

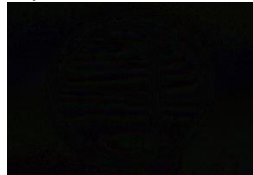

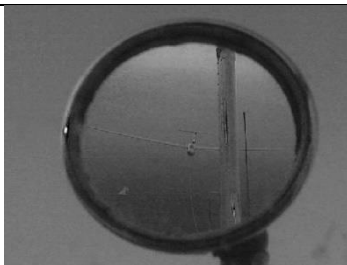

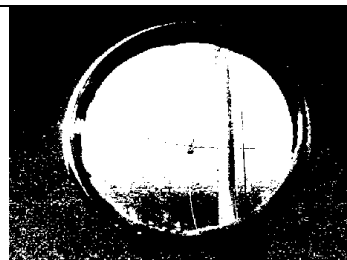









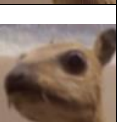
Informatique

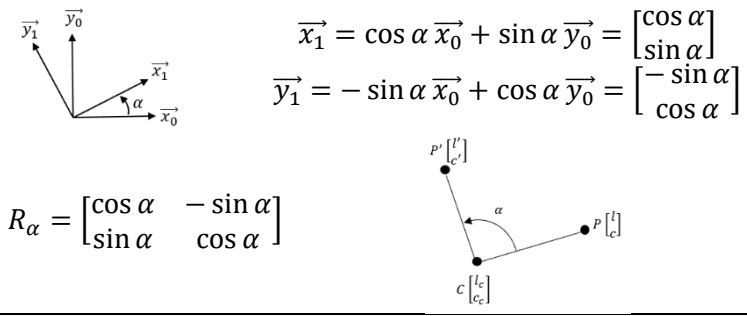


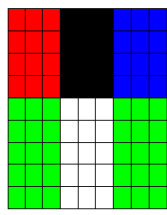
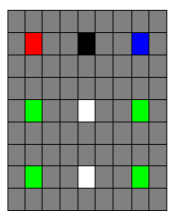

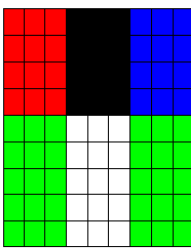
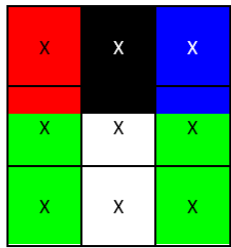
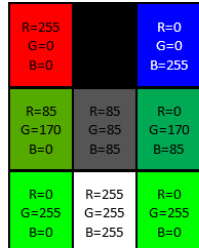

Matrices de pixels et images

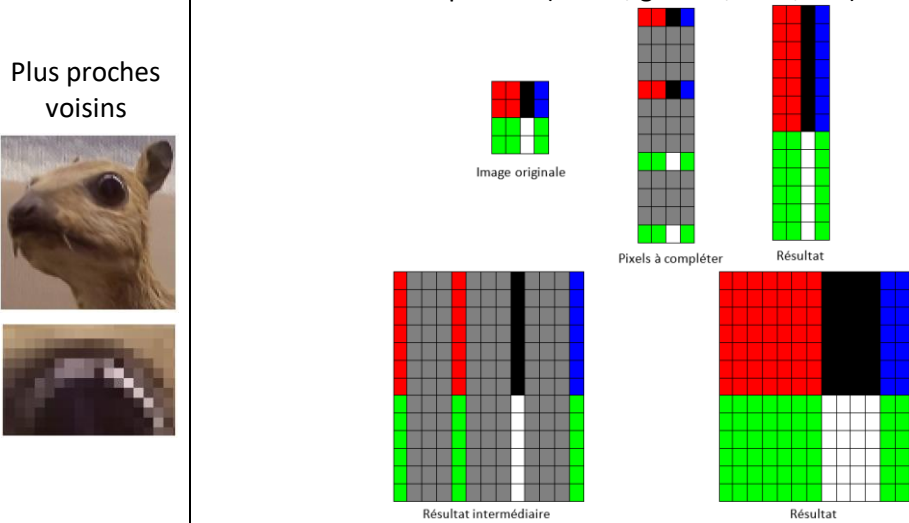
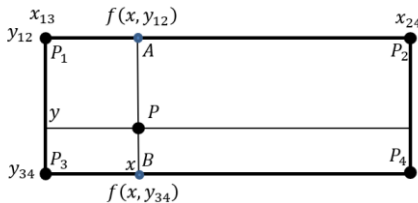
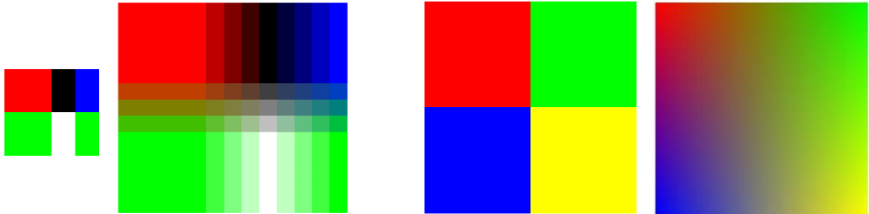
Résumé

