Images & turbulence — 20

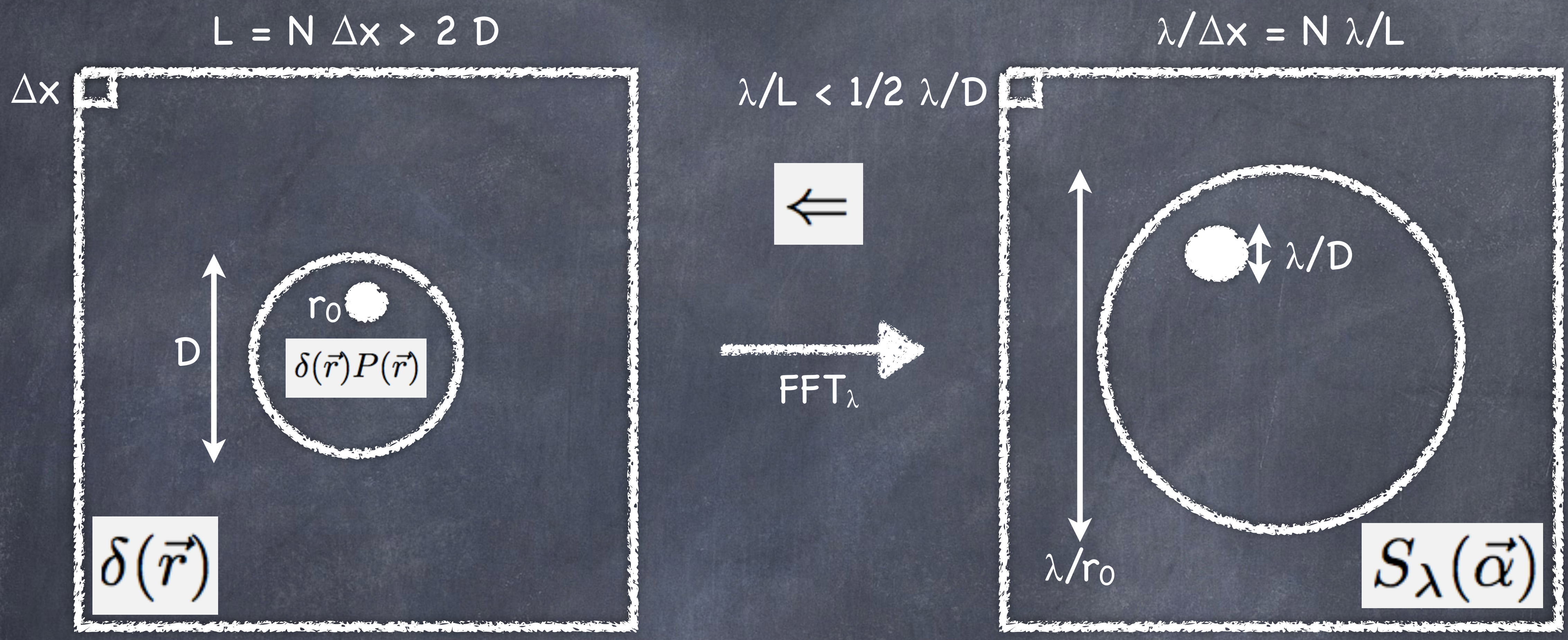
$$\Psi(\vec{r}) = A \ \exp\left(\imath\Phi(\vec{r})\right)$$

$$P(\vec{r}) \Rightarrow A\ P(\vec{r}) \ \exp\left(\imath\Phi(\vec{r})P(\vec{r})\right)$$

$$S_\lambda(\vec{\alpha}) \propto \| FT\{A\ P(\vec{r}) \ \exp\left(\imath\Phi(\vec{r})P(\vec{r})\right)\}\|^2$$

$$A = 1 \text{ and } \Phi(\vec{r}) = \frac{2\pi}{\lambda}\delta(\vec{r}) \Rightarrow S_\lambda(\vec{\alpha}) \propto \| FT\{P(\vec{r}) \ \exp\left(\imath\frac{2\pi}{\lambda}\delta(\vec{r})P(\vec{r})\right)\}\|^2$$

Images & turbulence — 21



Shannon (=Nyquist) criterium

=> the image pixel λ/L must be at most half the resolution element (resel!) λ/D
(in other words : one must have AT LEAST 2 image pixels per λ/D)

