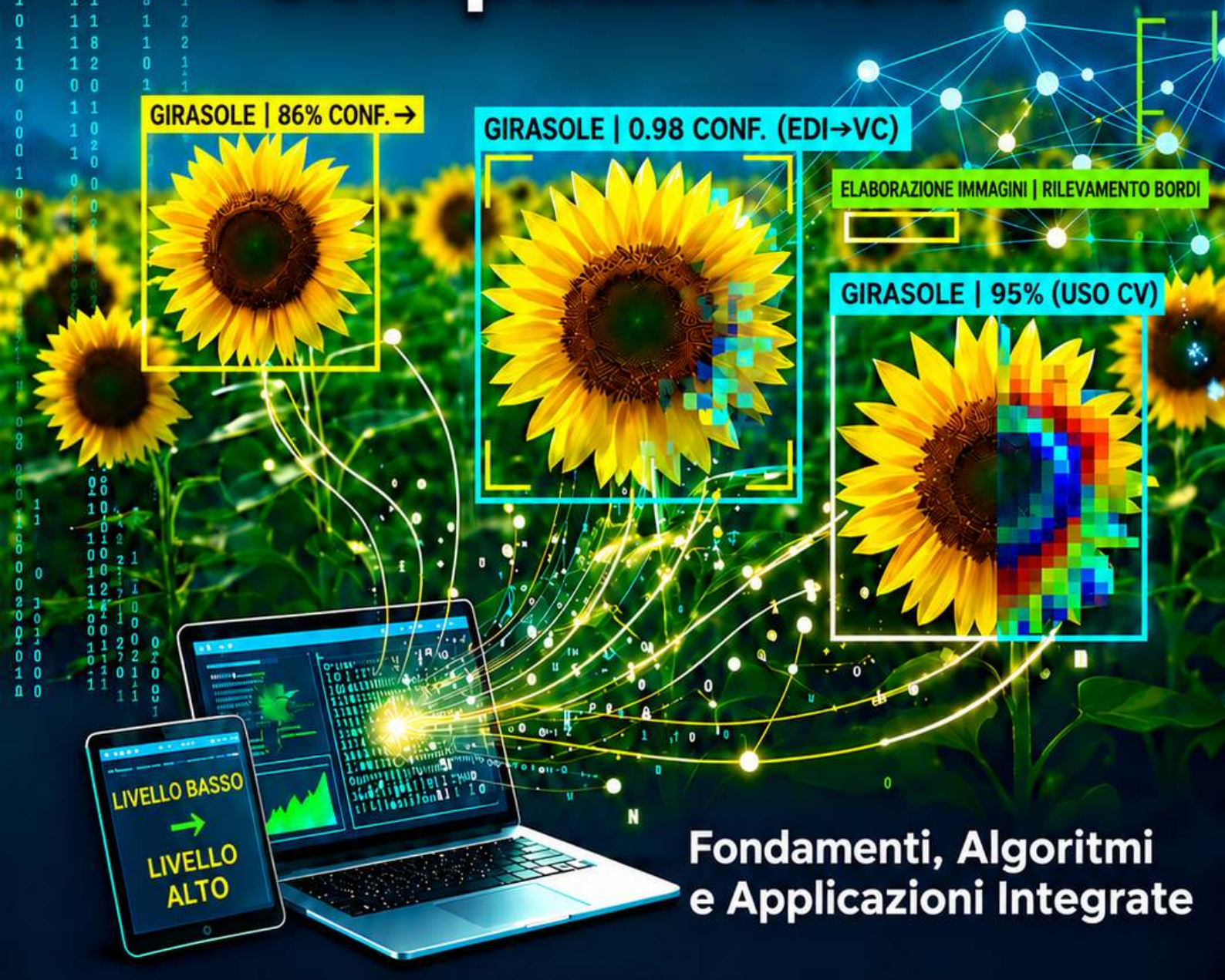
EDI & VC

Elaborazione Digitale delle Immagini e Visione Computazionale



Fondamenti, Algoritmi e Applicazioni Integrate

AUTORE: FRANCISCO DE ASSIS ZAMPIROLLI

Elaborazione Digitale delle Immagini e Visione Artificiale

Libro interattivo con C++

Francisco de Assis Zampirolli

Universidade Federal do ABC

20 settembre 2026

Projeto gráfico: Francisco de Assis Zampirolli
Capa: Francisco de Assis Zampirolli, com suporte de IA (Gemini e ChatGPT)

CATALOGAÇÃO NA FONTE
SISTEMA DE BIBLIOTECAS DA UNIVERSIDADE FEDERAL DO ABC

[CÓDIGO CUTTER] Zampirolli, Francisco de Assis
Processamento Digital de Imagens e Visão Computacional / Francisco de Assis Zampirolli – Santo André, SP : Edição do Autor, 2026.

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

ISBN: [ISBN A DEFINIR] (1ª Edição)

1. Processamento Digital de Imagens. 2. Visão Computacional. 3. Morfologia Matemática. 4. Python. 5. Correção Automática. 6. Moodle. I. Título.

CDD [XX] ed. – [CÓDIGO CDD]

Indice

Elenco delle Figure

Elenco delle Tabelle

EDI e VA — C++

Benvenuti nel libro EAI e Visione Artificiale — versione C++ / Italiano.

Organizzazione

- **Parte I — Fondamenti di EAI** — Rappresentazione, istogrammi, filtraggio, morfologia
- **Parte II — Visione Artificiale** — Segmentazione, descrittori, rilevamento, apprendimento profondo

---

Ficha Catalográfica

Prefazione

Questo libro costituisce un'**opera in costante aggiornamento** nei settori dell'**Elaborazione Digitale delle Immagini (EDI)** e della **Visione Artificiale (VA)**, concepita come materiale didattico interattivo per corsi di laurea triennale e magistrale in Informatica, Ingegneria e settori affini.

> **!** In evidenza
>
> L'opera si fonda sulla metodologia descritta in Zampirolli et al. (2025) — estensione di Zampirolli et al. (2024), lavoro premiato nel filone **Risorse e Ambienti Educativi** dell'**EduComp 2024**. Il contenuto integra la libreria `morph.py`, sviluppata dall'autore.

Questo non è un libro statico. Il suo contenuto evolve continuamente man mano che gli esempi vengono migliorati, nuove sezioni vengono incorporate e gli approcci pedagogici vengono affinati sulla base dell'esperienza d'uso e del riscontro di studenti e docenti. In tal modo, l'opera è trattata come un *progetto in costante evoluzione*, cercando di seguire sia i progressi tecnologici sia le migliori pratiche didattiche in EDI e VA.

Contesto della prima edizione

Il contenuto di questa edizione è stato elaborato tra maggio e agosto 2026, durante la prima offerta del corso di Elaborazione Digitale delle Immagini basata su questo materiale didattico. Il corso è stato tenuto in due classi di laurea triennale — composte prevalentemente da studenti di Informatica, nel periodo mattutino — e in una classe di laurea magistrale in Informatica, tutte presso l'UFABC. Questa prima edizione rappresenta il risultato di tale offerta iniziale, consolidando il contenuto sviluppato, testato e continuamente perfezionato nel corso del periodo accademico.

La metodologia di insegnamento adottata ha seguito una sequenza strutturata in ogni lezione. Inizialmente, veniva mostrato un video breve, della durata media di 7 minuti, generato in **Gemini Notebook**, alternando la presentazione della parte concettuale del capitolo e la risoluzione degli Esercizi di Programmazione (EP). In questo secondo caso, quasi tutti gli EP venivano risolti durante la stessa lezione. Successivamente, veniva presentato un insieme di circa 12 *slide*, anch'esse generate in **Gemini Notebook**, aventi come fonti il PDF completo del libro e il file `morph.py`. Queste *slide* venivano prodotte a partire da un *prompt* specifico per la generazione del contenuto teorico e pratico di ciascun capitolo.

Dopo questa fase iniziale, restavano circa 100 minuti di lezione per l'esplorazione dei due *notebook* Colab del capitolo, uno teorico e uno pratico, con enfasi sui simulatori interattivi e sull'esecuzione dei blocchi di codice, consentendo agli studenti di modificare i parametri e osservarne gli effetti. Nelle lezioni svolte in laboratorio, gli studenti trasferivano le risposte degli EP sviluppate in Colab direttamente nelle attività VPL del Moodle, dove venivano sottoposte alla valutazione automatica. Questo processo di pubblicazione e utilizzo degli EP è presentato alla fine di questa prefazione.

La valutazione continua includeva simulazioni quindicinali, somministrate con il **SEB (*Safe Exam Browser*)** in Moodle, contenenti EP simili a quelli delle liste di esercizi. Ogni simulazione concedeva un bonus del 5% sul voto finale. Anche le due prove del corso utilizzavano il SEB, ma erano composte da quesiti parametrizzati generati in **MCTest**, la cui descrizione combina LaTeX e parametri nel formato `[[code:variabile]]`, definiti in sezioni di Python delimitate da `[[def: ... ]]` nella stessa domanda. Questo processo ha permesso di generare 110 varianti di prova, sorteggiate individualmente tra gli studenti.

L'Appendice A descrive in dettaglio la creazione di tali domande in MCTest e la loro esportazione in Moodle;

l'Appendice B descrive la configurazione delle attività VPL, inclusi il processo di pubblicazione degli EP; l'Appendice C presenta la configurazione del SEB per le simulazioni e le prove; l'Appendice D descrive la generazione dei video e delle *slide* utilizzati nelle lezioni con **Gemini Notebook**; e l'Appendice E dettaglia l'uso dell'**Intelligenza Artificiale (IA)** nella generazione di *feedback* socratico per attività formative e valutative, integrando la correzione automatica effettuata dal VPL.

Come viene prodotto questo libro

Il contenuto di questo libro è sviluppato in **Quarto** ed è archiviato nella cartella `all` del repository **github.com/fzampirolli/pdi-vc**. A partire da un unico codice sorgente, il libro viene generato automaticamente e pubblicato in diversi formati, presentati nella Tabella 1.

Sebbene l'edizione in PDF registri lo stato dell'opera al termine della prima offerta del corso nel 2026, lo sviluppo del libro continua in modo permanente. A ogni aggiornamento del repository GitHub, vengono generate automaticamente nuove versioni delle pagine HTML, del PDF e dei notebook, mettendo immediatamente a disposizione dei lettori i miglioramenti.

Tabella 1: Diversi formati di pubblicazione generati automaticamente dallo stesso codice sorgente.

Formato	Descrizione
HTML	Versione per il *web*, con simulatori interattivi e navigazione tra i capitoli: fzampirolli.github.io/pdi-vc/
PDF	Versione adatta alla stampa o alla lettura *offline*: livro.pt.py.pdf
Notebook (`.ipynb`)	Compatibili con **Jupyter** e **Google Colab**, che consentono di eseguire, modificare e sperimentare gli esempi di codice e gli Esercizi di Programmazione (EP). Un filtro personalizzato mantiene i riferimenti incrociati, la numerazione di figure e tabelle, oltre a formattare automaticamente le citazioni secondo lo standard ABNT.

