

Traitemet des images

4MEO - 4MIN

Ces notes constituent la version papier 2016 du syllabus.

Afin de réduire l'empreinte carbone, le poids et le cout, celle-ci ne comporte qu'une centaine de pages extraites du syllabus accessible via Eole. De cette manière, l'étudiant dispose d'informations (équations, figures, schémas, tableaux, extraits de codes, ...) qu'il peut aisément annoter lors des séances de cours ou de laboratoire.

Attention, le syllabus disponible via Eole comporte différents fichiers (pdf, ods, vidéos, logiciels, pages internet, ...).

Il est possible qu'il soit complété durant le déroulement de l'activité conformément à ce qui est alors annoncé en séance. Il est donc important de reconsulter Eole au terme de l'activité pour s'assurer que l'on dispose bien des dernières nouveautés.

Contenu	1	
Lumière - vision - colorimétrie	3	
Diagramme de chromaticité	22	
RGB - CMJN - TSL	24	[1]
Bitmap sous Matlab (Clown)	29	[2]
Ecran - Capteur - YUV	30	
Diagramme R-Y / B-Y	31	
Capteur d'images à CCD	32	
Image sensor	34	
Compression - Image et vidéo	36	
Compression vidéo	40	[3]
ffmpeg	42	
3D - Coordonnées homogènes - exemple	43	
Dessin de pièce en 3D avec Octave	46	
Caméra - Position - Repères	47	
En langage Matlab	48	
Caméra - Python sous Windows	61	
1. Installation sous Python 2.7	61	
2. Tutoriel	61	
Introduction à OpenCV	65	[4]
Robot - Caméra embarquée - Shader	76	[5]
OpenGL ES Shadind Language 1.0 Ref Card	85	[6]
A Beginner's guide to coding graphics shaders	87	[7]
Example of Vertex Shader	88	
A link to WebGL intoduction	89	[8]
Résonance magnétique nucléaire	90	

Si des commentaires sont ajoutés sur des pdf, ils sont généralement en italique 10pt Times bleu foncé pour se distinguer de l'original.

Références

*Les liens commençant par **_E-** correspondent à des fichiers disponibles sur Eole que vous devez avoir préalablement chargés dans le même dossier que le présent fichier :*

_E_Syllabus_4MINEO_Images_2016.pdf

- [1] <http://www.la-photo-en-faits.com/2013/05/RVB-CMJN-TSL-conversion-definition.html>
- [2] <http://blogs.mathworks.com/steve/2006/02/03/all-about-pixel-colors-part-2/>
- [3] <https://ffmpeg.org/ffmpeg.html>
- [4] <http://wcours.gel.ulaval.ca/2014/h/GEL3014/default/5notes/documents/opencv.pdf>
- [5] <http://wcours.gel.ulaval.ca/2012/h/GEL3014/default/7references/opencv.pdf>
- [6] <http://opencv.org/documentation.html>
- [7] <http://gamedevelopment.tutsplus.com/tutorials/a-beginners-guide-to-coding-graphics-shaders--cms-23313>
- [8] <http://davidscottlyons.com/threejs/presentations/frontporch14/#slide-0>

LUMIÈRE - VISION - COLORIMÉTRIE

1. LA LUMIÈRE

On sait que la lumière est un rayonnement d'énergie électromagnétique. La vitesse de la lumière dans le vide est une constante universelle, désignée par **C** et approximativement égale à 300 000 km/s. La vitesse de propagation varie selon le milieu traversé, transparent ou translucide, sa valeur est :

$$V = \frac{C}{n}$$

n, l'indice de réfraction, dépend du milieu, il est égal à 1 dans le vide et très voisin de 1 dans l'air. La lumière monochromatique est dotée d'une fréquence **f** indépendante du milieu dans lequel elle se propage, tandis que sa longueur d'onde dépend du milieu traversé.

Dans un milieu d'indice de réfraction **n** :

$$\lambda = \frac{V}{f} = \frac{C}{n.f}$$

Il serait plus normal de caractériser une lumière monochromatique par sa fréquence qui est fixe que par sa longueur d'onde qui dépend du milieu traversé. L'usage a cependant choisi la longueur d'onde dans le vide pour la définir.

ANALYSE DE LA LUMIÈRE PAR UN PRISME

Un rayon de lumière solaire vient frapper un prisme selon un angle d'incidence θ_1 , à la perpendiculaire au plan du prisme (fig. 1). Dans le verre le rayon change de direction et se rapproche cette perpendiculaire, formant avec elle un angle θ_2 . C'est le phénomène de réfraction. À la sortie du prisme, lors du passage du verre à l'air, un second phénomène de réfraction se produit avec des angles identiques. On peut résumer l'ensemble des résultats par les formules suivantes :

$$V = \frac{1}{\sqrt{\epsilon \cdot \mu}} \quad \text{avec} \quad \mu = \mu_0 \cdot \mu_r, \quad \mu_r \approx 1 \text{ et } \mu_0 = 4\pi \cdot 10^{-7}$$

on déduit $n = \sqrt{\epsilon_r \cdot \mu_r}$

$$n_1 \cdot \sin \theta_1 = n_2 \cdot \sin \theta_2$$

Un milieu dispersif se caractérise par une vitesse de propagation, donc également un indice de réfraction, fonction de la fréquence. C'est le cas dans le verre, et par conséquent, la lumière blanche subit dans un prisme en verre une décomposition. Le rayonnement bleu est plus dévié que le rouge, donc la vitesse du bleu est plus altérée par le passage dans le verre que celle du rouge.

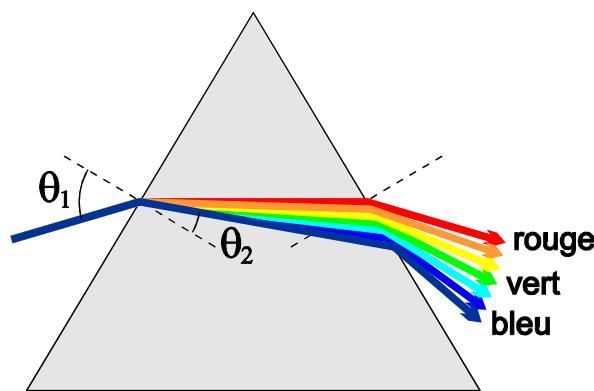


fig.1 Déviation de la lumière dans un prisme

Cette déviation par un prisme donne une analyse spectrale de la lumière solaire, elle montre que la lumière blanche n'existe pas et qu'elle est formée par la perception simultanée de plusieurs radiations dans un rapport précis.

Grâce à cette expérience, on peut définir la longueur d'onde des différentes couleurs.

COURBE SPECTRALE D'ÉNERGIE

La lumière étant un rayonnement d'énergie, il est possible de la mesurer pour chaque radiation monochromatique formant le spectre de cette lumière. On définit le blanc **E** d'égale énergie sur l'ensemble du spectre visible (c'est-à-dire entre 380 et 780 nm). Il est représenté à la figure 2.

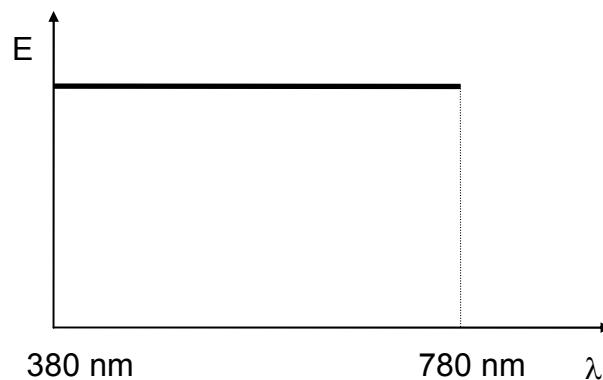


Fig. 2 Le blanc E est d'égale énergie sur le spectre visible

Les couleurs telles que perçues par l'individu moyen ont pour longueur d'onde λ :

Violet	390 - 455 nm
Bleu	455 - 492 nm
Vert	492 - 577 nm
Jaune	577 - 597 nm
Orange	597 - 622 nm
Rouge	622 - 780 nm

FILTRES

Les rayonnements lumineux parviennent à notre œil, soit directement, soit par l'intermédiaire des objets éclairés par la source. Ces objets peuvent altérer plus ou moins le rayonnement original selon qu'ils absorbent ou réfléchissent de façon préférentielle tout ou partie du spectre reçu.

Les corps qui se laissent traverser de façon égale par toutes les radiations sont incolores (verre, eau,...). Les corps qui *diffusent* dans toutes les directions et de façon égale tout le spectre sont blancs. Ceux qui absorbent toutes les radiations sont noirs.

Enfin, ceux qui absorbent ou diffusent inégalement ces radiations en fonction de leur longueur d'onde, sont colorés. Ils se comportent en filtres.

Un filtre laisse passer une certaine partie de la lumière qui le traverse. On trace sa courbe de transmittance. La courbe complémentaire est celle d'absorption. Un filtre est monochromatique lorsqu'il ne transmet que l'énergie correspondant à une seule longueur d'onde.

Il est possible d'associer plusieurs filtres en série. La courbe de transmittance résultante est évidemment obtenue en multipliant les transmittances des filtres pour chaque longueur d'onde du spectre.

Grand nombre de surfaces sont à la fois partiellement réfléchissantes et diffusantes. La réflexion se fait suivant une direction préférentielle et n'est pas sensible à la longueur d'onde tandis que la diffusion se fait dans toutes les directions avec un effet de filtre flagrant.

2. VISION et notions de PHOTOMETRIE

L'ŒIL HUMAIN

Le globe oculaire a une forme approximativement sphéroïde (fig. 3). La lumière entre dans l'œil par la *pupille*, un orifice circonscrit par l'*iris*, puis traverse différents milieux réfringents : l'humeur aqueuse qui est un liquide, le *cristallin* qui n'est autre qu'une lentille biologique dont la convergence est variable, et l'humeur vitrée. La lumière atteint alors la *rétine*. La rétine est la membrane photosensible qui tapisse le fond de l'œil. Elle est reliée au cerveau par les fibres du nerf optique. Elle comporte 2 types de cellules sensibles : les cônes et les bâtonnets.

Les cônes sont les éléments actifs en présence d'une lumière intense, comme celle qui existe durant le jour. La vision due aux cônes est appelée *photopique*. Les cônes se répartissent, en fonction de leurs pigments, en trois types d'éléments photosensibles caractérisés chacun par une certaine sensibilité spectrale, approximativement dans le rouge, dans le vert et dans le bleu. Au centre de la rétine, il y a une tache jaune d'environ 1.7 mm de diamètre appelée *fovea*. Elle est placée sur l'axe optique de l'œil, un peu en dehors du lieu d'épanouissement du nerf optique (lieu que l'on nomme papille de l'œil et qui est dépourvu de sensibilité, d'où son nom de tache aveugle). La fovea ne contient que des cônes en grande densité. Le reste de la rétine contient de moins en moins de cônes au fur et à mesure que l'on s'éloigne de la tache jaune et de plus en plus de bâtonnets.

Les bâtonnets sont beaucoup plus sensibles que les cônes. Ils permettent donc la vision lorsqu'il fait sombre. On parle de vision *scotopique*. Ils ne permettent pas de distinguer les couleurs (la nuit tous les chats sont gris) et offrent une résolution moins fine que les cônes. Ils sont saturés à la lumière du jour.

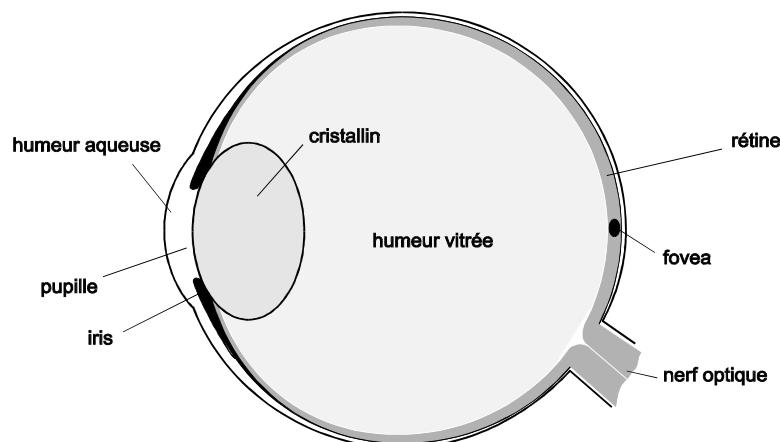


Fig. 3 L'œil humain

Pouvoir séparateur

La distance focale du cristallin est d'environ 2 cm et la distance entre cônes est de 1 à 5 µm, ce qui permet de déterminer l'angle d'acuité visuelle ou le pouvoir séparateur de l'œil. En effet,

$$\alpha \approx \sin \alpha = \frac{5 \mu\text{m}}{2 \text{ cm}} = \frac{1}{4000} \text{ radian} = 0.014 \text{ deg rés} = 1 \text{ minute}$$

on trouve un pouvoir séparateur d'environ une minute d'angle. L'œil ne peut donc pas distinguer des informations trop petites ou trop proches l'une de l'autre. Cette caractéristique permet de calculer la définition optimale d'une image par exemple télévisée. Le pouvoir séparateur est le plus petit pour une image en noir et blanc. Il est 2 à 5 fois plus élevé pour la transmission des couleurs. On en déduit que la reproduction des couleurs n'impose pas une finesse de détails aussi grande que celle exigée par le noir et blanc. En d'autres termes les vidéosignaux de luminance doivent avoir une bande passante nettement plus large que ceux qui transmettent les signaux de chrominance.

Persistante rétinienne

L'excitation de la rétine ne s'établit que progressivement lorsqu'elle est soumise à un rayonnement lumineux. La disparition de la stimulation n'interrompt pas immédiatement la sensation visuelle. Cette persistante rétinienne permet de concevoir les systèmes de cinéma et de télévision. Elle a une durée variable selon la fréquence d'excitation ; elle est plus longue pour le bleu que pour le rouge. Elle augmente également lorsque l'intensité lumineuse est plus importante.

La fréquence critique à laquelle la synthèse du mouvement peut encore s'établir, représente le seuil en dessous duquel un effet de papillotement est ressenti. La rétine traduit vers le cerveau la succession des impulsions lumineuses, ce qui fatigue l'œil, pouvant aller jusqu'à une sensation douloureuse. Le papillotement est plus ressenti à la périphérie du champ visuel qu'en son centre. La fréquence critique augmente avec

l'éclairement de l'image. C'est pourquoi on distingue parfois un papillotement seulement sur les parties blanches très éclairées.

PHOTOMETRIE, VISIBILITE RELATIVE

La photométrie est la science des mesures de l'intensité des rayonnements visibles ou proches du visible. Après des années d'expérience, la Commission Internationale de l'Eclairage (C.I.E.) a adopté, en 1924, une courbe conventionnelle qui caractérise l'observateur moyen (fig. 4). La sensibilité de l'oeil est maximale pour la lumière vert-jaune à 555 nm. Elle est très réduite pour le bleu et encore plus pour le rouge. Il faut donc fournir à l'oeil beaucoup plus d'énergie rouge et bleue que d'énergie verte pour obtenir une sensation lumineuse équivalente.

En vision nocturne (scotopique), on observe que les bleus sont plus visibles que les rouges ; la courbe de vision relative est décalée vers les bleus avec une vision maximale à 500 nm.

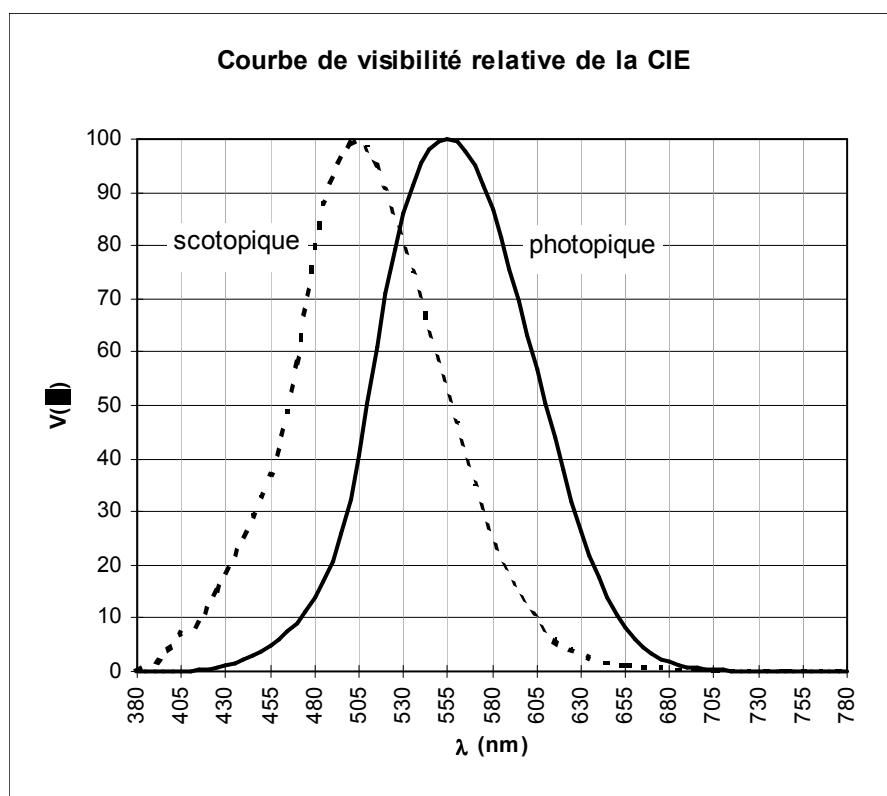


fig. 4 Courbe $V(\lambda)$ de visibilité relative

Les unités énergétiques

FLUX ÉNERGÉTIQUE : L'énergie par unité de temps, émise par une source stable, est appelée sa puissance ou son flux énergétique dont l'unité est le watt.

INTENSITÉ ÉNERGÉTIQUE : L'intensité énergétique d'une source ponctuelle est le flux énergétique émis par cette source dans l'unité d'angle solide. Son unité est le watt par stéradian.

LUMINANCE ÉNERGÉTIQUE : La luminance énergétique d'une source étendue est, dans une direction donnée, l'intensité énergétique fournie par l'unité de surface apparente de la source.

ÉCLAIREMENT ÉNERGÉTIQUE : L'éclairement énergétique, dont l'unité est le watt/m² est le flux énergétique reçu par unité de surface de l'objet éclairé.

Les unités lumineuses

LUMINANCE VISUELLE (ou luminance) : On appelle *luminance* visuelle d'une radiation monochromatique le produit, à un coefficient près, de la luminance énergétique par le coefficient de visibilité relative. L'unité de luminance visuelle est la candela par m² (cd/m²) ou nit.

INTENSITE LUMINEUSE : son unité est la *candela* (cd).

FLUX LUMINEUX : son unité est le lumen (lm) ou candela-stéradian,

ÉCLAIREMENT LUMINEUX : son unité est le *lux* : 1 lm/ m².

Lien entre les unités énergétiques et lumineuses

Par définition, l'oxyde de thorium porté à la température de solidification du platine (2042 K) a une luminance de 600 kcd/m². Cet oxyde est un corps noir, c'est à dire qu'il possède le spectre de rayonnement de tout corps noir, déterminé en fonction de sa température. Notons que le soleil est également un corps noir dont le spectre d'émission correspond à une température de 5870°C.

$$L \equiv 600 \text{ kcd} / \text{m}^2 = K_M \cdot \int_{400nm}^{700nm} (L_{e\lambda})_{CN} \cdot V(\lambda) \cdot d\lambda$$

avec $(L_{e\lambda})_{CN}$ la densité de luminance énergétique du corps noir concerné pour chaque valeur de λ . Elle s'exprime en W/(rd.m².nm) et est par exemple mesurée de 5 en 5 nm. Les mesures ont conduit à établir la constante K_M .

$$K_M = 683 \text{ lm} / \text{W}$$

Dès lors, une grandeur lumineuse quelconque G est reliée à son équivalent énergétique G_e par la relation

$$G = 683 \text{ lm} / \text{W} \cdot \int_{400nm}^{700nm} V(\lambda) \cdot G_{e\lambda} \cdot d\lambda$$

Par exemple, pour la luminance,

$$L = 683 \cdot \int_{400nm}^{700nm} V(\lambda) \cdot L_{e\lambda} \cdot d\lambda$$

Notons que les valeurs de table à utiliser pour la courbe $V(\lambda)$ sont les mêmes que celles que l'on donne plus loin pour la coordonnée Y du système international XYZ à condition de les diviser par la valeur maximale 1.0002 obtenue à 555 nm.

3. LA COLORIMETRIE

3.1. LOIS DE GRASSMANN

- La luminance d'un mélange est égale à la somme des luminances de ses composantes.
- En mélangeant dans des proportions déterminées trois radiations appropriées, on peut reproduire toute impression colorée quelconque.
- Si deux plages lumineuses produisent la même impression colorée, cette égalité d'impression subsistera lorsque la luminance de l'une et de l'autre sera multipliée ou divisée par un même nombre.
- Deux mélanges lumineux qui juxtaposés produisent la même impression colorée, se comportent aussi de façon identique dans le processus des mélanges. Ils peuvent donc se substituer l'un à l'autre.

La trichromie additive

On illumine un écran diffusant au moyen de trois projecteurs munis respectivement d'un filtre rouge d'un filtre vert et d'un filtre bleu. Un diaphragme permet de régler la luminance de chacune des sources (fig.5).

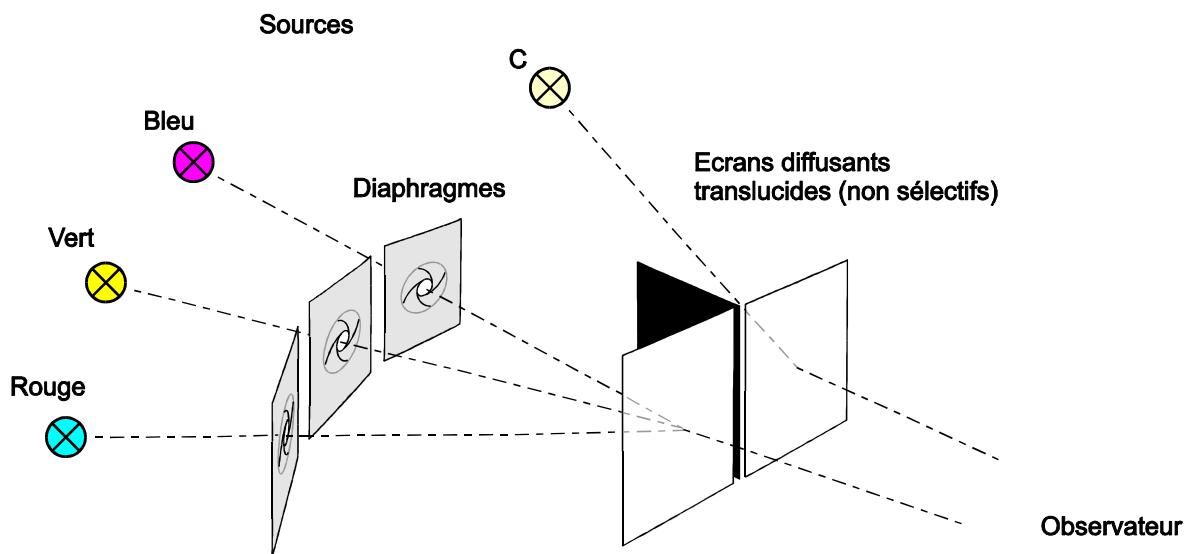


fig. 5 Reproduction d'une couleur C par trois couleurs R, G, B

Si les trois faisceaux ne sont pas centrés et si les luminances sont convenablement réglées on obtient sept plages (fig. 6.a) :

- au centre une tache blanche formée par l'addition des trois lumières primaires,
- à la périphérie trois taches fournies par les primaires rouge (R), vert (G), bleu (B),
- au stade intermédiaire trois taches obtenues par les mélanges suivants :

vert + rouge = jaune

bleu + rouge = magenta

vert + bleu = cyan

ces trois couleurs sont complémentaires aux primaires.

Si les trois faisceaux lumineux sont centrés sur l'écran et en agissant sur chacun des trois diaphragmes, on peut obtenir toutes les couleurs du spectre. La trichromie additive est appliquée lors de la projection de diapositives, en cinéma et en télévision en couleur, car les luminophores des trois primaires s'illuminent sous le flux des électrons.

La trichromie soustractive

Si on place devant un projecteur donnant une lumière blanche, trois filtres respectivement rouge, bleu, vert, dont la courbe de transmittance n'est pas très étroite, aucune lumière n'est visible ; on obtient le noir.

Si on désire retrouver les trois couleurs fondamentales, on peut soustraire de la lumière blanche les couleurs complémentaires correspondantes (fig. 6.b).

Donc en plaçant devant une source blanche trois filtres d'efficacité réglable respectivement cyan (C), jaune (Y) et magenta (M), on peut obtenir toutes les couleurs du spectre y compris le noir (K). En supprimant les filtres on a le blanc.

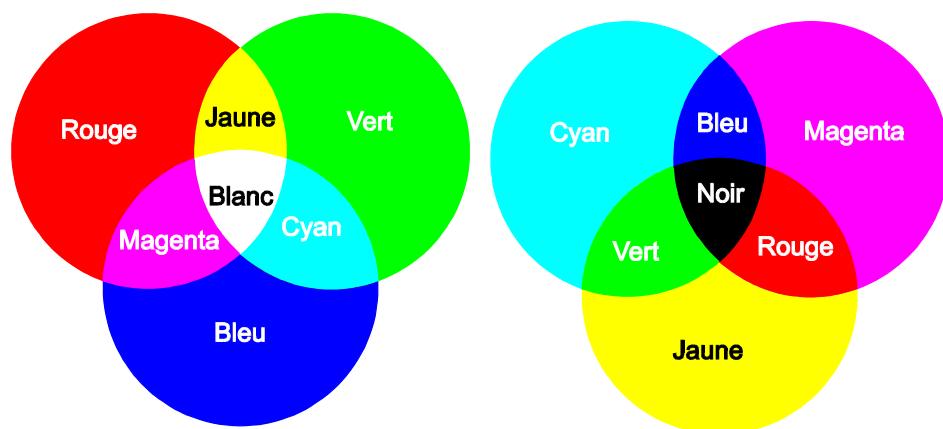


Fig. 6 Trichromies a. additive b. soustractive

La synthèse soustractive est appliquée en imprimerie et en peinture. Lorsque la toile est éclairée en lumière blanche, les peintures appliquées jouent le rôle de filtres et renvoient vers l'oeil une lumière composée par soustraction qui reconstitue la couleur désirée par l'artiste.

3.2. LE SYSTEME RGB DE LA CIE

Pour permettre la comparaison des indications fournies par les divers colorimètres employés, la Commission Internationale de l'Eclairage(CIE) a fixé conventionnellement un système de référence, dit RGB (ou plutôt RVB en bon français).

La CIE a choisi comme primaires, les trois radiations monochromatiques suivantes :

- Rouge: $\lambda_R = 700.0 \text{ nm}$ (filtre normalisé)
- Vert : $\lambda_G = 546.1 \text{ nm}$ (raie verte de l'arc au mercure)
- Bleu : $\lambda_B = 435.8 \text{ nm}$ (raie bleue de l'arc au mercure)

On peut prendre une certaine énergie de chacune de ces trois fondamentales telle qu'on obtienne par addition un blanc similaire au blanc E d'égale énergie défini plus haut (on notera l'indice W pour ce blanc pour ne pas confondre avec l'énergie E). On peut donc écrire pour ce blanc :

$$E_W = E_{RW} + E_{GW} + E_{BW}$$

Le système de coordonnées RGB a été établi en décidant que le blanc E devait se trouver aux coordonnées (1, 1, 1), soit

$$R_W = G_W = B_W = 1$$

On a évalué ensuite quelles étaient les coordonnées (R, G, B) de chaque composante monochromatique du spectre du blanc E. Pour ça, on a demandé à un échantillon de personnes de dire à partir de quand la pondération R, G, B des trois fondamentales leur donnait la même impression visuelle que la raie monochromatique qu'on leur montrait simultanément (on a évolué de 5 en 5 nm). On a ainsi établi, à un facteur près les fonctions de distribution $R(\lambda)$, $G(\lambda)$ et $B(\lambda)$ reprises au tableau suivant qui correspondent (à ce facteur près) aux coordonnées RGB de chaque raie. Elles indiquent par exemple que l'on aura une impression visuelle similaire à celle que produit la raie monochromatique à 525 nm en prenant $R_{525} = -0.08473$ de rouge, $G_{525} = 0.19113$ de vert, $B_{525} = 0.00830$ de bleu. On constate qu'il faut enlever du rouge (valeur négative) pour y arriver.

Pour voir du blanc E, on peut procéder de 2 façons (à un facteur près). On peut prendre, dans le système RGB, 1 de rouge, 1 de vert et 1 de bleu ; ou bien prendre, d'un point de vue énergétique, une valeur identique (par exemple 1) de chaque raie de l'ensemble du spectre. On doit donc avoir

$$\int_V R(\lambda) \cdot d\lambda = \int_V G(\lambda) \cdot d\lambda = \int_V B(\lambda) \cdot d\lambda = k$$

où k est simplement le facteur près entre les 2 façons d'avoir du blanc. Le calcul nous donne 3.7820 pour les sommes respectives des colonnes du tableau RGB correspondant à $R(\lambda)$, $G(\lambda)$ et $B(\lambda)$, sommes qu'on assimile aux intégrales à un facteur près.

Si on connaît le spectre énergétique $E(\lambda)$ d'un signal, alors on peut en déduire ses coordonnées RGB :

$$\begin{aligned} R &= \int_V E(\lambda) \cdot R(\lambda) \cdot d\lambda \\ G &= \int_V E(\lambda) \cdot G(\lambda) \cdot d\lambda \\ B &= \int_V E(\lambda) \cdot B(\lambda) \cdot d\lambda \end{aligned}$$

λ (nm)	R(λ)	G(λ)	B(λ)				
380	0.00003	-0.00001	0.00117	580	0.24526	0.1361	-0.00108
385	0.00005	-0.00002	0.00189	585	0.27989	0.11686	-0.00093
390	0.0001	-0.00004	0.00359	590	0.30928	0.09754	-0.00079
395	0.00017	-0.00007	0.00647	595	0.33184	0.07909	-0.00063
400	0.0003	-0.00014	0.01214	600	0.34429	0.06246	-0.00049
405	0.00047	-0.00022	0.01969	605	0.34756	0.04776	-0.00038
410	0.00084	-0.00041	0.03707	610	0.33971	0.03557	-0.0003
415	0.00139	-0.0007	0.06637	615	0.32265	0.02583	-0.00022
420	0.00211	-0.0011	0.11541	620	0.29708	0.01828	-0.00015
425	0.00266	-0.00143	0.18575	625	0.26348	0.01253	-0.00011
430	0.00218	-0.00119	0.24769	630	0.22677	0.00833	-0.00008
435	0.00036	-0.00021	0.29012	635	0.19233	0.00537	-0.00005
440	-0.00261	0.00149	0.31228	640	0.15968	0.00334	-0.00003
445	-0.00673	0.00379	0.3186	645	0.12905	0.00199	-0.00002
450	-0.01213	0.00678	0.3167	650	0.10167	0.00116	-0.00001
455	-0.01874	0.01046	0.31166	655	0.07857	0.00066	-0.00001
460	-0.02608	0.01485	0.29821	660	0.05932	0.00037	0
465	-0.03324	0.01977	0.27295	665	0.04366	0.00021	0
470	-0.03933	0.02538	0.22991	670	0.03149	0.00011	0
475	-0.04471	0.03183	0.18592	675	0.02294	0.00006	0
480	-0.04939	0.03914	0.14494	680	0.01687	0.00003	0
485	-0.05364	0.04713	0.10968	685	0.01187	0.00001	0
490	-0.05814	0.05689	0.08257	690	0.00819	0	0
495	-0.06414	0.06948	0.06246	695	0.00572	0	0
500	-0.07173	0.08536	0.04776	700	0.0041	0	0
505	-0.0812	0.10593	0.03688	705	0.00291	0	0
510	-0.08901	0.1286	0.02698	710	0.0021	0	0
515	-0.09356	0.15262	0.01842	715	0.00148	0	0
520	-0.09264	0.17468	0.01221	720	0.00105	0	0
525	-0.08473	0.19113	0.0083	725	0.00074	0	0
530	-0.07101	0.20317	0.00549	730	0.00052	0	0
535	-0.05316	0.21083	0.0032	735	0.00036	0	0
540	-0.03152	0.21466	0.00146	740	0.00025	0	0
545	-0.00613	0.21487	0.00023	745	0.00017	0	0
550	0.02279	0.21178	-0.00058	750	0.00012	0	0
555	0.05514	0.20588	-0.00105	755	0.00008	0	0
560	0.0906	0.19702	-0.0013	760	0.00006	0	0
565	0.1284	0.18522	-0.00138	765	0.00004	0	0
570	0.16768	0.17087	-0.00135	770	0.00003	0	0
575	0.20715	0.15429	-0.00123	775	0.00001	0	0
				780	0	0	0

Fonctions de distribution R(λ), G(λ) et B(λ) établies sur base de la vision de raies monochromatiques d'égale énergie (de 5 en 5 nm) par un échantillon de personnes

Exemple d'utilisation du tableau.

On peut avoir la même impression lumineuse que celle que procure une raie à 505 nm en construisant une lumière de coordonnées $R = -0.0812$, $G = 0.10593$, $B = 0.03688$

On remarque que pour y arriver, on peut prendre 10.6% du vert, y ajouter 3.7% du bleu et y soustraire 8.1% du rouge de référence. Soustraire n'est possible que s'il s'agit de filtrer des composantes énergétiques existantes, mais il n'est pas possible d'ajouter des composantes énergétiques négatives.

Relation entre coordonnées RGB et luminance

On trouve, à énergie égale et pour chaque longueur d'onde λ monochromatique, les coordonnées RGB dans le tableau précédent, tandis que dans le tableau XYZ présenté plus loin, on trouve en $Y(\lambda)$, la luminance relative $V(\lambda)$ correspondante. En particulier, pour les trois primaires, on a

coordonnées	$V(\lambda)$	
(0.0041, 0, 0)	0.0041	pour le rouge ($\lambda=700$ nm)
(0, 0.214, 0)	0.98	pour le vert (546.1 nm)
(0, 0, 0.295)	0.01775	pour le bleu (435.8 nm)

ce qui signifie, par exemple pour le vert et à un facteur près, que pour une proportion de 0.214 de vert, on a une luminance 0.9835. Pour réaliser du blanc, on a dit qu'il fallait prendre une proportion unitaire de chaque composante. La luminance du blanc est donc, en faisant la somme des luminances des trois composantes,

$$L_w = L_{rw} + L_{gw} + L_{bw} = \frac{0.0041}{0.0041} + \frac{0.98}{0.214} + \frac{0.01775}{0.295} = 1.000 + 4.5907 + 0.0601$$

Si maintenant on considère une lumière C quelconque de coordonnées (R, G, B), sa luminance est donnée (au même facteur près que pour le blanc) par

$$L_c = L_{rw} \cdot R + L_{gw} \cdot G + L_{bw} \cdot B = 1.000 \cdot R + 4.5907 \cdot G + 0.0601 \cdot B$$

Souvent, on souhaite dissocier la luminance de la **chromaticité**. On peut dire que la chromaticité se rapporte à la *direction* d'un vecteur dans l'espace RGB tandis que la luminance est fonction de sa longueur. Doubler le vecteur revient à doubler la luminance de la lumière correspondante sans en changer la chromaticité. Deux coordonnées peuvent suffire à déterminer la chromaticité. Considérons les coordonnées normalisées, ou coefficients trichromatiques r, g et b définis par

$$r = \frac{R}{R+G+B} , \quad g = \frac{G}{R+G+B} , \quad b = \frac{B}{R+G+B}$$

Il est évident que leur somme est toujours égale à 1. Par exemple pour le blanc E on obtient :

$$r_w = g_w = b_w = 0.33$$

En conséquence un couple de coordonnées suffit pour définir la chromaticité. Une couleur est définie par sa luminance et par deux coordonnées chromatiques. Le système RGB présente l'inconvénient d'avoir des valeurs négatives pour un grand nombre de couleurs. D'autre part, la luminance n'apparaît sur aucun axe. On lui préfère le système XYZ.

3.3. LE SYSTEME XYZ DE LA CIE

La CIE, par un simple changement de coordonnées, a défini le système XYZ qui a les caractéristiques suivantes:

- son origine est la même que celle du système RGB mais les axes XYZ ne sont pas perpendiculaires entre eux dans le système RGB
- les axes X et Z sont dans le plan de luminance nulle : $R \cdot L_{RW} + G \cdot L_{GW} + B \cdot L_{BW} = 0$ dès lors, seule la coordonnée Y rend compte de la luminance
- on prend $Y = R \cdot L_{RW} + G \cdot L_{GW} + B \cdot L_{BW}$ pour qu'Y soit la luminance
- les axes X, Y et Z sont tels que le cône de chromaticité passant par le **spectrum locus** (lieu des couleurs monochromatiques) soit inscrit au trièdre formé par les demi-axes positifs X, Y et Z.
- les valeurs des vecteurs unitaires U_X , U_Y et U_Z sont telles que le blanc E soit également situé sur la droite de coordonnées (k, k, k) dans le système XYZ

L'avant-dernier point implique que toute couleur réelle s'exprime par des coordonnées positives dans le système XYZ. En effet, toute couleur réelle a un spectre. Elle possède donc une somme de composantes spectrales et peut être obtenue par la somme de couleurs monochromatiques. Si toutes les couleurs monochromatiques ont des coordonnées XYZ positives, alors les couleurs qui en sont composées également.

Changement de coordonnées du système RGB au système XYZ

$$(R, G, B) \cdot \begin{pmatrix} \vec{U}_R \\ \vec{U}_G \\ \vec{U}_B \end{pmatrix} = (R, G, B) \cdot \begin{pmatrix} \text{matrice} \\ A \end{pmatrix} \cdot \begin{pmatrix} \vec{U}_X \\ \vec{U}_Y \\ \vec{U}_Z \end{pmatrix} = (X, Y, Z) \cdot \begin{pmatrix} \vec{U}_X \\ \vec{U}_Y \\ \vec{U}_Z \end{pmatrix} = (X, Y, Z) \cdot \begin{pmatrix} \text{matrice} \\ A \end{pmatrix}^{-1} \cdot \begin{pmatrix} \vec{U}_R \\ \vec{U}_G \\ \vec{U}_B \end{pmatrix}$$

$$\text{avec } \begin{pmatrix} \text{matrice} \\ A \end{pmatrix} = \begin{pmatrix} \cdot L_R \cdot \\ \cdot L_G \cdot \\ \cdot L_B \cdot \end{pmatrix} = \begin{pmatrix} 2,76889 & 1,00000 & 0,00000 \\ 1,75175 & 4,5907 & 0,0565 \\ 1,13016 & 0,0601 & 5,59429 \end{pmatrix}$$

On définit comme précédemment les coefficients trichromatiques :

$$x = \frac{X}{X + Y + Z} , \quad y = \frac{Y}{X + Y + Z} , \quad z = \frac{Z}{X + Y + Z}$$

Ils vérifient la relation : $x + y + z = 1$

La lumière blanche E a pour coordonnées :

$$x = y = z = 1/3 \text{ ou } 0,333$$

Le tableau suivant reprend les fonctions de distribution $X(\lambda)$, $Y(\lambda)$ et $Z(\lambda)$ qui peuvent se déduire des fonctions de distribution $R(\lambda)$, $G(\lambda)$ et $B(\lambda)$ par le changement de coordonnées précité.

λ (nm)	X(λ)	Y(λ) =V(λ)	Z(λ)		580	0.91629	0.86994	0.00164
380	0.00138	0.00005	0.00654		585	0.97864	0.81626	0.00139
385	0.00223	0.00007	0.01057		590	1.02633	0.75698	0.00109
390	0.00426	0.00013	0.02008		595	1.05666	0.69485	0.00094
395	0.00766	0.00023	0.03619		600	1.06216	0.63097	0.00078
400	0.01430	0.00038	0.06790		605	1.04558	0.56677	0.00057
405	0.02316	0.00064	0.11013		610	1.00259	0.50297	0.00033
410	0.04350	0.00118	0.20735		615	0.93838	0.44120	0.00022
415	0.07763	0.00216	0.37125		620	0.85443	0.38098	0.00019
420	0.13434	0.00399	0.64557		625	0.75137	0.32099	0.00009
425	0.21478	0.00725	1.03905		630	0.64240	0.26500	0.00002
430	0.28388	0.01160	1.38558		635	0.54189	0.21697	0.00002
435	0.32851	0.01683	1.62300		640	0.44795	0.17501	0.00002
440	0.34830	0.02299	1.74706		645	0.36078	0.13818	0
445	0.34807	0.02981	1.78255		650	0.28353	0.10699	0
450	0.33621	0.03802	1.77209		655	0.21869	0.08159	0
455	0.31865	0.04800	1.74410		660	0.16489	0.06101	0
460	0.29082	0.06000	1.66911		665	0.12125	0.04462	0
465	0.25107	0.07391	1.52807		670	0.08738	0.03199	0
470	0.19539	0.09099	1.28761		675	0.06362	0.02321	0
475	0.14208	0.11257	1.04188		680	0.04676	0.01700	0
480	0.09561	0.13898	0.81304		685	0.03288	0.01191	0
485	0.05799	0.16929	0.61624		690	0.02267	0.00819	0
490	0.03199	0.20797	0.46513		695	0.01583	0.00572	0
495	0.01470	0.25855	0.35334		700	0.01135	0.00410	0
500	0.00489	0.32297	0.27200		705	0.00805	0.00291	0
505	0.00240	0.40727	0.21230		710	0.00581	0.00210	0
510	0.00930	0.50293	0.15819		715	0.00409	0.00148	0
515	0.02911	0.60813	0.11166		720	0.00290	0.00105	0
520	0.06328	0.70994	0.07817		725	0.00204	0.00074	0
525	0.10958	0.79313	0.05723		730	0.00143	0.00052	0
530	0.16548	0.86195	0.04219		735	0.00099	0.00036	0
535	0.22574	0.91482	0.02981		740	0.00069	0.00025	0
540	0.29040	0.95394	0.02029		745	0.00047	0.00017	0
545	0.35968	0.98022	0.01342		750	0.00033	0.00012	0
550	0.43343	0.99491	0.00872		755	0.00022	0.00008	0
555	0.51214	1.00014	0.00575		760	0.00016	0.00006	0
560	0.59452	0.99492	0.00385		765	0.00011	0.00004	0
565	0.67842	0.97855	0.00274		770	0.00008	0.00003	0
570	0.76208	0.95196	0.00210		775	0.00003	0.00001	0
575	0.84246	0.91532	0.00183		780	0	0	0

Fonctions de distribution X(λ), Y(λ) et Z(λ) établies par changement de coordonnées appliqué sur R(λ), G(λ) et B(λ).
Notons que Y(λ) est similaire à la courbe de visibilité relative V(λ).

On trouve à la figure suivante le diagramme [x,y] appelé *diagramme de chromaticité* sur lequel est reporté le lieu des couleurs monochromatiques, le *spectrum locus* qui a pu être calculé sur base du tableau précédent. Le spectrum locus a une forme de fer à cheval. Il va de soi que

tous les vecteurs $[k.X, k.Y, k.Z]$ sont représentés par le même point $[X/(X+Y+Z), Y/(X+Y+Z)]$ dans le diagramme $[x,y]$. Le blanc E et toutes les nuances de gris qui lui correspondent sont évidemment aux coordonnées $(0.33, 0.33)$.

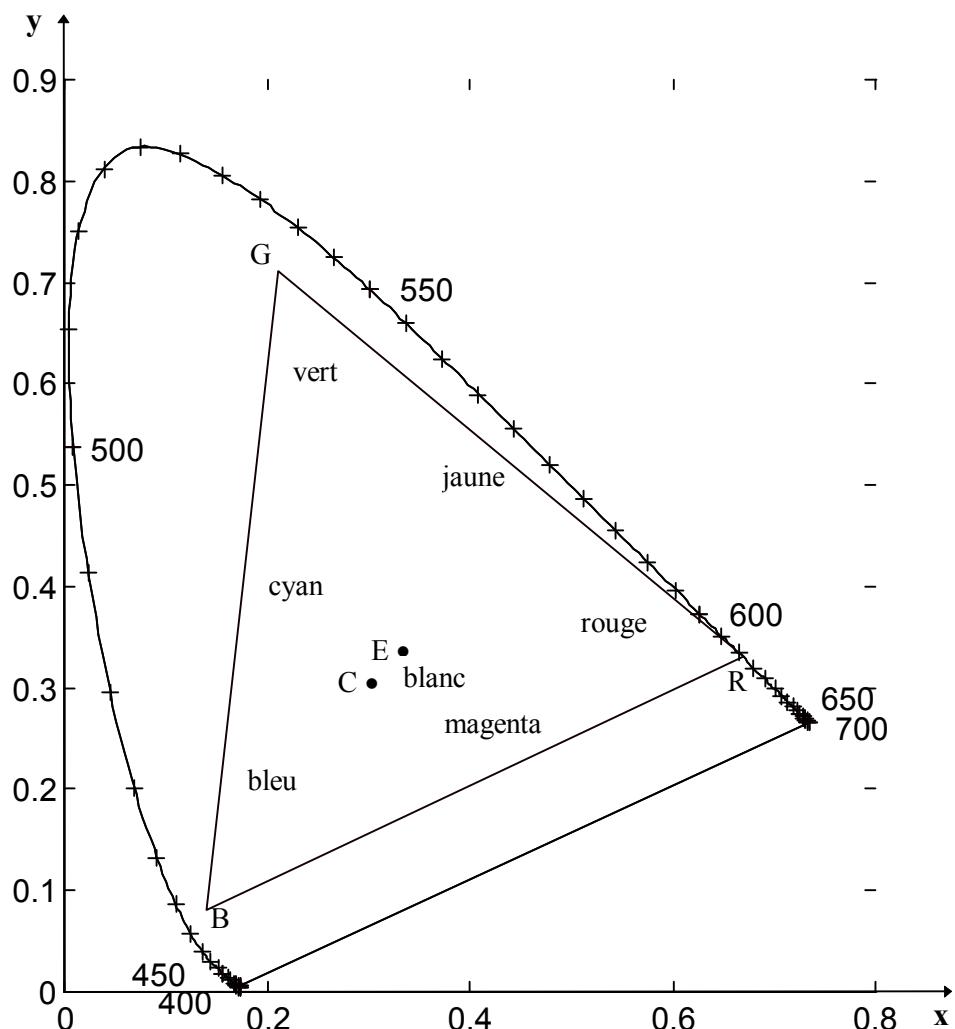


fig. 7 Diagramme de chromaticité

Les couleurs monochromatiques sont représentées sur le spectrum locus par des signes + pour les longueurs d'ondes évoluant de 5 en 5 nm. Les valeurs indiquées se réfèrent au signe + situé sur la même ligne.

MÉLANGE DE DEUX COULEURS

Toute couleur existante se trouve à l'intérieur de la zone délimitée par le spectrum locus et la droite joignant les extrémités de celui-ci. Cette droite comprend les pourpres saturés formés du mélange des deux lumières monochromatiques situées aux extrémités du spectre, violet et rouge.

