Chapitre 1

TRAITEMENT DE SIGNAL MULTIMEDIA

Introduction

Introduction

- Traitement d'images et vidéos : un nouveau domaine technologique
  - Mathématiques de l'information
  - Traitement du signal
  - Systèmes électroniques et optroniques
  - Architectures machines et micro-processeurs (VLSI, DSP)
- De nombreuses méthodes utilisées
  - Choix de celles qui sont adéquates et bien fondées
  - Besoin d'une méthodologie
- Vaste champ d'applications

On désigne par *traitement d'images numériques* l'ensemble des techniques permettant de modifier une image numérique dans le but de l'améliorer (qualitativement ou quantitativement avec le codage à compression) ou d'en extraire des informations. Le traitement d'images numériques est une discipline nouvelle qui s'est développée rapidement grâce à l'émergence des nouvelles technologies de l'information. Il s'appuie notamment sur les mathématiques liées à l'information, le traitement du signal, les systèmes électroniques, et sur l'avancée des capacités de calcul des microprocesseurs, notamment ceux qui sont développés exclusivement pour le traitement de signal et qui offrent de grandes capacité et vitesse de calculs (DSP, …).

La jeunesse du traitement d'images numériques et le vaste champ d'application qu'il s'est vu attribué ont fortement contribué à la nécessité de créer une méthodologie et de séparer les domaines d'applications. Le traitement d'image s'est ainsi distingué en fonction de quatre domaines d'applications majeurs : l'analyse, la synthèse, le codage et l'amélioration.

Dans un premier temps, la description de ces quatre fonctions principales permettra de mieux appréhender ce qu'est le traitement d'images. Puis, dans la suite, nous verrons comment acquérir les images numériques (numérisation d'images analogiques) que nous souhaitons traiter.

Introduction

« Une image vaut mille mots,
et une vidéo vaut mille phrases »

Beaucoup d'informations dans une donnée visuelle

Exemples d'images autour de nous

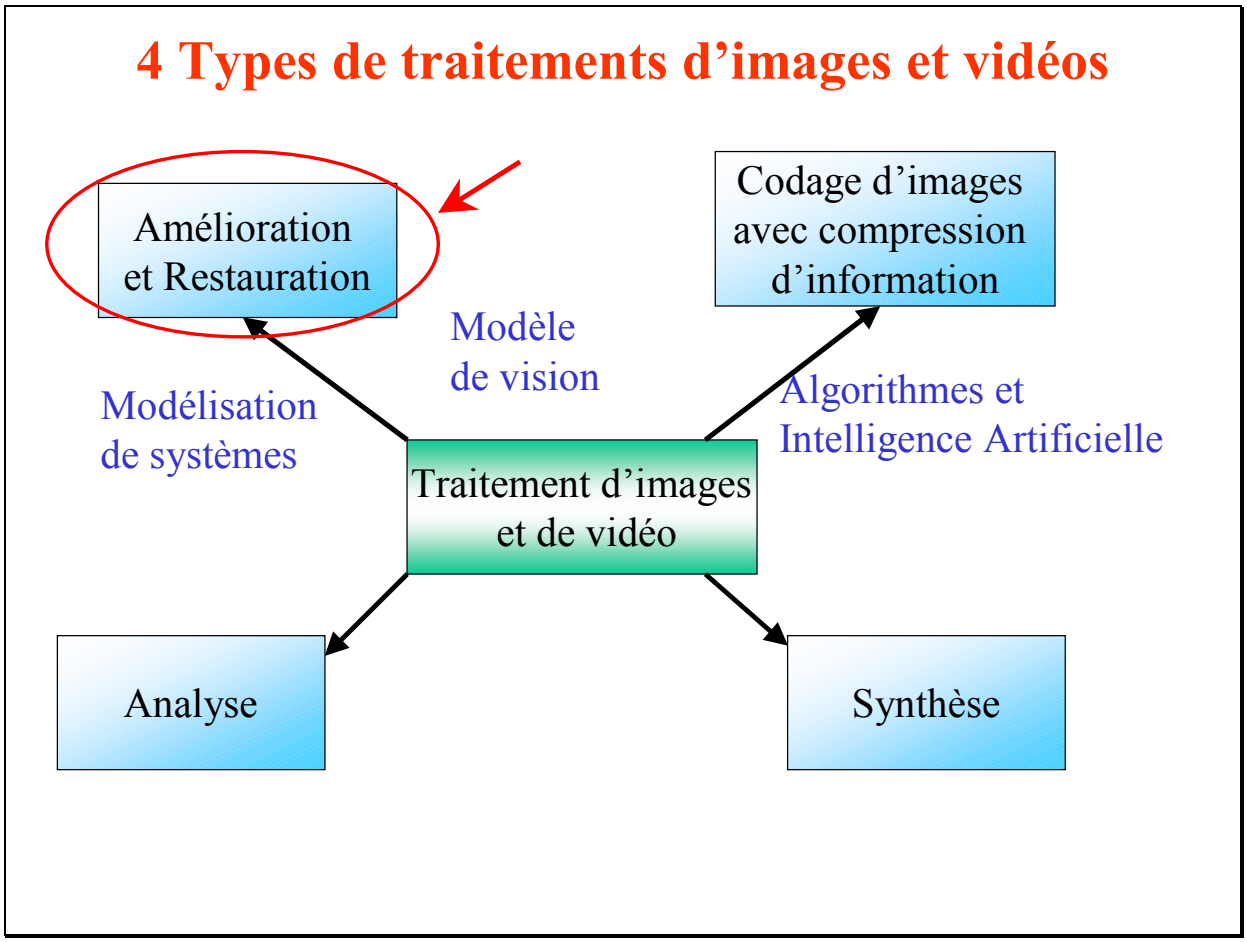
- les photographies de scènes naturelles
- les dessins artistiques et industriels
- les images scientifiques (satellite, médical, …)

« Images animées » => Vidéo

- les films, programmes TV
- les vidéos familiales
- la vidéo surveillance (autoroutes, trains, lieux publics,…)

Quotidiennement nous rencontrons des images de toutes sortes dans notre environnement : des photographies de paysages, de personnes, des peintures, des dessins par ordinateur, des images de radiologie médicale, des images prises par des satellites, … Certaines de ces images (satellite, médical, …) ne peuvent être observées directement, d'autres images présentent des caractéristiques à extraire automatiquement, à stocker et envoyer… les traitements envisageables sur les images sont très variés, car les images que nous rencontrons dans notre environnement sont diverses tant par leur nature et leurs caractéristiques que par la scène qu'elles décrivent. Elles sont toutes différentes et il n'est bien évidemment pas envisageable de créer un traitement spécifique pour chacune d'entre elles. Il a donc fallu créer une classification non pas en fonction des caractéristiques des images, mais en fonction de l'objectif du traitement. Quatre types de domaines d'application du traitement d'images numériques se distinguent :

- la restauration et l'amélioration d'images,
- l'analyse d'images,
- le codage avec compression d'information,
- la synthèse d'images.



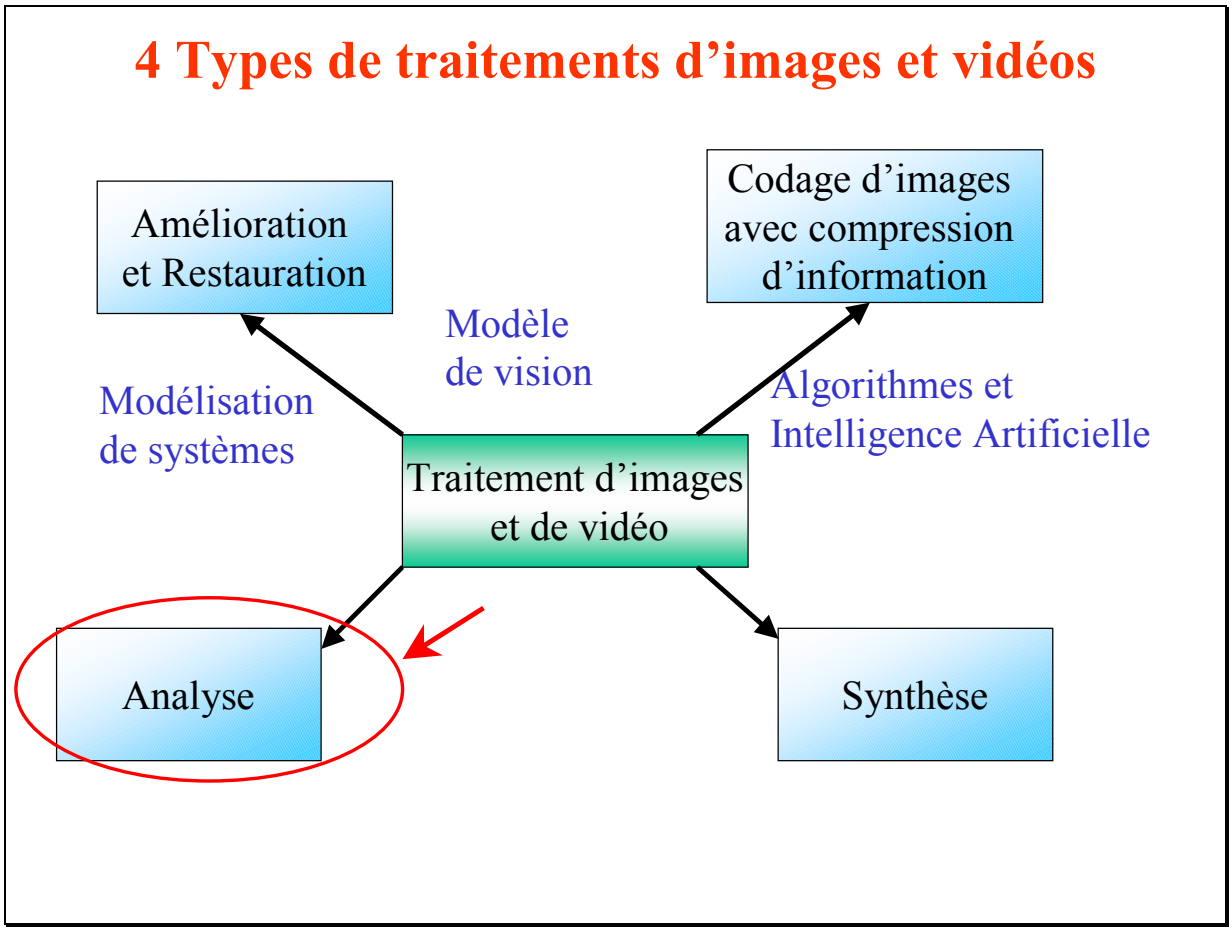
Définissons maintenant plus en détails les quatre domaines d'application liés au traitement numérique des images.

L'amélioration et la restauration :

Si on considère une image observée I_0 à laquelle on associe le signal s_0 que l'on modélise par : $\mathbf{s_0 = f(s_u, d, b)}$

où : - s_u est le signal utile d'image provenant d'une image idéale (sans dégradation) ;
- d est la fonction de distorsion qui opère sur l'image idéale (flou, …) ;
- b est le bruit ;
- *f* est une fonction d'observation dépendante des deux signaux s_u et b et de la fonction de distorsion.

Le traitement effectué sur I_0, qui donnera en sortie l'image transformée I_T, doit permettre d'exploiter l'information contenue dans I_T de manière plus efficace que l'information dans l'image directement observée I_0. Dans le cas d'une modification des caractéristiques de présentation de l'image, on parlera de « traitement d'amélioration », alors que dans le cas d'une inversion partielle du phénomène de dégradation on parlera de « traitement de restauration » (exemples : filtrage linéaire 2-D, filtrage 2-D adaptatif, …).

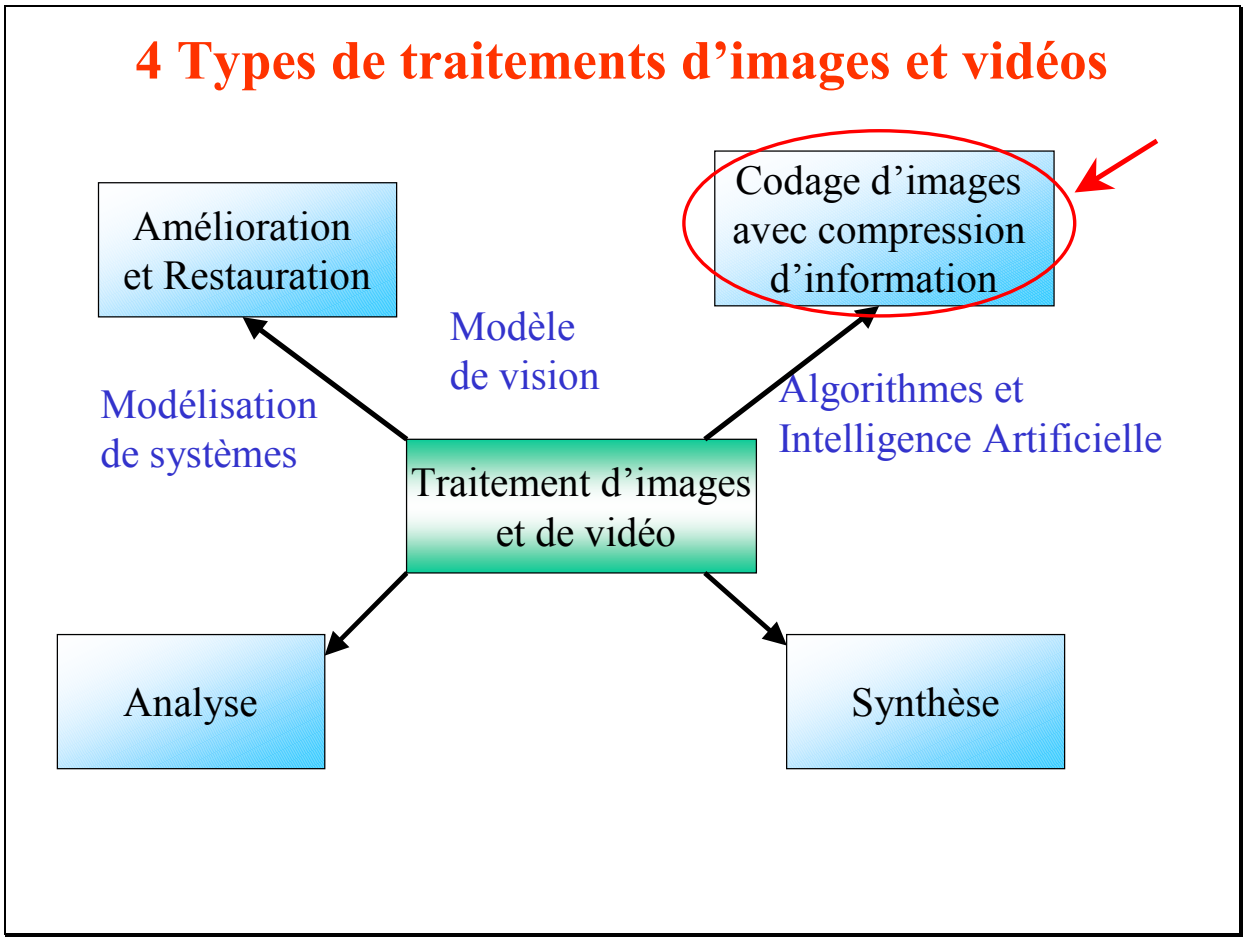


L'analyse d'images :

Il s'agit de décrire partiellement ou complètement la scène à partir de l'image observée (objets présents, dimensions, position dans l'espace, …).
Classiquement ce processus d'analyse se décompose en 3 phases successives :

- pré-traitement et extraction des traits caractéristiques,
- classification et reconnaissance de formes,
- description de l'image et, éventuellement, interprétation.

Il faut cependant noter que l'analyse d'images varie également en fonction du support : les problématiques soulevées par l'analyse d'images 2-D, l'analyse d'images 3-D, ou l'analyse d'images animées varient, si bien qu'on différencie ces dernières en raison des techniques mises en œuvres et de la nature des objectifs recherchés.



Le codage d'images avec compression d'information :

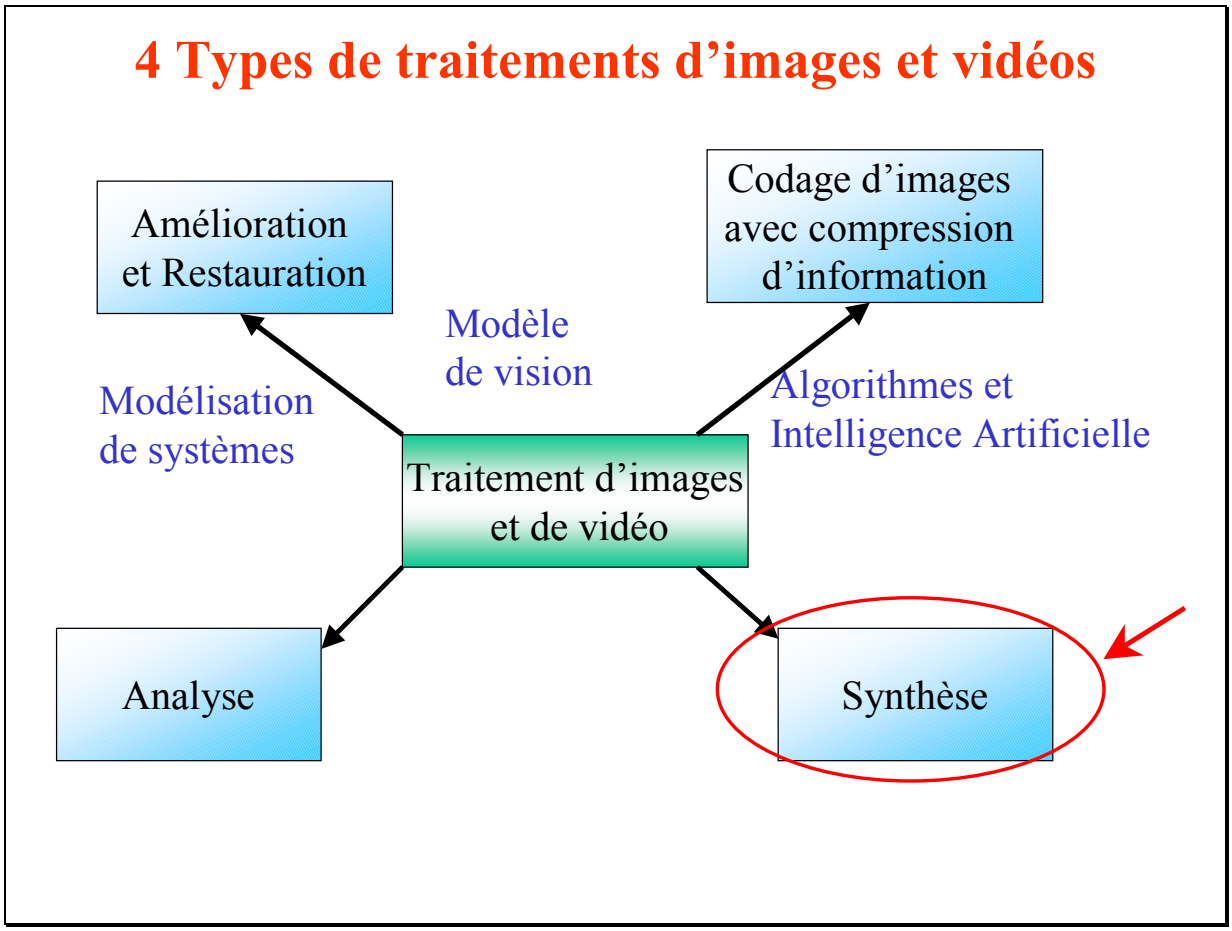
La représentation de base d'une image numérique correspond à un tableau 2-D rectangulaire d'éléments appelés pixels. Il faut donc mémoriser un grand nombre de pixels. Pour une simple image monochrome (en niveaux de gris), il faut mémoriser typiquement 512×512 pixels (soit 256 000 pixels) et 8 bits par pixel pour obtenir une bonne résolution (8 bits pour coder un pixel, ce qui donne par pixel un nombre de 2^8 = 256 valeurs possibles). Pour des images couleurs de haute résolution, et pour des images animées le nombre de bits nécessaires à la représentation de base devient vite très important pour mémoriser ou transmettre.
Le codage d'une image a donc pour but d'obtenir de cette dernière, une représentation qui ne nécessite qu'un nombre très réduit de bits en comparaison de l'image de base.
Pour mesurer cette réduction, on utilise le taux de compression τ défini par :

$$\tau = \frac{\text{nombre de bits pour la représentation de l'image de base}}{\text{nombre de bits pour la représentation de l'image après codage}}$$

Ce taux doit être supérieur à 1 pour effectuer vraiment une compression.

