Introduction à l'imagerie numérique

3I022-fev2018

Compression des images

Licence d'informatique

Février 2018

Introduction

La compression : pourquoi ?

- Réduire la taille de stockage des images (mais pas les dimensions de l'image ! ! !)

$\Rightarrow$ Essentiel en transmission des données

ou en stockage de masse (base de données images)

- Stocker des images de plus en plus grosses (12 millions pixels sur les appareils photo grand public, beaucoup plus pour des images satellites).
- Les capacités de transmission et de stockage grandissantes n'y font rien !

Introduction

La compression : comment ?

- Trois angles d'attaque :
  1. **redondance du codage**,
     les valeurs de l'image sont-elles codées de façon optimale ? en général, non : principe de codage par dictionnaire (compression non destructive).
  2. **redondance interpixel**,
     des images possèdent souvent des motifs ou des structures qui se répètent dans l'espace (et aussi le temps), comment les représenter efficacement ?
  3. **redondance psychovisuelle**,
     la vision humaine est plus sensible à certaines répartitions des couleurs/du contraste/... Comment en tenir compte ?
- La compression nécessite deux algorithmes : un encodeur (C) et un décodeur (D) :

$$f \longrightarrow C(f) \longrightarrow D(C(f))$$

Introduction

Quantifier la compression

- Mesure de la compression (en terme d'occupation mémoire) :

$$C_R = \frac{n_1}{n_2}$$

n_1 = nombre de bits pour coder l'image avant compression
n_2 = nombre de bits pour coder l'image après compression

- Qualité de la compression : on peut la mesurer en calculant l'erreur liée à la compression.

$$e(x,y) = D(C(f))(x,y) - f(x,y)$$

- Puis l'erreur quadratique moyenne :

$$e_{rms} = \left[\frac{1}{MN} \sum_{x,y} (e(x,y))^2 \right]^{\frac{1}{2}}$$

N,M dimensions de l'image.

Introduction

Quantifier la compression

- Une compression sans perte implique : $e_{rms} = 0$ (reconstruction parfaite).
- Si $e_{rms} > 0$ on a une mesure de l'erreur de reconstruction de l'image.
- Que mesure-t-elle ?

Introduction

Quantifier la compression

- Une compression sans perte implique : $e_{rms} = 0$ (reconstruction parfaite).
- Si $e_{rms} > 0$ on a une mesure de l'erreur de reconstruction de l'image.
- Que mesure-t-elle ? écart quadratique moyen entre l'image et sa reconstruction !
- Renseigne-t-elle sur la nature des erreurs ?

Introduction

Quantifier la compression

- Une compression sans perte implique : $e_{rms} = 0$ (reconstruction parfaite).
- Si $e_{rms} > 0$ on a une mesure de l'erreur de reconstruction de l'image.
- Que mesure-t-elle ? écart quadratique moyen entre l'image et sa reconstruction !
- Renseigne-t-elle sur la nature des erreurs ? non !
  - Dégrade-t-elle vraiment l'image et dans quelle proportion ?
  - On peut visualiser e (une image) et vérifier si les erreurs sont localisées spatialement
  - On peut calculer l'histogramme de e et voir comment sont réparties fréquentiellement les erreurs (haute, basse fréquences, ...)

Quantifier la compression

Un exemple

- Soit deux commandes `comp` et `decomp` :

```
comp lena.inr | decomp > lena.cmp
```

- Image des erreurs :

```
so lena.inr lena.cmp > erreur.inr
```

- Erreur quadratique moyenne :

```
car erreur.inr | ical | awk '{print $2}'
```

- Erreur moyenne absolue :

```
car erreur.inr | ra | ical | awk '{print $2}'
```

- Un script qui calcule l'erreur RMS :

```
#! /bin/bash
# calcul l'erreur entre deux images
so $1 $2 | car | ical | awk '{print $2}'
```

Structure générale : encodage/décodage

- ENCODAGE :

$$f \longrightarrow \text{MAPPER} \longrightarrow \text{QUANT.} \longrightarrow \text{COD. SYM.} \longrightarrow C(f)$$

  - MAPPER : élimination de la redondance interpixel. Cette phase est réversible.
  - QUANTIFICATION : élimination de la redondance psychovisuelle. On élimine les structures non perceptibles par la vision humaine, c'est donc une opération irréversible.
  - CODAGE DES SYMBOLES : élimination de la redondance du codage. Étape réversible.

- DECODAGE :

$$C(f) \longrightarrow \text{DEC. SYM.} \longrightarrow \text{QUAN. INV.} \longrightarrow \text{MAP. INV.} \longrightarrow D(C(f))$$

  - DECODAGE DES SYMBOLES : opération inverse du codage des symboles.
  - QUANTIFICATION INVERSE : transformation réciproque de la quantification.
  - MAP. INV. : réciproque de MAPPER

Redondances liées au codage

Taille du codage

- Contexte : une image ayant des pixels de certaines valeurs sur-représentées par rapport à d'autres.
- Notion de taille de codage moyen :
  - Soit une image de taille $N \times M$ codée sur L n.d.g.,
  - Soit p son histogramme normalisé : $p(r_i) = \frac{n_i}{N \times M}$ et n_i le nombre de pixels de valeur r_i.
  - Soit $l(r_i)$ la taille (en nombre de bits) pour coder la valeur r_i.
  - La taille de codage moyen (en bits par pixel) pour l'image est :

$$L_{avg} = \sum_i l(r_i)p(r_i)$$

  - La taille moyenne (en bits) pour l'image est $L_{avg} \times N \times M$
  - Évidemment, le codage informatique standard, attribue un nombre de bit constant, l, pour coder chaque valeur, ainsi on a toujours $L_{avg} = l$.

Taille du codage

- Voyons maintenant, à travers un exemple simple, comment on peut choisir un codage qui minimise la taille de codage moyen.
- Soit une image à 4 niveaux de gris, r_1, r_2, r_3, r_4 ayant respectivement les fréquences d'apparition suivantes : $0.1875(3/16), 0.5(1/2), 0.125(1/8), 0.1875$.
- Le tableau suivant calcule la taille de codage moyen pour deux codages différents :

r	$p(r)$	Code #1	l	Code #2	l
r_1	.1875	00	2	011	3
r_2	.5	01	2	1	1
r_3	.125	10	2	010	3
r_4	.1875	11	2	00	2

Taille du codage

- Code #1 : il correspond au codage informatique standard sur 2 bits.
- Code #2 : choisi de façon à minimiser L_{avg} :
  - on code sur 1 bit la valeur la plus fréquente (r_2),
  - on code sur 2 bits la seconde valeur plus fréquente (r_4),
  - on code les deux dernières valeurs sur 3 bits. Il faut bien 3 bits pour lever l'ambiguïté de lecture (décodage).
- Voir que le codage #2 est le meilleur !

Taille du codage

- Code #1 : il correspond au codage informatique standard sur 2 bits.
- Code #2 : choisi de façon à minimiser L_{avg} :
  - on code sur 1 bit la valeur la plus fréquente (r_2),
  - on code sur 2 bits la seconde valeur plus fréquente (r_4),
  - on code les deux dernières valeurs sur 3 bits. Il faut bien 3 bits pour lever l'ambiguïté de lecture (décodage).
- Voir que le codage #2 est le meilleur !

$$L_{\text{avg}} = 3 \times 0.1875 + 1 \times 0.5 + 3 \times 0.125 + 2 \times 0.1875 = 1.8125$$

Il n'est pas meilleur de prendre 011,1,00,010 :

$$L_{\text{avg}} = 1.875$$

- Le taux de compression est donc :

$$C_r = \frac{2}{1.8125} \simeq 1.103$$

Taille du codage

- Le codage #1 est sur 2 bits, le codage #3 varie entre 1 et 3 bits mais comme il y a beaucoup de valeurs codées sur 1 bit la taille de l'image sera plus petite.
- La conclusion :

> Dès lors que les histogrammes ne sont pas équidistribués (c'est toujours le cas des images "naturelles") on peut toujours trouver un codage qui autorise une compression significative de l'image.

Taille optimale de codage : l'entropie

- Résultat (admis) : la taille de codage est bornée, et on connaît la borne inférieure.

Definition (Entropie)

Soit p l'histogramme normalisé d'une image (c'est-à-dire sa loi de probabilité). L'entropie de p est définie par :

$$H(p) = -\sum_{l=1}^{L} p(l) \log_2(p(l))$$

L'entropie mesure (en bits) la quantité d'information moyenne contenu dans un pixel.

- Définition générale, qui s'applique à tout signaux, c'est-à-dire tout ensemble de valeurs ou de symboles.
- Taille optimale pour coder un symbole s : $-\log_2(p(s))$

Calcul de l'entropie

- Cette quantité est facile à calculer. Exemple avec des commandes Inrimage :

```
par -o -f cameraman.inr && echo
-o 1 -f
his -hn cameraman.inr > hn.inr
```

- Inrimage possède une commande qui calcule le logarithme népérien. Pour calculer en base 2, il faut diviser par $\ln(2) \simeq 0.693147$.
- $x \mapsto x \ln(x)$ se prolonge par continuité en 0 mais ajout d'un petit biais pour éviter le calcul de $\ln(0)$

```
bi hn.inr -n 0.000001 | lo |  mu hn.inr | sc -n -1.442695 > ent.inr
ical ent.inr
         0.000000      0.027036      0.209212
echo  '0.027036*255' | bc -l
6.894180
```

Calcul de l'entropie

- Autre façon de calculer l'entropie (plus précise) : on exclut les valeurs nulles :

```
#! /bin/sh
# calcul l'entropie d'images -o 1
for a in "$@" ; do
    his -hn $a |tpr -c -l 1  | \
        awk '
{ if ($1 != 0)
    somme += -$1*log($1)
}
END{print somme/log(2)}'
done
```

Calcul de l'entropie

Exemple

- Reprenons l'exemple de l'histogramme à 4 valeurs.
- L'entropie vaut :

$$\begin{aligned} H &= -2 \times \log_2(0.1875) \times 0.1875 - \log_2(0.5) \times 0.5 \\ &\quad - \log_2(0.125) \times 0.125 \\ &= 1.7806 \end{aligned}$$

- Remarquons que le codage # 2 avait une taille moyenne de 1.875 bits par pixel, donc proche de la limite théorique pour coder cette image.

Codes de Huffman (1952)

- Méthode pour trouver un dictionnaire de symboles (= un codage) qui soit optimal dans le sens vu précédemment (1952 - David Albert Huffman).
- Algorithme (choix des codes/symboles) :
  1. Calculer la table des fréquences d'apparition de chaque symbole. et ordonner la table par fréquences croissantes.
  2. Chaque élément de la table constitue les nœuds terminaux d'un arbre binaire.
     Un nœud contient un symbole et son poids (fréquence d'apparition du symbole).
     Cet arbre est construit ainsi :
     a) les deux nœuds de poids les plus faibles constituent les fils d'un nouveau nœud dont le poids est la somme du poids des fils.
     b) on réitère ce procédé en considérant les nœuds restants et celui nouvellement créé et en les réordonnant.
  3. Lorsque tous les nœuds ont été traités, un unique arbre binaire et connexe est constitué.

Codes de Huffman

Exemple

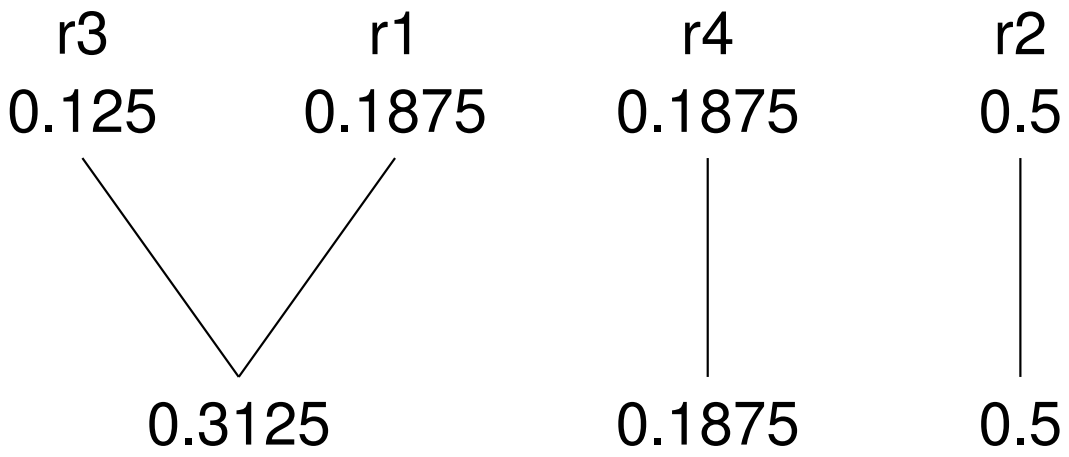
- Illustrons cette première étape sur notre exemple à 4 symboles :

r3	r1	r4	r2
0.125	0.1875	0.1875	0.5

Codes de Huffman

Exemple

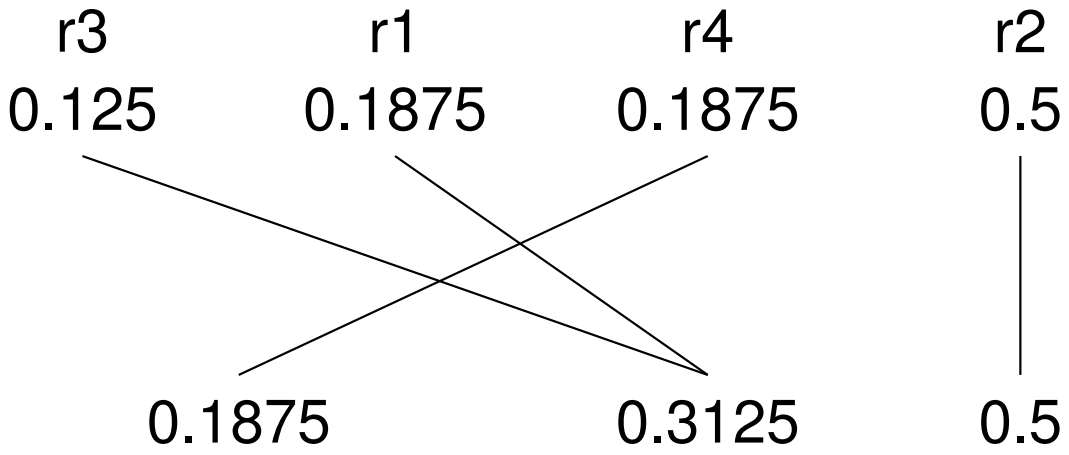
- Illustrons cette première étape sur notre exemple à 4 symboles :



Codes de Huffman

Exemple

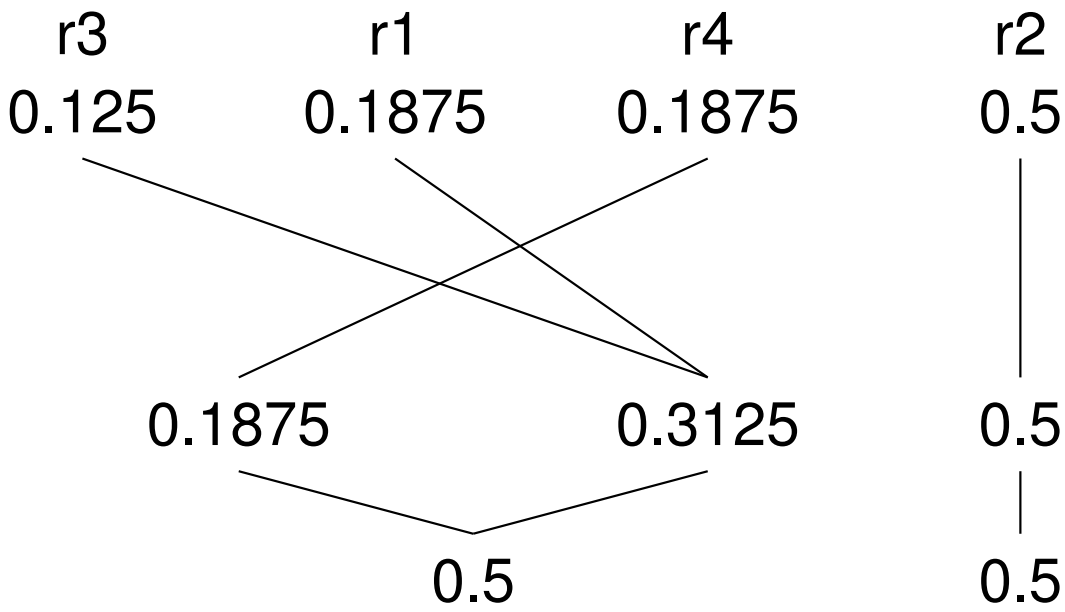
- Illustrons cette première étape sur notre exemple à 4 symboles :



Codes de Huffman

Exemple

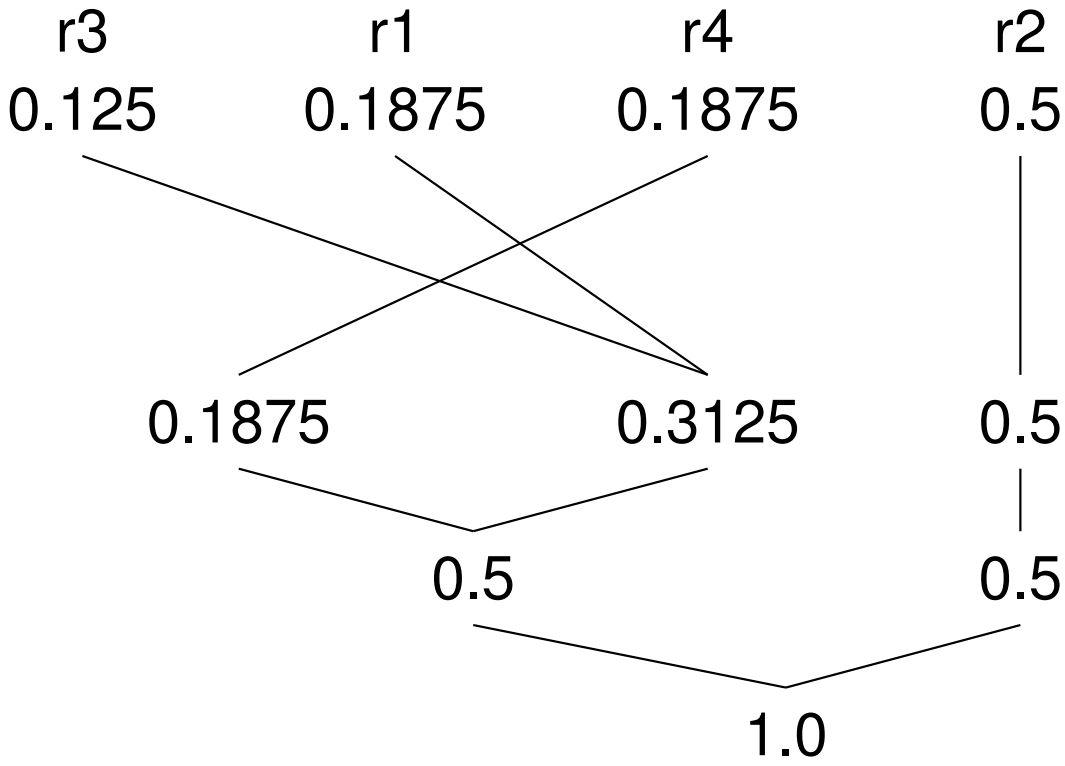
- Illustrons cette première étape sur notre exemple à 4 symboles :