On appelle lumières *monochromatiques complémentaires* deux lumières monochromatiques dont le mélange peut donner une lumière blanche de référence. Les couleurs du violet au cyan sont complémentaires des couleurs du jaune au rouge. Les verts sont couleurs

monochromatiques complémentaires des pourpres saturés (qui eux ne sont pas monochromatiques).

Une lumière *pure* ou *saturée* est une lumière qui ne comporte aucune trace de lumière blanche. Toute couleur C peut être obtenue par la somme d'une lumière blanche et d'une lumière pure ou saturée appelée *couleur dominante* de C et caractérisée par sa longueur d'onde. C'est elle qui rend compte de la *teinte* de C. L'ajout de blanc rend la couleur moins saturée, plus pâle (pastel) sans en changer la teinte. Si la couleur C est formée de blanc et de pourpre, la longueur d'onde de la couleur dominante n'a pas de sens. On considérera celle de la couleur complémentaire du pourpre.

On peut définir une couleur par ses coordonnées [x,y] ou par la longueur d'onde λ de la dominante et un *facteur de pureté*.

La *pureté colorimétrique* P_C exprime un lien entre luminances de la couleur et de la dominante

$$P_C = \frac{L_\lambda}{L_C} = \frac{L_\lambda}{L_\lambda + L_W}$$

Dans cette équation, l'indice W désigne le blanc E.

La *pureté d'excitation* P_E est définie à partir des coordonnées :

$$P_E = \frac{y_C - y_W}{y_\lambda - y_W} = P_C \cdot \frac{y_C}{y_\lambda}$$

La dernière égalité peut être montrée à l'aide d'une construction géométrique.

3.4. LE CHOIX DES PRIMAIRES EN TÉLÉVISION COULEUR

On a vu que la CIE avait défini trois primaires :

$$R = 700 \text{ nm}, G = 546.1 \text{ nm} \text{ et } B = 435.8 \text{ nm}$$

Pour la télévision en couleur la FCC a adopté pour le NTSC les primaires suivantes :

$$R' = 610 \text{ nm}, x = 0.67, y = 0.33$$

$$G' = 535 \text{ nm}, x = 0.21, y = 0.71$$

$$B' = 470 \text{ nm}, x = 0.14, y = 0.086$$

Celles-ci forment un triangle sur le diagramme de la figure 7. Tout écran basé sur ces primaires ne peut reproduire par addition que les couleurs situées à l'intérieur du triangle. Le rouge est monochromatique, mais le bleu et le vert sont légèrement désaturés.

Le blanc de référence choisi est le blanc C qui a pour coefficients :

$$x = 0.310 \text{ et } y = 0.316$$

Sa température de couleur est de 6500 K. On règle l'intensité des primaires pour obtenir le blanc C lorsque les trois primaires sont au maximum (100% ou 1). Le calcul de l'équation de la luminance est réalisé de façon similaire à ce qui a été fait pour les primaires de la CIE. On obtient l'équation suivante pour le lien entre composante E'_Y de luminance d'un signal et composantes E'_R , E'_G , E'_B .

$$E'_Y = 0.299 E'_R + 0.587 E'_G + 0.114 E'_B$$

Cette équation reste adoptée par convention pour les systèmes de télévision (notamment le PAL utilisé à la RTBF), bien que les primaires actuellement adoptées ne soient plus exactement les mêmes. En effet, l'UER a opté en 1970 pour les primaires suivantes :

$$R'' = 600 \text{ nm}, x = 0.64, y = 0,33$$

$$G'' = 546 \text{ nm}, x = 0.29, y = 0,60$$

$$B'' = 446 \text{ nm}, x = 0.15, y = 0,06$$

ainsi que pour le blanc de référence (D65) qui a pour coefficients :

$$x = 0.313 \text{ et } y = 0.329$$

Les luminophores utilisés dans les tubes écrans ont des coordonnées très proches des précédentes :

$$\text{Rouge : } x = 0.665, y = 0,34$$

$$\text{Vert : } x = 0.297, y = 0,617$$

$$\text{Bleu : } x = 0.150, y = 0,055$$

A partir de ces dernières coordonnées, on peut réaliser un diagramme de chromaticité sur écran de PC à l'aide de MatLab.

4. REALISATION DU DIAGRAMME DE CHROMATICITE SUR PC

Dans les notations qui suivent, on a utilisé la police `courrier new` pour représenter les instructions en langage MatLab. Les points 1. à 7. consistent à calculer les données essentielles du diagramme et à sauver celles-ci dans un fichier. Ils sont réalisés par le programme `XYDIAGMT.M`; l'affichage sur écran du diagramme est décrit à partir du point 8. Il est réalisé par le programme `XYDIAG.M`

1. Nombre de couleurs représentables

Le nombre de couleurs que l'on peut représenter sur un PC dépend

- *de la résolution de l'écran*
virtuellement infinie si les composantes R, G et B sont envoyées sous forme analogique, ce qui est le cas pour le standard VGA et ses variantes
- *de la carte gestionnaire d'écran* et notamment de la mémoire écran
une résolution 1024X768 en 256 couleurs nécessite environ 1Moctet de mémoire
- *du logiciel utilisé*

Il est fréquent actuellement de pouvoir employer 65536 couleurs (codage en 16 bits) ou même 16 millions de couleurs (24 bits), néanmoins, le logiciel MatLab 4.0 qui a été utilisé pour cette réalisation est limité à 256 couleurs. Il s'agit donc de déterminer le nombre de valeurs possibles pour chaque composante R , G et B en veillant à ce que le nombre total d'arrangements de celles-ci n'excède pas 256. On a choisi

$$nR = 7; \quad nG = 6; \quad nB = 6;$$

2. Calcul de la luminance des primaires TV

On peut exprimer les coordonnées $[X_A; Y_A; Z_A]$ d'une lumière A par la somme de trois vecteurs primaire $[x_P / y_P; 1; z_P / y_P]$ (avec P symbolisant R , G puis B) respectivement pondérés par des luminances Y_R , Y_G et Y_B . En particulier, pour le blanc C de référence, on a

$$\begin{bmatrix} x_R & x_G & x_B \\ y_R & y_G & y_B \\ z_R & z_G & z_B \end{bmatrix} \cdot \begin{bmatrix} 1/y_R & 0 & 0 \\ 0 & 1/y_G & 0 \\ 0 & 0 & 1/y_B \end{bmatrix} \cdot \begin{bmatrix} Y_R \\ Y_G \\ Y_B \end{bmatrix} = \frac{Y_C}{y_c} \cdot \begin{bmatrix} x_C \\ y_C \\ z_C \end{bmatrix}$$

c'est-à-dire, en nommant les matrices et en utilisant des notations plus proches de celles de MatLab

$$xyzRGB * diag(1 ./ yRGB) * YRGB = YC / yC * xyzC$$

2.1. On prend comme primaires ($xyzRGB$) celles qui correspondent en principe aux luminophores du tube écran du PC :

$$\begin{aligned} xRGB &= [0.665 \quad 0.297 \quad 0.150]; \\ yRGB &= [0.340 \quad 0.617 \quad 0.055]; \\ zRGB &= 1 - xRGB - yRGB; \\ xyzRGB &= [xRGB; yRGB; zRGB]; \end{aligned}$$

On considère comme blanc de référence le blanc D65. On sait que par définition du blanc de référence, il doit être obtenu en réglant les trois faisceaux R, G et B à 100%, soit pour

$$R_C = G_C = B_C = 1 \text{ et } Y_C = 1$$

Le blanc D65 a pour coordonnées $[x,y]$

$$xyzC = [0.313; \quad 0.329; \quad 1 - .313 - .329];$$

2.2. On en déduit les luminances des primaires ($YRGB$)

$$\begin{aligned} yC &= xyzC(2); \quad YC = 1; \\ YRGB &= inv(xyzRGB * diag(1 ./ yRGB)) * YC / yC * xyzC; \\ invxyzRGB &= inv(xyzRGB * diag(YRGB ./ yRGB')); \end{aligned}$$

Notons que pour une lumière Q quelconque, on a

$$\begin{bmatrix} x_R & x_G & x_B \\ y_R & y_G & y_B \\ z_R & z_G & z_B \end{bmatrix} \cdot \begin{bmatrix} 1/y_R & 0 & 0 \\ 0 & 1/y_G & 0 \\ 0 & 0 & 1/y_B \end{bmatrix} \cdot \begin{bmatrix} R \cdot Y_R \\ G \cdot Y_G \\ B \cdot Y_B \end{bmatrix} = \frac{Y_Q}{y_Q} \cdot \begin{bmatrix} x_Q \\ y_Q \\ z_Q \end{bmatrix}$$

c'est-à-dire

$$\begin{bmatrix} x_R & x_G & x_B \\ y_R & y_G & y_B \\ z_R & z_G & z_B \end{bmatrix} \cdot \begin{bmatrix} Y_R / y_R & 0 & 0 \\ 0 & Y_G / y_G & 0 \\ 0 & 0 & Y_B / y_B \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix} = \frac{Y_Q}{y_Q} \cdot \begin{bmatrix} x_Q \\ y_Q \\ z_Q \end{bmatrix}$$

En supposant $YQ / yQ = 1$, si on donne xQ et yQ , on peut trouver R , G et B

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \text{inv} \left(\begin{bmatrix} x_R & x_G & x_B \\ y_R & y_G & y_B \\ z_R & z_G & z_B \end{bmatrix} \cdot \begin{bmatrix} Y_R / y_R & 0 & 0 \\ 0 & Y_G / y_G & 0 \\ 0 & 0 & Y_B / y_B \end{bmatrix} \right) \cdot \begin{bmatrix} x_Q \\ y_Q \\ 1 - x_Q - y_Q \end{bmatrix}$$

c'est-à-dire

$$RGB = \text{invxyzRGB} * xyzQ$$

3. Tracé du Spectrum locus

Il faudra tracer les composantes normalisées pour $\lambda = 380:5:775 nm$ avec retour par une droite en $380 nm$

3.1. Chargement du tableau des fonctions de distribution contenant

- λ (longueurs d'ondes)
 - $XYZ_fd = [X(\lambda), Y(\lambda), Z(\lambda)]$
- ```
load colorim
```

#### 3.2. Passer de $XYZ(\lambda)$ à $xyz(\lambda)$

```
xyz_1 = XYZ_fd(1:80, :)'; ./ ([1;1;1]*sum(XYZ_fd(1:80, :)'));
xyz_1 = [xyz_1 xyz_1(:, 1)];
note: plot(xyz_1(1, :), xyz_1(2, :)) montrerait le spectrum locus dans le diag (x,y)
```

#### 3.3.interpolation par DFT pour avoir 512 composantes plutôt que 80

```
a=xyz_1(:, 1:80)';
aa=([a(1:80, :); a(80:-1:1, :)]);
faa=fft(aa);
faa1=[faa(1:80, :); zeros(1024-159, 3); faa(82:160, :)];
ifaal=real(ifft(faa1))*1024/160;
xyz_512 = ifaal([1:512 1], :)';
```

### 4. Calcul des composantes RGB pour tous les points x,y

On va se créer un rectangle constituant le diagramme (x,y). Des essais ont montré, pour un écran 800X600, que l'on pouvait prendre pour le rectangle 454 lignes X 403 colonnes pour que chaque élément de la matrice à créer corresponde exactement à 1 pixel de la figure (bitmap) :

$$\begin{aligned} y &= 0:0.9 \text{ en 454 lignes (NR)} \\ x &= 0:0.8 \text{ en 403 colonnes (NC)} \end{aligned}$$

$$NR = 454; NC = 403;$$

Compte tenu du risque de limitation de la mémoire vive disponible, on va créer la matrice en plusieurs fois, par zones de 50 lignes à la fois (nk), ce qui correspond à 160 Koctets environ. La

variable `zone` sera un vecteur ligne à 50 éléments Elle prendra successivement les valeurs 0:49, puis 50:99, puis 100:149, etc...

```
k = 0;
nk = 50;
for zone = [0:nk:NR-1; nk-1:nk:NR-2 NR-1],
 k = k+1;
 disp([k zone' NR])
```

- Construire la matrice `xx` contenant les coordonnées `x` de tous les pixels d'une zone de 50 lignes (50 lignes sauf pour la dernière zone)
 

```
x = .8*(0:NC-1) / (NC-1);
y = .9*(zone(1):zone(2))' / (NR-1);
xx = ones(size(y)) * x;
```
- Mettre toutes les coordonnées `x` contenues dans `xx` en un vecteur ligne appelé `x`
`nr = size(xx,1);`
`x1 = zeros(1,nr*NC); x1(:) = xx;`
`clear xx`
- De même, mettre les coord. `y` de tous les pixels dans un vecteur ligne appelé `y`
`yy = y * ones(size(x));`
`yl = zeros(1,nr*NC); yl(:) = yy;`
`clear yy`
- Calculer la matrice RGB à 3 lignes contenant les pondérations R, G, B de chaque pixel
 

```
RGB = invxyzRGB * [x1; y1; 1-x1-yl];
clear x1, clear y1
```
- Limiter ces composantes R, G, B à 100%
 

```
RGB = RGB ./ ([1 1 1]'*max(RGB));
```
- Retrancher les composantes négatives

Comme il n'est pas possible de produire les couleurs situées hors du triangle joignant les 3 primaires, on a décidé de représenter celles-ci par des couleurs plus sombres mais qui en fait n'ont pas de signification réelle. On retranche la somme des composantes négatives puis on met les composantes devenues négatives à 0. Ex:

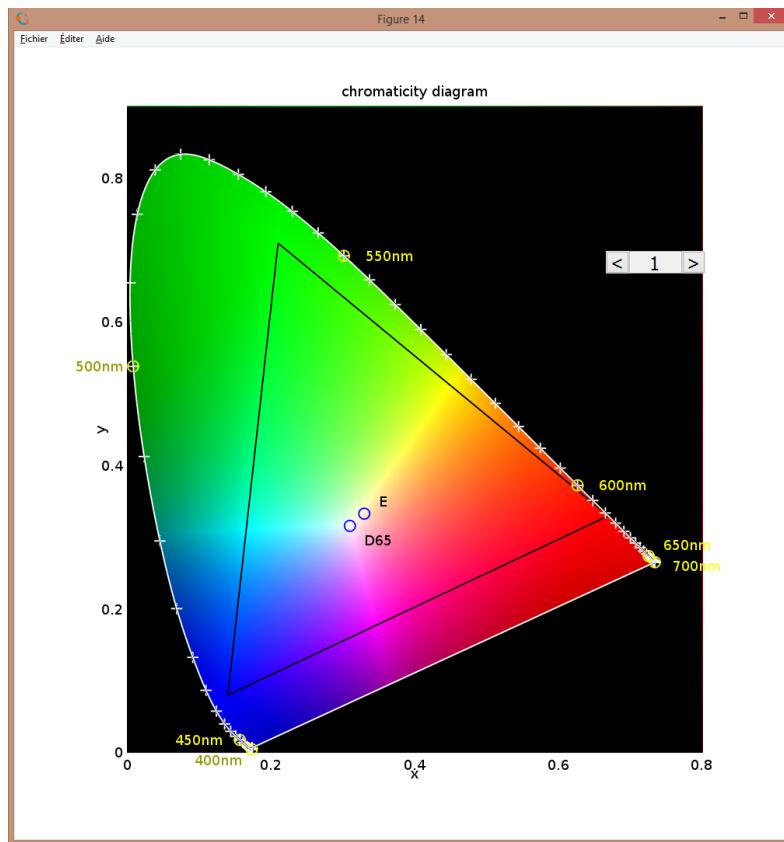
$$\begin{bmatrix} 1 \\ -0.2 \\ 0.1 \end{bmatrix} \text{ devient } \begin{bmatrix} 0.8 \\ -0.4 \\ -0.1 \end{bmatrix} \text{ puis } \begin{bmatrix} 0.8 \\ 0 \\ 0 \end{bmatrix} \quad \begin{bmatrix} 1 \\ -0.2 \\ -0.1 \end{bmatrix} \text{ devient } \begin{bmatrix} 0.7 \\ -0.5 \\ -0.4 \end{bmatrix} \text{ puis } \begin{bmatrix} 0.7 \\ 0 \\ 0 \end{bmatrix}$$

```
snRGB = sum(RGB.* (RGB<0));
RGB(1,:) = RGB(1,:) + snRGB;
RGB(2,:) = RGB(2,:) + snRGB;
RGB(3,:) = RGB(3,:) + snRGB;
clear snRGB;
```

- Mettre à 0 les composantes négatives
 

```
RGB = RGB .* (RGB>0);
```
- Remplacer RGB par 3 matrices R, G et B chacune à 50 lignes de 403 pixels (50 lignes sauf lors du dernier passage dans la boucle). En vue de quantifier les pondérations, les matrices R, G et B doivent contenir, non plus des valeurs comprises entre 0 et 1 mais des valeurs comprises respectivement dans les intervalles [0, nR-1], [0, nG-1] et [0, nB-1]. En pratique, on a arbitrairement étendu ces intervalles (ex: [0, nR-0.5] ) pour accroître le nombre de pixels saturés.

```
R = zeros(nr, NC);
R(:) = (nR-.5)*RGB(1,:);
G = zeros(nr, NC);
G(:) = (nG-0.5)*RGB(2,:);
B = zeros(nr, NC);
```



```
% Diagramme de chromaticité XYDIAG_oct.M utilise la matrice COULDIAG créée par XYDIAGMT.M
%-----
% (c) F. Gueuning, ECAM, 1996-2016 normalement à 7*6*6 couleurs (soit 252)
% Version pour Octave
% on utilise les matrices suivantes sauveées dans couldiag.mat par xydiagmt.m
% xRGB, yRGB coordonnées x,y des primaires TV
% xyzC coordonnées du blanc de référence
% blancTV nom du blanc utilisé
% COULDIAG couleurs de chaque pixel à l'intérieur du spectrum locus
% pour utilisation avec la fonction IMAGE
% colorRGB couleurs RGB pour colormap
% xyz_l coord. xyz pour chaque lambda de 5 en 5 nm
% xyz_512 interpolation des coord. précédentes pour 512 valeurs de lambda

%graphics_toolkit('fltk'); % fltk gère texture mais pas bouton
graphics_toolkit('qt'); % qt gère bouton mais pas texture
load colorim
load couldiag
xydiag15
hf = figure;
ha = axes('DataAspectRatio',[1 1 1]);
hold on

% diagramme de chromaticité
UD.hi = image([0 .8], [0 .9], RGB);

% ne fonctionne pas avec qt
%hi= surface([0 .8], [0 .9], -1e-3*ones(2) ...
% , 'cdata', RGB, 'facecolor', 'texturemap');

title('chromaticity diagram', 'handlevisibility', 'on')
set(hf,'Position',[137 0 525 562])
```

```

UD.Position = get(hf, 'Position');
 xlabel('x', 'handlevisibility', 'on')
 ylabel('y', 'handlevisibility', 'on')

% noircir la zone entourant le diagramme de chromaticité
UD.hfill = fill([xyz_512(1,:) 0 .8 .8 0 0 xyz_512(1,1)] ...
 , [xyz_512(2,:) 0 0 .9 .9 0 xyz_512(2,1)] ...
 , .001*ones(1, size(xyz_512,2)+6), 'k'); % Z pour masque légèrement au-dessus du rectangle coloré

% spectrum locus
plot3(xyz_512(1,:),xyz_512(2,:), .002*ones(1, size(xyz_512,2)), 'w')

% triangle RGB limité aux couleurs reproduites par l'écran
plot3([xRGB xRGB(1)], [yRGB yRGB(1)], zeros(1, size(xRGB,2)+1), 'k')

% repérage des blancs E et TV (C ou D65)
plot([xyzC(1);0.33],[xyzC(2);0.333], 'ob')
axis([0 .8 0 .9])

% repérage des longueurs d'ondes pour lumières monochromatiques
plot(xyz_l(1,:),xyz_l(2,:),'w+', 'handlevisibility', 'on')
plot(xyz_l(1,5:10:65),xyz_l(2,5:10:65), 'oy', 'handlevisibility', 'on')

% Sous octave, normalized ne modifie pas la taille absolue des caractères si la fenêtre grandit
UD.ht(9)=text(xyz_l(1,5)-.08,xyz_l(2,5)-.015,sprintf('%dnm',l(5)), 'color', [.6 .6 0]);
UD.ht(8)=text(xyz_l(1,15)-.09,xyz_l(2,15),sprintf('%dnm',l(15)), 'color', 'y');
UD.ht(7)=text(xyz_l(1,25)-.08,xyz_l(2,25),sprintf('%dnm',l(25)), 'color',[.6 .6 0]);
UD.ht(6)=text(xyz_l(1,35)+.03,xyz_l(2,35),sprintf('%dnm',l(35)), 'color','y');
UD.ht(5)=text(xyz_l(1,45)+.03,xyz_l(2,45),sprintf('%dnm',l(45)), 'color','y');
UD.ht(4)=text(xyz_l(1,55)+.02,xyz_l(2,55)+.015,sprintf('%dnm',l(55)), 'color','y');
UD.ht(3)=text(xyz_l(1,65)+.025,xyz_l(2,65)-.005,sprintf('%dnm',l(65)), 'color','y');

UD.ht(2)=text(xyzC(1)+.02,xyzC(2)-.02,blancTV);
UD.ht(1)=text(0.33+.02,0.33+.02,'E');
hold off

% modification de la luminance
Y = 1; % Luminosité à donner au diagramme de chromaticité, entre 0 et 1,
 % sera modifiée par les Pushbutton UD.hUI(1), UD.hUI(2) et UD.hUI(3)

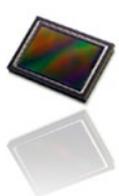
ModifLum = 'UD=get(gcf,"userdata"); set(UD.hi, "CData", Y*RGB)';
UD.hUI(1) = uicontrol('Style','Pushbutton','Position',[402 402 16 16],'String', '<', ...
 'Callback', ['Y=Y-.05; if Y<0, Y=0; end, set(UD.hUI(2),"String",num2str(Y)), ' ModifLum]');
set(UD.hUI(1),'FontUnits', 'normalized')
UD.hUI(2) = uicontrol('Style','Edit','Position',[418 402 36 16], 'String', ' 1',...
 'Callback', ['Y = str2num(get(UD.hUI(2),"String")); ' ModifLum]);
set(UD.hUI(2), 'FontUnits', 'normalized')
UD.hUI(3) = uicontrol('Style','Pushbutton','Position',[454 402 16 16],'String', '>', ...
 'Callback', ['Y=Y+.05; if Y>1, Y=1; end, set(UD.hUI(2),"String",num2str(Y)), ' ModifLum]');
set(UD.hUI(3), 'FontUnits', 'normalized')

for k=3:-1:1
 UD.UlcontPos{k} = get(UD.hUI(k), 'Position');
end

Pos = get(gcf,'Position');
set(findobj(gca, '-depth', 10, 'type','line'), 'markersize', 8*Pos(4)/UD.Position(4))
set(findobj(gca, '-depth', 10, 'type','line'), 'linewidth', 1*Pos(4)/UD.Position(4))
set(hf, 'userdata', UD)
% Resize characters because not made by Octave (in version 4.0.1)
set(ha, 'units', 'normalize', 'fontunits', 'normalize');
set(hf, 'resizefcn', 'resizefont')

```

# La photo en faits



comprendre la photo numérique

Accueil

Sommaire

Simulateur d'appareil photo

Calculateurs

Exprimez vous

À propos/Contact

Sites et blogs à voir



G+ 140

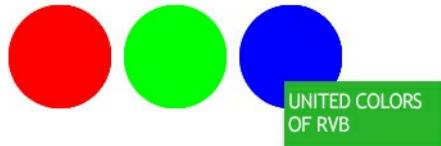
J'aime 386

Tweeter



17 mai 2013

## RVB, CMJN, TSL: explication et conversion des modes de couleurs



Cet article présente différentes manières de représenter les couleurs, nous regarderons le système RVB utilisé par les écrans et les capteurs des appareils photo, le système CMJN utilisé en imprimerie ou encore le système TSL utilisé en infographie.

Quelques généralités comme la réponse de l'oeil à une couleur seront également précisées.

En fin d'article un convertisseur vous permet de passer d'une mode de couleurs à un autre.

Bienvenue sur La Photo En Faits !

Un site pour comprendre la photo, par les faits.

Ici vous pourrez:

- Comprendre le fonctionnement de votre matériel
- Assimiler la technique au travers d'explications claires
- Découvrir l'univers de la photo et de ses plus grands photographes
- Apprendre à créer une image forte
- Utiliser les différents simulateurs pour parfaire vos connaissances

Vous pouvez suivre l'actualité du site sur les réseaux sociaux ainsi que sur [ma page perso Google+](#).

Merci pour la visite et à bientôt,

Pierre.

Remarque: il existe d'autres systèmes (Lab, CIEXYZ,...) mais ils sont relativement difficiles à appréhender, j'en ferai un article à part si besoin.

### Résumé de l'article

#### Qu'est-ce que le RVB ?

Un mode de représentation des couleurs utilisé par un écran ou un capteur d'appareil photo, la couleur y est décomposée selon ses composantes Rouge, Verte et Bleue (par synthèse additive).

#### Qu'est-ce que le CMJN ?

Ce format de couleur est utilisé en imprimerie, il permet de créer des couleurs à partir de composantes Cyan, Magenta, Jaune et Noire (par synthèse soustractive).

#### Qu'est-ce que le TSL ?

Une représentation des couleurs selon la Teinte, la Saturation et la Luminosité, utilisé notamment en infographie.

#### Qu'est-ce que la Luminance ?

La luminance mesure l'intensité de la lumière, elle peut être calculée en faisant la moyenne des composantes RVB ou en appliquant des coefficients à chaque composante, pour être fidèle au ressenti de l'oeil.

#### Peut-on convertir les différents modes de couleur ?

Oui, mais les différents modes n'offrant pas les mêmes possibilités il y a risque de modifier les couleurs.

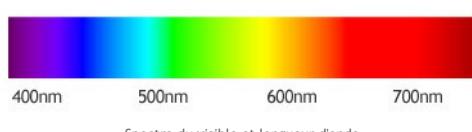
Téléchargez l'extension LPEF pour analyser des photos avec Chrome



- EXIF
- Histogrammes
- Pipette
- Zoom
- Repères
- et plus encore !

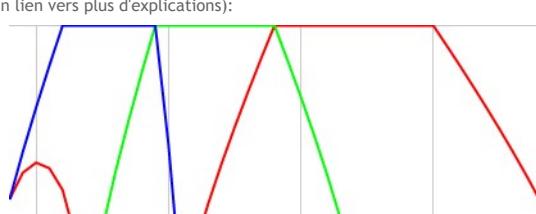
## Introduction: spectre du visible et perception par l'oeil

La lumière est une onde électromagnétique dont la longueur d'onde définit la couleur:



Ce spectre va de 380nm (limite des ultra-violets) à 780nm (limite des infra-rouges).

L'oeil possède des cônes: des récepteurs au Rouge, Vert et au Bleu (avec des pics respectivement à 564nm, 533nm et 437nm) lui permettant de distinguer les différentes couleurs, en effet on peut décomposer le spectre du visible en fonction de ces trois composantes (les valeurs sont approximatives, voir en fin d'article un lien vers plus d'explications):



### Articles les plus lus:

[Ouverture, vitesse et ISO: le triangle d'exposition](#)

[Choisir la résolution \(dpi\) d'impression d'une photo](#)

[La Profondeur De Champ: définition, formule et calculateur](#)

[Régler l'autofocus](#)

[100 \(et +\) Photographes Célèbres à découvrir](#)

[Une mise au point sur l'Autofocus](#)

[10 Astuces pour massacrer une photo en post prod](#)

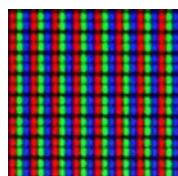
[Comparatif des reflex Canon, Nikon, Pentax et Sony](#)



## Le modèle RVB

Les capteurs des appareils photos et les écrans utilisent le même principe pour enregistrer ou créer une couleur, ils la décomposent selon les 3 couleurs primaires:

- R: Rouge
- V: Vert
- B: Bleu



Gros plan sur les pixels d'un écran

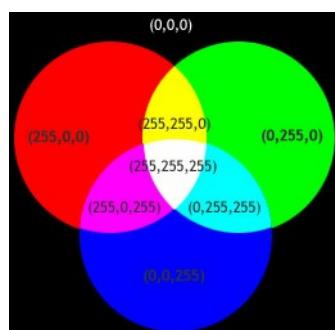
Basiquement, un écran est composé de pixels, chaque pixel intègre 3 diodes (toujours rouge, verte et bleue), en modulant l'intensité lumineuse de chaque diode on peut afficher n'importe quelle couleur visible, on parle de **synthèse additive**: le mélange de la lumière des 3 diodes crée la couleur.

Le nombre de niveaux possibles pour chaque composante est défini par le nombre de bits utilisés pour enregistrer les valeurs:

- 8 bits autorisent 256 niveau par composante, soit  $256 \times 256 \times 256 = 16$  millions de couleurs.
- 12 bits autorisent 4'096 niveaux par composante, soit plus de 60 milliards de couleurs.

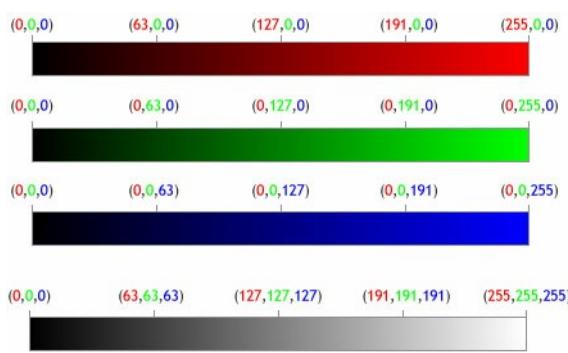
*Pour simplifier les choses, je considérerai un codage sur 8 bits dans la suite de l'article.*

Quelques exemples de couleurs en RVB:



L'addition des composantes RVB permet de créer les autres couleurs

Dans cette illustration toutes les composantes actives sont à leur valeur maximale (255), mais changer cette valeur permet de moduler la luminance:



Variation de la luminance des couleurs primaires et du gris

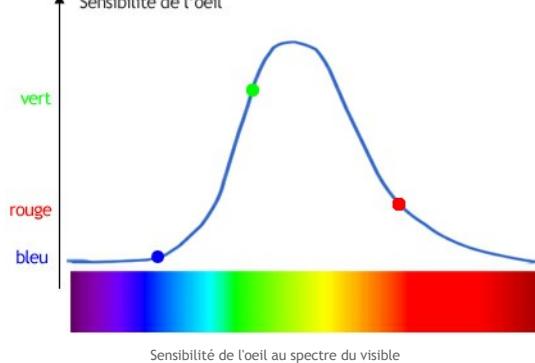
Une couleur RVB peut être représentée en hexadécimal sous la forme #RRVVBB avec RR, VV, BB les valeurs du rouge, du vert et du bleu en hexa.

## Luminance d'une couleur

La luminance L caractérise l'intensité lumineuse d'une source de lumière, en première approche on peut la calculer en faisant la moyenne des composantes RVB:

$$L = (R+V+B) / 3$$

Mais il faut tenir compte du fait que l'œil perçoit différemment la luminosité en fonction de la couleur:



Pour en revenir aux diodes d'un écran: même si elles sont allumées à la même intensité la led verte semblera plus lumineuse que la rouge et cette dernière plus lumineuse que la bleue.

On tient compte de ce phénomène en utilisant des coefficients pondérateurs dans le calcul de la luminance:

$$L = 0,3.R + 0,6.V + 0,1.B$$

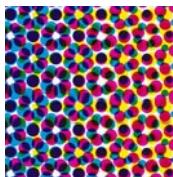
A noter que dans un logiciel tel que Photoshop l'histogramme RVB est calculé avec la première formule, l'histogramme de luminosité avec la seconde.

## Le modèle CMJN

### Définition du CMJN

Vous vous en rendez compte chaque fois que vous devez acheter de nouvelles cartouches pour votre imprimante à jet d'encre: on utilise couramment 4 couleurs pour réaliser une impression (on parle de **quadrichromie**):

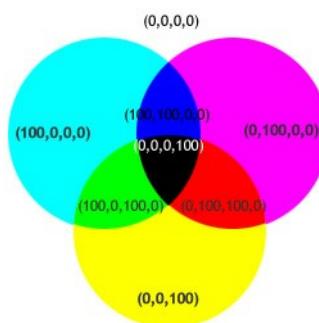
- C: Cyan (0,255,255)
- M: Magenta (255,0,255)
- J: Jaune (255,255,0)
- N: Noir (0,0,0)



Gros plan sur une impression en quadrichromie

Contrairement à un écran qui émet de la lumière, un tirage imprimé se contente de retenir des couleurs à lumière (supposée) blanche qui l'éclaire, on parle de **synthèse soustractive**, par exemple le cyan retient le rouge, le jaune retient le bleu.

Ici, on ne parle plus en bits mais en pourcentage, 0% si la couleur n'est pas utilisée, 100% dans le cas opposé. Cyan Magenta et Jaune sont les 3 couleurs primaires de la synthèse soustractive.



La soustraction des composantes CMJN permet de créer les autres couleurs

Dans cet exemple le noir pourrait être généré en utilisant CMJ à 100% et en gardant le noir à 0%, mais vous imaginez l'impact sur la consommation de vos cartouches d'encre.. dans la même logique on utilise plus ou moins de noir pour les autres couleurs.

### Utilité

On utilise le mode CMJN lorsque la finalité du travail est une **impression** en quadrichromie, en effet le CMJN ne permet pas de représenter toutes les couleurs du RVB, lors de la conversion de RVB en CMJN il y a **dégradation des couleurs**, il est donc important de travailler le rendu du tirage en CMJN pour avoir une impression fidèle à ce que l'on a à l'écran.

### Conversion de RVB <=> CMJN

Toujours pour 8 bits, les formules sont les suivantes:

$$\text{RVB} \Rightarrow \text{CMJN}$$

$$\text{CMJN} \Rightarrow \text{RVB}$$

$$N = \min(255-R, 255-V, 255-B) \quad R = \frac{255}{100} \times \left( 100 - C \times \left( 1 - \frac{N}{100} \right) - N \right)$$

$$C = 100 \times \frac{255 - R - N}{255 - N}$$

$$V = \frac{255}{100} \times \left( 100 - M \times \left( 1 - \frac{N}{100} \right) - N \right)$$

$$M = 100 \times \frac{255 - V - N}{255 - N}$$

$$B = \frac{255}{100} \times \left( 100 - J \times \left( 1 - \frac{N}{100} \right) - N \right)$$

$$J = 100 \times \frac{255 - B - N}{255 - N}$$

Attention: ces formules ne représentent qu'un cas théorique parfait, dans la vraie vie on applique des coefficients en fonction du papier, de l'encre, du taux max de noir que l'on veut utiliser,..

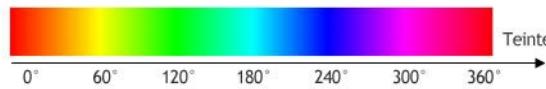
## Le modèle TSL

### Définition du TSL

Ce modèle est intéressant car il a une approche de la couleur plus intuitive que RVB ou CMJN, on a:

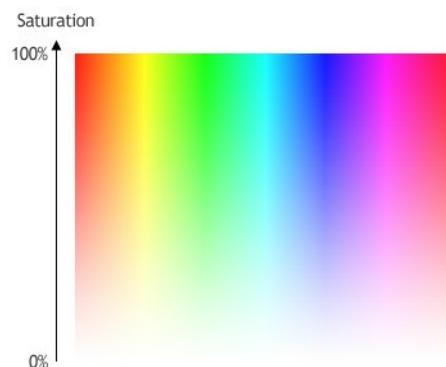
- T: La Teinte de la couleur
- S: La Saturation de la couleur
- L: la Luminosité de la couleur

La **teinte** représente une couleur "pure", elle est donnée en degré et vaut  $0^\circ$  pour le rouge,  $120^\circ$  pour le vert,  $240^\circ$  pour le bleu, on revient finalement sur le rouge à  $360^\circ$ :



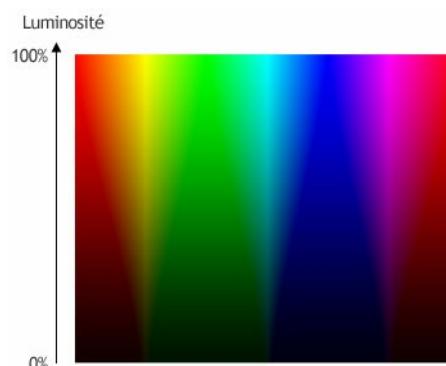
Variation de la Teinte dans le format TSL

La **saturation** représente "l'intensité" de la teinte, elle varie de 0% (gamme de gris allant du blanc au noir en fonction de la luminosité) à 100% (intensité maximale de la teinte):



Variation de la Saturation dans le format TSL (la luminosité est constante à 100%)

Pour finir, la **luminosité** donne une idée de la quantité de noir (ou de blanc en fonction de la saturation) ajoutée à la teinte, elle est également donnée en pourcentage:



Variation de la Luminosité dans le format TSL (la saturation est constante à 100%)

### Conversion de RVB <=> TSL

Pour faire les calculs il faut définir de nouvelles variables: M et m sont les max et min des composantes RVB, i et X sont des variables uniquement utilisées pour le calcul.

RVB => TSL

$$M = \max(R, V, B)$$

TSL => RVB

$$M = L \times \frac{255}{100}$$

$$m = \min(R, V, B)$$

$$m = \left( 100 - S \right) \times L \times \frac{255}{10000}$$

$$C = M - m$$

$$C = M - m$$

$$\text{Si } M=R : T = 60 \times \text{mod} \left( \left( \frac{V-B}{C} \right), 6 \right)$$

$$i = \frac{T}{60}$$

$$\text{Si } M=V : T = 60 \times \left( \left( \frac{B-R}{C} \right) + 2 \right)$$

$$X = C \times \text{mod}(i, 2)$$

$$\text{Si } M=B : T = 60 \times \left( \left( \frac{R-V}{C} \right) + 4 \right)$$

$$\begin{aligned} \text{Si } i < 1 : (R, V, B) &= (C+m, X+m, m) \\ \text{Si } i [1, 2] : (R, V, B) &= (X+m, C+m, m) \end{aligned}$$

$$S = 100 \times \frac{C}{M}$$

$$\begin{aligned} \text{Si } i [2, 3] : (R, V, B) &= (m, C+m, X+m) \\ \text{Si } i [3, 4] : (R, V, B) &= (m, X+m, C+m) \end{aligned}$$

$$L = 100 \times \frac{M}{255}$$

$$\begin{aligned} \text{Si } i [4, 5] : (R, V, B) &= (X+m, m, C+m) \\ \text{Sinon} : (R, V, B) &= (C+m, m, X+m) \end{aligned}$$

On peut voir la **saturation** comme un écart entre la couleur considérée et une couleur neutre (un niveau de gris), les gris ont donc une saturation nulle, pour être très saturée une couleur doit avoir une composante RVB très grande et une autre très petite.

La **luminosité** représente la "distance" entre la couleur considérée et le noir, pour être très lumineuse une couleur doit avoir une composante très élevée.

## Liens

Des infos sur la [conversion d'une longueur d'onde en RVB](#).

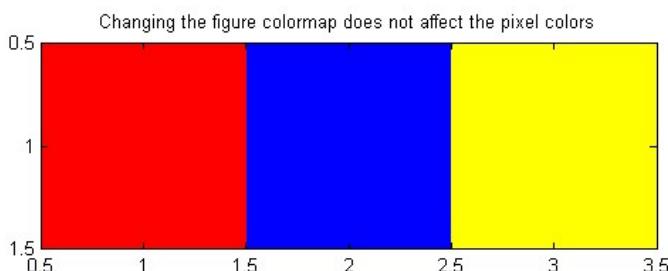
Adobe vous parle des [couleurs](#) et de leur [gestion](#).

## Convertisseur

- Ce convertisseur vous permet de convertir les couleurs **RVB**, **CMJN** et **TSL**.
- Le **CMJN** est simplifié, il ne tient pas compte du type de papier, d'encre,..
- L'information est également donnée en hexadécimal.
- Deux types de luminances sont calculées:
  - Luminance moyenne =  $R/3 + V/3 + B/3$ .
  - Luminance pondérée =  $0.3R + 0.6V + 0.1B$ , plus fidèle au ressenti de l'oeil.
- Faire varier la luminance donne une couleur en **nuance de gris**.
- La couleur choisie est affichée en haut du tableau.

**Conversion RVB/CMJN/TSL**

| RVB                                        | CMJN                       | TSL                            |
|--------------------------------------------|----------------------------|--------------------------------|
| R [0-255] <input type="text" value="64"/>  | C (%) <input type="text"/> | T [0-360] <input type="text"/> |
| V [0-255] <input type="text" value="128"/> | M (%) <input type="text"/> | S (%) <input type="text"/>     |
| B [0-255] <input type="text" value="192"/> | J (%) <input type="text"/> | L (%) <input type="text"/>     |
|                                            | N (%) <input type="text"/> |                                |
| Hex <input type="text"/>                   |                            |                                |
| Luminance (% moyen) <input type="text"/>   |                            |                                |
| Luminance (% percu) <input type="text"/>   |                            |                                |



### Indexed images

If the image CData is two-dimensional, then the CData values are treated as lookup indices into the figure's colormap. As an example, let's use an indexed image that ships with MATLAB, clown.mat (Ned's favorite).

```
s = load('clown') % This is the functional form of load. This form returns
% a structure whose fields are the variables stored
% in the MAT-file.
```

```
s =
X: [200x320 double]
map: [81x3 double]
caption: [2x1 char]
```

The X and map variables stored in clown.mat are both necessary to display the image correctly. X contains the pixel values, and map contains the associated colormap.

To get the color of the (5,5) pixel, first see what X(5,5) is:

```
s.X(5,5)
```

```
ans =
61
```

Then use that value as a row index into the colormap matrix, map:

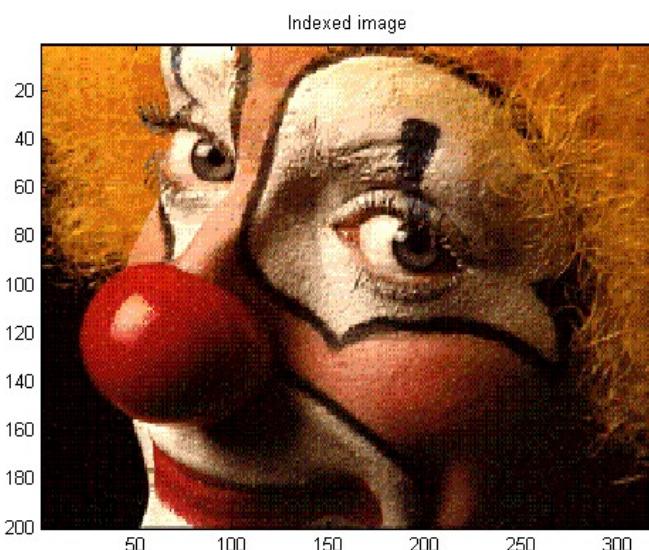
```
s.map(61,:)
```

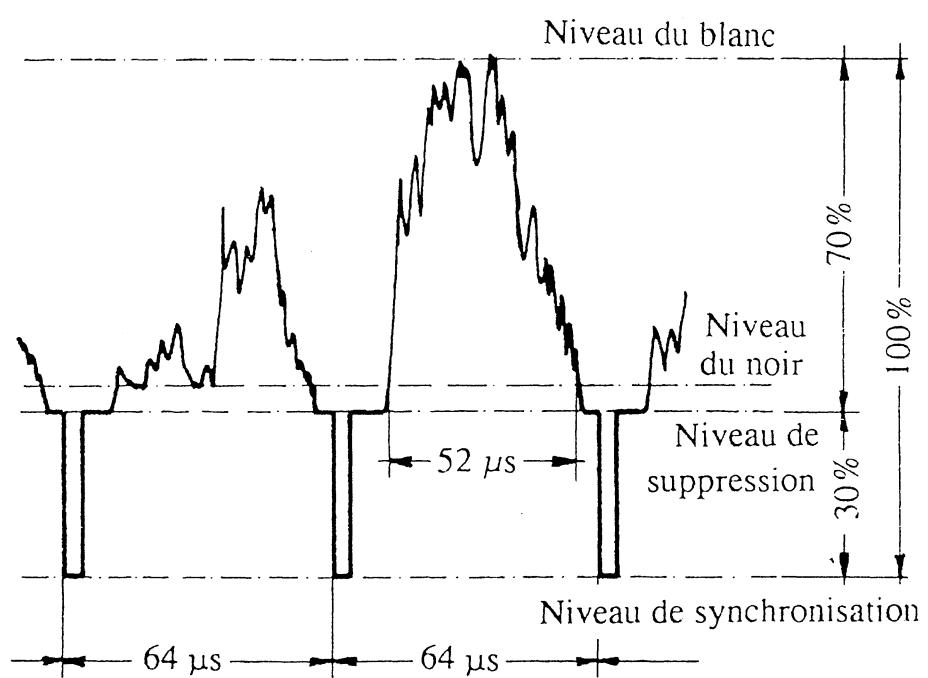
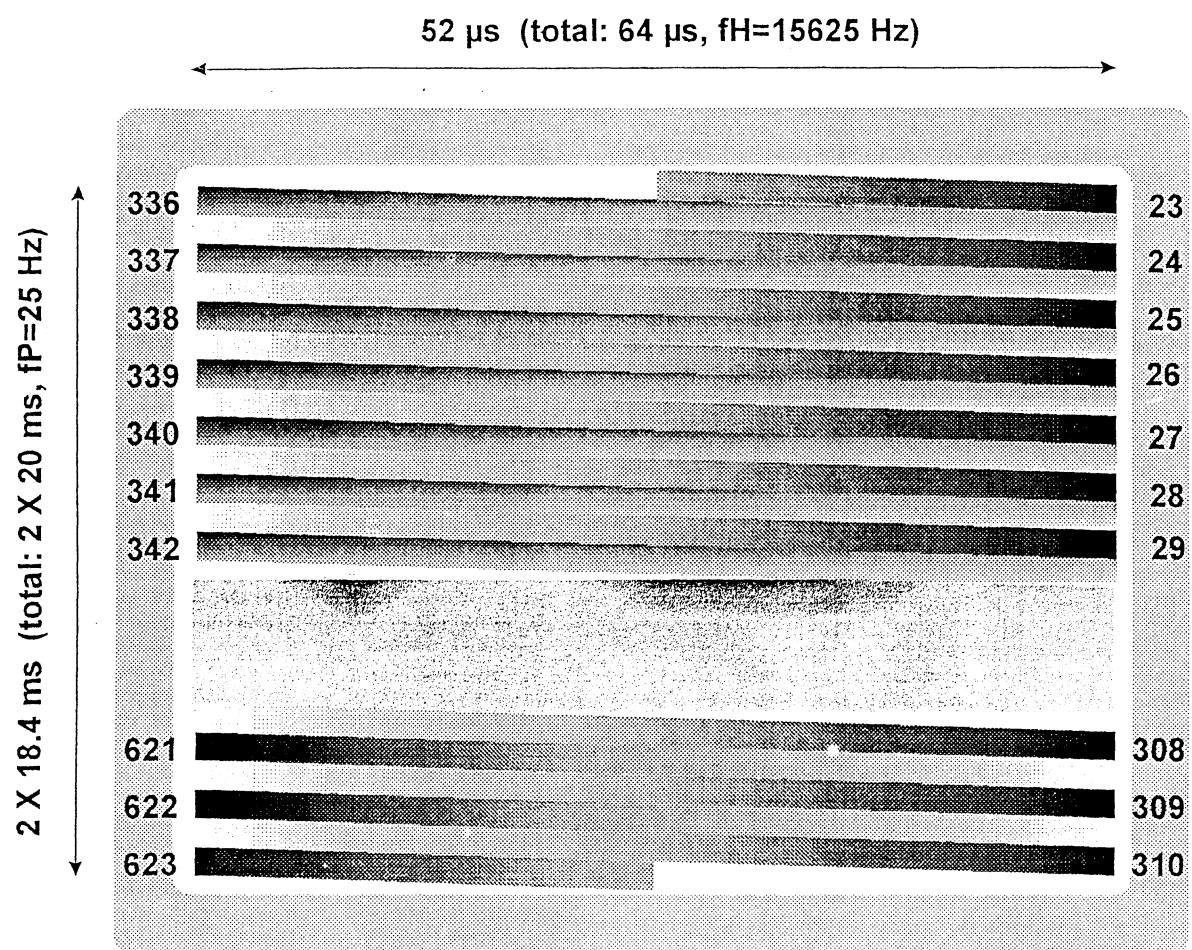
```
ans =
0.9961 0.5781 0.1250
```

So the (5,5) pixel has a lot of red, some green, and a small amount of blue.

