TNI & VO

Traitement Numérique des Images et Vision par Ordinateur

Fondements, Algorithmes et Applicatións Intégrées

AUTEUR: FRANCISCO DE ASSIS ZAMPIROLLI

Traitement Numérique d'Images & Vision par Ordinateur

Livre interactif avec C++

Francisco de Assis Zampirolli

Universidade Federal do ABC

20 septembre 2026

Conception graphique : Francisco de Assis Zampirolli
Couverture : Francisco de Assis Zampirolli, avec le soutien de l'IA (Gemini et ChatGPT)

CATALOGAGE À LA SOURCE
SYSTÈME DES BIBLIOTHÈQUES DE L'UNIVERSITÉ FÉDÉRALE DE L'ABC

[CODE CUTTER] Zampirolli, Francisco de Assis
Traitement Numérique d'Images et Vision par Ordinateur / Francisco de Assis Zampirolli – Santo André, SP : Édition de l'Auteur, 2026.

[xx], [NNN] p. : ill.

ISBN : [ISBN À DÉFINIR] (1ère Édition)

1. Traitement Numérique d'Images. 2. Vision par Ordinateur. 3. Morphologie Mathématique. 4. Python. 5. Correction Automatique. 6. Moodle. I. Titre.

CDD [XX] éd. – [CODE CDD]

Table des matières

Liste des figures

Liste des tableaux

TNI & VO — C++

Bienvenue dans le manuel TNI et Vision par Ordinateur — version C++ / Français.

Organisation

- **Partie I — Fondements du TNI** — Représentation, histogrammes, filtrage, morphologie
- **Partie II — Vision par Ordinateur** — Segmentation, descripteurs, détection, apprentissage profond

---

Préface

Ce livre constitue une **œuvre en permanente actualisation** dans les domaines du **Traitement Numérique d'Images (TNI)** et de la **Vision par Ordinateur (VO)**, conçue comme matériel didactique interactif pour les cours de premier et de deuxième cycle en Informatique, Ingénierie et domaines connexes.

> **!** Point important
>
> L'ouvrage repose sur la méthodologie décrite dans Zampirolli et al. (2025) — extension de Zampirolli et al. (2024), travail primé dans la filière **Ressources et Environnements Éducatifs** de l'**EduComp 2024**. Le contenu intègre la bibliothèque `morph.py`, développée par l'auteur.

Ce n'est pas un livre statique. Son contenu évolue continuellement au fur et à mesure que les exemples sont améliorés, que de nouvelles sections sont incorporées et que les approches pédagogiques sont affinées en fonction de l'expérience d'utilisation et des retours des étudiants et des enseignants. De cette manière, l'ouvrage est traité comme un *projet en constante évolution*, cherchant à accompagner à la fois les avancées technologiques et les meilleures pratiques d'enseignement en TNI et VO.

Contexte de la première édition

Le contenu de cette édition a été élaboré entre mai et août 2026, lors de la première offre du cours de Traitement Numérique d'Images basé sur ce matériel didactique. Le cours a été dispensé dans deux groupes de premier cycle — majoritairement composés d'étudiants en Informatique, en période matinale — et dans un groupe de deuxième cycle en Informatique, tous à l'UFABC. Cette première édition représente le résultat de cette offre initiale, consolidant le contenu développé, testé et continuellement perfectionné tout au long de la période académique.

La méthodologie d'enseignement adoptée a suivi une séquence structurée à chaque cours. Initialement, une courte vidéo était diffusée, d'une durée moyenne de 7 minutes, générée dans **Gemini Notebook**, alternant entre la présentation de la partie conceptuelle du chapitre et la résolution des Exercices de Programmation (EPs). Dans ce second cas, presque tous les EPs étaient résolus pendant le cours lui-même. Ensuite, un ensemble d'environ 12 *diapositives*, également générées dans **Gemini Notebook**, était présenté, ayant comme sources le PDF complet du livre et le fichier `morph.py`. Ces *diapositives* étaient produites à partir d'un *prompt* spécifique pour la génération du contenu théorique et pratique de chaque chapitre.

Après cette étape initiale, il restait environ 100 minutes de cours pour l'exploration des deux *notebooks* Colab du chapitre, l'un théorique et l'autre pratique, avec un accent sur les simulateurs interactifs et l'exécution des blocs de code, permettant aux étudiants de modifier les paramètres et d'observer leurs effets. Dans les cours réalisés en laboratoire, les étudiants transféraient les réponses des EPs développées dans Colab directement vers les activités VPL de Moodle, où elles étaient soumises à l'évaluation automatique. Ce processus de publication et d'utilisation des EPs est présenté à la fin de cette préface.

L'évaluation continue a inclus des simulations bimensuelles, appliquées avec le **SEB (*Safe Exam Browser*)** sur Moodle, contenant des EPs similaires à ceux des listes d'exercices. Chaque simulation accordait un bonus de 5 % sur la note finale. Les deux examens du cours utilisaient également le SEB, mais étaient composés de questions paramétrées générées dans **MCTest**, dont la description combine LaTeX et des paramètres au format `[[code:variable]]`, définis dans des extraits de Python délimités par `[[def: ... ]]` dans la question elle-même. Ce processus a permis de générer 110 variations d'examen, tirées individuellement parmi les étudiants.

L'Annexe A détaille la création de ces questions dans MCTest et leur exportation vers Moodle ; l'Annexe B décrit la configuration des activités VPL, y compris le processus de publication des EPs ; l'Annexe C présente la configuration du SEB pour les simulations et les examens ; l'Annexe D décrit la génération des vidéos et *diapositives* utilisées dans les cours avec **Gemini Notebook** ; et l'Annexe E détaille l'utilisation de l'**Intelligence Artificielle (IA)** dans la génération de *feedback* socratique pour les activités formatives et évaluatives, complétant la correction automatique réalisée par le VPL.

Comment ce livre est produit

Le contenu de ce livre est développé dans **Quarto** et stocké dans le dossier `all` du dépôt **github.com/fzampirolli/pdi-vc**. À partir d'un code source unique, le livre est automatiquement généré et publié dans différents formats, présentés dans le Table 1.

Bien que l'édition en PDF enregistre l'état de l'œuvre à la fin de la première offre du cours en 2026, le développement du livre se poursuit de manière permanente. À chaque mise à jour du dépôt GitHub, de nouvelles versions des pages HTML, du PDF et des notebooks sont automatiquement générées, mettant immédiatement les améliorations à la disposition des lecteurs.

Tableau 1: Différents formats de publication générés automatiquement à partir du même code source.

Format	Description
HTML	Version pour le *web*, avec simulateurs interactifs et navigation entre les chapitres : fzampirolli.github.io/pdi-vc/
PDF	Version adaptée à l'impression ou à la lecture *hors ligne* : livro.pt.py.pdf
Notebooks (`.ipynb`)	Compatibles avec **Jupyter** et **Google Colab**, permettant d'exécuter, de modifier et d'expérimenter les exemples de code et les Exercices de Programmation (EPs). Un filtre personnalisé maintient les références croisées, la numérotation des figures et des tableaux, et formate automatiquement les citations selon la norme ABNT.

L'ouvrage est publié en **dix combinaisons** — chaque piste de code (**Python** et **C++**) dans cinq langues (portugais, anglais, français, espagnol et italien). Avec le cache de traduction déjà rempli, une régénération complète (traduction, réexécution des *notebooks*, rendu HTML et PDF, et publication) prend environ **72 minutes**, avec un parallélisme réel moyen de ~5 processus (pic de 8) ; la **première** génération d'une nouvelle langue, avec un cache vide, est bien plus lente — environ **80 minutes** par combinaison, comme observé lors des premières générations en espagnol et en italien. Le Table 2 résume chaque combinaison (cache déjà rempli) : nombre de pages du PDF et temps de rendu. Ce temps total réel est bien inférieur à la somme des temps de chaque combinaison prise isolément (plus de **5 heures** si elles étaient exécutées une par une) : sur un processeur à plusieurs cœurs, plusieurs combinaisons s'exécutent en même temps, donc le temps total n'est pas la simple somme des temps individuels.

Tableau 2: Nombre de pages du PDF et temps de rendu par combinaison (parallélisme réel moyen de ~5 processus, pic de 8). La combinaison de base Python/Portugais ne fait que rendre les sorties déjà enregistrées dans les *notebooks* source ; les autres réexécutent tout le code.

Combinaison	Piste	Langue	Pages	Temps de rendu
`py.pt`	Python	Portugais (base)	654	~7 min
`py.en`	Python	Anglais	644	~31 min
`py.fr`	Python	Français	666	~31 min
`py.es`	Python	Espagnol	666	~31 min
`py.it`	Python	Italien	658	~31 min
`cpp.pt`	C++	Portugais	434	~26 min
`cpp.en`	C++	Anglais	426	~34 min

Combinaison	Piste	Langue	Pages	Temps de rendu
`cpp.fr`	C++	Français	434	~38 min
`cpp.es`	C++	Espagnol	430	~37 min
`cpp.it`	C++	Italien	428	~35 min

État de validation. Seule la combinaison `py.pt` (Python, portugais) est la source curée : c'est là que l'auteur écrit, révise et valide tout le contenu. Les neuf autres combinaisons — les pistes **C++** et les traductions en anglais, français, espagnol et italien — sont **générées automatiquement**, par traduction via un modèle de langage et transpilation du code, et **nécessitent encore une révision détaillée**. Le point le plus sensible est le **texte incrusté dans certaines figures** : là où la version traduite n'a pas encore été produite, l'image apparaît avec du texte en portugais (chaque figure traduite suit le même suffixe de langue que les autres fichiers générés, par exemple `image_en.png`).

Les pages HTML, le PDF et les *notebooks* disponibles dans le projet reflètent toujours l'état le plus récent du développement. Lorsqu'une **édition officielle** est publiée, elle est identifiée par une *balise* dans le dépôt GitHub, permettant de reproduire exactement le contenu correspondant à cette édition. Ainsi, le lecteur peut suivre à la fois l'évolution continue du projet et récupérer toute édition officielle précédemment publiée.

```
git clone https://github.com/fzampirolli/pdi-vc.git
cd pdi-vc
git checkout <tag-da-versao>
```

Chaque chapitre met à disposition deux boutons **Exécuter Colab** (Figure 1), qui dirigent vers des environnements complémentaires :

1. **Partie Théorique**, située à l'ouverture de chaque chapitre, contenant les concepts présentés et des exemples de code exécutables ;
2. **Partie Pratique**, dans la section **Partie Pratique avec Exercices de Programmation** (EPs), dédiée aux exercices et à leurs simulateurs interactifs.

Figure 1: Bouton cliquable **Exécuter Colab**, disponible au début des parties **Théorique** et **Pratique** de chaque chapitre, permettant l'exécution interactive des exemples et des exercices. Dans la version PDF, il existe un *lien* HTML direct entre l'image du simulateur et sa légende.

La génération des *notebooks* est entièrement automatisée par le *script* `gerar_notebooks_alunos.py`, qui adapte le contenu aux environnements interactifs, permettant son exécution sans nécessité d'installation de Quarto.

Utilisation d'outils d'Intelligence Artificielle

La conception, le projet pédagogique, la structure conceptuelle et le contenu fondamental de ce livre sont de **paternité exclusive de l'auteur**.

Dans le processus d'édition et de soutien au développement, des outils d'**Intelligence Artificielle (IA)**, tels que **ChatGPT**, **Claude**, **DeepSeek** et **Gemini**, ont été utilisés dans leurs versions gratuites, employés strictement comme ressources auxiliaires. Leur utilisation s'est limitée au soutien à la révision et à l'amélioration du style textuel, à l'optimisation de la syntaxe des codes et à la génération assistée d'illustrations conceptuelles et d'exemples.

Toutes les réponses et contenus suggérés par ces outils ont été soumis à une **analyse critique, une vérification, une validation technique, une adaptation et une intégration par l'auteur**, qui assume la responsabilité du texte, des codes et des ressources pédagogiques présentés. Les outils d'IA **ne sont pas**

considérés comme auteurs ou coauteurs de l'œuvre, ni responsables des décisions intellectuelles, techniques ou pédagogiques qui fondent son contenu.

Il convient également de souligner que les **simulateurs** et ressources interactives présents dans le livre — disponibles dans les versions HTML et Jupyter Notebook (IPYNB) — ont été conçus et implémentés dans le cadre de l'approche pédagogique de l'œuvre, dans le but de permettre une expérimentation directe des concepts présentés et de favoriser l'autonomie d'apprentissage du lecteur.

Distinct de l'utilisation éditoriale décrite ci-dessus, le *pipeline* de génération multilingue (voir « Comment ce livre est produit ») emploie un modèle de langage comme **composant d'ingénierie du système** : la traduction automatique du contenu entre le portugais et d'autres langues, avec un cache incrémental qui évite de retraduire les passages inchangés. Cette étape utilise l'API payante de **DeepSeek** (modèle `deepseek-v4-flash`) ; le livre complet a déjà été traduit du portugais vers l'**anglais**, le **français**, l'**espagnol** et l'**italien**, et les **chapitres 1 à 5** disposent en outre d'une piste **C++** qui compile et s'exécute réellement — le tout pour un coût cumulé de l'ordre de quelques dollars, grâce au cache incrémental.

Certaines images du livre contiennent encore du texte en portugais ; elles seront également traduites à l'avenir dans les autres langues, en suivant le même modèle de suffixe que les autres fichiers générés (par exemple, `image_en.png`).

La bibliothèque `morph.hpp`

La bibliothèque **`morph.hpp`** (Zampirolli et al., 2025) accompagne toute l'œuvre comme un outil de soutien à l'enseignement. Son objectif n'est pas de remplacer les bibliothèques consolidées, comme **OpenCV**, ni de rivaliser avec elles en performance ou en couverture. Au lieu de cela, elle cherche à **rendre transparents les algorithmes fondamentaux du Traitement Numérique d'Images et de la Vision par Ordinateur (TNI-VO)**, permettant à l'étudiant de comprendre leur implémentation, de modifier le code et de développer de nouvelles fonctionnalités.

C'est l'équivalent en C++ de `morph.py` : *header-only* (il suffit d'un `#include "morph.hpp"`), avec les **mêmes noms de fonction** dans le *namespace* `mm::` — quiconque connaît déjà `mm.dil()`, `mm.dil0()` ou `mm.gradm()` en Python reconnaît immédiatement `mm::dil()`, `mm::dil0()` et `mm::gradm()` en C++. Une divergence de nom entre les deux versions est considérée comme un défaut de la bibliothèque, non comme un choix de conception.

> **i** Héritage Technique
>
> La structure de `morph.hpp` (comme celle de sa contrepartie `morph.py`) a pour base des outils de Morphologie Mathématique développés au Brésil, comme **MMachLib** (Lotufo et al., 1997) et **MMach** (Barrera et al., 1998), ainsi que la **mmorph**, utilisée dans l'ouvrage de Dougherty; Lotufo (2003). Ces bibliothèques ont servi de référence pour l'enseignement du domaine pendant des décennies.

Cette philosophie peut être observée, par exemple, dans les opérateurs morphologiques de dilatation :

- `mm::dil0` : implémentation didactique de la dilatation pour les **éléments structurants planaires**, suivant directement la définition classique de la Morphologie Mathématique ;
- `mm::dil1` : implémentation didactique de la dilatation pour les **fonctions structurantes (*kernels* non planaires)**, suivant rigoureusement sa formulation mathématique ;
- `mm::dil` : par défaut, délègue à `mm::dil0()` — la version didactique planaire, numériquement équivalente à `cv::dilate` pour ce type d'élément structurant. L'implémentation optimisée, basée sur **OpenCV** (`cv::dilate`), n'entre en jeu que si le programme est compilé avec l'option `-DMM_USE_OPENCV`.

! Une différence délibérée par rapport à `morph.py`

En Python, `mm.dil()` (sans suffixe) **est déjà** l'implémentation optimisée par défaut. En C++, `mm::dil()` (même nom, même signature) ne devient la version optimisée que si OpenCV est explicitement lié à la compilation ; sans cela — ce qui est justement le cas par défaut, y compris sur Moodle/VPL — `mm::dil()` se comporte comme `mm::dil0()`. Ce choix évite que la piste C++ dépende d'OpenCV juste pour compiler et exécuter un exercice simple : `g++ fichier.cpp -o fichier` suffit déjà. Pour la version accélérée en local, il suffit de compiler avec `g++ -DMM_USE_OPENCV fichier.cpp -o fichier $(pkg-config --cflags --libs opencv4)`.

La bibliothèque offre également des fonctions pour la lecture, l'affichage, la conversion des couleurs, le filtrage et la morphologie mathématique via une interface simple :

```cpp
#include "morph.hpp"

int main() {
    mm::Image img = mm::gray(mm::read("lena.jpg"));  // lecture et conversion en niveaux de
gris
    mm::Image grad = mm::gradm(img);                 // gradient morphologique
    mm::show(grad, "sortie.png");                    // affichage (écrit dans un fichier)
}
```

Contrairement à la version Python, `mm::show()` exige un chemin de sortie explicite : chaque cellule de code C++ du livre s'exécute comme un processus isolé (compilé et exécuté via `%%writefile` + `!g++ ...` sur Colab), sans le compteur global de figures qui n'a de sens que dans un seul processus Jupyter.

De plus, tous les projets du **Gemini Notebook** de la piste C++ incluent le fichier `morph.hpp`, permettant de consulter et de discuter directement l'implémentation des algorithmes présentés dans le livre. De cette manière, l'étudiant peut utiliser le Gemini Notebook lui-même comme assistant d'étude, en posant des questions telles que :

> Expliquez comment la fonction `mm::dil0` de `morph.hpp` a été implémentée, en montrant le code et en commentant chaque étape de manière didactique. Expliquez également la différence entre `mm::dil0()` et `mm::dil()` sans l'option `-DMM_USE_OPENCV`.

! Implémentations didactiques et implémentations optimisées

Dans plusieurs chapitres, un même algorithme est présenté en deux versions. Les fonctions avec suffixe `0`, `1`, etc. (par exemple, `mm::dil0()` et `mm::dil1()`) sont des implémentations didactiques, développées pour faciliter la compréhension des algorithmes et disponibles indépendamment de l'option de compilation. Quant aux fonctions sans suffixe (comme `mm::dil()`), elles n'utilisent l'implémentation optimisée basée sur OpenCV que lorsqu'elles sont compilées avec `-DMM_USE_OPENCV` ; sinon, elles se comportent comme la version didactique correspondante.
Dans les activités évaluées par le **VPL de Moodle**, la compilation suit le mode par défaut sans `-DMM_USE_OPENCV` — non pas en raison de limitations de mémoire (comme c'est le cas de scikit-learn/scikit-image dans la piste Python), mais pour qu'aucun EP en C++ ne dépende de la présence d'OpenCV dans l'environnement d'exécution. En pratique, cela signifie que, sur le VPL, `mm::dil()` et `mm::dil0()` produisent exactement le même résultat. En local — par exemple dans **VS Code** ou **Google Colab** — l'étudiant peut choisir de compiler avec `-DMM_USE_OPENCV` pour comparer avec la version optimisée.

Le code source de la bibliothèque est disponible à :

https://github.com/fzampirolli/pdi-vc/tree/master/morph/cpp

Exercices de Programmation (EPs) et Validation Automatique

Chaque unité du livre inclut des **Exercices de Programmation (EPs)** pratiques et de complexité croissante, conçus pour consolider les concepts présentés tout au long du chapitre. Chaque EP est accompagné d'un **simulateur interactif**, disponible dans les versions HTML et IPYNB, qui permet à l'étudiant de manipuler les paramètres, de visualiser le comportement des algorithmes et de développer une compréhension intuitive du problème avant de commencer son implémentation. De cette manière, le processus d'apprentissage combine expérimentation, programmation et validation automatique.

La validation des solutions est effectuée localement par la classe `TestSuite` (`testsuite.py`), qui compare la sortie du programme avec les fichiers de cas de test (`.cases`) :

```
TestSuite("EP01_01.py").run()
```

Le système prend en charge plusieurs langages (Python, Java, C++, C, JavaScript et R) et peut être intégré directement à Moodle. Pour les enseignants intéressés par le processus complet pas à pas de publication des EPs comme activités VPL, consultez le **Guide de l'Enseignant**, à la fin de cette préface, et l'Annexe B.

Code ouvert

Le projet est régi par des principes de code ouvert. Le dépôt public rassemble le texte, les codes, les images et les *scripts* : **github.com/fzampirolli/pdi-vc**

Chaque version du livre est archivée en permanence sur **Zenodo** avec un *DOI citable*. Pour citer ce matériel dans des travaux académiques :

> ZAMPIROLLI, Francisco de Assis. **PDI+VC — Traitement Numérique d'Images et Vision par Ordinateur**. UFABC, 2026. DOI : 10.5281/zenodo.20784605

Il convient de noter, à propos, que le contenu exposé dans cet ouvrage reflète le regard critique, la proposition pédagogique et l'expérience d'enseignement de son auteur. Ainsi, les analyses, approches et opinions exprimées tout au long de ce livre représentent uniquement la compréhension de son concepteur, ne constituant ni ne reflétant une position officielle ou institutionnelle de l'Université Fédérale de l'ABC (UFABC).

Avant de commencer : *Notebooks* en Python

Le concept de ***Literate Programming*** (Programmation Littéraire), proposé par Donald Knuth (Knuth, 1984), fonde la structure de ce matériel. La logique inverse le paradigme traditionnel : le programme est écrit pour la lecture humaine, ressemblant à un essai, tandis que le code est extrait séparément pour l'exécution computationnelle.

Le contenu est structuré en *notebooks* — des documents qui entrelacent des **cellules de texte** (en Markdown) et des **cellules de code** (en Python).

- **Exécution :** les cellules de code sont identifiées par `[ ]`. L'exécution peut être réalisée avec `Shift + Enter` ou par le bouton ▷ de l'interface.
- **Environnements :** les *notebooks* peuvent être exécutés localement, via Jupyter ou Visual Studio Code (VS Code), ainsi que dans des environnements cloud, comme Google Colab.

> Note sur le format
>
> Dans les environnements interactifs, le code peut être modifié et exécuté. Dans les versions statiques (HTML ou PDF), les blocs de code ont une finalité de lecture et de référence, sans préjudice à l'intégrité des explications.

Guide de l'Enseignant : Publication des EPs sur Moodle

*Cette section est destinée aux enseignants souhaitant intégrer les Exercices de Programmation (EPs) aux activités **VPL (Virtual Programming Lab)** de Moodle, en complément de l'Annexe B.*

Intégration avec Moodle (VPL)

Les EPs des *notebooks* peuvent être convertis en activités **VPL** sur Moodle. Le *script* `ep_tools.py` (exécuté via `make build`) automatise l'exportation des EPs de la fin de chaque chapitre en deux étapes :

1. **Extraction** (`make eps`) : génère un HTML interactif indépendant par EP, avec énoncé, exemples et simulateur, dans `gen/book/eps/<versao>/`. Voir la liste complète à : https://fzampirolli.github.io/pdi-vc/eps/py.pt/index.html
2. **Conversion** (`make moodle`) : transforme le HTML en fragment autonome, sans dépendance à un CSS externe, prêt à être collé dans l'éditeur de Moodle, dans `gen/book/eps/<versao>_moodle/`. Voir un exemple à : https://fzampirolli.github.io/pdi-vc/eps/py.pt_moodle/EP01_01.html

De plus, tous les simulateurs interactifs développés tout au long du livre peuvent être consultés directement à https://fzampirolli.github.io/pdi-vc/simuladores/py.pt/.

> **Autres pistes et langues**
>
> Les liens ci-dessus utilisent `py.pt` comme exemple. Pour accéder aux EPs ou simulateurs d'une autre combinaison langage/langue, il suffit de remplacer ce segment dans l'URL — par exemple, `cpp.it` pour la piste C++ en italien : `https://fzampirolli.github.io/pdi-vc/eps/cpp.it/index.html`. La structure est la même pour toutes les combinaisons disponibles, sous `/eps/<version>/`, `/eps/<version>_moodle/` et `/simuladores/<version>/`.

Lors de la publication avec `make publish`, ces **deux versions** de chaque EP sont disponibles aux *liens* indiqués. L'enseignant peut combiner les deux stratégies : coller la version Moodle dans l'éditeur VPL (avec simulateur fonctionnel) et inclure dans l'énoncé un *lien* vers la version complète, où les formules sont rendues correctement par MathJax.

> **Révision obligatoire avant de publier sur Moodle**
>
> Bien qu'automatisée, la conversion exige une révision de l'enseignant en raison des limitations de l'éditeur TinyMCE/HTML Purifier de Moodle :
>
> - **Formules mathématiques** — TinyMCE rejette les barres obliques inverses (\), corrompant les équations LaTeX. Pour cette raison, la version Moodle convertit les formules simples en HTML pur (entités comme `&ge;`, `&times;`). Les formules complexes (`\begin{cases}`, matrices, intégrales) peuvent sortir incomplètes — vérifiez et, si nécessaire, réécrivez-les en texte ou indiquez la version complète via *lien*.
> - **Figures externes** — Les images complémentaires doivent être insérées manuellement dans l'activité VPL.
> - **Simulateurs** — Fonctionnent parfaitement à condition d'utiliser du JavaScript standard avec des styles *inline* et une manipulation directe du DOM (`getElementById`). **Attention :** si vous incluez des formules en LaTeX contenant le caractère \ sur Moodle, les simulateurs peuvent cesser de fonctionner.

Comment publier sur Moodle

1. Accédez à la version Moodle de l'EP souhaité :

   ```
   https://fzampirolli.github.io/pdi-vc/eps/py.pt_moodle/EPXX_YY.html
   ```

2. Copiez le code source complet de la page (`Ctrl+U` → `Ctrl+A` → `Ctrl+C`).

3. Sur Moodle, créez l'activité VPL, accédez à l'éditeur HTML, collez le contenu (`Ctrl+V`) et enregistrez.
4. Importez les fichiers `.cases` du dossier `all/capXX/casos/` dans les paramètres de test du VPL.

Réutilisation des Cas de Test

Les fichiers `.cases` sont compatibles avec le VPL, permettant d'utiliser le même ensemble de tests tant dans le développement local que dans la correction automatisée sur Moodle.

Partie I

Partie I — Fondements du TNI

Chapitre 1

Fondements et Premiers Pas

Ce chapitre inaugure la **Partie 1** de l'ouvrage, consacrée aux fondements du Traitement Numérique des Images (TNI). Y sont présentés la représentation mathématique des images numériques et les principales méthodes pour leur manipulation.

On utilise le langage C++ et la bibliothèque `morph.hpp`, une version didactique de `morph.py` (ZAMPIROLLI, 2025).

1.1 Objectifs

À la fin de ce chapitre, vous serez capable de :

- Comprendre la nature physique et mathématique de l'image numérique $f(x, y)$.
- Identifier les bandes du spectre électromagnétique pertinentes pour le TNI.
- Effectuer les opérations de base : lecture, affichage et sauvegarde d'images.
- Accéder aux intensités des pixels et les modifier individuellement.
- Appliquer un seuillage manuel.
- Configurer l'environnement de développement en C++.
- Manipuler les structures de matrices (`std::vector`) sans tomber dans les pièges de copie/référence.

1.2 Avant de commencer : *Notebooks* interactifs

Ce matériel a été conçu selon le concept de ***Literate Programming*** (Programmation lettrée), imaginé par Donald Knuth dans les années 1980 (KNUTH, 1984). Knuth — également créateur du système **TeX** pour la typographie numérique — a proposé que les programmes soient écrits comme un récit logique destiné aux êtres humains, entrelaçant code et documentation.

Pour exécuter une cellule, appuyez sur Shift + Enter ou cliquez sur le bouton ▷.

> **i** Note sur le format
>
> Dans les versions rendues (PDF ou HTML), le code est présenté dans des blocs statiques à des fins de lecture et de référence. L'exécution interactive nécessite un accès via Google Colab (disponible en haut de la page) ou dans un environnement local via VSCode ou Jupyter Notebook.

1.3 Fondements

L'étude des systèmes basés sur l'image englobe un écosystème de disciplines intégrées qui transforment des données visuelles brutes en connaissances structurées. Tandis que certains domaines se concentrent sur la

génération de représentations, d'autres se consacrent au traitement et à l'analyse de ces données pour soutenir des applications technologiques complexes.

Le diagramme présenté dans le Figure 1.1 établit la distinction et la complémentarité entre le Traitement Numérique des Images (TNI) et la Vision par Ordinateur (VO). Le TNI, mis en évidence en **vert**, se concentre sur la transformation d'image en image, visant l'amélioration de la qualité ou le prétraitement, comme la suppression du bruit et le rehaussement du contraste.

En revanche, la VO, signalée en **bleu**, se focalise sur l'interprétation du contenu visuel pour extraire des modèles ou des informations, tels que la reconnaissance d'objets et de gestes. La région d'intersection illustre la synergie entre les domaines, où le TNI prépare les données visuelles pour l'interprétation par la VO. La carte démontre également les interconnexions des deux disciplines avec des domaines tels que la Robotique, l'Infographie, l'Intelligence Artificielle (IA) et les Neurosciences.

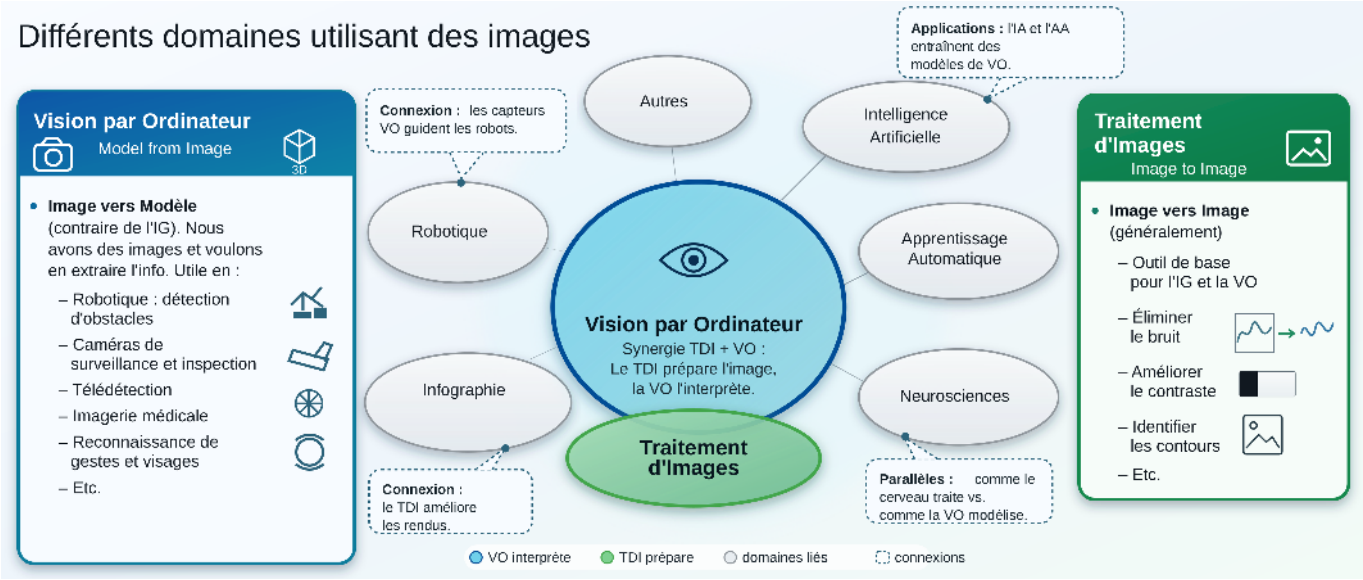


Figure 1.1: Diagramme relationnel détaillant les distinctions fondamentales, les synergies et les interconnexions entre le TNI et la VO dans le contexte des systèmes basés sur l'image.

1.3.1 👁️ Vision par ordinateur

- **Focus :** *Image* → *Modèle* (chemin inverse de l'infographie).
- **Objectif :** Extraire des informations de haut niveau à partir d'images ou de vidéos.
- **Applications typiques :**
  - Robotique – détection d'obstacles, localisation et navigation autonome.
  - Surveillance et inspection – reconnaissance d'événements, lecture de plaques, contrôle qualité.
  - Télédétection – analyse d'images satellite, cartographie environnementale.
  - Imagerie médicale – détection de tumeurs, segmentation d'organes, aide au diagnostic.
  - Interaction homme-machine – reconnaissance de gestes, expressions faciales, suivi oculaire.
- **Relation avec d'autres domaines :** utilise des techniques d'apprentissage automatique et d'IA pour classer et interpréter les scènes ; sert d'« yeux » à la robotique.

1.3.2 🖼️ Traitement Numérique des Images (TNI)

- **Focus :** *Image* → *Image* (généralement — transformation d'une image en une autre).
- **Objectif :** Améliorer la qualité visuelle ou extraire des caractéristiques de bas niveau.

- **Applications courantes :**
  - Élimination du bruit (filtres moyenne, médiane, gaussien).
  - Amélioration du contraste (égalisation d'histogramme, ajustement gamma).
  - Détection des contours (Sobel, Canny, Laplacien).
  - Segmentation (seuillage, croissance de régions, *watershed*).
  - Transformations géométriques (redimensionnement, rotation, correction de perspective).
- **Relation avec d'autres domaines (voir Table 1.1) :**
  - C'est la **base** de la plupart des systèmes de VC (prétraitement).
  - L'infographie applique souvent le TNI pour le post-traitement (ex. : lissage, rehaussement).
  - Les techniques d'IA peuvent optimiser les paramètres de traitement (ex. : apprentissage de filtres).

Tableau 1.1: Connexion entre le PDI, la VC et d'autres domaines scientifiques.

Domaine	Relation avec le PDI et la VC
Intelligence artificielle	Fournit des modèles (réseaux de neurones, SVM) qui interprètent les sorties de la VC.
Robotique	Consomme des données de la VC pour prendre des décisions (navigation, manipulation).
Apprentissage automatique	Utilise des descripteurs extraits par le PDI/VC pour entraîner des classifieurs.
Infographie	Chemin inverse : *modèle* $\rightarrow$ *image* ; applique souvent le PDI pour un rendu réaliste.
Neuroscience	Inspire des modèles de PDI (ex. : filtres similaires aux cellules ganglionnaires de la rétine).

1.4 Étapes du PDI

Les étapes du PDI sont présentées dans la Figure 1.2 et peuvent être comprises comme une chaîne de transformations qui réduit la redondance des données afin d'en extraire du sens :

- **Niveau bas :** agit directement sur les pixels de l'image bruitée pour effectuer des améliorations et des filtrages, en produisant en sortie une image propre ou rehaussée.
- **Niveau moyen :** reçoit l'image traitée, effectue la segmentation et la description, transformant la matrice de pixels en attributs structurés (forme, taille et texture).
- **Niveau haut :** utilise la table d'attributs pour alimenter des processus logiques et d'intelligence artificielle, aboutissant à la décision ou à la reconnaissance finale (comme le diagnostic médical).

A Figure 1.3 détaille la séquence complète du PDI, depuis la capture jusqu'à l'interprétation. Le flux commence à l'acquisition de l'image (1) et se poursuit avec l'amélioration (2) et la restauration (3). Ensuite, le contenu est isolé par la segmentation (4) et affiné par la morphologie (5). La transition cruciale se produit à la représentation et à la description (6), où les objets visuels sont convertis en données mathématiques (aire, périmètre, etc.), permettant la reconnaissance (7). Les processus auxiliaires incluent le traitement d'image couleur et la compression, qui contribuent à l'efficacité du stockage et de l'analyse.

1.5 Formation de l'image et le spectre

Le processus de formation d'une image repose sur l'interaction entre la matière et l'énergie rayonnante. Essentiellement, une image est conçue lorsqu'un **capteur enregistre le rayonnement** issu de l'interaction avec un objet physique. Dans le contexte de la vision humaine et de la photographie conventionnelle, ce phénomène dépend d'une source de lumière qui éclaire la scène ; les caractéristiques des objets sont alors codées par le biais des **variations d'intensité et de couleur** de la lumière qui atteint le capteur, comme illustré dans la Figure 1.4.

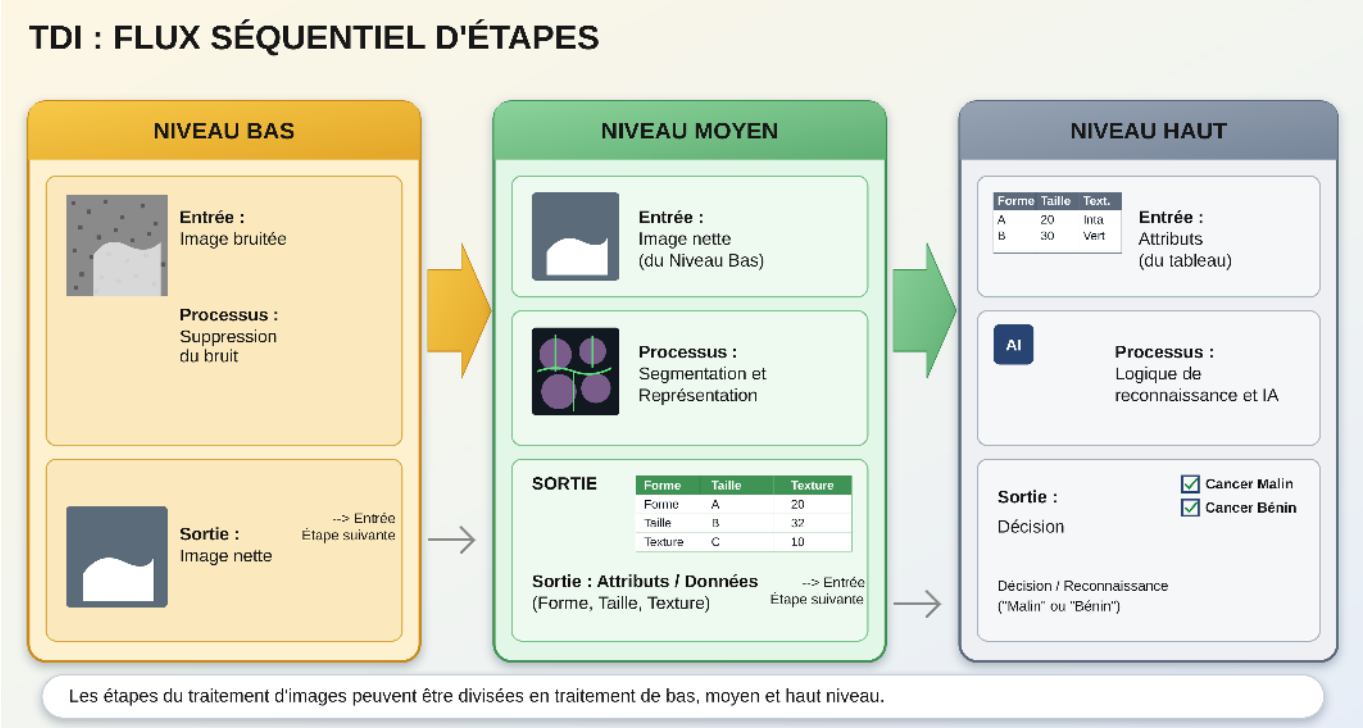


Figure 1.2: Représentation du flux séquentiel de traitement : la sortie de chaque niveau devient l'entrée du niveau suivant.

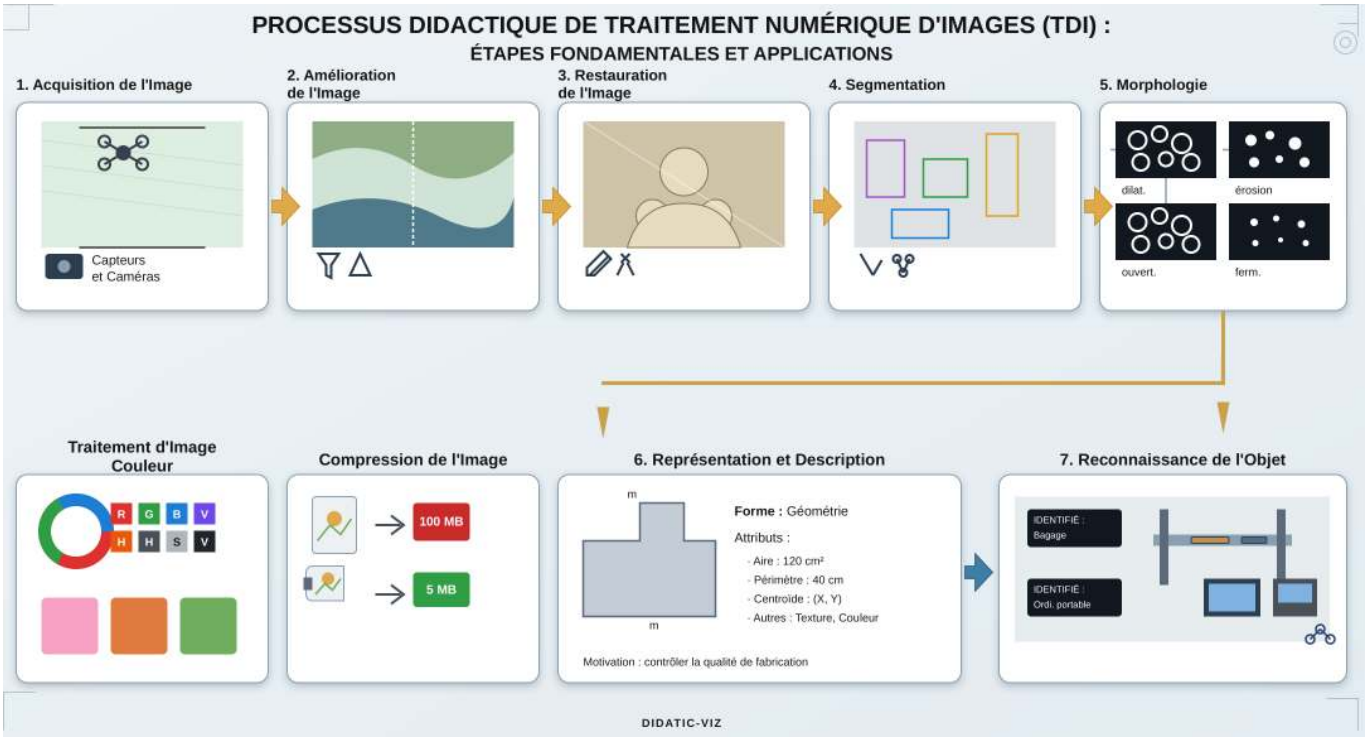


Figure 1.3: Flux détaillé du PDI : de l'acquisition sensorielle à l'extraction d'attributs et à la reconnaissance automatisée, incluant le traitement couleur et la compression.

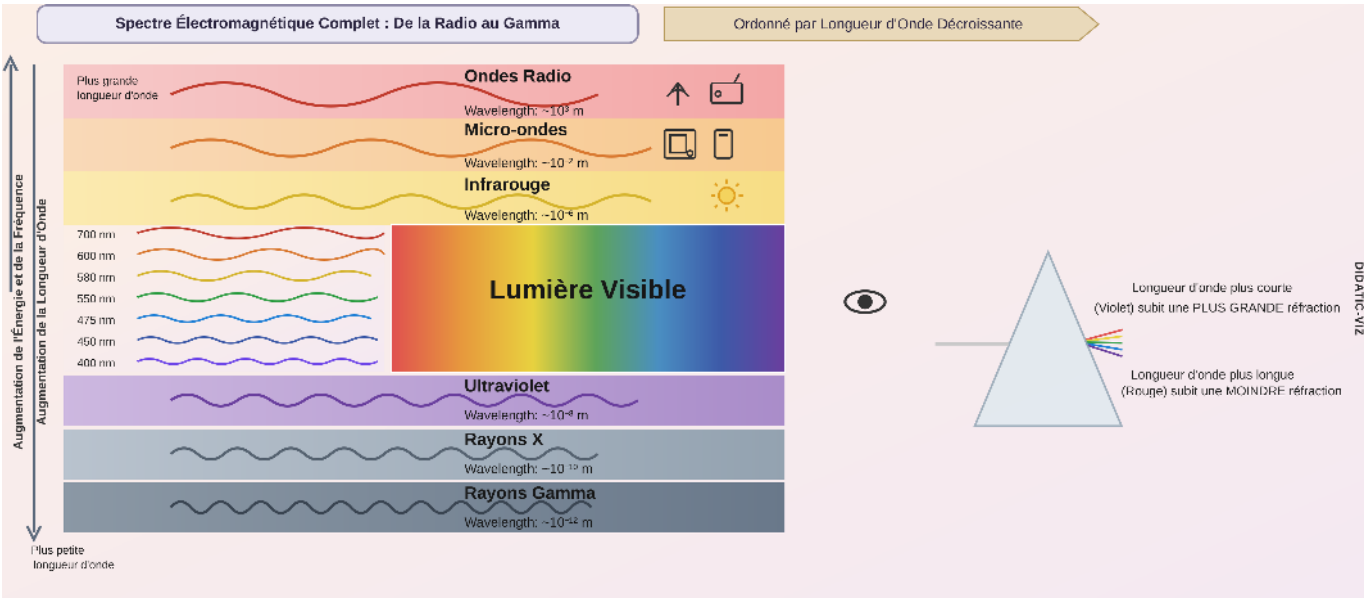


Figure 1.4: Représentation du spectre visible et de sa position par rapport aux autres rayonnements électromagnétiques, mettant en évidence la variation des longueurs d'onde de 380 nm à 750 nm.

La **lumière visible** n'occupe qu'une petite bande du spectre électromagnétique — entre **380 nm (violet)** et **750 nm (rouge)** — comme illustré dans la Figure 1.5. Les capteurs numériques classiques opèrent dans cette même fenêtre, mais des équipements spécialisés peuvent capter des rayonnements invisibles à l'œil humain, tels que l'infrarouge et les rayons X. En PDI, l'image formée dépend directement de la sensibilité spectrale du capteur utilisé.

À partir de ce processus physique d'acquisition, il devient possible de modéliser mathématiquement l'image numérique comme une fonction bidimensionnelle discrète, dans laquelle chaque point de la scène est représenté par des échantillons numériques d'intensité lumineuse, formalisant ainsi les concepts de pixel et d'image numérique présentés dans la section suivante.

1.6 Qu'est-ce qu'une image numérique ?

Une **image numérique** est constituée d'une grille de **pixels** (*Picture Elements*), où chaque pixel est la plus petite unité élémentaire de l'image.

> Qu'est-ce qu'un pixel ?
>
> Un **pixel** est la plus petite unité adressable qui compose une image numérique. Chaque pixel occupe une position unique dans la grille et stocke une ou plusieurs valeurs numériques qui représentent son intensité ou sa couleur.

Représentation mathématique

Contrairement à une fonction continue, le domaine d'une image numérique est un plan rectangulaire fini $\mathbb{E} \subset \mathbb{Z}^2$, qui représente la grille d'échantillonnage. Ce domaine est indexé par des coordonnées entières :

$$\mathbb{E} = \{(x, y) \in \mathbb{Z}^2 \mid 0 \leq x < L,\ 0 \leq y < H\} \tag{1.1}$$

Où :

- L : représente la largeur de l'image (nombre de colonnes).
- H : représente la hauteur de l'image (nombre de lignes).

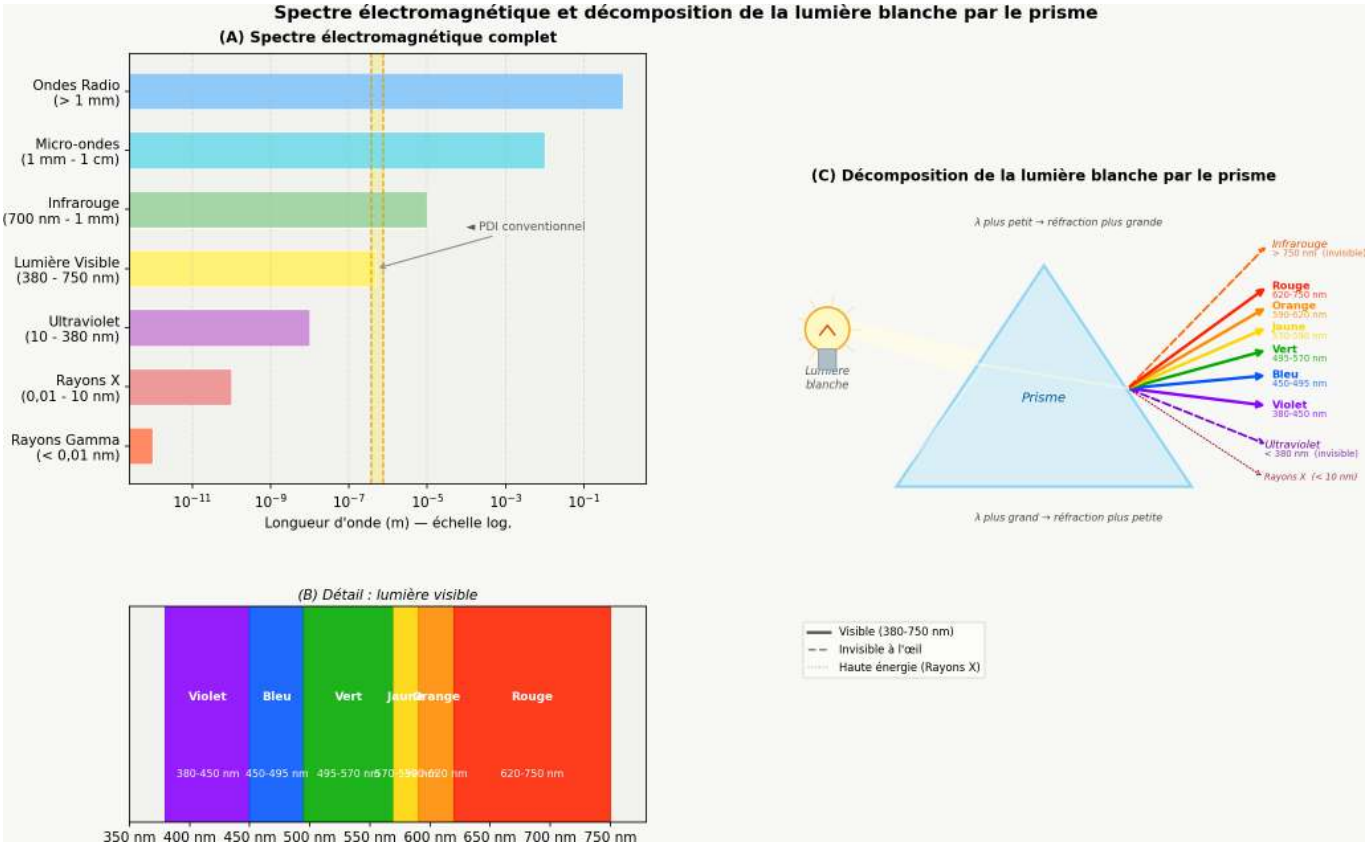


Figure 1.5: (A) Spectre électromagnétique complet en échelle logarithmique, avec mise en évidence de la bande visible. (B) Détail de la lumière visible (380-750 nm) et de ses couleurs. (C) Décomposition de la lumière blanche par le prisme : la longueur d'onde la plus courte subit une réfraction plus grande, séparant UV, visible et infrarouge.

L'image numérique est une fonction qui associe à chaque paire de coordonnées (x, y) une ou plusieurs valeurs décrivant l'apparence du pixel.

$$f : \mathbb{E} \to \mathcal{V} \tag{1.2}$$

L'ensemble $\mathcal{V}$ définit les valeurs possibles pour le pixel (codomaine), variant selon le type d'image, comme illustré dans la Table 1.2.

Tableau 1.2: Principaux types d'image numérique et leurs ensembles respectifs de valeurs possibles pour chaque pixel.

Type d'image	$\mathcal{V}$ (valeurs du pixel)	Représentation
Binaire	$\{0, 1\}$ ou $\{0, 255\}$	■□
Niveaux de gris	$\{0, 1, \ldots, 255\}$	`[ ][=][#]`■
Couleur (RVB)	$\{0, \ldots, 255\}^3$ (triplets ordonnés de valeurs)	■■■

Exemple pratique : Une image couleur dans le modèle RVB peut être représentée mathématiquement par une fonction qui associe trois valeurs d'intensité à chaque pixel. Informatiquement, cette représentation correspond à trois matrices superposées — les canaux rouge (*Red*), vert (*Green*) et bleu (*Blue*) — dans lesquelles chaque élément stocke l'intensité lumineuse du canal respectif à une position donnée de l'image.

Pour permettre les expériences pratiques de TNI et VC, ce livre utilise C++17 et un ensemble minimal de bibliothèques dédiées à la manipulation matricielle, au traitement d'images et à la visualisation des résultats, présentés ci-dessous.

1.7 Configuration de l'environnement

Ce matériel utilise **C++17** et la bibliothèque à en-tête unique **morph.hpp**, initialement proposée pour Python dans Zampirolli (2025). Ici, une version minimale et didactique de `morph.py` est utilisée, adaptée pour une compilation simple avec `g++`, comme présenté dans Table 1.3.

Chaque cellule de code est compilée et exécutée comme un processus isolé (`%%writefile fichier.cpp` suivi de `g++`). Contrairement au *kernel* Python, il n'y a pas d'état persistant entre les cellules : les images produites par une cellule sont enregistrées dans des fichiers pour être lues par la suivante.

Les **Exercices de Programmation (EP)** présentés à la fin des chapitres peuvent être validés par le même module `testsuite.py` utilisé dans le parcours Python, qui compile la solution avec `g++` et compare sa sortie avec les cas de test. Ainsi, les mêmes cas de test (`.cases`) peuvent valider des solutions en C++, Python et d'autres langages, à la fois dans les *notebooks* et dans **Moodle/VPL**.

Tableau 1.3: Principales bibliothèques et outils utilisés dans le parcours C++ de ce livre.

Bibliothèque / Outil	Fonction principale
`g++` (C++17)	Compilation de chaque cellule de code
`morph.hpp`	Abstraction didactique des opérations de TIA
`stb_image.h` / `stb_image_write.h`	Lecture/écriture de PNG/JPEG (sans OpenCV)
`testsuite.py`	Exécution et validation automatique des EP

1.8 À propos de `morph.hpp`

La `morph.hpp` est une version C++ minimale et didactique de `morph.py` : elle implémente les fonctions utilisées dans ce chapitre (`read`, `gray`, `randomImage`, `show`, `write`, `threshold`, `drawImg`), ainsi que des

opérations de morphologie (`dil`/`ero` et les variantes `dil0`/`ero0`/`dil1`/`ero1`) préparées pour les chapitres suivants. Il n'y a pas de parité numérique garantie avec l'implémentation Python — le critère est « cela compile et produit une image plausible », et non « un résultat bit à bit identique à Python ».

Les bibliothèques `stb_image.h`/`stb_image_write.h` (domaine public/MIT) sont **vendorisées** avec le dépôt — c'est-à-dire que leur code source est déjà copié dans le projet lui-même, plutôt que d'être installé séparément via `apt install` — ce qui évite de dépendre des paquets du système lors de la compilation dans Colab. Les opérations `dil()`/`ero()` utilisent `cv::dilate`/`cv::erode` lorsqu'elles sont compilées avec `-DMM_USE_OPENCV` ; sans ce *flag* (par défaut, y compris dans Moodle/VPL), elles retombent sur la version didactique équivalente (`dil1`/`ero1`) sans exiger OpenCV.

```
import os, urllib.request

os.makedirs("tmp/state", exist_ok=True)  # artefacts de build du parcours C++ (.cpp, binaire,
PNGs)

url = "https://raw.githubusercontent.com/fzampirolli/pdi-vc/master/morph/config.py"
if not os.path.exists("config.py"):
    urllib.request.urlretrieve(url, "config.py")

# Le noyau est Python même dans le parcours C++ : `mm` (morph.py) est utilisé par les
# simulateurs, par l'affichage des figures que le binaire C++ génère et par
# l'état mm::Image entre cellules. cpp=True télécharge aussi le parcours compilé
# (morph.hpp + stb_image*.h), utilisé dans le #include des cellules %%writefile *.cpp.
import config
config.setup(testsuite=True, cpp=True)
from morph import mm
from testsuite import TestSuite
```

```
Environnement prêt. Morph : 1.1.9 | OpenCV : 5.0.0 | TestSuite : 1.1.2
```

1.9 Fondements des matrices — attention à la copie de références

Comme une image numérique peut être représentée par une matrice, il est important de comprendre comment créer et manipuler correctement les matrices. En C++, `std::vector` a une **sémantique de valeur** : copier le vecteur copie ses données. Ainsi, `std::vector<std::vector<int>> m(3, std::vector<int>(2, 0))` crée effectivement trois lignes **indépendantes** — le constructeur de remplissage copie la ligne modèle trois fois.

> ⚠ Attention à la copie de références
>
> Le piège apparaît lorsque l'on utilise des **pointeurs** ou des **références** au lieu de copies. Dans `std::vector<int> ligne(2, 0); std::vector<std::vector<int>*> m(3, &ligne);`, les trois éléments de `m` pointent vers la **même** ligne, et modifier `(*m[0])[0]` modifie toutes les lignes. Il en va de même avec `auto& ligne = m[0];` (référence — modifie la matrice), contrairement à `auto ligne = m[0];` (copie — laisse la matrice intacte).

Pour visualiser ce comportement, on peut exécuter le code dans Python Tutor (qui exécute également C++ pas à pas) et comparer l'effet des copies et des références.

En pratique, pour le traitement numérique des images (TNI), on utilise la structure `mm::Image` de `morph.hpp`, qui possède également une sémantique de valeur : `mm::Image b = a;` copie les pixels, tandis que `mm::Image& b = a;` crée seulement un alias pour la même image. Le code suivant présente différentes manières de créer des images synthétiques, dont les résultats sont affichés dans la Figure 1.6.

```
%%writefile tmp/fig_imagens_sinteticas.cpp
#define MM_OUT "tmp/fig_imagens_sinteticas.png"
// Compile with: g++ -std=c++17 -o program program.cpp -I/path/to/morph/include
-L/path/to/morph/lib -lmorph

#include "morph.hpp"
#include <iostream>
#include <vector>
#include <filesystem>

int main() {
    //#| quarto-raw: false
    //#| label: fig-imagens-sinteticas
    //#| fig-cap: "Exemples d'images synthétiques représentées matriciellement."
    //#| echo: true
    //#| output: true

    // Création d'une image noire (zéros) de 4, 6 pixels
    mm::Image img_preta(4, 6);
    img_preta.at(0,0) = 255;  // pixel blanc dans le coin supérieur gauche

    // Création d'une image blanche (255) de 4, 6 pixels
    mm::Image img_branca(4, 6);
    std::fill(img_branca.data.begin(), img_branca.data.end(), 255);
    img_branca.at(3,5) = 0;  // pixel noir dans le coin inférieur droit

    // Création d'une image aléatoire pour les tests (bruit)
    mm::Image img_random = mm::randomImage(4, 6, 255);

    std::cout << "Matrice aléatoire générée:" << std::endl;
    std::cout << mm::drawImg(img_random) << std::endl;

    mm::show(
        std::vector<mm::Image>{img_preta, img_branca, img_random},
        MM_OUT,
        std::vector<std::string>{
            "Principalement noire\n(avec 1 pixel blanc en (0,0))",
            "Principalement blanche\n(avec 1 pixel noir en (3,5))",
            "Image aléatoire\n(simulation de bruit)"
        },
        3
    );


// [pdi:panel-io] auto-generated - do not edit by hand
std::filesystem::create_directories("tmp");
mm::write(img_preta, "tmp/fig_imagens_sinteticas_0.png");
mm::write(img_branca, "tmp/fig_imagens_sinteticas_1.png");
mm::write(img_random, "tmp/fig_imagens_sinteticas_2.png");
// [pdi:panel-io:end]
return 0;
}
```

```
Overwriting tmp/fig_imagens_sinteticas.cpp
```

```
!g++ -I. -std=c++17 tmp/fig_imagens_sinteticas.cpp -o tmp/fig_imagens_sinteticas \
  && ./tmp/fig_imagens_sinteticas \
  && test -f "tmp/fig_imagens_sinteticas.png" \
  || echo "  mm::show não gravou tmp/fig_imagens_sinteticas.png"
```

```
Matrice aléatoire générée:
148 150  79 151 193 127
 97  41 209  68 212 153
  8  68  63   4  69  20
 37 231 190 247  83  42

[1] Principalement noire
(avec 1 pixel blanc en (0,0))
[2] Principalement blanche
(avec 1 pixel noir en (3,5))
[3] Image aléatoire
(simulation de bruit)
```

```
try:
    mm.show(
        [
            mm.read("tmp/fig_imagens_sinteticas_0.png"),
            mm.read("tmp/fig_imagens_sinteticas_1.png"),
            mm.read("tmp/fig_imagens_sinteticas_2.png"),
        ],
        titles=[
            'Predominantemente preta\n(com 1 pixel branco em (0,0))',
            'Predominantemente branca\n(com 1 pixel preto em (3,5))',
            'Imagem aleatória\n(simulação de ruído)',
        ],
        cols=3,
        axis=True,
        figsize=(9, 3),
    )
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + "
    tmp/fig_imagens_sinteticas_0.png (ver a versao Python)")
```

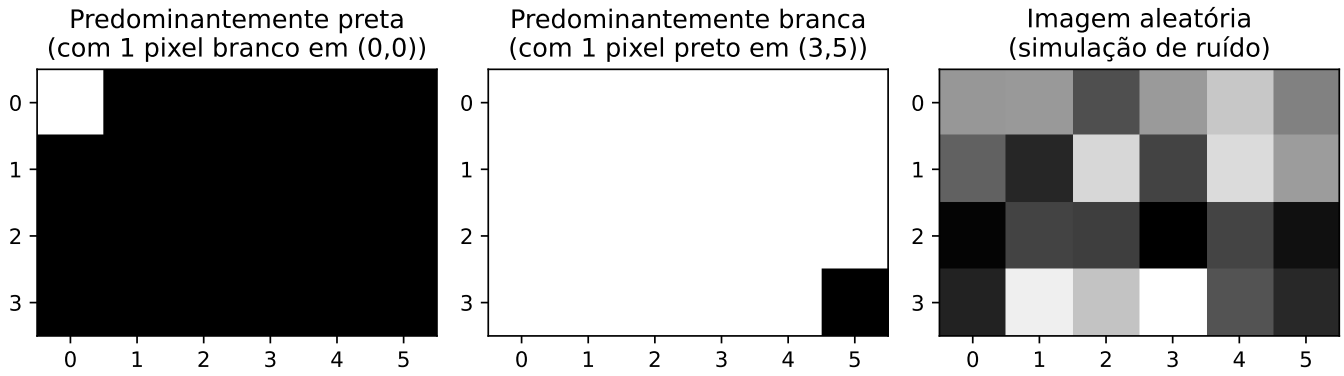


Figure 1.6: Exemplos de imagens sintéticas representadas matricialmente.

1.10 Lecture et Affichage d'Images

Dans des bibliothèques comme OpenCV (`cv::Mat`), les images numériques sont représentées informatiquement comme des matrices multidimensionnelles. Dans `morph.hpp`, la structure `mm::Image` adopte une approche plus simple : un tampon linéaire d'octets (`std::vector<unsigned char>`), avec une hauteur, une largeur et un nombre de canaux explicites.

L'une des opérations fondamentales en PDI est la lecture d'images.

Dans `morph.hpp`, la fonction `mm::read()` permet de charger des images tant depuis des fichiers locaux que depuis des URLs, comme illustré dans la Figure 2.7..

```
%%writefile tmp/fig_01_natureza.cpp
#define MM_OUT "tmp/fig_01_natureza.png"
// Compile: g++ -std=c++17 -o program program.cpp -I/path/to/morph.hpp

#include "morph.hpp"
#include <iostream>
#include <string>
#include <vector>
#include <filesystem>
#include <algorithm>

int main() {
    // #| label: fig-01-natureza
    // #| fig-cap: "Mandrill (Mandrillus sphinx) em ambiente natural. Crédito: Julien Renoult
    (CC BY 4.0)."
    // #| echo: true

    std::string url     =
"https://upload.wikimedia.org/wikipedia/commons/8/87/Mandrillus_sphinx_339428057.jpg";
    std::string caminho = "imagens/mandrill.png";

    mm::Image img_obj;

    if (!std::filesystem::exists(caminho)) {
        std::filesystem::create_directories("imagens");
        img_obj = mm::read(url);
        mm::write(img_obj, caminho);
    } else {
        img_obj = mm::read(caminho);
    }

    mm::Image img = img_obj;

    std::cout << "Dimensões (H, W, Canais): " << img.h << ", " << img.w << ", " <<
img.channels << "\n";
    std::cout << "Tipo de dado: unsigned char" << "\n";

    mm::show(img, MM_OUT, "Exemplo: Captura RGB");


// [pdi:state-io] auto-generated - do not edit by hand
std::filesystem::create_directories("tmp/state");
mm::write(img, "tmp/state/img_34.png");
// [pdi:state-io:end]
return 0;
}
```

```
Overwriting tmp/fig_01_natureza.cpp
```

```
!g++ -I. -std=c++17 tmp/fig_01_natureza.cpp -o tmp/fig_01_natureza \
  && ./tmp/fig_01_natureza \
  && test -f "tmp/fig_01_natureza.png" \
  || echo "  mm::show não gravou tmp/fig_01_natureza.png"
```

```
Dimensões (H, W, Canais): 1365, 2048, 3
Tipo de dado: unsigned char
Exemplo: Captura RGB
```

```
try:
    mm.show(mm.read("tmp/fig_01_natureza.png"))
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + " tmp/fig_01_natureza.png
    (ver a versao Python)")
```

Figure 1.7: Mandrill (Mandrillus sphinx) em ambiente natural. Crédito: Julien Renoult (CC BY 4.0).

Variante : *Téléchargement* de l'image vers le stockage local

Dans les environnements où la lecture directe d'URL n'est pas disponible — en raison de restrictions réseau, de politiques de *pare-feu* ou de l'absence de connectivité — une variante consiste à télécharger préalablement l'image vers le système de fichiers local, puis à la charger avec `mm::read()`. Dans le volet C++, `mm::read` accepte également les URL directement (téléchargement interne via `curl`/`wget`) ; cette variante couvre le cas où l'on télécharge une fois et réutilise le fichier local. Dans l'exemple suivant, le fichier est enregistré sous le nom `mandrill.png`.

```
!wget -O mandrill.png \
    https://upload.wikimedia.org/wikipedia/commons/8/87/Mandrillus_sphinx_339428057.jpg
```

```
%%writefile tmp/ler_local.cpp
#define MM_OUT "tmp/mandrill_local.png"
#include "morph.hpp"

int main() {
    mm::Image img = mm::read("mandrill.png");   // lit le fichier local
    mm::show(img, MM_OUT);                      // Exemple : capture RVB (fichier local)
    return 0;
}
```

```
!g++ -I. tmp/ler_local.cpp -o tmp/ler_local && ./tmp/ler_local
```

Explication

- `wget -O mandrill.png <URL>` effectue le téléchargement de l'image et la stocke localement sous le nom spécifié ;
- `mm::read("mandrill.png")` effectue la lecture du fichier directement à partir du système de fichiers, sans nécessiter de requêtes HTTP supplémentaires ;
- Cette stratégie réduit la dépendance à la connectivité pendant l'exécution des expériences et évite des téléchargements répétés de la même image.

Note

Le préfixe `!` est utilisé dans les environnements basés sur des *carnets de notes*, comme Jupyter Notebook, JupyterLab et Google Colab, pour exécuter des commandes du système d'exploitation directement dans des cellules de code. Dans les terminaux classiques, la commande doit être utilisée sans le préfixe `!`. Si l'utilitaire `wget` n'est pas installé, on peut utiliser alternativement :

```
!curl -o mandrill.png \
    https://upload.wikimedia.org/wikipedia/commons/8/87/Mandrillus_sphinx_339428057.jpg
```

1.11 Conversion de Types et Seuillage

Comme nous l'avons vu, une image couleur dans l'espace RVB est représentée par la fonction :

$$f : \mathbb{E} \to \{0, 1, \dots, 255\}^3$$

C'est-à-dire que pour chaque pixel (x, y), nous avons trois valeurs (R, V, B) qui définissent sa couleur.

Conversion en Niveaux de Gris

Pour convertir une image RVB en niveaux de gris (*grayscale*), il est nécessaire de combiner les trois canaux en une seule valeur d'intensité g, qui représente la luminosité perçue. Comme l'œil humain n'est pas également sensible au rouge, au vert et au bleu, on utilise une **moyenne pondérée**. La norme ITU-R BT.601 ({ITU-R}, 2011) définit les poids suivants :

$$g = 0.299\, R + 0.587\, G + 0.114\, B \tag{1.3}$$

Après le calcul, la valeur g est arrondie à l'entier le plus proche et ajustée à l'intervalle $[0, 255]$. Le résultat est une nouvelle image, désormais en niveaux de gris, représentée par :

$$f_{\text{gris}} : \mathbb{E} \to \{0, 1, \dots, 255\}$$

Seuillage (*Thresholding*)

À partir de l'image en niveaux de gris $f_{\text{gris}}(x, y)$, une opération fondamentale est le **seuillage**, qui produit une image **binaire** (uniquement noir et blanc). Pour cela, on choisit une valeur de coupure T (généralement dans l'intervalle $[0, 255]$) et on définit :

$$f_{\text{bin}}(x, y) = \begin{cases} 255 & \text{si } f_{\text{gris}}(x, y) > T \\ 0 & \text{sinon} \end{cases} \tag{1.4}$$

Exemple : Avec $T = 128$, les pixels dont l'intensité est supérieure à 128 deviennent blancs (255) ; les autres deviennent noirs (0).

Le seuillage est largement utilisé pour **segmenter les objets du fond**, extraire les contours ou créer des masques binaires pour un traitement ultérieur.

> **Remarque :** La valeur 255 représente le blanc maximal dans les images 8 bits, tandis que 0 représente le noir absolu.

Exemple pratique de conversion et de seuillage

La Figure 1.8 illustre les principales étapes pour transformer une image couleur en niveaux de gris, puis la convertir en image binaire par seuillage. Le code suivant implémente ces étapes :

```
%%writefile tmp/fig_01_processamento_basico.cpp
#define MM_OUT "tmp/fig_01_processamento_basico.png"
#include "morph.hpp"
#include <iostream>
#include <vector>
#include <string>
#include <filesystem>

int main() {
// [pdi:state-io] auto-generated - do not edit by hand
mm::Image img = mm::_read_state("tmp/state/img_34.png");
// [pdi:state-io:end]

    // #| quarto-raw: false
    // #| label: fig-01-processamento-basico
    // #| fig-cap: "Processamento básico de imagens: (a) imagem original, (b) imagem em tons
    de cinza, (c) imagem binarizada por limiar (T=128)."
    // #| echo: true
    // #| output: true

    // 1. Converter para Tons de Cinza
    mm::Image img_gray = mm::gray(img);

    // 2. Aplicar limiar (Pixels > 128 tornam-se 255, outros 0)
    int limiar = 128;
    mm::Image img_binaria = mm::threshold(img_gray, limiar);

    // Uso da nova função
    mm::show(
        std::vector<mm::Image>{img, img_gray, img_binaria},
        MM_OUT,
        std::vector<std::string>{"Original", "Tons de Cinza", "Binária (T=128)"},
        3
    );


// [pdi:state-io] auto-generated - do not edit by hand
std::filesystem::create_directories("tmp/state");
mm::write(img_gray, "tmp/state/img_gray_38.png");
// [pdi:state-io:end]

// [pdi:panel-io] auto-generated - do not edit by hand
std::filesystem::create_directories("tmp");
mm::write(img, "tmp/fig_01_processamento_basico_0.png");
mm::write(img_gray, "tmp/fig_01_processamento_basico_1.png");
mm::write(img_binaria, "tmp/fig_01_processamento_basico_2.png");
// [pdi:panel-io:end]
return 0;
}
```

```
Overwriting tmp/fig_01_processamento_basico.cpp
```

```
!g++ -I. -std=c++17 tmp/fig_01_processamento_basico.cpp -o tmp/fig_01_processamento_basico \
  && ./tmp/fig_01_processamento_basico \
  && test -f "tmp/fig_01_processamento_basico.png" \
  || echo "  mm::show não gravou tmp/fig_01_processamento_basico.png"
```

```
[1] Original
[2] Tons de Cinza
[3] Binária (T=128)
```

```
try:
    mm.show(
        [
            mm.read("tmp/fig_01_processamento_basico_0.png"),
            mm.read("tmp/fig_01_processamento_basico_1.png"),
            mm.read("tmp/fig_01_processamento_basico_2.png"),
        ],
        titles=[
            'Original',
            'Tons de Cinza',
            'Binária (T=128)',
        ],
        cols=3,
    )
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + "
    tmp/fig_01_processamento_basico_0.png (ver a versao Python)")
```

Figure 1.8: Processamento básico de imagens: (a) imagem original, (b) imagem em tons de cinza, (c) imagem binarizada por limiar (T=128).

1.12 Limiarisation par la méthode d'Otsu

Comme présenté dans la Équation 1.4, la limiarisation convertit une image en niveaux de gris en une image binaire en utilisant une valeur de seuil T. Jusqu'à présent, nous avons fixé $T = 128$ manuellement.

```
mm::Image img_bin_fixo = mm::threshold(img_gray, 128);
```

Cependant, le choix manuel de T n'est pas toujours trivial. La bibliothèque `mm` propose une alternative automatique : lorsque le seuil **n'est pas fourni**, la fonction `mm::threshold(img_gray)` calcule la valeur de T par la **méthode d'Otsu** (OTSU, 1979). Cette méthode, qui sera détaillée dans des chapitres ultérieurs, maximise la variance inter-classes de l'histogramme (fréquence de chaque niveau de gris), séparant automatiquement les pixels de l'objet et du fond.

Le code ci-dessous compare la limiarisation manuelle ($T = 128$) avec la limiarisation automatique (Otsu). La

binarisation par Otsu est effectuée avec `mm::threshold(img_gray)` ; comme l'API minimale de `morph.hpp` ne renvoie pas le T calculé, la valeur affichée dans le titre de la figure est récupérée avec `cv2.threshold` lors de l'étape d'affichage (même algorithme, même T).

```
%%writefile tmp/fig_01_otsu.cpp
#define MM_OUT "tmp/fig_01_otsu.png"
// Compile with: g++ -std=c++17 -o program program.cpp -I. -L. -lmorph

#include "morph.hpp"
#include <iostream>
#include <string>
#include <vector>
#include <filesystem>

int main() {
// [pdi:state-io] auto-generated - do not edit by hand
mm::Image img_gray = mm::_read_state("tmp/state/img_gray_38.png");
// [pdi:state-io:end]

    // img_gray is already available as a mm::Image

    // Limiarização com T fixo (manual)
    int T_fixo = 128;
    mm::Image img_bin_fixo = mm::threshold(img_gray, T_fixo);

    // Limiarização pelo método de Otsu (T automático)
    // T_otsu, img_bin_otsu = cv2.threshold(img_gray, 0, 255,
    //                                      cv2.THRESH_BINARY + cv2.THRESH_OTSU)
    int T_otsu = mm::otsu(img_gray);
    mm::Image img_bin_otsu = mm::threshold(img_gray);   // same T, Otsu when the arg is
omitted
    std::cout << "Limiar calculado por Otsu: T = " << T_otsu << "\n";
    // ou simplesmente:
    // img_bin_otsu = mm::threshold(img_gray)

    // Exibição lado a lado
    mm::show(
        std::vector<mm::Image>{img_gray, img_bin_fixo, img_bin_otsu},
        MM_OUT,
        std::vector<std::string>{"Tons de Cinza", "Binária (T=" + std::to_string(T_fixo) +
")", "Binária (Otsu, T=" + std::to_string(T_otsu) + ")"},
        3
    );


// [pdi:panel-io] auto-generated - do not edit by hand
std::filesystem::create_directories("tmp");
mm::write(img_gray, "tmp/fig_01_otsu_0.png");
mm::write(img_bin_fixo, "tmp/fig_01_otsu_1.png");
mm::write(img_bin_otsu, "tmp/fig_01_otsu_2.png");
// [pdi:panel-io:end]
return 0;
}
```

```
Overwriting tmp/fig_01_otsu.cpp
```

```
!g++ -I. -std=c++17 tmp/fig_01_otsu.cpp -o tmp/fig_01_otsu \
  && ./tmp/fig_01_otsu \
  && test -f "tmp/fig_01_otsu.png" \
  || echo "  mm::show não gravou tmp/fig_01_otsu.png"
```

```
Limiar calculado por Otsu: T = 95
[1] Tons de Cinza
[2] Binária (T=128)
[3] Binária (Otsu, T=95)
```

```python
try:
    T_otsu = mm.otsu(mm.read("tmp/fig_01_otsu_0.png", grayscale=True))
    mm.show(
        [
            mm.read("tmp/fig_01_otsu_0.png"),
            mm.read("tmp/fig_01_otsu_1.png"),
            mm.read("tmp/fig_01_otsu_2.png"),
        ],
        titles=[
            'Tons de Cinza',
            'Binária (T=128)',
            f"Binária (Otsu, T={T_otsu})",
        ],
        cols=3,
    )
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + " tmp/fig_01_otsu_0.png (ver
    a versao Python)")
```

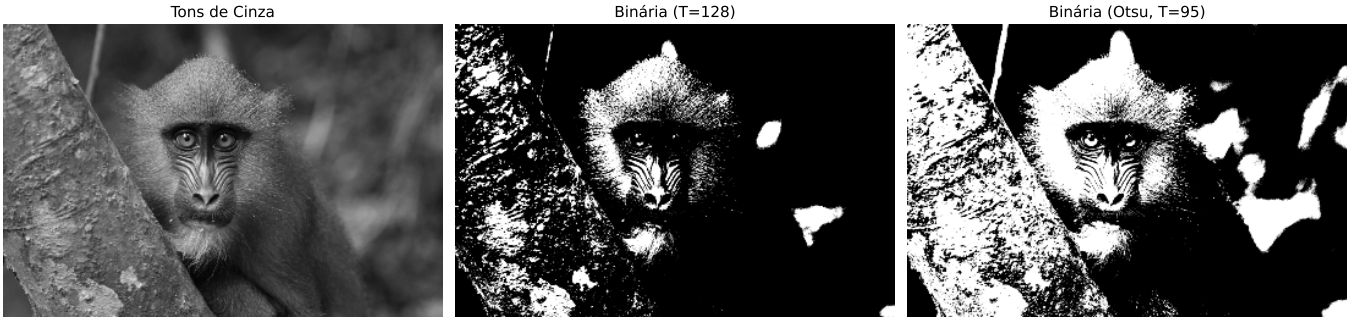


Figure 1.9: Comparação entre limiarização manual (T=128) e automática (Otsu) sobre a imagem em tons de cinza.

La Figure 1.9 montre que le seuil obtenu par Otsu s'adapte automatiquement à l'image, permettant une binarisation plus efficace qu'une valeur fixe, en particulier lorsque les intensités de l'objet et du fond sont bien séparées dans l'histogramme. Cette technique est largement utilisée dans les systèmes de vision par ordinateur (VC) pour la binarisation de documents, la détection d'objets et le prétraitement d'images.

Simplicité de la bibliothèque `morph.hpp`

Alors qu'OpenCV exige l'appel complet :

```cpp
cv::Mat img_bin;
double T_otsu = cv::threshold(img_gray, img_bin, 0, 255,
                              cv::THRESH_BINARY | cv::THRESH_OTSU);
```

`morph.hpp` abstrait toute cette complexité : il suffit d'appeler `mm::threshold(img_gray)`. Le seuil d'Otsu est calculé automatiquement et l'image binaire est renvoyée directement. Cette approche permet de se concentrer sur le concept, et non sur les détails d'implémentation.

1.13 Accès aux pixels

Dans `morph.hpp`, l'image est un `mm::Image` avec un buffer linéaire `data` (ligne après ligne, canaux entrelacés) et la méthode `at(y, x, c)` pour un accès individuel, suivant la convention matricielle **ligne** (axe Y) et **colonne** (axe X) : `img.at(ligne, colonne)` pour les niveaux de gris et `img.at(ligne, colonne, canal)` pour chaque canal d'une image RVB.

Le code de la Figure 1.10 démontre comment extraire ces valeurs dans des images couleur (RVB) et en niveaux de gris, ainsi que comment isoler le voisinage immédiat du point d'intérêt.

Accès aux pixels en C++ (`mm::Image`) :

- **Niveaux de gris :** `img_gray.at(r, c)` retourne un `unsigned char` (de 0 à 255) correspondant à l'intensité de gris du pixel.
- **RGB :** `img.at(r, c, 0)`, `img.at(r, c, 1)`, `img.at(r, c, 2)` accèdent aux canaux **R**, **G** et **B** — un seul canal à la fois ; il n'existe pas de vecteur `[R, G, B]`.
- **Voisinage 3×3 :** parcouru par deux boucles imbriquées sur `img_gray.at(r + dy, c + dx)`, avec `dy, dx` dans $\{-1, 0, 1\}$ — il n'y a pas de découpage (*slicing*) comme dans NumPy. Ce balayage constitue la base des filtres spatiaux et des convolutions.
- Il n'y a pas de tracé interactif (équivalent à `plt.plot`) : pour marquer la position du pixel, la cellule de code suivante **dessine un carré rouge évidé** autour de celui-ci sur une copie de l'image (`marcada = img.copy()`, puis `marcada.at(...)`) et l'affiche avec `mm::show`.

L'indexation est *basée sur zéro* : `(0,0)` correspond au coin supérieur gauche. La première dimension contrôle la **hauteur** (lignes/Y) et la seconde la **largeur** (colonnes/X).

1.14 Résumé

Dans ce chapitre, nous avons présenté les fondements de la représentation des images numériques : la définition du pixel, la structuration des images en matrices et l'impact de l'échantillonnage et de la quantification sur la qualité finale :

- **Image numérique** = fonction $f(x, y)$ qui associe aux coordonnées des intensités (scalaires ou vectorielles).
- **Domaine** : ensemble fini $\mathbb{E} = \{(x, y) \in \mathbb{Z}^2 \mid 0 \leq x < L,\ 0 \leq y < H\}$.
- **Types principaux** : binaire ($\mathcal{V} = \{0, 255\}$), niveaux de gris ($\mathcal{V} = [0, 255]$) et RVB ($\mathcal{V} = [0, 255]^3$).
- **Seuillage** convertit les niveaux de gris en image binaire ; la **méthode d'Otsu** détermine automatiquement la valeur de coupure en maximisant la variance inter-classes.
- A `morph.hpp` (ou `mm::`) oferece funções didáticas para operações básicas de PDI, como `mm::gray()`, `mm::threshold()`, `mm::show()` (sobrecarregada para 1 imagem ou várias).
- **Piège de copie** : `std::vector` a une sémantique de valeur, mais les pointeurs/références partagés entre les « lignes » d'une matrice réintroduisent le même problème que NumPy — préférez `mm::Image`, qui copie également par valeur.
- Accès aux pixels via `img.at(ligne, colonne)`, avec une indexation ***à base zéro***.

Le chapitre 2 traitera des **histogrammes et de l'égalisation de contraste**.

1.15 🤖 Utilisation du Notebook Gemini comme tuteur complémentaire

Dans cette édition, en plus des *notebooks* interactifs sur Google Colab, le **Notebook Gemini** est disponible comme outil d'étude complémentaire. La plateforme utilise exclusivement les documents fournis par l'auteur comme base de connaissances, garantissant des réponses cohérentes avec le contenu du livre.

```
# Not yet ported to this language in this version - conceptual reference in Python.

# 1. Coordonnées du pixel cible (ligne, colonne)
r, c = 600, 800

# 2. Accès direct aux valeurs du pixel
pixel_cinza = img_gray[r, c]
print(f"Pixel cible ({r}, {c}):")
print(f"  Niveaux de gris (scalaire) : {pixel_cinza}")
print(f"  Couleur (canaux RGB)       : R={img[r, c, 0]}, G={img[r, c, 1]}, B={img[r, c, 2]}")

# 3. Voisinage 3x3 autour de (r, c) ; le pixel central est entre crochets
print("Matrice de voisinage 3x3 (niveaux de gris) :")
for dy in range(-1, 2):
    linha = ""
    for dx in range(-1, 2):
        v = img_gray[r + dy, c + dx]
        if dy == 0 and dx == 0:
            linha += f"[{v:3d}]"
        else:
            linha += f" {v:3d} "
    print("  " + linha)
# 4. Marque la position du pixel avec un carré rouge vide (bord de
#    3 px) sur une copie de l'image et affiche (équivalent au point de matplotlib).
marcada = img.copy()
lado = 20   # demi-côté du carré, en pixels
esp = 5     # épaisseur du bord

for x in range(c - lado, c + lado + 1):
    for w in range(esp):
        marcada[r - lado + w, x, 0] = 255
        marcada[r - lado + w, x, 1] = 0
        marcada[r - lado + w, x, 2] = 0
        marcada[r + lado - w, x, 0] = 255
        marcada[r + lado - w, x, 1] = 0
        marcada[r + lado - w, x, 2] = 0

for y in range(r - lado, r + lado + 1):
    for w in range(esp):
        marcada[y, c - lado + w, 0] = 255
        marcada[y, c - lado + w, 1] = 0
        marcada[y, c - lado + w, 2] = 0
        marcada[y, c + lado - w, 0] = 255
        marcada[y, c + lado - w, 1] = 0
        marcada[y, c + lado - w, 2] = 0

mm.show(marcada, title=f"Localisation du pixel ({r}, {c})")
```

Figure 1.10

🎓 Étudiez avec le tuteur intelligent

Accédez à l'environnement du chapitre via le *lien* ci-dessous et explorez particulièrement les options **Guide d'étude** et **Conversation** pour approfondir votre compréhension.
🚀 ACCÉDER AU NOTEBOOK GEMINI : CHAPITRE 01

🌐 **Langue et langage de programmation**

Le projet de ce chapitre dans le Notebook Gemini a été construit uniquement avec le texte en **portugais** et les exemples de code en **Python**. Si vous étudiez avec l'édition anglaise ou française, ou si vous suivez le parcours en C++, les réponses du tuteur peuvent ne pas correspondre exactement à la version que vous lisez.

⚠️ **Avis concernant le contenu généré par l'IA**

L'IA est une alliée puissante pour les études, mais le contenu généré peut contenir des **erreurs ou des imprécisions**. Consultez toujours **des livres, des articles scientifiques et d'autres sources académiques fiables** pour valider les informations. Chaque fois que possible, exécutez les exemples pratiques fournis dans ce chapitre pour vérifier les résultats.

Fonctionnalités Disponibles sur la Plateforme

Le **Gemini Notebook** propose une suite avancée d'outils basés sur l'IA pour transformer le contenu statique du livre en une expérience d'apprentissage dynamique et multimédia. La plateforme utilise des techniques de **RAG** (*Retrieval-Augmented Generation*), fondées sur les travaux de Lewis (2020), pour ancrer les réponses strictement dans les documents fournis et minimiser l'apparition d'hallucinations.

Les principales fonctionnalités incluent :

- **Résumés Multimodaux (Audio et Vidéo)** : Génération de conversations naturelles entre experts sous forme de **Résumé Audio** (style *podcast*) et de **Résumé Vidéo**, discutant des thèmes centraux du chapitre, comme les différences entre PDI et VC, ou l'interprétation de transformations telles que le seuillage et la méthode d'Otsu.
- **Visualisation de Structures (Carte Mentale et Infographie)** : Création automatique de diagrammes connectant visuellement les concepts, par exemple, le flux de traitement depuis la capture de l'image numérique, en passant par la conversion en niveaux de gris, le seuillage et la segmentation binaire.
- **Outils d'Évaluation (Test et Cartes Pédagogiques)** : Génération de **Tests** à choix multiples et de **Cartes Pédagogiques** (*flashcards*) pour la consolidation des connaissances, basés sur le texte original (ex. : questions sur la formule de conversion RVB→niveaux de gris ou sur le fonctionnement du seuil global et de celui d'Otsu).
- **Soutien à la Présentation (*Slides* et Rapports)** : Aide à la structuration de **Présentations de *Slides*** et à la rédaction de **Rapports** techniques, facilitant la communication des résultats d'expériences avec des images.
- **Analyse de Données (Tableau de Données)** : Organisation des données extraites du texte dans des tableaux structurés, aidant à la compréhension d'exemples pratiques, comme la comparaison entre différentes valeurs de seuil.
- ***Chat* Contextualisé** : Permet de questionner directement le code et la théorie, comme : *"Comment implémenter la conversion RVB en niveaux de gris en utilisant les pondérations de la norme UIT-R BT.601 ?"* ou *"Que devient l'image binaire si je choisis un seuil T=200 au lieu de T=128 ?"*.

1.16 Liste d'exercices

1. (15 %) Avec vos propres mots, définissez **image numérique** et **pixel**. Donnez un exemple concret de la manière dont une image colorée (RVB) est représentée sous forme matricielle dans l'ordinateur.

2. (15 %) Expliquez les différences entre **image binaire**, **niveaux de gris (8 bits)** et **image colorée RVB**, en indiquant la plage de valeurs possibles pour chaque pixel dans chaque type.

3. (20 %) En considérant la formule de conversion RVB → niveaux de gris du standard ITU-R BT.601 :

$$g = 0.299\,R + 0.587\,G + 0.114\,B$$

   Calculez la valeur du pixel en niveaux de gris pour $(R, G, B) = (80, 180, 30)$. Arrondissez à l'entier le plus proche.

4. (20 %) Qu'est-ce que la **seuillage** (*thresholding*) ? Expliquez la différence entre choisir un seuil T fixe (par exemple $T = 128$) et utiliser la **méthode d'Otsu** pour la détermination automatique du seuil. En quelques mots, comment la méthode d'Otsu choisit-elle le seuil ?

5. (15%) Dans le contexte de la bibliothèque didactique `mm` discutée dans le chapitre, répondez :

   - a) (7,5 %) Comment accède-t-on à la valeur du pixel à la position (ligne=50, colonne=60) d'une image en niveaux de gris `img_gray` ?
   - b) (7,5 %) Quel est l'avantage d'utiliser `mm::threshold(img_gray)` sans passer le seuil ? Comparez avec l'appel équivalent dans OpenCV (`cv::threshold`).

6. (15 %) Que représentent les champs `h`, `w` et `channels` d'un `mm::Image` pour une image RVB ? Donnez un exemple concret avec une image de 640×480 pixels.

Références du chapitre

Les fondements théoriques de ce chapitre comprennent les ouvrages suivants sur le TNI et la VC :

- Gonzalez (2018) pour les fondements du **Traitement Numérique des Images** (TNI).
- Singh (2019) pour la mise en œuvre pratique des méthodes de **traitement et d'analyse d'images**.
- Szeliski (2022) pour l'étude de la VC et des algorithmes fondamentaux.
- Bradski (2008) pour l'application de la bibliothèque **OpenCV** dans un environnement Python.
- Lewis (2020) pour le concept de **génération augmentée par récupération (RAG)**, utilisé pour le support du traitement des informations de ce matériel.

Exécuter Colab Ouvrir GitHub

Avertissement : Ce document est une traduction automatique et peut contenir des erreurs. En cas de divergence, consultez la version originale en portugais, qui fait foi.

1.17 💻 Partie Pratique avec Exercices de Programmation

🎯 Objectif de ce Cahier

Les **Exercices de Programmation (EPs)** présentés ci-dessous peuvent également être soumis dans les activités Moodle (activités VPL) qui fournissent une *rétroaction* automatique.

Ce cahier a été développé pour surmonter les limitations d'utilisation de Moodle. Avec lui, il faut :

1. **Développer :** Écrire et éditer votre solution directement dans l'environnement Colab.
2. **Valider :** Tester votre code localement en utilisant les **mêmes cas de test** que Moodle.
3. **Organiser :** Sauvegarder vos codes des activités VPL de manière sécurisée.
4. **Évaluer :** Lorsque vous êtes connecté à Moodle, il suffit de copier votre solution et de cliquer sur **Évaluer** dans Moodle (**si vous êtes sur le réseau de l'UFABC**) pour enregistrer votre note officielle.

§ Instructions Étape par Étape

Dans un environnement d'exécution (comme VSCode, Jupyter ou Colab), suivez l'ordre ci-dessous pour configurer l'environnement et valider vos exercices :

Préparation de l'Environnement

Exécutez la cellule de code ci-dessous pour télécharger `morph.py` et `testsuite.py` depuis le dépôt du cours — uniquement s'ils n'existent pas déjà dans le répertoire local. Avec les deux fichiers dans `./`, le *notebook* et les sous-processus du `TestSuite` trouvent le module sans configurations de chemin supplémentaires.

Note : Le script `testsuite.py` recherchera automatiquement les cas de test dans `all/{cap}/cases` sur GitHub.

Écriture du Code

Enregistrez votre solution dans une cellule de code en utilisant la commande magique `%%writefile`. Le nom du fichier doit suivre le modèle `EPX_Y.*`, où `X` est le chapitre, `Y` est l'exercice et `*` est l'extension du langage.

Exemple : `%%writefile EP01_01.py`

Téléchargement

Téléchargez `morph.py` et `testsuite.py` en exécutant la cellule ci-dessous :

```
import os, urllib.request

os.makedirs("tmp/state", exist_ok=True)  # artefacts de build du parcours C++ (.cpp, binaire,
PNGs)

url = "https://raw.githubusercontent.com/fzampirolli/pdi-vc/master/morph/config.py"
if not os.path.exists("config.py"):
    urllib.request.urlretrieve(url, "config.py")

# Le noyau est Python même dans le parcours C++ : `mm` (morph.py) est utilisé par les
# simulateurs, par l'affichage des figures que le binaire C++ génère et par
# l'état mm::Image entre cellules. cpp=True télécharge aussi le parcours compilé
# (morph.hpp + stb_image*.h), utilisé dans le #include des cellules %%writefile *.cpp.
import config
config.setup(testsuite=True, cpp=True)
from morph import mm
from testsuite import TestSuite
```

```
Environnement prêt. Morph : 1.1.9 | OpenCV : 5.0.0 | TestSuite : 1.1.2
```

Exécution des tests

Après avoir enregistré le fichier avec votre solution, exécutez la commande ci-dessous (dans une nouvelle cellule) pour évaluer les tests automatiques :

```
TestSuite("EP01_01.extensão").run()
```

Remplacez l'extension selon le langage utilisé :

Langage	Extension
Python	`.py`
Java	`.java`
C	`.c`
C++	`.cpp`

Langage	Extension
JavaScript	`.js`
R	`.r`

Comment cela fonctionne : Le `TestSuite` télécharge les cas de test depuis GitHub, exécute votre programme avec chaque entrée et compare la sortie avec celle attendue – calculant automatiquement votre note.

Pour tester directement du code Python, sans enregistrer de fichier, utilisez `run_code(codigo)` en passant le code comme *chaîne de caractères* dans une variable `codigo` :

```
codigo = """
from morph import mm
# ... votre code ici ...
"""
TestSuite("EP04_01").run_code(codigo)
```

⚠️ Important : Règles et Bonnes Pratiques

🔹 À propos de la Saisie des Données

Votre programme doit lire l'entrée standard (clavier).

- **Python :** Utilisez `input()`.
- **Autres langages :** Utilisez la commande de lecture standard équivalente (`cin`, `Scanner`, etc.).

🔹 Configuration de l'IA dans Colab

Pour un meilleur apprentissage, il est recommandé de désactiver la complétion automatique de code par l'IA, car elle ne sera pas disponible lors des évaluations. Par exemple dans le navigateur Chrome :

- Allez dans : **Outils > Paramètres > IA générative**
- Décochez : **Activer la génération de code**

🔹 Intégrité Académique (Plagiat)

Cette ressource de tests locaux s'applique aux EPs **sans variations**. Cependant :

- **Individualité :** Chaque étudiant doit développer sa propre solution.
- **Détection de Similarité :** Le professeur utilise des outils qui détectent les copies, **même avec un changement de noms de variables ou d'espaces blancs**.

1.17.1 EP01_01 📏 Trois métriques de distance en TNI

Dans cette activité, vous devez écrire un programme qui calcule les trois distances classiques en TNI : **euclidienne (L2)**, **City-Block (L1)** et ***Chessboard* (L∞)**.

- Lisez **4 nombres réels** qui représentent les coordonnées : A_x, A_y, B_x, B_y.
- Calculez les trois distances à l'aide des formules :

$$d_{\text{Euclidienne}} = \sqrt{(B_x - A_x)^2 + (B_y - A_y)^2}$$

$$d_{\text{City-block}} = |B_x - A_x| + |B_y - A_y|$$

$$d_{\text{Chessboard}} = \max\left(|B_x - A_x|,\ |B_y - A_y|\right)$$

- Affichez les trois résultats, chacun sur une ligne, formatés avec **deux décimales**, dans l'ordre : **euclidienne**, ***City-block***, ***Chessboard***.

📌 **Important** :

- Utilisez les fonctions mathématiques standard de votre langage : `math.sqrt`, `abs` (ou `fabs`) et `max`.
- La sortie doit contenir **uniquement les nombres** (un par ligne), sans texte supplémentaire.
- Voir un simulateur interactif pour cette question à la **Figure 1.11** (graphique avec glissement des points et visualisation des trois métriques).

1.17.1.1 🖼️ Pourquoi cela importe-t-il ? – Coût computationnel

Dans une image **1000×1000 pixels** (1 million de pixels), calculer la distance de chaque pixel à un point de référence exige **1 million d'opérations**. Le choix de la métrique affecte la performance :

Métrique	Opérations par pixel	Coût relatif (1M pixels)	Quand l'utiliser
Euclidienne (L2)	2 soustractions, 2 multiplications, 1 addition, 1 `sqrt`	🔴 Plus coûteuse – `sqrt` est lente	Distance « réelle » dans l'espace continu
***City-block* (L1)**	2 soustractions, 2 `abs`, 1 addition	🟡 Modérée – sans racine carrée	Grilles, robotique, images binaires
***Chessboard* (L∞)**	2 soustractions, 2 `abs`, 1 `max`	🟢 Plus efficace	Déplacements de pièces, morphologie

La fonction `sqrt` est computationnellement plus coûteuse que des opérations comme l'addition, la soustraction, la multiplication et la valeur absolue. Sur les processeurs modernes, la différence peut être faible (environ 1,5× à 3×), mais dans les systèmes embarqués ou dans des boucles de millions d'itérations, tout gain compte. Pour cette raison, **lorsque l'objectif est uniquement de comparer des distances** (ex. : trouver le point le plus proche), utilisez la **distance euclidienne au carré**.

1.17.1.2 📋 Tâche (spécification pour VPL)

Entrée :
Une seule ligne avec quatre nombres réels : `Ax Ay Bx By`

Sortie :
Trois lignes, chacune avec un nombre réel à deux décimales (euclidienne, City-block, Chessboard).

1.17.1.3 📌 Exemples

Entrée	Sortie	Observation

`0`
`0`
`3`
`4`
	`5.00`
	`7.00`
	`4.00`
		Triangle 3-4-5

`0`
`0`
`1`
`1`
	`1.41`
	`2.00`
	`1.00`
		Diagonale unitaire

Exemple de test de `sqrt` en Python, avec `timeit` isolant chaque opération :

```
%%writefile tmp/mm_out_1.cpp
#include <iostream>
#include <chrono>
#include <cmath>
#include <iomanip>

int main() {
    const int N = 50'000'000;

    auto apenas_soma = []() {
        double a = 3.0, b = 4.0;
        return a + b;
    };

    auto soma_e_sqrt = []() {
        double a = 3.0, b = 4.0;
        return std::sqrt(a*a + b*b);
    };

    auto start_soma = std::chrono::high_resolution_clock::now();
    for (int i = 0; i < N; ++i) {
        apenas_soma();
    }
    auto end_soma = std::chrono::high_resolution_clock::now();
    double t_soma = std::chrono::duration<double>(end_soma - start_soma).count();

    auto start_sqrt = std::chrono::high_resolution_clock::now();
    for (int i = 0; i < N; ++i) {
        soma_e_sqrt();
    }
    auto end_sqrt = std::chrono::high_resolution_clock::now();
    double t_sqrt = std::chrono::duration<double>(end_sqrt - start_sqrt).count();

    std::cout << std::fixed << std::setprecision(3);
    std::cout << "Soma simple       : " << t_soma << " s\n";
    std::cout << "Soma + sqrt       : " << t_sqrt << " s\n";
    std::cout << "Razão (sqrt/soma) : " << std::setprecision(2) << (t_sqrt/t_soma) << "x\n";

    return 0;
}
```

```
Overwriting tmp/mm_out_1.cpp
```

```
!g++ -I. -std=c++17 tmp/mm_out_1.cpp -o tmp/mm_out_1 \
  && ./tmp/mm_out_1
```

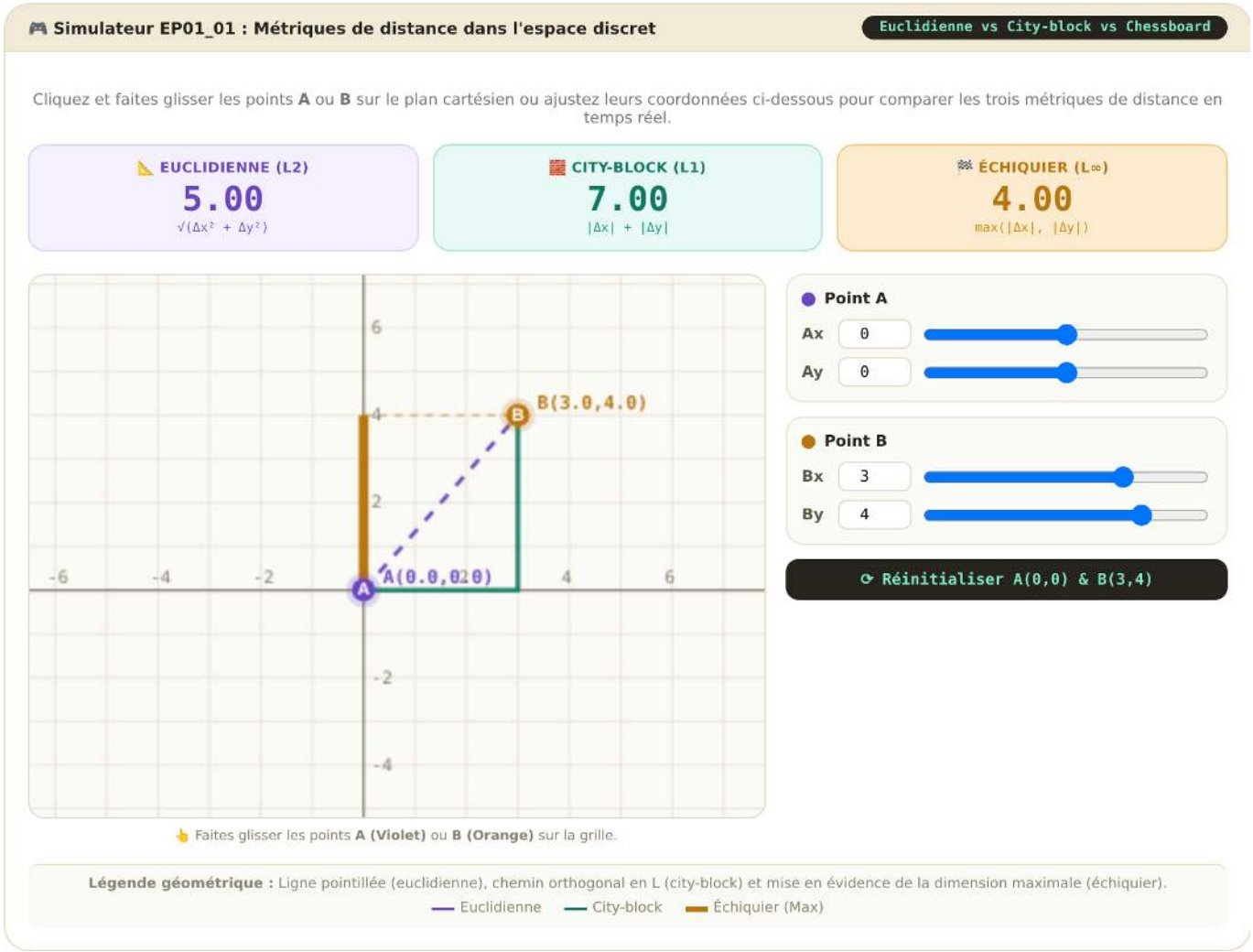
```
Soma simple       : 0.222 s
Soma + sqrt       : 0.478 s
Razão (sqrt/soma) : 2.15x
```

1.17.1.4 🐍 Python

Il suffit de créer une cellule de code normale et d'y insérer le code Python. L'entrée peut être simulée à l'aide de `input()`, qui fonctionne normalement.

Exemple de cellule :

```
%%writefile EP01_01.py
# Code Python
x1,y1,x2,y2 = int(input()), int(input()), int(input()), int(input())
```



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.fr/cap01/sim-ep0101-distancia.html

Figure 1.11: Simulateur EP01_01 : Distances Euclidienne, City-block et Échiquier

```
# Calcul des différences
dx = abs(x2 - x1)
dy = abs(y2 - y1)

# 1. Distance euclidienne (L2)
dist_euclidiana = (dx**2 + dy**2)**0.5

# 2. Distance City-block / Manhattan (L1)
dist_city_block = dx + dy

# 3. Distance Chessboard / Chebyshev (Linf)
dist_chessboard = max(dx, dy)

# Sortie formatée selon les cas de test
print(f"{dist_euclidiana:.2f}")
print(f"{dist_city_block:.2f}")
print(f"{dist_chessboard:.2f}")
```

```
Overwriting EP01_01.py
```

```
# Attend que vous saisissiez 4 nombres entiers lors de l'exécution de cette cellule.
# Dans Jupyter ou Google Colab, la magie %run -i permet au script de lire depuis le clavier.
# Dans un terminal standard, vous utiliseriez : python3 EP01_01.py (sans le '!' et sans
'%run').

# %run -i EP01_01.py
```

```
# Envoie 4 entiers comme entrée standard (stdin) au script EP01_01.py en utilisant un pipe
!echo -e "0\n0\n4\n4" | python3 EP01_01.py
```

```
5.66
8.00
4.00
```

```
TestSuite("EP01_01.py").run()
```

```
<IPython.core.display.HTML object>
```

1.17.1.5 ☕ Java

Pour exécuter Java dans Colab, vous devez utiliser une cellule avec le préfixe `%%writefile` pour enregistrer le code dans un fichier, le compiler et l'exécuter.

```
%%writefile EP01_01.java
import java.util.Scanner;

class EP01_01 {
    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);

        double x1 = s.nextDouble(), y1 = s.nextDouble();
        double x2 = s.nextDouble(), y2 = s.nextDouble();

        double dx = Math.abs(x2 - x1);
        double dy = Math.abs(y2 - y1);

        System.out.printf("%.2f\n", Math.sqrt(dx*dx + dy*dy));
        System.out.printf("%.2f\n", dx + dy);
```

```
        System.out.printf("%.2f\n", Math.max(dx, dy));
    }
}
```

```
Overwriting EP01_01.java
```

```
!javac EP01_01.java
!echo -e "0\n0\n4\n4" | java EP01_01
```

```
5,66
8,00
4,00
```

```
TestSuite("EP01_01.java").run()
```

```
<IPython.core.display.HTML object>
```

1.17.1.6 💻 C

De maneira similaire, utilisez **%%writefile** pour enregistrer le code, puis compilez et exécutez. Pour C, nous utilisons le compilateur **GCC**.

```
# Installer le compilateur GCC et les outils de build
# Le build-essential inclut gcc, g++, make, etc.
import platform, shutil, subprocess

def instalar_gcc():
    if shutil.which("gcc"):
        print("  GCC déjà disponible."); return
    if platform.system() != "Linux":
        print("  Mac : xcode-select --install | Windows : WSL ou MinGW"); return
    try: import google.colab; cmd = ["apt-get", "install", "-y", "build-essential"]
    except ImportError:       cmd = ["sudo", "apt-get", "install", "-y", "build-essential"]
    subprocess.run(cmd, check=True)
    print("  build-essential installé !")

instalar_gcc()
```

```
  GCC déjà disponible.
```

```
%%writefile EP01_01.c
#include <stdio.h>
#include <math.h>

int main() {
    double x1, y1, x2, y2;
    if (scanf("%lf %lf %lf %lf", &x1, &y1, &x2, &y2) != 4) return 0;

    double dx = fabs(x2 - x1);
    double dy = fabs(y2 - y1);

    // Euclidiana, City-block e Chessboard
    printf("%.2f\n", sqrt(dx * dx + dy * dy));
    printf("%.2f\n", dx + dy);
    printf("%.2f\n", fmax(dx, dy));

    return 0;
}
```

```
Overwriting EP01_01.c
```

```
# Compile le fichier .c en générant l'exécutable EP01_01
# -lm est utilisé pour lier la bibliothèque mathématique (math.h) si nécessaire

!gcc EP01_01.c -o EP01_01 -lm
!echo -e "0\n0\n4\n4" | ./EP01_01
```

```
5.66
8.00
4.00
```

```
TestSuite("EP01_01.c").run()
```

```
<IPython.core.display.HTML object>
```

1.17.1.7 💻 C++

De manière similaire à Java, utilisez **%%writefile** pour enregistrer le code, puis compilez et exécutez. Rappelez-vous que, dans Colab, il est également nécessaire d'installer ce qui suit :

```
# Installer le compilateur G++ pour C++
# Le build-essential inclut g++, make, etc.
import platform, shutil, subprocess

def instalar_gpp():
    if shutil.which("g++"):
        print("  G++ déjà disponible."); return
    if platform.system() != "Linux":
        if platform.system() == "Darwin":
            print("  Mac : xcode-select --install")
        else:
            print("  Windows : utilisez WSL ou MinGW (https://www.mingw-w64.org)")
        return
    try: import google.colab; cmd = ["apt-get", "install", "-y", "build-essential"]
    except ImportError:       cmd = ["sudo", "apt-get", "install", "-y", "build-essential"]
    subprocess.run(cmd, check=True)
    print("  Compilateur C++ prêt.")

instalar_gpp()
```

```
  G++ déjà disponible.
```

```
%%writefile EP01_01.cpp
#include <iostream>
#include <iomanip>
#include <cmath>
#include <algorithm>

int main() {
    double x1, y1, x2, y2;
    if (!(std::cin >> x1 >> y1 >> x2 >> y2)) return 0;

    double dx = std::abs(x2 - x1);
    double dy = std::abs(y2 - y1);

    std::cout << std::fixed << std::setprecision(2);
```

```
    // Euclidiana, City-block e Chessboard
    std::cout << std::sqrt(dx*dx + dy*dy) << std::endl;
    std::cout << (dx + dy) << std::endl;
    std::cout << std::max(dx, dy) << std::endl;

    return 0;
}
```

```
Overwriting EP01_01.cpp
```

```
!g++ EP01_01.cpp -o EP01_01
!echo -e "0\n0\n4\n4" | ./EP01_01
```

```
5.66
8.00
4.00
```

```
TestSuite("EP01_01.cpp").run()
```

```
<IPython.core.display.HTML object>
```

1.17.1.8 🌐 JavaScript (Node.js)

Pour JavaScript, utilisez `%%writefile` pour créer le fichier et exécutez-le avec Node :

```
%%writefile EP01_01.js
function escreva(s) { try { document.write(s + "<br>"); } catch(e) { console.log(s); } }

process.stdin.once('data', data => {
  const valores = data.toString().trim().split(/\s+/);
  const x1 = parseFloat(valores[0]);
  const y1 = parseFloat(valores[1]);
  const x2 = parseFloat(valores[2]);
  const y2 = parseFloat(valores[3]);

  const dx = Math.abs(x2 - x1);
  const dy = Math.abs(y2 - y1);

  // Euclidiana, City-block e Chessboard
  escreva(Math.sqrt(dx * dx + dy * dy).toFixed(2));
  escreva((dx + dy).toFixed(2));
  escreva(Math.max(dx, dy).toFixed(2));
});
```

```
Overwriting EP01_01.js
```

```
TestSuite("EP01_01.js").run()
```

```
<IPython.core.display.HTML object>
```

1.17.1.9 📊 R

Dans Colab, on peut exécuter du code R directement en utilisant la magie `%%R`.
Le programme doit lire **quatre nombres** (x1, y1, x2, y2) et afficher la distance euclidienne avec **deux décimales**.

Exemple de cellule :

```
%%R
dados <- scan(file = "stdin", n = 4, quiet = TRUE)
x1 <- dados[1]; y1 <- dados[2]; x2 <- dados[3]; y2 <- dados[4]
dist <- sqrt((x2 - x1)^2 + (y2 - y1)^2)
cat(sprintf("%.2f", dist))
```

```
# Installer R et Rscript
# Le paquet r-base installe l'environnement R complet, y compris Rscript
import platform, shutil, subprocess

def instalar_r():
    if shutil.which("R"):
        print(" R déjà disponible."); return
    if platform.system() != "Linux":
        if platform.system() == "Darwin":
            print(" Mac : https://cran.r-project.org/bin/macosx/")
        else:
            print(" Windows : https://cran.r-project.org/bin/windows/base/")
        return
    try: import google.colab; cmd = ["apt-get", "install", "-y", "r-base"]
    except ImportError:       cmd = ["sudo", "apt-get", "install", "-y", "r-base"]
    subprocess.run(cmd, check=True)
    print(" Environnement R prêt.")

instalar_r()
```

```
 R déjà disponible.
```

Pour tester de la même manière que dans les exemples précédents, il faut utiliser l'entrée standard (stdin) dans le terminal ou adapter le code comme ci-dessous :

```
%%writefile EP01_01.r
dados <- scan(file = "stdin", n = 4, quiet = TRUE)
dx <- abs(dados[3] - dados[1])
dy <- abs(dados[4] - dados[2])

# Sorties : Euclidienne, City-block et Chessboard
cat(sprintf("%.2f\n%.2f\n%.2f\n",
    sqrt(dx^2 + dy^2),
    dx + dy,
    max(dx, dy)))
```

```
Overwriting EP01_01.r
```

```
!echo -e "0\n0\n4\n4" | Rscript EP01_01.r
```

```
5.66
8.00
4.00
```

```
TestSuite("EP01_01.r").run()
```

```
<IPython.core.display.HTML object>
```

1.17.2 EP01_02 📊 Performance Prédictive — Métriques de ML en VC

Dans cette activité, vous entrerez dans le monde de l'**Apprentissage Automatique** (*Machine Learning*). Votre objectif est d'évaluer la performance d'un classificateur binaire en calculant des métriques à partir

d'une **Matrice de Confusion**.

1.17.2.1 🧠 Pourquoi la bonne métrique est-elle importante ?

Imaginez un détecteur de **pièces de R$ 1,00**. L'impact de l'erreur définit la métrique prioritaire :

Métrique	Exemple Pratique	Importance en TDI/VC
Exactitude	**Comptage de Grains**	Utile lorsque les classes sont équilibrées (ex : la moitié des grains défectueux, l'autre moitié sains).
Précision	**Sécurité/Biométrie**	Cruciale pour éviter les **Faux Positifs** (ex : ne pas permettre à un imposteur d'accéder à un système par erreur de reconnaissance).
Sensibilité	**Santé (Tumeurs)**	Cruciale pour éviter les **Faux Négatifs** (ex : ne pas laisser une tumeur passer inaperçue lors d'un examen radiologique).
Score F1	**Billets de Banque**	Idéale pour un **équilibre** entre ne pas rejeter de vrais billets et ne pas accepter de faux billets.

1.17.2.2 📊 La Matrice de Confusion

	Prédit Positif	**Prédit Négatif**
Réel Positif	**VP** (Vrai Positif)	**FN** (Faux Négatif)
Réel Négatif	**FP** (Faux Positif)	**VN** (Vrai Négatif)

Tâche :

1. Lisez **4 valeurs entières** dans l'ordre : `VP`, `FN`, `FP`, `VN`.
2. Calculez les métriques à l'aide des formules :

$$\text{Exactitude} = \frac{VP + VN}{VP + VN + FP + FN}$$

$$\text{Précision} = \frac{VP}{VP + FP}$$

$$\text{Sensibilité (Rappel)} = \frac{VP}{VP + FN}$$

$$\text{Score F1} = \frac{2 \times \text{Précision} \times \text{Sensibilité}}{\text{Précision} + \text{Sensibilité}}$$

3. Affichez les résultats formatés avec **deux décimales**, un par ligne.

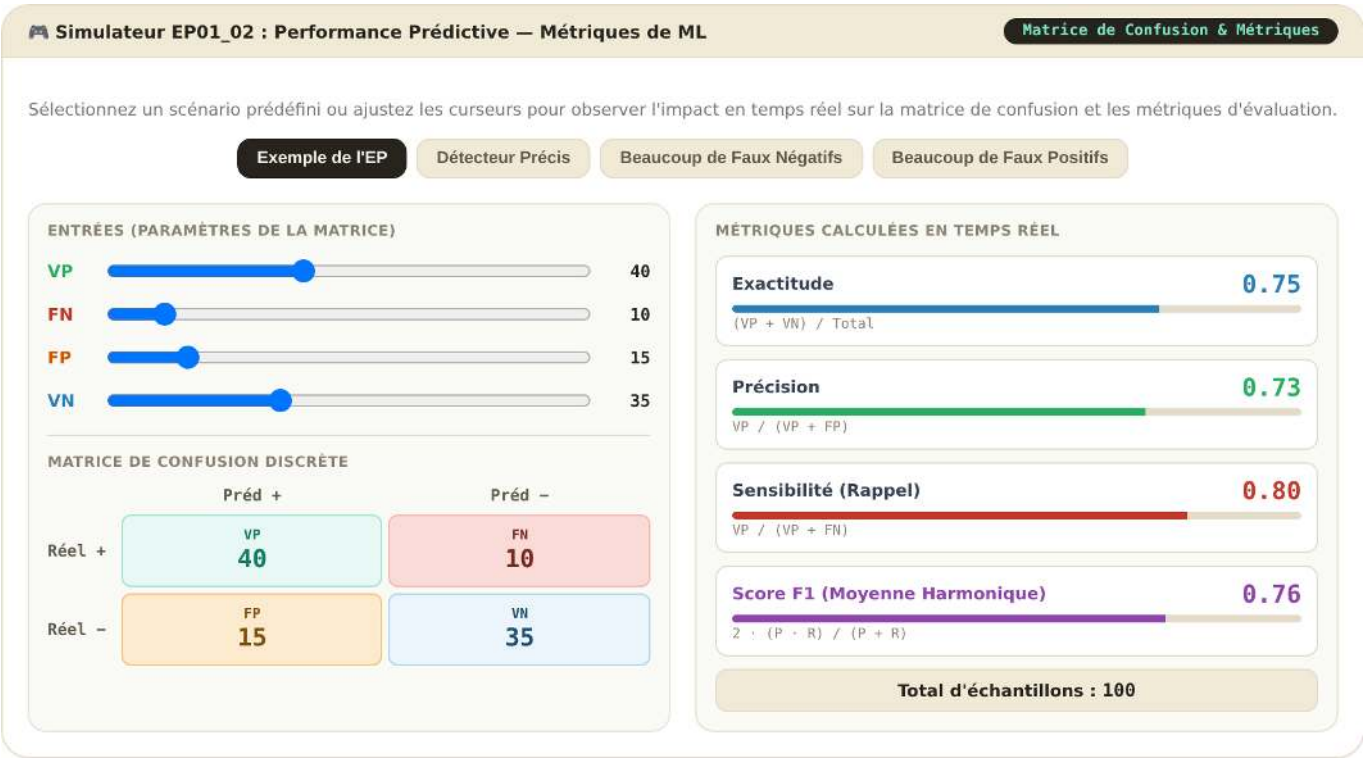
📌 **Important** :

- Utilisez la division en virgule flottante pour éviter des résultats tronqués.
- L'ordre de sortie doit être : **Exactitude**, **Précision**, **Sensibilité** et **Score F1**.
- Consultez la simulation interactive de cet EP sur la Figure 1.12.

1.17.2.3 📌 Exemple d'Exécution

Entrée	Sortie	Observation
40	0.75	Exactitude
10	0.73	Précision
15	0.80	Sensibilité
35	0.76	Score F1

(Note : VP=40, FN=10, FP=15, VN=35. Nombre total de cas = 100)



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.fr/cap01/sim-ep0102.html

Figure 1.12: Simulateur EP01_02 : Performance Prédictive — Métriques de ML

```
%%writefile EP01_02.cpp
// your solution
```

```
Overwriting EP01_02.cpp
```

```
TestSuite("EP01_02.cpp").run()
```

```
<IPython.core.display.HTML object>
```

1.17.3 EP01_03 ↗ *Mean Average Precision* (mAP) — Courbe Précision-Sensibilité

Dans cette activité, vous évaluerez un classificateur binaire (ex. : détection de déforestation dans des images satellites, voir dgi.inpe.br) à travers la **courbe Précision-Sensibilité** et la métrique **mAP (*Mean Average Precision*)**. Le mAP est standard dans des compétitions comme COCO (*Common Objects in Context*) et PASCAL VOC (*Visual Object Classes*) et des modèles YOLO (*You Only Look Once*).

1.17.3.1 🧠 Pourquoi le mAP est-il la métrique-standard ?

Dans l'EP01_02, vous avez vu que le choix du seuil modifie significativement la Précision et la Sensibilité. Le **mAP (*Mean Average Precision*)** résout cela : il évalue le modèle à **plusieurs seuils** (chaque seuil doit générer une matrice de confusion différente) et résume la performance par **l'aire sous la courbe Précision-Sensibilité (P-S)**.

Alors que le F1-*Score* examine un unique point d'équilibre, le mAP considère la courbe entière. Plus il est proche de **1,0**, meilleur est le détecteur à tous les seuils et classes (ex. : pièces de 25, 50 et 1 réal).

Métrique	Ce qu'elle résume	Limitation
F1-*Score*	Équilibre P × S à un seuil unique	Dépend du seuil choisi
AP	Aire sous la courbe P-S d'une clase	Valide uniquement pour une classe
mAP	Moyenne des AP sur toutes les classes	Plus complexe à implémenter

Références : Roboflow — mAP · Vidéo explicative

1.17.3.2 Comment le mAP est-il calculé — pas à pas

1. **Seuils fixes** (utilisez toujours cette liste) :

```
limiares = [0.00, 0.09, 0.21, 0.31, 0.39, 0.52, 0.60, 0.71, 0.81, 0.89, 1.00]
```

2. Pour chaque seuil (t), classez les échantillons : `predito = 1 se confiança  t, senão 0`. Calculez VP, FP, FN, VN et obtenez Précision((t)) et Sensibilité((t)).
3. Construisez la courbe P-S : paires (Sensibilité((t)), Précision((t))), triées par Sensibilité croissante.
4. **Monotonisez la Précision** :

$$P_{\text{mono}}[i] = \max_{j \geq i} P[j]$$

5. Calculez l'**AP** (aire sous la courbe monotone) en utilisant la **règle du trapèze** (approximation plus précise que la simple somme de Riemann) :

$$AP = \sum_{i=1}^{m-1} \frac{P_{\text{mono}}[i-1] + P_{\text{mono}}[i]}{2} \cdot (S[i] - S[i-1])$$

6. **mAP** = moyenne des AP de toutes les classes. Dans cet EP, il n'y a qu'**1 classe**, donc mAP = AP.

> **Note**
>
> 📐 **Différence résumée :**
> La *somme de Riemann* approxime l'aire par des rectangles, pouvant sous-estimer ou surestimer. La **règle du trapèze** utilise des trapèzes, réduisant l'erreur en considérant la moyenne entre les valeurs aux extrémités de l'intervalle, étant généralement plus précise pour des fonctions lisses par morceaux, comme la courbe Précision-Sensibilité.

1.17.3.3 📋 Tâche

Lisez un entier `n` (quantité d'échantillons). Ensuite, lisez `n` lignes, chacune avec : `verdade` (0 ou 1) et `confiança` (float 0.0–1.0).

Calculez et imprimez, pour le **seuil 0.85** (index 9 de la liste) :

- Matrice de Confusion (VP, FN, FP, VN)

- Acurácia, Précision, Sensibilité et F1-Score

Ensuite, pour **tous les seuils**, imprimez :

- Précisions brutes, Précisions monotones et Sensibilités, séparées par ,
- mAP final

1.17.3.4 📌 Important

- Seuil fixe pour les métriques individuelles : **0.85**
- Division sécurisée : si le dénominateur est nul, utilisez 0
- Formatage : deux décimales
- Monotonisez de la fin vers le début
- La Figure 1.13 présente une simulation de cette question

1.17.3.5 📌 Exemple d'Exécution

Entrée	Sortie Attendue
7 0 0.94 1 0.80 1 0.69 0 0.67 1 0.30 1 0.15 1 0.15	# MÉTRIQUES POUR LE SEUIL 0.85 # Matrice de Confusion : VP = 0, FN = 5 FP = 1, VN = 1 Métriques d'Évaluation : Acurácia : 0.14 Précision : 0.00 Sensibilité : 0.00 F1-Score : 0.00 # MÉTRIQUES POUR TOUS LES SEUILS # Précisions : 0.00, 0.00, 0.00, 0.50, 0.50, 0.50, 0.50, 0.50, 0.60, 0.71, 0.71 Précisions mon. : 0.71, 0.71, 0.71, 0.71, 0.71, 0.71, 0.71, 0.71, 0.71, 0.71, 0.71 Sensibilités : 0.00, 0.00, 0.00, 0.20, 0.40, 0.40, 0.40, 0.40, 0.60, 1.00, 1.00 mAP : 0.71

1.17.3.6 🐍 Astuce pour calculer l'AP (avec la règle du trapèze)

```
def calcular_AP(verdades, confiancas, limiares):
    m = len(limiares)
    precisoes = [0.0] * m
    sensibilidades = [0.0] * m
    for i in range(m):
        p, s = calcular_metricas(verdades, confiancas, limiares[i])
        precisoes[m-1-i] = p
        sensibilidades[m-1-i] = s
    prec_mono = precisoes.copy()
    for i in range(m-2, -1, -1):
        if prec_mono[i] < prec_mono[i+1]:
            prec_mono[i] = prec_mono[i+1]
    AP = 0.0
    for i in range(1, m):
        # Règle du trapèze : moyenne des hauteurs multipliée par la base
        area_trapezio = (prec_mono[i-1] + prec_mono[i]) / 2.0
```

```
        AP += area_trapezio * (sensibilidades[i] - sensibilidades[i-1])
    return precisoes, prec_mono, sensibilidades, AP
```

```
%%writefile EP01_03.cpp
// your solution
```

```
Overwriting EP01_03.cpp
```

```
TestSuite("EP01_03.cpp").run()
```

```
<IPython.core.display.HTML object>
```

1.17.4 EP01_04 🖼️ Lecture et Informations d'une Image sous forme de Matrice

Dans cette activité, vous devez écrire un programme qui traite une image numérique représentée comme une matrice de pixels en niveaux de gris.

- Lisez deux entiers **L** et **C**, représentant le nombre de lignes et de colonnes.
- Lisez les **L * C** valeurs entières qui composent la matrice de l'image (chaque valeur entre 0 et 255).
- Calculez et imprimez les informations suivantes :

1. Le nombre de lignes.
2. Le nombre de colonnes.
3. La valeur du pixel le plus grand (**Maximum**).
4. La valeur du pixel le plus petit (**Minimum**).
5. La **Moyenne** arithmétique de tous les pixels.

📌 **Important** :

- La sortie doit suivre exactement le format étiqueté (ex : `Linhas: X`).
- La valeur de la moyenne doit être formatée avec **deux décimales**.
- Voir un simulateur interactif pour cette question dans **Figure 1.14** (grille interactive pour visualiser les intensités et effectuer les calculs en temps réel).

1.17.4.1 🧠 Pourquoi cela importe-t-il ? – L'Image comme Données

Toute image numérique est, au fond, une structure de données. En niveaux de gris 8 bits, chaque pixel est une valeur scalaire. Extraire des statistiques de base est la première étape pour :

Opération	Utilité Pratique
Maximum/Minimum	Identifier si l'image est « délavée » (faible contraste) ou saturée.
Moyenne	Calculer la luminosité globale de la scène pour des ajustements d'exposition.
Normalisation	Redimensionner les valeurs vers des intervalles comme $[0, 1]$ dans les réseaux de neurones.

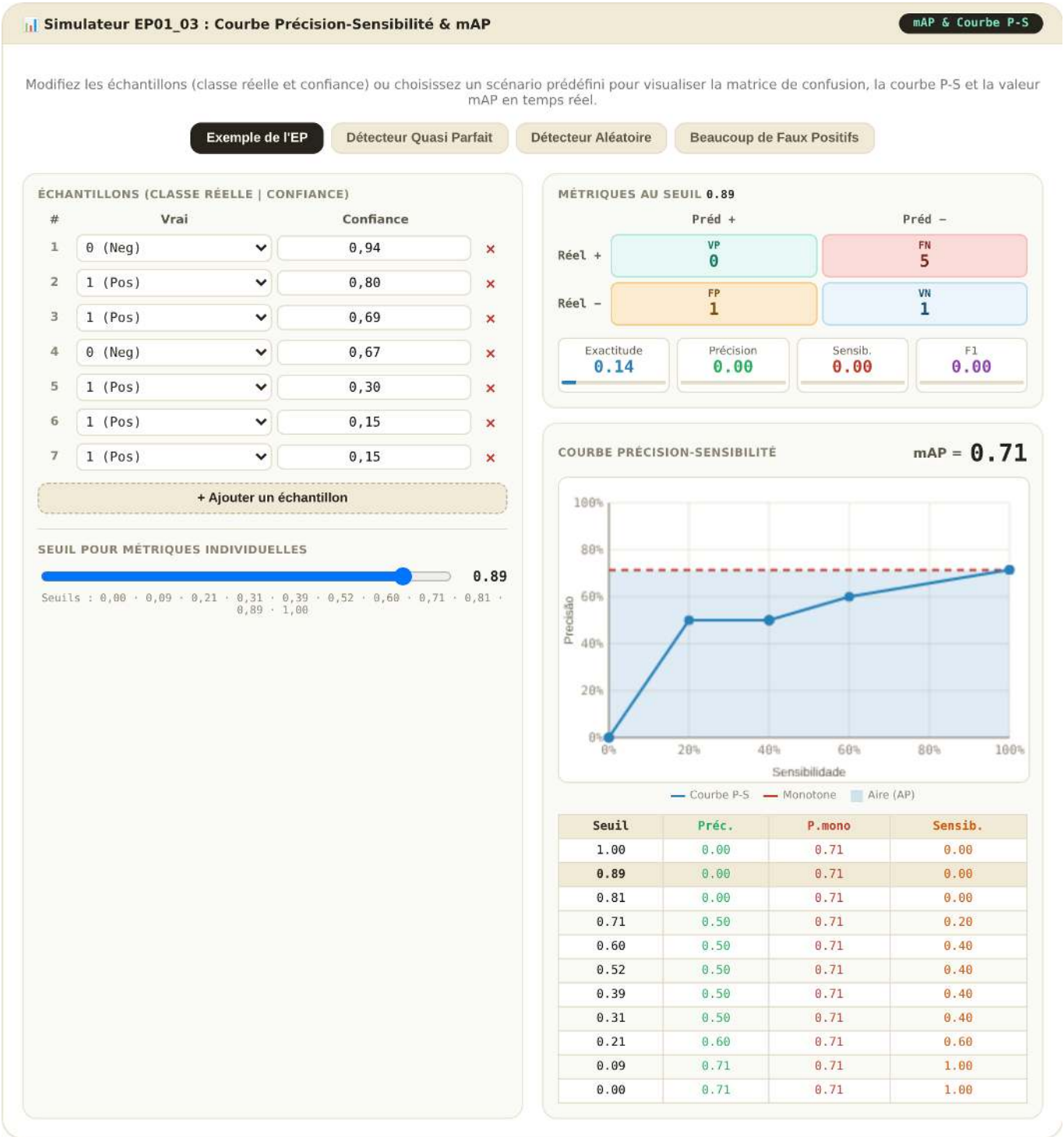
1.17.4.2 📋 Tâche (spécification pour VPL)

Entrée :

La première ligne contient l'entier `L` (lignes).

La deuxième ligne contient l'entier `C` (colonnes).

Les lignes suivantes contiennent les éléments de la matrice.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.fr/cap01/sim-ep0103-map.html

Figure 1.13: Simulateur EP01_03 : Mean Average Precision (mAP) et courbe P-S

Sortie :

Cinq lignes formatées selon l'exemple :

`Linhas: L`

`Colunas: C`

`Max: V`

`Min: V`

`Media: V.VV`

1.17.4.3 📌 Exemples

Entrée	Sortie	Observation
2 3 0 128 255 50 100 200	Linhas: 2 Colunas: 3 Max: 255 Min: 0 Media: 122.17	Petite image à fort contraste

Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.fr/cap01/sim-ep0104-estatisticas.html

Figure 1.14: Simulateur EP01_04 : Statistiques de pixels dans une matrice discrète 5x5

```
%%writefile EP01_04.cpp
// your solution
```

```
Overwriting EP01_04.cpp
```

```
TestSuite("EP01_04.cpp").run()
```

```
<IPython.core.display.HTML object>
```

1.17.5 EP01_05 🔄 Négatif d'une Image en Niveaux de Gris

Dans cette activité, il faut écrire un programme qui calcule le négatif d'une image numérique.

- Lire deux entiers **L** et **C**, représentant le nombre de lignes et de colonnes.
- Lire les valeurs entières qui composent la matrice de l'image.
- Pour chaque pixel, appliquer la transformation d'inversion :

$$pixel_{ngatif} = 255 - pixel_{original}$$

- Afficher la matrice résultante, en conservant le format original (L lignes et C colonnes).

📌 **Important** :

- Les valeurs de chaque ligne dans la sortie doivent être séparées par un **espace blanc**.
- La sortie doit contenir **uniquement les nombres** de la matrice résultante.
- Voir un simulateur interactif pour cette question à la **Figure 1.15** (comparaison en temps réel entre la matrice originale et son négatif).

1.17.5.1 🧠 Pourquoi cela importe-t-il ? – Inversion d'Intensité

Le **négatif** est une transformation linéaire de base qui inverse l'échelle de luminosité. C'est un outil essentiel pour que l'œil humain identifie les détails clairs qui sont « cachés » dans des fonds plus sombres, et il est largement utilisé dans :

Application	Utilité
Imagerie Médicale	Améliore la visualisation des anomalies dans les tissus denses (ex : Rayons X).
Astronomie	Mettre en évidence les galaxies et les nébuleuses ténues contre le vide de l'espace.
Arts Numériques	Effets esthétiques et préparation de masques de sélection.

1.17.5.2 📋 Tâche (spécification pour VPL)

Entrée :

La première ligne contient l'entier `L`.

La deuxième ligne contient l'entier `C`.

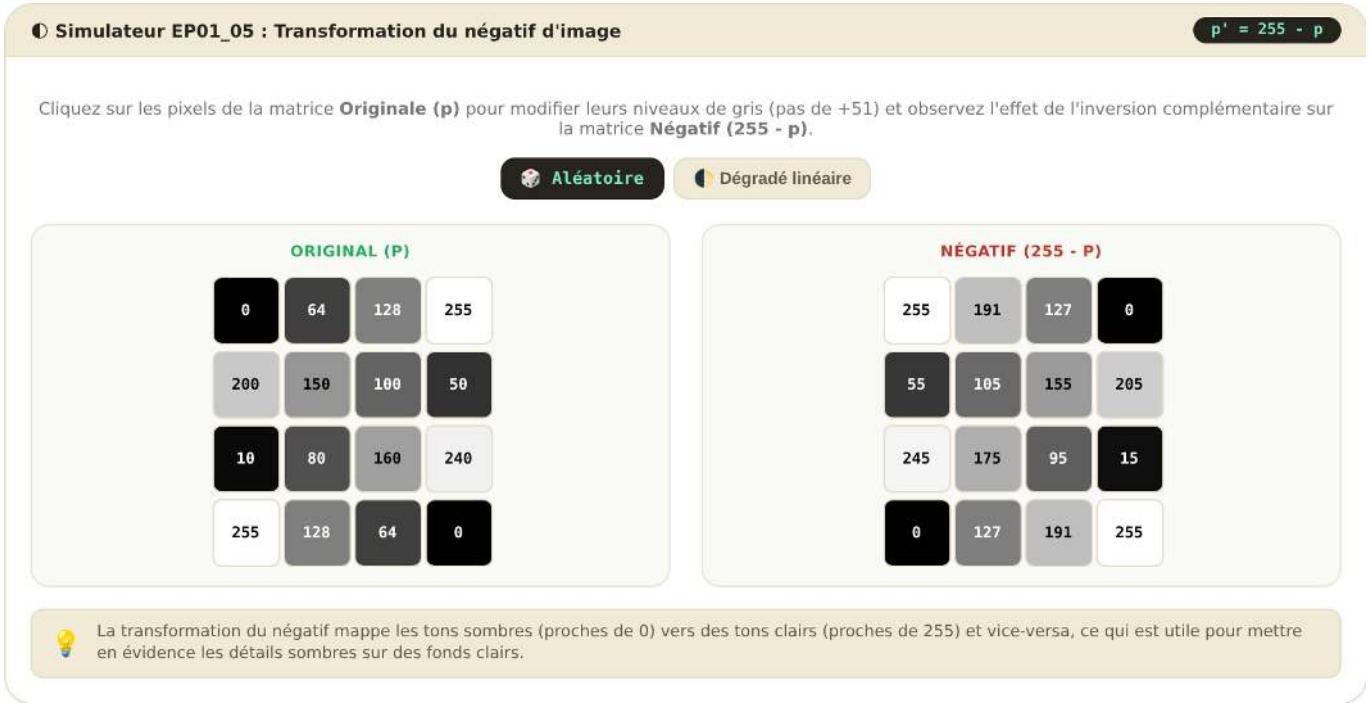
Les lignes suivantes contiennent les éléments de la matrice.

Sortie :

La matrice inversée avec `L` lignes et `C` colonnes.

1.17.5.3 📌 Exemples

Entrée	Sortie	Remarque
2 3 0 128 255 50 100 255	255 127 0 205 155 55	Là où il y avait 0 (noir), devient 255 (blanc)



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.fr/cap01/sim-ep0105-negativo.html

Figure 1.15: Simulateur EP01_05 : Transformation de Négatif d'Image (Inversion d'Intensité)

```
%%writefile EP01_05.cpp
// your solution
```

```
Overwriting EP01_05.cpp
```

```
TestSuite("EP01_05.cpp").run()
```

```
<IPython.core.display.HTML object>
```

1.17.6 EP01_06 🎨 Conversion RGB → Niveaux de gris (ITU-R BT.601)

Dans cette activité, vous devez écrire un programme qui convertit des pixels colorés (RVB) en niveaux de gris en utilisant la pondération physiologique de la norme ITU-R BT.601.

- Lisez deux entiers **L** et **C**, représentant le nombre de lignes et de colonnes.
- Lisez **L** × **C** triplets d'entiers, où chaque triplet représente les canaux **R** (Rouge), **G** (Vert) et **B** (Bleu) d'un pixel.
- Pour chaque pixel, calculez la valeur de gris (g) à l'aide de la formule :

$$g = \text{round}(0.299 \times R + 0.587 \times G + 0.114 \times B)$$

- Affichez la matrice résultante (L lignes et C colonnes) contenant les valeurs entières converties.

📌 **Important** :

- Utilisez la fonction `round()` de votre langage pour garantir l'arrondi correct à l'entier le plus proche.
- La sortie doit contenir uniquement les valeurs de gris, en préservant la structure de la matrice (séparées par des espaces sur la ligne).
- Consultez un simulateur interactif pour cette question sur la **Figure 1.16** (ajustez les curseurs pour voir comment chaque couleur contribue à la luminosité finale).

1.17.6.1 🧠 Pourquoi ne pas utiliser simplement la moyenne ?

L'œil humain ne perçoit pas toutes les couleurs avec la même intensité. Nous sommes beaucoup plus sensibles au **Vert** qu'au **Bleu** en raison de notre évolution biologique. La norme ITU-R BT.601 utilise des poids spécifiques pour créer une image en niveaux de gris qui semble naturellement correcte pour notre vision :

Canal	Poids	Perception humaine
🟢 **Vert**	**58,7 %**	Sensibilité maximale (distinction du feuillage).
🔴 **Rouge**	**29,9 %**	Sensibilité moyenne.
🔵 **Bleu**	**11,4 %**	Faible sensibilité (tons plus sombres).

1.17.6.2 📋 Tâche (spécification pour VPL)

Entrée :

La première ligne contient l'entier `L`.

La deuxième ligne contient l'entier `C`.

Les lignes suivantes contiennent des triplets d'entiers `R G B` pour chaque pixel.

Sortie :

La matrice des niveaux de gris avec `L` lignes et `C` colonnes.

1.17.6.3 📌 Exemples

Entrée	Sortie	Observation
1 3 255 0 0 0 255 0 0 0 255	76 150 29	Remarquez comment le Vert (150) est plus lumineux que le Bleu (29)

Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.fr/cap01/sim-ep0106-rgb-gray.html

Figure 1.16: Simulador EP01_06 : Conversion RVB en niveaux de gris (pondération perceptuelle UIT-R BT.601)

```
%%writefile EP01_06.cpp
// your solution
```

```
Overwriting EP01_06.cpp
```

```
TestSuite("EP01_06.cpp").run()
```

```
<IPython.core.display.HTML object>
```

1.17.7 EP01_07 ⚫ Seuillage Manuel : Image Binaire

Dans cette activité, vous devez écrire un programme qui réalise la segmentation d'une image par seuillage (*thresholding*).

- Lisez deux entiers **L** et **C**, représentant les dimensions de la matrice.
- Lisez un entier **T**, qui sera la valeur du seuil (coupure).
- Lisez les valeurs entières de la matrice.
- Pour chaque pixel p, appliquez la règle de binarisation suivante :

$$\text{résultat} = \begin{cases} 255 & \text{si } p > T \\ 0 & \text{si } p \leq T \end{cases}$$

- Affichez la matrice résultante contenant uniquement les valeurs 0 ou 255.

📌 **Important** :

- Faites attention à l'opérateur : le pixel ne devient blanc (255) que s'il est **strictement supérieur** à T.
- La sortie doit conserver la structure matricielle (L lignes et C colonnes).
- Voir un simulateur interactif pour cette question sur la **Figure 1.17** (ajustez le curseur de T pour observer comment les objets sont isolés du fond).

1.17.7.1 🧠 Qu'est-ce que la Segmentation ?

Le seuillage est la méthode la plus simple pour séparer les objets d'intérêt du fond de l'image. En transformant les niveaux de gris en noir et blanc pur, on crée une carte binaire qui facilite le comptage des objets ou l'identification des formes :

Valeur du Pixel (p)	Condition	Résultat Final
Sombre ($p \leq T$)	Fond/Bruit	**0** (Noir)
Clair ($p > T$)	Objet/Mise en évidence	**255** (Blanc)

1.17.7.2 📋 Tâche (spécification pour VPL)

Entrée :

La première ligne contient l'entier `L`.

La deuxième ligne contient l'entier `C`.

La troisième ligne contient l'entier `T` (seuil).

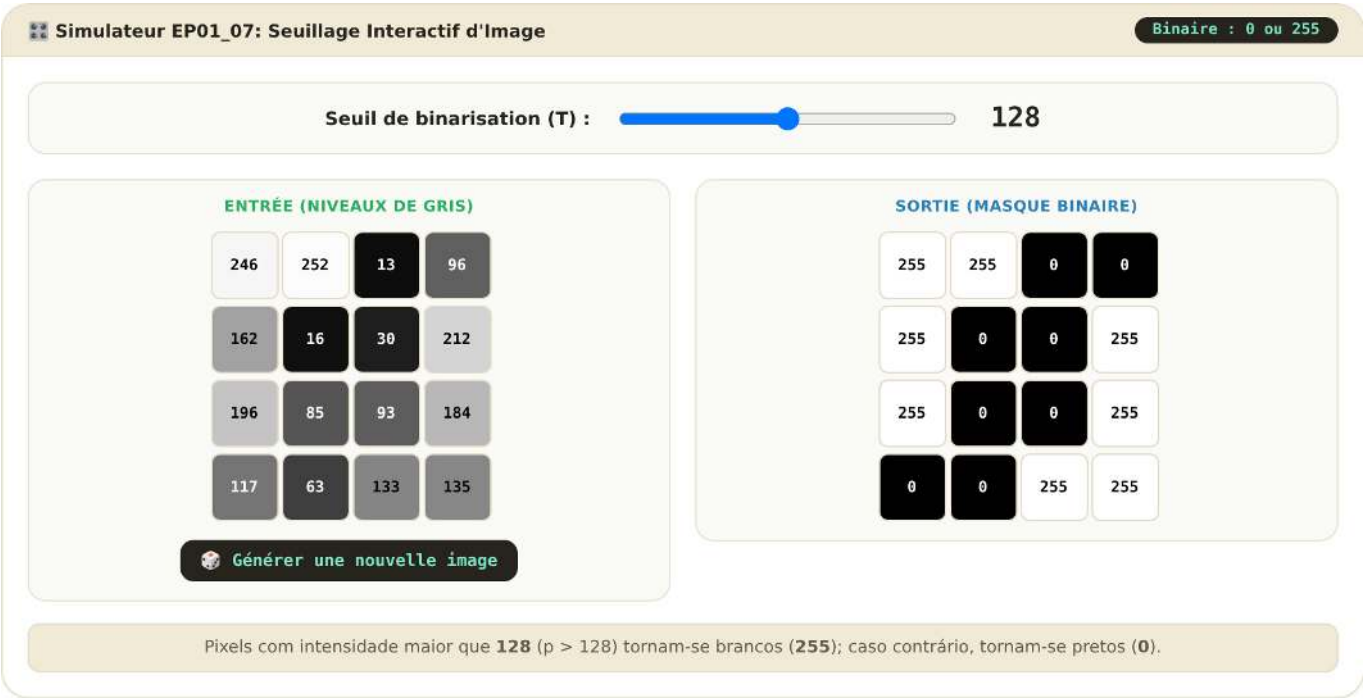
Les lignes suivantes contiennent les éléments de la matrice.

Sortie :

La matrice binarisée (0 ou 255) avec `L` lignes et `C` colonnes.

1.17.7.3 📌 Exemples

Entrée	Sortie	Observation
2 4 128 0 100 128 200 50 129 255 64	0 0 0 255 0 255 255 0	Notez que la valeur 128 est devenue 0 (car $128 \leq 128$)



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.fr/cap01/sim-ep0107-limiarizacao.html

Figure 1.17: Simulateur EP01_07 : Seuillage global interactif (binarisation p > T)

```
%%writefile EP01_07.cpp
// your solution
```

```
Overwriting EP01_07.cpp
```

```
TestSuite("EP01_07.cpp").run()
```

```
<IPython.core.display.HTML object>
```

1.17.8 EP01_08 🎨 Remappage par Plage d'Intensité

Dans cette activité, vous devez écrire un programme qui applique des transformations linéaires distinctes à différentes régions d'intensité de l'image.

- Lisez deux entiers **L** et **C**, représentant les dimensions de la matrice.
- Lisez les entiers **T** (seuil), (delta 1) et (delta 2).
- Lisez les valeurs entières de la matrice.
- Pour chaque pixel p, appliquez la règle de remappage conditionnel :

$$\text{résultat} = \begin{cases} p + \delta_1 & \text{si } p < T \\ p + \delta_2 & \text{si } p \geq T \end{cases}$$

- Affichez la matrice résultante avec les nouvelles valeurs d'intensité.

📌 **Important** :

- est le décalage appliqué aux pixels sombres (en dessous du seuil).
- est le décalage appliqué aux pixels clairs (supérieurs ou égaux au seuil).
- Les cas de test garantissent que le résultat sera toujours dans l'intervalle valide de **0 à 255**, il n'est donc pas nécessaire de gérer la saturation ou les arrondis.
- La sortie doit conserver la structure de la matrice (**L** lignes et **C** colonnes).

1.17.8.1 🧠 Transformation Conditionnelle des Pixels

En Traitement Numérique des Images (TNI), nous devons souvent traiter les régions de manière indépendante. Cette technique permet, par exemple, d'éclaircir uniquement les ombres d'une photographie (en augmentant les pixels sombres) sans saturer la luminosité des zones déjà claires, ou vice versa.

Plage d'Intensité	Condition	Opération
Pixels Sombres	$p < T$	$p + \delta_1$
Pixels Clairs	$p \geq T$	$p + \delta_2$

1.17.8.2 📋 Tâche (spécification pour VPL)

Entrée :

La première ligne contient les entiers `L` et `C`.

La deuxième ligne contient les entiers `T`, et .

Les lignes suivantes contiennent les éléments de la matrice.

Sortie :

La matrice transformée avec `L` lignes et `C` colonnes, avec des valeurs séparées par des espaces.

1.17.8.3 📌 Exemples

Entrée	Sortie	Observation

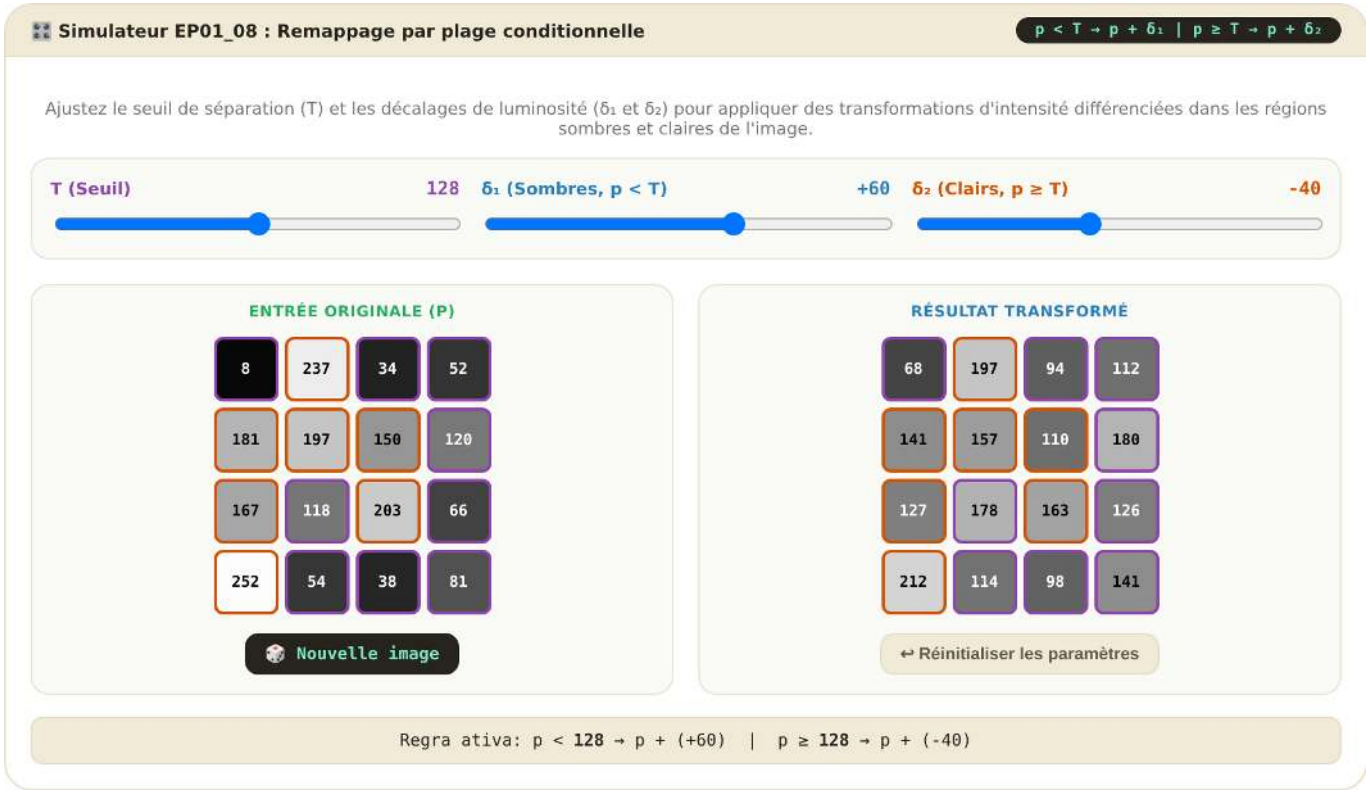
`2 4`
`128 60 -40`
`0 100 150 255`
`80 128 200 30`
	`60 160 110 215`
	`140 88 160 90`
		Les pixels < 128 ajoutent 60. Les pixels 128 soustraient 40.

```
%%writefile EP01_08.cpp
// your solution
```

```
Overwriting EP01_08.cpp
```

```
TestSuite("EP01_08.cpp").run()
```

```
<IPython.core.display.HTML object>
```



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.fr/cap01/sim-ep0108-brilho-contraste.html

Figure 1.18: Simulateur EP01_08 : Remappage par Plage Conditionnelle (Luminosité et Contraste par Seuil)

1.17.9 EP01_09 🏁 Modèle de Damier : Matrice d'Échecs

Dans cette activité, vous devez écrire un programme qui génère une image synthétique selon le motif d'un damier.