Codes de Huffman

Exemple

- Illustrons cette première étape sur notre exemple à 4 symboles :



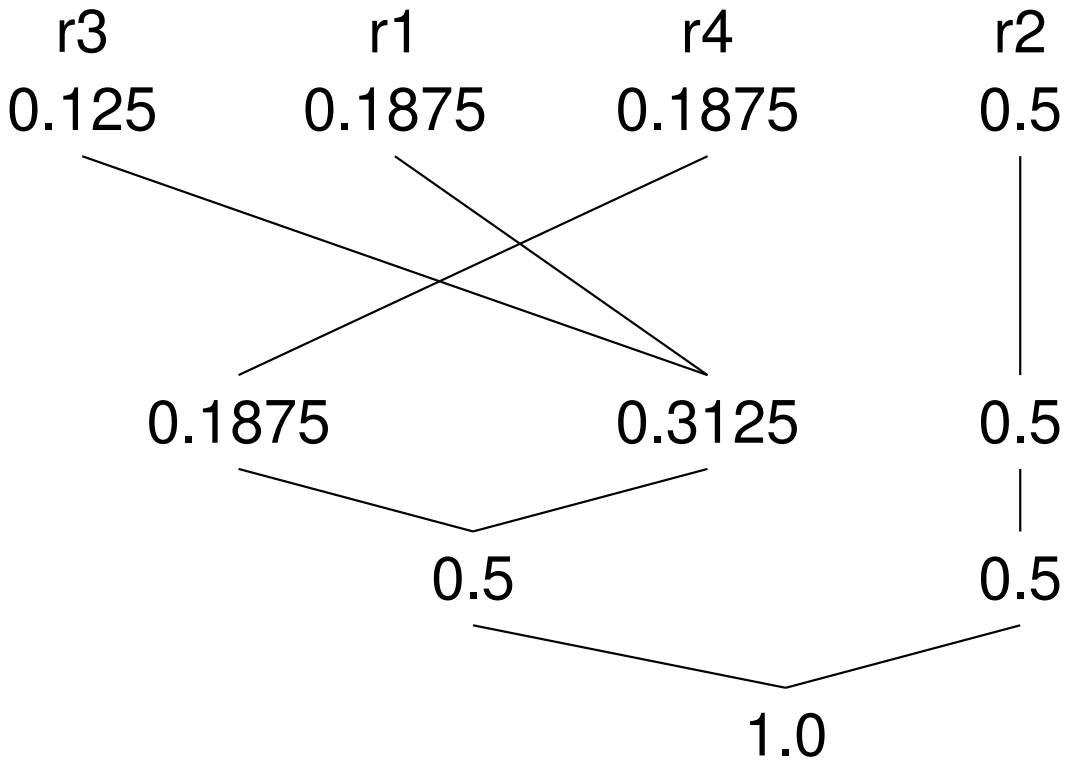
Codes de Huffman

Algorithme

- La dernière étape consiste à choisir les nouveaux symboles : ils sont obtenus en parcourant l'arbre.
  4. Par convention on nomme '0' les fils à gauche et '1' les fils à droite.
  5. Pour chaque valeur on détermine le code :
     - on parcourt l'arbre depuis la racine jusqu'à la valeur ;
     - on note la série de bits qui forme le chemin. Cette série donne le code.
- L'étape de ré-ordonnement des nœuds est cruciale car elle permettra de faire le décodage sans se tromper !

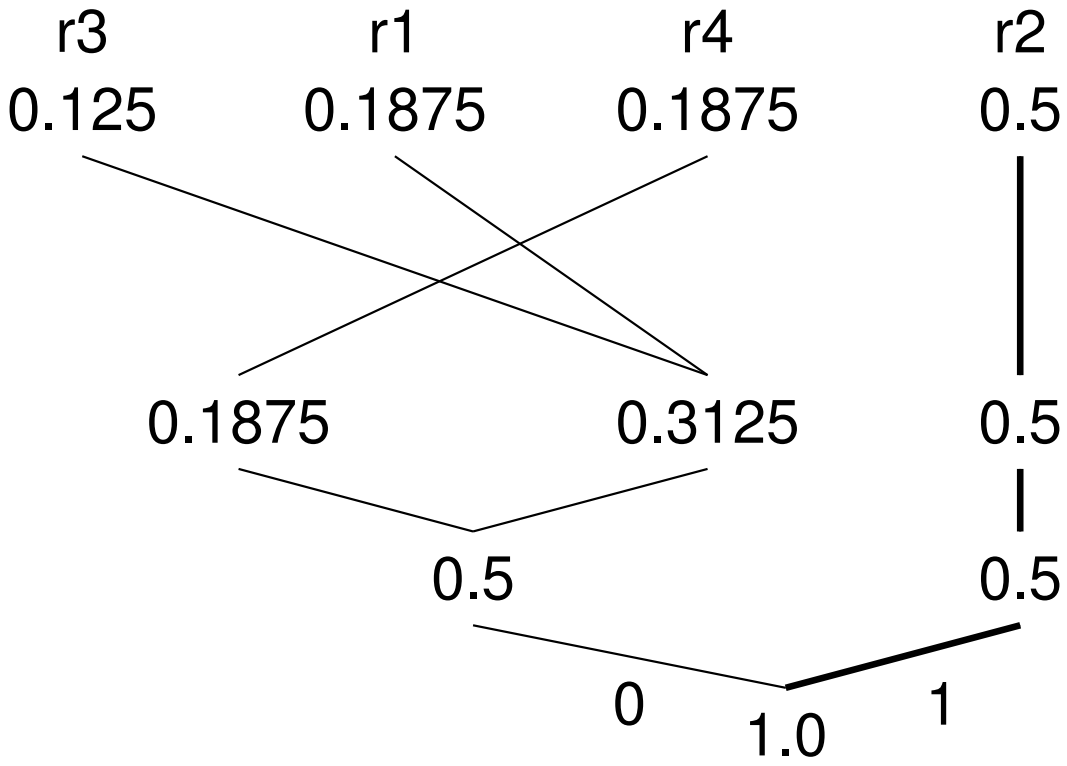
Codes de Huffman

Exemple



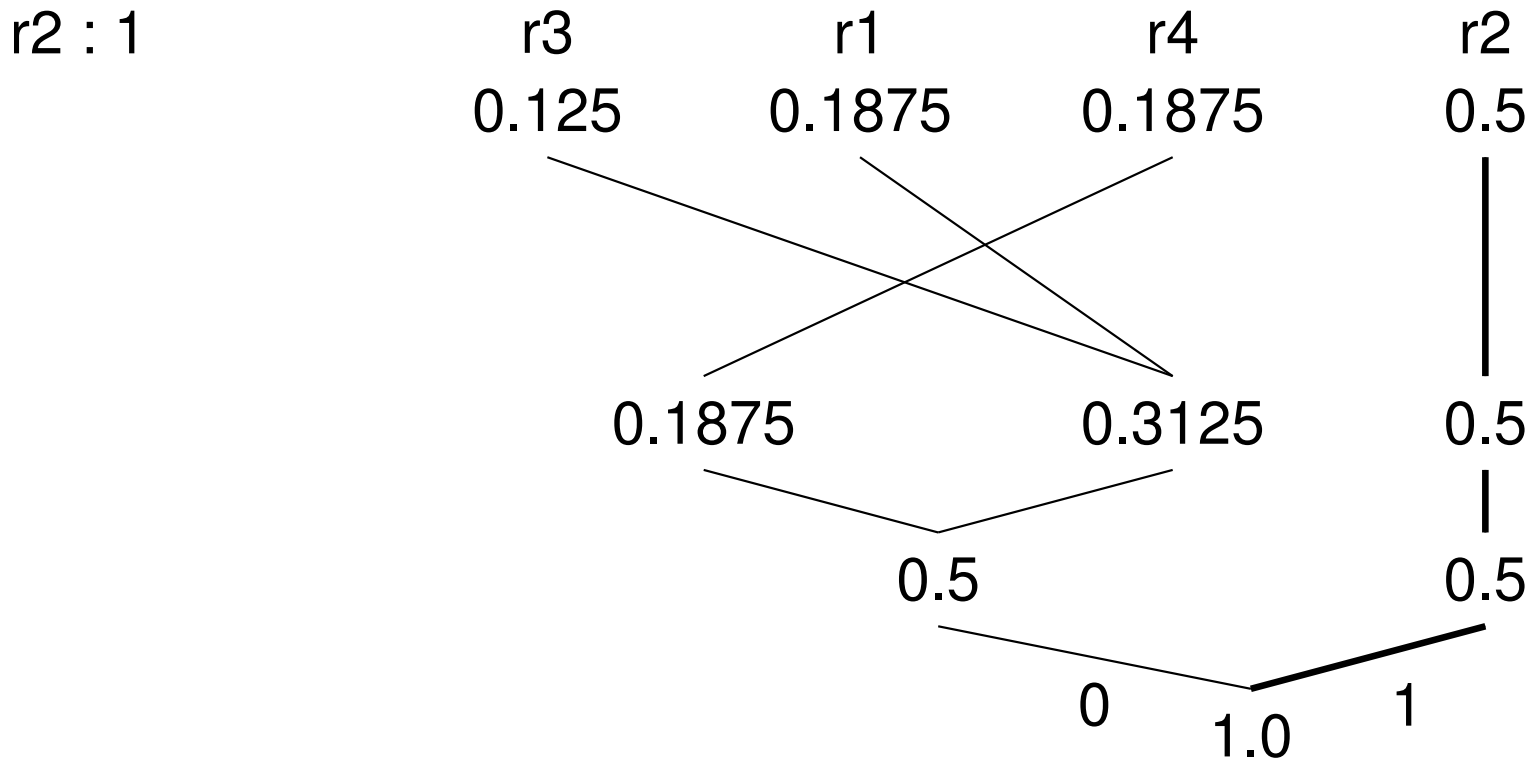
Codes de Huffman

Exemple



Codes de Huffman

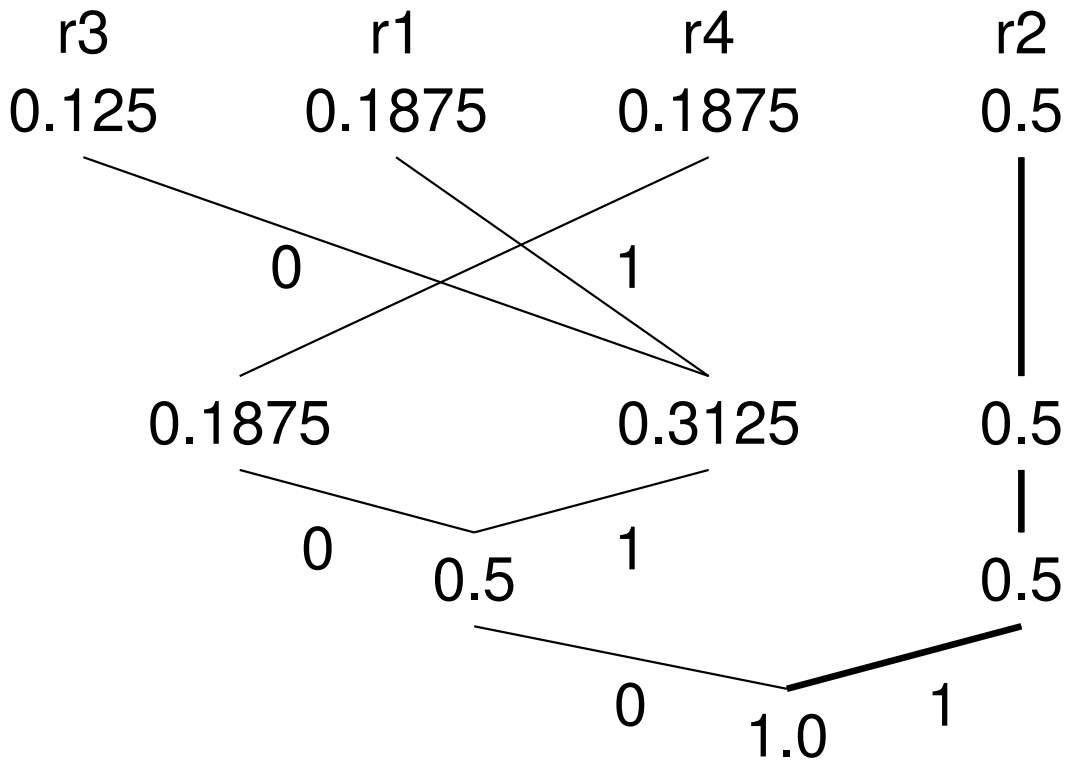
Exemple



Codes de Huffman

Exemple

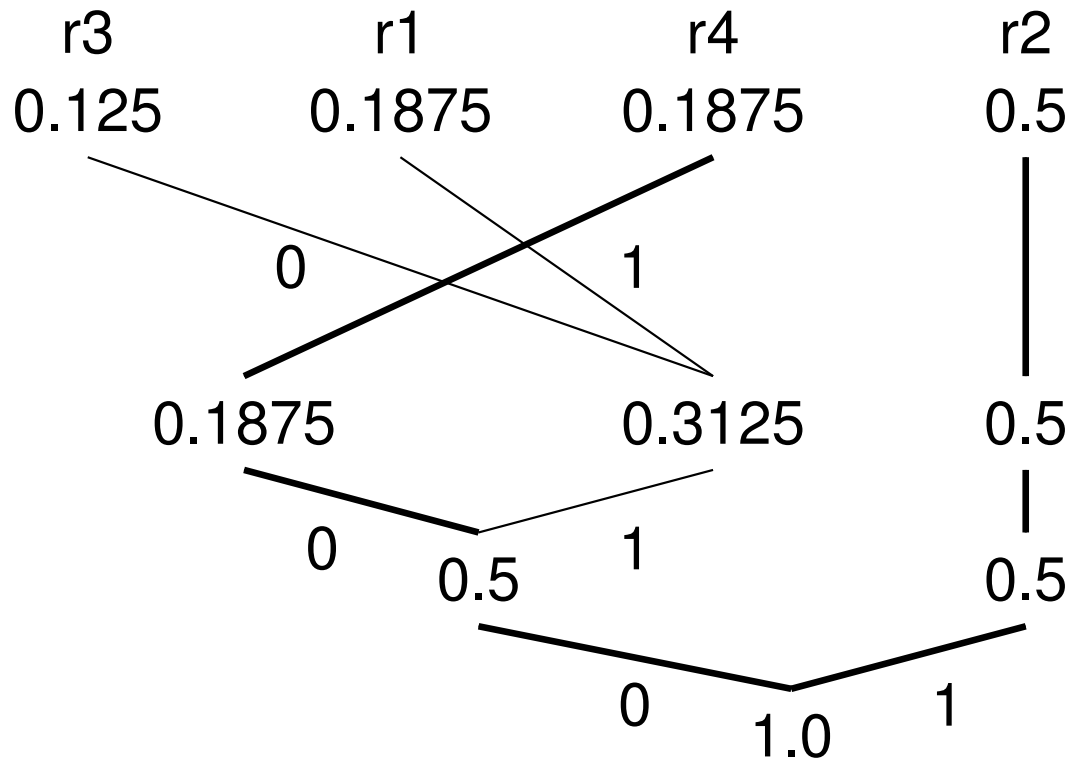
r2 : 1



Codes de Huffman

Exemple

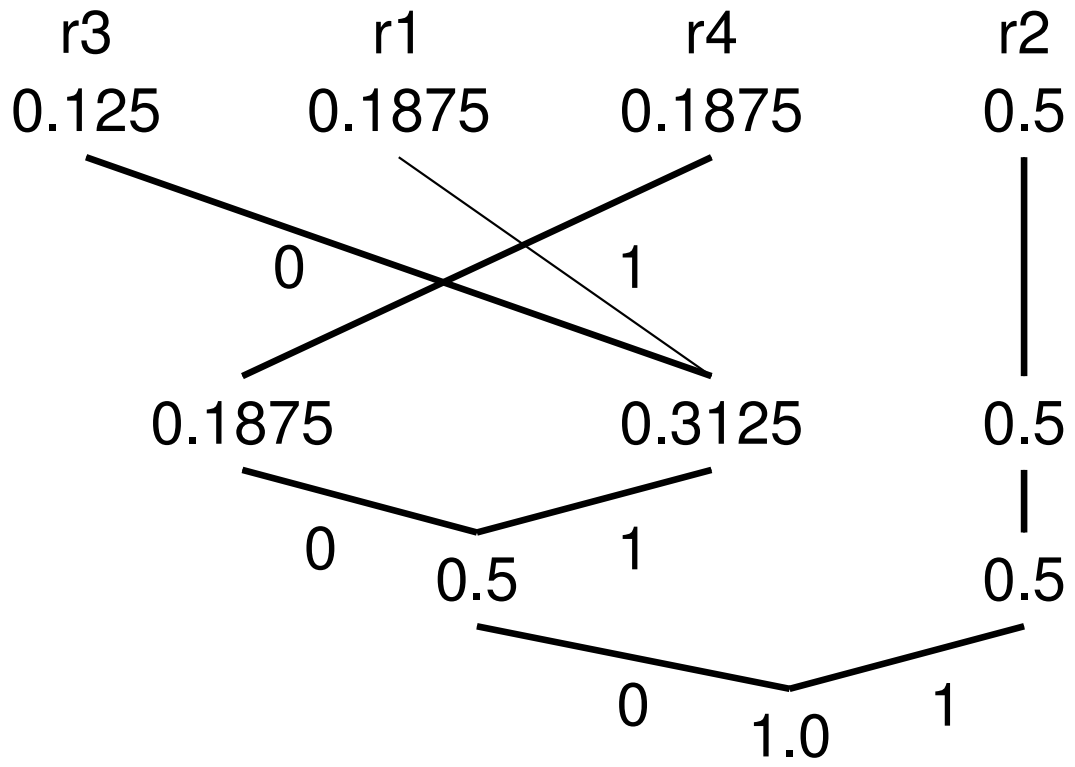
r2 : 1
r4 : 00



Codes de Huffman

Exemple

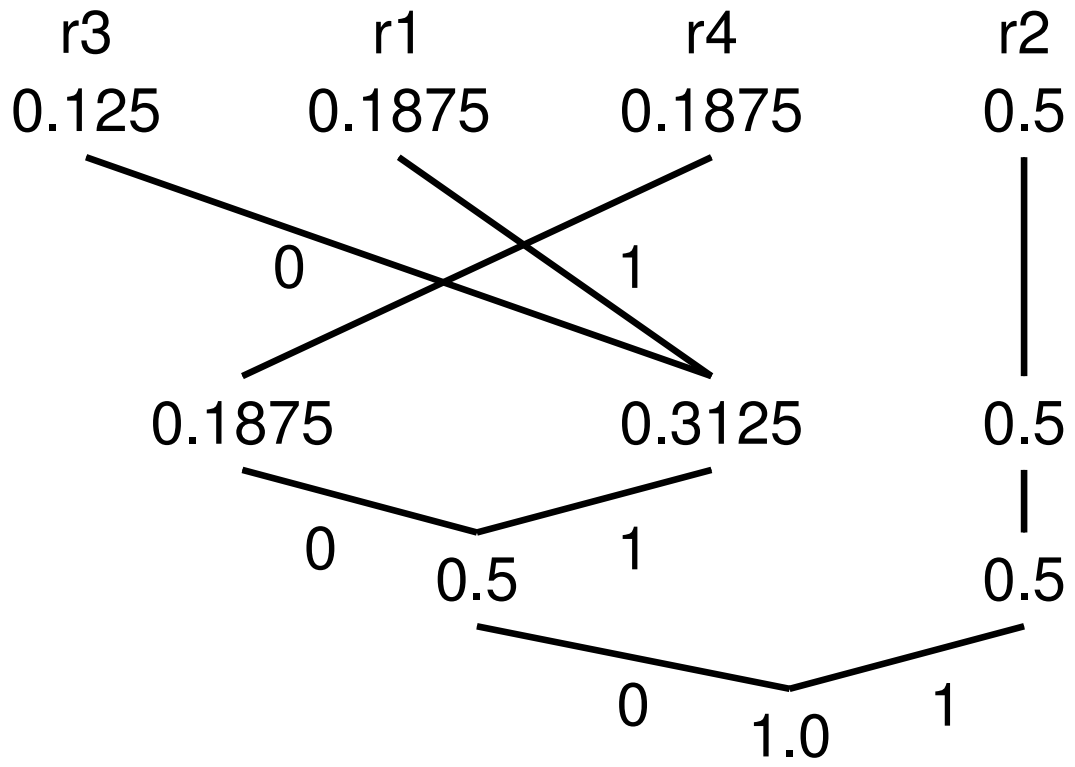
r2 : 1
r4 : 00
r3 : 010



Codes de Huffman

Exemple

r2 : 1
r4 : 00
r3 : 010
r1 : 011



Codes de Huffman

Encodage de l'image

- Une fois la table des codes obtenue, on procède à l'encodage des images : chaque valeur est remplacée par son code dans la table de Huffman.
- Exemple avec l'image 4×4 :