Displaying the image requires two MATLAB commands, one to create the image and one to set the figure's colormap:

```
image(s.X)
colormap(s.map)
title('Indexed image')
```





|                   | R    | G    | B    | Y    | B-Y   | R-Y   | G-Y   |
|-------------------|------|------|------|------|-------|-------|-------|
| Noir              | 0    | 0    | 0    | 0    | 0     | 0     | 0     |
| Sat. Bleu         | 0    | 0    | 1    | 0.11 | 0.89  | -0.11 | -0.11 |
| Sat. Vert         | 0    | 1    | 0    | 0.59 | -0.59 | 0.41  | 0.41  |
| Cyan              | 0    | 1    | 1    | 0.70 | 0.30  | -0.70 | 0.30  |
| Sat. Rouge        | 1    | 0    | 0    | 0.30 | -0.30 | 0.70  | -0.30 |
| Magenta           | 1    | 0    | 1    | 0.41 | 0.59  | 0.59  | -0.41 |
| Jaune             | 1    | 1    | 0    | 0.89 | -0.89 | 0.11  | 0.11  |
| Blanc             | 1    | 1    | 1    | 1    | 0     | 0     | 0     |
| Gris moyen        | 0.5  | 0.5  | 0.5  | 0.5  | 0     | 0     | 0     |
| Sat. Rouge sombre | 0.5  | 0    | 0    | 0.15 | -0.15 | 0.35  | -0.15 |
| Rouge non saturé  | 0.50 | 0.15 | 0.15 | 0.3  | -0.15 | 0.35  | -0.15 |

4ème quadrant, près de l'axe B-Y

3ème quadrant

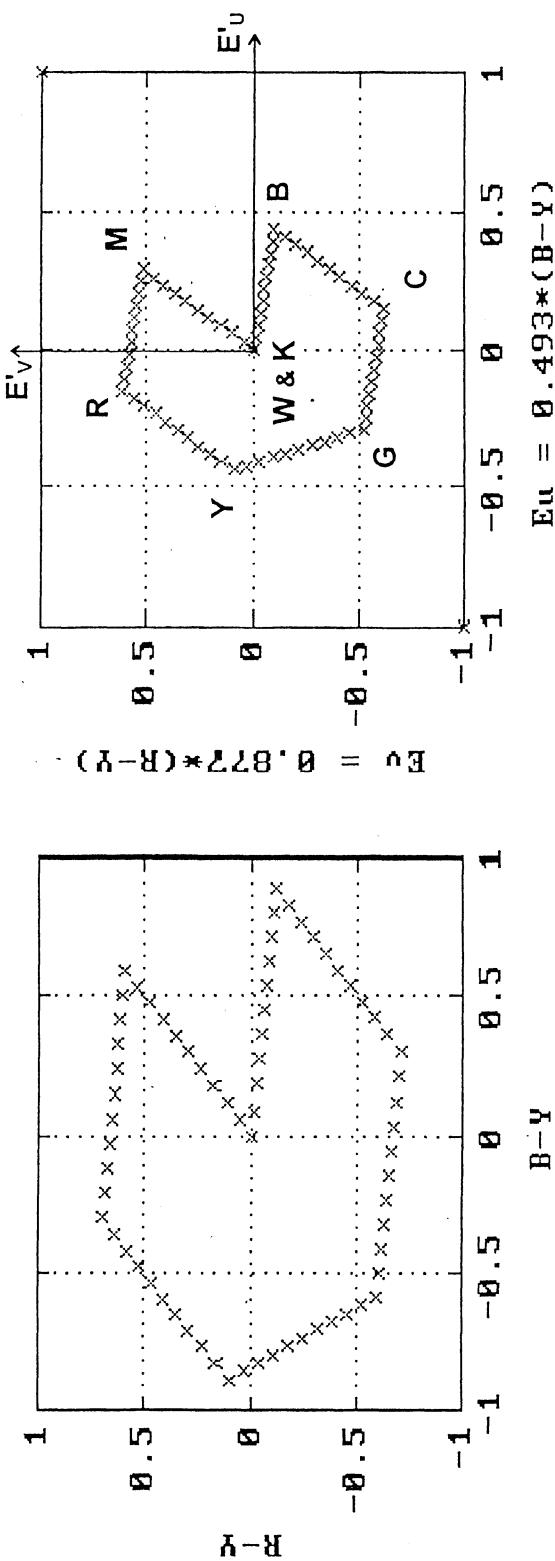
somme des 2 précédents

2ème quadrant

$$B-Y = R-Y = 0$$

idem

moitié du vecteur rouge lumineux  
même B-Y et R-Y que précédent



sera remplacé par un détecteur à l'état solide. Ces caméras seront légères, économiques et consommant peu, des tensions d'alimentation de faible valeur.

Le fonctionnement de ces dispositifs de prise de vue met en œuvre trois fonctions principales :

- la transformation des photons en charges électriques (détection) ;
- l'intégration de ces charges pendant la période image (accumulation) ;
- l'analyse de ces charges qui se traduit par un signal vidéo qui dépend du balayage utilisé.

#### DÉTECTION ET ACCUMULATION

Le silicium avec sa largeur de bande interdite de 1,12 eV permet d'obtenir une bonne réponse spectrale de 400 à 1000 nm. On observe une efficacité quantique très élevée rendant possible une très bonne détection des photons incidents.

Les cellules photoconductrices en technologie MOS sont très petites, quelques  $\mu\text{m}$  de côté ; le photon incident est très absorbé par le silicium et transformé en une paire électron-trou. Les charges ainsi créées pendant un temps d'intégration  $t$ , sont réparties et piégées par l'action du champ électrique appliquée aux bornes de la cellule. Ces charges sont une fonction linéaire de l'énergie reçue, jusqu'à la saturation ; le gamma est égal à 1. Elles sont accumulées pendant le temps  $t$  qui correspond à l'analyse d'une image, par exemple 40 ms pour 25 images/seconde ; de la même manière que sur la cible d'un tube analyseur.

#### ANALYSE

Ces charges sont ensuite stockées dans des registres verticaux et décalées ligne par ligne vers le registre horizontal. Chaque ligne d'image stockée dans ce registre est ensuite décalée point par point vers la sortie au rythme d'horloge. Ces registres de transfert de charge sont des CCD en technologie MOS.

##### • Analyse triphasée (fig. 7.29)

Les charges positives accumulées sont piégées par les électrodes 1.4.7 polarisées négativement ( $-10\text{ V}$ ) en commande triphasée. Au signal d'horloge suivant les électrodes 2.5.8 sont polarisées à  $-15\text{ V}$ . Cette tension plus négative attire les charges situées sous les électrodes 1.4.7. Au signal d'horloge suivant la tension des électrodes 2.5.8 revient à  $-10\text{ V}$ , alors que les électrodes voisines sont à  $-5\text{ V}$ . Les charges ont été transférées de 1.4.7 à 2.5.8.

Au signal suivant ce sont les électrodes 3.6.9 qui seront portées à  $-15\text{ V}$  attristant les charges. Avec l'analyse triphasée le registre est bidirectionnel et peut donc décaler les charges vers l'une ou l'autre de ses extrémités, par contre, elle exige un nombre important de transistors et trois horloges synchronisées entre elles.

##### • Analyse biphasée (fig. 7.30)

Avec deux horloges le registre CCD se simplifie et il devient unidirectionnel. Chaque horloge commande une électrode sur deux. La charge contenue dans la cellule (n) du registre doit passer dans la cellule (n + 1) dont l'accès doit être possible grâce à son champ électrique. Cependant l'horloge qui commande aussi (n - 1) et de ce fait la charge (n) va se répartir entre (n - 1) et (n + 1) ce qui n'est pas acceptable.

Il a fallu rendre les cellules dissymétriques, soit par la forme des électrodes, soit par dopage dissymétrique du semi-conducteur sous l'électrode. La figure 7.30 montre la solution classique à électrodes dissymétriques et le transfert des charges dans une seule direction. Le registre est ainsi plus simple, il est plus rapide, et il occupe moins de place sur la pastille de silicium.

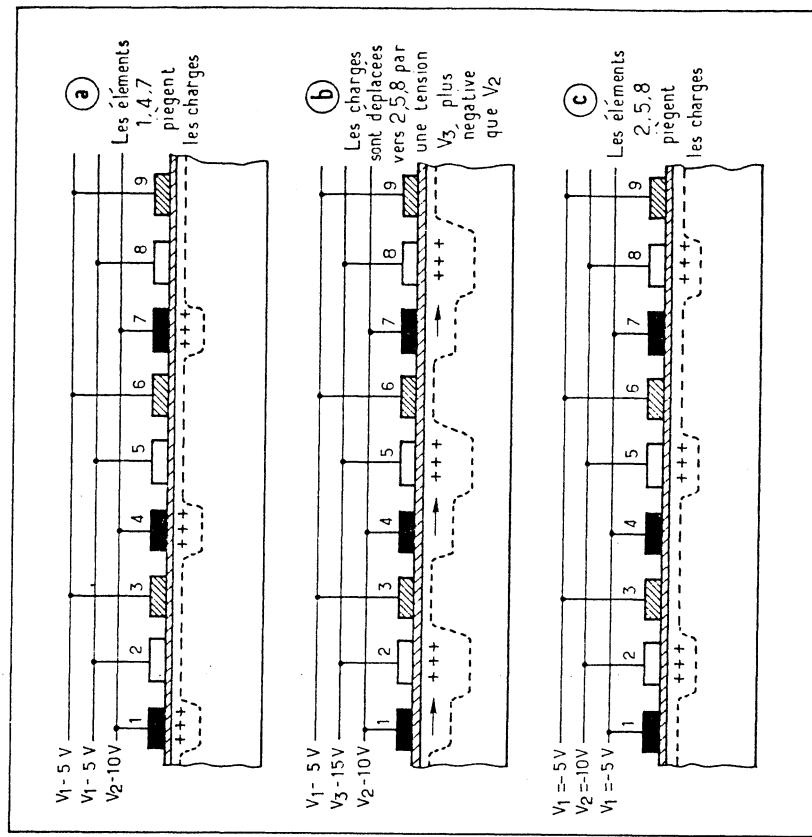


Fig. 7.29. Commande triphasée de circuit à transfert de charge CCD.

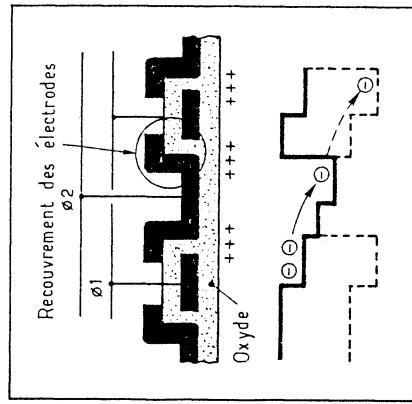


Fig. 7.30. Analyse biphasée avec cellules dissymétriques.

• *Analyse à phase virtuelle* (brevet Texas Instruments) (fig. 7.31)

La technologie MOS canal N à phase virtuelle est encore plus simple, puisqu'elle ne nécessite qu'une horloge et une seule électrode métallique à la surface de l'oxyde. Les électrodes métalliques de commande sont séparées par une jonction connectée au substrat, donc à un potentiel fixe. Cette jonction joue le même rôle qu'une électrode métallique dissymétrique commandée par une horloge.

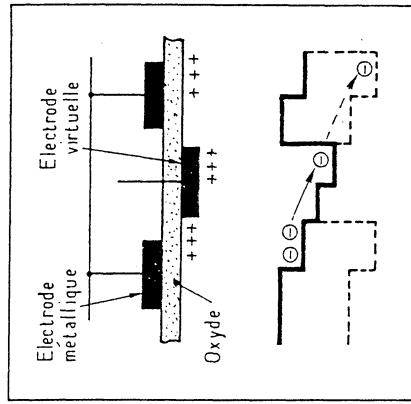


Fig. 7.31. Analyse à phase virtuelle  
(Texas Instruments).

Une telle technologie permet une simplification importante par rapport à la conception traditionnelle des CCD. Les risques de court-circuit entre les électrodes sont supprimés, ce qui augmente la fiabilité.

La vitesse de transfert dépend du nombre de cellules à analyser par ligne ; elle est limitée par des considérations technologiques. On atteint 20 MHz et on envisage 1 GHz.

#### LE CCD 221 FAIRCHILD

Ce détecteur, pris en exemple, est constitué de 488 lignes de 380 points de 185 440 points. La partie photosensible du circuit intégré mesure  $11,4 \times 8,8$  mm.

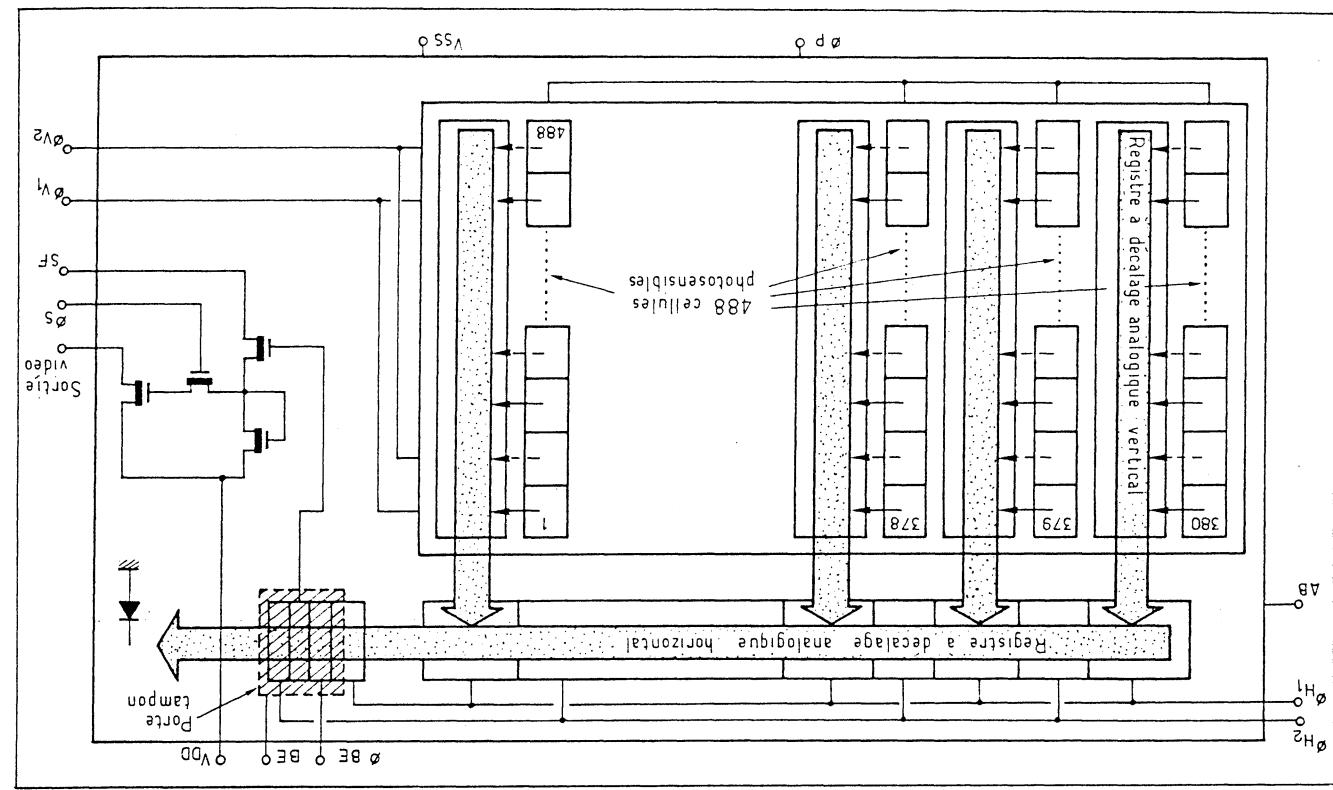
La figure 7.32 représente l'organisation interne. L'écoulement des charges se fait ligne par ligne, dans un premier temps les lignes impaires, dans un deuxième temps les lignes paires. L'image est entrelacée, compatible avec la télévision.

Prochainement des détecteurs de 1 024 lignes et de 1 024 points, soit 1 048 576 points image, sont prévus. C'est seulement à ce moment que des caméras de haute qualité pourront être commercialisées.

#### Pour une image en couleur

La solution la plus performante consisterait à incorporer à l'intérieur du capteur un filtre de sélection des couleurs, mais à l'heure actuelle aucun système de ce genre n'a donné des résultats satisfaisants.

Aussi, la solution retenue consiste à placer un filtre en mosaïque sur la fenêtre du capteur noir et blanc, à environ  $10 \mu\text{m}$ . Le positionnement des couleurs de la mosaïque doit être précis à  $\pm 1 \mu\text{m}$  près.





# Image sensor

From Wikipedia, the free encyclopedia

[Main page](#)  
[Contents](#)  
[Featured content](#)  
[Current events](#)  
[Random article](#)  
[Donate to Wikipedia](#)  
[Wikipedia store](#)

[Interaction](#)  
[Help](#)  
[About Wikipedia](#)  
[Community portal](#)  
[Recent changes](#)  
[Contact page](#)

[Tools](#)  
[What links here](#)  
[Related changes](#)  
[Upload file](#)  
[Special pages](#)  
[Permanent link](#)  
[Page information](#)  
[Wikidata item](#)  
[Cite this page](#)

[Print/export](#)  
[Create a book](#)  
[Download as PDF](#)  
[Printable version](#)

[In other projects](#)  
[Wikimedia Commons](#)



[Languages](#)

[Afrikaans](#)  
[العربية](#)

[Català](#)

[Čeština](#)

[Dansk](#)

[Deutsch](#)

[Eesti](#)

[Español](#)

[فارسی](#)

[Français](#)

[한국어](#)

[ਫਿੰਨੀ](#)

[Македонски](#)

[Nederlands](#)

[日本語](#)

[Norsk bokmål](#)

[Polski](#)

[Português](#)

[Русский](#)

[Slovenčina](#)

[Suomi](#)

[Svenska](#)

[ଓଡ଼ିଆ](#)

['ଓଡ଼ିଆ](#)

[Українська](#)

[Tiếng Việt](#)

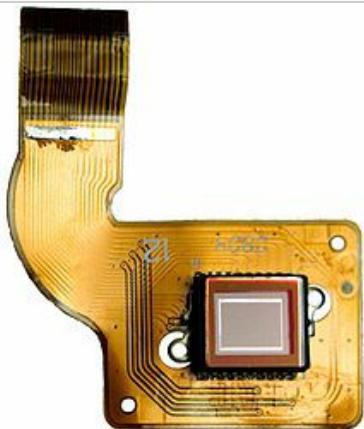
[中文](#)

An **image sensor** or **imaging sensor** is a **sensor** that detects and conveys the **information** that constitutes an **image**. It does so by converting the variable **attenuation of waves** (as they **pass through** or **reflect off** objects) into **signals**, the small bursts of **current** that convey the information. The waves can be **light** or other **electromagnetic radiation**. Image sensors are used in **electronic imaging devices** of both **analog** and **digital** types, which include **digital cameras**, **camera modules**, **medical imaging equipment**, **night vision equipment** such as **thermal imaging devices**, **radar**, **sonar**, and others. As **technology changes**, **digital imaging** tends to replace **analog imaging**.

Early analog sensors for visible light were **video camera tubes**. Currently, used types are **semiconductor charge-coupled devices** (CCD) or **active pixel sensors** in complementary metal–oxide–semiconductor (CMOS) or N-type metal–oxide–semiconductor (NMOS, Live MOS) technologies. Analog sensors for invisible radiation tend to involve **vacuum tubes** of various kinds. Digital sensors include **flat panel detectors**.

## Contents [hide]

- 1 CCD vs CMOS technology
- 2 Performance
- 3 Color separation
- 4 Specialty sensors
- 5 Sensors used in digital cameras
- 6 Companies
- 7 See also
- 8 References
- 9 External links



ACCD image sensor on a [flexible circuit board](#)

## CCD vs CMOS technology [edit]

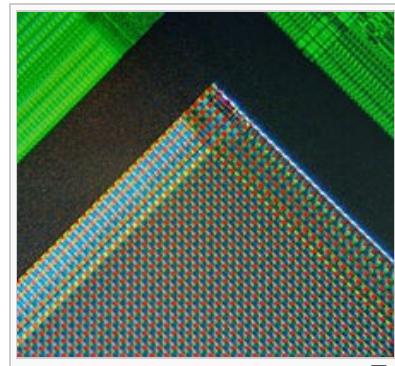
Today, most digital still cameras use a CMOS sensor because CMOS sensor technology in recent years has leapfrogged CCDs. CCD is still in use for cheap low entry cameras, but weak in burst mode.<sup>[1]</sup> Both types of sensor accomplish the same task of capturing light and converting it into electrical signals.

Each cell of a **CCD** image sensor is an analog device. When light strikes the chip it is held as a small electrical charge in each photo sensor. The charges are converted to voltage one pixel at a time as they are read from the chip. Additional circuitry in the camera converts the voltage into digital information.

A CMOS imaging chip is a type of **active pixel sensor** made using the **CMOS** semiconductor process. Extra circuitry next to each photo sensor converts the light energy to a voltage. Additional circuitry on the chip may be included to convert the voltage to digital data.

Neither technology has a clear advantage in image quality.<sup>[citation needed]</sup> On one hand, CCD sensors are more susceptible to vertical smear from bright light sources when the sensor is overloaded; high-end CMOS sensors in turn do not suffer from this problem. On the other hand, cheaper CMOS sensors are susceptible to undesired effects that come as a result of **rolling shutter**.<sup>[citation needed]</sup>

CMOS sensors can potentially be implemented with fewer components, use



A micrograph of the corner of the [photosensor array](#) of a 'webcam' digital camera



less power, and/or provide faster readout than CCD sensors.<sup>[2]</sup> CCD is a more [mature technology](#) and is in most respects the equal of CMOS.<sup>[3][4]</sup> CMOS sensors are less expensive to manufacture than CCD sensors.<sup>[citation needed]</sup>

Image sensor on the motherboard  
of a Nikon Coolpix L2 6 MP

Another hybrid CCD/CMOS architecture, sold under the name "sCMOS," consists of CMOS readout integrated circuits (ROICs) that are bump bonded to a CCD imaging substrate – a technology that was developed for infrared [staring arrays](#) and now adapted to silicon-based detector technology.<sup>[5]</sup> Another approach is to utilize the very fine dimensions available in modern CMOS technology to implement a CCD like structure entirely in CMOS technology. This can be achieved by separating individual poly-silicon gates by a very small gap. These hybrid sensors are still in the research phase and can potentially harness the benefits of both CCD and CMOS imagers.<sup>[6]</sup>

The newer sensor technology is [Back-side illuminated CMOS](#) (BSI-CMOS) which uses less electricity than traditional CMOS with better performance than CCD, so lower-end cameras still use CCD sensors such as those implemented by Fujifilm in its [Bridge cameras](#). CCD sensors are rarely used in new models, except for very high pixel count, big sensor cameras which still use CCDs.

## Performance [edit]

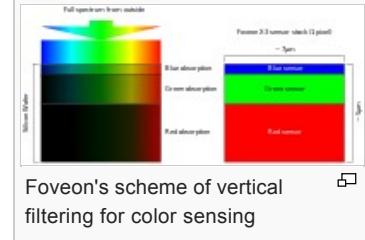
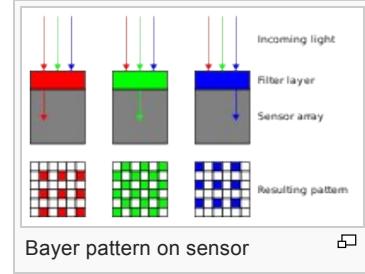
See also: [EMVA1288](#)

There are many parameters that can be used to evaluate the performance of an image sensor, including [dynamic range](#), [signal-to-noise ratio](#), and low-light sensitivity. For sensors of comparable types, the signal-to-noise ratio and dynamic range improve as the [size](#) increases.

## Color separation [edit]

There are several main types of color image sensors, differing by the type of color-separation mechanism:

- **Bayer filter sensor**, low-cost and most common, using a [color filter array](#) that passes red, green, or blue light to selected [pixel sensors](#), forming interlaced grids sensitive to red, green, and blue – the missing color samples are interpolated using a [demosaicing](#) algorithm. In order to avoid interpolated color information, techniques like [color co-site sampling](#) use a [piezo](#) mechanism to shift the color sensor in pixel steps. The Bayer filter sensors also include [back-illuminated sensors](#), where the light enters the sensitive silicon from the opposite side of where the transistors and metal wires are, such that the metal connections on the devices side are not an obstacle for the light, and the efficiency is higher.<sup>[7][8]</sup>
- **Foveon X3 sensor**, using an array of layered [pixel sensors](#), separating light via the inherent wavelength-dependent absorption property of silicon, such that every location senses all three color channels.
- **3CCD**, using three discrete image sensors, with the color separation done by a [dichroic prism](#).



## Specialty sensors [edit]

Special sensors are used in various applications such as [thermography](#), creation of [multi-spectral images](#), video [laryngoscopes](#), [gamma cameras](#), sensor arrays for [x-rays](#), and other highly sensitive arrays for [astronomy](#).<sup>[citation needed]</sup>

While in general digital cameras use a flat sensor, Sony prototyped a curved sensor in 2014 to reduce/eliminate [Petzval field curvature](#) that occurs with a flat sensor. Use of a curved sensor allows a shorter and smaller diameter of the lens with reduced elements and components with greater aperture and reduced light fall-off at the edge of the photo.<sup>[9]</sup>

## Sensors used in digital cameras [edit]

| Width (px) | Height (px) | Aspect ratio | Actual pixel count | Megapixels | Camera examples                                |
|------------|-------------|--------------|--------------------|------------|------------------------------------------------|
| 100        | 100         | 1:1          | 10,000             | 0.01       | Kodak (by Steven Sasson)<br>Prototype 1 (1975) |

# JPEG

JPEG définit aussi une méthode de compression sans perte de qualité, basée sur un codage statistique, mais la base de la norme est une méthode de compression qui permet de contrôler le niveau de perte de qualité de l'image et le taux de compression correspondant. Cette deuxième méthode de compression avec perte utilise une combinaison de différentes techniques, suivant le schéma général suivant :

1. division de l'image en zones de dimensions réduites,
2. réduction du nombre de paramètres significatifs par transformation DCT,
3. quantification des paramètres,
4. codage des paramètres quantifiés.

## **1. Division de l'image en zones de dimensions réduites.**

Ce découpage présente un avantage important : le nombre d'opérations (additions ou multiplications) à effectuer pour effectuer une transformation de type DCT sur une matrice de NXN pixels variant comme  $N^k$  où  $k > 2$ , cette décomposition réduit le nombre total d'opérations nécessaires, sans trop diminuer les performances de la méthode car les corrélations entre pixels éloignés sont peu importantes. JPEG a retenu une matrice de 8 x 8 pixels.

JPEG ne spécifie pas le type de codage des données représentant chaque pixel. Une image couleur est traitée comme trois images monochromes. Les implémentations courantes pour des images couleur utilisent le codage YUV qui correspond déjà à une compression par rapport au codage RVB, avec éventuellement un sous-échantillonnage des informations de chrominance.

## **2. Réduction du nombre de paramètres significatifs par une transformation DCT**

Cette étape en elle-même n'apporte aucune compression. La transformée DCT est une transformée de même nature qu'une transformée de Fourier. Si l'on fait une analogie simple, l'image est représentée par les coefficients de divers harmoniques, et les détails fins sont représentés par les coefficients des harmoniques de rang élevé, dont l'amplitude diminue progressivement en fonction du rang en moyenne. Le nombre de paramètres significatifs est effectivement réduit, car de nombreux coefficients auront une valeur faible; par contre, les coefficients de faible rang pourront avoir une valeur élevée, et le nombre de bits nécessaires pour les représenter sera plus important. Prenons l'exemple d'une image 10 x 10 pixels, en RGB 8 8 8. Il faut 100 valeurs de 24 bits soit 2400 bits pour la représenter. La transformée effectue une condensation de l'information, qui conduit à un nombre de coefficients significatifs inférieur à 100, mais dont certains auront une valeur supérieure à  $2^{24}$ ; il faudra toujours 2400 bits pour représenter l'image sans perte de définition. La figure suivante donne un exemple de matrice de pixels et de sa transformée.

| Pixels |     |     |     |     |     |     |     | Transformée |     |    |     |     |     |    |     |
|--------|-----|-----|-----|-----|-----|-----|-----|-------------|-----|----|-----|-----|-----|----|-----|
| 94     | 74  | 88  | 80  | 107 | 84  | 103 | 122 | 872         | -59 | 9  | -19 | 1   | 0   | -7 | -4  |
| 78     | 96  | 98  | 108 | 93  | 107 | 118 | 117 | -64         | -17 | 2  | 3   | 6   | 3   | 6  | 1   |
| 86     | 106 | 87  | 114 | 105 | 123 | 100 | 114 | -8          | 9   | 1  | 4   | 18  | -10 | -5 | 11  |
| 107    | 86  | 105 | 109 | 103 | 98  | 126 | 117 | 3           | -12 | 10 | -13 | -10 | 0   | 6  | 20  |
| 94     | 109 | 106 | 106 | 99  | 130 | 124 | 136 | -11         | -3  | 19 | 9   | 17  | 4   | 4  | 11  |
| 88     | 113 | 123 | 110 | 105 | 111 | 136 | 128 | -2          | 14  | -3 | 4   | 18  | 4   | 3  | 22  |
| 114    | 119 | 125 | 123 | 128 | 115 | 120 | 125 | -10         | -3  | 1  | 3   | 1   | -9  | -3 | -4  |
| 116    | 121 | 101 | 120 | 120 | 113 | 119 | 128 | 6           | 1   | -4 | 11  | 3   | -20 | 8  | -17 |

## **Calcul de la transformée en cosinus (DCT : Direct Cosine Transform)**

1. Construire la matrice carrée C de dimension NXN (en général, N=8)

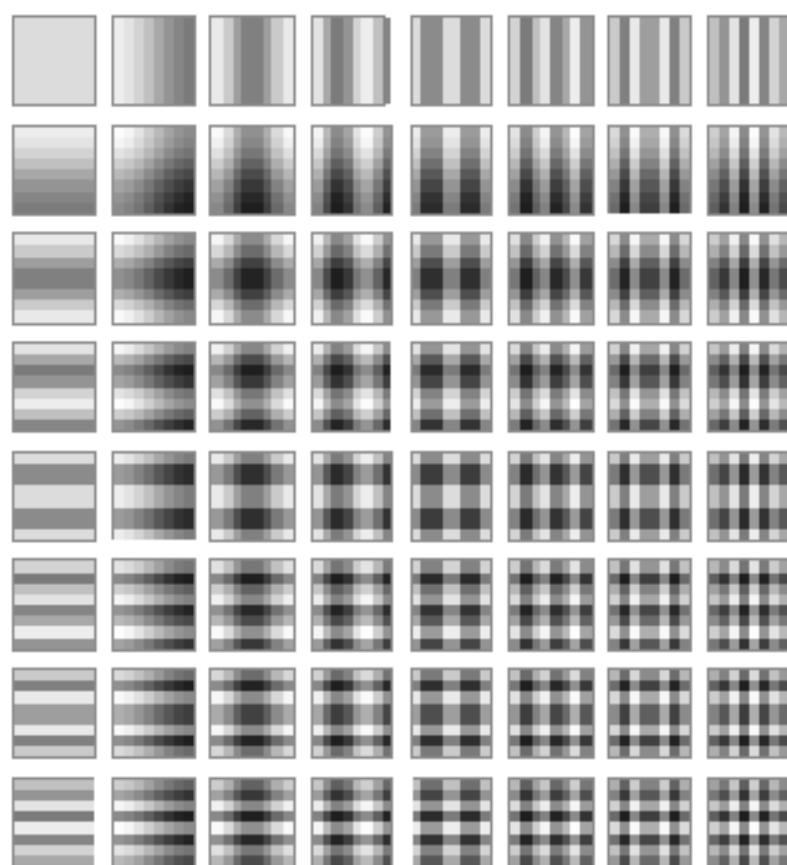
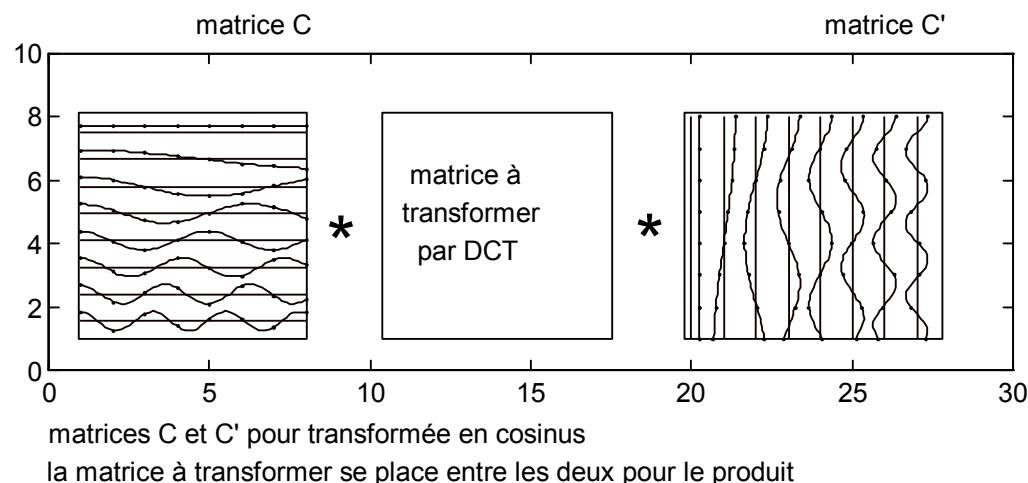
$$n = (0:N-1);$$

$$C = \text{sqrt}(2/N) * \text{Cos}( n' * (2*n+1) * \pi / (2*N) );$$

$$C(1,:) = \text{sqrt}(1/N) * \text{ones}(1,N);$$

2. La transformée en cosinus d'une matrice carrée X (de dimension NXN) est donnée par  $C * X * C'$

C'



### 3. Quantification de ces paramètres

C'est ici qu'intervient la compression. Les coefficients sont quantifiés avec un pas de quantification choisi en fonction de la qualité souhaitée. Un pas de 1 donne une image identique à l'original. Quand on augmente le pas, la précision avec laquelle les coefficients sont restitués diminue, et les coefficients de faible valeur deviennent tous nuls. En jouant sur le pas de quantification, ou la définition des composantes de chrominance YUV, on peut donc obtenir plusieurs taux de compression, avec en contrepartie une dégradation plus ou moins importante de l'image, le taux de compression le plus faible (de l'ordre de 5 à 10) donnant une image pratiquement "sans perte", de qualité équivalente à celle de l'image initiale.

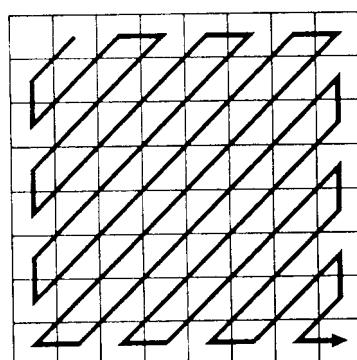
4.

Le pas de quantification des paramètres est fonction de leur position. Dans la matrice  $T$  des coefficients de la transformée, les valeurs élevées de  $i, j$  correspondent aux informations de fréquence élevée et aux détails fins, et se voient affecter des pas de quantification plus élevé, comme indiqué par exemple dans les tables de la figure suivante. La norme JPEG propose des tables de coefficients, mais l'utilisateur peut choisir ces coefficients, et en particulier les modifier en temps réel lors de la compression, ce qui permet de modifier dynamiquement le taux de compression et la qualité correspondante.

| Luminance |    |    |    |     |     |     |     |  | Chrominance |    |    |    |    |    |    |    |    |
|-----------|----|----|----|-----|-----|-----|-----|--|-------------|----|----|----|----|----|----|----|----|
| 16        | 11 | 10 | 16 | 24  | 40  | 51  | 61  |  | 17          | 18 | 24 | 47 | 66 | 99 | 99 | 99 | 99 |
| 12        | 12 | 14 | 19 | 26  | 58  | 60  | 55  |  | 18          | 21 | 26 | 66 | 99 | 99 | 99 | 99 | 99 |
| 14        | 13 | 16 | 24 | 40  | 57  | 69  | 56  |  | 24          | 26 | 56 | 99 | 99 | 99 | 99 | 99 | 99 |
| 14        | 17 | 22 | 29 | 51  | 87  | 80  | 62  |  | 47          | 66 | 99 | 99 | 99 | 99 | 99 | 99 | 99 |
| 18        | 22 | 37 | 56 | 68  | 109 | 103 | 77  |  | 99          | 99 | 99 | 99 | 99 | 99 | 99 | 99 | 99 |
| 24        | 35 | 55 | 64 | 81  | 104 | 113 | 92  |  | 99          | 99 | 99 | 99 | 99 | 99 | 99 | 99 | 99 |
| 49        | 64 | 78 | 87 | 103 | 121 | 120 | 101 |  | 99          | 99 | 99 | 99 | 99 | 99 | 99 | 99 | 99 |
| 72        | 92 | 95 | 98 | 112 | 100 | 103 | 99  |  | 99          | 99 | 99 | 99 | 99 | 99 | 99 | 99 | 99 |

Seuils de quantification proposés par JPEG

En première approche, les valeurs des coefficients et des seuils de quantification correspondants évoluent en fonction de la distance de  $T_{ij}$  à  $T_{00}$ , et s'organisent suivant les parallèles à la seconde diagonale de la matrice. Pour profiter de cette symétrie dans le codage qui va suivre, on range les coefficients en zigzag comme indiqué sur la figure suivante.



## 5. Codage des paramètres

La quantification effectuée précédemment, conjuguée avec le rangement en zigzag, conduit à une suite de paramètres qui comporte des plages de valeur constante, et notamment des plages étendues de valeurs nulles. JPEG utilise un codage hybride de type Run Length pour les valeurs nulles, et de code à longueur variable (voir code Huffmann) pour les valeurs non nulles. Le terme  $T_{00}$  de chaque matrice est traité différemment. Il représente la valeur moyenne de l'information de luminosité ou de couleur dans le carré représenté par la matrice  $I_{xy}$ . Au lieu de transmettre sa valeur absolue, on transmet sa valeur différentielle par rapport à la matrice  $T_{ij}$  précédente.

### Codage statistique

Le modèle utilisé est un modèle statistique. Une source d'information émet des messages dont le modèle permet de déterminer la probabilité d'apparition. Le simple bon sens indique que, pour améliorer l'efficacité de la transmission, il faut coder les messages en mots dont la longueur varie en sens inverse de leur fréquence, et le code Morse utilisait déjà cette technique.

Huffman a introduit le code qui porte son nom, et dont il a démontré que l'efficacité est maximale pour la classe de code qui travaille au niveau des messages élémentaires (ce qui implique que le nombre de bits par message est entier).

### Codage Huffman

Le codage Huffman est donc un procédé de codage dans lequel la longueur des mots codés varie en sens inverse de leur probabilité d'apparition, et qui en outre présente la propriété du préfixe, c'est-à-dire qu'aucun mot du code ne peut être obtenu en ajoutant des lettres (en binaire 0 ou 1) à un autre mot. Cette propriété permet de décoder sans ambiguïté toute séquence, ce qui évite d'avoir à inclure des séparateurs entre les mots

Exemple : Soit à transmettre un jeu de 8 messages  $m_1$  à  $m_8$ , qui apparaissent avec les probabilités  
 $0,3 - 0,2 - 0,15 - 0,10 - 0,10 - 0,05 - 0,05 - 0,05$ .

On peut utiliser le code Huffman suivant :

|       |       |
|-------|-------|
| $m_1$ | 10    |
| $m_2$ | 00    |
| $m_3$ | 110   |
| $m_4$ | 011   |
| $m_5$ | 010   |
| $m_6$ | 1110  |
| $m_7$ | 11111 |
| $m_8$ | 11110 |

La longueur moyenne des mots du code est 2,75 bits, au lieu des 3 bits nécessaires pour coder les 8 messages en mots de longueur fixe. Si le jeu de probabilités des messages avait été  
 $0,5 \ 0,2 \ 0,05 \ 0,05 \ 0,05 \ 0,05 \ 0,05 \ 0,05$ , le codage Huffman optimal aurait été :

|       |       |
|-------|-------|
| $m_1$ | 01    |
| $m_2$ | 10    |
| $m_3$ | 1101  |
| $m_4$ | 1100  |
| $m_5$ | 11111 |
| $m_6$ | 11110 |
| $m_7$ | 11101 |
| $m_8$ | 11100 |

et la longueur moyenne du mot de 2,3bit.

En jouant sur l'ordre de transmission des paramètres, c'est-à-dire en transmettant d'abord les premiers coefficients de toutes les matrices élémentaires, puis les seconds, etc., au lieu de transmettre tous les coefficients d'une matrice avant de transmettre ceux de la suivante, on peut obtenir une construction progressive de la qualité de l'image, ce qui permet d'avoir un premier aperçu, puis de continuer jusqu'à l'obtention de la qualité maximale ou de passer à l'image suivante.



# Compression vidéo

La **compression vidéo** est une méthode de [compression de données](#), qui consiste à réduire la quantité de données, en minimisant l'impact sur la qualité visuelle de la vidéo. L'intérêt de la compression vidéo est de réduire les coûts de stockage et de transmission des fichiers vidéo.

## Sommaire [masquer]

- [1 Historique et standards](#)
- [2 Principes fondamentaux des algorithmes de compression vidéo](#)
- [3 Sous-échantillonnage et interpolation](#)
- [4 Prédition compensée de mouvement](#)
- [5 Conclusion](#)
  - [5.1 Schéma de base du codage entre trames](#)
  - [5.2 Schéma blocs du codage DCT/DPCM](#)
  - [5.3 Schéma blocs du décodage DCT/DPCM](#)
  - [5.4 Fonctionnalités de MPEG-1](#)
- [6 MPEG-2](#)
- [7 Bibliographie](#)
- [8 Articles connexes](#)
- [9 Notes et références](#)
- [10 Liens externes](#)

## Historique et standards [ modifier | modifier le code ]

Les premières tentatives de compression vidéo datent des [années 1950](#)<sup>1</sup>, mais c'est à partir des années 1980 que les premières solutions viables voient le jour<sup>1</sup>. Le premier standard vidéo est [H.120](#), publié en 1984 par le [CCITT](#). Ce premier standard utilise la technique [DPCM](#), et est très limité dans ses fonctionnalités : débit binaire maximal de 2 Mbit/s, niveau de gris uniquement, et qualité peu satisfaisante<sup>1</sup>.

Les organismes de standardisation jouent un rôle très important dans l'avancée et la diffusion des techniques de compression vidéo. Initiée par le [CCITT](#) dans les années 1980, puis par son successeur l'[ITU-T](#)<sup>1</sup>. Ce sont ensuite l'[ITU-R](#), puis l'[ISO](#) et l'[IEC](#) qui coordonnent ces activités de normalisation<sup>1</sup>.

## Principes fondamentaux des algorithmes de compression vidéo [ modifier | modifier le code ]

Les séquences vidéo contiennent une très grande redondance statistique, aussi bien dans le domaine temporel que dans le domaine spatial.

La propriété statistique fondamentale sur laquelle les techniques de compression se fondent, est la corrélation entre [pixels](#). Cette corrélation est à la fois spatiale, les pixels adjacents de l'image courante sont similaires, et temporelle, les pixels des images passées et futures sont aussi très proches du pixel courant.

Ainsi, on suppose que l'importance d'un pixel particulier de l'image peut être prévue à partir des pixels voisins de la même image (utilisant des techniques de codage intra-image) ou des pixels d'une image voisine (utilisant des techniques inter-image). Intuitivement il est clair que dans certaines circonstances, par exemple, lors d'un changement de [plan](#) d'une séquence vidéo, la corrélation temporelle entre pixels entre images voisines est petite, voire nulle. Dans ce cas, ce sont les techniques de codage dites [Intra](#) qui sont les mieux appropriées, afin d'exploiter la corrélation spatiale pour réaliser une compression efficace de données.

Les algorithmes de compression vidéo de type [MPEG](#) utilisent une transformation appelée [DCT](#) (pour *Discrete Cosine Transform*, soit « [transformée en cosinus discrète](#) »), sur des blocs de 8x8 pixels, pour analyser efficacement les corrélations spatiales entre pixels voisins de la même image. D'autres méthodes ont été proposées, en utilisant les [fractales](#), les [ondelettes](#), ou même le [matching pursuit](#).

Cependant, si la corrélation entre pixel dans des trames voisines est grande, c'est-à-dire, dans les cas où deux trames consécutives ont un contenu semblable ou identique, il est souhaitable d'utiliser une technique de codage dite [Inter](#), par exemple la [DPCM](#) ([Differential PCM](#)), qui utilise la prévision temporelle (prévision compensée du mouvement entre trames).

Dans le schéma classique de la compression vidéo, une combinaison adaptative entre les deux mouvements (temporel et spatial) de l'information est utilisée pour réaliser une importante compression de donnée (codage vidéo hybride DPCM/DCT).

## Sous-échantillonnage et interpolation [ modifier | modifier le code ]

La plupart des techniques de compression (improprement appelée codage car il y a perte d'information) qu'on décrira dans cette partie font un [échantillonnage](#) et une [quantification](#) avant de coder l'information. Le concept de base du sous-échantillonnage est de réduire les dimensions (horizontale et verticale) de l'image vidéo et donc de diminuer le nombre de pixels à coder.

Certaines applications vidéo sous-échantillonnent aussi le mouvement temporel pour réduire le débit des images avant de coder. Le récepteur doit donc décoder les images et les interpoler avant de les afficher.

