

(IDL: other useful remarks)

- IDL help is called with: `IDL>> ?`
- `'?'` opens with a defined browser the file `'idl.htm'`
(`/usr/local/harris/idl89/help/online_help/Subsystems/idl/idl.htm`)
- Or also with the help of the unix command `'find'`:
`linux>> cd /`
`linux>> find . -name idl.htm`
- See also (for routines which are part of a third library):
`IDL>> doc_library, 'routine_name'`
- Return to main level after a crash: `IDL>> retall`
- Details on a var. `xxx`: `idl> help, xxx` (all var.: `idl> help`)
- Close last opened window: `idl> wdelete`
- Delete variable `xxx`: `idl> delvar, xxx`

Images & turbulence – 18

```
1 function wfgeneration, dim, length, L0, r0, lambda, SEED=seed
```

```
2 ;  
3 ; wave-front (wf) generation following von Karman model  
4 ; (infinite L0 -Kolmogorov model- not allowed here).  
5 ;
```

```
6 ; dim      = wf linear dimension [px],  
7 ; length  = wf physical length [m],  
8 ; L0      = wf outer-scale [m],  
9 ; seed    = random generation seed (OPTIONAL),  
10 ; r0      = Fried parameter at wavelength 'lambda' [m],  
11 ; lambda  = wavelength at which r0 is defined.  
12 ;
```

```
13 ; Marcel Carillet [marcel.carillet@unice.fr],  
14 ; lab. Lagrange (UCA, OCA, CNRS), Feb. 2013.  
15 ;
```

```
16 ; Last modification: March 2025.  
17 ;
```

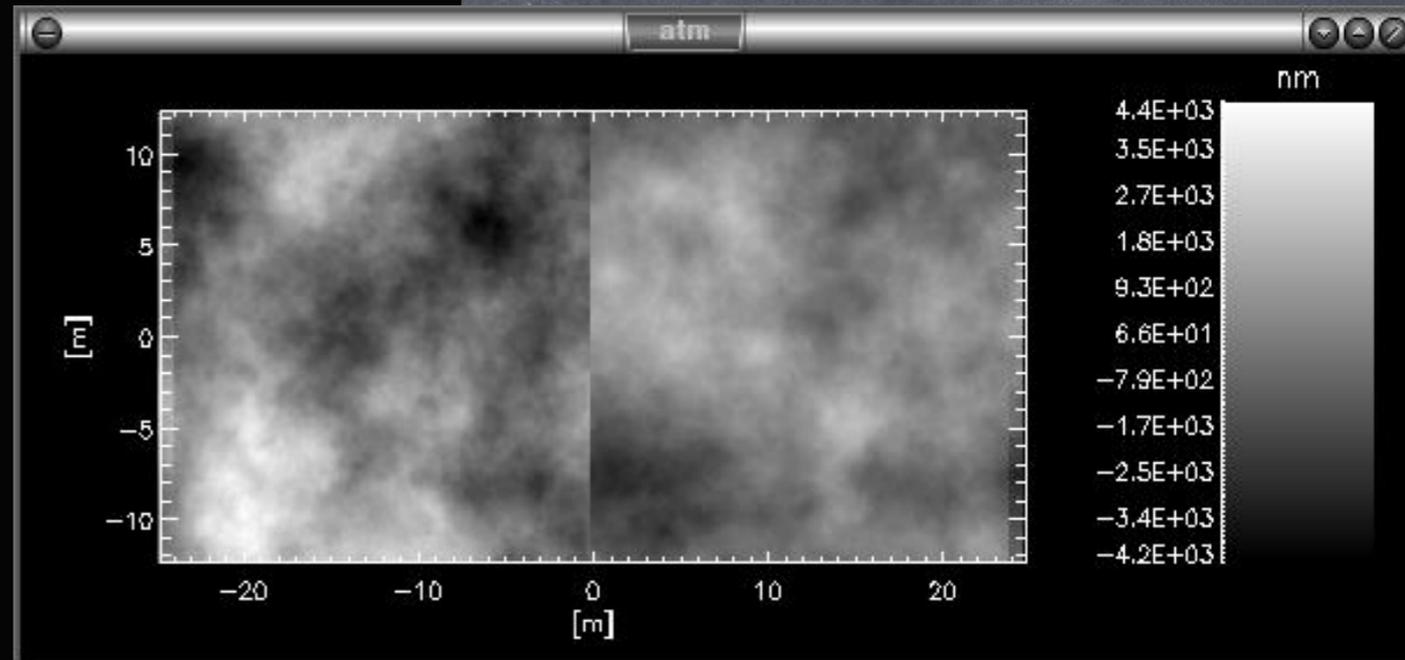
```
18 theta = (randomu(seed,dim,dim)-.5) * 2*!PI ; rnd uniformly distributed phase  
19 ; (=argument), between -PI and +PI,  
20 ; of the complex number describing  
21 ; the FFT of the phase screen phi  
22 freq  = dist(dim) ; spatial frequency array  
23 modul = sqrt(2)*sqrt(.0228)*(length/r0)^(5/6.)*(freq^2+(length/L0)^2)^(-11/12.)  
24 ; von Kármán model  
25 phi   = fft(modul*exp(complex(0,1)*theta), /INVERSE)
```

```
26 ;  
27 wf     = phi*lambda/(2*!PI)
```

```
28 wf     -= mean(wf)
```

```
29  
30 return, wf
```

```
31  
32 end
```



```
[IDL> .r wfgeneration  
; % Compiled module: WFGENERATION.  
[IDL> wf=wfgeneration(128, 2., 27., .1, 500E-9, SEED=seed)  
; % Compiled module: DIST.  
; % Loaded DLM: LAPACK.  
[IDL> wf1=float(wf)  
[IDL> wf2=imaginary(wf)  
[IDL> help, wf, wf1, wf2  
WF          COMPLEX      = Array[128, 128]  
WF1         FLOAT        = Array[128, 128]  
WF2         FLOAT        = Array[128, 128]  
[IDL> tvscl, [wf1,wf2]  
% Program caused arithmetic error: Floating overflow
```

