Fresnel numerical simulation of adaptive optics systems

Level 4H 2001/2

Department of Physics

1) Abstract

A simulator to model adaptive optics system including diffraction, scintillation and project effects due to non normal incidence of light onto optical components has been produced. The simulator uses a wave optics approach that produces predictions in agreement with a statistical model produced by Dr R.W. Wilson for a range of angular displacements between the science object and guide star. The use of scalar diffraction theory is analysed and the restriction to collimated beams is explained. The common techniques of far field imaging and finite lens impulse imaging are justified by the impracticality of propagating an optical field to an image plane with or without various co-ordinate system techniques investigated by the Author.

A Kolmogorov turbulence generator with the Authors low order mode correction is used to supply atmospheric turbulence. Distortion due to turbulence is demonstrated to increase the point spread function size and a ~50 times reduction in the peak image intensity. Atmospheric compensation is demonstrated for science objects at a range of angular displacements from the natural guide star showing angular anisoplanatism. Scintillation is demonstrated along with the corresponding reduction in the efficiencey of atmospheric compensation this causes. The presence of a ring about the 'corrected' point spread function is discussed and it is concluded that this is not sufficient evidence for diffraction limited imaging. The isoplanatic angle is shown to correspond to a 'corrected' PSF with a Strehl of 83% of the on-axis 'corrected' PSF Strehl.

2) Contents

3) Introduction

3.1) Aims and objectives

The purpose of this work is to produce a numerical simulation for the performance evaluation of adaptive optics (AO) systems for use in astronomical telescopes. Propagation and projection effects within the atmosphere and telescope are to be included in the model as these are not included in available existing simulations. The limitations of the numerical simulation devised by the author is explained giving the representable situations that may be correctly modelled. This approach to modelling is designed to be applicable to both classical AO and multi-conjugate AO systems. For the purposes of its development it will be tested against a statistical model of a classical AO system developed by Dr R. W. Wilson.

The image degradation caused by atmospheric turbulence is demonstrated, along with the effect of including a classical AO system to provide atmospheric compensation. For some typical atmospheric conditions ($r_0 = 0.1$m, see section 4.2), the point spread function and image Strehl will be determined as a gauge of the simulated AO system performance. This is then compared to the existing statistical nonpropagation model to demonstrate the validity of the simulation.

Sections 3-5 provide the reader with the information required to understand adaptive optics and the approach used to simulate it. Section 6 describes previous and currently ongoing simulation efforts. Section 7 discusses the inherent limitations of the simulation approach along with the techniques the Author has used to overcome the simulation limitations. Section 8 is a description of the simulation implementation along with some testing results that may be reproduced by the reader. Sections 9-12 are an analysis of a simulated classical adaptive optics system to demonstrate the functionality of the simulator.

3.2) Nomenclature

Large ground-based optical and near infra-red telescopes are reflecting instruments that use mirrors to collect the light and form primary and other foci. Mirrors are used in large telescopes primarily because they are easier to make and support than large transmissive elements. Figure 1 shows a reflecting telescope with a Cassegrain focus[1].

The science object is the object of interest in the sky. To record an image, charged coupled devices[1] (CCDs) are now used in the optical band ($\lambda<1\mu$m) (see figure 2) and HgCdTe detectors ($\lambda<2.5\mu$m) or InSb based detectors ($\lambda<5\mu$m) for the near infra-red band. However most science objects are not bright sources and the imaging equipment requires a significant length of time (integration time) to collect sufficient photons so that the recorded image is sufficiently bright (achieving a high signal to noise ratio). This provides the requirement that the mount must move and rotate the telescope to keep the starfield in an apparently fixed position and orientation relative to the imaging equipment to prevent smearing of any images produced. A larger primary mirror means more photons from the science object are collected, but there are technical obstacles

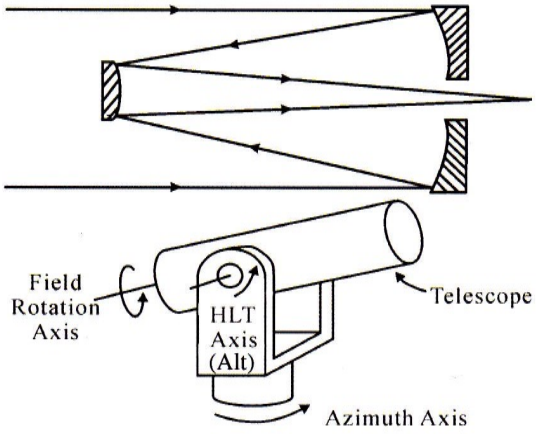
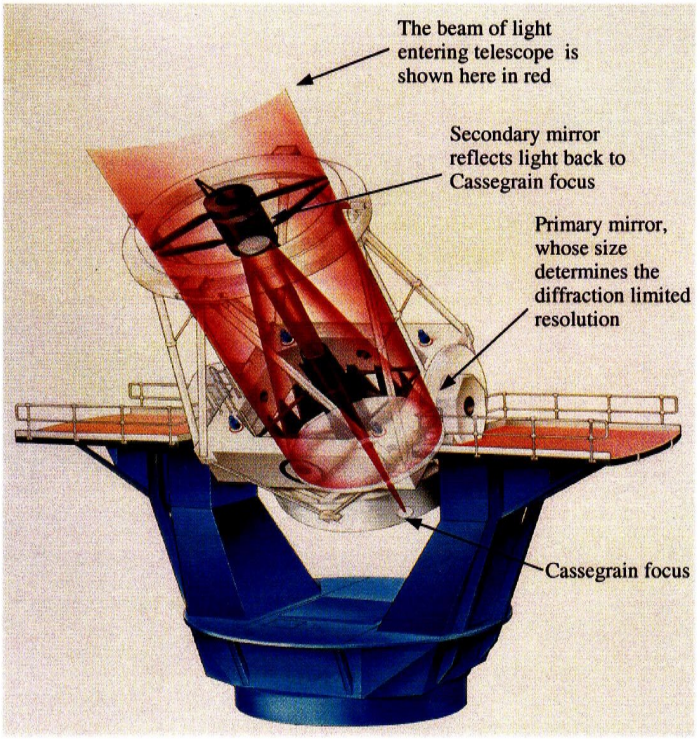


Figure 1[1]

The mirror configuration is shown above for a Classical Cassegrain focus. A parabodial primary mirror (the first mirror to reflect the light, on the right of the diagram) and a convex hyperboloidal secondary mirror (the smaller mirror, on the left of the diagram) is used to form the focus that projects out through a hole in the centre of the primary.

The mount of the WHT is shown. It is known as an Alt (Altitude)-Az(Azimuth) mount because of the orientation of the rotation axes.

On the right is a diagram of the William Hershel Telescope which has a Cassegrain focus. (Image courtesy of the Issac Newton Group of telescopes, La Palma)



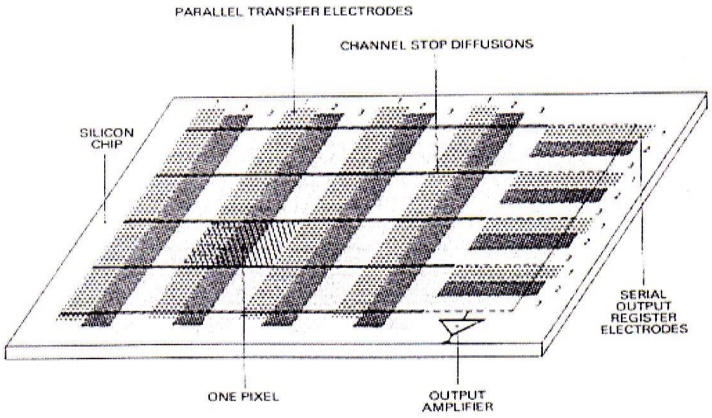


Figure 2[1]

A simplified diagram of a CCD showing 16 pixels and readout connections.

During exposure incident photons create electron-hole pairs. The electrons are trapped in wells beneath the electrodes. Only once an exposure is complete are the collected charges transferred out of the chip one at a time and amplified. There is a finite readout noise introduced during the transfer of collected charges from the chip, which is dependent on the CCD used. For high frequency imaging (short times between readouts), the pixels must be sufficiently large so that the accumulated charge per pixel is not masked by the readout noise.

in producing an arbitrarily large mirror. Increasing the integration time also allows more photons to be collected, but there is a point of declining return due to some sources of noise in the imaging equipment (for example due to radiation or cosmic rays).

As the optical beam is being brought to focus, its width is reducing at a rate described by the speed of the beam. The speed of the beam is the focal ratio f/D (f: focal length, D: diameter). A fast beam has a small focal ratio and is brought to focus relatively quickly. The field of view (FoV) for a telescope is the solid angle of the sky over which an image can be recorded. This is limited by 'off axis' aberrations viewing and vignetting. Vignetting is the obstruction of the beam by telescope components as the path the optical beam follows moves away from the 'on axis' position.

The 'seeing' obtained by a telescope is the angular resolution achieved by the telescope during observations. The achieved resolution is not only dependent on the primary collection mirror (producing an airy disk due to diffraction) or the pixel density in the CCD (used to detect the image) but also on the atmosphere. Atmospheric turbulence produces random fluctuations in the refractive index of the atmosphere, causing light that passes through it to be refracted. This distorts the image

produced by the telescope. The effect of this can be clearly seen in figure 3[2] which shows surface intensity plots of the images formed from a point source. These plots are known as the point spread functions (PSF) for the telescope in question. They show how the energy from a point source becomes distributed in the image. Atmospheric turbulence increases the image area the energy is spread over, reducing resolution, reducing the peak image intensity. Strehl is the ratio of the achieved peak image intensity under real conditions to that in a diffraction limited image (no distortions from turbulence). A higher Strehl. indicates that the image formed is closer to a diffraction limited image.

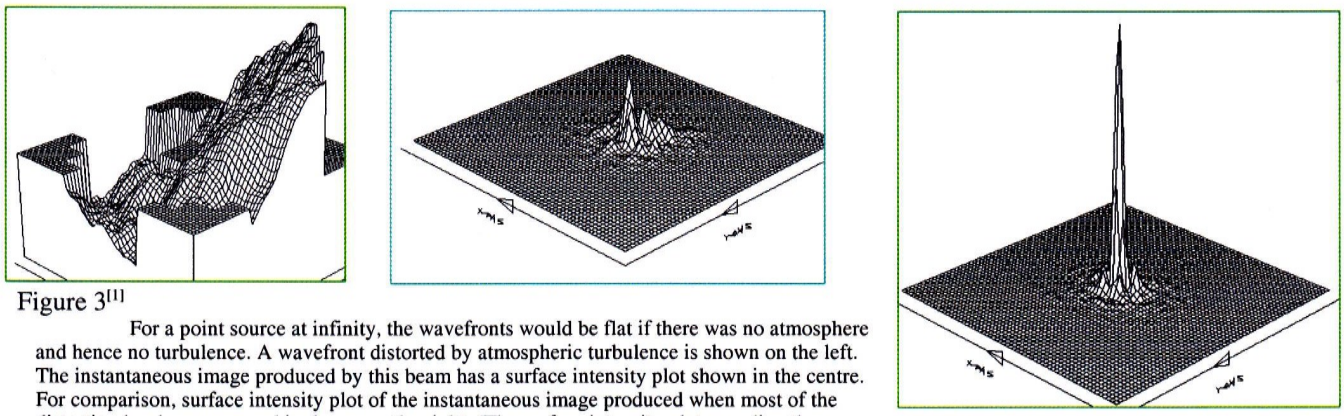


Figure 3[1]

For a point source at infinity, the wavefronts would be flat if there was no atmosphere and hence no turbulence. A wavefront distorted by atmospheric turbulence is shown on the left. The instantaneous image produced by this beam has a surface intensity plot shown in the centre. For comparison, surface intensity plot of the instantaneous image produced when most of the distortion has been removed is shown on the right. (The surface intensity plots are directly comparable)

The nomenclature the Author uses for some mathematical processes should be noted. $\Im[g]$ is the fourier transform of g. $\Im[g(x,y)]$ is the fourier transform of g across the variables x and y. $\Im[g]_{y\to b}^{x\to a}$ is the fourier transform of g, changing the variables x,y into a,b. Note that x,y,a & b could be functions themselves. $g*h$ is the convolution of function g and h. If any variables are specified then the convolution is over those variables.

4) Background

4.1) Optical propagation

Scalar diffraction theory[2] describes light ignoring it's polarisation. The assumption made is that the scalar amplitude of one transverse component of a single polarisation will describe the optical field sufficiently accurately to model a system. This is only possible if there is no polarisation dependance in the system of interest. This clearly prevents considerations of optically active components or a quantum mechanical model of absorption or emission. The environment is being restricted to be homogeneous and charge-free for the propagation of light. Structure in the environment will be introduced at a later point.

Maxwell's equations clearly describe that the electric and magnetic fields are coupled and cannot be isolated for a rigourous model. However, scalar diffraction theory provides very accurate results providing certain conditions are met:

1. Any diffracting aperture must be large compared to the wavelength of light involved.
2. A diffracted field must not be observed too close to the aperture.

These conditions are well satisfied for telescope simulation, allowing a consideration of the simpler scalar optical field.

For a system of point sources producing a monochromatic field, the complex valued amplitude field across an xy plane can be represented by **U(x,y,0)**. For a steady state situation the complex valued amplitude field in the infinite space can be determined from this plane. More specifically the complex valued amplitude across another xy plane, **U(x,y,z)**, can be determined using a linear spatial filter[2] (see equation 1). As the filter is linear, fourier transform techniques can be used to propagate the optical field from normal to the x,y,z=0 plane to another x,y,z plane.

The fourier decomposition of the optical field in the x,y,z=0 plane corresponds to the intersection of a series of 3D monochromatic plane waves with the x,y,z=0 plane. As the wavelength is exactly known, the 3D plane wave corresponding to each fourier component can be deduced (see figure 4). As 3D plane waves exist over all space, the projection into any other plane can be deduced. The propagation of the optical field can then be performed by assuming it is quasi-static such that the propagation velocity is much faster than any environmental change. This uses the linear superposition of the optical field (i.e. superposition of the quantity given by the phase vectors, complex amplitudes or complex scalar - 3 different terms for the same quantity)

To propagate from the x,y,z=0 to the x,y,z=z_1 plane, each fourier component must be 'adjusted'. It can easily be seen from figure 4 that the 'adjustment' required is a change of its phase (i.e. moving the components origin). The limitation of this approach is that it is implicitly assumed that the optical field is propagating in a direction close to the +ve z direction with a maximum angle away from +ve z direction that is dependant on the highest spatial frequency resolvable.

The required spatial filter can then be deduced form the geometry of the plane waves as shown in figure 4.

$$\Im[U(x,y,z)]=H(f_x, f_y, z)*\Im[U(x,y,0)] \quad \text{where} \quad H(f_x, f_y, z) = \begin{cases} \exp^{2j\pi\frac{z}{\lambda}\sqrt{1-(\lambda f_x)^2-(\lambda f_y)^2}} & f_x^2+f_y^2<\frac{1}{\lambda^2} \\ 0 & \textit{otherwise} \end{cases}$$

(equation 1)

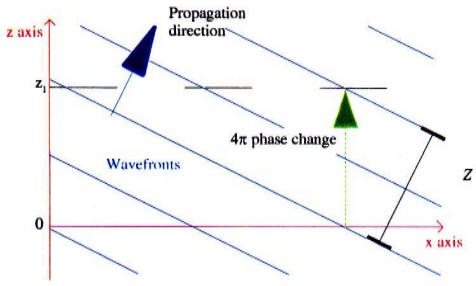


Figure 4
Slice perpendicular to y axis through an optical field showing the wavefronts (surfaces of constant phase). Propagation is normal to the wavefronts, in the +z, +x direction. In this case, propagation from the x,y,z=0 plane to the x,y,z=z plane is a 4π phase change.

For $(f_x,f_y) = (0,0)$, this corresponds to a plane wave propagating precisely along the +ve z axis.

For $0<f_x^2+f_y^2<\frac{1}{\lambda^2}$ this corresponds to a plane wave propagating at an angle to the +ve z axis so that there is some oscillation along the xy plane. The exponential factor in the transfer function is the phase change in propagating to the next plane.

For $f_x^2+f_y^2>\frac{1}{\lambda^2}$ this clearly corresponds to some oscillations in the xy plane with frequency greater than the frequency of light under consideration. These are evanescent waves and so long as the z propagation distance is at least a few wavelengths they will die away to zero. They are therefore explicitly removed in the transfer function $\mathbf{H(f_x,f_y,z)}$.

Note that this analysis automatically treats an optical field as coherent. This is acceptable for constructing point spread functions for a telescope design, but unacceptable for imaging of unrelated objects as interference would be simulated when it should not be.

4.2) Atmospheric model

The 'seeing' is due to the aberrations caused by atmospheric turbulence. Atmospheric turbulence reduces the spatial correlation of light passing through it because of atmospheric refractive index variations, adding local phase variations to the optical field. The Fried parameter, $\mathbf{r_0}$, is the spatial correlation length and is the aperture diameter across which there is approximately 1 radian of rms. phase aberration. The Fried parameter is defined (by Fried) to give the resolution limit on any image formed using $\alpha \approx 1.22\lambda/r_0$. The Rayleigh resolution criterion for angular resolution, for an aperture diameter **D** at wavelength λ, is $\alpha \approx 1.22\lambda/D$. Thus a telescope with $\mathbf{D>>r_0}$ has approximately the same resolution as a telescope with diameter $\mathbf{r_0}$.[3][4][5] This is known as atmospheric limited seeing

A theoretical description of atmospheric turbulence known as Kolmogorov Turbulence produces refractive index

fluctuations with a power spectrum, $\Phi(\kappa)$, given in equation 2.[3]

$$\Phi(\vec{\kappa}) = 0.033 C_n^2(z) |\vec{\kappa}|^{-11/3} \quad \text{for} \quad 1/L_0 < |\vec{\kappa}| < 1/l_0 \qquad \text{(equation 2)}$$

where $\mathbf{l_0}$ is the inner turbulence scale.
$\mathbf{L_0}$ is the outer turbulence scale.
$\mathbf{C_n \equiv C_n(z)}$ is the refractive index structure constant which is highly dependant on altitude, **z**

In Kolmogorov theory, the outer turbulence scale is the spatial scale at which atmospheric turbulence is generated. This can be 5 to 100m depending on the conditions and the environment. Energy is 'injected' into the turbulence at this scale, which then excites turbulence on a smaller scale. The energy cascades down to smaller and smaller scales until the physical limitation of the speed of sound limits the smallest scale for atmospheric turbulence. This is the inner turbulence scale and of the order of mm to cm. The refractive index structure constant, $\mathbf{C_n(z)}$, is the measure of the strength of the refractive index variations due to atmospheric turbulence at altitude **z**. Knowing the refractive index structure constant for a particular site determines the Fried parameter (see equation 3[3][5]). The Fried parameter is dependent on wavelength, $\mathbf{r_0 \propto \lambda^{6/5}}$, showing that longer wavelengths are less affected by atmospheric turbulence.

$$r_0 = [0.423 k^2 \sec(\xi) \int_0^{z_{max}} C_n^2(z) dz]^{-3/5} \qquad \text{(equation 3)}$$

where $k = \frac{2\pi}{\lambda}$ is the wavevector of light

$\mathbf{l = z \sec(\xi)}$ is the distance along the line of sight to altitude **z** looking at an angle ξ to the zenith
$\mathbf{z_{max}}$ is the top of the atmosphere

The 3 dimensional atmospheric turbulence is modelled as set of planes which contain the contributions a section of the atmosphere (Taylor approximation). It is assumed that each plane is frozen so that the pattern of refractive index variations is unchanged. In order to permit some atmospheric evolution, each plane is moving with the wind velocity. As time time passes the observed atmospheric turbulence pattern changes. A critical time constant, τ_0, is the interval over which the observed atmospheric turbulence is essentially unchanged (see equation 4[5]). The isoplanatic angle, θ_0, which is the angle through which the atmospheric turbulence pattern can be considered as essentially isotropic (see equation 5[5]).

$$\tau_0 = [2.91 k^2 \sec(\xi) \int_0^{z_{max}} C_n^2(z) v^{5/3}(z) dz]^{-3/5} \qquad \text{(equation 4)}$$

$$\vartheta_0 = [2.91 k^2 \sec^{8/3}(\xi) \int_0^{z_{max}} C_n^2(z) z^{5/3} dz]^{-3/5} \qquad \text{(equation 5)}$$

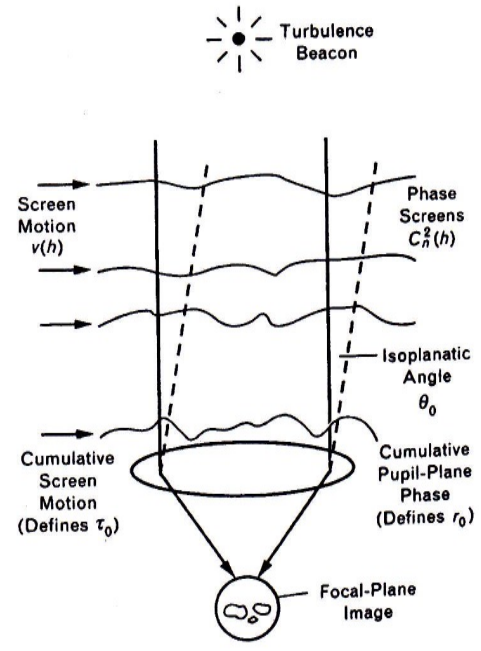


Figure 5
Model atmosphere containing phase screens of strength $C_n^2(z)$, moving with velocity v(z) across the line of sight

4.3) Zernike polynomials

Zernike polynomials (see figure 6) are an orthogonal basis set for functions defined over a unit circle (see equation 6, 7). They are indexed by mode number j and have radial degree $n \geq 0$, and azimuthal frequency $m \geq 0$[8]. The radial degree and azimuthal frequency are determined as follows:

1. Start from 1st mode (j=1).
2. Restrict choices of n,m so that $m \leq n$ and n-m is even.
3. Do not use an n,m combination that has been used before.
4. Choose smallest value of n possible.
5. Choose smallest value of m possible.
6. Repeat from 2) with next mode until all desired modes are indexed

$$Z_{\text{even } j} = \sqrt{n+1}\, R_n^m(r)\sqrt{(2)}\cos(m\vartheta) \qquad m \neq 0$$
$$Z_{\text{odd } j} = \sqrt{n+1}\, R_n^m(r)\sqrt{(2)}\sin(m\vartheta)$$
$$Z_j = \sqrt{n+1}\, R_n^0(r) \qquad m = 0$$

$$R_n^m(r) = \sum_{s=0}^{(n-m)/2} \frac{(-1)^s (n-s)!}{s![(n+m)/2-s]![(n-m)/2-s]!} r^{n-2s}$$

Equation 6
The definition of Zernike polynomials.

$$\int d^2 r\, W(r)\, z_i(r)\, z_j(r) = \delta_{ij}$$

$$W(r) = 1/\pi \quad r \leq 1$$
$$= 0 \quad r > 1$$

Equation 7
The Zernike orthoginality relation. Note that they are defined over the unit circle and must be scaled for any other radius

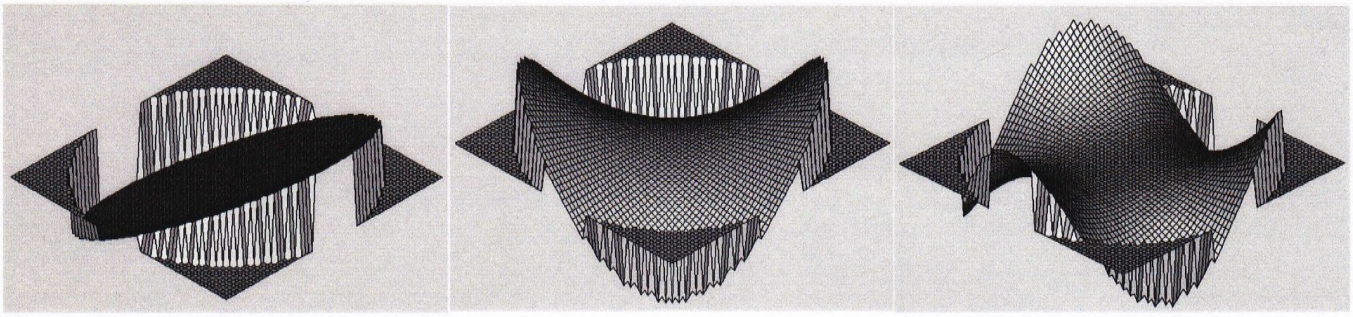


Figure 6
The Zernike modes 2,5 and 9 from left to right respectively. They are shown over a unit circle with the 'z axis' as the function value.

5) AO system design

5.1) Overview

Adaptive optics are a set of techniques that remove aberrations from an optical system using a closed loop feedback system. A closed loop feedback system is one in which a 'correction' is applied based on the monitoring of the system further down the optical path. This means that the system not only observes any aberrations present in the incoming optical field but also it's own 'corrections'. When the 'corrections' maximally cancel out the aberrations present, the system is in it's optimal condition. Attempting to put the system in this optimal condition is the function of the control loop. A control loop that does this is called a 'null seeking servo'. The control loop applies corrections in an attempt to place the system in its 'null' state, i.e. the absence of any aberrations. A diagrammatical representation of an AO system is shown in figure 7[5].

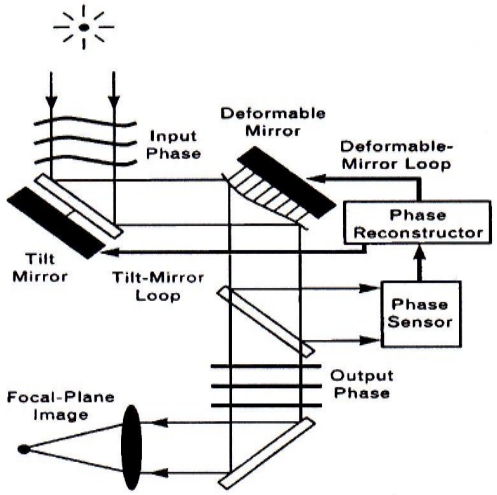


Figure 7[5]

Overview of an adaptive optics telescope design. The system removes aberrations introduced from the atmosphere by attempting to flatten incoming wavefronts

The phase (or wavefront) sensor is used to measure the aberrations in the optical field. The most common design is the Shack-Hartmann sensor which measures the optical phase gradient (slope of the wavefronts) of the compensated optical field. The Phase reconstructor turns these phase gradients into signals that drive the adaptive mirrors (tip-tilt and deformable mirrors). The adaptive mirrors apply optical path differences to the optical field as a function of position through the optical field. The optical path differences introduces phase differences in the optical field that are an attempt to cancel out the aberrations.

5.2) Guide stars

The phase measurements tend not to be made on the light from the science object under scrutiny, but on the light from a guide star. This is because the science object tends to be too dim (or spatially extended) to provide wavefront correction information. The AO system compensates for the turbulence sampled by the beam from the guide star. This compensation is applied to the science beam also. If the guide star and the science object are close together then they clearly sample the turbulence similarly. As their angular separation increases, they sample the turbulence differently (see figure 9). The isoplanatic angle, θ_o, is used to describe the maximum angular separation between the science object and guide star for effective

atmospheric compensation. In practice this corresponds to a gradual deterioration of science image quality as the angular separation increases. This effect is termed angular anisoplanatism.

The guide star must be bright for the phase sensor to be able to function (see section on wavefront sensor). This means that an AO system using a natural guide star (NGS) only functions on patches of sky near a suitably bright star, which while useful is restrictive (fractional sky coverage estimates varies, but a coverage of 10^{-6} is described in reference [5]). In order to overcome this limitation manmade guide stars can be produced. They are made from the scattering of a laser beam in the atmosphere (see figure 8). Two types of laser guide stars are typically used. Rayleigh laser guide stars have altitudes up to 20-25km high. Above this the atmosphere does not scatter the laser light sufficiently to form a bright enough guide star. The height of the guide star is controlled by emitting pulses of laser light and timing the guide star observations so that only the light scattered at a given altitude is observed. Sodium layer laser systems rely on the presence of a sodium rich layer at ~90km formed from meteorite collisions with the outer atmosphere. A suitably designed laser can use this high altitude layer to produce a laser guide star.

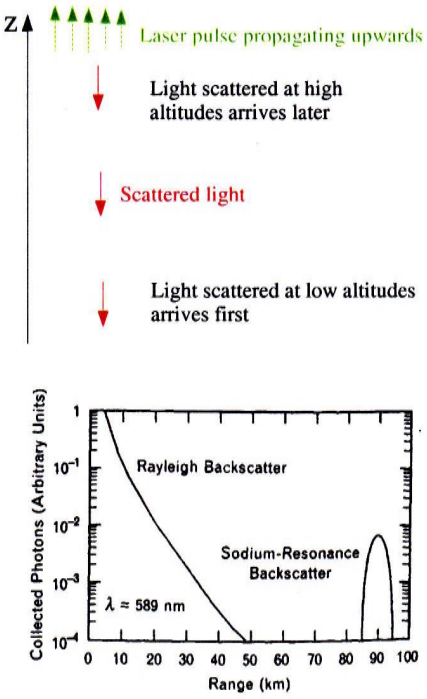


Figure 8

Top: Light scattered from a laser beam acts as a manmade guide star with altitude selection via photon arrival timing.
Bottom[5]: The scattering strength of 589 nm light as a function of altitude.

A laser guide star (LGS) is at a finite range. This means that the wavefronts coming from the LGS are spherical. This means that the atmospheric turbulence is sampled differently below the LGS and not at all above it. This is known as focal anisoplanatism and is shown in figure 9. As the majority of image aberrations are caused in the lower atmosphere, LGS's are still able to produce a compensated image.

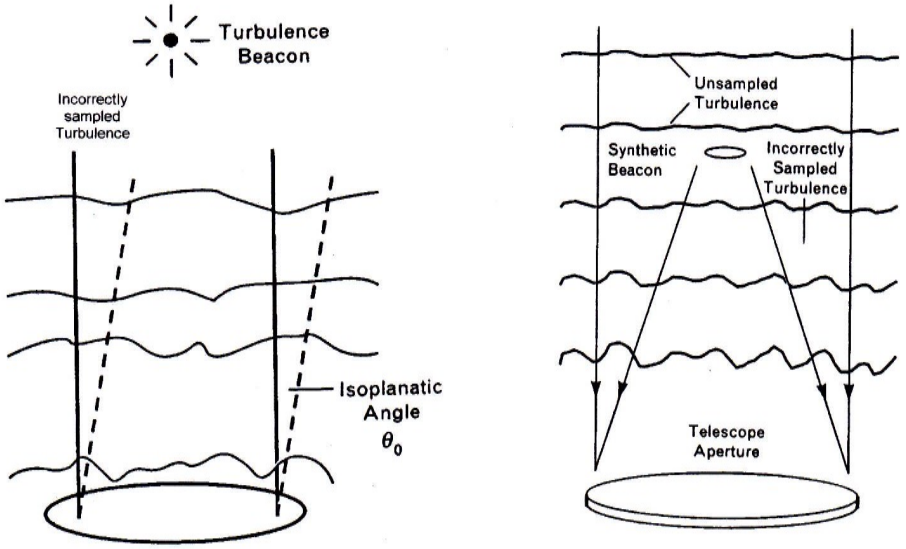


Figure 9[5]

Angular anisoplanatism (left) is applicable to both NGS's and LGS's. Focal anisoplanatism (right) is caused by the finite range of the LGS

5.3) Wavefront sensor

There are several existing designs of wavefront sensor. The most popular design is a Shack Hartmann design.[2][6] A Shack-Hartmann design is shown in figure 10[6].

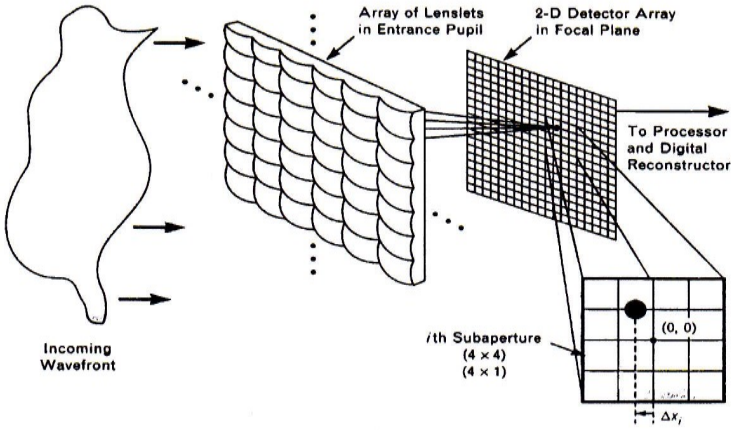


Figure 10[6]

A Shack Hartmann wavefront sensor made of a microlens array with a CCD in the focal plane

The basic principle of Shack Hartmann sensors is to break up the incident wavefront into small subapertures and focus these small sections of wavefronts onto a CCD. A microlens array is a 2 dimensional grid of small lenses that are joined together. The microlenses focus the wavefront into a series of spots. The position of each spot shows the wavefront slope over the aperture of the microlens concerned. Measurement of the position of each of these spots is a measurement of the gradient of the wavefront (see figure 11) with a resolution equal to that of the microlens separation.

