LABORATOIRE

INFORMATIQUE, SIGNAUX ET SYSTÈMES
DE SOPHIA ANTIPOLIS
UMR 6070

COURS DE TRAITEMENT D'IMAGES

Lingrand Diane

Projet RAINBOW

Rapport de recherche
ISRN I3S/RR–2004-05–FR

22 Janvier 2004

LABORATOIRE I3S: Les Algorithmes / Euclide B – 2000 route des Lucioles – B.P. 121 –
06903 Sophia-Antipolis Cedex, France – Tél. (33) 492 942 701 – Télécopie : (33) 492 942 898
http://www.i3s.unice.fr/I3S/FR/

RÉSUMÉ :

Ce document présente les bases du traitement d'images appliqué. Il traite des problèmes informatiques liés à l'implémentation des algorithmes permettant à tout informaticien de pouvoir coder chacun des algorithmes, de les comprendre et d'être capable d'analyser les résultats. Date : 22 Janvier 2004.

MOTS CLÉS :

Traitement d'Images, Informatique, Sciences de l'Ingénieur

ABSTRACT:

This document aims to be an applied image processing course. It helps computer scientist to implement and understand basics algorithms, and analyse results on several examples. Date : January 22, 2004.

KEY WORDS :

Image Processing, Computeur Science

Table des matières

Cet ouvrage a été réalisé entièrement à l'aide de logiciels libres, sous un système **Linux**. Les exemples ont été programmés en langage Java par l'auteur ainsi que l'exportation au format Postscript permettant leur insertion dans ce document **LaTeX**. Les figures ont été réalisée à partir de **xfig** ou **The Gimp**. Les transparents associés au cours sont écrits à l'aide d'**Open Office**.

Cet ouvrage est issu du cours de traitement d'images que je donne à l'ESSI[1], en seconde année de cursus ingénieur (Bac +4). Je tiens à remercier les différents chargés de TP qui ont accepté, au fil des années, la charge d'un groupe : Pierre Kornprobst (2001-2002), Frédéric Précioso (2002-2003), Ariane Herbulot et Johan Montagnat (2003-2004).

[1] **E**cole **S**upérieure en **S**ciences **I**nformatiques, `http ://www.essi.fr`

Chapitre 1

Introduction au traitement d'images

1.1 Formation des images

Commençons par nous intéresser à la formation des images. Une image est obtenue par transformation d'une scène réelle par un capteur. Dans une scène à imager, les objets soit émettent une onde électromagnétique, ce sont des sources, soit réfletent ou réfractent une onde électromagnétique. Bien sur, nous pensons aux ondes électromagnétiques dont les longueurs d'onde se situent dans le visible (de 400μm à 800μm) mais il ne faut pas oublier les autres longueurs d'ondes en infra-rouge ou ultra-violet. L'imagerie infra-rouge est utilisée pour détecter des zones émettant de la chaleur.

1.1.1 Les capteurs

Parmi les capteurs, on peut distinguer les capteurs chimiques tels que les systèmes biologiques (exemple : notre oeil), les films photographiques, les capteurs thermiques (thermopile) et les capteurs photoélectriques (photodiodes, CCD). Les appareils numériques disponibles sur le marché sont généralement équipés de cellules CCD. Il existe bien d'autres capteurs, notamment dans le domaine de l'imagerie médicale (IRM, Tomographie, ...) ou de l'imagerie sismique.

Le signal obtenu est caractérisé par sa dimension et sa nature. Un signal de dimension 1 (on dit 1D) correspond à une image linéique. C'est le type de signal que l'on a l'habitude de voir sur un oscilloscope par exemple.

C'est aussi le type d'image à une seule ligne (ou une seule colonne) que l'on peut obtenir à l'aide d'une barrette CCD. Un signal de dimension 2 (on dit 2D) est une image, souvent plane, tel qu'une photographie de vacances, par exemple. Une image de dimension 3 est une image volumique : par exemple une image IRM du cerveau. Une image de dimension 3 peut également être une vidéo. Dans ce cas, on parle d'image 2D+T (2 dimensions spatiales et une dimension temporelle). La nature du signal peut être analogique (continue) ou numérique (discrète). Dans ce document, par défaut, lorsque l'on parlera d'image, il s'agira d'image 2D, sauf mention contraire.

1.1.2 L'image est discrète

La représentation informatique d'une image est nécessairement discrète alors que l'image est de nature continue : le monde est continu. Certains capteurs effectuent une discrétisation : c'est le cas des appareils photos numériques. Par contre, les appareils photos argentiques n'effectuent pas cette discrétisation.

Si on regarde d'un peu plus près, la transformation d'un signal analogique 2D nécessite à la fois une discrétisation de l'espace : c'est l'échantillonnage, et une discrétisation des couleurs : c'est la quantification.

Pour un ordinateur, une image est un ensemble de pixels. Un pixel, c'est un élément d'image (*picture element*). On peut considérer qu'un pixel n'a pas de dimension car si on regarde une photo, un pixel d'un objet situé près de la caméra correspond à une taille plus petite qu'un pixel d'un objet situé loin de la caméra, même si ces deux pixels font partie de la même image. A contrario, par exemple pour une image de lamelle microscopique, on va vouloir relier la taille d'un pixel à celle des éléments sur la lamelle. Dans ce cas, tous les pixels correspondent à une même taille dans le monde 3D. Alors, le pixel peut avoir une dimension. C'est particulièrement vrai dans le cas des images médicales 3D où des mesures de volumes et autres sont effectuées. La taille d'un pixel est reliée à l'échantillonnage.

Un pixel possède une valeur qui peut être un scalaire et représenter un niveau de gris, ou un vecteur représentant une couleur, ou tout autre chose.

Les images dites en "noir et blanc" sont composées de pixels noirs ou blancs (deux valeurs possibles). Les images en niveaux de gris sont composées de pixels de valeurs scalaires représentant la luminosité. En général, les valeurs sont entières entre 0 (le noir) et 255 (le blanc). Les images couleurs sont composées de pixels dont les valeurs sont soit scalaires (on parle alors de couleurs indexées car la valeur représente un numéro de couleur) soit

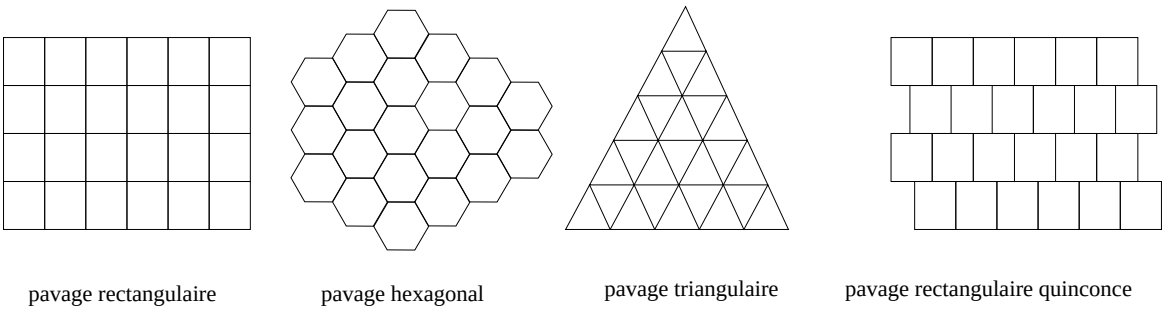


FIG. 1.1 – Différents pavages

vectorielles (les couleurs sont représentées selon trois composantes ou plus, voir le chapitre 2 sur la couleur pour plus de détails).

Le nombre de valeurs pouvant être prises par un scalaire est directement lié à la quantification. Lorsque le nombre de valeurs est limité, des défauts peuvent apparaître. Lorsqu'on augmente le nombre de valeurs, celles-ci prennent plus d'espace et l'image sera alors plus volumineuse. Il faut donc trouver un compromis acceptable entre qualité de l'image et mémoire occupée. Ce compromis va dépendre de l'application et de ses contraintes.

Pour donner un ordre de grandeur, si un pixel est codé sur 8 bits (1 octet), on dispose de $2^8 = 256$ couleurs ou niveaux de gris. Si on code ce même pixel sur 16 bits, on dispose alors de $2^{16} = 65536$ couleurs ou niveaux de gris, soit 256 fois plus pour un espace mémoire 2 fois plus grand.

1.1.3 Pavage ou tesselation

Les pixels sont généralement arrangés sous forme rectangulaire, dans un tableau 2D. Mais il peut exister également d'autres pavages ou tesselation comme par exemple, le pavage hexagonal ou triangulaire ou le pavage rectangulaire en quinconce (voir figure 1.1).

Certains pavages sont motivés par leur adéquation au système d'acquisition (le pavage en quinconce correspond notamment à certains systèmes d'acquisitions embarqués dans les satellites). D'autres sont motivés par des relations géométriques telles que le voisinage. Si on considère le voisinage dans le cas du pavage rectangulaire, les voisins ayant une arête en commun avec le pixel courant ne seront pas forcément considérés comme les voisins ayant seulement un sommet en commun. On parle alors de connexité 4 ou 8, représentant le nombre de voisins considérés. Par contre, dans le cas du

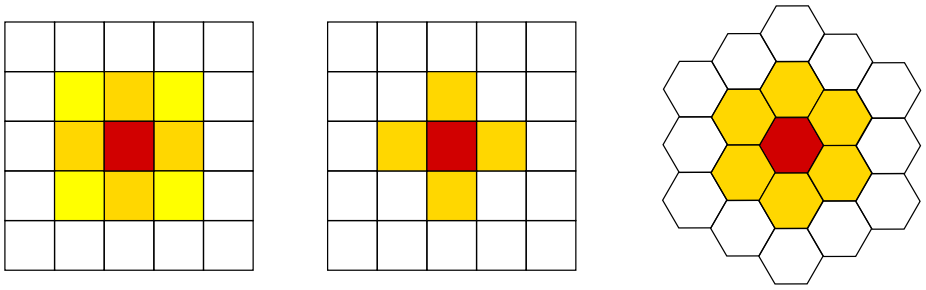


FIG. 1.2 – Notions de voisinage en pavage rectangulaire (connexité 8 ou 4) et en pavage hexagonal.

pavage hexagonal, on n'a pas ce problème ; tous les voisins sont au même niveau (voir figure 1.2).

Cependant, sauf cas particulier, on se placera par la suite dans le cas du pavage rectangulaire, et ce, pour des raisons de commodités : il correspond au stockage des pixels dans un tableau à 2 dimensions.

1.1.4 Distances

Après la notion de voisinage, vient naturellement la notion de distance entre pixels. Différentes distances peuvent être utilisées (voir figure 1.3 pour illustration).

La distance euclidienne :

$$\delta([ij],[kl]) = \sqrt{(k-i)^2 + (l-j)^2}$$

a l'avantage de ne privilégier aucun axe, ce qui n'est pas le cas de la distance par blocs (appelée également *Manhattan distance* pour référence à la ville de Manhattan où il faut contourner différents pâtés de maison (*blocks*) pour relier 2 points) :

$$\delta([ij],[kl]) = |k-i| + |l-j|$$

ou de la distance "tour d'échiquier" :

$$\delta([ij],[kl]) = max(|k-i|, |l-j|)$$

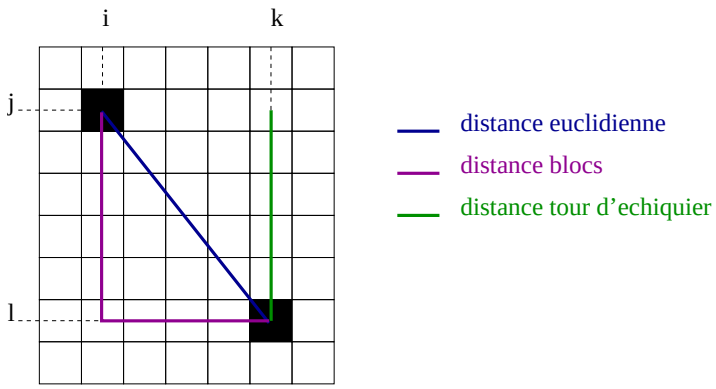


FIG. 1.3 – Différentes distances entre 2 pixels.

Par contre, ces deux dernières distances présentent des avantages au niveau du coût de calcul.

1.2 Stockage des images

On va s'intéresser maintenant au stockage des images en mémoire, c'est-à-dire sous forme de fichier. Les infos qui vont être stockées sont la largeur et la hauteur ainsi que les valeurs des pixels. On peut vouloir stocker d'autres informations telles que le type de données, l'auteur, la date, les conditions d'acquisition, ... On va ainsi stocker les informations concernant l'image dans un en-tête puis les données. Certains formats d'images stockent dans 2 fichiers différents l'en-tête et les données. Les données sont souvent stockées dans l'ordre des pixels de gauche à droite et de haut en bas, relatif au repère image. Nous allons étudier, à titre d'exemple, quelques formats classiques.

1.2.1 Le format pbm

Commençons par le plus simple : le format pbm. Ce format permet de stocker des images en noir et blanc, c'est-à-dire dont les pixels ne peuvent prendre que 2 valeurs : 0 (noir) ou 1 (blanc).

Regardons le début du fichier :

```
P1
# Mes vacances en Norvege en noir et blanc
```

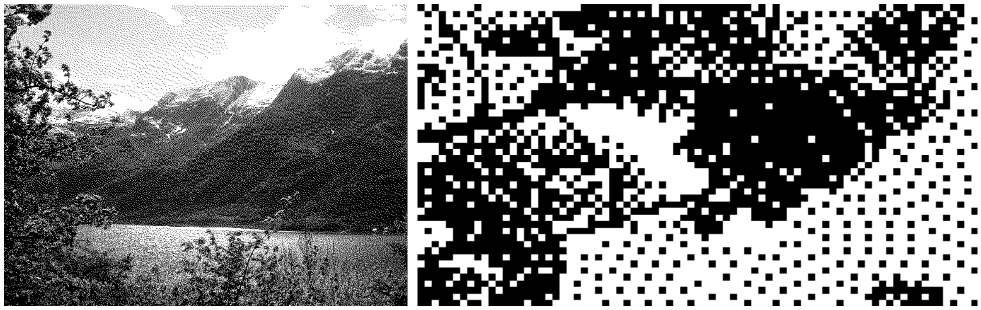


FIG. 1.4 – Une image au format pbm et le grossissement du coin supérieur gauche

```
640 480
1 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 ...
1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 ...
1 1 1 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ...
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ...
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ...
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ...
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ...
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ...
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ...
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ...
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ...
```

La première ligne `P1` signifie qu'il s'agit d'un format pbm et que les données seront stockées en ascii. La deuxième ligne, commençant par le caractère `#` est une ligne de commentaire. La ligne suivante `640 480` précise les dimensions de l'image : la largeur de l'image est de 640 pixels, la hauteur de 480 lignes. Viennent ensuite les données, ligne par ligne, et pour chaque ligne, de gauche à droite. On peut voir, sur le grossisement du côté supérieur gauche de l'image 1.4 que les données correspondent bien.

1.2.2 Le format pgm

Le format pgm permet de représenter des images en niveaux de gris dont les pixels ont des valeurs entières comprises entre 0 (noir) et 255 (blanc).

```
P2
```

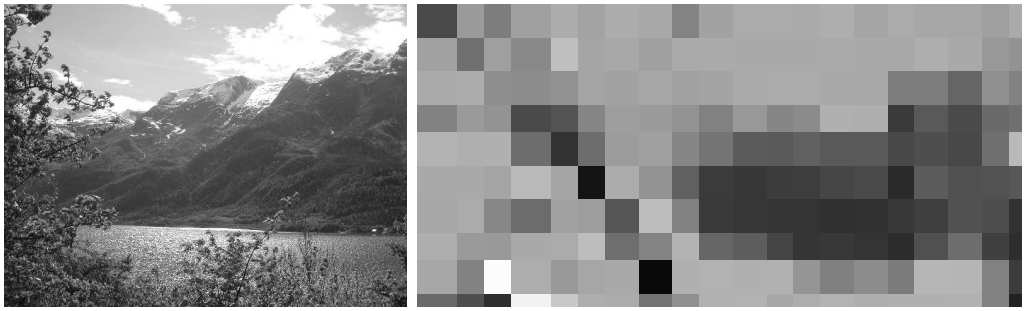


FIG. 1.5 – Une image au format pgm et le grossissement du coin supérieur gauche

```
# Mes vacances en Norvege
640 480
255
 73 154 125 160 172 163 172 168 131 162 171 171 170 174 165 171 167
166 159 170 164 170 157  93  54  67  86  96  50  75  61  88 135 163
 54  85  82  70  66  72  45  56  56  54  80  84 111 134 103 120 154
169 175 182 174 175 172 172 176 175 173 177 173 174 155 115 122 114
104  99  91 102  95 102 117 103 106 190 178 176 177 176 176 180 179
176 176 175 177 179 179 178 180 182 180 179 180 177 181 184 179 180
180 178 178 179 179 179 181 181 183 185 182 184 182 182 182 181 182
183 184 183 185 182 184 186 186 184 185 182 184 183 184 184 184 186
185 184 184 186 183 184 184 187 190 183 187 184 185 185 186 187 186
188 187 188 187 185 191 189 185 186 188 187 187 185 185 185 188 185
```

La première ligne `P2` signifie qu'il s'agit d'un format pgm et que les données seront stockées en ascii. Les deuxième et troisième lignes sont inchangées. La quatrième ligne précise la valeur maximale pouvant être prise par un pixel (ici, 255 correspondant au blanc). Viennent ensuite les données (voir figure 1.5).

1.2.3 Le format ppm

Le format ppm concerne les images couleurs. Chaque pixel a pour valeur un triplet (R,G,B) composé d'une composante rouge, verte et bleue. Chaque composante est représentée par un entier pouvant prendre ses valeurs entre 0

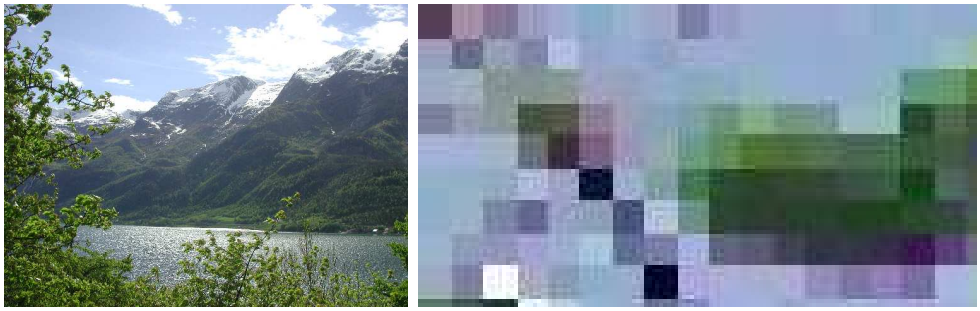


FIG. 1.6 – Une image au format ppm et le grossissement du coin supérieur gauche

et 255. Le triplet (0,0,0) correspond au noir, (255,0,0) au rouge, (255,255,0) au jaune, ..., et (255,255,255) au blanc.

```
P3
# Mes vacances en Norvege en couleurs
640 480
255
89  62  79 167 143 165 130 117 145 155 155 191 160 169 210
137 170 203 151 176 206 158 167 198 127 125 164 155 156 200
156 169 211 155 170 211 153 170 213 153 177 215 139 172 205
150 174 210 146 171 202 140 175 195 136 169 178 152 180 181
144 175 177 148 183 177 140 175 142  82 113  54  45  72  19
 62  79  45  82  97  64  92 108  71  46  63  18  72  92  33
 64  79   0  87 109  26 129 157  82 158 184 113  54  72   0
 79 107  30  71 102  43  60  88  37  60  86  21  68  88  35
 44  55  21  53  67  31  51  69  27  50  68  18  81  94  38
```

La première ligne `P3` signifie qu'il s'agit d'un format ppm et que les données seront stockées en ascii. Les trois lignes suivantes sont inchangées. Les valeurs des pixels sont données, pixel par pixel, composante par composante. Le premier pixel en haut à gauche de l'image a donc pour valeur (89, 62, 79) ce qui correspond à un violet un peu foncé. Le pixel suivant (167,143,165) est plus clair, toujours dans les teintes violet, ... (voir figure 1.6).

1.2.4 Les formats binaires pbm, pgm et ppm

Regardons maintenant les tailles des fichiers (sous Linux, par la commande `du -sm ph021-ascii.pbm` permet d'obtenir la taille en méga-octets ; sous Windows, en choissant Affichage/détails) :

PBM : 620 Ko

PGM : 1.2 Mo

PPM : 3.6 Mo

Ce qui est cohérent puisque, pour le format pbm, un pixel est exprimé par un caractère et un espace soit 2 caractères ce qui fait 2*640*480 = 620 Ko. Pour le format pgm, un pixel est exprimé par un nombre à 3 chiffres (ou moins) et un espace soit à peu près 4 caractères ce qui fait 4*640*480 = 1.2 Mo. Pour le format ppm, on a 3 composantes ce qui fait 3 fois plus de caractères soit 3*4*640*480 = 3.6 Mo.

Ces images sont volumineuses et on a va essayer de coder ces valeurs de façon plus optimale : en binaire. Les formats binaires ont un en-tête généralement en ASCII et les données en binaires. Cela signifie qu'on ne peut pas lire avec un éditeur simple le contenu et qu'il va falloir connaître la représentation des valeurs (entiers/décimaux, signés/non signés, sur n bits).

En C/C++, cela revient à utiliser :

```
size_t fread(void *data, size_t size, size_t nobj, FILE *stream)
size_t fwrite(const void *data, size_t size, size_t nobj, FILE *stream)
```

Mais attention, on ne parle pas bien-sûr des images ascii telles que celle-ci [1] :

[1] obtenue à l'aide du converteur `http ://www.text-image.com/convert/ascii.html`

Mais attention aux indiens ! Effectivement, lorsque l'on code en binaire des données, les bits peuvent être dans deux sens différents selon l'architecture de la machine sur laquelle les données auront été codées.

***little endian* :** les octets de poids faibles sont stockés aux adresses les plus petites (ex. : processeur Intel)

***big endian* :** les octets de poids forts sont stockés aux adresses les plus petites (ex. : processeur Motorola (Mac))

Il existe alors deux choix possibles :

- le format dépend de la machine qui a écrit le fichier : il y a alors nécessité de mémoriser cette information dans l'en-tête du fichier (TIFF, xwd (X Window Dump))
- le format ne dépend pas de la machine qui a écrit le fichier : on choisit un ordre donné pour ce format de fichier (Big endian : Adobe Photoshop, JPEG, MacPaint ; Little endian : BMP, GIF, QTM (QuickTime))

Les formats pbm, pgm et ppm utilisent le choix de l'ordre *little endian* . Si l'on reprend l'exemple du format pbm, un bit suffit à coder un pixel ce qui donne 1*640*480/8 = 38 Ko. Pour le format pgm, un pixel prend ses valeurs entre 0 et 255 soit 256 valeurs différentes ce qui se code sur 8 bits (1 octet). L'image a alors pour taille 1*640*480 = 300 Ko. Pour l'image au format ppm, on a 3 fois plus de valeurs soit 3*640*380 = 900 Ko. Ces calculs

négligent la taille de l'en-tête, ce qui explique les différences avec les résultats suivants (tailles réelles des fichiers) :

		
P1 PBM ascii : 620 Ko	P2 PGM ascii : 1.2 Mo	P3 PPM ascii : 3.6 Mo
P4 PBM raw : 40 Ko	P5 PGM raw : 308 Ko	P6 PPM raw : 908 Ko

1.2.5 Autres formats

Il existe bien d'autres formats basés sur ce principe mais différant par les informations dans l'en-tête ou le type de données : un pixel n'a pas toujours une valeur entière entre 0 et 255. Il peut s'agir de décimaux, de relatifs, ... Lorsque l'on va vouloir écrire ou lire une image d'un tel format, il est nécessaire de connaître les informations suivantes :

- taille occupée par la valeur d'un pixel (par exemple 8 bits)
- ordre des bits : *little endian* ou *big endian*
- représentation entière ou décimale
- représentation signée ou non signée

Ces formats sont généralement à diffusion plus restreinte (au sein d'une communauté scientifique, laboratoire, ...). On citera parmi eux le format `inrimage` ayant la particularité d'avoir une taille fixe d'en-tête (256 caractères ascii).

1.2.6 Quelques formats classiques

sQCIF : 128x96

QSIF : 160x120

QCIF : 176x144

SIF : 320x240

CIF : 352x288

VGA : 640x480

PAL : 720x576, 25 images par seconde, entrelacé

NTSC : 640x475, 30 images par seconde, entrelacé

SECAM : 720x576, 25 images par seconde, entrelacé

1.2.7 Formats avec compression

D'autres formats stockent de façon plus intelligente les données. Notamment, plusieurs pixels consécutifs peuvent avoir la même valeur. Plutôt que de mémoriser "blanc blanc blanc blanc blanc blanc blanc" on va préférer mémorise "7 fois blanc". Au pire, tous les pixels consécutifs sont différents : dans ce cas, on ne va pas réduire le nombre de données, au contraire, on va le doubler. Dans le cas le meilleure, tous les pixels sont identiques : on ne stocke que 2 valeurs ! Ce type de compression est sans perte : aucune information n'est perdue.

Il y a d'autres idées qui permettent de compresser les données. Par exemple, une couleur peu utilisée peut être remplacée par une autre couleur proche et plus présente dans l'image. On réduit ainsi le nombre de couleurs utilisées et donc le nombre de bits nécessaires à la représentation des couleurs. Par contre, on perd de l'information sur l'image. Pour que la compression soit effective, il faut renuméroter les couleurs : on parle alors de couleur indexée ou de représentation indirecte de la couleur.

Ce sont principalement ces idées qui sont à la base des algorithmes de compression que nous détaillerons au chapitre concernant ce sujet (chapitre 11). Elles sont appliquées soit directement aux pixels de l'image, soit à des transformées de l'image. On distingue deux types de compression :

compression sans pertes : TIFF (avec algorithme LZW), GIF (pour les images 8 bits), PNG (licence GNU www.visualtek.com/PNG)

compression avec pertes : JPEG, JPEG 2000

Pour avoir un ordre de grandeur, l'image précédente, en couleur et aux mêmes dimensions (640x480) ne fait plus que 88 Ko au format compressé JPEG.

1.3 Représentation informatique des images

Maintenant que nous avons fait le tour des formats des fichiers on va s'intéresser à la représentation des images comme structure de données. Lorsque l'on va manipuler des images, on va vouloir :

- accéder aux valeurs des pixels en fonction des coordonnées (i,j) dans l'image
- parcourir une image du premier au dernier pixel (par exemple lors de la lecture ou de l'écriture d'un fichier)
- accéder aux voisins d'un pixel (i,j)

On pense naturellement à une structure de tableau. Deux solutions sont alors possibles : le tableau à une dimension ou le tableau à deux dimensions.

Le tableau à une dimension facilite le parcours de tous les pixels de l'image (une seule boucle) :

```
for (int i = 0 ; i < image.getWidth()*image.getHeight(); i++)
{
        ...
}
```

tandis que le tableau à deux dimensions nécessite deux boucles imbriquées :

```
for( int i = 0 ; i < image.getWidth() ; i++) {
        for ( int j = 0 ; j < image.getHeight(); j++) {
                ...
        }
}
```

A contrario, pour accéder aux voisins, c'est plus facile avec un tableau à deux dimensions : voisins horizontaux et verticaux sont obtenus en décalant les indices d'une unité (voir figure 1.7).

1.3.1 En langage C

Voici un exemple de structure image considérant un tableau à une dimension :

```
typedef struct {
        int width;
        int height;
```

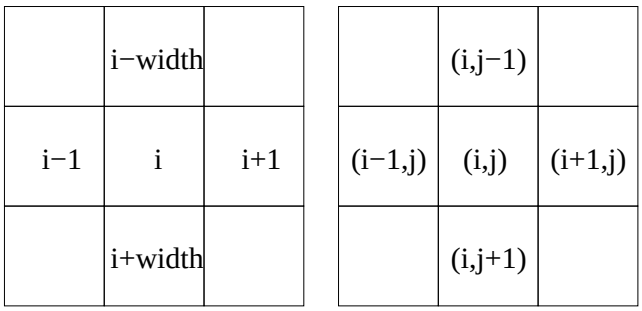


FIG. 1.7 – Indices des pixels voisins dans le cas du tableau 1D et du tableau 2D.

```
        unsigned char *data;
} ImageChar ;
```

Il existe plusieurs astuces pour bénéficier des avantages des deux options. On peut utiliser par exemple une variable intermédiaire :

```
int ij = i + j*image.getWidth();
```

Cependant, cette méthode introduit des calculs supplémentaires qui peuvent s'avérer coûteux s'ils sont effectués de nombreuses fois.

Une autre méthode consiste à ajouter un tableau `img` ayant autant de cases qu'il y a de lignes dans l'image et dont les éléments sont des pointeurs sur les lignes de l'images (voir figure 1.8). On note `p` le pointeur sur les données. Il est initialisé par :

```
char *p = &img[0][0];
```

ou :

```
char *p = &data[0];
```

Si on veut utiliser cette astuce, il suffit d'ajouter un champ à notre structure :

```
typedef struct {
        int width;
        int height;
```

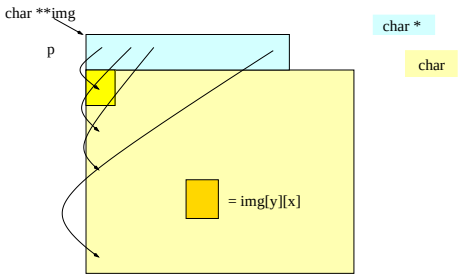


FIG. 1.8 – L'ajout d'une tableau de pointeurs sur les lignes permet de bénéficier à la fois des avantages des tableaux 1D et 2D. Un pointeur sur les données est initialisé : char *p = &img[0][0]

```
        unsigned char **img;
        unsigned char *data;
} ImageChar ;
```

Aucun calcul supplémentaire n'est nécessaire. Par contre, l'image prendra un peu plus d'espace en mémoire. Combien ? Il suffit de multiplier le nombre de lignes par la taille d'un pointeur ... c'est-à-dire `nombre de lignes * 32 bits` (en réalité, cela dépend de l'adressage de la machine sur laquelle vous travaillez et peut être 64 bits).

On peut accéder à un pixel (i,j) de la façon suivante : `img[i][j]` et on peut parcourir tous les pixels i de 0 à `width*height` par `p[i]`.

1.3.2 En langage C++

L'avantage du C++ et des *template* est de permettre de pouvoir déclarer une structure valable pour toutes les images, quelque soit le type de données : char, int, float, double, ...

```
template <class Type> class Image {
        public:
                Image( const Type&);
        private:
                Type *data;
                int width, height;
        ...
}
```

Par contre, lors de la lecture d'une image, on ne connait le type des données que lors de la lecture de l'en-tête. Or, le template doit être instancié avant d'être passé à la méthode de lecture. Une façon de s'en sortir est d'utiliser le polymorphisme : on crée une classe mère `BaseImage` dont hérite la classe template. A la lecture, on crée une `BaseImage` qui est en réalité une `Image <short int> i` ou `Image <double> i` ou ...

1.3.3 En langage Java

On peut définir une classe image comme suit :

```
public class Image {
        private int width;
        private int height;
        private int [] data;

        public Image(int w, int h) {
                        height = h;
                        width = w;
                        data = new int [h*w];
        }

        public int getPixel(int i, int j) { ... }
        public int getPixel(int ij) { ... }
        public int setPixel(int i, int j, int value) { ... }
...
}
```

Les pixels des images de cette classe ont pour valeur un int, composé de 4 octets permettant de coder les trois composantes de la couleur ainsi qu'une composante α que l'on décrira au chapitre 2.

Comment passer de composantes couleurs à un entier et réciproquement ? Lors de la constitution d'un entier, on n'oubliera pas de vérifier que les composantes de couleur sont comprises entre 0 et 255 compris. La valeur entière est alors définie par : `int col = 256*256*red + 256*green + blue ;`

Lors de la décomposition, il faut également faire attention que Java ne possède pas de type non signé. On récupère les composantes couleurs de la façon suivante :

```
int red = (col >> 16) & 0x000000ff;
int green = (col >> 8) & 0x000000ff;
int blue =  col & 0x000000ff;
```

En réalité, il n'est pas nécessaire de créer une telle classe puisque l'API Java dispose déjà d'un ensemble de classes pour gérer les images. On distingue :

Push Model ou Producer/Consumer : présent depuis les `jdk 1.1.*, 1.2,*` avec les classes :
- Image
- ImageProducer
- ImageConsumer
- ImageObserver

Immediate Mode ou Image Buffer Model : apparu avec Java2D. Parmi les classes disponibles, on peut citer :
- BufferedImage, Raster
- BufferedImageOp, RasterOp

Pull Model : présent dans `JAI` (*Java Advanced Imaging*). On peut citer les classes :
- RenderableImage
- RenderableImageOp

Dans les travaux pratiques proposés, on ne s'intéressera pas à `JAI` mais on tachera d'implémenter nous-même les algorithmes étudiés. On se focalisera sur le modèle *Image Buffer Model* et on utilisera la classe `BufferedImage` pour stocker les images.

1.3.4 Entrées-sorties avec Java

La classe `javax.imageio.ImageIO` est apparue avec la `j2sdk 1.4.0` et facilite notablement la lecture et l'écriture de fichiers images. Elle comprend les formats gif (seulement en lecture), jpeg, png et xbm. Un mécanisme d'ajout d'autres formats est prévu via des *plugins*.

Lecture d'un fichier

```
File f = new File(''toto.gif'');
BufferedImage bi;
```

```
try {
        bi = ImageIO.read(f);
}
catch(IOException e) {...}
```

Ecriture d'un fichier

```
File f = new File(''toto.jpg'');
try {
        ImageIO.write(bi, ''jpg'', f);
}
catch(IOException e) {...}
```

1.3.5 Une image incontournable

Le traitement d'image tel que nous l'entendons aujourd'hui est apparu dans les années 70. La non disponibilité d'appareil photographique numérique ou de scanner ainsi que les possibilités de stockage de nombreuses images réduites ont contribué à l'utilisation d'un nombre réduit d'images. De plus, une volonté de tester différents algorithmes sur une même image afin de comparer les résultats ont conduit à la célébrité de quelques images, et surtout à celle de Léna, issue d'une photographie du magazine Playboy.

1.4 Logiciels

1.4.1 Outils de visualisation, traitement d'images et dessin vectoriel

Dans cette section, nous présentons brièvement une liste d'outils pour les systèmes Linux (Unix), Windows et Mac OS. Cette liste est loin d'être exhaustive et évolue constamment. Elle a pour unique ambition de donner des points de départ aux novices.

Les principauxs outils de visualisation disponibles sous Linux sont `gqview`, `display` et `xv`. Sous Windows, on trouve principalement `ACDSee`.

Parmis les outils de traitement, les plus répandus sous Linux sont `The Gimp`, très complet mais nécessitant un apprentissage, ainsi que `xpaint`, ancien, assez primitif mais très simple d'emploi. `The Gimp` est également disponible sous Windows et MacOS. Sous Windows, `Photoshop` est très répandu.

FIG. 1.9 – La très célèbre Léna. A gauche, telle qu'on la connait en traitement d'images ; à droite, la photographie originale.

Parmi les outils de dessin, on peut distinguer les outils de dessin au niveau *bitmap* tels que `The Gimp`, `xpaint`, `kpaint` (paint version KDE) et `Photoshop` et les outils de dessin vectoriel tels que `xfig` (outil ancien mais toujours efficace), `dia` (linux), `Corel Draw` (Windows et Mac), Quark Express (outil souvent utilisé par les professionnels de l'imprimerie). Parmi les formats de dessin vectoriel, on peut noter `Postscript`, `SVG` mais aussi de nombreux formats propriétaires tels que ceux de `Corel Draw`.

1.4.2 Bibliothèques de développement

En Java, il y a l'API Java2D que nous utiliserons dans les rubriques "Mise en pratique". On peut s'intéresser à JAI (Java Advanced Imaging) pour des applications plus haut niveau.

En C/C++, il y a OpenCV (Intel corporation) mais aussi beaucoup d'autres...

1.5 Exercices

Exercice 1 : Soit une image carrée au format ppm (ascii) de 160kO. Quelles sont, approximativement, les dimensions de cette image ? Quelle serait la taille du fichier si cette image était stockée en binaire ?

Exercice 2 : Soit une image de dimensions 1024 x 768 dont les pixels sont codés sur 16 bits. On stocke cette image au format inrimage pour lequel l'entête a pour taille 256 caractères et les données sont stockées en binaire. Quelle est la taille du fichier ?

Exercice 3 : Soit le morceau de code C suivant :

```
...
for(i = 0; i < 1024; i++)
        for(j = 0; j < 768; j++)
                fprintf(file, ''%d '', data[i][j]);
...
```

Le fichier image "file" sera t-il codé en ascii ou en binaire ? Comment modifier le code pour utiliser l'autre codage ?

1.6 Mise en pratique

Créer une interface en Java permettant de lire un fichier image, de l'afficher et de le sauvegarder en choisissant le format. On demande que lorsque la souris se déplace sur l'image, les coordonnées ainsi que la(les) valeur(s) des pixels soient affichées, par exemple dans un `JLabel`.

Ajouter la possibilité de lire et écrire des images au format PPM (ascii et raw).

Les quelques lignes de code suivantes peuvent aider à lire un fichier ascii :

```
import java.io.*;
...
        String filename = "monFichierAmoi";

        BufferedReader br ;

        try {
            br = new BufferedReader(new FileReader(filename));
            // c'est bon, on a ouvert le fichier
        }
        catch(FileNotFoundException e) {
            System.out.println("Fichier " + filename + " inexistant.");
            // mettre ici le code dans le cas ou le fichier n'existe pas
        }

        String s;

        try {
            do {
                s = br.readLine(); // s vaut null en fin de fichier
                if(s != null && !s.equals(""))
                    System.out.print("j'ai lu une nouvelle ligne du fichier'');
                    System.out.println("et la voici :\n"+ s);
            } while(s != null && !s.equals(""));
        }
        catch(IOException e){
            System.out.println("Erreur de lecture "+filename+" !");
            // cas d'erreur de lecture
```

```
    }
```

Afin d'extraire les nombres dans une chaine de caractères, utiliser la classe `StringTokenizer`.

Ne pas oublier pas que si la lecture échoue (fichier manquant, mauvais format de fichier, nombre de pixels erroné, ...), il faut le préciser (un affichage de message d'erreur ne suffit pas), par exemple en utilisant des Exceptions avec des messages explicites.

