Encodage et compression d'images et videos.

Contact: damien.rohmer@cpe.fr

Sommaire

Images:
- Types d'images
- Format Vectoriel
- Format Bitmap
- Encodage images
- Compression sans pertes
- Compression avec pertes
- Comparaison d'images
- Cas du Jpeg

Video:
- Deplacement par blocs
- Interpolation d'images
- Méthode de recherche par blocs

Les types d'images

Image vectorielle

Michele Brami: inkscape

Image bitmap

http://mahamudras.blogspot.fr/2011/08/une-vache-en-or.html

Les types d'images

Image vectorielle	Image bitmap
+ Controle + Précision	+ Reproduction réelle + nbr degrés libertés
- Reproduction réelle - nbr degrés libertés	- Controle - Précision
ex. Inkscape Illustrator, InDesign	ex. Gimp Photoshop

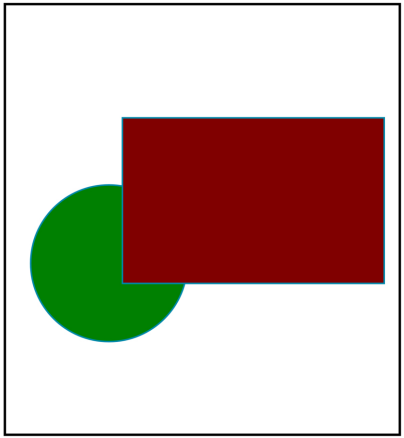
Images:

- Types d'images
- **Format Vectoriel**
- Format Bitmap
- Encodage images
- Compression sans pertes
- Compression avec pertes
- Comparaison d'images
- Cas du Jpeg

Video:

- Deplacement par blocs
- Interpolation d'images
- Méthode de recherche par blocs

Format d'image vectorielle



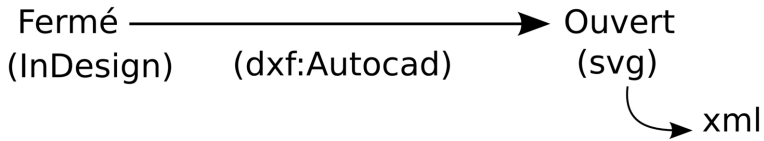
...

```
    inkscape:window-maximized="0" />
  <metadata
     id="metadata4">
    <rdf:RDF>
      <cc:Work
         rdf:about="">
        <dc:format>image/svg+xml</dc:format>
        <dc:type
           rdf:resource="http://purl.org/dc/dcmitype/StillImage" />
        <dc:title></dc:title>
      </cc:Work>
    </rdf:RDF>
  </metadata>
  <g
     inkscape:label="Layer 1"
     inkscape:groupmode="layer"
     id="layer1">
    <path
       sodipodi:type="arc"
       style="fill:#008000;fill-opacity:1;stroke:#0088aa;stroke-width:2;stroke-miterl
imit:4;stroke-opacity:1;stroke-dasharray:none;stroke-dashoffset:0"
       id="path5481"
       sodipodi:cx="282.85715"
       sodipodi:cy="440.93362"
       sodipodi:rx="100"
       sodipodi:ry="100"
       d="m 382.85715,440.93362 a 100,100 0 1 1 -200,0 100,100 0 1 1 200,0 z" />
    <rect
       style="fill:#800000;fill-opacity:1;stroke:#0088aa;stroke-width:2;stroke-miterl
imit:4;stroke-opacity:1;stroke-dasharray:none;stroke-dashoffset:0"
       id="rect5483"
       width="334.28571"
       height="211.42857"
       x="300"
       y="255.21933" />
  </g>
</svg>
```

Description textuelle/paramétrique

Format d'image vectorielle

Pas de format standard:
1 logiciel = 1 format



Souvent originaire format CAO

Format d'image vectorielle

Rem. Format très concis

(svg)

```
<path
   style="fill:#666666;stroke:none"
   d="m 464.13032,375.49958 5.20734,5.78416 -6.74689,-1.89962 z"
   id="path3992"
   inkscape:connector-curvature="0" />
<path
   inkscape:connector-curvature="0"
   id="path3994"
   d="m 454.69709,372.75113 5.20734,5.78416 -6.74689,-1.89962 z"
   style="fill:#666666;stroke:none" />
<path
   style="fill:#666666;stroke:none"
   d="m 444.59616,369.62225 5.20734,5.78416 -6.74689,-1.89962 z"
   id="path3996"
   inkscape:connector-curvature="0" />
<path
   inkscape:connector-curvature="0"
   id="path3998"
   d="m 435.00765,366.79616 5.20734,5.78416 -6.74689,-1.89962 z"
   style="fill:#666666;stroke:none" />
```

.svg.tar.gz

41Ko

(zoom)