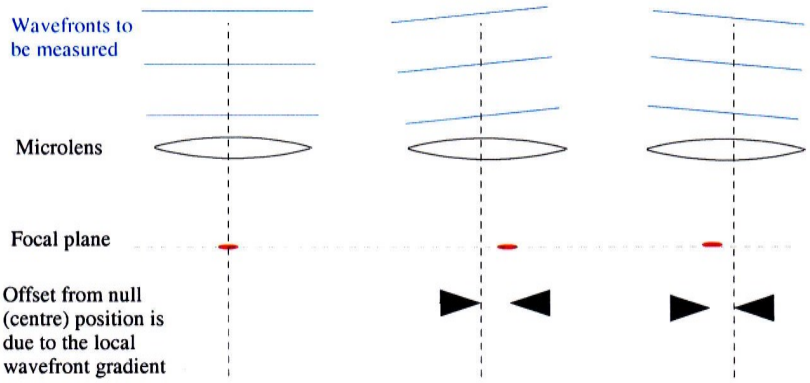


Figure 11

The local wavefront gradient incident on a micro lens produces a spot, whose position can be measured to deduce the wavefront gradient.

There are limits on the resolution of the CCD that is used because of the limited flux available. As the adaptive optic system must be able to respond in real time to the atmosphere, there is only a very short time available for a wavefront measurement to be made. It is therefore not possible to have a long integration time for the Shack-Hartmann CCD. Clearly, using a lower resolution CCD and larger microlenses increases the number of photons collected per measurement, increasing the signal to noise ratio obtained. The resolution chosen depends on the degree of compensation the system is designed to achieve and the intensity of the guide beam on which these measurements are made.

In order to provide any compensation, the adaptive optics system must be able to change the shape of the wavefronts. This is achieved by introducing optical path length differences across the beam. The optical path differences applied are local and adjust the shape of the wavefront as a function of position across the beam. The optical path length differences can be introduced by using a mirror surface that can be controllably deformed (see Figure 12).

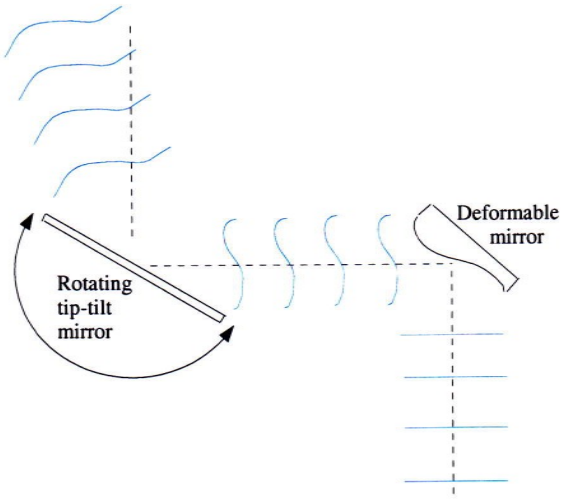


Figure 12
The tip-tilt mirror (often referred to as the fast steering mirror, FSM) keeps the centre of instantaneous images stationary while the deformable mirror sharpens up the instantaneous images by removing optical path differences that have been experienced by the beam as it passes through the atmosphere. The result of this is that the wavefronts are flattened and in the ideal case returned to a pure plane wave.

The deformable mirror (DM) has a surface that can be controllably deformed to change the shape of incident wavefronts. If supplied appropriate phase information a DM can flatten a wavefront to the resolution of its actuators. The tip-tilt mirror (often known as a fast steering mirror, FSM) applies a uniform phase gradient to the optical field, or tilts the wavefronts. The FSM can then be used to keep the centre of the image stationary. If the centre of the image is held still and the wavefronts flattened then the image formed will be 'sharper' and have a higher resolution.

The FSM is separate from the deformable mirror because of the stroke requirements of the tip-tilt modes. By removing the tip-tilt modes the complexity of the DM can be reduced. Designs used for DM's are shown in figure 13. They all use actuators (peizo electric crystals) to modify the mirror surface, but are constructed differently. DM's can suffer from hysteresis effects and some designs include position sensors such as strain gauges with the actuators.[1]

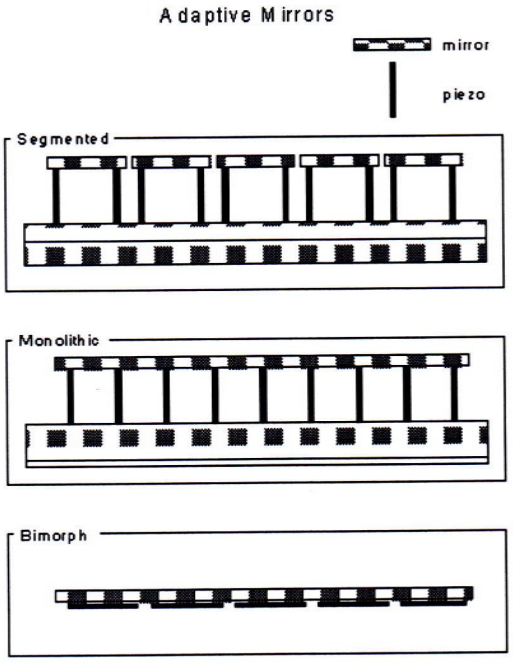


Figure 13
3 designs of deformable mirror. The segmented mirror has individual sections that can elevated or tipped by actuator stacks. The monolithic mirror has a surface that can flex according to the extension and contraction of the actuator stacks. The bimorph mirror controls its shape by flexing its surface.

5.5) AO closed loop feedback

An AO system works as a closed loop. It senses phase aberrations using wavefront sensors, calculates a 'correction' to remove the aberration and applies this to the optical beams. As the wavefront sensor is placed after the DM, it is the residual phase aberrations in the guide star beam that are measured. Thus the control loop (or phase reconstructor) acts as a feedback mechanism. This is closed loop operation; only the uncorrected components of the atmosphere are sensed, allowing the system to react to an evolving atmosphere and allow for a higher tolerance on the deformable mirror, making its production more practical. The feedback loop is generally calibrated by measuring the effect that the tip-tilt and deformable mirrors have on the wavefront (from the output of the wavefront sensor) under small actuator signals. The control matrix (see equation 8) can then be deduced for optimal atmospheric correction.

$$x = C \cdot s$$

Equation 8

The control signals, x, for atmospheric compensation using wavefront sensor output, s, produced by the control matrix, C. Note that x, s and C are matrices.

6) Previous modelling software

6.1) Models by Dr R. W. Wilson

Dr Wilson has developed two models for evaluating the performance of telescopes equipped with adaptive optic systems. The 'Modal covariance model' uses the statistics of Kolmogorov turbulence to construct point spread functions. It relies on an analytic solution to the correlation between Zernike modes for focal and angular anisoplanatism for Kolmogorov turbulence (see figure 14). Focal anisoplanatism involves the correlation of the Zernike modes on one scale to another scale. Angular anisoplanatism involves the correlation of the Zernike modes given a spatial displacement.

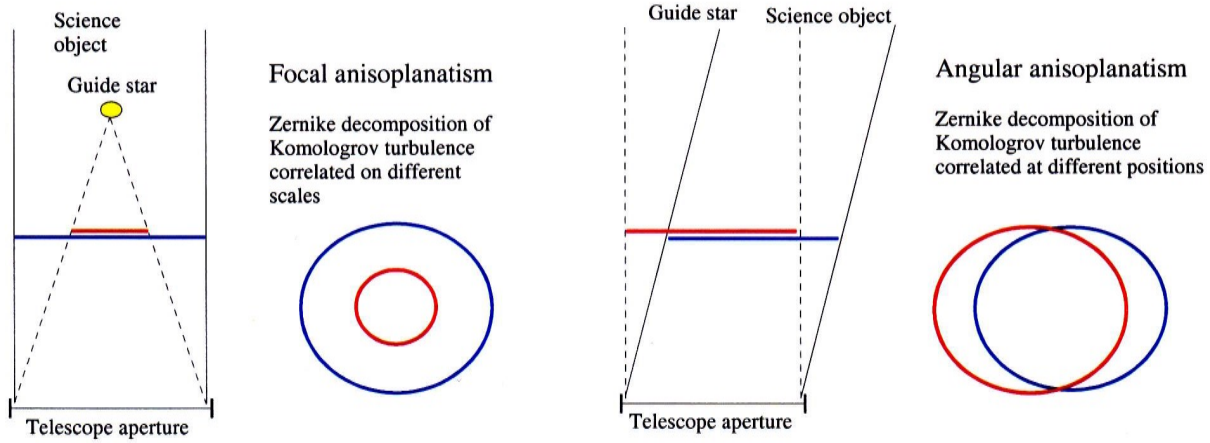


Figure 14

The red circles represent the turbulence sampled by the guide star beam and is the information available for the AO system to perform compensation. The blue circles represent the turbulence the science beam experiences. For Kolmogorov turbulence the correlation between the sampled and experienced turbulence can be analytically determined

The AO system model used in this model is an open loop system that removes the lowest orders of the sampled Zernike modes from the science beam. The effects of a time delay between sensing and applying the corrections are included in this model. The model constructs the PSF using the statistically described 'corrected' science beam. No atmospheric propagation is performed and intensity distributions are not considered. There is no simulation of noise in the system and so applies to high intensity situations. As this system is based on a statistical analysis, it does not operate as a time based simulation and does not simulate the dynamic mechanical evolution of the system. The computational requirements for the higher order Zernike modes needed in higher order AO systems is too expensive. This is the statistical model that is being used to validate the wave optics model.

The 'Monte-Carlo simulation' is a time based simulation of the phase differences in atmospheric phase screen as a beam passes down through the atmosphere. No propagation is involved, only the phase differences encountered. The atmospheric phase screens are given by Kolmogorov turbulence, but the atmosphere is made to be non periodic, with a turbulence layer made up of tiled random phase screens. This produces discontinuities in the atmosphere but allows the use of many individually created phase screens, avoiding any dependance on a few untypical phase screens. A closed loop AO system is used with tip-tilt

and wavefront sensors to measure the aberrations and a zonal DM to apply the time delayed 'correction' phase. Random noise is included in this model. The residual phase differences in the 'corrected' science beam are then used to produce both instantaneous and time integrated PSF. This model has been found to run at $\frac{1}{10}$ - $\frac{1}{20}$ real time on readily available PC's.

6.2) The Brent Ellerbroek approach

The approach for higher-order modelling of adaptive optics systems by Brent Ellerbroek[7] proposes that wave optics propagation of optical fields is required to go beyond many of the existing first order geometrical models in order to describe effects ranging from scintillation and diffraction to realistic system component response effects (such as misregistration to AO control loop dynamics). This approach uses discrete grids to represent optical fields, the atmosphere and telescope. The atmosphere is broken up into several 2D layers that apply turbulence to an optical field as it is propagated through the atmosphere. The atmosphere is modelled to a resolution of several cm using a grid that represents ~50m across. The deformable mirrors and wavefront sensors are simulated to provide an accurate description of scintillation, wavefront misregistration and imperfect deformable mirrors. LGS elongation and aberration effects can be described by wave optics propagation in order to simulate multi-conjugate adaptive optics systems. The reported results use 100 cycles (i.e. propagation sweeps from the science object to imaging element in telescope) to produce a time averaged Strehl. ratio for the system. This requires 1 - 6 days to perform on a up to date PC. The work undertaken by Brent Ellerbroek was incomplete at the time the author was last in personal contact.

7) Simulation theory

7.1) Optical field representation

A coherent optical field can be represented on a discrete square grid with a side of physical length **l** with **n** discrete points (see figure 15). The complex valued amplitude at each discrete point defines the field.

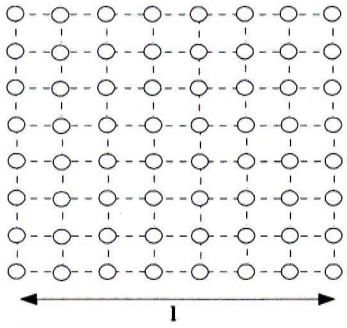


Figure 15
An **n=8** square grid of side **l**

The propagation technique has fixed plane wave decomposition basis set producing a limit on the greatest angle that energy will propagate with respect to the z axis (see section 4.1). Thus no optical field with a plane wave component inclined at an angle greater than this can be represented. The largest phase difference between 2 neighbouring points in the grid must be π radians as the phase is wrapped between 0 and 2 π by the complex scalar representation. Any larger phase difference will be mapped to a smaller one. However a limiting situation, as shown in figure 16, leads to 2 possible plane wave reconstructions. This is an effect of using the highest spatial frequency and does not uniquely represent the optical field. Using the second highest spatial frequency possible reduces the phase difference between two neighbouring grid points to $\pi/2$ radians. This removes the plane wave reconstruction and determines the limiting angle for energy propagation on a grid, $\theta_{grid\ max}$. (see equation 9 & figure 17). The most severe restriction occurs if the plane wave is propagating such that the wavefronts cross exactly at a grid diagonal as the grid point spacing is largest.

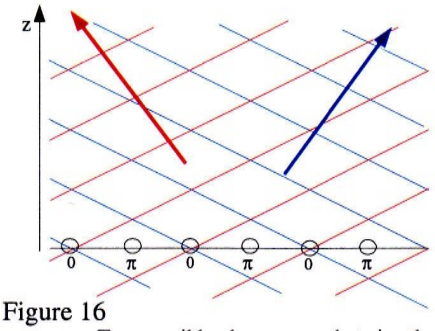


Figure 16
Two possible plane waves that give the same phase pattern on discrete grid points. The values are the phase at each grid point. The arrows are the direction of propagation.

As described above and shown in figure 16, the highest spatial frequency components on a grid will not be correctly propagated and will move energy in an incorrect fashion. This error is an easily measurable by finding the rms amplitudes of the extreme spatial frequencies. By assuming the errors are independent, they can be spread uniformly over the grid to obtain an estimate for the optical field representation error. This error need only be evaluated when measurements are made on the optical field as they are intrinsically carried within the optical field representation.

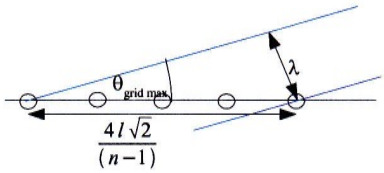


Figure 17
The wavefronts of an optical field crossing the grid to give the maximum angle for accurate propagation.

$$\vartheta_{grid\ max} = \arcsin\left(\frac{(n-1)\lambda}{4l\sqrt{2}}\right)$$

Equation 9
The maximum angle a plane wave can be to the z axis for accurate propagation.

The use of fourier transform representation for optical propagation results in a periodic representation of the optical

field. This means that energy can flow off one edge of the optical mesh and onto the other edge. In some situations this is acceptable, such as in atmospheric propagation where it has been assumed that the atmosphere is periodic in order to translate atmospheric phase grids to allow for off axis line of sights. In other situations such as in an isolated beam, this is unacceptable. In such situations energy would be able to move off one edge of the grid and onto the other in an unphysical way. Clearly, the isolated beam situation can be correctly represented if there is no energy near the edges of the grid.

To enforce isolated beam behaviour the edges of the grid can be masked out before applying a phase screen. This is shown in figure 18 where the masked width, **d**, is user defined. The appropriate size for this masking is case dependant. As the beam is isolated, the amplitude near the edges of the grid should be small. The purpose of the masking is to remove the energy that is propagating away to infinity. Applying the mask must be done as a phase screen is applied to the isolated beam. The error estimation is a two stage process. The amplitude that has managed to wrap around the edges must be estimated before applying the phase screen in order to estimate this propagation error.

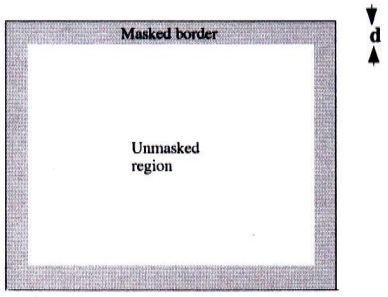


Figure 18
Masking out the edge of the grid prevents energy from wrapping around the edge of the grid as the optical field is propagated due to the periodic nature of the fourier transform techniques employed. The width of the masking, **d**, is case dependant.

The wrapping effects can be estimated by averaging the amplitude at the edges of the grid and asserting that the error produced by wrapping is less than or equivalent to this average edge amplitude at all points in the grid (see equation 10). The phase screen is then applied. The error due to masking is estimated by asserting that the error is the integrated amplitude within the masked border spread evenly over the unmasked region (see equation 11).

Equation 10
The wrapping error is assumed to be constant over the entire grid and is less than or equal to the average amplitude around the grid edge.

$$\text{Wrapping error} = \frac{\text{integrated intensity around grid edge}}{4n - 2}$$

Equation 11
The masking error is estimated by assuming that the effect that the unmasked region has on the masked border (per unit area) is the same as the effect the masked border has on the unmasked region. This is then assumed to be uniform across the grid.

$$\text{Masking error} = \frac{\text{integrated intensity within masked border}}{(n-2d)^2}$$

Strictly this applies a wrapping error twice (once before applying a phase screen and once afterwards). However the masking error is measuring the error in removing additional parts of the optical field while the wrapping error is measuring an unwanted propagation effect due to the data representation. The errors produced are assumed to be independent of any other error and evenly spread over the grid in order to produce an acceptably simple way of quantifying these effects. Note that these are the estimated errors due to discrete representation and are stored as fractional errors (to keep the errors in scale with the total intensity in the grid as masking is applied).

7.2) Spherical wavefronts

A spherical wavefront diverging from or converging to a point can be described in a plane given by equation 12. The only information required is if the wavefront is converging or diverging and the distance to the converging or diverging point (see figure 19). The diverging or converging point will be referred to as a focal point with the converging or diverging possibility fixed by context. The convention used here is that $|\mathbf{f}|$ is the shortest distance from the plane to the focal point with a sign convention for $\mathbf{f}$ shown in figure 19. The phase pattern is simply 2π times the no of waves that will fit between a given point and the focal point with some sign manipulation in order to follow the convention.

$$u = \underbrace{\frac{1}{\sqrt{f^2 + x^2 + y^2}}}_{1/r^2 \text{ intensity law}} \underbrace{e^{\frac{-ikf}{|f|}\sqrt{f^2 + x^2 + y^2}}}_{\text{Phase pattern}}$$

Equation 12

The complex amplitude for a spherical wave in a plane where $|\mathbf{f}|$ is the distance to the focal point, $\mathbf{x}$ & $\mathbf{y}$ are the Cartesian position in the plane using the origin as the closest point in the plane to the focal point. The sign convention for $\mathbf{f}$ is shown in figure 110.

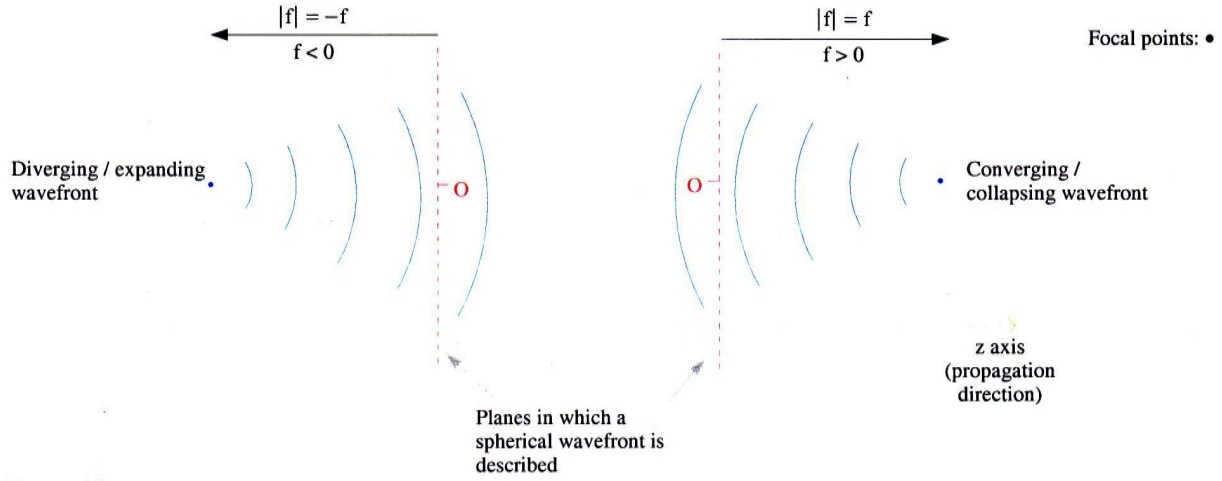


Figure 19

Wavefronts of a spherical wave expanding from and collapsing to a focal point. The planes have a complex amplitude given by equation 104. Note the sign convention for $\mathbf{f}$. $\mathbf{f > 0}$ if the spherical wave is collapsing & $\mathbf{f < 0}$ if the spherical wave is expanding as it propagates along the z axis. The origins of the planes are marked with a 'O', being the closest point in the planes to their respective focal point.

7.3) Beam speed restriction

As described earlier, there is a maximum angle, $\theta_{\text{grid max}}$, between the direction energy can propagate and the +ve z axis of the grid used in the optical field representation. This makes it difficult to focus the beam for fast lenses (small focal ratio) as the geometric light ray path near the edge of the lens must be bent to a large angle, θ, with respect to the grid +ve z axis. Clearly this cannot exceed the maximum dictated by the optical field representation otherwise the simulation is invalid. In the case of a plane wave propagating in the +ve z direction, the limiting beam speed (see equation 13) is given by the shortest focal length that permits a geometric ray originating from the corner of the grid to reach the focus (see figure 20).

This particular plane wave case ignores any variation in the optical field due to information carried in the beam. This pessimistic situation is offset however by the system design. An isolated beam should never be stretched right across the grid. Masking (see optical field representation section above) and optical component size (see system design section below) limits the effective diameters to less than the situation considered here, allowing for the variations in the beam itself. Any situation where

this does not apply will cause large amplitudes in the highest spatial frequencies of the optical field. This possibility is then automatically included in the optical field representation error estimation.

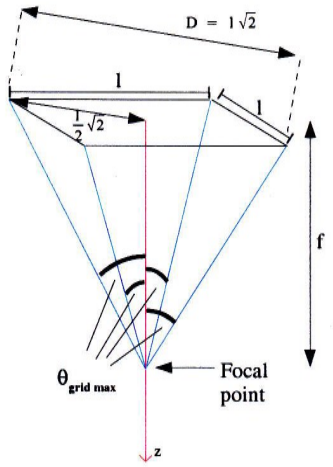


Figure 20
The limiting beam speed for a plane wave propagating along the z-axis is given by the requirement that energy in the corners of the grid is correctly propagated towards the focal point.

$$\frac{|f|}{D} \geq \frac{1}{2\tan(\vartheta_{grid\,max})}$$

For $l = 0.1$m, $\lambda = 500$ nm, $n = 1000$; $\theta_{grid\,max} \sim 3.5 \times 10^{-3}$ radians

This is very small, allowing the use of small angle approximations to give equation 13

$$\frac{|f|}{D} \geq \frac{2l\sqrt{2}}{(n-1)\lambda}$$

Equation 13
The minimum beam speed that can be represented using a grid of side l with n discrete points along a side for wavelength λ. Note that this applies to both diverging and converging beams

7.4) Focussing and lenses

To focus a beam of light using a lens or mirror, optical path differences are introduced across the beam cross-section such that the energy propagates towards the foci. To produce a lens in the simulation, its effects must be described within a plane. The definition of a lens that I am using is a plane that applies a phase change equivalent to the phase of a spherical wavefront collapsing to (for a converging lens) or expanding from (for a diverging lens) the focal point (see figure 21 and section on spherical wavefronts above). A converging lens will cause an incoming normal plane wave to be focussed at the focal point and conversely produce an outgoing normal plane wave from an incoming wavefront produced by a point source at the focal point (see figure 21)

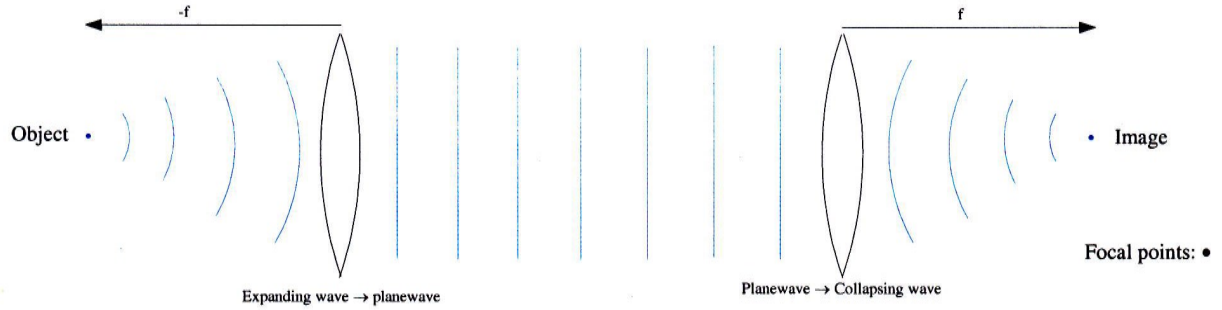


Figure 21
Wavefronts from a point source focussed into an image. Approximating these lenses to planes only requires producing a multiplicative phase screen equivalent to the phase of a collapsing wave a distance f from its focal point. (see equation 104 above)

Imaging can then be achieved by putting a converging lens phase screen in a beam, forming an image in the conjugate plane to the object. The beam speed restriction limits the strength of lens that can be simulated. As the primary mirror in a telescope has a speed in the order of 10's, while the representable speed is of the order of 1000's, it is not a possible to simulate these lenses as phase screens. As can be seen from equation 12, the speed of lens that can be simulated is proportional to 1/n.

However the processing time of fast fourier transforms is proportional to $n^2\ln(n)$. Boosting the resolution of the simulation to the point where these fast beams can be included would be computationally impractical (see simulation timings later).

7.5) Co-ordinate systems

As discussed previously, representations of fast lenses using planes in a physical Cartesian co-ordinate system is impractical because of the high resolution required. This suggests that altering the propagation by changing the co-ordinate system may be able to reduce the required wavefront (i.e. phase) gradient that must be be represented. If possible this would allow a much lower resolution to simulate fast beams. There is a restriction that this places on the optical fields themselves; a beam must be dominated by a single point object. All of the co-ordinate systems that the Author has considered is intimately connected to the position of a point object. To consider several point objects would require an equal number of optical beams. Each beam would deal with one point object alone, but the physical optical field is simply the superposition of all the beams.

The Author stresses that this search was unsuccessful. The purpose of this section is to demonstrate the limitations of the numerical representation and justify the existance of the techniques described in section 7.6. This property of numerical representation prevents a direct simulation of the physical optical system. Several approaches the Author attempted to use in order to find a co-ordinate system are described below including the reason for them being inappropriate.

7.5.1) Lens speed rescaling

As a fast beam cannot be represented, attempt to replace it with a slower beam and propagate the beam for longer so that it reaches focus at a equivalent point. If this approach is valid then there should be no difference in the analytic expressions for the optical fields produced by each approach. Changing the speed of the beam produced is equivalent to using different strength lenses

	Lens focal length **f** (control situation)	Lens focal length **f'** ($\neq$**f**)
Test input	$\mathbf{u_0 = a\, e^{2\pi i\,(x\alpha+y\beta)}\, \exp(ik\,(f^2+x^2+y^2)^{1/2})}$	$\mathbf{u_0 = a\, e^{2\pi i\,(x\alpha+y\beta)}\, \exp(ik\,(f^2+x^2+y^2)^{1/2})}$
Apply lens	$\mathbf{u_1 = \exp(-ik\,(f^2+x^2+y^2)^{1/2})\, u_0}$ $\mathbf{= a\, e^{2\pi i\,(x\alpha+y\beta)}}$	$\mathbf{u_1' = \exp(-ik\,(f'^2+x^2+y^2)^{1/2})\, u_0}$ $\mathbf{= a\, e^{2\pi i\,(x\alpha+y\beta)}\, \exp(ik\,[(f^2+x^2+y^2)^{1/2}-(f'^2+x^2+y^2)^{1/2}])}$
	$\mathbf{U_1 = \Im[u_1]}$ $= \mathbf{a}$ for $(f_x,f_y) = (\alpha,\beta)$ $\mathbf{0}$ otherwise	$\mathbf{U_1' = \Im[u_1']}$

Note that by using a lens of focal length **f'** rather than one of **f** (the control), what was a plane wave of a single frequency is now a complex collection of spatial frequencies. Propagation only changes the relative phase of each of those

frequency components but not their amplitude. They will therefore be present in whatever plane you consider down the propagation path. There is clearly no way these two representations will be equal after propagating a distance **z = f** and **f'** respectively.

The only freedom we have available in order to make them equivalent would be to change the co-ordinate system and rescale the optical fields (by changing the grid size, **l**) appropriately. As this would have to apply to all values of (**α,β**), we can evaluate the possibility of this approach using the (**α,β**) = (0,0) situation. The lens of focal length **f** produces a uniform amplitude field. The lens of focal length **f'** produces an amplitude field containing oscillations. As there is no degeneracy in a fourier decomposition, there is no way to remove these oscillations using a grid size rescaling.

Lens speed rescaling clearly does not work in this case and so cannot work in general. This approach is therefore rejected. The problem with this approach can be interpreted as being due to the different amounts of evolution the beam undergoes in the different propagation lengths. The number of waves that fit along the propagation length changed and hence the propagation evolution must be different.

7.5.2) Converging co-ordinate system

Rather than altering the physical conditions (e.g. the speed of a lens), we can try to deduce a co-ordinate system (see figure 22) so that some, or all, of the spherical wavefront term is evaluated analytically. To do this it is necessary to be able to split (or hence combine) spherical wavefront terms into two terms.

Define $\tilde{u}$ to be the optical field equivalent to u but with some spherical wavefront component removed and **s** as the signal contained within the beam.

$$u \equiv s\, e^{-ik\sqrt{f^2+x^2+y^2}} \simeq s\, e^{-ikf}\, e^{\frac{-ik}{2}\left(\frac{x^2+y^2}{f}\right)} \qquad \text{valid for } \frac{x^2+y^2}{f^2} \ll 1$$

Assert that the spherical term can be split into two spherical terms with focal lengths f_1 and f_2, allowing for a phase factor.

$$u = \tilde{u} e^{-ik\sqrt{f_1^2+x^2+y^2}} \simeq \tilde{u} e^{-ikf_1}\, e^{\frac{-ik}{2}\left(\frac{x^2+y^2}{f_1}\right)} \qquad \text{valid for } \frac{x^2+y^2}{f_1^2} \ll 1$$

where

$$\tilde{u} = s\, e^{-ik\phi}\, e^{-ik\sqrt{f_2^2+x^2+y^2}} \simeq s\, e^{-ik\phi}\, e^{-ikf_2}\, e^{\frac{-ik}{2}\left(\frac{x^2+y^2}{f_2}\right)}$$

This requires that

$$e^{-ik\left(f+\frac{x^2+y^2}{2f}\right)} = e^{-ik\phi}\, e^{-ik\left(f_1+\frac{x^2+y^2}{2f_1}\right)}\, e^{-ik\left(f_2+\frac{x^2+y^2}{2f_2}\right)} \quad \text{i.e.} \quad \phi = f - f_1 - f_2 + \frac{(x^2+y^2)}{2}\left(\frac{1}{f} - \frac{1}{f_1} - \frac{1}{f_2}\right)$$

For spherical waves to be split into two spherical waves, ϕ must be independent of x & y (as it is allowed to be a constant only) which requires $\frac{1}{f} = \frac{1}{f_1} - \frac{1}{f_2}$, the thin lens law. Thus the spherical wavefront phase pattern produced by a lens

can be separated into two spherical wavefront patterns so long as $\frac{x^2+y^2}{f^2}$, $\frac{x^2+y^2}{f_1^2}$, $\frac{x^2+y^2}{f_2^2} \ll 1$ or $\frac{f}{D} \gg 0.5$. This is clearly not ideal, but includes the lenses (or mirrors) of the order of those found in telescopes.

In order to deduce a new co-ordinate system we can consider the relationship between 2 planes in the beam. As both planes describe the same stable beam, they must both produce the same image at focus. It is therefore valid to propagate both planes to focus and set them equal. This will produce the converging co-ordinate system (equation 14) described by B.L. Ellerbroek[7].