L'opera è pubblicata in **dieci combinazioni** — ogni percorso di codice (**Python** e **C++**) in cinque lingue (portoghese, inglese, francese, spagnolo e italiano). Con la cache di traduzione già popolata, una rigenerazione completa (traduzione, riesecuzione dei *notebook*, rendering di HTML e PDF e pubblicazione) richiede circa **72 minuti**, con un parallelismo reale medio di ~5 processi (picco di 8); la **prima** generazione di una nuova lingua, con la cache vuota, è molto più lenta — circa **80 minuti** per combinazione, come osservato nelle prime generazioni di spagnolo e italiano. La Tabella 2 riassume ogni combinazione (con la cache già popolata): numero di pagine del PDF e tempo di rendering. Questo tempo totale reale è molto inferiore alla somma dei tempi di ciascuna combinazione presa singolarmente (oltre **5 ore** se eseguite una alla volta): su una CPU con molti core, più combinazioni vengono elaborate contemporaneamente, quindi il tempo totale non è la semplice somma dei tempi individuali.

Tabella 2: Numero di pagine del PDF e tempo di rendering per combinazione (parallelismo reale medio di ~5 processi, picco di 8). La combinazione base in Python/Portoghese si limita a renderizzare le uscite già registrate nei *notebook* sorgente; le altre rieseguono tutto il codice.

Combinazione	Percorso	Lingua	Pagine	Tempo di rend.
`py.pt`	Python	Portoghese (base)	654	~7 min
`py.en`	Python	Inglese	644	~31 min
`py.fr`	Python	Francese	666	~31 min
`py.es`	Python	Spagnolo	666	~31 min
`py.it`	Python	Italiano	658	~31 min
`cpp.pt`	C++	Portoghese	434	~26 min
`cpp.en`	C++	Inglese	426	~34 min
`cpp.fr`	C++	Francese	434	~38 min

Combinazione	Percorso	Lingua	Pagine	Tempo di rend.
`cpp.es`	C++	Spagnolo	430	~37 min
`cpp.it`	C++	Italiano	428	~35 min

Stato di validazione. Solo la combinazione `py.pt` (Python, portoghese) è la fonte curata: è lì che l'autore scrive, revisiona e valida tutto il contenuto. Le altre nove combinazioni — i percorsi in **C++** e le traduzioni in inglese, francese, spagnolo e italiano — sono **generate automaticamente**, tramite traduzione con un modello linguistico e transpilazione del codice, e **richiedono ancora una revisione dettagliata**. Il punto più delicato è il **testo incorporato in alcune figure**: dove la versione tradotta non è ancora stata prodotta, l'immagine appare con testo in portoghese (ogni figura tradotta segue lo stesso suffisso di lingua degli altri file generati, ad esempio `immagine_en.png`).

Le pagine HTML, il PDF e i *notebook* disponibili nel progetto riflettono sempre lo stato più recente dello sviluppo. Quando viene pubblicata un'**edizione ufficiale**, essa viene identificata da un *tag* nel repository GitHub, consentendo di riprodurre esattamente il contenuto corrispondente a quell'edizione. In questo modo, il lettore può seguire sia l'evoluzione continua del progetto sia recuperare qualsiasi edizione ufficiale precedentemente pubblicata.

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

Ogni capitolo mette a disposizione due pulsanti **Esegui Colab** (Figura 1), che indirizzano ad ambienti complementari:

1. **Parte Teorica**, situata all'apertura di ogni capitolo, contenente i concetti presentati ed esempi di codice eseguibili;
2. **Parte Pratica**, nella sezione **Parte Pratica con Esercizi di Programmazione** (EP), dedicata agli esercizi e ai loro simulatori interattivi.

Figura 1: Pulsante cliccabile **Esegui Colab**, disponibile all'inizio delle parti **Teorica** e **Pratica** di ogni capitolo, che consente l'esecuzione interattiva degli esempi e degli esercizi. Nella versione PDF, esiste un *link* HTML diretto tra l'immagine del simulatore e la sua didascalia.

La generazione dei *notebook* è completamente automatizzata dallo *script* `gerar_notebooks_alunos.py`, che adatta il contenuto ad ambienti interattivi, consentendone l'esecuzione senza la necessità di installare Quarto.

Uso di strumenti di Intelligenza Artificiale

La concezione, il progetto pedagogico, la struttura concettuale e il contenuto fondamentale di questo libro sono di **esclusiva autorship dell'autore**.

Nel processo di redazione e supporto allo sviluppo, sono stati utilizzati, nelle loro versioni gratuite, strumenti di **Intelligenza Artificiale (IA)**, come **ChatGPT**, **Claude**, **DeepSeek** e **Gemini**, impiegati strettamente come risorse ausiliarie. Il loro uso si è limitato al supporto nella revisione e nel miglioramento dello stile testuale, all'ottimizzazione della sintassi dei codici e alla generazione assistita di illustrazioni concettuali ed esempi.

Tutte le risposte e i contenuti suggeriti da questi strumenti sono stati sottoposti a **analisi critica, verifica, validazione tecnica, adattamento e integrazione da parte dell'autore**, che si assume la responsabilità per il testo, i codici e le risorse pedagogiche presentate. Gli strumenti di IA **non sono considerati autori o coautori dell'opera**, né responsabili delle decisioni intellettuali, tecniche o pedagogiche che ne fondano il contenuto.

Si sottolinea, inoltre, che i **simulatori** e le risorse interattive presenti nel libro — disponibili nelle versioni HTML e Jupyter Notebook (IPYNB) — sono stati progettati e implementati come parte dell'approccio pedagogico dell'opera, con l'obiettivo di offrire una sperimentazione diretta dei concetti presentati e favorire l'autonomia di apprendimento del lettore.

Distinto dall'uso editoriale sopra descritto, il *pipeline* di generazione multilingua (vedi "Come questo libro è prodotto") impiega un modello linguistico come **componente ingegneristica del sistema**: la traduzione automatica del contenuto tra il portoghese e altre lingue, con cache incrementale che evita di ritradurre porzioni non modificate. Questa fase utilizza l'API a pagamento di **DeepSeek** (modello `deepseek-v4-flash`); il libro completo è già stato tradotto dal portoghese in **inglese**, **francese**, **spagnolo** e **italiano**, e i **Capitoli da 1 a 5** dispongono inoltre di un percorso in **C++** che compila ed esegue realmente — a un costo complessivo dell'ordine di pochi dollari, grazie alla cache incrementale.

La libreria `morph.hpp`

La libreria **`morph.hpp`** (Zampirolli et al., 2025) accompagna l'intera opera come strumento di supporto all'insegnamento. Il suo obiettivo non è sostituire librerie consolidate, come **OpenCV**, né competere con esse in prestazioni o ampiezza. Piuttosto, cerca di **rendere trasparenti gli algoritmi fondamentali di Elaborazione Digitale delle Immagini e Visione Computazionale (PDI-VC)**, consentendo allo studente di comprenderne l'implementazione, modificare il codice e sviluppare nuove funzionalità.

È l'equivalente in C++ della `morph.py`: *header-only* (basta un `#include "morph.hpp"`), con gli **stessi nomi di funzione** nel *namespace* `mm::` — chi già conosce `mm.dil()`, `mm.dil0()` o `mm.gradm()` in Python riconosce immediatamente `mm::dil()`, `mm::dil0()` e `mm::gradm()` in C++. Una divergenza di nome tra le due versioni è considerata un difetto della libreria, non una scelta di progetto.

> **i** Eredità Tecnica
>
> La struttura della `morph.hpp` (come quella della sua controparte `morph.py`) ha come base strumenti di Morfologia Matematica sviluppati in Brasile, come **MMachLib** (Lotufo et al., 1997) e **MMach** (Barrera et al., 1998), oltre alla **mmorph**, utilizzata nell'opera di Dougherty; Lotufo (2003). Queste librerie sono state un punto di riferimento per l'insegnamento del settore per decenni.

Questa filosofia può essere osservata, ad esempio, negli operatori morfologici di dilatazione:

- `mm::dil0`: implementazione didattica della dilatazione per **elementi strutturanti planari**, seguendo direttamente la definizione classica della Morfologia Matematica;
- `mm::dil1`: implementazione didattica della dilatazione per **funzioni strutturanti (*kernel* non planari)**, seguendo rigorosamente la loro formulazione matematica;
- `mm::dil`: per impostazione predefinita, delega a `mm::dil0()` — la versione didattica planare, numericamente equivalente a `cv::dilate` per questo tipo di elemento strutturante. L'implementazione ottimizzata, basata su **OpenCV** (`cv::dilate`), entra in gioco solo se il programma viene compilato con il flag `-DMM_USE_OPENCV`.

> **!** Una differenza deliberata rispetto a `morph.py`
>
> In Python, `mm.dil()` (senza suffisso) è **già** l'implementazione ottimizzata per impostazione predefinita. In C++, `mm::dil()` (stesso nome, stessa firma) diventa la versione ottimizzata solo se OpenCV viene collegato esplicitamente in fase di compilazione; senza di ciò — che è proprio il caso predefinito, incluso su Moodle/VPL — `mm::dil()` si comporta come `mm::dil0()`. Questa scelta evita che il percorso C++ dipenda da OpenCV solo per compilare ed eseguire un esercizio semplice: `g++ file.cpp -o file` è già sufficiente. Chi desidera la versione accelerata in locale può compilare con `g++ -DMM_USE_OPENCV file.cpp -o file $(pkg-config --cflags --libs opencv4)`.

La libreria offre anche funzioni per lettura, visualizzazione, conversione dei colori, filtraggio e morfologia matematica tramite un'interfaccia semplice:

```cpp
#include "morph.hpp"

int main() {
    mm::Image img = mm::gray(mm::read("lena.jpg"));  // lettura e conversione in scala di
grigi
    mm::Image grad = mm::gradm(img);                 // gradiente morfologico
    mm::show(grad, "output.png");                    // visualizzazione (scrive su file)
}
```

A differenza della versione Python, `mm::show()` richiede un percorso di output esplicito: ogni cella di codice C++ del libro viene eseguita come processo isolato (compilato ed eseguito tramite `%%writefile` + `!g++ ...` su Colab), senza il contatore globale delle figure che ha senso solo all'interno di un unico processo Jupyter.