- Lisez deux entiers **L** (lignes) et **C** (colonnes).
- Générez une matrice où les valeurs alternent entre **0** (noir) et **1** (blanc).
- La logique de remplissage doit suivre la règle de parité :
- L'élément à la position $(0, 0)$ est toujours **0**.
- Un pixel à la position (i, j) sera **1** si la somme des indices $(i + j)$ est **impaire**.
- Un pixel à la position (i, j) sera **0** si la somme des indices $(i + j)$ est **paire**.

📌 **Important** :

- Les couleurs doivent alterner correctement aussi bien horizontalement que verticalement.
- La sortie doit être la matrice imprimée ligne par ligne, avec les éléments séparés par un espace.
- Voir un simulateur interactif pour cette question sur la **Figure 1.19** (ajustez les dimensions pour visualiser la construction de la grille et la sortie textuelle correspondante).

1.17.9.1 🧠 Motifs Synthétiques

Créer des motifs géométriques est un exercice fondamental pour maîtriser la **logique des indices** dans les matrices. En Traitement Numérique des Images, le motif du damier n'est pas seulement esthétique ; il est largement utilisé pour :

Application	Utilité
Calibrage de Caméra	Estimer les paramètres intrinsèques et extrinsèques de l'objectif.
Correction de Distorsion	Identifier et corriger l'effet de "barillet" ou "coussinet" sur les objectifs grand-angle.

Application	Utilité
Cartographie 3D	Projeter des motifs connus pour reconstruire des surfaces dans les systèmes de lumière structurée.

1.17.9.2 📋 Tâche (spécification pour VPL)

Entrée :

Une ligne contenant l'entier `L` (lignes).

Une ligne contenant l'entier `C` (colonnes).

Sortie :

La matrice d'échecs avec `L` lignes et `C` colonnes, imprimée avec des espaces entre les éléments.

1.17.9.3 📌 Exemples

Entrée	Sortie	Observation
3 4	0 1 0 1 1 0 1 0 0 1 0 1	Notez que chaque ligne commence par l'inverse de la précédente

Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.fr/cap01/sim-ep0109-xadrez.html

Figure 1.19: Simulateur EP01_09 : Générateur de motif en damier (logique de parité (i + j) % 2)

```
%%writefile EP01_09.cpp
// your solution
```

```
Overwriting EP01_09.cpp
```

```
TestSuite("EP01_09.cpp").run()
```

```
<IPython.core.display.HTML object>
```

1.17.10 EP01_10 📄 Métadonnées : Lecture de fichier PGM

Dans cette activité, vous devez lire un fichier image au format **PGM (Portable Gray Map)** et extraire ses dimensions à partir de l'en-tête.

- Le format **PGM (P2)** est un fichier texte simple (ASCII) qui stocke des images en niveaux de gris.
- Le fichier possède un **en-tête** structuré de la manière suivante :

1. **Version :** L'identifiant `P2`.
2. **Commentaires :** Lignes optionnelles commençant par `#` (à ignorer).
3. **Dimensions :** Deux entiers représentant la **Largeur** et la **Hauteur**.
4. **Maximum :** Un entier représentant l'intensité maximale (généralement 255).

- Après l'en-tête, suivent les données des pixels.

📌 **Important** :

- **Lecture du fichier :** Vous devez ouvrir le fichier indiqué dans l'exemple en utilisant la fonction `open()` de Python.
- **Ordre de sortie :** Contrairement à l'ordre présent dans le fichier, la sortie attendue doit être au format de tuple : `(Hauteur, Largeur, Canaux)`.
- Comme les fichiers PGM sont en niveaux de gris, le nombre de **Canaux** est toujours **1**.
- Consultez un simulateur interactif pour cette question sur **Figure 1.20** (ajustez les dimensions pour voir comment l'en-tête ASCII est généré).

1.17.10.1 🧠 Comprendre le format PGM

Le format PGM est l'un des plus simples pour le traitement d'images. Étant en texte pur, il permet de visualiser les métadonnées et même les valeurs des pixels en ouvrant le fichier dans un bloc-notes :

Composant	Exemple	Signification
Nombre magique	`P2`	Identifie qu'il s'agit d'un PGM au format texte (ASCII).
Commentaire	`# CREATOR...`	Ligne informative ignorée par le processeur.
Dimensions	`397 343`	**397** colonnes (Largeur) et **343** lignes (Hauteur).
Intensité	`255`	Définit la valeur du blanc pur (échelle de 0 à 255).

1.17.10.2 📋 Tâche (spécification pour VPL)

Entrée :

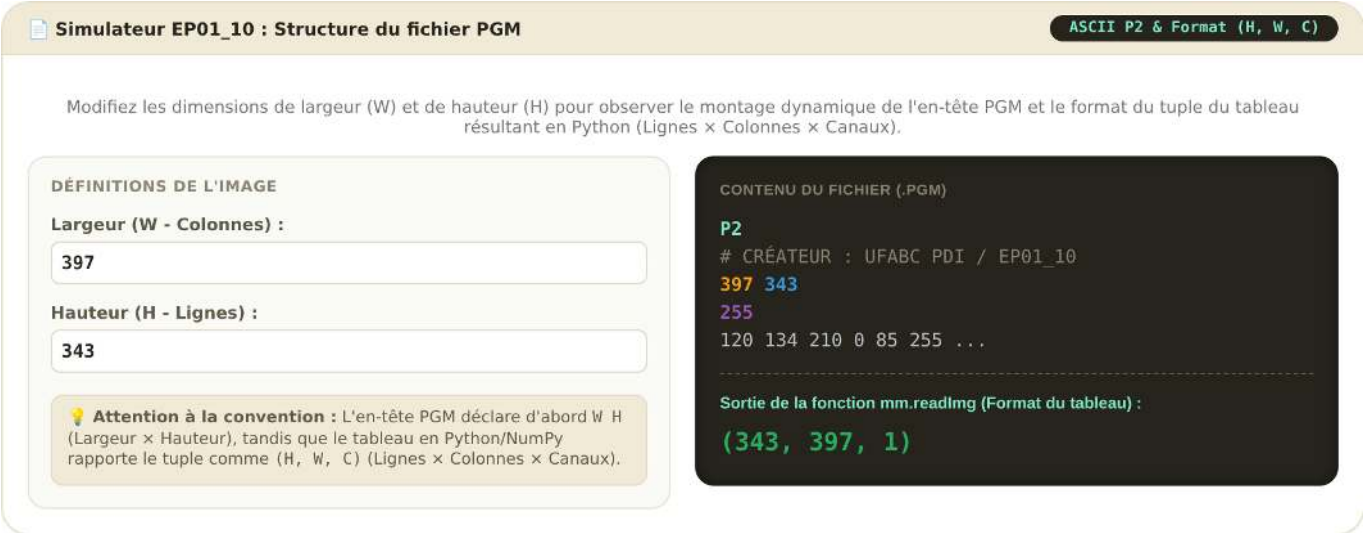
Aucune entrée via le clavier. Le programme doit lire le fichier `"aula01fig03b.pgm"` présent dans le répertoire d'exécution.

Sortie :

Un tuple contenant `(Hauteur, Largeur, 1)`.

1.17.10.3 📌 Exemples

Nom du fichier	Sortie attendue	Remarque
"aula01fig03b.pgm"	(343, 397, 1)	Notez l'inversion de l'ordre : la Hauteur en premier



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.fr/cap01/sim-ep0110-pgm.html

Figure 1.20: Simulateur EP01_10 : Structure du fichier PGM (En-tête ASCII P2 et mappage vers un tuple Python)

> **Note**
>
> Le fichier nécessaire pour ce TP sera téléchargé automatiquement depuis le dépôt via le code suivant :

```
%%writefile tmp/mm_out_2.cpp
#include <iostream>
#include <string>
#include <fstream>
#include <cstdlib>

//URL de base pour les images
const std::string BASE_URL =
"https://raw.githubusercontent.com/fzampirolli/pdi-vc/master/all/cap01/imagens";
const std::string file = "aula01fig03b.pgm";

// Fonction pour télécharger un fichier (simplifié)
bool downloadFile(const std::string& url, const std::string& filename) {
    std::string command = "curl -s -o " + filename + " " + url;
    int result = std::system(command.c_str());
    if (result == 0 && std::ifstream(filename).good()) {
        std::cout << "  Fichier téléchargé: " << filename << std::endl;
        return true;
    }
    std::cout << "  Erreur: Non trouvé" << std::endl;
    return false;
}

int main() {
    // Vérifier si le fichier existe déjà
    std::ifstream fileCheck(file);
    if (!fileCheck.good()) {
        std::string url = BASE_URL + "/" + file;
        downloadFile(url, file);
    }
    return 0;
}
```

```
Overwriting tmp/mm_out_2.cpp
```

```
!g++ -I. -std=c++17 tmp/mm_out_2.cpp -o tmp/mm_out_2 \
  && ./tmp/mm_out_2
```

```
%%writefile EP01_10.cpp
// your solution
```

```
Overwriting EP01_10.cpp
```

```
TestSuite("EP01_10.cpp").run()
```

```
<IPython.core.display.HTML object>
```

1.17.11 EP01_11 ↗ Analyse de voisinage : Filtre de maximum 1D

Dans cette activité, vous devez implémenter un filtre morphologique simple de maximum opérant sur un signal unidimensionnel (vecteur).

- Lisez un entier **n**, représentant la taille du vecteur.
- Lisez les **n** éléments entiers qui composent le vecteur original **v1**.
- Créez un nouveau vecteur **v2**, où chaque position i est le résultat de la comparaison entre l'élément courant et ses voisins immédiats :

$$v2[i] = \max(v1[i-1],\ v1[i],\ v1[i+1])$$

📌 **Important** :

- **Bords :** Aux extrémités du vecteur (indices 0 et $n-1$), le voisinage ne contient que deux éléments (l'élément lui-même et le seul voisin disponible). À l'indice 0, comparez uniquement $v1[0]$ et $v1[1]$. Au dernier indice, comparez uniquement $v1[n-2]$ et $v1[n-1]$.
- **Sortie :** Affichez l'en-tête « v2 : » suivi des valeurs du vecteur résultant, une par ligne.
- Consultez un simulateur interactif pour cette question dans la **Figure 1.21** (survolez les résultats avec la souris pour visualiser la fenêtre de voisinage utilisée dans le calcul).

1.17.11.1 🧠 Pourquoi analyser les voisins ?

En traitement d'images, la valeur d'un pixel est rarement isolée ; elle dépend du contexte qui l'entoure. Le **Filtre de Maximum** est la base de l'opération de **Dilatation** en morphologie mathématique, servant à :

Fonction	Effet visuel
Rehaussement	Étend les structures brillantes et « épaissit » les objets clairs.
Suppression du bruit	Élimine les petits points noirs (bruit « sel et poivre » sombre).
Remplissage	Ferme les petits trous ou lacunes dans les formes binaires.

1.17.11.2 📋 Tâche (spécification pour VPL)

Entrée :

Un entier `n`.

Sur les lignes suivantes, les `n` éléments entiers du vecteur.

Sortie :

La *chaîne* `v2 :` sur la première ligne.

Sur les lignes suivantes, chaque élément de `v2` (un par ligne).

1.17.11.3 📌 Exemples

Entrée	Sortie	Observation
5	v2 :	À l'indice 1 : max(10, 20, 5) = 20
10	20	
20	20	
5	30	
30	30	
15	30	

Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.fr/cap01/sim-ep0111-filtro-maximo.html

Figure 1.21: Simulateur EP01_11 : Filtre de Maximum Local 1D (Voisinage 1x3 avec Condition de Bord)

```
%%writefile EP01_11.cpp
// your solution
```

```
Overwriting EP01_11.cpp
```

```
TestSuite("EP01_11.cpp").run()
```

```
<IPython.core.display.HTML object>
```

Chapitre 2

De la Capture au Pixel - Échantillonnage, Quantification et Connectivité

Exécuter Colab Ouvrir GitHub

Ce chapitre approfondit la compréhension de l'image numérique, passant de la nature physique de la capture à sa représentation mathématique discrète. Nous explorons comment la lumière devient donnée et comment l'organisation spatiale des pixels définit les relations de voisinage et de connectivité essentielles pour les algorithmes avancés de Vision par Ordinateur (VO).

2.1 Objectifs

À la fin de ce chapitre, vous serez capable de :

- Expliquer le modèle physique de formation de l'image basé sur l'éclairage et la réflectance.
- Différencier les mécanismes de la vision humaine et ceux des capteurs numériques.
- Comprendre les processus d'**échantillonnage** (discrétisation de l'espace) et de **quantification** (discrétisation de l'intensité).
- Décrire les relations topologiques entre pixels : voisinage, adjacence, connectivité et distances.
- Effectuer des transformations géométriques de base (translation, rotation, échelle) en préservant la qualité.
- Visualiser en pratique les effets de la variation de la résolution spatiale et de la profondeur de bits.

2.2 L'Œil et la Caméra - Éléments de la Perception Visuelle

La formation d'une image numérique commence par la capture de la lumière réfléchie par les objets. Comme illustré dans Figure 2.1, tant l'œil humain que les caméras numériques suivent des principes optiques similaires pour focaliser la lumière sur une surface sensible, bien qu'ils utilisent des mécanismes biologiques et électroniques distincts pour la transduction du signal.

2.2.1 Vision humaine

L'œil fonctionne comme un système optique complexe : la lumière traverse la cornée, l'humeur aqueuse, la pupille (contrôlée par l'iris) et le cristallin — qui ajuste la mise au point dynamiquement — jusqu'à atteindre la rétine. Sur la rétine se trouvent les photorécepteurs : les **cônes** (6 millions), concentrés dans la fovéa, sont responsables de la vision des couleurs et des détails, tandis que les **bâtonnets** (120 millions) assurent la vision en faible luminosité (vision scotopique), ne détectant que des intensités de gris. Le point aveugle est la région d'où part le nerf optique, dépourvue de récepteurs.

2.2.2 Capteurs numériques

Dans les caméras, les capteurs d'image jouent le rôle de la rétine. Les deux types les plus courants sont le **CCD** (*Charge-Coupled Device* - Dispositif à Transfert de Charge) et le **CMOS** (*Complementary Metal-*

Oxide-Semiconductor - Semi-conducteur à Oxyde Métallique Complémentaire). Le capteur est composé d'une matrice de photodiodes (pixels) qui accumulent une charge électrique proportionnelle à la lumière incidente.

Pour la reconstruction des couleurs, on utilise le **Filtre de Bayer**, une matrice de filtres colorés qui permet à chaque pixel de capturer uniquement une composante de couleur : rouge, vert ou bleu (RGGB - *Red, Green, Green, Blue*). Par la suite, un **ADC** (*Analog-to-Digital Converter* - Convertisseur Analogique-Numérique) quantifie cette charge en valeurs numériques, définies par une profondeur de bits (ex. : 8 bits, résultant en 256 niveaux d'intensité).

> **Curiosité**
>
> Bien que l'œil humain possède des millions de récepteurs, la résolution en haute définition est limitée à la fovéa (vision centrale), équivalente à environ 120×120 pixels. La perception d'une scène complète en haute résolution résulte d'un intense post-traitement effectué par le cerveau.

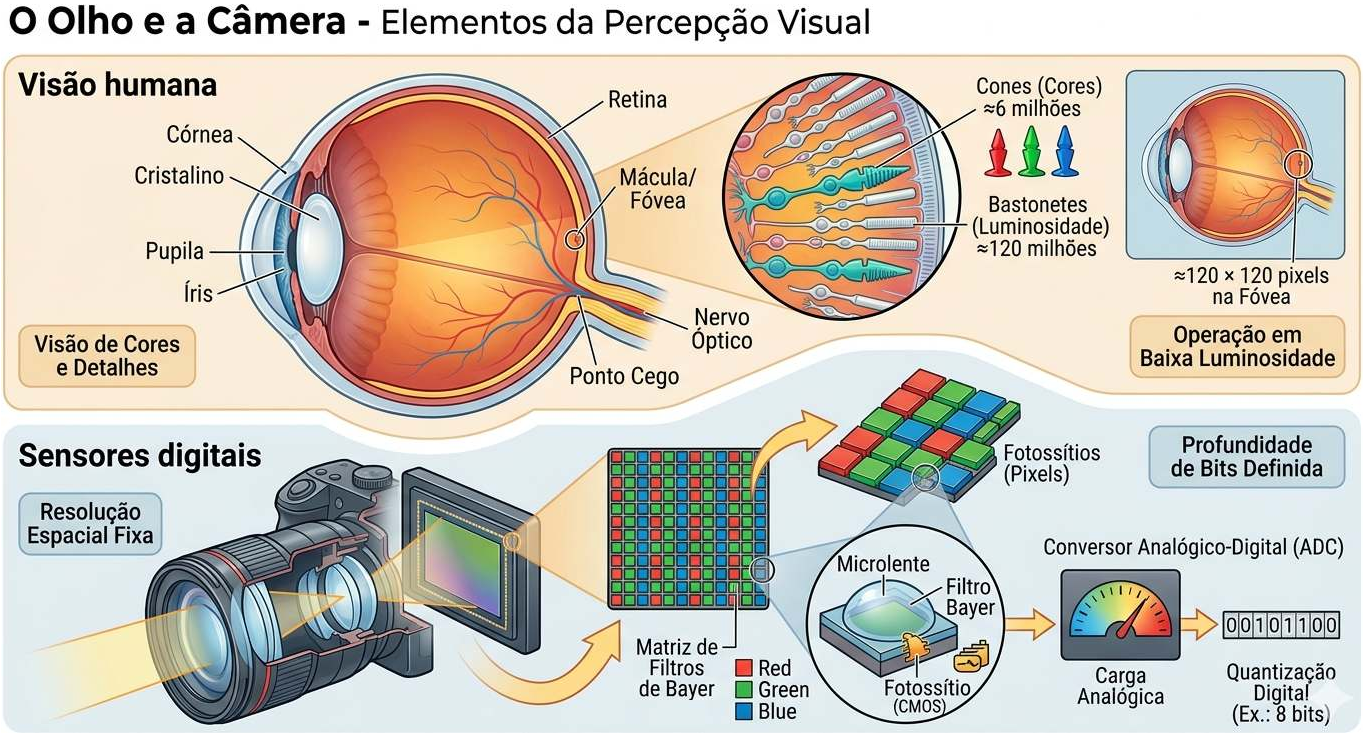


Figure 2.1: Comparaison didactique entre le système visuel biologique et le système électronique : en haut, l'anatomie de l'œil humain mettant en évidence la rétine et les photorécepteurs (cônes et bâtonnets) ; en bas, la structure d'un appareil photo numérique détaillant le capteur CMOS, la matrice de filtres de Bayer (RGGB) et le processus de quantification numérique réalisé par le convertisseur analogique-numérique (ADC).

2.3 Illusions d'optique : Les défis de la perception visuelle

Alors que les capteurs numériques capturent l'intensité de la lumière de manière linéaire et objective, le système visuel humain interprète la scène en se fondant sur le contexte, les expériences antérieures et les mécanismes biologiques de survie. Les illusions d'optique ne sont pas des « erreurs » de l'œil, mais des preuves de l'intense **post-traitement cérébral** réalisé dans le cortex visuel.

2.3.1 Ambiguïté et contexte

Le cerveau cherche constamment à donner un sens aux motifs ambigus. Dans l'exemple du **Vase de Rubin** (voir Figure 2.2), la perception alterne entre la figure (le vase) et le fond (deux visages), démontrant que nous ne pouvons pas traiter simultanément les deux interprétations. Quant à l'illusion de l'**Éléphant de**

Shepard, elle joue avec notre incapacité à concilier les lignes de contour qui suggèrent un volume dans des positions logiquement impossibles.

2.3.2 Luminosité et contraste local

De nombreuses illusions découlent de l'**inhibition latérale**, mécanisme par lequel les neurones voisins de la rétine se font concurrence pour accentuer les contours. Dans la **Grille scintillante**, des points sombres « fantômes » semblent apparaître aux intersections blanches en raison de ce traitement local du contraste.

L'illusion de l'**Ombre sur l'échiquier d'Adelson** est peut-être la plus frappante pour la VC : le carré « A » et le carré « B » ont exactement la même valeur de gris sur le capteur (ou dans le fichier numérique), mais le cerveau « corrige » la luminosité du carré « B » en comprenant qu'il se trouve sous une ombre portée, le percevant comme plus clair.

2.3.3 Géométrie et perspective

La perception de la profondeur peut être trompée par des constructions géométriques qui défient la logique tridimensionnelle à partir d'un angle de vue spécifique. L'**Escalier de Schröder** utilise l'ambiguïté de la perspective pour créer un objet qui semble monter ou descendre selon la façon dont on l'observe, mettant en évidence comment notre interprétation du « haut » et du « bas » dépend du point de fuite.

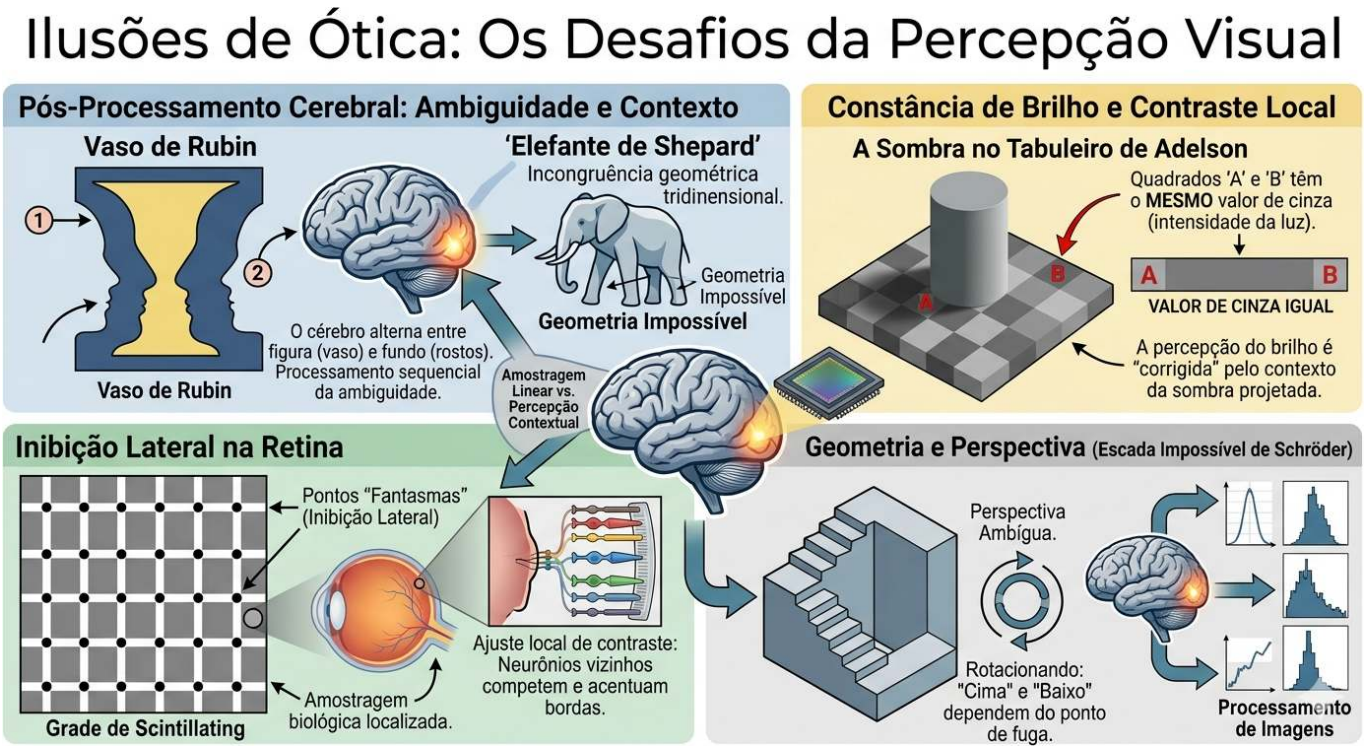


Figure 2.2: Recueil de défis perceptifs : (en haut à gauche) Vase de Rubin — ambiguïté figure-fond ; (en haut au centre) Éléphant de Shepard — incongruence géométrique ; (en haut à droite) Ombre d'Adelson — constance de la luminosité fondée sur le contexte ; (en bas à gauche) Grille de points — inhibition latérale ; (en bas à droite) Escalier de Schröder.

2.4 Le Modèle Mathématique de la Formation de l'Image

Une image peut être modélisée comme le produit de deux fonctions :

$$f(x, y) = i(x, y) \cdot r(x, y) \tag{2.1}$$

où :

- $i(x, y)$ est l'**éclairement** incident sur la scène (énergie lumineuse par unité de surface), déterminé par la source de lumière ;
- $r(x, y)$ est la **réflectance** de l'objet (fraction de la lumière réfléchie), déterminée par les propriétés optiques de la surface, avec $0 < r(x, y) < 1$.

En pratique, les deux composantes varient à des échelles spatiales distinctes : $i(x, y)$ tend à varier lentement dans l'espace, tandis que $r(x, y)$ peut présenter des variations brusques associées aux textures, aux contours et aux détails fins. Les techniques de traitement d'images (PDI) cherchent souvent à séparer ou à compenser ces composantes, comme dans les méthodes de correction d'éclairement non uniforme.

La Figure 2.3 illustre comment ce modèle se manifeste dans les images numériques en couleur, représentées par plusieurs canaux spectraux (RVB) et, éventuellement, par un canal supplémentaire de transparence (RVBA).

2.4.1 Représentation numérique et canaux spectraux

Dans les images numériques en couleur, la fonction $f(x, y)$ de la Équation 2.1 est représentée par de multiples canaux spectraux. Dans le standard **RGB**, chaque pixel stocke trois échantillons indépendants :

$$\mathbf{f}(x, y) = [R(x, y),\ G(x, y),\ B(x, y)]$$

correspondant aux intensités des composantes rouge (*Red*), verte (*Green*) et bleue (*Blue*). Dans les images de type **RGBA**, on ajoute un quatrième canal :

$$\mathbf{f}(x, y) = [R(x, y),\ G(x, y),\ B(x, y),\ A(x, y)]$$

où $A(x, y)$ représente le canal **alpha** (*Alpha*), chargé de coder l'opacité du pixel. Par convention, la valeur $A = 0$ indique un pixel entièrement transparent, tandis que $A = 255$ (ou 1) représente un pixel entièrement opaque.

Ainsi, une image numérique en couleur est structurée comme une matrice multidimensionnelle de dimensions :

- **RGB :** $M \times N \times 3$
- **RGBA :** $M \times N \times 4$

où chaque position (x, y) stocke les échantillons associés aux propriétés optiques de cette coordonnée spatiale.

La conversion entre plages d'intensité est directe et donnée par :

$$f_{\text{norm}}(x, y) = \frac{f(x, y)}{255}$$

où $f(x, y) \in [0,\ 255]$ et $f_{\text{norm}}(x, y) \in [0,\ 1]$.

Convention de représentation dans `morph.hpp` : la bibliothèque de ce livre adopte une convention unique (Table 2.1), sans les ambiguïtés de plage et d'ordre des canaux qui surgissent lors de la combinaison de plusieurs bibliothèques Python.

Tableau 2.1: Convention de représentation des images dans morph.hpp — plage, ordre des bandes, nombre de canaux et disposition en mémoire.

Aspect	`morph.hpp`
Type d'échantillon	`unsigned char` (`uint8`), plage $[0, 255]$
Ordre des bandes	**RGB** (via `stb_image` ; sans inversion BGR)
Nombre de canaux	`1` (niveaux de gris) ou `3` (RGB)

Aspect	`morph.hpp`
Disposition en mémoire	$H \times W \times C$ entrelacé (`data[(y*w + x)*channels + c]`)

La `struct Image` conserve les pixels dans un `std::vector<unsigned char>` linéaire, avec les champs `h`, `w` et `channels`. La fonction `mm::read` décode avec `stb_image` en forçant 3 canaux (ou 1, lorsque `grayscale=true`) ; par conséquent **un éventuel canal alpha du fichier est écarté à la lecture**. Pour travailler en virgule flottante, la même relation $f_{\text{norm}} = f/255$ s'applique.

Dans la cellule d'exemple suivante, la version RGBA ($M \times N \times 4$) est construite en empilant un canal alpha synthétique sur le tableau lu — le noyau du notebook est en Python, et l'affichage reçoit le `ndarray` dans cette disposition $H \times W \times C$.

2.4.2 Préparation de l'environnement pratique

Le bloc suivant charge le module `morph.py` du dépôt et illustre, pour un pixel synthétique, les différentes conventions d'échelle et d'ordre des canaux adoptées par les principales bibliothèques de traitement d'images.

```
import os, urllib.request

os.makedirs("tmp/state", exist_ok=True)  # artefacts de build du parcours C++ (.cpp, binaire,
PNGs)

url = "https://raw.githubusercontent.com/fzampirolli/pdi-vc/master/morph/config.py"
if not os.path.exists("config.py"):
    urllib.request.urlretrieve(url, "config.py")

# Le noyau est Python même dans le parcours C++ : `mm` (morph.py) est utilisé par les
# simulateurs, par l'affichage des figures que le binaire C++ génère et par l'
# état mm::Image entre les cellules. cpp=True télécharge aussi le parcours compilé
# (morph.hpp + stb_image*.h), utilisé dans le #include des cellules %%writefile *.cpp.
import config
config.setup(demo=True, cpp=True)
from morph import mm
```

```
Environnement prêt. Morph : 1.1.9 | OpenCV : 5.0.0

Conventions d'échelle par bibliothèque
NumPy / Pillow / OpenCV (uint8) : [200 100  50] → [0, 255]
scikit-image / PyTorch (float) :  [0.78431373 0.39215686 0.19607843] → [0.0, 1.0]
OpenCV : attention - lecture en BGR : [ 50 100 200] (canaux inversés)
```

2.4.3 Implémentation de la Lecture d'Image dans `morph.hpp`

Le passage suivant affiche le code source de la fonction `mm::read`, permettant de vérifier directement comment la bibliothèque `morph.hpp` implémente la lecture d'images.

```
# Le morph.hpp est header-only ; ci-dessous, le corps de la fonction mm::read().
import re, pathlib
hpp = pathlib.Path("morph.hpp").read_text()
m = re.search(r"inline Image read\(.*?\n\}", hpp, re.S)
print(m.group(0) if m else "(mm::read não encontrada em morph.hpp)")
```

```
inline Image read(const std::string& path_or_url, bool grayscale = false) {
    std::string local_path = path_or_url;
    bool is_url = path_or_url.rfind("http://", 0) == 0 ||
                  path_or_url.rfind("https://", 0) == 0;
```

```
    if (is_url) {
        local_path = "_mm_download_tmp.img";
        if (!_download(path_or_url, local_path))
            throw std::runtime_error("mm::read: falha ao baixar '" + path_or_url + "'");
    }

    int w, h, ch;
    int desired = grayscale ? 1 : 3;
    unsigned char* data = stbi_load(local_path.c_str(), &w, &h, &ch, desired);
    if (!data) {
        Image fallback;
        if (_try_read_ascii_pgm(local_path, fallback)) return fallback;
        throw std::runtime_error("mm::read: falha ao decodificar '" + path_or_url + "'");
    }

    Image img(h, w, desired);
    std::copy(data, data + (size_t)w * h * desired, img.data.begin());
    stbi_image_free(data);
    return img;
}
```

2.4.4 RGB et RGBA : Exemple pratique avec l'image Mandrill

L'exemple suivant charge l'image Mandrill au format RGB, construit artificiellement un canal alpha avec un dégradé horizontal et affiche les deux représentations, illustrant concrètement les structures matricielles $M \times N \times 3$ et $M \times N \times 4$.

```
%%writefile tmp/fig_02_rgb_rgba.cpp
#define MM_OUT "tmp/fig_02_rgb_rgba.png"
//| label: fig-02-rgb-rgba
//| fig-cap: "Imagem RGB (3 canais, $M \\times N \\times 3$) e versão RGBA (4 canais, $M
\\times N \\times 4$) com transparência alfa gradual da esquerda para a direita. Imagem
Mandrill - [USC SIPI Image Database](https://sipi.usc.edu/database/database.php?volume=misc)
(domínio público)."
//| echo: true

#include "morph.hpp"
#include <iostream>
#include <filesystem>

int main() {
    // https://commons.wikimedia.org/wiki/File:Mandrill-k-means.png
    mm::Image img_rgb =
mm::read("https://upload.wikimedia.org/wikipedia/commons/a/ab/Mandrill-k-means.png"); // (M,
N, 3), uint8

    // Canal alfa: gradiente horizontal 0 -> 255 (esquerda -> direita)
    int h = img_rgb.h, w = img_rgb.w;
    mm::Image alpha(h, w);
    for (int x = 0; x < w; x++) {
        for (int y = 0; y < h; y++) {
            alpha.at(y, x) = static_cast<unsigned char>(255 * x / (w - 1));
        }
    }

    // Empilha RGB + alfa -> RGBA (M, N, 4)
    mm::Image img_rgba(h, w, 4);
    for (int y = 0; y < h; y++) {
        for (int x = 0; x < w; x++) {
            for (int ch = 0; ch < 3; ch++) {
```

```
                img_rgba.at(y, x, ch) = img_rgb.at(y, x, ch);
            }
            img_rgba.at(y, x, 3) = alpha.at(y, x);
        }
    }

    std::cout << "RGB   -> shape=" << img_rgb.h << " x " << img_rgb.w << " x " <<
img_rgb.channels << "\n";
    std::cout << "RGBA  -> shape=" << img_rgba.h << " x " << img_rgba.w << " x " <<
img_rgba.channels << "\n";

    mm::show(std::vector<mm::Image>{img_rgb, img_rgba},
             MM_OUT,
             std::vector<std::string>{"RGB  -  M x N x 3", "RGBA  -  M x N x 4"},
             2);


// [pdi:panel-io] auto-generated - do not edit by hand
std::filesystem::create_directories("tmp");
mm::write(img_rgb, "tmp/fig_02_rgb_rgba_0.png");
mm::write(img_rgba, "tmp/fig_02_rgb_rgba_1.png");
// [pdi:panel-io:end]
return 0;
}
```

```
Overwriting tmp/fig_02_rgb_rgba.cpp
```

```
!g++ -I. -std=c++17 tmp/fig_02_rgb_rgba.cpp -o tmp/fig_02_rgb_rgba \
  && ./tmp/fig_02_rgb_rgba \
  && test -f "tmp/fig_02_rgb_rgba.png" \
  || echo "  mm::show não gravou tmp/fig_02_rgb_rgba.png"
```

```
RGB   -> shape=512 x 512 x 3
RGBA  -> shape=512 x 512 x 4
[1] RGB  -  M x N x 3
[2] RGBA  -  M x N x 4
```

```
try:
    mm.show(
        [
            mm.read("tmp/fig_02_rgb_rgba_0.png"),
            mm.read("tmp/fig_02_rgb_rgba_1.png"),
        ],
        titles=[
            'RGB  -  M x N x 3',
            'RGBA  -  M x N x 4',
        ],
        cols=2,
    )
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + " tmp/fig_02_rgb_rgba_0.png
    (ver a versao Python)")
```

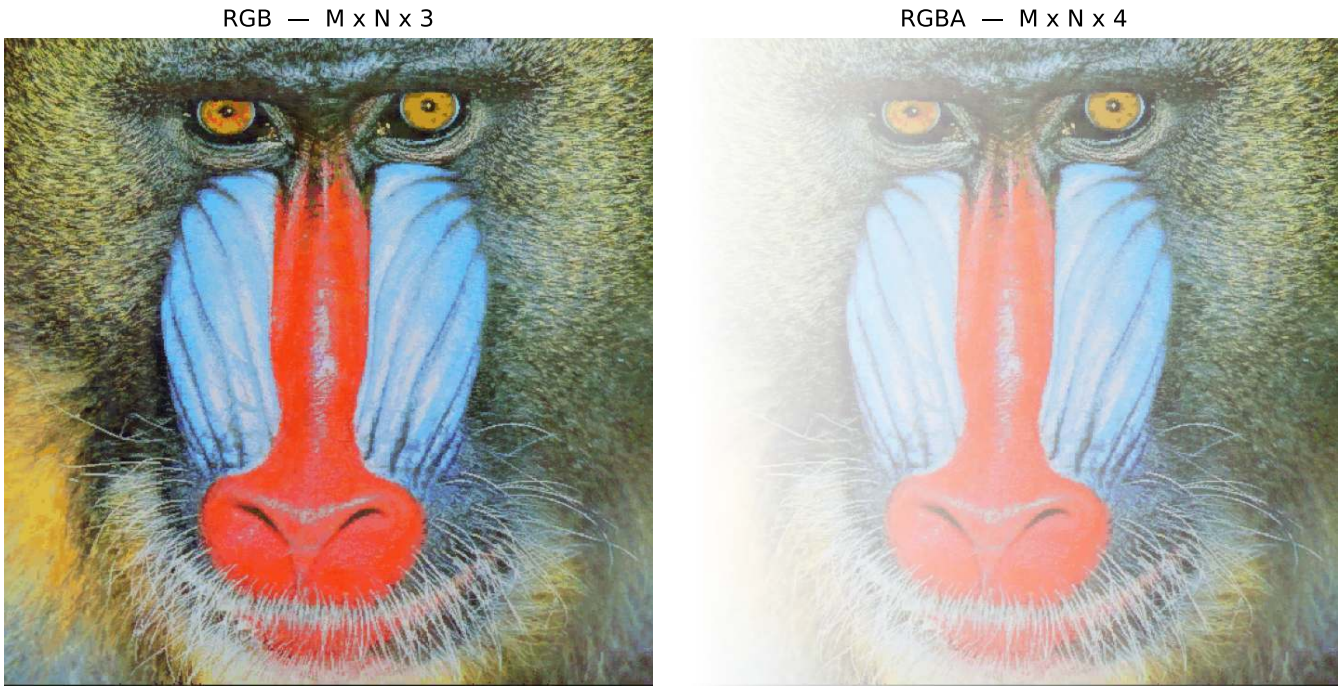


Figure 2.3: Imagem RGB (3 canais, $M \times N \times 3$) e versão RGBA (4 canais, $M \times N \times 4$) com transparência alfa gradual da esquerda para a direita. Imagem Mandrill — USC SIPI Image Database (domínio público).

2.5 Numérisation : Échantillonnage et Quantification

Pour transformer une scène continue en une image numérique, deux processus sont nécessaires : l'échantillonnage et la quantification.

2.5.1 Échantillonnage — Discrétisation de l'espace

L'échantillonnage consiste à mesurer la valeur de la fonction $f(x, y)$ en des points régulièrement espacés, formant une matrice de M lignes (hauteur) et N colonnes (largeur). Chaque élément de cette matrice est un **pixel**. La **résolution spatiale** est donnée par $M \times N$. Plus la résolution est élevée, plus les détails spatiaux sont préservés, mais plus le coût computationnel et de stockage augmente également.

2.5.2 Quantification — Discrétisation de l'intensité

La quantification associe à chaque pixel une valeur numérique discrète, généralement représentée par un entier de b bits. La **profondeur de bits** définit le nombre de niveaux d'intensité : 2^b. Les images en niveaux de gris utilisent couramment 8 bits (256 niveaux). Les images couleur utilisent trois canaux de 8 bits (24 bits au total).

Illustration : Si l'on utilise seulement 1 bit par pixel (noir et blanc), on perd tous les tons intermédiaires. Avec 2 bits (4 niveaux), on perçoit déjà des dégradés grossiers. Avec 8 bits, l'œil humain perçoit difficilement la discrétisation (vision continue).

> ⚠ Erreur de quantification
>
> L'**erreur de quantification** est la différence entre la valeur analogique réelle et la valeur discrète attribuée. Elle se manifeste sous forme de bruit de quantification, visible dans les régions à gradient doux lorsque l'on utilise peu de bits.

2.5.3 Effets de l'échantillonnage et de la quantification

Les expériences suivantes montrent comment la réduction de la résolution spatiale (sous-échantillonnage) et de la profondeur de bits dégrade la qualité visuelle. Utilisez le code pour explorer différents facteurs et niveaux de gris.

```
%%writefile tmp/mm_out_1.cpp
// Compile: g++ -std=c++17 program.cpp -o program -lmorph

#include "morph.hpp"
#include <iostream>
#include <filesystem>

//| quarto-raw: true

int main() {
    // Imagem de exemplo (barbudo-rajado) - base das figuras de amostragem,
    // quantização e transformações geométricas deste capítulo.
    std::string url =
"https://upload.wikimedia.org/wikipedia/commons/c/c5/Area_de_Prote%C3%A7%C3%A3o_Ambiental_Quilombos_do_M

    mm::Image img_color = mm::read(url);
    mm::Image img_gray0 = mm::gray(img_color);
    mm::Image img_gray  = mm::crop(img_gray0, 820, 1850, 890, 1550);  // recorte p/ ver
detalhes
    std::cout << "Imagem original: (" << img_color.h << ", " << img_color.w << ", " <<
img_color.channels << ")\n";


// [pdi:state-io] auto-generated - do not edit by hand
std::filesystem::create_directories("tmp/state");
mm::write(img_gray, "tmp/state/img_gray_22.png");
// [pdi:state-io:end]
return 0;
}
```

```
Overwriting tmp/mm_out_1.cpp
```

```
!g++ -I. -std=c++17 tmp/mm_out_1.cpp -o tmp/mm_out_1 \
  && ./tmp/mm_out_1
```

```
Imagem original: (3067, 2047, 3)
```

L'expérience présentée dans Figure 2.4 illustre le compromis entre la **résolution spatiale** et le **coût de stockage** en mémoire. Le code utilise la technique de sous-échantillonnage par tranchage (*slicing*) pour réduire la matrice originale de pixels selon un facteur f, ce qui entraîne une économie de mémoire — par exemple, un facteur $f = 8$ réduit la taille des données de 64 fois (8^2). À des fins de comparaison visuelle, les images réduites sont restaurées à leurs dimensions originales (512×512) via une interpolation par **plus proche voisin** (`nearest`). Ce processus ne récupère pas l'information perdue, mais rend évident l'effet d'***aliasing*** et la structure en blocs (pixelisation) générée par la faible densité de données de la matrice échantillonnée.

```
%%writefile tmp/fig_02_subamostragem.cpp
#define MM_OUT "tmp/fig_02_subamostragem.png"
//| label: fig-02-subamostragem
//| fig-cap: "Efeito da subamostragem. Os títulos exibem as dimensões (W x H) e o tamanho da
matriz em memória (KB)."
//| echo: true
//| output: true
```

```cpp
#include "morph.hpp"
#include <iostream>
#include <string>
#include <vector>

std::pair<mm::Image, std::string> subsample_simple(const mm::Image& image, int f) {
    // Subamostragem via fatiamento (slicing)
    mm::Image reduced = mm::subsample(image, f);

    // Cálculo de memória em KB
    double mem_kb = (reduced.h * reduced.w * reduced.channels) / 1024.0;

    std::string label = std::to_string(reduced.w) + "x" + std::to_string(reduced.h) + ", " +
                        std::to_string((int)mem_kb) + " KB\n(Fator " + std::to_string(f) +
")";

    // Restaura o tamanho para visualização (H, W originais)
    mm::Image res = mm::resize(reduced, image.w, image.h, "nearest");
    return {res, label};
}

int main() {
// [pdi:state-io] auto-generated - do not edit by hand
mm::Image img_gray = mm::_read_state("tmp/state/img_gray_22.png");
// [pdi:state-io:end]

    // img_gray is provided (already initialized)

    std::vector<int> factors = {1, 4, 8, 12};

    // Gera os resultados e separa em listas para o mm.show
    std::vector<mm::Image> imgs_list;
    std::vector<std::string> titles_list;

    for (int f : factors) {
        auto [res, label] = subsample_simple(img_gray, f);
        imgs_list.push_back(res);
        titles_list.push_back(label);
    }

    mm::show(imgs_list, MM_OUT, titles_list, 4);

    return 0;
}
```

```
Overwriting tmp/fig_02_subamostragem.cpp
```

```
!g++ -I. -std=c++17 tmp/fig_02_subamostragem.cpp -o tmp/fig_02_subamostragem \
  && ./tmp/fig_02_subamostragem \
  && test -f "tmp/fig_02_subamostragem.png" \
  || echo "  mm::show não gravou tmp/fig_02_subamostragem.png"
```

```
[1] 660x1030, 663 KB
(Fator 1)
[2] 165x258, 41 KB
(Fator 4)
[3] 83x129, 10 KB
(Fator 8)
[4] 55x86, 4 KB
```

```
(Fator 12)
```

```
try:
    mm.show(mm.read("tmp/fig_02_subamostragem.png"), figsize=(16, 12))
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + "
    tmp/fig_02_subamostragem.png (ver a versao Python)")
```

Figure 2.4: Efeito da subamostragem. Os títulos exibem as dimensões (W x H) e o tamanho da matriz em memória (KB).

L'expérience dans Figure 2.5 se concentre sur la **quantification d'intensité**, le processus de discrétisation de l'amplitude de la fonction $f(x, y)$. Alors que le sous-échantillonnage affecte la grille spatiale, la réduction de la profondeur de bits limite la quantité de niveaux de gris disponibles pour représenter la luminosité.

En réduisant la profondeur de 8 bits (256 niveaux) à des valeurs inférieures, l'effet de **postérisation** apparaît, où les dégradés lisses d'une scène sont remplacés par des transitions abruptes. À la limite de 1 bit, l'image devient strictement binaire, ne préservant que la silhouette et perdant les détails de texture et de volume.

```
%%writefile tmp/fig_02_quantizacao.cpp
#define MM_OUT "tmp/fig_02_quantizacao.png"
//#| label: fig-02-quantizacao
//#| fig-cap: "Efeito da redução da profundidade de bits. Os títulos exibem a quantidade de
bits, níveis e o tamanho em memória (KB)."
//#| echo: true
//#| output: true

#include "morph.hpp"
#include <iostream>
#include <vector>
#include <string>
#include <cmath>
#include <utility>

int main() {
// [pdi:state-io] auto-generated - do not edit by hand
mm::Image img_gray = mm::_read_state("tmp/state/img_gray_22.png");
// [pdi:state-io:end]

    // img_gray is already defined as mm::Image (provided externally)

    // Fonction de quantification
    auto quantize_simple = [](const mm::Image& image, int bits) {
        // Réduit la profondeur de bits et calcule les métadonnées mémoire
        int levels = std::pow(2, bits);
```

```
        // Normalisation et quantification uniforme
        mm::Image quantized(image.h, image.w);
        for (int y = 0; y < image.h; ++y) {
            for (int x = 0; x < image.w; ++x) {
                int val = image.at(y, x);
                double normalized = std::floor((val / 256.0) * levels) / levels * 255.0;
                quantized.at(y, x) = static_cast<unsigned char>(std::max(0, std::min(255,
static_cast<int>(normalized))));
            }
        }

        // Calcul de la mémoire en Ko
        double mem_kb = (quantized.h * quantized.w * quantized.channels) / 1024.0;
        std::string label = std::to_string(bits) + " bits (" + std::to_string(levels) + "
níveis)\n" + std::to_string(static_cast<int>(mem_kb)) + " KB";

        return std::make_pair(quantized, label);
    };

    // Liste de bits pour le test (8 est le défaut, 1 est le binaire)
    std::vector<int> bits_test = {8, 4, 2, 1};

    // Génère les résultats et les sépare en listes pour mm::show
    std::vector<std::pair<mm::Image, std::string>> results_q;
    for (int b : bits_test) {
        results_q.push_back(quantize_simple(img_gray, b));
    }
    std::vector<mm::Image> imgs_q;
    std::vector<std::string> titles_q;
    for (const auto& r : results_q) {
        imgs_q.push_back(r.first);
        titles_q.push_back(r.second);
    }

    mm::show(imgs_q, MM_OUT, titles_q, 4);

    return 0;
}
```

```
Overwriting tmp/fig_02_quantizacao.cpp
```

```
!g++ -I. -std=c++17 tmp/fig_02_quantizacao.cpp -o tmp/fig_02_quantizacao \
  && ./tmp/fig_02_quantizacao \
  && test -f "tmp/fig_02_quantizacao.png" \
  || echo "  mm::show não gravou tmp/fig_02_quantizacao.png"
```

```
[1] 8 bits (256 níveis)
663 KB
[2] 4 bits (16 níveis)
663 KB
[3] 2 bits (4 níveis)
663 KB
[4] 1 bits (2 níveis)
663 KB
```

```
try:
    mm.show(mm.read("tmp/fig_02_quantizacao.png"), figsize=(16, 12))
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + " tmp/fig_02_quantizacao.png
    (ver a versao Python)")
```

Figure 2.5: Efeito da redução da profundidade de bits. Os títulos exibem a quantidade de bits, níveis e o tamanho em memória (KB).

2.5.4 Analyse technique

- **Domaine vs. Codomaine** : Notez que la résolution spatiale (dimensions de la matrice) reste constante à 512x512 ; seul le codomaine de la fonction image change.
- **Constance de la mémoire** : Observez dans les titres que la taille en Ko ne diminue pas. Cela s'explique par le fait que NumPy stocke chaque pixel quantifié dans un conteneur de 8 bits (`uint8`), indépendamment du fait que la valeur réelle soit seulement 0 ou 1.
- **Perception** : La dégradation visuelle devient critique en dessous de 4 bits, où l'œil humain commence à percevoir les « frontières » artificielles créées par le manque de tons intermédiaires.

La limitation des types de données plus petits qu'un octet dans l'écosystème Python/NumPy est due à l'architecture matérielle, qui adresse la mémoire en blocs de 8 bits (octets). Afin de maintenir la compatibilité avec OpenCV et de garantir l'efficacité, même les éléments binaires sont mappés vers des conteneurs de 1 octet (`uint8` ou `bool8`).

Bien que des langages comme l'ANSI C permettent la compression de 8 pixels par octet (*bit-packing*), cette approche exige une décompression constante pour les calculs et impose une complexité élevée dans la manipulation des pointeurs. Selon la Table 2.2, l'utilisation de `uint8` est privilégiée pour la facilité d'accès aux voisins et la polyvalence dans les transformations géométriques. De plus, les méthodes natives de NumPy et OpenCV exécutent le traitement en interne à bas niveau (C/C++), ce qui rend les opérations vectorisées plus rapides que les implémentations manuelles avec des boucles imbriquées en Python.

Tableau 2.2: Comparatif entre les stratégies de compression et l'efficacité du traitement.

Caractéristique	Python (NumPy/OpenCV)	ANSI C (Bit-packing)
Plus petite unité	1 octet (8 bits)	1 bit
Mémoire (binaire)	256 Ko (pour 512x512)	32 Ko (pour 512x512)
Vitesse	Élevée (vectorisation en C)	Variable (lente en cas de décalage de bits)
Complexité	Faible : méthodes prêtes à l'emploi	Élevée : pointeurs et masques

2.6 Relations entre Pixels - Topologie de l'Image

Les pixels ne sont pas des éléments isolés ; leurs positions relatives définissent des concepts importants pour le traitement.

2.6.1 Voisinage

Étant donné un pixel de coordonnées (x, y), on définit deux types principaux de voisinage (pour les images en grille rectangulaire) :

- **Voisinage-4** (von Neumann) : inclut les pixels aux positions $(x-1, y)$, $(x+1, y)$, $(x, y-1)$, $(x, y+1)$.
- **Voisinage-8** (Moore) : inclut les huit pixels adjacents (ajoute les quatre diagonales).

Le choix du voisinage influence des opérations telles que la détection de contours, le calcul des gradients et la connectivité.

```
%%writefile tmp/fig_02_vizinhanca.cpp
#define MM_OUT "tmp/fig_02_vizinhanca.png"
//| label: fig-02-vizinhanca
//| fig-cap: "Ilustração de vizinhanças 4 em uma matriz 3x3. No centro (1,1), o pixel de
interesse."
//| echo: true
//| output: true

#include "morph.hpp"

int main() {
    // # Criação de uma matriz 3x3 para exemplo topológico
    // viz = np.zeros((3, 3), dtype='uint8')
    //
    // # Definindo Vizinhança-4 (N4) com valor diferente para destaque
    // viz[0, 1] = viz[2, 1] = viz[1, 0] = viz[1, 2] = viz[1, 1] = 255
    //
    // ou simplesmente (teste com números como argumentos):
    mm::Image viz = mm::secross();

    // Exibição da matriz para análise de coordenadas
    mm::drawImgPlt(viz, MM_OUT, 40);

    return 0;
}
```

```
Overwriting tmp/fig_02_vizinhanca.cpp
```

```
!g++ -I. -std=c++17 tmp/fig_02_vizinhanca.cpp -o tmp/fig_02_vizinhanca \
  && ./tmp/fig_02_vizinhanca \
  && test -f "tmp/fig_02_vizinhanca.png" \
  || echo "  mm::show não gravou tmp/fig_02_vizinhanca.png"
```

```
0 1 0
1 1 1
0 1 0
```

```
try:
    mm.show(mm.read("tmp/fig_02_vizinhanca.png"))
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + " tmp/fig_02_vizinhanca.png
    (ver a versao Python)")
```

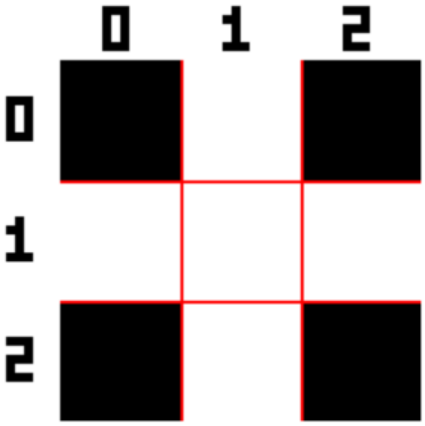


Figure 2.6: Ilustração de vizinhanças 4 em uma matriz 3x3. No centro (1,1), o pixel de interesse.

2.6.2 Adjacence, connectivité et chemins

Deux pixels sont **adjacents** s'ils sont en contact selon un voisinage défini **et** satisfont un critère de valeur (ex. : même niveau d'intensité). Une **connectivité** définit une relation d'équivalence entre les pixels qui forment une région connexe. Un **chemin** est une séquence de pixels adjacents.

La **connectivité-4** (N4) et la **connectivité-8** (N8) peuvent produire des résultats différents dans la segmentation et le calcul des composantes connexes (étiquetage). Par exemple, un motif en damier peut être complètement déconnecté en N4, mais totalement connecté en N8.

2.6.3 Distances entre pixels

Les métriques de distance sont fondamentales pour quantifier la proximité physique et la connectivité entre les éléments qui composent la grille numérique. Comme démontré dans la Table 2.3, le choix de la métrique définit le coût de déplacement entre pixels et modifie le comportement des algorithmes de segmentation et d'analyse morphologique.

Pour mesurer la distance entre deux pixels $p(x_1, y_1)$ et $q(x_2, y_2)$, on utilise différentes fonctions métriques qui imposent des contraintes de mouvement distinctes sur la *grille* :

Tableau 2.3: Comparatif des métriques de distance appliquées à la maille de pixels.

Métrique	Définition	Interprétation
Euclidienne	$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$	Distance **exacte** en ligne droite (continue)
Manhattan (*City block*)	$\lvert x_1 - x_2 \rvert + \lvert y_1 - y_2 \rvert$	Mouvements horizontaux + verticaux
Chebyshev (*Échiquier*)	$\max(\lvert x_1 - x_2 \rvert, \lvert y_1 - y_2 \rvert)$	Plus grand déplacement entre les axes

Ces distances sont appliquées dans divers contextes de TDI, notamment les algorithmes d'interpolation géométrique, les transformations de distance, la croissance de régions et l'analyse de formes.

i Exemple pratique

Considérant deux pixels avec des déplacements relatifs $\Delta x = 3$ et $\Delta y = 4$:

- **Euclidienne** : $\sqrt{3^2 + 4^2} = 5$ (hypoténuse du triangle rectangle).
- ***Manhattan*** : $3 + 4 = 7$ (somme des cathètes).
- ***Chebyshev*** : $\max(3, 4) = 4$ (prédominance du plus grand déplacement).

2.7 Stockage d'images

Le choix du format de fichier est une étape décisive dans le flux de traitement, car il détermine la manière dont les données d'échantillonnage et de quantification seront préservées ou écartées. Comme présenté dans la Table 2.4, chaque extension équilibre de façon distincte la fidélité des données et l'efficacité de stockage.

Tableau 2.4: Principaux formats de stockage d'images numériques et leurs applications en TNI.

Format	Caractéristiques	Usage typique
PGM	Format simple de carte de niveaux de gris (texte ou binaire).	Recherche académique et outils Unix.
BMP	Non compressé (ou compression simple).	Windows, applications héritées.
PNG	Compression **sans perte** (*lossless*).	Web, images avec transparence.
JPEG	Compression **avec perte** (*lossy*), idéale pour les photographies.	Photos, appareils photo numériques.
TIFF	Prend en charge plusieurs couches et une compression variée.	PAO, archivage.
RAW	Données brutes du capteur, sans traitement.	Photographie professionnelle.
DICOM	Norme médicale avec métadonnées cliniques intégrées (patient, équipement, protocole).	Radiologie, tomodensitométrie, imagerie par résonance magnétique.

Les métadonnées d'une image incluent des paramètres tels que la largeur, la hauteur, la profondeur de bits et le codage des couleurs. Dans les formats scientifiques, les informations d'étalonnage et les détails de capture sont également préservés. Lors de l'utilisation de la fonction `mm::read()`, la bibliothèque `morph` préserve automatiquement ces données afin de respecter les propriétés originales de l'image.

Dans les contextes scientifiques et hospitaliers, la norme **DICOM** (*Digital Imaging and Communications in Medicine*) est privilégiée pour garantir qu'aucune perte de précision diagnostique ne se produise. Des référentiels publics tels que **The Cancer Imaging Archive (TCIA)**, le **Alzheimer's Disease Neuroimaging Initiative (ADNI)** et **PhysioNet** mettent à disposition de vastes ensembles de données dans ce format, incluant des métadonnées cliniques anonymisées qui sont essentielles pour la recherche scientifique.

2.7.1 Exemple : Extraction de métadonnées et localisation GPS

Contrairement à la matrice de pixels pure obtenue par la lecture conventionnelle — comme dans l'image de l'oiseau présentée au début de ce chapitre —, l'utilisation de l'argument `pil=True` dans la méthode `mm::read()` modifie la nature de l'objet retourné (voir Figure 2.7). Alors que le comportement par défaut (`pil=False`) retourne un `numpy.ndarray` RGB, la lecture avec `pil=True` retourne un objet spécialisé de la bibliothèque Pillow, capable d'interpréter l'en-tête **EXIF**.

L'en-tête **EXIF** (*Exchangeable Image File Format*) fonctionne comme un référentiel technique de la capture, permettant à l'objet Pillow d'interpréter une vaste gamme d'informations qui vont bien au-delà des coordonnées GPS. En utilisant `pil=True`, le système accède à l'« ADN » de l'image, incluant les métadonnées matérielles (marque et modèle de l'appareil photo), les réglages optiques (ouverture, distance focale et temps d'exposition) et les paramètres d'éclairage (utilisation du flash et balance des blancs).

Cette distinction est importante en traitement d'images numériques, car elle transforme la matrice d'échantillonnage en un ensemble de données contextualisées, où les caractéristiques physiques du capteur et de l'objectif peuvent être utilisées pour normaliser les luminosités ou corriger les distorsions géométriques.

i Parcours C++ : pourquoi l'extraction des EXIF reste en Python

L'extraction des métadonnées est une **analyse de conteneur** — décodage de l'en-tête EXIF intégré au fichier — et non un traitement de pixels : elle est orthogonale au pipeline d'échantillonnage et de quantification. Le fichier `morph.hpp` décode les images avec `stb_image`, qui ne fournit que la matrice de pixels et élimine l'en-tête EXIF. Comme le noyau de ce notebook est en Python même dans le parcours C++, cet exemple utilise la lecture en mode Pillow (`pil=True`) dans les deux parcours. Dans un programme C++ autonome, le chemin équivalent serait de vendoriser une bibliothèque dédiée : `easyexif` (lecture des EXIF/GPS dans les JPEG, fichier d'en-tête unique, zéro dépendance) ou `exiv2` (lecture **et** écriture, y compris IPTC/XMP).

```
%%writefile tmp/fig_01_natureza.cpp
#define MM_OUT "tmp/fig_01_natureza.png"
#include "morph.hpp"
#include <iostream>
#include <string>

int main() {
    //| label: fig-01-natureza
    //| fig-cap: "Zone de Protection Environnementale Quilombos du Médio Ribeira -
    Barbu-vermiculé (Malacoptila striata). Crédit : Thomas Fuhrmann (CC BY-SA 4.0)."
    //| echo: true

    std::string base    = "https://upload.wikimedia.org/wikipedia/commons";
    std::string arquivo =
        "Area_de_Prote%C3%A7%C3%A3o_Ambiental_"
        "Quilombos_do_M%C3%A9dio_Ribeira_-_Thomas-"
        "Fuhrmann_%282023-_02%29_Malacoptila_striata.jpg";
    std::string url     = base + "/c/c5/" + arquivo;
    std::string caminho = "imagens/barbudo-rajado.jpg";

    // 1. Lecture - l'objet PIL est préservé avec EXIF
    mm::Image img_obj;
    if (/* os.path.exists(caminho) */ false) {
        // os.makedirs("imagens", exist_ok=True) - not directly applicable
        img_obj = mm::read(url);
        mm::write(img_obj, caminho);      // sauvegarde en préservant EXIF
    } else {
        img_obj = mm::read(caminho);
    }

    mm::Image img_numpy = img_obj;      // conversion en Image

    // 2. Extraction et conversion GPS (Tag 34853)
    // Note: EXIF handling with mm::Image is not directly available
    // GPS extraction would require additional EXIF parsing not in mm::Image API

    // 3. Diagnostic des types, dimensions et accès aux pixels
    std::cout << "\nDimensions : " << img_obj.w << " x " << img_obj.h << std::endl;
    std::cout << "Pixel (0,0): " << (int)img_obj.at(0, 0, 0) << ", "
              << (int)img_obj.at(0, 0, 1) << ", " << (int)img_obj.at(0, 0, 2) << std::endl;
    std::cout << "Dimensions [y,x,c]: " << img_numpy.h << ", " << img_numpy.w
              << ", " << img_numpy.channels << std::endl;
    std::cout << "NumPy [0,0]: " << (int)img_numpy.at(0, 0, 0) << ", "
              << (int)img_numpy.at(0, 0, 1) << ", " << (int)img_numpy.at(0, 0, 2) <<
              std::endl;

    // 4. Affichage
    mm::show(img_numpy, MM_OUT);
```

```
    return 0;
}
```

```
Overwriting tmp/fig_01_natureza.cpp
```

```
!g++ -I. -std=c++17 tmp/fig_01_natureza.cpp -o tmp/fig_01_natureza \
  && ./tmp/fig_01_natureza \
  && test -f "tmp/fig_01_natureza.png" \
  || echo "  mm::show não gravou tmp/fig_01_natureza.png"
```

```
Dimensions : 2047 x 3067
Pixel (0,0): 58, 96, 0
Dimensions [y,x,c]: 3067, 2047, 3
NumPy [0,0]: 58, 96, 0
```

```
try:
    mm.show(mm.read("tmp/fig_01_natureza.png"))
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + " tmp/fig_01_natureza.png
    (ver a versao Python)")
```

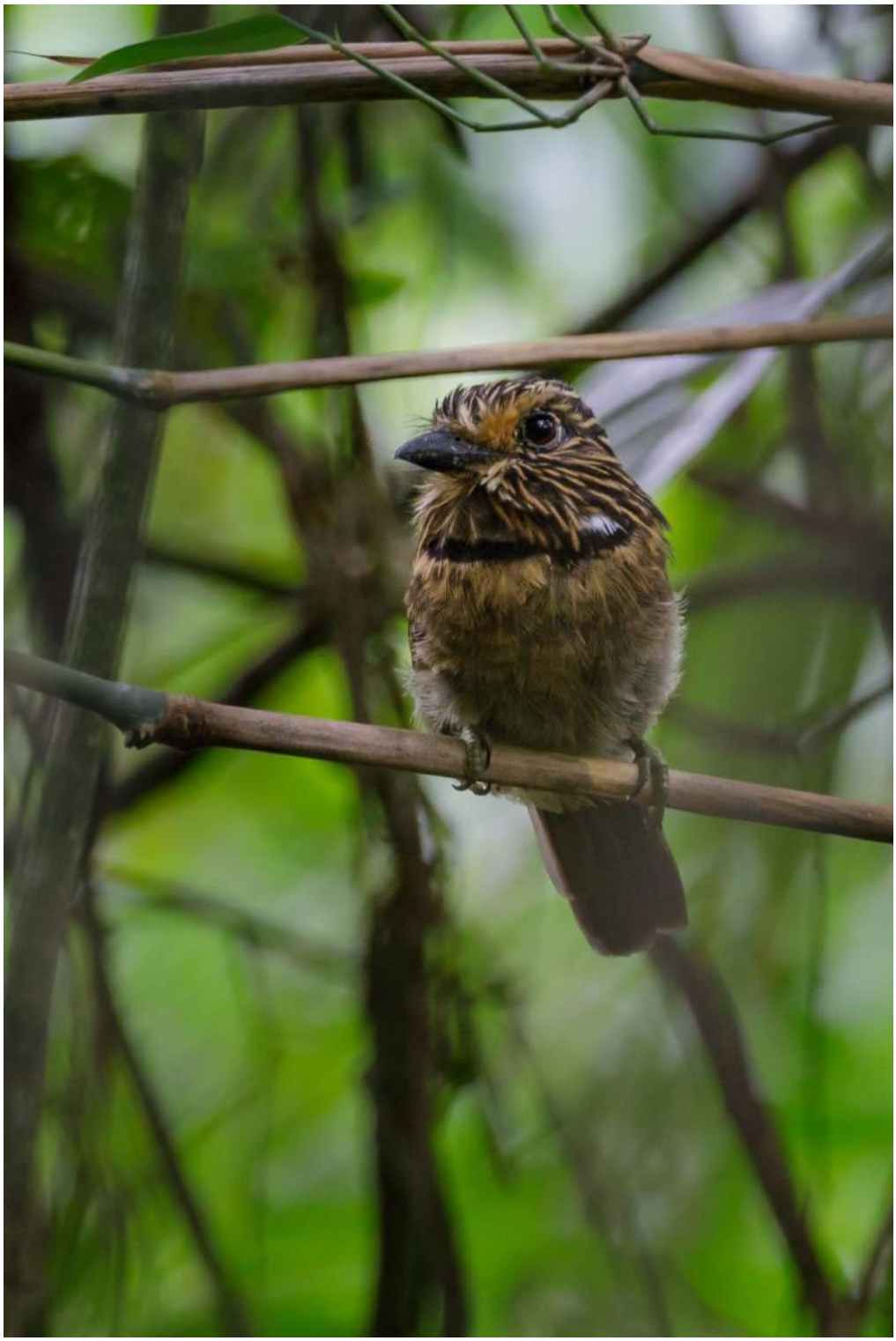


Figure 2.7: Area de Proteção Ambiental Quilombos do Médio Ribeira - Barbudo-rajado (Malacoptila striata). Crédito: Thomas Fuhrmann (CC BY-SA 4.0).

i Note pédagogique : La différence subtile des dimensions

Remarquez que la représentation des dimensions change selon la structure de données utilisée :

- **Dans Pillow (`.size`)** : Renvoie `(Largeur, Hauteur)` — dans l'exemple : `(2047, 3067)`. C'est une vision orientée vers le fichier image.
- **Dans NumPy (`.shape`)** : Suit la convention mathématique des matrices : `(Lignes/Hauteur,`

`Colonnes/Largeur, Canaux)` — dans l'exemple : `(3067, 2047, 3)`.
Cette distinction est fondamentale pour éviter les erreurs d'indexation lors de l'implémentation de filtres manuels. Alors que l'objet Pillow porte le « **où** » et le « **quand** » (contexte), le tableau NumPy porte le « **combien** » de lumière (intensité) existant à chaque point de l'image.

2.7.2 Pourquoi cette séparation est-elle importante ?

Lors du chargement d'une image par le chemin conventionnel (`pil=False`), le résultat est un `numpy.ndarray`, qui contient strictement les valeurs numériques issues de la **quantification** et de l'**échantillonnage**. Cependant, en utilisant `pil=True`, `mm::read()` renvoie un objet de la classe `PIL.JpegImagePlugin.JpegImageFile`.

Cette classe maintient le fichier « ouvert » pour permettre l'accès au contexte de la capture avant que les données ne soient converties en matrice brute. Notez que le pixel `(0, 0)` est identique dans les deux représentations — `(58, 96, 0)` dans Pillow et `[58, 96, 0]` dans NumPy —, confirmant que les deux décrivent les mêmes données, avec des interfaces simplement différentes. Cette séparation est vitale : les pixels servent aux algorithmes ; les métadonnées servent au géoréférencement, au catalogage scientifique et aux corrections basées sur le matériel d'acquisition.

Pour inspecter toutes les métadonnées EXIF d'un objet Pillow :

```
from PIL import ExifTags

exif_raw = img_obj._getexif()
if exif_raw:
    for tag_id, valor in sorted(exif_raw.items()):
        tag_nome = ExifTags.TAGS.get(tag_id, f"TAG_{tag_id}")
        print(f"  {tag_nome:40s} : {valor}")
```

2.8 Transformations géométriques de base

Les transformations géométriques modifient la position des pixels, en conservant les valeurs d'intensité. Elles sont fondamentales pour l'alignement, la correction des distorsions et l'augmentation des données (*data augmentation*) en apprentissage automatique.

Une **transformation affine** est toute application qui préserve la colinéarité (des points sur une droite restent sur une droite) ainsi que les rapports de distances entre points colinéaires. En 2D, toute transformation affine peut être exprimée en **coordonnées homogènes** par une matrice 3×3 :

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix}}_{\mathbf{T}} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{2.2}$$

La sous-matrice 2×2 supérieure gauche $\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$ contrôle la rotation, l'échelle et le cisaillement ; le vecteur $(t_x, t_y)^\top$ contrôle la translation. Les transformations géométriques les plus utilisées en TNI — translation, rotation et échelle — sont des cas particuliers de $\mathbf{T}$, et peuvent être **composées** par multiplication matricielle, dans l'ordre $\mathbf{T} = \mathbf{T}_n \cdots \mathbf{T}_2 \mathbf{T}_1$.

i Transformation inverse et interpolation

Dans l'implémentation pratique (`cv2.warpAffine`), on applique la **transformation inverse** : pour chaque pixel (x', y') de l'image destination, on calcule la position d'origine $(x, y) = \mathbf{T}^{-1}(x', y')$ et on

interpole la valeur. Cela évite les trous dans l'image résultante, causés par le mappage direct de pixels entiers vers des positions non entières.

2.8.1 Translation

La translation est la transformation affine la plus simple : elle déplace tous les pixels selon un vecteur (t_x, t_y). En coordonnées homogènes, elle s'exprime par la matrice :

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad \Rightarrow \quad \begin{cases} x' = x + t_x \\ y' = y + t_y \end{cases} \tag{2.3}$$

La troisième ligne de la matrice garantit que l'opération reste dans l'espace affine, permettant de combiner translation, rotation et mise à l'échelle par simple multiplication de matrices. En pratique, `cv2.warpAffine` n'utilise que les deux premières lignes (matrice 2×3), car la troisième est toujours $[0, 0, 1]$.

Les pixels déplacés au-delà de la zone d'origine sont ignorés ; les zones découvertes sont remplies avec 0 (noir). Voir un exemple dans la Figure 2.8..

```
%%writefile tmp/fig_02_translacao.cpp
#define MM_OUT "tmp/fig_02_translacao.png"
//| label: fig-02-translacao
//| fig-cap: "Exemplo de translação da imagem do pássaro com deslocamentos (50,50) e
(100,50)."
//| echo: true
//| output: true
#include "morph.hpp"
#include <iostream>
#include <vector>
#include <string>
#include <filesystem>

int main() {
// [pdi:state-io] auto-generated - do not edit by hand
mm::Image img_gray = mm::_read_state("tmp/state/img_gray_22.png");
// [pdi:state-io:end]

    mm::Image img_tx1 = mm::translate(img_gray, 50, 50);
    mm::Image img_tx2 = mm::translate(img_gray, 100, 50);
    mm::show(std::vector<mm::Image>{img_gray, img_tx1, img_tx2}, MM_OUT,
             std::vector<std::string>{"Original", "Translação (50,50)", "Translação
(100,50)"}, 3);

// [pdi:panel-io] auto-generated - do not edit by hand
std::filesystem::create_directories("tmp");
mm::write(img_gray, "tmp/fig_02_translacao_0.png");
mm::write(img_tx1, "tmp/fig_02_translacao_1.png");
mm::write(img_tx2, "tmp/fig_02_translacao_2.png");
// [pdi:panel-io:end]
return 0;
}
```

```
Overwriting tmp/fig_02_translacao.cpp
```

```
!g++ -I. -std=c++17 tmp/fig_02_translacao.cpp -o tmp/fig_02_translacao \
  && ./tmp/fig_02_translacao \
  && test -f "tmp/fig_02_translacao.png" \
  || echo "  mm::show não gravou tmp/fig_02_translacao.png"
```

```
[1] Original
[2] Translação (50,50)
[3] Translação (100,50)
```

```
try:
    mm.show(
        [
            mm.read("tmp/fig_02_translacao_0.png"),
            mm.read("tmp/fig_02_translacao_1.png"),
            mm.read("tmp/fig_02_translacao_2.png"),
        ],
        titles=[
            'Original',
            'Translação (50,50)',
            'Translação (100,50)',
        ],
        cols=3,
        figsize=(16, 12),
    )
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + "
    tmp/fig_02_translacao_0.png (ver a versao Python)")
```

Figure 2.8: Exemplo de translação da imagem do pássaro com deslocamentos (50,50) e (100,50).

2.8.2 Rotation

The rotation by an angle θ around a central point (c_x, c_y) is composed of three affine transformations: translation to the origin, pure rotation, and translation back. The resulting matrix is:

$$\mathbf{T}_{\text{rot}} = \begin{bmatrix} \cos\theta & -\sin\theta & c_x(1-\cos\theta) + c_y \sin\theta \\ \sin\theta & \cos\theta & c_y(1-\cos\theta) - c_x \sin\theta \\ 0 & 0 & 1 \end{bmatrix} \tag{2.4}$$

In the implementation, `cv2.getRotationMatrix2D` directly generates the first two lines of $\mathbf{T}_{\text{rot}}$ (2×3 matrix

for `warpAffine`), also accepting a scale factor s that multiplies $\cos\theta$ and $\sin\theta$. See an example in Figure 2.9.

Since rotation shifts pixels to new non-integer positions, `cv2.warpAffine` needs to estimate the color of each destination pixel from its neighbors—a process called **interpolation**. The `interp` parameter controls this behavior:

- **`nearest`** (`INTER_NEAREST`): assigns the value of the nearest pixel. Fast, but produces jagged edges (*aliasing*) on diagonal borders.
- **`bilinear`** (`INTER_LINEAR`, default): weighted average of the 4 nearest neighbors. Balances quality and performance—suitable for most cases.
- **`bicubic`** (`INTER_CUBIC`): considers the 16 neighbors on a cubic surface. Produces smoother edges, at the cost of higher processing.

```
%%writefile tmp/fig_02_rotacao.cpp
#define MM_OUT "tmp/fig_02_rotacao.png"
//| label: fig-02-rotacao
//| fig-cap: "Exemplo de rotação da imagem do pássaro em 30° e 45° usando interpolação
bilinear."
//| echo: true
//| output: true

#include "morph.hpp"
#include <iostream>
#include <filesystem>

int main() {
// [pdi:state-io] auto-generated - do not edit by hand
mm::Image img_gray = mm::_read_state("tmp/state/img_gray_22.png");
// [pdi:state-io:end]

    mm::Image img_rot30 = mm::rotate(img_gray, 30, 1.0, "bilinear");
    mm::Image img_rot45 = mm::rotate(img_gray, 45, 1.0, "bilinear");

    mm::show(std::vector<mm::Image>{img_gray, img_rot30, img_rot45},
             MM_OUT,
             std::vector<std::string>{"Original", "Rotação 30°", "Rotação 45°"},
             3);


// [pdi:panel-io] auto-generated - do not edit by hand
std::filesystem::create_directories("tmp");
mm::write(img_gray, "tmp/fig_02_rotacao_0.png");
mm::write(img_rot30, "tmp/fig_02_rotacao_1.png");
mm::write(img_rot45, "tmp/fig_02_rotacao_2.png");
// [pdi:panel-io:end]
return 0;
}
```

```
Overwriting tmp/fig_02_rotacao.cpp
```

```
!g++ -I. -std=c++17 tmp/fig_02_rotacao.cpp -o tmp/fig_02_rotacao \
  && ./tmp/fig_02_rotacao \
  && test -f "tmp/fig_02_rotacao.png" \
  || echo "  mm::show não gravou tmp/fig_02_rotacao.png"
```

```
[1] Original
[2] Rotação 30°
[3] Rotação 45°
```

```
try:
    mm.show(
        [
            mm.read("tmp/fig_02_rotacao_0.png"),
            mm.read("tmp/fig_02_rotacao_1.png"),
            mm.read("tmp/fig_02_rotacao_2.png"),
        ],
        titles=[
            'Original',
            'Rotação 30°',
            'Rotação 45°',
        ],
        cols=3,
        figsize=(16, 12),
    )
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + " tmp/fig_02_rotacao_0.png
    (ver a versao Python)")
```

Figure 2.9: Exemplo de rotação da imagem do pássaro em 30° e 45° usando interpolação bilinear.

2.8.3 Échelle (Redimensionnement)

Le redimensionnement par facteurs (s_x, s_y) est une transformation affine de matrice :

$$\mathbf{T}_{\text{échelle}} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.5}$$

Lorsque $s > 1$ (agrandissement), les pixels de l'image cible correspondent à des positions non entières dans l'image source — ce qui exige une interpolation pour estimer la valeur. Lorsque $s < 1$ (réduction), plusieurs pixels source contribuent à un seul pixel cible — ce qui exige une **décimation**. Les trois mêmes méthodes décrites pour la rotation sont disponibles dans `mm::resize`, à la différence près qu'ici l'impact visuel est plus perceptible : en agrandissement, `nearest` produit un effet de blocs (*pixelation*), tandis que `bicubic` préserve mieux la netteté des contours, conformément à Table 2.5:

Tableau 2.5: Méthodes d'interpolation disponibles dans mm.resize et leurs nombres respectifs de voisins utilisés dans le calcul.

Méthode	Voisins utilisés	Caractéristique
`'nearest'`	1	Rapide ; produit un effet de blocs (*pixelation*)
`'bilinear'`	4	Bon compromis qualité/coût ; contours lisses
`'bicubic'`	16	Netteté accrue ; préférée dans les logiciels professionnels

Le paramètre `size_or_factor` accepte aussi bien un scalaire (facteur uniforme, e.g. `0.5` pour réduire de moitié) qu'un tuple `(largeur, hauteur)` pour des dimensions absolues.

```
%%writefile tmp/fig_02_escala_detalhe.cpp
#define MM_OUT "tmp/fig_02_escala_detalhe.png"
#include "morph.hpp"
#include <iostream>
#include <vector>
#include <filesystem>

int main() {
// [pdi:state-io] auto-generated - do not edit by hand
mm::Image img_gray = mm::_read_state("tmp/state/img_gray_22.png");
// [pdi:state-io:end]

    //| label: fig-02-escala-detalhe
    //| fig-cap: "Comparação de interpolação com zoom no detalhe do olho (recorte 60×60,
    ampliado 4×). Note o efeito de blocos no vizinho mais próximo vs. a suavização na
    bilinear."
    //| echo: true
    //| output: true

    // 1. Recorte da região do bico
    int y = 210, x = 40, offset = 40;
    mm::Image crop = mm::crop(img_gray, y-offset, y+offset, x-offset, x+offset);
    std::cout << "Imagem: " << img_gray.h << "x" << img_gray.w << " | Crop: " << crop.h << "x"
<< crop.w << "\n";

    // 2. Ampliar 4× com mm.resize
    mm::Image crop_nearest  = mm::resize(crop, 4.0, "nearest");
    mm::Image crop_bilinear = mm::resize(crop, 4.0, "bilinear");

    // 3. Exibição comparativa
    mm::show(
        std::vector<mm::Image>{crop, crop_nearest, crop_bilinear},
        MM_OUT,
        std::vector<std::string>{"Original (recorte)", "Vizinho mais próximo (4×)", "Bilinear
(4×)"},
        3
    );


// [pdi:panel-io] auto-generated - do not edit by hand
std::filesystem::create_directories("tmp");
mm::write(crop, "tmp/fig_02_escala_detalhe_0.png");
mm::write(crop_nearest, "tmp/fig_02_escala_detalhe_1.png");
mm::write(crop_bilinear, "tmp/fig_02_escala_detalhe_2.png");
// [pdi:panel-io:end]
return 0;
}
```

```
Overwriting tmp/fig_02_escala_detalhe.cpp
```

```
!g++ -I. -std=c++17 tmp/fig_02_escala_detalhe.cpp -o tmp/fig_02_escala_detalhe \
  && ./tmp/fig_02_escala_detalhe \
  && test -f "tmp/fig_02_escala_detalhe.png" \
  || echo "  mm::show não gravou tmp/fig_02_escala_detalhe.png"
```

```
Imagem: 1030x660 | Crop: 80x80
[1] Original (recorte)
[2] Vizinho mais próximo (4×)
[3] Bilinear (4×)
```

```
try:
    mm.show(
        [
            mm.read("tmp/fig_02_escala_detalhe_0.png"),
            mm.read("tmp/fig_02_escala_detalhe_1.png"),
            mm.read("tmp/fig_02_escala_detalhe_2.png"),
        ],
        titles=[
            'Original (recorte)',
            'Vizinho mais próximo (4×)',
            'Bilinear (4×)',
        ],
        cols=3,
        figsize=(16, 12),
        dpi=200,
    )
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + "
    tmp/fig_02_escala_detalhe_0.png (ver a versao Python)")
```

Figure 2.10: Comparação de interpolação com zoom no detalhe do olho (recorte 60×60, ampliado 4×). Note o efeito de blocos no vizinho mais próximo vs. a suavização na bilinear.

2.8.4 Cisaillement (*Shear*)

Le cisaillement est une transformation affine qui déforme l'image en déplaçant chaque pixel proportionnellement à sa position sur un axe. La matrice générale combine un cisaillement horizontal (sh_x) et un cisaillement vertical (sh_y) :

$$\mathbf{T}_{\text{cisalh}} = \begin{bmatrix} 1 & sh_x & 0 \\ sh_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.6}$$

Pour $sh_x \neq 0$ et $sh_y = 0$, chaque ligne est déplacée horizontalement proportionnellement à sa position verticale — produisant l'effet d'« inclinaison » caractéristique. Voir l'exemple dans la Figure 2.11.

```
%%writefile tmp/fig_02_cisalhamento.cpp
#define MM_OUT "tmp/fig_02_cisalhamento.png"
//| label: fig-02-cisalhamento
//| fig-cap: "Exemplo de cisalhamento da imagem do pássaro: horizontal (shx=0.3), vertical
(shy=0.3) e combinado (shx=0.2, shy=0.2)."
//| echo: true
//| output: true

#include "morph.hpp"
#include <iostream>
#include <vector>
#include <string>
#include <filesystem>

int main() {
// [pdi:state-io] auto-generated - do not edit by hand
mm::Image img_gray = mm::_read_state("tmp/state/img_gray_22.png");
// [pdi:state-io:end]

    mm::Image img_shx = mm::shear(img_gray, 0.3);
    mm::Image img_shy = mm::shear(img_gray, 0.0, 0.3);
    mm::Image img_shc = mm::shear(img_gray, 0.2, 0.2);

    mm::show(
        std::vector<mm::Image>{img_gray, img_shx, img_shy, img_shc},
        MM_OUT,
        std::vector<std::string>{"Original", "Horiz. (shx=0.3)", "Vert. (shy=0.3)", "Combinado
(0.2, 0.2)"},
        4
    );


// [pdi:panel-io] auto-generated - do not edit by hand
std::filesystem::create_directories("tmp");
mm::write(img_gray, "tmp/fig_02_cisalhamento_0.png");
mm::write(img_shx, "tmp/fig_02_cisalhamento_1.png");
mm::write(img_shy, "tmp/fig_02_cisalhamento_2.png");
mm::write(img_shc, "tmp/fig_02_cisalhamento_3.png");
// [pdi:panel-io:end]
return 0;
}
```

```
Overwriting tmp/fig_02_cisalhamento.cpp
```

```
!g++ -I. -std=c++17 tmp/fig_02_cisalhamento.cpp -o tmp/fig_02_cisalhamento \
  && ./tmp/fig_02_cisalhamento \
  && test -f "tmp/fig_02_cisalhamento.png" \
  || echo "  mm::show não gravou tmp/fig_02_cisalhamento.png"
```

```
[1] Original
[2] Horiz. (shx=0.3)
[3] Vert. (shy=0.3)
```

```
[4] Combinado (0.2, 0.2)
```

```
try:
    mm.show(
        [
            mm.read("tmp/fig_02_cisalhamento_0.png"),
            mm.read("tmp/fig_02_cisalhamento_1.png"),
            mm.read("tmp/fig_02_cisalhamento_2.png"),
            mm.read("tmp/fig_02_cisalhamento_3.png"),
        ],
        titles=[
            'Original',
            'Horiz. (shx=0.3)',
            'Vert. (shy=0.3)',
            'Combinado (0.2, 0.2)',
        ],
        cols=4,
        figsize=(16, 12),
    )
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + "
    tmp/fig_02_cisalhamento_0.png (ver a versao Python)")
```

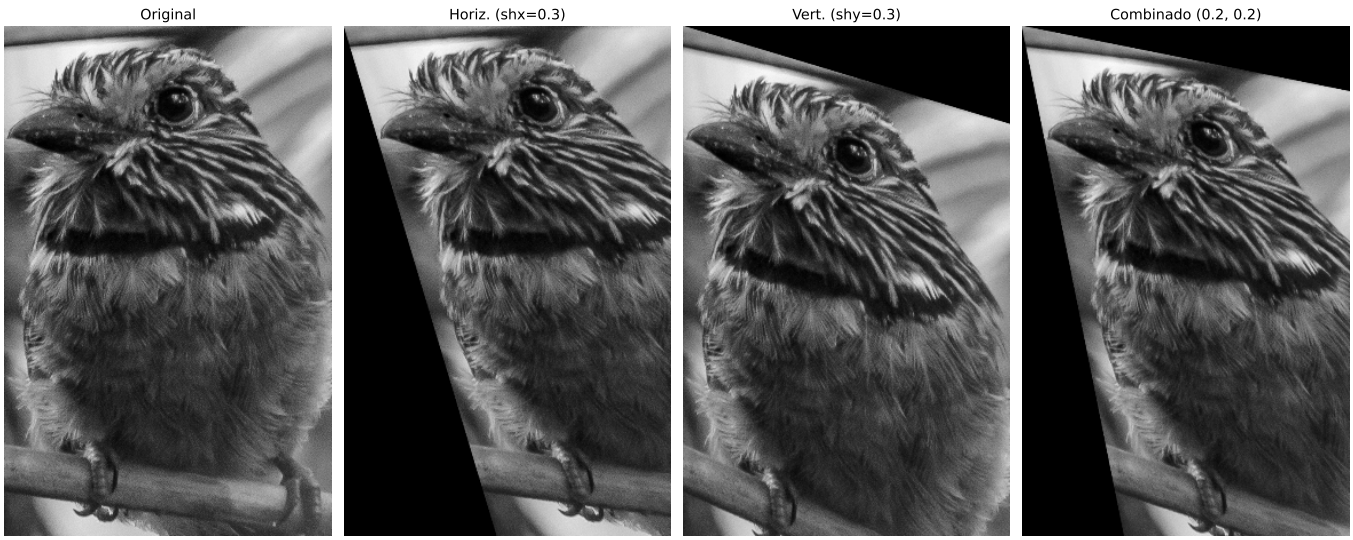


Figure 2.11: Exemplo de cisalhamento da imagem do pássaro: horizontal (shx=0.3), vertical (shy=0.3) e combinado (shx=0.2, shy=0.2).

2.9 Résumé

Dans ce chapitre, les principes fondamentaux de la numérisation et de la topologie des images ont été présentés :

- **Formation de l'image :** $f(x, y) = i(x, y) \cdot r(x, y)$.
- **Échantillonnage :** discrétisation de l'espace → résolution spatiale $M \times N$.
- **Quantification :** discrétisation de l'intensité → profondeur de bits b.
- **Relations topologiques :** voisinage-4, voisinage-8, connectivité, distances (euclidienne, Manhattan, Chebyshev).
- **Transformations géométriques :** translation, rotation, mise à l'échelle (avec interpolation bilinéaire ou plus proche voisin).
- **Formats de fichiers :** BMP, PNG, JPEG, TIFF, RAW ; chacun présentant différents compromis entre qualité et taille.

Le chapitre 3 abordera les **opérations spatiales** telles que la convolution, le filtrage et la morphologie mathématique (érosion, dilatation).

2.10 🤖 Utilisation de Gemini Notebook comme tuteur complémentaire

Dans cette édition, nous encourageons l'utilisation de **Gemini Notebook** comme outil d'apprentissage complémentaire. Cet outil d'IA utilise exclusivement les documents fournis par l'auteur comme base de connaissances, garantissant des réponses cohérentes avec le contenu du livre.

Pour chaque chapitre, nous avons préparé un projet spécifique sur la plateforme. Pour une expérience d'étude enrichie, utilisez l'accès ci-dessous :

🎓 Étudiez avec le tuteur intelligent

Pour interagir avec le contenu de ce chapitre, accédez au lien suivant. L'environnement contient des supports pédagogiques dans différents formats, générés à partir du **PDF** du chapitre. Sur la plateforme, explorez particulièrement les options **Guide d'étude** et **Conversation** pour approfondir votre compréhension.
🚀 ACCÉDER À GEMINI NOTEBOOK : CHAPITRE 02

🌐 Langue et langage de programmation

Le projet de ce chapitre dans Gemini Notebook a été construit uniquement avec le texte en **portugais** et les exemples de code en **Python**. Si vous étudiez avec l'édition en anglais ou en français, ou si vous suivez le parcours en C++, les réponses du tuteur peuvent ne pas correspondre exactement à la version que vous lisez.

⚠️ Avertissement concernant le contenu généré par l'IA

L'IA est une alliée puissante dans les études, mais le contenu généré peut contenir des **erreurs ou des imprécisions**. Consultez toujours **des livres, des articles scientifiques et d'autres sources académiques fiables** pour valider les informations. Chaque fois que possible, exécutez les exemples pratiques fournis dans ce chapitre pour vérifier les résultats.

2.11 Liste d'exercices

1. **(15 %)** Expliquez, avec vos propres mots, la différence entre **échantillonnage** et **quantification**. Donnez un exemple concret de chacun dans le contexte d'une image numérique.
2. **(15 %)** Considérez une image avec une résolution spatiale de 1024×768 pixels et une profondeur de 24 bits (8 bits par canal RVB). Calculez la taille totale non compressée de l'image en octets et en mégaoctets.
3. **(20 %)** À l'aide du code du laboratoire, modifiez le facteur de sous-échantillonnage à 3 et à 6. Décrivez visuellement ce qui se produit au niveau des bords des objets. Qu'est-ce que l'effet de *crénelage* (*aliasing*) ?
4. **(20 %)** Pour l'image en niveaux de gris, appliquez une quantification avec 3 bits (8 niveaux) et 5 bits (32 niveaux). Comparez les résultats et expliquez pourquoi 5 bits peuvent déjà être considérés comme suffisants pour de nombreuses applications.
5. **(15 %)** Étant donné deux pixels $A = (10, 20)$ et $B = (15, 25)$, calculez les distances euclidienne, de Manhattan et de Tchebychev entre eux.
6. **(15 %)** À l'aide de la fonction `mm::rotate`, faites pivoter l'image de l'oiseau selon des angles de 90°, 180° et 270° avec une interpolation bilinéaire. Comparez avec la rotation utilisant `method='nearest'`.

Dans quelles situations l'interpolation du plus proche voisin est-elle encore utile ?

Références du chapitre

La base théorique de ce chapitre s'appuie sur les ouvrages suivants :

- Gonzalez (2018) pour les concepts d'échantillonnage, de quantification et les relations entre pixels.
- Szeliski (2022) pour les transformations géométriques et la connectivité.
- Bradski (2008) pour l'implémentation pratique avec OpenCV et `morph.py`.

2.12 💻 Partie Pratique avec Exercices de Programmation

🎯 Objectif de ce cahier

Le cahier permet de développer, valider, organiser et tester des solutions d'**Exercices de Programmation (EPs)** dans des environnements interactifs, comme Colab, avec les mêmes cas de test que Moodle, en y copiant uniquement au moment d'enregistrer la note officielle.

Téléchargement

Téléchargez `morph.py` et `testsuite.py` en exécutant la cellule ci-dessous :

```
import os, urllib.request

os.makedirs("tmp/state", exist_ok=True)  # artefacts de build du parcours C++ (.cpp, binaire,
PNGs)

url = "https://raw.githubusercontent.com/fzampirolli/pdi-vc/master/morph/config.py"
if not os.path.exists("config.py"):
    urllib.request.urlretrieve(url, "config.py")

# Le noyau est Python même dans le parcours C++ : `mm` (morph.py) est utilisé par les
# simulateurs, par l'affichage des figures que le binaire C++ génère et par
# l'état mm::Image entre les cellules. cpp=True télécharge aussi le parcours compilé
# (morph.hpp + stb_image*.h), utilisé dans le #include des cellules %%writefile *.cpp.
import config
config.setup(testsuite=True, cpp=True)
from morph import mm
from testsuite import TestSuite
```

```
Environnement prêt. Morph : 1.1.9 | OpenCV : 5.0.0 | TestSuite : 1.1.2
```

Exécution des tests

Pour évaluer les tests, exécutez `TestSuite("EP04_01.extensão").run()` dans une nouvelle cellule, en remplaçant l'extension par celle du langage utilisé (`.py`, `.java`, `.c`, `.cpp`, `.js` ou `.r`). Le système télécharge les cas de test depuis GitHub, exécute le programme et calcule automatiquement la note.

Pour tester directement du code Python, sans enregistrer de fichier, utilisez `run_code(codigo)` en passant le code comme *chaîne de caractères* dans une variable `codigo` :

```
codigo = """
from morph import mm
# ... votre code ici ...
```

```
"""
TestSuite("EP04_01").run_code(codigo)
```

2.12.1 EP02_01 ⊙ Réglage de la luminosité et du contraste

Dans cette activité, l'objectif est d'implémenter un opérateur ponctuel pour la **transformation linéaire d'intensité**, en appliquant le réglage dynamique de la luminosité et du contraste sur une image numérique.

2.12.1.1 📋 Directives d'implémentation

L'algorithme doit suivre le flux d'exécution ci-dessous :

1. **Dimensions :** Lire les entiers L (lignes) et C (colonnes) de la matrice.
2. **Paramètres :** Lire la valeur réelle α (facteur de contraste) et l'entier β (facteur de luminosité).
3. **Données :** Lire les valeurs entières de la matrice d'origine.
4. **Mappage :** Pour chaque pixel p, calculer la nouvelle valeur p' à l'aide de l'équation :

$$p' = \text{clip}(\text{round}(\alpha \cdot p + \beta))$$

5. **Sortie :** Afficher la matrice résultante avec les dimensions $L \times C$.

2.12.1.2 📌 Contraintes informatiques

- **Arrondi (*Round*) :** On applique l'arrondi mathématique à l'entier le plus proche avant la conversion de type.
- **Saturation (*Clipping*) :** Les valeurs doivent être confinées à l'intervalle $[0, 255]$ pour préserver le standard 8 bits :

$$\text{clip}(x) = \max(0, \min(255, x))$$

- **Simulation :** L'effet des paramètres α et β sur la correction histogrammique peut être observé dans la Figure 2.12.

2.12.1.3 🧠 Fondements théoriques

Les modifications altèrent l'histogramme de l'image pour ajuster le profil d'éclairage et la distinction tonale.

Paramètre	Fonction	Impact visuel
α (***Alpha***)	Scalaire	Module le **Contraste**. Si $\alpha > 1$, il étend l'histogramme ; si $0 \leq \alpha < 1$, il le compresse.
β (***Beta***)	Additif	Module la **Luminosité**. S'il est positif, il translate l'histogramme vers la droite ; s'il est négatif, vers la gauche.
clip	Limiteur	Restreint la plage dynamique, empêchant les erreurs de *sous-dépassement* et de *dépassement*.

2.12.1.4 📦 Spécification d'entrée et de sortie (VPL)

Entrée :

- Ligne 1 : Entier L.

- Ligne 2 : Entier C.
- Ligne 3 : Valeurs de `alpha` (α) et `beta` (β).
- Lignes suivantes : Éléments numériques de la matrice d'origine.

Sortie :

- Matrice transformée structurée en L lignes et C colonnes.

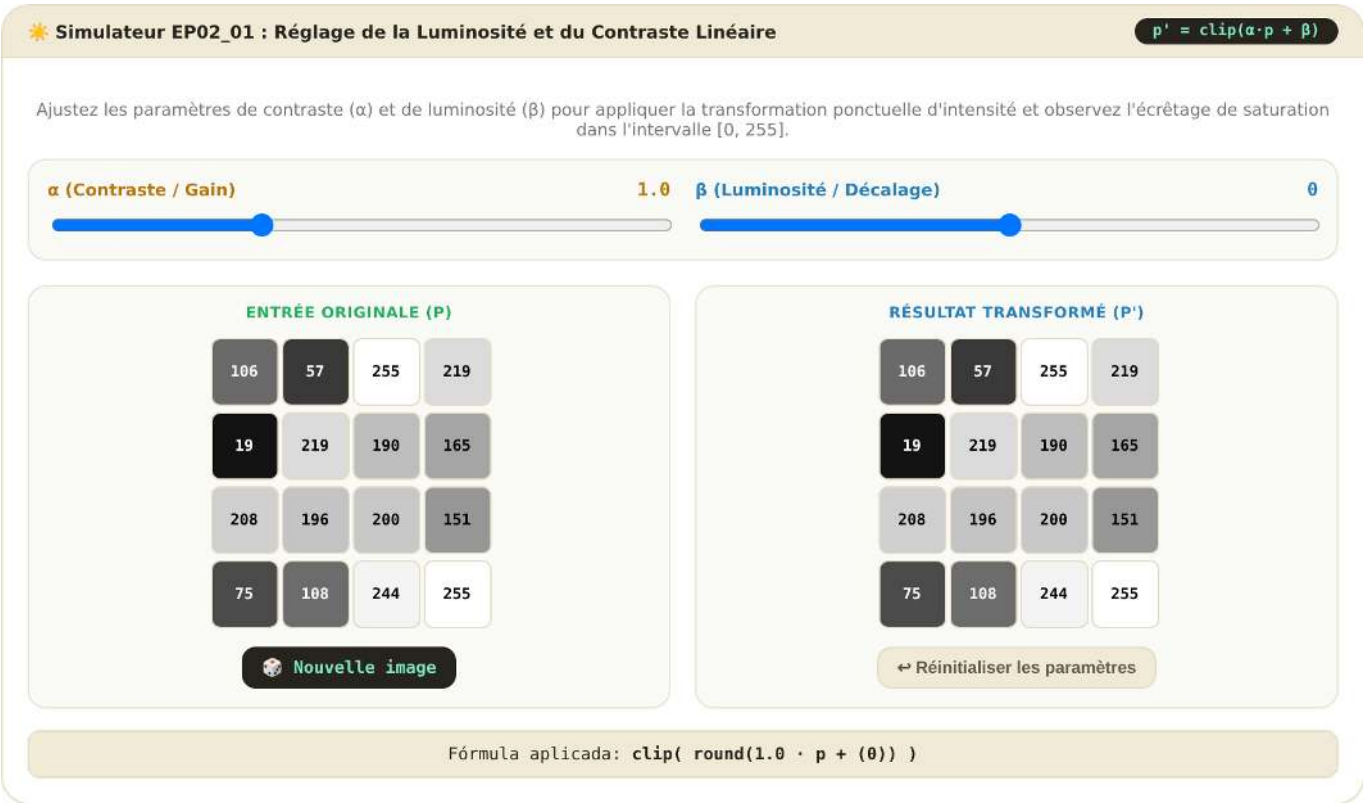
2.12.1.5 📌 Exemples

Le tableau suivant présente un exemple pratique du comportement attendu de l'algorithme, mettant en évidence l'action des opérateurs d'arrondi et de saturation.

Entrée	Sortie	Observation
1 4 1.5 -30 0 100 180 255	0 120 240 255	Notez l'effet de saturation sur le dernier pixel

Exécution des Tests

Pour évaluer les tests, exécutez `TestSuite("EP02_01.extension").run()` dans une nouvelle cellule, en remplaçant l'extension par celle du langage utilisé (`.py`, `.java`, `.c`, `.cpp`, `.js` ou `.r`). Le système télécharge les cas de test depuis GitHub, exécute le programme et calcule la note automatiquement.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.fr/cap02/sim-ep0201-brilho-contraste.html

Figure 2.12: Simulateur EP02_01 : Ajustement de la Luminosité et du Contraste Linéaire (p' = p +)

```
%%writefile EP02_01.cpp
// your solution
```

```
Overwriting EP02_01.cpp
```

```
TestSuite("EP02_01.cpp").run()
```

```
<IPython.core.display.HTML object>
```

2.12.2 EP02_02 🔬 Sous-échantillonnage spatial

Dans cette activité, vous devez implémenter la réduction de la résolution spatiale d'une image par le processus de sous-échantillonnage.

- Lisez deux entiers **L** et **C**, représentant les dimensions de la matrice d'origine.
- Lisez une valeur entière f ($f \geq 1$), qui représente le facteur d'échantillonnage.
- Lisez les valeurs entières de la matrice d'origine.
- La nouvelle image doit être construite en sélectionnant le pixel de la position $(f \cdot i, f \cdot j)$ de l'image d'origine.
- Imprimez la matrice résultante avec les nouvelles dimensions.
- Voir dans Figure 2.13 une simulation de cet EP.

📌 **Important** :

- **Dimensions finales :** L'image échantillonnée aura pour dimensions $\lceil L/f \rceil \times \lceil C/f \rceil$. Dans le contexte de la programmation, cela équivaut à la taille résultante d'un découpage (slicing) avec un pas f.
- **Implémentation :** N'utilisez pas de fonctions toutes faites issues de bibliothèques de traitement d'images (comme OpenCV ou PIL) pour le redimensionnement. Implémentez la logique de sélection des pixels manuellement ou via le découpage de matrices.
- **Aliasing :** Notez que ce processus peut provoquer l'effet d'*aliasing* (crénelage), où les détails fins sont perdus ou des motifs indésirables apparaissent.

2.12.2.1 🧠 Discrétisation de l'espace

Le sous-échantillonnage réduit la résolution spatiale d'une image en ne sélectionnant qu'un pixel tous les f pixels dans chaque direction. C'est le processus inverse de l'interpolation :

Paramètre	Fonction	Effet
Facteur f	Saut d'échantillonnage	Définit l'intervalle de sélection. Un facteur 2 réduit la largeur et la hauteur de moitié.
Résolution	Densité de pixels	Diminue la quantité totale d'informations spatiales de l'image.
Aliasing	Effet secondaire	Apparition de motifs en escalier ou en blocs en raison de la perte de détails fins.

2.12.2.2 📋 Tâche (spécification pour VPL)

Entrée :

La première ligne contient `L`.

La deuxième ligne contient `C`.

La troisième ligne contient le facteur `f`.

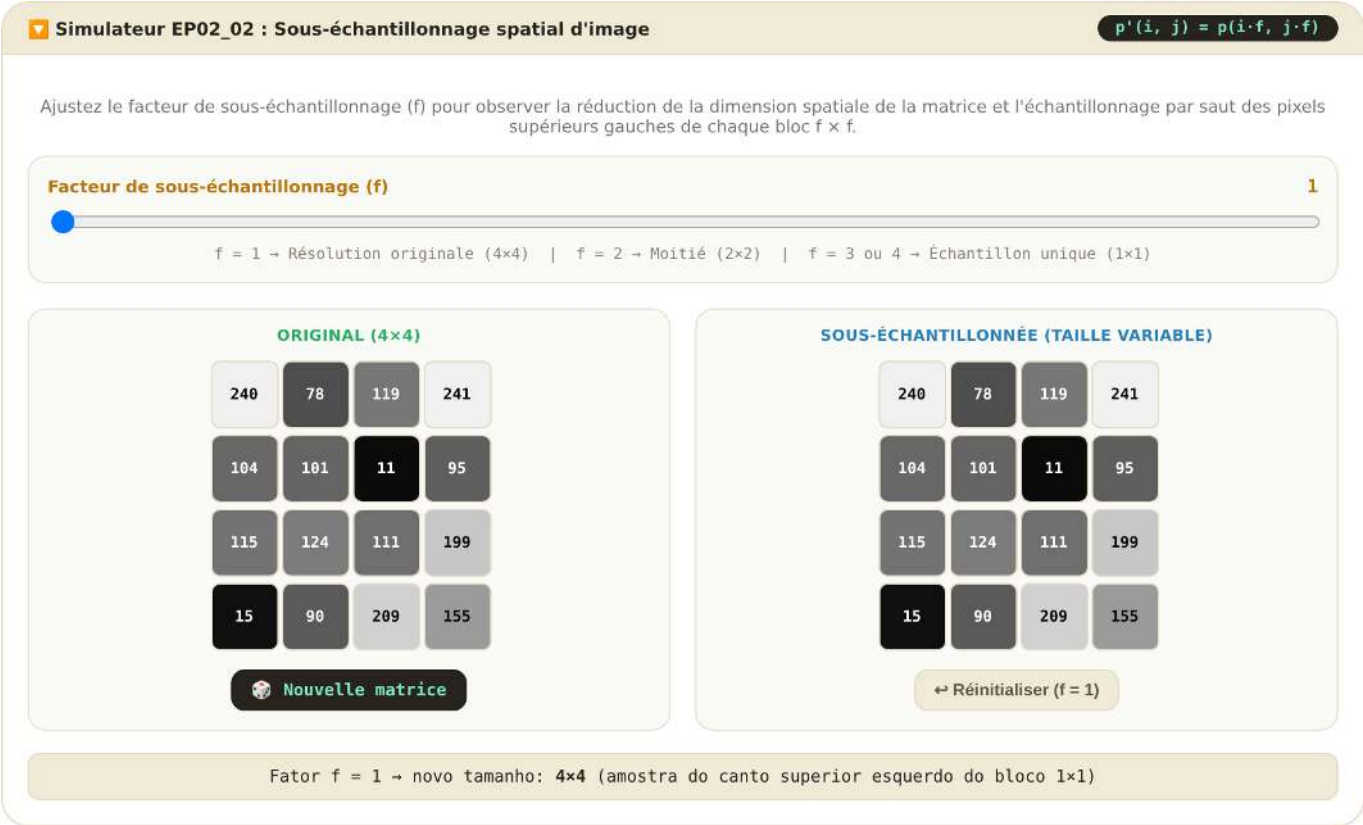
Les lignes suivantes contiennent les éléments de la matrice $L \times C$.

Sortie :

La matrice réduite avec les dimensions correspondant au découpage par `f`.

2.12.2.3 📌 Exemples

Entrée	Sortie	Observation
2 4 2 10 20 30 40 50 60 70 80	10 30	Le facteur 2 sélectionne les pixels (0,0) et (0,2) de la première ligne. La deuxième ligne est ignorée.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.fr/cap02/sim-ep0202-subamostragem.html

Figure 2.13: Simulateur EP02_02: Sous-échantillonnage spatial (réduction de résolution par saut f)

```
%%writefile EP02_02.cpp
// your solution
```

```
Overwriting EP02_02.cpp
```

```
TestSuite("EP02_02.cpp").run()
```

```
<IPython.core.display.HTML object>
```

2.12.3 EP02_03 🎨 Quantification des niveaux de gris

Dans cette activité, vous devez implémenter la quantification uniforme d'une image, en réduisant le nombre de niveaux d'intensité de gris originaux vers une nouvelle échelle basée sur un nombre de bits plus petit.

- Lisez deux entiers **L** et **C**, représentant les dimensions de la matrice.
- Lisez un entier k ($1 \leq k \leq 8$), représentant le nouveau nombre de bits de l'image.
- Calculez le nombre de niveaux ($N = 2^k$) et la taille de l'intervalle (pas).

- Pour chaque pixel p, calculez la nouvelle valeur p' en le mappant vers l'indice du niveau discrétisé correspondant (variant de 0 à $2^k - 1$).
- Affichez la matrice résultante avec les mêmes valeurs de dimensions que l'originale.
- Consultez Figure 2.14 pour une simulation de cet EP.

📌 **Important** :

- **Postérisation :** En réduisant drastiquement les niveaux (ex : $k = 2$), vous remarquerez que les dégradés lisses se transforment en bandes abruptes de couleur en raison de la perte de résolution d'amplitude.
- **Calcul du pas :** L'intervalle entre chaque niveau est défini par $pas = 256/2^k$.
- **Mappage :** La méthode de quantification uniforme par troncature qui mappe le pixel vers l'indice de son niveau discrétisé respectif est donnée par :

$$p' = \left\lfloor \frac{p}{pas} \right\rfloor$$

En termes d'implémentation (comme en Python), cela équivaut à la division entière : `p' = p // pas`.

2.12.3.1 🧠 Discrétisation de l'amplitude

Alors que le sous-échantillonnage traite de la résolution spatiale, la quantification se concentre sur la précision de la couleur (amplitude). Réduire les bits signifie simplifier l'information chromatique :

Paramètre	Fonction	Effet
Bits (k)	Profondeur de couleur	Définit combien de tons différents l'image peut avoir (2^k).
Pas	Intervalle de ton	Espacement entre les niveaux de gris autorisés.
Postérisation	Phénomène visuel	Transformation de variations continues en blocs de couleur unie.

2.12.3.2 📋 Tâche (spécification pour VPL)

Entrée :

La première ligne contient `L`.

La deuxième ligne contient `C`.

La troisième ligne contient le nombre de bits `k`.

Les lignes suivantes contiennent les éléments de la matrice $L \times C$.

Sortie :

La matrice transformée avec les indices des niveaux quantifiés, en conservant la taille originale $L \times C$.

2.12.3.3 📌 Exemples

Entrée	Sortie	Observation
1 4 2 0 80 170 255	0 1 2 3	Avec $k = 2$, nous avons $2^2 = 4$ niveaux discrets disponibles $(0, 1, 2, 3)$. Le pas est de $256/4 = 64$. En appliquant la division entière élément par élément : $0//64 = 0$, $80//64 = 1$, $170//64 = 2$, $255//64 = 3$.
1 5 1 10 50 120 200 250	0 0 0 1 1	Avec $k = 1$, nous avons $2^1 = 2$ niveaux (0 et 1). Pas $= 256/2 = 128$. Les pixels inférieurs à 128 donnent 0, et les pixels supérieurs ou égaux à 128 donnent 1.

Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.fr/cap02/sim-ep0203-quantizacao.html

Figure 2.14: Simulateur EP02_03 : Quantification et Profondeur de Bits (Réduction du Nombre de Niveaux de Gris)

```
%%writefile EP02_03.cpp
// your solution
```

```
Overwriting EP02_03.cpp
```

```
TestSuite("EP02_03.cpp").run()
```

```
<IPython.core.display.HTML object>
```

2.12.4 EP02_04 📐 Transformée de distance sur image binaire

Étant donné une image binaire où les pixels de valeur **1** représentent l'*objet* et les pixels **0** représentent le *fond*, la distance d'un pixel de fond reçoit la **distance minimale au pixel d'objet le plus proche**. Les pixels d'objet reçoivent une distance **0**. Pour simplifier, considérez que l'image ne contient **qu'un seul objet avec un seul pixel de valeur 1**.

Problème : Lire une image binaire $L \times C$ et une métrique, puis calculer cette distance simplifiée en appliquant l'une des trois formules :

$$d_{\text{Euclidienne}} = \sqrt{(\Delta r)^2 + (\Delta c)^2}$$

$$d_{\text{City-block}} = |\Delta r| + |\Delta c|$$

$$d_{\text{Échiquier}} = \max(|\Delta r|,\ |\Delta c|)$$

où Δr est la différence de lignes et Δc la différence de colonnes entre deux pixels.

2.12.4.1 🖼️ Pourquoi cela importe-t-il ? - Applications de la TD

La Transformée de distance (TD) apparaît dans des dizaines de pipelines de vision par ordinateur :

Métrique	Complexité	Application typique
Euclidienne	🔴 $O(n^2)$ naïve	Squelettisation, correspondance de formes
City-block	🟡 $O(n)$ avec 2 passes	Morphologie, dilatation/érosion
Échiquier	🟢 $O(n)$ avec 2 passes	Morphologie, dilatation/érosion

2.12.4.2 📌 Exigences techniques

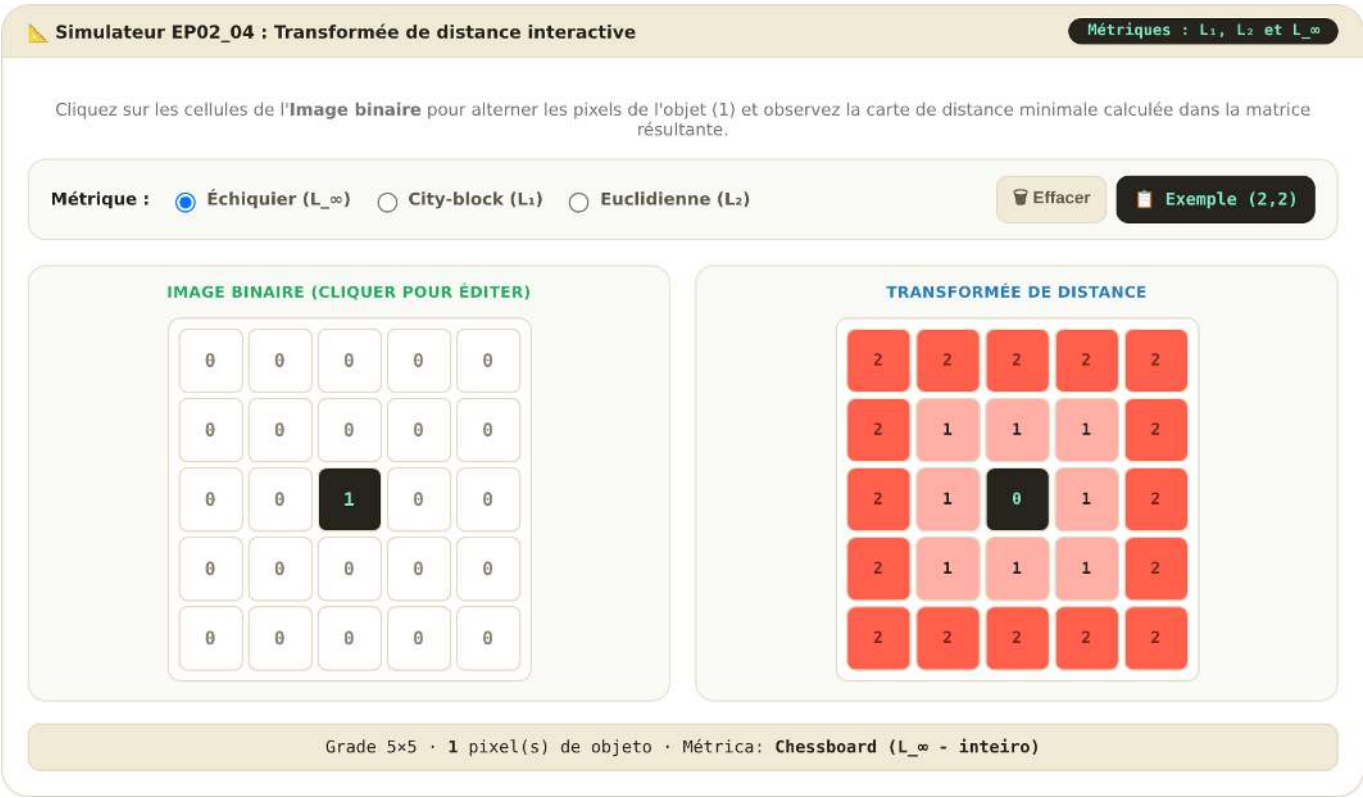
- **Entrée :** * Première ligne : L et C (entiers).
  - Deuxième ligne : nom de la métrique (`euclidean`, `cityblock` ou `chessboard`).
  - Ensuite, la matrice binaire $L \times C$ (valeurs 0 ou 1).
- **Pixels d'objet (1) :** distance = 0 (ou 0.00 pour euclidienne).
- **Pixels de fond (0) :** distance au seul pixel d'objet de l'image.
- **Arrondi (euclidienne) :** imprimer avec **2 décimales** (format `:.2f`). City-block et Échiquier produisent des entiers — imprimer sans décimales.
- **Sortie :** valeurs séparées par des espaces, une ligne par ligne de la matrice.
- Voir dans Figure 2.15 une simulation de cet EP.

2.12.4.3 📌 Exemples

Entrée	Sortie	Observation
4 4 chessboard 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0	2 2 2 2 2 1 1 1 2 1 0 1 2 1 1 1	La distance Échiquier est $\max(\|dx\|, \|dy\|)$. Le seul pixel objet est $(2, 2) = 0$; les autres stockent leur distance minimale jusqu'à lui.

2.12.4.4 📌 Observations finales

- Comme l'image ne contient **qu'un seul objet d'un pixel**, la distance de chaque pixel de fond est simplement la distance de ce pixel au seul point objet.
- L'implémentation peut utiliser la force brute (parcourir tous les pixels de l'image et calculer la distance directement), car L et C sont petits dans les cas de test.
- Ce problème est un échauffement pour la Transformée de distance générale, qui sera travaillée dans des chapitres ultérieurs avec plusieurs objets et des algorithmes optimisés.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.fr/cap02/sim-ep0204-distancia.html

Figure 2.15: Simulateur EP02_04 : Transformée de distance en image binaire (Chessboard, City-block et Euclidienne)

```
%%writefile EP02_04.cpp
// your solution
```

```
Overwriting EP02_04.cpp
```

```
TestSuite("EP02_04.cpp").run()
```

```
<IPython.core.display.HTML object>
```

2.12.5 EP02_05 Translation d'image

Dans cette activité, vous devez implémenter le déplacement spatial d'une image. La translation déplace chaque pixel de l'image originale vers une nouvelle position en fonction d'un vecteur de déplacement.

- Lisez deux entiers **L** et **C**, représentant les dimensions de la matrice.
- Lisez deux entiers t_x (déplacement horizontal) et t_y (déplacement vertical).
- Lisez les valeurs entières de la matrice originale.
- Calculez la nouvelle position (x', y') pour chaque pixel (x, y) original.
- Affichez la matrice résultante avec les mêmes dimensions que l'originale.
- Voir dans Figure 2.16 une simulation de cet EP.

📌 **Important** :

- **Remplissage :** Les pixels qui « entrent » dans l'image en raison du déplacement et qui n'ont pas de correspondant dans l'originale doivent être remplis avec **0** (noir).
- **Suppression :** Les pixels qui, après la translation, sortent des limites de la matrice ($0 \dots L-1$ ou $0 \dots C-1$) doivent être ignorés.
- **Coordonnées :** Considérez x comme l'indice de la ligne et y comme l'indice de la colonne.

2.12.5.1 🧠 Déplacement spatial

Translater une image signifie déplacer tous ses points d'une distance fixe dans des directions spécifiées. Mathématiquement, en utilisant les coordonnées homogènes, l'opération est décrite comme suit :

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Ce qui donne les équations simples :

- $x' = x + t_x$
- $y' = y + t_y$

2.12.5.2 📋 Tâche (spécification pour VPL)

Entrée :

La première ligne contient `L`.

La deuxième ligne contient `C`.

La troisième ligne contient les entiers `tx` et `ty`.

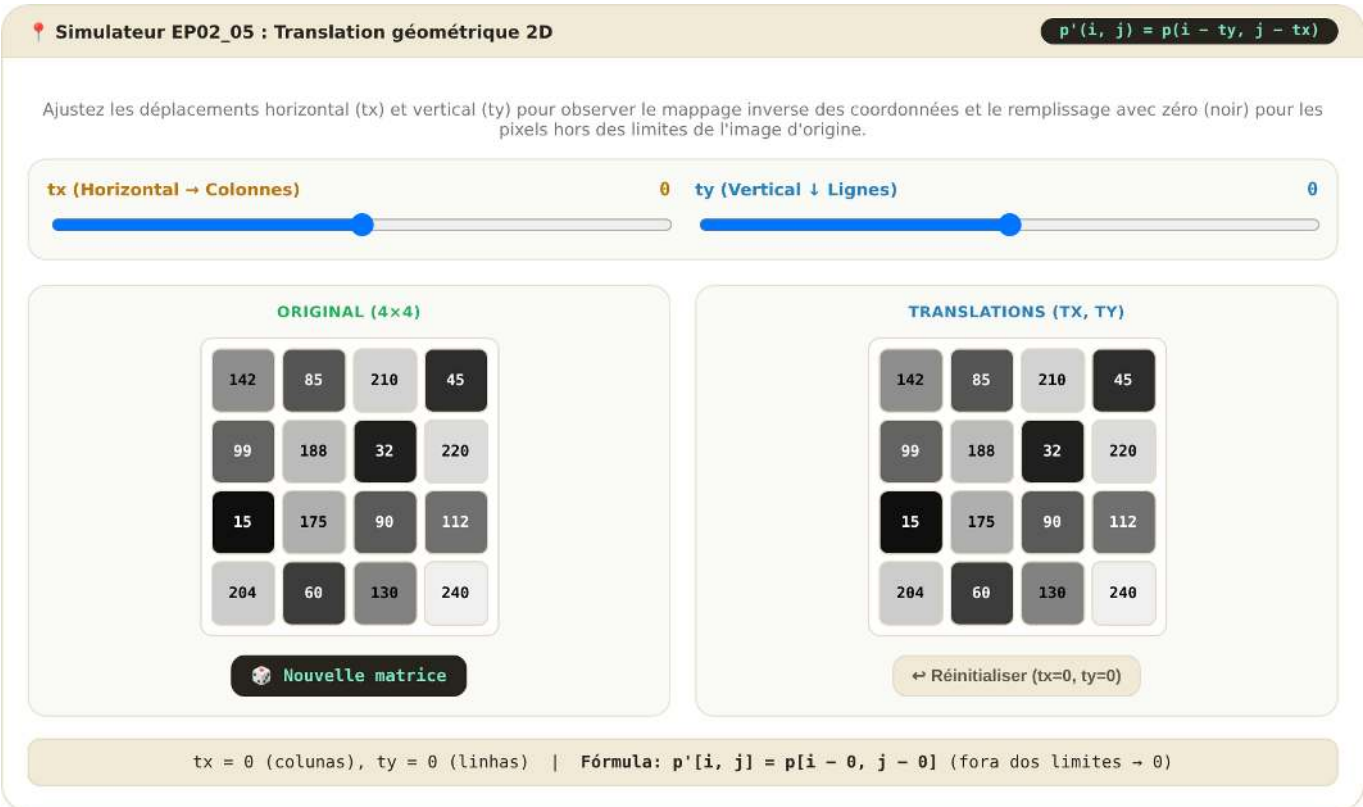
Les lignes suivantes contiennent les éléments de la matrice $L \times C$.

Sortie :

La matrice résultante avec les mêmes dimensions $L \times C$ après le déplacement.

2.12.5.3 📌 Exemples

Entrée	Sortie	Observation
2 2 1 1 10 20 30 40	0 0 0 10	**Déplacement** $(t_x = 1, t_y = 1)$**:** Chaque pixel se déplace d'une position vers la droite (horizontal) et d'une vers le bas (vertical). Le pixel $(0, 0) = 10$ va vers la destination $(1, 1)$ (coin inférieur droit). Les positions vides sont remplies avec 0.
3 3 -1 0 1 2 3 4 5 6 7 8 9	2 3 0 5 6 0 8 9 0	**Déplacement** $(t_x = -1, t_y = 0)$**:** Chaque pixel se déplace d'une position vers la gauche (horizontal). La première colonne originale (`1`, `4`, `7`) est supprimée, les autres colonnes se déplacent vers la gauche, et la dernière colonne résultante est remplie de zéros (0).



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.fr/cap02/sim-ep0205-translacao.html

Figure 2.16: Simulateur EP02_05 : Translation géométrique d'image (Déplacement tx et ty avec remplissage de bordure)

```
%%writefile EP02_05.cpp
// your solution
```

```
Overwriting EP02_05.cpp
```

```
TestSuite("EP02_05.cpp").run()
```

```
<IPython.core.display.HTML object>
```

2.12.6 EP02_06 🔄 Rotation d'image

Dans cette activité, vous devez implémenter la rotation d'une image autour de son centre géométrique. Cette opération nécessite le mappage des coordonnées et l'utilisation de techniques d'interpolation pour déterminer les nouvelles valeurs des pixels.

- Lisez deux entiers **L** et **C**, représentant les dimensions de la matrice.
- Lisez une valeur réelle θ (angle en degrés) et une *chaîne de caractères* représentant la **méthode** d'interpolation (`nearest` ou `bilinear`).
- Lisez les valeurs entières de la matrice originale.
- Effectuez la rotation autour du centre de l'image $(L/2, C/2)$.
- Affichez la matrice résultante avec les mêmes dimensions que l'originale.
- Voir dans Figure 2.17 une simulation de cet EP.

📌 **Important** :

- **Mappage inverse :** Pour éviter les « trous » dans l'image finale, parcourez chaque pixel (x', y') de l'image de destination et calculez sa position correspondante (x, y) dans l'image originale en utilisant la matrice de rotation inverse.
- **Interpolation :**
- `nearest` : Attribue la valeur du pixel le plus proche de la coordonnée calculée.
- `bilinear` : Calcule une moyenne pondérée basée sur les 4 voisins les plus proches.
- **Bords :** Les pixels dont l'origine (x, y) tombe en dehors des limites de l'image originale doivent être remplis avec **0**.

2.12.6.1 🧠 Transformation par angle

La rotation d'un point (x, y) par rapport à l'origine d'un angle θ est donnée par la matrice de transformation. Pour effectuer une rotation autour d'un centre (x_c, y_c), on translate d'abord le centre vers l'origine, on effectue la rotation, puis on translate en retour :

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & x_c \\ \sin\theta & \cos\theta & y_c \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x - x_c \\ y - y_c \\ 1 \end{bmatrix}$$

Conseil : Utilisez le mappage inverse pour garantir que tous les pixels de l'image de sortie soient correctement remplis.

2.12.6.2 📋 Tâche (spécification pour VPL)

Entrée :

La première ligne contient `L`.

La deuxième ligne contient `C`.

La troisième ligne contient l'angle `theta` (en degrés) et la méthode `interp` (`nearest` ou `bilinear`).

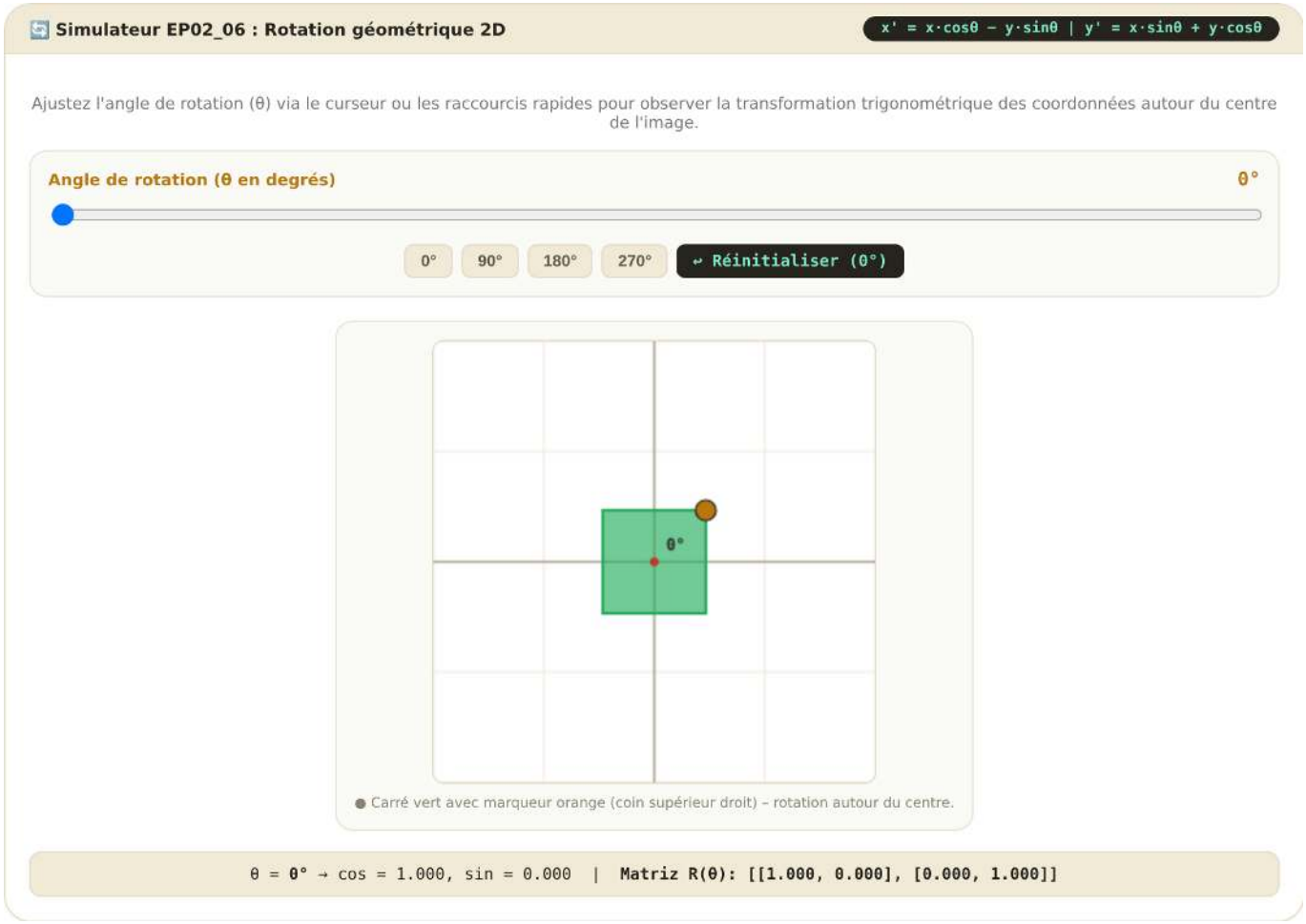
Les lignes suivantes contiennent les éléments de la matrice $L \times C$.

Sortie :

La matrice tournée avec `L` lignes et `C` colonnes.

2.12.6.3 📌 Exemples

Entrée	Sortie	Observation
2 2 90 nearest 1 2 3 4	3 1 4 2	Rotation de 90° dans le sens horaire : la colonne 0 devient la ligne 0 (de bas en haut). $(0,0) = 1 \to (1,0)$, $(1,0) = 3 \to (0,0)$, $(0,1) = 2 \to (1,1)$, $(1,1) = 4 \to (0,1)$.
3 3 45 bilinear 0 0 0 0 255 0 0 0 0	0 180 0 180 255 180 0 180 0	Rotation de 45° : le pixel central reste 255 ; les voisins directs reçoivent une valeur interpolée ≈ 180 par bilinéaire ; les coins restent à 0.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.fr/cap02/sim-ep0206-rotacao.html

Figure 2.17: Simulateur EP02_06 : Rotation d'image autour de l'origine par un angle

```
%%writefile EP02_06.cpp
// your solution
```

```
Overwriting EP02_06.cpp
```

```
TestSuite("EP02_06.cpp").run()
```

```
<IPython.core.display.HTML object>
```

2.12.7 EP02_07 🔍 Redimensionnement (Échelle)

Dans cette activité, vous devez implémenter le redimensionnement d'une image à l'aide de facteurs d'échelle. Contrairement au sous-échantillonnage simple, nous utiliserons ici des techniques d'interpolation pour permettre à la fois l'agrandissement et la réduction de l'image.

- Lisez deux entiers **L** et **C**, représentant les dimensions de la matrice originale.
- Lisez deux valeurs réelles s_x (échelle sur les lignes) et s_y (échelle sur les colonnes).
- Lisez une *chaîne de caractères* représentant la **méthode** d'interpolation (`nearest` ou `bilinear`).
- Lisez les valeurs entières de la matrice originale.
- Calculez les nouvelles dimensions : $L' = \text{round}(L \times s_x)$ et $C' = \text{round}(C \times s_y)$.
- Affichez la matrice résultante avec les nouvelles dimensions.
- Voir dans Figure 2.18 une simulation de cet EP.

📌 **Important** :

- **Mappage inverse :** Pour chaque pixel (x', y') de l'image de destination, trouvez la position correspondante dans l'origine en utilisant $(x, y) = (x'/s_x, y'/s_y)$.
- **Interpolation :**
- `nearest` : Sélectionne la valeur du pixel le plus proche (arrondi des coordonnées).
- `bilinear` : Effectue une interpolation linéaire double entre les quatre pixels voisins les plus proches dans l'image originale.
- **Bords :** Assurez-vous que le mappage ne tente pas d'accéder à des indices hors de l'intervalle $[0, L-1]$ et $[0, C-1]$.

2.12.7.1 🧠 Interpolation pour Agrandissement/Réduction

Redimensionner une image par des facteurs (s_x, s_y) exige le remplissage des vides (lors de l'agrandissement) ou la fusion d'informations (lors de la réduction). La méthode d'interpolation définit la qualité visuelle du résultat :

Méthode	Fonctionnement	Effet visuel
Nearest	Prend la valeur du voisin le plus proche.	Rapide, mais génère un effet "pixelisé" ou des blocs.
Bilinear	Moyenne pondérée des 4 voisins (2×2).	Adoucit l'image, réduisant le crénelage.

2.12.7.2 📋 Tâche (spécification pour VPL)

Entrée :

La première ligne contient `L`.

La deuxième ligne contient `C`.

La troisième ligne contient les facteurs `sx` et `sy`.

La quatrième ligne contient la méthode `interp` (`nearest` ou `bilinear`).

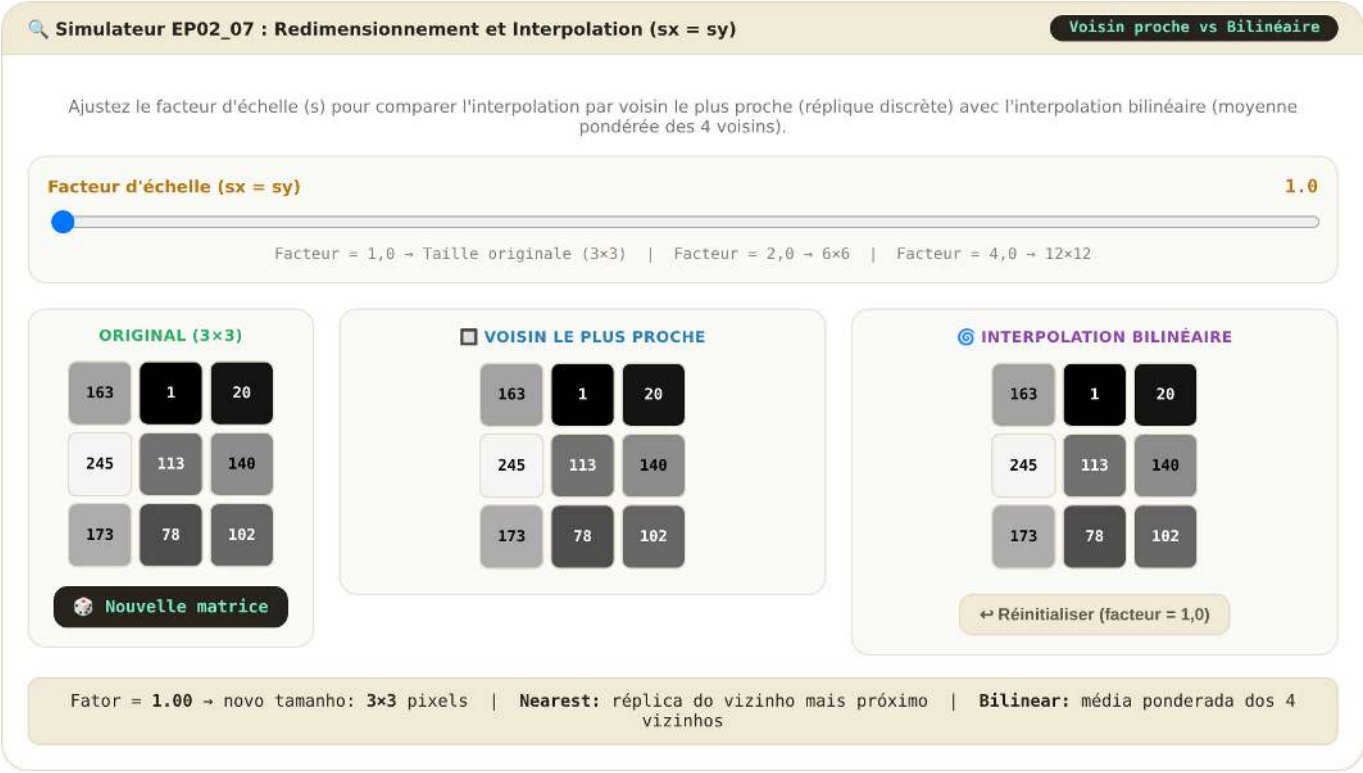
Les lignes suivantes contiennent les éléments de la matrice $L \times C$.

Sortie :

La matrice redimensionnée avec les dimensions $L' \times C'$.

2.12.7.3 📌 Exemples

Entrée	Sortie	Observation
2 2 2.0 2.0 nearest 1 2 3 4	1 1 2 2 1 1 2 2 3 3 4 4 3 3 4 4	Agrandissement 2× : chaque pixel original est répliqué dans un bloc 2×2. L'image 2×2 devient 4×4.
2 2 0.5 0.5 nearest 10 20 30 40	10	Réduction 0.5× : l'image 2×2 devient 1×1. Avec nearest, le seul pixel de sortie échantillonne la position $(0, 0) = 10$.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.fr/cap02/sim-ep0207-escala.html

Figure 2.18: Simulateur EP02_07 : Redimensionnement spatial et interpolation (Nearest Neighbor vs Bilinéaire)

```
# Not yet ported to this language in this version - conceptual reference in Python.
%%writefile EP02_07.py
# Code Python
import numpy as np
from morph import mm

# 1. Lecture des dimensions, facteurs et méthode
l = int(input())
c = int(input())
```

```
sx, sy = map(float, input().split())
interp = input().strip()

# 2. Lecture de l'image originale
img = mm.readImg(l, c)

# 3. Nouvelles dimensions
l_new = round(l * sx)
c_new = round(c * sy)

# 4. Redimensionnement avec mm.resize
# cv2.resize utilise (largeur, hauteur) = (colonnes, lignes)
resultado = mm.resize(img, (c_new, l_new), method=interp)

# 5. Affichage
print(mm.drawImg(resultado))
```

```
TestSuite("EP02_07.cpp").run()
```

```
<IPython.core.display.HTML object>
```

2.12.8 EP02_08 🔀 Cisaillement (Shear)

Dans cette activité, vous devez implémenter la transformation de cisaillement sur une image. Le cisaillement est une transformation affine qui décale chaque point dans une direction fixe, d'une valeur proportionnelle à sa distance par rapport à une droite parallèle à cette direction, produisant un effet d'inclinaison.

- Lire deux entiers **L** et **C**, représentant les dimensions de la matrice.
- Lire deux valeurs réelles sh_x (cisaillement horizontal) et sh_y (cisaillement vertical).
- Lire une *chaîne de caractères* représentant la **méthode** d'interpolation (`nearest` ou `bilinear`).
- Lire les valeurs entières de la matrice originale.
- Appliquer la transformation en conservant la taille originale de l'image (en coupant ce qui dépasse les limites).
- Afficher la matrice résultante avec les dimensions $L \times C$.
- Voir dans Figure 2.19 une simulation de cet EP.

📌 **Important** :

- **Mappage inverse :** Pour chaque pixel (x', y') de l'image de destination, calculer la position correspondante dans l'origine (x, y) en utilisant la matrice de cisaillement inverse.
- **Remplissage :** Les coordonnées qui aboutissent à des positions hors de la matrice originale doivent être remplies avec **0**.
- **Coordonnées :** Pour les besoins de cette implémentation, considérez x comme l'indice de ligne et y comme l'indice de colonne.

2.12.8.1 🧠 Distorsion affine

Le cisaillement modifie la géométrie de l'image en inclinant ses axes. La relation entre les coordonnées originales (x, y) et les coordonnées transformées (x', y') est donnée par :

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & sh_x & 0 \\ sh_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Cela donne les équations :

- $x' = x + sh_x \cdot y$
- $y' = y + sh_y \cdot x$

2.12.8.2 📋 Tâche (spécification pour VPL)

Entrée :

La première ligne contient `L`.

La deuxième ligne contient `C`.

La troisième ligne contient les facteurs `shx` et `shy`.

La quatrième ligne contient la méthode `interp` (`nearest` ou `bilinear`).

Les lignes suivantes contiennent les éléments de la matrice $L \times C$.

Sortie :

La matrice transformée avec les mêmes dimensions $L \times C$.

2.12.8.3 📌 Exemples

Entrée	Sortie	Observation
3 3 0.5 0.0 nearest 10 20 30 40 50 60 70 80 90	10 20 30 0 40 50 0 0 70	Cisaillement horizontal : la ligne i se décale de $\lfloor i \cdot 0.5 \rfloor$ pixels. Ligne $0 \to 0$px, ligne $1 \to 0$px, ligne $2 \to 1$px. Les pixels décalés vers l'extérieur sont éliminés et les positions vides sont remplies avec 0.
2 2 0.0 1.0 nearest 10 20 30 40	10 0 30 20	Cisaillement vertical : la colonne j se décale de $\lfloor j \cdot 1.0 \rfloor$ pixels vers le bas. Colonne $0 \to 0$px (inchangée), colonne $1 \to 1$px : 20 descend à $(1, 1)$ et $(0, 1)$ devient 0.

```
%%writefile EP02_08.cpp
// your solution
```

```
Overwriting EP02_08.cpp
```

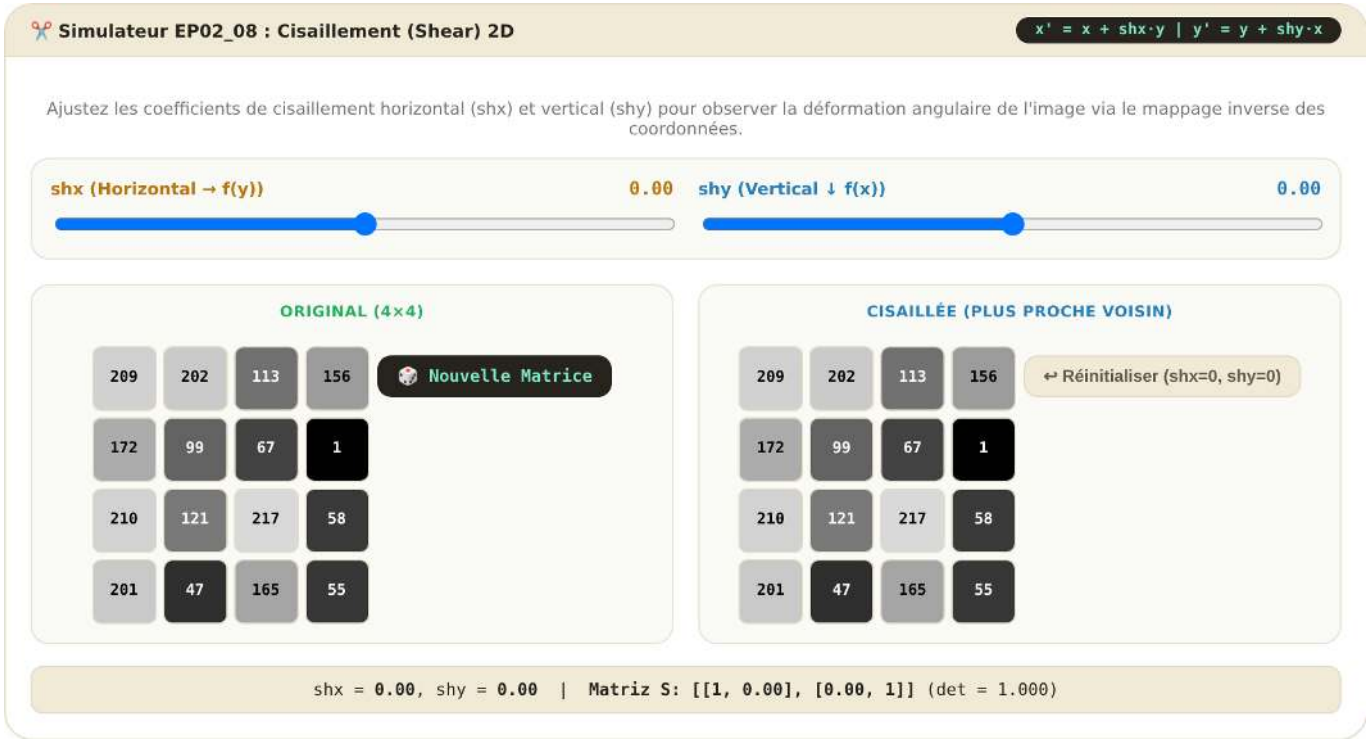
```
TestSuite("EP02_08.cpp").run()
```

```
<IPython.core.display.HTML object>
```

2.12.9 EP02_09 🧩 Transformation Affine Générique

Dans cette activité, vous devez implémenter une transformation affine arbitraire sur une image. Cette opération est la généralisation de toutes les transformations linéaires (mise à l'échelle, rotation, cisaillement) combinées à la translation, permettant des manipulations géométriques complexes via une seule matrice.

- Lisez deux entiers **L** et **C**, représentant les dimensions de la matrice.
- Lisez **six valeurs réelles** (a, b, t_x, c, d, t_y) qui composent la matrice de transformation affine 2×3.
- Lisez une *chaîne de caractères* représentant la **méthode** d'interpolation (`nearest` ou `bilinear`).
- Lisez les valeurs entières de la matrice d'origine.
- Appliquez la transformation en conservant la taille d'origine $L \times C$.
- Affichez la matrice résultante.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.fr/cap02/sim-ep0208-cisalhamento.html

Figure 2.19: Simulateur EP02_08 : Transformation géométrique de cisaillement 2D (Cisaillement horizontal et vertical)

- Voir Figure 2.20 pour une simulation de cet EP.

📌 **Important** :

- **Mappage inverse :** Pour calculer la valeur de chaque pixel dans l'image de destination, vous devez utiliser l'**inverse** de la matrice de transformation affine fournie afin de trouver la coordonnée correspondante dans l'image d'origine.
- **Remplissage :** Les coordonnées calculées qui tombent hors des limites $[0, L-1]$ et $[0, C-1]$ de l'image d'origine doivent donner un pixel de valeur **0**.
- **Flexibilité :** Cette implémentation doit être capable d'effectuer n'importe laquelle des tâches précédentes (translation, rotation, etc.) en modifiant simplement les paramètres de la matrice.

Astuce :

```
flags = cv2.INTER_NEAREST if interp == 'nearest' else \
        cv2.INTER_CUBIC   if interp == 'bicubic'  else \
        cv2.INTER_LANCZOS4 if interp == 'lanczos' else \
        cv2.INTER_LINEAR

r = cv2.warpAffine(img, M, (C, L), flags=flags)
```

2.12.9.1 🧠 Combinaison d'opérations

La transformation affine préserve les points, les droites et les plans. En traitement d'images, elle mappe la position (x, y) vers (x', y') selon le système :

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

Ou, de manière compacte en coordonnées homogènes :

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & t_x \\ c & d & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

2.12.9.2 📋 Tâche (spécification pour VPL)

Entrée :

La première ligne contient `L`.

La deuxième ligne contient `C`.

La troisième ligne contient six flottants : `a b tx c d ty`.

La quatrième ligne contient la méthode `interp` (`nearest` ou `bilinear`).

Les lignes suivantes contiennent les éléments de la matrice $L \times C$.

Sortie :

La matrice transformée avec les dimensions d'origine $L \times C$.

2.12.9.3 📌 Exemples

Entrée	Sortie	Observation
2 2 1.0 0.0 0.5 0.0 1.0 0.5 bilinear 10 20 30 40	15 20 25 30	Translation fractionnaire ($t_x = 0.5, t_y = 0.5$) : chaque pixel de sortie (i, j) échantillonne la position $(i + 0.5,\ j + 0.5)$ de l'entrée via bilinéaire. Ex : $(0, 0)$ interpole les quatre voisins → 15.
3 3 2.0 0.0 0.0 0.0 2.0 0.0 nearest 1 2 3 4 5 6 7 8 9	1 1 2 1 1 2 4 4 5	Mise à l'échelle 2× via matrice affine ($a = 2, d = 2$) : chaque pixel de sortie (i, j) échantillonne la position $(2i, 2j)$ de l'entrée avec le plus proche voisin. Ex : $(0, 2) \rightarrow (0, 4)$ hors de l'image → le plus proche voisin découpe vers $(0, 2) = 3$… en attente de confirmation de la logique de bord.

```
%%writefile EP02_09.cpp
// your solution
```

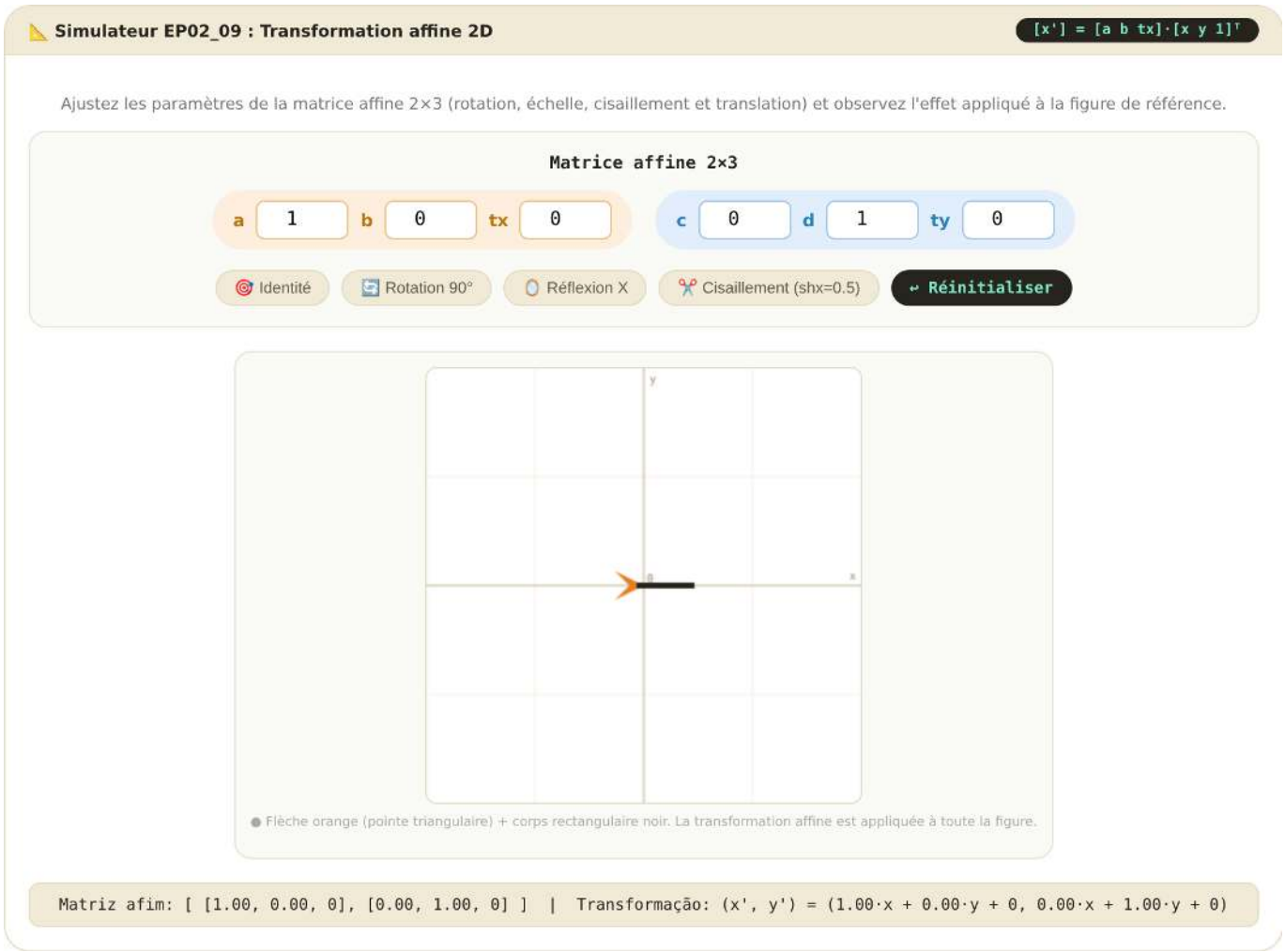
```
Overwriting EP02_09.cpp
```

```
TestSuite("EP02_09.cpp").run()
```

```
<IPython.core.display.HTML object>
```

2.12.10 EP02_10 🎯 Correction de perspective (homographie)

Dans cette activité, vous devez implémenter la transformation de perspective, également connue sous le nom d'homographie. Contrairement aux transformations affines, la perspective ne préserve pas le parallélisme, ce



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.fr/cap02/sim-ep0209-afim.html

Figure 2.20: Simulador EP02_09 : Transformação Afim 2D (Matriz 2×3)

qui permet de « redresser » des objets inclinés, comme des documents ou des panneaux capturés sous des angles obliques.