Propagation of optical field **u** from a plane to focus, a distance f away, gives the focus optical field $\mathbf{u_f}$.

$$\Im[u_f] = H(f)\Im[u] \quad \text{where H(f) is the propagator to focus.}$$

or equivalently $u_f = h(f)*u$ where $h(z) = \Im^{-1}[H(z)] = \frac{e^{ikz}}{i\lambda z} e^{\frac{ik}{2z}(x^2+y^2)}$

(* means convolution over x,y co-ordinates)

The propagation kernel, **h(z)**, ignores evanescent waves considered previously in the optical propagator, **H(z)**. This is acceptable as the resolution of the grids used to represent these optical fields will never be fine enough to resolve such evanescent waves and they need not be considered.

$$u_f = \iint u \frac{e^{ikf}}{i\lambda f} e^{\frac{ik}{2f}((x-x_i)^2+(y-y_i)^2)} dx_i dy_i$$

$$u_f = \iint s\, e^{-ikf} e^{\frac{-ik}{2f}(x_i^2+y_i^2)} \frac{e^{ikf}}{i\lambda f} e^{\frac{ik}{2f}(x^2+y^2)} e^{\frac{-ik}{f}(x x_i + y y_i)} e^{\frac{ik}{2f}(x_i^2+y_i^2)} dx_i dy_i$$

The total spherical term in the optical field, **u**, cancels out a similar term in the propagation kernel, turning the expression into a fourier transform.

$$u_f = \frac{1}{i\lambda f} e^{\frac{ik}{2f}(x^2+y^2)} \iint s\, e^{\frac{-ik}{f}(x x_i + y y_i)} dx_i dy_i$$

$$u_f = \frac{e^{\frac{ik}{2f}(x^2+y^2)}}{i\lambda f} \Im[s]_{y_i \to \frac{y}{\lambda f}}^{x_i \to \frac{x}{\lambda f}}$$

Note that the entire spherical term in the optical is required to turn the integral into a fourier transform. The spherical component of the optical field must therefore be kept separate, i.e. $\mathbf{u \equiv u(s,f)}$. Any spherical term in s would need to be removed and included within f by 'adding' spherical terms $\mathbf{f_1}$ & $\mathbf{f_2}$ using the thin lens law as discussed above. To create the co-ordinate system propagate two optical fields, $\mathbf{s_1}$ & $\mathbf{s_2}$ to focus (a distance α & β respectively) and set them equal.

$$s_1 * h(\alpha) = u_f = s_2 * h(\beta)$$

$$\frac{e^{\frac{ik}{2\alpha}(x^2+y^2)}}{i\lambda\alpha}\Im[s_1]^{x\to\frac{x}{\lambda\alpha}}_{y\to\frac{y}{\lambda\alpha}} = \frac{e^{\frac{ik}{2\beta}(x^2+y^2)}}{i\lambda\beta}\Im[s_2]^{x\to\frac{x}{\lambda\beta}}_{y\to\frac{y}{\lambda\beta}}$$

$$\Im[s_2]^{x\to\frac{x}{\lambda\beta}}_{y\to\frac{y}{\lambda\beta}} = \frac{\beta}{\alpha}e^{\frac{ik}{2}(x^2+y^2)(\frac{1}{\alpha}-\frac{1}{\beta})}\Im[s_1]^{x\to\frac{x}{\lambda\alpha}}_{y\to\frac{y}{\lambda\alpha}}$$

$$s_2 = \frac{\beta}{\alpha}\Im^{-1}[\,e^{\frac{ik}{2}(x^2+y^2)(\frac{1}{\alpha}-\frac{1}{\beta})}\Im[s_1]^{x\to\frac{x}{\lambda\alpha}}_{y\to\frac{y}{\lambda\alpha}}\,]^{\frac{x}{\lambda\beta}\to x}_{\frac{y}{\lambda\beta}\to y}$$

Identifying spatial frequencies $f_x \equiv \frac{x}{\lambda\alpha}$, $f_y \equiv \frac{y}{\lambda\alpha}$ & $f^2 = f_x^2+f_x^2$.

$$s_2 = \frac{\beta}{\alpha}\Im^{-1}[e^{i\pi f^2\lambda\frac{\alpha}{\beta}(\beta-\alpha)}\Im[s_1]^{x\to f_x}_{y\to f_y}\,]^{\frac{f_x\alpha}{\beta}\to x}_{\frac{f_y\alpha}{\beta}\to y}$$

Using the identity $\Im^{-1}[G(af_x, bf_y)] = \frac{1}{|ab|}g(\frac{x}{a},\frac{y}{b})$ where $G = \Im[g]$

$$s_2(\frac{\beta}{\alpha}x,\frac{\beta}{\alpha}y) = \frac{\alpha}{\beta}\Im^{-1}[e^{-i\pi f^2\lambda\frac{\alpha}{\beta}(\alpha-\beta)}\Im[s_1]\,] \quad \text{(equation 14)}$$

propagator with $z = \frac{\alpha}{\beta}(\alpha-\beta)$

(1st order expansion of square root in H(z), valid for $\lambda f \ll 1$)

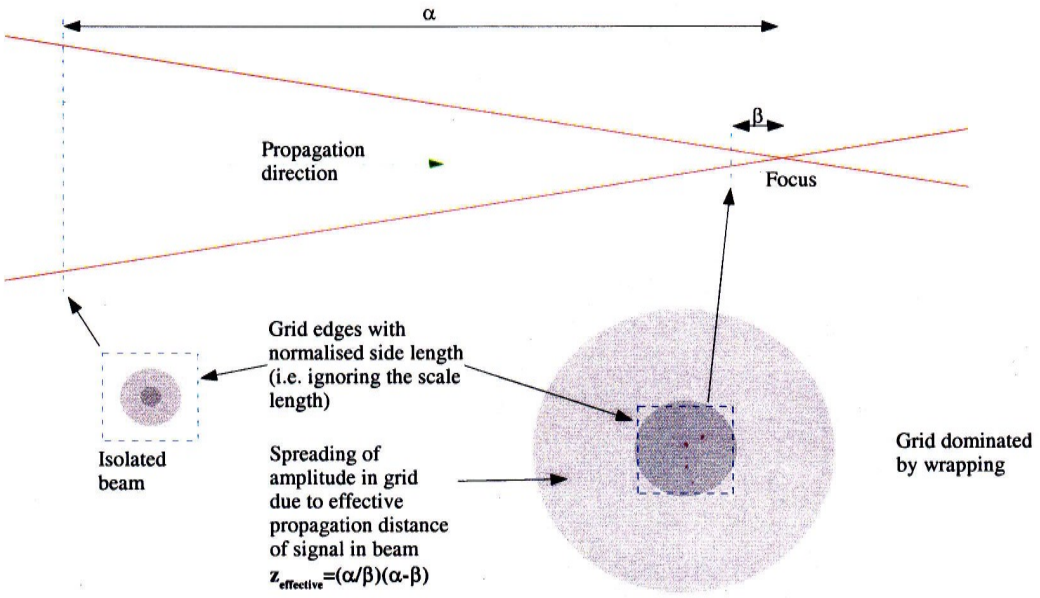


Figure 22

The converging co-ordinate system (top) showing the scale length in red as a beam propagates through focus. The grey regions represent the relative optical field amplitude (darker → larger) in the optical representation. As the optical field is propagated in the grid, it tends to spread out. As the plane of interest moves closer towards focus, the effective propagation length for the grid becomes disproportionately larger until it is inevitable that amplitude will wrap around the grid edges to a high degree.

This approach is successful in that it has not artificially changed the evolution in the beam, only its required representation. The only 'alteration' to the optical field has been the removal of the spherical terms. The propagator has not been modified to produce this expression and has been reproduced (for the case of $\lambda f \ll 1$, which is valid for all practical grids) in a form suitable for a changing scale length. The only limitation is that this analysis is restricted to $\frac{f}{D} \gg 0.5$ cases. This is

produced by the spherical wavefront approximation $u \equiv s\, e^{-ik\sqrt{f^2+x^2+y^2}} \simeq s\, e^{-ikf}\, e^{\frac{-ik}{2f}(x^2+y^2)}$. It is only possible to combine spherical terms because of this approximation, leading to the thin lens law. The analysis cannot be generalised to real lenses because of the non linearity of the square root. There is therefore no analogous 'thin lens law' for real lenses as demonstrated below.

$$u \equiv s\, e^{-ik\sqrt{f^2+x^2+y^2}} = s\, e^{-ik\phi}\, e^{-ik\sqrt{f_1^2+x^2+y^2}}\, e^{-ik\sqrt{f_2^2+x^2+y^2}}$$

$$\phi = \sqrt{f^2+x^2+y^2}-\sqrt{f_1^2+x^2+y^2}-\sqrt{f_2^2+x^2+y^2}$$

Expanding the square root as a power series and requiring ϕ from x & y determines the relationship between $\mathbf{f_1}$, $\mathbf{f_2}$ & $\mathbf{f}$.

$$\phi = (f-f_1-f_2)+\frac{(x^2+y^2)}{2}\left(\frac{1}{f}-\frac{1}{f_1}-\frac{1}{f_2}\right)-\frac{(x^2+y^2)^2}{4}\left(\frac{1}{f^3}-\frac{1}{f_1^3}-\frac{1}{f_2^3}\right)+\ldots$$

ϕ cannot be made independent of x & y and therefore the converging co-ordinate system does not apply beyond the $\frac{f}{D} \gg 0.5$ regime.

The converging co-ordinate system (equation 14) appears to be the solution to finite grid resolution, but it has hidden problems. It tacitly uses non-periodic fourier transforms. As the plane in a beam moves towards focus, the grid scale length is reduced (by a factor of $\frac{\beta}{\alpha}$), but the spread of the signal is increased during its effective propagation of $z_{effective} = \frac{\alpha}{\beta}(\alpha-\beta)$. If this spread is too much then wrapping effects become greatly amplified (see figure 22). This is a particular problem in propagating to a plane close to the focus as $\mathbf{z_{effective}}$ becomes very large, potentially causing so much wrapping that all signal is lost. The isolated beam condition is lost under these conditions due to the periodicity of the fourier transforms producing these artefacts. It is in fact impossible to reach focus ($\beta \rightarrow 0$ towards focus) as $z_{effective} \rightarrow \infty$, which is not possible to evaluate. This does not make the approach invalid, only its application to spatial numerical representation. The image made by focussing is the far field intensity pattern. This analysis has been done in terms of position (rather than angle), leading to an infinite spread (in grid position) of the signal being multiplied by a scale length of zero.

Another problem is in removing all spherical terms from within the signal, **s**, so that they do not modify the signals propagation over the distance $\mathbf{z_{effective}}$. As this distance can be quite long, all spherical terms must be removed to a high degree of accuracy. However there are an infinite continuum of spherical terms that might be present in the signal. All the spherical terms must be exactly removed for the analysis to be valid. Identifying all these modes and removing them is a process which the author assessed as being unlikely to be practical.

These reasons make the converging co-ordinate system unsuitable for general use in an optical system as the optical field simply cannot be determined in some sections of the beam.

7.5.2) Gaussian co-ordinate system

The problem with the converging co-ordinate system is produced by the requirement that all the spherical term be removed from the optical field, **u**, to leave only the signal, **s**. To reach focus the signal, **s**, must be propagated to infinity and there are serious wrapping issues. To prevent these effects some spherical term would need to be left in the signal. A Gaussian beam has analogous properties, inspiring the Author to attempt the approach below.

A Gaussian laser beam has a self reproducing Gaussian beam profile of intensity across the beam. This profile has a half width appropriately modified by focussing and suggests that following the evolution of a Gaussian beam could provide an appropriate for a co-ordinate system for focussing. As a coherent Gaussian beam propagates it increases its half width until in the distant field case it is linearly diverging (see figure 23). This behaviour is applicable to focussing of the beam by considering the beam half width being reduced down to its minimum waist in the focal plane (see figure 23). The beam half width as a function of propagation length may then provide a co-ordinate system.

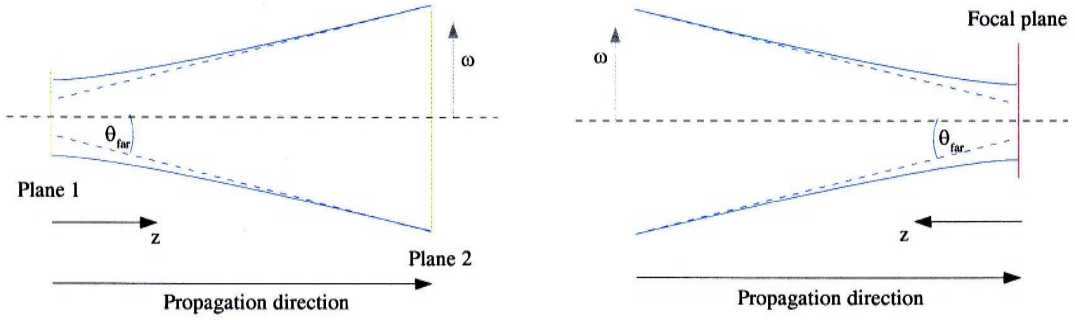


Figure 23

Left shows the width (ω) of a Gaussian beam as it propagates from plane 1 to plane 2. For a large distance away the beam diverges at an angle of θ_{far}. Reversing the direction of propagation (shown on the right) is focussing a Gaussian beam into the focal plane. $\theta_{far} = \sin^{-1}(\omega/f)$ in the far field case.

To demonstrate this define a Gaussian intensity profile as $e^{\frac{-(x^2+y^2)}{\omega^2}}$, giving an amplitude profile of $e^{\frac{-(x^2+y^2)}{2\omega^2}}$

Use the Fresnel propagation kernel, **h(z)**, to propagate a Gaussian optical field from plane 1 to plane 2.

Optical field in plane 1, $u_1 = A_1 e^{\frac{-(x_1^2+y_1^2)}{2\omega_1^2}}$

$$u_2 = u_1 * h(z)$$

$$u_2 = \frac{A_1 e^{ikz}}{i\lambda z} \iint e^{\frac{-(x_1^2+y_1^2)}{2\omega_1^2}} e^{\frac{ik}{2z}((x_2-x_1)^2+(y_2-y_1)^2)} dx_1 dy_1$$

$$u_2 = \frac{A_1 e^{ikz}}{i\lambda z} e^{\frac{ik}{2z}(x_2^2+y_2^2)} \iint e^{-(x_1^2+y_1^2)(\frac{1}{2\omega_1^2}-\frac{ik}{z})(x_1 x_2+y_1 y_2)} dx_1 dy_1$$

$$u_2 = \frac{A_1 e^{ikz}\, \pi\, a^{-1}}{i\lambda z} e^{-(x_2^2+y_2^2)(\frac{k^2 a^{-1}}{4z^2}-\frac{ik}{2z})} \quad \text{where } a = (\frac{1}{2\omega_1^2}-\frac{ik}{2z})$$

$$u_2 = A_1 e^{ikz} \frac{k\omega_1^2(k\omega_1^2 - iz)}{k^2\omega_1^4 + z^2} e^{\frac{-(x_2^2+y_2^2)k^2\omega_1^2}{2(z^2+k^2\omega_1^4)}} e^{\frac{i(x_2^2+y_2^2)kz}{2(z^2+k^2\omega_1^4)}} \quad \text{(equation 15)}$$

Amplitude | Gaussian | Phase pattern

Use the Gaussian term to determine the half width in plane 2

$$\omega_2^2 = \frac{z^2 + k^2\omega_1^4}{k^2\omega_1^2} = \omega_1^2 + \frac{z^2}{k^2\omega_1^2} \quad \text{(equation 16) The half width evolution of a Gaussian optical field.}$$

Note that this is non-linear in beam half width ω $\omega_1' = a\omega_1 \rightarrow \omega_2'^2 = a^2\omega_1^2 + \frac{z^2}{k^2a^2\omega_1^2} \neq a^2\omega_2^2$ or distance **z**.

It is the self reproducing profile of Gaussian fields (see equation 16) that suggested to the Author that they may be useful in producing a co-ordinate system that describes focussing. The co-ordinate system would need to provide a scale length for the grid representing the optical field as a function of position. Any system constructed would naturally need to be applicable to an input Gaussian field as a special case of an arbitrary input. We can use this to construct a co-ordinate system for focussing a Gaussian and then test to see if it can focus an arbitrary input grid.

The Gaussian half width far from focus can be described by a geometrical model as the Gaussian half width can be approximated to a function that is linear in **z** (see equation 17). This provides a convenient way of describing the half width of the Gaussian at all points (see figure 24 and equation 19) by using terms analogous to beam speed and the distance to focus.

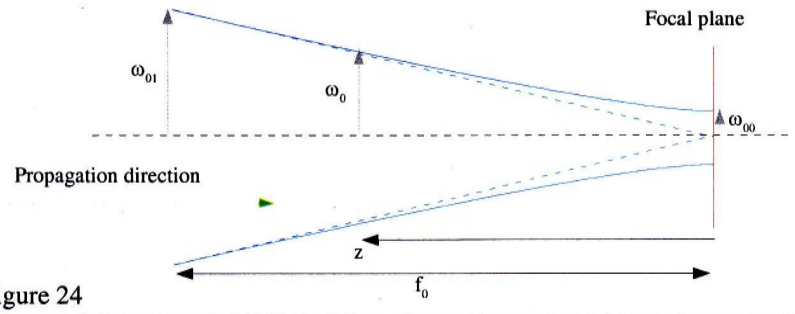


Figure 24

A Gaussian optical field of width ω_{01} focussed into a plane a distance f_0 away, producing a Gaussian of width ω_{00} in the focal plane. The Gaussian is shown to trace out its width as a function of distance from the plane, z, by the solid blue line. The dashed line is the linear far from focus limit which is the geometrical ray equivalent.

For $\frac{z}{k} \gg \omega_{00}^2$ $\omega_0 = \frac{z}{k\omega_{00}}$ (equation 17) Far from focus regime.

By inspection of figure 24, for $\mathbf{f_0}$ in the far from focus regime $\omega_0 = \omega_{01}\frac{z}{f_0}$ i.e. $\omega_{00} = \frac{f_0}{k\omega_{01}}$ (equation 18)

Using this expression for ω_{00} gives an expression for both near and far field $\omega_0 = \frac{f_0^2}{k^2\omega_{01}^2} + \frac{z^2\omega_{01}^2}{f_0^2}$ (equation 19)

The above situation is the real space description of the field and does not involve any co-ordinate system. This is the situation the Author wishes to represent using a co-ordinate system to make the optical representation practical. To introduce a new co-ordinate system, we assert that ω_{01} is in a plane immediately after a lens which has formed the focal plane a distance $\mathbf{f_0}$ away. We use the thin lens formula (see section 7.5.2) to split the focus from $\mathbf{f_0}$ (true focal length) to a mix of $\mathbf{f_1}$ analytically

added via co-ordinate system and $\mathbf{f_2}$ intrinsically present in the optical field representation (see equation 21,22). The Gaussian co-ordinate system the Author is considering provides scale length rescaling that follows the Gaussian half width profile. Any construction that is consistent with the physical situation (i.e. produces the same ω_0 at the same position for a Gaussian) can provide this co-ordinate system.

Convention

ω_{x0} : half width in focal plane
ω_{x1} : half width immediately after lens
ω_x : half width at arbitrary position

x=0 : physical (actual) situation
x=1 : co-ordinate system scale factor
x=2 : optical field representation within co-ordinate system

The co-ordinate system rescales a propagated Gaussian representation, ω_2, to the physical situation, ω_0, via scaling ω_1. (ω_{11} is a normalisation factor)

$$\omega_0 = \frac{\omega_1}{\omega_{11}}\omega_2$$

This must obviously apply both immediately after the lens and in the focal plane, i.e.

$$\omega_{01} = \omega_{21} \quad \text{and} \quad \omega_{00} = \frac{\omega_{10}}{\omega_{11}}\omega_{20}$$

Using the above and equation 18.

$$\omega_{00} = \frac{f_1 f_2}{k^2 \omega_{11}^2 \omega_{21}}$$

$$\frac{f_0}{k\omega_{01}} = \frac{f_1 f_2}{k^2 \omega_{11}^2 \omega_{01}}$$

This requires that

$$f_0 = \frac{f_1 f_2}{k\omega_{11}^2} \qquad \text{(equation 20)}$$

Another restriction has been placed on the possible values of $\mathbf{f_0}$, $\mathbf{f_1}$ and $\mathbf{f_2}$. ω_{11} is determined by the scaling factor immediately before the lens and cannot be changed to be convenient. Simultaneously solving this with the thin lens formula determines the sole possible values of $\mathbf{f_1}$ and $\mathbf{f_2}$. Two possibilities are produced by this (i.e. swapping the values of $\mathbf{f_1}$ and $\mathbf{f_2}$), but numerically representing a long distance focus term is the least demanding.

$$f_1 = \frac{k\omega_{11}^2}{2}\left(1-\sqrt{1-\frac{4f_0}{k\omega_{11}^2}}\right) \sim \frac{f_0}{\omega_{11}^2} \quad \text{(equation 21) Co-ordinate system focus term}$$

$$f_2 = \frac{k\omega_{11}^2}{2}\left(1+\sqrt{1-\frac{4f_0}{k\omega_{11}^2}}\right) \sim k\omega_{11}^2 \quad \text{(equation 22) Optical representation focus term}$$

This places a negligible requirement on the optical representation for such a long distance focus term, $\mathbf{f_2}$, (as $k \sim 10^7$). This scheme however cannot be applied to an arbitrary optical field. Gaussian optical fields have been described by their widths and distance to their minimal extent at a reference point (see equation 19) that is in the far from focus regime. The distance to the focal plane, $\mathbf{f_0}$, is highly dependant on the initial Gaussian used (see equation 20). This approach is then only applicable to a single Gaussian shape at a time. To apply it to an input requires that a hypothetical Gaussian decomposition must contain Gaussians of a single half width only so that the same co-ordinate system is applicable to each component. This is clearly a

restriction that prevents the generalisation of this technique to an arbitrary input. An arbitrary input would contain Gaussian components of different widths, each of which would need to follow a different co-ordinate system for physically correct propagation. Only one co-ordinate system can be applied to an arbitrary field, which would result in unphysical distortions as most of the Gaussian components would be propagated incorrectly. This is unacceptable and caused the Author to reject this final approach to producing a co-ordinate system.

This co-ordinate system was constructed in an attempt counter the problems identified in the converging co-ordinate system. An optical field representation with some focussing was permitted ($\mathbf{f_2}$) in an attempt to prevent wrapping effects dominating, and the finite size of a focussed Gaussian eliminates the zero scale factor and infinite propagation distance to focus. Since this approach has failed, the Author is unable to identify any further possible improvements to producing a co-ordinate system. Furthermore the Author suggests that a discrete grid representation is fundamentally limited to represent an optical system that requires using lenses to describe it. This limits the situations that can be represented to those described in section 7.6

7.6) Representable situations and image formation

All the Authors attempts to produce an optical representation with a rescaling co-ordinate system that can be applied to a realistic optical system have been described above. They have been all rejected due to undesirable (and some unphysical) properties they introduce. The optical field representation therefore restricts the possible optical systems that can be modelled, to ones that involve collimated or near collimated beams such that a Cartesian co-ordinate system is used. The limitations described in section 7.2 must be accepted as a consequence of a spatial grid representation for an optical field.

Despite the restrictions it is possible to simulate real systems approximately. Faithful reproduction of some optical apparatus may be impossible, but by using a functional description rather than a physical description of an optical system, equivalent approaches can be found. Conjugate (object/image planes) can be accessed by propagating backwards (i.e. moving back up beam) or forwards in a way that cannot physically occur. The grid always follows a Cartesian co-ordinate system, but its scale factor can be changed to emulate beam expanders. Image formation cannot be achieved through propagation as sufficiently strong lenses would exceed the resolution of the optical representation. Two imaging techniques are required for use in the simulation. Far field imaging, which uses a fourier transform to determine the angular image for an object at infinity, and the lens impulse response technique for objects at a finite distance[2] (see equation 23, 24). Normally for a telescope system in practice there is no pupil to the imaging components, i.e. $P(x,y) = 1.0$ as the collecting pupil has a smaller projected size on the imaging component. Figure 25 shows physical systems and approximated versions suitable for simulation.

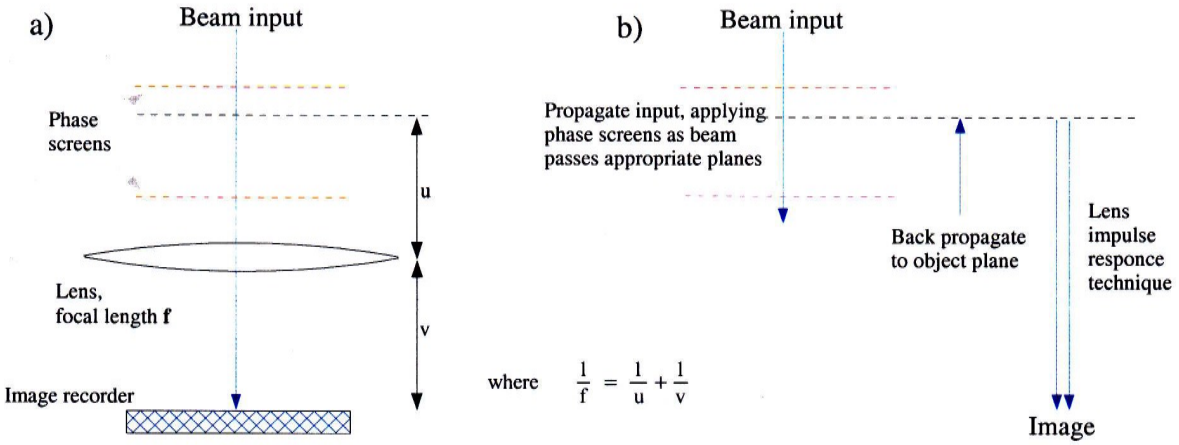


Figure 25

An optical system (a) cannot be directly represented by simulation. The system can be approximated in a way that can be simulated (b). The use of 'back propagation' where the beam is propagated a -ve distance back to the object plane.

Image intensity $I(\frac{x}{\lambda f},\frac{y}{\lambda f}) = \frac{1}{\lambda^2 f^2}|\Im[p(x,y)u_l(x,y)]_{\vec{r}\to\frac{\vec{r}}{\lambda f}}|^2$ for object at infinity $u = \infty$ (equation 23)

Image intensity $I(-x\frac{(u-f)}{f},-y\frac{(u-f)}{f}) = (\frac{u-f}{f})^2|\Im^{-1}[p(-x\lambda u,-y\lambda u)\Im[u_o(x,y)]_{\vec{r}\to\vec{r}'}]_{\vec{r}'\to\vec{r}\frac{u-f}{f}}|^2$ for object at finite distance u. (equation 24)

u distance from object plane
f focal length of lens
u_l optical field immediately before imaging lens
u_o optical field in object plane

Image formation techniques:

Object at infinity (equation 23). Note the scaling factor λf to the spatial frequencies produced by the fourier transform

Object at finite distance (equation 24) using the response of the lens (given by the pupil) to convolve with the object plane and hence produce the image plane.

Note the magnification effects changing the scaling factor in the image

The Author would like to stress that although these techniques are frequently used for optical simulations, the justification for using them is that the physical situation cannot be directly modelled as demonstrated in section 7.5.

7.7) Periodic turbulence

The Kolmogorov description of the atmosphere described in section 4.2 only concerns itself with the spatial refractive index power spectrum produced by the turbulence. It places no restriction on the temporal evolution of the refractive index of an air 'element'. As this is not described at all, it is common to assume that the refractive index pattern is fixed and does not change. This is known as the frozen atmosphere assumption. The refractive index pattern is translated into a phase screen by conforming to the turbulence power spectrum[3], but choosing a random origin for the spatial frequency component amplitudes (see equation 25). The spatial statistics of the turbulence is measured using the structure function (see equation 27).

Eqquation 25

$$\phi(\vec{x}) = e^{i\Im^{-1}_{real}[\frac{0.023}{r_o^{5/3}}|\vec{\kappa}|^{-11/3} e^{2\pi i r_{\vec{\kappa}}}]}$$

The Kolmogorov phase screen generator to produce turbulence that gives a seeing of r_0 for this layer. r_κ is a random number between 0 & 1 to provide a random spatial positioning to the fourier components. Note that a real inverse transform enforces $F(\kappa) = F(-\kappa)^*$, making $f(x)$ real.

The power spectrum for Kolmogorov turbulence is defined for a continuous atmosphere. This representation is periodic due the use of discrete fourier transforms and results in some distortion. The periodicity of the representation introduces a correlation between points in the phase screen. This is most clearly seen if you consider two points separated by the periodicity length; they must have exactly the same value. This clearly will distort the spatial statistics of the atmosphere representation (see figure 26). These distortions are manifested by a reduction in the structure function on scales that are greater than ~0.1*periodicity length. As the power spectrum of turbulence is isotropic, the ideal case would still only produce a structure function that agrees with the desired statistics up to 0.5*periodicity length. This is because the structure function extends over opposite directions and meets up a distance 0.5*periodicity length away. The structure function can be increased over these larger scales to compensate for the periodicity by boosting the lower spatial frequency components. The scheme devised by the Author to do this is shown in equation 26.

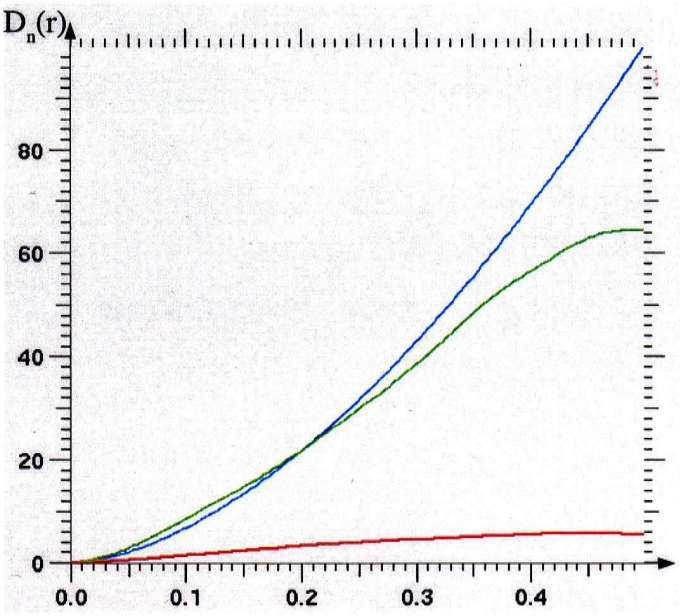


Figure 26
The structure function averaged over twenty numerical realisations.
Red is without periodicity correction
Green is using the Authors periodicity correction
Blue is the statistical structure function for non periodic Kolmogorov turbulence

$$\phi(\vec{x}) = e^{i\Im^{-1}_{real}\left[\frac{0.023}{r_o^{5/3}}\left(\frac{\alpha+|\vec{\kappa}|^{\beta}}{|\vec{\kappa}|^{\beta}}\right)|\vec{\kappa}|^{-11/3}\, e^{2\pi i r_{\vec{\kappa}}}\right]}$$

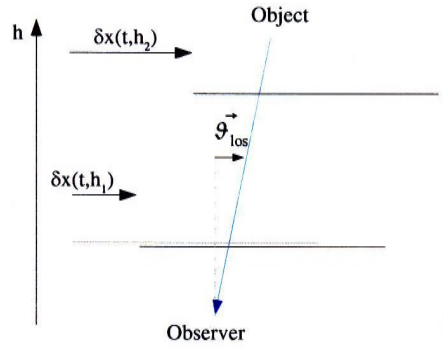
equation 26
The modified Kolmogorov phase screen generator to overcome the effects of a periodically described atmosphere. The author has found the values of α=3.1 and β=3.0 to give a satisfactory improvement.

$$D_n(\vec{\Delta}r) = \langle(\phi(\vec{r})-\phi(\vec{r}+\vec{\Delta}r))^2\rangle = 6.88\left(\frac{|\Delta r|}{r_o}\right)^{5/3}$$

Equation 27
The structure function $D_n(\Delta r)$ and the statistical prediction for Kolmogorov turbulence in a 2D plane.
$\phi(r)$ is the phase change introduced by the turbulence in a unit volume

To allow for a dynamic atmosphere, the refractive index pattern is allowed to move with a constant velocity (wind). Also the line of sight through the atmosphere may not be constant during a simulation (guide star - science object angular anisoplanatism) the turbulence phase patterns need to be translated horizontally. As the atmospheric section of the simulation is periodic, this can be achieved by changing the phase of the turbulence spatial frequency components (see figure 27 and equation 28). As the 3D atmosphere turbulence information has been approximated to 2D turbulence, there is no way to allow for

different angles of incidence sampling the turbulence differently within a turbulence plane. Projection effects (see section7.8) are inappropriate for the atmosphere as it is in actuality a 3D object rather than a 2D transmission mask. All angle of incidence dependance is provided by making the atmosphere pseudo 3D by using turbulence planes at finite heights so that different line of sights sample the turbulence differently. Different angles of incidence will cause a beam to intersect each plane in a different place and hence sample the turbulence differently.



$$t(\vec{x}+\vec{\delta x}) = \Im^{-1}\left[e^{(-2\pi i \vec{\delta x}\cdot\vec{\kappa})}\Im\left[t(\vec{x})\right]_{\vec{x}\to\vec{\kappa}}\right]_{\vec{\kappa}\to\vec{x}}$$

where

$$\vec{\delta x}(h,t) = \vec{v}(h)t + h\,\vec{\vartheta}_{los}$$

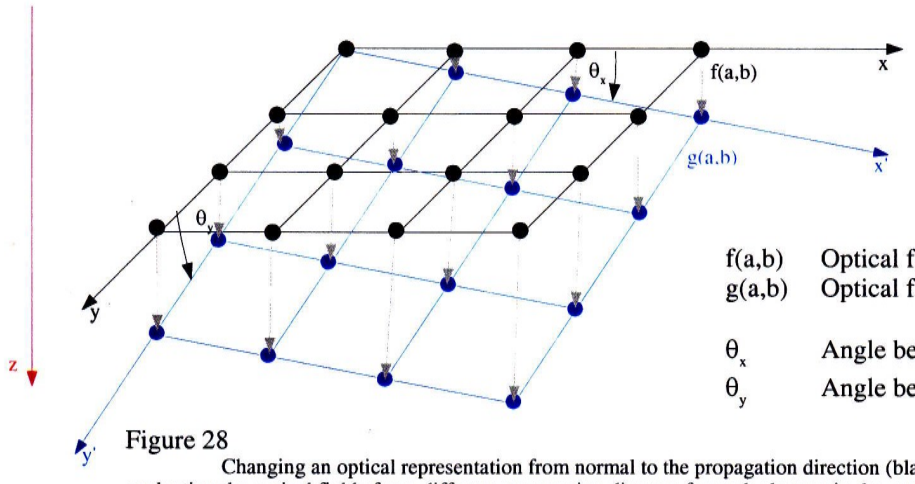
Wind motion — Line of sight through atmosphere

Figure 27 and equation 28

The relative position of the atmosphere to the line of sight (blue line) changes with the line of sight and as the wind causes layers of the atmosphere to move. To obtain the apparent atmosphere pattern along the line of sight, the periodic atmosphere is translated via the fourier shift theorem.