Pour le débugage et pour un test systématique une image très petite comme par exemple la suivante est utile :

```
P3
#ca, c'est mon image pour le debug
2 3
255
0 0 0 1 1 1 2 2 2 3 4 5 12 23 34 134 231 244
```

Lors de la lecture, déterminer automatiquement si le fichier image est au format ascii ou binaire (P3 signifie ascii, P6 binaire). Ne pas oublier que les données sont stockée par octets (à un octet correspond une composante couleur d'un pixel).

Ne pas oublier pas de tester soigneusement le code.

Chapitre 2

Notions à propos de la couleur

2.1 Qu'est-ce que la couleur ?

La couleur est à la fois un phénomène psychophysique faisant intervenir la physique de la matière, notamment les interactions des ondes électromagnétiques avec les matériaux physiques. C'est également un phénomène psychophysiologique par l'interprétation des phénomènes psychophysiques par le système visuel constitué notamment de l'oeil et du cerveau.

On connait le spectre de la lumière blanche mis en évidence par Isaac Newton en 1666 lors de la diffraction de la lumière blanche par un prisme (voir figure 2.1). Ce sont également les couleurs présentes dans l'arc en ciel, phénomène résultant de la diffraction de la lumière du soleil dans les goutelettes d'eau.

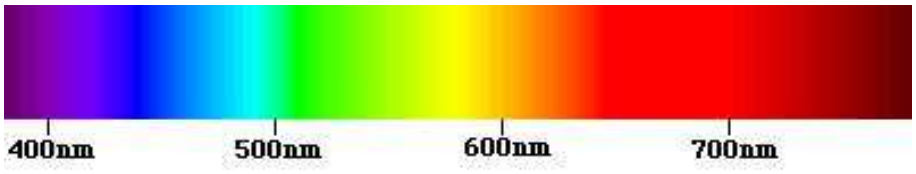


FIG. 2.1 – Spectre de la lumière blanche, couleurs de l'arc en ciel.

2.1.1 Quelques définitions

Une lumière contient une part de lumière achromatique et une part de lumière chromatique. Une lumière est dite achromatique lorsqu'elle contient toutes les longueurs d'onde de façon approximativment égales. C'est une définition théorique que l'on ne peut utiliser qu'en discrétisant le domaine des longueurs d'ondes. Nous reviendrons sur ce point plus tard.

La **teinte** est le nom de la couleur, c'est-à-dire de la longueur d'onde dominante. C'est une grandeur qui est repérable : on peut déterminer aisément la longueur d'onde dominante et donner un nom en fonction du spectre vu figure 2.1. Par contre, cette grandeur est non mesurable et non additive : en effet, on ne peut pas déterminer la couleur résultant d'une addition de 2 autres couleurs. On verra au paragraphe concernant la représentation de la couleur (2.2) qu'il existe une sorte d'addition et de soustraction mais avec un sens différent de l'addition de grandeurs physiques.

La **saturation** représente le degré de dilution de la couleur dans la lumière blanche.

La **luminosité** est l'intensité de la lumière achromatique. Elle est mesurable et additive. L'unité de brillance est le candela par mètres carrés ($\mathrm{cd.m^{-2}}$) dont l'unité correspond à 10 nits : 1 $\mathrm{cd^{-2}} = 10$ nits

Un candela, c'est l'intensité lumineuse dans une direction donnée d'une source qui émet un rayonnement monochromatique de fréquence $f = 540.10^{12} Wsd^{-1}$ et dont l'intensité énergétique dans cette direction est de $1.46410^{-3} Wsd^{-1}$. Dans le vide ou l'air, la longueur d'onde est de 555nm. Le tableau suivant donne quelques ordres de grandeur (le minimum minimorum est l'intensité la plus faible que nous puissions détecter à l'oeil) :

soleil à midi	$3.10^8 cd.m^{-2}$
neige au soleil	$310^4 cd.m^{-2}$
lecture normale	$3cd.m^{-2}$
papier blanc en pleine lune	$310^{-3} cd.m^{-2}$
papier blanc sous les étoiles	$310^{-5} cd.m^{-2}$
minimum minimorum	$310^{-7} cd.m^{-2}$

2.1.2 Lois de Grassman

La relation fondamentale exprime la décomposition de la lumière L en une composante chromatique L_λ et une composante achromatique L_w (w pour *white*, blanc en anglais).

Relation fondamentale : $L = L_\lambda + L_w$

1ère loi : $L = L' \Rightarrow kL = kL'$

2ème loi : $L = L' \Rightarrow L + L'' = L' + L''$

Ces lois s'appliquent à toute sensation colorée c'est-à-dire celles de l'arc en ciel (dont on peut déterminer la longueur d'onde dominante) ainsi que les pourpres (dont les verts sont complémentaires).

On définit par couleur complémentaire une couleur dont l'ajout à une autre couleur donne une lumière blanche : L(vert) + L(pourpre) = L_w. Ainsi : L(pourpre) = L_w - L(vert)

2.2 Représentation de la couleur

2.2.1 Les atlas

Une représentation de la couleur nécessite forcément une discrétisation. On peut opérer cette dernière sur des valeurs de teinte, saturation et luminosité. Les premières représentations de la lumière ont été des atlas de couleurs, notamment celui de Munsell en 1929. Les couleurs étaient classées selon leur tonalité, leur clarté et leur saturation. Cet atlas comporte 10 tonalités différentes, 10 niveaux de clarté et différents niveaux de saturation. Plusieurs professions ont développé des atlas : on peut citer le nuancier Pantone pour les encres d'imprimerie, le catalogue Oberthur pour les chrisantémistes, ...

L'atlas de Munsell, par son classement, est un précurseur des systèmes de représentations dans un espace à 3 dimensions, tels que nous les utilisons aujourd'hui. Il existe plusieurs systèmes à 3 dimensions, soit en fonction de grandeur physique telles que la teinte, la saturation et la luminosité, soit en fonction de couleurs (RGB, YCM, ...).

2.2.2 Représentation sur 3 composantes de couleurs

L'espace des couleurs primaires RGB (*Red Green Blue*) également appelé RVB en français (Rouge Vert Bleu) est calqué sur notre perception visuelle. Il utilise trois couleurs de base : le rouge ($\lambda = 700$ nm), le vert ($\lambda = 546$ nm) et le bleu ($\lambda = 435.8$ nm).

L'espace des couleurs secondaires YCM (*Yellow Cyan Magenta*) est basé sur trois couleurs : le jaune, le cyan et le magenta.

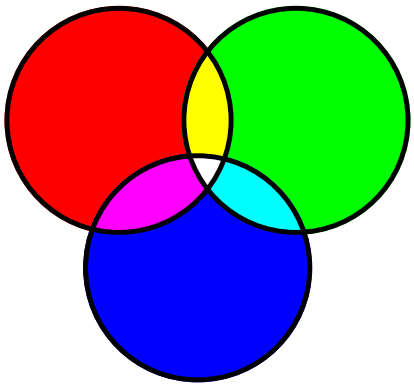


FIG. 2.2 – Synthèse additive des couleurs.

2.2.3 Couleurs additives

Les couleurs additives sont les couleurs primaires : rouge, vert et bleu. On obtient les autres couleurs par addition de ces couleurs : c'est ce qui se produit par exemple lors de la projection : trois faisceaux rouge, vert et bleu en proportions identiques conduisent à une lumière blanche.

Si l'on projette un faisceau rouge et un faisceau bleu, on obtient une lumière magenta ; un faisceau rouge et vert, une lumière jaune ; un faisceau vert et bleu, une lumière cyan. On obtient ainsi les couleurs secondaires par ajout des couleurs primaires, deux à deux. Vu autrement, l'ajout de vert et magenta donne du blanc : ce sont donc des couleurs complémentaires, de même pour le bleu et le jaune ainsi que le rouge et le cyan. Les couleurs primaires et secondaires sont donc complémentaires (voir figure 2.2).

Ce même phénomène se produit sur un moniteur : l'oeil effectue un mélange de 3 points spatialement distincts mais proches (les phosphores) et effectue ainsi une synthèse additive.

2.2.4 Couleurs soustractives

La synthèse soustractive se produit en imprimerie. C'est pourquoi les imprimeurs utilisent les composantes YCM. Si on soustrait la lumière magenta de la lumière blanche (par exemple un filtre), on obtient de la lumière verte. Si on soustrait la lumière cyan, on obtient de la lumière rouge et si on soustrait la lumière jaune, on obtient de la lumière bleue. Si on soustrait à la

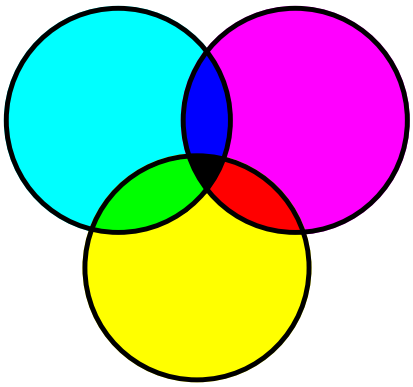


FIG. 2.3 – Synthèse soustractive des couleurs.

fois de la lumière magenta, cyan et jaune (par exemple en superposant trois filtres), on n'obtient plus de lumière, donc du noir (voir figure 2.3).

2.2.5 Couleurs métaméres

On parle de couleurs métaméres lorsque deux signaux ont des spectres différents (voir figure 2.4) mais ne sont pas différenciés par l'oeil. En effet, les cellules réceptrices dédiées à la couleur (les cônes) possèdent un pigment de longueurs d'onde privilégiées qui sont soient dans la zone du vert, soit celle du bleu, soit encore celle du rouge. Ainsi, il se passe une intégration autour de ces trois longueur d'onde primaire et c'est la valeur résultat qui nous fait percevoir la couleur. Dans le cas de la figure 2.4, l'intégration autour de chacune des couleurs primaires est identique pour les deux signaux.

Cette notion est difficile à réaliser. On le comprend mieux lorsque l'on s'intéresse au daltonisme. Du point de vue du daltonien, plusieurs couleurs qui paraissent différentes pour un non daltonien sont perçues de façon identique par le daltonien. En effet, le daltonisme correspond à un défaut d'un des trois types de pigments présents dans les cones : l'intégration se fait uniquement selon 2 composantes au lieu de 3.

2.2.6 Composantes chromatiques

Les composantes chromatiques (r, g, b) d'une lumière (ou valeurs de chromaticité sont les proportions dans lesquelles les couleurs primaires sont mélangées

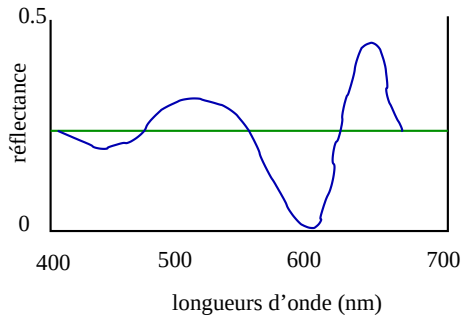


FIG. 2.4 – Les spectres dessinés en vert et en bleu correspondent à des signaux différents mais perçus de façon identique par l'oeil humain.

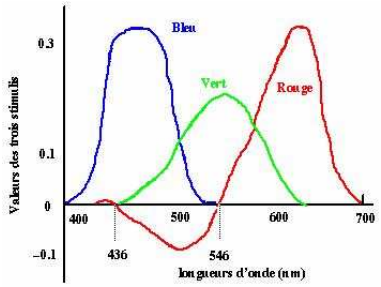


FIG. 2.5 – Les composantes RGB nécessaires pour composer les couleurs de longueurs d'onde dominante dans le domaine du visible.

afin d'obtenir cette lumière, selon la synthèse additive. Une couleur C s'exprime selon :

$$C = rR + gG + bB$$

La figure 2.5 nous permet de déterminer les composantes chromatiques nécessaires à l'obtention d'une couleur de longueur d'onde choisie dans le domaine du visible. Certaines longueurs d'onde vont poser problème car certaines composantes chromatiques vont être négatives : comment enlever une quantité non présente de peinture ? Un compromis consiste à ajouter suffisamment de blanc (donc un mélange de rouge, de vert et de bleu) afin de rendre toutes les composantes chromatiques positives. L'inconvénient est que la couleur obtenue sera dilué : il est donc impossible d'exprimer toutes les couleurs saturées en composantes RBG.

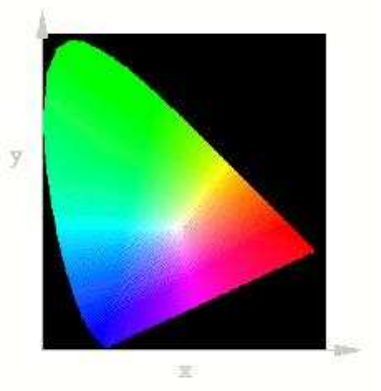


FIG. 2.6 – Diagramme de chromaticité : c'est la projection de l'espace XYZ de la CIE dans le plan x+y+z=1

2.2.7 Diagramme de chromaticité de la CIE

En 1935, la CIE (Commission Internationale de l'Eclairage) a défini un nouveau triplet de couleurs permettant de représenter l'ensemble des couleurs avec des composantes positives. Ce sont les composantes X, Y et Z qui ne représentent pas des couleurs réelles mais répondent aux propriétés suivantes :

- la somme des trois composantes en quantités égales donne du blanc
- la composante Y représente la luminosité
- toute couleur s'exprime comme mélange de X, Y et Z en proportions positives

2.2.8 Autres modèles

La luminance s'exprime par : Y = 0.299 R + 0.587 G + 0.114 B. Il existe plusieurs modèles à trois composantes basés sur la séparation des informations de luminance et chrominance. Un modèle bien connu est le format YUV pour lequel :

- U = 0.493 (B - Y)
- V = 0.877 (R - Y)

Les formats vidéo sont généralement inspirés de ce format et en proposent des variantes :

- Betacam (Y p_b p_r)
  - p_b = 0.5 (B - Y) / (1 - 0.114)

  - p_r = 0.5 (R - Y) / (1 - 0.299)
- Digital Video System (Y C_b C_r)
  - $C_b = 128 + 112\ p_b$
  - $C_r = 128 + 112\ p_r$

Il existe également d'autres modèles à quatre composantes permettant notamment de stocker, en plus des informations de luminance et chrominance, des informations de transparence dans certains cas ou des informations d'echelle de couleur à partir du coefficient γ (voir paragraphe 2.2.9).

D'autres modèles sont bien plus complets mais également plus complexes. On peut évoquer l'imagerie multispectrale ainsi que les des modèles s'intéressant à la physiques des interactions de lumière, séparant les composantes lambertiennes, spéculaires, ambiantes, comme cela est couramment le cas dans le domaine de la synthèse d'images.

2.2.9 Le paramètre γ et le format sRGB

Il vous est surement déjà arrivé ce type d'expérience. Vous vous réjouissez des couleurs éclatantes de vos photographies de vacances que vous regardez sur votre ordinateur personnel. Fier de vous, vous décider d'envoyer votre meilleure photo à vos amis sur Internet. Pendant ce temps-la, vous décidez d'en imprimer un exemplaire sur votre imprimante tandis que vous faites une commande sur papier photo à votre photographe préféré. Mais voila, certains amis vous répondent (sans aucune forme de politesse) que certaines photographies sont jolies mais trop claires ou trop foncées) et vous constatez vous-même que les couleurs sur votre exemplaire imprimé ne correspondent pas à ce que vous aviez à l'écran. C'est à ce moment là que vous découvrez l'utilité du fameux paramètre γ.

Le paramètre γ a été introduit pour prendre en compte le fait que l'intensité mesurée par les appareils photographiques et une fonction concave logarithmique de l'intensité réelle. Les couleurs devraient apparaitre moins saturées. Or, l'intensité rendu par un moniteur CRT est une fonction convexe de l'intensité en entrée. Ainsi les deux phénomènes peuvent se compenser ou bien l'un l'emporte sur l'autre (voir figure 2.8) ; cela dépend du type d'appareil photographique et du type de moniteur.

$$I_{out} = A.{I_{in}}^{\gamma}$$

La figure 2.8 montre un exemple de correction de gamma. Des valeurs plus

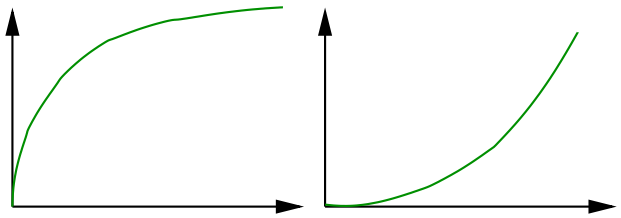


FIG. 2.7 – Les réponses des caméras (à gauche) et des moniteurs (à droite) sont différentes mais suivent un loi γ.

image originale $\gamma = 0.7$ $\gamma = 1.5$

FIG. 2.8 – Exemples de correction gamma.

grandes que 1 conduisent à un éclaircissement des couleurs foncées tandis que des valeurs plus petites que 1 conduisent à un assombrissement des couleurs claires.

[...] Alors, en tatonnant et au prix de quelques impressions, vous obtenez un résultat à peu près satisfaisant (dans ce cas, votre qualité d'évaluateur est généralement proportionnelle à votre patience). Mais attention ! Si à chaque itération vous sauvegardez l'image puis repartez du résultat précédant, vos couleurs seront dégradées. Pour comprendre pourquoi, reportez-vous au chapitre sur la quantification. Et vous commencez à comprendre que votre écran ne restitue pas les couleurs de la même façon que votre imprimante et que c'est encore différent lorsque vous changez d'écran ou d'imprimante.

Le format sRGB[1] est issu de la volonté de normaliser la représentation

[1] http ://www.srgb.com

des couleurs des systèmes RGB et YCMK pour toutes les imprimantes, les moniteurs, ... Dans le format sRGB, les composantes RGB utilisées sont celles qui correspondent aux couleurs sur un moniteur dans des conditions probables d'éclairement avec une valeur $\gamma = 2.2$. C'est une echelle de couleur qui s'approche de notre échelle de perception. C'est cette convention qui a été choisie afin de ne pas avoir à effectuer de transformation lors de la projection sur un moniteur. C'est ce format qui est utilisé dans Java2D.

2.3 Utilisation de l'information couleur

L'information couleur peut être utilisée simplement en considérant la couleur d'un pixel comme un vecteur en trois dimensions. Selon les applications, différents espaces de couleur ne fourniront pas les mêmes résultats. Il convient pour cela de bien cerner le problème et de bien maitriser la représentation de la couleur.

L'information couleur permet dans certains cas de lever des ambiguïtés. Ce n'est pas en détection de contours que son apport est très marqué. Par contre, il est beaucoup plus significatif lors de la détection de régions.

2.4 Perception de la couleur et ses illusions

2.4.1 Grille d'Hermann et bande de Mach

Nous allons nous intéresser à deux illusions bien connues. La figure 2.10 présente l'illusion de la grille d'Hermann. Dès que vous apercevez un point noir et que votre regard est attiré par ce dernier, il disparait. Lorsque vous regardez la grille d'Hermann en périphérie de votre champ de vision, vous distinguez des points noir. Lorsque vous fixez la grille, vous ne voyez pas de points noir.

La première question est : pourquoi voit-on parfois des points noirs ? Cette question est alors suivie de : pourquoi disparaissent-ils lorsqu'on les fixe ?

Pour cela, nous allons nous intéresser maintenant aux cellules ganglionnaires (figure 2.11).

Regardons maintenant l'illusion des bandes de Mach(figures 2.12 et 2.13).

Revenons maintenant à la grille d'Hermann (figure 2.14).

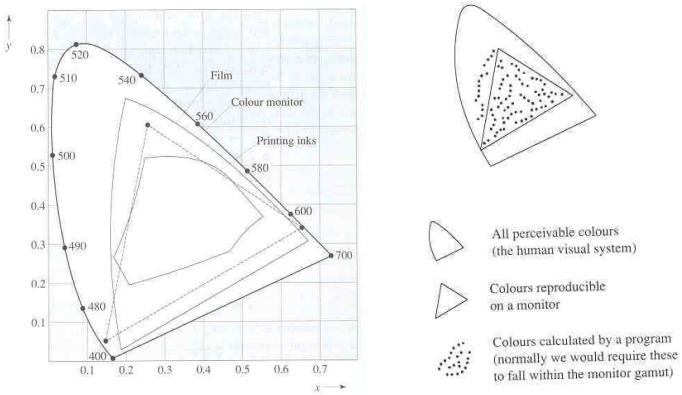


FIG. 2.9 – Toutes les couleurs du diagramme de chromaticité (celles que nous percevons) ne sont pas représentables sur un moniteur. Certaines couleurs sont représentables sur un moniteur mais pas par une imprimante qui elle-même sait représenter des couleurs que ni le moniteur ni les films photographiques peuvent représenter.

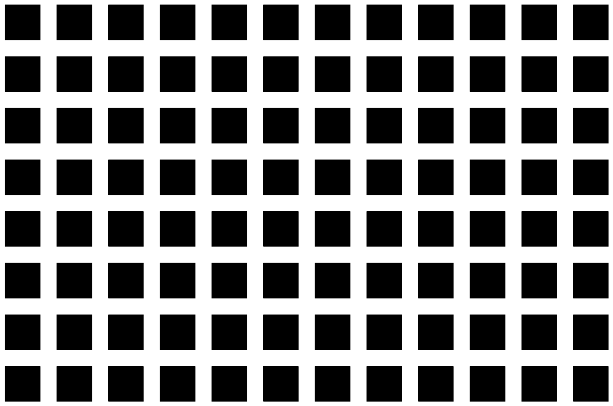


FIG. 2.10 – Illusion de la grille d'Hermann. Comptez les points noirs et les points blances. Combien en trouvez-vous de chaque ?

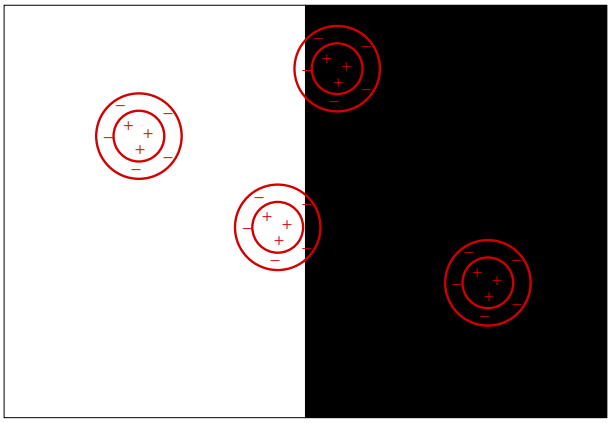


FIG. 2.11 – Certaines cellules ganglionnaires sont de type “ON - center “, c’est-à-dire de centre excitateur et de pourtour inhibiteur.

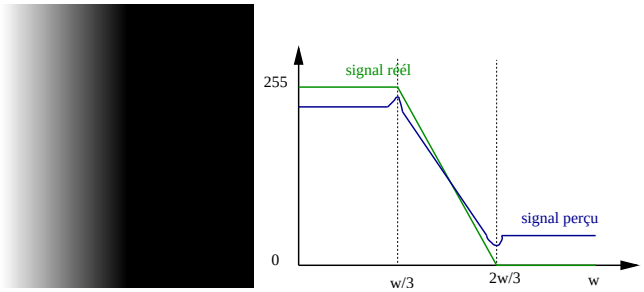


FIG. 2.12 – Bande de Mach

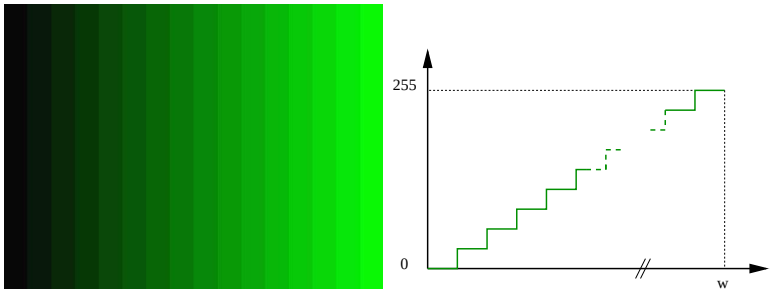


FIG. 2.13 – Bandes de Mach observées sur une image de dégradés de vert (on aurait pu choisir toute autre couleur.

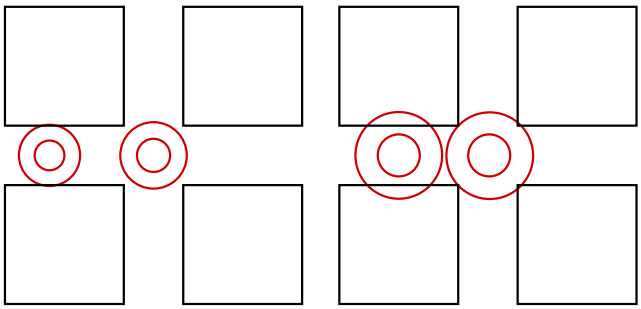


FIG. 2.14 – Les cellules ganglionnaires se sont pas de même dimension dans la fovéa et en périphérie

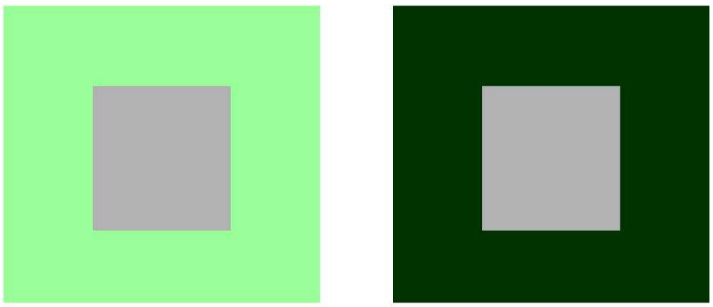


FIG. 2.15 – Les carrés gris de gauche et droite vous paraissent-ils identiques ?

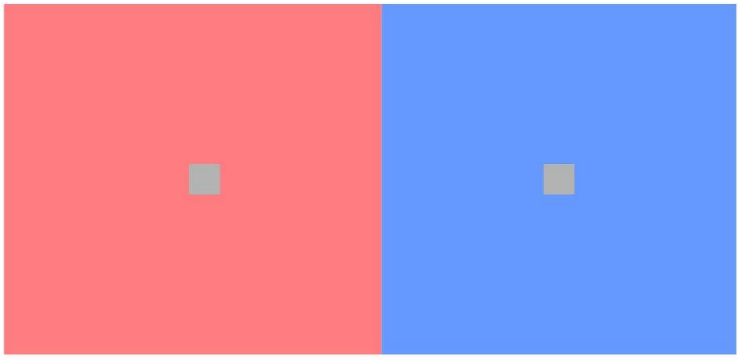


FIG. 2.16 – Les petits carrés de gauche et droite vous paraissent-ils de teinte identique ?

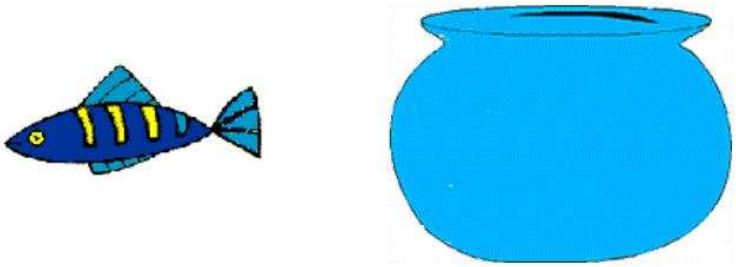


FIG. 2.17 – Illusion du poisson rouge

2.5 Exercices Pratiques

Exercice 1 : Compléter le programme Java du chapitre précédent en ajoutant la possibilité de ne visualiser qu'une composante d'une image : la composante rouge, verte ou bleue. Vous pourrez afficher cette composante soit en niveaux de gris soit en niveaux de cette couleur. Par exemple, la composante rouge d'un pixel (r, g, b) peut se visualiser par (r, r, r) ou (r, 0, 0). Quelle(s) image(s) montre(nt) le plus d'informations ? En général, utilisez les images synthétiques pour vérifier le bon fonctionnement de vos opérateurs.

Exercice 2 : Ecrivez une méthode permettant de convertir une image au format RGB en image au format YUV et inversément. Appliquez successivement ces deux méthodes sur une image et observez les dégradations.

Exercice 3 : Permettez la création d'images synthétiques de dimension celles de la fenêtre actuelle telles que l'intensité varie linéairement en fonction de l'indice de la colonne de :

- noir à rouge pur
- noir à vert pur
- noir à bleu pur

2.6 Bibliographie

- "L'oeil, le cerveau et la vision"par David Hubel, collection l'Univers des Sciences

– Poyntons's Color Technology Page : http ://www.inforamp.net/~ poynton/Poynton-colour.html
– "la couleur en Vision par Ordinateur" par Quan-Tuan Luong, Rapport de Recherche INRIA RR1251, Juin 1990
– International Color Consortium http ://www.color.org

Chapitre 3

Notions de perception visuelle

3.1 Motivations

Les animaux sont dotés de vision leur permettant de se déplacer dans leur environnement, de reconnaître d'autres animaux, des objets, des proies. C'est un système extrêmement complexe et performant.

L'étude de la perception visuelle est intéressante pour le traitement d'images pour deux principales raisons. La première est qu'elle peut nous mettre sur la voie de nouveaux algorithmes reflétant les mécanismes naturels. La seconde est qu'elle nous permet de connaitre les limites de notre perception. Ainsi, il est inutile de représenter plus de couleurs que nous savons en percevoir lors d'une application de visualisation.

On verra enfin que notre système de perception peut également avoir quelques défaillances mises en évidence par des expériences dites d'illusions optiques

Dans un système d'analyse d'image, on distingue la lumière (onde électromagnétique) captée par un récepteur (caméra), transmise par les transmetteurs (cables ou autres) à l'analyseur (l'ordinateur).

On peut effectuer la même décomposition avec la perception visuelle : la lumière est captée par l'oeil. L'information visuelle est transmise via les nerfs et neurones (synapses) vers l'analyseur qu'est le cerveau.

3.2 Anatomie de l'oeil

3.2.1 L'oeil

- La cornée : lentille de l' oeil
- Le cristallin : pour la mise au point
- L'iris : le diaphragme de l'oeil
- Pupille : centre de l'iris (2 à 8 mm)
- Tache aveugle (nerf optique)

3.2.2 Troubles de la vision :

- Myopie : cornée trop courbe, l'image se forme devant la rétine
- Hypermétropie : cornée pas assez courbe, l'image se forme derrière la rétine
- Astigmate : cornée non sphérique
- Presbytie : perte de souplesse de la cornée

3.2.3 La rétine

Lieu du prétraitement de l'information visuelle. Constituée de :

- Cellules photo réceptrices : cônes et bâtonnets
- Cellules bipolaires (premier neurone) : reliées à un ou plusieurs récepteurs
- Cellules ganglionnaires (deuxième neurone) reliées à une ou plusieurs cellules bipolaires
- Nerf optique

3.2.4 Les pigments

Il s'agit d'une substance chimique localisée dans le segment externe.

- rhodopsine : dans les bâtonnets
- iodopsine : dans les cônes (rouge, vert, bleu)

Cônes	Bâtonnets
120 millions	6 millions
macula (fovéa)	périphérie
grande précision	faible précision
forte intensité	faible intensité
100 photons	1 photon
vision couleur	vision monochrome

Les cellules bipolaires sont reliées à un seul récepteur (dans la fovéa) ou à un groupe de récepteurs (en périphérie) : il y a donc une acuité visuelle maximale dans la fovéa.

Les cellules ganglionnaires sont reliées à une seule cellule bipolaire dans la fovéa. Les axones sont les fibres du nerf optique. Les petites cellules X ont une réponse tonique tandis que les grandes cellules Y ont une réponse phasique : elles permettent une bonne détection du mouvement

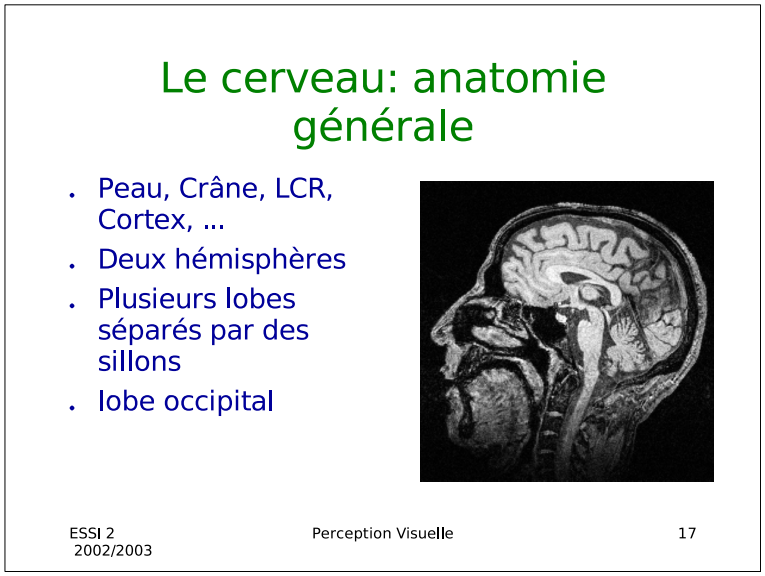
Quelques données numériques :

- 140 millions de photorécepteurs
- globe oculaire : 24.5 mm, 5.5 cm3, 7.5 g
- Epaisseur de la cornée 0.54 mm au centre ; 0.65 en périphérie, diamètre 11.5
- lentille (cristallin) = 4 mm d épaisseur, diamètre 9 mm
- densité de cônes dans la fovéa = 200 000 par mm2
- Nombre de fibres dans le nerf optique = 1 200 000
- Nombre de cellules dans le cortex visuel = 538 000 000
- Plus grande densité de bâtonnets = 160 000 par mm2
- Densité de cônes dans la fovéa = 200 000 mm2
- Diamètre de la fovéa = 1.5 mm
- Aire de la rétine = 2 500 mm2
- Epaisseur de la rétine = 120 microns

3.3 Anatomie du cerveau

```
De la rétine au cortex visuel :
        Séparation des flux optiques entre les différentes hémisphères
        chiasme
        corps genouillé latéral

Conséquence de lésions
```




```
Le cortex visuel
Dans le lobe occipital
Différentes aires: V1 - V4, MT
V1 : aire primaire:
        organisée en colonnes (1mm2)
        direction  : D-G-D-G-~
        direction > : détection lumière directionnelle
V1 niveau + complexe: détection de toute direction
V3 : forme
V4 : couleur
MT ou V5: mouvement
```

3.4 Etude du fonctionnement du système visuel

```
Signaux électromagnétiques
Neurones => activité électromagnétique
```

Etude du fonctionnement du système visuel

- Chez la mouche, le singe, …
- Chez l'homme
- Etude des pathologies
- Approches invasives / non invasives
- Mesures de signaux électriques / débits sanguins
- Utilisation de stimuli visuels

26

Signaux électromagnétiques

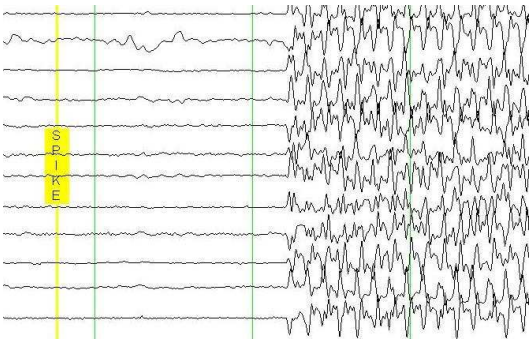
- Neurones => activité électromagnétique
- Potentiel électrique à la surface du scalp : EEG (ElectroEncephaloGraphie)
- Champ magnétique à l'extérieur du crâne : MEG (MagnetoEncephaloGraphie)
- Electrodes profondes

EEG : Electro-EncephaloGraphie

. Mesure de potentiels à la surface du scalp

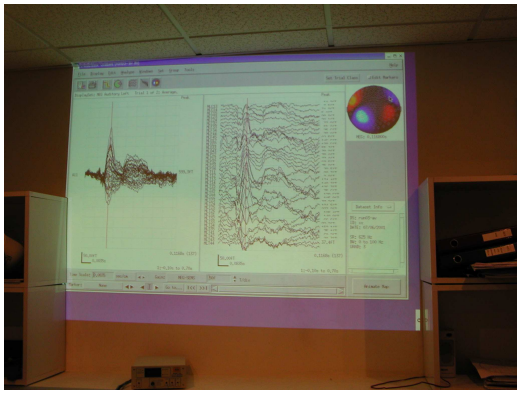
signal EEG

. typiquement: 64 canaux



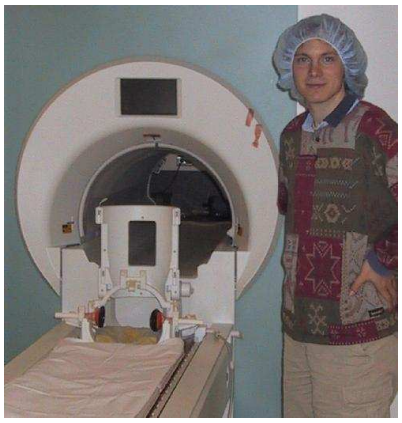
Acquisition MEG

Résultat



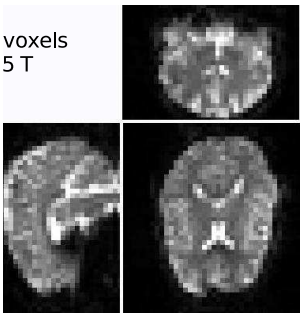
Mesures de débits sanguins

- IRM fonctionnelle



Acquisition IRMf

Images volumiques
typiquement :65x65x25 voxels
résolution 5x5x5 mm, 1.5 T



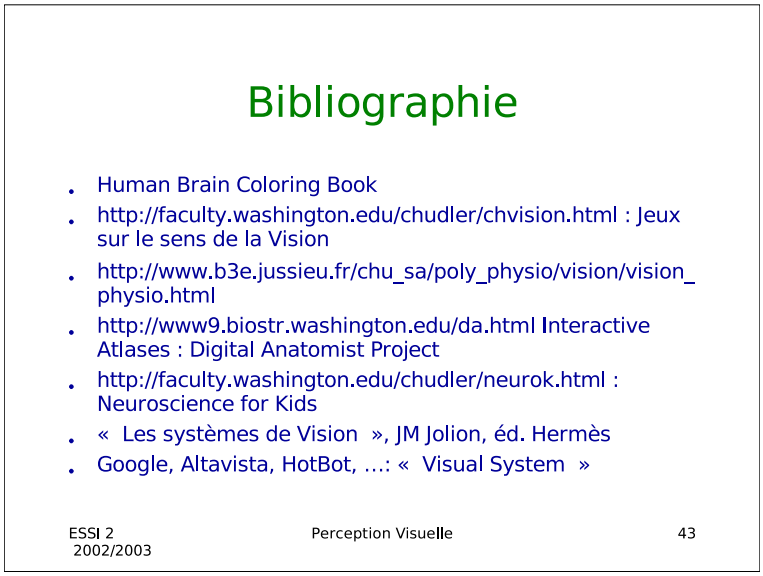
- stimulis
- images de référence
- tests statistiques

ESSI 2
2002/2003

Perception Visuelle

35

Comparaison des différents types de signaux cérébraux




```
Potentiel électrique à la surface du scalp : EEG (ElectroEncephaloGraphie)
Champ magnétique à l'extérieur du crâne : MEG (MagnetoEncephaloGraphie)
Electrodes profondes

Reconstruction de l'activité
        localisation de dipôles
        modèle de sources (e.g. dipôles)
        modèle de tête
        méthodes de résolution numériques

PET = Positron Emission Tomographie
        Utilisation de marqueurs radioactifs (glucose, ...)
        => nécessité d'un cyclotron
        Détection des rayons gamma

Comparaison des différents types de signaux cérébraux
        EEG et MEG: très bonne résolution temporelle, mauvaise résolution spatiale
        IRMf et PET: bonne résolution spatiale, mauvaise résolution temporelle
        électrodes profondes: bonnes résolutions temporelle et spatiale ~ mais invasi
        => fusion des données non invasives
```

Chapitre 4

Notions d'échantillonnage et de quantification

L'échantillonnage et la quantification sont deux opérations permettant de numériser une image. Mais pourquoi cherche t-on à numériser les images ? Tout simplement pour les visualiser sur un moniteur, les imprimer, les traiter sur un ordinateur, les stocker sur des supports informatiques ou les transmettre sur des réseaux informatiques.