Le nombre de bits pour la représentation de l'image de base étant fixé, le taux de compression est en fait inversement proportionnel au nombre de bits pour la représentation de l'image après codage. Si on souhaite une reconstruction exacte de l'image après décodage, il faut alors utiliser un codage réversible. Ceci génère une contrainte telle que le taux de compression est

très souvent peu élevé. Pour augmenter significativement le taux de compression, il suffit de se contenter d'une représentation visuellement exacte. Dans ce cas, l'œil ne distingue aucune différence entre l'image d'origine et l'image reconstituée après décodage. De plus, la complexité du codage et du décodage doit être limitée. Le codage d'une image numérique implique donc de chercher un juste équilibre pour obtenir un taux de compression assez important afin de faciliter la mémorisation et la transmission, mais suffisamment bas pour ne pas engendrer de dégradations gênantes, tout en gardant à l'esprit que la complexité du décodage doit être maîtrisée.

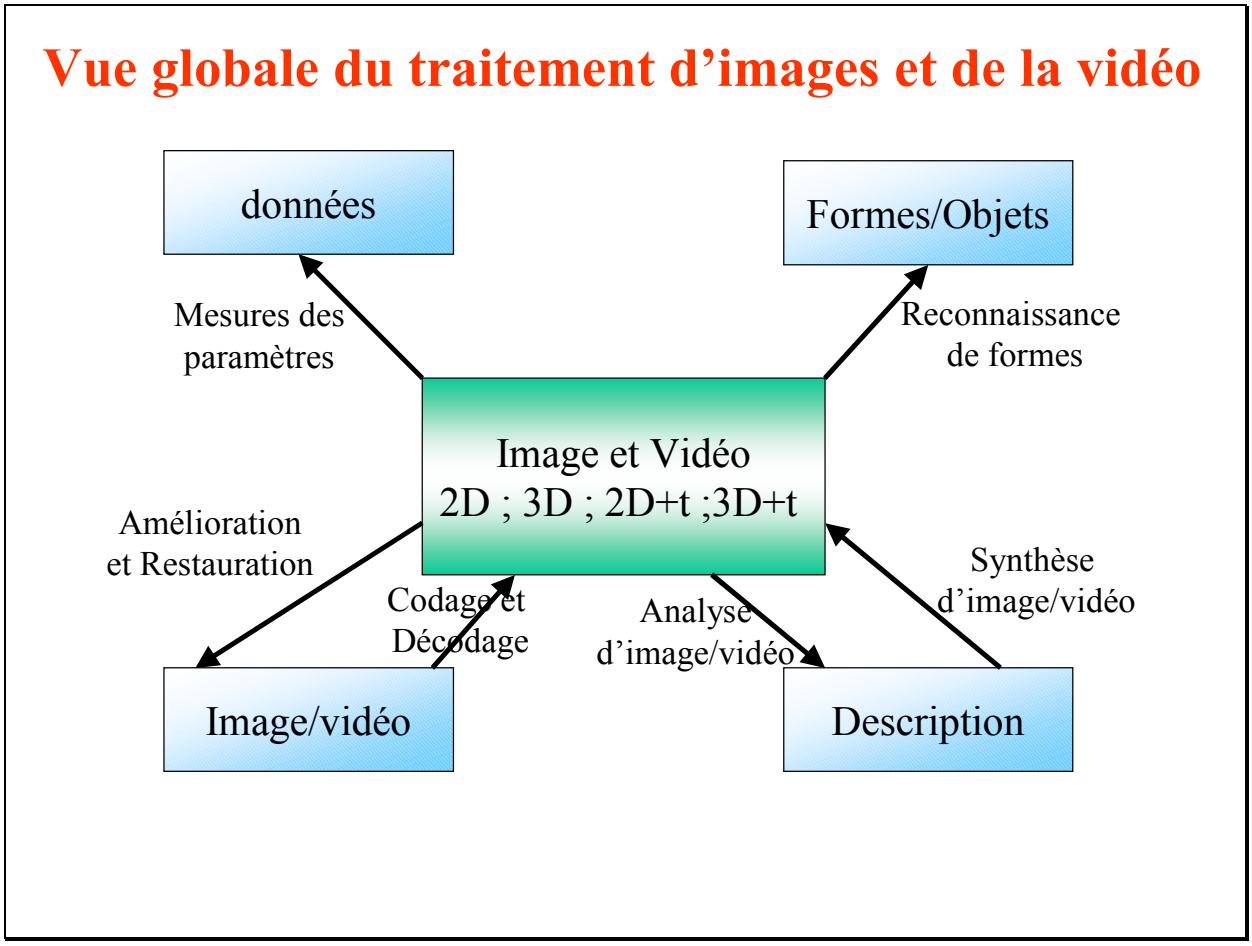


La synthèse d'images :

Le but est de reconstruire une image qui ressemble à l'image de la scène simulée, à partir d'une description de la scène, des objets qui la composent, des caractéristiques de l'éclairement (incidence, intensité, …) et, enfin, du dispositif de prise de vue (caméra CCD, CMOS, …). Cette scène reconstruite peut être ressemblante à la réalité ou purement imaginaire. Les premières applications liées aux images de synthèse furent d'abord orientées vers les simulateurs d'entraînement (vol, engins terrestres, …) puis se sont amplement diversifiées (audiovisuel, cinéma, art, …).

Enfin, il convient de signaler que le traitement d'images s'appuie également sur les études liées à la **structure des machines de traitement**. De fait, une image comporte un nombre de données très important. Il y a en effet, pour une image animée, N×M×P échantillons par seconde à traiter (N points par lignes, M lignes et P images par secondes). Le traitement d'image nécessite donc une importante puissance de calcul. Il demande des architectures très performantes à haut degré de parallélisme et à haute vitesse de traitement.

Le traitement numérique des images vient d'être présenté en fonction de ses quatre grands domaines d'application. Il est également possible d'aborder différemment cette présentation en s'intéressant maintenant à la nature des résultats du traitement.



On peut caractériser le traitement d'image, non plus directement en fonction de ses domaines d'application, mais en fonction de la nature des résultats obtenus en sortie. Deux types d'entrées peuvent être considérés : une image ou une description.

- En partant d'une image en entrée

La nature de la sortie peut être :

- de type image
  C'est le cas du codage avec compression d'information, de l'amélioration d'image et de la restauration d'images dégradées ; tous trois ayant été présentés précédemment.

- de type donnée
  C'est le cas lorsqu'on effectue une analyse d'image de façon élémentaire. On s'intéresse par exemple aux dimensions spatiales d'un objet de la scène, à sa position, ou à sa couleur.

- De type forme
  C'est également le cas de l'analyse d'images dans une version plus élaborée. Il s'agit d'extraire et de reconnaître les formes observées dans la scène.

- De type description de scène
  Il s'agit également d'une sortie possible lors d'une analyse d'images, mais dans sa version la plus avancée. L'image est entièrement décomposée, afin que chaque objet présent dans la scène puisse être reconnu. La scène est totalement décrite, et peut être interprétée.

- En partant d'une description en entrée

Pour la sortie, la seule nature à prendre en compte est alors de type image. Le domaine d'application mis en jeu est alors « la synthèse d'image ». On souhaite reconstruire l'image en fonction de la description donnée : quels objets sont présents ? quelles sont leurs dimensions ? leur emplacement dans la scène ? comment sont-ils éclairés ? Quels sont les paramètres (focale, angle de vue, ...) du capteur qui effectue la prise de vue ? …

Le traitement d'image étant maintenant présenté sous deux aspects différents, mais néanmoins fortement liés entre eux, il convient de s'attarder sur les caractéristiques du signal que l'on souhaite traiter : l'image.

Propriétés de base des Images et Vidéos

- L'image (resp. la vidéo) est un signal 2-D (resp. 2-D + t) scalaire ou vectoriel qui est complexe car:
  - Non stationnaire
  - Non gaussien
  - Non isotrope
- Deux (*images fixes*) ou trois (*vidéo*) caractéristiques principales
  - les contours: changement abrupt dans une caractéristique importante
  - la texture: variation spatiale (souvent la luminance) du signal 2-D pour laquelle la valeur moyenne ne change pas

    Les bords sont localement des signaux 1-D, alors que la texture est toujours un signal 2-D
  - les mouvements: pour la vidéo (variation temporelle du signal à une même coordonnée spatiale)

L'image est un signal 2-D scalaire (image monochrome) ou vectoriel (image couleur par exemple). La vidéo, qui est une suite d'images ordonnées temporellement, est un signal 2-D+t (t : temps) scalaire (vidéo monochrome) ou vectoriel (vidéo couleur).
Le signal « Image » est complexe car :

- il est non stationnaire : son contenu en fréquences spatiales change avec les coordonnées spatiales ;
- il est non gaussien : ses propriétés statistiques ne suivent pas une loi de probabilité gaussienne ;
- il n'est pas isotrope : les propriétés du signal d'image ne sont pas les mêmes avec l'orientation (par exemple, dans les images prises au sol, les directions horizontales et verticales sont plus fréquentes pour les contours que les directions obliques).

Les méthodes et les outils classiques, utilisés en traitement du signal, sont très fréquemment conçus pour des signaux stationnaires, gaussiens ou isotropes (Transformée de Fourier Discrète, …). Ils ne peuvent donc pas être appliqués directement aux images tels quels.

Par ailleurs, les images sont principalement caractérisées par deux types d'éléments : les ***contours*** et les ***textures*** :

- les contours correspondent à des changements rapides de caractéristiques importantes quand on passe d'une zone A à une zone B de la scène (valeur moyenne, description de texture). Les bords peuvent être considérés localement comme des signaux 1-D ;
- la texture correspond aux variations spatiales du signal 2-D d'image autour de sa valeur moyenne locale ;
- Le mouvement des objets dans une scène entraîne des modifications temporelles dans les images successives d'une vidéo.

Les images et le traitement d'images numériques sont maintenant présentés dans leur globalité. Dans la suite de ce chapitre, on s'intéressera d'une part aux méthodes de représentation des images (numérisation puis codage) qu'il faut employer afin de pouvoir effectuer un traitement numérique, d'autre part à des exemples concrets de résultats liés aux différents domaines d'application vus précédemment.

Chapitre 1

TRAITEMENT DE SIGNAL MULTIMEDIA

Exemples de Traitements d'Images

Analyse d'images et de vidéos

Les quatre grands domaines d'application du traitement d'image ont été présentés de façon globale : quelle problématique ? quels objectifs ? avec quelles méthodes et quels outils? Il est donc intéressant de s'attarder, à présent, sur des applications concrètes issues de ces différents domaines.

Analyse d'images et de vidéos

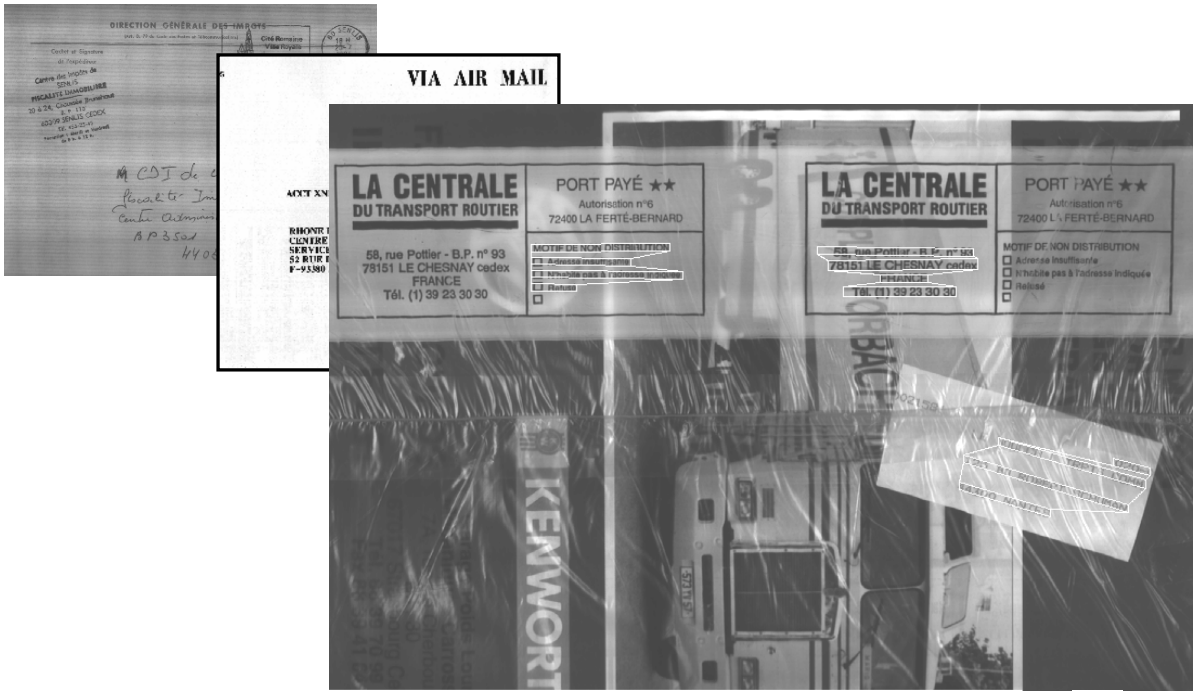
- Objectifs
  - Détection d'objets et extraction (segmentation)
    Objets d'intérêt
  - Reconnaissance de formes
    Classification et identification d'objets
  - Interprétation et analyse de la scène
    - Relationnelle
    - Description quantitative
    - Description qualitative

Le premier domaine auquel on s'intéresse est « l'analyse d'images ». Il recouvre un très grand nombre d'applications potentielles, c'est sans doute le domaine pour lequel les exemples sont les plus variés (analyse médicale : classification des chromosomes ; lecture automatique : authentification de signatures ; la télédétection : segmentation et classification de zones géographiques, …).
On peut néanmoins proposer une démarche type en analyse d'images qui consiste à :

- ***Détecter*** les différents objets présents dans la scène et les extraire (segmentation, découpage en régions) ;
- ***Caractériser*** ces objets (dimensions, emplacement, …) ;
- ***Reconnaître*** les objets et effectuer leur classification ;
- ***Analyser*** et ***Interpréter*** la scène en fonction des résultats précédents. La scène pourra, entre autres, être décrite de façon quantitative et/ou qualitative.

Il faut cependant rappeler que les applications sont très variées, et l'approche globale proposée ici peut parfois nécessiter quelques modifications dans les étapes de traitement, en fonction des difficultés rencontrées.

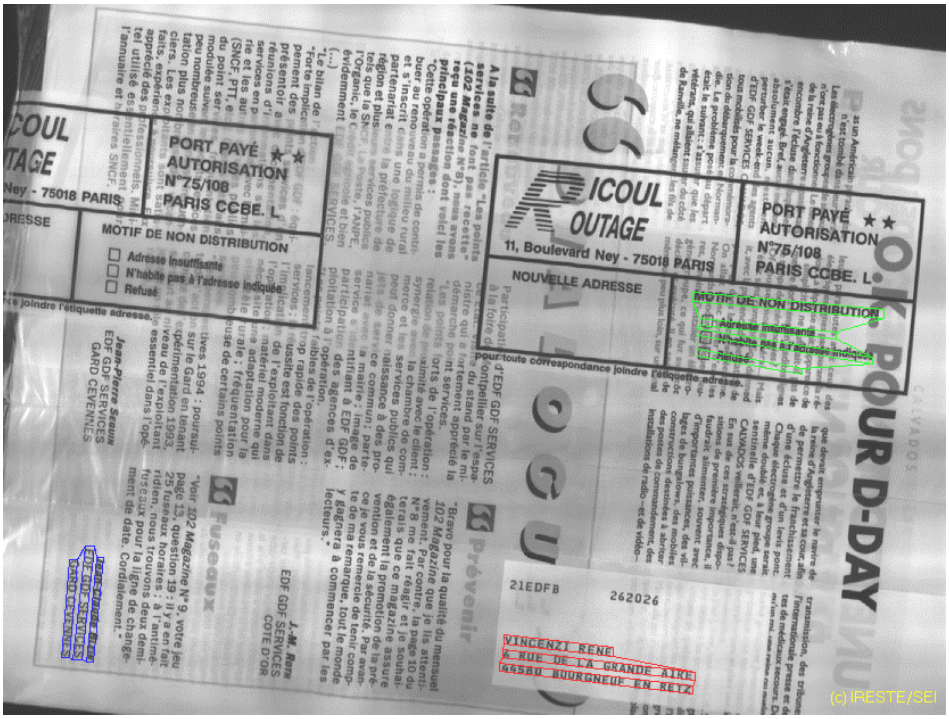
Détection et extraction d'un bloc adresse



L'exemple, présenté sur la figure ci-dessus, met en avant la détection et l'extraction d'une adresse de livraison. L'application servirait au tri automatique dont la nature dépend des objets postaux et des modes de dépôts. La plupart des pays industrialisés utilisent des systèmes de tri automatique, les obstacles varient d'un pays à l'autre en raison des différences qui existent, entre autres, pour les codes postaux (chiffres exclusivement ? taille ? …).

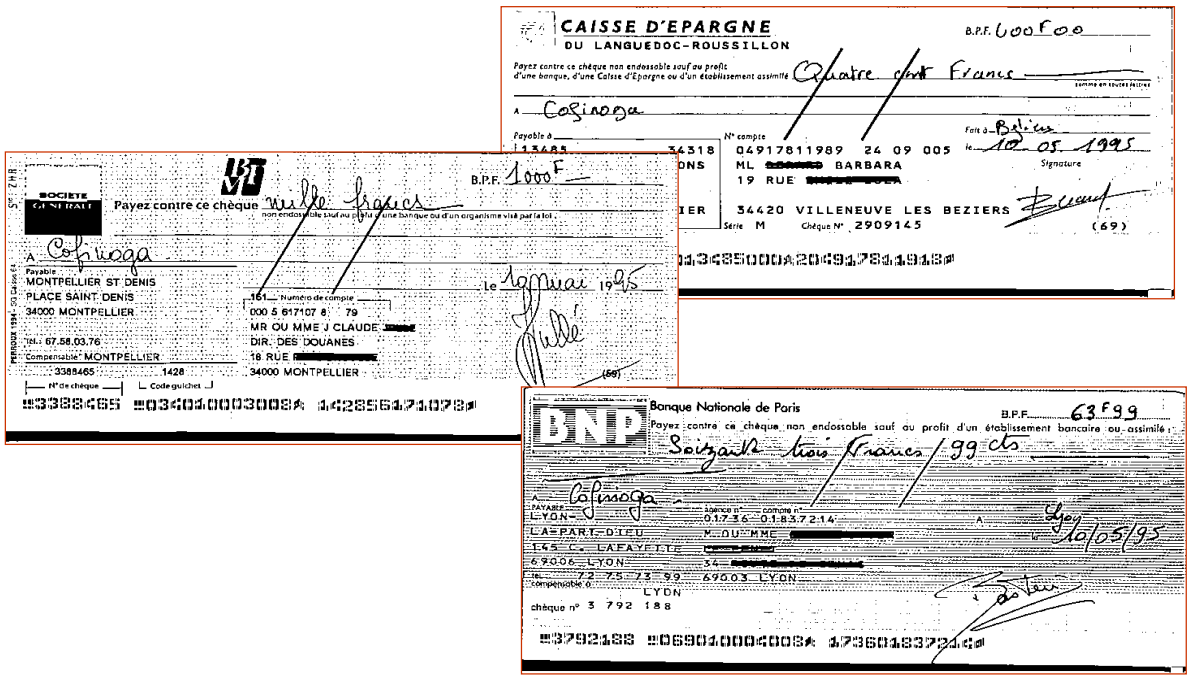
Divers algorithmes de traitement existent pour y parvenir. On peut par exemple remarquer que les caractères du texte forment une texture régulière contenant des contours verticaux allongés horizontalement. En appliquant des filtrages directionnels, il est alors possible de mettre en avant les pixels de l'image liés au texte. On peut par exemple effectuer une binarisation suivie d'un post traitement afin d'extraire les rectangles englobant des zones de texte. Le texte étant séparé du reste de l'image, on pourrait ensuite effectuer une analyse des composantes connexes pour trouver celles correspondant aux zones du texte. Le traitement global de l'image étant effectué, il faudra alors appliquer des traitements locaux et considérer des contraintes diverses (par exemple géométriques) pour éviter les erreurs. Ces différents traitements seront présentés dans les chapitres à venir.

Détection et extraction d'un bloc adresse



L'image ci-dessus est également le résultat d'un traitement d'analyse. L'objectif est de nouveau de détecter et d'extraire sur l'image les zones de textes liées à l'adresse du destinataire. Cependant l'image ci-dessus présente de nouvelles difficultés : l'arrière plan semble beaucoup moins uniforme que celui présenté précédemment, car il est composé de texte. Il faut donc également être en mesure de distinguer parmi les caractères qui se chevauchent, ceux qui sont liés au « bloc adresse de l'expéditeur » et ceux qui sont liés au reste de l'image. Cependant le texte de l'arrière plan et le texte lié à l'adresse sont perpendiculaires l'un à l'autre. Il parait donc possible de segmenter l'image en appliquant des filtres directionnels adaptés. Puis reprendre la chaîne Caractérisation – Reconnaissance – Interprétation d'un traitement d'analyse d'images, afin d'obtenir comme sur l'image présentée une détection précise du bloc d'adresse destinataire.