7.8) Phase screens applied to isolated beams

A phase screen applied to an optical beam acts as a complex valued transmission mask. The method used for optical field propagation moves the optical field representation between parallel planes normal to the propagation direction. This would cause you to naively expect that it is impossible to apply phase screens in any orientation other than in the same orientation as the optical field representation. The optical field can be propagated to non-normal planes (see figure 28), but it requires the use of unevenly sampled fourier transforms (see equation 29). To do this would be too computationally costly as fast fourier transforms could no longer be used. Fast fourier transforms scale as $n^2\log(n)$ (as there are n^2 elements in the optical grid), while a fourier transform implementation that can use an arbitrary basis and sampling set would scale as n^4. The time required for the simulation to run would be so long that it would not be useful at all.



f(a,b) Optical field in xy plane at position (a,b)
g(a,b) Optical field in x'y' plane at position (a,b)

θ_x Angle between x & x' axis
θ_y Angle between y & y' axis

Figure 28

Changing an optical representation from normal to the propagation direction (black) to an arbitrary plane (blue) requires evaluating the optical field after a different propagation distance for each element in the new plane. The points on the blue plane are projections of the points on the black plane moved along the z axis. An arbitrary plane such as shown above is described by the angles between the x & x' and y & y' axes (θ_x and θ_y).

The optical field in a plane normal to the propagation direction $f(a,b) = \sum_{c,d} \alpha_{c,d} e^{\frac{2\pi i}{n}(a.c+b.d)}$ (c,d spatial frequencies)

Expanded optical propagator for $\lambda f_{spatial} << 1$ is $H(c,d,z) = e^{2\pi iz(\frac{1}{\lambda} - \lambda\frac{(c^2+d^2)}{2l^2})}$

Thus propagating a distance z will give

$$F(c,d,z) = \alpha_{c,d} e^{2\pi iz(\frac{1}{\lambda} - \lambda\frac{(c^2+d^2)}{2l^2})}$$

$$f(a,b,z) = \sum_{c,d} \alpha_{c,d} e^{\frac{2\pi i}{n}(a.c+b.d+nz(\frac{1}{\lambda} - \lambda\frac{(c^2+d^2)}{2l^2}))}$$

The arbitrarily orientated plane can be made up of sections of normal planes propagated different distances

$$g(a,b,\vartheta_x,\vartheta_y) = h(a,b,z = \frac{1}{n}(a\ \tan(\vartheta_x)+b\ \tan(\vartheta_y)))$$

$$g(a,b,\vartheta_x,\vartheta_y) = \sum_{c,d} \alpha_{c,d} e^{\frac{2\pi i}{n}(a.c+b.d+1(\frac{1}{\lambda} - \lambda\frac{(c^2+d^2)}{2l^2})(a\ \tan(\vartheta_x)+b\ \tan(\vartheta_y)))}$$

$$g(a,b,\vartheta_x,\vartheta_y) = \sum_{c,d} \alpha_{c,d} e^{\frac{2\pi i}{n}(a(c+1(\frac{1}{\lambda} - \lambda\frac{(c^2+d^2)}{2l^2})\tan(\vartheta_x))+b(d+1(\frac{1}{\lambda} - \lambda\frac{(c^2+d^2)}{2l^2})\tan(\vartheta_y)))}$$

New spatial frequency along x' | New spatial frequency along y'

Equation 29

The optical field in an arbitrary plane can be found by changing the fourier basis set. Inspection of the new spatial frequencies show that this is not a linear basis transformation.

Approximating a plane to a normal incidence position is often used in AO models. This is known as the normal incidence approximation. This approximation would be completely removed by propagating the optical field to the non-normal plane to apply the phase screen. However this is not practical due to processing requirements as described above. By considering the projection of the mesh points on the plane, the normal approximation is partially lifted. Projection effects changing the viewed geometry of the screen are included (see figure 29).

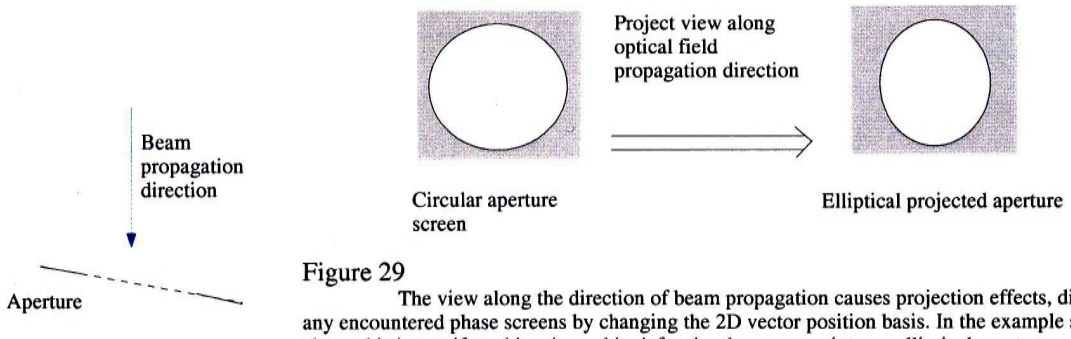


Figure 29

The view along the direction of beam propagation causes projection effects, distorting any encountered phase screens by changing the 2D vector position basis. In the example shown above this is manifested in a 'squashing'of a circular aperture into an elliptical aperture experienced by the beam.

To define the projection effects it is necessary to define a plane orientation in 3D. In order to hide the co-ordinate considerations the Author chose the definitions given in equation 30. By using angles to define a plane rather than basis vectors, orthonormality is automatically satisfied and any combination of $(\theta_x,\theta_y,\phi_z)$ are valid (except $\theta_x,\theta_y = \pm\pi$).

$$\hat{z}'\cdot\hat{x} \equiv \sin(\vartheta_x)$$
$$\hat{z}'\cdot\hat{y} \equiv \sin(\vartheta_y)$$
$$\hat{x}'\cdot(\hat{z}'\wedge\hat{x}) \equiv \cos(\vartheta_x)\sin(\phi_z)$$

(equation 30)

$$\hat{z}' \neq \hat{x}$$

θ_x measures projection of **z'** along x axis
θ_y measures projection of **z'** along y axis
ϕ_z measures the rotation of the plane about its own **z'** axis by measuring the projection of **x'** out of the **z' x** plane

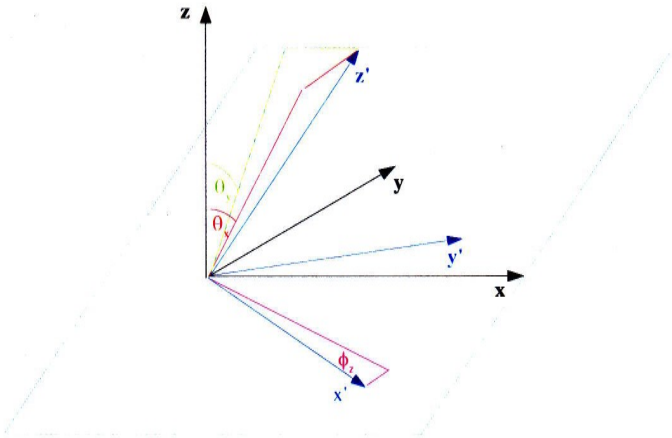


Figure 30

The global co-ordinate system **(x,y,z)** is shown in black. The plane or beam co-ordinate system (**x',y',z'**) is shown in blue. The plane or beam co-ordinate system is described by (θ_x,θ_y,ϕ_z).

The red lines show the projection of **z'** onto the **x** axis, defining θ_x.

The green lines show the projection of **z'** onto the **y** axis, defining θ_y.

The cyan box is a section of the plane formed by **z'x** vectors. The magenta line shows the projection of **x'** out of the **z'x** plane, defining ϕ_z.

Note that all basis vectors are normal and that ^ denotes cross-product

The co-ordinate system for the plane **(x',y',z')** can then be reconstructed from (θ_x,θ_y,ϕ_z) (see figure 30) using the formula given in equation 31.

$$\hat{z}' = (\sin(\vartheta_x),\ \sin(\vartheta_y),\ (1-\sin(\vartheta_x)^2-\sin(\vartheta_y)^2)^{1/2})$$
$$\hat{x}' = \cos(\phi_z)\hat{x}'' + \sin(\phi_z)\hat{y}''$$
$$\hat{y}' = -\sin(\phi_z)\hat{x}'' + \cos(\phi_z)\hat{y}''$$

where

$$\Phi = \arccos(\sin(\vartheta_x))$$
$$\vec{\omega} = \hat{x}-\hat{z}'$$
$$\Theta = \arcsin\left(\frac{\sin(\Phi)}{|\vec{\omega}|}\right)$$
$$\Psi = \frac{\pi}{2}-\Theta$$

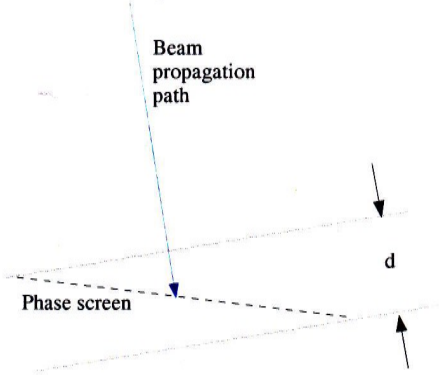
$$\sigma = \frac{1}{\sin(\Psi)}$$
$$\eta = \sigma * \sin(\Theta)$$
$$\hat{x}'' = \frac{\hat{z}' + \frac{\sigma\vec{\omega}}{|\vec{\omega}|}}{\eta}$$
$$\hat{y}'' = \hat{z}'\wedge\hat{x}''$$

Equation 31

Unit vectors (**x',y',z'**) of plane (θ_x,θ_y,ϕ_z) where (**x'',y''**) form the plane before rotation about the **z'** axis by ϕ_z

The beam grid is projected into the plane by moving the grid points along the beam propagation direction into the plane. The phase screen in one of these planes is a discrete representation of a function that can be calculated on the fly for any set of grid projections. This allows the phase screen to be displaced and placed at an arbitrary orientation without changing the function generating the phase screen. Note that the beam itself is not restricted to propagation along the global **z** axis and has its own θ_x,θ_y angles and corresponding x(z) and y(z) offsets for its propagation path.

The error produced by not propagating to the arbitrarily orientated plane is due to the evolution of the optical field. The upper bound on this error can be obtained by measuring the rms difference of the complex scalar produced by propagating the a distance d which is expressed in terms of the error in the intensity. (see figure 31 and equation 32).



$$\text{Incidence error} = \frac{|u(s+d)|^2-|u(s)|^2}{n^2}$$

Figure 31 and equation 32

The error due to non-normal incidence is due to the evolution of the optical field as it propagates. This is approximated by the intensity difference produced by propagating the optical field a distance:
$d = 2^{0.5}\,l\,\tan(\cos^{-1}(\mathbf{z}_{beam}\bullet\mathbf{z}_{plane}))$.

8) Simulation realisation

8.1) Simulator description and implemented components

It is the Authors intention to give the reader a qualitative knowledge about the simulator implementation, rather than a thorough analysis of all techniques employed (contact the Author for more details). The language used is Python 2.1 with Numeric, FFT, RandomArray and Gist extensions and has been executed on a 1.4GHz Athlon with 256MB memory using Redhat Linux 7.2.

The simulator implemented is made up of a general purpose optical and electronic simulator (see figure 32 for a toy diagram). The simulation works with discrete time steps, resolving the optical simulator time step before the electronic simulator time step. All components are inherently object orientated that make suitable adjustments to the simulation to gain access to the required data and make available references for output data as a component is created. As the data being passed is extremely large, a garbage collection system is included that intelligently removes any data as soon as it is not needed by counting the number of components that require access to it and 'ticking them off' as they access the data. As all of the data is identified via pointers, when multiple components require a single piece of data, copies of the data is made as all but the last component accesses it. Note that this requires that the data have 'good' duplication properties (no unpredictable or undesirable behaviour - see Python library reference on copy module for more information). Simulation objects can have an internal state that determines their behaviour to provide hysteresis (history dependant) behaviour. The execution order of optical or electronic components respectively is in the order they are defined by the user. See appendix for program listings

The optical simulator is built upon the theory described in section 7). Multiple optical fields are permitted to travel down a beam simultaneously where upon they all interact with the optical components at the same time. Beams may be split or combined with selection criteria for the optical fields involved (e.g. propagation direction). Optical components are permitted to sample or alter the optical fields in order to interact with it. Interactions with the optical field are driven by a geometric position map which gives the position of where each discrete optical field element intercepts the plane of the component. The optical propagation is considered to be instantaneous along the beam axis. The positions and orientation of all optical elements and fields are defined relative to a beam axis. This beam axis is the path of a geometric axial ray that does not interact with anything. Only optical fields are passed along through the optical system and notably only one signal (the beam of light) is produced on the outputs. Defined optical objects are:- (note some are diagnostic)

1. Optical field
   - Provides field representation, propagation, generic sampling and interaction code
2. Beam axis
   - Provides a structural framework for positioning and orientation as well as ordering of optical components
3. Beam splitters

- Combine or split beam axis and may have conditional selection criteria

4. Phase screens
   - Function describing a transmission screen (either static or dynamic, with or without electronic inputs) across an arbitrarily orientated plane (e.g. a deformable mirror, atmospheric turbulence or pupil)
5. Sensors
   - Uses optical field to produce an electronic output or produce simulation results (e.g. wavefront sensor or imaging component)
6. Optical field snapshot
   - Record optical fields passing a point for later analysis

The electronic simulator models an electronic system as an ensemble of black boxes that have one output data line and can accept an arbitrary number of input data lines. The notable differences to the optical simulator are that time is treated as a continuous which is evaluated discretely and the inputs and outputs are of undefined types. As an electronic component processes a data signal, an appropriate time delay is added to the time of any resulting output signal. The input signals are evaluated in time of arrival order as multiple signals can pass down any signal line. The requirement on the electronic components is that their processing can be resolved (computationally) immediately (e.g. any time delay buffering to avoid processing must be done internally). The electronic simulation connects to the optical simulation via some optical components being able to receive or send electrical signals. Defined electronic objects are:- (note some are diagnostic)

1. Wavefront sensors
   - Produces a signal containing an optical path difference description
2. Deformable mirrors
   - Applies the negative optical path difference based on a received optical path difference signal
3. Electronic signal snapshot
   - Saves all signals received

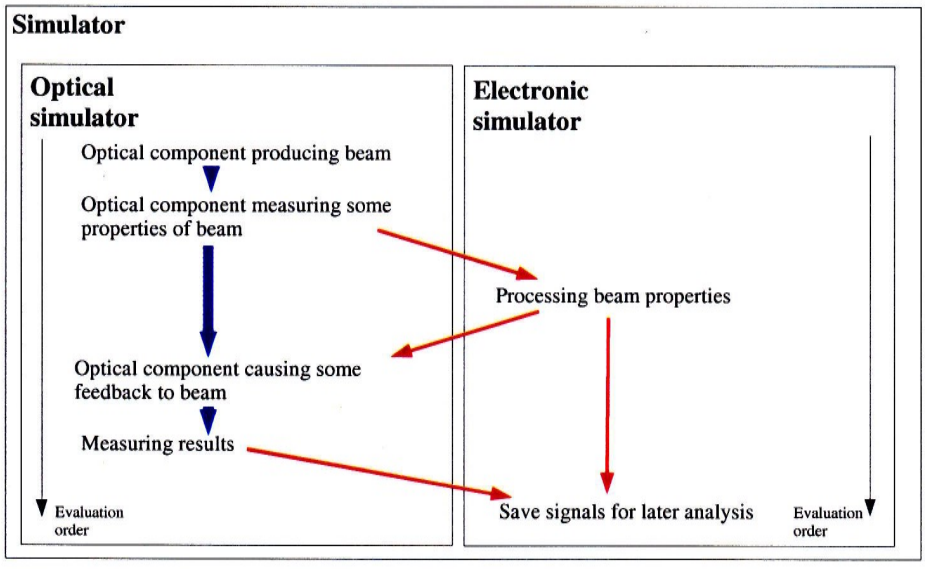


Figure 32
Toy model of simulator. The black arrows indicate evaluation order within the optical & electronic simulators (optical evaluated before electronic). The blue arrows show the flow of optical beam data being moved about the system. The red arrows similarly show the flow of electronic signals, which incur a time delay as they are processed.

8.2) Simulator demonstration

To demonstrate that the simulator functions the Author produced a short program 'testing.py' given in the appendix (Note electronic versions available on request). The command line for execution (in directory with all code) is 'python2.1 testing.py'. What is presented below is the graphical and text output generated along with an explanation. The program is designed to show several of the simulator functions that can been seen to be appropriately functioning.

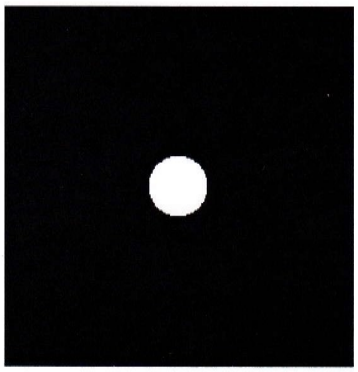
Upon starting the demonstration, the text below (figure 33) should be printed and a 'pygist' window should appear. If the window does not appear or an error message is reported, try again. If this problem persists the correct modules are not installed or the gist environment is incorrect. Note that the data output path in 'testing.py' must be changed to something appropriate for the computer used to run the program (that contains no file or directory named 'sim' - program expects to be able to create this directory from scratch). Contact the Author for a demonstration on aipc18, where the software was developed.

```
"
Welcome to simulation testing (4 rounds of tests)
Please respond to prompts for displays of processed data (remember to select this text output window in
order to provide input).
Please be patient as simulation is being run to produce output data.

Testing gist display - restart if error reported (known to work correctly on aipc18 as user rmd)

Type return to continue
"
```

Figure 33
Text message produced on starting the demonstration program 'testing.py'

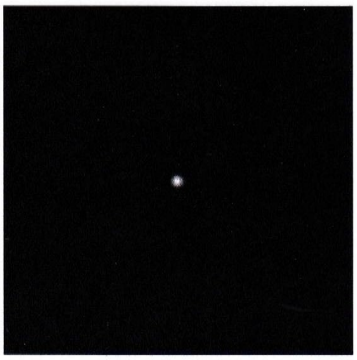
8.2.1) Image formation

This is the 1st demonstration and is intended to show that the imaging components function. A circular aperture is illuminated by normal plane waves. The aperture is imaged by focussing a camera on it (see figure 34). The far field image is then taken showing a fuzzy radially symmetric blob (figure 35) and the central region is displayed zoomed up as a surface intensity plot, allowing you to see the first diffraction ring (figure 36).



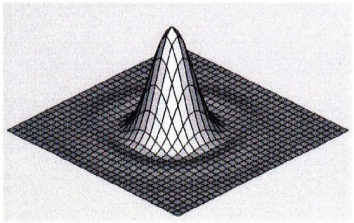
```
"
Performing 1st simulation
Imaging a circular aperture and far field image as a demonstration of
simulation imaging components

Simulating
Simulation ended
Image taken with camera 1.0m from object object plane (impulse responce
technique).
Scale = 0.006
Type return to continue
"
```

Figure 34
Image produced by a camera focussed onto a circular aperture from 1m away. Note the scale of the picture is0.006x0.006m.



```
"
Image taken with object plane at infinity (far field technique).
Scale = 0.006
Type return to continue
"
```

Figure 35
Far field image produced by diffraction

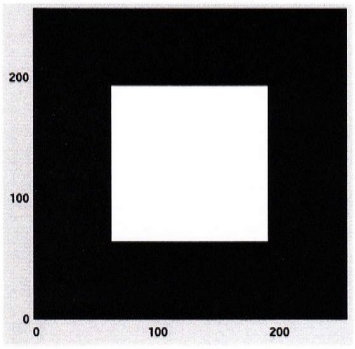


```
"
Surface intensity plot of central region.
Scale = 0.00075
"
```

Figure 36
Surface intensity plot of the centre of the far field image. The first diffraction ring is visible around the central maximum.

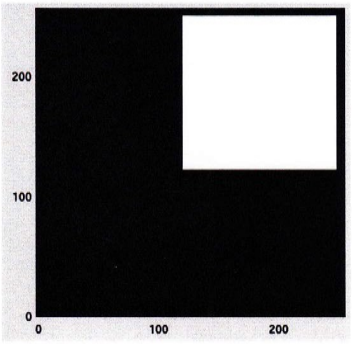
8.2.2) Co-ordinate system demonstration

This is the 2nd demonstration and is intended to show that the simulation correctly projects an arbitrarily orientated plane into an optical beam. A square aperture 1x1m across is illuminated by plane waves and then imaged. The aperture definition is not changed, but the plane the aperture is on is moved and rotated to give a different projection (see figures 37 - 42). The images shown have the grid co-ordinates along the axis to see how the aperture has been moved.



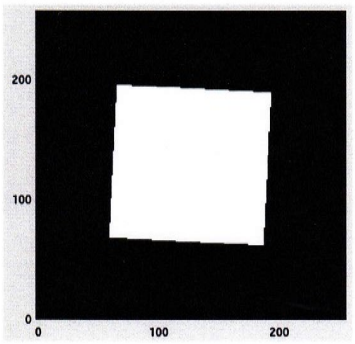

```
"
Performing 2nd simulation
A 1.0x1.0 m square aperture with various offsets and orientations to
demonstrate the simulation co-ordinate system

Simulating
Simulation ended
Type return to continue
Image of square aperture centred at (0,0)
Scale = 2.0
Type return to continue
"
```

Figure 37
The 1x1m meter aperture with zero displacement and orientated normal to the beam




```
"
Image of square aperture centred at (0.45,0.45).
Scale = 2.0
Type return to continue
"
```

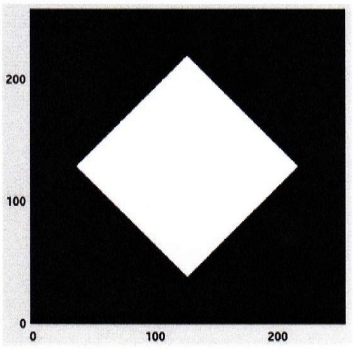
Figure 38
The square aperture displaced by (0.45,0.45)m placing it 0.05m from the grid edge




```
"
Image of square aperture centred at (0,0) rotated by 0.05 radians around
l.o.s. axis.
Scale = 2.0
Type return to continue
"
```

Figure 39

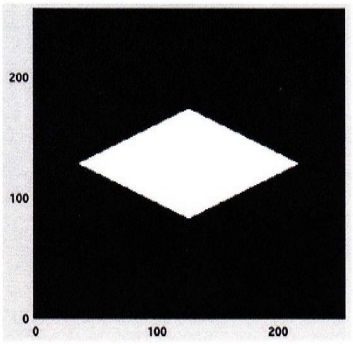
The plane is placed at the origin and rotated by 0.05 radians about plane z axis.




```
"
Image of square aperture centred at (0.45,0.45) rotated by pi/4.0
radians about l.o.s. axis
Scale = 2.0
Type return to continue
"
```

Figure 40

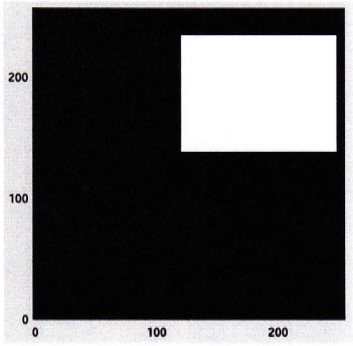
The plane is now rotated by $\pi/4$ radians about its z axis




```
"
Image of square aperture centred at (0,0) rotated by pi/3.0 radians
about x_axis and pi/4.0 radians around l.o.s. axis.
Scale = 2.0
Type return to continue
"
```

Figure 41

The plane has been rotated to θ_x = pi/3.0 radians. This hinges the plane about a horizontal line in the image causing a projection effect, shrinking the vertical image size by a factor of two. The plane is then rotated about its z axis, which no longer coincides with the propagation direction, to form the diamond shape seen in figure 607, but squashed by projection.




```
"
Image of square aperture centred at (0.45,0.45) rotated by 0.7227
radians about x-axis.
Scale = 2.0
"
```

Figure 42

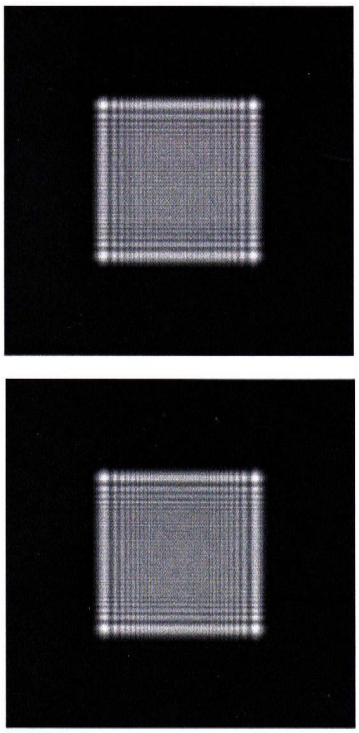
The plane has been displaced to (0.45,0.45) and rotated so that θ_x = 0.7227 radians. This causes a projection effect, squashing the vertical image direction by a factor of 4/3. Compare with figure 605 to see the projection effect.

Note that the plane co-ordinate system applies θ_x and θ_y before applying ϕ_z. This is a direct result of the co-ordinate system definition (see section 7.8) used to describe a plane and that rotations do not commute.

8.2.3) Fresnel propagation demonstration

The 3rd demonstration is intended to show that the simulation does produce Fresnel propagation. A square aperture of side 0.02m is illuminated normally by a plane wave. The optical field is propagated 0.5m and then imaged. The discrete representation causes an estimated error of 0.04 (done by propagation code, see section 7.8). This is then repeated using the Fresnel integrals. These are numerically integrated (to an accuracy of 1.3×10^{-5}) and then used to construct the image of the

equivalent Fresnel diffraction pattern. The two images are then compared (see figure 43) and a rms difference of 0.024 is measured.



```
"
Performing 3rd simulation
Demonstration of fresnel propagation effects by comparison with results
given by the cornu spiral.
Using a square aperture of side 0.002m in grid of side 0.004m after a
propagation distance of 0.05m.

Simulating
Simulation ended
Numerically evaluating fresnel integrals & finding same image
Tabulating Fresnel integrals to 1.20556718699e-05 accuracy

Type return to continue
Intensity pattern 0.05m away from aperture according to wave optics (err
estimate due to discrete representation = 0.0394396007712 )
Scale = 0.004
Type return to continue
Intensity pattern 0.05m away from aperture according to fresnel
integrals.
Scale = 0.004
Type return to continue
Rms difference is 0.0234219171279
"
```

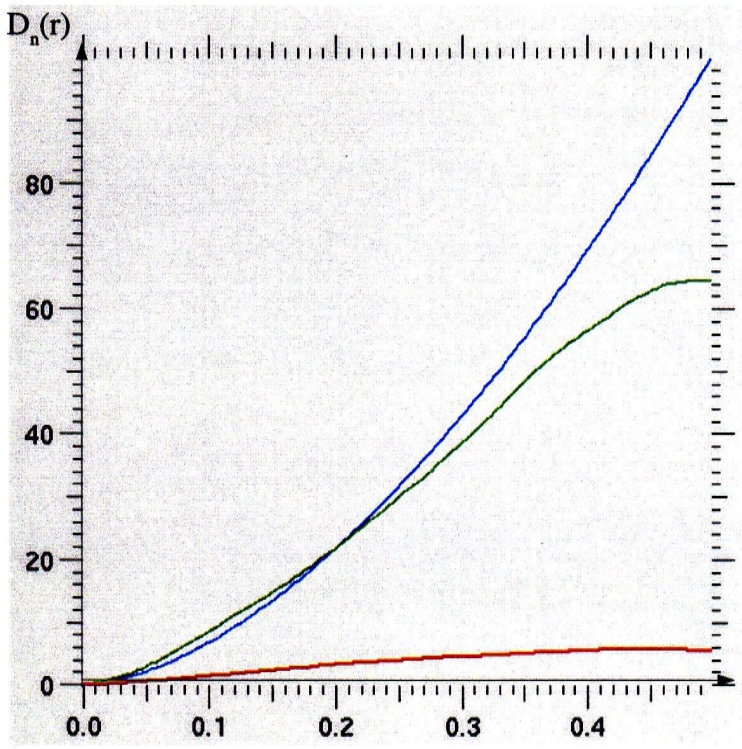
Figure 43

The image of the diffraction pattern produced by a 0.002x0.002m square aperture 0.5m away. The Fresnel prediction is on the top left and the wave optics prediction is on the bottom left. The text produced by the simulation is shown above. Note the similarity of the images down to the smallest features a pixel or so in size, on the scale of the discrete representation.

The intensity of the light passing through the aperture has been chosen to be unity, allowing direct comparison between the estimated fractional representation error (0.04) and the rms difference (0.024) between the wave optics and numerically integrated Fresnel integrals (tabulation error estimated as $1.3e^{-5}$). The error estimate is successful in that it is of the same order of magnitude as the rms difference and slightly exceeding it so the error estimate is an upper bound. Comparison of the images in figure 43 shows that the wave optics and Fresnel integrals have good agreement down to the scale of the grid used (256x256) at which point the structure of the two images is seen to differ. This discrete representation error can be made arbitrarily small by using arbitrarily large grids with arbitrarily fine resolution. A practical choice is dictated by the desired signal to noise ratio in the simulation results.

8.2.4) Kolmogorov turbulence

The 4th demonstration is the realisation of Kolmogorov turbulence screens. 20 realisations of a r_0=0.1 turbulence screen have their structure functions measured. This is done with and without the Authors 'periodicity correction' to demonstrate the difference (see section 7.7), as can be immediately seen in figure 44.




```
"
Performing 4th test
Demonstration of the realisation of
kolmogorov with and without the authors
modification (using averages over 20
realisations)
Using a grid of side 1m with r0=0.1m

Type return to continue
Structure functions of kolmogorov
turbulence.
Blue: statistical realisation
Red: numerical realisation (suffering from
periodicity)
Green: numerical realisation (periodicity
improvement)

Type return to continue
Done
"
```

Figure 44

The structure function of the Kolmogorov turbulence produced by the simulation (green) with 'periodicity correction' and (red) without the correction. The statistically desired result is the blue line. Note that as this is an average of several realisations, there is significant variation in this plot when it is produced.

The effects of periodicity can be seen as the separation of two points are of the scale of the periodicity length. Without the Authors empirically devised correction to the periodicity effects on the atmosphere then it can be seen from figure 44 that the atmosphere would not at all be accurately described. The atmosphere statistics produced are 'good' up to 1/4 - 1/3 of the periodicity distance as shown by the close agreement between the atmosphere realised and the statistically desired atmosphere up to this scale. The deviations in this range are to be expected as the desired structure function is a grand ensemble average while it is being compared to the averaged structure function of 20 realisations. Some variation can therefore be seen between consecutive runs of the demonstration program, indicating that there is significant variation in individual atmosphere realisations.

9) Telescope system under comparison

In order to perform a comparison with Richard Wilson's statistical model, the same telescope and atmosphere configuration must used. The primary limitation is the system used by the statistical model. It uses a single dominating atmospheric layer to introduce turbulence. The wavefront sensor used directly senses the phase of the optical field and decomposes it into the first 10 zernike modes. The deformable mirror supplies an optical path difference built up from the first 10 zernike modes. There is notably no time delay involved in the adaptive optics system for the statistical model. The simulated system is shown in figure 45. The wavefront sensor and deformable mirror are pupil conjugated.

The appropriate wavefront sensor for the wave optics simulation is produced by 'unwrapping' the argument of the complex scalar used to represent the optical field (see figure 46). The limiting criteria for doing so is that the argument cannot change by more than π between neighbouring grid elements. This produces the relative phase of the optical field across the optical field which can be decomposed into zernike components and transformed into optical path differences using the near field limit (i.e. no evolution on the scale of the wavefront retardation).

The deformable mirror constructs the optical path differences from supplied zernike modes, converts this to a phase difference and wraps the phase differences into a transmission mask. The finite extent of the deformable mirror is included by having zero transmission beyond the mirror edge.

The images produced (arbitrary units) are integrated (averaged) over a period of time to produce peak intensities and point spread functions. In order to calculate the Strehl of the final image, a diffraction limited version of the telescope is simulated for a single time step (turbulence & AO system absent).

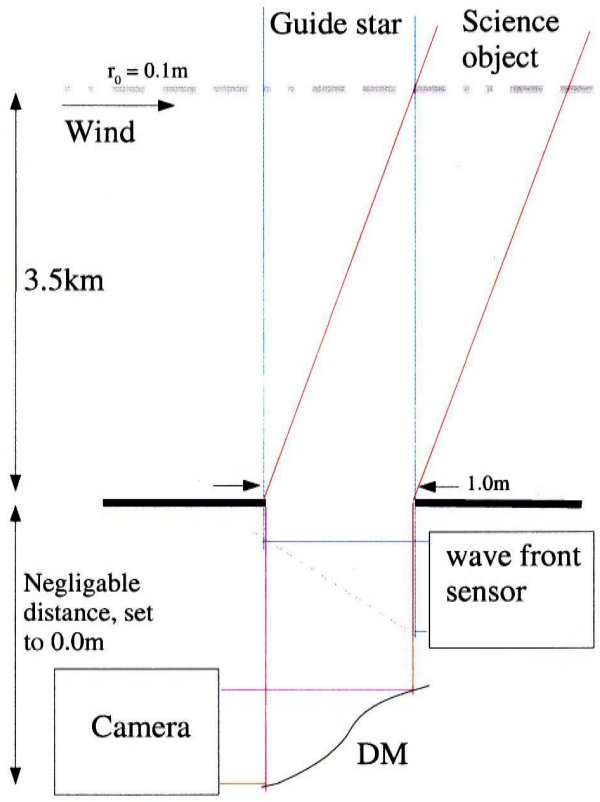


Figure 45

A diagrammatical representation of the system under comparison. The propagation within the AO system is removed by placing all components in the telescope up against one another, giving a zero propagation length.

