La transformée en ondelettes discrète dans le tatouage des images numériques

FATIMA EZ-ZAHRA HILALI
32304

Motivation

https://medium.com/

À l’ère numérique, l’accès aux œuvres d’art, de toutes ses formes, s’est démocratisé à une vitesse sans précédent. Souvent, ces œuvres sont utilisées, modifiées ou revendues sans autorisation, au mépris des droits de leurs auteurs.

www.techpaf.com

- Le watermarking s’impose comme une solution pour répondre à ce problème.

- La transformée en ondelettes discrète sera l’outil principal analysé dans cette étude. Pour simplifier les calculs et l’étude, on se focalise sur les ondelettes de Haar.

Problématique:

Comment peut-on réaliser le tatouage numérique des images numériques en utilisant la transformée en ondelettes discrète ?

Plan

1.Les ondelettes de Haar : quelques propriétés
2. La transformée en ondelettes discrète, et son inverse: signal 1-D
3. Généralisation pour un signal bidimensionnel
4. Algorithme adopté pour effectuer le tatouage numérique
5. Implémentation en python et évaluation des résultats.

1
Les ondelettes de Haar

Les ondelettes de Haar

- *L'ondelette mère:*

$$\psi(x) = \begin{cases} -1 & \text{si } 0 \leq x < \frac{1}{2}, \\ 1 & \text{si } \frac{1}{2} \leq x < 1, \\ 0 & \text{sinon.} \end{cases}$$

- *L'ondelette père (fontion d'échelle)*

$$\phi(t) = \begin{cases} 1 & \text{si } 0 \leq t < 1 \\ 0 & \text{sinon.} \end{cases}$$

Quelques propriétés

Relation entre ψ et ϕ

$$\frac{1}{\sqrt{2}}\phi\left(\frac{t}{2}\right) = \sum_{n=-\infty}^{+\infty} h[n]\phi(t-n),$$

$$\text{avec} \quad h[n] = \left\langle \frac{1}{\sqrt{2}}\phi\left(\frac{t}{2}\right), \phi(t-n) \right\rangle.$$

avec $\langle f, g\rangle = \int_{-\infty}^{+\infty} f(t)g^*(t)\,dt$

h[n] est un filtre discret passe-bas

Pour les ondelettes de Haar

$$h[n] = \begin{cases} \frac{1}{\sqrt{2}} & \text{si } n = 0, 1 \\ 0 & \text{sinon} \end{cases}$$

$$\frac{1}{\sqrt{2}}\psi\left(\frac{t}{2}\right) = \sum_{n=-\infty}^{+\infty} g[n]\phi(t-n)$$

$$\text{avec} \quad g[n] = \frac{1}{\sqrt{2}}\left\langle \psi\left(\frac{t}{2}\right), \phi(t-n) \right\rangle.$$

g[n] est un filtre discret passe-haut

Pour les ondelettes de Haar

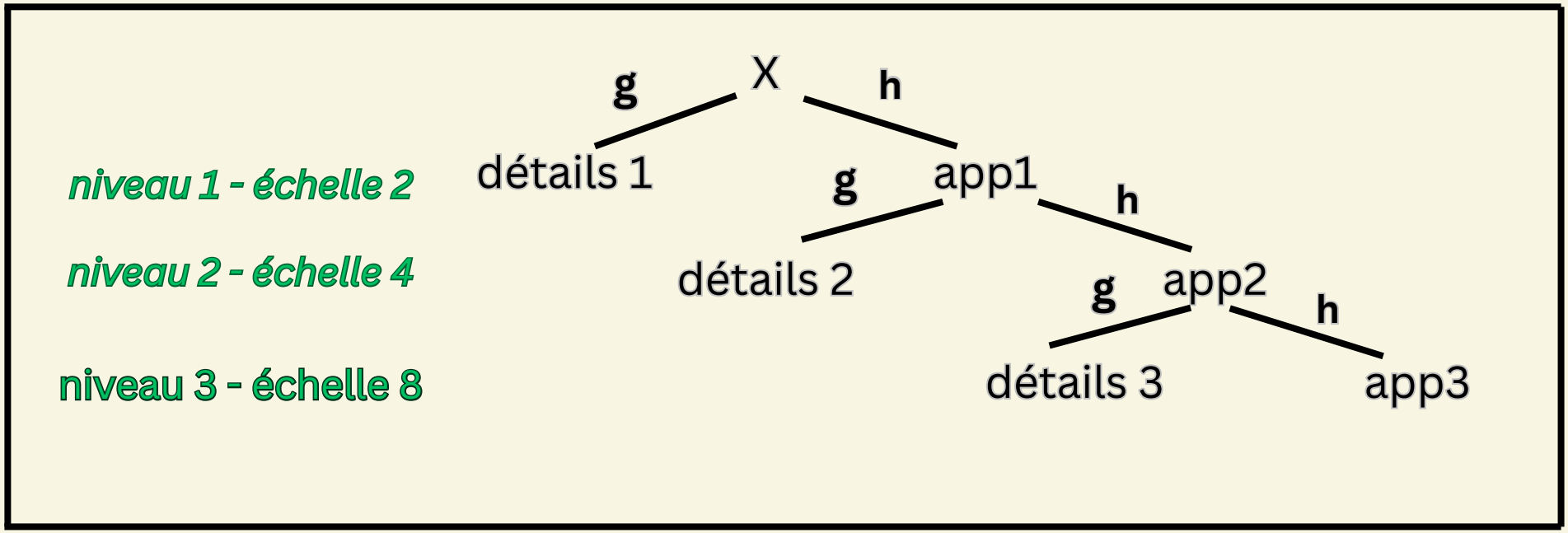
$$g[n] = \begin{cases} \frac{-1}{\sqrt{2}} & \text{si } n = 0 \\ \frac{1}{\sqrt{2}} & \text{si } n = 1 \\ 0 & \text{sinon} \end{cases}$$

2
La transformée en ondelettes discrète et son inverse: signal 1-D

DWT pour un signal 1-D

Soit le signal discret: X={x[O],x[1],.....,x[N-1]} de longueur N= 2^J

La DWT est la donnée d' un filtrage successif par h[n] (passe-bas) et g[n] (passe-haut), suivi d'un sous-échantillonnage d'un facteur 2, et ne gardant qu'une valeur sur 2.



DWT pour un signal 1-D

Sachant que le filtrage se définit par la convolution suivante :

$$s(n) = \sum_{m=-\infty}^{\infty} x(m)H(n-m)$$

avec H[n] un filtre discret

la transformée en ondelettes de Haar discrète au niveau j et à l'échelle 2^j se calcule par:

$$c_{j+1,k} = \frac{1}{\sqrt{2}}(c_{j,2k} + c_{j,2k+1}), \quad 0 \leq k \leq 2^{J-(j+1)} - 1 = N/2^{j+1} - 1,$$

$$d_{j+1,k} = \frac{1}{\sqrt{2}}(-c_{j,2k} + c_{j,2k+1}), \quad 0 \leq k \leq 2^{J-(j+1)} - 1 = N/2^{j+1} - 1$$

c est le coefficient d'ondelettes relatifs à l'approximation, et **d** le coefficient d'ondelettes relatifs aux détails.