L'échantillonnage concerne la discrétisation de l'espace 2D : c'est le nombre de points que l'on va pouvoir colorier. La quantification concerne la discrétisation de l'espace des couleurs ou niveaux de gris : c'est le nombre de crayons de couleur différentes que l'on va pouvoir utiliser pour dessiner notre image. Lors de la numérisation, on cherche à conserver une qualité maximale (un nombre de pixels maximal, un nombre de couleurs maximal) et à obtenir des données les moins volumineuses possibles. Le problème est que ces deux besoins sont antagonistes. Il conviendra alors de chercher la résolution et le nombre de niveaux de couleurs satisfaisants les besoins. Par exemple, si le but est uniquement la visualisation sur un moniteur donné, il est inutile d'échantillonner l'image à une résolution supérieure à celle du moniteur.

L'échantillonnage et la quantification ne sont pas limités à la numérisation d'une image. Ce sont des techniques que l'on utilise également sur des images déjà numérisées, afin de modifier la résolution (on rééchantillonne l'image) ou le nombre de couleurs utilisées (quantification). Cependant, il est dans ce cas fait appel à des techniques de reconstruction d'images, c'est-à-dire de retrouver un signal 2D continu en espace et/ou en couleur.

La figure 4.1 présente des exemples d'échantillonnage à différentes résolutions

FIG. 4.1 – Echantillonnage et Quantification d'un signal 2D continu selon différentes résolutions spatiales et colorimétriques.

et de quantification sur différents nombres de bits.

4.1 Echantillonnage

Comment passer d'un signal continu à un signal discret ? La réponse technologique pour les images consiste à utiliser un appareil numérique (ce sont alors les capteurs CCD qui numérisent le signal) ou un scanner pour numériser des photos. La réponse théorique à cette question est la théorie de l'échantillonnage.

Pour commencer, donnons quelques définitions :

résolution verticale : nombre de lignes dans l'image

résolution horizontale : nombre de colonnes dans l'image

résolution spatiale : = résolution verticale x résolution horizontale

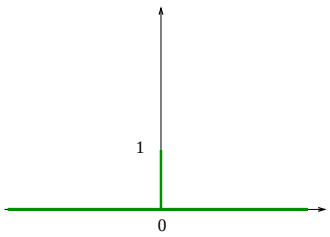


FIG. 4.2 – Echelon de Dirac mono-dimensionnel

densité de résolution : nombre de pixels par unité de longueur. S'exprime en ppi (pixels per inch) ou dpi (dots per inch).

4.2 Rappels théoriques pour la reconstruction

Nous présentons ici les éléments de théorie du signal dont nous avons besoin pour la reconstruction d'images. Cette partie est inspirée de [6].

4.2.1 Dirac

Nous rappelons la définition d'une impulsion de Dirac en dimension 1 (voir figure 4.2) :

$$\begin{cases} \delta(x) = 1 & \text{si } x = 0 \\ \delta(x) = 0 & \text{sinon} \end{cases}$$

En dimension 2, il s'agit du produit de deux impulsions mono-dimensionnelles : $\delta(x, y) = \delta(x).\delta(y)$ (voir figure 4.3) :

$$\begin{cases} \delta(x, y) = 1 & \text{si } x = 0 \text{ et } y = 0 \\ \delta(x, y) = 0 & \text{sinon} \end{cases}$$

Un pixel de position (x,y) et de valeur I(x,y) peut alors être défini comme une impulsion de Dirac de dimension2, centrée en (x,y) et d'amplitude I(x,y). Une image en dimension 2, de dimensions w×h est un ensemble de pixel de position (x,y) avec x et y, deux entiers variant respectivement de 0 à w et de 0 à h.

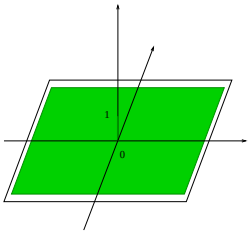


FIG. 4.3 – Echelon de Dirac bi-dimensionnel

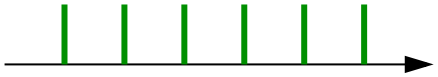


FIG. 4.4 – Peigne d'impulsions de Dirac en dimension 1 :
$p(x) = \sum_{m=-\infty}^{\infty} \delta(x - m\Delta x)$

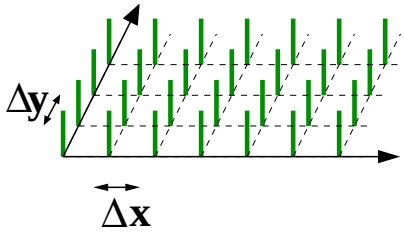


FIG. 4.5 – Brosse d'impulsions de Dirac en dimension 2 :
$b(x,y) = \left(\sum_{m=-\infty}^{\infty} \delta(x - m\Delta x) \right) . \left(\sum_{n=-\infty}^{\infty} \delta(y - n\Delta y) \right)$

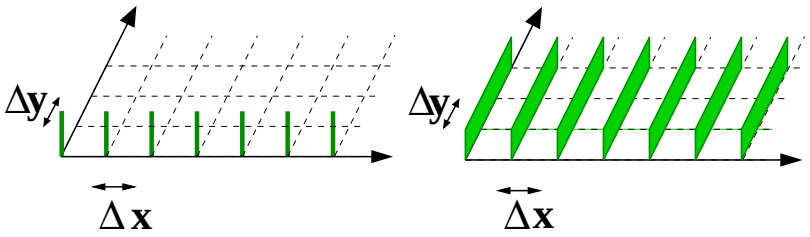


FIG. 4.6 – A gauche : peigne 2D en X. A droite : peigne étendu en X

Tandis que l'échantillonnage en dimension 1 s'effectue par le produit avec un peigne d'impulsions de Dirac (voir figure 4.4), l'échantillonnage en dimension 2 s'effectue par le produit avec une brosse d'impulsions de Dirac (voir figure 4.5).

En dimension 2, le peigne en x est défini par :

$$p_x(x,y) = \delta(y) \sum_{m=-\infty}^{\infty} \delta(x - m\Delta x)$$

tandis que le peigne étendu, en x est défini par :

$$p_x(x,y) = \sum_{m=-\infty}^{\infty} \delta(x - m\Delta x)$$

On remarquera que cette équation ressemble fortement à celle du peigne en dimension 1 mais qu'il ne faut pas confondre ces deux peignes (voir figure 4.6).

La brosse est définie par le produit de 2 peignes étendus :

$$\begin{cases} p_x(x,y) = \sum_{m=-\infty}^{\infty} \delta(x - m\Delta x) \\ p_y(x,y) = \sum_{m=-\infty}^{\infty} \delta(y - n\Delta y) \\ b(x,y) = p_x(x,y)p_y(x,y) \end{cases}$$

Notons $f(x,y)$ le signal image continu et $g(x,y)$ le signal image échantillonné par le produit avec une brosse :

$$g(x,y) = f(x,y).b(x,y) = f(x,y).\left(\sum_{m=-\infty}^{\infty} \delta(x - m\Delta x)\right).\left(\sum_{n=-\infty}^{\infty} \delta(y - n\Delta y)\right)$$

Puisque l'on sait maintenant comment échantillonner une image continue, on s'intéresse au rééchantillonnage. Celui-ci passe par une phase de reconstruction préalable à un nouvel échantillonnage.

4.3 Reconstruction d'image

La reconstruction d'image consiste à retrouver l'image qui a été échantillonnée. Cela est impossible étant donné que, lors de l'échantillonnage, on a perdu de l'information. On cherche donc à retrouver au mieux ce signal.

4.3.1 Principe de la reconstruction

La méthode classique consiste à reconstituer la transformée de Fourier du signal continu à partir de la transformée de Fourier du signal discrétisé.

Commençons par déterminer la transformée de Fourier de la brosse. C'est la convolution des transformées des peignes étendus :

$$b(x,y) = p_x(x,y)p_y(x,y) \Rightarrow B(u,v) = P_x(u,v) * P_y(u,v)$$

On s'intéresse donc aux transformées des peignes étendus :

$$p_x(x,y) = \sum_{m=-\infty}^{\infty} \delta(x - m\Delta x) \Rightarrow P_x(u,v) = \delta(u) \sum_{k=-\infty}^{\infty} \delta(u - \frac{k}{\Delta x})$$
$$p_y(x,y) = \sum_{n=-\infty}^{\infty} \delta(y - n\Delta y) \Rightarrow P_y(u,v) = \delta(u) \sum_{l=-\infty}^{\infty} \delta(v - \frac{l}{\Delta y})$$

qui sont elle mêmes des peignes en dimension 2. La transformée de Fourier d'un brosse de pas Δx et Δy est la convolution de deux peignes : c'est une brosse de pas $\frac{1}{\Delta x}$ et $\frac{1}{\Delta y}$.

La convolution avec une brosse revient à une somme de convolutions avec des impulsions de Dirac décalées :

$$G(u,v) = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} F(u - \frac{k}{\Delta x}, v - \frac{l}{\Delta y}) \tag{4.1}$$

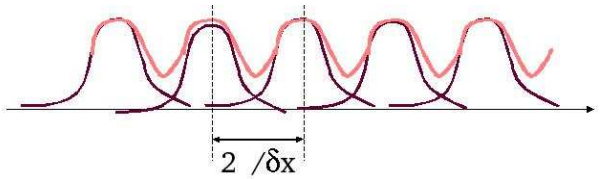


FIG. 4.7 – cas de repliement spectral. Le signal original est le signal violet. Le signal obtenu par la somme des signaux translatés fera apparaître des hautes fréquences surestimées.

4.3.2 Repliement spectral (*aliasing*)

L'équation 4.1 fait apparaitre une somme de termes obtenues par translations de $F(u)$. Si, à partir de $G(u)$, on cherche à déterminer $F(u)$, alors, il est nécessaire que le pas de translation soit inférieure au domaine de valeurs non nulles de $F(u)$. Si tel n'est pas le cas, des défauts apparaissent lors de la reconstruction : il s'agit d'effet de Gibbs ou halo, des détails supplémentaires erronés. En traitement du signal, on parle de repliement spectral. Une illustration que tout le monde connait bien est donnée dans les Western où les roues des roulottes paraissent tourner dans le mauvais sens (on voit également, mais moins fréquemment, ce phénomène dans des films plus récent au niveau des roues des voitures). A la télévision, une chemise aux rayures trop serrées fera également apparaître des motifs erronés. Cela s'explique car les fréquences hautes sont surestimées et donc des signaux de hautes fréquences sont ajoutés (voir figure 4.7). La qualité de la reconstruction dépend ainsi des pas d'échantillonnage : on veut $\frac{1}{\delta x}$ et $\frac{1}{\delta y}$ grands donc δx et δy petits ce qui signifie beaucoup de capteurs peu espacés.

4.3.3 Transformée de Fourier d'une image

Ce paragraphe constitue une parenthèse dans ce chapitre. On parle ici de transformée de Fourier d'une image. Mais que peut-on voir dans une image de la transformée de Fourier ? On y voit le spectre de l'image. Dans le cas d'une image périodique, on distinguera des raies correspondant aux fréquences verticales et horizontales. C'est le cas de la figure 4.8. Pour une photographie réelle, l'image de la transformée est plus difficile à interpréter à l'oeil nu. La figure 4.9 montre un exemple. Le repère de ces images est

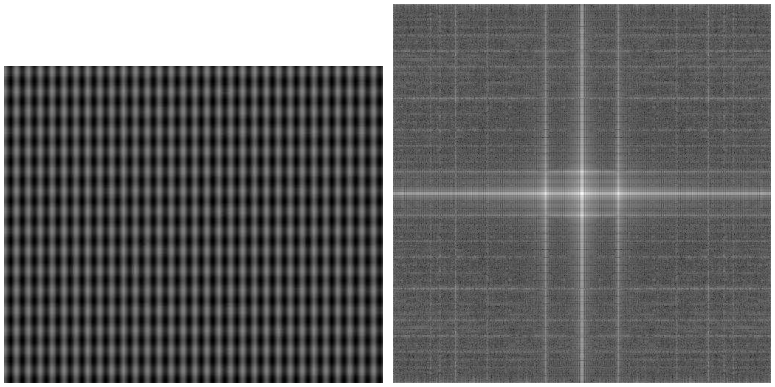


FIG. 4.8 – Image synthétique périodique (sinusoïdale) et sa transformée de Fourier.

construit de telle sorte que le centre de l'image correspond au point $(0; 0)$. On observe donc les basses fréquences au centre de l'image et les hautes fréquences en périphérie.

Le filtrage passe-bas se comprend très bien sur l'image de la transformée de Fourier : seule la partie centrale de l'image est conservé. Il suffit alors d'effectuer la transformée de Fourier inverse pour obtenir le résultat. La figure 4.10 (resp. 4.11) présente un exemple de filtre passe-bas sur l'image de la figure 4.8 (resp. 4.9). La figure 4.11 met en évidence les défauts des filtres passe-bas : on observe, dans les zones unies (costume du Père Noël, mur blanc) des oscillations.

Les illustrations de ce paragraphe ont été obtenues en utilisant l'algorithme de transformée de Fourier rapide disponible sous forme d'applet sur l'excellent site *telesun*[1] de l'INSA de Lyon.

4.3.4 Théorème de Nyquist (1928) - Shannon (1948)

Le théorème de Nyquist et Shannon précise les conditions de validité de la reconstruction afin de s'affranchir des problèmes de repliement spectral :

- La transformée de Fourier est bornée
- La fréquence d'échantillonnage doit être au moins égale au double de la fréquence maximale de l'image.

[1] http ://telesun.insa-lyon.fr/~telesun/Traitement.L04/fft2D.html

FIG. 4.9 – Image naturelle et sa transformée de Fourier.

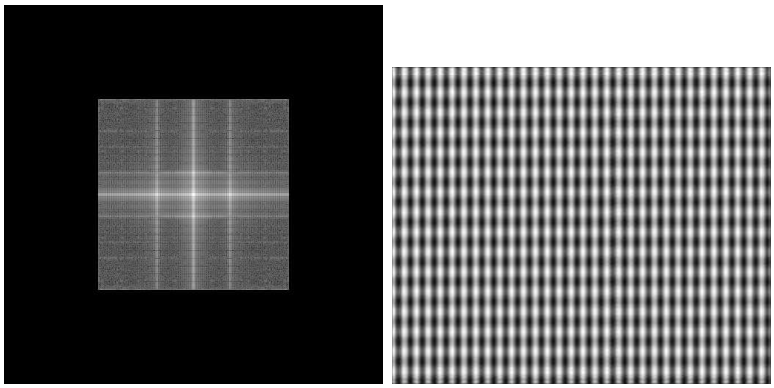


FIG. 4.10 – Filtre passe-bas vu sur la transformée de Fourier de l'image synthétique périodique et sa transformée inverse.

FIG. 4.11 – Filtre passe-bas vu sur la transformée de Fourier de l'image du Père Noël et sa transformée inverse.

Par exemple, en vidéo, un capteur de 768 pixels par lignes, avec une durée de ligne de 52μs donne une fréquence d'échantillonnage minimale de $768/52.10^{-6} = 14.75$ MHz. La fréquence de repliement est dans ce cas de 7.37 MHz.

Afin de respecter les conditions de ce théorème, on peut soit effectuer un filtrage passe-bas de l'image afin d'éliminer les hautes fréquences de celle-ci et ainsi réduire sa fréquence maximale. Cependant, l'image perdre des détails et aura un aspect plus flou. On peut également augmenter le pas d'échantillonnage. Il faut cependant vérifier que le spectre est borné. Le suréchantillonnage consiste à avoir plus d'échantillons que de pixels.

Il est intéressant de constater que justement, les caméras et appareils photos du commerce utilisent un filtrage passe-bas optique en utilisant les propriétés de biréfringence des lames de quartz. Il en résulte une atténuation des contours corrigée par un filtre rehausseur de contours.

Sur les capteurs CCD, seul 1/3 de la surface est utilisée pour capter la lumière. Afin d'améliorer la reproduction des détails fins de l'image, les capteurs "verts"sont quelques fois décalés de 1/2 pixel par rapport aux capteurs "rouges"et "bleus".

Reconstruction CRT (*Cathodic Ray Tube*) : la réponse des phosphores est une gaussienne. Plus le spot est large, plus la transformée de Fourier est étroite est on perd les hautes fréquences et atténu les basses fréquences.

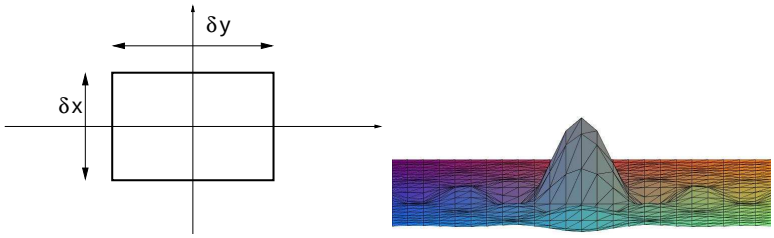


FIG. 4.12 – Domaine fréquentiel. Domaine spatial.

4.3.5 Filtre de reconstruction idéal

h, filtre de recontruction idéal :

$$h(x,y) = \frac{\sin \pi x}{\pi x}.\frac{\sin \pi y}{\pi y}$$

f, fonction reconstruite :

$$f(x,y) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} g(m,n) \frac{\sin \pi(x-m)}{\pi(x-m)}.\frac{\sin \pi(y-n)}{\pi(y-n)}$$

Il s'agit d'une filtre d'étendue spatiale infinie. Or une image est de dimensions bornées. La somme est infinie ce qui signifie que chaque pixel contribu au calcul d'un point. Il en suit des difficultés de calculs.

4.3.6 Approximations du filtre idéal

Si on tronque la somme infinie en négligeant des termes, il apparait des phénomènes de Gibbs (ondulations erronées).

Regardons sur la figure 4.13 les défauts des filtres de recontruction :

Approximation à l'ordre 0 L'approximation la plus grossière du sinus cardinal est un simple créneau dont la transformée de Fourier est elle-même ... un sinus cardinal.

Approximation à l'ordre 1 Ce filtre est simple à calculer. Les hautes fréquences sont atténuées et il produit donc des artefacts.

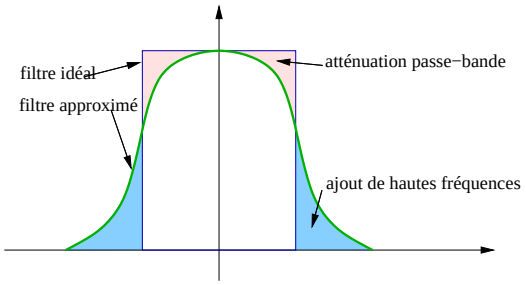


FIG. 4.13 – Les défauts des filtres de reconstruction

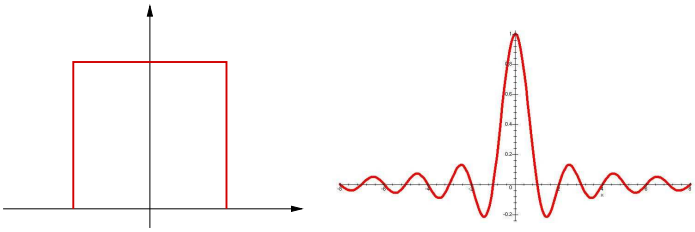


FIG. 4.14 – Approximation à l'ordre 0 du filtre de reconstruction idéal. A gauche : domaine spatial. A droite : domaine fréquentiel.

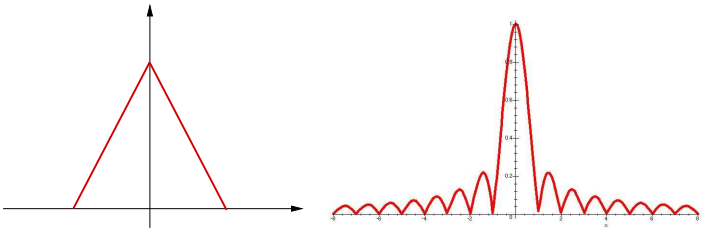


FIG. 4.15 – Approximation à l'ordre 1 du filtre de reconstruction idéal. A gauche : domaine spatial. A droite : domaine fréquentiel.

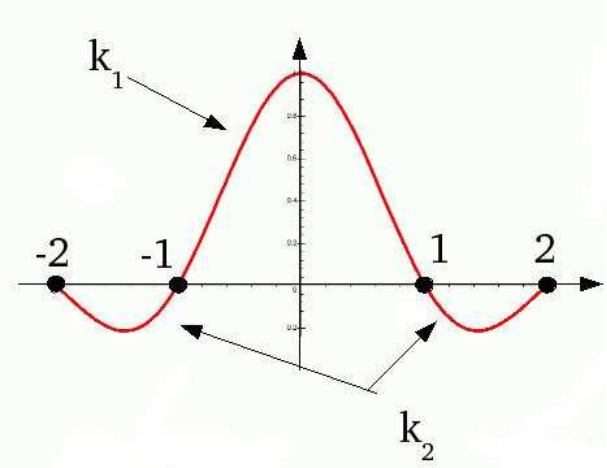


FIG. 4.16 – Approximation par polynômes de Mitchell du sinus cardinal

Approximation à l'ordre 3 Il s'agit de l'interpolation par des splines, selon la famille des polynômes de Mitchell. Ces polynômes sont définis par morceaux tels que :

$$k(x) = \begin{cases} k_1(x) = A_1|x|^3 + B_1x^2 + C_1|x| + D_1 & |x| \leq 1 \\ k_2(x) = A_2|x|^3 + B_2x^2 + C_2|x| + D_2 & 1 \leq |x| \leq 2 \\ 0 & |x| \geq 2 \end{cases}$$

avec des contraintes :

- de symétrie : $k(-x) = k(x)$
- de continuité : $k_1(1) = k_2(1)$ et $k_2(2) = 0$
- de continuité C^1 : $k_1'(0) = 0$, $k_1'(1) = k_2'(1)$ et $k_2'(2) = 0$
- de somme unitaire : $\sum k(x-n) = k_2(1+\epsilon)+k_1(\epsilon)+k_1(\epsilon-1)+k_2(\epsilon-2) = 1$ avec $0 < \epsilon < 1$

Utilisons maintenant ces contraintes :

$k_1(1) = k_2(1) \Rightarrow A_1 + B_1 + C_1 + D_1 = A_2 + B_2 + C_2 + D_2$ (1)
$k(2) = 0 \Rightarrow 8A_2 + 4B_2 + 2C_2 + D_2 = 0$ (2)
$k_1'(0) = 0 \Rightarrow C_1 = 0$ (3)
$k_2'(2) = 0 \Rightarrow 12A_2 + 4B_2 + C_2 = 0 => C_2 = -12A_2 - 4B_2$ (4)
$k_1'(1) = k_2'(1) \Rightarrow 3A_1 + 2B_1 = -9A_2 - 2B_2$ (5)
(2) et (4) $\Rightarrow D_2 = -8A_2 - 4B_2 + 24A_2 + 8B_2 = 16A_2 + 4B_2$
(1) et (2) $\Rightarrow A_1 + B_1 + D_1 = A_2 + B_2 - 12A_2 - 4B_2 + 16A_2 + 4B_2 = 5A_2 + B_2$
$\Rightarrow D_1 = 5A_2 + B_2 - A_1 - B_1$
$\sum k(x - n) = 1 \Rightarrow 9A_2 + 5B_2 + 3C_2 + 2D_2 + A_1 + B_1 + 2D_1 = 1$

d'où : $15A2 + 3B2 - A1 - B1 = 1$ soit :

$$A_1 = -39A_2 - 8B_2 + 2 \quad B_1 = 54A_2 + 11B_2 - 3 \quad C_1 = 0$$
$$D_1 = -10A_2 - 2B_2 + 1 \quad C_2 = -12A_2 - 4B_2 \qquad D_2 = 16A_2 + 4B_2$$

En posant : $6A_2 + B_2 = -C$ et $5A_2 + B_2 = B/6$, on obtient :

$$k(x) = \frac{1}{6}\begin{cases} (12 - 9B - 6C)|x|^3 + (-18 + 12B + 6C)x^2 + (6 - 2B) & |x| \leq 1 \\ -(B + 6C)|x|^3 + (6B + 30C)x^2 - (12B + 48C)|x| + (8B + 24C) & 1 \leq |x| \leq 2 \\ 0 & 2 \leq |x| \end{cases}$$

Voici quelques exemples :
- si B=0 alors k(0)=1 et k(1)=0 comme sinc
- Cardinal splines : B=0 ; C=-a
- Catmull-Rom Spline : B=0 ; C=0.5
- Cubic B-spline : B=1 ; C=0

Cette approximation donne de meilleure résultat qu'une troncature d'un sinus cardinal grace à la continuité C^1 de ces polynômes (les discontinuités faisant apparaître des détails erronés, souvent appelés *ripples*).

Nous étudierons plus en détails les applications de la reconstruction d'images et notamment l'interpolation dans le chapitre concernant les transformations 2D (chapitre 5).

4.4 Quantification

Rappelons qu'il s'agit de discrétiser l'ensemble des niveaux de gris ou couleurs de l'image. Imaginons que l'on cherche à colorier la partie gauche de la figure 4.17 selon le modèle présenté dans la partie droite.

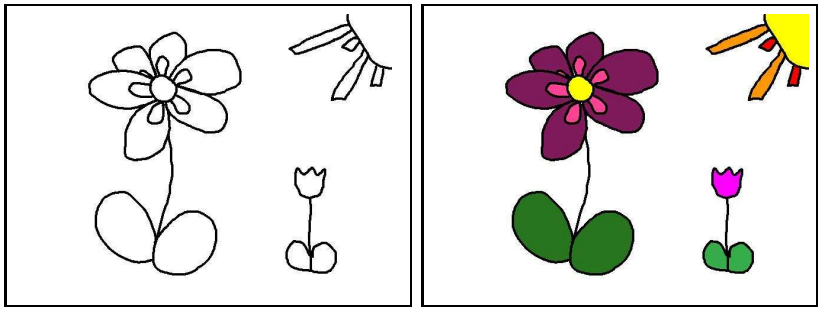


FIG. 4.17 – A gauche : coloriage à colorier. A droite : modèle pour le coloriage.

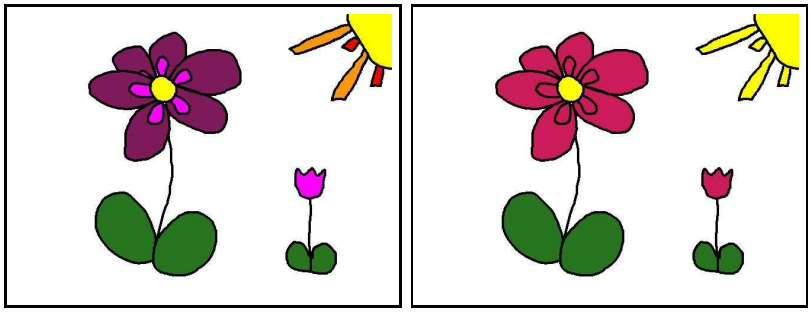


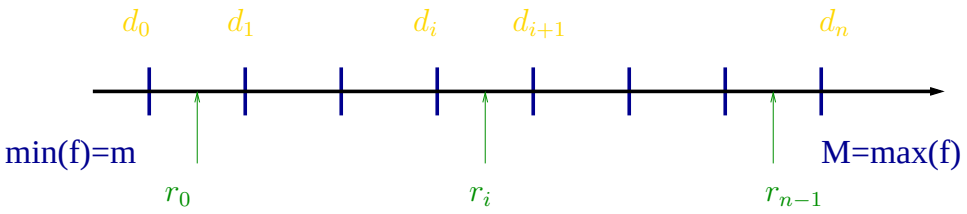
FIG. 4.18 – A gauche : résultat obtenu avec 6 couleurs différentes. A droite : résultat obtenu avec 3 couleurs seulement.

Si l'on nous permet de choisir 8 couleurs, on obtiendra le résultat escompté. Si on ne peut qu'utiliser 6 couleurs seulement, ou même 3 couleurs seulement, on va instinctivement essayer d'utiliser la même couleur pour des couleurs du modèle qui se ressemble (voir figure 4.18).

Le principe consiste à remplacer toute valeur située entre deux niveaux de décision consécutifs d_i et d_{i+1} par un niveau de reconstruction r_i (voir figure 4.4).

Si on considère les niveaux de gris d'une image entre 0 et 255, cela consiste à découper l'intervalle [0 ;255] en plusieurs intervalles et, pour chacun d'eux, déterminer le niveau de reconstruction. Le problème consiste alors à savoir comment on découpe notre intervalle et, comment, pour chaque intervalle on va déterminer le niveau de reconstruction.

Pour résoudre ce problème, on va ajouter une contrainte : on souhaite que la nouvelle image ressemble le plus possible à l'image initiale. On mesure



l'erreur entre ces deux images. Soit f la fonction des intensités de l'image initiale et g la fonction des intensités de l'image quantifiée. On considère que les valeurs de f sont des variables aléatoire de densité de probabilité $p(f)$. On note m (respectivement M) la valeur minimale (respectivement maximale) pouvant être atteinte par f.

$$\mathcal{E} = E\{(f-g)^2\} = \int_m^M (f-g)^2 p(f) df = \sum_{i=0}^{n-1} \int_{d_i}^{d_{i+1}} (f-r_i)^2 p(f) df$$

L'hypothèse que nous allons faire pour la suite consiste à dire que la densité de probabilité de chaque niveau de quantification est constant et vaut $P(r_i)$. Cette hypothèse va nous simplifier fortement les calculs. Elle peut être assez réaliste en fonction des intervalles choisis.

L'expression de l'erreur devient :

$$\mathcal{E} = \sum_{i=0}^{n-1} p(r_i) \int_{d_i}^{d_{i+1}} (f-r_i)^2 df$$

On peut maintenant intégrer :

$$\mathcal{E} = \frac{1}{3} \sum_{i=0}^{n-1} p(r_i) \left[(d_{i+1}-r_i)^3 - (d_i - r_i)^3 \right]$$

En minimisant par rapport à r_i, on obtient :

$$\frac{\partial \mathcal{E}}{\partial r_i} = 0 \Longrightarrow r_i = \frac{d_i + d_{i+1}}{2}$$

d'où :

$$\mathcal{E} = \frac{1}{12} \sum_{i=0}^{n-1} p(r_i)(d_{i+1} - d_i)^3$$

Comment placer les niveaux de décision ? Panter et Dite (1951)répondent :

$$\begin{aligned} \mathcal{E} &= \frac{1}{12}\sum_{i=0}^{n-1} p(r_i)(d_{i+1}-d_i)^3 \\ d_i &= \frac{(M-m)\int_m^{a_i}[p(f)]^{-\frac{1}{3}}df}{\int_m^M [p(f)]^{-\frac{1}{3}}df} \\ a_i &= \frac{i(M-m)}{n} + m \end{aligned} \tag{4.2}$$

Si l'on fait maintenant une hypothèse de densité de probabilité uniforme (on considère alors que tous les niveaux de gris sont équiprobables), on obtient des intervalles de quantification de taille constante et des niveaux de quantification centrés. Voila qui est bien plus facile à calculer !

Cependant, il faut bien conserver à l'esprit que ceci n'est pas toujours vrai. Effectivement, certaines images sont plutôt sombres, d'autres plutôt clair. Imaginez que vous deviez reproduire un tableau mais que vous n'ayez le droit de n'utiliser qu'on nombre déterminé de feutres. Suivant les couleurs utilisées dans le tableau, vous n'allez pas choisir les mêmes feutres. Pour un paysage de campagne verte, vous allez choisir différentes nuances de vert, pour une scène maritime, différentes nuances de bleus, ... Dans le cas général, on peut s'en sortir, à condition de connaitre les différentes probabilités (voir le paragraphe sur les histogrammes), avec des formules récurrentes. Il existe également un certain nombre de tables lorsque les densités de probabilités sont de type gaussiennes, poissons, ...

Un exemple de format est le format SVGA pour lequel chaque composante est quantifiée sur 5 bits.

4.5 Quantification uniforme

4.5.1 Quantification des niveaux de gris

Le nombre de niveaux dépend de la représentation que l'on peut avoir de ces niveaux. Celle-ci est effectuée à partir d'un certain nombre de bits n : on dispose alors de 2^n niveaux. Pour des niveaux de gris représentés sur 1 octet, on dispose de 256 niveaux de gris différents. Il faut relier ceci à la capacité de l'oeil humain qui est de 10 à 15 niveaux en absolu et beaucoup plus en relatif. Les photos en niveaux de gris représentées sur 8 bits sont donc de qualité tout à fait satisfaisante pour nos yeux.

La figure 4.19 illustre la méthode de quantification uniforme sur n bits. L'intervalle de niveaux de gris $[0; 255]$ est découpé en 2^n intervalles $[k * \frac{256}{2^n} -$

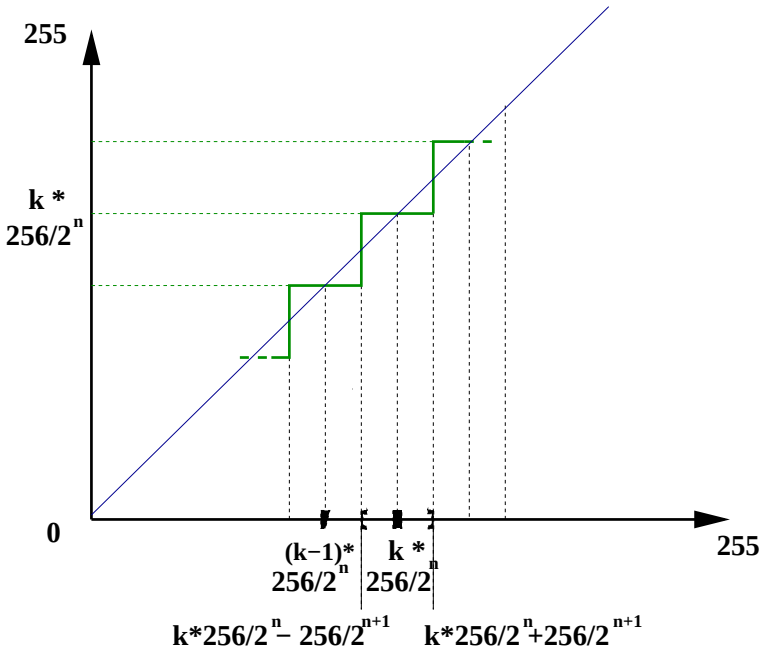


FIG. 4.19 – Quantification uniforme sur n bits. Toutes les valeurs situées dans l'intervalle $[k*256/2^n - 256/2^{n+1}; k*256/2^n + 256/2^{n+1}]$ sont remplacées par $k*256/2^n$.

$\frac{256}{2^{n+1}}; k*\frac{256}{2^n} + \frac{256}{2^{n+1}}]$ de longueurs identiques $\frac{256}{2^n}$. Comme l'a montré la théorie, toute valeur situé dans un de ces intervalles est remplacée par la valeur centrale $k*\frac{256}{2^n}$. En pratique, il faut maintenant déterminer sur quel intervalle se situe une valeur c donnée, c'est-à-dire la valeur de k associée :

$$c \in [k*\frac{256}{2^n} - \frac{256}{2^{n+1}}; k*\frac{256}{2^n} + \frac{256}{2^{n+1}}] \Leftrightarrow (c*\frac{2^n}{256}) \in [k - \frac{1}{2}; k + \frac{1}{2}]$$

k est ainsi déterminé comme la valeur entière la plus proche de $c*\frac{2^n}{256}$.

On veillera, lors de l'implémentation, de ne pas calculer les puissances de 2 à chaque pixel !

4.5.2 Autres quantifications

La quantification uniforme est simple et rapide à implémenter. Cependant, elle n'est pas toujours satisfaisante et on peut vouloir lui préférer des

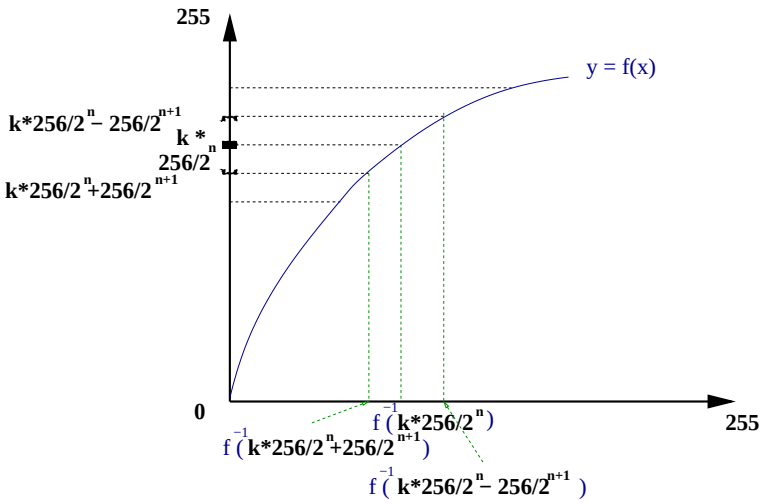


FIG. 4.20 – Quantification de type exponentielle

quantifications logarithmiques, exponentielles, racine carré, ... Or, l'équation 4.2 donnant une bonne approximation pour calculer les niveaux de décision peut se réveler coûteuse, surtout si la racine cubique de la fonction n'a pas d'intégrale analytique. Une astuce répandue consiste alors à travailler dans l'espace image de la fonction f de quantification, à y effectuer une quantification uniforme puis effectuer la transformation inverse f^{-1}.