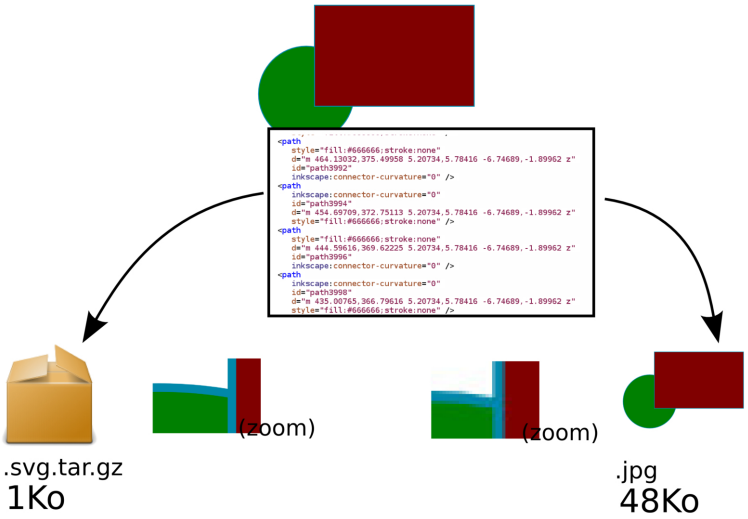
(zoom)

.jpg

162Ko

Format d'image vectorielle

Rem. Format très concis: dépend du contenu



Format d'image vectorielle

ex. Ecrire un manipulateur d'images vectorielle

ex. Mon propre format d'animation la plus concise!

mais:
- pas de lecteur
- forme limitées

Images:

- Types d'images
- Format Vectoriel
- **Format Bitmap**
- Encodage images
- Compression sans pertes
- Compression avec pertes
- Comparaison d'images
- Cas du Jpeg

Video:

- Deplacement par blocs
- Interpolation d'images
- Méthode de recherche par blocs

Format d'image bitmap

Description bas-niveau

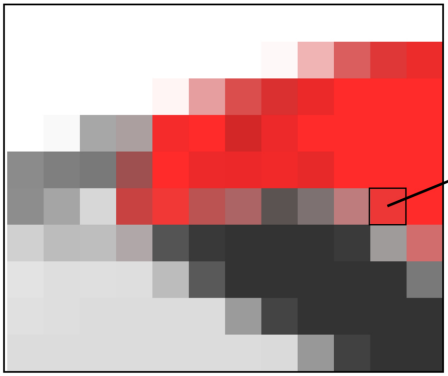


image bitmap=
tableau N_x x N_y de pixels

Format d'image bitmap

Avantage:

Peut représenter quasiment toutes les images
(taille donnée)

- Image HD standard:
  1920x1080 pixels

- 1 pixel => 256x256x256 couleurs possibles

Nombre d'images possibles:
34 789 235 097 600
=> beaucoup de libertés

Images:

- Types d'images
- Format Vectoriel
- Format Bitmap
- **Encodage images**
- Compression sans pertes
- Compression avec pertes
- Comparaison d'images
- Cas du Jpeg

Video:

- Deplacement par blocs
- Interpolation d'images
- Méthode de recherche par blocs

Format d'image bitmap

Inconvénients:

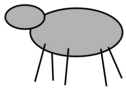
Controle quasi impossible pour un humain
Description lourde
Limité en résolution

Mais: Reste le format d'échange d'images fixe le plus standard

Stockage d'images bitmaps

Il existe plusieurs formats standards
(indépendant des logiciels)

Contraintes du format idéal:



- Léger
- Facile à lire/ecrire
- Rapide à lire/ecrire
- Preserve l'image originale
- Contient les informations nécessaires

Les formats standards bitmap

ex. d'image HD 1920x1080

Encodage brute en mémoire

```
struct couleur
{
  int rouge;
  int vert;
  int bleu;
};

class image_hd
{
  std::vector<couleur> pixels;
public:
  image_hd()
  {pixels.resize(1920*1080);}
};

int main()
{
  image_hd mon_image;

  return 0;
}
```

taille en RAM
$=N_x \times N_y \times$ sizeof(couleur)
$=1920 \times 1080 \times 12$
$=23.7$ Mo

valgrind:

```
==1297== Memcheck, a memory error detector
==1297== Copyright (C) 2002-2012, and GNU GPL'd, by Julian Seward et al.
==1297== Using Valgrind-3.8.1 and LibVEX; rerun with -h for copyright info
==1297== Command: ./a.out
==1297==
12
==1297==
==1297== HEAP SUMMARY:
==1297==     in use at exit: 0 bytes in 0 blocks
==1297==   total heap usage: 1 allocs, 1 frees, 24,883,200 bytes allocated
==1297==
==1297== All heap blocks were freed -- no leaks are possible
==1297==
==1297== For counts of detected and suppressed errors, rerun with: -v
```

OK

Rem. Pour une vidéo à 25 images/s =>600 Mo/s !!!