119	123	168	119
123	119	168	168
119	119	107	119
107	107	119	119

r	p(r)	Code
107	0.1875	011
119	0.5	1
123	0.125	010
168	0.1875	00

- En supposant une organisation par ligne, l'image encodée est :

$$101000101010000110111011011111$$

soit 30 bits, au lieu de $4 \times 4 \times 2 = 32$ bits pour un codage constant sur 2 bits.

Décodage de l'image

- La table de Huffman doit être connue pour décoder.
- Dans certaines situations, on peut déterminer à l'avance la table. Exemples :
  - encoder du texte dans une langue donnée, où les fréquences de chaque lettre sont statistiquement connues.
  - encoder des images prises dans des conditions similaires d'acquisition.
- Le décodage est trivial, on lit bit par bit :
  - d'abord un premier bit, on cherche dans la table la correspondance,
  - si la correspondance n'est pas trouvée on lit un nouveau bit, et on cherche la séquence de 2 bits dans la table, *etc*.
- Dans les autres cas, il faut embarquer, avec l'image encodée, la table de Huffman ou bien donner l'histogramme de l'image source.
- Les formats JPEG, GIF, PNG utilisent un codage de Huffman.
- Les commandes `zip`, `gzip`, `lzh`, `lha` également.

Codage arithmétique

- Le codage de Huffman n'est pas optimal : il faut un nombre **entier** pour coder un symbole (là où il faudrait un nombre non entier).
- Exemple : si $p('128') = 0.3$ alors la taille optimale est : $-\log_2(0.9) = 0.15$ soit 1 bit, pour encoder cette valeur là où on pourrait en mettre 6 !
- Codage arithémique : il est optimal (si le nombre de symbole à encoder tends vers l'infini).
- Il est plus complexe à mettre en œuvre ...

Codage arithmétique

principe

- La suite de symboles est lue dans sa totalité et est représentée par un réel dans l'intervalle $[0, 1]$
- Un nombre réel a un développement infini et peut donc représenter une suite quelconque de symboles
- Un choix judicieux des chiffres du réel permet une représentation **optimale** de la suite des symboles
- Première étape :
  1. On calcule l'histogramme normalisé (on a n symboles)
  2. On divise l'intervalle [0,1] en n sous intervalles disjoints, tel que la longueur de chaque intervalle correspond à la fréquence d'apparition d'un symbole :

$$[0, 1[= \bigcup_{i=1}^{n} [a_i, a_i + p(s_i)[$$

avec $p(s_i)$ probabilité d'apparition du symbole s_i

Codage arithmétique

Principe de l'encodage

- On examine un premier symbole, on construit l'intervalle I_1 qui contient sa fréquence
- On examine un second symbole, on construit l'intervalle $I_2 \subset I_1$ qui contient sa fréquence
- On réitère ce procédé pour tous les symboles à encoder, la borne inférieure donne le résultat
- On procède donc par composition d'intervalles. Exemples :
  - Soit un symbole *a* dans l'intervalle [0.3, 0.5[(de longueur 0.2)
  - La suite *aa* sera donc dans l'intervalle :
    $[0.3 + 0.2 \times 0.3, 0.3 + 0.2 \times 0.5[= [0.36, 0.40[$
  - Soit un symbole *b* dans l'intervalle [0.6, 0.7[
  - Alors la suite *ab* sera dans l'intervalle
    $[0.3 + 0.2 \times 0.6, 0.3 + 0.2 \times 0.7[= [0.42, 0.44[$

Codage arithmétique

Principe du décodage

- Un réel est la séquence à décoder
- On regarde dans quel intervalle il appartient : ceci fourni le premier symbole
- On retranche au réel la borne inférieure de l'intervalle, on le divise par la longueur de l'intervalle. On réitère le procédé.
- Exemples :
  - $0.42 \in [0.3, 0.5[\Rightarrow a$
  - $(0.42 - 0.3)/0.2 = 0.6$
  - $0.6 \in [0.6, 0.7[\Rightarrow b$

  - $0.36 \in [0.3, 0.5[\Rightarrow a$
  - $(0.36 - 0.3)/0.2 = 0.3$
  - $0.3 \in [0.3, 0.5[\Rightarrow a$

Codage arithmétique

Exemple : encodage

- Soit l'alphabet à 9 symboles :

s	$p(s)$	inf.	sup.
a	1/10	0	0.1
c	1/10	0.1	0.2
d	1/10	0.2	0.3
e	2/10	0.3	0.5
g	1/10	0.5	0.6
l	1/10	0.6	0.7
n	1/10	0.7	0.8
o	1/10	0.8	0.9
'	1/10	0.9	1

- Encodage de la suite : `l'encodage`

symbole	borne inf.	borne sup.
l	0,6	0,7

Codage arithmétique

Exemple : encodage

- Soit l'alphabet à 9 symboles :

s	$p(s)$	inf.	sup.
a	1/10	0	0.1
c	1/10	0.1	0.2
d	1/10	0.2	0.3
e	2/10	0.3	0.5
g	1/10	0.5	0.6
l	1/10	0.6	0.7
n	1/10	0.7	0.8
o	1/10	0.8	0.9
'	1/10	0.9	1

- Encodage de la suite : `l'encodage`

symbole	borne inf.	borne sup.
l	0,6	0,7
'	0,69	0,70

Codage arithmétique

Exemple : encodage

- Soit l'alphabet à 9 symboles :

s	$p(s)$	inf.	sup.
a	1/10	0	0.1
c	1/10	0.1	0.2
d	1/10	0.2	0.3
e	2/10	0.3	0.5
g	1/10	0.5	0.6
l	1/10	0.6	0.7
n	1/10	0.7	0.8
o	1/10	0.8	0.9
'	1/10	0.9	1

- Encodage de la suite : `l'encodage`

symbole	borne inf.	borne sup.
l	0,6	0,7
'	0,69	0,70
e	0,693	0,695

Codage arithmétique

Exemple : encodage

- Soit l'alphabet à 9 symboles :

s	$p(s)$	inf.	sup.
a	1/10	0	0.1
c	1/10	0.1	0.2
d	1/10	0.2	0.3
e	2/10	0.3	0.5
g	1/10	0.5	0.6
l	1/10	0.6	0.7
n	1/10	0.7	0.8
o	1/10	0.8	0.9
'	1/10	0.9	1

- Encodage de la suite : `l'encodage`

symbole	borne inf.	borne sup.
l	0,6	0,7
'	0,69	0,70
e	0,693	0,695
n	0,6944	0,6946

Codage arithmétique

Exemple : encodage

- Soit l'alphabet à 9 symboles :

s	$p(s)$	inf.	sup.
a	1/10	0	0.1
c	1/10	0.1	0.2
d	1/10	0.2	0.3
e	2/10	0.3	0.5
g	1/10	0.5	0.6
l	1/10	0.6	0.7
n	1/10	0.7	0.8
o	1/10	0.8	0.9
'	1/10	0.9	1

- Encodage de la suite : `l'encodage`

symbole	borne inf.	borne sup.
l	0,6	0,7
'	0,69	0,70
e	0,693	0,695
n	0,6944	0,6946
c	0,69442	0,69444

Codage arithmétique

Exemple : encodage

- Soit l'alphabet à 9 symboles :

s	$p(s)$	inf.	sup.
a	1/10	0	0.1
c	1/10	0.1	0.2
d	1/10	0.2	0.3
e	2/10	0.3	0.5
g	1/10	0.5	0.6
l	1/10	0.6	0.7
n	1/10	0.7	0.8
o	1/10	0.8	0.9
'	1/10	0.9	1

- Encodage de la suite : `l'encodage`

symbole	borne inf.	borne sup.
l	0,6	0,7
'	0,69	0,70
e	0,693	0,695
n	0,6944	0,6946
c	0,69442	0,69444
o	0,694436	0,694438

Codage arithmétique

Exemple : encodage

- Soit l'alphabet à 9 symboles :

s	$p(s)$	inf.	sup.
a	1/10	0	0.1
c	1/10	0.1	0.2
d	1/10	0.2	0.3
e	2/10	0.3	0.5
g	1/10	0.5	0.6
l	1/10	0.6	0.7
n	1/10	0.7	0.8
o	1/10	0.8	0.9
'	1/10	0.9	1

- Encodage de la suite : `l'encodage`

symbole	borne inf.	borne sup.
l	0,6	0,7
'	0,69	0,70
e	0,693	0,695
n	0,6944	0,6946
c	0,69442	0,69444
o	0,694436	0,694438
d	0,6944364	0,6944366

Codage arithmétique

Exemple : encodage

- Soit l'alphabet à 9 symboles :

s	$p(s)$	inf.	sup.
a	1/10	0	0.1
c	1/10	0.1	0.2
d	1/10	0.2	0.3
e	2/10	0.3	0.5
g	1/10	0.5	0.6
l	1/10	0.6	0.7
n	1/10	0.7	0.8
o	1/10	0.8	0.9
'	1/10	0.9	1

- Encodage de la suite : `l'encodage`

symbole	borne inf.	borne sup.
l	0,6	0,7
'	0,69	0,70
e	0,693	0,695
n	0,6944	0,6946
c	0,69442	0,69444
o	0,694436	0,694438
d	0,6944364	0,6944366
a	0,6944364	0,69443642

Codage arithmétique

Exemple : encodage

- Soit l'alphabet à 9 symboles :

s	$p(s)$	inf.	sup.
a	1/10	0	0.1
c	1/10	0.1	0.2
d	1/10	0.2	0.3
e	2/10	0.3	0.5
g	1/10	0.5	0.6
l	1/10	0.6	0.7
n	1/10	0.7	0.8
o	1/10	0.8	0.9
'	1/10	0.9	1

- Encodage de la suite : `l'encodage`

symbole	borne inf.	borne sup.
l	0,6	0,7
'	0,69	0,70
e	0,693	0,695
n	0,6944	0,6946
c	0,69442	0,69444
o	0,694436	0,694438
d	0,6944364	0,6944366
a	0,6944364	0,69443642
g	0,69443641	0,694436412

Codage arithmétique

Exemple : encodage

- Soit l'alphabet à 9 symboles :

s	$p(s)$	inf.	sup.
a	1/10	0	0.1
c	1/10	0.1	0.2
d	1/10	0.2	0.3
e	2/10	0.3	0.5
g	1/10	0.5	0.6
l	1/10	0.6	0.7
n	1/10	0.7	0.8
o	1/10	0.8	0.9
'	1/10	0.9	1

- Encodage de la suite : `l'encodage`

symbole	borne inf.	borne sup.
l	0,6	0,7
'	0,69	0,70
e	0,693	0,695
n	0,6944	0,6946
c	0,69442	0,69444
o	0,694436	0,694438
d	0,6944364	0,6944366
a	0,6944364	0,69443642
g	0,69443641	0,694436412
e	0,6944364106	0,694436411

Codage arithmétique

Exemple : encodage

- Soit l'alphabet à 9 symboles :

s	$p(s)$	inf.	sup.
a	1/10	0	0.1
c	1/10	0.1	0.2
d	1/10	0.2	0.3
e	2/10	0.3	0.5
g	1/10	0.5	0.6
l	1/10	0.6	0.7
n	1/10	0.7	0.8
o	1/10	0.8	0.9
'	1/10	0.9	1

- Décodage de la suite : 0,6944364106

Nombre codé	intervalle	symbole
0,6944364106	$\in [0.6, 0.7[$	l

Codage arithmétique

Exemple : encodage

- Soit l'alphabet à 9 symboles :

s	$p(s)$	inf.	sup.
a	1/10	0	0.1
c	1/10	0.1	0.2
d	1/10	0.2	0.3
e	2/10	0.3	0.5
g	1/10	0.5	0.6
l	1/10	0.6	0.7
n	1/10	0.7	0.8
o	1/10	0.8	0.9
'	1/10	0.9	1

- Décodage de la suite : 0,6944364106

Nombre codé	intervalle	symbole
0,6944364106	$\in [0.6, 0.7[$	l
0,944364106	$\in [0.9, 1[$	'

Codage arithmétique

Exemple : encodage

- Soit l'alphabet à 9 symboles :

s	$p(s)$	inf.	sup.
a	1/10	0	0.1
c	1/10	0.1	0.2
d	1/10	0.2	0.3
e	2/10	0.3	0.5
g	1/10	0.5	0.6
l	1/10	0.6	0.7
n	1/10	0.7	0.8
o	1/10	0.8	0.9
'	1/10	0.9	1

- Décodage de la suite : 0,6944364106

Nombre codé	intervalle	symbole
0,6944364106	$\in [0.6, 0.7[$	l
0,944364106	$\in [0.9, 1[$	'
0,44364106	$\in [0.3, 0.5[$	e

Codage arithmétique

Exemple : encodage

- Soit l'alphabet à 9 symboles :

s	$p(s)$	inf.	sup.
a	1/10	0	0.1
c	1/10	0.1	0.2
d	1/10	0.2	0.3
e	2/10	0.3	0.5
g	1/10	0.5	0.6
l	1/10	0.6	0.7
n	1/10	0.7	0.8
o	1/10	0.8	0.9
'	1/10	0.9	1

- Décodage de la suite : 0,6944364106

Nombre codé	intervalle	symbole
0,6944364106	$\in [0.6, 0.7[$	l
0,944364106	$\in [0.9, 1[$	'
0,44364106	$\in [0.3, 0.5[$	e
0,7182053	$\in [0.7, 0.8[$	n

Codage arithmétique

Exemple : encodage

- Soit l'alphabet à 9 symboles :

s	$p(s)$	inf.	sup.
a	1/10	0	0.1
c	1/10	0.1	0.2
d	1/10	0.2	0.3
e	2/10	0.3	0.5
g	1/10	0.5	0.6
l	1/10	0.6	0.7
n	1/10	0.7	0.8
o	1/10	0.8	0.9
'	1/10	0.9	1

- Décodage de la suite : 0,6944364106

Nombre codé	intervalle	symbole
0,6944364106	$\in [0.6, 0.7[$	l
0,944364106	$\in [0.9, 1[$	'
0,44364106	$\in [0.3, 0.5[$	e
0,7182053	$\in [0.7, 0.8[$	n
0,182053	$\in [0.1, 0.2[$	c

Codage arithmétique

Exemple : encodage

- Soit l'alphabet à 9 symboles :

s	$p(s)$	inf.	sup.
a	1/10	0	0.1
c	1/10	0.1	0.2
d	1/10	0.2	0.3
e	2/10	0.3	0.5
g	1/10	0.5	0.6
l	1/10	0.6	0.7
n	1/10	0.7	0.8
o	1/10	0.8	0.9
'	1/10	0.9	1

- Décodage de la suite : 0,6944364106

Nombre codé	intervalle	symbole
0,6944364106	$\in [0.6, 0.7[$	l
0,944364106	$\in [0.9, 1[$	'
0,44364106	$\in [0.3, 0.5[$	e
0,7182053	$\in [0.7, 0.8[$	n
0,182053	$\in [0.1, 0.2[$	c
0,82053	$\in [0.8, 0.9[$	o

Codage arithmétique

Exemple : encodage

- Soit l'alphabet à 9 symboles :

s	$p(s)$	inf.	sup.
a	1/10	0	0.1
c	1/10	0.1	0.2
d	1/10	0.2	0.3
e	2/10	0.3	0.5
g	1/10	0.5	0.6
l	1/10	0.6	0.7
n	1/10	0.7	0.8
o	1/10	0.8	0.9
'	1/10	0.9	1

- Décodage de la suite : 0,6944364106

Nombre codé	intervalle	symbole
0,6944364106	$\in [0.6, 0.7[$	l
0,944364106	$\in [0.9, 1[$	'
0,44364106	$\in [0.3, 0.5[$	e
0,7182053	$\in [0.7, 0.8[$	n
0,182053	$\in [0.1, 0.2[$	c
0,82053	$\in [0.8, 0.9[$	o
0,2053	$\in [0.2, 0.3[$	d

Codage arithmétique

Exemple : encodage

- Soit l'alphabet à 9 symboles :

s	$p(s)$	inf.	sup.
a	1/10	0	0.1
c	1/10	0.1	0.2
d	1/10	0.2	0.3
e	2/10	0.3	0.5
g	1/10	0.5	0.6
l	1/10	0.6	0.7
n	1/10	0.7	0.8
o	1/10	0.8	0.9
'	1/10	0.9	1