Extraction et lecture du montant sur les chèques



Le dernier exemple présenté pour l'analyse d'image concerne l'extraction et la lecture du montant inscrit sur un chèque. Elle bénéficie d'une attention particulière compte tenu de ses enjeux industriels et économiques. De nombreuses publications et méthodes existent dans le domaine, les utilisateurs potentiels de la reconnaissance automatique semblent néanmoins en attente de résultats plus fiables. De ce fait, le problème est considéré comme étant encore ouvert. Il s'agit donc principalement d'une reconnaissance de chiffres et de lettres manuscrits dans un environnement dont la complexité impose l'utilisation de techniques appropriées. En effet, outre la grande variabilité dans la morphologie des caractères d'un individu à l'autre, il est nécessaire de prendre en considération des problèmes tels que la non standardisation du format d'un chèque (position des champs, type de fond) et le chevauchement des entités composant le montant (chiffres, ligne de base, séparateur décimal).

Les principaux produits ou prototypes disponibles en France ont pour origine : le SRTP (La Poste) avec Dassault AT (Automate de remise de chèques lisant le montant numérique seulement) ; Matra (MSI) et son produit LireChèques ; et enfin le produit InterCheque de a2ia qu'on retrouve chez plusieurs banques (Société Générale, Banque Populaire, ...) et dont les taux de reconnaissance avoisinent les 70 % grâce à une technologie s'appuyant sur les approches par réseaux neuronaux et sur les modélisations Markoviennes.

Exemples de Traitement d'images

Codage des images et vidéos avec compression d'information

Après ces quelques exemples d'applications de l'analyse d'images, nous allons à présent nous intéresser aux applications liées au codage d'image avec compression de données.

Contexte

- Extension des techniques numériques à plusieurs domaines du traitement d'information : capture , stockage et transmission

  ⇒ désir d'une méthode unique de stockage, traitement et transmission indépendamment de l'image

- Les signaux audio-visuels ont des propriétés qui requièrent des méthodes dédiées à une représentation efficace de ces derniers :

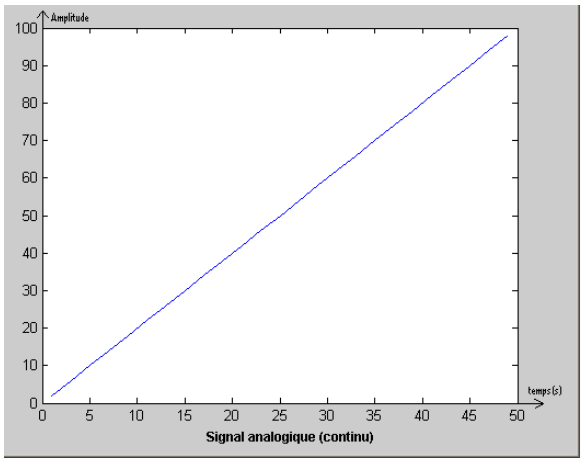
  - les données sont la représentation d'un signal analogique

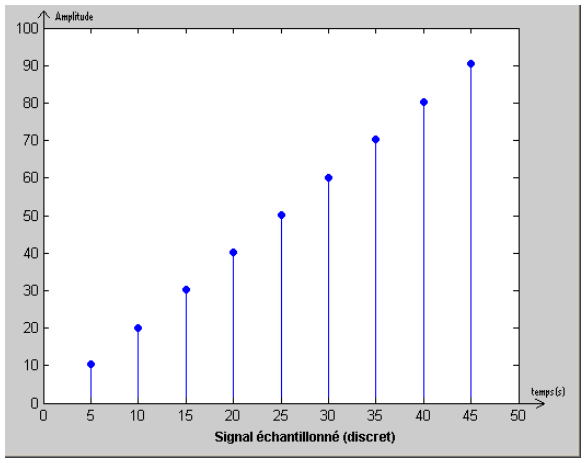
    ⇒ échantillonnage + quantification

  - grande quantité de données pour représenter une petite partie de la source d'information originale (ex: 1s d'audio/vidéo)

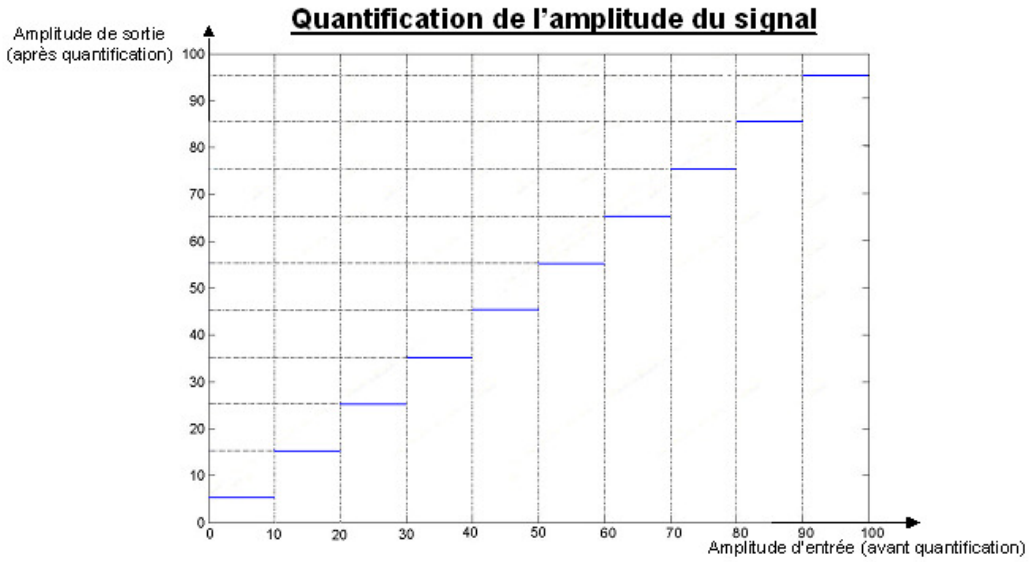
Comme cela a été indiqué dans la présentation, la problématique résulte de la nécessité de réduire de manière importante la quantité d'éléments binaires d'information nécessaire à la représentation des images à des fins de mémorisation ou de transmission à distance. Bien que les images soient très variées, et qu'en conséquence les informations qu'elles contiennent puissent toutes être très différentes (couleur, lumière, objets, …), il demeure préférable de mettre en place une procédure unique de codage à compression d'information, que l'on appliquera à chaque image indépendamment de ses caractéristiques.

Par ailleurs, les signaux audiovisuels sont des signaux analogiques (continus) dont la représentation nécessite une très importante quantité de données (ex : pour une seule seconde de vidéo, il faut être capable de stocker une séquence de 25 images). La représentation efficace de tels signaux requière donc la mise en place de méthodes adéquates.
La représentation binaire d'un signal analogique n'est possible que si ce dernier a subi un échantillonnage (classiquement temporel, spatial, …), afin de pouvoir travailler sur un nombre finis d'échantillons de ce signal. Les amplitudes liées à un signal analogique s'étendent également sur un intervalle continu qu'il va falloir discrétiser selon différents critères : il s'agit de la quantification du signal. Ces deux étapes sont présentées ici brièvement pour un signal 1-D, mais elle seront reprises et détaillées dans la suite de ce chapitre pour un signal image 2-D.





La figure ci-dessus représente un signal de type rampe avant et après échantillonnage. La fréquence d'échantillonnage est égale à 0,2Hz ($\frac{1}{5}$ s^{-1}). L'amplitude de ce signal varie continûment sur l'intervalle [0, 100], il faut donc effectuer une quantification.



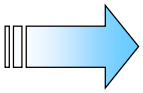
La figure ci-dessus représente un exemple de quantification : si un échantillon du signal a une amplitude comprise dans l'intervalle [10k ; 10 (k + 1)[, « k » étant un entier naturel, sa valeur d'amplitude sera ramenée par quantification à 10k + 5.

Après quantification, les données brutes représentent une grandeur discrète qu'il va être possible de coder. Typiquement on utilise un codage binaire.

Grande quantité de données pour représenter les informations visuelles

- Conditions sur les tailles de données

– **une page Télétexte = 25 lignes × 40 char**	**=> 1 Kbyte**
– **une page A4 de texte = 50 lignes × 80 char**	**=> 4 Kbytes**
– **une seconde de téléphone (300 Hz - 3400 Hz)**	**=> 8 Kbytes**
– **une seconde d'audio en hi-fi stéréo**	**=> 176.4 Kbytes**
– **un fax A4**	**=> 1.1 Mbytes**
– **une image couleur (qualité diapositive: 2000 × 3000)**	**=> 18 Mbytes**
– **une seconde de programme TV (signal vidéo)**	**=> 21 Mbytes**
– **une seconde de programme HDTV**	**=> 230 Mbytes**



Besoin de compression de données

Pour illustrer les problèmes liés à la quantité nécessaire de données pour représenter peu d'information d'un signal image, le document ci-dessus donne à titre indicatif les tailles de données requises pour stocker et transmettre différentes images, que l'on trouve régulièrement dans notre environnement. Attention, les tailles sont proposées ici en unités anglo-saxonnes : 1 byte = 1 octet = 8 bits.
Par exemple, une simple feuille standard (A4) de fax nécessiterait pour la transmission de l'image de base une capacité de 1,1Mo. Ces différents exemples montrent combien il est important de réaliser un codage à compression de données lors d'une transmission ou d'un stockage d'image.

Contexte

- Besoin de mélanger dans le même document différents types d'informations
  - Texte (commun ; intrinsèquement structuré ; niveau sémantique)
  - Tables de données (communes ; intrinsèquement structurées)
  - Graphiques (données vectorielles ; information structurée)
  - Audio (liste de nombreux échantillons temporels ; faible niveau de représentation (non structuré et non sémantique))
  - Image (liste d'échantillons scalaires/vectoriels issus d'un scannage plan 2-D ; faible niveau de représentation (non structuré et non sémantique))
  - Vidéo (groupe de données « Image » selon l'axe du temps ; même propriétés que la donnée « Image »)

Outre le fait qu'une grande quantité de données soit nécessaire au stockage et à la transmission de peu d'information dans le cas d'un signal image, la représentation de ce signal est également rendue difficile par la nature même du signal image. En effet, ce signal mélange des types d'informations parfois nombreux et variés. De fait, une image peut comporter des zones de textes, des tables de données, des graphiques, …

Les compromis « Taux de compression/Qualité » dépendront donc aussi directement du contenu de l'image. Si on considère par exemple une carte géographique, accompagnée d'une légende, il ne serait pas possible de se contenter de pouvoir reconnaître les formes, et les frontières des divers continents, mers, océans, … après décodage. L'écriture ne devra pas être trop dégradée, afin que légendes et noms figurant sur la carte restent facilement lisibles pour tout utilisateur.

Principaux aspects de la compression d'images

- Type d'images à compresser ? Pour quels services ?
- Qualité de l'image :
  - codage d'image sans perte
  - ou codage d'image avec perte

    dans ce cas : avec ou sans dégradations visibles?
- ***Rapport Taux de compression / Qualité***
- Complexité : codage , décodage
- Sensibilité aux erreurs (*robustesse face aux erreurs*):
  - effets des erreurs de transmission sur l'image décodée
- Taux de compression fixe ou variable :

  $\Rightarrow$ *conditions de transmission*

Avant d'appliquer à une image un codage à compression de données, il faut se poser quelques questions, parfois triviales, mais qui influent généralement fortement sur les résultats obtenus après la compression :

- À quel type d'image a-t-on affaire ? pour quelle utilisation ?
- Faut il un codage exact (réversible, mais forte contrainte et donc faible valeur du taux de compression) ? sinon les dégradations peuvent-elles être ou non visibles par l'œil humain (là aussi, le taux de compression sera directement affecté) ?
- Le taux de compression doit être d'autant plus fort que l'on veut gagner en capacité de stockage et de transmission d'images, mais la qualité en sera affectée, quel équilibre souhaite-t-on alors ?
- Une augmentation du taux de compression entraîne une hausse de la complexité du décodeur et donc de la charge de calculs pour le système. Là aussi, il faut trouver un juste équilibre en fonction de l'utilisation prévue.
- Quelle robustesse face aux erreurs ? car les erreurs de transmission pourront avoir des répercutions directes fortes sur l'image décodée.
- Le taux de compression est il fixe ou variable ? dans quelles conditions de transmission se trouvera-t-on ?

En définitive, l'utilisation que l'on veut faire du codage est aussi importante que le codage lui-même. Il faut donc définir avant de commencer le type de compression, et en fonction de ses propres objectifs, choisir le rapport taux de compression/qualité que l'on souhaite obtenir.

En résumé

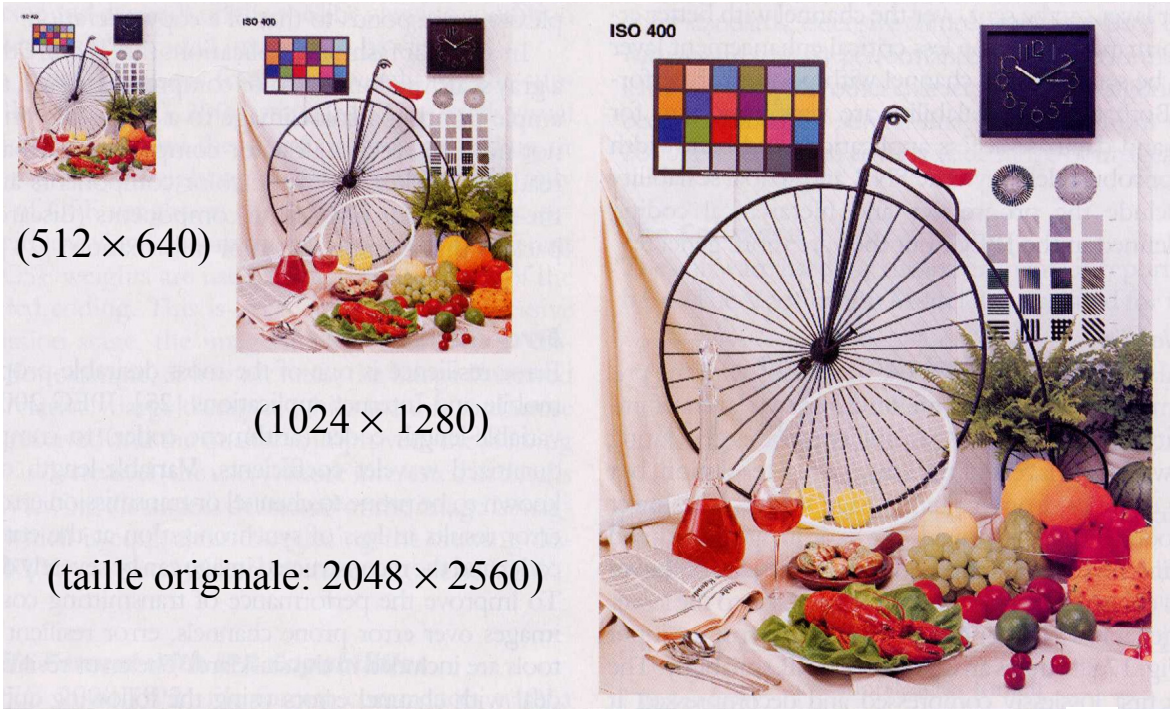
- **Applications complexes**
  - important volume d'information
  - large champ d'applications et de services

  photographie numérique (MMDB)
  ↓
  HDTV
- **Dépendances multiples**
  - quelles images (≈ types et services)
  - quels taux d'images/s, résolution,....
  - quelle qualité requise (*niveau d'acceptabilité*)
  - quel réseau de communication
- **Conditions multiples de résolutions**
  (multi-résolution, qualités ...)
- **Autres fonctionnalités que le codage**

En résumé, ce qui vient d'être présenté conduit à la conclusion que la mise en place d'un codage d'image à compression de données est une procédure complexe. D'une part, un très important volume d'information et un large spectre d'applications et de services sont considérés. D'autre part, le codage en lui-même dépendra de nombreux paramètres : l'image à traiter (type, contenu, utilisation, …) ; la résolution ; la qualité nécessaire à l'utilisation envisagée et le seuil de dégradation accepté ; le réseau de transmission utilisé… À ces difficultés peut venir s'ajouter une complexité, liée à des fonctionnalités autres que le codage lui-même.
La suite présente le cas d'un codage pour une image haute définition, cette dernière génère, de par sa nature, une contrainte qui influe directement sur la complexité de la compression.

Exemple d'une image HD : Image ***Bike***

(décodée à trois niveaux résolution depuis le même traitement de codage)



La figure ci-dessus présente l'image haute définition (HD) « *Bike* » qui après codage à compression de données, à été décodée à 3 niveaux de résolution et de tailles différents, mais en conservant le même traitement. La présentation est faite de façon pyramidale, de la résolution la plus grossière (à gauche de l'image) à la plus fine (à droite). L'image qui possède la plus grande résolution apparaît nette, les couleurs et les détails sont bien restitués. Les dégradation ne semblent pas perceptibles par le système visuel humain dans ce cas. Cependant, pour une image HD telle que « *Bike* » à la résolution 2048×2560, de nouveaux problèmes de codage à compression apparaissent.

Remarque : attention toutefois à la représentation donnée sur la figure ci-dessus, si l'affichage ne permet pas de garder les mêmes résolutions que celles d'origine, il y'a un risque de sous-échantillonnage.

Problèmes

- ***Numérisation des signaux audio-visuels***
  - problèmes spécifiques avec les images / vidéo HD
    - capteurs d'image (capteurs multi-spectral, résolution, sensibilité, capture à grande vitesse de luminance)
    - quantification (grande précision pour l'audio / quelques systèmes d'imagerie, grande vitesse de conversion)
- ***Besoin d'un fort taux de compression en gardant une bonne qualité***

  Pour stockage et transmission (réduction de bande passante)
- ***Autres conditions***
  - adaptations multiples (taux de transmission, résolution, robustesse face aux erreurs, paquets)
  - accès au contenu partiel, description du contenu, …

  ⇨ Standards JPEG2000, MPEG4, MPEG7

Lors de la numérisation des signaux audiovisuels, des problèmes spécifiques aux images « haute définition » (HD) apparaissent :

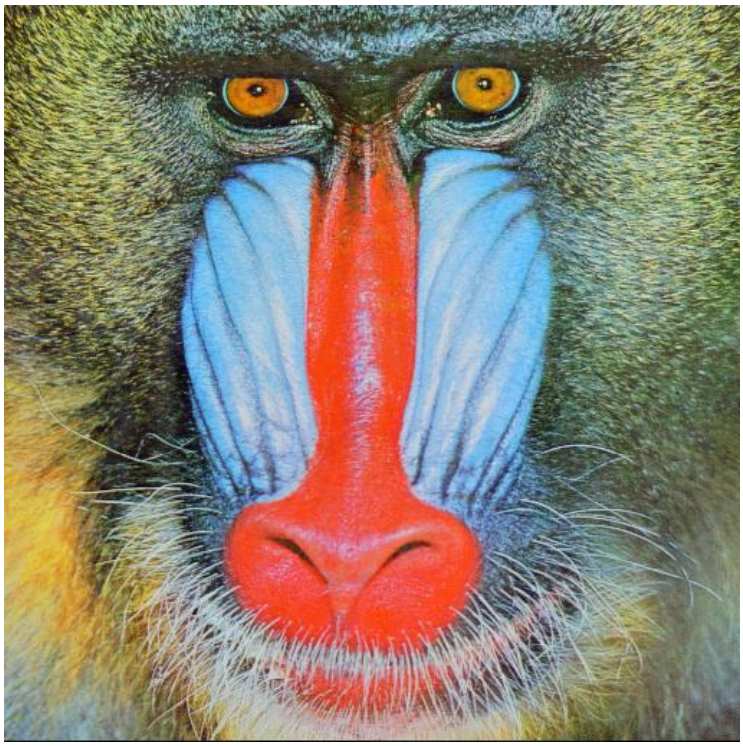
- Le type de capteur à employer (capteurs de type multispectral, résolution, sensibilité, capture à grande vitesse de luminance) ;
- La manière dont est choisie la loi de quantification.

De plus, on cherche à atteindre des taux de compression importants, au vue des applications envisagées. Cependant, par définition, on souhaite conserver un degré élevé de qualité (image haute définition). Le compromis entre le taux de compression et la qualité atteint donc des limites. Enfin, en prenant en compte d'autres types de contraintes, tels que la nécessité de l'adaptation du codage (fonction de la robustesse aux erreurs, du mode de transmission, …), il est apparu essentiel de créer des standards de compression qui pourraient répondre aux différentes contraintes dégagées par ces problèmes et s'appliquer aux images indépendamment de leur nature. Les standards les plus répandus pour le codage et la compression des images et de la vidéo sont : JPEG, GIF, JPEG2000, et MPEG4.
La suite présente, à titre d'illustration, différents résultats obtenus pour l'image « *Mandrill* » qui a été codée en JPEG avec des paramètres et des taux de compression différents.