Les formats standards bitmap

ex. d'image HD 1920x1080

Encodage brute en mémoire

```
struct couleur
{
  int rouge;
  int vert;
  int bleu;
};

class image_hd
{
  std::vector<couleur> pixels;
public:
  image_hd()
  {pixels.resize(1920*1080);}
};

int main()
{
  image_hd mon_image;

  return 0;
}
```

Inutile de stocker au dela de 256

sauf pour l'HDR

Les formats standards bitmap

ex. d'image HD 1920x1080

Encodage brute en mémoire

```
struct couleur
{
  unsigned char rouge;
  unsigned char vert;
  unsigned char bleu;
};

class image_hd
{
  std::vector<couleur> pixels;
public:
  image_hd()
  {pixels.resize(1920*1080);}
};

int main()
{
  image_hd mon_image;

  return 0;
}
```

sizeof(couleur)=3

taille totale
=1920x1080x3
=6 Mo

```
6,220,800 bytes allocated
```

Rem. Pour une vidéo à 25 images/s => 150 Mo/s

Les formats standards bitmap

Stockage fichiers:

L'image mémoire:

pas standard

Nx	Ny	Donnees (r0 g0 b0 r1 ...)
1	1	6220800

pour image 1920x1080

~6Mo

Le standard le plus simple: ppm

(Portable PixMap)

ex.

```
P3
# Commentaires
3 3
255
255   0   0     0  90 255   222 255   0
255 255 255   127 236 129    25  25  24
177 162  90    47 128 143   170 128 143
```

expliquez ce format

Les formats standards bitmap

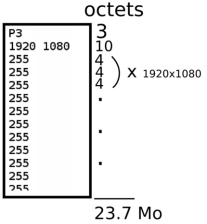
Avantage PPM:

- lisible humain (ASCII)
- indépendant endianess
- lecteur/ecriture triviale

Inconvénient PPM:

- Aussi volumineux qu'un mauvais codage
- Lecture/ecriture excessivement lent (ASCII)

ex. image blanche 1920x1080:



Les formats standards bitmap

PPM: Il existe différentes versions

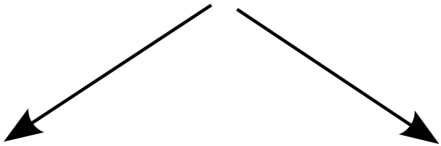
P1: n&b, ASCII
P2: gris, ASCII
P3: (r,g,b), ASCII

P4: n&b, binaire
P5: gris, binaire
P6: (r,g,b), binaire

```
P6
3 3
255
\377^@^@^@Z\377\336\377^@\377\37
7\377^?\354\201^Y^Y^X\261\242Z/\
200\217\252\200\217
```

Les formats standards bitmap

Pour être plus efficace, il faut compresser:



Compression sans pertes

png
tiff
...

Compressions avec pertes

jpeg
jpeg2000
...

Images:

- Types d'images
- Format Vectoriel
- Format Bitmap
- Encodage images
- → **Compression sans pertes**
- Compression avec pertes
- Comparaison d'images
- Cas du Jpeg

Video:

- Deplacement par blocs
- Interpolation d'images
- Méthode de recherche par blocs

Compression sans pertes

Rappel compression octets:

100100011110000001011111100011

ex.1 11201130416011105130 21

ex.2 12131111610111113 11

ex.3 112134611532

Rem: On compresse mieux si:
On a des motifs qui se répètent
On a des suites du même nombre

Idéale: de longues suites de 0

Compression sans pertes

Les grandes approches standards:

RLE
(Run Length Encoding)

(.bmp)

LZW
(Gunzip)

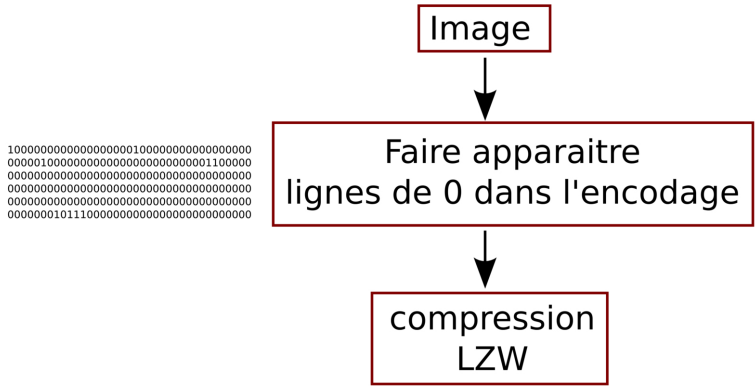
Dictionnaire + code

taille des mots variables
les + utilisés = les + petits

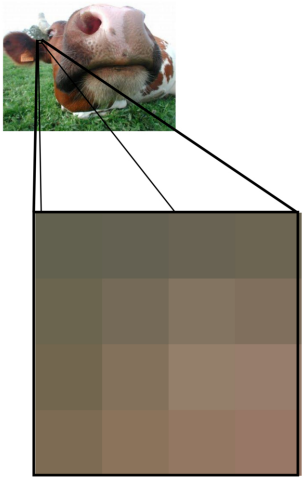
(.png)

Compression sans pertes

But pour une image:



Recherche de lignes de 0



98 97 79	100 97 82	105 99 83	108 101 82
107 101 79	117 106 88	131 116 97	128 111 93
114 102 78	131 113 91	148 127 106	151 125 108
125 107 83	139 115 91	147 119 98	152 119 102

bits/valeurs

3x1920x1080x8
=6 Mo

Quel est le codage/decodage?

encode:
98 107 114 125 2 5 3 10 14 -3 17 17 3 14 8 5
97 101 102 107 0 2 2 5 10 5 11 14 -2 8 4 0
79 79 78 83 3 1 -1 9 9 -4 13 15 2 8 7 4

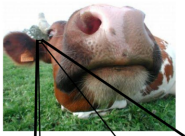
98 97 79	2 0 3	5 2 1	3 2 -1
107 101 79	10 5 9	14 10 9	-3 5 -4
114 102 78	17 11 13	17 14 15	3 -2 2
125 107 83	14 8 8	8 4 7	5 0 4

bits/valeurs

3x1080x8+3x1919x1079x5
=3.7 Mo

Recherche de lignes de 0

Utilisation d'un prédicteur



98 97 79	100 97 82	105 99 83	108 101 82
107 101 79	109 101 82	122 108 89	134 118 96
114 102 78	124 107 87	145 123 100	145 122 102
125 107 83	142 118 96	156 129 106	150 127 100

prédiction (type png)

98 97 79	100 97 82	105 99 83	108 101 82
107 101 79	117 106 88	131 116 97	128 111 93
114 102 78	131 113 91	148 127 106	151 125 108
125 107 83	139 115 91	147 119 98	152 119 102

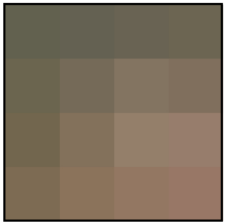
bits/valeurs

3x1920x1080x8
=6 Mo

98 97 79	2 0 3	5 2 1	3 2 1
9 4 0	8 5 6	9 8 8	6 -7 -3
7 1 -1	7 6 4	3 4 6	6 3 6
11 5 5	-3 -3 -5	-9 -10 -8	2 -8 2

3x(4x8+(1079+1919)x4
+1079x1919x4)
=2.9 Mo

Recherche de lignes de 0



98 97 79	100 97 82	105 99 83	108 101 82
107 101 79	117 106 88	131 116 97	128 111 93
114 102 78	131 113 91	148 127 106	151 125 108
125 107 83	139 115 91	147 119 98	152 119 102

98 97 79	2 0 3	5 2 1	3 2 1
9 4 0	8 5 6	9 8 8	6 -7 -3
7 1 -1	7 6 4	3 4 6	6 3 6
11 5 5	-3 -3 -5	-9 -10 -8	2 -8 2

Avantages:

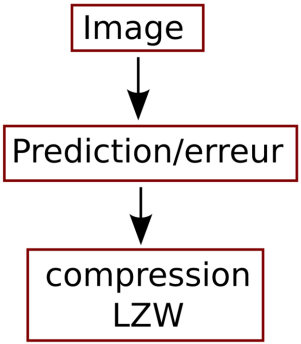
- Gain direct de place possible
- Beaucoup de (faibles) nombres identiques => LZW très efficace
- Encodage/decodage rapide
- Pas d'utilisation mémoire

Inconvénient:

- Perte de robustesse (propagation d'erreur)

Format PNG

(Portable Network Graphics)



Format PNG

+ Gestion canal alpha
+ Options nombre couleurs
+ Standard/répendu/reconnu

Commence toujours par:

```
8950 4e47 0d0a 1a0a 0000 000d 4948 4452  .PNG........
0000 0785 0000 03db 0806 0000 005b 8d5d  .........[.]
a500 0000 0473 4249 5408 0808 087c 0864  .....sBIT....d
8800 0000 0970 4859 7300 001e c300 001e  .....pHYs.....
c301 bc97 97da 0000 0019 7445 5874 536f  ..........tEXtSo
```

Autres formats

BMP: RVB
Encodage RLC
uniquement MS

ancien, lourd

GIF: 256 couleurs uniquement
permet animation

faible qualité
encore présent sur internet
(gif animé)

PCX: 256 couleurs uniquement
Encodage RLC
Painbrush uniquement

TGA: RVB
Encodage RLC

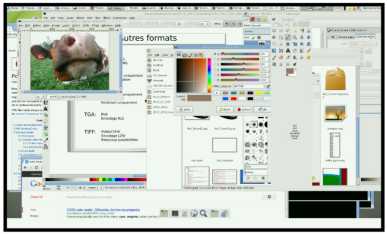
Plutôt artistes sous Windows

TIFF: RVBA/CMYK
Diverses encodage/compression
(none, LZW, Deflate, jpeg)
Beaucoup possibilitées

Très complexe, applications multiples
(médical, 3D, imprimeur)
Format professionnel