Format sous python	Images au format BMP, gérées sous Python avec plt.imread sous forme d'array (numpy) de triplets d'entiers uint8 codés sur 8 bits (3 octets/pixel). Attention aux problèmes d'overflow dans les opérations +/- . Une image est donc un array (IM pour l'exemple) de L lignes et C colonnes. Attention, IM[0,0] est le pixel en haut à gauche de l'image... Ne pas utiliser IM[0][0], mais bien IM[0,0] – Sinon erreurs avec les slices																								
Pixel	<p>Un pixel IM[ligne, colonne] de l'image IM est un array contenant 3 entiers uint8 [R, G, B] (éventuellement [R, G, B, A]). Les bits « de gauche » des nombres entiers R, G et B sont appelés bits de poids forts, et ceux « de droite » les bits de poids faible</p> <table><tr><td></td><td colspan="2">R = 0</td><td></td><td colspan="2">R = 255</td></tr><tr><td></td><td>G = 0</td><td>G = 255</td><td></td><td>G = 0</td><td>G = 255</td></tr><tr><td>B = 0</td><td>[0,0,0]</td><td>[0,255,0]</td><td>B = 0</td><td>[255,0,0]</td><td>[255,255,0]</td></tr><tr><td>B = 255</td><td>[0,0,255]</td><td>[0,255,255]</td><td>B = 255</td><td>[255,0,255]</td><td>[255,255,255]</td></tr></table>		R = 0			R = 255			G = 0	G = 255		G = 0	G = 255	B = 0	[0,0,0]	[0,255,0]	B = 0	[255,0,0]	[255,255,0]	B = 255	[0,0,255]	[0,255,255]	B = 255	[255,0,255]	[255,255,255]
	R = 0			R = 255																					
	G = 0	G = 255		G = 0	G = 255																				
B = 0	[0,0,0]	[0,255,0]	B = 0	[255,0,0]	[255,255,0]																				
B = 255	[0,0,255]	[0,255,255]	B = 255	[255,0,255]	[255,255,255]																				
Exemple	<p>Ci-dessous, une image de 18 pixels bleus, rouges et verts</p> <table><tr><td>R=255 G=0 B=0</td><td>R=0 G=255 B=0</td><td>R=0 G=0 B=255</td><td>R=0 G=255 B=0</td><td>R=0 G=0 B=255</td><td>R=255 G=0 B=0</td></tr><tr><td>R=0 G=255 B=0</td><td>R=0 G=0 B=255</td><td>R=255 G=0 B=0</td><td>R=0 G=0 B=255</td><td>R=0 G=255 B=0</td><td>R=0 G=0 B=255</td></tr><tr><td>R=255 G=0 B=0</td><td>R=0 G=0 B=255</td><td>R=0 G=255 B=0</td><td>R=255 G=0 B=0</td><td>R=0 G=255 B=0</td><td>R=0 G=255 B=0</td></tr></table> <p>IM = $\begin{bmatrix} [255 & 0 & 0] & [0 & 255 & 0] & [0 & 0 & 255] & [0 & 255 & 0] & [0 & 0 & 255] & [255 & 0 & 0] \\ [0 & 255 & 0] & [0 & 0 & 255] & [255 & 0 & 0] & [0 & 0 & 255] & [0 & 255 & 0] & [0 & 0 & 255] \\ [255 & 0 & 0] & [0 & 0 & 255] & [0 & 255 & 0] & [255 & 0 & 0] & [0 & 255 & 0] & [0 & 255 & 0] \end{bmatrix}$</p>	R=255 G=0 B=0	R=0 G=255 B=0	R=0 G=0 B=255	R=0 G=255 B=0	R=0 G=0 B=255	R=255 G=0 B=0	R=0 G=255 B=0	R=0 G=0 B=255	R=255 G=0 B=0	R=0 G=0 B=255	R=0 G=255 B=0	R=0 G=0 B=255	R=255 G=0 B=0	R=0 G=0 B=255	R=0 G=255 B=0	R=255 G=0 B=0	R=0 G=255 B=0	R=0 G=255 B=0						