La transformée en ondelettes discrète

En supposant que $c_{0,k} = x[k]$

les ondelettes de Haar simplifient le calcul de reconstruction.
A l'échelle 2^{j+1} , les coefficients des ondelettes à l'échelle 2^j sont retrouvés par:

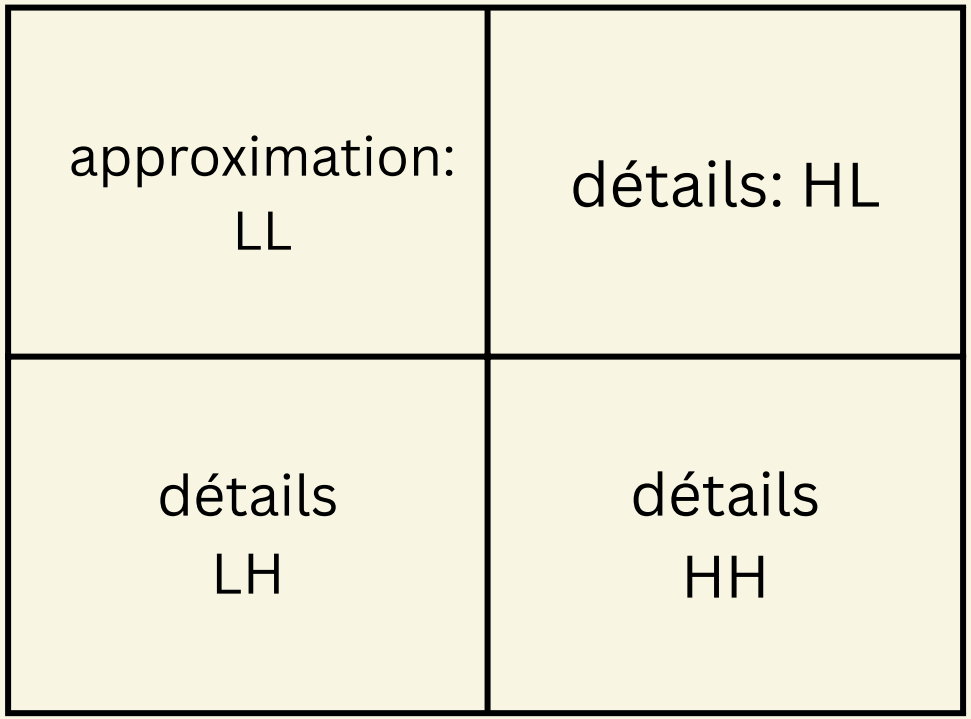
$$c_{j,k} = \frac{1}{\sqrt{2}}\left(c_{j+1,k/2} - d_{j+1,k/2}\right) \quad \text{pour } k \text{ pair et } 0 \leq k \leq 2^{J-j} - 1,$$

$$c_{j,k} = \frac{1}{\sqrt{2}}\left(c_{j+1,(k-1)/2} + d_{j+1,(k-1)/2}\right) \quad \text{pour } k \text{ impair et } 0 \leq k \leq 2^{J-j} - 1.$$

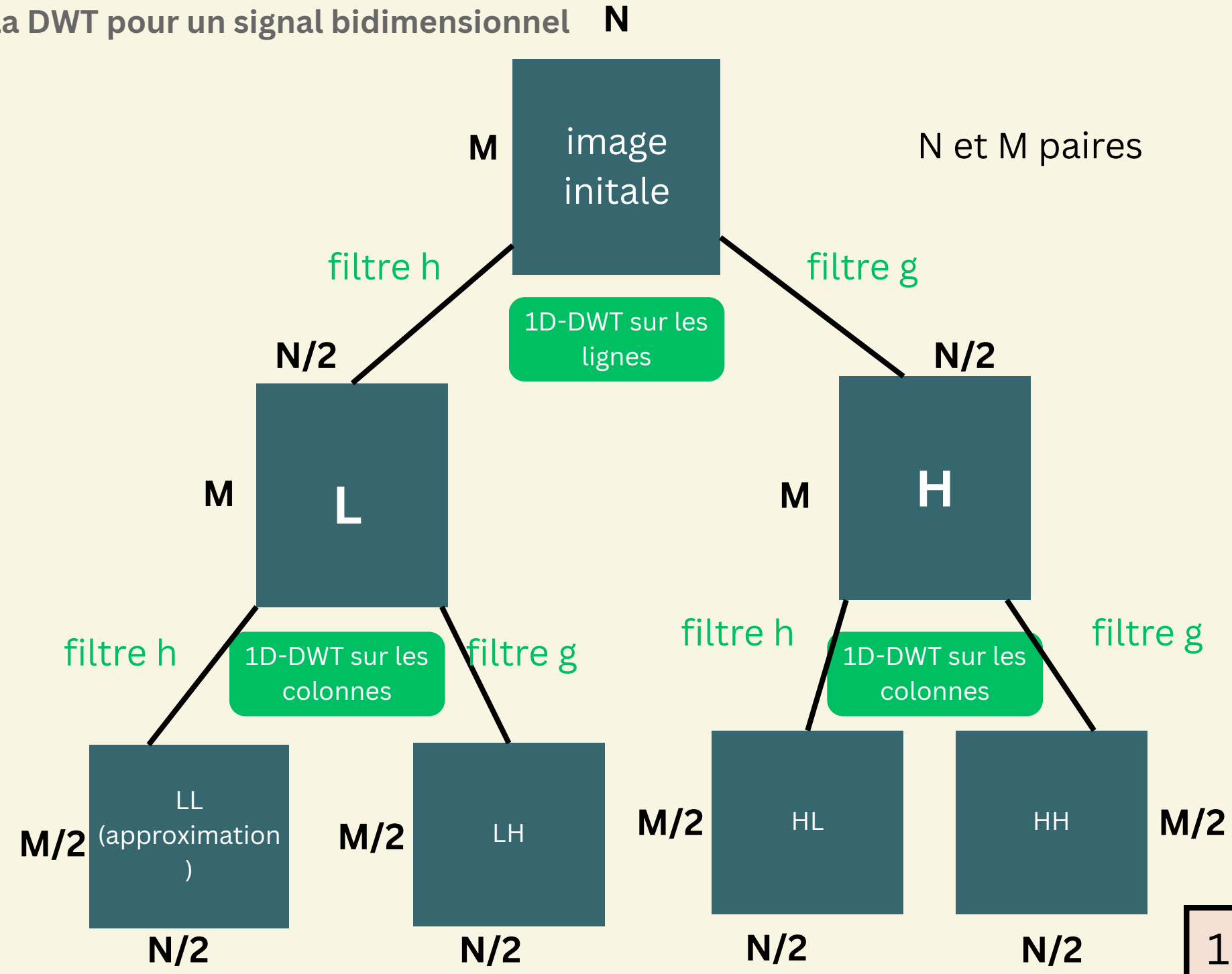
Généralisation pour les signaux 2-D

La DWT pour un signal bidimensionnel

La généralisation de la DWT à un signal 2D se fait en appliquant la DWT séparément et successivement sur les lignes et les colonnes, décomposant ainsi l'image en 4 sous-bandes: une d'approximation et trois de détails.