Images & turbulence – 19

wf generation: generate a cube of statistically independent wf (e.g 100)

=> compute mean *rms* for different $[r_0, L_0]$

```
1 function wfcube2, dim, length, L0, r0, lambda, n_wf, filewf
2
3 ;+
4 ; example of use:
5 ; dim      = 128L      ; [px] wf dimension
6 ; length   = 2.        ; [m] wf physical dimension
7 ; L0       = 27.       ; [m] outerscale of turbulence
8 ; r0       = .1        ; [m] Fried parameter
9 ; lambda   = 500E-9    ; [m] r0 wavelength
10 ; n_wf     = 100L     ; nb of generated wf
11 ; filewf   = 'cube.sav' ; cube of wf filename
12 ;
13 ; print, wfcube2(128L, 2., 27., .1, 500E-9, 100L, 'wf_r0=10cm_L0=10m.sav')*1E9
14 ; -> compute the cube of wf, save it, and print the rms value in nm
15 ;
16 ; sub-routines needed:
17 ; wfgeneration.pro, compute_rms.pro
18 ;
19 ; Marcel Carillet [marcel.carillet@unice.fr],
20 ; lab. Lagrange (UCA, OCA, CNRS), Feb. 2018.
21 ; Last modification: 11th March 2024
22 ;-
23
24 ; preliminary
25 cube = fltarr(dim,dim,n_wf) ; initialize cube of wf
26
27 ; compute and save cube of wf
28 for i=0, n_wf/2-1 do begin ; generate wf
29     wf = wfgeneration(dim, length, L0, r0, lambda, SEED=seed)
30     cube[*,*,2*i] = float(wf)
31     cube[*,*,2*i+1] = imaginary(wf)
32 endfor
33 save, cube, FILE=filewf ; save cube of wf to disk
34
35 ; compute mean rms
36 rms = compute_rms(cube) ; compute rms
37
38 return, rms ; return back
39 end
```

```
[IDL> .r wfcube2
% Compiled module: WFCUBE2.
[IDL> print, wfcube2(128L, 2., 27., .1, 500E-9, 100L, 'wf_r0=10cm_L0=10m.sav')*1E9
368.186
% Program caused arithmetic error: Floating underflow
IDL>
```