=> the simulated wavefronts must be at least twice the telescope diameter (L>2D)

In addition

- λ/r_0 should be smaller than λ/Δx (=> N large enough)

Images & turbulence — 22

```
function wfimg, dim, length, L0, r0, lambda_r0, obs, diam, lambda_psf, n_psf, filename
;
; use:
; dim        = 128L      ; [px] wf dimension
; length     = 2.        ; [m] wf physical dimension
; L0         = 27.       ; [m] outerscale
; r0         = .1        ; [m] Fried parameter
; lambda_r0 = 500E-9     ; [m] r0 wavelength
; obs        = 0. [0-1]  ; (linear) obscuration ratio
; diam       = dim/2     ; [px] telescope pupil dimension
; lambda_psf= 500E-9     ; [m] PSF wavelength
; n_psf      = 100L      ; nb of generated statistically independent PSFs
; filename   = 'cube.sav'; cube of PSFs filename
;
; print, wfimg(dim,length,L0,r0,lambda_r0,obs,diam,lambda_psf,n_psf,filename)
;
; sub-routines needed: image.pro, wfgeneration.pro, makepup.pro
;
; Marcel Carbillet [marcel.carbillet@unice.fr], Lagrange (UCA, OCA, CNRS), Feb. 2018.
;
cube = fltarr(dim,dim,n_psf)

for i=0, n_psf/2-1L do begin
  dummy = wfgeneration(dim,length,L0,r0,lambda_r0,SEED=seed)
  wf1   = float(dummy)
  wf2   = imaginary(dummy)
  dummy = makepup(dim,diam,obs)
  img1  = image(dummy,wf1,lambda_psf)
  img2  = image(dummy,wf2,lambda_psf)
  cube[*,*,2*i]   = img1
  cube[*,*,2*i+1] = img2
endfor

save, cube, FILENAME=filename

return, 'Cube of PSFs '+filename+' saved on disk...'
end
```

<u>image formation:</u>

1- cube of instantaneous PSFs (500nm & H-band)

```
function image, pup, wf, lambda
;
; image computation from a wavefront
;
; pup    = pupil,
; wf     = wavefront [float],
; lambda = wavelength at which image is computed.
;
; Marcel Carbillet [marcel.carbillet@unice.fr],
; UMR 7293 Lagrange (UNS/CNRS/OCA), Feb. 2013.
;
; Last modification: Feb. 2019
;
dim = (size(wf))[1]
img = (abs(fft(pup*exp(complex(0,1)*2*!PI/lambda*wf*pup))))^2
; NB: (abs(fft(pup*exp(complex(0,1)*2*!PI/lambda*wf))))^2 would suffice
img = shift(temporary(img), dim/2, dim/2)
; NB: shift(img, dim/2, dim/2) OK too

return, img
end
```

```
IDL> .r wfimg
% Compiled module: WFIMG.
IDL> print, wfimg(128L,2.,27.,0.1,500E-9,0.,64L,500E-9,100L,'cube.sav')
Cube of PSFs cube.sav saved on disk...
```

(IDL – 4)

Useful remarks concerning IDL

- IDL help is called with: IDL>> ?
- '?' opens with a defined browser the file 'idl.htm', here:
  …/exelis/…/idl/idl.htm
- This file can also be found with the unix command 'find':
  unix>> cd /
  unix>> find . -name idl.htm
- See also (for routines which are part of a third library):
  IDL>> doc_library, 'routine_name'
- return to main level of programming after a crash: **retall**
- details on a variable xxx: **idl> help, xxx**
  (see all variables: **idl> help**)