Plusieurs contraintes sont alors imposée à f : $f(0) = 0$, $f(255) = 255$ et il est possible de déterminer $f^{-1}(x)$ pour toute valeur de x entre 0 et 255. La figure 4.20 donne un exemple sur la quantification de type exponentielle où f est définie par :

$$y = f(x) = 255 * log(x/10 + 1)/log(25.6)$$

Ainsi, toute valeur c de l'intervalle $[f^{-1}(k * 256/2^n - 256/2^{n+1}); f^{-1}(k * 256/2^n + 256/2^{n+1})]$ est remplacée par la valeur $f^{-1}(k*256/2^n)$. Sur l'exemple de l'exponentielle, les valeurs claires vont être beaucoup plus écrasées que les valeurs sombres.

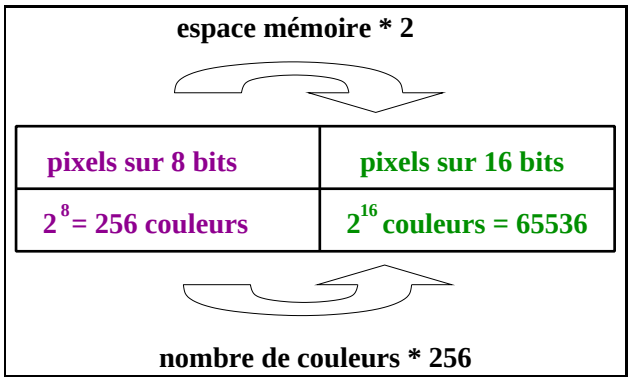
Un exemple sur l'image de la maison est donné figure 4.23.

4.5.3 Quantification des couleurs

La quantification peut s'effectuer composante par composante. On peut définir des nombres de niveaux différents pour chacune des composantes. On se reporte ensuite aux méthodes de quantification des niveaux de gris.

Nous avons déjà dit que 256 niveaux de gris constitue une qualité suffisante pour notre vision. Cependant, et même si on s'en est satisfait pendant de nombreuses années, 256 couleurs n'est pas toujours satisfaisant. C'est bien sûr suffisant pour un certain nombre de logo ou schéma mais trop réducteur pour des belles photos de paysages.

Si le nombre de bits est quelque fois limité, c'est souvent pour des questions d'espace mémoire, rarement pour des raisons technologiques (restitution des couleurs sur un écran). Cependant, en partant d'une image sur 8 bits (256 couleurs), on obtient une image sur 16 bits (65536 couleurs), soit 256 fois plus de couleurs en ne multipliant que par 2 la taille de l'espace utilisé. De même, on peut voir qu'en limitant le nombre de couleurs, on peut effectuer une compression mais celle-ci peut se réveler peu efficace malgré de lourdes pertes en qualité. Il sera ainsi préférable d'utiliser d'autres techniques de compression.



4.5.4 Quantification des couleurs sur un moniteur

Pour connaitre le nombre de bits utilisés par la carte graphique, il suffit, sous Linux, de consulter le fichier : `/etc/X11/XF86Config`

```
[...]
Section "Screen"
        [...]
```

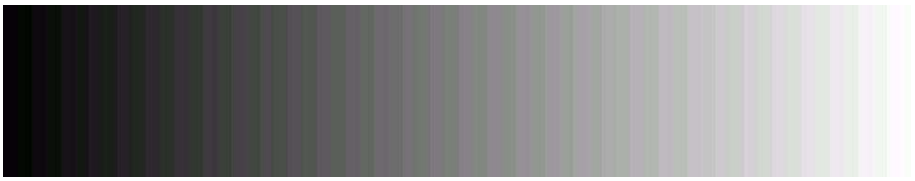


FIG. 4.21 – Dégradé de gris de 0 à 255 vu avec un display de 16 bits. Des couleurs parasites apparaissent dues au fait que les 3 composantes couleurs ne sont pas échantillonnées sur le même nombre de bits.

```
        DefaultDepth     24
[...]
```

Sous Windows, il faut aller sur le tableau de bord ou tableau de paramètres, sélectionner le moniteur et regarder le panel des propriétés.

Ceci signifie que les couleurs sont représentées sur 24 bits soit 3 fois 8 : chaque composante R, G ou B dispose de 256 valeurs différentes entre 0 et 255. Lorsque la profondeur est de 16 bits, chaque composante dispose de 5 bits et on observe qu'il reste alors 1 bit : sachant que nous sommes plus sensibles aux variations de composante verte que des autres composantes, celle-ci dispose alors de 6 bits. Pour mettre ce phénomène en évidence, composez des nuanciers de rouge, de bleu et de vert, et visualiser les en profondeur 16 bits. Vous remarquerez des bandes de couleurs de même épaisseur en rouge et bleu, et d'épaisseur moitié en vert. Visualisez ensuite un nuancier en niveaux de gris : des couleurs apparaissent (voir figure 4.21).

La profondeur peut également être de 8 bits. Dans ce cas, les bits ne sont pas répartis entre les différentes composantes car sinon, le résultat ne serait visuellement pas très correct. Une table de couleurs (*colormap*) de 256 couleurs est construite en fonction des couleurs utilisées.

4.5.5 Quelques exemples

Dans la figure 4.22, nous présentons des illustrations à différents niveaux de quantification. Dans la colonne de gauche, nous présentons un nuancier en niveaux de gris. Dans la colonne du milieu, nous présentons une image en niveaux de gris, quantifiée sur différents niveaux. Pour un nombre de bits donné n, 2^n niveaux de gris sont disponibles. Dans la colonne de droite, nous présentons une image en couleurs (r,g,b) dont chaque composante est

quantifiée sur le même nombre de bits que les deux images en niveaux de gris de la ligne.

Regardons le nuancier : on s'aperçoit que l'effet de bandes commence à disparaitre à 7 bits, bien qu'on observe encore quelques défauts. Avec 8 bits, on observe un dégradé bien lisse. Si on regarde maintenant l'image en niveaux de gris : le ciel est visuellement correct vers 6 bits tandis que les maisons l'étaient déjà à 3 bits : il serait donc préférable d'utiliser une quantification non uniforme pour cette image. Pour la version couleurs, on observe le même phénomène.

4.6 Exercices pratiques

Ajouter l'ouverture d'images de nuancier. Ces images ne seront pas ouvertes à partir de fichier mais créée en fonction de la taille courante de la fenêtre. On créera trois nuanciers : un nuancier de rouge, un de vert et un de bleu. Pour chacun, chaque ligne est identique mais les couleurs des pixels varient linéairement en fonction de la colonne du noir vert la couleur saturée (pour le nuancier rouge, on part de (0,0,0) vers (255,0,0)). Observez le résultat : observez-vous un dégradé lisse ? Pourquoi ? En modifiant la taille de votre application, voyez-vous toujours autant de bandes ? Si possible, refaites la méme expérience avec des configurations X différentes.

Que donne à l'écran un nuancier en niveaux de gris ?

Implémenter la quantification des images en niveaux de gris :

– quantification uniforme : faire varier le nombre de bits de 1 à 8

– essayer d'autres quantifications : log, racine carrée, racine cubique

puis la quantification des images couleurs RGB : même chose mais sur chaque composante prise isolément

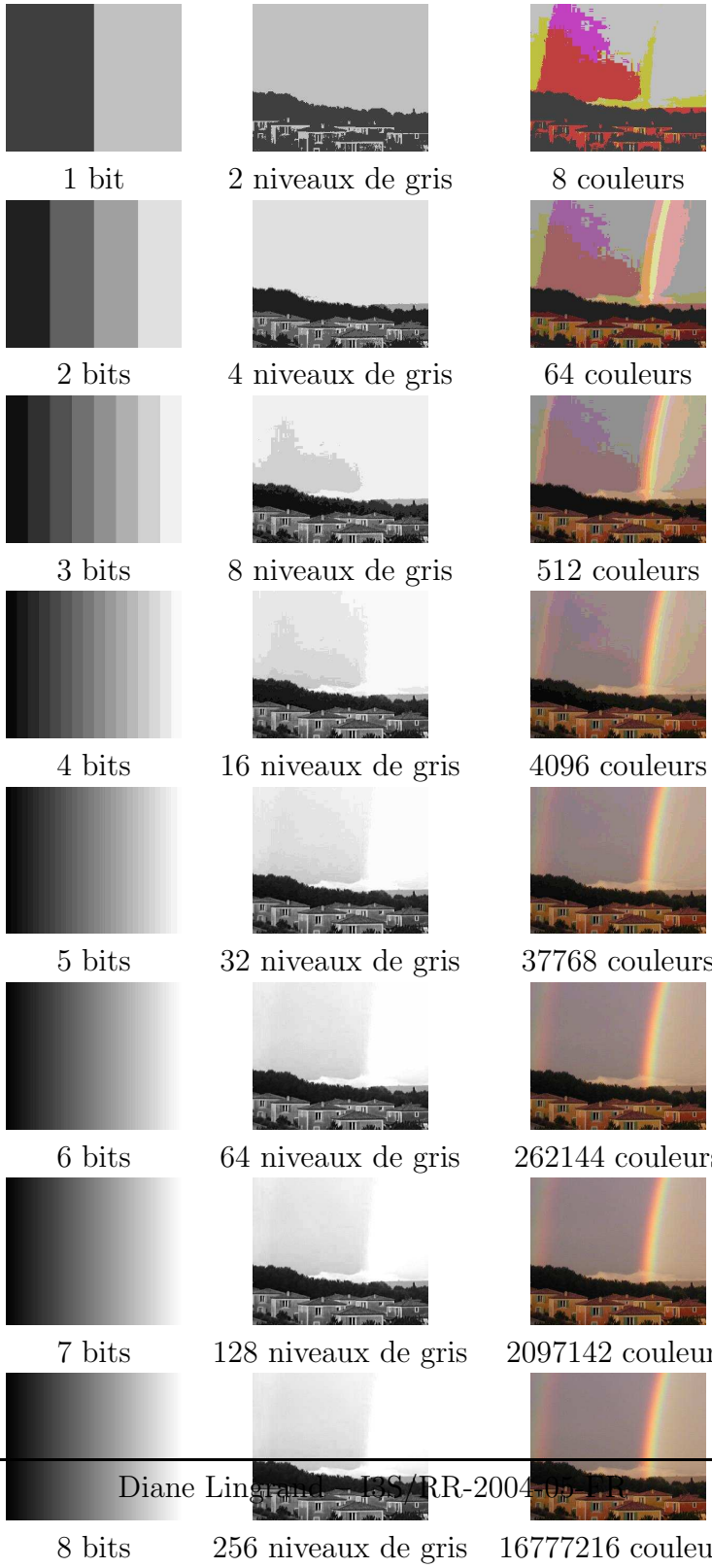


FIG. 4.22 – Exemples de quantification uniformes

4 bits 16 niveaux de gris 4096 couleurs

FIG. 4.23 – Exemples de quantification logarithmique

Chapitre 5

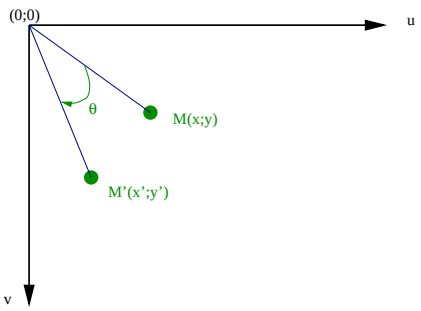
Transformations géométriques 2D

On entend par transformations géométriques 2D l'ensemble des transformations pouvant être appliquées aux pixels de l'image, sans considération d'intensité. Lorsque l'on veut afficher une image dans une fenêtre, on peut vouloir que l'image occupe tout l'espace de la fenêtre. Si les dimensions de la fenêtre et de l'image sont différentes, il faut alors effectuer un changement d'échelle de l'image. Si, par contre, on veut éclaircir une image, il ne s'agit pas de transformation géométrique.

En un mot, les transformations géométriques consistent à créer une nouvelle image en modifiant les pixels d'une précédente image. Certaines transformations sont très simple à appliquer : par exemple, une translation vers la gauche de 2 pixels. Il faudra cependant penser comment faire au bord droit de l'image. Mais comment appliquer une translation vers la gauche de 1.5 pixels ? On a pour cela besoin de retrouver le signal continu et d'accèder aux valeurs des images sur des coordonnées non entières : il s'agit de l'interpolation.

5.1 Transformations euclidiennes

Parmi les transformations géométriques, on peut citer la famille des transformations euclidiennes ou rigides (voir figure 5.1 : elles préservent les distances et les angles. Ces transformations sont les compositions de rotations et translations. Une rotation est déterminée par un centre (deux coordonnées)



et un angle. Si un point (x,y) subit une rotation d'angle θ et de centre (u_0, v_0), sa nouvelle position (x',y') vérifie :

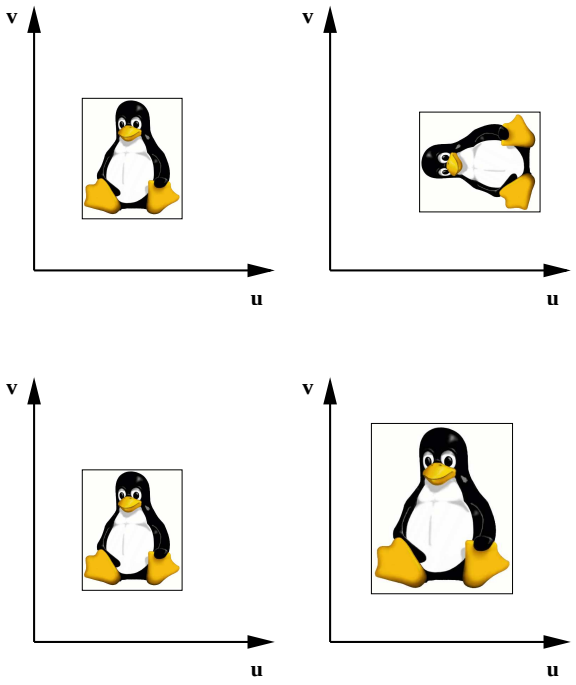
$$\begin{cases} x' = x_0 + (x - x_0)\cos(\theta) + (y - y_0)\sin(\theta) \\ y' = y_0 - (x - x_0)\sin(\theta) + (y - y_0)\cos(\theta) \end{cases}$$

On peut remarquer que, par rapport aux équations dans un repère direct, ces équations présentent un changement de signe dû au repère indirect généralement utilisé en image. On rappelle qu'il est très simple de déterminer une rotation inverse : c'est une rotation d'angle opposé.

Une translation est définie par un vecteur à deux coordonnées. Si un point (x,y) subit une translation (t_x, t_y), sa nouvelle position (x',y') vérifie :

$$\begin{cases} x' = x + t_x \\ y' = y + t_y \end{cases}$$

Le produit d'une rotation et d'une translation n'est pas commutatif (sauf cas de la rotation d'angle nul ou de la translation nulle). Tout mouvement rigide, donc toute combinaison de rotation et de translation peut s'exprimer comme la composition d'une rotation et d'une translation. Tout mouvement rigide peut également s'exprimer comme la composition d'une translation et d'une rotation. Cette décomposition, dans les deux cas, est unique si l'on fixe que la rotation est de centre l'origine. En regardant l'équation (5.1), on observe que la rotation autour d'un point M_0 est équivalente à la composition d'une translation de vecteur $O\vec{M}_0$, d'une rotation autour de l'origine, de même angle et d'une translation de vecteur $-O\vec{M}_0$.



5.2 Les homothéties

Les homothéties sont des changements d'échelle selon l'axe des x et l'axe des y, à partir d'une origine donnée (voir un exemple en figure 5.2).

Une homothétie de rapports λ_x et λ_y s'exprime par :

$$\begin{cases} x' = \lambda_x x \\ y' = \lambda_y y \end{cases}$$

Pour exprimer une homothétie de centre M_0, on se translate à l'origine du repère puis on effectue une homothétie de centre 0 puis on effectue à nouveau une translation opposée à la première.

5.2.1 Exemple d'application : la transformation Window-Viewport

C'est une application très couramment employée et qui consiste à visualiser dans une portion d'écran (Viewport) une portion du monde (Window).

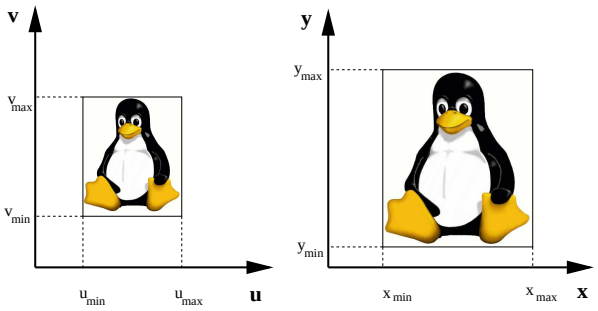


FIG. 5.1 – La transformation *Window-Viewport.*

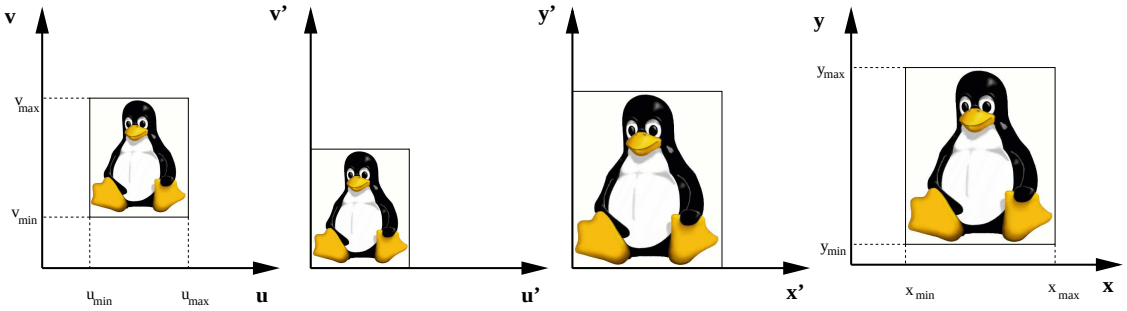


FIG. 5.2 – Décomposition en étapes élémentaires de la transformation *Window-Viewport.*

Les coordonnées dans le Viewport sont des coordonnées écran (en pixels) tandis que les coordonnées du monde peuvent être de différentes unités (mètres, kilomètres, micromètres, ...). On considère que le monde est vu au travers d'une fenêtre (Window) et affiché dans une portion d'écran (Viewport).

La figure 5.1 présente le problème : il s'agit simplement d'un changement de repère et changement d'échelle. Nous allons décomposer ce problème en trois étapes (voir figure 5.2) :

- translation à l'origine du repère du *Viewport*
- changement d'échelle
- translation au point de coordonnées (x_{min}, y_{min})

Les transformations effectuées lors de ces trois étapes s'expriment très simplement et leur combinaison est aisée. Il faut cependant mentionner que des erreurs sont fréquemment faites lorsque l'on essaie de déterminer directement

l'expression de la transformation globale.

Translation à l'origine

$$\begin{cases} u' = u - u_{min} \\ v' = v - v_{min} \end{cases}$$

Changement d'échelle

$$\begin{cases} x' = \frac{x_{max}-x_{min}}{u_{max}-u_{min}} u' \\ y' = \frac{y_{max}-y_{min}}{v_{max}-v_{min}} v' \end{cases}$$

Translation au point de coordonnées (x_{min}, y_{min})

$$\begin{cases} x = x' + x_{min} \\ y = y' + y_{min} \end{cases}$$

Transformation complète : Il suffit de composer les transformations précédentes.

$$\begin{cases} x = x_{min} + \frac{x_{max}-x_{min}}{u_{max}-u_{min}}(u - u_{min}) \\ y = y_{min} + \frac{y_{max}-y_{min}}{v_{max}-v_{min}}(v - v_{min}) \end{cases}$$

5.2.2 Exemple d'application : zoom local autour d'un point sélectionné à la souris

Dans cette application, on désire agrandir l'image sur une fenêtre donnée (dimensions DxD) autour d'un point sélectionné à la souris (x_s, y_s). Le facteur de zoom est prédeterminé et a pour valeur λ. Le point cliqué sera le point fixe de la transformation (voir figure 5.3).

Pour déterminer l'expression de la transformation, il est nécessaire d'effectuer la composition :

- d'une translation pour amener le point fixe à l'origine
- d'un zoom de facteur λ
- d'une translation pour amener l'origine au point cliqué

translation pour amener le point fixe à l'origine

$$\begin{cases} x' = x - x_s \\ y' = y - y_s \end{cases}$$

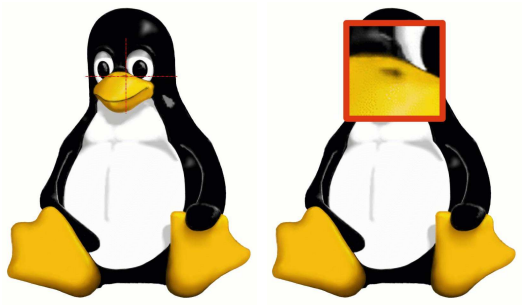


FIG. 5.3 – A gauche, l'image original. A droite, une partie de l'image, centrée sur la narine gauche de Tux, a été agrandie (facteur 4).

zoom de facteur λ

$$\begin{cases} x'' = \lambda x' \\ y'' = \lambda y' \end{cases}$$

translation pour amener l'origine au point cliqué

$$\begin{cases} x''' = x'' + x_s \\ y''' = y'' + y_s \end{cases}$$

transformation globale :

$$\begin{cases} x''' = x_s + \lambda(x - x_s) \\ y''' = y_s + \lambda(y - y_s) \end{cases}$$

En pratique, pour tous les points (x, y) tels que $x_s - D/2 \leq x \leq x_s + D/2$ et $y_s - D/2 \leq y \leq y_s + D/2$, on affiche le point de l'image en $(x_s + \frac{(x-x_s)}{\lambda}, y_s + \frac{y-y_s}{\lambda})$.

5.3 Les similitudes

Les similitudes ont pour propriétés que les distances sont multipliées par un coefficient commun à toute l'image, que les angles sont préservés ainsi que les formes (un cercle reste un cercle, un carré reste un carré, ...)

Les similitudes consistent en une généralisation des transformations euclidiennes et homothéties :

$$\begin{cases} x' = \sigma \cos\alpha x - \sigma \sin\alpha y + t_x \\ y' = \pm\sigma \sin\alpha y \pm \sigma \cos\alpha y + t_y \end{cases}$$

5.4 Le cisaillement (ou *shear*)

Le cisaillement est une transformation qui fait penser à un étirement selon un axe. Le cisaillement selon l'axe des x ne modifie pas la coordonnée en y tandis que le cisaillement selon l'axe des y ne modifie par la coordonnée en x. On peut également combiner ces deux cisaillements. Des exemples sont donnés figure 5.4.

Cisaillement selon les absisses :

$$\begin{cases} x' = x + shx * y \\ y' = y \end{cases}$$

Cisaillement selon les ordonnées :

$$\begin{cases} x' = x \\ y' = shy * x + y \end{cases}$$

Cisaillement général :

$$\begin{cases} x' = x + shx * y \\ y' = shy * x + y \end{cases}$$

5.5 Transformations affines

Les transformations affines préservent le plan affine et le parallélisme de droites. Elles sont composées de translation, rotation, changement d'échelle (*scaling*) et cisaillement (*shear*). Elles sont paramétrisées par 6 paramètres que l'on peut choisir comme les paramètres des mouvements qui les composent soit de façon plus arbitraire comme voici :

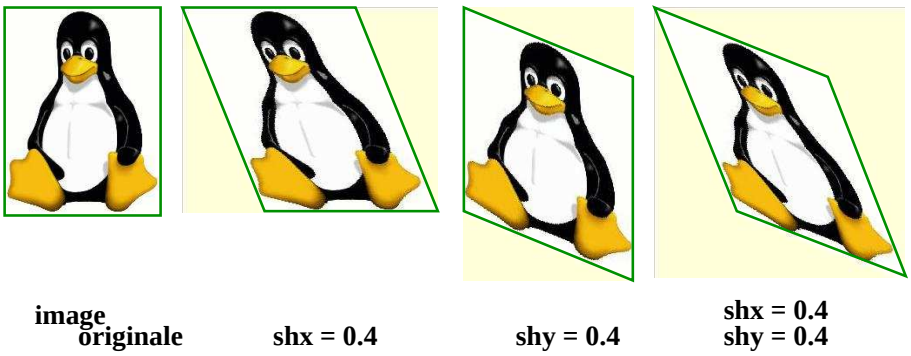


FIG. 5.4 – Transformations de cisaillement

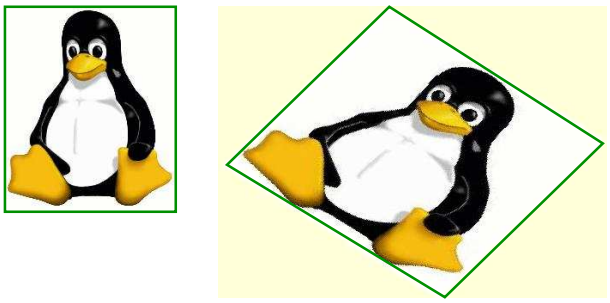


FIG. 5.5 – Exemple de transformation affine (voir équation 5.1).

$$\begin{cases} x' &= (\sigma_{xy}\sin(\alpha)+\sigma_x\cos(\alpha))x &+(\sigma_{xy}\cos(\alpha)-\sigma_x\sin(\alpha))y &+t_x \\ y' &= \sigma_y\sin(\alpha)x &+\sigma_y\cos(\alpha)y &+t_y \end{cases}$$

$$\begin{cases} x' = a_{00}x + a_{01}y + a_{02} \\ y' = a_{10}x + a_{11}y + a_{12} \end{cases}$$

Un exemple de transformation affine est donné figure 5.5. La transformation illustrée est la suivante :

$$\begin{bmatrix} 1.27 & -0.82 & 200 \\ 0.78 & 0.78 & 20 \end{bmatrix} \tag{5.1}$$

5.5.1 Détermination d'une application affine

Il existe deux façons de déterminer une application. La première consiste à déterminer les coefficients en fonction des différentes composantes (rotation, translation, ...) et est triviale. La seconde consiste à déterminer les coefficients de l'application affine en fonction du déplacement de quelques points. En effet, puisque la transformation affine dépend de 6 paramètres et que un couple $\{(x, y), (x', y')\}$ de points (antécédent et image) fournit 2 équations, 3 couples de points $\{(x_i, y_i), (x'_i, y'_i)\}$, $0 \leq i \leq 2$ suffisent à déterminer une application affine, solution du système suivant :

$$\underbrace{\begin{bmatrix} x'_0 \\ y'_0 \\ x'_1 \\ y'_1 \\ x'_2 \\ y'_2 \end{bmatrix}}_{\mathbf{Y}} = \underbrace{\begin{bmatrix} x_0 & y_0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_0 & y_0 & 1 \\ x_1 & y_1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_1 & y_1 & 1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_2 & y_2 & 1 \end{bmatrix}}_{\mathbf{A}} \underbrace{\begin{bmatrix} a_{00} \\ a_{01} \\ a_{02} \\ a_{10} \\ a_{11} \\ a_{12} \end{bmatrix}}_{\mathbf{X}}$$

Ce système admet une unique solution si le déterminant est non nul. Or, ce déterminant se ramène à :

$$\det(\mathbf{A}) = -\begin{vmatrix} x_0 & y_0 & 1 \\ x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \end{vmatrix}^2$$

La non nullité de ce déterminant est assurée par le non alignement des trois points. La solution est alors :

$$X = A^{-1}Y$$

Une application concrète est le morphing. En effet, sur une image initiale, on positionne un triangulage régulier du plan. On modifie ensuite les positions des sommets des triangles pour s'adapter à une seconde image. Chaque triangle possède trois points ayant chacun deux positions : on détermine ainsi l'application affine à appliquer à la portion d'image délimitée par le triangle.

5.6 Transformations projectives

Les transformations projectives sont paramétrisées par huit paramètres. Bien que les équations suivantes mettent en jeu 9 variables, seuls huit pa-

ramètres sont indépendants (pour s'en convaincre, on peut multiplier tous les paramètres par une constante non nulle et obtenir le même résultat).

$$\begin{cases} x' = \frac{p_{00}x+p_{01}y+p_{02}}{p_{20}x+p_{21}y+p_{22}} \\ y' = \frac{p_{10}x+p_{11}y+p_{12}}{p_{20}x+p_{21}y+p_{22}} \end{cases}$$

Puisque les transformations projectives ont 8 degrés de liberté, il suffit de 4 points pour déterminer une telle application. Imaginons que nous superposions une grille sur un portait. Si nous voulons effectuer un morphing vers un autre portrait il suffit de déterminer le déplacement des sommets de cette grille pour être capable, pour chaque élément de notre quadrillage, de déterminer la déformation à appliquer à celui-ci.

De la même façon que pour les applications affines, on peut se servir des applications projectives pour effectuer un morphing, cette fois-ci sur un maillage en carré (4 sommets définissent une transformation projective).

5.7 Autres transformations

Les transformations que nous n'avons pas encore abordées dans cette partie sont, parmi les transformations globales, les transformations non linéaires. On peut citer parmi elles les transformations espace cartésien vers espace polaire et réciproquement, la correction de distorsions (ou l'introduction de distorsions, ...).

$$\begin{cases} \rho &= \sqrt{x^2+y^2} \\ \theta &= \arctan\frac{y}{x} \end{cases}$$

Il existe également d'autres transformations qui ne sont pas globales à l'image. Par exemple le cas d'application au morphing de la section 5.6.

5.8 Interpolation

L'interpolation, c'est être capable de déterminer une méthode `getPixel(double x, double y)`.

5.8.1 Pourquoi est-il nécessaire d'interpoler ?

Reprenons le problème de la rotation. On souhaite faire tourner une image de $\theta = 10\,\text{deg}$. D'après les équations vues précédemment :

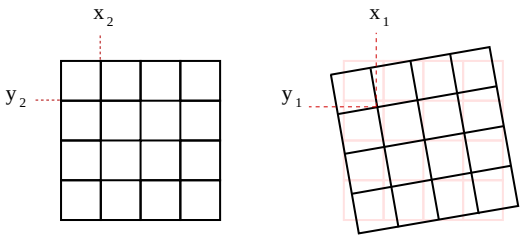


FIG. 5.6 – Après rotation, un pixel ne se retrouve plus exactement sur une position entière.

$$\begin{cases} x_2 = \cos(\theta)x_1 + \sin(\theta)y_1 \\ y_2 = -\sin(\theta)x_1 + \cos(\theta)y_1 \end{cases}$$

et $I(x_2, y_2) = I(x_1, y_1)$ d'où $I(x_2, y_2) = I(\cos(\theta)x_2 - \sin(\theta)y_2, \sin(\theta)x_2 + \cos(\theta)y_2)$

La figure 5.6 illustre le fait que, partant de pixels à coordonnées entières, on obtient, après rotation, des coordonnées non entières. Une idée intuitive est de considérer la position entière la plus proche ou bien d'effectuer une moyenne des voisins, pondéré en fonction de la distance à ces voisins. En réalité, on va utiliser la théorie de l'échantillonnage afin d'échantillonner l'image initiale pour obtenir la valeur au point de coordonnées non entières (x_1, y_1).

5.8.2 Interpolation en 1D

Ordre 0	Π	$R_0(x) = 1 \ \|x\| \leq \frac{1}{2}$
Ordre 1	$\Lambda = \Pi * \Pi$	$R_1(x) = \begin{cases} x+1 & -1 \leq x \leq 0 \\ 1-x & 0 \leq x \leq 1 \end{cases}$
Ordre 2	$\Lambda * \Pi$	$R_2(x) = \begin{cases} \frac{1}{2}(x+\frac{3}{2})^2 & -\frac{3}{2} \leq x \leq -\frac{1}{2} \\ \frac{3}{4} - x^2 & -\frac{1}{2} \leq x \leq \frac{1}{2} \\ \frac{1}{2}(x-\frac{3}{2})^2 & \frac{1}{2} \leq x \leq \frac{3}{2} \end{cases}$
Ordre 3	$\Lambda * \Lambda$	$R_3(x) = \begin{cases} \frac{2}{3} + \frac{1}{2}\|x\|^3 - x^2 & 0 \leq \|x\| \leq 1 \\ \frac{1}{6}(2-\|x\|)^3 & 1 \leq \|x\| \leq 2 \end{cases}$

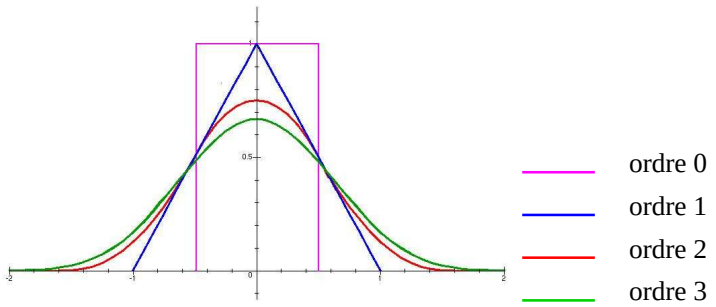


FIG. 5.7 – Les filtres de reconstruction d'ordres 0 à 4.

5.8.3 Interpolation en 2D

Le filtre de reconstruction $r(x, y)$ est séparable : $R(x, y) = R(x)R(y)$. La fonction reconstruite est :

$$I(x, y) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} I(m, n)R(x - m, y - m) \tag{5.2}$$

Cubique B-Spline :

$$k(x) = \frac{1}{6} \begin{cases} 3|x|^3 - 6x^2 + 4 & \text{si } |x| \leq 1 \\ -|x|^3 + 6x^2 - 12|x| + 8 & \text{si } 1 \leq |x| \leq 2 \\ 0 & \text{partout ailleurs} \end{cases}$$

Autre cubique B-Spline

$$k(x) = \begin{cases} (2 - C)|x|^3 + (-3 + C)x^2 + 1 & \text{si } |x| \leq 1 \\ -C|x|^3 + 5Cx^2 - 8C|x| + 4C & \text{si } 1 \leq |x| \leq 2 \\ 0 & \text{partout ailleurs} \end{cases}$$

5.8.4 interpolation au plus proche voisin

L'interpolation au plus proche voisin est l'interpolation d'ordre 0, c'est-à-dire la plus simple.

Reprenons tout de même l'équation de reconstruction 5.2. Le filtre de reconstruction d'ordre 0 est :

$$R(x, y) = 1 \text{ pour } |x| \leq \frac{1}{2} \text{ et } |y| \leq \frac{1}{2}$$

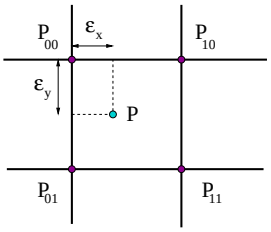


FIG. 5.8 – Interpolation d'ordre 1 : notations.

5.8.5 interpolation bilinéaire

Il s'agit d'une interpolation à l'ordre 1.

$$I(P) = (1-\epsilon_x)(1-\epsilon_y)I(P_{00}) + (1-\epsilon_x)\epsilon_y I(P_{01}) + \epsilon_x(1-\epsilon_y)I(P_{10}) + \epsilon_x\epsilon_y I(P_{11})$$

5.8.6 interpolation d'ordre 2 (*Bell*)

On applique le même raisonnement que précédemment. Supposons $0 \leq \epsilon_{x,y} \leq 0.5$ (les autres cas sont traités par symétrie).

$m = 0 : R(x) = \frac{1}{2}(x - \frac{3}{2})^2 = 0.5(\epsilon_x - 0.5)^2$
$m = 1 : R(x-1) = \frac{3}{4} - (x-1)^2 = \frac{3}{4} - \epsilon_x^2$
$m = 2 : R(x-2) = R(\epsilon_x - 1) = \frac{1}{2}(\epsilon_x + \frac{1}{2})2$
$I(P) = \sum_{m,n} R(x-m, y-n)I_{mn} = \sum_{m,n} R(x-m)R(y-n)I_{mn}$
$= R(x)R(y)I_{00} + R(x-1)R(y)I_{10} + R(x-2)R(y)I_{20}$
$+R(x)R(y-1)I_{01} + R(x)R(y-2)I_{02} + R(x-1)R(y-1)I_{11}$
$+R(x-1)R(y-2)I_{12} + R(x-2)R(y-1)I_21 + R(x-2)R(y-2)I_{22}$

5.8.7 interpolations d'ordre 3

On applique le même raisonnement que pour les ordres précédents. Avec $B = 1$ et $C = 0$, on obtient les polynômes de Mitchell suivants :

$$k(x) = \frac{1}{6}\begin{cases} 3|x|^3 - 6x^2 + 4 & \text{si } |x| \leq 1 \\ -|x|^3 + 6x^2 - 12|x| + 8 & \text{si } 1 \leq |x| \leq 2 \\ 0 & \text{partout ailleurs} \end{cases}$$

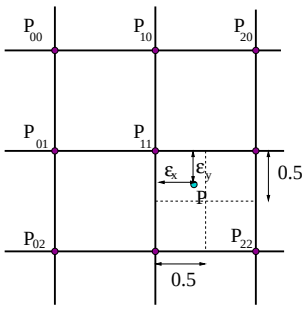


FIG. 5.9 – Interpolation d'ordre 2 : notations.

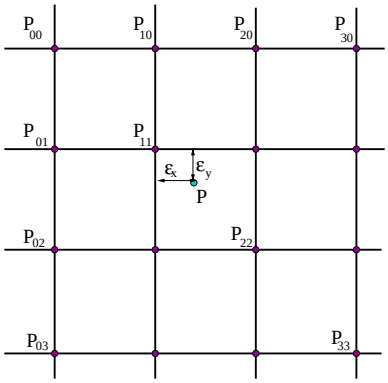


FIG. 5.10 – Interpolation d'ordre 3 : notations.