Exemple de codage JPEG : original de *Mandrill*

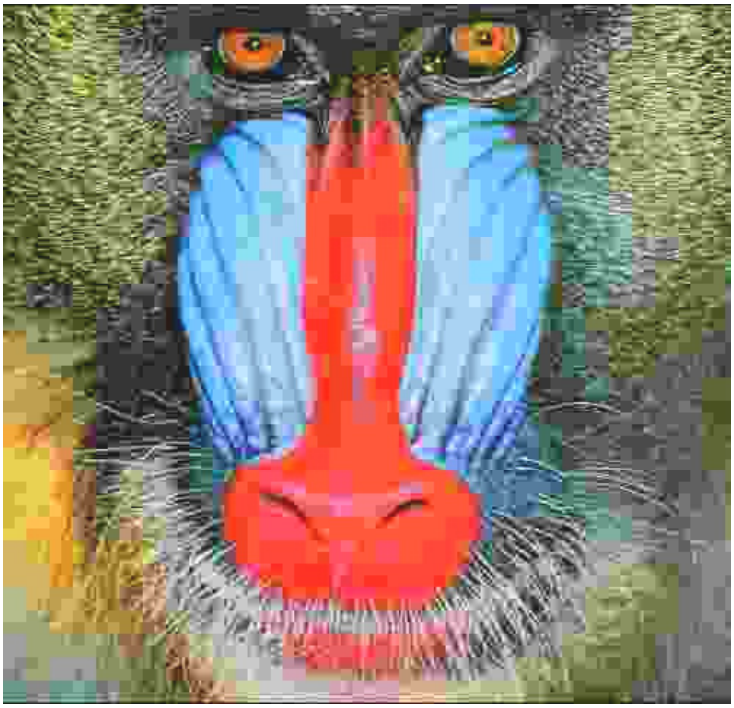
Original:
512×512
$\Rightarrow$24 bpp

$\Rightarrow$ **768 Ko**



Ci-dessus l'image « *Mandrill* » originale avec une taille de 512×512 pixels. Un pixel est codé sur 24 bits (trois fois 8 bits car la représentation initiale de l'image couleur est R-V-B, il y'a donc 2^{24} niveaux possibles pour chaque pixel). L'image est donc représentée par un volume de 512×512×24 = 768 Ko. Nous allons faire subir à cette image une série de codages JPEG, avec différents paramètres, pour comparer la qualité visuelle des résultats et les taux de compression obtenus.

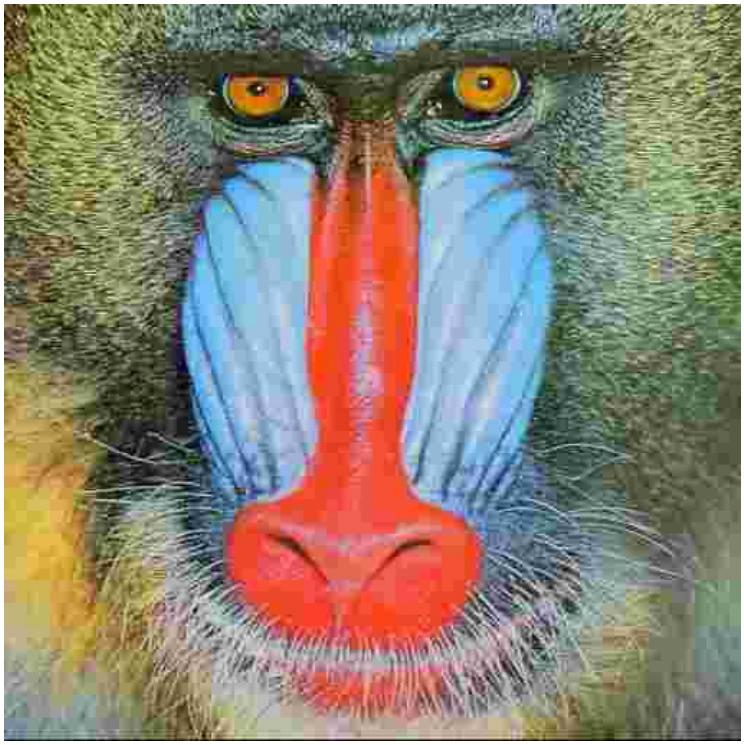
Exemple de codage JPEG (q = 5 ; 9 Ko)



$\tau_C = 768/9$
≈ 85

Ci-dessus l'image « *Mandrill* » qui a subie un codage JPEG. La quantification a été réalisée en utilisant un facteur de qualité q faible (q = 5). Le gain en compression est donc très important. La taille des données décrivant l'image n'est plus que de 9Ko. Le taux de compression τ_C est très fort, et vaut 768/9 = 85. Cependant, l'image a énormément perdu en qualité visuelle. Les formes et les couleurs sont beaucoup plus grossières que sur l'image originale, et un effet de bloc est apparu très nettement sur l'ensemble de l'image (des blocs de pixels sont représentés quasiment par la même couleur).

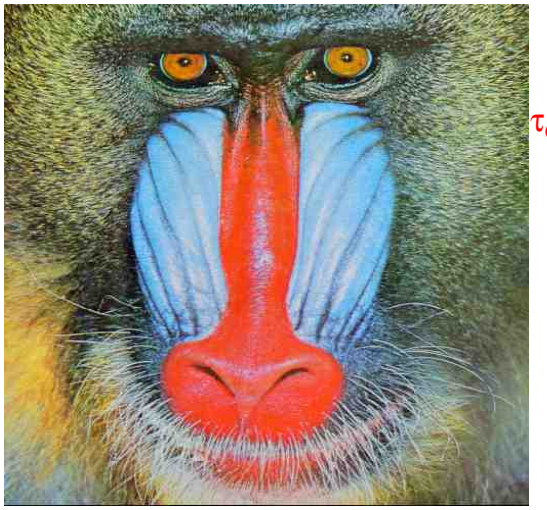
Exemple de codage JPEG (q = 10 ; 17 Ko)



$\tau_C = 768/17$
≈ 45

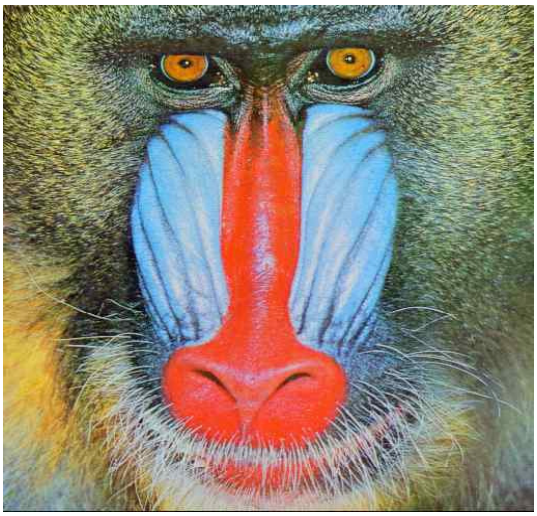
Un autre exemple de résultat avec un facteur de qualité plus fort quoique restant faible (q = 10). Le taux de compression est moins important que précédemment ($\tau_C = 45$). L'image a bien gagné en qualité, et l'effet de bloc est beaucoup moins apparent, bien qu'il soit encore visible sur les couleurs vives rouges et bleues du museau du babouin. La gamme de couleurs est moins importante que sur l'image originale, mais elle permet tout de même d'obtenir un meilleur rendu sur le pelage du singe, que lors du codage précédent. Notons qu'en doublant le facteur de qualité q, le taux de compression a été divisé par 1,88 : il est presque deux fois moindre.

Exemple de codage JPEG (q = 25 ; 32 Ko)



τ_C =768/32
≈ 24

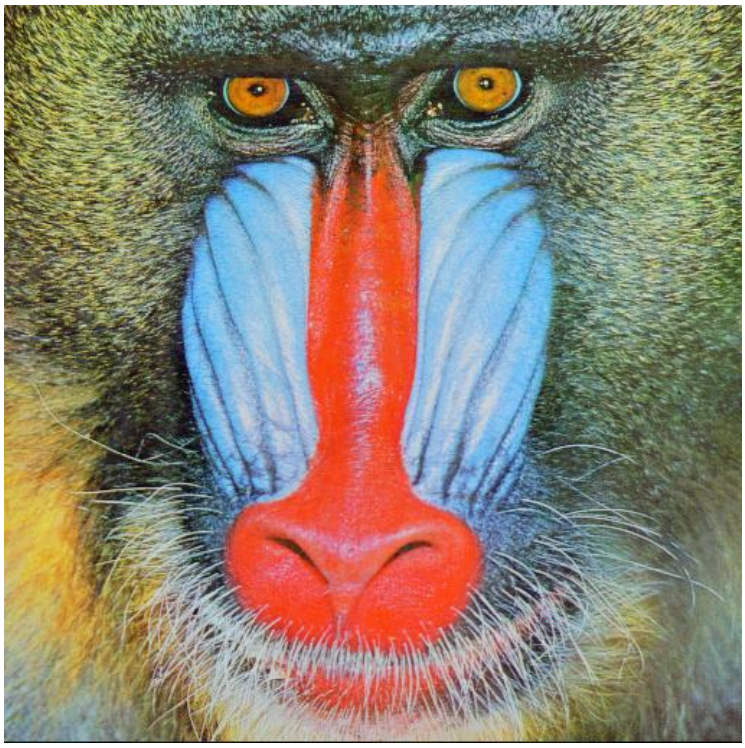
Exemple de codage JPEG (q = 35 ; 40 Ko)



τ_C =768/40
≈ 19.2

Deux autres résultats obtenus en cherchant à privilégier encore un peu plus la qualité de l'image obtenue par rapport au gain en compression. Cependant le taux de compression n'est plus divisé que par 1,25 cette fois-ci. L'évolution du taux de compression en fonction de la quantification n'est en général pas une fonction linéaire.

Exemple de codage JPEG (q = 75 ; 76 Ko)



τ_C =768/76
≈ 10.1

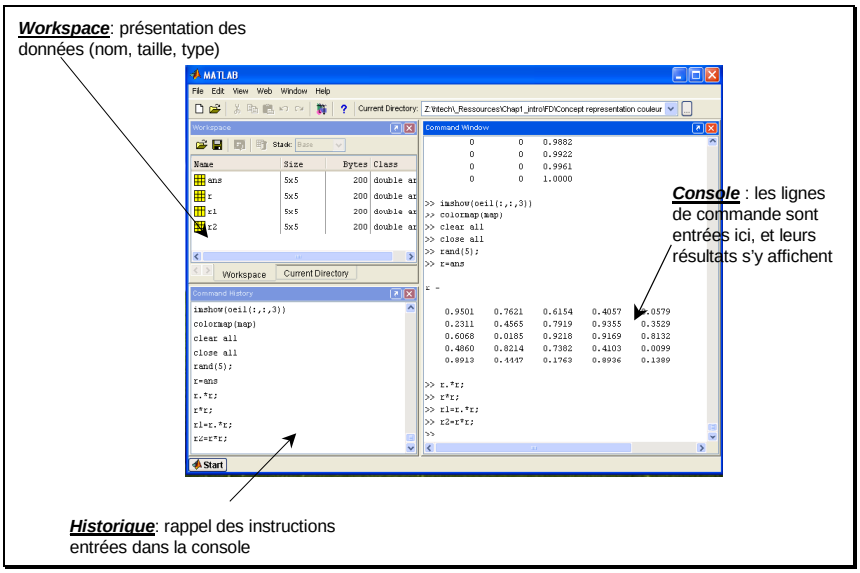
Enfin l'image ci-dessus présente un résultat où la qualité a été fortement privilégiée par rapport à l'importance de la compression. Le nombre de bits nécessaires au stockage de l'image codée est simplement 10 fois moins important que celui nécessaire au stockage de l'image originale. Il est cependant très difficile pour le système visuel humain de détecter les dégradations subies par cette image : la différencier de l'image d'origine au niveau de la gamme de couleur et de la netteté des contours et des textures n'est guère possible.

Le traitement d'image a donc été présenté sous différents aspects. Il est né des nouveaux besoins qu'ont engendrés les nouvelles technologies, et s'est grandement appuyé sur ces mêmes nouvelles technologies pour se développer très rapidement. Avec son expansion se sont dégagés quatre grands domaines d'application : la restauration, le codage, l'analyse, et la synthèse d'images.
Dans ce cours, les domaines de l'analyse et du codage ont ensuite été accompagnés d'exemples d'applications concrètes, afin d'obtenir une première vision des possibilités et des résultats qu'offrent les différents traitements d'image.
Toutefois, dans la nature, les images qui nous entourent ne peuvent généralement pas être traitées directement, et ne sont pas numériques. Il faut donc les numériser pour pouvoir les traiter avec des algorithmes et des systèmes adaptés. C'est à cette étape de numérisation que nous nous intéresserons dans la ressource suivante.

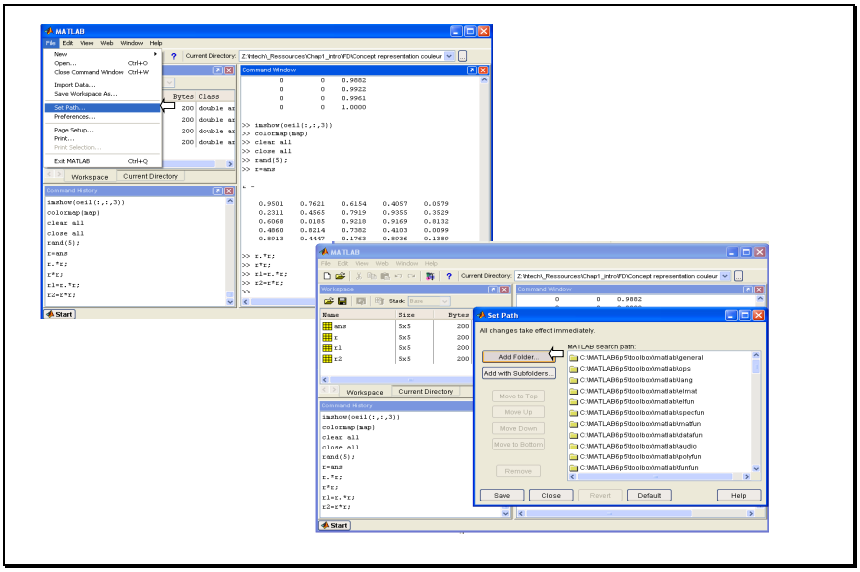
Exercice Chapitre 1– Prise en main du logiciel Matlab

MATLAB est un langage de calcul scientifique de haut niveau et un environnement interactif pour le développement d'algorithmes, la visualisation et l'analyse de données, ou encore le calcul numérique.

Lorsque vous lancez Matlab, vous disposez par défaut d'une console, d'un espace de travail (workspace) et d'un historique de commandes. Un éditeur est également accessible par la simple commande ***edit***.



ACTION : Lancez Matlab et ajoutez à la liste des chemins dans le navigateur de chemin (path browser) vos propres répertoires de travail (ne pas oublier d'ajouter les sous-répertoires). Les commandes Unix ***pwd, cd, ls*** sont disponibles.



*Remarque : lorsqu'un utilisateur lance une commande dans la console, Matlab va chercher dans le répertoire - que vous avez indiqué à travers le path browser - si une **fonction** ou un **script** correspond à cette commande, c'est alors le premier trouvé qui est utilisé (attention donc à l'ordre des répertoires et aux noms des scripts et des fonctions).*

Exercice 1 – Les opérateurs
Entre chaque exercice, il est conseillé de lancer les commandes ***clear*** et ***close all*** de manière à vider le workspace et à fermer toutes les figures.

1.1 – Entrez la commande *a=[1 4 5 ;4 8 9]*, à quoi cette commande correspond elle ?
Entrez dans la console la commande *a=rand(5)*. Puis entrez la commande ***help rand*** pour obtenir une description de l'opération effectuée par la fonction ***rand***. Enfin entrez la commande : *a=rand(5)* suivie d'un point virgule « *; »*
Quelle différence observe t'on dans la console avec la commande précédente *a=rand(5)* ? En déduire le rôle sous Matlab du « ; ».

1.2 – Sous Matlab, l'opérateur « : » est très utile. Il permet entre autres de balayer les éléments d'une ligne ou d'une colonne d'une matrice.
Mise en garde : L'indice 0 en Matlab n'existe pas. Le premier élément d'une matrice s'accède par l'indice 1. Par exemple, *matrice(1,1)* permet pour les images d'accéder à la valeur du pixel (1ère ligne, 1ère colonne). Le premier indice est pour les lignes, le second indice pour les colonnes.
Afin de bien assimiler ces concepts essayez les commandes :

- *a(2,1)*
- *a(1:3,1)*
- *a(:,2)*
- *a(1, :)*
- *a([1 4], :)*
- *a(1 :2 :5, :)* (la commande *1:2:5* permet de balayer l'intervalle [1, 5] par pas de 2)

Attention cependant à ne pas mettre de « ; » en fin de ligne pour observer les résultats obtenus dans la console.

1.3 – Matlab est un outil très intéressant pour le calcul matriciel. Les différentes opérations matricielles classiques (addition, multiplication, valeurs propres, …) y sont bien sur présentes, mais des opérations « élément par élément » très utiles en traitement d'images sont également disponibles par l'intermédiaire d'un opérateur pointé (exemple : « ***.*** », « ***./*** »).
Entrez les commandes suivantes (sans « ; » en fin de ligne pour observer les résultats) :

- *a = [0 0 0; 1 1 1; 2 2 2]*
- *b = a + 4*
- *c = a * b*
- *e = a .* b*

Expliquez la différence entre « ***c*** » et « ***e*** ».

ACTION : Créez une matrice *A* de taille 4×4 selon la méthode de votre choix (utilisez ***rand*** ou entrez les éléments un à un). Comment accéder à :

- la première ligne de *A* ?
- la quatrième colonne de *A* ?
- les trois premiers éléments de la quatrième ligne de *A* ?

Exercice 2 – L'affichage

Matlab offre aussi des possibilités d'affichage très diverses et pratiques. Il est par exemple possible de tracer des fonctions sur une échelle logarithmique ou de visualiser en image les valeurs d'une matrice.

2.1 – L'affichage d'un vecteur se fait par l'intermédiaire de la commande ***plot***. Entrez la commande ***help plot*** pour obtenir plus d'informations sur cette fonction et sur les fonctions qui ont des propriétés similaires (***subplot***, ***semilogx***, …).
Essayez les commandes suivantes :

- *x=1:3:10*
- *plot(x)* puis *plot(x,'r')*
- *y=rand(4,1)*, puis *plot(y)*, puis *plot(x,y,'g')*. Interprétez la différence de ces deux commandes.

La fonction ***plot*** est donc très utile pour obtenir par exemple les courbes de différentes fonctions planes.

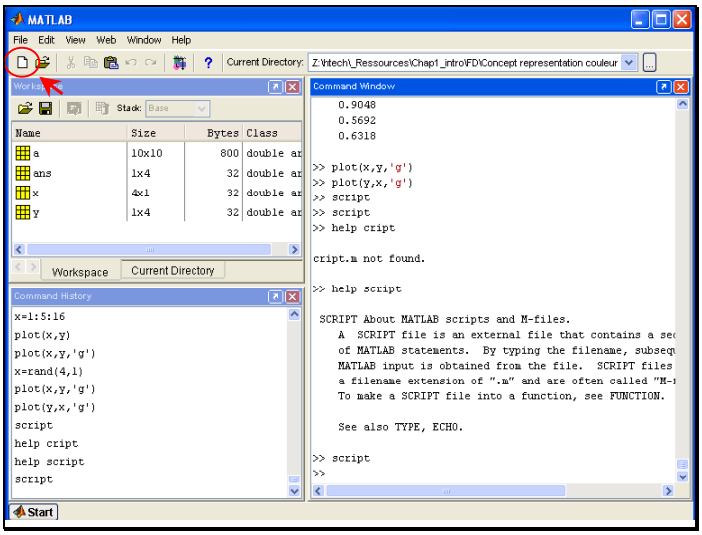
2.2 – L'affichage d'une matrice correspond lui à celui d'une image. Chaque élément (m, n) de la matrice est considéré comme étant la valeur du pixel (m, n) auquel il est associé. Vérifiez ceci en entrant les commandes suivantes :

- *a=rand(10)*10 ;* (pour que les éléments ne soient plus bornés à [0,1]),
- *a=exp(a) ;* (pour obtenir de plus grands écarts entre les éléments du vecteur a),
- *image(a)*

L'affichage des images peut se faire avec les fonctions ***image***, ***imagesc***, et ***imshow***. Essayez de zoomer à l'aide de la souris, quel type d'interpolation est utilisé ?

