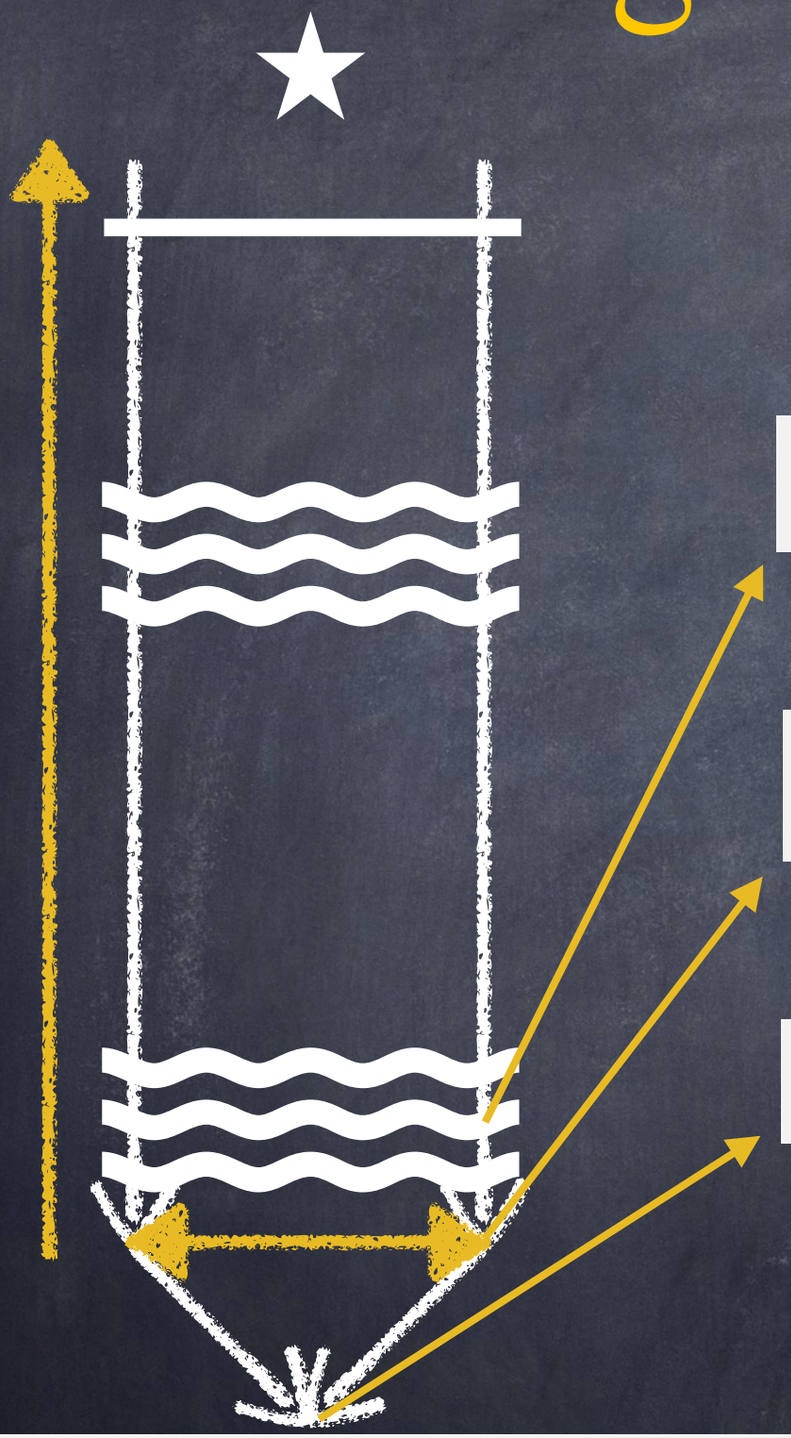
Images & turbulence - 11



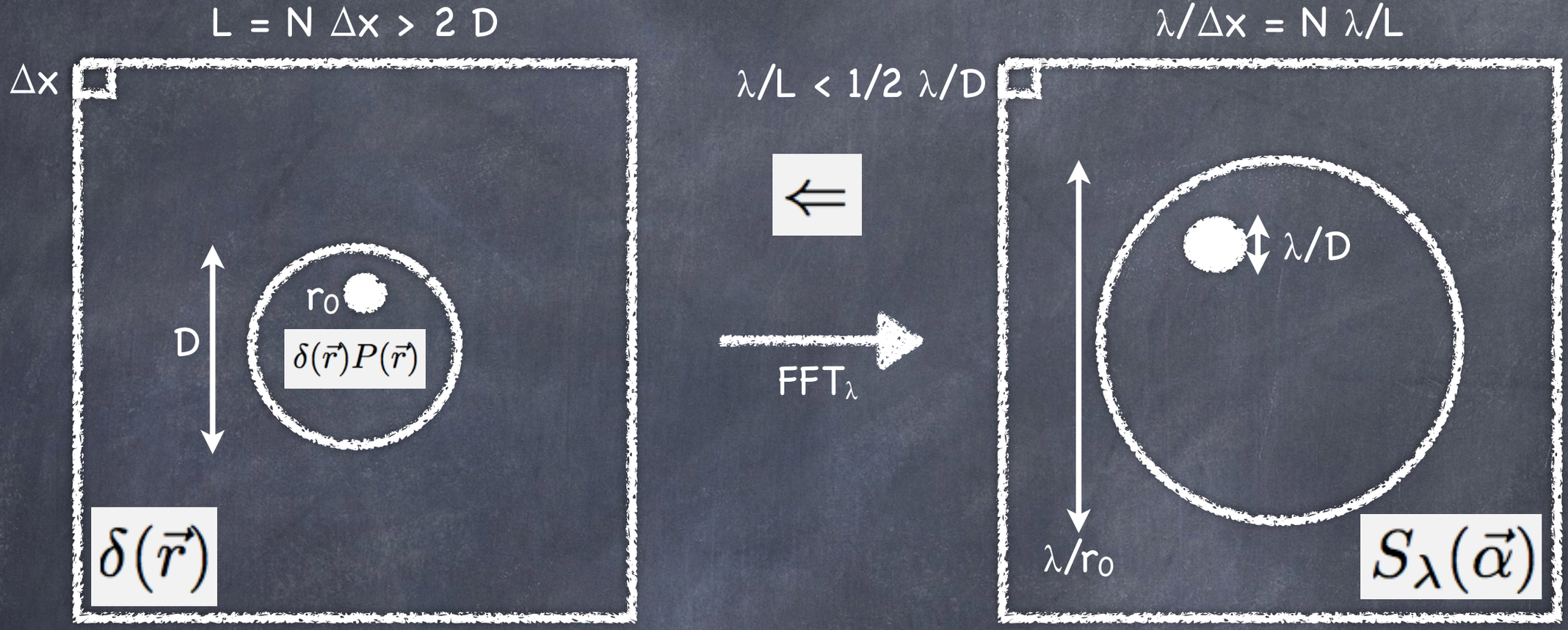
$$\Psi(\vec{r}) = A \ \exp\left(\imath\Phi(\vec{r})\right)$$

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

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

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

Images & turbulence - 12



Shannon (=Nyquist)

=> pixel image λ/L : au plus la moitié de l'élément de résolution (resel!) λ/D

(en d'autres termes : on doit avoir AU MOINS 2 pixels par λ/D dans l'image)

=> le front d'onde simulé doit être d'au moins 2 fois le diamètre du télescope (L>2D)

De plus

- λ/r_0 doit être plus petit que $\lambda/\Delta x$ (=> N assez grand)

Images & turbulence - 13

```
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>formation d'image :</u>
1- cube de PSF instantanées ($2\,\lambda$)

```
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), February 2013.
;
; Last modification: Feb. 2014
;
dim = (size(wf))[1]
img = (abs(fft(pup*exp(complex(0,1)*2*!PI/lambda*wf*pup))))^2
img = shift(temporary(img), dim/2, dim/2)