$m = 0 : R(x) = (1 - \epsilon_x)^3/6$
$m = 1 : R(x-1) = \frac{2}{3} + (\frac{\epsilon_x}{2} - 1)\epsilon_x^2$
$m = 2 : R(x-2) = \frac{2}{3} - \frac{1}{2}(1 + \epsilon_x)(1 - \epsilon_x)^2$
$m = 3 : R(x-3) = \epsilon_x^3/6$
$I(P) = I_{00}R(x)R(y) + I_{10}R(x-1)R(y) + ... + I_{33}R(x-3)R(y-3)$

Tandis qu'avec $B = 0$ et $C = 1$, on obtient :

$$k(x) = \begin{cases} (2-C)|x|^3 + (-3+C)x^2 + 1 & \text{si } |x| \leq 1 \\ -C|x|^3 + 5Cx^2 - 8C|x| + 4C & \text{si } 1 \leq |x| \leq 2 \\ 0 & \text{partout ailleurs} \end{cases}$$

$m = 0 : R(x) = (-\epsilon_x 3 + 3\epsilon_x^2 - 3\epsilon_x + 1)/6$
$m = 1 : R(x-1) = (\frac{\epsilon_x}{2} - 1)\epsilon_x^2 + \frac{2}{3}$
$m = 2 : R(x-2) = (-3\epsilon_x^3 + 3\epsilon_x^2 + 3\epsilon_x + 1)/6$
$m = 3 : R(x-3) = \frac{\epsilon_x^3}{6}$
$I(P) = I_{00}R(x)R(y) + I_{10}R(x-1)R(y) + ... + I_{33}R(x-3)R(y-3)$

5.8.8 Exemples d'interpolation

Simple rotation : Dans un premier exemple, nous allons considérer une rotation d'une image autour de son centre, d'angle 10 degrés et comparer 5 méthodes d'interpolation. Les résultats sont présentés figure 5.11.

L'interpolation au plus proche voisin donne un résultat vraissemblable mais les détails de l'image sont mal reproduit. Notamment, tous les contours fins sont pixelisés (regarder les bords de la corde orange par exemple). L'interpolation bilinéaire fournit un bien meilleur résultat. L'apport des ordres d'interpolation supérieurs n'est pas visible dans ce cas.

Multiples rotations : Nous allons maintenant appliquer 9 rotations successives de 40 degrés, toujours autour du centre de l'image (ce qui fait 360 degrés donc un tour complet. Les résultats sont présentés en figure 5.12.

On observe que l'interpolation au plus proche voisin donne une nouvelle fois le moins bon résultat. Les interpolations bilinéaires, d'ordre 2 ou d'ordre 3 avec $B = 1$ et $C = 0$ donne un résultat flou. Le meilleur résultat est celui obtenu avec l'interpolation d'ordre 3 avec $B = 1$ et $C = 0$. Ainsi, ce n'est pas seulement l'ordre 3 qui est meilleur, mais seulement certaines interpolation d'ordre 3, en choisissant judicieusement les coefficients.

image originale | ordre 0 | ordre 1

ordre 2 | ordre 3 (B=1 ;C=0) | ordre 3 (B=0 ;C=1)

FIG. 5.11 – Rotation de 10 degrés autour du centre de l'image.

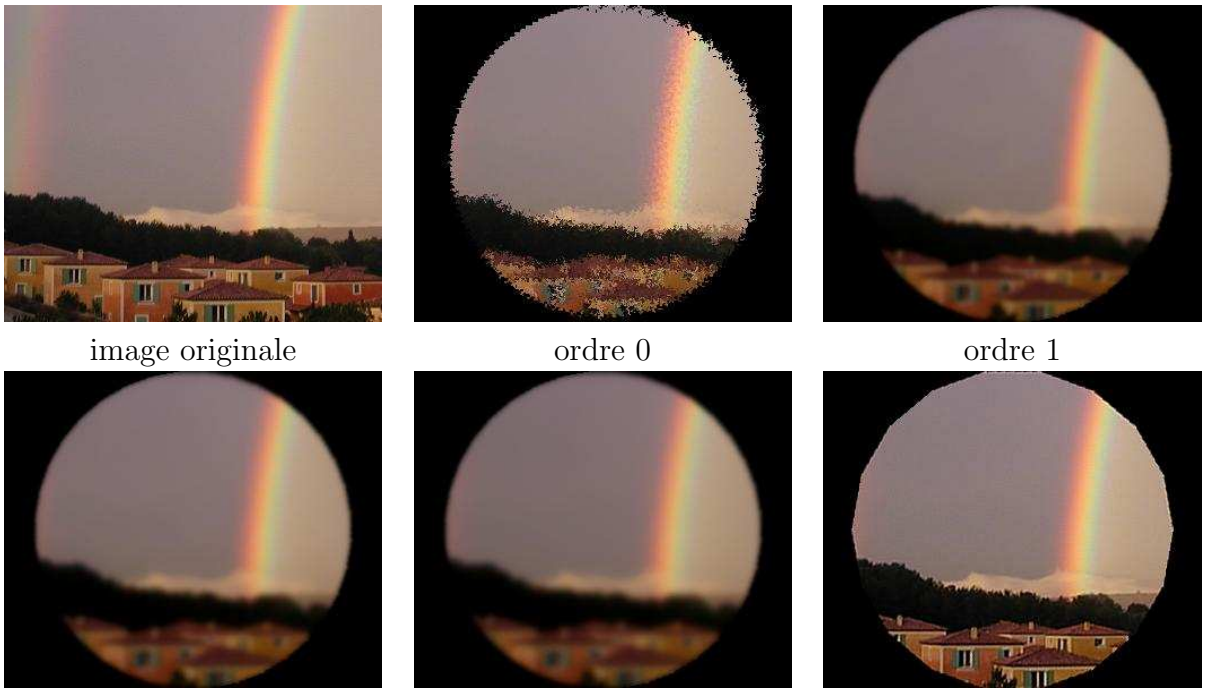


image originale | ordre 0 | ordre 1

ordre 2 | ordre 3 (B=1 ;C=0) | ordre 3 (B=0 ;C=1)

FIG. 5.12 – 9 rotation successives de 40 degrés autour du centre de l'image.

image originale | ordre 0 | ordre 1

ordre 2 | ordre 3 (B=1 ;C=0) | ordre 3 (B=0 ;C=1)

FIG. 5.13 – Zoom de facteur 1.4

Zoom : On fait ici l'expérience d'un zoom de facteur 1.4. Les résultats sont présentés sur la figure 5.13.

Les conclusions sont identiques au paragraphe précédent : c'est l'interpolation d'ordre 3 avec $B = 0$ et $C = 1$ qui fournit les meilleures résultats.

5.9 Interpolation et dégradés de couleur

On souhaite s'intéresser ici à la création de dégradés de couleur (ou "gradient de couleurs") à partir de 3 couleurs. On va partir de 2 images très simples, constituées d'une ligne comportant dans un cas, 3 colonnes et dans l'autres, 5 colonnes. Trois couleurs primitives sont utilisées : le rouge, le vert et le bleu. On effectue un zoom de facteur 200 selon différents types d'interpolation (voir figure 5.14).

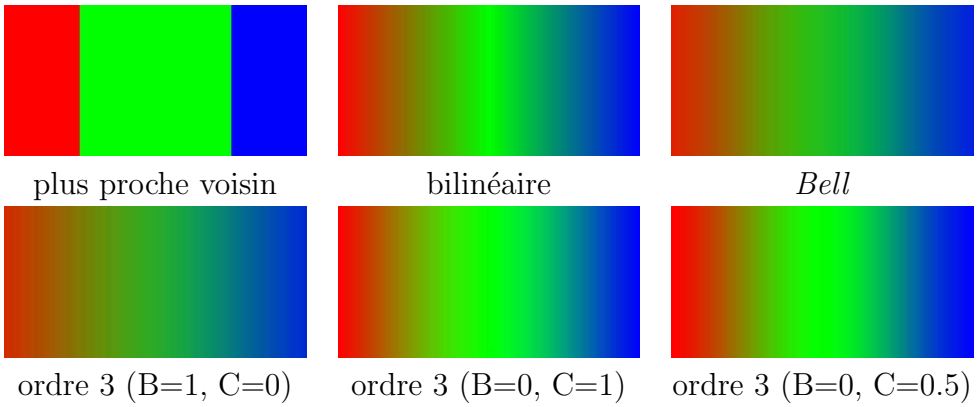


FIG. 5.14 – Interpolation entre les couleurs Rouge, Vert et Bleu en utilisant l'interpolation sur les composantes RGB, séparément.

5.10 Exercices pratiques

Implémenter l'interpolation :
- plus proche voisin
- bilinéaire
- ordre 3 (B=0,C=0.5)

Appliquer à la rotation et au zoom. Expérimentez.

Si ce n'est pas déjà fait, ajoutez les opérations suivantes :
- vidéo inverse
- seuillage (min, max)
- éclaircissement, assombrissement
- somme et différence de 2 images

S'il vous reste du temps : implémentez l'ordre 3 en permettant de spécifier B et C et testez plusieurs valeurs (B=0 et C=1, B=1 et C=0, ...)

Chapitre 6

Recherche de primitives : Détection de contours

La recherche de primitives dans une image nous intéresse car elle constitue l'étape nécessaire pour détecter un objet dans une scène, par exemple un robot footballeur qui cherche le ballon sur le terrain. La recherche d'objets et la décomposition d'une scène en plusieurs objets est un pas vers la représentation vectorielle d'une scène pouvant entre autres servir à la compression (voir les normes MPEG4 et MPEG7). La détection de primitives peut également servir à effectuer des mesures : des mesures de conformité de pièces en usine, des mesures de volumes d'organe en imagerie médicale, ...

Les primitives sont de plusieurs types. Elles peuvent consister en des contours, des régions, des points d'intérêt (ou coins), des lignes, des courbes. Selon le type de primitives que l'on recherche, différentes méthodes peuvent être employées. Le lecteur sera toutefois conscient que la détection automatique de tout objet dans n'importe quel type d'image est un problème qui est loin d'être résolu. Par contre, des applications ciblées (un certain type d'objet dans des images possédant certaines propriétés) sont tout à fait réalisables.

6.1 Détection de contours

Nous nous intéressons ici à la détection de contours c'est-à-dire à la détection des lieux de sauts d'intensité. Nous allons commencer par un petit exemple naïf afin de mieux saisir le problème.

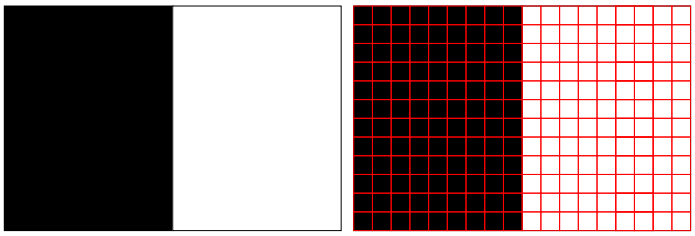


FIG. 6.1 – Une image naïve pour comprendre la détection de contour. A droite, un grossissement des pixels.

6.1.1 Exemple naïf

Nous allons commencer par une image très simple (figure 6.1). La frontière que nous recherchons est très simple, il s'agit de la frontière entre le noir et le blanc. Regardons les pixels de plus près. Nous remarquons qu'à gauche de l'image, tous les pixels sont noirs et qu'à droite de l'image, tous les pixels sont blancs. Jusque là, rien de sorcier. Mais quels sont sont les pixels du contours ? Et bien, nous pouvons considérer que ce sont les pixels noirs dont le voisin de droite est blanc ainsi que les pixels blancs dont le voisin de gauche est noir. On voit alors que notre contour sera épais de 2 pixels. D'autre part, il n'est pas aisé de construire un algorithme qui s'intéresse tantôt au voisin de gauche, tantôt au voisin de droite. On va faire ici un choix qui est le voisin de gauche.

Construisons maintenant une nouvelle image, de même dimension mais donc chaque pixel a pour valeur la différence entre la valeur du pixel (i,j) et celle du pixel (i-1,j). Par abus de langage, il est souvent dit que l'on effectue la différence des pixels. Nous abuserons de cela par la suite. En supposant qu'on parte d'une image en niveaux de gris de 0 à 255, cette image peut obtenir des valeurs de -255 à +255. Il faut soit renormaliser les valeurs entre 0 et 255, soit prendre la valeur absolue. C'est cette dernière solution que nous utilisons ici car nous nous moquons de l'orientation du contour. Dès que l'on va chercher à implémenter cet algorithme, va se poser le problème des bords de l'image. Plusieurs hypothèses sont classiquement faites :

- les pixels situés au bord ne sont pas traités
- l'extérieur de l'image est composé de pixels de couleur unie (par exemple le noir)
- les bords sont dupliqués à l'extérieur de l'image

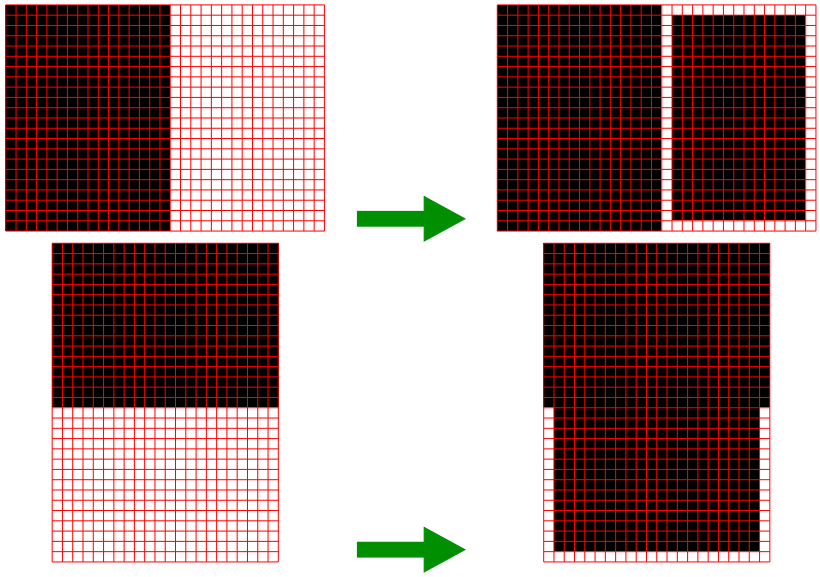


FIG. 6.2 – Dans la seconde colonne, on visualise le résultat de la détection de contours des images de la première colonne. On observe que notre détecteur convient bien pour les contours verticaux mais pas pour les contours horizontaux.

– l'image est répétée en cyles (I[i+width][j+height] = I[i][j])

On retrouve deux de ces hypothèses en Java avec les constantes

– `java.awt.image.ConvolveOp.EDGE_NO_OP` et

– `java.awt.image.ConvolveOp.EDGE_ZERO_FILL`.

En ce qui concerne les contours, on se moque généralement des contours qui pourraient se trouver au bord d'un image. Ainsi, on peut se contenter de ne pas traiter les pixels aux bords ce qui modifie nos habituelles boucles :

```
for(int i = 1; i < width - 1 ; i++) {
        for(int j = 1; j < height - 1 ; j++) {
                ....
```

Nous visualisons sur la figure 6.2 le résultat de cet algorithme sur notre exemple simple ainsi que sur un autre exemple.

On remarque que notre algorithme fonctionne correctement pour détecter la frontière verticale de la première image mais pas pour la frontière horizontale de la deuxième image. On va améliorer notre détecteur de contour naïf en composant deux détecteurs :

– un détecteur de contours horizontaux :

$$I_{cv}[i][j] = I[i][j] - I[i-1][j]$$

– et un détecteur de contours verticaux :

$$I_{ch}[i][j] = I[i][j] - I[i][j-1]$$

Notre détecteur va utiliser la norme du vecteur $[I_{cv}\ I_{ch}]$. Parmi les normes existantes, on peut utiliser la norme euclidienne ou la norme tour d'échiquier. Si on choisit la norme euclidienne, il faudra prendre garde à une erreur rencontrée fréquemment et qui consiste à écrire :

```
        I[i][j] = Math.sqrt( Math.pow(I_{cv}[i][j],2)
                           + Math.pow(I_{ch}[i][j],2));
au lieu de
        I[i][j] = Math.sqrt( I_{cv}[i][j] * I_{cv}[i][j]
                           + I_{ch}[i][j] * I_{ch}[i][j]);
```

En effet, la méthode `Math.pow(double x, double y)` permet d'élever n'importe quel x à la puissance n'importe quel y. Ainsi, le calcul n'est pas fait de façon itérative comme pour les puissances entières mais fait référence aux exponentielles et donc aux séries entières. Or, si les mathématiciens savent bien manipuler les sommes infinies, l'informaticien doit tronquer ces sommes. Le résultat sera donc approché et obtenu en plus de temps qu'une simple multiplication. Il ne faut donc pas utiliser cette méthode pour une élévation au carré.

Essayons maintenant notre détecteur de contours sur une nouvelle image plus compliquée, mais toujours sans bruit (voir figure 6.3). L'image obtenue contient des valeurs supérieures à 255 ($255\sqrt{2}$). Il a donc été nécessaire de renormaliser l'image entre 0 et 255 faisant apparaitre les contours du carré noir en gris (de niveau 180). Les pixels ont volontairement été grossis afin de visualiser la ligne diagonale noire. Etant d'épaisseur 1 pixel, les deux contours de cette ligne sont collés montrant une ligne de contour d'épaisseur 1. On voit ici apparaître les difficultés de détection de fins détails par des détecteurs de contours.

Si on veut pouvoir étiqueter les pixels appartenant aux contours ou non, il faut positionner un seuil de détection. Ici, la valeur 128 conviendrait bien. Il s'agit cependant d'une image synthétique à deux couleurs et très simple. Ce seuil n'a apparemment aucune raison d'être valable sur d'autres images.

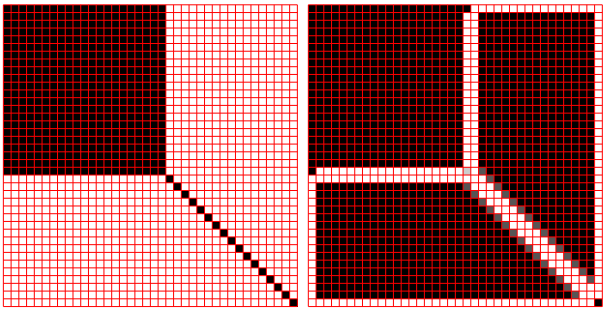


FIG. 6.3 – Une image synthétique et l'image de ses contours. Les pixels sont volontairement grossis.

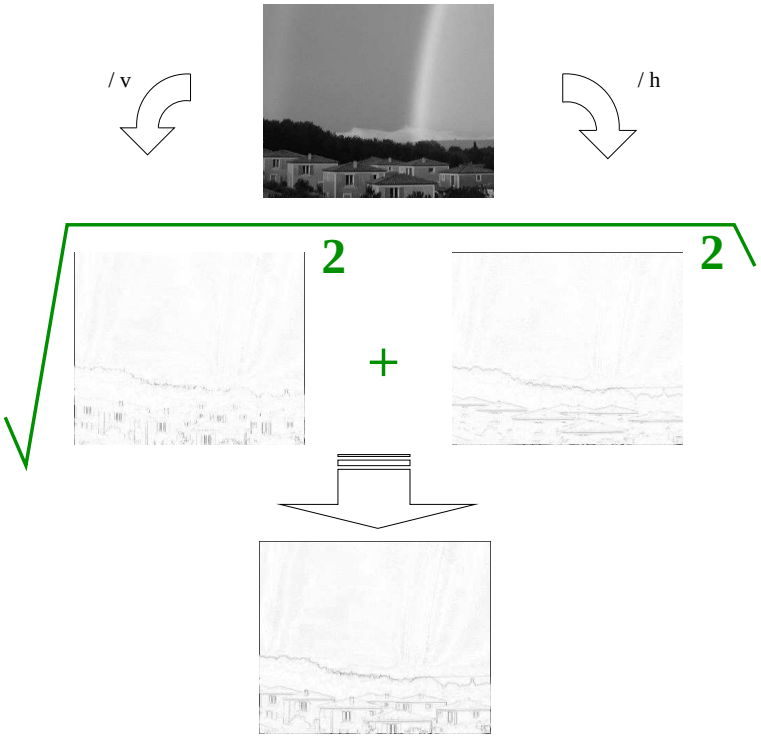


FIG. 6.4 – Notre détecteur naïf amélioré sur une image réelle.

Essayons maintenant notre détecteur sur une photographie, image réelle (figure 6.4).

Notre détecteur possède plusieurs avantages : il est très simple à comprendre et à implémenter et il est rapide d'exécution. Par contre, il est sensible au bruit, détecte trop de contours et détecte mieux des contours diagonaux que horizontaux ou verticaux.

6.1.2 Approche par convolution

Nous allons améliorer notre détecteur naïf. Pour faciliter l'expression des différents détecteurs que nous allons maintenant étudier, nous allons introduire la convolution.

convolution

Rappelons tout d'abord la convolution d'une fonction continue f par une fonction continue g :

$$f * g(x,y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)g(u-x,v-y)dudv$$

Pour des fonctions discrètes :

$$f * g(x,y) = \sum_{u=-\infty}^{\infty}\sum_{v=-\infty}^{\infty} f(x,y)g(u-x,v-y)$$

Si on applique ceci à une image I_1 de dimensions finies et à un noyau de convolution K de dimensions 3x3x (K pour *kernel*), les pixels de l'image I_2 obtenue par convolution de I_1 par K :

$$I_2(i,j) = \sum_{k=0}^{2}\sum_{l=0}^{2} I_1(i+k-1, j+l-1)K(k,l)$$

Une illustration de ceci est donnée figure 6.5.

Si nous revenons à notre détecteur naïf, il peut être décomposé en une convolution par un noyau K_x et un noyau K_y tel que :

$$K_x = \begin{bmatrix} 0 & 0 & 0 \\ -1 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad K_y = \begin{bmatrix} 0 & -1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

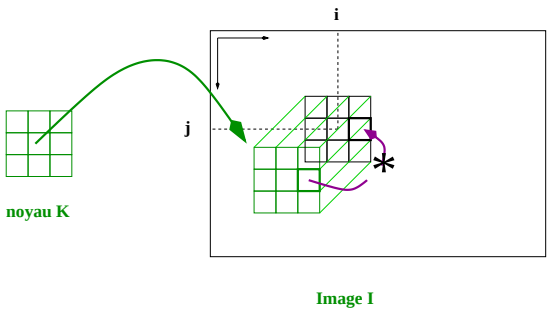


FIG. 6.5 – Effectuer la convolution d'une image I par un masque de convolution K revient à remplacer chaque pixel (i,j) par la somme des produits des éléments du masque avec les pixels correspondant, en centrant le masque sur le pixel (i,j).

Afin de bien saisir la manipulation du masque de convolution, la figure 6.6 complète la figure 6.5 dans le cas du noyau K_x.

Lorsque l'on effectue cette convolution, on ne conserve que la valeur absolue du résultat. Dans JavaD, il existe une classe d'opérateurs de convolution `java.awt.image.ConvolveOp` que l'on peut utiliser comme ceci :

```
BufferedImage biInput, biOutput;
[...]
float [] filtre = {-0.0f, 0.0f, 0.0f,
                   -1.0f, 1.0f, 0.0f,
                   -0.0f, 0.0f, 0.0f };
Kernel kernel = new Kernel(3, 3, filtre);
ConvolveOp cop = new ConvolveOp( kernel, ConvolveOp.EDGE_NO_OP, null);
cop.filter(biInput,biOutput);
```

Si les pixels de l'image en entrée (`biInput`) ont des composantes entières de 0 à 255, il en est de même pour les pixels en sortie(`biOutput`). Ainsi, tout pixel résultat de la convolution et devant être négatif sera ramené à zéro : on ne peut ainsi détecter que des contours foncés vers clairs mais pas inversément. Plusieurs solutions sont envisageables :

- effectuer la convolution avec un filtre et le filtre opposé et faire la somme des deux résultats
- utiliser un autre format de `BufferedImage` (compliquera surement le code ...)

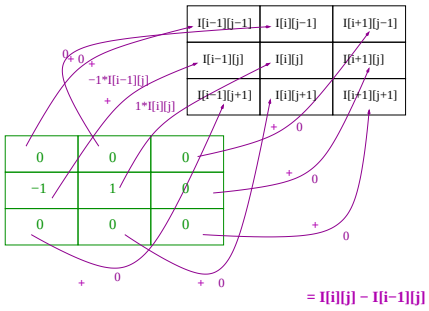


FIG. 6.6 – Convolution par le noyau K_x. Exemple du pixel (i,j).

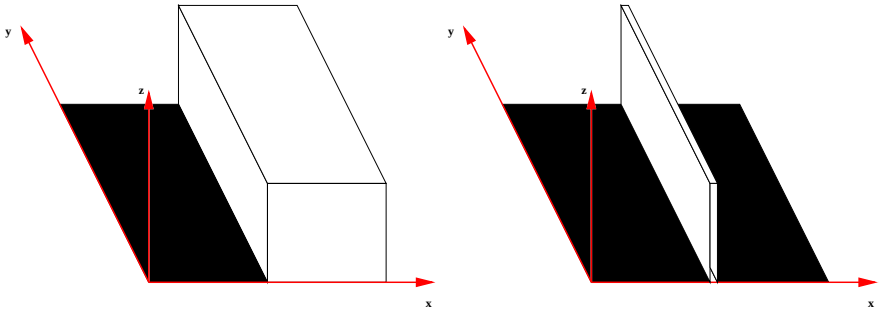


FIG. 6.7 – Vue 3D de l'image 6.1 et de sa dérivée

– recoder une méthode de convolution conservant directement la valeur absolue du résultat (c'est surement la solution la plus efficace).

Dérivée première

Redessinons la première image simple (figure 6.1 que nous avons utilisée dans ce chapitre en ajoutant une dimension spatiale qui correspond à l'intensité (voir figure 6.7). Le lieu de contour correspond au saut de hauteur. Si nous dessinons la dérivée, nous observons que le lieu de contour correspond aux pics de la dérivée. Nous allons donc nous intéresser maintenant à la dérivée de l'image.

Rappelons la définition de la dérivée selon x d'une fonction f en x :

$$f(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Dans le cas d'une image discrète, h peut valoir au minimum 1 (0 n'est pas admissible, on doit pouvoir diviser par h). On va donc approximer la dérivée selon x par :

$$\frac{\partial f}{\partial x} = I[x+1] - I[x]$$

Cela est loin de satisfaire les mathématiciens mais peut nous contenter.

Détecteur de Roberts (1965) Le détecteur de Roberts est assez similaire à notre détecteur naïf mais cherche les dérivées selon des directions diagonales. Il est décomposé en deux masques de convolution :

$$\frac{dI}{dx} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \quad \frac{dI}{dy} = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$$

Le module ou force du contour est calculé par la norme du vecteur composé par les deux composantes de dérivée :

$$\sqrt{(\frac{dI}{dx})^2 + (\frac{dI}{dy})^2}$$

ou

$$\max(\frac{dI}{dx}, \frac{dI}{dy})$$

La normale au contour est déterminée par :

$$\arctan((\frac{dI}{dy})/(\frac{dI}{dx}))$$

Ce détecteur de contour comporte des défauts similaires à notre détecteur précédent.

Pourquoi les images dérivées sont-elles très bruitées ? Supposons, de façon assez simpliste, que l'image que l'on observe I soit obtenue par l'ajout à une image non bruitée I_{pure} d'un bruit sous la forme :

$$I = I_{pure} + \epsilon \sin(\omega x)$$

Lorsque l'on dérive, on obtient :

$$I' = I'_{pure} + \epsilon\omega \cos(\omega x)$$

FIG. 6.8 – Filtre de Roberts sur une image réelle.

Ainsi, le bruit d'amplitude ϵ de l'image est amplifié sur la dérivée par un facteur $\omega = 2\pi\nu$ où ν est la fréquence. Des bruits de hautes fréquences vont donc fortement perturber la dérivée. Il convient alors d'effectuer un filtre passe-bas ou un lissage afin d'éliminer ces hautes fréquences.

Lors d'un lissage, la précision de la localisation va être diminuée mais leur détection sera améliorée. Parmi les lissages on pourra utiliser un filtre moyenneur de noyau :

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} /9$$

ou un filtre gaussien discrétisé et tronqué :

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 4 & 1 \\ 0 & 1 & 0 \end{bmatrix} /8 \quad \begin{bmatrix} 1 & 1 & 1 \\ 1 & 8 & 1 \\ 1 & 1 & 1 \end{bmatrix} /16$$

Détecteur de Sobel (1970) Le filtre de Sobel combine à la fois le lissage [121] et la dérivée [10 − 1]. On peut voir le résultat comme la moyenne des

dérivées des moyennes :

$$I_2(x,y-1) = (I(x-1,y-1) + 2I(x,y-1) + I(x+1,y-1))/4$$
$$I_2(x,y) = (I(x-1,y) + 2I(x,y) + I(x+1,y))/4$$
$$I_2(x,y+1) = (I(x-1,y+1) + 2I(x,y+1) + I(x+1,y+1))/4$$

$$\frac{dI}{dy}(x,y+1) = I(x,y+1) - I(x,y)$$
$$\frac{dI}{dy}(x,y) = I(x,y) - I(x,y-1)$$
$$\frac{dI}{dy}(x,y+1) + \frac{dI}{dy}(x,y) = I(x,y+1) - I(x,y)$$

$$\frac{dI_2}{dy}(x,y) = (I_2(x,y+1) - I_2(x,y-1))$$
$$= [(I(x-1,y+1) + 2I(x,y+1) + I(x+1,y+1))$$
$$-(I(x-1,y-1) + 2I(x,y-1) + I(x+1,y-1))]/8$$

Ce qui donne les masques de convolution suivant :

$$\frac{dI}{dx} = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} /4 \quad \frac{dI}{dy} = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} /4$$

On peut appliquer le détecteur de Sobel sur des images couleurs en appliquant le même algorithme sur les différentes composantes RGB prises séparément. On peut également appliquer le détecteur de Sobel uniquement sur la composante de luminance.

D'autres filtres basés sur les dérivées première existent : ils s'intéressent généralement à des directions supplémentaires. C'est le cas des filtres de Prewitt et de Kirsch.

Détecteur de Prewitt (1970) Le module du contour est obtenu par la valeur absolue maximale des convolutions avec les différents masques de convolution :

$$\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} /3, \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & -1 & -1 \end{bmatrix} /3, \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} /3, \begin{bmatrix} 0 & 1 & 1 \\ -1 & 0 & 1 \\ -1 & -1 & 0 \end{bmatrix} /3$$

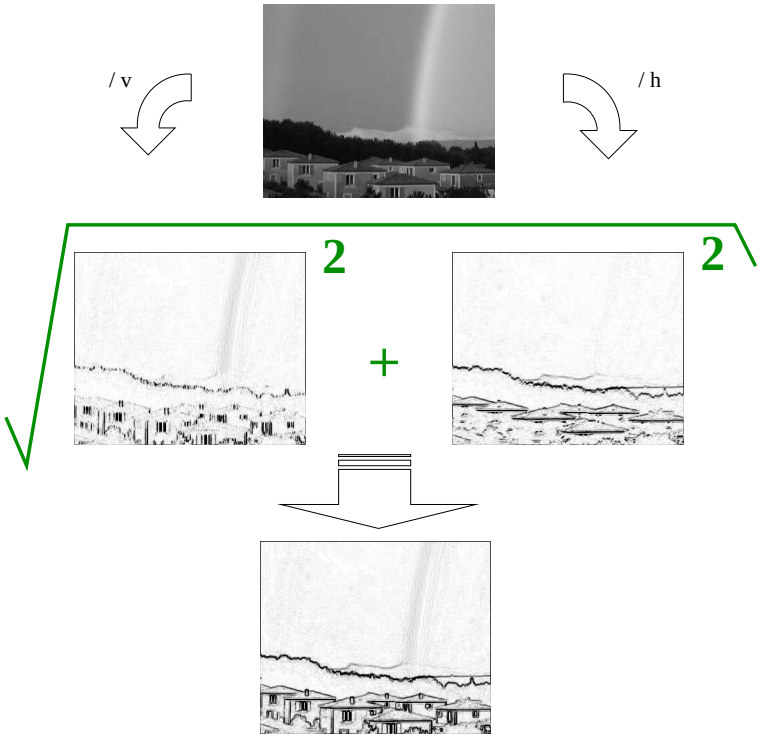


FIG. 6.9 – Filtre de Sobel sur une image réelle.

image `solEssi3212` image `michoko2476` image `orangerie2989`

FIG. 6.10 – Filtre de Sobel appliqué à trois images réelles. On remarque que certains contours sont bien détectés tandis que d'autres posent problèmes. Notamment, beaucoup de contours parasites apparaissent. Les portions d'images contenant trop de détails sont mal segmentées. On remarque également que certains contours sont des frontières entre objets, d'autres dues à de la texture et enfin, comme par exemple l'image de gauche, à des ombres.

FIG. 6.11 – A gauche : l'image originale (`orangerie2989`). A milieu : filtre de Prewitt. A droite : filtre de Kirsch. Les deux images de contours sont montrées en vidéo inverse.

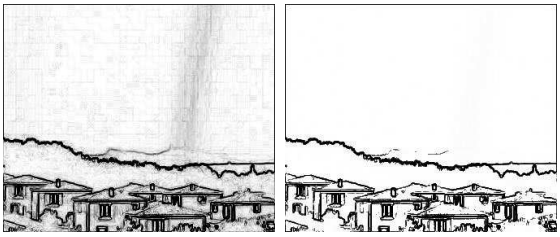


FIG. 6.12 – A gauche : filtre de Sobel et vidéo inverse. A droite, même chose après seuillage simple.

Détecteur de Kirsch (1971) Ce détecteur est un peu plus complexe que le précédent : il utilise 8 noyaux de convolution :

$$\begin{bmatrix}5 & -3 & -3\\ 5 & 0 & -3\\ 5 & -3 & -3\end{bmatrix}/15,\ \begin{bmatrix}5 & 5 & -3\\ 5 & 0 & -3\\ -3 & -3 & -3\end{bmatrix}/15,\ \begin{bmatrix}5 & 5 & 5\\ -3 & 0 & -3\\ -3 & -3 & -3\end{bmatrix}/15,\ \begin{bmatrix}-3 & 5 & 5\\ -3 & 0 & 5\\ -3 & -3 & -3\end{bmatrix}/15,$$

$$\begin{bmatrix}-3 & -3 & 5\\ -3 & 0 & 5\\ -3 & -3 & 5\end{bmatrix}/15,\ \begin{bmatrix}-3 & -3 & -3\\ -3 & 0 & 5\\ -3 & 5 & 5\end{bmatrix}/15,\ \begin{bmatrix}-3 & -3 & -3\\ -3 & 0 & -3\\ 5 & 5 & 5\end{bmatrix}/15,\ \begin{bmatrix}-3 & -3 & -3\\ 5 & 0 & -3\\ 5 & 5 & -3\end{bmatrix}/15$$

Seuillage Le but de la détection de contours est de pouvoir, justement, déterminer des contours d'objets, de régions, ... dans les images. Les filtres précédents nous fournissent une mesure de sauts ou différences d'intensité dans l'image. Le seuillage permet d'éliminer des points en lequel la différence d'intensité est trop faible pour que le point soit un point de contour. On peut pousser le seuillage afin d'obtenir une image binaire : tout point de l'image est un point de contour ou n'en est pas un. On peut cependant vouloir conserver une information moins binaire tout en éliminant certaines valeurs d'intensité.

Le seuillage le plus simple consiste à éliminer les points dont le module de contour est en dessous d'un seuil minimum. Il ne tient pas compte de la topologie et peut conserver un point bruité isolé mais éliminer une portion de contour d'intensité faible mais réelle.

Une autre forme de seuillage, le seuillage par hystérésis fonctionne à l'aide de deux seuils : un seuil bas et un seuil haut. Tout point dont le module de contour est au-dessus du seuil haut est un point de contour. Tout point dont le module de contour est en dessous du seuil bas n'est pas un point de contour.

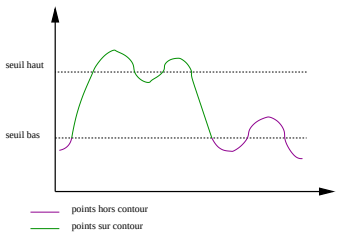


FIG. 6.13 – Seuillage hystérésis.

Les points de module entre le seuil bas et le seuil haut sont des points de contour s'ils possèdent un point de contour dans leur voisinage. La figure 6.13 illustre le fonctionnement de ce seuillage.

Dérivée seconde

Nous venons de voir des filtres basés sur la recherche de maxima de la dérivée première. Les filtres que nous allons maintenant étudier sont basés sur les zéros de la dérivée seconde, ou, plus précisément, du laplacien :

$$\nabla^2 I = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}$$

Les considérations concerant le bruit dans la dérivée première sont encore plus importante dans les calculs de dérivée seconde (elles sont cette fois-ci, non pas en E mais en ϵ^2). On utilise donc couramment une combinaison de lissage et laplacien, ce qui correspond au laplacien d'une gaussienne.

Laplace Voici deux noyaux de convolution correspond au laplacien d'une gaussienne en connexité 4 puis en connexité 8. La figure 6.14 présente les résultats, assez bruités.

$$\begin{pmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{pmatrix} /4 \quad \begin{pmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{pmatrix} /8$$

Laplacien DOG (années 80) de Marr et Hildreth Ce détecteur repose sur l'idée que le Laplacien peut être vu comme la différence entre deux lissages

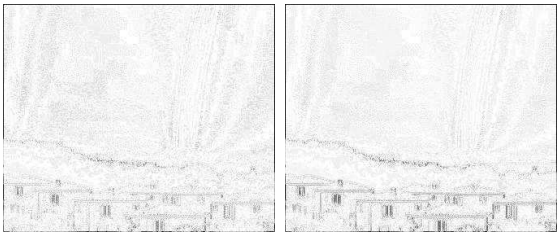


FIG. 6.14 – Filtre de Laplace appliqué à l'image `maisonGrey.png`. A gauche, connexité 4 ; à droite, connexité 8. Les deux images ont subi une correction gamma de 0.3 afin d'être lisibles.

FIG. 6.15 – Filtre Laplacien DOG : différence entre lissage gaussien 3x3 et lissage gaussien 7x7.

gaussiens de tailles différentes. L'image de contour est obtenue comme la différence entre une image peu lissée et une image fortement lissée. La figure 6.15 présente le résultat de la différence entre l'image `maisonGrey.png` lissée par un noyau gaussien 3x3 et l'image lissée par un noyau gaussien 7x7, après vidéo inverse et correction gamma de 0.3.