Exercice 3 – Ecriture de scripts

L'extension classique d'un fichier MATLAB est .m. On peut trouver 2 types de fichiers m : les fichiers de fonctions (abordés plus loin dans cet exercice) et les fichiers de script qui sont un ensemble de commandes pouvant s'enchaîner. L'utilisation de fichiers script permet de sauvegarder vos commandes d'une session Matlab à une autre. Pour ouvrir un fichier script :
- soit vous tapez la commande ***edit***,
- soit vous cliquez : file ⇒ new ⇒ M-file,
- soit vous cliquez directement sur l'icône représentant une page blanche.



Pour exécuter un script :

- soit vous lancez dans la fenêtre de commande la commande ***nom_du_fichier*** (sans l'extension .m), en vous assurant que la liste des chemins est cohérente ;
- soit vous sélectionnez des lignes du fichier .m dans la fenêtre d'édition et vous tapez la touche F9 (pratique pour exécuter une partie isolée du script).

ACTION : Entrez l'ensemble des commandes de la partie 1.2 dans un script que vous sauvegardez (exemple : ***test.m***). Lancez le script dans la console et par l'intermédiaire de la touche F9.

Exercice 4 – Type de données

Durant les séances d'exercices, nous serons amenés à utiliser la **toolbox *Image Processing***. Celle-ci contient un grand nombre de fonctions déjà écrites (n'hésitez pas à consulter l'aide via les commandes ***help*** ou ***helpwin***) ainsi que des démonstrations. Tantôt, nous écrirons nos propres fonctions tantôt nous utiliserons celles de la toolbox. Néanmoins, il faudra faire attention au type des données. En effet, classiquement et par défaut sous Matlab, tout est matrice de ***double***, or la plupart des fonctions de la toolbox *Image Processing* travaille avec le type ***uint8*** pour représenter les valeurs des pixels. Il faudra donc procéder au trans-typage chaque fois que cela est nécessaire (vous pouvez facilement voir le type de vos données dans le workspace).

ACTION : De la banque d'images, téléchargez l'image '*FRUIT_LUMI*' dans votre répertoire de travail. Lisez et affichez le fichier image respectivement à l'aide des commandes ***imread*** et ***imshow*** :

```
fruit = imread('FRUIT_LUMI.bmp');
imshow(fruit);
```

En consultant le workspace, observez la taille et le type de la donnée. De manière à afficher une sous-image correspondant au coin haut gauche de 64×64 pixels, testez la commande :

```
imshow(fruit(1:64,1:64));
```

La matrice *fruit* représente maintenant l'image '*FRUIT_LUMI*'. Essayez d'additionner directement cette matrice avec un nombre quelconque (ex : *fruit+8*). La console retourne le message d'erreur :

Function '+' is not defined for values of class 'uint8'

En effet, l'opérateur '+' est défini uniquement pour des éléments de type « double ». Pour effectuer cette opération, il est donc nécessaire de forcer le type « double » des éléments de la matrice en tapant : *double(fruit)*. Réessayez après ce transtypage.

Exercice 5 – Écriture de vos fonctions

Les fichiers de fonction sont à utiliser comme en programmation impérative classique. Ce sont également des *fichiers.m*. Pour utiliser une fonction il faut l'appeler avec des paramètres d'appels soit dans un script soit dans une autre fonction.

ACTION : Utilisez le fichier ***template.m*** pour écrire votre propre fonction ***max_min*** qui effectue la différence entre le plus grand et le plus petit élément d'une matrice. On pourra utiliser les fonctions Matlab ***max*** et ***min*** **.** Attention, il est nécessaire d'enregistrer le nom de votre fonction (exemple ***max_min*** **)** dans un fichier de même nom (exemple ***max_min.m*** **) .**

Corps du fichier ***template.m*** :

```
function[out1, out2] = template(arg1, arg2)

%-----------------------------------------
%
%    Description:
%
%    Entree :
%          arg1 :
%          arg2 :
%
%    Sortie :
%          out1 :
%          out2 :
%
%    Date :
%    Modif:
%-----------------------------------------
```

Remarque : L'utilisation d'une fonction peut s'effectuer directement au sein de la fenêtre de commandes (sous réserve des bons chemins !) ou encore au sein d'un script ou d'une autre fonction.

Correction de l'exercice sur la prise en main Matlab

Cet exercice a pour premier objectif de vous familiariser avec Matlab dans le cas d'une première prise en main, et de rappeler les concepts fondamentaux du logiciel Matlab à tous ceux qui l'avaient déjà utilisé.

1 – Les opérateurs

1.2 –

La commande *a=[1 4 5 ;4 8 9]* retourne la matrice 2×3 : $\begin{pmatrix} 1 & 4 & 5 \\ 1 & 8 & 9 \end{pmatrix}$.

La commande *a=rand(5)* retourne une matrice 5×5 composée de valeurs aléatoires comprises entre 0 et 1.
Enfin sous Matlab l'opérateur « ; » permet de ne pas afficher le résultat d'une commande dans la console. Cet opérateur est utile par exemple lorsqu'on travaille avec de grandes matrices (comme les images) pour lesquelles l'affichage est long et souvent peu représentatif de la donnée.

1.3 –

Manipulation de l'opérateur « : ».

1.4 –

Sous Matlab deux types d'opérations sur les matrices existent avec les opérateurs « * » et « / » :

- multiplication et division matricielle,
- multiplication et division élément par élément (utilisation conjointe avec l'opérateur « . ».

La commande *c=a*b* fera la multiplication matricielle de la matrice « a » par la matrice « b » : $c_{ij}=\sum_k a_{ik}.b_{kj}$.

La commande *e=a.*b* fera la multiplication élément par élément de la matrice « a » par la matrice « b » : $e_{ij}=a_{ij}.b_{ij}$. Les matrices doivent donc être de même taille.

ACTION :

Créons une matrice *A* de taille 4×4 en entrant directement les coefficients un à un : A=[1 2 3 4;5 6 7 8;9 10 11 12;13 14 15 16]

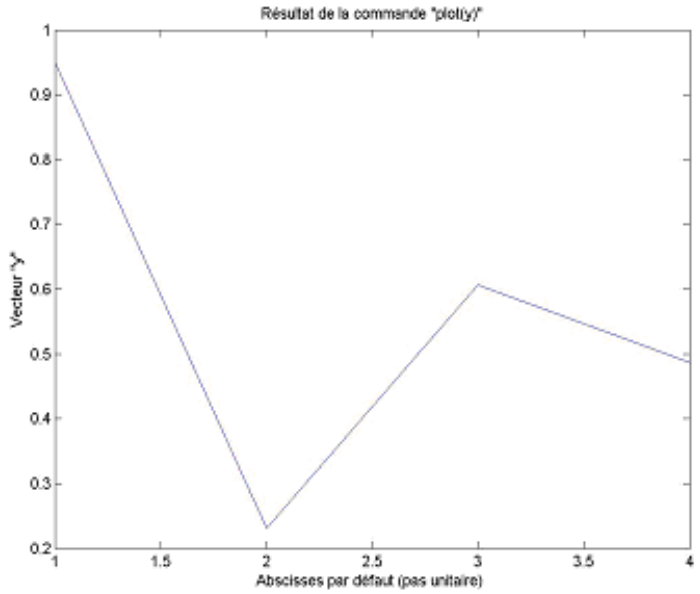
$$\mathbf{A}=\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix}$$

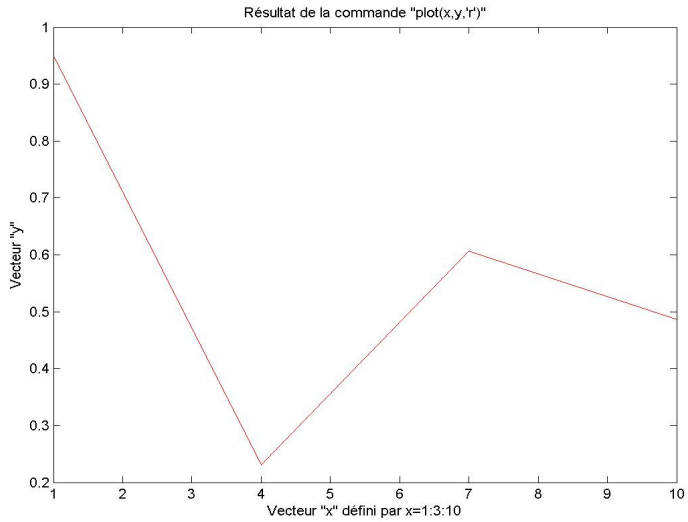
La 1ère ligne de A est donnée par la commande : *A(1,:)*.
La 4ème colonne de A est donnée par la commande : *A(:,4)*.
Les 3 premiers éléments de la 4ème ligne de A sont donnés par la commande : *A(4,1:3)*.

2 – L’affichage

2.1 –

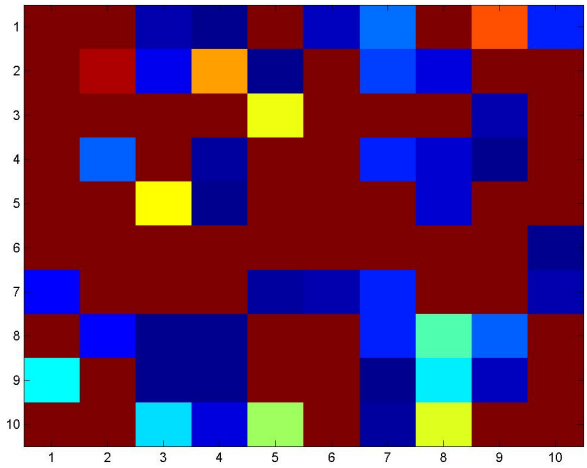
La commande *y=rand(4,1)* renvoie un vecteur de 4 éléments. En tapant *plot(y)*, une courbe apparaît. Cette courbe représente l’évolution du vecteur « y » en fonction des indices des éléments du vecteur. Il est cependant possible de modifier les abscisses en créant un vecteur d’abscisse « x » dans une unité voulue de même taille que le vecteur « y » : vous tracez y en fonction de x par la commande *plot(x,y)*.



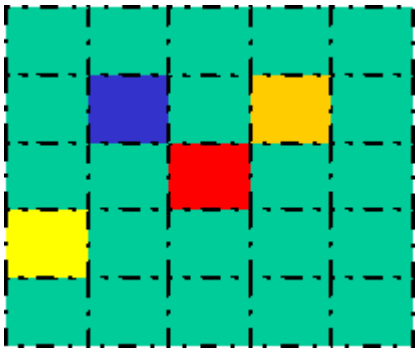


2.2 –

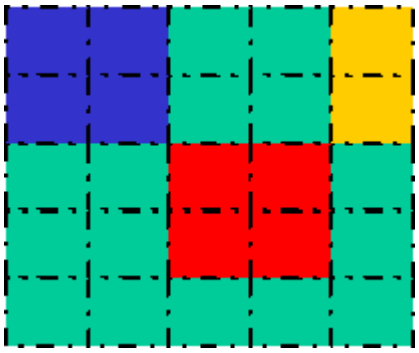
Voici un exemple de résultat obtenu en tapant les commandes indiquées avec un affichage par *image(a)*.



La matrice de taille 10×10 est ici représentée par 10×10 pixels carrés de couleurs différentes. Les couleurs correspondent aux valeurs des différents éléments de la matrice.
Le zoom utilise une interpolation d'ordre 0 dite du plus proche voisin. Il s'agit en fait d'une simple recopie d'un pixel image sur plusieurs pixels de l'écran d'affichage.
Considérons par exemple un écran de 5×5 pixels, sur lequel est représenté une image 5×5. Les pixels écran correspondent au quadrillage représenté.



La figure ci dessous présente un zoom ×2 autour du pixel rouge situé au centre dans le cas d'une interpolation du plus proche voisin.



Le pixel rouge est simplement « dupliqué » deux fois en hauteur et en largeur sur les pixels écran.

3 – Ecriture de scripts

Manipulation d’un script.

4 – Type de données

Manipulation de données uint8.

5 – Écriture de vos fonctions

La fonction « ***max*** » de matlab (resp. la fonction « ***min*** ») retourne pour une matrice M×N en entrée, un vecteur de taille N dont chaque élément e_k est l’élément maximal (resp. l’élément minimal) sur la colonne k de la matrice.

Voici la fonction solution :

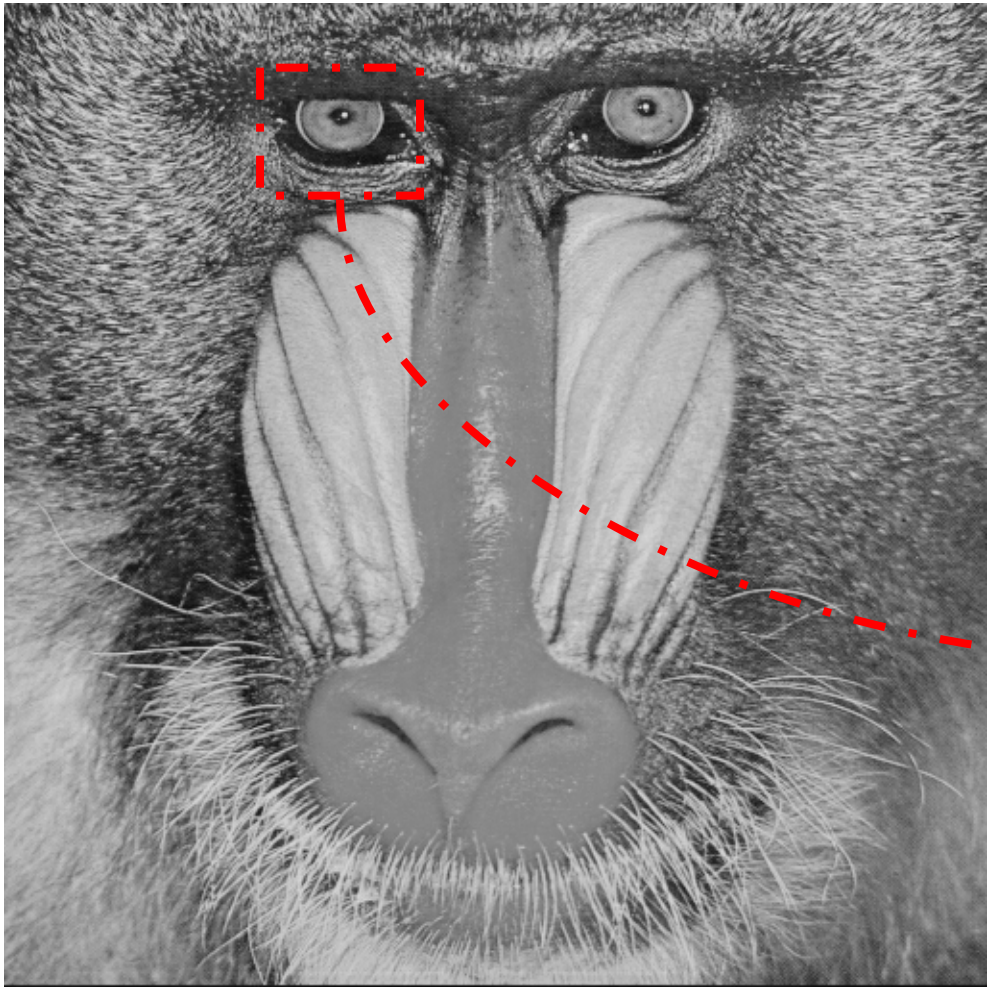
```
function[out] = max_min(A)

%-------------------------------------------
%     Description: Différence des éléments max et min
%     d'une matrice A correspondant à une image monochrome
%
%     Entree :
%           A : la matrice sur laquelle s'effectue
%           la recherche
%
%     Sortie :
%           out : la valeur de la différence
%-------------------------------------------

out = max(max(A))-min(min(A));
```

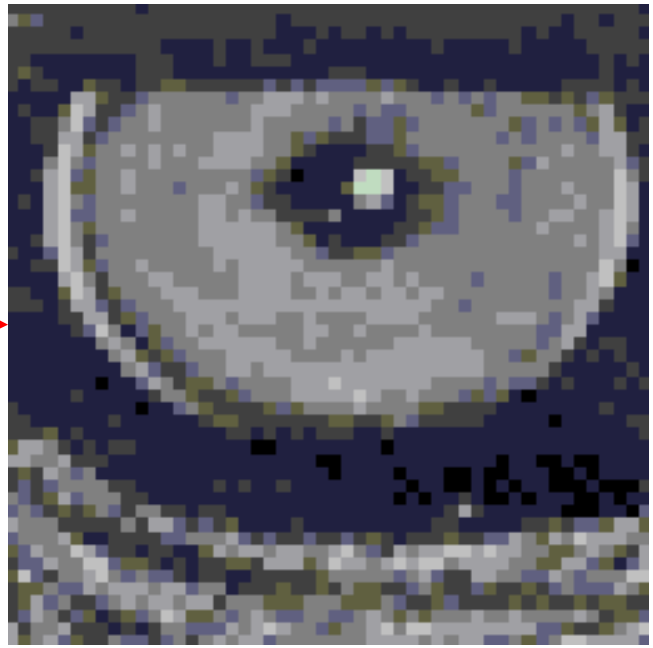
Comment se décompose une image en niveaux de gris?

Image monochrome Mandrill (512 × 512 pixels)



Zoom sur un bloc de 50 × 50 pixels

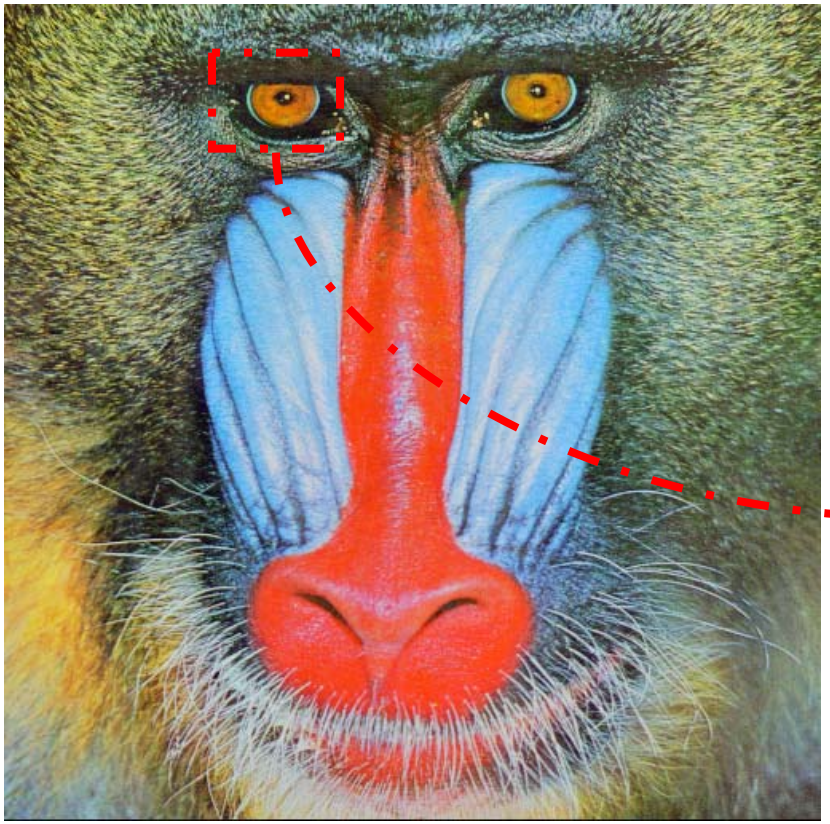
⇒ Mise en évidence des pixels avec différents niveaux de luminance (256 niveaux possibles)



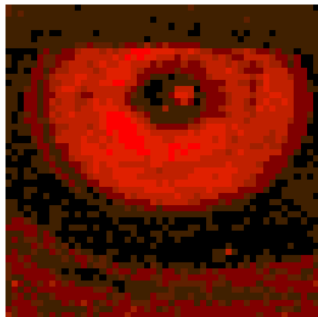
Comment se décompose une image en couleur?

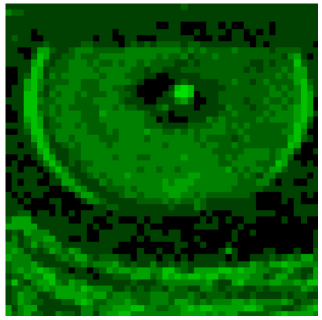
Image couleur Mandrill (512 × 512)

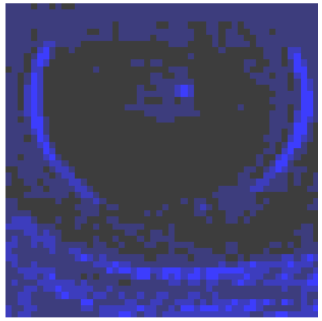
Blocs en annulant les plans complémentaires



bloc 50 × 50







- **Sur écran, la couleur est obtenue par synthèse additive des trois composantes : ROUGE - VERT - BLEU**

Note : **En reproduction sur papier, il s'agit d'une synthèse soustractive (composantes : Jaune, Cyan, Magenta + Noir).**

Exercice Chapitre 1 - Eclatement d'une image Couleur

Une image couleur est composée de 3 composantes. Nous avons déjà vu dans le chapitre 1 deux modes de représentation : la décomposition en Rouge, Vert, Bleu et la décomposition en Luminance, Chrominance1, Chrominance2 (il en existe d'autres). Nous allons voir la décomposition Rouge, Vert, Bleu sous Matlab.