La DWT pour un signal bidimensionnel



Le calcul, pour les ondelettes, de Haar fait que:

1. DWT sur les lignes :

$$\text{lignes\_basses}[i, \frac{j}{2}] = \frac{1}{\sqrt{2}} \cdot \text{image}[i, j] + \frac{1}{\sqrt{2}} \cdot \text{image}[i, j+1]$$

$$\text{lignes\_hautes}[i, \frac{j}{2}] = \frac{1}{\sqrt{2}} \cdot \text{image}[i, j] - \frac{1}{\sqrt{2}} \cdot \text{image}[i, j+1]$$

2. DWT sur les colonnes :

$$LL\left[\frac{i}{2}, j\right] = \frac{1}{\sqrt{2}} \cdot \text{lignes\_basses}[i, j] + \frac{1}{\sqrt{2}} \cdot \text{lignes\_basses}[i+1, j]$$

$$LH\left[\frac{i}{2}, j\right] = \frac{1}{\sqrt{2}} \cdot \text{lignes\_basses}[i, j] - \frac{1}{\sqrt{2}} \cdot \text{lignes\_basses}[i+1, j]$$

$$HL\left[\frac{i}{2}, j\right] = \frac{1}{\sqrt{2}} \cdot \text{lignes\_hautes}[i, j] + \frac{1}{\sqrt{2}} \cdot \text{lignes\_hautes}[i+1, j]$$

$$HH\left[\frac{i}{2}, j\right] = \frac{1}{\sqrt{2}} \cdot \text{lignes\_hautes}[i, j] - \frac{1}{\sqrt{2}} \cdot \text{lignes\_hautes}[i+1, j]$$

IDWT pour un signal 2-D

1. **IDWT sur les colonnes, pour récupérer les lignes basses et hautes**

$$\text{lignes\_basses}[2i, j] = \frac{LL[i, j] + LH[i, j]}{\sqrt{2}}$$

$$\text{lignes\_basses}[2i + 1, j] = \frac{LL[i, j] - LH[i, j]}{\sqrt{2}}$$

$$\text{lignes\_hautes}[2i, j] = \frac{HL[i, j] + HH[i, j]}{\sqrt{2}}$$

$$\text{lignes\_hautes}[2i + 1, j] = \frac{HL[i, j] - HH[i, j]}{\sqrt{2}}$$

2. 1D-IDWT (sur les lignes)

$$\text{image}[i, 2j] = \frac{1}{\sqrt{2}} \cdot \text{lignes\_basses}[i, j] + \frac{1}{\sqrt{2}} \cdot \text{lignes\_hautes}[i, j]$$

$$\text{image}[i, 2j+1] = \frac{1}{\sqrt{2}} \cdot \text{lignes\_basses}[i, j] - \frac{1}{\sqrt{2}} \cdot \text{lignes\_hautes}[i, j]$$

Pour passer aux autres niveaux de DWT pour une image (2-D), on réitère sur la sous-bandes LL

DWT à 1 niveau appliquer sur l’image (512x512) suivante, en niveaux de gris :

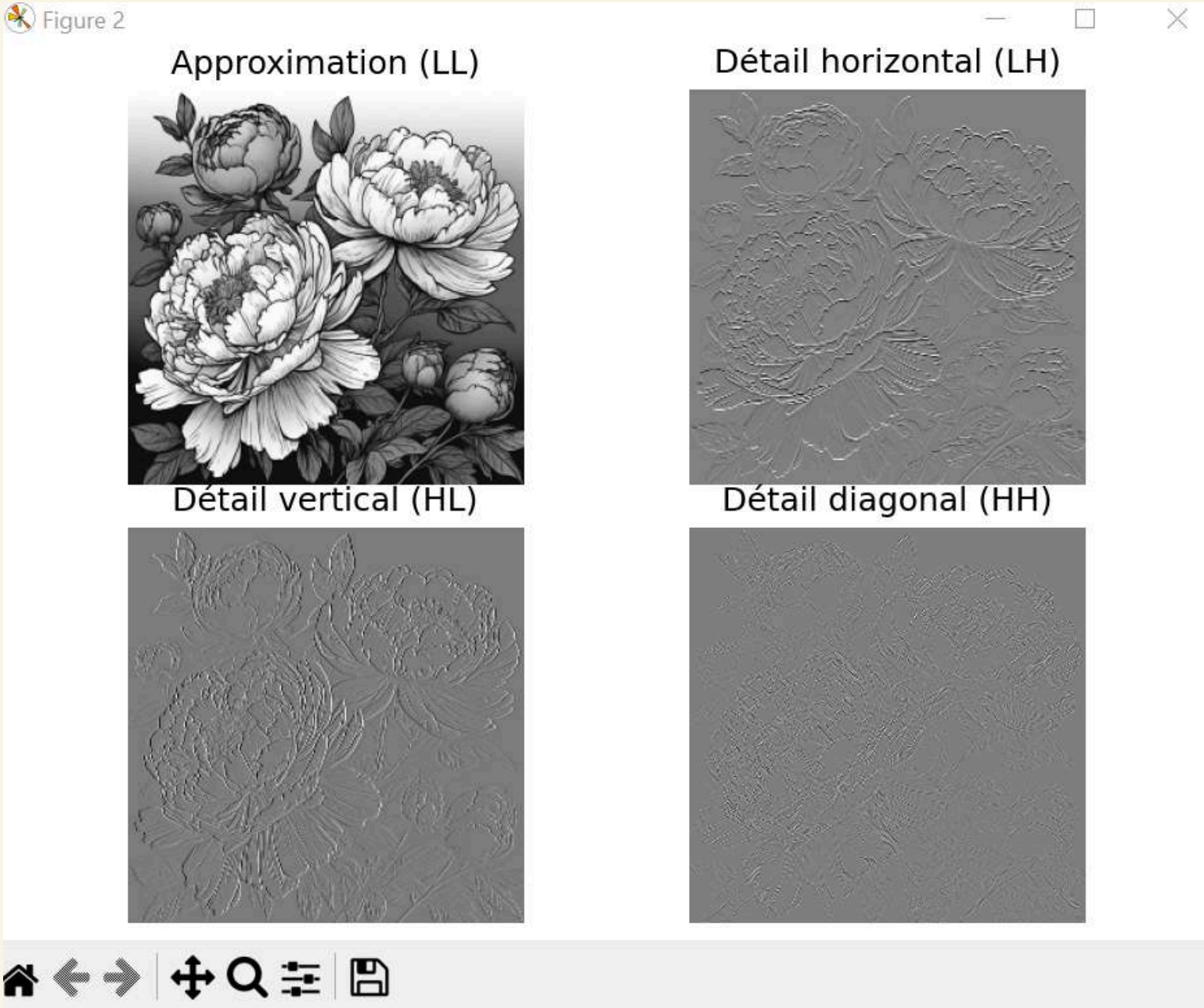


Figure 2
Approximation (LL)
Détail horizontal (LH)
Détail vertical (HL)
Détail diagonal (HH)