R=255 G=0 B=0	R=0 G=255 B=0	R=0 G=0 B=255	R=0 G=255 B=0	R=0 G=0 B=255	R=255 G=0 B=0																				
R=0 G=255 B=0	R=0 G=0 B=255	R=255 G=0 B=0	R=0 G=0 B=255	R=0 G=255 B=0	R=0 G=0 B=255																				
R=255 G=0 B=0	R=0 G=0 B=255	R=0 G=255 B=0	R=255 G=0 B=0	R=0 G=255 B=0	R=0 G=255 B=0																				
Création d'une image vierge	<p>Attention, ne créer qu'un array ne fonctionne pas, il faut spécifier le type int8 :</p> <pre>import numpy as np Nl,Nc = 100,100 Image_Noire = np.zeros((Nl,Nc,3),dtype='uint8') Image_Blanche = 255*np.ones((Nl,Nc,3),dtype='uint8')</pre>																								
Quelques outils																									
On oublie scipy pour matplotlib 😊	import matplotlib.pyplot as plt																								
Format .bmp RGB ou RGBA transformé en RGB « read only » : copier pour modifier : image.copy()	image = plt.imread("Nom.bmp") image = image[:,:,:3]																								
Ouverture/Sauvegarde au format array	import numpy as np image = np.load("Nom.npy") np.save('Nom',image)																								
Affichage Penser à balader la souris sur les pixels pour avoir les infos ligne (y), colonne (x) et triplet [R,G,B]	import matplotlib.pyplot as plt def f_affiche(image): plt.figure() plt.imshow(image) plt.axis('off') plt.show() plt.pause(0.00001) f_affiche(image)																								
Dimensions : lignes et colonnes	Nl,Nc = image.shape[0:2]																								
Accès à un pixel à la ligne l (int) et colonne c (int)	Pixel = image[l,c]																								
Parcours des pixels Utiliser (l,c) plutôt que (i,j) ou (y,x)	for c in range(Nc): for l in range(Nl):																								
Utilisation des slices (gain de temps vis-à-vis d'un for²)	image[:,:] = [127,127,127]																								
Accès aux entiers R, G et B	R,G,B = Pixel																								
Attention : manipulant des array, modifier des pixels modifie l'image en mémoire. Si on veut créer une copie pour garder l'image d'origine, écrire : image_2 = np.copy(image_1)																									
Enregistrement plt.imsave("Image.bmp", image) sans afficher (dimensions de l'image = lignes colonnes) ou plt.savefig("Image.png") en 640x480 après l'avoir affichée																									