Images & turbulence — 23

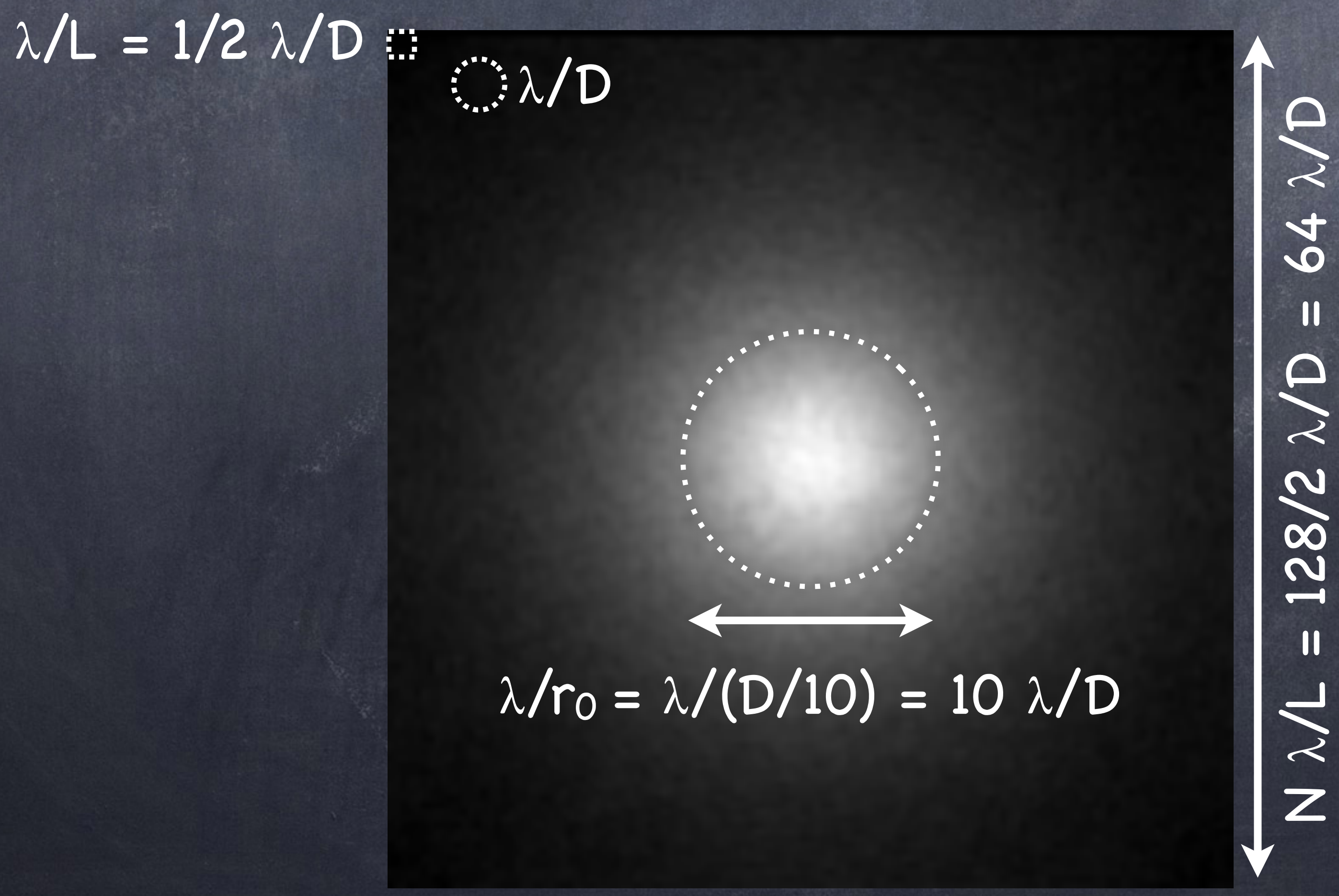
```
IDL> restore, 'cube.sav'
IDL> help
% At $MAIN$
CUBE            FLOAT     = Array[128, 128, 100]
Compiled Procedures:
    $MAIN$

Compiled Functions:
  COMPUTE_RMS DIST        IMAGE        MAKEPUP      WFCUBE
    WFGENERATION        WFIMG

IDL> for i=0,99 do tvscl, cube[*,*,i]
```

```
IDL> longexp = total(cube, 3)
IDL> tvscl, longexp^.1
```