Cette technique de compression peut être considérée comme une des plus élémentaires, qui tient en compte les caractéristiques physiologiques de l'œil et qui enlève la redondance contenue dans les données vidéo.

Les yeux humains sont plus sensibles aux variations de luminosité que de couleur. À cause de ce défaut de l'œil, la majorité des algorithmes de compression vidéo représentent les images dans l'[espace couleur YUV](#), qui comprend une composante de [luminosité](#) et deux de [chrominance](#). Ensuite les composantes chromatiques sont sous-échantillonnées en fonction de la composante de luminance avec un rapport Y : U : V spécifique à une application particulière (par exemple, en [MPEG-2](#) le rapport est de 4:1:1 ou 4:2:2).

## Prédiction compensée de mouvement [ modifier | modifier le code ]

La prédiction compensée de mouvement, ou compensation de mouvement, est un puissant moyen pour réduire les redondances temporelles entre images, et elle est utilisée dans [MPEG-1](#) et [MPEG-2](#) comme prédiction pour le codage [DPCM](#). Le concept de la compensation du mouvement est basé sur l'estimation du mouvement entre images vidéo ; si tous les éléments d'une scène vidéo sont proches dans l'espace, le mouvement entre trames peut être décrit avec un nombre limité de paramètres (vecteurs de mouvement des pixels).

La meilleure prédiction d'un pixel est donnée par la prédiction de mouvement de la trame précédente. Le codage de l'information de mouvement pour chaque pixel de l'image n'est pas nécessaire.

Si la corrélation spatiale entre vecteurs de mouvement est assez haute, un vecteur de mouvement pourra représenter un bloc de pixels adjacents.

Ces blocs sont souvent constitués d'un groupe de 16x16 pixels, et seulement un vecteur de mouvement est estimé, codé et transmis pour chaque bloc.

## Conclusion [ modifier | modifier le code ]

La combinaison des techniques de prédiction compensée de mouvement et de la transformation DCT peuvent être définies comme les éléments clé de la compression vidéo de type [MPEG](#). Un troisième élément caractéristique est que ces techniques sont utilisées pour des petits blocs d'image (souvent 16x16 pixels pour la

compensation de mouvement et 8x8 pixels pour le codage DCT).

Pour ces raisons le codage MPEG fait partie des algorithmes hybrides DPCM/DCT.

### Schéma de base du codage entre trames [\[ modifier \]](#) [\[ modifier le code \]](#)

La technique de compression de base de MPEG-1 (et de MPEG-2) est basée sur une structure de macro-blocs. L'algorithme code la première trame d'une séquence avec un codage Intra-frame (trame I). Chaque trame successive est codée en utilisant la prédiction Inter-frame (trames P) ; seules les données de la trame codée juste précédemment (trames I ou P) seront utilisées pour la prédiction. Chaque couleur d'une trame est partitionnée en macro-blocs.

Chaque macro-bloc contient les données sur la luminosité et la chrominance : 4 blocs de luminosité (Y1, Y2, Y3, Y4) et deux pour la chrominance (U, V), chacun de 8x8 pixels ; ce qui donne un rapport entre luminosité et chrominance de 4:1:1.

### Schéma blocs du codage DCT/DPCM [\[ modifier \]](#) [\[ modifier le code \]](#)

La première trame d'une séquence vidéo (trame I) est codée avec le mode Intra-frames sans aucune référence sur des trames passées ou futures. La DCT est appliquée sur chaque bloc 8x8 de luminosité et de chrominance, chacun des 64 coefficients DCT est quantifié uniformément (Q). Après quantification, le coefficient plus petit (coefficient continu dit DC) est traité différemment par rapport aux autres (coefficients AC). Le coefficient DC correspond à l'intensité moyenne du bloc et il est codé avec une méthode de prédiction différentielle DC. Le reste des coefficients non nuls sont ensuite codés en zigzag comme dans le codage JPEG.

### Schéma blocs du décodage DCT/DPCM [\[ modifier \]](#) [\[ modifier le code \]](#)

Le [décodeur](#) effectue l'opération inverse. Il commence par extraire et décoder les données des différents coefficients DCT pour chaque bloc. Avec la reconstruction (Q\*) des coefficients non nuls il fait la DCT inverse ( $DCT^{-1}$ ) et les valeurs de quantification des pixels des blocs sont reconstituées. Toutes les blocs de chaque image sont décodés et reconstitués.

Pour coder les [trames](#) P, les trames précédentes N-1(trames I ou P) sont mémorisées temporairement dans FS (frame store). La compensation de mouvement (MC) est effectué sur la base des macro-blocs. Un [buffer](#) vidéo (VB) est nécessaire pour assurer un débit constant de flux vidéo.

Conditions de remplissage

Une caractéristique apportée par l'algorithme MPEG-1 est la possibilité de mettre à jour les informations des macro-blocs au décodeur seulement si nécessaire (si le contenu d'un bloc est changé par rapport au contenu du même bloc de l'image précédente). La clé pour un codage efficient des séquences vidéo avec un faible débit de bits par seconde est le bon choix de l'algorithme de prédiction. Le standard MPEG distingue principalement trois méthodes (types MB) :

- MB par saut : Prédiction depuis les trames précédentes avec zéro vecteurs de mouvement. Aucune information sur les macro-blocs n'est codée ni transmise au récepteur.
- Inter MB : Prédiction depuis les trames précédentes. Le type MB, l'adresse MB si demandé, le vecteur de mouvement, coefficients DCT et le pas de quantification sont transmis.
- Intra MB : Aucune prédiction est utilisée. Seulement le type MB, l'adresse MB, le vecteur de mouvement, les DCT et le pas de quantification sont transmis.

### Fonctionnalités de MPEG-1 [\[ modifier \]](#) [\[ modifier le code \]](#)

Pour accéder à un support média, l'[algorithme](#) MPEG-1 fut pensé pour supporter différentes fonctionnalités comme l'accès aléatoire, la recherche rapide en avant (FF-fast forward) et en arrière (FR-fast reverse) dans le flux vidéo, etc.

Pour incorporer ces fonctionnalités et pour tirer plus d'avantage de la compensation de mouvement et de l'interpolation de mouvement, l'algorithme MPEG-1 introduit le concept d'images prédictives et interpolées bidirectionnellement (trames B).

Trois types de trames sont considérées :

- Trames I : Ces trames sont codées sans aucune référence à une autre image de la séquence vidéo, comme expliqué avant. Les trames I permettent de réaliser l'accès aléatoire et les fonctionnalités FF/FR, bien qu'elles ne permettent qu'un très bas taux de compression.
- Trames P : Ces trames sont codées avec une référence à l'image précédente (trame I ou trame P). Ces trames sont utilisées pour la prédiction de trames futures ou passées et elles ne peuvent pas être utilisées pour réaliser l'accès aléatoire et les fonctionnalités FF/FR.
- Trames B : Elles ont besoin des trames futures et passées comme référence pour être codées. Elles sont utilisées pour obtenir un très haut taux de compression. Elles ne sont jamais utilisées comme référence.

L'utilisateur peut arranger la séquence des différents types de trame selon le besoins de l'application. Généralement une séquence vidéo codée en utilisant seulement des trames I (I I I I . . .) donne un haut degré d'accès aléatoire, de FF/FR et d'édition, mais un taux très bas de compression. Une séquence vidéo codée seulement avec des trames P (P P P P P I P P P . . .) permet un degré moyen d'accès aléatoire et de FF/FR.

Si on utilise les trois types de trames (I B B P B B P B I B B P . . .) on arrive à un grand taux de compression et un raisonnable degré d'accès aléatoire et de FF/FR, mais on augmente beaucoup le temps de codage. Pour des applications comme la [vidéotéléphonie](#) ou la [vidéoconférence](#) ce temps peut devenir intolérable.

### MPEG-2 [\[ modifier \]](#) [\[ modifier le code \]](#)

L'algorithme MPEG-2 a été conçu pour avoir une qualité au moins équivalente à celle de NTSC/PAL et supérieure à celle du CCIR 60.

À la base, l'algorithme de MPEG-2 est identique à celui de MPEG-1 et il est donc compatible. Chaque décodeur MPEG-2 doit être capable de décoder un flux vidéo MPEG-1 valide. Plusieurs algorithmes ont été ajoutés pour s'adapter aux nouveaux besoins. MPEG-2 permet de traiter des images entrelacées.

MPEG-2 introduit le concept de « profils » et de « niveaux » pour être compatible avec les systèmes qui n'implémentent pas toutes ces fonctionnalités.

Chaque niveau spécifie la plage des différents paramètres utiles pour le codage.

Le niveau principal supporte au maximum une densité de 720 pixels en horizontal et 576 pixels en vertical, un débit d'images de 30 trames par seconde et un débit de 15 Mbit/s.

### Bibliographie [\[ modifier \]](#) [\[ modifier le code \]](#)

- *Techniques de compression des images*, Jean-Paul Guillois, Éditions Hermès, 1996, ISBN 2-86601-536-3
- *Compression de données, compression des images*, Jean-Paul Guillois, Techniques de l'Ingénieur. Traité d'électronique, E5340. 1998
- *Compression et codage des images et des vidéos* (Traité IC2, série traitement du signal et de l'image), sous la direction de Michel Barlaud et Claude Labit, Hermès-Lavoisier. 2002

### Articles connexes [\[ modifier \]](#) [\[ modifier le code \]](#)

- Voir aussi : [Codec](#), [MPEG-1](#), [MPEG-2](#), [MPEG-4](#), [Moving Picture Experts Group](#), [Transcodage](#).

### Notes et références [\[ modifier \]](#) [\[ modifier le code \]](#)

1. ↑ [a](#), [b](#), [c](#), [d](#) et [e](#) Christine Fernandez-Maloigne, Frédérique Robert-Inacio, Ludovic Macaire, *Couleur Numérique : acquisition, perception, codage et rendu*, Lavoisier, coll. « Traité IC2 », 348 p. (ISBN 978-2-7462-2555-8), p. 249

### Liens externes [\[ modifier \]](#) [\[ modifier le code \]](#)

# ffmpeg Documentation

## Table of Contents

- [1 Synopsis](#)
- [2 Description](#)
- [3 Detailed description](#)
  - [3.1 Filtering](#)
    - [3.1.1 Simple filtergraphs](#)
    - [3.1.2 Complex filtergraphs](#)
  - [3.2 Stream copy](#)
- [4 Stream selection](#)
- [5 Options](#)
  - [5.1 Stream specifiers](#)
  - [5.2 Generic options](#)
  - [5.3 AVOptions](#)
  - [5.4 Main options](#)
  - [5.5 Video Options](#)
  - [5.6 Advanced Video options](#)
  - [5.7 Audio Options](#)
  - [5.8 Advanced Audio options](#)
  - [5.9 Subtitle options](#)
  - [5.10 Advanced Subtitle options](#)
  - [5.11 Advanced options](#)
  - [5.12 Preset files](#)
    - [5.12.1 ffpreset files](#)
    - [5.12.2 avpreset files](#)
- [6 Examples](#)
  - [6.1 Video and Audio grabbing](#)
  - [6.2 X11 grabbing](#)
  - [6.3 Video and Audio file format conversion](#)
- [7 See Also](#)
- [8 Authors](#)

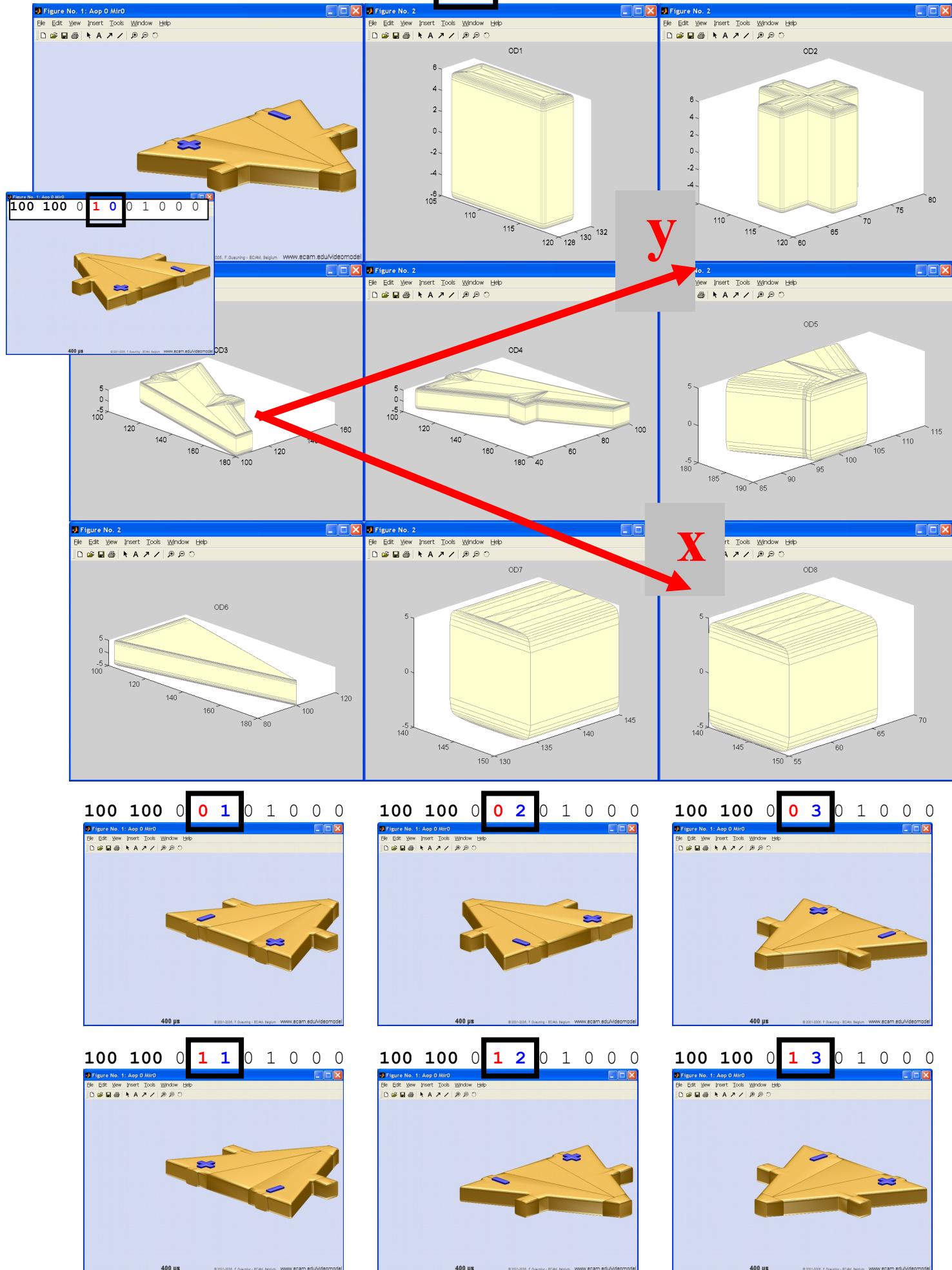
## [1 Synopsis](#)

ffmpeg [*global\_options*] {[*input\_file\_options*] -i *input\_file*} ... {[*output\_file\_options*] *output\_file*} ...

## [2 Description](#)

ffmpeg is a very fast video and audio converter that can also grab from a live audio/video source. It can also convert between arbitrary sample rates and resize video on the fly with a high quality polyphase filter.

Part 100 100 0 0 0 0 0 0 0 0 0 0 'AOP' ''



## Dessin des pièces repensé en fonction d'OpenGL

On définit des pièces de base susceptibles de subir une transformation matricielle 4X4 du type

$$\begin{array}{c} \text{pièce après} \\ \left( \begin{array}{cccccc} X_1 & X_2 & X_3 & \dots & \dots & X_N \\ Y_1 & Y_2 & Y_3 & \dots & \dots & Y_N \\ Z_1 & Z_2 & Z_3 & \dots & \dots & Z_N \\ W_1 & W_2 & W_3 & \dots & \dots & W_N \end{array} \right) \end{array} = \begin{array}{c} \text{transformation} \\ \left( \begin{array}{cccc} A_{XX} & A_{XY} & A_{XZ} & A_{XW} \\ A_{YX} & A_{YY} & A_{YZ} & A_{YW} \\ A_{ZX} & A_{ZY} & A_{ZZ} & A_{ZW} \\ A_{WX} & A_{WY} & A_{WZ} & A_{WW} \end{array} \right) \end{array} \cdot \begin{array}{c} \text{pièce avant} \\ \left( \begin{array}{cccccc} X_1 & X_2 & X_3 & \dots & \dots & X_N \\ Y_1 & Y_2 & Y_3 & \dots & \dots & Y_N \\ Z_1 & Z_2 & Z_3 & \dots & \dots & Z_N \\ W_1 & W_2 & W_3 & \dots & \dots & W_N \end{array} \right) \end{array}$$

La coordonnée W est la valeur par laquelle on divise les trois autres pour obtenir les vraies coordonnées XYZ. Si W=1, on est dans le cas habituel. Si W=0, la pièce est reportée à l'infini (utile en fait pour une lampe, ce qui n'est jamais le cas ici).

### Succession des transformations de la pièce

#### 1. Positionnement correct de la pièce en tant qu'élément du composant de Librairie (L)

- Dilatation<sub>L</sub> (par exemple: étirement d'un conducteur)
- Rotation<sub>L</sub>
- Miroir<sub>L</sub> en X (autour de l'axe Y)
- Translation<sub>L</sub>



#### 2. Positionnement du composant dans le schéma (S)

- Coordonnée verticale en fonction du potentiel (pour neutraliser l'effet de Dilatations<sub>S</sub>, diviser par D<sub>Z</sub>)
- Dilatations<sub>S</sub> (par exemple: étirement d'un conducteur ou dilatation d'un cône de courant)
- Rotation<sub>S</sub>
- Miroir<sub>S</sub> en X (autour de l'axe Y)
- Translation<sub>S</sub>

##### - Coordonnée verticale en fonction du potentiel

On suppose connus les potentiels V<sub>i</sub> en 3 points de référence (non alignés) F, G et H. On a l'incrément Z<sub>i</sub> en ces 3 points par Z<sub>i</sub> = V<sub>i</sub> \* facteurV. Appelons ces incréments respectivement Z<sub>F</sub>, Z<sub>G</sub> et Z<sub>H</sub> et les coordonnées XY correspondantes (X<sub>F</sub>, Y<sub>F</sub>), (X<sub>G</sub>, Y<sub>G</sub>) et (X<sub>H</sub>, Y<sub>H</sub>). Ces 3 points doivent former un plan d'équation Z = A<sub>ZX</sub> \* X + A<sub>ZY</sub> \* Y + A<sub>ZW</sub> donc

$$\begin{pmatrix} Z_F \\ Z_G \\ Z_H \end{pmatrix} = \begin{pmatrix} X_F & Y_F & 1 \\ X_G & Y_G & 1 \\ X_H & Y_H & 1 \end{pmatrix} \cdot \begin{pmatrix} A_{ZX} \\ A_{ZY} \\ A_{ZW} \end{pmatrix} \Rightarrow \boxed{\begin{pmatrix} A_{ZX} \\ A_{ZY} \\ A_{ZW} \end{pmatrix}} = \begin{pmatrix} X_F & Y_F & 1 \\ X_G & Y_G & 1 \\ X_H & Y_H & 1 \end{pmatrix}^{-1} \cdot \begin{pmatrix} Z_F \\ Z_G \\ Z_H \end{pmatrix}$$

On verra plus loin que dans le cas d'effet miroir, il faudra prendre l'opposé de ces incréments avant miroir pour obtenir les bonnes valeurs après miroir (réalisé par rotation de 180°, ce qui exige certaines conditions). Notons M le facteur produisant l'opposé qui vaut 1 (pas miroir) ou -1 (miroir).

On a donc que toute coordonnée Z devient

$$Z + M * A_{ZX} * X + M * A_{ZY} * Y + M * A_{ZW}$$

La transformation de la pièce est donc

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ M \cdot A_{ZX} & M \cdot A_{ZY} & 1 & M \cdot A_{ZW} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

##### - Dilatation (par exemple: étirement d'un conducteur)

On peut faire une dilatation<sup>1</sup> dans les directions X,Y et/ou Z, supposons que le point (0,0,0) soit invariant, pour une dilatation de facteur respectivement D<sub>X</sub>, D<sub>Y</sub>, D<sub>Z</sub>, on a la transformation

<sup>1</sup> Dans le cas de la direction Z, il faut sans doute la faire avant la "transformation en fonction du potentiel"

$$\begin{pmatrix} D_X & 0 & 0 & 0 \\ 0 & D_Y & 0 & 0 \\ 0 & 0 & D_Z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Dans le cas d'un conducteur de longueur 90 plutôt que 10 (suivant l'axe X), ça nous donne

$$\begin{pmatrix} 90/10 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

### - Rotations

$$\begin{pmatrix} \cos \varphi & -\sin \varphi & 0 & 0 \\ \sin \varphi & \cos \varphi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



### - Miroirs en X (autour de l'axe Y)

On suppose la pièce telle qu'on peut remplacer l'effet miroir par une rotation de 180° autour de l'axe Y. L'intérêt est que les normales aux mailles de la surface restent sortantes sans nécessité d'inverser l'ordre des points. Ceci exige qu'à plat, la pièce soit symétrique en Z (par rapport au plan XY) et que l'intervention du potentiel ait été faite en négatif. Alors, on peut faire l'éventuelle rotation de 180° autour de l'axe Y :

$$\begin{pmatrix} M & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & M & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad \text{avec } M = 2 * (0.5 - \text{Mirror}) \quad \text{où Mirror= 0 ou 1}$$

### - Translation

Translation du point 1 de (0, 0) vers (X<sub>0</sub>, Y<sub>0</sub>) :

$$\begin{pmatrix} 1 & 0 & 0 & X_0 \\ 0 & 1 & 0 & Y_0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Au total pour les transformations S (déformation du composant dans le schéma avec potentiel non dilaté)

$$\begin{aligned} & \begin{pmatrix} 1 & 0 & 0 & X_0 \\ 0 & 1 & 0 & Y_0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} M & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & M & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \cos \varphi & -\sin \varphi & 0 & 0 \\ \sin \varphi & \cos \varphi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} D_X & 0 & 0 & 0 \\ 0 & D_Y & 0 & 0 \\ 0 & 0 & D_Z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ M \cdot A_{ZX}/D_Z & M \cdot A_{ZY}/D_Z & 1 & M \cdot A_{ZW}/D_Z \\ 0 & 0 & 0 & 1 \end{pmatrix} \\ & = \begin{pmatrix} M \cdot D_X \cdot \cos \varphi & -M \cdot D_Y \cdot \sin \varphi & 0 & X_0 \\ D_X \cdot \sin \varphi & D_Y \cdot \cos \varphi & 0 & Y_0 \\ M^2 \cdot D_Z \cdot A_{ZX} & M^2 \cdot D_Z \cdot A_{ZY} & M \cdot D_Z & M^2 \cdot D_Z \cdot A_{ZW} \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} M \cdot D_X \cdot \cos \varphi & -M \cdot D_Y \cdot \sin \varphi & 0 & X_0 \\ D_X \cdot \sin \varphi & D_Y \cdot \cos \varphi & 0 & Y_0 \\ A_{ZX} & A_{ZY} & M \cdot D_Z & A_{ZW} \\ 0 & 0 & 0 & 1 \end{pmatrix} \end{aligned}$$

car M<sup>2</sup> = 1

## Dessin de pièce en 3D à l'aide d'Octave

```
% modif 150929
[X, Y, Z] = sphere(50);

% pour octave qui est plus approximatif que matlab 150929
X(find(abs(X)<1e-12))=0;
X(find(abs(X-1)<1e-12))=1;
X(find(abs(X+1)<1e-12))=-1;
Y(find(abs(Y)<1e-12))=0;
Y(find(abs(Y-1)<1e-12))=1;
Y(find(abs(Y+1)<1e-12))=-1;
Z(find(abs(Z)<1e-12))=0;
Z(find(abs(Z-1)<1e-12))=1;
Z(find(abs(Z+1)<1e-12))=-1;

X1 = X + 4 * sign(X);
Y1 = Y + 4 * sign(Y);
Z1 = Z + 4 * sign(Z);

figure
surface(X1, Y1, Z1, 'FaceColor', [0, 0.7, 0], 'EdgeColor', 'none')
set(gca, 'dataaspectratio', [1 1 1])

[r, c] = size(X1);
NewX = reshape(X1', 1, r*c);
NewY = reshape(Y1', 1, r*c);
NewZ = reshape(Z1', 1, r*c);

MatXYZ = [NewX
 NewY
 NewZ
 ones(1, length(NewX))];

M = -1;
phi = 60*pi/180;
%phi = 0;

Dx = 1;
Dy = 10;
Dz = 1;

X0 = 10;
Y0 = 0;

Azx = 0;
Azy = 5;
Azw = 0;

D = [M*Dx*cos(phi) -M*Dy*sin(phi) 0 X0
 M*Dy*cos(phi) M*Dy*sin(phi) 0 Y0
 Azx Azy M*Dz Azw
 0 0 0 1];
T = D * MatXYZ;

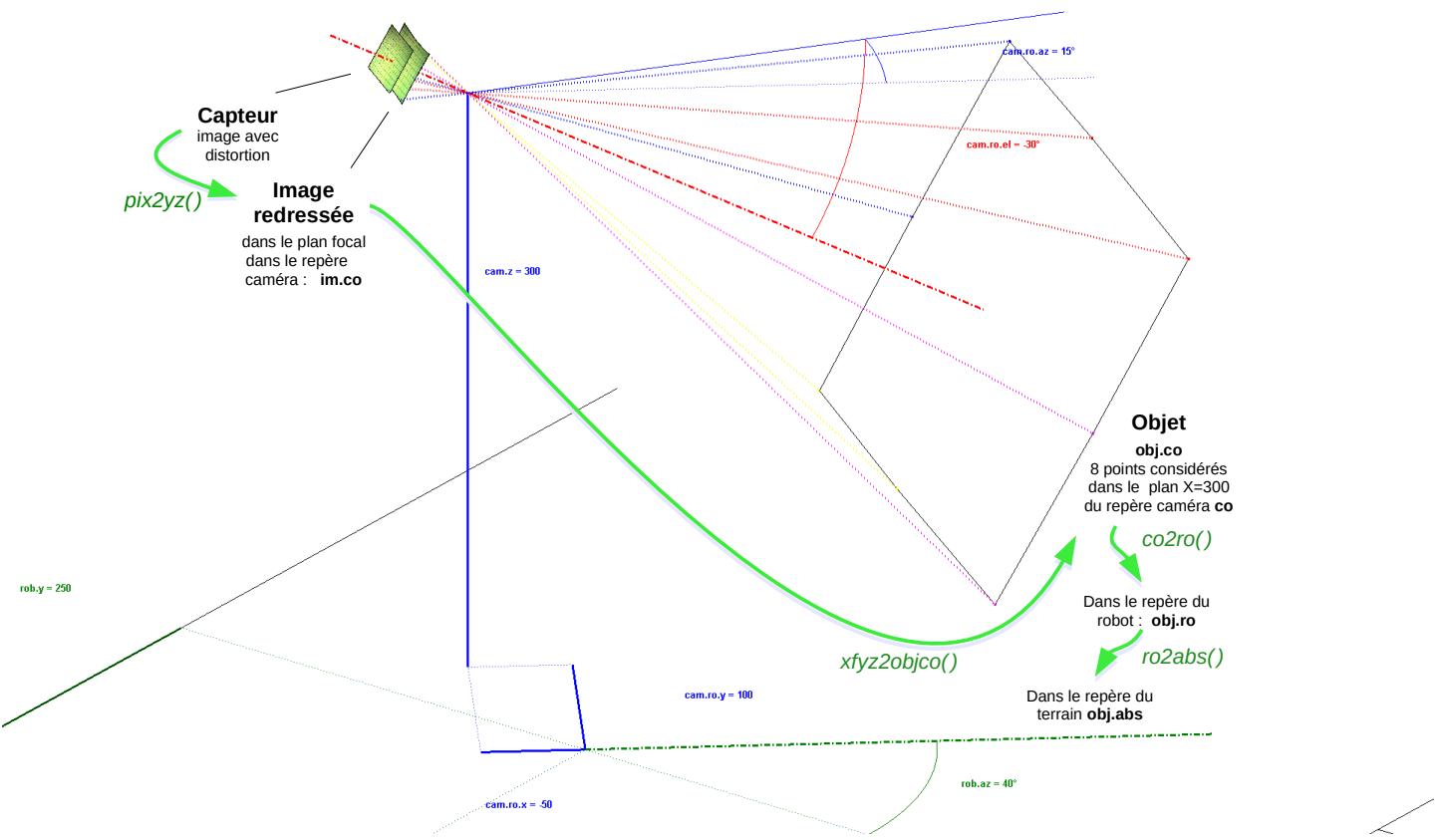
EndX = reshape(T(1, :), r, c);
EndY = reshape(T(2, :), r, c);
EndZ = reshape(T(3, :), r, c);

hold on
%surface(EndX, EndY, EndZ, 'FaceColor', [0, 0, 0.5], 'EdgeColor', 'none')
%light('Position', [1 0 0], 'Style', 'infinite')
surfl(EndX, EndY, EndZ, 'FaceColor', [0, 0, 0.5])
surfl(EndX, EndY, EndZ, 'FaceColor', [0, 0, 0.5], 'EdgeColor', 'none')
set(gca, 'dataaspectratio', [1 1 1])
```

# Position from camera

Comment passer des points d'une image à la position correspondante de l'objet dans les repères de la caméra (co), du robot (ro) et du terrain (abs).

Ceci est illustré par la fonction **positionfromcamera15.m**



---

## Table of Contents

|                                                                                                                                   |    |
|-----------------------------------------------------------------------------------------------------------------------------------|----|
| .....                                                                                                                             | 1  |
| Introduction .....                                                                                                                | 1  |
| Initialisation des coordonnées de la caméra et du robot .....                                                                     | 4  |
| Dessin des axes principaux .....                                                                                                  | 5  |
| Dessin des coordonnées du robot .....                                                                                             | 6  |
| Dessin des coordonnées de la caméra (sur le robot) .....                                                                          | 7  |
| dessin de l'image (donc jpeg retourné de 180°) corrigée en distortion dans le plan focal .....                                    | 8  |
| dessin de l'image (donc jpeg retourné de 180°) sur le capteur dans un plan légèrement arrière au<br>plan focal de la caméra ..... | 9  |
| dessin de l'objet .....                                                                                                           | 10 |
| Conversion de coordonnées du repère caméra only vers celui du robot .....                                                         | 11 |
| Conversion de coordonnées du repère du robot vers celui du terrain .....                                                          | 12 |

```
function positionfromcamera15()
```

## Introduction

```
% Détection de position par caméra embarquée
% F. Gueuning, 2010-2015 Unité Electronique et informatique
ECAM, Bruxelles
%
% SPc 150314: positioncamera3 + en 3d: dessin du capteur, de l'image
corrigée en distortion (en plan focal) et objet
% SPn 130211: création depuis positionfromcamera2, découpage en
plusieurs fonctions
% SPn 130210: corrections orthographiques
% SPn 120323: distinction cr, ch, co plutot que ca + correction
formule MAIS RESTE PROBLEME DE SOMME az ET SOMME el !
% SPn 120301: function avec pix, co, dis
% SPn 120220: Dessin des systèmes d'axes et calcul de distorsion
% SPn 110404: Premières réflexions
%
% Les fonctions suivantes sont successivement appelées :
% imyz = pix2yz(trg, sens): Conversion des pixels en coordonnées de
points images (en mm) dans le plan du capteur
% et dans le repère de la caméra non
redressé (co: camera only)
% objco = xfyz2objco(f, imyz, ref): Conversion des points d'image en
points d'objets dans le repère co (camera only)
% Ceci nécessite des informations supplémentaires à choisir
parmi les deux options suivantes :
% - 4 points d'image dont les distances entre les points
d'objet correspondants sont connues.
% (voir "Finding 3D Positions from 2D Images Feasibility
Analysis, H. G. Lochana Prematunga, ICONS 2012")
% - l'appartenance des points d'objet à un plan connu
% co2ro Conversion du repère co vers coordonnées dans le repère ro
du robot
```

---

```

% ro2abs Conversion du repère ro vers coordonnées dans le repère du
terrain
%
% IN: pix structure des pixels à traiter
% .r indices des lignes (rangées)
% coord structure de coordonnées de robot et caméra
% sens structure de caractéristiques du capteur
%
% On a
% - 4 systèmes de coordonnées : robot (ro), caméra redressée (ca),
image (im), absolu ()
% - une cible (target, tar) ou une balise (beacon, bea) visibles sur
une image
% - les coordonnées x, y, z, r, az, el, ah
% - indices de rangée (ir) et de colonne (ic) d'un point sur l'image
%
% Caméra :
% cam.z Hauteur du centre de l'objectif de la caméra
% cam.ro.x,.y Coordonnées du centre de l'objectif dans le système de
coord du robot
% .az Orientation azimutale (proche de 0 si la caméra regarde
devant, donc suivant l'axe X
% .el Orientation d'élévation (négatif car la caméra regarde
vers le bas)
% .ah Angle de l'horizon avec le bas de l'image, proche de 0°
% Positif si l'horizon apparaît plus haut à droite qu'à
gauche sur l'image
% Lors du calcul des coordonnées de cibles et balises exprimées
dans le repère redressé de la
% caméra (cr), on a neutralisé az, el et ah de la caméra. Le
repère (cr) est donc
% simplement une translation du repère (ro) valant
cam.ro.x,.y,.z
%
% Robot :
% rob.x, .y coord absolues du robot
% .az orientation azimutale du robot sur le terrain
%
%
% Cible (target) :
% tar.im.ir indice de rangée du point d'image de la cible
(target)
% .ic colonne
% .y, .z coord y et z du point au niveau de l'image après
correction de distortion et
% redressement compensant cam.ro.ah
% .cr.az azimut de la cible dans le repère redressé de la
caméra (donc orienté comme le robot)
% .el élévation de la cible telle que perçue depuis le
repère redressé de la caméra
% .ro.r distance entre cible et origine du robot
% .az azimut de la cible en coord du robot
% .x coord de la cible dans un repère absolu (repère du
terrain)
% .y
% .z
%
```

---

```

% Balise (beacon) :
% bea... similaire à tar mais pour une balise (beacon)
% on connaît les coordonnées tar.x,.y,.z des balises
%
% - On suppose connus les 6 paramètres de position de la caméra dans
% le système de coord du robot :
% cam.ro.x, .y, .z, .az, .el, .ah
% - A partir des coordonnées d'un point tar.im.ir,.ic de l'image, on
% peut déduire les coord
% correspondantes tar.cr.az,.el de la cible.
% - D'abord calculer tar.im.y et tar.im.z (à exprimer en mm en
% supposant nulles au centre de l'image)
% Pour la cmucam3, si on se réfère aux mesures de distorsion
% réalisées en 2010, extrait de polymais.m :
% % distorsion en barillet dans le cas où on veut simuler le
% comportement de la caméra
% % 521e-6 a été déterminé expérimentalement 100328 avec les
% étudiants de 4MEO:
% % un carreau de 32 pixels au centre devient 27 pixels à
% 150 pixels du centre 100328
% % dérivée au centre: 32/32, à 150 pixels: 27/32 =
% 1-2*a*150 => a = 521e-6
% PImYZ = PIm(i).Y + j*PIm(i).Z;
% PImYZ =
% abs(PImYZ).*(1-521e-6*abs(PImYZ)).*exp(j*angle(PImYZ));
% PIm(i).Y = real(PImYZ)*8.2/9; % le nombre d'unités en
% largeur est à diminuer car plus larges
% Dans notre cas, ce sont les opérations inverses qu'il faut
% faire puisqu'on doit corriger une image
% prise par la caméra
% Exemple :
im0 = imread('distorsion en barillet.jpg');
% Ajout d'un contour noir à l'image (utile uniquement ici pour
son affichage, sinon à éviter)
[r,c,p]=size(im0);
bord=4; % pixels
im1 = uint8(zeros(r+2*bord, c+2*bord, p));
im1(bord+1:end-bord, bord+1:end-bord, :) = im0;
[r,c,p]=size(im1);
XY = 2*ones(r+1,1)*(0:c)*9/8.2 + j*(0:r)'*ones(1,c+1);
M=mean(mean(XY));
k=1; PImYZ{k} = XY;
k=2; PImYZ{k} = M + abs(XY-M).*(1+521e-6*abs(XY-
M)).*exp(j*angle(XY-M));
k=3; PImYZ{k} = M + abs(XY-M).*(1+50e-6*abs(XY-
M).^1.5).*exp(j*angle(XY-M));
k=4; PImYZ{k} = M + abs(XY-M).*(1+700e-6*abs(XY-
M)).*exp(j*angle(XY-M));
Tit = {'original' '521e-6' '50e-6 et
'^1.5' '700e-6'};
Col = [0 0 0; 0 .5 0; 1 0 0; 0 0 1];
for k= [];%[1 4]
figure
plot([-50 450], [-50 350], '.k'), hold on

```

---

---

```

 hs = surf(real(PImYZ{k}), imag(PImYZ{k}), zeros(r+1,c+1),
double(im1)/255, 'edgecolor','none');
 hold off
 title(Tit{k}, 'Color', Col(k,:))
 view(0,90)
 set(gca, 'DataAspectRatio', [1 1 1])
 pause(.5)
 end
% Ne pas oublier de neutraliser l'effet de cam.ro.ah
%
% - Puis tenir compte de cam.F (distance focale de la caméra) pour
calculator tar.cr.az et tar.cr.el
% tar.cr.az = atan(tar.im.y/cam.F)+cam.ro.az; % cam.F négatif
RELATION INCORRECTE, SOMME VALABLE UNIQUEMENT SI AXE OPTIQUE
HORIZONTAL
% tar.cr.el = atan(tar.im.z/(cam.F/cos(tar.cr.az)))+cam.ro.el; %
correction 120323 SOMME INCORRECTE

% - Comment déduire tar.ro.r,.az à partir de tar.cr.az,.el et tar.z ?
% Si cam.ro.x=0 et cam.ro.y=0 alors on a directement
tar.ro.az=tar.cr.az
% Sinon il faut par exemple que tar.cr.el soit non nulle pour
déduire tar.ro.r,.az
% ce qui nécessite que caméra et cible ne soient pas à la même
hauteur :
% tar.cr.r = (cam.z-tar.z)*tan(tar.cr.el)
% Autre possibilité : se baser sur la taille de l'image de la cible
(fonction de son éloignement).
% - Comment déterminer l'orientation rob.az et la position rob.x,.y du
robot sur le terrain ?
% CECI EST EN GESTATION, IL FAUDRAIT VOIR DES IMAGES POUR SE FAIRE
UNE IDEE
% Pour une balise, si on peut déduire bea.ro.r,.az comme pour une
cible, 2 balises suffisent pour
% connaitre la position (ainsi que l'orientation) du robot, sinon il
faut 3 balises.
% On peut aussi se baser sur 2 images à des positions différentes et
se contenter de 2 balises : si on
% sait qu'on a avancé en ligne droite d'une distance D entre les 2
images, avec les 2 azimuts, on a une
% information similaire à bea.ro.r,.az
% Autre possibilité : tenir compte de l'orientation du bord du
terrain sur l'image

% Dessin des systèmes de coordonnées
%-----

```

## Initialisation des coordonnées de la caméra et du robot

```

cam.z = 300; % [mm] hauteur du centre de l'objectif de la caméra
cam.ro.x = -50;

```

---

---

```

cam.ro.y = 100; % [mm] coordonnées du centre de l'objectif dans le
 système de coord du robot
cam.ro.az = 15; % [°] orientation azimutale (proche de 0 si la caméra
 regarde devant, donc suivant l'axe X
cam.ro.el = -30; % [°] orientation d'élévation (négatif car la caméra
 regarde vers le bas)
cam.ro.ah = 0; % [°] angle de l'horizon avec le bas de l'image, proche
 de 0
 % positif si l'horizon apparaît plus haut à droite qu'à
 gauche sur l'image
rob.x = 240;
rob.y = 250; % [mm] coord absolues du robot
rob.az = 40; % [°] orientation azimutale du robot sur le terrain
roxy = rob.x + j*rob.y; % robot

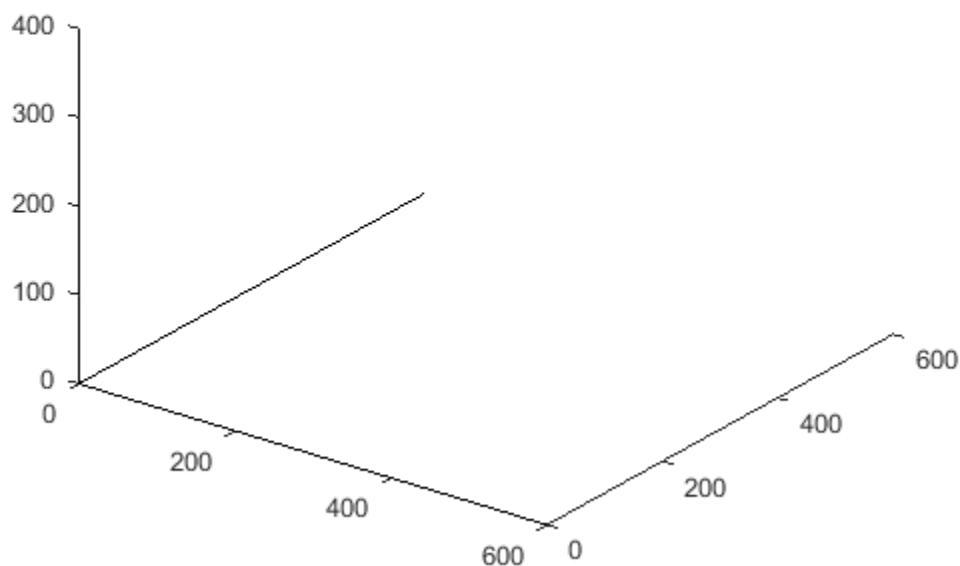
```

## Dessin des axes principaux

```

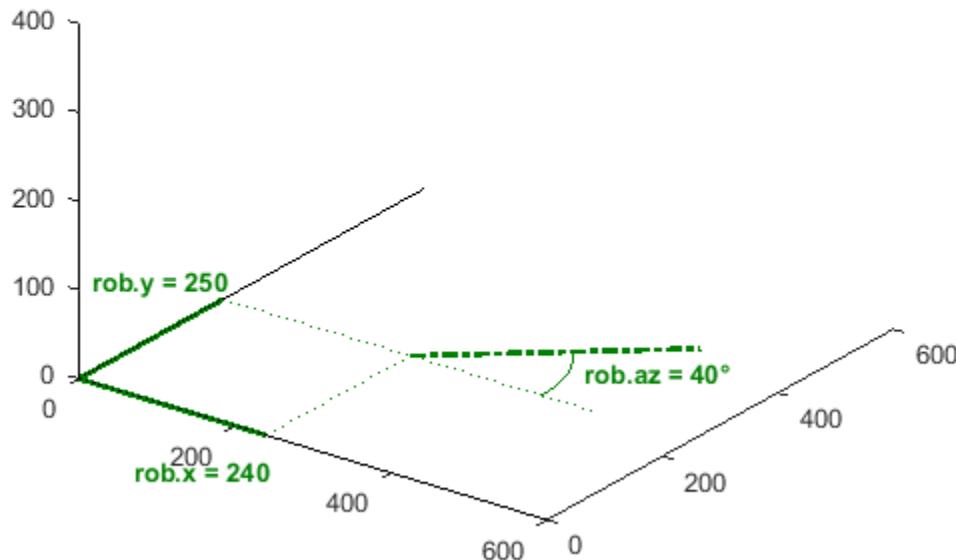
figure
plot3([0 0 0; 600 0 0], [0 0 0; 0 600 0], [0 0 0; 0 0 400], 'k'),
 set(gca, 'DataAspectRatio', [1 1 1])
view(-10.5, 26)
view(36.5, 24)
hold on

```



# Dessin des coordonnées du robot

```
cosraz = cos(rob.az*pi/180);
sinraz = sin(rob.az*pi/180);
plot3([0 0; rob.x 0], [0 0; 0 rob.y], [0 0; 0 0], 'Color', [0 .5
0], 'linewidth', 2) % Lignes sur axes
plot3(rob.x*[0 1; 2 1], rob.y*[1 0; 1 1], [0 0; 0 0], ':', 'Color',
[0 .5 0]) % pointillés ...
plot3(rob.x+[0; 300*cosraz], rob.y+[0; 300*sinraz], [0;
0], '-.', 'Color', [0 .5 0], 'linewidth', 2) % trait d'axe .-..
text(rob.x, -110, 0, ['rob.x = '
num2str(rob.x)],'HorizontalAlignment', 'Center', 'Color', [0 .5
0], 'FontWeight', 'Bold')
text(-80, rob.y, 0, ['rob.y = '
num2str(rob.y)],'HorizontalAlignment', 'Center', 'Color', [0 .5
0], 'FontWeight', 'Bold')
plo = roxy + .7*rob.x*rot((0:abs(rob.az))*sign(rob.az)); % pour dessin
d'angle rob.az
plot3(real(plo), imag(plo), zeros(size(plo)), 'Color', [0 .5 0])
text(1.7*rob.x, 1.3*rob.y, 0, ['rob.az = '
num2str(rob.az) '°'], 'Color', [0 .5 0], 'FontWeight', 'Bold')
```



