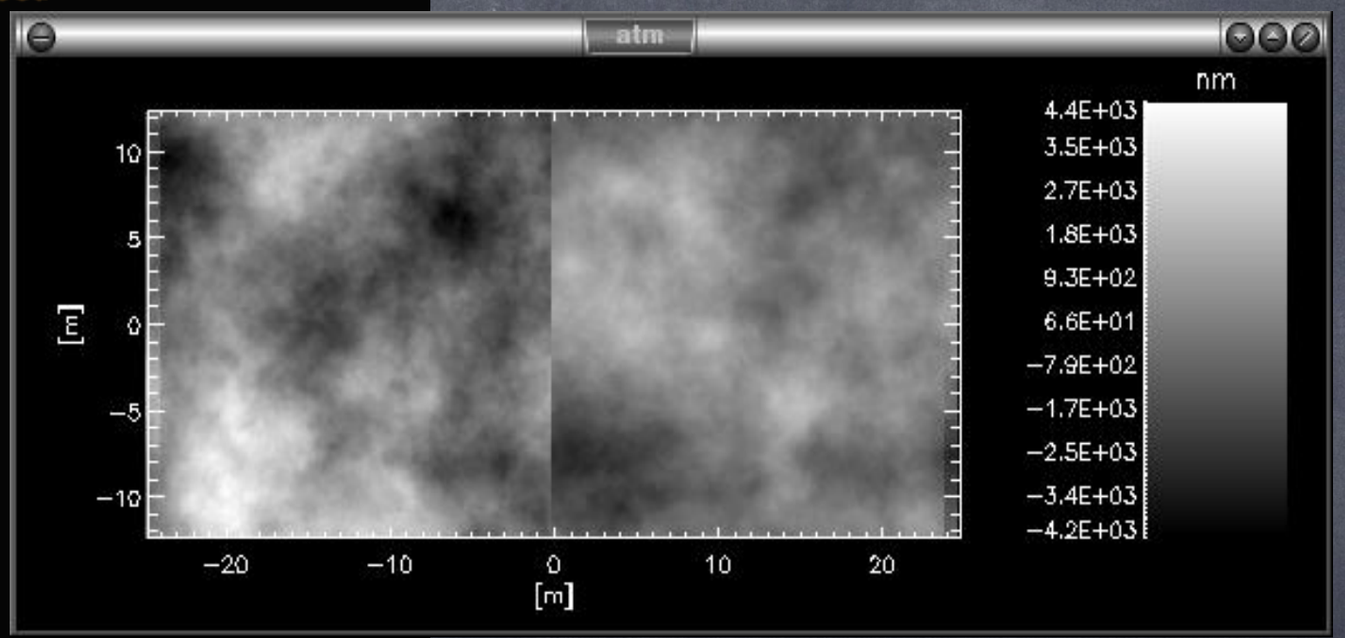
Images & turbulence — 18

```
function wfgeneration, dim, length, L0, r0, lambda, SEED=seed
;
; wave-front (wf) generation following von Karman model
; (infinite L0 -Kolmogorov model- not allowed here).
;
; dim    = wf linear dimension [px],
; length = wf physical length [m],
; L0     = wf outer-scale [m],
; seed   = random generation seed (OPTIONAL),
; r0     = Fried parameter at wavelength 'lambda' [m],
; lambda = wavelength at which r0 is defined.
;
; Marcel Carbillet [marcel.carbillet@unice.fr],
; lab. Lagrange (UCA, OCA, CNRS), Feb. 2013.
;
; Last modification: March 2025.
;
theta = (randomu(seed,dim,dim)-.5) * 2*!PI    ; rnd uniformly distributed phase
                                              ; (=argument), between -PI and +PI,
                                              ; of the complex number describing
                                              ; the FFT of the phase screen phi
freq  = dist(dim)                             ; spatial frequency array
modul = sqrt(2)*sqrt(.0228)*(length/r0)^(5/6.)*(freq^2+(length/L0)^2)^(-11/12.)
                                              ; von Kármán model
phi   = fft(modul*exp(complex(0,1)*theta), /INVERSE)
                                              ;
wf    = phi*lambda/(2*!PI)                    ;
wf   -= mean(wf)                             ;
                                              ;
return, wf                                    ;
                                              ;
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]

```
function wfcube2, dim, length, L0, r0, lambda, n_wf, filewf

;+
; example of use:
; dim       = 128L          ; [px] wf dimension
; length    = 2.            ; [m] wf physical dimension
; L0        = 27.           ; [m] outerscale of turbulence
; r0        = .1            ; [m] Fried parameter
; lambda    = 500E-9        ; [m] r0 wavelength
; n_wf      = 100L          ; nb of generated wf
; filewf    = 'cube.sav'    ; cube of wf filename
;
```

