# Quantification vectorielle LBG

Axel Benadiba
Côme Larger
Stéphane Bajou

TELECOM SudParis
IP PARIS

# Intérêt de l’algorithme Linde-Buzo-Gray (LBG)

## Compression d’images :

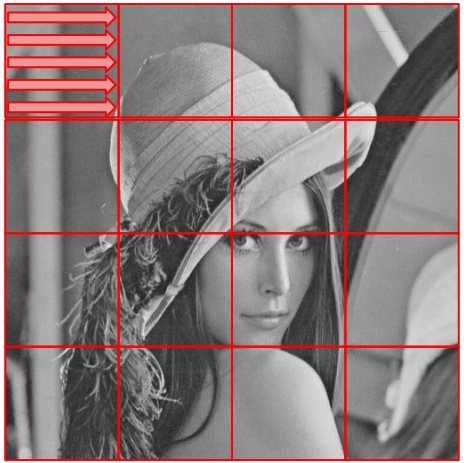
- Permet de réduire la taille des images.
- Représentation des données de l’image avec un nombre limité de vecteurs (“codes”).
- Conception efficace d’un codebook avec un minimum de distorsion d’erreur

## Classification et segmentation d’images :

- Vecteurs de code → Classer les pixels en groupe similaires.
- Facilite l’identification de différentes régions d’image.
- Reconnaissance d’objet et détection de frontières.

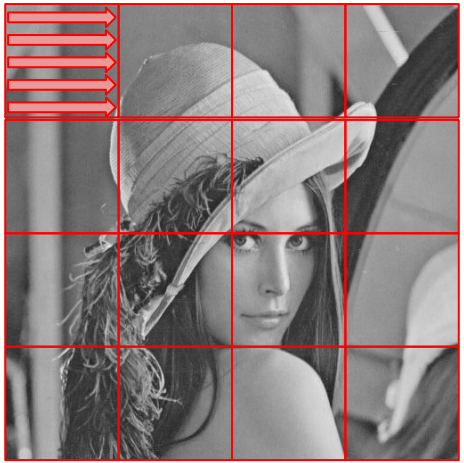
# L’algorithme Linde - Buzo - Gray

Le codebook est construit à partir d’un ensemble d’images d’apprentissage. On décompose chaque image en blocs carrés de dimensions **NxN**. On met ensuite chaque bloc sous la forme de vecteur de dimension $N^2$, obtenu en balayant *raster-scan* ses pixels. On obtient alors des vecteurs comprenant $N^2$ valeurs comprises entre 0 et 255 (niveau de gris)



# L’algorithme Linde - Buzo - Gray

**Etape 1** : Déterminer tout d’abord le dictionnaire composé d’un seul vecteur prototype, noté $\mathbf{C_0^0}$ . Le critère étant la minimisation de l’erreur quadratique moyenne, ce vecteur sera le centre de gravité de l’ensemble des blocs de l’image.



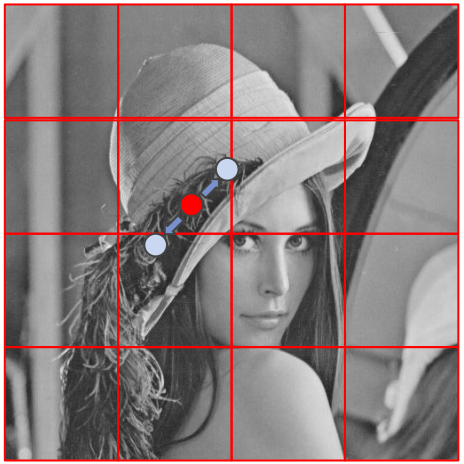
# L’algorithme Linde - Buzo - Gray

**Etape 2** : Partager ce vecteur en deux vecteurs, notés $C_0^1$ et $C_1^1$ , donnés par :

$$C_0^1 = C_0^0 + \varepsilon$$

$$C_1^1 = C_0^0 - \varepsilon$$

où ε est un *vecteur aléatoire de perturbation*, suffisamment petit.



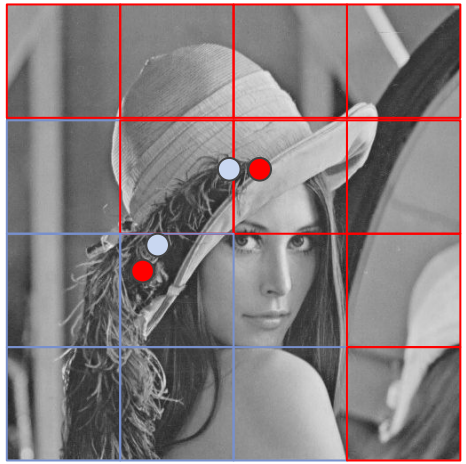
# L’algorithme Linde - Buzo - Gray

**Etape 3** : On classe tous les vecteurs de la base d’apprentissage relativement à ces deux vecteurs en utilisant le critère de la distance euclidienne minimale.

$$||A||_F = [\sum_{i,j} abs(a_{i,j})^2]^{1/2}$$

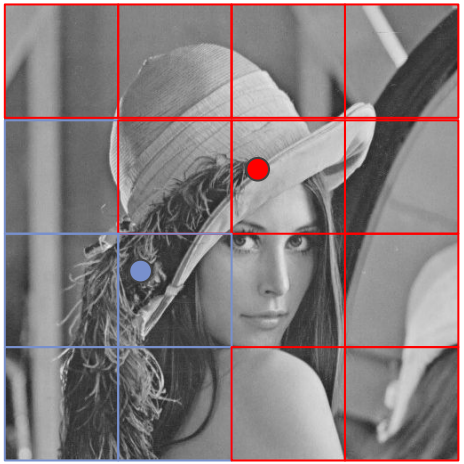
# L’algorithme Linde - Buzo - Gray

**Etape 4.A** : On calcule les centres de gravité des deux classes ainsi obtenues qui remplaceront les anciens prototypes $C_0^1$ et $C_1^1$.