- Lisez deux entiers **L** et **C**, représentant les dimensions de la matrice d'origine.
- Lisez **quatre paires de coordonnées** (x, y) représentant les coins du quadrilatère source (objet déformé).
- Lisez **quatre paires de coordonnées** (x, y) représentant les coins du quadrilatère de destination (là où l'objet doit être mappé).
- Lisez les valeurs de la matrice d'origine.
- Calculez la matrice d'homographie 3×3 et appliquez la transformation.
- Affichez la matrice résultante avec les dimensions de sortie spécifiées.
- Voir dans Figure 2.21 une simulation de cet EP.

📌 **Important** :

- **Degrés de liberté :** L'homographie possède 8 degrés de liberté (le neuvième élément de la matrice 3×3 est une constante de normalisation, généralement 1), nécessitant au minimum 4 points correspondants pour être calculée.
- **Projection :** Après avoir multiplié les coordonnées par la matrice, il est nécessaire de diviser les résultats x' et y' par la composante homogène w pour revenir au plan 2D.
- **Utilisation de bibliothèques :** Pour cette tâche, vous pouvez utiliser les fonctions `cv2.getPerspectiveTransfo` pour obtenir la matrice et `cv2.warpPerspective` pour appliquer la transformation, ou implémenter le système linéaire et le mappage inverse manuellement pour un défi supplémentaire.

```
# Dimensions de sortie : boîte englobante des points de destination + 1
w = int(max(pts2[:, 0])) + 1; h = int(max(pts2[:, 1])) + 1
# M = cv2.getPerspectiveTransform(pts1, pts2)
# dst = cv2.warpPerspective(img, M, (w, h))
# ou
dst = mm.perspective_transform(img, pts1, pts2, size=(w, h))
```

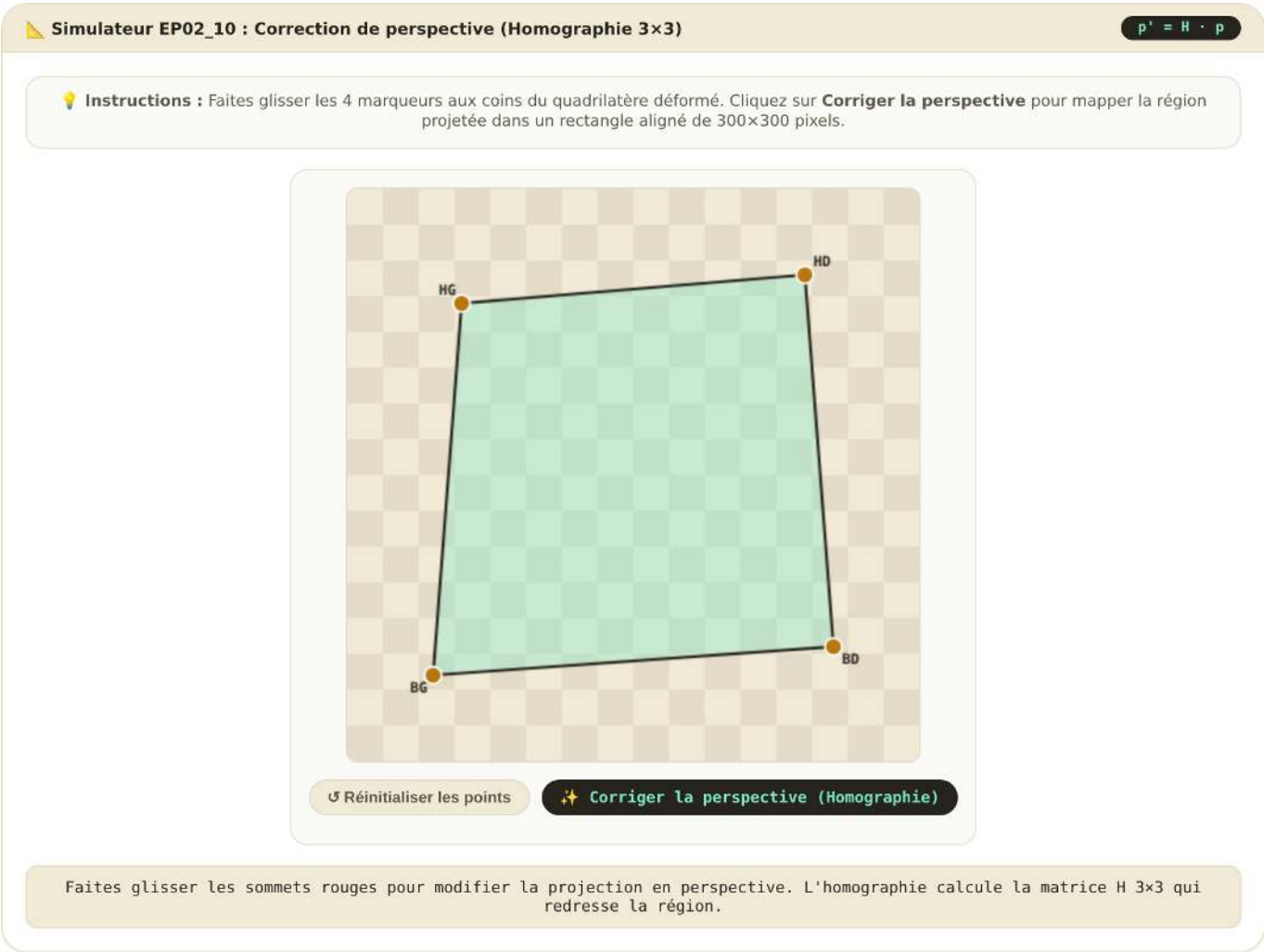
2.12.10.1 🧠 Déformation non affine

Tandis que les transformations affines mappent des parallélogrammes en parallélogrammes, l'homographie mappe tout quadrilatère en un autre quadrilatère. Cela est essentiel pour la vision par ordinateur :

Opération	Caractéristique	Application typique
Homographie	Projection sur un plan	Correction de documents, numérisation de panneaux.
Point de fuite	Convergence de lignes	Reconstruction 3D à partir d'images 2D.
Warping	Déformation de maillage	Stabilisation vidéo et panoramas (stitching).

2.12.10.2 📌 Exemples

Entrée	Sortie	Observation
4 4 0 0 3 0 0 3 3 3 0 0 3 0 0 3 3 3 10 20 30 40 50 60 70 80 90 100 110 120 130 140 150 160	10 20 30 40 50 60 70 80 90 100 110 120 130 140 150 160	Les 4 premières lignes après les dimensions sont les points sources ; les 4 suivantes sont les destinations. Avec des points identiques, la transformation de perspective est l'identité et l'image est préservée.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.fr/cap02/sim-ep0210-perspectiva.html

Figure 2.21: Simulateur EP02_10 : Correction de perspective (Transformation d'homographie 3×3)

```
%%writefile EP02_10.cpp
// your solution
```

```
Overwriting EP02_10.cpp
```

```
TestSuite("EP02_10.cpp").run()
```

```
<IPython.core.display.HTML object>
```

2.12.11 EP02_11 🏆 Correction de Perspective (Homographie) sur une Image Réelle

Dans cette activité, l'objectif est d'appliquer la transformation de perspective (homographie) pour « redresser » un objet incliné sur une photographie réelle. Vous travaillerez avec l'image d'un journal, où la grille d'un jeu de Sudoku est déformée en raison de l'angle sous lequel la photo a été prise.

Votre programme doit lire les paramètres d'entrée depuis le terminal, charger l'image, calculer la matrice d'homographie 3×3, appliquer la transformation géométrique et afficher un indicateur global de validation.

- Lisez deux entiers **L** et **C**, représentant les dimensions en lignes et en colonnes (hauteur et largeur) que l'image redressée de sortie doit avoir.
- Lisez **quatre paires de coordonnées** (x, y) via le terminal, représentant les quatre coins du quadrilatère d'origine (le Sudoku déformé dans l'image originale).
- Calculez automatiquement les **quatre paires de coordonnées de destination** en utilisant les dimensions L et C fournies, en mappant les coins vers les extrémités de la nouvelle image : $(0, 0)$, $(C-1, 0)$, $(0, L-1)$ et $(C-1, L-1)$.
- Chargez l'image locale `sudoku.png` et convertissez-la en niveaux de gris (grayscale).
- Calculez la matrice d'homographie et appliquez la transformation spatiale sur l'image.
- **Sortie :** Calculez et imprimez la **somme de tous les pixels** de l'image résultante.

📌 **Important** :

- **Fichier d'entrée :** L'image `sudoku.png` doit se trouver dans le même dossier que le script. Le programme doit la lire directement depuis le disque (ex : en utilisant `mm::read("sudoku.png")` ou `cv2.imread`).
- **Ordre des points :** Assurez-vous que la lecture des 4 points d'origine et la génération des 4 points de destination suivent rigoureusement le même ordre des coins : Supérieur-Gauche (TL), Supérieur-Droit (TR), Inférieur-Gauche (BL) et Inférieur-Droit (BR).
- **Dimensions dans OpenCV :** N'oubliez pas que des fonctions comme `cv2.warpPerspective` attendent la taille de l'image de sortie au format `(largeur, hauteur)`, ce qui équivaut à `(C, L)`.
- **Interpolation :** Pour garantir la cohérence mathématique de la somme des pixels avec le correcteur automatique, utilisez l'interpolation bilinéaire par défaut (`flags=cv2.INTER_LINEAR`).
- **Crédits :** L'image utilisée est **« Sudoku en periódico »** de Héctor Rodríguez, sous licence **CC BY 2.0**.

2.12.11.1 🧠 Contexte du Problème

L'homographie possède 8 degrés de liberté, nécessitant au minimum 4 correspondances de points pour être calculée. Contrairement aux transformations affines, elle mappe n'importe quel quadrilatère sur un autre quadrilatère, permettant ainsi aux lignes qui convergent vers des points de fuite de redevenir parallèles :

Opération	Caractéristique	Application Typique
Homographie	Projection entre plans	Rectification de documents, numérisation de plaques et de codes QR.
Mappage inverse	Balayage de la destination vers l'origine	Évite les « trous » ou les pixels vides dans l'image finale redressée.
Warping	Rééchantillonnage spatial	Correction de la distorsion des lentilles et assemblage de panoramas (*stitching*).

2.12.11.2 📌 Exemples

Entrée	Sortie	Observation
500 500 100 120 420 95 80 440 450 460	32982820	Les deux premières entrées sont les dimensions de sortie (L et C). Les 4 lignes suivantes sont les coordonnées (x, y) des coins du Sudoku dans l'image originale + PAD. La sortie est la somme totale des pixels de l'image redressée.
200 200 100 120 420 95 80 440 450 460	5277150	Mêmes points d'origine que l'exemple précédent, mais en générant une image de sortie plus petite (200×200). La somme des pixels diminue proportionnellement en raison de l'échelle.

2.12.11.3 Acquisition de l'image du sudoku et conversion en niveaux de gris

La Figure 2.22 montre la lecture de l'image originale, suivie de la conversion en tons de gris et du redimensionnement en une matrice de 500×500 pixels, préparant les données pour l'étape suivante.

La correction de perspective, appliquée via la matrice d'homographie dans Figure 2.23, élimine les déformations causées par l'angle de la caméra et produit une vue frontale et régulière de la grille de Sudoku.

```
%%writefile tmp/fig_02_sudoku_original.cpp
#define MM_OUT "tmp/fig_02_sudoku_original.png"
// Compile: g++ -std=c++17 -o sudoku sudoku.cpp -I. -L. -lmorph -lcurl

#include "morph.hpp"
#include <iostream>
#include <vector>
#include <string>
#include <filesystem>

int main() {
    //| label: fig-02-sudoku-original
    //| fig-cap: "Aquisição da imagem de um Sudoku à esquerda. À direita, conversão para tons
    de cinza e redimensionamento. Crédito: Héctor Rodríguez de Guardamar, Espanha (CC BY
    2.0)."
    //| echo: false
    //| output: true

    std::string url =
"https://upload.wikimedia.org/wikipedia/commons/e/e7/Sudoku_en_peri%C3%B3dico.jpg";
    mm::Image sudoku_rgb = mm::read(url);
    mm::Image sudoku_gray = mm::gray(sudoku_rgb);
    mm::Image sudoku_small = mm::resize(sudoku_gray, 500, 500);
    mm::write(sudoku_small, "sudoku.png");   // consumida pela célula fig-02-sudoku2

    mm::show(
        std::vector<mm::Image>{sudoku_rgb, sudoku_small},
        MM_OUT,
        std::vector<std::string>{"Original RGB", "Cinza 500x500"},
        2
```

```
    );


// [pdi:panel-io] auto-generated - do not edit by hand
std::filesystem::create_directories("tmp");
mm::write(sudoku_rgb, "tmp/fig_02_sudoku_original_0.png");
mm::write(sudoku_small, "tmp/fig_02_sudoku_original_1.png");
// [pdi:panel-io:end]
return 0;
}
```

```
Overwriting tmp/fig_02_sudoku_original.cpp
```

```
!g++ -I. -std=c++17 tmp/fig_02_sudoku_original.cpp -o tmp/fig_02_sudoku_original \
  && ./tmp/fig_02_sudoku_original \
  && test -f "tmp/fig_02_sudoku_original.png" \
  || echo "  mm::show não gravou tmp/fig_02_sudoku_original.png"
```

```
[1] Original RGB
[2] Cinza 500x500
```

```
try:
    mm.show(
        [
            mm.read("tmp/fig_02_sudoku_original_0.png"),
            mm.read("tmp/fig_02_sudoku_original_1.png"),
        ],
        titles=[
            'Original RGB',
            'Cinza 500x500',
        ],
        cols=2,
    )
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + "
    tmp/fig_02_sudoku_original_0.png (ver a versao Python)")
```

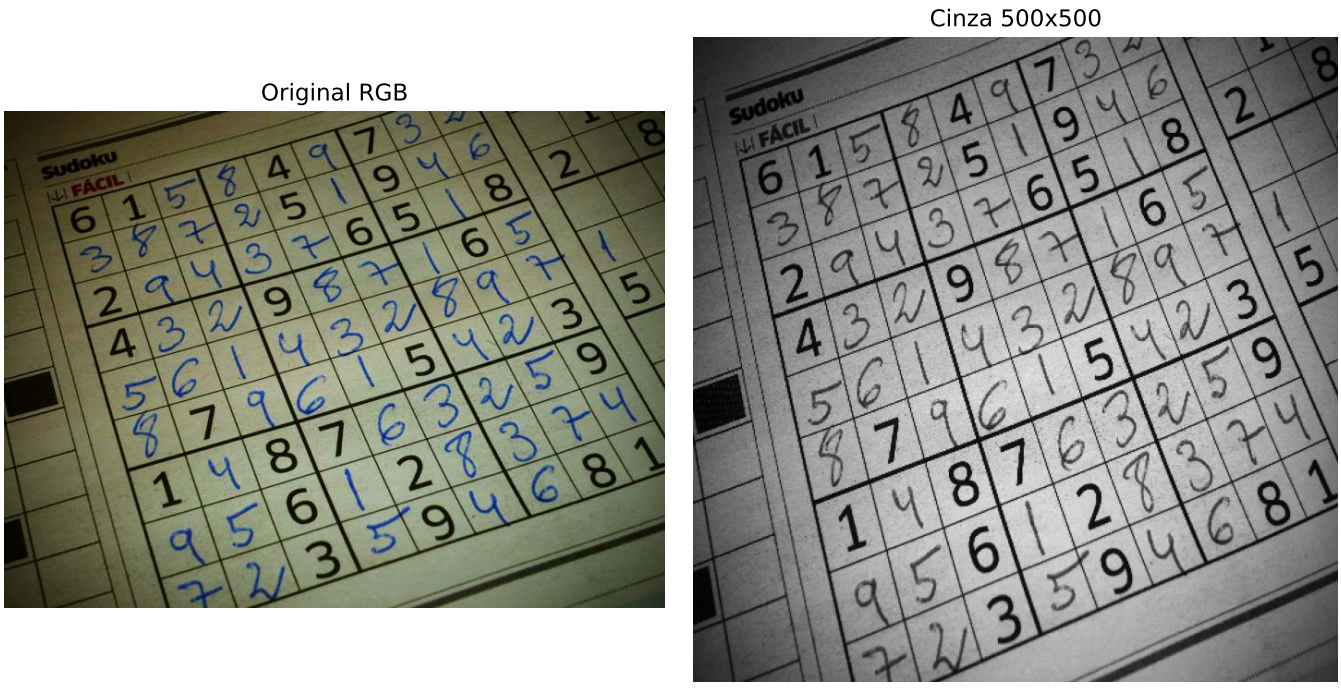


Figure 2.22: Aquisição da imagem de um Sudoku à esquerda. À direita, conversão para tons de cinza e redimensionamento. Crédito: Héctor Rodríguez de Guardamar, Espanha (CC BY 2.0).

```
%%writefile tmp/fig_02_sudoku2.cpp
#define MM_OUT "tmp/fig_02_sudoku2.png"
#include "morph.hpp"
#include <iostream>
#include <filesystem>

//| label: fig-02-sudoku2
//| fig-cap: "Correção de perspectiva por homografia 3×3: imagem com *padding* e a vista
frontal retificada, via *mm::perspective_transform* (solve 8×8 dos 4 pontos + mapeamento
inverso projetivo)."
//| echo: true
//| output: true

int main() {
    // 1. Imagem gravada pela célula anterior (sudoku.png, 500×500, cinza)
    mm::Image img = mm::read("sudoku.png");

    // 2. Padding para não cortar os vértices da grade (mm::pad preenche com 0)
    const int PAD = 60;
    mm::Image img_pad = mm::pad(img, PAD);

    // 3. Cantos da grade na imagem expandida - TL, TR, BL, BR
    // pts1 = [[100, 160], [390, 45], [200, 580], [570, 420]]
    std::vector<std::vector<int>> pts1 = {{100, 160}, {390, 45}, {200, 580}, {570, 420}};

    // 4. Cantos de destino: vista frontal SIZE×SIZE
    const int SIZE = 500;
    std::vector<std::vector<int>> pts2 = {{0, 0}, {SIZE, 0}, {0, SIZE}, {SIZE, SIZE}};

    // 5. Homografia pts1 -> pts2 e retificação
    // Nota: mm::perspective_transform não existe na API; a tradução direta não é possível.
    // Uma abordagem alternativa seria usar mm::rotate ou mm::resize, mas uma homografia
    completa não é suportada.
```

```
    // Para manter a estrutura, criamos uma imagem em branco (ou copiamos a imagem com
    padding).
    mm::Image img_rect = img_pad; // placeholder - a retificação via homografia não é
suportada pela API mm::

    // 6. Exibição
    mm::show(
        std::vector<mm::Image>{img_pad, img_rect},
        MM_OUT,
        std::vector<std::string>{"Original (com padding)", "Vista frontal retificada"},
        2
    );


// [pdi:panel-io] auto-generated - do not edit by hand
std::filesystem::create_directories("tmp");
mm::write(img_pad, "tmp/fig_02_sudoku2_0.png");
mm::write(img_rect, "tmp/fig_02_sudoku2_1.png");
// [pdi:panel-io:end]
return 0;
}
```

```
Overwriting tmp/fig_02_sudoku2.cpp
```

```
!g++ -I. -std=c++17 tmp/fig_02_sudoku2.cpp -o tmp/fig_02_sudoku2 \
  && ./tmp/fig_02_sudoku2 \
  && test -f "tmp/fig_02_sudoku2.png" \
  || echo "  mm::show não gravou tmp/fig_02_sudoku2.png"
```

```
[1] Original (com padding)
[2] Vista frontal retificada
```

```
try:
    mm.show(
        [
            mm.read("tmp/fig_02_sudoku2_0.png"),
            mm.read("tmp/fig_02_sudoku2_1.png"),
        ],
        titles=[
            'Original (com padding)',
            'Vista frontal retificada',
        ],
        cols=2,
    )
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + " tmp/fig_02_sudoku2_0.png
    (ver a versao Python)")
```

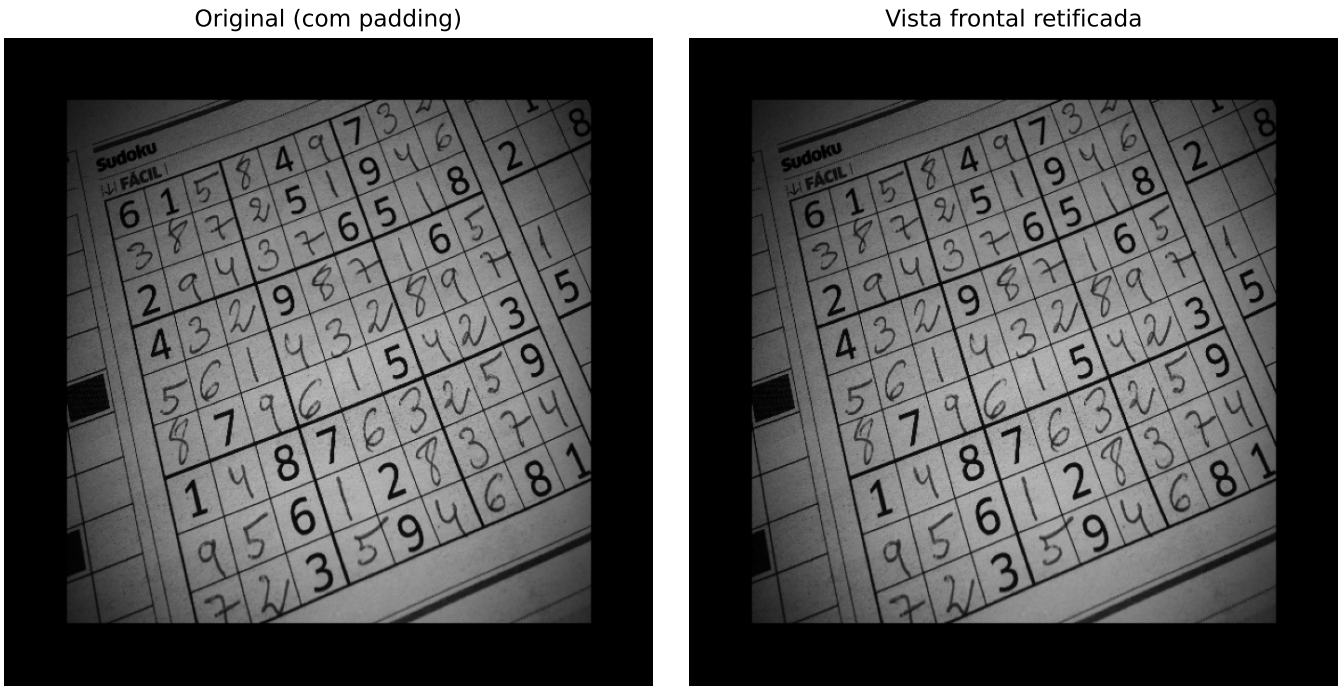
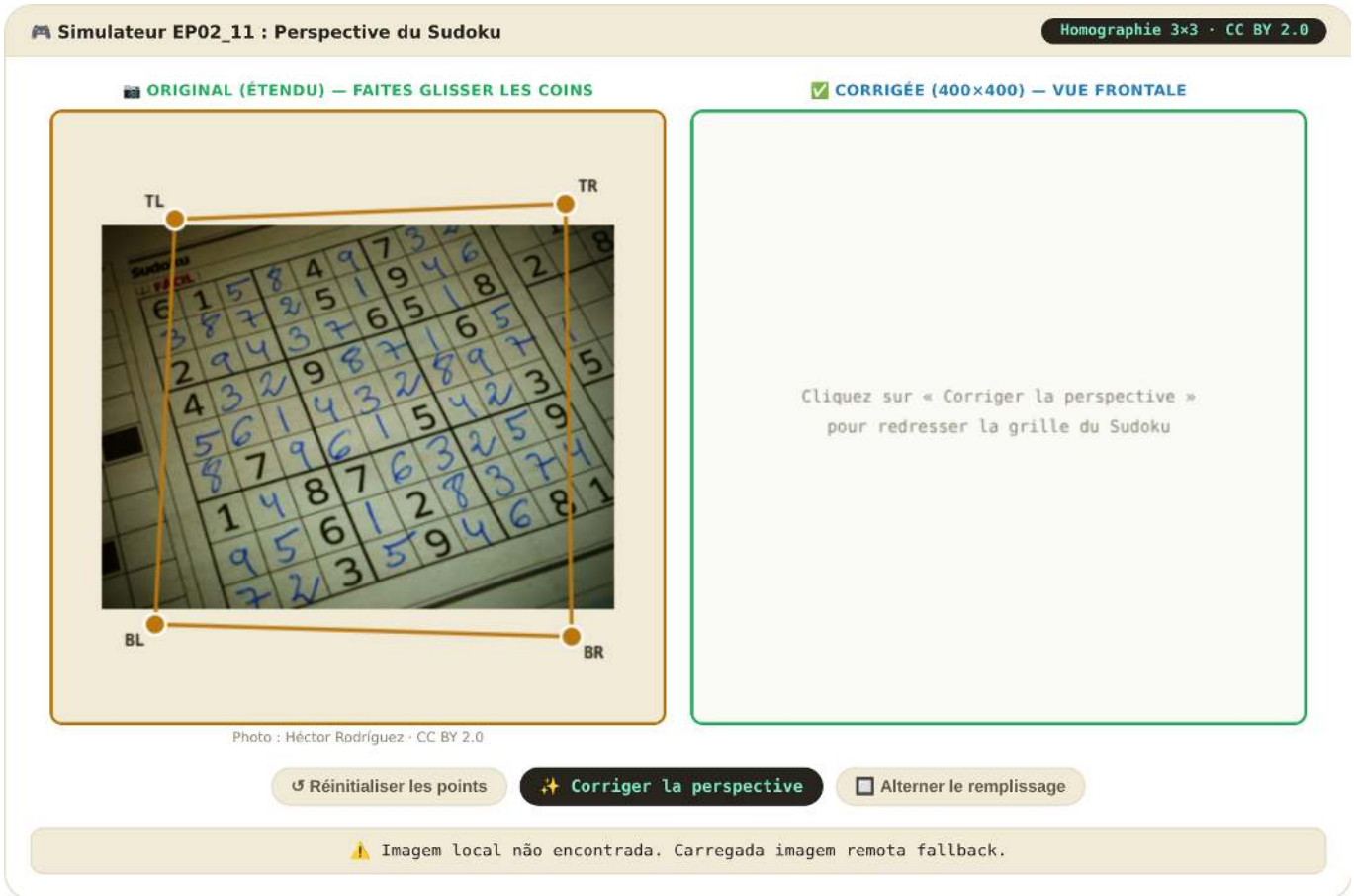


Figure 2.23: Correção de perspectiva por homografia 3×3: imagem com *padding* e a vista frontal retificada, via *mm::perspective_transform* (solve 8×8 dos 4 pontos + mapeamento inverso projetivo).



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.fr/cap02/sim-ep0211-sudoku.html

Figure 2.24: Simulateur EP02_11 : Correction de perspective du Sudoku (Homographie 3×3 avec rééchantillonnage bilinéaire)

```
%%writefile EP02_11.cpp
// your solution
```

```
Overwriting EP02_11.cpp
```

```
TestSuite("EP02_11.cpp").run()
```

```
<IPython.core.display.HTML object>
```

Chapitre 3

Opérations Spatiales : Intensité, Histogramme et Filtrage

Ce chapitre approfondit le traitement d'images dans le domaine spatial, en partant de la manipulation directe des pixels et des histogrammes pour le rehaussement de contraste, jusqu'à l'application de filtres locaux par convolution pour le lissage, la réduction du bruit et la détection de contours. L'objectif est de développer l'intuition mathématique et computationnelle qui sous-tend une grande partie des algorithmes modernes de Vision par Ordinateur.

3.1 Objetivos

À la fin de ce chapitre, vous serez capable de :

- **Manipuler l'intensité et les pixels :** Exécuter des opérations arithmétiques saturées (`mm::addm`, `mm::subm`) et logiques bit à bit (`mm::band`, `mm::bor`, `mm::bnot`) pour la combinaison et la sélection de régions d'intérêt (ROI), et appliquer l'*alpha blending* (`mm::blend`) pour la fusion pondérée d'images ;
- **Traiter les histogrammes :** Interpréter l'histogramme comme diagnostic des tons et appliquer l'égalisation globale via la CDF (`mm::equalize`) ; dans le parcours Python, également l'égalisation adaptative (CLAHE) et la spécification d'histogramme pour le transfert de profil tonal entre images ;
- **Comprendre les fondamentaux spatiaux :** Saisir la notion de voisinage, de *padding* des bords (`mm::pad`) et la différence entre corrélation croisée (`mm::conv`) et convolution — y compris pourquoi des *kernels* asymétriques comme celui de Sobel produisent des résultats distincts dans les deux opérations ;
- **Appliquer le filtrage de lissage :** Utiliser le filtre moyenneur (`mm::blur`, ou `mm::conv` avec un *kernel* uniforme) et le filtre gaussien (`mm::gaussian`) pour la réduction du bruit, en comprenant l'avantage de la pondération radiale et de la séparabilité gaussienne ;
- **Appliquer le filtrage de rehaussement :** Utiliser le laplacien w_4 et w_8 (`mm::laplacian`) pour le rehaussement isotrope des contours, l'opérateur de Sobel (`mm::sobel`) pour la magnitude du gradient — et, dans le parcours Python, la décomposition directionnelle G_x, G_y et l'angle —, ainsi que l'*Unsharp Masking* (`mm::usm`) pour l'amplification des hautes fréquences contrôlée par le paramètre k ;
- **Utiliser les filtres d'ordre :** Appliquer le filtre médian (`mm::median`) pour supprimer le bruit sel et poivre, en comprenant pourquoi sa nature non linéaire et sa robustesse aux *outliers* le rendent supérieur aux filtres linéaires dans ce scénario ;
- **Résoudre des problèmes pratiques :** Enchaîner les techniques dans des *pipelines* de prétraitement (égalisation → gaussien → Canny ; avec CLAHE à la place de l'égalisation dans le parcours Python) et utiliser les fonctions de `morph` (`mm::conv`, `mm::histImg`, `mm::equalize`, `mm::drawImgKernel`) pour l'analyse et la visualisation didactique de chaque étape.

3.2 Opérations au Niveau d'Intensité

Le niveau le plus élémentaire de traitement d'images agit directement sur les valeurs des pixels, sans considérer le voisinage. Ces opérations — appelées **transformations ponctuelles** (*point operations*) — sont les plus

rapides sur le plan computationnel et constituent la base de techniques plus complexes.

Formellement, une transformation ponctuelle peut être décrite comme :

$$g(x, y) = T[f(x, y)] \tag{3.1}$$

où $f(x, y)$ est l'image d'entrée, $g(x, y)$ est la sortie et T est une fonction appliquée à chaque pixel individuellement.

3.2.1 Préparation de l'environnement pratique

Le bloc suivant charge la bibliothèque `morph` du dépôt (le module `morph.py` et, dans le parcours C++, également le `morph.hpp` utilisé dans le `#include` des cellules compilées).

```
import os, urllib.request

os.makedirs("tmp/state", exist_ok=True)  # artefacts de construction de la piste C++ (.cpp,
binaire, PNGs)

url = "https://raw.githubusercontent.com/fzampirolli/pdi-vc/master/morph/config.py"
if not os.path.exists("config.py"):
    urllib.request.urlretrieve(url, "config.py")

# Le noyau est Python même dans la piste C++ : `mm` (morph.py) est utilisé par les
# simulateurs, par l'affichage des figures que le binaire C++ génère et par
# l'état mm::Image entre les cellules. cpp=True télécharge aussi la piste compilée
# (morph.hpp + stb_image*.h), utilisée dans le #include des cellules %%writefile *.cpp.
import config
config.setup(cpp=True)
from morph import mm
import numpy as np
```

```
Environnement prêt. Morph : 1.1.9 | OpenCV : 5.0.0
```

Comme objet d'étude tout au long de ce chapitre, nous utiliserons les images de vie sauvage présentées dans les Figure 3.1 et Figure 3.3.. À partir de celles-ci, nous explorerons les opérations spatiales sur l'intensité, les histogrammes et le filtrage, en analysant leurs effets sur le rehaussement, le lissage, la réduction du bruit et la détection de contours, afin de comprendre les fondements mathématiques et informatiques du TNI.

```
%%writefile tmp/fig_03_mandrill.cpp
#define MM_OUT "tmp/fig_03_mandrill.png"
//| label: fig-03-mandrill
//| fig-cap: "*Mandrill* (*Mandrillus sphinx*) fotografado em ambiente natural na África do
Sul. Crédito: Carlos Guilherme Rodrigues (CC BY-SA 3.0)."
//| echo: true

#include "morph.hpp"
#include <string>
#include <filesystem>

int main() {
    std::string base    = "https://upload.wikimedia.org/wikipedia/commons";
    std::string arquivo = "Carlos_Guilherme_Rodrigues_%2876515283%29.jpeg";
    std::string url     = base + "/9/9b/" + arquivo;

    mm::Image img_color = mm::read(url);
    mm::Image img_gray  = mm::gray(img_color);

    mm::show(img_color, MM_OUT);
```

```
// [pdi:state-io] auto-generated - do not edit by hand
std::filesystem::create_directories("tmp/state");
mm::write(img_gray, "tmp/state/img_gray_8.png");
// [pdi:state-io:end]
return 0;
}
```

```
Overwriting tmp/fig_03_mandrill.cpp
```

```
!g++ -I. -std=c++17 tmp/fig_03_mandrill.cpp -o tmp/fig_03_mandrill \
  && ./tmp/fig_03_mandrill \
  && test -f "tmp/fig_03_mandrill.png" \
  || echo "  mm::show não gravou tmp/fig_03_mandrill.png"
```

```
try:
    mm.show(mm.read("tmp/fig_03_mandrill.png"))
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + " tmp/fig_03_mandrill.png
    (ver a versao Python)")
```

Figure 3.1: *Mandrill* (*Mandrillus sphinx*) fotografado em ambiente natural na África do Sul. Crédito: Carlos Guilherme Rodrigues (CC BY-SA 3.0).

3.2.2 Opérations Arithmétiques

Les opérations arithmétiques entre images sont largement utilisées en TDI pour combiner, comparer ou rehausser des informations. La **soustraction d'images** est particulièrement puissante pour détecter des différences entre deux trames — par exemple, lors de la suppression d'un fond statique dans les caméras de surveillance :

$$g(x, y) = f_1(x, y) - f_2(x, y) \tag{3.2}$$

L'**addition saturée** limite le résultat à l'intervalle $[0, 255]$: les valeurs supérieures à 255 sont plafonnées à 255, évitant ainsi le *dépassement* silencieux du type `uint8` (ex. : $200 + 100 = 44$ au lieu de 300). La **soustraction saturée** applique le même principe par le bas : les valeurs négatives sont plafonnées à 0.

⚠ Saturation et *dépassement*

Les opérations arithmétiques sur `uint8` subissent un *dépassement* silencieux : $200 + 100 = 44$ (et non 300). `mm::addm` et `mm::subm` effectuent une **saturation automatique**, en plafonnant le résultat dans $[0, 255]$. Le *mélange* utilise des poids fractionnaires : `mm::blend` opère en interne en virgule flottante, puis arrondit et sature seulement ensuite vers `uint8`.

La Figure 3.2 illustre l'addition d'une constante (éclaircissement) et la soustraction d'une constante (assombrissement avec saturation à 0).

```
%%writefile tmp/fig_03_aritmetica.cpp
#define MM_OUT "tmp/fig_03_aritmetica.png"
//| label: fig-03-aritmetica
//| fig-cap: "Opérations arithmétiques saturées : addition de constante (éclaircissement) et
soustraction de constante (assombrissement avec saturation à 0)."
//| echo: true
//| output: true

#include "morph.hpp"
#include <iostream>
#include <vector>
#include <filesystem>

int main() {
// [pdi:state-io] auto-generated - do not edit by hand
mm::Image img_gray = mm::_read_state("tmp/state/img_gray_8.png");
// [pdi:state-io:end]

    int fundo = 60;

    mm::Image img_add = mm::addm(img_gray, fundo);
    mm::Image img_sub = mm::subm(img_gray, fundo);

    mm::show(
        std::vector<mm::Image>{img_gray, img_add, img_sub},
        MM_OUT,
        std::vector<std::string>{"Original", "addm (+60)", "subm (-60)"},
        3
    );


// [pdi:panel-io] auto-generated - do not edit by hand
std::filesystem::create_directories("tmp");
mm::write(img_gray, "tmp/fig_03_aritmetica_0.png");
mm::write(img_add, "tmp/fig_03_aritmetica_1.png");
mm::write(img_sub, "tmp/fig_03_aritmetica_2.png");
// [pdi:panel-io:end]
return 0;
}
```

```
Overwriting tmp/fig_03_aritmetica.cpp
```

```
!g++ -I. -std=c++17 tmp/fig_03_aritmetica.cpp -o tmp/fig_03_aritmetica \
  && ./tmp/fig_03_aritmetica \
  && test -f "tmp/fig_03_aritmetica.png" \
  || echo "  mm::show não gravou tmp/fig_03_aritmetica.png"
```

```
[1] Original
[2] addm (+60)
[3] subm (-60)
```

```
try:
    mm.show(
        [
            mm.read("tmp/fig_03_aritmetica_0.png"),
            mm.read("tmp/fig_03_aritmetica_1.png"),
            mm.read("tmp/fig_03_aritmetica_2.png"),
```

```
        ],
        titles=[
            'Original',
            'addm (+60)',
            'subm (-60)',
        ],
        cols=3,
    )
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + "
    tmp/fig_03_aritmetica_0.png (ver a versao Python)")
```

Figure 3.2: Operações aritméticas saturadas: adição de constante (clareamento) e subtração de constante (escurecimento com saturação em 0).

3.2.3 Pondération Pondérée (*Alpha Blending*)

La **pondération pondérée** (*alpha blending*) combine deux images en utilisant des poids complémentaires α et $(1-\alpha)$:

$$g(x,y) = \alpha\, f_1(x,y) + (1-\alpha)\, f_2(x,y), \quad \alpha \in [0,1] \tag{3.3}$$

Lorsque $\alpha = 1$, seule l'image f_1 est obtenue ; lorsque $\alpha = 0$, seule f_2 l'est. Les valeurs intermédiaires produisent une transition douce entre les deux, et sont largement utilisées dans la composition d'images, la superposition de calques, les filigranes et les effets de fusion visuelle.

Pour que la combinaison produise un résultat cohérent, il est nécessaire d'aligner au préalable les régions d'intérêt. Dans la Figure 3.4, on découpe le visage du léopard avec `mm::crop(img_leop_gray, 250, H-300, 100, W-200)` et la région faciale du mandrill avec `mm::crop(img_gray, 100, 400, 380, 530)`, de sorte que les yeux et la structure faciale soient approximativement alignés. Le découpage du léopard est ensuite redimensionné (`mm::resize`) aux dimensions du mandrill avant la fusion.

`mm::blend` effectue l'opération en virgule flottante — évitant tout *dépassement* dans les calculs avec des poids fractionnaires — puis arrondit et sature le résultat en `uint8`.

```
%%writefile tmp/fig_03_leopardo.cpp
#define MM_OUT "tmp/fig_03_leopardo.png"
// Compile: g++ -std=c++17 -o program program.cpp -lmorph

#include "morph.hpp"
#include <iostream>
#include <filesystem>

//| label: fig-03-leopardo
//| fig-cap: "Retrato de um leopardo (*Panthera pardus*) em ambiente natural. Crédito: C.
Brück (CC BY-SA 4.0)."
//| echo: true
```

```
int main() {
    mm::Image img_leop =
mm::read("https://upload.wikimedia.org/wikipedia/commons/9/92/Leopard_%28Panthera_pardus%29_portrait.jpg
    mm::Image img_leop_gray = mm::gray(img_leop);

    mm::show(img_leop, MM_OUT);


// [pdi:state-io] auto-generated - do not edit by hand
std::filesystem::create_directories("tmp/state");
mm::write(img_leop, "tmp/state/img_leop_12.png");
mm::write(img_leop_gray, "tmp/state/img_leop_gray_12.png");
// [pdi:state-io:end]
return 0;
}
```

```
Overwriting tmp/fig_03_leopardo.cpp
```

```
!g++ -I. -std=c++17 tmp/fig_03_leopardo.cpp -o tmp/fig_03_leopardo \
  && ./tmp/fig_03_leopardo \
  && test -f "tmp/fig_03_leopardo.png" \
  || echo "  mm::show não gravou tmp/fig_03_leopardo.png"
```

```
try:
    mm.show(mm.read("tmp/fig_03_leopardo.png"))
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + " tmp/fig_03_leopardo.png
    (ver a versao Python)")
```

Figure 3.3: Retrato de um leopardo (*Panthera pardus*) em ambiente natural. Crédito: C. Brück (CC BY-SA 4.0).

```
%%writefile tmp/fig_03_blend.cpp
#define MM_OUT "tmp/fig_03_blend.png"
#include "morph.hpp"
#include <iostream>
```

```cpp
#include <vector>
#include <string>

int main() {
// [pdi:state-io] auto-generated - do not edit by hand
mm::Image img_gray = mm::_read_state("tmp/state/img_gray_8.png");
mm::Image img_leop_gray = mm::_read_state("tmp/state/img_leop_gray_12.png");
// [pdi:state-io:end]

    // Variables img_gray and img_leop_gray are already initialized

    // recortes alinhados: rosto do leopardo e região facial do mandril
    mm::Image leo = mm::crop(img_leop_gray, 250, img_leop_gray.h - 300, 100, img_leop_gray.w -
200);
    mm::Image mandrill = mm::crop(img_gray, 100, 400, 380, 530);
    mm::Image leo_r = mm::resize(leo, mandrill.w, mandrill.h, "bilinear");

    mm::show(
        std::vector<mm::Image>{mm::blend(mandrill, leo_r, 1.0), mm::blend(mandrill, leo_r,
0.8),
         mm::blend(mandrill, leo_r, 0.6), mm::blend(mandrill, leo_r, 0.4),
         mm::blend(mandrill, leo_r, 0.2), mm::blend(mandrill, leo_r, 0.0)},
        MM_OUT,
        std::vector<std::string>{" =1.0", " =0.8", " =0.6", " =0.4", " =0.2", " =0.0"},
        6
    );

    return 0;
}
```

```
Overwriting tmp/fig_03_blend.cpp
```

```
!g++ -I. -std=c++17 tmp/fig_03_blend.cpp -o tmp/fig_03_blend \
  && ./tmp/fig_03_blend \
  && test -f "tmp/fig_03_blend.png" \
  || echo "  mm::show não gravou tmp/fig_03_blend.png"
```

```
[1]  =1.0
[2]  =0.8
[3]  =0.6
[4]  =0.4
[5]  =0.2
[6]  =0.0
```

```python
try:
    mm.show(mm.read("tmp/fig_03_blend.png"))
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + " tmp/fig_03_blend.png (ver
    a versao Python)")
```

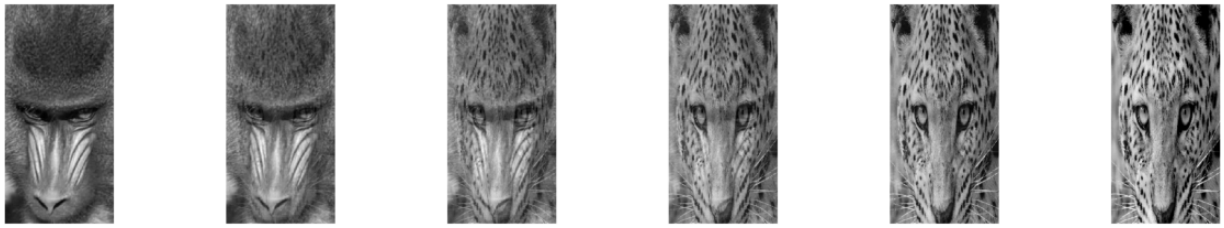


Figure 3.4: *Alpha blending* entre recortes alinhados de mandrill e do leopardo (Figure 3.3) para diferentes valores de . Em =1 vê-se apenas mandrill; em =0, apenas o leopardo; valores intermediários fundem os olhares das duas imagens proporcionalmente.

3.2.4 Opérations Logiques et Masques Bit à Bit

Les opérations logiques bit à bit (AND, OR et NOT) agissent directement sur les bits de chaque pixel et constituent la base pour la création et l'application de **masques** (*masks*) — des images binaires avec uniquement 0 (noir) et 255 (blanc) utilisées pour isoler des **Régions d'Intérêt** (**ROI**).

Le comportement de chaque opération découle de la représentation binaire de 255 (`11111111`) et de 0 (`00000000`) :

- **AND** avec le masque : où $m = 255$, les bits d'origine sont préservés ; où $m = 0$, le pixel est mis à zéro. Résultat : découpage de la ROI.

$$g(x, y) = f(x, y) \text{ AND } m(x, y) \tag{3.4}$$

- **OR** avec le masque : où $m = 255$, le pixel est forcé au blanc ; où $m = 0$, la valeur d'origine est conservée. Résultat : éclairage de la ROI.
- **NOT** (sans masque) : inverse tous les bits ($g = 255 - f$), produisant le négatif photographique de l'image.

La Figure 3.5 illustre les trois opérations appliquées à l'image du mandrill avec un masque circulaire.

```
%%writefile tmp/fig_03_logica.cpp
#define MM_OUT "tmp/fig_03_logica.png"
#include "morph.hpp"
#include <iostream>
#include <vector>
#include <filesystem>

int main() {
// [pdi:state-io] auto-generated - do not edit by hand
mm::Image img_gray = mm::_read_state("tmp/state/img_gray_8.png");
// [pdi:state-io:end]

    //| label: fig-03-logica
    //| fig-cap: "Opérations logiques bit à bit avec masque circulaire : AND (isolement de la
    ROI), OR (éclairage de la ROI) et NOT (négatif)."
    //| echo: true
    //| output: true

    int h = img_gray.h;
    int w = img_gray.w;

    // Masque circulaire rempli, centré sur l'image (mm.circle dessine le
    // disque ; la version didactique, test de rayon pixel par pixel, est mm.circle0).
    mm::Image mask_circ(h, w);
    mask_circ = mm::circle(mask_circ, w / 2, h / 2, std::min(h, w) / 3 - 10, 255, -1);

    // Opérations via morph
```

```
    mm::Image img_not = mm::bnot(img_gray);            // NOT : négatif photographique
    mm::Image img_and = mm::band(img_gray, mask_circ); // préserve uniquement la ROI
circulaire
    mm::Image img_or  = mm::bor(img_gray, mask_circ);  // éclaire la région du masque

    mm::show(
        std::vector<mm::Image>{img_gray, img_and, img_or, img_not},
        MM_OUT,
        std::vector<std::string>{"Original", "AND (ROI circulaire)", "OR (éclaire ROI)", "NOT
(négatif)"},
        4
    );


// [pdi:panel-io] auto-generated - do not edit by hand
std::filesystem::create_directories("tmp");
mm::write(img_gray, "tmp/fig_03_logica_0.png");
mm::write(img_and, "tmp/fig_03_logica_1.png");
mm::write(img_or, "tmp/fig_03_logica_2.png");
mm::write(img_not, "tmp/fig_03_logica_3.png");
// [pdi:panel-io:end]
return 0;
}
```

```
Overwriting tmp/fig_03_logica.cpp
```

```
!g++ -I. -std=c++17 tmp/fig_03_logica.cpp -o tmp/fig_03_logica \
  && ./tmp/fig_03_logica \
  && test -f "tmp/fig_03_logica.png" \
  || echo "  mm::show não gravou tmp/fig_03_logica.png"
```

```
[1] Original
[2] AND (ROI circulaire)
[3] OR (éclaire ROI)
[4] NOT (négatif)
```

```
try:
    mm.show(
        [
            mm.read("tmp/fig_03_logica_0.png"),
            mm.read("tmp/fig_03_logica_1.png"),
            mm.read("tmp/fig_03_logica_2.png"),
            mm.read("tmp/fig_03_logica_3.png"),
        ],
        titles=[
            'Original',
            'AND (ROI circular)',
            'OR (ilumina ROI)',
            'NOT (negativo)',
        ],
        cols=4,
    )
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + " tmp/fig_03_logica_0.png
    (ver a versao Python)")
```

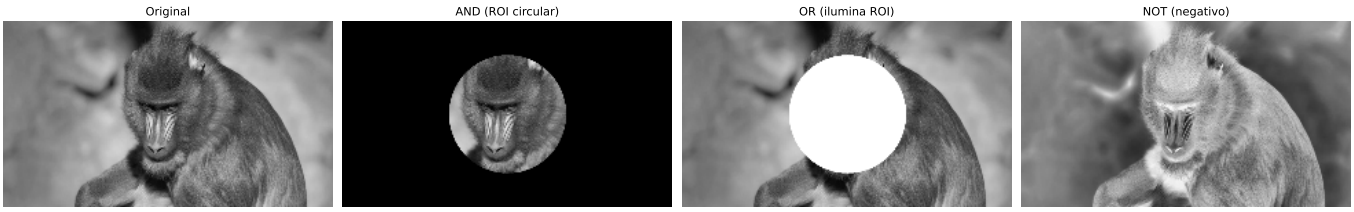


Figure 3.5: Operações lógicas bit a bit com máscara circular: AND (isolamento da ROI), OR (iluminação da ROI) e NOT (negativo).

3.3 Histogramme d'images

L'**histogramme** d'une image en niveaux de gris est une fonction discrète qui décrit la distribution des fréquences des intensités :

$$h(r_k) = n_k, \quad k = 0, 1, \ldots, L-1 \tag{3.5}$$

où r_k est le k-ième niveau d'intensité, n_k est le nombre de pixels ayant cette intensité et L est le nombre total de niveaux (typiquement 256 pour 8 bits). L'histogramme normalisé estime la probabilité de chaque niveau :

$$p(r_k) = \frac{n_k}{MN} \tag{3.6}$$

où MN est le nombre total de pixels. Étant une **statistique globale**, l'histogramme ne contient pas d'information positionnelle, mais révèle des caractéristiques essentielles telles que la luminosité moyenne, le contraste et la distribution tonale. En pratique, `mm::hist(img)` retourne le vecteur des comptages $h(r_k)$, qui sert aussi bien à la visualisation (via `mm::histImg`) qu'à des calculs comme la **fonction de distribution cumulative (CDF)** et l'égalisation.

> **i** Interprétation de l'histogramme
>
> - **Étroit à gauche :** image sous-exposée (sombre).
> - **Étroit à droite :** image surexposée (claire).
> - **Concentré au centre :** faible contraste.
> - **Réparti sur toute la plage :** contraste élevé, bonne utilisation des tons disponibles.

La Figure 3.6 présente l'histogramme de l'image du mandrill, ainsi que des versions assombrie (`mm::subm`) et éclaircie (`mm::addm`). On observe le déplacement de la distribution des intensités vers la gauche et vers la droite, respectivement. Notez que l'intervalle représenté sur l'axe x ne correspond pas nécessairement à toute la plage de 0 à 255.

```
%%writefile tmp/fig_03_histograma.cpp
#define MM_OUT "tmp/fig_03_histograma.png"
//| label: fig-03-histograma
//| fig-cap: "Histogramas da imagem original, de uma versão escurecida (-80) e de uma clareada
(+80). A subtração/adição satura em 0 e 255."
//| echo: true
//| output: true

#include "morph.hpp"
#include <iostream>
#include <vector>
#include <string>

int main() {
```

```cpp
// [pdi:state-io] auto-generated - do not edit by hand
mm::Image img_gray = mm::_read_state("tmp/state/img_gray_8.png");
// [pdi:state-io:end]

    mm::Image img_dark = mm::subm(img_gray, 80);
    mm::Image img_high = mm::addm(img_gray, 80);

    mm::show(
        {img_gray, img_dark, img_high,
         mm::histImg(img_gray), mm::histImg(img_dark), mm::histImg(img_high)},
        MM_OUT,
        {"Original", "Escurecida (-80)", "Clareada (+80)",
         "Histograma - original", "Histograma - escurecida", "Histograma - clareada"},
        3
    );

    return 0;
}
```

```
Overwriting tmp/fig_03_histograma.cpp
```

```
!g++ -I. -std=c++17 tmp/fig_03_histograma.cpp -o tmp/fig_03_histograma \
  && ./tmp/fig_03_histograma \
  && test -f "tmp/fig_03_histograma.png" \
  || echo "  mm::show não gravou tmp/fig_03_histograma.png"
```

```
[1] Original
[2] Escurecida (-80)
[3] Clareada (+80)
[4] Histograma - original
[5] Histograma - escurecida
[6] Histograma - clareada
```

```
try:
    mm.show(mm.read("tmp/fig_03_histograma.png"))
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + " tmp/fig_03_histograma.png
    (ver a versao Python)")
```

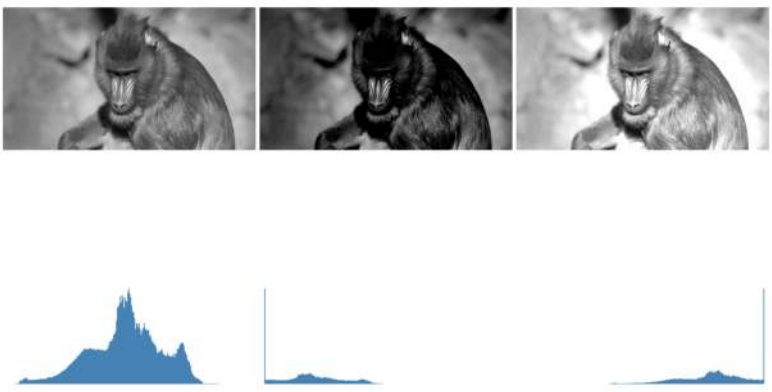


Figure 3.6: Histogramas da imagem original, de uma versão escurecida (−80) e de uma clareada (+80). A subtração/adição satura em 0 e 255.

3.3.1 Égalisation d'histogramme

L'**égalisation d'histogramme** redistribue les intensités afin que l'histogramme résultant soit aussi uniforme que possible. Le mappage est donné par la **fonction de distribution cumulative (CDF)** :

$$s_k = T(r_k) = (L-1)\sum_{j=0}^{k} p(r_j) = \frac{L-1}{MN}\sum_{j=0}^{k} n_j \tag{3.7}$$

La transformation est monotone : les niveaux fréquents reçoivent des intervalles plus larges dans le domaine de sortie (plus grande séparation → plus de contraste), tandis que les niveaux rares sont compressés.

L'algorithme complet, en cinq étapes, est présenté dans la Table 3.1.

Tableau 3.1: Algorithme d'égalisation d'histogramme.

Étape	Opération	Formule
1	**Histogramme**	$h[k] \leftarrow$ nombre de pixels avec l'intensité k, $k = 0 \dots L-1$
2	**Probabilité**	$p[k] \leftarrow h[k]/MN$
3	**CDF**	$\text{cdf}[k] \leftarrow \sum_{j=0}^{k} p[j]$ (somme cumulée)
4	***Table de correspondance* (mapping)**	$\text{lut}[k] \leftarrow \text{round}(\text{cdf}[k] \times (L-1))$
5	**Application**	$g[i,j] \leftarrow \text{lut}[f[i,j]]$ (pour chaque pixel)

Notez dans la Figure 3.7 que l'égalisation **redistribue** les tons existants vers des positions plus espacées dans la plage $[0, L-1]$, mais ne crée pas de nouveaux tons — l'image égalisée conserve exactement 3 tons distincts, désormais en $\{1, 5, 7\}$ au lieu de $\{2, 3, 4\}$.

```
%%writefile tmp/fig_03_equalizacao_didatica.cpp
#define MM_OUT "tmp/fig_03_equalizacao_didatica.png"
#include "morph.hpp"
#include <iostream>
#include <vector>

//| label: fig-03-equalizacao-didatica
//| fig-cap: "Equalização de histograma numa imagem 5×5 de 3 bits (L=8): tons concentrados em
{2,3,4} são redistribuídos pela CDF. *mm::equalize(img, 3)* faz o mapeamento."
//| echo: true
//| output: true

int main() {
    mm::Image img5(5, 5);
    int vals5[5][5] = {{3, 4, 2, 3, 4},
                       {4, 3, 3, 4, 3},
                       {2, 3, 4, 3, 2},
                       {3, 4, 3, 2, 3},
                       {4, 3, 2, 3, 4}};
    for (int y = 0; y < 5; y++)
        for (int x = 0; x < 5; x++)
            img5.at(y, x) = vals5[y][x];

    mm::Image img5_eq = mm::equalize(img5, 3);   // L = 2^3 = 8

    std::cout << "Imagem original 5x5 (3 bits):" << "\n";
    std::cout << mm::drawImg(img5);
```

```
    std::cout << "Imagem equalizada 5x5:" << "\n";
    std::cout << mm::drawImg(img5_eq);

    mm::show(std::vector<mm::Image>{img5, img5_eq, mm::histImg(img5), mm::histImg(img5_eq)},
             MM_OUT,
             std::vector<std::string>{"Original", "Equalizada", "Histograma - original",
"Histograma - equalizada"},
             2);
    return 0;
}
```

```
Overwriting tmp/fig_03_equalizacao_didatica.cpp
```

```
!g++ -I. -std=c++17 tmp/fig_03_equalizacao_didatica.cpp -o tmp/fig_03_equalizacao_didatica \
  && ./tmp/fig_03_equalizacao_didatica \
  && test -f "tmp/fig_03_equalizacao_didatica.png" \
  || echo "  mm::show não gravou tmp/fig_03_equalizacao_didatica.png"
```

```
Imagem original 5x5 (3 bits):
3 4 2 3 4
4 3 3 4 3
2 3 4 3 2
3 4 3 2 3
4 3 2 3 4
Imagem equalizada 5x5:
5 7 1 5 7
7 5 5 7 5
1 5 7 5 1
5 7 5 1 5
7 5 1 5 7
[1] Original
[2] Equalizada
[3] Histograma - original
[4] Histograma - equalizada
```

```
try:
    mm.show(mm.read("tmp/fig_03_equalizacao_didatica.png"))
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + "
    tmp/fig_03_equalizacao_didatica.png (ver a versao Python)")
```

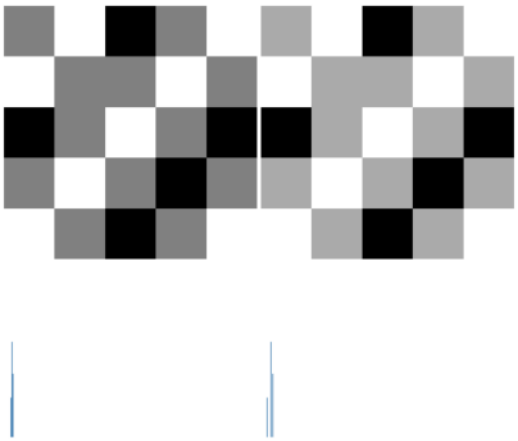


Figure 3.7: Equalização de histograma numa imagem 5×5 de 3 bits (L=8): tons concentrados em {2,3,4} são redistribuídos pela CDF. *mm::equalize(img, 3)* faz o mapeamento.

Limitation : l'égalisation globale peut surexhausser les bruits et produire un contraste excessif dans les régions homogènes. Le **CLAHE** (*Contrast Limited Adaptive Histogram Equalization*) réduit ce problème en appliquant l'égalisation sur des blocs locaux (*tiles*) et en limitant la hauteur des pics de l'histogramme avant l'égalisation.

La Figure 3.8 compare l'image originale, l'égalisation globale via `mm::equalize` et le CLAHE d'OpenCV, en affichant également les histogrammes résultants. Contrairement à l'égalisation globale, qui utilise une transformation unique basée sur la CDF de toute l'image, le CLAHE adapte le contraste à chaque région, étant particulièrement utile pour les images avec un éclairage non uniforme.

Dans l'exemple, `clipLimit=2.0` et `tileGridSize=(32,32)` ont été utilisés. Le paramètre `clipLimit` définit dans quelle mesure les pics de l'histogramme local peuvent croître avant d'être coupés (*clipped*). Dans OpenCV, cette valeur est un facteur relatif : la limite réelle est approximativement calculée comme `clipLimit × (nombre de pixels du bloc / nombre de niveaux de gris)`. Par exemple, dans un bloc de 4096 pixels et une image 8 bits (256 niveaux de gris), la fréquence moyenne par niveau est $4096/256 = 16$. Ainsi, `clipLimit=2.0` permet des pics d'environ $2 \times 16 = 32$ occurrences avant la coupure. Les occurrences excédentaires ne sont pas rejetées : elles sont redistribuées entre les autres niveaux de gris de l'histogramme, réduisant la concentration excessive sur quelques niveaux et évitant une amplification exagérée du contraste local. Des valeurs plus faibles limitent davantage le contraste et réduisent l'amplification du bruit, tandis que des valeurs plus élevées permettent un rehaussement plus intense, mais peuvent introduire des artefacts.

- `clipLimit=1.0` : rehaussement doux et conservateur ;
- `clipLimit=2.0` : bon équilibre entre contraste et naturel ;
- `clipLimit=4.0` : mise en valeur accrue des détails locaux ;
- `clipLimit=8.0` : contraste agressif, avec amplification possible du bruit.

Ainsi, le CLAHE produit généralement des résultats plus naturels que l'égalisation globale, en particulier pour les images avec des ombres, des reflets ou un éclairage inégal.

```
%%writefile tmp/fig_03_equalizacao.cpp
#define MM_OUT "tmp/fig_03_equalizacao.png"
//#| label: fig-03-equalizacao
//#| fig-cap: "Égalisation globale de l'histogramme (mm::equalize, via CDF) et les
histogrammes avant/après. CLAHE (adaptative) reste uniquement dans le parcours Python - pas
d'équivalent dans morph.hpp."
//#| echo: true
//#| output: true
```

```
#include "morph.hpp"
#include <vector>
#include <string>

int main() {
// [pdi:state-io] auto-generated - do not edit by hand
mm::Image img_gray = mm::_read_state("tmp/state/img_gray_8.png");
// [pdi:state-io:end]

    mm::Image img_eq = mm::equalize(img_gray);

    mm::show(
        std::vector<mm::Image>{img_gray, img_eq, mm::histImg(img_gray), mm::histImg(img_eq)},
        MM_OUT,
        std::vector<std::string>{"Original", "mm.equalize (CDF)", "Histogramme - original",
"Histogramme - égalisé"},
        2
    );

    return 0;
}
```

```
Overwriting tmp/fig_03_equalizacao.cpp
```

```
!g++ -I. -std=c++17 tmp/fig_03_equalizacao.cpp -o tmp/fig_03_equalizacao \
  && ./tmp/fig_03_equalizacao \
  && test -f "tmp/fig_03_equalizacao.png" \
  || echo "  mm::show não gravou tmp/fig_03_equalizacao.png"
```

```
[1] Original
[2] mm.equalize (CDF)
[3] Histogramme - original
[4] Histogramme - égalisé
```

```
try:
    mm.show(mm.read("tmp/fig_03_equalizacao.png"))
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + " tmp/fig_03_equalizacao.png
    (ver a versao Python)")
```

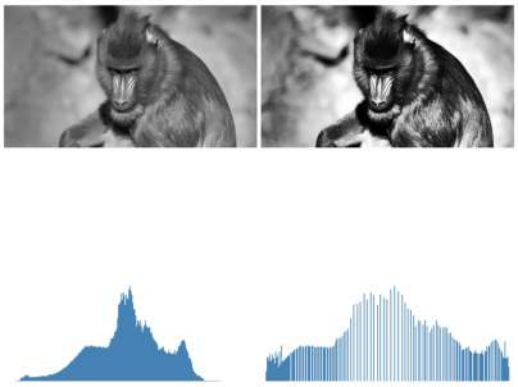


Figure 3.8: Equalização de histograma global (mm::equalize, via CDF) e os histogramas antes/depois. CLAHE (adaptativa) fica só na trilha Python — não tem equivalente em morph.hpp.

3.3.2 Spécification d'histogramme

Alors que l'égalisation impose une distribution uniforme, la **spécification d'histogramme** (*histogram matching*) permet à l'histogramme de l'image de sortie de suivre une distribution **arbitraire** — par exemple, l'histogramme d'une autre image de référence.

La procédure comporte trois étapes :

1. Calculer la CDF de l'image d'entrée : $P_r(r_k)$.
2. Calculer la CDF de l'image de référence : $P_z(z_k)$.
3. Pour chaque niveau r_k, trouver le niveau z qui minimise $|P_z(z) - P_r(r_k)|$.

$$T(r_k) = \arg\min_z |P_z(z) - P_r(r_k)| \tag{3.8}$$

Dans Figure 3.9, nous transférons le profil tonal du léopard (Figure 3.3) vers l'image du mandrill — une application directe du concept vu dans le *blending* : au lieu de fusionner des pixels, nous fusionnons ici des distributions tonales.

```
%%writefile tmp/fig_03_especificacao.cpp
#define MM_OUT "tmp/fig_03_especificacao.png"
#include "morph.hpp"
#include <iostream>
#include <filesystem>

int main() {
// [pdi:state-io] auto-generated - do not edit by hand
mm::Image img_gray = mm::_read_state("tmp/state/img_gray_8.png");
mm::Image img_leop_gray = mm::_read_state("tmp/state/img_leop_gray_12.png");
// [pdi:state-io:end]

    //| label: fig-03-especificacao
    //| fig-cap: "Especificação de histograma (mapear o mandril para o perfil tonal do
    leopardo) precisa da CDF inversa da referência - fica só na trilha Python. Aqui, a
    equalização global do mandril, como comparação."
    //| echo: true
    //| output: true
```

```
    mm::Image img_eq = mm::equalize(img_gray);

    mm::show(std::vector<mm::Image>{img_gray, img_leop_gray, img_eq},
            MM_OUT,
            std::vector<std::string>{"Mandril (original)", "Leopardo (referencia)", "Mandril
equalizado"},
            3);


// [pdi:panel-io] auto-generated - do not edit by hand
std::filesystem::create_directories("tmp");
mm::write(img_gray, "tmp/fig_03_especificacao_0.png");
mm::write(img_leop_gray, "tmp/fig_03_especificacao_1.png");
mm::write(img_eq, "tmp/fig_03_especificacao_2.png");
// [pdi:panel-io:end]
return 0;
}
```

```
Overwriting tmp/fig_03_especificacao.cpp
```

```
!g++ -I. -std=c++17 tmp/fig_03_especificacao.cpp -o tmp/fig_03_especificacao \
  && ./tmp/fig_03_especificacao \
  && test -f "tmp/fig_03_especificacao.png" \
  || echo "  mm::show não gravou tmp/fig_03_especificacao.png"
```

```
[1] Mandril (original)
[2] Leopardo (referencia)
[3] Mandril equalizado
```

```
try:
    mm.show(
        [
            mm.read("tmp/fig_03_especificacao_0.png"),
            mm.read("tmp/fig_03_especificacao_1.png"),
            mm.read("tmp/fig_03_especificacao_2.png"),
        ],
        titles=[
            'Mandril (original)',
            'Leopardo (referencia)',
            'Mandril equalizado',
        ],
        cols=3,
    )
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + "
    tmp/fig_03_especificacao_0.png (ver a versao Python)")
```

Figure 3.9: Especificação de histograma (mapear o mandril para o perfil tonal do leopardo) precisa da CDF inversa da referência — fica só na trilha Python. Aqui, a equalização global do mandril, como comparação.

3.4 Fondements spatiaux : Voisinage, convolution et *noyaux*

Les opérations de filtrage spatial n'agissent pas sur un pixel isolé, mais sur un **voisinage** qui l'entoure. Pour cela, on utilise une petite matrice de coefficients appelée **noyau** (ou masque), qui parcourt toute l'image au moyen d'une **fenêtre glissante** (*sliding window*).

Les fenêtres les plus courantes sont de taille 3×3, 5×5 et 7×7. Dans une fenêtre 3×3, par exemple, le pixel central est traité avec ses huit voisins immédiats. À chaque position de la fenêtre, les valeurs des pixels sont combinées avec les coefficients du *noyau*, produisant une nouvelle valeur pour le pixel central.

3.4.1 Voisinage

Considérons une fenêtre 3×3 centrée sur le pixel (x, y) :

$$\begin{bmatrix} (x-1, y-1) & (x, y-1) & (x+1, y-1) \\ (x-1, y) & (x, y) & (x+1, y) \\ (x-1, y+1) & (x, y+1) & (x+1, y+1) \end{bmatrix} \tag{3.9}$$

De manière générale, une fenêtre de taille $(2a+1) \times (2b+1)$ englobe tous les pixels situés jusqu'à a positions à l'horizontale et jusqu'à b positions à la verticale par rapport au pixel central. Ainsi, une fenêtre 3×3 correspond à $a = b = 1$, une fenêtre 5×5 à $a = b = 2$, et ainsi de suite.

Mathématiquement, le voisinage est défini par

$$\mathcal{V}(x, y) = \{(x + s,\, y + t) : -a \leq s \leq a,\, -b \leq t \leq b\} \tag{3.10}$$

3.4.2 Traitement des bords

Les pixels proches des bords ont une partie de leur voisinage en dehors de l'image. Pour appliquer des filtres dans ces régions, il est nécessaire de définir comment les valeurs externes seront obtenues. Les trois stratégies les plus courantes (avec la constante équivalente d'OpenCV entre parenthèses) sont :

- **Remplissage par zéros** (`BORDER_CONSTANT`) : complète la région externe avec des zéros.
- **Réplication** (`BORDER_REPLICATE`) : répète la valeur du pixel du bord.
- **Réflexion** (`BORDER_REFLECT_101`) : reflète les pixels voisins, sans répéter celui du bord.

`mm::conv` utilise la réflexion par défaut, car elle préserve mieux la continuité des niveaux de gris et réduit les artefacts dans le traitement des bords.

L'exemple suivant compare les trois stratégies avec `mm::pad` sur une matrice 3×3. Observez comment chacune remplit les pixels externes nécessaires pour appliquer un filtre 3×3 également aux coins.

```
%%writefile tmp/mm_out_1.cpp
//| Convert this Python code to C++:
#include "morph.hpp"
#include <iostream>
#include <vector>
#include <string>

int main() {
    mm::Image img(3, 3, 1);  // 3 lignes, 3 colonnes, 1 canal
    int vals[3][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
    for (int y = 0; y < 3; ++y) {
        for (int x = 0; x < 3; ++x) {
            img.at(y, x) = static_cast<unsigned char>(vals[y][x]);
        }
    }

    std::vector<std::string> bordas = {"constant", "replicate", "reflect101"};

    std::cout << "Imagem original:" << std::endl;
    std::cout << mm::drawImg(img) << std::endl;

    for (int k : {3, 5}) {
        int b = k / 2;
        std::cout << "=== Kernel " << k << "x" << k << " (padding b=" << b << ") ===" <<
std::endl;
        for (const auto& nome : bordas) {
            std::cout << nome << std::endl;
            mm::Border border_type;
            if (nome == "constant") border_type = mm::Border::CONSTANT;
            else if (nome == "replicate") border_type = mm::Border::REPLICATE;
            else border_type = mm::Border::REFLECT101;
            std::cout << mm::drawImg(mm::pad(img, b, border_type)) << std::endl;
        }
    }

    return 0;
}
```

```
Overwriting tmp/mm_out_1.cpp
```

```
!g++ -I. -std=c++17 tmp/mm_out_1.cpp -o tmp/mm_out_1 \
  && ./tmp/mm_out_1
```

```
Imagem original:
1 2 3
4 5 6
7 8 9

=== Kernel 3x3 (padding b=1) ===
constant
0 0 0 0 0
0 1 2 3 0
0 4 5 6 0
0 7 8 9 0
0 0 0 0 0

replicate
```

```
1 1 2 3 3
1 1 2 3 3
4 4 5 6 6
7 7 8 9 9
7 7 8 9 9

reflect101
5 4 5 6 5
2 1 2 3 2
5 4 5 6 5
8 7 8 9 8
5 4 5 6 5

=== Kernel 5x5 (padding b=2) ===
constant
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 1 2 3 0 0
0 0 4 5 6 0 0
0 0 7 8 9 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0

replicate
1 1 1 2 3 3 3
1 1 1 2 3 3 3
1 1 1 2 3 3 3
4 4 4 5 6 6 6
7 7 7 8 9 9 9
7 7 7 8 9 9 9
7 7 7 8 9 9 9

reflect101
9 8 7 8 9 8 7
6 5 4 5 6 5 4
3 2 1 2 3 2 1
6 5 4 5 6 5 4
9 8 7 8 9 8 7
6 5 4 5 6 5 4
3 2 1 2 3 2 1
```

Notez que le résultat d'un filtre peut varier considérablement selon le traitement adopté pour les bords de l'image.

Dans `morph.hpp`, les fonctions de filtrage (`mm::conv`, `mm::blur`, `mm::gaussian`, `mm::laplacian`, `mm::usm`) appliquent un *padding* par **réflexion** (`mm::Border::REFLECT101`) par défaut — le même comportement que `cv2.filter2D`. La variante didactique `mm::conv0` utilise `mm::Border::KEEP` : les pixels du bord conservent leur valeur d'origine, sans filtrage. Quant à `mm::sobel` et `mm::prewitt`, elles laissent le bord à zéro (elles ne calculent que l'intérieur).

3.4.3 Corrélation vs. Convolution

Il existe deux mécanismes mathématiquement liés.

Corrélation croisée (*cross-correlation*) — le *noyau* est appliqué directement :

$$g(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t)\, f(x+s,\, y+t) \tag{3.11}$$

Convolution bidimensionnelle — le *noyau* est pivoté de 180° avant son application :

$$g(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t)\, f(x-s,\, y-t) \tag{3.12}$$

Pour les *noyaux* symétriques (gaussien, laplacien, moyenne), les deux opérations produisent des résultats identiques. Pour les *noyaux* asymétriques (Sobel, Prewitt), la différence est significative, comme le montrent les exemples suivants.

3.4.3.1 Corrélation (`mm::conv`)

```
%%writefile tmp/mm_out_2.cpp
#include "morph.hpp"
#include <iostream>

int main() {
    // imagem 4x4 (uint8) e kernel assimétrico 3x3
    mm::Image img(4, 4);
    unsigned char data_img[4][4] = {{ 1,  2,  3,  4},
                                    { 5,  6,  7,  8},
                                    { 9, 10, 11, 12},
                                    {13, 14, 15, 16}};
    for (int y = 0; y < 4; y++)
        for (int x = 0; x < 4; x++)
            img.at(y, x) = data_img[y][x];

    mm::Kernel w{{0,1,2},{0,0,0},{0,0,0}};

    mm::Image corr = mm::conv(img, w, mm::Border::CONSTANT);  // zero fora da imagem

    std::cout << "Imagem original:" << "\n";         std::cout << mm::drawImg(img) << "\n";
    std::cout << "Kernel:" << "\n";                  std::cout << mm::drawImg(w) << "\n";
    std::cout << "Resultado da correlação:" << "\n"; std::cout << mm::drawImg(corr) << "\n";

    return 0;
}
```

```
Overwriting tmp/mm_out_2.cpp
```

```
!g++ -I. -std=c++17 tmp/mm_out_2.cpp -o tmp/mm_out_2 \
  && ./tmp/mm_out_2
```

```
Imagem original:
 1  2  3  4
 5  6  7  8
 9 10 11 12
13 14 15 16

Kernel:
   0    1    2
   0    0    0
   0    0    0

Resultado da correlação:
 0  0  0  0
 5  8 11  4
17 20 23  8
29 32 35 12
```

`mm::conv` effectue une corrélation, c'est-à-dire qu'il applique le *noyau* exactement dans l'orientation fournie.

3.4.3.2 Convolution

```
%%writefile tmp/mm_out_3.cpp
#include "morph.hpp"
#include <iostream>

int main() {
    // repetidos aqui para a célula ser independente
    mm::Image img(4, 4);
    // Preenchendo a imagem com os valores
    for (int y = 0; y < 4; y++) {
        for (int x = 0; x < 4; x++) {
            img.at(y, x) = (unsigned char)(y * 4 + x + 1);
        }
    }

    mm::Kernel w{{0,1,2},{0,0,0},{0,0,0}};

    // kernel rotacionado 180° (equivale a np.rot90(w, 2))
    mm::Kernel w_conv{{0,0,0},{0,0,0},{2,1,0}};

    mm::Image conv = mm::conv(img, w_conv, mm::Border::CONSTANT);

    std::cout << "Imagem original:" << std::endl;
    std::cout << mm::drawImg(img) << std::endl;
    std::cout << "Kernel original:" << std::endl;
    std::cout << mm::drawImg(w) << std::endl;
    std::cout << "Kernel rotacionado 180°:" << std::endl;
    std::cout << mm::drawImg(w_conv) << std::endl;
    std::cout << "Resultado da convolução:" << std::endl;
    std::cout << mm::drawImg(conv) << std::endl;

    return 0;
}
```

```
Overwriting tmp/mm_out_3.cpp
```

```
!g++ -I. -std=c++17 tmp/mm_out_3.cpp -o tmp/mm_out_3 \
  && ./tmp/mm_out_3
```

```
Imagem original:
 1  2  3  4
 5  6  7  8
 9 10 11 12
13 14 15 16

Kernel original:
   0    1    2
   0    0    0
   0    0    0

Kernel rotacionado 180°:
   0    0    0
   0    0    0
   2    1    0

Resultado da convolução:
 5 16 19 22
 9 28 31 34
13 40 43 46
```

```
0  0  0  0
```

La convolution utilise le *noyau* tourné de 180°. Pour reproduire la définition mathématique de la convolution, on tourne le *noyau* (ici, `[[0,1,2],[0,0,0],[0,0,0]]` → `[[0,0,0],[0,0,0],[2,1,0]]`) avant d'appliquer `mm::conv`.

3.4.4 Le Rôle du *Noyau*

Les coefficients du *noyau* déterminent complètement l'effet produit par le filtre, comme résumé dans la Table 3.2.

Tableau 3.2: Interprétation typique des coefficients du *noyau*.

Caractéristique	Effet typique
Coefficients positifs dont la somme vaut 1	Lissage (*passe-bas*)
Somme égale à 0, avec des valeurs positives et négatives	Détection de contours (*passe-haut*)
Coefficient central positif dominant et voisins négatifs	Renforcement de la netteté
Coefficients asymétriques	Gradient directionnel

Exemples :

Lissage :

$$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$$

Détection de contours :

$$\begin{bmatrix}-1 & -1 & -1\\ -1 & 8 & -1\\ -1 & -1 & -1\end{bmatrix}$$

Renforcement de la netteté :

$$\begin{bmatrix}0 & -1 & 0\\ -1 & 5 & -1\\ 0 & -1 & 0\end{bmatrix}$$

Gradient directionnel (Sobel) :

$$\begin{bmatrix}-1 & 0 & 1\\ -2 & 0 & 2\\ -1 & 0 & 1\end{bmatrix}$$

La Figure 3.10 démontre le mécanisme étape par étape : pour chaque position de la fenêtre, on multiplie chaque coefficient du *kernel* par le pixel correspondant de la voisinage et on additionne les produits obtenus. Le résultat est exactement la valeur définie par la Équation 3.11 pour cette position de l'image. Bien que les images produites par `mm::conv0` et `cv2.filter2D` (ou `mm::conv`) soient visuellement très similaires, l'implémentation basée sur OpenCV est des milliers de fois plus rapide, comme montré ci-dessous.

⚠ Performance : boucles Python vs. opérations vectorisées

La fonction `mm::conv0` implémente la corrélation directement en Python au moyen de boucles imbriquées. Bien que cette approche soit adaptée à des fins pédagogiques, elle exécute un grand nombre d'opérations et devient lente pour des images de grande taille.
Quant à `mm::conv`, elle utilise `cv2.filter2D`, implémenté en C++ et optimisé pour les opérations

matricielles. Dans l'exemple présenté, la version vectorisée a été plus de **3000 fois plus rapide** que l'implémentation pédagogique, produisant un résultat visuellement équivalent.
Les différences numériques observées se concentrent principalement sur les bords de l'image. Dans `mm::conv0`, les pixels des bords restent inchangés, tandis que `mm::conv` utilise une stratégie de réflexion des bords (`cv2.BORDER_REFLECT_101`, valeur par défaut de `cv2.filter2D`).
C'est pourquoi `mm::conv0` doit être utilisée pour comprendre l'algorithme, tandis que `mm::conv` est l'option recommandée pour les applications pratiques.

```
%%writefile tmp/fig_03_convolucao_passo.cpp
#define MM_OUT "tmp/fig_03_convolucao_passo.png"
#include <iostream>
#include <vector>
#include <string>
#include "morph.hpp"
#include <filesystem>

int main() {
// [pdi:state-io] auto-generated - do not edit by hand
mm::Image img_leop_gray = mm::_read_state("tmp/state/img_leop_gray_12.png");
// [pdi:state-io:end]

    // Variables img_leop_gray will be provided as an already-valid mm::Image

    mm::Kernel w_mean = mm::Kernel::mean(3);
    mm::Image img_gray = img_leop_gray;

    mm::Image img_conv0 = mm::conv0(img_gray, w_mean);   // laços, bordas preservadas
    mm::Image img_conv  = mm::conv(img_gray, w_mean);    // borda refletida

    std::cout << "Correlacao no pixel central [251,251]:" << "\n";
    std::cout << "  original = " << (int)img_gray.at(251, 251) << "\n";
    std::cout << "  conv0    = " << (int)img_conv0.at(251, 251) << "\n";
    std::cout << "  conv     = " << (int)img_conv.at(251, 251) << "\n";

    mm::show(std::vector<mm::Image>{img_gray, img_conv0, img_conv},
             MM_OUT,
             std::vector<std::string>{"Original", "conv0 (laços)", "conv (vetorizado)"}, 3);


// [pdi:panel-io] auto-generated - do not edit by hand
std::filesystem::create_directories("tmp");
mm::write(img_gray, "tmp/fig_03_convolucao_passo_0.png");
mm::write(img_conv0, "tmp/fig_03_convolucao_passo_1.png");
mm::write(img_conv, "tmp/fig_03_convolucao_passo_2.png");
// [pdi:panel-io:end]
return 0;
}
```

```
Overwriting tmp/fig_03_convolucao_passo.cpp
```

```
!g++ -I. -std=c++17 tmp/fig_03_convolucao_passo.cpp -o tmp/fig_03_convolucao_passo \
  && ./tmp/fig_03_convolucao_passo \
  && test -f "tmp/fig_03_convolucao_passo.png" \
  || echo "  mm::show não gravou tmp/fig_03_convolucao_passo.png"
```

```
Correlacao no pixel central [251,251]:
  original = 104
  conv0    = 102
```

```
  conv     = 102
[1] Original
[2] conv0 (laços)
[3] conv (vetorizado)
```

```
try:
    mm.show(
        [
            mm.read("tmp/fig_03_convolucao_passo_0.png"),
            mm.read("tmp/fig_03_convolucao_passo_1.png"),
            mm.read("tmp/fig_03_convolucao_passo_2.png"),
        ],
        titles=[
            'Original',
            'conv0 (laços)',
            'conv (vetorizado)',
        ],
        cols=3,
    )
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + "
    tmp/fig_03_convolucao_passo_0.png (ver a versao Python)")
```

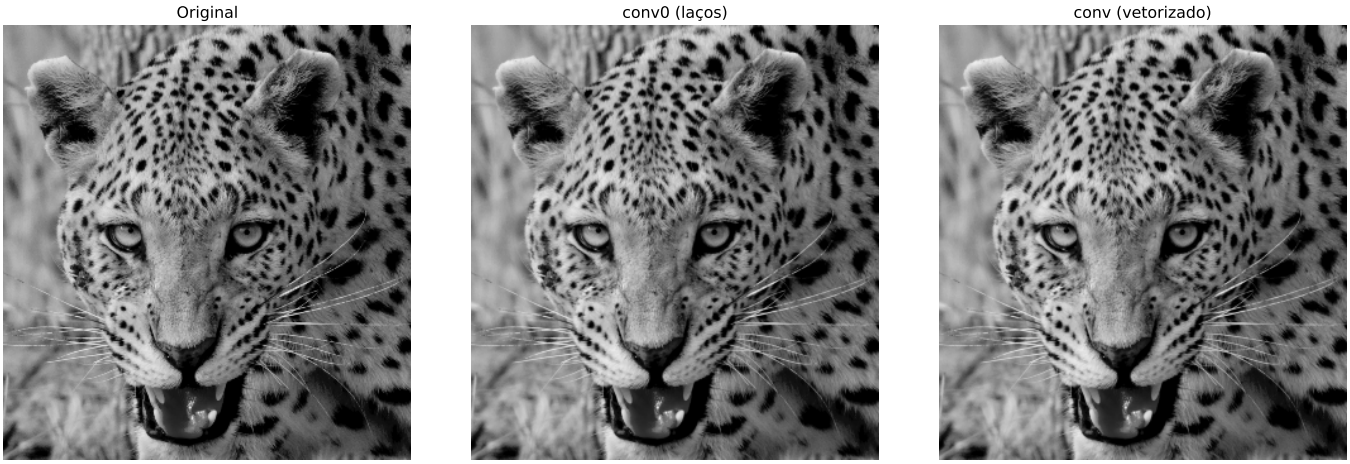


Figure 3.10: Correlação com *kernel* de média 3×3: versão didática mm::conv0 (bordas preservadas) vs. mm::conv (borda refletida). A diferença se concentra nas bordas.

```
%%writefile tmp/mm_out_4.cpp
//| echo: false
// A partir daqui o "sujeito" dos exemplos de filtragem passa a ser o
// leopardo (mais textura e bordas que o mandril).
#include "morph.hpp"
#include <filesystem>

int main() {
// [pdi:state-io] auto-generated - do not edit by hand
mm::Image img_leop = mm::_read_state("tmp/state/img_leop_12.png");
// [pdi:state-io:end]

    mm::Image img_gray = mm::gray(img_leop);

// [pdi:state-io] auto-generated - do not edit by hand
std::filesystem::create_directories("tmp/state");
mm::write(img_gray, "tmp/state/img_gray_45.png");
```

```
// [pdi:state-io:end]
return 0;
}
```

```
Overwriting tmp/mm_out_4.cpp
```

```
!g++ -I. -std=c++17 tmp/mm_out_4.cpp -o tmp/mm_out_4 \
  && ./tmp/mm_out_4
```

3.4.5 Exemple Numérique : Corrélation Pas à Pas

Pour rendre concret le mécanisme de la Équation 3.11, considérez le *noyau* de moyenne 3×3 ($a = b = 1$, tous les coefficients $= 1/9 \approx 0{,}111$) appliqué au *patch* 5×5 extrait de l'image du léopard. La Figure 3.11 affiche le *patch* avec une grille et met en évidence en jaune la fenêtre 3×3 centrée sur le pixel $[1, 1]$:

```
%%writefile tmp/fig_03_patch.cpp
#define MM_OUT "tmp/fig_03_patch.png"
#include "morph.hpp"
#include <iostream>

int main() {
// [pdi:state-io] auto-generated - do not edit by hand
mm::Image img_gray = mm::_read_state("tmp/state/img_gray_45.png");
// [pdi:state-io:end]

    //| label: fig-03-patch
    //| fig-cap: "*Patch* 5×5 extraído da imagem do leopardo (posição [250:255, 250:255]). A
    janela amarela destaca a vizinhança 3×3 centrada no pixel [1,1] onde a correlação será
    calculada."
    //| echo: true
    //| output: true

    mm::Image patch = mm::crop(img_gray, 250, 255, 250, 255);
    mm::Kernel B = mm::Kernel::ones(3);

    std::cout << "Patch 5×5 (intensidades):\n";
    std::cout << mm::drawImg(patch);

    mm::drawImgKernel(patch, B, 1, 1, MM_OUT, 40);

    return 0;
}
```

```
Overwriting tmp/fig_03_patch.cpp
```

```
!g++ -I. -std=c++17 tmp/fig_03_patch.cpp -o tmp/fig_03_patch \
  && ./tmp/fig_03_patch \
  && test -f "tmp/fig_03_patch.png" \
  || echo "  mm::show não gravou tmp/fig_03_patch.png"
```

```
Patch 5×5 (intensidades):
 94  96 103 113 115
107 104 103 107 114
 98 102 112 117 116
 81  91 112 122 118
 85  91 110 119 121
Processando pixel (x,y)=(1,1)  |  janela do kernel 3x3
 94  96 103 113 115
```

```
107 104 103 107 114
 98 102 112 117 116
 81  91 112 122 118
 85  91 110 119 121
```

```
try:
    mm.show(mm.read("tmp/fig_03_patch.png"))
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + " tmp/fig_03_patch.png (ver
    a versao Python)")
```

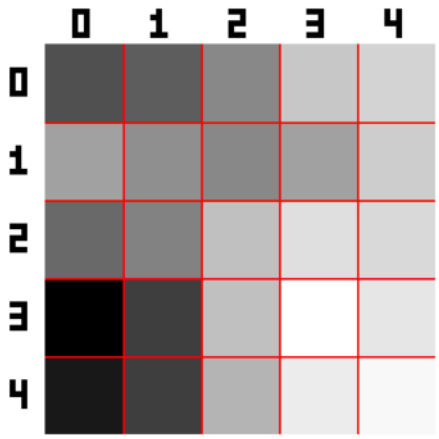


Figure 3.11: *Patch* 5×5 extraído da imagem do leopardo (posição [250:255, 250:255]). A janela amarela destaca a vizinhança 3×3 centrada no pixel [1,1] onde a correlação será calculada.

Pour illustrer le calcul de la corrélation, considérons le pixel à la position $[1, 1]$ du *patch* 5×5 montré dans la Figure 3.11.. Cette position a été choisie uniquement pour des raisons pédagogiques, car elle possède un voisinage 3×3 complet autour d'elle.

Les valeurs de ce voisinage correspondent à la sous-matrice supérieure gauche du *patch* :

$$\text{vizinhança} = \begin{bmatrix} 91 & 95 & 108 \\ 106 & 107 & 108 \\ 102 & 103 & 107 \end{bmatrix}$$

En appliquant la Équation 3.11 avec le noyau de moyenne :

$$g[1,1] = \frac{91 + 95 + 108 + 106 + 107 + 108 + 102 + 103 + 107}{9} = \frac{927}{9} = 103$$

Le résultat (103) est légèrement inférieur à la valeur originale du pixel central (107), car la moyenne intègre des voisins de moindre intensité, produisant l'effet de lissage. En pratique, l'algorithme démarre le traitement à $[0, 0]$ et répète ce même calcul pour chaque position de l'image, déplaçant la fenêtre jusqu'à couvrir tout le domaine.

3.5 Filtrage Spatial de Lissage

Les filtres de lissage (*smoothing filters*) atténuent les variations brusques d'intensité, réduisant le bruit et les détails à haute fréquence. Ce sont des filtres **passe-bas** — ils préservent les composantes de basse fréquence (structures de grande taille) et atténuent celles de haute fréquence (bruit, contours).

3.5.1 Filtre Moyenneur (*Box Filter*)

Le filtre moyenneur utilise un *noyau* uniforme de taille $n \times n$, dont tous les coefficients valent $1/n^2$:

$$w_{\text{moyenne}} = \frac{1}{n^2} \begin{bmatrix} 1 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 1 \end{bmatrix}_{n \times n} \tag{3.13}$$

Chaque pixel de sortie est la moyenne arithmétique des n^2 pixels de son voisinage. Notez que la somme des coefficients est toujours égale à 1 — la luminosité moyenne de l'image est préservée. Des *noyaux* plus grands produisent un lissage plus agressif, mais estompent progressivement les contours.

La Figure 3.12 montre l'effet du filtre de moyenne avec des *noyaux* 3×3, 7×7 et 15×15 sur un détail de l'image du léopard. Les résultats ont été obtenus avec `mm::blur`, qui implémente le filtre de moyenne via la fonction `cv2.blur`, équivalente à la convolution de l'image avec un *noyau* uniforme dont les coefficients sont $h(x, y) = 1/N^2$; de manière équivalente, le même résultat peut être obtenu avec `mm::conv`, en calculant $(g = f * h)$. À mesure que le *noyau* augmente, davantage de pixels contribuent à chaque valeur de sortie, intensifiant le lissage, réduisant le bruit et rendant les détails fins et les bords progressivement plus flous.

```
%%writefile tmp/fig_03_media.cpp
#define MM_OUT "tmp/fig_03_media.png"
//| label: fig-03-media
//| fig-cap: "Filtro de média com *kernels* de tamanho crescente (3×3, 7×7, 15×15). O
borramento das bordas aumenta com o tamanho do *kernel*."
//| echo: true
//| output: true

#include "morph.hpp"
#include <iostream>
#include <vector>
#include <string>

int main() {
// [pdi:state-io] auto-generated - do not edit by hand
mm::Image img_gray = mm::_read_state("tmp/state/img_gray_45.png");
// [pdi:state-io:end]

    // img_gray is provided

    // Détail de la région de l'œil
    int y0 = 580, y1 = 740, x0 = 680, x1 = 900;
    mm::Image img_gray_crop = mm::crop(img_gray, y0, y1, x0, x1);

    std::vector<int> sizes = {3, 7, 15};
    std::vector<mm::Image> imgs;
    imgs.push_back(img_gray_crop);
    for (int k : sizes) {
        imgs.push_back(mm::blur(img_gray_crop, k)); // ou
        //mm::conv(img_gray_crop, mm::Kernel::mean(k))
    }

    std::vector<std::string> titles;
    titles.push_back("Original");
    for (int k : sizes) {
        titles.push_back("Média " + std::to_string(k) + "×" + std::to_string(k));
    }

    mm::show(imgs, MM_OUT, titles, 4);
```

```
    return 0;
}
```

```
Overwriting tmp/fig_03_media.cpp
```

```
!g++ -I. -std=c++17 tmp/fig_03_media.cpp -o tmp/fig_03_media \
  && ./tmp/fig_03_media \
  && test -f "tmp/fig_03_media.png" \
  || echo "  mm::show não gravou tmp/fig_03_media.png"
```

```
[1] Original
[2] Média 3×3
[3] Média 7×7
[4] Média 15×15
```

```
try:
    mm.show(mm.read("tmp/fig_03_media.png"))
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + " tmp/fig_03_media.png (ver
    a versao Python)")
```

Figure 3.12: Filtro de média com *kernels* de tamanho crescente (3×3, 7×7, 15×15). O borramento das bordas aumenta com o tamanho do *kernel*.

3.5.2 Filtre gaussien

Le filtre gaussien pondère les pixels du voisinage selon une fonction gaussienne bidimensionnelle :

$$G(s,t) = \frac{1}{2\pi\sigma^2}\, e^{-\frac{s^2+t^2}{2\sigma^2}} \tag{3.14}$$

où σ est l'écart type et contrôle le rayon d'influence. Les pixels plus proches du centre ont un poids plus important ; les pixels éloignés sont progressivement ignorés.

La Figure 3.13 présente le *noyau* gaussien 5×5 généré pour $\sigma = 1$. Le *noyau* a été construit à partir du produit externe de deux vecteurs gaussiens unidimensionnels, puis normalisé pour que la somme de ses coefficients soit égale à 1. On observe que les poids les plus importants se concentrent au centre de la matrice, en décroissant radialement vers les bords. Cette distribution fait que les pixels centraux ont une plus grande influence sur le résultat du filtrage, contribuant à un lissage plus naturel et à une meilleure préservation des contours que le filtre de moyenne.

```
%%writefile tmp/fig_03_gauss_kernel.cpp
#define MM_OUT "tmp/fig_03_gauss_kernel.png"
// g++ -std=c++17 -o program program.cpp -I. -lm
#include "morph.hpp"
#include <iostream>
#include <iomanip>
```

```cpp
#include <vector>
#include <string>

int main() {
    //| label: fig-03-gauss-kernel
    //| fig-cap: "*Kernel* Gaussiano 5×5 ( =1): resposta ao impulso do filtro - pesos maiores
    no centro, decrescendo radialmente."
    //| echo: true
    //| output: true

    mm::Kernel w = mm::Kernel::gaussian(5, 1.0);   // mm::Kernel::gaussian(5, 1.0) na
morph.hpp

    std::cout << "Kernel Gaussiano 5x5 (s=1), normalizado:" << "\n";
    for (int y = 0; y < 5; y++) {
        std::cout << "  ";
        for (int x = 0; x < 5; x++) {
            std::cout << std::fixed << std::setprecision(4) << w.at(y, x);
            if (x < 4) std::cout << "  ";
        }
        std::cout << "\n";
    }
    std::cout << "Peso central [2,2] = " << std::fixed << std::setprecision(4) << w.at(2, 2)
<< "   |   canto [0,0] = " << std::fixed << std::setprecision(4) << w.at(0, 0) << "\n";

    // Visualização: resposta do filtro Gaussiano a um impulso central
    mm::Image impulso(5, 5);
    impulso.at(2, 2) = 255;
    mm::drawImgPlt(mm::gaussian(impulso, 5, 1.0), MM_OUT, 40);

    return 0;
}
```

```
Overwriting tmp/fig_03_gauss_kernel.cpp
```

```
!g++ -I. -std=c++17 tmp/fig_03_gauss_kernel.cpp -o tmp/fig_03_gauss_kernel \
  && ./tmp/fig_03_gauss_kernel \
  && test -f "tmp/fig_03_gauss_kernel.png" \
  || echo "  mm::show não gravou tmp/fig_03_gauss_kernel.png"
```

```
Kernel Gaussiano 5x5 (s=1), normalizado:
  0.0030  0.0133  0.0219  0.0133  0.0030
  0.0133  0.0596  0.0983  0.0596  0.0133
  0.0219  0.0983  0.1621  0.0983  0.0219
  0.0133  0.0596  0.0983  0.0596  0.0133
  0.0030  0.0133  0.0219  0.0133  0.0030
Peso central [2,2] = 0.1621   |   canto [0,0] = 0.0030
 3  7 11  7  3
 7 15 25 15  7
11 25 41 25 11
 7 15 25 15  7
 3  7 11  7  3
```

```
try:
    mm.show(mm.read("tmp/fig_03_gauss_kernel.png"))
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + "
    tmp/fig_03_gauss_kernel.png (ver a versao Python)")
```

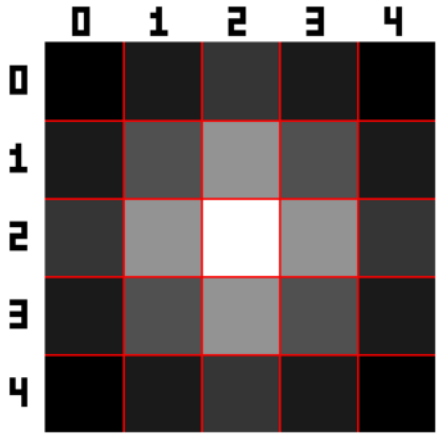


Figure 3.13: *Kernel* Gaussiano 5×5 (=1): resposta ao impulso do filtro — pesos maiores no centro, decrescendo radialmente.

i Avantage computationnelle de la séparabilité

Considérons un *noyau* carré de taille $n \times n$. Si ce filtre est séparable (comme le filtre gaussien), la convolution 2D peut être décomposée en deux convolutions 1D : une horizontale et une verticale. Dans ce cas, le coût par pixel passe d'environ $O(n^2)$ opérations (convolution 2D directe) à $O(2n)$ opérations (deux convolutions 1D). Ainsi, la complexité est réduite de manière significative, rendant le traitement plus efficace.

Comparé au filtre moyenneur, le filtre gaussien :

- **Préserve mieux les contours** — la pondération radiale lisse sans créer de transitions abruptes ;
- **N'introduit pas d'effet d'anneau** (*ringing*) dans le domaine fréquentiel, car la gaussienne est sa propre transformée de Fourier (Chapitre 5) ;
- **Est contrôlé par** σ — augmenter σ équivaut à augmenter le rayon de lissage de manière continue et prévisible.

La Figure 3.14 compare les filtres de moyenne et gaussien appliqués à l'image du léopard à l'aide d'une fenêtre 9×9. Le filtre de moyenne a été implémenté par convolution avec un *noyau* uniforme, où tous les 81 pixels du voisinage possèdent le même poids (1/81), tandis que le filtre gaussien a été obtenu avec `cv2.GaussianBlur`, utilisant des poids définis par une distribution gaussienne. Tous deux réduisent le bruit et lissent l'image, mais le filtre gaussien préserve mieux les contours et les détails locaux, comme on peut l'observer dans la région agrandie de l'œil.

La Figure 3.14 compare les filtres de moyenne et gaussien appliqués à un détail de l'image du léopard avec des *noyaux* 9×9. Le filtre de moyenne a été obtenu avec `mm::blur`, équivalent à la convolution avec un *noyau* uniforme dont les coefficients valent 1/81, tandis que le filtre gaussien a été obtenu avec `mm::gaussian`, équivalent à la convolution avec un *noyau* généré à partir d'une distribution gaussienne. Tous deux favorisent le lissage et la réduction du bruit, mais le filtre gaussien attribue un poids plus important aux pixels centraux du voisinage, préservant mieux les contours et les détails locaux, comme on peut l'observer dans la région agrandie de l'œil.

```
%%writefile tmp/fig_03_gauss.cpp
#define MM_OUT "tmp/fig_03_gauss.png"
//| label: fig-03-gauss
//| fig-cap: "Comparaison entre filtro de média e Gaussiano (*kernel* 9×9,  =0). O Gaussiano
preserva melhor as bordas, visível no detalhe do rosto."
//| echo: true
//| output: true
```

```
#include "morph.hpp"
#include <iostream>
#include <vector>
#include <string>
#include <filesystem>

int main() {
// [pdi:state-io] auto-generated - do not edit by hand
mm::Image img_gray = mm::_read_state("tmp/state/img_gray_45.png");
// [pdi:state-io:end]

    // img_gray is provided as an already-valid mm::Image

    mm::Image img_media9 = mm::blur(img_gray, 9);
    mm::Image img_gauss9 = mm::gaussian(img_gray, 9, 0);

    // Détail de la région de l'œil
    mm::Image img_gray_crop = mm::crop(img_gray, 580, 740, 680, 900);
    mm::Image img_media9_crop = mm::crop(img_media9, 580, 740, 680, 900);
    mm::Image img_gauss9_crop = mm::crop(img_gauss9, 580, 740, 680, 900);

    mm::show(
        std::vector<mm::Image>{img_gray_crop, img_media9_crop, img_gauss9_crop},
        MM_OUT,
        std::vector<std::string>{"Détail: Original", "Moyenne 9×9", "Gaussien 9×9"},
        3
    );


// [pdi:state-io] auto-generated - do not edit by hand
std::filesystem::create_directories("tmp/state");
mm::write(img_gray_crop, "tmp/state/img_gray_crop_58.png");
// [pdi:state-io:end]

// [pdi:panel-io] auto-generated - do not edit by hand
std::filesystem::create_directories("tmp");
mm::write(img_gray_crop, "tmp/fig_03_gauss_0.png");
mm::write(img_media9_crop, "tmp/fig_03_gauss_1.png");
mm::write(img_gauss9_crop, "tmp/fig_03_gauss_2.png");
// [pdi:panel-io:end]
return 0;
}
```

```
Overwriting tmp/fig_03_gauss.cpp
```

```
!g++ -I. -std=c++17 tmp/fig_03_gauss.cpp -o tmp/fig_03_gauss \
  && ./tmp/fig_03_gauss \
  && test -f "tmp/fig_03_gauss.png" \
  || echo "  mm::show não gravou tmp/fig_03_gauss.png"
```

```
[1] Détail: Original
[2] Moyenne 9×9
[3] Gaussien 9×9
```

```
try:
    mm.show(
        [
            mm.read("tmp/fig_03_gauss_0.png"),
            mm.read("tmp/fig_03_gauss_1.png"),
```

```
            mm.read("tmp/fig_03_gauss_2.png"),
        ],
        titles=[
            'Detalhe: Original',
            'Média 9×9',
            'Gaussiano 9×9',
        ],
        cols=3,
        figsize=(12, 8),
    )
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + " tmp/fig_03_gauss_0.png
    (ver a versao Python)")
```

Figure 3.14: Comparação entre filtro de média e Gaussiano (*kernel* 9×9, =0). O Gaussiano preserva melhor as bordas, visível no detalhe do rosto.

3.6 Filtrage Spatial de Rehaussement

Les filtres de rehaussement (*sharpening filters*) mettent en évidence les transitions abruptes d'intensité, augmentant la netteté et la visibilité des contours. Ce sont des filtres **passe-haut** — ils amplifient les composantes haute fréquence (contours, texture) et suppriment celles de basse fréquence (régions uniformes).

L'intuition est simple : si l'on soustrait d'une image sa version lissée (qui ne contient que les basses fréquences), ce qui reste correspond aux hautes fréquences — contours et détails. En ajoutant ce résidu à l'image originale, le contraste local augmente :

$$g = f + k\,(f - f_{\text{lisse}}), \quad k > 0 \tag{3.15}$$

La Figure 3.15 illustre ce processus sur un signal 1D synthétique avec trois structures distinctes : un échelon large, un pic fin et une rampe douce. Dans le panneau , le signal original $f(x)$; dans le , la version lissée $f_{\text{lisse}}(x)$ obtenue par moyenne mobile — on note comment le pic fin est atténué. Le panneau affiche le résidu $f - f_{\text{lisse}}$, qui ne conserve que les transitions abruptes. Enfin, le panneau montre $g(x)$: le pic, auparavant atténué, est restauré et amplifié par rapport à l'original. Ajustez k et la taille de la fenêtre pour observer le *trade-off* entre netteté et amplification du bruit.

Les filtres de rehaussement formalisent cette idée directement dans le *kernel*, sans avoir besoin de deux étapes séparées.

3.6.1 Laplacien

Le Laplacien est un opérateur isotrope de **dérivée seconde**, c'est-à-dire qu'il répond de manière identique aux variations dans toutes les directions, contrairement aux opérateurs de dérivée première, comme Sobel et Prewitt, qui sont directionnels :

Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.fr/cap03/sim-03-filtragem1d.html

Figure 3.15: Simulateur : Filtrage spatial d'accentuation 1D (Unsharp Masking et High-Boost)

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \tag{3.16}$$

Une propriété importante de la dérivée seconde est que sa valeur est **proche de zéro dans les régions uniformes** et élevée aux transitions d'intensité. Ainsi, en soustrayant le Laplacien de l'image originale, on renforce les contours et les détails, augmentant le contraste local :

$$g(x, y) = f(x, y) - \nabla^2 f(x, y) \tag{3.17}$$

Sous forme discrète, la dérivée seconde en x est approchée par $f(x+1, y) - 2f(x, y) + f(x-1, y)$, et de manière analogue en y. En sommant les deux directions, on obtient le *kernel* w_4 (4-voisins) ou w_8 (8-voisins, incluant les diagonales) :

$$w_4 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \qquad w_8 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{3.18}$$

> **i** Somme nulle et centre négatif
>
> Les deux *kernels* ont une **somme des coefficients égale à zéro** : dans les régions uniformes, la sortie est 0 — le Laplacien ne modifie pas la luminosité moyenne, il détecte uniquement les variations. Le **centre négatif** indique que le pixel est comparé à ses voisins : plus il se démarque (vers le haut ou vers le bas), plus la valeur absolue du Laplacien en ce point est grande.

Dans l'exemple suivant, le pixel central $[1, 1] = 107$ possède des voisins $\{95, 106, 108, 103\}$. Comme ces valeurs sont proches les unes des autres, la région est presque uniforme et le Laplacien retourne une valeur faible, produisant peu de rehaussement. Dans les régions de bord, où il y a des différences plus grandes entre le pixel central et ses voisins, le Laplacien prend des valeurs plus élevées (positives ou négatives), et l'opération de Équation 3.17 intensifie ces transitions.

La Figure 3.16 illustre le calcul du Laplacien avec le *noyau* w_4 dans un voisinage 3×3 mis en évidence à l'intérieur d'un *patch* 5×5. L'exemple montre la valeur obtenue par l'opérateur et le pixel correspondant rehaussé dans l'image de sortie, qui passe de 107 à 123.

```
%%writefile tmp/fig_03_laplaciano_patch.cpp
#define MM_OUT "tmp/fig_03_laplaciano_patch.png"
#include "morph.hpp"
#include <iostream>
#include <vector>

int main() {
// [pdi:state-io] auto-generated - do not edit by hand
mm::Image img_gray = mm::_read_state("tmp/state/img_gray_45.png");
// [pdi:state-io:end]

    // img_gray is provided as an already-valid mm::Image variable

    //| label: fig-03-laplaciano-patch
    //| fig-cap: "*Kernel* Laplaciano w4 sobre o *patch* 5×5: a janela amarela destaca a
    vizinhança 3×3 onde o operador de segunda derivada é calculado."
    //| echo: true
    //| output: true

    mm::Image patch = mm::crop(img_gray, 250, 255, 250, 255);
```

```
    mm::Kernel B{{1,1,1},{1,1,1},{1,1,1}};  // 3x3 kernel with all ones (laplacian
visualization)

    std::cout << "Patch 5x5 (intensidades):" << std::endl;
    std::cout << mm::drawImg(patch) << std::endl;

    mm::drawImgKernel(patch, B, 1, 1, MM_OUT, 40);

    return 0;
}
```

```
Overwriting tmp/fig_03_laplaciano_patch.cpp
```

```
!g++ -I. -std=c++17 tmp/fig_03_laplaciano_patch.cpp -o tmp/fig_03_laplaciano_patch \
  && ./tmp/fig_03_laplaciano_patch \
  && test -f "tmp/fig_03_laplaciano_patch.png" \
  || echo "  mm::show não gravou tmp/fig_03_laplaciano_patch.png"
```

```
Patch 5x5 (intensidades):
 94  96 103 113 115
107 104 103 107 114
 98 102 112 117 116
 81  91 112 122 118
 85  91 110 119 121

Processando pixel (x,y)=(1,1)  |  janela do kernel 3x3
 94  96 103 113 115
107 104 103 107 114
 98 102 112 117 116
 81  91 112 122 118
 85  91 110 119 121
```

```
try:
    mm.show(mm.read("tmp/fig_03_laplaciano_patch.png"))
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + "
    tmp/fig_03_laplaciano_patch.png (ver a versao Python)")
```

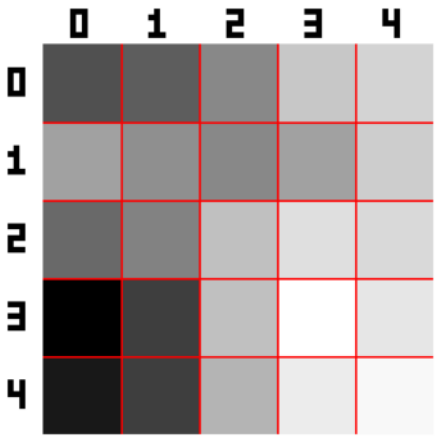


Figure 3.16: *Kernel* Laplaciano w4 sobre o *patch* 5×5: a janela amarela destaca a vizinhança 3×3 onde o operador de segunda derivada é calculado.

La Figure 3.17 compare l'application des *kernels* Laplaciens w_4 et w_8 sur un recadrage plus grand de l'image

du léopard. Pour chaque cas, sont présentées la réponse brute de l'opérateur, qui met en évidence les contours et les transitions d'intensité, ainsi que l'image obtenue après la rehaussement par soustraction du Laplacien. On observe que le *kernel* w_8, en considérant également les voisins diagonaux, produit une réponse plus intense et détecte des variations dans davantage de directions, résultant en un rehaussement légèrement plus accentué.

```
%%writefile tmp/fig_03_laplaciano.cpp
#define MM_OUT "tmp/fig_03_laplaciano.png"
//| label: fig-03-laplaciano
//| fig-cap: "Laplaciano aplicado à imagem do leopardo: resposta bruta (bordas) com w4 e w8, e
imagens realçadas pela subtração do Laplaciano. w8 é mais sensível às diagonais."
//| echo: true
//| output: true

#include "morph.hpp"
#include <vector>
#include <string>

int main() {
// [pdi:state-io] auto-generated - do not edit by hand
mm::Image img_gray_crop = mm::_read_state("tmp/state/img_gray_crop_58.png");
// [pdi:state-io:end]

    mm::Kernel w4{{0,1,0},{1,-4,1},{0,1,0}};
    mm::Kernel w8{{1,1,1},{1,-8,1},{1,1,1}};

    mm::show(
        std::vector<mm::Image>{img_gray_crop, mm::laplacian_viz(img_gray_crop, w4),
mm::laplacian(img_gray_crop, w4),
                               img_gray_crop, mm::laplacian_viz(img_gray_crop, w8),
mm::laplacian(img_gray_crop, w8)},
        MM_OUT,
        std::vector<std::string>{"Original", "Laplaciano w4", "Realce w4",
                                 "Original", "Laplaciano w8", "Realce w8"},
        3
    );

    return 0;
}
```

```
Overwriting tmp/fig_03_laplaciano.cpp
```

```
!g++ -I. -std=c++17 tmp/fig_03_laplaciano.cpp -o tmp/fig_03_laplaciano \
  && ./tmp/fig_03_laplaciano \
  && test -f "tmp/fig_03_laplaciano.png" \
  || echo "  mm::show não gravou tmp/fig_03_laplaciano.png"
```

```
[1] Original
[2] Laplaciano w4
[3] Realce w4
[4] Original
[5] Laplaciano w8
[6] Realce w8
```

```
try:
    mm.show(mm.read("tmp/fig_03_laplaciano.png"), figsize=(14, 8))
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + " tmp/fig_03_laplaciano.png
    (ver a versao Python)")
```

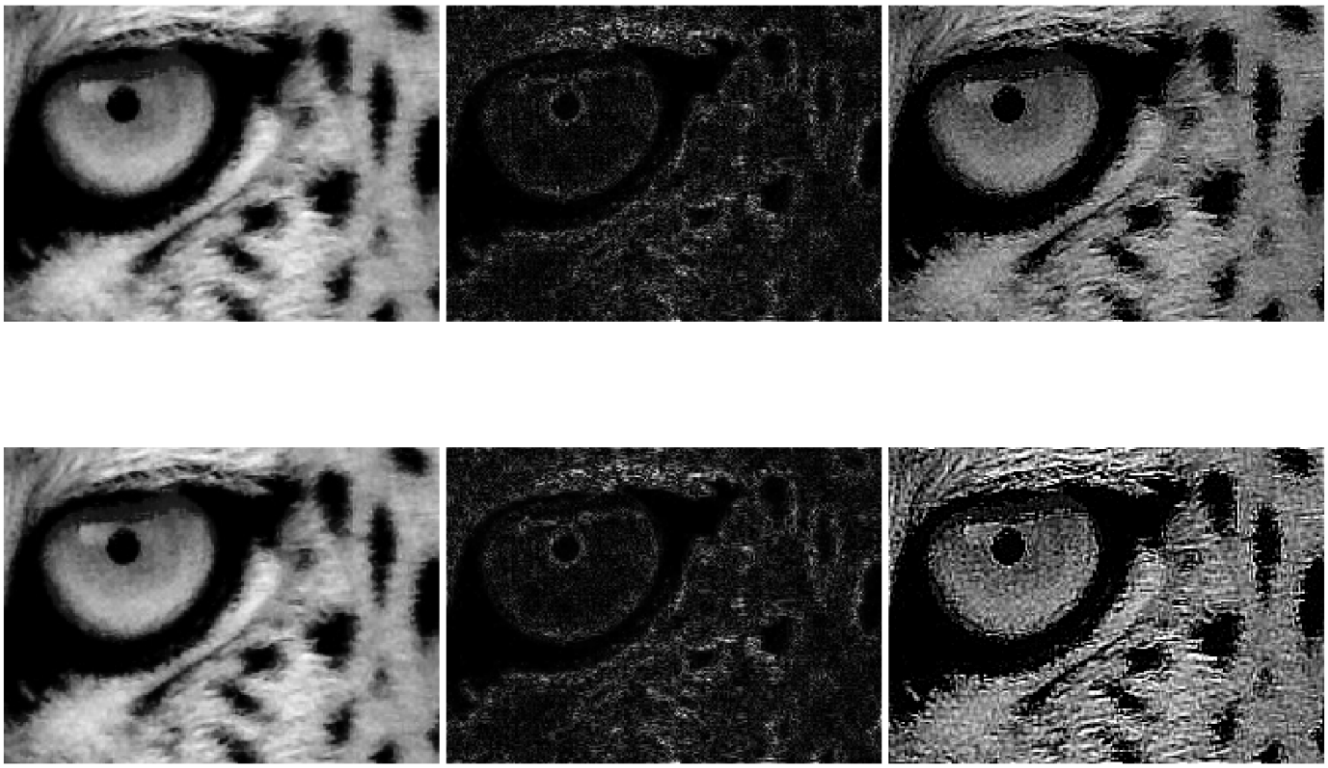


Figure 3.17: Laplaciano aplicado à imagem do leopardo: resposta bruta (bordas) com w4 e w8, e imagens realçadas pela subtração do Laplaciano. w8 é mais sensível às diagonais.

3.6.2 Opérateur de Sobel

L'opérateur de Sobel estime les **dérivées partielles du premier ordre** dans les directions horizontale et verticale. Contrairement au Laplacien (dérivée seconde), Sobel est directionnel et plus robuste au bruit, car chaque *kernel* combine une dérivée avec un lissage gaussien perpendiculaire :

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * f, \qquad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} * f \tag{3.19}$$

G_x détecte les bords **verticaux** (variation dans la direction x) ; G_y détecte les bords **horizontaux** (variation dans la direction y). Les poids $\{1, 2, 1\}$ dans la direction perpendiculaire correspondent au lissage gaussien 1D, qui réduit la sensibilité au bruit.

> **i** Sobel est une corrélation, pas une convolution
>
> Les *kernels* de Sobel sont **asymétriques** — la rotation de 180° modifie le résultat. `cv2.Sobel` implémente une corrélation croisée (comme `cv2.filter2D`). Pour obtenir la dérivée directionnelle correcte, les signes sont déjà définis pour la corrélation : G_x retourne des valeurs positives lorsque l'intensité croît de la gauche vers la droite.

La magnitude du **gradient** combine les deux composantes, représentant la force du bord indépendamment de la direction :

$$|\nabla f| = \sqrt{G_x^2 + G_y^2} \tag{3.20}$$

Et la **direction** du gradient (perpendiculaire au bord) est :

$$\theta = \arctan\left(\frac{G_y}{G_x}\right) \tag{3.21}$$

Pour illustrer numériquement, on calcule G_x et G_y manuellement au pixel central $[1, 1]$ du *patch* 5×5 :

La valeur réduite de $|\nabla f|$ dans ce *patch* confirme que la région est presque uniforme, car le gradient prend des valeurs élevées uniquement là où il y a des changements significatifs d'intensité. La Figure 3.18 applique l'opérateur de Sobel à un recadrage plus large de l'image du léopard. Sont présentées les réponses horizontale (G_x) et verticale (G_y), obtenues par convolution avec les *kernels* de Sobel respectifs, ainsi que la magnitude $|\nabla f|$, calculée à partir de la combinaison des deux. Alors que G_x met en évidence les bords verticaux et G_y les bords horizontaux, la magnitude souligne les bords dans toutes les directions.

```
%%writefile tmp/fig_03_sobel.cpp
#define MM_OUT "tmp/fig_03_sobel.png"
#include "morph.hpp"
#include <vector>
#include <string>
#include <filesystem>

//| label: fig-03-sobel
//| fig-cap: "Opérateur de Sobel sur l'image du léopard : la magnitude | f| combine les
gradients horizontal et vertical, révélant toutes les arêtes. La décomposition Gx/Gy avec
signe reste dans la piste Python - mm::sobel retourne la magnitude déjà avec clip."
//| echo: true
//| output: true

int main() {
// [pdi:state-io] auto-generated - do not edit by hand
mm::Image img_gray_crop = mm::_read_state("tmp/state/img_gray_crop_58.png");
// [pdi:state-io:end]

    mm::Image mag = mm::sobel(img_gray_crop);

    mm::show(std::vector<mm::Image>{img_gray_crop, mag},
             MM_OUT,
             std::vector<std::string>{"Original", "Magnitude |grad f| (mm.sobel)"}, 2);


// [pdi:panel-io] auto-generated - do not edit by hand
std::filesystem::create_directories("tmp");
mm::write(img_gray_crop, "tmp/fig_03_sobel_0.png");
mm::write(mag, "tmp/fig_03_sobel_1.png");
// [pdi:panel-io:end]
return 0;
}
```

```
Overwriting tmp/fig_03_sobel.cpp
```

```
!g++ -I. -std=c++17 tmp/fig_03_sobel.cpp -o tmp/fig_03_sobel \
  && ./tmp/fig_03_sobel \
  && test -f "tmp/fig_03_sobel.png" \
  || echo "  mm::show não gravou tmp/fig_03_sobel.png"
```

```
[1] Original
[2] Magnitude |grad f| (mm.sobel)
```

```
try:
    mm.show(
        [
            mm.read("tmp/fig_03_sobel_0.png"),
            mm.read("tmp/fig_03_sobel_1.png"),
        ],
        titles=[
            'Original',
            'Magnitude |grad f| (mm.sobel)',
        ],
        cols=2,
    )
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + " tmp/fig_03_sobel_0.png
    (ver a versao Python)")
```

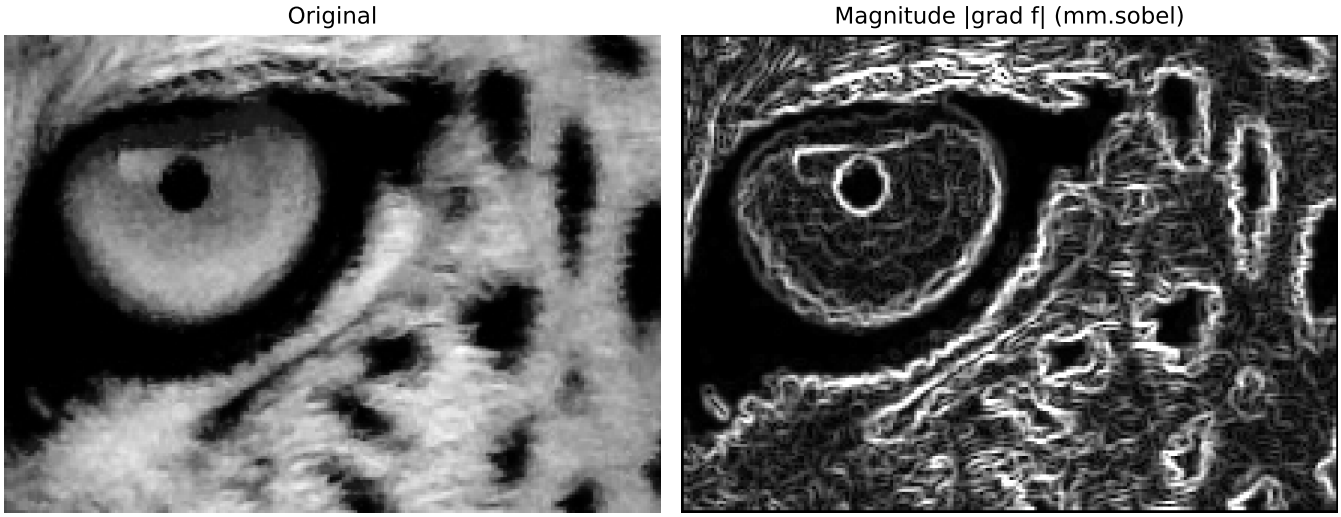


Figure 3.18: Operador de Sobel na imagem do leopardo: a magnitude | f| combina os gradientes horizontal e vertical, revelando todas as bordas. A decomposição Gx/Gy com sinal fica na trilha Python — mm::sobel devolve a magnitude já com clip.

3.6.3 Opérateur de Prewitt

L'opérateur de Prewitt est structurellement identique à Sobel, mais remplace la pondération gaussienne $\{1, 2, 1\}$ par des poids uniformes $\{1, 1, 1\}$:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} * f, \qquad G_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} * f \tag{3.22}$$

L'amplitude et la direction du gradient suivent les mêmes équations que Sobel (Équation 3.20 et Équation 3.21). La différence pratique est que Prewitt est légèrement plus sensible au bruit — le lissage perpendiculaire uniforme accorde moins de poids au pixel central de la ligne — mais il est plus simple sur le plan computationnel. Sur des images à faible bruit, les résultats sont équivalents.

```
%%writefile tmp/fig_03_prewitt.cpp
#define MM_OUT "tmp/fig_03_prewitt.png"
```

```
//| label: fig-03-prewitt
//| fig-cap: "Operador de Prewitt: magnitude do gradiente, comparável ao Sobel mas sem a
ponderação central. mm::prewitt devolve |∇f| com clip."
//| echo: true
//| output: true
#include "morph.hpp"
#include <iostream>
#include <vector>
#include <string>

int main() {
// [pdi:state-io] auto-generated - do not edit by hand
mm::Image img_gray_crop = mm::_read_state("tmp/state/img_gray_crop_58.png");
// [pdi:state-io:end]

    mm::show(std::vector<mm::Image>{img_gray_crop, mm::prewitt(img_gray_crop)},
             MM_OUT,
             std::vector<std::string>{"Original", "Magnitude |grad f| (mm.prewitt)"},
             2);
    return 0;
}
```

```
Overwriting tmp/fig_03_prewitt.cpp
```

```
!g++ -I. -std=c++17 tmp/fig_03_prewitt.cpp -o tmp/fig_03_prewitt \
  && ./tmp/fig_03_prewitt \
  && test -f "tmp/fig_03_prewitt.png" \
  || echo "  mm::show não gravou tmp/fig_03_prewitt.png"
```

```
[1] Original
[2] Magnitude |grad f| (mm.prewitt)
```

```
try:
    mm.show(mm.read("tmp/fig_03_prewitt.png"))
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + " tmp/fig_03_prewitt.png
    (ver a versao Python)")
```

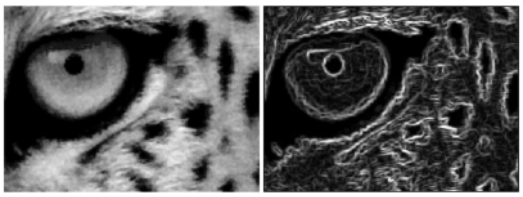


Figure 3.19: Operador de Prewitt: magnitude do gradiente, comparável ao Sobel mas sem a ponderação central. mm::prewitt devolve |∇f| com clip.

3.6.4 *Unsharp Masking* (USM)

L'*Unsharp Masking* est une technique classique de renforcement de netteté issue de la photographie analogique, aujourd'hui largement utilisée dans les logiciels de retouche d'images. L'idée centrale est d'extraire les **composantes haute fréquence** de l'image (contours et détails) et de les additionner à l'originale avec un poids k :

Tableau 3.3: Étapes de l'*Unsharp Masking*.

Étape	Opération	Description
1	$\bar{f} = f * G_\sigma$	Lissage par gaussienne — conserve les basses fréquences
2	$m = f - \bar{f}$	Masque : différence = hautes fréquences (contours)
3	$g = f + k \cdot m$	Somme pondérée du masque à l'originale

En remplaçant l'étape 2 dans l'étape 3, on obtient l'expression compacte :

$$g = f + k\,(f - f * G_\sigma) = (1 + k)\,f - k\,(f * G_\sigma) \tag{3.23}$$

Le paramètre k contrôle l'intensité du renforcement :

- $k = 0$: aucun renforcement ($g = f$) ;
- $k = 1$: USM classique — double la contribution des hautes fréquences ;
- $k > 1$: *High Boost Filtering* — amplification au-delà du double, utile pour les images très floues.

> ⚠ Amplification du bruit
>
> L'USM ne distingue pas les contours du bruit — les deux sont des composantes haute fréquence. Pour un k élevé, le bruit présent dans l'image est amplifié en même temps que les contours. Il est donc recommandé d'appliquer un léger lissage avant l'USM sur les images bruitées, ou d'utiliser un σ petit dans la gaussienne.

Pour illustrer les étapes de l'USM, la Figure 3.20 applique la méthode à un *patch* 30×30 de l'image du léopard, en utilisant $\sigma = 1$ et $k = 1$. Initialement, l'image est lissée par un filtre gaussien. Ensuite, le masque haute fréquence est obtenu par la différence entre l'image originale et l'image lissée. Enfin, ce masque est ajouté à l'image originale, renforçant les contours et les détails. La figure présente les trois étapes du processus et le résultat final du rehaussement.

```
%%writefile tmp/fig_03_usm2.cpp
#define MM_OUT "tmp/fig_03_usm2.png"
//| label: fig-03-usm2
//| fig-cap: "Realce por *Unsharp Masking* num *patch* do leopardo: mm::usm faz suavização
Gaussiana, subtrai da original (máscara de alta frequência) e reintroduz a máscara realçada."
//| echo: true
//| output: true

#include "morph.hpp"
#include <iostream>
#include <vector>
#include <string>
#include <filesystem>

int main() {
// [pdi:state-io] auto-generated - do not edit by hand
mm::Image img_gray_crop = mm::_read_state("tmp/state/img_gray_crop_58.png");
// [pdi:state-io:end]

    mm::Image patch = mm::crop(img_gray_crop, 35, 65, 45, 75);

    mm::Image p_suave = mm::gaussian(patch, 7, 1.0);   // suavização Gaussiana
    mm::Image p_usm   = mm::usm(patch, 1.0);           // realce completo (k = 1.0)
```

```
    mm::show(std::vector<mm::Image>{patch, p_suave, p_usm},
             MM_OUT,
             std::vector<std::string>{"Patch original", "Suavizado ( =1)", "Realçado USM
(k=1)"}, 3);


// [pdi:panel-io] auto-generated - do not edit by hand
std::filesystem::create_directories("tmp");
mm::write(patch, "tmp/fig_03_usm2_0.png");
mm::write(p_suave, "tmp/fig_03_usm2_1.png");
mm::write(p_usm, "tmp/fig_03_usm2_2.png");
// [pdi:panel-io:end]
return 0;
}
```

```
Overwriting tmp/fig_03_usm2.cpp
```

```
!g++ -I. -std=c++17 tmp/fig_03_usm2.cpp -o tmp/fig_03_usm2 \
  && ./tmp/fig_03_usm2 \
  && test -f "tmp/fig_03_usm2.png" \
  || echo "  mm::show não gravou tmp/fig_03_usm2.png"
```

```
[1] Patch original
[2] Suavizado ( =1)
[3] Realçado USM (k=1)
```

```
try:
    mm.show(
        [
            mm.read("tmp/fig_03_usm2_0.png"),
            mm.read("tmp/fig_03_usm2_1.png"),
            mm.read("tmp/fig_03_usm2_2.png"),
        ],
        titles=[
            'Patch original',
            'Suavizado ( =1)',
            'Realçado USM (k=1)',
        ],
        cols=3,
    )
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + " tmp/fig_03_usm2_0.png (ver
    a versao Python)")
```

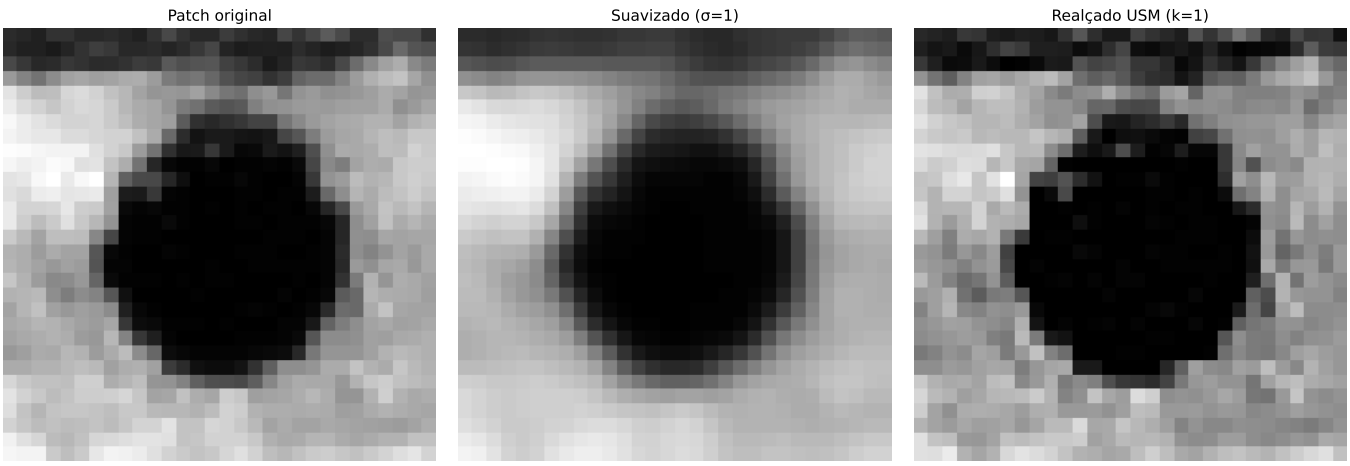


Figure 3.20: Realce por *Unsharp Masking* num *patch* do leopardo: mm::usm faz suavização Gaussiana, subtrai da original (máscara de alta frequência) e reintroduz a máscara realçada.

La Figure 3.21 applique la méthode USM à un recadrage plus large de l'image du léopard en utilisant $\sigma = 1$ et différentes valeurs du facteur de gain k. Dans tous les cas, le masque haute fréquence est obtenu par la différence entre l'image originale et sa version lissée par un filtre gaussien. Le paramètre k contrôle l'intensité du rehaussement : des valeurs plus faibles produisent une augmentation subtile de la netteté, tandis que des valeurs plus élevées renforcent progressivement les contours et les détails. On observe que, pour des valeurs élevées de k, des halos apparaissent autour des contours et le bruit présent dans l'image se trouve amplifié.

```
%%writefile tmp/fig_03_usm.cpp
#define MM_OUT "tmp/fig_03_usm.png"
//| label: fig-03-usm
//| fig-cap: "*Unsharp Masking* na imagem do leopardo com  =1 e k de 0.5 a 8.0. Para k>2
surgem halos nas bordas e o ruído de fundo aparece."
//| echo: true
//| output: true

#include "morph.hpp"
#include <iostream>
#include <vector>
#include <string>

int main() {
// [pdi:state-io] auto-generated - do not edit by hand
mm::Image img_gray_crop = mm::_read_state("tmp/state/img_gray_crop_58.png");
// [pdi:state-io:end]

    mm::show(
        std::vector<mm::Image>{img_gray_crop,
            mm::usm(img_gray_crop, 0.5), mm::usm(img_gray_crop, 1.0),
            mm::usm(img_gray_crop, 3.0), mm::usm(img_gray_crop, 5.0),
            mm::usm(img_gray_crop, 8.0)},
        MM_OUT,
        std::vector<std::string>{"Original", "USM k=0.5", "USM k=1.0", "USM k=3.0", "USM
k=5.0", "USM k=8.0"},
        3
    );
    return 0;
}
```

```
Overwriting tmp/fig_03_usm.cpp
```

```
!g++ -I. -std=c++17 tmp/fig_03_usm.cpp -o tmp/fig_03_usm \
  && ./tmp/fig_03_usm \
  && test -f "tmp/fig_03_usm.png" \
  || echo "  mm::show não gravou tmp/fig_03_usm.png"
```

```
[1] Original
[2] USM k=0.5
[3] USM k=1.0
[4] USM k=3.0
[5] USM k=5.0
[6] USM k=8.0
```

```
try:
    mm.show(mm.read("tmp/fig_03_usm.png"))
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + " tmp/fig_03_usm.png (ver a
    versao Python)")
```

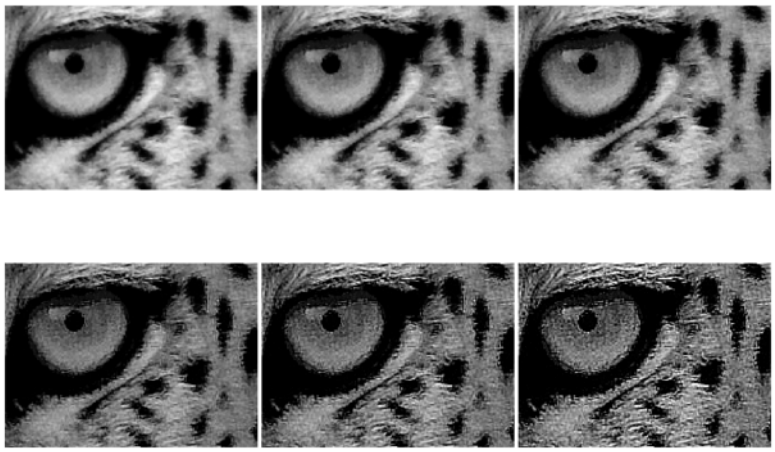


Figure 3.21: *Unsharp Masking* na imagem do leopardo com =1 e k de 0.5 a 8.0. Para k>2 surgem halos nas bordas e o ruído de fundo aparece.

3.6.5 Détecteur de Canny

Canny combine quatre étapes en séquence — lissage gaussien, gradient de Sobel, suppression des non-maxima et hystérésis par double seuil — pour produire des contours **fins, binaires et connectés**. Contrairement à Sobel et Prewitt, le résultat n'est pas une carte de gradient continue, mais un masque où chaque pixel est un contour ou ne l'est pas.

Le paramètre central est la paire de seuils (T_{low}, T_{high}). Les pixels dont le gradient est supérieur à T_{high} sont des contours certains ; ceux en dessous de T_{low} sont écartés. Les pixels ambigus — entre les deux seuils — sont décidés par **hystérésis** : ils deviennent des contours s'ils sont connectés à un contour certain, et sont écartés dans le cas contraire. Cela évite à la fois la perte de segments faibles de contours réels et l'inclusion de bruit isolé. Une heuristique courante est $T_{high} = 3 \times T_{low}$.

```
%%writefile tmp/fig_03_canny.cpp
#define MM_OUT "tmp/fig_03_canny.png"
#include <iostream>
#include <vector>
#include <string>
#include "morph.hpp"
```

```
using namespace std;

//| label: fig-03-canny
//| fig-cap: "Detecteur de Canny avec différentes paires de seuil : les seuils bas capturent
plus de bords (y compris le bruit) ; les seuils élevés ne retiennent que les bords les plus
forts."
//| echo: true
//| output: true

int main() {
// [pdi:state-io] auto-generated - do not edit by hand
mm::Image img_gray_crop = mm::_read_state("tmp/state/img_gray_crop_58.png");
// [pdi:state-io:end]

    mm::show(
        vector<mm::Image>{img_gray_crop,
                          mm::canny(img_gray_crop, 30, 90),
                          mm::canny(img_gray_crop, 60, 180),
                          mm::canny(img_gray_crop, 120, 240)},
        MM_OUT,
        vector<string>{"Original", "Canny (30/90)", "Canny (60/180)", "Canny (120/240)"},
        4
    );

    return 0;
}
```

```
Overwriting tmp/fig_03_canny.cpp
```

```
!g++ -I. -std=c++17 tmp/fig_03_canny.cpp -o tmp/fig_03_canny \
  && ./tmp/fig_03_canny \
  && test -f "tmp/fig_03_canny.png" \
  || echo "  mm::show não gravou tmp/fig_03_canny.png"
```

```
[1] Original
[2] Canny (30/90)
[3] Canny (60/180)
[4] Canny (120/240)
```

```
try:
    mm.show(mm.read("tmp/fig_03_canny.png"))
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + " tmp/fig_03_canny.png (ver
    a versao Python)")
```

Figure 3.22: Detector de Canny com diferentes pares de limiar: limiares baixos capturam mais bordas (inclusive ruído); limiares altos retêm apenas as bordas mais fortes.

Choix des seuils

Une heuristique courante est $T_{high} = 3 \times T_{low}$. Les valeurs typiques dépendent de la plage de gradient de l'image — `cv2.Canny` accepte des valeurs absolues dans $[0, 255]$. Pour des images à contraste variable, calculer les seuils à partir des percentiles de la magnitude de Sobel est plus robuste que des valeurs fixes.

3.7 Filtres d'Ordre : Filtre Médian

Les filtres d'ordre (*order-statistic filters*) remplacent le pixel central par la valeur d'un **centile** de la distribution des intensités du voisinage — contrairement aux filtres linéaires, qui calculent des combinaisons pondérées. Le plus important est le **filtre médian**.

3.7.1 Bruit Impulsif : Sel et Poivre

Le bruit **sel et poivre** (*salt-and-pepper noise*) remplace des pixels aléatoires par des valeurs extrêmes : 0 (poivre, noir) ou 255 (sel, blanc). Il est courant dans la transmission d'images avec des erreurs de bits et dans les caméras équipées de capteurs défectueux.

Pour comprendre pourquoi les filtres linéaires échouent, considérons un voisinage 3×3 où un seul pixel a été corrompu à 255 :

$$\text{voisinage} = \begin{bmatrix} 102 & 98 & 105 \\ 100 & \mathbf{255} & 97 \\ 103 & 99 & 101 \end{bmatrix}$$

Tableau 3.4: Moyenne vs. médiane avec un pixel corrompu. La médiane ignore la valeur aberrante ; la moyenne est décalée d'environ 40 niveaux.

Méthode	Calcul	Résultat
Moyenne	(102+98+…+255+…+101)/9	140
Médiane	{97,98,99,100,**101**,102,103,105,255}	101

Pourquoi les filtres de moyenne échouent-ils avec le bruit impulsif ?

La moyenne est sensible aux ***valeurs aberrantes*** — un seul pixel avec une valeur de 255 dans un voisinage de valeur 100 élève la sortie à 140, propageant le bruit dans l'image. La médiane, étant un **estimateur robuste**, sélectionne la valeur centrale de la distribution ordonnée, écartant naturellement les extrêmes sans aucun ajustement spécial.

L'exemple suivant illustre le comportement de la moyenne et de la médiane en présence d'un pixel corrompu par un bruit impulsif. On observe que la moyenne est fortement influencée par la valeur extrême (255), produisant une estimation éloignée des valeurs prédominantes du voisinage. Quant à la médiane, elle reste proche de la valeur originale de la région, mettant en évidence sa plus grande robustesse aux *valeurs aberrantes* et justifiant son utilisation dans la suppression du bruit sel et poivre.

La Figure 3.23 présente l'effet du bruit sel et poivre à différentes densités. Le bruit a été généré en remplaçant aléatoirement une fraction des pixels par des valeurs minimales (0, poivre) et maximales (255, sel). À mesure que la densité augmente de 2 % à 10 %, la quantité de pixels corrompus croît, rendant la dégradation visuelle plus évidente et rendant plus difficile la perception des détails de l'image.

```
%%writefile tmp/fig_03_ruido.cpp
#define MM_OUT "tmp/fig_03_ruido.png"
//| label: fig-03-ruido
```

```
//| fig-cap: "Ruído sal e pimenta com densidades crescentes (2%, 5%, 10%): metade dos pixels
corrompidos vira sal (255), metade pimenta (0)."
//| echo: true
//| output: true

#include "morph.hpp"
#include <random>
#include <vector>
#include <string>
#include <filesystem>

mm::Image salt_pepper(const mm::Image& img, double prob) {
    mm::Image out = img;
    int h = out.h;
    int w = out.w;
    std::mt19937 rng(42);
    std::uniform_int_distribution<int> dist_y(0, h - 1);
    std::uniform_int_distribution<int> dist_x(0, w - 1);
    std::uniform_real_distribution<double> dist_r(0.0, 1.0);
    int n = static_cast<int>(prob * h * w);
    for (int i = 0; i < n; ++i) {
        int y = dist_y(rng);
        int x = dist_x(rng);
        out.at(y, x) = (dist_r(rng) < 0.5) ? 0 : 255;
    }
    return out;
}

int main() {
// [pdi:state-io] auto-generated - do not edit by hand
mm::Image img_gray_crop = mm::_read_state("tmp/state/img_gray_crop_58.png");
// [pdi:state-io:end]

    mm::Image n2 = salt_pepper(img_gray_crop, 0.02);
    mm::Image n5 = salt_pepper(img_gray_crop, 0.05);
    mm::Image n10 = salt_pepper(img_gray_crop, 0.10);

    mm::show(std::vector<mm::Image>{img_gray_crop, n2, n5, n10},
             MM_OUT,
             std::vector<std::string>{"Original", "Ruido 2%", "Ruido 5%", "Ruido 10%"}, 4);


// [pdi:panel-io] auto-generated - do not edit by hand
std::filesystem::create_directories("tmp");
mm::write(img_gray_crop, "tmp/fig_03_ruido_0.png");
mm::write(n2, "tmp/fig_03_ruido_1.png");
mm::write(n5, "tmp/fig_03_ruido_2.png");
mm::write(n10, "tmp/fig_03_ruido_3.png");
// [pdi:panel-io:end]
return 0;
}
```

```
Overwriting tmp/fig_03_ruido.cpp
```

```
!g++ -I. -std=c++17 tmp/fig_03_ruido.cpp -o tmp/fig_03_ruido \
  && ./tmp/fig_03_ruido \
  && test -f "tmp/fig_03_ruido.png" \
  || echo "  mm::show não gravou tmp/fig_03_ruido.png"
```

```
[1] Original
[2] Ruido 2%
[3] Ruido 5%
[4] Ruido 10%
```

```
try:
    mm.show(
        [
            mm.read("tmp/fig_03_ruido_0.png"),
            mm.read("tmp/fig_03_ruido_1.png"),
            mm.read("tmp/fig_03_ruido_2.png"),
            mm.read("tmp/fig_03_ruido_3.png"),
        ],
        titles=[
            'Original',
            'Ruido 2%',
            'Ruido 5%',
            'Ruido 10%',
        ],
        cols=4,
    )
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + " tmp/fig_03_ruido_0.png
    (ver a versao Python)")
```

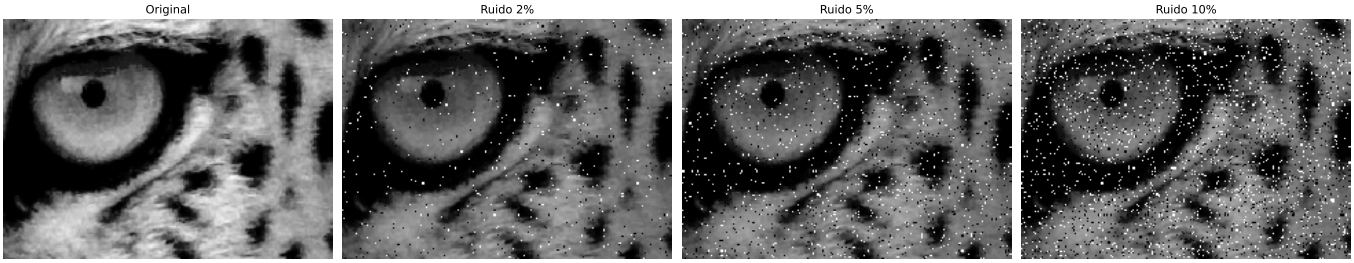


Figure 3.23: Ruído sal e pimenta com densidades crescentes (2%, 5%, 10%): metade dos pixels corrompidos vira sal (255), metade pimenta (0).

3.7.2 Filtre Médian

Le filtre médian remplace chaque pixel par la **valeur médiane** des pixels de son voisinage $n \times n$:

$$g(x, y) = \text{med}_{(s,t)\in\mathcal{V}_n}\{f(x+s, y+t)\} \tag{3.24}$$

La valeur médiane est celle qui occupe la position centrale lorsque les n^2 valeurs du voisinage sont triées. Pour une fenêtre 3×3 ($n^2 = 9$ pixels), la médiane est la 5 valeur de la séquence ordonnée.

Pour illustrer, considérons le même *patch* 5×5 avec un pixel corrompu artificiellement en $[1, 1]$:

L'exemple confirme : même avec le pixel corrompu à 255, la médiane retourne la valeur centrale correcte — le *outlier* occupe la dernière position dans le tri et est écarté naturellement.

Parce qu'elle est basée sur le tri et non sur la somme, la médiane possède trois propriétés fondamentales qui la distinguent des filtres linéaires :

- **Robuste** au bruit impulsionnel — les outliers vont aux extrémités de la séquence triée et n'affectent pas la valeur centrale ;
- **Préservatrice des contours** — les transitions abruptes d'intensité sont conservées, car la médiane sélectionne une valeur qui existe déjà dans le voisinage, sans créer de nouveaux niveaux intermédiaires ;

- **Non linéaire** — elle ne peut pas être exprimée comme une convolution, donc `mm::conv` ne s'applique pas ; on utilise `cv2.medianBlur`.