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...
```

Images & turbulence - 14

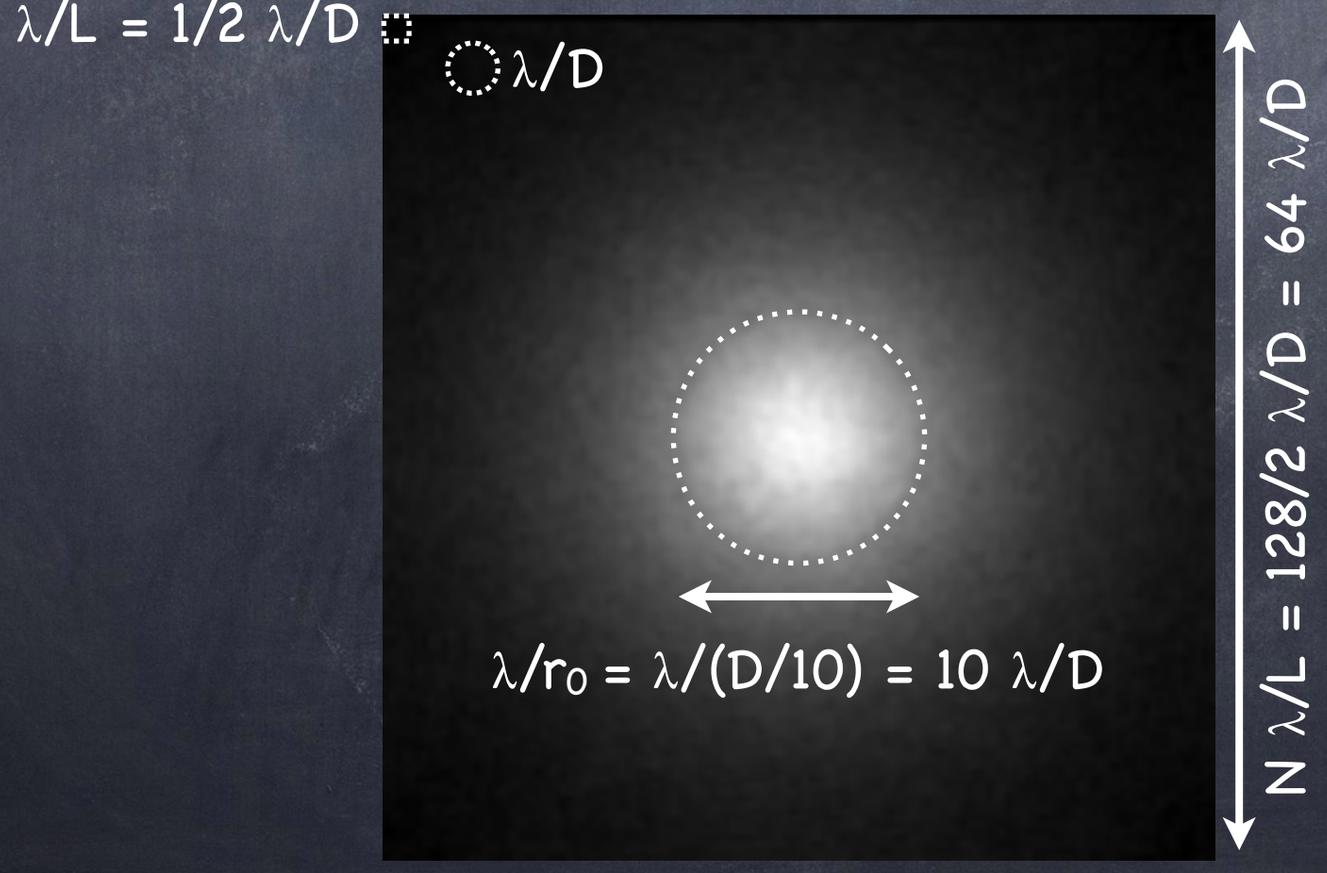
```
IDL> restore, 'cube.sav'
IDL> help
% At $MAIN$
CUBE            FLOAT     = Array[128, 128, 100]
Compiled Procedures:
     $MAIN$

