

## **PjE – Projet Encadré IVI**

### **Semaine 4 : filtrage d'image et compléments sur OpenCV**

**Master ASE** : <http://master-ase.univ-lille1.fr>  
**Master Informatique** : <http://www.fil.univ-lille1.fr>  
**Spécialité IVI** : <http://master-ivi.univ-lille1.fr>

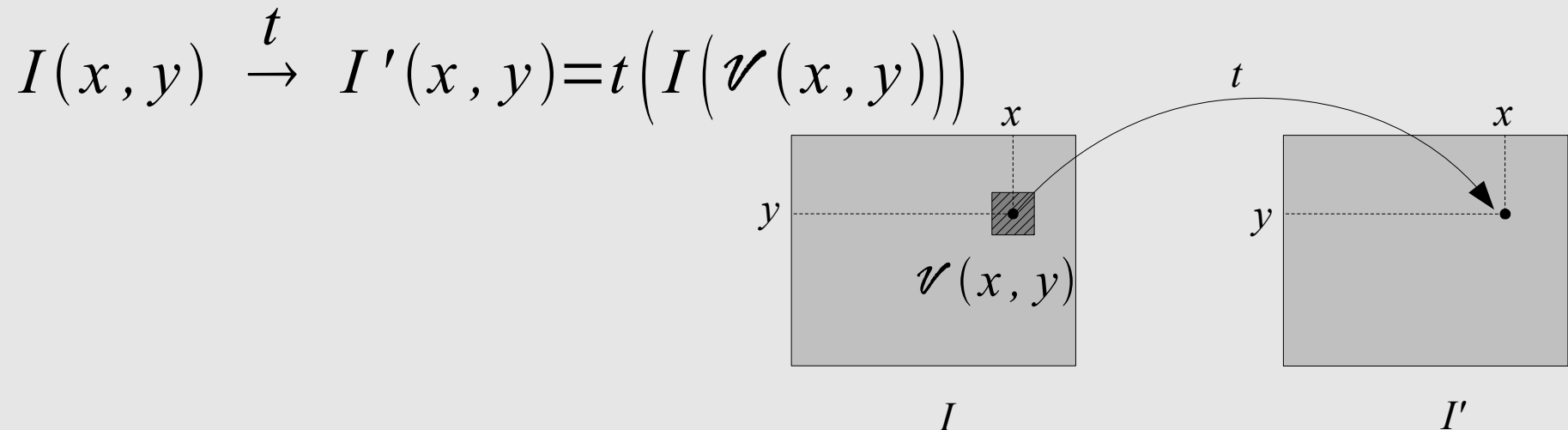
## Plan du cours

- **1 – Transformations de voisinage**
  - Principe de la transformation et voisinages
  - Filtrage par convolution 2D
  - Filtres de lissage
  - Filtres morphologiques
- **2 – Filtrer avec OpenCV**
  - Filtres de convolution
  - Filtres morphologiques
- **3 – Compléments sur OpenCV**
  - Histogrammes : structure et création
  - Dessiner sur une image

## Principe des transformations de voisinage (1/2)

### • Principe

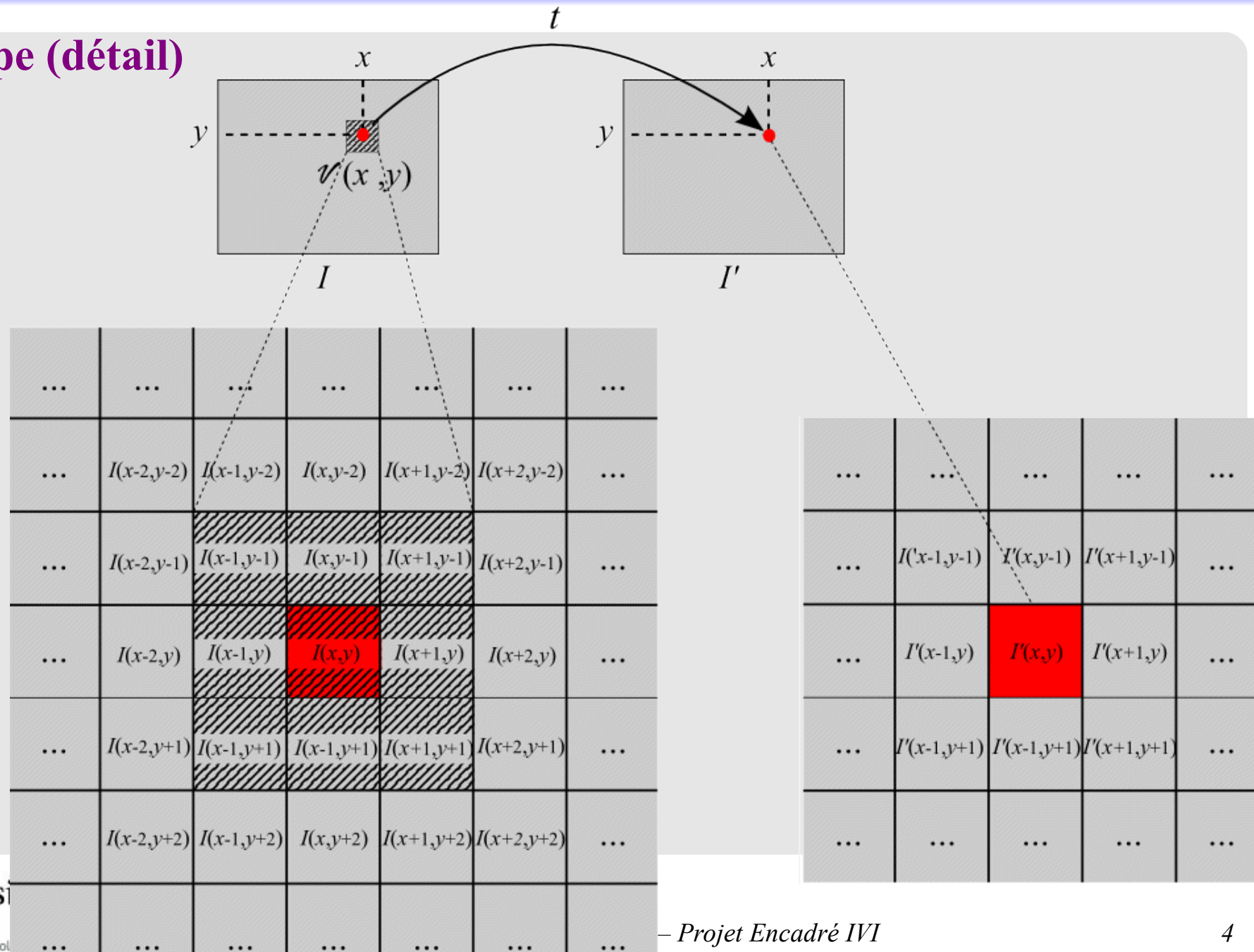
- ➔ Pour calculer la valeur du pixel de coordonnées  $(x,y)$  dans l'image résultat  $I'$ , on utilise, dans l'image initiale, non seulement la valeur du pixel  $I(x,y)$  mais aussi celles des pixels situés dans un **voisinage** de ce dernier  $I(\mathcal{V}(x,y))$ .