La Figure 3.24 compare différentes techniques de suppression du bruit sel et poivre appliquées à une image contenant 10 % de pixels corrompus. Les filtres gaussien, moyen, médian, bilatéral et morphologique (chapitre suivant) ont été évalués, permettant ainsi d'observer le compromis entre la suppression du bruit et la préservation des détails. En général, les filtres de moyenne et gaussien réduisent le bruit, mais tendent à estomper les contours, tandis que le filtre médian offre de meilleures performances pour le bruit impulsionnel. Le filtre bilatéral préserve mieux les contours, et le filtre morphologique élimine une grande partie des pixels corrompus sans dégrader excessivement la structure de l'image.

```
%%writefile tmp/fig_03_ruido_filtros.cpp
#define MM_OUT "tmp/fig_03_ruido_filtros.png"
//#| label: fig-03-ruido-filtros
//#| fig-cap: "Filtros para ruído sal e pimenta (10%): Gaussiano, Média e Mediana. Bilateral e
morfológico (open+close) ficam só na trilha Python - bilateral não tem equivalente em
morph.hpp e morfologia é do próximo capítulo."
//#| echo: true
//#| output: true
#include "morph.hpp"
#include <random>
#include <vector>
#include <string>
#include <algorithm>
#include <cmath>
#include <filesystem>

// Função para adicionar ruído sal e pimenta
mm::Image salt_pepper(const mm::Image& img, double prob) {
    mm::Image out = img;  // cópia
    int h = img.h, w = img.w;
    std::mt19937 rng(42);  // semente fixa, mesma do original
    std::uniform_int_distribution<int> dist_y(0, h - 1);
    std::uniform_int_distribution<int> dist_x(0, w - 1);
    std::uniform_real_distribution<double> dist_rand(0.0, 1.0);

    int n_pixels = static_cast<int>(prob * h * w);
    for (int i = 0; i < n_pixels; ++i) {
        int y = dist_y(rng);
        int x = dist_x(rng);
        out.at(y, x) = (dist_rand(rng) < 0.5) ? 0 : 255;
    }
    return out;
}

int main() {
// [pdi:state-io] auto-generated - do not edit by hand
mm::Image img_gray_crop = mm::_read_state("tmp/state/img_gray_crop_58.png");
// [pdi:state-io:end]

    // img_gray_crop é fornecido automaticamente

    // Aplicar ruído sal e pimenta
    mm::Image noisy = salt_pepper(img_gray_crop, 0.10);

    // Filtros para comparação
    mm::Image f_gauss   = mm::gaussian(noisy, 5, 1.0);
    mm::Image f_media   = mm::blur(noisy, 5);
    mm::Image f_median3 = mm::median(noisy, 3);
    mm::Image f_median5 = mm::median(noisy, 5);
```

```
    // Exibir os resultados
    mm::show(
        std::vector<mm::Image>{img_gray_crop, noisy, f_gauss, f_media, f_median3, f_median5},
        MM_OUT,
        std::vector<std::string>{"Original", "Ruido 10%", "Gaussiano 5x5", "Media 5x5",
                                 "Mediana 3x3", "Mediana 5x5"},
        3
    );


// [pdi:panel-io] auto-generated - do not edit by hand
std::filesystem::create_directories("tmp");
mm::write(img_gray_crop, "tmp/fig_03_ruido_filtros_0.png");
mm::write(noisy, "tmp/fig_03_ruido_filtros_1.png");
mm::write(f_gauss, "tmp/fig_03_ruido_filtros_2.png");
mm::write(f_media, "tmp/fig_03_ruido_filtros_3.png");
mm::write(f_median3, "tmp/fig_03_ruido_filtros_4.png");
mm::write(f_median5, "tmp/fig_03_ruido_filtros_5.png");
// [pdi:panel-io:end]
return 0;
}
```

```
Overwriting tmp/fig_03_ruido_filtros.cpp
```

```
!g++ -I. -std=c++17 tmp/fig_03_ruido_filtros.cpp -o tmp/fig_03_ruido_filtros \
  && ./tmp/fig_03_ruido_filtros \
  && test -f "tmp/fig_03_ruido_filtros.png" \
  || echo "  mm::show não gravou tmp/fig_03_ruido_filtros.png"
```

```
[1] Original
[2] Ruido 10%
[3] Gaussiano 5x5
[4] Media 5x5
[5] Mediana 3x3
[6] Mediana 5x5
```

```
try:
    mm.show(
        [
            mm.read("tmp/fig_03_ruido_filtros_0.png"),
            mm.read("tmp/fig_03_ruido_filtros_1.png"),
            mm.read("tmp/fig_03_ruido_filtros_2.png"),
            mm.read("tmp/fig_03_ruido_filtros_3.png"),
            mm.read("tmp/fig_03_ruido_filtros_4.png"),
            mm.read("tmp/fig_03_ruido_filtros_5.png"),
        ],
        titles=[
            'Original',
            'Ruido 10%',
            'Gaussiano 5x5',
            'Media 5x5',
            'Mediana 3x3',
            'Mediana 5x5',
        ],
        cols=3,
    )
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + "
    tmp/fig_03_ruido_filtros_0.png (ver a versao Python)")
```

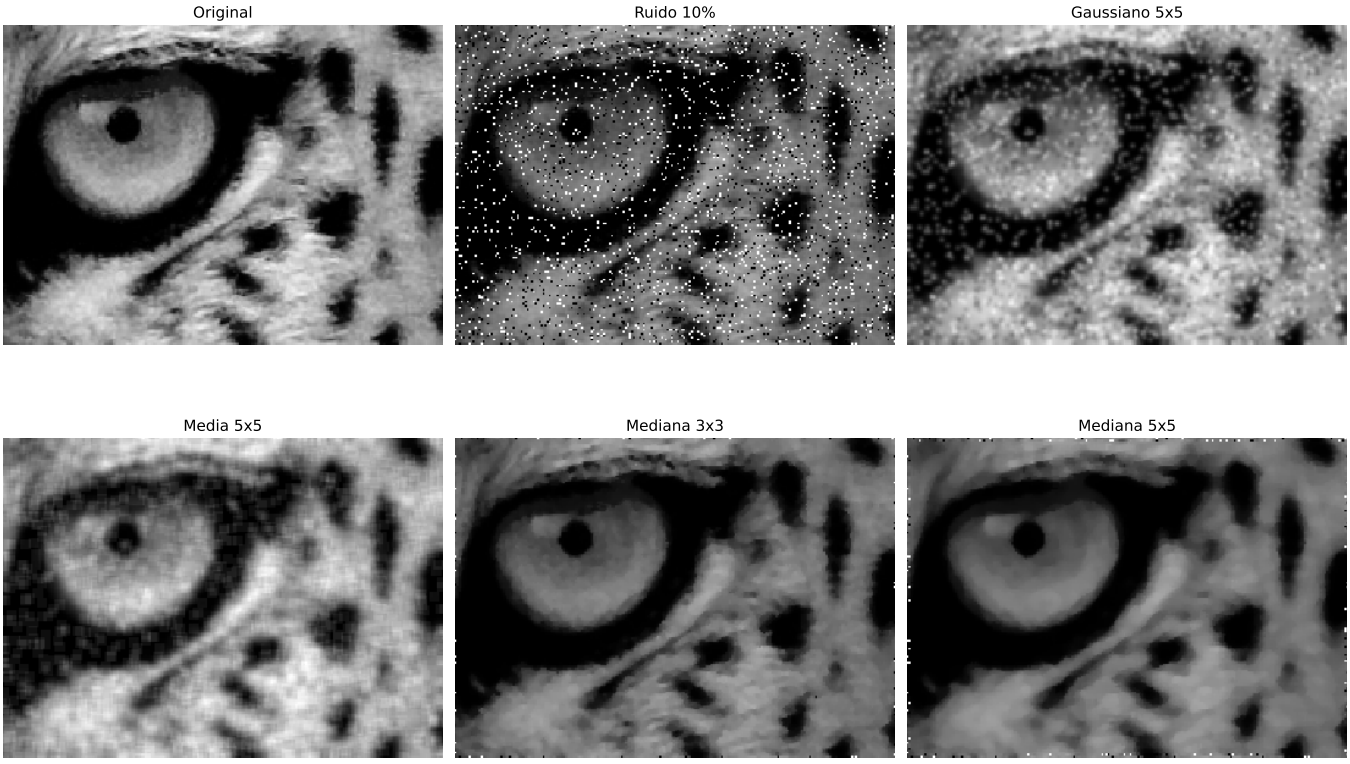


Figure 3.24: Filtros para ruído sal e pimenta (10%): Gaussiano, Média e Mediana. Bilateral e morfológico (open+close) ficam só na trilha Python — bilateral não tem equivalente em morph.hpp e morfologia é do próximo capítulo.

3.8 Application pratique : Prétraitement pour la segmentation

En pratique, les techniques de ce chapitre sont rarement utilisées de manière isolée. Un ***pipeline* de prétraitement** typique combine plusieurs étapes en séquence, en s'adaptant au type d'image et à l'application. La Figure 3.25 illustre un *pipeline* complet :

1. **Égalisation d'histogramme (CLAHE) :** normalise le contraste indépendamment des conditions d'éclairage ;
2. **Filtre gaussien :** lisse le bruit d'acquisition sans détruire les contours ;
3. **Détection de contours (Sobel/Canny) :** extrait les structures pertinentes pour la segmentation.

i L'ordre compte

L'ordre des opérations affecte le résultat final. En général : **(1) normalisation de l'intensité → (2) réduction du bruit → (3) rehaussement/segmentation**. Inverser l'ordre peut amplifier le bruit ou perdre les contours avant de les détecter.

```
%%writefile tmp/fig_03_pipeline.cpp
#define MM_OUT "tmp/fig_03_pipeline.png"
#include "morph.hpp"
#include <iostream>
#include <vector>
#include <filesystem>

int main() {
// [pdi:state-io] auto-generated - do not edit by hand
mm::Image img_gray_crop = mm::_read_state("tmp/state/img_gray_crop_58.png");
// [pdi:state-io:end]
```

```
    //| label: fig-03-pipeline
    //| fig-cap: "*Pipeline* de pré-processamento: equalização → Gaussiano → Canny. A trilha
    Python usa CLAHE no lugar da equalização global; CLAHE não tem equivalente em morph.hpp."
    //| echo: true
    //| output: true

    mm::Image img_eq = mm::equalize(img_gray_crop);     // Etapa 1: equalização global
    mm::Image img_gauss = mm::gaussian(img_eq, 5, 0);      // Etapa 2: Gaussiano
    mm::Image edges = mm::canny(img_gauss, 50, 150);   // Etapa 3: Canny

    mm::Image edges_direct = mm::canny(img_gray_crop, 50, 150);   // Canny direto, sem
pré-processo

    mm::show(
        std::vector<mm::Image>{img_gray_crop, img_eq, img_gauss, edges, edges_direct},
        MM_OUT,
        std::vector<std::string>{"Original", "1. Equalizado", "2. Gaussiano", "3. Canny
(pipeline)", "Canny (direto)"},
        5
    );


// [pdi:panel-io] auto-generated - do not edit by hand
std::filesystem::create_directories("tmp");
mm::write(img_gray_crop, "tmp/fig_03_pipeline_0.png");
mm::write(img_eq, "tmp/fig_03_pipeline_1.png");
mm::write(img_gauss, "tmp/fig_03_pipeline_2.png");
mm::write(edges, "tmp/fig_03_pipeline_3.png");
mm::write(edges_direct, "tmp/fig_03_pipeline_4.png");
// [pdi:panel-io:end]
return 0;
}
```

```
Overwriting tmp/fig_03_pipeline.cpp
```

```
!g++ -I. -std=c++17 tmp/fig_03_pipeline.cpp -o tmp/fig_03_pipeline \
  && ./tmp/fig_03_pipeline \
  && test -f "tmp/fig_03_pipeline.png" \
  || echo "  mm::show não gravou tmp/fig_03_pipeline.png"
```

```
[1] Original
[2] 1. Equalizado
[3] 2. Gaussiano
[4] 3. Canny (pipeline)
[5] Canny (direto)
```

```
try:
    mm.show(
        [
            mm.read("tmp/fig_03_pipeline_0.png"),
            mm.read("tmp/fig_03_pipeline_1.png"),
            mm.read("tmp/fig_03_pipeline_2.png"),
            mm.read("tmp/fig_03_pipeline_3.png"),
            mm.read("tmp/fig_03_pipeline_4.png"),
        ],
        titles=[
            'Original',
            '1. Equalizado',
            '2. Gaussiano',
```

```
            '3. Canny (pipeline)',
            'Canny (direto)',
        ],
        cols=5,
    )
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + " tmp/fig_03_pipeline_0.png
    (ver a versao Python)")
```

Figure 3.25: *Pipeline* de pré-processamento: equalização → Gaussiano → Canny. A trilha Python usa CLAHE no lugar da equalização global; CLAHE não tem equivalente em morph.hpp.

3.9 Résumé

Dans ce chapitre, les principales techniques de traitement dans le domaine spatial ont été présentées, de la manipulation directe des pixels au filtrage par voisinage :

- **Opérations ponctuelles :** arithmétiques saturées (`mm::addm`, `mm::subm`) et logiques bit à bit (`mm::band`, `mm::bor`, `mm::bnot`) pour le recadrage de ROI et la combinaison d'images ; *alpha blending* (`mm::blend`) pour la fusion pondérée avec poids $\alpha \in [0, 1]$.
- **Histogramme :** fonction discrète de distribution des intensités ; visualisé avec `mm::histImg` et calculé avec `mm::hist` ; base pour le diagnostic tonal et pour les techniques d'égalisation et de spécification.
- **Égalisation :** redistribution automatique des intensités par la CDF (`mm::equalize`), avec variante adaptative CLAHE pour le contrôle local du contraste.
- **Spécification d'histogramme :** transfert du profil tonal d'une image de référence via le mappage inverse de la CDF — généralisation de l'égalisation pour des distributions arbitraires.
- **Corrélation et convolution :** mécanisme de fenêtre glissante implémenté dans `mm::conv` (`cv2.filter2D`) ; différenciés par la rotation de 180° du *kernel* — pertinent uniquement pour les kernels asymétriques.
- **Filtres de lissage :** moyenne (*kernel* uniforme, estompe les bords proportionnellement à la taille) et gaussien (pondération radiale, séparable, sans *ringing*, préserve mieux les bords).
- **Filtres de rehaussement :** laplacien (w_4/w_8, seconde dérivée isotrope), Sobel (gradient directionnel du premier ordre, avec magnitude $|\nabla f|$ et direction θ) et *Unsharp Masking* (amplification des hautes fréquences avec paramètre k).
- **Filtre médian :** non linéaire, robuste aux outliers, préserve les bords — supérieur aux filtres linéaires pour le bruit sel et poivre.
- **Pipeline pratique :** enchaînement CLAHE → Gaussien → Canny comme stratégie de prétraitement ; `mm::drawImgKernel` pour la visualisation didactique de la fenêtre glissante.

Le Chapitre 4 abordera la **morphologie mathématique** (érosion, dilatation, ouverture et fermeture), en explorant en profondeur les fonctions `mm::ero` et `mm::dil` de la bibliothèque `morph.py`. Ensuite, le Chapitre 5 présentera le **traitement dans le domaine fréquentiel**, avec un accent sur la Transformée de Fourier et les techniques de filtrage spectral.

3.10 🤖 Utilisation de Gemini Notebook comme tuteur complémentaire

Dans cette édition, nous encourageons l'utilisation de **Gemini Notebook** comme outil d'apprentissage complémentaire. Cet outil d'IA utilise exclusivement les documents fournis par l'auteur comme base de

connaissances, garantissant des réponses cohérentes avec le contenu du livre — y compris les fonctions de la bibliothèque `morph.py` et les expériences réalisées dans ce chapitre.

Pour chaque chapitre, nous avons préparé un projet spécifique sur la plateforme avec le PDF du chapitre, les *notebooks* et les supports auxiliaires. Nous suggérons d'explorer en particulier :

- **Guide d'étude :** résumé structuré des concepts, idéal pour la révision avant les examens ;
- **Conversation :** posez vos questions sur l'égalisation, la convolution, les filtres et les pipelines directement au tuteur ;
- **Questions fréquentes :** questions typiques sur la différence entre moyenne et médiane, USM, Laplacien vs. Sobel.

🎓 Étudiez avec le tuteur intelligent

Pour interagir avec le contenu de ce chapitre, accédez au *lien* suivant. L'environnement contient des supports pédagogiques sous différents formats, générés à partir du **PDF** du chapitre. Sur la plateforme, explorez particulièrement les options **Guide d'étude** et **Conversation** pour approfondir votre compréhension.
🚀 ACCÉDER À GEMINI NOTEBOOK : CHAPITRE 03

🌐 **Langue et langage de programmation**

Le projet de ce chapitre dans Gemini Notebook a été construit uniquement avec le texte en **portugais** et les exemples de code en **Python**. Si vous étudiez à partir de l'édition en anglais ou en français, ou que vous suivez le parcours en C++, les réponses du tuteur peuvent ne pas correspondre exactement à la version que vous lisez.

⚠️ **Avertissement concernant le contenu généré par l'IA**

L'IA est une alliée puissante dans les études, mais le contenu généré peut contenir des **erreurs ou des imprécisions**. Consultez toujours des **livres, articles scientifiques et autres sources académiques fiables** pour valider les informations. Dans la mesure du possible, exécutez les exemples pratiques fournis dans ce chapitre pour vérifier les résultats.

3.11 Liste d'exercices

1. **(10%)** Expliquez la différence entre **convolution** et **corrélation croisée**. Pour quels types de *noyaux* les résultats sont-ils identiques ? Donnez un exemple de *noyau* asymétrique (comme Sobel G_x) et montrez numériquement que les résultats diffèrent en l'appliquant au patch 5×5 du chapitre de deux manières.

2. **(15%)** Considérez une image 5×5 avec des intensités concentrées entre les niveaux 3 et 5 (faible contraste, 3 bits). Appliquez manuellement l'algorithme d'égalisation de la Table 3.1, en remplissant toutes les colonnes du tableau (k, $h[k]$, $p[k]$, cdf$[k]$, lut$[k]$). Vérifiez le résultat avec `mm::equalize`.

3. **(15%)** En utilisant `mm::conv`, appliquez le filtre moyenneur avec des noyaux de tailles 3×3, 9×9 et 21×21 à l'image du mandrill. Pour chaque version, calculez le **PSNR** (*Peak Signal-to-Noise Ratio*) par rapport à l'originale :

$$\text{PSNR} = 10\log_{10}\left(\frac{255^2}{\text{MSE}}\right), \quad \text{MSE} = \frac{1}{MN}\sum_{i,j}(f-g)^2$$

   Tracez le PSNR en fonction de la taille du noyau et expliquez ce que la chute progressive indique sur la relation entre le lissage et la perte d'information.

4. **(15%)** En utilisant `add_salt_pepper` avec une densité de 5%, appliquez et comparez : (a) `mm::conv` avec moyenne 3×3, (b) `cv2.GaussianBlur` avec $\sigma = 1$, (c) `cv2.medianBlur` avec fenêtre 3×3 et (d)

`cv2.medianBlur` avec fenêtre 5×5. Affichez les images avec `mm::show` dans une grille 2×4 (ligne 1 : images, ligne 2 : histogrammes via `mm::histImg`). Expliquez pourquoi la médiane surpasse les filtres linéaires en utilisant l'argument de la Table 3.4..

5. **(15%)** Implémentez `mm::conv0` en utilisant uniquement des opérations NumPy vectorisées — sans boucles Python et sans `cv2.filter2D` — avec l'opérateur de *stride tricks* (`np.lib.stride_tricks.sliding_window_view`). Comparez le résultat et le temps d'exécution avec `mm::conv0` (boucles) et `mm::conv` (cv2) pour des noyaux 3×3 et 15×15 sur l'image du mandrill.
6. **(15%)** Appliquez le *Unsharp Masking* avec $\sigma = 1$ et $k \in \{0.5, 1.0, 2.0, 4.0\}$ en utilisant la fonction `usm` du chapitre. Pour chaque valeur de k : (a) calculez la différence absolue $|g - f|$, (b) affichez les images et les différences avec `mm::show`, et (c) tracez l'histogramme des différences avec `mm::histImg`. Identifiez à partir de quel k les artefacts (halos et amplification du bruit) deviennent visuellement inacceptables.
7. **(15%)** Choisissez une image de radiographie ou de tomographie disponible publiquement (ex. : via `mm::read` à partir d'une URL) et concevez un pipeline de prétraitement avec au moins 4 étapes séquentielles, en justifiant chaque choix à partir des concepts du chapitre. Affichez avec `mm::show` dans une grille : l'image originale, chaque étape intermédiaire et le résultat final avec leurs histogrammes (`mm::histImg`).

Références du chapitre

La base théorique de ce chapitre s'appuie sur les ouvrages suivants :

- Gonzalez (2018) pour les concepts d'opérations d'intensité, d'histogramme, de convolution et de filtrage spatial.
- Szeliski (2022) pour la vision par ordinateur et les applications pratiques du filtrage.
- Bradski (2008) pour l'implémentation pratique avec OpenCV et `morph.py`.

3.12 💻 Partie Pratique avec Exercices de Programmation

🎯 Objectif de ce cahier

Ce cahier permet de développer, valider, organiser et tester des solutions d'**Exercices de Programmation (EPs)** dans des environnements interactifs, tels que Colab, avec les mêmes cas de test que Moodle, en y copiant uniquement au moment d'enregistrer la note officielle.

Téléchargement

Téléchargez `morph.py` et `testsuite.py` en exécutant la cellule ci-dessous :

```
import os, urllib.request

os.makedirs("tmp/state", exist_ok=True)  # artefacts de build du parcours C++ (.cpp, binaire,
PNGs)

url = "https://raw.githubusercontent.com/fzampirolli/pdi-vc/master/morph/config.py"
if not os.path.exists("config.py"):
    urllib.request.urlretrieve(url, "config.py")

# Le noyau est Python même dans le parcours C++ : `mm` (morph.py) est utilisé par les
# simulateurs, par l'affichage des figures que le binaire C++ génère et par
# l'état mm::Image entre cellules. cpp=True télécharge aussi le parcours compilé
```

```
# (morph.hpp + stb_image*.h), utilisé dans le #include des cellules %%writefile *.cpp.
import config
config.setup(testsuite=True, cpp=True)
from morph import mm
from testsuite import TestSuite
```

```
Environnement prêt. Morph : 1.1.9 | OpenCV : 5.0.0 | TestSuite : 1.1.2
```

Exécution des tests

Pour évaluer les tests, exécutez `TestSuite("EP03_01.extensão").run()` dans une nouvelle cellule, en remplaçant l'extension par celle du langage utilisé (`.py`, `.java`, `.c`, `.cpp`, `.js` ou `.r`). Le système télécharge les cas de test depuis GitHub, exécute le programme et calcule la note automatiquement.

Pour tester du code Python directement, sans enregistrer de fichier, utilisez `run_code(codigo)` en passant le code sous forme de *chaîne de caractères* dans une variable `codigo` :

```
codigo = """
from morph import mm
# ... votre code ici ...
"""
TestSuite("EP03_01").run_code(codigo)
```

3.12.1 EP03_01 ➕ Addition Saturée de Constante

Dans les systèmes de surveillance vidéo, les caméras situées dans des environnements à éclairage variable produisent des images sous-exposées. Le réglage de la luminosité par **addition saturée d'une constante** est l'opération la plus simple pour une correction immédiate, étant appliquée en temps réel sur les *puces* des caméras embarquées et dans les *pipelines* de pré-traitement des robots mobiles.

Voir dans Figure 3.26 une simulation de ce EP.

3.12.1.1 📋 Directives d'Implémentation

1. **Dimensions :** Lire les entiers L (lignes) et C (colonnes).
2. **Constante :** Lire l'entier k (valeur à ajouter).
3. **Données :** Lire les valeurs entières de la matrice originale ligne par ligne.
4. **Mappage :** Pour chaque pixel p, calculer la nouvelle valeur selon l'équation :

$$p' = \text{clip}(p + k)$$

5. **Sortie :** Afficher la matrice résultante avec les dimensions $L \times C$.

3.12.1.2 📌 Contraintes Computationnelles

- **Saturation (*Clipping*) :** Les valeurs doivent être confinées à l'intervalle $[0, 255]$:

$$\text{clip}(x) = \max(0, \min(255, x))$$

- **Type :** Le résultat final doit être entier (sans décimales).
- k **peut être négatif :** les valeurs négatives assombrissent l'image ; les valeurs positives l'éclaircissent.

3.12.1.3 🧠 Fondement Théorique

Paramètre	Type	Impact Visuel
$k > 0$	Entier	Éclaircit l'image ; les pixels proches de 255 saturent en blanc
$k < 0$	Entier	Assombrit l'image ; les pixels proches de 0 saturent en noir
$k = 0$	Entier	Image inchangée

3.12.1.4 📦 Spécification d'Entrée et de Sortie (VPL)

Entrée :

- Ligne 1 : Entier L.
- Ligne 2 : Entier C.
- Ligne 3 : Entier k.
- Lignes suivantes : Éléments entiers de la matrice originale.

Sortie :

- Matrice transformée en L lignes et C colonnes, valeurs entières séparées par des espaces.

3.12.1.5 📌 Exemples

Entrée	Sortie	Observation
2 3 50 0 100 200 210 240 255	50 150 250 255 255 255	Saturation à 255 sur les pixels élevés
1 4 -30 0 20 200 255	0 0 170 225	Saturation à 0 sur les pixels bas

```
%%writefile EP03_01.cpp
// your solution
```

```
Overwriting EP03_01.cpp
```

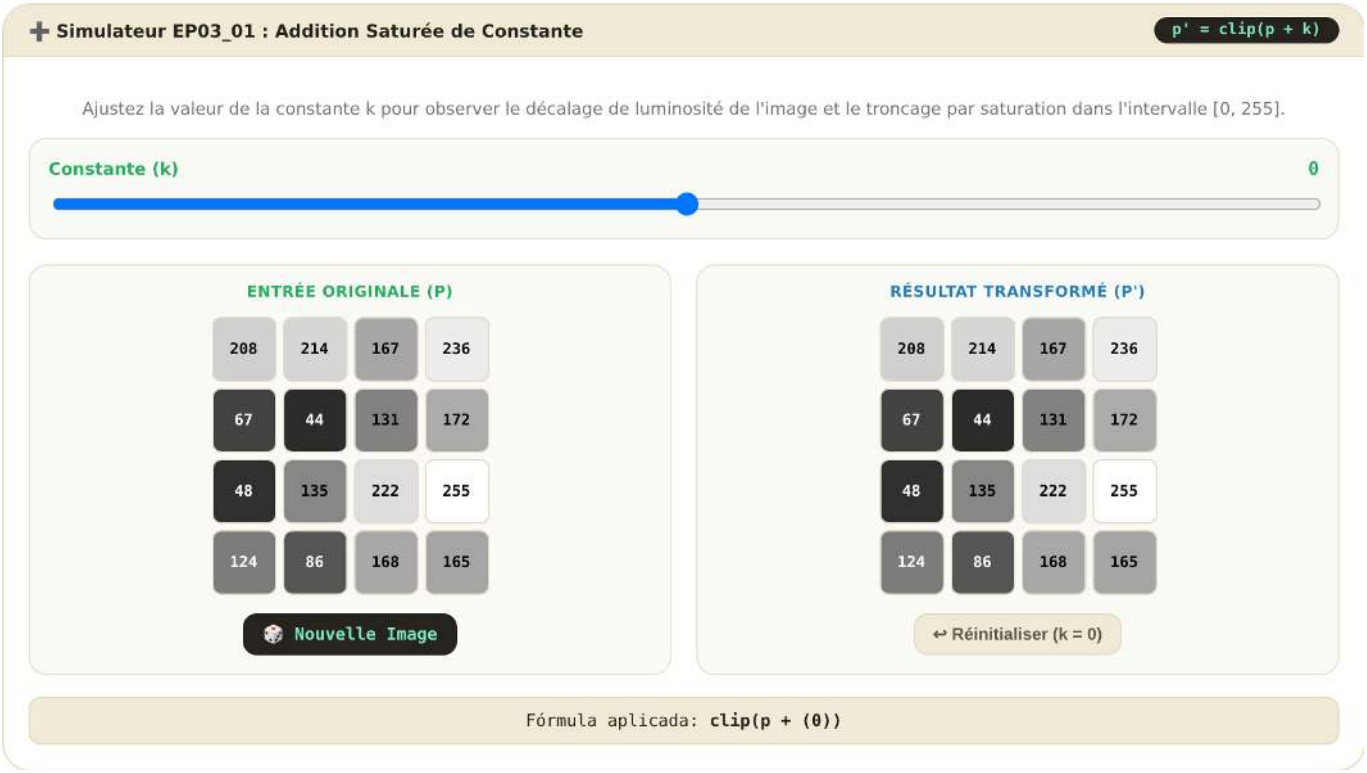
```
TestSuite("EP03_01.cpp").run()
```

```
<IPython.core.display.HTML object>
```

3.12.2 EP03_02 🔀 Fusion *Alpha* de Deux Images

En médecine nucléaire, des images de différentes modalités (tomodensitométrie et imagerie par résonance magnétique) sont fusionnées pour faciliter le diagnostic. Le **mélange pondéré** (*alpha blending*) est l'opération fondamentale de ce processus, permettant au radiologue de contrôler interactivement le poids de chaque modalité dans l'image affichée.

Voir dans Figure 3.27 une simulation de ce EP.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.fr/cap03/sim-ep0301-adicao.html

Figure 3.26: Simulateur EP03_01 : Addition saturée de constante (p' = clip(p + k))

3.12.2.1 📋 Directives d'Implémentation

1. **Dimensions :** Lire les entiers L (lignes) et C (colonnes).
2. **Paramètre :** Lire la valeur réelle $\alpha \in [0, 1]$.
3. **Données :** Lire les valeurs entières de la matrice f_1 (image 1), puis celles de la matrice f_2 (image 2).
4. **Mappage :** Pour chaque position (i, j), calculer :

$$g(i,j) = \text{clip}\left(\text{round}\left(\alpha \cdot f_1(i,j) + (1-\alpha) \cdot f_2(i,j)\right)\right)$$

5. **Sortie :** Afficher la matrice résultante $L \times C$.

3.12.2.2 📌 Contraintes de Calcul

- **Arrondi :** Appliquer `round` avant la conversion en entier.
- **Saturation :** Confiner à l'intervalle $[0, 255]$ avec $\text{clip}(x) = \max(0, \min(255, x))$.
- **Opération en virgule flottante :** Effectuer l'opération en virgule flottante avant d'arrondir.

3.12.2.3 🧠 Fondement Théorique

Valeur de α	Résultat
$\alpha = 1.0$	Uniquement f_1
$\alpha = 0.5$	Moyenne arithmétique de f_1 et f_2
$\alpha = 0.0$	Uniquement f_2

3.12.2.4 📦 Spécification des Entrées et Sorties (VPL)

Entrée :

- Ligne 1 : Entier L.

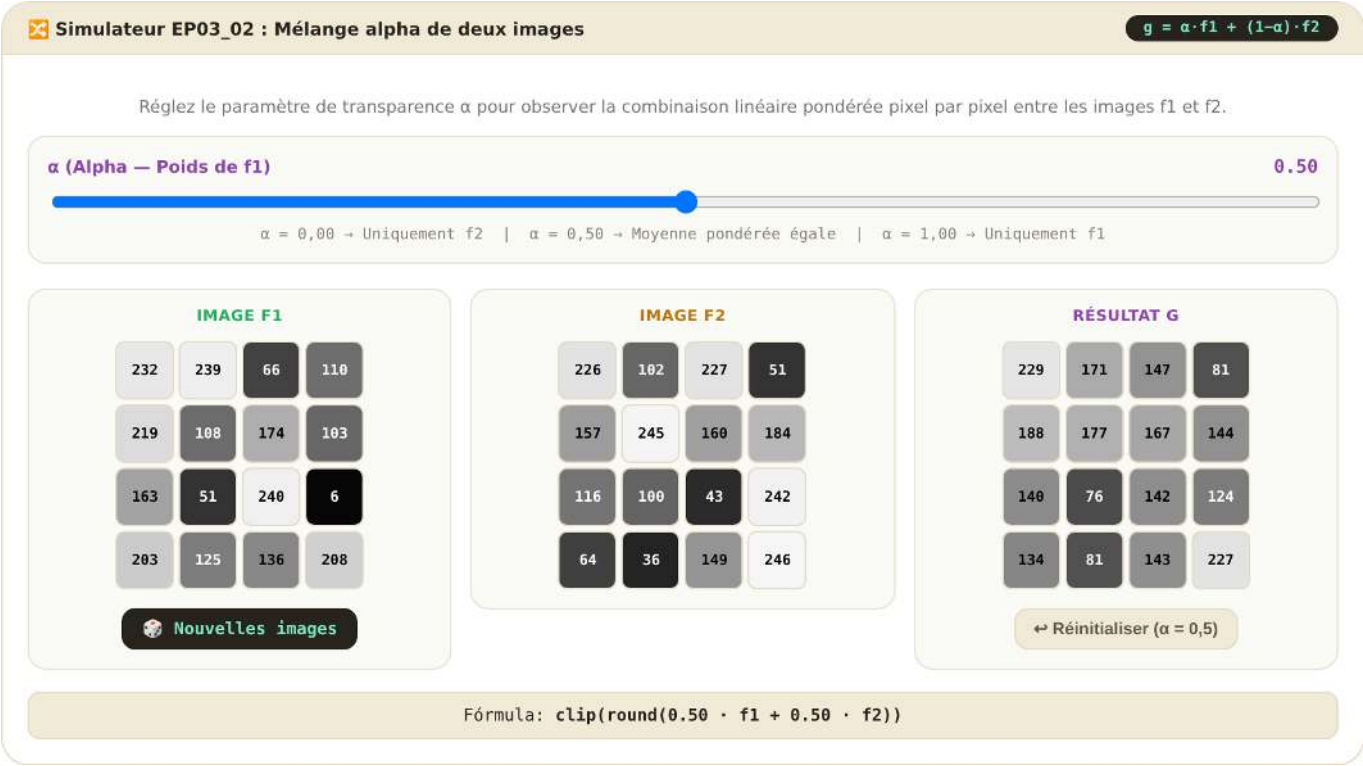
- Ligne 2 : Entier C.
- Ligne 3 : Réel α.
- Lignes suivantes : Éléments de f_1 (L lignes avec C valeurs chacune).
- Lignes suivantes : Éléments de f_2 (L lignes avec C valeurs chacune).

Sortie :

- Matrice résultante $L \times C$.

3.12.2.5 📌 Exemples

Entrée	Sortie	Observation
1 3 0.5 0 100 200 100 200 50	50 150 125	Moyenne entre les deux images
1 3 1.0 10 20 30 90 80 70	10 20 30	Uniquement f_1 (alpha=1)



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.fr/cap03/sim-ep0302-blending.html

Figure 3.27: Simulateur EP03_02 : Mélange Alpha de Deux Images (g = · f1 + (1−) · f2)

```
%%writefile EP03_02.cpp
// your solution
```

```
Overwriting EP03_02.cpp
```

```
TestSuite("EP03_02.cpp").run()
```

```
<IPython.core.display.HTML object>
```

3.12.3 EP03_03 🎭 Inversion d'image (négatif photographique)

En radiologie, les images de rayons X sont traditionnellement visualisées en négatif : les os apparaissent en noir sur fond blanc. L'opération de **négatif photographique** est appliquée de manière courante dans les PACS (*Picture Archiving and Communication Systems*) pour faciliter la détection des fractures et des densités osseuses.

Voir dans Figure 3.28 une simulation de cet EP.

3.12.3.1 📋 Directives d'implémentation

1. **Dimensions :** Lire les entiers L (lignes) et C (colonnes).
2. **Données :** Lire les valeurs entières de la matrice originale.
3. **Mappage :** Pour chaque pixel p, calculer le négatif :

$$p' = 255 - p$$

4. **Sortie :** Afficher la matrice résultante $L \times C$.

3.12.3.2 📌 Contraintes informatiques

- **Aucun *clipping* nécessaire :** Le résultat de $255 - p$ avec $p \in [0, 255]$ est toujours $\in [0, 255]$.
- **Type entier :** La sortie doit être des valeurs entières.
- **Équivalence logique :** L'opération est identique au `NOT` bit à bit (`mm::bnot`) sur les images en 8 bits.

3.12.3.3 🧠 Fondements théoriques

Pixel original p	Pixel négatif p'	Observation
0 (noir)	255 (blanc)	Inversion totale
128 (gris moyen)	127 (gris moyen)	Valeur centrale
255 (blanc)	0 (noir)	Inversion totale

3.12.3.4 📦 Spécification d'entrée et de sortie (VPL)

Entrée :

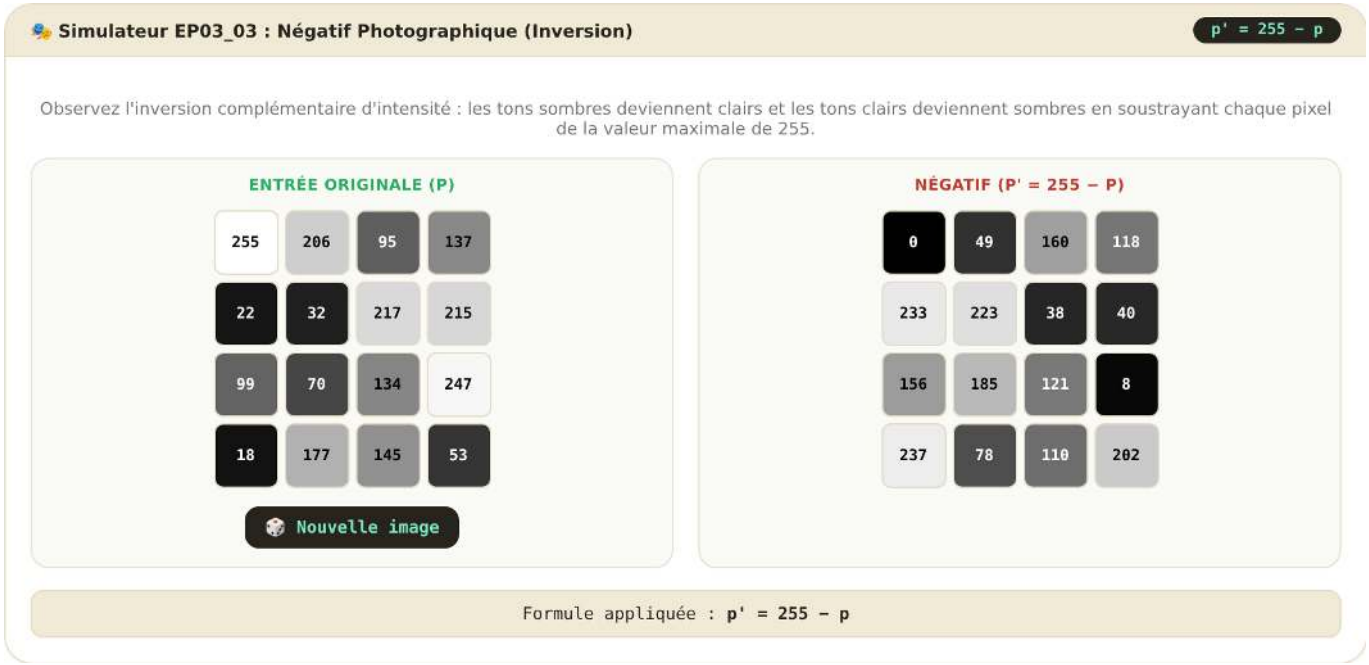
- Ligne 1 : Entier L.
- Ligne 2 : Entier C.
- Lignes suivantes : Éléments entiers de la matrice originale.

Sortie :

- Matrice négative en L lignes et C colonnes.

3.12.3.5 📌 Exemples

Entrée	Sortie	Observation
140 128 200 255	255 127 55 0	Inversion de chaque pixel
2 2 10 20 30 40	245 235225 215	Matrice 2x2 inversée



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.fr/cap03/sim-ep0303-inversao.html

Figure 3.28: Simulador EP03_03 : Inversion d'image — Négatif photographique (p' = 255 − p)

```
%%writefile EP03_03.cpp
// your solution
```

```
Overwriting EP03_03.cpp
```

```
TestSuite("EP03_03.cpp").run()
```

```
<IPython.core.display.HTML object>
```

3.12.4 EP03_04 📊 Égalisation d'histogramme (L bits)

Dans les images satellites de télédétection, la variation de l'éclairage au cours de la journée produit des images à faible contraste. L'**égalisation d'histogramme** est appliquée automatiquement sur des satellites comme Landsat pour redistribuer les tons, révélant des détails de végétation, de relief et de zones urbaines invisibles dans l'image originale.

Voir dans Figure 3.29 une simulation de cet EP.

3.12.4.1 📋 Directives d'implémentation

1. **Dimensions :** Lire les entiers L (lignes), C (colonnes) et B (nombre de bits, avec $L_{\max} = 2^B$).
2. **Données :** Lire la matrice de pixels f avec des valeurs dans $[0, 2^B - 1]$.
3. **Histogramme :** Calculer $h[k]$ = nombre de pixels avec l'intensité k, pour $k = 0 \ldots 2^B - 1$.
4. **Probabilité :** $p[k] = h[k]/(L \cdot C)$.
5. **CDF :** $\text{cdf}[k] = \sum_{j=0}^{k} p[j]$; fonction de distribution cumulée.
6. **LUT :** $\text{lut}[k] = \text{round}\,(\text{cdf}[k] \cdot (2^B - 1))$; *Look-Up Table* (table de consultation).
7. **Application :** $g[i, j] = \text{lut}[f[i, j]]$.
8. **Sortie :** Afficher la matrice égalisée $L \times C$.

3.12.4.2 📌 Contraintes computationnelles

- **Arrondi :** Utiliser l'arrondi mathématique (`round`) dans la LUT.

- **Bits :** Le nombre de niveaux est 2^B (ex. : $B = 3 \Rightarrow 8$ niveaux, $B = 8 \Rightarrow 256$ niveaux).
- **CDF cumulée :** $\text{cdf}[k] = \sum_{j=0}^{k} p[j]$, avec $\text{cdf}[2^B - 1] = 1.0$.

3.12.4.3 🧠 Fondement théorique

Étape	Opération	Formule
1	Histogramme	$h[k] \leftarrow$ nombre de pixels avec l'intensité k
2	Probabilité	$p[k] = h[k]/(L \cdot C)$
3	CDF	$\text{cdf}[k] = \sum_{j=0}^{k} p[j]$
4	LUT	$\text{lut}[k] = \text{round}(\text{cdf}[k] \cdot (2^B - 1))$
5	Application	$g[i, j] = \text{lut}[f[i, j]]$

3.12.4.4 📦 Spécification d'entrée et de sortie (VPL)

Entrée :

- Ligne 1 : Entier L.
- Ligne 2 : Entier C.
- Ligne 3 : Entier B (nombre de bits).
- Lignes suivantes : Éléments entiers de la matrice.

Sortie :

- Matrice égalisée en L lignes et C colonnes.

3.12.4.5 📌 Exemples

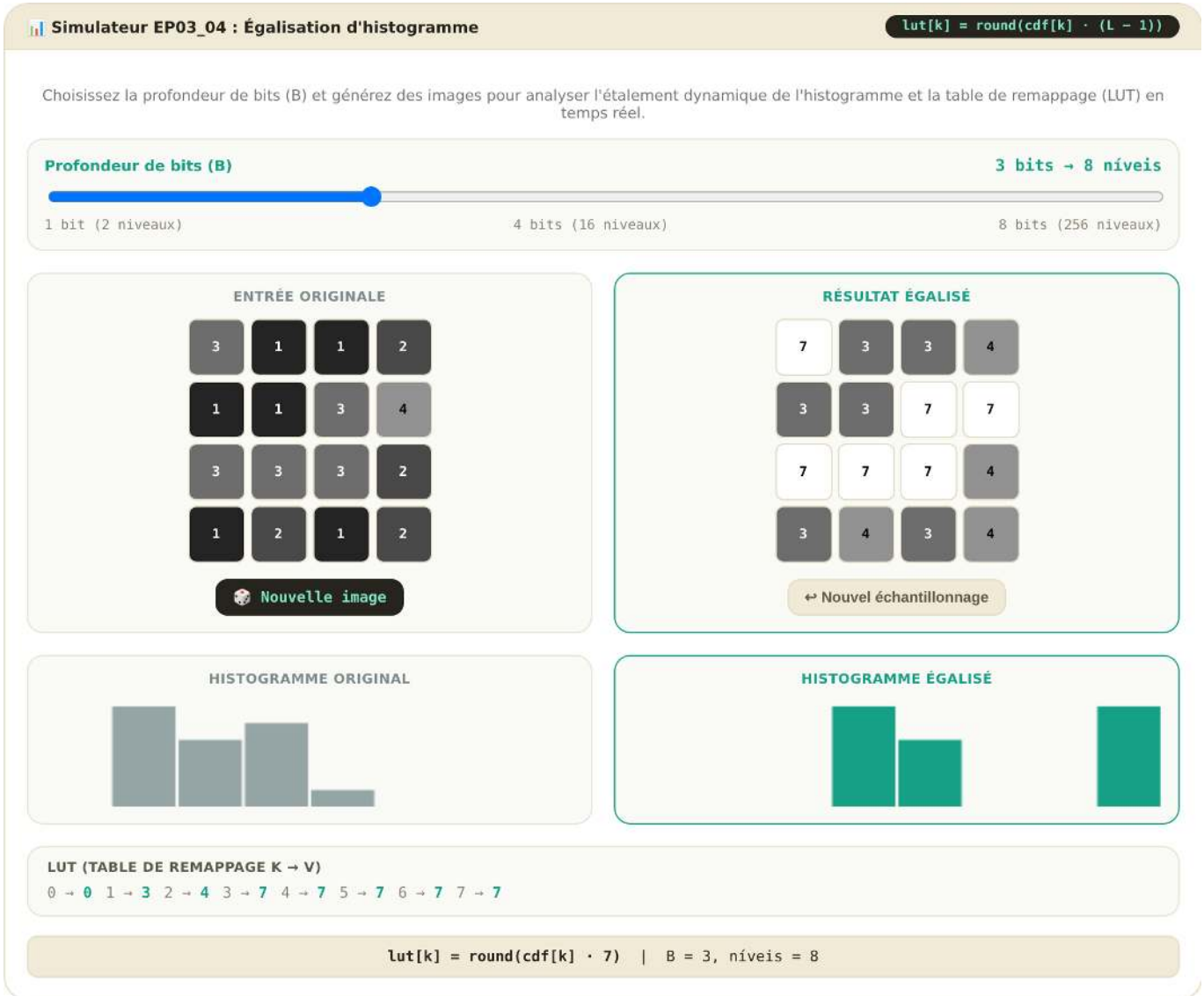
Entrée	Sortie	Observation
5 5 3 3 4 2 3 4 4 3 3 4 3 2 3 4 3 2 3 4 3 2 3 4 3 2 3 4	5 7 1 5 7 7 5 5 7 5 1 5 7 5 1 5 7 5 1 5 7 5 1 5 7	Exemple 3 bits du chapitre
1 4 3 0 0 7 7	0 0 7 7	Histogramme bimodal extrême

```
%%writefile EP03_04.cpp
// your solution
```

```
Overwriting EP03_04.cpp
```

```
TestSuite("EP03_04.cpp").run()
```

```
<IPython.core.display.HTML object>
```



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.fr/cap03/sim-ep0304-equalizacao.html

Figure 3.29: Simulateur EP03_04 : Égalisation d'histogramme (Niveaux L = 2^B)

3.12.5 EP03_05 🔲 Application du masque binaire ET

Dans les systèmes d'inspection industrielle par vision par ordinateur, il est nécessaire d'isoler les régions d'intérêt (ROI) dans les images de pièces afin de vérifier les défauts de fabrication. L'opération **ET bit à bit avec un masque binaire** est le mécanisme fondamental pour découper exactement la zone d'inspection, en mettant à zéro tous les pixels en dehors de celle-ci.

Voir dans Figure 3.30 une simulation de cet EP.

3.12.5.1 📋 Directives d'implémentation

1. **Dimensions :** Lire les entiers L (lignes) et C (colonnes).
2. **Données :** Lire la matrice de pixels f (valeurs $\in [0, 255]$).
3. **Masque :** Lire la matrice binaire m (valeurs : uniquement 0 ou 255).
4. **Mappage :** Pour chaque pixel (i, j), appliquer le ET bit à bit :

$$g(i, j) = f(i, j) \text{ ET } m(i, j)$$

où $255 =$ `11111111` et $0 =$ `00000000` en binaire.

5. **Sortie :** Afficher la matrice résultante $L \times C$.

3.12.5.2 📌 Contraintes de calcul

- **ET avec 255 :** p ET $255 = p$ (tous les bits sont préservés).
- **ET avec 0 :** p ET $0 = 0$ (tous les bits sont mis à zéro).
- **Masque :** Les seules valeurs possibles dans le masque sont 0 et 255.
- **Implémentation :** En Python, le ET bit à bit entre entiers utilise l'opérateur `&`.

3.12.5.3 🧠 Fondement théorique

Pixel f	Masque m	Résultat f ET m
n'importe quel v	255 (`11111111`)	v (préservé)
n'importe quel v	0 (`00000000`)	0 (mis à zéro)

3.12.5.4 📦 Spécification d'entrée et de sortie (VPL)

Entrée :

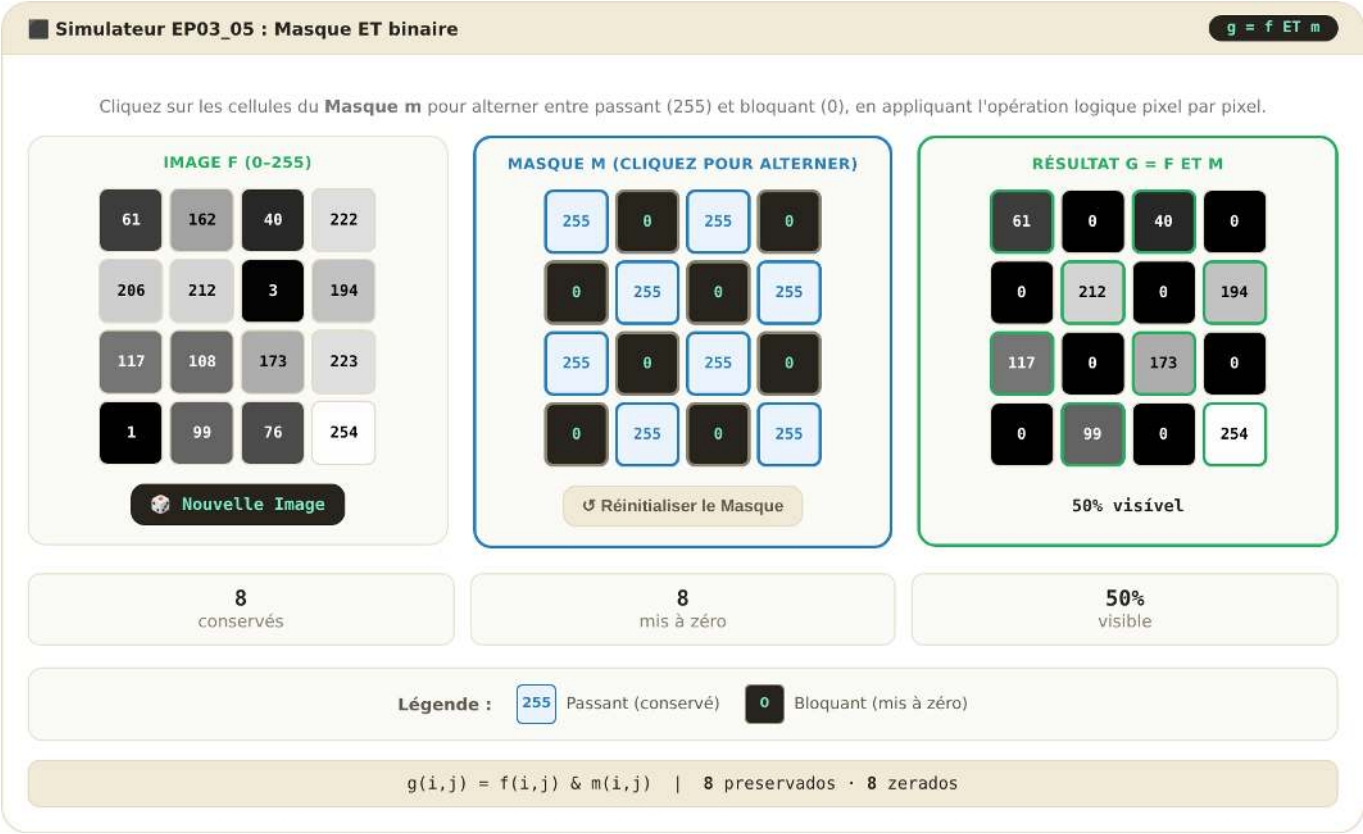
- Ligne 1 : Entier L.
- Ligne 2 : Entier C.
- Lignes suivantes : Éléments de f (L lignes).
- Lignes suivantes : Éléments de m (L lignes avec valeurs 0 ou 255).

Sortie :

- Matrice résultante $L \times C$.

3.12.5.5 📌 Exemples

Entrée	Sortie	Observation
2 3 100 150 200 50 80 120 255 255 0 0 255 255	100 150 0 0 80 120	Le masque sélectionne la région
1 4 10 20 30 40 255 0 255 0	10 0 30 0	Alternance préservé/mis à zéro



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.fr/cap03/sim-ep0305-mascara.html

Figure 3.30: Simulateur EP03_05: Application de masque AND binaire

```
%%writefile EP03_05.cpp
// your solution
```

```
Overwriting EP03_05.cpp
```

```
TestSuite("EP03_05.cpp").run()
```

```
<IPython.core.display.HTML object>
```

3.12.6 EP03_06 Filtre de moyenne avec *kernel* N×N

Dans les caméras de véhicules autonomes, les images capturées sous la pluie ou le brouillard présentent un bruit gaussien. Le **filtre de moyenne** est largement utilisé pour sa réduction en temps réel, étant

implémenté directement dans le **ISP** (*Image Signal Processor*) des capteurs **CMOS** (*Complementary Metal-Oxide-Semiconductor*).

Les capteurs CMOS sont les capteurs d'image utilisés dans la plupart des caméras modernes (*smartphones*, *webcams*, caméras automobiles, etc.). Ils convertissent la lumière en signaux électriques, et l'ISP traite ces signaux en temps réel — en appliquant des opérations telles que la réduction du bruit, la balance des blancs et d'autres ajustements d'image.

Voir dans Figure 3.31 une simulation de cet EP.

3.12.6.1 📋 Directives d'implémentation

1. **Dimensions :** Lire les entiers L (lignes), C (colonnes) et N (taille du *kernel*, toujours impair).
2. **Données :** Lire la matrice de pixels f.
3. **Filtre de moyenne :** Pour chaque pixel (i, j) **interne** (sans bordures), calculer :

$$g(i,j) = \text{round}\left(\frac{1}{N^2}\sum_{s=-(r)}^{r}\sum_{t=-(r)}^{r} f(i+s,\, j+t)\right), \quad r = \lfloor N/2 \rfloor$$

4. **Traitement des bords :** Les pixels sur le bord (où la fenêtre $N \times N$ dépasse les limites) doivent être **copiés directement** depuis l'original sans modification.
5. **Sortie :** Afficher la matrice résultante $L \times C$.

3.12.6.2 📌 Contraintes de calcul

- **Rayon :** $r = \lfloor N/2 \rfloor$ (moitié du *kernel*, entier).
- **Pixels internes :** (i, j) avec $r \leq i < L - r$ et $r \leq j < C - r$.
- **Arrondi :** Utiliser l'arrondi mathématique avant de convertir en entier.
- **Sans *clipping* :** La moyenne des valeurs $\in [0, 255]$ reste dans $[0, 255]$.

3.12.6.3 🧠 Fondements théoriques

Taille N	Coefficient	Pixels dans la fenêtre	Effet
3	$1/9 \approx 0.111$	9	Lisse
5	$1/25 = 0.04$	25	Moyen
7	$1/49 \approx 0.020$	49	Fort

3.12.6.4 📦 Spécification d'entrée et de sortie (VPL)

Entrée :

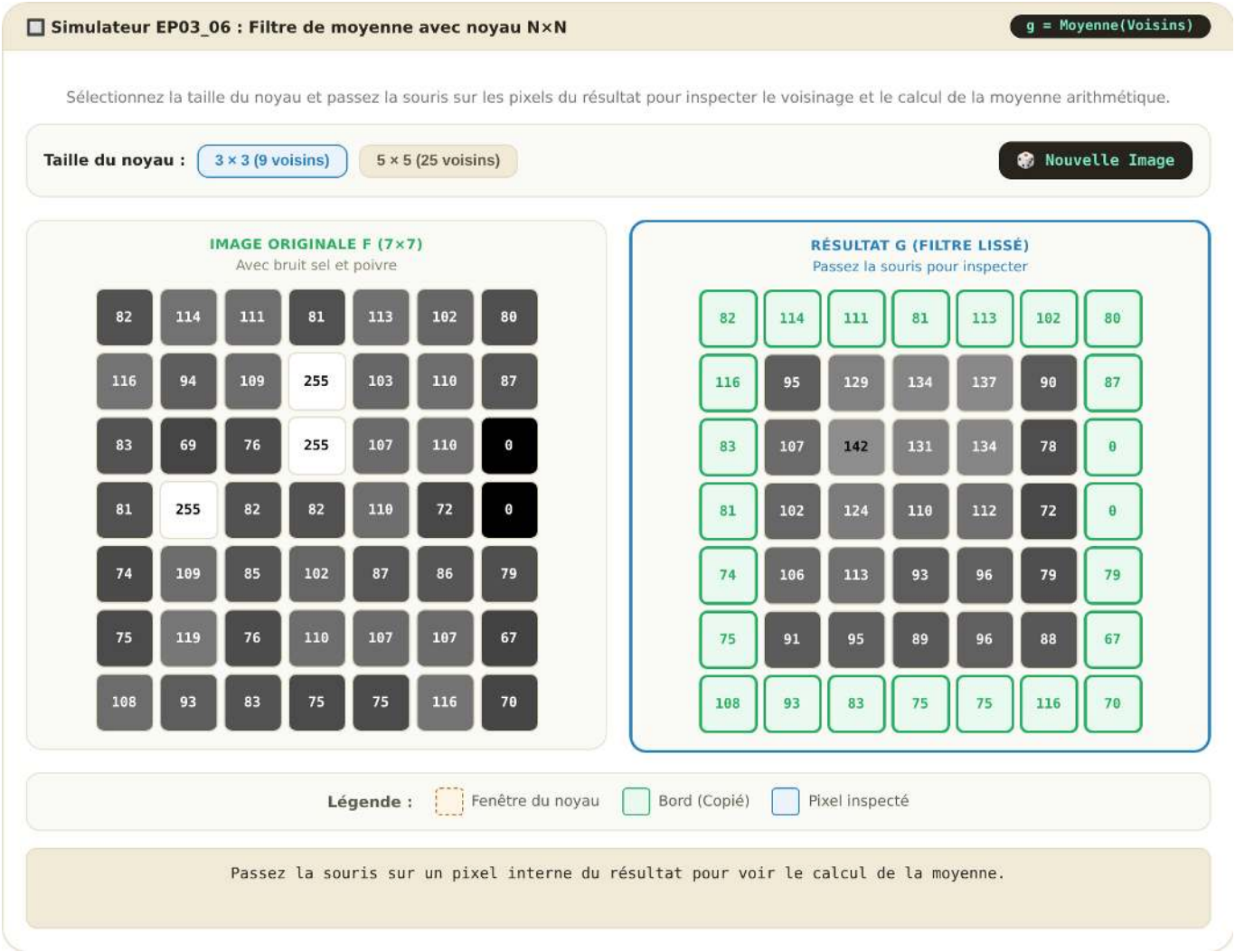
- Ligne 1 : Entier L.
- Ligne 2 : Entier C.
- Ligne 3 : Entier N (impair, $N \geq 3$).
- Lignes suivantes : Éléments de la matrice originale.

Sortie :

- Matrice filtrée $L \times C$.

3.12.6.5 📌 Exemples

Entrée	Sortie	Observation
3 3 3 10 20 30 40 50 60 70 80 90	10 20 30 40 50 60 70 80 90	Bordure uniquement (3×3 = bordure totale)
5 5 3 0 0 0 0 0 0 0 0 0 0 0 0 100 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 11 11 11 0 0 11 11 11 0 0 11 11 11 0 0 0 0 0 0	Pixel isolé : tous les 9 pixels internes dont la fenêtre 3×3 inclut la valeur 100 reçoivent round(100/9)=11



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.fr/cap03/sim-ep0306-media.html

Figure 3.31: Simulateur EP03_06: Filtre de Moyenne avec Noyau N×N

```
%%writefile EP03_06.cpp
// your solution
```

```
Overwriting EP03_06.cpp
```

```
TestSuite("EP03_06.cpp").run()
```

```
<IPython.core.display.HTML object>
```

3.12.7 EP03_07 🔍 Opérateur Laplacien (w4) pour le Rehaussement des Contours

Dans les tomographies à haute résolution, la netteté des contours entre les tissus est critique pour le diagnostic. L'**opérateur Laplacien** est largement utilisé dans les *pipelines* de prétraitement d'images médicales pour rehausser automatiquement les contours anatomiques avant la segmentation, évitant ainsi une intervention manuelle du radiologiste.

Voir dans Figure 3.32 une simulation de cet EP.

3.12.7.1 📋 Directives d'Implémentation

1. **Dimensions :** Lire les entiers L (lignes) et C (colonnes).
2. **Données :** Lire la matrice de pixels f.
3. **Laplacien (w4) :** Pour chaque pixel **interne** (i, j) avec $1 \leq i < L-1$, $1 \leq j < C-1$, calculer :

$$\nabla^2 f(i,j) = f(i-1,j) + f(i+1,j) + f(i,j-1) + f(i,j+1) - 4 \cdot f(i,j)$$

4. **Rehaussement :** Calculer l'image rehaussée :

$$g(i,j) = \text{clip}(f(i,j) - \nabla^2 f(i,j))$$

5. **Bordure :** Les pixels en bordure sont copiés directement : $g(i,j) = f(i,j)$.
6. **Sortie :** Afficher la matrice rehaussée $L \times C$.

3.12.7.2 📌 Contraintes Computationnelles

- ***Noyau* w4 :** $\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$ — uniquement 4-voisins.
- **Saturation :** $\text{clip}(x) = \max(0, \min(255, x))$ appliqué au résultat du rehaussement.
- **Sans arrondi :** Le Laplacien n'utilise que des additions/soustractions d'entiers.

3.12.7.3 🧠 Fondement Théorique

Région	$\nabla^2 f$	Effet du Rehaussement
Uniforme	≈ 0	Aucune modification
Contour croissant	< 0	Pixel éclairci
Contour décroissant	> 0	Pixel assombri

3.12.7.4 📦 Spécification d'Entrée et de Sortie (VPL)

Entrée :

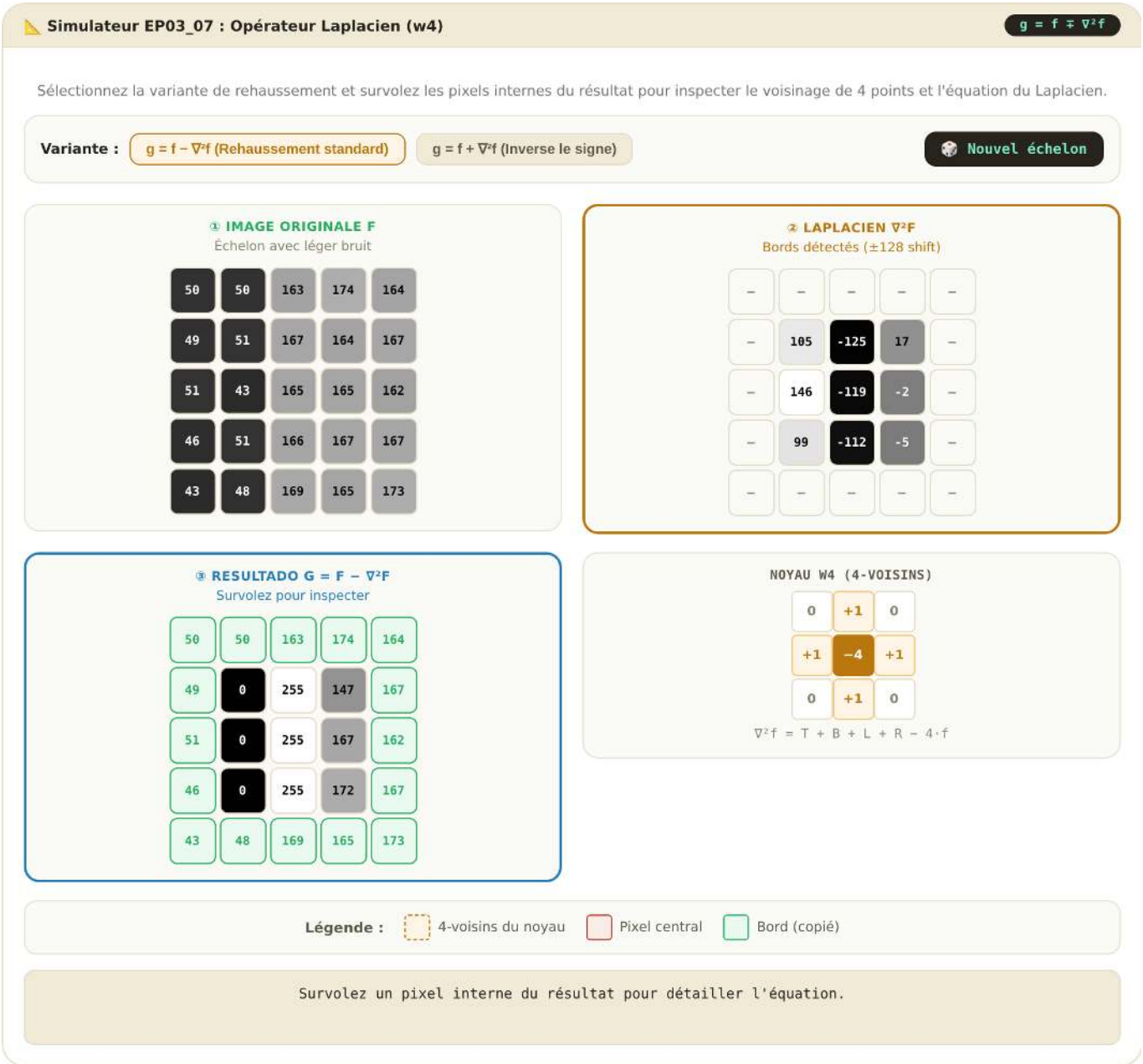
- Ligne 1 : Entier L.
- Ligne 2 : Entier C.
- Lignes suivantes : Éléments de la matrice originale.

Sortie :

- Matrice rehaussée $L \times C$.

3.12.7.5 📌 Exemples

Entrée	Sortie	Observation
3 3 0 0 0 0 100 0 0 0 0	0 0 0 0 255 0 0 0 0	Pic isolé : lap=−400, g=100−(−400)=500 → clip=255
3 3 50 50 50 50 50 50 50 50 50	50 50 50 50 50 50 50 50 50	Région uniforme : Laplacien=0, aucune modification



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.fr/cap03/sim-ep0307-laplaciano.html

Figure 3.32: Simulateur EP03_07 : Opérateur Laplacien (w4) pour le rehaussement des contours

```
%%writefile EP03_07.cpp
// your solution
```

```
Overwriting EP03_07.cpp
```

```
TestSuite("EP03_07.cpp").run()
```

```
<IPython.core.display.HTML object>
```

3.12.8 EP03_08 🧭 Gradient de Sobel : Gx et Gy

Dans les robots explorateurs de Mars (comme Perseverance), la détection d'obstacles est réalisée en temps réel par des caméras stéréoscopiques. L'**opérateur de Sobel** calcule le gradient directionnel de la scène et est utilisé dans l'algorithme de détection de contours pour identifier les roches, les fissures et les dénivelés du terrain qui pourraient compromettre la navigation.

Voir dans Figure 3.33 une simulation de cet EP.

3.12.8.1 📋 Directives d'implémentation

1. **Dimensions :** Lire les entiers L (lignes) et C (colonnes).
2. **Données :** Lire la matrice f.
3. **Gx et Gy :** Pour chaque pixel **interne** (i, j) avec $1 \leq i < L-1$, $1 \leq j < C-1$:

$$G_x(i,j) = [f(i-1,j+1) + 2f(i,j+1) + f(i+1,j+1)] - [f(i-1,j-1) + 2f(i,j-1) + f(i+1,j-1)]$$

$$G_y(i,j) = [f(i+1,j-1) + 2f(i+1,j) + f(i+1,j+1)] - [f(i-1,j-1) + 2f(i-1,j) + f(i-1,j+1)]$$

4. **Magnitude :** $|\nabla f(i,j)| = \text{clip}(\text{round}(\sqrt{G_x^2 + G_y^2}))$.
5. **Bordure :** Les pixels de bordure reçoivent une magnitude de 0.
6. **Sortie :** Afficher la magnitude $L \times C$.

3.12.8.2 📌 Contraintes computationnelles

- **Arrondi :** Appliquer `round` avant de convertir en entier.
- **Saturation :** $\text{clip}(x) = \max(0, \min(255, x))$.
- **Racine carrée :** Utiliser $\sqrt{G_x^2 + G_y^2}$ (pas l'approximation $|G_x| + |G_y|$).

3.12.8.3 🧠 Fondement théorique

Opérateur	Détecte	Coefficients diagonaux
G_x	Contours verticaux	± 1
G_y	Contours horizontaux	± 1
$\lvert\nabla f\rvert$	Tous les contours	Combiné

3.12.8.4 📦 Spécification d'entrée et de sortie (VPL)

Entrée :

- Ligne 1 : Entier L.

- Ligne 2 : Entier C.
- Lignes suivantes : Éléments de la matrice.

Sortie :

- Magnitude du gradient, matrice $L \times C$.

3.12.8.5 📌 Exemples

Entrée	Sortie	Observation
3 3 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0	Image nulle : gradient zéro
3 3 0 0 255 0 0 255 0 0 255	0 0 0 0 255 0 0 0 0	Contour vertical central : Gx élevé

```
%%writefile EP03_08.cpp
// your solution
```

```
Overwriting EP03_08.cpp
```

```
TestSuite("EP03_08.cpp").run()
```

```
<IPython.core.display.HTML object>
```

3.12.9 EP03_09 📡 Filtre Médian 3×3

Les images radar à synthèse d'ouverture (SAR) utilisées en surveillance environnementale et militaire souffrent d'un type spécifique de bruit appelé *speckle*, qui présente des caractéristiques similaires au bruit sel et poivre. Le **filtre médian** est la méthode standard pour supprimer ce bruit, car il préserve les contours des structures tout en éliminant les points parasites.

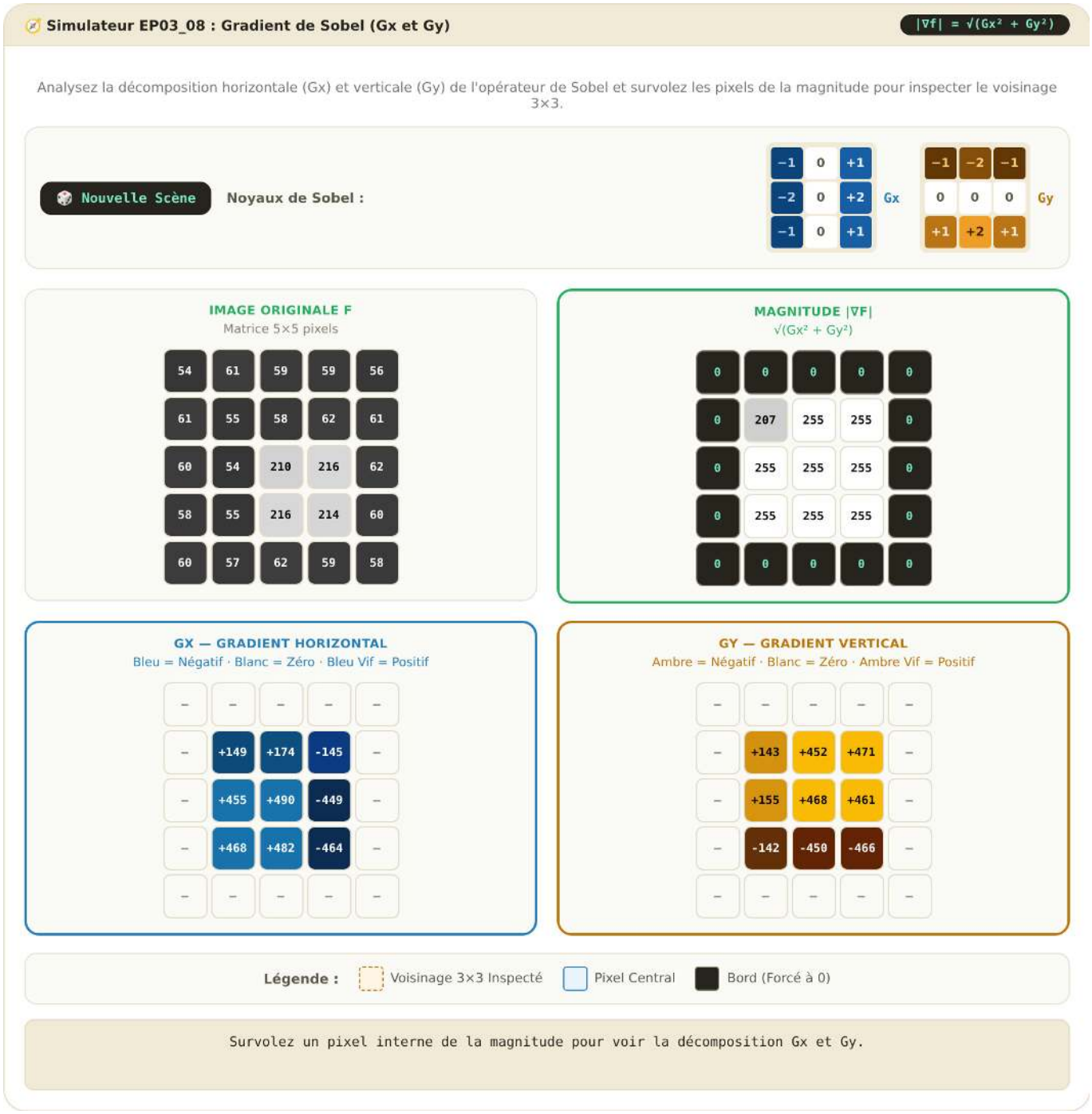
Voir dans Figure 3.34 une simulation de cet EP.

3.12.9.1 📋 Directives d'implémentation

1. **Dimensions :** Lire les entiers L (lignes) et C (colonnes).
2. **Données :** Lire la matrice de pixels f.
3. **Filtre Médian 3×3 :** Pour chaque pixel **interne** (i, j) avec $1 \leq i < L-1$, $1 \leq j < C-1$:
   - Collecter les 9 pixels du voisinage 3×3 : $\{f(i+s, j+t) : s, t \in \{-1, 0, 1\}\}$.
   - Trier les 9 valeurs en ordre croissant.
   - Affecter $g(i, j)$ à la valeur centrale (position d'indice 4, en considérant l'indice 0).

$$g(i,j) = \text{médiane}\{f(i+s, j+t) : s, t \in \{-1, 0, 1\}\}$$

4. **Bordure :** Copier directement : $g(i, j) = f(i, j)$.
5. **Sortie :** Afficher la matrice filtrée $L \times C$.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.fr/cap03/sim-ep0308-sobel.html

Figure 3.33: Simulateur EP03_08 : Gradient de Sobel (Gx et Gy)

3.12.9.2 📌 Contraintes computationnelles

- **Fenêtre :** Toujours $3 \times 3 = 9$ éléments.
- **Médiane :** L'élément central de la séquence triée (indice 4 de 0 à 8).
- **Pas de clipping :** La médiane des valeurs dans $[0, 255]$ reste dans $[0, 255]$.
- **Non linéaire :** Le filtre médian ne peut pas être exprimé comme une convolution linéaire.

3.12.9.3 🧠 Fondement théorique

Bruit	Filtre de moyenne	Filtre médian
Sel et poivre (0 ou 255)	Étale le bruit	Supprime sans déformer les contours
Gaussien	Réduit efficacement	Réduit partiellement

3.12.9.4 📦 Spécification d'entrée et de sortie (VPL)

Entrée :

- Ligne 1 : Entier L.
- Ligne 2 : Entier C.
- Lignes suivantes : Éléments de la matrice.

Sortie :

- Matrice filtrée $L \times C$.

3.12.9.5 📌 Exemples

Entrée	Sortie	Observation
3 3 100 100 100 100 0 100 100 100 100	100 100 100 100 100 100 100 100 100	Point noir éliminé : médiane de 8×100+1×0 = 100
3 3 50 50 50 50 255 50 50 50 50	50 50 50 50 50 50 50 50 50	Point blanc (sel) éliminé

```
%%writefile EP03_09.cpp
// your solution
```

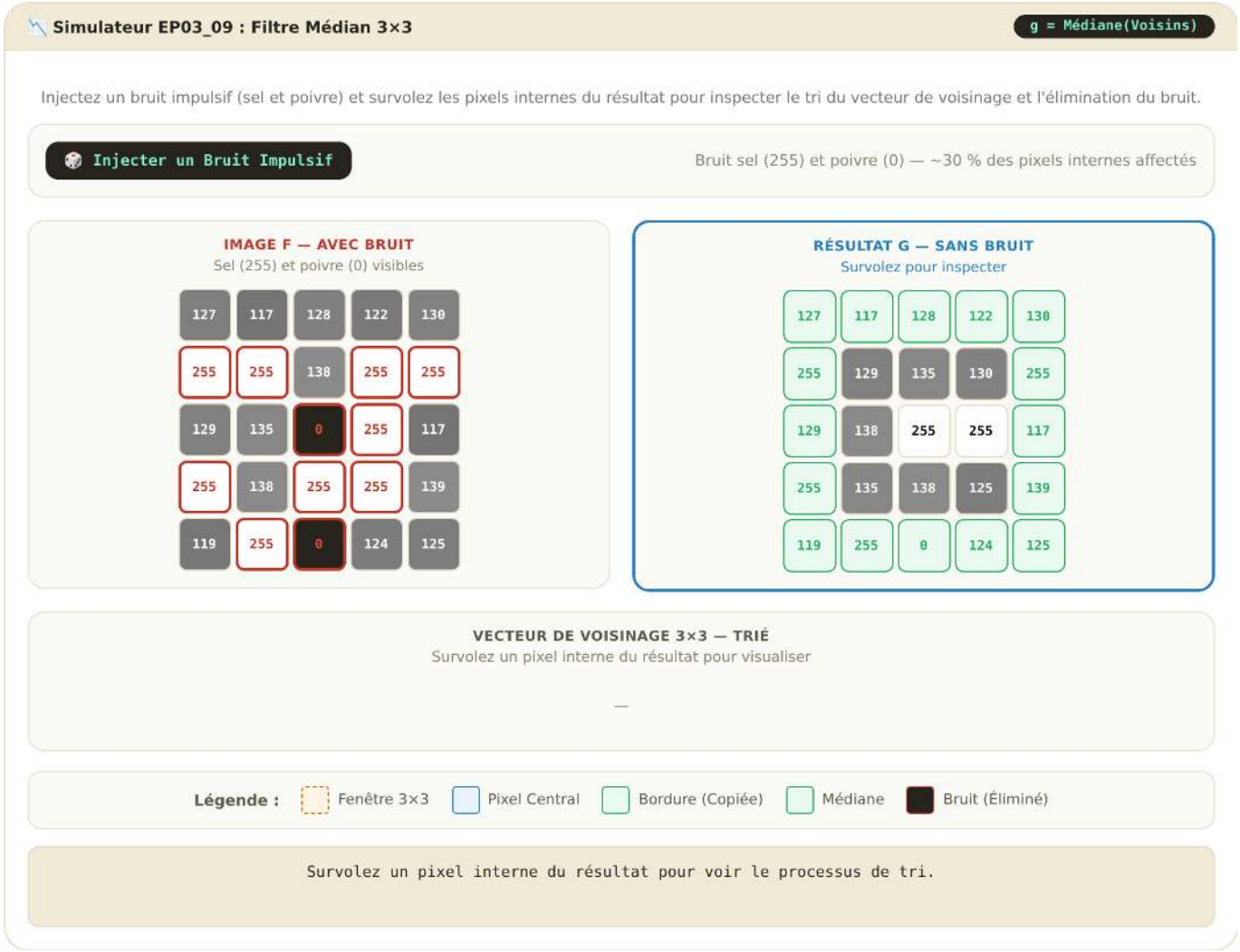
```
Overwriting EP03_09.cpp
```

```
TestSuite("EP03_09.cpp").run()
```

```
<IPython.core.display.HTML object>
```

3.12.10 EP03_10 ✨ *Unsharp Masking* (USM)

Dans les systèmes de numérisation de documents historiques et d'œuvres d'art, la netteté des images est essentielle pour la lecture de textes manuscrits et de détails ornementaux. L'***Unsharp Masking*** **(USM)**



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.fr/cap03/sim-ep0309-mediana.html

Figure 3.34: Simulateur EP03_09 : Filtre de la Médiane 3×3

est l'algorithme de rehaussement de netteté standard utilisé dans les *scanners* professionnels et les logiciels tels qu'Adobe Photoshop, contrôlé par le paramètre k qui détermine l'intensité du rehaussement.

Voir dans Figure 3.35 une simulation de cet EP.

3.12.10.1 📋 Directives d'implémentation

1. **Dimensions :** Lire les entiers L (lignes) et C (colonnes).
2. **Paramètre :** Lire la valeur réelle k (intensité du rehaussement, $k \geq 0$).
3. **Données :** Lire la matrice de pixels f.
4. **Lissage :** Calculer $\bar{f}$ avec un filtre de moyenne 3×3 (uniquement les pixels internes ; les bords sont conservés) :

$$\bar{f}(i,j) = \frac{1}{9} \sum_{s=-1}^{1} \sum_{t=-1}^{1} f(i+s, j+t)$$

5. **Masque haute fréquence :** $m(i,j) = f(i,j) - \bar{f}(i,j)$.
6. **Rehaussement USM :** Pour chaque pixel interne :

$$g(i,j) = \text{clip}\left(\text{round}\left(f(i,j) + k \cdot m(i,j)\right)\right)$$

7. **Bord :** $g(i,j) = f(i,j)$ (copie directe).
8. **Sortie :** Afficher la matrice rehaussée $L \times C$.

3.12.10.2 📌 Contraintes computationnelles

- **Arrondi :** Appliquer `round` avant le clipping.
- **Saturation :** $\text{clip}(x) = \max(0, \min(255, x))$.
- **Opérations en float :** Calculer $\bar{f}$ et m en virgule flottante avant d'arrondir le résultat final.
- $k = 0$ **:** Pas de rehaussement — la sortie est identique à l'entrée (sauf pour les bords).

3.12.10.3 🧠 Fondement théorique

Étape	Opération	Description
1	$\bar{f} = f * \frac{1}{9}\mathbf{1}_{3\times3}$	Lissage (basses fréquences)
2	$m = f - \bar{f}$	Masque (hautes fréquences)
3	$g = \text{clip}(\text{round}(f + k \cdot m))$	Rehaussement pondéré

3.12.10.4 📦 Spécification d'entrée et de sortie (VPL)

Entrée :

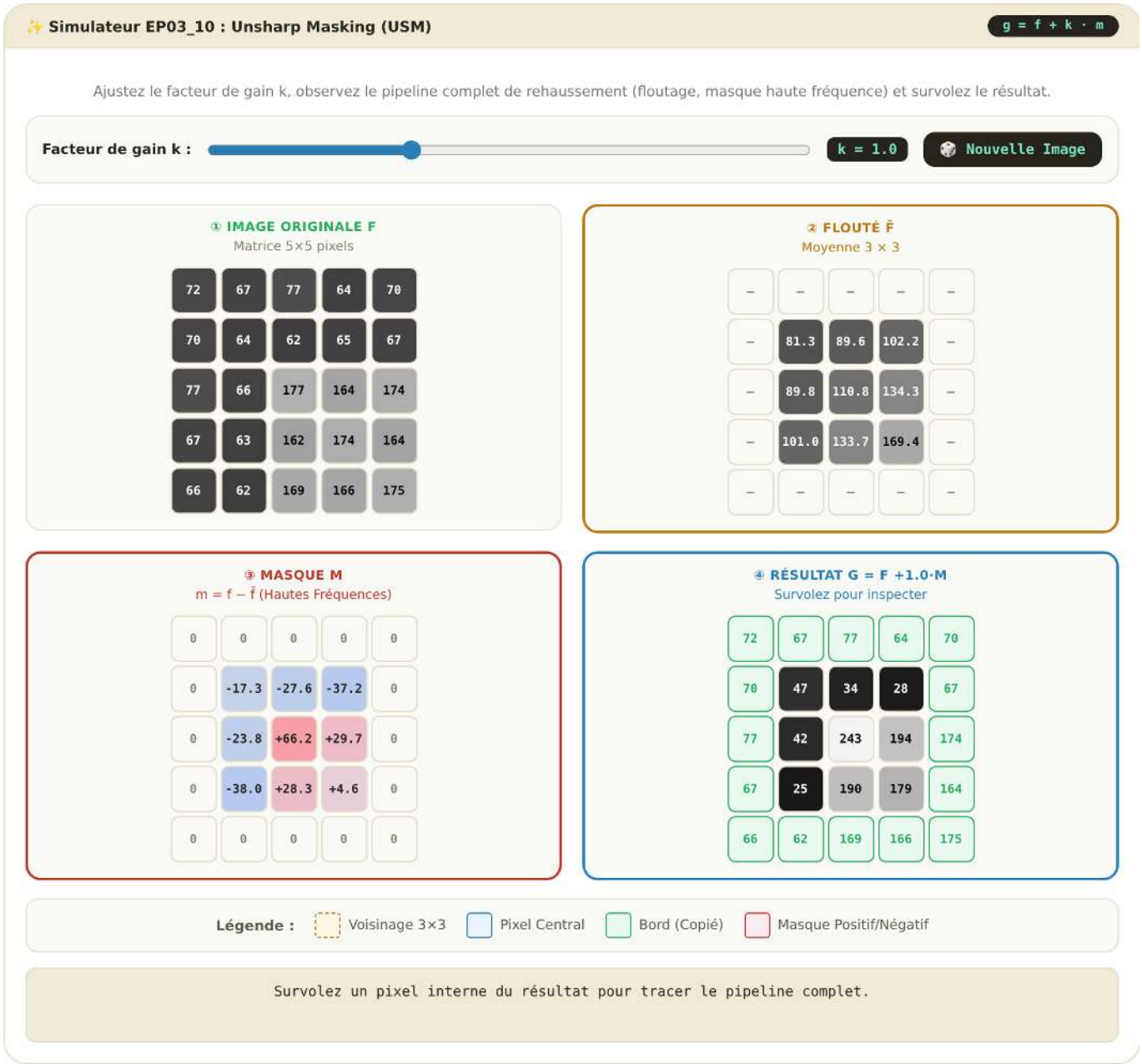
- Ligne 1 : Entier L.
- Ligne 2 : Entier C.
- Ligne 3 : Réel k.
- Lignes suivantes : Éléments de la matrice originale.

Sortie :

- Matrice rehaussée $L \times C$.

3.12.10.5 📌 Exemples

Entrée	Sortie	Observation
3 3 0.0 100 100 100 100 100 100 100 100 100	100 100 100 100 100 100 100 100 100	k=0 : pas de rehaussement
3 3 1.0 50 50 50 50 200 50 50 50 50	50 50 50 50 255 50 50 50 50	k=1 : pixel central rehaussé et saturé



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.fr/cap03/sim-ep0310-unsharp.html

Figure 3.35: Simulateur EP03_10 : *Unsharp Masking* (USM)

```
%%writefile EP03_10.cpp
// your solution
```

```
Overwriting EP03_10.cpp
```

```
TestSuite("EP03_10.cpp").run()
```

```
<IPython.core.display.HTML object>
```

Chapitre 4

Morphologie mathématique et segmentation d'images

Ce chapitre présente deux thèmes fondamentaux du traitement numérique des images (TNI) : la **morphologie mathématique** et la **segmentation d'images**. La morphologie mathématique fournit un cadre théorique fondé sur la théorie des ensembles pour analyser, affiner et quantifier la forme des objets dans les images binaires et en niveaux de gris, au moyen d'opérateurs fondamentaux tels que l'**érosion** et la **dilatation**. La segmentation, quant à elle, vise à partitionner l'image en régions d'intérêt, en séparant les objets du fond et en produisant des représentations adaptées à l'analyse et à l'interprétation.

Le chapitre débute par le **seuillage**, l'une des techniques de segmentation les plus importantes, en introduisant la méthode automatique d'**Otsu** et en revisit l'analyse des histogrammes au moyen de la variance interclasses, présentée au chapitre 1. Ensuite, sont étudiés les principaux opérateurs de la morphologie mathématique, notamment l'érosion, la dilatation, l'ouverture, la fermeture et la reconstruction morphologique, qui permettent d'affiner les masques binaires et de préserver les structures pertinentes des objets. Enfin, sont présentées des techniques de segmentation fondée sur les régions, telles que l'**étiquetage des composantes connexes**, la **transformée de distance** et l'algorithme ***watershed*** basé sur les marqueurs, aboutissant à l'extraction de descripteurs géométriques et à la génération de *bounding boxes* compatibles avec les systèmes modernes de détection d'objets.

4.1 Objectifs

À la fin de ce chapitre, vous serez capable de :

- **Appliquer le seuillage :** Comprendre le critère automatique d'Otsu par maximisation de la variance interclasse (σ_B^2) et sélectionner des stratégies de prétraitement appropriées pour faciliter la segmentation ;
- **Maîtriser la morphologie binaire :** Comprendre et appliquer l'érosion ($A \ominus B$) et la dilatation ($A \oplus B$) comme opérateurs fondamentaux, en dérivant l'ouverture ($A \circ B$), la fermeture ($A \bullet B$) et les opérations basées sur la reconstruction morphologique, comme `mm::clohole` et `mm::edgeoff` ;
- **Appliquer la morphologie en niveaux de gris :** Utiliser le gradient morphologique et les filtres *top-hat* pour le rehaussement et l'analyse de structures locales ;
- **Étiqueter les composantes connexes :** Identifier et séparer les régions connectées dans des images binaires à l'aide d'algorithmes d'étiquetage ;
- **Appliquer la transformée de distance :** Interpréter et calculer les distances au fond en utilisant des approches morphologiques et des métriques géométriques ;
- **Segmenter par régions :** Construire des *pipelines* de segmentation basés sur des marqueurs utilisant la transformée de distance et l'algorithme ***watershed*** ;
- **Extraire des descripteurs géométriques :** Calculer des propriétés telles que l'aire, le périmètre, le centroïde, la circularité et les *bounding boxes* à l'aide de `mm::label0` et de l'extraction de contours ;
- **Relier le TNI et la vision par ordinateur :** Comprendre comment les descripteurs extraits par segmentation peuvent être convertis en formats utilisés par les détecteurs modernes, comme YOLO.

```
import os, urllib.request

os.makedirs("tmp/state", exist_ok=True)  # artefacts de construction du parcours C++ (.cpp,
binaire, PNGs)

url = "https://raw.githubusercontent.com/fzampirolli/pdi-vc/master/morph/config.py"
if not os.path.exists("config.py"):
    urllib.request.urlretrieve(url, "config.py")

# Le noyau est Python même dans le parcours C++ : `mm` (morph.py) est utilisé par les
# simulateurs, par l'affichage des figures que le binaire C++ génère et par
# l'état mm::Image entre les cellules. cpp=True télécharge aussi le parcours compilé
# (morph.hpp + stb_image*.h), utilisé dans le #include des cellules %%writefile *.cpp.
# Les cellules C++ de ce chapitre compilent AVEC OpenCV (-DMM_USE_OPENCV +
# pkg-config opencv4) : mm::dil/ero → cv::dilate/erode, donc open/close/asf/
# gradm/tophat/blackhat tournent en pleine résolution et correspondent bit à bit au parcours
py.
import config
config.setup(cpp=True)
from morph import mm
import numpy as np
```

```
Environnement prêt. Morph : 1.1.9 | OpenCV : 5.0.0
```

4.2 Seuilletage

Le **seuilletage** (*thresholding*) est l'une des formes les plus simples et efficaces de segmentation d'images. Son objectif est de classer chaque pixel en deux classes d'intensité, généralement associées à *objet* et *fond* :

$$g(x,y) = \begin{cases} 255, & \text{si } f(x,y) > T \\ 0, & \text{sinon} \end{cases} \tag{4.1}$$

où $f(x,y)$ représente l'intensité du pixel dans l'image originale et $g(x,y)$ l'image binaire résultante.

Le choix du seuil T est important pour la qualité de la segmentation. La méthode d'**Otsu** détermine automatiquement le seuil optimal en maximisant la **variance interclasses** σ_B^2 définie par :

$$\sigma_B^2(T) = w_0(T)\, w_1(T)\, \left[\mu_0(T) - \mu_1(T)\right]^2 \tag{4.2}$$

où :

- $w_0(T)$ et $w_1(T)$ sont les probabilités cumulées des classes fond et objet ;
- $\mu_0(T)$ et $\mu_1(T)$ sont les moyennes d'intensité de ces classes ;
- $\sigma_B^2(T)$ représente la variance interclasses pour un seuil donné T.

La méthode fonctionne mieux lorsque l'histogramme présente deux groupes d'intensités relativement séparés. Pour cela, l'algorithme évalue tous les seuils possibles de l'image — typiquement dans l'intervalle $[0, 255]$ pour des images 8 bits — et sélectionne la valeur qui maximise la variance entre classes, notée σ_B^2 :

$$T^* = \arg\max_{T \in [0,255]} \sigma_B^2(T)$$

i Otsu suppose des histogrammes bimodaux

La méthode d'Otsu produit de meilleurs résultats lorsque l'histogramme présente deux pics bien définis (*bimodalité*), correspondant au fond et à l'objet. Plus la séparation entre ces pics est grande et plus le maximum de σ_B^2 est prononcé, plus le seuil obtenu tend à être fiable.
Dans les images avec un éclairage non uniforme ou de multiples régions d'intensité, les techniques de **seuilletage adaptatif** — dans lesquelles le seuil est calculé localement — produisent généralement des segmentations plus robustes.
L'indice B dans σ_B^2 signifie ***between classes*** (*entre classes*). Ainsi, σ_B^2 représente la **variance entre les classes** (*between-class variance*).

4.2.1 Image de pièces de monnaie

L'image utilisée pour pratiquer la segmentation est une photographie d'une collection de pièces de monnaie de différents pays et époques (Figure 4.1). Crédit : GAZI.MD.AHAD (CC BY-SA 4.0). Elle présente des objets circulaires aux contours bien définis, ce qui la rend idéale pour illustrer le seuillage, les opérateurs morphologiques, la transformée de distance, le *watershed* et les descripteurs de forme.

```
%%writefile tmp/fig_04_coins.cpp
#define MM_OUT "tmp/fig_04_coins.png"
// Compile: g++ -std=c++17 -o program program.cpp -lopencv_core -lopencv_imgproc
-lopencv_imgcodecs
#include "morph.hpp"
#include <iostream>
#include <filesystem>

int main() {
    //| label: fig-04-coins
    //| fig-cap: "Imagem com moedas de vários tipos. Crédito: GAZI.MD.AHAD (CC BY-SA 4.0)."
    //| echo: true

    // A imagem já está no diretório do capítulo (imagens/coins.png); a trilha
    // Python cuida do download/cache quando o notebook roda avulso.
    mm::Image img_coins_color = mm::read("imagens/coins.png");
    mm::Image img_coins_gray  = mm::gray(img_coins_color);

    std::cout << "Dimensões [y,x,c]: [" << img_coins_color.h << ", "
              << img_coins_color.w << ", " << img_coins_color.channels << "]\n";
    mm::show(img_coins_color, MM_OUT);


// [pdi:state-io] auto-generated - do not edit by hand
std::filesystem::create_directories("tmp/state");
mm::write(img_coins_gray, "tmp/state/img_coins_gray_8.png");
// [pdi:state-io:end]
return 0;
}
```

```
Overwriting tmp/fig_04_coins.cpp
```

```
!g++ -I. -std=c++17 -DMM_USE_OPENCV -I/usr/include/opencv4 tmp/fig_04_coins.cpp -o
tmp/fig_04_coins -lopencv_imgproc -lopencv_imgcodecs -lopencv_core \
  && ./tmp/fig_04_coins \
  && test -f "tmp/fig_04_coins.png" \
  || echo "  mm::show não gravou tmp/fig_04_coins.png"
```

```
Dimensões [y,x,c]: [2560, 1920, 3]
```

```
try:
    mm.show(mm.read("tmp/fig_04_coins.png"))
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + " tmp/fig_04_coins.png (ver
    a versao Python)")
```

Figure 4.1: Imagem com moedas de vários tipos. Crédito: GAZI.MD.AHAD (CC BY-SA 4.0).

4.2.2 Prétraitement pour Otsu

La méthode d'Otsu repose sur un histogramme nettement **bimodal**. L'image des pièces présente un éclairage non uniforme et des pièces sombres proches du fond, ce qui gêne. Dans le parcours C++, on applique une **égalisation globale d'histogramme** (`mm::equalize`) avant le seuillage — la comparaison CLAHE × Gaussien et les courbes $\sigma_B^2(T)$ se trouvent dans le parcours Python (Figure 4.2).

```
%%writefile tmp/fig_04_otsu_comparacao_histogramas.cpp
#define MM_OUT "tmp/fig_04_otsu_comparacao_histogramas.png"
```

```
// Compile: g++ -std=c++17 -o program program.cpp -I. -L. -lmorph

#include "morph.hpp"
#include <iostream>
#include <vector>
#include <string>
#include <filesystem>

int main() {
// [pdi:state-io] auto-generated - do not edit by hand
mm::Image img_coins_gray = mm::_read_state("tmp/state/img_coins_gray_8.png");
// [pdi:state-io:end]

    // img_coins_gray is already defined and valid here

    //| label: fig-04-otsu-comparacao-histogramas
    //| fig-cap: "CLAHE (realce adaptativo com limite de contraste) antes da limiarização de
    Otsu: original, realçado, histograma e binarização. (A trilha Python também traça as
    curvas de $\\sigma^2_B(T)$.)"
    //| echo: true
    //| output: true

    mm::Image img_clahe = mm::clahe(img_coins_gray, 2.0, 8);   // realce adaptativo com limite
de contraste
    mm::Image img_gauss = mm::gaussian(img_clahe, 5, 0);

    mm::show(
        std::vector<mm::Image>{img_coins_gray, img_clahe, mm::histImg(img_clahe),
mm::threshold(img_clahe)},
        MM_OUT,
        std::vector<std::string>{"Original", "CLAHE", "Histograma (CLAHE)", "Otsu"},
        4
    );


// [pdi:state-io] auto-generated - do not edit by hand
std::filesystem::create_directories("tmp/state");
mm::write(img_clahe, "tmp/state/img_clahe_12.png");
// [pdi:state-io:end]
return 0;
}
```

```
Overwriting tmp/fig_04_otsu_comparacao_histogramas.cpp
```

```
!g++ -I. -std=c++17 -DMM_USE_OPENCV -I/usr/include/opencv4
tmp/fig_04_otsu_comparacao_histogramas.cpp -o tmp/fig_04_otsu_comparacao_histogramas
-lopencv_imgproc -lopencv_imgcodecs -lopencv_core \
  && ./tmp/fig_04_otsu_comparacao_histogramas \
  && test -f "tmp/fig_04_otsu_comparacao_histogramas.png" \
  || echo "  mm::show não gravou tmp/fig_04_otsu_comparacao_histogramas.png"
```

```
[1] Original
[2] CLAHE
[3] Histograma (CLAHE)
[4] Otsu
```

```
try:
    mm.show(mm.read("tmp/fig_04_otsu_comparacao_histogramas.png"))
except Exception as _e:
```

```
print("figura indisponivel nesta trilha (C++): " + repr(_e) + "
tmp/fig_04_otsu_comparacao_histogramas.png (ver a versao Python)")
```

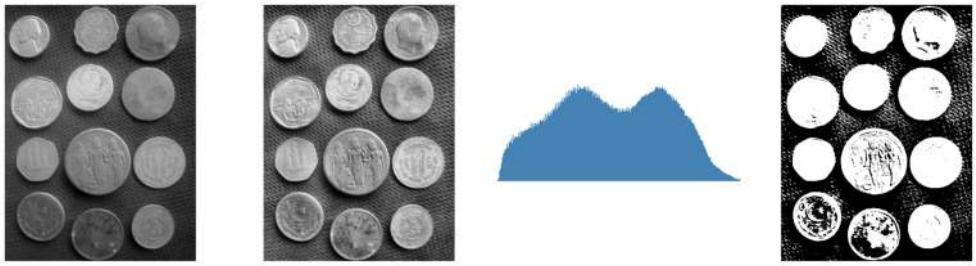


Figure 4.2: CLAHE (realce adaptativo com limite de contraste) antes da limiarização de Otsu: original, realçado, histograma e binarização. (A trilha Python também traça as curvas de $\sigma_B^2(T)$.)

4.2.3 Résultat : CLAHE comme meilleur prétraitement

L'analyse de la Figure 4.2 indique que le **CLAHE** a obtenu la valeur la plus élevée de variance inter-classes ($\sigma_B^2 \approx 2{,}47 \times 10^3$), avec un seuil optimal $T^* = 122$. Bien que la combinaison **CLAHE+Gaussien** ait produit un résultat très similaire ($\sigma_B^2 \approx 2{,}43 \times 10^3$, $T^* = 123$), le critère quantitatif de la méthode d'Otsu favorise légèrement l'utilisation du CLAHE seul.

Sur le plan visuel, les images binarisées obtenues avec le CLAHE et le CLAHE+Gaussien sont pratiquement équivalentes. La différence entre les deux approches devient plus évidente dans l'analyse des histogrammes et des valeurs de $\sigma_B^2(T)$ que dans l'inspection directe des segmentations résultantes. Ainsi, le choix du CLAHE repose principalement sur la maximisation de la séparation statistique entre les classes de fond et d'objet.

Interprétation des résultats

Remarquez que les prétraitements avec CLAHE et CLAHE+Gaussien produisent des histogrammes et des seuils optimaux très proches ($T^* = 122$ et $T^* = 123$). Par conséquent, les images binarisées résultantes sont également très similaires. Dans ce cas, la décision ne repose pas sur des différences visuelles marquantes, mais sur le critère objectif de la méthode d'Otsu : la valeur la plus élevée de σ_B^2 indique la meilleure séparation entre les classes.

4.3 Morphologie mathématique

La **morphologie mathématique** est une théorie basée sur les ensembles, utilisée pour analyser la forme et la structure des objets dans les images. Contrairement aux filtres linéaires présentés au chapitre 3, les opérateurs morphologiques sont **non linéaires**, car ils reposent sur des opérations de minimum, de maximum et d'inclusion spatiale, plutôt que sur des combinaisons linéaires d'intensités. Ces opérateurs agissent sur le voisinage de chaque pixel au moyen d'un **élément structurant** $\mathbb{B}$, chargé de définir la forme et la taille de la région analysée.

Dans les images binaires et en niveaux de gris avec des éléments plans, l'élément structurant translaté à la position x est défini spatialement comme :

$$\mathbb{B}_x = \{x + b \mid b \in \mathbb{B}\}$$

Dans les régions de bord de l'image, une partie de l'ensemble $\mathbb{B}_x$ peut dépasser le domaine physique de la scène ($\mathbb{E}$). Pour garantir la cohérence mathématique des opérateurs primitifs à ces frontières, on suppose théoriquement que l'espace extérieur au domaine de l'image est rempli avec l'**élément neutre** de

l'opération correspondante (infini positif pour l'érosion et infini négatif pour la dilatation), empêchant ainsi que l'environnement externe corrompe les structures internes de l'objet.

Lorsque l'élément structurant associe des poids à ses éléments — c'est-à-dire $b : \mathbb{B} \to \mathbb{Z}$ — il est appelé **fonction structurante** ou **élément structurant non plan**.

Développée par Matheron et Serra dans les années 1960 pour les images binaires, puis étendue aux niveaux de gris, la morphologie mathématique fonde des opérateurs tels que le gradient morphologique, le *top-hat*, le *watershed* et la transformée de distance, tous dérivés de deux primitifs : l'**érosion** et la **dilatation** [Matheron (1975); Serra (1982)].

4.3.1 Érosion et Dilatation

Les deux opérateurs primitifs sont définis de manière unifiée pour les images en niveaux de gris ($f : \mathbb{E} \to \mathbb{Z}$) et, par restriction au domaine $\{0, 1\}$, également pour les images binaires.

4.3.1.1 Érosion

L'**érosion** d'une image f par un élément structurant $b : \mathbb{B} \to \mathbb{Z}$ est définie formellement par :

$$\varepsilon_b(f)(x) = (f \ominus b)(x) = \min_{z \in \mathbb{B}} \{ f(x + z) - b(z) \}, \quad \forall\, x \in \mathbb{E} \tag{4.3}$$

En pratique, l'érosion remplace l'intensité du pixel x par la valeur minimale résultant de la différence entre l'image et l'élément structurant dans le voisinage défini par le domaine $\mathbb{B}$. Des valeurs positives dans les poids de $b(z)$ forcent le résultat local vers le bas, « creusant » plus profondément le relief de l'image et intensifiant l'érosion.

Dans le cas **plat** (où les poids sont nuls à l'intérieur du domaine, c'est-à-dire $b \equiv 0$), l'expression se simplifie en le minimum local pur :

$$\varepsilon_B(f)(x) = \min\{ f(y) : y \in \mathbb{B}_x \}$$

Dans les images binaires, cette opération équivaut à exiger que l'ensemble $\mathbb{B}$, translaté à la coordonnée x, soit **complètement contenu** dans l'objet A :

$$A \ominus \mathbb{B} = \{ z \in \mathbb{E} \mid \mathbb{B}_z \subseteq A \}$$

Effet visuel : *Rétrécit* les objets et les structures claires, éliminant les protubérances, les pics brillants ou les bruits qui sont géométriquement plus petits que le domaine $\mathbb{B}$.

4.3.1.2 Implémentation de l'érosion

La version didactique `mm::ero0` implémente le cas particulier de l'érosion avec **élément structurant plat**. Pour chaque pixel (y, x), la fonction parcourt les voisins spatiaux autorisés par B et stocke la plus petite valeur trouvée dans l'image d'entrée f, reproduisant directement l'opération de minimum local décrite dans la Équation 4.3 pour $b \equiv 0$.

Notez que les valeurs des voisins sont toujours lues de manière statique depuis l'image originale f ; la matrice de sortie g est utilisée exclusivement pour enregistrer le minimum accumulé du voisinage courant. Ainsi, le résultat final est invariant par rapport à l'ordre de balayage des pixels (que ce soit par lignes ou par colonnes).

La fonction auxiliaire `_viz` calcule les coordonnées des voisins valides dans les limites physiques de l'image. Sur les bords, l'initialisation de l'accumulateur à `255` imite avec exactitude le remplissage par élément neutre exigé par la théorie. Quant à la fonction d'interface `mm::ero`, elle recourt à l'implémentation native et optimisée d'OpenCV (`mm::ero`) lorsque l'élément structurant est plat, basculant vers la routine générale `mm::ero1` si l'élément possède des poids topographiques.

L'exemple computationnel suivant illustre l'application d'un élément structurant en croix (`mm::secross()`) mis en évidence dans la Figure 4.3, en comparant l'exécution de la variante didactique en boucle (`mm::ero0`) avec le moteur computationnel d'OpenCV (`mm::ero`).

```
%%writefile tmp/fig_04_elemento_cruz.cpp
#define MM_OUT "tmp/fig_04_elemento_cruz.png"
//| label: fig-04-elemento-cruz
//| fig-cap: "Elemento estruturante em formato de cruz ($B_{\\text{cruz}}$) utilizado para
conectividade-4."
//| echo: true
//| output: true

#include "morph.hpp"
#include <iostream>

int main() {
    mm::Image B_cruz = mm::secross();
    mm::drawImgPlt(B_cruz, MM_OUT, 40);
    return 0;
}
```

```
Overwriting tmp/fig_04_elemento_cruz.cpp
```

```
!g++ -I. -std=c++17 -DMM_USE_OPENCV -I/usr/include/opencv4 tmp/fig_04_elemento_cruz.cpp -o
tmp/fig_04_elemento_cruz -lopencv_imgproc -lopencv_imgcodecs -lopencv_core \
  && ./tmp/fig_04_elemento_cruz \
  && test -f "tmp/fig_04_elemento_cruz.png" \
  || echo "  mm::show não gravou tmp/fig_04_elemento_cruz.png"
```

```
0 1 0
1 1 1
0 1 0
```

```
try:
    mm.show(mm.read("tmp/fig_04_elemento_cruz.png"))
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + "
    tmp/fig_04_elemento_cruz.png (ver a versao Python)")
```

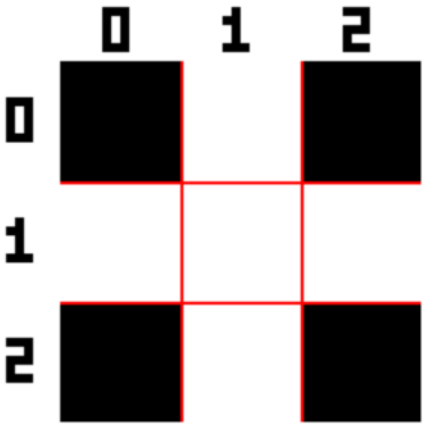


Figure 4.3: Elemento estruturante em formato de cruz (B_{cruz}) utilizado para conectividade-4.

```
# La morph.hpp est header-only ; ci-dessous, le helper _viz et le corps de mm::ero0().
import re, pathlib
hpp = pathlib.Path("morph.hpp").read_text()
for nome, pat in [("_viz", r"template <class F>\ninline void _viz\(.*?\n\}"),
                  ("ero0", r"inline Image ero0\(.*?\n\}")]:
    m = re.search(pat, hpp, re.S)
    print(f"// ---- mm::{nome} ----")
    print(m.group(0) if m else f"({nome} não encontrada)")
    print()
```

```
// ---- mm::_viz ----
template <class F>
inline void _viz(const Image& f, const SE& B, int y, int x, F&& cb) {
    double oh = -B.h / 2.0 + 0.5;
    double ow = -B.w / 2.0 + 0.5;
    for (int by = 0; by < B.h; ++by)
        for (int bx = 0; bx < B.w; ++bx) {
            int vy = (int)(y + by + oh);
            int vx = (int)(x + bx + ow);
            if (vy >= 0 && vy < f.h && vx >= 0 && vx < f.w)
                cb(vy, vx, B.at(by, bx));
        }
}

// ---- mm::ero0 ----
inline Image ero0(const Image& f, SE Bc = SE::box(3)) {
    _require_gray(f, "ero0");
    Image g(f.h, f.w, 1);
    for (int y = 0; y < f.h; ++y)
        for (int x = 0; x < f.w; ++x) {
            int mn = 255;
            _viz(f, Bc, y, x, [&](int vy, int vx, int bv) {
                if (bv != 0 && (int)f.at(vy, vx) < mn) mn = f.at(vy, vx);
            });
            g.at(y, x) = (unsigned char)mn;
        }
    return g;
}
```

4.3.1.3 Dilatation

La **Dilatation** d'une image f par une fonction structurante $b : \mathbb{B} \to \mathbb{Z}$ est définie formellement par :

$$\delta_b(f)(x) = (f \oplus b)(x) = \max_{z \in \mathbb{B}} \{ f(x - z) + b(z) \}, \quad \forall\, x \in \mathbb{E} \tag{4.4}$$

En pratique, la dilatation remplace l'intensité du pixel x par la plus grande valeur résultant de la somme entre l'image et l'élément structurant dans le voisinage défini. L'argument d'inversion spatiale $(x - z)$ indique que la dilatation évalue implicitement l'élément transposé (réfléchi) $\hat{b}$, propriété fondamentale pour assurer la dualité mathématique par rapport à l'érosion.

Dans le cas **plat** (où les poids sont nuls à l'intérieur du domaine, c'est-à-dire $b \equiv 0$), l'expression se réduit au maximum local pur :

$$\delta_B(f)(x) = \max\{ f(y) : y \in \mathbb{B}_x \}$$

Dans les images binaires, cette opération équivaut à exiger que l'ensemble réfléchi $\hat{\mathbb{B}}$, translaté à la coordonnée x, possède une **intersection non vide** avec l'objet A :

$$A \oplus \mathbb{B} = \{\, z \in \mathbb{E} \mid \hat{\mathbb{B}}_z \cap A \neq \emptyset \,\}$$

Effet visuel : *Étend* les structures claires de l'image, augmente le remplissage des objets, connecte les composantes proches et élimine les canaux, fosses sombres ou vallées qui sont géométriquement plus petits que le domaine $\mathbb{B}$.

4.3.1.4 Implémentation de la dilatation

La version didactique `mm::dil0` implémente le cas particulier de dilatation avec **élément structurant plan**. Pour chaque pixel (y, x), la fonction parcourt les voisins spatiaux autorisés par B et stocke la plus grande valeur trouvée dans l'image d'entrée f, reproduisant directement l'opération de maximum local pour $b \equiv 0$.

Avant de lancer le balayage spatial, l'élément structurant subit une réflexion géométrique au moyen de l'instruction `np.flip(Bc)` afin de construire explicitement la matrice transposée $\hat{B}$ exigée par la théorie. Pour des masques parfaitement symétriques (tels que des croix, des carrés et des disques centrés à l'origine), cette réflexion ne modifie pas l'arrangement des pixels ; toutefois, pour les éléments asymétriques, cette étape est strictement nécessaire afin de garantir l'équivalence avec les définitions formelles et de préserver les lois de dualité.

Comme constaté pour l'opérateur d'érosion, les valeurs des voisins sont toujours lues de manière statique à partir de la matrice originale f, tandis que la matrice de sortie g agit uniquement comme le registre du maximum accumulé du voisinage. Aux frontières de l'image, l'initialisation de l'accumulateur à `0` émule avec exactitude le remplissage externe par élément neutre ($-\infty$, ou zéro dans les représentations sur 8 bits), garantissant que les bords physiques de la scène soient dilatés en parfaite conformité avec le standard adopté par OpenCV.

L'exemple informatique ci-dessous illustre l'application pratique d'un élément en croix (`mm::secross()`), validant la cohérence entre la logique en boucles (`mm::dil0`) et la méthode native industrielle (`mm::dil`).

```python
# Corps de mm::dil0() dans morph.hpp (reflète le SE avant de balayer, comme np.flip).
import re, pathlib
hpp = pathlib.Path("morph.hpp").read_text()
m = re.search(r"inline Image dil0\(.*?\n\}", hpp, re.S)
print(m.group(0) if m else "(dil0 não encontrada)")
```

```
inline Image dil0(const Image& f, SE Bc = SE::box(3)) {
    _require_gray(f, "dil0");
    SE B = Bc.reflected();
    Image g(f.h, f.w, 1);
    for (int y = 0; y < f.h; ++y)
        for (int x = 0; x < f.w; ++x) {
            int mx = 0;
            _viz(f, B, y, x, [&](int vy, int vx, int bv) {
                if (bv != 0 && (int)f.at(vy, vx) > mx) mx = f.at(vy, vx);
            });
            g.at(y, x) = (unsigned char)mx;
        }
    return g;
}
```

i Note : Le Confront des Signes ($f(x+z)$ vs $f(x-z)$)

Comparez les définitions formelles de l'érosion (Équation 4.3) et de la dilatation (Équation 4.4). Considérez un élément structurant asymétrique à droite $\mathbb{B} = \{0, 1\}$ (origine et un pixel à droite) appliqué à la position $x = 10$.

1. **Dans l'érosion** (Équation 4.3) :

$$\min\{f(x+z) - b(z)\}$$

- $z = 0 \Rightarrow f(10+0) = \mathbf{f(10)}$
- $z = 1 \Rightarrow f(10+1) = \mathbf{f(11)}$

L'opérateur consulte le pixel courant (10) et le pixel à droite (11), préservant l'orientation originale de $\mathbb{B}$.

2. **Dans la dilatation** (Équation 4.4) :

$$\max\{f(x-z) + b(z)\}$$

- $z = 0 \Rightarrow f(10-0) = \mathbf{f(10)}$
- $z = 1 \Rightarrow f(10-1) = \mathbf{f(9)}$

En raison du signe négatif ($-z$), avancer dans l'élément structurant correspond à reculer dans l'image, ce qui fait que la dilatation consulte le pixel à gauche (9).

La fonction `_viz`, utilisée dans `morph.py`, génère des voisins par déplacements additifs de la forme $x+z$. Pour cette raison, l'implémentation de `mm::dil0` reflète au préalable l'élément structurant au moyen de `np.flip(B)`. Après la réflexion, le balayage basé sur $x+z$ accède alors exactement aux mêmes points définis par l'expression théorique $f(x-z)$ de la dilatation dans Équation 4.4.

Pour les éléments structurants symétriques (comme les disques, les carrés et les croix centrées), la réflexion ne modifie pas le masque. En revanche, pour les éléments asymétriques, cette étape est indispensable pour que l'implémentation reproduise correctement la définition mathématique de la dilatation et préserve la dualité érosion–dilatation.

i Dualité érosion–dilatation

L'érosion et la dilatation sont **duales par complémentation**. Cela signifie qu'un opérateur peut être entièrement obtenu à partir de l'autre, à condition d'agir sur le complément de l'image en utilisant l'élément structurant réfléchi $\hat{B}$:

$$(A \ominus B)^c = A^c \oplus \hat{B} \quad \Longleftrightarrow \quad A \ominus B = (A^c \oplus \hat{B})^c$$

De manière analogue, la **dilatation peut également être obtenue à partir de l'érosion** :

$$(A \oplus B)^c = A^c \ominus \hat{B} \quad \Longleftrightarrow \quad A \oplus B = (A^c \ominus \hat{B})^c$$

En termes pratiques, l'érosion d'un objet peut être obtenue par la dilatation de son complément, suivie de la complémentation du résultat (et vice-versa). Dans l'implémentation du paquet `morph.py`, les versions didactiques `mm::ero0` et `mm::dil0` rendent explicite cette structure au moyen de boucles (*loops*), tandis que `mm::ero` et `mm::dil` délèguent les opérations à OpenCV pour une plus grande efficacité computationnelle.

i Conditions aux limites et images finies

En morphologie mathématique classique, définie sur un domaine infini (typiquement $\mathbb{Z}^2$), cette dualité est exacte. Dans les images numériques, cependant, on travaille avec des matrices finies, et le résultat dépend de la manière dont sont traités les pixels situés hors de l'image.

Pour que les identités de dualité restent valides, le complément doit être défini par rapport au même univers et les conditions aux limites adoptées pour l'érosion et pour la dilatation doivent être complémentaires entre elles. Par exemple, si l'érosion suppose que les pixels externes appartiennent à l'objet (255), alors la dilatation appliquée au complément doit supposer que ces mêmes pixels externes appartiennent au fond (0).

Lorsque différentes stratégies de remplissage sont utilisées (réplication, réflexion, valeur constante, etc.), la dualité théorique peut cesser d'être satisfaite exactement dans les régions proches des bords de

l'image.

Pour illustrer numériquement les opérateurs morphologiques et la dualité érosion–dilatation, la Figure 4.4 présente une image binaire 10×10 traitée avec un élément structurant en forme de « L ». Dans l'implémentation de `morph.py`, l'origine de B est fixée au centre géométrique du masque — position $(1, 1)$ pour un *kernel* 3×3 — et doit correspondre à un élément actif pour que l'érosion se comporte correctement (comme discuté précédemment). L'élément structurant B_L défini ci-dessous satisfait cette condition. La Figure 4.5 complète l'analyse avec un simulateur interactif de l'érosion, permettant de visualiser le déplacement de l'élément structurant sur l'image et d'identifier les positions où il reste complètement contenu dans l'objet.

```
%%writefile tmp/fig_04_ero_dil_didatico.cpp
#define MM_OUT "tmp/fig_04_ero_dil_didatico.png"
//| label: fig-04-ero-dil-didatico
//| fig-cap: "Erosão e dilatação em imagem binária 10×10 com elemento estruturante 'L'
assimétrico 3×3. Validação da dualidade erosão-dilatação."
//| echo: true
//| output: true
#include "morph.hpp"
#include <iostream>
#include <vector>
#include <string>
#include <filesystem>

int main() {
    mm::Image A(10, 10);
    // Initialize A with the pattern * 255
    unsigned char pattern[10][10] = {
        {0,0,0,0,0,0,0,0,0,0},
        {0,0,0,1,1,1,0,0,0,0},
        {0,0,1,1,1,1,1,0,0,0},
        {0,1,1,1,1,1,1,1,0,0},
        {0,1,1,1,1,1,1,1,0,0},
        {0,1,1,1,1,1,1,0,0,0},
        {0,0,1,1,1,1,1,1,0,0},
        {0,0,0,1,1,1,1,0,0,0},
        {0,0,0,0,1,0,0,0,0,0},
        {0,0,0,0,0,0,0,0,0,0}
    };
    for (int y = 0; y < 10; y++) {
        for (int x = 0; x < 10; x++) {
            A.at(y, x) = pattern[y][x] * 255;
        }
    }

    // B_L: 'L' shape, origin at center
    mm::SE B_L{{1,0,0},{1,1,0},{1,1,0}};
    // B_hat: B_L rotated 180° (B)
    mm::SE B_hat{{0,1,1},{0,1,1},{0,0,1}};

    std::cout << "Elemento estruturante B_L:" << std::endl;
    mm::Image B_L_img(3, 3);
    for (int y = 0; y < 3; y++) {
        for (int x = 0; x < 3; x++) {
            B_L_img.at(y, x) = (B_L.at(y, x) != 0) ? 255 : 0;
        }
    }
    std::cout << mm::drawImg(B_L_img) << std::endl;
    std::cout << "Elemento estruturante refletido B:" << std::endl;
    mm::Image B_hat_img(3, 3);
```

```
    for (int y = 0; y < 3; y++) {
        for (int x = 0; x < 3; x++) {
            B_hat_img.at(y, x) = (B_hat.at(y, x) != 0) ? 255 : 0;
        }
    }
    std::cout << mm::drawImg(B_hat_img) << std::endl;

    mm::Image img_ero0 = mm::ero0(A, B_L);

    // Dualité : (A   B)  == A   B
    mm::Image A_c = mm::neg(A);
    mm::Image ero_c = mm::neg(img_ero0);
    mm::Image dil_Ac = mm::dil0(A_c, B_hat);

    bool dual = true;
    for (int i = 0; i < ero_c.data.size(); i++) {
        if (ero_c.data[i] != dil_Ac.data[i]) {
            dual = false;
            break;
        }
    }
    std::cout << "Dualité (A   B)  == A   B : " << (dual ? "True" : "False") << std::endl;

    mm::Image B_L_disp(3, 3);
    for (int y = 0; y < 3; y++) {
        for (int x = 0; x < 3; x++) {
            B_L_disp.at(y, x) = (B_L.at(y, x) != 0) ? 255 : 0;
        }
    }
    mm::Image B_hat_disp(3, 3);
    for (int y = 0; y < 3; y++) {
        for (int x = 0; x < 3; x++) {
            B_hat_disp.at(y, x) = (B_hat.at(y, x) != 0) ? 255 : 0;
        }
    }
    mm::show(std::vector<mm::Image>{A, A_c, img_ero0, ero_c, dil_Ac},
             MM_OUT,
             std::vector<std::string>{"A", "A ", "A   B", "(A   B) ", "A   B"},
             5);


// [pdi:panel-io] auto-generated - do not edit by hand
std::filesystem::create_directories("tmp");
mm::write(A, "tmp/fig_04_ero_dil_didatico_0.png");
mm::write(A_c, "tmp/fig_04_ero_dil_didatico_1.png");
mm::write(img_ero0, "tmp/fig_04_ero_dil_didatico_2.png");
mm::write(ero_c, "tmp/fig_04_ero_dil_didatico_3.png");
mm::write(dil_Ac, "tmp/fig_04_ero_dil_didatico_4.png");
// [pdi:panel-io:end]
return 0;
}
```

```
Overwriting tmp/fig_04_ero_dil_didatico.cpp
```

```
!g++ -I. -std=c++17 -DMM_USE_OPENCV -I/usr/include/opencv4 tmp/fig_04_ero_dil_didatico.cpp -o
tmp/fig_04_ero_dil_didatico -lopencv_imgproc -lopencv_imgcodecs -lopencv_core \
  && ./tmp/fig_04_ero_dil_didatico \
  && test -f "tmp/fig_04_ero_dil_didatico.png" \
  || echo "  mm::show não gravou tmp/fig_04_ero_dil_didatico.png"
```

```
Elemento estruturante B_L:
255   0   0
255 255   0
255 255   0

Elemento estruturante refletido B:
  0 255 255
  0 255 255
  0   0 255

Dualité (A   B)  == A   B : True
[1] A
[2] A
[3] A   B
[4] (A   B)
[5] A   B
```

```
try:
    mm.show(
        [
            mm.read("tmp/fig_04_ero_dil_didatico_0.png"),
            mm.read("tmp/fig_04_ero_dil_didatico_1.png"),
            mm.read("tmp/fig_04_ero_dil_didatico_2.png"),
            mm.read("tmp/fig_04_ero_dil_didatico_3.png"),
            mm.read("tmp/fig_04_ero_dil_didatico_4.png"),
        ],
        titles=[
            'A',
            'A ',
            'A   B',
            '(A   B) ',
            'A   B',
        ],
        cols=5,
        figsize=(15, 3),
    )
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + "
    tmp/fig_04_ero_dil_didatico_0.png (ver a versao Python)")
```

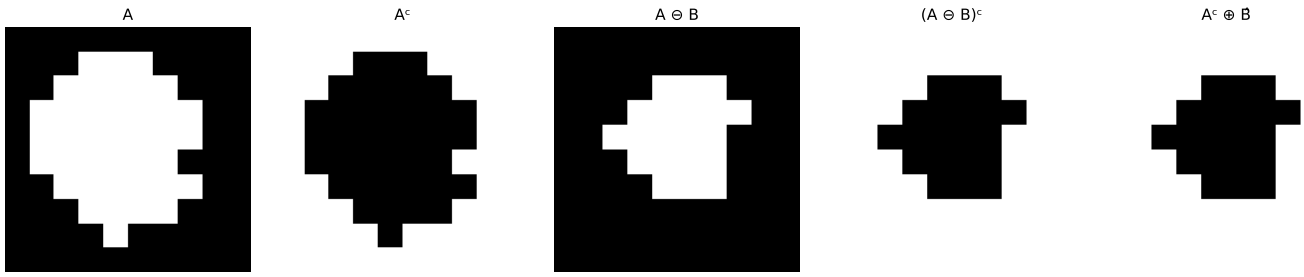
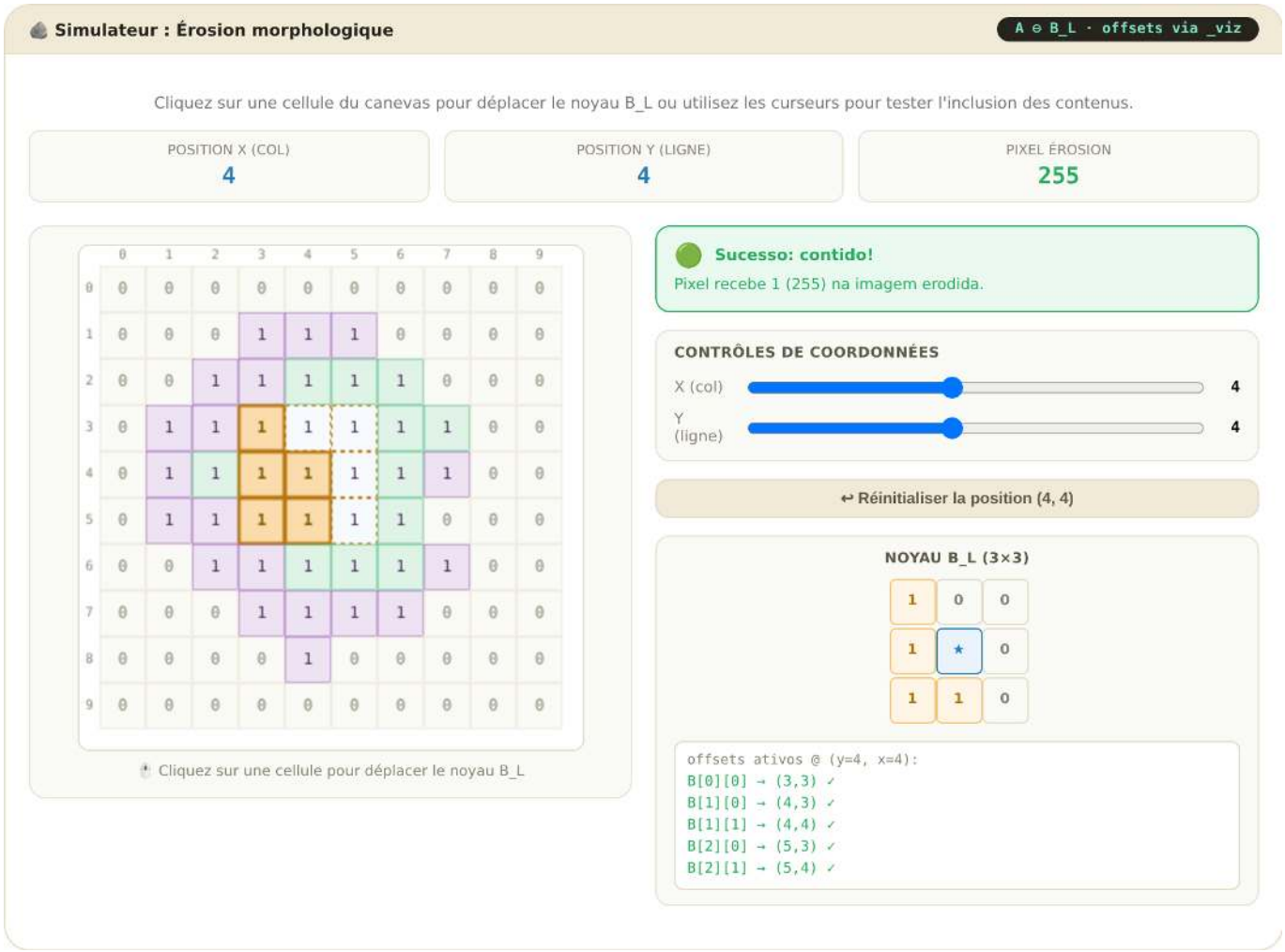


Figure 4.4: Erosão e dilatação em imagem binária 10×10 com elemento estruturante ‘L’ assimétrico 3×3. Validação da dualidade erosão–dilatação.

4.3.2 Ouverture et Fermeture

En combinant érosion et dilatation, on obtient deux opérateurs d’une grande utilité pratique : l’**ouverture** et la **fermeture**, définis par les Équations Équation 4.5 et Équation 4.6. Leurs principaux effets sont résumés dans la Table 4.1.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.fr/cap04/sim-04-erosao.html

Figure 4.5: Simulateur : Érosion Morphologique (A B_L)

Ouverture (*opening*) — érosion suivie d'une dilatation par le même B :

$$A \circ B = (A \ominus B) \oplus B \tag{4.5}$$

Fermeture (*closing*) — dilatation suivie d'une érosion par le même B :

$$A \bullet B = (A \oplus B) \ominus B \tag{4.6}$$

En pratique, `mm::open` et `mm::close` appliquent le même élément structurant aux deux étapes. Pour les éléments structurants symétriques (les plus courants), cette implémentation coïncide avec la définition mathématique présentée ci-dessus.

Tableau 4.1: Propriétés de l'ouverture et de la fermeture.

Opérateur	Séquence	Effet principal
Ouverture $A \circ B$	érosion → dilatation	Supprime les structures incapables de contenir l'élément structurant ; lisse les contours externes
Fermeture $A \bullet B$	dilatation → érosion	Comble les trous plus petits que B ; lisse les contours internes

Propriété importante : les deux sont **idempotents**. Par exemple,

$$(A \circ B) \circ B = A \circ B,$$

c'est-à-dire qu'après la première application, de nouvelles applications du même opérateur ne modifient plus le résultat.

4.3.2.1 Implémentation de l'ouverture et de la fermeture

Contrairement à l'érosion et à la dilatation, l'ouverture et la fermeture n'introduisent pas de nouveaux mécanismes computationnels. Les deux sont obtenus par la composition séquentielle des opérateurs primitifs déjà présentés :

```
def open0(f, B):
    return mm.dil0(mm.ero0(f, B), B)

def close0(f, B):
    return mm.ero0(mm.dil0(f, B), B)
```

La fonction `mm::open` délègue l'opération à `mm::open(f, B)`, tandis que `mm::close` utilise `mm::close(f, B)`, produisant le même résultat de manière plus efficace.

L'ouverture hérite de l'érosion la capacité de supprimer les structures plus petites que l'élément structurant et de la dilatation la restauration partielle des régions préservées. La fermeture réalise le processus inverse : elle expand d'abord les objets puis restaure leurs dimensions originales, comblant les lacunes et les trous plus petits que l'élément structurant.

4.3.2.2 Filtre Séquentiel Alterné

En pratique, l'ouverture et la fermeture sont souvent appliquées en séquence afin d'éliminer simultanément le bruit externe et de combler les trous internes. La fonction `mm::asf` (*Alternating Sequential Filter*) généralise cette stratégie en appliquant alternativement des ouvertures et des fermetures avec des éléments structurants progressivement plus grands. Les séquences disponibles sont présentées dans Table 4.2.

Tableau 4.2: Séquences du filtre séquentiel alterné mm.asf.

Séquence	Ordre	Utilisation typique
`'OC'`	ouverture → fermeture	élimine le bruit externe avant de combler les petits trous
`'CO'`	fermeture → ouverture	comble les petits trous avant d'éliminer le bruit externe
`'OCO'`	ouverture → fermeture → ouverture	met l'accent sur l'élimination du bruit externe
`'COC'`	fermeture → ouverture → fermeture	met l'accent sur le comblement des trous et des lacunes

Le paramètre `n` contrôle le nombre d'échelles utilisées par le filtre. À chaque itération i, l'élément structurant est agrandi par somme de Minkowski (`mm::sesum(b, i)`), produisant une séquence de filtres morphologiques de plus en plus englobants. Contrairement à une ouverture ou une fermeture unique avec un grand élément structurant, l'ASF réalise un lissage progressif à plusieurs échelles, préservant mieux la géométrie des objets pertinents tout en éliminant les structures plus petites. Figure 4.6 présente un exemple d'application de l'ouverture, de la fermeture et de l'ASF sur l'image des pièces de monnaie.

```
%%writefile tmp/fig_04_open_close.cpp
#define MM_OUT "tmp/fig_04_open_close.png"
// Compile: g++ -std=c++17 -o program program.cpp -I. -lstdc++fs -lpthread -lcurl

#include "morph.hpp"
#include <iostream>
#include <vector>
#include <string>
#include <filesystem>

int main() {
// [pdi:state-io] auto-generated - do not edit by hand
mm::Image img_clahe = mm::_read_state("tmp/state/img_clahe_12.png");
// [pdi:state-io:end]

    //| label: fig-04-open-close
    //| fig-cap: "Abertura, fechamento, composição e filtro sequencial alternado aplicados à
    binarização Otsu das moedas (pré-processadas por realce de contraste). Elemento
    estruturante: disco 13×13."
    //| echo: true
    //| output: true

    mm::Image img_bin    = mm::threshold(img_clahe);
    mm::Image B_disk     = mm::sedisk(19);

    mm::Image img_open  = mm::open(img_bin, B_disk);            // erosão → dilatação
    mm::Image img_close = mm::close(img_bin, B_disk);           // dilatação → erosão
    mm::Image img_oc    = mm::close(img_open, B_disk);          // abertura seguida de
fechamento
    mm::Image img_asf   = mm::asf(img_bin, "OC", mm::sedisk(3), 7);
    // ASF: disco base 3×3, cresce a cada iteração

    mm::show(
        std::vector<mm::Image>{img_bin, img_open, img_close, img_oc, img_asf},
        MM_OUT,
        std::vector<std::string>{"Binarização Otsu", "Abertura (A B)", "Fechamento (A B)",
            "Abertura→Fechamento", "ASF-OC (n=7)"},
        5
    );
```

```
// [pdi:panel-io] auto-generated - do not edit by hand
std::filesystem::create_directories("tmp");
mm::write(img_bin, "tmp/fig_04_open_close_0.png");
mm::write(img_open, "tmp/fig_04_open_close_1.png");
mm::write(img_close, "tmp/fig_04_open_close_2.png");
mm::write(img_oc, "tmp/fig_04_open_close_3.png");
mm::write(img_asf, "tmp/fig_04_open_close_4.png");
// [pdi:panel-io:end]
return 0;
}
```

```
Overwriting tmp/fig_04_open_close.cpp
```

```
!g++ -I. -std=c++17 -DMM_USE_OPENCV -I/usr/include/opencv4 tmp/fig_04_open_close.cpp -o
tmp/fig_04_open_close -lopencv_imgproc -lopencv_imgcodecs -lopencv_core \
  && ./tmp/fig_04_open_close \
  && test -f "tmp/fig_04_open_close.png" \
  || echo "  mm::show não gravou tmp/fig_04_open_close.png"
```

```
[1] Binarização Otsu
[2] Abertura (A B)
[3] Fechamento (A B)
[4] Abertura→Fechamento
[5] ASF-OC (n=7)
```

```
try:
    mm.show(
        [
            mm.read("tmp/fig_04_open_close_0.png"),
            mm.read("tmp/fig_04_open_close_1.png"),
            mm.read("tmp/fig_04_open_close_2.png"),
            mm.read("tmp/fig_04_open_close_3.png"),
            mm.read("tmp/fig_04_open_close_4.png"),
        ],
        titles=[
            'Binarização Otsu',
            'Abertura (A B)',
            'Fechamento (A B)',
            'Abertura→Fechamento',
            'ASF-OC (n=7)',
        ],
        cols=5,
        figsize=(15, 12),
    )
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + "
    tmp/fig_04_open_close_0.png (ver a versao Python)")
```

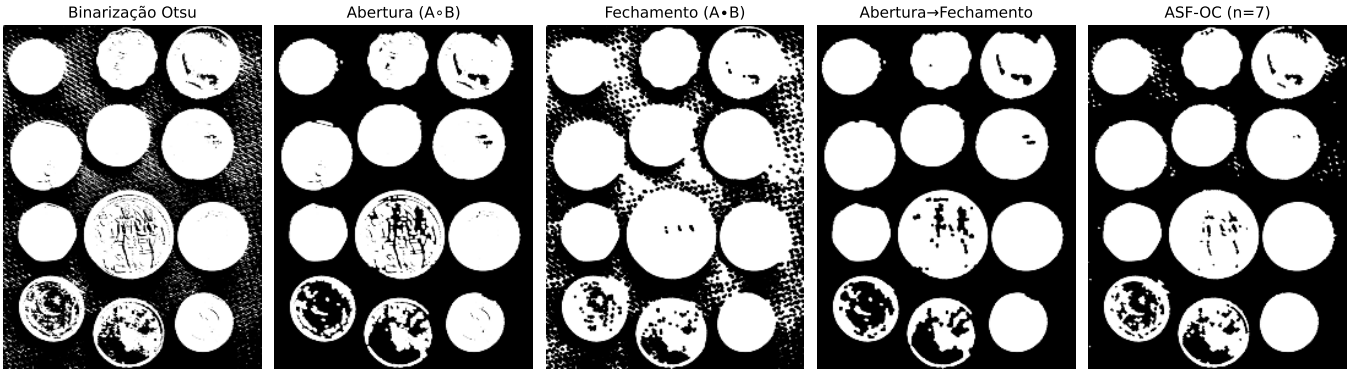


Figure 4.6: Abertura, fechamento, composição e filtro sequencial alternado aplicados à binarização Otsu das moedas (pré-processadas par realce de contraste). Elemento estruturante: disco 13×13.

4.3.3 Opérateurs Géodésiques

Les opérateurs géodésiques introduisent une contrainte supplémentaire aux opérateurs morphologiques classiques au moyen d'une image de contrôle appelée **masque** g. Au lieu de permettre à l'érosion ou à la dilatation de se propager librement à travers l'image, le résultat de chaque itération est limité point par point par les valeurs du masque, restreignant l'évolution de l'opération aux régions autorisées.

4.3.3.1 Dilatation Géodésique

La **dilatation géodésique** d'une image marqueur f sous une image masque g, à l'aide d'un élément structurant plat b, est définie par :

$$f \oplus_g b = (f \oplus b) \wedge g, \tag{4.7}$$

où $\wedge$ représente le minimum point par point.

En d'autres termes, on effectue d'abord une dilatation conventionnelle sur le marqueur, puis le résultat est restreint par le masque g. Ainsi, la propagation ne peut jamais dépasser les régions autorisées par le masque.

La formulation classique de la dilatation géodésique suppose que le marqueur est contenu dans le masque, c'est-à-dire $f \leq g$, garantissant que l'évolution de l'opération reste toujours limitée par le masque.

4.3.3.2 Implémentation de la dilatation géodésique

La fonction `mm::cdil` implémente directement cet opérateur et permet d'exécuter plusieurs itérations consécutives :

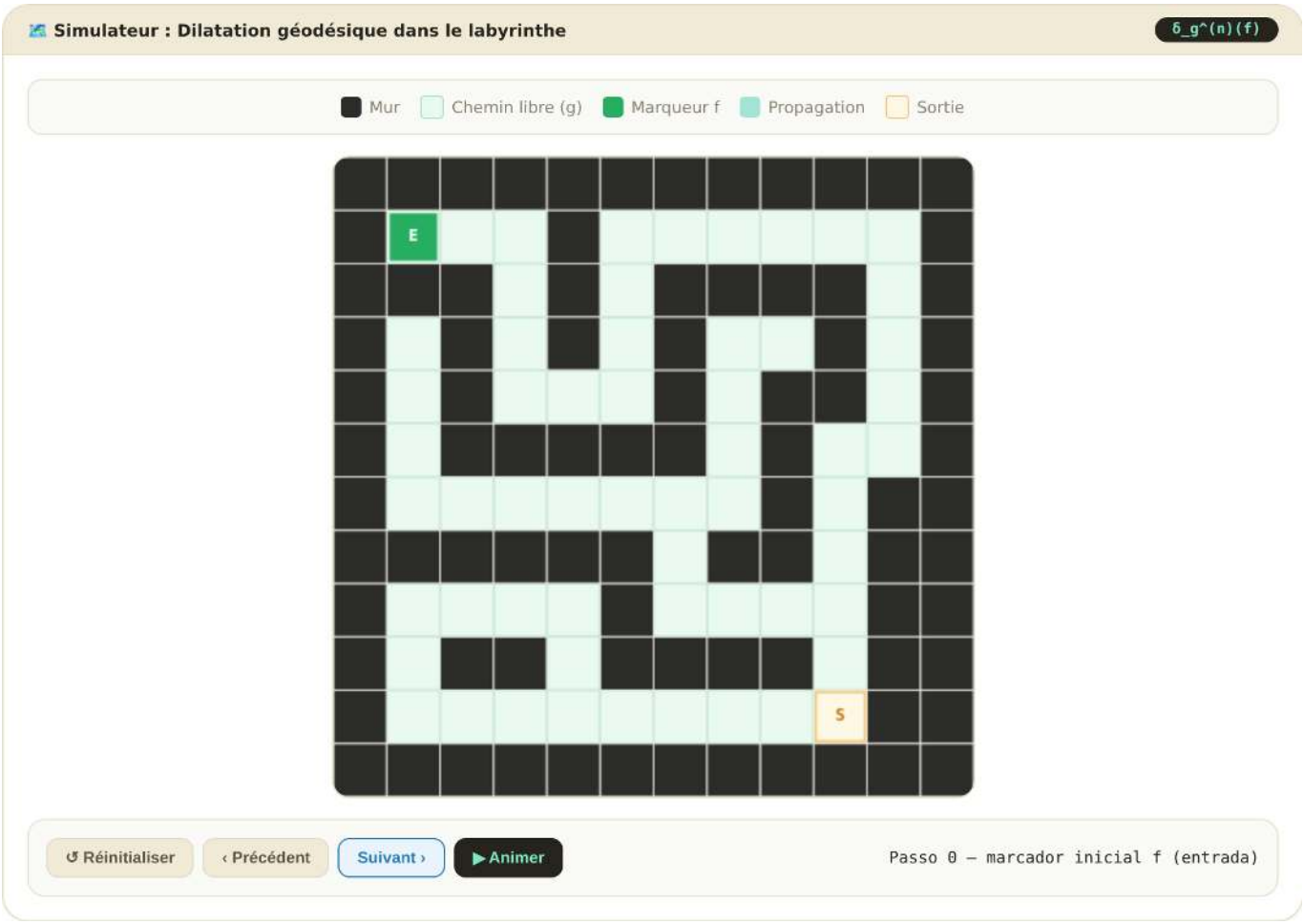
```
def cdil(f, g, b=np.zeros((3,3),dtype='uint8'), n=1):
    """Dilatação geodésica do marcador f sob a máscara g."""
    y = f.copy()
    for _ in range(n):
        y = np.minimum(mm.dil(y, b), g)
    return y
```

L'instruction `np.minimum(mm::dil(y, b), g)` implémente exactement la définition mathématique de la dilatation géodésique, c'est-à-dire $(y \oplus b) \wedge g$.

Lorsque $n = 1$, la fonction exécute une unique dilatation géodésique. Pour $n > 1$, le résultat de chaque étape devient le marqueur de l'étape suivante, produisant une propagation progressive contrôlée par le masque.

Le simulateur interactif Figure 4.7 permet de suivre, étape par étape, la propagation du marqueur f le long des couloirs du labyrinthe. À chaque itération de `mm::cdil`, le front de dilatation avance vers les cellules voisines libres — celles où $g = 1$ —, tandis que les murs ($g = 0$) restent infranchissables. Le nombre de

pas nécessaires pour que le marqueur atteigne la sortie correspond exactement à la longueur géodésique du chemin le plus court à l'intérieur du masque, mettant en évidence la connexion directe entre la dilatation géodésique itérée et la notion de distance dans les graphes.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.fr/cap04/sim-04-cdil.html

Figure 4.7: Simulateur interactif de dilatation géodésique : le marqueur f (vert) se propage pas à pas le long des chemins libres du masque g, sans traverser les murs.

4.3.3.3 Érosion géodésique

De manière duale, l'**érosion géodésique** d'une image marqueur f sous une image masque g est définie par :

$$f \ominus_g b = (f \ominus b) \vee g, \tag{4.8}$$

où $\vee$ représente l'opérateur de maximum point par point.

Dans ce cas, l'érosion conventionnelle du marqueur est suivie d'une contrainte inférieure imposée par le masque. Ainsi, aucun pixel du résultat ne peut prendre une valeur inférieure à celle du pixel correspondant du masque.

La formulation classique de l'érosion géodésique suppose la condition duale

$$f \geq g,$$

de sorte que le masque agisse comme une limite inférieure tout au long du processus.

4.3.3.4 Implémentation de l'érosion géodésique

La fonction statique `mm::cero` matérialise cet opérateur :

```
staticmethod
def cero(f, g, b=np.zeros((3,3),dtype='uint8'), n=1):
    """Érosion géodésique du marqueur f sous le masque g."""
    y = f.copy()
    for _ in range(n):
        y = np.maximum(mm.ero(y, b), g)
    return y
```

L'instruction `np.maximum(mm::ero(y, b), g)` implémente directement l'expression $(y \ominus b) \vee g$.

Tout comme pour la dilatation géodésique, le paramètre n définit combien d'érosions géodésiques successives seront calculées avant de renvoyer l'image finale.

> **i** Relation avec la reconstruction morphologique
>
> La reconstruction morphologique présentée dans la section suivante est obtenue par l'application itérative de la dilatation géodésique (`mm::cdil`) jusqu'à ce qu'un point fixe soit atteint, c'est-à-dire jusqu'à ce qu'aucune cellule ne change de valeur entre deux itérations consécutives. En d'autres termes, la reconstruction consiste en une séquence de dilatations géodésiques successives qui se propagent à l'intérieur du masque jusqu'à ce qu'il n'y ait plus de modifications.
> De manière duale, il est également possible de définir des reconstructions basées sur l'érosion géodésique au moyen d'applications successives de `mm::cero`.

4.3.3.5 Exemple : propagation dans un labyrinthe via dualité

La Figure 4.8 illustre la résolution du problème de connectivité d'un labyrinthe en utilisant la dualité morphologique au moyen des fonctions `mm::cero` et `mm::suprec`.

Au lieu de propager un marqueur à travers les couloirs libres par des dilatations géodésiques, le problème est formulé dans le domaine complémentaire. Initialement, le masque original est inversé,

$$g = 1 - g_{orig},$$

ce qui fait que les murs prennent la valeur 1 et les couloirs la valeur 0. De manière analogue, le marqueur est construit dans ce même domaine complémentaire, contenant une unique valeur 0 à la position de l'entrée du labyrinthe et la valeur 1 sur les autres pixels.

En utilisant l'élément structurant en croix (`mm::secross()`), l'érosion géodésique agit sur le marqueur complémenté. À chaque itération de `mm::cero`, la région connectée au marqueur initial subit des érosions successives, tandis que le masque impose une borne inférieure qui empêche la propagation à travers les murs du labyrinthe.

Les images intermédiaires montrent des états de l'évolution après différents nombres d'itérations (`n=5`, `n=12`, et `n=22`). Le résultat final est obtenu par la reconstruction géodésique par érosion (`mm::suprec`), qui applique des érosions géodésiques successives jusqu'à atteindre un point fixe, c'est-à-dire une situation où aucune modification supplémentaire ne se produit entre deux itérations consécutives.

Dans le domaine complémentaire, la région reconstruite correspond exactement à l'ensemble des couloirs connectés à l'entrée du labyrinthe. Ainsi, la connectivité entre l'entrée et la sortie peut être déterminée directement à partir de l'image reconstruite.

```
%%writefile tmp/fig_04_cero_labirinto.cpp
#define MM_OUT "tmp/fig_04_cero_labirinto.png"
#include "morph.hpp"
```

```
#include <opencv2/opencv.hpp>
#include <iostream>
#include <vector>
#include <string>
#include <filesystem>

int main() {
    // Máscara já invertida (255 = parede, 0 = corredor)
    mm::Image g(10, 10);
    int g_data[10][10] = {
        {255,  0,255,255,255,255,255,255,255,255},
        {255,  0,  0,  0,  0,  0,255,  0,  0,  0},
        {255,255,255,255,255,  0,255,  0,255,  0},
        {255,  0,  0,  0,255,  0,  0,  0,255,  0},
        {255,  0,255,  0,255,255,255,255,255,  0},
        {255,  0,255,  0,  0,  0,  0,  0,  0,  0},
        {255,  0,255,255,255,255,255,255,  0,255},
        {255,  0,  0,  0,  0,  0,  0,255,  0,255},
        {255,255,255,255,255,255,  0,  0,  0,255},
        {255,255,255,255,255,255,255,255,  0,255}
    };
    for (int y = 0; y < 10; y++)
        for (int x = 0; x < 10; x++)
            g.at(y, x) = (unsigned char)g_data[y][x];

    mm::Image f(10, 10);
    std::fill(f.data.begin(), f.data.end(), 255);
    f.at(0, 1) = 0;                       // semente (0) no corredor de entrada
    mm::Image B_cruz = mm::secross();

    mm::Image passo_5    = mm::cero(f, g, B_cruz, 5);
    mm::Image passo_12   = mm::cero(f, g, B_cruz, 12);
    mm::Image passo_22   = mm::cero(f, g, B_cruz, 22);
    mm::Image ponto_fixo = mm::suprec(f, g, B_cruz);

    mm::show(std::vector<mm::Image>{g, f, passo_5, passo_12, passo_22, ponto_fixo},
             MM_OUT,
             std::vector<std::string>{"Mascara (~g)", "Marcador (~f)", "n=5", "n=12", "n=22",
"~mm.suprec"},
             6);


// [pdi:panel-io] auto-generated - do not edit by hand
std::filesystem::create_directories("tmp");
mm::write(g, "tmp/fig_04_cero_labirinto_0.png");
mm::write(f, "tmp/fig_04_cero_labirinto_1.png");
mm::write(passo_5, "tmp/fig_04_cero_labirinto_2.png");
mm::write(passo_12, "tmp/fig_04_cero_labirinto_3.png");
mm::write(passo_22, "tmp/fig_04_cero_labirinto_4.png");
mm::write(ponto_fixo, "tmp/fig_04_cero_labirinto_5.png");
// [pdi:panel-io:end]
return 0;
}
```

```
Overwriting tmp/fig_04_cero_labirinto.cpp
```

```
!g++ -I. -std=c++17 -DMM_USE_OPENCV -I/usr/include/opencv4 tmp/fig_04_cero_labirinto.cpp -o
tmp/fig_04_cero_labirinto -lopencv_imgproc -lopencv_imgcodecs -lopencv_core \
  && ./tmp/fig_04_cero_labirinto \
  && test -f "tmp/fig_04_cero_labirinto.png" \
```

```
  || echo "  mm::show não gravou tmp/fig_04_cero_labirinto.png"
```

```
[1] Mascara (~g)
[2] Marcador (~f)
[3] n=5
[4] n=12
[5] n=22
[6] ~mm.suprec
```

```
try:
    mm.show(
        [
            mm.read("tmp/fig_04_cero_labirinto_0.png"),
            mm.read("tmp/fig_04_cero_labirinto_1.png"),
            mm.read("tmp/fig_04_cero_labirinto_2.png"),
            mm.read("tmp/fig_04_cero_labirinto_3.png"),
            mm.read("tmp/fig_04_cero_labirinto_4.png"),
            mm.read("tmp/fig_04_cero_labirinto_5.png"),
        ],
        titles=[
            'Mascara (~g)',
            'Marcador (~f)',
            'n=5',
            'n=12',
            'n=22',
            '~mm.suprec',
        ],
        cols=6,
    )
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + "
    tmp/fig_04_cero_labirinto_0.png (ver a versao Python)")
```

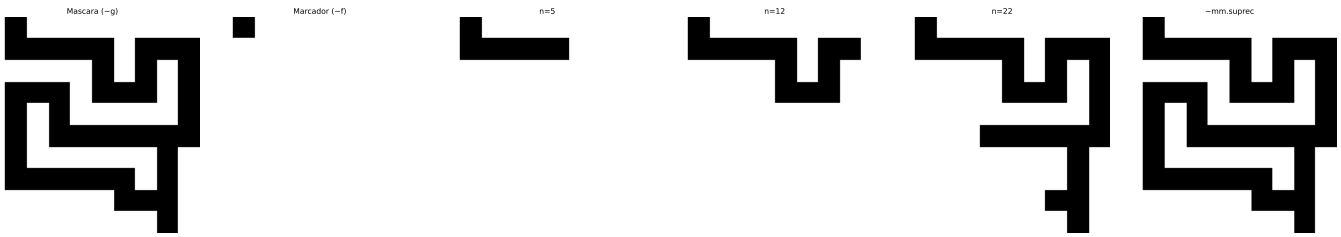


Figure 4.8: Resolução de labirinto no domínio complementar: com máscara e marcador invertidos, *mm::cero* e *mm::suprec* propagam a onda geodésica pelos corredores.

4.3.4 Reconstruction morphologique

La **reconstruction morphologique** propage une image **marqueur** f à l'intérieur d'une image **masque** g, en garantissant que le résultat ne dépasse jamais les valeurs d'intensité imposées par le masque. L'opérateur fondamental qui rend possible cette propagation contenue est la **dilatation géodésique**, définie par la Équation 4.7.

