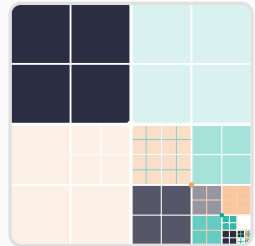


MP 2026

Super-résolution fractale : Étude comparative des cycles PIFS et FIF appliqués à l'imagerie médicale.

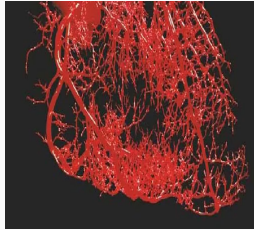
Mohamed Amine Hzikar





Chou Romanesco

Fig. 1



Réseau vasculaire

Fig. 2



Fougère

Fig. 3

Illustration de l'auto-similarité et du motif itératif dans la nature

” Face à l'interpolation polynomiale, quelle approche fractale, **PIFS** ou **FIF**, optimise le compromis PSNR/SSIM pour la super-résolution d'images médicales ?”

- I. Cadrage Théorique : De Banach aux Algorithmes Fractals**
- II. Confrontation Qualitative : Le Choc des Métriques**
- III. Le Mur de la Complexité : L'Échappatoire Quad-tree**
- IV. L'Alliance Différentielle : Conception du Modèle Hybride**
- V. Bilan et Arbitrage Final**

Théorème du Point Fixe de Banach

Soit (E, d) un espace métrique complet.

Soit $W : E \rightarrow E$ une application contractante de rapport $c \in [0, 1[$:

$$d(W(x), W(y)) \leq c \cdot d(x, y)$$

$\implies$ **Existence et unicité**
d'un point fixe $a \in E$ tel que
 $W(a) = a$.

Application au Contexte Médical

- **L'espace E** : L'espace des images.
- **Notation** : $f : \Omega \rightarrow \mathcal{C}$
(Fonction spatiale vers intensité lumineuse).
- **La métrique L^2** : Erreur Quadratique Moyenne :

$$d_2(f, g) = \left(\frac{1}{|\Omega|} \iint_{\Omega} |f(x, y) - g(x, y)|^2 dx dy \right)^{\frac{1}{2}}$$

- **L'opérateur W** : Application contractante de transformation d'image.

Convergence absolue des itérées :

$$\forall x_0 \in E, \quad \lim_{n \rightarrow \infty} W^{\circ n}(x_0) = a$$

Application à l'image : $\lim_{n \rightarrow \infty} W^{\circ n}(I_0) = A$

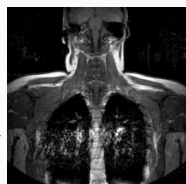
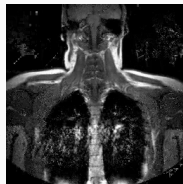
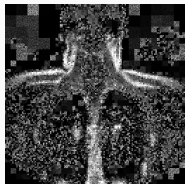
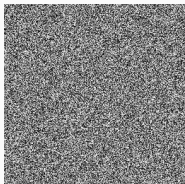


Fig. 4 (Kaggle)

Bruit / I_0

$W^{\circ 5}(I_0)$

$W^{\circ 10}(I_0)$

Attracteur A

Le Théorème du Collage

$$d(I, A) \leq \frac{d(I, W(I))}{1 - c}$$

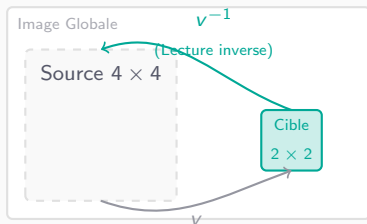
où c est le facteur de contraction

L'objectif algorithmique est de minimiser $d(I, W(I))$
pour forcer l'attracteur A à cloner notre image médicale I .

$$W_{local}(f)(x, y) = s \cdot f(v^{-1}(x, y)) + \text{shift}(x, y)$$

Exemple élémentaire :

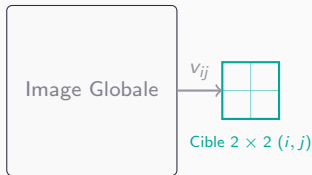
Si v contracte une source 4×4 vers une cible 2×2 (ratio $1/2$), l'ordinateur utilise l'inverse v^{-1} (ratio 2) pour multiplier les coordonnées du pixel cible afin d'aller lire la couleur correspondante dans la source.



La condition de contraction absolue : $\forall \text{ morceau, } |s| < 1$

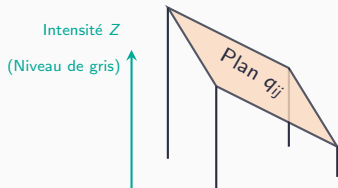
Équation FIF (par bloc d'indices i, j)

$$W_{ij}(f)(x, y) = s \cdot f(v_{ij}^{-1}(x, y)) + q_{ij}(x, y)$$



- s est la *rugosité* fractale (fixée globalement).
- q_{ij} est un plan d'ajustement bilinéaire, calculé par un système linéaire 4×4 exact.

$$q_{ij}(x, y) = ax + by + cxy + d$$



Le but : Forcer une continuité absolue C^0 : $W(I) = I$

Équation PIFS (par bloc cible R_i)

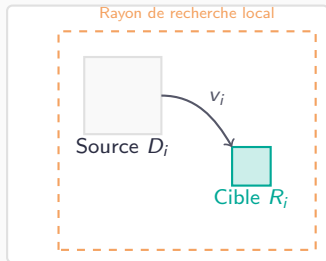
$$W_i(f)(x, y) = s_i \cdot f(v_i^{-1}(x, y)) + o_i$$

On minimise l'EQM :

$$\min \sum_{(x,y)} (s_i \cdot D_i + o_i - R_i)^2$$

Calcul par moindres carrés :

$$s_i = \frac{n \sum R_i D_i - \sum R_i \sum D_i}{n \sum D_i^2 - (\sum D_i)^2} \quad ; \quad o_i = \frac{1}{n} \left(\sum R_i - s_i \sum D_i \right)$$



Le but : Minimiser l'erreur : $W(I) \approx I$

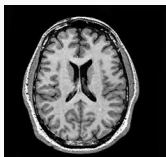


Fig. 5 (Kaggle)

IRM Cérébrale



Fig. 6 (Kaggle)

Scanner Pulmonaire



Fig. 7 (Kaggle)

IRM du Rachis

Fidélité Absolue : PSNR

$$\text{PSNR} = 20 \log_{10} \left(\frac{\text{MAX}_I}{\sqrt{\text{EQM}}} \right)$$

MAX_I correspond à la valeur maximale possible du pixel (ex : 255).

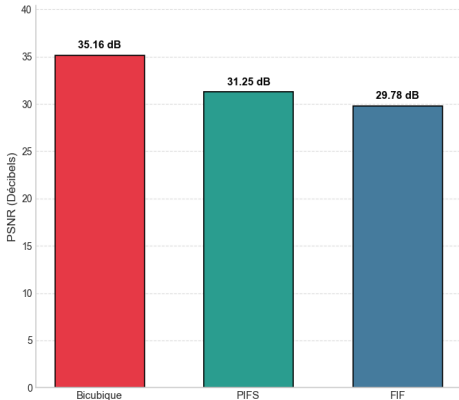
L'EQM (Erreur Quadratique Moyenne) est la transposition numérique de notre distance théorique d_2 .

Fidélité Clinique : SSIM

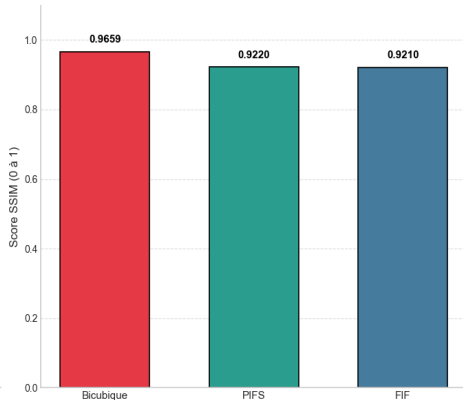
Indice perceptuel complexe évaluant la **luminance**, le **contraste** et la **corrélacion spatiale** des structures, s'affranchissant du biais du pixel-à-pixel pour capter la texture tissulaire globale.

Résultats de l'Expérience 1 : Évaluation Quantitative (Moyennes globales)

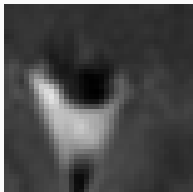
Peak Signal-to-Noise Ratio (PSNR)
↑ Plus haut = Moins d'erreur absolue



Structural Similarity Index (SSIM)
↑ 1.0 = Structure identique au Ground Truth



1. Vérité Clinique (GT)



Vérité Clinique (Ground Truth)

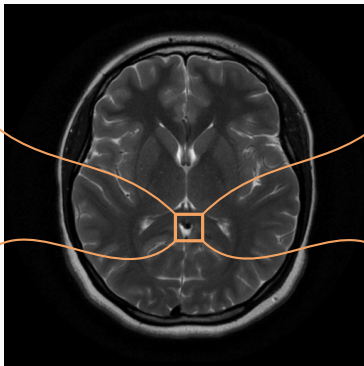
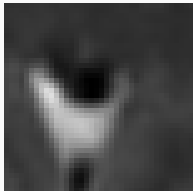
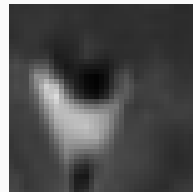
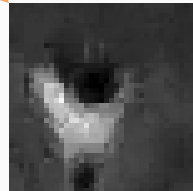


Fig. 8 (Kaggle)

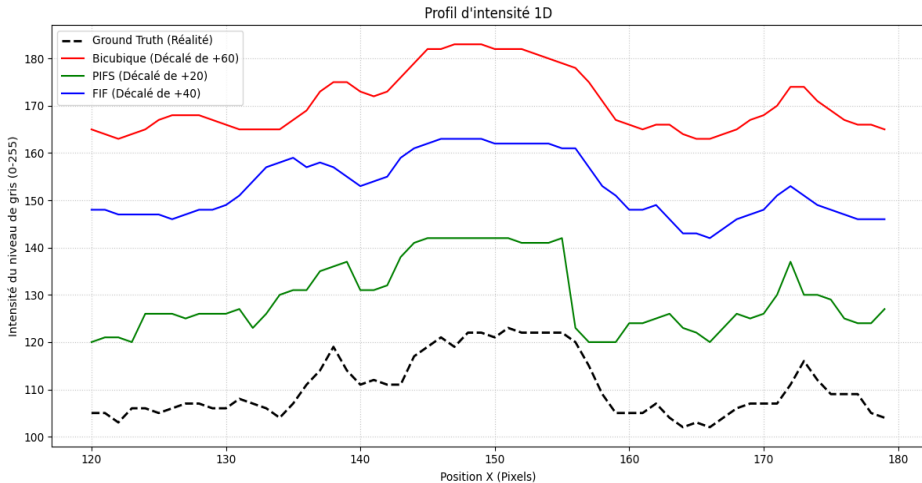
3. Approche Algébrique (FIF)