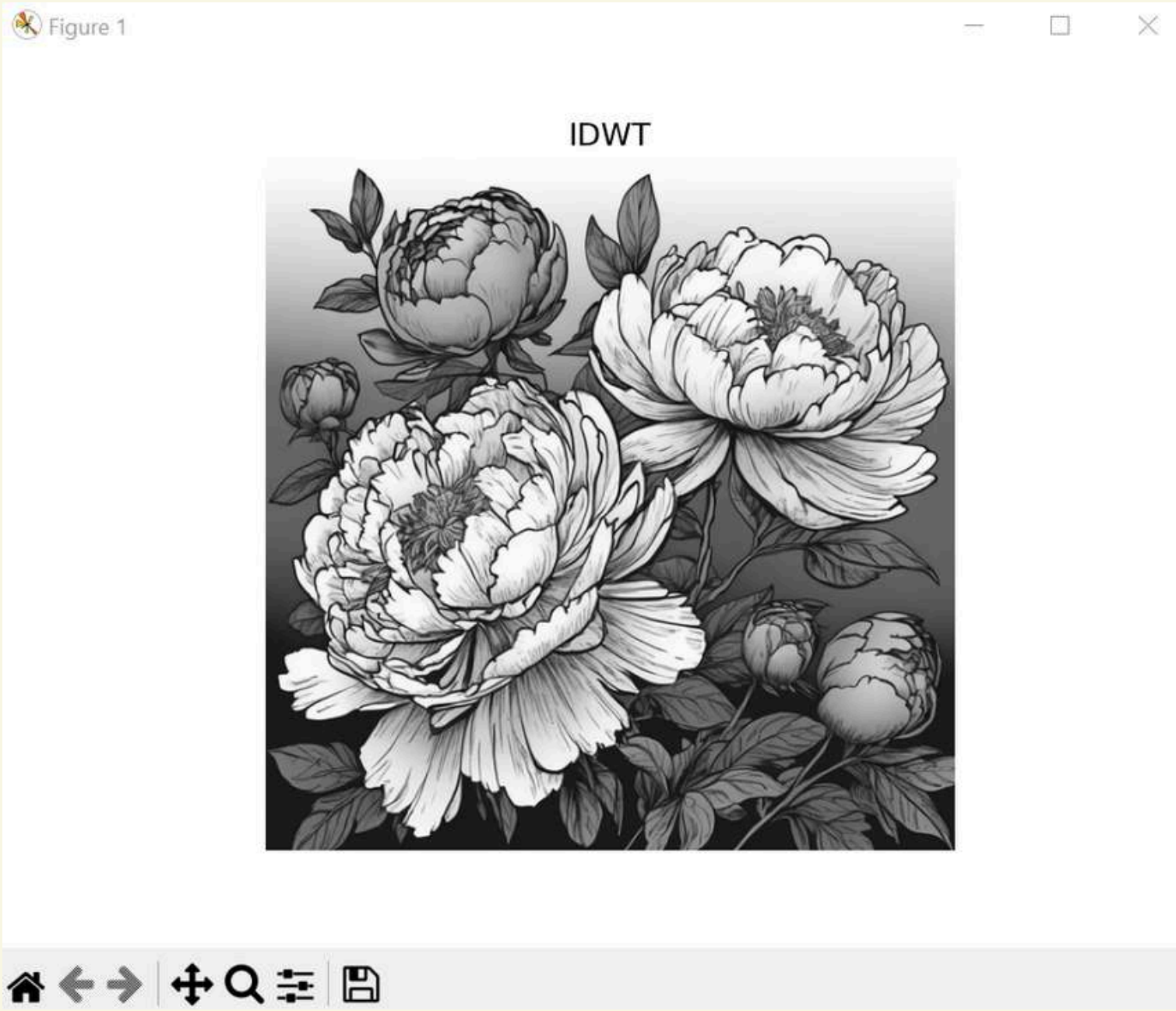


Figure 1
IDWT

Algorithme du tatouage numérique en utilisant la DWT

Algorithme de tatouage (niveaux de gris)

Entrées

image hôte (à tatouer)

le filigrane

DWT à 2 niveaux

Fonction d'insertion:
LL_tatouée= LL2*(1-alpha)+alpha*LLf2

IDWT à 2 niveaux

Sortie

image tatouée

Implémentation python et évaluation des résultats

on prend deux images de même taille 512x512

image hôte (à tatouer)