- ➔  $I'$  a même taille que  $I$ , mais des propriétés plus intéressantes.
- ➔ En chaque pixel  $P(x,y)$  considéré, le voisinage  $\mathcal{V}$  est défini de manière identique (« forme » identique) mais relativement à lui ( $\mathcal{V}(x,y)$ ).

# Principe des transformations de voisinage (2/2)

## • Principe (détail)



# Notion de voisinage

## • Voisinage $\mathcal{V}$ d'un pixel $P(x,y)$

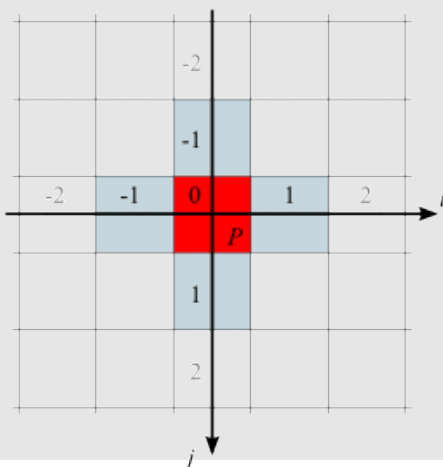
### → Hypothèses

- $\mathcal{V}$  est centré en  $P$  ;
- les pixels sont disposés selon une **maille** carrée (**treillis**).

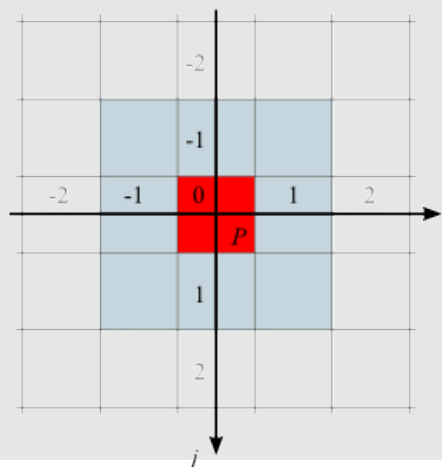
→ **Définition** :  $\mathcal{V}(P)$  est l'ensemble des pixels situés à moins d'une certaine distance de  $P$ .

→ La forme du voisinage (et le nombre de voisins) de  $P$  dépendent de la distance considérée.

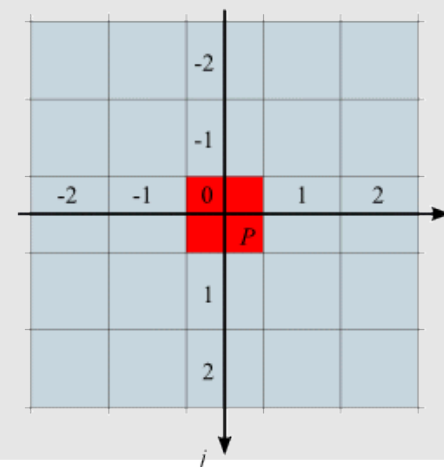
→ Voisinages les plus usités en traitement d'images :



4-voisinage



8-voisinage ou « voisinage 3x3 »



« voisinage 5x5 »

Notations en **coordonnées relatives** si  $Q$  est voisin de  $P(x,y)$ , alors  $Q(x+i,y+j)$  avec  $i,j$  entiers relatifs.

