README — `SobelxFIMF.py`

Qu'est-ce que fait ce script ?

Pipeline simple "lissage → accents de contours" pour TP de vision :

1. **FIMF-like** (médiane isotrope) pour **réduire le bruit** sans casser les bords.
2. **Sobel** pour **accentuer/détecter les contours** (magnitude du gradient).

Il lit une image (ou génère une **démo synthétique** si aucun chemin n'est donné), applique les deux étapes et **sauvegarde** les résultats dans un dossier de sortie.

Pourquoi ce choix d'algos ?

- **FIMF-like** : médiane avec **noyau circulaire** (isotrope) → lissage robuste au bruit impulsionnel, bords mieux préservés qu'un noyau carré.
- **Sobel** : gradient **rapide et stable** (X/Y + magnitude) → visualisation claire des transitions après lissage.

Fichiers générés

Dans `--outdir` (par défaut `TP6/`) :

- `fimf_r{R}.png` — image lissée (FIMF-like, rayon `R`).
- `sobel_mag_r{R}_k{K}.png` — magnitude du gradient (contours).
- `sobel_gx_r{R}_k{K}.png` — composante **Gx** (viz).
- `sobel_gy_r{R}_k{K}.png` — composante **Gy** (viz).
- `image.png` — (uniquement en mode démo) l'image synthétique d'entrée.

Dépendances (requirements)

- **Python 3.9+** recommandé
- **Obligatoire** : `numpy`, `opencv-python`
- **Optionnel (plus rapide, meilleur isotrope)** : `scikit-image`

Installation rapide :

```
pip install numpy opencv-python scikit-image
```

> Si `scikit-image` est absent, le script bascule automatiquement sur OpenCV (noyau carré) ou un fallback NumPy vectorisé.

Utilisation

```
# 1) Avec une image
python fimf_sobel_pipeline.py path/to/image.png --radius 1 --sobel-ksize 3 --
outdir TP6

# 2) Mode démo (génère une image synthétique si aucun chemin n'est fourni)
python fimf_sobel_pipeline.py --radius 2 --sobel-ksize 3
```

Paramètres

- `image` *(optionnel)* : chemin vers l'image d'entrée. Si omis → démo synthétique.
- `--radius` *(int, défaut 1)* : rayon du noyau circulaire du FIMF-like.
  - `1` ≈ **3×3**, `2` ≈ **5×5**, `3` ≈ **7×7**…
- `--sobel-ksize` *(3,5,7 ; défaut 3)* : taille du noyau Sobel.
- `--outdir` *(str, défaut `TP6`)* : dossier de sortie.

Exemples rapides :

```
# Lissage doux + contours nets
python fimf_sobel_pipeline.py img.jpg --radius 1 --sobel-ksize 3

# Lissage plus fort (scènes bruitées)
python fimf_sobel_pipeline.py img.jpg --radius 2 --sobel-ksize 3

# Contours encore plus “épais”
python fimf_sobel_pipeline.py img.jpg --radius 1 --sobel-ksize 5
```

Détails d'implémentation

Étape 1 — FIMF-like (fonction `fimf`)

- Essaie, **dans cet ordre** :
  1. `skimage.filters.rank.median` avec **footprint circulaire** (`disk(radius)`) → **isotrope & rapide**.
  2. `cv2.medianBlur(ksize)` (ksize impair ≥ 3) → **approximation** (noyau **carré**).
  3. **Fallback NumPy vectorisé** (circular mask + `sliding_window_view`) → isotrope, plus lent.
- Conversion interne **uint8** pour les chemins rapides, remise à l'échelle vers le type d'origine en sortie.

Étape 2 — Sobel (fonction `sobel_magnitude`)

- Calcule **Gx** et **Gy** (Sobel), puis la magnitude `sqrt(Gx² + Gy²)`.
- Normalisation **0–255** pour un rendu direct en PNG.

Schéma rapide du flux

```
Image (ou démo) ⟶ FIMF-like (radius R) ⟶ Sobel (ksize K) ⟶ Sauvegarde
(fimf/grad/Gx/Gy)
```

Résultats attendus (interprétation)

- `fimf_r*.png` : image **plus lisse**, **bords préservés** (meilleur avec scikit-image actif).
- `sobel_mag_*.png` : **contours nets**, prêts pour seuillage (ex. Otsu) si besoin.
- `sobel_gx_* / sobel_gy_*` : diagnostics utiles pour le rapport (direction des gradients).

Conseils pratiques

- Active `scikit-image` pour la **vitesse** et la **vraie isotropie**.
- Sur images très bruitées : `--radius 2`. Sur images fines : `--radius 1`.
- Si la magnitude est trop “faible” → augmente `--sobel-ksize` (5 ou 7).
- Pour binariser les contours : ajoute un seuillage (p.ex. Otsu) après la magnitude.

Structure de repo (suggestion)

```
.
├─ image.png
└─ TP6/
   ├─ SobelxFIMF.py                  # si démo
   ├─ fimf_r1.png
   ├─ sobel_mag_r1_k3.png
   ├─ sobel_gx_r1_k3.png
   └─ sobel_gy_r1_k3.png
```

Crédit scientifique (contexte)

- **FIMF** : approche de **filtrage médian isotrope rapide** (noyau circulaire + optimisations).
- **Sobel** : opérateur de gradient classique (X/Y), robuste et léger pour TP.