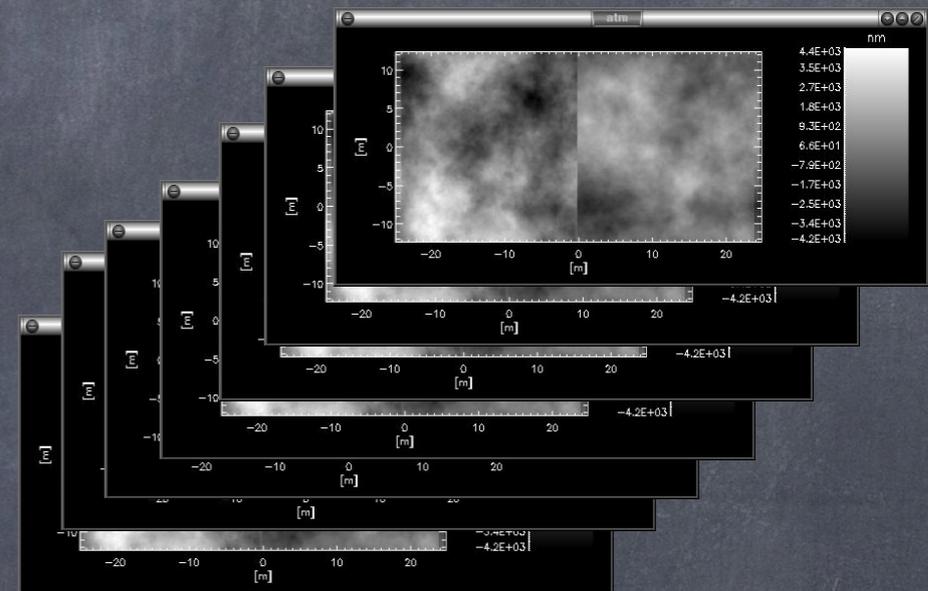
```
function compute_rms, cube
; cube: cube of wavefronts (square wf, no pupil!)

n_wf = (size(cube))[3]
rms = fltarr(n_wf)

for i=0,n_wf-1 do begin
    toto = moment(cube[*,*,i], SDEV=dummy)
    rms[i] = dummy
endfor

rms_moy = mean(rms)

return, rms_moy
end
```



Logbook "Imaging through turbulence" (M1 MAUCA)

- Preliminary measures

- + introduction/context
- + PSD(r_0 , L_0)
- + => influence of r_0 and L_0
- + rms(r_0 , L_0)

(more to come...)

Images & turbulence - 20



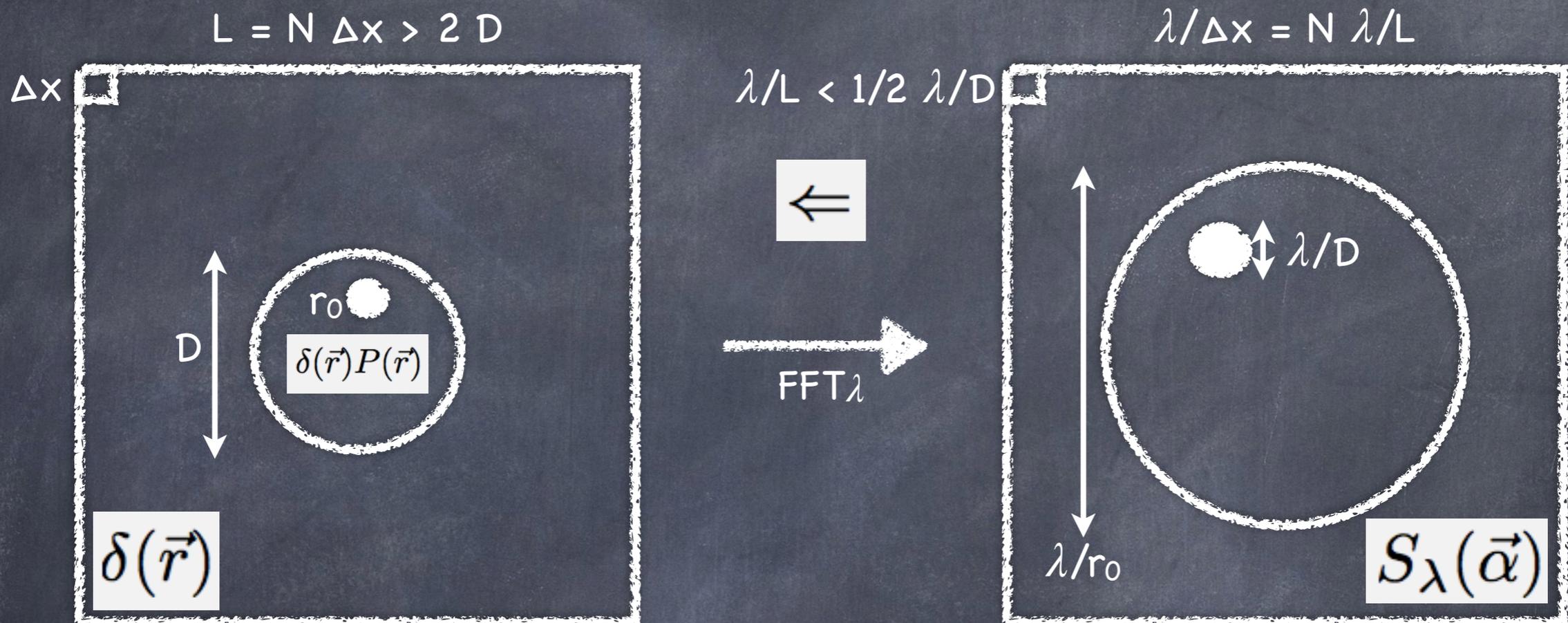
$$\Psi(\vec{r}) = A \exp(i\Phi(\vec{r}))$$