- Décodage de la suite : 0,6944364106

Nombre codé	intervalle	symbole
0,6944364106	$\in [0.6, 0.7[$	l
0,944364106	$\in [0.9, 1[$	'
0,44364106	$\in [0.3, 0.5[$	e
0,7182053	$\in [0.7, 0.8[$	n
0,182053	$\in [0.1, 0.2[$	c
0,82053	$\in [0.8, 0.9[$	o
0,2053	$\in [0.2, 0.3[$	d
0,053	$\in [0, 0.1[$	a

Codage arithmétique

Exemple : encodage

- Soit l'alphabet à 9 symboles :

s	$p(s)$	inf.	sup.
a	1/10	0	0.1
c	1/10	0.1	0.2
d	1/10	0.2	0.3
e	2/10	0.3	0.5
g	1/10	0.5	0.6
l	1/10	0.6	0.7
n	1/10	0.7	0.8
o	1/10	0.8	0.9
'	1/10	0.9	1

- Décodage de la suite : 0,6944364106

Nombre codé	intervalle	symbole
0,6944364106	$\in [0.6, 0.7[$	l
0,944364106	$\in [0.9, 1[$	'
0,44364106	$\in [0.3, 0.5[$	e
0,7182053	$\in [0.7, 0.8[$	n
0,182053	$\in [0.1, 0.2[$	c
0,82053	$\in [0.8, 0.9[$	o
0,2053	$\in [0.2, 0.3[$	d
0,053	$\in [0, 0.1[$	a
0,53	$\in [0.5, 0.6[$	g

Codage arithmétique

Exemple : encodage

- Soit l'alphabet à 9 symboles :

s	$p(s)$	inf.	sup.
a	1/10	0	0.1
c	1/10	0.1	0.2
d	1/10	0.2	0.3
e	2/10	0.3	0.5
g	1/10	0.5	0.6
l	1/10	0.6	0.7
n	1/10	0.7	0.8
o	1/10	0.8	0.9
'	1/10	0.9	1

- Décodage de la suite : 0,6944364106

Nombre codé	intervalle	symbole
0,6944364106	$\in [0.6, 0.7[$	l
0,944364106	$\in [0.9, 1[$	'
0,44364106	$\in [0.3, 0.5[$	e
0,7182053	$\in [0.7, 0.8[$	n
0,182053	$\in [0.1, 0.2[$	c
0,82053	$\in [0.8, 0.9[$	o
0,2053	$\in [0.2, 0.3[$	d
0,053	$\in [0, 0.1[$	a
0,53	$\in [0.5, 0.6[$	g
0,3	$\in [0.3, 0.5[$	e

Codage arithmétique

Exemple : encodage

- Soit l'alphabet à 9 symboles :

s	$p(s)$	inf.	sup.
a	1/10	0	0.1
c	1/10	0.1	0.2
d	1/10	0.2	0.3
e	2/10	0.3	0.5
g	1/10	0.5	0.6
l	1/10	0.6	0.7
n	1/10	0.7	0.8
o	1/10	0.8	0.9
'	1/10	0.9	1

- Décodage de la suite : 0,6944364106

Nombre codé	intervalle	symbole
0,6944364106	$\in [0.6, 0.7[$	l
0,944364106	$\in [0.9, 1[$	'
0,44364106	$\in [0.3, 0.5[$	e
0,7182053	$\in [0.7, 0.8[$	n
0,182053	$\in [0.1, 0.2[$	c
0,82053	$\in [0.8, 0.9[$	o
0,2053	$\in [0.2, 0.3[$	d
0,053	$\in [0, 0.1[$	a
0,53	$\in [0.5, 0.6[$	g
0,3	$\in [0.3, 0.5[$	e
0	Fin	

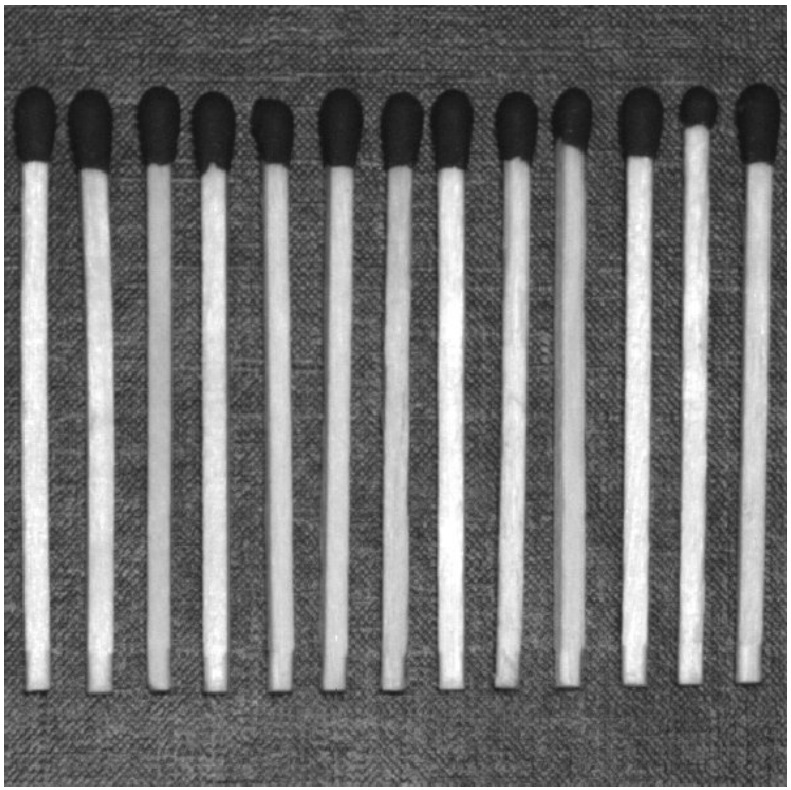
Codage arithmétique

- En pratique : on ne peut pas utiliser de réel !
- Les flottants sont en précision limitée !
- Il faut encoder avec des nombres entiers : l'algorithme devient plus subtil.
- En pratique : le codage arithémique est bien plus performant que Huffman.

Redondances inter-pixels

Constat

- Soit les deux images :



(a) $H = 7.35806$

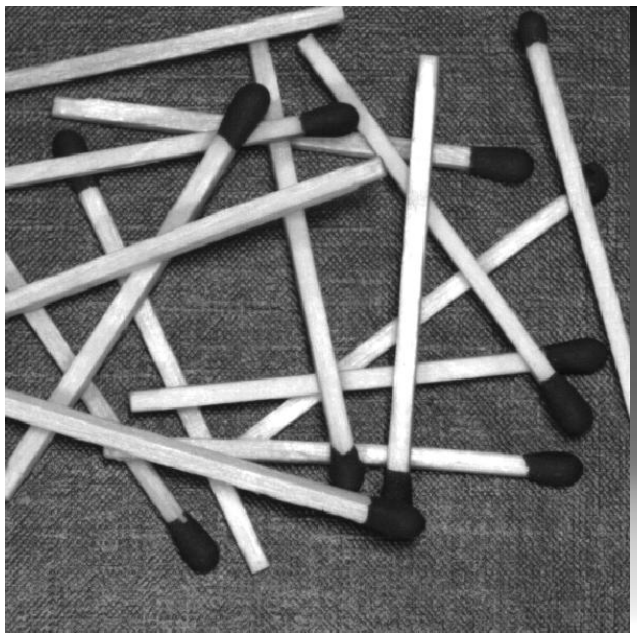
(b) $H = 7.44837$

Redondances inter-pixels

Constat

- De façon extrême, on peut mélanger les pixels d'une image, on gardera la même entropie :

```
tpr -l 1 -c matches.inr | sort -n | cim `par -x -y matches.inr` > m.inr
entropy matches.inr m.inr
7.44837
7.44837
```



Redondances inter-pixels

Constat

- Pourtant l'image non triée contient bien plus d'information que l'image triée :

→ **cette information est contenue dans les relations spatiales entre pixels.**

- Image structurée spatialement : on peut tirer parti de cette information.

Exemple

Un pixel a une probabilité forte d'avoir une valeur proche de celle de ses voisins car les contours sont souvent marginaux dans une image.

Redondances inter-pixels

Principe

- Construire une fonction appelée **prédicteur** qui prédit la valeur d'un pixel en fonction des valeurs des voisins.
- Le prédicteur doit être réversible pour permettre la phase de décodage.
- Pour espérer une bonne compression, le prédicteur ne doit pas trop se tromper : en effet, l'erreur $f - P(f)$ sera d'autant plus faible que le prédicteur P est bon.
- Idéalement l'image des erreurs aura des valeurs proche de zéro : un algorithme d'encodage de type Huffman sera donc très performant pour encoder l'image des erreurs !

Redondances inter-pixels

Quel prédicteur ?

- Pas de prédicteur universel car cela dépend de l'image.
- On peut toutefois considérer des prédicteurs très simple qui donnant de bon résultats sur des images "régulières".
- Prédicteurs linéaires et directionnels :

$$P(f)(l,c) = \text{round}\left(\sum_{i=1}^{m} \alpha_i f(l, c-i)\right)$$

$\rightarrow$ on prédit la valeur d'un pixel à partir des m pixels précédents sur la même ligne.

$\rightarrow$ on peut envisager d'autres directions mais pas plusieurs car la fonction doit être réversible.

Redondances inter-pixels

Prédicteurs linéaires

- Pour se convaincre qu'un tel prédicteur peut être efficace (en terme de compression), prenons un exemple.
- Soit le prédicteur 1D suivant :

$$P(f)(l, c) = \text{round}\left(\alpha f(l, c-1)\right)$$

il prédit donc à partir de la valeur du pixel précédent.

- L'idée ici est donc d'encoder que la différence avec le pixel précédent en espérant que la dynamique sera moindre.

Remarque

- *Si l'image en entrée est codée sur m bits, il faut en principe* $m+1$ *bits en sortie pour coder le signe.*
- *Néanmoins, la dynamique doit baisser significativement (sinon ce n'est intéressant), on garde alors le même nombre de bits.*

Redondance inter-pixels

Prédicteur linéaire du premier ordre

- Exemple avec l'image 4×4 :

$f(l, c)$

119	123	168	119
123	119	168	168
119	119	107	119
107	107	119	119

$P(f)(l, c) = f(l, c-1)$

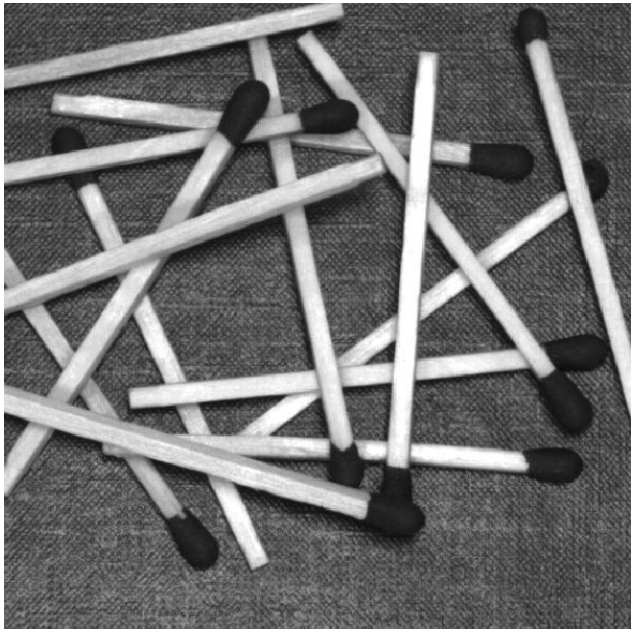
0	119	123	168
0	123	119	168
0	119	119	107
0	107	107	119

- On décale donc d'une colonne vers droite en complétant avec des zéros.
- Différence des deux images :

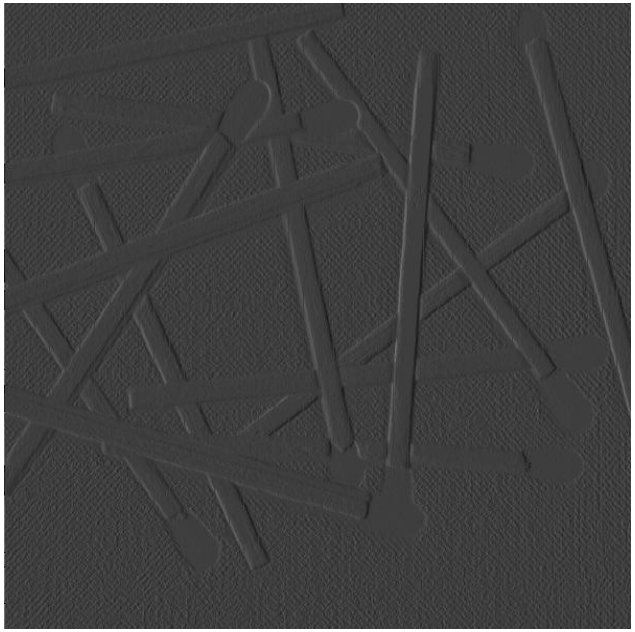
119	4	45	-49
123	-4	49	0
119	0	-12	12
107	0	12	0

Redondance inter-pixels

Exemple sur image naturelle



(c) Image originale, I



(d) $I - P(I)$ (normalisé)

Image	taille	H	taille après Huffman	C_R
I	360256	7.44	337808	1.07
$I - P(I)$	360256	5.74	260854	1.38

Redondance inter-pixels

Décodage

FIGURE – Les 100 premiers pixels de 4 lignes successives

- La première colonne contient les valeurs originales (on leur a soustrait 0).
- Les autres colonnes contiennent les différences des valeurs des pixels avec celles des pixels précédents.
- Exemple sur l'image 4×4 :

119	4	45	-49
123	-4	49	0
119	0	-12	12
107	0	12	0

Redondance inter-pixels

Décodage

- Il faut faire une somme cumulée des colonnes :

119	4	45	-49
123	-4	49	0
119	0	-12	12
107	0	12	0

119	123	45	-49
123	119	49	0
119	119	-12	12
107	107	12	0

119	123	168	-49
123	119	168	0
119	119	107	12
107	107	119	0

119	123	168	119
123	119	168	168
119	119	107	119
107	107	119	119

Redondances psychovisuelles

- Une méthode **destructive** pour compresser de l'image.
- Éliminer l'information non pertinente ou plutôt la moins pertinente **visuellement**.
- Vision humaine :
  - sensible aux variations d'intensité
  - moins sensible aux nuances de couleur
  - sensibilité différente selon les couleurs
  - ...
- On peut éliminer potentiellement beaucoup d'information et donc espérer un gain de compression important.

Redondances psychovisuelles

Quantification scalaire uniforme

- L'œil est sensible aux variations fortes de la luminosité : on peut alors privilégier les valeurs les plus fortes au détriment des plus faibles.

$\Rightarrow$ mettre les bits de poids faible à 0.

Exemple (Mettre les 4 bits de poids faible à zéro)

```
pour chaque pixel p d'une image I faire
    I(p) = I(p)  AND  11110000
fin pour
```

En C on peut écrire :

```
I[p] = I[p] & 0xF0;
```

ou encore :

```
I[p] &= 0xF0;
```

Redondances psychovisuelles

Quantification scalaire uniforme

- On peut illustrer facilement cette opération avec la commande Inrimage `logic`.
- Visualiser les 4 bits de poids fort et de poids faible :

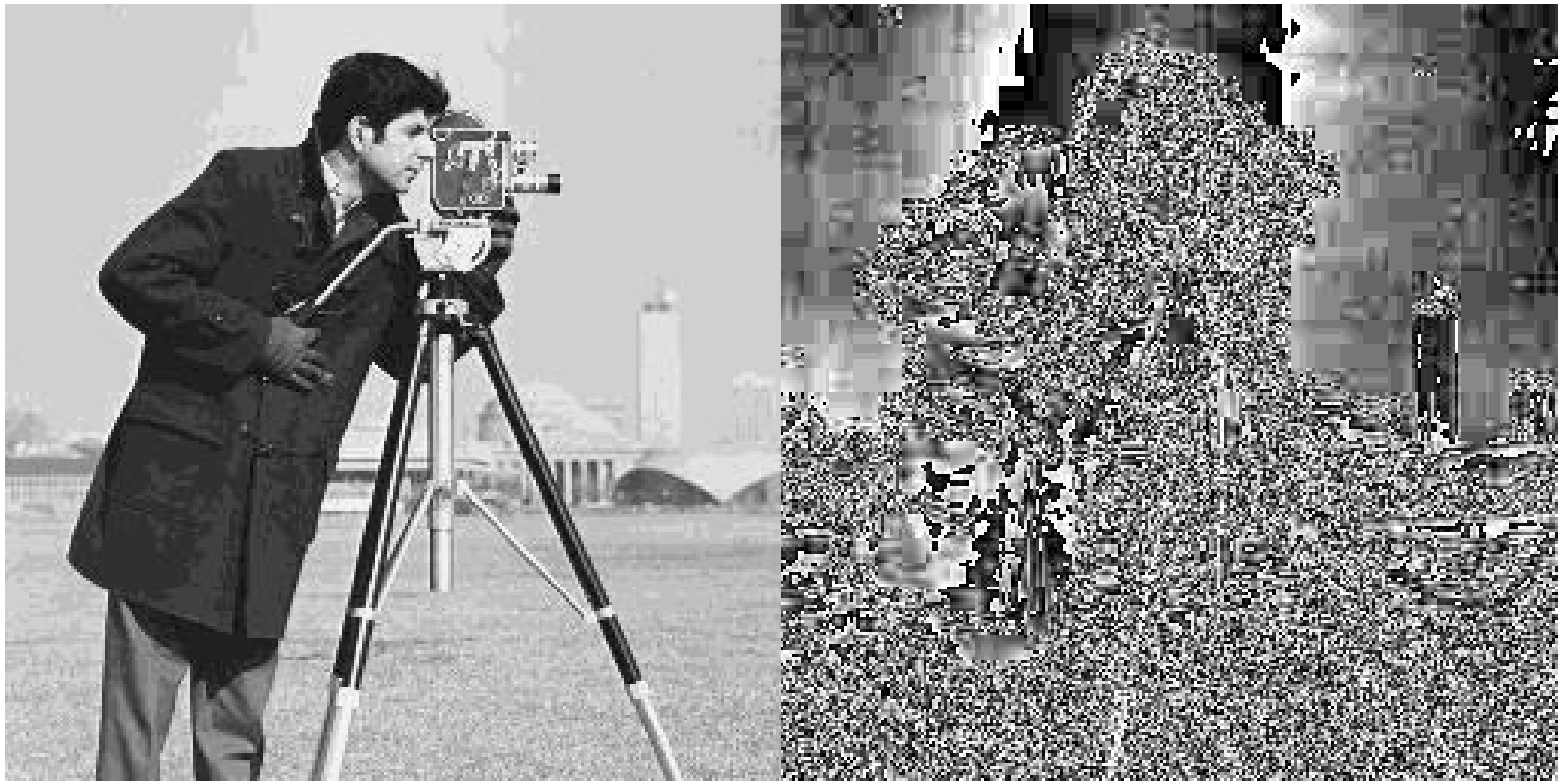
```
logic -ma F0 cameraman.inr | xvis -nu
logic -ma 0F cameraman.inr | xvis -nu
```

Redondances psychovisuelles

Quantification scalaire uniforme

- Les bits de poids faible ne comportent que peu d'information structurelle :

```
logic -ma F0 cameraman.inr | xvis -nu
logic -ma 0F cameraman.inr | norma | xvis -nu
```



Redondances psychovisuelles

Quantification scalaire uniforme

```
logic -ma F0 cameraman.inr | xvis -nu
logic -ma E0 cameraman.inr | xvis -nu
```

(a) 4 bits

(b) 3 bits

Redondances psychovisuelles

Quantification scalaire uniforme

```
logic -ma C0 cameraman.inr | xvis -nu
logic -ma 80 cameraman.inr | xvis -nu
```

(c) 2 bits

(d) 1 bit

Redondances psychovisuelles

Quantification scalaire uniforme

- Les bits de poids faible étant à zéro, une compression Huffman sera très rentable.
- On peut également utiliser les capacités du format INRIMAGE pour ne stocker que *n* bits.

