [3*N, 2*N]
for i=0, Nfsteps-1 do begin
   image = [[bytscl(psfms1[*,*,i]^gamma), $
             bytscl(psfzs1[*,*,i]^gamma), $
             bytscl(psfps1[*,*,i]^gamma)],$
            [bytscl(psfms2[*,*,i]^gamma), $
             bytscl(psfzs2[*,*,i]^gamma), $
             bytscl(psfps2[*,*,i]^gamma)]]
   tv, image
   legend = 'defocus = '+string(fpositions[i],format='(f+6.2)')+' mm'
   xyouts, 3*N-4, 4, /device, legend, alignment=1.0
   legend = 'gamma = '+string(gamma,format='(f+6.2)')
   xyouts, 3*N-4, 4+N, /device, legend, alignment=1.0
   xyouts, 4, 4, /device, 'Method 1 (df keyword)'
   xyouts, 4, 4+N, /device, 'Method 2 (defocus.pro)'
```

```
   xyouts, N/2, 2*N-10, /device, 'm = -1', alignment=0.5
   xyouts, N/2+N, 2*N-10, /device, 'm = 0', alignment=0.5
   xyouts, N/2+2*N, 2*N-10, /device, 'm = +1', alignment=0.5
   filename = imfile+string(i,format='(i03)')+'.png'
   write_png, filename, tvrd(0, 0, 3*N, 2*N)
endfor

message,'Converting images to movie mdef.m2v...', /informational
spawn,'convert -quality 100 '+imfile+'*.png ./mdef.m2v'

end
```

D.5 Source Code: moses_test1.pro

```
;NAME:
;  moses_test1
;PURPOSE:
;  Simple demonstration of moses_wfe for MOSES with added astigmatism.
;CALLING SEQUENCE:
;  .run moses_test1
;MODIFICATION HISTORY:
;  2009-Jun-02  CCK, based on moses_ideal.pro

N = 512 ;size of mesh (enlarged to keep PSF within field of view)
lambda = 303.8e-7 ;wavelength in mm (He II)

;Construct WFE array corresponding to
;     x = (design sphere) + A*y*z,
;where x,y,z are in mm, and A = 1e-7 / mm.
coord = 80*(findgen(N)/(N-1) - 0.5) ;coordinate axis (y or z) on 80mm grating.
ones = replicate(1.0, N)
z = ones # coord  ;y coordinates for entire grating surface
y = coord # ones  ;x coordinates for entire grating surface
wfe_grating = 1e-7 * y * z * (2 / lambda)

;Calculate system WFE at best focus for each order.
wfem = moses_wfe(-1, N=N, /optimize, lambda=lambda, rmswfe=rmswfem, $
   df=dfm, wfe_grating = wfe_grating)
```

```
wfez = moses_wfe( 0, N=N, /optimize, lambda=lambda, rmswfe=rmswfez, $
   df=dfz, wfe_grating = wfe_grating)
wfep = moses_wfe(+1, N=N, /optimize, lambda=lambda, rmswfe=rmswfep, $
   df=dfp, wfe_grating = wfe_grating)

;Visualize WFE in all 3 spectral orders.
set_plot,'ps'
device, /encapsulated, filename='test1_wfe.eps', /color, bits=8, $
   xsize=18, ysize=6 ;bounding box size in cm.
loadct, 4
;window, 0, xsize = 3*N, ysize = N
images = [[[wfem]], [[wfez]], [[wfep]]]
range=ceil(max(abs(images)))
   ;colorbar does not do well with uneven or non-integer ranges.
images = bytscl(images, min=-range, max=range)
!p.multi = [0,3,1] ;3 columns and 1 row of plots
plot_image, images[*,*,0], origin=[-40,-40], scale = 80.0/(N-1), charsize=1, $
   /noscale, xtitle='y (mm)', ytitle='z (mm)'
plot_image, images[*,*,1], origin=[-40,-40], scale = 80.0/(N-1), charsize=1, $
   /noscale, xtitle='y (mm)', ytitle='z (mm)'
plot_image, images[*,*,2], origin=[-40,-40], scale = 80.0/(N-1), charsize=1, $
   /noscale, xtitle='y (mm)', ytitle='z (mm)'
colorbar, range=[-range, range], color=0, /vertical, divisions = range, $
   position = [0.94, 0.20, 0.96, 0.85]
;write_png, 'test1_wfe_maps.png', tvrd(/true)
device,/close
set_plot,'x'

print
print,'MOSES_test1 results'
print,'m         rmswfe        delta_x      delta_y      delta_z'
print,'-1:',rmswfem,dfm
print,' 0:',rmswfez,dfz
print,'+1:',rmswfep,dfp
print

;Calculate point spread functions
fratio = 9480.0/2.0/80.0 ;MOSES f-ratio (used for all 3 orders)
```

```
lambda *= 1e3 ;convert wavelength from mm to microns.
psfm = wfe2psf(wfem, fratio=fratio, lambda=lambda, xarr=xarr, dx=dx)
psfz = wfe2psf(wfez, fratio=fratio, lambda=lambda)
psfp = wfe2psf(wfep, fratio=fratio, lambda=lambda)

;3-panel EPS figure showing point spread functions
pix = 13.5  ;size of pixel in microns (Marconi/E2V spec)
set_plot,'ps'
device, /encapsulated, filename='test1_psf.eps', /color, bits=8, $
   xsize=18, ysize=6 ;bounding box size in cm.
;window, 1, xsize=1440, ysize=512
loadct, 0
!p.multi = [0,3,1] ;3 columns and 1 row of plots
plot_image, -psfm, origin=[xarr[0],xarr[0]], scale = dx, charsize=1, $
   xtitle='delta_y (microns)', ytitle='delta_z (microns)'
oplot, (pix/2)*[-1,-1,1,1,-1], (pix/2)*[-1,1,1,-1,-1] ;pixel outline
plot_image, -psfz, origin=[xarr[0],xarr[0]], scale = dx, charsize=1, $
   xtitle='delta_y (microns)', ytitle='delta_z (microns)'
oplot, (pix/2)*[-1,-1,1,1,-1], (pix/2)*[-1,1,1,-1,-1] ;pixel outline
plot_image, -psfp, origin=[xarr[0],xarr[0]], scale = dx, charsize=1, $
   xtitle='delta_y (microns)', ytitle='delta_z (microns)'
oplot, (pix/2)*[-1,-1,1,1,-1], (pix/2)*[-1,1,1,-1,-1] ;pixel outline
device, /close
set_plot,'x'

end
```

D.6 Source Code: moses_test2.pro

```
;Simple MOSES focus series test
;  % ssw_batch moses_test2 moses_test2.log /date

set_plot, 'z' ;Prevent firing up any x-windows

message,'Calculating MOSES focus series...', /info
moses_defocus, /quiet  ;the /quiet feature eliminates real-time plotting to x.

message,'Producing plots and images...', /info
```

```
mdef_analyze

end
```

D.7 Source Code: moses_test3.pro