Comparaisons formats

image test:
1920x1080



png: 0.64 Mo
tiff: 0.87 Mo
tga: 1.3 Mo

jpeg: 0.45Ko

Images:

- Types d'images
- Format Vectoriel
- Format Bitmap
- Encodage images
- Compression sans pertes
- **Compression avec pertes**
- Comparaison d'images
- Cas du Jpeg

Video:

- Deplacement par blocs
- Interpolation d'images
- Méthode de recherche par blocs

Méthode directe: perte definition spatiale

860x645

1.1Mo

430x323

313Ko

215x162

79Ko

108x81

21Ko

jpeg: 84Ko

Méthode directe: perte definition couleur

	256	128	64	32
troncature	 277Ko	 258Ko	 239Ko	 194Ko
clusturing	 312Ko	 245Ko	 194Ko	 131Ko

jpeg: 84Ko 

Comment améliorer?

Approches + évoluées ?

Comment comparer ?

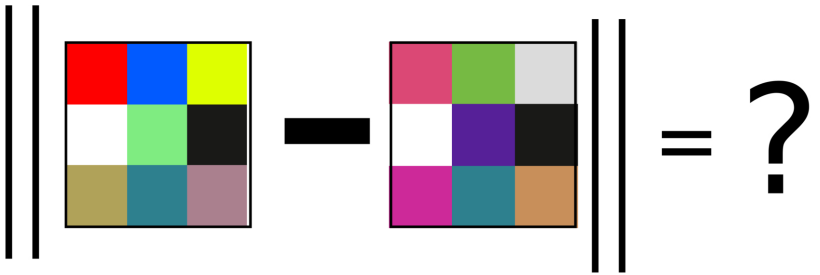
Images:

- Types d'images
- Format Vectoriel
- Format Bitmap
- Encodage images
- Compression sans pertes
- Compression avec pertes
- → **Comparaison d'images**
- Cas du Jpeg

Video:

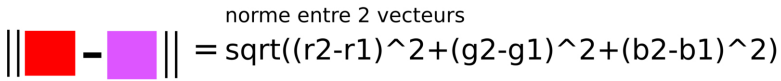
- Deplacement par blocs
- Interpolation d'images
- Méthode de recherche par blocs

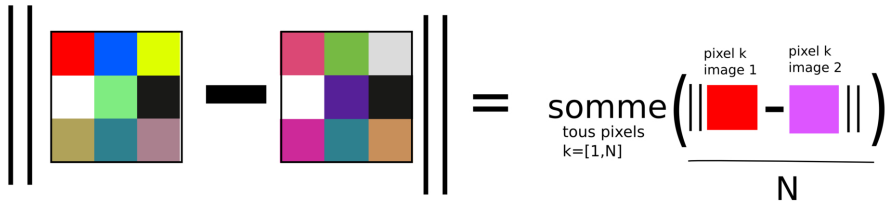
Comparer 2 images



Comparer 2 images

Approche standard: norme vectorielle





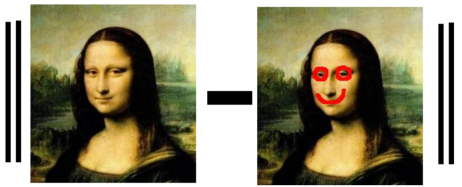
Comparer 2 images

$$\left\| \text{  } - \text{  } \right\| = 0$$

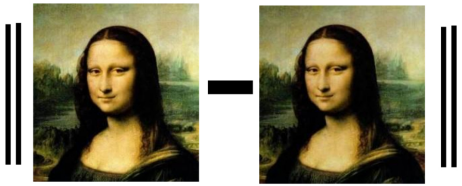
$$\left\| \text{  } - \text{  } \right\| = 33$$

$$\left\| \text{  } - \text{  } \right\| = 155$$

Attention: Non perceptuel !



$= 3$



$= 55$

Il existe mieux:

Couleur, déformation, multiresolution, ...

Images:

- Types d'images
- Format Vectoriel
- Format Bitmap
- Encodage images
- Compression sans pertes
- Compression avec pertes
- Comparaison d'images
- → **Cas du Jpeg**

Video:

- Deplacement par blocs
- Interpolation d'images
- Méthode de recherche par blocs

Approches plus avancées

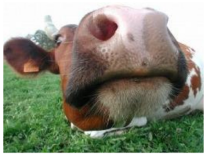
Oeil + sensible contraste N&B

Qu'aux couleurs

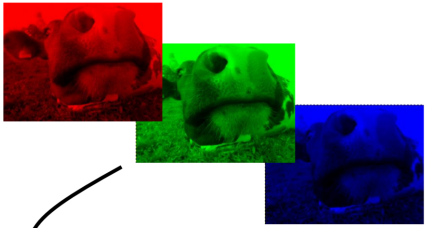
Idée: Séparer composante luminance N&B des informations de chrominance