Ici ne sont représentées que les coordonnées  $i,j$

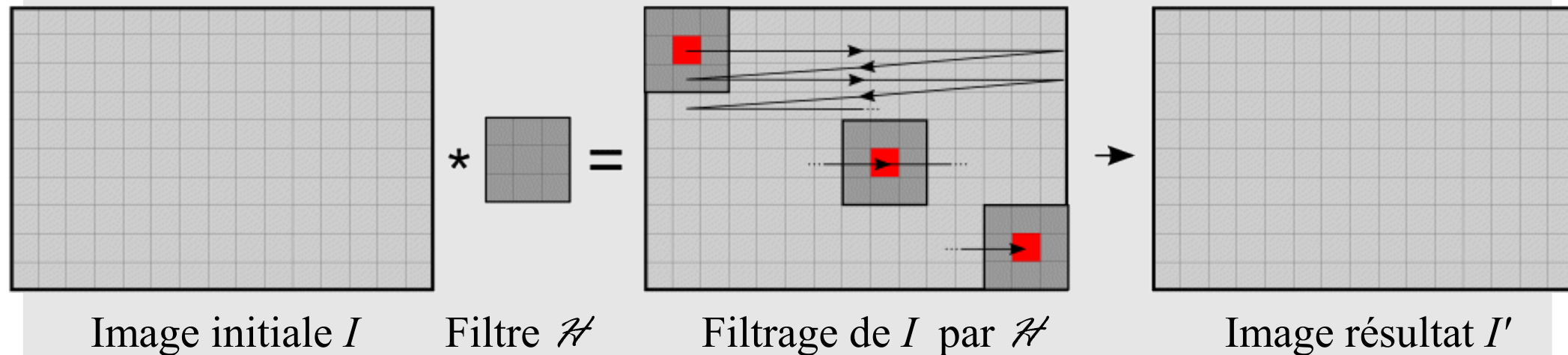
# Filtrage

- **Principe**

- déplacer le filtre sur chacun des pixels ( $P$  = pixel d'*analyse*) de l'image  $I$
- remplacer le niveau en  $P$  par une valeur dépendant des niveaux de ses voisins
- le nombre de voisins considérés dépend de la taille du filtre

- **2 principaux types de filtres** : le niveau en  $P$  est remplacé par

- une combinaison linéaire des niveaux voisins  
(filtre **linéaire**, réalisable par **convolution**)



- une transformation non linéaire des niveaux voisins (filtre **non linéaire**).

## Filtrage par convolution discrète 2D (1/2)

### • Principe

- L'image  $I$  est une fonction de 2 variables discrètes ( $x$  et  $y$ ).
- Le filtre de convolution  $\mathcal{H}$  appliqué sur  $I$  est lui aussi à 2D (matrice).
- $\mathcal{H}$  est appelé **filtre**, **masque**, **noyau** ou **fenêtre** de convolution.

- Souvent *carré* et de taille *impaire* (3x3, 5x5, ...), pour être centré sans ambiguïté sur le pixel d'analyse.
- Souvent à valeurs *symétriques* par rapport à l'élément central :  $h_{-1,-1}=h_{+1,+1}$ ,  $h_{0,-1}=h_{0,+1}$ ,  $h_{+1,-1}=h_{-1,+1}$ , ...
- Souvent à *somme unité* (**normalisé**) pour conserver la luminance de l'image.

$$\mathcal{H} = \begin{bmatrix} h_{-1,-1} & h_{0,-1} & h_{+1,-1} \\ h_{-1,0} & h_{0,0} & h_{0,-1} \\ h_{-1,+1} & h_{0,+1} & h_{+1,+1} \end{bmatrix}$$

### • Formule

- L'image  $I'$  résultat de la convolution (notée  $*$ ) de  $I$  par  $\mathcal{H}$  est donnée par :

$$I'(x, y) = (I * \mathcal{H})(x, y) = \sum_i \sum_j I(x-i, y-j) \cdot \mathcal{H}(i, j)$$

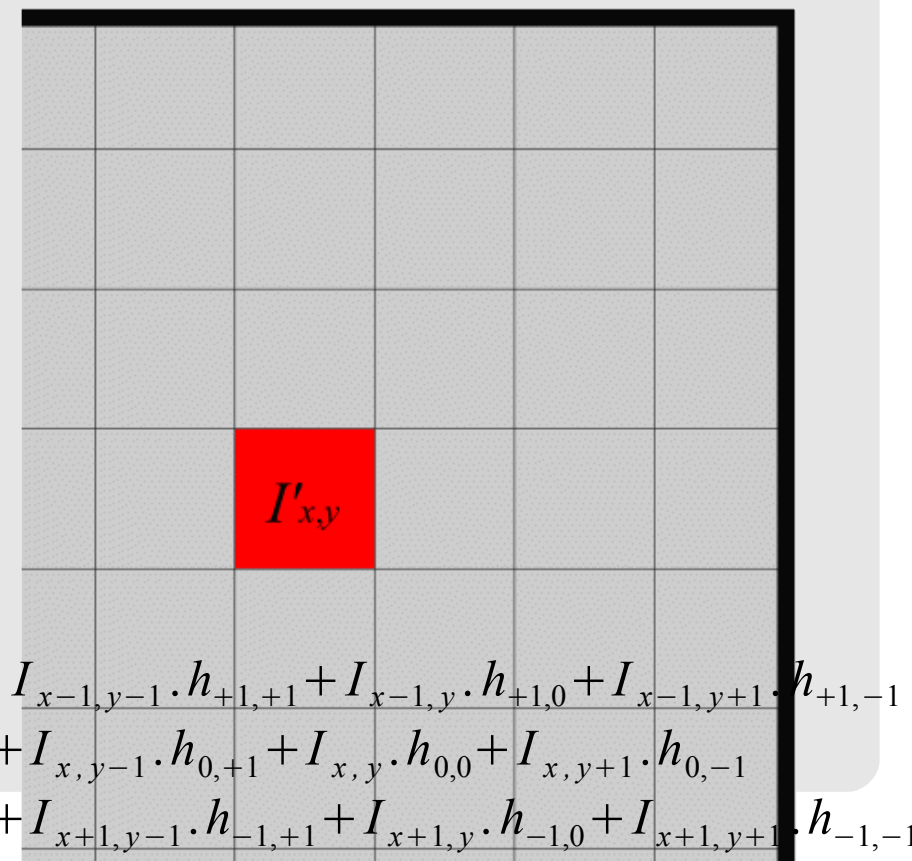
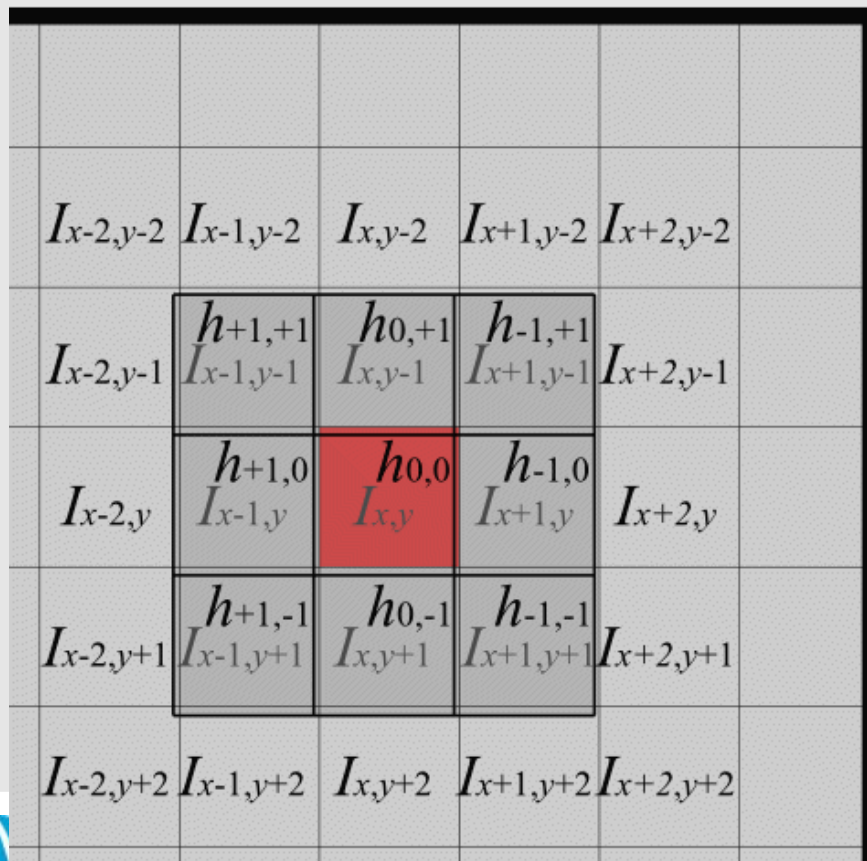
- C'est un filtrage **linéaire** (somme de multiplications entre les niveaux de gris de l'image initiale  $I$  et les **coefficients** du masque  $\mathcal{H}$ ).