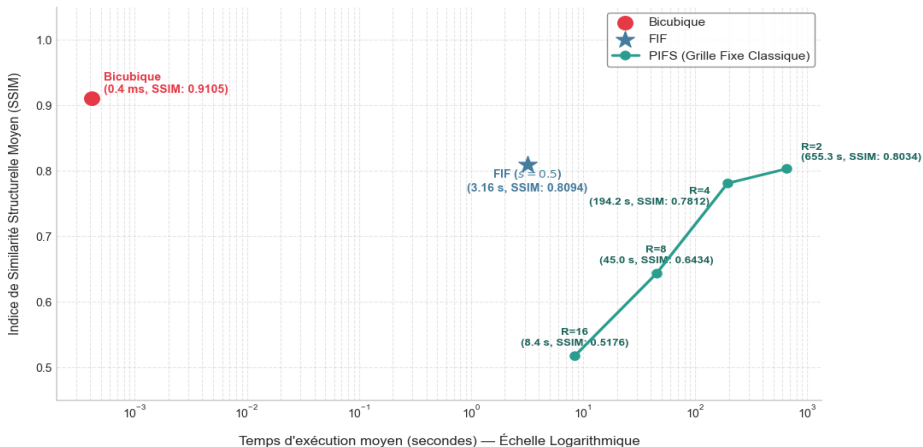
2. Interpolation Bicubique



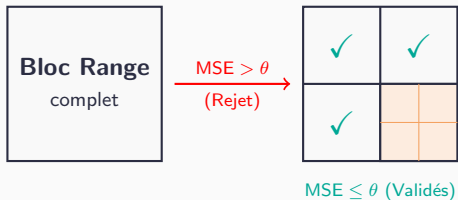
4. Approche Spatiale (PIFS)



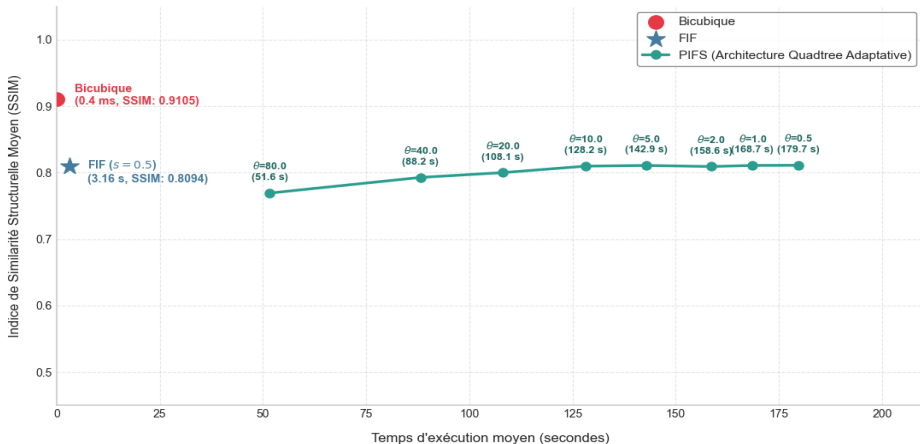
Frontière de Pareto : Compromis Temps de Calcul vs. Fidélité Structurale (SSIM)

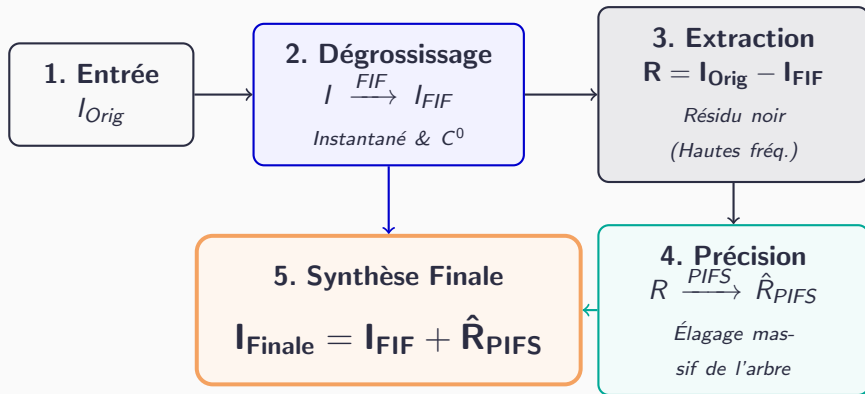


- ▶ **Taille dynamique** : Le partitionnement s'adapte à la topologie de l'image.
- ▶ **Évaluation de l'erreur** : $MSE(R_i, W_i(D_i))$ calculée à chaque étape.
- ▶ **Seuil de tolérance θ** : Si l'erreur dépasse θ , le bloc est scindé récursivement.



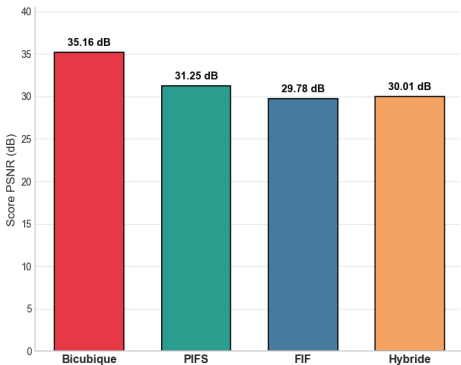
Frontière de Pareto : Évaluation de l'Architecture Adaptive Quadtree



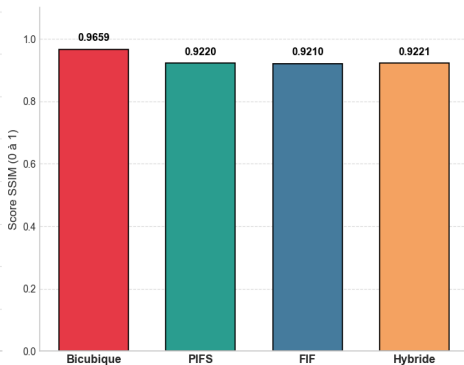


Comparaison Statistique : Approches Classique, Fractales et Hybride

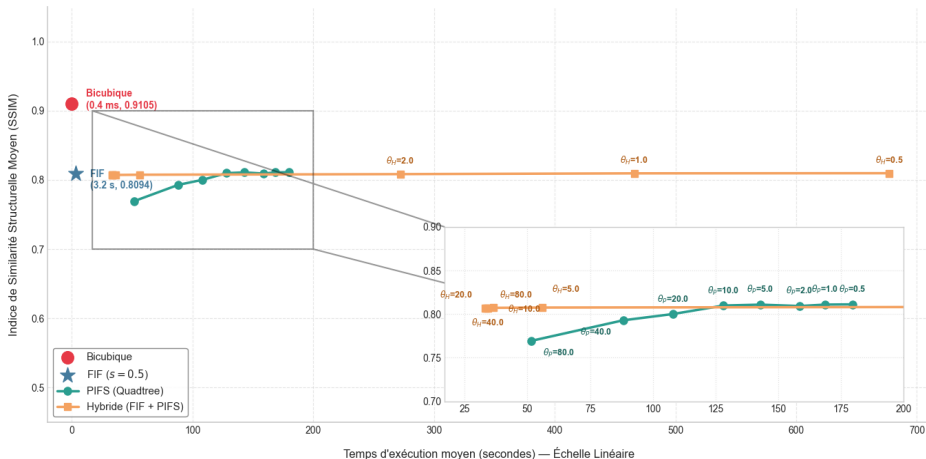
Fidélité du Signal (PSNR)
↑ Plus haut = Moins d'erreur quadratique



Similarité Structurale (SSIM)
↑ 1.0 = Structure identique au Ground Truth



Frontières de Pareto : Évaluation de l'Architecture Hybride vs Quadtree



Approche Spatiale (PIFS)

Texture excellente, mais
coût temporel élevé
et artefacts locaux.

VS

Approche Algébrique (FIF)

Vitesse fulgurante, mais lissage
excessif de la rugosité clinique.

Synthèse Différentielle (Le Modèle Hybride)

- **Sécurité Topologique** (Structure continue C^0 garantie par le FIF)
- **Précision Clinique** (Rugosité restaurée localement par le PIFS)
- **Viabilité Hospitalière** (Temps optimisé par l'élagage Quadtree)



Merci pour votre attention.

Super-résolution fractale en imagerie médicale

Illustrations Naturelles

Fig. 1 : Chou-fleur Romanesco

<https://www.graines-bocquet.fr/blog/post/1131/>

`tout-savoir-sur-le-chou-fleur-romanesco`

Fig. 2 : Le cœur fractal et chaotique

<https://www.pourlascience.fr/sd/medecine/>

`le-cœur-fractal-et-chaotique-5157.php`

Fig. 3 : Fougères à l'ombre

<https://lejardindemamere.fr/jardin/fougères-a-l-ombre->

`les-plantes-parfaites-pour-les-zones-difficiles/`

Datasets Cliniques (Kaggle)

Fig. 4 & 7 : Spinal Vertebrae Segmentation

Auteur : Unique Data

<https://www.kaggle.com/datasets/trainingdatapro/>

`spinal-vertebrae-segmentation`

Fig. 5 & 8 : Brain MRI Images for Tumor Detection

Auteur : Navoneel Chakrabarty

<https://www.kaggle.com/datasets/navoneel/>

`brain-mri-images-for-brain-tumor-detection`

Fig. 6 : CT-Medical-Images

Auteur : Rishan Hasan Tenis

<https://www.kaggle.com/datasets/rishantenis/>

`ct-medical-images`

Cadre : Espace métrique complet (E, d) , opérateur contractant $W : E \rightarrow E$ de rapport $k \in [0, 1[$.