Travail sur les images															
Bits de poids forts et faibles	Image complète 														
	4 Bits de poids forts, les 4 faibles=0 	4 Bits de poids faibles, les 4 forts=0 													
Nuances de gris	On obtient une nuance de gris en égalisant pour chaque pixel $R = G = B = Val$ à une valeur entre 0 et 255 (ou n'imposer que la valeur Val sans utiliser de triplets)														
	<table><tr><td>R=0</td><td>R=200</td></tr><tr><td>G=0</td><td>G=200</td></tr><tr><td>B=0</td><td>B=200</td></tr><tr><td>R=100</td><td>R=0</td></tr><tr><td>G=100</td><td>G=0</td></tr><tr><td>B=100</td><td>B=0</td></tr></table>			R=0	R=200	G=0	G=200	B=0	B=200	R=100	R=0	G=100	G=0	B=100	B=0
	R=0	R=200													
	G=0	G=200													
	B=0	B=200													
R=100	R=0														
G=100	G=0														
B=100	B=0														
On pourra privilégier une couleur plus qu'une autre en calculant Val en jouant sur (r, g, b) :															
$Val = \text{int}(rR + gG + bB) \in [0, 255]$ avec $r + g + b = 1 \quad (r, g, b) \in [0, 1]^3$															
Attention à l'overflow possible si $R + G + B > 255$, bien passer par la formule de Val															
	$r = 0.33 \quad g = 0.34 \quad b = 0.33$ Pas de couleur privilégiée	$r = 0 \quad g = 0 \quad b = 1$ Accent sur le bleu	$r = 1 \quad g = 0 \quad b = 0$ Accent sur le rouge												
															