## Filtrage par convolution discrète 2D (2/2)

### Calcul pratique

- Tourner le masque de 180° autour de son centre (0,0) ;
- Superposer le masque obtenu à l'image  $I$  de sorte qu'il soit centré en  $(x,y)$  ;
- Multiplier chaque coefficient du masque par le niveau du pixel sous-jacent ;
- Additionner chacun de ces produits pour obtenir  $I'(x,y)$ .



$$I'_{x,y} = I_{x-1,y-1} \cdot h_{+1,+1} + I_{x-1,y} \cdot h_{+1,0} + I_{x-1,y+1} \cdot h_{+1,-1} \\ + I_{x,y-1} \cdot h_{0,+1} + I_{x,y} \cdot h_{0,0} + I_{x,y+1} \cdot h_{0,-1} \\ + I_{x+1,y-1} \cdot h_{-1,+1} + I_{x+1,y} \cdot h_{-1,0} + I_{x+1,y+1} \cdot h_{-1,-1}$$



## Filtres de lissage (1/2)

- **Bruits dans une image**

- Sources : environnement (poussières), capteur (dérive), quantification ...
- **Restauration** d'image : on tente de retrouver l'image idéale  $\underline{I}$  (obtenue si l'acquisition était parfaite) en éliminant le bruit de l'image réelle  $I$ .
- **Modélisation** : souvent considéré comme aléatoire
  - bruit **additif** :  $I(x,y) = \underline{I}(x,y) + b(x,y)$
  - bruit **multiplicatif** :  $I(x,y) = \underline{I}(x,y) \cdot b(x,y)$

- **Ex.**



Additif gaussien ( $\sigma = 25$ )



Multiplicatif gaussien



« Poivre et sel »  
(20% de pixels affectés)

## Filtres de lissage (2/2)

I



- Filtres linéaires (*exemples*)

- Filtres moyennneurs

- + réduction du bruit

- forte atténuation des contours

$$\mathcal{H} = \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad \text{ou} \quad \frac{1}{10} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

moy3x3

- Filtres gaussiens

- + limitent l'effet de flou

- + degré de lissage paramétrable ( $\sigma$ )

$$\mathcal{H}_{\sigma=0,8} = \frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

méd3x3



- Filtres non linéaires. *Exemple* : le filtre médian

- Le niveau de gris résultat est le ndg médian des pixels voisins.

18	20	25
14	27	211
22	27	25

→ 14 ≤ 18 ≤ 20 ≤ 22 ≤ 25 ≤ 25 ≤ 27 ≤ 27 ≤ 211 →



|← 4 valeurs →| médiane |← 4 valeurs →|

25

 $I(\mathcal{V}(x,y))$ 

Valeurs de  $I(\mathcal{V}(x,y))$  triées par ordre croissant

 $I'(x,y)$ 


## Filtres morphologiques (1/2)

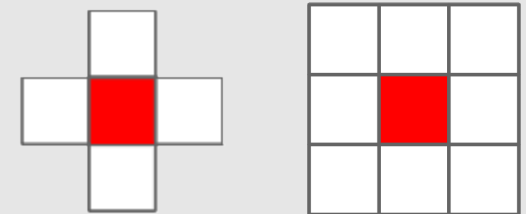
- **Filtres de rang (ou *filtres d'ordre*)**

- Généralisation du filtre médian.
- Principe : au lieu du niveau médian, on choisit le niveau d'un **rang** donné.
- Cas particuliers : on choisit le niveau local minimum ( $i_{min}$ ) ou maximum ( $i_{max}$ ) → filtres **morphologiques**.