Objectif : Prouver l'unicité du point fixe $A \in E$ et la convergence de la suite $x_{n+1} = W(x_n)$.

Preuve de l'Unicité :

Soient deux points fixes A et B (tels que $W(A) = A$ et $W(B) = B$). On a $d(A, B) = d(W(A), W(B)) \leq k \cdot d(A, B)$. Puisque $k < 1$, cette inégalité stricte impose obligatoirement $d(A, B) = 0$, d'où $A = B$.

Preuve de l'Existence (Convergence) :

Pour $x_0 \in E$ arbitraire, on évalue la distance entre itérations successives :

$$d(x_{n+1}, x_n) \leq \dots \leq k^n \cdot d(x_1, x_0).$$

Pour prouver que (x_n) est une suite de Cauchy, on évalue $d(x_m, x_n)$ pour $m > n$ par l'inégalité triangulaire :

$$d(x_m, x_n) \leq \sum_{i=n}^{m-1} d(x_{i+1}, x_i) \leq d(x_1, x_0) \sum_{i=n}^{m-1} k^i \leq d(x_1, x_0) \frac{k^n}{1-k}$$

Puisque $\lim_{n \rightarrow \infty} k^n = 0$, la suite (x_n) est bien de Cauchy. L'espace E étant complet, elle converge vers $A \in E$.

W étant contractante, elle est continue. Par passage à la limite :

$$W(A) = W(\lim_{n \rightarrow \infty} x_n) = \lim_{n \rightarrow \infty} W(x_n) = A.$$

Cadre : On cherche un opérateur W dont l'attracteur A approxime une image cible I .

Objectif : Borner la distance $d(I, A)$ par l'erreur de "collage" $d(I, W(I))$.

Preuve :

D'après l'inégalité triangulaire appliquée à l'image cible I , à sa transformée $W(I)$, et à l'attracteur A , on a :

$$d(I, A) \leq d(I, W(I)) + d(W(I), A)$$

Or, l'attracteur A est le point fixe de W (donc $W(A) = A$), et l'opérateur W est contractant de rapport $k < 1$. Par conséquent :

$$d(W(I), A) = d(W(I), W(A)) \leq k \cdot d(I, A)$$

En réinjectant cette majoration dans la première inégalité, on obtient directement :

$$d(I, A) \leq d(I, W(I)) + k \cdot d(I, A)$$

Puisque $k < 1$, on peut isoler la distance à l'attracteur $d(I, A)$:

$$d(I, A) \leq \frac{1}{1 - k} \cdot d(I, W(I))$$

Conclusion : Rendre l'attracteur indiscernable de la cible ($d(I, A) \rightarrow 0$) revient algorithmiquement à minimiser l'erreur locale $d(I, W(I))$. C'est le fondement de l'optimisation PIFS/FIF.

Cadre : L'opérateur unifié W agit sur des morceaux indexés i . L'équation locale est :
 $(Wf)(x, y) = s_i \cdot f(v_i^{-1}(x, y)) + \text{shift}_i(x, y)$ pour tout $(x, y) \in \text{morceau } i$

Majoration locale (Norme du maximum L^∞) :

Soient $f, g \in E$. Sur un morceau i , la différence absolue s'écrit (les termes shift_i s'annulent) :

$$|(Wf) - (Wg)| = |s_i| \cdot |f(v_i^{-1}) - g(v_i^{-1})|$$

Prenons le supremum de cette différence **sur le morceau cible** i . Le point

$(u, v) = v_i^{-1}(x, y)$ parcourt alors un domaine source D_i :

$$\sup_{\text{morceau } i} |Wf - Wg| = |s_i| \cdot \sup_{(u, v) \in D_i} |f(u, v) - g(u, v)|$$

Or, le supremum sur le sous-domaine D_i est nécessairement majoré par celui sur l'image entière Ω :

$$\sup_{D_i} |f(u, v) - g(u, v)| \leq \sup_{\Omega} |f(u, v) - g(u, v)| = d_\infty(f, g)$$

On obtient donc la majoration locale : $\sup_{\text{morceau } i} |Wf - Wg| \leq |s_i| \cdot d_\infty(f, g)$

Contraction globale :

Pour obtenir la distance globale sur l'image entière, on prend le maximum sur l'ensemble des morceaux i :

$$d_\infty(Wf, Wg) = \max_i \left(\sup_{\text{morceau } i} |Wf - Wg| \right) \leq \left(\max_i |s_i| \right) \cdot d_\infty(f, g)$$

Dans l'espace L^∞ , W est donc contractant si et seulement si $\forall i, |s_i| < 1$.

Problème métrique : En clinique et dans notre code, la qualité n'est pas mesurée par la norme uniforme L^∞ , mais via l'Erreur Quadratique Moyenne (EQM), qui dérive de la métrique L^2 . La condition de contraction $|s_i| < 1$ y est-elle toujours valable ?

Dimension Finie :

Une image médicale numérique n'est pas une surface spatiale continue infinie. C'est un objet discret mathématiquement défini par une matrice de pixels $N \times M$.

Notre espace des images E est donc un espace vectoriel isomorphe à $\mathbb{R}^{N \times M}$, qui est de **dimension finie**.

Théorème fondamental :

En topologie, **dans un espace vectoriel de dimension finie, toutes les normes sont équivalentes**.

Conclusion pour l'Algorithme :

Nous avons prouvé la contraction sous L^∞ . Par équivalence topologique, cette contrainte agit comme un garde-fou universel. Elle garantit automatiquement que l'opérateur W sera contractant sous L^2 . L'algorithme converge donc inconditionnellement vis-à-vis de l'EQM.

Cadre de l'Évaluation Globale

Corpus : 5 images par catégorie (Total : 15 images)

Résolution cible (Ground Truth) : 256×256

Dégradation : Facteur de zoom 2 ($128 \rightarrow 256$)

Exp. 1 : Algorithme PIFS pur

- Quadtree : $16 \rightarrow 2$
- Seuil tolérance θ : 1.0
- Recherche : 32 — Pas : 4

Exp. 1 : Algorithme FIF pur

- Facteur rugosité s : 0.5

Exp. 3 : Synthèse Différentielle (Modèle Hybride)

1. Phase de Dégrossissage (FIF)

- Rugosité maintenue à $s = 0.5$
(Identique à l'expérience 1)

2. Phase de Précision (PIFS)

- Quadtree max réduit à $8 \rightarrow 2$
(Cible directement le résidu fin)
- Seuil strict θ : 0.5
(Garantit la capture des micro-textures)

Catégorie	Image	Bicubique		PIFS		FIF		Hybride	
		PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM
Cerveau	1 no.jpeg	33.77	0.964	29.96	0.916	29.25	0.926	29.38	0.925
Cerveau	10 no.jpg	39.91	0.982	34.83	0.946	33.59	0.941	33.90	0.943
Cerveau	11 no.jpg	32.89	0.979	29.07	0.950	28.39	0.951	28.43	0.952
Cerveau	12 no.jpg	34.53	0.976	29.21	0.920	29.21	0.938	29.46	0.939
Cerveau	13 no.jpg	33.98	0.974	29.45	0.924	27.68	0.933	27.85	0.927
Poumon	000108 (3).png	35.58	0.964	32.72	0.923	29.80	0.906	30.25	0.911
Poumon	000109 (2).png	35.86	0.958	31.69	0.911	30.02	0.911	30.37	0.913
Poumon	000109 (4).png	35.22	0.956	31.68	0.916	29.46	0.905	29.84	0.908
Poumon	000109 (5).png	34.41	0.952	31.32	0.911	29.66	0.900	29.88	0.901
Poumon	000112 (2).png	35.31	0.955	32.07	0.917	30.03	0.905	30.31	0.907
Rachis	IM000001.jpg	31.65	0.941	27.25	0.865	26.86	0.871	27.19	0.873
Rachis	IM000002.jpg	33.03	0.951	29.23	0.889	28.09	0.892	28.41	0.894
Rachis	IM000003.jpg	35.48	0.966	30.92	0.919	29.46	0.918	29.93	0.920
Rachis	IM000004.jpg	37.73	0.982	34.46	0.960	32.42	0.956	32.32	0.956
Rachis	IM000005.jpg	38.05	0.982	34.85	0.958	32.65	0.956	32.55	0.955

Le Défi du Passage à l'Échelle (Haute Résolution)

Corpus : 1 image par catégorie (Total : 3 images)

Résolution cible (Ground Truth) : 512×512

Dégradation massive : Facteur de zoom 4 ($128 \rightarrow 512$)

Exp. 2A : PIFS Classique

Variable de balayage :

- Taille de bloc $R \in \{16, 8, 4, 2\}$

Constantes spatiales :

- Rayon : 32 — Pas : 4

Exp. 2B : PIFS Quadtree

Variable de balayage :

- Seuil $\theta \in [0.5, \dots, 80.0]$

Constantes spatiales :

- Blocs limités de $16 \rightarrow 2$
- Rayon : 32 — Pas : 4

Le Point de Référence Algébrique

Algorithme FIF pur **fixé** avec un facteur de rugosité $s = 0.5$

Configuration	Cerveau		Poumon		Rachis	
	<i>1 no.jpeg</i>		<i>000108 (3).png</i>		<i>IM0000001.jpg</i>	
	Temps (s)	SSIM	Temps (s)	SSIM	Temps (s)	SSIM
Bicubique	< 0.01	0.931	< 0.01	0.929	< 0.01	0.870
FIF ($s = 0.5$)	3.01	0.857	3.13	0.835	3.33	0.735
PIFS ($R = 16$)	8.23	0.617	8.70	0.557	8.19	0.378
PIFS ($R = 8$)	42.74	0.688	43.66	0.721	48.57	0.520
PIFS ($R = 4$)	192.32	0.814	198.50	0.836	191.90	0.692
PIFS ($R = 2$)	652.27	0.839	666.78	0.848	646.82	0.721