Inoltre, tutti i progetti del **Gemini Notebook** del percorso C++ includono il file `morph.hpp`, consentendo di consultare e discutere direttamente l'implementazione degli algoritmi presentati nel libro. In questo modo, lo studente può utilizzare il Gemini Notebook stesso come assistente di studio, ponendo domande come:

> Spiega come è stata implementata la funzione `mm::dil0` di `morph.hpp`, mostrando il codice e commentando ogni fase in modo didattico. Spiega inoltre la differenza tra `mm::dil0()` e `mm::dil()` senza il flag `-DMM_USE_OPENCV`.

! Implementazioni didattiche e implementazioni ottimizzate

In diversi capitoli, lo stesso algoritmo viene presentato in due versioni. Le funzioni con suffisso `0`, `1`, ecc. (ad esempio, `mm::dil0()` e `mm::dil1()`) sono implementazioni didattiche, sviluppate per facilitare la comprensione degli algoritmi e disponibili indipendentemente dal flag di compilazione. Le funzioni senza suffisso (come `mm::dil()`), invece, utilizzano l'implementazione ottimizzata basata su OpenCV **solo** quando compilate con `-DMM_USE_OPENCV`; altrimenti si comportano come la versione didattica corrispondente.
Nelle attività valutate dal **VPL del Moodle**, la compilazione segue la modalità predefinita senza `-DMM_USE_OPENCV` — non per limiti di memoria (come accade con scikit-learn/scikit-image nel percorso Python), ma affinché nessun EP in C++ dipenda dalla presenza di OpenCV nell'ambiente di esecuzione. In pratica, ciò significa che, nel VPL, `mm::dil()` e `mm::dil0()` producono esattamente lo stesso risultato. In locale — ad esempio in **VS Code** o **Google Colab** — lo studente può scegliere di compilare con `-DMM_USE_OPENCV` per confrontare con la versione ottimizzata.

Il codice sorgente della libreria è disponibile presso:

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

Esercizi di Programmazione (EP) e Validazione Automatica

Ogni unità del libro include **Esercizi di Programmazione (EP)** pratici e di complessità crescente, progettati per consolidare i concetti presentati nel corso del capitolo. Ogni EP è accompagnato da un **simulatore interattivo**, disponibile nelle versioni HTML e IPYNB, che consente allo studente di manipolare parametri, visualizzare il comportamento degli algoritmi e sviluppare una comprensione intuitiva del problema prima di iniziare la sua implementazione. In questo modo, il processo di apprendimento combina sperimentazione, programmazione e validazione automatica.

La validazione delle soluzioni viene effettuata localmente dalla classe `TestSuite` (`testsuite.py`), che confronta l'output del programma con i file dei casi di test (`.cases`):

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

Il sistema supporta più linguaggi (Python, Java, C++, C, JavaScript e R) e può essere integrato direttamente in Moodle. Per i docenti interessati alla procedura completa passo passo di pubblicazione degli EP come attività VPL, si consulti la **Guida per il Docente**, alla fine di questa prefazione, e l'Appendice B.

Codice aperto

Il progetto è disciplinato da principi di codice aperto. Il repository pubblico riunisce il testo, i codici, le immagini e gli *script*: **github.com/fzampirolli/pdi-vc**

Ogni versione del libro è archiviata in modo permanente su **Zenodo** con un *DOI citabile*. Per citare questo materiale in lavori accademici:

> ZAMPIROLLI, Francisco de Assis. **PDI+VC — Elaborazione Digitale delle Immagini e Visione Artificiale**. UFABC, 2026. DOI: 10.5281/zenodo.20784605

Va altresì registrato, a proposito, che il contenuto esposto in quest'opera riflette lo sguardo critico, la proposta pedagogica e l'esperienza didattica del suo autore. Pertanto, le analisi, gli approcci e le opinioni espressi lungo questo libro rappresentano esclusivamente la comprensione del suo ideatore, non costituendo né riflettendo una posizione ufficiale o istituzionale dell'Università Federale dell'ABC (UFABC).

Prima di iniziare: *Notebook* in Python

Il concetto di ***Literate Programming*** (Programmazione Letterata), proposto da Donald Knuth (Knuth, 1984), costituisce il fondamento della struttura di questo materiale. La logica inverte il paradigma tradizionale: il programma è scritto per la lettura umana, assimilabile a un saggio, mentre il codice viene estratto separatamente per l'esecuzione computazionale.

Il contenuto è strutturato in *notebook* — documenti che intercalano **celle di testo** (in Markdown) e **celle di codice** (in Python).

- **Esecuzione:** le celle di codice sono identificate da `[ ]`. L'esecuzione può essere effettuata con `Shift + Enter` o tramite il pulsante ▷ dell'interfaccia.
- **Ambienti:** i *notebook* possono essere eseguiti localmente, tramite Jupyter o Visual Studio Code (VS Code), nonché in ambienti cloud, come Google Colab.

> **Nota sul formato**
>
> In ambienti interattivi, il codice può essere modificato ed eseguito. Nelle versioni statiche (HTML o PDF), i blocchi di codice hanno finalità di lettura e riferimento, senza pregiudicare l'integrità delle spiegazioni.

Guida per il Docente: Pubblicazione degli EP su Moodle

*Questa sezione è destinata ai docenti che desiderano integrare gli Esercizi di Programmazione (EP) nelle attività **VPL (Virtual Programming Lab)** di Moodle, integrando l'Appendice B.*

Integrazione con Moodle (VPL)

Gli EP dei *notebook* possono essere convertiti in attività **VPL** su Moodle. Lo *script* `ep_tools.py` (eseguito tramite `make build`) automatizza l'esportazione degli EP dalla fine di ogni capitolo in due fasi:

1. **Estrazione** (`make eps`): genera un HTML interattivo autonomo per ogni EP, con enunciato, esempi e simulatore, in `gen/book/eps/<versione>/`. Vedi l'elenco completo su: https://fzampirolli.github.io/pdi-vc/eps/py.pt/index.html

2. **Conversione** (`make moodle`): trasforma l'HTML in un frammento autocontenuto, senza dipendenza da CSS esterni, pronto per essere incollato nell'editor di Moodle, in `gen/book/eps/<versione>_moodle/`. Vedi un esempio su: https://fzampirolli.github.io/pdi-vc/eps/py.pt_moodle/EP01_01.html

Inoltre, tutti i simulatori interattivi sviluppati nel corso del libro possono essere consultati direttamente su https://fzampirolli.github.io/pdi-vc/simuladores/py.pt/.

Altri percorsi e lingue

I link sopra usano `py.pt` come esempio. Per accedere agli EP o ai simulatori di un'altra combinazione linguaggio/lingua, basta sostituire quel segmento nell'URL — ad esempio, `cpp.it` per il percorso C++ in italiano: `https://fzampirolli.github.io/pdi-vc/eps/cpp.it/index.html`. La struttura è la stessa per tutte le combinazioni disponibili, in `/eps/<versione>/`, `/eps/<versione>_moodle/` e `/simuladores/<versione>/`.

Pubblicando con `make publish`, queste **due versioni** di ogni EP sono disponibili nei *link* indicati. Il docente può combinare le due strategie: incollare la versione Moodle nell'editor VPL (con simulatore funzionante) e includere nell'enunciato un *link* alla versione completa, dove le formule vengono renderizzate correttamente da MathJax.

Revisione obbligatoria prima della pubblicazione su Moodle

Sebbene automatizzata, la conversione richiede una revisione da parte del docente a causa delle limitazioni dell'editor TinyMCE/HTML Purifier di Moodle:

- **Formule matematiche** — TinyMCE elimina le barre rovesciate (\), corrompendo le equazioni LaTeX. Per questo motivo, la versione Moodle converte le formule semplici in HTML puro (entità come `&ge;`, `&times;`). Le formule complesse (`\begin{cases}`, matrici, integrali) possono risultare incomplete — verifica e, se necessario, riscrivile in testo o indica la versione completa tramite *link*.
- **Figure esterne** — Le immagini complementari devono essere inserite manualmente nell'attività VPL.
- **Simulatori** — Funzionano perfettamente purché utilizzino JavaScript standard con stili *inline* e manipolazione diretta del DOM (`getElementById`). **Attenzione:** se si includono formule in LaTeX contenenti il carattere \ in Moodle, i simulatori potrebbero smettere di funzionare.

Come Pubblicare su Moodle

1. Accedi alla versione Moodle dell'EP desiderato:

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

2. Copia il codice sorgente completo della pagina (`Ctrl+U` → `Ctrl+A` → `Ctrl+C`).
3. Su Moodle, crea l'attività VPL, accedi all'editor HTML, incolla il contenuto (`Ctrl+V`) e salva.
4. Importa i file `.cases` dalla cartella `all/capXX/casos/` nelle impostazioni dei test VPL.

Riutilizzo dei Casi di Test

I file `.cases` sono compatibili con VPL, consentendo di utilizzare lo stesso set di test sia nello sviluppo locale che nella correzione automatizzata su Moodle.

Parte I

Parte I — Fondamenti di EAI

Capitolo 1

Fondamenti e Primi Passi

Questo capitolo apre la **Parte 1** del libro, dedicata ai fondamenti dell'Elaborazione Digitale delle Immagini (EDI). Vengono presentati la rappresentazione matematica delle immagini digitali e i principali metodi per la loro manipolazione.

Si utilizza il linguaggio C++ e la libreria `morph.hpp`, una porta didattica della `morph.py` (ZAMPIROLLI, 2025).

1.1 Obiettivi

Al termine di questo capitolo, sarai in grado di:

- Comprendere la natura fisica e matematica dell'immagine digitale $f(x, y)$.
- Identificare le bande dello spettro elettromagnetico rilevanti per l'elaborazione digitale delle immagini (PDI).
- Eseguire operazioni di base: lettura, visualizzazione e salvataggio delle immagini.
- Accedere e modificare le intensità dei pixel singolarmente.
- Applicare la sogliatura manuale.
- Configurare l'ambiente di sviluppo in C++.
- Gestire le strutture di array (`std::vector`) senza cadere in insidie di copia/riferimento.

1.2 Prima di iniziare: *Notebooks* interattivi

Questo materiale è stato realizzato secondo il concetto di ***Literate Programming*** (Programmazione Letteraria), ideato da Donald Knuth negli anni '80 (KNUTH, 1984). Knuth — anche creatore del sistema **TeX** per la tipografia digitale — propose che i programmi fossero scritti come una narrazione logica per gli esseri umani, alternando codice e documentazione.

Per eseguire una cella, premere Shift + Invio oppure fare clic sul pulsante ▷.

> **Nota sul formato**
>
> Nelle versioni renderizzate (PDF o HTML), il codice è presentato in blocchi statici a scopo di lettura e riferimento. L'esecuzione interattiva richiede l'accesso tramite Google Colab (disponibile in cima alla pagina) o in ambiente locale con VSCode o Jupyter Notebook.