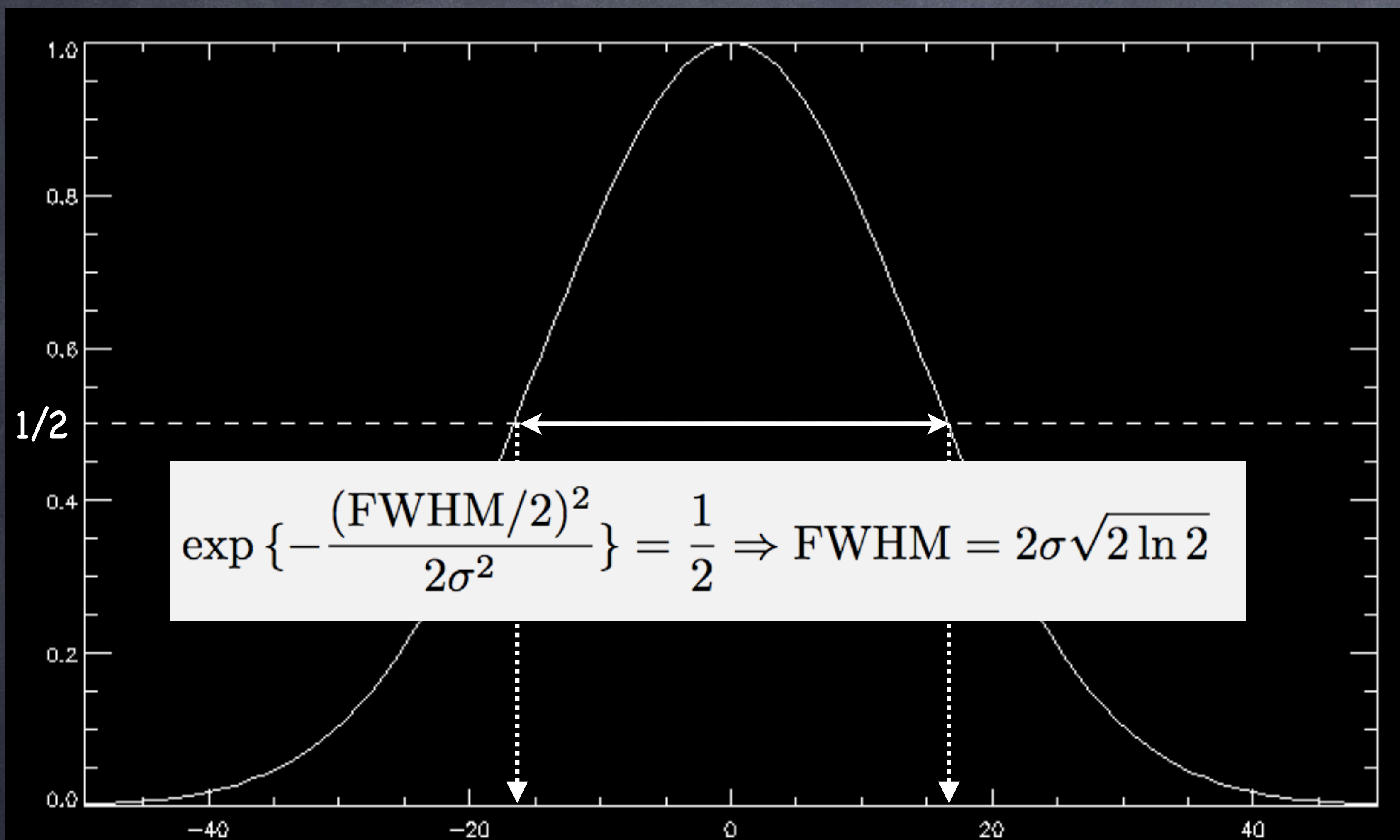
<u>image formation:</u>

1- cube of instantaneous PSFs (500nm & H-band)

2- long-exposure PSF



Images & turbulence — 24



<u>image formation:</u>

1- cube of instantaneous PSFs (500nm & H-band)