Noir et blanc	Pixel noir : $R = G = B = 0$ - Pixel blanc : $R = G = B = 255$ (ou une seule valeur)														
	<table><tr><td>R=0</td><td>R=255</td></tr><tr><td>G=0</td><td>G=255</td></tr><tr><td>B=0</td><td>B=255</td></tr><tr><td>R=0</td><td>R=0</td></tr><tr><td>G=0</td><td>G=0</td></tr><tr><td>B=0</td><td>B=0</td></tr></table>			R=0	R=255	G=0	G=255	B=0	B=255	R=0	R=0	G=0	G=0	B=0	B=0
	R=0	R=255													
	G=0	G=255													
	B=0	B=255													
R=0	R=0														
G=0	G=0														
B=0	B=0														
Il suffira de définir un seuil sur Val après avoir transformé l'image en nuances de gris :															
$\begin{cases} Val = 0 & \text{si } Val < \\ Val = 255 & \text{si } Val \geq \text{Seuil} \end{cases}$															
	Seuil = 50	Seuil = 127	Seuil = 200												
															

Convolution			
Principe avec un noyau de dimensions 3x3	<p>Noyau $K = \begin{bmatrix} K_{11} & K_{12} & K_{13} \\ K_{21} & K_{22} & K_{23} \\ K_{31} & K_{32} & K_{33} \end{bmatrix}$ à n lignes et n colonnes</p> <p>$Im = \begin{bmatrix} P_{11} & P_{12} & P_{13} & P_{14} & P_{15} \\ P_{21} & P_{22} & P_{23} & P_{24} & P_{25} \\ P_{31} & P_{32} & P_{33} & P_{34} & P_{35} \\ P_{41} & P_{42} & P_{43} & P_{44} & P_{45} \\ P_{51} & P_{52} & P_{53} & P_{54} & P_{55} \end{bmatrix}$; $P_{ij} = [R_{ij}, G_{ij}, B_{ij}]$</p> <p>$Im' = \begin{bmatrix} P'_{11} & P'_{12} & P'_{13} & P'_{14} & P'_{15} \\ P'_{21} & P'_{22} & P'_{23} & P'_{24} & P'_{25} \\ P'_{31} & P'_{32} & P'_{33} & P'_{34} & P'_{35} \\ P'_{41} & P'_{42} & P'_{43} & P'_{44} & P'_{45} \\ P'_{51} & P'_{52} & P'_{53} & P'_{54} & P'_{55} \end{bmatrix}$; $P'_{ij} = [R'_{ij}, G'_{ij}, B'_{ij}]$</p> <p>$P'_{ij} = K * M_{ij}$; $M_{ij} = \begin{bmatrix} P_{i-1,j-1} & P_{i-1,j} & P_{i-1,j+1} \\ P_{i,j-1} & P_{i,j} & P_{i,j+1} \\ P_{i+1,j-1} & P_{i+1,j} & P_{i+1,j+1} \end{bmatrix}$</p> <p>$P'_{ij} = K_{11}P_{i-1,j-1} + K_{12}P_{i-1,j} + K_{13}P_{i-1,j+1} + K_{21}P_{i,j-1} + K_{22}P_{i,j} + K_{23}P_{i,j+1} + K_{31}P_{i+1,j-1} + K_{32}P_{i+1,j} + K_{33}P_{i+1,j+1}$</p> <p>$K_{ij}P_{ij} = K_{ij}[R_{ij}, G_{ij}, B_{ij}] = [K_{ij}R_{ij}, K_{ij}G_{ij}, K_{ij}B_{ij}]$</p>		
	Moyenneur Floutage	$K = \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
	Repoussage	$K = \begin{bmatrix} -2 & -1 & 0 \\ -1 & 1 & 1 \\ 0 & 1 & 2 \end{bmatrix}$	
	Laplacien Contours	$K = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
Filtres	Réhausseur	$K = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
	Gaussien 3x3	$K = \frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} V = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \Rightarrow K = \frac{V * V^T}{16}$	
	Gaussien 5x5	$K = \frac{1}{256} \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix} V = \begin{bmatrix} 1 \\ 4 \\ 6 \\ 4 \\ 1 \end{bmatrix} \Rightarrow K = \frac{V * V^T}{256}$	
Gestion des bords	On peut les laisser à leur valeur d'origine, redimensionner l'image de sortie en les enlevant, ou traiter les formules au cas par cas		
Normalisation	Afin de maintenir une moyenne des RGB par convolution, on peut normaliser la matrice en divisant chacun de ses termes par la somme de tous ses termes		
Re limitation des résultats	Dans le cadre des BMP par exemple et pour éviter l'overflow, on relimite les valeurs de R, G et B afin de les maintenir dans l'intervalle [0,255] avec une formule comme : $E = \min(\max(E, 0), 255)$		