1.3 Fondamenti

Lo studio dei sistemi basati su immagini comprende un ecosistema di discipline integrate che trasformano dati visivi grezzi in conoscenza strutturata. Mentre alcune aree si concentrano sulla generazione di rappresentazioni, altre si dedicano al trattamento e all'analisi di tali dati per supportare applicazioni tecnologiche complesse.

Il diagramma presentato in Figura 1.1 stabilisce la distinzione e la complementarità tra l'Elaborazione Digitale delle Immagini (EDI) e la Visione Artificiale (VA). L'EDI, evidenziata in **verde**, si concentra sulla trasformazione da immagine a immagine, mirando al miglioramento della qualità o alla pre-elaborazione, come la rimozione del rumore e l'accentuazione del contrasto.

Al contrario, la VA, contrassegnata in **blu**, si concentra sull'interpretazione del contenuto visivo per estrarre modelli o informazioni, come il riconoscimento di oggetti e gesti. La regione di intersezione illustra la sinergia tra le aree, dove l'EDI prepara i dati visivi per l'interpretazione da parte della VA. La mappa dimostra inoltre le interconnessioni di entrambe le discipline con aree quali la Robotica, la Grafica Computerizzata, l'Intelligenza Artificiale (IA) e le Neuroscienze.

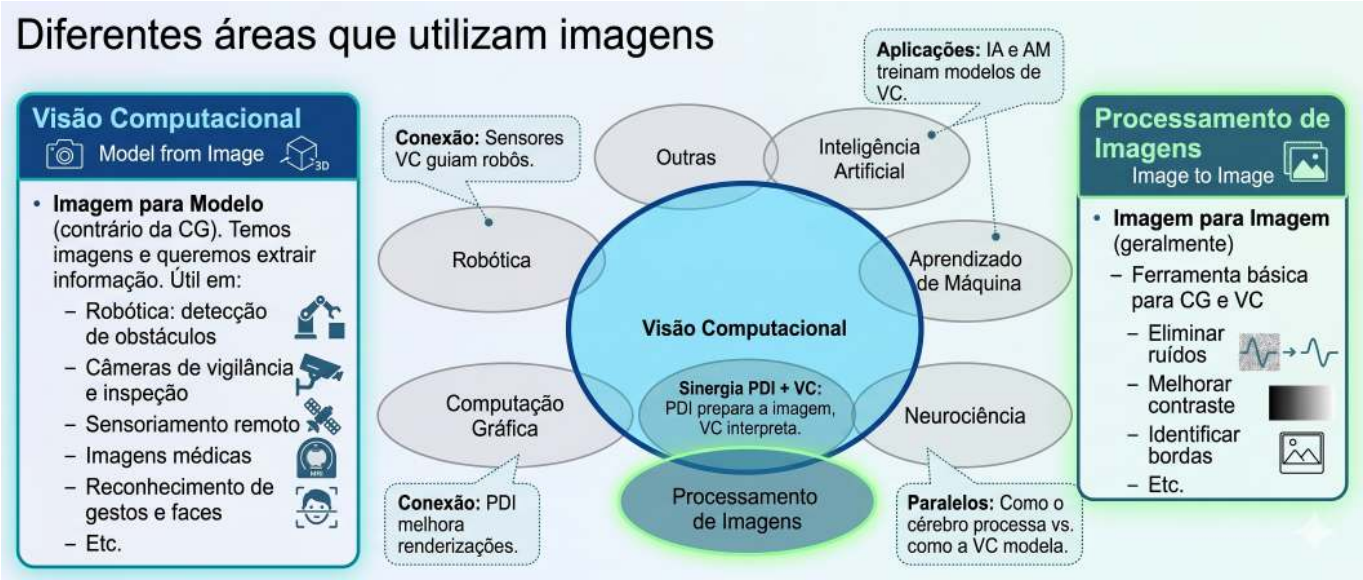


Figura 1.1: Diagramma relazionale che illustra le distinzioni fondamentali, le sinergie e le interconnessioni tra EDI e VA nel contesto dei sistemi basati su immagini.

1.3.1 👁 Visione Computazionale

- **Focus:** *Immagine* → *Modello* (percorso inverso rispetto alla Computer Grafica).
- **Obiettivo:** Estrarre informazioni di alto livello da immagini o video.
- **Applicazioni tipiche:**
  - Robotica: rilevamento di ostacoli, localizzazione e navigazione autonoma.
  - Sorveglianza e ispezione: riconoscimento di eventi, lettura di targhe, controllo qualità.
  - Telerilevamento: analisi di immagini satellitari, mappatura ambientale.
  - Imaging medico: rilevamento di tumori, segmentazione di organi, supporto alla diagnosi.
  - Interazione uomo-computer: riconoscimento di gesti, espressioni facciali, eye tracking.
- **Relazione con altri ambiti:** utilizza tecniche di Apprendimento Automatico e IA per classificare e interpretare le scene; funge da "occhi" della Robotica.

1.3.2 🖼 Elaborazione Digitale delle Immagini (PDI)

- **Obiettivo:** *Immagine* → *Immagine* (generalmente - trasformazione di un'immagine in un'altra).
- **Scopo:** Migliorare la qualità visiva o estrarre caratteristiche di basso livello.

- **Applicazioni comuni:**
  - Rimozione del rumore (filtri media, mediana, gaussiano).
  - Miglioramento del contrasto (equalizzazione dell'istogramma, regolazione gamma).
  - Rilevamento dei bordi (Sobel, Canny, Laplaciano).
  - Segmentazione (sogliatura, crescita delle regioni, *watershed*).
  - Trasformazioni geometriche (ridimensionamento, rotazione, correzione prospettica).
- **Relazione con altre aree (vedi Tabella 1.1):**
  - È la **base** per la maggior parte dei sistemi di visione artificiale (pre-elaborazione).
  - La Computer Grafica applica spesso PDI per la post-elaborazione (es.: smussatura, miglioramento).
  - Le tecniche di IA possono ottimizzare i parametri di elaborazione (es.: apprendimento di filtri).

Tabella 1.1: Connessione tra PDI, VC e altre aree della scienza.

Área	Relação com PDI e VC
Intelligenza Artificiale	Fornisce modelli (reti neurali, SVM) che interpretano le uscite della VC.
Robotica	Consuma dati di VC per prendere decisioni (navigazione, manipolazione).
Apprendimento Automatico	Utilizza descrittori estratti da PDI/VC per addestrare classificatori.
Computer Grafica	Percorso inverso: *modello* → *immagine*; spesso applica PDI per una resa realistica.
Neuroscienze	Ispira modelli di PDI (es.: filtri simili alle cellule gangliari della retina).

1.4 Fasi del PDI

Le fasi del PDI sono presentate nella Figura 1.2 e possono essere interpretate come una catena di trasformazioni che riduce la ridondanza dei dati per ricercarne il significato:

- **Basso Livello:** opera direttamente sui pixel dell'immagine disturbata per apportare miglioramenti e filtraggi, producendo come output un'immagine pulita o esaltata.
- **Medio Livello:** riceve l'immagine trattata ed esegue la segmentazione e la descrizione, trasformando la matrice dei pixel in attributi strutturati (forma, dimensione e texture).
- **Alto Livello:** utilizza la tabella degli attributi per alimentare processi di logica e intelligenza artificiale, portando alla decisione o al riconoscimento finale (come la diagnosi medica).

La Figura 1.3 descrive l'intera sequenza dell'elaborazione delle immagini digitali (PDI), dall'acquisizione all'interpretazione. Il flusso inizia con l'Acquisizione dell'Immagine (1) e prosegue con il Miglioramento (2) e il Restauro (3). Successivamente, il contenuto viene isolato tramite la Segmentazione (4) e raffinato mediante la Morfologia (5). La transizione cruciale avviene nella Rappresentazione e Descrizione (6), dove gli oggetti visivi vengono convertiti in dati matematici (area, perimetro, ecc.), consentendo il Riconoscimento (7). I processi ausiliari includono l'Elaborazione dell'Immagine a Colori e la Compressione, che contribuiscono all'efficienza dell'archiviazione e dell'analisi.

1.5 Formazione dell'Immagine e Spettro

Il processo di formazione di un'immagine si basa sull'interazione tra materia ed energia radiante. Essenzialmente, un'immagine viene concepita quando un **sensore registra la radiazione** risultante dall'interazione con un oggetto fisico. Nel contesto della visione umana e della fotografia convenzionale, questo fenomeno dipende da una sorgente luminosa che illumina la scena; le caratteristiche degli oggetti vengono quindi codificate attraverso le **variazioni di intensità e colore** della luce che raggiunge il sensore, come illustrato nella Figura 1.4.

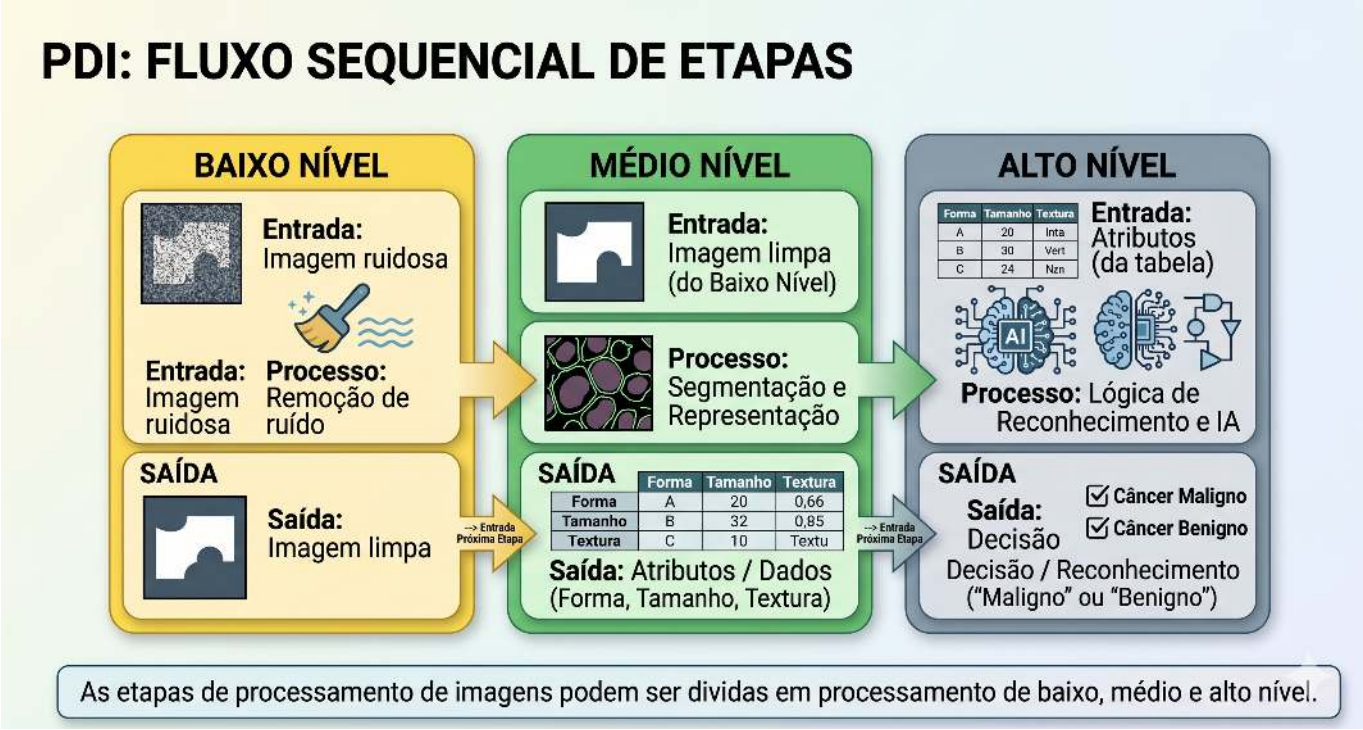


Figura 1.2: Rappresentazione del flusso sequenziale di elaborazione: l'output di ciascun livello diventa l'input del livello successivo.