- **Filtres morphologiques**

Utilisent un **élément structurant**  $\mathcal{B}$

≡ définition du voisinage (ou de la *connexité*), ex.:



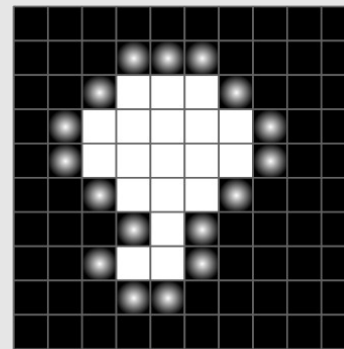
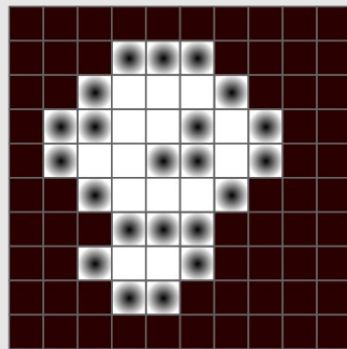
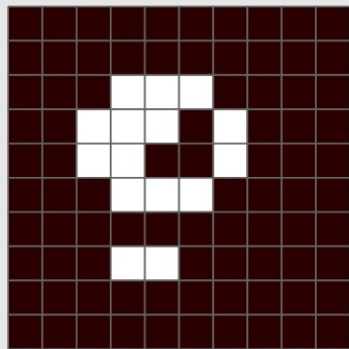
- **Dilatation** : affecte au pixel d'analyse  $P$  la valeur **maximale** de son voisinage :  
$$\delta_{\mathcal{B}}(I)(P) = \sup \{ I(Q) \mid Q \in \mathcal{B}(P) \}$$
- **Érosion** : affecte au pixel d'analyse  $P$  la valeur **minimale** de son voisinage :  
$$\varepsilon_{\mathcal{B}}(I)(P) = \inf \{ I(Q) \mid Q \in \mathcal{B}(P) \}$$

## Filtres morphologiques (2/2)

### Opérateurs morphologiques sur images binaires

#### → Fermeture (dilatation puis érosion) $\varepsilon_{\mathcal{B}}[\delta_{\mathcal{B}}(I)]$

- ferme les objets, remplit les trous

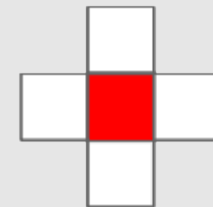


*mise à 1 par dilatation*



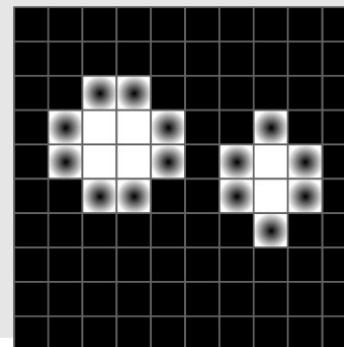
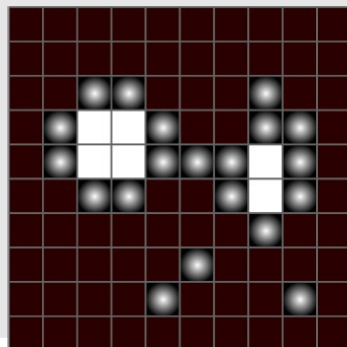
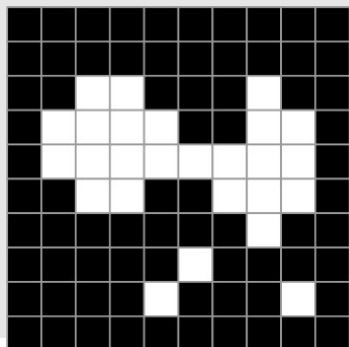
*mise à 0 par érosion*

*Élément structurant :*



#### → Ouverture (érosion puis dilatation) $\delta_{\mathcal{B}}[\varepsilon_{\mathcal{B}}(I)]$

- supprime les objets de largeur inférieure à celle de l'élément structurant
- sépare les objets fusionnés à tort



## Filtres de convolution sous OpenCV (1/2)

### Filtres de lissage prédéfinis

#### → Fonction

```
void cvSmooth( const CvArr* src, CvArr* dst, int smoothtype = CV_GAUSSIAN,
               int param1 = 3, int param2 = 0, double param3 = 0, double param4 = 0 )
```

#### → Paramètres

smoothtype	Nom	src ≡ dst possible ?	Profondeur de src      de dst		Description
CV_BLUR	Moyennage	Oui	8u, 32f	8u, 32f	Moyennage sur voisinage param1xparam2
CV_BLUR_NO_SCALE	Somme non normalisée	Non	8u, 32f	16s (src 8u), 32f (src 32f)	Somme sur voisinage param1xparam2
CV_MEDIAN	Filtrage médian	Non	8u	8u	Valeur médiane sur voisinage carré param1xparam1
CV_GAUSSIAN	Filtrage gaussien	Oui	8u, 32f	8u (src 8u), 32f (src 32f)	Filtre gaussien ( $\sigma$ =param3) sur voisinage param1xparam2
CV_BILATERAL	Filtrage bilatéral	Non	8u	8u	Filtrage bilatéral 3x3 de $\sigma_{img}$ =param1 et $\sigma_{coul}$ =param2

## Filtres de convolution sous OpenCV (2/2)

### Filtres définis par l'utilisateur

#### → Fonction

```
void cvFilter2D( const CvArr* src, CvArr* dst, const CvMat* kernel,
    CvPoint anchor = cvPoint(-1,-1) )
```

#### → Paramètres

*src*, *dst* : images source et destination (peuvent être la même image (*in-place*)<sup>1</sup>)

*kernel* : masque (ou noyau) de convolution, à coefficients **réels** (CV\_32FC1)

*anchor* : point d'ancrage (par défaut, le centre du masque)

<sup>1</sup>Rem.: les pixels de bord sont répliqués :  $in(-dx,y)=in(0,y)$ ,  $in(x,h-1+dy)=in(x,h-1)$ , ...