Transformations		
Rotation	R_α  $\vec{x}_1 = \cos \alpha \vec{x}_0 + \sin \alpha \vec{y}_0 = \begin{bmatrix} \cos \alpha \\ \sin \alpha \end{bmatrix}$ $\vec{y}_1 = -\sin \alpha \vec{x}_0 + \cos \alpha \vec{y}_0 = \begin{bmatrix} -\sin \alpha \\ \cos \alpha \end{bmatrix}$ $R_\alpha = \begin{bmatrix} \cos \alpha & -\sin \alpha \\ \sin \alpha & \cos \alpha \end{bmatrix}$	
	<p>Direct</p>  <p>Pour chaque pixel $P(l, c)$ de l'image source, on trouve l'image $P'(l', c')$ par rotation d'angle α et de centre $C(l_c, c_c)$ dans l'image cible, à laquelle on applique le triplet RGB de P : $Im'(l', c') = Im(l, c)$</p> $\overrightarrow{CP'} = R_\alpha \overrightarrow{CP} = \begin{bmatrix} \cos \alpha & -\sin \alpha \\ \sin \alpha & \cos \alpha \end{bmatrix} \begin{bmatrix} l - lc \\ c - cc \end{bmatrix} = \begin{bmatrix} a \\ b \end{bmatrix} ; \quad \begin{bmatrix} l' \\ c' \end{bmatrix} = \begin{bmatrix} lc + [a] \\ cc + [b] \end{bmatrix}$ <p>Cette méthode crée des trous, les coordonnées (l', c') ne tombant pas sur 100% des pixels de l'image cible</p>	
	<p>Inverse</p>  <p>Pour chaque pixel $P'(l', c')$ de l'image cible, on trouve l'antécédent $P(l, c)$ par rotation d'angle $-\alpha$ et de centre $\begin{bmatrix} lc \\ cc \end{bmatrix}$ dans l'image source, et on applique à P' le triplet RGB de P : $Im'(l', c') = Im(l, c)$.</p> $\overrightarrow{CP} = R_{-\alpha} \overrightarrow{CP'} = \begin{bmatrix} \cos \alpha & \sin \alpha \\ -\sin \alpha & \cos \alpha \end{bmatrix} \begin{bmatrix} l' - lc \\ c' - cc \end{bmatrix} = \begin{bmatrix} a \\ b \end{bmatrix} ; \quad \begin{bmatrix} l \\ c \end{bmatrix} = \begin{bmatrix} lc + [a] \\ cc + [b] \end{bmatrix}$ <p>Il est aussi possible de déterminer la valeur de P' par interpolation bilinéaire sur les 4 pixels entourant les coordonnées flottantes (lf, cf) avec $\begin{bmatrix} lf \\ cf \end{bmatrix} = \begin{bmatrix} lc + a \\ cc + b \end{bmatrix}$</p>	
Réduction	<p>Prendre 1 pixels tous les n pixels. Risque d'effet d'aliasing (lignes visibles)</p> <p>Pour limiter cet effet, appliquer un filtre moyennneur avant suppression par exemple</p> <div style="display: flex; align-items: center;">    </div> <p style="text-align: center;">Image originale Suppression Résultat</p>	
	<p>Prendre la moyenne par paquets de nxn pixels, et ce tous les n pixels tous les n pixels en ligne et en colonne</p> <div style="display: flex; align-items: center;">   <div style="margin: 0 10px;">  </div>  </div> <p style="text-align: center;">Image originale Pixels à garder/modifier Résultat intermédiaire Résultat</p> <p>Cette méthode appliquée à de grandes images donne des résultats plus concluants que la précédente par une transition plus douce d'une couleur à l'autre</p>	

Agrandissement	Principe	Ajouter des lignes et/ou colonnes à l'image et déterminer la couleur à imposer aux nouveaux pixels par interpolation
	Plus proches voisins	<p>P prend la valeur du triplet du pixel le plus proche avec un choix de priorité (droite, gauche, haut, bas)</p>  <p>Exemple avec priorité à gauche ou au-dessus</p>
	Interpolation bilinéaire	<p>On connaît les 4 triplets RGB autour d'une zone dans laquelle on ajoute des pixels P aux coordonnées (x, y)</p>  <p>On détermine le triplet RGB en fonction des 4 pixels avoisinants à l'aide d'une fonction f telle que :</p> $f(x, y) = \frac{1}{(y_{34} - y_{12})(x_{24} - x_{13})} [y_{34} - y \quad y - y_{12}] \begin{bmatrix} P_1 & P_2 \\ P_3 & P_4 \end{bmatrix} \begin{bmatrix} x_{24} - x \\ x - x_{13} \end{bmatrix}$ $f(x, y) = ax + by + cxy + d$ <p>Si P est à 1/3 du bord gauche et 1/3 du bord inférieur (comme sur l'image)</p> $P = \frac{2}{3}B + \frac{1}{3}A \quad ; \quad A = \frac{2}{3}P_1 + \frac{1}{3}P_2 \quad ; \quad B = \frac{2}{3}P_3 + \frac{1}{3}P_4$ 
	Interpolation bicubique	<p>Sans rentrer dans les détails, on propose une interpolation qui va utiliser, en plus des valeurs des 4 pixels avoisinants, des valeurs plus éloignées (12 pixels de plus) afin de tenir compte, en plus de l'interpolation bilinéaire, des évolutions des valeurs des pixels selon l'espace.</p> $f(x, y) = \sum_{i=0}^3 \sum_{j=0}^3 a_{ij} x^i y^j$