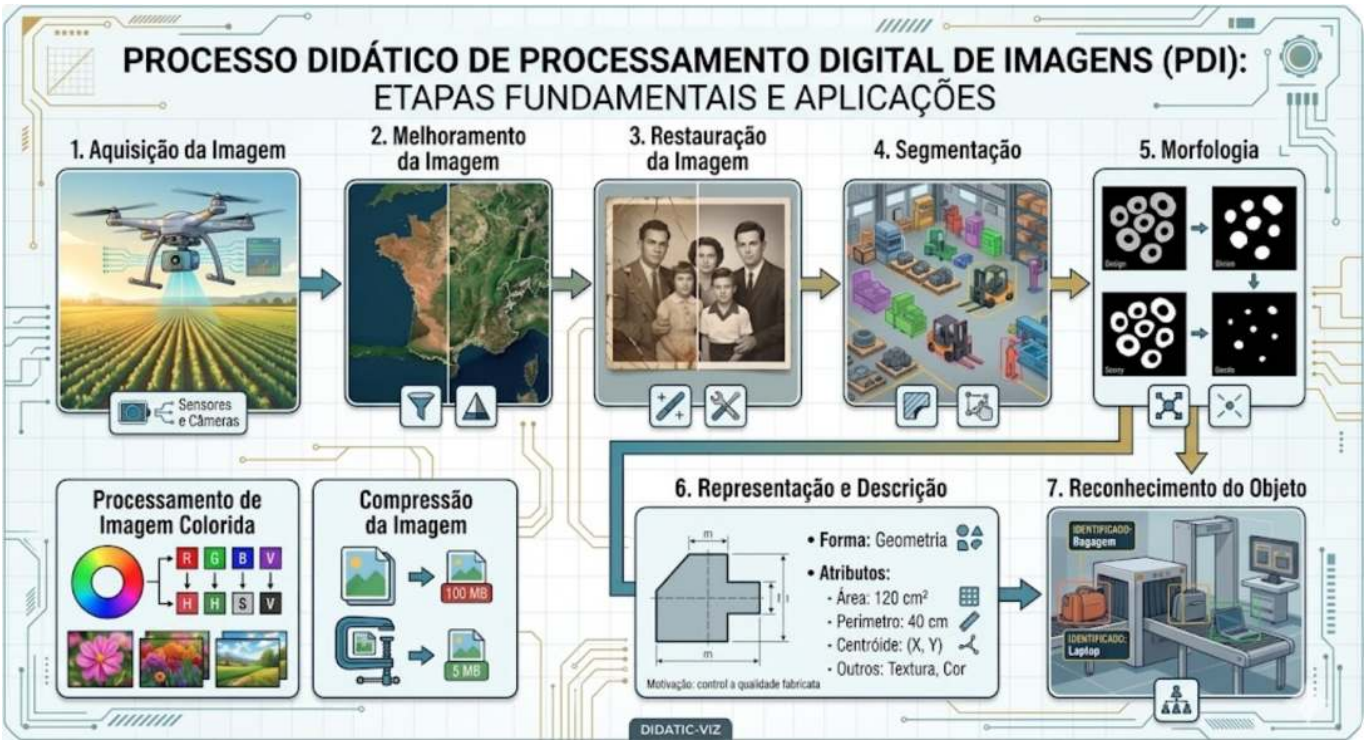


Figura 1.3: Flusso dettagliato del PDI: dall'acquisizione sensoriale all'estrazione di attributi e al riconoscimento automatizzato, includendo l'elaborazione a colori e la compressione.

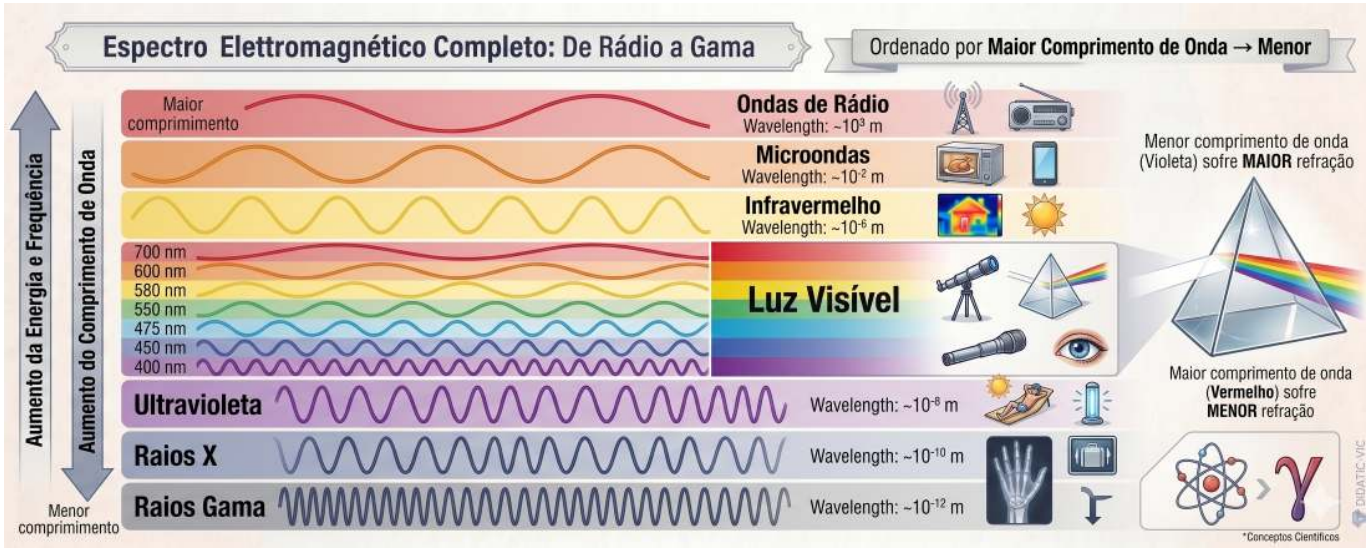


Figura 1.4: Rappresentazione dello spettro visibile e della sua posizione rispetto alle altre radiazioni elettromagnetiche, evidenziando la variazione delle lunghezze d'onda da 380 nm a 750 nm.

La **luce visibile** occupa solo una piccola banda dello spettro elettromagnetico — tra **380 nm (viola)** e **750 nm (rosso)** — come illustrato nella Figura 1.5. I sensori digitali convenzionali operano in questa stessa finestra, ma apparecchiature specializzate possono captare radiazioni invisibili all'occhio umano, come l'infrarosso e i raggi X. In PDI, l'immagine formata dipende direttamente dalla sensibilità spettrale del sensore utilizzato.

A partire da questo processo fisico di acquisizione, diventa possibile modellare matematicamente l'immagine digitale come una funzione bidimensionale discreta, nella quale ogni punto della scena è rappresentato da campioni numerici di intensità luminosa, formalizzando così i concetti di pixel e di immagine digitale presentati nella sezione successiva.

1.6 Cos'è un'Immagine Digitale?

Un'**immagine digitale** è formata da una griglia di **pixel** (*Picture Elements*), dove ogni pixel è la più piccola unità elementare dell'immagine.

> **Cos'è un Pixel?**
>
> Un **pixel** è la più piccola unità indirizzabile che compone un'immagine digitale. Ogni pixel occupa una posizione unica nella griglia e memorizza uno o più valori numerici che rappresentano la sua intensità o colore.

Rappresentazione Matematica

A differenza di una funzione continua, il dominio di un'immagine digitale è un piano rettangolare finito $\mathbb{E} \subset \mathbb{Z}^2$, che rappresenta la griglia di campionamento. Questo dominio è indicizzato da coordinate intere:

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

Dove:

- L: rappresenta la larghezza dell'immagine (numero di colonne).
- H: rappresenta l'altezza dell'immagine (numero di righe).