$$P(\vec{r}) \Rightarrow A P(\vec{r}) \exp(i\Phi(\vec{r})P(\vec{r}))$$

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

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

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 $\lambda/\Delta x$ ($\Rightarrow N$ large enough)

Images & turbulence - 22

```
function wfimg2, diam, obs, lambda_psf, filewf, filepsf
;+
; example of use:
; diam      = 64L           ; [px] telescope pupil dimension
; obs       = 0. [0-1]     ; (linear) obscuration ratio
; lambda_psf= 500E-9       ; [m] PSF wavelength
; filewf    = 'cube.sav'   ; cube of wf filename
; filepsf   = 'cube_psf.sav'; cube of PSFs filename
; print, wfimg2(diam,obs,lambda_psf,filewf,filepsf)
; -> compute the cube of PSFs, save it, and tell how it went
;
; sub-routines needed: make_PSF.pro, wfgeneration.pro, makepup.pro
;
; Marcel Carillet [marcel.carillet@unice.fr], Lagrange (UniCA, OCA, CNRS)
; written: Feb. 2018, last modified: March 11th 2024.
;-

; preliminaries
restore, filewf           ; restores variable 'cube' containing nn wf
dim= (size(cube))(1)     ; linear size of wf
nn = (size(cube))(3)     ; nb of wf
cube_psf=fltarr(dim,dim,nn) ; initialize cube of PSFs

; compute and save PSFs
pup = makepup(dim,diam,obs) ; compute entrance pupil
for i=0, nn-1L do cube_psf[*,* ,i] = make_PSF(pup,cube[*,* ,i],lambda_psf)
; compute the PSF corresponding to each wf
save, cube_psf, FI=filepsf ; save cube of PSFs to disk

; return back
return, 'Cube of PSFs '+filepsf+' saved on disk...'
end
```

image formation:

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

```
function make_PSF, pup, wf, lambda
;+
; PSF computation from a wavefront
;
; pup      = input pupil,
; wf       = input wavefront [float],
; lambda   = wavelength at which PSF is computed.
; PSF = make_PSF(pup, wf, lambda)
; -> compute the PSF corresponding to wf and pup, at wavelength lambda
;
; Marcel Carillet [marcel.carillet@unice.fr],
; UMR 7293 Lagrange (UNS/CNRS/OCA), Feb. 2013.
; Last modification: February 25th 2026 (normalize PSF)
;-

; preliminary
dim = (size(wf))[1]

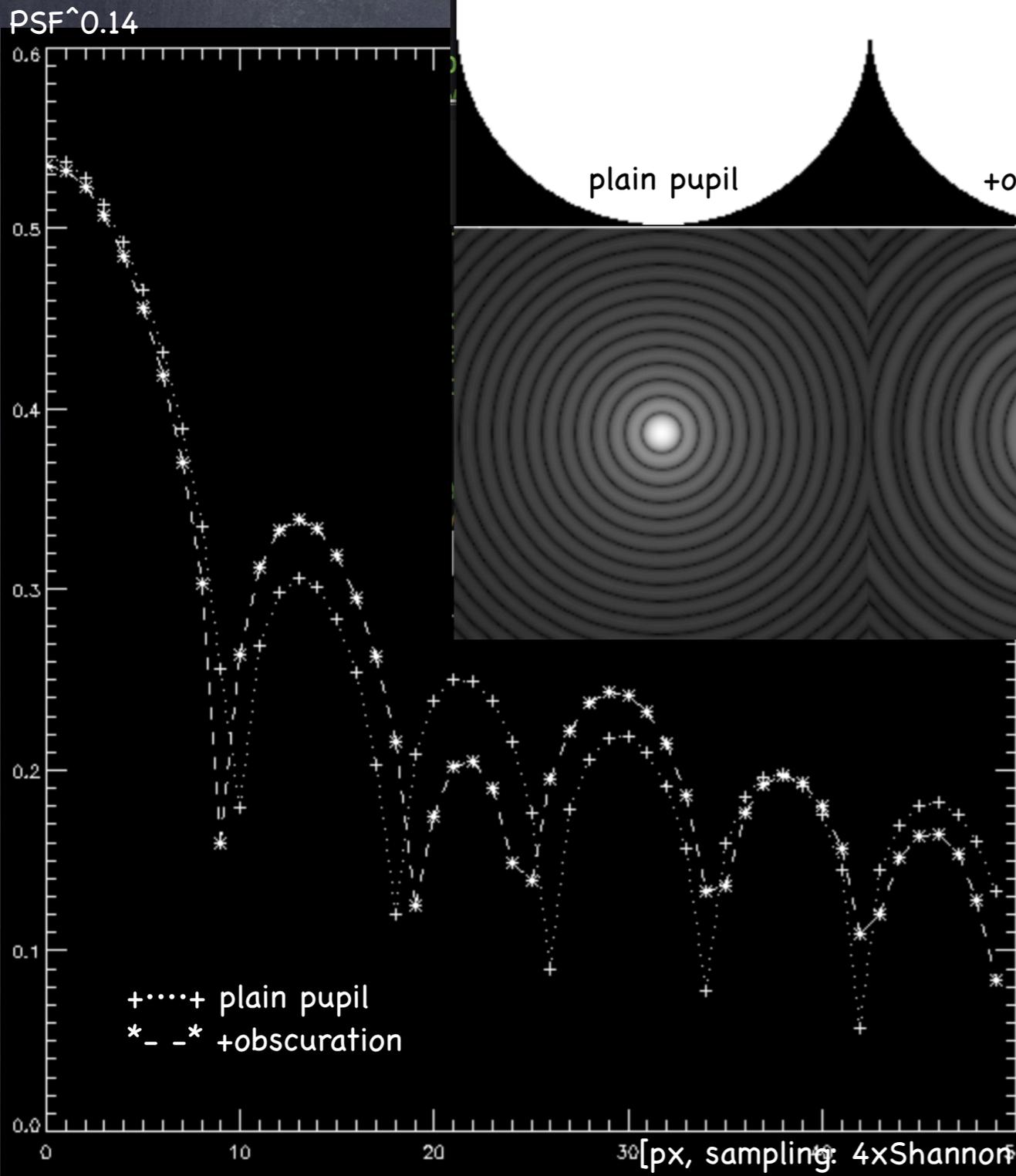
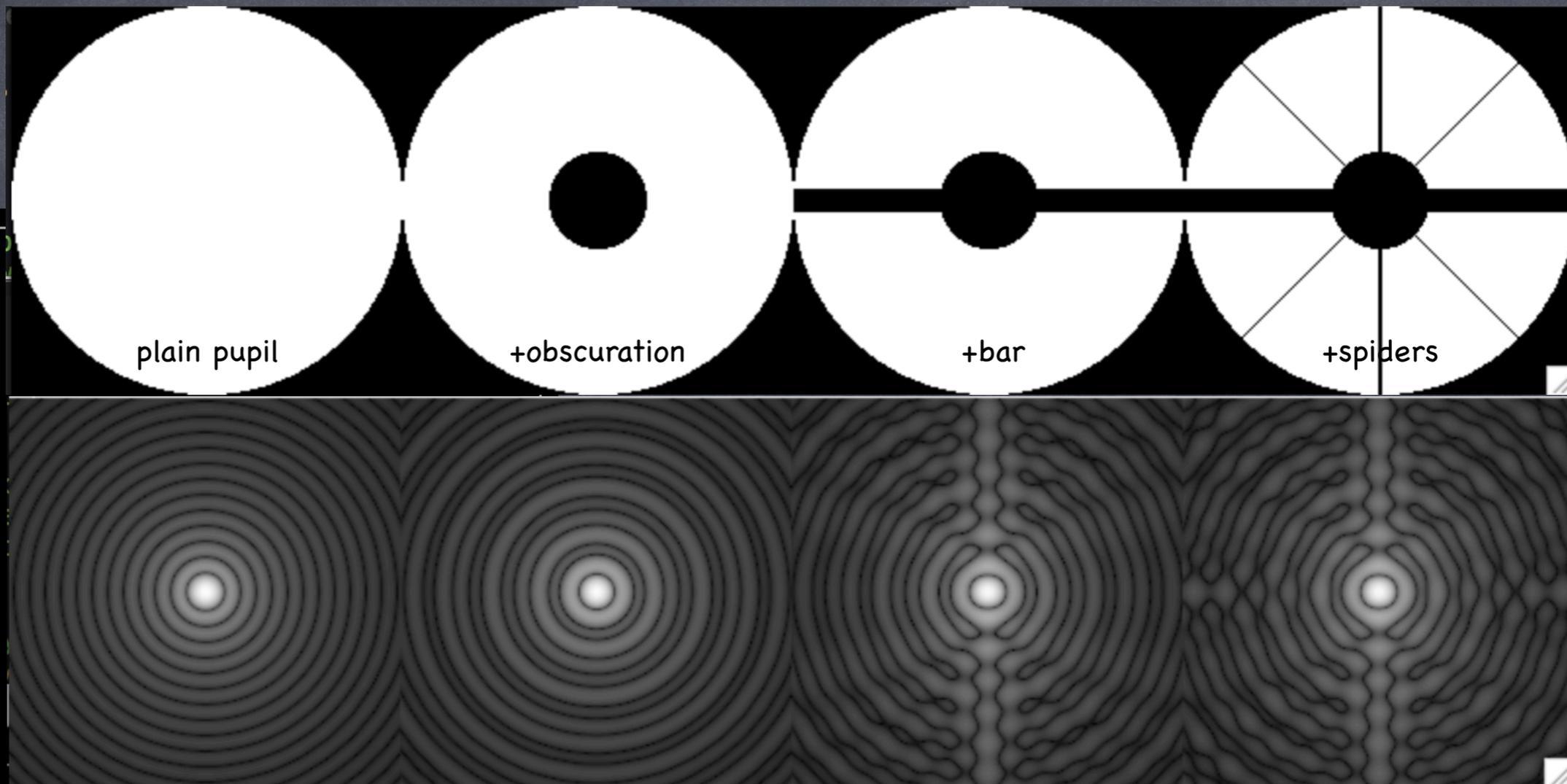
; compute PSF
psf = (abs(fft(pup*exp(complex(0,1)*2*!PI/lambda*wf*pup))))^2
; NB: (abs(fft(pup*exp(complex(0,1)*2*!PI/lambda*wf))))^2 would suffice
psf = shift(psf, dim/2, dim/2)

; normalize it
psf/= total(psf)

; return back
return, psf
end
```

```
IDL> .r wfimg2
% Compiled module: WFIMG2.
IDL> print, wfimg2(64L, 0., 500E-9, 'wf_r0=10cm_L0=10m.sav', 'PSF_r0=10cm_L0=10m_lambda=500nm.sav')
Cube of PSFs PSF_r0=10cm_L0=10m_lambda=500nm.sav saved on disk...
```

Images & turbulence - 23



(influence of the pupil shape, no aberration)

zero-padding (200x200-px pupil within 1600x1600-px screen)