#### → Exemple

// Définir le filtre moyennneur 3x3 normalisé

```
CvMat* H = cvCreateMat( 3, 3, CV_32FC1 );
cvSetZero( H ); cvAddS( H, cvScalar(1.0), H );
cvNormalize( H, H, 1.0, 0.0, CV_L1 );
```

// Appliquer le filtre ainsi défini

```
cvFilter2D ( imSrc, imFiltered, H );
```

$$\mathcal{H} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

$$\mathcal{H} = \frac{1}{\sum_{i,j} |\mathcal{H}(i,j)|} \cdot \mathcal{H}$$



## Filtres morphologiques sous OpenCV (1/2)

### • Dilatation et érosion

#### → Fonctions

```
void cvDilate( IplImage* src, IplImage* dst, IplConvKernel* B = NULL,  
              int iterations = 1 )
```

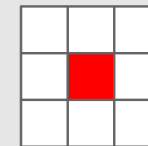
```
void cvErode( IplImage* src, IplImage* dst, IplConvKernel* B = NULL,  
              int iterations = 1 )
```

#### → Paramètres

*src*, *dst* : images source et destination (peuvent être la même image)

*B* : élément structurant (par défaut, 3x3 centré)

*iterations* : nombre d'itérations



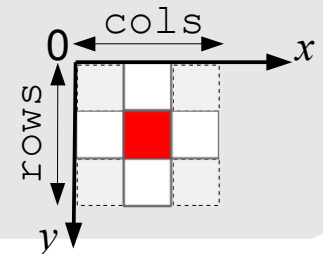
#### → Définition d'un élément structurant

##### ■ Fonction

```
IplConvKernel* cvCreateStructuringElementEx( int cols, int rows,  
                                             int anchor_x, int anchor_y, int shape, int* values= NULL )
```

##### ■ Exemple

```
IplConvKernel* B = cvCreateStructuringElementEx(  
    3, 3, 1, 1, CV_SHAPE_CROSS );
```



## Filtres morphologiques sous OpenCV (2/2)

### Filtres morphologiques avancés

#### → Fonction

```
void cvMorphologyEx( const CvArr* src, CvArr* dst, CvArr* temp,
    IplConvKernel* element, int operation, int iterations = 1 )
```

#### → Nouveaux paramètres

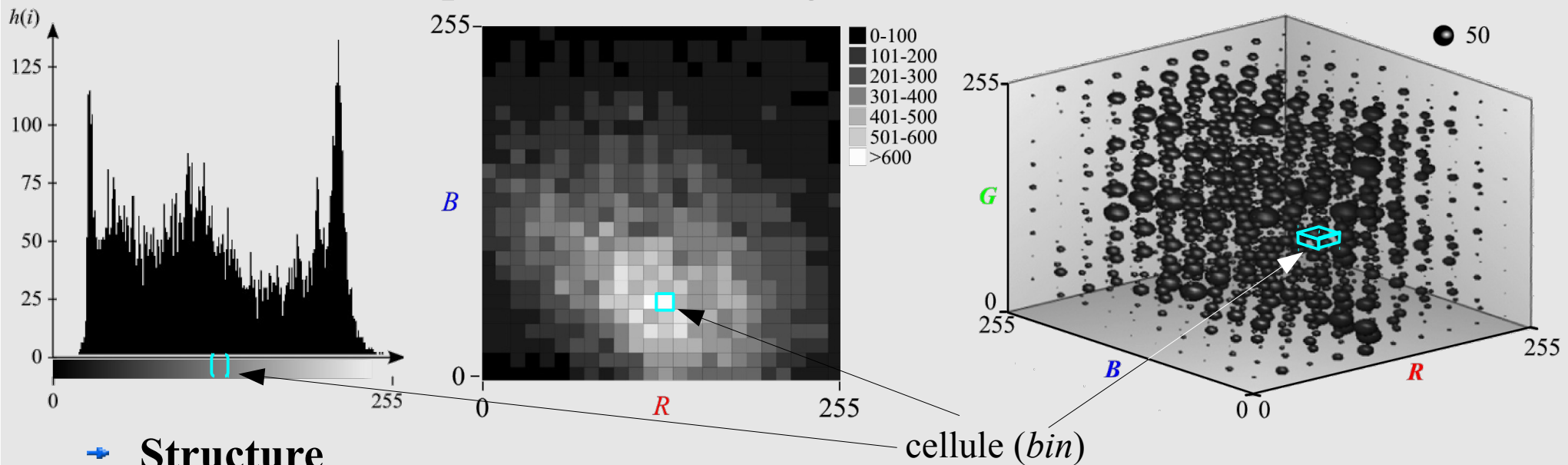
*temp* : tableau nécessaire à certaines opérations (même taille alors que *src*)

<i>operation</i>	Opérateur	Description de <i>dst</i>	<i>temp</i> requis ?	Utilisation
CV_MOP_CLOSE	Fermeture	$\varepsilon_{\mathcal{B}}[\delta_{\mathcal{B}}(src)]$	Non	Suppression des artefacts de bruit
CV_MOP_OPEN	Ouverture	$\delta_{\mathcal{B}}[\varepsilon_{\mathcal{B}}(src)]$	Non	Connexion des objets proches
CV_MOP_GRADIENT	Gradient	$\delta_{\mathcal{B}}(src) - \varepsilon_{\mathcal{B}}(src)$	Oui	Contours des objets
CV_MOP_TOPHAT	Chapeau haut-de-forme	$src - \delta_{\mathcal{B}}[\varepsilon_{\mathcal{B}}(src)]$	Si $src=dst$	Objets plus clairs que leur voisinage
CV_MOP_BLACKHAT	Chapeau noir	$\varepsilon_{\mathcal{B}}[\delta_{\mathcal{B}}(src)] - src$	Si $src=dst$	Objets plus sombres que leur voisinage