Exemple

En C, on peut utiliser l'opérateur de décalage de bits (vers la droite) :

```
/* decalage de n bits */
valeur = valeur >> n;
/* ou encore */
valeur >>= n;
```

puis écrire l'image résultat dans un format `PACKEE` *sur n bits.*

Redondances psychovisuelles

Quantification scalaire uniforme

- La commande Inrimage `cco` réalise directement ce genre de conversion.

```
cco -b 4 -p cameraman.inr | xvis -nu
cco -b 3 -p cameraman.inr | xvis -nu
```

(e) 4 bits

(f) 3 bits

Redondances psychovisuelles

Quantification scalaire uniforme

```
cco -b 2 -p cameraman.inr | xvis -nu
cco -b 1 -p cameraman.inr | xvis -nu
```

(g) 2 bits

(h) 1 bits

Redondances psychovisuelles

Avantages/inconvénients

- Cela peut être satisfaisant car on préserve l'information **structurelle** de l'image (i.e. les contours), mais on perd sur les textures.
- On peut faire apparaitre des faux contours (penser à un dégradé marqué).
- La réduction du codage est brutale : en général la vision humaine est plus sensible aux changements de luminosité (contours).
- L'idée est alors proposer une quantification qui tienne compte des caractéristiques de la vision humaine (*i.e.* qui préserve les contours) et qui n'en créé pas de nouveaux.

Redondances psychovisuelles

Quantification IGS

- La quantification IGS (improved gray scale) consiste à pertuber chaque pixel par une petite valeur aléatoire, c'est-à-dire les bits les moins significatifs

```
algo IGS;
  S := 0;
  pour chaque valeur A de l'image faire
    si  lowbit(A,n) == lowbit(0xFF,n) alors
      S := A;
    sinon
      S := A + lowbit(S,n)
    fin si
    A = S >> n;
  fin pour
fin algo
```

`lowbit()` retourne la valeur des *n* bits de poids faible.

Quantification IGS

Exemple

pixel	valeur	S	code IGS
-	-	0000 0000	-
i	0110 1100	0110 1100	0110
i+1	1000 1011	1001 0111	1001
i+2	1000 0111	1000 1110	1000
i+3	1111 0100	1111 0100	1111

Quantification IGS

Exemple

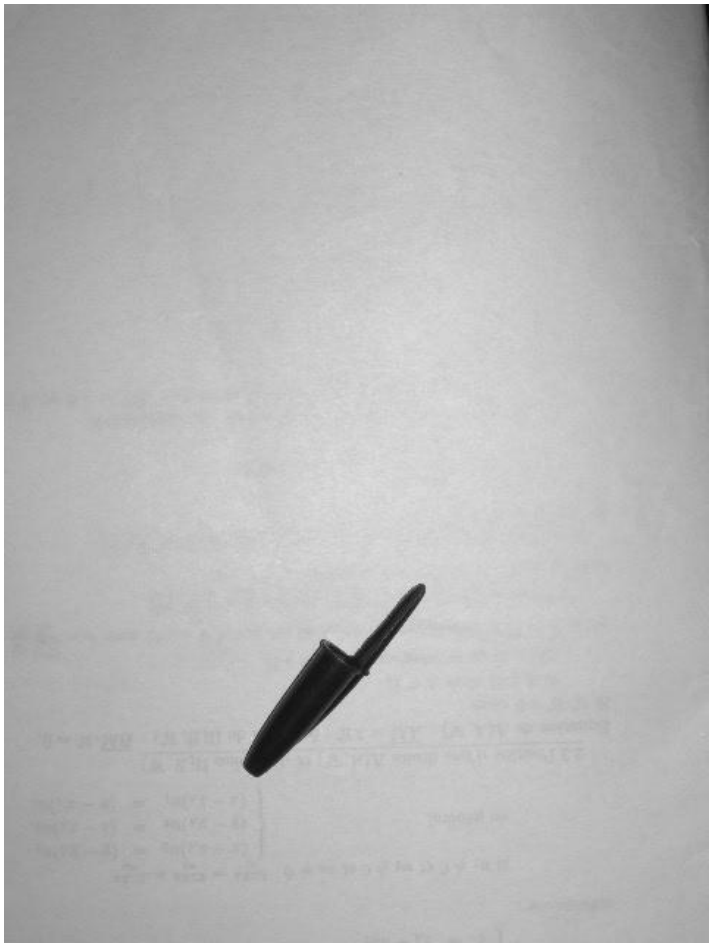
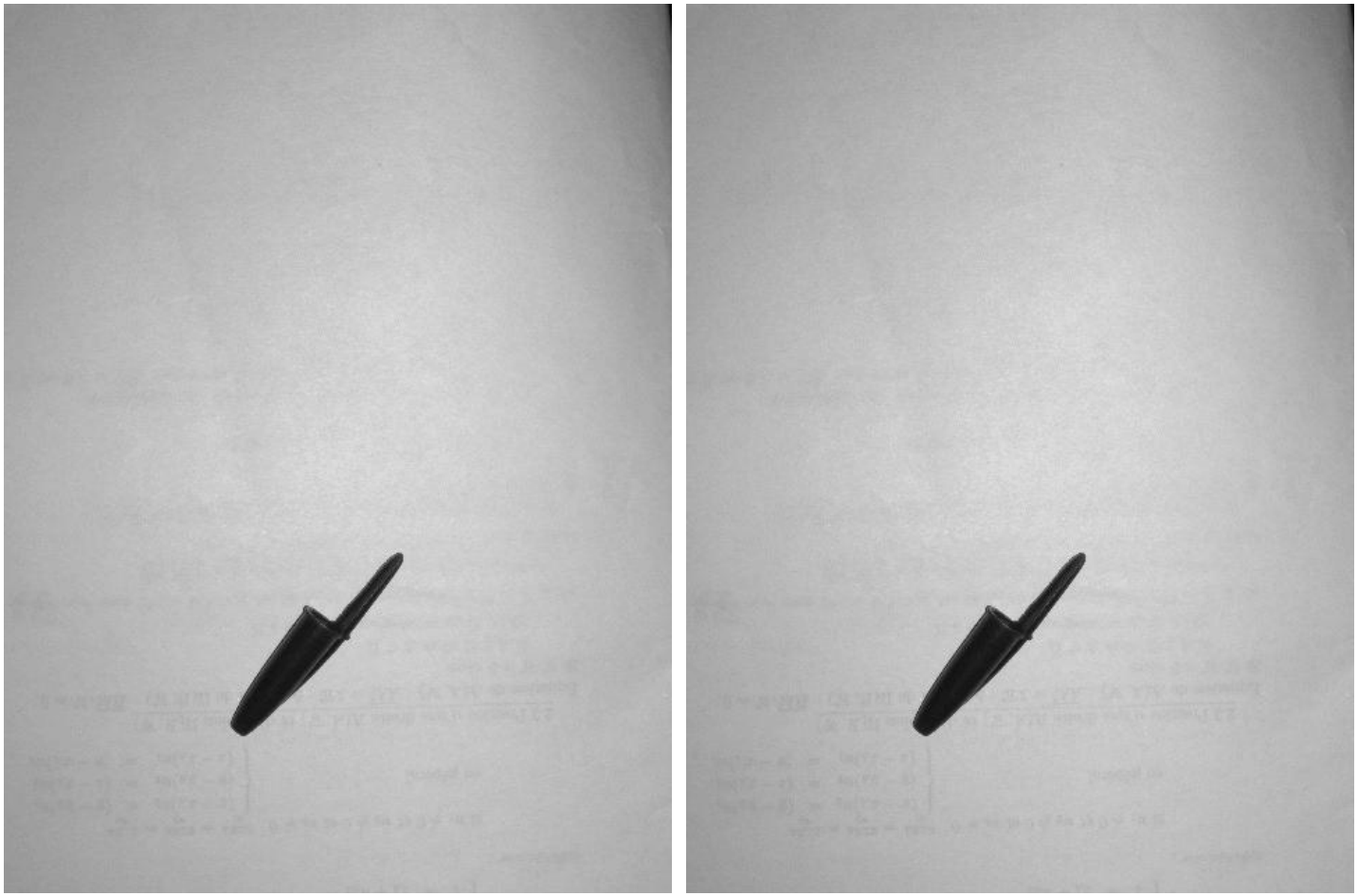