---

# Dessin des coordonnées de la caméra (sur le robot)

```
camroxy = cam.ro.x + j*cam.ro.y;
camxy = camroxy*rot(rob.az);
plo = roxy + [0 0 camroxy; cam.ro.x j*cam.ro.y camroxy]*rot(rob.az); %
Pour lignes sur axes
plot3(real(plo), imag(plo), [0 0 0; 0 0
cam.z], 'Color', 'B', 'LineWidth', 2)
plo = roxy + [cam.ro.x j*cam.ro.y camroxy; camroxy+[0 0
300]]*rot(rob.az); % Pour pointillés ...
plot3(real(plo), imag(plo), [0 0 cam.z; 0 0 cam.z], 'Color', 'B',
'LineStyle', ':')
plo = roxy + camxy + [0; 300*rot(rob.az+cam.ro.az)]; % Pour trait
d'axe .-.-.
plot3(real(plo), imag(plo), ones(size(plo))*cam.z, 'Color', 'B',
'LineStyle', '-')
plo = roxy -80j; % Pour texte cam.ro.x
text(real(plo), imag(plo), 0, ['cam.ro.x = '
num2str(cam.ro.x)], 'Color', 'B', 'FontWeight', 'Bold')
plo = roxy +80j; % Pour texte cam.ro.y
text(real(plo), imag(plo), 0, ['cam.ro.y = '
num2str(cam.ro.y)], 'Color', 'B', 'FontWeight', 'Bold')
plo = roxy+camxy + 10; % Pour texte cam.z
text(real(plo), imag(plo), .7*cam.z, ['cam.z = '
num2str(cam.z)], 'Color', 'B', 'FontWeight', 'Bold')
plo = roxy+camxy + 200*rot(rob.az
+((0:abs(cam.ro.az))*sign(cam.ro.az))); % pour dessin d'angle
cam.ro.az
plot3(real(plo), imag(plo), ones(size(plo))*cam.z, 'Color', 'B')
plo = roxy+camxy + 260*rot(rob.az+cam.ro.az/2); % pour texte cam.ro.az
text(real(plo), imag(plo), cam.z, ['cam.ro.az = '
num2str(cam.ro.az) '°'], 'Color', 'B', 'FontWeight', 'Bold')
plo = roxy + camxy + [-80; 300]*rot(rob.az
+cam.ro.az)*cosd(cam.ro.el); % Pour trait d'axe optique .-.-.
plot3(real(plo), imag(plo), cam.z+[-80;
300]*sind(cam.ro.el), 'Color', 'R', 'LineStyle', '-.', 'LineWidth',
2)
plo = roxy+camxy + 200*rot(rob.az+cam.ro.az)*cosd(0:abs(cam.ro.el)); %
pour dessin d'angle cam.ro.el
plot3(real(plo), imag(plo), cam.z
+200*sind(0:abs(cam.ro.el))*sign(cam.ro.el), 'Color', 'R')
plo = roxy+camxy + 260*rot(rob.az+cam.ro.az)*cosd(cam.ro.el/2); % pour
texte cam.ro.el
text(real(plo), imag(plo), cam.z+200*sind(cam.ro.el/2)-10, ['cam.ro.el =
' num2str(cam.ro.el) '°'], 'Color', 'R', 'FontWeight', 'Bold')

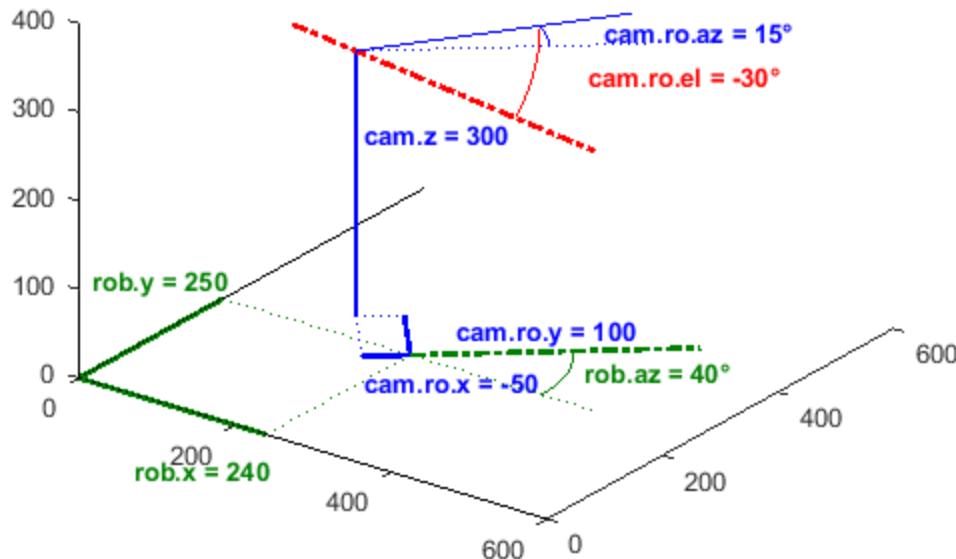
nG = 18; % taille de grille surface: 18x18 Attention! semble produire
des erreurs au-delà de 18 !? (surface ne donne plus des rectangles)
surfgrid.ir = (1:(r-1)/(nG-1):r)/*ones(1,nG); % Grille nGxN de
surface, indices de rangées, valeurs extremes: [1:r]
```

---

```

surfgrid.ic = ones(nG,1) * (1:(c-1)/(nG-1):c); % Grille nGxnG surface,
 indices de colonnes, valeurs extremes: [1 c]
kLook = 10; % multiplicateur pour le dessin du capteur pour qu'il soit
 plus loin du centre optique et plus grand

```

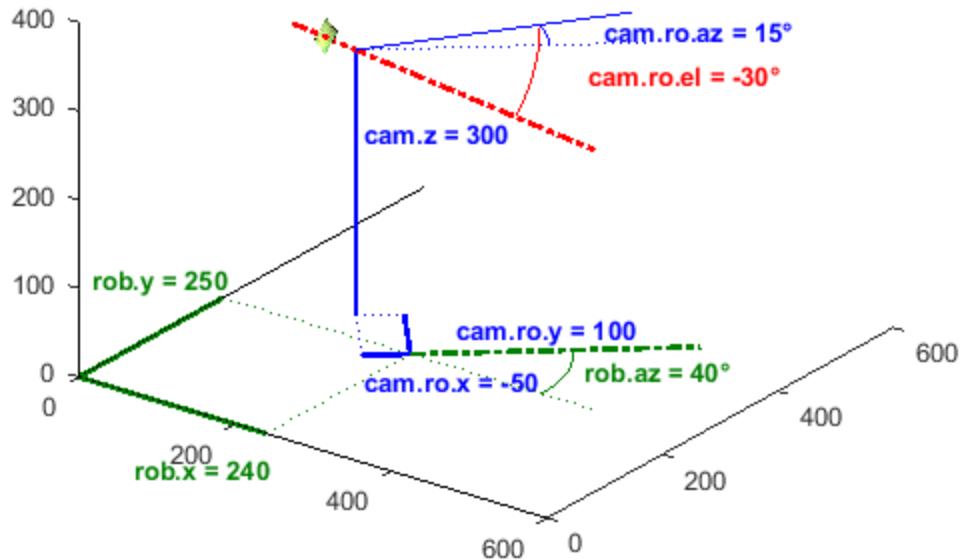


## **dessin de l'image (donc jpeg retourné de 180°) corrigée en distortion dans le plan focal**

```

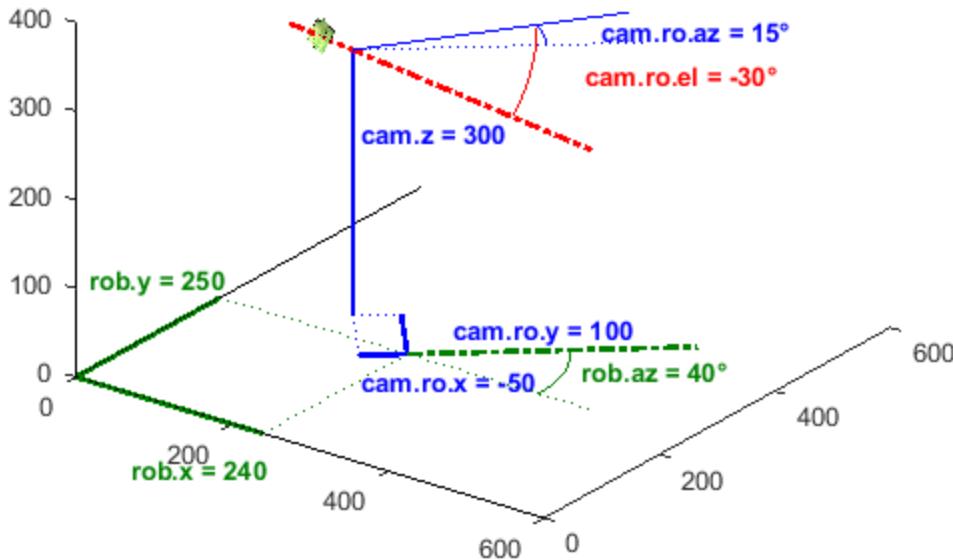
[imnodisto.co, sens] = pix2yz(surfgrid, 'cmucam3_half'); % grille qui
 portera la texture corrigée en distortion (repère caméra only)
imnodisto.co.x = kLook * (-sens.F) * ones(size(imnodisto.co.y));
imnodisto.co.y = -kLook * imnodisto.co.y;
imnodisto.co.z = -kLook * imnodisto.co.z;
imnodisto.ro = co2ro(imnodisto.co, cam); % dans le repère du robot
imnodisto.abs = ro2abs(imnodisto.ro, rob); % dans le repère du terrain
hs = surface(imnodisto.abs.x, imnodisto.abs.y, imnodisto.abs.z ...
 , 'FaceColor', 'texturemap', 'cdata',
 double(im1)/255, 'edgecolor', 'none');

```



**dessin de l'image (donc jpeg retourné de 180°)  
sur le capteur dans un plan légèrement arrière  
au plan focal de la caméra**

```
sensk2null = sens;
sensk2null.k2 = 0; % capteur cmucam3_full sans correction de
distortion
[im.co, sens] = pix2yz(surfgrid, sensk2null); % grille qui portera la
texture d'image brute sur le capteur
im.co.x = kLook * (-sens.F*1.2) * ones(size(im.co.y));
im.co.y = -kLook * im.co.y;
im.co.z = -kLook * im.co.z;
im.ro = co2ro(im.co, cam); % dans le repère du robot
im.abs = ro2abs(im.ro, rob); % dans le repère du terrain
hs = surface(im.abs.x, im.abs.y, im.abs.z ...
, 'FaceColor', 'texturemap', 'cdata',
double(im1)/255, 'edgecolor', 'none');
```



## dessin de l'objet

`objco = xfyz2objco(f, imyz, ref)`: Conversion des points d'image en points d'objets dans le repère co (camera only) Ceci nécessite des informations supplémentaires à choisir parmi les deux options suivantes :  
 - 4 points d'image dont les distances entre les points d'objet correspondants sont connues. (voir "Finding 3D Positions from 2D Images Feasibility Analysis, H. G. Lochana Prematunga, ICONS 2012")  
 - l'appartenance des points d'objet à un plan connu (dans le repère de la caméra) utilisé ici où on dessinera l'objet dans le plan X=300 perpendiculaire à la caméra, ABC=[1/300 0 0]

```
% Objet constitué de 8 points remarquables (contour coplanaire) de
% l'image 'distorsion en barillet.jpg'
imobj.ic = [177.4; 175.0; 89.2; 15.4; 13.5; 17.0; 89.4; 174.3];
imobj.ir = [140.6; 16.0; 6.1; 12.5; 141.9; 268.8; 275.2; 265.6];
imobjyz = pix2yz(imobj, 'cmucam3_half');
imobj.co.x = kLook * (-sens.F) * ones(size(imobjyz.y));
imobj.co.y = -kLook * imobjyz.y;
imobj.co.z = -kLook * imobjyz.z;
imobj.ro = co2ro(imobj.co, cam); % dans le repère du robot
imobj.abs = ro2abs(imobj.ro, rob); % dans le repère du terrain
% Supposer l'objet dans le plan X=300 perpendiculaire à la caméra
ABC = [-1/300 0 0]; % Equation du plan de l'objet: A*X + B*Y + C*Z + 1
= 0
obj.co = xfyz2objco(sens.F, imobjyz, ABC);
obj.ro = co2ro(obj.co, cam); % dans le repère du robot
obj.abs = ro2abs(obj.ro, rob); % dans le repère du terrain
% Dessin de l'objet (8 points en principe d'un rectangle noir)
```

---

```

plot3(obj.abs.x([1:end 1]), obj.abs.y([1:end 1]), obj.abs.z([1:end
1]), 'Color', 'K', 'linewidth', 1)
Mgta = [1 0 1]; Red = [1 0 0]; Blue = [0 0 1]; Yell = [1 1 0];
set(gca, 'ColorOrder', [Mgta; Red; Red; Blue; Blue; Yell; Yell; Mgta])
plot3([obj.abs.x obj.abs.x]', [obj.abs.y obj.abs.y]', [obj.abs.z
obj.abs.z]', '.', 'linewidth', 4)
% Dessin des rayons depuis ces points jusqu'à l'image dans le plan
focal (pointillé noir)
plot3([obj.abs.x imobj.abs.x]', [obj.abs.y imobj.abs.y]', [obj.abs.z
imobj.abs.z]', ':', 'linewidth', 2)

function expangdeg = rot(angdeg)
expangdeg = exp(j*angdeg*pi/180);

```

## Conversion de coordonnées du repère caméra only vers celui du robot

```

function objro = co2ro(objco, cam)

% IN: objco structure de coordonnées de l'objet exprimé dans le
repère de la caméra (caméra only)
% .x,.y,.z [mm] matrices de coordonnées des points de
l'objet
% soit de memes dim, soit l'une ou l'autre
scalaire
% cam
% .z [mm] hauteur du centre de l'objectif de la caméra
% .ro dans le système de coord du robot
% .x,.y [mm] coordonnées du centre de l'objectif
% .az [°] orientation azimutale (proche de 0 si la
caméra regarde devant, donc suivant l'axe X)
% .el [°] orientation d'élévation (négatif car la
caméra regarde vers le bas)
% .ah [°] angle de l'horizon avec le bas de l'image,
proche de 0
% positif si l'horizon apparaît plus haut à
droite qu'à gauche sur l'image
% OUT: objro structure de coordonnées de l'objet exprimé dans le
repère du robot
% .x,.y,.z [mm] matrices de coordonnées des points de
l'objet
% de memes dim

hommat1 = homrot_abc2ay0([0 cosd(cam.ro.ah) sind(cam.ro.ah)]); %
Correction de ah: Rotation autour de l'axe X (A invariant) pour que
A,B,C arrive dans le plan XY (Z=0)
hommat2 = homrot_abc2xb0([cosd(-cam.ro.el) 0 sind(-cam.ro.el)]); %
Correction de el: Rotation autour de l'axe Y (B invariant) pour que
A,B,C arrive dans le plan XY (Z=0)
hommat3 = homrot_abc2x0c([cosd(-cam.ro.az) sind(-cam.ro.az) 0]); %
Correction de az: Rotation autour de l'axe Z (C invariant) pour que
A,B,C arrive dans le plan XZ (Y=0)

```

---

```

hommat4 = homtrans([0;0;0], [cam.ro.x, cam.ro.y, cam.z]);
hommatall = hommat4*hommat3*hommat2*hommat1; % Matrice de
 transformation homogène pour passer du repère de la caméra vers celui
 du robot
[r,c] = size(objco.x);
if length(objco.y)>1
 [r,c] = size(objco.y);
elseif length(objco.z)>1
 [r,c] = size(objco.z);
end
if max(r,c)>1
 if length(objco.x)==1
 objco.x = objco.x * ones(r,c);
 end
 if length(objco.y)==1
 objco.y = objco.y * ones(r,c);
 end
 if length(objco.z)==1
 objco.z = objco.z * ones(r,c);
 end
end
if r>1
 objco.x = reshape(objco.x, 1, r*c);
 objco.y = reshape(objco.y, 1, r*c);
 objco.z = reshape(objco.z, 1, r*c);
end
xyzw = hommatall * [objco.x; objco.y; objco.z; ones(1,r*c)];
objro.x = reshape(xyzw(1,:), r, c);
objro.y = reshape(xyzw(2,:), r, c);
objro.z = reshape(xyzw(3,:), r, c);

```

## Conversion de coordonnées du repère du robot vers celui du terrain

```

function objabs = ro2abs(objro, rob)
% IN: objro structure de coordonnées de l'objet exprimé dans le
 repère du robot
% .x,.y,.z [mm] matrices de coordonnées des points de
 l'objet
% soit de memes dim, soit l'une ou l'autre
% scalaire
% rob
% .x,.y [mm] coordonnées du point de référence du robot
 dans le repère du terrain
% .az [°] orientation azimutale du robot
% OUT: objabs structure de coordonnées de l'objet exprimé dans le
 repère du terrain
% .x,.y,.z [mm] matrices de coordonnées des points de
 l'objet
% de memes dim

```

---

```

hommat1 = homrot_abc2x0c([cosd(-rob.az) sind(-rob.az) 0]); %
Correction de az: Rotation autour de l'axe Z (C invariant) pour que
A,B,C arrive dans le plan XZ (Y=0)
hommat2 = homtrans([0;0;0], [rob.x, rob.y, 0]);
hommatall = hommat2*hommat1; % Matrice de transformation homogène pour
 passer du repère de la caméra vers celui du robot
[r,c] = size(objro.x);
if length(objro.y)>1
 [r,c] = size(objro.y);
elseif length(objro.z)>1
 [r,c] = size(objro.z);
end
if max(r,c)>1
 if length(objro.x)==1
 objro.x = objro.x * ones(r,c);
 end
 if length(objro.y)==1
 objro.y = objro.y * ones(r,c);
 end
 if length(objro.z)==1
 objro.z = objro.z * ones(r,c);
 end
end
if r>1
 objro.x = reshape(objro.x, 1, r*c);
 objro.y = reshape(objro.y, 1, r*c);
 objro.z = reshape(objro.z, 1, r*c);
end
xyzw = hommatall * [objro.x; objro.y; objro.z; ones(1,r*c)];
objabs.x = reshape(xyzw(1,:), r, c);
objabs.y = reshape(xyzw(2,:), r, c);
objabs.z = reshape(xyzw(3,:), r, c);

```

*Published with MATLAB® R2016a*

# Utilisation de la camera en Python sous Windows

## 1. Installation sous Python 2.7

Code tenté en ligne de commande de Python 2.7.11 (32-bit sous Windows 8.1) et résultat

```
import numpy as np
Traceback (most recent call last):
File "<pyshell#0>", line 1, in <module>
 import numpy as np
ImportError: No module named 'numpy'
```

solution: lancer une invite de commande Windows en ayant sélectionné (bouton droit) dans ses propriétés "en tant qu'administrateur".

- Aller dans le dossier où est installé python 2.7 puis
  - `python -m pip install numpy`
  - `python -m pip install matplotlib`

Download **opencv** 3.1 from <https://sourceforge.net/projects/opencvlibrary/files/>

Le dézipper n'importe où et copier son fichier ..\opencv\build\python\2.7\x86\cv2.pyd dans le sous-dossier de python 2.7 **Lib\site-packages**

tester `import cv2` (en principe ok)

## 2. Tutoriel

ref: [http://opencv-python-tutorials.readthedocs.org/en/latest/py\\_tutorials/py\\_gui/py\\_video\\_display/py\\_video\\_display.html](http://opencv-python-tutorials.readthedocs.org/en/latest/py_tutorials/py_gui/py_video_display/py_video_display.html)

```
import numpy as np
import cv2
cap = cv2.VideoCapture(0)
while(True):
 # Capture frame-by-frame
 ret, frame = cap.read()
 # Our operations on the frame come here
 gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
 # Display the resulting frame
 cv2.imshow('frame',gray)
 if cv2.waitKey(1) & 0xFF == ord('q'):
 break
When everything done, release the capture
cap.release()
cv2.destroyAllWindows()
```



latest

Search docs

# Getting Started with Videos

## Goal

- Learn to read video, display video and save video.
- Learn to capture from Camera and display it.
- You will learn these functions : `cv2.VideoCapture()`, `cv2.VideoWriter()`

## Capture Video from Camera

Often, we have to capture live stream with camera. OpenCV provides a very simple interface to this. Let's capture a video from the camera (I am using the in-built webcam of my laptop), convert it into grayscale video and display it. Just a simple task to get started.

To capture a video, you need to create a `VideoCapture` object. Its argument can be either the device index or the name of a video file. Device index is just the number to specify which camera. Normally one camera will be connected (as in my case). So I simply pass 0 (or -1). You can select the second camera by passing 1 and so on. After that, you can capture frame-by-frame. But at the end, don't forget to release the capture.

```
import numpy as np
import cv2

cap = cv2.VideoCapture(0)

while(True):
 # Capture frame-by-frame
 ret, frame = cap.read()

 # Our operations on the frame come here
 gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

 # Display the resulting frame
 cv2.imshow('frame',gray)
 if cv2.waitKey(1) & 0xFF == ord('q'):
 break

 # When everything done, release the capture
 cap.release()
 cv2.destroyAllWindows()
```

`cap.read()` returns a bool (True/False). If frame is read correctly, it will be True. So you can check end of the video by checking this return value.

Sometimes, `cap` may not have initialized the capture. In that case, this code shows error. You can check whether it is initialized or not by the method `cap.isOpened()`. If it is True, OK. Otherwise open it using `cap.open()`.

You can also access some of the features of this video using `cap.get(propId)` method where propId is a number from 0 to 18. Each number denotes a property of the video (if it is applicable to that video) and full details can be seen here: [Property Identifier](#). Some of these values can be modified using `cap.set(propId, value)`. Value is the new value you want.

For example, I can check the frame width and height by `cap.get(3)` and `cap.get(4)`. It gives me 640x480 by default. But I want to modify it to 320x240. Just use `ret = cap.set(3,320)` and `ret = cap.set(4,240)`.

### Note

If you are getting error, make sure camera is working fine using any other camera application (like Cheese in Linux).

It is same as capturing from Camera, just change camera index with video file name. Also while displaying the frame, use appropriate time for `cv2.waitKey()`. If it is too less, video will be very fast and if it is too high, video will be slow (Well, that is how you can display videos in slow motion). 25 milliseconds will be OK in normal cases.

```
import numpy as np
import cv2

cap = cv2.VideoCapture('vtest.avi')

while(cap.isOpened()):
 ret, frame = cap.read()

 gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

 cv2.imshow('frame',gray)
 if cv2.waitKey(1) & 0xFF == ord('q'):
 break

cap.release()
cv2.destroyAllWindows()
```

#### >Note

Make sure proper versions of ffmpeg or gstreamer is installed. Sometimes, it is a headache to work with Video Capture mostly due to wrong installation of ffmpeg/gstreamer.

## Saving a Video

So we capture a video, process it frame-by-frame and we want to save that video. For images, it is very simple, just use `cv2.imwrite()`. Here a little more work is required.

This time we create a **VideoWriter** object. We should specify the output file name (eg: output.avi). Then we should specify the **FourCC** code (details in next paragraph). Then number of frames per second (fps) and frame size should be passed. And last one is **isColor** flag. If it is True, encoder expect color frame, otherwise it works with grayscale frame.

**FourCC** is a 4-byte code used to specify the video codec. The list of available codes can be found in [fourcc.org](http://fourcc.org). It is platform dependent. Following codecs works fine for me.

- In Fedora: DIVX, XVID, MJPG, X264, WMV1, WMV2. (XVID is more preferable. MJPG results in high size video. X264 gives very small size video)
- In Windows: DIVX (More to be tested and added)
- In OSX : (*I don't have access to OSX. Can some one fill this?*)

FourCC code is passed as `cv2.VideoWriter_fourcc('M','J','P','G')` or `cv2.VideoWriter_fourcc(*'MJPG')` for MJPG.

Below code capture from a Camera, flip every frame in vertical direction and saves it.

```
import numpy as np
import cv2

cap = cv2.VideoCapture(0)

Define the codec and create VideoWriter object
fourcc = cv2.VideoWriter_fourcc(*'XVID')
out = cv2.VideoWriter('output.avi',fourcc, 20.0, (640,480))

while(cap.isOpened()):
 ret, frame = cap.read()
 if ret==True:
 frame = cv2.flip(frame,0)

 # write the flipped frame
 out.write(frame)

 cv2.imshow('frame',frame)
 if cv2.waitKey(1) & 0xFF == ord('q'):
 break
 else:
 break
```

```
Release everything if job is finished
cap.release()
out.release()
cv2.destroyAllWindows()
```

## Additional Resources

### Exercises

 Previous

Next 

---

© Copyright 2013, Alexander Mordvintsev & Abid K. Revision 43532856.

Built with [Sphinx](#) using a [theme](#) provided by [Read the Docs](#).

|                                |
|--------------------------------|
| Design III                     |
| Marc-André Gardner             |
| Présentation de OpenCV         |
| Installation                   |
| Paramètres d'une caméra        |
| Acquisition et affichage vidéo |
| Opérations sur les couleurs    |
| Opérations de base             |
| Segmentation                   |
| Transformations géométriques   |
| Reconnaissance de formes       |
| Calibration                    |
| Assembler des fonctions        |
| Démonstration                  |
| Pour en savoir plus            |

# Introduction à OpenCV

## Design III : Intégration

Marc-André Gardner

Département de génie électrique, génie informatique  
Faculté des sciences et de génie  
Université Laval

Hiver 2014

|                                |
|--------------------------------|
| Design III                     |
| Marc-André Gardner             |
| Présentation de OpenCV         |
| Installation                   |
| Paramètres d'une caméra        |
| Acquisition et affichage vidéo |
| Opérations sur les couleurs    |
| Opérations de base             |
| Segmentation                   |
| Transformations géométriques   |
| Reconnaissance de formes       |
| Calibration                    |
| Assembler des fonctions        |
| Démonstration                  |
| Pour en savoir plus            |

## OpenCV, c'est...

- un framework très puissant pour la vision artificielle ;
- une compilation d'algorithmes intéressants pour la vision et faciles à utiliser ;
- des binaires optimisés pour augmenter la performance au maximum ;
- une documentation bien écrite avec de nombreux exemples pour se familiariser avec son fonctionnement.



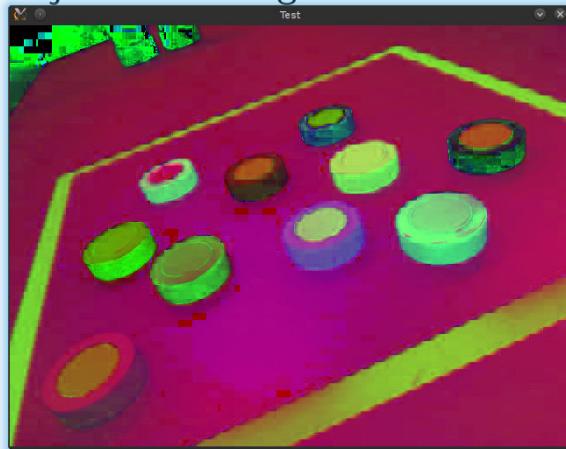
|                                |
|--------------------------------|
| Design III                     |
| Marc-André Gardner             |
| Présentation de OpenCV         |
| Installation                   |
| Paramètres d'une caméra        |
| Acquisition et affichage vidéo |
| Opérations sur les couleurs    |
| Opérations de base             |
| Segmentation                   |
| Transformations géométriques   |
| Reconnaissance de formes       |
| Calibration                    |
| Assembler des fonctions        |
| Démonstration                  |
| Pour en savoir plus            |

## Changement d'espace colorimétrique

- La fonction *cvtColor* permet de passer d'un espace de couleurs à l'autre :

```
import cv2
img_bgr = cv2.imread("mon_image.jpg")
img_hsv = cv2.cvtColor(img_bgr, cv2.cv.CV_BGR2HSV)
img_bgr_again = cv2.cvtColor(img_hsv, cv2.cv.CV_HSV2BGR)
img_gray = cv2.cvtColor(img_bgr, cv2.cv.CV_BGR2GRAY)
```

- OpenCV encode la teinte sur 180 degrés au lieu de 360 !
- Attention à l'affichage : OpenCV suppose que l'image à afficher est toujours une image BGR !



|                                |
|--------------------------------|
| Design III                     |
| Marc-André Gardner             |
| Présentation de OpenCV         |
| Installation                   |
| Paramètres d'une caméra        |
| Acquisition et affichage vidéo |
| Opérations sur les couleurs    |
| Opérations de base             |
| Segmentation                   |
| Transformations géométriques   |
| Reconnaissance de formes       |
| Calibration                    |
| Assembler des fonctions        |
| Démonstration                  |
| Pour en savoir plus            |

## Traitements au niveau des pixels et filtrage

- Les fonctions *dilate* et *erode* sont disponibles.

### Astuce

Il vaut parfois mieux faire plusieurs **itérations** d'érosion ou de dilatation plutôt que d'augmenter la taille du noyau.

- On peut obtenir ouverture et fermeture en combinant *erode* et *dilate* dans l'ordre voulu (par exemple *dilate(erode(img))* pour une ouverture).
- Il est possible de faire un filtrage avec un filtre moyenneur standard (fonction *blur*), un filtre à noyau gaussien (fonction *gaussianBlur*), etc.

# Segmentation

Design III

Marc-André  
Gardner

Présentation  
de OpenCV

Installation

Paramètres  
d'une caméra

Acquisition et  
affichage vidéo

Opérations sur  
les couleurs

Opérations de  
base

Segmentation

Transformations  
géométriques

Reconnaissance  
de formes

Calibration

Assembler des  
fonctions

Démonstration

Pour en savoir  
plus

- La fonction *threshold* est très utile pour discriminer en une dimension.
- Pour choisir un intervalle plutôt qu'un seuil, la fonction *inRange* peut être préférée.
- Ces fonctions sont souvent utilisées pour retourner un **masque** binaire qui peut être appliqué sur l'image, et qui ne conservent que les parties intéressantes : rondelles, tableaux de commande, obstacles, etc.

|                                   |
|-----------------------------------|
| Design III                        |
| Marc-André<br>Gardner             |
| Présentation<br>de OpenCV         |
| Installation                      |
| Paramètres<br>d'une caméra        |
| Acquisition et<br>affichage vidéo |
| Opérations sur<br>les couleurs    |
| Opérations de<br>base             |
| Segmentation                      |
| Transformations<br>géométriques   |
| Reconnaissance<br>de formes       |
| Calibration                       |
| Assembler des<br>fonctions        |
| Démonstration                     |
| Pour en savoir<br>plus            |

# Segmentation par couleur

- Sauf situation particulière, il est de *très loin* préférable de travailler en HSV pour segmenter selon la couleur.
- Deux paramètres principaux pour chaque composant : la valeur recherchée et la tolérance sur celle-ci.
- OpenCV peut trouver le rectangle englobant une zone segmentée (fonction *boundingRect*), le cercle de rayon minimal (*minEnclosingCircle*), ou même un polygone arbitraire (*approxPolyDP*) !



Image originale



Segmentée (bleu)



Segmentée (vert)

|                                |
|--------------------------------|
| Design III                     |
| Marc-André Gardner             |
| Présentation de OpenCV         |
| Installation                   |
| Paramètres d'une caméra        |
| Acquisition et affichage vidéo |
| Opérations sur les couleurs    |
| Opérations de base             |
| Segmentation                   |
| Transformations géométriques   |
| Reconnaissance de formes       |
| Calibration                    |
| Assembler des fonctions        |
| Démonstration                  |
| Pour en savoir plus            |

## Transformations géométriques

- Dans certains cas, on peut vouloir effectuer certaines transformations géométriques (rotation, mise en perspective, etc.) sur l'image.
- OpenCV généralise le concept de **transformation affine**. Les rotations, translations et mises à l'échelle sont des transformations affines (car elles conservent les parallèles).
- On applique une transformation affine en utilisant *warpAffine*. La matrice de transformation peut avoir été obtenue en donnant à *getAffineTransform* les paires départ/arrivée, ou en utilisant des fonctions spécialisées comme *getRotationMatrix2D*.
- Le passage de la vue en perspective à la vue orthogonale n'est **pas** affine. Dans ce cas, on peut alors utiliser *getPerspectiveTransform* et *warpPerspective*.
- On peut aussi redimensionner l'image (par exemple pour économiser les ressources) à l'aide de *resize*.

|                                |
|--------------------------------|
| Design III                     |
| Marc-André Gardner             |
| Présentation de OpenCV         |
| Installation                   |
| Paramètres d'une caméra        |
| Acquisition et affichage vidéo |
| Opérations sur les couleurs    |
| Opérations de base             |
| Segmentation                   |
| Transformations géométriques   |
| Reconnaissance de formes       |
| Calibration                    |
| Assembler des fonctions        |
| Démonstration                  |
| Pour en savoir plus            |

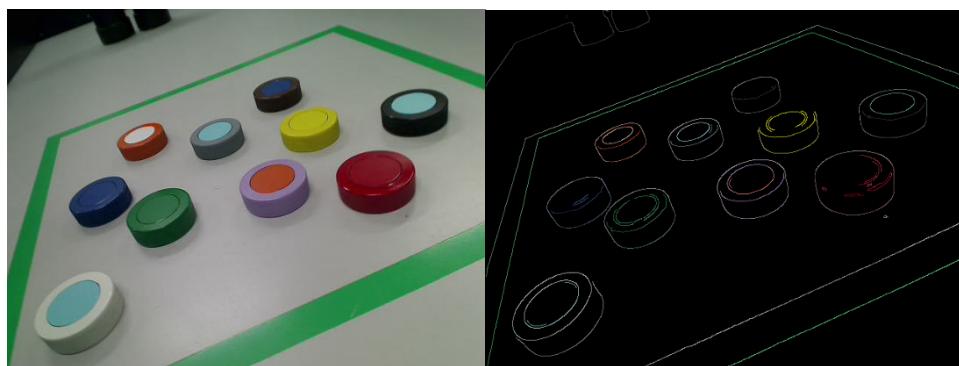
## Reconnaissance de formes I

- OpenCV offre déjà plusieurs fonctions permettant de retrouver des formes de base dans une image.

### Attention

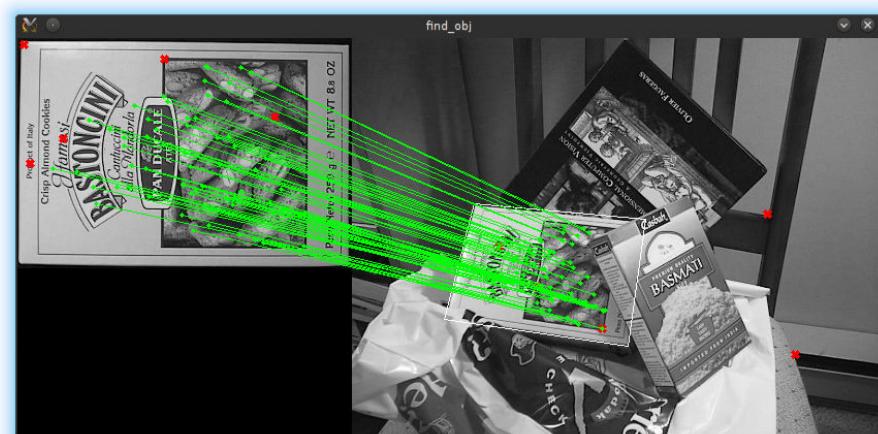
Ces fonctions sont générales et utiles, mais limitées. Elles forment une bonne base, mais en connaissant bien votre problème, vous pouvez faire mieux !

- *Canny* permet de trouver les bords



## Extraction de caractéristiques

- Il existe des techniques bien plus robustes que le *template matching* : SIFT, SURF, ORB, etc.
- Elles sont insensibles à la rotation et à l'échelle, et beaucoup moins sensibles à l'éclairage et aux bruits parasites.
- Elles sont aussi beaucoup plus complexes.
- Voyez l'exemple *find\_obj.py* ou *matcher\_simple.cpp*, fournis avec OpenCV.



## Limites de la calibration

- Une seule caméra (un seul point de vue) ne permet **pas** de voir réellement en 3D
- Par contre, il est possible de calibrer la caméra de façon à obtenir la transformation entre le plan caméra et un autre plan (dans ce cas-ci, la table de jeu)
- Les coordonnées obtenues *ne seront valables que si elles composent des objets sur le plan de la table*

Vous n'avez **pas** besoin de calibration pour :

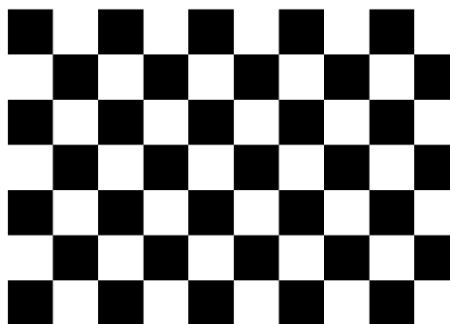
- Segmenter une image
- Identifier des angles dans un environnement connu

## Logiciels ou utilitaires de calibration

Afin d'obtenir les paramètres intrinsèques et extrinsèques de votre caméra, vous devez la calibrer. Plusieurs méthodes peuvent être utilisées :

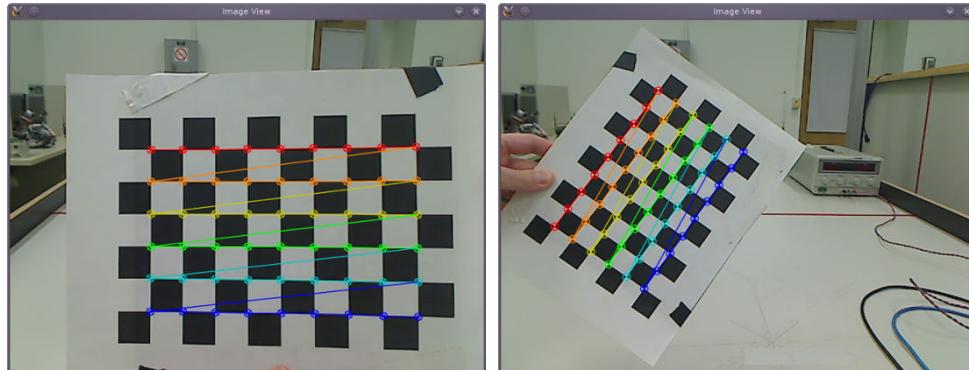
- Utiliser les programmes fournis sur le site du cours
- Utiliser les fonctions de OpenCV (fonctions *calibrateCamera* / *calibrateCamera2*)
- Tout faire à la main (mauvaise idée)

Utilisation d'une cible de calibration :



## Exemple de calibration avec OpenCV I

### 1 Acquisition de multiples vues de la cible de calibration



La cible doit être à **plat** et les orientations et positions doivent être variées

#### Pour une meilleure précision

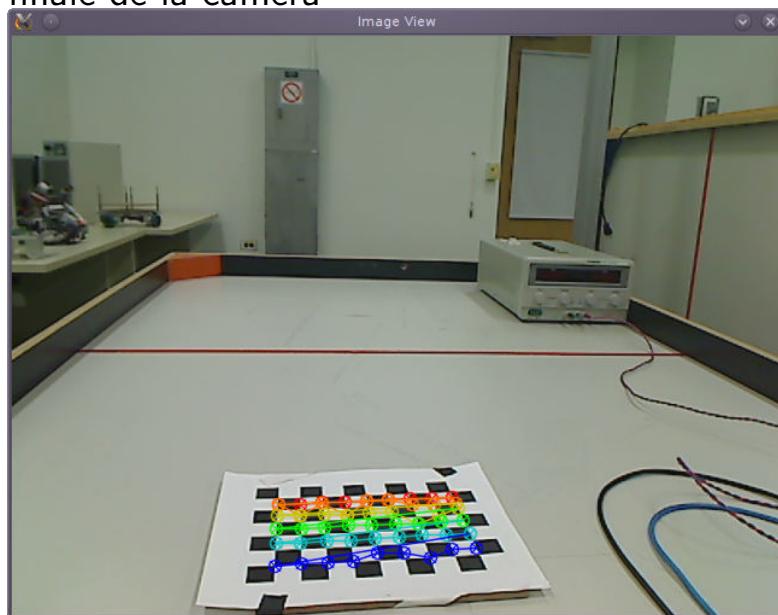
Il est important de s'assurer que la position des points détectés et leur ordre sont corrects (autrement, la calibration donnera de mauvais résultats). Pour un damier, voir les fonctions *findChessboardCorners* et *drawChessboardCorners*.

Le nombre d'images à utiliser n'est pas fixé, mais la précision augmente avec le nombre de points de correspondance (100 points est un minimum).



## Exemple de calibration avec OpenCV II

- ② Acquisition de la vue de la cible à partir de la position finale de la caméra



On remarque que cette image n'est pas une bonne prise de vue (les coins sont mal détectés)



## Exemple de calibration avec OpenCV III

- ③ Calculer les paramètres intrinsèques (indépendants du point de vue) et extrinsèques (une matrice de transformation par image de calibration)

### Exemple de sortie, format OpenCV (matrice caméra)

```
camera_matrix : !opencv-matrix
rows : 3
cols : 3
dt : d
data : [5.4930485270614201e+02, 0., 3.3492436211632804e+02, 0.,
2.1497026854091169e+02, 0., 0., 1.]
```

### Exemple de sortie, format OpenCV (matrice des paramètres extrinsèques)

```
extrinsic_parameters : !opencv-matrix
rows : 10
cols : 6
dt : d
data : [-3.3978911503191261e-01, 5.6482576726218862e-02, 1.1185484133915687e-02,
-1.0072583328494488e+01, -2.9544486841482156e+00, 3.3393291178210653e+01, ...
```

La matrice des paramètres extrinsèques à utiliser est celle correspondant à l'image telle que vue par la caméra lorsque sur le robot.



## Exemple de calibration avec OpenCV IV

### ④ Charger et utiliser les paramètres calculés dans OpenCV. Plusieurs choix sont possibles :

- Intégrer ces paramètres directement au code (chaque changement de paramètre nécessite une recompilation) : peu pratique, mais simple.
- Utiliser la classe *FileStorage* d'OpenCV, qui peut lire des fichiers XML ou YAML spécialement formatés :

```
cv::FileStorage fData("camera.yml", cv::FileStorage::READ);
cv::Mat intrinseques, extrinseques;
fData["camera_matrix"] >> intrinseques;
fData["extrinsic_parameters"] >> extrinseques;
```



## Suppression de la distorsion radiale I

- La distorsion radiale est un artefact dû aux lentilles.
- Fait apparaître courbes les lignes droites (de façon particulièrement marquée sur les bords de l'image).
- La Logitech C905 (votre caméra) ne possède pas une distorsion radiale importante.
- Toutefois, cela peut faire la différence pour la reconnaissance de contours...
- De plus, la plupart des algorithmes de calibration incluent la possibilité de calculer les *coefficients de distorsion*.
- OpenCV fournit des fonctions permettant de corriger les images à partir de ces coefficients (voir *remap*, *initUndistortRectifyMap* et *undistort*).



## Suppression de la distorsion radiale II

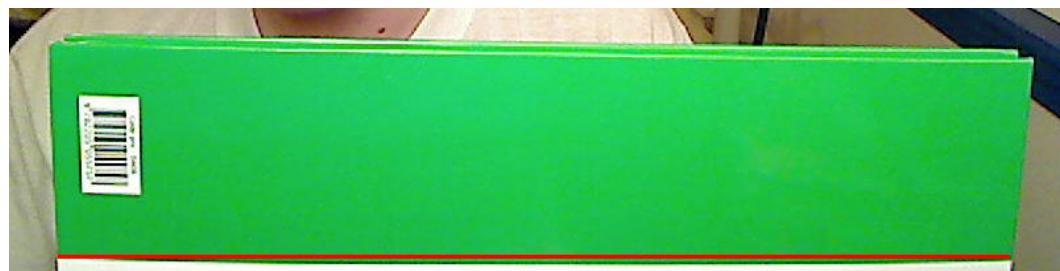


Image avec distorsion radiale (la ligne rouge est un segment de droite qui devrait être confondu avec la séparation blanc/vert)



Image une fois la distorsion radiale supprimée (on remarque les portions noires en bas et sur les côtés de l'image, qui sont inhérentes à la transformation effectuée pour supprimer la distorsion)

## Liste de fonctions disponibles

- Capture directe à partir de la caméra (classe *VideoCapture* en C++, *cvCaptureFromCam* en C)
- Érosion et dilatation (*erode* et *dilate*)
- Ouverture et fermeture (*open* et *close*)
- Seuillage (*threshold* et *adaptiveThreshold*)
- Filtrage (utile par exemple pour rehausser les contours)
- Transformations géométriques (*remap*), permettant par exemple de redresser une image
- Découpage en régions (par exemple *minAreaRect* et *fitEllipse*), pratique pour la segmentation
- **Affichage et acquisition vidéo (module highgui), très utile pour le débogage**

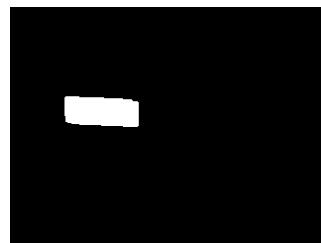
Note importante : pour afficher une image dans une fenêtre OpenCV, la fonction *imshow* doit être utilisée. Cependant, l'affichage n'est pas effectif avant l'appel à *waitKey* avec un délai suffisant !

## Par couleur

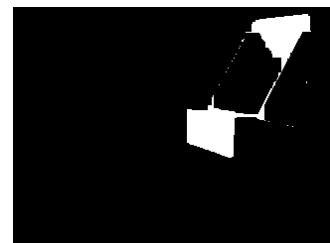
- OpenCV facilite la segmentation d'une image.
- Sauf situation particulière, il est de *très loin* préférable de travailler en HSV pour segmenter selon la couleur.
- Deux paramètres principaux pour chaque composant : la valeur recherchée et la tolérance sur celle-ci.



Image originale



Segmentée (bleu)



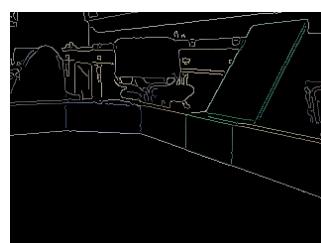
Segmentée (vert)

## Par région

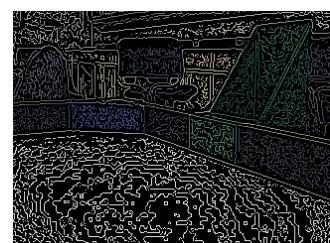
- Plusieurs algorithmes sont implémentés et peuvent être utilisés directement.
- Utilisation de l'algorithme de Canny de recherche de contours (fonctions *canny* et *findContours*)
- Ces contours peuvent par la suite être segmentés (extraction de leur position)
- Attention à l'ajustement du seuil (*threshold*) !



Image originale



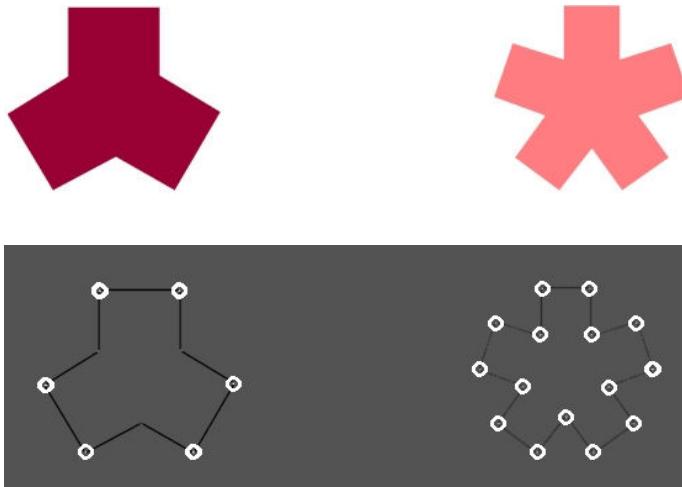
Contours trouvés



Mauvais seuil

## Recherche des coins (Harris)

- OpenCV implémente le détecteur de Harris (*cornerHarris*).
- Ce détecteur peut être pratique pour une première estimation, mais attention aux problèmes de détection !
- Seuil du détecteur difficile à définir de manière absolue...
- La fonction *goodFeaturesToTrack* permet d'obtenir un nombre de points inférieur, mais plus fiable.



Notez que certains coins internes ne sont pas détectés...

## Pour en savoir plus

- Il existe une multitude d'autres fonctions utiles...
- Une excellente référence : *Learning OpenCV : Computer Vision with the OpenCV Library* (disponible en ligne à la bibliothèque de l'Université Laval)
- **Fouillez dans la documentation** (y compris les exemples) !

# GPU Raspberry Pi

## Utilisation du GPU pour analyse d'image contenant des pièces colorées

Le GPU de la Raspberry pi peut atteindre 24 Gflops.  
 Il est ici utilisé pour 2 phases consécutives à la prise d'image :  
     la détection de couleurs appartenant à certaines plage  
     et le calcul de caractéristiques de ces plages.