# Histogrammes sous OpenCV (1/4)

## Histogramme

→ Possibilité de représenter des histogrammes de toutes dimensions, *ex.*



## Structure

```
typedef struct CvHistogram {
    int      type; // type de stockage interne des données (matrice dense ou creuse)
    CvArr*   bins; // définition des cellules
    float    thresh[CV_MAX_DIM][2]; // seuils pour histogrammes uniformes
    float**  thresh2; // seuils pour histogrammes non uniformes
    CvMatND  mat; // matrice stockant les données (éventuellement CvSparseMat)
} CvHistogram;
```

## Histogrammes sous OpenCV (2/4)

### • Création d'un histogramme

#### ➔ Fonction

```
CvHistogram* cvCreateHist(  
    int      dims,  
    int*     sizes,  
    int      type,  
    float**  ranges = NULL,  
    int      uniform = 1  
)
```

#### ➔ Paramètres

*dims* : nombre de dimensions

*sizes* : tableau ( $0 \dots \text{dims}-1$ ) du nombre de cellules (taille) de chaque dimension

*type* : format de stockage interne (CV\_HIST\_ARRAY ou CV\_HIST\_SPARSE)

*ranges* : tableau ( $0 \dots \text{dims}-1$ ) des plages définissant les cellules. *ranges[i]* est :  
si *uniform*=1, tableau des 2 valeurs *min* et *max* de la dimension *i* ;

si *uniform*=0, tableau de *sizes[i]+1* valeurs seuils de la dimension *i* :

[ *min*<sub>0</sub>, *max*<sub>0</sub>, *max*<sub>1</sub>, *max*<sub>2</sub>, ..., *max*<sub>*sizes[i]-1*</sub> ] (implicite: *min*<sub>1</sub>=*max*<sub>0</sub>, *min*<sub>2</sub>=*max*<sub>1</sub>, ...)


*uniform* : booléen indiquant si les cellules sont définies par des plages uniformes

## Histogrammes sous OpenCV (3/4)

### • Utilisation d'un histogramme (*bases*)

#### → Calcul à partir d'image(s)

```
void cvCalcHist( IplImage** images, CvHistogram* hist,  
                int accumulate = 0, const CvArr* mask = NULL )
```

-  `images` est un **tableau** d'`IplImage*` pointant sur des images **mono-canal**
- le `mask` booléen indique les pixels à prendre en compte (ceux  $\neq 0$ )

#### → Accès aux cellules

- `double cvQueryHistValue_●D(CvHistogram* hist, <indice(s)>)`  
retourne la valeur de la cellule donnée par son (ses) indice(s) (cf. `cvGetReal_●D()`)
- `float* cvGetHistValue_●D(CvHistogram* hist, <indice(s)>)`  
retourne un pointeur sur la cellule donnée par son (ses) indice(s) (cf. `cvPtr_●D()`)
- Accès direct aux cellules (plus performant) : `hist->mat.data.fl`

#### → Accès à la structure (*ex.*)

- `int dim_i_nbins = hist->mat.dim[i].size;`
- `int dim_i_borne_max = hist->thresh[i][1];` // hist. uniforme
- `int dim_i_bin_j_max = hist->thresh2[i][j+1];` // hist. non uniforme

## Histogrammes sous OpenCV (4/4)

### • Utilisation d'un histogramme (*suite*)

#### → Désallocation

```
void cvReleaseHist( CvHistogram** hist )
```

#### → Remise à 0 de toutes les cellules

```
void cvClearHist( CvHistogram* hist )
```

#### → Valeurs minimale et maximale

```
void cvGetMinMaxHistValue( CvHistogram* hist, float* min_value,  
    float* max_value, int* min_idx = NULL, int* max_idx = NULL)
```

#### → Normalisation

```
void cvNormalizeHist( CvHistogram* hist, double factor)
```

Normalise `hist` à `factor` (souvent 1), chaque cellule en représentant une fraction

#### → Seuillage

```
void cvThreshHist( CvHistogram* hist, double factor)
```

Toutes les cellules de `hist` de valeur inférieure à `factor` sont mises à 0.

Utilisation pratique : annulation des cellules ne contenant que quelques points (*bruit*)

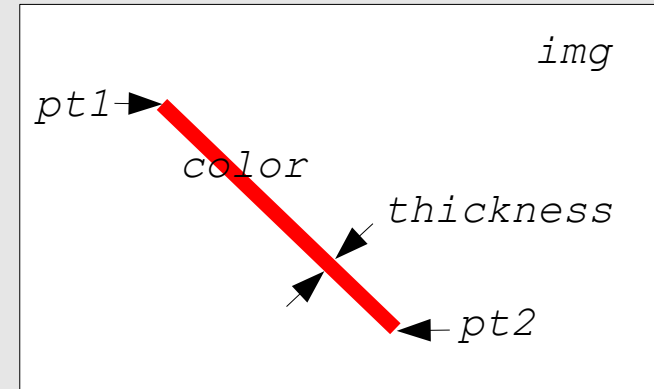


## Dessiner avec OpenCV (1/3)

### • Formes simples

#### → Ligne

```
void cvLine(  
    CvArr*    img,  
    CvPoint   pt1,  
    CvPoint   pt2,  
    CvScalar  color,  
    int       thickness = 1,  
    int       lineType = 8  
)
```