le filigrane

En niveaux de gris

image hôte

filigrane

insertion dans LL

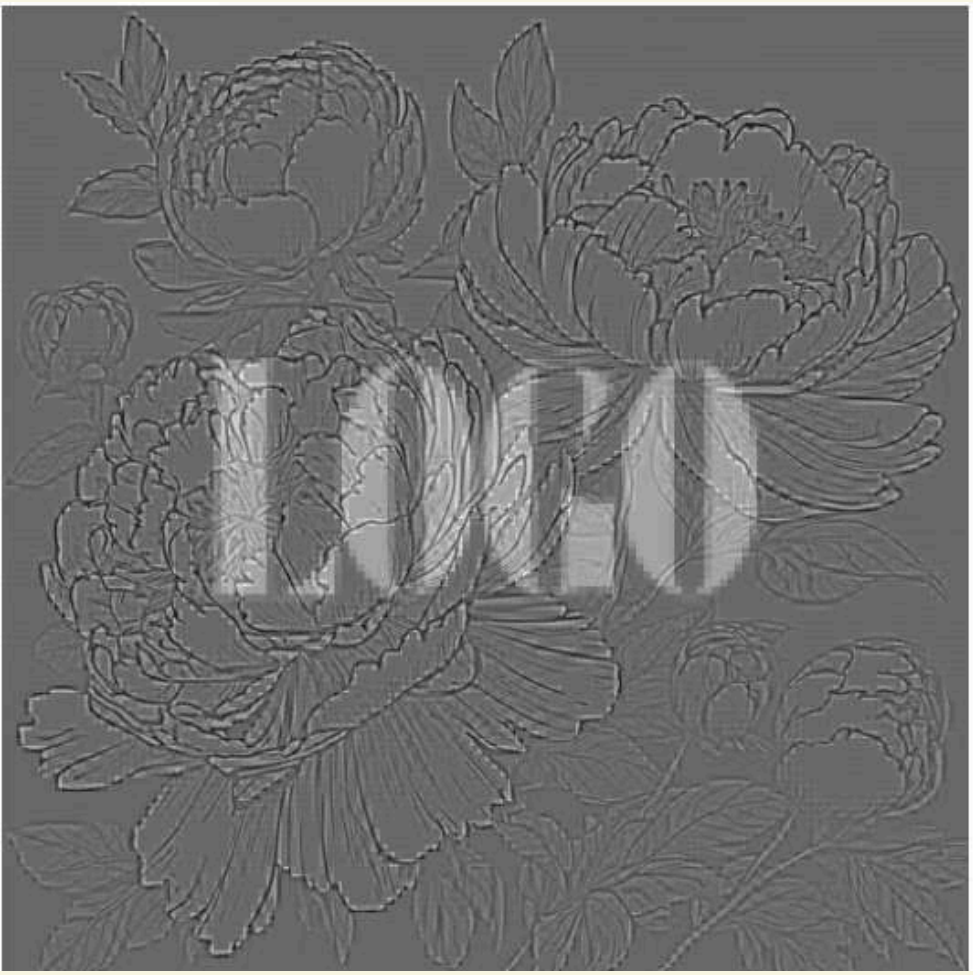
Image originale

Image tatouée

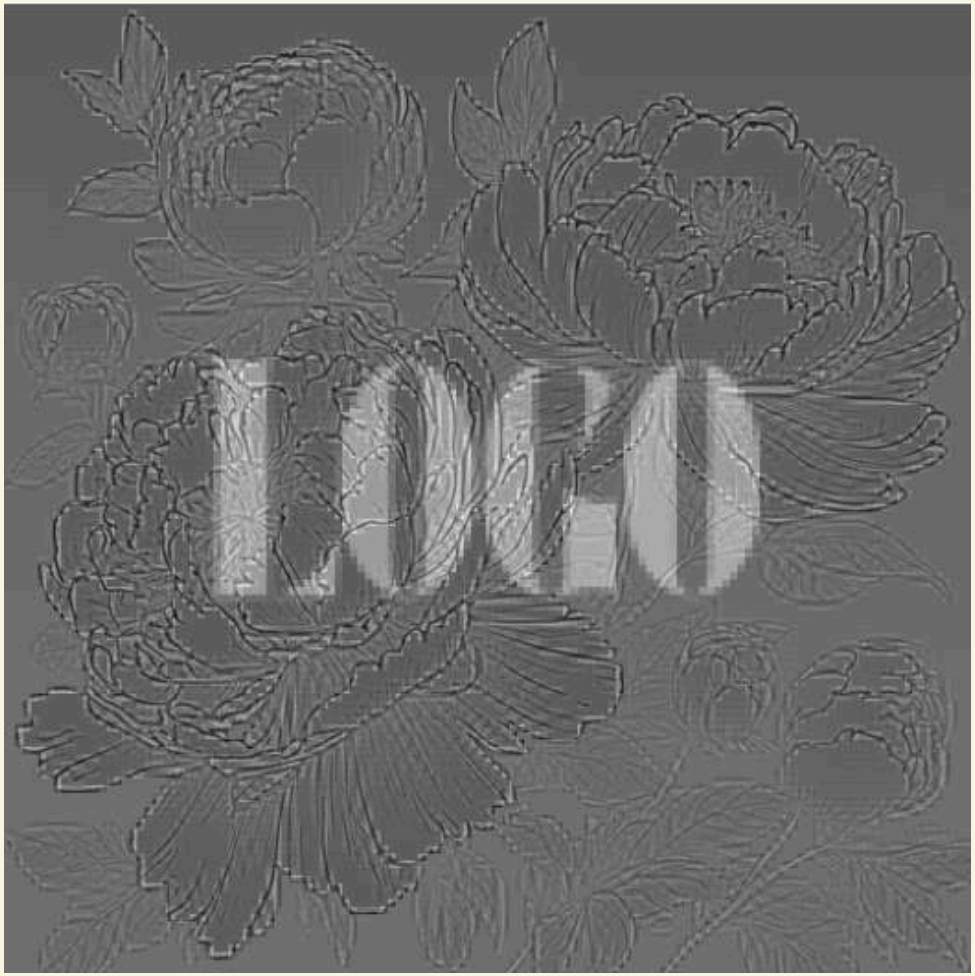
alpha =0.3

alpha=0.5

alpha=0.7



alpha=1



alpha=1.2

Tatouage numérique pour une image en RGB

On applique le même algorithme de tatouage sur chaque canal séparémant, puis on recombine les canaux R, G,B

1-DWT RGB

alpha=0.2

alpha=0.4

alpha=0.7

Tatouage numérique pour une image en RGB-(2-DWT)

Image originale

Image tatouée (visible RGB)

alpha=0.2

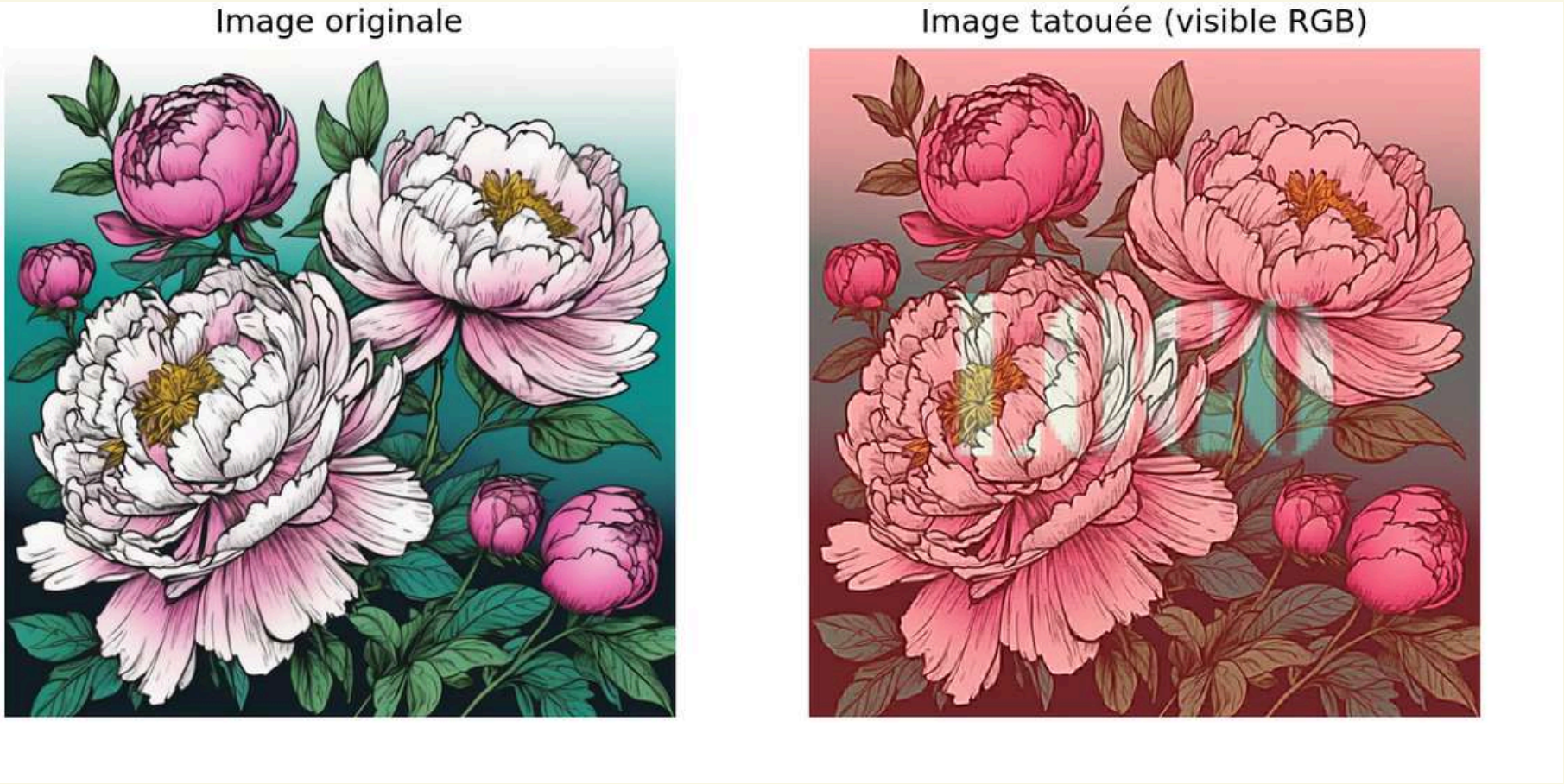
Tatouage numérique pour une image en RGB (2-DWT)