FIGURE – Image avec un dégradé important

Quantification IGS

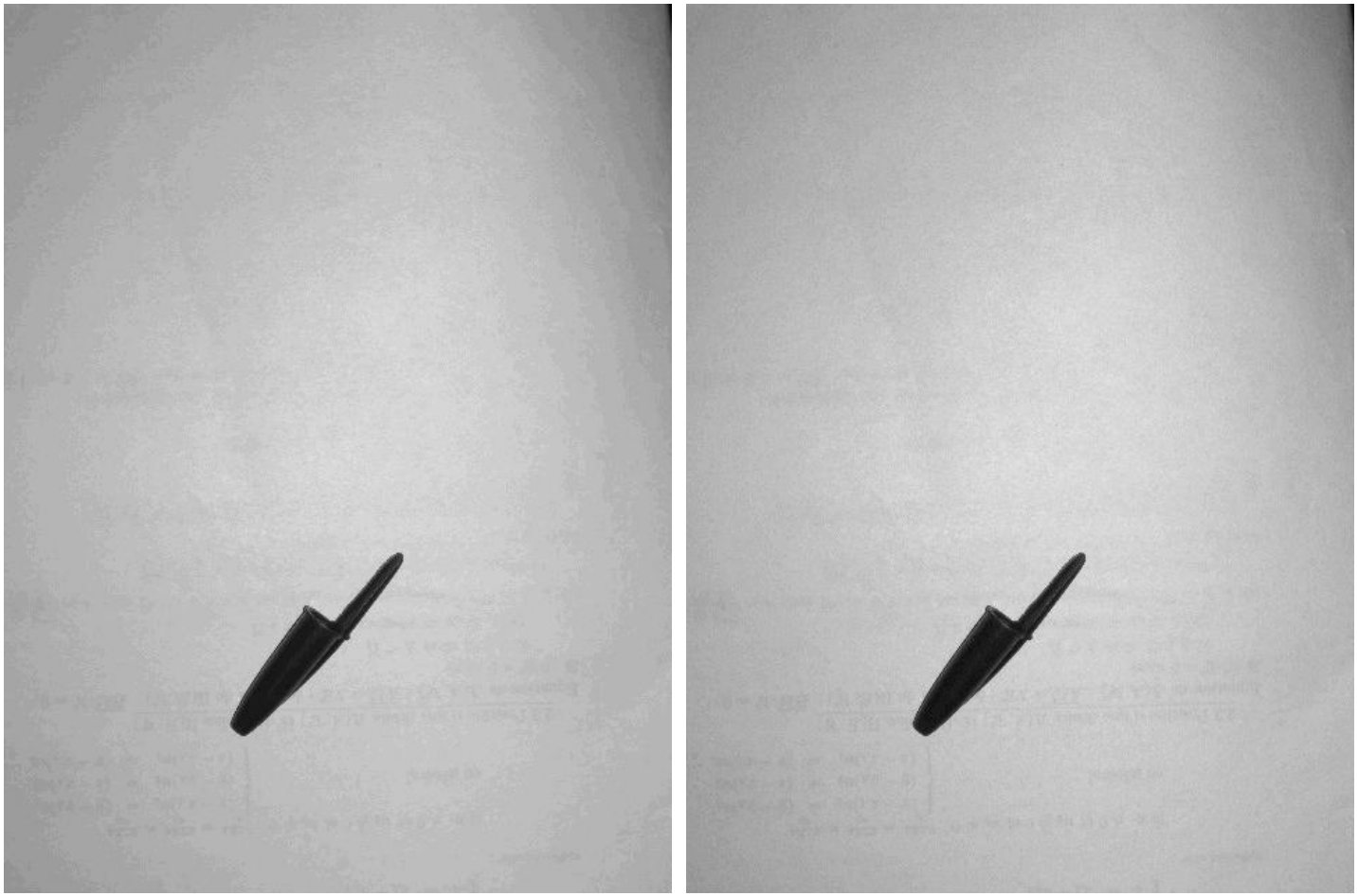
Exemple



(a) `cco -b 6 capuch.inr` (b) `igs -b 6 capuch.inr`

Quantification IGS

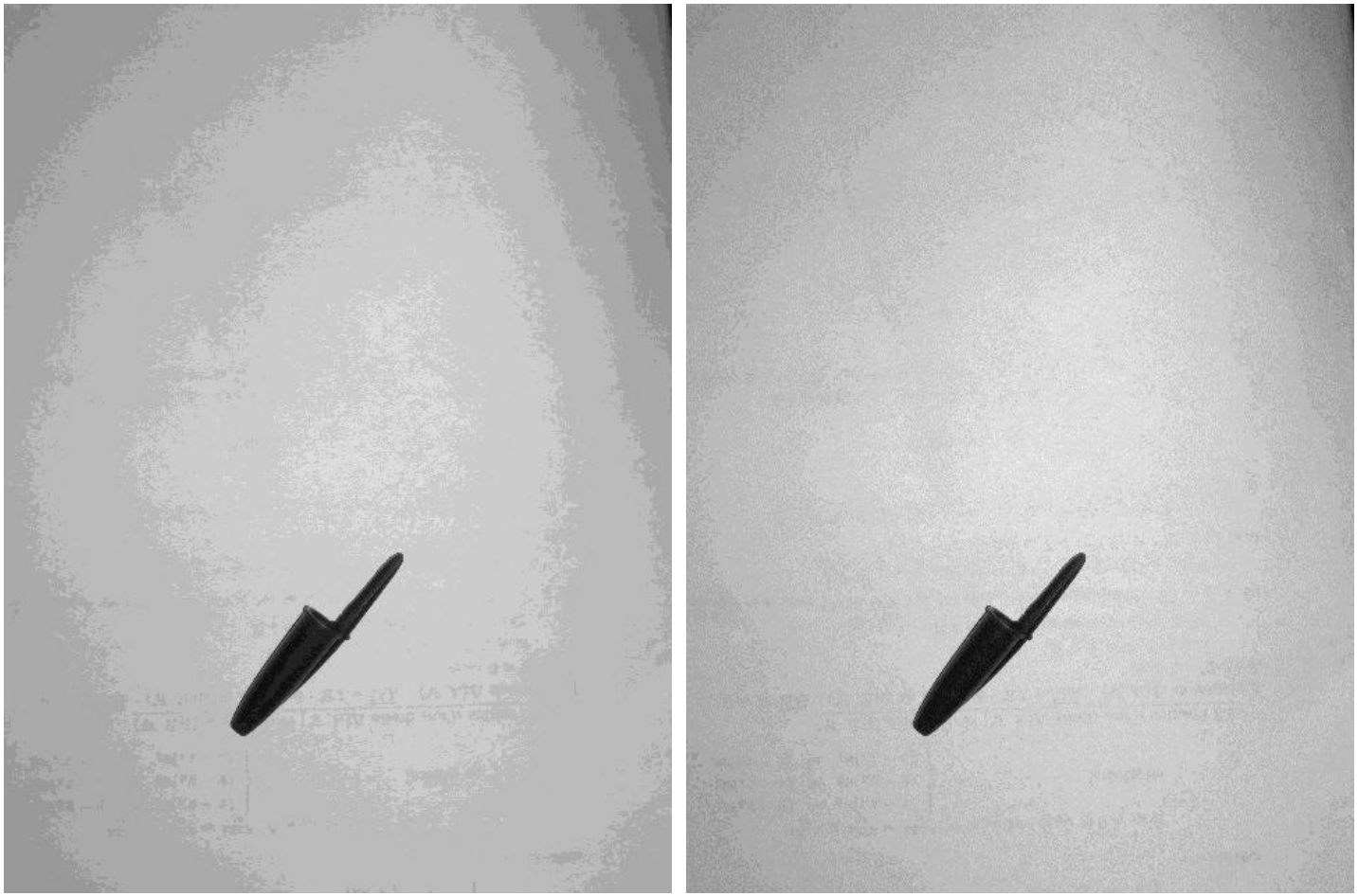
Exemple



(c) `cco -b 5 capuch.inr` (d) `igs -b 5 capuch.inr`

Quantification IGS

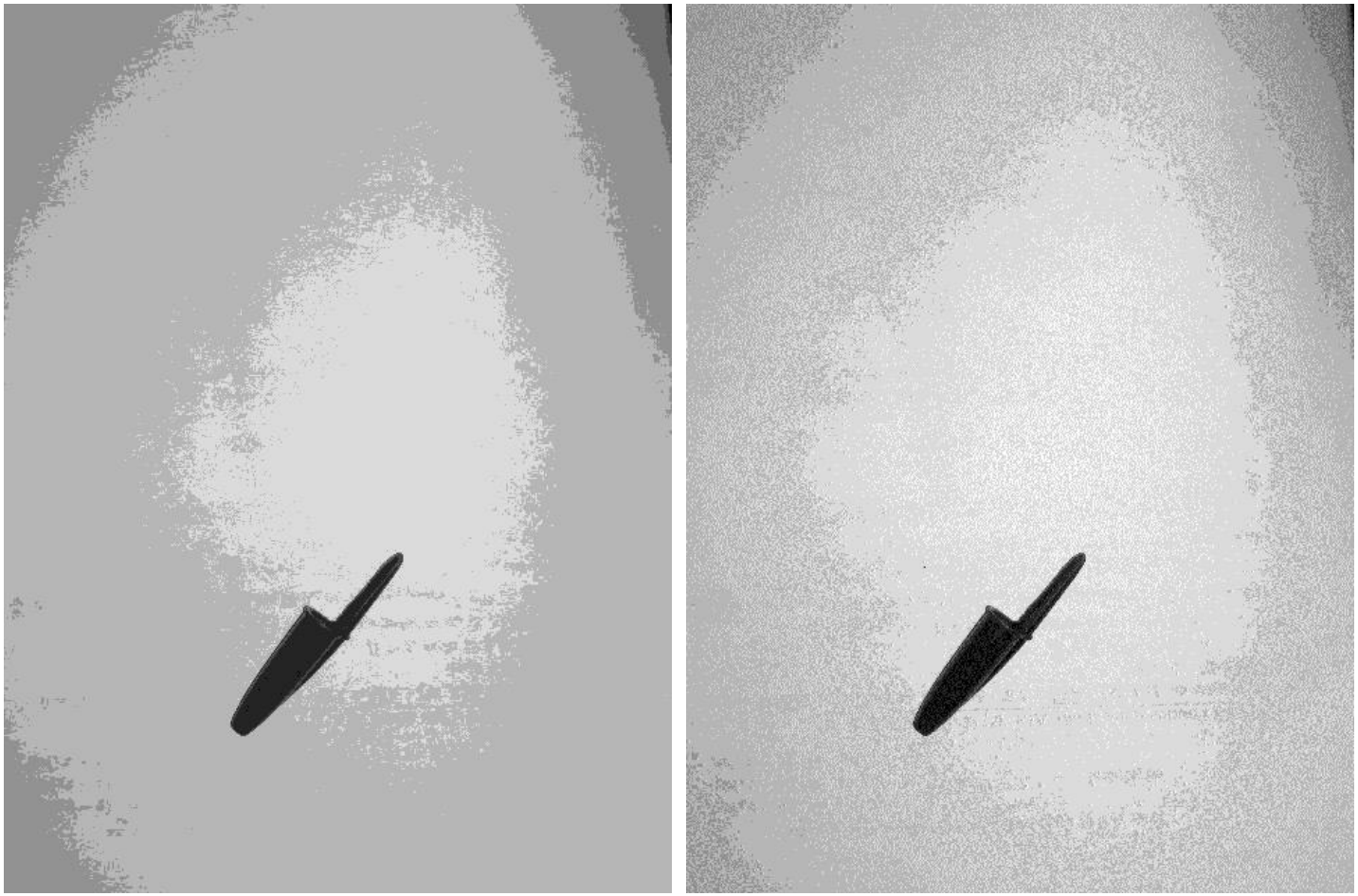
Exemple



(e) `cco -b 4 capuch.inr` (f) `igs -b 4 capuch.inr`

Quantification IGS

Exemple



(g) `cco -b 3 capuch.inr` (h) `igs -b 3 capuch.inr`

Compression JPEG

Un peu d'histoire

- JPEG : norme ISO de compression créée dans les années 80 par un comité international (PEG, Photographic Export Group). Norme effective en 1992
- Objectif : un format permettant à la fois de fort taux de compression (utile pour les transmissions) mais qui préserve une bonne qualité de l'image.
- Le comité fusionne en 1987 avec le CEI (International Electrotechnical Commission) pour former JPEG (Join-PEG)
- Site officiel : http://www.jpeg.org/jpeg/

Compression JPEG

Principes

- La compression JPEG est une méthode qui met en œuvre tous les principes précédemment vue.
- L'état de quantification est la plus importante : une analyse de type Fourier éliminera une grande partie de l'information non visuellement pertinente.
- La compression destructive permet un gain de 1/10, voire davantage, en gardant une excellente qualité visuelle.
- Implémentation libre de droits écrite en C : http://www.ijg.org (très largement utilisée).

JPEG : une compression destructive

- Cette compression consiste également à réduire une redondance psychovisuelle : l'œil humain distingue difficilement les différences de nuance entre deux pixels voisins de couleurs proche.
- Le bruit est souvent associé a un signal de haute fréquence.
- Dans l'espace de Fourier, cela signifie que la vision humaine est moins sensible aux fréquences les plus hautes : on ne gardera que les basses fréquences.
- C'est ce seuillage, brutal, qui permet une réduction conséquente de la taille de codage de l'image.
- En pratique, on ne calcule pas la transformée de Fourier mais plutôt une transformée en cosinus.

Transformée en cosinus

- Définition : Discret Cosinus Transform (DCT) 2D.

$$DCT(u,v) = \frac{\alpha(u)\alpha(v)}{\sqrt{MN}} \times \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} I(x,y)\cos\left[\frac{(2x+1)u\pi}{2M}\right]\cos\left[\frac{(2y+1)v\pi}{2N}\right]$$

avec $\alpha(u) = \frac{1}{\sqrt{2}}$ si $u = 0$, 1 sinon.

- On remarque que cette transformée est séparable (DCT 1D en x composée par une DCT 1D en y),
- pourquoi une telle transformée ?

Transformée en cosinus

- Définition : Discret Cosinus Transform (DCT) 2D.

$$DCT(u,v) = \frac{\alpha(u)\alpha(v)}{\sqrt{MN}} \times \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} I(x,y) \cos\left[\frac{(2x+1)u\pi}{2M}\right] \cos\left[\frac{(2y+1)v\pi}{2N}\right]$$

avec $\alpha(u) = \frac{1}{\sqrt{2}}$ si $u = 0$, 1 sinon.

- On remarque que cette transformée est séparable (DCT 1D en x composée par une DCT 1D en y),
- pourquoi une telle transformée ?
  - l'image est une fonction réelle, en la symétrisant, on obtient une fonction paire
  - dans ce cas la TF est réelle (tous les sinus s'annulent), on obtient une transformée très proche de la transformée en cosinus
  - la transformée en cosinus est bien adaptée aux images naturelles, elle nécessite moins de coefficients que la TF.

Transformée en cosinus

1. la transformée en cosinus, tout comme la transformée de Fourier est parfaitement réversible (Inverse Discret Cosinus Transform) :

$$IDCT(x,y) = \frac{2\alpha(u)\alpha(v)}{\sqrt{MN}} \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} DCT(u,v) \times \cos\left[\frac{(2x+1)u\pi}{2M}\right] \cos\left[\frac{(2y+1)v\pi}{2N}\right]$$

2. la transformée de Fourier représente dans $\mathbb{C}$ une image réelle car elle décompose un signal en fréquence mais aussi en phase.
3. l'information de phase est essentielle : on doit la préserver **intacte** pour la reconstruction mais elle est inutile dans le cadre de la compression.
4. La DCT décompose uniquement en fréquence. On pourra éliminer les hautes fréquences et reconstruire l'image sans avoir à garder une information sur la phase.

Décomposition en blocs

- La DCT n'est pas calculée sur l'image entière mais sur des blocs 8×8.
- En effet, on s'intéresse qu'aux hautes fréquences, il est donc inutile et moins efficace de considérer l'image entière. Une analyse locale est suffisante.
- Après calcul de la DCT, on trouve :
  - la fréquence $(0, 0)$, moyenne de l'image (comme pour la transformée de Fourier) ;
  - les fréquences sont d'autant plus hautes que l'on s'éloigne du pixel $(0, 0)$ dans l'espace de la DCT.

Calcul de la DCT sur un bloc

Un exemple

- un bloc 8×8 de pixels :

139	144	149	153	155	155	155	155
144	151	153	156	159	156	156	156
150	155	160	163	158	156	156	156
159	161	162	160	160	150	159	159
159	160	161	162	162	155	179	155
161	161	161	161	160	157	157	157
162	162	161	163	162	157	157	157
162	162	161	161	163	158	158	158

Calcul de la DCT sur un bloc

Un exemple

- la DCT sur ce bloc :

1260	-1	-12	-5	2	-2	-3	1
-23	-17	-6	-3	-3	0	0	-1
-11	-9	-2	2	0	-1	-1	0
-7	-2	0	1	1	0	0	0
-1	-1	1	2	0	-1	1	1
2	0	2	0	-1	1	1	-1
-1	0	0	-1	0	2	1	-1
-3	2	-4	-2	2	1	-1	0

Quantification JPEG

- La DCT en elle-même ne compresse pas.
- La quantification est l'étape de réduction de l'information proprement dit, puisque on va atténuer voire supprimer certaines fréquences, c'est-à-dire certaines composantes de la DCT.
- Chaque bloc est divisé (terme à terme) par une matrice de quantification puis arrondi à l'entier le plus proche.
- C'est la matrice de quantification qui va régler le taux de compression et donc, la qualité de l'image après décodage.

Quantification JPEG

Exemple

- Par exemple : cette table correspond à une compression moyenne (et une qualité moyenne)

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

Quantification JPEG

Exemple

- divisé à la DCT du bloc image précédent, on obtient :

79	0	-1	0	0	0	0	0	0	0
-2	-1	0	0	0	0	0	0	0	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

Quantification JPEG

Exemple

- Cette matrice de quantification a une compression encore plus forte (et une qualité dégradé).

18	21	26	66	99	99	99	99
24	26	56	99	99	99	99	99
47	66	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99

Ordre de visite des pixels d'un bloc

- La quantification créé beaucoup de coefficients nuls au delà d'une certaine distance du coefficient $(0, 0)$ ou de valeurs qui se répètent.
- La répétition de valeurs identiques, en particulier 0, peut être codée efficacement à condition qu'ils soient contiguës.
- Pour cette raison, on parcours les éléments du bloc en "zig-zag"
- Le tableau ci-dessous donne pour chaque élément du bloc, l'ordre de visite.

1	2	6	7	15	16	28	29
3	5	8	14	17	27	30	43
4	9	13	18	26	31	42	44
10	12	19	25	32	41	45	54
11	20	24	33	40	46	53	55
21	23	34	39	47	52	56	61
22	35	38	48	51	57	60	62
36	37	49	50	58	59	63	64

Encodage RLE, puis statistique

- On utilise un codage de type RLE (Run Length Encoding) : les répétitions de valeurs identique sont codées.
- Exemple avec le bloc quantifié précédent : après un parcours en Zig-zag, on obtient la suite :

$$79, 0, -2, -1, -1, -1, 0, -1, 0, 0, \cdots$$

  la suite de zéros se répète 56 fois.
- Codage RLE :

$$79, 1, 0, 1, -2, 1, -1, 3, 0, 1, -1, 1, 0, 56$$

- Après réunion de tous les blocs quantifiés et encodés en RLE, on applique un encodage par dictionnaire de Huffman ou arithémique (plus coûteux).

Codage des pixels et espace de couleurs

- La norme JPEG prévoit d'encoder des images monochromes (1 octet) ou couleur (3 octets).
- On peut encoder directement les RGB, en pratique on choisit un autre espace de couleurs :
  - YUV
  - YCbCr
- En effet, la vision humaine est très sensible aux variations de luminance et moins aux variations chromatiques. Il est donc intéressant de privilégier le canal de luminance au détriment des canaux chromatiques.
- En pratique : on sous-échantillonne (d'un facteur 2) les canaux chromatiques dans la direction verticale ou horizontale ou les deux. La luminance n'est **jamais** sous-échantillonnée.
- Évidemment, un tel sous-échantillonnage n'est pas possible en RGB sans impact conséquent sur l'image.

Encodage JPEG : chaîne de traitement

1. Lecture image
2. Découpage en blocs 8x8
3. Pour chaque bloc faire :
   - 3.1 transformation des couleurs
   - 3.2 sous-échantillonage
   - 3.3 transformée en cosinus
   - 3.4 division par la table de quantification
   - 3.5 encodage RLE
4. Réunion des blocs
5. Encodage Huffman /pp arithmétique
6. Écriture de l'image JPEG.

Encodage JPEG et performance

- Les processeurs des années 90 n'étaient pas performant pour l'encodage/décodage JPEG : utilisation de processeur dédié (DSP ou chipset graphique : la transformée en cosinus y est câblé)
- Les blocs peuvent être traitée indépendamment :
  - architecture parallèle
  - utilisation de pipe-line
- Aujourd'hui la puissance des processeurs permet un encodage/décodage en temps réel.

Décodage JPEG

- La chaîne de décodage suit exactement la chaîne inverse à savoir :
  1. Lecture image JPEG
  2. Décodage Huffman
  3. Découpage en blocs
  4. Pour chaque bloc :
     - 4.1 Décodage RLE
     - 4.2 Quantification Inverse
     - 4.3 IDCT
     - 4.4 Sur-échantillonnage
     - 4.5 Transformation des couleurs
  5. Réunion des blocs
  6. Écriture image décodée

Quantification inverse

- Quantification (encodage) : division par une matrice de quantification.
- Quantification inverse (décodage) : multiplication par la matrice de quantification.
- Cette opération est-elle réversible ?

Quantification inverse

- Quantification (encodage) : division par une matrice de quantification.
- Quantification inverse (décodage) : multiplication par la matrice de quantification.
- Cette opération est-elle réversible ? Non car en divisant puis en arrondissant on perd l'information et c'est ce qui permet la compression.
- Les pertes d'information dans la compression JPEG sont donc :
  - le sous-échantillonnage des canaux chromatiques
  - la quantification.

Exemples avec libjpeg

- Le logiciel libjpeg version 6.2 (développé bénévolement par une équipe nommée Independant JPEG Group) permet l'encodage et de décodage d'images JPEG.
- Une librairie documentée de fonctions C pour coder/décoder des images JPEG.
- Des commandes de manipulations d'image JPEG.
  - `cjpeg` pour encoder
  - `djpeg` pour décoder
  - `jpegtran` pour réaliser quelques opérations de transformation sur l'encodage JPEG d'une image
  - `rdjpegcom`/`wrjpegcom` pour lire/écrire des commentaires dans une image JPEG

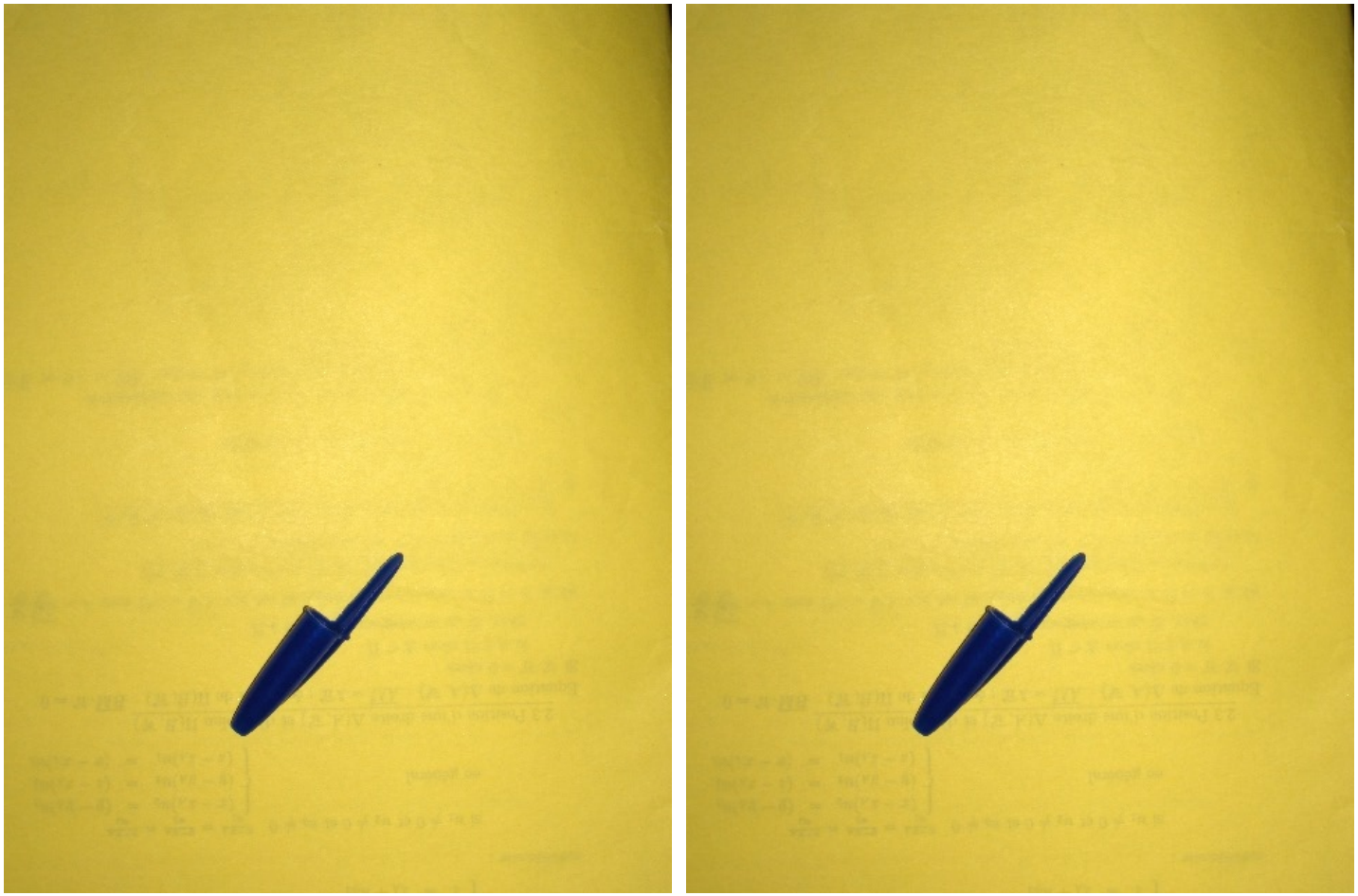
Libjpeg

- La librairie et les commandes `cjpeg`/`djpeg` paramètrent chaque étape de la compression/décompression JPEG.
- Le paramètre le plus important : la qualité l'image JPEG : (option `-quality`)
  - $q = 100$: pas de compression destructive
  - $q = 5$: compression maximale (et perte d'information maximale.
- Chaque valeur de q correspond à une table de quantification.
- Faisons varier la qualité :

```
cjpeg -quality q -outfile capuchon.jpg capuchon.ppm
```