```
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>
```

```
;-

; preliminary
cube = fltarr(dim,dim,n_wf) ; initialize cube of wf

; compute and save cube of wf
for i=0, n_wf/2-1 do begin  ; generate wf
    wf = wfgeneration(dim, length, L0, r0, lambda, SEED=seed)
    cube[*,*,2*i]   = float(wf)
    cube[*,*,2*i+1] = imaginary(wf)
endfor
save, cube, FILE=filewf     ; save cube of wf to disk

; compute mean rms
rms = compute_rms(cube)     ; compute rms

return, rms                 ; return back
end
```

```
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
```

(IDL stuff — 4)

```
; call with: IDL> @Exo2
Diam  =1.0
r0    =0.3
N     = 10

J = (N+1)*(N+2)/2-1
Noll = .2944*J^(-sqrt(3)/2)*(Diam/r0)^(5./3)
S = exp(-Noll)
; see result with: IDL> print, S
```

batch: all variables are accessible.

```
; call with: IDL> .rn Exo2_main
Diam  =1.0
r0    =0.3
N     = 10

J = (N+1)*(N+2)/2-1
Noll = .2944*J^(-sqrt(3)/2)*(Diam/r0)^(5./3)
S = exp(-Noll)

end
; see result with: IDL> print, S
```

main: idem (« .r » : run ; « .rn » : run new).

```
; call with: IDL> .rn Exo2_proc
;             IDL> Exo2_proc, Diam, r0, N, S
; with, e.g: Diam=1.0, r0=0.3, N=10, S undefined
pro Exo2_proc, Diam, r0, N, S

J = (N+1)*(N+2)/2-1
Noll = .2944*J^(-sqrt(3)/2)*(Diam/r0)^(5./3)
S = exp(-Noll)

end
; see result with: IDL> print, S
```

procedure: (input/output) parameters are accessible, but variables defined within the procedure are not.

```
; call with: IDL> .rn Exo2_func
;             IDL> print, Exo2_func(Diam, r0, N)
; with, e.g: Diam=1.0, r0=0.3, N=10
function Exo2_func, Diam, r0, N

J = (N+1)*(N+2)/2-1
Noll = .2944*J^(-sqrt(3)/2)*(Diam/r0)^(5./3)
S = exp(-Noll)