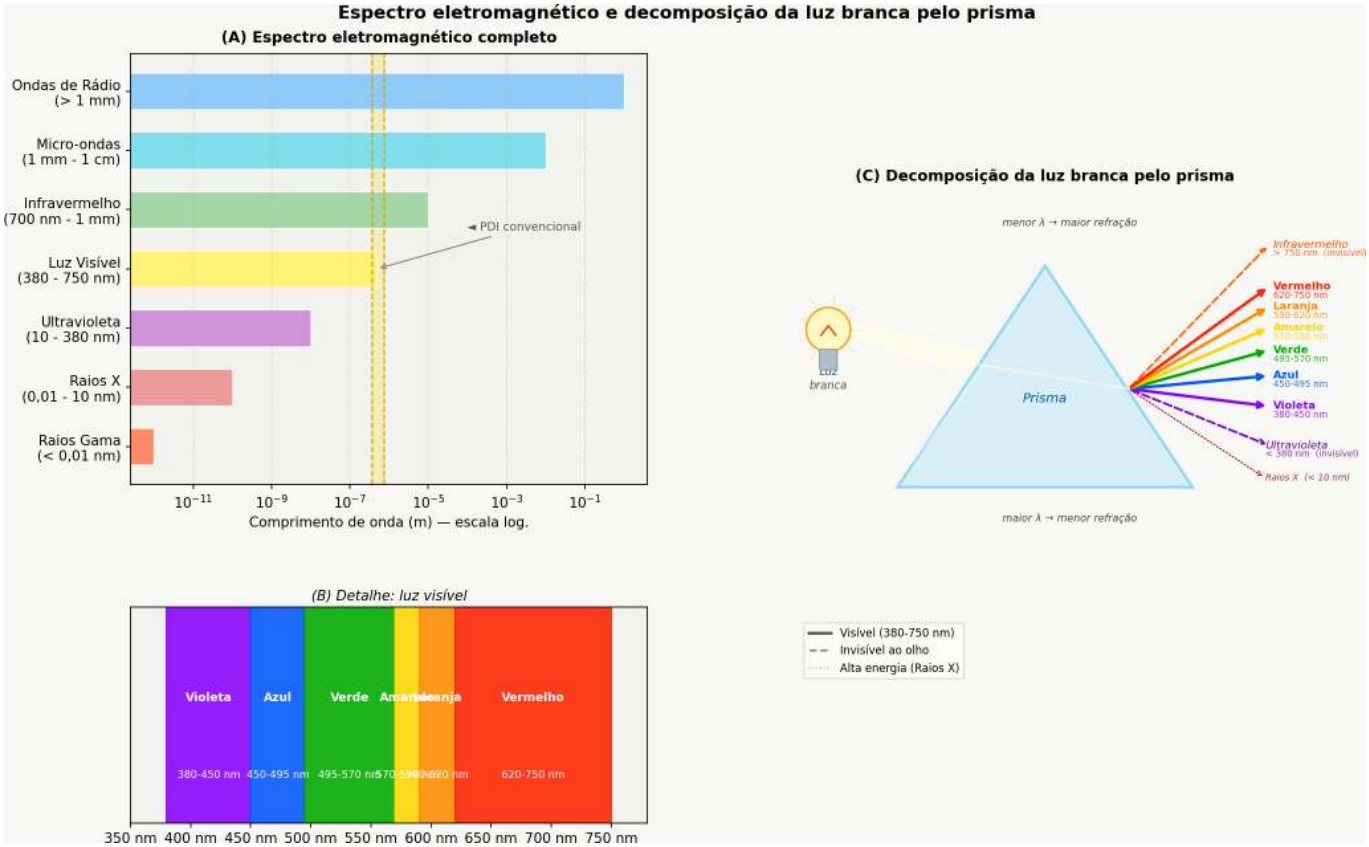


Figura 1.5: (A) Spettro elettromagnetico completo in scala logaritmica, con evidenza della fascia visibile. (B) Dettaglio della luce visibile (380-750 nm) e dei suoi colori. (C) Scomposizione della luce bianca tramite prisma: la lunghezza d'onda minore subisce una rifrazione maggiore, separando UV, visibile e infrarosso.

L'immagine digitale è una funzione che associa a ogni coppia di coordinate (x, y) uno o più valori che descrivono l'aspetto del pixel.

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

L'insieme $\mathcal{V}$ definisce i valori possibili per il pixel (codominio), variando a seconda del tipo di immagine, come dimostrato nella Tabella 1.2.

Tabella 1.2: Principali tipi di immagine digitale e i rispettivi insiemi di valori possibili per ogni pixel.

Tipo di immagine	$\mathcal{V}$ (valori del pixel)	Rappresentazione
Binaria	$\{0, 1\}$ o $\{0, 255\}$	■□
Scala di grigi	$\{0, 1, \ldots, 255\}$	`[ ][=][#]`■
A colori (RGB)	$\{0, \ldots, 255\}^3$ (terne ordinate di valori)	■■■

Esempio pratico: Un'immagine a colori nel modello RGB può essere rappresentata matematicamente da una funzione che associa tre valori di intensità a ogni pixel. Computazionalmente, questa rappresentazione corrisponde a tre matrici sovrapposte — i canali rosso (*Red*), verde (*Green*) e blu (*Blue*) — nelle quali ogni elemento memorizza l'intensità luminosa del rispettivo canale in una determinata posizione dell'immagine.

Per rendere possibili gli esperimenti pratici di PDI e VC, questo libro utilizza C++17 e un insieme minimo di librerie orientate alla manipolazione matriciale, all'elaborazione delle immagini e alla visualizzazione dei risultati, presentate di seguito.

1.7 Configurazione dell'Ambiente

Questo materiale utilizza **C++17** e la libreria a singolo header **morph.hpp**, originariamente proposta per Python in Zampirolli (2025). Qui viene utilizzata una versione minima e didattica della `morph.py`, adattata per una compilazione semplice con `g++`, come presentato in Tabella 1.3.

Ogni cella di codice viene compilata ed eseguita come un processo isolato (`%%writefile file.cpp` seguito da `g++`). Diversamente dal *kernel* Python, non esiste uno stato persistente tra le celle: le immagini prodotte da una cella vengono salvate in file per essere lette dalla successiva.

Gli **Esercizi di Programmazione (EP)** presentati alla fine dei capitoli possono essere validati dallo stesso modulo `testsuite.py` utilizzato nel percorso Python, che compila la soluzione con `g++` e confronta l'output con i casi di test. In questo modo, gli stessi casi di test (`.cases`) possono validare soluzioni in C++, Python e altri linguaggi, sia nei *notebook* che in **Moodle/VPL**.

Tabella 1.3: Principali librerie e strumenti utilizzati nel percorso C++ di questo libro.

Libreria / Strumento	Funzione principale
`g++` (C++17)	Compilazione di ogni cella di codice
`morph.hpp`	Astrazione didattica delle operazioni di PDI
`stb_image.h` / `stb_image_write.h`	Lettura/scrittura di PNG/JPEG (senza OpenCV)
`testsuite.py`	Esecuzione e validazione automatica degli EP

1.8 Informazioni sulla `morph.hpp`

La `morph.hpp` è una versione C++ minima e didattica della `morph.py`: implementa le funzioni utilizzate in questo capitolo (`read`, `gray`, `randomImage`, `show`, `write`, `threshold`, `drawImg`), oltre alle operazioni di

morfologia (`dil/ero` e varianti `dil0/ero0/dil1/ero1`) preparate per i capitoli successivi. Non è garantita una parità numerica con l'implementazione Python — il criterio è "compila e produce un'immagine plausibile", non "risultato bit-a-bit identico a Python".

Le librerie `stb_image.h/stb_image_write.h` (dominio pubblico/MIT) sono **vendorizzate** insieme al repository — cioè, il loro codice sorgente è già copiato all'interno del progetto stesso, invece di essere installato separatamente tramite `apt install` — evitando così di dipendere dai pacchetti di sistema durante la compilazione su Colab. Le operazioni `dil()/ero()` utilizzano `cv::dilate/cv::erode` quando compilate con `-DMM_USE_OPENCV`; senza questa *flag* (predefinita, anche su Moodle/VPL), si ricade sulla versione didattica equivalente (`dil1/ero1`) senza richiedere OpenCV.

```
import os, urllib.request

os.makedirs("tmp/state", exist_ok=True)  # artefatti di build della pista C++ (.cpp, binario,
PNG)

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")

# Il kernel è comunque Python anche nella pista C++: `mm` (morph.py) è usato dai
# simulatori, dalla visualizzazione delle figure che il binario C++ genera e dallo
# stato mm::Image tra le celle. cpp=True scarica anche la pista compilata
# (morph.hpp + stb_image*.h), usata nell'#include delle celle %%writefile *.cpp.
import config
config.setup(testsuite=True, cpp=True)
from morph import mm
from testsuite import TestSuite
```

```
Ambiente pronto. Morph: 1.1.9 | OpenCV: 5.0.0 | TestSuite: 1.1.2
```

1.9 Fondamenti delle Matrici — attenzione alla copia dei riferimenti

Poiché un'immagine digitale può essere rappresentata da una matrice, è importante comprendere come creare e manipolare correttamente le matrici. In C++, `std::vector` ha **semantica di valore**: copiare il vettore copia i suoi dati. Pertanto, `std::vector<std::vector<int>> m(3, std::vector<int>(2, 0))` crea effettivamente tre righe **indipendenti** — il costruttore di riempimento copia la riga-modello tre volte.

> ⚠ Attenzione alla copia dei riferimenti
>
> La trappola compare quando si usano **puntatori** o **riferimenti** invece di copie. In `std::vector<int> riga(2, 0); std::vector<std::vector<int>*> m(3, &riga);`, i tre elementi di `m` puntano alla **stessa** riga, e modificare `(*m[0])[0]` modifica tutte. Lo stesso accade con `auto& riga = m[0];` (riferimento — modifica la matrice), in contrasto con `auto riga = m[0];` (copia — lascia la matrice intatta).

Per visualizzare questo comportamento, si può eseguire il codice su Python Tutor (che esegue anche C++ passo dopo passo) e confrontare l'effetto di copie e di riferimenti.

Nella pratica, per l'elaborazione digitale delle immagini (PDI), si utilizza la struttura `mm::Image` della `morph.hpp`, che ha anch'essa semantica di valore: `mm::Image b = a;` copia i pixel, mentre `mm::Image& b = a;` crea solo un alias per la stessa immagine. Il codice seguente presenta diverse modalità di creazione di immagini sintetiche, i cui risultati sono mostrati nella Figura 1.6..

```
%%writefile tmp/fig_imagens_sinteticas.cpp
#define MM_OUT "tmp/fig_imagens_sinteticas.png"
```

```
// Compile: g++ -std=c++17 -o programa programa.cpp -I. -lpng -ljpeg
#include "morph.hpp"
#include <iostream>
#include <filesystem>

//| quarto-raw: false
//| label: fig-imagens-sinteticas
//| fig-cap: "Exemplos de imagens sintéticas representadas matricialmente."
//| echo: true
//| output: true

int main() {
    // Criando uma imagem preta (zeros) de 4, 6 pixels
    mm::Image img_preta(4, 6);
    img_preta.at(0, 0) = 255;  // pixel branco no canto superior esquerdo

    // Criando uma imagem branca (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 preto no canto inferior direito

    // Criando uma imagem aleatória para testes (ruído)
    mm::Image img_random = mm::randomImage(4, 6, 255);

    std::cout << "Matriz aleatória gerada:\n";
    std::cout << mm::drawImg(img_random) << "\n";

    mm::show(
        std::vector<mm::Image>{img_preta, img_branca, img_random},
        MM_OUT,
        std::vector<std::string>{
            "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)"
        },
        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"
```

```
Matriz aleatória gerada:
147 128 202 237 219 146
115 102  81 242 109  14
142 155 193  47  65  78
190 103 180 176 143  43
```

```
[1] Predominantemente preta
(com 1 pixel branco em (0,0))
[2] Predominantemente branca
(com 1 pixel preto em (3,5))
[3] Imagem aleatória
(simulação de ruído)
```

```python
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)")
```

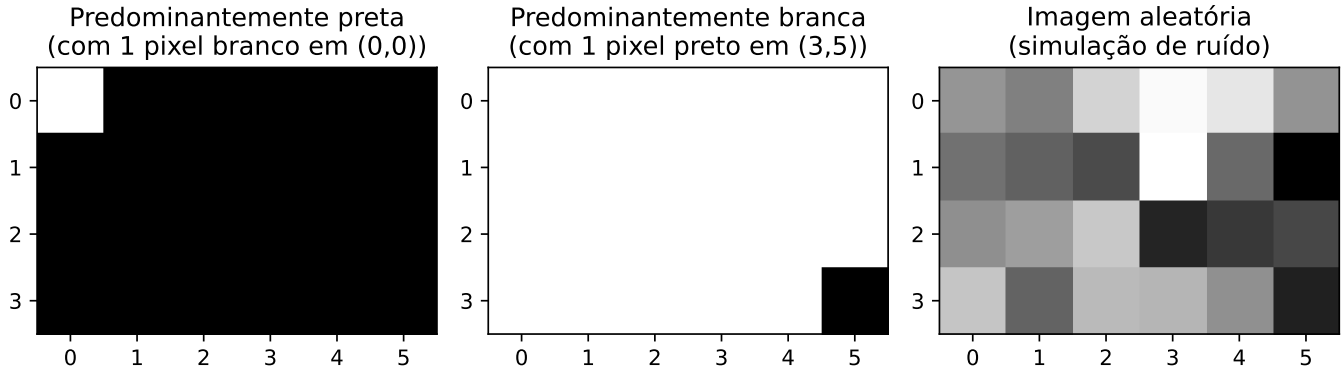


Figura 1.6: Exemplos de imagens sintéticas representadas matricialmente.