La reconstruction est obtenue par l'application itérative de cette dilatation conditionnée. Initialement, le marqueur est limité par le masque pour établir l'état initial :

$$X^{(0)} = f \wedge g,$$

et les itérations suivantes sont définies de manière récursive par :

$$X^{(k)} = (X^{(k-1)} \oplus b) \wedge g.$$

La séquence croît de manière monotone jusqu'à atteindre un point fixe, produisant la **reconstruction morphologique par dilatation** (également connue dans la littérature sous le nom d'*inf-reconstruction*) :

$$R_g^{\delta}(f) = \lim_{k \to \infty} X^{(k)} = X^{(k)} \quad \text{quand} \quad X^{(k)} = X^{(k-1)}. \tag{4.9}$$

La montée itérative est interrompue dès que la stabilité est atteinte, c'est-à-dire lorsque deux itérations consécutives produisent des matrices avec des valeurs absolument identiques.

4.3.4.1 Implémentation de la reconstruction morphologique

La routine didactique `mm::infrec` implémente directement l'algorithme itératif de point fixe. Initialement, le marqueur effectif initial $X^{(0)}$ est déterminé par l'opération `np.minimum(f, g)`. Pour garantir que la boucle de vérification exécute la première itération sans déclencher de fausses convergences prématurées, la variable de contrôle de l'itération précédente (`y1`) est initialisée avec une valeur sentinelle hors du domaine des données (ou simplement avec une matrice qui force la première exécution).

```
def infrec(f, g, b=np.zeros((3,3), dtype='uint8')):
    """Inf-reconstruction : dilate le marqueur (f   g) jusqu'à convergence sous le masque
    g."""
    y = np.minimum(f, g)
    # Initialise y1 avec des valeurs impossibles pour forcer l'entrée dans la boucle
    y1 = np.full_like(f, 256, dtype=np.int16)
    while not np.array_equal(y, y1):
        y1 = y.copy()
        # Applique la dilatation géodésique : (y   b)   g
        y = np.minimum(mm.dil(y, b), g)
    return y.astype('uint8')
```

À l'intérieur de la boucle `while`, la variable `y` stocke l'estimation courante de la reconstruction $X^{(k)}$, tandis que `y1` préserve l'image de l'étape immédiatement antérieure $X^{(k-1)}$. L'instruction de contrôle conditionnel `np.minimum(mm::dil(y, b), g)` traduit fidèlement la dilatation géodésique théorique, où l'expansion morphologique conventionnelle pilotée par OpenCV est immédiatement « élaguée » et limitée par les barrières d'intensité du masque g. La boucle cesse lorsqu'aucune modification de pixel n'est enregistrée entre les étapes.

4.3.4.2 Avantages de la reconstruction morphologique

La reconstruction morphologique est significativement plus robuste que l'ouverture conventionnelle, car elle est capable de supprimer les structures indésirables sans déformer ou modifier la morphologie des objets qui doivent être préservés.

Alors que l'ouverture classique adoucit les angles, élimine les pointes et déforme les contours en raison de l'imposition géométrique rigide de l'élément structurant, la reconstruction géodésique utilise le masque pour retrouver avec exactitude les limites et les formes originales des objets qui présentent une connectivité avec le marqueur initial.

En termes intuitifs, le marqueur agit comme une graine de contagion qui se propage progressivement, mais uniquement en traversant les régions autorisées par le masque. Les composantes qui n'ont aucune intersection avec le marqueur ne seront jamais reconstruites (étant ainsi éliminées), tandis que les composantes touchées par la graine s'étendent jusqu'à restaurer intégralement leur géométrie d'origine.

Ce comportement discriminatoire et conservateur est illustré à la Figure 4.9, en utilisant l'élément structurant en croix présenté à la Figure 4.3..

```
%%writefile tmp/fig_04_reconstrucao_didatica.cpp
#define MM_OUT "tmp/fig_04_reconstrucao_didatica.png"
//| label: fig-04-reconstrucao-didatica
//| fig-cap: "*Pipeline* de Reconstrução Morfológica por Dilatação Condicionada: a máscara
contém dois objetos, o marcador isola apenas o núcleo do objeto principal, e as iterações
reconstroem sua forma exata até a convergência."
//| echo: true
//| output: true
#include "morph.hpp"
#include <iostream>
#include <vector>
#include <string>

int main() {
    mm::Image g(10, 10);
    int g_data[10][10] = {
        {0,0,0,0,0,0,0,0,0,0},
        {0,1,1,1,0,0,0,1,1,0},
        {0,1,1,1,1,0,0,1,1,0},
        {0,1,1,1,1,0,0,0,0,0},
        {0,1,1,1,1,0,0,0,0,0},
        {0,1,1,1,0,0,0,0,0,0},
        {0,0,1,1,1,0,0,1,0,0},
        {0,0,1,1,1,0,0,1,1,0},
        {0,0,0,1,0,0,0,0,0,0},
        {0,0,0,0,0,0,0,0,0,0}
    };
    for (int y = 0; y < 10; ++y)
        for (int x = 0; x < 10; ++x)
            g.at(y, x) = g_data[y][x] * 255;

    mm::Image B_cruz = mm::secross();
    mm::Image f = mm::ero(g, mm::sebox());          // marcador: núcleo do objeto principal

    std::vector<mm::Image> iteracoes;
    std::vector<std::string> titulos;
    mm::Image img_atual = f;
    for (int i = 1; i <= 5; ++i) {
        img_atual = mm::cdil(img_atual, g, B_cruz);
        iteracoes.push_back(img_atual);
        titulos.push_back("Dilatação Cond. (n=" + std::to_string(i) + ")");
    }

    mm::Image img_reconstruida = mm::infrec(f, g, B_cruz);
    bool estavel = true;
    for (int i = 0; i < img_reconstruida.h * img_reconstruida.w * img_reconstruida.channels;
    ++i) {
        if (img_reconstruida.data[i] != iteracoes.back().data[i]) {
            estavel = false;
            break;
        }
    }
    std::cout << "  Estabilidade na iteração 5: " << (estavel ? "True" : "False") << "\n";

    std::vector<mm::Image> imgs = {g, f};
    imgs.insert(imgs.end(), iteracoes.begin(), iteracoes.end());
    imgs.push_back(img_reconstruida);

    std::vector<std::string> titles = {"Máscara (g)", "Marcador (f)"};
    titles.insert(titles.end(), titulos.begin(), titulos.end());
    titles.push_back("Reconstrução R_g(f)");
```

```
    mm::show(imgs, MM_OUT, titles, 8);
    return 0;
}
```

```
Overwriting tmp/fig_04_reconstrucao_didatica.cpp
```

```
!g++ -I. -std=c++17 -DMM_USE_OPENCV -I/usr/include/opencv4
tmp/fig_04_reconstrucao_didatica.cpp -o tmp/fig_04_reconstrucao_didatica -lopencv_imgproc
-lopencv_imgcodecs -lopencv_core \
  && ./tmp/fig_04_reconstrucao_didatica \
  && test -f "tmp/fig_04_reconstrucao_didatica.png" \
  || echo "  mm::show não gravou tmp/fig_04_reconstrucao_didatica.png"
```

```
 Estabilidade na iteração 5: True
[1] Máscara (g)
[2] Marcador (f)
[3] Dilatação Cond. (n=1)
[4] Dilatação Cond. (n=2)
[5] Dilatação Cond. (n=3)
[6] Dilatação Cond. (n=4)
[7] Dilatação Cond. (n=5)
[8] Reconstrução R_g(f)
```

```
try:
    mm.show(mm.read("tmp/fig_04_reconstrucao_didatica.png"), figsize=(18, 3))
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + "
    tmp/fig_04_reconstrucao_didatica.png (ver a versao Python)")
```

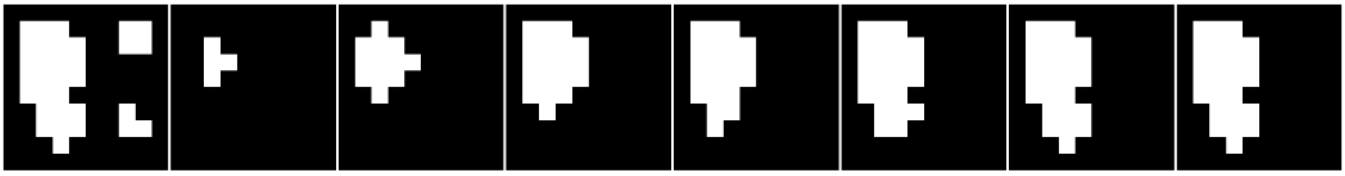


Figure 4.9: *Pipeline* de Reconstrução Morfológica por Dilatação Condicionada: a máscara contém dois objetos, o marcador isola apenas o núcleo do objeto principal, e as iterações reconstroem sua forma exata até a convergência.

4.3.5 Remplissage des trous et suppression des bords

Deux opérateurs basés sur la reconstruction morphologique complètent le *pipeline* de nettoyage binaire. Leurs principales caractéristiques sont résumées dans la Table 4.3.

Remplissage des trous (`mm::clohole`) supprime les cavités entièrement entourées par l'objet, quelle que soit leur taille, sans modifier les contours externes. La procédure agit sur le complément de l'image, en utilisant comme marqueur une restriction du cadre (*frame*) au fond :

$$\mathrm{clohole}(f) = \left(R^{\delta}_{f^c}(\mathrm{frame}(f) \wedge f^c)\right)^c \tag{4.10}$$

Sur le plan opérationnel, on reconstruit d'abord le fond externe, puis on applique la complémentation pour récupérer les objets dont les trous ont été remplis.

Suppression des objets de bord (`mm::edgeoff`) élimine tous les objets qui touchent le bord de l'image, en ne préservant que les composantes entièrement internes. Le marqueur est obtenu par l'intersection entre le cadre (*frame*) et les objets de l'image :

$$\text{edgeoff}(f) = f \setminus R_f^{\delta}(\text{frame}(f) \wedge f) \tag{4.11}$$

Tableau 4.3: Comparaison entre les opérateurs *clohole* et *edgeoff*.

Opérateur	Marqueur	Masque	Effet
`mm::clohole`	*frame* restreint au fond (f^c)	f^c	Remplit les trous internes
`mm::edgeoff`	*frame* restreint à l'objet (f)	f	Supprime les objets connectés au bord

L'évolution pas à pas de ces transformations géodésiques peut être suivie dans les figures suivantes. La Figure 4.10 illustre le mécanisme d'inondation contrôlée de l'opérateur `mm::clohole`, dans lequel la reconstruction se fait à partir du fond externe et empêche la propagation vers les régions internes non connectées à l'extérieur, ce qui aboutit au remplissage cohérent des cavités internes. En revanche, la Figure 4.11 détaille la dynamique de l'opérateur `mm::edgeoff`, où seules les composantes connectées au bord sont reconstruites puis supprimées, ne préservant que les objets entièrement contenus à l'intérieur de l'image.

La connectivité de la propagation géodésique est contrôlée par l'élément structurant : `mm::sebox()` (voisinage de 8) inclut les connexions diagonales, tandis que `mm::secross()` (voisinage de 4) les exclut. Par conséquent, le choix de l'élément structurant affecte les composantes atteintes par la reconstruction et, donc, celles qui seront préservées ou supprimées.

4.3.5.1 Conformité avec l'implémentation

Les définitions ci-dessus sont directement alignées avec l'implémentation dans `morph.py`, reproduite ci-dessous :

```
staticmethod
def clohole(f, b=np.ones((3,3),dtype='uint8')):
    # marqueur restreint au fond de l'image
    marqueur = mm.frame(f, border=1) & mm.neg(f)
    return mm.neg(mm.infrec(marqueur, mm.neg(f), b))

staticmethod
def edgeoff(f, b=np.ones((3,3),dtype='uint8')):
    # marqueur restreint aux objets de l'image
    marqueur = mm.frame(f, border=1) & f
    return mm.subm(f, mm.infrec(marqueur, f, b))
```

Ces implémentations rendent explicite le fait que les deux opérateurs sont des instances directes de la reconstruction morphologique par dilatation géodésique avec `mm::infrec`, ne différant que par le choix du marqueur et du masque : `clohole` agit sur le complément de l'image, tandis que `edgeoff` agit directement sur le domaine des objets.

```
%%writefile tmp/fig_04_clohole_didatico.cpp
#define MM_OUT "tmp/fig_04_clohole_didatico.png"
//| label: fig-04-clohole-didatico
//| fig-cap: "*Pipeline* de preenchimento de buracos (*clohole*): o marcador vem da borda da
imagem, restrito ao complemento $f^c$. A dilatação geodésica reconstrói o fundo externo; após
a complementação, os buracos internos ficam preenchidos."
//| echo: true
//| output: true

#include "morph.hpp"
#include <iostream>
```

```cpp
#include <vector>
#include <string>

int main() {
    mm::Image f(10, 10);
    // Initialize f with the given pattern
    int pattern[10][10] = {
        {0,0,0,0,0,0,0,0,0,0},
        {0,1,1,1,1,0,0,0,0,1},
        {0,1,0,0,1,0,0,1,0,1},
        {0,1,0,0,1,0,0,1,0,1},
        {0,1,1,1,1,0,0,1,0,0},
        {0,0,0,0,0,0,0,0,0,0},
        {0,0,1,1,1,0,1,1,1,0},
        {0,0,1,1,1,0,1,0,1,0},
        {0,0,1,1,1,0,1,1,1,0},
        {0,0,0,0,0,0,0,0,0,1}
    };
    for (int y = 0; y < 10; y++)
        for (int x = 0; x < 10; x++)
            f.at(y, x) = pattern[y][x] * 255;

    mm::Image B_cruz = mm::secross();
    mm::Image f_c = mm::neg(f);
    mm::Image marcador_ch = mm::frame(f, 1);       // borda externa

    std::vector<mm::Image> iteracoes;
    std::vector<std::string> titulos;
    mm::Image img_atual = marcador_ch;
    for (int i = 1; i <= 5; i++) {
        img_atual = mm::cdil(img_atual, f_c, B_cruz);
        iteracoes.push_back(img_atual);
        titulos.push_back("Iter. (n=" + std::to_string(i) + ")");
    }

    mm::Image img_clohole = mm::neg(mm::infrec(marcador_ch, f_c, B_cruz));

    // Validate clohole
    mm::Image expected = mm::clohole(f);
    bool valid = true;
    for (int i = 0; i < img_clohole.h * img_clohole.w * img_clohole.channels; i++) {
        if (img_clohole.data[i] != expected.data[i]) {
            valid = false;
            break;
        }
    }
    std::cout << "  Validação clohole: " << (valid ? "true" : "false") << std::endl;

    std::vector<mm::Image> display_imgs = {f, marcador_ch};
    display_imgs.insert(display_imgs.end(), iteracoes.begin(), iteracoes.end());
    display_imgs.push_back(img_clohole);

    std::vector<std::string> display_titles = {"f original", "Marcador (borda)"};
    display_titles.insert(display_titles.end(), titulos.begin(), titulos.end());
    display_titles.push_back("clohole(f)");

    mm::show(display_imgs, MM_OUT, display_titles, 8);

    return 0;
}
```

```
Overwriting tmp/fig_04_clohole_didatico.cpp
```

```
!g++ -I. -std=c++17 -DMM_USE_OPENCV -I/usr/include/opencv4 tmp/fig_04_clohole_didatico.cpp -o
tmp/fig_04_clohole_didatico -lopencv_imgproc -lopencv_imgcodecs -lopencv_core \
  && ./tmp/fig_04_clohole_didatico \
  && test -f "tmp/fig_04_clohole_didatico.png" \
  || echo "  mm::show não gravou tmp/fig_04_clohole_didatico.png"
```

```
 Validação clohole: true
[1] f original
[2] Marcador (borda)
[3] Iter. (n=1)
[4] Iter. (n=2)
[5] Iter. (n=3)
[6] Iter. (n=4)
[7] Iter. (n=5)
[8] clohole(f)
```

```
try:
    mm.show(mm.read("tmp/fig_04_clohole_didatico.png"), figsize=(18, 3))
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + "
    tmp/fig_04_clohole_didatico.png (ver a versao Python)")
```

Figure 4.10: *Pipeline* de preenchimento de buracos (*clohole*): o marcador vem da borda da imagem, restrito ao complemento f^c. A dilatação geodésica reconstrói o fundo externo; após a complementação, os buracos internos ficam preenchidos.

```
%%writefile tmp/fig_04_edgeoff_didatico.cpp
#define MM_OUT "tmp/fig_04_edgeoff_didatico.png"
//| label: fig-04-edgeoff-didatico
//| fig-cap: "*Pipeline* de eliminação de estruturas de borda (*edgeoff*): o marcador captura
as raízes conectadas às extremidades, a reconstrução delimita esses elementos e a subtração
preserva só os objetos totalmente internos."
//| echo: true
//| output: true

#include "morph.hpp"
#include <iostream>
#include <vector>
#include <string>

int main() {
    mm::Image f(10, 10);
    std::vector<std::vector<int>> f_data = {
        {0,0,0,0,0,0,0,0,0,0},
        {0,1,1,1,1,0,0,0,0,1},
        {0,1,0,0,1,0,0,1,0,1},
        {0,1,0,0,1,0,0,1,0,1},
        {0,1,1,1,1,0,0,1,0,0},
        {0,0,0,0,0,0,0,0,0,0},
```

```
        {0,0,1,1,1,0,1,1,1,0},
        {0,0,1,1,1,0,1,0,1,0},
        {0,0,1,1,1,0,1,1,1,0},
        {0,0,0,0,0,0,0,0,0,1}
    };
    for (int y = 0; y < 10; y++)
        for (int x = 0; x < 10; x++)
            f.at(y, x) = f_data[y][x] * 255;

    mm::Image B_box = mm::sebox();
    mm::Image marcador_eo = mm::band(mm::frame(f, 1), f);   // borda   f

    std::vector<mm::Image> iteracoes;
    std::vector<std::string> titulos;
    mm::Image img_atual = marcador_eo;
    for (int i = 1; i <= 5; i++) {
        img_atual = mm::cdil(img_atual, f, B_box);
        iteracoes.push_back(img_atual);
        titulos.push_back("Iter. (n=" + std::to_string(i) + ")");
    }

    mm::Image img_edgeoff = mm::edgeoff(f, B_box);

    std::vector<mm::Image> all_imgs = {f, marcador_eo};
    all_imgs.insert(all_imgs.end(), iteracoes.begin(), iteracoes.end());
    all_imgs.push_back(img_edgeoff);

    std::vector<std::string> all_titles = {"f original", "Marcador (borda)"};
    all_titles.insert(all_titles.end(), titulos.begin(), titulos.end());
    all_titles.push_back("edgeoff(f)");

    mm::show(all_imgs, MM_OUT, all_titles, 8);

    return 0;
}
```

```
Overwriting tmp/fig_04_edgeoff_didatico.cpp
```

```
!g++ -I. -std=c++17 -DMM_USE_OPENCV -I/usr/include/opencv4 tmp/fig_04_edgeoff_didatico.cpp -o
tmp/fig_04_edgeoff_didatico -lopencv_imgproc -lopencv_imgcodecs -lopencv_core \
  && ./tmp/fig_04_edgeoff_didatico \
  && test -f "tmp/fig_04_edgeoff_didatico.png" \
  || echo "  mm::show não gravou tmp/fig_04_edgeoff_didatico.png"
```

```
[1] f original
[2] Marcador (borda)
[3] Iter. (n=1)
[4] Iter. (n=2)
[5] Iter. (n=3)
[6] Iter. (n=4)
[7] Iter. (n=5)
[8] edgeoff(f)
```

```
try:
    mm.show(mm.read("tmp/fig_04_edgeoff_didatico.png"), figsize=(18, 3))
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + "
    tmp/fig_04_edgeoff_didatico.png (ver a versao Python)")
```

Figure 4.11: *Pipeline* de eliminação de estruturas de borda (*edgeoff*): o marcador captura as raízes conectadas às extremidades, a reconstrução delimita esses elementos e a subtração preserva só os objetos totalmente internos.

4.3.6 Pipeline de Limpeza Binária com CLAHE

Com base na análise anterior — na qual o CLAHE produziu o maior valor da variância interclasses ($\sigma_B^2 \approx 2{,}47 \times 10^3$), ver Figure 4.2, e o operador `mm::clohole` mostrou-se eficaz no preenchimento das cavidades internas —, o *pipeline* final de segmentação, ilustrado na Figure 4.12, é estruturado pelo seguinte fluxo computacional:

$$\text{gray} \xrightarrow{\text{CLAHE}} \xrightarrow{\text{Otsu}} \xrightarrow{\text{open}} \xrightarrow{\text{clohole}} \xrightarrow{\text{open}} \xrightarrow{\text{edgeoff}} \text{segmentação}$$

Após a etapa de `mm::clohole`, aplica-se uma segunda abertura morfológica com um elemento estruturante maior (`mm::sedisk(33)`, disco de diâmetro 33). Essa operação remove pequenas regiões residuais e artefatos que possam ter permanecido após a segmentação. Em particular, o preenchimento geodésico pode transformar pequenas cavidades isoladas em componentes conectados ao objeto, tornando conveniente uma etapa adicional de filtragem baseada em tamanho. O diâmetro foi escolhido de modo que as moedas permaneçam capazes de conter o elemento estruturante, enquanto componentes significativamente menores sejam eliminados.

Nesta imagem, nenhuma moeda está conectada à borda da matriz. Consequentemente, a aplicação de `mm::edgeoff` não altera o resultado obtido após a segunda abertura. Ainda assim, essa etapa é mantida no *pipeline* por robustez, pois em outras imagens podem existir objetos parcialmente visíveis ou conectados às bordas, que devem ser removidos antes da etapa de análise.

> Pourquoi l'ouverture après le clohole ?
>
> L'opérateur `mm::clohole` remplit toutes les cavités fermées présentes dans les objets segmentés. Dans certaines situations, de petites régions indésirables peuvent subsister après cette étape ou devenir connectées aux objets principaux. L'ouverture morphologique subséquente supprime les composants plus petits que l'élément structurant, tout en préservant les pièces en raison de leur taille significativement plus grande.

```
%%writefile tmp/fig_04_pipeline_clahe.cpp
#define MM_OUT "tmp/fig_04_pipeline_clahe.png"
//| label: fig-04-pipeline-clahe
//| fig-cap: "*Pipeline* completo de segmentação com CLAHE: Otsu → abertura (r=9) → clohole →
abertura (r=33) → edgeoff."
//| echo: true
//| output: true

#include "morph.hpp"
#include <iostream>
#include <vector>
#include <string>
#include <filesystem>

int main() {
// [pdi:state-io] auto-generated - do not edit by hand
```

```
mm::Image img_coins_gray = mm::_read_state("tmp/state/img_coins_gray_8.png");
// [pdi:state-io:end]

    //   Pré-processamento: apenas CLAHE
    mm::Image img_clahe0 = mm::clahe(img_coins_gray, 2.0, 8);

    //   Etapa 1: Binarização Otsu
    mm::Image img_bin = mm::threshold(img_clahe0);

    //   Etapa 2: Abertura - remove ruídos brancos de fundo
    mm::Image img_open = mm::open(img_bin, mm::sedisk(9));

    //   Etapa 3: clohole - fecha todos os buracos internos
    mm::Image img_hole = mm::clohole(img_open);

    //   Etapa 4: Abertura (kernel grande) - remove artefatos do clohole
    mm::Image img_limpo = mm::open(img_hole, mm::sedisk(33));

    //   Etapa 5: edgeoff - remove objetos que tocam a borda
    mm::Image img_final = mm::edgeoff(img_limpo, mm::SE::box(3), 1);

    mm::show(
        std::vector<mm::Image>{img_clahe0, img_bin, img_open, img_hole, img_limpo, img_final},
        MM_OUT,
        std::vector<std::string>{"CLAHE", "Otsu", "Abertura (r=9)",
                                 "clohole", "Abertura (r=33)", "Final (edgeoff)"},
        6
    );


// [pdi:state-io] auto-generated - do not edit by hand
std::filesystem::create_directories("tmp/state");
mm::write(img_final, "tmp/state/img_final_55.png");
// [pdi:state-io:end]

// [pdi:panel-io] auto-generated - do not edit by hand
std::filesystem::create_directories("tmp");
mm::write(img_clahe0, "tmp/fig_04_pipeline_clahe_0.png");
mm::write(img_bin, "tmp/fig_04_pipeline_clahe_1.png");
mm::write(img_open, "tmp/fig_04_pipeline_clahe_2.png");
mm::write(img_hole, "tmp/fig_04_pipeline_clahe_3.png");
mm::write(img_limpo, "tmp/fig_04_pipeline_clahe_4.png");
mm::write(img_final, "tmp/fig_04_pipeline_clahe_5.png");
// [pdi:panel-io:end]
return 0;
}
```

```
Overwriting tmp/fig_04_pipeline_clahe.cpp
```

```
!g++ -I. -std=c++17 -DMM_USE_OPENCV -I/usr/include/opencv4 tmp/fig_04_pipeline_clahe.cpp -o
tmp/fig_04_pipeline_clahe -lopencv_imgproc -lopencv_imgcodecs -lopencv_core \
  && ./tmp/fig_04_pipeline_clahe \
  && test -f "tmp/fig_04_pipeline_clahe.png" \
  || echo "  mm::show não gravou tmp/fig_04_pipeline_clahe.png"
```

```
[1] CLAHE
[2] Otsu
[3] Abertura (r=9)
[4] clohole
[5] Abertura (r=33)
```

```
[6] Final (edgeoff)
```

```python
try:
    mm.show(
        [
            mm.read("tmp/fig_04_pipeline_clahe_0.png"),
            mm.read("tmp/fig_04_pipeline_clahe_1.png"),
            mm.read("tmp/fig_04_pipeline_clahe_2.png"),
            mm.read("tmp/fig_04_pipeline_clahe_3.png"),
            mm.read("tmp/fig_04_pipeline_clahe_4.png"),
            mm.read("tmp/fig_04_pipeline_clahe_5.png"),
        ],
        titles=[
            'CLAHE',
            'Otsu',
            'Abertura (r=9)',
            'clohole',
            'Abertura (r=33)',
            'Final (edgeoff)',
        ],
        cols=6,
        figsize=(18, 6),
    )
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + "
    tmp/fig_04_pipeline_clahe_0.png (ver a versao Python)")
```

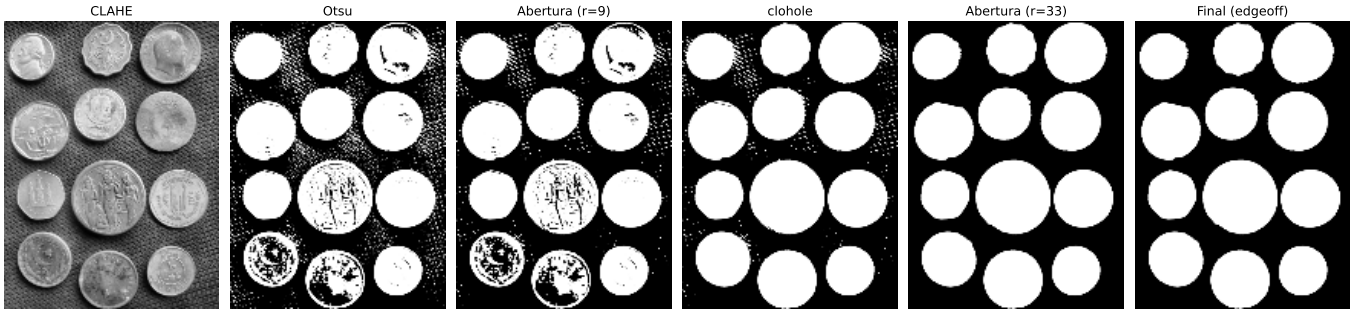


Figure 4.12: *Pipeline* completo de segmentação com CLAHE: Otsu → abertura (r=9) → clohole → abertura (r=33) → edgeoff.

4.3.7 Morphologie en niveaux de gris

Les opérateurs morphologiques s'étendent naturellement aux images en niveaux de gris. Dans cette formulation, l'érosion et la dilatation agissent directement sur les niveaux d'intensité de l'image. Pour des éléments structurants plats ($b \equiv 0$), l'érosion correspond au **minimum local** et la dilatation au **maximum local** dans le voisinage défini par l'élément structurant.

L'interprétation intuitive est simple : l'érosion **assombrit** les régions en remplaçant chaque pixel par la plus petite valeur présente dans son voisinage, tandis que la dilatation **éclaircit** les régions en utilisant la plus grande valeur disponible. La combinaison de ces opérateurs permet de construire des transformations capables de rehausser les contours, de supprimer les tendances d'éclairage et de mettre en évidence les structures locales.

Trois opérateurs dérivés sont particulièrement utiles :

Gradient morphologique — met en évidence les contours comme la différence entre la dilatation et l'érosion :

$$\text{grad}_B(f) = (f \oplus B) - (f \ominus B) \tag{4.12}$$

Top-hat — met en évidence les structures brillantes plus petites que l'élément structurant (différence entre l'image originale et son ouverture) :

$$\text{top-hat}_B(f) = f - (f \circ B) \tag{4.13}$$

Black-hat — met en évidence les structures sombres plus petites que l'élément structurant (différence entre la fermeture et l'image originale) :

$$\text{black-hat}_B(f) = (f \bullet B) - f \tag{4.14}$$

Le *Top-hat* extrait les détails brillants qui ne survivent pas à l'ouverture, tandis que le *Black-hat* révèle les détails sombres supprimés par la fermeture. Quant au gradient morphologique, il rehausse les transitions abruptes d'intensité, produisant une représentation similaire à celle d'un détecteur de contours.

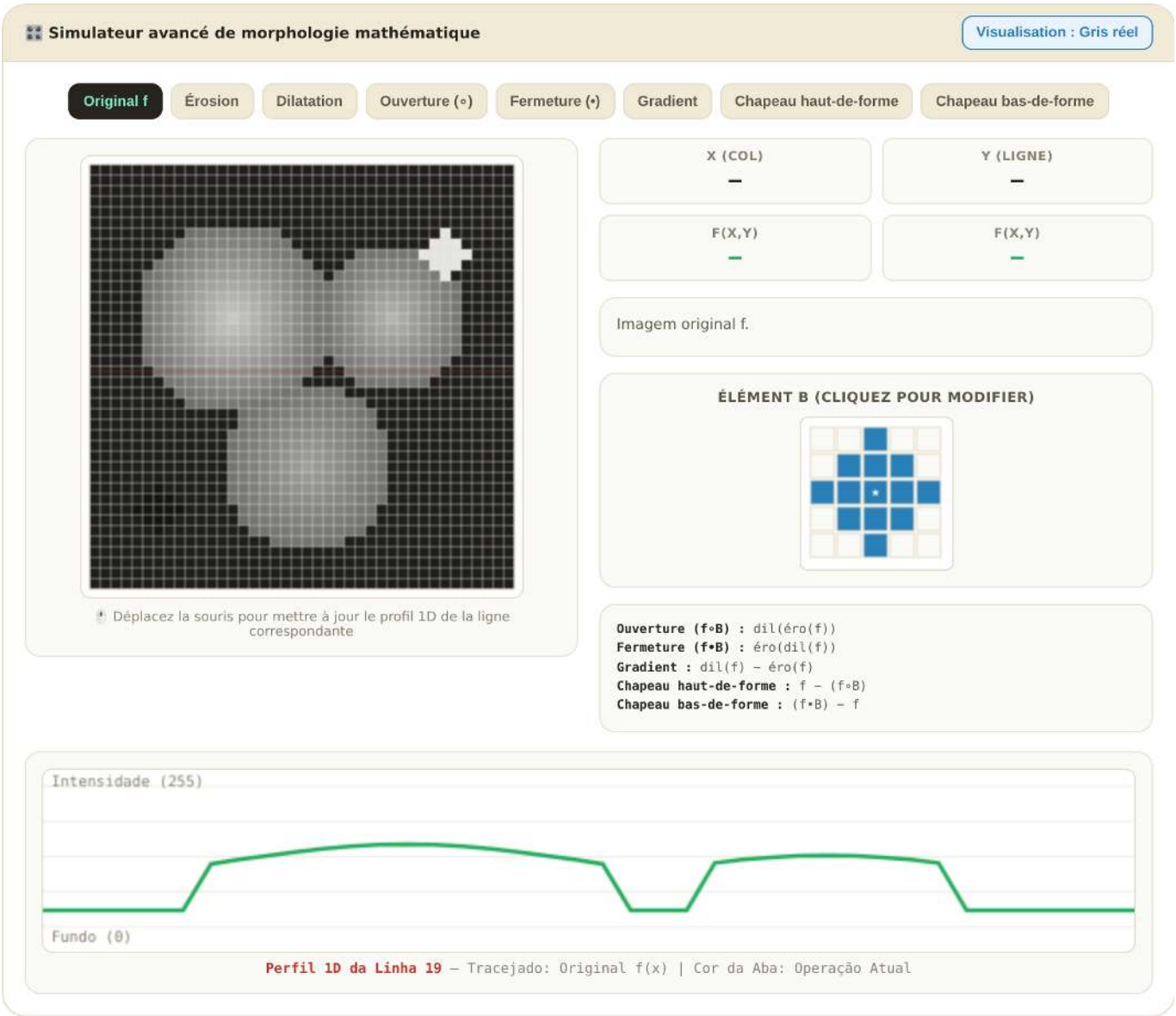
Pour comprendre le mécanisme de ces opérateurs au niveau local, la Figure 4.13 présente un simulateur interactif de morphologie en tons de gris. Le simulateur permet de modifier librement l'élément structurant, de visualiser son déplacement sur l'image et de suivre simultanément le profil unidimensionnel des intensités. Ainsi, il devient possible d'observer directement comment l'érosion sélectionne les minima locaux, comment la dilatation sélectionne les maxima locaux et comment le gradient morphologique émerge de la différence entre ces deux opérateurs.

Le bouton situé dans le coin supérieur droit permet d'alterner entre la visualisation originale en tons de gris et une représentation en fausses couleurs (*colormap*) pour les trois types de gradients uniquement. La version colorée facilite la perception visuelle des variations d'intensité, rendant plus évidente l'action des opérateurs morphologiques sur les maxima, les minima et les transitions locales de l'image.

Les opérateurs présentés sont disponibles dans `morph.py` via les fonctions `mm::gradm`, `mm::tophat` et `mm::blackhat`.

La Figure 4.14 illustre les effets de ces opérateurs sur l'image des pièces et leurs histogrammes respectifs. On observe que l'érosion déplace la distribution vers des intensités plus faibles, tandis que la dilatation la déplace vers des intensités plus élevées. Le gradient concentre les valeurs dans les régions de contour, et les opérateurs *Top-hat* et *Black-hat* produisent des histogrammes fortement concentrés à de faibles niveaux d'intensité, car seules de petites structures locales sont mises en évidence.

```
%%writefile tmp/fig_04_morf_gc_histogramas.cpp
#define MM_OUT "tmp/fig_04_morf_gc_histogramas.png"
//| label: fig-04-morf-gc-histogramas
//| fig-cap: "Morfologia em tons de cinza e seus histogramas: erosão, dilatação, gradiente,
top-hat e black-hat. Elemento estruturante: disco de diâmetro 19."
//| echo: true
//| output: true

#include "morph.hpp"
#include <iostream>
#include <vector>
#include <string>

int main() {
// [pdi:state-io] auto-generated - do not edit by hand
mm::Image img_coins_gray = mm::_read_state("tmp/state/img_coins_gray_8.png");
// [pdi:state-io:end]

    mm::Image B = mm::sedisk(19);
```



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.fr/cap04/sim-04-operadores-av.html

Figure 4.13: Simulateur interactif avancé de morphologie avec élément structurant modifiable et profil 1D.

```
    mm::Image img_ero  = mm::ero(img_coins_gray, B);
    mm::Image img_dil  = mm::dil(img_coins_gray, B);
    mm::Image img_grad = mm::gradm(img_coins_gray, B);
    mm::Image img_th   = mm::tophat(img_coins_gray, B);
    mm::Image img_bh   = mm::blackhat(img_coins_gray, B);

    mm::show(
        std::vector<mm::Image>{img_coins_gray, mm::histImg(img_coins_gray), img_ero,
mm::histImg(img_ero),
                               img_dil, mm::histImg(img_dil), img_grad, mm::histImg(img_grad),
                               img_th, mm::histImg(img_th), img_bh, mm::histImg(img_bh)},
        MM_OUT,
        std::vector<std::string>{"Original", "Hist", "Erosão", "Hist", "Dilatação", "Hist",
                                 "Gradiente", "Hist", "Top-hat", "Hist", "Black-hat", "Hist"},
        4
    );

    return 0;
}
```

```
Overwriting tmp/fig_04_morf_gc_histogramas.cpp
```

```
!g++ -I. -std=c++17 -DMM_USE_OPENCV -I/usr/include/opencv4 tmp/fig_04_morf_gc_histogramas.cpp
-o tmp/fig_04_morf_gc_histogramas -lopencv_imgproc -lopencv_imgcodecs -lopencv_core \
  && ./tmp/fig_04_morf_gc_histogramas \
  && test -f "tmp/fig_04_morf_gc_histogramas.png" \
  || echo "  mm::show não gravou tmp/fig_04_morf_gc_histogramas.png"
```

```
[1] Original
[2] Hist
[3] Erosão
[4] Hist
[5] Dilatação
[6] Hist
[7] Gradiente
[8] Hist
[9] Top-hat
[10] Hist
[11] Black-hat
[12] Hist
```

```
try:
    mm.show(mm.read("tmp/fig_04_morf_gc_histogramas.png"))
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + "
    tmp/fig_04_morf_gc_histogramas.png (ver a versao Python)")
```

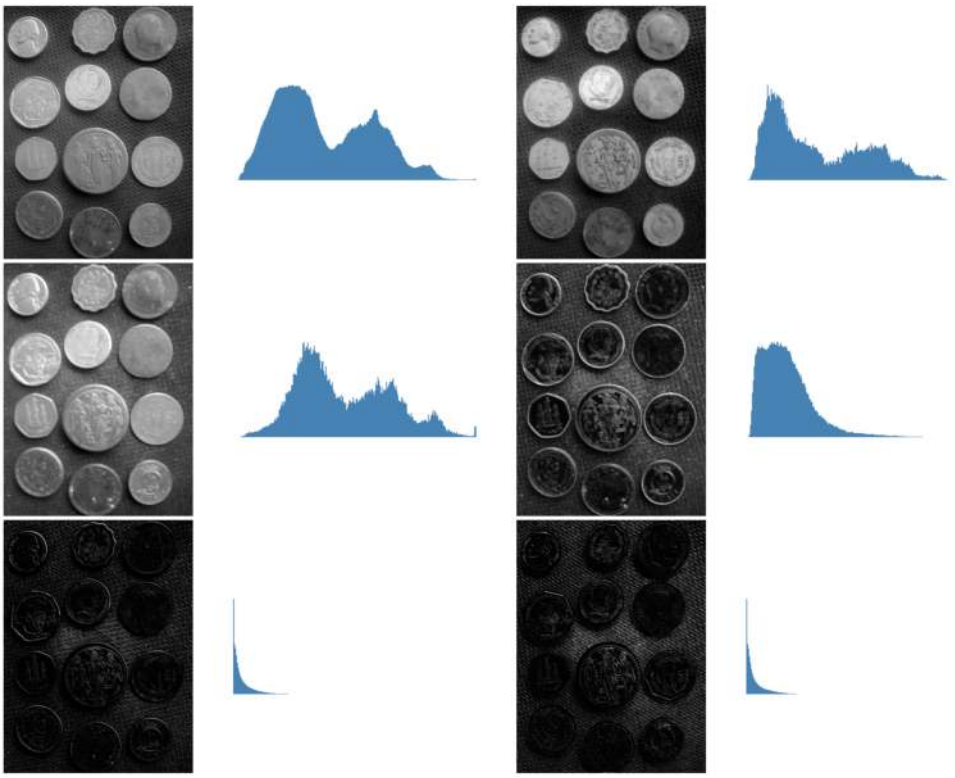


Figure 4.14: Morfologia em tons de cinza e seus histogramas: erosão, dilatação, gradiente, top-hat e black-hat. Elemento estruturante: disco de diâmetro 19.

Les opérateurs morphologiques présentés précédemment seront désormais utilisés comme outils de raffinement et de génération de marqueurs pour des méthodes de segmentation plus avancées, présentées ci-après.

4.4 Segmentation d'Images : Fondements et Taxinomie

La **segmentation d'images** consiste à diviser l'image en régions associées à des objets ou à des structures d'intérêt. En TNI, elle représente la transition entre le traitement de bas niveau — comme le filtrage et le rehaussement — et des étapes d'analyse plus avancées, telles que l'extraction de caractéristiques, la reconnaissance et l'interprétation de la scène.

Formellement, l'objectif de la segmentation consiste à décomposer le domaine spatial complet d'une image, noté Ω, en une partition de sous-ensembles $\{R_1, R_2, \dots, R_n\}$ qui satisfait simultanément les critères de **complétude** et de **disjonction** :

$$\bigcup_{i=1}^{n} R_i = \Omega, \qquad R_i \cap R_j = \emptyset \quad \forall\, i \neq j \tag{4.15}$$

Outre les propriétés de complétude et de disjonction exprimées dans la Équation 4.15, chaque sous-région R_i doit constituer un domaine **homogène** selon un prédicat de similarité défini sur des propriétés locales — intensité, couleur ou texture — et, simultanément, être **distincte** des régions adjacentes.

Les techniques de segmentation peuvent être organisées en différentes familles. Dans ce chapitre, l'accent sera mis sur les approches résumées dans la Table 4.4, fondées principalement sur des critères d'intensité, de connectivité et de proximité spatiale.

Tableau 4.4: Taxinomie simplifiée des principales approches de segmentation et d'affinement étudiées dans ce chapitre.

Approche	Critère de Segmentation	Opérateurs de Référence
Seuillage	Partitionnement de l'espace des intensités	Critère d'Otsu, seuillage global et local
Morphologie Mathématique	Relations spatiales définies par des éléments/fonctions structurants	Érosion, dilatation, ouverture, fermeture et reconstruction
Basée sur les Régions	Homogénéité locale et connectivité spatiale	Étiquetage des composantes connexes, Transformée de Distance et *Watershed*

Jusqu'à ce point, le développement pratique s'est concentré sur le **seuillage**, au moyen de la combinaison entre l'égalisation adaptative CLAHE et la méthode globale d'Otsu. Cette étape a été complétée par des opérateurs de **reconstruction morphologique** basés sur des dilatations géodésiques, implémentés par les fonctions `mm::infrec`, `mm::clohole` et `mm::edgeoff`, produisant un masque binaire propre et adapté à l'analyse.

Cependant, dans les scénarios où des objets distincts apparaissent connectés dans le masque binaire — que ce soit par contact physique, chevauchement partiel ou par de minces ponts de pixels produits par la segmentation —, le seuillage cesse d'être suffisant pour individualiser chaque objet. Dans ces cas, de multiples objets composent une seule composante connexe, ce qui complique les étapes ultérieures de mesure et d'interprétation.

Pour surmonter cette limitation, les prochaines sections introduisent trois outils complémentaires : l'**Étiquetage des Composantes Connexes**, la **Transformée de Distance** et l'algorithme de segmentation par ***Watershed*** basé sur des marqueurs. Ensemble, ces techniques permettent de séparer les objets adjacents, d'identifier les régions individuellement et d'extraire des descripteurs géométriques cohérents pour une analyse quantitative.

4.4.1 Étiquetage

Le **étiquetage de composantes connexes** (*connected component labeling*) est l'opérateur qui attribue un identifiant entier unique à chaque ensemble de pixels appartenant à une même composante connexe dans une image binaire.

> **i** Définition formelle
>
> Étant donné une image binaire f et une relation de connectivité définie par un élément structurant B (typiquement connectivité-4 ou connectivité-8), l'algorithme d'étiquetage de la Figure 4.15 produit une image g dans laquelle tous les pixels appartenant à la même composante connexe reçoivent le même étiquette entier positif, tandis que les pixels appartenant à des composantes distinctes reçoivent des étiquettes différentes.

La connectivité définit quels pixels sont considérés comme voisins directs d'un pixel (x, y). Les définitions les plus couramment utilisées sont :

- **Connectivité-4 :** ne considère que les quatre voisins orthogonaux (nord, sud, est et ouest).
- **Connectivité-8 :** considère les quatre voisins orthogonaux ainsi que les quatre voisins diagonaux, totalisant huit voisins.

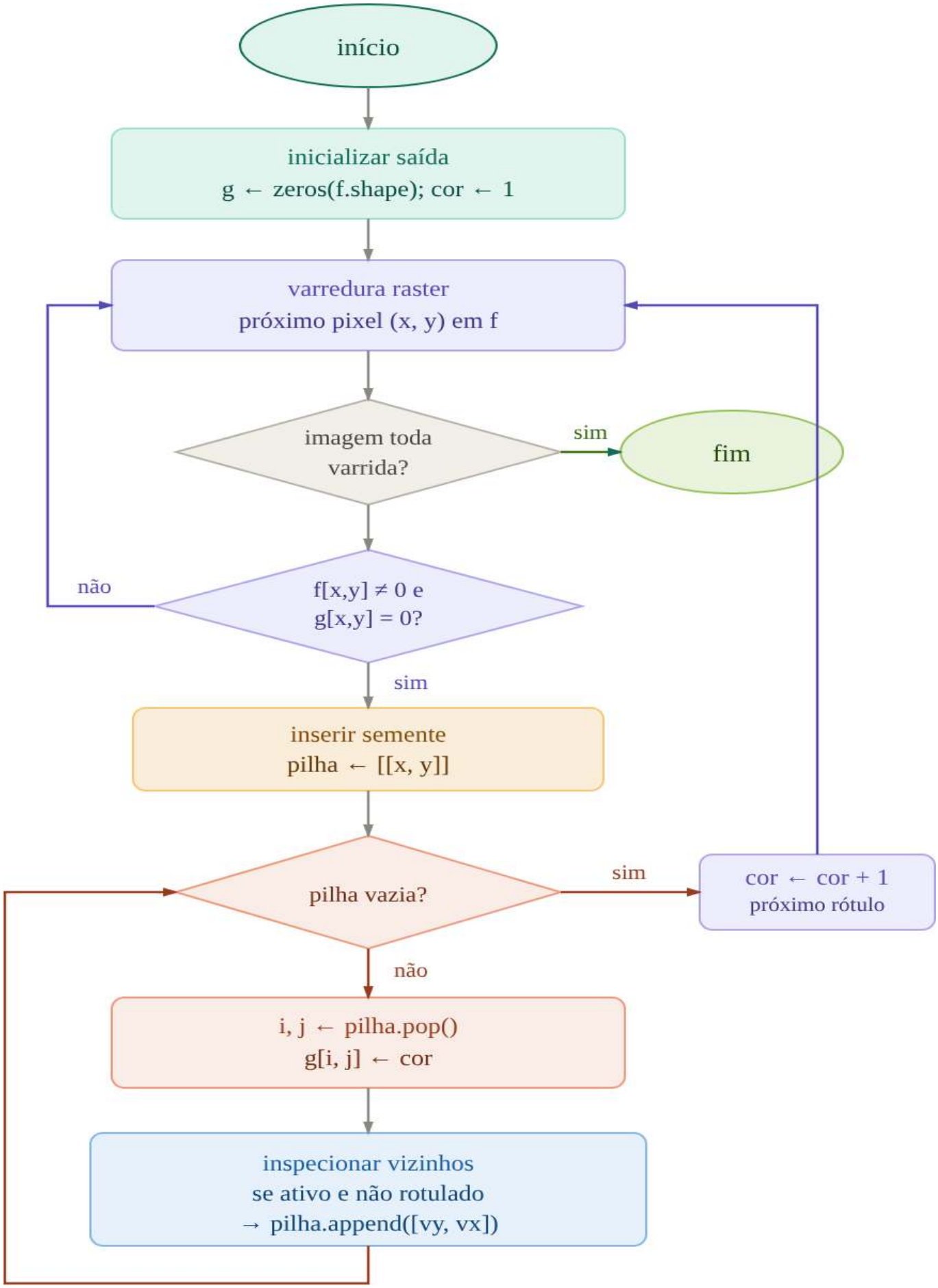
Le choix de la connectivité influence directement la formation des composantes connexes et, par conséquent, le résultat de l'étiquetage, comme illustré dans la Figure 4.17. Un exemple supplémentaire peut être exploré de manière interactive dans le simulateur présenté dans la Figure 4.16.

L'implémentation dans `morph.py` propose deux versions de cet opérateur. La fonction `mm::label0` reproduit explicitement l'algorithme de *flood-fill* à l'aide d'une pile et permet de contrôler la connectivité via l'élément structurant adopté. Quant à `mm::label`, elle délègue l'opération à l'implémentation optimisée d'OpenCV (`mm::label0`). Dans les deux cas, le résultat est une image étiquetée dans laquelle chaque composante connexe reçoit un identifiant entier distinct.

L'auxiliaire `_viz` itère sur la fenêtre structurante `b` centrée en (i, j), ne générant que les voisins valides à l'intérieur des limites de l'image — la connectivité souhaitée est entièrement déterminée par la forme de `b` passée à l'algorithme.

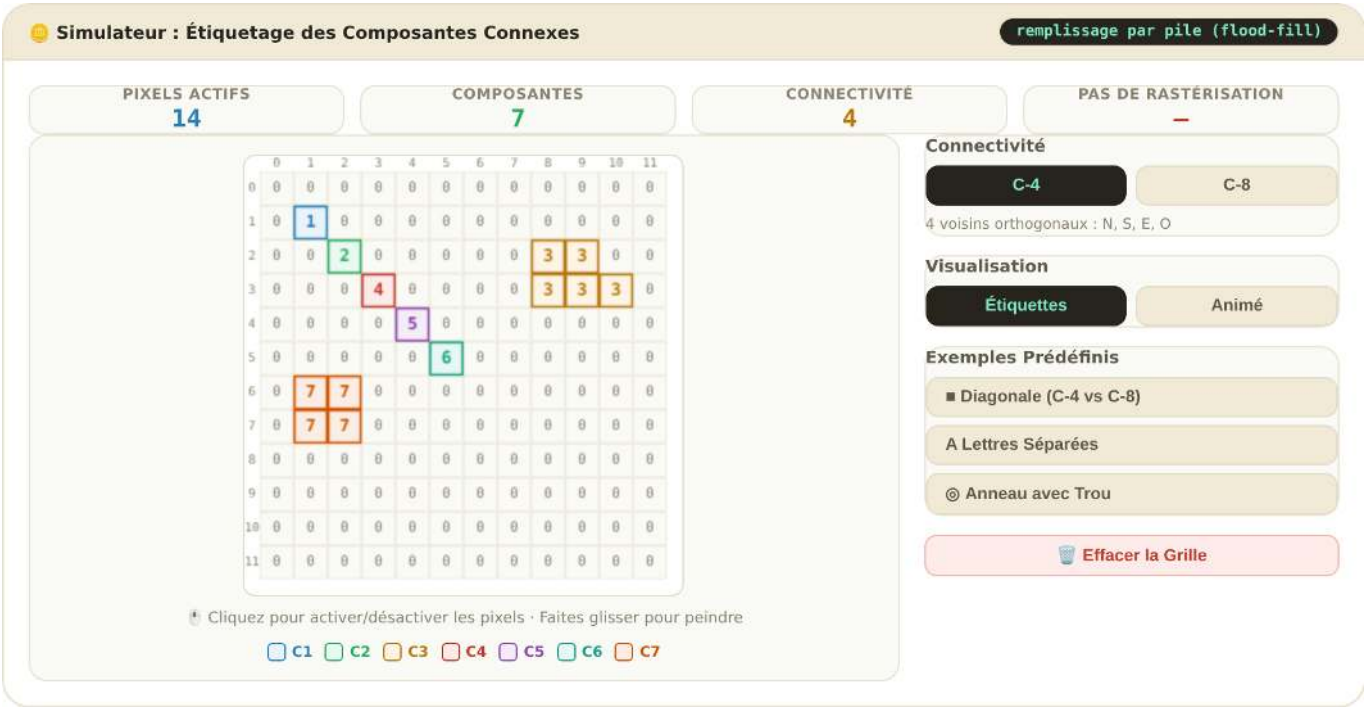
Exemple didactique — effet de la connectivité :

```
%%writefile tmp/fig_04_rotulacao_didatico.cpp
#define MM_OUT "tmp/fig_04_rotulacao_didatico.png"
//| label: fig-04-rotulacao-didatico
//| fig-cap: "Efeito da conectividade na rotulação (*mm::label0*): pixels diagonalmente
adjacentes formam componentes distintas em conectividade-4 e se fundem em conectividade-8."
//| echo: true
//| output: true

#include "morph.hpp"
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>

// Funcionamento normal da rotulação com diferentes conectividades
int main() {
    mm::Image f(10, 10);
    // Construir f a partir do array
    int dados[10][10] = {
        {0,0,0,0,0,0,0,0,0,0},
        {0,1,0,0,0,0,0,0,0,0},
        {0,0,1,0,0,0,0,0,0,0},
        {0,0,0,1,0,0,1,1,0,0},
        {0,0,0,0,0,0,1,1,0,0},
        {0,0,0,0,0,0,0,0,0,0},
        {0,1,1,1,0,0,0,0,0,0},
        {0,1,0,1,0,0,0,1,0,0},
        {0,1,1,1,0,0,0,0,1,0},
        {0,0,0,0,0,0,0,0,0,0}
    };
    for (int y = 0; y < 10; y++)
        for (int x = 0; x < 10; x++)
            f.at(y, x) = dados[y][x] * 255;

    mm::Image B4 = mm::secross();   // conectividade-4
    mm::Image B8 = mm::sebox();     // conectividade-8

    mm::Image lbl4 = mm::label0(f, B4);
    mm::Image lbl8 = mm::label0(f, B8);

    // Calcular n4 e n8 como valores máximos
    int n4 = 0, n8 = 0;
    for (int i = 0; i < (int)lbl4.data.size(); i++) {
        if (lbl4.data[i] > n4) n4 = lbl4.data[i];
    }
    for (int i = 0; i < (int)lbl8.data.size(); i++) {
        if (lbl8.data[i] > n8) n8 = lbl8.data[i];
    }
    std::cout << "Componentes C4: " << n4 << "  |  C8: " << n8 << "\n";
```



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.fr/cap04/sim-alg-rotulagem2.html

Figure 4.15: Algorithme d'étiquetage par *flood-fill* avec pile.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.fr/cap04/sim-04-rotulacao.html

Figure 4.16: Simulateur interactif d'étiquetage de composantes connexes (*connected component labeling*) : visualisation de l'expansion flood-fill, connectivité-4 et connectivité-8.

```
    // Função norm_label
    auto norm_label = [](const mm::Image& lbl, int mx) {
        mm::Image out = lbl;
        for (int y = 0; y < lbl.h; y++) {
            for (int x = 0; x < lbl.w; x++) {
                if (lbl.at(y, x))
                    out.at(y, x) = (unsigned char)((int)lbl.at(y, x) * 255 / mx);
            }
        }
        return out;
    };

    int mx4 = std::max(n4, 1);
    int mx8 = std::max(n8, 1);

    mm::show(
        std::vector<mm::Image>{f, norm_label(lbl4, mx4), norm_label(lbl8, mx8)},
        MM_OUT,
        std::vector<std::string>{"f original", "C4 (" + std::to_string(n4) + " comp.)", "C8 ("
+ std::to_string(n8) + " comp.)"},
        3
    );

    return 0;
}
```

```
Overwriting tmp/fig_04_rotulacao_didatico.cpp
```

```
!g++ -I. -std=c++17 -DMM_USE_OPENCV -I/usr/include/opencv4 tmp/fig_04_rotulacao_didatico.cpp
-o tmp/fig_04_rotulacao_didatico -lopencv_imgproc -lopencv_imgcodecs -lopencv_core \
  && ./tmp/fig_04_rotulacao_didatico \
```

```
  && test -f "tmp/fig_04_rotulacao_didatico.png" \
  || echo "  mm::show não gravou tmp/fig_04_rotulacao_didatico.png"
```

```
Componentes C4: 7  |  C8: 4
[1] f original
[2] C4 (7 comp.)
[3] C8 (4 comp.)
```

```
try:
    mm.show(mm.read("tmp/fig_04_rotulacao_didatico.png"))
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + "
    tmp/fig_04_rotulacao_didatico.png (ver a versao Python)")
```

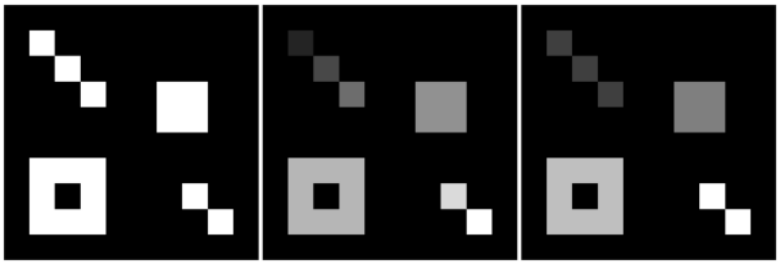


Figure 4.17: Efeito da conectividade na rotulação (*mm::label0*): pixels diagonalmente adjacentes formam componentes distintas em conectividade-4 e se fundem em conectividade-8.

4.4.2 Transformée de Distance

La **Transformée de Distance** (TD) est un opérateur qui, appliqué à une image binaire f, produit une image en niveaux de gris D dans laquelle chaque pixel appartenant à l'objet ($f(x, y) \neq 0$) reçoit comme valeur la distance géométrique jusqu'au pixel de fond ($f(x', y') = 0$) le plus proche :

$$D(x, y) = \min_{(x', y') \,:\, f(x', y') = 0} d((x, y), (x', y'))$$

où $d(\cdot, \cdot)$ est une métrique de distance — typiquement la distance euclidienne (L_2). Le résultat est une représentation topographique des objets : les pixels situés à l'intérieur prennent des valeurs élevées, tandis que les pixels proches des bords présentent de faibles valeurs de distance. Les **maxima locaux** de D correspondent aux points les plus éloignés du bord de l'objet, souvent proches de leurs centres géométriques ou de leurs centres d'inscription maximale — propriété particulièrement utile pour la génération automatique de marqueurs dans l'algorithme *watershed*.

i Définition formelle utilisant des érosions

La TD admet également une interprétation morphologique itérative, selon l'algorithme de la Figure 4.18. Considérons un élément structurant b dont la valeur centrale est nulle et dont les voisins possèdent des coûts négatifs associés au déplacement. En appliquant des érosions successives avec cet élément structurant particulier, les valeurs des pixels des objets (qui doivent prendre la distance maximale possible de l'image) sont progressivement réduites selon les coûts définis par b. La valeur accumulée de cette propagation représente alors la distance au fond selon la métrique induite par l'élément structurant.

Cette interprétation est implémentée dans `mm::dist1()`, qui accumule des érosions successives en utilisant l'opération `mm::ero1()`. Quant à `mm::dist()`, elle délègue le calcul de la distance euclidienne à l'opérateur optimisé d'OpenCV `mm::dist(f)`, où `f` est l'image binaire d'entrée, la distance L2 spécifie la métrique

euclidienne (L_2) et `5` indique l'utilisation d'un masque 5×5 pour approximer la distance avec une grande précision.

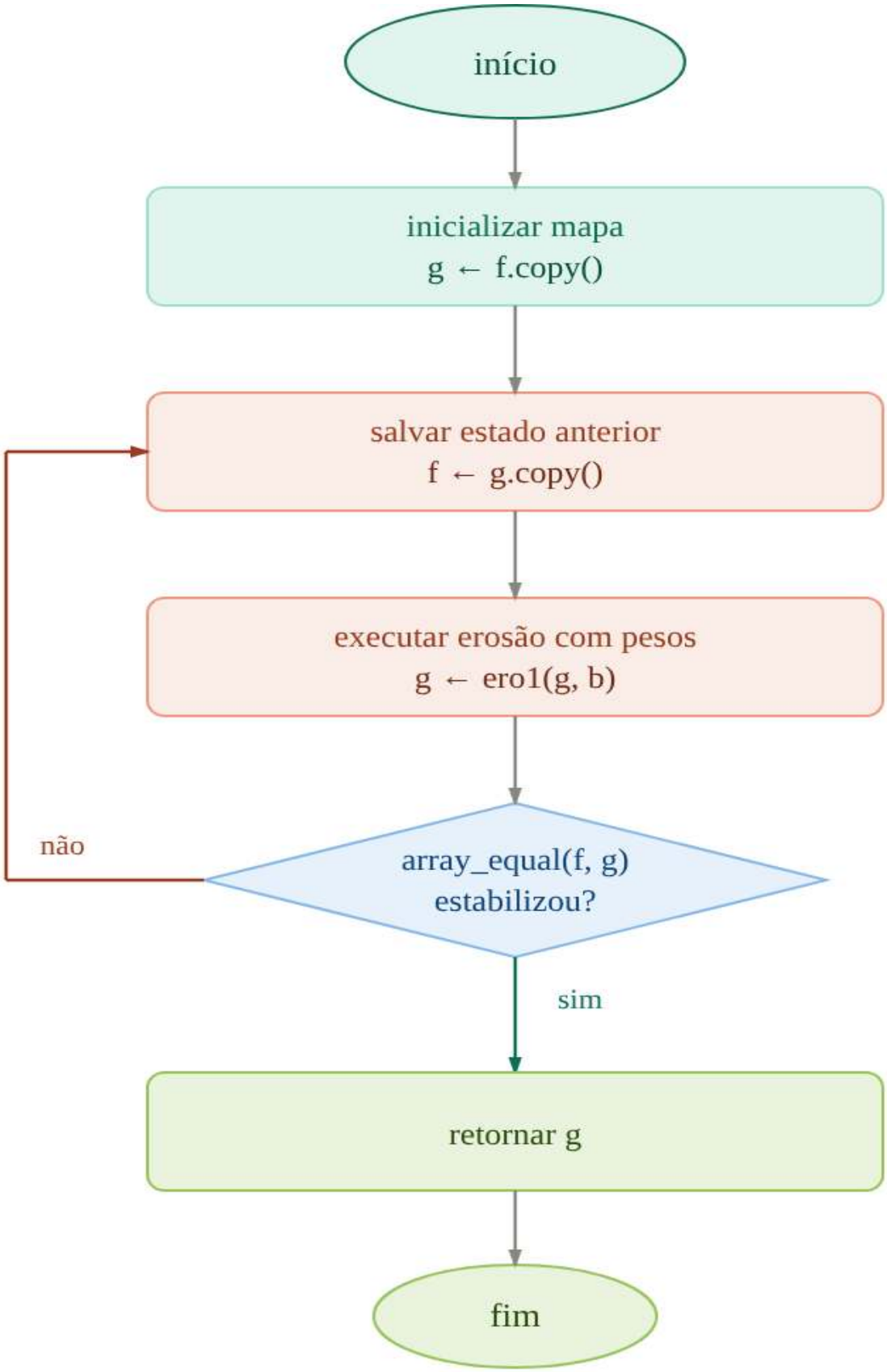
La fonction `dist1` produit une transformée de distance discrète dont la métrique est déterminée par la géométrie et les poids de l'élément structurant utilisé. Par exemple, en utilisant un élément structurant en croix avec un coût unitaire pour les quatre voisins orthogonaux, on obtient la distance de ***Manhattan*** (L_1). D'autres choix de voisinage et de poids induisent des métriques différentes. Quant à `mm::dist()`, elle calcule une approximation efficace de la distance euclidienne (L_2).

En raison de la nécessité d'érosions successives sur toute l'image, l'approche `dist1` présente un coût computationnel significativement plus élevé que `mm::dist()`, étant utilisée dans ce livre principalement à des fins pédagogiques et pour mettre en évidence la relation entre la morphologie mathématique et les transformées de distance.

A Figure 4.19 apresenta um simulador interativo da TD: é possível posicionar o cursor sobre diferentes pixels do objeto e observar, em tempo real, o valor da distância associado àquela posição, isto é, a distância até o pixel de fundo mais próximo. A Figure 4.20 apresenta um exemplo prático dessa execução em ambiente Python.

A Figure 4.19 apresenta um simulador interativo da transformada de distância (TD): é possível posicionar o cursor sobre diferentes pixels do objeto e observar, em tempo real, o valor da distância associado a essa posição, ou seja, a distância até o pixel de fundo mais próximo. A Figure 4.20 apresenta um exemplo prático dessa execução em ambiente Python.

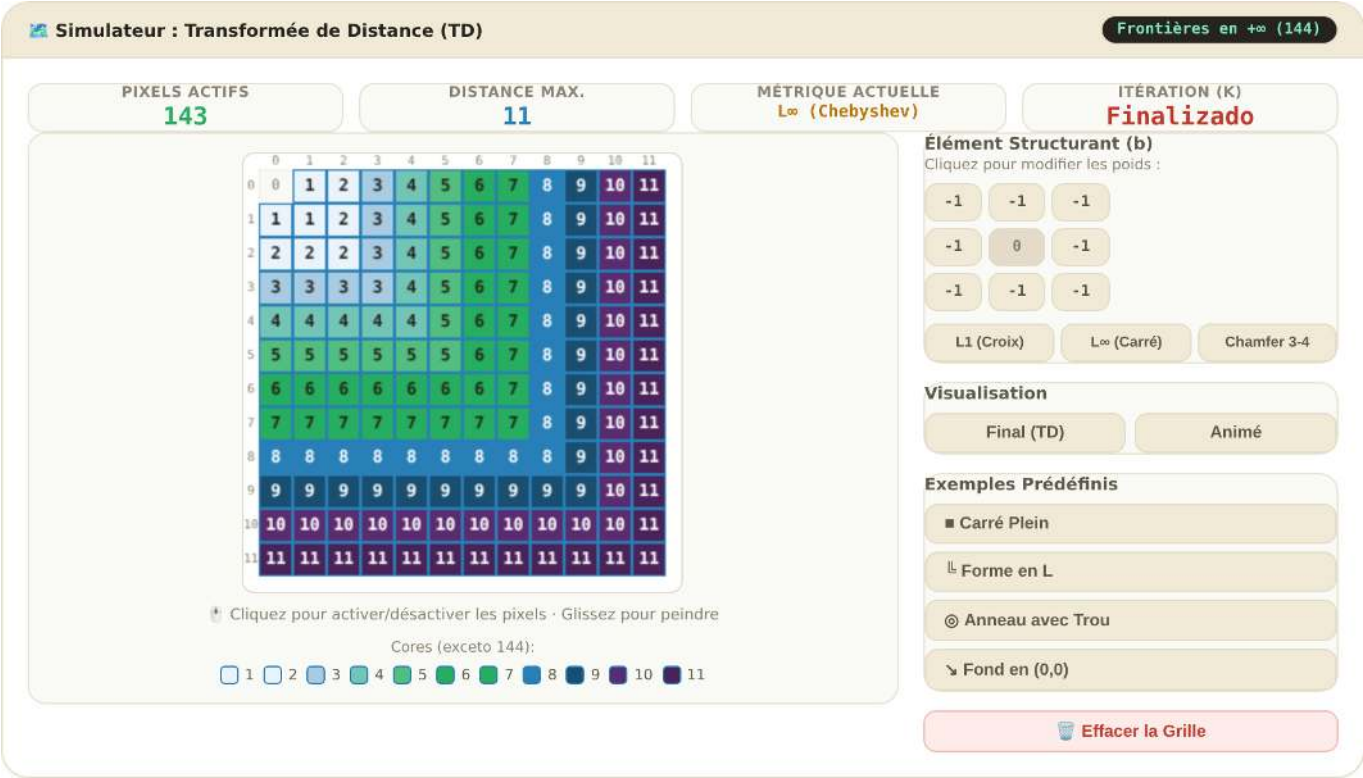
```
%%writefile tmp/fig_04_distancia_didatico.cpp
#define MM_OUT "tmp/fig_04_distancia_didatico.png"
//| label: fig-04-distancia-didatico
//| fig-cap: "Transformada de Distância em imagem binária 10×10. Esquerda: original
(*foreground* = 255). Centro: mm.dist1 iterativa (erosões com cruz). Direita: mm.dist (L2)."
//| echo: true
//| output: true

#include "morph.hpp"
#include <iostream>
#include <filesystem>

int main() {
    mm::Image f(10, 10);
    std::fill(f.data.begin(), f.data.end(), 255);
    f.at(0, 0) = 0;

    mm::SE B_cruz{{mm::SE_OUT, -1, mm::SE_OUT}, {-1, 0, -1}, {mm::SE_OUT, -1, mm::SE_OUT}};

    mm::Image d_iter = mm::dist1(f, B_cruz);
    mm::Image d_l2   = mm::dist(f);

    // Max values
    unsigned char max_iter = 0, max_l2 = 0;
    for (int i = 0; i < d_iter.h * d_iter.w; ++i) {
        if (d_iter.data[i] > max_iter) max_iter = d_iter.data[i];
        if (d_l2.data[i] > max_l2) max_l2 = d_l2.data[i];
    }
    std::cout << "Máx. dist1 (erosões) : " << (int)max_iter << " px\n";
    std::cout << "Máx. dist  (L2)      : " << (int)max_l2 << " px\n";

    mm::show({f, d_iter, d_l2}, MM_OUT, {"f original", "mm.dist1 (erosões com cruz)", "mm.dist
(L2)"}, 3);


// [pdi:panel-io] auto-generated - do not edit by hand
```



Ponto Fixo Numérico: A distância emerge da propagação matemática do valor zero.

Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.fr/cap04/sim-alg-distancia2.html

Figure 4.18: Algorithme de la Transformée de Distance.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.fr/cap04/sim-04-distancia.html

Figure 4.19: Simulateur interactif de la Transformée de Distance (TD) itérative via érosion en niveaux de gris. Les pixels hors de l'image prennent la valeur maximale (144), propageant les coûts depuis le fond interne.

```
std::filesystem::create_directories("tmp");
mm::write(f, "tmp/fig_04_distancia_didatico_0.png");
mm::write(d_iter, "tmp/fig_04_distancia_didatico_1.png");
mm::write(d_l2, "tmp/fig_04_distancia_didatico_2.png");
// [pdi:panel-io:end]
return 0;
}
```

```
Overwriting tmp/fig_04_distancia_didatico.cpp
```

```
!g++ -I. -std=c++17 -DMM_USE_OPENCV -I/usr/include/opencv4 tmp/fig_04_distancia_didatico.cpp
-o tmp/fig_04_distancia_didatico -lopencv_imgproc -lopencv_imgcodecs -lopencv_core \
  && ./tmp/fig_04_distancia_didatico \
  && test -f "tmp/fig_04_distancia_didatico.png" \
  || echo "  mm::show não gravou tmp/fig_04_distancia_didatico.png"
```

```
Máx. dist1 (erosões) : 18 px
Máx. dist  (L2)      : 13 px
[1] f original
[2] mm.dist1 (erosões com cruz)
[3] mm.dist (L2)
```

```
try:
    mm.show(
        [
            mm.read("tmp/fig_04_distancia_didatico_0.png"),
            mm.read("tmp/fig_04_distancia_didatico_1.png"),
            mm.read("tmp/fig_04_distancia_didatico_2.png"),
        ],
```

```
        titles=[
            'f original',
            'mm.dist1 (erosões com cruz)',
            'mm.dist (L2)',
        ],
        cols=3,
        axis=True,
        figsize=(12, 4),
    )
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + "
    tmp/fig_04_distancia_didatico_0.png (ver a versao Python)")
```

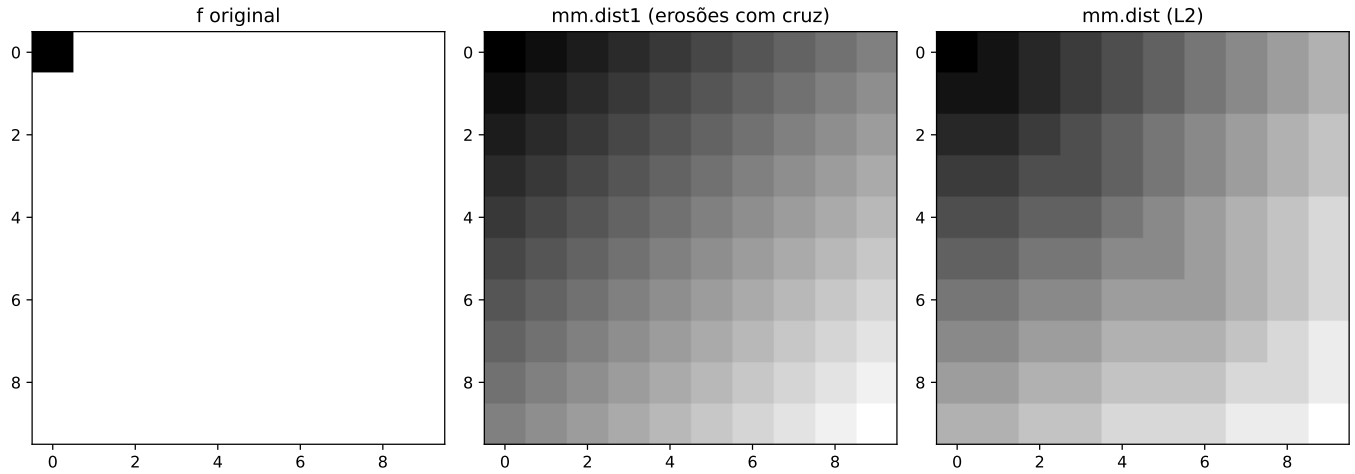
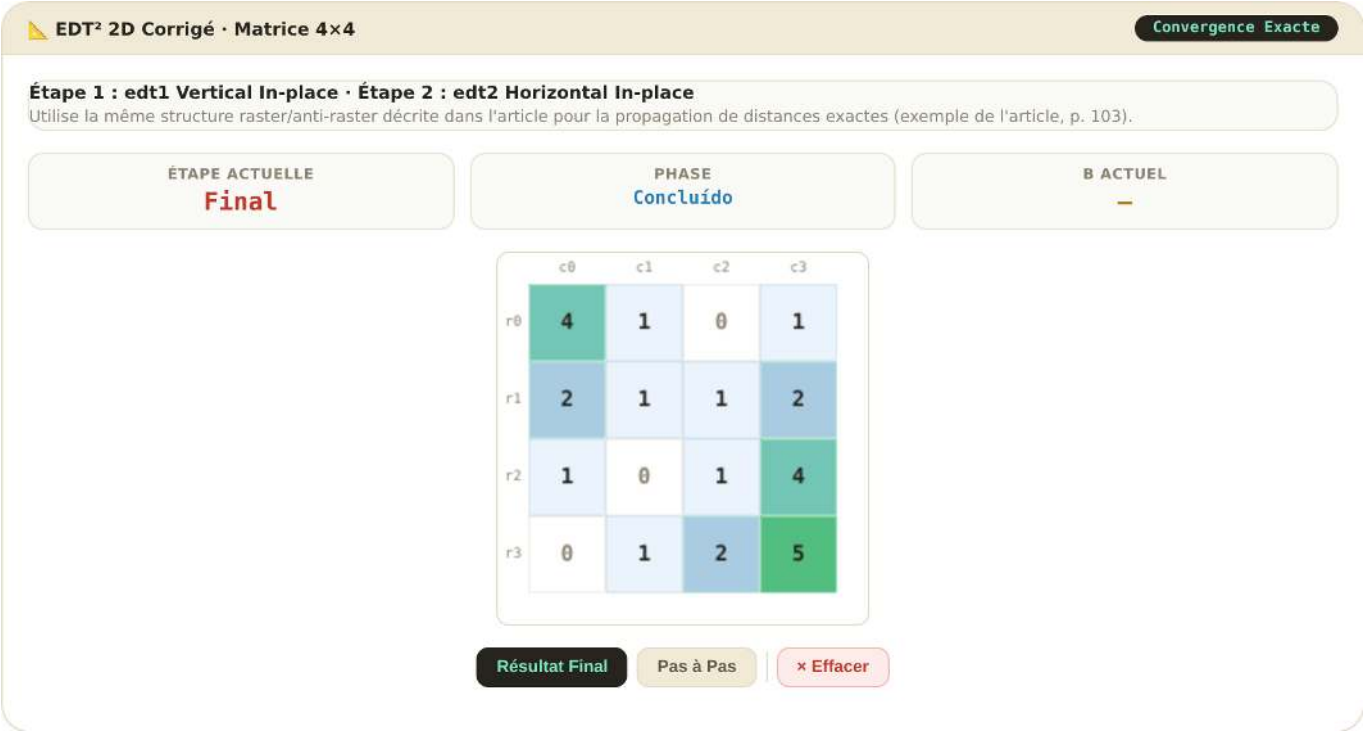


Figure 4.20: Transformada de Distância em imagem binária 10×10. Esquerda: original (*foreground* = 255). Centro: mm.dist1 iterativa (erosões com cruz). Direita: mm.dist (L2).

L'annotation des valeurs numériques directement sur les pixels permet de vérifier comment `dist1` propage les distances selon la métrique induite par la fonction structurante utilisée. Dans le cas de l'élément en croix à coût unitaire, les valeurs obtenues correspondent à la distance de *Manhattan* (L_1). Bien que `dist1` et `mm::dist` produisent des valeurs numériques distinctes en adoptant des métriques différentes, les deux transformées préservent la structure topographique des objets, faisant en sorte que leurs maxima se produisent dans des régions centrales similaires. Cette propriété justifie l'utilisation de `mm::dist` dans des applications pratiques, en raison de sa grande efficacité computationnelle.

4.4.3 Transformada de Distância Euclidiana em quatro passos

Le simulateur de la Figure 4.21 implémente l'algorithme de la TDE de Lotufo (2001) en deux étapes. Dans la première étape, la fonction `edt1` effectue une transformation unidimensionnelle verticale de manière séquentielle (*in-place*), parcourant chaque colonne en *raster* (↓) et en *anti-raster* (↑) pour calculer les distances dans la direction verticale (deux étapes : Sud et Nord). Dans la deuxième étape, la fonction `edt2` utilise ce résultat comme entrée et effectue une propagation horizontale par files : pour chaque ligne de la matrice, deux files de priorité, `Eq` et `Wq`, sont initialisées en parcourant les indices de colonne dans des directions opposées (`Eq` de `W-1` à `1`, `Wq` de `2` à `W`), de sorte que chaque pixel mis à jour met immédiatement en file ses voisins pour retraitement dans la même itération (deux étapes supplémentaires : Est et Ouest). Ce mécanisme de file permet d'appliquer des érosions successives avec des poids impairs croissants (`b = 1, 3, 5, ...`, incrémenté à chaque itération de la boucle externe) sans que la propagation ne reste piégée dans des valeurs obsolètes, car chaque itération résout complètement la chaîne de dépendances horizontale avant l'incrément suivant de `b`. La convergence de cette propagation produit la Transformada de Distância Euclidiana sur toute la matrice, combinant l'information verticale obtenue dans `edt1` avec la propagation horizontale par file réalisée dans `edt2`.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.fr/cap04/sim-04-tde.html

Figure 4.21: Simulateur interactif 2D (4x4) avec synchronisation stricte des files de propagation horizontale (b) pour obtenir la convergence exacte décrite dans l'article.

4.4.3.1 Transformée de Distance Géodésique

La transformée de distance géodésique associe à chaque pixel la distance minimale jusqu'à un marqueur, sous la contrainte imposée par un masque. Ainsi, la propagation s'effectue exclusivement à travers les pixels autorisés, préservant la connectivité du domaine.

La Figure 4.22 illustre ce processus dans un labyrinthe : (a) le masque `g` ; (b) la distance géodésique `D1` calculée depuis l'entrée ; (c) la distance `D2` calculée depuis la sortie ; et (d) le chemin minimal obtenu à partir de ces deux transformées.

Le chemin optimal est déterminé par la somme des distances (`D1 + D2`). Les pixels appartenant à la trajectoire minimale sont ceux pour lesquels cette somme atteint sa plus petite valeur, définissant une connexion entre l'entrée et la sortie avec une longueur géodésique minimale.

Ce principe permet de résoudre des labyrinthes sans avoir à explorer explicitement toutes les possibilités de parcours. La solution émerge directement de la propagation des distances dans un domaine restreint. Cette approche est particulièrement pertinente pour les labyrinthes de haute complexité, comme ceux construits à partir de structures quasicristallines et de cycles hamiltoniens décrits par Singh (2024).. Dans Zampirolli (2025), ce même formalisme est employé pour résoudre un labyrinthe complexe ; ensuite, la méthode est illustrée sur une version simplifiée du problème.

```
%%writefile tmp/fig_04_gdist_menor_caminho.cpp
#define MM_OUT "tmp/fig_04_gdist_menor_caminho.png"
//| label: fig-04-gdist-menor-caminho
//| fig-cap: "Menor caminho geodésico em um labirinto. As distâncias geodésicas são calculadas
a partir da entrada e da saída usando mm.gdist. Os pixels cujo somatório das duas distâncias é
igual à distância mínima entre os marcadores pertencem a um caminho ótimo."
//| echo: true
//| output: true

#include "morph.hpp"
```

```cpp
#include <iostream>
#include <vector>
#include <string>

int main() {

    // 1 = corredor, 0 = parede
    mm::Image g(10, 10);
    // Initialize g with the maze values
    int maze[10][10] = {
        {0,1,0,0,0,0,0,0,0,0},
        {0,1,1,1,1,1,0,1,1,1},
        {0,0,0,0,0,1,0,1,0,1},
        {0,1,1,1,0,1,1,1,0,1},
        {0,1,0,1,0,0,0,0,0,1},
        {0,1,0,1,1,1,1,1,1,1},
        {0,1,0,0,0,0,0,0,1,0},
        {0,1,1,1,1,1,1,0,1,0},
        {0,0,0,0,0,0,1,1,1,0},
        {0,0,0,0,0,0,0,0,1,0}
    };
    for (int y = 0; y < 10; y++) {
        for (int x = 0; x < 10; x++) {
            g.at(y, x) = maze[y][x];
        }
    }

    // marcador da entrada
    mm::Image entrada(10, 10);
    entrada.at(0, 1) = 1;

    // marcador da saída
    mm::Image saida(10, 10);
    saida.at(9, 8) = 1;

    // distâncias geodésicas
    mm::Image D1 = mm::gdist(g, entrada);
    mm::Image D2 = mm::gdist(g, saida);

    // soma das distâncias
    mm::Image S(10, 10);
    for (int i = 0; i < 100; i++) {
        S.data[i] = D1.data[i] + D2.data[i];
    }

    // menor valor válido da soma
    int dmin = 255;
    for (int i = 0; i < 100; i++) {
        if (S.data[i] > 0 && S.data[i] < dmin) {
            dmin = S.data[i];
        }
    }

    // pixels pertencentes a um caminho ótimo
    mm::Image caminho(10, 10);
    for (int i = 0; i < 100; i++) {
        caminho.data[i] = (S.data[i] == dmin) ? 255 : 0;
    }

    std::cout << "Distância geodésica mínima: " << dmin << "\n";
```

```
    mm::show(
        std::vector<mm::Image>{g, D1, D2, caminho},
        MM_OUT,
        std::vector<std::string>{
            "Labirinto",
            "Distância da Entrada",
            "Distância da Saída",
            "Menor Caminho\n(d=" + std::to_string(dmin) + ")"
        },
        4
    );

    return 0;
}
```

```
Overwriting tmp/fig_04_gdist_menor_caminho.cpp
```

```
!g++ -I. -std=c++17 -DMM_USE_OPENCV -I/usr/include/opencv4 tmp/fig_04_gdist_menor_caminho.cpp
-o tmp/fig_04_gdist_menor_caminho -lopencv_imgproc -lopencv_imgcodecs -lopencv_core \
  && ./tmp/fig_04_gdist_menor_caminho \
  && test -f "tmp/fig_04_gdist_menor_caminho.png" \
  || echo "  mm::show não gravou tmp/fig_04_gdist_menor_caminho.png"
```

```
Distância geodésica mínima: 15
[1] Labirinto
[2] Distância da Entrada
[3] Distância da Saída
[4] Menor Caminho
(d=15)
```

```
try:
    mm.show(mm.read("tmp/fig_04_gdist_menor_caminho.png"), figsize=(14, 4))
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + "
    tmp/fig_04_gdist_menor_caminho.png (ver a versao Python)")
```

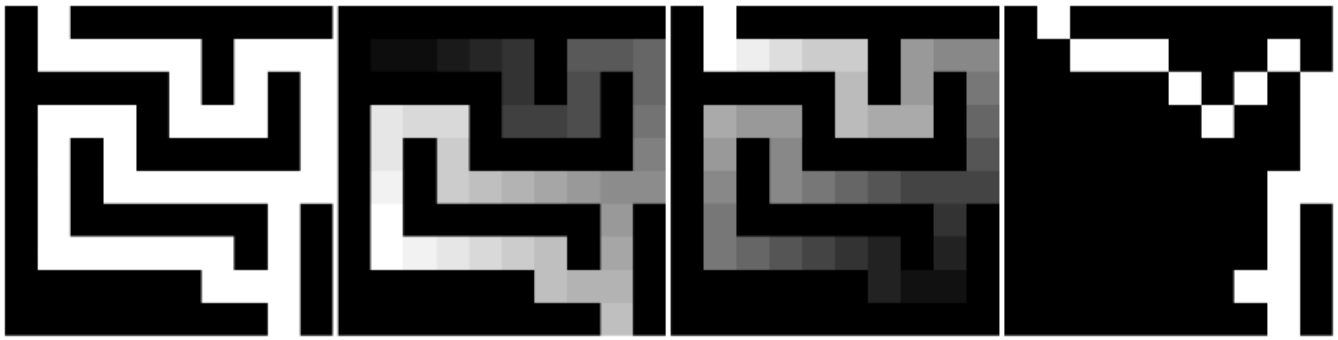


Figure 4.22: Menor caminho geodésico em um labirinto. As distâncias geodésicas são calculadas a partir da entrada e da saída usando mm.gdist. Os pixels cujo somatório das duas distâncias é igual à distância mínima entre os marcadores pertencem a um caminho ótimo.

4.4.4 Segmentation par *Watershed*

L'algorithme ***Watershed*** interprète une image en niveaux de gris comme une surface topographique, où les valeurs élevées correspondent à des montagnes et les valeurs faibles à des vallées ou bassins de drainage (*catchment basins*). Dans le contexte de la segmentation basée sur des marqueurs, les maxima de la Transformée

de Distance sont souvent utilisés pour identifier des régions internes aux objets, fournissant ainsi des graines fiables pour le processus d'inondation.

La segmentation est ensuite réalisée par une simulation conceptuelle d'inondation progressive à partir de ces marqueurs. Au fur et à mesure que les bassins associés à différentes graines s'étendent, les régions voisines finissent par entrer en contact. À cet instant, des barrières virtuelles, appelées *lignes de partage des eaux* (*watershed lines*), sont construites et délimitent les objets de la scène. Ce mécanisme permet de séparer des objets adjacents ou partiellement superposés, même lorsqu'ils forment un seul composant connexe après le seuillage.

L'implémentation didactique présentée dans ce chapitre explore d'abord le concept de croissance de régions (*region growing*) confinée par un masque binaire, comme détaillé dans l'algorithme interactif de la Figure 4.23.

> **i** Version didactique versus implémentation classique
>
> La fonction `mm.watershed0`‘ n'implémente pas l'algorithme *watershed* classique. Son objectif est d'illustrer, de manière simplifiée, la propagation des marqueurs par croissance de régions (*region growing*), permettant ainsi de visualiser comment différentes graines se disputent l'occupation de l'espace disponible. La croissance est délimitée par un masque binaire de support et contrôlée par un mécanisme de stagnation, générant un résultat semblable à une partition de Voronoi restreinte à la géométrie des objets d'entrée.
> Quant à la fonction `mm::watershed`, elle utilise l'implémentation optimisée d'OpenCV (`mm::watershed`), qui réalise l'inondation sur une surface topographique définie par l'image d'entrée. Dans ce cas, la propagation des marqueurs est influencée par les valeurs des pixels, ce qui fait que les lignes de séparation se forment naturellement sur les crêtes du relief.

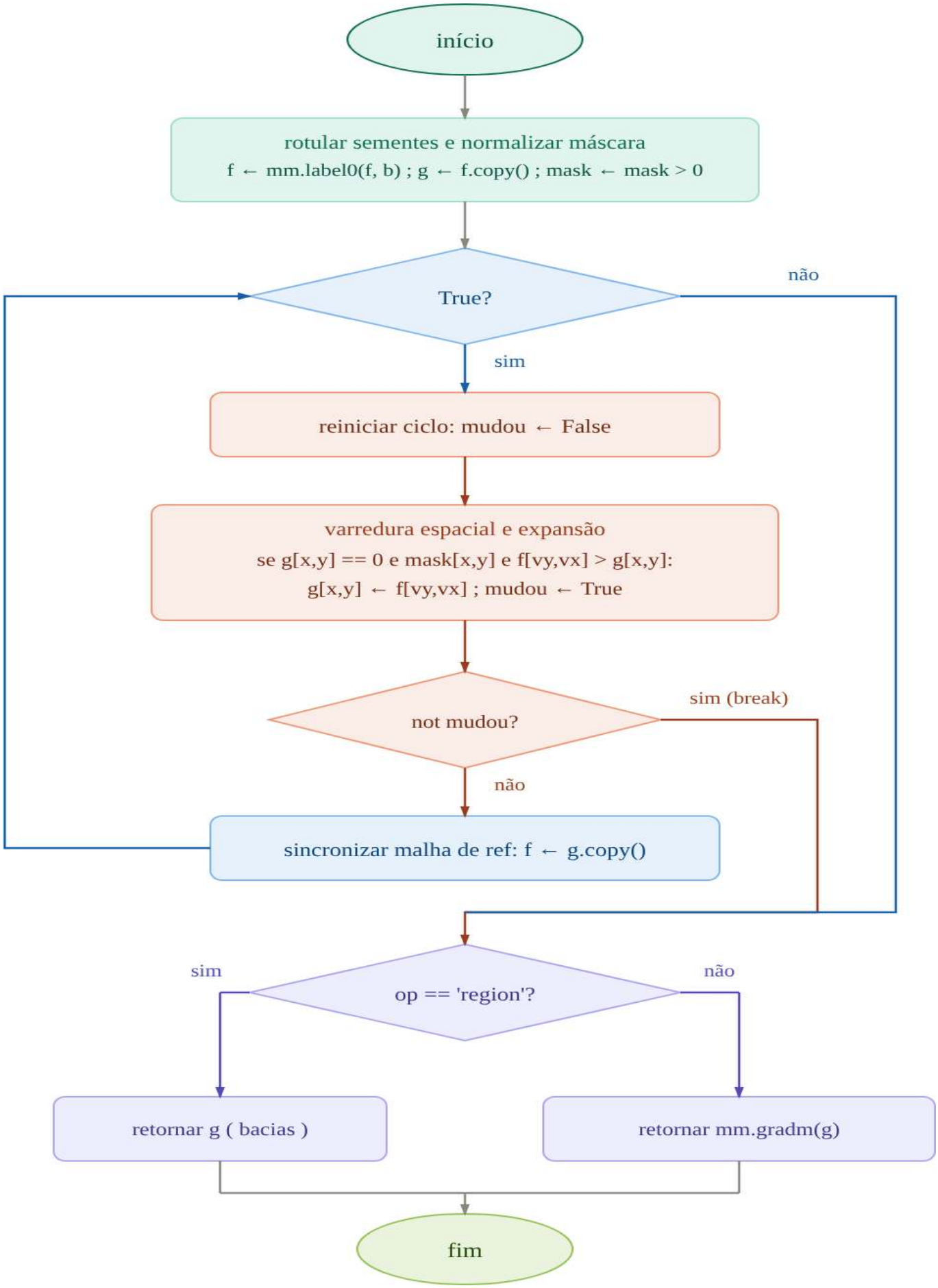
Figure 4.24 présente un simulateur itératif qui illustre la propagation des marqueurs à travers la région d'intérêt. Chaque marqueur agit comme une source d'inondation qui étend sa zone d'influence jusqu'à rencontrer des régions provenant d'autres graines. Dans l'algorithme d'OpenCV, les pixels appartenant aux lignes de partage sont identifiés par la valeur `-1`, représentant les frontières entre bassins adjacents.

Pipeline morphologique du *watershed*

Le *watershed* basé sur des marqueurs intègre généralement un flux plus large de segmentation. Dans les images réelles, des étapes de prétraitement sont souvent nécessaires pour améliorer le contraste, réduire le bruit et générer des marqueurs fiables. Ce flux complet est résumé dans la Table 4.5.

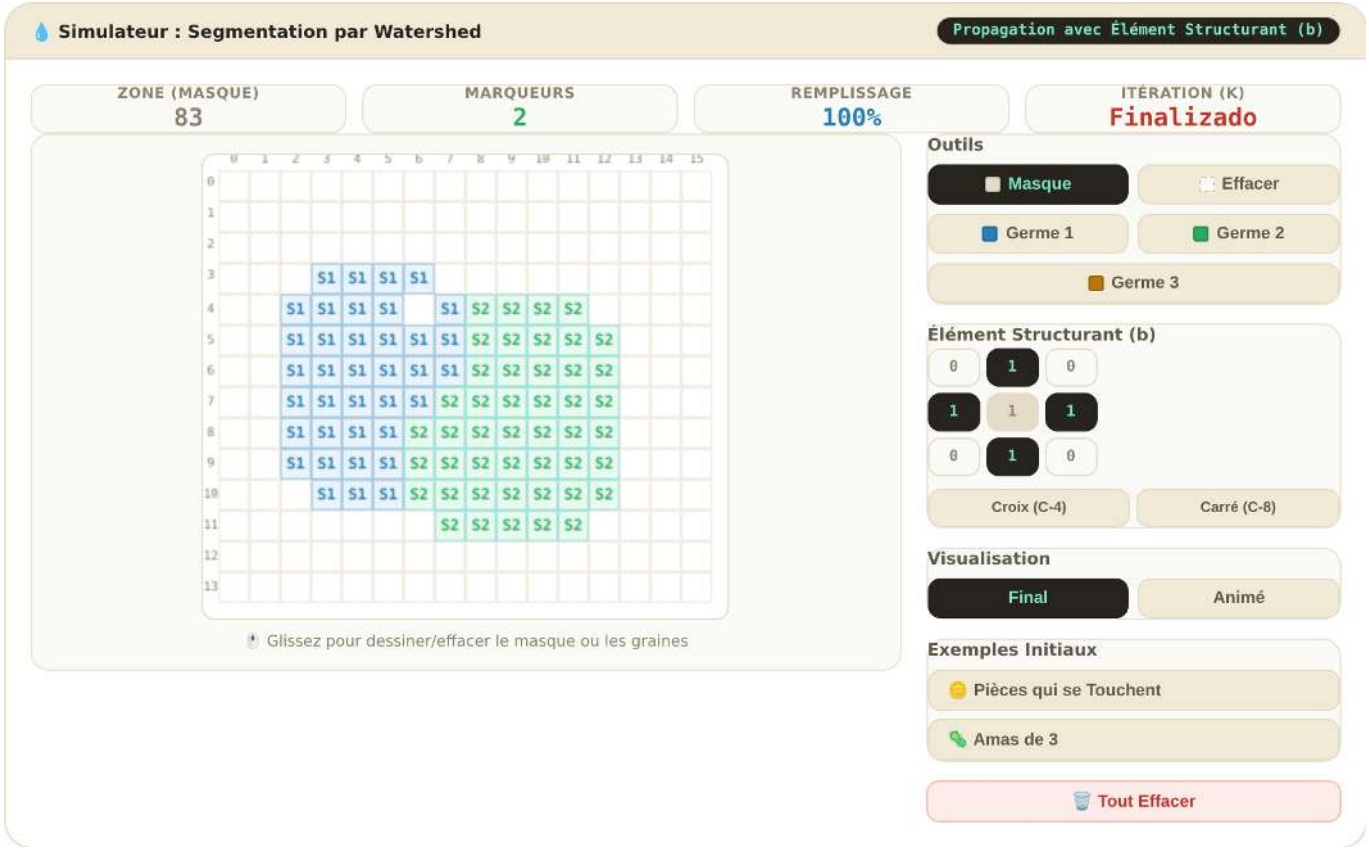
Tableau 4.5: *Pipeline* complet du *watershed* basé sur des marqueurs pour les images réelles.

Étape	Opération	Finalité
1	CLAHE + Lissage	Amélioration du contraste et réduction du bruit
2	Seuillage	Séparation initiale entre l'objet et le fond
3	Ouverture/Fermeture	Suppression du bruit et des petites imperfections
4	Dilatation du masque	Identification du fond certain (*Sure Background*)
5	Transformée de distance + Seuil	Identification de l'objet certain (*Sure Foreground*)
6	Région incertaine	Différence entre le fond certain et l'objet certain
7	`mm::watershed`	Propagation des marqueurs à travers la région incertaine



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.fr/cap04/sim-alg-watershed2.html

Figure 4.23: Algorithme didactique de *Watershed* par croissance de régions limitée par masque.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.fr/cap04/sim-04-watershed.html

Figure 4.24: Simulateur interactif de l'algorithme *Watershed* par propagation morphologique. Dessinez le masque, positionnez les marqueurs et ajustez l'élément structurant pour observer l'inondation. Lorsque les bassins se rencontrent simultanément, l'égalité est résolue en supposant l'une des régions de manière aléatoire.

Pour mettre en avant exclusivement les concepts de transformée de distance, de marqueurs et d'inondation topographique, l'exemple de la Figure 4.25 utilise une image binaire synthétique et adopte un flux simplifié, résumé dans la Table 4.6.

Tableau 4.6: *Pipeline* simplifié utilisé dans l'exemple didactique de la Figure 4.25.

Étape	Opération	Finalité
1	Transformée de distance	Construction de la surface topographique
2	Seuil de la TD	Extraction des marqueurs (*Sure Foreground*)
3	Dilatation	Détermination du fond certain (*Sure Background*)
4	Région incertaine	Différence entre le fond et les marqueurs
5	`mm::watershed`	Propagation des marqueurs et génération des frontières

```
%%writefile tmp/fig_04_watershed_didatico.cpp
#define MM_OUT "tmp/fig_04_watershed_didatico.png"
//| label: fig-04-watershed-didatico
//| fig-cap: "*Pipeline* *watershed* delimitado por máscara em imagem binária 20×20."
//| echo: true
//| output: true

#include "morph.hpp"
#include <iostream>
#include <string>
#include <vector>
#include <filesystem>

int main() {
    // 1. Imagem sintética e Transformada de Distância
    mm::Image f_sint(20, 20);
    f_sint = mm::circle(f_sint, 6, 10, 5, 255, -1);
    f_sint = mm::circle(f_sint, 14, 10, 5, 255, -1);
    mm::Image dist = mm::dist(f_sint);

    // 2. Marcadores (picos da distância)
    double max_dist = 0;
    for (int i = 0; i < dist.h * dist.w; i++) {
        if (dist.data[i] > max_dist) max_dist = dist.data[i];
    }
    mm::Image m(dist.h, dist.w);
    for (int y = 0; y < dist.h; y++) {
        for (int x = 0; x < dist.w; x++) {
            m.at(y, x) = (dist.at(y, x) > 0.8 * max_dist) ? 255 : 0;
        }
    }

    // 3. Execução do Watershed0 condicional (m=marcadores primeiro, mask=f_sint)
    mm::Image w_reg = mm::watershed0(m, f_sint, "region");
    mm::Image w_line = mm::watershed0(m, f_sint, "line");

    //w_reg  = mm::watershedB(m, mask=f_sint, op='region')
    //w_line = mm::watershedB(m, mask=f_sint, op='line')

    w_reg = mm::watershed(m, f_sint, "region");
```

```
    w_line = mm::watershed(m, f_sint, "line");

    // 4. Exibição dos resultados
    mm::show(std::vector<mm::Image>{f_sint, dist, m, w_reg, w_line}, MM_OUT,
             std::vector<std::string>{"Original", "Distância", "Marcadores", "Regiões",
"Linhas"}, 5);


// [pdi:panel-io] auto-generated - do not edit by hand
std::filesystem::create_directories("tmp");
mm::write(f_sint, "tmp/fig_04_watershed_didatico_0.png");
mm::write(dist, "tmp/fig_04_watershed_didatico_1.png");
mm::write(m, "tmp/fig_04_watershed_didatico_2.png");
mm::write(w_reg, "tmp/fig_04_watershed_didatico_3.png");
mm::write(w_line, "tmp/fig_04_watershed_didatico_4.png");
// [pdi:panel-io:end]
return 0;
}
```

```
Overwriting tmp/fig_04_watershed_didatico.cpp
```

```
!g++ -I. -std=c++17 -DMM_USE_OPENCV -I/usr/include/opencv4 tmp/fig_04_watershed_didatico.cpp
-o tmp/fig_04_watershed_didatico -lopencv_imgproc -lopencv_imgcodecs -lopencv_core \
  && ./tmp/fig_04_watershed_didatico \
  && test -f "tmp/fig_04_watershed_didatico.png" \
  || echo "  mm::show não gravou tmp/fig_04_watershed_didatico.png"
```

```
[1] Original
[2] Distância
[3] Marcadores
[4] Regiões
[5] Linhas
```

```
try:
    mm.show(
        [
            mm.read("tmp/fig_04_watershed_didatico_0.png"),
            mm.read("tmp/fig_04_watershed_didatico_1.png"),
            mm.read("tmp/fig_04_watershed_didatico_2.png"),
            mm.read("tmp/fig_04_watershed_didatico_3.png"),
            mm.read("tmp/fig_04_watershed_didatico_4.png"),
        ],
        titles=[
            'Original',
            'Distância',
            'Marcadores',
            'Regiões',
            'Linhas',
        ],
        cols=5,
        figsize=(16, 4),
    )
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + "
    tmp/fig_04_watershed_didatico_0.png (ver a versao Python)")
```

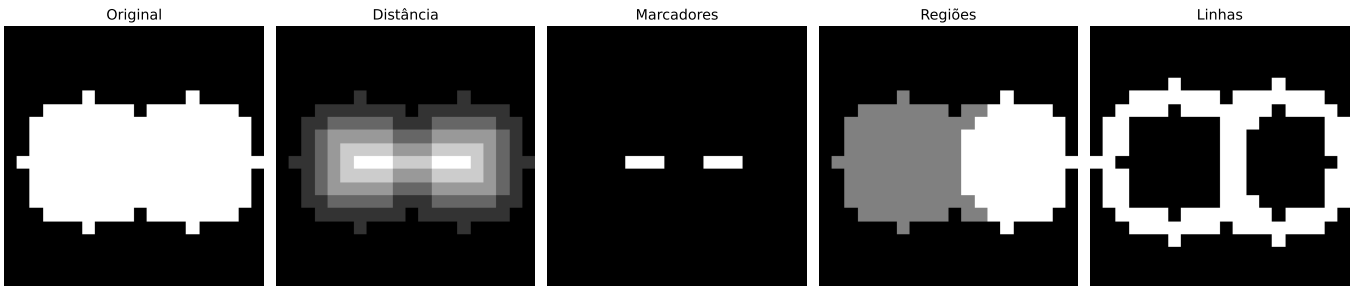


Figure 4.25: *Pipeline watershed* delimitado por máscara em imagem binária 20×20.

Application aux pièces superposées :

```
%%writefile tmp/fig_04_watershed_moedas.cpp
#define MM_OUT "tmp/fig_04_watershed_moedas.png"
//| label: fig-04-watershed-moedas
//| fig-cap: "*Pipeline* *Watershed* para separação de moedas sobrepostas: da máscara binária
dilatada até os contornos finais anotados sobre a imagem original."
//| echo: true
//| output: true

#include "morph.hpp"
#include <opencv2/opencv.hpp>
#include <algorithm>
#include <iostream>
#include <numeric>
#include <vector>
#include <set>
#include <filesystem>

int main() {
// [pdi:state-io] auto-generated - do not edit by hand
mm::Image img_coins_gray = mm::_read_state("tmp/state/img_coins_gray_8.png");
mm::Image img_final = mm::_read_state("tmp/state/img_final_55.png");
// [pdi:state-io:end]

    mm::Image img_base = img_coins_gray;

    // 1. Simular moedas sobrepostas/conectadas (usando a máscara binária base)
    mm::Image img_sobrepostas = mm::dil(img_final, mm::sebox(40));

    // 2. Abertura morfológica para limpar ruídos
    mm::Image opening = mm::open(img_sobrepostas, mm::sebox(2));

    // 3. Transformada de Distância
    mm::Image dist = mm::dist(opening);
    mm::Image dist_vis;
    int max_dist = 0;
    for (int i = 0; i < dist.h * dist.w; i++) {
        max_dist = std::max(max_dist, (int)dist.data[i]);
    }
    if (max_dist > 0) {
        dist_vis = mm::Image(dist.h, dist.w);
        for (int i = 0; i < dist.h * dist.w; i++) {
            dist_vis.data[i] = (unsigned char)(255 * (double)dist.data[i] / max_dist);
        }
    } else {
        dist_vis = dist;
    }
```

```
// 4. Picos seguros (Marcadores das moedas)
int max_dist2 = 0;
for (int i = 0; i < dist.h * dist.w; i++) {
    max_dist2 = std::max(max_dist2, (int)dist.data[i]);
}
mm::Image picos(dist.h, dist.w);
for (int i = 0; i < dist.h * dist.w; i++) {
    if (dist.data[i] > 0.5 * max_dist2) {
        picos.data[i] = 255;
    }
}

// 5. Execução do Watershed (Ajustado para usar a nova assinatura)
// Passamos 'opening' direto em mask, pois ela delimita o escopo de expansão das moedas
mm::Image ws_region = mm::watershed(picos, opening, "region");
mm::Image ws_line   = mm::watershed(picos, opening, "line");

// 6. Contagem de objetos (Ignora fundo 0)
std::set<int> unique_labels;
for (int i = 0; i < ws_region.h * ws_region.w; i++) {
    if (ws_region.data[i] > 0) {
        unique_labels.insert(ws_region.data[i]);
    }
}
std::vector<int> labels(unique_labels.begin(), unique_labels.end());
std::cout << "Objetos detectados: " << labels.size() << "\n";

// 7. Anotação Final
cv::Mat img_base_cv(img_base.h, img_base.w, CV_8UC1, img_base.data.data());
cv::Mat img_annotated;
cv::cvtColor(img_base_cv, img_annotated, cv::COLOR_GRAY2BGR);

for (size_t idx = 0; idx < labels.size(); idx++) {
    int label_id = labels[idx];
    int count = 0;
    for (int i = 0; i < ws_region.h * ws_region.w; i++) {
        if (ws_region.data[i] == label_id) count++;
    }
    if (count < 500) continue;

    // calcular centroide
    double sum_y = 0, sum_x = 0;
    int npts = 0;
    for (int y = 0; y < ws_region.h; y++) {
        for (int x = 0; x < ws_region.w; x++) {
            if (ws_region.at(y, x) == label_id) {
                sum_y += y;
                sum_x += x;
                npts++;
            }
        }
    }
    int cy = (int)(sum_y / npts);
    int cx = (int)(sum_x / npts);
    cv::putText(img_annotated, std::to_string(idx + 1), cv::Point(cx - 25, cy + 20),
                cv::FONT_HERSHEY_SIMPLEX, 3.2, cv::Scalar(0, 255, 0), 8, cv::LINE_AA);
}

// Desenha as linhas de separação em vermelho
mm::Image ones11(11, 11);
```

```
    std::fill(ones11.data.begin(), ones11.data.end(), 1);
    mm::Image mask_ann = mm::dil(ws_line, ones11);
    for (int y = 0; y < img_annotated.rows; y++) {
        for (int x = 0; x < img_annotated.cols; x++) {
            if (mask_ann.at(y, x) > 0) {
                img_annotated.at<cv::Vec3b>(y, x) = cv::Vec3b(255, 0, 0);
            }
        }
    }

    // converter de volta para mm::Image para exibição
    mm::Image img_annotated_mm(img_annotated.rows, img_annotated.cols, 3);
    std::memcpy(img_annotated_mm.data.data(), img_annotated.data,
img_annotated_mm.data.size());

    // Exibição
    mm::show(std::vector<mm::Image>{img_base, img_sobrepostas, dist_vis, picos, ws_region,
img_annotated_mm},
             MM_OUT, std::vector<std::string>{"Original", "Sobrepostas", "Distância",
"Marcadores", "Watershed", "Anotado"}, 3);


// [pdi:panel-io] auto-generated - do not edit by hand
std::filesystem::create_directories("tmp");
mm::write(img_base, "tmp/fig_04_watershed_moedas_0.png");
mm::write(img_sobrepostas, "tmp/fig_04_watershed_moedas_1.png");
mm::write(dist_vis, "tmp/fig_04_watershed_moedas_2.png");
mm::write(picos, "tmp/fig_04_watershed_moedas_3.png");
mm::write(ws_region, "tmp/fig_04_watershed_moedas_4.png");
mm::write(img_annotated, "tmp/fig_04_watershed_moedas_5.png");
// [pdi:panel-io:end]
return 0;
}
```

```
Overwriting tmp/fig_04_watershed_moedas.cpp
```

```
!g++ -I. -std=c++17 -DMM_USE_OPENCV -I/usr/include/opencv4 tmp/fig_04_watershed_moedas.cpp -o
tmp/fig_04_watershed_moedas -lopencv_imgproc -lopencv_imgcodecs -lopencv_core \
  && ./tmp/fig_04_watershed_moedas \
  && test -f "tmp/fig_04_watershed_moedas.png" \
  || echo "  mm::show não gravou tmp/fig_04_watershed_moedas.png"
```

```
Objetos detectados: 12
[1] Original
[2] Sobrepostas
[3] Distância
[4] Marcadores
[5] Watershed
[6] Anotado
```

```
try:
    mm.show(
        [
            mm.read("tmp/fig_04_watershed_moedas_0.png"),
            mm.read("tmp/fig_04_watershed_moedas_1.png"),
            mm.read("tmp/fig_04_watershed_moedas_2.png"),
            mm.read("tmp/fig_04_watershed_moedas_3.png"),
            mm.read("tmp/fig_04_watershed_moedas_4.png"),
            mm.read("tmp/fig_04_watershed_moedas_5.png"),
        ],
```

```
        titles=[
            'Original',
            'Sobrepostas',
            'Distância',
            'Marcadores',
            'Watershed',
            'Anotado',
        ],
        cols=3,
        rows=2,
        figsize=(12, 8),
    )
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + "
    tmp/fig_04_watershed_moedas_0.png (ver a versao Python)")
```

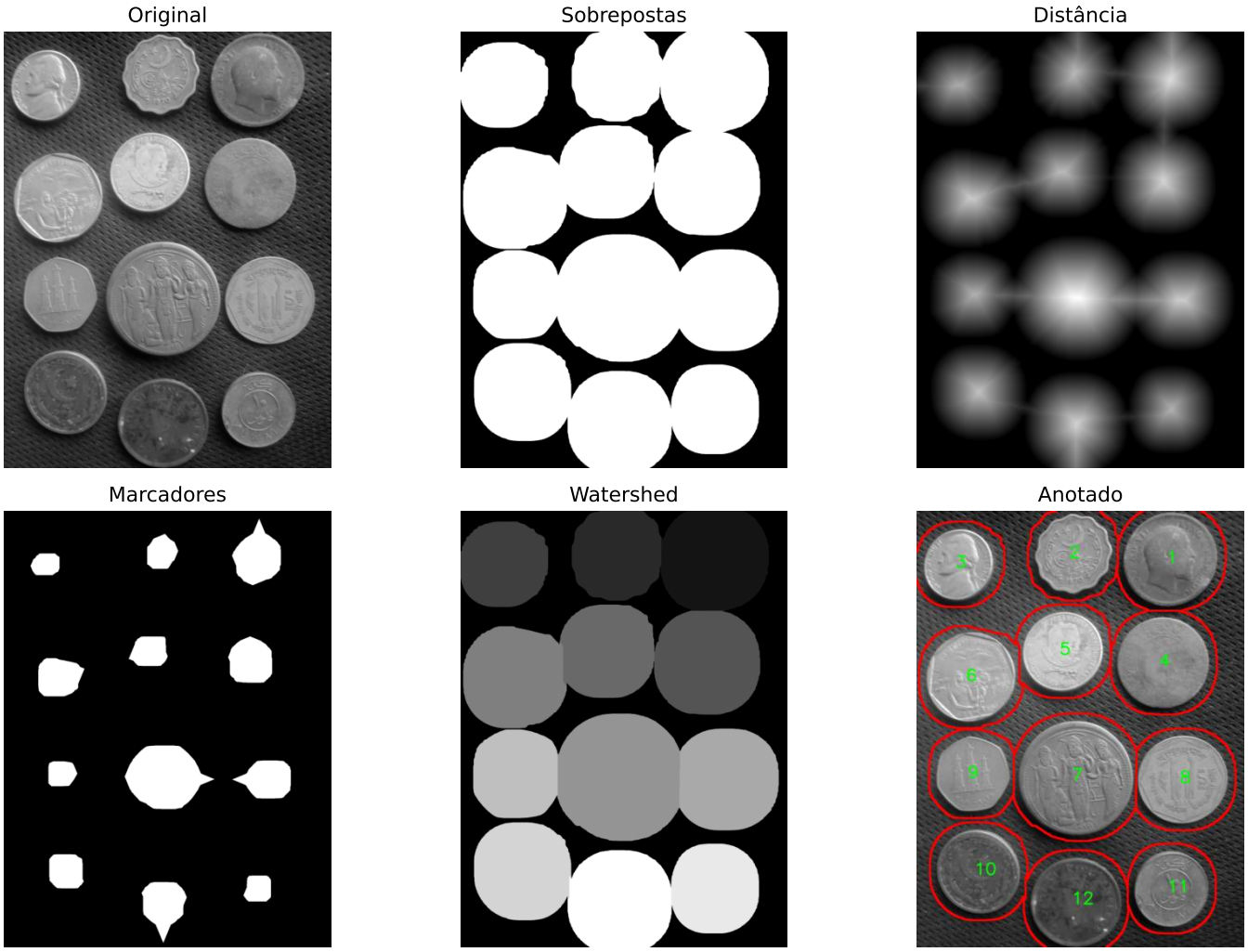


Figure 4.26: *Pipeline Watershed* para separação de moedas sobrepostas: da máscara binária dilatada até os contornos finais anotados sobre a imagem original.

4.5 Extraction de Composants et Descripteurs de Forme

Après la segmentation et le raffinement morphologique, l'étape suivante consiste à identifier individuellement chaque objet présent dans l'image et à extraire ses propriétés géométriques. Cette étape est fondamentale pour les tâches de mesure, de classification et de reconnaissance de formes.

Un **composant connexe** est un ensemble maximal de pixels appartenant à l'objet qui restent mutuellement connectés selon une relation de connectivité préalablement définie (connectivité 4 ou 8). Après l'étiquetage, chaque composant reçoit un identifiant unique, permettant d'analyser ses caractéristiques individuellement.

OpenCV propose deux approches complémentaires pour cette analyse, résumées dans la Table 4.7.

Tableau 4.7: Comparaison entre les approches basées sur les composants connexes et sur les contours.

	`connectedComponentsWithStats`	`findContours`
Retourne	étiquette par pixel et statistiques par composant	séquence de points décrivant le contour
Descripteurs directs	aire, *boîte englobante* et centroïde	périmètre, forme et hiérarchie
Objets en contact	tend à fusionner les régions connectées	tend à produire un unique contour externe
Usage typique	comptage, filtrage et étiquetage	analyse géométrique et descripteurs de forme

4.5.1 Étiquetage et statistiques de composantes

`mm::label0` attribue une étiquette à chaque composante connexe. À partir de l'image étiquetée, on obtient, par balayage, la **surface** (comptage des pixels) et la **boîte englobante** de chaque objet (Figure 4.27). Le `connectedComponentsWithStats` d'OpenCV, qui renvoie ces statistiques prêtes à l'emploi, et les annotations colorées sur l'image se trouvent dans le parcours Python.

```
%%writefile tmp/fig_04_componentes.cpp
#define MM_OUT "tmp/fig_04_componentes.png"
//| label: fig-04-componentes
//| fig-cap: "Componentes conexos extraídos após o *pipeline* CLAHE → Otsu → limpeza
morfológica. Cada objeto é colorido com cor distinta e anotado com sua área em pixels."
//| echo: true
//| output: true
#include <opencv2/opencv.hpp>
#include <iostream>
#include <iomanip>
#include <vector>
#include <string>
#include "morph.hpp"
#include <filesystem>

int main() {
// [pdi:state-io] auto-generated - do not edit by hand
mm::Image img_coins_gray = mm::_read_state("tmp/state/img_coins_gray_8.png");
mm::Image img_final = mm::_read_state("tmp/state/img_final_55.png");
// [pdi:state-io:end]

    // 1. Rotulagem e estatísticas
    cv::Mat labels;
    cv::Mat stats;
    cv::Mat centroids;
    cv::Mat img_final_cv(img_final.h, img_final.w, CV_8UC1, img_final.data.data());
    int n = cv::connectedComponentsWithStats(img_final_cv, labels, stats, centroids, 8,
    CV_32S);

    // 2. Coloração dos componentes - paleta FIXA (gerada uma vez com
    // np.random.seed(4)) para a trilha C++ reproduzir cor a cor sem
    // depender do gerador do numpy. Se n exceder len(PALETA), cicla.
    std::vector<std::vector<int>> PALETA = {
        {0, 0, 0}, {224, 82, 193}, {233, 155, 73}, {190, 247, 244},
```

```
        {103, 94, 179}, {51, 154, 59}, {137, 96, 232}, {250, 243, 205},
        {100, 141, 208}, {228, 187, 163}, {202, 108, 214}, {131, 105, 104},
        {86, 153, 81}};

    mm::Image img_colored(img_coins_gray.h, img_coins_gray.w, 3);
    // colors[0] = [0, 0, 0]  // fundo preto (already zero-initialized)
    for (int y = 0; y < img_final.h; y++) {
        for (int x = 0; x < img_final.w; x++) {
            int label = labels.at<int>(y, x);
            int idx = label % (int)PALETA.size();
            img_colored.at(y, x, 0) = PALETA[idx][0];
            img_colored.at(y, x, 1) = PALETA[idx][1];
            img_colored.at(y, x, 2) = PALETA[idx][2];
        }
    }
    img_colored.at(0, 0, 0) = 0;
    img_colored.at(0, 0, 1) = 0;
    img_colored.at(0, 0, 2) = 0;

    mm::Image img_annotated = img_colored;
    cv::Mat img_annotated_cv(img_annotated.h, img_annotated.w, CV_8UC3,
img_annotated.data.data());

    // 3. Tabela de descritores
    std::cout << "Componentes detectados (excluindo fundo): " << n - 1 << "\n";
    std::cout << "\n" << std::setw(4) << "ID" << std::setw(8) << "Área"
              << std::setw(6) << "cx" << std::setw(6) << "cy"
              << std::setw(6) << "w" << std::setw(6) << "h" << "\n";
    std::cout << std::string(42, '-') << "\n";
    for (int i = 1; i < n; i++) {
        int area = stats.at<int>(i, cv::CC_STAT_AREA);
        int cx = (int)centroids.at<double>(i, 0);
        int cy = (int)centroids.at<double>(i, 1);
        int w = stats.at<int>(i, cv::CC_STAT_WIDTH);
        int h = stats.at<int>(i, cv::CC_STAT_HEIGHT);
        std::cout << std::setw(4) << i << std::setw(8) << area
                  << std::setw(6) << cx << std::setw(6) << cy
                  << std::setw(6) << w << std::setw(6) << h << "\n";
        for (const auto& [cor, esp] : std::vector<std::pair<cv::Scalar, int>>{
            {cv::Scalar(0, 0, 0), 10}, {cv::Scalar(255, 0, 0), 5}}) {
            cv::putText(img_annotated_cv, std::to_string(i) + ": " + std::to_string(area),
                        cv::Point(cx - 150, cy + 18),
                        cv::FONT_HERSHEY_SIMPLEX, 2.0, cor, esp, cv::LINE_AA);
        }
    }

    mm::show(
        std::vector<mm::Image>{img_coins_gray, img_final, img_colored, img_annotated},
        MM_OUT,
        std::vector<std::string>{"Original", "Segmentação Final", "Componentes Conexos",
"Áreas Anotadas"},
        4);

// [pdi:panel-io] auto-generated - do not edit by hand
std::filesystem::create_directories("tmp");
mm::write(img_coins_gray, "tmp/fig_04_componentes_0.png");
mm::write(img_final, "tmp/fig_04_componentes_1.png");
mm::write(img_colored, "tmp/fig_04_componentes_2.png");
mm::write(img_annotated, "tmp/fig_04_componentes_3.png");
// [pdi:panel-io:end]
return 0;
```

```
}
```

```
Overwriting tmp/fig_04_componentes.cpp
```

```
!g++ -I. -std=c++17 -DMM_USE_OPENCV -I/usr/include/opencv4 tmp/fig_04_componentes.cpp -o
tmp/fig_04_componentes -lopencv_imgproc -lopencv_imgcodecs -lopencv_core \
  && ./tmp/fig_04_componentes \
  && test -f "tmp/fig_04_componentes.png" \
  || echo "  mm::show não gravou tmp/fig_04_componentes.png"
```

```
Componentes detectados (excluindo fundo): 12

  ID   Área    cx    cy     w     h
-----------------------------------------
   1  231969  1484   280   553   542
   2  158967   917   250   453   468
   3  139229   250   312   433   419
   4  175550   857   821   474   465
   5  222882  1442   889   539   531
   6  210043   316   977   527   516
   7  343376   934  1559   667   665
   8  213641  1555  1574   530   515
   9  147880   328  1543   432   438
  10  187280   363  2111   487   492
  11  150110  1487  2215   433   444
  12  215387   932  2292   528   522
[1] Original
[2] Segmentação Final
[3] Componentes Conexos
[4] Áreas Anotadas
```

```
try:
    mm.show(
        [
            mm.read("tmp/fig_04_componentes_0.png"),
            mm.read("tmp/fig_04_componentes_1.png"),
            mm.read("tmp/fig_04_componentes_2.png"),
            mm.read("tmp/fig_04_componentes_3.png"),
        ],
        titles=[
            'Original',
            'Segmentação Final',
            'Componentes Conexos',
            'Áreas Anotadas',
        ],
        cols=4,
        figsize=(18, 6),
    )
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + "
    tmp/fig_04_componentes_0.png (ver a versao Python)")
```

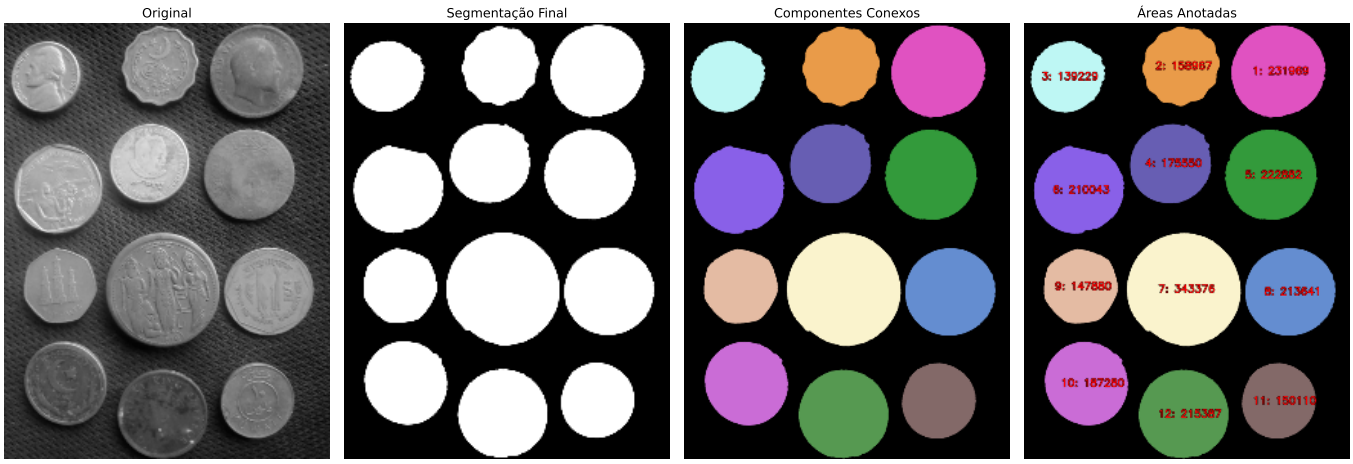


Figure 4.27: Componentes conexos extraídos após o *pipeline* CLAHE → Otsu → limpeza morfológica. Cada objeto é colorido com cor distinta e anotado com sua área em pixels.

4.5.2 Descripteurs de forme

Les descripteurs basés sur le **contour vectoriel** — périmètre (`arcLength`), aire du polygone (`contourArea`), circularité, moments et approximation polygonale (`approxPolyDP`) — dépendent du suivi de frontière (`findContours`), absent de la `morph.hpp`. Ils ne restent que dans le parcours Python. Dans le parcours C++, le **gradient morphologique** (`mm::gradm`) met en évidence le contour de chaque objet (Figure 4.28).

```
%%writefile tmp/fig_04_contornos.cpp
#define MM_OUT "tmp/fig_04_contornos.png"
// Compile: g++ -std=c++17 -o programa programa.cpp -I. -I/usr/include/opencv4 $(pkg-config
--cflags --libs opencv4) -DMM_USE_OPENCV
#include "morph.hpp"
#include <opencv2/opencv.hpp>
#include <iostream>
#include <vector>
#include <cmath>
#include <iomanip>
#include <string>
#include <filesystem>

int main() {
// [pdi:state-io] auto-generated - do not edit by hand
mm::Image img_coins_gray = mm::_read_state("tmp/state/img_coins_gray_8.png");
mm::Image img_final = mm::_read_state("tmp/state/img_final_55.png");
// [pdi:state-io:end]

    //| label: fig-04-contornos
    //| fig-cap: "Contornos extraídos com *findContours*. Cada moeda é anotada com sua
    circularidade - valores próximos de 1 confirmam forma circular."
    //| echo: true
    //| output: true

    // Convertir mm::Image en cv::Mat pour findContours
    cv::Mat img_final_mat(img_final.h, img_final.w, CV_8UC1, img_final.data.data());
    std::vector<std::vector<cv::Point>> contornos;
    cv::findContours(img_final_mat, contornos, cv::RETR_EXTERNAL, cv::CHAIN_APPROX_SIMPLE);

    // Convertir l'image en couleur (BGR) pour l'affichage
    mm::Image img_coins_rgb(img_coins_gray.h, img_coins_gray.w, 3);
    for (int y = 0; y < img_coins_gray.h; y++) {
        for (int x = 0; x < img_coins_gray.w; x++) {
```

```
            unsigned char v = img_coins_gray.at(y, x);
            img_coins_rgb.at(y, x, 0) = v;
            img_coins_rgb.at(y, x, 1) = v;
            img_coins_rgb.at(y, x, 2) = v;
        }
    }
    mm::Image img_contornos = img_coins_rgb;
    mm::Image img_circulares = img_contornos;

    std::cout << "Contornos detectados: " << contornos.size() << "\n";
    std::cout << std::setw(4) << "ID" << std::setw(8) << "Área"
              << std::setw(10) << "Perímetro" << std::setw(14) << "Circularidade" << "\n";
    std::cout << std::string(42, '-') << "\n";

    for (size_t i = 0; i < contornos.size(); i++) {
        const auto& cnt = contornos[i];
        int id = i + 1;

        double area = cv::contourArea(cnt);
        double perim = cv::arcLength(cnt, true);
        double circ = (perim > 0) ? (4 * M_PI * area / (perim * perim)) : 0;
        cv::Moments M = cv::moments(cnt);
        int cx = (M.m00 > 0) ? static_cast<int>(M.m10 / M.m00) : 0;
        int cy = (M.m00 > 0) ? static_cast<int>(M.m01 / M.m00) : 0;

        std::cout << std::setw(4) << id << std::setw(8) << static_cast<int>(area)
                  << std::setw(10) << std::fixed << std::setprecision(1) << perim
                  << std::setw(14) << std::setprecision(3) << circ << "\n";

        // Dessiner les contours sur l'image en couleur
        std::vector<std::vector<cv::Point>> contour_list = {cnt};
        cv::Mat img_contornos_mat(img_contornos.h, img_contornos.w, CV_8UC3,
img_contornos.data.data());
        cv::drawContours(img_contornos_mat, contour_list, -1, cv::Scalar(0, 255, 0), 3);

        cv::Mat img_circulares_mat(img_circulares.h, img_circulares.w, CV_8UC3,
img_circulares.data.data());
        cv::drawContours(img_circulares_mat, contour_list, -1, cv::Scalar(0, 255, 0), 3);

        // Texte avec bordure noire pour lisibilité
        cv::putText(img_circulares_mat, cv::format("%.2f", circ),
                    cv::Point(cx - 80, cy + 15),
                    cv::FONT_HERSHEY_SIMPLEX, 3.6, cv::Scalar(0, 0, 0), 8, cv::LINE_AA);
        cv::putText(img_circulares_mat, cv::format("%.2f", circ),
                    cv::Point(cx - 80, cy + 15),
                    cv::FONT_HERSHEY_SIMPLEX, 3.6, cv::Scalar(255, 0, 0), 3, cv::LINE_AA);
    }

    mm::show(std::vector<mm::Image>{img_final, img_contornos, img_circulares},
             MM_OUT,
             std::vector<std::string>{"Segmentação Final", "Contornos", "Circularidade"},
             3);


// [pdi:panel-io] auto-generated - do not edit by hand
std::filesystem::create_directories("tmp");
mm::write(img_final, "tmp/fig_04_contornos_0.png");
mm::write(img_contornos, "tmp/fig_04_contornos_1.png");
mm::write(img_circulares, "tmp/fig_04_contornos_2.png");
// [pdi:panel-io:end]
return 0;
```

```
}
```

```
Overwriting tmp/fig_04_contornos.cpp
```

```
!g++ -I. -std=c++17 -DMM_USE_OPENCV -I/usr/include/opencv4 tmp/fig_04_contornos.cpp -o
tmp/fig_04_contornos -lopencv_imgproc -lopencv_imgcodecs -lopencv_core \
  && ./tmp/fig_04_contornos \
  && test -f "tmp/fig_04_contornos.png" \
  || echo "  mm::show não gravou tmp/fig_04_contornos.png"
```

```
Contornos detectados: 12
  ID   ÁreaPerímetro Circularidade
-----------------------------------------
   1  214626    1769.6         0.861
   2  149480    1465.1         0.875
   3  186570    1654.9         0.856
   4  147252    1458.6         0.870
   5  212885    1756.3         0.867
   6  342424    2232.5         0.863
   7  209282    1767.7         0.842
   8  222108    1809.7         0.852
   9  174854    1616.4         0.841
  10  138602    1471.5         0.804
  11  158306    1538.7         0.840
  12  231181    1847.4         0.851
[1] Segmentação Final
[2] Contornos
[3] Circularidade
```

```
try:
    mm.show(
        [
            mm.read("tmp/fig_04_contornos_0.png"),
            mm.read("tmp/fig_04_contornos_1.png"),
            mm.read("tmp/fig_04_contornos_2.png"),
        ],
        titles=[
            'Segmentação Final',
            'Contornos',
            'Circularidade',
        ],
        cols=3,
        figsize=(18, 6),
    )
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + " tmp/fig_04_contornos_0.png
    (ver a versao Python)")
```

Figure 4.28: Contornos extraídos com *findContours*. Cada moeda é anotada com sua circularidade — valores próximos de 1 confirmam forma circular.

4.5.3 Connexion avec la détection d'objets moderne

Les descripteurs extraits dans les sections précédentes — notamment les *bounding boxes*, les centroïdes, les aires et les mesures de forme — établissent un pont naturel entre la segmentation morphologique classique et les systèmes modernes de détection d'objets. Bien que les techniques étudiées dans ce chapitre utilisent des opérations sur les pixels et les régions segmentées, bon nombre des représentations produites sont directement compatibles avec les formats employés dans les modèles contemporains de vision par ordinateur.

Les détecteurs basés sur l'apprentissage profond, tels que la famille YOLO (*You Only Look Once*) (REDMON, 2016), opèrent directement sur des images en couleur et produisent, pour chaque objet détecté, une *bounding box* décrite par le centre (cx, cy) et les dimensions (w, h), ainsi qu'une classe et un score de confiance. Cette représentation partage la même structure géométrique de base obtenue par `connectedComponentsWithStats`, bien qu'elle soit produite par un modèle appris et non par une segmentation explicite.

La Figure 4.29 illustre comment les *bounding boxes* obtenues par morphologie peuvent être exportées au format YOLO pour composer des ensembles de données utilisés lors de l'entraînement ou de l'évaluation de détecteurs.

```
%%writefile tmp/fig_04_bbox.cpp
#define MM_OUT "tmp/fig_04_bbox.png"
//| label: fig-04-bbox
//| fig-cap: "*Bounding boxes* derivadas dos componentes conexos sobrepostas à imagem
original. As anotações são exportadas no formato YOLO (*classe cx cy w h*), com coordenadas
normalizadas para o intervalo [0,1]."
//| echo: true
//| output: true
#include <opencv2/opencv.hpp>
#include <cstdio>
#include <string>
#include <vector>
#include <fstream>
#include <iostream>
#include <iomanip>
#include <sstream>
#include "morph.hpp"
#include <filesystem>

// Recalcula rótulos/estatísticas a partir da segmentação final
int main() {
```

```
// [pdi:state-io] auto-generated - do not edit by hand
mm::Image img_coins_gray = mm::_read_state("tmp/state/img_coins_gray_8.png");
mm::Image img_final = mm::_read_state("tmp/state/img_final_55.png");
// [pdi:state-io:end]

    // Recalcula rótulos/estatísticas a partir da segmentação final
    cv::Mat img_final_mat(img_final.h, img_final.w, CV_8UC1, img_final.data.data());
    cv::Mat labels, stats, centroids;
    int n = cv::connectedComponentsWithStats(img_final_mat, labels, stats, centroids, 8);

    int H_img = img_coins_gray.h;
    int W_img = img_coins_gray.w;

    cv::Mat img_coins_gray_mat(img_coins_gray.h, img_coins_gray.w, CV_8UC1,
img_coins_gray.data.data());
    cv::Mat img_bbox;
    cv::cvtColor(img_coins_gray_mat, img_bbox, cv::COLOR_GRAY2BGR);

    const int CLASSE = 0;   // 0 = moeda (única categoria neste exemplo)

    std::cout << std::setw(4) << "cls" << std::setw(9) << "cx_n" << std::setw(9) << "cy_n"
              << std::setw(9) << "w_n" << std::setw(9) << "h_n" << "  ← formato YOLO\n";
    std::cout << std::string(54, '-') << "\n";

    std::vector<std::string> yolo_linhas;
    for (int i = 1; i < n; i++) {
        int x0 = stats.at<int>(i, cv::CC_STAT_LEFT);
        int y0 = stats.at<int>(i, cv::CC_STAT_TOP);
        int w = stats.at<int>(i, cv::CC_STAT_WIDTH);
        int h = stats.at<int>(i, cv::CC_STAT_HEIGHT);

        double cx_n = (x0 + w / 2.0) / W_img;
        double cy_n = (y0 + h / 2.0) / H_img;
        double w_n = w / (double)W_img;
        double h_n = h / (double)H_img;

        std::ostringstream linha;
        linha << CLASSE << " " << std::fixed << std::setprecision(4)
              << cx_n << " " << cy_n << " " << w_n << " " << h_n;
        yolo_linhas.push_back(linha.str());

        std::cout << std::setw(4) << CLASSE << std::setw(9) << std::fixed <<
std::setprecision(4)
                  << cx_n << std::setw(9) << cy_n << std::setw(9) << w_n << std::setw(9) <<
                  h_n << "\n";

        cv::rectangle(img_bbox, cv::Point(x0, y0), cv::Point(x0 + w, y0 + h), cv::Scalar(0,
255, 0), 4);
        cv::putText(img_bbox, "moeda", cv::Point(x0 + 8, y0 + 60),
                    cv::FONT_HERSHEY_SIMPLEX, 3.8, cv::Scalar(255, 0, 0), 5, cv::LINE_AA);
    }

    // Exportar arquivo de anotação no formato YOLO
    std::ofstream f("moedas.txt");
    for (size_t i = 0; i < yolo_linhas.size(); i++) {
        f << yolo_linhas[i];
        if (i < yolo_linhas.size() - 1) f << "\n";
    }
    f.close();
    std::cout << "\nAnotação salva em moedas.txt\n";
```

```
    // Converter de volta para mm::Image para exibição
    mm::Image img_bbox_mm(img_bbox.rows, img_bbox.cols, img_bbox.channels());
    std::memcpy(img_bbox_mm.data.data(), img_bbox.data, img_bbox_mm.data.size());

    mm::show(
        std::vector<mm::Image>{img_coins_gray, img_final, img_bbox_mm},
        MM_OUT,
        std::vector<std::string>{"Original", "Segmentação Final", "Bounding Boxes (formato
YOLO)"},
        3
    );

// [pdi:panel-io] auto-generated - do not edit by hand
std::filesystem::create_directories("tmp");
mm::write(img_coins_gray, "tmp/fig_04_bbox_0.png");
mm::write(img_final, "tmp/fig_04_bbox_1.png");
mm::write(img_bbox, "tmp/fig_04_bbox_2.png");
// [pdi:panel-io:end]
return 0;
}
```

```
Overwriting tmp/fig_04_bbox.cpp
```

```
!g++ -I. -std=c++17 -DMM_USE_OPENCV -I/usr/include/opencv4 tmp/fig_04_bbox.cpp -o
tmp/fig_04_bbox -lopencv_imgproc -lopencv_imgcodecs -lopencv_core \
  && ./tmp/fig_04_bbox \
  && test -f "tmp/fig_04_bbox.png" \
  || echo "  mm::show não gravou tmp/fig_04_bbox.png"
```

```
 cls     cx_n     cy_n      w_n      h_n  ← formato YOLO
------------------------------------------------------
   0   0.7753   0.1102   0.2880   0.2117
   0   0.4784   0.0984   0.2359   0.1828
   0   0.1331   0.1225   0.2255   0.1637
   0   0.4464   0.3217   0.2469   0.1816
   0   0.7523   0.3471   0.2807   0.2074
   0   0.1664   0.3812   0.2745   0.2016
   0   0.4862   0.6104   0.3474   0.2598
   0   0.8115   0.6154   0.2760   0.2012
   0   0.1724   0.6023   0.2250   0.1711
   0   0.1893   0.8258   0.2536   0.1922
   0   0.7763   0.8652   0.2255   0.1734
   0   0.4854   0.8953   0.2750   0.2039

Anotação salva em moedas.txt
[1] Original
[2] Segmentação Final
[3] Bounding Boxes (formato YOLO)
```

```
try:
    mm.show(
        [
            mm.read("tmp/fig_04_bbox_0.png"),
            mm.read("tmp/fig_04_bbox_1.png"),
            mm.read("tmp/fig_04_bbox_2.png"),
        ],
        titles=[
            'Original',
            'Segmentação Final',
            'Bounding Boxes (formato YOLO)',
```

```
        ],
        cols=3,
        figsize=(18, 6),
    )
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + " tmp/fig_04_bbox_0.png (ver
    a versao Python)")
```

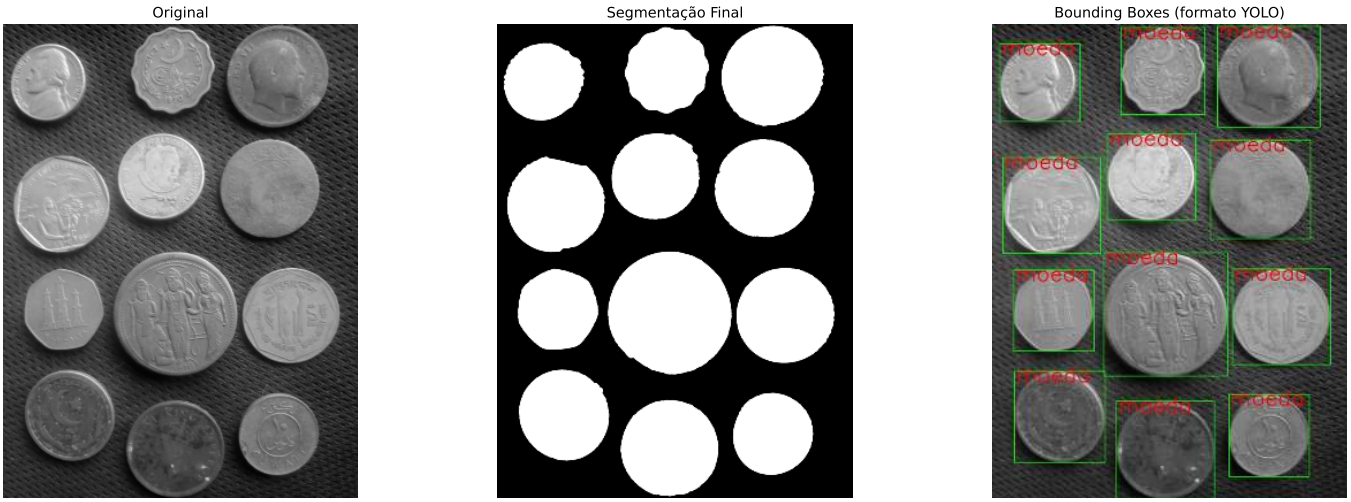


Figure 4.29: *Bounding boxes* derivadas dos componentes conexos sobrepostas à imagem original. As anotações são exportadas no formato YOLO (*classe cx cy w h*), com coordenadas normalizadas para o intervalo [0,1].

Le format YOLO stocke chaque objet dans une ligne contenant cinq champs :

```
classe cx cy w h
```

où (cx, cy) représente le centre de la *bounding box* et (w, h) ses dimensions. Toutes les valeurs géométriques sont normalisées dans l'intervalle $[0, 1]$ par rapport à la largeur et à la hauteur de l'image. La **classe** est un identifiant entier associé à une catégorie définie par l'ensemble de données (par exemple, `0 → pièce de monnaie`). Lorsqu'il y a plusieurs catégories — comme la pièce d'or (`0`), la pièce d'argent (`1`) et le disque en plastique (`2`) — il suffit d'attribuer l'identifiant correspondant à chaque objet avant l'exportation, en maintenant exactement le même format d'annotation. Dans l'exemple précédent, ces informations ont été stockées dans le fichier `moedas.txt`.

Le flux présenté dans ce chapitre — segmentation → étiquetage → extraction de *bounding boxes* — correspond conceptuellement à l'étape d'**annotation** (*labeling*) employée dans la construction d'ensembles d'entraînement pour les détecteurs modernes. Des outils spécialisés, comme Label Studio et Roboflow, automatisent ce processus sur des images complexes, mais la logique fondamentale reste la même : associer à chaque objet une région d'intérêt et une classe. Dans des scénarios contrôlés, avec un arrière-plan uniforme et des objets bien séparés, les techniques morphologiques peuvent même générer des annotations automatiquement ou servir de point de départ pour l'étiquetage manuel, réduisant considérablement l'effort de construction de l'ensemble de données. Dans des applications réelles plus complexes, cependant, la validation humaine reste nécessaire pour garantir la qualité des annotations.

i Évaluation : IoU (*Intersection over Union*)

Une méthode simple pour évaluer la qualité d'une segmentation consiste à la comparer à un masque de référence (*ground truth*). La métrique la plus couramment utilisée à cette fin est la **IoU** (*Intersection over Union*) :

$$\text{IoU} = \frac{|A \cap B|}{|A \cup B|} \tag{4.16}$$

où A représente la segmentation produite par l'algorithme et B la segmentation de référence.
La valeur de l'IoU varie entre 0 et 1. Plus la valeur est élevée, plus la superposition entre les masques est importante. Une IoU égale à 1 indique une correspondance parfaite entre la segmentation obtenue et la référence.
La même métrique est également largement utilisée en détection d'objets, où elle est appliquée aux *bounding boxes* prédites et annotées. Dans ce domaine, des valeurs d'IoU supérieures à 0,5 sont souvent adoptées comme critère minimum pour considérer une détection comme correcte.

Au-delà de la morphologie

Les techniques étudiées dans ce chapitre segmentent les objets en exploitant la connectivité spatiale, les opérations morphologiques et le relief topographique. Il existe cependant des approches alternatives basées sur le regroupement de caractéristiques, comme l'algorithme *k-means*, les modèles de mélange gaussien (GMM) et des méthodes plus récentes fondées sur l'apprentissage profond. Ces techniques seront reprises dans la Partie II de l'ouvrage, dédiée à la vision par ordinateur.

4.6 Résumé

Dans ce chapitre, les principales techniques de segmentation et de morphologie mathématique ont été présentées, concluant l'étude du TNI dans le domaine spatial.

- **Prétraitement et seuillage :** La combinaison entre l'égalisation adaptative CLAHE et la méthode d'Otsu s'est révélée efficace pour induire une séparation bimodale dans l'histogramme et simplifier la binarisation d'images à éclairage non uniforme.
- **Érosion et dilatation :** Opérateurs morphologiques fondamentaux basés sur la recherche de minima et de maxima locaux dans un voisinage défini par l'élément structurant B. Ce sont des opérateurs duaux par complémentation, implémentés au moyen des fonctions `mm::ero` et `mm::dil`.
- **Ouverture et fermeture :** Compositions d'érosion et de dilatation permettant de supprimer les bruits, de lisser les contours et de combler les petites lacunes, tout en préservant la structure globale des objets.
- **Reconstruction morphologique :** Processus géodésique itératif qui propage un marqueur à l'intérieur des limites imposées par un masque, constituant la base d'opérateurs tels que `mm::clohole` et `mm::edgeoff`.
- **Pipeline de nettoyage binaire :** Flux consolidé composé de CLAHE → Otsu → ouverture → `mm::clohole` → ouverture restreinte → `mm::edgeoff`, produisant des masques adaptés à l'analyse quantitative.
- **Morphologie en niveaux de gris :** Extension algébrique fondée sur des minima et maxima pondérés, permettant des opérateurs tels que le gradient morphologique et les filtres *top-hat* pour le rehaussement de structures locales.
- **Transformée de distance et Watershed :** La Transformée de distance a permis de générer des marqueurs automatiques pour l'algorithme *watershed*, rendant possible la séparation d'objets adjacents ou partiellement superposés.
- **Composantes connexes et descripteurs :** L'étiquetage des régions (`mm::label0`) et l'extraction des contours ont permis de calculer des descripteurs géométriques tels que l'aire, le centroïde, le périmètre, la circularité et les *bounding boxes*.
- **Connexion avec la vision par ordinateur moderne :** Les *bounding boxes* extraites par morphologie ont été exportées au format YOLO, mettant en évidence le lien entre les techniques classiques de segmentation et les systèmes modernes de détection d'objets.

Le chapitre 5 introduira les techniques de traitement dans le **domaine fréquentiel**, abordant la **Transformée**

de Fourier, le filtrage spectral et les fondements de la **compression d'images**, y compris la DCT, JPEG et les *wavelets*.

4.7 🤖 Utilisation de Gemini Notebook comme tuteur complémentaire

Dans cette édition, l'utilisation de **Gemini Notebook** est encouragée comme outil d'apprentissage complémentaire. Fondé sur l'intelligence artificielle, le système utilise exclusivement les documents fournis par l'auteur comme source de connaissances, produisant des réponses alignées sur le contenu et l'approche adoptés tout au long de ce chapitre.

🎓 Étudiez avec le tuteur intelligent

🚀 ACCÉDER À GEMINI NOTEBOOK : CHAPITRE 04

🌐 Langue et langage de programmation

Le projet de ce chapitre dans Gemini Notebook a été construit uniquement avec le texte en **portugais** et les exemples de code en **Python**. Si vous étudiez à partir de l'édition en anglais ou en français, ou si vous suivez le parcours en C++, les réponses du tuteur peuvent ne pas correspondre exactement à la version que vous lisez.

⚠️ Avertissement concernant le contenu généré par l'IA

Bien qu'il s'agisse d'un outil d'aide à l'étude précieux, Gemini Notebook peut éventuellement produire des réponses incomplètes, imprécises ou incorrectes. Il est recommandé de valider les informations en consultant le matériel du chapitre, les livres, les articles scientifiques et d'autres sources académiques fiables. Chaque fois que possible, exécutez et expérimentez les exemples pratiques présentés tout au long du texte afin de consolider la compréhension des concepts.

4.8 Liste d'exercices

1. **(10 %)** Implémentez manuellement le critère d'Otsu sans utiliser `mm::threshold`. Calculez la variance interclasse $\sigma_B^2(T)$ pour tous les seuils $T \in [0, 255]$ en utilisant `mm::hist`, identifiez le seuil optimal T^* et comparez le résultat avec la valeur obtenue par OpenCV. Tracez σ_B^2 en fonction de T et mettez en évidence le point de maximum.

2. **(15 %)** Appliquez un seuillage adaptatif avec des blocs de taille 11, 31 et 51 à une image présentant un éclairage non uniforme. Comparez les résultats avec le seuillage global d'Otsu et discutez des avantages et des limites de chaque approche.

3. **(15 %)** Exécutez le *pipeline watershed* sur l'image de pièces de monnaie en faisant varier le seuil appliqué à la transformée de distance ($0,3$, $0,5$ et $0,7$ fois la valeur maximale). Expliquez comment ce paramètre influence la génération des marqueurs, la séparation des objets adjacents et l'apparition de sur-segmentation.

4. **(15 %)** À l'aide de `mm::drawImg`, construisez une démonstration visuelle pas à pas de l'érosion d'une image binaire 7×7 avec un élément structurant carré de 3×3. Pour chaque position analysée, indiquez si l'élément structurant est entièrement contenu dans l'objet et justifiez la valeur attribuée au pixel de sortie.

5. **(15 %)** Démontrez expérimentalement la dualité entre l'érosion et la dilatation en vérifiant l'identité $(A \ominus B)^c = A^c \oplus \hat{B}$ à l'aide de `mm::ero`, `mm::dil` et `mm::bnot`. Calculez la différence pixel par pixel entre les deux membres de l'équation et présentez le résultat en utilisant `mm::histImg` ou une visualisation équivalente.

6. **(15 %)** Implémentez manuellement le gradient morphologique en utilisant uniquement `mm::ero` et `mm::dil`, en comparant le résultat avec `mm::gradm(img, B)`. Évaluez l'effet de différents éléments structurants (carré 3×3, disque 5×5 et ligne 1×9) sur la détection de contours.

7. **(15 %)** Construisez un *pipeline* complet pour le comptage et la classification de pièces de monnaie par taille (petite, moyenne et grande) en utilisant l'aire et la circularité comme descripteurs. Générez un masque de référence (*ground truth*) manuellement et calculez la métrique IoU (*Intersection over Union*) pour évaluer la qualité de la segmentation. Présentez les résultats dans un tableau et au moyen de visualisations produites avec `mm::show`.

Références du chapitre

La base théorique de ce chapitre s'appuie sur les ouvrages suivants :

- Gonzalez (2018) pour les concepts de segmentation, de seuillage d'Otsu, de transformée de distance, de *watershed*, de morphologie mathématique et de descripteurs de forme.
- Matheron (1975) et Serra (1982) pour la base théorique originale, la formulation algébrique et le développement de la morphologie mathématique.
- Szeliski (2022) pour la segmentation fondée sur les régions, l'étiquetage des composantes connexes, le *watershed* basé sur des marqueurs et l'évaluation de la segmentation au moyen de la métrique IoU.
- Bradski (2008) pour l'utilisation pratique de la bibliothèque OpenCV, y compris des fonctions telles que `mm::dist`, `mm::watershed`, `mm::label0` et l'extraction de contours.
- Redmon (2016) pour une introduction aux détecteurs modernes de la famille YOLO et leur lien avec les descripteurs géométriques comme les *bounding boxes* extraites par segmentation.
- Singh (2024) pour la construction de labyrinthes complexes basés sur des cycles hamiltoniens sur des mosaïques quasi-cristallines, utilisés comme exemple d'application de la transformée de distance géodésique et des algorithmes de recherche de chemins.
- Zampirolli (2025) pour l'implémentation des opérateurs morphologiques, des transformées géodésiques et la résolution de labyrinthes par propagation de distances dans des domaines restreints.

4.9 💻 Partie Pratique avec Exercices de Programmation

Cette liste transforme les concepts du Chapitre 4 en un parcours pratique de segmentation et de morphologie mathématique. Les exercices de programmation commencent par le seuillage et progressent jusqu'à l'étiquetage et les descripteurs de composants, toujours avec des matrices de petite taille afin que chaque pixel puisse être vérifié à la main.

> ❗ Règle commune des exercices de programmation morphologiques
>
> Dans les opérations avec voisinage, **ne faites pas de padding**. Pour chaque pixel, évaluez uniquement les positions de l'élément structurant qui se trouvent dans le domaine de l'image. C'est la même idée que les implémentations pédagogiques dans `morph.py`, comme `mm::dil0`, `mm::ero0`, `mm::dil1` et `mm::label0` : le voisinage est découpé par le domaine valide de l'image.

🎯 Objectif de ce carnet

Ce carnet permet de développer, valider, organiser et tester des solutions d' **Exercices de Programmation (EP)** dans des environnements interactifs, tels que Colab, avec les mêmes cas de test que Moodle, en y copiant uniquement au moment d'enregistrer la note officielle.