### Contenu

|                                                         |   |
|---------------------------------------------------------|---|
| 1. Quelques références .....                            | 1 |
| 2. Exemple de prise d'image et utilisation du GPU ..... | 2 |
| 3. Détection de couleurs par fragment shader .....      | 2 |
| 4. Caractéristiques des zones détectées .....           | 4 |

## 1. Quelques références

From <http://jan.newmarch.name/LinuxSound/Diversions/RaspberryPiOpenGL/>

### Using the Raspberry Pi's GPU

The Raspberry Pi (RPi) has a very underpowered CPU but a very powerful GPU. Using the GPU is not well documented. This chapter looks at drawing, rendering images, etc, using the GPU.

[skip table of contents](#)

[Show table of contents](#)

### **Resources**

- [openGL-RPi-tutorial](#)
- [OpenMAX+GLES2 : Decode and render JPEG \(with source code\): jpeg\\_gles2](#)
- [Using OpenGL ES 2.0 on the Raspberry Pi without X windows](#)
- [KHR\\_image\\_base](#) describes EGLImageKHR type
- [OpenGL Programming/Modern OpenGL Tutorial Text Rendering 01](#)
- [Using OpenGL ES to Accelerate Apps with Legacy 2D GUIs](#)
- [glEGLImageTargetTexture2DOES](#)
- [Raspberry Pi VideoCore APIs](#)
- [OpenGL® ES 2.0 Programming Guide](#) by Aaftab Munshi, Dan Ginsburg and Dave Shreiner
- [opengles-book-samples](#)
- [TGA File Format Summary](#)

## 2. Exemple de prise d'image et utilisation du GPU

ref: <http://robotblogging.blogspot.be/2013/10/gpu-accelerated-camera-processing-on.html>

Cet article et le logiciel associé développé en C++ constituent la base de cette étude-ci. Des images sont prises en continu avec la caméra de la Raspberry et traitées de différentes façons à l'aide du GPU. L'affichage à l'écran est de 5 à 10 images par seconde environ. Il est fait appel aux textures et fragment shaders de OpenGL ES 2.0.



## 3. Détection de couleurs par fragment shader

Pour chaque *texel* (texture element), on souhaite détecter s'il est dans une plage de couleurs parmi 4.

Pour chaque plage, on définit sa composante principale (parmi R, G et B) ainsi que le minimum et le maximum exigés pour chacune des composantes. Afin d'être peu sensible à l'éclairement, les composantes secondaires sont testées après avoir été divisées par la composante principale. Le shader ci-dessous a été testé avec succès.

```
// Detect if texel is inside one of up to 4 color ranges
// ECAM - Bruxelles, GEI, F. Gueuning 150210

gngdetectcolfragshader.glsL

varying vec2 tcoord;
uniform sampler2D tex0;
uniform sampler2D tex1;
uniform sampler2D tex2;

uniform vec4 min0; // Min (of first range to test) for color divided by its divby component
uniform vec4 max0; // Max
uniform vec4 col0; // Color to attribute to texel if it is detected inside first range
uniform vec4 min1;
uniform vec4 max1;
uniform vec4 col1;
uniform vec4 min2;
uniform vec4 max2;
uniform vec4 col2;
uniform vec4 min3; // Min (of last possible range to test) for color divided by its divby component
uniform vec4 max3; // Max
uniform vec4 col3; // Color to attribute to texel if it is detected inside last possible range
uniform ivec4 divby; // for each of the up to 4 colors, the divby (0-3) means :
 // 1 divide components by r
 // 2 g
 // 3 b
 // 0 previous was the last color to test
int detectCol(vec4 pix, int Divby, vec4 Min, vec4 Max, vec4 Col) {
 // Detect if color of pix (with components other than Divby divided by component Divby) is between
 Min and Max
```

```

vec4 pixrel;
pixrel = pix;
if (Divby==1) { // Divide by r
 if (pix.r >= Min.r) { // g,b relative to r
 pixrel.g = pix.g / pix.r;
 pixrel.b = pix.b / pix.r;
 }
 else return(0);
}
else if (Divby==2) { // Divide by g
 if (pix.g >= Min.g) { // r,b relative to g
 pixrel.r = pix.r / pix.g;
 pixrel.b = pix.b / pix.g;
 }
 else return(0);
}
else if (Divby==3) { // Divide by b
 if (pix.b >= Min.b) { // r,g relative to b
 pixrel.r = pix.r / pix.b;
 pixrel.g = pix.g / pix.b;
 }
 else return(0);
}
if (all(greaterThanEqual(pixrel, Min))) {
 if (all(lessThanEqual(pixrel, Max))) {
 gl_FragColor = Col; // Color detected
 return(1);
 }
}
return(0);
}

void main(void)
{
 float y = texture2D(tex0,tcoord).r;
 float u = texture2D(tex1,tcoord).r;
 float v = texture2D(tex2,tcoord).r;

 vec4 pix;
 // res.r = (y + (1.370705 * (v-0.5)));
 // res.g = (y - (0.698001 * (v-0.5)) - (0.337633 * (u-0.5)));
 // res.b = (y + (1.732446 * (u-0.5)));
 pix.r = (y + (1.403 * (v-0.5)));
 pix.g = (y - (0.714 * (v-0.5)) - (0.344 * (u-0.5)));
 pix.b = (y + (1.773 * (u-0.5)));
 pix.a = 1.0;
 pix = clamp(pix,vec4(0),vec4(1));

 if (divby.x>0) { // if first color to detect is defined
 if (detectCol(pix, divby.x, min0, max0, col0)==0) { // if first color not detected
 if (divby.y>0) { // if second color is defined
 if (detectCol(pix, divby.y, min1, max1, col1)==0) { // if second not detected
 if (divby.z>0) { // if third is defined
 if (detectCol(pix, divby.z, min2, max2, col2)==0) {
 if (divby.w>0) { // if last is defined
 if (detectCol(pix, divby.w, min3, max3, col3)==0) {
 //gl_FragColor = clamp(pix,vec4(0),vec4(1));
 gl_FragColor = pix;
 }
 }
 }
 }
 }
 }
 }
 }
 else gl_FragColor = pix;
}
}
else gl_FragColor = pix;
}
}

```

## 4. Caractéristiques des zones détectées

Il s'agit de calculer progressivement :

- N : nombre de texels détectés dans une plage de couleur
- $\Sigma X$ : somme des coord X de ces texels
- $\Sigma Y$ : somme des coord Y de ces texels
- $\Sigma D^2 = \Sigma X^2 + \Sigma Y^2$

On calculera également ce que l'on pourrait appeler un "coefficient de compacté"

$$K = 2\pi * (\Sigma D^2 - ((\Sigma X)^2 + (\Sigma Y)^2) / N) / N^2$$

Pour un disque parfait, K vaudrait 1, il semble que si K est supérieur à environ 5, il ne s'agit vraiment pas d'une pièce massive (à tester).

Une première étape consiste à créer des textures d'initialisation, chacune avec son shader. Cette première étape sera réalisée sur 16 texels voisins. Si par exemple la texture d'origine est de dimensions 768x512, chaque texture en cours de calcul est de dimensions 192x128.

- Texture pour N *init\_n\_fragshader.gsls*
  - Init : float n = 0.0;
  - Pour chacun des 16 texels : si couleur détectée, n += 0.04;
  - Convertir n sur rgba
- Texture pour  $\Sigma X$  *nit\_sx\_fragshader.gsls*
  - Init : float sx = 0.0;
  - Pour chacun des 16 texels : si couleur détectée, sx += tcoord.x;
  - Convertir n sur rgba
- Texture pour  $\Sigma Y$  *init\_sy\_fragshader.gsls*
  - Init : float sy = 0.0;
  - Pour chacun des 16 texels : si couleur détectée, sy += tcoord.y;
  - Convertir n sur rgba
- Texture pour  $\Sigma D^2$  *init\_sd2\_fragshader.gsls*
  - Init : float sd2 = 0.0;
  - Pour chacun des 16 texels : si coul. détect., sd2 = tcoord.x \* tcoord.x + tcoord.y \* tcoord.y;
  - Convertir n sur rgba

L'étape suivante consiste à réaliser des textures 2x2 fois plus petites qui à partir des précédentes récupèrent 4 texels voisins, convertissent leur rgba en un float, font la somme et la reconvertisse en rgba. Par exemple, à partir de textures 192x128, il s'agit de construire successivement des textures 96x64, 48x32, 24x16, 12x8, 6x4, 3x2.

Le shader utilisé ici est

Il est donc nécessaire de disposer d'une conversion d'un float 32-bit vers 4 float rgba 8-bit normalisés (et vice versa) en ayant à l'esprit que les bits du float ne sont pas accessibles.

On utilise la routine proposée en

<http://stackoverflow.com/questions/7059962/how-do-i-convert-a-vec4-rgba-value-to-a-float>

Les premiers essais donnent pour chaque rgba 8-bit des textures après traitement :

255 255 255 0

Plutôt que de passer par encode32 et decode32, on va coder directement en rgba 8-bit tcoordx et tcoordy. On suppose que ces coordonnées sont fournies entre 0 et 1. Ex: il y a 2 textels, ils sont

### Exemple détaillé de calcul de $N$ , $\Sigma X$ , $\Sigma Y$ , $\Sigma D^2$ et $K$ par GPU et textures

| <b>N</b> | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|----------|---|---|---|---|---|---|---|---|---|---|----|----|----|
| 0        | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  |
| 1        | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  |
| 2        | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1  | 0  |    |
| 3        | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1  | 1  |    |
| 4        | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1  | 1  | 1  |
| 5        | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1  | 0  |    |
| 6        | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  |

| <b><math>\Sigma X</math></b> | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|------------------------------|---|---|---|---|---|---|---|---|---|---|----|----|----|
| 0                            | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  |
| 1                            | 0 | 0 | 2 | 3 | 4 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  |
| 2                            | 0 | 0 | 2 | 3 | 4 | 0 | 0 | 0 | 0 | 0 | 0  | 11 | 0  |
| 3                            | 0 | 1 | 2 | 3 | 4 | 5 | 0 | 0 | 0 | 0 | 0  | 11 | 12 |
| 4                            | 0 | 0 | 2 | 3 | 4 | 5 | 0 | 0 | 0 | 0 | 10 | 11 | 12 |
| 5                            | 0 | 0 | 0 | 3 | 4 | 0 | 0 | 0 | 0 | 0 | 0  | 11 | 0  |
| 6                            | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  |

| <b><math>\Sigma Y</math></b> | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|------------------------------|---|---|---|---|---|---|---|---|---|---|----|----|----|
| 0                            | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  |
| 1                            | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  |
| 2                            | 0 | 0 | 2 | 2 | 2 | 0 | 0 | 0 | 0 | 0 | 0  | 2  | 0  |
| 3                            | 0 | 3 | 3 | 3 | 3 | 3 | 0 | 0 | 0 | 0 | 0  | 3  | 3  |
| 4                            | 0 | 0 | 4 | 4 | 4 | 4 | 0 | 0 | 0 | 0 | 4  | 4  | 4  |
| 5                            | 0 | 0 | 0 | 5 | 5 | 0 | 0 | 0 | 0 | 0 | 0  | 5  | 0  |
| 6                            | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  |

| <b><math>\Sigma D^2</math></b> | 0 | 1  | 2  | 3  | 4  | 5  | 6 | 7 | 8 | 9 | 10  | 11  | 12  |
|--------------------------------|---|----|----|----|----|----|---|---|---|---|-----|-----|-----|
| 0                              | 0 | 0  | 0  | 0  | 0  | 0  | 0 | 0 | 0 | 0 | 0   | 0   | 0   |
| 1                              | 0 | 0  | 5  | 10 | 17 | 0  | 0 | 0 | 0 | 0 | 0   | 0   | 0   |
| 2                              | 0 | 0  | 8  | 13 | 20 | 0  | 0 | 0 | 0 | 0 | 125 | 0   |     |
| 3                              | 0 | 10 | 13 | 18 | 25 | 34 | 0 | 0 | 0 | 0 | 0   | 130 | 153 |
| 4                              | 0 | 0  | 20 | 25 | 32 | 41 | 0 | 0 | 0 | 0 | 116 | 137 | 160 |
| 5                              | 0 | 0  | 0  | 34 | 41 | 0  | 0 | 0 | 0 | 0 | 0   | 146 | 0   |
| 6                              | 0 | 0  | 0  | 0  | 0  | 0  | 0 | 0 | 0 | 0 | 0   | 0   | 0   |

| <b>K</b> | 0     | 1     | 2     | 3     | 4     | 5     | 6     | 7     | 8     | 9     | 10    | 11    | 12    |
|----------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
| 0        | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| 1        | 0.000 | 0.000 | 0.008 | 0.008 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| 2        | 0.000 | 0.000 | 0.008 | 0.008 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.008 | 0.000 |       |
| 3        | 0.000 | 0.008 | 0.008 | 0.008 | 0.008 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.008 | 0.008 |       |
| 4        | 0.000 | 0.000 | 0.008 | 0.008 | 0.008 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.008 | 0.008 |       |
| 5        | 0.000 | 0.000 | 0.000 | 0.008 | 0.008 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.008 | 0.000 |
| 6        | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |

Pour image 256x256, par couleur :  
 56 textures (max possible = 8 ?)  
 environ 7\*342KB = 2.34MB

Position du centroïde

| <b>N</b> | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|----------|---|---|---|---|---|---|---|
| 0        | 0 | 2 | 1 | 0 | 0 | 0 | 0 |
| 1        | 1 | 4 | 3 | 0 | 0 | 2 | 1 |
| 2        | 0 | 3 | 3 | 0 | 0 | 3 | 1 |
| 3        | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| <b>N</b> | 0 | 1 | 2 | 3 |
|----------|---|---|---|---|
| 0        | 7 | 4 | 2 | 1 |
| 1        | 3 | 3 | 3 | 1 |

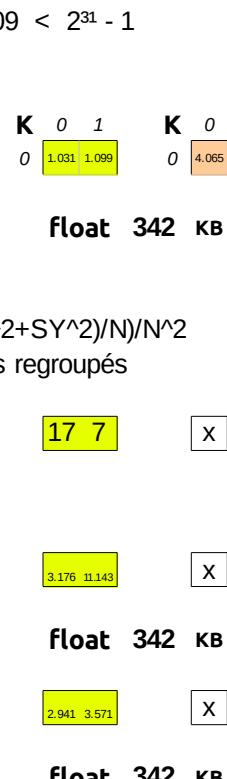
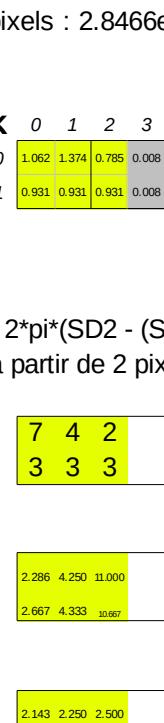
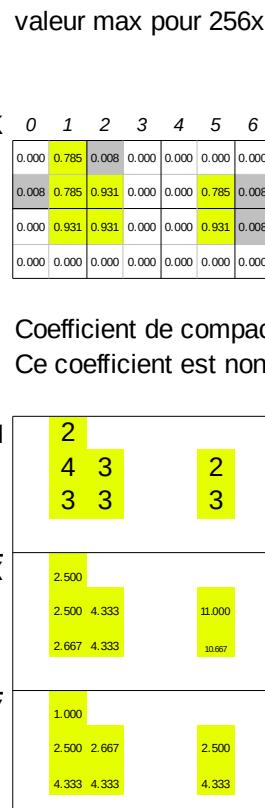
| <b>N</b> | 0  | 1 | 0  |
|----------|----|---|----|
| 0        | 17 | 7 | 24 |

| <b>K</b> | 1.062 | 1.099 | 0.785 | 0.008 |
|----------|-------|-------|-------|-------|
| 1        | 0.931 | 0.931 | 0.931 | 0.008 |
| 2        | 0.931 | 0.931 | 0.931 | 0.008 |
| 3        | 0.000 | 0.000 | 0.000 | 0.000 |

| <b><math>\Sigma X</math></b> | 0  | 1  | 2  | 3  |
|------------------------------|----|----|----|----|
| 0                            | 16 | 17 | 22 | 12 |
| 1                            | 8  | 13 | 32 | 12 |

| <b><math>\Sigma Y</math></b> | 0  | 1  | 2  | 3 |
|------------------------------|----|----|----|---|
| 0                            | 15 | 9  | 5  | 3 |
| 1                            | 13 | 13 | 13 | 4 |

| <b><math>\Sigma D^2</math></b> | 0  | 1   | 2   | 3   |
|--------------------------------|----|-----|-----|-----|
| 0                              | 77 | 96  | 255 | 153 |
| 1                              | 79 | 114 | 399 | 160 |
| 2                              | 0  | 0   | 0   | 0   |
| 3                              | 0  | 0   | 0   | 0   |



respectivement aux coordx 1/4 et 3/4. Il y en a 6, ils sont en 1/12, 3/12, ... 11/12.  
 En supposant des maxima de 768 et 512 pixels, travaillons en  $(768 \times 2 \times 2) = 3072$   
 Les essais indiquent que le codage des coordonnées des vertices se traduit en float à une résolution limitée adaptée à la taille de la texture. Le fragment shader ci-dessous montre que l'on obtient un décalage entre les premières et dernières valeurs. Des variantes ont été tentées pour obtenir des valeurs exactes mais sans succès, il semble que le problème vienne bien de la résolution des float fournis.

```

varying vec2 tcoord;

void main(void)
{
 float N;
 float H;
 vec4 res;
 N = floor(tcoord.x * 3072.0 + .5); // + .5 because floor
 H = floor(N/256.0);
 res.r = H/255.0;
 res.g = (N-256.0*H) / 255.0;
 N = floor(tcoord.y * 3072.0 + .5); // + .5 because floor
 H = floor(N/256.0);
 res.b = H / 255.0;
 res.a = (N-256.0*H) / 255.0;
 gl_FragColor = res;
}

```

Le shader ci-dessus a été utilisé pour plusieurs textures : 0 192x128 à partir de 384x256  
 1 96x64 192x128  
 2 48x32 96x64  
 3 24x16 48x32  
 4 12x8 24x16  
 5 6x4 12x8  
 6 3x2 6x4

On peut déjà voir sur les résultats de la première qu'il y a un décalage :

```
k=0; im_=double(im(k+1),x); kx=3072/(768/2^(k+2)); ky=3072/(512/2^(k+2)); disp(sprintf('texture %d: (%d, %d) > (%d, %d)', k, (im_(:,:,1,1)*256+im_(:,:,1,2))/kx, (im_(:,:,1,1)*256+im_(:,:,1,2))/ky, (im_(:,:,end,end,1)*256+im_(:,:,end,end,2))/kx*0.5, (im_(:,:,end,end,1)*256+im_(:,:,end,end,2))/ky*0.5), if size(im_,2)>16, im_(:,:,1:5 end-4:end],[1:8 end-7:end],); else im_, end
```

texture 0: (4.375000e-01, 4.583333e-01) -> (192, 128)

Pour tcoord.x: texture0(1, :, 2) =

```
7 23 39 55 71 87 103 119 ... 136 152 168 184 200 216 232 248
```

zut! l'écart entre les valeurs aurait dû être chaque fois de 16, les premières valeurs auraient dû être 8, 24, ...

Pour tcoord.y: texture0(:, 1, 4)' =

```
11 35 59 83 107 ... 148 172 196 220 244
```

zut! l'écart entre les valeurs aurait dû être chaque fois de 24, les premières valeurs auraient dû être 12, 36, ...

On va donc utiliser une résolution permettant d'exprimer les positions des texels par pas de 0.5 de la texture de départ. Ce qui nous donne :

|   | Pour structure | à partir de structure (texelsize) | Résolution pour calculs: 2. * texelsize |
|---|----------------|-----------------------------------|-----------------------------------------|
| 0 | 192x128        | 384x256                           | 768x512                                 |
| 1 | 96x64          | 192x128                           | 384x256                                 |
| 2 | 48x32          | 96x64                             | 192x128                                 |
| 3 | 24x16          | 48x32                             | 96x64                                   |

|   |      |       |       |
|---|------|-------|-------|
| 4 | 12x8 | 24x16 | 48x32 |
| 5 | 6x4  | 12x8  | 24x16 |
| 6 | 3x2  | 6x4   | 12x8  |

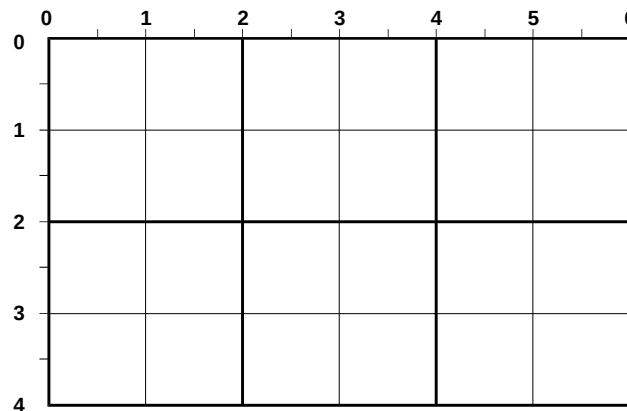
ex: Dans une texture 6x4, on obtient comme positions des texels d'une texture 3x2

Pour tcoord.x: `texture6(1, :, 2) =`

2 6 10 à diviser par 2, soit 1 3 5

Pour tcoord.y: `texture6(:, 1, 4)' =`

2 6 à diviser par 2, soit 1 3



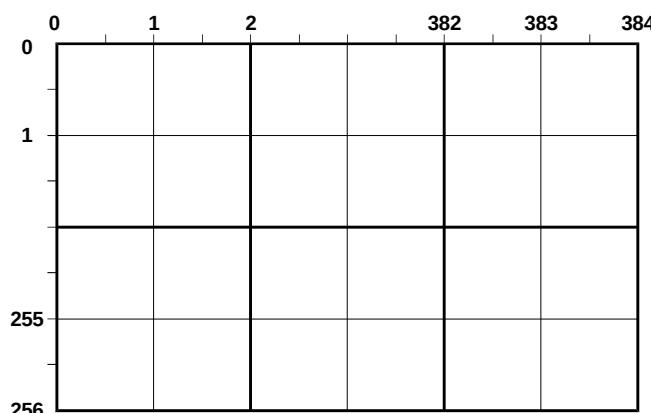
ex: Dans une texture 192x128, on obtient comme positions des texels d'une texture 384x256

Pour tcoord.x: `texture0(1, :, 2) =`

2 6 10 ... 762 766 à diviser par 2, soit 1 3 5 ... 381 383

Pour tcoord.y: `texture0(:, 1, 4)' =`

2 6 10 ... 506 510 à diviser par 2, soit 1 3 5 ... 253 255



## 5. Tests de temps d'exécution

Les tests réalisés sur une raspberry 1 (voir encadré) ont montré que l'initialisation nécessaire chaque fois que l'on fournit une texture pour calcul au GPU (indépendamment de sa taille) nécessite en moyenne 1.8ms.

```

for(int Num = 0; Num < 1; Num++) // For each of the 4 colors to detect
 // framerate for 0 color: 15.2 frames/s, (66ms) for 1 color: 11.2 frames/s (66+ 24ms)
 // for 2 colors: 6.77 frames/s (66+24+ 58ms), for 3 colors: 4.61 frames/s (66+24+58+ 69ms),
 // for 4 colors: 3.62 frames/s (66+24+58+69+ 59ms)
{
 // col_n_textures assuming size(col_sxsy0_texture) is 4x4 size(col_n_textures[0])
 InitCol_n_TextureRect(&col_sxsy0_texture, -1.f, -1.f, 1.f, 1.f, CheckGL, &col_n_textures[Num][0],
 colpar[Num].getCol(3)); // idcol for col[Num]
 for(int texidx = 1; texidx<PSEUDOMIPMAPLEVELS; texidx++)
 Col_sum_TextureRect(&col_n_textures[Num][texidx-1], -1.f, -1.f, 1.f, 1.f, CheckGL, &col_n_textures[Num][texidx]);
 // col_sx_textures
 InitCol_sx_TextureRect(&col_sxsy0_texture, -1.f, -1.f, 1.f, 1.f, CheckGL, &col_sx_textures[Num][0],
 colpar[Num].getCol(3)); // idcol for col[Num]
 for(int texidx = 1; texidx<PSEUDOMIPMAPLEVELS; texidx++)
 Col_sum_TextureRect(&col_sx_textures[Num][texidx-1], -1.f, -1.f, 1.f, 1.f, CheckGL, &col_sx_textures[Num][texidx]);
 // col_sy_textures
 InitCol_sy_TextureRect(&col_sxsy0_texture, -1.f, -1.f, 1.f, 1.f, CheckGL, &col_sy_textures[Num][0],
 colpar[Num].getCol(3)); // idcol for col[Num]
 for(int texidx = 1; texidx<PSEUDOMIPMAPLEVELS; texidx++)
 Col_sum_TextureRect(&col_sy_textures[Num][texidx-1], -1.f, -1.f, 1.f, 1.f, CheckGL, &col_sy_textures[Num][texidx]);
 // col_sd2_textures
 InitCol_sd2_TextureRect(&col_sxsy0_texture, -1.f, -1.f, 1.f, 1.f, CheckGL, &col_sd2_textures[Num][0],
 colpar[Num].getCol(3)); // idcol for col[Num]
 for(int texidx = 1; texidx<PSEUDOMIPMAPLEVELS; texidx++)
 Col_sum_TextureRect(&col_sd2_textures[Num][texidx-1], -1.f, -1.f, 1.f, 1.f, CheckGL,
 &col_sd2_textures[Num][texidx]);
 // col_k_textures
 for(int texidx = 2; texidx<PSEUDOMIPMAPLEVELS; texidx++)
 Col_k_TextureRect(&col_n_textures[Num][texidx], &col_sx_textures[Num][texidx], &col_sy_textures[Num][texidx],
 ,&col_sd2_textures[Num][texidx], -1.f, -1.f, 1.f, 1.f, CheckGL, &col_k_textures[Num][texidx]);
}

// test framerate with this part added to "for 1 color: 11.2 frames/s (66+ 24ms)"
/// 20 ii with texidx(1): 8.0 frames/s (66+24+ 36ms)
// for(int ii= 0; ii<20; ii++)
// for(int texidx = 1; texidx<2; texidx++)
// Col_sum_TextureRect(&col_n_textures[0][texidx-1], -1.f, -1.f, 1.f, 1.f, CheckGL, &col_n_textures[0][texidx]);
// 20 ii with texidx(4): 8.0 frames/s (66+24+ 36ms)
// for(int ii= 0; ii<20; ii++)
// for(int texidx = 4; texidx<5; texidx++)
// Col_sum_TextureRect(&col_n_textures[0][texidx-1], -1.f, -1.f, 1.f, 1.f, CheckGL, &col_n_textures[0][texidx]);
// conclusion : GPU computing time is negligible, but initialisation takes about 1.8 ms/texture
```

Dans notre cas où il faut calculer pour 4 couleurs N, ΣX, ΣY, ΣD<sup>2</sup> et K sur 6 niveaux, on a pour chaque image 4x5x6 = 120 initialisations de textures, ce qui prend environ 216ms.

Peut-être pourrait-on utiliser la génération mipmap automatique de OpenGL ES 2.0 en utilisant **glGenerateMipmap** mais d'après la littérature (ex: iPhone 3D Programming: Developing Graphical Applications with OpenGL ES) cela ne permet pas de choisir son propre algorithme de traitement, par contre s'il suffit de faire de la moyenne sur plusieurs pixels, c'est probablement possible mais à vérifier car le hardware choisit peut-être lui-même l'algorithme qu'il juge optimal (je n'ai pas trop creusé).

On peut diminuer sensiblement le nombre de textures à condition d'en utiliser des plus grosses. Les calculs du GPU seront sans doute un peu plus compliqués mais ce ne sont pas eux qui coutent. L'idée est de regrouper sur 16 texels contigus (par exemple 4x4) les opérations N, ΣX, ΣY, ΣD<sup>2</sup> pour les 4 couleurs puisque ces opérations peuvent être réalisées en parallèle. Ceci est illustré sur

la figure suivante.

## col\_sxsy0\_texture

**nsss\_cccc\_texture[0]** où chaque texel contient la somme d'infos de 4x4 texels de col\_sxsy0\_texture

# OpenGL ES Shading Language 1.0 Quick Reference Card

The OpenGL® ES Shading Language is two closely-related languages which are used to create shaders for the vertex and fragment processors contained in the OpenGL ES processing pipeline.

[n.n] and [Table n.n] refer to sections and tables in the OpenGL ES Shading Language 1.0 specification at [www.opengl.org/registry/gles](http://www.opengl.org/registry/gles)

## Types [4.1]

A shader can aggregate these using arrays and structures to build more complex types. There are no pointer types.

### Basic Types

|                     |                                                  |
|---------------------|--------------------------------------------------|
| void                | no function return value or empty parameter list |
| bool                | Boolean                                          |
| int                 | signed integer                                   |
| float               | floating scalar                                  |
| vec2, vec3, vec4    | n-component floating point vector                |
| bvec2, bvec3, bvec4 | Boolean vector                                   |
| ivec2, ivec3, ivec4 | signed integer vector                            |
| mat2, mat3, mat4    | 2x2, 3x3, 4x4 float matrix                       |
| sampler2D           | access a 2D texture                              |
| samplerCube         | access cube mapped texture                       |

### Structures and Arrays [4.1.8, 4.1.9]

|            |                                                                                                                                      |                                                             |
|------------|--------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------|
| Structures | struct type-name {<br>members<br>} struct-name[];                                                                                    | // optional variable declaration,<br>// optionally an array |
| Arrays     | float foo[3];<br>* structures and blocks can be arrays<br>* only 1-dimensional arrays supported<br>* structure members can be arrays |                                                             |

## Operators and Expressions

**Operators** [5.1] Numbered in order of precedence. The relational and equality operators > <= >= == != evaluate to a Boolean. To compare vectors component-wise, use functions such as lessThan(), equal(), etc.

| Operator          | Description                                                                                                                 | Associativity |
|-------------------|-----------------------------------------------------------------------------------------------------------------------------|---------------|
| 1. ()             | parenthetical grouping                                                                                                      | N/A           |
| 2. [] () . ++ --  | array subscript<br>function call & constructor structure field or method selector, swizzler postfix increment and decrement | L - R         |
| 3. ++ -- + - !    | prefix increment and decrement unary                                                                                        | R - L         |
| 4. * /            | multiplicative                                                                                                              | L - R         |
| 5. + -            | additive                                                                                                                    | L - R         |
| 7. <> <= >=       | relational                                                                                                                  | L - R         |
| 8. == !=          | equality                                                                                                                    | L - R         |
| 12. &&            | logical and                                                                                                                 | L - R         |
| 13. ^             | logical exclusive or                                                                                                        | L - R         |
| 14.               | logical inclusive or                                                                                                        | L - R         |
| 15. ?:            | selection (Selects one entire operand. Use mix() to select individual components of vectors.)                               | L - R         |
| 16. = += -= *= /= | assignment arithmetic assignments                                                                                           | L - R         |
| 17. ,             | sequence                                                                                                                    | L - R         |

### Vector Components [5.5]

In addition to array numeric subscript syntax, names of vector components are denoted by a single letter. Components can be swizzled and replicated, e.g.: pos.xx, pos.yw

|              |                                                               |
|--------------|---------------------------------------------------------------|
| {x, y, z, w} | Use when accessing vectors that represent points or normals   |
| {r, g, b, a} | Use when accessing vectors that represent colors              |
| {s, t, p, q} | Use when accessing vectors that represent texture coordinates |

## Preprocessor [3.4]

### Preprocessor Directives

The number sign (#) can be immediately preceded or followed in its line by spaces or horizontal tabs.

```
#define #undef #if #ifdef #ifndef #else
#elif #endif #error #pragma #extension #version #line
```

### Examples of Preprocessor Directives

- "#version 100" in a shader program specifies that the program is written in GLSL ES version 1.00. It is optional. If used, it must occur before anything else in the program other than whitespace or comments.
- #extension extension\_name : behavior, where behavior can be require, enable, warn, or disable; and where extension\_name is the extension supported by the compiler

### Predefined Macros

|                            |                                                                                                               |
|----------------------------|---------------------------------------------------------------------------------------------------------------|
| __LINE__                   | Decimal integer constant that is one more than the number of preceding new-lines in the current source string |
| __FILE__                   | Decimal integer constant that says which source string number is currently being processed.                   |
| __VERSION__                | Decimal integer, e.g.: 100                                                                                    |
| GL_ES                      | Defined and set to integer 1 if running on an OpenGL-ES Shading Language.                                     |
| GL_FRAGMENT_PRECISION_HIGH | 1 if highp is supported in the fragment language, else undefined [4.5.4]                                      |

## Qualifiers

### Storage Qualifiers [4.3]

Variable declarations may be preceded by one storage qualifier.

|           |                                                                                                                                        |
|-----------|----------------------------------------------------------------------------------------------------------------------------------------|
| none      | (Default) local read/write memory, or input parameter                                                                                  |
| const     | Compile-time constant, or read-only function parameter                                                                                 |
| attribute | Linkage between a vertex shader and OpenGL ES for per-vertex data                                                                      |
| uniform   | Value does not change across the primitive being processed, uniforms form the linkage between a shader, OpenGL ES, and the application |
| varying   | Linkage between a vertex shader and fragment shader for interpolated data                                                              |

### Precision and Precision Qualifiers [4.5]

Any floating point, integer, or sampler declaration can have the type preceded by one of these precision qualifiers:

|         |                                                                                                                               |
|---------|-------------------------------------------------------------------------------------------------------------------------------|
| highp   | Satisfies minimum requirements for the vertex language. Optional in the fragment language.                                    |
| mediump | Satisfies minimum requirements for the fragment language. Its range and precision is between that provided by lowp and highp. |
| lowp    | Range and precision can be less than mediump, but still represents all color values for any color channel.                    |

For example:

```
lowp float color;
varying mediump vec2 Coord;
lowp ivec2 foo(lowp mat3);
highp mat4 m;
```

Ranges & precisions for precision qualifiers (FP=floating point):

|         | FP Range                              | FP Magnitude Range                    | FP Precision              | Integer Range                         |
|---------|---------------------------------------|---------------------------------------|---------------------------|---------------------------------------|
| highp   | (-2 <sup>62</sup> , 2 <sup>62</sup> ) | (2 <sup>-62</sup> , 2 <sup>62</sup> ) | Relative 2 <sup>-16</sup> | (-2 <sup>16</sup> , 2 <sup>16</sup> ) |
| mediump | (-2 <sup>14</sup> , 2 <sup>14</sup> ) | (2 <sup>-14</sup> , 2 <sup>14</sup> ) | Relative 2 <sup>-10</sup> | (-2 <sup>10</sup> , 2 <sup>10</sup> ) |
| lowp    | (-2, 2)                               | (2 <sup>-8</sup> , 2)                 | Absolute 2 <sup>-8</sup>  | (-2 <sup>8</sup> , 2 <sup>8</sup> )   |

A precision statement establishes a default precision qualifier for subsequent int, float, and sampler declarations, e.g.:

precision highp int;

### Invariant Qualifiers Examples [4.6]

|                                       |                                            |
|---------------------------------------|--------------------------------------------|
| #pragma STDGL invariant(all)          | Force all output variables to be invariant |
| invariant gl_Position;                | Qualify a previously declared variable     |
| invariant varying mediump vec3 Color; | Qualify as part of a variable declaration  |

### Order of Qualification [4.7]

When multiple qualifications are present, they must follow a strict order. This order is as follows.

invariant, storage, precision  
storage, parameter, precision

## Aggregate Operations and Constructors

### Matrix Constructor Examples [5.4]

```
mat2(float) // init diagonal
mat2(vec2, vec2); // column-major order
mat2(float, float, // column-major order
 float, float);
```

### Structure Constructor Example [5.4.3]

```
struct light {float intensity; vec3 pos;};
light lightVar = light(3.0, vec3(1.0, 2.0, 3.0));
```

### Matrix Components [5.6]

Access components of a matrix with array subscripting syntax.

For example:

```
mat4 m; // m represents a matrix
m[1] = vec4(2.0); // sets second column to all 2.0
m[0][0] = 1.0; // sets upper left element to 1.0
m[2][3] = 2.0; // sets 4th element of 3rd column to 2.0
```

Examples of operations on matrices and vectors:

```
m = f * m; // scalar * matrix component-wise
v = f * v; // scalar * vector component-wise
```

v = v \* v; // vector \* vector component-wise
m = m +/- m; // matrix component-wise addition/subtraction
m = m \* m; // linear algebraic multiply
m = v \* m; // row vector \* matrix linear algebraic multiply
m = m \* v; // matrix \* column vector linear algebraic multiply
f = dot(v, v); // vector dot product
v = cross(v, v); // vector cross product
m = matrixCompMult(m, m); // component-wise multiply

### Structure Operations [5.7]

Select structure fields using the period(.) operator. Other operators include:

|       |                |
|-------|----------------|
| .     | field selector |
| == != | equality       |
| =     | assignment     |

### Array Operations [4.1.9]

Array elements are accessed using the array subscript operator [ ]. For example:

diffuseColor += lightIntensity[3] \* NdotL;

# OpenGL ES Shading Language 1.0 Quick Reference Card

## Built-In Inputs, Outputs, and Constants [7]

Shader programs use Special Variables to communicate with fixed-function parts of the pipeline. Output Special Variables may be read back after writing. Input Special Variables are read-only. All Special Variables have global scope.

### Vertex Shader Special Variables [7.1]

#### Outputs:

| Variable                    | Description                                       | Units or coordinate system |
|-----------------------------|---------------------------------------------------|----------------------------|
| highp vec4 gl_Position;     | transformed vertex position                       | clip coordinates           |
| mediump float gl_PointSize; | transformed point size (point rasterization only) | pixels                     |

### Fragment Shader Special Variables [7.2]

Fragment shaders may write to `gl_FragColor` or to one or more elements of `gl_FragData[]`, but not both.

The size of the `gl_FragData` array is given by the built-in constant `gl_MaxDrawBuffers`.

#### Inputs:

| Variable                   | Description                                                 | Units or coordinate system    |
|----------------------------|-------------------------------------------------------------|-------------------------------|
| mediump vec4 gl_FragCoord; | fragment position within frame buffer                       | window coordinates            |
| bool gl_FrontFacing;       | fragment belongs to a front-facing primitive                | Boolean                       |
| mediump int gl_PointCoord; | fragment position within a point (point rasterization only) | 0.0 to 1.0 for each component |

#### Outputs:

| Variable                     | Description                           | Units or coordinate system |
|------------------------------|---------------------------------------|----------------------------|
| mediump vec4 gl_FragColor;   | fragment color                        | RGB color                  |
| mediump vec4 gl_FragData[n]; | fragment color for color attachment n | RGB color                  |

## Built-In Functions

### Angle & Trigonometry Functions [8.1]

Component-wise operation. Parameters specified as `angle` are assumed to be in units of radians. T is float, `vec2`, `vec3`, `vec4`.

|                      |                    |
|----------------------|--------------------|
| T radians(T degrees) | degrees to radians |
| T degrees(T radians) | radians to degrees |
| T sin(T angle)       | sine               |
| T cos(T angle)       | cosine             |
| T tan(T angle)       | tangent            |
| T asin(T x)          | arc sine           |
| T acos(T x)          | arc cosine         |
| T atan(Ty, T x)      | arc tangent        |
| T atan(Ty, over_x)   | arc tangent        |

### Exponential Functions [8.2]

Component-wise operation. T is float, `vec2`, `vec3`, `vec4`.

|                    |                     |
|--------------------|---------------------|
| T pow(T x, T y)    | $x^y$               |
| T exp(T x)         | $e^x$               |
| T log(T x)         | $\ln x$             |
| T exp2(T x)        | $2^x$               |
| T log2(T x)        | $\log_2 x$          |
| T sqrt(T x)        | square root         |
| T inversesqrt(T x) | inverse square root |

### Common Functions [8.3]

Component-wise operation. T is float, `vec2`, `vec3`, `vec4`.

|                                             |                                               |
|---------------------------------------------|-----------------------------------------------|
| T abs(T x)                                  | absolute value                                |
| T sign(T x)                                 | returns -1.0, 0.0, or 1.0                     |
| T floor(T x)                                | nearest integer $\leq x$                      |
| T ceil(T x)                                 | nearest integer $\geq x$                      |
| T fract(T x)                                | $x - \text{floor}(x)$                         |
| T mod(T x, T y)                             | modulus                                       |
| T mod(T x, float y)                         |                                               |
| T min(T x, T y)                             | minimum value                                 |
| T min(T x, float y)                         |                                               |
| T max(T x, T y)                             | maximum value                                 |
| T max(T x, float y)                         |                                               |
| T clamp(T x, T minVal, T maxVal)            |                                               |
| T clamp(T x, float minVal, float maxVal)    | $\min(\max(x, \text{minVal}), \text{maxVal})$ |
| T mix(T x, T y, T a)                        | linear blend of x and y                       |
| T mix(T x, T y, float a)                    |                                               |
| T step(T edge, T x)                         | 0.0 if $x < \text{edge}$ , else 1.0           |
| T step(float edge, T x)                     |                                               |
| T smoothstep(T edge0, T edge1, T x)         | clip and smooth                               |
| T smoothstep(float edge0, float edge1, T x) |                                               |

### Geometric Functions [8.4]

These functions operate on vectors as vectors, not component-wise. T is float, `vec2`, `vec3`, `vec4`.

|                                 |                                                           |
|---------------------------------|-----------------------------------------------------------|
| float length(T x)               | length of vector                                          |
| float distance(T p0, T p1)      | distance between points                                   |
| float dot(T x, T y)             | dot product                                               |
| vec3 cross(vec3 x, vec3 y)      | cross product                                             |
| T normalize(T x)                | normalize vector to length 1                              |
| T faceforward(T N, T I, T Nref) | returns N if $\dot{\text{dot}}(Nref, I) < 0$ , else -N    |
| T reflect(T I, T N)             | reflection direction $I - 2 * \dot{\text{dot}}(N, I) * N$ |
| T refract(T I, T N, float eta)  | refraction vector                                         |

### Matrix Functions [8.5]

Type mat is any matrix type.

mat matrixCompMult(mat x, mat y) multiply x by y component-wise

### Vector Relational Functions [8.6]

Compare x and y component-wise. Sizes of input and return vectors for a particular call must match. Type bvec is `bvecn`; vec is `vecn`; ivec is `ivcn` (where n is 2, 3, or 4). T is the union of vec and ivec.

|                                 |                                      |
|---------------------------------|--------------------------------------|
| bvec lessThan(T x, T y)         | $x < y$                              |
| bvec lessThanEqual(T x, T y)    | $x \leq y$                           |
| bvec greaterThan(T x, T y)      | $x > y$                              |
| bvec greaterThanEqual(T x, T y) | $x \geq y$                           |
| bvec equal(T x, T y)            | $x = y$                              |
| bvec equal(bvec x, bvec y)      | $\text{bvec} \neq y$                 |
| bvec notEqual(T x, T y)         | $x \neq y$                           |
| bvec notEqual(bvec x, bvec y)   | $\text{bvec} \neq y$                 |
| bool any(bvec x)                | true if any component of x is true   |
| bool all(bvec x)                | true if all components of x are true |
| bvec not(bvec x)                | logical complement of x              |

### Texture Lookup Functions [8.7]

Available only in vertex shaders.

|                                                                 |
|-----------------------------------------------------------------|
| vec4 texture2DLod(sampler2D sampler, vec2 coord, float lod)     |
| vec4 texture2DProjLod(sampler2D sampler, vec3 coord, float lod) |
| vec4 texture2DProjLod(sampler2D sampler, vec4 coord, float lod) |
| vec4 textureCubeLod(samplerCube sampler, vec3 coord, float lod) |

Available only in fragment shaders.

|                                                               |
|---------------------------------------------------------------|
| vec4 texture2D(sampler2D sampler, vec2 coord, float bias)     |
| vec4 texture2DProj(sampler2D sampler, vec3 coord, float bias) |
| vec4 texture2DProj(sampler2D sampler, vec4 coord, float bias) |
| vec4 textureCube(samplerCube sampler, vec3 coord, float bias) |

Available in vertex and fragment shaders.

|                                                   |
|---------------------------------------------------|
| vec4 texture2D(sampler2D sampler, vec2 coord)     |
| vec4 texture2DProj(sampler2D sampler, vec3 coord) |
| vec4 texture2DProj(sampler2D sampler, vec4 coord) |
| vec4 textureCube(samplerCube sampler, vec3 coord) |

## Built-In Constants With Minimum Values [7.4]

| Built-in Constant                                 | Minimum value |
|---------------------------------------------------|---------------|
| const mediump int gl_MaxVertexAttribs             | 8             |
| const mediump int gl_MaxVertexUniformVectors      | 128           |
| const mediump int gl_MaxVaryingVectors            | 8             |
| const mediump int gl_MaxVertexTextureImageUnits   | 0             |
| const mediump int gl_MaxCombinedTextureImageUnits | 8             |
| const mediump int gl_MaxTextureImageUnits         | 8             |
| const mediump int gl_MaxFragmentUniformVectors    | 16            |
| const mediump int gl_MaxDrawBuffers               | 1             |

### Built-In Uniform State [7.5]

Specifies depth range in window coordinates. If an implementation does not support highp precision in the fragment language, and state is listed as highp, then that state will only be available as mediump in the fragment language.

struct `gl_DepthRangeParameters` {

```
 highp float near; // n
 highp float far; // f
 highp float diff; // f - n
};
```

uniform `gl_DepthRangeParameters` gl\_DepthRange;

## Statements and Structure

### Iteration and Jumps [6]

| Function Call | call by value-return              |
|---------------|-----------------------------------|
| Iteration     | for (;;) { break, continue }      |
|               | while () { break, continue }      |
|               | do { break, continue } while () ; |
| Selection     | if () {}                          |
|               | if () {} else {}                  |
| Jump          | break, continue, return           |
|               | discard // Fragment shader only   |
| Entry         | void main()                       |

## Sample Program

A shader pair that applies diffuse and ambient lighting to a textured object.

### Vertex Shader

```
uniform mat4 mvp_matrix; // model-view-projection matrix
uniform mat3 normal_matrix; // normal matrix
uniform vec3 ec_light_dir; // light direction in eye coords

attribute vec4 a_vertex; // vertex position
attribute vec3 a_normal; // vertex normal
attribute vec2 a_texcoord; // texture coordinates

varying float v_diffuse;
varying vec2 v_texcoord;
```

void main(void)

```
{
 // put vertex normal into eye coords
 vec3 ec_normal = normalize(normal_matrix * a_normal);

 // emit diffuse scale factor, texcoord, and position
 v_diffuse = max(dot(ec_light_dir, ec_normal), 0.0);
 v_texcoord = a_texcoord;
 gl_Position = mvp_matrix * a_vertex;
}
```

### Fragment Shader

```
precision mediump float;
uniform sampler2D t_reflectance;
uniform vec4 i_ambient;
varying float v_diffuse;
varying vec2 v_texcoord;

void main (void)
{
 vec4 color = texture2D(t_reflectance, v_texcoord);
 gl_FragColor = color * (vec4(v_diffuse) + i_ambient);
}
```



A scene in Minecraft, before and after applying [a few shaders](#).