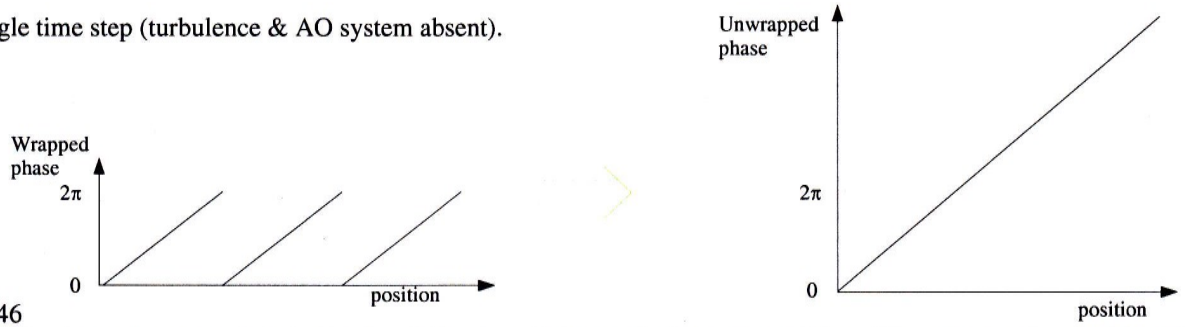


Figure 46

The phase unwrapper works by looking for neighbouring grid elements that have a wrapped phase difference greater than π and then adjusting the phase by integer multiples of 2π in order to make the unwrapped phase be continuous. This can be done in 2 dimensions for use as part of a wave front sensor

10) Results

Some example instantaneous and time integrated images are shown in figures 47 and 48. The corrected image in figure 48 is with the adaptive optics turned on and is clearly a drastic improvement over the uncorrected seeing limited image.

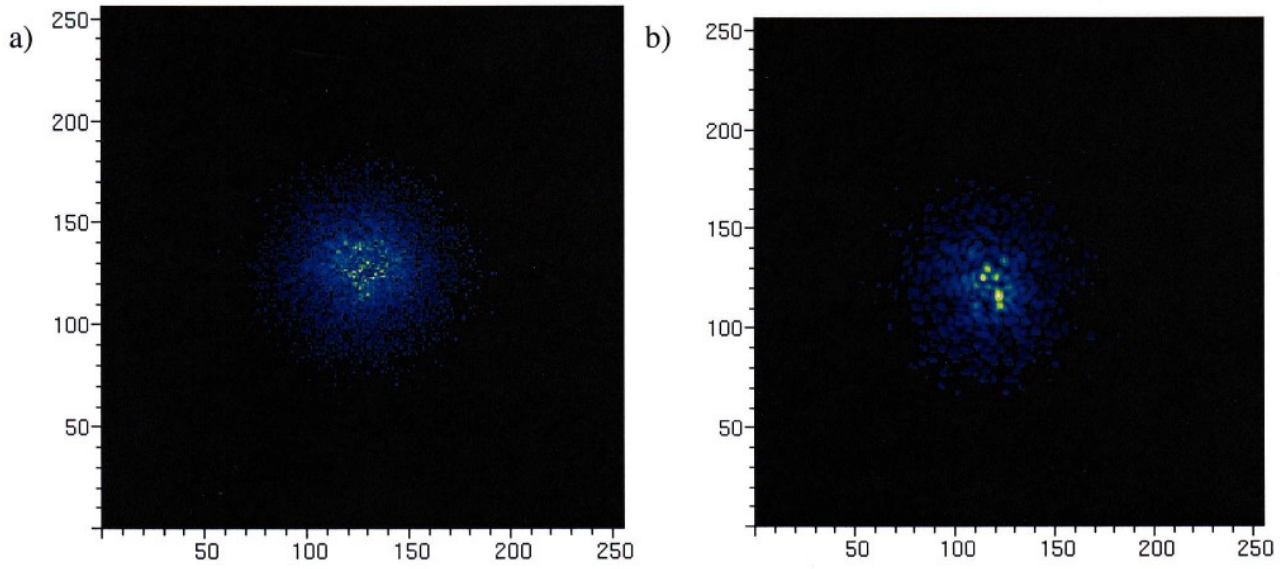


Figure 47
The instantaneous guide star images formed a) with adaptive optics system and b)without.

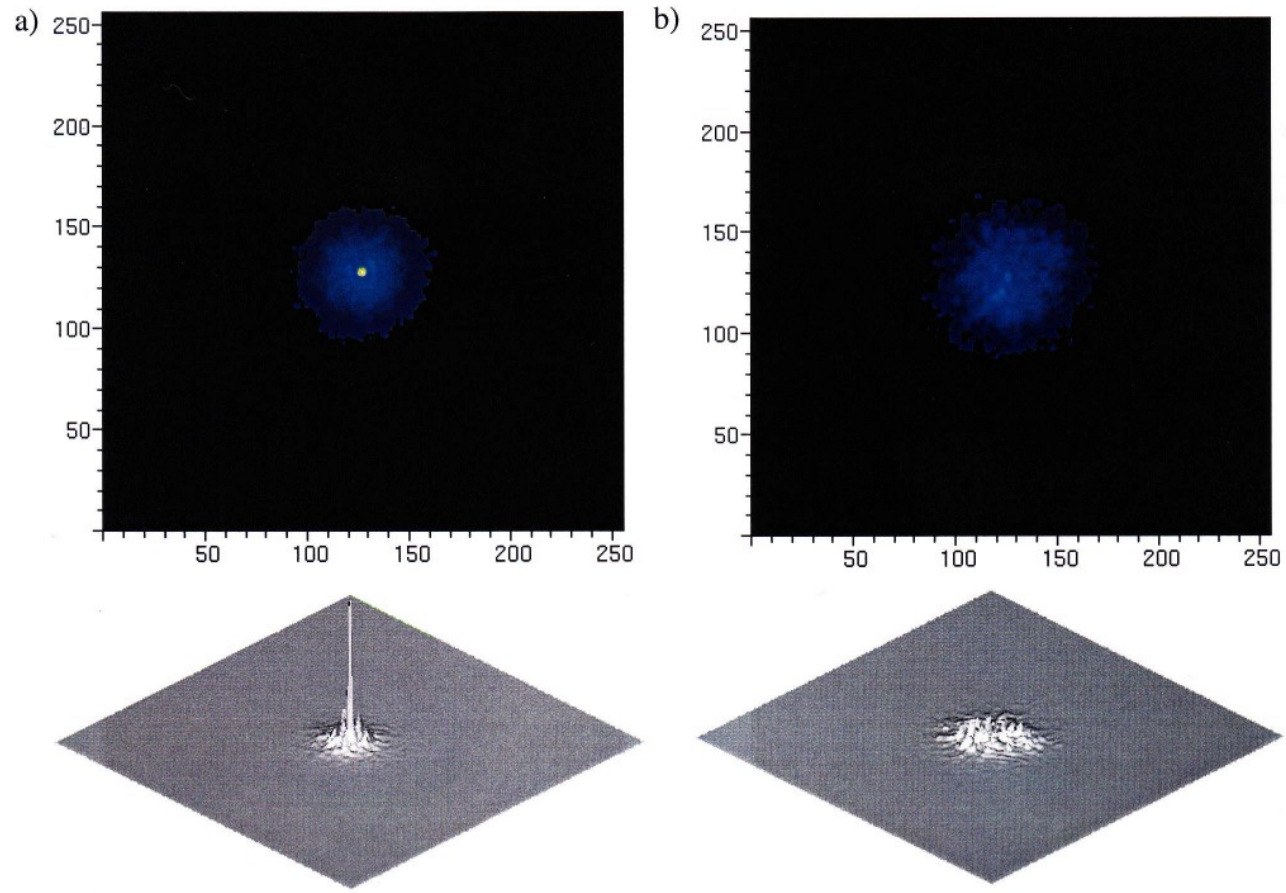
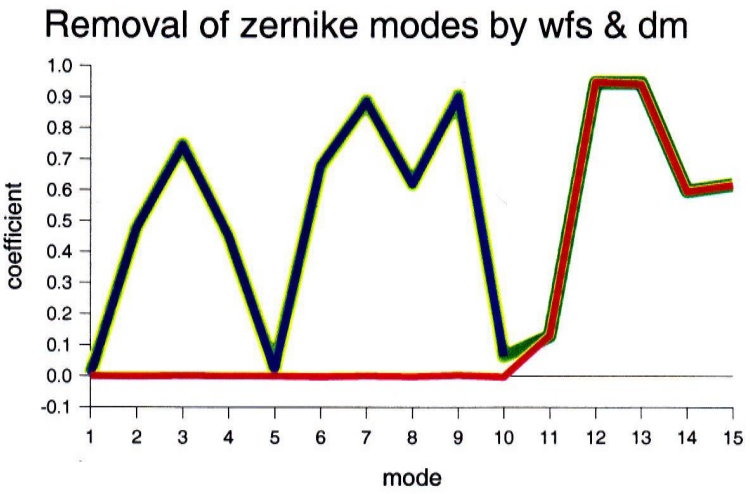


Figure 48
The integrated images formed and surface intensity plots for guide star imaging a) with adaptive optics system (Strehl = 0.145) and b) without (Strehl = 0.020) for a 5 second integration using a 3m grid with 0.02s and 0.0118m resolution. The image scale is 8.8 arcseconds across

The corrected instantaneous images are produced by the performance of the wave front sensor - deformable mirror

system. The optical path differences in the beam are sensed in zernike modes. Only the 2nd-10th modes are sensed by and removed by the adaptive optics system, 'flattening the wavefront'. The implemented wavefront and deformable mirror system is a high pass zernike mode system with a lower order mode rejection fidelity of 99.8% in the absence of scintillation. It is worth noting that this small degradation can be explained by the discretisation of the zernike modes. The normality of the modes is only true in the limit that the modes are continuous. As you consider discretised representations of the zernike modes, the normality is removed to a small degree. This leads to the rejection fidelity not being 100% as 'cross-talk' exists between the modes. This is a property of the wavefront sensor under finite simulation resolution which can be improved by operating the simulator at a higher spatial resolution if computationally feasible. Figures 49 and 50 demonstrate this for an input made up of random zernike modes for the first 15 modes.



mode	Input	sensed	Measured output
1	0.0000	0.0000	0.0000
2	0.4750	0.4756	-0.0006
3	0.7440	0.7430	0.0010
4	0.4481	0.4505	-0.0002
5	0.0208	0.0191	0.0000
6	0.6723	0.6769	-0.0016
7	0.8848	0.8834	0.0014
8	0.6165	0.6173	-0.0008
9	0.9022	0.8990	0.0031
10	0.0616	0.0641	-0.0025
11	0.1282		0.1310
12	0.9441		0.9479
13	0.9435		0.9412
14	0.5938		0.5947
15	0.6158		0.6164

Figure 49

A zernike wave front sensor and a dm applied to a 'toy' input wave front composed of random coefficients for the first 15 zernike modes (green). The input is sensed by a wave front sensor, measuring the 2nd-10th modes (blue) which are then removed by a deformable mirror. The remaining modes are left in the beam (red). The compensation error in this case is approximately 0.2% for corrected modes. Note that the measured output is found by using a higher order wave front sensor The graph above is of the data shown on the right,

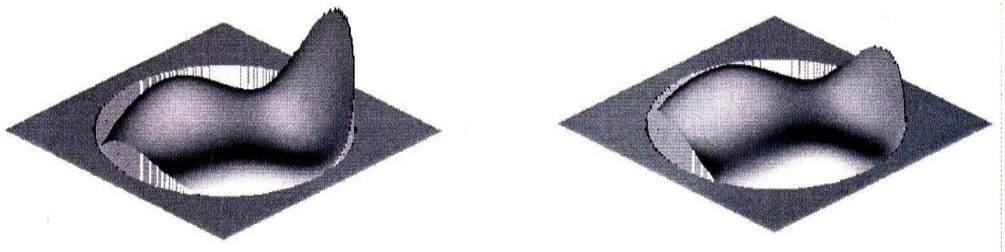


Figure 50

The wave fronts before (left) and after correction (right). The plots are directly comparable and are the wave fronts from figure 802. Note that the circular pupil is the area the wave front sensor and deformable mirror is defined on.

The integrated images are an average of the telescope response to the environmental conditions. Clearly any measured Strehl is dependant on the number of instantaneous images averaged and the degree of atmosphere evolution that occurs during the integration. The number of instantaneous images used for integration can be investigated by changing the integration time. The effect of environmental conditions can be investigated by changing the wind speed to simulate a slowly or rapidly changing atmosphere (see figure 51). The repeatability for a wind speed of 10 m/s is shown in figure 52.

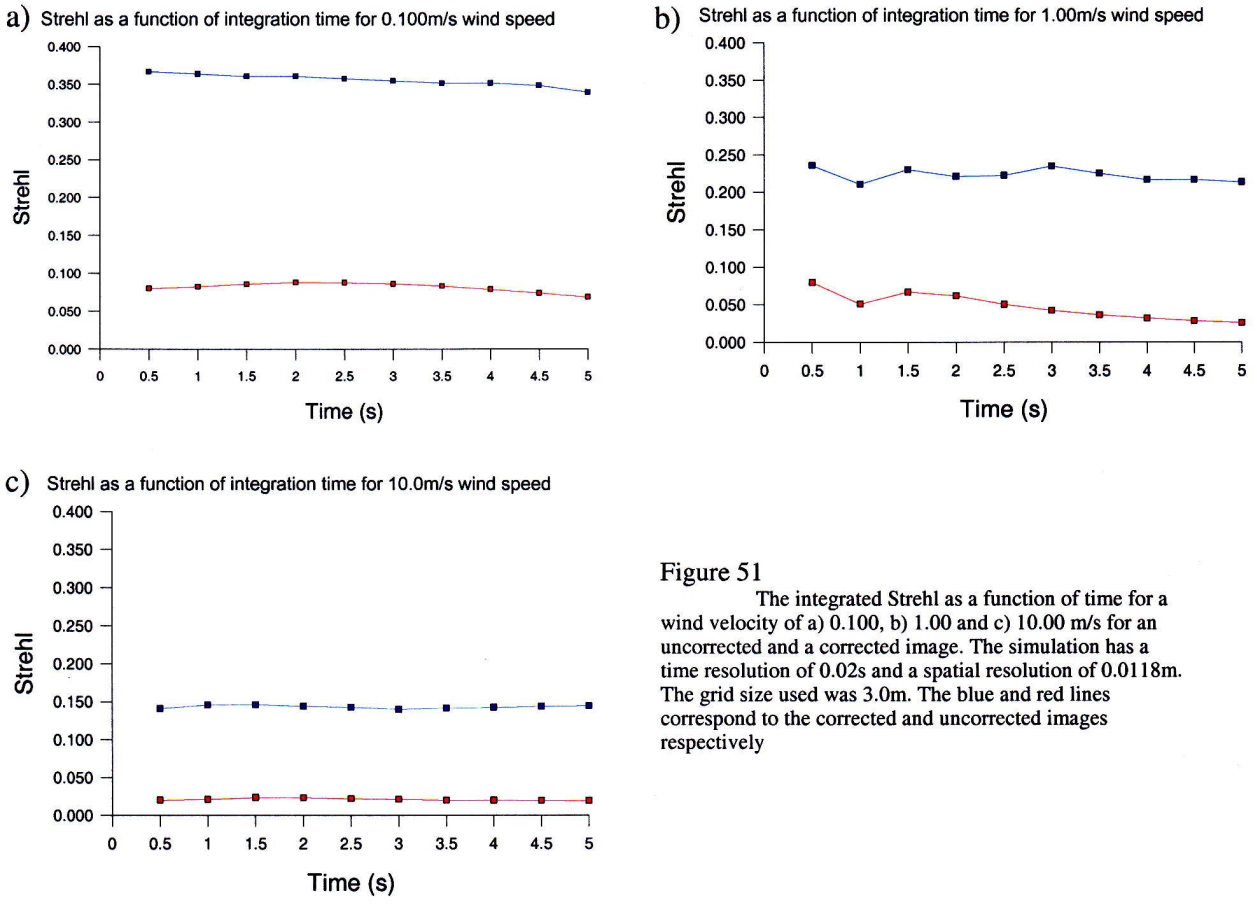


Figure 51
The integrated Strehl as a function of time for a wind velocity of a) 0.100, b) 1.00 and c) 10.00 m/s for an uncorrected and a corrected image. The simulation has a time resolution of 0.02s and a spatial resolution of 0.0118m. The grid size used was 3.0m. The blue and red lines correspond to the corrected and uncorrected images respectively

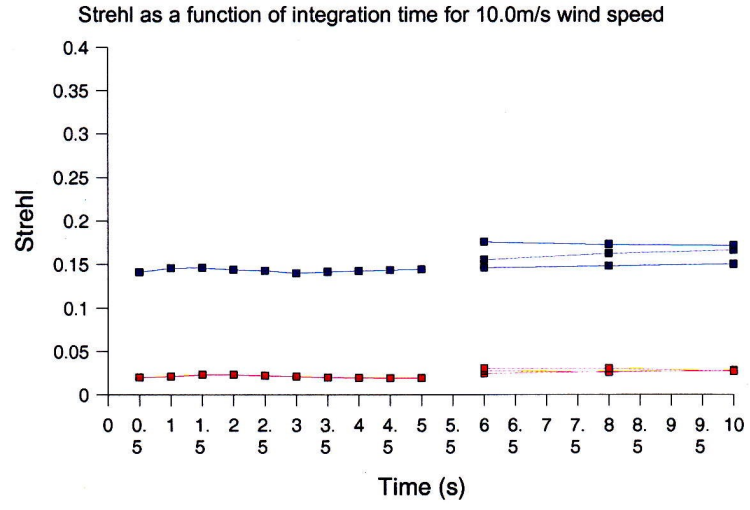


Figure 52
For a Wind speed of 10 m/s, the Strehl for 4 simulation runs using various integration times. The time resolution used was 0.02s and the spatial resolution of 0.0118m over a 3m grid. The 0.5 to 5s integration times are those shown in figure 804. Three other simulations with integration times between 6 and 10s are also shown. The blue and red lines correspond to the corrected and uncorrected images respectively

The atmosphere must be suitably sampled in order to ensure that 'local' (i.e. on scales much less than periodicity length)

do not dominate the simulation. The wind speed and simulation time step frequency determine how the atmosphere is sampled. The time step frequency is fixed at 50Hz, a suitable operating frequency for a real adaptive optics system. Figure 51 shows the corrected Strehl as a function of integration time for 0.1, 1 & 10 m/s wind speed. These particular results have been selected in order to show that there can be a significant effect on the measured Strehl for low wind speeds where the atmosphere is poorly sampled and some good local conditions along the line of sight can drastically increase the corrected Strehl. This is the situation in figure 51 a) & b). The atmosphere realisation has not been sufficiently well sampled to produce an accurate result. Higher wind speeds result in a greater atmosphere 'area' being sampled and hence a more reliable result.

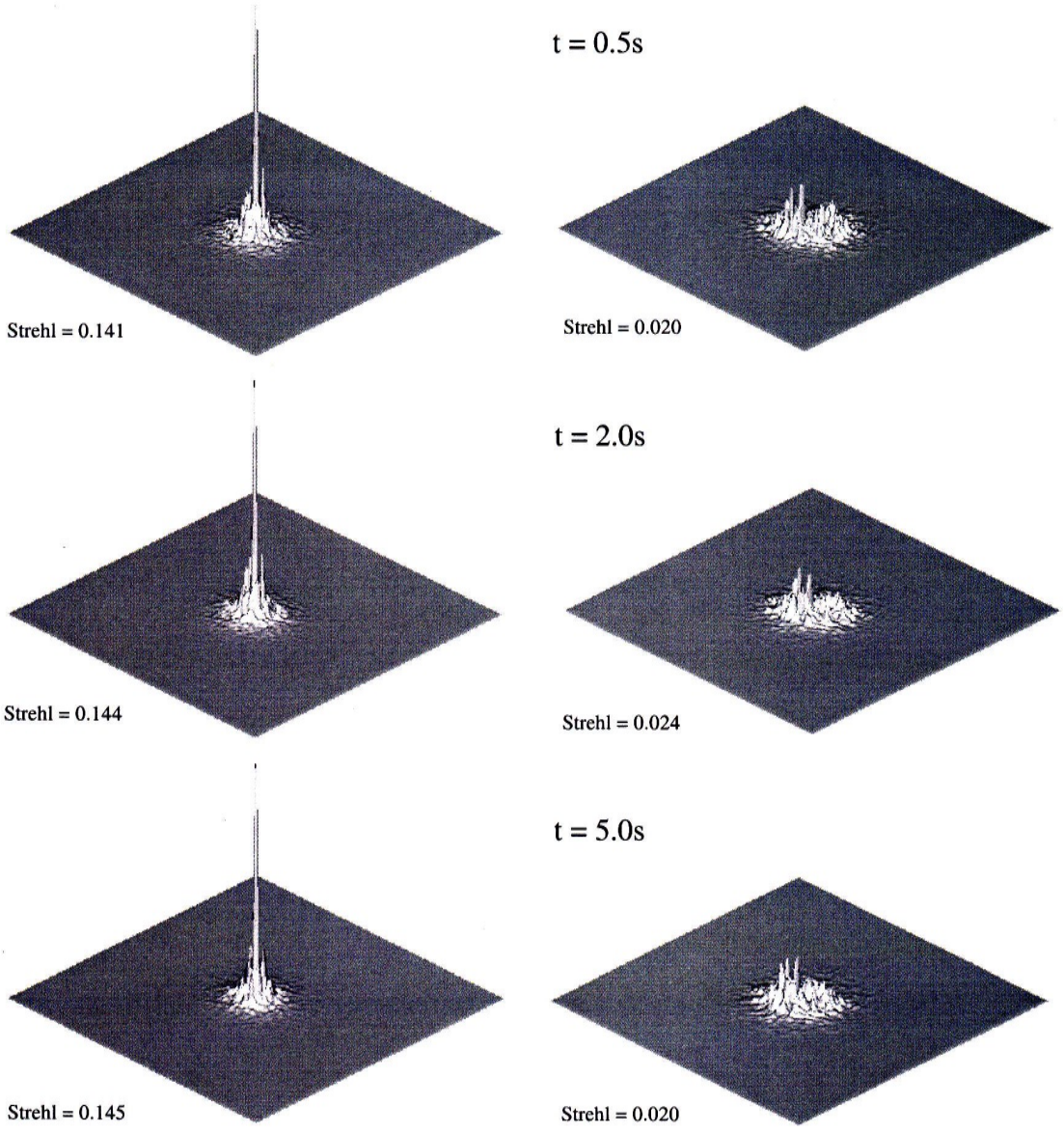


Figure 53
Measured point spread functions at times 0.5s, 2.0s and 5.0s for an imaged guide star with and without the adaptive optics system on the left and right respectively. The image scale is 8.8 arcseconds across.

The instantaneous images formed with and without correction are are both made up of multiple 'spots' (see figure 47) that move from instantaneous image to image. The time integrated images are quire different however. The corrected image has

a central peak surrounded by some noise while the seeing limited, uncorrected image is a diffuse spot. The contributions from individual instantaneous images have built up the central peak by being held at the centre. The surrounding noise is due to the finite number of corrected zernike modes leaving some atmospheric distortion to pass through into the image. The time averaging has reinforced the central peak and spread the noise energy over a larger part of the image. This noise can be clearly seen in the seeing limited spot as the image has been formed from the distortions causing energy to be distributed over a large region, preventing the formation of a narrow peak.

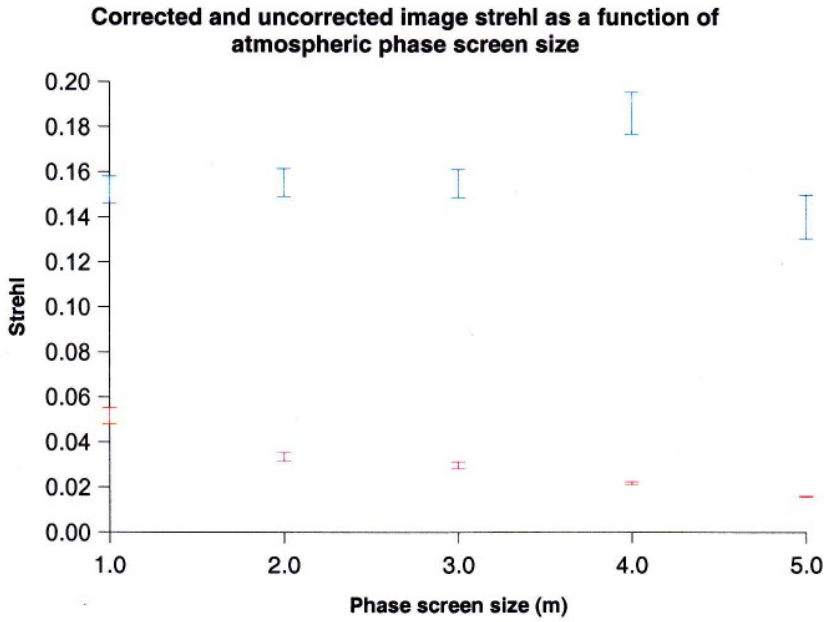


Figure 54

The measured Strehls for corrected (blue) and uncorrected (red) guide star images using a 1.0s integration time for a 1.0m telescope as a function of the turbulence phase screen size. The time steps are 0.02s. The spatial resolutions and tabulated Strehls are shown in the two tables below

Phase screen size (m)	Spatial resolution (m)
1.0	0.004
2.0	0.008
3.0	0.012
4.0	0.016
5.0	0.020

Figure 54 shows the effect of changing the atmospheric turbulence phase screen size for a 1.0m telescope. The results are dominated by two effects. As the phase screen size increases, the uncorrected Strehl decreases. This is due to the statistics of the atmospheric turbulence only following the desired statistics for distances smaller than the screen size. As the phase screen size approaches the telescope diameter, the turbulence is under represented and produces fewer low order mode distortions. This results in a higher Strehl for the uncorrected image at smaller screen sizes. This has a negligible effect on the corrected image Strehl as these low order modes are removed by adaptive optics system anyway. As the phase screen size increases, the spatial resolution has been decreased as the number of grid points used is constant. Up to and including 3.0m screen sizes the corrected Strehl is constant to within the error bars. For a 4.0m screen size, the Strehl has increased. This is because the spatial resolution is no longer fine enough to represent the higher order distortion modes and so they are no longer present in the light. The turbulence is again under represented, but this time in the absence of the corrected lower order modes it has the noticeable effect of increasing the corrected image Strehl. For a 5.0m screen size, the resolution is not fine enough for the wavefront sensors to work correctly. The wavefront sensors sense the phase of the field by 'unwrapping' the argument of the complex scalar (range 0 to 2π) describing the field into an unbounded phase (i.e. in range $-\infty$ to ∞) quantity (see sections 4.1 and 9). This relies on the assumption that the phase change between neighbouring grid points is small (always $< \pi$). If this condition is not true then the

wavefront sensor does not correctly reconstruct the phase and so cannot sense it correctly. The adaptive optics system is then not providing atmospheric compensation as it should as it cannot sense the field properly under these conditions. The result is that the Strehl of the corrected image has been dramatically reduced.

In addition to the phase variations introduced by the atmosphere there are also intensity variations in the optical field across the pupil which are shown in figure 55. This leads to scintillation in the wavefront sensor if there is a minimum light intensity required for sensing the atmospheric optical path differences (see figure 56) producing a degradation in the corrected image Strehl (see figure 57).

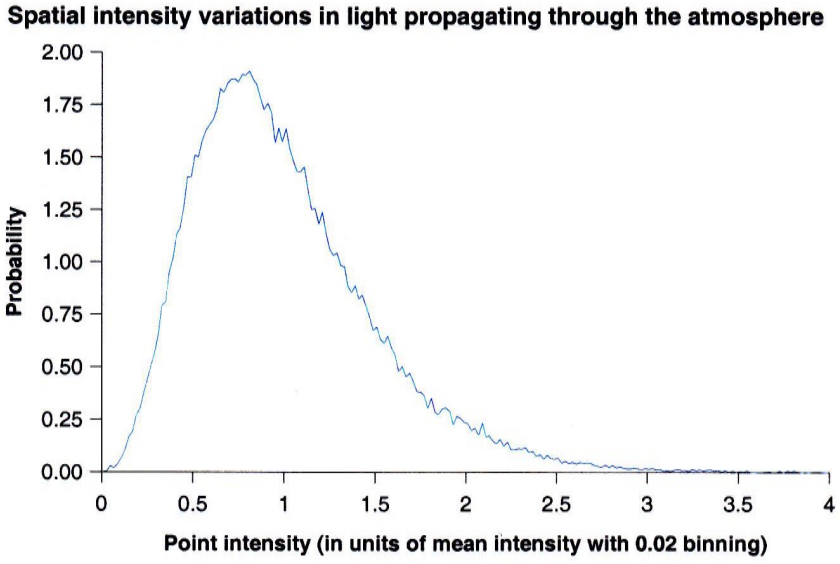


Figure 55
The variation in intensity of light from the guide star after propagating through the modelled atmosphere. The distribution is for the intensity at the limit of the simulation resolution, 0.012m in this case. The intensities have been binned with 0.02 bin width and are in units of the mean received intensity from the guide star.
99.96% of the pupil has an intensity under 4 (the section of the distribution range shown here) with a variance of 0.248 (in units of the mean intensity).

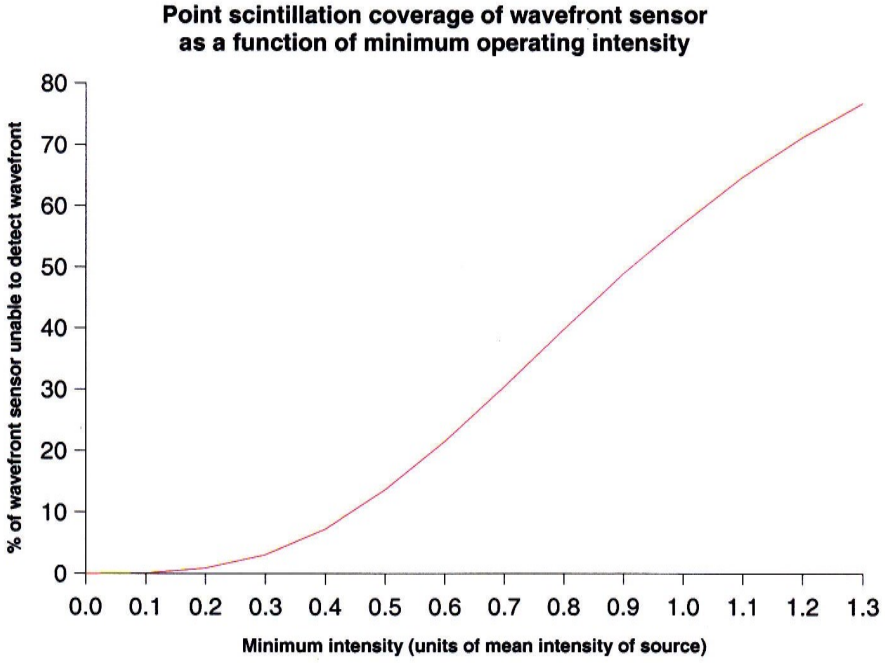


Figure 56
The percentage of the wavefront sensor unable to sense the wavefront due to a limiting intensity threshold imposed by the sensing device.

As the statistical model did not include scintillation of the optical field the response of the wavefront sensor to incident light amplitude is undefined. Correspondingly the Author was free to choose a suitable wavefront sensor behaviour. If the light

is below an arbitrary sensing threshold (which can be chosen), then the sensed wavefront for that point is random between the extreme values of the sensed optical path difference. This introduces noise to sections of the wavefront sensor as the scintillation occurs. For comparison with the statistical model, the sensing threshold is set to zero, allowing sensing regardless of the intensity variations. A non-zero sensing threshold causes scintillation to prevent sensing over a given fraction of the sensor (see figure 56). As the wavefront sensor has a resolution equal to that of the simulation, it senses the wavefront at every grid point within its circular shape. The intensity distribution (see figure 55) may have insufficient probability below the sensing threshold to cause scintillation in a Shack-Hartmann sensor due to the finite collection area, but it can cause scintillation in the zernike wavefront sensor as the collection area is point like.

As the scintillation increases from zero the wavefront sensor cannot sense the field as it should. The adaptive optics system cannot provide the atmospheric compensation it was designed to. This results in a reduction in the corrected image Strehl which decays to the uncorrected image Strehl in the limit of severe scintillation. This is shown in figure 57. The non monatomic behaviour of the image Strehl about the 40% scintillation point is interpreted as a statistical fluctuation and is not significant.