Récupérez les images *CLOWN* et *CLOWN_LUMI*. Créez un répertoire et placez y ces images. Mettez à jour la liste des chemins dans le path browser.

1 – Différence entre une image monochrome et une image couleur

Créez deux variables « *im1* » et « *im2* » dans lesquelles vous chargez respectivement les images *CLOWN* et *CLOWN_LUMI* (utilisation de la fonction ***imread***). Visualisez ces deux images à l'aide de la fonction ***image***. Observez leur type et la taille des données associées, quelles différences observez vous ?

2 – Visualisation des plans Rouge, Vert, Bleu d'une image couleur

La variable « *im1* » est un tableau à trois dimensions de la forme N×M×P. De fait, pour une image couleur, N×M représente le nombre de pixels de l'image (N points par lignes, M lignes) et P représente les plans Rouge - Vert - Bleu. On aura donc pour une image couleur P=3 et pour une image en niveaux de gris P=1.
Créez 3 nouvelles variables *imr*, *imv*, et *imb* dans lesquelles vous chargez respectivement la composante rouge (P=1), la composante verte (P=2) et la composante bleue (P=3) de l'image *CLOWN*. Visualisez ces 3 plans.

Remarque importante: Pour visualiser les 3 plans RVB, qui sont chargés dans les variables *imr*, *imv*, et *imb*, vous aurez besoin d'une LUT adéquate (les LUT : « Look-up Table » seront présentées en détail dans le chapitre 2).
Trois LUT ont été créées et placées à cet effet dans les fichiers *lutr.m*, *lutv.m*, et *lutb.m* pour visualiser respectivement le plan rouge, le plan bleu et le plan vert. Rapatriez les fichiers *lutr.m*, *lutv.m*, et *lutb.m* dans le répertoire dans lequel vous travaillez. Après avoir affiché l'un des trois plans de l'image *CLOWN*, en prenant bien garde à sélectionner la fenêtre dans laquelle ce plan apparaît (fenêtre active pour Matlab), lancez dans la console la commande *lutr* pour le plan rouge, *lutv* pour le plan vert, et *lutb* pour le plan bleu.

Correction de l'exercice sur l'éclatement d'une image couleur

Cet exercice a pour but de vous montrer sur un exemple concret comment se réalise la synthèse additive Rouge Vert Bleu (RVB) pour une image couleur. Matlab utilise cette décomposition RVB pour l'affichage des couleurs et sera donc un outil idéal pour visualiser les différents plans.

1 – Différence entre une image couleur et une image monochrome

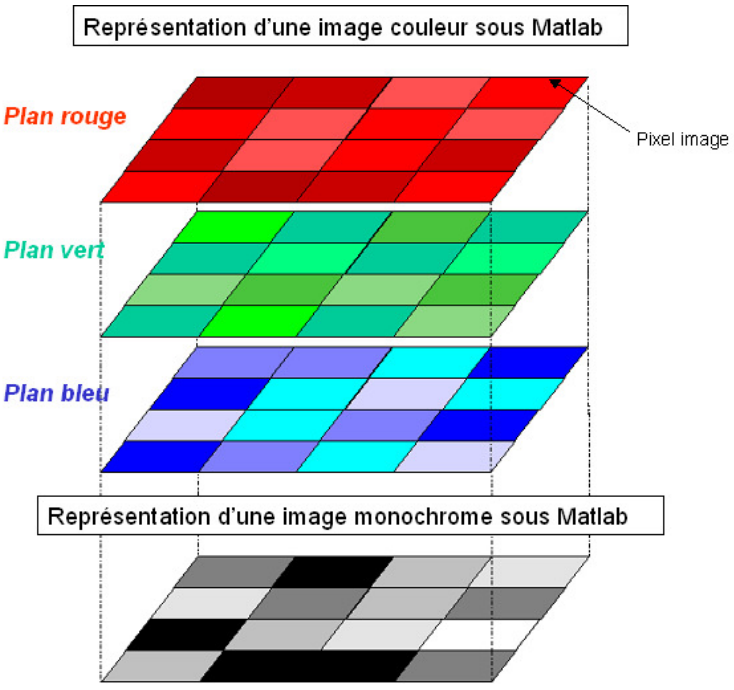
Pour acquérir les images *CLOWN* et *CLOWN_LUMI* dans les variables *im1* et *im2* il faut taper les commandes :

```
im1=imread('CLOWN.BMP') ;
im2=imread('CLOWN_LUMI.BMP') ;
```

Pour les afficher tapez:

```
image(im1) ;
image(im2) ;
```

En observant le workspace, il apparaît que *im1* (image couleur) est un tableau à trois dimensions (512×512×3) alors que *im2* (image monochrome) est un tableau à deux dimensions seulement (512×512). La taille en octet de *im1* (786432 octets) est également trois fois plus grande que celle de *im2* (262144 octets).
Les deux premières dimensions des tableaux *im1* et *im2* représentent le nombre de points par ligne et le nombre de lignes pour l'image chargée. L'image *CLOWN* quelle soit affichée en niveaux de gris ou en couleur est une image de 512×512 pixels, il est donc normal que pour les variables *im1* et *im2* les deux premières dimensions soient identiques. La troisième dimension de *im1* indique que l'on est sur le plan Rouge, Vert, ou Bleu. Cette dimension est donc égale à 3. Pour une image monochrome, représentée uniquement par des niveaux de gris variant sur [0, 255], cette composante vaut 1 et n'est donc pas indiquée dans le workspace. Une image couleur correspond donc à la superposition de trois images monochromes RVB.



2 – Visualisation des plans Rouge, Vert, Bleu d'une image couleur

Voici le script pour acquérir les trois plans RVB dans les variables *imr*, *imv*, et *imb* :

```
imr=im1(:,:,1);      % Plan Rouge
imv=im1(:,:,2);      % Plan Vert
imb=im1(:,:,3);      % Plan Bleu
```

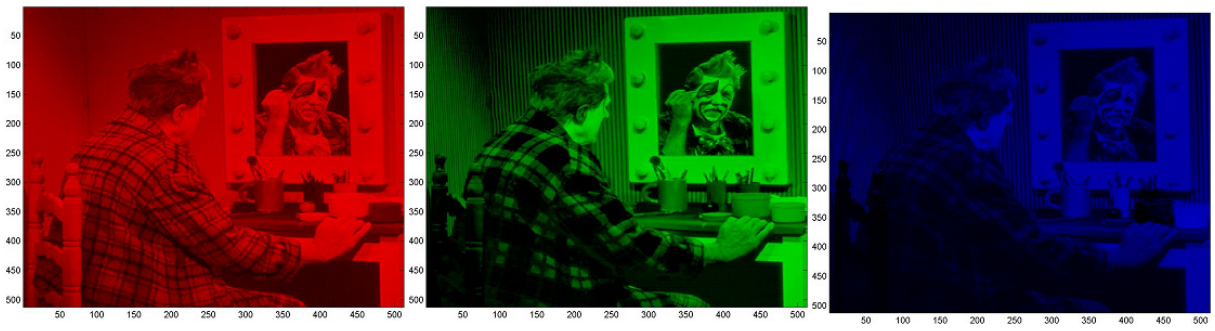
Après avoir rapatrié les fichiers *lutr*, *lutb*, et *lutv* dans votre répertoire de travail, voici les scripts pour afficher les trois plans de l'image:

```
% Affichage du plan Rouge
figure      % Crée une nouvelle fenêtre pour l'affichage d'une image
image(imr);
lutr;

% Affichage du plan Vert
figure
image(imv);
lutv;

% Affichage du plan Bleu
figure
image(imb);
lutb;
```

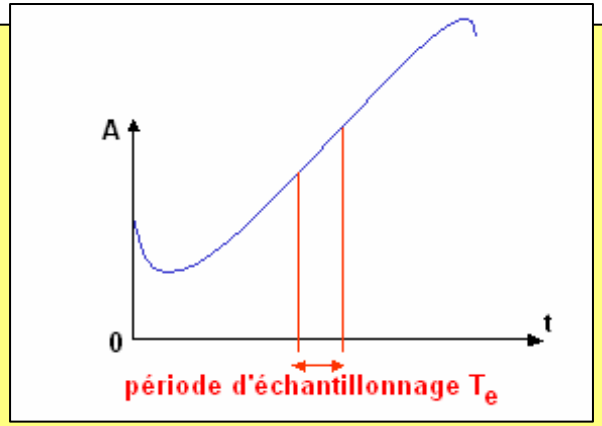
Voici les résultats obtenus :



D'un signal continu (analogique) à un signal discret (numérique): 3 étapes

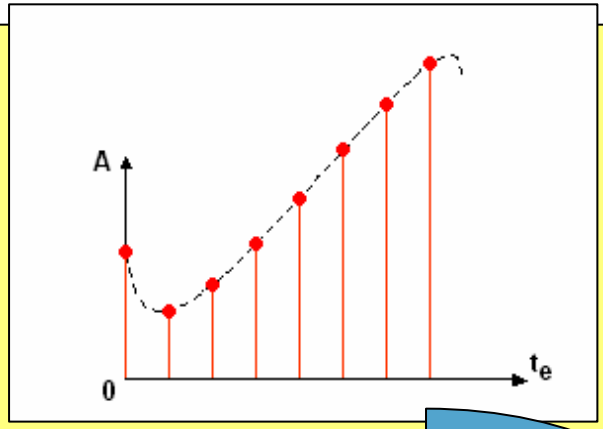
Chaîne de numérisation d'un signal temporel (1-D)

Signal continu (référence)



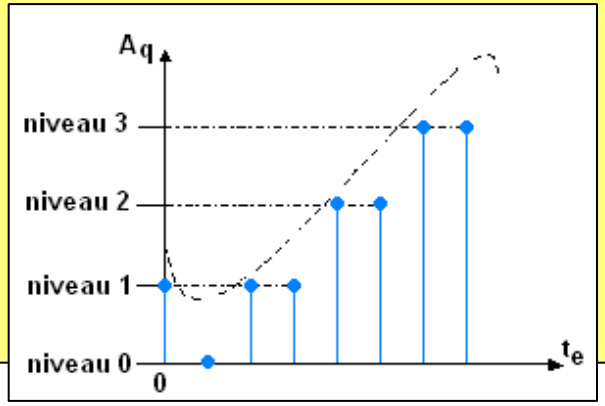
Étape 1

Signal échantillonné



Étape 2

Signal quantifié (4 niveaux)



Étape 3

Codage (4 niveaux ⇒ 2bits)

	CODAGE	
	Bit 2	Bit 1
niveau 3	1	1
niveau 2	1	0
niveau 1	0	1
niveau 0	0	0

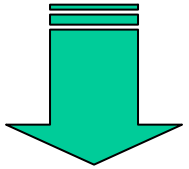
D'un signal continu (analogique) à un signal discret (numérique): 3 étapes

➢ **Échantillonnage :** l'évolution du signal suivant la dimension « t » (ici le temps) est représentée par un nombre fini de ses valeurs. Les valeurs du signal sont prises régulièrement à une période d'échantillonnage T_e.

➢ **Quantification :** l'amplitude du signal échantillonné est représentée par un nombre fini de valeurs d'amplitude (niveaux de quantification).

➢ **Codage :** les niveaux de quantification sont codés sous la forme d'un mot binaire sur k bits ($\Rightarrow 2^k$ niveaux possibles).

CAS DES IMAGES



➢ Les images sont des signaux 2-D, l'échantillonnage se fait selon les dimensions spatiales « x » et « y » (et non pas selon le temps comme précédemment).

➢ Le nombre de niveaux de quantification de la luminance est généralement de 256, chaque niveau étant alors codé sur 8 bits (code binaire naturel).

Chapitre 1

TRAITEMENT DE SIGNAL MULTIMEDIA

Numérisation vidéo

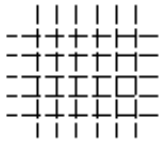
Échantillonnage et quantification

Échantillonnage et Quantification

• Les systèmes de traitement numériques des images manipulent des données de type « discrètes »

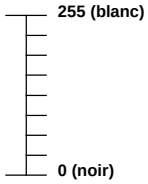
• Échantillonnage

- Échantillonnage des valeurs de l'image aux nœuds d'une grille rectangulaire disposée dans le plan de l'image
- Un pixel (m, n) (*picture element*) représente la valeur du niveau de gris de l'image monochrome au point de la grille indexé par les coordonnées entières (m, n)



• Quantification

- Transformation d'une amplitude à valeurs continues en une amplitude qui ne peut prendre qu'un nombre fini de valeurs distinctes
- Pour une image à 256 niveaux de gris, chaque amplitude est codée sur un octet (8 bits)



Nous avions brièvement abordé au cours de la ressource précédente (Traitement d'images : Exemples d'applications), le fait que la numérisation d'une image suppose une double discrétisation :

- D'une part du domaine de définition de l'image : le domaine spatial 2-D pour les images fixes (respectivement le domaine spatial 2-D et le temps pour les images animées) : il s'agit de l'***échantillonnage***.
- D'autre part de l'amplitude du signal image : du signal de luminance dans le cas des images monochromes, des trois signaux « couleur » dans le cas des images couleurs (en représentation « Rouge, Vert, Bleu » : RVB ou « Luminance, Chrominance1, Chrominance2 » : Y, Cr, Cb) : il s'agit de la ***quantification***.

On parlera donc d'images numériques (ou numérisées) seulement dans le cas où cette double discrétisation aura été effectuée : en espace et en amplitude. La représentation de base (appelée aussi canonique) d'une image correspondra donc à un tableau 2-D, dont chaque élément correspond à un pixel. Dans le cas d'une image monochrome, chaque pixel sera codé par exemple sur 8 bits et pourra donc ainsi prendre 256 valeurs différentes (effet de la quantification). Dans le cas d'une image couleur, le pixel aura trois composantes qui représenteront suivant le modèle choisi : les composantes (R, V, B) ou les composantes (Y, Cr, Cb), selon les cas.

Remarque : si R, V, B prennent chacun leur valeur sur [0, 255], et Y aussi, les deux composantes de chrominance Cr et Cb prennent leurs valeurs au départ sur [-128, +127]. En pratique, on les code en ajoutant un offset de 128, d'où une dynamique également de [0, 255].

Scénario typique d'un Traitement d'Image

Rappel : Qu'est ce qu'une Image?

– Une fonction I(x,y) à deux coordonnées spatiales du plan

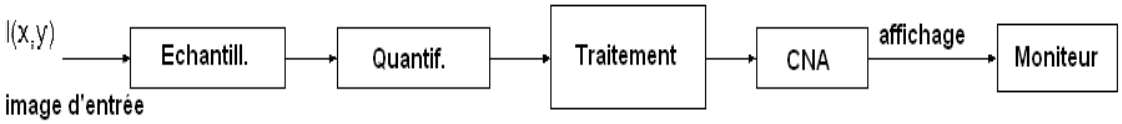
– Pour obtenir des données utilisables en traitement numérique :

• *Échantillonnage (spatial) et Quantification des valeurs de luminance*

• *Peut être effectué par des capteurs*

e.g. Charge-coupled Device (capteur CCD)

Scénario typique du traitement d'image



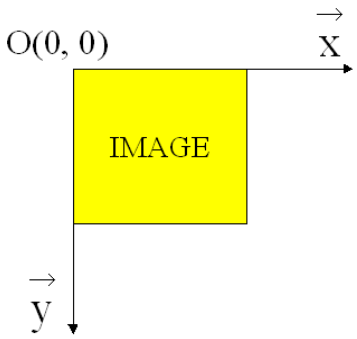
L'image ci dessus présente la chaîne complète utilisée pour effectuer un traitement d'image quelconque :

- L'image est représentée par une fonction de deux variables, ces variables correspondent aux coordonnées spatiales (x, y).
- La première fonction est *Echantill* : on échantillonne le signal d'image (discrétisation des coordonnées spatiales).
- Le résultat est ensuite injecté dans la fonction *Quantif* : très souvent pour une image monochrome, on fixe 256 niveaux de luminance.
- L'image est maintenant numérisée, on peut donc appliquer le traitement d'image (*cf. Traitement*) que l'on souhaite : binarisation, augmentation du contraste, …
- L'image numérique traitée est finalement envoyée vers un convertisseur Numérique/Analogique (CNA) pour obtenir un signal affichable sur un moniteur.

La chaîne fonctionnelle typique d'un traitement d'image nécessite donc un traitement pour numériser les images que l'on souhaite ensuite traiter. Nous avons vu que ce traitement correspond à un échantillonnage suivi d'une quantification. Ces deux notions ont pu être abordées en traitement du signal et en électronique. Cependant, nous allons voir que la conception et la réalisation de ces deux étapes en imagerie est sensiblement différente notamment parce que les images sont des signaux 2-D au niveau spatial et que leur représentation en image couleur nécessite de considérer trois numérisations différentes.

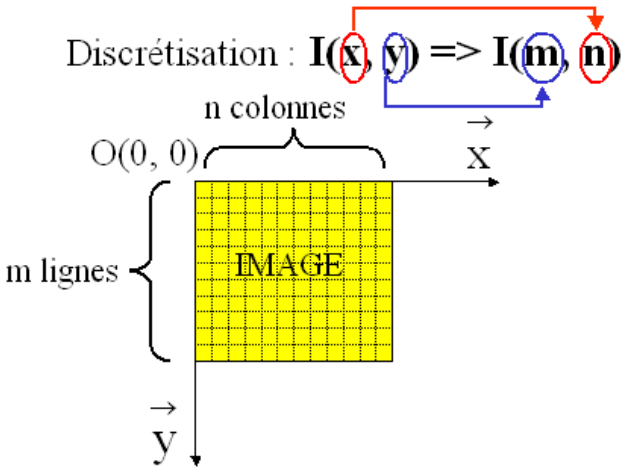
Deux remarques importantes :

1 – Le signal d'entrée I(x, y) est une image 2-D. En image, les axes (Ox) et (Oy) sont représentés et orientés comme sur la figure suivante :



L'axe (Oy) est donc dirigé du haut vers le bas de l'image.

2 – Une fois ces deux axes discrétisés, on obtient « m » lignes et « n » colonnes :

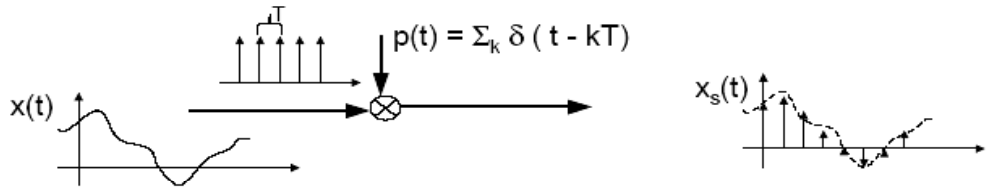


En traitement d'images, **on parle alors d'un signal d'image I(m, n)** et non pas I(n, m). **L'ordre des coordonnées est donc modifié** puisque « m » correspond à la discrétisation selon (Oy) et « n » selon (Ox).

Rappel : Échantillonnage 1-D

- Domaine temporel

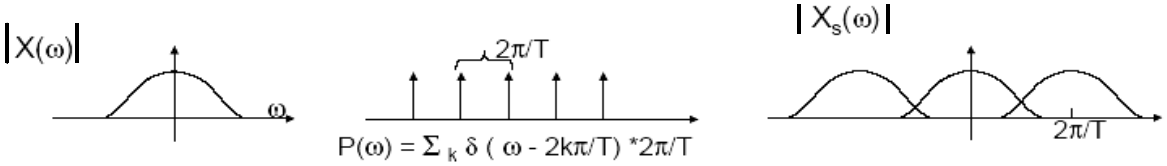
  - Multiplication du signal continu par un train d'impulsion



- Domaine fréquentiel

  - Dualité : échantillonner dans un domaine ⇔ rendre le signal périodique dans l'autre domaine

    - la Transformée de Fourier d'un train d'impulsion est un train d'impulsion



Avant de pouvoir passer à l'échantillonnage d'un signal 2-D, il est nécessaire de faire quelques rappels sur l'échantillonnage des signaux 1-D. Le type d'échantillonnage le plus répandu en électronique et en traitement du signal est l'échantillonnage temporel. La figure ci-dessus présente les différentes conséquences engendrées par un tel traitement dans les domaines temporels et fréquentiels.