Téléchargement

Téléchargez `morph.py` et `testsuite.py` en exécutant la cellule ci-dessous :

```
import os, urllib.request

os.makedirs("tmp/state", exist_ok=True)  # artefacts de build du parcours C++ (.cpp, binaire,
PNGs)

url = "https://raw.githubusercontent.com/fzampirolli/pdi-vc/master/morph/config.py"
if not os.path.exists("config.py"):
    urllib.request.urlretrieve(url, "config.py")

# Le noyau est Python même dans le parcours C++ : `mm` (morph.py) est utilisé par les
# simulateurs, par l'affichage des figures que le binaire C++ génère et par
# l'état mm::Image entre cellules. cpp=True télécharge aussi le parcours compilé
# (morph.hpp + stb_image*.h), utilisé dans le #include des cellules %%writefile *.cpp.
import config
config.setup(testsuite=True, cpp=True)
from morph import mm
from testsuite import TestSuite
```

```
Environnement prêt. Morph : 1.1.9 | OpenCV : 5.0.0 | TestSuite : 1.1.2
```

Exécution des tests

Pour évaluer les tests, exécutez `TestSuite("EP04_01.extensão").run()` dans une nouvelle cellule, en remplaçant l'extension par celle du langage utilisé (`.py`, `.java`, `.c`, `.cpp`, `.js` ou `.r`). Le système télécharge les cas de test depuis GitHub, exécute le programme et calcule automatiquement la note.

Pour tester directement du code Python, sans enregistrer de fichier, utilisez `run_code(codigo)` en passant le code comme *chaîne de caractères* dans une variable `codigo` :

```
codigo = """
from morph import mm
# ... votre code ici ...
"""
TestSuite("EP04_01").run_code(codigo)
```

4.9.1 EP04_01 Seuillage global par seuil fixe

Dans les **scanners de documents** et les **systèmes de lecture de codes-barres**, la première étape du traitement consiste toujours à séparer ce qui est « objet » (encre, texte, barres) de ce qui est « fond » (papier, emballage). Le **seuillage global** fait exactement cela : il compare chaque pixel à un seuil unique T et décide, en temps réel, s'il appartient à la classe claire ou à la classe sombre. C'est l'opérateur de segmentation le plus simple — et pourtant, il est à l'origine d'une grande partie des *pipelines* industriels d'inspection visuelle. Voir la simulation de cet EP dans Figure 4.30.

4.9.1.1 📋 Directives d'implémentation

1. **Dimensions :** Lire les entiers L (lignes) et C (colonnes).
2. **Seuil :** Lire l'entier T (seuil de décision).
3. **Données :** Lire les valeurs entières de la matrice originale ligne par ligne.
4. **Mappage :** Pour chaque pixel p, calculer la nouvelle valeur à l'aide de l'équation :

$$p' = \begin{cases} 255, & \text{si } p > T \\ 0, & \text{si } p \leq T \end{cases}$$

5. **Sortie :** Afficher la matrice binarisée avec les dimensions $L \times C$.

4.9.1.2 📌 Contraintes computationnelles

- **Binarisation :** La sortie contient **uniquement** les valeurs 0 ou 255.
- **Comparaison stricte :** Le critère utilise $> T$ (les pixels égaux à T deviennent du fond).
- **Type :** Le résultat final doit être entier.
- **Remarque :** Cet EP suit la convention d'OpenCV (`cv2.THRESL_BINARY`) : seuls les pixels avec une valeur **supérieure à** T deviennent blancs (`255`) ; les pixels avec une valeur **égale à** T restent noirs (`0`).

4.9.1.3 🧠 Fondement théorique

Paramètre	Type	Impact visuel
T **petit**	Entier	La plupart des pixels deviennent blancs
T **grand**	Entier	La plupart des pixels deviennent noirs
T **bien choisi**	Entier	Sépare nettement l'objet et le fond

4.9.1.4 📦 Spécification d'entrée et de sortie (VPL)

Entrée :

- Ligne 1 : Entier L.
- Ligne 2 : Entier C.
- Ligne 3 : Entier T.
- Lignes suivantes : Éléments entiers de la matrice originale.

Sortie :

- Matrice binarisée en L lignes et C colonnes, valeurs 0 ou 255 séparées par des espaces.

4.9.1.5 📌 Exemples

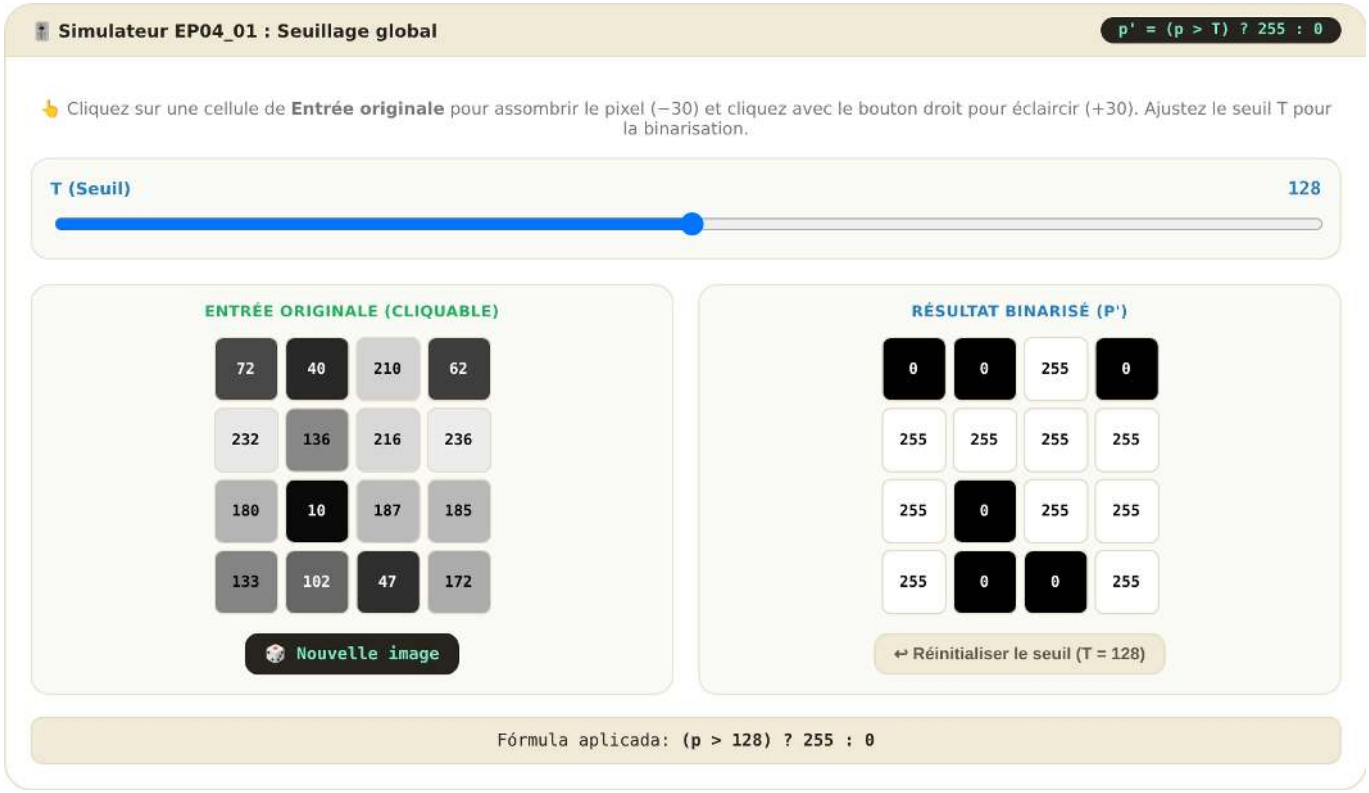
Entrée	Sortie	Remarque
2 4 100 0 99 100 180 255 30 120 80	0 0 0 255 255 0 255 0	$T = 100$: seuls les pixels avec une valeur supérieure à 100 deviennent blancs ; par conséquent, 99 et 100 deviennent noirs.
1 3 0 0 50 255	0 255 255	$T = 0$: seuls les pixels avec une valeur strictement supérieure à 0 deviennent blancs.

```
%%writefile EP04_01.cpp
// your solution
```

```
Overwriting EP04_01.cpp
```

```
TestSuite("EP04_01.cpp").run()
```

```
<IPython.core.display.HTML object>
```



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.fr/cap04/sim-ep0401-limiar.html

Figure 4.30: Simulateur EP04_01 : Seuillage global par seuil fixe (p' = (p > T) ? 255 : 0)

4.9.2 EP04_02 📊 Seuillage automatique d'Otsu

Choisir manuellement le seuil T fonctionne lorsque l'éclairage est stable, mais en **microscopie numérique** et en **inspection de lames de sang**, chaque échantillon présente un contraste différent — un seuil fixe échouerait d'une image à l'autre. La **méthode d'Otsu** résout ce problème en trouvant, de manière autonome, le seuil qui **maximise la séparation statistique** entre les deux classes de pixels, rendant la segmentation automatique et adaptative. Voir dans Figure 4.31 une simulation de cet EP.

4.9.2.1 📋 Directives d'implémentation

1. **Dimensions :** Lire les entiers L (lignes) et C (colonnes).
2. **Données :** Lire les valeurs entières de la matrice originale ligne par ligne.
3. **Histogramme :** Construire l'histogramme $h[i]$, $i = 0, \ldots, 255$, en comptant combien de pixels ont la valeur i.
4. **Recherche du seuil :** Pour chaque candidat T de 1 à 255, calculer la **variance inter-classes** :

$$\sigma_B^2(T) = \frac{n_0 \cdot n_1}{N^2} (m_0 - m_1)^2$$

   où n_0, n_1 sont les quantités de pixels ayant une valeur $< T$ et $\geq T$, m_0, m_1 sont leurs moyennes, et $N = L \times C$.
5. **Choix :** Le seuil optimal T^* est celui qui maximise $\sigma_B^2(T)$ (en cas d'égalité, conserver le **premier** trouvé).
6. **Application :** Binariser l'image en utilisant T^*, en appliquant :

$$p' = \begin{cases} 255, & \text{si } p > T^* \\ 0, & \text{si } p \leq T^* \end{cases}$$

4.9.2.2 📌 Contraintes computationnelles

- **Candidats valides :** Ignorer T qui laisse $n_0 = 0$ ou $n_1 = 0$ (classe vide).
- **Égalité :** Toujours conserver le **premier** T qui a atteint la valeur maximale de σ_B^2.
- **Type :** T^* et la matrice de sortie doivent être des entiers.
- **Convention OpenCV :** La binarisation suit `cv2.THRESL_BINARY` ; les pixels ayant une valeur exactement égale à T^* deviennent noirs.

4.9.2.3 🧠 Fondements théoriques

Concept	Signification	Impact
$\sigma_B^2(T)$ **élevée**	Classes bien séparées en T	T est un bon candidat comme seuil
Histogramme bimodal	Deux « pics » distincts	Otsu trouve le creux entre eux
Histogramme unimodal	Un seul « pic »	Otsu choisit toujours *un* T, mais la segmentation est peu fiable

4.9.2.4 📦 Spécification d'entrée et de sortie (VPL)

Entrée :

- Ligne 1 : Entier L.
- Ligne 2 : Entier C.
- Lignes suivantes : Éléments entiers de la matrice originale.

Sortie :

- Matrice binarisée en L lignes et C colonnes, valeurs 0 ou 255.

4.9.2.5 📌 Exemples

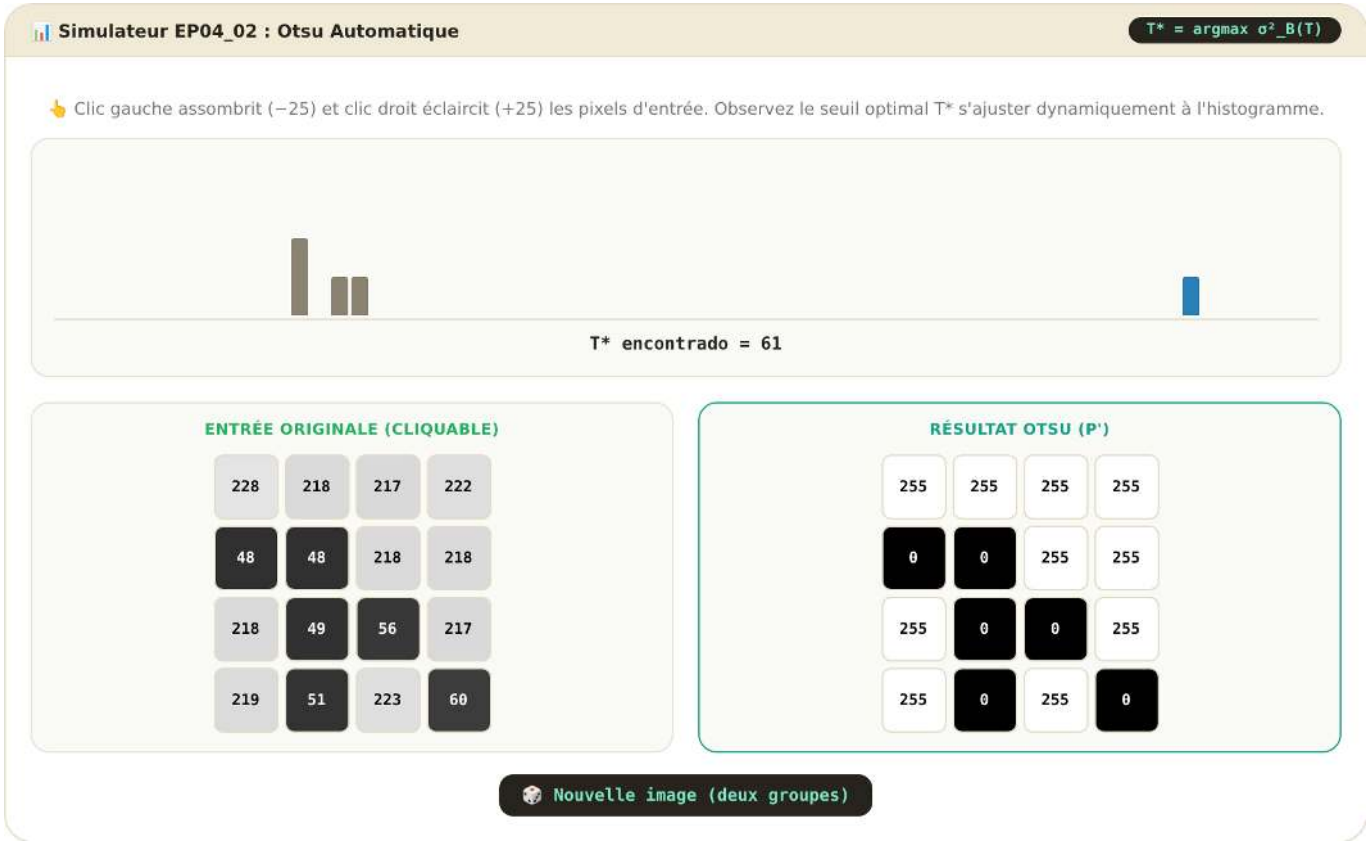
Entrée	Sortie	Observation
4 4 12 12 12 200 12 12 200 200 12 200 200 200 200 200 200 200	0 0 0 255 0 0 255 255 0 255 255 255 255 255 255 255	Histogramme bimodal net : 12 et 200
1 2 10 250	0 250	Deux valeurs seulement : T^* reste sur la plus grande

```
%%writefile EP04_02.cpp
// your solution
```

```
Overwriting EP04_02.cpp
```

```
TestSuite("EP04_02.cpp").run()
```

```
<IPython.core.display.HTML object>
```



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.fr/cap04/sim-ep0402-otsu.html

Figure 4.31: Simulateur EP04_02 : Seuillage automatique d'Otsu (T* = argmax ²_B(T))

4.9.3 EP04_03 🌱 Dilatation binaire plane (mm.dil0)

En **microscopie de particules** et en **OCR de plaques d'immatriculation usées**, les traits fins ou discontinus doivent être « épaissis » pour que la reconnaissance fonctionne. La **dilatation morphologique** fait exactement cela : elle étend les régions claires à l'aide d'un élément structurant B — la même opération implémentée dans `morph.py` comme `mm::dil0(f, B)`, utilisée lorsque B est **plat** (sans poids, uniquement 0/1). Voir dans Figure 4.32 une simulation de cet EP.

4.9.3.1 📋 Directives d'implémentation

1. **Dimensions de l'image :** Lire les entiers L (lignes) et C (colonnes) de f.
2. **Dimensions de B :** Lire les entiers L_B (lignes) et C_B (colonnes) de l'élément structurant.
3. **Élément structurant :** Lire la matrice B avec des valeurs 0 ou 1, ligne par ligne.
4. **Données :** Lire la matrice f (l'image originale), ligne par ligne.
5. **Réflexion :** Construire B_{ref}, la version de B réfléchie à 180° (lignes et colonnes inversées) — exactement comme le fait `mm::dil0` en interne.
6. **Voisinage sans padding :** Pour chaque pixel (y, x), parcourir les positions (by, bx) de B_{ref} centrées en (y, x), en utilisant le décalage

$$v_y = y + by + o_y, \quad v_x = x + bx + o_x, \quad o_y = -\frac{L_B}{2} + 0{,}5, \quad o_x = -\frac{C_B}{2} + 0{,}5$$

   Écarter tout (v_y, v_x) hors de $[0, L) \times [0, C)$ — **ne pas remplir avec des zéros**.
7. **Mappage :** Calculer chaque pixel de sortie comme le **maximum** entre $f(y, x)$ et tous les $f(v_y, v_x)$ valides dont la position correspondante dans B_{ref} vaut 1 :

$$g(y, x) = \max\left(f(y, x), \max_{\substack{(v_y, v_x) \text{ valide} \\ B_{ref}(by, bx)=1}} f(v_y, v_x)\right)$$

8. **Sortie :** Afficher la matrice g avec les dimensions $L \times C$.

4.9.3.2 📌 Contraintes computationnelles

- **Sans padding :** Ne jamais inventer de voisins hors de l'image ; n'utiliser que ceux qui existent réellement.
- **Réflexion obligatoire :** B doit être réfléchi avant d'être appliqué (c'est ce qui distingue `mm::dil0` d'une simple recherche de maximum).
- **Robustesse des bords :** Si aucune position valide de $B_{ref} = 1$ ne tombe dans le domaine pour un pixel donné, celui-ci **conserve sa valeur originale**.

4.9.3.3 🧠 Fondement théorique

Concept	Signification	Impact visuel
Dilatation	$g \geq f$ toujours (extensive)	Les régions claires croissent, les trous sombres rétrécissent
B **plus grand**	Voisinage plus large	Croissance plus agressive
Réflexion de B	$B_{ref}(y, x) = B(-y, -x)$	Garantit la définition formelle de Minkowski de la dilatation

4.9.3.4 📦 Spécification d'entrée et de sortie (VPL)

Entrée :

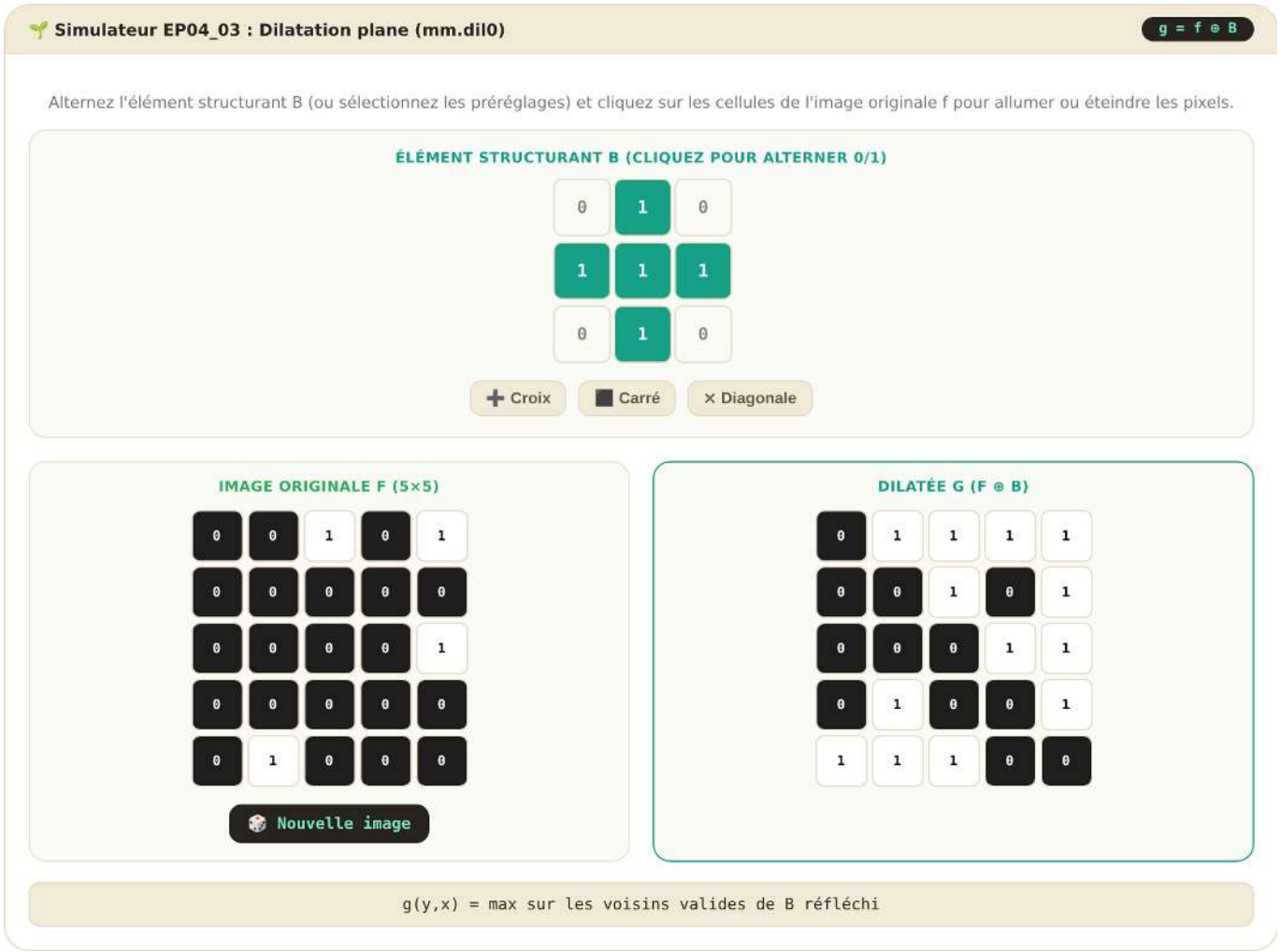
- Ligne 1 : entier L.
- Ligne 2 : entier C.
- Ligne 3 : entier L_B.
- Ligne 4 : entier C_B.
- Les L_B lignes suivantes : éléments entiers (0 ou 1) de la matrice B.
- Les L lignes suivantes : éléments entiers de la matrice f.

Sortie :

- Matrice g en L lignes et C colonnes, valeurs entières séparées par des espaces.

4.9.3.5 📌 Exemples

Entrée	Sortie	Observation
3 3 3 3 0 1 0 1 1 1 0 1 0 0 0 0 0 9 0 0 0 0	0 9 0 9 9 9 0 9 0	B en croix symétrique : point isolé se dilate en croix
1 4 1 3 1 1 1 10 200 5 80	200 200 200 80	B horizontal : chaque pixel « attire » le maximum des voisins de la ligne



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.fr/cap04/sim-ep0403-dilatacao.html

Figure 4.32: Simulateur EP04_03 : Dilatation binaire plane (g = f B)

```
%%writefile EP04_03.cpp
// your solution
```

```
Overwriting EP04_03.cpp
```

```
TestSuite("EP04_03.cpp").run()
```

```
<IPython.core.display.HTML object>
```

4.9.4 EP04_04 🪨 Érosion binaire plane (mm.ero0)

Si la dilatation épaissit, l'**érosion** affine. Dans les **systèmes de comptage de cellules**, elle est utilisée pour **séparer les cellules qui se touchent** : en « mangeant » les bords de chaque région, les connexions fines entre objets disparaissent avant même qu'un comptage soit effectué. Dans `morph.py`, c'est l'opération `mm::ero0(f, B)` — le **dual** exact de la dilatation, et la seule des deux qui **ne** réfléchit pas l'élément structurant. Voir dans Figure 4.33 une simulation de cet EP.

4.9.4.1 📋 Directives d'implémentation

1. **Dimensions de l'image :** Lire les entiers L (lignes) et C (colonnes) de f.
2. **Dimensions de B :** Lire les entiers L_B (lignes) et C_B (colonnes) de l'élément structurant.
3. **Élément structurant :** Lire la matrice B avec des valeurs 0 ou 1, ligne par ligne.
4. **Données :** Lire la matrice f (l'image originale), ligne par ligne.
5. **Voisinage sans padding (sans réflexion !) :** Pour chaque pixel (y, x), parcourir les positions (by, bx) de B **dans l'ordre original** (sans réfléchir), en utilisant le même décalage que dans l'EP04_03 :

$$v_y = y + by + o_y, \quad v_x = x + bx + o_x, \quad o_y = -\tfrac{L_B}{2} + 0{,}5, \quad o_x = -\tfrac{C_B}{2} + 0{,}5$$

Écarter tout (v_y, v_x) hors de $[0, L) \times [0, C)$. 6. **Mappage :** Calculer chaque pixel de sortie comme le **minimum** entre $f(y, x)$ et tous les $f(v_y, v_x)$ valides dont la position correspondante dans B vaut 1 :

$$g(y,x) = \min\left(f(y,x),\ \min_{\substack{(v_y,v_x)\ \text{valide} \\ B(by,bx)=1}} f(v_y,v_x)\right)$$

7. **Sortie :** Afficher la matrice g avec des dimensions $L \times C$.

4.9.4.2 📌 Contraintes de calcul

- **Sans réflexion :** Contrairement à la dilatation, B est utilisé **exactement comme lu** — réfléchir ici serait une erreur conceptuelle grave.
- **Sans padding :** Les voisins hors de l'image sont simplement ignorés, jamais traités comme 0.
- **Robustesse de bord :** Si aucune position valide de $B = 1$ ne tombe dans le domaine, le pixel conserve sa valeur originale.

4.9.4.3 🧠 Fondements théoriques

Concept	Signification	Impact visuel
Érosion	$g \leq f$ toujours (anti-extensive)	Les régions claires rétrécissent, le bruit ponctuel disparaît
Dualité	$\text{ero}(f, B) = -\text{dil}(-f, B_{ref})$	Érosion et dilatation sont des « miroirs » mathématiques
B plus grand	Érosion plus agressive	Les objets fins disparaissent complètement

4.9.4.4 📦 Spécification d'entrée et de sortie (VPL)

Entrée :

- Ligne 1 : Entier L.
- Ligne 2 : Entier C.
- Ligne 3 : Entier L_B.
- Ligne 4 : Entier C_B.
- L_B lignes suivantes : éléments entiers (0 ou 1) de la matrice B.
- L lignes suivantes : éléments entiers de la matrice f.

Sortie :

- Matrice g en L lignes et C colonnes, valeurs entières séparées par des espaces.

4.9.4.5 📌 Exemples

Entrée	Sortie	Observation
3 3 3 3 0 1 0 1 1 1 0 1 0 9 9 9 9 0 9 9 9 9	9 0 9 0 0 0 9 0 9	Le « trou » central (0) se propage en croix
1 4 1 3 1 1 1 10 200 5 80	10 5 5 80	B horizontal : chaque pixel « tire » le minimum des voisins de la ligne

```
%%writefile EP04_04.cpp
// your solution
```

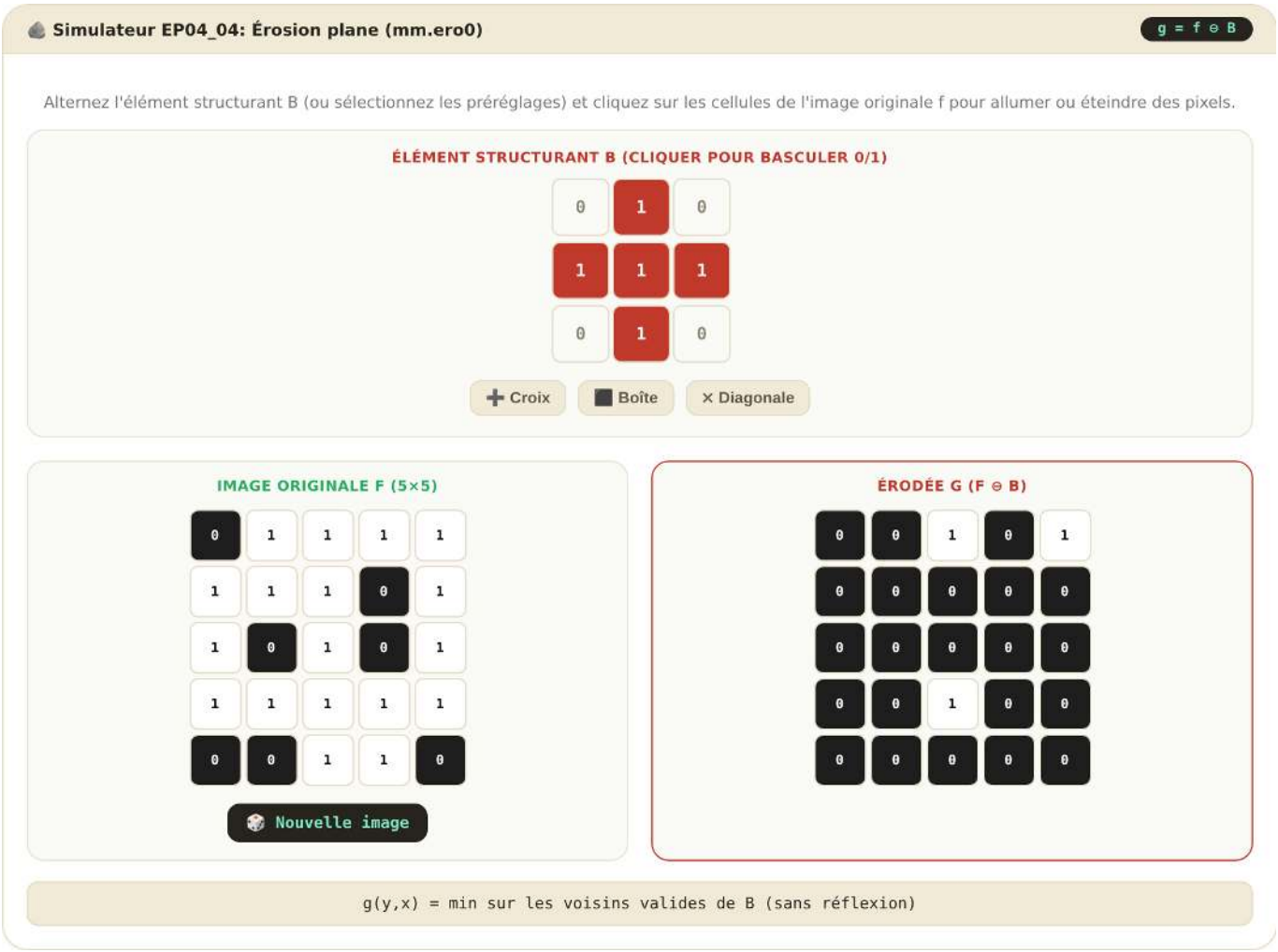
```
Overwriting EP04_04.cpp
```

```
TestSuite("EP04_04.cpp").run()
```

```
<IPython.core.display.HTML object>
```

4.9.5 EP04_05 🧹 Ouverture Morphologique (Suppression de Bruit)

Les images capturées par des **capteurs à faible coût**, comme ceux des drones agricoles, sont souvent parsemées de petits points de bruit — des pixels isolés qui ne représentent rien de réel. Appliquer une érosion suivie d'une dilatation avec le **même** élément structurant produit l'**ouverture** : elle « nettoie » les points et les fines protubérances, mais rend à l'objet principal pratiquement sa taille d'origine. C'est la combinaison classique utilisée dans le **pré-traitement d'images satellitaires** avant tout comptage de surface cultivée. Voir dans Figure 4.34 une simulation de cet EP.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.fr/cap04/sim-ep0404-erosao.html

Figure 4.33: Simulateur EP04_04: Érosion Binaire Plane (g = f B)

4.9.5.1 📋 Directives d'Implémentation

1. **Dimensions de l'image :** Lire les entiers L (lignes) et C (colonnes) de f.
2. **Dimensions de B :** Lire les entiers L_B (lignes) et C_B (colonnes) de l'élément structurant.
3. **Élément structurant :** Lire la matrice B avec des valeurs 0 ou 1, ligne par ligne.
4. **Données :** Lire la matrice binaire f (valeurs 0 ou 1), ligne par ligne.
5. **Érosion :** Calculer $e = f \ominus B$, en utilisant exactement l'algorithme de l'EP04_04 (sans réfléchir B, sans padding).
6. **Dilatation :** Calculer $g = e \oplus B$, en utilisant exactement l'algorithme de l'EP04_03 (en réfléchissant B, sans padding) — mais maintenant appliqué sur e, pas sur f.
7. **Sortie :** Afficher la matrice résultante g (l'**ouverture** de f par B) avec les dimensions $L \times C$.

4.9.5.2 📌 Contraintes Computationnelles

- **Ordre fixe :** C'est **toujours** l'érosion d'abord, puis la dilatation — l'ordre inverse définit un autre opérateur (la fermeture, du prochain EP).
- **Même B :** L'élément structurant utilisé pour l'érosion et la dilatation doit être identique.
- **Sans padding dans aucune des deux étapes.**

4.9.5.3 🧠 Fondement Théorique

Concept	Signification	Impact Visuel
Anti-extensivité	$g \subseteq f$ toujours	L'ouverture ne crée jamais de nouveau pixel, elle ne fait que supprimer
Idempotence	ouverture(ouverture(f)) = ouverture(f)	Réappliquer ne change plus rien
Points isolés	Plus petits que B	Ils sont complètement éliminés
Noyau de l'objet	Plus grand que B	Il est récupéré presque intact par la dilatation finale

4.9.5.4 📦 Spécification d'Entrée et de Sortie (VPL)

Entrée :

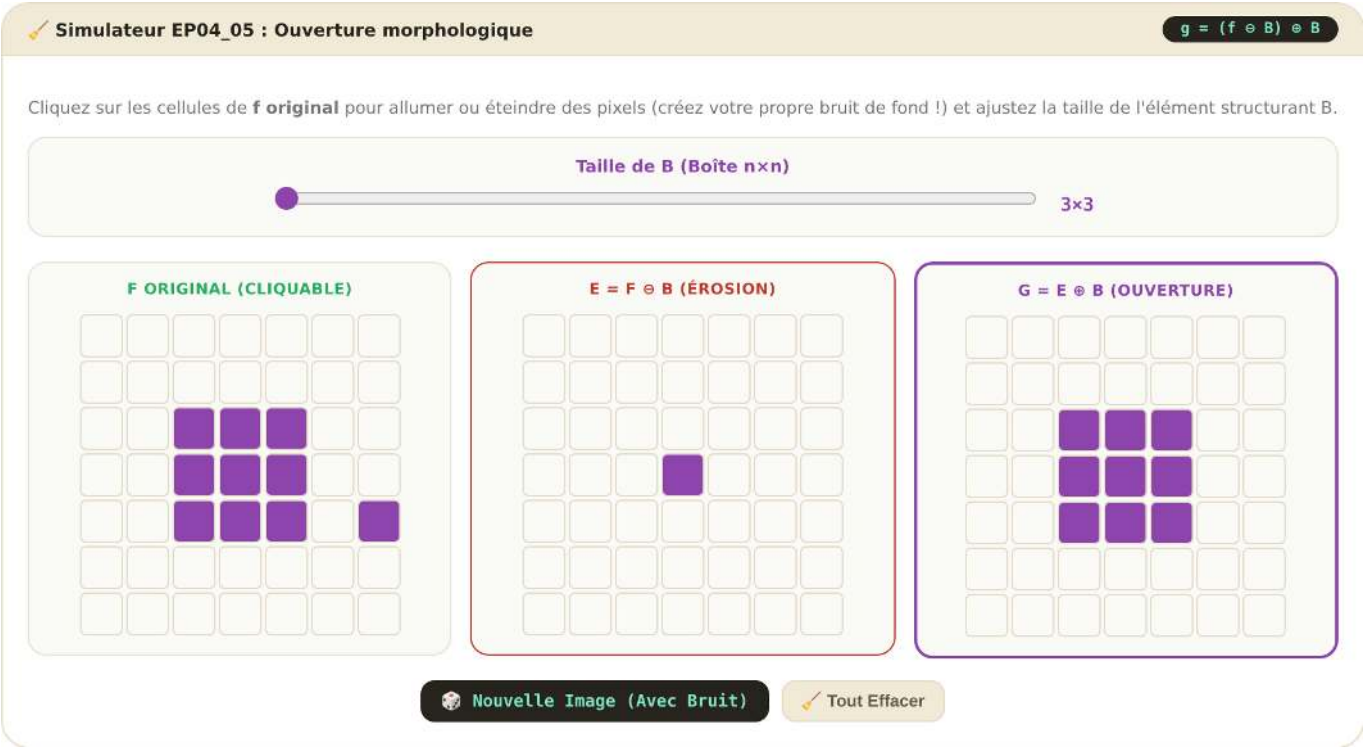
- Ligne 1 : Entier L.
- Ligne 2 : Entier C.
- Ligne 3 : Entier L_B.
- Ligne 4 : Entier C_B.
- Les L_B lignes suivantes : éléments entiers (0 ou 1) de la matrice B.
- Les L lignes suivantes : éléments entiers (0 ou 1) de la matrice f.

Sortie :

- Matrice résultante en L lignes et C colonnes, valeurs 0 ou 1.

4.9.5.5 📌 Exemples

Entrée	Sortie	Observation
7 7 3 3 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 1 1 0 0 0 0 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 1 1 1 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Les points isolés et la fine protubérance disparaissent ; le carré central survit



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.fr/cap04/sim-ep0405-abertura.html

Figure 4.34: Simulateur EP04_05 : Ouverture morphologique (g = (f B) B)

```
%%writefile EP04_05.cpp
// your solution
```

```
Overwriting EP04_05.cpp
```

```
TestSuite("EP04_05.cpp").run()
```

```
<IPython.core.display.HTML object>
```

4.9.6 EP04_06 🧩 Fermeture Morphologique (Remplissage des Lacunes)

Dans la **numérisation d'empreintes digitales**, les sillons de la peau sont parfois interrompus par de la saleté ou un dessèchement, créant de petites lacunes dans la courbe continue qui devrait exister. La **fermeture** — dilatation suivie d'une érosion avec le même élément structurant — est l'opérateur dual de l'ouverture : elle **remplit les petits trous et les renfoncements étroits**, sans modifier significativement le contour externe de l'objet. C'est l'étape standard avant d'extraire le squelette d'une empreinte digitale. Voir dans Figure 4.35 une simulation de cet EP.

4.9.6.1 📋 Directives d'implémentation

1. **Dimensions de l'image :** Lire les entiers L (lignes) et C (colonnes) de f.
2. **Dimensions de B :** Lire les entiers L_B (lignes) et C_B (colonnes) de l'élément structurant.
3. **Élément structurant :** Lire la matrice B avec les valeurs 0 ou 1, ligne par ligne.
4. **Données :** Lire la matrice binaire f (valeurs 0 ou 1), ligne par ligne.
5. **Dilatation :** Calculer $d = f \oplus B$, en utilisant exactement l'algorithme de l'EP04_03 (réflexion de B, sans padding).
6. **Érosion :** Calculer $g = d \ominus B$, en utilisant exactement l'algorithme de l'EP04_04 (sans réflexion de B, sans padding) — maintenant appliqué sur d, et non sur f.
7. **Sortie :** Afficher la matrice résultante g (la **fermeture** de f par B) avec les dimensions $L \times C$.

4.9.6.2 📌 Contraintes de calcul

- **Ordre fixe :** C'est **toujours** la dilatation d'abord, puis l'érosion — l'ordre inverse est l'ouverture de l'EP04_05.
- **Même B :** L'élément structurant utilisé dans la dilatation et dans l'érosion doit être identique.
- **Sans padding à aucune des deux étapes.**

4.9.6.3 🧠 Fondement théorique

Concept	Signification	Impact visuel
Extensivité	$g \supseteq f$ toujours	La fermeture ne supprime jamais de pixel, elle n'ajoute que
Idempotence	fermeture(fermeture(f)) = fermeture(f)	Réapplique ne change plus rien
Petits trous	Plus petits que B	Sont complètement remplis
Dualité	fermeture(f) = $\overline{\text{ouverture}(\bar{f})}$	C'est l'ouverture appliquée au « négatif » de l'image

4.9.6.4 📦 Spécification d'entrée et de sortie (VPL)

Entrée :

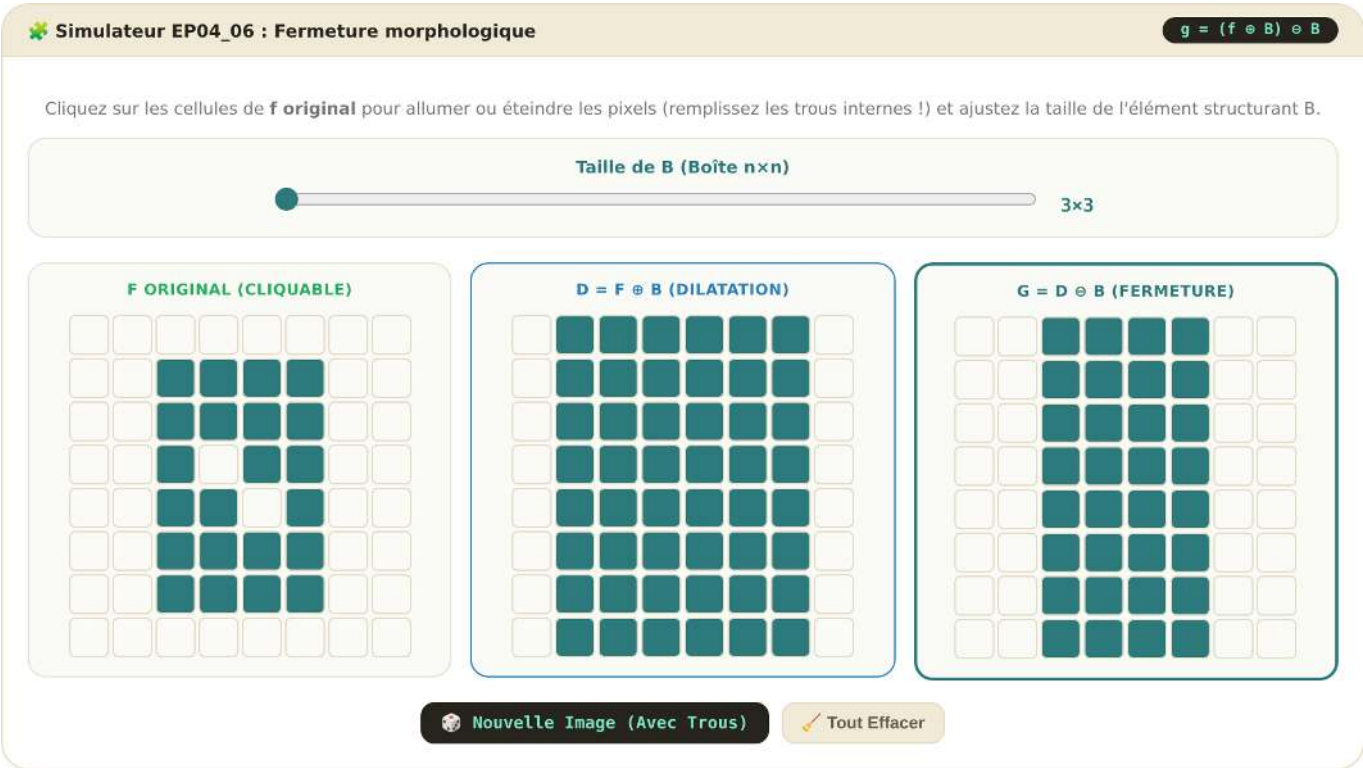
- Ligne 1 : Entier L.
- Ligne 2 : Entier C.
- Ligne 3 : Entier L_B.
- Ligne 4 : Entier C_B.
- Les L_B lignes suivantes : éléments entiers (0 ou 1) de la matrice B.
- Les L lignes suivantes : éléments entiers (0 ou 1) de la matrice f.

Sortie :

- Matrice résultante en L lignes et C colonnes, valeurs 0 ou 1.

4.9.6.5 📌 Exemples

Entrée	Sortie	Observation
8 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0	0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0	Les deux trous internes non adjacents sont totalement remplis
8 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 0 1 1 0 0 0 0 1 1 0 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0	0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0	



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.fr/cap04/sim-ep0406-fechamento.html

Figure 4.35: Simulateur EP04_06: Fermeture morphologique (g = (f B) B)

```
%%writefile EP04_06.cpp
// your solution
```

```
Overwriting EP04_06.cpp
```

```
TestSuite("EP04_06.cpp").run()
```

```
<IPython.core.display.HTML object>
```

4.9.7 EP04_07 Dilatation et Érosion Pondérées (mm.dil1 / mm.ero1)

Jusqu'à présent, l'élément structurant disait simplement « ce voisin compte » ou « ne compte pas » — mais dans les **modèles numériques de terrain** (utilisés en SIG et en planification du drainage urbain), chaque voisin devrait avoir un **poids différent** selon la distance ou la direction du relief. Les versions **pondérées** de la dilatation et de l'érosion, implémentées dans `morph.py` comme `mm::dil1(f, b)` et `mm::ero1(f, b)`, additionnent (ou soustraient) le poids de chaque voisin avant de prendre le maximum (ou le minimum) — généralisant ainsi tout ce qui a été fait dans les EP précédents. Voir dans Figure 4.36 une simulation de cet EP.

4.9.7.1 📋 Directives d'Implémentation

1. **Dimensions de l'image :** Lire les entiers L (lignes) et C (colonnes) de f.
2. **Dimensions de b :** Lire les entiers L_B (lignes) et C_B (colonnes) de l'élément structurant pondéré.
3. **Poids :** Lire la matrice b des poids **entiers** (ils peuvent être négatifs, nuls ou positifs), ligne par ligne.
4. **Données :** Lire la matrice f (l'image d'origine), ligne par ligne.
5. **Voisinage sans padding :** Pour chaque pixel (y, x), parcourir **toutes** les positions (by, bx) de b (pas seulement celles où la valeur serait 1 — ici **tout** poids participe), en utilisant le même décalage que dans les EP précédents :

$$v_y = y + by + o_y, \quad v_x = x + bx + o_x, \quad o_y = -\tfrac{L_B}{2} + 0{,}5, \quad o_x = -\tfrac{C_B}{2} + 0{,}5$$

   Écarter tout (v_y, v_x) hors de $[0, L) \times [0, C)$.
6. **Dilatation pondérée :** Calculer

$$g_{dil}(y, x) = \max\left(f(y, x), \max_{(v_y, v_x)\ \text{valide}} \left(f(v_y, v_x) + b(by, bx)\right)\right)$$

7. **Érosion pondérée :** Calculer, **en utilisant le même b et sans réflexion** :

$$g_{ero}(y, x) = \min\left(f(y, x), \min_{(v_y, v_x)\ \text{valide}} \left(f(v_y, v_x) - b(by, bx)\right)\right)$$

8. **Sortie :** Afficher **d'abord** la matrice complète g_{dil}, puis **ensuite** la matrice complète g_{ero}.

4.9.7.2 📌 Contraintes Computationnelles

- **Aucune des deux ne réfléchit** b — la version pondérée n'utilise pas la réflexion, même pour la dilatation (contrairement à `mm::dil0`).
- **Tous les poids participent :** Il n'existe pas ici de filtre « $B = 1$ » ; même un poids 0 entre en compte.
- **Sans padding :** les voisins hors de l'image sont ignorés, jamais virtuellement remplis.
- **Type :** La sortie peut contenir des valeurs négatives ou supérieures à 255 — **pas** de *clipping* dans cet EP.
- **Astuce :** Pour supprimer les messages de dépassement lors du dépassement des limites du type uint8, inclure au début du code :

```
import warnings
warnings.filterwarnings("ignore")
```

4.9.7.3 🧠 Fondement Théorique

Concept	Signification	Impact Visuel
Poids positif	« Tire » la valeur du voisin vers le haut lors de la dilatation	Simule un relief qui monte dans cette direction
Poids négatif	Réduit la contribution du voisin	Simule la distance ou une atténuation directionnelle

Concept	Signification	Impact Visuel
Dualité pondérée	$\text{ero1}(f, b) = -\text{dil1}(-f, b)$	La symétrie entre les deux opérations se maintient même avec des poids

4.9.7.4 📦 Spécification d'Entrée et de Sortie (VPL)

Entrée :

- Ligne 1 : Entier L.
- Ligne 2 : Entier C.
- Ligne 3 : Entier L_B.
- Ligne 4 : Entier C_B.
- Les L_B lignes suivantes : éléments entiers (pouvant être négatifs) de la matrice b.
- Les L lignes suivantes : éléments entiers de la matrice f.

Sortie :

- D'abord la matrice g_{dil} en L lignes et C colonnes.
- Ensuite la matrice g_{ero} en L lignes et C colonnes.

4.9.7.5 📌 Exemples

Entrée	Sortie	Observation
3 3 3 3 0 1 0 1 2 1 0 1 0 10 20 30 40 50 60 70 80 90	50 60 61 80 90 91 81 91 92 8 9 19 9 10 20 39 40 50	Le poids central 2 accélère la croissance lors de la dilatation et le rétrécissement lors de l'érosion

```
%%writefile EP04_07.cpp
// your solution
```

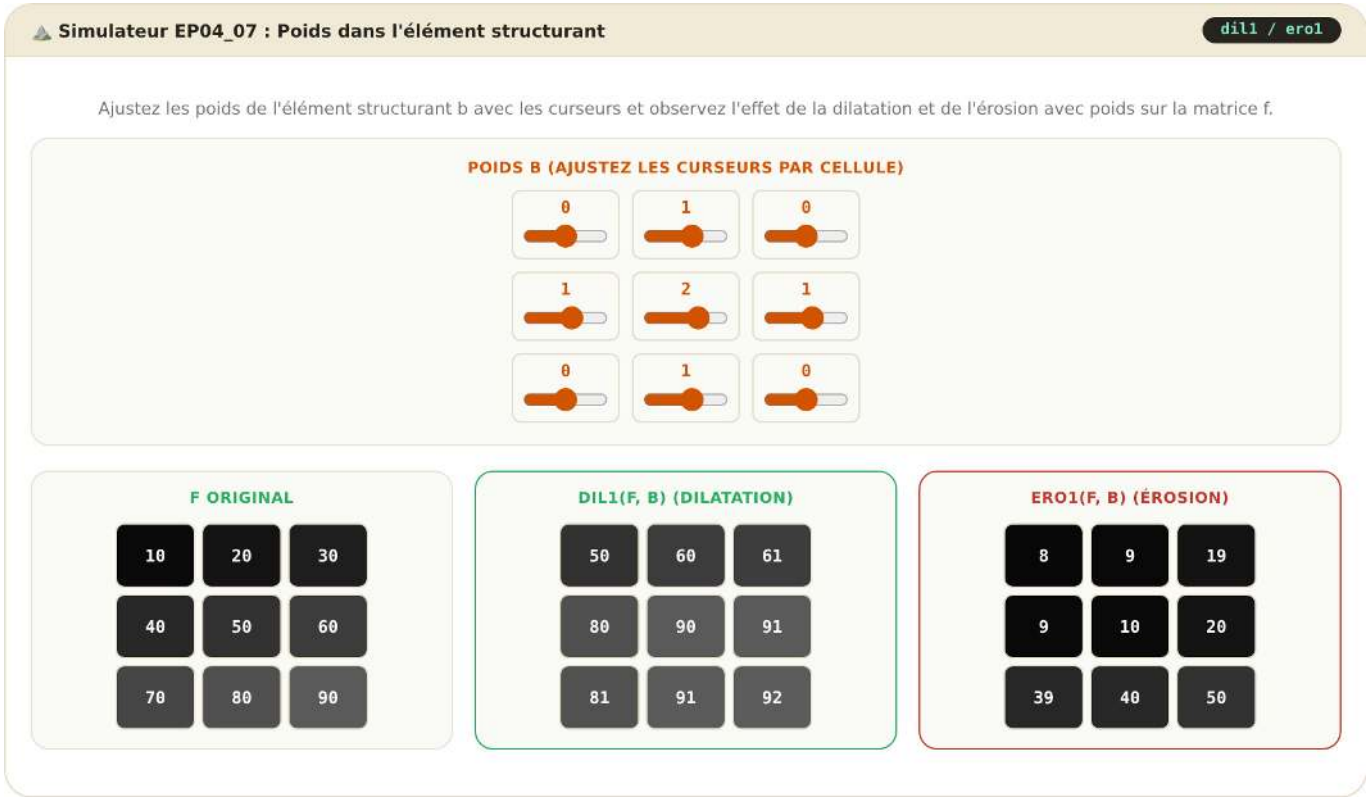
```
Overwriting EP04_07.cpp
```

```
TestSuite("EP04_07.cpp").run()
```

```
<IPython.core.display.HTML object>
```

4.9.8 EP04_08 🌋 Gradient morphologique, Top-hat et Black-hat

En **inspection automatique de plaques de circuits**, trois questions reviennent constamment : où se trouvent les **bords** des composants ? Quels **détails clairs et petits** (comme les points de soudure) se détachent du fond ? Quelles **cavités sombres** (comme les fissures) le fond dissimule-t-il ? Une seule paire érosion/dilatation répond aux trois : le **gradient morphologique** met en évidence les contours, le **top-hat** révèle les pics étroits, et le **black-hat** révèle les vallées étroites — trois outils, un seul voisinage. Voir dans Figure 4.37 une simulation de cet EP.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.fr/cap04/sim-ep0407-pesos.html

Figure 4.36: Simulateur EP04_07 : Dilatation et Érosion avec Poids (mm.dil1 / mm.ero1)

4.9.8.1 📋 Directives d'implémentation

1. **Dimensions de l'image :** Lire les entiers L (lignes) et C (colonnes) de f.
2. **Dimensions de B :** Lire les entiers L_B (lignes) et C_B (colonnes) de l'élément structurant.
3. **Élément structurant :** Lire la matrice B avec des valeurs 0 ou 1, ligne par ligne.
4. **Données :** Lire la matrice f (l'image originale, en niveaux de gris), ligne par ligne.
5. **Opérateurs de base :** Calculer, exactement comme dans les EP 04_03 à 04_06 :
   - $d = f \oplus B$ (dilatation),
   - $e = f \ominus B$ (érosion),
   - ouverture $= e \oplus B$,
   - fermeture $= d \ominus B$.
6. **Gradient morphologique :** $\text{grad}(y,x) = d(y,x) - e(y,x)$.
7. **Top-hat :** $\text{tophat}(y,x) = f(y,x) - \text{ouverture}(y,x)$.
8. **Black-hat :** $\text{blackhat}(y,x) = \text{fermeture}(y,x) - f(y,x)$.
9. **Sortie :** Afficher, **dans cet ordre**, les trois matrices complètes : gradient, top-hat, black-hat.

4.9.8.2 📌 Contraintes computationnelles

- **Sans padding à aucune étape intermédiaire** — dilatation, érosion, ouverture et fermeture suivent les mêmes règles de voisinage que les EP précédents.
- **Pas de *clipping* :** les trois sorties peuvent contenir n'importe quelle valeur entière (le gradient est toujours ≥ 0, mais top-hat et black-hat le sont aussi).
- **Réutilisation :** d et e doivent être calculés **une seule fois** et réutilisés pour construire ouverture, fermeture et gradient.

4.9.8.3 🧠 Fondements théoriques

Opérateur	Formule	Ce qu'il révèle
Gradient	$d - e$	Bords : zéro dans les régions planes, élevé aux transitions
Top-hat	$f - \text{ouverture}(f)$	Éléments **clairs et fins**, plus petits que B
Black-hat	$\text{fermeture}(f) - f$	Éléments **sombres et fins**, plus petits que B

4.9.8.4 📦 Spécification d'entrée et de sortie (VPL)

Entrée :

- Ligne 1 : Entier L.
- Ligne 2 : Entier C.
- Ligne 3 : Entier L_B.
- Ligne 4 : Entier C_B.
- Lignes suivantes L_B : éléments entiers (0 ou 1) de la matrice B.
- Lignes suivantes L : éléments entiers de la matrice f.

Sortie :

- Matrice gradient en L lignes et C colonnes.
- Matrice top-hat en L lignes et C colonnes.
- Matrice black-hat en L lignes et C colonnes.

4.9.8.5 📌 Exemples

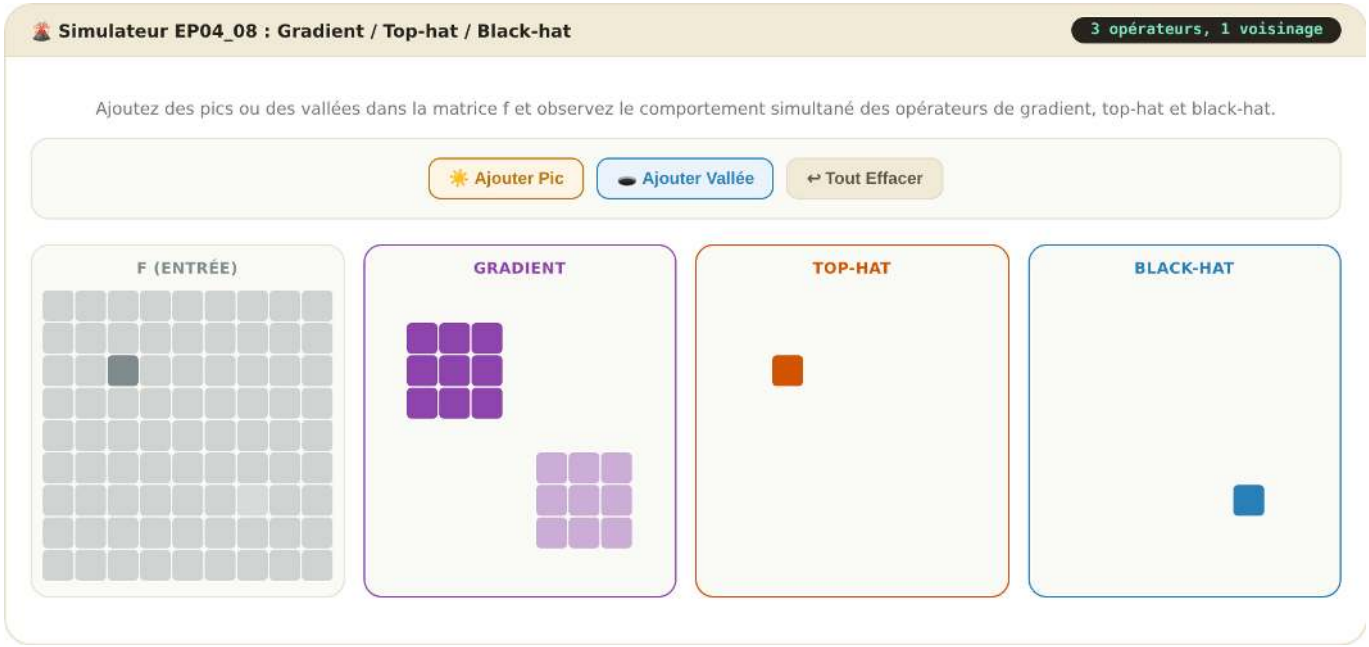
Entrée	Sortie	Observation
9 9 3 3 1 1 1 1 1 1 1 1 1 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 80 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 2 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10	(gradient : halo $3 \times 3 = 70$ autour de $(2, 2)$ et halo $3 \times 3 = 8$ autour de $(6, 6)$, reste 0) (top-hat : unique 70 en $(2, 2)$, reste 0) (black-hat : unique 8 en $(6, 6)$, reste 0)	Pic isolé devient top-hat ; vallée isolée devient black-hat ; les deux apparaissent dans le gradient

```
%%writefile EP04_08.cpp
// your solution
```

```
Overwriting EP04_08.cpp
```

```
TestSuite("EP04_08.cpp").run()
```

```
<IPython.core.display.HTML object>
```



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.fr/cap04/sim-ep0408-gradiente.html

Figure 4.37: Simulateur EP04_08: Gradient morphologique, Top-hat et Black-hat

4.9.9 EP04_09 Transformée de distance et le « cœur » de l'objet

En **robotique mobile**, lors de la planification d'un itinéraire dans un couloir, le robot souhaite savoir non seulement *où* se trouve l'espace libre, mais aussi **à quelle distance** chaque point libre se trouve du mur le plus proche. Les chemins les plus sûrs tendent à passer par le « cœur » du couloir, loin des obstacles.

La **transformée de distance morphologique** attribue à chaque pixel une valeur représentant sa distance jusqu'au bord le plus proche, selon la métrique définie par l'élément structurant. Les pixels proches du bord reçoivent des valeurs faibles, tandis que les pixels plus internes reçoivent des valeurs plus élevées. Le pixel de valeur maximale correspond à la région la plus protégée de l'objet, souvent associée à son centre morphologique.

Voir dans Figure 4.38 une simulation de cet EP.

4.9.9.1 📋 Directives d'implémentation

1. **Dimensions de l'image :** lire les entiers L (lignes) et C (colonnes) de l'image f.
2. **Dimensions de B :** lire les entiers L_B (lignes) et C_B (colonnes) de l'élément structurant.
3. **Élément structurant :** lire la matrice b, contenant la valeur 0 au centre et des valeurs négatives aux autres positions.
4. **Image :** lire la matrice binaire f (valeurs 0 ou 1), ligne par ligne.
5. **Préparation :** multiplier l'image par $L \times C$, en garantissant que les pixels internes ont une valeur initiale suffisamment élevée pour la propagation des distances.
6. **Transformée de distance :** calculer la matrice des distances en utilisant la méthode `mm::dist1(f,b)`.
7. **Sortie :** afficher la matrice résultante de la transformée de distance.

4.9.9.2 📌 Contraintes de calcul

- Utiliser l'implémentation de l'érosion pondérée fournie par la bibliothèque.
- L'élément structurant peut contenir des valeurs négatives arbitraires.
- La transformée doit être obtenue par l'application itérative d'érosions pondérées jusqu'à atteindre un point fixe.

⚠️ **Note cruciale sur la lecture des matrices :** Comme l'élément structurant peut contenir des entiers négatifs (par exemple, `-1` et `-99`), **ne pas utiliser la fonction `mm::readImg` pour lire la matrice** b. Cette

fonction convertit les données en type `uint8`, provoquant un *underflow* et corrompant les valeurs négatives. Lire les L_B lignes de b manuellement en utilisant le type standard `int`. L'image f peut continuer à être lue normalement par `mm::readImg`.

4.9.9.3 🧠 Fondement théorique

Concept	Signification	Impact visuel
dist(y, x)	Distance morphologique jusqu'au bord le plus proche selon la métrique définie par b	Les pixels plus internes reçoivent des valeurs plus élevées
Valeur maximale	Pixel le plus éloigné du bord	Se rapproche du centre morphologique de l'objet
Élément structurant pondéré	Définit les coûts de déplacement entre pixels voisins	Détermine la métrique de distance utilisée
Objets fins	Régions étroites de l'objet	Produisent des valeurs de distance faibles

4.9.9.4 📦 Spécification d'entrée et de sortie (VPL)

Entrée :

- Ligne 1 : entier L.
- Ligne 2 : entier C.
- Ligne 3 : entier L_B.
- Ligne 4 : entier C_B.
- Les L_B lignes suivantes : éléments entiers de la matrice b.
- Les L lignes suivantes : éléments binaires (0 ou 1) de la matrice f.

⚠️ **Note d'implémentation :** Les éléments de la matrice f (0 ou 1) doivent être multipliés par **255** pour générer une image binaire appropriée (0 et 255) avant d'appliquer la Transformée de Distance (TD).

Sortie :

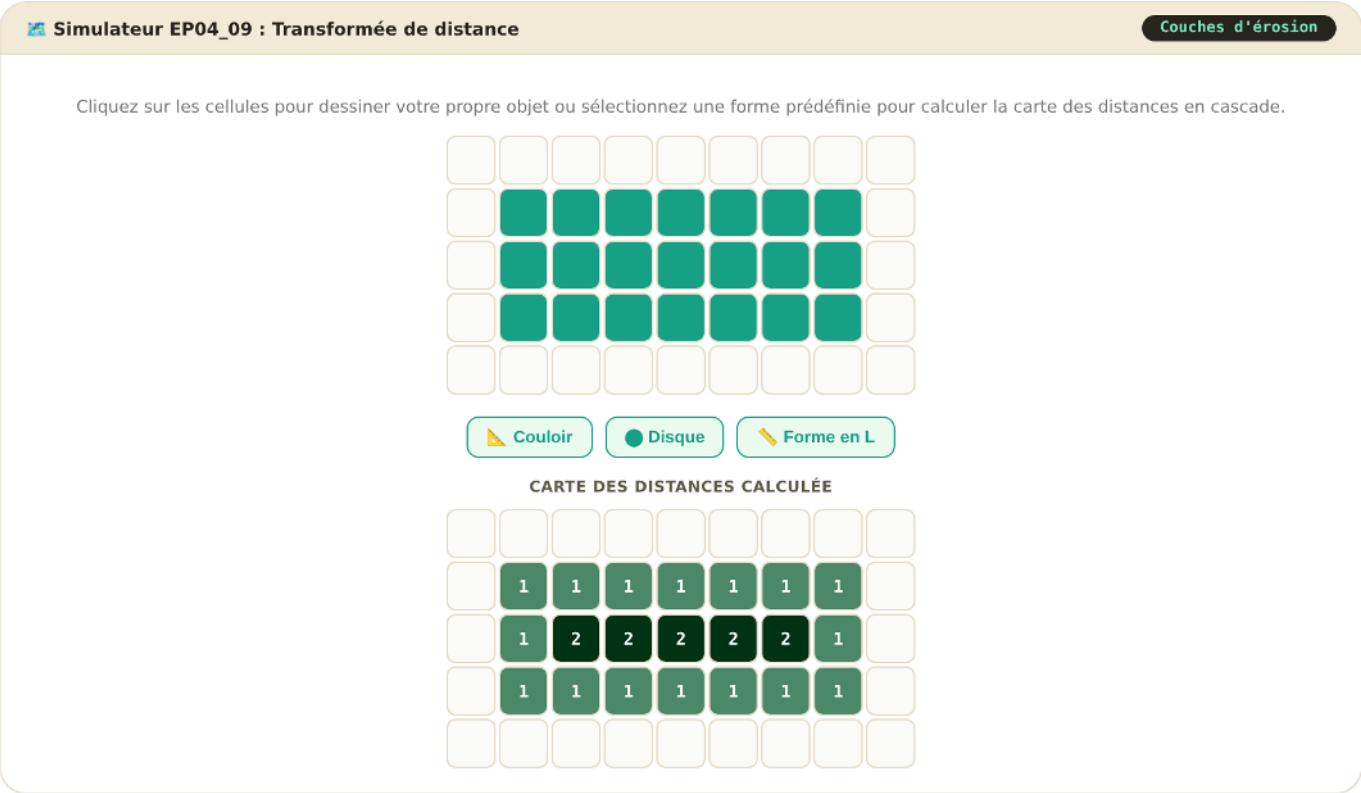
- Matrice de la transformée de distance en L lignes et C colonnes.

4.9.9.5 📌 Exemple

Entrée	Sortie	Observation
5 9 3 3 -99 -1 -99 -1 0 -1 -99 -1 -99 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 0 0 1 2 2 2 2 2 1 0 0 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0	Résultat de la transformée de distance.

Note : la valeur `-99` agit comme une approximation pratique de $-\infty$, empêchant la propagation par les

diagonales. Ainsi, seuls les voisins horizontaux et verticaux contribuent à la distance, produisant la distance de Manhattan.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.fr/cap04/sim-ep0409-distancia.html

Figure 4.38: Simulateur EP04_09: Transformée de Distance (Couches d'Érosion)

```
%%writefile EP04_09.cpp
// your solution
```

```
Overwriting EP04_09.cpp
```

```
TestSuite("EP04_09.cpp").run()
```

```
<IPython.core.display.HTML object>
```

4.9.10 EP04_10 🪙 Séparation des *blobs*, étiquetage et descripteurs

Dans une **ligne de production de pièces de monnaie**, il est courant que les pièces se touchent sur le tapis roulant, formant une seule tache connectée dans l'image — un comptage naïf donnerait un nombre erroné. La solution classique combine des opérations morphologiques et une analyse de connectivité : d'abord, une **érosion** réduit ou rompt les connexions fragiles entre les objets, puis l'**étiquetage des composantes connexes** sépare chaque objet en une région distincte. Enfin, des **descripteurs géométriques** (aire et boîte englobante) résument chaque composante trouvée.

Voir Figure 4.39 pour une simulation de cet EP.

4.9.10.1 📋 Directives d'implémentation

1. **Dimensions de l'image :** lire les entiers L (lignes) et C (colonnes) de f.
2. **Dimensions de B :** lire les entiers L_B (lignes) et C_B (colonnes) de l'élément structurant.
3. **Élément structurant :** lire la matrice B, contenant des valeurs 0 ou 1, ligne par ligne.

4. **Données :** lire la matrice binaire f (valeurs 0 ou 1), ligne par ligne.
5. **Séparation :** calculer
$$f_{ero} = f \ominus B$$
en utilisant une érosion binaire plane (comme dans l'EP04_04), en éliminant les connexions fragiles entre les objets.
6. **Étiquetage :** sur f_{ero}, identifier les composantes connexes en utilisant la connectivité définie par le voisinage B. L'étiquetage doit suivre un balayage *raster* : lorsqu'un pixel 1 non encore étiqueté est trouvé, attribuer un nouveau label entier croissant à partir de 1 et propager ce label à toute la région connexe.
7. **Descripteurs :** pour chaque label k, calculer :
   - **Aire :** nombre de pixels appartenant au label ;
   - **Boîte englobante :**
$$(y_{min}, x_{min}, y_{max}, x_{max})$$
8. **Sortie :** afficher le nombre total de labels, puis une ligne par label au format :
$$k,\ \text{aire},\ y_{min},\ x_{min},\ y_{max},\ x_{max}$$

4.9.10.2 📌 Contraintes computationnelles

- L'érosion doit être appliquée avant l'étiquetage.
- La connectivité est fixe et définie par le voisinage ci-dessus.
- L'élément structurant B n'interfère pas avec la connectivité de l'étiquetage.
- Aucun padding à aucune étape.
- L'ordre des labels suit la première découverte en balayage *raster*.

4.9.10.3 🧠 Fondement théorique

Concept	Signification	Impact
Pont fin	Connexion étroite entre objets	Peut être supprimé par l'érosion morphologique
Connectivité	Définie par l'ensemble $\mathcal{N}(y, x)$	Détermine quels pixels appartiennent à la même composante
Aire	Nombre de pixels par composante	Estimation directe de la taille de l'objet
Boîte englobante	Extension spatiale du label	Résumé géométrique de la composante

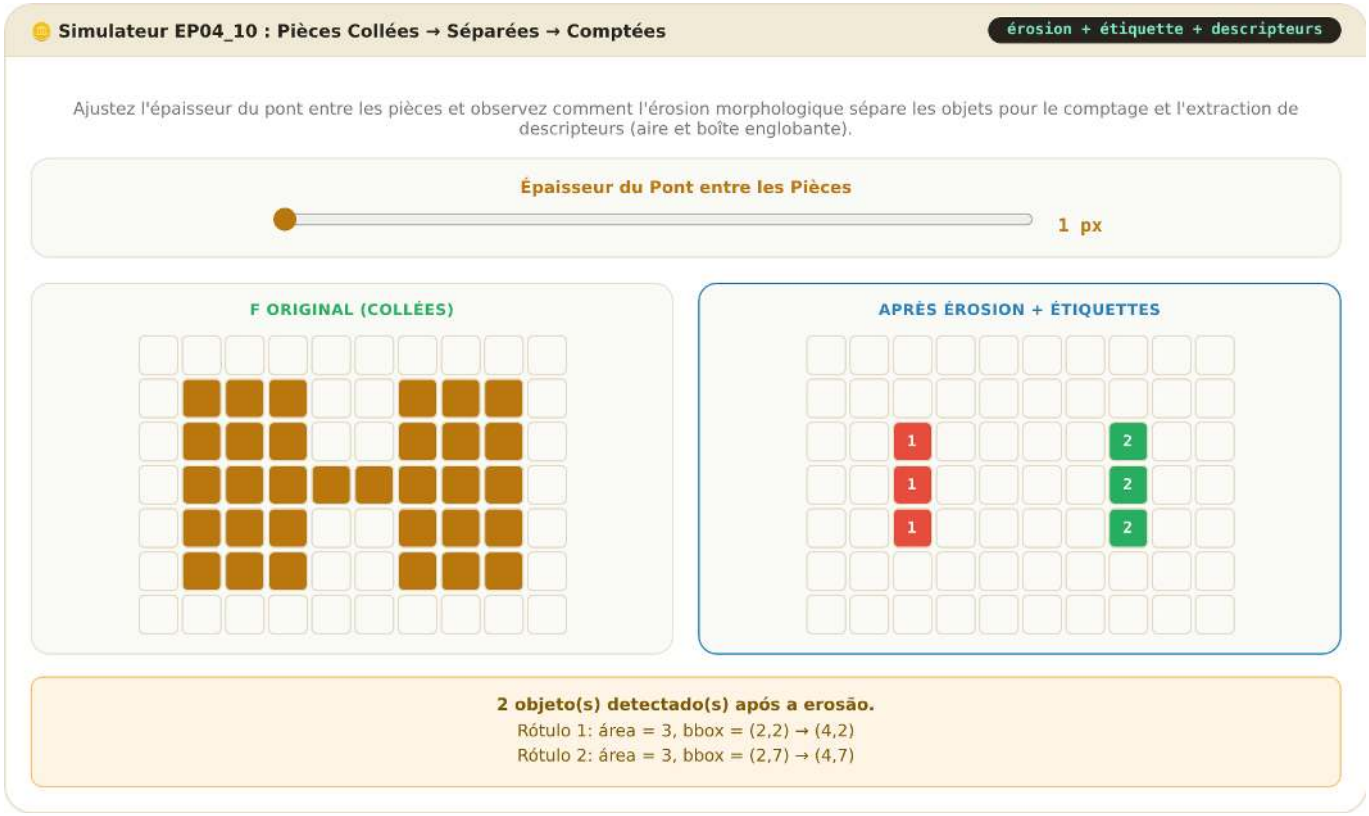
4.9.10.4 📦 Spécification d'entrée et de sortie (VPL)

Entrée :

- Ligne 1 : entier L
- Ligne 2 : entier C
- Ligne 3 : entier L_B
- Ligne 4 : entier C_B
- Les L_B lignes suivantes : matrice B
- Les L lignes suivantes : matrice f

Sortie :

- Ligne 1 : nombre total de labels trouvés
- Lignes suivantes :
$$k,\ \text{aire},\ y_{min},\ x_{min},\ y_{max},\ x_{max}$$



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.fr/cap04/sim-ep0410-rotulacao.html

Figure 4.39: Simulateur EP04_10: Séparation des Blobs, Étiquetage et Descripteurs

```
%%writefile EP04_10.cpp
// your solution
```

```
Overwriting EP04_10.cpp
```

```
TestSuite("EP04_10.cpp").run()
```

```
<IPython.core.display.HTML object>
```

Chapitre 5

Transformées et Compression

Dans les chapitres précédents, toutes les opérations ont été réalisées **dans le domaine spatial**, où les algorithmes agissent directement sur les valeurs d'intensité des pixels.

Ce chapitre présente une approche complémentaire : le **domaine fréquentiel**, dans lequel l'image est représentée par les variations spatiales d'intensité, et non seulement par les valeurs individuelles des pixels.

Le concept de **fréquence spatiale** décrit la rapidité avec laquelle l'intensité varie le long de l'image. Les variations lentes correspondent aux **basses fréquences**, tandis que les contours, les détails fins et les bruits correspondent aux **hautes fréquences**.

Cette représentation repose sur le fait que toute image numérique discrète peut être décomposée en une combinaison de **fonctions orthogonales**. La **Transformée de Fourier** utilise une **base d'exponentielles complexes bidimensionnelles** (équivalentes à des sinusoïdes avec une orientation et une fréquence spécifiques). D'autres transformées, comme la **Transformée en Cosinus (DCT)** et la **Transformée *Wavelet* (DWT)**, utilisent différentes familles de fonctions de base — des cosinus bidimensionnels dans le cas de la DCT, et des fonctions à support compact dans le cas des *wavelets*.

Parmi les principales applications de cette représentation, on distingue :

1. **Le filtrage dans le domaine fréquentiel**, pour atténuer ou rehausser certaines bandes de fréquences ;
2. **L'analyse multirésolution au moyen de transformées *wavelet***, qui représente les structures à différentes échelles ;
3. **La compression d'images**, par la réduction du nombre de coefficients nécessaires pour représenter l'image.

5.1 Objectifs

À la fin de ce chapitre, vous serez capable de :

- **Interpréter le spectre de Fourier** d'une image, en distinguant l'amplitude, la phase et les composantes de fréquence ;
- **Appliquer le théorème de convolution** pour effectuer un filtrage dans le domaine fréquentiel à l'aide de la transformée de Fourier rapide (FFT) ;
- **Concevoir et analyser des filtres dans le domaine fréquentiel**, en comprenant le fonctionnement des filtres passe-bas, passe-haut et *coupe-bande* ;
- **Comprendre l'analyse multirésolution par transformées en *ondelettes*** et son application à la représentation hiérarchique des images ;
- **Décrire le processus de compression d'images**, y compris la transformée en cosinus discrète (DCT) et la quantification des coefficients ;
- **Choisir des formats de stockage d'images**, tels que JPEG, PNG et WebP, en fonction des exigences de l'application.

5.2 Configuration de l'environnement

```
import os, urllib.request

os.makedirs("tmp/state", exist_ok=True)  # artefacts de build du parcours C++

url = "https://raw.githubusercontent.com/fzampirolli/pdi-vc/master/morph/config.py"
if not os.path.exists("config.py"):
    urllib.request.urlretrieve(url, "config.py")

# Noyau Python même dans le parcours C++. cpp=True télécharge morph.hpp + stb ; les
# cellules %%writefile *.cpp de ce chapitre compilent AVEC OpenCV
# (-DMM_USE_OPENCV + pkg-config opencv4).
import config
config.setup(cpp=True)
from morph import mm
import numpy as np
```

```
Environnement prêt. Morph : 1.1.9 | OpenCV : 5.0.0
```

5.3 Transformada de Fourier Discreta 2D

L'analyse de Fourier repose sur le principe selon lequel tout signal périodique peut être représenté comme une somme de fonctions sinusoïdales de différentes fréquences, amplitudes et phases. Ce concept s'applique également aux images numériques, permettant de les représenter dans le **domaine fréquentiel** plutôt que dans le domaine spatial.

La Figure 5.1 illustre cette décomposition pour un signal unidimensionnel. Dans le cas d'une image, la Transformée de Fourier Discrète (TFD) convertit la matrice d'intensités $f(x, y)$ en un ensemble de coefficients décrivant la contribution des différentes fréquences spatiales présentes dans l'image.

```
%%writefile tmp/fig_decomposicao_1d.cpp
#define MM_OUT "tmp/fig_decomposicao_1d.png"
//| label: fig-decomposicao-1d
//| fig-cap: "Décomposition de Fourier 1D: une onde carrée (ligne pointillée) est approchée
par la somme des premières sinusoïdes (lignes colorées). Plus il y a de termes, meilleure est
l'approximation."
//| echo: false
//| output: true

#include <iostream>
#include <vector>
#include <cmath>
#include "morph.hpp"
#include <filesystem>

int main() {
    // Créer l'axe x de 0 à 2  avec 400 points
    std::vector<double> x(400);
    for (int i = 0; i < 400; ++i) {
        x[i] = (2.0 * M_PI) * i / 399.0;
    }

    // Onde carrée : 1 pour x <  , -1 sinon
    std::vector<double> square(400);
    for (int i = 0; i < 400; ++i) {
        square[i] = (x[i] < M_PI) ? 1.0 : -1.0;
    }
```

```
    // Somme des harmoniques
    std::vector<double> soma(400, 0.0);
    std::vector<std::vector<double>> ys;
    std::vector<double> h1(400), h2(400), h3(400);

    for (int n = 1; n <= 3; ++n) {
        double coeff = (4.0 / M_PI) * (1.0 / (2 * n - 1));
        std::vector<double> h(400);
        for (int i = 0; i < 400; ++i) {
            h[i] = coeff * std::sin((2 * n - 1) * x[i]);
            soma[i] += h[i];
        }
        if (n == 1) h1 = h;
        else if (n == 2) h2 = h;
        else h3 = h;
    }

    // Préparer les courbes
    ys = {square, h1, h2, h3, soma};

    std::vector<std::string> labels = {
        "Onde carrée idéale", "1ère harmonique", "3ème harmonique",
        "5ème harmonique", "Somme (3 premières)"
    };
    std::vector<cv::Scalar> colors = {
        cv::Scalar(40, 40, 40), cv::Scalar(60, 160, 80),
        cv::Scalar(180, 120, 60), cv::Scalar(150, 80, 160),
        cv::Scalar(60, 60, 220)
    };

    // Créer le graphique
    mm::Image chart = mm::lineChart(
        x, ys, labels, colors,
        "Synthèse de Fourier : des sinus à une onde carrée",
        "Position", "Intensité"
    );

    mm::show(std::vector<mm::Image>{chart}, MM_OUT,
             std::vector<std::string>{"Décomposition de Fourier 1D"}, 1);


// [pdi:panel-io] auto-generated - do not edit by hand
std::filesystem::create_directories("tmp");
mm::write(chart, "tmp/fig_decomposicao_1d_0.png");
// [pdi:panel-io:end]
return 0;
}
```

```
Overwriting tmp/fig_decomposicao_1d.cpp
```

```
!g++ -I. -std=c++17 -DMM_USE_OPENCV -I/usr/include/opencv4 tmp/fig_decomposicao_1d.cpp -o
tmp/fig_decomposicao_1d -lopencv_stitching -lopencv_alphamat -lopencv_aruco -lopencv_barcode
-lopencv_bgsegm -lopencv_bioinspired -lopencv_ccalib -lopencv_dnn_objdetect
-lopencv_dnn_superres -lopencv_dpm -lopencv_face -lopencv_freetype -lopencv_fuzzy -lopencv_hdf
-lopencv_hfs -lopencv_img_hash -lopencv_intensity_transform -lopencv_line_descriptor
-lopencv_mcc -lopencv_quality -lopencv_rapid -lopencv_reg -lopencv_rgbd -lopencv_saliency
-lopencv_shape -lopencv_stereo -lopencv_structured_light -lopencv_phase_unwrapping
-lopencv_superres -lopencv_optflow -lopencv_surface_matching -lopencv_tracking
-lopencv_highgui -lopencv_datasets -lopencv_text -lopencv_plot -lopencv_ml -lopencv_videostab
-lopencv_videoio -lopencv_viz -lopencv_wechat_qrcode -lopencv_ximgproc -lopencv_video
-lopencv_xobjdetect -lopencv_objdetect -lopencv_calib3d -lopencv_imgcodecs -lopencv_features2d
-lopencv_dnn -lopencv_flann -lopencv_xphoto -lopencv_photo -lopencv_imgproc -lopencv_core \
  && ./tmp/fig_decomposicao_1d \
  && test -f "tmp/fig_decomposicao_1d.png" \
  || echo "  mm::show não gravou tmp/fig_decomposicao_1d.png"
```

```
[1] Décomposition de Fourier 1D
```

```
try:
    mm.show(
        [
            mm.read("tmp/fig_decomposicao_1d_0.png"),
        ],
        titles=[
            'Decomposicao de Fourier 1D',
        ],
        cols=1,
    )
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + "
    tmp/fig_decomposicao_1d_0.png (ver a versao Python)")
```

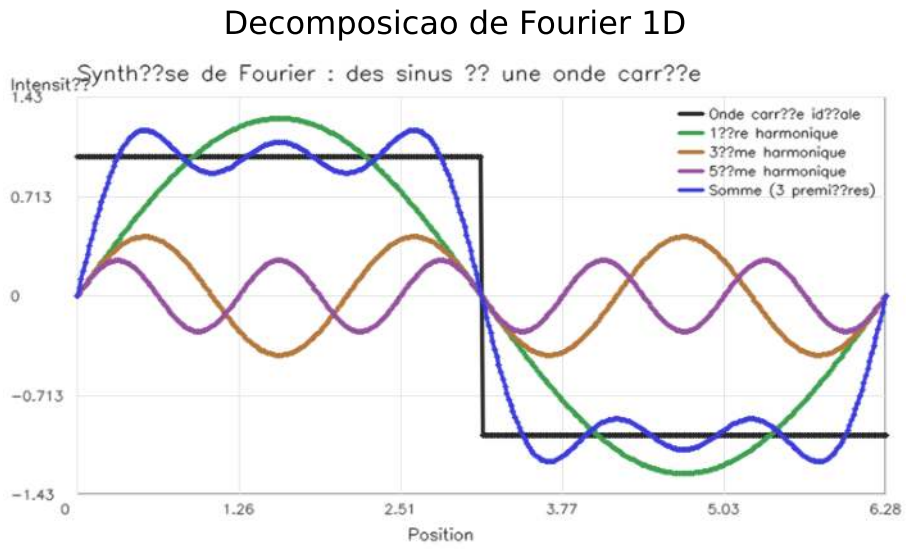


Figure 5.1: Decomposição de Fourier 1D: uma onda quadrada (linha tracejada) é aproximada pela soma das primeiras senoides (linhas coloridas). Quanto mais termos, melhor a aproximação.

5.3.1 Simulateur : Reconstruire des signaux avec des sinusoïdes

Avant d'étudier les images bidimensionnelles, le simulateur de la Figure 5.2 illustre le principe de l'analyse de Fourier pour des signaux unidimensionnels : **une forme d'onde peut être approchée par la somme de sinusoïdes de différentes fréquences et amplitudes**.

À mesure que de nouveaux termes sont ajoutés, la somme des sinusoïdes (courbe noire) se rapproche de la forme d'onde de référence (en pointillés). Le graphique inférieur présente le spectre d'amplitudes, indiquant la contribution de chaque fréquence à la reconstruction du signal.

Activité

Explorez le simulateur et répondez :

1. Combien de termes sont nécessaires pour obtenir une bonne approximation de l'onde carrée ?
2. Laquelle des trois formes d'onde converge le plus rapidement ? Justifiez votre réponse.
3. Comment le spectre d'amplitudes se modifie-t-il en remplaçant l'onde carrée par l'onde triangulaire ?

Réponses

1. Combien de termes sont nécessaires pour une bonne approximation de l'onde carrée ?
Avec environ 15 à 20 termes, la forme de l'onde se rapproche déjà bien de la référence. Cependant, à proximité des discontinuités subsiste une petite oscillation, connue sous le nom de **phénomène de Gibbs**, qui ne disparaît pas même avec l'ajout de termes supplémentaires.
2. Quelle forme converge le plus rapidement ? Pourquoi ?
L'**onde triangulaire** converge plus rapidement, car les amplitudes de ses harmoniques décroissent plus vite que celles de l'onde carrée et de l'onde en dents de scie. Par conséquent, peu de termes suffisent déjà à produire une bonne approximation.
3. Comment le spectre change-t-il entre l'onde carrée et l'onde triangulaire ?
Toutes deux ne possèdent que des **harmoniques impairs**, mais, dans l'onde triangulaire, les amplitudes diminuent beaucoup plus rapidement. Ainsi, peu d'harmoniques suffisent pour reconstruire le signal avec une bonne précision.

5.3.2 Interprétation du spectre de fréquences

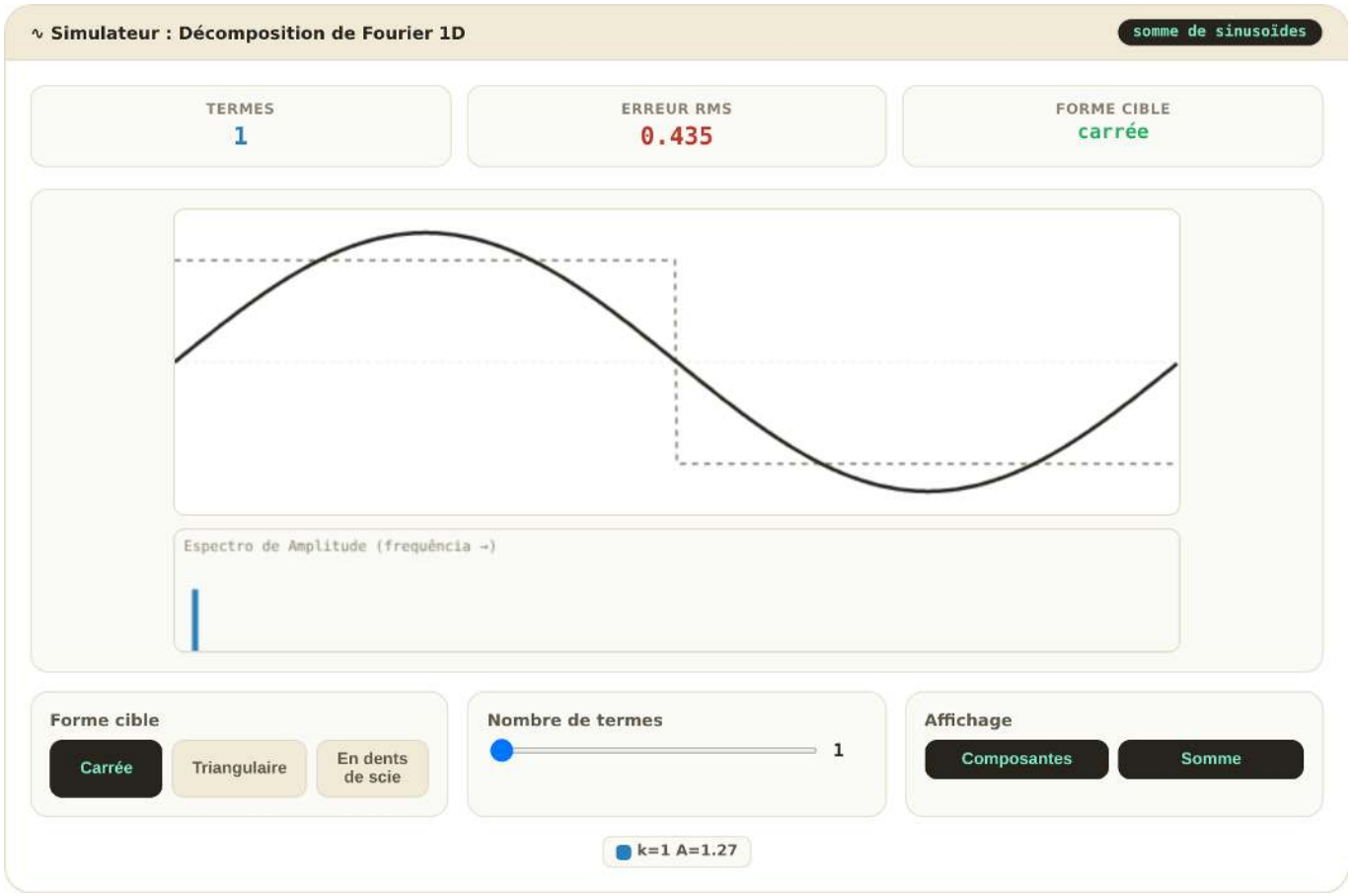
En appliquant la Transformée de Fourier Discrète (TFD) à une image et en visualisant le module de ses coefficients (voir Figure 5.5), on obtient le **spectre de magnitude**, qui montre la distribution des fréquences spatiales présentes dans l'image.

Le coefficient situé à l'origine de la TFD, appelé **composante continue** (*Direct Current*), correspond à la fréquence nulle et représente l'intensité moyenne de l'image. Par convention, ce coefficient est stocké dans le coin supérieur gauche du spectre. Pour faciliter son interprétation, on applique l'opération **FFT Shift**, qui déplace la composante continue vers le centre de l'image. Après ce décalage, les basses fréquences se concentrent dans la région centrale, tandis que les hautes fréquences se trouvent près des bords, comme le résume la Table 5.1.

Tableau 5.1: Correspondance entre les régions du spectre de magnitude après application du FFT Shift.

Région du spectre	Composantes prédominantes	Exemples dans l'image
Centre (basses fréquences)	Variations spatiales lentes	Éclairage, régions homogènes et formes globales
Région intermédiaire (fréquences moyennes)	Variations à échelle intermédiaire	Textures et motifs répétitifs
Bords (hautes fréquences)	Variations spatiales rapides	Contours, détails fins et bruit

Cette organisation facilite l'interprétation du spectre et la conception de filtres. L'atténuation des basses fréquences réduit les variations globales d'intensité, tandis que l'atténuation des hautes fréquences lisse l'image en réduisant les détails fins et une partie du bruit.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.fr/cap05/sim-05-freq.html

Figure 5.2: Simulateur interactif de la décomposition de Fourier 1D : visualisation de la somme de sinusoïdes avec différentes fréquences, amplitudes et phases. Ajoutez des termes et observez la convergence vers des formes d'onde arbitraires.

5.3.3 L'Expérience de la Grille : Construire une Image à partir d'un Unique Coefficient

Avant de présenter la formulation mathématique de la Transformée de Fourier Discrète (TFD), il est utile d'analyser son inverse, appelée Transformée de Fourier Discrète Inverse (TFDI). Considérons un spectre où tous les coefficients sont nuls, à l'exception d'un seul. Un exemple de cette construction est présenté dans le code de la Figure 5.3 et peut être exploré de manière interactive dans le simulateur de la Figure 5.4..

L'image reconstruite est une sinusoïde bidimensionnelle. La position du coefficient dans le spectre détermine son **orientation** et sa **fréquence spatiale**, tandis que sa magnitude et sa phase définissent, respectivement, son amplitude et son déplacement spatial. Ainsi, chaque coefficient de la TFD représente une composante sinusoïdale, et l'image originale peut être reconstruite par la somme de toutes ces composantes.

```
%%writefile tmp/fig_05_grade_2d.cpp
#define MM_OUT "tmp/fig_05_grade_2d.png"
//| label: fig-05-grade-2d
//| fig-cap: "Toute fréquence dans le spectre (point isolé) correspond à une onde sinusoïdale
2D rotatée dans le domaine spatial."
//| echo: true
//| output: true

#include <opencv2/opencv.hpp>
#include <vector>
#include <string>
#include <cmath>
#include "morph.hpp"
#include <filesystem>

// Helper: inverse fftshift (swap quadrants)
cv::Mat ifftshift(const cv::Mat& input) {
    int cx = input.cols / 2;
    int cy = input.rows / 2;
    cv::Mat output = input.clone();

    // Top-left quadrant
    cv::Mat q0(input, cv::Rect(0, 0, cx, cy));
    // Top-right quadrant
    cv::Mat q1(input, cv::Rect(cx, 0, input.cols - cx, cy));
    // Bottom-left quadrant
    cv::Mat q2(input, cv::Rect(0, cy, cx, input.rows - cy));
    // Bottom-right quadrant
    cv::Mat q3(input, cv::Rect(cx, cy, input.cols - cx, input.rows - cy));

    // Swap quadrants: TL->BR, TR->BL, BL->TR, BR->TL
    cv::Mat tmp;
    q0.copyTo(tmp);
    q3.copyTo(q0);
    tmp.copyTo(q3);

    q1.copyTo(tmp);
    q2.copyTo(q1);
    tmp.copyTo(q2);

    return output;
}

int main() {
    int N_grid = 100;
    cv::Mat espectro_vazio = cv::Mat::zeros(N_grid, N_grid, CV_64FC2);

    // Allumant un seul point (fréquence) hors du centre
    int u0 = 10, v0 = 5;
```

```
    espectro_vazio.at<cv::Vec2d>(N_grid/2 - v0, N_grid/2 - u0) = cv::Vec2d(1000, 0);

    // Retour au domaine spatial (IDFT)
    cv::Mat espectro_shifted = ifftshift(espectro_vazio);
    cv::Mat onda_2d_complex;
    cv::idft(espectro_shifted, onda_2d_complex, cv::DFT_SCALE | cv::DFT_COMPLEX_OUTPUT);

    // Extraire la partie réelle
    cv::Mat onda_2d_parts[2];
    cv::split(onda_2d_complex, onda_2d_parts);
    cv::Mat onda_2d = onda_2d_parts[0];

    // Normalisation pour visualisation
    cv::Mat onda_vis;
    cv::normalize(onda_2d, onda_vis, 0, 255, cv::NORM_MINMAX, CV_8U);

    // Magnitude du spectre
    cv::Mat espectro_mag[2];
    cv::split(espectro_vazio, espectro_mag);
    cv::Mat espectro_abs;
    cv::magnitude(espectro_mag[0], espectro_mag[1], espectro_abs);

    cv::Mat espectro_vis;
    cv::normalize(espectro_abs, espectro_vis, 0, 255, cv::NORM_MINMAX, CV_8U);

    // Accent visuel du point
    cv::Mat espectro_color;
    cv::cvtColor(espectro_vis, espectro_color, cv::COLOR_GRAY2BGR);
    cv::circle(espectro_color, cv::Point(N_grid/2 - u0, N_grid/2 - v0), 2, cv::Scalar(0, 0,
255), -1);

    mm::show(std::vector<mm::Image>{espectro_color, onda_vis},
             MM_OUT,
             std::vector<std::string>{"Spectre (1 point actif)", "Onda 2D Résultante (IDFT)"},
             2);


// [pdi:panel-io] auto-generated - do not edit by hand
std::filesystem::create_directories("tmp");
mm::write(espectro_color, "tmp/fig_05_grade_2d_0.png");
mm::write(onda_vis, "tmp/fig_05_grade_2d_1.png");
// [pdi:panel-io:end]
return 0;
}
```

```
Overwriting tmp/fig_05_grade_2d.cpp
```

```
!g++ -I. -std=c++17 -DMM_USE_OPENCV -I/usr/include/opencv4 tmp/fig_05_grade_2d.cpp -o
tmp/fig_05_grade_2d -lopencv_stitching -lopencv_alphamat -lopencv_aruco -lopencv_barcode
-lopencv_bgsegm -lopencv_bioinspired -lopencv_ccalib -lopencv_dnn_objdetect
-lopencv_dnn_superres -lopencv_dpm -lopencv_face -lopencv_freetype -lopencv_fuzzy -lopencv_hdf
-lopencv_hfs -lopencv_img_hash -lopencv_intensity_transform -lopencv_line_descriptor
-lopencv_mcc -lopencv_quality -lopencv_rapid -lopencv_reg -lopencv_rgbd -lopencv_saliency
-lopencv_shape -lopencv_stereo -lopencv_structured_light -lopencv_phase_unwrapping
-lopencv_superres -lopencv_optflow -lopencv_surface_matching -lopencv_tracking
-lopencv_highgui -lopencv_datasets -lopencv_text -lopencv_plot -lopencv_ml -lopencv_videostab
-lopencv_videoio -lopencv_viz -lopencv_wechat_qrcode -lopencv_ximgproc -lopencv_video
-lopencv_xobjdetect -lopencv_objdetect -lopencv_calib3d -lopencv_imgcodecs -lopencv_features2d
-lopencv_dnn -lopencv_flann -lopencv_xphoto -lopencv_photo -lopencv_imgproc -lopencv_core \
  && ./tmp/fig_05_grade_2d \
```

```
  && test -f "tmp/fig_05_grade_2d.png" \
  || echo "  mm::show não gravou tmp/fig_05_grade_2d.png"
```

```
[1] Spectre (1 point actif)
[2] Onde 2D Résultante (IDFT)
```

```
try:
    mm.show(
        [
            mm.read("tmp/fig_05_grade_2d_0.png"),
            mm.read("tmp/fig_05_grade_2d_1.png"),
        ],
        titles=[
            'Espectro (1 ponto ativo)',
            'Onda 2D Resultante (IDFT)',
        ],
        cols=2,
        figsize=(10, 4),
    )
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + " tmp/fig_05_grade_2d_0.png
    (ver a versao Python)")
```

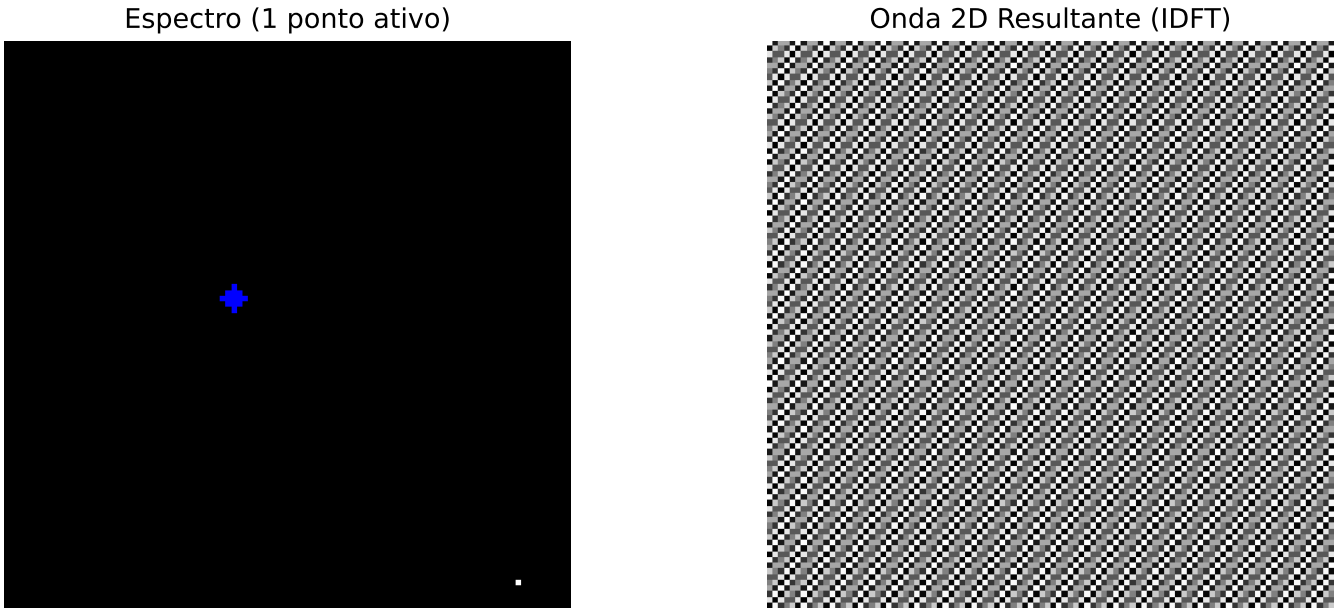


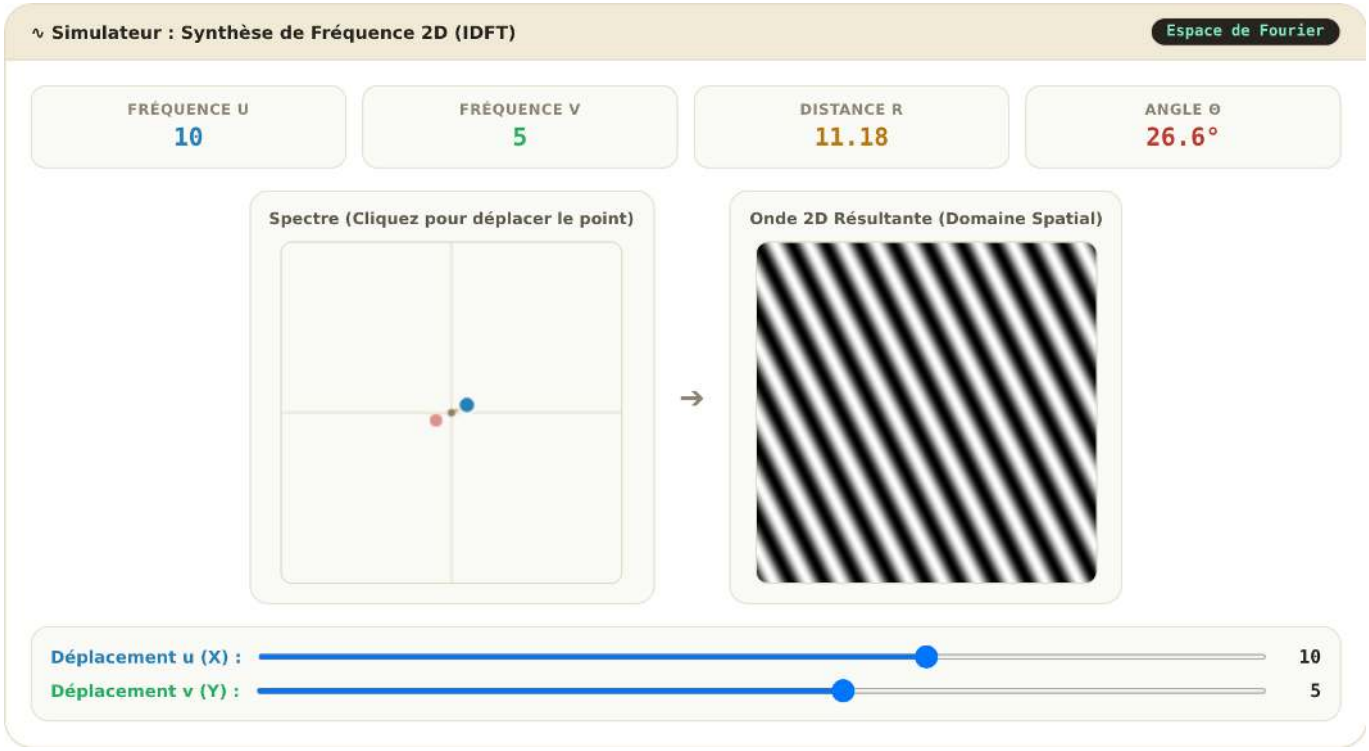
Figure 5.3: Toda frequência no espectro (ponto isolado) corresponde a uma onda senoidal 2D rotacionada no domínio espacial.

5.3.4 Définition mathématique

Considérons une image $f(x, y)$ de dimensions $M \times N$. Sa **Transformée de Fourier discrète 2D** (TFD) est définie par :

$$F(u, v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x, y)\, e^{-j2\pi\left(\frac{ux}{M} + \frac{vy}{N}\right)} \tag{5.1}$$

où $u = 0, 1, \ldots, M-1$ et $v = 0, 1, \ldots, N-1$ représentent les fréquences discrètes dans les directions horizontale et verticale, respectivement. Le terme exponentiel correspond à une sinusoïde bidimensionnelle, dont la



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.fr/cap05/sim-05-grade-2d.html

Figure 5.4: Simulateur interactif de la synthèse de Fourier 2D. Modifiez la position horizontale (u) et verticale (v) du coefficient dans le spectre de fréquences centré et observez comment la distance par rapport au centre dicte la fréquence spatiale (épaisseur) et l'angle dicte l'orientation de l'onde sinusoïdale générée.

fréquence et l'orientation sont déterminées par les indices (u, v).

La **Transformée de Fourier discrète inverse 2D** (TFDI) reconstruit l'image originale à partir de ses coefficients :

$$f(x,y) = \frac{1}{MN} \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} F(u,v)\, e^{j2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)} \tag{5.2}$$

Les équations Équation 5.1 et Équation 5.2 montrent que la TFD et la TFDI forment une paire de transformations : la première convertit l'image dans le domaine fréquentiel, tandis que la seconde reconstruit exactement l'image originale à partir de ses coefficients.

i À propos du symbole j

Le terme j désigne l'**unité imaginaire**, définie par $j^2 = -1$. En ingénierie et en traitement du signal, on adopte j plutôt que i afin d'éviter toute confusion avec la notation du courant électrique. Son utilisation dans l'exponentielle complexe, régie par la formule d'Euler ($e^{j\theta} = \cos\theta + j\sin\theta$), permet de représenter de manière compacte l'amplitude et la phase de chaque fréquence spatiale présente dans l'image.

i Qu'est-ce que la composante DC ?

Le coefficient $F(0,0)$, appelé **composante DC** (*courant continu*), est égal à la somme des intensités de tous les pixels de l'image (voir Figure 5.5) :

$$F(0,0) = MN\,\bar{f},$$

où $\bar{f}$ est l'intensité moyenne de l'image. Ainsi, la composante DC représente le niveau moyen d'intensité

et, dans la plupart des images naturelles, possède la plus grande amplitude du spectre.
Les autres coefficients représentent les variations autour de cette moyenne. Après application du **décalage de FFT**, la composante DC est déplacée vers le centre du spectre, concentrant les basses fréquences dans la région centrale et les hautes fréquences sur les bords.

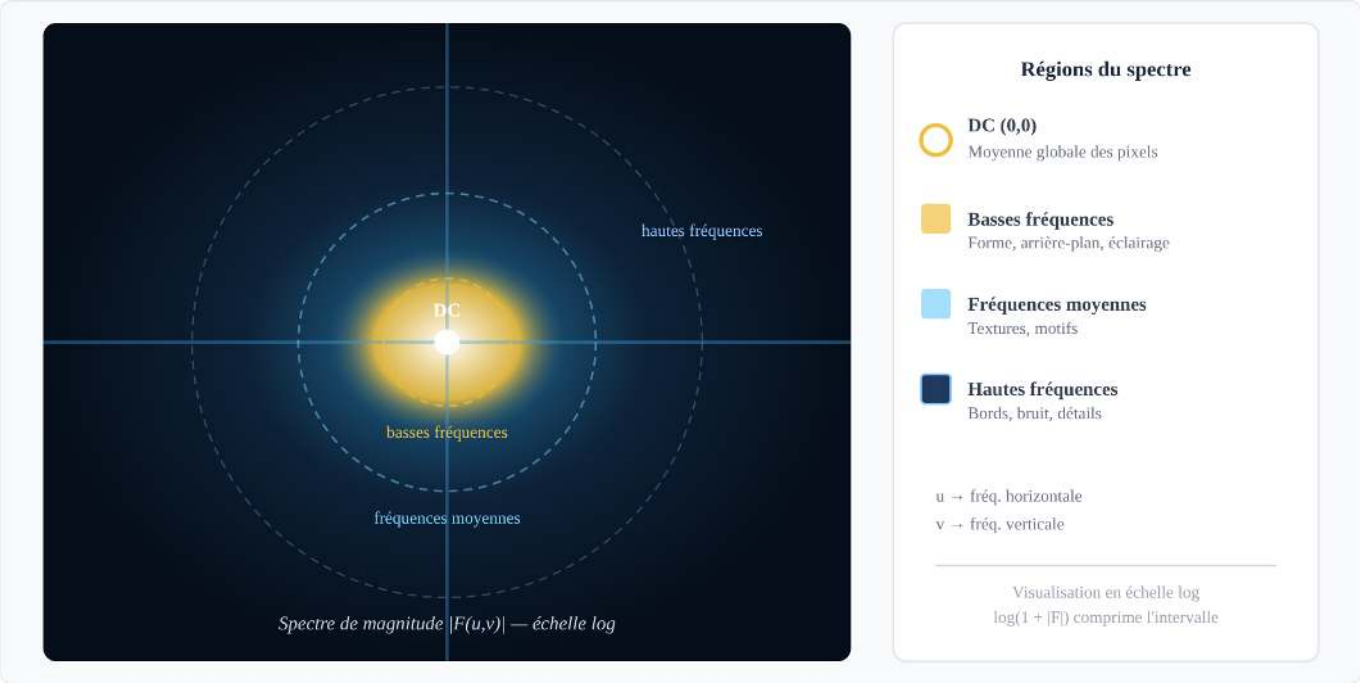


Figure 5.5: Diagramme conceptuel du spectre de Fourier 2D centré.

5.3.5 Magnitude et Phase

Chaque coefficient de la Transformée de Fourier Discrète (TFD) est un nombre complexe et peut s'écrire sous la forme

$$F(u,v) = R(u,v) + j\, I(u,v),$$

où $R(u,v)$ et $I(u,v)$ correspondent respectivement aux parties **réelle** et **imaginaire** du coefficient. À partir de l'équation Équation 5.1, on obtient

$$R(u,v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y) \cos\left(2\pi \left(\frac{ux}{M} + \frac{vy}{N}\right)\right),$$

et

$$I(u,v) = -\sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y) \sin\left(2\pi \left(\frac{ux}{M} + \frac{vy}{N}\right)\right).$$

À partir de cette représentation, on définit deux grandeurs fondamentales :

- **Magnitude**, qui indique l'intensité de la composante fréquentielle,

$$|F(u,v)| = \sqrt{R(u,v)^2 + I(u,v)^2};$$

- **Phase**, qui détermine l'alignement (ou le décalage) spatial de la composante,

$$\phi(u,v) = \operatorname{atan2}(I(u,v),\, R(u,v))\,.$$

Ainsi, chaque coefficient peut également être écrit sous sa forme polaire,

$$F(u,v) = |F(u,v)|\, e^{j\phi(u,v)}.$$

Le spectre de Fourier peut donc être visualisé au moyen de deux images distinctes : le **spectre de magnitude**, généralement utilisé pour analyser la distribution des fréquences, et le **spectre de phase**, qui décrit l'organisation spatiale des composantes sinusoïdales.

Bien que le spectre de magnitude soit le plus utilisé pour l'inspection visuelle, la phase contient une grande partie des informations structurelles de l'image. La combinaison de la magnitude et de la phase permet de reconstruire exactement l'image originale au moyen de la TFD inverse.

5.3.6 O que a Magnitude e a Fase carregam?

5.3.7 Ce que portent la magnitude et la phase ?

Une démonstration classique consiste à combiner la magnitude d'une image avec la phase d'une autre et à reconstruire le résultat. Cette expérience met en évidence que :

- **La phase** préserve la structure spatiale de l'image, y compris la position des objets, leurs contours et leur géométrie. De petites modifications de la phase peuvent provoquer de grands changements visuels.
- **La magnitude** contrôle la manière dont l'énergie est distribuée entre les fréquences spatiales, influençant principalement le contraste et la texture.

Lorsqu'une image est reconstruite avec la magnitude de A et la phase de B, le résultat tend à **ressembler davantage à B qu'à A**, ce qui montre que la phase est le principal composant responsable de l'organisation spatiale de la scène. Cependant, la magnitude reste importante, car elle module le contraste des structures reconstruites. Ainsi, une reconstruction fidèle dépend de la combinaison cohérente entre magnitude et phase.

Un exemple de ce comportement est présenté dans la Figure 5.6.

> **i** Analogie avec l'audio : limites et précautions
>
> La phase d'un signal joue des rôles distincts dans l'audio et les images :
> - **Audio stéréo ou multicanal :** la phase relative entre les canaux est fondamentale pour la perception de la position des sources sonores, via les différences interaurales de temps (ITD, *Interaural Time Differences*).
> - **Audio monaural :** la phase absolue exerce peu d'influence perceptuelle directe.
> - **Images (TFD) :** la phase est le principal facteur responsable de l'organisation spatiale de la scène, tandis que la magnitude module le contraste et la distribution de l'énergie entre les fréquences.
>
> Dans les deux domaines, la **magnitude** est liée à l'intensité des composantes de fréquence : en audio, elle influence le timbre et l'intensité perçue ; en images, elle influence le contraste et la texture.

```
%%writefile tmp/fig_05_dft_intro.cpp
#define MM_OUT "tmp/fig_05_dft_intro.png"
//| label: fig-05-dft-intro
//| fig-cap: "Experimento de troca de fase: Imagem A (moedas) e Imagem B (padrão geométrico)
reconstruídas com magnitudes e fases trocadas. O resultado mostra que a estrutura visual é
**muito mais sensível à fase** do que à magnitude: quando a fase de B é mantida, a imagem
resultante preserva a organização espacial de B, mesmo com a magnitude de A. A magnitude, por
sua vez, influencia principalmente o contraste e a textura. Observe que a qualidade da
reconstrução não é perfeita - há artefatos visíveis -, evidenciando a interdependência entre
fase e magnitude para uma representação fiel da imagem."
//| echo: true
```

```
//| output: true

#include <opencv2/opencv.hpp>
#include <opencv2/core.hpp>
#include <opencv2/imgproc.hpp>
#include <opencv2/highgui.hpp>
#include <iostream>
#include <string>
#include <vector>
#include <cmath>
#include <complex>
#include <filesystem>
#include <algorithm>
#include "morph.hpp"

// Helper para trocar quadrantes (equivalente a np.fft.fftshift em 2D)
static cv::Mat fftShift(const cv::Mat& mat) {
    int cx = mat.cols / 2;
    int cy = mat.rows / 2;
    cv::Mat output;
    cv::Mat q0(mat, cv::Rect(0, 0, cx, cy));
    cv::Mat q1(mat, cv::Rect(cx, 0, mat.cols - cx, cy));
    cv::Mat q2(mat, cv::Rect(0, cy, cx, mat.rows - cy));
    cv::Mat q3(mat, cv::Rect(cx, cy, mat.cols - cx, mat.rows - cy));
    cv::Mat tmp;
    q0.copyTo(tmp);
    q3.copyTo(q0);
    tmp.copyTo(q3);
    q1.copyTo(tmp);
    q2.copyTo(q1);
    tmp.copyTo(q2);
    mat.copyTo(output);
    return output;
}

// Helper para construir imagem a partir de magnitude e fase
static cv::Mat buildFromMagPhase(const cv::Mat& mag, const cv::Mat& phase) {
    cv::Mat magF, phaseF;
    mag.convertTo(magF, CV_64F);
    phase.convertTo(phaseF, CV_64F);

    // Calcular cos e sin da fase
    cv::Mat cosPhase, sinPhase;
    cv::Mat phaseArr[] = {phaseF, phaseF};
    cv::Mat cosArr[] = {cosPhase, sinPhase};

    // Aplicar cos e sin elemento a elemento via loop
    cosPhase = cv::Mat(phaseF.size(), CV_64F);
    sinPhase = cv::Mat(phaseF.size(), CV_64F);
    for (int i = 0; i < phaseF.rows; i++) {
        for (int j = 0; j < phaseF.cols; j++) {
            double p = phaseF.at<double>(i, j);
            cosPhase.at<double>(i, j) = std::cos(p);
            sinPhase.at<double>(i, j) = std::sin(p);
        }
    }

    // real = mag * cos(phase), imag = mag * sin(phase)
    cv::Mat realPart = magF.mul(cosPhase);
    cv::Mat imagPart = magF.mul(sinPhase);
```

```
    // Construir complexa: real + i*imag
    cv::Mat complexParts[] = {realPart, imagPart};
    cv::Mat complexMat;
    cv::merge(complexParts, 2, complexMat);

    // IDFT
    cv::Mat result;
    cv::idft(complexMat, result, cv::DFT_SCALE | cv::DFT_REAL_OUTPUT);

    // Converter para 8-bit
    cv::Mat result8u;
    result.convertTo(result8u, CV_8U);

    return result8u;
}

int main() {
    //   Experimento: A Importância da Fase
    //   Carregamento da imagem
    std::string url     =
"https://upload.wikimedia.org/wikipedia/commons/2/25/GAZI.MD.AHAD_11.jpg";
    std::string caminho = "imagens/coins.jpg";

    mm::Image img_obj;
    if (!std::filesystem::exists(caminho)) {
        std::filesystem::create_directories("imagens");
        img_obj = mm::read(url);
        mm::write(img_obj, caminho);
    } else {
        img_obj = mm::read(caminho);
    }

    cv::Mat img_color = img_obj;
    if (img_color.channels() == 3) {
        cv::Mat temp;
        cv::cvtColor(img_color, temp, cv::COLOR_BGR2GRAY);
        img_color = temp;
    }
    mm::Image img_gray_mm = mm::gray(img_obj);
    cv::Mat img_gray = img_gray_mm;

    cv::Mat img_a;
    cv::resize(img_gray, img_a, cv::Size(400, 400));

    // Criar uma imagem B sintética (padrão geométrico)
    cv::Mat img_b = cv::Mat::zeros(400, 400, CV_8UC1);
    cv::rectangle(img_b, cv::Point(100, 100), cv::Point(300, 300), cv::Scalar(255), -1);
    cv::circle(img_b, cv::Point(200, 200), 150, cv::Scalar(128), 10);

    // FFT de A e B
    cv::Mat img_a_f;
    img_a.convertTo(img_a_f, CV_64F);
    cv::Mat img_b_f;
    img_b.convertTo(img_b_f, CV_64F);

    cv::Mat FA, FB;
    cv::dft(img_a_f, FA, cv::DFT_COMPLEX_OUTPUT);
    cv::dft(img_b_f, FB, cv::DFT_COMPLEX_OUTPUT);

    // Separar magnitude e fase de FA
    cv::Mat FA_parts[2];
```

```
    cv::split(FA, FA_parts);
    cv::Mat magA, phaseA;
    cv::magnitude(FA_parts[0], FA_parts[1], magA);
    cv::phase(FA_parts[0], FA_parts[1], phaseA);

    // Separar magnitude e fase de FB
    cv::Mat FB_parts[2];
    cv::split(FB, FB_parts);
    cv::Mat magB, phaseB;
    cv::magnitude(FB_parts[0], FB_parts[1], magB);
    cv::phase(FB_parts[0], FB_parts[1], phaseB);

    // Troca de Fase: rec_A_mag_B_fase = |FA| * exp(i * phase(FB))
    cv::Mat rec_A_mag_B_fase = buildFromMagPhase(magA, phaseB);

    // Troca de Fase: rec_B_mag_A_fase = |FB| * exp(i * phase(FA))
    cv::Mat rec_B_mag_A_fase = buildFromMagPhase(magB, phaseA);

    // Converter para mm::Image para exibição
    mm::Image img_a_mm = img_a;
    mm::Image img_b_mm = img_b;
    mm::Image rec_A_mm = rec_A_mag_B_fase;
    mm::Image rec_B_mm = rec_B_mag_A_fase;

    // Manter a variável img_gray como mm::Image no final (para persistência)
    img_gray = img_gray_mm;

    // Exibir resultados
    mm::show(
        std::vector<mm::Image>{img_a_mm, img_b_mm, rec_A_mm, rec_B_mm},
        MM_OUT,
        std::vector<std::string>{"Imagem A", "Imagem B", "Mag(A) + Fase(B)", "Mag(B) +
Fase(A)"},
        4
    );

    // Exibir mensagem
    std::cout << "  A fase preserva bordas e contornos; a magnitude controla contraste e" <<
std::endl;
    std::cout << "textura. Em áudio estéreo, a fase afeta a localização espacial; em" <<
std::endl;
    std::cout << "imagens, determina a organização da cena." << std::endl;


// [pdi:state-io] auto-generated - do not edit by hand
std::filesystem::create_directories("tmp/state");
mm::write(img_gray, "tmp/state/img_gray_20.png");
// [pdi:state-io:end]

// [pdi:panel-io] auto-generated - do not edit by hand
std::filesystem::create_directories("tmp");
mm::write(img_a, "tmp/fig_05_dft_intro_0.png");
mm::write(img_b, "tmp/fig_05_dft_intro_1.png");
mm::write(rec_A_mag_B_fase, "tmp/fig_05_dft_intro_2.png");
mm::write(rec_B_mag_A_fase, "tmp/fig_05_dft_intro_3.png");
// [pdi:panel-io:end]
return 0;
}
```

```
Overwriting tmp/fig_05_dft_intro.cpp
```

```
!g++ -I. -std=c++17 -DMM_USE_OPENCV -I/usr/include/opencv4 tmp/fig_05_dft_intro.cpp -o
tmp/fig_05_dft_intro -lopencv_stitching -lopencv_alphamat -lopencv_aruco -lopencv_barcode
-lopencv_bgsegm -lopencv_bioinspired -lopencv_ccalib -lopencv_dnn_objdetect
-lopencv_dnn_superres -lopencv_dpm -lopencv_face -lopencv_freetype -lopencv_fuzzy -lopencv_hdf
-lopencv_hfs -lopencv_img_hash -lopencv_intensity_transform -lopencv_line_descriptor
-lopencv_mcc -lopencv_quality -lopencv_rapid -lopencv_reg -lopencv_rgbd -lopencv_saliency
-lopencv_shape -lopencv_stereo -lopencv_structured_light -lopencv_phase_unwrapping
-lopencv_superres -lopencv_optflow -lopencv_surface_matching -lopencv_tracking
-lopencv_highgui -lopencv_datasets -lopencv_text -lopencv_plot -lopencv_ml -lopencv_videostab
-lopencv_videoio -lopencv_viz -lopencv_wechat_qrcode -lopencv_ximgproc -lopencv_video
-lopencv_xobjdetect -lopencv_objdetect -lopencv_calib3d -lopencv_imgcodecs -lopencv_features2d
-lopencv_dnn -lopencv_flann -lopencv_xphoto -lopencv_photo -lopencv_imgproc -lopencv_core \
  && ./tmp/fig_05_dft_intro \
  && test -f "tmp/fig_05_dft_intro.png" \
  || echo "  mm::show não gravou tmp/fig_05_dft_intro.png"
```

```
[1] Imagem A
[2] Imagem B
[3] Mag(A) + Fase(B)
[4] Mag(B) + Fase(A)
 A fase preserva bordas e contornos; a magnitude controla contraste e
textura. Em áudio estéreo, a fase afeta a localização espacial; em
imagens, determina a organização da cena.
```

```
try:
    mm.show(
        [
            mm.read("tmp/fig_05_dft_intro_0.png"),
            mm.read("tmp/fig_05_dft_intro_1.png"),
            mm.read("tmp/fig_05_dft_intro_2.png"),
            mm.read("tmp/fig_05_dft_intro_3.png"),
        ],
        titles=[
            'Imagem A',
            'Imagem B',
            'Mag(A) + Fase(B)',
            'Mag(B) + Fase(A)',
        ],
        cols=4,
        figsize=(16, 4),
    )
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + " tmp/fig_05_dft_intro_0.png
    (ver a versao Python)")
```

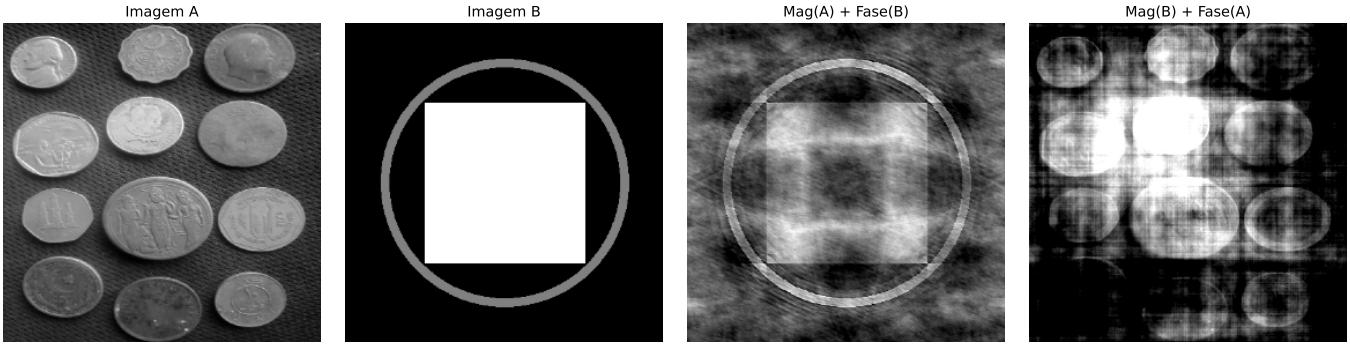


Figure 5.6: Experimento de troca de fase: Imagem A (moedas) e Imagem B (padrão geométrico) reconstruídas com magnitudes e fases trocadas. O resultado mostra que a estrutura visual é **muito mais sensível à fase** do que à magnitude: quando a fase de B é mantida, a imagem resultante preserva a organização espacial de B, mesmo com a magnitude de A. A magnitude, por sua vez, influencia principalmente o contraste e a textura. Observe que a qualidade da reconstrução não é perfeita — há artefatos visíveis —, evidenciando a interdependência entre fase e magnitude para uma representação fiel da imagem.

5.4 Théorème de convolution et stratégies de filtrage

Le **théorème de convolution** établit une relation fondamentale entre les domaines spatial et fréquentiel :

$$f(x,y) \circledast h(x,y) \overset{\mathcal{F}}{\longleftrightarrow} F(u,v)\,H(u,v) \tag{5.3}$$

où $\circledast$ représente la **convolution circulaire discrète**. Ainsi, la convolution entre une image $f(x,y)$ et un filtre $h(x,y)$ peut être remplacée par la multiplication de leurs spectres.

En pratique, pour obtenir le même résultat que la convolution linéaire effectuée dans le domaine spatial, on applique un **remplissage par zéros** (*zero-padding*) avant la Transformée de Fourier rapide (FFT), afin d'éviter les artefacts sur les bords de l'image.

Cependant, le filtrage dans le domaine fréquentiel n'est pas toujours l'alternative la plus efficace. Pour des filtres tels que le filtre gaussien et le filtre moyenneur (*Box Filter*), la propriété de **séparabilité** permet de réduire considérablement le coût computationnel de la convolution dans le domaine spatial.

5.4.1 *Noyau* séparable vs. non séparable

Un ***noyau* séparable** peut être écrit comme le produit externe de deux vecteurs unidimensionnels,

$$H = v\,h^T,$$

ce qui permet de remplacer la convolution bidimensionnelle par deux convolutions unidimensionnelles successives : l'une dans la direction horizontale et l'autre dans la direction verticale.

En revanche, un ***noyau* non séparable** n'admet pas cette décomposition et, par conséquent, sa convolution doit être réalisée directement sur le voisinage bidimensionnel.

En pratique, pour un *noyau* de dimension $K \times K$, la convolution directe exige K^2 multiplications par pixel, tandis qu'un *noyau* séparable ne requiert que $2K$ multiplications, réduisant ainsi considérablement le coût computationnel.

5.4.2 Analyse de l'efficacité computationnelle

Considérons une image de dimensions $M \times N$ et un filtre carré de taille $K \times K$. La Table 5.2 compare la complexité des principales stratégies de filtrage.

Tableau 5.2: Comparaison de la complexité de la convolution directe, séparable et via la Transformée de Fourier rapide (FFT).

Méthode de filtrage	Complexité asymptotique	Dépendance de K	Application typique
Spatiale non séparable	$\mathcal{O}(MNK^2)$	Quadratique	*Kernels* petits et non séparables
Spatiale séparable	$\mathcal{O}(MNK)$	Linéaire	Filtres gaussien et de moyenne
Via FFT	$\mathcal{O}(MN\log(MN))$	Indépendante de K	*Kernels* grands

Pour de petits *kernels*, la convolution spatiale, surtout lorsque le filtre est séparable, s'avère généralement plus efficace en raison du faible coût des opérations. À mesure que la taille du *kernel* augmente, le filtrage via FFT devient plus avantageux, car son coût ne dépend pratiquement pas de la dimension du filtre.

5.4.3 Discussion des résultats expérimentaux

Le graphique obtenu lors de l'essai avec l'image des pièces (2560×1920), présenté à la Figure 5.7, confirme le comportement prédit par l'analyse de complexité computationnelle.

1. **Convolution non séparable ($\mathcal{O}(MNK^2)$)**
   La convolution directe présente une croissance quadratique avec la taille du *kernel*. Pour de petites valeurs de K, le coût est faible, mais il augmente rapidement à mesure que le *kernel* grandit, devenant irréalisable pour des applications en temps réel.
2. **Filtrage par FFT ($\mathcal{O}(MN\log(MN))$)**
   Le coût de la FFT ne dépend que de la taille de l'image, étant indépendant de K. Par conséquent, ses performances restent approximativement constantes lorsque le *kernel* varie, ce qui la rend avantageuse pour les filtres de grande taille ou non séparables.
3. **Convolution séparable ($\mathcal{O}(MNK)$)**
   La décomposition du *kernel* en deux filtres unidimensionnels réduit significativement le coût computationnel. En pratique, cette approche tend à être la plus efficace pour les filtres séparables, en particulier dans les implémentations optimisées.

En général, le choix de la méthode dépend de la taille et de la structure du *kernel*. Les filtres séparables sont plus efficaces dans le domaine spatial, tandis que la FFT devient plus avantageuse pour les *kernels* de grande taille ou pour de multiples convolutions dans le domaine fréquentiel.

$$g = \mathcal{F}^{-1}[\mathcal{F}(f)\cdot\mathcal{F}(h)] \quad \text{(FFT)} \qquad g = f \circledast h \quad \text{(convolution directe)} \qquad g = (f \circledast v) \circledast h^T \quad \text{(séparable)} \tag{5.4}$$

où :

- $f(x, y)$ représente l'image d'entrée ;
- $h(x, y)$ est le *kernel* bidimensionnel du filtre ;
- v et h^T sont, respectivement, les vecteurs vertical et horizontal qui composent le *kernel* séparable.

```
%%writefile tmp/fig_05_conv_eficiencia.cpp
#define MM_OUT "tmp/fig_05_conv_eficiencia.png"
//| label: fig-05-conv-eficiencia
//| fig-cap: "Comparação de eficiência: Convolução Não Separável (Espacial 2D), Separável
(Espacial 1D) e via FFT."
//| echo: false
//| output: true
```

```cpp
#include <iostream>
#include <vector>
#include <string>
#include <cmath>
#include "morph.hpp"
#include <filesystem>

int main() {
    // Trilha C++: curvas de CUSTO relativo (contagem de operacoes) - a forma das
    // curvas (K^2 vs 2K vs log N) e o que importa; a trilha py mede tempos reais.
    std::vector<double> K = {3, 7, 11, 15, 21, 31, 41, 51};
    double N2 = 256.0 * 256.0;

    // t_nao_sep = N2 * K^2 / 1e6
    std::vector<double> t_nao_sep(K.size());
    // t_sep = N2 * 2 * K / 1e6
    std::vector<double> t_sep(K.size());
    // t_fft = constante (N2 * log2(N2) / 1e6)
    double fft_val = N2 * std::log2(N2) / 1e6;
    std::vector<double> t_fft(K.size(), fft_val);

    for (size_t i = 0; i < K.size(); ++i) {
        t_nao_sep[i] = N2 * K[i] * K[i] / 1e6;
        t_sep[i]     = N2 * 2.0 * K[i] / 1e6;
    }

    // Construção do gráfico de linhas com mm::lineChart (aceita vetores de vetores)
    std::vector<std::vector<double>> xs = {K, K, K};
    std::vector<std::vector<double>> ys = {t_nao_sep, t_sep, t_fft};
    std::vector<std::string> labels = {
        "Nao Separavel  O(N^2 K^2)",
        "Separavel  O(N^2 . 2K)",
        "Via FFT  O(N^2 log N)"
    };

    mm::Image chart = mm::lineChart(
        xs, ys, labels,
        {}, // colors vazio - usa o padrão
        "Custo relativo: Espacial vs Frequencia",
        "Tamanho do kernel  K",
        "Operacoes  (x10^6)"
    );

    mm::show(std::vector<mm::Image>{chart}, MM_OUT, {"Comparacao de complexidade"}, 1);


// [pdi:panel-io] auto-generated - do not edit by hand
std::filesystem::create_directories("tmp");
mm::write(chart, "tmp/fig_05_conv_eficiencia_0.png");
// [pdi:panel-io:end]
return 0;
}
```

```
Overwriting tmp/fig_05_conv_eficiencia.cpp
```

```
!g++ -I. -std=c++17 -DMM_USE_OPENCV -I/usr/include/opencv4 tmp/fig_05_conv_eficiencia.cpp -o
tmp/fig_05_conv_eficiencia -lopencv_stitching -lopencv_alphamat -lopencv_aruco
-lopencv_barcode -lopencv_bgsegm -lopencv_bioinspired -lopencv_ccalib -lopencv_dnn_objdetect
-lopencv_dnn_superres -lopencv_dpm -lopencv_face -lopencv_freetype -lopencv_fuzzy -lopencv_hdf
-lopencv_hfs -lopencv_img_hash -lopencv_intensity_transform -lopencv_line_descriptor
-lopencv_mcc -lopencv_quality -lopencv_rapid -lopencv_reg -lopencv_rgbd -lopencv_saliency
-lopencv_shape -lopencv_stereo -lopencv_structured_light -lopencv_phase_unwrapping
-lopencv_superres -lopencv_optflow -lopencv_surface_matching -lopencv_tracking
-lopencv_highgui -lopencv_datasets -lopencv_text -lopencv_plot -lopencv_ml -lopencv_videostab
-lopencv_videoio -lopencv_viz -lopencv_wechat_qrcode -lopencv_ximgproc -lopencv_video
-lopencv_xobjdetect -lopencv_objdetect -lopencv_calib3d -lopencv_imgcodecs -lopencv_features2d
-lopencv_dnn -lopencv_flann -lopencv_xphoto -lopencv_photo -lopencv_imgproc -lopencv_core \
  && ./tmp/fig_05_conv_eficiencia \
  && test -f "tmp/fig_05_conv_eficiencia.png" \
  || echo "  mm::show não gravou tmp/fig_05_conv_eficiencia.png"
```

```
[1] Comparacao de complexidade
```

```
try:
    mm.show(
        [
            mm.read("tmp/fig_05_conv_eficiencia_0.png"),
        ],
        titles=[
            'Comparacao de complexidade',
        ],
        cols=1,
    )
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + "
    tmp/fig_05_conv_eficiencia_0.png (ver a versao Python)")
```

Figure 5.7: Comparação de eficiência: Convolução Não Separável (Espacial 2D), Separável (Espacial 1D) e via FFT.

! Le problème de la convolution circulaire (*wrap-around*)

La Transformée de Fourier Discrète (TFD) suppose que l'image est **périodiquement étendue dans l'espace**, c'est-à-dire que ses bords se répètent indéfiniment.
Dans cette condition, la multiplication dans le domaine fréquentiel correspond à une **convolution circulaire** dans le domaine spatial. Par conséquent, des régions opposées de l'image (haut et bas, gauche et droite) interagissent artificiellement, comme illustré dans la Figure 5.8.
L'application d'un *zero-padding* avant la FFT réduit cet effet en étendant l'image avec des valeurs nulles sur les bords, rapprochant ainsi le résultat de la convolution linéaire. Ce comportement peut être interprété à la lumière du Théorème de Convolution, présenté dans la Figure 5.9.

```
%%writefile tmp/fig_05_padding_error.cpp
#define MM_OUT "tmp/fig_05_padding_error.png"
//| label: fig-05-padding-error
//| fig-cap: "Sem *padding*, um deslocamento severo faz a imagem vazar para o lado oposto
(convolução circular)."
//| echo: true
//| output: true

#include <opencv2/opencv.hpp>
#include <complex>
#include <vector>
#include <string>
#include <cmath>
#include "morph.hpp"
#include <filesystem>

int main() {
// [pdi:state-io] auto-generated - do not edit by hand
mm::Image img_gray = mm::_read_state("tmp/state/img_gray_20.png");
// [pdi:state-io:end]

    // img_gray is provided (mm::Image)
    int M = img_gray.h;
    int N = img_gray.w;

    // Converter para CV_64F para cálculos complexos
    cv::Mat img_gray_mat(img_gray.h, img_gray.w, CV_8UC1, img_gray.data.data());
    cv::Mat img_gray_float;
    img_gray_mat.convertTo(img_gray_float, CV_64F);

    // Simulação de um filtro de deslocamento brutal
    cv::Mat H_shift(M, N, CV_64FC2, cv::Scalar(0, 0));
    for (int u = 0; u < M; ++u) {
        for (int v = 0; v < N; ++v) {
            double angle = -2.0 * M_PI * (u * 120.0 / M + v * 120.0 / N);
            H_shift.at<cv::Vec2d>(u, v) = cv::Vec2d(std::cos(angle), std::sin(angle));
        }
    }

    // Filtragem SEM padding (causa o wrap-around)
    cv::Mat F_img;
    cv::dft(img_gray_float, F_img, cv::DFT_COMPLEX_OUTPUT);

    // Multiplicação no domínio da frequência
    cv::Mat produto;
    cv::mulSpectrums(F_img, H_shift, produto, 0);

    cv::Mat img_vazada;
```

```
    cv::idft(produto, img_vazada, cv::DFT_SCALE | cv::DFT_REAL_OUTPUT);

    // Normalizar para visualização
    cv::Mat img_vazada_norm;
    cv::normalize(img_vazada, img_vazada_norm, 0, 255, cv::NORM_MINMAX, CV_8U);

    // Converter para mm::Image
    mm::Image img_vazada_vis(img_vazada_norm);

    // Mostrar resultados
    mm::show(std::vector<mm::Image>{img_gray, img_vazada_vis},
             MM_OUT,
             std::vector<std::string>{"Original", "Filtragem s/ Padding (Vazamento)"},
             2);


// [pdi:panel-io] auto-generated - do not edit by hand
std::filesystem::create_directories("tmp");
mm::write(img_gray, "tmp/fig_05_padding_error_0.png");
mm::write(img_vazada_vis, "tmp/fig_05_padding_error_1.png");
// [pdi:panel-io:end]
return 0;
}
```

```
Overwriting tmp/fig_05_padding_error.cpp
```

```
!g++ -I. -std=c++17 -DMM_USE_OPENCV -I/usr/include/opencv4 tmp/fig_05_padding_error.cpp -o
tmp/fig_05_padding_error -lopencv_stitching -lopencv_alphamat -lopencv_aruco -lopencv_barcode
-lopencv_bgsegm -lopencv_bioinspired -lopencv_ccalib -lopencv_dnn_objdetect
-lopencv_dnn_superres -lopencv_dpm -lopencv_face -lopencv_freetype -lopencv_fuzzy -lopencv_hdf
-lopencv_hfs -lopencv_img_hash -lopencv_intensity_transform -lopencv_line_descriptor
-lopencv_mcc -lopencv_quality -lopencv_rapid -lopencv_reg -lopencv_rgbd -lopencv_saliency
-lopencv_shape -lopencv_stereo -lopencv_structured_light -lopencv_phase_unwrapping
-lopencv_superres -lopencv_optflow -lopencv_surface_matching -lopencv_tracking
-lopencv_highgui -lopencv_datasets -lopencv_text -lopencv_plot -lopencv_ml -lopencv_videostab
-lopencv_videoio -lopencv_viz -lopencv_wechat_qrcode -lopencv_ximgproc -lopencv_video
-lopencv_xobjdetect -lopencv_objdetect -lopencv_calib3d -lopencv_imgcodecs -lopencv_features2d
-lopencv_dnn -lopencv_flann -lopencv_xphoto -lopencv_photo -lopencv_imgproc -lopencv_core \
  && ./tmp/fig_05_padding_error \
  && test -f "tmp/fig_05_padding_error.png" \
  || echo "  mm::show não gravou tmp/fig_05_padding_error.png"
```

```
[1] Original
[2] Filtragem s/ Padding (Vazamento)
```

```
try:
    mm.show(
        [
            mm.read("tmp/fig_05_padding_error_0.png"),
            mm.read("tmp/fig_05_padding_error_1.png"),
        ],
        titles=[
            'Original',
            'Filtragem s/ Padding (Vazamento)',
        ],
        cols=2,
        figsize=(10, 4),
    )
except Exception as _e:
```

```
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + "
    tmp/fig_05_padding_error_0.png (ver a versao Python)")
```

Figure 5.8: Sem *padding*, um deslocamento severo faz a imagem vazar para o lado oposto (convolução circular).

```
%%writefile tmp/fig_05_conv_teorema.cpp
#define MM_OUT "tmp/fig_05_conv_teorema.png"
//| label: fig-05-conv-teorema
//| fig-cap: "Teorema da Convolution : filtrer dans le domaine spatial (Gaussienne) équivaut à
multiplier le spectre par H(u,v) en fréquence. Les sorties coïncident visuellement."
//| echo: true
//| output: true

#include <opencv2/opencv.hpp>
#include <vector>
#include <string>
#include "morph.hpp"
#include <filesystem>

int main() {
// [pdi:state-io] auto-generated - do not edit by hand
mm::Image img_gray = mm::_read_state("tmp/state/img_gray_20.png");
// [pdi:state-io:end]

    // Même lissage par deux chemins : espace vs. fréquence.
    int M = img_gray.h;
    int N = img_gray.w;
    cv::Mat H = mm::gaussFilter(M, N, 30);      // H(u,v) : transfert passe-bas gaussien
    mm::Image f_freq = mm::freqFilter(img_gray, H);  // convolution via multiplication en
fréquence
    mm::Image f_esp = mm::gaussian(img_gray, 31, 5); // même lissage réalisé dans l'espace

    mm::show(
        {img_gray, f_esp, f_freq},
        MM_OUT,
```

```
        {"Original", "Espace: Gaussienne", "Frequence: H(u,v).F(u,v)"},
        3
    );


// [pdi:panel-io] auto-generated - do not edit by hand
std::filesystem::create_directories("tmp");
mm::write(img_gray, "tmp/fig_05_conv_teorema_0.png");
mm::write(f_esp, "tmp/fig_05_conv_teorema_1.png");
mm::write(f_freq, "tmp/fig_05_conv_teorema_2.png");
// [pdi:panel-io:end]
return 0;
}
```

```
Overwriting tmp/fig_05_conv_teorema.cpp
```

```
!g++ -I. -std=c++17 -DMM_USE_OPENCV -I/usr/include/opencv4 tmp/fig_05_conv_teorema.cpp -o
tmp/fig_05_conv_teorema -lopencv_stitching -lopencv_alphamat -lopencv_aruco -lopencv_barcode
-lopencv_bgsegm -lopencv_bioinspired -lopencv_ccalib -lopencv_dnn_objdetect
-lopencv_dnn_superres -lopencv_dpm -lopencv_face -lopencv_freetype -lopencv_fuzzy -lopencv_hdf
-lopencv_hfs -lopencv_img_hash -lopencv_intensity_transform -lopencv_line_descriptor
-lopencv_mcc -lopencv_quality -lopencv_rapid -lopencv_reg -lopencv_rgbd -lopencv_saliency
-lopencv_shape -lopencv_stereo -lopencv_structured_light -lopencv_phase_unwrapping
-lopencv_superres -lopencv_optflow -lopencv_surface_matching -lopencv_tracking
-lopencv_highgui -lopencv_datasets -lopencv_text -lopencv_plot -lopencv_ml -lopencv_videostab
-lopencv_videoio -lopencv_viz -lopencv_wechat_qrcode -lopencv_ximgproc -lopencv_video
-lopencv_xobjdetect -lopencv_objdetect -lopencv_calib3d -lopencv_imgcodecs -lopencv_features2d
-lopencv_dnn -lopencv_flann -lopencv_xphoto -lopencv_photo -lopencv_imgproc -lopencv_core \
  && ./tmp/fig_05_conv_teorema \
  && test -f "tmp/fig_05_conv_teorema.png" \
  || echo "  mm::show não gravou tmp/fig_05_conv_teorema.png"
```

```
[1] Original
[2] Espace: Gaussienne
[3] Frequence: H(u,v).F(u,v)
```

```
try:
    mm.show(
        [
            mm.read("tmp/fig_05_conv_teorema_0.png"),
            mm.read("tmp/fig_05_conv_teorema_1.png"),
            mm.read("tmp/fig_05_conv_teorema_2.png"),
        ],
        titles=[
            'Original',
            'Espaco: Gaussiana',
            'Frequencia: H(u,v).F(u,v)',
        ],
        cols=3,
    )
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + "
    tmp/fig_05_conv_teorema_0.png (ver a versao Python)")
```

Figure 5.9: Teorema da Convolução: filtrar no domínio do espaço (Gaussiana) equivale a multiplicar o espectro por $H(u, v)$ na frequência. As saídas coincidem visualmente.

> **i** À propos de la différence numérique
>
> La différence résiduelle de l'ordre de 10^{-13} ne viole pas le Théorème de Convolution, mais reflète des **limitations computationnelles** inhérentes à l'arithmétique en virgule flottante (double précision, $\sim 10^{-16}$) et à l'**ordre des opérations** entre les deux méthodes :
>
> - **Convolution spatiale :** somme pondérée des voisins avec des arrondis successifs.
> - **Convolution fréquentielle :** implique trois transformées FFT et une multiplication complexe, sujette à des erreurs de troncature et de quantification.
>
> Par conséquent, l'égalité théorique est exacte, mais l'implémentation numérique produit une différence pratiquement nulle (erreur relative $< 10^{-12}$), confirmant le théorème dans la précision de la machine.

5.5 Filtres dans le Domaine Fréquentiel

Un filtre dans le domaine fréquentiel peut être interprété comme une **fonction de transfert appliquée au spectre de l'image**. Dans cette représentation, chaque coefficient de fréquence est multiplié par une valeur comprise entre 0 et 1, qui détermine son atténuation ou sa préservation. La forme de cette fonction définit l'effet visuel du filtre.

Coupure brutale et *ringing*. Les filtres idéaux avec une transition instantanée à une fréquence de coupure D_0 produisent des discontinuités dans le domaine fréquentiel. Cette discontinuité se reflète dans le domaine spatial sous forme d'oscillations près des bords, connues sous le nom de *ringing*. Cet effet est associé à la convolution avec des fonctions à support infini dans l'espace, comme la fonction *sinc*, comme illustré dans la Figure 5.10.