Configuration	Cerveau		Poumon		Rachis	
	<i>1 no.jpeg</i>		<i>000108 (3).png</i>		<i>IM000001.jpg</i>	
	Temps (s)	SSIM	Temps (s)	SSIM	Temps (s)	SSIM
Bicubique	< 0.01	0.931	< 0.01	0.929	< 0.01	0.870
FIF ($s = 0.5$)	3.55	0.857	5.14	0.835	4.54	0.735
PIFS ($E = 0.5$)	149.96	0.841	178.83	0.857	210.27	0.734
PIFS ($E = 1.0$)	137.72	0.838	167.08	0.859	201.15	0.734
PIFS ($E = 2.0$)	129.63	0.832	155.68	0.861	190.46	0.733
PIFS ($E = 5.0$)	115.90	0.835	132.36	0.864	180.45	0.732
PIFS ($E = 10.0$)	101.24	0.835	110.61	0.862	172.69	0.731
PIFS ($E = 20.0$)	81.72	0.835	88.95	0.839	153.68	0.725
PIFS ($E = 40.0$)	68.80	0.835	68.03	0.828	127.76	0.714
PIFS ($E = 80.0$)	59.93	0.819	37.53	0.801	57.42	0.686

Objectif : Superposition des courbes de Pareto sur le même environnement de test

(Mêmes conditions : Résolution 512×512

- Dégradation Zoom $\times 4$
- 3 images cibles)

Paramétrage du Modèle Hybride pour le Balayage

1. Phase FIF (Ajustée)

- Rugosité abaissée à $s = 0.4$
(Dégrossissage optimisé pour le calcul du résidu)

2. Phase PIFS (Alignement strict)

- Quadtree intégral : $16 \rightarrow 2$
(Garantit une comparaison équitable avec l'Exp. 2B)

Variable de Balayage Temporel & Qualitatif

Seuil de tolérance dynamique pour l'hybride :

$$\theta \in \{0.5, 1.0, 2.0, 5.0, 10.0, 20.0, 40.0, 80.0\}$$

Paramètre	Cerveau		Poumon		Rachis	
	<i>1 no.jpeg</i>		<i>000108 (3).png</i>		<i>IM000001.jpg</i>	
	Temps (s)	SSIM	Temps (s)	SSIM	Temps (s)	SSIM
0.5	455.76	0.856	816.39	0.837	758.62	0.735
1.0	192.45	0.855	665.05	0.837	540.05	0.736
2.0	31.08	0.852	411.59	0.836	373.47	0.735
5.0	24.92	0.852	61.41	0.835	81.59	0.734
10.0	26.25	0.852	40.34	0.835	42.93	0.733
20.0	25.93	0.852	43.32	0.835	31.73	0.733
40.0	25.37	0.852	41.65	0.835	35.39	0.733
80.0	30.51	0.852	35.92	0.835	33.08	0.733

Configuration Matérielle

- **Processeur (CPU) :**
Intel Core i5-1035G1
@ 1.00GHz (Base 1.19GHz)
- **Architecture :**
4 cœurs physiques / 8 threads
- **Mémoire Vive (RAM) :**
16 Go

Environnement Logiciel

- **Système d'Exploitation :**
Windows (64-bit)
- **Interpréteur Core :**
Python 3.10.11
- **Parallélisme
(Multiprocessing) :**
Distribution inter-images
(Aucune accélération intra-image)

Protocole de Mesure Temporelle

Le temps de calcul de chaque algorithme est mesuré de manière stricte sur un seul thread CPU.

Le parallélisme n'a servi qu'à traiter le corpus global sans impacter les scores individuels.

```
1 import numpy as np
2 import cv2
3
4
5 def base_plane(pillars, corners, s):
6     # interpolation:  $q(u,v) = au + bv + cuv + d$ 
7     # subtract the self-similar contribution fi
8     rst
9     z0 = pillars[0] - s * corners[0]
10    z1 = pillars[1] - s * corners[1]
11    z2 = pillars[2] - s * corners[2]
12    z3 = pillars[3] - s * corners[3]
13
14    d = z0
15    a = z1 - z0
16    b = z2 - z0
17    c = z3 - a - b - d
18    return a, b, c, d
19
20 def roughness(p, max_s=0.5):
21     contrast = np.max(p) - np.min(p)
22     s = (contrast / 255.0) * max_s
23     return np.clip(s, 0.0, 0.99)
24
25
26 def encode_fif(image, max_roughness=0.5):
27     h, w = image.shape
28
29     corners = [image[0, 0], image[0, w-1], image[h-1, 0], image[h-1, w-1]]
30
31     # pad by 1 so we can always grab a 2x2 neighborhood
```

```
32     padded = cv2.copyMakeBorder(image, 0, 1, 0,
33     1, cv2.BORDER_REPLICATE)
34
35     S_mat = np.zeros((h, w))
36     A_mat = np.zeros((h, w))
37     B_mat = np.zeros((h, w))
38     C_mat = np.zeros((h, w))
39     D_mat = np.zeros((h, w))
40
41     for y in range(h):
42         for x in range(w):
43             p = [padded[y, x], padded[y, x+1],
44             padded[y+1, x], padded[y+1, x+1]]
45
46             s = roughness(p, max_s=max_roughness)
47
48             a, b, c, d = base_plane(p, corners,
49             s)
50
51             S_mat[y, x] = s
52             A_mat[y, x] = a
53             B_mat[y, x] = b
54             C_mat[y, x] = c
55             D_mat[y, x] = d
56
57     return {
58         'metadata': {'original_shape': (h, w),
59         'max_roughness': max_roughness},
60         'matrices': {'s': S_mat, 'a': A_mat, 'b': B_mat, 'c': C_mat, 'd': D_mat}
61     }
```

```
59 def decode_fif(fractal_data, iterations=8, zoom
_factor=2):
60     orig_h, orig_w = fractal_data['metadata']['
        original_shape']
61     m = fractal_data['matrices']
62
63     h = orig_h * zoom_factor
64     w = orig_w * zoom_factor
65
66     img = np.ones((h, w)) * 128.0
67
68     # sub-pixel grid within each zoomed cell
69     u = np.linspace(0, 1, zoom_factor, endpoint
=False)
70     v = np.linspace(0, 1, zoom_factor, endpoint
=False)
71     U, V = np.meshgrid(u, v)
72
73     for _ in range(iterations):
74         nxt = np.zeros((h, w))
75
76         # global attractor shrunk to zoom_factor
x zoom_factor
77         g = cv2.resize(img, (zoom_factor, zoom_
factor), interpolation=cv2.INTER_AREA)
78
79         for y in range(orig_h):
80             for x in range(orig_w):
81                 s = m['s'][y, x]
82                 a = m['a'][y, x]
83                 b = m['b'][y, x]
84                 c = m['c'][y, x]
85                 d = m['d'][y, x]
```

```
87         q = a*U + b*V + c*(U*V) + d
88         block = s * g + q
89
90         nxt[y*zoom_factor:(y+1)*zoom_fa
ctor, x*zoom_factor:(x+1)*zoom_factor] = block
91
92         img = nxt
93
94     return np.clip(img, 0, 255).astype(np.uint8
)
```

```

1 import numpy as np
2 import cv2
3
4 def isometries(block):
5     variants = []
6     for k in range(4):
7         variants.append(np.rot90(block, k))
8     flipped = np.fliplr(block)
9     for k in range(4):
10        variants.append(np.rot90(flipped, k))
11    return variants
12
13
14 def fit_affine(R, D):
15     # least squares fit for s and o such that s
16     *D + o ~ R
17     # with Banach contractivity enforced: |s| <
18     1
19     R_f = R.astype(np.float64).flatten()
20     D_f = D.astype(np.float64).flatten()
21     n = len(R_f)
22     sR = np.sum(R_f)
23     sD = np.sum(D_f)
24     denom = n * np.sum(D_f**2) - sD**2
25     if denom == 0:
26         s = 0.0
27     else:
28         s = (n * np.sum(R_f * D_f) - sR * sD) /
29         denom
30         s = np.clip(s, -0.99, 0.99)
31         o = (sR - s * sD) / n
32         return s, o

```

```

32
33 def encode_pifs(image, range_size=8, search_rad
34 ius=32, domain_step=4):
35     h, w = image.shape
36     dom_size = range_size * 2
37     transforms = []
38
39     for ry in range(0, h, range_size):
40         for rx in range(0, w, range_size):
41             R = image[ry:ry+range_size, rx:rx+r
42                 range_size]
43
44             best_err = float('inf')
45             best = None
46
47             y0 = max(0, ry - search_radius)
48             y1 = min(h - dom_size, ry + search_
49                 radius)
50
51             x0 = max(0, rx - search_radius)
52             x1 = min(w - dom_size, rx + search_
53                 radius)
54
55             for dy in range(y0, y1, domain_step
56                 ):
57                 for dx in range(x0, x1, domain_
58                     step):
59                     D_large = image[dy:dy+dom_s
60                         ize, dx:dx+dom_size]
61                     D_small = cv2.resize(D_larg
62                         e, (range_size, range_size), interpolation=cv2.
63                         INTER_AREA)
64
65                     for idx, D_iso in enumerate
66                         (isometries(D_small)):

```

```

56         s, o = fit_affine(R, D_
iso)
57         err = np.mean((R - (s *
D_iso + o))**2)
58
59         if err < best_err:
60             best_err = err
61             best = {
62                 'rx': rx, 'ry':
ry,
63                 'dx': dx, 'dy':
dy,
64                 'iso': idx,
65                 's': s, 'o': o
66             }
67
68         transforms.append(best)
69
70     return {
71         'metadata': {
72             'original_shape': (h, w),
73             'range_size': range_size,
74             'search_radius': search_radius,
75             'domain_step': domain_step
76         },
77         'transformations': transforms
78     }
79
80
81 def decode_pifs(fractal_data, iterations=8, zoo
m_factor=1):
82     orig_h, orig_w = fractal_data['metadata']['
original_shape']

```

```

83     bsz = fractal_data['metadata']['range_size'
]
84     transforms = fractal_data['transformations'
]
85     h = orig_h * zoom_factor
86     w = orig_w * zoom_factor
87     rsz = bsz * zoom_factor
88     dsz = rsz * 2
89     canvas = np.ones((h, w)) * 128.0
90
91     for _ in range(iterations):
92         nxt = np.zeros((h, w))
93
94         for t in transforms:
95             rx = t['rx'] * zoom_factor
96             ry = t['ry'] * zoom_factor
97             dx = t['dx'] * zoom_factor
98             dy = t['dy'] * zoom_factor
99
100             D_large = canvas[dy:dy+dsz, dx:dx+d
sz]
101             D_small = cv2.resize(D_large, (rsz,
rsz), interpolation=cv2.INTER_AREA)
102             D_iso = isometries(D_small)[t['iso'
]]
103
104             nxt[ry:ry+rsz, rx:rx+rsz] = t['s']
* D_iso + t['o']
105
106             canvas = nxt
107
108     return np.clip(canvas, 0, 255).astype(np.ui
nt8)