Note : $\omega=2\pi f$, avec ω : pulsation en rad.s^{-1} et f : fréquence en Hz

- Dans le domaine temporel : en partant d'un signal « x » dépendant du temps, il faut choisir une période d'échantillonnage « T ». Le signal échantillonné $x_S(t)$ sera donc représenté par ses valeurs aux instants t = T, t = 2T, t = 3T, … Pratiquement pour réaliser un tel échantillonnage, il faut réaliser le produit du signal analogique x(t) avec un train d'impulsion p(t). Ce train d'impulsion correspond à la somme d'impulsion de Dirac aux instants kT, k étant un entier naturel.

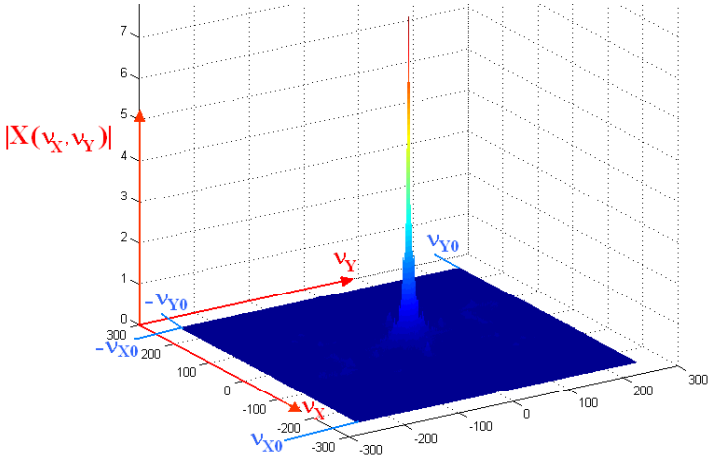
- Dans le domaine fréquentiel : il y'a une dualité entre le domaine temporel et le domaine fréquentiel. En effet, le fait d'échantillonner un signal dans l'un des domaines entraîne la périodisation de ce même signal dans l'autre domaine. Ici X(ω) (resp. P(ω)) représente le spectre fréquentiel de x(t) (resp. p(t)). L'échantillonnage de période T dans le domaine temporel a donc entraîné la périodisation en fréquence (période 1/T) du spectre dans le domaine fréquentiel (voir représentation de $X_S(\omega)$).

Cette dualité entre les deux domaines peut engendrer des effets de repliement dans le spectre du signal échantillonné, certaines contraintes sont donc à satisfaire.

Extension à l'échantillonnage 2-D (1)

- Signal 2-D à bande limitée
  - sa transformée de Fourier est nulle en dehors de la région de fréquence spatiale : $|\nu_X| < \nu_{X0}$ et $|\nu_Y| < \nu_{Y0}$

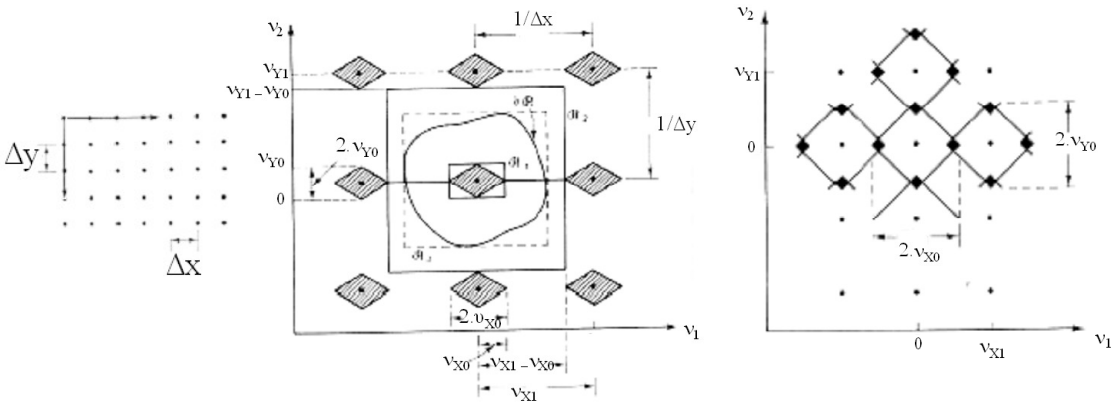
Exemple : l'image *Barbara* et son spectre



Les problèmes posés par l'échantillonnage des signaux bidimensionnels sont a priori similaires à ceux des signaux monodimensionnels. Le critère de Nyquist s'applique de la même manière. Toutefois, une interprétation dans un espace à deux dimensions est nécessaire pour mieux comprendre les phénomènes liés à l'échantillonnage, comme les effets de la géométrie du motif d'échantillonnage.
La figure ci dessus représente un exemple de transformée de Fourier d'une fonction 2-D à bande limitée (image *Barbara*). On peut dans ce cas travailler avec un spectre borné. Le spectre s'étale sur un plan fréquentiel défini par deux vecteurs de base.

Extension à l'échantillonnage 2-D (2)

- Train d'impulsion 2-D : fonction comb
  - $\mathrm{comb}(x, y; \Delta x, \Delta y) = \sum_{m,n} \delta(x - n.\Delta x, y - m.\Delta y)$
  - $\mathrm{FT} \Rightarrow \mathrm{COMB}(\nu_X, \nu_Y) = \mathrm{comb}(\nu_X, \nu_Y; 1/\Delta x, 1/\Delta y)/\Delta x.\Delta y$
- Échantillonnage et Réplication
  - Critère de Nyquist ($2.\nu_{X0}$ et $2.\nu_{Y0}$) et Aliasing



Les variations selon $\vec{x}$ (Δx) et les variations selon $\vec{y}$ (Δy) vont définir le plan dans lequel se trouve l'image 2-D. La structure d'échantillonnage est régulière (figure de gauche) donc l'arrangement des points échantillonnés (pixels) est régulier dans le plan (x, y). La position du point de rang (m, n) est donc donnée par : nΔx + mΔy.

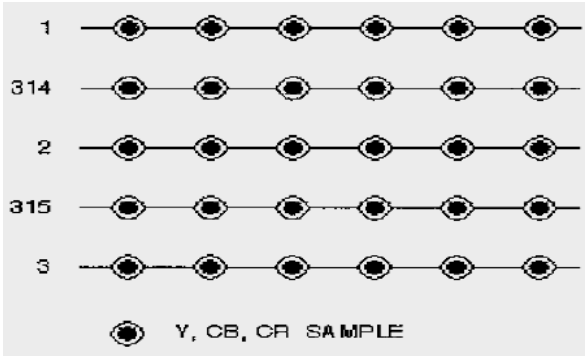
Dans le domaine fréquentiel, les points sont définis par leurs coordonnées « fréquentielles » ν_X et ν_Y. En prenant une fonction *f*(x,y), à laquelle on associe le spectre d'amplitude $F(\nu_X,\nu_Y)$, après échantillonnage on aura :

- $f_e(x,y) = f(x,y).\sum_m \sum_n \delta(x - n\Delta x, y - m\Delta y)$
- $F_e(\nu_x, \nu_y) = \left(\frac{1}{\Delta x.\Delta y}\right) \sum_m \sum_n F(\nu_x - n\Delta\nu_x, \nu_y - m\Delta\nu_y)$

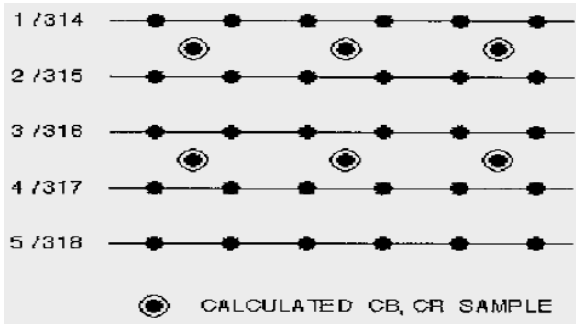
Pour un signal d'image à spectre borné, on peut alors retrouver l'image d'origine (non échantillonnée) à partir de l'image échantillonnée idéalement si les spectres du signal d'origine ne se chevauchent pas (figure centrale en opposition avec l'image de droite). Pour cela on utilise un interpolateur R de gain fréquentiel constant (fréquences spatiales), égal à ΔxΔy sur le domaine de F et nul à l'extérieur de ce domaine, tel que celui-ci n'ait aucun recouvrement avec les translatés de F. On a alors un interpolateur linéaire bidimensionnel idéal : c'est le filtre d'interpolation spatiale de Nyquist.

Structures d'échantillonnage en images et vidéos couleur

Structure 4 : 4 : 4



Structure 4 : 2 : 0



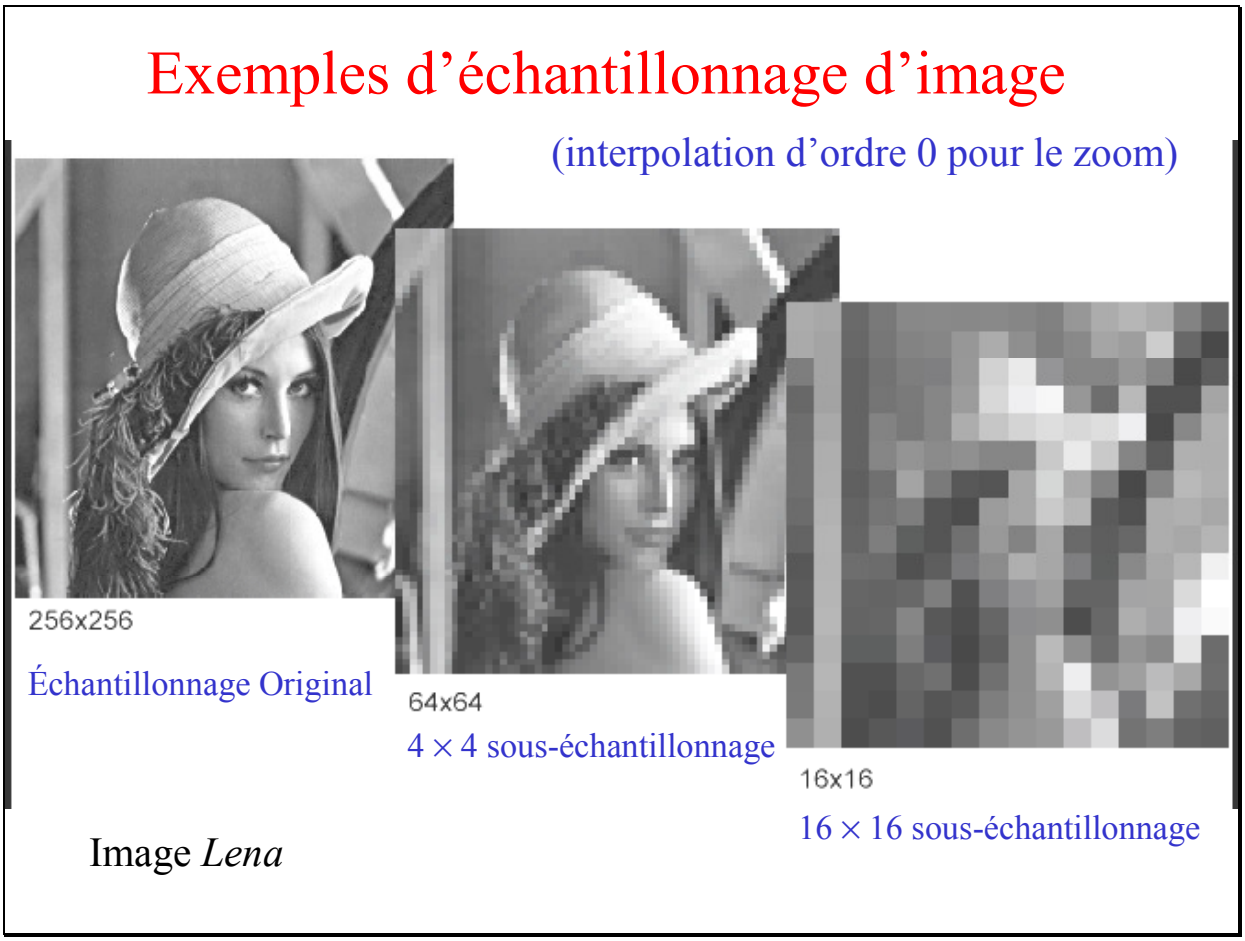
Structure d'échantillonnage orthogonal

La figure ci-dessus présente deux structures usuelles d'échantillonnage orthogonal d'images couleurs :

- Une structure 4 : 4 : 4 : Pour chaque pixel, on dispose de la composante de luminance et des deux composantes de chrominance Cr et Cb. Il n'y a donc dans ce cas aucun sous-échantillonnage des composantes de chrominance.

- Une structure 4 : 2 : 0 : Pour chaque pixel, on dispose d'une composante de luminance (Y) et seulement des deux composantes de chrominance Cr et Cb pour un groupe de 2×2 pixels de luminance. La résolution de la chrominance est divisée d'un facteur 2 suivant les deux dimensions spatiales. Ce format (utilisé sur les DVD) réduit les résolutions verticale et horizontale.

Pour ces structures d'échantillonnage, nous sommes passés au préalable au sous échantillonnage de la représentation RVB de l'image à la représentation Y, Cr, Cb. La réduction de la fréquence d'échantillonnage spatiale des deux composantes de chrominance est rendue possible du fait que, dans les images naturelles, les deux composantes ont une largeur de bande en fréquences spatiales, beaucoup plus réduite que celle de la luminance.



Sur cette figure, on a représenté l'image « *Lena* » échantillonnée avec deux structures d'échantillonnage différentes. L'image à gauche est l'image de référence (dimensions spatiales : 256×256 pixels). La seconde image est échantillonnée avec une cadence d'échantillonnage quatre fois moindre suivant chacune des deux dimensions spatiales. Elle est donc de dimension 64×64 pixels. Elle est cependant ramenée, pour des besoins de visualisation, à la même grandeur que l'image d'origine grâce à un zoom. Celui-ci est en fait une interpolation d'ordre 0 (chaque pixel est dupliqué 16 fois pour former à l'affichage sur l'écran un carré de 4×4 pixels identiques). Un effet de pixellisation apparaît clairement, mais il est encore possible de distinguer les différents éléments de l'image (le visage, le chapeau, …). La troisième image (à droite) présente le résultat après un sous-échantillonnage de 16 par rapport à l'image de référence suivant les deux directions spatiales. Toujours pour visualiser à la même taille, on a utilisé un zoom d'un facteur 16 (interpolation d'ordre 0 engendrant pour chaque échantillon une reconstruction par 16×16 pixels de même amplitude). L'image est alors très fortement pixélisée et on ne distingue quasiment plus les objets présents dans la scène.

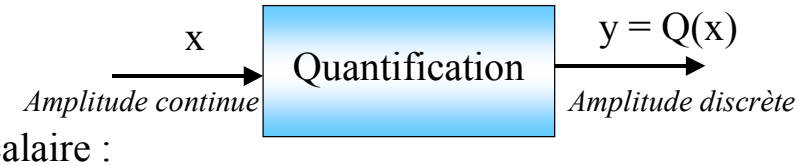
Après s'être intéressé à l'échantillonnage bidimensionnel d'un signal 2-D (dans le cas présent une image), il faut maintenant - pour obtenir l'image numérisée finale - déterminer comment procéder à la quantification de l'image échantillonnée.

Numérisation vidéo

Quantification

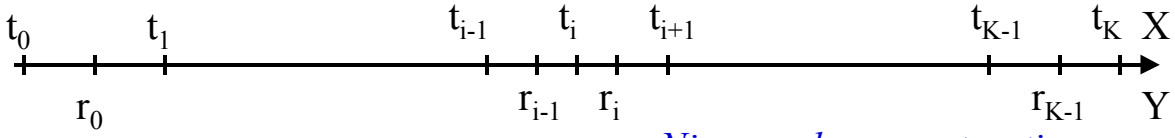
Quantification

- **Quantification Scalaire**



- Quantification scalaire :

Seuils de décision



Niveaux de reconstruction

- **Fonction de Quantification**

pour $t_j \leq x < t_{j+1}$ alors $y = r_j$ $\forall j = 0, \ldots, K-1$

et $t_0 = x_{min}$; $t_K = x_{max}$

L'image est à présent échantillonnée dans le domaine spatial. Elle est composée d'un ensemble de pixels. Il convient maintenant de réaliser une quantification pour que chaque amplitude du signal d'image soit représentée par une valeur approchée. On choisit donc un ensemble de valeurs prédéfinies appelées niveaux de reconstruction, reparties sur la dynamique $[x_{min}, x_{max}]$ du signal x à quantifier.

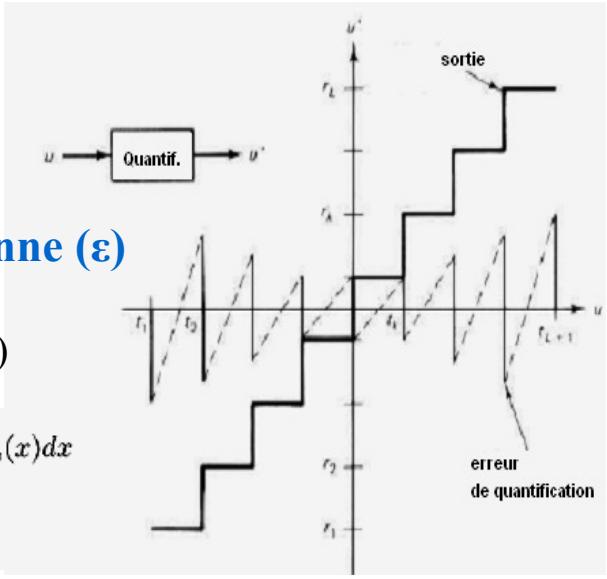
Sur la figure ci-dessus, ces niveaux de reconstruction sont définis par la suite $(r_i)_{i\in[0,K-1]}$ constituée de K valeurs. Ces niveaux sont accompagnés de seuils de décision définis par la suite $(t_i)_{i\in[0,K]}$ avec $t_0 = x_{min}$ et $t_K = x_{max}$. Prenons par exemple le cas d'un signal d'amplitude continue « x » en entrée du quantificateur, le signal quantifié « y » en sortie sera défini par la condition:

$$\forall i \in [0..K-1], \text{ si } t_i \leq x < t_{i+1}, \text{ alors } y = r_i \text{ et } t_0 = x_{min} ; t_K = x_{max}$$

Il existe de multiples façons de quantifier. En effet, la manière de choisir les niveaux de reconstruction et les seuils de décision n'est pas forcément de type répartition uniforme sur $[x_{min}, x_{max}]$, et dépend fortement des caractéristiques du signal d'image à quantifier.

Exemple de loi de quantification

- Multiple-to-one mapping
- Besoin de minimiser l'erreur



- **Erreur quadratique moyenne (ε)**
  - Cas d'une entrée « u » avec une densité de probabilité $p_u(x)$

$$\mathcal{E} = E[(u-u')^2] = \int_{t_1}^{t_{L+1}} (x-u'(x))^2 p_u(x)dx$$
$$= \sum_{i=1}^{L} \int_{t_i}^{t_{i+1}} (x-r_i)^2 p_u(x)dx$$

Le graphique ci-dessus donne un exemple de loi de quantification. L'entrée du quantificateur correspond au signal « u », et la sortie au signal « u' ». Les seuils de décision sont donnés par la suite $(t_i)_i$ et les niveaux de reconstruction sont donnés par la suite $(r_i)_i$. La quantification est un traitement qui engendre des pertes irréversibles contrairement à certaines formes de l'échantillonnage vu précédemment. Cependant, selon le paramétrage de la quantification, l'œil humain ne détectera pas forcément ces erreurs. Il faut donc essayer de les minimiser. Typiquement, on cherche à minimiser l'erreur quadratique moyenne. Par définition cette erreur quadratique ε^2 est donnée par la relation :

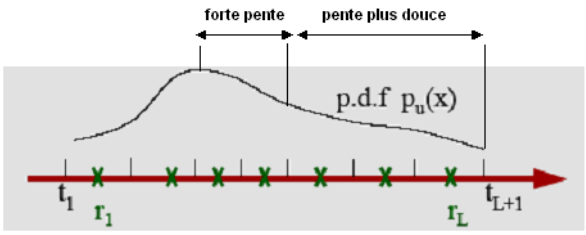
$$\varepsilon^2 = E\left[(u-u')^2\right] = \int_{t_1}^{t_{L+1}} (x-u'(x))^2 . p_u(x).dx$$

Où : E désigne l'espérance mathématique et $p_u(x)$ la densité de probabilité de u.
En décomposant l'intégrale suivant les relation de Chasles, on obtient finalement :

$$\varepsilon^2 = \sum_{i=1}^{L} \int_{t_i}^{t_{i+1}} (x-r_i)^2 . p_u(x).dx$$

La quantification doit donc être choisie de façon à minimiser l'erreur quadratique moyenne, mais nous allons voir que d'autres critères liés aux caractéristiques de l'image entrent également en compte.

Quel type de loi de quantification utiliser?