Huertas-Médioni (1986) Réécrivons l'expression du Laplacien d'une gaussienne :

$$\Delta g(x,y) = \frac{1}{G_0}(2 - \frac{x^2+y^2}{2\sigma^2})e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Ce filtre est séparable et peut s'exprimer sous la forme :

$$g_1(x).g_2(y) + g_1(y).g_2(x)$$

FIG. 6.16 – Filtre de Huertas-Medioni appliqué à l'image `maisonGrey.png`.

avec :

$$g_1(x) = \frac{1}{\sqrt{2G_0}}(1 - \frac{x^2}{2\sigma^2})e^{-\frac{x^2}{2\sigma^2}}$$
$$g_2(x) = \frac{1}{\sqrt{2G_0}}e^{-\frac{x^2}{2\sigma^2}}$$

On discrétise ensuite les filtres en prenant par exemple $\frac{1}{G_0} = 4232$ et $\sigma^2 = 2$:

g_1 = [-1 -6 -17 -17 18 46 18 -17 -17 -6 -1]

g_2 = [0 1 5 17 36 46 36 17 5 1 0]

Un exemple est proposé figure 6.16.

6.2 Travaux pratiques

- Essayer les filtres : Roberts, Sobel, Prewitt, Kirsch, Laplace
- Essayer différents lissages gaussiens de différentes tailles de masques
- Combiner filtres, lissages et seuillage : noter les meilleures résultats

Observation Observer le résultat du détecteur de Sobel sur une image couleur en utilisant les composantes prises séparement puis ajoutées. Regarder le résultat sur chaque image de composante (l'image rouge, l'image verte et l'image bleue). Regarder également sur les images de composantes Y, U et V. Comparer.

Chapitre 7

Recherche de primitives : Opérateurs morphomathématiques

Les opérateurs morphomathématiques se sont initialement appliqués sur des images en noir et blanc (Matheron et Serra, 1965). Ils ont ensuite été étendus à des images en niveaux de gris par Dougherty en 1978. Pour les appliquer à des images couleurs, il suffit alors de les appliquer séparément à chaque composante couleur.

7.1 Notions de base

Avant de s'attaquer au vif du sujet, nous allons commencer par revoir des notions de base sur les ensembles. On supposera cependant connues les notions suivantes :

- notion d'appartenance d'un élément à un ensemble
- notions d'intersection et d'union de deux ensembles
- notion de complémentaire d'un ensemble

On rappelle la notion de complémentaire d'un ensemble B par rapport à un ensemble A : c'est l'ensemble des éléments de A n'appartenant pas à B :

$$A/B = A \cap B^c = A/(A \cap B)$$

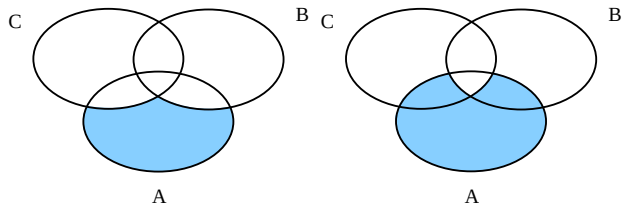


FIG. 7.1 – A gauche, loi $A/(B \cup C) = (A/B) \cap (A/C)$. A droite, loi $A/(C \cap C) = (A/B) \cup (A/C)$

On rappelle également les lois de Morgan (voir figure 7.1) :

$$A/(B \cup C) = (A/B) \cap (A/C)$$
$$A/(C \cap C) = (A/B) \cup (A/C)$$

7.1.1 Réflexion

$$-A = \{-a : a \in A\}$$

7.1.2 Addition de Minkowski

$$B = A + x = \{a + x : a \in A\}$$

$$C = A \oplus B = \bigcup (A + b : b \in B) = \bigcup (B + a : a \in A) = \{x : (-B + x) \cap A \neq \emptyset\}$$

L'addition de Minkowski est commutative. Elle est également appelée "dilatation".

7.1.3 Eléments structurants

Les éléments structurants permettent de définir le type de voisinage que l'on souhaite considérer. Nous donnons ici quelques exemples de voisinages 3x3. Le voisinage en connexité 4 est donné par k_c, celui en connexité 8 par k_s.

D'autres voisinages, moins utilisés, concernent les voisins horizontaux, k_h, et les voisins verticaux, k_v.

$$k_c = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix} \quad k_h = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

$$k_s = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad k_v = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

7.2 Dilatation

Le résultat de la dilatation de l'ensemble A par l'ensemble B est l'ensemble des points tels que lorsque B est centré sur un de ces points il y a une intersection non vide entre A et B.

Nous allons maintenant nous intéresser à la mise en pratique de la dilatation.

7.2.1 Algorithme de dilatation d'image noir et blanc

Soit k un élément structurant de taille $2\kappa + 1 \times 2\kappa + 1$. L'algorithme de dilatation est le suivant : on parcours tous les pixels de l'image excepté les bords de l'image d'épaisseur κ (on parcourt les lignes de κ à $h - \kappa$ et les colonnes de κ à $w - \kappa$). Pour chaque pixel du fond rencontré, s'il possède un voisin, au sens de l'élément structurant, qui appartienne à un objet, il prend la couleur de l'objet, sinon, il n'est pas modifié. Concrètement, pour une image en noir et blanc, on considère que les pixels noirs appartiennent au fond tandis que les pixels blancs appartiennent aux objets. Dans ce type d'algorithme, il est important d'écrire le résultat dans une nouvelle image afin de traiter chaque pixel indépendemment du résultat des voisins.

Nous présentons un exemple de dilatation en figure 7.2. Dans ce chapitre, tous les exemples donnés utilise l'élément structurant k_s (voisinage en connexité 8).

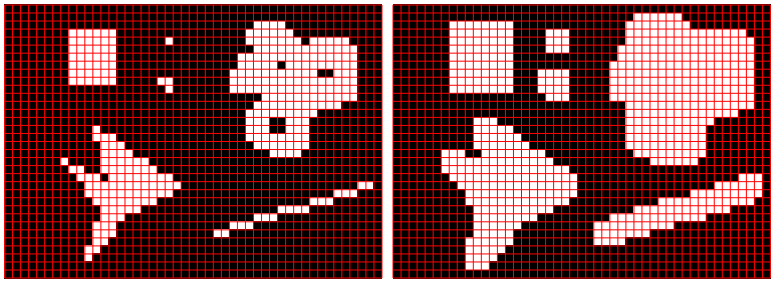


FIG. 7.2 – Exemple de dilatation. A gauche : l'image originale. A droite : sa dilatée.

7.2.2 Algorithme de dilatation d'image en niveaux de gris

Nous devons apporter de légère transformations à l'algorithme précédent. Les pixels du fond sont toujours les pixels noirs ou les pixels de valeurs inférieures à un seuil bas. Les autres pixels appartiennent donc aux objets. La condition de modification d'un pixel est identique. Par contre, dans le cas ou plusieurs voisins appartiennent aux objets, il faut choisir quelle valeur va être utilisée pour colorier le pixel courant : on prend alors la valeur maximale.

7.3 Erosion

L'érosion est définie par :

$$C = A \ominus B = \bigcap (A - b : b \in B) = \{x : B + x \subset A\}$$

On définit la soustraction de Minkowski par :

$$C = A - (-B) = \bigcap (A + b : b \in B)$$

On a alors équivalence entre érosion et soustraction de Minkowski si B est symétrique (si B = -B).

L'érodé de A par B correspond à l'ensemble des points tels que si B est centré sur ces points, B est entièrement inclus dans A.

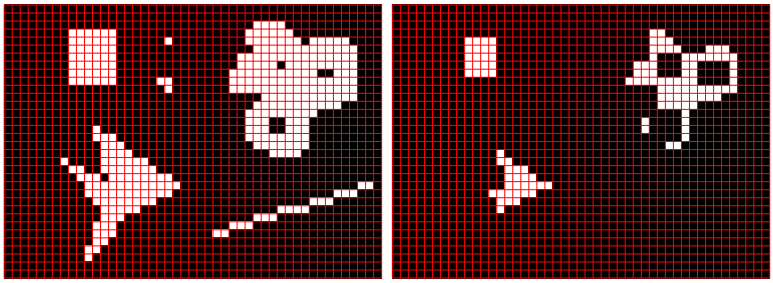


FIG. 7.3 – Exemple d'érosion. A gauche : l'image originale. A droite : son érodée.

7.3.1 Algorithme d'érosion d'image noir et blanc

Cet algorithme ressemble très fortement à l'algorithme de dilatation : on parcourt tous les pixels de l'image excepté les bords de l'image d'épaisseur κ. Pour chaque pixel rencontré n'appartenant pas au fond, s'il possède un voisin, au sens de l'élément structurant, qui appartienne au fond, il prend la couleur de l'objet, sinon, il n'est pas modifié.

Un exemple d'érosion est présenté figure 7.3. Vous pouvez vérifier que si on avait dilaté la video-inverse de l'image, on aurait obtenu la vidéo inverse de ce résultat.

7.3.2 Algorithme d'érosion d'images en niveaux de gris

Tout comme pour la dilatation d'images en niveaux de gris, nous devons apporter de légère transformations à l'algorithme précédent. Les pixels du fond sont les pixels de valeurs inférieures à un seuil. Les autres pixels appartiennent donc aux objets. La condition de modification d'un pixel est donc identique. Par contre, dans le cas ou plusieurs voisins appartiennent au fond, on va choisir la valeur minimale pour colorier le pixel courant.

7.3.3 Dualité

L'application duale d'une application Ψ est définie par :

$$[\Psi(A)]^D = [\Psi(A^c)]^c$$

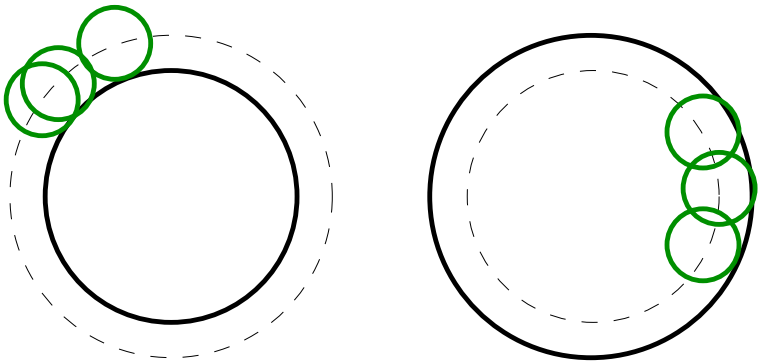


FIG. 7.4 – Erosion et Dilatation

Ainsi, l'érosion par B est duale de la dilatation par -B :

$$C = A \ominus B = (A^c \oplus (-B))^c$$

De même, la dilatation par B est duale de l'érosion par -B :

$$C = A \oplus B = (A^c \ominus (-B))^c$$

Ainsi, si l'on dispose d'une méthode de dilatation d'image par B, on pourra également effectuer des érosions. Pour cela, seule une méthode de vidéo-inverse est nécessaire : le résultat sera la vidéo-inverse de la dilatation de la vidéo-inverse. Et vice et versa pour l'érosion et la dilatation.

On remarquera également que la dilatation est commutative alors que l'érosion ne l'est pas.

7.4 Ouverture et fermeture

L'ouverture (*opening*)consiste en une érosion suivie d'une dilatation :

$$A \circ B = (A \ominus B) \oplus B$$

Les petites structures disparaissent et ne sont par recrées lors de la dilatation. Une opération d'ouverture est ainsi intéressante pour éliminer des petits morceaux de contours bruités dans une image de contours.

La fermeture (*closing*) consiste en une dilatation suivie d'une érosion :

$$A \bullet B = (A \oplus B) \ominus B$$

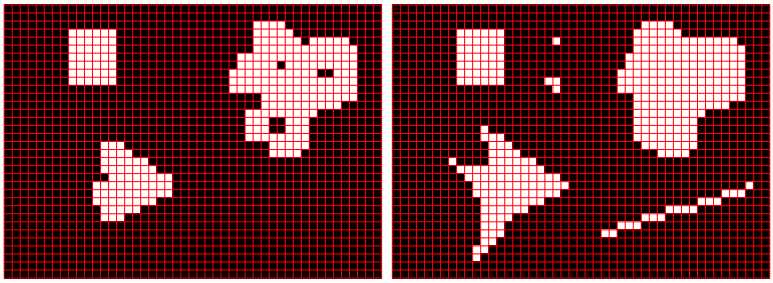


FIG. 7.5 – A gauche : ouverture. A droite : fermeture.

Les structures proches fusionnent lors de la dilatation mais seules les fusions ponctuelles redisparaissent à nouveau lors de l'érosion.

Tout comme il y a dualité entre érosion et dilatation, on retrouve cette dualité entre ouverture et fermeture :

$$A \circ B = (A^c \bullet B)^c$$
$$A \bullet B = (A^c \circ B)^c$$

Ceci signifie que pour effectuer la fermeture d'une image en noir et blanc, il suffit de prendre la vidéo inverse de l'ouverture de la vidéo inverse.

7.5 Gradient morphologique

On obtient un contour intérieur en prenant le complémentaire de A par rapport à son érodé : $A/(A \ominus B)$.

On obtient le contour extérieur en prenant le complémentaire du dilaté par rapport à A : $(A \oplus B)/A$

Le contour moyen, appelé également gradient morphologique est le complémentaire du dilaté par rapport à l'érodé de A : $(A \oplus B)/(A \ominus B)$.

Des exemples figurent en figures 7.6 et 7.7.

7.6 Amincissement et squelettisation

L'intérêt de la squelettisation est dans la reconnaissance de caractères, la détection de réseaux routiers, la planification de trajectoire, ... Sur une image

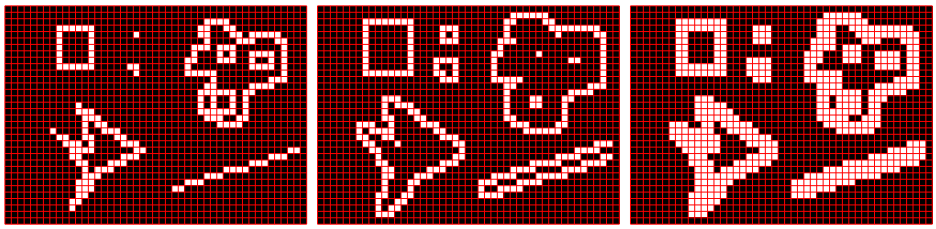


FIG. 7.6 – A gauche : contour intérieur. Au milieu : contour extérieur. A droite : gradient morphologique.

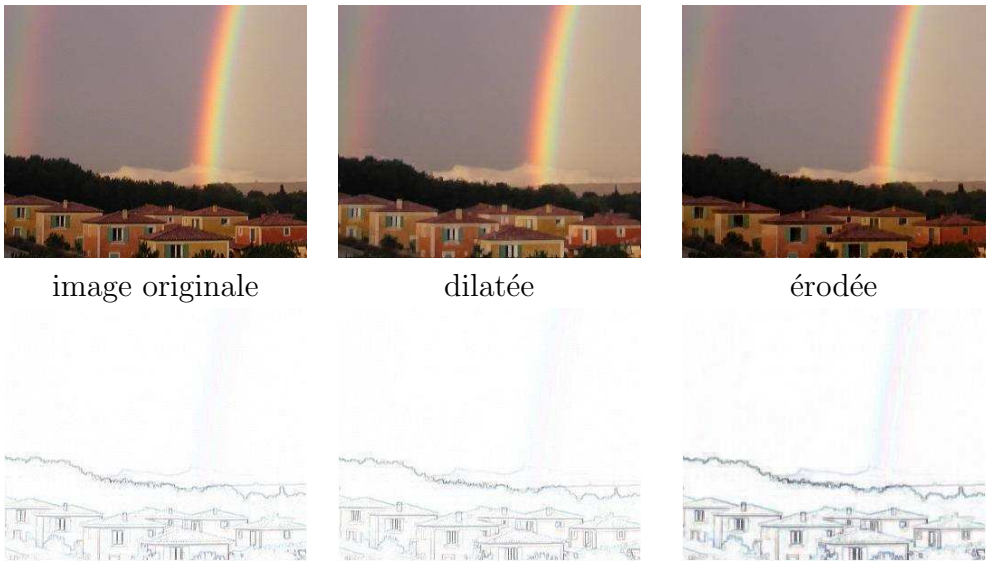


FIG. 7.7 – Un autre exemple de contours morphologiques sur une photographie couleur. On notera que les 3 images de contours sont présentées en vidéo-inverse pour une meilleure impression.

de contours, la squelettisation permet d'affiner les contours ce qui facilitera l'étape ultérieure de chaînage.

L'axe médian (*medial axis*) est la trace des centres des disques maximaux entièrement contenus dans une région. C'est le squelette (skeleton). Pour obtenir un squelette, une méthode couramment employée est un amincissement répété (jusqu'à convergence, c'est-à-dire lorsque que l'amincissement n'apporte plus de modification).

7.6.1 *Hit or Miss transformation*

Nous allons définir cette transformation afin de définir, au paragraphe suivant, l'amincissement :

$$A \otimes B = (A \ominus B) \cap (A^c \ominus B_+$$

où B_+ est le complémentaire local de B. Cela signifie que $A \otimes B$ est l'ensemble des points qui appartiennent à l'érodé de A par B ainsi qu'à l'érodé du complémentaire de A par le complémentaire local de B.

7.6.2 Amincissement (*thinning*)

L'amincissement est défini par :

$$\text{thin}(A) = A/(A \otimes B)$$

ce qui correspond grossièrement à une érosion qui conserve les structures des formes et connexions entre formes. En effet, des érosions successives de A par B conduisent à la disparition complète de A tandis que des amincissements successifs de A par B conduisent au squelette.

Plutôt que d'effectuer l'amincissement en utilisant cette équation, on préfère, pour des raisons d'efficacité, utiliser l'algorithme suivant, en deux étapes (voir figure 7.8).

Pour cela, un tableau de booléens b, de même dimension que l'image I est initialisé à false. Pour chacune des 2 passes, pour chaque pixel (x,y) de l'image, un ensemble de 6 conditions doit être vérifié pour que b[x][y] ne soit pas mis à true. En fin de parcours de l'image, les pixels (x,y) tels que b[x][y] est vrai sont mis à la couleur du fond. On peut, pour simplifier les expressions, considérer que le fond est de couleur noir.

On utilisera également un vecteur de booléens pour déterminer les voisins n'appartenant pas au fond. Les notations suivantes sont utilisées :

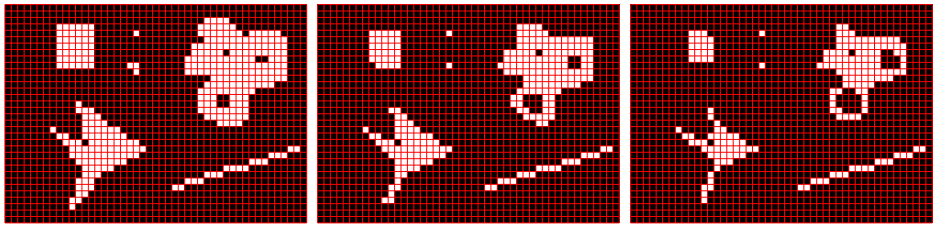


FIG. 7.8 – Algorithme d'amincissement. Image originale puis après la passe 1 et la passe 2.

v[5]=(I[x-1][y-1] !=0)	v[4]=(I[x][y-1] !=0)	v[3]=(I[x+1][y-1] !=0)
v[6]=(I[x-1][y] !=0)	I[x][y]	v[2]=(I[x+1][y] !=0)
v[7]=(I[x-1][y+1] !=0)	v[0]=(I[x][y+1] !=0)	v[1]=(I[x][y+1] !=0)

Pour la première passe, les conditions sont :

1. $I[x][y] == 0$ (le pixel est déjà à la couleur du fond)
2. nombre de voisins < 2 (permet de conserver les squelettes d'épaisseur inférieure à 2)
3. nombre de voisins > 6 (assure d'être en bordure d'un masque 3x3)
4. nombre de transitions `false-true` différent de 1 (permet d'éviter la fragmentation)
5. $v[0]\&\&v[2]\&\&v[4]$ vaut true
6. $v[2]\&\&v[4]\&\&v[6]$ vaut true (ces deux dernières conditions permetent la conservation des lignes connectées)

Pour la seconde passe, les conditions sont identiques sauf les deux dernières :

1. $I[x][y] == 0$
2. nombre de voisins < 2
3. nombre de voisins > 6
4. nombre de transitions `false-true` différent de 1
5. $v[0]\&\&v[2]\&\&v[6]$ vaut true
6. $v[0]\&\&v[4]\&\&v[6]$ vaut true

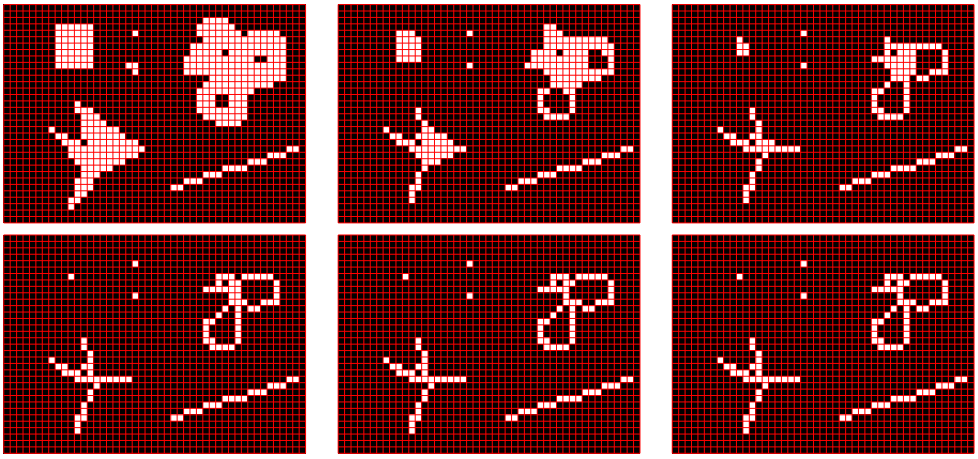


FIG. 7.9 – Amincissements successifs pour l'obtention d'un squelette (de haut en bas puis de gauche à droite).

7.6.3 Squelettisation

Pour obtenir le squelette, il suffit d'appliquer des amincissements successifs et de s'arrêter lorsque l'amincissement n'apporte plus de modification. La figure 7.9 montre les différentes itérations. Le cas présenté propose un squelette après 4 amincissements.

Les figures 7.10 et 7.11 fournissent quelques exemples d'applications.

7.7 Travaux pratiques

Pour cet TP, vous pourrez considérer un voisinage 3x3 en connexité 8.

Implémenter : érosion, dilatation, ouverture, fermeture, gradient morphologique. En étant astucieux, vous pouvez n'implémenter qu'une ou deux méthodes que vous utilisez habilement pour les autres.

Implémentez également l'amincissement et la squeletisation.

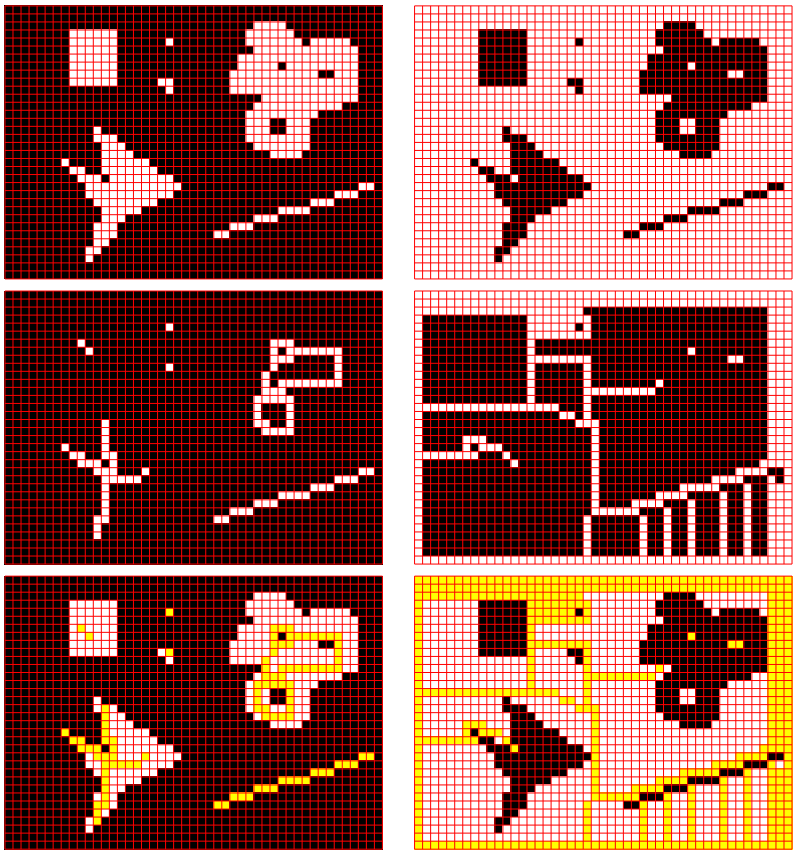


FIG. 7.10 – Dans la colonne de gauche on visualise, de haut en bas, l'image originale, son squelette puis la soustraction des deux en utilisant un filtre de couleur jaune. Dans la colonne de droite, on visualise le même résultat sur la vidéo inverse : on obtient le squelette du fond de l'image originale : c'est l'ensemble des lieux les plus éloignés des obstacles, par exemple.

FIG. 7.11 – Une image d'écriture manuscrite utilisant un crayon épais. En prétraitement de méthodes de reconnaissance d'écriture, on peut utiliser la squelettisation. Il ne subsiste alors que les informations principales.

Chapitre 8

Recherche de primitives : Détection de régions

8.1 Détection de régions par seuillage

L'approche la plus basique pour déterminer une région d'intensité uniforme est le seuillage. C'est une approche qui fonctionne bien sur un certain nombre d'images bien particulières ou d'images qui ont subi des prétraitements, par exemple du lissage. Comme il s'agit d'une méthode très simple et très rapide, il ne faut pas se priver de l'utiliser lorsqu'elle est suffisante[1].

La figure 8.1 présente un exemple de seuillage qui a pour but de séparer le ciel des autres éléments. Le résultat du seuillage est presque satisfaisant : il subsiste des points isolés qui sont mal étiquetés. Deux érosions suivies de deux dilatations d'élément structurant k_s (3x3) (il s'agit finalement d'une ouverture d'élément structurant de taille supérieure) permettent d'éliminer les points erronés. Il faut noter cependant qu'une telle segmentation n'a pas permis de conserver les suspentes du parapente.

Essayez sur vos images. Que pensez-vous du résultat ? Et oui, l'image du parapentiste est bien sympathique. Seulement, dans la plupart des cas, une information d'intensité ne suffit pas pour la segmentation ; il faut tenir compte d'autres informations, notamment des positions relatives des points.

[1] Et pourtant, les shadocks pensent souvent "Pourquoi faire simple quand on peut faire compliqué ?"

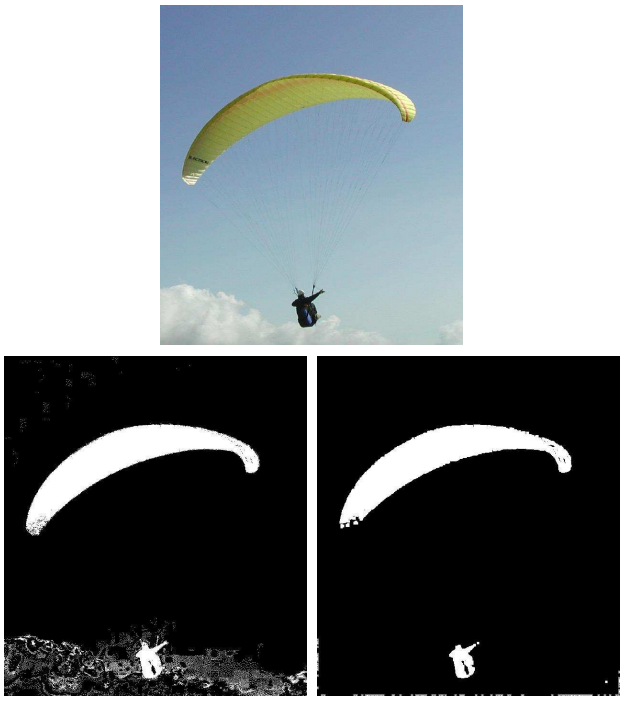


FIG. 8.1 – En haut, l'image originale. En bas à gauche, après lissage (3x3) puis seuillage sur la teinte bleue : certains points isolés erronés apparaissent. En bas à droite : après 2 érosions suivies de 2 dilatations d'élément structurant k_s.

8.2 Détection de contours fermés

La détection de contours fermés passe par le chaînage de contours et la fermeture de contours.

8.2.1 Chaînage de contours

Le chaînage de contours peut s'effectuer par un parcours de points de contours voisins et la fabrication de polygones ouverts. Il peut également se réaliser par une approximation polygonale ou autre d'un ensemble de points (par exemple, par la division récursive d'une droite ou la transformée de Hough).

On suppose ici qu'une détection de contours et un seuillage ont été effectués. On va parcourir l'image ligne par ligne et colonne par colonne et construire de façon récursive des listes de polygones qui sont alors des listes de points. Il convient de ne pas oublier de marquer les points déjà enregistrés dans un polygone.

```
parcours des pixels de l'image
        si le point courant est un point de contour
                créer un nouveau polygone vide
                ajouter le point courant à ce polygone
                marquer le point courant comme deja pris
                si le point en colonne suivante est aussi de contour
                        itérer la construction du polygone courant
                        ajouter le polygone à la liste des polygones
                si le point en ligne suivante est aussi de contour
                        itérer la construction du polygone courant
                        ajouter le polygone à la liste des polygones
                si le point en diagonale descendante est aussi de contour
                        itérer la construction du polygone courant
                        ajouter le polygone à la liste des polygones
```

Une fois ces listes de polygones détectées, il faut maintenant simplifier les polygones et ne retenir que les sommets pertinents. Il faut également essayer de fusionner les polygones qui se suivent.

La figure 8.2 montre la liste de polygones obtenus à l'issu de l'algorithme proposé. On pourra utiliser le codage de Freeman pour représenter ces listes de polygones et les simplifier ensuite.

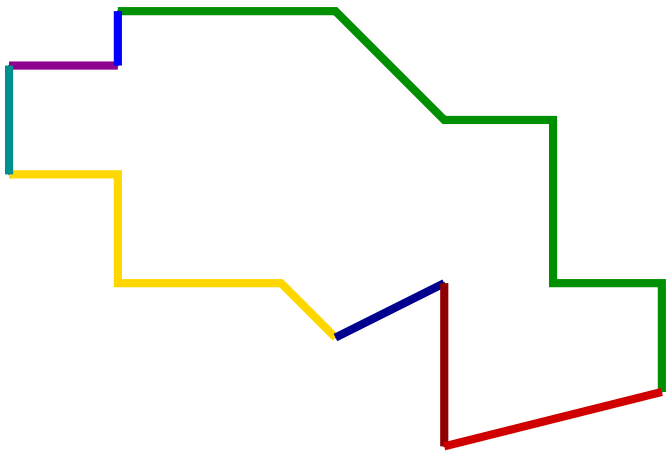


FIG. 8.2 – Chaque polygone détecté est colorié avec une couleur différente.

8.2.2 Code de Freeman

Le code de Freeman permet de coder un contour chaîné en ne stockant que l'ensemble des déplacements relatifs. En effet, l'algorithme présenté au paragraphe précédent conserve les contours sous forme de liste de points et une représentation réduite pourrait être donnée sous la forme d'un point et d'un codade de Freeman.

La figure 8.3 présente les huit directions possibles, ainsi que les numéros associés, permettant de coder le passage d'un point au suivant. Un tel code considère implicitement un voisinage en connexité 8.

Dans l'exemple présenté figure 8.3, on choisit, par exemple, parmi les deux points les plus à gauche, celui le plus haut. A partir de ce point, le code de obtenu est :

`11007764606544133442`

Toute permutation circulaire de ce code conduit à la même forme. Afin de rendre unique le code de Freeman, on impose de plus que le nombre formé soit minimal. Le code de Freeman associé à la figure 8.3 est donc :

`00776460654413344211`

Ce code peut également être compressé en comptant les occurences successives de chaque code élémentaire. C'est également ce calcul de compression

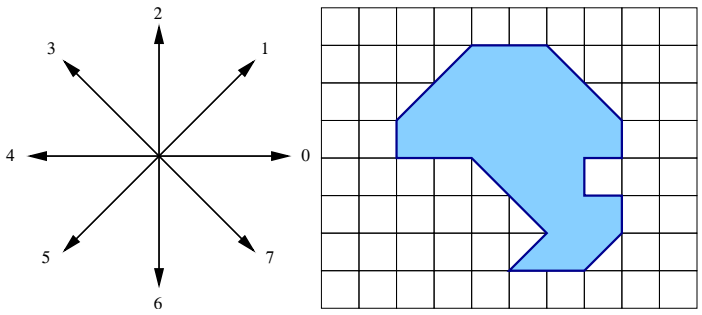


FIG. 8.3 – Code de Freeman. Application sur un exemple.

qui peut conduire à la détermination d'un polynôme (on ne tient compte que des sommets.

On pourra noter aussi que cette représentation permet le calcul aisé de rotation de quart de tour (multiple de $\frac{\pi}{2}$), d'homothétie de rapport entier et de translation entière.

8.2.3 Division récursive d'une droite (*iterative end-point fitting*)

La division récursive d'une droite est une façon de construire un polygone à partir d'une liste de points (points de contours). En partant des points extrêmes P_0 et P_1 (le point le plus à gauche et le point le plus à droite, par exemple), on commence par construire un premier segment reliant ces deux points. On calcule ensuite la somme des distances de tous les points à ce segment. Si cette somme est inférieure à un seul qu'on s'est imposé, on a fini. Sinon, on choisit comme point intermédiaire P_2 le point le plus loin du segment actuel et on recommence sur les segments P_0P_2 et P_2P_1 ... jusqu'à convergence (voir figure 8.4. Evidemment, cette approche ne convient pas dans toutes les configurations (par exemple les contours fermés).

8.2.4 Fermeture de contours

La fermeture de contours consiste à éliminer les trous dans les contours. Il existe différentes méthodes partant des images obtenues après détection de contours. Soyons clair, il n'existe aucune méthode fonctionnant sur tous les types d'images.

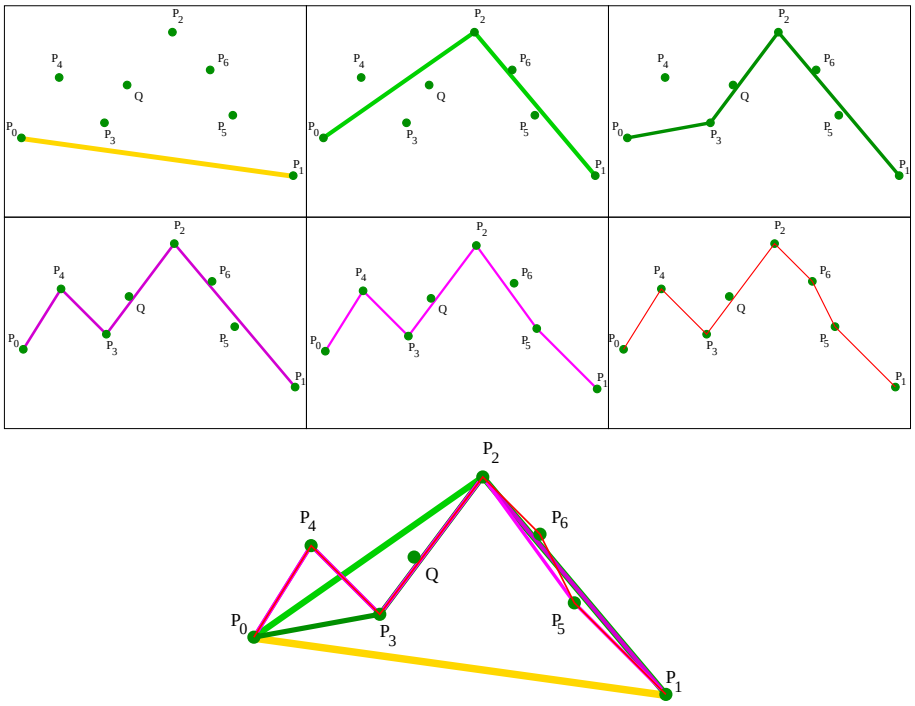


FIG. 8.4 – Division récursive d'une droite : polygones intermédiaires et polygone final obtenu.

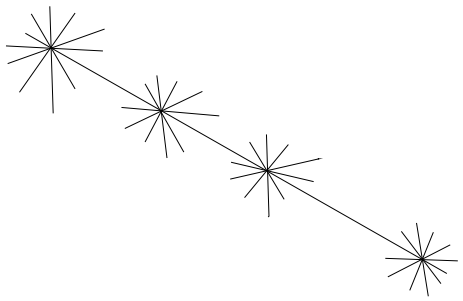


FIG. 8.5 – Chaque point de la droite vote pour toutes les droites passant par lui. La droite qui va remporter le plus de votes est celle qui sera commune au plus de points possibles, c'est-à-dire la droite joignant ces points.

Une première méthode consiste à utiliser la transformée de Hough. Elle fonctionne bien pour des structures polygonales et pas trop nombreuses. Elle peut cependant conduire à des erreurs en fusionnant des segments qui ne devraient pas l'être.

Mais on peut également utiliser la fermeture morphomathématiques. Ou bien tenter de prolonger les contours à leurs extrémités, dans la direction tangente au contour, et déterminer si on rencontre ou non d'autres points de contours.

8.3 Transformée de Hough d'une droite

C'est une méthode connue pour la recherche de droites. Elle est basée sur la représentation d'éléments géométriques dans l'espace des paramètres les représentant. Cet espace des paramètres est quantifié et le tracé dans l'espace des paramètres correspond à un vote. Dans le cas d'une droite, les paramètres sont au nombre de deux : l'espace des paramètres est donc un tableau à deux dimensions.

Pourquoi ça marche ? Pour un point de l'image n'appartenant pas au fond, on va chercher toutes les droites de l'image qui passent par ce point. Chaque droite trouvée vote une fois. Si les points sont alignés, chaque point va voter pour cette droite et elle va ainsi l'emporter sur toutes les autres droites. Une illustration est donnée figure 8.5.

En suivant ce raisonnement, on se rend bien compte que plus un segment

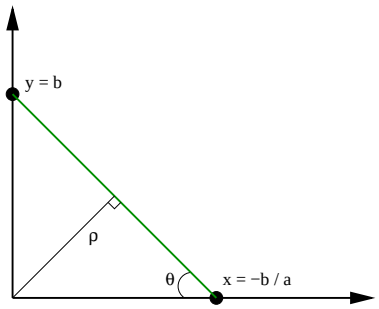


FIG. 8.6 – Paramétrisation d'une droite

est cours, moins il va emporter de votes. Ainsi, dans une images comportant plusieurs segments, le premier trouvé sera le plus long.