## The Aim of This Guide

Shader programming sometimes comes off as an enigmatic black magic and is often misunderstood. There are lots of code samples out there that show you how to create incredible effects, but offer little or no explanation. This guide aims to bridge that gap. I'll focus more on the basics of writing and understanding shader code, so you can easily tweak, combine, or write your own from scratch!

This is a general guide, so what you learn here will apply to anything that can run shaders.

## So What is a Shader?

A shader is simply a program that runs in the graphics pipeline and tells the computer how to render each pixel. These programs are called shaders because they're often used to control [lighting](#) and [shading](#) effects, but there's no reason they can't handle other special effects.

Shaders are written in a special shading language. Don't worry, you don't have to go out and learn a completely new language; we will be using GLSL (OpenGL Shading Language) which is a C-like language. (There are a bunch of [shading languages](#) out there for different platforms, but since they're all adapted to run on the GPU, they're all very similar)

*Click*

## Let's Jump In!

We'll use [ShaderToy](#) for this tutorial. This lets you start programming shaders right in your browser, without the hassle of setting anything up! (It uses WebGL for rendering, so you'll need a browser that can support that.) [Creating an account](#) is optional, but handy for saving your code.

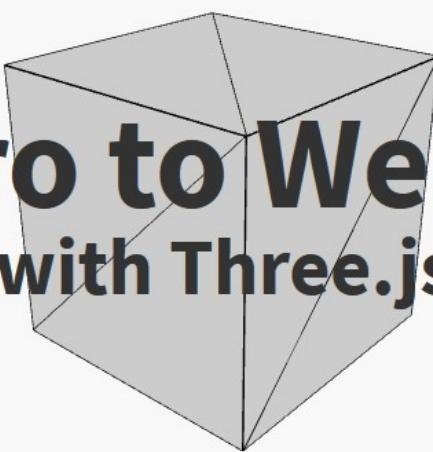
*Note:* ShaderToy is in beta at the time of writing this article. Some small UI/syntax details may be slightly different.

Upon clicking [New Shader](#), you should see something like this:

```

// init_nsss_vertshader.glsl - vertex shader used here essentially to compute coordinates to send to
fragment shader
// ECAM - Bruxelles, GEI, F. Gueuning 150302
attribute vec4 vertex;
5 uniform vec2 offset; // probably (-1., -1.)
uniform vec2 scale; // probably (2., 2.)
uniform vec2 texturesize; // probably (384., 256.)
// For varying, vertex shader computes only one value at each vertex. Values are then interpolated
by opengl
10 // between vertices in order to be given to fragment shader for each fragment.
varying highp vec2 tcoord; // current texel coordinates
varying highp vec2 cxry; // cx: current column between 0 and texturesize.x-1)
// ry: current row (between 0 and texturesize.y-1)
void main(void)
15 {
 vec4 pos = vertex;
 tcoord.xy = pos.xy;
 cxry = floor(pos.xy * texturesize);
 pos.xy = pos.xy*scale+offset;
20 gl_Position = pos;
}

```



# **Intro to WebGL with Three.js**

<http://davidscottlyons.com/threejs/presentations/frontporch14/#slide-0>

## Résonance magnétique nucléaire (RMN)

Extrait de "Principes de l'imagerie RM", document Philips, division Matériel Médical, 1985

### Histoire de la résonance magnétique

La résonance magnétique (RM) est un phénomène qui fut découvert séparément en 1946 par Purcell et Bloch. Tous deux se virent décerner, en 1952, le Prix Nobel de physique pour les découvertes et développements ultérieurs de méthodes spectroscopiques basées sur ce phénomène. Depuis lors, l'instrumentation RM a continué à évoluer et s'avère un outil précieux dans le domaine de la chimie physique et de la biochimie.

Au début des années 70, le principe de la RM a été repris dans une proposition faite par Lauterbur, qui suggérait que le spectroscope pourrait être modifié de manière à fournir des signaux à code spatial, afin de permettre d'étudier des objets non-homogènes le début de l'imagerie RM en médecine. On découvrit que cette méthode permettait d'obtenir, dans les tissus mous, un contraste supérieur à celui qu'offraient les autres techniques employées. Par la suite, les techniques d'imagerie RM ont fait l'objet d'améliorations incessantes et, en 1983, on disposait déjà de systèmes permettant de réaliser des diagnostics du corps humain, avec une durée d'imagerie réduite d'une heure environ à quelques minutes seulement. Simultanément, la résolution spatiale avait été améliorée de 6 mm à moins de 1 mm.

### Le spin et la précession du noyau

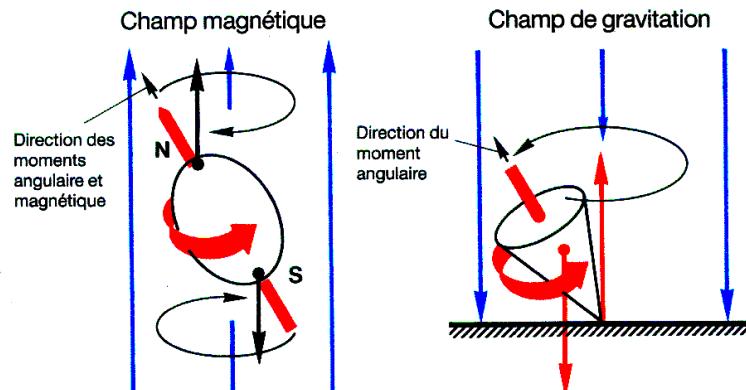
Une toupie présente deux mouvements simultanés, c'est-à-dire une rotation autour de son axe et un mouvement de précession autour d'un axe vertical (ou direction du champ de gravitation). La vitesse de précession d'une toupie dépend des facteurs suivants :

- le moment angulaire (c'est-à-dire le spin) et les caractéristiques mécaniques de la toupie
- la force du champ de gravitation.

Certains noyaux atomiques sont assez semblables à une toupie. Le moment angulaire intrinsèque ou spin est l'une des propriétés fondamentales de la matière au niveau des particules élémentaires telles que les protons et les neutrons. De ce fait, certains noyaux atomiques présentent un spin. Un noyau atomique possédant un nombre impair de neutrons ou de protons (ou des deux) présente un moment angulaire.

L'étroite analogie qui existe entre le noyau en rotation d'un champ magnétique et une toupie est représentée sur la figure ci-contre.

L'une des différences fondamentales entre la toupie et le noyau est que le noyau possède un spin intrinsèque, alors qu'il faut faire tournoyer la toupie. Néanmoins, tous deux présentent également un mouvement de précession. Les deux tiers environ de tous les noyaux stables présentent un spin. Étant donné que tous les noyaux ont une charge positive, un courant circule dans tous les noyaux en spin et, par conséquent, ils possèdent un magnétisme associé. Un noyau en spin se comporte comme un minuscule barreau aimanté qui gravite autour de son axe longitudinal. Autrement dit, un noyau en spin et chargé produit un moment magnétique. (Un moment magnétique est un vecteur



physique exprimant l'importance et la direction d'un champ magnétique quelconque). Le moment magnétique suit la direction de l'axe du spin.

Considérons un compas dans un champ magnétique. Le fait que son aiguille, après avoir été déviée manuellement, reprend toujours une orientation favorite, signale qu'il existe un champ. Ceci est à la base de toutes les techniques RM. Certains noyaux atomiques se comportent comme les aiguilles d'un compas. Néanmoins, pour mesurer diverses caractéristiques des noyaux, telle que leur fréquence de précession, il faut au préalable les perturber. S'ils sont perturbés en présence d'un champ magnétique uniforme, les noyaux en spin présentent un mouvement de précession dans le sens du champ. Ce phénomène est analogue à la précession de la toupie dans le sens du champ de gravitation.

La vitesse de précession d'un noyau dépend en majeure partie des paramètres suivants:

- les caractéristiques du noyau en question
- l'induction  $B$  du champ magnétique

Les noyaux qui présentent un spin sont soumis à un couple qui les force à tournoyer dans le sens du champ. Leurs moments magnétiques correspondent à la force du couple. Les axes de spin changent constamment de direction.

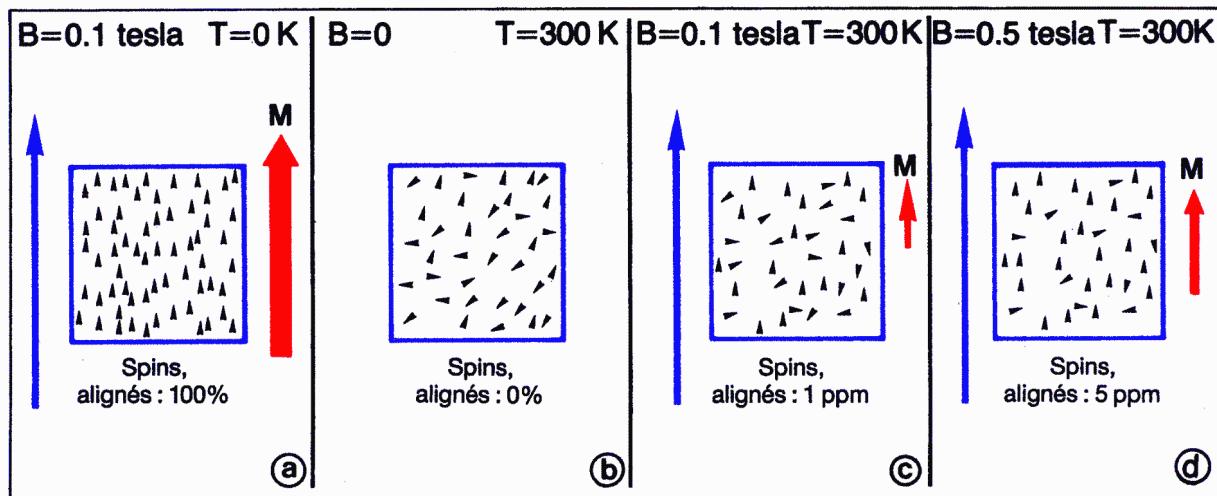
Ce mouvement est dénommé *précession de Larmor*.

La précession est un effet de résonance. Si un système donné possède une résonance ou fréquence d'oscillation propre, c'est à cette fréquence que l'énergie peut lui être appliquée de la manière la plus efficace. La précession du noyau peut être perturbée par absorption d'énergie électromagnétique à la fréquence de précession. Ainsi, la résonance des noyaux est excitée et, au retour à l'équilibre, de l'énergie se dégage à la même fréquence.

### Magnétisation résultante

Il est impossible d'effectuer des mesures au niveau d'un seul noyau. Il faut donc étudier le comportement de nombreux noyaux dans un corps ou dans un échantillon. Du fait des mouvements produits par l'énergie thermique, les noyaux d'un ensemble d'atomes sont normalement orientés de manière aléatoire et le moment magnétique macroscopique d'un échantillon de la matière sera nul. Par contre, dans un champ magnétique, les moments magnétiques des divers noyaux interagissent avec le champ appliqué et produisent un moment magnétique résultant au sein de l'échantillon.

*Représentation de l'influence de la température et de la puissance du champ magnétique appliquée sur la magnétisation nette.*



Si la température de l'échantillon est le 0 absolu, et si on en avait extrait l'énergie totale, tous les spins seraient alignés et l'on obtiendrait alors une importante magnétisation résultante. Ceci

est représenté sur la figure a (page précédente) sur laquelle M est le moment magnétique résultant. (L'unité d'induction magnétique est le tesla T).

Si la température n'est pas égale à 0 et en l'absence de tout champ magnétique, le mouvement dû à l'énergie thermique rend les orientations entièrement aléatoires (voir figure b). Dans le cadre d'une expérience réelle, c'est-à-dire en présence d'un champ magnétique et d'énergie thermique, les tendances opposées entraînent une certaine orientation du champ et, par conséquent, un faible taux de magnétisation résultante M (voir figure c).

Dans un échantillon contenu dans un champ de 0,1 tesla environ et à la température ambiante, l'orientation résultante est en réalité uniquement due à un proton par million environ. Par conséquent, la magnétisation résultante n'est égale qu'à un millionième de ce qu'elle deviendrait au zéro absolu.

En RM, c'est la précession de ce minuscule vecteur de magnétisation résultante M que l'on observe. Dès lors, les signaux RM sont extrêmement faibles et il est difficile d'obtenir un rapport signal/bruit élevé. Pour faire contribuer davantage de protons au signal RM, il faut accroître la force du champ appliqué (voir figure d); néanmoins, le moment magnétique résultant demeure petit.

### **Précession de Larmor**

Pour un certain nombre de raisons, dont la principale est que les protons ( $H^+$ ) sont le type de noyau le plus abondant dans les tissus biologiques, la plupart des images *in vivo* produites sont basées sur le proton. Par conséquent, certaines explications données ici sont parfois basées sur l'hypothèse que le noyau examiné est celui de l'hydrogène, étant donné que le corps humain se compose en majeure partie d'eau.

Une molécule d'eau contient deux atomes d'hydrogène, le noyau de l'isotope d'hydrogène commun étant un proton. Le proton est une particule chargée qui présente un spin et, par conséquent, un moment magnétique. S'il était possible d'examiner les divers protons d'un verre d'eau reposant dans un champ magnétique uniforme, on constaterait que leur précession survient approximativement à une fréquence identique mais que, du fait de l'énergie thermique, ces précessions s'effectuent dans des directions différentes et sont totalement déphasées les unes par rapport aux autres. Suite à l'application d'une impulsion à leur fréquence angulaire propre, on constaterait que la précession de certains protons s'effectue en phase pendant un certain temps, car ils ont absorbé de l'énergie électromagnétique. Ce mouvement est dénommé *précession de Larmor* et la fréquence angulaire s'appelle *fréquence de Larmor*.

Durant leur relaxation ou leur retour à l'équilibre thermique, les noyaux dégagent, à la même fréquence, de l'énergie électromagnétique. Ce phénomène peut être détecté par une antenne dans laquelle un courant a été induit, ce qui produit le signal RM. Le phénomène ainsi observé est dénommé résonance magnétique nucléaire.

Lorsque la fréquence de l'impulsion correspond à la fréquence de Larmor d'un certain type de noyau, la précession de ce type de noyau sera perturbée ou excitée.

La fréquence de précession de Larmor (fréquence angulaire) est directement proportionnelle à la force du champ magnétique  $B_0$ , comme l'indique l'équation ci-dessous:

$$f = \frac{\gamma}{2\pi} \cdot B_0 \quad (\text{équation de Larmor})$$

dans laquelle  $\gamma$  est la constante de proportionnalité, dénommée moment gyromagnétique.

La valeur du moment gyromagnétique dépend du type de noyau en cause. Ainsi, lorsque l'on connaît la force du champ, on peut déterminer le type de noyau (ou élément), pourvu que l'on puisse mesurer la fréquence de précession.

Pour récapituler: on observe le phénomène de RM à l'aide de la séquence d'événements suivante :

- Dans un premier temps, les noyaux sont orientés de manière aléatoire.
- Lorsqu'ils entrent dans un puissant champ magnétique, ils ont tendance à s'aligner sur ce champ.
- Les précessions sont alors excitées par rayonnement électromagnétique à la fréquence de Larmor appropriée.
- A la fin de l'impulsion, les noyaux se relaxent et dégagent, à la même fréquence, l'énergie précédemment absorbée.

### Le moment gyromagnétique

| Isotope   | $\gamma / 2\pi$ |
|-----------|-----------------|
| $^1H$     | 42.58           |
| $^{19}F$  | 40.05           |
| $^{31}P$  | 17.24           |
| $^{23}Na$ | 11.26           |
| $^{13}C$  | 10.71           |
| $^{17}O$  | 5.77            |

Si on exprime l'induction magnétique en teslas et la fréquence en hertz, on obtient les valeurs du moment gyromagnétique  $\gamma$  (divisé par  $2\pi$ ) reprises au tableau de gauche pour les principaux isotopes avec

spin nucléaire survenant dans les tissus. Le tableau de droite démontre que, pour un champ quelconque, le type de noyau qui sera excité dépendra de la fréquence d'excitation sélectionnée. Par exemple, à une force de champ de 0,5T, une impulsion à 21,29 MHz excitera les protons, tandis qu'une impulsion à 8,62 MHz excitera les noyaux de phosphore. On a produit des images médicales à l'aide de puissances situées dans la plage figurant sur le tableau, et pour tous les isotopes énumérés. Ainsi, la fréquence de Larmor constatée sera dans la gamme de 1 MHz à 100 MHz et l'impulsion de rayonnement électromagnétique nécessaire à l'excitation sera une radiofréquence. C'est pourquoi on emploie des impulsions en radiofréquences (RF) pour former des images par résonance magnétique nucléaire.

| Isotope   | Champ (Tesla) | Champ (kGauss) | Fréquence de Larmor (Mhz) |
|-----------|---------------|----------------|---------------------------|
| $^1H$     | 0.04          | 0.4            | 1.703                     |
|           | 0.15          | 1.5            | 6.387                     |
|           | 0.50          | 5.0            | 21.29                     |
|           | 1.00          | 10.0           | 42.58                     |
|           | 1.50          | 15.0           | 63.87                     |
|           | 2.00          | 20.0           | 85.16                     |
| $^{31}P$  | 0.50          | 5.0            | 8.62                      |
|           | 1.00          | 10.0           | 17.24                     |
|           | 1.50          | 15.0           | 25.86                     |
|           | 2.00          | 20.0           | 34.48                     |
| $^{23}Na$ | 2.00          | 20.0           | 22.52                     |

### Décalage chimique (intérêt en spectroscopie RM, pas en imagerie)

Dans la pratique, la fréquence à laquelle s'effectue la précession d'un noyau donné ne correspond pas toujours exactement au calcul établi sur la base du moment gyromagnétique, car elle est également influencée par certains facteurs chimiques de l'environnement. Cet écart de fréquence est dénommé le *décalage chimique*, il est dû à un léger manque d'uniformité du champ magnétique local créé par la structure moléculaire environnante.

Par exemple, dans l'adénosine triphosphate ou ATP, trois atomes de phosphore se trouvent dans des sites moléculaires différents, c'est-à-dire que leur environnement chimique diffère. Un isotope qui s'est avéré particulièrement intéressant dans les études métaboliques est le phosphore  $^{31}(^{31}P)$  dans les trois groupes de phosphate différents de l'ATP, qui constituent la source de l'énergie biochimique. Trois sommets apparaissent dans le spectre relatif à cette molécule. Par conséquent, chaque type de noyau de phosphore produit une composante en fréquence légèrement différente.

La fréquence de Larmor et le décalage chimique de cette fréquence sont tous deux proportionnels à la force du champ. Pour normaliser les spectres obtenus avec des champs de forces différentes, on divise généralement l'échelle par la fréquence correspondant à un point

de référence central, à la force de champ utilisée et on exprime le décalage en termes fractionnaires ou ppm (parties par million). Ainsi, les valeurs de décalage chimique demeurent indépendantes de la force du champ.

Pour extraire des informations détaillées au niveau moléculaire, il faut disposer d'une très fine résolution des fréquences du spectre et, par conséquent, de champs extrêmement puissants (c'est-à-dire supérieurs à 1 tesla). Plus le champ est puissant, plus la résolution peut être accrue. C'est pourquoi la spectroscopie RM est dénommée "RM haute résolution" (il y a lieu de ne pas confondre le terme "résolution", avec "résolution spatiale", ce dernier ayant trait aux images).

Ce degré de résolution des fréquences est généralement superflu dans l'imagerie; par conséquent, les puissances de champ nécessaires à la production d'images peuvent être bien moindres. Cependant, on a pu produire des images à décalage chimique protonique; l'une d'entre elles représente les protons de l'eau et une autre des protons contenus dans des liquides. Pour obtenir ces images, il faut disposer de champs de très grande puissance.

En outre, la spectroscopie RM exige une très grande uniformité du champ (de l'ordre de 1:10<sup>7</sup>) dans le petit volume d'échantillonnage, étant donné que les fréquences de Larmor sont en rapport avec la puissance du champ, sans quoi les résultats ne seront daucun intérêt. Ceci va à l'encontre des exigences d'imagerie: champ de grande dimension, mais uniformité moins importante.

### Eléments autres que l'hydrogène

En principe, tout isotope présentant un spin nucléaire et qui se trouve (ou peut être introduit) dans le corps, peut s'utiliser pour l'imagerie. Néanmoins, l'utilité d'un isotope quelconque pour la détection et le diagnostic dépend des facteurs suivants:

- son abondance naturelle dans le corps
- sa sensibilité (c'est-à-dire la puissance relative de ses signaux)
- son moment gyromagnétique

Dans le tableau ci-contre, les isotopes à spin nucléaire que l'on trouve dans les tissus mammaires sont agencés en ordre d'abondance naturelle ou concentration molaire.

On remarquera que, bien que le carbone et l'oxygène soient abondants dans le corps, ils possèdent peu d'isotopes à spin. Les plus abondants sont <sup>12</sup>C et <sup>16</sup>O, et ces isotopes ne présentent pas de spin. Seul un faible pourcentage de carbone ou d'oxygène est de type <sup>13</sup>C ou <sup>17</sup>O. Par conséquent, ce n'est pas uniquement

l'abondance naturelle de l'élément, mais l'abondance en pourcentage des formes isotopiques à spin nucléaire, dont il faut tenir compte.

| Isotopes avec spin | Concentration molaire (mole/L) | Moment $\gamma / 2\pi$ (MHz/T) | Sensibilité par rapport à <sup>1</sup> H |
|--------------------|--------------------------------|--------------------------------|------------------------------------------|
| <sup>1</sup> H     | 99.0                           | 42.58                          | 1                                        |
| <sup>14</sup> N    | 1.6                            | 3.08                           | -                                        |
| <sup>31</sup> P    | 0.35                           | 17.24                          | 0.066                                    |
| <sup>13</sup> C    | 0.10                           | 10.71                          | 0.016                                    |
| <sup>23</sup> Na   | 0.078                          | 11.26                          | 0.093                                    |
| <sup>39</sup> K    | 0.045                          | 1.99                           | 0.0005                                   |
| <sup>17</sup> O    | 0.031                          | 5.77                           | 0.029                                    |
| <sup>2</sup> H     | 0.015                          | 6.53                           | 0.0096                                   |
| <sup>19</sup> F    | 0.0066                         | 40.05                          | 0.830                                    |

La *sensibilité* est une mesure de la puissance des signaux provenant d'un certain nombre de noyaux de l'isotope en question, par rapport à celle d'un nombre égal de noyaux <sup>1</sup>H. D'après le tableau, on pourra constater que le signal le plus puissant s'obtient à partir de <sup>1</sup>H, étant donné que cet isotope est non seulement le plus abondant, mais possède en outre la sensibilité la plus élevée. Par contraste, l'abondance et la sensibilité de <sup>31</sup>P sont beaucoup plus faibles. Autre exemple: la sensibilité de <sup>19</sup>F est élevée, mais il est le plus rare.

Par conséquent, il est beaucoup plus facile de produire des images basées sur le noyau d'hydrogène (le proton) que sur des noyaux de phosphore ou de fluor. Cependant, l'imagerie

basée sur des éléments autres que l'hydrogène est intéressante, car chaque élément joue son propre rôle dans le métabolisme, et peut donc fournir d'autres informations intéressantes pour le diagnostic.

La valeur du moment gyromagnétique est également importante dans l'évaluation de l'utilité d'un isotope quelconque. Plus cette valeur est élevée, plus la fréquence de Larmor augmente à une induction donnée. Les inductions utilisées en pratique fournissent généralement des fréquences de Larmor dans la "fenêtre RF" du corps. Néanmoins, en règle générale, l'absorption augmente en fonction de la fréquence. Si l'absorption est excessive, il faudra disposer de puissances RF supérieures pour exciter les précessions, et les signaux renvoyés seront soumis à un niveau de réabsorption plus élevé.

### Magnétisation transversale à la résonance

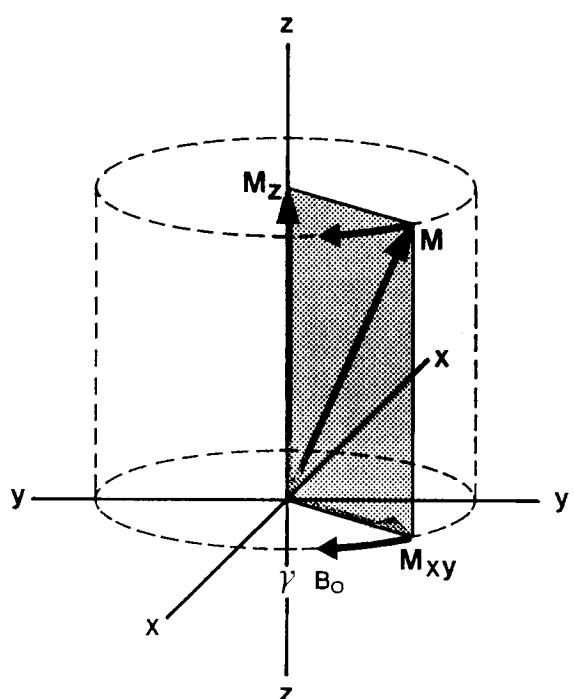
Supposons que l'on applique à un ensemble de protons (une zone du corps humain par exemple) un champ  $B_0$  et que d'un point de vue macroscopique on se trouve dans un état d'équilibre. La magnétisation résultante produite par l'ensemble de protons est alignée sur le champ magnétique  $B_0$ . S'il est perturbé, le système se comporte comme un noyau individuel, c'est-à-dire qu'il présente, vu de façon macroscopique, un mouvement de précession autour de la direction du champ  $B_0$ , et comme pour le noyau individuel, la fréquence de précession peut être calculée par l'équation de Larmor et dépend de la puissance du champ magnétique. Pour provoquer la perturbation, on peut faire appel à un champ magnétique additionnel  $B_1$  appliqué perpendiculairement au champ principal  $B_0$ , et qui tourne à la fréquence de Larmor  $f_1$ , que l'on obtient en calculant  $f_1 = \gamma \cdot B_0$ .

La magnétisation en précession s'écarte de plus en plus de la direction du champ principal  $z$ , cet écart étant fonction de l'intensité et de la durée du champ  $B_1$ , lequel est une brève impulsion magnétique en radiofréquence. Le mouvement de précession décrit un cône. Au fur et à mesure que la perturbation augmente, ce cône s'aplatit jusqu'à ce qu'il devienne un disque, ou même un cône inversé.

Le vecteur de magnétisation  $M$  peut être considéré comme possédant une composante  $M_z$  le long de l'axe du champ principal, et une autre composante  $M_{xy}$ , située dans un plan perpendiculaire à l'axe du champ (voir figure).

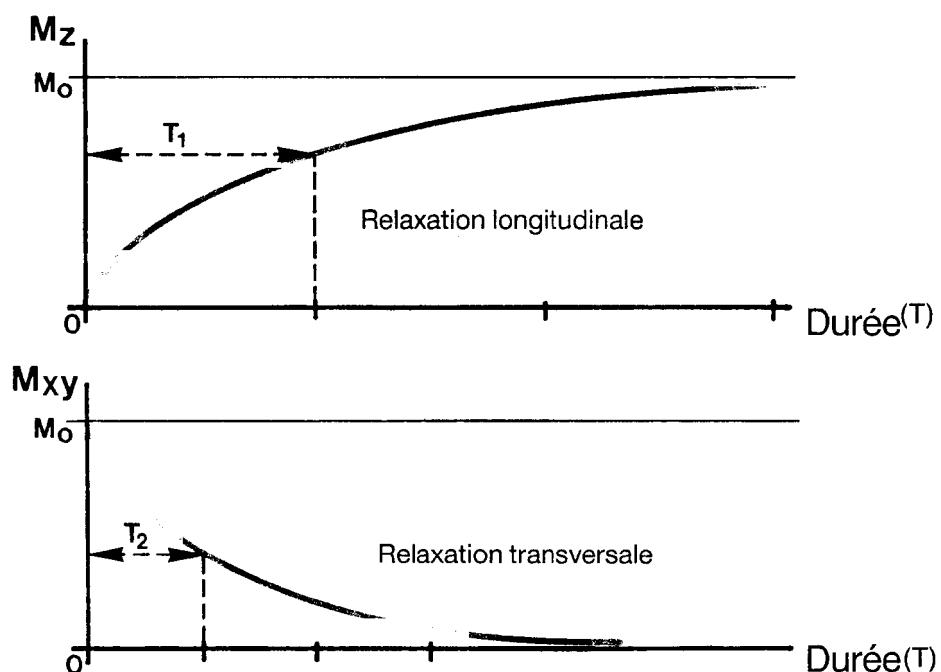
La "magnétisation longitudinale,,  $M_z$  au point d'équilibre est égale et équivalente à la magnétisation résultante  $M_0$ . La "magnétisation transversale,,  $M_{xy}$  au point d'équilibre est nulle, Lors d'une excitation par impulsion radiofréquence,  $M_z$  décroît au fur et à mesure que  $M_{xy}$  augmente,  $M_{xy}$  est en précession autour de l'axe  $z$  à la fréquence de Larmor obtenue en multipliant la valeur appropriée du moment gyromagnétique par la puissance de champ  $B_0$ . L'impulsion magnétique (transversale) est appelée *impulsion à 90°* si elle provoque une oscillation transversale  $M_{xy}$  des noyaux à la fréquence de Larmor, et *impulsion à 180°* si elle conduit à  $M_z$  en sens opposé à  $B_0$ .

En résumé, à partir d'un champ  $B_1$  perpendiculaire à  $B_0$ , on peut produire une magnétisation résultante  $M$  de l'ensemble des protons comportant une composante  $M_{xy}$  perpendiculaire à  $B_0$  avec mouvement de précession, ou même une composante longitudinale  $M_z$  en opposition par rapport à  $B_0$ , bien que produite par un champ  $B_1$  transverse (c'est-à-dire perpendiculaire à  $B_0$ ) appliqué judicieusement.



### Temps de relaxation longitudinale ou spin-réseau T1

Après excitation, les noyaux reviennent à l'état d'équilibre. Ils perdent de l'énergie par rayonnement électromagnétique ainsi que par transfert d'énergie aux molécules qui les entourent. Ce processus est appelé "relaxation" et débute à la fin de l'impulsion RF. La magnétisation longitudinale  $M_z$  revient exponentiellement avec une constante de temps  $T_1$  vers sa valeur initiale  $M_0$ . Dans l'intervalle, la magnétisation transversale  $M_{xy}$  aura diminué jusqu'à zéro également de façon exponentielle, mais avec une constante  $T_2$  plus faible (figure). Les valeurs de  $T_1$  et  $T_2$  dépendent entre autres de la structure moléculaire de la matière, de son état physique (liquide ou solide par exemple) et de sa température. De plus, le temps  $T_1$  de relaxation longitudinale augmente en fonction de l'accroissement de puissance du champ magnétique.

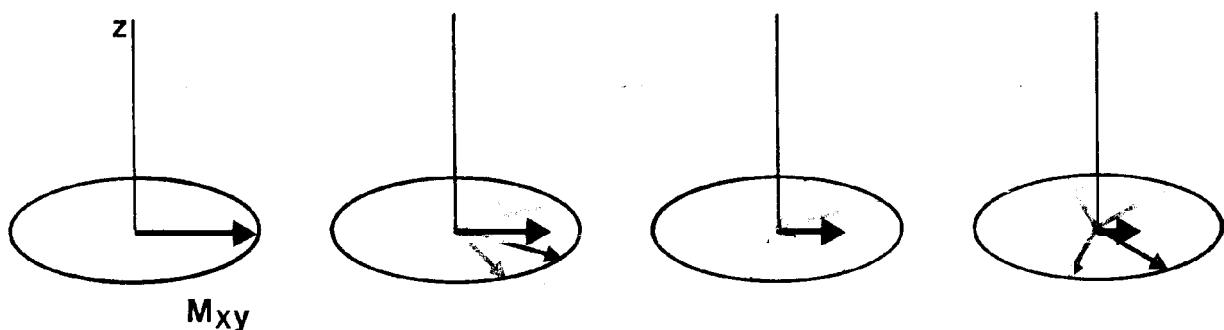


Ce temps  $T_1$  est appelé temps de relaxation spin-réseau parce que son état d'équilibre est atteint par l'intermédiaire d'un échange d'énergie avec l'environnement (ou réseau) dans lequel les spins sont noyés.  $T_1$  caractérise la vitesse à laquelle l'énergie peut être transférée entre les noyaux tournants et le réseau, par collisions aléatoires entre les molécules. Dans les tissus biologiques, la valeur de  $T_1$  va de 50 millisecondes environ à quelques secondes.

### Temps de relaxation transversale ou "spin-spin" T2

Au point d'équilibre thermique, et dans un champ magnétique, la précession de tous les protons s'effectue à la fréquence de Larmor de ces protons mais, dû au manque de coordination en phase, le seul effet résultant est dans l'orientation longitudinale. *L'impulsion à 90°* agit sur les protons en cours de rotation, de manière à ce qu'ils commencent à tourner en phase et produisent un vecteur de magnétisation transversale à la fréquence de Larmor.

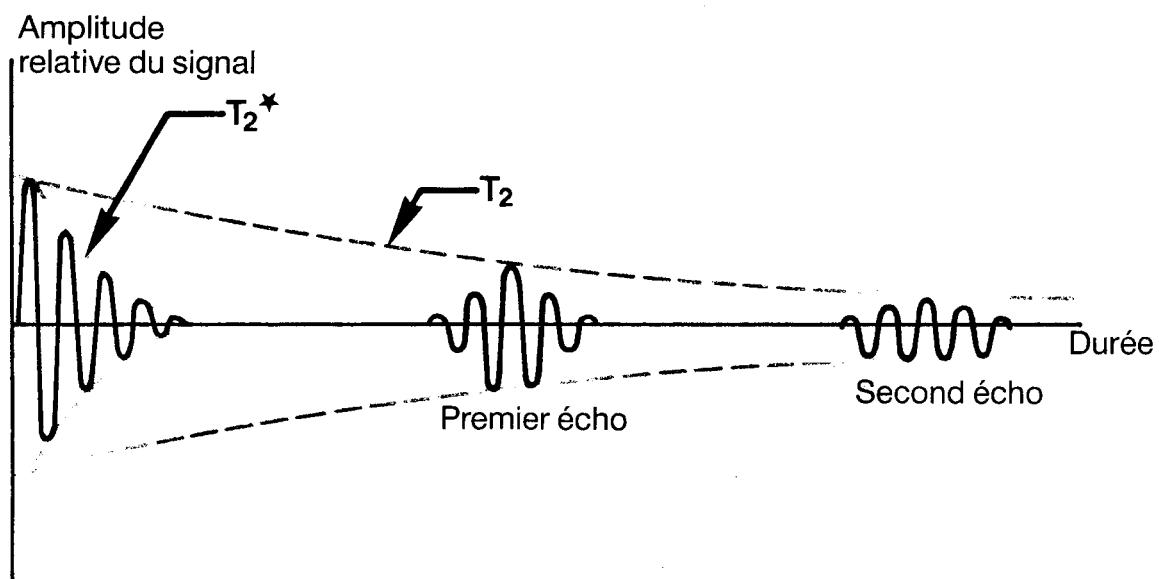
Le temps de relaxation  $T_2$  est une mesure qui correspond à la période durant laquelle les noyaux résonants conservent la magnétisation transversale temporaire. Immédiatement après l'excitation, tous les protons entrent ensemble en précession, et produisent une précession correspondante de la composante transversale. Néanmoins, étant donné que les interactions spin-spin entraînent des fluctuations locales aléatoires du champ magnétique, la fréquence de précession fluctue également de manière aléatoire autour de la fréquence de Larmor. Ceci entraîne un déphasage progressif et aléatoire des protons et, par conséquent, une réduction de la composante transversale de la magnétisation comme sur la figure qui suit.



Si l'on utilise une antenne de détection pour mesurer la magnétisation en plan transversal, le signal ainsi induit aura une composante de base à la fréquence de Larmor et la précession de la magnétisation surviendra à cette fréquence. Ce signal apparaît au point de départ de la figure ci-dessus.

Si le seul facteur se répercutant sur le déphasage était l'interaction spin-spin, l'amplitude de l'enveloppe du signal diminuerait avec une constante de temps  $T_2$ . Néanmoins, un autre facteur - le manque d'uniformité du champ principal - cause un déphasage additionnel. Il en résulte que l'enveloppe du signal diminue selon une constante de temps inférieure à  $T_2$  et dénommée  $T_2^*$ . On régénère le signal (à l'aide d'une *impulsion à 180°*, voir pourquoi plus loin) pour déterminer la valeur  $T_2$  réelle. Le signal régénéré ou écho est représenté sur la figure ci-dessous.

Dans les solides,  $T_2$  est très bref (micro-secondes), étant donné que les solides sont composés de molécules fixes qui conservent les variations de champ locales, ce qui entraîne une rapide perte de cohérence. La magnétisation transversale est conservée plus longtemps dans les liquides (secondes). Dans les tissus biologiques, les valeurs de  $T_2$  vont de 40 millisecondes environ jusqu'à une seconde approximativement. Dans la pratique, la relaxation de la composante transversale ne suit pas une simple courbe exponentielle, mais présente un comportement multi-exponentiel. Toutes mesures effectuées *in vivo* sur  $T_2$  et  $T_1$  produisent des valeurs qui se rapportent à la valeur moyenne des tissus proprement dits.



### Signal FID de décroissance à induction libre (Free Induction Decay)

Ce signal est celui que capte l'antenne à la suite d'une *impulsion à 90°*. Notons que l'*impulsion à 180°*, elle, ne produit aucun signal étant donné qu'elle ne cause aucune composante transversale (oscillant à la fréquence de Larmor).

Le signal FID possède les caractéristiques suivantes :

- il oscille avec les composantes aux fréquences de Larmor des noyaux excités
- il possède une amplitude initiale relative à la densité des noyaux détectés au point de mesure
- son amplitude diminue dans le temps, surtout en fonction du processus de relaxation transversale

Pour diverses raisons, les pointes du spectre réel sont noyées dans le bruit. Il faut donc faire appel à une technique de calcul de moyenne des signaux (que l'on dénomme, en spectroscopie RM, cumul des spectres) pour améliorer le rapport signal/bruit (S/B). En présence de bruit aléatoire, l'addition de spectres améliore le rapport signal/bruit. Le signal augmente linéairement tandis que le bruit, étant aléatoire, dépend d'un facteur de la racine carrée de n, dans laquelle n est le nombre de spectres additionnés. Par exemple, l'addition de spectres obtenus à partir du même échantillon et sous des conditions analogues par 16 mesures distinctes, quadruple le rapport signal/bruit par rapport à une mesure unique.

Il faut également reprendre les mesures dans différentes conditions de champ magnétique, afin de produire des informations d'image. Dès lors, l'acquisition de données en RM risque d'être longue, sauf si l'on prend des mesures spéciales pour l'abréger et donc, en pratique, on peut ne répéter les mesures que très peu de fois. Néanmoins, en additionnant quatre signaux FID seulement, on double le rapport signal/bruit.

### Mesure de la magnétisation résultante

Le phénomène de la résonance magnétique a été observé avec des ondes continues aussi bien qu'avec des impulsions. Cette seconde méthode convient mieux à l'imagerie, et c'est la seule que nous allons étudier ici. Pour exciter les noyaux afin de leur donner davantage d'énergie, l'énergie électromagnétique RF est fournie sous forme d'une séquence d'impulsions d'une intensité et d'une durée données, par une antenne émettrice. Le signal RM observé varie en fonction de la séquence d'impulsions utilisée; on emploie des séquences différentes pour faire varier l'influence des caractéristiques de relaxation T1 et T2 des tissus sur le contraste des images produites par le signal. On peut observer la réponse RM en étudiant le signal FID produit par les méthodes de *saturation-récupération* et d'*inversion-récupération*, ou par la méthode de l'*écho de spin*.

Il est évident, d'après ce qui précède, qu'on peut générer des signaux de réponse RM par diverses méthodes, et que la réponse FID n'en constitue qu'un exemple. Bien que cette caractéristique constitue l'un des grands atouts de l'imagerie RM, étant donné que les différents types d'images correspondent potentiellement du moins à diverses informations de diagnostic, elle constitue par contre une difficulté lorsqu'on tente d'expliquer les différences existant entre des images du même organe, obtenues avec diverses séquences d'impulsions.

### **Séquence saturation-récupération**

Toutes les formes de signaux RM non traités dépendent en priorité de la densité nucléaire, Lorsqu'il n'y a pas de protons dans le plan imagé, il ne peut pas y avoir de signal, La méthode de mesure décrite ici peut s'utiliser pour produire des images basées en majeure partie sur la densité des protons.

L'application d'une *impulsion 90°* unique et la détection du signal FID en résultant est dénommée la méthode de *saturation-récupération*. Cette séquence est également connue sous le nom de technique FID répétée, lorsqu'on effectue plusieurs mesures avec impulsion 90°.

Si l'on répète cette séquence (aux fins de calcul de la moyenne des signaux ou pour des besoins d'imagerie), et que l'intervalle entre les impulsions 90° soit suffisant pour que le vecteur de magnétisation se réaligne totalement à partir du plan x-y jusqu'à la direction z, l'amplitude initiale du signal détecté ne présente aucune dépendance significative par rapport à T1 ou T2. Dans ce cas, le signal dépend en majeure partie de la densité des noyaux participants.

Il faut prévoir un intervalle de plusieurs secondes entre les impulsions 90° en ce qui concerne les tissus car pour éviter toute subordination à T1, il faut utiliser une durée de répétition bien supérieure à T1.

En effet, si l'on utilise un bref intervalle, les tissus ou fluides possédant une valeur T1 supérieure à la moyenne, produisent un signal faible et apportent peu à l'image. Par conséquent, plus T1 est grand, plus le "contraste sombre" l'est également. En ce cas, par exemple, le fluide cérébrospinal dont T1 est long semble sombre dans l'image obtenue par saturation-récupération, tandis que les tissus possédant un T1 court produisent des signaux puissants.

Néanmoins, d'autres tissus peuvent apparaître sombres dans le même type d'image, mais pour des raisons différentes. Par exemple, les tissus osseux du cortex donnent un signal faible parce qu'ils contiennent peu d'hydrogène.

Il existe une autre exception. Le sang présente un T1 prolongé et on pourrait s'attendre à ce qu'il produise un signal faible. Néanmoins, le sang circulant dans la région d'intérêt n'aura pas été excité. Il se trouvera en fait dans l'état de récupération et produira un signal plus puissant que celui auquel on pourrait s'attendre durant un bref intervalle. Par conséquent, le sang peut produire une zone plus lumineuse que prévue, selon la direction et l'importance du débit.

Dans la méthode de saturation-récupération (SR), et surtout lorsque la durée de répétition est prolongée, le contraste de l'image n'est pas rehaussé sur la base des propriétés de relaxation des tissus. Par conséquent, une séquence de saturation-récupération produit des images dont le contraste est inférieur à celui qu'on peut obtenir à l'aide des techniques de l'écho de spin ou de l'inversion-récupération, décrites ci-après, et qui mettent à profit les variations spéciales des valeurs T1 ou T2.

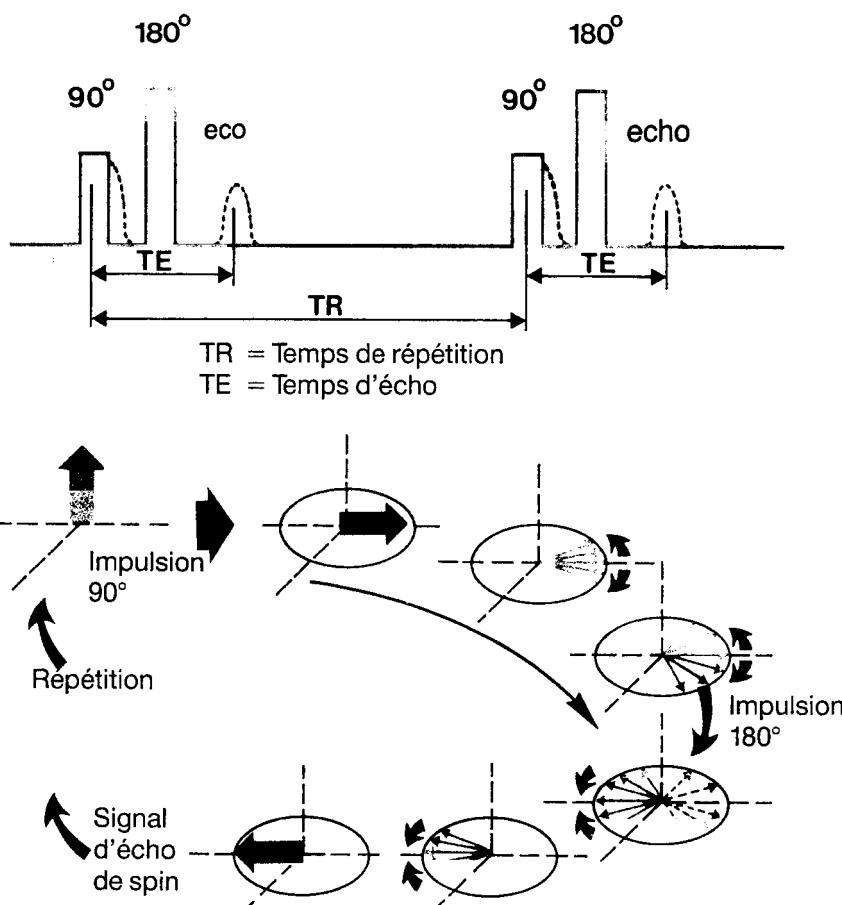
## Séquence de l'écho de spin

La magnétisation transversale diminue du fait d'interactions spin-spin à une vitesse caractérisée par la période de relaxation  $T_2$ . Néanmoins, comme mentionné dans le chapitre sur la relaxation transversale, le signal de réponse FID diminue en réalité plus rapidement du fait des non-homogénéités du champ magnétique principal.

Si la vitesse de réduction du signal FID suite à une seule impulsion  $90^\circ$  est mesurée, c'est  $T_2^*$  que l'on mesure et non  $T_2$ . Les interactions spin-spin et les inhomogénéités du champ principal causent une précession des noyaux à des fréquences légèrement différentes les unes des autres. Ce déphasage réduit encore le signal caractérisé par la période de relaxation  $T_2^*$ .

Pour mesurer la période de relaxation spin-spin  $T_2$ , il faut éliminer l'effet du manque d'homogénéité du champ. Ceci s'effectue aisément, étant donné que le déphasage dû à l'interaction spin-spin est un phénomène aléatoire, tandis que *le déphasage dû à l'inhomogénéité du champ est systématique*.