LibJPEG

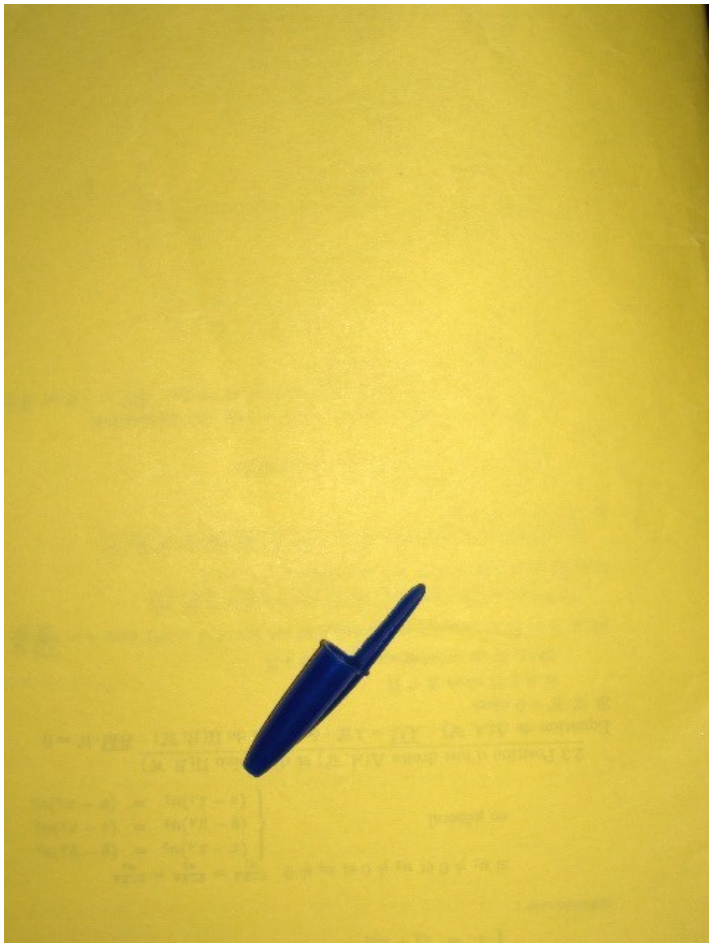
Exemple



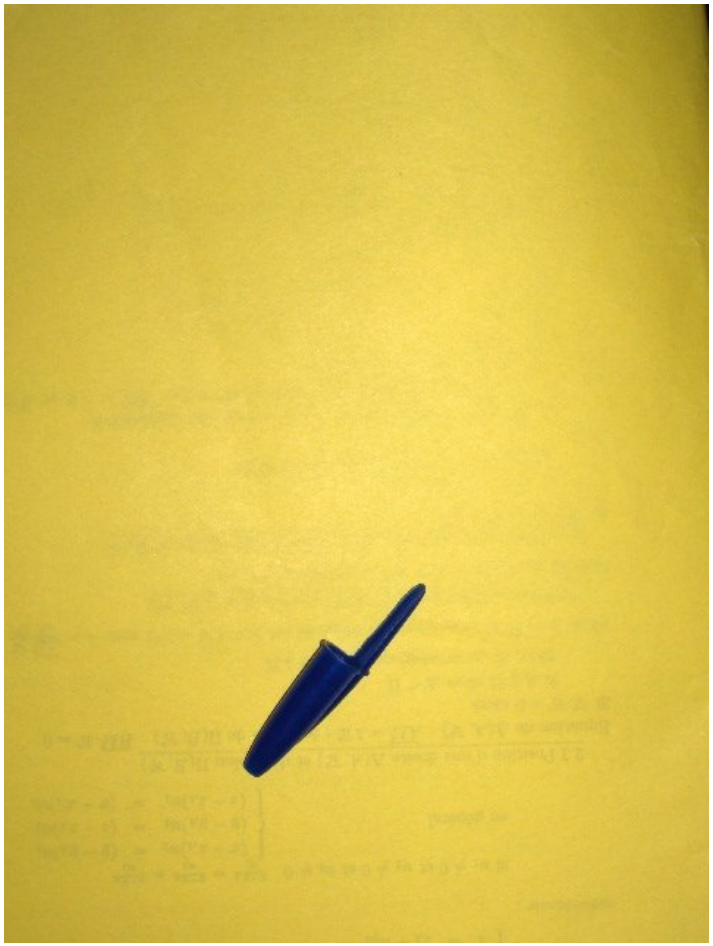
(a) $q = 100, C_R = 4, H = 18.3$ (b) $q = 95, C_R = 6.7, H = 18.3$