La transformée de Hough peut s'appliquer à d'autres formes paramétrées par plus que deux paramètres. L'algorithme est identique. C'est le temps d'exécution ainsi que la place mémoire qui sont bien plus importants.

8.3.1 Algorithme

Une droite peut être paramétrisée selon ses coordonnées polaires (voir figure 8.6) :

$$\rho = x\cos(\theta) + y\sin(\theta) \tag{8.1}$$

ou cartésiennes :

$$y = ax + b \tag{8.2}$$

Considérant la représentation polaire (équation 8.1), l'algorithme est le suivant :

```
Pour chaque point (x, y) de l'image n'appartenant pas au fond
     Pour chaque valeur θ
          calcul de ρ = x cos(θ) + y sin(θ)
          ajout d'un vote pour (ρ, θ) :
               vote[ρ][θ] + +;
```

tandis que pour la représentation cartésienne (équation 8.2), l'algorithme est le suivant :

```
Pour chaque point (x, y) de l'image n'appartenant pas au fond
```

```
Pour chaque valeur a
     calcul de b = y − ax
     ajout d'un vote pour (a, b)
          vote[a][b] + +;
```

8.3.2 Mise en oeuvre de l'algorithme

Lors de l'implémentation de l'algorithme de Hough, plusieurs questions se posent.

`Pour chaque valeur` a : Il faut déterminer l'intervalle $[a_{min}\ a_{max}]$de valeurs possibles de `a` ainsi que le pas de quantification Δade l'intervalle. Ainsi, la traduction de "`Pour chaque valeur` a" sera de la forme :

```
for(int a = aMin ; a <= aMax ; a+= DeltaA) { ...
```

`ajout d'un vote pour` (a, b) Cela signifie qu'il va falloir également quantifier l'espace des paramètres `b`. De plus, lorsque l'on va vouloir dessiner les images de transformées, il va également falloir quantifier les valeurs du tableau de vote, par exemple sur des entiers entre 0 et 255.

Choix des intervalles et pas en représentation polaire. Revenons aux choix d'intervalles et de pas et reprenons le cas de la représentation polaire. Afin d'exprimer toutes les droites présentes dans l'image, on doit considérer tous les angles θ entre 0 et π. La valeur maximale de ρ correspond à la diagonale de l'image, à savoir $\sqrt{w^2 + h^2}$. Sa valeur minimale est pas 0.

Quels pas choisir ? Plus le pas est petit, meilleure sera la solution, plus grande sera l'image de la transformée ainsi que les temps de calcul.

On peut raisonnablement choisir, par exemple, des pas de 1 degrés pour θ et des pas de $\sqrt{w^2 + h^2}/200$.

Afin d'obtenir des résultats plus fins, on peut commencer par une estimation grossière sur les intervalles $[0\ \pi]$ et $[0\ \sqrt{w^2 + h^2}]$ et afiner ensuite dans un intervalle réduit, centré sur la solution obtenue et avec un pas réduit.

Choix des intervalles et pas en représentation cartésienne. Le paramètre a représente la pente qui varie de $-\infty$ à $+\infty$. En pratique, la dernière pente observable sur une image avant la verticale est celle présentant un écart en abscisse de ± 1 pour toute la hauteur (voir figure 8.7).

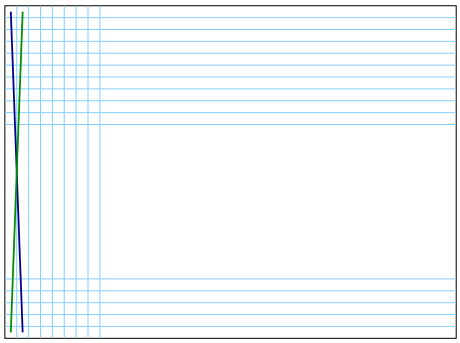


FIG. 8.7 – La droite bleue (resp. verte) est la droite la plus verticale avant la verticale pure, et de pente positive (resp. négative).

Si on exclue les droites totalement verticales, l'intervalle des valeurs de a est $[-h \ +h]$.

Prenons maintenant le cas d'une droite de pente $2 * h + 1$. Lors de sa discrétisation, elle sera assimilée à une droite purement verticale. Ainsi, l'intervalle des valeurs de a est $[-h \ 2h + 1]$.

La valeur maximale de b est obtenue pour une droite de pente minimale, passant par le point $(w - 1 \ h - 1) : b = h - 1 + h(w - 1) = hw - 1$ La valeur minimale de b est obtenue pour une droite de pente maximale, passant par le point $(w - 1 \ 0) : b = 0 - (2h + 1)(w - 1)$ L'intervalle des valeurs de b est donc $[-(2h + 1)(w - 1) \ hw - 1]$

Un exemple : La figure 8.8 présente un exemple de détection de droite dans une image selon les deux représentations. L'erreur observée provient de la quantification des paramètres.

8.3.3 Détection de plusieurs droites

Une première idée intuitive consiste à ne plus simplement considérer le maximum dans l'image transformée mais les maxima. Mais combien de maxima ? humm ...

Supposons tout d'abord que l'on sait ce que l'on cherche (c'est déjà une grande information) et que, par exemple, on s'attende à trouver 4 droites. Et bien, très simple, prenons les quatre valeurs maximales et regardons le résultat (figure 8.9). On vient de mettre en évidence un des problèmes causés

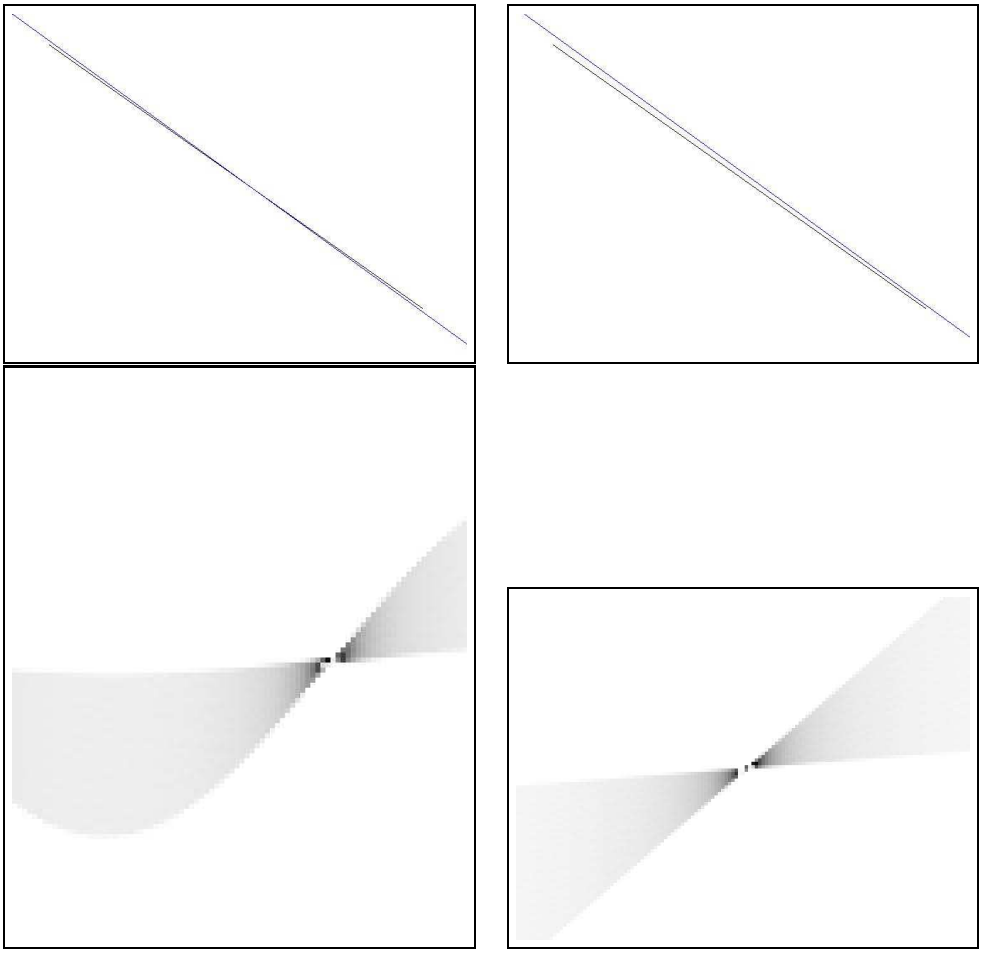


FIG. 8.8 – Nous montrons ici deux exemples, l'un concernant une représentation polaire (colonne de gauche), l'autre une représentation cartésienne (colonne de droite). Nous montrons le segment de droite dans les images (en noir) ainsi que la droite trouvée (en bleu) pour chacune des représentations (ligne du haut) ainsi que les images de transformées (ligne du bas).

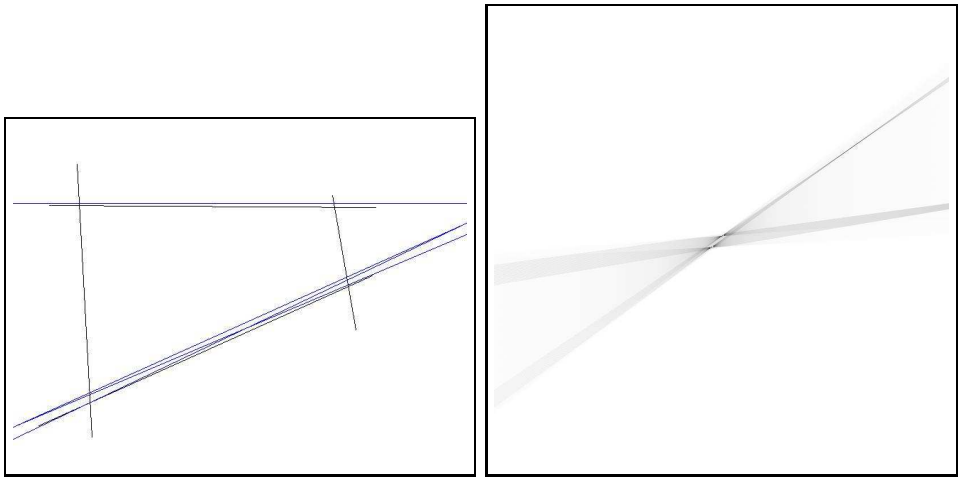


FIG. 8.9 – A gauche : les droites (noires) à détecter et les droites (bleues) trouvées. A droite : la transformée de Hought présentant effectivement 4 maxima. On en déduit qu'il ne faut pas prendre des maxima voisins dans cet exemple.

par la quantification des paramètres. Si le "vrai"paramètre était $\theta = 14.5$ degrés alors que seuls les θ entiers sont considérés, on aura un fort score pour $\theta = 14$ et pour $\theta = 15$, alors qu'il s'agit de la même droite.

Il convient alors de définir une fenêtre d'exclusion autour d'un maxima afin d'éviter de détecter deux fois la même droite. Notons toutefois que pour des figures ou l'on souhaite détecter des droites proches (mais différentes), il conviendra de ne pas prendre des voisinages trop grands.

L'algorithme de détection de droites à partir de la transformée devient donc :

```
Initialisation d'un masque  de la taille de la transformée
     (tous les éléments à true)
Recherche du maximum de la transformée : (x,y)
faire
     Modification du masque au voisinage de (x,y) :
          les éléments prennent false pour valeur
     Recherche du maximum de la transformée selon le masque : (x,y)
tant qu'un maximum supérieur à un seuil est trouvé
```

A titre d'exercice, vous essaierez différentes valeurs de seuil (premier

maximum - 10%, ...) et différentes tailles de voisinages (3x3, ...).

8.3.4 Détection de courbes par transformée de Hough

Nous allons prendre l'exemple d'une courbe conique d'équation $ax^2 + bxy + cy^2 = 1$. La transformée est une image à trois dimensions selon a, b et c. L'algorithme est le suivant :

```
Pour chaque valeur de a entre a_min et a_max
     pour chaque valeur de b entre b_min et b_max
          calcul de c t.q. ax^2 + bxy + cy^2 = 1
          ajout d'un vote pour (a,b,c) : vote[a][b][c]++;
```

En ajoutant une dimension à l'espace de recherche, on augmente la complexité en terme de calculs ainsi que l'espace mémoire utilisé.

8.4 Etiquetage de régions

ou Etiquetage en composantes connexes

L'étiquetage de régions consiste à créer un masque permettant de déterminer à quelle région appartient un pixel : on colorie chaque région avec un numéro de couleur différent sur le masque. Le point de départ est une image binaire : on sait pour chaque pixel s'il appartint ou non au contour. Les contours ont généralement subit quelques traitements pour être fermés. On détermine l'état initial du masque en affectant 0 aux points de contour et 1 aux autres points. L'algorithme s'effectue en deux passes seulement (voir figure 8.10).

Lors de la première passe, les pixels sont parcourus ligne par ligne et colonne par colonne. Une variable de numéro de couleur `couleur` est initialisée à 1. Lorsqu'un pixel rencontré n'est pas un pixel de contour, il prend la valeur de `couleur`. Si le pixel situé juste au dessus n'appartient pas à un contour et qu'il est d'une autre couleur, alors, on mémorise que ces deux couleurs sont identiques dans un tableau de correspondances. Si, on observant que les couleurs 6 et 2 sont identiques de même que 6 et 2, on mémorise les couples avec le second élément le plus faible, à savoir (6 ;1) et (2 ;1). Lorsqu'un pixel de contour est rencontré, la variable `couleur` est incrémentée. Pour l'exemple de la figure 8.10 on obtient le tableau de correspondances de couleurs suivant :

1	2	3	4	5	6	7	8	9	10
	1		1	3	1	3	3	3	3

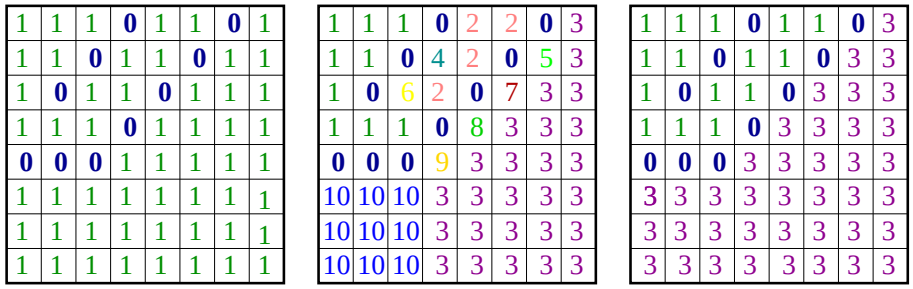


FIG. 8.10 – Algorithme d'étiquetage de régions

Lors de la seconde passe, les couleurs sont mises à jour à l'aide du tableau de correspondances de couleur : lorsqu'une entrée dans la table des correspondances existe, la couleur est remplacée par la valeur dans la table.

8.5 Les algorithmes que nous n'avons pas détaillés.

8.5.1 Agglomération de pixels (*Region growing*)

On part d'un pixel puis on lui attache les pixels voisins s'ils respectent des conditions identiques (faible variation d'intensité, de variance, ...). C'est un algorithme qui agit au niveau du pixel.

8.5.2 Division et fusion (*Split and Merge*)

Ces algorithmes fonctionnent par divisions et regroupements locaux des pixels voisins présentant des propriétés communes. La phase de division fait intervenir des *quadtree* et s'arrête lorsque les carrés satisfont des propriétés d'appartenance à une même région. Les carrés voisins sont alors examinés afin d'effectuer d'éventuelles fusion. Cet algorithme agit à un niveau de détail moins fin que le précédent (on ne descend pas forcément au niveau pixelique).

8.5.3 Approche par classification

Il s'agit de méthodes probabilistes. Les lecteurs intéressés peuvent se reporter aux ouvrages cités dans la bibliographie.

FIG. 8.11 – Recherche du logo ESSI dans une photographie

8.6 Recherche d'un motif

La figure 8.11 présente un exemple de recherche d'un motif : il s'agit de recherchee des images comportant le logo ESSI. Il faut alors détailler les informations disponibles :

- Connait-on la taille du logo ?
- Connait-on la position du logo ?
- Connait-on l'orientation du logo ?

Si l'on connait la taille, l'orientation et la position éventuelle du logo, il est aisé de déterminer la présence ou non du logo en effectuant une convolution du logo avec la portion de l'image ayant pour origine la position donnée. Le résultat de la convolution est entre -1 et +1. Les résultats proches de 1 correspondent à la présence du logo, les autres, à son absence. Le seuil de décision (par exemple 0.90) dépendra du niveau de bruit admissible.

Si l'on ne connait pas tous les paramètres de positions, orientation et taille, on va construire un tableau dans l'espace de ces paramètres de la façon suivante : on parcourt l'espace des paramètres et, pour chaque pixel ou voxel de cet espace, on calcule le score de corrélation. C'est le maximum de cet espace qui va fournir les différents paramètres. Si ce maximum est au dessus du seuil : il y a présence du logo. Sinon, le logo n'est pas présent.

Une autre application : essayer d'éliminer les publicités au milieu des films dans le cadre d'un caméscope numérique. Les débuts et fin de séquences de publicité pourront être repérés par la détection de logos spécifiques.

8.7 Mise en pratique

Implémenter la transformée de Hough en choisissant une paramétrisation de la droite (polaire ou cartésienne). Tracer, sur l'image originale, la droite correspondant au maximum dans l'image de la transformée de Hough. Trouver les droites de l'image. Déterminer le polygone correspondant. Si vous êtes plusieurs, il est intéressant que dans le groupe, certains choisissent la droite y = ax +b tandis que d'autres rho = x cos(theta) ... Et que vous en discutiez après.

Implémenter l'algorithme de chainage de points de contours. Afficher les polygones trouvés avec différentes couleurs. Fusionner les polygones ouverts en polygones fermés.

Chapitre 9

Recherche de primitives : Contours déformables

Dans les chapitres précédents, nous avons cherché à determiner une région soit en chainant des pixels appartenant à un contour de façon à obtenir un contour fermé soit en agglomérant des pixels ayant des propriétés similaires. Nous avons vu que ces méthodes n'étaient pas sans difficultés. Il est souvent difficile de fermer un contour. Il est également difficile de trouver un compromis entre trop ou pas assez de pixels agglomérés.

Nous allons étudier dans ce chapitre des méthodes qui sont basées sur l'évolution d'un contour fermé. Ainsi, on s'affranchit des problèmes de chainages ou de contours parasites.

Le principe consiste à faire évoluer un contour fermé initial vers une position d'équilibre. L'évolution est régie par une équation mettant en jeu des forces à appliquer au contour, généralement selon la normale. Le critère d'arrêt correspond à la minimisation d'une énergie dont la dérivée correspond aux forces à appliquer.

Il convient donc d'étudier maintenant les différentes représentations d'un contour, les critères, les schémas numériques associés ainsi que les tests de convergence.

Un contour peut être représenté par une approximation polygonale, une courbe paramétrée (exemple : BSplines) ou une courbe d'isovaleur 0 d'une image d'ensemble de niveaux.

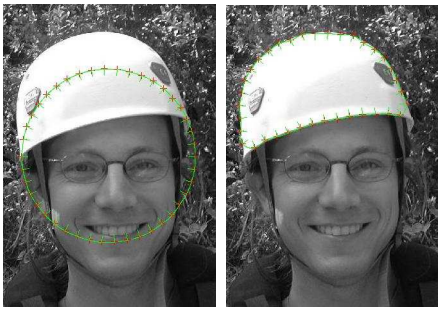


FIG. 9.1 – A gauche : contour initial circulaire. A droite : contour final. Les contours sont ici représentés sous forme de BSplines uniformes.

9.1 Représentation polygonale

Une représentation polygonale consiste à condidérer n sommets P_0, P_1, ... , P_{n-1} appartenant au contour et reliés par des segments de droites. La normale en P_i est définie comme le vecteur normal au segment $P_{i-1}P_{i+1}$ passant par P_i. Cette représentation possède l'avantage d'être très simple à manipuler.

9.2 Courbes ou surfaces paramétrées

Une courbe paramétrée par le paramètre t est un ensemble de points $P(t) = (xt, y(t))$. La normale en P est donnée par :

$$N(t) = (-y', x')/\sqrt{x'^2 + y'^2}$$

La courbure est données par :

$$\kappa(t) = (x'y'' - x''y')/(x'^2 + y'^2)^{(\frac{3}{2})}$$

Ces expressions peuvent être simplifiées dans certains cas.

Nous allons maintenant développer le cas des B-splines uniformes : une courbe B-spline est définie par un ensemble de points de contrôle Q_i ...

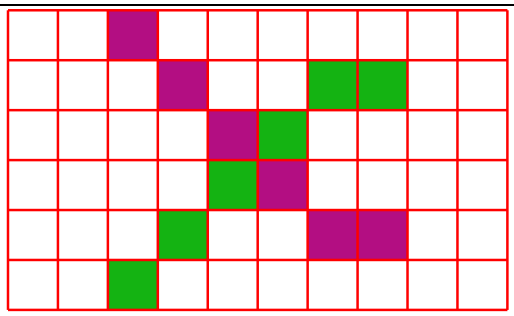


FIG. 9.2 – Il faut se méfier de ce type de configuration et ne pas oublier, lorsque x et y sont modifiés d'examiner (x,yOld) et (y,yOld).

9.2.1 Tracé d'une courbe

Pour chaque point P_i, $O \leq i \leq n-1$, on trace le point P_i. Puis, pour t variant de 0 à 1, on trace le point (x(t),y(t)) sachant :

$$x(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3$$
$$y(t) = b_0 + b_1 t + b_2 t^2 + b_3 t^3$$

Mais, quel pas prendre pour t ? On peut se fixer $\frac{1}{max}$ avec max le maximum de $|x(P_{i+1}) - x(P_i)|$ et $|y(P_{i+1}) - y(P_i)|$. Quelle épaisseur de trait ? Il faudra vérifier qu'on ne fait pas de trou dans la courbe.

9.2.2 Gestion des auto-intersections

On dessine la courbe dans un tableau en changeant de couleur à chaque segment (segment i de couleur (i+1)). Lors du dessin d'un segment, on vérifie que chaque point n'est pas déjà colorié avec une autre couleur. On tolère néanmoins que le premier point d'un segment i soit déjà colorié avec la couleur i (du précédent segment).

Il est important de bien dessiner des contours fermés et d'examiner, lors du passage du point (xOld,yOld) à (x,y), lorsque les deux coordonnées ont été modifiées, les points "cotés" (xOld,y) et (yOld) comme illustré par la figure 9.2.

Lorsque l'on détecte une auto-intersection, on sait qu'il s'agit d'une intersection entre le segment en cours de dessin (le segment i) et le segment correspondant à la couleur déjà présente sur le point que l'on veut dessiner

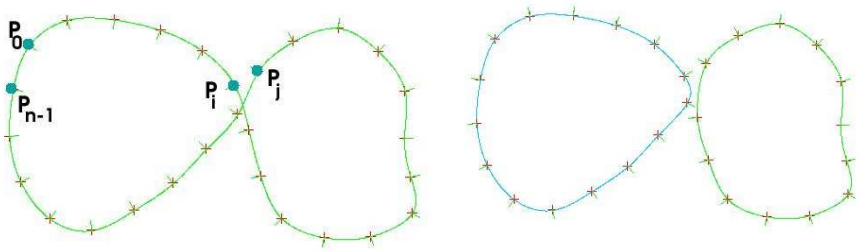


FIG. 9.3 – Elimination d'une auto-intersection.

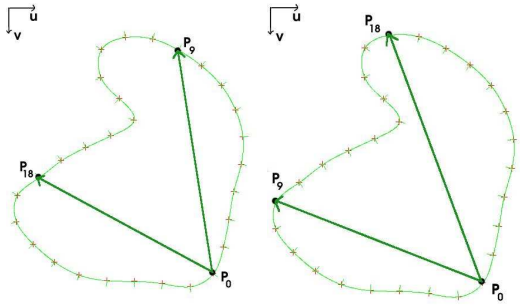


FIG. 9.4 – A gauche, orientation incorrecte. A droite, orientation correcte.

(de couleur j donc de segment j-1). On va redessiner 2 nouvelles BSplines : une comportant les points de P_0 à P_i et P_{j+1} à P_{n-1} et l'autre comportant les points P_{i+1} à P_j. Il conviendra de vérifier l'orientation des courbes ainsi obtenue (voir figure 9.4).

9.2.3 Orientation d'une Bspline

Dans les hypothèses de base, les normales doivent pointer vers l'intérieur de la surface. Une vérification exhaustive de l'orientation est très couteuse alors qu'un test simple mais approximatif s'avère bien efficace : vérifions simplement que les points P_0, $P_{n/2}$ et $P_{2n/3}$ forme un repère direct. On calcule pour cela le produit vectoriel : $P_0P_{n/2} \wedge P_0P_{2n/3} = k\mathbf{w}$ avec $k > 0$.

Il faut garder à l'esprit qu'il existe des configurations où l'orientation peut être correcte mais le repère indirect. Celles-ci sont suffisamment rares pour

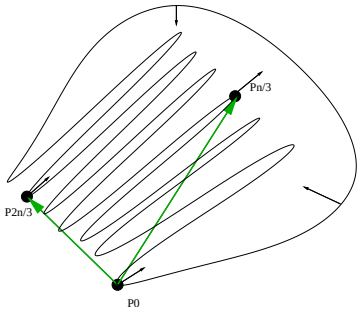


FIG. 9.5 – Ce type de courbe BSpline peut être convenablement orientée alors que le simple test échoue (le repère formé est indirect).

qu'on les néglige. Notamment, dans une application où on cherche des objets dont on a une idée de la forme, il est simple de déterminer si ces configurations ont une chance de se produire (voir la figure 9.5 pour un contre-exemple).

9.2.4 Gestion de l'intérieur / extérieur

On utilise l'algorithme de coloriage de région vu en section 8.4 avec les modifications suivantes :

- chaque Bspline b est tracée avec la couleur (-b)
- les points de contours sont donc des points négatifs, les régions sont étiquetées avec des couleurs positives
- en supposant le point (0,0) hors de toute région, on détermine le fond
- il reste à permuter les étiquettes de région de telle sorte que toute région voisine d'un point de contour de couleur (-b) prenne l'étiquette b.

Cet algorithme ne fonctionne que lorsqu'il n'y a aucune intersection entre BSplines. On peut alors détecter en premier lieu si on est en présence d'intersection ou pas.

Il convient également de s'assurer que les contours dessinés sont effectivement fermés, au sens de la connectivité 4 ou 8.

9.2.5 Gestion des courbes "identiques"

Lors d'un processus de segmentation, il est inutile de segmenter un objet avec plusieurs courbes. On souhaite éliminer les courbes trop "proches". On peut mettre en oeuvre de façon satisfaisante un critère de ressemblance de

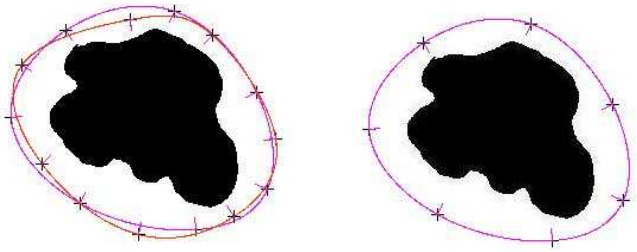


FIG. 9.6 – A gauche : deux courbes segmentent le même objet. A droite : une seule courbe est conservée.

2 courbes : si 80 % des points à l'intérieur de la courbe 1 sont également à l'intérieur de la courbe 2 et constituent au moins 80 % des points à l'intérieur de la courbe 2, alors, les courbes segmentent le même objet.

9.2.6 Les Bsplines uniformes sont uniformes

Cela signifie qu'il faut vérifier que cette propriété est conservée au cours de l'évolution de la courbe et, dans le cas contraire, procéder à la redistribution des points sur la courbe de façon uniforme.

Pour vérifier cette propriété et pouvoir redistribuer les points sur la courbe, il faut pouvoir :

- calculer la longueur de la courbe : c'est la somme des longueurs des segments
- calculer la longueur d'un segment : le calcul explicite est impossible car l'expression de la longueur n'est pas intégrable analytiquement. Il faut alors procéder à une approximation du calcul de l'intégrale.
- insérer un point sur la courbe à la distance d d'un point P_i : il suffit de déterminer t tel que dist(P_i,P(x(t),y(t)) = d. Aucune solution analytique n'existe. Une approximation numérique est également nécessaire.

9.3 Ensembles de niveaux (ou *levelsets*)

Soit une courbe C. On definit une image dont chaque pixel a pour valeur sa distance à la courbe. La courbe est donc le niveau 0 de cette image.

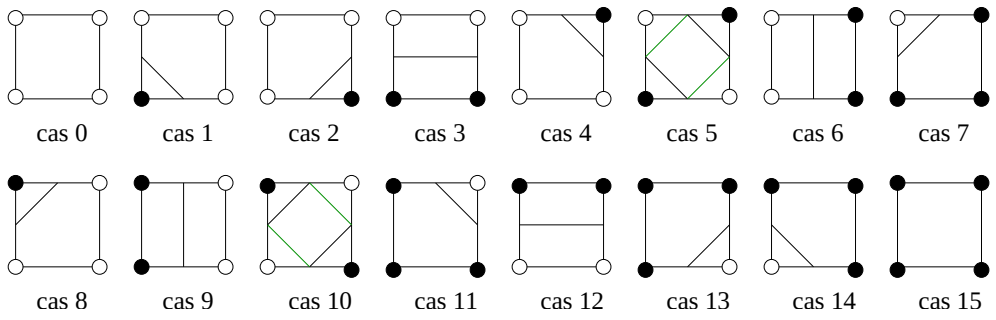


FIG. 9.7 – Les carrées types des *Marching Squares*. On note que les cas 5 et 10 sont ambigüs.

Tracé d'une telle courbe : Méthode des Marching Squares en 2D

(cf *http ://www.marchingcubes.fr.st* par Raphaël Gervaise et Karen Richard, ESSI2 2001/02)

Intérêts :

– Très facilement extensible aux dimensions supérieures.

– Les changements de topologie se gèrent très facilement

Chapitre 10

Restauration d'images

La restauration d'images tend à améliorer la qualité (souvent visuelle) d'une image. Par exemple, une photographie dont le négatif aurait reçu une poussière conduira à un défaut sur l'image. Un bougé lors de l'acquisition d'une photographie pourra conduire à une image floue. Une sous-illumination conduira à des couleurs erronnées ... Certains défauts peuvent être éliminés ou réduits. Mais il ne faut pas non plus attendre que la restauration d'images permettent de restaurer tout et n'importe quoi. Meilleure sera la prise d'image, meilleure sera l'image ensuite.

10.1 Bruit

10.1.1 Causes du bruit

Avant de s'intéresser à la restauration, nous allons d'abord nous intéresser au bruit. Les sources de bruit sont très variées. On peut distinguer plusieurs familles de causes. Une première cause est liée au contexte d'acquisition : une sur ou sous illumination réduit l'intervalle de couleurs de la scène mais pas son nombre alors que le nombre de couleurs utilisées pour représenter cette scène sera réduit dans le cas d'un appareil photo numérique. Parmi les bruits liés au contexte d'acquisition, on peut également citer la perturbation des capteurs, par exemple des perturbations magnétiques pendant une acquisition IRM (Imagerie par Résonance Magnétique), ou des perturbations sur une antenne de télévision lors de la réception sur une carte tuner. Certains capteurs (notamment ceux présents sur les appareils photos) induisent

FIG. 10.1 – Un bien bel example d'image bruitée

des distorsions géométriques ou d'intensité, dûes à la présence d'un certain nombre de dioptres. L'étape d'échantillonnage est source de bruit, notamment de phénomène de Moiré lorsque les conditions de Shannon et Nyquist ne sont pas respectées, ou de bruit poivre et sel lorsque des objets de la scène sont projetés dans une image de la taille d'un pixel. La quantification apporte un bruit dit de quantification. Enfin, la transmission des images est l'occasion de perturbation : perte de données ou corruption des données.

Un exemple d'image bruitée est donné figure 10.1.

10.1.2 Distorsions géométriques

Les distorsions géométriques de l'image sont dûes à des défauts de l'optique des appareils photos : mauvais alignement des centres optiques des dioptres, courbure non parfaite des lentilles, ...

Nous nous intéressons ici à une première approximation de ces distorsions : ce sont les distorsions radiales. En pratique, c'est la distorsion géométrique

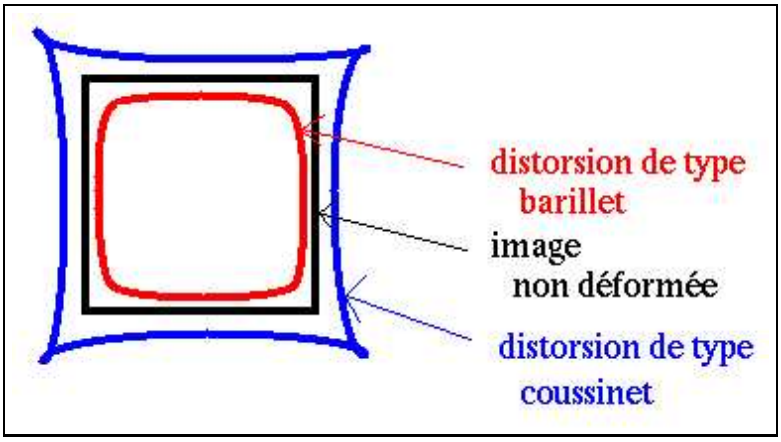


FIG. 10.2 – Les types de distorsions les plus couramment observés : barillet et coussinet

radiale symétrique qui a l'influence la plus importante. Les distorsions géométriques radiales les plus couramment observées sont de deux formes, comme illustré par la figure 10.2 :

- distorsions en forme de barillet
- distorsions en forme de coussinet

On découple généralement la distorsion en deux composantes radiale et tangentielle. Si on note ρ et θ les coordonnées polaires d'un point (x, y) de l'image, $\delta_\rho(\rho, \theta)$ et $\delta_\theta(\rho, \theta)$ les composantes radiale et tangentielle de la distorsion, $\delta_x(x, y)$ et $\delta_y(x, y)$, les composantes selon x et y de la distorsion, s'expriment au premier ordre selon :

$$\left\{ \begin{array}{rcl} \delta_x(x,y) & = & \cos(\theta)\,\delta_\rho(\rho,\theta) - \sin(\theta)\,\delta_\theta(\rho,\theta) \\ \delta_y(x,y) & = & \sin(\theta)\,\delta_\rho(\rho,\theta) + \cos(\theta)\,\delta_\theta(\rho,\theta) \end{array} \right.$$

Les distorsions radiales concernent les défauts de courbure des lentilles constituant la caméra qui induisent des distorsions purement radiales. Cette déformation de la courbure est cependant utilisée dans le but d'avoir une luminosité constante sur toute la surface de l'image, afin d'éliminer l'effet de vignetage (luminosité forte au centre, plus faible en périphérie). Une approximation suffisante en pratique pour modéliser est donnée par :

$$\left\{ \begin{array}{rcl} \delta_x(x,y) & = & k_1\, x\,(x^2+y^2) \\ \delta_y(x,y) & = & k_1\, y\,(x^2+y^2) \end{array} \right.$$

FIG. 10.3 – Image originale : des droites ont été superposées afin de mettre en évidence les distorsions

k_1 étant une constante dont le signe détermine le sens de distorsion ($k_1 > 0$ correspond aux distorsions en forme de coussinet ; $k_1 < 0$ aux distorsions en forme de barillet)

D'autre part, l'effet des distorsions étant minime dans la zone fovéale de l'image, on pourra la négliger si on se concentre sur cette zone.

Dans les images 10.3, 10.4 et 10.5, des droites ont été tracées afin de mettre en évidence la courbure de l'image du batiment de l'ESSI. Cette image a été prise avec un appareil photo argentique standard et développée sur support numérique.

10.1.3 Modélisation du bruit

Le bruit peut être dépendant des données de l'image (par exemple le bruit de quantification) ou indépendant (par exemple les poussières sur l'objectif). On peut alors modéliser le bruit comme additif ou multiplicatif. Il est bien plus simple de traiter un bruit additif que multiplicatif. Afin d'éliminer le bruit, on peut considérer qu'il concerne des hautes fréquences non présentes

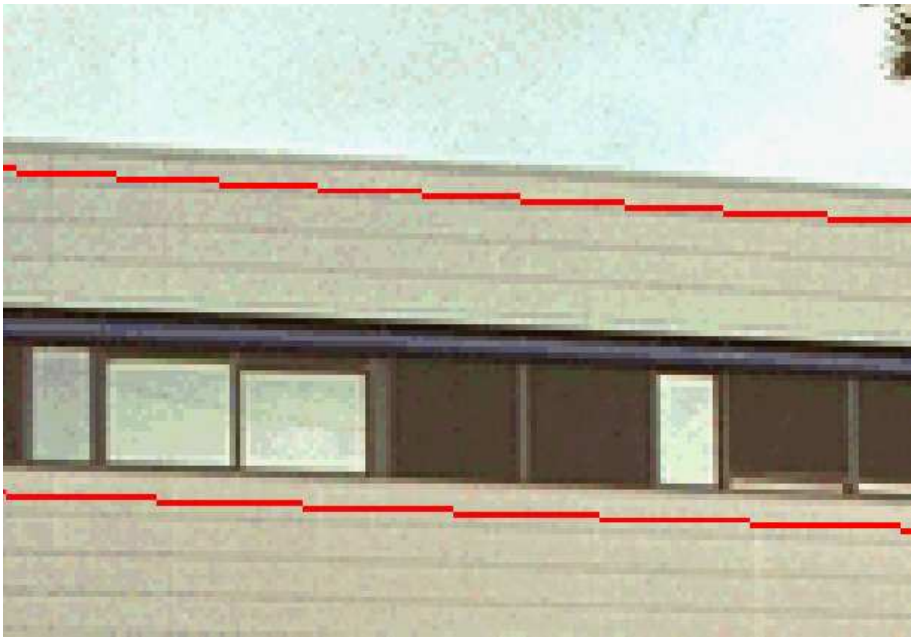


FIG. 10.4 – Zoom sur un détail

FIG. 10.5 – Image corrigée (k=0.03) : on voit ici que les droites tracées coïncident avec les arêtes dans l'image.

FIG. 10.6 – A gauche, l'image originale. A droite, l'image bruitée par un bruit blanc additif de $\sigma = 16$.

dans l'image et qu'il suffit d'effectuer un filtre passe-bas pour améliorer l'image. Malheureusement, cela n'est pas toujours aussi simple.