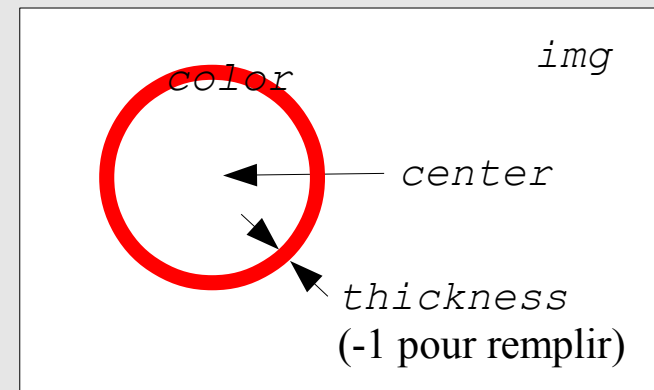


**Ex. :** `cvLine(img, cvPoint(100,100), cvPoint(200,200), CV_RGB(255,0,0), 5)`

→ **Rectangle** : idem, mais *pt1* et *pt2* = coins opposés et des valeurs négatives de *thickness* (ex. `CV_FILLED`) produisent un rectangle plein

#### → Cercle

```
void cvCircle(  
    CvArr*    img,  
    CvPoint   center,  
    int       radius,  
    CvScalar  color,  
    int       thickness = 1,  
    int       lineType = 8  
)
```



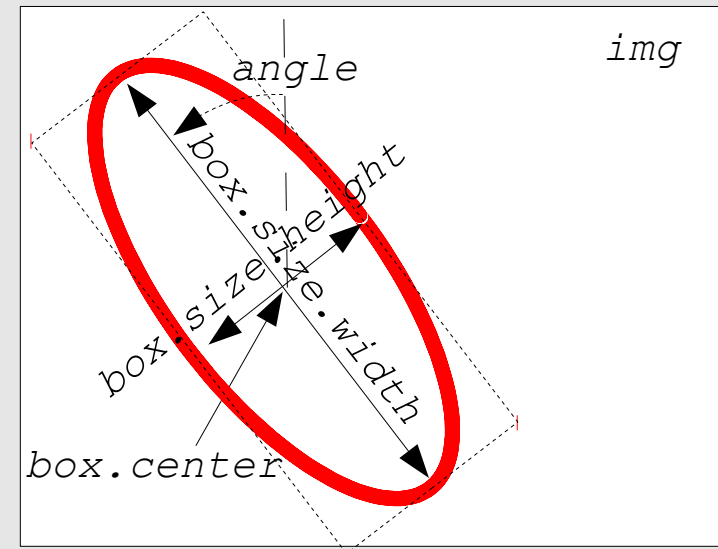
## Dessiner avec OpenCV (2/3)

### Autres formes

#### → Ellipses : 2 possibilités

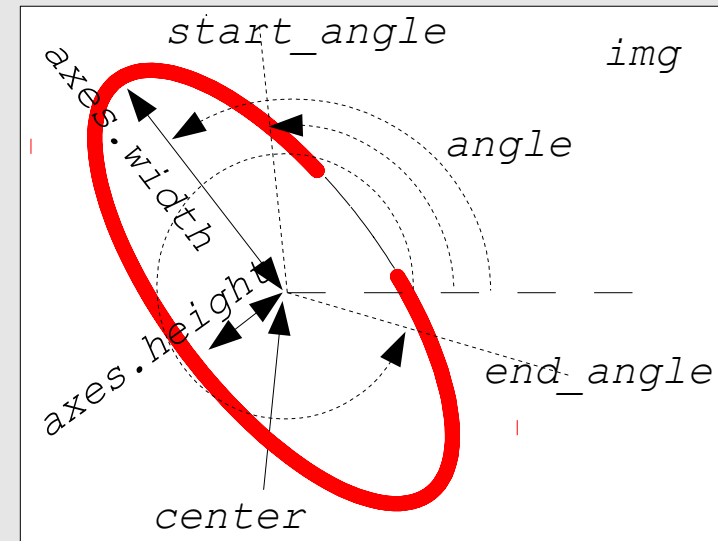
- Utilisation d'une boîte englobante

```
void cvEllipseBox(
    CvArr*      img,
    CvBox2D     box,
    CvScalar     color,
    int          thickness = 1,
    int          lineType = 8
)
```



- Arc d'ellipse généralisé

```
void cvEllipse(
    CvArr*      img,
    CvPoint     center,
    CvSize      axes,
    double       angle,
    double       start_angle,
    double       end_angle,
    CvScalar     color,
    int          thickness = 1,
    int          lineType = 8
)
```



## Dessiner avec OpenCV (3/3)

### • Texte

#### → Création d'une variable `CvFont`

```
void cvInitFont( CvFont* font, int font_face, double hscale, double vscale,  
                double shear = 0, int thickness = 1, int lineType = 8 )
```

avec échelles  $hscale, vscale \in \{0.5, 1.0\}$

inclinaison  $shear \in [0.0, 1.0]$  où  $1.0 \equiv 45^\circ$

police  $font\_face \in \{CV\_FONT\_*\}$  (cf. [doc en ligne](#) et [exemple](#))

#### → Ajout de texte à une image

```
void cvPutText( CvArr* img, const char* text, CvPoint origin,  
               const CvFont* font, CvScalar color )
```

avec `font` initialisé précédemment par `cvInitFont()`

#### → Exemple

```
CvFont font;  
IplImage* img = cvLoadImage("lena.jpg");  
cvInitFont( &font, CV_FONT_HERSHEY_SCRIPT_COMPLEX, 1.0, 1.0 );  
cvPutText ( img, "Bonjour", cvPoint(320, 340), &font, CV_RGB(0,0,0));
```