```

```
1 import numpy as np
2 import cv2
3
4
5 def isometries(block):
6     variants = []
7     for k in range(4):
8         variants.append(np.rot90(block, k))
9     flipped = np.fliplr(block)
10    for k in range(4):
11        variants.append(np.rot90(flipped, k))
12    return variants
13
14
15 def build_pools(image, step, sizes):
16     # precompute downsampled domain blocks and
17     their scalar stats
18     # so the inner loop can use dot products in
19     stead of materializing arrays
20     h, w = image.shape
21     pools = {}
22
23     for S in sizes:
24         D_size = S * 2
25         pool = []
26
27         for y in range(0, h - D_size + 1, step):
28             for x in range(0, w - D_size + 1, step):
29                 dom = image[y:y+D_size, x:x+D_size]
```

```
30                 dom_small = cv2.resize(dom, (S, S), interpolation=cv2.INTER_AREA).astype(np.float64)
31                 flat = dom_small.ravel()
32                 sD = np.sum(flat)
33                 sD2 = np.dot(flat, flat)
34                 isos = [iso.ravel() for iso in isometries(dom_small)]
35                 pool.append((x, y, isos, sD, sD2))
36
37         pools[S] = pool
38
39     return pools
40
41
42 def encode_pifs(image, max_size=16, min_size=2, threshold=5.0, search_radius=8, domain_step=4):
43     h, w = image.shape
44     image = image.astype(np.float64)
45
46     sizes = []
47     s = max_size
48     while s >= min_size:
49         sizes.append(s)
50         s //= 2
51
52     print(f"[PIFS] sizes={sizes}, threshold={threshold}")
53     pools = build_pools(image, domain_step, sizes)
```

```

82         # scalar SSE – avoids allocation
      g the residual array
83         sse = sR2 - 2*s_val*sRD - 2*o_v
al*sR + s_val**2*sD2 + 2*s_val*o_val*sD + n*o_v
al**2
84         err = max(0.0, sse / n)
85
86         if err < best_err:
87             best_err = err
88             best = {'rx': x, 'ry': y, '
size': size,
89                   'dx': dx, 'dy': dy,
90                   's': s_val, 'o': o_
val}
91
92         # fallback if nothing passed the radius
filter (can happen at borders)
93         if best is None:
94             sdX = max(0, min(x, w - size*2))
95             sdY = max(0, min(y, h - size*2))
96             best = {'rx': x, 'ry': y, 'size': s
ize,
97                   'dx': sdX, 'dy': sdY, 'dir'
: 0,
98                   's': 0.0, 'o': sR / n}
99
100         if best_err > threshold and size > min_
size:
101             half = size // 2
102             split(x, y, half)
103             split(x + half, y, half)
104             split(x, y + half, half)
105             split(x + half, y + half, half)

```

```
106         else:
107             blocks.append(best)
108
109     for y in range(0, h, max_size):
110         for x in range(0, w, max_size):
111             split(x, y, max_size)
112
113     print(f"[PIFS] done, {len(blocks)} blocks")
114     return {
115         'metadata': {
116             'original_shape': (h, w),
117             'quadtree_max': max_size,
118             'quadtree_min': min_size,
119             'threshold_used': threshold
120         },
121         'transformations': blocks
122     }
123
124
125 def decode_pifs(fractal_data, iterations=8, zoom
126 m_factor=1):
127     orig_h, orig_w = fractal_data['metadata']['
128 original_shape']
129     blocks = fractal_data['transformations']
130
131     h = orig_h * zoom_factor
132     w = orig_w * zoom_factor
133
134     canvas = np.ones((h, w), dtype=np.float64)
135     * 128.0
136
137     for _ in range(iterations):
138         nxt = np.zeros((h, w), dtype=np.float64
139 )
```

```
136
137     for b in blocks:
138         rx = b['rx'] * zoom_factor
139         ry = b['ry'] * zoom_factor
140         rsz = b['size'] * zoom_factor
141
142         dx = b['dx'] * zoom_factor
143         dy = b['dy'] * zoom_factor
144         dsz = b['size'] * 2 * zoom_factor
145
146         D_large = canvas[dy:dy+dsz, dx:dx+
147 dsz]
148         D_small = cv2.resize(D_large, (rsz
149 , rsz), interpolation=cv2.INTER_AREA)
150         D_trans = isometries(D_small)[b['d
151 ir']]
152
153         nxt[ry:ry+rsz, rx:rx+rsz] = b['s']
154         * D_trans + b['o']
155
156     canvas = nxt
157
158     return np.clip(canvas, 0, 255).astype(np.ui
159 nt8)
```

```

1 import numpy as np
2 import cv2
3 import time
4
5 # -----
6 # -----
7 # FIF internals (float64 throughout)
8 # -----
9
10 def _fif_base_plane(P, C, s):
11     d = P[0] - s * C[0]
12     a = (P[1] - s * C[1]) - d
13     b = (P[2] - s * C[2]) - d
14     c = (P[3] - s * C[3]) - a - b - d
15     return a, b, c, d
16
17
18 def _fif_encode(img, max_roughness=0.5):
19     h, w = img.shape
20     S = np.zeros((h, w), dtype=np.float64)
21     A = np.zeros((h, w), dtype=np.float64)
22     B = np.zeros((h, w), dtype=np.float64)
23     CC = np.zeros((h, w), dtype=np.float64)
24     D = np.zeros((h, w), dtype=np.float64)
25
26     corners = [img[0, 0], img[0, w-1], img[h-1,
27 0], img[h-1, w-1]]
28
29     for y in range(h):
30         for x in range(w):
31             yn = min(y+1, h-1)
32             xn = min(x+1, w-1)

```

```

32         p = [img[y, x], img[y, xn], img[yn,
33 x], img[yn, xn]]
34
35         contrast = np.max(p) - np.min(p)
36         rough = (contrast / 255.0) * max_roughness
37
38         a, b, c, d = _fif_base_plane(p, corners, rough)
39
40         S[y, x] = rough
41         A[y, x] = a
42         B[y, x] = b
43         CC[y, x] = c
44         D[y, x] = d
45
46     return {'meta': {'shape': (h, w)}, 'mats':
47 {'s': S, 'a': A, 'b': B, 'c': CC, 'd': D}}
48
49
50 def _fif_decode(data, iters=8, zoom=2):
51     orig_h, orig_w = data['meta']['shape']
52     m = data['mats']
53     h = orig_h * zoom
54     w = orig_w * zoom
55     canvas = np.ones((h, w), dtype=np.float64)
56     * 128.0
57     u = np.linspace(0, 1, zoom, endpoint=False)
58     v = np.linspace(0, 1, zoom, endpoint=False)
59     U, V = np.meshgrid(u, v)
60     for _ in range(iters):
61         nxt = np.zeros((h, w), dtype=np.float64)
62
63     # cv2.resize needs float32

```

```

59         g = cv2.resize(canvas.astype(np.float32
60         ), (zoom, zoom), interpolation=cv2.INTER_AREA).
61         astype(np.float64)
62         for y in range(orig_h):
63             for x in range(orig_w):
64                 s = m['s'][y, x]
65                 a = m['a'][y, x]
66                 b = m['b'][y, x]
67                 c = m['c'][y, x]
68                 d = m['d'][y, x]
69                 base = a*U + b*V + c*U*V + d
70                 nxt[y*zoom:(y+1)*zoom, x*zoom:(
71                 x+1)*zoom] = base + s * g
72
73         canvas = nxt
74
75         return canvas
76
77         # -----
78         # -----
79         # PIFS internals (float64, supports negative re
80         # siduals)
81         # -----
82         # -----
83
84         def _isometries(block):
85             variants = []
86             for k in range(4):
87                 variants.append(np.rot90(block, k))
88             flipped = np.fliplr(block)
89             for k in range(4):
90                 variants.append(np.rot90(flipped, k))
91             return variants

```

```

87
88
89     def _build_pools(img, step, sizes):
90         h, w = img.shape
91         pools = {}
92         for S in sizes:
93             dsz = S * 2
94             pool = []
95             for y in range(0, h - dsz + 1, step):
96                 for x in range(0, w - dsz + 1, step
97                 ):
98                     dom = img[y:y+dsz, x:x+dsz]
99                     dom_small = cv2.resize(dom.asty
100                     pe(np.float32), (S, S), interpolation=cv2.INTER
101                     _AREA).astype(np.float64)
102                     pool.append((x, y, dom_small))
103                 pools[S] = pool
104             return pools
105
106         def _pifs_encode_qt(residual, max_size, min_siz
107         e, threshold, search_radius, domain_step):
108             sizes = [max_size // (2**i) for i in range(
109             int(np.log2(max_size / min_size)) + 1)]
110             pools = _build_pools(residual, domain_step,
111             sizes)
112             transforms = []
113
114             def split(x, y, size):
115                 R = residual[y:y+size, x:x+size]
116                 best_err = float('inf')
117                 best = None
118                 for dx, dy, D_dom in pools[size]:

```

```

114         if abs(dx - x) > search_radius or a
bs(dy - y) > search_radius:
115             continue
116
117     for dir_idx, D_t in enumerate(_isom
etries(D_dom)):
118         D_mean = np.mean(D_t)
119         D_var = np.var(D_t)
120         R_mean = np.mean(R)
121
122         if D_var < 1e-6:
123             s_val = 0.0
124             o_val = float(R_mean)
125         else:
126             cov = np.cov(R.flatten(),
D_t.flatten())[0, 1]
127             s_val = np.clip(cov / D_var
, -1.0, 1.0)
128             o_val = float(R_mean - s_val
l * D_mean)
129
130         err = np.mean((R - (s_val * D_t
+ o_val))**2)
131
132         if err < best_err:
133             best_err = err
134             best = {'dx': dx, 'dy': dy,
'dir': dir_idx, 's': float(s_val), 'o': o_val}
135
136     if best_err <= threshold or size <= min
_size:
137         transforms.append({'rx': x, 'ry': y
, 'size': size, **best})
138     else:

```

```

139         half = size // 2
140         split(x, y, half)
141         split(x + half, y, half)
142         split(x, y + half, half)
143         split(x + half, y + half, half)
144
145     h, w = residual.shape
146     for y in range(0, h, max_size):
147         for x in range(0, w, max_size):
148             split(x, y, max_size)
149
150     return transforms
151
152
153 def _pifs_decode(transforms, orig_shape, iters=
8, zoom=2):
154     orig_h, orig_w = orig_shape
155     h = orig_h * zoom
156     w = orig_w * zoom
157     # residual starts at 0, not 128
158     canvas = np.zeros((h, w), dtype=np.float64)
159     for _ in range(iters):
160         nxt = np.zeros((h, w), dtype=np.float64)
161
162         for t in transforms:
163             rx = t['rx'] * zoom
164             ry = t['ry'] * zoom
165             rsz = t['size'] * zoom
166             dx = t['dx'] * zoom
167             dy = t['dy'] * zoom
168             dsz = t['size'] * 2 * zoom
169             D_large = canvas[dy:dy+dsz, dx:dx+
dsz]

```

```

169         D_small = cv2.resize(D_large.astype
170                                e(np.float32), (rsz, rsz), interpolation=cv2.IN
171                                TER_AREA).astype(np.float64)
172         D_trans = _isometries(D_small)[t['
173         dir']]
174         nxt[ry:ry+rsz, rx:rx+rsz] = t['s']
175         * D_trans + t['o']
176
177         canvas = nxt
178
179         return canvas
180
181     # -----
182     # Public API
183     # -----
184
185     def encode_hybrid(image, fif_roughness=0.5, pif
186                       s_max=16, pifs_min=2, pifs_theta=3.0):
187         t0 = time.perf_counter()
188         print("[hybrid] step 1: FIF base layer")
189         img_f = image.astype(np.float64)
190         fif_data = _fif_encode(img_f, max_roughne
191                               ss=fif_roughness)
192         fif_lr = _fif_decode(fif_data, iters=8,
193                              zoom=1)
194         residual = img_f - fif_lr
195         print("[hybrid] step 2: PIFS residual layer
196               ")
197         pifs_data = _pifs_encode_qt(
198             residual,
199             max_size=pifs_max, min_size=pifs_min,

```

```

193         threshold=pifs_theta,
194         search_radius=32, domain_step=4
195     )
196
197     elapsed = time.perf_counter() - t0
198     print(f"[hybrid] done in {elapsed:.2f}s, {l
199           en(pifs_data)} residual blocks")
200
201     return {
202         'fif_data': fif_data,
203         'pifs_data': pifs_data,
204         'metadata': {'original_shape': img_f.s
205                      hape}
206     }
207
208     def decode_hybrid(hybrid_data, iterations=8, zo
209                       om_factor=2):
210         print(f"[hybrid] decoding at zoom x{zoom_fa
211               ctor}")
212         orig_shape = hybrid_data['metadata']['origi
213                                   nal_shape']
214         fif_hr = _fif_decode(hybrid_data['fif_data
215                               '], iters=iterations, zoom=zoom_factor)
216         pifs_hr = _pifs_decode(hybrid_data['pifs_da
217                                 ta'], orig_shape, iters=iterations, zoom=zoom_fa
218                                   ctor)
219         result = np.clip(fif_hr + pifs_hr, 0, 255).
220                       astype(np.uint8)
221         print("[hybrid] done")
222         return result

```

```
1 import os
2 import cv2
3 import numpy as np
4 import csv
5 import time
6 from concurrent.futures import ProcessPoolExecutor
7
8 from skimage.metrics import structural_similarity as calculate_ssim
9 from skimage.metrics import peak_signal_noise_ratio as calculate_psnr
10
11 SAMPLE_SIZE_PER_CAT = 5
12 TEST_RESOLUTION = 256
13 ZOOM_TARGET = 2
14
15 # PIFS quadtree parameters
16 PIFS_MAX_SIZE = 16
17 PIFS_MIN_SIZE = 2
18 PIFS_THRESHOLD = 0.5
19 PIFS_SEARCH_RADIUS = 64
20
21 FIF_ROUGHNESS = 0.5
22
23 DATASET_BASE_PATH = r"C:\Users\hp\Desktop\TIPE\Datasets"
24 OUTPUT_CSV = "experiment_1_results_new_new.csv"
25 OUTPUT_DIR = "exp1_visual_proofs_new_new"
26
27 def get_images(folder, n):
28     extensions = ('.jpg', '.jpeg', '.png')
29     found = []
```

```
31     for root, dirs, files in os.walk(folder):
32         for f in files:
33             if f.lower().endswith(extensions):
34                 found.append(os.path.join(root,
35                                         f))
36
37         if len(found) == n:
38             return found
39
40 def process_single_image(args):
41     category, img_path, idx, pifs_max, pifs_min,
42     pifs_thresh, pifs_radius, fif_roughness, res,
43     zoom, output_dir = args
44     filename = os.path.basename(img_path)
45     img_raw = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
46     img_gt = cv2.resize(img_raw, (res, res), interpolation=cv2.INTER_AREA)
47     low_res = res // zoom
48     img_degraded = cv2.resize(img_gt, (low_res, low_res), interpolation=cv2.INTER_AREA)
49     img_bicubic = cv2.resize(img_degraded, (res, res), interpolation=cv2.INTER_CUBIC)
50
51     import PIFS
52     import FIF
53
54     encoded_pifs = PIFS.encode_pifs(
55         img_degraded,
56         max_size=pifs_max,
57         min_size=pifs_min,
58         threshold=pifs_thresh,
59         search_radius=pifs_radius
```

```

58     )
59     img_pifs = PIFS.decode_pifs(encoded_pifs, i
iterations=8, zoom_factor=zoom)
60
61     encoded_fif = FIF.encode_fif(img_degraded,
max_roughness=fif_roughness)
62     img_fif = FIF.decode_fif(encoded_fif, itera
tions=8, zoom_factor=zoom)
63
64     psnr_bic = calculate_psnr(img_gt, img_bicub
ic, data_range=255)
65     ssim_bic = calculate_ssim(img_gt, img_bicub
ic, data_range=255)
66
67     psnr_pifs = calculate_psnr(img_gt, img_pifs
, data_range=255)
68     ssim_pifs = calculate_ssim(img_gt, img_pifs
, data_range=255)
69
70     psnr_fif = calculate_psnr(img_gt, img_fif,
data_range=255)
71     ssim_fif = calculate_ssim(img_gt, img_fif,
data_range=255)
72
73     if idx == 0:
74         os.makedirs(output_dir, exist_ok=True)
75         cv2.imwrite(os.path.join(output_dir, f"
{category}_1_GroundTruth.png"), img_gt)
76         cv2.imwrite(os.path.join(output_dir, f"
{category}_2_Degraded.png"), img_degraded)
77         cv2.imwrite(os.path.join(output_dir, f"
{category}_3_Bicubic.png"), img_bicubic)
78         cv2.imwrite(os.path.join(output_dir, f"
{category}_4_PIFS.png"), img_pifs)

```

```

79         cv2.imwrite(os.path.join(output_dir, f"
{category}_5_FIF.png"), img_fif)
80
81     return [category, filename, psnr_bic, ssim_
bic, psnr_pifs, ssim_pifs, psnr_fif, ssim_fif]
82
83
84 def run_experiment():
85     print("Démarrage de l'expérience 1...")
86     print(f"Dataset : {DATASET_BASE_PATH}")
87     print(f"Chargement de {SAMPLE_SIZE_PER_CAT}
images par catégorie...\n")
88
89     categories = {
90         "Brain": get_images(os.path.join(DATASE
T_BASE_PATH, "brainMRI"), SAMPLE_SIZE_PER_CAT),
91         "Lung": get_images(os.path.join(DATASE
T_BASE_PATH, "lungCT"), SAMPLE_SIZE_PER_CAT),
92         "Spine": get_images(os.path.join(DATASE
T_BASE_PATH, "spineXray"), SAMPLE_SIZE_PER_CAT)
93     }
94
95     for cat, paths in categories.items():
96         print(f" {cat} : {len(paths)} images t
rouvées")
97
98     tasks = []
99     for category, image_paths in categories.ite
ms():
100         for idx, img_path in enumerate(image_pa
ths):
101             tasks.append((
102                 category, img_path, idx,

```

```

103         PIFS_MAX_SIZE, PIFS_MIN_SIZE, P
IFS_THRESHOLD, PIFS_SEARCH_RADIUS,
104         FIF_ROUGHNESS, TEST_RESOLUTION,
ZOOM_TARGET, OUTPUT_DIR
105     ))
106
107     print(f"\nTraitement de {len(tasks)} images
en parallèle...")
108     start_time = time.time()
109
110     csv_data = []
111     headers = ["Category", "Filename", "Bicubic
_PSNR", "Bicubic_SSIM", "PIFS_PSNR", "PIFS_SSIM
", "FIF_PSNR", "FIF_SSIM"]
112     stats = {
113         "Bicubic": {"psnr": [], "ssim": []},
114         "PIFS": {"psnr": [], "ssim": []},
115         "FIF": {"psnr": [], "ssim": []}
116     }
117
118     with ProcessPoolExecutor() as executor:
119         for res in executor.map(process_single_
image, tasks):
120             csv_data.append(res)
121             print(f" {res[1]} ({res[0]}) - ter
miné")
122
123             stats["Bicubic"]["psnr"].append(res
[2])
124             stats["Bicubic"]["ssim"].append(res
[3])
125             stats["PIFS"]["psnr"].append(res[4]
)

```

```

126         stats["PIFS"]["ssim"].append(res[5]
)
127         stats["FIF"]["psnr"].append(res[6])
128         stats["FIF"]["ssim"].append(res[7])
129
130     elapsed = time.time() - start_time
131     print(f"\nCalculs terminés en {elapsed:.2f}
s\n")
132
133     print("Résultats (moyennes globales)")
134     print("-" * 40)
135     for algo in ["Bicubic", "PIFS", "FIF"]:
136         avg_psnr = np.mean(stats[algo]["psnr"])
137         avg_ssim = np.mean(stats[algo]["ssim"])
138         print(f" {algo}")
139         print(f"      PSNR : {avg_psnr:.2f} dB")
140         print(f"      SSIM : {avg_ssim:.4f}")
141     print("-" * 40)
142
143     with open(OUTPUT_CSV, mode='w', newline='')
as f:
144         writer = csv.writer(f)
145         writer.writerow(headers)
146         writer.writerows(csv_data)
147
148     print(f"\nDonnées exportées dans '{OUTPUT_C
SV}'")
149     print(f"Images sauvegardées dans '{OUTPUT_D
IR}'")
150
151
152 if __name__ == "__main__":
153     run_experiment()