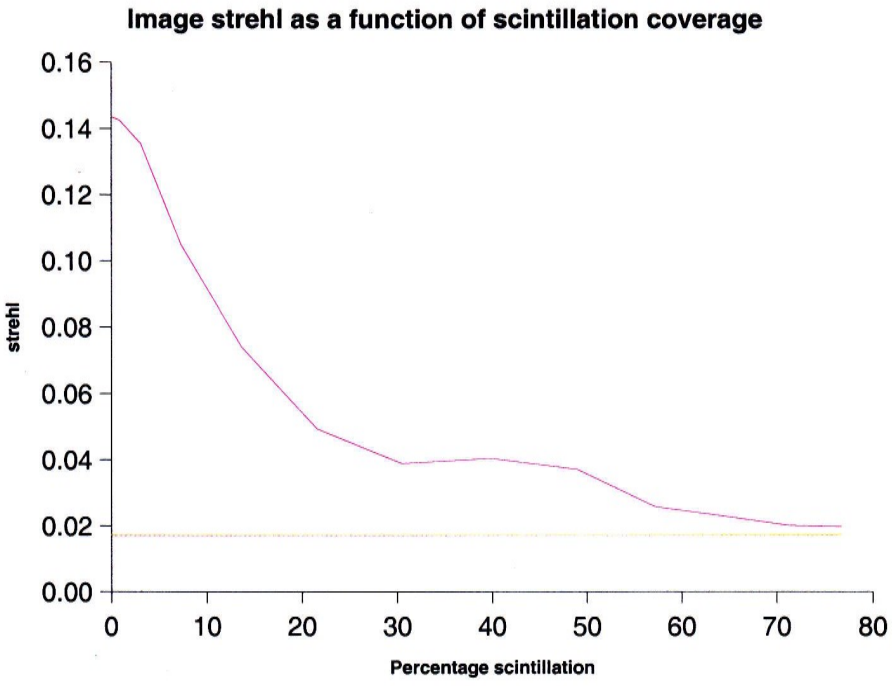
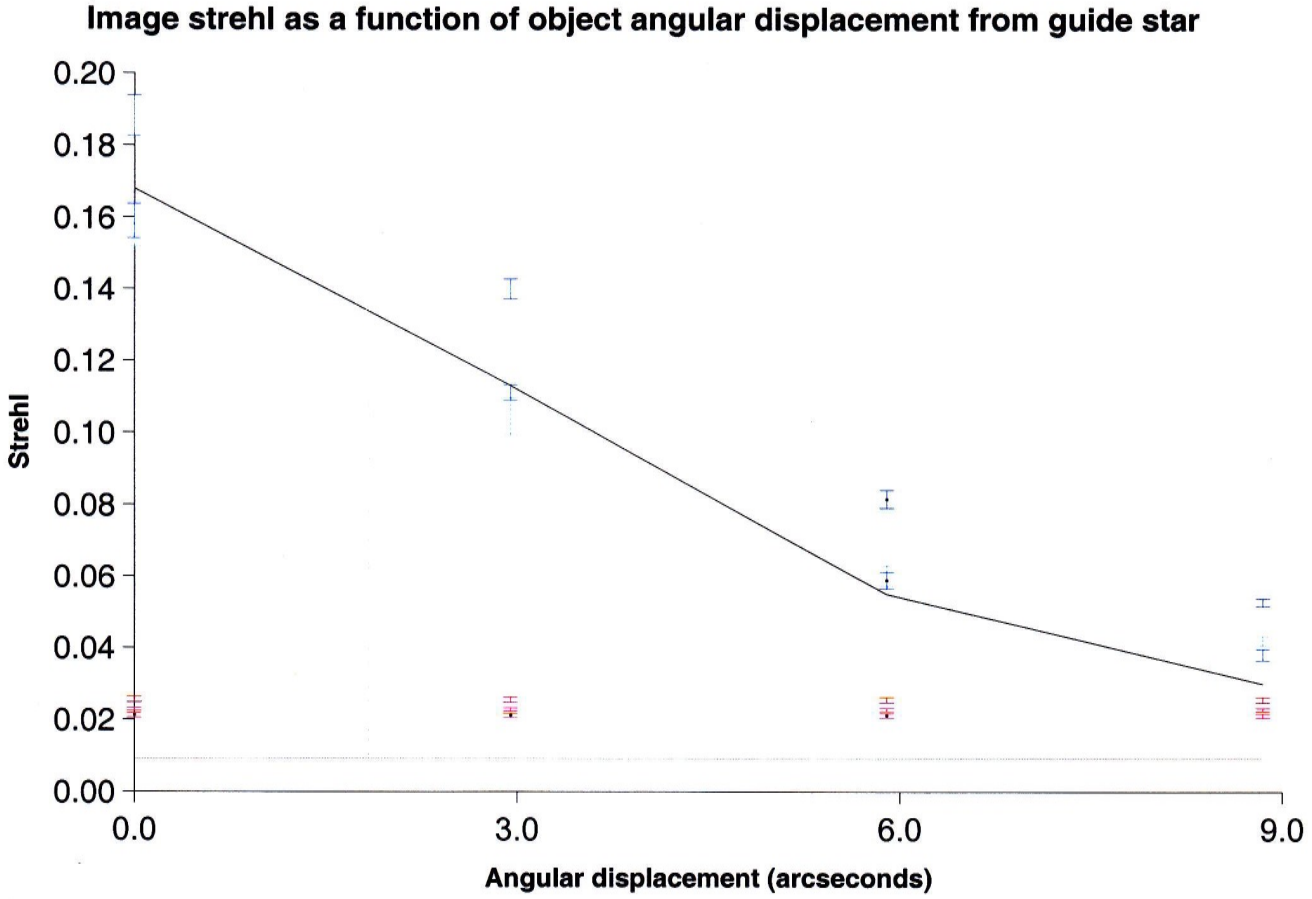


Figure 57
On-axis corrected image Strehl degradation due to scintillation. The fraction of the wavefront that cannot be sensed (scintillation) is shown against the resulting image Strehl (magenta) in comparison the seeing limited Strehl (red).

For a direct comparison with Dr Richard Wilson's model, the image Strehl for various off-axis objects have been measured. The discrete representation errors, the error based on the scatter of 5 realisations (plotted error bars) and the measured Strehls are shown in figure 58.



	256x256 grid covering 3.0x3.0m			256x256 grid covering 4.0x4.0m			512x512 grid covering 4.0x4.0m			Richard Wilsons model
Angle from guide star (arcseconds)	Average corrected strehl	Standard error based on scatter	Discrete representation error	Average corrected strehl	Standard error based on scatter	Discrete representation error	Average corrected strehl	Standard error based on scatter	Discrete representation error	Corrected strehl
0.00	0.1549	0.0027	0.0017	0.1883	0.0043	0.0034	0.1589	0.0035	0.0017	0.168
2.95	0.1043	0.0048	0.0012	0.1400	0.0040	0.0025	0.1111	0.0023	0.0012	0.113
5.90	0.0601	0.0029	0.0007	0.0816	0.0039	0.0015	0.0589	0.0022	0.0006	0.055
8.85	0.0421	0.0012	0.0005	0.0528	0.0029	0.0010	0.0382	0.0013	0.0004	0.030
	Average uncorrected strehl	Standard error based on scatter	Discrete representation error	Average uncorrected strehl	Standard error based on scatter	Discrete representation error	Average uncorrected strehl	Standard error based on scatter	Discrete representation error	Uncorrected strehl
	0.0229	0.0008	0.0003	0.0256	0.0007	0.0005	0.0213	0.0005	0.0002	0.0092

Figure 58

A 1.0 meter telescope with a adaptive optic system functioning as a high pass zernike mode filter of wavefront aberrations giving corrected image Strehls as a function of angular separation between object and guide star. The black and grey lines are predictions given by Dr R. Wilson's statistical covariance model for corrected and uncorrected images. The blue and red coloured data points correspond to corrected and uncorrected images respectively. The wind speed for the wave optics prediction is 30.0m/s and the integration time is 1.0s (with 0.02s resolution). The Strehls are an average of five realisations of the atmospheric turbulence which is then applied to each viewing angle. The error bars are the standard errors based on the scatter of the results. The dashed grey line is the isoplanatic angle for the modelled atmosphere of 1.85arcseconds corresponding to a Strehl that is 83% of the on-axis Strehl

The grid resolution and scale are as follows: turquoise & magenta correspond to a 256x256 grid covering 3.0x3.0m area (0.012m resolution)
dark blue & dark red correspond to a 256x256 grid covering a 4.0x4.0m area (0.016m resolution)
light blue & light red correspond to a 512x512 grid covering a 4.0x4.0m area (0.0078m resolution)

The primary difference between the two simulator results is the uncorrected image Strehl. This is due to the effects of using a single atmospheric turbulence layer described previously. This effect is echoed in the well off-axis regime. The wave

optics prediction for the Strehl is higher than the statistical model. This can be explained because the uncompensated image Strehl is higher causing the corrected image Strehl to decay to a different level as the science object is moved off-axis. The 256x256 grid covering 3.0x3.0m and the 512x512 grid covering 4.0x4.0m provides good agreement with the statistical model. The on-axis Strehl predicted by the wave optics model is slightly lower than the statistical model. It is not clear if this is a statistical fluctuation or degradation due to the 0.2% noise in the wave adaptive optics system. The 256x256 grid covering 4.0x4.0m shows consistently higher Strehls and is suffering from insufficient resolution in resolving the turbulence, as discussed previously.

A typical on-axis point spread function for a 512x512 grid covering 4.0x4.0m for a 1.0m telescope simulation (one of the simulations that is part of figure 58) is shown in figure 59. The principle thing to see is that the full width half maximum (FWHM) for the diffraction limited image and for the corrected image is the same. This would suggest that the adaptive optics system has managed to restore the image resolution to the diffraction limited case. The Strehl however is significantly less than 1. This means that there is a 'skirt' to the corrected image peak in which much of the energy has been scattered. This can be seen in the 0.1*peak intensity contour for the corrected image. The lowest intensity sections of the corrected PSF peak is larger and has a ring around it that does not appear in the diffraction limited image. This ring has a diameter of 0.47 arcseconds. The first diffraction ring in the diffraction limited image has a diameter of 0.33 arcseconds. This would suggest that this ring is not dominated by diffraction through the pupil, suggesting that the corrected image is not diffraction limited.

The seeing limited image is extended over a large area approximately 0.8 arcseconds in diameter. There is no suitable definition for the FWHM if the image is not dominated by a single peak. The appearance of multiple peaks can be interpreted as insufficient sampling of the atmosphere to provide a smooth PSF.

It should be noted that the execution time for 4 corrected field of views on a 256MB 1.4GHz Athlon running Redhat Linux 7.2 is roughly 25 minutes (256x256 grid); a time practical for repeated use.

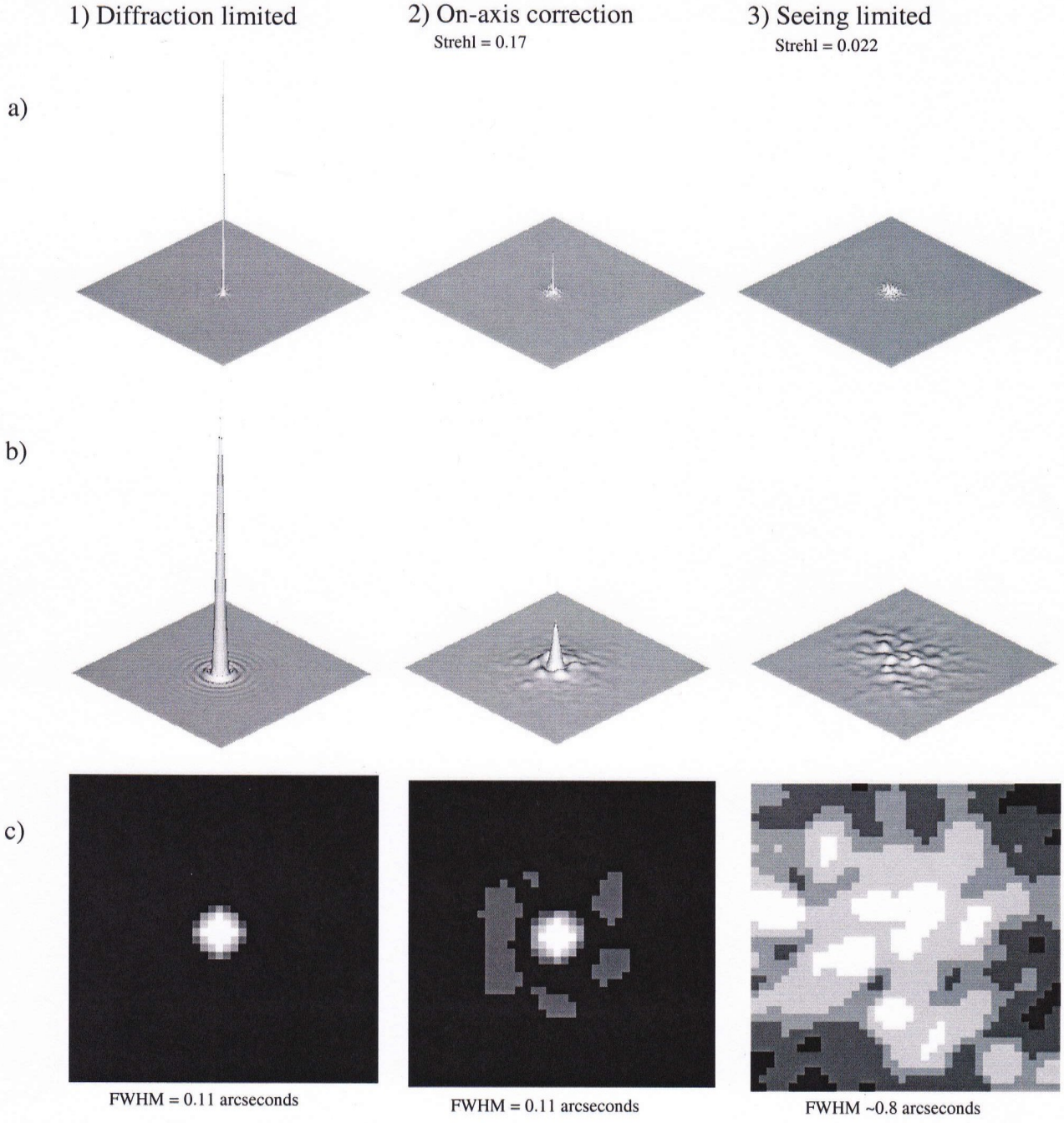


Figure 59

Surface intensity plots and intensity contours for 1) diffraction limited 2) on-axis correction and 3) uncorrected images.
a) A surface intensity plot of the image with a scale of 13.2 arcseconds across
b) A surface intensity plot of the central section of the image with a scale of 2.58 arcseconds across
c) Contours showing the intensity levels relative to the peak intensity with a scale of 1.03 arcseconds across.

Black	< 0.1 times peak intensity
Dark grey	> 0.1 times peak intensity
Mid grey	> 0.2 times peak intensity
Light grey	> 0.3 times peak intensity
White	> 0.5 time peak intensity

11) Discussion

11.1) Simulator properties

11.1.1) Kolmogorov statistics

The atmospheric turbulence representation is periodic and is designed to follow Kolmogorov statistics (see section 7.7). The deviation between the desired statistics and the realised statistics on scales approaching the screen size is due to the periodic nature of the atmosphere representation. This is caused by the induced correlation between the phase distortion on scales approaching the periodicity length generated by a periodic representation. This has a severe effect on the Kolmogorov structure function unless some kind of correction is applied (see figure 26).

The scheme the Author devised is a simple way of providing a selective boost to the structure function on certain scales. As the realised structure function is smaller than the ideal grand ensemble Kolmogorov structure function on a large scale, then there is insufficient distortions at low spatial frequencies. To correct for this a frequency dependant fudge factor was added to the atmospheric distortion fourier component amplitude. This fudge factor contained two fittable parameters that could only be identified via trial and error (see section 7.7).

The justification for the validity of this is that upon inspection the resulting structure function is closer to the desired statistics over a far greater scale range than without any correction. The alternative approach that does not use any fudge factor is to make the atmospheric turbulence screens at least 20 times larger than the pupil. This would place the region of agreement between the realised turbulence and ideal Kolmogorov turbulence up to the scale of the pupil. This is superior in that it does not require empirically found parameters, but it is computationally too expensive for practical use. The spatial resolution must not be reduced (see later for discussion) resulting in much greater memory requirements and processing time (by a factor of approximately 30). Using the fudge factor is therefore an appropriate trade off between speed and elegance.

11.1.2) Resolution and size of atmospheric phase screens

The pupil diameter must be no bigger than the scale on which there is good agreement between the realised and ideal structure functions. If this is not satisfied then there is an under representation of the atmospheric turbulence on the scale of the pupil. This results in a significant increase in the uncorrected image Strehl (see figure 54) as the atmospheric turbulence phase screen size approaches the pupil size. This effect is not significant for the corrected image Strehl as this large scale turbulence is compensated for by the adaptive optics system.

The compensated image Strehl is sensitive to the smaller scale turbulence. This can seen in figure 54 and 58 as the spatial resolution drops to 0.016m (corresponding to a 4.0x4.0m screen represented by a 256x256 grid). There is a significant increase in the corrected Strehl as the smaller scale turbulence is no longer resolved correctly. This effect occurs at this resolution because of the order of the adaptive optics correction. If the adaptive optics system provided higher order corrections

then turbulence on smaller scales would be more significant and the simulation would have to run at higher resolutions for accurate results.

As described in section 9, the zernike wavefront sensors fail at a resolution of 0.020m. This is however a property of the wavefront sensor rather than of the simulation. Shack-Hartmann sensors would not be affected in this way. A Shack-Hartmann sensor (see section 5.3) finds the far field image of each subaperture and measures the central position of each spot to sense the local wavefront gradient. This is not nearly as sensitive to the resolution as it does not involve the phase unwrapping that occurs as part of the zernike wavefront sensors.

11.1.3) Point spread functions (PSF)

The imaging of the system is forced into 2 possibilities; the far field image and finite lens impulse response (see section 7.6). This is due to the absence of any other technique the Author is aware of or has been able to construct (see section 7.5). The available imaging techniques are satisfactory for the current case, however the applicability to a given telescope system must be evaluated on a case by case basis. There is no evidence that the two techniques will ever be insufficient but that is not proof that they will always be suitable.

The far field images have an intensity distribution determined by the fourier mode amplitudes. There is a direct one to one mapping between individual fourier modes and the pixels in the image (see section 7.6). The uncorrected instantaneous images show a collection of smooth shapes with a 'ripple like' quality to them (see figure 47). This would indicate that the turbulent light is dominated by the fourier modes around many spatial frequencies (in the plane of the grid, i.e. perpendicular to the direction of propagation of the beam as a whole). That the transition from 'blob' to 'blob' in the image is smooth indicates that the fourier mode amplitude distribution is also smooth. This is in contrast to the 'corrected' instantaneous images (see figure 47) which appears to have multiple sharp peaks. The sharpness of the peaks indicates that the 'corrected' light is dominated by fourier modes that have a narrow spread of frequencies. The adaptive optics system has removed some distortion, reducing the spread of spatial frequencies in the 'corrected' light. This is a description of 'sharpening up the image' with the resulting time integrated images reflecting this.

The corrected PSF have a smooth central peak surrounded by an uneven ring (see figure 58). The presence of a ring in a corrected image has sometimes been hailed as indicating that an image with diffraction limited resolution has been produced. The results of the test system (see section 9) do not support this. The ring in the corrected image is 0.47 arcseconds in diameter while the first diffraction ring in the diffraction limited image is 0.33 arcseconds in diameter. As these two diameters are significantly different then pupil diffraction cannot be dominating the image. As the corrected image ring is larger than the pupil diffraction ring it corresponds to a feature that is smaller than the pupil. This can in fact be thought of as the 'corrected seeing' of the remaining distortions in the light. If more of the distortions are removed then the 'corrected seeing' is improved and the ring size would approach the diffraction ring size.

The uncorrected PSF (time integrated) is not smooth as it would be expected to be if it were a grand ensemble average.

It contains multiple peaks distributed in a region almost an order of magnitude larger than the diffraction limited FWHM (see figure 59). This indicates that insufficient atmosphere 'area' hass been sampled to obtain the grand ensemble average. To achieve this with one atmospheric turbulence screen is unrealistic as the size of the screen required would be far too large and require too much computational resources. The turbulence limited image Strehl for a single atmospheric turbulence layer is therefore an overestimate. Note that the requirements for a realistic turbulence limited image are more strict than for a 'corrected' image as the lowest spatial frequency turbulence distortions dominate the uncorrected PSF. The large turbulence screen requirement is then required to increase the number of these large turbulence components into the atmospheric layer.

11.1.4) Atmospheric turbulence screens

The atmospheric turbulence screen is a realisation of the atmosphere statistics where the only differences between the realisations are the origins of each fourier mode (i.e. each fourier mode has a translational freedom within the turbulence screen that is fixed for a given realisation). The wind motion translates the turbulence my moving all the origins uniformly so that the section of atmosphere being sampled by the light is changed. The distortions produced in light passing through the turbulence can vary remarkably across the phase screen (see figure 51). If very small wind speeds are used so that only a small section of the turbulence is sampled then the resulting image Strehl can vary remarkably from a long term average. This is due to 'local' regions of the turbulence that do not reflect the atmosphere as a whole. The atmosphere has been under sampled in this case. For a realistic 30m/s wind speed, the turbulence layers are sufficiently well sampled to prevent this, but there is still significant variations between turbulence realisations (see figure 52). This results in an error estimate based on the spread of measured Strehls to be larger than the discrete representation error.

The problems of atmospheric sampling could be dramatically improved by using multiple atmospheric turbulence layers, each moving with a different wind velocity. By using multiple layers (6+ is a realistic number) the dependance on a single layer is removed allowing robust result from a single simulation run. As the layers are moving with different velocity, the spread of possible turbulence sampled along a given line of sight is dramatically increased. This means that the atmosphere being modelled is much closer to the grand ensemble desired. Any realistic simulation should make use of this to ensure that that statistical effects over single turbulence layer realisations do not dominate.

The simulator is able to demonstrate scintillation which is not present in statistical models. The effects can be seen in figure 57 as a decay of the corrected image Strehl down to the uncorrected image Strehl level as scintillation becomes extreme. This is a diffraction effect of light as it passes through the atmosphere and is simulated by wave optics. This is a very important effect for the specifications of adaptive optics systems and can now be simulated using this simulator.

In the test cases a natural guide star is used from which the wavefronts arrive as a plane wave and pass through all the atmosphere. Angular anisoplanatism is shown in figure 58 where the corrected image Strehl for off axis science objects is reduced as the science beam and guide star beam follow different paths. Laser guide stars could also be simulated. They would be an expanding spherical wavefront (see section 7.2) that originates at a finite altitude with a finite size. The light beam would

be expanding slowly (well within the beam speed restriction, see section 7.3) as it propagates down to the telescope. This would be simulating focal anisoplanatism as the turbulence would be sampled differently to the science beam. The notable consideration with this is that the laser light would need to be propagated both up and down the atmosphere in order to simulate the consequences of the laser light propagating upwards through the atmosphere (see section 5.2).

11.2) Comparison with statistical model

The adaptive optics system described in section 9 has been used in order to be comparable to the statistical simulations performed by Dr Richard Wilson. The propagation distances within the telescope have been set to zero in order to remove any propagation effects between the pupil, wavefront sensor and deformable mirror. Figure 58 shows the statistical model predictions alongside the wave optics predictions. There are some differences between the model predictions but these are either identified as due to effects discussed above or are on the level of the wave optics error bars. The corrected image FWHM is the same as a diffraction limited image (0.11 arcseconds), but there is significant amounts of energy as noise about the peak of the PSF as suggested by a Strehl of 0.16. The Rayleigh criterion gives a resolution of 0.13 arcseconds. This is a different quantity than the FWHM, but it does confirm that the images are of the correct scale. The seeing limited image has an approximate FWHM of 0.8 arcseconds while the Fried criterion gives a resolution of 1.3 arcseconds. This is a weaker criterion and is only asserted by analogy using the Fried parameter as the effective telescope diameter. As it is only approximate, the greater fractional difference between the wave optics PSF and the Fried resolution criterion reflects this. The definition of a diffraction limited image is clearly dependant on the telescope use and requires some clarification in order to convey useful information.

The good agreement between the two models that operate in very different ways indicates that the effects being simulated are real and that the wave optics simulation is making physically relevant predications. The test situation was constructed with considerable effort to ensure that the modelled situations were comparable. The zernike wavefront sensor and deformable mirrors were created solely for this purpose. The control matrix used is then simply the negative identity matrix.

The turbulence used has a Fried parameter of 0.1m. This corresponds to a refractive index structure constant of 6.95×10^{-13}. The critical time constant is then 0.00105s and the isoplanatic angle 1.85 arcseconds (see section 4.2). The time step used in the simulations is 0.02s showing that there has been significant atmosphere evolution between instantaneous images and hence each instantaneous images is a significant addition to the PSF. The isoplanatic angle of 1.85 arcseconds can be seen to correspond to a region on the shy where science objects have a corrected image Strehl at least 83% of the on-axis image Strehl (using figure 58).

12) Conclusions

A wave optics simulator produces Fresnel propagation allowing diffraction effects to be investigated. It improves upon the normal incidence assumption common in many simulations by considering the projected shape of optical components. Optical components are given 6 degrees of freedom (3 translational and 3 rotational) and may interact with the optical field in an arbitrary way or as a transmission mask. The errors involved in a discrete representation of an optical field along with representational effects such as wrapping and higher order projection effects are collected as the simulation proceeds. The optical and electronic component library can be extended to include any components desired for a telescope design. Kolmogorov turbulence has been realised to provide atmospheric distortions though it could be easily replaced with any other statistics the user desired. Complex electronic systems can be included with realistic processing time delays.

The simulation proceeds at at rate on common PCs that is practical for everyday use with memory and computation time that scale as n^2 and $n^2\ln(n)$ respectively where the optical representation is on n^2 grid points. The simulator has been verified as working correctly by comparison with a statistical model devised by Dr R.W. Wilson. It correctly predicts the 'corrected' image Strehls for science objects up to 9 arcseconds away from a guide star for turbulence dominated at a single layer 3.5km away (along the line of sight). Scintillation produced by the atmosphere can be observed and its effects measured. Point spread function predictions for a telescope design can be constructed. The presence of a ring about the PSF peak is not sufficient evidence of diffraction limited imaging and should not be used as such without careful consideration. The isoplanatic angle corresponds to a patch of sky that has a 'corrected' image Strehl that is at least 83% of the on-axis 'corrected' image Strehl

The atmosphere should be modelled as containing multiple turbulence layers to provide an accurate description of the atmosphere. The required resolution for the atmospheric turbulence to be resolved can be found by steadily increasing the resolution until the corrected image Strehl stabilises at a constant value. The size of the atmospheric turbulence screens for accurate atmosphere portrayal can be found by increasing the screen size until the uncorrected image Strehl stabilises at a constant value. Natural or laser guide stars can be included to drive the modelled adaptive optics system. The integration time required to achieve a robust Strehl is shorter for a corrected PSF than for an uncorrected PSF. The Strehl must be measured for a range of integration times to deduce a suitable time to produce robust results.

13) References

[1] "Optical and Near-Infrared Astronomical Instrumentation". Astronomy postgraduate course by Richard Myers, Durham University.

[2] "Introduction to Fourier Optics" by J.W. Goodman, published by McGraw-Hill.

[3] "A short course in adaptive optics" May 11 to 13 1998, Blackett Laboratory, Prince Consort Road, London SW7 2BZ

[4] "Atmospheric-Turbulence Compensation Experiments Using Co-operative Beacons" by D.V. Murphy, The Lincoln Laboratory Journal 1992, vol. 5 no. 1, PG 25 - 92

[5] "Adaptive Optics for Astronomy" by R.R. Parenti, The Lincoln Laboratory Journal 1992, vol. 5 no.1, PG 93-114

[6] "The SWAT Wavefront Sensor" by H.T. Barclay et al, The Lincoln Laboratory Journal 1992, vol. 5 no. 1, PG 115-130

[7] "A Wave Optics Propagation Code for Multi-Conjugate Adaptive Optics" by B.L. Ellerbroek, Pre-print "http://lenin.pd.astro.it/venice2001/proceedings/index.html".

Private correspondence 2001.

[8] "Zernike polynomials and atmospheric turbulence" by Robert J. Noll, JOSA 1976, vol. 66, no. 3, PG 207-211

14) Acknowledgements

The support of the Astronomical Instrumentation group at Durham University permitted this project to be undertaken. The suggestions made by Richard Myers, Richard Wilson, Nirmal Bissonauth and Chris Saunters were invaluable to the progress of this project. The author would like to express his gratitude to Andy Vick for the correspondence regarding unevenly sampled fast fourier transforms and wavelet representations and to Brent Ellerbroek for his correspondence regarding the converging co-ordinate system described in section 7.5.1).

15) Appendices

15.1) Program listings

'simaxis.py'

```
"""
from simdata import *
from simerrors import *

class newaxis:
   def __init__(self, sim=None, name="", length=0.0, inp=""):
      "Add new axis of length <length> called <name> to simulation <sim>, getting input from beam splitter <inp>"
      self.simobj = sim.simobj                                                    #Recording simulation object
      if (type(name) != type("")) or (type(inp) != type("")):
         raise simerror, simerror('<axis> name, inp must a string '+elapsedtime()+'\n')
      if name=="" or inp=="":
         raise simerror, simerror('<axis> "" is not a valid name for a simulation element '+elapsedtime()+'\n')
      names = allnames(self.simobj)
      if names.count(name) != 0:
         raise simerror, simerror("<axis> Name '"+name+"' already exists "+elapsedtime()+'\n')
      if self.simobj.splitters.has_key(inp) != 1:
         raise simerror, simerror("<axis> Beam splitter '"+inp+"' does not exist "+elapsedtime()+'\n')
      self.l = length
      self.simobj.axes[name] = self                                               #Adding axis to simulation
      obj = self.simobj.splitters[inp]                                            #Finding input splitter
      self.input = obj.output                                                     #Obtaining input mesh
      self.components = []                                                        #Creating empty list of components in axis
"""
```