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
```

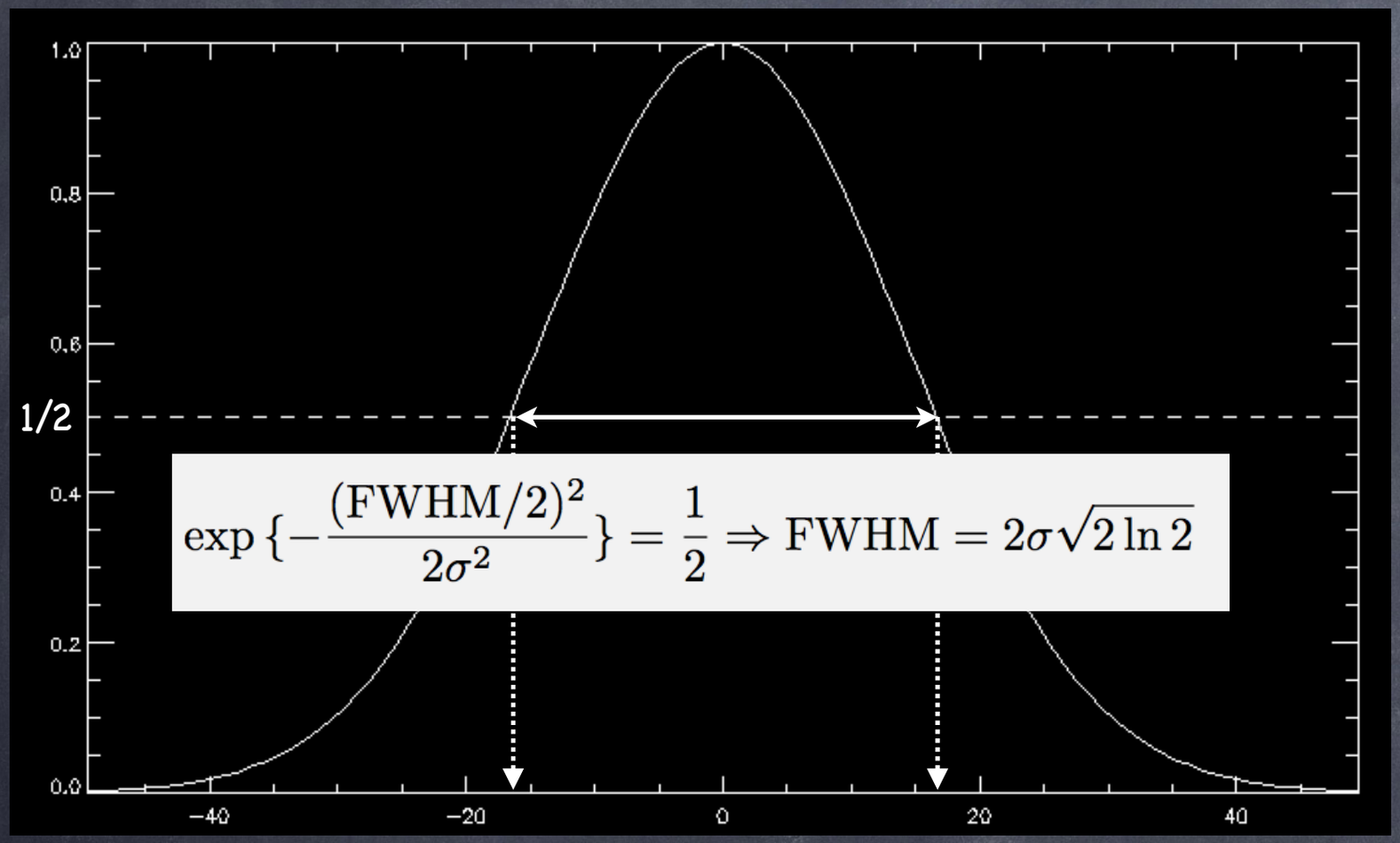
<u>formation d'image :</u>

1- cube de PSF instantanées ($2\,\lambda$)

2- PSF longue pose



Images & turbulence - 15



<u>formation d'image :</u>
1- cube de PSF instantanées (2 λ)
2- PSF longue pose
3- ajuster avec une gaussienne et comparer FWHM et λ/r_0 (seeing), en fonction aussi de L_0.

-> Lire aussi 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
```