# L’algorithme Linde - Buzo - Gray

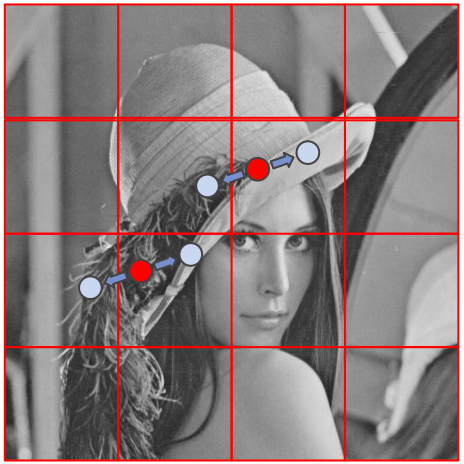
**Etape 4.B** : On réitère le procédé jusqu’à ce que la décroissance de la distorsion moyenne devienne inférieure à un seuil pré-défini.



Distorsion moyenne : moyenne des distances euclidiennes carrées de tous les vecteurs à leurs centres respectifs

# L’algorithme Linde - Buzo - Gray

**Etape 5** : Partition de chaque nouveau vecteur obtenu en deux, conformément à l’étape 2 et réitération de l’étape 3. L’algorithme s’arrête lorsque le nombre désiré de vecteurs est obtenu.



# Résultats attendus

Image initiale
Taille : 512 x 512

Bloc : 4 x 4
Codebook_size : 8

Bloc : 4 x 4
Codebook_size : 1024

Source : LBG/README.md at master · droidadroit/LBG (github.com)

# Dataset test

# Premiers résultats obtenus

Image : 512 x 512
Bloc : 16 x 16
Database : 10 img
Taille CB : 64

Image : 512 x 512
Bloc : 8 x 8
Database : 10 img
Taille CB : 64

# Base de données utilisée

Le choix de la base de données d'entraînement influence directement l'efficacité du **codebook**. Si l'on souhaite quantifier des images d'un certain genre (par exemple des images d'animaux), alors il sera plus pertinent d'avoir une base de donnée constitué majoritairement d'images d'animaux, car le dictionnaire ainsi créée comportera des motifs propres aux animaux.

**Base de donnée choisie** : 16 images d'êtres vivants divers, en 512x512 pixels.

# Base de données utilisée

# Hyperparamètres utilisés

Le temps de calcul du **codebook** peut être très élevé, il convient donc de choisir un jeu d'hyperparamètres approprié à notre application. Les paramètres premièrement utilisés sont les suivants :

- Taille de blocs **NxN** : 32x32. Les images étant de taille 512x512 pixels, on obtient 256 blocs par image. Nous disposons de 16 images, soit un total de 4096 blocs. Chaque bloc étant représenté par un vecteur, **notre pool est donc composé de 4096 vecteurs.**
- Seuil de décroissance de la distorsion moyenne : **0.1**
- Nombre de vecteurs prototypes voulu pour le codebook : **512**

# Complexité algorithmique de LBG

L'étape prenant la majorité du temps dans l'algorithme est l'**étape 3**, consistant à la classification des vecteurs de notre pool, relativement aux vecteurs prototypes.

**Classification = calculs de distance euclidienne.**

Après l'étape 2 : 2 vecteurs prototypes. Pool : 4096 vecteurs => 8192 calculs de distance/étape de convergence.

**~5 étapes de convergence** pour la distorsion moyenne **=> plus de 40000 calculs de distance pour passer de 2 à 4 vecteurs prototypes.**

# Complexité algorithmique de LBG

Doubler le codebook (**étape 2**) **=>** Doubler le nombre de calculs de distances à effectuer.

Nombre moyen d'étape de convergence ~ 5 (optimiste)

Estimation du nombre **c** de calculs de distance à effectuer au minimum pour obtenir un codebook de taille **$n=2^k$**, en utilisant des blocs de taille **NxN** sur **K** images de taille **MxM** :

$$\sum_{i=1}^{k} 5L \times 2^i = 5L \times (2 \times (2^k - 1)) = 2 \times 5L \times (n - 1)$$

$L = \frac{K \times M \times M}{N \times N}$ : Nombre de vecteurs contenus dans notre pool.

**Complexité** : O(K) ; $O(M^2)$ ; $O(1/(N^2))$ ; O(n).

# Conclusion

Avec les hyperparamètres choisis, on estime le nombre minimal de calculs de distance à effectuer **c=5*4096*2*(512-1)=20 930 560.** Durée de calcul : **plus de 12 heures.**

Utiliser des blocs de taille **16x16** multiplierait par **4** le nombre de vecteurs du pool, et multiplierait également par **4** la durée de l'algorithme.

**CCL** : Algorithme puissant pour la quantification vectorielle, pratique à utiliser une fois que le codebook est établi. En revanche, le codebook peut être extrêmement long à se construire, même en utilisant une taille de blocs assez grande, problème dû à la complexité en $O(1/N^2)$.