LibJPEG

Exemple



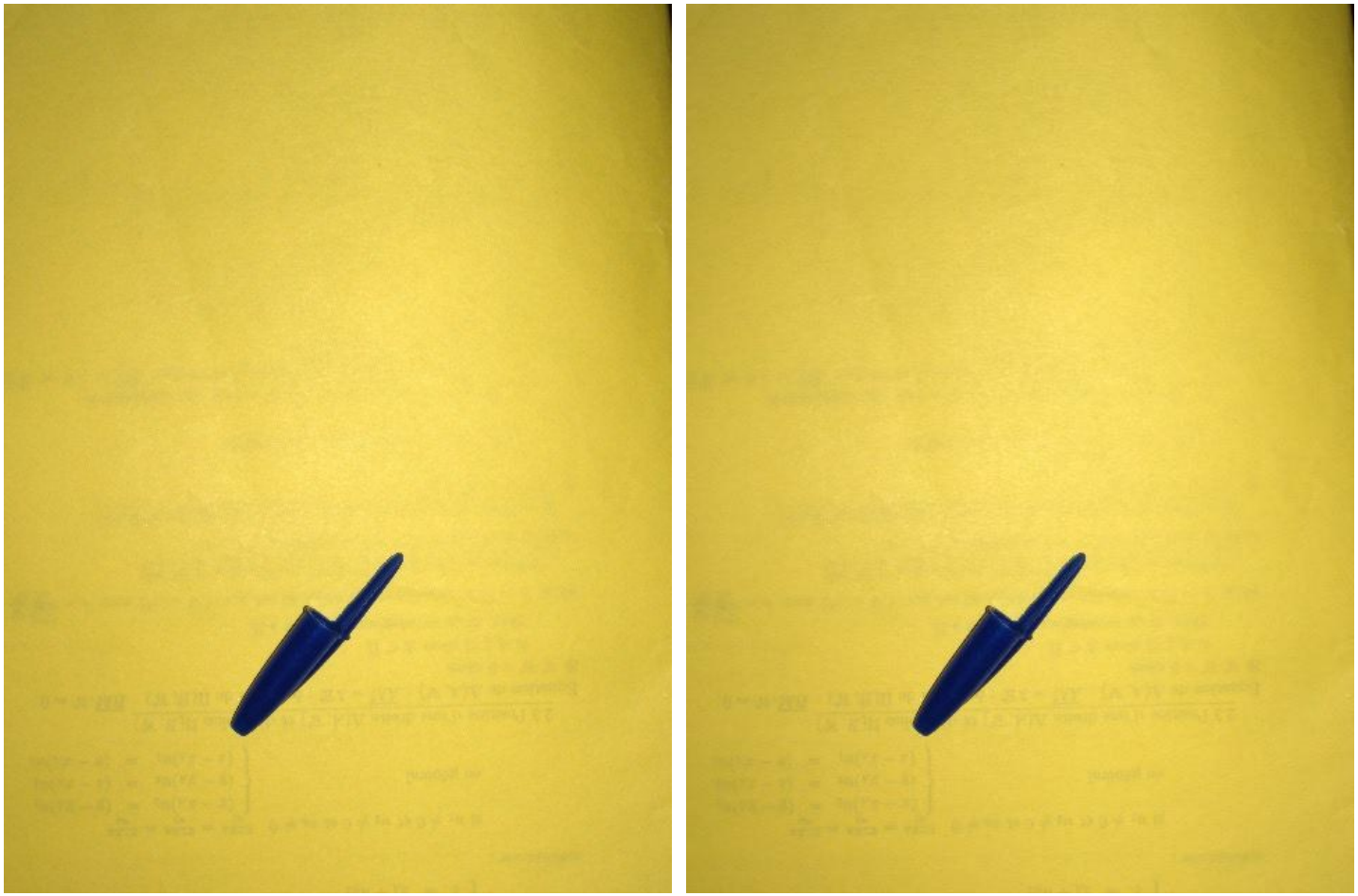
(c) $q = 90, C_R = 8.5, H = 18.3$



(d) $q = 85, C_R = 11.5, H = 18.3$

LibJPEG

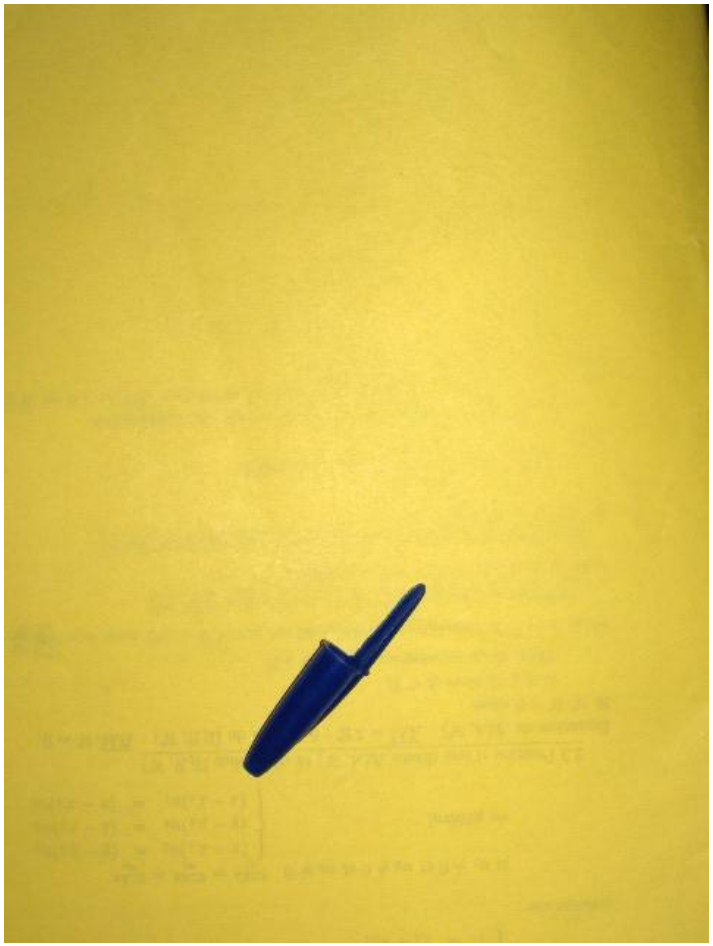
Exemple



(e) $q = 80, C_R = 15.4, H = 18.3$ (f) $q = 75, C_R = 20.4, H = 18.3$

LibJPEG

Exemple

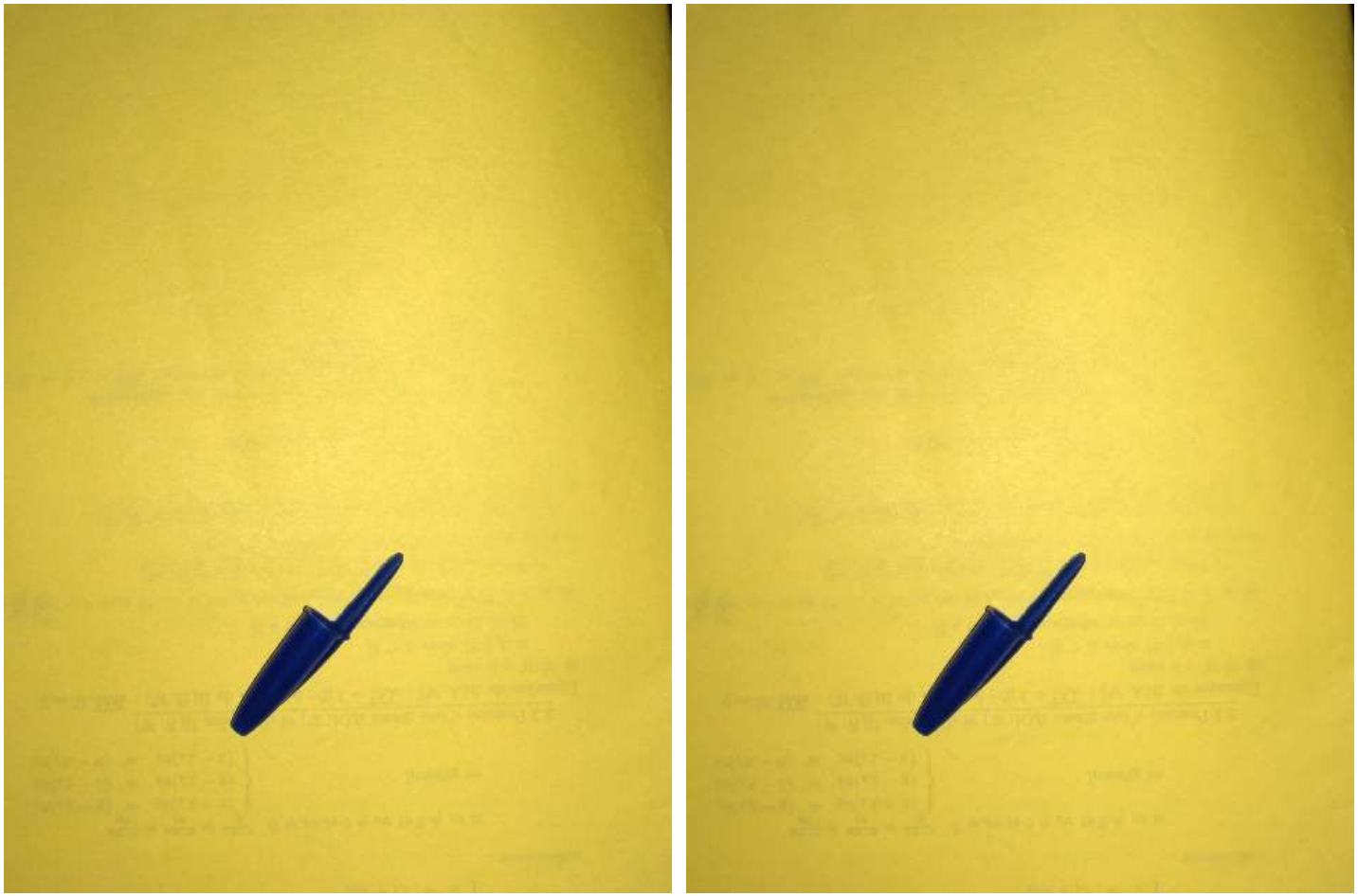


(g) $q = 70, C_R = 24.4, H = 18.3$

(h) $q = 65, C_R = 28.5, H = 18.3$

LibJPEG

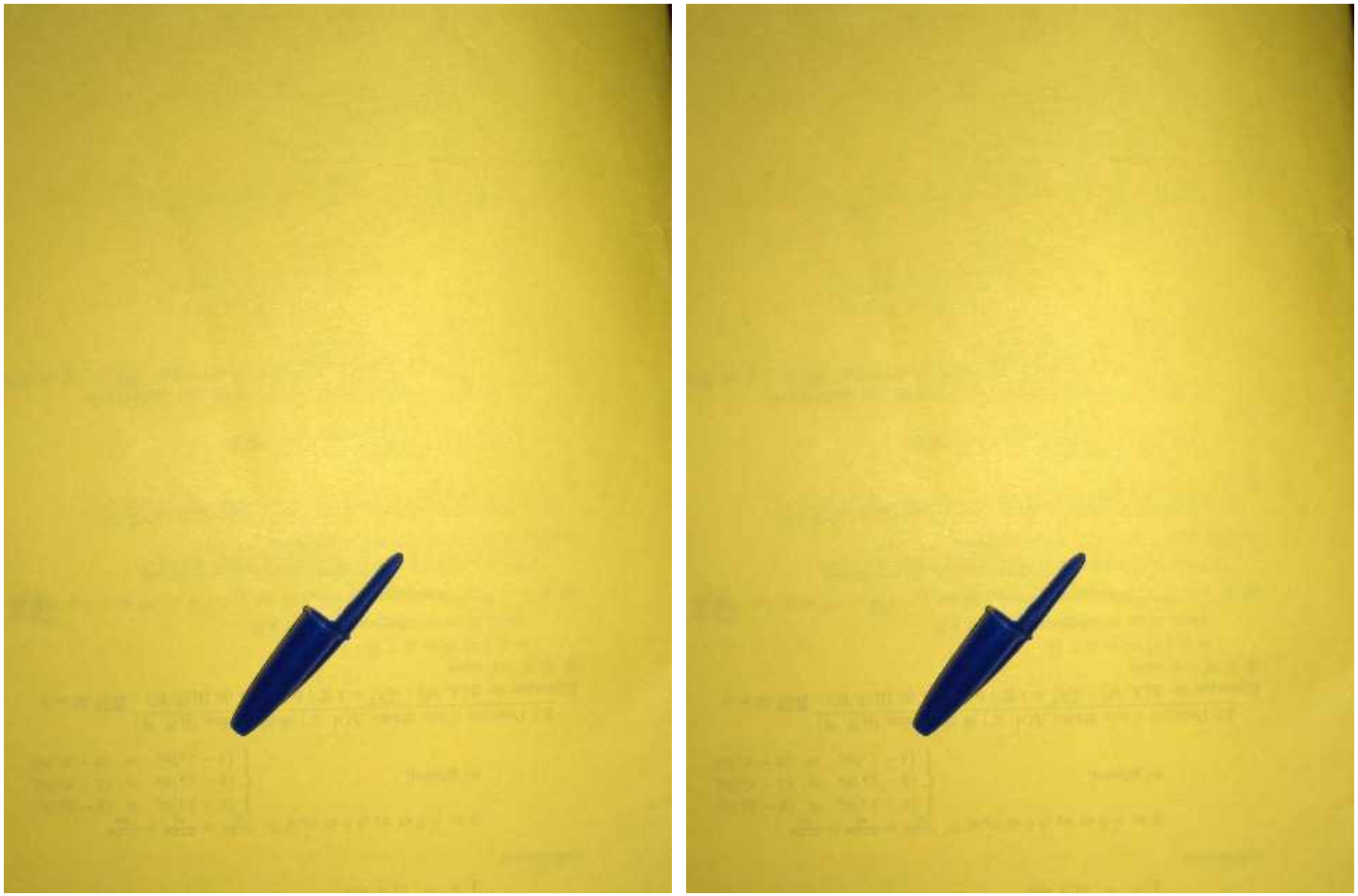
Exemple



(i) $q = 60, C_R = 33.4, H = 18.3$ (j) $q = 55, C_R = 39, H = 18.2$

LibJPEG

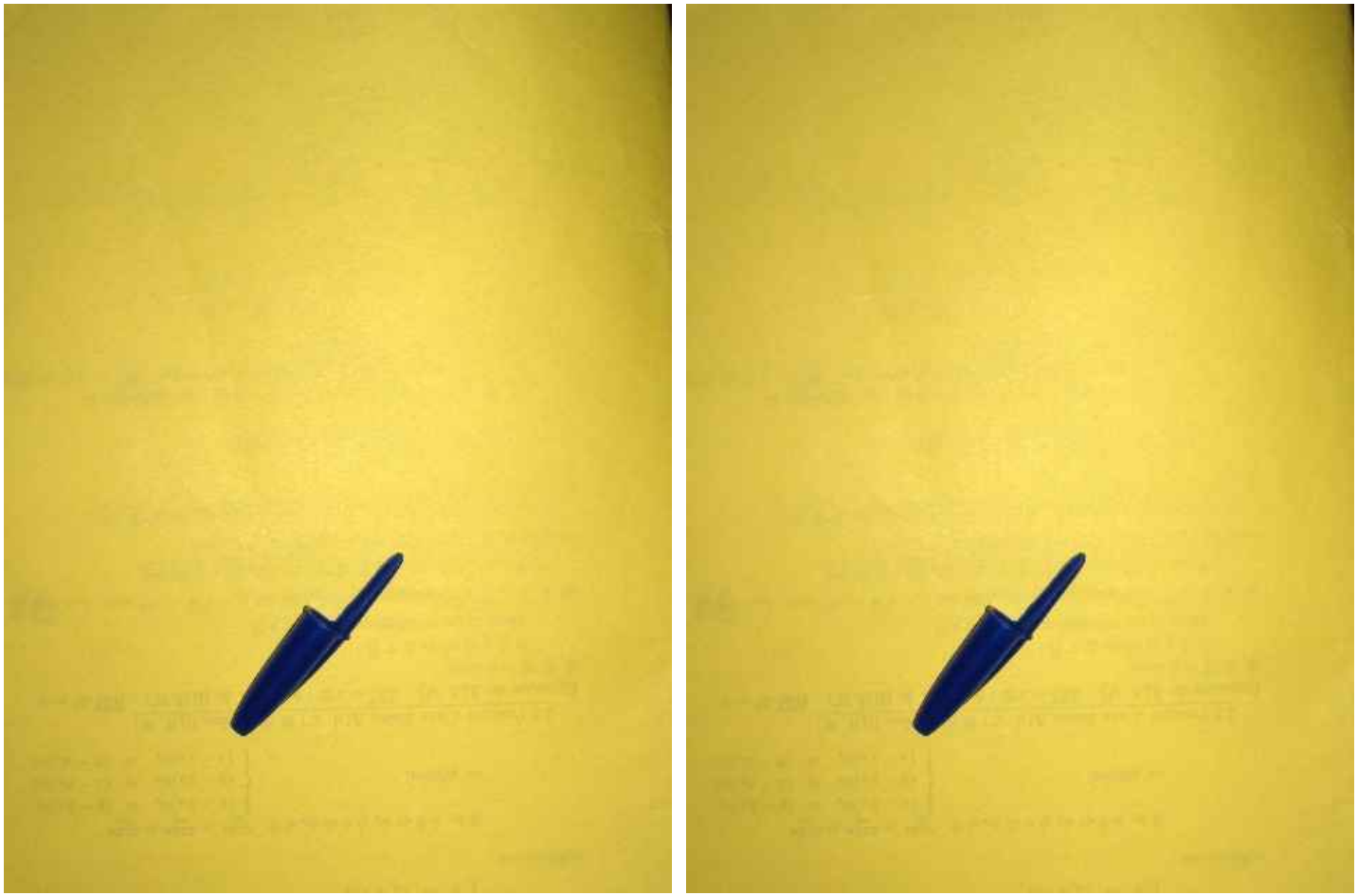
Exemple



(k) $q = 50, C_R = 43.3, H =$ 18.2 (l) $q = 45, C_R = 47.5, H =$ 18.1

LibJPEG

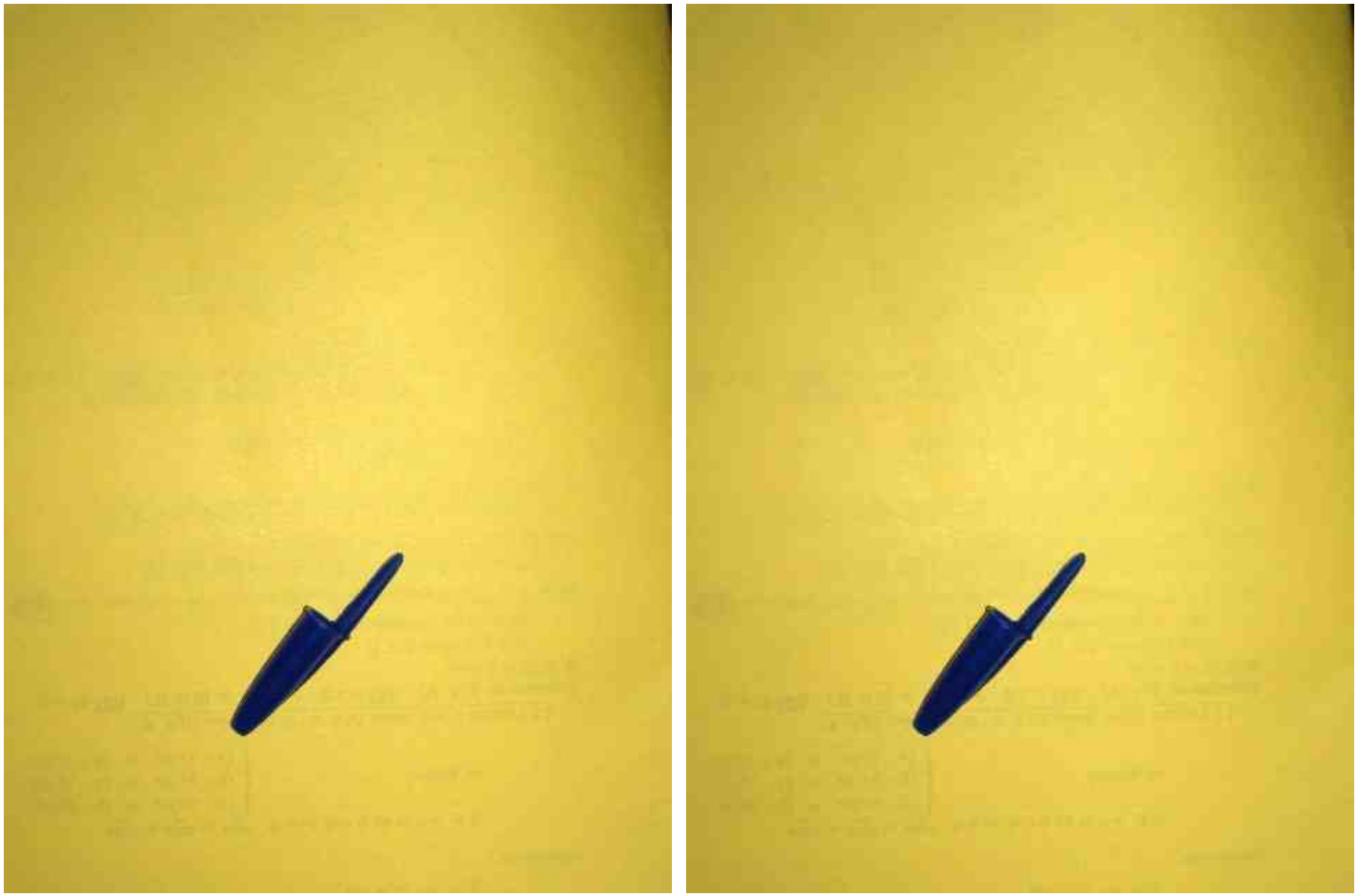
Exemple



(m) $q = 40, C_R = 54.3, H = 17.8$ (n) $q = 35, C_R = 61.9, H = 17.7$

LibJPEG

Exemple

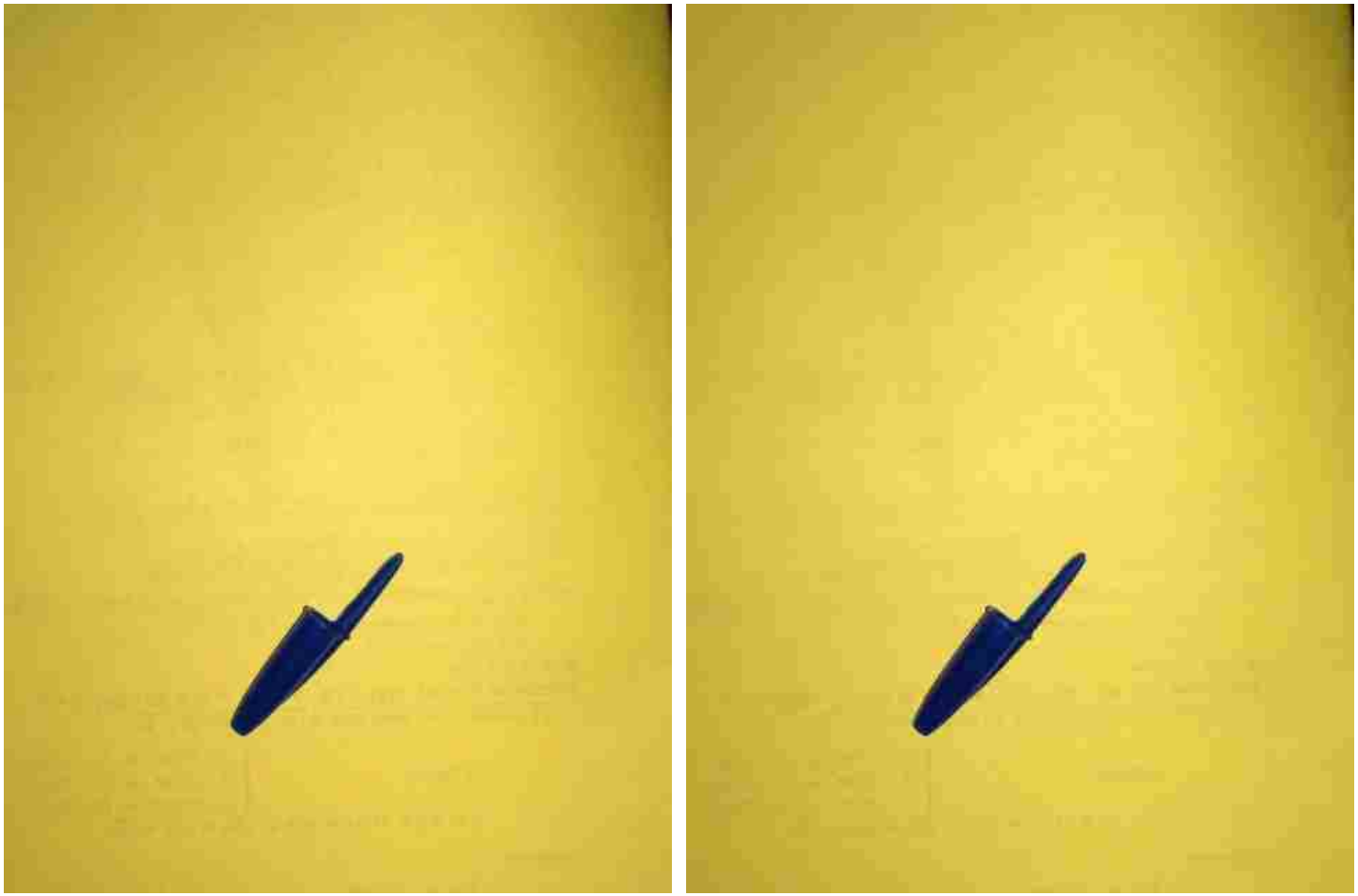


(o) $q = 30, C_R = 70.8, H = 17.2$

(p) $q = 25, C_R = 81.4, H = 16.4$

LibJPEG

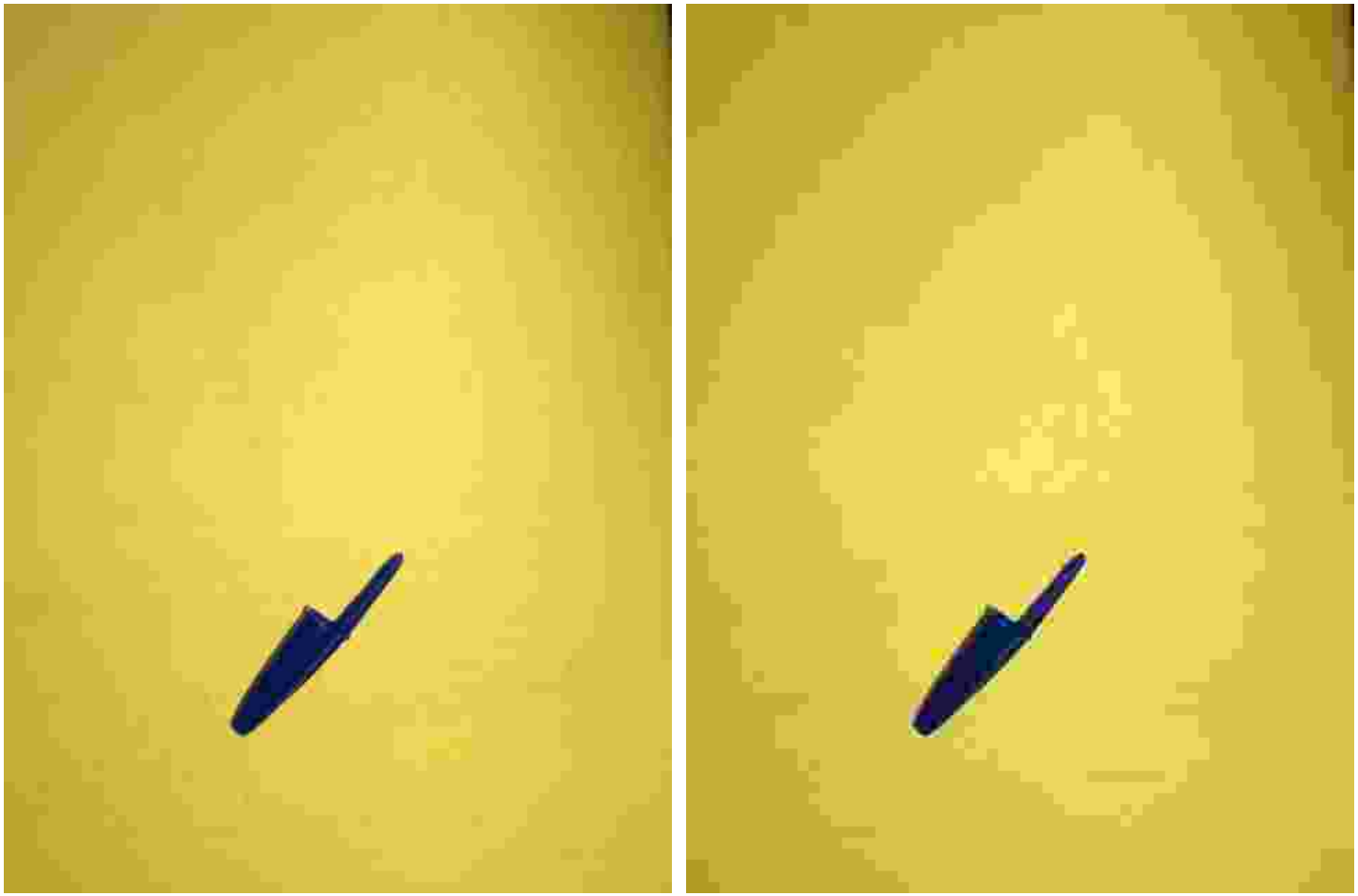
Exemple



(q) $q = 20, C_R = 94.6, H = 14.3$ (r) $q = 15, C_R = 101, H = 12.0$

LibJPEG

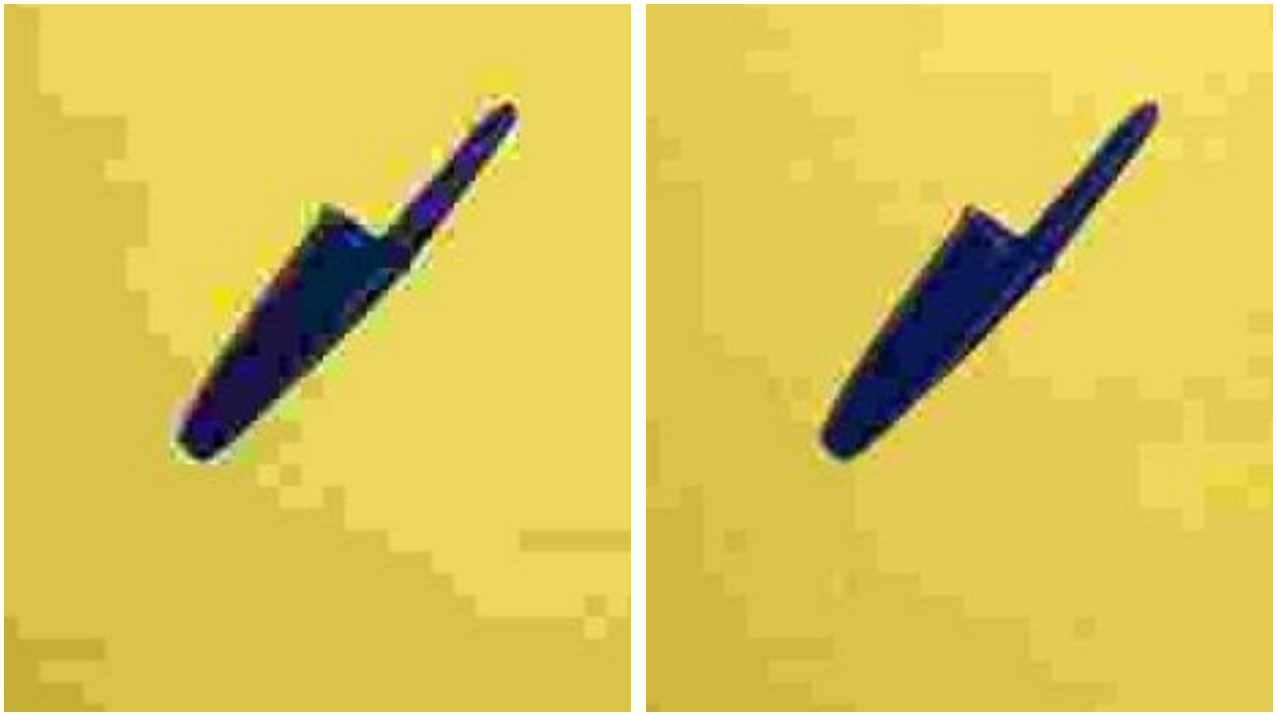
Exemple



(s) $q = 10, C_R = 117, H = 9.2$ (t) $q = 5, C_R = 117, H = 6.2$

LibJPEG

Exemple



(a) $q = 5$ (b) $q = 10$

LibJPEG

Exemple

(c) $q = 15$

(d) $q = 30$

LibJPEG

Exemple

(e) $q = 40$

(f) $q = 60$

LibJPEG

Autres options utiles

- Choix du sous-échantillonage (`-sample`).
- Choix de la table de quantification (`-qtable`).
- Calcul de la DCT (`-dct`) : en virgule fixe ou flottante.
- JPEG progressive (`-progressive`).

Format MPEG

- MPEG est un format de compression vidéo.
- vidéo = succession d'images.
- Principe de la compression MPEG : on utilise un prédicteur temporel. Il prédit l'image suivante.
- On encode la différence entre l'image suivante et la prédiction à partir de l'image courante.
- Si la prédiction est bonne, cette différence sera proche de zéro.
- On compresse avec la méthode JPEG cette différence et la compression sera très efficace !
- On peut avoir des prédicteurs temporels plus ou moins sophistiqués selon les standards (MPEG1,MPEG2,MPEG4).