Image originale

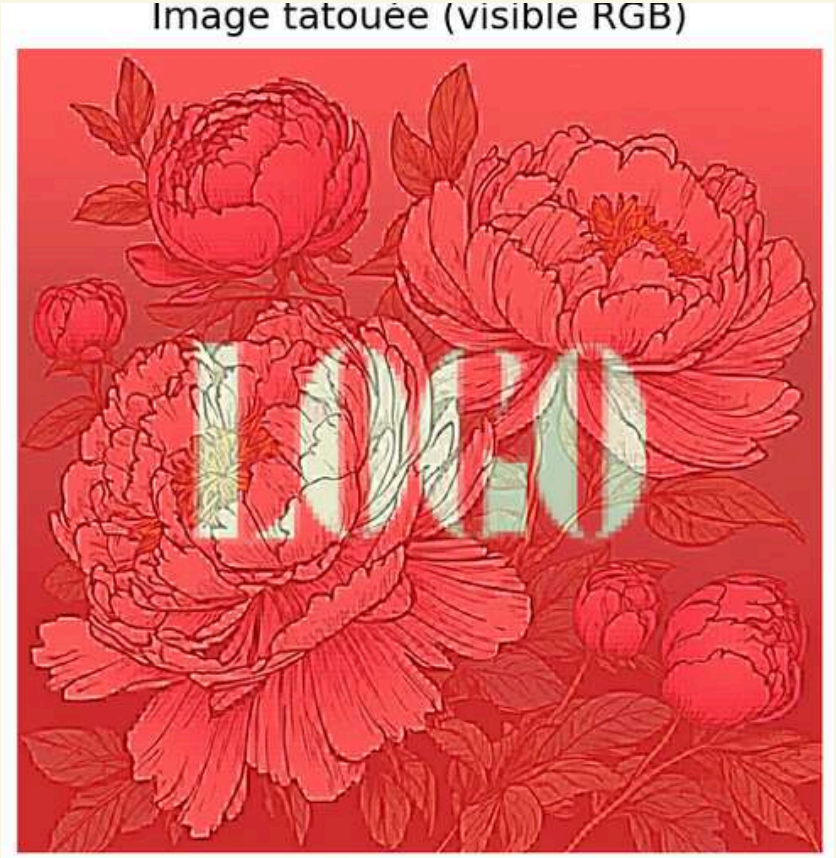
Image tatouée (visible RGB)

alpha = 0.3

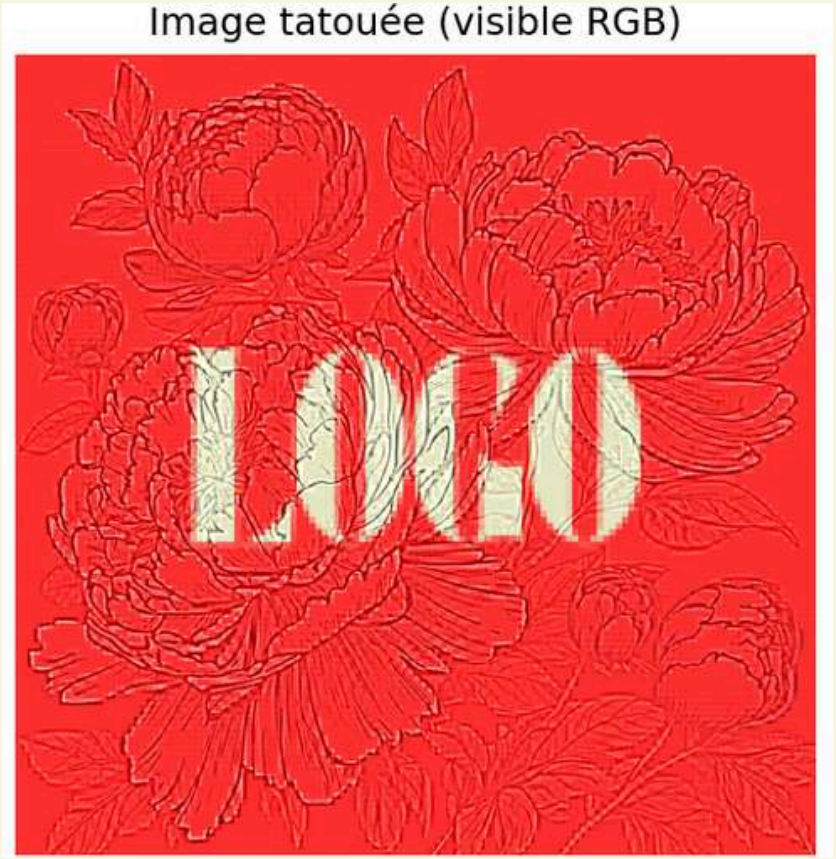
Tatouage numérique pour une image en RGB



alpha=0.4



alpha=0.8



alpha=0.7

Tatouage numérique pour une image en RGB (2-DWT)

En inversant les rôles

Image originale

Image tatouée (visible RGB)

alpha=0.8

problème de résolution

Pour augmenter la résolution, on fait diminuer le niveau de la dwt:

Image originale

Image tatouée (visible RGB)

alpha= 0.8 /1-DWT/rôles inversés

Image originale

Image tatouée (visible RGB)

alpha=0.9

2-DWT
alpha=0.4

1-DWT(rôles inversés)
alpha=0.8

- La 2-DWT offre la mailleur visibilité du filigrane, alors que la 1-DWT avec rôles inversés est d'un point de vue esthétique (ou visuel), car elle préserve les couleurs de l'image d'origine, et apporte moins de résolution comparée à la 2-DWT

Robustesse ?

ON ATTAQUE l’image taouée en utilisant 2-DWT

modification de saturation

Bruits

Ainsi, le filigrane résiste à ces attaques, faisant preuve de robustesse.

Que distingue la DWT de la simple sommation ?

Image hôte

Tatouage (fusion directe)

Méthode par sommation directe :

✅ Code très simple et rapide à mettre en œuvre,

❌ mais offre peu de contrôle (transparence fixe).

Méthode via DWT (transformée en ondelettes discrète) :

✅ Permet un meilleur contrôle des paramètres :
transparence du filigrane,
visibilité du message,

❌ Code plus complexe et plus long à exécuter.

1-DWT

2-DWT
alpha=0.4

1-DWT(rôles inversés)
alpha=0.8

CONCLUSION

Bien que plus complexe que la fusion directe,la transformée en ondelettes(de Haar) discrète (DWT) offre un outil puissant pour le tatouage visible, permettant d’insérer un filigrane avec un meilleur contrôle sur la transparence et la visibilité du filigrane, ce qui en fait une méthode robuste et esthétique pour le watermarking.