Nous allons maintenant étudier plus précisément 2 types de bruit : le bruit impulsionnel (bruit blanc gaussien, bruit exponentiel, ...) et le bruit de poivre et sel.

10.1.4 Bruit impulsionnel

Un bruit impulsionnel est défini par une densité de probabilité :

$$f(a) = C.e^{-K|a|^{\alpha}}$$

Pour $\alpha = 1$, il s'agit d'un bruit exponentiel et pour $\alpha = 2$, il s'agit d'un bruit gaussien.

Le bruit blanc est un bruit gaussien de moyenne nulle. Sa variance est σ^2. Ce bruit tend à modéliser le bruit d'acquisition. La figure 10.6 donne un exemple de bruit blanc ajouté à une image.

10.1.5 Bruit de poivre et sel (ou *salt and pepper noise*)

Le bruit de poivre et sel modélise assez bien les poussières sur un objectif ou scanner, des petits objets sur l'image (on imagine par exemple un objet clair sur un fond foncé et dont la taille dans l'image serait proche du pixel : il pourrait apparaitre ou disparaitre lors d'une séquence vidéo créant ainsi du bruit) ainsi que des pertes de données.

FIG. 10.7 – A gauche, l'image originale. A droite, l'image bruitée avec un bruit de poivre et sel de 10 %

Pour bruiter une image de dimension wxh en poivre et sel de p%, il suffit de colorier `whp/2` pixels en noir et `whp/2` pixels en blanc, ces pixels étant choisis aléatoirement. La figure 10.7 donne un exemple d'un tel bruit.

10.2 Restaurer

10.2.1 Mise en garde

Ce n'est pas parce qu'il existe des méthodes de restauration d'images qu'il faut en attendre des miracles. Un bon conseil : ne ratez pas vos photos !

10.2.2 Notions de voisinage

Dans la suite, nous allons considérer des voisinages d'un pixel dont l'étendue et la forme peuvent varier. Nous avions déjà étudié la connexité 4 ou 8. Il existe d'autres formes de voisinage lorsque l'on considère une étendue plus importante : croix, diamant ou carré. Nous présentons ici des exemples en dimensions 3x3 et 7x7. Le "0" signifie que le pixel n'est pas considéré tandis que le "1" signifie qu'il est considéré. Ces masques sont à appliquer à l'image, en les centrant sur le pixel considéré.

Voisinages en dimensions 3x3 :

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix} \qquad \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

croix (connexité 4) carré (connexité 8)

Voisinage en dimensions 7x7 :

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \quad \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

croix diamant carré

10.2.3 Filtrage

Nous allons nous intéresser à du filtrage spatial : c'est-à-dire un filtrage qui s'applique sur un voisinage d'un pixel dans une image. Parmi les différents types de filtrage, certains sont linéaires, s'exprimant sous forme de convolution, d'autres sont non-linéaires (filtrage conservatif, filtrage médian, ...). Les filtres peuvent effectuer plusieurs types d'opérations comme du lissage ou du rehaussement de contours.

10.2.4 Filtre moyenneur

Appelé également *mean filtering*, *averaging* ou *Box filtering*.

Le principe est très simple : un pixel est remplacé par la moyenne de lui même et de ses voisins. C'est dans la définition du voisinage que les filtres vont différer. On peut considérer un voisinage en connexité 4 ou 8, ou même encore plus large.

$$\begin{bmatrix} 0 & \frac{1}{5} & 0 \\ \frac{1}{5} & \frac{1}{5} & \frac{1}{5} \\ 0 & \frac{1}{5} & 0 \end{bmatrix} \qquad \begin{bmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{bmatrix}$$

connexité 4 connexité 8

image originale

image bruitée par un bruit blanc ($\sigma = 16$)

image lissée par un noyau 3x3

image lissée par un noyau 5x5

FIG. 10.8 – On observe que le lissage permet d'éliminer une partie du bruit, notamment dans la partie du ciel mais que les détails sont altérés, notamment sur les maisons et le paysage.

Pour une implémentation plus rapide, on préfera utiliser des filtres avec des coefficients entiers puis diviser ensuite le résultat par la somme des coefficients (cela permet d'effectuer des opérations sur des entiers plutôt que des *float* ou des *double*.

Le filtre moyenneur est un filtre passe-bas permettant ainsi d'éliminer les hautes fréquences, correspondant au bruit. Son inconvénient est qu'il élimine également les hautes fréquences correspondants aux détails de l'image : il rend ainsi l'image moins bruitée mais plus floue (voir figure 10.8).

10.2.5 Filtre gaussien

Appelé également *gaussian filtering*.

Le principe de ce filtrage est une convolution avec une gaussienne.

Nous rappelons l'expression d'une gaussienne en dimension 2, de moyenne nulle :

$$G_\sigma(\mathbf{x}) = \frac{1}{2\pi\sigma^2} e^{(-\frac{|\mathbf{x}|}{2\sigma^2})}$$

Pour effectuer une convolution avec une gaussienne, on utilise un masque de convolution obtenu par discrétisation d'une gaussienne sur un noyau généralement de taille (2p+1 x 2p+1). Certains masques sont à coefficients entiers pour permettre des calculs plus rapides, voire à coefficient puissance de deux (une multiplication ou division par 2 d'un entier revient à un decalage de 1 des bits le composant).

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 1 & 2 & 1 & 1 \\ 1 & 2 & 4 & 2 & 1 \\ 2 & 4 & 8 & 4 & 2 \\ 1 & 2 & 4 & 2 & 1 \\ 1 & 1 & 2 & 1 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 1 & 2 & 2 & 2 & 1 & 1 \\ 1 & 2 & 2 & 4 & 2 & 2 & 1 \\ 2 & 2 & 4 & 8 & 4 & 2 & 2 \\ 2 & 4 & 8 & 16 & 8 & 4 & 2 \\ 2 & 2 & 4 & 8 & 4 & 2 & 2 \\ 1 & 2 & 2 & 4 & 2 & 2 & 1 \\ 1 & 1 & 2 & 2 & 2 & 1 & 1 \end{bmatrix}$$

Exemples de noyaux gaussiens. Attention de ne pas oublier de diviser par la somme des coefficients ! (10.1)

Le lissage gaussien permet de corriger le bruit dans les parties homogènes des images mais est moins efficace que le lissage moyenneur. Cependant, il dégrade moins les détails que le lissage moyenneur (voir figure 10.9).

10.2.6 Filtre conservatif

Ce type de filtrage n'est pas linéaire.

Le principe du filtre conservatif est de ne conserver la valeur d'un pixel donné que si celle-ci se situe à l'intérieur de l'intervalle déterminé par les valeurs des pixels voisins. Si la valeur du pixel est inférieure à la borne inférieure de cet intervalle, le pixel prend pour valeur la borne inférieure. Si la valeur du pixel est supérieure à la borne supérieure, le pixel prend pour valeur la borne supérieure.

Ce type de filtrage n'est pas efficace pour un bruit gaussien mais permet de bien corriger le bruit poivre et sel, lorsque le bruit n'est pas trop important

image originale

image bruitée par un bruit blanc ($\sigma = 16$)

image lissée par un noyau gaussien 3x3

image lissée par un noyau gaussien 7x7

FIG. 10.9 – On observe que le lissage gaussien permet d'éliminer une partie du bruit, notamment dans la partie du ciel mais de façon moins efficace que le lissage moyenneur. On observe également que les détails sont altérés, notamment sur les maisons et le paysage, mais un peu moins qu'avec le lissage moyenneur.

(voir figure 10.10). En effet, le blanc et le noir sont des valeurs extrêmes pouvant être prises par les pixels. Un pixel blanc ou noir isolé est surement dû à du bruit poivre et sel et sera donc à l'extérieur de l'intervalle des voisins. Si le taux de bruit est trop élevé, il devient plus plausible d'avoir un pixel bruité dans le voisinage d'un pixel bruité, ne permettant alors pas de le corriger.

10.2.7 Filtre médian

Le filtre médian (*median filter*) est une amélioration du filtre conservatif. Son principe consiste à remplacer un pixel par la médiane de ses voisins. Ainsi, même si plusieurs pixels voisins sont bruités, on peut corriger le pixel courant. Ce filtre induit cependant un lissage puisque même des pixels corrects peuvent être modifiés. De plus, ce filtrage est plus coûteux car nécessite d'effectuer un tri des voisins pour chaque pixel. Plus le voisinage considéré est plus, plus l'algorithme sera coûteux. On pensera donc, lors de l'implémentation, à utiliser un algorithme de tri rapide tel que le *quick sort*.

Le filtre médian permet d'obtenir de bons résultats sur du bruit poivre et sel mais est aussi peu performant que le filtre conservatif pour le bruit gaussien (voir figure 10.11).

10.2.8 Filtres rehausseurs de contours

Afin de corriger l'effet de lissage des filtres précédents, on peut vouloir utiliser, après restauration, un filtre rehausseur de contours (*edge cripening*). Il s'agit cette fois-ci de l'ajout d'un filtre passe-haut. Ils ont par conséquence l'effet fâcheux d'augmenter le bruit.

Voici différents noyaux de convolution :

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 10 & -1 \\ 0 & -1 & -1 \end{bmatrix} \quad \begin{bmatrix} 1 & -2 & 1 \\ -2 & 5 & -2 \\ 1 & -2 & 1 \end{bmatrix} \quad \begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

On notera que l'effet est augmenté en diminuant le coefficient central de ces noyaux.

image originale

image bruitée par un bruit blanc ($\sigma = 16$)

puis lissée par un filtre conservatif

image bruitée par un bruit poivre et sel (10%)

puis lissée par un filtre conservatif

image bruitée par un bruit poivre et sel (2%)

puis lissée par un filtre conservatif

FIG. 10.10 – Le filtre conservatif permet d'éliminer une partie du bruit poivre et sel mais est assez inefficace pour le bruit gaussien. Cependant, il ne permet d'éliminer correctement le bruit poivre et sel que lorsque le taux de bruit est faible.

image originale

image bruitée par un bruit blanc ($\sigma = 16$)

puis lissée par un filtre médian 3x3

image bruitée par un bruit poivre et sel (10%)

puis lissée par un filtre médian 3x3

puis lissée 2 fois par un filtre médian 3x3

FIG. 10.11 – Le filtre médian est bien plus efficace sur le bruit poivre et sel que sur le bruit blanc. Il est également bien meilleur que le filtrage conservatif mais lisse un peu les détails (regardez les maisons).

image originale

image bruitée par un bruit poivre et sel (10%) puis lissée 2 fois par un filtre médian 3x3

puis rehaussée avec le noyau $\begin{bmatrix} 0 & -1 & 0 \\ -1 & 10 & -1 \\ 0 & -1 & -1 \end{bmatrix}$

image bruitée par un bruit blanc ($\sigma = 16$) puis lissée par un filtre gaussien 7x7

puis rehaussée avec le noyau $\begin{bmatrix} 0 & -1 & 0 \\ -1 & 10 & -1 \\ 0 & -1 & -1 \end{bmatrix}$

FIG. 10.12 – Le filtre médian est bien plus efficace sur le bruit poivre et sel que sur le bruit blanc. Il est également bien meilleure que le filtrage conservatif mais lisse un peu les détails (regardez les maisons).

10.2.9 Conclusion sur les filtres de restauration

Dans une première conclusion, nous pouvons dire que les filtres gaussiens sont plus adaptés au bruit gaussien tandis que les filtres médians corrigent bien le bruit de type poivre et sel. On notera également que les filtres conservatifs corrigent assez bien le bruit poivre et sel lorsque celui-ci n'est pas très important. Par contre, dans tous les cas, on observe une dégradation des détails. Il en vient une idée naturelle qui consisterait à vouloir lisser les zones unies et conserver les zones de détails, par exemple les contours.

C'est cette idée qui est à la base de méthodes de lissage avec préservation des contours. La mise en oeuvre part de l'observation que la solution d'une équation de la chaleur :

$$\frac{\partial u}{\partial t} = \Delta u = \frac{\partial^2 u}{\partial x_1^2} + \frac{\partial^2 u}{\partial x_2^2}$$

est une gaussienne. On sait implémenter le lissage par une gaussienne à l'aide de convolution, mais on peut aussi l'implémenter à l'aide de cette équation. L'intérêt de cette équation, Perona et Malik l'ont bien vu : il suffit d'introduire une fonction c qui dépend du gradient dans l'équation :

$$\frac{\partial u}{\partial t} = \mathrm{div}(c(|\nabla u|^2)\nabla u)$$

et on va choisir le profil de c de telle sorte que dans des zones éloignées des contours (gradient faible), c va être proche de 1 tandis que dans les zones de contours (fort gradient), c va être proche de 0. On peut affiner cela en imposant de plus sur c d'autres propriétés afin de lisser les contours dans la direction tangente et pas normale (ceci impose une contrainte sur le comportement asymptotique à l'infini de c).

Pour en savoir plus sur les méthodes à EDP en imagerie, se reporter à [1].

10.3 Travaux pratiques

Ajouter la possibilité de bruiter des images :
- bruit blanc gaussien
- bruit de poivre et sel

Implémenter des filtres de restauration :

- filtres moyenneurs, gaussiens, rehaussement de contraste
- filtre conservatif (3x3)
- filtre médian (3x3)
- filtres de rehaussement de contours

Bruitez et restaurez en modifiant à chaque fois les paramètres. Observez et comparez les résultats obtenus sur les images.

Chapitre 11

Compression

11.1 Motivations et objectifs

11.1.1 Pourquoi vouloir compresser une image ?

On s'intéresse à la compression à la fois pour du stockage sur des supports de mémoire et pour la transmission. Certes, les capacités des disques durs de nos ordinateurs et le débit des réseaux ne cessent d'augmenter. Cependant, notre utilisation de l'image également.

De nombreux autres appareils numériques ont fait leur apparition. Il ne faut plus parler uniquement de mémoire pour un ordinateur mais également pour un assistant personnel (PDA), pour un téléphone portable, un GPS, un appareil photo numérique ... et les applications actuelles n'envisagent plus de se passer de l'image. Une page web sans image, l'imaginez-vous encore ? Et on essaie de nous imposer l'image de l'interlocuteur au téléphone ...

Dans d'autres domaines professionnels tels que l'imagerie médicale, des masses gigantesques de données sont acquises chaque jour. On chiffre à environ 10 Teraoctets la masse de données d'un service radiologique d'un hopital dans un pays industrialisé (source *Montagnat etal HealthGrid Conf. 2003*).

La compression d'images est donc encore plus d'actualité aujourd'hui.

11.1.2 Objectif de la compression

En fonction de l'application recherchée, différentes qualités vont être demandées à un algorithme de compression. Parmi elles, on cite la rapidité de

la compression et décompression. En effet, il serait dommage, dans une application de transmission, que le temps gagné par une réduction de la taille des données à transmettre soit inférieur au temps passé à la compression ou décompression. Cette qualité sera cependant moins cruciale dans des applications visant à l'archivage de données. Toujours dans un but de transmission d'images, la robustesse de l'algorithme est importante : si un bit est perdu ou modifié, on voudrait éviter de perdre l'image entière.

Viennent ensuite deux qualités antagonistes : le taux de compression et la qualité de l'image après un cycle de compression / décompression. Il existe des algorithmes de compression sans perte mais dont le taux de compression est limité (nous verrons cette limite par la suite). Les algorithmes avec pertes d'informations peuvent obtenir de meilleur taux de compression mais en jouant sur les dégradations. Selon l'application visée, on voudra obtenir une qualité suffisante pour distinguer certaines informations (un panneau routier, ...), ou bien une qualité visuelle parfaite du point de vue d'un humain, ou bien encore conserver la qualité la meilleure possible afin de pouvoir effectuer des traitements ultérieures sur l'image et éviter des artefacts dus à la compression.

11.2 Notions générales à propos de la théorie de l'information

11.2.1 Notion d'histogramme

Histogramme ... ou l'inventaire des couleurs.

définitions

Histogramme h[i] :nombre d' occurrences de l'intensité i

Histogramme cumulé (cumulative histogram) h[i] = h[i-1] + nb. d'occurences de i

Probabilités estimées : à partir des valeurs de l'histogramme divisées par le nombre total de pixel

Probabilités cumulées : même chose à partir de l'histogramme cumulé

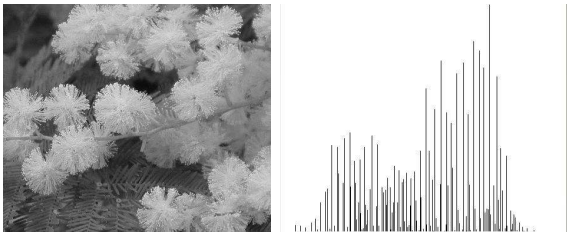


FIG. 11.1 – image en niveaux de gris et son histogramme

propriétés

- Information sur la répartition des intensités : moyenne, variance, énergie, entropie, contraste, illumination, ...
- Ne représente pas la répartition spatiale : deux images peuvent posséder le même histogramme sans pour autant se ressembler
- Certaines transformations n'ont aucune influence sur l'histogramme

Calculs d'informations à partir d'un histogramme On note $p(b)$ la probabilité estimée, c'est-à-dire $h(b)/N$.

moyenne :

$$\mu = \sum_{b=0}^{255} b.p(b)$$

variance :

$$\sigma^2 = \sum_{b=0}^{255} (b-\mu)^2.p(b)$$

énergie :

$$\sum_{b=0}^{255} p(b)^2$$

entropie :

$$-\sum_{b=0}^{255} h(b).\log_2(p(b))$$

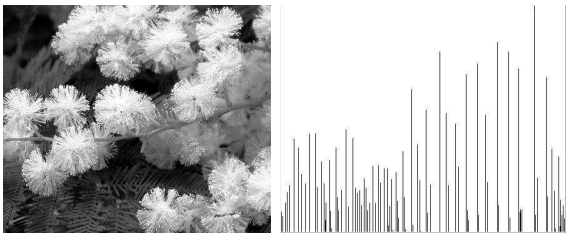


FIG. 11.2 – Egalisation de l'histogramme et image résultat.

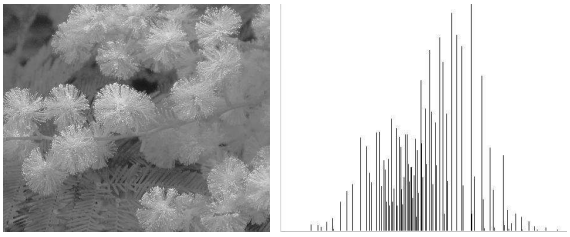


FIG. 11.3 – Normalisation de l'histogramme et image résultat.

Opérations sur les histogrammes

- Egalisation : dans le but d'améliorer le contraste, on alloue + de niveaux d'intensités dans le zones hautes de l'histogramme et vice-versa.
- Normalisation
- Recadrage, dilatation des zones claires, dilatation des zones sombres, extraction d'une fenêtre d'intensité, vidéo inverse, correction non linéaire.

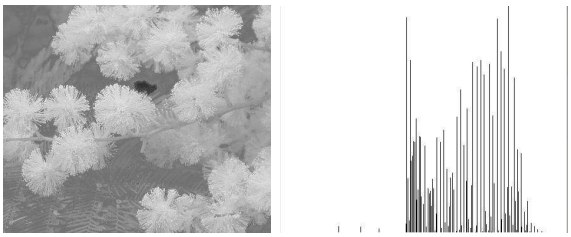


FIG. 11.4 – Piecewise

11.2.2 Quantité d'informations

La théorie de l'information permet d'évaluer la quantité d'information dans une image. Pour cela, chaque point d'une image est considéré comme une variable aléatoire. Une image est alors considérée comme une suite de w*h variables aléatoires.

Soit P un point d'une image en niveaux de gris. C'est une variable aléatoire dont les valeurs sont des entiers de l'intervalle [0 255]. Soit $p(n_i)$ la probabilité pour que le niveau de gris en P soit n_i. La quantité d'information est donnée par :

$$I(n_i) = \log_a(\frac{1}{p(n_i)})$$

On définit alors l'entropie par analogie avec la thermodynamique. L'entropie d'un point :

$$H(P) = \sum_{i=0}^{255} p(n_i)I(n_i) = \sum_{i=0}^{255} p(n_i)\log_2\frac{1}{p(n_i)}$$

où 255 est la valeur maximale de niveau de gris, l'entropie H(P) est exprimé en bits et le logarithme est en base 2 car un bit peut avoir 2 valeurs différentes.

Sous certaines hypothèses, notamment que les propriétés statistiques ne dépendent pas du temps (c'est l'hypothèse de stationnarité) et que les moyennes temporelle et statistiques correspondent (c'est l'hypothèse d'ergodicité), l'entropie d'un bloc est définie par :

$$H(P^L) = LH(P)$$

11.2.3 Théorème de codage sans pertes

Le théorème de codage sans pertes permet de déterminer le nombre de bits minimal pour coder une image sans perdre d'information. Les pixels sont codés par blocs avec un mot binaire par bloc. La taille des mots binaires mot_i est variable est vaut l_i. La longueur moyenne de la taille des mots binaires est notée l :

$$l = \frac{1}{L}E\{l_i\} \text{ avec } :E\{l_i\} = \sum_{i=1}^{m} p(l_i)l_i = \sum_{i=1}^{m} p(B_i)l_1$$

Le théorème du codage sans pertes s'énonce par :

$$\forall\delta > 0 \;\; \exists L \;\; \forall\text{mot}_i \;\; H(P) \leq l < H(P) + \delta$$

Cela signifie deux choses. La première est qu'il existe une borne minimale en dessous de laquelle on ne peut pas descendre et qui se calcule : c'est l'entropie. La seconde est qu'on peut théoriquement s'approcher aussi près de cette borne qu'on le souhaite à condition de choisir convenablement la taille des blocs.

11.2.4 Taux de compression

Le taux de compression est défini comme le rapport du nombre de bits utilisés par l'image originale et du nombre de bits utilisés par l'image compressée.

Les méthodes réversibles ont un taux de compression entre 1 et 2.5 tandis que les méthodes irréversibles peuvent avoir de bien meilleur taux de compression mais avec une distorsion.

La mesure de la distorsion est un sujet difficile. Il existe deux mesures couramment employées : l'erreur quadratique moyenne EQM (ou MSE pour Mean Square Error) :

$$EQM = \frac{1}{N}\sum_{i=1}^{N}(\hat{n_i} - n_1)^2$$

et le rapport signal à bruit crête PSNR (*Peak Signal Noise Ratio*) :

$$PSNR = 10\log_{10}(\frac{NdG_{max}^2}{EQM}) \text{ en dB}$$

11.3 Présentation d'algorithmes de compression

Quelques idées pour la compression Les principales idées pour la compression sont basées sur (i) la quantification des niveaux de gris ou composantes couleurs ou bien des coefficients dans les images transformées, (ii) la mémorisation des occurrences (on remplace la chaine "0 0 0 0 0" par "5 0") et (iii) le codage des valeurs avec un code de longueur inversemenet proportionnelle aux occurrences.

Nous allons maintenant étudier deux algorithmes largement utilisés : le codage d'Huffman (utilisé entre autres dans le format JPEG) et le codage LZW (utilisé entre autres dans le format GIF).

11.3.1 Codage d'Huffman

Principe du codage d'Huffman

- Source de m éléments
- Principe du codage d'Huffman
  - l'alphabet de la source ne comporte que 2 symboles : 0 et 1
  - à un niveau donné m, on combine deux symboles de probabilités minimales. On obtient alors un ensemble de niveau (m-1) comportant un élément de moins. Le code de chaque élément est identique, sauf pour les combinaisons
  - les codes associés aux 2 caractères combinés à un niveau m sont obtenus à partir de leur code au niveau inférieur (m-1) auquel on ajoute 0 et 1
- Algorithme en deux phases
  - phase ascendante : on part de l'ensemble entier vers un ensemble à deux éléments (l'un codé 0, l'autre codé 1)
  - phase descendante : on détermine tous les codes depuis les deux derniers éléments jusqu'à l'ensemble entiers en concaténant 0 ou 1 à chaque noeud de l'arbre remonté

Exemple de codage

valeur n_i	n_0	n_1	n_2	n_3	n_4	n_5	n_6	n_7
probabilités	0.35	0.15	0.12	0.11	0.10	0.09	0.05	0.03

- phase descendante :
  - 1ère étape : passage de m=8 à m=7 : n6 et n7 ont les probabilités les plus faibles : on les regroupe en un seul élément n7,8 de probabilité 0.08
  - ...
  - dernière étape : il reste un unique élément de probabilité 1
- phase ascendante : on concatène les codes à chaque branche de l'arbre

Gain du codage d'Huffman

- Codage ordinaire : sommes des p(ni)*3bits
- Codage d'Huffman : somme des p(ni)*li
- Dans notre exemple : 3 bits contre 2.57 bits
- image (640*480) : économie de 132096 bits=13 kO

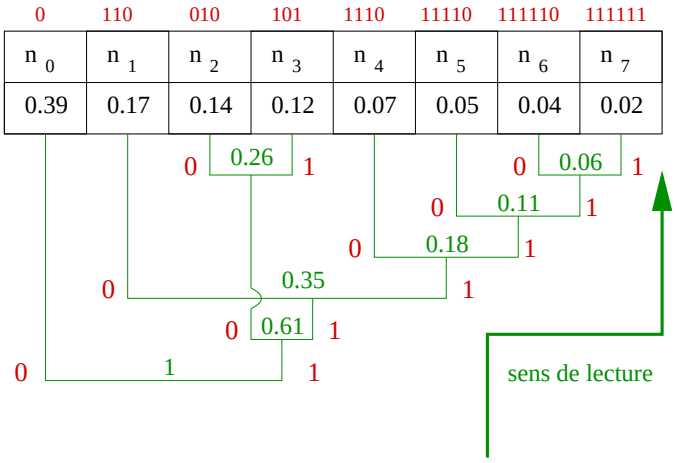


FIG. 11.5 – Codage d'Huffman

11.3.2 LZW ou Lempel-Ziv Welch

Utilisé :
- dans le format gif (images couleurs 8 bits)
- peut être utilisé dans le format tiff
- dans les fichiers .Z (compress)
- dans les fichiers .gzip (gnu zip)

Fait l'objet d'un copyright détenu par Compuserve et Unisys
- Compression sans perte (réversible)
- Fonctionne bien pour les images présentant de grandes zones homogènes
- Algorithme : découpage de l'ensemble des pixels en mots et attribution d'un code à chaque mot
- Considère les pixels comme 1 tableau monodimensionnel (ne tient pas compte des redondances verticales)
- Découper la suite de pixels en chaînes les + longues possibles
- Construction d'une table :
  - on commence par les pixels individuels
  - puis toute chaîne de + en + longue
- Le code d'une chaîne est indépendant de la taille de la chaîne

codage LZW

```
chaîne précédente <- chaîne vide
```

```
tant que (lecture d'un caractère c)
        si concaténation(mot+c) existe dans le dictionnaire
                mot <- concaténation de mot et c
        sinon
                ajouter  concaténation de mot et c dans le dictionnaire
                retourner le code de mot
                mot <- c
```

Dans l'exemple suivant, nous considérons $@_0$ la première adresse disponible pour stocker notre table. La chaîne que nous allons coder est la suivante :

ABRACADABRACADA ...

mot	c	mot + c	existe ?	retour	entrée	adresse
A	B	AB	non	@(A)	AB	$@_0$
B	R	BR	non	@(B)	BR	$@_0$ +1
R	A	RA	non	@(R)	RA	$@_0$ +2
A	C	AC	non	@(A)	AC	$@_0$ +3
C	A	CA	non	@(C)	CA	$@_0$ +4
A	D	AD	non	@(A)	AD	$@_0$ +5
D	A	DA	non	@(D)	DA	$@_0$ +6
A	B	AB	oui			
	R	ABR	non	@(AB)=$@_0$	ABR	$@_0$ +7
R	A	RA	oui			
	C	RAC	non	@(RA)=$@_0$+2	RAC	$@_0$ +8
C	A	CA	oui			
	D	CAD	non	@(CA)=$@_0$ +4	CAD	$@_0$ +9

décodage LZW

```
mot précédent initialisé à vide
tant que (lecture d'un code)
        mot courant <- le contenu de l'@code
        retourner mot courant
        c <- 1er caractère de mot courant
        à la 1ère @ libre : stocker la concaténation de mot précédent. + c
        mot précédent <- mot courant
```

Exemple :

code reçu	mot courant	sortie	c	entrée	@	mot
A	A	A				A
B	B	B	B	AB	$@_0$	B
R	R	R	R	BR	$@_0$ +1	R
A	A	A	A	RA	$@_0$ +2	A
C	C	C	C	AC	$@_0$ +3	C
A	A	A	A	CA	$@_0$ +4	A
D	D	D	D	AD	$@_0$ +5	D
100	AB	AB	A	DA	$@_0$ +6	AB
102	RA	RA	R	ABR	$@_0$ +7	RA
104	CA	CA	C	RAC	$@_0$ +8	CA

ABRACADABRACADA ...

11.3.3 Transformées discrètes

Il existe la transformée de Fourier discrète. Or cette dernière possède des coefficients complexes. On lui préfère ainsi la transformée en cosinus discrète (DCT ou *discret cosin transform*) dont les coefficients sont réels et plus petits.

$$n_{\text{dct}}(u,v) = \frac{2}{N} C(u)C(v) \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} \cos\left[\frac{\pi}{N} u(i+\frac{1}{2})\right] \cos\left[\frac{\pi}{N} v(j+\frac{1}{2})\right] n(i,j)$$

avec $C(0) = \frac{1}{\sqrt{2}}$ et $\forall \alpha \;\; C(\alpha) = 0$

11.4 Exemple de JPEG

Une valeur optimale de taille de blocs pour JPEG est estimée à 8. Dans le codage JPEG, l'image est ainsi codée par blocs de 8x8 pixels. On effectue alors la transformée en cosinus discrète de chacun de ces blocs. Le résultat est alors quantifié. C'est cette étape de quantification qui apporte des pertes de données mais qui permet également de faire varier le taux de compression.

Les coefficients les plus importants sont situés en haut à gauche de l'image transformée. Au parcours ligne par ligne, on préfère le *Zig-zag scanning* (voir figure 11.6) qui consiste à parcourir le bloc 8x8 de façon à lire en premier les coefficients forts puis de terminer par les plus faibles et d'augmenter la probabilité d'obtenir un nombre important de zéros consécutifs.

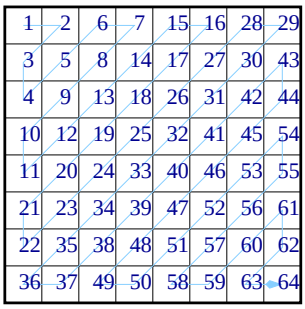


FIG. 11.6 – Le zig-zag scanning est un mode de parcours d'un bloc 8x8.

Les zéros sont ensuite codés sous forme du nombre d'occurences par un codage par plages (ou *run length coding*) : il s'agit de compter le nombre de zéros séparant deux valeurs non nulles et de remplacer les zéros par leur occurence. Exemple :

```
... 200 0 80 0 0 4 0 0 0 0 1 ...
```

est remplacé par :

```
... 0 200 1 80 2 4 4 1 ...
```

Ces données sont ensuite compressées par le codage d'Huffman.

Le tableau de la figure 11.7 présente différents résultats de compression ainsi que la taille des fichiers résultants (il s'agit d'une image de taille 640x480).

11.5 Introduction à JPEG 2000

La principale différence sur le principe de l'algorithme est le remplacement de la transformée en cosinus discrète (DCT) par une décomposition en base d'ondelettes.

Le principe des ondelettes, c'est la décomposition du signal sur une base d'ondelettes. Une base d'ondelettes est générée par dilatation et translation d'une ondelette mère :

$$\psi_{a,b}(x) = |a|^{\frac{1}{2}}\psi(\frac{x-b}{a})$$

JPEG originale : 88 kO

GIF : 232 kO

JPEG à 50 % : 68 kO

JPEG à 25 % : 36 kO

JPEG à 12 % : 20 kO

JPEG à 5 % : 12 kO

JPEG à 1 % : 8 kO

FIG. 11.7 – Quelques compressions.

avec $a \neq 0$.

On obtient alors une base orthogonale :

$$\psi_{m,n}(x) = 2^{-\frac{m}{2}}\psi(2^{-m}x - n)$$

et des coefficients d'ondelettes :

$$c_{m,n}(f) =< f, \psi_{m,n} >= \int f(x)\bar{\psi}_{m,n}(x)dx$$

Parmi les différentes bases, on peut citer la base de Haar :

$$\psi(x) = \begin{cases} 1 & \text{si} \quad x \in [0, \frac{1}{2}[\\ -1 & \text{si} \quad x \in [\frac{1}{2}, 1[\\ 0 & \text{ailleurs} \end{cases}$$

Les principaux avantages de JPEG 2000 sont :
- train binaire imbriqué : on peut tronquer un fichier
- train binaire organisé de façon progressive
- compression avec ou sans pertes efficace
- des régions d'intérêts peuvent être sélectionnées pour différents taux de compression
- mécanisme de résistance aux erreurs

Une implémentation de JPEG 2000 est disponible à l'adresse suivante : `http ://jpeg2000.epfl.ch`.

11.6 Compression vidéo

caractéristiques vidéos
- nombre de frames par seconde
- dimension des frames
- stockage de composantes couleurs sur les frames
- débit
- ...

11.6.1 MJPEG

Le MJPEG est un acronyme pour Motion JPEG.

11.6.2 MPEG

Les différents formats :

MPEG 1 (1992) : norme des images animées à faible résolution pour des applications multimédias. Qualilté VHS, compatible CDRom et CD Vidéo. Débit : 1.5 Mbits/s

MPEG 2 (1994) : reprise de MPEG 1 en améliorant la qualité. Norme utilisée sur les DVD vidéo. Débit de 3 à 10 Mbits/s en vidéo standard et de l'ordre de 300 Mbits/s en haute définition.

MPEG 3 : norme prévue initialement pour la vidéo haute qualité mais finalement absorbée par MPEG2

MPEG 4 : norme de codage orienté objet (nécessite une segmentation)

MPEG 7 : moyen de description normalisé du contenu multimédia

SIF : Le SIF (*Source Intermediate Format*) est un format résultant de la réduction de moitié de la résolution spatiale et temporelle. Cela correspond à 360 pixels par 288 lignes à 25 Hz pour la luminance, 180 pixels par 144 lignes pour les chrominances.

GOP : Le GOP (*Group Of Pictures*) représente un certain nombre d'images consécutives que l'on va coder en se référant à la première. Il est composé de trois types d'images :

I (intra) : sous forme SIF, codée en JPEG

P (prédite) : prédite à partir d'une image I ou P. Elle est codée uniquement à partir de vecteurs mouvement et peut propager des erreurs.

B (bidirectionnelle) : construite par interpolation bidirectionnelle des I ou P passés et futures, à l'aide des vecteurs mouvement. C'est la moins volumineuse des trois types d'images. Elle ne peut pas propager d'erreur (aucune autre image n'est construite à partir d'une B).

Dans un GOP, la première image est de type I. C'est d'ailleurs la seule image de type I. Les images I sont les plus volumineuses. Ainsi, plus le GOP est long, plus la compression est élevée... et la qualité réduite. On comprend bien que sur une scène globalement fixe, la qualité sera bien meilleure que sur une scène ou il y aura beaucoup de mouvement (par exemple un changement de décor).

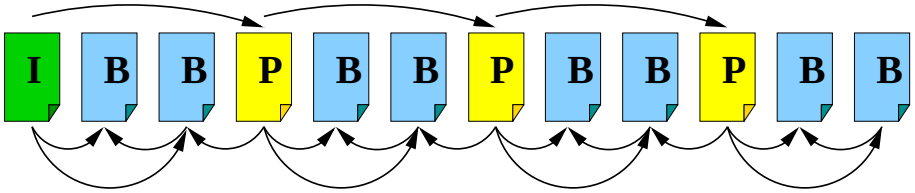


FIG. 11.8 – Un GOP typique est composé de 12 images.

Une organisation typique pour un GOP est de 12 images. On note souvent M l'intervalle entre deux P et N l'intervalle entre deux I. Des valeurs typiques sont donc $M = 3$ et $N = 12$.

Lorsque l'on souhaite accéder à une image, on va accéder à une image I. Il n'est pas possible de couper un GOP : s'il manque une image, on ne pourra pas reconstruire celles qui en dépendent. Le GOP est ainsi l'unité d'accès aléatoire.

Lors de la décompression, certaines images dépendent des suivantes : on ne pourra donc décoder entièrement un GOP que lorsque celui-ci sera entièrement reçu. Le GOP représente donc aussi le retard à la décompression.

Estimation du mouvement L'estimation du mouvement entre deux images se fait au niveau de macroblocs . Ces macroblocs sont constitués de 4 blocs de luminance et 2 ou 4 blocs de chrominance. La notion de bloc est similaire aux blocs utilisés lors de la compression JPEG : il s'agit de blocs de 8x8 pixels.

Nous ne détaillons pas les algorithmes d'estimation de mouvement mais précisons uniquement les différentes étapes :

- recherche des macroblocs semblables entre une image et la précédente (correspondances)
- calcul des vecteurs mouvement
- construction de l'image prédite à partir de l'image précédente et des vecteurs mouvement
- comparaison entre l'image courante et l'image prédite et calcul des erreurs de prédiction
- codage et transmission des vecteurs mouvement et des erreur de prédiction

Bibliographie

[1] Gilles Aubert and Pierre Kornprobst. *Mathematical problems in image processing : Partial Differential Equations and the Calculus of Variations*. Springer : Applied Mathematical Science, 2001.

[2] Philippe Bellaïche. *Les secrets de l'image vidéo*. Eyrolles, 2004.

[3] J.-P. Cocquerez and S. Philipp. *Analyse d'Images : filtrage et segmentation*. Masson, 1995.

[4] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing*. Prentice Hall, Collection Informatique, 2002. Second Edition.

[5] Jean-Paul Guillois. *Techniques de compression*. Hermès, Paris, 1996.

[6] Joël Leroux. *Notes de cours sur le traitement numérique du signal*. http ://www.essi.fr/ leroux, 2004.

[7] Tomas Lozano-Perez and Jovan Popovic. *Lecture Notes*. http ://graphics.lcs.mit.edu/classes/6.837/F01/notes.html, 2001.

[8] Douglas A. Lyon. *Image Processing in Java*. Prentice Hall PTR, 1999.

[9] William K. Pratt. *Digital Image Processing*. Wiley-Interscience, 2001. Third Edition.

Index