Dans cet exemple, la FWHM est ~15.54px et, puisqu'on a : 1px=(λ/D)/2, on a donc : FWHM~7.77 (λ/D)

```
12 + formation de l'image et calcul de FWHM(r0 ou lambda, év. L0)
13 + => ccl sur l'influence de r0 ou lambda (et év. L0)
14 + => comparaison avec le 'seeing' lambda/r0
```

Images & turbulence - 16

-> Bruits de détection

• Au tout début : bruit de photon (*photon noise* ou *shot noise*), poissonien, transformation de l'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) = probabilité de détecter *n* photons quand N sont attendus

Pour N grand : ~Gaussien…

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

Images & turbulence - 17

-> Bruits de détection

• Au tout début : bruit de photon (*photon noise* ou *shot noise*), poissonien, transformation de l'image.

• Tout à la fin : bruit de lecture (*read-out noise, RON*), gaussien de moyenne nulle et d'écart-type σ_e [e-/px], bruit additif.

• Entre les deux : bruit de courant d'obscurité (*dark current noise*), bruit d'amplification (*amplification noise*) et *exotic dark current noise* dans le cas des EMCCD, bruit dû à l'étalonnage (*calibration*) de champ plat (*flat field*), bruit 'poivre et sel' (*'salt & pepper' noise, i.e.* pixels 'chauds' et 'froids'), etc.

Images & turbulence - 18

```
;; 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
```

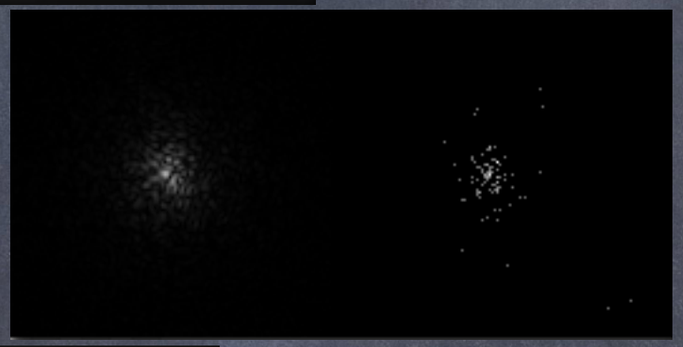
<u>images bruitées :</u>

1- ajouter le bruit de photon sur une PSF instantannée (fct de N)
2- PSF longue pose (100N photons!)
3- ajouter le bruit de photon sur longue pose
4- comparer images bruitées longue pose et courte pose…

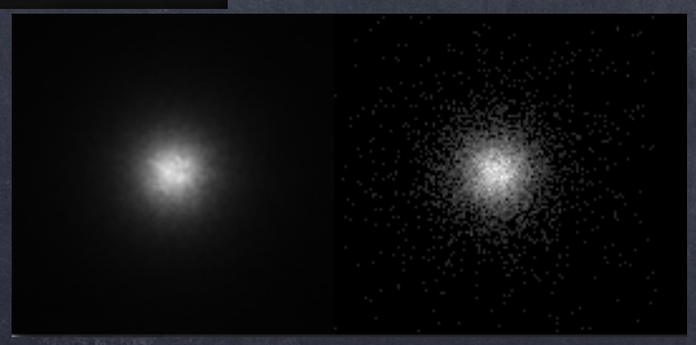
```
; CALLING SEQUENCE:
;    noisy_image = addnoise(input_image,              $
;                           PHOT_NOISE=phot_noise,    $
;                           SIGMA_DARK=sigma_dark,    $
;                           DELTA_T=delta_t,          $
;                           EXODARK=exodark,        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

Images & turbulence - 19

```
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
```



+ formation de l'image en présence de bruit

<u>images bruitées :</u>

1- ajouter le bruit de photon sur une PSF instantannée (fct de N)
2- PSF longue pose (100N photons!)
3- ajouter le bruit de photon sur longue pose
4- comparer images bruitées longue pose et courte pose…