- Allouer plus de valeurs dans les domaines de valeurs les plus probables
- Minimiser l'erreur au sens des probabilités

- EQM minimale (MMSE en anglais) :

$$\begin{aligned}\mathcal{E} &= E[(u-u')^2] = \int_{t_1}^{t_{L+1}} (x-u'(x))^2 p_u(x)dx \\ &= \sum_{i=1}^{L}\int_{t_i}^{t_{i+1}} (x-r_i)^2 p_u(x)dx\end{aligned}$$

  - Grande EQM $\Rightarrow$ Fortes pertes
  - la minimalisation de l'EQM passe par l'utilisation d'outils mathématiques

- Problème à résoudre lors d'une optimisation

- Quels $\{t_k\}$ et $\{r_k\}$ utiliser? Loi de quantification optimisée si : dérivées partielles de l'EQM égales à 0

La question posée ici est de savoir quels sont les critères à prendre en compte pour bien choisir une loi de quantification. La figure ci-dessus représente un quantificateur Q dont l'entrée est le signal « u », et la sortie le signal « u' ». Les seuils de décision sont donnés par la suite (t_i) et les niveaux de reconstruction sont donnés par la suite (r_i). La fonction de densité de probabilité $p_u(x)$ est également représentée. Dans cet exemple le quantificateur n'est plus linéaire : les seuils et les niveaux de reconstruction ne sont pas disposés régulièrement. En effet, en observant attentivement la densité de probabilité de « u », on s'aperçoit que la pente de la fonction est plus faible (ou plus forte) pour certaines valeurs de t_i que d'autres. Lorsque la pente est faible, on peut raisonnablement penser qu'il faut peu de niveaux de reconstruction pour représenter la plage de valeurs décrite. Inversement, quand la pente de la courbe devient plus raide, de nombreuses valeurs sont atteintes en peu de temps, et il faut donc choisir beaucoup plus de niveaux de reconstruction. D'autre part, nous avons vu que pour choisir la loi de quantification il fallait également minimiser l'erreur quadratique moyenne dont l'expression dépend directement des paramètres (t_i) et (r_i). L'optimisation de la quantification passe donc par le choix des seuils de décision et des niveaux de reconstruction. Ces paramètres peuvent être directement déterminés en minimisant l'erreur quadratique moyenne (calcul de dérivées partielles, …).

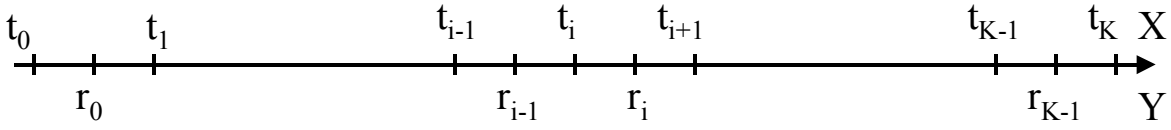
Quantification Optimale (1)

- Critère de quantification optimale
  - Erreur Quadratique Moyenne (y = Q(x))

$$\varepsilon^2 = E\{(X-Y)^2\} = \int_{-\infty}^{+\infty} (x-y)^2 \, p_X(x)\, dx$$

$$\varepsilon^2 = \sum_i \int_{t_i}^{t_{i+1}} (x-y_i)^2 \, p_X(x)\, dx$$

minimisation de ε^2: $\begin{cases} t_{i+1} = (r_i + r_{i+1}) / 2 \\ r_i = E\{X / x \in [t_i , t_{i+1}]\} \end{cases}$



- **Cas spécial** : fonction de densité de probabilité uniforme : $p_X(x)$ est constant alors $t_{i+1} = (r_i + r_{i+1}) / 2$ et $r_i = (t_i + t_{i+1}) / 2$

Le critère de quantification optimale est lié à la minimalisation de l'erreur quadratique moyenne. L'entrée du quantificateur correspond au signal X, la sortie au signal Y. Les niveaux de reconstruction sont donnés ici par (r_i), et les seuils de décision par (t_i). Nous allons donc chercher à déterminer les relations qui découlent de ce critère d'optimisation et de minimalisation:

Pour minimiser ε^2, on calcule ses dérivées partielles :

- Par rapport à t_i : $\frac{\partial \varepsilon^2}{\partial t_i} = p_X(t_i).(t_i - r_{i-1})^2 - p_X(t_i).(t_i - r_i)^2$
- Par rapport à r_i : $\frac{\partial \varepsilon^2}{\partial r_i} = -2.\int_{t_i}^{t_{i+1}} p_X(x).(x - r_i).dx$

En annulant la première dérivée partielle, on obtient la relation : $t_{i+1} = \frac{r_i + r_{i+1}}{2}$

En annulant la seconde dérivée partielle, on obtient :

$$\int_{t_i}^{t_{i+1}} p_X(x).(x - r_i).dx = 0 \Leftrightarrow \int_{t_i}^{t_{i+1}} p_X(x).x.dx = \int_{t_i}^{t_{i+1}} p_X(x).r_i.dx$$

$$\Leftrightarrow E\{X; x \in [t_i, t_{i+1}[\} = r_i . \int_{t_i}^{t_{i+1}} p_X(x).dx = r_i$$

Ainsi, la minimisation de l'erreur quadratique moyenne (quantification optimale) implique que les seuils de décision d_i et les niveaux de reconstruction q_i soient donnés par les relations : $\begin{cases} d_{i+1} = \frac{q_i + q_{i+1}}{2} \\ q_i = E\{X; x \in [d_i, d_{i+1}[\} \end{cases}$.

Ce sont les deux relations obtenues par Max. Ces dernières montrent que le quantificateur optimal d'un signal à loi de probabilité non-uniforme n'est pas un quantificateur linéaire.
Dans le cas particulier d'un signal d'entrée dont la densité de probabilité est constante sur $[x_{min}, x_{max}]$, on a : $p_X(x) = \frac{1}{(x_{max} - x_{min})}$, et on obtient alors les relations:

$$\begin{cases} r_i = \frac{t_i + t_{i+1}}{2} \\ t_{i+1} = \frac{r_i + r_{i+1}}{2} \end{cases}$$

Le quantificateur optimal est, dans ce cas, linéaire : $\Delta t_i = \frac{x_{max} - x_{min}}{K}$.

Pour un signal normalisé sur [0, 1], l'erreur quadratique minimisée vaut : $\varepsilon^2 = \frac{1}{12.K^2}$.

Nous avons vu que pour optimiser le quantificateur, il fallait, avec ce critère, minimiser l'erreur quadratique moyenne de quantification. Naturellement, d'autres critères peuvent être utilisés, en particulier en prenant en compte des aspects visuels.

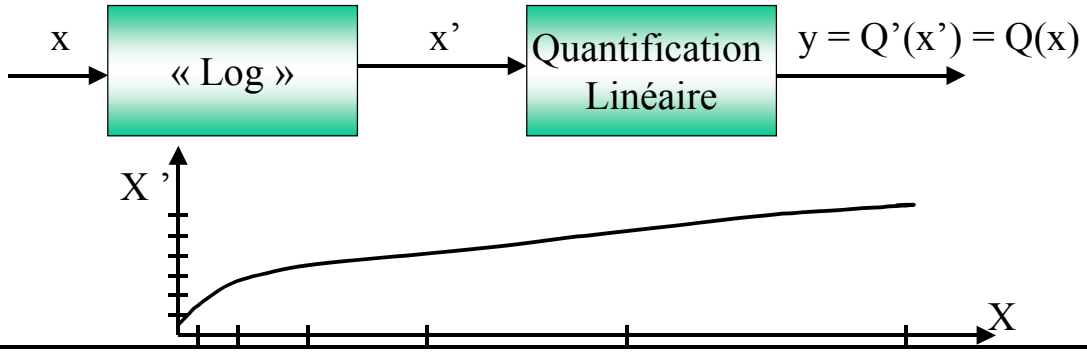
Quantification Optimale (2)

- **Quantification à entropie maximale** : H(Y) maximum si Y est à loi de probabilité constante

$$\Rightarrow \quad p_i = \int_{t_i}^{t_{i+1}} p_x(x)\,dx = \text{constant} \quad \forall i = 1, \ldots\ldots, K$$

- **Quantification Psycho-visuelle des images**

  - Perception de petits stimulus sur l'arrière plan : $\Delta L / L = cst$
  - $Cst \cong 0.01$ dans la gamme des luminances affichées sur écran TV (loi de Weber)

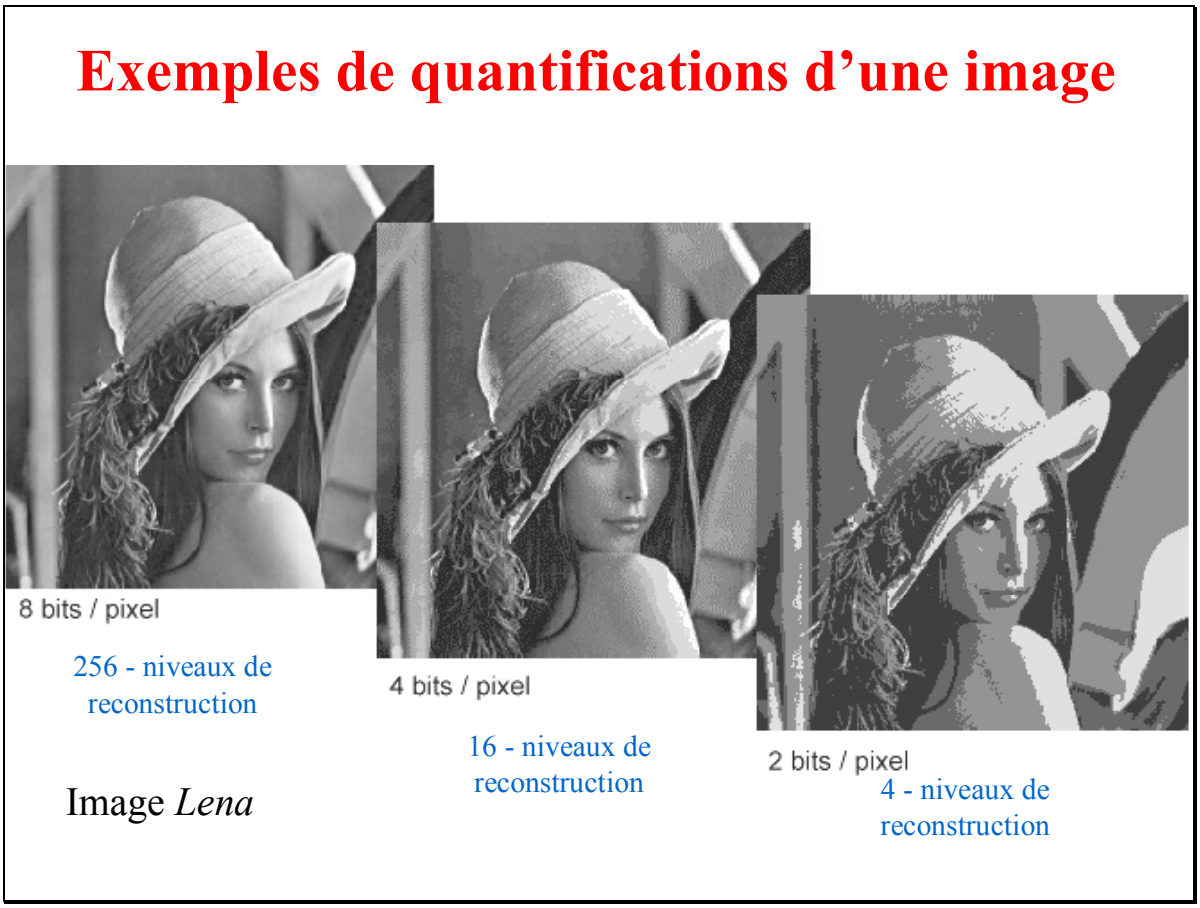


Les caractéristiques du système visuel humain peuvent être utilisées pour concevoir la loi de quantification. Pour une dégradation de forme donnée et d'amplitude ΔL fixée, le seuil de visibilité $\Delta L / L$ est à peu près constant (L étant la luminance). Dans la gamme des luminances TV, il varie légèrement quand L passe du noir au blanc. Pour une quantification au seuil de visibilité, on a donc $r_{i+1}-r_i=\Delta L$. Il est alors possible d'utiliser une quantification uniforme sur un signal compressé par une non-linéarité « f » (voir schéma bloc ci-dessus). Il faudra bien évidemment effectuer l'opération inverse à la sortie du quantificateur avec une fonction d'expansion.

Cette fonction de compression non linéaire est définie par $f(x) = \int_{L_0}^{x} \Delta L(x).dx \approx \lambda.Log(x / L_0)$.

En télévision, une compression « f » est déjà utilisée : c'est la correction gamma donnée par $f(x) = x^{\frac{1}{\gamma}}$ qui compense la non-linéarité des écrans TV de type CRT. On applique ensuite une quantification linéaire sur 8 bits. En télévision couleur, on quantifie usuellement les signaux de luminance et de chrominance séparément. En effet, pour à la perception, l'espace (Y, Cr, Cb) est plus uniforme que l'espace (R, V, B).

On voit donc par les applications du type télévision que malgré les propriétés intéressantes du quantificateur optimal, il peut parfois être plus approprié d'utiliser une fonction compression ponctuelle non-linéaire puis de réaliser une quantification linéaire (même si la loi de probabilité du signal n'est pas uniforme). L'ensemble « fonction de compression – quantification – fonction d'expansion » sera choisi de façon à approximer la loi de quantification optimale.



Cette figure montre des exemples d'une quantification effectuée sur l'image *Lena* :

- Pour l'image de gauche : la quantification est suivie d'un codage binaire naturel avec 8 bits par pixel. Il y'a donc $2^8 = 256$ niveaux de reconstruction pour la représentation des amplitudes de chaque pixel. C'est le cas typique d'une image monochrome (en niveaux de gris uniquement).
- Pour l'image au centre : la quantification est effectuée avec un codage sur 4 bits par pixel et donc $2^4 = 16$ niveaux de reconstruction. Les contours sont bien rendus mais les textures moins bien rendues dans certains cas. Ce sont les zones à variation spatiales lentes du signal qui sont les plus dégradées par l'apparition de faux contours (dégradations sur les joues et sur l'épaule).
- Pour l'image de droite : la quantification est effectuée avec un codage sur 2 bits par pixel et donc $2^2 = 4$ niveaux de reconstruction. Les dégradations observées sur l'image précédente sont encore nettement plus accentuées.

Nous avons présentés les diverses étapes dans la numérisation d'une image : la double discrétisation par échantillonnage spatial puis par quantification. Les images sont maintenant numériques et prêtes à être traitées avec des méthodes appropriées, dépendantes des applications.

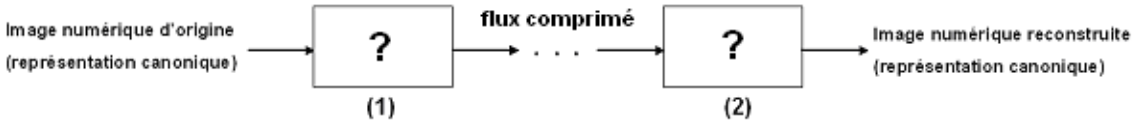
Chapitre 1 – Introduction au traitement d'images numériques

TEST

1 – Dans le tableau suivant, indiquez par OUI ou par NON si l' « ***Objectif*** » est concerné par le « ***Domaine*** » du traitement d'image indiqué.

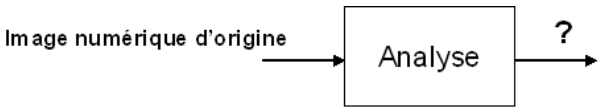
Domaine / ***Objectif***	Codage avec compression d'information	Synthèse d'images	Amélioration et restauration d'images	Analyse d'images
Mesurer la taille d'un objet				
Visualiser l'image d'un objet				
Réduire le bruit				
Réduire le débit binaire				
Déterminer la couleur d'un objet				
Accroître le contraste d'une image				
Reconnaître une forme dans une image				

2 – Le schéma suivant représente, très synthétiquement, une application de compression d'images ou de vidéos numériques :



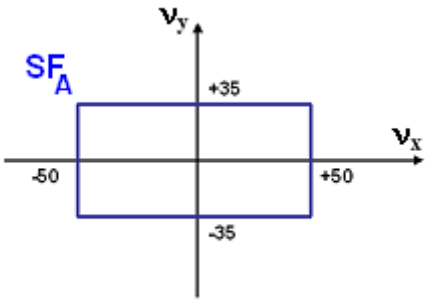
Quel est l'objet de chacune des deux grandes fonctions (boites noires (1) et (2)) ?

3 – Pour une application d'analyse d'images, donnez deux exemples de données à la sortie du bloc Analyse.



4 – Échantillonnage

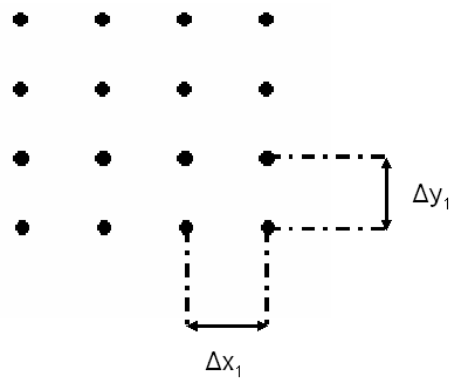
Soit une image analogique I_A (signal d'image $f_A(x, y)$) dont le spectre d'amplitude F_A est de support donné par $SF_A(\nu_x, \nu_y)$ tel que :



- ν_x, ν_y : fréquences spatiales horizontale et verticale, respectivement.

On échantillonne cette image I_A par une structure d'échantillonnage carrée E_1 avec trois valeurs de pas $(\Delta x_1, \Delta y_1)$:

- a) $\Delta x_1 = \Delta y_1 = 1/40$
- b) $\Delta x_1 = \Delta y_1 = 1/80$
- c) $\Delta x_1 = \Delta y_1 = 1/120$

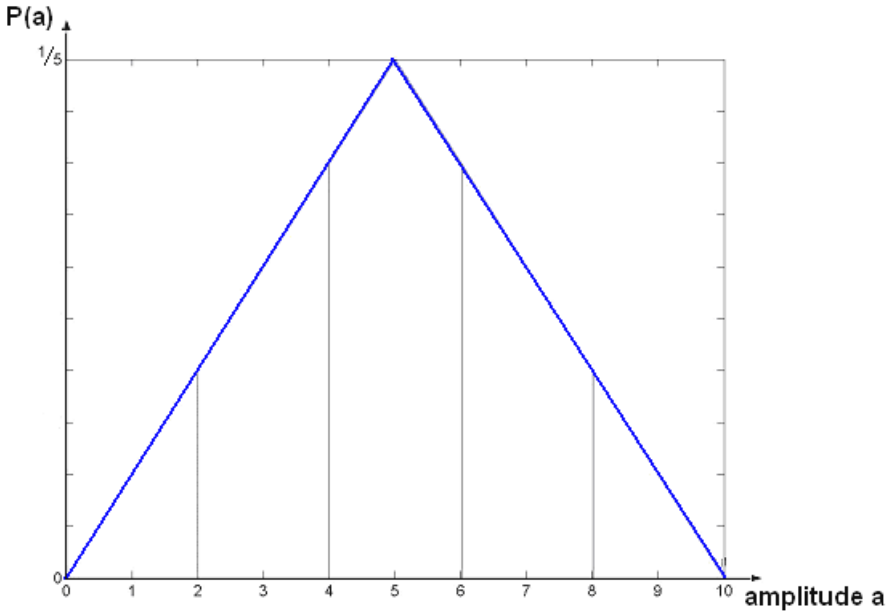


Pour chacun des cas a), b), et c), indiquez s'il existe un effet d'*aliasing* dans la structure horizontale ? dans la structure verticale ? justifiez à chaque fois votre réponse.

Qu'en concluez vous sur les différences avec l'échantillonnage de signaux purement temporels ?

5 – Quantification

Soit une image monochrome échantillonnée, dont la luminance suit la distribution de probabilité suivante :



Il s'agit d'une loi linéaire symétrique par rapport à 5.
On veut quantifier la luminance sur 5 niveaux. Pour cela, on fixe à priori les 4 seuils de décision t_i de façon à ce que les 5 intervalles $[t_i, t_{i+1}[$ soient de même mesure. On a donc : $t_0=0$; $t_1=2$; $t_2=4$; $t_3=6$; $t_4=8$; $t_5=10$.

a) Pour chacun des 5 intervalles $[t_i, t_{i+1}[$; i = {0, 1, 2 , 3, 4, 5} déterminez le niveau de quantification r_i optimal qui minimise l'erreur quadratique de reconstruction.

b) À partir de ces 5 valeurs r_1, r_2, r_3, r_4, r_5 optimales de quantification, quelles seraient les valeurs optimales des 4 seuils de décision t_1, t_2, t_3, t_4 ?

On peut remarquer qu'il suffirait de réitérer cet ensemble de 2 phases pour aboutir finalement, après stabilisation des valeurs de quantification et de décision, à la loi de quantification optimale.