'simdata.py'

```
"""
#Large number
infinity = 1e1000000
#Small number
tiny = 1e-10
#Infinitesial number
infinitesimal = 1e-100
import cPickle
from vector import *
from copy import *
from Numeric import *
from FFT import *
from simerrors import *
from simphase import *

#
#Mapping base classses
#
class newplane:
   def __init__(self, x_off=0.0, y_off=0.0, x_ang=0.0, y_ang=0.0, z_rot=0.0, plane=None):          #Select orientation of plane
      if plane == None:
         self.x_pos = x_off
         self.y_pos = y_off
         self.x_angle = x_ang
         self.y_angle = y_ang
         self.z_rotation = z_rot
      else:
         self.x_pos = plane.x_pos
         self.y_pos = plane.y_pos
         self.x_angle = plane.x_angle
         self.y_angle = plane.y_angle
         self.z_rotation = plane.z_rotation



class newscreenfunction(newplane):
   def __init__(self, x_off=0.0, y_off=0.0, x_ang=0.0, y_ang=0.0, z_rot=0.0, plane=None, iso='yes'):
      newplane.__init__(self, x_off, y_off, x_ang, y_ang, z_rot, plane)
      self.iso = iso
   def function(self, map, lambd):                                                #Very dull function - default
      screen = ones(map.shape)
      return screen.astype(map.typecode())                                        #It is essential that element type and array shape is the
same as for map
```

```
#
#Some default objects
#
plane = newplane()
unitscreenfunction = newscreenfunction()


#
#Optical field and signal definitions
#
class newopticalfield(newplane):
   def __init__(self, x=0, y=0, z=0, x_ang=0, y_ang=0, wave=5e-7, l=1, res=2**8, sfunc=None, mask=0.0):                    #Must not have a reference
to simobject as it would then be copied during deepcopy
      newplane.__init__(self, x, y, x_ang, y_ang, 0.0,sfunc)
      self.masklen = mask
      self.wavelength = wave
      self.scale = l
      self.z_pos = z
      self.n = res
      self.relerr = 0
      if sfunc == None:
         self.mesh = None
         self.space = 'none'
      else:
         self.mesh = sfunc.function(self.projection(sfunc),self.wavelength)
         self.space = 'real'
   def isolate(self):                                                    #perform isolation masking and error approximation
      if self.space == "frequency":
         self.mesh = inverse_fft2d(self.mesh)
         self.space = "real"
      elif self.space != "real":
         raise simerror, simerror("<ofld> Attempt to sample a non-existant mesh "+elapsedtime()+'\n')
      temp = abs(self.mesh)
      tot = sum(sum(abs(temp)**2.0))+infinitesimal
      n = self.n
      wraperr = (sum(abs(temp[0:n,0])**2.0)+sum(abs(temp[(n-1),1:(n-1)])**2.0)+sum(abs(temp[0:n,(n-1)])**2.0)+sum(abs(temp[0,1:(n-
1)])**2.0))*float(n**2)/float(4*n-4) #Total wrapping error over mesh
      m = int(float(n)*self.masklen/self.scale)
      blankerr = float(n**2)*(tot-sum(sum(abs(temp[m:(n-m),m:(n-m)])**2.0)))*float(n**2)/float((n-2*m)**2)                    #Total masking error over
mesh
      self.relerr = sqrt(self.relerr**2.0+(wraperr**2+blankerr**2)/(tot**2))                                    #Adding relative error in quadrature as
independant
      self.mesh[0:m,:] = 0.0                                                #Perform masking
      self.mesh[(n-m):n,:] = 0.0
      self.mesh[:,0:m] = 0.0
      self.mesh[:,(n-m):n] = 0.0
   def propagate(self, z):
      dist = z - self.z_pos
      newopticalfield.pc = -1.0j*pi*self.wavelength*dist/(self.scale**2)                                    #Propagation constant
      newopticalfield.off = self.n/2 - 1
      def prop1(x,y):
         return exp(newopticalfield.pc*(x**2+y**2))
      def prop2(x,y):
         return exp(newopticalfield.pc*((newopticalfield.off-x)**2+y**2))
      def prop3(x,y):
         return exp(newopticalfield.pc*(x**2+(newopticalfield.off-y)**2))
      def prop4(x,y):
         return exp(newopticalfield.pc*((newopticalfield.off-x)**2+(newopticalfield.off-y)**2))
      p = zeros((self.n,self.n), Complex)
      p[0:self.n/2+1,0:self.n/2+1] = fromfunction(prop1,(self.n/2+1,self.n/2+1))
      p[self.n/2+1:self.n,0:self.n/2+1] = fromfunction(prop2,(self.n/2-1,self.n/2+1))
      p[0:self.n/2+1,self.n/2+1:self.n] = fromfunction(prop3,(self.n/2+1,self.n/2-1))
      p[self.n/2+1:self.n,self.n/2+1:self.n] = fromfunction(prop4,(self.n/2-1,self.n/2-1))
      if self.space == "real":
         self.mesh = fft2d(self.mesh)
         self.space = "frequency"
      elif self.space != "frequency":
         raise simerror, simerror("<ofld> Attempt to propogate a non-existant mesh "+elapsedtime()+'\n')
      self.mesh = self.mesh * p
      self.z_pos = z
      self.x_pos += tan(self.x_angle) * dist
      self.y_pos += tan(self.y_angle) * dist
   def projection(self, p, mag = 1.0):                                    #Produce projection map: x,y co-ordinates is screen plane
of where mesh would be projected to (with strides magnified by mag)
      if self.n <= 1:
         raise simerror, simerror("<ofld> Attempt to make a projection map of an undefined mesh resolution "+elapsedtime()+'\n')
      planecs = cs(p.x_angle,p.y_angle,p.z_rotation)
      meshcs = cs(self.x_angle,self.y_angle,0.0)
      offset = array([self.x_pos,self.y_pos,0.0],Float) - array([p.x_pos,p.y_pos,0.0],Float)                    #ofield & plane at same z
      offset = offset - meshcs[2]*(dp(offset,planecs[2])/dp(meshcs[2],planecs[2]))                    #Put mesh origin projection into plane -
produces the offset in global coordinates
```

```
        offset = convert(offset,planecs)                                    #Offset is now in plane coordinates
        if abs(offset[2])>tiny:                                             #Check offset is in plane
            raise simerror, simerror('<ofld> Attempt to project origin into a plane failed '+elapsedtime()+'\n')
        x_stride = meshcs[0]
        x_stride = x_stride - meshcs[2]*(dp(x_stride,planecs[2])/dp(meshcs[2],planecs[2]))      #Put mesh x unit vector projection into
plane
        x_stride = convert(x_stride,planecs)                                #convert to plane coordinates
        if abs(x_stride[2])>tiny:                                           #Check x_stride is in plane
            raise simerror, simerror('<ofld> Attempt to project mesh x unit vector into a plane failed '+elapsedtime()+'\n')
        y_stride = meshcs[1]
        y_stride = y_stride - meshcs[2]*(dp(y_stride,planecs[2])/dp(meshcs[2],planecs[2]))      #Put mesh y unit vector projection into
plane
        y_stride = convert(y_stride,planecs)                                #convert to plane coordinates
        if abs(y_stride[2])>tiny:                                           #Check y_stride is in plane
            raise simerror, simerror('<ofld> Attempt to project mesh y unit vector into a plane failed '+elapsedtime()+'\n')
        newopticalfield.mid = float(self.n-1)/2.0
        newopticalfield.xs = x_stride / mag                                 #magnified view of plane (mag = 1.0 normally)
        newopticalfield.ys = y_stride / mag                                 #magnified view of plane (mag = 1.0 normally)
        def project(a,b):
            x = (a-newopticalfield.mid)*newopticalfield.xs[0] + (b-newopticalfield.mid)*newopticalfield.ys[0]
            y = (a-newopticalfield.mid)*newopticalfield.xs[1] + (b-newopticalfield.mid)*newopticalfield.ys[1]
            return x + y*1.0j
        dat = self.scale*fromfunction(project, (self.n, self.n))/float(self.n) + (offset[0] + 1.0j*offset[1])      #Projection map in physical units
        if dat.typecode() != Complex:
            dat.astype(Complex)
        return dat
    def applyphasescreen(self, func = newscreenfunction(), mag = 1.0):     #Apply a phase screen (defined by func) to
mesh with co-ordinate magnification mag
        if self.space == "frequency":
            self.mesh = inverse_fft2d(self.mesh)
            self.space = "real"
        elif self.space != "real":
            raise simerror, simerror("<ofld> Attempt to sample a non-existant mesh "+elapsedtime()+'\n')
        if func.iso=='yes':
            self.isolate()                                                  #Ensure beam is isolated if desired
        self.mesh = self.mesh * func.function(self.projection(func, mag = mag), self.wavelength)      #Find mesh projection onto function
func and multiplicatively apply phase screen defined by func to mesh
        temp = deepcopy(self.mesh)                                          #Perform error analysis
        theta = arccos(dp(cs(self.x_angle,self.y_angle,0.0)[2],cs(func.x_angle,func.y_angle,func.z_rotation)[2]))
        self.propagate(sqrt(2)*self.scale*tan(theta))
        if self.space == "frequency":
            self.mesh = inverse_fft2d(self.mesh)
            self.space = "real"
        elif self.space != "real":
            raise simerror, simerror("<ofld> Attempt to sample a non-existant mesh "+elapsedtime()+'\n')
        temp = abs(temp)**2.0 - abs(self.mesh)**2.0
        tot = sum(sum(abs(self.mesh)**2.0))+infinitesimal
        self.relerr = sqrt(self.relerr**2.0 + (sum(sum(temp))/tot)**2.0)    #Adding relative error in quadrature
    def getrelerr(self):
        n = self.n
        if self.space == "frequency":
            m = self.mesh
            freqerr = 0.5*(sum(abs(m[n/2+1,:]))+sum(abs(m[:,n/2+1]))-abs(m[n/2+1,n/2+1]))      #Summing intensity produced by
unrepresentable modes
            self.mesh = inverse_fft2d(self.mesh)
            self.space = "real"
            tot = sum(sum(abs(self.mesh)**2.0))+infinitesimal
        elif self.space != "real":
            raise simerror, simerror("<ofld> Attempt to sample a non-existant mesh "+elapsedtime()+'\n')
        if self.space == "real":
            tot = sum(sum(abs(self.mesh)**2.0))+infinitesimal
            self.mesh = fft2d(self.mesh)
            self.space = "frequency"
            m = self.mesh
            freqerr = 0.5*(sum(abs(m[n/2+1,:]))+sum(abs(m[:,n/2+1]))-abs(m[n/2+1,n/2+1]))      #Summing intensity produced by
unrepresentable modes
        elif self.space != "frequency":
            raise simerror, simerror("<ofld> Attempt to sample a non-existant mesh "+elapsedtime()+'\n')
        return sqrt(self.relerr**2.0+(freqerr/tot)**2.0)


class newoptdata:                                                           #Optical data handler
    def __init__(self):
        self.total_ref = 0
        self.current_ref = 0
        self.data = None
    def add_ref(self):                                                      #Register a reference to stored data
        self.total_ref += 1
    def reset(self):                                                        #Reset handler
        self.empty()
        self.total_ref = 0
```

```
  def retrieve(self):                                          #Return stored data, decrementing the reference count
    if self.current_ref <= 0:
      raise simerror, simerror("<odat> Data requested more times than reference count permits "+elapsedtime()+'\n')
    else:
      self.current_ref -= 1
      if self.current_ref>0:
        temp = deepcopy(self.data)
      else:
        temp = self.data
        del self.data
      return temp
  def store(self, dat):                                        #Set stored data
    if self.current_ref != 0:
      raise simerror, simerror("<odat> Attempt to store data when reference count indicates existing data still needed "+elapsedtime()+'\n')
    else:
      if self.total_ref > 0:
        self.current_ref = self.total_ref
        self.data = dat
  def empty(self):                                             #Empty stored data, regardless of existing references
    if self.current_ref > 0:
      self.current_ref = 0
      del self.data



class newsignal:                                               #Wrapper for signal data - must not have reference to sim object
as it would be copied during deepcopy
  def __init__(self, tvalid = infinity, a = 'yes', dat = None):
    self.time_valid = tvalid
    self.accessed = a
    self.data = dat



class newelecdata:                                             #Electronic data handler
  def __init__(self):
    self.total_ref = 0
    self.signals = []
    self.current_ref = []
  def add_ref(self):                                           #Register a reference to stored data
    self.total_ref += 1
  def reset(self):                                            #Reset handler
    self.empty()
    self.total_ref = 0
  def opensignalline(self):                                    #Reset signals accessability for retrieves
    for i in range(len(self.signals)):
      s = self.signals[i]
      if s != None:
        s.accessed = 'no'
  def retrieve(self,time):                                     #Return earliest unaccessed signal valid before or at the current
time
    mint = infinity
    mini = None
    for i in range(len(self.signals)):
      s = self.signals[i]
      if s != None:
        tvalid = s.time_valid
        if (tvalid <= time) and (tvalid < mint) and (s.accessed == 'no'):
          mint = tvalid
          mini = i
    if mini == None:
      sig = None
    else:
      self.signals[mini].accessed = 'yes'
      self.current_ref[mini] -= 1
      if self.current_ref[mini] <= 0:
        sig = self.signals[mini]
        self.signals[mini] = None
      else:
        sig = deepcopy(self.signals[mini])
    return sig
  def store(self, tvalid, dat):                                #Add a signal
    if dat!=None:                                             #Signal cannot be None
      if self.signals.count(None) == 0:                            #note: this routine cannot spot if a signal has been
mistakenly stored multiple times.
        self.signals.append(newsignal(tvalid, 'yes', dat))
        self.current_ref.append(self.total_ref)
      else:
        i = self.signals.index(None)
        self.signals[i] = newsignal(tvalid, 'yes', dat)
        self.current_ref[i] = self.total_ref
#   def exhausted(self,time):                                  #Designed to check all valid signals have been exhausted -
```

```
i.e. accessed the expected number of times
#       for i in range(len(self.signals)):
#           s = self.signals[i]
#           if s != None:
#               if (s.time_valid <= time):
#                   raise simerror, simerror("<edat> Valid signal persisting beyond current time_step "+elapsedtime()+'\n')
    def empty(self):                                                    #Empty stored data, regardless of existing references
        self.signals = []
        self.current_ref = []



#
#Component base class
#
class simcomponent:                                                     #To ensure all components keep track of the simulation
object they belong to
    def __init__(self, component=None, name=""):
        if component==None:
            raise simerror,simerror("<scom> Attempting to add simulation component without simulation object "+elapsedtime()+"\n")
        self.simobj = component.simobj
        names = allnames(self.simobj)
        if (type(name) != type("")):
            raise simerror, simerror('<scom> Name must a string '+elapsedtime()+'\n')
        if names.count(name) != 0:
            raise simerror, simerror("<scom> Name '"+name+"' already exists "+elapsedtime()+'\n')
        if name=="":
            raise simerror, simerror('<scom> "" is not a valid name for a simulation element '+elapsedtime()+'\n')
        self.name = name
    def startstep(self):                                                #Do something at start of step
        return None
    def endstep(self):                                                  #Do something at end of step
        return None



#
#Component name functions
#
def allnames(sim):
    return sim.splitters.keys() + sim.axes.keys() + sim.optical_comp.keys() + sim.electronic_comp.keys()

def allcomp(sim):
    comp = {}
    comp.update(sim.splitters)
    comp.update(sim.optical_comp)
    comp.update(sim.electronic_comp)
    return comp

#
#Useful functions
#
def record(file,x):
    f = open(file,'w')
    p = cPickle.Pickler(f)
    p.dump(x)
    f.close()
"""
```

'simelec.py'

```
"""
from simdata import *
import os



class electronicin:
    def __init__(self, sim, elecinput=[]):
        if type(elecinput)!=type([]):
            raise simerror, simerror("<elin> Electronic input must be a list "+elapsedtime()+'\n')
        self.linein=[]
        for i in elecinput:
            if type(i)!=type(""):
                raise simerror, simerror("<elin> Electronic inputs must be names of components "+elapsedtime()+'\n')
            comps = allcomp(sim)
            if comps.has_key(i)!=1:
                raise simerror, simerror("<elin> Electronic input '"+i+"' does not exist "+elapsedtime()+'\n')
            inp = comps[i].lineout
            inp.add_ref()
            self.linein.append(inp)
```

```
  def elecinput(self, elecinput=[]):                                    #Only call for a class also descended from simcomponent -
would cause an error otherwise (no simobj member data)
    for i in elecinput:
      if type(i)!=type(""):
        raise simerror, simerror("<elin> Electronic inputs must be names of components "+elapsedtime()+'\n')
      comps = allcomp(sim)
      if comps.has_key(i)!=1:
        raise simerror, simerror("<elin> Electronic input '"+i+"' does not exist "+elapsedtime()+'\n')
      inp = comps[i].lineout
      inp.addref()
      self.linein.append(inp)
  def process(self):                                                    #Only call for a class also descended from simcomponent -
would cause an error otherwise (no simobj member data)
    for i in self.linein:
      i.opensignalline()
    time = self.simobj.config['time']                                   #Get the time
    data = len(self.linein)*[0]
    signals = len(data)*[None]
    for i in range(len(data)):                                          #Populate signals with earliest signal on each line
      signals[i] = self.linein[i].retrieve(time)
      if signals[i]!=None:
        data[i]=1
    while sum(data)!=0:                                                 #While all input signals not exhausted
      mint = infinity
      mini = None
      for i in range(len(data)):                                        #Find earliest signal amongst inputs
        if data[i]==1:                                                  #Considering signal lines with signals
          if signals[i].time_valid < mint:
            mini = i
            mint = signals[i].time_valid
      if mini == None:
        raise simerror, simerror("<elin> Electronic signals present, but cannot find earliest for component "+self.name+" "+elapsedtime()+'\n')
      self.compute(signals[mini])                                       #Process earliest signal before current time
      data[mini]=0
      signals[mini] = self.linein[mini].retrieve(time)                  #Replace processed signal with any present
unprocessed signal from that line
      if signals[mini]!=None:
        data[mini]=1
  def compute(self,sig):                                                #Override compute to do something with signals
    return None


class electronicout:                                                    #Call self.lineout.store(time_valid,data) for each signal you
wish to send
  def __init__(self, sim):
    self.lineout = newelecdata()
    sim.elec_data.append(self.lineout)


class eleccomponent(simcomponent,electronicout,electronicin):
  def __init__(self, sim=None, name="", elecinput = []):
    simcomponent.__init__(self,sim,name)
    electronicout.__init__(self,sim)
    electronicin.__init__(self,sim,elecinput)
    self.simobj.electronic_comp[name] = self
    self.simobj.electronic_list.append(self)
  def apply(self):
    self.startstep()                                                    #Perform any calculations for timestep before processing
    self.process()                                                      #Process inputs
    self.endstep()                                                      #Perform any calculations for timestep after processing, call
self.lineout.store(time_valid,data) for each signal you wish to send


class elecsave(eleccomponent):
  def __init__(self, sim=None, name="", elecinput=[]):
    eleccomponent.__init__(self, sim, name, elecinput)
    os.makedirs(self.simobj.direct()+self.name)
  def startstep(self):
    self.signals = []
  def compute(self,sig):
    self.signals.append(sig)
  def endstep(self):
    logfile.write('<save> '+self.name+' saving signals '+elapsedtime()+'\n')                    #Record activity
    record((self.simobj.direct() + self.name + '/'+str(self.simobj.config['time'])),self.signals)
"""
```

'simerrors.py'

```
"""
from simlog import *

#Errors
class msgerror:
   def __init__(self,t="An error occurred"):
      self.txt = t
      logfile.write(t)
   def msg(self):
      print self.txt

class simerror(msgerror):
   def __init__(self,t="A fatal simulation error occurred"):
      msgerror.__init__(self,t)

#Electrical component signal exception
class nosignal:
   def __init__(self,t="No signal"):
      self.txt = t
"""
```

'simlog.py'

```
""""
#Log file and timing
from time import *
starttime = time()
def timetxt(t = None):
   if t == None:
      t = time()
   return strftime('%H:%M:%S %d/%m/%Y',localtime(t))
def elapsedtime():
   return '['+str(int(time()-starttime))+'s]'                                    #Note simulation not designed to run for more than 68
years (int would roll over). If you are worried about this I suggest you need more powerful computers!
logfile = open('log.txt','w',1)
"""
```

'simopt.py'

```
"""
from simdata import *
from simlog import *
from simerrors import *
from simelec import *
from simphase import *
from Numeric import *
from RandomArray import *
import os


class optcomponent(simcomponent):
   def __init__(self,sim=None, name="", axis="", z=0.0):
      simcomponent.__init__(self,sim,name)                                       #Initialise simcomponent
      if (type(axis) != type("")):                                              #Check axis optcomponent to be placed into is a string
         raise simerror, simerror('<ocom> Axis name must a string '+elapsedtime()+'\n')
      if self.simobj.axes.has_key(axis) != 1:                                    #Check axis exists
         raise simerror, simerror("<ocom> Axis '"+axis+"' does not exist "+elapsedtime()+'\n')
      axis = self.simobj.axes[axis]                                             #Get reference to axis
      if len(axis.components)>0:                                                #If axis contains components, input is output of last
component
         self.input = axis.components[-1].output
      else:                                                                     #If axis has no components, input is input to axis
         self.input = axis.input
      axis.components.append(self)
      self.input.add_ref()                                                      #Registering need for data
      self.output = newoptdata()                                                #Creating output optical data handler
      self.simobj.opt_data.append(self.output)                                  #Putting output optical data handler into simulation
opt_data list
      self.simobj.optical_comp[name] = self                                     #Add to optical components
      self.simobj.optical_list.append(self)                                     #Add to optical execution list
      self.z_pos = z
   def apply(self):
      self.startstep()                                                          #Do whatever needed at start of step
      beam = self.input.retrieve()                                              #Get input data
      for ofield in beam:
         ofield.propagate(self.z_pos)                                           #Propagate to component
      self.action(beam)                                                         #Perform any action to optical field
      self.output.store(beam)                                                   #Store output data
```

```
    self.endstep()                                                    #Do whatever needed at end of step
  def action(self,beam):
    return None                                                       #Base class does nothing


class optscreen(optcomponent,newscreenfunction):
  def __init__(self,sim=None,name="",axis="",z=0.0, x=0.0, y=0.0, x_ang=0.0, y_ang=0.0, z_rot=0.0, plane=None,iso='yes'):
    optcomponent.__init__(self,sim,name,axis,z)
    newscreenfunction.__init__(self, x, y, x_ang, y_ang, z_rot, plane, iso)
  def action(self,beam):
    logfile.write('<osrn> '+self.name+' applied to beam '+elapsedtime()+'\n')
    for ofield in beam:
      ofield.applyphasescreen(self)


class optsave(optcomponent):
  def __init__(self,sim=None,name="",axis="",z=0.0):
    optcomponent.__init__(self,sim,name,axis,z)
    os.makedirs(self.simobj.direct()+self.name)
  def action(self,beam):                                              #Pickle optical field data
    logfile.write('<save> '+self.name+' saving beam '+elapsedtime()+'\n')                    #Record activity
    record((self.simobj.direct() + self.name + '/'+str(self.simobj.config['time'])),beam)


class lens(optcomponent):
  def __init__(self, sim=None, name="", axis="", z=0.0, f=1.0):
    optcomponent.__init__(self, sim, name, axis, z)
    self.f = f
  def action(self, beam):
    logfile.write('<lens> Lens '+self.name+' applied to beam '+elapsedtime()+'\n')
    n = self.simobj.config['mesh resolution']
    l = infinity
    wave = infinity
    for ofield in beam:
      if (abs(l-ofield.scale)+abs(wave-ofield.wavelength))>tiny:                    #If last lens screen calculated doesn't apply to
next optical field, recalculate lens screen
        l = ofield.scale
        wave = ofield.wavelength
        lens.m = float(n-1)/2.0
        lens.d = float(n)/l
        lens.c = -2.0j*pi*self.f/(wave*abs(self.f))
        lens.foc = self.f
        def lns(x,y):
          return exp(lens.c*sqrt(lens.foc**2.0+((x-lens.m)/lens.d)**2.0+((y-lens.m)/lens.d)**2.0))
        screen = fromfunction(lns, (n,n))
      if ofield.space == "frequency":                                  #Ensure optical field in real space representation
        ofield.mesh = inverse_fft2d(ofield.mesh)
        ofield.space = "real"
      elif ofield.space != "real":
        raise simerror, simerror("<lens> Attempt to sample a non-existant mesh "+elapsedtime()+'\n')
      fmin = (4.0*(l**2)/(float(n-1)*wave))
      if fmin>self.f:                                                  #Error to apply a lens that is too strong for beam
        raise simerror, simerror('<lens> Lens '+self.name+' f='+str(self.f)+' too strong for beam (min f='+str(fmin)+') '+elapsedtime()+'\n')
      ofield.mesh = ofield.mesh*screen                                 #Apply lens to each optical field


class circularaperture(optscreen):
  def __init__(self,sim=None,name="",axis="",z=0.0, x=0.0, y=0.0, x_ang=0.0, y_ang=0.0, z_rot=0.0, plane=None,radius=1.0,iso='yes'):
    optscreen.__init__(self,sim,name,axis,z,x,y,x_ang,y_ang,z_rot,plane,iso)
    self.r = radius
  def function(self,pmap,lambd):
    screen = less_equal(abs(pmap)**2.0,self.r**2.0).astype(pmap.typecode())
    return screen


class squareaperture(optscreen):
  def __init__(self,sim=None,name="",axis="",z=0.0, x=0.0, y=0.0, x_ang=0.0, y_ang=0.0, z_rot=0.0, plane=None,side=1.0,iso='yes'):
    optscreen.__init__(self,sim,name,axis,z,x,y,x_ang,y_ang,z_rot,plane,iso)
    self.s = side
  def function(self,pmap,lambd):
    screen = greater_equal(pmap.real,-self.s/2.0)*less_equal(pmap.real,self.s/2.0)*greater_equal(pmap.imag,-self.s/2.0)*less_equal(pmap.imag,self.s/2.0)
    return screen


class zernikephase(newscreenfunction):
  def __init__(self, x=0.0, y=0.0, x_ang=0.0, y_ang=0.0, z_rot=0.0, plane=None, radius=1.0, comp=[0.0,0.0],iso='yes'):
```

```
      newscreenfunction.__init__(self, x, y, x_ang, y_ang, z_rot, plane, iso)
      self.r = radius
      self.comp = comp
    def function(self, pmap, lambd):
      return wrapper.field(zernike.compose(self.comp, self.r, pmap))


class randomzernikephase(newscreenfunction):
  def __init__(self, z=0.0, x=0.0, y=0.0, x_ang=0.0, y_ang=0.0, z_rot=0.0, plane=None, radius=1.0, modes=3, iso='yes'):
    newscreenfunction.__init__(self, x, y, x_ang, y_ang, z_rot, plane, iso)
    self.r = radius
    self.m = modes
  def function(self, pmap, lambd):
    return wrapper.field(zernike.compose(random(self.m+1), self.r, pmap))


class turbulence(optcomponent):
  def __init__(self, sim=None, name="", axis="", z=0.0, l=1.0, r0=0.1, f=1.0, v=[0.0,0.0]):
    "turbulent layer producing seeing of r0 at 5e-7m with turbulence fraction f in layer"
    optcomponent.__init__(self,sim,name,axis,z)
    self.Cn2 = kolmogorov.Cn2(5e-7,r0)*f
    self.l = l
    self.v = [v[0],v[1]]
    n = self.simobj.config['mesh resolution']
    self.phase = kolmogorov.turbulence(self.l,n)
    #Generate translation map
    def pos(x,y):
      return 1.0*x + 1.0j*y
    self.k = zeros((n,n),Complex)
    self.k[0:n/2+1,0:n/2+1] = fromfunction(pos,(n/2+1,n/2+1))/l
    self.k[n/2+1:n,0:n/2+1] = (fromfunction(pos,(n/2-1,n/2+1))-float(n/2-1))/l
    self.k[0:n/2+1,n/2+1:n] = (fromfunction(pos,(n/2+1,n/2-1))-1.0j*float(n/2-1))/l
    self.k[n/2+1:n,n/2+1:n] = (fromfunction(pos,(n/2-1,n/2-1))-(1.0+1.0j)*float(n/2-1))/l
  def action(self, beam):
    logfile.write('<turb> '+self.name+' applied to beam '+elapsedtime()+'\n')
    for ofield in beam:
      screen = wrapper.field(kolmogorov.normalise(self.phase,ofield.wavelength,self.Cn2))
      #translate
      x = ofield.x_pos + self.v[0]*self.simobj.config['time']
      y = ofield.y_pos + self.v[1]*self.simobj.config['time']
      #q = self.k.real*x/self.l + self.k.imag*y/self.l
      q = self.k.real*x + self.k.imag*y
      screen = inverse_fft2d(exp((-2.0j*pi)*q)*fft2d(screen))
      if ofield.space == "frequency":
        ofield.mesh = inverse_fft2d(ofield.mesh)
        ofield.space = "real"
      elif ofield.space != "real":
        raise simerror, simerror("<turb> Attempt to sample a non-existant mesh "+elapsedtime()+'\n')
      ofield.mesh = ofield.mesh*screen


class deformablezernikemirror(optscreen, electronicin):
  def __init__(self, sim=None, name="", axis="", z=0.0, x=0.0, y=0.0, x_ang=0.0, y_ang=0.0, z_rot=0.0, plane=None, radius=1.0, elecinput=[], iso='yes'):
    optscreen.__init__(self, sim, name, axis, z, x, y, x_ang, y_ang, z_rot,plane,iso)
    electronicin.__init__(self, sim, elecinput)
    self.r = radius
  def startstep(self):
    self.data = []
    self.process()                                                    #Get input signals
  def compute(self, sig):
    #print self.name,sig.data
    if len(self.data)==0:
      self.data=sig.data
    else:
      raise simerror, simerror("<dzrm> More than one control signal passed "+elapsedtime()+'\n')
  def function(self, pmap, lambd):
    if len(self.data)==0:
      self.data = [0.0,0.0]
    correction = zernike.compose(self.data, self.r, pmap)*(-2.0*pi/lambd)
    return wrapper.field(correction)


class zernikewavefrontsensor(optscreen, electronicout):
  def __init__(self, sim=None, name="", axis="", z=0.0, x=0.0, y=0.0, x_ang=0.0, y_ang=0.0, z_rot=0.0, plane=None, radius=1.0, modes=20, min_intens=-
0.0, num=5, iso='yes'):
    optscreen.__init__(self, sim, name, axis, z, x, y, x_ang, y_ang, z_rot, plane, iso)
    electronicout.__init__(self, sim)
    n = self.simobj.config["mesh resolution"]
```

```
        self.r = radius
        self.m = modes
        self.sens = min_intens
        self.num = num
    def action(self, beam):
        logfile.write('<zrws> '+self.name+' zernike wave front sensor applied to beam '+elapsedtime()+'\n')
        if len(beam)!=1:
            raise simerror, simerror('<zrws> Cannot use '+str(len(beam))+' optical fields for wave front sensing, must have 1 only. '+elapsedtime()+'\n')
        ofield = beam[0]
        if ofield.space == "frequency":
            ofield.mesh = inverse_fft2d(ofield.mesh)
            ofield.space = "real"
        elif ofield.space != "real":
            raise simerror, simerror("<zrws> Attempt to sample a non-existant mesh "+elapsedtime()+'\n')
        self.opd = wrapper.unwrap(ofield.mesh)*(ofield.wavelength/(2.0*pi))                                    #Get phase
        #Determine points zernike sensing working off
        pmap = ofield.projection(self)
        (n,n) = pmap.shape
        included = less_equal(abs(pmap),self.r)
        tpoints = sum(sum(included))
        fact = int(sqrt(float(tpoints)/float(self.num**2)))
        #Only cut down sampling points if there is a need to
        if fact>=2:
            #Find point closest to centre of wfs
            def pos(x,y):
                return x+1j*y
            p = fromfunction(pos,(n,n))
            inc = 2.0*min(reshape(abs(pmap),(-1,)))
            minim = less_equal(abs(pmap),2.0*min(reshape(abs(pmap),(-1,))))
            centre = sum(sum(p*minim))/sum(sum(minim))
            centre = int(centre.real) + 1j*int(centre.imag)                                    #centre must be an integer
            ax = int(max([centre.real-int(self.num/2.0)*fact,0]))                                    #num should be odd
            bx = int(min([centre.real+int(self.num/2.0)*fact+1,ofield.n]))
            ay = int(max([centre.imag-int(self.num/2.0)*fact,0]))
            by = int(min([centre.imag+int(self.num/2.0)*fact+1,ofield.n]))
            #Strip inputs down to selected discrete points
            pmap = pmap[ax:bx:fact,ay:by:fact]
            self.opd = self.opd[ax:bx:fact,ay:by:fact]
            #Include scintillation
            self.intensities = abs(ofield.mesh[ax:bx:fact,ay:by:fact])**2.0
        else:
            self.intensities = abs(ofield.mesh)**2.0
        self.sensing = greater(self.intensities,self.sens)
        (t,t) = self.sensing.shape
        mi = min(reshape(self.opd,(-1,)))
        ma = max(reshape(self.opd,(-1,)))
        s = ma - mi
        ra = s*random([t,t]) + mi
        sc = (1.0-self.sensing.astype(Float))
        self.opd = self.opd*self.sensing.astype(Float)  + sc*ra  #Scintillation occuring (random opd within existing range) where intensity insufficient for sensing
        #print self.name,'using',sum(sum(self.sensing)),'points for sensing at time',self.simobj.config['time']
        self.d = zernike.decompose(self.m, self.opd, self.r, pmap)                                    #Decompose signal
        self.d[1] = 0.0                                                                    #No piston mode sensed
        #print self.d
        self.lineout.store(-1.0,self.d)                                                        #Signal output immediately valid


class recordingzernikewavefrontsensor(zernikewavefrontsensor):
    def __init__(self, sim=None, name="", axis="", z=0.0, x=0.0, y=0.0, x_ang=0.0, y_ang=0.0, z_rot=0.0, plane=None, radius=1.0, resolution=8, modes = 3,
min_intens = -0.0, num=5, iso='yes'):
        zernikewavefrontsensor.__init__(self, sim, name, axis, z, x, y, x_ang, y_ang, z_rot, plane, radius, modes, min_intens, num, iso)
        os.makedirs(self.simobj.direct()+self.name)
    def endstep(self):
        record((self.simobj.direct() + self.name + '/'+str(self.simobj.config['time'])),[self.d,self.intensities,self.sensing,sum(sum(1-self.sensing))])   #Recording
decomposition & scintillation


class imager(optcomponent):
    def __init__(self, sim=None, name="", axis="", z=0.0, aperture = unitscreenfunction, image = 0.0, mag = 1.0, iso='yes'):
        optcomponent.__init__(self,sim,name,axis,z)
        self.pupil = aperture
        self.u = image
        self.mag = mag                                                                    #Magnification = image_scale/ofield.scale
        os.makedirs(self.simobj.direct()+self.name)
        self.iso = iso
        aperture.iso = iso
    def action(self,beam):
        if len(beam)!=1:
            raise simerror, simerror('<imge> Cannot image '+str(len(beam))+' beams; must be 1 only '+elapsedtime()+'\n')
        ofield = beam[0]
```

```
        if (self.u == infinity):
            logfile.write('<imge> Imaging object plane at infinity '+elapsedtime()+'\n')
            self.im = self.far(ofield)
        else:
            logfile.write('<imge> Using lens impulse responce to image object plane a distance '+str(self.u)+' away '+elapsedtime()+'\n')
            self.im = self.backprop(ofield)
    def far(self, ofield):                                          #Image far field
        ofield.applyphasescreen(self.pupil)
        err = ofield.getrelerr()
        if ofield.space == "frequency":
            ofield.mesh = inverse_fft2d(ofield.mesh)
            ofield.space = "real"
        elif ofield.space != "real":
            raise simerror, simerror("<imge> Attempt to sample a non-existant mesh "+elapsedtime()+'\n')
        t = (abs(fft2d(ofield.mesh))**2.0)/(self.mag**2.0)
        (n,n) = ofield.mesh.shape
        i = zeros((n,n), Float)
        i[0:(n/2-1),0:(n/2-1)] = t[(n/2+1):n,(n/2+1):n]                 #Reorder fourier into image ordering
        i[(n/2-1):n,0:(n/2-1)] = t[0:(n/2+1),(n/2+1):n]
        i[0:(n/2-1),(n/2-1):n] = t[(n/2+1):n,0:(n/2+1)]
        i[(n/2-1):n,(n/2-1):n] = t[0:(n/2+1),0:(n/2+1)]
        return {'scale':ofield.scale*self.mag, 'image':i, 'relerror':err}
    def backprop(self,ofield):                                      #Lens impulse technique
        ofield.propagate(ofield.z_pos-self.u)                       #Propagate back to object plane
        if ofield.space == "real":
            ofield.mesh = fft2d(ofield.mesh)
            ofield.space = "frequency"
        elif ofield.space != "frequency":
            raise simerror, simerror("<ofld> Attempt to sample a non-existant mesh "+elapsedtime()+'\n')
        if self.iso == 'yes':
            ofield.isolate()
        err = ofield.getrelerr()
        self.x_pos = ofield.x_pos                                   #Camera orientated for beam
        self.y_pos = ofield.y_pos
        self.x_angle = ofield.x_angle
        self.y_angle = ofield.y_angle
        self.z_rotation = 0.0
        pup = self.pupil.function(ofield.projection(self,-1.0/(ofield.wavelength*self.u+infinitesimal)),ofield.wavelength)  #pupil effect on image
        (n,n) = ofield.mesh.shape
        t = zeros((n,n),Complex)                                    #Reorder pupil effect into fourier ordering
        t[0:(n/2+1),0:(n/2+1)] = pup[(n/2-1):n,(n/2-1):n]
        t[(n/2+1):n,0:(n/2+1)] = pup[0:(n/2-1),(n/2-1):n]
        t[0:(n/2+1),(n/2+1):n] = pup[(n/2-1):n,0:(n/2-1)]
        t[(n/2+1):n,(n/2+1):n] = pup[0:(n/2-1),0:(n/2-1)]
        i = (abs(inverse_fft2d(t*ofield.mesh))**2.0)/(self.mag**2.0)
        return {'scale':ofield.scale*self.mag, 'image':i, 'relerror':err}


class instantaneousimage(imager):
    def __init__(self, sim=None, name="", axis="", z=0.0, image = 0.0, mag = 1.0, aperture = unitscreenfunction, iso='yes'):
        imager.__init__(self, sim, name, axis, z, aperture, image, mag, iso)
    def endstep(self):
        logfile.write('<inst> '+self.name+' saving image '+elapsedtime()+'\n')                  #Record activity
        record((self.simobj.direct() + self.name + '/'+str(self.simobj.config['time'])),self.im)


class integratingimager(instantaneousimage):
    def __init__(self, sim=None, name="", axis="", z=0.0, image = 0.0, mag = 1.0, start=0.0, stop=0.0, aperture = unitscreenfunction, iso='yes',inst='yes'):
        instantaneousimage.__init__(self,sim,name,axis,z,image,mag,aperture,iso)
        self.start = start
        self.stop = stop
        self.integrate = None
        self.num = 0
        self.inst = inst
    def action(self,beam):
        if (self.simobj.config['time']-self.start)>(-0.2/self.simobj.config['step frequency']):
            if (self.simobj.config['time']-self.stop)<(0.2/self.simobj.config['step frequency']):
                imager.action(self,beam)
    def endstep(self):
        if self.integrate == None:
            self.integrate = deepcopy(self.im)
        else:
            if abs(self.integrate['scale']-self.im['scale'])>(self.integrate['scale']*tiny):
                raise simerror, simmerror('<intg> Cannot integrate optical field that changes its scaling '+elapsedtime()+'\n')
            self.integrate['image'] = self.integrate['image'] + self.im['image']
            self.integrate['relerror'] = sqrt(self.integrate['relerror']**2.0 + self.im['relerror']**2.0)
        self.num = self.num + 1
        if abs(self.simobj.config['time']-self.stop)<(0.2/self.simobj.config['step frequency']):
            self.save()
        if self.inst=='yes':
```

```
            instantaneousimage.endstep(self)
    def save(self):
        logfile.write('<intg> '+self.name+' saving image '+elapsedtime()+'\n')                    #Record activity
        self.integrate['image'] = self.integrate['image']/float(self.num)
        self.integrate['relerror'] = self.integrate['relerror']/float(self.num)
        record((self.simobj.direct() + self.name + '/integrate'+str(self.simobj.config['time'])),self.integrate)
        print "Integrating imager '"+self.name+"' found image with peak intensity of",max(reshape(self.integrate['image'],(-1,))),"with relative
error",self.integrate['relerror'],"at time",self.simobj.config['time']
"""
```