Puis sous échantillonner chrominance

Approches plus avancées

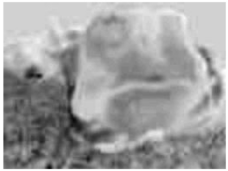


=

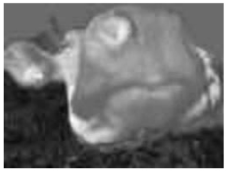


Y

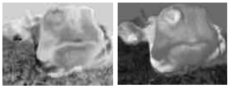
Cb



Cr

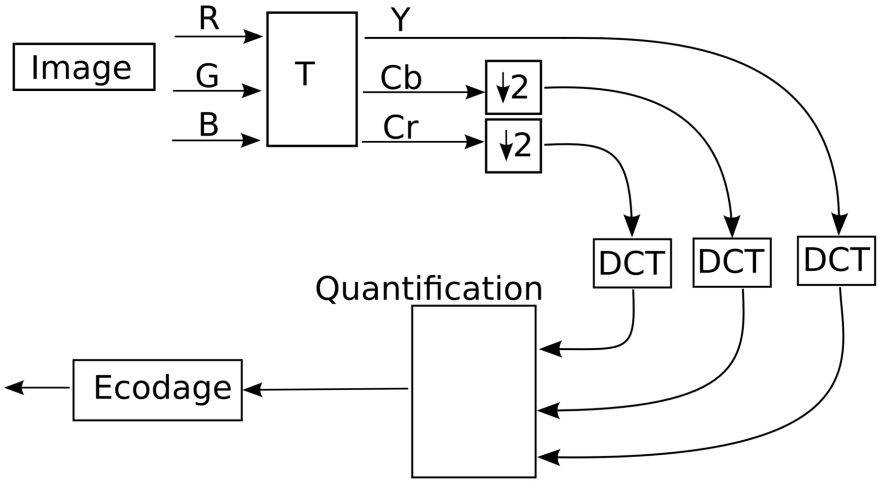


/2



gain: 50% !

JPEG



JPEG: DCT

Discrete Cosine Transform

= Transformée de Fourier (suivant cos uniquement)

bloc 8x8

DCT

$$\sum_{x=0}^{7}\sum_{y=0}^{7}\alpha(u)\alpha(v)g_{x,y}\cos\left[\frac{\pi}{8}\left(x+\frac{1}{2}\right)u\right]\cos\left[\frac{\pi}{8}\left(y+\frac{1}{2}\right)v\right]$$

wikipedia

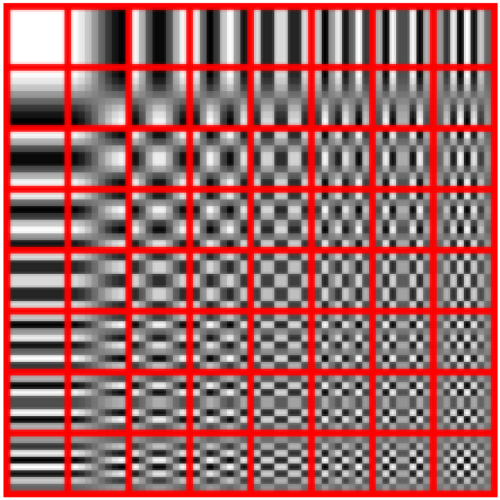
8x8 coefficients

$$\begin{bmatrix}
6.1917 & -0.3411 & 1.2418 & 0.1492 & 0.1583 & 0.2742 & -0.0724 & 0.0561 \\
0.2205 & 0.0214 & 0.4503 & 0.3947 & -0.7846 & -0.4391 & 0.1001 & -0.2554 \\
1.0423 & 0.2214 & -1.0017 & -0.2720 & 0.0789 & -0.1952 & 0.2801 & 0.4713 \\
-0.2340 & -0.0392 & -0.2617 & -0.2866 & 0.6351 & 0.3501 & -0.1433 & 0.3550 \\
0.2750 & 0.0226 & 0.1229 & 0.2183 & -0.2583 & -0.0742 & -0.2042 & -0.5906 \\
0.0653 & 0.0428 & -0.4721 & -0.2905 & 0.4745 & 0.2875 & -0.0284 & -0.1311 \\
0.3169 & 0.0541 & -0.1033 & -0.0225 & -0.0056 & 0.1017 & -0.1650 & -0.1500 \\
-0.2970 & -0.0627 & 0.1960 & 0.0644 & -0.1136 & -0.1031 & 0.1887 & 0.1444
\end{bmatrix}$$