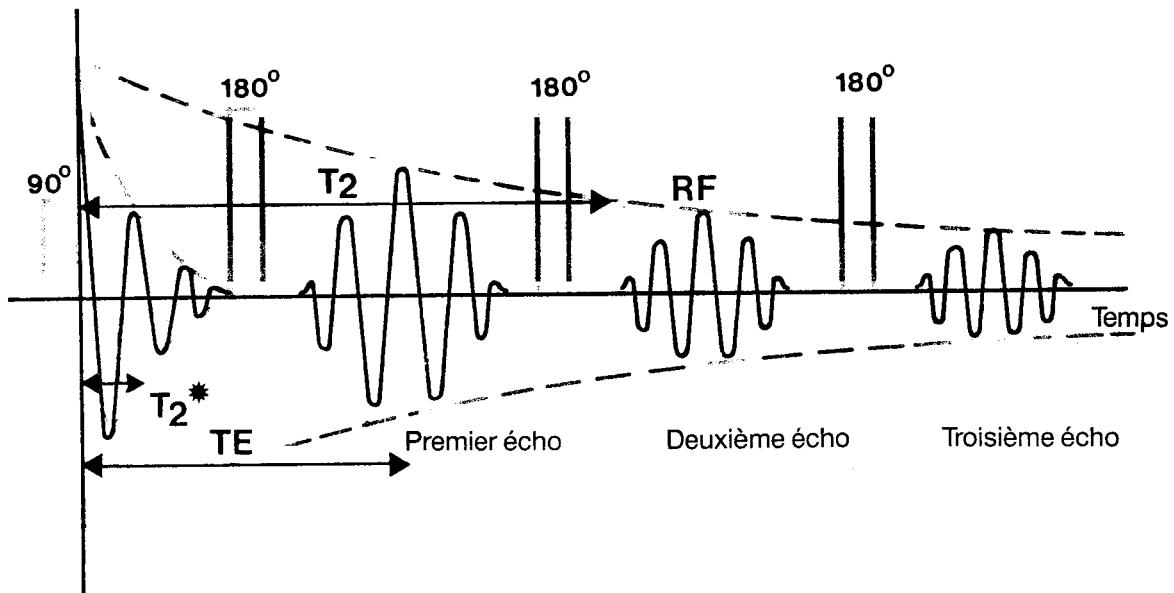
Considérons des coureurs sur une piste circulaire, par un jour de grand vent. Peu après le début de la course, les coureurs les plus rapides seront devant les plus lents. Leurs aptitudes différentes, ainsi que les rafales de vent, répartissent les coureurs sur la piste. Si, sur un ordre lancé un certain temps après le début de la course, les coureurs doivent se retourner en conservant leur propre vitesse mais avec les variations dues au vent, ils reviendront tous à leur point de départ à peu près au même moment. Ils n'auront été séparés que par les effets aléatoires du vent, étant donné que les coureurs les plus lents parcourront une distance inférieure à celle que devront parcourir les coureurs les plus rapides, puisqu'ils auront tout d'abord couru moins loin.



Les effets systématiques dus à l'inhomogénéité du champ et qui causent le déphasage peuvent être inversés par application d'une *impulsion  $180^\circ$*  un certain temps après l'impulsion  $90^\circ$  initiale. Ceci fait pivoter les moments magnétiques individuels sur  $180^\circ$  autour de l'axe  $x$ , comme représenté sur la figure ci-dessus, ce qui produit une remise en phase et une régénération ou écho du signal. La séquence d'impulsions décrite est dénommée *méthode d'écho de spin* (ES).

Cet écho atteint une pointe dont l'amplitude est inférieure à celle du signal initial, étant donné que les effets de relaxation aléatoire dus à l'interaction spin-spin ne seront pas inversés. Pour finir, l'écho commencera à s'affaiblir pour les mêmes raisons ayant entraîné l'affaiblissement du signal FID d'origine. Néanmoins, on peut utiliser une autre impulsion  $180^\circ$  pour générer un

second écho. Ce processus peut être repris plusieurs fois. L'enveloppe globale du signal s'affaiblit à une vitesse qui reflète la valeur  $T_2$  "réelle". L'intervalle entre l'impulsion  $90^\circ$  et la pointe du premier écho est dénommé la période d'écho de spin TE (voir figure ci-dessous). En répétant l'impulsion  $180^\circ$  à des intervalles égaux à TE, on produit des échos à intervalles analogues, mais dont les amplitudes de pointe décroissent sans cesse.



Cette technique peut s'utiliser pour rehausser le contraste des images en mettant à profit la propriété  $T_2$  du tissu examiné. Considérons par exemple une image obtenue par échantillonnage du signal de réponse au niveau du troisième écho. Le signal provenant de zones du corps à  $T_2$  court produira un signal plus petit que les zones à  $T_2$  long. Par conséquent, plus  $T_2$  est court, plus l'image est sombre. Les tissus possédant de longs  $T_2$  produisent un "contraste brillant". Ceci est à l'opposé de la méthode SR dans laquelle les tissus à longue période de relaxation  $T_1$  affaiblissent le signal et produisent donc un "contraste sombre".

### Séquence d'inversion-récupération

Pour produire un signal RM influencé par  $T_1$ , on crée une *impulsion à  $180^\circ$* , donc, comme déjà mentionné, on introduit un champ électromagnétique RF perpendiculairement à la direction du champ principal, avec une amplitude et une durée suffisantes pour réorienter le vecteur de magnétisation résultante à  $180^\circ$  par rapport à la direction initiale, c'est-à-dire pour inverser la magnétisation. Les noyaux dont la fréquence de Larmor est égale à la fréquence du basculement de l'impulsion sur  $180^\circ$  et le vecteur de magnétisation résultante  $M$  sont inversés.

Lorsque l'impulsion RF cesse, les noyaux ne sont plus soumis à un champ magnétique en direction  $x$ , et commencent à se réorienter vers le sens du champ statique principal. Pourvu qu'il ne soit pas perturbé, ce processus de relaxation se poursuivra jusqu'à rétablissement de l'équilibre, point auquel la magnétisation résultante est revenue dans sa direction initiale. Durant la relaxation, la magnétisation résultante  $M$  doit demeurer dans la direction  $z$  (composante  $M_z$  uniquement), étant donné que les forces du champ magnétique RF ne sont plus présentes, et n'agissent donc plus sur l'orientation.

On fait basculer de  $90^\circ$  la résultante  $M$  à l'aide d'une *impulsion à  $90^\circ$*  (à la fréquence de Larmor), ce qui crée une composante  $M_{xy}$  proportionnelle à la composante  $M_z$  qui existait à l'instant précédent. Le rayonnement de  $M_{xy}$  peut être perçu par l'antenne à la fréquence de Larmor juste après application de l'impulsion à  $90^\circ$ . Il est dû aux tissus dont le temps de relaxation  $T_1$  est suffisamment court pour qu'ils soient revenus à une composante  $M_z$  suffisante.

A l'inverse, ceux dont  $M_z$  était encore fort proche de 0 (T1 long) au moment de l'impulsion à  $90^\circ$  ne produiront qu'une faible composante  $M_{xy}$  et donc une image sombre. Cette méthode permet donc de distinguer les tissus en fonction de leur temps de relaxation T1.

Il est également possible d'appliquer à nouveau une impulsion à  $180^\circ$  après l'impulsion à  $90^\circ$ , mais alors pour lui faire jouer un rôle similaire aux impulsions  $180^\circ$  de la séquence d'écho de spin.

### **Codage spatial du signal RM**

Nous allons décrire une méthode permettant de produire une image unidimensionnelle d'un tube long et fin contenant plusieurs types de tissus, dont chacun possède une densité protonique différente.

Si l'échantillon est placé dans un champ magnétique uniforme B, tous les protons auront la même fréquence de Larmor nominale, dont la valeur se calcule à l'aide de l'équation de Larmor. Une fois l'échantillon excité par une impulsion en radiofréquence à la fréquence de Larmor, et le signal d'affaiblissement à induction libre (FID) enregistré, ce signal comportera une seule composante, c'est-à-dire la fréquence de Larmor. Par contre, si l'on place l'échantillon dans un champ magnétique dont la puissance augmente dans le sens de la longueur du tube, les fréquences de Larmor des protons seront relatives à leur position sur le même axe. *Si le gradient du champ est linéaire et augmente en fonction de la distance x le long de l'échantillon, la fréquence de Larmor de chaque proton sera en rapport direct avec la position spatiale du proton.* Nous soulignons que le sens du gradient n'est pas nécessairement le même que celui du champ magnétique appliqué pour le produire. On convient de dénommer la direction du champ principal la direction z. Le champ du gradient est toujours aligné dans la direction z, mais le gradient lui-même peut s'orienter dans les directions x, y ou z.

Si nous excitons ensuite l'échantillon avec une *impulsion RF à bande large*, c'est-à-dire avec une énergie électromagnétique dont la gamme de fréquences se situe entre la valeur la plus basse et la plus élevée des fréquences de Larmor existant au sein de l'échantillon, le signal FID n'aura plus une fréquence unique, mais possédera des composantes dont les fréquences se situent dans la même gamme. De plus, l'intensité de chacune de ces composantes sera relative au nombre de protons qui contribuent à la produire.

Si l'on décompose le signal de réponse RM en ses diverses fréquences à l'aide d'une transformée de Fourier, on obtient un spectre dans lequel la fréquence de la composante fournit des informations spatiales et l'intensité relative indique la densité protonique.

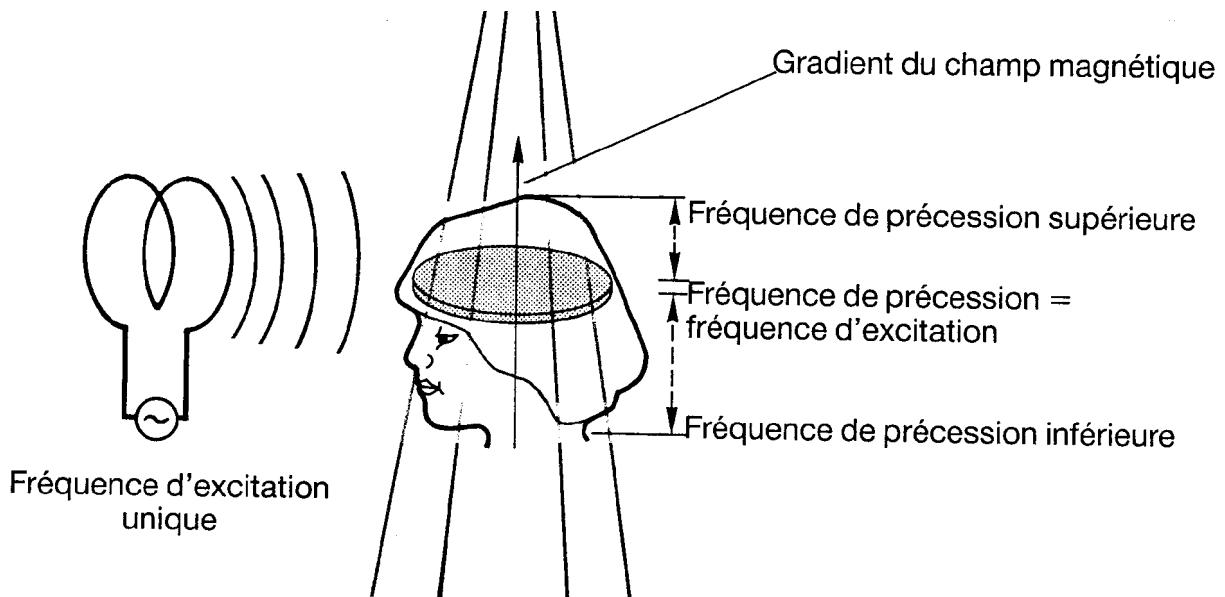
### **Méthodes par plans ou sélection de coupes**

Dans ces méthodes, l'information est recueillie simultanément à partir d'une coupe complète du corps. C'est une méthode beaucoup plus efficace que les techniques par points et par lignes, mais qui est plus exigeante au point de vue des calculs.

Deux méthodes par plans se sont avérées les plus utiles:

- Méthode de rétro-projection (codage: fréquence-angle)
- Méthode d'imagerie bidimensionnelle de Fourier (codage: fréquence-phase).

La différence entre les deux méthodes est la manière dont l'information spatiale provenant du plan est codée dans le signal de réponse. Avant d'étudier ces méthodes et leurs mérites relatifs de manière plus détaillée, il faut décrire les techniques d'excitation limitant le signal à la couche d'intérêt, étant donné que ceci constitue la première opération essentielle, et que celle-ci est commune aux deux techniques. Les deux principales méthodes d'excitation qui ont été appliquées en imagerie des plans, sont *l'excitation sélective* et *le gradient oscillant*. C'est la première méthode qui s'est avérée la plus satisfaisante.



La méthode d'excitation sélective fait appel à un gradient de champ magnétique relativement faible, qui vient s'ajouter au champ principal dans la direction z. Il en résulte que la fréquence propre des noyaux augmente le long de cet axe. La couche ou coupe à sélectionner, qui doit posséder une épaisseur finie, contient donc une bande étroite de fréquences de Larmor. En excitant les spins avec une radiation électromagnétique située dans la même étroite bande ou fréquence "unique", au moyen d'une impulsion RF de forme spéciale, on limite l'excitation à une coupe particulière. Cette forme d'excitation sélective d'une coupe unique est illustrée sur la figure ci-dessus. Pour une impulsion d'excitation donnée, l'épaisseur de coupe dépend de l'importance du gradient, que l'on peut modifier aisément pour faire varier cette épaisseur. Dans les systèmes RM pratiques, la fréquence d'impulsions est fixe alors que l'on peut faire varier le champ magnétique pour sélectionner des coupes différentes.

En variante, on peut faire appel à un *gradient magnétique oscillant* ou alternatif. En ce cas, les fréquences de Larmor de tous les noyaux de l'objet, sauf ceux qui se trouvent dans une coupe de part et d'autre de laquelle le gradient alterne, fluctueront continuellement.

On produit alors l'excitation en émettant une série d'impulsions RF extrêmement brèves, dans un spectre de fréquences relativement étendu. En ce cas, les noyaux du corps entier émettront des signaux mais seuls ceux qui se trouvent dans la coupe autour de laquelle le gradient alterne émettront des signaux à une fréquence constante. Partout ailleurs, les fréquences varieront et s'annuleront mutuellement.

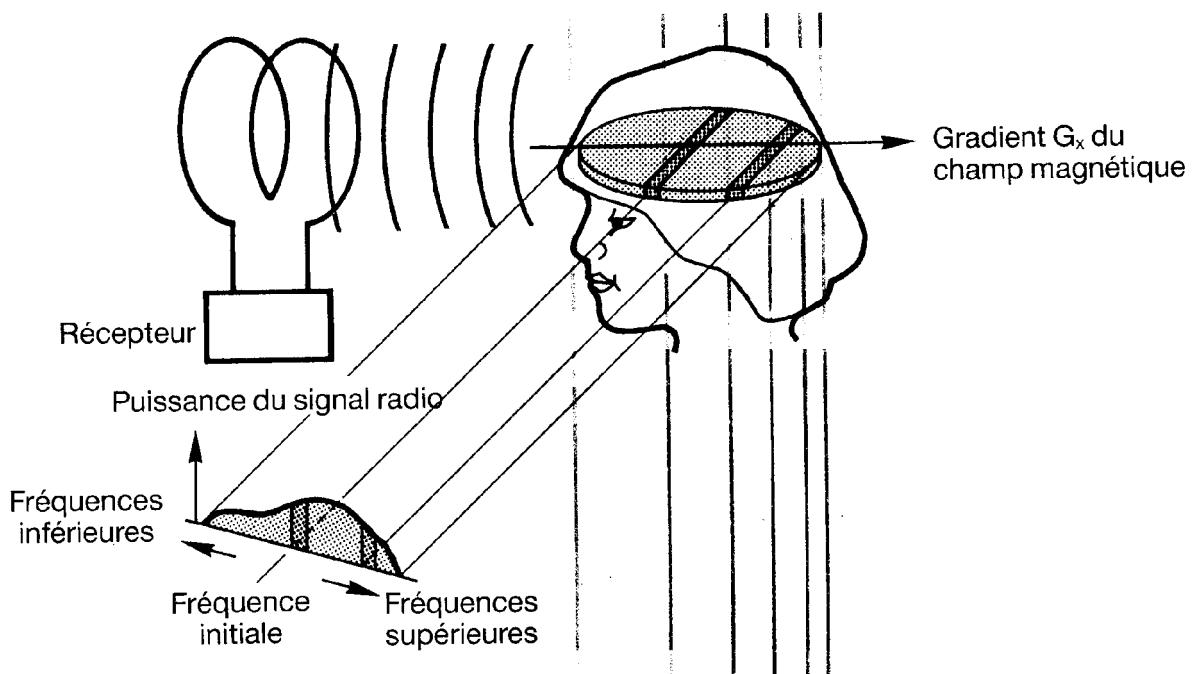
Il s'est avéré plus aisé d'utiliser la technique d'excitation sélective. Si le gradient sélectif est appliqué dans la direction x ou y plutôt que dans la direction z, on obtiendra des images *sagittales* ou *coronales* au lieu d'images *transaxiales*.

En excitation sélective, l'impulsion d'excitation RF est appliquée en même temps que le gradient de sélection de coupe. Pour produire une image de la coupe (supposée perpendiculaire à la direction z), il faut obtenir des informations dans les directions x et y. Pour ce faire, on applique d'autres séquences de gradient. On a utilisé deux méthodes d'imagerie en plans: la méthode par rétro-projection et la méthode d'imagerie Fourier bidimensionnelle.

### Méthode par rétro-projection

Dans la méthode par projection-reconstruction, la coupe ou couche est sélectionnée comme décrit au point précédent. Pour produire un codage spatial, on applique ensuite un second gradient magnétique  $G_x$  perpendiculairement à la direction du gradient de sélection de coupe. Ce cas est représenté sur la figure suivante, sur laquelle le second gradient est orienté dans la direction x.

Du fait du gradient qui traverse le plan en direction x, la fréquence du signal de réponse est fonction de la position en x. Seuls les noyaux présents sur l'axe (où l'induction  $B_0$  est restée la même que lors de l'application de l'impulsion à  $90^\circ$ ) poursuivront leur précession à la fréquence originale. La transformée de Fourier du signal détecté est une projection de l'objet sur l'axe x. Elle fournit un profil unidimensionnel de la densité nucléaire au sein de l'objet. L'amplitude de la composante correspondant à chaque fréquence est en rapport avec le nombre de noyaux, en direction y, qui contribuent à produire cette composante.



En appliquant le gradient à diverses orientations, on peut obtenir des informations spatiales provenant de plus d'une direction, puis reconstruire une image du plan à partir de ces informations. Dans la pratique, on peut appliquer deux gradients statiques pour modifier l'orientation du profil. On peut modifier les valeurs relatives de ces deux gradients afin de faire varier l'orientation du gradient résultant à l'intérieur du plan. En faisant tourner le gradient, puis en réappliquant les impulsions d'excitation, on peut obtenir de nombreux spectres semblables à celui que nous avons représenté, dont chacun correspond à une orientation différente. La durée d'acquisition de données dépendra, entre autres, de la séquence d'impulsions employée et de la dimension de matrice retenue (c'est-à-dire du nombre de pixels ou points d'image), mais cette acquisition ne prendra néanmoins que quelques minutes (on utilise en pratique certaines techniques pour réduire la durée réelle d'acquisition des données).

La méthode par rétro-projection est relativement simple et fait appel à des algorithmes de reconstruction d'image éprouvés. Néanmoins, elle est extrêmement sensible aux artefacts et à l'inhomogénéité des champs. C'est pourquoi on a dû mettre au point d'autres méthodes, dont la méthode bidimensionnelle de Fourier.

## La méthode bidimensionnelle de Fourier

Dans la méthode bidimensionnelle de Fourier (2DF), on procède tout d'abord à une excitation sélective de la couche ou coupe. On applique ensuite un gradient de détection de champ magnétique  $G_x$  dans la direction  $x$  pour définir un profil de projection linéaire. Ces deux opérations sont analogues à la procédure initiale de la technique de rétro-projection. Cependant, au lieu de réaliser des combinaisons de gradient suivant  $G_x$  et  $G_y$  pour obtenir des gradients résultants de directions variées, le gradient  $G_x$  demeure fixe et est précédé d'un gradient de préparation à codage de phase  $G_y$  appliqué brièvement perpendiculairement à  $G_x$ .

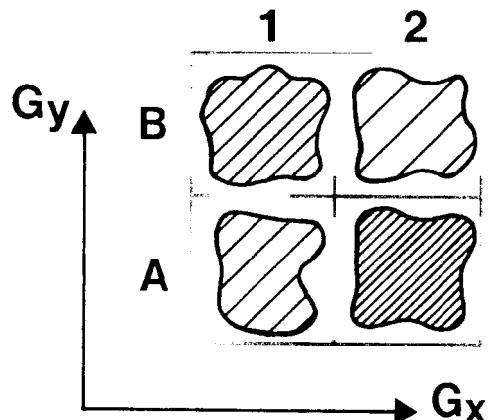
Lorsque l'on applique le gradient de sélection de coupe  $G_z$ , suivi de l'impulsion d'excitation RF nécessaire, tous les noyaux du plan amorcent leur précession en phase l'un par rapport à l'autre, à la même fréquence nominale. Durant la période du gradient  $G_y$ , les fréquences propres des noyaux seront modifiées en fonction de leur position dans la direction  $y$ . Autrement dit, la fréquence de précession augmentera dans la direction  $y$ .

Si l'on cesse d'appliquer le gradient  $G_y$  et que l'on applique le gradient  $G_x$ , les noyaux situés le long de la direction  $y$  présenteront à nouveau une précession à une fréquence identique, mais les phases des diverses précessions auront changé proportionnellement à la force locale du champ imposée durant la période du gradient de préparation  $G_y$ . La fréquence de précession ne varie alors qu'en sens  $x$ , mais *il y aura variation de phases en sens  $y$* .

Pour comprendre le principe de l'imagerie 2DF, considérons un objet simple composé de quatre éléments, d'après lequel il faut produire une image possédant quatre pixels ou points. La figure ci-dessous illustre ce cas.

En supposant que l'image doit être basée sur la densité protonique, il faut déterminer des valeurs pour les points A1, A2, B1 et B2. Étant donné qu'il existe quatre inconnues, quatre équations au moins sont requises. Si, dans un premier temps, on effectue une mesure en n'appliquant que le gradient  $G_x$  après l'impulsion d'excitation (c'est-à-dire sans gradient  $G_y$ ), la composante signal provenant de l'élément A, sera à la même fréquence et à la même phase que la composante provenant de B. Par conséquent, ces deux composantes s'additionnent simplement et on peut trouver le résultat à partir du spectre de fréquences du signal de réponse. De même, les composantes des éléments A2 et B2 seront en phase et à une fréquence identique, bien que cette fréquence diffère de celle des composantes de la première colonne. On peut donc trouver leur somme à la fréquence appropriée du spectre. Jusqu'ici, nous avons trouvé les sommes correspondant à deux équations, à savoir :

$$\begin{aligned} S_1 &= A_1 + B_1 \\ S_2 &= A_2 + B_2 \end{aligned}$$



On réalise ensuite une seconde mesure mais, cette fois, en faisant précéder  $G_x$  d'un gradient de codage en phase  $G_y$ . Durant la période de  $G_y$ , la précession des noyaux de la rangée B sera plus rapide que celle des noyaux de la rangée A. Supposons que  $G_y$  cesse d'être appliqué au moment où le déphasage entre les deux rangées égale  $180^\circ$ . Après application de  $G_x$ , il existera à nouveau un déphasage entre les colonnes 1 et 2, mais il existera également alors un déphasage entre les rangées A et B. Le spectre contient maintenant les valeurs de deux autres équations:

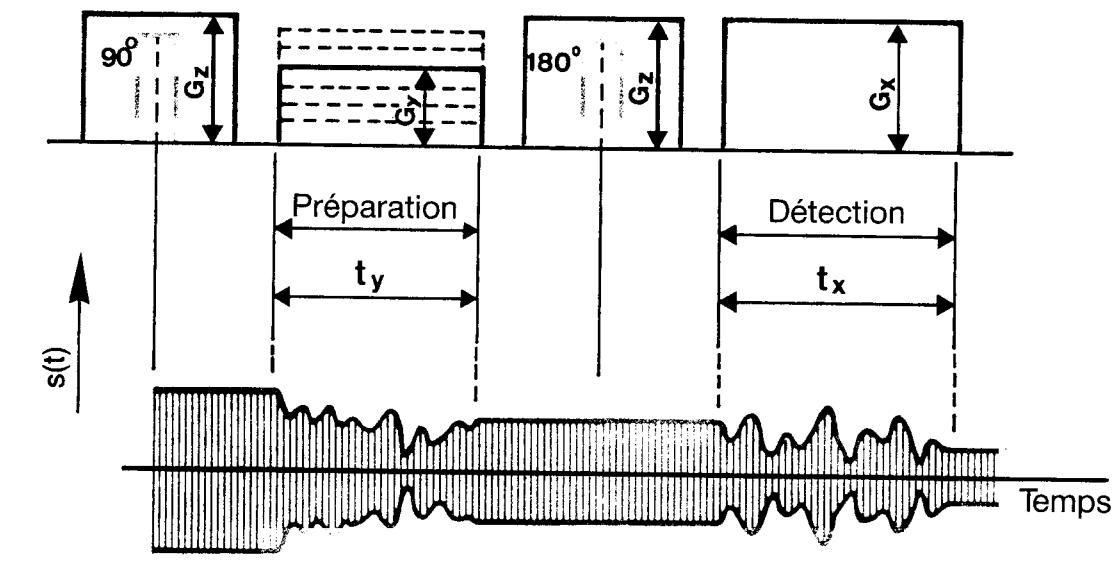
$$\begin{aligned} S_3 &= A_1 - B_1 \\ S_4 &= A_2 - B_2 \end{aligned}$$

Ainsi, s'il s'agit d'une image de quatre pixels, il faut prendre deux mesures avec une puissance

différente pour Gy. Chacune fournit un signal qui, après transformation de Fourier produit des valeurs pour les équations de deux colonnes. On obtient donc en tout les valeurs correspondant aux quatre équations.

Dans le cas général d'une image de  $N \times N$  points, il faut exécuter la mesure  $N$  fois, en modifiant le degré de codage en phase pour chaque mesure. Chacune des mesures ainsi obtenues fournit la valeur d'une équation pour chaque colonne, c'est-à-dire  $N$  équations. On obtient ainsi  $N \times N$  équations pour  $N \times N$  inconnues.

Le courant induit dans l'antenne de détection constitue un signal de réponse qui regroupe les signaux de tous les noyaux dont la précession a lieu soit à des fréquences différentes, soit à des phases diverses. La composante correspondant à une fréquence particulière est la somme de toutes les contributions vectorielles provenant d'une rangée de noyaux en direction y. Cette somme vectorielle varie en fonction du degré de codage en phase (figure ci-dessous). Ceci revient à dire qu'elle varie en fonction du gradient Gy. Ce processus se répète en accroissant le codage en phase, par augmentation de la puissance de Gy, jusqu'à ce que l'on dispose de données suffisantes pour permettre de reconstruire une image.



En rétro-projection, les transformations de Fourier fournissent des informations unidimensionnelles, et l'on fait pivoter le gradient de codage en fréquence pour recueillir des informations en quantité suffisante pour permettre de calculer une image du plan. En imagerie 2DF, le processus de mesure est repris avec le gradient de codage en fréquence dans la même direction, mais en augmentant le degré du gradient de codage en phase jusqu'à acquisition d'une quantité d'informations suffisante pour produire une image. Pour chaque valeur du gradient de codage en phase Gy, les spins des colonnes de la direction y sont déphasés durant la période de préparation et se recombinent de manière différente. Ainsi, à chaque fréquence du profil dans la direction x, on peut déterminer la distribution nucléaire le long de la colonne dans la direction y, étant donné que chaque projection est codée en fréquence dans la direction x, et codée en phase dans la direction y. Les valeurs sont récupérées pixel par pixel dans la transformation de Fourier des projections individuelles, avec calcul basé sur l'ensemble des projections. Aucune technique de reconstruction d'images spéciale n'est requise.

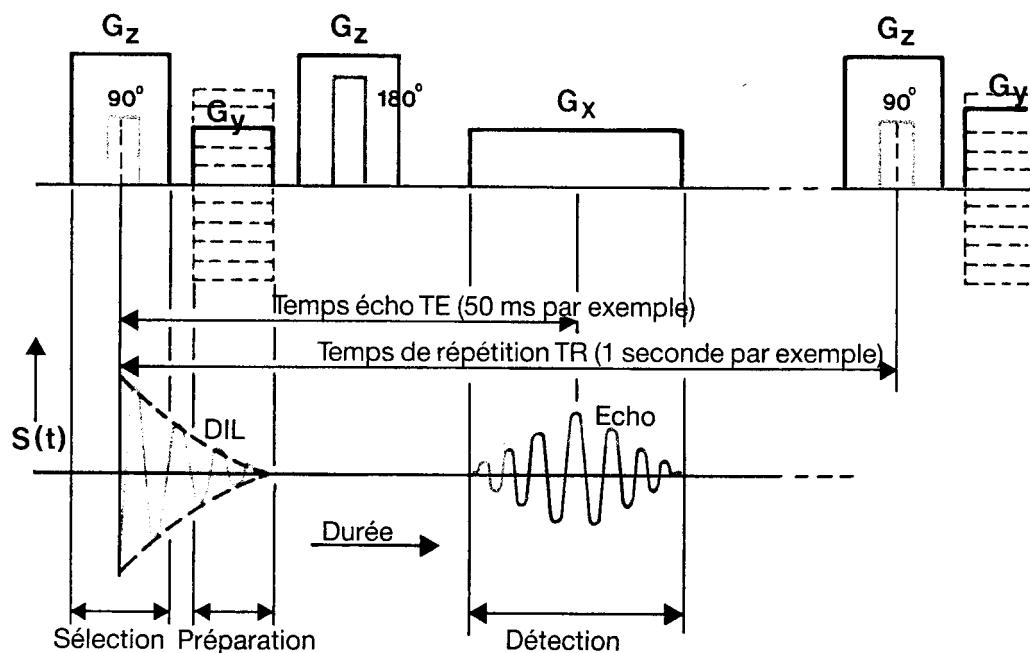
La méthode du Spin Warp est une variante de la méthode Fourier bidimensionnelle à codage en phase que nous venons de décrire. La spirale de phase est créée de manière analogue, mais on modifie le degré de codage en phase en changeant la puissance de l'impulsion du gradient de préparation de codage en phase. Cette caractéristique permet au système

d'imagerie de tolérer les inhomogénéités du champ magnétique statique et les mouvements irréguliers.

### Séquences caractéristiques d'impulsions et de gradients

Il faut maintenant regrouper les nombreux concepts et techniques discutés plus tôt étant donné que, dans un système d'imagerie RM pratique, elles sont toutes combinées en une seule opération d'iconographie. Les manières dont les signaux sont pondérés par les caractéristiques de relaxation des tissus et dont on obtient le codage spatial ont été auparavant décrites séparément. Les premières dépendent de séquences d'impulsions et les secondes des séquences de gradients. De plus, en choisissant judicieusement les impulsions RF et les gradients, on sélectionne la coupe ou couche appropriée. Nous allons maintenant examiner deux combinaisons typiques de séquences d'impulsions et de gradients.

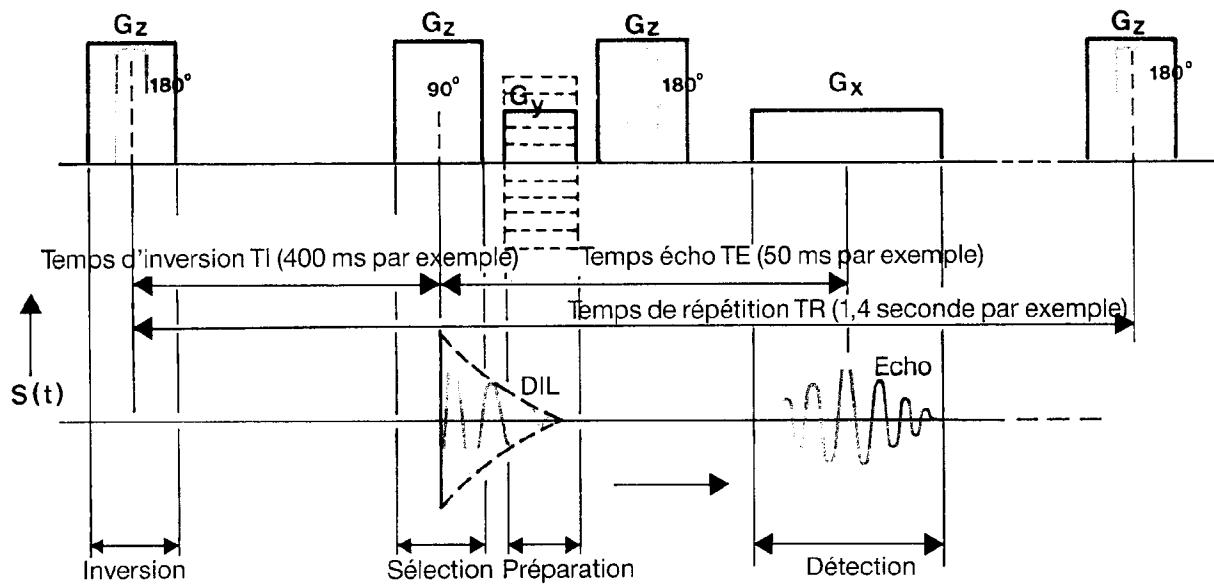
La figure suivante représente une séquence *spin-écho 2DF*. Une impulsion RF de  $90^\circ$ , émise durant l'application d'un gradient  $G_z$ , sélectionne une couche ou coupe. Le signal FID apparaît, mais s'affaiblit rapidement du fait des effets combinés de la relaxation transversale et du déphasage par non-homogénéité du champ magnétique. Le gradient suivant,  $G_y$ , produit le codage de phase correspondant à une dimension spatiale. On applique ensuite l'impulsion à  $180^\circ$  pour produire un signal écho. Durant la remise en phase, un gradient  $G_x$  est appliqué pour coder le signal en fréquence pour la seconde dimension. Le signal écho est détecté, puis décomposé. Ce jeu d'impulsions et de gradients est répété de nombreuses fois, avec une cadence de répétition TR entre jeux; chaque jeu possède un  $G_y$  de valeur légèrement différente, et l'opération se poursuit jusqu'à ce que l'on ait acquis suffisamment de données pour construire une image.



On peut reprendre cette séquence avant de changer  $G_y$ , aux fins de calcul de moyenne des signaux. En faisant varier le temps de l'impulsion à  $180^\circ$  et, dès lors, le temps écho TE, on fait varier le degré de mise en valeur T2 de l'image. Précisons que l'impulsion  $180^\circ$  ne supprime pas le codage en phase spatiale. Elle ne sert qu'à éliminer le déphasage indésirable dû aux imperfections du champ magnétique principal.

Le second exemple, illustré ci-dessous, est une séquence d'*imagerie 2DF par inversion-récupération*. La première impulsion RF à  $180^\circ$ , appliquée en même temps que le gradient de sélection de coupe  $G_z$ , inverse la magnétisation. Après le temps d'inversion TI, durant lequel survient un certain degré de relaxation longitudinale, on applique une impulsion à  $90^\circ$  pour produire le signal FID, dont l'amplitude initiale est en rapport avec la densité protonique et la relaxation T, qui survient dans la période TI. Jusqu'ici, le train d'impulsions constitue une

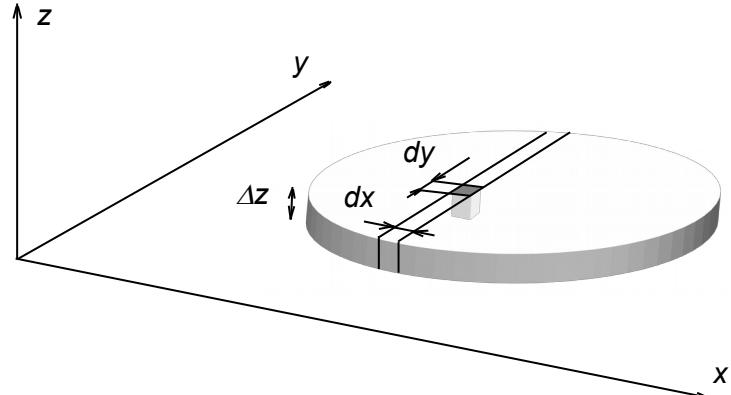
séquence d'inversion-récupération. On applique alors la seconde impulsion de remise en phase  $180^\circ$  pour produire un signal écho. Ceci est suivi des gradients de codage spatial (comme dans le premier exemple) qui permettent d'identifier le signal de réponse d'après l'origine.



### Expression mathématique de l'intervention de la 2DF

1. Appelons  $\bar{m}_{xy}(x, y, t) \cdot dx \cdot dy$  le moment magnétique perpendiculaire résultant d'un élément de la tranche excitée d'épaisseur  $\Delta z$ , de largeur  $dx$  et de longueur  $dy$ .

Considérons que ce moment magnétique tourne à la fréquence de Larmor dans le plan  $x, y$  et nous pouvons séparer l'amplitude et la phase :



$$\bar{m}_{xy}(x, y, t) = m_0(x, y) \cdot e^{-j\omega t}$$

avec, par la formule de Larmor

$$\omega = \gamma \cdot B$$

où l'induction magnétique  $B$  est dirigée suivant l'axe  $Z$ . On peut considérer que  $m(x, y)$  est proportionnel à la concentration en spins dans l'élément et donc mesurer  $m(x, y)$  permet de connaître cette concentration en fonction de  $x$  et  $y$ .

En l'absence de gradient  $G_x$  ou  $G_y$  appliqués et en supposant (pour simplifier) que la phase se conserve, le moment magnétique perpendiculaire global (dû à la tranche entière) vaut

$$\bar{M}_{xy}(t) = \int \int \bar{m}_{xy}(x, y, t) \cdot dy \cdot dx = e^{-j\gamma \cdot B \cdot t} \cdot \int \left( \int m_0(x, y) \cdot dy \right) \cdot dx$$

dont on peut simplifier la notation de l'intégrale en  $y$  (qui après intégration en  $y$  n'est plus qu'une fonction de  $x$ ) :

$$\bar{M}_{xy}(t) = e^{-j\gamma \cdot B \cdot t} \cdot \int f(x) \cdot dx \quad \text{vecteur tournant à la pulsation } -\gamma \cdot B$$

2. Par application d'un gradient  $G_x$ ,  $B$  devient  $(1+G_x \cdot x) \cdot B$  (tout en restant toujours parallèle à l'axe  $z$ ) et le moment global devient

$$\bar{M}_{xy}(t) = e^{-j\cdot y \cdot B \cdot t} \cdot \int f(x) \cdot e^{-j \cdot y \cdot B \cdot G_x \cdot t \cdot x} dx$$

dont la partie réelle correspond au signal radio reçu par l'antenne. Ce moment est en quelque sorte une somme infinie de vecteurs tournant à différentes fréquences  $\mathbf{f} = (1+G_x \cdot x) \cdot \gamma \cdot B / 2\pi$ . On constate dans l'équation que la fréquence  $\mathbf{f}$  est une fonction linéaire de la position en  $\mathbf{x}$  (donc  $\mathbf{f}$  est l'image de  $\mathbf{x}$ ). Dès lors le spectre en fréquence du signal reçu (par la transformée de Fourier) exprime la concentration en fonction de  $\mathbf{x}$ .

3. Avant d'appliquer le gradient  $G_x$  qui sera présent durant la réception, appliquons un gradient  $G_y$  durant un temps fixé  $T$ . Ce gradient accroît la vitesse de rotation en fonction de  $\mathbf{y}$  et au terme du temps  $T$  a ajouté un déphasage proportionnel à  $\mathbf{y}$ . On peut donc écrire :

$$\int m(x, y) dy = \int m_0(x, y) e^{-j \cdot y \cdot B \cdot G_y \cdot T \cdot y} dy$$

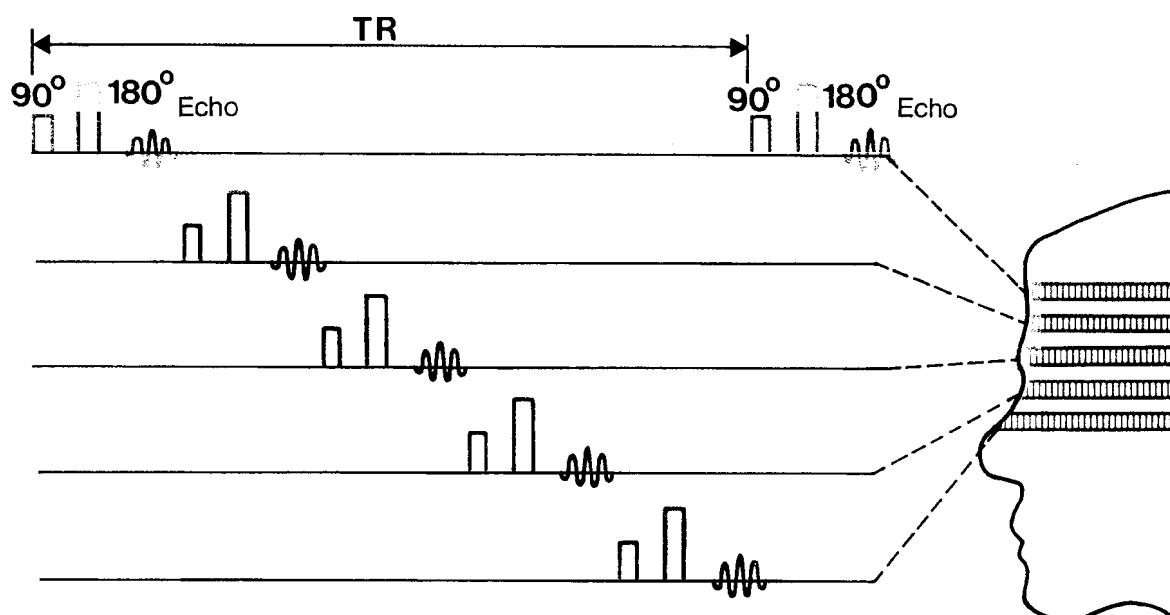
où  $m_0(x, y)$  correspond au moment obtenu pour  $G_y$  nul. Et la formule complète devient

$$\bar{M}_{xy}(G_x \cdot t, G_y \cdot T) = e^{-j \cdot y \cdot B \cdot t} \cdot \int \left( \int m_0(x, y) e^{-j \cdot y \cdot B \cdot G_y \cdot T \cdot y} dy \right) e^{-j \cdot y \cdot B \cdot G_x \cdot t \cdot x} dx$$

où l'on peut disposer de mesures pour différentes valeurs de  $t$  et de  $G_y$ . On constate que le codage en phase que l'on fait jouer à  $\mathbf{y}$  est similaire au codage en fréquence que l'on fait jouer à  $\mathbf{x}$ , si ce n'est que l'un est discret alors que l'autre est continu (mais comme on échantillonne le signal avant calcul, en pratique les deux sont discrets pour le calcul). A ce détail près, le rôle de la transformée de Fourier est le même pour  $\mathbf{x}$  et en  $\mathbf{y}$ . C'est donc par une transformée de Fourier bidimensionnelle sur les signaux radio reçus pour différentes valeurs de  $G_y$  qu'on pourra déduire  $m_0(x, y)$  pour les différentes valeurs de  $\mathbf{x}$  et  $\mathbf{y}$ .

### Techniques multi-coupes

Les mesures réalisées dans un plan donné sont répétées plusieurs fois, notamment pour établir une moyenne des signaux. Néanmoins, une partie du temps de répétition TR comporte une période de récupération qui constitue virtuellement un délai d'attente. On peut améliorer la performance de l'imagerie en plan en utilisant la période d'attente nécessaire à une coupe pour mesurer d'autres coupes. Ce principe est représenté sur la figure ci-dessous.



Les couches ou coupes peuvent être quasi contiguës. Il faut néanmoins séparer quelque peu les coupes mesurées en séquence, afin d'éviter toute confusion entre les signaux correspondant à diverses coupes. Pour obtenir des coupes contiguës, on peut exciter tout d'abord les coupes à numéros impairs, puis les coupes paires. Cette technique produit parfois des surprises. Par exemple: des vaisseaux qui, en règle générale, ne produisent pas de signaux en technique par coupe unique étant donné que le sang excité a quitté la coupe au

moment de la mesure, apparaissent parfois soudain dans l'image acquise par les techniques multi-coupes.

### **Méthodes tridimensionnelles**

La rétro-projection et l'imagerie 2DF peuvent être utilisées pour la troisième dimension nécessaire à l'acquisition d'images tridimensionnelles. Au lieu d'exciter une coupe unique ou de filtrer les signaux provenant d'une seule coupe, on excite un volume complet, puis on lui donne un code spatial tridimensionnel. Dans la méthode tridimensionnelle de Fourier (3DF), le volume total des noyaux est excité par une impulsion RF avant l'application des gradients. On applique ensuite successivement des gradients de codage en phase dans la direction y et dans la direction z (plutôt que dans la direction y uniquement, comme dans la méthode 2DF). Pour finir, on applique le gradient de codage en fréquence (comme pour la méthode 2DF). Ainsi, l'angle de phase cumulé correspondant à un noyau particulier au début de la période de mesure est fonction de la position spatiale qu'il occupe sur les coordonnées y et z.

Pour chacun des degrés de codage en phase z, il faut prendre n mesures avec divers degrés de codage en phase y. S'il est nécessaire de disposer de m degrés de codage en phase z, pour garantir une résolution spatiale adéquate dans la direction z, il faut acquérir un nombre de signaux égal à m x n.

Le signal est en ce cas simultanément acquis à partir de toutes les régions appartenant à un volume important du corps. Étant donné que le signal provient d'un gros volume plutôt que d'un plan de faible épaisseur ou d'une ligne étroite, cette méthode présente, par rapport aux autres, une amélioration du rapport signal/bruit. Néanmoins, dans cette technique, il faut tenir compte de bien d'autres facteurs que les seules inconnues  $N^2$  de la technique monocoupe. C'est pourquoi les besoins en mémorisation et en calculs informatiques sont infiniment supérieurs. L'acquisition des données prend actuellement (en 1985) plus de 15 minutes. Néanmoins, une fois les mesures réalisées, les images peuvent être affichées dans n'importe quelle orientation, sans aucune autre exploration.

### **Puissances des champs magnétiques**

Il est évident que la sélection de la puissance du champ est assez complexe. Le choix dépend largement du type d'application clinique, de la qualité d'images envisagée, et des durées d'acquisition admissibles en fonction du budget alloué et des contraintes d'installation.

On considère qu'une homogénéité du champ de 100 ppm environ est admissible en imagerie protonique. Cependant, s'il s'agit d'imager des isotopes tels que  $^{31}\text{P}$  ou  $^{13}\text{C}$ , cette homogénéité doit être portée à 1 ppm environ pour l'analyse par décalage chimique. Les examens *in vitro* s'effectuent entre 1T et 3T, ce qui permet d'obtenir la résolution spectrale nécessaire. Le diamètre intérieur de l'aimant doit être suffisamment grand pour contenir, non seulement le patient, mais encore les antennes RF et de gradients, la cage de Faraday et, dans certains systèmes, les circuits de refroidissement. L'aimant doit en outre correspondre aux exigences d'homogénéité dans la totalité du champ de mesure.

Le champ magnétique principal peut être produit de trois manières :

- Par un aimant permanent
- Par un électro-aimant à résistance
- Par un électro-aimant supraconducteur

On a produit des systèmes d'imagerie RM équipés de ces trois types d'aimants, et dont certains regroupent plus d'un type. Chacun présente ses avantages et ses inconvénients.

Terminons par une figure montrant l'agencement typique des bobines. Dans cet agencement, les bobines sont en forme de selles. Les antennes d'excitation RF détectrices et superficielles ne sont pas représentées.