return, S
end
```

function: no output parameters, inside variables not accessible, result of the function returned.

(IDL stuff — 5)

- IDL help is called with: IDL>> ?
- '?' opens with a defined browser the file 'idl.htm', which can be also found directly somewhere like:

  .../harris/idl89/help/online_help/Subsystems/idl/idl.htm
- Or also with the help of the unix command 'find':

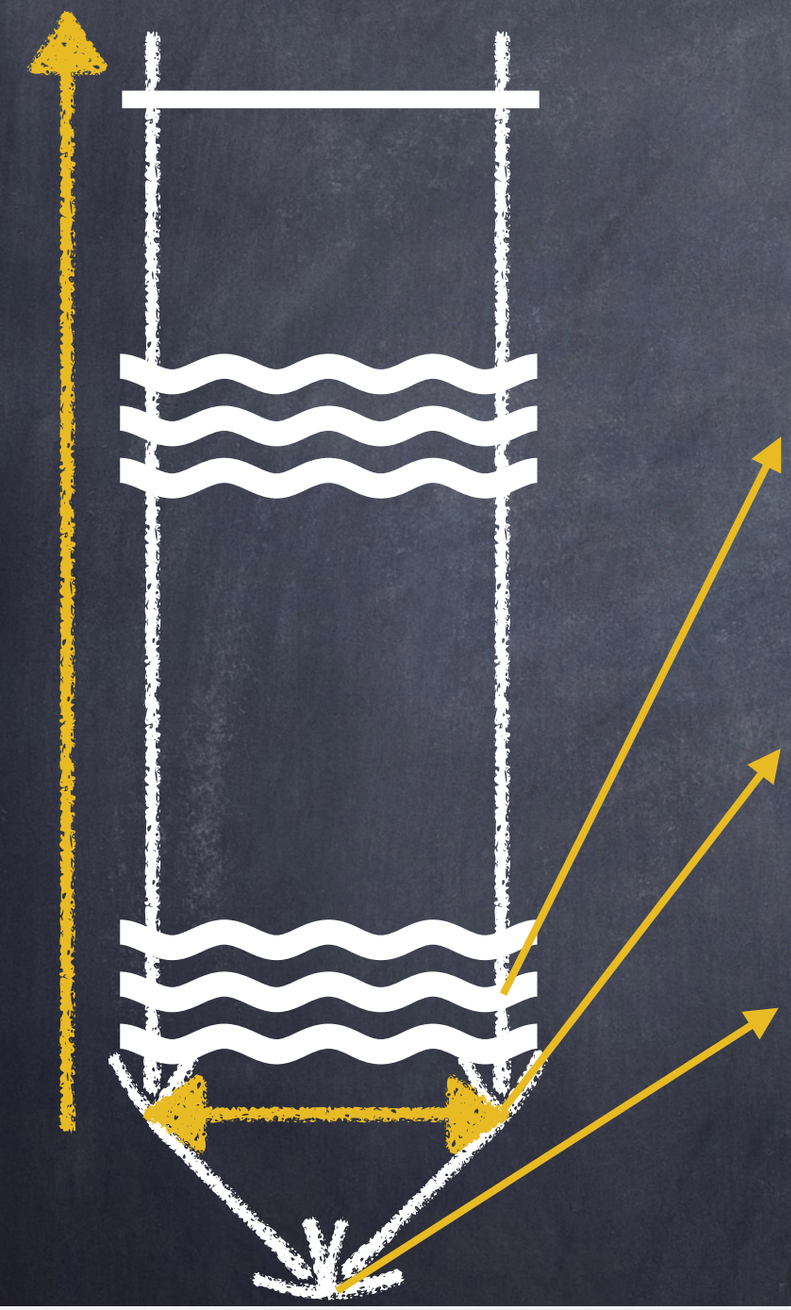
  unix>> cd /

  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**

  (all variables: **idl> help**)
- Close last opened window: **idl> wdelete**

Images & turbulence — 20



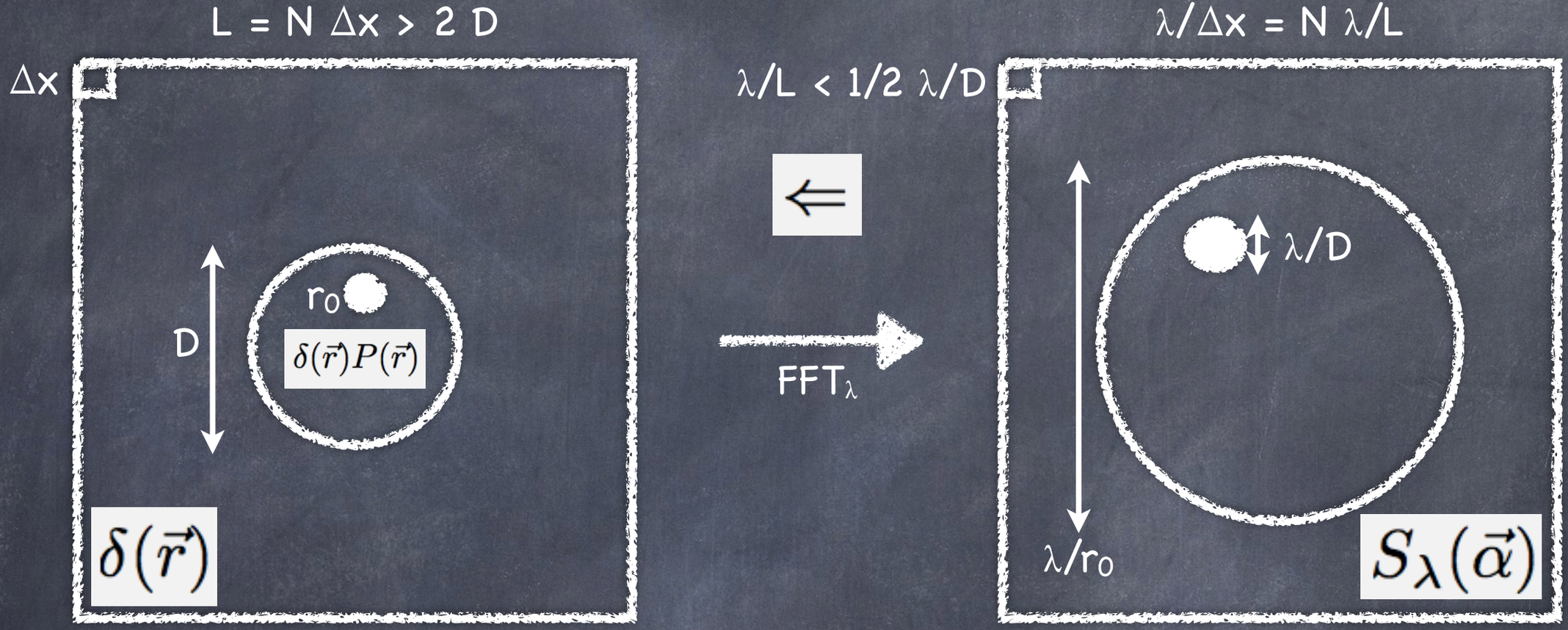
$$\Psi(\vec{r}) = A \ \exp\left(\imath\Phi(\vec{r})\right)$$

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

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

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

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 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 Carbillet [marcel.carbillet@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 Carbillet [marcel.carbillet@unice.fr],
; UMR 7293 Lagrange (UNS/CNRS/OCA), Feb. 2013.
; Last modification: March 11th 2024
;-

; 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)

; 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...
```