2- long-exposure PSFs

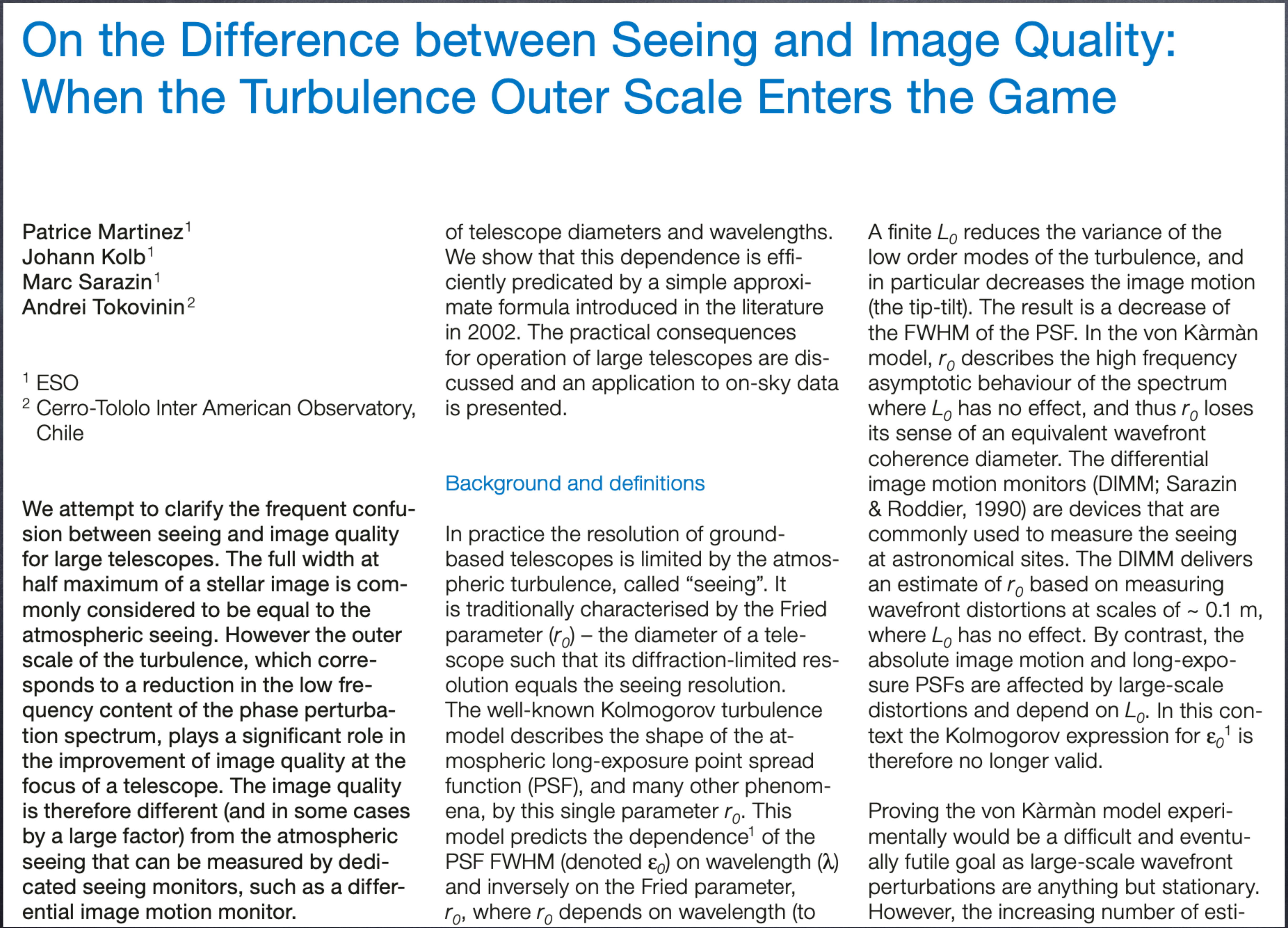
3- fit with gaussian and compare FWHM vs. λ/r_0 (seeing), also in function of the outerscale L_0.

-> Also read Martinez...

```
IDL> restore, 'cube.sav'
IDL> longexp=total(cube,3)
IDL> tvscl, longexp
IDL> res=gauss2dfit(longexp,a)
% Program caused arithmetic error: Floating underflow
IDL> print, 2*(a[2]+a[3])/2*sqrt(2*alog(2))
      15.5423
```

In this example, the FWHM is ~15.54px and, since we have here: 1px=(λ/D)/2, we have hence: FWHM≈7.77 (λ/D)
[i.e. 7.77/10≈0.78 arcsec here (λ=500nm)]

Images & turbulence — 25



On the Difference between Seeing and Image Quality: When the Turbulence Outer Scale Enters the Game

Patrice Martinez[1]
Johann Kolb[1]
Marc Sarazin[1]
Andrei Tokovinin[2]

[1] ESO
[2] Cerro-Tololo Inter American Observatory, Chile

We attempt to clarify the frequent confusion between seeing and image quality for large telescopes. The full width at half maximum of a stellar image is commonly considered to be equal to the atmospheric seeing. However the outer scale of the turbulence, which corresponds to a reduction in the low frequency content of the phase perturbation spectrum, plays a significant role in the improvement of image quality at the focus of a telescope. The image quality is therefore different (and in some cases by a large factor) from the atmospheric seeing that can be measured by dedicated seeing monitors, such as a differential image motion monitor.

of telescope diameters and wavelengths. We show that this dependence is efficiently predicated by a simple approximate formula introduced in the literature in 2002. The practical consequences for operation of large telescopes are discussed and an application to on-sky data is presented.