```

```
1 import pandas as pd
2 import matplotlib.pyplot as plt
3 import numpy as np
4
5 CSV_FILE = "experiment_1_results.csv"
6
7 COLORS = ['#E63946', '#2A9D8F', '#457B9D']
8 LABELS = ['Bicubique', 'PIFS', 'FIF']
9
10 def generate_bar_charts(csv_path):
11     df = pd.read_csv(csv_path)
12
13     psnr_means = [df['Bicubic_PSNR'].mean(), df
14 ['PIFS_PSNR'].mean(), df['FIF_PSNR'].mean()]
15     ssim_means = [df['Bicubic_SSIM'].mean(), df
16 ['PIFS_SSIM'].mean(), df['FIF_SSIM'].mean()]
17
18     plt.style.use('seaborn-v0.8-whitegrid')
19     fig, (ax1, ax2) = plt.subplots(1, 2, figsize
20 e=(14, 6))
21     fig.suptitle("Résultats de l'Expérience 1 :
22 Évaluation Quantitative (Moyennes globales)",
23     fontsize=16, fontweight='bold'
24 )
25
26     bars1 = ax1.bar(LABELS, psnr_means, color=C
27 OLORS, edgecolor='black', linewidth=1.2, width=
28 0.6)
29
30     ax1.set_title("Peak Signal-to-Noise Ratio (
31 PSNR)\n↑ Plus haut = Moins d'erreur absolue", f
32 ontsize=12, pad=15)
33
34     ax1.set_ylabel("PSNR (Décibels)", fontsize=
35 12)
```

```
25     ax1.set_ylim(0, max(psnr_means) * 1.15)
26     for bar in bars1:
27         yval = bar.get_height()
28         ax1.text(bar.get_x() + bar.get_width()
29 / 2, yval + 0.5,
30                 f"{yval:.2f} dB", ha='center',
31                 va='bottom', fontsize=11, fontweight='bold')
32
33     bars2 = ax2.bar(LABELS, ssim_means, color=C
34 OLORS, edgecolor='black', linewidth=1.2, width=
35 0.6)
36
37     ax2.set_title("Structural Similarity Index
38 (SSIM)\n↑ 1.0 = Structure identique au Ground T
39 ruth", fontsize=12, pad=15)
40
41     ax2.set_ylabel("Score SSIM (0 à 1)", fontsi
42 ze=12)
43
44     ax2.set_ylim(0, 1.1)
45     for bar in bars2:
46         yval = bar.get_height()
47         ax2.text(bar.get_x() + bar.get_width()
48 / 2, yval + 0.02,
49                 f"{yval:.4f}", ha='center', va
50 ='bottom', fontsize=11, fontweight='bold')
51
52     for ax in [ax1, ax2]:
53         ax.spines['top'].set_visible(False)
54         ax.spines['right'].set_visible(False)
55         ax.spines['left'].set_linewidth(1.2)
56         ax.spines['bottom'].set_linewidth(1.2)
57         ax.tick_params(axis='x', labelsize=11)
58         ax.tick_params(axis='y', labelsize=10)
59         ax.grid(axis='y', linestyle='--', alpha
60 =0.7)
61
62         ax.grid(axis='x', visible=False)
```

```
49     plt.tight_layout()
50     fig.subplots_adjust(top=0.82)
51
52
53     output_filename = "exp1_bar_charts.png"
54     plt.savefig(output_filename, dpi=300, bbox_
inches='tight')
55     print(f"Graphique sauvegardé : {output_file
name}")
56     plt.show()
57
58
59 if __name__ == "__main__":
60     generate_bar_charts(CSV_FILE)
```

```

1 import cv2
2 import matplotlib.pyplot as plt
3 import os
4 import numpy as np
5
6 DIR = "exp1_visual_proofs(for_intensity_profile
7 )"
8 IMG_GT_NAME = "Lung_1_GroundTruth.png"
9 IMG_BICUBIC_NAME = "Lung_3_Bicubic.png"
10 IMG_PIFS_NAME = "Lung_4_PIFS.png"
11 IMG_FIF_NAME = "Lung_5_FIF.png"
12
13 ROW_INDEX = 50
14
15 def plot_intensity_profile():
16     img_gt = cv2.imread(os.path.join(DIR,
17     IMG_GT_NAME), cv2.IMREAD_GRAYSCALE)
18     img_bicubic = cv2.imread(os.path.join(DIR,
19     IMG_BICUBIC_NAME), cv2.IMREAD_GRAYSCALE)
20     img_pifs = cv2.imread(os.path.join(DIR,
21     IMG_PIFS_NAME), cv2.IMREAD_GRAYSCALE)
22     img_fif = cv2.imread(os.path.join(DIR,
23     IMG_FIF_NAME), cv2.IMREAD_GRAYSCALE)
24
25     profile_gt = img_gt[ROW_INDEX, :].astype(np.float32)
26     profile_bicubic = img_bicubic[ROW_INDEX, :].
27     .astype(np.float32)
28     profile_pifs = img_pifs[ROW_INDEX, :].as
29     type(np.float32)
30     profile_fif = img_fif[ROW_INDEX, :].as
31     type(np.float32)
32

```

```

26 x_range = range(120, 180)
27
28 offset_pifs = 20
29 offset_fif = 40
30 offset_bic = 60
31
32 plt.figure(figsize=(12, 6))
33 plt.plot(x_range, profile_gt[x_range],
34         label='Ground Truth',
35         color='black', linewidth=2, linestyle=
36         '--')
37 plt.plot(x_range, profile_bicubic[x_range]
38 + offset_bic, label=f'Bicubique ({offset_bic})
39 ', color='red', linewidth=1.5)
40 plt.plot(x_range, profile_pifs[x_range]
41 + offset_pifs, label=f'PIFS ({offset_pifs})',
42         color='green', linewidth=1.5)
43 plt.plot(x_range, profile_fif[x_range]
44 + offset_fif, label=f'FIF ({offset_fif})',
45         color='blue', linewidth=1.5)
46
47 plt.title("Porfil d'intensité 1D")
48 plt.xlabel("Position X (pixels)")
49 plt.ylabel("Intensité (0-255)")
50 plt.legend()
51 plt.grid(True, linestyle=':', alpha=0.7)
52 plt.tight_layout()
53 plt.show()
54
55 if __name__ == "__main__":
56     plot_intensity_profile()

```

```
1 import os
2 import cv2
3 import numpy as np
4 import csv
5 import time
6 from concurrent.futures import ProcessPoolExecutor
7
8 from skimage.metrics import structural_similarity as calculate_ssim
9
10 SAMPLE_SIZE_PER_CAT = 1
11 TEST_RESOLUTION = 512
12 ZOOM_TARGET = 4
13
14 PIFS_RANGE_SIZES = [16, 8, 4, 2]
15 FIF_ROUGHNESS = 0.5
16
17 DATASET_BASE_PATH = r"C:\Users\hp\Desktop\TIPE\Datasets"
18 OUTPUT_CSV = "experiment_2_raw_data.csv"
19
20 def get_images(folder, n):
21     extensions = ('.jpg', '.jpeg', '.png')
22     found = []
23     for root, dirs, files in os.walk(folder):
24         for f in files:
25             if f.lower().endswith(extensions):
26                 found.append(os.path.join(root,
27 f))
28                 if len(found) == n:
29                     return found
30     return found
```

```
31
32
33 def evaluate_algorithm(task):
34     category, img_path, res, zoom, algo_type, param = task
35     filename = os.path.basename(img_path)
36
37     img_raw = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
38     img_gt = cv2.resize(img_raw, (res, res), interpolation=cv2.INTER_AREA)
39
40     low_res = res // zoom
41     img_degraded = cv2.resize(img_gt, (low_res, low_res), interpolation=cv2.INTER_AREA)
42
43     start_time = time.perf_counter()
44
45     if algo_type == 'BICUBIC':
46         img_recon = cv2.resize(img_degraded, (res, res), interpolation=cv2.INTER_CUBIC)
47
48     elif algo_type == 'PIFS':
49         import PIFS_old
50         encoded = PIFS.encode_pifs(img_degraded, range_size=param)
51         img_recon = PIFS.decode_pifs(encoded, iterations=8, zoom_factor=zoom)
52
53     elif algo_type == 'FIF':
54         import FIF
55         encoded = FIF.encode_fif(img_degraded, max_roughness=param)
```

```

56         img_recon = FIF.decode_fif(encoded, iterations=8, zoom_factor=zoom)
57
58         duration = time.perf_counter() - start_time
59         ssim_val = calculate_ssim(img_gt, img_recon, data_range=255)
60
61         return (category, filename, algo_type, param, duration, ssim_val)
62
63     def run_experiment():
64         print("Démarrage de l'expérience 2 : Frontière de Pareto")
65         print(f"Chargement de {SAMPLE_SIZE_PER_CAT} image(s) par anatomie...\n")
66
67         categories = {
68             "Brain": get_images(os.path.join(DATABASE_PATH, "brainMRI"), SAMPLE_SIZE_PER_CAT),
69             "Lung": get_images(os.path.join(DATABASE_PATH, "lungCT"), SAMPLE_SIZE_PER_CAT),
70             "Spine": get_images(os.path.join(DATABASE_PATH, "spineXray"), SAMPLE_SIZE_PER_CAT)
71         }
72
73         tasks = []
74         for cat, paths in categories.items():
75             for img_path in paths:
76                 tasks.append((cat, img_path, TEST_RESOLUTION, ZOOM_TARGET, 'BICUBIC', 0))
77                 tasks.append((cat, img_path, TEST_RESOLUTION, ZOOM_TARGET, 'FIF', FIF_ROUGHNESS))
78             for r in PIFS_RANGE_SIZES:

```

```

80                 tasks.append((cat, img_path, TEST_RESOLUTION, ZOOM_TARGET, 'PIFS', r))
81
82                 print(f"Lancement de {len(tasks)} benchmark s en parallèle...")
83
84                 raw_csv_data = []
85                 headers = ["Category", "Filename", "Algorithm", "Parameter", "Execution_Time_Sec", "SSIM"]
86                 results = []
87
88                 max_workers = max(1, os.cpu_count() - 1)
89                 with ProcessPoolExecutor(max_workers=max_workers) as executor:
90                     for res in executor.map(evaluate_algorithm, tasks):
91                         results.append(res)
92                         cat, filename, algo, param, duration, ssim_val = res
93
94                     param_str = f"R={param}" if algo == "PIFS" else (f"s={param}" if algo == "FIF" else "N/A")
95                     print(f" {cat} | {algo:7} ({param_str:5}) | {filename[:15]}... | {duration:.4f}s")
96
97                     raw_csv_data.append([cat, filename, algo, param_str, duration, ssim_val])
98
99                     with open(OUTPUT_CSV, mode='w', newline='') as f:
100                         writer = csv.writer(f)
101                         writer.writerow(headers)

```

```
102         writer.writerow(raw_csv_data)
103
104     print(f"\nDonnées exportées dans '{OUTPUT_CSV}'")
105
106     # Compute and display Pareto averages
107     pareto_points = {}
108     for cat, filename, algo, param, duration, s
sim_val in results:
109         key = (algo, param)
110         if key not in pareto_points:
111             pareto_points[key] = {'time': [], '
ssim': []}
112         pareto_points[key]['time'].append(durat
ion)
113         pareto_points[key]['ssim'].append(ssim_
val)
114
115     print("\nMoyennes – Frontière de Pareto")
116     print(f"{'Algorithme':<10} | {'Paramètre':<
10} | {'Temps moyen (s)':<17} | SSIM moyen")
117     print("-" * 60)
118     for (algo, param), data in pareto_points.it
ems():
119         mean_time = np.mean(data['time'])
120         mean_ssime = np.mean(data['ssim'])
121         param_str = f"R={param}" if algo == "PI
FS" else (f"s={param}" if algo == "FIF" else "N
/A")
122         print(f"{algo:<10} | {param_str:<10} |
{mean_time:<17.4f} | {mean_ssime:<17.4f}")
123
124
125 if __name__ == "__main__":
```

```
126     run_experiment()
```

```
1 import pandas as pd
2 import matplotlib.pyplot as plt
3
4 CSV_FILE = "experiment_2_raw_data(256).csv"
5
6 def generate_pareto_chart(csv_path):
7     df = pd.read_csv(csv_path)
8     df['Parameter'] = df['Parameter'].fillna('N/A')
9
10     grouped = df.groupby(['Algorithm', 'Parameter'], as_index=False)[['Execution_Time_Sec', 'SSIM']].mean()
11
12     bicubic_data = grouped[grouped['Algorithm'] == 'BICUBIC'].iloc[0]
13     fif_data = grouped[grouped['Algorithm'] == 'FIF'].iloc[0]
14     pifs_data = grouped[grouped['Algorithm'] == 'PIFS'].copy()
15
16     pifs_data['R_val'] = pifs_data['Parameter'].str.extract(r'(\d+)').astype(int)
17     pifs_data = pifs_data.sort_values('R_val', ascending=False)
18
19     plt.style.use('seaborn-v0_8-whitegrid')
20     fig, ax = plt.subplots(figsize=(11, 6.5))
21
22     ax.scatter(bicubic_data['Execution_Time_Sec'], bicubic_data['SSIM'],
23               color='#E63946', s=160, zorder=5, label='Bicubique')
```

```
25     ax.annotate(
26         f"Bicubique\n({bicubic_data['Execution_Time_Sec']*1000:.1f} ms, SSIM: {bicubic_data['SSIM']:.4f})",
27         xy=(bicubic_data['Execution_Time_Sec'], bicubic_data['SSIM']),
28         xytext=(10, 5), textcoords='offset points',
29         fontsize=10, fontweight='bold', color='#E63946', ha='left'
30     )
31
32     ax.scatter(fif_data['Execution_Time_Sec'], fif_data['SSIM'],
33               color='#457B9D', marker='*', s=260, zorder=5, label='FIF')
34     ax.annotate(
35         f"FIF ($s=0.5$)\n({fif_data['Execution_Time_Sec']:.2f} s, SSIM: {fif_data['SSIM']:.4f})",
36         xy=(fif_data['Execution_Time_Sec'], fif_data['SSIM']),
37         xytext=(0, -25), textcoords='offset points',
38         fontsize=10, fontweight='bold', color='#457B9D', ha='center'
39     )
40
41     ax.plot(pifs_data['Execution_Time_Sec'], pifs_data['SSIM'],
42           color='#2A9D8F', linestyle='-', linewidth=2.5, marker='o', markersize=8,
43           zorder=4, label='PIFS (Grille Fixe Classique)')
```

```
44
45 offsets = {
46     'R=16': (0, 10),
47     'R=8': (0, 10),
48     'R=4': (-15, -20),
49     'R=2': (12, 8)
50 }
51
52 for _, row in pifs_data.iterrows():
53     p_name = row['Parameter']
54     ox, oy = offsets.get(p_name, (0, 10))
55     ha_val = 'left' if p_name == 'R=2' else
56     ('right' if p_name == 'R=4' else 'center')
57     ax.annotate(
58         f"{p_name}\n({row['Execution_Time_Sec']:.1f} s, SSIM: {row['SSIM']:.4f})",
59         xy=(row['Execution_Time_Sec'], row['SSIM']),
60         xytext=(ox, oy), textcoords='offset
61         points',
62         fontsize=9.5, fontweight='semibold',
63         color='#1b655c', ha=ha_val
64     )
65
66     ax.set_xscale('log')
67     ax.set_title("Frontière de Pareto : Compromis Temps de Calcul vs. Fidélité Structurale (SSIM)",
68                 fontsize=14, fontweight='bold',
69                 pad=25)
70     ax.set_xlabel("Temps d'exécution moyen (secondes) - Échelle Logarithmique", fontsize=12, labelpad=12)
```

```
67     ax.set_ylabel("Indice de Similarité Structurale Moyen (SSIM)", fontsize=12, labelpad=12)
68     ax.set_ylim(0.2, 1.05)
69
70     ax.spines['top'].set_visible(False)
71     ax.spines['right'].set_visible(False)
72     ax.spines['left'].set_linewidth(1.2)
73     ax.spines['bottom'].set_linewidth(1.2)
74     ax.grid(True, which="both", linestyle='--', alpha=0.5)
75     ax.legend(loc='upper right', frameon=True, facecolor='white', edgecolor='gray', framealpha=0.9, fontsize=11)
76
77     plt.tight_layout()
78     plt.show()
79
80
81 if __name__ == "__main__":
82     generate_pareto_chart(CSV_FILE)
```

```
1 import os
2 import cv2
3 import numpy as np
4 import PIFS
5
6 INPUT_IMAGE_PATH = "C:/Users/hp/Desktop/TIPE/code/exp1_visual_proofs(for_intensity_profile)/Sp
   ine_1_GroundTruth.png"
7 OUTPUT_DIR = "C:/Users/hp/Desktop/TIPE/code/sna
   pshots"
8
9 TOTAL_ITERATIONS = 5
10 NUM_SNAPSHOTS = 4
11
12 MAX_SIZE = 16
13 MIN_SIZE = 2
14 THRESHOLD = 10.0
15 SEARCH_RADIUS = 8
16
17
18 def main():
19     os.makedirs(OUTPUT_DIR, exist_ok=True)
20
21     original_img = cv2.imread(INPUT_IMAGE_PATH,
22                               cv2.IMREAD_GRAYSCALE)
23     h, w = original_img.shape
24
25     print("Encodage PIFS en cours...")
26     fractal_data = PIFS.encode_pifs(
27         original_img,
28         max_size=MAX_SIZE,
29         min_size=MIN_SIZE,
30         threshold=THRESHOLD,
31         search_radius=SEARCH_RADIUS
```

```
31     )
32     transformations = fractal_data['transformat
   ions']
33     print("Encodage terminé.")
34
35     # Uniformly sample iteration indices from 0
   to TOTAL_ITERATIONS
36     # e.g. for 5 iterations and 4 snapshots: [0
   , 1, 3, 5]
37     snapshots_to_save = np.linspace(0, TOTAL_IT
   ERATIONS, NUM_SNAPSHOTS, dtype=int)
38     print(f"Iérations ciblées : {snapshots_to_
   save}")
39
40     canvas = np.random.randint(0, 256, (h, w)).
   astype(np.float64)
41     file_index = 1
42
43     if 0 in snapshots_to_save:
44         path = os.path.join(OUTPUT_DIR, f"snap
   hot_{file_index}_iter_0.png")
45         cv2.imwrite(path, canvas.astype(np.uint
   8))
46         print(f"Sauvegardé : {path}")
47         file_index += 1
48
49     for i in range(1, TOTAL_ITERATIONS + 1):
50         next_canvas = np.zeros((h, w), dtype=np
   .float64)
51
52         for code in transformations:
53             rx, ry, r_size = code['rx'], code['
   ry'], code['size']
54             dx, dy = code['dx'], code['dy']
```

```
55         d_size = code['size'] * 2
56
57         D_large = canvas[dy:dy + d_size, dx
58 :dx + d_size]
59         D_shrunk = cv2.resize(D_large, (r_s
60 ize, r_size), interpolation=cv2.INTER_AREA)
61
62         isometries = PIFS.get_isometries(D_
63 shrunk)
64         D_trans = isometries[code['dir']]
65         R_rebuilt = code['s'] * D_trans + c
66         ode['o']
67
68         next_canvas[ry:ry + r_size, rx:rx +
69 r_size] = R_rebuilt
70
71         canvas = next_canvas
72
73         if i in snapshots_to_save:
74             path = os.path.join(OUTPUT_DIR, f"s
75 napsnot_{file_index}_iter_{i}.png")
76             cv2.imwrite(path, np.clip(canvas, 0
77 , 255).astype(np.uint8))
78             print(f"Sauvegardé : {path}")
79             file_index += 1
80
81         print("Snapshots générés avec succès.")
82
83 if __name__ == "__main__":
84     main()
```