JPEG: DCT

Discrete Cosine Transform

= décomposition couleur suivant cos



JPEG: Quantification

basse fréquences

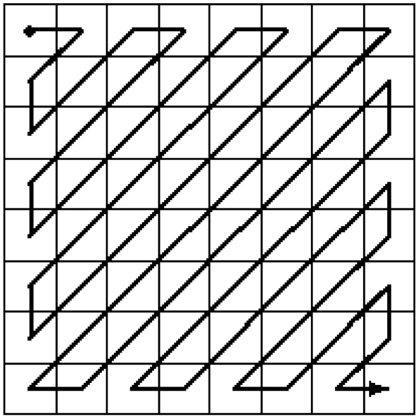
$$Q = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}$$

hautes fréquences horizontale

hautes fréquences verticales

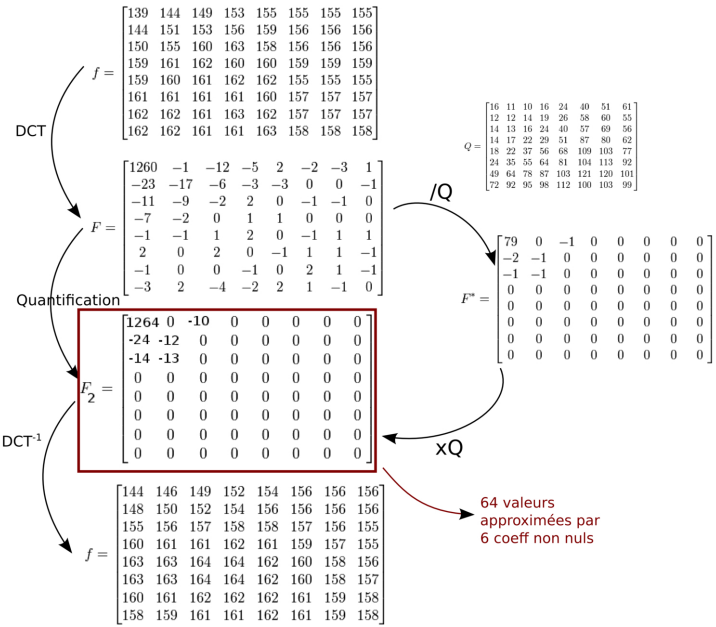
JPEG: Codage

Encodage RLZ



Suivit codage (Huffmann)

JPEG: Exemple



JPEG : Taux de compression

Choix du taux de compression:

Matrice de quantification

100%
460 Ko

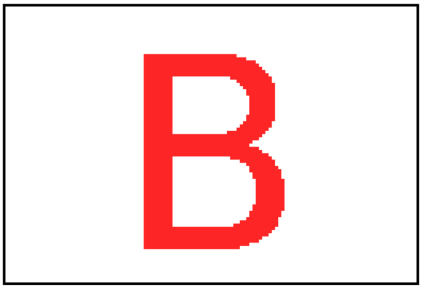
80%
117 Ko

50%
65 Ko

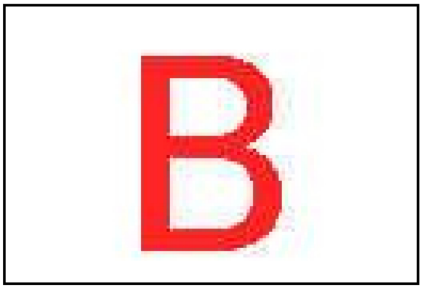
30%
45 Ko

10%
19 Ko

JPEG : toujours meilleur?



.png
469 octets



.jpg
1700 octets

.jpg => images naturelles
Il existe plus adapté pour les images non naturelles

JPEG : artefacts

block:
8x8

DCT

JPEG : mieux?

jpeg2000:
fait mieux que jpeg + options, bien que reste moins répendue

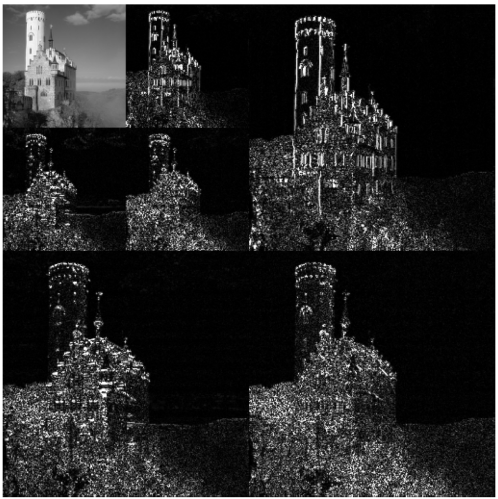
jpg

jpg2000

flou plutôt qu'apparition d'artefacts

JPEG 2000

Basée ondelettes:



Images:

- Types d'images
- Format Vectoriel
- Format Bitmap
- Encodage images
- Compression sans pertes
- Compression avec pertes
- Comparaison d'images
- Cas du Jpeg

→ **Video**:

- Deplacement par blocs
- Interpolation d'images
- Méthode de recherche par blocs

Video: débit

Si transmet images par images

débit admissible: 4Mo/s

25 images/s => 160Ko/images max

Si on veut meilleur qualitée: approche temporelle

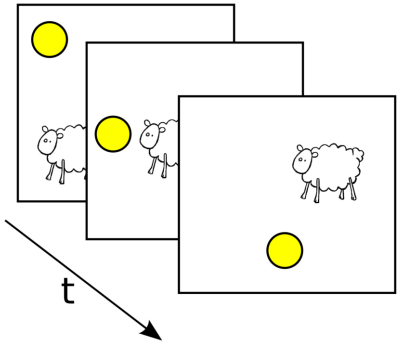
Images:

- Types d'images
- Format Vectoriel
- Format Bitmap
- Encodage images
- Compression sans pertes
- Compression avec pertes
- Comparaison d'images
- Cas du Jpeg

Video:

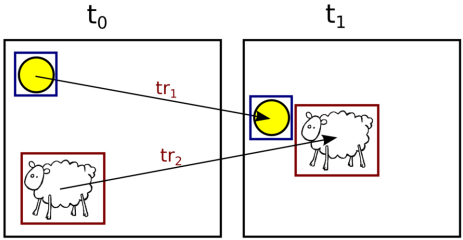
- → **Deplacement par blocs**
- Interpolation d'images
- Méthode de recherche par blocs

Video: deplacement de blocs



recherche de blocs
qui se ressemblent

transmet
translation, pas le bloc



Video: deplacement de blocs

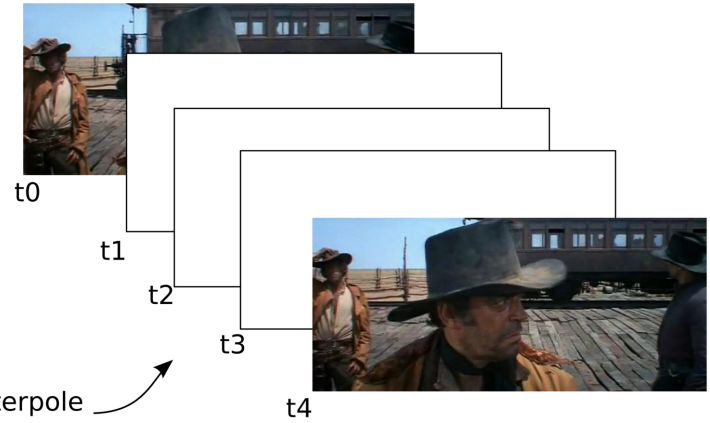
Typiquement:
Recherche bloc 8x8 → jpeg!

Images:
- Types d'images
- Format Vectoriel
- Format Bitmap
- Encodage images
- Compression sans pertes
- Compression avec pertes
- Comparaison d'images
- Cas du Jpeg

Video:
- → **Deplacement par blocs**
- Interpolation d'images
- Méthode de recherche par blocs

Video: interpolation d'images

Encode 1 image sur N



$I(t) = w\, I(t0) + (1-w)\, I(t1)$

Video: interpolation d'images

Fonctionne uniquement
si peu de changements

I(t1)

3/4 I(t1)
+1/4 I(t2)

1/2 I(t1)
+1/2 I(t2)

1/4 I(t1)
+3/4 I(t2)

I(t2)

Images:
- Types d'images
- Format Vectoriel
- Format Bitmap
- Encodage images
- Compression sans pertes
- Compression avec pertes
- Comparaison d'images
- Cas du Jpeg

Video:
- Deplacement par blocs
- Interpolation d'images
- **Méthode de recherche par blocs**

Méthode de recherche de blocs

Algorithme brute:

Pour tous les blocs 8x8 de I(t1)
  Pour tous les blocs de I(t2)
    si $\| \text{bloc}_{t1} - \text{bloc}_{t2} \| < e$
    Alors
      Transmet translation
  Transmet nouveau bloc

Pour une image:
1920x1080:

en moyenne:
30 000 000 comparaison

=> 2 150 000 000 000 calculs

complexité en O(N^2)
N=nbr pixels

Méthode de recherche de blocs

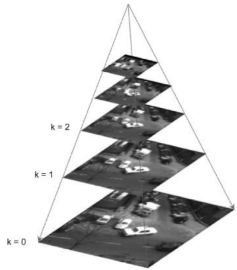
Amélioration "gratuite":

Commencer la recherche autour du bloc original

=> animation continue la plupart du temps!

Méthode de recherche de blocs

Méthode multiresolution :



Méthode flux optique :

Les formats standards

Conteneurs

Video compressée
Audio compressée (mp3, ...)
Sous-titres
...

AVI
MP4
FLV
QuickTime
Ogg
Matroska
DivX
WMW

Compression

H.263
H.264
MPEG-1
MPEG-2
MPEG-4
AVC

Outils disponibles

Convertir video vers .png :

```
$ mplayer -nosound -vo png:outdir=<directory> <video>.ogv
```

Convertir conteneur vers avi:

```
$ mencoder -idx <video_in>.ogv -ovc raw -oac mp3lame -o <video_out>.avi
```

Encoder en mp4:

```
$ ffmpeg -i <video_in>.avi -vcodec mpeg4 -b 1500k -y <video_out>.mp4
```

Convertir png en mp4 :

```
$ ffmpeg -r 25 -f image2 -i <picture>%0nd.png -vcodex mpeg4 -b 1500k -y <video_out>.mp4
```

nbr de zeros