Background and definitions

In practice the resolution of ground-based telescopes is limited by the atmospheric turbulence, called "seeing". It is traditionally characterised by the Fried parameter (r_0) – the diameter of a telescope such that its diffraction-limited resolution equals the seeing resolution. The well-known Kolmogorov turbulence model describes the shape of the atmospheric long-exposure point spread function (PSF), and many other phenomena, by this single parameter r_0. This model predicts the dependence[1] of the PSF FWHM (denoted ε_0) on wavelength (λ) and inversely on the Fried parameter, r_0, where r_0 depends on wavelength (to

A finite L_0 reduces the variance of the low order modes of the turbulence, and in particular decreases the image motion (the tip-tilt). The result is a decrease of the FWHM of the PSF. In the von Kàrmàn model, r_0 describes the high frequency asymptotic behaviour of the spectrum where L_0 has no effect, and thus r_0 loses its sense of an equivalent wavefront coherence diameter. The differential image motion monitors (DIMM; Sarazin & Roddier, 1990) are devices that are commonly used to measure the seeing at astronomical sites. The DIMM delivers an estimate of r_0 based on measuring wavefront distortions at scales of ~ 0.1 m, where L_0 has no effect. By contrast, the absolute image motion and long-exposure PSFs are affected by large-scale distortions and depend on L_0. In this context the Kolmogorov expression for ε_0[1] is therefore no longer valid.

Proving the von Kàrmàn model experimentally would be a difficult and eventually futile goal as large-scale wavefront perturbations are anything but stationary. However, the increasing number of esti-

```
REPORT

- Preliminary measures
+ introduction
+ PSD(r0, L0) plot
+ => ccl on the influence of r0 and L0
+ rms(r0, L0) plot or table
+ => ccl on the influence of r0 and L0
+ image formation and FWHM(r0 or lambda, possibly L0)
+ => ccl on the influence of r0 or lambda (and poss. L0)
+ => comparison with the 'seeing' lambda/r0
+ (more to come...)
```

Images & turbulence — 26

-> Detection noises:

• At first: *photon noise* (or *shot noise*), poissonian, actually a transformation of the image.

$$p(n) = \frac{N^n e^{-N}}{n!}, \text{with} : N = \frac{L\Delta t}{h\nu}, L = \text{luminosity}, \Delta t = \text{time exp.}$$

p(n) = probability to detect n photons when N are expected

For large N: ~gaussian…

$$p(n) \simeq \exp\left(-\frac{(n-N)^2}{2N}\right)$$

Images & turbulence — 27

-> Detector noises:

• At first: *photon noise* (or *shot noise*), poissonian, actually a transformation of the image.

• At last: *read-out noise* (*RON*), gaussian with zero mean and rms σ_e [e-/px], additive noise.

• In between: *dark current noise*, *amplification noise* & *exotic dark current noise* in the case of EMCCDs, noise due to the *calibration* of the *flat field*, *'salt & pepper' noise* ('hot' and 'cold' pixels), etc.