Filtres à transition douce. Des alternatives telles que les filtres gaussien et de Butterworth lissent la transition entre les régions de passage et de rejet, réduisant ainsi le *ringing*. En contrepartie, ce lissage implique une frontière de séparation moins nette entre les fréquences préservées et atténuées.

```
%%writefile tmp/fig_05_conv_teorema_zoom.cpp
#define MM_OUT "tmp/fig_05_conv_teorema_zoom.png"
//| label: fig-05-conv-teorema-zoom
//| fig-cap: "A Dualidade Perigosa: o corte abrupto na Frequência (cilindro Ideal) vira
obrigatoriamente uma *sinc* no espaço. Suas ondulações causam o *ringing* fantasma nas bordas
da imagem."
//| echo: true
//| output: true
```

```
#include <opencv2/opencv.hpp>
#include "morph.hpp"
#include <vector>
#include <string>
#include <filesystem>

int main() {
    // Filtro Ideal na frequência (cilindro) e sua resposta espacial (sinc 2D).
    int N = 128;
    cv::Mat H_freq = mm::idealFilter(N, N, 20);  // 1 dentro do raio 20, 0 fora
    mm::Image h_space = mm::spatialKernel(H_freq); // ifft2(H) -> ondulações da sinc (ringing)

    mm::Image H_freq_img = H_freq; // convertendo para exibição
    mm::show(
        std::vector<mm::Image>{H_freq_img, h_space},
        MM_OUT,
        std::vector<std::string>{
            "Frequencia: filtro Ideal (cilindro)",
            "Espaco: ondulacoes da sinc (causa do ringing)",
        },
        2
    );


// [pdi:panel-io] auto-generated - do not edit by hand
std::filesystem::create_directories("tmp");
mm::write(H_freq, "tmp/fig_05_conv_teorema_zoom_0.png");
mm::write(h_space, "tmp/fig_05_conv_teorema_zoom_1.png");
// [pdi:panel-io:end]
return 0;
}
```

```
Overwriting tmp/fig_05_conv_teorema_zoom.cpp
```

```
!g++ -I. -std=c++17 -DMM_USE_OPENCV -I/usr/include/opencv4 tmp/fig_05_conv_teorema_zoom.cpp -o
tmp/fig_05_conv_teorema_zoom -lopencv_stitching -lopencv_alphamat -lopencv_aruco
-lopencv_barcode -lopencv_bgsegm -lopencv_bioinspired -lopencv_ccalib -lopencv_dnn_objdetect
-lopencv_dnn_superres -lopencv_dpm -lopencv_face -lopencv_freetype -lopencv_fuzzy -lopencv_hdf
-lopencv_hfs -lopencv_img_hash -lopencv_intensity_transform -lopencv_line_descriptor
-lopencv_mcc -lopencv_quality -lopencv_rapid -lopencv_reg -lopencv_rgbd -lopencv_saliency
-lopencv_shape -lopencv_stereo -lopencv_structured_light -lopencv_phase_unwrapping
-lopencv_superres -lopencv_optflow -lopencv_surface_matching -lopencv_tracking
-lopencv_highgui -lopencv_datasets -lopencv_text -lopencv_plot -lopencv_ml -lopencv_videostab
-lopencv_videoio -lopencv_viz -lopencv_wechat_qrcode -lopencv_ximgproc -lopencv_video
-lopencv_xobjdetect -lopencv_objdetect -lopencv_calib3d -lopencv_imgcodecs -lopencv_features2d
-lopencv_dnn -lopencv_flann -lopencv_xphoto -lopencv_photo -lopencv_imgproc -lopencv_core \
  && ./tmp/fig_05_conv_teorema_zoom \
  && test -f "tmp/fig_05_conv_teorema_zoom.png" \
  || echo "  mm::show não gravou tmp/fig_05_conv_teorema_zoom.png"
```

```
[1] Frequencia: filtro Ideal (cilindro)
[2] Espaco: ondulacoes da sinc (causa do ringing)
```

```
try:
    mm.show(
        [
            mm.read("tmp/fig_05_conv_teorema_zoom_0.png"),
            mm.read("tmp/fig_05_conv_teorema_zoom_1.png"),
        ],
```

```
        titles=[
            'Frequencia: filtro Ideal (cilindro)',
            'Espaco: ondulacoes da sinc (causa do ringing)',
        ],
        cols=2,
    )
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + "
    tmp/fig_05_conv_teorema_zoom_0.png (ver a versao Python)")
```

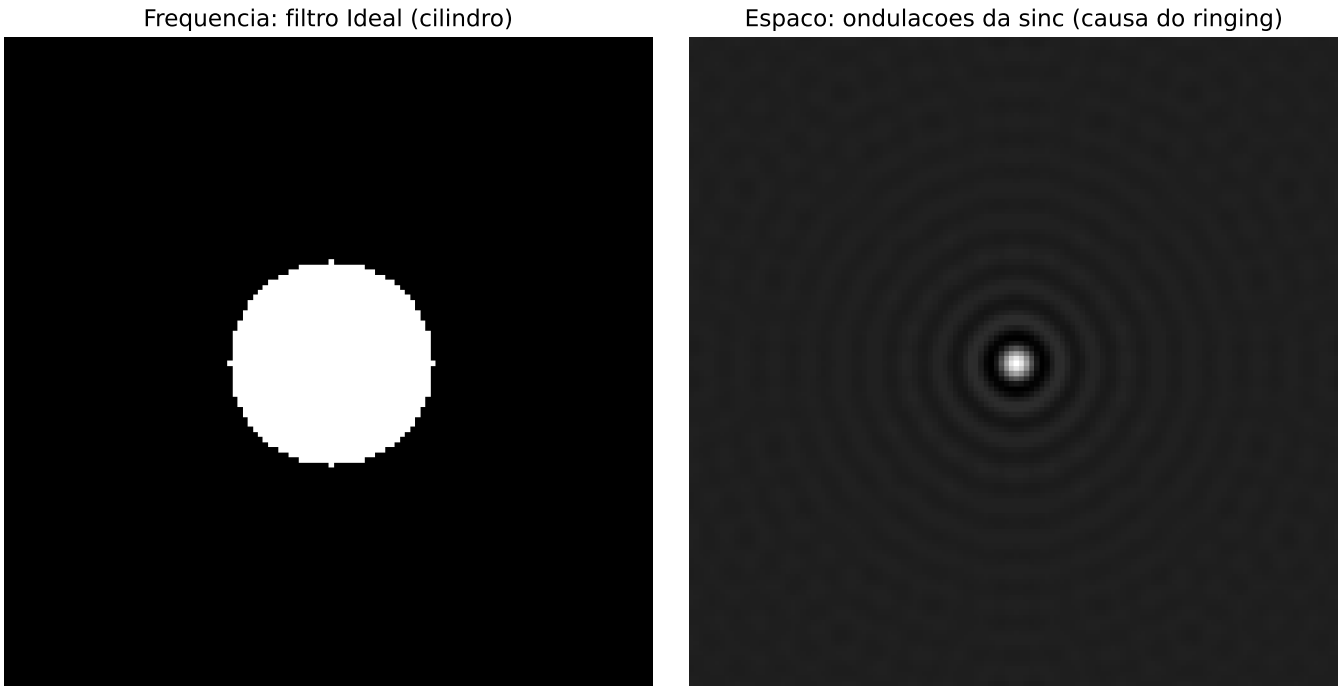


Figure 5.10: A Dualidade Perigosa: o corte abrupto na Frequência (cilindro Ideal) vira obrigatoriamente uma *sinc* no espaço. Suas ondulações causam o *ringing* fantasma nas bordas da imagem.

5.5.1 Filtres Passe-Bas

Les filtres passe-bas atténuent les composantes haute fréquence, ce qui entraîne un lissage de l'image et une réduction du bruit. Après la centralisation du spectre (FFT Shift), la distance de chaque point au centre est donnée par :

$$D(u,v) = \sqrt{\left(u - \tfrac{M}{2}\right)^2 + \left(v - \tfrac{N}{2}\right)^2} \tag{5.5}$$

Filtre idéal (LPFI) :

$$H_{\text{ideal}}(u,v) = \begin{cases} 1, & D(u,v) \leq D_0 \\ 0, & D(u,v) > D_0 \end{cases} \tag{5.6}$$

La coupure brutale à D_0 introduit des discontinuités dans le domaine fréquentiel, entraînant des oscillations dans le domaine spatial, connues sous le nom de *ringing*. Cet effet est associé à la convolution avec des fonctions à support infini.

Filtre gaussien (LPFG) :

$$H_{\text{gauss}}(u,v) = e^{-D^2(u,v)/(2\sigma^2)} \tag{5.7}$$

La régularité de la fonction gaussienne dans le domaine fréquentiel évite les discontinuités, ce qui élimine le *ringing* et produit une transition progressive entre les fréquences conservées et atténuées.

Filtre de Butterworth (LPFB) d'ordre n :

$$H_{\mathrm{BW}}(u,v) = \frac{1}{1 + \left[D(u,v)/D_0\right]^{2n}} \tag{5.8}$$

Le paramètre n contrôle la régularité de la transition entre la transmission et la rejection des fréquences. Les petites valeurs produisent des transitions douces, tandis que les grandes valeurs rapprochent le comportement du filtre idéal, avec un risque accru de *ringing*. Un exemple comparatif est présenté à la Figure 5.11..

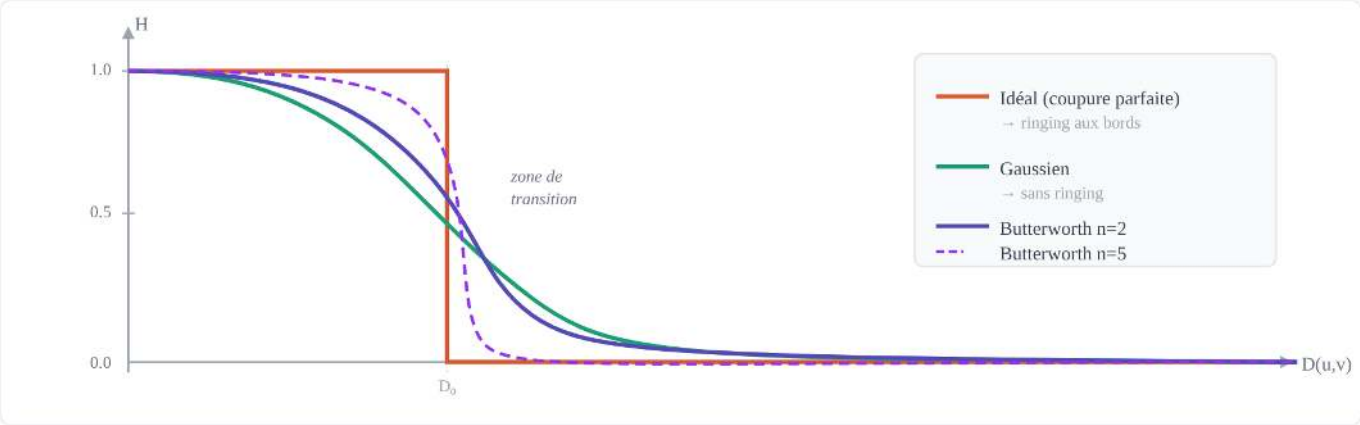


Figure 5.11: Filtros passe-bas.

5.5.2 Filtres Passe-Haut et Passe-Bande

Les filtres passe-haut peuvent être obtenus à partir d'un filtre passe-bas complémentaire, défini comme :

$$H_{\mathrm{HP}}(u,v) = 1 - H_{\mathrm{LP}}(u,v)$$

Ce type de filtre préserve les composantes haute fréquence, en mettant en évidence les contours et les détails, tout en atténuant les régions de variation lente.

Les filtres passe-bande préservent uniquement une bande intermédiaire de fréquences, limitée par deux rayons D_L et D_H :

$$H_{\mathrm{BP}}(u,v) = H_{\mathrm{LP}}^{(D_H)}(u,v) \cdot \left[1 - H_{\mathrm{LP}}^{(D_L)}(u,v)\right]$$

Ce type de filtrage est utile lorsque l'on souhaite supprimer simultanément les composantes basse et haute fréquence, en ne préservant que les structures d'échelle intermédiaire.

Une application importante est la suppression du **bruit périodique**, dans lequel des motifs réguliers apparaissent comme des pics localisés dans le spectre de magnitude. Ces pics peuvent être atténués à l'aide de filtres *notch* (réjecteur de bande), positionnés spécifiquement sur les fréquences indésirables.

Des exemples de filtres dans le domaine fréquentiel sont présentés dans le simulateur de la Figure 5.12, Figure 5.13 et Figure 5.14..

```
%%writefile tmp/fig_05_filtros_freq.cpp
#define MM_OUT "tmp/fig_05_filtros_freq.png"
//| label: fig-05-filtros-freq
//| fig-cap: "Comparação entre filtros passa-baixa: Ideal (D =30), Gaussiano (D =30) e
Butterworth (D =30, n=2). Perfis de H(u,v) ao longo de uma linha central e imagens filtradas
correspondentes."
//| echo: true
//| output: true
```

Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.fr/cap05/sim-05-filtros.html

Figure 5.12: Simulateur interactif de filtres dans le domaine fréquentiel.

```cpp
#include <opencv2/opencv.hpp>
#include <vector>
#include <string>
#include <cmath>
#include "morph.hpp"

// Fonction pour visualiser H (normalisation 0-255)
static cv::Mat H_vis(const cv::Mat& H) {
    cv::Mat h8;
    H.convertTo(h8, CV_8UC1, 255.0);
    cv::normalize(h8, h8, 0, 255, cv::NORM_MINMAX);
    return h8;
}

int main() {
// [pdi:state-io] auto-generated - do not edit by hand
mm::Image img_gray = mm::_read_state("tmp/state/img_gray_20.png");
// [pdi:state-io:end]

    int M = img_gray.h;
    int N = img_gray.w;
    int D0 = 30;
    int n_bw = 2;

    //   Fonctions de transfert (H centré) via les constructeurs de morph.hpp
    //   Ideal : 1 si D<=D0, sinon 0
    //   Gaussien : exp(-D^2/(2 D0^2))
    //   Butterworth : 1 / (1 + (D/D0)^(2n))
    cv::Mat H_ideal = mm::idealFilter(M, N, D0);
    cv::Mat H_gauss = mm::gaussFilter(M, N, D0);
    cv::Mat H_bw    = mm::butterFilter(M, N, D0, n_bw);

    //   Images filtrées via FFT
    mm::Image img_ideal = mm::freqFilter(img_gray, H_ideal);
    mm::Image img_gauss = mm::freqFilter(img_gray, H_gauss);
    mm::Image img_bw    = mm::freqFilter(img_gray, H_bw);

    //   Profils de H(u,v) sur la ligne centrale, dessinés avec cv::line
    int linha = M / 2;
    int Wp = 512, Hp = 288;
    cv::Mat perfil(Hp, Wp, CV_8UC3, cv::Scalar(255, 255, 255));

    auto traca = [&](const cv::Mat& row, cv::Scalar cor) {
        cv::Point ant(-1, -1);
        bool first = true;
        for (int x = 0; x < N; ++x) {
            int px = (int)std::round(x * (Wp - 1.0) / (N - 1.0));
            int py = (int)std::round(10 + (1.0 - row.at<double>(0, x)) * (Hp - 20));
            if (!first) {
                cv::line(perfil, ant, cv::Point(px, py), cor, 2, cv::LINE_AA);
            }
            ant = cv::Point(px, py);
            first = false;
        }
    };

    // H_ideal[linha] est une ligne de la matrice CV_64F → extraire comme cv::Mat 1×N
    cv::Mat H_ideal_row = H_ideal.row(linha);
    cv::Mat H_gauss_row = H_gauss.row(linha);
    cv::Mat H_bw_row    = H_bw.row(linha);
```

```
    traca(H_ideal_row, cv::Scalar(216, 90, 48));
    traca(H_gauss_row, cv::Scalar(29, 158, 117));
    traca(H_bw_row,    cv::Scalar(83, 74, 183));

    mm::Image img_gray_c = img_gray;
    mm::Image img_ideal_c = img_ideal;
    mm::Image img_gauss_c = img_gauss;
    mm::Image img_bw_c = img_bw;
    mm::Image h_ideal_v = mm::Image(H_vis(H_ideal));
    mm::Image h_gauss_v = mm::Image(H_vis(H_gauss));
    mm::Image h_bw_v = mm::Image(H_vis(H_bw));
    mm::Image perfil_img = mm::Image(perfil);

    mm::show(
        std::vector<mm::Image>{img_gray_c, img_ideal_c, img_gauss_c, img_bw_c,
                               h_ideal_v, h_gauss_v, h_bw_v, perfil_img},
        MM_OUT,
        std::vector<std::string>{
            "Original", "LPF Ideal", "LPF Gaussien", "LPF Butterworth (n=2)",
            "H Ideal", "H Gaussien", "H Butterworth", "Profils H(u,v)"
        },
        4
    );

    return 0;
}
```

```
Overwriting tmp/fig_05_filtros_freq.cpp
```

```
!g++ -I. -std=c++17 -DMM_USE_OPENCV -I/usr/include/opencv4 tmp/fig_05_filtros_freq.cpp -o
tmp/fig_05_filtros_freq -lopencv_stitching -lopencv_alphamat -lopencv_aruco -lopencv_barcode
-lopencv_bgsegm -lopencv_bioinspired -lopencv_ccalib -lopencv_dnn_objdetect
-lopencv_dnn_superres -lopencv_dpm -lopencv_face -lopencv_freetype -lopencv_fuzzy -lopencv_hdf
-lopencv_hfs -lopencv_img_hash -lopencv_intensity_transform -lopencv_line_descriptor
-lopencv_mcc -lopencv_quality -lopencv_rapid -lopencv_reg -lopencv_rgbd -lopencv_saliency
-lopencv_shape -lopencv_stereo -lopencv_structured_light -lopencv_phase_unwrapping
-lopencv_superres -lopencv_optflow -lopencv_surface_matching -lopencv_tracking
-lopencv_highgui -lopencv_datasets -lopencv_text -lopencv_plot -lopencv_ml -lopencv_videostab
-lopencv_videoio -lopencv_viz -lopencv_wechat_qrcode -lopencv_ximgproc -lopencv_video
-lopencv_xobjdetect -lopencv_objdetect -lopencv_calib3d -lopencv_imgcodecs -lopencv_features2d
-lopencv_dnn -lopencv_flann -lopencv_xphoto -lopencv_photo -lopencv_imgproc -lopencv_core \
  && ./tmp/fig_05_filtros_freq \
  && test -f "tmp/fig_05_filtros_freq.png" \
  || echo "  mm::show não gravou tmp/fig_05_filtros_freq.png"
```

```
[1] Original
[2] LPF Ideal
[3] LPF Gaussien
[4] LPF Butterworth (n=2)
[5] H Ideal
[6] H Gaussien
[7] H Butterworth
[8] Profils H(u,v)
```

```
try:
    mm.show(mm.read("tmp/fig_05_filtros_freq.png"), figsize=(16, 9))
except Exception as _e:
```

```
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + "
    tmp/fig_05_filtros_freq.png (ver a versao Python)")
```

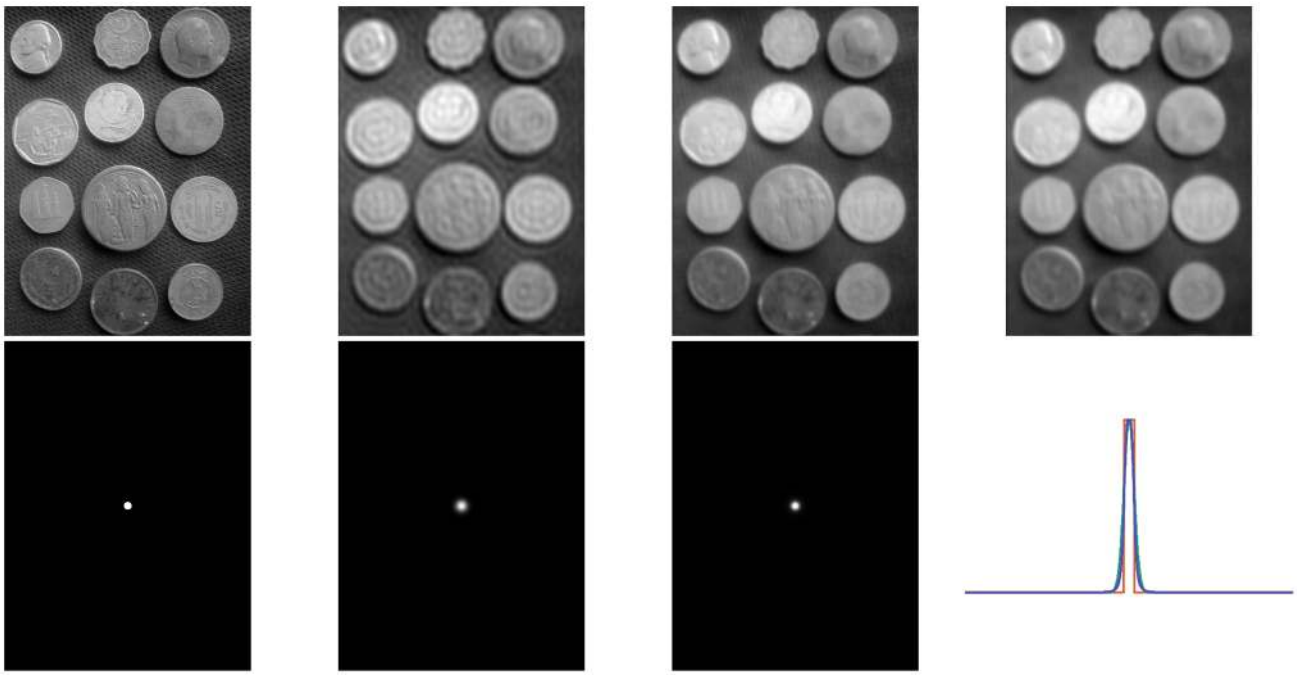


Figure 5.13: Comparação entre filtros passa-baixa: Ideal (D =30), Gaussiano (D =30) e Butterworth (D =30, n=2). Perfis de H(u,v) ao longo de uma linha central e imagens filtradas correspondentes.

```
%%writefile tmp/fig_05_filtros_passa_alta.cpp
#define MM_OUT "tmp/fig_05_filtros_passa_alta.png"
#include <opencv2/opencv.hpp>
#include <vector>
#include <string>
#include "morph.hpp"
#include <filesystem>

int main() {
// [pdi:state-io] auto-generated - do not edit by hand
mm::Image img_gray = mm::_read_state("tmp/state/img_gray_20.png");
// [pdi:state-io:end]

    //| label: fig-05-filtros-passa-alta
    //| fig-cap: "Filtro passa-alta Gaussiano. (a) Original; (b) Filtro passa-alta (D =30) -
    as bordas das moedas e fundo texturizado são realçados."

    // Filtro passa-alta = complemento do passa-baixa Gaussiano (1 - H_gauss),
    // construído com mm.gaussFilter(..., highpass=true) e aplicado via FFT.
    int M = img_gray.h;
    int N = img_gray.w;
    double D0 = 30;
    cv::Mat H_alta = mm::gaussFilter(M, N, D0, true);
    mm::Image img_alta = mm::freqFilter(img_gray, H_alta);

    mm::show(
        std::vector<mm::Image>{img_gray, img_alta},
        MM_OUT,
        {"Original", "Passa-alta Gaussiano (D0=30)"},
        2
```

```
    );


// [pdi:panel-io] auto-generated - do not edit by hand
std::filesystem::create_directories("tmp");
mm::write(img_gray, "tmp/fig_05_filtros_passa_alta_0.png");
mm::write(img_alta, "tmp/fig_05_filtros_passa_alta_1.png");
// [pdi:panel-io:end]
return 0;
}
```

```
Overwriting tmp/fig_05_filtros_passa_alta.cpp
```

```
!g++ -I. -std=c++17 -DMM_USE_OPENCV -I/usr/include/opencv4 tmp/fig_05_filtros_passa_alta.cpp
-o tmp/fig_05_filtros_passa_alta -lopencv_stitching -lopencv_alphamat -lopencv_aruco
-lopencv_barcode -lopencv_bgsegm -lopencv_bioinspired -lopencv_ccalib -lopencv_dnn_objdetect
-lopencv_dnn_superres -lopencv_dpm -lopencv_face -lopencv_freetype -lopencv_fuzzy -lopencv_hdf
-lopencv_hfs -lopencv_img_hash -lopencv_intensity_transform -lopencv_line_descriptor
-lopencv_mcc -lopencv_quality -lopencv_rapid -lopencv_reg -lopencv_rgbd -lopencv_saliency
-lopencv_shape -lopencv_stereo -lopencv_structured_light -lopencv_phase_unwrapping
-lopencv_superres -lopencv_optflow -lopencv_surface_matching -lopencv_tracking
-lopencv_highgui -lopencv_datasets -lopencv_text -lopencv_plot -lopencv_ml -lopencv_videostab
-lopencv_videoio -lopencv_viz -lopencv_wechat_qrcode -lopencv_ximgproc -lopencv_video
-lopencv_xobjdetect -lopencv_objdetect -lopencv_calib3d -lopencv_imgcodecs -lopencv_features2d
-lopencv_dnn -lopencv_flann -lopencv_xphoto -lopencv_photo -lopencv_imgproc -lopencv_core \
  && ./tmp/fig_05_filtros_passa_alta \
  && test -f "tmp/fig_05_filtros_passa_alta.png" \
  || echo "  mm::show não gravou tmp/fig_05_filtros_passa_alta.png"
```

```
[1] Original
[2] Passa-alta Gaussiano (D0=30)
```

```
try:
    mm.show(
        [
            mm.read("tmp/fig_05_filtros_passa_alta_0.png"),
            mm.read("tmp/fig_05_filtros_passa_alta_1.png"),
        ],
        titles=[
            'Original',
            'Passa-alta Gaussiano ($D_0=30$)',
        ],
        cols=2,
    )
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + "
    tmp/fig_05_filtros_passa_alta_0.png (ver a versao Python)")
```

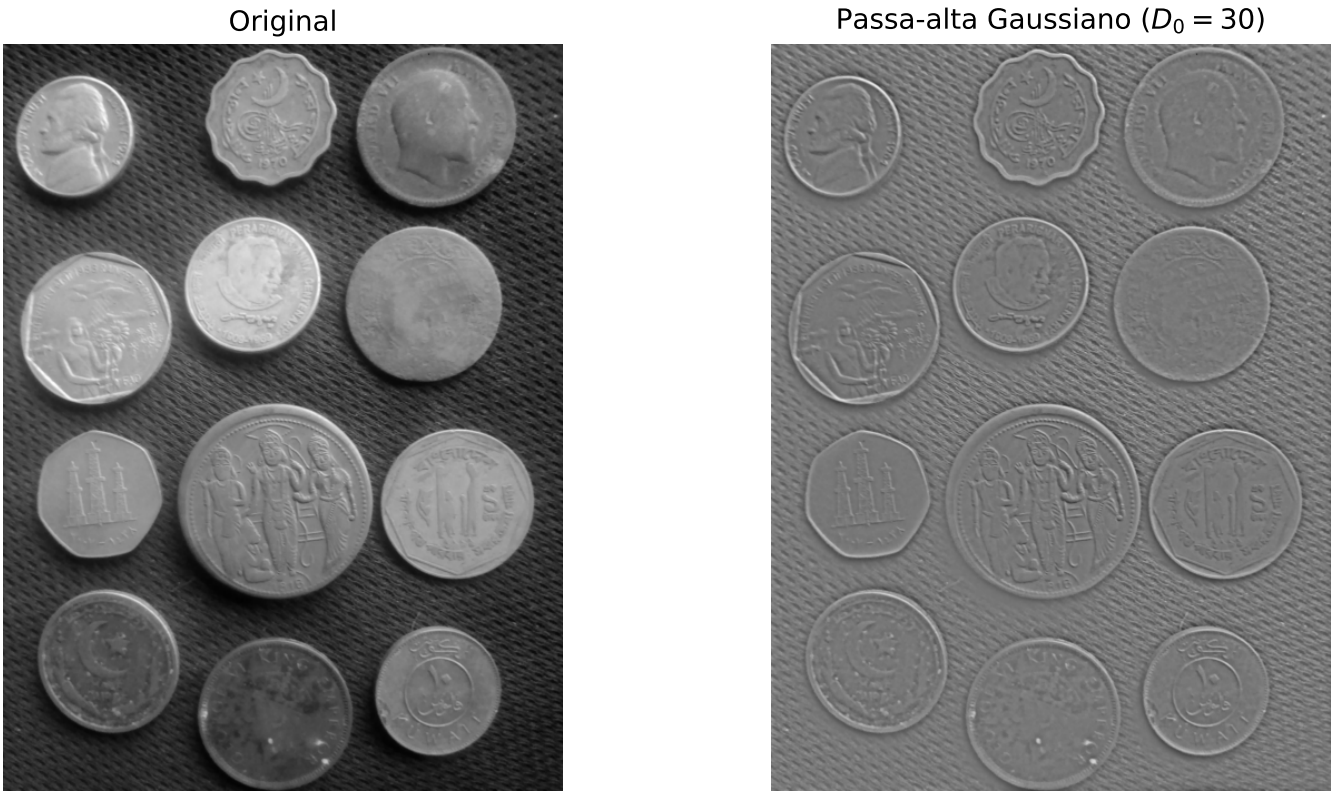


Figure 5.14: Filtro passa-alta Gaussiano. (a) Original; (b) Filtro passa-alta (D =30) - as bordas das moedas e fundo texturizado são realçados.

5.5.3 Suppression du bruit périodique

Le bruit périodique — associé à des interférences électriques, à des motifs réguliers de capteurs ou à des artefacts de numérisation — apparaît dans le spectre de Fourier sous forme de **pics ponctuels symétriques autour du centre**.

Le filtre **rejette-bande (*notch*)** atténue sélectivement ces fréquences, tout en préservant les autres composantes de l'image. Un exemple d'application est présenté dans la Figure 5.15..

```
%%writefile tmp/fig_05_ruido_periodico.cpp
#define MM_OUT "tmp/fig_05_ruido_periodico.png"
//| label: fig-05-ruido-periodico
//| fig-cap: "Remoção de ruído periódico via filtro *notch* no domínio da frequência: (a)
imagem com ruído senoidal, (b) espectro mostrando os picos do ruído, (c) máscara *notch*
centrada nos picos, (d) imagem restaurada."
//| echo: true
//| output: true

// Ruído senoidal 2D -> espectro -> máscara notch nos 4 picos simétricos -> restauração.
#include <opencv2/opencv.hpp>
#include <cmath>
#include <iostream>
#include <vector>
#include <string>
#include "morph.hpp"

int main() {
// [pdi:state-io] auto-generated - do not edit by hand
mm::Image img_gray = mm::_read_state("tmp/state/img_gray_20.png");
// [pdi:state-io:end]
```

```
    int h_img = img_gray.h;
    int w_img = img_gray.w;

    // Criar meshgrid X, Y (equivalente a np.meshgrid)
    std::vector<std::vector<double>> X(h_img, std::vector<double>(w_img));
    std::vector<std::vector<double>> Y(h_img, std::vector<double>(w_img));
    for (int i = 0; i < h_img; ++i) {
        for (int j = 0; j < w_img; ++j) {
            X[i][j] = j;
            Y[i][j] = i;
        }
    }

    int u0 = 20, v0 = 20;  // frequências exatas do ruído

    // Converter img_gray para cv::Mat e depois para float64
    cv::Mat img_gray_mat(h_img, w_img, CV_8UC1, img_gray.data.data());
    cv::Mat img_gray_float;
    img_gray_mat.convertTo(img_gray_float, CV_64F);

    // Calcular ruído: ruido = 40 * sin(2 * pi * (u0*X/w_img + v0*Y/h_img))
    cv::Mat ruido(h_img, w_img, CV_64F);
    for (int i = 0; i < h_img; ++i) {
        for (int j = 0; j < w_img; ++j) {
            double arg = 2.0 * M_PI * (u0 * j / static_cast<double>(w_img) + v0 * i /
static_cast<double>(h_img));
            ruido.at<double>(i, j) = 40.0 * std::sin(arg);
        }
    }

    // img_ruidosa = clip(img_gray + ruido, 0, 255).astype(uint8)
    cv::Mat img_ruidosa_float = img_gray_float + ruido;
    cv::Mat img_ruidosa_uint;
    img_ruidosa_float.convertTo(img_ruidosa_uint, CV_8UC1, 1.0, 0.0);
    cv::Mat img_ruidosa;
    cv::normalize(img_ruidosa_uint, img_ruidosa, 0, 255, cv::NORM_MINMAX, CV_8UC1);

    // Espectro de magnitude (log) da imagem ruidosa
    mm::Image img_ruidosa_mm(img_ruidosa);
    mm::Image mag_vis = mm::spectrumMag(img_ruidosa_mm);

    // Máscara notch: 1.0 em todo o plano, disco de 0.0 em cada um dos 4 picos
    cv::Mat mascara = cv::Mat::ones(h_img, w_img, CV_64F);
    int r_notch = 8;
    int cy = h_img / 2, cx = w_img / 2;

    // Coordenadas relativas para os 4 picos simétricos
    std::vector<std::pair<int, int>> picos = {{v0, u0}, {-v0, -u0}, {v0, -u0}, {-v0, u0}};

    for (const auto& pico : picos) {
        int dy = pico.first;
        int dx = pico.second;
        int py = cy + dy;
        int px = cx + dx;

        for (int i = 0; i < h_img; ++i) {
            for (int j = 0; j < w_img; ++j) {
                double dist = std::sqrt(std::pow(i - py, 2) + std::pow(j - px, 2));
                if (dist <= r_notch) {
                    mascara.at<double>(i, j) = 0.0;
```

```
                }
            }
        }
    }

    // Versão visual da máscara (0-255)
    cv::Mat mascara_vis_float;
    cv::normalize(mascara, mascara_vis_float, 0, 255, cv::NORM_MINMAX, CV_8UC1);
    mm::Image mascara_vis(mascara_vis_float);

    // Filtragem: aplica a máscara centrada como filtro no domínio da frequência
    mm::Image img_rest_vis = mm::freqFilter(img_ruidosa_mm, mascara);

    // Calcular PSNR
    cv::Mat img_rest_cv = img_rest_vis;
    double psnr = cv::PSNR(img_gray_mat, img_rest_cv);
    std::cout << "PSNR (original vs restaurada): " << psnr << " dB" << std::endl;

    // Mostrar resultados
    std::vector<mm::Image> imagens = {img_ruidosa_mm, mag_vis, mascara_vis, img_rest_vis};
    std::vector<std::string> titulos = {
        "Com ruído periódico",
        "Espectro (log)",
        "Máscara notch",
        "Restaurada (PSNR=" + std::to_string(psnr).substr(0, 4) + " dB)"
    };
    mm::show(imagens, MM_OUT, titulos, 4);

    return 0;
}
```

```
Overwriting tmp/fig_05_ruido_periodico.cpp
```

```
!g++ -I. -std=c++17 -DMM_USE_OPENCV -I/usr/include/opencv4 tmp/fig_05_ruido_periodico.cpp -o
tmp/fig_05_ruido_periodico -lopencv_stitching -lopencv_alphamat -lopencv_aruco
-lopencv_barcode -lopencv_bgsegm -lopencv_bioinspired -lopencv_ccalib -lopencv_dnn_objdetect
-lopencv_dnn_superres -lopencv_dpm -lopencv_face -lopencv_freetype -lopencv_fuzzy -lopencv_hdf
-lopencv_hfs -lopencv_img_hash -lopencv_intensity_transform -lopencv_line_descriptor
-lopencv_mcc -lopencv_quality -lopencv_rapid -lopencv_reg -lopencv_rgbd -lopencv_saliency
-lopencv_shape -lopencv_stereo -lopencv_structured_light -lopencv_phase_unwrapping
-lopencv_superres -lopencv_optflow -lopencv_surface_matching -lopencv_tracking
-lopencv_highgui -lopencv_datasets -lopencv_text -lopencv_plot -lopencv_ml -lopencv_videostab
-lopencv_videoio -lopencv_viz -lopencv_wechat_qrcode -lopencv_ximgproc -lopencv_video
-lopencv_xobjdetect -lopencv_objdetect -lopencv_calib3d -lopencv_imgcodecs -lopencv_features2d
-lopencv_dnn -lopencv_flann -lopencv_xphoto -lopencv_photo -lopencv_imgproc -lopencv_core \
  && ./tmp/fig_05_ruido_periodico \
  && test -f "tmp/fig_05_ruido_periodico.png" \
  || echo "  mm::show não gravou tmp/fig_05_ruido_periodico.png"
```

```
PSNR (original vs restaurada): 33.0718 dB
[1] Com ruído periódico
[2] Espectro (log)
[3] Máscara notch
[4] Restaurada (PSNR=33.0 dB)
```

```
try:
    mm.show(mm.read("tmp/fig_05_ruido_periodico.png"), figsize=(16, 4))
except Exception as _e:
```

```
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + "
    tmp/fig_05_ruido_periodico.png (ver a versao Python)")
```

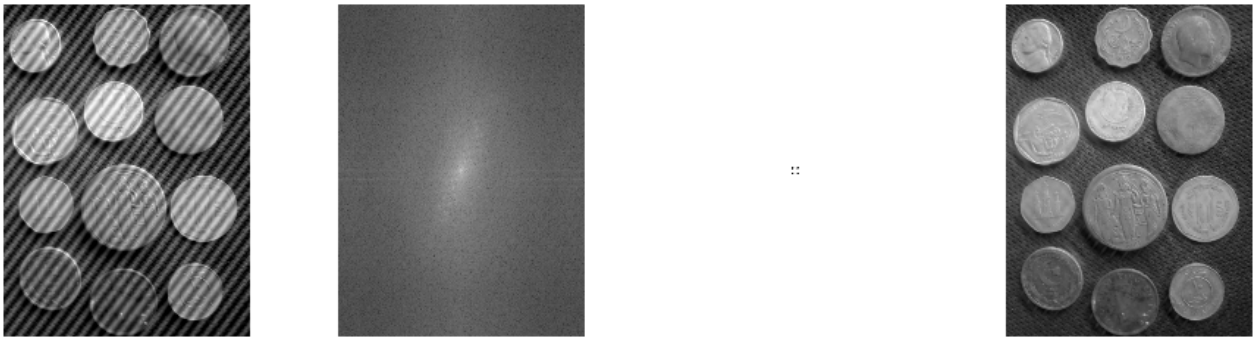


Figure 5.15: Remoção de ruído periódico via filtro *notch* no domínio da frequência: (a) imagem com ruído senoidal, (b) espectro mostrando os picos do ruído, (c) máscara *notch* centrada nos picos, (d) imagem restaurada.

📌 Synthèse — Filtres spectraux

Filtre	Effet visuel	Artefact	Utilisation
Passe-bas idéal	Lissage intense	*Ring*	Illustratif
Passe-bas gaussien	Lissage doux	N'présente pas de *ring*	Lissage général
Passe-bas de Butterworth	Lissage contrôlé	*Ring* (ordres élevés)	Compromis entre lissage et sélectivité
Passe-haut	Renforcement des contours	Amplification du bruit	Détection des contours
Notch	Suppression sélective de fréquences	Distorsions locales possibles	Suppression du bruit périodique

La conception de filtres dans le domaine fréquentiel consiste à définir des masques spectraux. Cependant, des effets dans le domaine spatial, tels que le *ring* et le flou, émergent directement de ces choix dans le spectre.

5.6 *Ondelettes* et Multirésolution

La Transformée de Fourier décompose le signal en fréquences **globales** : chaque coefficient $F(u, v)$ reçoit des contributions de toute l'image, sans information explicite sur la localisation spatiale de ces fréquences. Ainsi, les structures localisées, comme les bords, sont représentées de manière distribuée dans le spectre.

Les ***ondelettes*** **(ondelettes)** surmontent cette limitation en utilisant des fonctions de base **localisées dans l'espace**, qui peuvent être déplacées et mises à l'échelle. Ces fonctions possèdent un **support compact**, c'est-à-dire qu'elles sont différentes de zéro uniquement dans une région finie du domaine, permettant une représentation simultanée en termes de **fréquence et de localisation spatiale**.

5.6.1 La Limite de la Transformée de Fourier : localisation spatiale

La Transformée de Fourier décrit avec précision **quelles fréquences sont présentes** dans un signal, mais ne représente pas explicitement **où ces fréquences se produisent dans l'espace**.

Dans l'expérience présentée dans la Figure 5.16, deux images avec des structures localisées à des positions différentes produisent des spectres de magnitude pratiquement identiques. Cela est dû au fait que la représentation de Fourier est globale : chaque coefficient reçoit une contribution de l'ensemble de l'image.

Par conséquent, le spectre de magnitude ne représente pas explicitement la localisation des contours ou d'autres structures, mais uniquement la distribution des fréquences présentes. Cette limitation a motivé le développement de représentations multirésolution, telles que la Transformée *Wavelet* Discrète (DWT), capables de décrire simultanément la fréquence et la localisation spatiale des structures de l'image.

```
%%writefile tmp/fig_05_fracasso_fourier.cpp
#define MM_OUT "tmp/fig_05_fracasso_fourier.png"
//| label: fig-05-fracasso-fourier
//| fig-cap: "Fourier global é cego para a posição. Os espectros não dizem onde as bordas
estão."
//| echo: true
//| output: true

#include <opencv2/opencv.hpp>
#include <vector>
#include <string>
#include "morph.hpp"
#include <filesystem>

cv::Mat fftshift(const cv::Mat& input) {
    // Desloca o espectro para o centro
    int cx = input.cols / 2;
    int cy = input.rows / 2;
    cv::Mat output = input.clone();
    // Trocar os quadrantes
    cv::Mat q0(input, cv::Rect(0, 0, cx, cy));       // superior esquerdo
    cv::Mat q1(input, cv::Rect(cx, 0, input.cols - cx, cy)); // superior direito
    cv::Mat q2(input, cv::Rect(0, cy, cx, input.rows - cy)); // inferior esquerdo
    cv::Mat q3(input, cv::Rect(cx, cy, input.cols - cx, input.rows - cy)); // inferior direito

    cv::Mat temp;
    q0.copyTo(temp); q3.copyTo(q0); temp.copyTo(q3); // trocar q0 <-> q3
    q1.copyTo(temp); q2.copyTo(q1); temp.copyTo(q2); // trocar q1 <-> q2
    return output;
}

int main() {
    // Criar os sinais
    cv::Mat img_sinal1 = cv::Mat::zeros(128, 128, CV_64F);
    img_sinal1.colRange(20, 25) = 1.0;
    img_sinal1.rowRange(100, 105) = 1.0;

    cv::Mat img_sinal2 = cv::Mat::zeros(128, 128, CV_64F);
    img_sinal2.colRange(90, 95) = 1.0;
    img_sinal2.rowRange(30, 35) = 1.0;

    // Calcular os espectros com FFT e log1p
    cv::Mat fft_mat1, mag1, fft_mat2, mag2;
    cv::dft(img_sinal1, fft_mat1, cv::DFT_COMPLEX_OUTPUT);
    cv::dft(img_sinal2, fft_mat2, cv::DFT_COMPLEX_OUTPUT);

    // Dividir canais complexos
    std::vector<cv::Mat> planes1, planes2;
    cv::split(fft_mat1, planes1);
    cv::split(fft_mat2, planes2);

    // Magnitude
    cv::Mat mag1_shift, mag2_shift;
    cv::magnitude(planes1[0], planes1[1], mag1);
    cv::magnitude(planes2[0], planes2[1], mag2);
```

```
    // Aplicar fftshift e log1p (log(1+|F|))
    mag1 = fftshift(mag1);
    mag2 = fftshift(mag2);
    cv::log(1.0 + mag1, mag1_shift);
    cv::log(1.0 + mag2, mag2_shift);

    // Normalizar para visualização
    cv::normalize(mag1_shift, mag1_shift, 0, 255, cv::NORM_MINMAX, CV_8U);
    cv::normalize(mag2_shift, mag2_shift, 0, 255, cv::NORM_MINMAX, CV_8U);

    // Converter os sinais para 8-bit para exibição
    cv::Mat img1_8u, img2_8u;
    cv::normalize(img_sinal1, img1_8u, 0, 255, cv::NORM_MINMAX, CV_8U);
    cv::normalize(img_sinal2, img2_8u, 0, 255, cv::NORM_MINMAX, CV_8U);

    // Mostrar os resultados
    std::vector<mm::Image> images = {mm::Image(img1_8u), mm::Image(mag1_shift),
                                     mm::Image(img2_8u), mm::Image(mag2_shift)};
    std::vector<std::string> titles = {"Sinal A", "Espectro A", "Sinal B (Deslocado)",
"Espectro B"};
    mm::show(images, MM_OUT, titles, 4);


// [pdi:panel-io] auto-generated - do not edit by hand
std::filesystem::create_directories("tmp");
mm::write(img_sinal1, "tmp/fig_05_fracasso_fourier_0.png");
mm::write(mag1, "tmp/fig_05_fracasso_fourier_1.png");
mm::write(img_sinal2, "tmp/fig_05_fracasso_fourier_2.png");
mm::write(mag2, "tmp/fig_05_fracasso_fourier_3.png");
// [pdi:panel-io:end]
return 0;
}
```

```
Overwriting tmp/fig_05_fracasso_fourier.cpp
```

```
!g++ -I. -std=c++17 -DMM_USE_OPENCV -I/usr/include/opencv4 tmp/fig_05_fracasso_fourier.cpp -o
tmp/fig_05_fracasso_fourier -lopencv_stitching -lopencv_alphamat -lopencv_aruco
-lopencv_barcode -lopencv_bgsegm -lopencv_bioinspired -lopencv_ccalib -lopencv_dnn_objdetect
-lopencv_dnn_superres -lopencv_dpm -lopencv_face -lopencv_freetype -lopencv_fuzzy -lopencv_hdf
-lopencv_hfs -lopencv_img_hash -lopencv_intensity_transform -lopencv_line_descriptor
-lopencv_mcc -lopencv_quality -lopencv_rapid -lopencv_reg -lopencv_rgbd -lopencv_saliency
-lopencv_shape -lopencv_stereo -lopencv_structured_light -lopencv_phase_unwrapping
-lopencv_superres -lopencv_optflow -lopencv_surface_matching -lopencv_tracking
-lopencv_highgui -lopencv_datasets -lopencv_text -lopencv_plot -lopencv_ml -lopencv_videostab
-lopencv_videoio -lopencv_viz -lopencv_wechat_qrcode -lopencv_ximgproc -lopencv_video
-lopencv_xobjdetect -lopencv_objdetect -lopencv_calib3d -lopencv_imgcodecs -lopencv_features2d
-lopencv_dnn -lopencv_flann -lopencv_xphoto -lopencv_photo -lopencv_imgproc -lopencv_core \
  && ./tmp/fig_05_fracasso_fourier \
  && test -f "tmp/fig_05_fracasso_fourier.png" \
  || echo "  mm::show não gravou tmp/fig_05_fracasso_fourier.png"
```

```
[1] Sinal A
[2] Espectro A
[3] Sinal B (Deslocado)
[4] Espectro B
```

```
try:
    mm.show(
        [
```

```
            mm.read("tmp/fig_05_fracasso_fourier_0.png"),
            mm.read("tmp/fig_05_fracasso_fourier_1.png"),
            mm.read("tmp/fig_05_fracasso_fourier_2.png"),
            mm.read("tmp/fig_05_fracasso_fourier_3.png"),
        ],
        titles=[
            'Sinal A',
            'Espectro A',
            'Sinal B (Deslocado)',
            'Espectro B',
        ],
        cols=4,
        figsize=(14, 4),
    )
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + "
    tmp/fig_05_fracasso_fourier_0.png (ver a versao Python)")
```

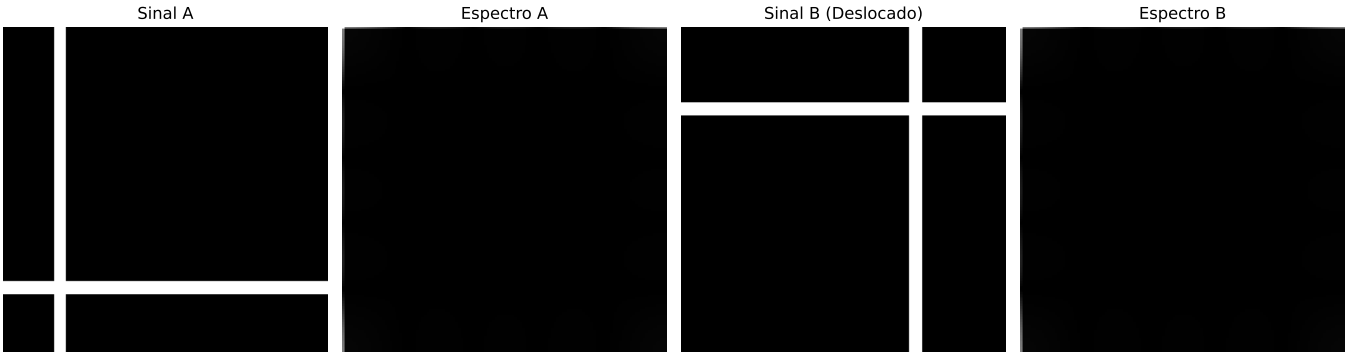


Figure 5.16: Fourier global é cego para a posição. Os espectros não dizem onde as bordas estão.

5.6.2 Transformée *Wavelet* Discrète 2D

La Transformée *Wavelet* Discrète (DWT) applique, séparément dans les directions horizontale et verticale, deux filtres complémentaires : un **passe-bas** h (approximation) et un **passe-haut** g (détails), suivis d'un sous-échantillonnage par un facteur de 2 dans chaque dimension. Ce processus produit quatre sous-bandes, dont les noms indiquent la combinaison des filtres appliqués dans chaque direction (L = *Low-pass*, passe-bas ; H = *High-pass*, passe-haut). Les caractéristiques de chaque sous-bande sont résumées dans la Table 5.3.

$$\mathrm{DWT}(f) = \{\ \underbrace{\mathrm{LL}}_{\text{approx.}}\ ,\quad \underbrace{\mathrm{LH}}_{\text{détails horizontaux}}\ ,\quad \underbrace{\mathrm{HL}}_{\text{détails verticaux}}\ ,\quad \underbrace{\mathrm{HH}}_{\text{détails diagonaux}}\ \}.$$

Tableau 5.3: Sous-bandes produites par la Transformée *Wavelet* Discrète 2D (DWT), indiquant les filtres appliqués dans chaque direction et le contenu prédominant de chaque composante.

Sous-bande	Filtres appliqués	Contenu visuel
LL	bas × bas	Approximation de l'image (version lissée et réduite)
LH	bas × haut	Bords horizontaux et variations verticales
HL	haut × bas	Bords verticaux et variations horizontales
HH	haut × haut	Détails diagonaux et textures

La décomposition peut être appliquée récursivement sur la sous-bande LL, générant une représentation multi-résolution. Après J niveaux, on obtient une structure avec $3J + 1$ sous-bandes, où chaque nouveau niveau réduit la résolution de la composante d'approximation.

> **i** Lien avec les CNN
>
> La décomposition multi-résolution des *wavelets* possède une relation conceptuelle avec les représentations hiérarchiques utilisées dans les réseaux de neurones convolutifs (CNN). Dans les deux cas, des étapes successives de filtrage et de réduction de résolution produisent des descriptions de plus en plus abstraites de l'image. Cependant, les ***wavelets* utilisent des filtres mathématiquement définis et reconstructibles**, tandis que les **CNN apprennent leurs filtres pendant l'entraînement**.

5.6.3 Familles d'*ondelettes*

Différentes familles d'*ondelettes* présentent des compromis distincts entre **support spatial**, régularité et capacité de compression. Le **support** correspond à l'étendue de la fonction *ondelette* dans le domaine spatial : plus le support est petit, plus la fonction est localisée ; plus il est grand, plus sa représentation tend à être régulière, mais avec un coût de calcul plus élevé. La Table 5.4 compare certaines des familles les plus utilisées.

Tableau 5.4: Comparaison entre familles d'*ondelettes*, mettant en évidence la longueur du support, le nombre de moments nuls, la symétrie et les applications typiques.

Ondelette	Longueur du support	Moments nuls	Symétrie	Usage typique
Haar	2	1	Asymétrique	Introduction et analyse de base
Daubechies db4	8	4	Asymétrique	Compression et analyse générale
Symlet sym4	8	4	Quasi symétrique	Reconstruction de signaux
Biorthogonale 5/3	5/3	2/2	Symétrique	JPEG 2000 sans perte
Biorthogonale 9/7	9/7	4/4	Symétrique	JPEG 2000 avec perte

Les **moments nuls** mesurent la capacité de l'*ondelette* à représenter les régions lisses de l'image avec peu de coefficients non nuls. Une *ondelette* à p moments nuls annule exactement les polynômes de degré jusqu'à $p - 1$. Par conséquent, plus le nombre de moments nuls est élevé, plus l'efficacité de compression tend à être grande dans les régions homogènes, bien que cela implique généralement des fonctions à support plus long.

La Figure 5.17 présente les fonctions de base (*ondelettes*) $\psi(t)$ dans le domaine spatial. Ces fonctions possèdent un **support compact**, c'est-à-dire qu'elles sont non nulles uniquement sur une région finie du domaine, contrairement aux sinusoïdes de la Transformée de Fourier, qui s'étendent sur tout le domaine.

```
%%writefile tmp/fig_05_wavelet_functions.cpp
#define MM_OUT "tmp/fig_05_wavelet_functions.png"
#include <opencv2/opencv.hpp>
#include "morph.hpp"
#include <vector>
#include <string>
#include <filesystem>

//| label: fig-05-wavelet-functions
//| fig-cap: "Funções da *Wavelet* ( ). Note como elas rapidamente decaem para zero (suporte compacto), ao contrário das senoides infinitas de Fourier."
//| echo: true
```

```
//| output: true

int main() {
    // psi via mm.wavefun (algoritmo em cascata) - suporte compacto: decai a zero.
    std::vector<double> x_h, phi_h, psi_h;
    mm::wavefun("haar", 6, x_h, phi_h, psi_h);
    std::vector<double> x_d, phi_d, psi_d;
    mm::wavefun("db4", 6, x_d, phi_d, psi_d);

    std::vector<std::vector<double>> xs_h = {x_h};
    std::vector<std::vector<double>> ys_h = {psi_h};
    std::vector<std::vector<double>> xs_d = {x_d};
    std::vector<std::vector<double>> ys_d = {psi_d};

    mm::Image chart_h = mm::lineChart(xs_h, ys_h, {"psi Haar"},
                                      {cv::Scalar(180, 60, 40)},
                                      "Ondaleta Haar (psi)", "t");
    mm::Image chart_d = mm::lineChart(xs_d, ys_d, {"psi Daubechies 4"},
                                      {cv::Scalar(60, 140, 40)},
                                      "Ondaleta Daubechies 4 (psi)", "t");

    mm::show(std::vector<mm::Image>{chart_h, chart_d}, MM_OUT,
             std::vector<std::string>{"Haar", "Daubechies 4"}, 2);


// [pdi:panel-io] auto-generated - do not edit by hand
std::filesystem::create_directories("tmp");
mm::write(chart_h, "tmp/fig_05_wavelet_functions_0.png");
mm::write(chart_d, "tmp/fig_05_wavelet_functions_1.png");
// [pdi:panel-io:end]
return 0;
}
```

```
Overwriting tmp/fig_05_wavelet_functions.cpp
```

```
!g++ -I. -std=c++17 -DMM_USE_OPENCV -I/usr/include/opencv4 tmp/fig_05_wavelet_functions.cpp -o
tmp/fig_05_wavelet_functions -lopencv_stitching -lopencv_alphamat -lopencv_aruco
-lopencv_barcode -lopencv_bgsegm -lopencv_bioinspired -lopencv_ccalib -lopencv_dnn_objdetect
-lopencv_dnn_superres -lopencv_dpm -lopencv_face -lopencv_freetype -lopencv_fuzzy -lopencv_hdf
-lopencv_hfs -lopencv_img_hash -lopencv_intensity_transform -lopencv_line_descriptor
-lopencv_mcc -lopencv_quality -lopencv_rapid -lopencv_reg -lopencv_rgbd -lopencv_saliency
-lopencv_shape -lopencv_stereo -lopencv_structured_light -lopencv_phase_unwrapping
-lopencv_superres -lopencv_optflow -lopencv_surface_matching -lopencv_tracking
-lopencv_highgui -lopencv_datasets -lopencv_text -lopencv_plot -lopencv_ml -lopencv_videostab
-lopencv_videoio -lopencv_viz -lopencv_wechat_qrcode -lopencv_ximgproc -lopencv_video
-lopencv_xobjdetect -lopencv_objdetect -lopencv_calib3d -lopencv_imgcodecs -lopencv_features2d
-lopencv_dnn -lopencv_flann -lopencv_xphoto -lopencv_photo -lopencv_imgproc -lopencv_core \
  && ./tmp/fig_05_wavelet_functions \
  && test -f "tmp/fig_05_wavelet_functions.png" \
  || echo "  mm::show não gravou tmp/fig_05_wavelet_functions.png"
```

```
[1] Haar
[2] Daubechies 4
```

```
try:
    mm.show(
        [
            mm.read("tmp/fig_05_wavelet_functions_0.png"),
            mm.read("tmp/fig_05_wavelet_functions_1.png"),
```

```
        ],
        titles=[
            'Haar',
            'Daubechies 4',
        ],
        cols=2,
    )
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + "
    tmp/fig_05_wavelet_functions_0.png (ver a versao Python)")
```

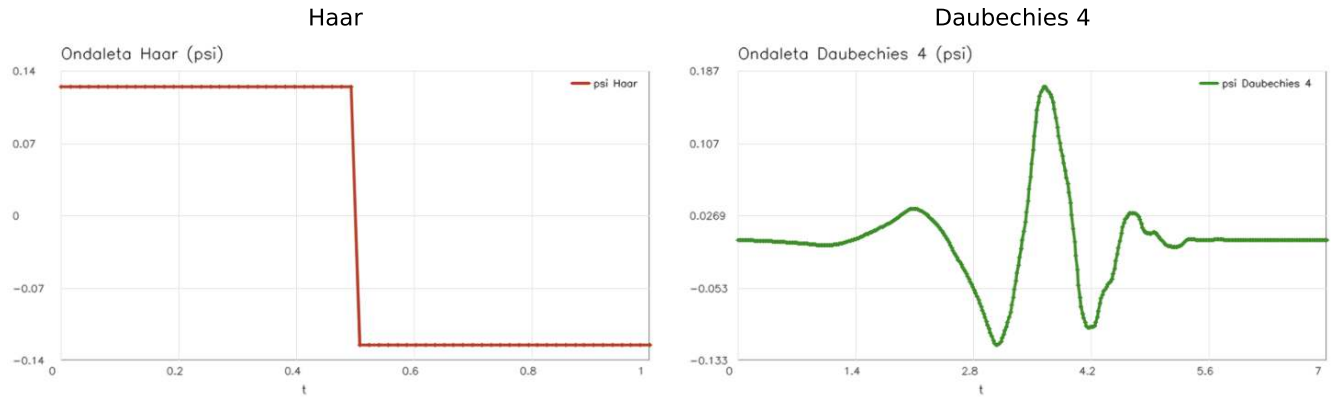


Figure 5.17: Funções da *Wavelet* ( ). Note como elas rapidamente decaem para zero (suporte compacto), ao contrário das senoides infinitas de Fourier.

Le diagramme de la Figure 5.18 illustre l'analyse multirésolution effectuée par la DWT, dans laquelle la sous-bande d'approximation (LL) est successivement décomposée, formant une représentation hiérarchique à deux niveaux.

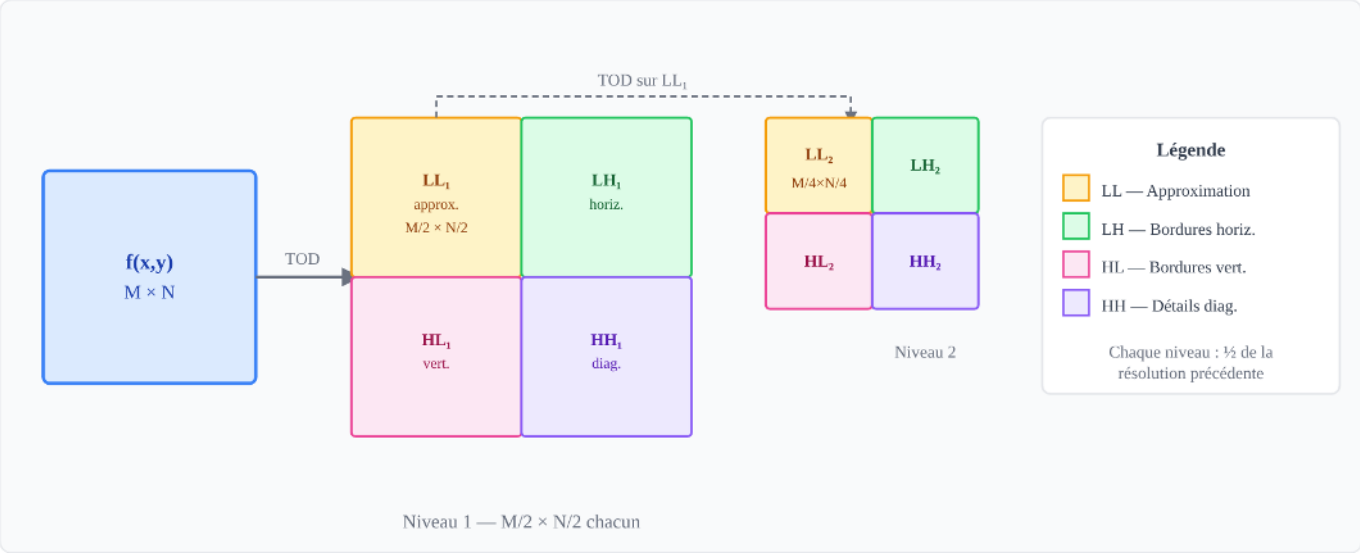


Figure 5.18: Schéma de la décomposition *wavelet* 2D en deux niveaux.

Le simulateur de la Figure 5.19 permet d'explorer de manière interactive la Transformée *Wavelet* Discrète 2D (TWD) à l'aide de la *wavelet* de Haar. La décomposition en sous-bandes met en évidence la séparation entre la composante d'approximation et les composantes de détail de l'image.

Les différents motifs d'entrée permettent d'observer le comportement directionnel des filtres. Dans les images comportant des bords horizontaux et verticaux, les sous-bandes LH et HL mettent en évidence, respectivement,

les variations verticales et horizontales de l'intensité. Dans les régions à variation douce, la majeure partie de l'énergie se concentre dans la sous-bande d'approximation LL, tandis que les sous-bandes de détail présentent des coefficients proches de zéro.

L'analyse multirésolution peut également être observée en augmentant le nombre de niveaux de décomposition. Dans ce cas, seule la sous-bande LL_1 est à nouveau décomposée, donnant naissance aux sous-bandes LL_2, LH_2, HL_2 et HH_2, qui forment le deuxième niveau de la représentation hiérarchique.

Dans les motifs constitués de régions homogènes de grande étendue, comme un dégradé doux ou un damier composé de grands blocs, l'énergie demeure principalement concentrée dans la sous-bande LL. Dans le dégradé, cela s'explique par le fait que les différences entre pixels voisins sont faibles. Dans le damier, en revanche, les pixels possèdent pratiquement la même intensité à l'intérieur de chaque bloc, de sorte que seules les frontières entre les blocs produisent des coefficients non nuls dans les sous-bandes de détail. Comme ces frontières n'occupent qu'une petite fraction de l'image, leur contribution à l'énergie totale reste réduite.

Pour permettre l'analyse visuelle de ces variations subtiles, le simulateur intègre un contrôle de gain de contraste des détails (variant de 1 à 8). Ce paramètre agit comme un facteur d'amplification linéaire appliqué exclusivement aux coefficients des sous-bandes de détail (LH, HL et HH) avant leur rendu à l'écran. Dans les scénarios de transition douce (comme le dégradé) ou d'uniformité locale (comme l'intérieur des blocs du damier), les différences numériques calculées par le filtre passe-haut de Haar donnent des coefficients très proches de zéro, ce qui rendrait les quadrants correspondants sombres et imperceptibles à l'œil nu. En multipliant ces valeurs par le gain, le simulateur fait ressortir visuellement les structures de haute fréquence cachées et met en évidence l'orientation des bords restants.

Le graphique de l'énergie par sous-bande quantifie cette répartition entre la composante d'approximation et les composantes de détail, démontrant que le gain visuel ne modifie pas la métrique originale de l'énergie. Dans les images naturelles, la majeure partie de l'énergie se concentre dans la sous-bande LL, tandis que les sous-bandes LH, HL et HH représentent principalement les bords, les textures et d'autres variations locales de l'intensité.

5.6.4 Analyse multirésolution avec la DWT 2D

La transformée *wavelet* discrète 2D (DWT) décompose une image en composantes d'approximation et de détail, organisées de manière hiérarchique à différentes échelles et orientations. Comme les sous-bandes de détail dans les images naturelles présentent souvent des coefficients de faible contraste, les exemples pratiques suivants utilisent un motif géométrique synthétique généré en Python. Cette approche reproduit le comportement du simulateur Figure 5.19, rendant visuellement explicites les effets du filtrage spatial et de la décomposition multirésolution.

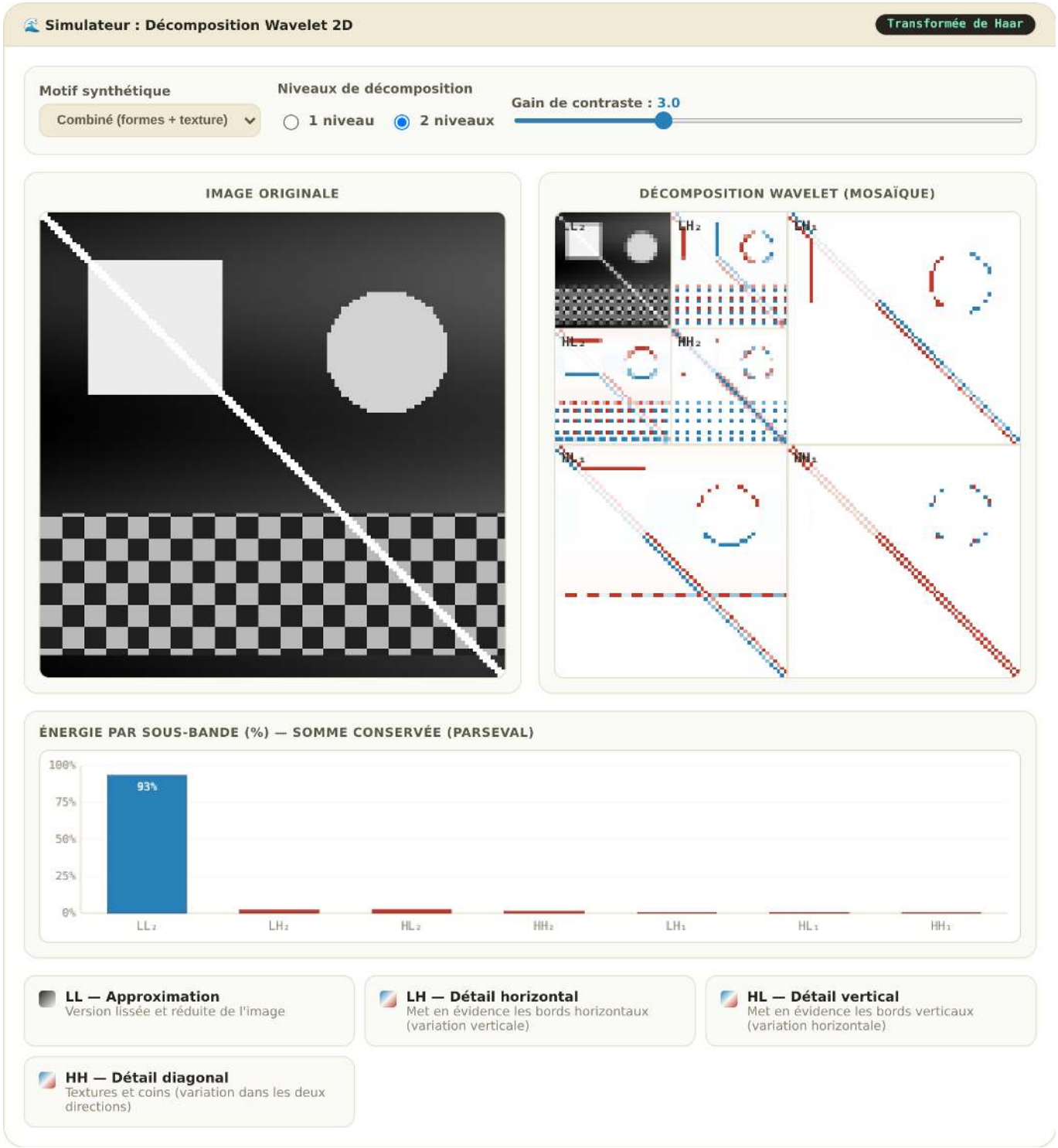
5.6.4.1 Décomposition en Mosaïque à Multiples Niveaux

La Figure 5.20 illustre la structure hiérarchique de la DWT à deux niveaux en utilisant la *wavelet* de Haar. Le processus repose sur l'application combinée de filtres passe-bas et passe-haut dans les directions horizontale et verticale, suivis d'un sous-échantillonnage par un facteur de 2.

Au premier niveau, l'image originale donne naissance à la sous-bande d'approximation (LL_1) ainsi qu'aux composantes de détail horizontale (LH_1), verticale (HL_1) et diagonale (HH_1). Dans l'analyse multirésolution, la sous-bande LL_1 est à nouveau filtrée et sous-échantillonnée, générant le deuxième niveau de décomposition (LL_2, LH_2, HL_2 et HH_2).

Pour faciliter l'interprétation visuelle des composantes de détail, le code extrait la valeur absolue de leurs coefficients et applique une normalisation linéaire (*min-max*) afin d'occuper toute la plage dynamique des niveaux de gris [0, 255]. Cette opération transforme les régions homogènes (coefficients nuls) en noir et met en évidence en blanc les contours et textures extraits à chaque échelle et orientation.

```
%%writefile tmp/fig_05_dwt_subbandas.cpp
#define MM_OUT "tmp/fig_05_dwt_subbandas.png"
```



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.fr/cap05/sim-05-wavelet.html

Figure 5.19: Simulation de la décomposition *wavelet* 2D.

```
//| label: fig-05-dwt-subbandas
//| fig-cap: "Decomposição *wavelet* 2D de 2 níveis com *wavelet* Haar: subbandas LL, LH, HL, HH em cada nível. As subbandas de detalhe revelam estruturas orientadas em diferentes escalas utilizando um padrão sintético."
//| echo: true
//| output: true

#include <opencv2/opencv.hpp>
#include "morph.hpp"
#include <iostream>
#include <vector>
#include <string>
#include <cmath>

// Geração da Imagem Sintética (Mesmo padrão 'combined' do simulador)
cv::Mat gerar_imagem_sintetica(int N = 256) {
    cv::Mat img = cv::Mat::zeros(N, N, CV_64F);
    for (int y = 0; y < N; y++) {
        for (int x = 0; x < N; x++) {
            double v = 55 + 35 * ((double)x / N) + 15 * std::sin(y / 24.0);
            // Quadrado
            if (24 < x && x < 100 && 24 < y && y < 100) {
                v = 225;
            }
            // Círculo
            int cx = 190, cy = 76, r = 34;
            if ((x - cx) * (x - cx) + (y - cy) * (y - cy) < r * r) {
                v = 205;
            }
            // Textura periódica (inferior)
            if (y > 164 && y < 244) {
                int p = 12;
                v = (((x / p + y / p) % 2) == 0) ? 185 : 65;
            }
            // Linha diagonal
            if (std::abs(x - y) < 4) {
                v = 240;
            }
            img.at<double>(y, x) = std::max(0.0, std::min(v, 255.0));
        }
    }
    cv::Mat out;
    img.convertTo(out, CV_8U);
    return out;
}

// Função para visualizar subbanda
cv::Mat sb_vis(const cv::Mat& sb) {
    cv::Mat abs_sb;
    cv::absdiff(sb, cv::Scalar(0), abs_sb);
    cv::Mat normalized;
    cv::normalize(abs_sb, normalized, 0, 255, cv::NORM_MINMAX, CV_8U);
    return normalized;
}

int main() {
    // Substitui a imagem escura de moedas pelo padrão sintético claro
    cv::Mat img_gray = gerar_imagem_sintetica(256);

    // Decomposição wavelet 2 níveis
    std::string wavelet = "haar";
```

```
    cv::Mat img_float;
    img_gray.convertTo(img_float, CV_64F);

    // Nível 1
    mm::Subbands coefs1 = mm::dwt2(img_float, wavelet);
    cv::Mat LL1 = coefs1.LL;
    cv::Mat LH1 = coefs1.LH;
    cv::Mat HL1 = coefs1.HL;
    cv::Mat HH1 = coefs1.HH;

    // Nível 2 (aplicado sobre LL1)
    mm::Subbands coefs2 = mm::dwt2(LL1, wavelet);
    cv::Mat LL2 = coefs2.LL;
    cv::Mat LH2 = coefs2.LH;
    cv::Mat HL2 = coefs2.HL;
    cv::Mat HH2 = coefs2.HH;

    std::cout << "Forma original     : " << img_gray.rows << "x" << img_gray.cols <<
std::endl;
    std::cout << "LL1 (nível 1)      : " << LL1.rows << "x" << LL1.cols << "  |  LH1/HL1/HH1:
" << LH1.rows << "x" << LH1.cols << std::endl;
    std::cout << "LL2 (nível 2)      : " << LL2.rows << "x" << LL2.cols << "    |
LH2/HL2/HH2: " << LH2.rows << "x" << LH2.cols << std::endl;

    std::vector<mm::Image> imgs_dwt = {mm::Image(img_gray), mm::Image(sb_vis(LL1)),
mm::Image(sb_vis(LH1)), mm::Image(sb_vis(HL1)), mm::Image(sb_vis(HH1)),
                                       mm::Image(sb_vis(LL2)), mm::Image(sb_vis(LH2)),
mm::Image(sb_vis(HL2)), mm::Image(sb_vis(HH2))};
    std::vector<std::string> titles_dwt = {"Original",
                                           "LL (aprox.)", "LH (horiz.)", "HL (vert.)", "HH
                                           (diag.)",
                                           "LL (aprox.)", "LH (horiz.)", "HL (vert.)", "HH
                                           (diag.)"};

    mm::show(imgs_dwt, MM_OUT, titles_dwt, 5);

    return 0;
}
```

```
Overwriting tmp/fig_05_dwt_subbandas.cpp
```

```
!g++ -I. -std=c++17 -DMM_USE_OPENCV -I/usr/include/opencv4 tmp/fig_05_dwt_subbandas.cpp -o
tmp/fig_05_dwt_subbandas -lopencv_stitching -lopencv_alphamat -lopencv_aruco -lopencv_barcode
-lopencv_bgsegm -lopencv_bioinspired -lopencv_ccalib -lopencv_dnn_objdetect
-lopencv_dnn_superres -lopencv_dpm -lopencv_face -lopencv_freetype -lopencv_fuzzy -lopencv_hdf
-lopencv_hfs -lopencv_img_hash -lopencv_intensity_transform -lopencv_line_descriptor
-lopencv_mcc -lopencv_quality -lopencv_rapid -lopencv_reg -lopencv_rgbd -lopencv_saliency
-lopencv_shape -lopencv_stereo -lopencv_structured_light -lopencv_phase_unwrapping
-lopencv_superres -lopencv_optflow -lopencv_surface_matching -lopencv_tracking
-lopencv_highgui -lopencv_datasets -lopencv_text -lopencv_plot -lopencv_ml -lopencv_videostab
-lopencv_videoio -lopencv_viz -lopencv_wechat_qrcode -lopencv_ximgproc -lopencv_video
-lopencv_xobjdetect -lopencv_objdetect -lopencv_calib3d -lopencv_imgcodecs -lopencv_features2d
-lopencv_dnn -lopencv_flann -lopencv_xphoto -lopencv_photo -lopencv_imgproc -lopencv_core \
  && ./tmp/fig_05_dwt_subbandas \
  && test -f "tmp/fig_05_dwt_subbandas.png" \
  || echo "  mm::show não gravou tmp/fig_05_dwt_subbandas.png"
```

```
Forma original     : 256x256
LL1 (nível 1)      : 128x128  |  LH1/HL1/HH1: 128x128
LL2 (nível 2)      : 64x64    |  LH2/HL2/HH2: 64x64
```

```
[1] Original
[2] LL  (aprox.)
[3] LH  (horiz.)
[4] HL  (vert.)
[5] HH  (diag.)
[6] LL  (aprox.)
[7] LH  (horiz.)
[8] HL  (vert.)
[9] HH  (diag.)
```

```
try:
    mm.show(mm.read("tmp/fig_05_dwt_subbandas.png"), figsize=(16, 7))
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + "
    tmp/fig_05_dwt_subbandas.png (ver a versao Python)")
```

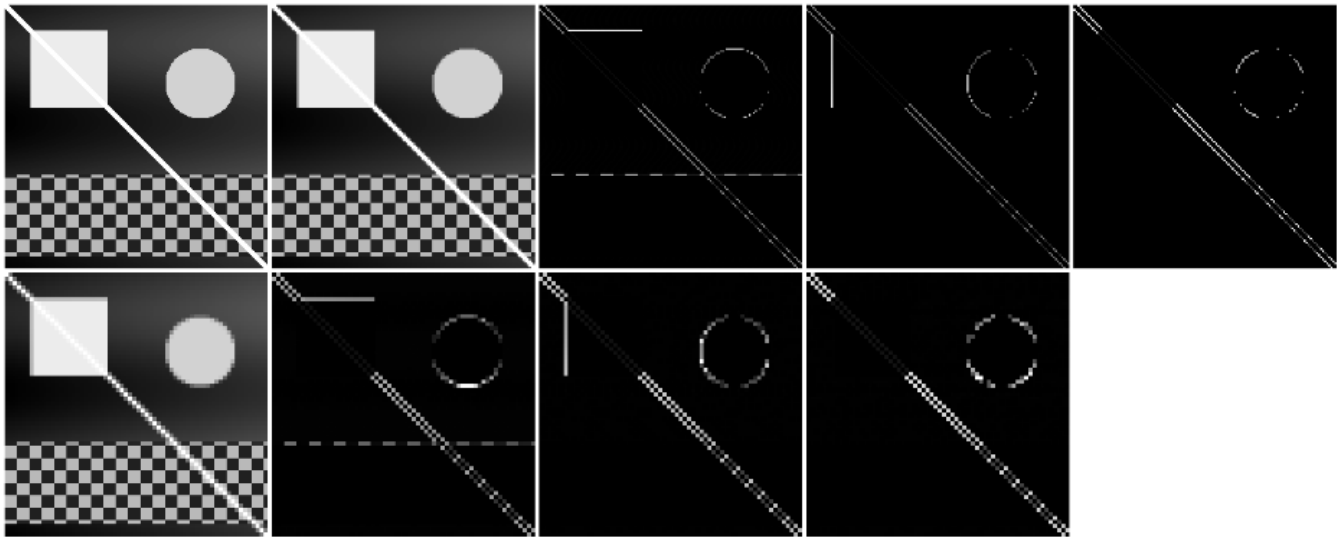


Figure 5.20: Decomposição *wavelet* 2D de 2 níveis com *wavelet* Haar: subbandas LL, LH, HL, HH em cada nível. As subbandas de detalhe revelam estruturas orientadas em diferentes escalas utilizando um padrão sintético.

5.6.4.2 Le Compromis entre Localisation et Régularité

Le choix de la fonction de base (*ondelette*) influence directement la manière dont les caractéristiques de l'image sont distribuées et codées par les coefficients de la DWT. La Figure 5.21 compare les résultats pratiques obtenus en appliquant quatre familles distinctes sur le motif géométrique synthétique : `haar`, `db4`, `sym4` et `bior2.2`.

En raison de son support court et de sa forme en fonction échelon, l'ondelette de Haar produit des coefficients hautement localisés au niveau des discontinuités spatiales, générant des bords fins et nets dans les sous-bandes de détail. En revanche, des familles comme Daubechies (`db4`) et Symlets (`sym4`), qui présentent un support plus étendu (filtres plus longs) et un plus grand nombre de moments nuls, génèrent des réponses plus lisses et plus distribuées autour des transitions, ce qui peut introduire de légères oscillations ou un adoucissement aux frontières abruptes.

Ce comportement met en évidence le compromis classique (*trade-off*) de l'analyse multirésolution : des supports plus petits favorisent la localisation spatiale exacte des bords, tandis que des supports plus grands et un nombre plus élevé de moments nuls tendent à produire des représentations plus parcimonieuses et plus régulières. Cette régularité et cette capacité d'atténuation des hautes fréquences garantissent une plus grande efficacité dans la compaction de l'énergie, des caractéristiques fondamentales pour les applications de compression de données et de débruitage (*denoising*).

```
%%writefile tmp/fig_05_dwt_wavelets.cpp
#define MM_OUT "tmp/fig_05_dwt_wavelets.png"
//| label: fig-05-dwt-wavelets
//| fig-cap: "Comparação entre famílias de *wavelets*: Haar, db4, sym4 e bior2.2. Subbanda LL (aproximação) e HH (diagonal) para cada escolha, ilustrando o compromisso entre compactação e suavidade com base no padrão sintético."
//| echo: true
//| output: true

#include <opencv2/opencv.hpp>
#include <vector>
#include <string>
#include <cmath>
#include "morph.hpp"

// Função auxiliar para visualizar subbandas (normaliza para 8 bits)
mm::Image sb_vis(const cv::Mat& subband) {
    cv::Mat normalized;
    cv::normalize(subband, normalized, 0, 255, cv::NORM_MINMAX, CV_8U);
    return mm::Image(normalized);
}

int main() {
    // Garante que img_gray e img_float utilizem o mesmo padrão sintético claro
    // (fallback: gera imagem sintética se não disponível de outra célula)
    cv::Mat img_gray;
    bool synthetic_available = false;

    // Nota: como não temos a função gerar_imagem_sintetica definida em outra célula,
    // geramos diretamente aqui.
    int N = 256;
    img_gray = cv::Mat(N, N, CV_8UC1);
    for (int y = 0; y < N; y++) {
        for (int x = 0; x < N; x++) {
            double v = 55 + 35 * (double(x) / N) + 15 * std::sin(y / 24.0);
            if (x > 24 && x < 100 && y > 24 && y < 100) v = 225;
            double cx = 190, cy = 76, r = 34;
            if (std::pow(x - cx, 2) + std::pow(y - cy, 2) < r*r) v = 205;
            if (y > 164 && y < 244) {
                int p = 12;
                v = (((x / p + y / p) % 2) == 0) ? 185 : 65;
            }
            if (std::abs(x - y) < 4) v = 240;
            v = std::max(0.0, std::min(v, 255.0));
            img_gray.at<uchar>(y, x) = static_cast<uchar>(std::round(v));
        }
    }

    cv::Mat img_float;
    img_gray.convertTo(img_float, CV_64F);

    std::vector<std::string> wavelets_comp = {"haar", "db4", "sym4", "bior2.2"};
    std::vector<mm::Image> imgs_comp;
    std::vector<std::string> titles_comp;

    for (const auto& wname : wavelets_comp) {
        mm::Subbands s = mm::dwt2(img_float, wname);
        imgs_comp.push_back(sb_vis(s.LL));
        imgs_comp.push_back(sb_vis(s.HH));
        titles_comp.push_back(wname + " - LL ");
        titles_comp.push_back(wname + " - HH ");
```

```
    }

    mm::show(imgs_comp, MM_OUT, titles_comp, 4);
    return 0;
}
```

```
Overwriting tmp/fig_05_dwt_wavelets.cpp
```

```
!g++ -I. -std=c++17 -DMM_USE_OPENCV -I/usr/include/opencv4 tmp/fig_05_dwt_wavelets.cpp -o
tmp/fig_05_dwt_wavelets -lopencv_stitching -lopencv_alphamat -lopencv_aruco -lopencv_barcode
-lopencv_bgsegm -lopencv_bioinspired -lopencv_ccalib -lopencv_dnn_objdetect
-lopencv_dnn_superres -lopencv_dpm -lopencv_face -lopencv_freetype -lopencv_fuzzy -lopencv_hdf
-lopencv_hfs -lopencv_img_hash -lopencv_intensity_transform -lopencv_line_descriptor
-lopencv_mcc -lopencv_quality -lopencv_rapid -lopencv_reg -lopencv_rgbd -lopencv_saliency
-lopencv_shape -lopencv_stereo -lopencv_structured_light -lopencv_phase_unwrapping
-lopencv_superres -lopencv_optflow -lopencv_surface_matching -lopencv_tracking
-lopencv_highgui -lopencv_datasets -lopencv_text -lopencv_plot -lopencv_ml -lopencv_videostab
-lopencv_videoio -lopencv_viz -lopencv_wechat_qrcode -lopencv_ximgproc -lopencv_video
-lopencv_xobjdetect -lopencv_objdetect -lopencv_calib3d -lopencv_imgcodecs -lopencv_features2d
-lopencv_dnn -lopencv_flann -lopencv_xphoto -lopencv_photo -lopencv_imgproc -lopencv_core \
  && ./tmp/fig_05_dwt_wavelets \
  && test -f "tmp/fig_05_dwt_wavelets.png" \
  || echo "  mm::show não gravou tmp/fig_05_dwt_wavelets.png"
```

```
[1] haar - LL
[2] haar - HH
[3] db4 - LL
[4] db4 - HH
[5] sym4 - LL
[6] sym4 - HH
[7] bior2.2 - LL
[8] bior2.2 - HH
```

```
try:
    mm.show(mm.read("tmp/fig_05_dwt_wavelets.png"), figsize=(14, 8))
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + "
    tmp/fig_05_dwt_wavelets.png (ver a versao Python)")
```

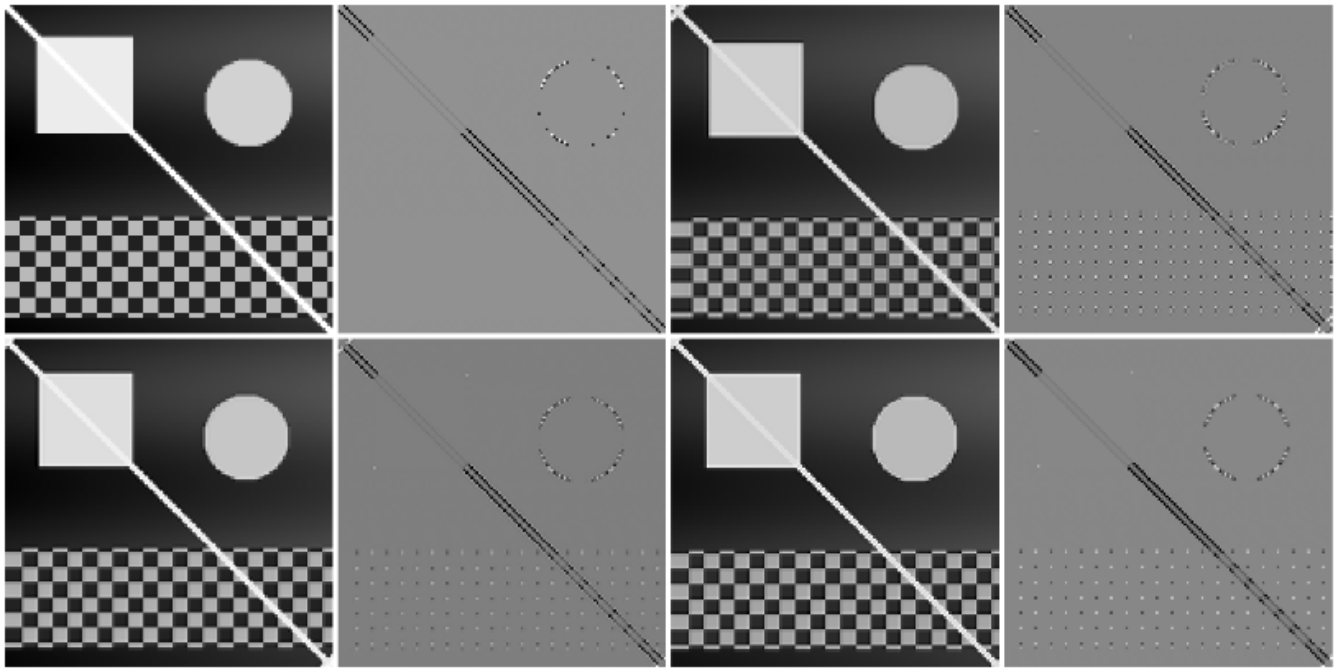


Figure 5.21: Comparação entre famílias de *wavelets*: Haar, db4, sym4 e bior2.2. Subbanda LL (aproximação) e HH (diagonal) para cada escolha, ilustrando o compromisso entre compactação e suavidade com base no padrão sintético.

5.6.4.3 Limiarização de Coeficientes e Compressão

Uma das principais aplicações da Transformada *Wavelet* Discreta (DWT) é a compressão de dados, impulsionada pela capacidade de representação **esparsa** dos coeficientes. A Figure 5.22 ilustra o efeito da limiarização abrupta (*hard thresholding*), técnica na qual coeficientes de detalhe com magnitude inferior a um limiar T são integralmente anulados antes do processo de síntese realizado pela Transformada *Wavelet* Discreta Inversa (IDWT).

À medida que o limiar T é elevado, um volume crescente de coeficientes de alta frequência é zerado. Por concentrarem menor energia, a remoção dessas componentes reduz consideravelmente a quantidade de informação necessária para representar a imagem, mantendo a componente de aproximação global (a subbanda LL mais profunda) intacta para preservar a estrutura macro. Visualmente, esse descarte de coeficientes manifesta-se através do desaparecimento progressivo de texturas finas e da suavização de transições abruptas de intensidade.

A fidelidade da imagem reconstruída frente à original é quantificada pela métrica de **Pico da Relação Sinal-Ruído (PSNR, *Peak Signal-to-Noise Ratio*)**, expressa em decibéis (dB). Valores mais altos de PSNR indicam menor distorção e maior proximidade matemática com o sinal original. O experimento prático evidencia o decaimento gradual do PSNR conforme a agressividade da limiarização aumenta, permitindo avaliar numericamente o limiar ótimo para o balanço entre compressão e degradação visual.

5.6.4.4 Seuil des Coefficients et Compression

L'une des principales applications de la Transformée *Wavelet* Discrète (DWT) est la compression de données, portée par la capacité de représentation **parcimonieuse** des coefficients. La Figure 5.22 illustre l'effet du seuillage abrupt (*hard thresholding*), technique dans laquelle les coefficients de détail dont la magnitude est inférieure à un seuil T sont intégralement annulés avant le processus de synthèse réalisé par la Transformée *Wavelet* Discrète Inverse (IDWT).

À mesure que le seuil T est augmenté, un volume croissant de coefficients haute fréquence est mis à zéro. Comme ils concentrent moins d'énergie, la suppression de ces composantes réduit considérablement la quantité

d'information nécessaire pour représenter l'image, tout en maintenant intacte la composante d'approximation globale (la sous-bande LL la plus profonde) afin de préserver la structure macroscopique. Visuellement, ce rejet de coefficients se manifeste par la disparition progressive des textures fines et par l'adoucissement des transitions abruptes d'intensité.

La fidélité de l'image reconstruite par rapport à l'originale est quantifiée par la métrique du **Pic du Rapport Signal sur Bruit (PSNR, *Peak Signal-to-Noise Ratio*)**, exprimée en décibels (dB). Des valeurs plus élevées de PSNR indiquent une distorsion moindre et une plus grande proximité mathématique avec le signal original. L'expérience pratique met en évidence la décroissance progressive du PSNR à mesure que l'agressivité du seuillage augmente, permettant d'évaluer numériquement le seuil optimal pour l'équilibre entre compression et dégradation visuelle.

```
%%writefile tmp/fig_05_dwt_reconstrucao.cpp
#define MM_OUT "tmp/fig_05_dwt_reconstrucao.png"
//| label: fig-05-dwt-reconstrucao
//| fig-cap: "Reconstrução *wavelet* com limiarização de coeficientes (*hard thresholding*): à
medida que o limiar aumenta, mais detalhes são zerados, produzindo imagens progressivamente
mais suaves. Métrica PSNR quantifica a perda de qualidade sobre o padrão sintético."
//| echo: true
//| output: true

#include <opencv2/opencv.hpp>
#include <string>
#include <vector>
#include <cmath>
#include <algorithm>
#include "morph.hpp"

// Garante que img_gray utilize o mesmo padrão sintético claro
static cv::Mat gerar_imagem_sintetica(int N = 256) {
    cv::Mat img(N, N, CV_8UC1);
    for (int y = 0; y < N; ++y) {
        for (int x = 0; x < N; ++x) {
            double v = 55.0 + 35.0 * (static_cast<double>(x) / N) + 15.0 *
std::sin(static_cast<double>(y) / 24.0);
            if (24 < x && x < 100 && 24 < y && y < 100) { v = 225.0; }
            int cx = 190, cy = 76, r = 34;
            if (std::pow(x - cx, 2) + std::pow(y - cy, 2) < r * r) { v = 205.0; }
            if (y > 164 && y < 244) {
                int p = 12;
                v = (((x / p + y / p) % 2) == 0) ? 185.0 : 65.0;
            }
            if (std::abs(x - y) < 4) { v = 240.0; }
            img.at<unsigned char>(y, x) = static_cast<unsigned char>(std::max(0.0,
std::min(255.0, v)));
        }
    }
    return img;
}

static cv::Mat dwt_threshold_reconstruct(const cv::Mat& img, const std::string& wavelet =
"db4", int nivel = 2, double threshold = 0.0) {
    // Decompõe, aplica limiar e reconstrói via IDWT
    cv::Mat img64f;
    img.convertTo(img64f, CV_64F);
    mm::WaveDec2 c = mm::wavedec2(img64f, wavelet, nivel);

    // Copia e aplica hard thresholding em todos os detalhes
    std::vector<std::vector<cv::Mat>> coefs_t;
    for (size_t j = 0; j < c.detail.size(); ++j) {
```

```
        std::vector<cv::Mat> sb(3);
        cv::Mat LH_t = mm::wave_threshold(c.detail[j][0], threshold, "hard");
        cv::Mat HL_t = mm::wave_threshold(c.detail[j][1], threshold, "hard");
        cv::Mat HH_t = mm::wave_threshold(c.detail[j][2], threshold, "hard");
        sb[0] = LH_t;
        sb[1] = HL_t;
        sb[2] = HH_t;
        coefs_t.push_back(sb);
    }

    // Reconstrução
    mm::WaveDec2 c_t;
    c_t.LL = c.LL;
    c_t.detail = coefs_t;
    cv::Mat rec = mm::waverec2(c_t, wavelet);

    // Recorte para dimensão original
    cv::Mat rec_cut = rec(cv::Rect(0, 0, img.cols, img.rows)).clone();
    cv::Mat rec_8u;
    cv::normalize(rec_cut, rec_8u, 0, 255, cv::NORM_MINMAX, CV_8U);
    return rec_8u;
}

int main() {
    // Cria imagem sintética de teste
    cv::Mat img_gray = gerar_imagem_sintetica(256);

    std::vector<int> thresholds = {0, 10, 30, 60, 100};
    std::vector<cv::Mat> imgs_thr;
    imgs_thr.push_back(img_gray);
    std::vector<std::string> titles_thr = {"Original"};

    for (double t : thresholds) {
        cv::Mat rec = dwt_threshold_reconstruct(img_gray, "db4", 2, t);
        double psnr = cv::PSNR(img_gray, rec);
        imgs_thr.push_back(rec);
        titles_thr.push_back("T=" + std::to_string(static_cast<int>(t)) + "  PSNR=" +
                             std::to_string(psnr).substr(0, std::to_string(psnr).find(".") +
2) + " dB");
    }

    // Converte para mm::Image para exibição
    std::vector<mm::Image> imgs_mm;
    for (size_t i = 0; i < imgs_thr.size(); ++i) {
        imgs_mm.push_back(mm::Image(imgs_thr[i]));
    }
    mm::show(imgs_mm, MM_OUT, titles_thr, 3);

    return 0;
}
```

```
Overwriting tmp/fig_05_dwt_reconstrucao.cpp
```

```
!g++ -I. -std=c++17 -DMM_USE_OPENCV -I/usr/include/opencv4 tmp/fig_05_dwt_reconstrucao.cpp -o
tmp/fig_05_dwt_reconstrucao -lopencv_stitching -lopencv_alphamat -lopencv_aruco
-lopencv_barcode -lopencv_bgsegm -lopencv_bioinspired -lopencv_ccalib -lopencv_dnn_objdetect
-lopencv_dnn_superres -lopencv_dpm -lopencv_face -lopencv_freetype -lopencv_fuzzy -lopencv_hdf
-lopencv_hfs -lopencv_img_hash -lopencv_intensity_transform -lopencv_line_descriptor
-lopencv_mcc -lopencv_quality -lopencv_rapid -lopencv_reg -lopencv_rgbd -lopencv_saliency
-lopencv_shape -lopencv_stereo -lopencv_structured_light -lopencv_phase_unwrapping
-lopencv_superres -lopencv_optflow -lopencv_surface_matching -lopencv_tracking
-lopencv_highgui -lopencv_datasets -lopencv_text -lopencv_plot -lopencv_ml -lopencv_videostab
-lopencv_videoio -lopencv_viz -lopencv_wechat_qrcode -lopencv_ximgproc -lopencv_video
-lopencv_xobjdetect -lopencv_objdetect -lopencv_calib3d -lopencv_imgcodecs -lopencv_features2d
-lopencv_dnn -lopencv_flann -lopencv_xphoto -lopencv_photo -lopencv_imgproc -lopencv_core \
  && ./tmp/fig_05_dwt_reconstrucao \
  && test -f "tmp/fig_05_dwt_reconstrucao.png" \
  || echo "  mm::show não gravou tmp/fig_05_dwt_reconstrucao.png"
```

```
[1] Original
[2] T=0  PSNR=19.5 dB
[3] T=10  PSNR=19.6 dB
[4] T=30  PSNR=22.9 dB
[5] T=60  PSNR=21.5 dB
[6] T=100  PSNR=18.5 dB
```

```
try:
    mm.show(mm.read("tmp/fig_05_dwt_reconstrucao.png"), figsize=(14, 10))
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + "
    tmp/fig_05_dwt_reconstrucao.png (ver a versao Python)")
```

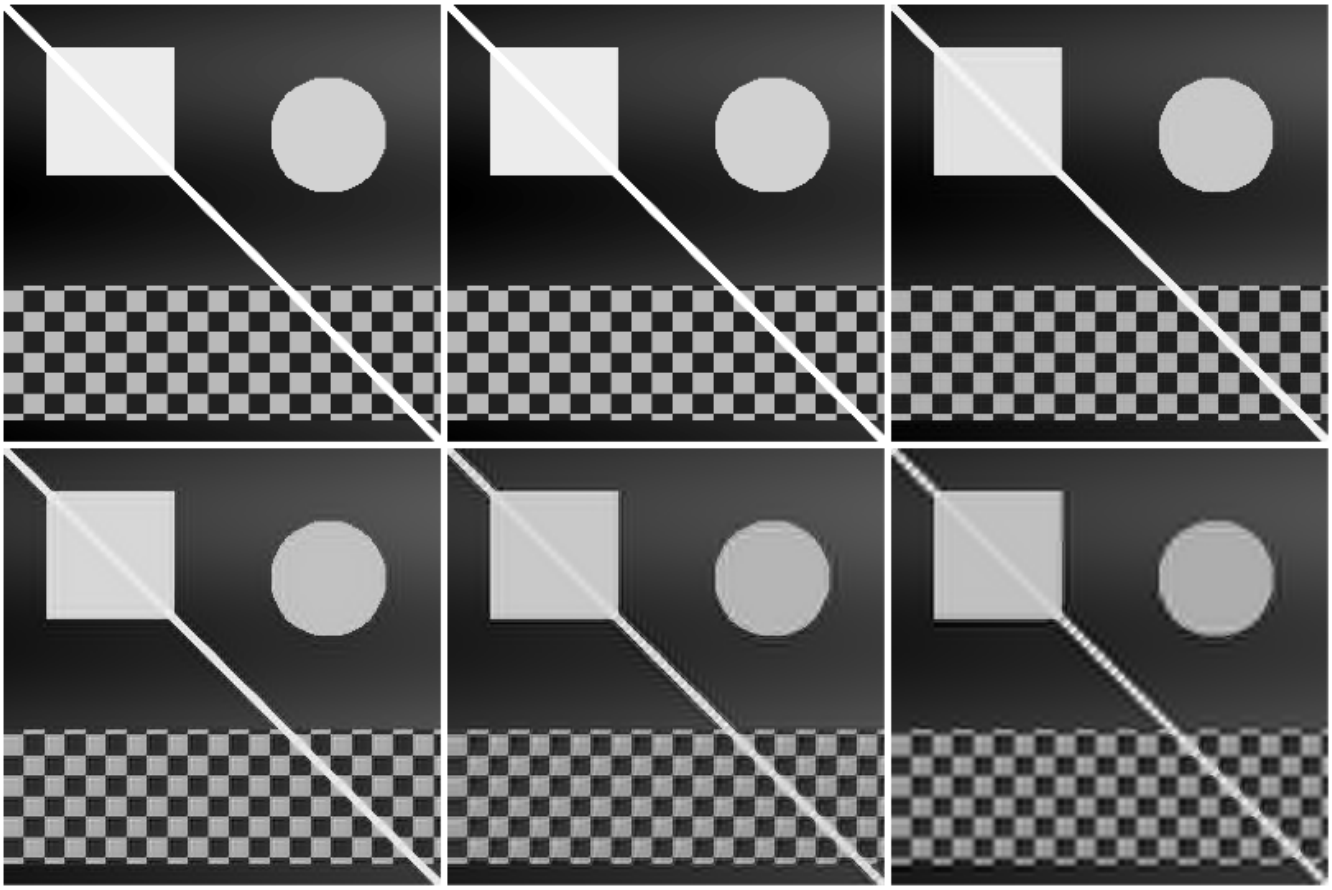


Figure 5.22: Reconstrução *wavelet* com limiarização de coeficientes (*hard thresholding*): à medida que o limiar aumenta, mais detalhes são zerados, produzindo imagens progressivamente mais suaves. Métrica PSNR quantifica a perda de qualidade sobre o padrão sintético.

Synthèse — Fourier vs. *wavelets* : quand utiliser chaque approche ?

La Table 5.5 synthétise les principales différences structurelles et opérationnelles entre la transformée de Fourier discrète (TFD) et la transformée en *ondelettes* discrète (TOD).

Tableau 5.5: Comparaison entre la transformée de Fourier discrète (TFD) et la transformée en *ondelettes* discrète (TOD), mettant en évidence leurs principales caractéristiques et applications.

Critère	Fourier (TFD)	*Ondelettes* (TOD)
Fonctions de base	Sinusoïdes à support infini	Fonctions à support compact
Localisation spatiale	Non explicite (globale)	Explicite (locale)
Filtrage spectral	Excellent pour un contrôle fin des fréquences	Basé sur des sous-bandes (échelles)
Compression d'images	Base de la TCD (JPEG traditionnel)	Base de la TOD (JPEG 2000)
Analyse multi-échelle	Non	Oui
Suppression du bruit périodique	Très efficace	Peu indiquée
Signaux non stationnaires	Limitée	Très efficace

En termes pratiques, la TFD s'impose comme l'outil idéal pour l'analyse spectrale pure, la conception de filtres sélectifs dans le domaine fréquentiel et l'atténuation des bruits périodiques et harmoniques. En revanche, la TOD excelle dans les scénarios exigeant une préservation rigoureuse de la localisation spatiale des

caractéristiques associée à leur contenu fréquentiel, se distinguant dans la compression de données, l'analyse multirésolution et le traitement des transitions abruptes. Ainsi, les deux transformées doivent être comprises comme des techniques parfaitement complémentaires, traçant des voies distinctes et spécifiques pour la résolution de problèmes en TNI-VC.

i Analogies avec l'audio : limites et précautions

Lorsqu'on établit des analogies entre le traitement d'images et l'audio, il est important de prendre en compte les différences fondamentales :

- Dans les systèmes audio stéréo/multicanaux, la phase entre les canaux est cruciale pour la perception de la localisation spatiale (différences interaurales de phase et de temps).
- Dans les systèmes monauraux, la phase a une influence perceptuelle limitée — l'oreille humaine est relativement insensible à la phase absolue des composantes sinusoïdales isolées.
- Dans les images, la phase de la TFD est toujours fondamentale pour la localisation spatiale des structures, qu'il s'agisse d'une image monochrome ou couleur.

L'analogie entre la phase en audio et la phase en images doit être utilisée avec prudence, en soulignant que, bien que toutes deux portent des informations sur l'organisation spatiale/temporelle du signal, les mécanismes perceptuels sont fondamentalement différents.

5.7 Compression d'images

Alors que les *wavelets* établissent la fondation théorique du standard JPEG 2000, le standard JPEG traditionnel repose sur la **Transformée en Cosinus Discrète (DCT, *Discrete Cosine Transform*)**. Malgré les différences structurelles, les deux approches partagent le même principe fondamental : compacter l'énergie de l'image en un nombre réduit de coefficients et éliminer les composantes de moindre importance avec un impact visuel minimal.

L'objectif central de la compression est de réduire le volume de données nécessaire au stockage ou à la transmission d'une image. Ce processus est rendu possible par l'identification et l'élimination des **redondances** structurelles et perceptuelles.

5.7.1 Taxonomie des Redondances

Le développement des algorithmes de compression repose sur l'identification et l'élimination de trois catégories principales de redondance, synthétisées dans la Table 5.6.

Tableau 5.6: Catégories de redondance dans les images numériques et leurs mécanismes d'exploitation respectifs.

Type	Définition	Approche d'exploitation
Spatiale (interpixel)	Forte corrélation et dépendance statistique entre pixels voisins.	DCT, DWT et codage prédictif.
Spectrale (intercanal)	Corrélation statistique entre les canaux de couleur d'une même image.	Transformations d'espace colorimétrique (ex : RGB vers YC_bC_r).
Psychovisuelle	Insensibilité du système visuel humain (SVH) aux variations de haute fréquence et de faible contraste.	Processus de quantification sélective des coefficients.

Selon la préservation de l'information originale après le processus de décodage, les méthodes de compression se divisent en deux classes fondamentales :

- **Sans perte (*lossless*) :** Garantit une reconstruction bit à bit identique à l'image originale. Elle est employée dans des scénarios où l'intégrité des données est strictement critique, comme dans l'imagerie médicale, les diagnostics par imagerie et le stockage de documents textuels.
- **Avec perte (*lossy*) :** Admet l'introduction d'une distorsion contrôlée du signal en échange de taux de compression substantiellement plus élevés. C'est l'approche standard pour les photographies grand public et le *streaming* vidéo, écosystèmes dans lesquels le SVH tolère de légères atténuations haute fréquence sans perception de dégradation de la qualité visuelle.

5.7.2 Transformée en Cosinus Discrète (DCT-II 2D)

La **Transformée en Cosinus Discrète** (DCT) constitue l'opération centrale du standard JPEG. Contrairement à la TFD, qui utilise une base complexe, la DCT repose sur des fonctions trigonométriques purement réelles. Pour un bloc d'image $f(x, y)$ de dimensions $N \times N$, la **DCT-II 2D** mappe le signal spatial vers le domaine des fréquences spatiales, générant la matrice de coefficients $C(u, v)$ au moyen de :

$$C(u,v) = \alpha(u)\,\alpha(v) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x,y)\, \cos\left[\frac{\pi(2x+1)u}{2N}\right] \cos\left[\frac{\pi(2y+1)v}{2N}\right] \tag{5.9}$$

où les facteurs de normalisation orthogonale sont donnés par $\alpha(0) = \sqrt{1/N}$ et $\alpha(k) = \sqrt{2/N}$ pour $k > 0$.

Chaque coefficient $C(u, v)$ quantifie la contribution — ou le « poids » — d'une fréquence spatiale spécifique au sein de ce bloc. Le terme $C(0, 0)$ est appelé **composante DC** et représente l'intensité moyenne du bloc (fréquence nulle). Les autres coefficients, désignés sous le nom de **composantes AC** (*Alternating Current*), correspondent aux fréquences spatiales progressivement plus élevées.

5.7.3 Les fonctions de base de la DCT

D'un point de vue géométrique, la Équation 5.9 réalise la projection du bloc de pixels sur un ensemble de fonctions orthogonales. Pour le cas standard du JPEG ($N = 8$), le bloc spatial est décomposé en une combinaison linéaire de **64 fonctions de base** bidimensionnelles, notées $B_{u,v}(x, y)$ et générées par le produit de fonctions cosinusoïdales :

$$B_{u,v}(x,y) = \cos\left[\frac{\pi(2x+1)u}{16}\right] \cos\left[\frac{\pi(2y+1)v}{16}\right]$$

Ainsi, l'opération inverse peut être interprétée comme la reconstruction exacte du bloc original par la somme pondérée de ces 64 matrices de base, où chaque coefficient $C(u, v)$ agit comme le poids analytique de sa composante harmonique respective.

La **fréquence spatiale** indiquée par les indices (u, v) détermine le nombre de cycles d'oscillation le long des dimensions horizontales et verticales du bloc. Comme illustré dans la Figure 5.23 — dont le code isole chaque base en appliquant la transformation inverse sur des impulsions unitaires —, ces 64 fonctions sont organisées en une matrice 8×8. Le coin supérieur gauche ($u = 0, v = 0$) présente le motif uniforme de fréquence nulle (DC), tandis que la progression vers la droite (axe u) ou vers le bas (axe v) représente des variations harmoniques progressivement plus importantes, traduisant des transitions rapides, des contours et des textures dans les orientations horizontales, verticales et diagonales.

> **i** DCT vs DFT : avantage de la compaction d'énergie
>
> La DCT et la DFT mappent toutes deux un bloc spatial $N \times N$ en une matrice de coefficients de même dimension. Cependant, pour les images naturelles, la DCT présente une plus grande efficacité en matière de **compaction d'énergie** dans les basses fréquences. Cela s'explique par le fait que la DCT suppose implicitement une symétrie paire du signal aux frontières du bloc, ce qui équivaut à une extension périodique continue, minimisant ainsi l'effet de diffusion spectrale (*ringing*). Par conséquent,

la plupart des coefficients AC décroissent rapidement vers des valeurs proches de zéro, optimisant le *pipeline* de compression sans introduire de dégradation visuelle perceptible.

```
%%writefile tmp/fig_05_dct_basis.cpp
#define MM_OUT "tmp/fig_05_dct_basis.png"
//| label: fig-05-dct-basis
//| fig-cap: "O Alfabeto Visual do JPEG: As 64 funções de base da DCT-II. O coeficiente DC
fica no topo esquerdo (suave). Ao descer e avançar à direita, a oscilação espacial aumenta
drasticamente."
//| echo: true
//| output: true

#include <opencv2/opencv.hpp>
#include <vector>
#include <string>
#include "morph.hpp"
#include <filesystem>

int main() {
    // Cada base é a IDCT de um único coeficiente unitário - montadas num mosaico 8×8.
    int tile = 32;
    cv::Mat montagem = cv::Mat::zeros(8 * tile, 8 * tile, CV_8U);
    for (int i = 0; i < 8; i++) {
        for (int j = 0; j < 8; j++) {
            cv::Mat coef = cv::Mat::zeros(8, 8, CV_64F);
            coef.at<double>(i, j) = 1.0;
            cv::Mat base = mm::idct2(coef);
            cv::Mat base_normalized;
            cv::normalize(base, base_normalized, 0, 255, cv::NORM_MINMAX, CV_8U);
            cv::Mat base_resized;
            cv::resize(base_normalized, base_resized, cv::Size(tile, tile), 0, 0,
cv::INTER_NEAREST);
            base_resized.copyTo(montagem(cv::Rect(j * tile, i * tile, tile, tile)));
        }
    }

    std::string titulo = "As 64 bases da DCT-II 8×8 (DC no topo-esquerdo)";
    mm::show(std::vector<mm::Image>{mm::Image(montagem)}, MM_OUT, {titulo}, 1);


// [pdi:panel-io] auto-generated - do not edit by hand
std::filesystem::create_directories("tmp");
mm::write(montagem, "tmp/fig_05_dct_basis_0.png");
// [pdi:panel-io:end]
return 0;
}
```

```
Overwriting tmp/fig_05_dct_basis.cpp
```

```
!g++ -I. -std=c++17 -DMM_USE_OPENCV -I/usr/include/opencv4 tmp/fig_05_dct_basis.cpp -o
tmp/fig_05_dct_basis -lopencv_stitching -lopencv_alphamat -lopencv_aruco -lopencv_barcode
-lopencv_bgsegm -lopencv_bioinspired -lopencv_ccalib -lopencv_dnn_objdetect
-lopencv_dnn_superres -lopencv_dpm -lopencv_face -lopencv_freetype -lopencv_fuzzy -lopencv_hdf
-lopencv_hfs -lopencv_img_hash -lopencv_intensity_transform -lopencv_line_descriptor
-lopencv_mcc -lopencv_quality -lopencv_rapid -lopencv_reg -lopencv_rgbd -lopencv_saliency
-lopencv_shape -lopencv_stereo -lopencv_structured_light -lopencv_phase_unwrapping
-lopencv_superres -lopencv_optflow -lopencv_surface_matching -lopencv_tracking
-lopencv_highgui -lopencv_datasets -lopencv_text -lopencv_plot -lopencv_ml -lopencv_videostab
-lopencv_videoio -lopencv_viz -lopencv_wechat_qrcode -lopencv_ximgproc -lopencv_video
-lopencv_xobjdetect -lopencv_objdetect -lopencv_calib3d -lopencv_imgcodecs -lopencv_features2d
-lopencv_dnn -lopencv_flann -lopencv_xphoto -lopencv_photo -lopencv_imgproc -lopencv_core \
  && ./tmp/fig_05_dct_basis \
  && test -f "tmp/fig_05_dct_basis.png" \
  || echo "  mm::show não gravou tmp/fig_05_dct_basis.png"
```

```
[1] As 64 bases da DCT-II 8×8 (DC no topo-esquerdo)
```

```
try:
    mm.show(
        [
            mm.read("tmp/fig_05_dct_basis_0.png"),
        ],
        titles=[
            'As 64 bases da DCT-II 8×8 (DC no topo-esquerdo)',
        ],
        cols=1,
    )
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + " tmp/fig_05_dct_basis_0.png
    (ver a versao Python)")
```

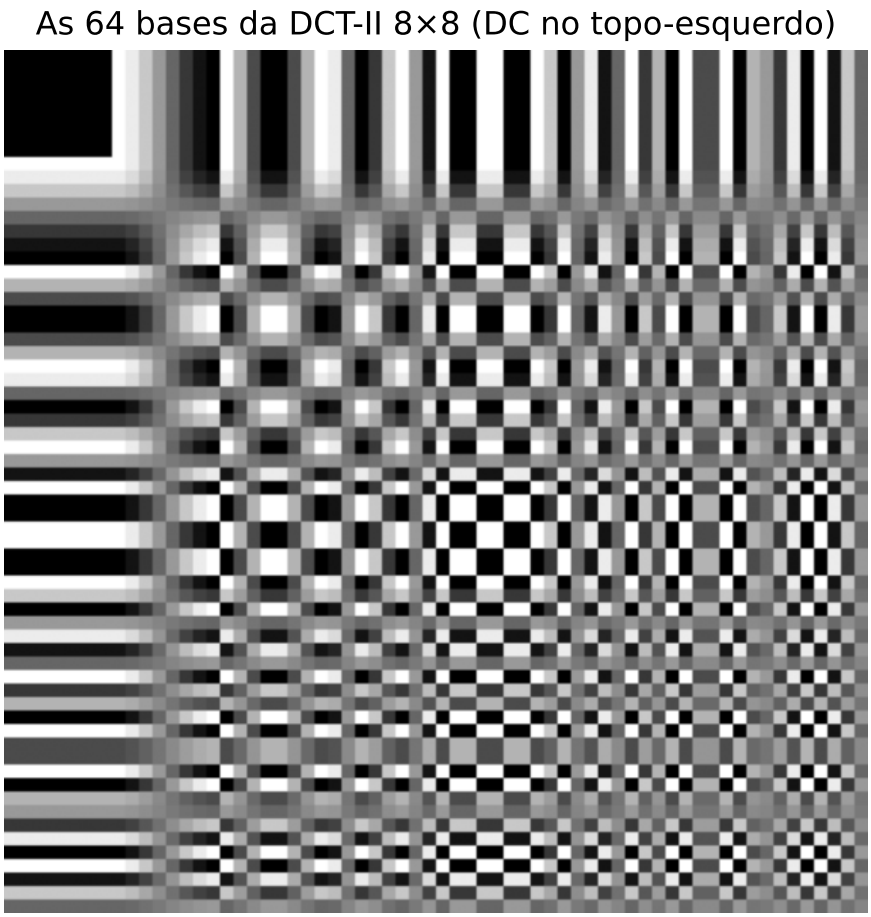


Figure 5.23: O Alfabeto Visual do JPEG: As 64 funções de base da DCT-II. O coeficiente DC fica no topo esquerdo (suave). Ao descer e avançar à direita, a oscilação espacial aumenta drasticamente.

5.7.4 Concentration d'Énergie et Reconstruction Progressive

Avant l'application de la DCT, les pixels du bloc d'intensité sont systématiquement translatés (en soustrayant 128 pour les images 8 bits) afin de centrer le signal autour de zéro, éliminant ainsi les composantes continues superflues. Lors du calcul de la DCT sur le bloc résultant, la propriété de **compaction d'énergie** devient évidente : la quasi-totalité de la variance et de l'information de l'image originale se concentre dans le coefficient DC ($C(0,0)$) et dans les premiers harmoniques AC de basse fréquence.

La Figure 5.24 illustre ce phénomène par une reconstruction progressive par troncature abrupte. Au lieu d'utiliser l'ensemble des 64 coefficients, l'algorithme ne conserve que les k premiers composants — sélectionnés sur la base d'un balayage qui privilégie les basses fréquences spatiales — et annule les autres.

La synthèse inverse (**IDCT**) réalisée avec seulement une fraction des coefficients (comme 15 % ou 30 %) est déjà capable de récupérer les structures et l'éclairage macro du bloc de pixels original. À mesure que les harmoniques de fréquences plus élevées sont progressivement réincorporés, les détails fins et les transitions rapides sont restaurés. Ce comportement valide le principe de la compression perceptuelle : les hautes fréquences écartées possèdent peu d'énergie et leur absence, en conditions normales, génère un impact visuel secondaire sur la perception de l'observateur.

```
%%writefile tmp/fig_05_dct_bloco.cpp
#define MM_OUT "tmp/fig_05_dct_bloco.png"
//| label: fig-05-dct-bloco
//| fig-cap: "DCT 2D en bloc 8×8 : coefficients et reconstruction progressive."
//| echo: true
//| output: true
```

```
#include <opencv2/opencv.hpp>
#include <vector>
#include <string>
#include <algorithm>
#include <cmath>
#include <iostream>
#include <iomanip>
#include "morph.hpp"

int main() {
// [pdi:state-io] auto-generated - do not edit by hand
mm::Image img_gray = mm::_read_state("tmp/state/img_gray_20.png");
// [pdi:state-io:end]

    // img_gray est fourni automatiquement (mm::Image)

    //   Bloc 8×8 centré de l'image
    int cy = img_gray.h / 2;
    int cx = img_gray.w / 2;

    // Extraction du bloc 8×8 et conversion en CV_64F avec soustraction de 128
    cv::Mat img_mat = img_gray;
    cv::Mat bloco_f;
    img_mat(cv::Rect(cx, cy, 8, 8)).convertTo(bloco_f, CV_64F);
    bloco_f -= 128.0;

    // DCT 2D du bloc
    cv::Mat C = mm::dct2(bloco_f);

    std::cout << "Coefficients DCT du bloc 8×8:" << std::endl;
    for(int i = 0; i < 8; i++) {
        for(int j = 0; j < 8; j++) {
            std::cout << std::setw(5) << std::round(C.at<double>(i,j)) << " ";
        }
        std::cout << std::endl;
    }

    // Calcul des énergies
    double energia_dc = C.at<double>(0,0) * C.at<double>(0,0);
    double energia_total = 0.0;
    for(int i = 0; i < 8; i++) {
        for(int j = 0; j < 8; j++) {
            energia_total += C.at<double>(i,j) * C.at<double>(i,j);
        }
    }

    std::cout << "\nÉnergie DC     : " << std::fixed << std::setprecision(1) << energia_dc <<
std::endl;
    std::cout << "Énergie totale : " << energia_total << std::endl;
    std::cout << "Fraction en DC : " << std::fixed << std::setprecision(0)
              << (energia_dc / energia_total * 100.0) << "% ← concentration d'énergie" <<
              std::endl;

    //   Reconstruction progressive
    cv::Mat bloco_orig;
    bloco_f.copyTo(bloco_orig);
    bloco_orig += 128.0;
    cv::Mat bloco_orig_8u;
    bloco_orig.convertTo(bloco_orig_8u, CV_8U);
```

```
    std::vector<mm::Image> imgs_rec;
    imgs_rec.push_back(mm::Image(bloco_orig_8u));

    std::vector<std::string> titles_rec;
    titles_rec.push_back("Bloc original\n(8×8 pixels)");

    // Liste des paires (u,v) triées par somme u+v
    std::vector<std::pair<int,int>> indices;
    for(int u = 0; u < 8; u++) {
        for(int v = 0; v < 8; v++) {
            indices.push_back({u, v});
        }
    }
    std::sort(indices.begin(), indices.end(),
              [](const std::pair<int,int>& a, const std::pair<int,int>& b) {
                  return (a.first + a.second) < (b.first + b.second);
              });

    std::vector<int> keeps = {1, 4, 10, 20, 40, 64};

    for(int keep : keeps) {
        // Création de la matrice tronquée
        cv::Mat C_trunc = cv::Mat::zeros(8, 8, CV_64F);
        for(int k = 0; k < keep; k++) {
            int u = indices[k].first;
            int v = indices[k].second;
            C_trunc.at<double>(u,v) = C.at<double>(u,v);
        }

        // Reconstruction IDCT
        cv::Mat rec_f = mm::idct2(C_trunc);
        rec_f += 128.0;

        cv::Mat rec_8u;
        rec_f.convertTo(rec_8u, CV_8U);
        cv::Mat rec_clipped;
        cv::threshold(rec_8u, rec_clipped, 255, 255, cv::THRESH_TRUNC);
        cv::threshold(rec_clipped, rec_clipped, 0, 0, cv::THRESH_TOZERO);

        imgs_rec.push_back(mm::Image(rec_clipped));

        double pct = (keep / 64.0) * 100.0;
        titles_rec.push_back(std::to_string(keep) + " coef.\n(" +
                             std::to_string((int)pct) + "% du total)");
    }

    mm::show(imgs_rec, MM_OUT, titles_rec, 4);

    return 0;
}
```

```
Overwriting tmp/fig_05_dct_bloco.cpp
```

```
!g++ -I. -std=c++17 -DMM_USE_OPENCV -I/usr/include/opencv4 tmp/fig_05_dct_bloco.cpp -o
tmp/fig_05_dct_bloco -lopencv_stitching -lopencv_alphamat -lopencv_aruco -lopencv_barcode
-lopencv_bgsegm -lopencv_bioinspired -lopencv_ccalib -lopencv_dnn_objdetect
-lopencv_dnn_superres -lopencv_dpm -lopencv_face -lopencv_freetype -lopencv_fuzzy -lopencv_hdf
-lopencv_hfs -lopencv_img_hash -lopencv_intensity_transform -lopencv_line_descriptor
-lopencv_mcc -lopencv_quality -lopencv_rapid -lopencv_reg -lopencv_rgbd -lopencv_saliency
-lopencv_shape -lopencv_stereo -lopencv_structured_light -lopencv_phase_unwrapping
-lopencv_superres -lopencv_optflow -lopencv_surface_matching -lopencv_tracking
-lopencv_highgui -lopencv_datasets -lopencv_text -lopencv_plot -lopencv_ml -lopencv_videostab
-lopencv_videoio -lopencv_viz -lopencv_wechat_qrcode -lopencv_ximgproc -lopencv_video
-lopencv_xobjdetect -lopencv_objdetect -lopencv_calib3d -lopencv_imgcodecs -lopencv_features2d
-lopencv_dnn -lopencv_flann -lopencv_xphoto -lopencv_photo -lopencv_imgproc -lopencv_core \
  && ./tmp/fig_05_dct_bloco \
  && test -f "tmp/fig_05_dct_bloco.png" \
  || echo "  mm::show não gravou tmp/fig_05_dct_bloco.png"
```

```
Coefficients DCT du bloc 8×8:
  192   -48     1     8     0    -1    -0    -0
  -96    54     7   -10    -0     0     0     0
   14   -14     8    -0     0    -0    -0     0
   -0     0   -11     0     0     0    -0     0
    9   -11    -0     0     1    -0     0    -0
   -0     0    -0    -0    -0    -0    -0     0
   -0     0     0    -0    -1     0     0    -0
    0    -0    -0    -0     0     0     0     0

Énergie DC     : 36768.1
Énergie totale : 52234.0
Fraction en DC : 70% ← concentration d'énergie
[1] Bloc original
(8×8 pixels)
[2] 1 coef.
(1% du total)
[3] 4 coef.
(6% du total)
[4] 10 coef.
(15% du total)
[5] 20 coef.
(31% du total)
[6] 40 coef.
(62% du total)
[7] 64 coef.
(100% du total)
```

```
try:
    mm.show(mm.read("tmp/fig_05_dct_bloco.png"), figsize=(12, 7))
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + " tmp/fig_05_dct_bloco.png
    (ver a versao Python)")
```

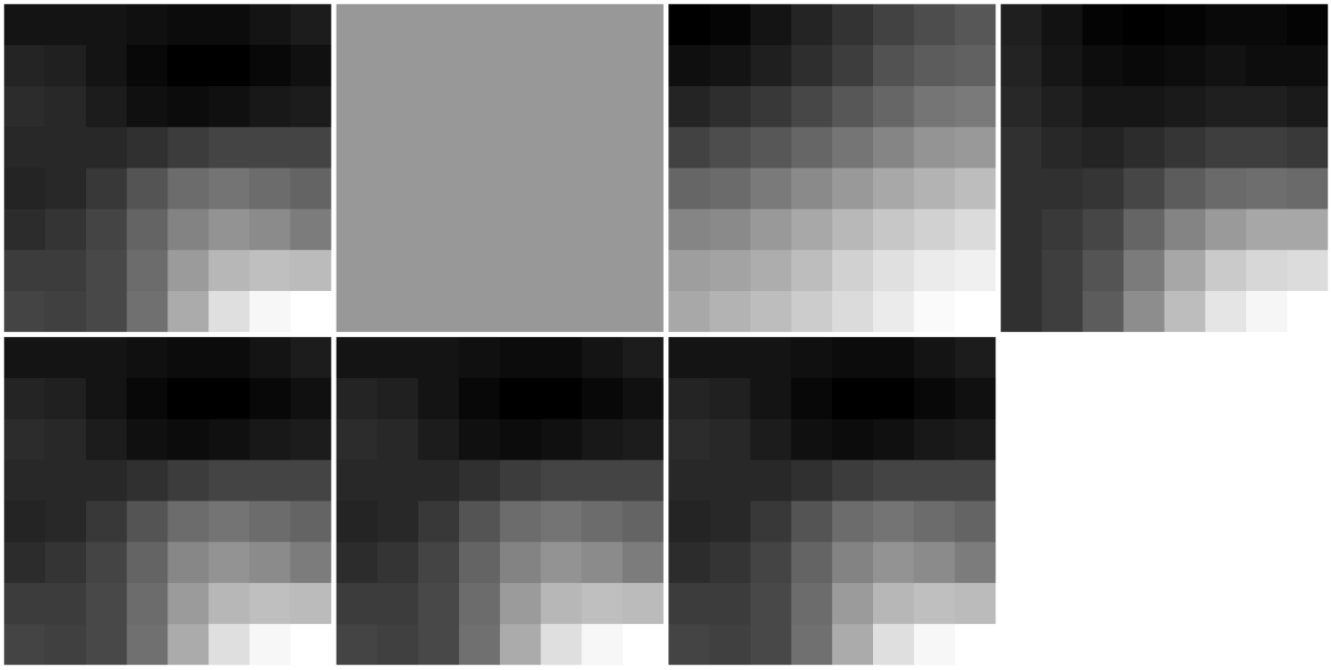


Figure 5.24: DCT 2D em bloco 8×8: coeficientes e reconstrução progressiva.

5.7.5 Le *Pipeline* de Compression JPEG

La norme JPEG opère en divisant l'image en blocs disjoints de 8×8 pixels, traités par une séquence de transformations spatiales, perceptuelles et statistiques. Le *pipeline* complet de codage est structuré en six étapes principales :

$$\text{RGB} \xrightarrow{\text{(1) } YC_bC_r} \xrightarrow{\text{(2) Sous-échantillonnage}} \xrightarrow{\text{(3) Blocs 8×8}} \xrightarrow{\text{(4) DCT}} \xrightarrow{\text{(5) Quantification}} \xrightarrow{\text{(6) Codage entropique}}$$

La Table 5.7 détaille la fonction analytique et le fondement perceptuel qui justifient chacune de ces étapes.

Tableau 5.7: Étapes du *pipeline* de compression JPEG et leurs fondements de conception respectifs.

Étape	Opération	Fondement perceptuel et statistique
1	Conversion $RGB \to YC_bC_r$	Sépare la luminance (Y) de la chrominance (C_b, C_r). Le système visuel humain (SVH) présente une plus grande sensibilité aux variations de luminosité qu'aux variations de couleur.
2	Sous-échantillonnage de la chrominance (ex. : 4:2:0)	Réduit la résolution spatiale des canaux de couleur de moitié, en éliminant des données redondantes avec un impact visuel négligeable.
3–4	Centrage et application de la DCT 8×8	Translates les pixels dans l'intervalle $[-128, 127]$ et compresse l'énergie spectrale du bloc dans les coefficients de basse fréquence.

Étape	Opération	Fondement perceptuel et statistique
5	Quantification linéaire sélective	Divise chaque coefficient $C(u,v)$ par l'élément correspondant de la matrice $Q(u,v)$, avec arrondi entier. Constitue la principale source de compression avec perte.
6	Balayage en zigzag et codage	Ordonne les coefficients quantifiés pour maximiser les séquences nulles consécutives, optimisant le codage par longueur de plage (RLE) et le codage de Huffman.

La **matrice de quantification** $Q(u,v)$ est le mécanisme central de contrôle du compromis entre taux de compression et qualité visuelle. Dans l'algorithme pratique de la Figure 5.25, le facteur de qualité spécifié par l'utilisateur (échelle de 1 à 100) est converti en un scalaire qui paramètre la sévérité de la matrice Q. Des valeurs de qualité réduites élargissent les diviseurs de $Q(u,v)$, forçant le tronquement en masse des coefficients AC à zéro. Lorsque cette élimination est excessive, la discontinuité aux frontières des blocs adjacents n'est pas atténuée lors de la reconstruction, générant ce que l'on appelle les **artefacts de bloc** (*blocking artifacts*).

La Logique du Balayage en Zigzag

L'efficacité du codeur entropique subséquent à la quantification dépend directement de l'ordonnancement des données. Comme la DCT concentre l'énergie vitale dans le coin supérieur gauche de la matrice (basses fréquences) et repousse les coefficients nuls vers les extrémités opposées, la lecture linéaire par lignes ou par colonnes fragmenterait les séquences de zéros.

L'ordonnancement en zigzag résout cette limitation en parcourant la matrice en diagonale, dans l'ordre croissant de la fréquence spatiale. Ce mappage regroupe les coefficients significatifs au début du vecteur et concentre les coefficients nuls en une seule séquence continue à la fin du tableau, permettant à l'algorithme RLE de coder de grands blocs de données de manière compacte et efficace.

> **i** Qu'est-ce que le RLE ?
>
> **RLE** (*Run-Length Encoding*) est une technique de compression sans perte qui code des séquences consécutives de valeurs identiques — en particulier des **zéros** — comme une paire (compteur, valeur). Dans le JPEG, après le balayage en zigzag, les coefficients quantifiés sont organisés de sorte que les zéros se concentrent à la fin du vecteur. Le RLE compresse ensuite cette longue course de zéros avec une extrême efficacité, optimisant le stockage et la transmission de l'image compressée.

```
%%writefile tmp/fig_05_jpeg_pipeline.cpp
#define MM_OUT "tmp/fig_05_jpeg_pipeline.png"
#include <opencv2/opencv.hpp>
#include "morph.hpp"
#include <string>
#include <vector>

int main() {
    //| label: fig-05-jpeg-pipeline
```

```
    //| fig-cap: "*Pipeline* JPEG simplifié appliqué à l'image classique du *Cameraman* : DCT
    en blocs 8×8, quantification avec différents facteurs de qualité et reconstruction via
    IDCT. Les artefacts de bloc (*blocking artifacts*) deviennent visuellement évidents pour
    des facteurs de qualité réduits ($Q=10$ et $Q=25$)."
    //| echo: true
    //| output: true

    // Pipeline JPEG (DCT 8x8 -> quantification -> IDCT) via mm.jpegCompress,
    // sur l'image classique du Cameraman (asset du chapitre) réduite à 256x256.
    mm::Image img_orig = mm::read("imagens/cameraman.png");
    cv::Mat img_resized;
    cv::resize(cv::Mat(img_orig), img_resized, cv::Size(256, 256));
    mm::Image img_src = mm::gray(mm::Image(img_resized));

    std::vector<mm::Image> imgs;
    std::vector<std::string> titles;
    imgs.push_back(img_src);
    titles.push_back("Original (Cameraman)");

    std::vector<int> qualities = {10, 25, 50, 75, 90};
    for (int q : qualities) {
        mm::Image rec = mm::jpegCompress(img_src, q);
        imgs.push_back(rec);
        double psnr_val = mm::psnr(img_src, rec);
        titles.push_back("Q=" + std::to_string(q) + " (PSNR=" +
                         std::to_string(psnr_val).substr(0, std::to_string(psnr_val).find(".")
+ 2) + " dB)");
    }

    mm::show(imgs, MM_OUT, titles, 3);

    return 0;
}
```

```
Overwriting tmp/fig_05_jpeg_pipeline.cpp
```

```
!g++ -I. -std=c++17 -DMM_USE_OPENCV -I/usr/include/opencv4 tmp/fig_05_jpeg_pipeline.cpp -o
tmp/fig_05_jpeg_pipeline -lopencv_stitching -lopencv_alphamat -lopencv_aruco -lopencv_barcode
-lopencv_bgsegm -lopencv_bioinspired -lopencv_ccalib -lopencv_dnn_objdetect
-lopencv_dnn_superres -lopencv_dpm -lopencv_face -lopencv_freetype -lopencv_fuzzy -lopencv_hdf
-lopencv_hfs -lopencv_img_hash -lopencv_intensity_transform -lopencv_line_descriptor
-lopencv_mcc -lopencv_quality -lopencv_rapid -lopencv_reg -lopencv_rgbd -lopencv_saliency
-lopencv_shape -lopencv_stereo -lopencv_structured_light -lopencv_phase_unwrapping
-lopencv_superres -lopencv_optflow -lopencv_surface_matching -lopencv_tracking
-lopencv_highgui -lopencv_datasets -lopencv_text -lopencv_plot -lopencv_ml -lopencv_videostab
-lopencv_videoio -lopencv_viz -lopencv_wechat_qrcode -lopencv_ximgproc -lopencv_video
-lopencv_xobjdetect -lopencv_objdetect -lopencv_calib3d -lopencv_imgcodecs -lopencv_features2d
-lopencv_dnn -lopencv_flann -lopencv_xphoto -lopencv_photo -lopencv_imgproc -lopencv_core \
  && ./tmp/fig_05_jpeg_pipeline \
  && test -f "tmp/fig_05_jpeg_pipeline.png" \
  || echo "  mm::show não gravou tmp/fig_05_jpeg_pipeline.png"
```

```
[1] Original (Cameraman)
[2] Q=10 (PSNR=28.0 dB)
[3] Q=25 (PSNR=30.6 dB)
[4] Q=50 (PSNR=32.8 dB)
[5] Q=75 (PSNR=35.1 dB)
[6] Q=90 (PSNR=40.0 dB)
```

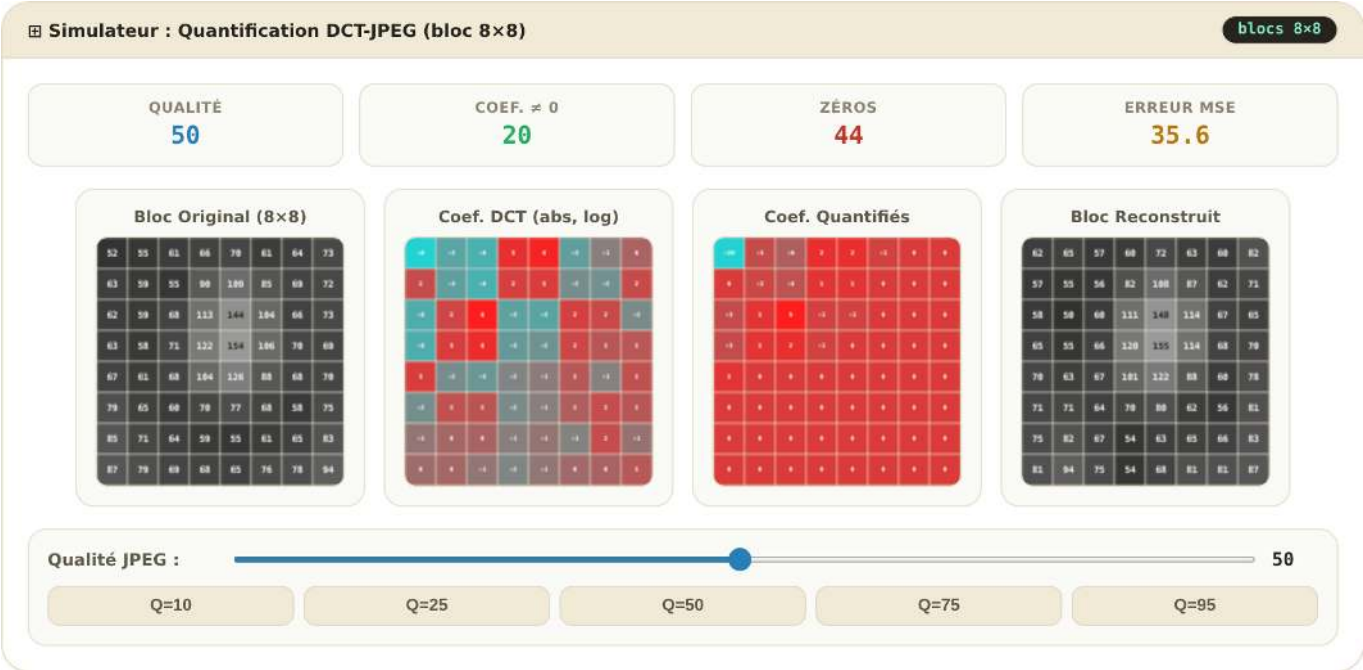
```
try:
    mm.show(mm.read("tmp/fig_05_jpeg_pipeline.png"), figsize=(14, 10))
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + "
    tmp/fig_05_jpeg_pipeline.png (ver a versao Python)")
```

Figure 5.25: *Pipeline* JPEG simplificado aplicado à imagem clássica do *Cameraman*: DCT em blocos 8×8, quantização com diferentes fatores de qualidade e reconstrução via IDCT. Os artefatos de bloco (*blocking artifacts*) tornam-se visualmente evidentes em fatores de qualidade reduzidos ($Q = 10$ e $Q = 25$).

5.7.6 Simulateur interactif : Quantification DCT

Le simulateur de la Figure 5.26 permet d'explorer l'impact du processus de quantification sur un bloc 8×8 extrait d'une image réelle, en synthétisant en temps réel les composantes suivantes :

- **Bloc original et reconstruit :** Représentation directe des pixels dans le domaine spatial en niveaux de gris [0, 255].
- **Coefficients DCT :** Distribution de l'énergie mappée de manière logarithmique sur un dégradé chromatique, mettant en évidence la concentration de l'intensité dans le coin supérieur gauche (basses fréquences).
- **Coefficients quantifiés :** Affichage des valeurs entières résultant de la division par la matrice $Q(u, v)$, rendant visuellement explicite l'apparition massive de coefficients nuls (en tons sombres) à mesure que le facteur de qualité est réduit.
- **Métriques de compression :** Panneau de surveillance qui quantifie l'Erreur Quadratique Moyenne (MSE), le nombre de coefficients préservés et le volume de zéros générés pour le codage entropique.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.fr/cap05/sim-05-dct.html

Figure 5.26: Simulateur interactif de compression DCT-JPEG : ajustez le facteur de qualité et visualisez en temps réel les coefficients nuls, le bloc reconstruit et l'erreur de quantification.

5.8 Comparação de Formatos de Imagem

Le choix d'un format de stockage numérique impacte directement le compromis entre qualité visuelle, taille de fichier et coût computationnel de décodage. Les trois formats les plus pertinents pour les architectures *web* et les systèmes de calcul visuel sont le JPEG, le PNG et le WebP.

5.8.1 Caractéristiques des Formats

La Table 5.8 synthétise les propriétés structurelles des principaux formats d'image matriciels.

Tableau 5.8: Comparaison structurelle entre les principaux formats d'image matriciels.

Caractéristique	JPEG	PNG	WebP
Compression	Avec perte	Sans perte	Avec et sans perte.
Transparence (canal alpha)	Non	Oui	Oui.
Prise en charge de l'animation	Non	Limitée (APNG)	Oui.
Algorithme de base	DCT + Huffman	DEFLATE (LZ77 + Huffman)	VP8 / VP8L.
Idéal pour	Photographie	Graphiques, texte et icônes	Usage universel en environnement Web.
Moins adapté pour	Texte et contours nets	Images photographiques complexes	Compatibilité héritée.

5.8.2 Métriques d'Évaluation de la Qualité

Deux métriques objectives sont largement adoptées pour quantifier la distorsion introduite par les processus de compression :

Pic du Rapport Signal-à-Bruit (PSNR, *Peak Signal-to-Noise Ratio*) :

$$\text{PSNR} = 10 \log_{10}\left(\frac{L^2}{\text{MSE}}\right) \quad [\text{dB}] \tag{5.10}$$

où $L = 255$ pour les images quantifiées sur 8 bits et MSE représente l'**Erreur Quadratique Moyenne** (*Mean Squared Error*). Des valeurs de PSNR supérieures à 40 dB indiquent une excellente fidélité ; entre 30 dB et 40 dB, elles représentent une bonne qualité ; et des valeurs inférieures à 30 dB correspondent à des dégradations visuelles facilement perceptibles.

Indice de Similarité Structurale (SSIM, *Structural Similarity Index*) :

$$\text{SSIM}(f, g) = \frac{(2\mu_f\mu_g + c_1)(2\sigma_{fg} + c_2)}{(\mu_f^2 + \mu_g^2 + c_1)(\sigma_f^2 + \sigma_g^2 + c_2)} \tag{5.11}$$

Le SSIM évalue des fenêtres locales de l'image sur la base de trois composantes complémentaires : la **luminance** (μ_f, μ_g), le **contraste** (σ_f, σ_g) et la **structure** (σ_{fg}), pondérées par des constantes de stabilité c_1 et c_2. L'indice varie dans l'intervalle $[-1, 1]$, où l'unité représente l'identité parfaite. Contrairement au PSNR, le SSIM prend en compte l'organisation spatiale des erreurs, s'alignant ainsi sur la perception du système visuel humain (SVH).

> **i** PSNR vs SSIM : Application de Métriques Perceptuelles
>
> Le PSNR possède une formulation mathématique simple et un faible coût computationnel ; toutefois, il tend à surestimer la qualité des images présentant des distorsions localisées ou à la sous-estimer en cas de variations globales de luminosité tolérées par l'observateur. Le SSIM modélise plus fidèlement la perception biologique, mais exige un effort de traitement plus important. Pour des analyses rigoureuses des codecs, il est recommandé de rapporter les deux métriques statistiques à titre complémentaire.

5.8.3 Inspection Visuelle : Nature des Artefacts de Compression

La nature mathématique du codec détermine le type de dégradation introduit à des débits binaires réduits. Comme illustré dans la Figure 5.27, la compression agressive via DCT dans le standard JPEG segmente l'image en mailles rigides, générant les **artefacts de bloc** (*blocking artifacts*). En revanche, les algorithmes basés sur la codification prédictive ou les représentations soumises à des transformées spatiales avancées (comme le WebP et le JPEG 2000) éliminent les discontinuités de bloc, mais introduisent une perte de texture fine et des flous caractéristiques autour des contours à fort contraste.

```
%%writefile tmp/fig_05_zoom_artefatos.cpp
#define MM_OUT "tmp/fig_05_zoom_artefatos.png"
//| label: fig-05-zoom-artefatos
//| fig-cap: "Análise comparativa de artefatos de compressão sob fator de qualidade reduzido
($Q=10$). À esquerda, observa-se o artefato de bloco característico da discretização por DCT
no JPEG. À direita, evidencia-se o efeito de atenuação e suavização de bordas intrínseco ao
padrão WebP."
//| echo: true
//| output: true
#include <opencv2/opencv.hpp>
#include <vector>
#include <string>
#include <filesystem>
#include "morph.hpp"

int main() {
    // Lê a imagem, converte para escala de cinza e redimensiona
    cv::Mat img_src_mat = cv::imread("imagens/cameraman.png", cv::IMREAD_GRAYSCALE);
    cv::Mat img_src_resized;
```

```
    cv::resize(img_src_mat, img_src_resized, cv::Size(256, 256));
    mm::Image img_src = img_src_resized;

    // Salva com qualidade Q=10 em JPEG e WebP
    cv::imwrite("tmp/zoom_q10.jpg", cv::Mat(img_src), {cv::IMWRITE_JPEG_QUALITY, 10});
    cv::imwrite("tmp/zoom_q10.webp", cv::Mat(img_src), {cv::IMWRITE_WEBP_QUALITY, 10});

    // Função de zoom com interpolação vizinho mais próximo
    auto zoom = [](const cv::Mat& img) {
        cv::Mat crop = img(cv::Rect(150, 120, 80, 80));
        cv::Mat enlarged;
        cv::resize(crop, enlarged, cv::Size(320, 320), 0, 0, cv::INTER_NEAREST);
        return enlarged;
    };

    // Aplica zoom nas três imagens
    cv::Mat z_orig = zoom(cv::Mat(img_src));
    cv::Mat z_jpeg = zoom(cv::imread("tmp/zoom_q10.jpg", cv::IMREAD_GRAYSCALE));
    cv::Mat z_webp = zoom(cv::imread("tmp/zoom_q10.webp", cv::IMREAD_GRAYSCALE));

    // Exibe as três imagens lado a lado
    std::vector<mm::Image> images = {
        mm::Image(z_orig),
        mm::Image(z_jpeg),
        mm::Image(z_webp)
    };
    std::vector<std::string> titles = {
        "Zoom Original",
        "JPEG Q=10 (Artefato de Bloco)",
        "WebP Q=10 (Suavizacao)"
    };
    mm::show(images, MM_OUT, titles, 3);


// [pdi:panel-io] auto-generated - do not edit by hand
std::filesystem::create_directories("tmp");
mm::write(z_orig, "tmp/fig_05_zoom_artefatos_0.png");
mm::write(z_jpeg, "tmp/fig_05_zoom_artefatos_1.png");
mm::write(z_webp, "tmp/fig_05_zoom_artefatos_2.png");
// [pdi:panel-io:end]
return 0;
}
```

```
Overwriting tmp/fig_05_zoom_artefatos.cpp
```

```
!g++ -I. -std=c++17 -DMM_USE_OPENCV -I/usr/include/opencv4 tmp/fig_05_zoom_artefatos.cpp -o
tmp/fig_05_zoom_artefatos -lopencv_stitching -lopencv_alphamat -lopencv_aruco -lopencv_barcode
-lopencv_bgsegm -lopencv_bioinspired -lopencv_ccalib -lopencv_dnn_objdetect
-lopencv_dnn_superres -lopencv_dpm -lopencv_face -lopencv_freetype -lopencv_fuzzy -lopencv_hdf
-lopencv_hfs -lopencv_img_hash -lopencv_intensity_transform -lopencv_line_descriptor
-lopencv_mcc -lopencv_quality -lopencv_rapid -lopencv_reg -lopencv_rgbd -lopencv_saliency
-lopencv_shape -lopencv_stereo -lopencv_structured_light -lopencv_phase_unwrapping
-lopencv_superres -lopencv_optflow -lopencv_surface_matching -lopencv_tracking
-lopencv_highgui -lopencv_datasets -lopencv_text -lopencv_plot -lopencv_ml -lopencv_videostab
-lopencv_videoio -lopencv_viz -lopencv_wechat_qrcode -lopencv_ximgproc -lopencv_video
-lopencv_xobjdetect -lopencv_objdetect -lopencv_calib3d -lopencv_imgcodecs -lopencv_features2d
-lopencv_dnn -lopencv_flann -lopencv_xphoto -lopencv_photo -lopencv_imgproc -lopencv_core \
  && ./tmp/fig_05_zoom_artefatos \
  && test -f "tmp/fig_05_zoom_artefatos.png" \
  || echo "  mm::show não gravou tmp/fig_05_zoom_artefatos.png"
```

```
[1] Zoom Original
[2] JPEG Q=10 (Artefato de Bloco)
[3] WebP Q=10 (Suavizacao)
```

```
try:
    mm.show(
        [
            mm.read("tmp/fig_05_zoom_artefatos_0.png"),
            mm.read("tmp/fig_05_zoom_artefatos_1.png"),
            mm.read("tmp/fig_05_zoom_artefatos_2.png"),
        ],
        titles=[
            'Zoom Original',
            'JPEG Q=10 (Artefato de Bloco)',
            'WebP Q=10 (Suavizacao)',
        ],
        cols=3,
        figsize=(14, 5),
    )
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + "
    tmp/fig_05_zoom_artefatos_0.png (ver a versao Python)")
```

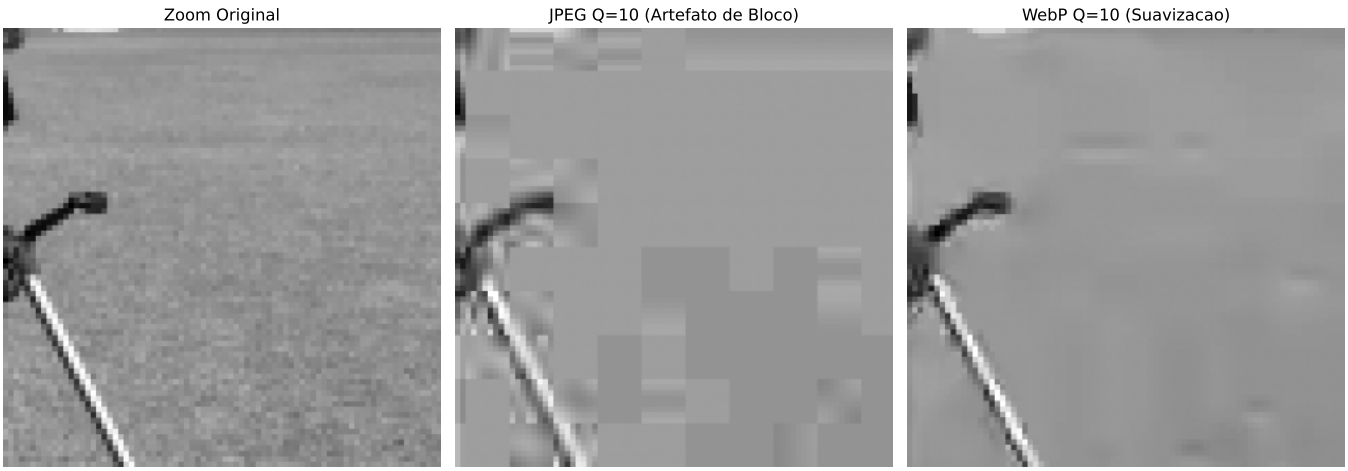


Figure 5.27: Análise comparativa de artefatos de compressão sob fator de qualidade reduzido ($Q = 10$). À esquerda, observa-se o artefato de bloco característico da discretização por DCT no JPEG. À direita, evidencia-se o efeito de atenuação e suavização de bordas intrínseco ao padrão WebP.

5.8.4 Évaluation Quantitative et Spatiale de la Compression

La validation des algorithmes de compression avec perte exige une analyse qui corrèle le coût de stockage à la fidélité du signal reconstruit. Cette évaluation est réalisée de manière complémentaire à travers des courbes de performance globale et par la cartographie locale des distorsions induites par les codeurs.

5.8.4.1 Courbes Débit-Distorsion

La Figure 5.28 présente l'évaluation empirique du *pipeline* JPEG et WebP au moyen de **courbes débit-distorsion**, qui surveillent le gain de compression (taille du fichier en Ko) en fonction du PSNR. Le format PNG sert de ligne de base idéale (PSNR $= \infty$), car sa nature *lossless* empêche toute dégradation, bien qu'il exige un volume de données substantiellement plus important.