'simphase.py'

```
"""
from simdata import *
from Numeric import *
from FFT import *
from RandomArray import *


def fact(x):
    "Factorial"
    x = int(x)
    f = 1
    for a in range(1,x+1):
        f=f*a
    return f

def even(x):
    "Even?"
    e = 0
    if 2*int(x/2)==x:
        e = 1
    return e


class newwrapper:
    def field(self,phase,amplitude = 1.0):
        "change phase into optical field with amplitude that can be a scalar or a array the same size as phase"
        return amplitude * exp(1.0j*phase)
    def getdiff(self,ph,refmap,opmap):
        (n,n) = ph.shape
        d = zeros((n,n),Complex)
        d[1:n,:] = (ph[1:n,:]-ph[0:n-1,:])*opmap[1:n,:]*refmap.real[1:n,:]
        d[:,1:n] = d[:,1:n] + 1.0j*(ph[:,1:n]-ph[:,0:n-1])*opmap[:,1:n]*refmap.imag[:,1:n]
        return d
    def getwrapping(self,d):                                                    #Average phase differences measured to be within
-2.0pi to 2.0pi
        wrapping = (d/pi).astype(Int)
        wrapping += where(greater(wrapping,0),1,0)
        wrapping -= where(less(wrapping,0),1,0)
        wrapping = -2.0*pi*(wrapping/2)
        return wrapping                                                         #Average phase differences wrapped to be within -pi to pi
    def getphase(self,d,refmap,ref):
        p1 = add.accumulate(d.real,0) * refmap.real
        p2 = add.accumulate(d.imag,1) * refmap.imag
        return (p1+p2)/ref
    def unwrap(self,field):
        "Unwrap the phase in an optical field"
        (n,n) = field.shape
        a = arctan(field.imag/(field.real+infinitesimal))
        b = pi + a
        phasemod1 = where(greater_equal(field.real,0.0),a,b)                    #Phase deduced in range -pi/2 to 3pi/2
        phasemod = where(greater(phasemod1,pi),phasemod1-2.0*pi,phasemod1)      #Phase deduced in range -pi to
pi
        refmap = zeros((n,n),Complex)
        opmap = greater(abs(field),tiny)
        refmap[1:n,:] = opmap[0:n-1,:]
        refmap[:,1:n] = refmap[:,1:n] + 1.0j*opmap[:,0:n-1]
        ref = choose((1.1*(refmap.real+refmap.imag)).astype(Int),(1.0,1.0,2.0))
        #1st round - construct an attempt at phase
        diff = self.getdiff(phasemod,refmap,opmap)
        difference = diff + self.getwrapping(diff.real)*refmap.real + 1.0j*self.getwrapping(diff.imag)*refmap.imag
        phase = self.getphase(difference,refmap,ref)
        #2nd round - measure actual diffrences & compare with difference & supply corrections
        diff2 = difference - self.getdiff(phase,refmap,opmap)
        return phase + self.getphase(diff2,refmap,ref)

wrapper = newwrapper()


class newkolmogorov:
```

```
  def turbulence(self,l,n,alpha = 3.1 ,beta = 3.0):
    "Obtain a kolmogorov phase distribution"
    newkolmogorov.fa = alpha
    newkolmogorov.fb = beta
    def amp(x,y):
      x = x.astype(Float)
      y = y.astype(Float)
      f2 = (x**2+y**2+infinitesimal)
      g = f2**newkolmogorov.fb
      fudge = (newkolmogorov.fa + g)/(1.0 + g)
      return fudge * f2**(-11.0/12.0)
    m = n/2 + 1
    a = ((float(n**2)*(l**(5.0/6.0))*0.1517)*fromfunction(amp,(m,m))).astype(Complex)
    b = zeros((n,m),Complex)
    ph = exp(2j*pi*random((n,m)))
    #Killing dc term
    ph[0,0] = 0
        #Killing off highest frequency terms
        ph[m-1:m+1,:]=0.0
        ph[:,m-1:m+1]=0.0
        #Done
    b[0:m,0:m] = a[0:m,0:m]
    b[m:n,0:m] = a[1:(m-1),0:m][::-1,::1]
    b = b*ph
    phase = inverse_real_fft2d(b)                                    #phase variations real
    return phase
  def fried(self, wavelength, Cn2):
    return (2.91/6.88*(2.0*pi/wavelength)**2.0*Cn2)**-0.6
  def Cn2(self,wavelength, r0):
    return r0**(-5.0/3.0) *6.88/2.91*(wavelength/(2.0*pi))**2
  def normalise(self,phase,wavelength,Cn2):
    "Normalise phase screen for a given wavelength & integrated turbulence"
    return phase*(self.fried(wavelength,Cn2)**(-5.0/6.0))
  def structure(self,phase,l):
    (n,n) = phase.shape
    m = n/2 + 1
    strip = phase - phase[0,:]
    strip = 2*strip**2
    strip[1:(m-1),:] = (strip[1:(m-1),:] + strip[m:n,:][::-1,::1])/2.0
    strip = strip[0:m,:]
    strip = sum(strip,1)/float(n)
    pos = arange(0,m,typecode=Float)
    pos = (l*float(n-1)/float(n**2))*pos
    return [pos,strip]
  def statstructure(self,wavelength,Cn2,pos):
    return [pos,6.88*(pos/self.fried(wavelength,Cn2))**(5.0/3.0)]

kolmogorov = newkolmogorov()


class newzernike:
  def getnm(self,j):
    num=0
    n=-1
    m=0
    e = 0
    while(num<j):
      if m<n:
        m=m+1
      else:
        n=n+1
        m=0
      if even(n-m):
        if m==0:
          num = num+1
        else:
          num = num+2
    return (n,m)
  def getang(self,pmap):
    a = arctan(pmap.imag/(pmap.real+infinitesimal))
    b = pi + a
    angle = where(greater_equal(pmap.real,0.0),a,b)                  #Angle deduced in range -pi/2 to 3pi/2
    return angle
  def radial(self,n,m,pmap):
    rad = zeros(pmap.shape,Float)
    r = abs(pmap)
    for s in range(0,(n-m)/2+1):
      coeff = ((-1)**s)*fact(n-s)
      coeff = coeff/(fact(s)*fact((n+m)/2-s)*fact((n-m)/2-s))
      rad = rad + coeff*(r**(n-2*s))
    return rad
  def mode(self,j,r,pmap):
```

```
      (n,m) = self.getnm(j)
      pmap = pmap/r                                          #Normalise pmap
      angle = self.getang(pmap)
      if m!=0:
        if even(j)==1:                                       #zernike angular dependance
          angular = sqrt(2)*cos(m*angle)
        else:
          angular = sqrt(2)*sin(m*angle)
      else:
        angular = ones(angle.shape,Float)
      radial = self.radial(n,m,pmap)                         #zernike radial dependance
      circle = less_equal(abs(pmap),1.0)
      z = sqrt(float(n+1))*radial*angular*circle             #j th zernike over circle of radius r on position map
pmap
      return z
    def decompose(self,n,wavefront,r,pmap):
      comp = range(0,n+1)
      (a,a) = pmap.shape
      vol = sum(sum(less_equal(abs(pmap),r)))
      for s in range(1,n+1):
        comp[s] = sum(sum(wavefront*self.mode(s,r,pmap)))/vol
      return comp
    def compose(self,comp,r,pmap):
      wavefront = zeros(pmap.shape,Float)
      for s in range(1,len(comp)):
        wavefront = wavefront + comp[s]*self.mode(s,r,pmap)
      return wavefront

zernike = newzernike()
"""
```

'simsplitters.py'

```
"""
from simdata import *
from simlog import *
from simerrors import *


class beamcollector(simcomponent):
  def __init__(self, sim=None, name="", inp=[""]):
    "Add new beam collection splitter called <name> to simulation <sim>, getting input from the axes <inp>"
    simcomponent.__init__(self,sim,name)
    for i in inp:
      if i=="":
        raise simerror, simerror('<splt> "" is not a valid name for a simulation element '+elapsedtime()+'\n')
    for ax in inp:
      if (type(ax) != type("")):
        raise simerror, simerror('<splt> Axis name must a string '+elapsedtime()+'\n')
      if self.simobj.axes.has_key(ax) != 1:
        raise simerror, simerror("<splt> Axis '"+ax+"' does not exist "+elapsedtime()+'\n')
    self.simobj.splitters[name] = self                       #Adding beam splitter to simulation
    self.simobj.optical_list.append(self)                    #Add to execution list
    objects = []
    for ax in inp:                                           #Finding input axes
      objects.append(self.simobj.axes[ax])
    self.l = []
    self.input = []
    for obj in objects:                                      #Recording length of previous axes and obtaining input meshes
      self.l.append(obj.l)
      if len(obj.components)>0:
        self.input.append((obj.components[-1]).output)       #Input is output of last object in axis
      else:
        self.input.append(obj.input)                         #Input is input of last axis as no objects in last axis
    for opt in self.input:                                   #Register need for data in inputs
      opt.add_ref()
    self.output = newoptdata()                               #Creating output optical data handler
    self.simobj.opt_data.append(self.output)                 #Putting output optical data handler into simulation
opt_data list
  def apply(self):
    logfile.write('<splt> '+self.name+' collecting beams '+elapsedtime()+'\n')   #Record activity
    x = []
    for i in range(len(self.input)):                         #Propogate optical fields to end of previous axis and collect
      b = (self.input[i]).retrieve()                         #Obtain input beam
      for m in b:                                            #Loop over each mesh in beam
        m.z_pos -= self.l[i]                                 #Resetting z_position of mesh for next axis
        x.append(m)                                          #Collecting meshes in output beam
    x = self.action(x)
    self.output.store(x)                                     #Store output
  def init():
```

```
        return None
    def action(self,beam):                                          #Do nothing to beam
        return beam


class beamcreator(simcomponent):
    def __init__(self, sim=None, name="", wave=5.0e-7, l=1.0, screenfuncs=[unitscreenfunction], angles=[], dist=0.0):      #dist appears to have no function
        "Add new beam creator called <name> to simulation <sim>, produces beams from screen function <screenfunc> of width <l>"
        simcomponent.__init__(self,sim,name)
        if len(angles)!=0.0:
            if len(angles)!=len(inp):
                raise simerror, simerror('<bmcr> Angular direction of new optical fields overridden, but incorrect number supplied '+elapsedtime()+'\n')
            else:
                for i in array(len(screenfuncs)):
                    screenfuncs[i].x_angle = angles[i][0]
                    screenfuncs[i].y_angle = angles[i][1]
                    screenfuncs[i].x_pos = -dist * tan(screenfuncs[i].x_angle)
                    screenfuncs[i].y_pos = -dist * tan(screenfuncs[i].y_angle)
        self.simobj.splitters[name] = self                          #Adding beam splitter to simulation
        self.simobj.optical_list.append(self)                       #Add to execution list
        self.sfuncs = screenfuncs
        self.wavelength = wave
        self.scale = l
        self.output = newoptdata()                                  #Creating output optical data handler
        self.simobj.opt_data.append(self.output)                    #Putting output optical data handler into simulation
opt_data list
    def apply(self):
        logfile.write('<splt> '+self.name+' creating beam '+elapsedtime()+'\n')                 #Record activity
        x = []
        for sfunc in self.sfuncs:
            x.append(newopticalfield(wave=self.wavelength, l=self.scale, res=self.simobj.config['mesh resolution'], sfunc=sfunc,
mask=self.simobj.config['mask']))     #Create beams
        self.output.store(x)
    def init():
        return None


class angularbeamselector(beamcollector):
    def __init__(self, sim=None, name="", inp=[""], x_ang=0.0, y_ang=0.0, tol=pi):
        beamcollector.__init__(self, sim, name, inp)
        self.coords = cs(x_ang, y_ang, 0.0)
        self.match = cos(tol)
    def action(self, beam):                                         #Only pass ofields travelling in desired direction
        y = []
        for ofield in beam:
            c = cs(ofield.x_angle, ofield.y_angle, 0.0)
            t = dp(c[2],self.coords[2])
            if t>self.match:
                y.append(ofield)
        return y


class chromaticselector(beamcollector):
    def __init__(self, sim=None, name="", inp=[""], wavelength=5e-7, tol=infinity):
        beamcollector.__init__(self, sim, name, inp)
        self.wavelength = wavelength
        self.match = tol
    def action(self, beam):
        y = []
        for ofield in beam:
            if abs(ofield.wavelength-self.wavelength)<self.match:
                y.append(ofield)
        return y


class beamexpander(beamcollector):                                  #Apply an expansion to beam
    def __init__(self, sim=None, name="", inp=[""], expansion=1.0):
        beamcollector.__init__(self, sim, name, inp)
        self.f = expansion
    def action(self, beam):
        for ofield in beam:
            ofield.propagate(0.0)
            ofield.scale = ofield.scale * self.f                    #Scale boosted by expansion
            ofield.mesh = ofield.mesh /self.f                       #Amplitude reduced by expansion
            ofield.x_angle = arctan(tan(ofield.x_angle)/self.f)     #Angle ofield propagating at reduced by expansion
            ofield.y_angle = arctan(tan(ofield.y_angle)/self.f)
        return beam
"""
```

'simsys.py'

```
"""
from simlog import *
logfile.write('<main> Program started at '+timetxt(starttime)+'\n')

#Modules containing simulation components
from Numeric import *
from simerrors import *
from simdata import *
from simaxis import *
from simsplitters import *
from simopt import *
from simelec import *
from simphase import *
from RandomArray import *
import os
seed()


class newsimulation:
   "Main simulation object"
   #constructor
   def __init__(self):
      logfile.write('<simo> New simulation object produced '+elapsedtime()+'\n')
      self.splitters = {}
      self.axes = {}
      self.optical_comp = {}
      self.electronic_comp = {}
      self.file = {'path':'', 'name':'sim'}                                    #put simulation data into 'sim' directory within home
directory
      self.config = {'step frequency':1.0, 'sim length':0.0, 'time':0.0, 'mesh resolution':256, 'mask':0.0}
      self.optical_list = []
      self.electronic_list = []
      self.opt_data = []
      self.elec_data = []
      self.simobj = self
   #
   #Internal methods
   #
   #
   #initialise for new run
   def initialise(self):
      for comp in (self.opt_data+self.elec_data):                              #Reinitialise system for reassembly
         comp.reset
   def direct(self):
      return self.file['path']+'/'+self.file['name']+'/'
   #
   #User methods
   #
   #simulate system
   def simulate(self):
      print "Simulating"
      logfile.write('<simo> Begining simulation '+elapsedtime()+'\n')
      try:
         self.initialise()                                                     #get ready to simulate
         l = self.config['sim length']
         f = self.config['step frequency']
         for self.config['time'] in (arange(int(l*f)+1)/f):                    #evaluate simulation at each time step
            logfile.write('<simo> Starting simulation time '+str(self.config['time'])+' '+elapsedtime()+'\n')
            for current_opt in self.optical_list:                              #loop over each optical component
               current_opt.apply()                                             #apply component to current beam
            logfile.write('<simo> Optical components done '+elapsedtime()+'\n')
            for current_elec in self.electronic_list:                          #loop over each electronic component
               current_elec.apply()                                            #apply electronic components
            logfile.write('<simo> Electronic components done '+elapsedtime()+'\n')
#           for elec in self.elec_data:                                        #Ensure electronic components all exhausted
#              elec.exhausted(self.config['time'])
      except simerror, err:                                                    #error
         err.msg()
         logfile.write('<simo> Emptying data store '+elapsedtime()+'\n')
         for dat in (self.opt_data+self.elec_data):                            #ensure data_store is emptied in the event of an error
            dat.empty()
      print "Simulation ended"
      logfile.write('<simo> Simulation ended '+elapsedtime()+'\n')
   #
   #set options
   def set_options(self, path = '~/', name = 'sim', length = 0.0, frequency = 1.0, res = 8, mask = 0.0):
      "Path to place simulation data in directory namexxxx (where xxxx is combination no.). Simulation over time length with steps of given frequency using a
mesh of resolution 2^res with isolated beam masking fraction of mask"
```

```
        self.file['path'] = path
        self.file['name'] = name
        if length>=0.0:
            self.config['sim length'] = length
        else:
            raise simerror, simerror('<simo> Simulation length '+str(length)+' unphysical '+elapsedtime()+'\n')
        if frequency>0.0:
            self.config['step frequency'] = frequency
        else:
            raise simerror, simerror('<simo> Unacceptable frequency '+str(frequency)+' '+elapsedtime()+'\n')
        if abs(float(int(res))-res)>tiny:
            raise simerror, simerror('<simo> Supplied mesh resolution not an integer, '+str(res)+' '+elapsedtime()+'\n')
        self.config['mesh resolution'] = 2**int(res)
        if mask<0.0 or mask>=0.5:
            raise simerror, simerror('<simo> Unacceptable masking fraction '+str(mask)+' '+elapsedtime()+'\n')
        else:
            self.config['mask'] = mask
"""
```

'testing.py'

```
"""
from simsys import *
from gist import *
from pl3d import *
from plwf import *


data_path = '/home/rmd/sim/'    #Ensure this directory is empty


def pos(l,n):
    def xy(x,y):
        return x + 1.0j*y
    p = fromfunction(xy, (n,n))
    p = p - float(n-1)*(0.5+0.5j)
    return l*p/float(n)


pldefault(marks=0, width=0.0, type=1, style="nobox.gs", dpi=100)
winkill(0)
print "Welcome to simulation testing (4 rounds of tests)"
print "Please respond to prompts for displays of processed data (remember to select this text output window in order to provide input)."
print "Please be patient as simulation is being run to produce output data."
print
print "Testing gist display - restart if error reported (known to work correctly on aipc18 as user rmd)"
window(0, wait=1, dpi=100)
palette('gray.gp')
fmaa()

print
raw_input("Type return to continue")
print
print
print "Performing 1st simulation"
print "Imaging a circular aperture and far field image as a demonstration of simulation imaging components"
print

sim = newsimulation()
sim.set_options(path=data_path, name="test1")
beamcreator(sim, name="source", l=0.006, screenfuncs=[unitscreenfunction])
newaxis(sim,name="axis0", inp="source")
circularaperture(sim, axis="axis0", name="pupil", z=0.0, radius=0.0005)
beamcollector(sim, name="splitter", inp=["axis0"])
newaxis(sim, name="axis1", inp="splitter")
instantaneousimage(sim, name="camera1", axis="axis1", z=1.0, image=1.0)
newaxis(sim, name="axis2", inp="splitter")
instantaneousimage(sim, name="camera2", axis="axis2", z=0.0, image=infinity)
newaxis(sim, name="axis3", inp="splitter")
sim.simulate()

f = open(data_path+'/test1/camera1/0.0')
u = cPickle.Unpickler(f)
dat = u.load()
pli(dat['image'])
print "Image taken with camera 1.0m from object object plane (impulse responce technique)."
print "Scale =",dat['scale']
fmaa()

f = open(data_path+'/test1/camera2/0.0')
u = cPickle.Unpickler(f)
```

```
dat = u.load()
raw_input("Type return to continue")
pli(dat['image'])
print "Image taken with object plane at infinity (far field technique)."
print "Scale =",dat['scale']
fmaa()

d = dat['image'][112:144,112:144]
p = pos(1,32)
raw_input("Type return to continue")
set_draw3_(0)
orient3()
light3()
plwf(d,p.imag,p.real,shade=1,edges=1)
lim = draw3(1)
limits(lim[0],lim[1],lim[2],lim[3])
print "Surface intensity plot of central region."
print "Scale =",0.125*dat['scale']
fmaa()




print
print
print "Performing 2nd simulation"
print "A 1.0x1.0 m square aperture with various offsets and orientations to demonstrate the simulation co-ordinate system"
print

sim = newsimulation()
sim.set_options(path=data_path, name="test2")
beamcreator(sim, name="source", l=2.0, screenfuncs=[unitscreenfunction])
newaxis(sim,name="axis0", inp="source")
squareaperture(sim, axis="axis0", name="pupil0", z=0.0, x=0.0, y=0.0, side=1.0)
instantaneousimage(sim, name="camera0", axis="axis0", z=0.0)
newaxis(sim,name="axis1", inp="source")
squareaperture(sim, axis="axis1", name="pupil1", z=0.0, x=0.45, y=0.45, side=1.0)
instantaneousimage(sim, name="camera1", axis="axis1", z=0.0)
newaxis(sim,name="axis2", inp="source")
squareaperture(sim, axis="axis2", name="pupil2", z=0.0, z_rot=0.05, side=1.0)
instantaneousimage(sim, name="camera2", axis="axis2", z=0.0)
newaxis(sim,name="axis3", inp="source")
squareaperture(sim, axis="axis3", name="pupil3", z=0.0, z_rot=pi/4.0, side=1.0)
instantaneousimage(sim, name="camera3", axis="axis3", z=0.0)
newaxis(sim,name="axis4", inp="source")
squareaperture(sim, axis="axis4", name="pupil4", z=0.0, x_ang=pi/3.0, z_rot=pi/4.0, side=1.0)
instantaneousimage(sim, name="camera4", axis="axis4", z=0.0)
newaxis(sim,name="axis5", inp="source")
squareaperture(sim, axis="axis5", name="pupil5", z=0.0, x=0.45, y=0.45, x_ang=0.7227, side=1.0)
instantaneousimage(sim, name="camera5", axis="axis5", z=0.0)
sim.simulate()


f = open(data_path+'/test2/camera0/0.0')
u = cPickle.Unpickler(f)
dat = u.load()
raw_input("Type return to continue")
winkill(0)
pldefault(marks=0, width=0.0, type=1, style="boxed.gs", dpi=100)
window(0, wait=1, dpi=100)
pli(dat['image'])
print "Image of square aperture centred at (0,0)"
print "Scale =",dat['scale']
fmaa()

f = open(data_path+'/test2/camera1/0.0')
u = cPickle.Unpickler(f)
dat = u.load()
raw_input("Type return to continue")
pli(dat['image'])
print "Image of square aperture centred at (0.45,0.45)."
print "Scale =",dat['scale']
fmaa()

f = open(data_path+'/test2/camera2/0.0')
u = cPickle.Unpickler(f)
dat = u.load()
raw_input("Type return to continue")
pli(dat['image'])
print "Image of square aperture centred at (0,0) rotated by 0.05 radians around l.o.s. axis."
print "Scale =",dat['scale']
```

```
fmaa()

f = open(data_path+'/test2/camera3/0.0')
u = cPickle.Unpickler(f)
dat = u.load()
raw_input("Type return to continue")
pli(dat['image'])
print "Image of square aperture centred at (0.45,0.45) rotated by pi/4.0 radians about l.o.s. axis"
print "Scale =",dat['scale']
fmaa()

f = open(data_path+'/test2/camera4/0.0')
u = cPickle.Unpickler(f)
dat = u.load()
raw_input("Type return to continue")
pli(dat['image'])
print "Image of square aperture centred at (0,0) rotated by pi/3.0 radians about x_axis and pi/4.0 radians around l.o.s. axis."
print "Scale =",dat['scale']
fmaa()

f = open(data_path+'/test2/camera5/0.0')
u = cPickle.Unpickler(f)
dat = u.load()
raw_input("Type return to continue")
pli(dat['image'])
print "Image of square aperture centred at (0.45,0.45) rotated by 0.7227 radians about x-axis."
print "Scale =",dat['scale']
fmaa()

print
print
print "Performing 3rd simulation"
print "Demonstration of fresnel propagation effects by comparison with results given by the cornu spiral."
print "Using a square aperture of side 0.002m in grid of side 0.004m after a propagation distance of 0.05m."
print

sim = newsimulation()
sim.set_options(path=data_path, name="test3")
beamcreator(sim, name="source", l=0.004, screenfuncs=[unitscreenfunction])
newaxis(sim,name="axis0", inp="source")
squareaperture(sim, axis="axis0", name="pupil0", z=0.0, side=0.002, iso='no')
instantaneousimage(sim, name="camera0", axis="axis0", z=0.05)
sim.simulate()


print "Numerically evaluating fresnel integrals & finding same image"

lambd = 5.0e-7        # Wavelength
n = 2**8              # n = 2**i
z = 0.05              # propagation distance
l = 4.0e-3            # length of side of planes
scale = l*(2.0/(z*lambd))**0.5        # 0:scale is desired range of u,v values
num = 2000000                         # number of entries in u,v tables
acc = scale**3*(1+pi*scale**2)*pi/(12*float(num)**2)
print "Tabulating Fresnel integrals to",acc,"accuracy"
print
# Fresnel integral precalculations
def fc(n, m):
   a = zeros((2,n), Float)
   a[0,:] = arange(0, (1.0+1.0/(2*(n-1.0))), (1.0/(n-1.0)), Float)
   a[1,:] = cos(pi*(m*a[0,:])**2/2.0)
   a[1,:] = m*(add.accumulate(a[1,:])-(a[1,0]+a[1,:])/2.0)/n
   return a

def fs(n, m):
   a = zeros((2,n), Float)
   a[0,:] = arange(0, (1.0+1.0/(2*(n-1.0))), (1.0/(n-1.0)), Float)
   a[1,:] = sin(pi*(m*a[0,:])**2/2.0)
   a[1,:] = m*(add.accumulate(a[1,:])-(a[1,0]+a[1,:])/2.0)/n
   return a

ctable = fc(num, scale)
stable = fs(num, scale)

#Functions

def cval(w):
   if (w>0.0):
      v = 1
   else:
      w = -w
      v = -1
```

```
    i = searchsorted(ctable[0,:],w)
    val = v*((w-ctable[0,(i-1)])/(ctable[0,i]-ctable[0,(i-1)])*(ctable[1,i]-ctable[1,(i-1)]) + ctable[1,(i-1)])
    return val

def sval(w):
    if (w>0.0):
        v = 1
    else:
        w = -w
        v = -1
    i = searchsorted(stable[0,:],w)
    val = v*((w-stable[0,(i-1)])/(stable[0,i]-stable[0,(i-1)])*(stable[1,i]-stable[1,(i-1)]) + stable[1,(i-1)])
    return val

xfact = zeros((n,), Float)        # creating fresnel (multiplicative) contributions

x = 0
while (x < n):
    u1 = (float(x)-(float(n)/4.0-0.5))/float(n)
    u2 = (float(x)-(3.0*float(n)/4.0-0.5))/float(n)
    xfact[x] = (cval(u2)-cval(u1))**2+(sval(u2)-sval(u1))**2
    x = x+1
yfact = xfact                     # as aperature is a rectangle, xfact and yfact are identicle

c = xfact[NewAxis,:]*yfact[:,NewAxis]/4.0

pldefault(marks=0, width=0.0, type=1, style="nobox.gs", dpi=100)
winkill(0)
window(0, wait=1, dpi=100)
palette('gray.gp')
fmaa()

f = open(data_path+'/test3/camera0/0.0')
u = cPickle.Unpickler(f)
dat = u.load()
raw_input("Type return to continue")
pli(dat['image'])
print "Intensity pattern 0.05m away from aperture according to wave optics (err estimate due to discrete representation =",dat['relerror'],")"
print "Scale =",dat['scale']
fmaa()

raw_input("Type return to continue")
pli(c)
print "Intensity pattern 0.05m away from aperture according to fresnel integrals."
print "Scale =",0.004
fmaa()

raw_input("Type return to continue")
print "Rms difference is",sqrt(sum(sum(abs(dat['image']-c)**2.0/65536.0)))
fmaa()


print
print
print "Performing 4th test"
print "Demonstration of the realisation of kolmogorov with and without the authors modification (using averages over 20 realisations)"
print "Using a grid of side 1m with r0=0.1m"
print

w = 5e-7
n = 256
l = 1.0
r0 = 0.1
c = kolmogorov.Cn2(w,r0)
struct1 = [(l*float(n-1)/float(n**2))*arange(0,n/2+1,typecode=Float),zeros((n/2+1,))]
struct2 = [(l*float(n-1)/float(n**2))*arange(0,n/2+1,typecode=Float),zeros((n/2+1,))]
stat = [(l*float(n-1)/float(n**2))*arange(0,n/2+1,typecode=Float),zeros((n/2+1,))]
for i in range(20):
    p1 = kolmogorov.turbulence(l,n,0.0,0.0)
    p2 = kolmogorov.turbulence(l,n)
    p1 = kolmogorov.normalise(p1,w,c)
    p2 = kolmogorov.normalise(p2,w,c)
    struct1[1] = struct1[1]+kolmogorov.structure(p1,l)[1]
    struct2[1] = struct2[1]+kolmogorov.structure(p2,l)[1]
struct1[1] = struct1[1]/20.0
struct2[1] = struct2[1]/20.0
stat = kolmogorov.statstructure(w,c,stat[0])

raw_input("Type return to continue")
winkill(0)
pldefault(marks=0, width=3.0, type=1, style="work.gs", dpi=100)
window(0, wait=1)
```

```
palette('rainbow.gp')
plg(stat[1],stat[0],color=130)
plg(struct2[1],struct2[0],color=90)
plg(struct1[1],struct1[0],color=0)
fmaa()
print "Structure functions of kolmogorov turbulence."
print "Blue: statistical realisation"
print "Red: numerical realisation (suffering from periodicity)"
print "Green: numerical realisation (periodicity improvement)"
print



raw_input("Type return to continue")
winkill(0)
print "Done"
"""
```

'vector.py'

```
"""
from Numeric import *
#vectors are arrays
#coordinate systems are lists of vectors

def cp(a,b):                                                    #Cross product
   return array([a[1]*b[2] - a[2]*b[1], a[2]*b[0] - a[0]*b[2], a[0]*b[1] - a[1]*b[0]],Float)

def dp(a,b):                                                    #Dot product
   return sum(a*b)

def cs(x_angle=0.0,y_angle=0.0,z_rotation=0.0):
   z_ = array([sin(x_angle), sin(y_angle), sqrt(1 - sin(x_angle)**2 - sin(y_angle)**2)])
   phi = arccos(z_[0])
   w_ = array([1.0,0.0,0.0]) - z_
   w = sqrt(dp(w_,w_))
   theta = arcsin(sin(phi)/w)
   psi = pi/2 - theta
   sigma = 1.0/sin(psi)
   eta = sin(theta)*sigma
   x__ = (z_ + sigma*w_/w)/eta
   y__ = cp(z_,x__)
   x_ = cos(z_rotation)*x__ + sin(z_rotation)*y__
   y_ = -sin(z_rotation)*x__ + cos(z_rotation)*y__
   return [x_,y_,z_]

def convert(r, cs):
   "Convert global representation of vector r into a local representation cs (a list of the x,y,z orthonormal vectors)"
   gx = array([1,0,0],Float)
   gy = array([0,1,0],Float)
   gz = array([0,0,1],Float)
   xp = array([dp(gx,cs[0]),dp(gx,cs[1]),dp(gx,cs[2])],Float)
   yp = array([dp(gy,cs[0]),dp(gy,cs[1]),dp(gy,cs[2])],Float)
   zp = array([dp(gz,cs[0]),dp(gz,cs[1]),dp(gz,cs[2])],Float)
   return r[0]*xp + r[1]*yp + r[2]*zp
"""
```