1.10 Lettura e Visualizzazione delle Immagini

In biblioteche come OpenCV (`cv::Mat`), le immagini digitali sono rappresentate computazionalmente come matrici multidimensionali. In `morph.hpp`, la struttura `mm::Image` adotta un approccio più semplice: un buffer lineare di byte (`std::vector<unsigned char>`), con altezza, larghezza e numero di canali espliciti.

Una delle operazioni fondamentali nell'elaborazione delle immagini digitali è la lettura delle immagini.

In `morph.hpp`, la funzione `mm::read()` consente di caricare immagini sia da file locali che da URL, come illustrato nella Figura 2.7..

```
%%writefile tmp/fig_01_natureza.cpp
#define MM_OUT "tmp/fig_01_natureza.png"
//| label: fig-01-natureza
```

```
//| fig-cap: "Mandrill (Mandrillus sphinx) em ambiente natural. Crédito: Julien Renoult (CC BY
4.0)."
//| echo: true

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

int main() {
    mm::Image img =
mm::read("https://upload.wikimedia.org/wikipedia/commons/8/87/Mandrillus_sphinx_339428057.jpg");

    std::cout << "Dimensões (H, W, Canais): (" << img.h << ", " << img.w << ", " <<
img.channels << ")\n";
    std::cout << "Tipo de dado: uint8\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: uint8
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)")
```

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

Alternativa: *Download* dell'immagine per archiviazione locale

In ambienti in cui la lettura diretta degli URL non è disponibile — a causa di restrizioni di rete, policy di *firewall* o assenza di connettività — un'alternativa consiste nel scaricare preventivamente l'immagine nel file system locale e quindi caricarla con `mm::read()`. Nel percorso C++, `mm::read` accetta anche URL direttamente (download interno tramite `curl`/`wget`); questa alternativa copre il caso di scaricare una volta e riutilizzare il file locale. Nell'esempio seguente, il file viene salvato come `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");   // legge dal file locale
    mm::show(img, MM_OUT);                      // Esempio: Cattura RGB (file locale)
    return 0;
}
```

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

Spiegazione

- `wget -O mandrill.png <URL>` esegue il *download* dell'immagine e la archivia localmente con il nome specificato;
- `mm::read("mandrill.png")` effettua la lettura del file direttamente dal file system, senza necessità di richieste HTTP aggiuntive;
- questa strategia riduce la dipendenza dalla connettività durante l'esecuzione degli esperimenti ed evita *download* ripetuti della stessa immagine.

Nota

Il prefisso `!` è utilizzato in ambienti basati su *notebook*, come Jupyter Notebook, JupyterLab e Google Colab, per eseguire comandi del sistema operativo direttamente in celle di codice. In terminali convenzionali, il comando deve essere utilizzato senza il prefisso `!`.
Se l'utilità `wget` non è installata, si può utilizzare in alternativa:

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

1.11 Conversione di Tipi e Soglia

Come abbiamo visto, un'immagine a colori nello spazio RGB è rappresentata dalla funzione:

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

Cioè, per ogni pixel (x, y), abbiamo tre valori (R, G, B) che ne definiscono il colore.

Conversione in Scala di Grigi

Per convertire un'immagine RGB in scala di grigi (*grayscale*), è necessario combinare i tre canali in un unico valore di intensità g, che rappresenta la luminosità percepita. Poiché l'occhio umano non è ugualmente sensibile al rosso, al verde e al blu, si utilizza una **media ponderata**. Lo standard ITU-R BT.601 ({ITU-R}, 2011) definisce i seguenti pesi:

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

Dopo il calcolo, il valore g viene arrotondato all'intero più vicino e adattato all'intervallo $[0, 255]$. Il risultato è una nuova immagine, ora in scala di grigi, rappresentata da:

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

Soglia (*Thresholding*)

A partire dall'immagine in scala di grigi $f_{\text{grigi}}(x, y)$, un'operazione fondamentale è la **soglia**, che produce un'immagine **binaria** (solo bianco e nero). A tale scopo, si sceglie un valore di taglio T (generalmente nell'intervallo $[0, 255]$) e si definisce:

$$f_{\text{bin}}(x, y) = \begin{cases} 255 & \text{se } f_{\text{grigi}}(x, y) > T \\ 0 & \text{altrimenti} \end{cases} \tag{1.4}$$

Esempio: Con $T = 128$, i pixel con intensità superiore a 128 diventano bianchi (255); gli altri diventano neri (0).

La soglia è ampiamente utilizzata per **segmentare gli oggetti dallo sfondo**, estrarre i bordi o creare maschere binarie per l'elaborazione successiva.

> **Nota:** Il valore 255 rappresenta il bianco massimo nelle immagini a 8 bit, mentre 0 rappresenta il nero assoluto.

Esempio pratico di conversione e soglia

La Figura 1.8 illustra i passaggi principali per trasformare un'immagine a colori in scala di grigi e, successivamente, convertirla in un'immagine binaria mediante soglia. Il codice seguente implementa questi passaggi:

```
%%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]

    // Leitura da imagem já fornecida: img

    // 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)")
```

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

1.12 Soglia con il metodo di Otsu

Come presentato in Equazione 1.4, la sogliatura converte un'immagine in scala di grigi in binaria utilizzando un valore di taglio T. Finora abbiamo fissato $T = 128$ manualmente.

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

Tuttavia, la scelta manuale di T non è sempre banale. La libreria `mm` offre un'alternativa automatica: quando la soglia **non viene fornita**, la funzione `mm::threshold(img_gray)` calcola il valore di T tramite il **metodo di Otsu** (OTSU, 1979). Questo metodo, che verrà dettagliato nei capitoli successivi, massimizza la varianza tra le classi dell'istogramma (frequenza di ogni tono di grigio), separando automaticamente i pixel dell'oggetto e dello sfondo.

Il codice seguente confronta la sogliatura manuale ($T = 128$) con quella automatica (Otsu). La binarizzazione con Otsu viene eseguita con `mm::threshold(img_gray)`; poiché l'API minima di `morph.hpp` non restituisce il T calcolato, il valore mostrato nell'etichetta della figura viene recuperato con `cv2.threshold` nella fase di visualizzazione (stesso algoritmo, stesso T).

```
%%writefile tmp/fig_01_otsu.cpp
#define MM_OUT "tmp/fig_01_otsu.png"
//| quarto-raw: false
//| label: fig-01-otsu
```

```
//| fig-cap: "Comparação entre limiarização manual (T=128) e automática (Otsu) sobre a imagem
em tons de cinza."
//| 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_38.png");
// [pdi:state-io:end]

    // 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)
    int T_otsu = mm::otsu(img_gray);
    mm::Image img_bin_otsu = mm::threshold(img_gray);  // mesmo T de Otsu
    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=128)", "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)
```

```
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)")
```

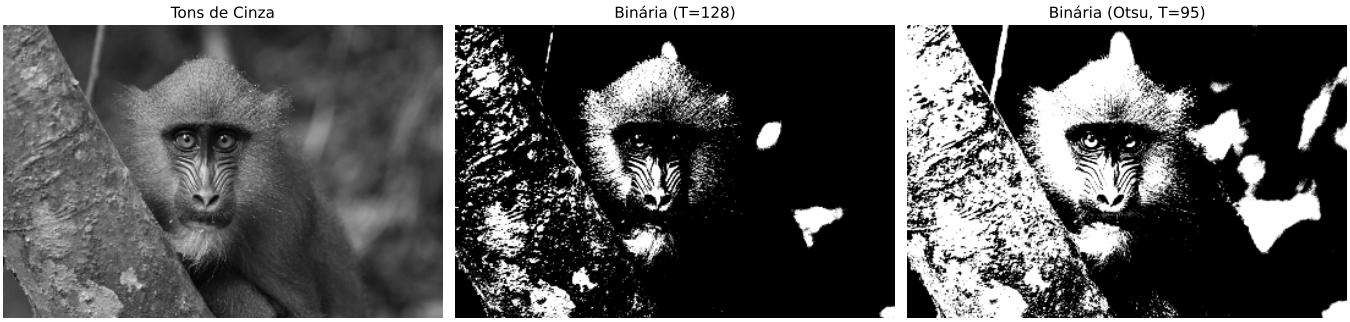


Figura 1.9: Comparação entre limiarização manual (T=128) e automática (Otsu) sobre a imagem em tons de cinza.

La Figura 1.9 mostra che la soglia ottenuta mediante Otsu si adatta automaticamente all'immagine, producendo una binarizzazione più efficace rispetto a un valore fisso, soprattutto quando le intensità dell'oggetto e dello sfondo sono ben separate nell'istogramma. Questa tecnica è ampiamente utilizzata nei sistemi di visione artificiale (VC) per la binarizzazione di documenti, il rilevamento di oggetti e il pre-processing delle immagini.

Semplicità della libreria `morph.hpp`

Mentre OpenCV richiede la chiamata completa:

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

la `morph.hpp` astrae tutta questa complessità: basta chiamare `mm::threshold(img_gray)`. La soglia di Otsu viene calcolata automaticamente e l'immagine binaria viene restituita direttamente. Questo approccio consente di concentrarsi sul concetto, non sui dettagli implementativi.

1.13 Accesso ai Pixel

In `morph.hpp`, l'immagine è un `mm::Image` con un buffer lineare `data` (riga dopo riga, canali interallacciati) e il metodo `at(y, x, c)` per l'accesso individuale, seguendo la convenzione matriciale **riga** (asse Y) e **colonna** (asse X): `img.at(riga, colonna)` per i toni di grigio e `img.at(riga, colonna, canale)` per ogni canale di un'immagine RGB.

Il codice della Figura 1.10 dimostra come estrarre questi valori in immagini a colori (RGB) e in toni di grigio, oltre a isolare l'intorno immediato del punto di interesse.

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

# 1. Coordinate del pixel bersaglio (riga, colonna)
r, c = 600, 800

# 2. Accesso diretto ai valori del pixel
pixel_cinza = img_gray[r, c]
print(f"Pixel bersaglio ({r}, {c}):")
print(f"  Toni di grigio (scalare) : {pixel_cinza}")
print(f"  Colorato (canali RGB)    : R={img[r, c, 0]}, G={img[r, c, 1]}, B={img[r, c, 2]}")

# 3. Intorno 3x3 attorno a (r, c): il pixel centrale va tra parentesi quadre
print("Matrice dell'intorno 3x3 (toni di grigio):")
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. Marca la posizione del pixel con un quadrato rosso vuoto (bordo di
#    3 px) su una copia dell'immagine e visualizza (equivalente al punto del matplotlib)
marcada = img.copy()
lado = 20   # metà lato del quadrato, in pixel
esp = 5     # spessore del bordo

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"Posizione del pixel ({r}, {c})")
```