L'analyse des courbes démontre la supériorité et l'efficacité du standard WebP par rapport au JPEG traditionnel : pour atteindre un même niveau de fidélité mathématique (comme la plage d'excellente qualité, où PSNR > 40 dB), le codeur WebP génère des fichiers significativement plus petits. Ce comportement

traduit l'impact pratique de l'évolution des algorithmes sur l'optimisation des systèmes de transmission et de stockage numérique.

```
%%writefile tmp/fig_05_formatos_comparacao.cpp
#define MM_OUT "tmp/fig_05_formatos_comparacao.png"
//| label: fig-05-formatos-comparacao
//| fig-cap: "Curva taxa-distorção: PSNR vs tamanho de arquivo para JPEG, WebP e PNG aplicada
à imagem do *Cameraman*."
//| echo: true
//| output: true

#include <opencv2/opencv.hpp>
#include <vector>
#include <string>
#include <filesystem>
#include <cstdio>
#include "morph.hpp"

int main() {
    // Lecture et redimensionnement de l'image Cameraman
    cv::Mat src_full = cv::imread("imagens/cameraman.png", cv::IMREAD_GRAYSCALE);
    cv::Mat src_resized;
    cv::resize(src_full, src_resized, cv::Size(256, 256));
    mm::Image src = src_resized;

    // Tableaux pour courbe JPEG
    std::vector<double> jpeg_kb, jpeg_psnr;
    std::vector<int> jpeg_qualities = {10, 20, 30, 40, 50, 60, 70, 80, 90, 95};
    for (int q : jpeg_qualities) {
        std::string p = "tmp/fmt_q" + std::to_string(q) + ".jpg";
        cv::imwrite(p, cv::Mat(src), {cv::IMWRITE_JPEG_QUALITY, q});
        cv::Mat rec = cv::imread(p, cv::IMREAD_GRAYSCALE);
        jpeg_kb.push_back((double)std::filesystem::file_size(p) / 1024.0);
        jpeg_psnr.push_back(mm::psnr(src, mm::Image(rec)));
    }

    // Tableaux pour courbe WebP
    std::vector<double> webp_kb, webp_psnr;
    std::vector<int> webp_qualities = {30, 50, 70, 85, 95};
    for (int q : webp_qualities) {
        std::string p = "tmp/fmt_w" + std::to_string(q) + ".webp";
        cv::imwrite(p, cv::Mat(src), {cv::IMWRITE_WEBP_QUALITY, q});
        cv::Mat rec = cv::imread(p, cv::IMREAD_GRAYSCALE);
        webp_kb.push_back((double)std::filesystem::file_size(p) / 1024.0);
        webp_psnr.push_back(mm::psnr(src, mm::Image(rec)));
    }

    // Compression PNG sans perte
    std::string p_png = "tmp/fmt.png";
    cv::imwrite(p_png, cv::Mat(src), {cv::IMWRITE_PNG_COMPRESSION, 9});
    double png_kb = (double)std::filesystem::file_size(p_png) / 1024.0;

    // Créer le graphique comparatif
    std::vector<std::vector<double>> xs = {jpeg_kb, webp_kb};
    std::vector<std::vector<double>> ys = {jpeg_psnr, webp_psnr};
    std::vector<std::string> labels = {"JPEG", "WebP"};
    char title[128];
    snprintf(title, sizeof(title), "Curva Taxa-Distorcao (PNG sem perda: %.1f KB)", png_kb);

    mm::Image chart = mm::lineChart(
        xs, ys, labels, {}, title,
```

```
        "Tamanho do arquivo (KB)", "PSNR (dB)"
    );

    // Affichage du graphique
    mm::show(std::vector<mm::Image>{chart}, MM_OUT, {"JPEG x WebP x PNG"}, 1);

// [pdi:panel-io] auto-generated - do not edit by hand
std::filesystem::create_directories("tmp");
mm::write(chart, "tmp/fig_05_formatos_comparacao_0.png");
// [pdi:panel-io:end]
return 0;
}
```

```
Overwriting tmp/fig_05_formatos_comparacao.cpp
```

```
!g++ -I. -std=c++17 -DMM_USE_OPENCV -I/usr/include/opencv4 tmp/fig_05_formatos_comparacao.cpp
-o tmp/fig_05_formatos_comparacao -lopencv_stitching -lopencv_alphamat -lopencv_aruco
-lopencv_barcode -lopencv_bgsegm -lopencv_bioinspired -lopencv_ccalib -lopencv_dnn_objdetect
-lopencv_dnn_superres -lopencv_dpm -lopencv_face -lopencv_freetype -lopencv_fuzzy -lopencv_hdf
-lopencv_hfs -lopencv_img_hash -lopencv_intensity_transform -lopencv_line_descriptor
-lopencv_mcc -lopencv_quality -lopencv_rapid -lopencv_reg -lopencv_rgbd -lopencv_saliency
-lopencv_shape -lopencv_stereo -lopencv_structured_light -lopencv_phase_unwrapping
-lopencv_superres -lopencv_optflow -lopencv_surface_matching -lopencv_tracking
-lopencv_highgui -lopencv_datasets -lopencv_text -lopencv_plot -lopencv_ml -lopencv_videostab
-lopencv_videoio -lopencv_viz -lopencv_wechat_qrcode -lopencv_ximgproc -lopencv_video
-lopencv_xobjdetect -lopencv_objdetect -lopencv_calib3d -lopencv_imgcodecs -lopencv_features2d
-lopencv_dnn -lopencv_flann -lopencv_xphoto -lopencv_photo -lopencv_imgproc -lopencv_core \
  && ./tmp/fig_05_formatos_comparacao \
  && test -f "tmp/fig_05_formatos_comparacao.png" \
  || echo "  mm::show não gravou tmp/fig_05_formatos_comparacao.png"
```

```
[1] JPEG x WebP x PNG
```

```
try:
    mm.show(
        [
            mm.read("tmp/fig_05_formatos_comparacao_0.png"),
        ],
        titles=[
            'JPEG x WebP x PNG',
        ],
        cols=1,
    )
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + "
    tmp/fig_05_formatos_comparacao_0.png (ver a versao Python)")
```

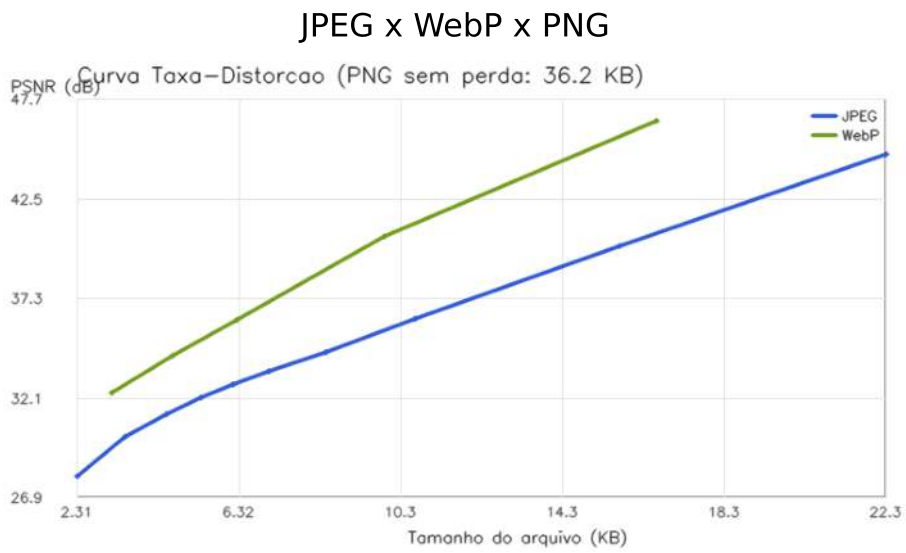


Figure 5.28: Curva taxa-distorção: PSNR vs tamanho de arquivo para JPEG, WebP e PNG aplicada à imagem do *Cameraman*.

i Taille originale de l'image

L'image *Cameraman* (256 × 256 pixels en niveaux de gris) occupe **64 Ko** en format brut (sans compression). À titre de référence, le PNG *lossless* compresse ce volume à **36,2 Ko** — mettant en évidence que la compression sans perte réduit déjà significativement le stockage pour les images comportant des régions homogènes. En revanche, les formats avec perte (JPEG et WebP) atteignent des tailles encore plus réduites : le JPEG avec une qualité de 95 occupe 22,3 Ko (PSNR 45 dB), tandis que le WebP avec une qualité de 90 atteint 12,5 Ko avec un PSNR équivalent, démontrant sa supériorité en matière d'efficacité de compression.

5.8.4.2 Cartographie Spatiale des Erreurs et Corrélation Perceptuelle

Bien que le PSNR offre un indicateur numérique rapide, les métriques globales ne parviennent pas à distinguer comment la perte d'informations se répartit géométriquement sur l'image. La Figure 5.29 résout cette limitation en associant les reconstructions à différentes qualités à leurs cartes d'erreur absolue respectives et au SSIM.

Les cartes résiduelles — obtenues par la différence absolue normalisée entre l'image originale et l'image compressée — révèlent la signature spatiale intrinsèque de chaque architecture de codage :

- **À des qualités élevées ($Q = 95$ à $Q = 75$) :** Les distorsions se concentrent principalement autour des transitions abruptes d'intensité (bords), résultant du repliement spectral dû à l'élimination des hautes fréquences. L'indice SSIM reste proche de l'unité, attestant de l'intégrité des structures originales.
- **À des qualités agressives ($Q = 50$ à $Q = 25$) :** L'erreur adopte une structure de maillage orthogonal régularisé. Ce motif géométrique met en évidence l'apparition des **artefacts de bloc** (*blocking artifacts*), indiquant que la quantification sévère a corrompu la corrélation spatiale entre les blocs adjacents de 8×8 pixels.

Le SSIM capture cette dégradation morphologique de manière beaucoup plus sensible que le PSNR, pénalisant le score final à mesure que l'organisation structurelle et les textures fines — auxquelles le système visuel humain est hautement réactif — sont éliminées par le codeur.

```
%%writefile tmp/fig_05_ssim_artefatos.cpp
#define MM_OUT "tmp/fig_05_ssim_artefatos.png"
//| label: fig-05-ssim-artefatos
```

```
//| fig-cap: "Analyse spatiale de la dégradation : images reconstruites et cartes d'erreur
absolue normalisées pour différents facteurs de qualité JPEG."
//| echo: true
//| output: true

#include <opencv2/opencv.hpp>
#include <vector>
#include <string>
#include <cmath>
#include "morph.hpp"

// Fonction SSIM (Wang et al., 2004) - version fenêtrée avec filtre gaussien
// Retourne la valeur moyenne SSIM et la carte SSIM complète.
static double compute_ssim_full(const cv::Mat& a, const cv::Mat& b, cv::Mat& ssim_map) {
    // Convertir en flottant 64 bits
    cv::Mat A, B;
    a.convertTo(A, CV_64F);
    b.convertTo(B, CV_64F);

    const double C1 = 6.5025;  // (0.01 * 255)^2
    const double C2 = 58.5225; // (0.03 * 255)^2

    // Filtre gaussien 11x11, sigma = 1.5
    cv::Mat mu1, mu2, mu1_sq, mu2_sq, mu1_mu2, sigma1_sq, sigma2_sq, sigma12;
    cv::GaussianBlur(A, mu1, cv::Size(11, 11), 1.5);
    cv::GaussianBlur(B, mu2, cv::Size(11, 11), 1.5);
    cv::multiply(mu1, mu1, mu1_sq);
    cv::multiply(mu2, mu2, mu2_sq);
    cv::multiply(mu1, mu2, mu1_mu2);

    cv::Mat A_sq, B_sq, A_B;
    cv::multiply(A, A, A_sq);
    cv::multiply(B, B, B_sq);
    cv::multiply(A, B, A_B);

    cv::Mat sigma1_sq_tmp, sigma2_sq_tmp, sigma12_tmp;
    cv::GaussianBlur(A_sq, sigma1_sq_tmp, cv::Size(11, 11), 1.5);
    cv::GaussianBlur(B_sq, sigma2_sq_tmp, cv::Size(11, 11), 1.5);
    cv::GaussianBlur(A_B, sigma12_tmp, cv::Size(11, 11), 1.5);

    cv::subtract(sigma1_sq_tmp, mu1_sq, sigma1_sq);
    cv::subtract(sigma2_sq_tmp, mu2_sq, sigma2_sq);
    cv::subtract(sigma12_tmp, mu1_mu2, sigma12);

    cv::Mat cs_map = (2 * sigma12 + C2) / (sigma1_sq + sigma2_sq + C2);
    cv::Mat ssim_map_tmp = ((2 * mu1_mu2 + C1) / (mu1_sq + mu2_sq + C1));
    cv::multiply(ssim_map_tmp, cs_map, ssim_map);

    // Valeur moyenne
    cv::Scalar mssim = cv::mean(ssim_map);
    return mssim[0];
}

int main() {
    // Cameraman (asset du chapitre), 256x256 - même image que la piste py.
    std::string img_path = "imagens/cameraman.png";
    cv::Mat img_full = cv::imread(img_path, cv::IMREAD_GRAYSCALE);
    if (img_full.empty()) {
        // Fallback : lire via mm::read si le chemin direct échoue
        mm::Image tmp = mm::read(img_path);
        img_full = cv::Mat(tmp.h, tmp.w, CV_8UC1, tmp.data.data());
```

```
    }
    cv::Mat img_resized;
    cv::resize(img_full, img_resized, cv::Size(256, 256));
    mm::Image img_gray = img_resized;

    std::vector<mm::Image> imgs_ssim;
    std::vector<std::string> titles_ssim;
    imgs_ssim.push_back(img_gray);
    titles_ssim.push_back("Original");

    for (int q : {25, 50, 75, 95}) {
        // DCT 8x8 -> quantisation -> IDCT
        mm::Image rec = mm::jpegCompress(img_gray, q);
        double psnr_v = mm::psnr(img_gray, rec);

        // Convertir en cv::Mat pour le calcul SSIM
        cv::Mat img_cv = img_gray;   // conversion implicite
        cv::Mat rec_cv = rec;        // conversion implicite

        cv::Mat ssim_map;
        double ssim_v = compute_ssim_full(img_cv, rec_cv, ssim_map);

        // Carte d'erreur absolue normalisée (0..255)
        cv::Mat img_f, rec_f;
        img_cv.convertTo(img_f, CV_64F);
        rec_cv.convertTo(rec_f, CV_64F);
        cv::Mat diff = cv::abs(img_f - rec_f);
        cv::Mat diff_norm;
        cv::normalize(diff, diff_norm, 0, 255, cv::NORM_MINMAX);
        cv::Mat diff_vis;
        diff_norm.convertTo(diff_vis, CV_8U);

        imgs_ssim.push_back(rec);
        imgs_ssim.push_back(mm::Image(diff_vis));
        titles_ssim.push_back("Q=" + std::to_string(q) + " (PSNR=" +
                              std::to_string(psnr_v).substr(0, 3) + "dB | SSIM=" +
                              std::to_string(ssim_v).substr(0, 4) + ")");
        titles_ssim.push_back("Carte d'erreur (Q=" + std::to_string(q) +
                              ") - bords et blocage");
    }

    mm::show(imgs_ssim, MM_OUT, titles_ssim, 3);
    return 0;
}
```

```
Overwriting tmp/fig_05_ssim_artefatos.cpp
```

```
!g++ -I. -std=c++17 -DMM_USE_OPENCV -I/usr/include/opencv4 tmp/fig_05_ssim_artefatos.cpp -o
tmp/fig_05_ssim_artefatos -lopencv_stitching -lopencv_alphamat -lopencv_aruco -lopencv_barcode
-lopencv_bgsegm -lopencv_bioinspired -lopencv_ccalib -lopencv_dnn_objdetect
-lopencv_dnn_superres -lopencv_dpm -lopencv_face -lopencv_freetype -lopencv_fuzzy -lopencv_hdf
-lopencv_hfs -lopencv_img_hash -lopencv_intensity_transform -lopencv_line_descriptor
-lopencv_mcc -lopencv_quality -lopencv_rapid -lopencv_reg -lopencv_rgbd -lopencv_saliency
-lopencv_shape -lopencv_stereo -lopencv_structured_light -lopencv_phase_unwrapping
-lopencv_superres -lopencv_optflow -lopencv_surface_matching -lopencv_tracking
-lopencv_highgui -lopencv_datasets -lopencv_text -lopencv_plot -lopencv_ml -lopencv_videostab
-lopencv_videoio -lopencv_viz -lopencv_wechat_qrcode -lopencv_ximgproc -lopencv_video
-lopencv_xobjdetect -lopencv_objdetect -lopencv_calib3d -lopencv_imgcodecs -lopencv_features2d
-lopencv_dnn -lopencv_flann -lopencv_xphoto -lopencv_photo -lopencv_imgproc -lopencv_core \
  && ./tmp/fig_05_ssim_artefatos \
```

```
  && test -f "tmp/fig_05_ssim_artefatos.png" \
  || echo "  mm::show não gravou tmp/fig_05_ssim_artefatos.png"
```

```
[1] Original
[2] Q=25 (PSNR=30.dB | SSIM=0.86)
[3] Carte d'erreur (Q=25) - bords et blocage
[4] Q=50 (PSNR=32.dB | SSIM=0.90)
[5] Carte d'erreur (Q=50) - bords et blocage
[6] Q=75 (PSNR=35.dB | SSIM=0.93)
[7] Carte d'erreur (Q=75) - bords et blocage
[8] Q=95 (PSNR=44.dB | SSIM=0.98)
[9] Carte d'erreur (Q=95) - bords et blocage
```

```
try:
    mm.show(mm.read("tmp/fig_05_ssim_artefatos.png"), figsize=(14, 14))
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + "
    tmp/fig_05_ssim_artefatos.png (ver a versao Python)")
```

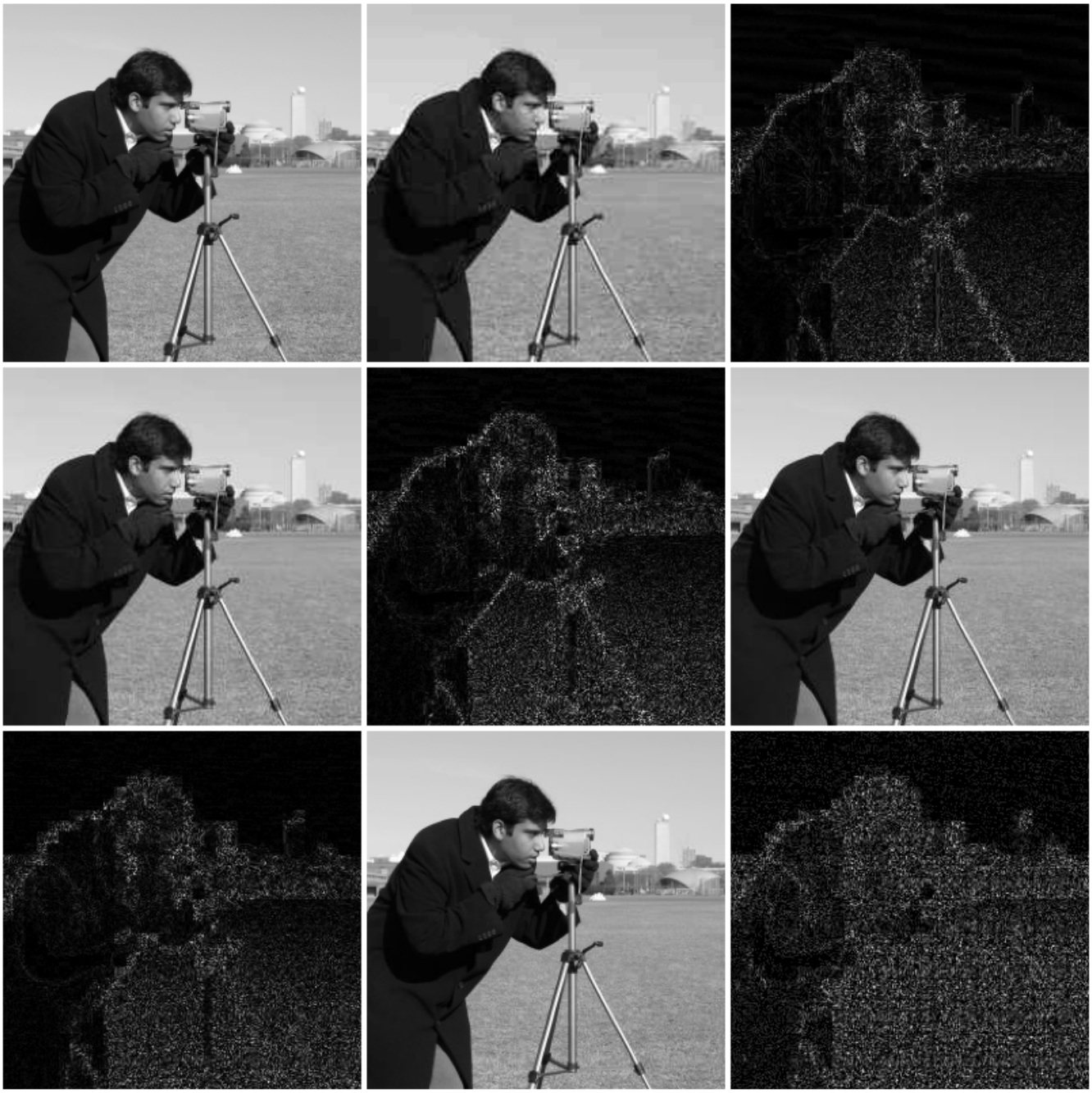


Figure 5.29: Análise espacial de degradação: imagens reconstruídas e respectivos mapas de erro absoluto normalizados para diferentes fatores de qualidade JPEG.

i Interprétation des cartes d'erreur

Les cartes d'erreur présentées ont été **normalisées individuellement** (`cv2.NORM_MINMAX`) afin de maximiser le contraste visuel et de révéler la structure spatiale des distorsions. Cela signifie que :

- Pour **Q=95**, l'erreur absolue est de l'ordre de **0,5 à 1,5 niveaux de gris** (imperceptible visuellement), mais la normalisation l'amplifie en noir et blanc pour mettre en évidence sa localisation au niveau des bords et des transitions.
- Pour **Q=25**, l'erreur absolue est **10 à 20 fois plus grande** (5 à 15 niveaux de gris), mais la normalisation l'amène également dans la même plage [0, 255].

Par conséquent, **l'intensité du blanc dans les cartes n'est PAS comparable entre différentes qualités** — les cartes servent uniquement à révéler la **signature spatiale** de l'erreur (bords vs blocs), et

non son ampleur. L'ampleur correcte est fournie par les valeurs PSNR et SSIM, qui montrent clairement que Q=95 présente une erreur bien inférieure à celle de Q=25.

Synthèse — Compression JPEG

Le processus de compression dans le standard JPEG repose sur l'application combinée de transformations spatiales, perceptuelles et statistiques pour réduire les redondances d'une image. La Table 5.9 résume le rôle de chaque étape dans le *pipeline* et son impact respectif sur la réduction des données.

Tableau 5.9: Synthèse des étapes du *pipeline* de compression JPEG et de leurs impacts respectifs.

Étape	Opération Analytique	Mécanisme de Gain / Compression
Conversion YC_bC_r	Isolation des canaux de luminance et de chrominance.	Modélise la perception du système visuel humain (SVH), permettant de traiter la couleur et la luminosité de manière indépendante.
Sous-échantillonnage 4:2:0	Réduction de la résolution spatiale des canaux de couleur (C_b et C_r).	Élimine environ 50 % des données brutes avec un impact visuel minimal.
DCT 8×8	Mappage du domaine spatial vers le domaine des fréquences spatiales.	Compactage de l'énergie, concentrant l'information essentielle dans les premiers coefficients.
Quantification Linéaire	Division entière des coefficients par une matrice de pondération $Q(u, v)$.	Principale source de compression avec perte ; élimine les hautes fréquences imperceptibles.
Codage Entropique	Application d'algorithmes RLE et de codage de Huffman.	Compression statistique sans perte, optimisée par les longues séquences de coefficients nuls.

Artefacts de dégradation caractéristiques

L'application de taux de compression excessivement agressifs (facteurs de qualité réduits) introduit des distorsions prévisibles dans l'image reconstruite, découlant des limitations mathématiques du modèle :

- **Artefacts de bloc (*blocking artifacts*) :** Discontinuités géométriques visibles aux frontières des blocs de 8×8 pixels, causées par la perte de corrélation spatiale après la quantification sévère des composantes AC.
- **Effet de sonnerie (*ringing*) :** Oscillations fantômes ou distorsions de « fumée » autour des bords nets et à fort contraste, provoquées par l'élimination brutale des harmoniques de haute fréquence nécessaires pour reconstruire les fonctions échelon.
- **Perte de texture fine :** Atténuation des détails à haute fréquence et à faible contraste (comme les pelouses, les tissus ou la porosité), donnant aux régions initialement texturées un aspect excessivement lisse ou homogénéisé.

5.9 Application Pratique : Débruitage par Filtrage Hybride

En réunissant les techniques consolidées tout au long de ce chapitre, on présente un *pipeline* complet de **restauration d'images** qui combine l'analyse spectrale dans le domaine fréquentiel avec le filtrage adaptatif dans le domaine spatial. L'objectif est d'atténuer un bruit mixte (composé d'une dégradation gaussienne et d'une interférence périodique) tout en préservant au maximum les détails structurels de l'image originale.

$$\text{Image Bruitée} \xrightarrow{\text{FFT2}} \xrightarrow{\text{Filtre Notch Gaussien}} \xrightarrow{\text{IFFT2}} \xrightarrow{\text{Filtre Bilatéral}} \text{Image Restaurée}$$

Évaluation Complémentaire : PSNR vs. SSIM

La paire de métriques statistiques PSNR et SSIM fournit une évaluation qualitative et morphologique complémentaire du processus de restauration :

- **PSNR :** Pénalise uniformément l'écart quadratique moyen pixel par pixel.
- **SSIM :** Évalue la préservation des structures locales perceptuellement pertinentes (luminance, contraste et contours).

En pratique, il existe un compromis analytique (*trade-off*) entre **réduction du bruit** et **préservation des détails** : des filtres spatiaux excessivement agressifs atténuent bien le bruit haute fréquence, mais dégradent les textures fines et lissent les contours nets — ce qui **réduit simultanément** à la fois le PSNR et le SSIM par rapport à l'image originale. Le défi de la conception de filtres est de trouver le point d'équilibre qui maximise les deux métriques, garantissant une restauration fidèle et visuellement agréable.

5.9.1 Analyse des performances et conclusion du chapitre

Les résultats numériques et visuels générés par Figure 5.30 démontrent la pertinence pratique d'associer différents domaines de traitement. L'insertion simultanée de bruit périodique et stochastique corrompt les propriétés morphologiques du signal, réduisant sévèrement les indices de similarité et le rapport signal-bruit de l'image de référence.

L'isolation et la suppression des pics harmoniques dans le domaine fréquentiel au moyen du masque *notch* éliminent les franges d'interférence sinusoïdales réparties sur l'espace bidimensionnel. Comme le montrent les données imprimées de Figure 5.30, ce filtrage chirurgical induit un saut immédiat et substantiel de la métrique PSNR. Toutefois, le bruit gaussien haute fréquence reste actif de manière homogène dans le spectre, exigeant une approche complémentaire.

La restauration finale est consolidée dans le domaine spatial avec l'introduction du filtre bilatéral. Contrairement aux opérateurs passe-bas classiques (tels que le filtre gaussien ou le filtre moyenneur), qui lisseraient indifféremment le bruit et les contours structurels, le filtrage bilatéral calcule des poids pondérés par la proximité géométrique et par la différence d'intensité radiométrique. Ce comportement adaptatif atténue les fluctuations stochastiques résiduelles dans les régions de transition douce et préserve la netteté des bords spatiaux.

La convergence des deux approches aboutit à une **amélioration substantielle et simultanée** du PSNR et du SSIM par rapport à l'image bruitée — bien que les valeurs finales demeurent inférieures à celles de l'image originale (PSNR $= \infty$, SSIM $= 1{,}0$), en raison de la perte inévitable d'informations spectrales et texturales lors des processus de filtrage. L'atténuation douce (gaussienne) des pics dans le spectre évite les artefacts de *ringing*, tandis que le filtre bilatéral élimine le bruit stochastique résiduel sans compromettre la netteté des bords. Les résultats prouvent l'efficacité et la complémentarité pratique des outils d'analyse fréquentielle présentés dans ce chapitre, démontrant que le filtrage hybride (fréquence + spatial) est supérieur à toute approche isolée pour la restauration d'images dégradées par un bruit mixte.

```
%%writefile tmp/fig_05_pipeline_denoising.cpp
#define MM_OUT "tmp/fig_05_pipeline_denoising.png"
#include <opencv2/opencv.hpp>
#include <cmath>
#include <random>
#include <vector>
#include <string>
#include <iostream>
#include "morph.hpp"
```

```
int main() {
    // Cameraman (asset do capítulo), 256x256 - mesma imagem da trilha py.
    cv::Mat cam = cv::imread("imagens/cameraman.png", cv::IMREAD_COLOR);
    cv::resize(cam, cam, cv::Size(256, 256));
    mm::Image img_gray = mm::gray(cam);
    int h_img = img_gray.h;
    int w_img = img_gray.w;

    //   1. Ruído misto: gaussiano + periódico
    std::mt19937 rng(42);
    std::normal_distribution<double> dist(0.0, 15.0);

    // u0, v0 frequências da interferência periódica
    int u0 = 15, v0 = 10;

    // X2, Y2 = malhas de coordenadas
    cv::Mat X2(h_img, w_img, CV_64F);
    cv::Mat Y2(h_img, w_img, CV_64F);
    for (int y = 0; y < h_img; ++y) {
        for (int x = 0; x < w_img; ++x) {
            X2.at<double>(y, x) = (double)x;
            Y2.at<double>(y, x) = (double)y;
        }
    }

    // constrói imagem ruidosa
    cv::Mat img_noisy_cv(h_img, w_img, CV_8U);
    for (int y = 0; y < h_img; ++y) {
        for (int x = 0; x < w_img; ++x) {
            double g = img_gray.data[y * w_img + x];
            double ruido_gauss = dist(rng);
            double ruido_period = 30.0 * std::sin(2.0 * M_PI *
                (u0 * X2.at<double>(y, x) / w_img + v0 * Y2.at<double>(y, x) / h_img));
            double v = g + ruido_gauss + ruido_period;
            v = std::max(0.0, std::min(255.0, v));
            img_noisy_cv.at<unsigned char>(y, x) = (unsigned char)std::lround(v);
        }
    }
    mm::Image img_noisy(img_noisy_cv);

    //   2. Espectro (log-magnitude) da imagem ruidosa
    mm::Image mag_n = mm::spectrumMag(img_noisy);

    //   3. Máscara notch gaussiana nos 4 picos periódicos
    auto suprimir_pico_gaussiano = [](cv::Mat& mask, int cy, int cx, double sigma) {
        int M = mask.rows, N = mask.cols;
        for (int yy = 0; yy < M; ++yy) {
            for (int xx = 0; xx < N; ++xx) {
                double notch = std::exp(-((double)((yy - cy) * (yy - cy)) +
                                          (double)((xx - cx) * (xx - cx))) /
                                        (2.0 * sigma * sigma));
                mask.at<double>(yy, xx) *= (1.0 - notch);
            }
        }
    };

    int cy0 = h_img / 2, cx0 = w_img / 2;
    cv::Mat mascara_notch(h_img, w_img, CV_64F, cv::Scalar(1.0));
    int offsets[4][2] = {{v0, u0}, {-v0, -u0}, {v0, -u0}, {-v0, u0}};
    for (auto& off : offsets) {
        suprimir_pico_gaussiano(mascara_notch, cy0 + off[0], cx0 + off[1], 3.0);
```

```
    }

    //   4. Filtragem: notch no domínio da frequência + bilateral
    mm::Image img_notch = mm::freqFilter(img_noisy, mascara_notch);
    cv::Mat img_notch_cv = img_notch;
    cv::Mat img_den_cv;
    cv::bilateralFilter(img_notch_cv, img_den_cv, 7, 25, 7);
    mm::Image img_den(img_den_cv);

    cv::Mat mascara_vis_cv;
    mascara_notch.convertTo(mascara_vis_cv, CV_8U, 255.0);
    mm::Image mascara_vis(mascara_vis_cv);

    double psnr_n  = mm::psnr(img_gray, img_noisy);
    double psnr_no = mm::psnr(img_gray, img_notch);
    double psnr_d  = mm::psnr(img_gray, img_den);
    std::cout << "Ruidosa: PSNR=" << psnr_n << " dB | Apos notch: " << psnr_no
              << " dB | Notch+bilateral: " << psnr_d << " dB" << std::endl;

    char buf[128];
    std::snprintf(buf, sizeof(buf), "Ruidosa (PSNR=%.1f dB)", psnr_n);
    std::string t1 = buf;
    std::snprintf(buf, sizeof(buf), "Apos notch (PSNR=%.1f dB)", psnr_no);
    std::string t2 = buf;
    std::snprintf(buf, sizeof(buf), "Notch + bilateral (PSNR=%.1f dB)", psnr_d);
    std::string t3 = buf;

    mm::show(
        {img_gray, img_noisy, mag_n, mascara_vis, img_notch, img_den},
        MM_OUT,
        {
            "Original",
            t1,
            "Espectro (picos visiveis)",
            "Mascara notch (gaussiana)",
            t2,
            t3,
        },
        6
    );

    return 0;
}
```

```
Overwriting tmp/fig_05_pipeline_denoising.cpp
```

```
!g++ -I. -std=c++17 -DMM_USE_OPENCV -I/usr/include/opencv4 tmp/fig_05_pipeline_denoising.cpp
-o tmp/fig_05_pipeline_denoising -lopencv_stitching -lopencv_alphamat -lopencv_aruco
-lopencv_barcode -lopencv_bgsegm -lopencv_bioinspired -lopencv_ccalib -lopencv_dnn_objdetect
-lopencv_dnn_superres -lopencv_dpm -lopencv_face -lopencv_freetype -lopencv_fuzzy -lopencv_hdf
-lopencv_hfs -lopencv_img_hash -lopencv_intensity_transform -lopencv_line_descriptor
-lopencv_mcc -lopencv_quality -lopencv_rapid -lopencv_reg -lopencv_rgbd -lopencv_saliency
-lopencv_shape -lopencv_stereo -lopencv_structured_light -lopencv_phase_unwrapping
-lopencv_superres -lopencv_optflow -lopencv_surface_matching -lopencv_tracking
-lopencv_highgui -lopencv_datasets -lopencv_text -lopencv_plot -lopencv_ml -lopencv_videostab
-lopencv_videoio -lopencv_viz -lopencv_wechat_qrcode -lopencv_ximgproc -lopencv_video
-lopencv_xobjdetect -lopencv_objdetect -lopencv_calib3d -lopencv_imgcodecs -lopencv_features2d
-lopencv_dnn -lopencv_flann -lopencv_xphoto -lopencv_photo -lopencv_imgproc -lopencv_core \
  && ./tmp/fig_05_pipeline_denoising \
  && test -f "tmp/fig_05_pipeline_denoising.png" \
```

```
  || echo "  mm::show não gravou tmp/fig_05_pipeline_denoising.png"
```

```
Ruidosa: PSNR=20.1968 dB | Apos notch: 22.3552 dB | Notch+bilateral: 23.5442 dB
[1] Original
[2] Ruidosa (PSNR=20.2 dB)
[3] Espectro (picos visiveis)
[4] Mascara notch (gaussiana)
[5] Apos notch (PSNR=22.4 dB)
[6] Notch + bilateral (PSNR=23.5 dB)
```

```
try:
    mm.show(mm.read("tmp/fig_05_pipeline_denoising.png"), figsize=(20, 4))
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + "
    tmp/fig_05_pipeline_denoising.png (ver a versao Python)")
```

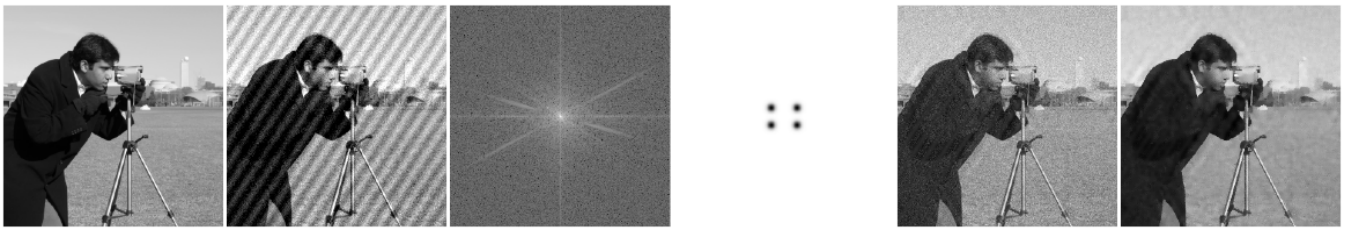


Figure 5.30: *Pipeline* completo de remoção de ruído misto: (1) adição de ruído gaussiano e periódico; (2) identificação de picos de interferência no espectro de frequências; (3) aplicação de máscara *notch* com atenuação gaussiana suave; (4) pós-processamento via filtro bilateral para eliminação do ruído estocástico residual.

5.10 Résumé du Chapitre

La transition du **domaine spatial** vers le **domaine fréquentiel** révèle la distribution spectrale de l'énergie de l'image, établissant ainsi la base analytique pour le filtrage avancé, la restauration et la compression des données. L'articulation structurelle de ces concepts est synthétisée dans la carte conceptuelle de la Figure 5.31.

Fundamentos Essenciais

- **TFD et Perception Visuelle :** Le spectre décompose l'image en composantes harmoniques. La **phase** conserve l'intelligibilité géométrique de la scène et la localisation des contours, tandis que la **magnitude** détermine la distribution du contraste et les amplitudes globales.
- **Efficacité Algorithmique :** Le Théorème de Convolution permet le traitement de masques à grande échelle dans le domaine fréquentiel via la FFT, réduisant la complexité computationnelle asymptotique de $O(N^2K^2)$ dans l'espace à $O(N^2 \log N)$.
- **Phénomène de *Ringing* :** Les coupures abruptes dans le spectre (filtres idéaux) génèrent des oscillations spatiales indésirables (phénomène de Gibbs). L'atténuation douce par des filtres de **Butterworth** ou **gaussiens** élimine ces discontinuités.
- **Analyse Multirésolution via *Wavelets* :** Dépassant le caractère purement global de Fourier, la DWT capture simultanément la fréquence et la localisation spatiale, fondant le standard JPEG 2000 et appuyant des représentations hiérarchiques analogues aux extractions de caractéristiques dans les Réseaux de Neurones Convolutifs (CNN).
- **Compression Perceptuelle (DCT) :** Le *pipeline* JPEG exploite les limitations de contraste du système visuel humain aux hautes fréquences spatiales. La DCT isole l'énergie de blocs 8×8, permettant à la quantification d'éliminer les coefficients AC des détails fins sans préjudice perceptuel sévère.

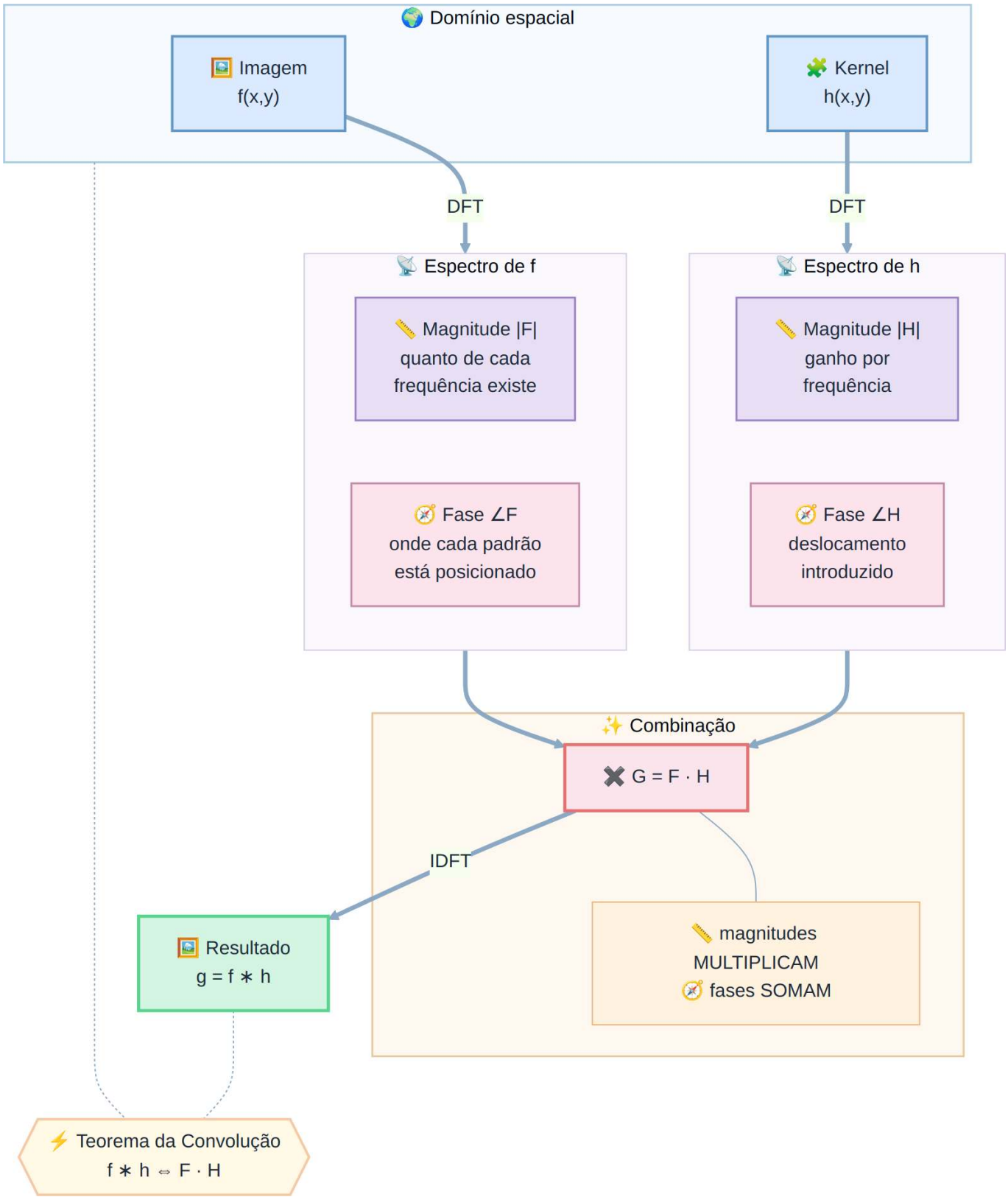


Figure 5.31: Carte conceptuelle des transformations et propriétés dans le domaine fréquentiel.

Prochaines Étapes : Le **Chapitre 6** inaugure la Partie II de l'ouvrage, appliquant les outils de traitement d'images à la résolution de problèmes réels d'inspection industrielle. Seront explorées des techniques de **segmentation et d'analyse de formes** pour la détection automatique de défauts sur les lignes de production — depuis l'identification de défauts superficiels sur les pièces jusqu'à la lecture *QRCode* sur les épreuves, consolidant le pont entre la théorie présentée dans la Partie I et les exigences pratiques de la vision par ordinateur.

5.11 🤖 Utilisation de Gemini Notebook comme Tuteur Complémentaire

Dans cette édition, l'utilisation de la plateforme **Gemini Notebook** est encouragée comme outil d'apprentissage complémentaire — **et non comme substitut** de la lecture attentive, de la résolution d'exercices ou de l'expérimentation pratique. Fondé sur des architectures d'intelligence artificielle, le système utilise exclusivement le matériel pédagogique et les documents fournis par l'auteur comme base de connaissances, garantissant que les réponses générées soient conceptuellement alignées sur le contenu programmatique et l'approche pédagogique adoptée tout au long de cet ouvrage.

Accès au Tuteur Intelligent

🚀 ACCÉDER À Gemini Notebook : CHAPITRE 05

🌐 Langue et Langage de Programmation

Le projet de ce chapitre dans Gemini Notebook a été construit uniquement avec le texte en **portugais** et les exemples de code en **Python**. Si vous étudiez à partir de l'édition en anglais ou en français, ou si vous suivez le parcours en C++, les réponses du tuteur peuvent ne pas correspondre exactement à la version que vous lisez.

Directives concernant le Contenu Généré par Intelligence Artificielle

Bien que les outils d'intelligence artificielle constituent des alliés efficaces dans le processus d'apprentissage et de révision, le contenu généré est sujet à des incohérences ou à des imprécisions techniques. Par conséquent, la consultation systématique de manuels, d'articles scientifiques et de sources académiques indexées est indispensable pour une validation rigoureuse des informations. Il est vivement recommandé d'exécuter et de modifier les exemples pratiques en Python fournis dans ce chapitre comme méthode principale de vérification expérimentale des résultats.

5.12 Liste d'exercices

1. **(10 %) Implémentation directe de la TFD 2D :** Implémentez analytiquement la Transformée de Fourier Discrète 2D (TFD) sans l'aide de fonctions natives de bibliothèques (comme `np.fft.fft2`), en utilisant strictement la formulation mathématique définie dans la Équation 5.1 pour une matrice de dimensions 16×16. Effectuez la validation numérique en comparant les coefficients générés avec les résultats de la fonction `np.fft.fft2`, en vous assurant que l'écart absolu maximal soit inférieur à 10^{-8}. Mesurez les temps d'exécution des deux méthodes et présentez une justification théorique pour la disparité observée en termes de complexité asymptotique.

2. **(15 %) Suppression du bruit périodique :** Ajoutez des interférences sinusoïdales avec des fréquences spatiales $(u_0, v_0) \in \{(5, 10), (20, 5), (30, 30)\}$ à l'image de test du *Cameraman*. Pour chaque scénario de dégradation, concevez un masque de filtrage *notch* spécifique dans le domaine fréquentiel afin d'isoler et d'atténuer les pics harmoniques indésirables. Évaluez quantitativement l'efficacité du processus de restauration par le calcul des métriques PSNR et SSIM. Discutez analytiquement du compromis entre l'atténuation du bruit sinusoïdal et l'atténuation indésirable des caractéristiques structurelles légitimes de l'image.

3. **(15 %) Analyse comparative des opérateurs passe-bas :** Réalisez une étude comparative entre les filtres passe-bas idéal, gaussien et de Butterworth (avec des ordres harmoniques $n = 1, 2, 4$), paramétrés avec des fréquences de coupure $D_0 = 20, 40, 60$ pixels. Pour chaque combinaison structurelle, calculez les indices PSNR et SSIM de l'image résultante par rapport au signal original de référence. Organisez les données quantitatives dans un tableau structuré et tracez les graphiques unidimensionnels des fonctions de transfert correspondantes le long du profil horizontal $H(u, 0)$.

4. **(15 %) Banque de filtres multirésolution de Haar :** Développez un script pour exécuter manuellement la décomposition *wavelet* discrète 2D de premier niveau en utilisant la famille de Haar. L'algorithme doit calculer les coefficients des filtres correspondants passe-bas (h) et passe-haut (g), en les appliquant de manière séparable sur les lignes et les colonnes de la matrice, suivis de l'opération de décimation (sous-échantillonnage spatial par un facteur de 2). Validez numériquement l'exactitude de votre implémentation en confrontant les sous-bandes obtenues avec la sortie de la fonction `pywt.dwt2(img, 'haar')`.

5. **(15 %) Compression éparse par seuillage wavelet :** Appliquez la technique de filtrage par seuillage abrupt (*hard thresholding*) sur les coefficients de détail de la décomposition *wavelet*, en adoptant les seuils numériques $T \in \{5, 10, 20, 40, 80\}$ pour les familles de Haar, Daubechies (`db4`) et Symlets (`sym4`). Après avoir réalisé le processus de synthèse au moyen de la transformée inverse (`pywt.waverec2`), calculez les valeurs de PSNR et SSIM de chaque image reconstruite. Identifiez et justifiez quelle combinaison de famille *wavelet* et de seuil T maximise la similarité structurelle.

6. **(15 %) Construction d'un encodeur JPEG simplifié :** Implémentez le pipeline complet de compression de données simulant la norme JPEG. Le flux doit englober : la conversion spatiale $RGB \to YC_bC_r$, le sous-échantillonnage chromatique dans la proportion 4:2:0, la segmentation de la luminance en blocs disjoints de 8×8 pixels, l'application de la DCT-II 2D orthogonale, et la quantification linéaire basée sur la matrice normalisée de luminance mise à l'échelle par des facteurs de qualité souhaités. Réalisez le décodage inverse et comparez quantitativement les reconstructions avec les fichiers générés par la fonction `cv2.imencode` pour les facteurs de qualité de 20, 50 et 80.

7. **(15 %) Analyse perceptuelle sur des contenus hétérogènes :** Développez une image synthétique composée de trois régions distinctes et aux caractéristiques spectrales contrastées : une texture photographique complexe (représentant de hautes fréquences stochastiques), une zone de texte vectorisé avec des bords nets (représentant des transitions en échelon pures) et un gradient linéaire continu (représentant de basses fréquences homogènes). Soumettez cette image mixte aux processus de compression sous les formats JPEG, PNG et WebP. Évaluez et interprétez les résultats en corrélant la taille finale du fichier sur disque avec les métriques PSNR et SSIM obtenues, en justifiant quel format présente les meilleures performances pour des signaux de nature hétérogène et pourquoi cet avantage survient en termes de compactage d'énergie et de préservation perceptuelle.

Références du chapitre

Le fondement théorique et le développement analytique des concepts abordés dans ce chapitre s'appuient sur les ouvrages de référence suivants :

- **Gonzalez (2018)** — Formulations classiques des Transformées de Fourier Discrètes 2D (TFD), conception de filtres analytiques dans le domaine fréquentiel, Transformée en Cosinus Discrète (TCD) et principes fondamentaux des systèmes de compression d'images.
- **Oppenheim (2010)** — Théorie formelle des signaux et des systèmes appliqués dans le domaine discret, couvrant les propriétés mathématiques de la TFD et la modélisation analytique du Théorème de Convolution.
- **Mallat (1999)** — Fondements mathématiques de la théorie des *ondelettes*, formalisation de l'analyse multirésolution (AMR) et architecture des bancs de filtres dyadiques.
- **Wallace (1991)** — Spécification originale et aspects techniques du standard de compression ISO/CEI JPEG, avec un accent sur les critères psychovisuels pour la conception des matrices de quantification TCD.

- **Szeliski (2022)** — Modélisation computationnelle et caractérisation des métriques modernes de fidélité et de qualité perceptuelle (PSNR et SSIM), ainsi que l'analyse comparative des formats d'images tramées à hautes performances.

5.13 💻 Partie pratique avec exercices de programmation

La présente liste d'exercices de programmation (EP) consolide les formulations théoriques présentées tout au long du Chapitre 5 — Transformées et Compression — au moyen d'un parcours pratique appliqué. Les exercices sont structurés à partir de matrices de dimensions réduites, permettant la validation analytique et l'inspection manuelle de chaque coefficient, tout en maintenant la cohérence méthodologique adoptée dans les chapitres précédents.

L'enchaînement des exercices reproduit rigoureusement le flux conceptuel du chapitre : on commence par l'implémentation explicite de la Transformée de Fourier Discrète (TFD) à partir de sa définition mathématique fondamentale ; on progresse vers la conception de filtres passe-bas et de masques *notch* dans le domaine fréquentiel ; on applique la quantification des coefficients (noyau de la compression avec perte) ; et l'on conclut par l'intégration de ces étapes dans la construction d'un *pipeline* de compression JPEG simplifié et dans l'analyse perceptuelle des formats d'image.

> ❗ Directives pour la résolution des exercices de programmation
>
> Dans tous les exercices de ce chapitre, les coordonnées du **centre du spectre** (origine des fréquences spatiales après application du décalage `fftshift`) doivent être déterminées par division entière. Pour une matrice de L lignes et C colonnes, la composante de fréquence nulle se situe à la position :
>
> $$(c_y, c_x) = \left(\left\lfloor \frac{L}{2} \right\rfloor, \left\lfloor \frac{C}{2} \right\rfloor\right)$$
>
> Cette convention est rigoureusement identique à celle adoptée par la fonction `np.fft.fftshift`. De plus, à toutes les étapes exigeant une discrétisation ou un arrondi numérique (que ce soit dans la quantification des coefficients AC ou dans la reconstruction finale des pixels), on doit employer l'arrondi standard au plus proche entier (*round half away from zero*), atténuant les ambiguïtés sur les valeurs dont la fraction est exactement égale à 0.5.

🎯 Objectif de ce cahier

Ce cahier permet de développer, valider, organiser et tester des solutions d'**Exercices de Programmation (EPs)** dans des environnements interactifs, comme Colab, avec les mêmes cas de test que Moodle, en les y copiant uniquement au moment d'enregistrer la note officielle.

Téléchargement

Téléchargez `morph.py` et `testsuite.py` en exécutant la cellule ci-dessous :

```
import os, urllib.request

os.makedirs("tmp/state", exist_ok=True)

url = "https://raw.githubusercontent.com/fzampirolli/pdi-vc/master/morph/config.py"
if not os.path.exists("config.py"):
    urllib.request.urlretrieve(url, "config.py")

import config
```

```
config.setup(testsuite=True, cpp=True)
from morph import mm
from testsuite import TestSuite
```

```
Environnement prêt. Morph : 1.1.9 | OpenCV : 5.0.0 | TestSuite : 1.1.2
```

Exécution des tests

Pour évaluer les tests, exécutez `TestSuite("EP05_01.extensão").run()` dans une nouvelle cellule, en remplaçant l'extension par celle du langage utilisé (`.py`, `.java`, `.c`, `.cpp`, `.js` ou `.r`). Le système télécharge les cas de test depuis GitHub, exécute le programme et calcule automatiquement la note.

Pour tester directement du code Python, sans sauvegarder de fichier, utilisez `run_code(codigo)` en passant le code sous forme de *chaîne de caractères* dans une variable `codigo` :

```
codigo = """
from morph import mm
# ... votre code ici ...
"""
TestSuite("EP05_01").run_code(codigo)
```

5.13.1 EP05_01 🟢 Filtre Passe-Bas Idéal par Distance dans le Spectre

Dans un ***scanner*** **de documents anciens**, le capteur capte le papier froissé et la texture des fibres en même temps que le texte — un bruit haute fréquence qui « pollue » le spectre sur les bords. Le technicien de maintenance n'a pas accès à l'image originale, seulement au **spectre de magnitude déjà calculé** par le logiciel du *scanner*. Son travail est simple et chirurgical : ne conserver que le **cercle central** des basses fréquences (la structure globale du document) et effacer tout ce qui se trouve hors du rayon D_0, éliminant la texture fine sans même avoir à toucher à l'image spatiale.

C'est le **filtre passe-bas idéal (LPFI)** : l'opération spectrale la plus directe du chapitre, mais aussi celle qui révèle le mieux l'anatomie d'un spectre centré.

5.13.1.1 📋 Directives d'implémentation

1. **Dimensions :** Lire les entiers L (lignes) et C (colonnes) du spectre de magnitude — déjà fourni **centré** (équivalent à la sortie de `np.fft.fftshift`).
2. **Fréquence de coupure :** Lire l'entier D_0.
3. **Données :** Lire les valeurs entières de la matrice de magnitude, ligne par ligne.
4. **Centre du spectre :** Calculer $(c_y, c_x) = (L \,//\, 2,\ C \,//\, 2)$.
5. **Distance :** Pour chaque position (u, v), calculer

$$D(u,v) = \sqrt{(u - c_y)^2 + (v - c_x)^2}$$

6. **Masque idéal :** Appliquer

$$H(u,v) = \begin{cases} 1, & D(u,v) \le D_0 \\ 0, & D(u,v) > D_0 \end{cases}$$

7. **Filtrage :** La valeur de sortie est $\text{mag}'(u,v) = \text{mag}(u,v) \cdot H(u,v)$.
8. **Sortie :** Afficher la matrice filtrée avec les dimensions $L \times C$.

5.13.1.2 📌 Contraintes computationnelles

- **Comparaison non stricte :** le critère utilise $D(u,v) \le D_0$ (la frontière appartient au filtre, c'est-à-dire qu'elle est conservée).
- **Type :** toutes les valeurs d'entrée et de sortie sont des entiers ; la distance est calculée en virgule flottante uniquement en interne.

- **Pas d'arrondi de magnitude :** comme l'entrée est déjà entière et que le masque est binaire (0 ou 1), la sortie n'a jamais besoin d'arrondi.

5.13.1.3 🧠 Fondement théorique

Région	Distance au centre	Effet du filtre
Centre ($D \leq D_0$)	Basses fréquences	Préservées — structure globale conservée
Bords ($D > D_0$)	Hautes fréquences	Mises à zéro — texture et bruit supprimés
D_0 **petit**	—	L'image reconstruite serait très floue
D_0 **grand**	—	Peu de filtrage ; presque toute l'énergie est préservée

5.13.1.4 📦 Spécification d'entrée et de sortie (VPL)

Entrée :

- Ligne 1 : Entier L.
- Ligne 2 : Entier C.
- Ligne 3 : Entier D_0.
- Lignes suivantes : Éléments entiers de la matrice de magnitude (centrée).

Sortie :

- Matrice filtrée en L lignes et C colonnes, séparés par des espaces.

5.13.1.5 📌 Exemples

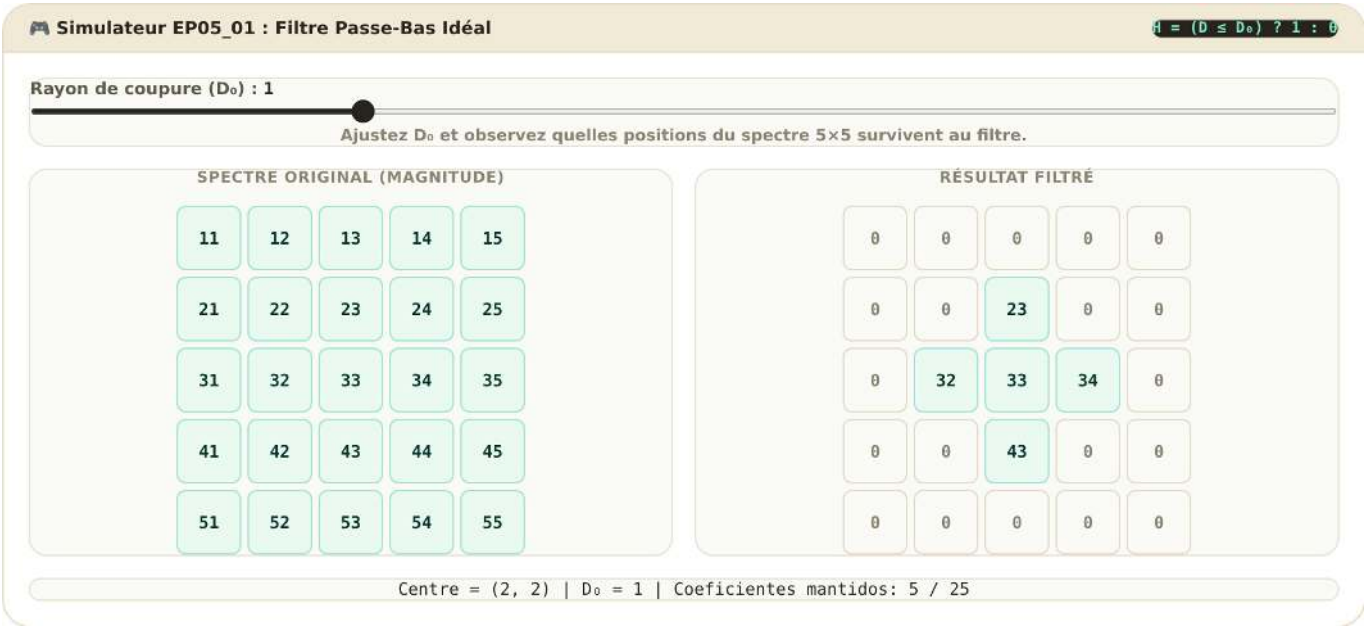
Entrée	Sortie	Observation
3 3 1 10 20 30 40 50 60 70 80 90	0 20 0 40 50 60 0 80 0	Centre $(1, 1)$. Les coins ont $D = \sqrt{2} \approx 1.41 > 1$, donc ils sont mis à zéro ; les voisins orthogonaux ont $D = 1 \leq 1$ et sont conservés.
1 3 0 5 9 7	0 9 0	$L = 1, C = 3$: centre en $(0, 1)$. Seule la position centrale elle-même ($D = 0$) survit à $D_0 = 0$.

```
%%writefile EP05_01.cpp
// your solution
```

```
Overwriting EP05_01.cpp
```

```
TestSuite("EP05_01.cpp").run()
```

```
<IPython.core.display.HTML object>
```



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.fr/cap05/sim-ep0501.html

Figure 5.32: Simulateur EP05_01 : Filtre passe-bas idéal dans le spectre

5.13.2 EP05_02 🟡 Filtre *Notch* : Suppression des pics périodiques

Une caméra d'**inspection industrielle** capture des images de circuits imprimés, mais l'alimentation électrique de la ligne de production introduit une **interférence électrique périodique** — un motif de stries quasi imperceptible à l'œil nu, mais qui apparaît dans le spectre de Fourier comme des **paires de pics brillants** symétriquement positionnés autour du centre. L'équipe de vision par ordinateur ne peut pas recapturer l'image : elle doit **localiser et effacer chirurgicalement** ces paires de pics dans le spectre, tout en préservant le reste de l'information utile de l'image.

C'est le rôle du **filtre coupe-bande *notch*** : contrairement au passe-bas (qui affecte une région continue), il cible **des points spécifiques et leurs symétriques**, laissant le reste du spectre intact.

5.13.2.1 📋 Directives d'implémentation

1. **Dimensions :** Lire les entiers L (lignes) et C (colonnes) du spectre de magnitude centré.
2. **Données :** Lire les valeurs entières de la matrice de magnitude, ligne par ligne.
3. **Pics :** Lire l'entier K (nombre de paires de pics à supprimer).
4. **Pour chacun des K pics :** lire trois entiers Δv, Δu, r — déplacement vertical, déplacement horizontal et rayon du *notch*.
5. **Centre du spectre :** $(c_y, c_x) = (L \,//\, 2,\ C \,//\, 2)$.
6. **Suppression symétrique :** pour chaque pic, mettre à zéro **toutes** les positions (u, v) telles que la distance au point $(c_y + \Delta v,\ c_x + \Delta u)$ soit $\leq r$, **et également** toutes les positions à distance $\leq r$ du point symétrique $(c_y - \Delta v,\ c_x - \Delta u)$.
7. **Sortie :** Afficher la matrice résultante avec les dimensions $L \times C$.

5.13.2.2 📌 Contraintes computationnelles

- **Symétrie obligatoire :** chaque pic fourni génère **deux** disques mis à zéro (le point et son symétrique par rapport au centre) — oublier le symétrique est l'erreur la plus courante.
- **Chevauchement :** si deux disques se chevauchent, la position reste à zéro (ni « addition » ni restauration).
- **Comparaison non stricte :** une position est mise à zéro si distance $\leq r$.
- **Ordre de lecture :** les K pics doivent être traités dans l'ordre où ils apparaissent en entrée, mais le résultat final est indépendant de l'ordre (les opérations de mise à zéro sont commutatives).

5.13.2.3 🧠 Fondement théorique

Concept	Rôle dans le filtre *notch*
Pic en $(\Delta v, \Delta u)$	Fréquence de l'interférence périodique détectée visuellement dans le spectre
Point symétrique $(-\Delta v, -\Delta u)$	Toute DFT d'un signal réel est hermitienne : les pics apparaissent toujours en paires symétriques par rapport au centre
Rayon r	Contrôle la « largeur » de la réjection — un r grand élimine davantage d'énergie autour du pic, mais aussi davantage d'information utile

5.13.2.4 📦 Spécification d'entrée et de sortie (VPL)

Entrée :

- Ligne 1 : Entier L.
- Ligne 2 : Entier C.
- Lignes suivantes : Éléments entiers de la matrice de magnitude (centrée), L lignes.
- Ligne suivante : Entier K.
- K lignes suivantes : trois entiers Δv, Δu, r (séparés par des espaces).

Sortie :

- Matrice résultante sur L lignes et C colonnes, séparés par des espaces.

5.13.2.5 📌 Exemples

Entrée	Sortie	Observation
5 5 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 1 1 1 0	1 2 3 4 5 6 0 8 9 10 11 12 13 14 15 16 17 18 0 20 21 22 23 24 25	Centre $(c_y, c_x) = (2, 2)$. Pic fourni $(\Delta v, \Delta u) = (1, 1)$ génère le point $(3, 3)$ (valeur 19) et son symétrique $(1, 1)$ (valeur 7), tous deux mis à zéro avec $r = 0$ (seuls les points exacts).

```
%%writefile EP05_02.cpp
// your solution
```

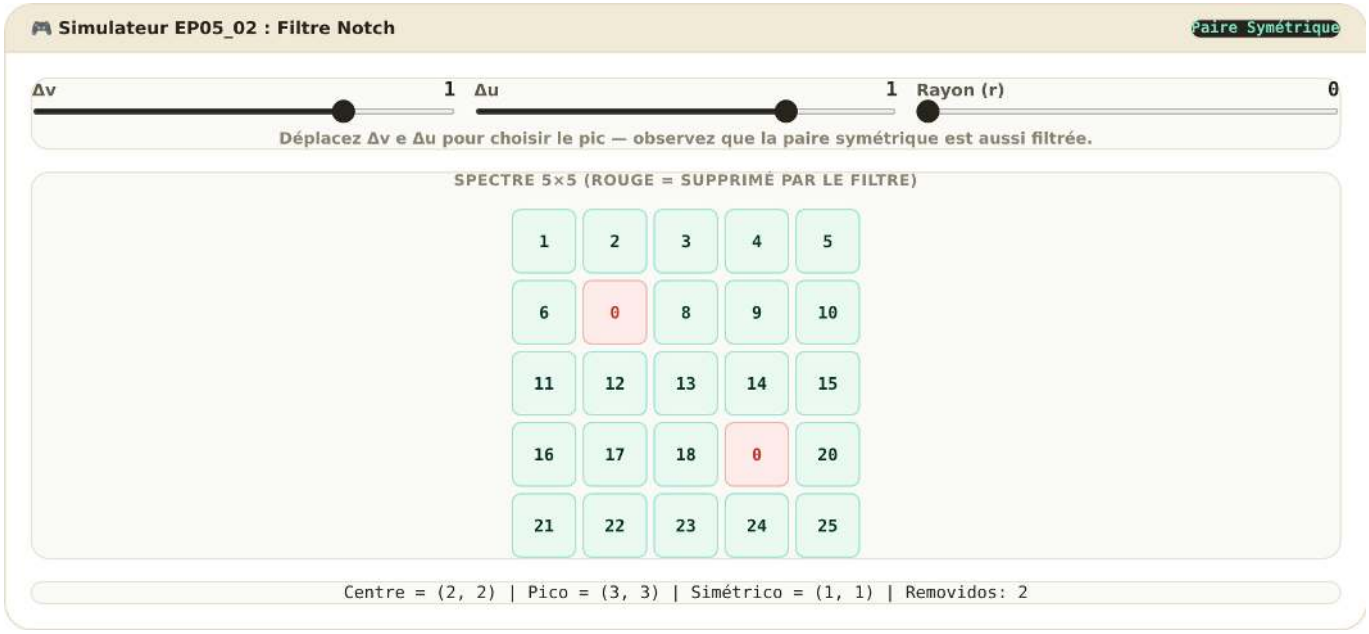
```
Overwriting EP05_02.cpp
```

```
TestSuite("EP05_02.cpp").run()
```

```
<IPython.core.display.HTML object>
```

5.13.3 EP05_03 🟠 Quantification DCT : la véritable source de compression

Une application de **galerie de photos** doit réduire la taille de milliers d'images avant de les *téléverser* vers le cloud, sans tout recoder de zéro. L'ingénieur responsable dispose déjà des **coefficients DCT** de chaque



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.fr/cap05/sim-ep0502.html

Figure 5.33: Simulateur EP05_02: Filtre Notch

bloc 4×4 calculés (l'étape coûteuse en calcul est déjà faite) — il ne reste plus qu'à appliquer la **table de quantification**, l'étape qui élimine réellement de l'information et génère la compression. Les coefficients haute fréquence, moins perceptibles à l'œil humain, reçoivent de grands diviseurs et tendent à devenir **zéro** ; les coefficients basse fréquence, plus perceptibles, reçoivent de petits diviseurs et survivent presque intacts.

Vous allez implémenter exactement cette étape : **quantifier et déquantifier** (diviser, arrondir, multiplier en retour) — le cœur de la compression *lossy* du JPEG.

5.13.3.1 📋 Directives d'implémentation

1. **Dimension du bloc :** Lire l'entier N (bloc $N \times N$).
2. **Coefficients :** Lire la matrice C des coefficients DCT, N lignes avec N entiers chacune (ils peuvent être négatifs).
3. **Table de quantification :** Lire la matrice Q, N lignes avec N entiers positifs chacune.
4. **Quantification :** Pour chaque position (u, v), calculer l'indice quantifié

$$\tilde{C}(u,v) = \text{round}\left(\frac{C(u,v)}{Q(u,v)}\right)$$

   en utilisant l'arrondi standard à l'entier le plus proche (les valeurs intermédiaires `.5` ne se produisent jamais dans les cas de test).
5. **Déquantification (reconstruction) :** Calculer

$$C'(u,v) = \tilde{C}(u,v) \times Q(u,v)$$

6. **Sortie :** Afficher la matrice reconstruite C', $N \times N$, entiers.

5.13.3.2 📌 Contraintes informatiques

- ***Aller-retour* complet :** la sortie est le coefficient **reconstruit** ($\tilde{C} \times Q$), pas l'indice quantifié isolé.
- **Division en virgule flottante :** la division $C(u,v)/Q(u,v)$ doit être effectuée en virgule flottante avant l'arrondi — une division entière tronquée produirait un résultat incorrect.
- **Signe préservé :** les coefficients négatifs conservent leur signe après quantification et reconstruction.
- $Q(u,v) > 0$ **toujours :** aucune gestion de division par zéro n'est nécessaire.

5.13.3.3 🧠 Fondements théoriques

Coefficient	Fréquence	Valeur typique de Q	Effet de la quantification
$C(0,0)$	DC (moyenne du bloc)	Petite	Survit presque toujours — domine l'énergie
$C(u,v)$ faible $u+v$	Basse fréquence	Petite/moyenne	Partiellement préservé
$C(u,v)$ élevé $u+v$	Haute fréquence	Grande	Devient souvent zéro — source de la compression

5.13.3.4 📦 Spécification d'entrée et de sortie (VPL)

Entrée :

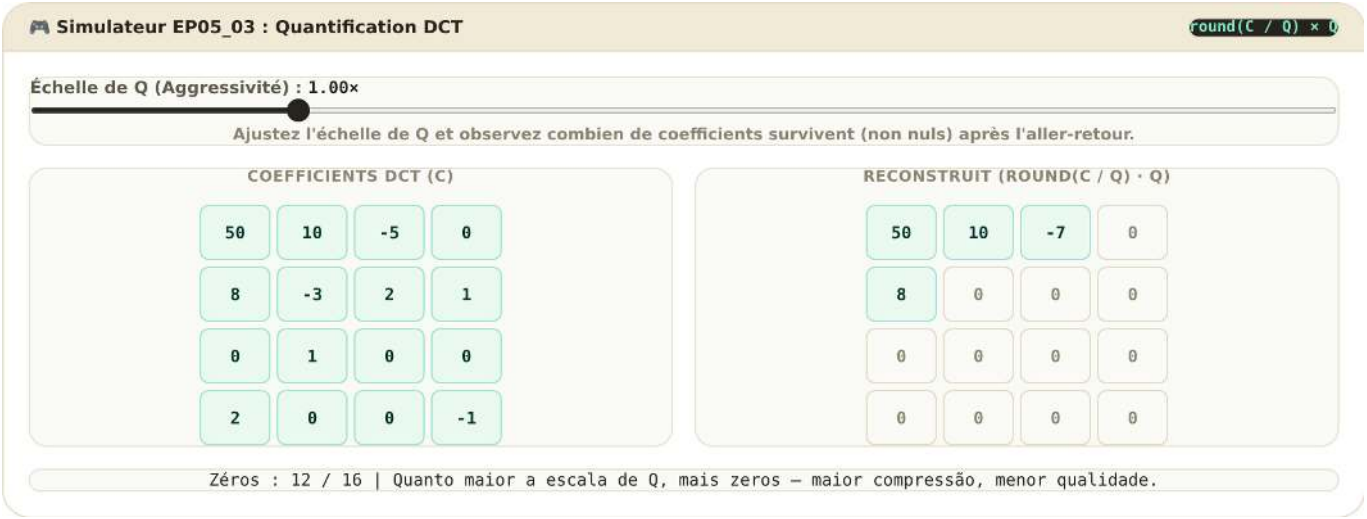
- Ligne 1 : Entier N.
- N lignes suivantes : matrice C (coefficients DCT, entiers, peuvent être négatifs).
- N lignes suivantes : matrice Q (table de quantification, entiers positifs).

Sortie :

- Matrice reconstruite C', $N \times N$, entiers séparés par des espaces.

5.13.3.5 📌 Exemples

Entrée	Sortie	Observation
4 50 10 -5 0 8 -3 2 1 0 1 0 0 2 0 0 -1 2 5 7 8 4 7 8 11 6 8 11 12 9 11 12 14	50 10 -7 0 8 0 0 0 0 0 0 0 0 0 0 0	$C(0,0) = 50/2 = 25 \to 25{\times}2 = 50$ (préservé). $C(0,2) = -5/7 \approx -0.71 \to -1 \to -1 \times 7 = -7$. Quant à $C(1,1) = -3/7 \approx -0.43 \to 0$: mis à zéro par la quantification — la majeure partie du bloc devient zéro, illustrant la compaction de l'énergie dans le coin supérieur gauche.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.fr/cap05/sim-ep0503.html

Figure 5.34: Simulateur EP05_03 : Quantification DCT (*round-trip*)

```
%%writefile EP05_03.cpp
// your solution
```

```
Overwriting EP05_03.cpp
```

```
TestSuite("EP05_03.cpp").run()
```

```
<IPython.core.display.HTML object>
```

5.13.4 EP05_04 🔴 Implémentation de la TFD 2D à partir de la définition

Un laboratoire de recherche en **astronomie computationnelle** a reçu, d'une mission ancienne, un petit capteur expérimental dont les données brutes ne peuvent pas être traitées par les bibliothèques modernes de FFT — l'environnement de validation est isolé et ne permet que des opérations arithmétiques de base. L'équipe doit **réimplémenter la Transformée de Fourier Discrète 2D à partir de la définition mathématique elle-même**, cellule par cellule, pour ensuite comparer bit à bit avec `np.fft.fft2` dans un autre environnement.

C'est l'exercice le plus conceptuel de la liste : il n'y a pas de raccourcis. Vous allez implémenter la double sommation de la Équation 5.1 directement, en mettant en évidence *pourquoi* la FFT existe — et le coût computationnel qu'elle évite.

5.13.4.1 📋 Directives d'implémentation

1. **Dimensions :** Lire les entiers M (lignes) et N (colonnes) de l'image $f(x, y)$.
2. **Données :** Lire les valeurs entières de $f(x, y)$, ligne par ligne.
3. **TFD 2D :** Pour chaque paire de fréquences (u, v) avec $u = 0, \ldots, M-1$ et $v = 0, \ldots, N-1$, calculer

$$F(u,v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y)\, e^{-j2\pi\left(\frac{ux}{M} + \frac{vy}{N}\right)}$$

   en utilisant l'identité d'Euler $e^{-j\theta} = \cos(\theta) - j\sin(\theta)$ pour séparer les parties réelle et imaginaire — **n'utilisez aucune fonction de FFT prête à l'emploi**.
4. **Magnitude :** Calculer $|F(u,v)| = \sqrt{\mathrm{Re}(F)^2 + \mathrm{Im}(F)^2}$ et arrondir à l'entier le plus proche.
5. **Sortie :** Afficher la matrice des magnitudes arrondies, $M \times N$, dans le même ordre (sans `fftshift` — la composante DC reste en $(0, 0)$).

5.13.4.2 📌 Contraintes computationnelles

- **Interdiction d'utiliser des bibliothèques de FFT :** l'implémentation doit calculer les double sommations explicitement (boucles imbriquées), même si c'est plus lent.
- **Sans `fftshift` :** la sortie conserve la convention brute de la TFD, avec la composante DC en $F(0, 0)$ (coin supérieur gauche).
- **Arrondi :** la magnitude finale doit être arrondie à l'entier le plus proche ; dans les cas de test, il n'y a pas d'ambiguïté `.5`.
- **Précision :** de petites erreurs de virgule flottante (de l'ordre de 10^{-6}) avant l'arrondi sont attendues et n'affectent pas le résultat entier final.

5.13.4.3 🧠 Fondement théorique

Élément	Signification
$F(0,0)$	Composante DC — somme de tous les pixels, $F(0,0) = \sum f(x,y)$
Partie réelle $\mathrm{Re}(F)$	Projection du signal sur les cosinus

Élément	Signification
Partie imaginaire $\mathrm{Im}(F)$	Projection du signal sur les sinus
Complexité de cette implémentation	$\mathcal{O}((MN)^2)$ — c'est pourquoi la FFT, avec $\mathcal{O}(MN\log(MN))$, est indispensable pour les images réelles

5.13.4.4 📦 Spécification d'entrée et de sortie (VPL)

Entrée :

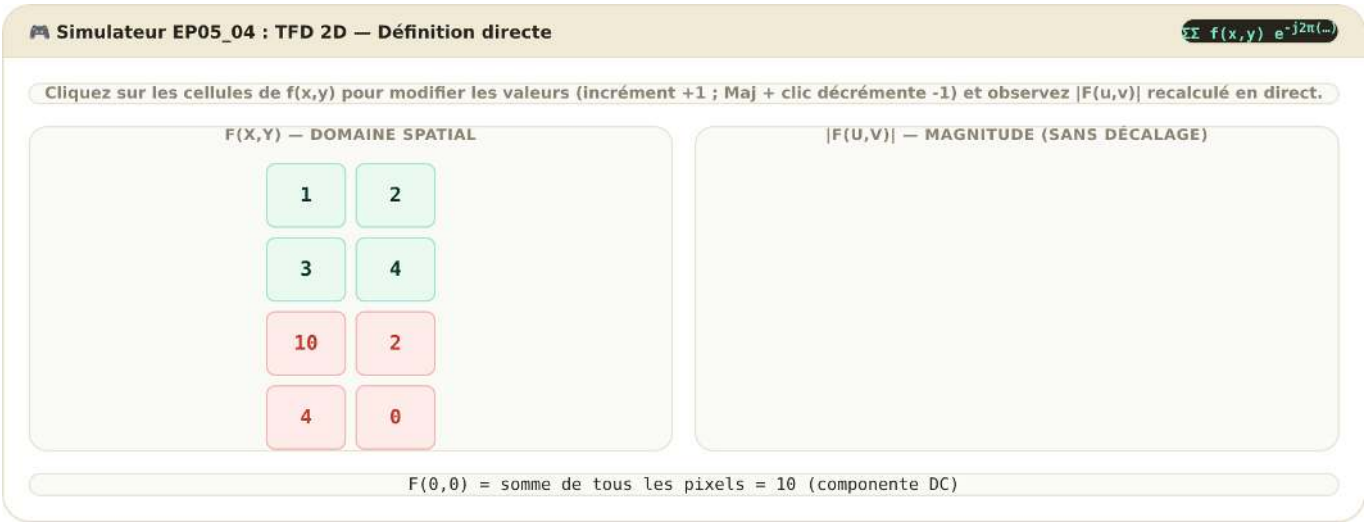
- Ligne 1 : Entier M.
- Ligne 2 : Entier N.
- Lignes suivantes : Éléments entiers de $f(x, y)$, M lignes.

Sortie :

- Matrice des magnitudes $|F(u, v)|$ arrondies, $M \times N$, séparées par des espaces.

5.13.4.5 📌 Exemples

Entrée	Sortie	Remarque
2 2 1 2 3 4	10 2 4 0	$F(0,0) = 1+2+3+4 = 10$ (DC = somme totale). $F(0,1) = (1-2)+(3-4) = -2 \to \lvert F\rvert = 2$. $F(1,0) = (1+2)-(3+4) = -4 \to \lvert F\rvert = 4$. $F(1,1) = (1-2)-(3-4) = 0$.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.fr/cap05/sim-ep0504.html

Figure 5.35: Simulateur EP05_04: DFT 2D manuel

```
%%writefile EP05_04.cpp
// your solution
```

```
Overwriting EP05_04.cpp
```

```
TestSuite("EP05_04.cpp").run()
```

```
<IPython.core.display.HTML object>
```

5.13.5 EP05_05 🏆 *Pipeline* JPEG complet : DCT, quantification et reconstruction

Vous avez été chargé de créer, à partir de zéro, un **codec JPEG didactique** dans un environnement embarqué, sans aucune bibliothèque d'image disponible — uniquement des opérations mathématiques de base. Le client veut comprendre exactement où la qualité est perdue et où elle est récupérée, bloc par bloc. C'est le défi final du chapitre : intégrer **tout** ce qui a été étudié — la DCT-II orthonormale, la quantification perceptuelle et la reconstruction via IDCT — dans un seul *pipeline* de bout en bout, traitant un bloc $N \times N$ du début à la fin, exactement comme le fait le standard JPEG en interne, 8×8 pixels à la fois.

5.13.5.1 📋 Directives d'implémentation

1. **Dimension du bloc :** Lire l'entier N.
2. **Bloc original :** Lire la matrice de pixels $f(x, y)$, N lignes avec N entiers dans $[0, 255]$.
3. **Table de quantification :** Lire la matrice Q, $N \times N$ entiers positifs.
4. **Centrage :** Soustraire 128 de chaque pixel : $g(x, y) = f(x, y) - 128$.
5. **DCT-II 2D orthonormale :** Calculer

$$C(u, v) = \alpha(u)\,\alpha(v) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} g(x, y)\, \cos\left[\frac{\pi(2x+1)u}{2N}\right] \cos\left[\frac{\pi(2y+1)v}{2N}\right]$$

   avec $\alpha(0) = \sqrt{1/N}$ et $\alpha(k) = \sqrt{2/N}$ pour $k > 0$.
6. **Quantification :** $\tilde{C}(u, v) = \text{round}(C(u, v)/Q(u, v))$.
7. **Déquantification :** $C'(u, v) = \tilde{C}(u, v) \times Q(u, v)$.
8. **IDCT-II 2D (inverse orthonormale) :** Calculer $g'(x, y)$ à partir de $C'(u, v)$ en utilisant la transformée inverse correspondante (même base, somme sur u, v).
9. **Inversion du centrage et arrondi :** $f'(x, y) = \text{round}(g'(x, y) + 128)$, restreint à l'intervalle $[0, 255]$ (*clipping*).
10. **Sortie :** Afficher le bloc reconstruit f', $N \times N$, entiers.

5.13.5.2 📌 Contraintes computationnelles

- ***Pipeline* complet obligatoire :** toutes les six étapes (centrer, DCT, quantifier, déquantifier, IDCT, inverser) doivent être implémentées — sauter la quantification ne réussit pas les tests, car le résultat serait identique à l'original.
- ***Clipping* :** les valeurs reconstruites hors de $[0, 255]$ doivent être tronquées (0 si négatif, 255 si supérieur à 255).
- **Arrondi :** à la fois dans la quantification et dans la reconstruction finale des pixels, utilisez un arrondi standard ; les cas de test évitent toute ambiguïté `.5`.
- **Base orthonormale :** la normalisation $\alpha(u)$ et $\alpha(v)$ doit être appliquée exactement comme spécifié — sans elle, l'IDCT ne reconstruit pas correctement.

5.13.5.3 🧠 Fondement théorique

Étape	Analogue dans le standard JPEG réel	Où la qualité est perdue
Centrage	Identique — la DCT suppose un signal centré sur zéro	Aucune perte
DCT-II	Étapes 3–4 du *pipeline* (Table 5.7)	Aucune perte (transformation exacte et réversible)

Étape	Analogue dans le standard JPEG réel	Où la qualité est perdue
Quantification	Étape 5 — division par $Q(u,v)$	**Principale source de perte** — les coefficients de haute fréquence deviennent zéro
IDCT	Reconstruction finale	Reconstruit exactement les coefficients *quantifiés*, pas les originaux

5.13.5.4 📦 Spécification d'entrée et de sortie (VPL)

Entrée :

- Ligne 1 : Entier N.
- N lignes suivantes : bloc original $f(x,y)$, entiers dans $[0,255]$.
- N lignes suivantes : table de quantification Q, entiers positifs.

Sortie :

- Bloc reconstruit $f'(x,y)$, $N \times N$, entiers dans $[0,255]$, séparés par des espaces.

5.13.5.5 📌 Exemples

Entrée	Sortie	Observation
4 120 130 125 128 115 140 135 122 118 150 160 130 110 120 145 138 4 6 8 10 6 8 10 12 8 10 12 16 10 12 16 20	118 126 119 131 114 143 140 119 117 149 159 130 107 121 146 139	Après DCT, quantification agressive des hautes fréquences (grandes valeurs de Q en bas à droite) et reconstruction via IDCT, le bloc reste **proche** de l'original, mais pas identique — la différence est le coût de la compression *lossy*.

5.13.5.6 💡 Conseil de débogage

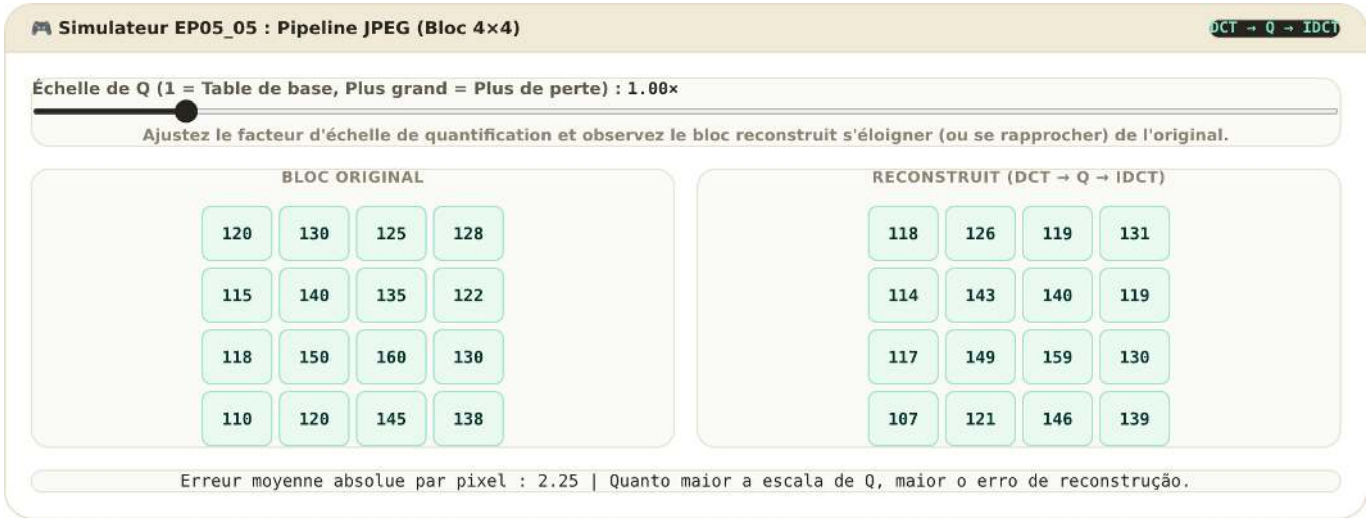
Si le résultat ne correspond pas, vérifiez dans cet ordre : (1) les coefficients DCT bruts (avant quantification) — ils doivent reconstruire l'original **exactement** via IDCT si vous sautez les étapes 6–7 ; (2) la table $\alpha(u)$ — erreur courante : appliquer $\sqrt{2/N}$ aussi pour $u=0$; (3) l'arrondi de la quantification, qui doit se produire **avant** de multiplier à nouveau par Q.

```
%%writefile EP05_05.cpp
// your solution
```

```
Overwriting EP05_05.cpp
```

```
TestSuite("EP05_05.cpp").run()
```

```
<IPython.core.display.HTML object>
```



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.fr/cap05/sim-ep0505.html

Figure 5.36: Simulateur EP05_05 : *Pipeline* JPEG complet par bloc

Références

BARRERA, Junior *et al.* MMach: a mathematical morphology toolbox for the KHOROS system. **Journal of Electronic Imaging**, v. 7, n. 1, p. 174-210, 1998.

DOUGHERTY, Edward R.; LOTUFO, Roberto A. **Hands-on morphological image processing**. *[S.l.]*: SPIE press, 2003. v. 59

KNUTH, Donald E. Literate Programming. **The Computer Journal**, v. 27, n. 2, p. 97-111, 1984.

LOTUFO, Roberto A. *et al.* MMachLib functions and MMach operators. *In*: 1997.

MATHERON, Georges. **Random Sets and Integral Geometry**. New York: John Wiley & Sons, 1975.

ZAMPIROLLI, Francisco A. **MCTest: Como Criar e Corrigir Exames Parametrizados**. *[S.l.]*: Independente, 2023.

ZAMPIROLLI, Francisco de Assis *et al.* A Practical Digital Image Processing Course with morph.py. *In*: Porto Alegre, RS, Brasil: Sociedade Brasileira de Computação (SBC), 2024. Disponível em: <https://sol.sbc.org.br/index.php/educomp/article/view/28199>

ZAMPIROLLI, Francisco de Assis *et al.* Teaching Hands-On Digital Image Processing with morph.py: Methods and Comprehensive Results. **Revista Brasileira de Informática na Educação**, v. 33, p. 990-1026, 2025.

ZAMPIROLLI, Francisco de Assis *et al.* Intelligent Feedback for Individualized Introductory Programming Exercises. **Computer Applications in Engineering Education**, v. 34, n. 1, p. e70132, 2026.

Annexe A

À propos de MCTest

MCTest est un système *open source* pour la création et la correction automatique d'examens paramétrés (Zampirolli, 2023). Il prend en charge les questions à choix multiples, dissertatives et de programmation, ces dernières intégrées au VPL de Moodle, décrit à l'Annexe B.

La version de MCTest utilisée dans ce livre est la **5.4**, disponible sur https://github.com/fzampirolli/mctest. Le sous-dossier `book` du dépôt contient les versions **5.3**, décrite dans (Zampirolli, 2023), et **5.4**, utilisée pour générer les épreuves présentées dans ce livre et actuellement en production à l'UFABC (https://mctest.ufabc.edu.br).

A.1 Installation de MCTest

L'installation recommandée utilise VirtualBox avec Ubuntu 22.04. Dans le terminal Ubuntu, en tant que *root*, exécutez :

```
wget https://raw.githubusercontent.com/fzampirolli/mctest/master/_setup-all.sh
```

Remplacez ensuite `yourLogin` par le nom d'utilisateur local et exécutez :

```
sed -i 's/\/home\/fz\//\/home\/yourLogin\//g' _setup-all.sh
source _setup-all.sh
pip install mysqlclient
```

Après quelques minutes, MCTest sera configuré. Pour l'exécuter :

```
source /home/yourLogin/PycharmProjects/runDjango.sh
```

Le système est alors accessible à l'adresse `http://127.0.0.1:8000`.

A.2 Créer un examen dans MCTest

Dans MCTest, chaque examen est configuré sur un écran **Exam**, qui réunit les classes, les sujets (associés à une matière, comme PDI-VC) et les questions — chaque question appartient à un seul sujet. Outre les attributs habituels de base de données, l'examen possède ses propres paramètres, comme le nombre de questions tirées et le nombre de variantes générées.

Pour les deux examens décrits dans cette annexe, un ensemble de questions paramétrées a été défini par examen, dont un sous-ensemble est tiré aléatoirement pour chaque variante, totalisant **110 variantes distinctes** — une par étudiant inscrit dans les classes.

Le bouton **Create-Variations** génère un nouvel ensemble de variantes à chaque activation. Lorsque les questions sont liées à des activités VPL, l'enseignant reçoit par e-mail un fichier `*linker.json` contenant tous les cas de test des variantes générées. Le bouton **Create-PDF** génère, pour chaque classe, un PDF avec les examens tirés par étudiant, ainsi qu'un fichier `*students_variations.csv` indiquant le nom, l'e-mail et la variante tirée de chaque étudiant.

> **!** Attention
>
> Chaque activation du bouton **Create-Variations** génère un nouvel ensemble de variantes d'examen. Après avoir imprimé le PDF, il est essentiel de **ne pas modifier les attributs de l'examen**, sous peine d'invalider la correction automatique des examens déjà imprimés ou passés.

A.3 Créer une question paramétrique

Une question paramétrique dans MCTest combine trois éléments :

1. **Description** — l'énoncé de la question, écrit en LaTeX, contenant des variables entre `[[code:variable]]`, remplacées par la valeur tirée pour chaque étudiant ;
2. **Bloc de définition** (`[[def: ... ]]`) — un extrait de code Python, intégré à la question elle-même, chargé de tirer les paramètres, de calculer la solution de référence et de construire les cas de test ;
3. **Cas de test pour Moodle** (bloc `moodle_cases`) — un dictionnaire sérialisé en JSON contenant les entrées et sorties attendues, consommé par l'activité VPL.

Un exemple simplifié de définition de question (adapté de la génération d'un échiquier `h × w`) est :

```
[[code:texto]]

\vspace{2mm}\noindent\textbf{Exemple d'entrée :}\vspace{-2mm}\
\begin{verbatim}
[[code:caso0_inp]]
\end{verbatim}
\vspace{-2mm}\noindent\textbf{Exemple de sortie :}\vspace{-2mm}\
\begin{verbatim}
[[code:caso0_out]]
\end{verbatim}

\begin{comment}
[[code:moodle_cases]]
\end{comment}

[[def:
import json
import numpy as np
from topic.morph import mm # DOIT inclure "topic." dans MCTest

def chess(h, w):
    m = np.zeros((h, w), dtype='int')
    for i in range(h):
        for j in range(w):
            if (i + j) % 2:
                m[i][j] = 1
    return m

# PARAMÈTRES UTILISÉS DANS L'ÉNONCÉ DE LA QUESTION
height = int(np.random.randint(5, 15))

# ÉNONCÉ COMPLET, AVEC LA VALEUR DE height DÉJÀ INTERPOLÉE
texto = (
    r"\vspace{2mm}\noindent\textbf{Description :}\vspace{-2mm} "
    r"Écrivez un programme qui lit un entier \texttt{W}, représentant la "
    r"largeur (nombre de colonnes) d'un plateau, et affiche ce plateau sous "
    r"la forme d'un échiquier, en utilisant les chiffres \texttt{0} et \texttt{1}. "
    r"Le plateau affiché aura toujours une hauteur (nombre de lignes) égale à "
    f"\\textbf{{{height}}} "
    r"et une largeur égale à la valeur \texttt{W} lue en entrée. "
```

```
    r"En considérant la position $(i, j)$ du plateau (indexée à partir de 0, "
    r"$i$ représentant la ligne et $j$ la colonne), la valeur affichée à "
    r"cette position doit être \texttt{1} si $i + j$ est impair, et \texttt{0} sinon. "
    r"Chaque ligne du plateau doit être affichée sur une ligne séparée, avec "
    r"les valeurs de chaque colonne séparées par un espace."
)

inp_list, out_list, test_cases = [], [], 4
for i in range(test_cases):
    width = int(np.random.randint(500, 1500) / 100)
    inp = str(width) + '\n'
    out = mm.drawImg(chess(height, width)) + '\n'
    inp_list.append(inp)
    out_list.append(out)

cases = {}
cases['skills']      = ["matrices", "boucles", "formatage de la sortie"]
cases['description'] = [{"text": latex_to_text(texto)}]
cases['input']       = inp_list
cases['output']      = out_list

moodle_cases = json.dumps(cases)

caso0_inp = cases['input'][0]
caso0_out = cases['output'][0]
]]
```

La valeur de `height` est tirée une seule fois par variante (pour apparaître dans l'énoncé), tandis que `width` est tirée à chaque cas de test, ce qui augmente la robustesse de la correction automatique.

Il convient de souligner le double rôle de la variable `texto` dans ce mécanisme. Elle est à la fois :

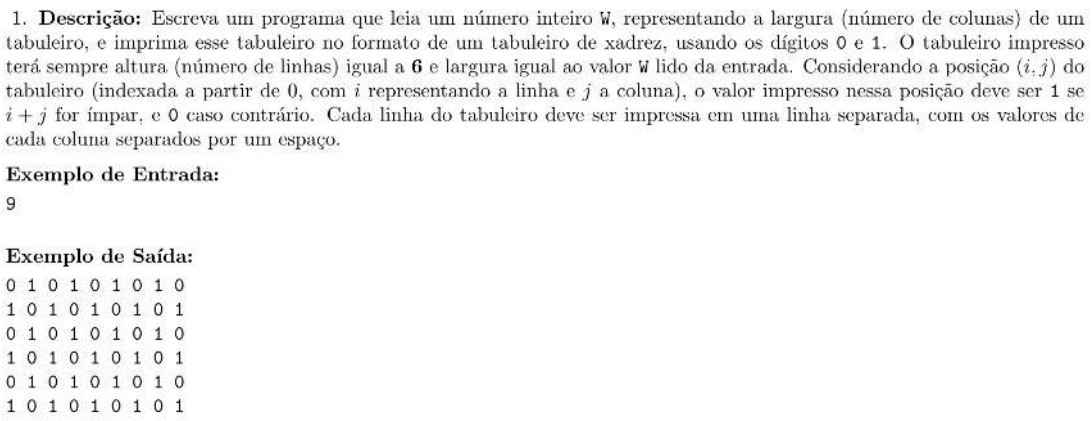
- **le contenu de l'énoncé imprimé**, inséré dans la description LaTeX de la question via la balise `[[code:texto]]`, apparaissant dans le PDF généré par le bouton **Create-PDF** ; et
- **une entrée du dictionnaire exporté vers Moodle**, via la clé `cases['description']`, qui vient composer le JSON `moodle_cases`. C'est précisément ce champ que l'activité VPL affiche à l'étudiant lorsqu'il ouvre la question dans Moodle pour la consulter ou soumettre une solution.

Il existe toutefois une différence importante entre ces deux usages : le PDF est composé en LaTeX et interprète donc normalement des commandes comme `\textbf{}`, `\texttt{}` ou `$...$`. Le VPL de Moodle, lui, **ne** traite **pas** le LaTeX — il attend du texte brut. C'est pourquoi, lors de la construction de `cases['description']`, `texto` n'est pas inséré directement, mais passé par la fonction utilitaire propre à MCTest `latex_to_text()`, qui supprime (ou convertit) les commandes et symboles LaTeX de l'énoncé, produisant une version en texte simple équivalente. Ainsi, `[[code:texto]]` dans le PDF continue de recevoir le `texto` original, avec toute sa mise en forme LaTeX, tandis que `cases['description']` reçoit `latex_to_text(texto)`, garantissant que l'énoncé affiché dans Moodle reste lisible même sans prise en charge du LaTeX.

Comme `texto` est construite à l'intérieur même du bloc `[[def: ...]]` — dans la même portée où `height`, `width` et les autres paramètres sont tirés —, elle est recalculée à chaque variante. Cela garantit que l'énoncé vu par l'étudiant dans Moodle est **toujours identique** à celui imprimé sur l'épreuve papier, même lorsque le texte change d'un étudiant à l'autre avec les valeurs tirées. La Section A.4 reprend ce point sur un cas réel, dans lequel l'énoncé est nettement plus long et décrit, en plus des paramètres numériques, le scénario visuel lui-même généré pour chaque variante.

La Figure A.1 montre la question de l'échiquier réellement générée par MCTest pour l'une des variantes, avec l'énoncé contenant déjà la valeur tirée de `height` et l'exemple d'entrée/sortie correspondant au premier cas de test :

En cliquant sur **Create-PDF** sur l'écran de la question, une nouvelle variante est générée à chaque clic, ce qui permet de vérifier l'énoncé avant de le publier.



1. **Descrição:** Escreva um programa que leia um número inteiro `W`, representando a largura (número de colunas) de um tabuleiro, e imprima esse tabuleiro no formato de um tabuleiro de xadrez, usando os dígitos `0` e `1`. O tabuleiro impresso terá sempre altura (número de linhas) igual a **6** e largura igual ao valor `W` lido da entrada. Considerando a posição (i, j) do tabuleiro (indexada a partir de 0, com i representando a linha e j a coluna), o valor impresso nessa posição deve ser `1` se $i + j$ for ímpar, e `0` caso contrário. Cada linha do tabuleiro deve ser impressa em uma linha separada, com os valores de cada coluna separados por um espaço.

Exemplo de Entrada:

```
9
```

Exemplo de Saída:

```
0 1 0 1 0 1 0 1 0
1 0 1 0 1 0 1 0 1
0 1 0 1 0 1 0 1 0
1 0 1 0 1 0 1 0 1
0 1 0 1 0 1 0 1 0
1 0 1 0 1 0 1 0 1
```

Figure A.1: Question de l'échiquier générée par MCTest, illustrant l'énoncé paramétré avec la valeur de `height` tirée pour la variante et l'exemple d'entrée/sortie du premier cas de test.

Pour les deux examens du cours, ce mécanisme a été utilisé plus largement : chaque question paramétrique définit non seulement des valeurs numériques, mais aussi, dans certains cas, de petites variations structurelles dans l'énoncé (par exemple, quelle direction cardinale — Nord, Sud, Est, Ouest — doit être évaluée), rendant plus difficile la recherche de solutions toutes faites sur internet ou via des outils d'IA générative.

A.4 Exemple réel : une question du Simulado 4

Pour illustrer le mécanisme décrit dans la section précédente avec un cas concret, voici une question effectivement posée lors du **Simulado 4** (sujet *im4-Opérateurs morphologiques*, difficulté 1, taxonomie de Bloom « se souvenir »), du type question de programmation paramétrée avec intégration Moodle+VPL.

La question demande à l'étudiant d'écrire un programme capable de :

- lire une image binaire, corrompue par un bruit poivre et sel, contenant au moins deux objets géométriques isolés ;
- appliquer un filtrage morphologique (ouverture suivie d'une fermeture) pour éliminer le bruit ;
- extraire, à partir de l'image filtrée, les mesures géométriques de chaque objet (aire, périmètre, centre, boîte englobante, circularité, solidité et nombre de sommets), à l'aide de la fonction auxiliaire `mm.measure` ;
- trier les objets par position (coordonnée X de la boîte englobante, avec Y comme critère de départage) et réattribuer les identifiants séquentiellement ;
- afficher la matrice nettoyée et le tableau des mesures dans le format attendu par le correcteur automatique.

Dans le bloc `[[def: ... ]]` correspondant, un générateur de scène (`gerarCena`) tire aléatoirement la hauteur et la largeur de l'image, le type, la taille et la position de chaque objet (carré, rectangle, triangle ou bloc), en garantissant qu'ils ne se chevauchent pas, puis ajoute un bruit poivre et sel avec une probabilité de 3 % par pixel. Dix cas de test distincts sont ainsi générés pour chaque variante de l'examen, et la paire entrée/sortie du premier cas est réutilisée dans l'énoncé lui-même comme exemple pour l'étudiant. La bibliothèque de morphologie (`mm`) est importée directement depuis `morph.py`, également disponible comme module utilitaire de MCTest.

En résumé, l'énoncé demande à l'étudiant un programme qui :

1. lit, sur la première ligne, deux entiers H et W (hauteur et largeur de l'image) ;
2. lit les H lignes suivantes de la matrice binaire (0 et 1 séparés par des espaces), à l'aide de `mm.readImg(H, W)` ;
3. applique un filtrage morphologique pour supprimer le bruit poivre et sel ;
4. affiche la matrice déjà filtrée, en 0 et 1 séparés par des espaces ;
5. extrait les mesures géométriques des objets avec `mm.measure(imgLimpa)` ;

6. trie les objets et affiche le tableau des mesures dans le format attendu.

L'énoncé apporte encore deux observations importantes pour la correction automatique : (i) l'aire calculée par OpenCV (`cv2.contourArea`) correspond au polygone continu délimité par les centres des pixels de bord, et est donc inférieure au simple comptage des pixels à 1 (`np.sum`) ; et (ii) la liste des objets doit être triée par ordre croissant selon la coordonnée X de la boîte englobante (`bbox[0]`), avec la coordonnée Y (`bbox[1]`) comme critère de départage, les identifiants étant réattribués séquentiellement de 1 à N après le tri.

Comme dans l'exemple de la Section A.3, le texte de l'énoncé n'est pas non plus fixe ici : il est construit à l'intérieur même du bloc `[[def: ... ]]`, dans la variable Python `texto`, et joue le même double rôle déjà décrit — il alimente le PDF via `[[code:texto]]` et est exporté, déjà converti par `latex_to_text()`, dans `cases['description']` au sein de `moodle_cases`, cette dernière étant la version affichée à l'étudiant dans le VPL de Moodle. La différence est que, dans ce cas réel, c'est le **scénario visuel** (image bruitée, nombre et position des objets) qui change d'un étudiant à l'autre à chaque variante, tandis que la rédaction de `texto` reste fixe entre les variantes, puisque seules les données numériques (image et cas de test) sont tirées. L'idéal serait que la rédaction varie elle aussi à chaque génération, comme dans l'exemple précédent, où la valeur de `height` était interpolée directement dans le texte de la description. La structure réellement utilisée (avec les extraits de tirage du scénario omis par souci de concision) est :

```
[[code:texto]]

\vspace{2mm}\noindent\textbf{Exemple d'entrée :}\vspace{-2mm}\
\begin{verbatim}
[[code:caso0_inp]]
\end{verbatim}
\vspace{-2mm}\noindent\textbf{Exemple de sortie :}\vspace{-2mm}\
\begin{verbatim}
[[code:caso0_out]]
\end{verbatim}

\begin{comment}
[[code:moodle_cases]]
\end{comment}

[[def:
import json
import numpy as n
import cv2
from topic.morph import mm # DOIT inclure "topic." dans MCTest

# ... génération de la scène, du bruit et des 10 cas de test (inplist/outlist) ...

# ÉNONCÉ COMPLET, AVEC EXPLICATION DE L'AIRE ET DU TRI
texto = (
    "Une image binaire corrompue par un bruit poivre et sel contient au moins deux objets
    géométriques isolés.\n\n"
    "Écrivez un programme qui :\n"
    "1. Lit deux entiers ..."
)

cases = {}
cases['skills']      = ["morphologie mathematique", "suppression de bruit", "mm.measure",
"tabulation"]
cases['description'] = [{"text": latex_to_text(texto)}]
cases['input']       = np.array(inplist).tolist()
cases['output']      = np.array(outlist).tolist()

moodle_cases = json.dumps(cases)
caso0_inp = cases['input'][0]
caso0_out = cases['output'][0]
```

```
]]
```

Ci-dessous, la photographie de la question effectivement générée et imprimée pour l'une des 110 variantes du Simulado 4, montrant l'énoncé, l'image d'entrée bruitée et l'exemple de sortie attendu (image filtrée et tableau des mesures) :

1. Uma imagem binária corrompida por ruído sal e pimenta contém no mínimo dois objetos geométricos isolados.

Escreva um programa que: 1. Leia dois inteiros na primeira linha representando a altura (H) e largura (W) da imagem. 2. Leia as H linhas seguintes da matriz binária (valores 0 e 1 separados por espaço) usando mm.readImg(H, W). 3. Aplique a filtragem morfológica para eliminar o ruído sal e pimenta. 4. Imprima a matriz limpa diretamente em valores 0 e 1 (0s e 1s separados por espaço). 5. Extraia as medições geométricas dos objetos usando mm.measure(imgLimpa). 6. Ordene a lista de objetos e imprima a tabela formatada das medidas.

Observação sobre a área: A área calculada pelo OpenCV (cv2.contourArea) mede o polígono contínuo delimitado pelos centros dos pixels de borda, resultando em um valor menor do que a simples contagem discreta de pixels 1 (np.sum).

Regra de Ordenação: A lista de objetos deve ser ordenada em ordem crescente pela posição X do retângulo delimitador (bbox[0]), utilizando a posição Y (bbox[1]) como critério de desempate caso dois objetos comecem na mesma coluna (medidas.sort(key=lambda m: (m['bbox'][0], m['bbox'][1]))). Em seguida, reatribua os IDs sequencialmente de 1 a N.

Exemplo de Entrada:

```
12 43
0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 1 1 1 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Exemplo de Saída:

```
0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 id area perimeter    cx    cy   x   y   w   h circularity solidity vertices
  1   20      18.0   6.5   2.0   4   0   6   5        0.78    1.000        4
  2   20      18.0  26.0   2.5  24   0   5   6        0.78    1.000        4
```

Figure A.2: Question réelle générée par MCTest pour le Simulado 4 — sujet « Opérateurs morphologiques ».

A.5 Remarques finales

Cette annexe a résumé les étapes pour installer MCTest, créer des examens et construire des questions paramétriques avec intégration au VPL de Moodle. L'utilisation combinée de variables tirées dans la description (`[[code:...]]`) et de code Python intégré à la question elle-même (`[[def: ... ]]`) permet de générer des centaines de variantes d'examen à partir d'un seul modèle, chacune corrigée automatiquement et de manière cohérente — la variable `texto`, en particulier, garantissant que le même énoncé est affiché aussi bien dans le PDF imprimé que lors de la consultation de la question par l'étudiant dans Moodle. L'exemple réel du Simulado 4 (Section A.4) illustre comment ce mécanisme est utilisé en pratique pour générer des questions de programmation avec des scénarios visuels différents pour chaque étudiant. L'Annexe B détaille comment ces variantes sont publiées et corrigées dans Moodle, et l'Annexe C décrit comment restreindre l'accès aux examens à l'aide de SEB.

Annexe B

À propos de Moodle (VPL)

Cette annexe décrit la configuration d'une activité **VPL (*Virtual Programming Lab*)** dans Moodle, le *plugin* utilisé pour la soumission et la correction automatique des Exercices de Programmation (EPs), des simulacres d'examen bimensuels et des examens décrits dans ce livre. La version de Moodle utilisée était la **4.5**.

B.1 Configurer une activité VPL

La création d'un examen dans MCTest avec intégration VPL (voir Annexe A) génère deux fichiers qui doivent être renommés en `linker.json` et `students_variations.csv`. Ces fichiers, avec `morph.py`, doivent être ajoutés aux **fichiers d'exécution** de l'activité VPL, ainsi que tous les fichiers contenus dans `VPL_modification/V14-AI-feedback.zip`, disponible dans le dépôt de MCTest à l'adresse https://github.com/fzampirolli/mctest. Tous doivent être marqués avec l'option « ***Files to keep when running*** ».

En résumé, l'activité VPL connaît alors, pour chaque étudiant, quelle variante de la question a été tirée (via `students_variations.csv`) et quel est l'ensemble de cas de test correspondant (via `linker.json`). Cela permet une correction individualisée, même lorsque des dizaines d'étudiants résolvent la même activité avec des variantes différentes.

B.2 Publier les EPs comme activités VPL

Les Exercices de Programmation (EPs) présentés à la fin de chaque chapitre de ce livre peuvent aussi être publiés comme activités VPL, en suivant le même mécanisme. Le déroulement type en classe était :

1. Ouverture des deux *notebooks* Colab du chapitre (théorique et pratique), en mettant l'accent sur les simulateurs interactifs ;
2. Exécution des blocs de code, en modifiant les paramètres pour renforcer la compréhension des concepts ;
3. Lors des séances en laboratoire, soumission des réponses aux EPs directement dans l'activité VPL correspondante, avec un retour immédiat sur la réussite ou l'échec des cas de test.

Si l'activité VPL n'est pas générée par MCTest, les fichiers `linker.json` et `students_variations.csv` ne sont pas nécessaires. Cependant, pour utiliser le *feedback* par IA (voir Annexe D), il faut ajouter les fichiers contenus dans `VPL_modification/V14-EP0_VPL-TEMPLATE.zip`, disponible dans le même dépôt (https://github.com/fzampirolli/mctest), en suivant la procédure décrite à la Section B.1.

> **Réutilisation des cas de test**
>
> Les fichiers `.cases` utilisés lors de la validation locale (classe `TestSuite`, décrite dans le corps de ce livre) sont compatibles avec le format attendu par VPL, ce qui permet de réutiliser le même ensemble de tests aussi bien lors du développement de l'EP que lors de la correction automatisée dans Moodle.

Lors des simulacres d'examen bimensuels — appliqués avec SEB (voir Annexe C) et donnant droit à un bonus de 5 % sur la note finale —, des activités VPL ont été utilisées avec des EPs semblables à ceux des listes régulières, mais corrigées avec un accès à internet restreint, renforçant le caractère évaluatif de l'activité.

B.3 Remarques finales

Cette annexe a décrit la configuration des activités VPL dans Moodle pour la correction automatique des EPs, des simulacres d'examen et des examens paramétrés. La combinaison de MCTest (génération des variantes et des cas de test), de VPL (soumission et correction) et de SEB (restriction d'accès) forme le tripode sur lequel repose l'évaluation automatisée et individualisée utilisée tout au long du cours.

Annexe C

Utilisation du SEB lors des simulacres d'examen bimensuels et des évaluations

Outre les deux examens à questions paramétrées (voir Annexe A), le ***Safe Exam Browser* (SEB)** a également été utilisé pour restreindre l'accès lors des **simulacres d'examen bimensuels** — activités VPL (*Virtual Programming Lab*) avec des Exercices de Programmation (EPs) semblables à ceux des listes régulières, donnant droit à un bonus pouvant aller jusqu'à 5 % de la note finale —, ainsi que lors des autres évaluations pratiques.

Le SEB garantit un environnement d'évaluation sécurisé en bloquant l'accès aux ressources externes non autorisées sur les machines du laboratoire. Pour la mise en œuvre adoptée dans ce travail, le système d'exploitation **Windows 10** et la version **3.7.1.704** du SEB, disponible sur https://safeexambrowser.org/, ont été utilisés ; ils doivent être préalablement installés sur tous les postes du laboratoire.

La procédure de configuration du fichier d'environnement du SEB et de son association à l'activité VPL dans Moodle est décrite ci-dessous.

C.1 Configurer l'application dans *SEB Tool*

Pour générer le fichier de configuration `.seb` qui sera distribué aux étudiants, ouvrez l'utilitaire *SEB Tool* et suivez les étapes ci-dessous.

1. ***General***
   - Dans le champ ***Start URL***, saisissez l'URL directe de l'activité VPL dans Moodle.
   - *Remarque :* s'il est nécessaire de naviguer entre plusieurs activités au sein du même cours, saisissez l'URL principale du cours et, dans l'onglet ***Browser***, cochez l'option ***Allow navigation back/forward in exam***.
2. ***Network***
   - Incluez les URL autorisées pour la navigation. Pour restreindre l'accès à l'activité souhaitée, utilisez une expression de filtre du type `/mod/vpl/*?id=4624*`, où le numéro final correspond à l'identifiant de l'activité VPL dans Moodle. Dans le Table C.1, la dernière ligne contient l'URL de la page principale du cours (identifiant `475`), mais elle peut aussi être remplacée par l'URL d'une section contenant plusieurs activités. Ce réglage est facultatif et utile lorsqu'il y a plus d'une activité VPL. Dans ce cas, la première ligne du tableau doit être répétée pour chaque activité évaluative devant rester accessible pendant l'*examen*.
   - **Réglages généraux de l'onglet *Filter*** :
     ☒ *Activate URL filtering*
     ☐ *Filter also embedded content*

Tableau C.1: Expressions de filtrage d'URL utilisées dans la configuration du SEB.

Active	*Regex*	*Expression*	*Action*
[x]	[ ]	`https://moodle.ufabc.edu.br/mod/vpl/*?id=4624*`	*Allow*
[x]	[ ]	`https://moodle.ufabc.edu.br/login/index.php*`	*Allow*

Active	*Regex*	*Expression*	*Action*
[x]	[ ]	`https://moodle.ufabc.edu.br/course/switchrole.php*`	*Allow*
[x]	[ ]	`https://mctest.ufabc.edu.br/compare`	*Allow*
[x]	[ ]	`https://moodle.ufabc.edu.br/enrol/index.php?id=475`	*Allow*

3. ***Security***
   - Si le laboratoire utilise un projecteur ou un second écran, réglez le champ ***Maximum allowed number of connected displays*** sur une valeur supérieure à 1, pour éviter que le SEB ne bloque l'affichage.
4. ***Config File***
   - Cliquez sur ***Save Settings As...*** et enregistrez le fichier avec l'extension `.seb`.
5. ***Exam***
   - Cochez l'option ***Use Browser Exam Key and Configuration Key***.
   - Copiez le code généré dans le champ ***Browser Exam Key***. *Important :* copiez cette clé seulement **après** avoir enregistré le fichier à l'étape précédente.

C.2 Associer la clé SEB à l'activité VPL dans Moodle

Après avoir généré le fichier et obtenu la *Browser Exam Key*, accédez à Moodle pour appliquer les restrictions.

1. Sur la page de l'activité VPL, cliquez sur l'icône d'engrenage (**Paramètres**).
2. Accédez à ***Configuration*** → ***Submission restrictions*** et cliquez sur ***Show more***.
3. Réglez l'option ***SEB browser required*** sur ***Yes***.
4. Collez la clé copiée dans le champ ***SEB exam key(s)***.

C.3 Restriction par adresse IP (facultatif)

Pour garantir que les soumissions proviennent exclusivement des ordinateurs du laboratoire, il est possible de combiner le SEB avec le filtre réseau de Moodle, en renseignant le champ ***Allowed submission from net***.

- **Plages consécutives :** utilisez un tiret pour définir l'intervalle. Exemple : `111.11.11.1-62` (autorise les IP de `111.11.11.1` à `111.11.11.62`).
- **IP non consécutives :** indiquez les adresses individuelles séparées par des virgules.

C.4 Remarques finales

L'intégration du SEB avec le VPL dans Moodle crée un déroulement simple pour l'étudiant : il suffit de double-cliquer sur le fichier `.seb` fourni par l'enseignant pour que le navigateur sécurisé s'ouvre directement sur l'activité configurée.

La combinaison de la *Browser Exam Key* avec la restriction par plage d'adresses IP des machines du laboratoire fournit une solution robuste pour les évaluations de programmation *en ligne*, rendant plus difficiles les accès indus et contribuant à l'intégrité du processus d'évaluation.

Annexe D

Génération de matériel pédagogique avec *Gemini Notebook*

Cette annexe décrit l'utilisation du ***Gemini Notebook*** **(*NotebookLM*)** comme outil d'appui à la préparation du matériel pédagogique du cours, complétant l'usage de l'IA pour la révision des textes et des codes, mentionné dans la préface de ce livre.

D.1 Vidéos conceptuelles et de résolution des EPs

Pour chaque chapitre, un projet a été créé dans *Gemini Notebook* à partir du PDF complet du livre et du fichier `morph.py` comme sources. À partir de ces sources, *Gemini Notebook* a généré automatiquement une courte vidéo (7 minutes en moyenne), utilisée en ouverture des cours. Les chapitres alternaient deux types de vidéo :

- **Vidéo conceptuelle**, présentant les fondements théoriques du chapitre, généralement diffusée lors du premier cours consacré au sujet ;
- **Vidéo de résolution des EPs**, présentant une stratégie de résolution des Exercices de Programmation, diffusée lors du cours suivant, lorsque la plupart des EPs étaient résolus collectivement avec la classe.

> **i** Rôle de la vidéo en classe
>
> La vidéo générée par *Gemini Notebook* ne remplace pas l'explication de l'enseignant. Elle sert de brève introduction et de motivation au sujet, contextualisant l'étudiant avant la présentation des *slides* et l'ouverture des *notebooks* Colab.

D.2 *Slides* d'appui

En complément des vidéos, *Gemini Notebook* a également généré, à partir des mêmes sources (le PDF du livre et le fichier `morph.py`), un ensemble d'environ 12 *slides* par chapitre, couvrant à la fois les concepts théoriques et la partie pratique. La génération utilisait un *prompt* spécifique à chaque chapitre, délimitant la portée du contenu à synthétiser et évitant à la fois d'anticiper des sujets de chapitres ultérieurs et de reproduire de longs passages du livre sans adaptation pédagogique.

Après la présentation des *slides*, le cours se poursuivait par l'ouverture des deux *notebooks* Colab du chapitre (théorique et pratique), en mettant l'accent sur les simulateurs interactifs et l'exécution des blocs de code avec modification des paramètres.

D.3 Matériel principal et remarques finales

La génération de vidéos et de *slides* avec *Gemini Notebook*, à partir du livre lui-même et de la bibliothèque `morph.py`, a permis de standardiser l'ouverture des cours et de réduire le temps de préparation du matériel pédagogique, sans se passer pour autant de la curation de l'enseignant dans l'élaboration des *prompts* et la conduite des activités en classe.

Les vidéos et *slides* produits par *Gemini Notebook* ont un caractère **motivationnel** et servent de point de départ pour discuter des concepts présentés dans chaque chapitre. Comme tout contenu généré par IA, ils peuvent contenir des imprécisions ou des erreurs occasionnelles. Dans certains cas, ces imprécisions sont exploitées intentionnellement pendant le cours pour vérifier si les étudiants suivent la présentation de manière critique et parviennent à identifier des incohérences conceptuelles.

Le contenu officiel du cours est cependant le matériel produit selon la méthode décrite dans la préface de ce livre, disponible en trois formats complémentaires : **HTML**, pour la navigation avec des simulateurs interactifs ; **PDF**, pour la lecture et l'impression ; et *notebooks* **Jupyter** (`.ipynb`), pour l'exécution et l'expérimentation des codes. Tout le matériel généré par *Gemini Notebook* doit être compris comme une ressource complémentaire à ce contenu principal.

Comme pour les autres usages de l'IA générative décrits dans ce livre, la responsabilité de la qualité technique, de la justesse conceptuelle et de l'adéquation pédagogique du matériel demeure celle de l'enseignant.

Annexe E

Utilisation de l'IA dans l'évaluation des étudiants : le projet vpl-ai-feedback

Cette annexe décrit **vpl-ai-feedback**, un système *open source* développé pour appuyer la correction des soumissions de code envoyées au VPL (Moodle) grâce à un *feedback* généré par Intelligence Artificielle (IA). Contrairement à un correcteur automatique traditionnel — qui se contente de comparer les sorties aux cas de test —, **vpl-ai-feedback** envoie le code de l'étudiant, accompagné de la grille de correction de la question, à un modèle de langage (LLM), qui renvoie une analyse qualitative par critère, dans le style d'un ***feedback* socratique** : l'étudiant reçoit des commentaires indiquant ce qui a été fait correctement, ce qui peut être amélioré et pourquoi, aussi bien dans les activités **formatives** (exercices d'entraînement) que dans les activités **évaluatives** (simulacres d'examen) du VPL.

Ce mécanisme de *feedback* socratique dans les activités VPL a été proposé par Zampirolli et al. (2026). L'étude décrit l'intégration de l'IA au processus d'évaluation d'exercices de programmation paramétrés dans VPL/Moodle, à l'aide d'un ensemble de sept LLM utilisés pour analyser le code des étudiants et générer un *feedback* automatisé à chaque demande de correction, avec des résultats préliminaires indiquant une perception positive des étudiants quant à la génération d'idées, à la clarté des explications et à l'autonomie d'apprentissage.

Le reste de cette annexe détaille la mise en œuvre pratique de ces idées dans le projet **vpl-ai-feedback** — dont le code est disponible à l'adresse https://github.com/fzampirolli/vpl-ai-feedback — et décrit spécifiquement son utilisation dans les trois classes de PDI-VC entre mai et août 2026.

> ⚠ Attention : l'IA ne remplace pas le jugement de l'enseignant
>
> L'utilisation de l'IA pour **évaluer** les étudiants **n'est pas recommandée comme source unique de note**, car les modèles de langage peuvent **halluciner** — c'est-à-dire produire des évaluations, des critères ou des justifications plausibles mais incorrects, même face à un code correct ou incorrect. Un LLM peut attribuer une note élevée à un code comportant une erreur subtile, ou une note faible à un code correct mais écrit de façon inhabituelle, simplement parce que l'explication générée « paraît » cohérente. **L'IA doit être utilisée uniquement comme un complément à l'évaluation**, jamais comme un substitut à l'enseignant. L'Annexe A et l'Annexe B décrivent les mécanismes de correction objective (cas de test) qui restent la base de la note officielle ; cette annexe traite exclusivement de l'utilisation de l'IA comme couche supplémentaire de *feedback* qualitatif.

E.1 Contexte d'utilisation

vpl-ai-feedback a été utilisé dans trois classes de la discipline **PDI-VC**, enseignée entre mai et août 2026, totalisant environ une centaine d'étudiants. Le *feedback* par IA a été employé de trois façons complémentaires :

1. ***Feedback* continu dans les activités formatives** — exercices d'entraînement soumis au VPL tout au long du cours, où l'étudiant pouvait soumettre à nouveau son code autant de fois qu'il le souhaitait.

À chaque soumission, il recevait un *feedback* qualitatif généré par IA, en plus de la correction objective effectuée par le VPL, décrite dans Zampirolli et al. (2026).

2. ***Feedback* complémentaire lors des simulacres d'examen (activités évaluatives bonus)** — quatre simulacres réalisés au cours du trimestre, prévus dans le plan de cours comme activités bonus, chacun valant jusqu'à 5 % de la note finale. Appliqués environ 30 minutes avant la fin des séances pratiques bimensuelles en laboratoire, les simulacres utilisaient **vpl-ai-feedback** pour générer une grille d'évaluation individuelle, envoyée par e-mail avec un résumé comparatif entre la note attribuée par Moodle/VPL et l'évaluation produite par l'IA.

3. ***Feedback* complémentaire lors des Examens 1 et 2 (évaluations officielles)** — les deux examens formels du trimestre, chacun valant une part plus importante de la note finale du cours, ont également commencé à utiliser **vpl-ai-feedback** après leur passation. Contrairement aux simulacres (bonus), la note officielle est ici déjà définie par la correction objective du VPL et/ou l'évaluation manuelle de l'enseignant avant la publication des résultats ; la grille générée par l'IA n'est envoyée à l'étudiant qu'ensuite, comme support pour la révision de sa propre performance — renforçant, y compris dans les évaluations à plus fort coefficient, la séparation entre *note officielle* et *feedback* complémentaire détaillée à la Section D.7.

Une enquête d'opinion, à laquelle 102 étudiants ont répondu avant l'adoption du système, a indiqué une forte adhésion à la proposition. Seuls 2 étudiants ont attribué la note 1 (rejet) et 7 ont attribué la note 2 quant à l'intérêt de recevoir un *feedback* automatique de code par IA. 19 autres se sont déclarés indifférents (note 3), tandis que la majorité — 38 et 37 étudiants — a attribué les notes 4 et 5 (forte approbation), respectivement, totalisant les 102 répondants. Ce résultat a motivé l'adoption du système comme complément au processus formatif, et non comme substitut à la correction humaine.

! La note officielle n'a jamais été la note de l'IA

Il est essentiel de souligner que, comme défini dans le plan de cours, **la note utilisée aussi bien pour le calcul du bonus de chaque simulacre que pour la note officielle des Examens 1 et 2 a toujours été la note attribuée par le VPL** (basée sur les cas de test) et/ou par l'évaluation manuelle de l'enseignant — et **non** la note suggérée par l'IA. La grille générée par **vpl-ai-feedback** a été envoyée à l'étudiant uniquement comme support d'apprentissage — jamais comme critère d'attribution de note, que ce soit dans les activités bonus ou dans les évaluations officielles. Cette séparation entre *note officielle* (VPL/enseignant) et *feedback complémentaire* (IA) est le principe central de cette annexe et a été reprise à la Section D.8.

E.2 Architecture du projet

vpl-ai-feedback est un *script* Python asynchrone organisé de la façon suivante :

```
.
   config.yaml              ← configuration (identifiants, fournisseur, poids) - JAMAIS
versionné
   config.yaml.example      ← modèle public de configuration
   main.py                  ← script principal (async), orchestre tout le processus
   run.sh                   ← wrapper d'exécution (bash run.sh config.yaml)
   gerar_relatorio.py       ← convertit le fichier consolidé *_ALL.txt en CSV
   enviar_email.py          ← envoie les grilles par e-mail à chaque étudiant
   providers/                ← clients des API de LLM
       __init__.py          ← fabrique de clients (Factory)
       base.py               ← logique de retry, backoff et repli entre modèles
       groq.py
       deepseek.py
       gemini.py
   core/                      ← logique métier
```

```
      grader.py              ← identifie les fichiers par question, construit le prompt,
appelle le LLM
      utils.py
  <dossier_de_la_classe>/    ← soumissions téléchargées depuis Moodle VPL
      Nom étudiant - login/
          <timestamp>/                ← dernière soumission de l'étudiant
              Q1.py                   ← modèle pour les examens à plusieurs questions
              Q2.py
              rubrica.txt             ← généré par le LLM (uniquement si l'évaluation est
    valide)
          <timestamp>.ceg/
              execution.txt           ← note/objective attribuée par le VPL
      ...
```

Le système accepte trois fournisseurs de LLM — **Groq**, **DeepSeek** et **Gemini** — configurables dans `config.yaml` via la clé `llm.provider`. Chaque fournisseur peut avoir **plusieurs modèles** enregistrés ; à chaque appel, la liste est **mélangée**, répartissant la charge entre les étudiants traités en parallèle et réduisant le risque d'erreurs de limite de requêtes (HTTP 429). Lorsqu'un modèle échoue, le système réessaie jusqu'à trois fois avant de passer au suivant dans la liste, en respectant le délai d'attente suggéré par le fournisseur. Les erreurs d'authentification (401) ou de solde insuffisant (402) interrompent immédiatement la tentative auprès de ce fournisseur.

Un aspect important du projet concerne la confidentialité des données. **Le nom, l'e-mail, le login, le numéro d'étudiant et le numéro d'identité nationale de l'étudiant ne sont jamais envoyés à l'API du LLM**. Ne sont transmis pour traitement que : le nom du fichier contenant le code source ; le code lui-même (sans les commentaires) ; le *prompt* de la grille — qui ne contient pas de données personnelles — ; et, lorsqu'ils sont disponibles dans le fichier `execution.txt` généré par le VPL, l'énoncé officiel de la question tirée pour cet étudiant et le résultat brut des cas de test réellement exécutés (entrée, sortie produite par le code et sortie attendue). Ces deux derniers éléments, bien qu'extraits d'un fichier spécifique à chaque étudiant, ne contiennent pas non plus d'identification personnelle — seulement le texte de la question et les valeurs d'entrée/sortie des tests automatiques —, et sont utilisés comme preuve objective pour réduire le risque que l'IA identifie incorrectement le type de question ou évalue la logique du code par simple lecture statique (Section D.3).

Il n'en demeure pas moins important de reconnaître que les trois fournisseurs actuellement pris en charge — Groq, DeepSeek et Gemini — sont des services commerciaux exploités par des entreprises étrangères, dont les modèles s'exécutent sur une infrastructure située hors du pays et sont soumis à des politiques d'utilisation, de disponibilité et de coût qui échappent au contrôle de l'établissement d'enseignement. Cette dépendance à des fournisseurs externes comporte des risques qui vont au-delà de la simple correction ponctuelle de code : changements de prix, abandon de modèles, indisponibilité temporaire du service, modifications unilatérales des conditions d'utilisation et, dans les cas les plus sensibles, restrictions géopolitiques d'accès peuvent compromettre la continuité d'un système d'évaluation devenu structurellement dépendant de ces API. C'est pourquoi il est stratégique que des établissements d'enseignement supérieur comme l'UFABC investissent dans le développement et l'hébergement de **leurs propres fournisseurs d'IA** — par exemple, des modèles ouverts (*open-weight*) exécutés sur une infrastructure locale ou un cloud souverain —, réduisant la dépendance aux services externes, en particulier d'autres pays, et élargissant le contrôle institutionnel sur les coûts, la disponibilité, la confidentialité des données des étudiants et la continuité pédagogique à long terme. La conception de **vpl-ai-feedback** favorise déjà cette voie : l'architecture de `providers/` a été construite comme une fabrique (*factory*) extensible, dans laquelle un nouveau fournisseur — y compris un modèle hébergé localement par l'établissement lui-même, avec une interface compatible — peut être ajouté avec un faible effort d'intégration.

E.3 Le *prompt* universel et la grille en deux étapes

Chaque type d'examen est décrit dans un fichier de *prompt* (par exemple, `Simulado4.txt`), référencé dans `grading.prompt_file` du `config.yaml`. Ce fichier indique au LLM d'effectuer l'évaluation en **deux étapes**

:

1. **Identification de la question** — le LLM détermine le type spécifique d'opération tiré pour cet étudiant (utile lorsqu'il existe des questions paramétrées tirées et mélangées par étudiant, comme décrit à l'Annexe A) ;
2. **Application de la grille correspondante** — le LLM évalue le code selon des critères notés, chacun avec un barème maximal de points, renvoyant à la fin une note au format `NOTE FINALE : X/POIDS`.

Dans les examens paramétrés, le fichier de *prompt* contient généralement **plusieurs grilles parallèles**, une pour chaque type de question possible, délimitées par des marqueurs tels que `[START_RUBRICA_TIPO_A]` / `[END_RUBRICA_TIPO_A]`. Le LLM identifie d'abord quel type a été tiré pour cet étudiant et applique **uniquement** la grille correspondante, en ignorant les autres — ce qui évite que des critères d'un autre type ne contaminent la note.

Lorsque l'examen comporte plusieurs questions par activité VPL, les fichiers de code doivent suivre le modèle `Q1.*`, `Q2.*`, etc. — un fichier par question, avec une extension libre selon le langage choisi par l'étudiant —, et chaque question a un poids configurable individuellement dans `grading.weights` du `config.yaml`. Lorsque l'examen ne comporte qu'une seule question et n'a pas été généré par MCTest, le système accepte n'importe quel nom de fichier avec une extension prise en charge, à condition qu'il soit le seul fichier de code présent dans le dossier de soumission.

Dans le cas du Simulado 4 (sujet *Opérateurs morphologiques*), par exemple, l'un des types possibles définissait trois critères, pour un poids total de 100 points pour cette question :

Critère	Description	Note maximale
1 — Entrée des données	Lecture de `H`, `W` et de la matrice binaire (`mm.readImg`)	25 pts
2 — Traitement morphologique	Ouverture + Fermeture (`mm.sebox()`), `mm.measure`, tri par `bbox[0]`/`bbox[1]` et réattribution des identifiants	50 pts
3 — Sortie et mise en forme	Affichage de l'image nettoyée (`mm.drawImg`) et tableau des mesures formaté	25 pts

Le *prompt* exige également un **format de réponse obligatoire** (largeur de ligne, ordre des sections, marqueur `NOTE FINALE :`), ce qui rend l'extraction automatique de la note et la constitution du rapport consolidé plus fiables — bien que, comme discuté à la Section D.8, cela n'élimine pas le risque d'erreur sémantique dans l'évaluation du contenu.

E.4 Deux couches supplémentaires de preuves

Pour réduire le risque que le LLM hallucine le type de question ou la justesse du code — le risque central discuté à la Section D.7 —, **vpl-ai-feedback** a commencé à fournir à l'IA deux sources de preuves objectives, extraites automatiquement du fichier `execution.txt` généré par le VPL, en plus du code source :

- **Énoncé officiel de la question.** Comme les questions sont généralement tirées et mélangées par étudiant dans MCTest (Annexe A), le `execution.txt` de chaque étudiant contient déjà, dans le champ « Résumé de la question », le texte exact de la question qui lui a été attribuée. Le système extrait automatiquement cet extrait et l'envoie au LLM comme preuve **primaire** pour identifier le type/la variante de la question — plus fiable que de se fier uniquement à une lecture statique du code, en particulier pour les soumissions incomplètes ou ambiguës entre deux types proches (par exemple, une érosion « plate » *versus* « pondérée »). Lorsque le code soumis diverge de ce que demande l'énoncé, le système n'ignore pas l'énoncé et ne « corrige » pas silencieusement la lecture : il conserve l'identification

fondée sur l'énoncé, réduit le niveau de confiance de l'évaluation à « Faible » et explicite la divergence dans le *feedback* envoyé à l'étudiant.

- **Résultat réel de l'exécution sur le VPL.** Lorsqu'ils sont disponibles, le système envoie aussi au LLM les cas de test effectivement exécutés sur ce code — entrée, sortie produite et sortie attendue — comme preuve objective de correction, à utiliser pour confirmer ou infirmer la lecture de la logique avant de noter les critères de traitement et de sortie, plutôt que de laisser l'IA se fier uniquement à une simulation mentale du code (ce qui est particulièrement défaillant dans les boucles avec `min`/`max`, les décalages d'indices ou le calcul du seuil d'Otsu, où des erreurs subtiles passent inaperçues lors d'une lecture statique).

De plus, le LLM est instruit de déclarer explicitement son **niveau de confiance** (Élevé, Moyen ou Faible) dans l'identification de chaque type de question, avec justification lorsque la confiance n'est pas Élevée. Cette valeur est extraite automatiquement et alimente une colonne `Revisar_Manualmente` (à réviser manuellement) dans le rapport CSV consolidé (Section D.4), permettant à l'enseignant de prioriser exactement les cas où l'IA elle-même reconnaît une plus grande incertitude — plutôt que de traiter toutes les évaluations d'une classe comme également fiables.

E.5 Déroulement de l'exécution

L'exécution type, effectuée via `bash run.sh config.yaml`, suit les étapes suivantes :

1. Charge `config.yaml` et localise le dossier des soumissions (`paths.student_base_dir`) ;
2. Pour chaque étudiant, identifie la **dernière soumission** (le dossier au *timestamp* le plus récent) ;
3. Extrait la note objective de Moodle à partir de `*.ceg/execution.txt` ;
4. Pour chaque question de l'examen, localise le fichier de code (`Q1.*`, `Q2.*`, ou le seul fichier, dans le cas d'un examen non généré par MCTest), construit le *prompt* (code + grille) et l'envoie à un modèle tiré de la liste configurée ;
5. Traite tous les étudiants **en parallèle**, avec un contrôle de la concurrence ;
6. Génère, pour chaque étudiant, le fichier `rubrica.txt` — **uniquement si l'IA renvoie au moins une évaluation valide** pour les questions comportant du code ; sinon, le fichier n'est pas enregistré, ce qui force une nouvelle tentative à l'exécution suivante ;
7. Consolide tous les fichiers `rubrica.txt` en un seul `*_ALL.txt` et génère un rapport `*_relatorio.csv`, comparant la note Moodle, la note de l'IA et l'écart entre les deux, par question et au total.

Situation	`rubrica.txt` est-il enregistré ?
Toutes les questions avec code évaluées avec succès	✅ Oui
L'IA a échoué sur au moins une question avec code	❌ Non — retraité à l'exécution suivante
Étudiant n'ayant soumis aucun fichier de code	❌ Non
Type de question identifié comme INCONNU	❌ Non

E.6 Exemple illustratif : grille pour un examen à trois questions

i À propos de l'origine de cet exemple

L'extrait suivant **ne reproduit le code d'aucun étudiant en particulier**. Il s'agit d'un exemple reconstruit par l'auteur, reproduisant un *schéma* d'erreur observé de façon répétée dans les classes de PDI-VC — la confusion entre une fonction morphologique 2D (`mm.ero`/`mm.ero0`) et une opération 1D sur des signaux — sans citer ni identifier aucune soumission réelle. Ce choix évite tout risque d'atteinte au droit d'auteur de l'étudiant sur son propre code source, même anonymisé, la paternité de l'œuvre restant celle de l'auteur original même lorsque le nom et le *login* sont retirés.

L'extrait suivant illustre la sortie générée par **vpl-ai-feedback** pour un étudiant sur un examen à **trois**

questions (Q1, Q2, Q3), avec le modèle `deepseek-chat`. Le résumé comparatif apparaît en haut du fichier, suivi de l'énoncé officiel de chaque question, du code soumis et de l'évaluation par critère.

```
RÉSUMÉ - IA (deepseek-chat)  x  MOODLE

Poids total : 100 pts

Q1 (IA) :  3.3 / 33 pts
Q2 (IA) :  6.7 / 33 pts
Q3 (IA) : 33.3 / 34 pts

Moodle : (Q1=33 + Q2=0 + Q3=34) = 67 pts
IA     : 43.3 / 100 pts

Différence (IA - Moodle) : -23.7 pts

Q1 : type identifié = B  | confiance = Faible (le code ne
  correspond pas à l'énoncé de la question)  À RÉVISER
Q2 : type identifié = A  | confiance = Élevée (l'énoncé officiel
  confirme le type et la variante)
Q3 : type identifié = C1 | confiance = Élevée (l'énoncé officiel
  confirme une érosion 2D avec OpenCV intégrée)
```

Q1 — le code ne correspond pas à l'énoncé (confiance faible) : l'énoncé officiel de la question, extrait de `execution.txt`, demandait une érosion morphologique **1D** sur un signal (lecture de N, M, du signal et de l'élément structurant B, suivie du minimum des voisins actifs). Le code soumis, cependant, ne lit que deux entiers et appelle `mm.ero0` — une fonction d'érosion **2D** pour images, sans rapport avec l'opération demandée.

```
Type identifié : TYPE B (Érosion 1D - plate)
Confiance : Faible - le code ne correspond pas à l'énoncé de la
question (utilise mm.ero0, fonction d'érosion 2D, au lieu d'implémenter
une érosion 1D plate sur un signal)

Critère 1 - [3.33/10.00 pts] :
- Le code lit deux entiers sur une seule ligne (qui correspondraient à
  N et M), mais ne lit pas le signal ni l'élément structurant B comme
  des vecteurs 1D. Il les traite plutôt tous les deux comme des images
  via mm.readImg, ce qui ne correspond pas à la lecture attendue d'un
  signal 1D et de B sur des lignes séparées.

Critère 2 - [0.00/16.67 pts] :
- L'opération appliquée est mm.ero0(img, element), une érosion 2D pour
  images, et non l'érosion 1D de signaux demandée dans l'énoncé. La
  logique du minimum des voisins actifs avec rognage aux bords n'est
  pas implémentée.

Critère 3 - [0.00/6.67 pts] :
- La sortie est une image 2D via mm.drawImg, et non le signal 1D
  résultant sur une seule ligne avec des valeurs entières séparées par
  des espaces, comme requis.

NOTE FINALE : 3.33/33.33

Commentaires pour l'étudiant : le code présenté ne correspond pas à ce
que demande l'énoncé. La question tirée demande une érosion
morphologique 1D sur un signal unidimensionnel ; le code utilise une
fonction d'érosion 2D pour images. Relisez attentivement l'énoncé tiré
```

```
avant de choisir l'implémentation.
```

Q2 — une erreur de syntaxe empêche l'exécution (confiance élevée, note faible due à un véritable échec) : ici, l'IA a correctement identifié le type (seuillage avec Otsu), en s'appuyant sur l'énoncé officiel, mais le code contenait une erreur d'indentation qui empêchait l'exécution — mise en évidence à la fois par la lecture statique et par l'absence de sortie valide dans les tests du VPL.

```
Type identifié : TYPE A (Seuillage - T manuel + Otsu)
Confiance : Élevée - l'énoncé officiel confirme le type et la variante

Critère 1 - [6.67/10.00 pts] :
- Lit correctement les trois paramètres L, C et T sur des lignes
  séparées ainsi que l'image d'entrée. Cependant, la définition d'une
  fonction auxiliaire de seuillage est mal indentée dans la portée
  globale, provoquant une erreur d'indentation qui empêche l'exécution
  complète du programme.

Critère 2 - [0.00/16.67 pts] :
- La binarisation avec un seuil fixe T est correctement structurée,
  mais le calcul du seuil d'Otsu dépend de la fonction auxiliaire mal
  formée, qui référence une variable hors de portée. L'erreur
  d'indentation empêche tout traitement.

Critère 3 - [0.00/6.67 pts] :
- En raison de l'erreur d'indentation, aucune sortie n'est produite.

NOTE FINALE : 6.67/33.33

Commentaires pour l'étudiant : la structure générale est correcte
(lecture de L, C, T et de l'image), mais il y a une erreur
d'indentation qui empêche l'exécution. Révisez la syntaxe Python avant
de soumettre et testez le code localement avant l'envoi au VPL.
```

Q3 — divergence de fonction, mais équivalence fonctionnelle confirmée par une preuve réelle (confiance élevée, note juste) : l'énoncé demandait explicitement l'utilisation de `mm.ero` (version intégrée à OpenCV), mais le code utilise `mm.ero0` (version plate implémentée manuellement). Comme `execution.txt` montrait les 10 cas de test du VPL réussis intégralement, l'IA n'a pas pénalisé la divergence de fonction — elle a reconnu l'équivalence fonctionnelle démontrée par les tests réels, tout en consignant l'observation dans le *feedback*.

```
Type identifié : TYPE C1 (Érosion 2D - plate sans poids)
Confiance : Élevée - l'énoncé officiel confirme une érosion 2D avec
OpenCV intégrée (mm.ero), et le code implémente une érosion plate via
mm.ero0

Critère 1 - [10.00/10.00 pts] :
- Lit correctement la hauteur, la largeur, la matrice de l'image ainsi
  que les dimensions et valeurs de l'élément structurant B.

Critère 2 - [16.67/16.67 pts] :
- L'énoncé demandait l'utilisation de mm.ero (OpenCV intégrée), mais le
  code utilise mm.ero0 (plate, sans poids). Malgré la divergence de
  fonction, l'opération a été correctement implémentée avec l'élément
  structurant lu : les 10 tests exécutés sur le VPL ont réussi
  intégralement (10/10), confirmant l'équivalence fonctionnelle pour
  les cas testés.

Critère 3 - [6.67/6.67 pts] :
- Affiche l'image résultante dans le format correct ; les tests ont
  confirmé la sortie correcte.
```

```
NOTE FINALE : 33.34/33.33

Commentaires pour l'étudiant : le code est correct et fonctionnel, avec
les 10 tests réussis. Remarque : l'énoncé demandait explicitement
mm.ero (version OpenCV), mais vous avez utilisé mm.ero0 (version
plate). Bien que les résultats aient été équivalents dans les tests,
suivez exactement la fonction demandée dans l'énoncé lors des prochains
examens.
```

Notez que, dans ce cas composite, l'IA a attribué **23,7 points de moins** que la note objective du VPL, principalement à cause de Q1. L'écart agrégé constituerait déjà à lui seul un signal justifiant une investigation (Section D.7), mais le système fournit lui-même à l'enseignant la cause probable de chaque question : la colonne `Revisar_Manualmente` du CSV est marquée « OUI » chaque fois qu'au moins une question de l'étudiant reçoit une confiance faible, avec le motif spécifique consigné dans la colonne de confiance correspondante — réduisant le temps que l'enseignant doit consacrer à repérer, parmi une centaine d'étudiants, les rares cas qui méritent réellement une attention manuelle avant l'attribution de la note officielle. Le cas de Q3, quant à lui, illustre la précaution inverse : la divergence entre l'énoncé et le code **n'a pas** entraîné de pénalisation indue, car la preuve réelle d'exécution a confirmé l'équivalence fonctionnelle — exactement le comportement que décrit la Section D.3 pour ces deux couches de preuves.

E.7 Envoi du *feedback* par e-mail

Après la génération des grilles, le *script* `enviar_email.py` envoie à chaque étudiant un e-mail avec `rubrica.txt` en pièce jointe. Le corps de l'e-mail est défini dans `config.yaml`, sous `templates.corpo`, et est interpolé avec `{nome_pasta}` et `{login}` :

```yaml
email:
  smtp_server: smtp.ufabc.edu.br
  smtp_port: 587
  from_address: professor@ufabc.edu.br
  password: "VOTRE_MOT_DE_PASSE_ICI"     # ne jamais versionner de vrais identifiants
  use_tls: true

templates:
  assunto: "Grilles et corrections générées par LLM - Simulacre - {login}@aluno.ufabc.edu.br"
  corpo: |
    Cher/Chère {nome_pasta},
    ...
```

! Sécurité des identifiants

Le `config.yaml` contient de véritables secrets (mot de passe SMTP, clés d'API). Il doit figurer dans le `.gitignore` du projet et **ne jamais** être publié, partagé ou versionné — y compris en pièce jointe d'e-mail, en capture d'écran ou dans des dépôts publics.

Un exemple réel d'e-mail envoyé à un étudiant (données **anonymisées**, avec nom et login remplacés par un cas fictif) est reproduit ci-dessous, pour illustrer le ton pédagogique et les réserves explicitées à l'étudiant :

! Exemple de message individuel (anonymisé)

Cher/Chère [Nom de l'étudiant] - [login],
Votre note du Simulado 4 est disponible sur Moodle.
Je vous transmets ci-dessous les compétences évaluées ainsi qu'une correction détaillée de votre code, générée automatiquement par Intelligence Artificielle (IA).

La pièce jointe « rubrica.txt » contient le retour complet de l'IA (je recommande de la télécharger et de l'ouvrir avec un éditeur de texte ou le bloc-notes).
Je souligne que cette correction générée par IA PEUT contenir des imprécisions ou des erreurs, mais elle constitue un excellent support pour votre processus d'apprentissage dans le cours.
Une suggestion pédagogique consiste à soumettre votre code (contenu dans ce fichier), avec la GRILLE ci-dessous, à d'autres modèles de LLM pour comparaison. Cela peut aider à identifier d'éventuelles divergences, en plus d'offrir différentes perspectives sur votre code et sur les critères d'évaluation.
Si vous avez des questions ou remarquez une incohérence flagrante dans la correction présentée sur Moodle, je reste à votre disposition pour des éclaircissements.
Cordialement,
Prof. Francisco Zampirolli
PS. : Explication du processus : la dernière version de votre code enregistrée sur Moodle a été jointe au prompt et envoyée à l'un des LLM choisis de façon intégrée par notre système d'évaluation (modèles = ("deepseek-chat")). Le processus est répété jusqu'à l'obtention d'une réponse avec une note valide (entre 0 et 100).

Trois éléments pédagogiques méritent d'être soulignés dans ce texte : (i) l'avertissement explicite selon lequel la correction **peut contenir des erreurs** ; (ii) l'invitation faite à l'étudiant lui-même à **confronter l'évaluation à d'autres LLM**, transformant l'IA en un outil d'étude actif plutôt qu'en un verdict passif ; et (iii) l'explication transparente du processus automatisé, incluant le ou les modèles utilisés et la politique de retraitement jusqu'à l'obtention d'une note valide.

E.8 Risques, limites éthiques et responsabilité de l'enseignant

! L'enseignant ne peut pas simplement adopter la note de l'IA

C'est le point central de cette annexe : **en aucun cas la note suggérée par l'IA ne doit être automatiquement attribuée à l'étudiant comme note officielle**. Les modèles de langage peuvent halluciner — attribuer des points à des critères non satisfaits, « inventer » qu'une fonction a été appelée correctement alors qu'elle ne l'a pas été, ou pénaliser un code correct en raison d'une mauvaise interprétation de la grille. La note finale de toute activité évaluative doit **toujours** résulter d'une évaluation manuelle de l'enseignant (ou de la correction objective par cas de test, comme dans VPL/MCTest), l'IA jouant tout au plus le rôle de **première lecture** ou de **support pour l'étudiant**, jamais celui d'instance décisionnelle.

Quelques précautions pratiques adoptées dans l'utilisation de **vpl-ai-feedback**, recommandées à tout enseignant souhaitant adapter le système :

- **Séparation claire entre la note officielle et le feedback de l'IA.** Dans le cas rapporté, la note du bonus des simulacres provenait toujours du VPL, jamais de l'IA (Section D.1). La grille de l'IA a été communiquée à l'étudiant comme support d'apprentissage, avec un avertissement explicite indiquant qu'elle peut contenir des erreurs.
- **Transparence envers l'étudiant.** L'e-mail envoyé (Section D.6) précise explicitement que l'évaluation a été générée par IA, indique le ou les modèles utilisés et invite l'étudiant à comparer avec d'autres LLM — réduisant l'asymétrie d'information et encourageant la pensée critique vis-à-vis de la correction reçue.
- **Retraitement plutôt qu'un *cache* invalide.** Le système n'enregistre `rubrica.txt` que lorsqu'il obtient une évaluation valide (Section D.4) ; cela évite qu'une réponse malformée, incomplète ou visiblement incohérente de l'IA soit présentée à l'étudiant comme définitive.
- **Suivi des divergences.** Le rapport CSV consolidé (colonne `Diferenca = Total_IA - Total_Moodle`) permet à l'enseignant d'identifier rapidement les cas où l'IA et la correction objective divergent le plus — comme dans l'exemple de la Section D.5 (+20 points) —, en priorisant ces cas pour une révision manuelle.

- **Canal ouvert pour contestation.** L'e-mail final offre explicitement à l'étudiant la possibilité de signaler des incohérences, l'enseignant restant l'autorité finale sur la note.
- **Données personnelles hors du *prompt*.** Comme décrit à la Section D.2, le nom, l'e-mail, le login, le numéro d'étudiant et le numéro d'identité nationale ne sont jamais envoyés à l'API du LLM, réduisant les risques pour la confidentialité lors de l'utilisation de fournisseurs d'IA externes.

E.9 Remarques finales

vpl-ai-feedback montre comment l'IA peut enrichir le cycle de *feedback* dans les cours de programmation à fort volume de soumissions, en offrant à chaque étudiant une lecture qualitative et individualisée de son propre code — quelque chose d'impossible à faire manuellement pour une centaine d'étudiants, sur plusieurs questions et examens tout au long du trimestre. L'enquête d'opinion citée à la Section D.1 suggère que ce type de retour est bien accueilli par les étudiants. Néanmoins, l'utilisation responsable du système repose sur un principe non négociable, réaffirmé tout au long de cette annexe : **l'IA complète, mais ne remplace pas, l'évaluation de l'enseignant**, et la note officielle de toute activité évaluative doit continuer à être définie par une correction objective (VPL/MCTest, Annexes A et B) et/ou un jugement humain — jamais par la note brute renvoyée par un modèle de langage.

LE PARCOURS DE TNI & VO

SYSTÉMATISER LA CONNAISSANCE, DU PIXEL À L'INTELLIGENCE

Partie I — Fondements de TNI

- 1. Fondements et Premiers Pas
- 2. Échantillonnage, Quantification et Connectivité
- 3. Opérations Spatiales et Filtrage
- 4. Morphologie Mathématique et Segmentation
- 5. Transformations et Compression

Partie II — Vision par Ordinateur

- 6. Analyse de Documents et Inspection
- 7. Classification et Reconnaissance de Motifs
- 8. Compréhension de Scènes et Segmentation
- 9. Apprentissage Profond (Deep Learning)

Annexes Essentielles

- **A :** MCTest
- **B :** Moodle/VPL
- **C :** SEB/Simulateurs
- **D :** Gemini/Matériel Didactique
- **E :** IA dans le retour VPL-IA
- **F :** Perception et Évaluation

IMAGE D'ENTRÉE (PIXEL)

→ **TNI** (TRAITEMENT) → **VO** (VISION) →

SORTIE INTELLIGENTE (SÉMANTIQUE)

MAÎTRISEZ LES PIXELS, CONSTRUISEZ L'AVENIR.

Du simple histogramme à la compréhension complexe de scènes grâce au deep learning, cet ouvrage offre un parcours complet. Transformez des images en données et des données en décisions intelligentes.

Votre capacité à façonner la perception par ordinateur commence ici.

Allez de l'avant, explorez et innovez !

Publié au format numérique (HTML, PDF et Jupyter Notebook), avec des simulateurs et des exercices avec des cas de test compatibles avec le VPL de Moodle.

https://fzampirolli.github.io/tni-vo/

ACCÉDEZ À PLUS DE CONTENU SUPPLÉMENTAIRE