Annexe 1

```
1  import numpy as np
2  from PIL import Image
3  import matplotlib.pyplot as plt
4
```

```python
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt

# décomposition par DWT (1 niveau))
image = Image.open("C:\\Users\\Fati\\Downloads\\expp\\flower 3.png").convert("L")
image_np= np.array(image, dtype=float)

def Haar_1DWT(image):
    h = np.array([1/np.sqrt(2), 1/np.sqrt(2)])
    g = np.array([1/np.sqrt(2), -1/np.sqrt(2)])

    L, C = image.shape

    lignes_basses = np.zeros((L, C // 2))
    lignes_hautes = np.zeros((L, C // 2))

    # Passe sur les colonnes (1D DWT ligne par ligne)
    for i in range(L):
        for j in range(0, C, 2):
            lignes_basses[i, j // 2] = h[0] * image[i, j] + h[1] * image[i, j + 1]
            lignes_hautes[i, j // 2] = g[0] * image[i, j] + g[1] * image[i, j + 1]

    # Passe sur les lignes (1D DWT colonne par colonne)
    LL = np.zeros((L // 2, C // 2))
    LH = np.zeros((L // 2, C // 2))
    HL = np.zeros((L // 2, C // 2))
    HH = np.zeros((L // 2, C // 2))

    for j in range(C // 2):
        for i in range(0, L-1, 2):
            LL[i // 2, j] = h[0] * lignes_basses[i, j] + h[1] * lignes_basses[i + 1, j]
            LH[i // 2, j] = g[0] * lignes_basses[i, j] + g[1] * lignes_basses[i + 1, j]
            HL[i // 2, j] = h[0] * lignes_hautes[i, j] + h[1] * lignes_hautes[i + 1, j]
            HH[i // 2, j] = g[0] * lignes_hautes[i, j] + g[1] * lignes_hautes[i + 1, j]

    return LL, LH, HL, HH

LL, LH, HL, HH = Haar_1DWT(image_np)
```

```
LL, LH, HL, HH = haar_1DWT(image_np)

#affichage

plt.figure(figsize=(10, 8))

plt.subplot(2, 2, 1)
plt.title('Approximation (LL)')
plt.imshow(LL, cmap='gray')
plt.axis('off')

plt.subplot(2, 2, 2)
plt.title('Détail horizontal (LH)')
plt.imshow(LH, cmap='gray')
plt.axis('off')

plt.subplot(2, 2, 3)
plt.title('Détail vertical (HL)')
plt.imshow(HL, cmap='gray')
plt.axis('off')

plt.subplot(2, 2, 4)
plt.title('Détail diagonal (HH)')
plt.imshow(HH, cmap='gray')
plt.axis('off')

plt.tight_layout()
plt.show()
```

```
67
68  def haar_1idwt(LL, LH, HL, HH):
69      M, N = LL.shape
70
71      lignes_basses = np.zeros((2 * M, N))
72      lignes_hautes = np.zeros((2 * M, N))
73
74      for j in range(N):
75          for i in range(M):
76              lignes_basses[2 * i, j] = (LL[i, j] + LH[i, j]) / np.sqrt(2)
77              lignes_basses[2 * i + 1, j] = (LL[i, j] - LH[i, j]) / np.sqrt(2)
78              lignes_hautes[2 * i, j] = (HL[i, j] + HH[i, j]) / np.sqrt(2)
79              lignes_hautes[2 * i + 1, j] = (HL[i, j] - HH[i, j]) / np.sqrt(2)
80
81      image = np.zeros((2 * M, 2 * N))
82
83      for i in range(2 * M):
84          for j in range(N):
85              image[i, 2 * j] = (lignes_basses[i, j] + lignes_hautes[i, j]) / np.sqrt(2)
86              image[i, 2 * j + 1] = (lignes_basses[i, j] - lignes_hautes[i, j]) / np.sqrt(2)
87
88      return image
89
90  LL, LH, HL, HH = Haar_1DWT(image_np)
91
92  im=haar_1idwt(LL, LH, HL,HH)
93
94  plt.imshow(im, cmap='gray')
95  plt.title('IDWT')
96  plt.axis('off')
97  plt.show()
98
```

```python
def haar_dwt_recursive(image, niveau=2):
    if niveau == 0:
        return image
    LL, LH, HL, HH = Haar_1DWT(image)
    if niveau == 1:
        return (LL, LH, HL, HH)
    else:
        LL_rec = haar_dwt_recursive(LL, niveau - 1)
        return (LL_rec, LH, HL, HH)


def haar_2idwt(coeff):
    LL2, LH2, HL2, HH2 = coeff
    if isinstance(LL2, tuple):
        LL1 = haar_2idwt(LL2)
    else:
        LL1 = LL2
    return haar_1idwt(LL1, LH2, HL2, HH2)

```

```python
def tatouage(hote, filigrane, alpha):
    # DWT de l'image originale
    coeffs_hote = haar_dwt_recursive(hote, niveau=2)

    # DWT du filigrane
    coeffs_filigrane = haar_dwt_recursive(filigrane, niveau=2)

    # Insertion du filigrane dans la composante LL la plus profonde (niveau 2)
    # coeffs_hote[0] est la composante LL ou tuple des coefficients niveau 2
    LL_hote, LH_hote, HL_hote, HH_hote = coeffs_hote[0]
    LL_fi, _, _, _ = coeffs_filigrane[0]

    LL_tatouee = LL_hote*(1-alpha) + alpha * LL_fi

    # Recréer la structure des coefficients avec la LL modifiée
    coeffs_hote_mod = list(coeffs_hote)
    coeffs_hote_mod[0] = (LL_tatouee, LH_hote, HL_hote, HH_hote)

    # Reconstruction de l'image tatouée
    image_tatouee = haar_2idwt(coeffs_hote_mod)

    return image_tatouee
```

```
# Chargement des images (en niveaux de gris)
image = Image.open("C:\\Users\\Fati\\Downloads\\expp\\flower 3.png").convert("L")
image_np = np.array(image, dtype=float)

imageW = Image.open("C:\\Users\\Fati\\Downloads\\expp\\LOGO 2.png").convert("L")
imageW_np = np.array(imageW, dtype=float)

wm_image =tatouage(image_np, imageW_np, 0.3)
# === Affichage ===
plt.figure(figsize=(10,5))
plt.subplot(1,2,1)
plt.imshow(image_np.astype(np.uint8), cmap="gray")
plt.title("Image originale")
plt.axis("off")

plt.subplot(1,2,2)
plt.imshow(wm_image,cmap="gray")
plt.title("Image tatouée ")
plt.axis("off")
plt.show()
```

```python
# ===  watermark visible en RGB ===
def DWT1_tat(image_rgb, watermark_rgb, alpha):
    result_channels = []

    for c in range(3):  # pour chaque canal R, G, B
        host = image_rgb[:, :, c]
        wm = watermark_rgb[:, :, c]

        LL_h, LH_h, HL_h, HH_h = Haar_1DWT(host)
        LL_wm, _, _, _ = Haar_1DWT(wm)

        LL_fused = LL_h*(1-alpha) + alpha*LL_wm

        fused_channel = haar_1idwt(LL_fused, LH_h, HL_h, HH_h)
        fused_channel = np.clip(fused_channel, 0, 255)
        result_channels.append(fused_channel)

    return np.stack(result_channels, axis=2).astype(np.uint8)


image= Image.open("C:\\Users\\Fati\\Downloads\\expp\\flower 3.png").convert("RGB")
image_np = np.array(image, dtype=float)

imageW = Image.open("C:\\Users\\Fati\\Downloads\\expp\\LOGO 2.png").convert("RGB")
imageW_np = np.array(imageW, dtype=float)
```

```
28
29  # === Appliquer ===
30  wm_rgb = DWT1_tat(imageW_np, image_np, alpha=0.8)
31
32  # === Affichage ===
33  plt.figure(figsize=(10,5))
34  plt.subplot(1,2,1)
35  plt.imshow(image_np.astype(np.uint8))
36  plt.title("Image originale")
37  plt.axis("off")
38
39  plt.subplot(1,2,2)
40  plt.imshow(wm_rgb)
41  plt.title("Image tatouée (visible RGB)")
42  plt.axis("off")
43  plt.show()
```

```python

# Chargement des images
image= Image.open("C:\\Users\\Fati\\Downloads\\expp\\flower 3.png").convert("RGB")
image_np = np.array(image, dtype=float)

imageW = Image.open("C:\\Users\\Fati\\Downloads\\expp\\LOGO 2.png").convert("RGB")
imageW_np = np.array(imageW, dtype=float)

def tatouage_rgb(image_rgb, filigrane_rgb, alpha):

    # Séparer les canaux
    R, G, B = image_rgb[:,:,0], image_rgb[:,:,1], image_rgb[:,:,2]
    Rw, Gw, Bw = filigrane_rgb[:,:,0], filigrane_rgb[:,:,1], filigrane_rgb[:,:,2]

    R_tatoue = tatouage(R, Rw, alpha)
    G_tatoue = tatouage(G, Gw, alpha)
    B_tatoue = tatouage(B, Bw, alpha)

    # Recomposer l'image RGB tatouée
    image_tatouee = np.stack([R_tatoue, G_tatoue, B_tatoue], axis=2)

    image_tatouee = np.clip(image_tatouee, 0, 255).astype(np.uint8)

    return image_tatouee

wm_image = tatouage_rgb(image_np, imageW_np, 0.8)
```

```
27  # === Affichage ===
28  plt.figure(figsize=(10,5))
29  plt.subplot(1,2,1)
30  plt.imshow(image_np.astype(np.uint8))
31  plt.title("Image originale")
32  plt.axis("off")
33
34  plt.subplot(1,2,2)
35  plt.imshow(wm_image)
36  plt.title("Image tatouée (visible RGB)")
37  plt.axis("off")
38  plt.show()
39
```