Images & turbulence — 28

```
;; Photon noise (Poisson)
if keyword_set(PHOT_NOISE) then begin
   idx=where((image GT 0.) AND (image LT 1E8),c)
                                              ; For values higher than 1E8, should one
   if (c NE 0) then for i=0l,c-1l do $       ; really has to worry about photon noise ?
      noisy_image[idx[i]]=randomn(seed_pn,POISSON=image[idx[i]],/DOUBLE)
endif

;; Additive dark-current noise (Poisson)
if keyword_set(SIGMA_DARK) then begin
   if not(keyword_set(DELTA_T)) then begin
      message, "dark-current noise calculation does need a time exposure value!!"
   endif else noisy_image+=randomn(seed_dark,npx,npy,POISSON=sigma_dark*delta_t,/DOUBLE)
endif

;; EMCCD noises
; Additive exotic (time-exposure-independent) dark-current noise (Poisson)
if keyword_set(EXODARK) then noisy_image+=randomn(seed_xd,npx,npy,POISSON=exodark,/DOUBLE)

; Additive main EMCCD noise (Gamma)
if keyword_set(GAIN_L3CCD) then begin
   idx=where(image GT 0, c)
   if (c NE 0) then for i=0l,c-1l do $
      noisy_image[idx[i]]+=gain_l3ccd*randomn(seed_l3ccd,GAMMA=image[idx[i]],/DOUBLE)
;   noisy_image=long(temporary(noisy_image))
endif

;; Flat-field calibration residuals
if keyword_set(FFOFFSET) then begin
   ffres=randomn(seed_ff,npx,npy)*ffoffset+1.
   idx = where(ffres LE 0., c)
   if (c NE 0) then ffres[idx]=1.            ; Put possible<=0 ff values to 1.
   noisy_image*=ffres
endif

;; Additive read-out noise (Gaussian)
if keyword_set(SIGMA_RON) then $
   noisy_image+=randomn(seed_ron,npx,npy,/NORMAL,/DOUBLE)*sigma_ron

; Force to zero negative values
if keyword_set(POSITIVE) then begin
   idx=where(noisy_image LT 0, c)
   if (c GT 0) then noisy_image[idx]=0.
endif
```

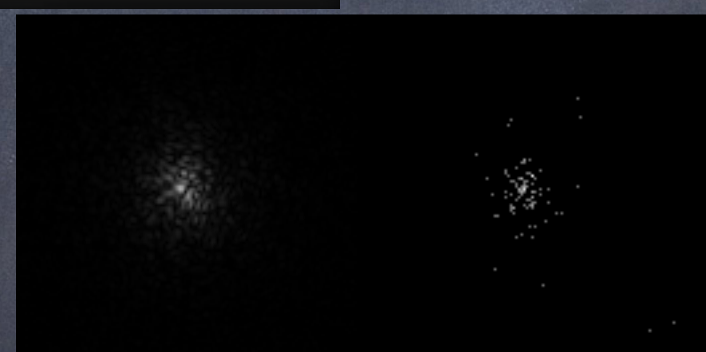
```
; CALLING SEQUENCE:
;    noisy_image = addnoise(input_image,               $
;                           PHOT_NOISE=phot_noise,     $
;                           SIGMA_DARK=sigma_dark,     $
;                           DELTA_T=delta_t,           $
;                           EXODARK=exodark,           $
;                           GAIN_L3CCD=gain_l3ccd,     $
;                           FFOFFSET=ffoffset,         $
;                           SIGMA_RON=sigma_ron,       $
;                           POSITIVE=positive,         $
;                           OUT_TYPE=out_type          )
```

<u>img formation w/noise:</u>

1- ‘add’ photon noise on one short-exp. PSF (in function of N…),
2- long-exp. PSF (100N photons!),
3- ‘add’ photon noise on the long-exp. PSF,
4- compare long-exp. & short-exp. noisy images (and ‘clean’ images).

Images & turbulence — 29

```
IDL> restore, 'cube.sav'
IDL> help
% At $MAIN$
CUBE            FLOAT     = Array[128, 128, 100]
Compiled Procedures:
    $MAIN$

Compiled Functions:

IDL> shortexp=cube[*,*,0]
IDL> print, total(shortexp)
     0.197022
IDL> shortexp=shortexp/total(shortexp)*100.
IDL> shortnoisy=addnoise(shortexp, /PHOT_NOISE)
% Compiled module: ADDNOISE.
```



```
IDL> tvscl, [shortexp,shortnoisy]^.5
IDL> longexp=total(cube,3)
IDL> longexp=longexp/total(longexp)*100.*100L
IDL> longnoisy=addnoise(longexp, /PHOT_NOISE)
IDL> tvscl, [longexp,longnoisy]^.5
```

<u>img formation w/noise:</u>

1- 'add' photon noise on one short-exp. PSF (in function of N...),
2- long-exp. PSF (100N photons!),
3- 'add' photon noise on the long-exp. PSF,
4- compare long-exp. & short-exp. noisy images (and 'clean' images).

```
---
REPORT
---

- Preliminary measures
+ introduction/context
+ PSD(r0, L0)
+ => influence of r0 and L0
+ rms(r0, L0)
+ => influence of r0 and L0
+ FWHM(r0 or lambda=>r0, L0)
+ => influence of r0 and L0
+ => comparison with the "seeing" lambda/r0
+ noisy images
```