Annexe 2

```python
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt

# Charger les images en RGB
image_hote = Image.open("C:\\Users\\Fati\\Downloads\\expp\\flower 3.png").convert("RGB")
image_filigrane = Image.open("C:\\Users\\Fati\\Downloads\\expp\\LOGO 2.png").convert("RGB")

# Convertir en arrays numpy
im_host = np.array(image_hote, dtype=float)
im_wm = np.array(image_filigrane, dtype=float)

alpha = 0.5
image_tatouee = (1 - alpha) * im_host + alpha * im_wm
image_tatouee = np.clip(image_tatouee, 0, 255)




# Afficher
plt.figure(figsize=(12, 4))
plt.subplot(1, 3, 1)
plt.imshow(im_hote.astype(np.uint8))
plt.title("Image hôte")
plt.axis('off')


plt.subplot(1, 3, 2)
plt.imshow(image_tatouee.astype(np.uint8))
plt.title("Tatouage (fusion directe)")
plt.axis('off')
plt.show()

```

```python
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from skimage import util
import numpy as np

# Chargement image
path = r"C:\Users\Fati\Downloads\expp\tatouée.png"
image = mpimg.imread(path)


# Bruits disponibles : 'gaussian', 'salt & pepper', 'speckle', 'poisson', 'localvar'
noisy_gaussian = util.random_noise(image, mode='gaussian', mean=0, var=0.1)
noisy_sp = util.random_noise(image, mode='s&p', amount=0.1)
noisy_speckle = util.random_noise(image, mode='speckle', var=0.1)

# Affichage
plt.figure(figsize=(8, 8))
plt.subplot(2,2,1)
plt.title("Image originale")
plt.imshow(image)
plt.axis('off')

plt.subplot(2,2,2)
plt.title("Bruit Gaussien")
plt.imshow(noisy_gaussian)
plt.axis('off')

plt.subplot(2,2,3)
plt.title("Bruit Sel et Poivre")
plt.imshow(noisy_sp)
plt.axis('off')

plt.subplot(2,2,4)
plt.title("Bruit Speckle")
plt.imshow(noisy_speckle)
plt.axis('off')

plt.show()

```

Preuve : Les filtres h[n] et g[n] pour les ondelettes de Haar

on a:

$$\frac{1}{\sqrt{2}}\phi\left(\frac{t}{2}\right) = \sum_{n=-\infty}^{+\infty} h[n]\phi(t-n),$$

$$\text{avec} \quad h[n] = \left\langle \frac{1}{\sqrt{2}}\phi\left(\frac{t}{2}\right), \phi(t-n) \right\rangle.$$

$$\phi(t) \neq 0 \iff 0 \leq t < 1,$$

et on a

$$\phi(2t-n) \neq 0 \iff 0 \leq 2t-n < 1,$$

donc

$$n \leq 2t < n+1,$$

et

$$n > 2t-1 > -1,$$

d'où

$$n = 0, 1.$$

Preuve:

$$
\begin{aligned}
h[0] = 2^{-1/2}\langle \phi(t/2), \phi(t) \rangle & \\
&= \frac{1}{\sqrt{2}} \int_{-\infty}^{+\infty} \phi(t/2)\phi(t)\, dt \\
&= \frac{1}{\sqrt{2}} \int_{0}^{1} \phi(t/2)\phi(t) dt \\
&= \frac{1}{\sqrt{2}} \left(\int_{0}^{1} 1\, dt \right) \\
&= \frac{1}{\sqrt{2}}.
\end{aligned}
$$

$$
\begin{aligned}
h[1] = 2^{-1/2}\langle \phi(t/2), \phi(t-1) \rangle & \\
&= \frac{1}{\sqrt{2}} \int_{-\infty}^{+\infty} \phi(t/2)\phi(t-1)\, dt \\
&= \frac{1}{\sqrt{2}} \int_{0}^{1} \phi(t/2)\phi(t-1) dt \\
&= \frac{1}{\sqrt{2}} \left(\int_{0}^{1} 1\, dt \right) \\
&= \frac{1}{\sqrt{2}}.
\end{aligned}
$$

Preuve:

$$
\begin{aligned}
g[0] = 2^{-1/2}\langle \psi(t/2), \phi(t) \rangle \\
&= \frac{1}{\sqrt{2}} \int_{-\infty}^{+\infty} \psi(t/2)\phi(t)\, dt \\
&= \frac{1}{\sqrt{2}} \int_{0}^{1} \psi(t/2)\phi(t) dt \\
&= \frac{1}{\sqrt{2}} \left(\int_{0}^{1} -1.1\, dt \right) \\
&= \frac{-1}{\sqrt{2}}.
\end{aligned}
$$

$$
\begin{aligned}
g[1] = 2^{-1/2}\langle \psi(t/2), \phi(t-1) \rangle \\
&= \frac{1}{\sqrt{2}} \int_{-\infty}^{+\infty} \psi(t/2)\phi(t-1)\, dt \\
&= \frac{1}{\sqrt{2}} \left(\int_{0}^{1} -1.0 dt + \int_{1}^{2} 1 dt \right) \\
&= \frac{1}{\sqrt{2}}.
\end{aligned}
$$