Figura 1.10

Accesso ai pixel in C++ (`mm::Image`):

- **Scala di grigi:** `img_gray.at(r, c)` restituisce un `unsigned char` (da 0 a 255) con l'intensità del grigio nel pixel.
- **RGB:** `img.at(r, c, 0)`, `img.at(r, c, 1)`, `img.at(r, c, 2)` accedono ai canali **R**, **G** e **B** — un indice di canale alla volta; non esiste un vettore `[R, G, B]`.
- **Intorno 3×3:** esaminato tramite due cicli annidati su `img_gray.at(r + dy, c + dx)`, con `dy, dx` in $\{-1, 0, 1\}$ — non esiste lo slicing come in NumPy. Questa scansione è alla base per filtri spaziali e convoluzioni.
- Non esiste plottaggio interattivo (equivalente a `plt.plot`): per marcare la posizione del pixel, la cella di codice seguente **disegna un quadrato rosso vuoto** attorno ad esso in una copia dell'immagine (`marcata = img.copy()`, poi `marcata.at(...)`) e la visualizza con `mm::show`.

L'indicizzazione è *zero-based*: `(0,0)` è l'angolo in alto a sinistra. La prima dimensione controlla l'**altezza** (righe/Y) e la seconda la **larghezza** (colonne/X).

1.14 Riassunto

In questo capitolo sono stati presentati i fondamenti della rappresentazione delle immagini digitali: la definizione di pixel, la strutturazione delle immagini in matrici e l'impatto del campionamento e della quantizzazione sulla qualità finale:

- **Immagine digitale** = funzione $f(x, y)$ che mappa coordinate in intensità (scalari o vettoriali).
- **Dominio**: insieme finito $\mathbb{E} = \{(x, y) \in \mathbb{Z}^2 \mid 0 \leq x < L,\ 0 \leq y < H\}$.
- **Tipi principali**: binaria ($\mathcal{V} = \{0, 255\}$), toni di grigio ($\mathcal{V} = [0, 255]$) e RGB ($\mathcal{V} = [0, 255]^3$).
- **Sogliatura** converte i toni di grigio in immagine binaria; il **metodo di Otsu** determina automaticamente il valore di soglia massimizzando la varianza tra le classi.
- La `morph.hpp` (o `mm::`) offre funzioni didattiche per operazioni di base dell'elaborazione delle immagini, come `mm::gray()`, `mm::threshold()`, `mm::show()` (sovraccaricata per 1 immagine o più).
- **Trappola della copia**: `std::vector` ha semantica di valore, ma puntatori/riferimenti condivisi tra "righe" di una matrice reintroducono lo stesso problema di NumPy — preferisci `mm::Image`, che copia anch'essa per valore.
- Accesso ai pixel tramite `img.at(riga, colonna)`, con indicizzazione ***zero-based***.

Il Capitolo 2 tratterà **istogrammi ed equalizzazione del contrasto**.

1.15 🤖 Uso del Gemini Notebook come Tutor Complementare

In questa edizione, oltre ai *notebook* interattivi su Google Colab, il **Gemini Notebook** è disponibile come strumento complementare di studio. La piattaforma utilizza esclusivamente i documenti forniti dall'autore come base di conoscenza, garantendo risposte coerenti con il contenuto del libro.

🎓 Studia con il Tutor Intelligente

Accedi all'ambiente del capitolo tramite il *link* qui sotto ed esplora in particolare le opzioni **Guida allo Studio** e **Conversazione** per approfondire la tua comprensione.
🚀 ACCEDI AL GEMINI NOTEBOOK: CAPITOLO 01

🌐 **Lingua e Linguaggio di Programmazione**

Il progetto di questo capitolo nel Gemini Notebook è stato costruito esclusivamente con il testo in **portoghese** e gli esempi di codice in **Python**. Se stai studiando dall'edizione in inglese o francese,

oppure seguendo il percorso in C++, le risposte del tutor potrebbero non corrispondere esattamente alla versione che stai leggendo.

⚠️ **Avvertenza sul Contenuto Generato dall'IA**

L'IA è un potente alleato nello studio, ma il contenuto generato può contenere **errori o imprecisioni**. Consulta sempre **libri, articoli scientifici e altre fonti accademiche affidabili** per validare le informazioni. Quando possibile, esegui gli esempi pratici forniti in questo capitolo per verificare i risultati.

Funzionalità Disponibili sulla Piattaforma

Il **Gemini Notebook** offre una suite avanzata di strumenti basati sull'IA per trasformare il contenuto statico del libro in un'esperienza di apprendimento dinamica e multimediale. La piattaforma utilizza tecniche di **RAG** (*Retrieval-Augmented Generation*), fondate sul lavoro di Lewis (2020), per basare le risposte strettamente sui documenti forniti e minimizzare il verificarsi di allucinazioni.

Le principali funzionalità includono:

- **Sintesi Multimodali (Audio e Video)**: Generazione di conversazioni naturali tra esperti nel formato di **Sintesi Audio** (stile *podcast*) e **Sintesi Video**, che discutono i temi centrali del capitolo, come le differenze tra PDI e VC, o l'interpretazione di trasformazioni come la sogliatura e il metodo di Otsu.
- **Visualizzazione di Strutture (Mappa Mentale e Infografica)**: Creazione automatica di diagrammi che collegano visivamente i concetti, ad esempio, il flusso di elaborazione dalla cattura dell'immagine digitale, passando per la conversione in toni di grigio, la sogliatura e la segmentazione binaria.
- **Strumenti di Valutazione (Test e Schede Didattiche)**: Generazione di **Test** a scelta multipla e **Schede Didattiche** (*flashcards*) per il consolidamento delle conoscenze, basati sul testo d'autore (es.: domande sulla formula di conversione RGB→grigio o sul funzionamento della soglia globale e di Otsu).
- **Supporto alla Presentazione (*Slide* e Relazioni)**: Assistenza nella strutturazione di **Presentazioni di *Slide*** e nella redazione di **Relazioni** tecniche, facilitando la comunicazione dei risultati di esperimenti con le immagini.
- **Analisi dei Dati (Tabella Dati)**: Organizzazione dei dati estratti dal testo in tabelle strutturate, aiutando la comprensione di esempi pratici, come il confronto tra diversi valori di soglia.
- ***Chat* Contestualizzata**: Consente di porre domande dirette sul codice e sulla teoria, come: *"Come implementare la conversione da RGB a toni di grigio utilizzando i pesi dello standard ITU-R BT.601?"* o *"Cosa succede all'immagine binaria se scelgo una soglia T=200 invece di T=128?"*.

1.16 Elenco di Esercizi

1. (15%) Con le tue parole, definisci **immagine digitale** e **pixel**. Fornisci un esempio concreto di come un'immagine a colori (RGB) è rappresentata in forma matriciale nel computer.

2. (15%) Spiega le differenze tra **immagine binaria**, **toni di grigio (8 bit)** e **a colori RGB**, indicando l'intervallo di valori possibili per ogni pixel in ciascun tipo.

3. (20%) Considerando la formula di conversione RGB → toni di grigio dello standard ITU-R BT.601:

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

   Calcola il valore del pixel in toni di grigio per $(R, G, B) = (80, 180, 30)$. Arrotonda al numero intero più vicino.

4. (20%) Che cos'è la **sogliatura** (*thresholding*)? Spiega la differenza tra scegliere una soglia T fissa (es.: $T = 128$) e utilizzare il **metodo di Otsu** per la determinazione automatica della soglia. In poche parole, come sceglie la soglia il metodo di Otsu?

5. (15%) Nel contesto della libreria didattica `mm` discussa nel capitolo, rispondi:
   - a) (7,5%) Come si accede al valore del pixel nella posizione (riga=50, colonna=60) di un'immagine in scala di grigi `img_gray`?
   - b) (7,5%) Qual è il vantaggio di usare `mm::threshold(img_gray)` senza passare la soglia? Confronta con la chiamata equivalente in OpenCV (`cv::threshold`).
6. (15%) Cosa rappresentano i campi `h`, `w` e `channels` di un `mm::Image` per un'immagine RGB? Fornisci un esempio concreto con un'immagine di 640×480 pixel.

Riferimenti del Capitolo

La base teorica di questo capitolo comprende le seguenti opere di PDI e VC:

- Gonzalez (2018) per i fondamenti di **Elaborazione Digitale delle Immagini** (EDI).
- Singh (2019) per l'implementazione pratica di metodi di **elaborazione e analisi delle immagini**.
- Szeliski (2022) per lo studio di VC e algoritmi fondamentali.
- Bradski (2008) per l'applicazione della libreria **OpenCV** in ambiente Python.
- Lewis (2020) per il concetto di **generazione aumentata da recupero (RAG)**, utilizzato a supporto dell'elaborazione delle informazioni di questo materiale.

1.17 💻 Parte pratica con esercizi di programmazione

🎯 Obiettivo di questo Quaderno

Gli **Esercizi di Programmazione (EPs)** presentati di seguito possono essere sottomessi anche in attività Moodle (attività VPL) che forniscono *feedback* automatico.

Questo quaderno è stato sviluppato per superare le limitazioni d'uso di Moodle. Con esso, si deve:

1. **Sviluppare:** Scrivere e modificare la propria soluzione direttamente nell'ambiente Colab.
2. **Validare:** Testare il proprio codice localmente utilizzando gli **stessi casi di test** di Moodle.
3. **Organizzare:** Salvare in modo sicuro i propri codici delle attività VPL.
4. **Valutare:** Quando si è connessi a Moodle, basta copiare la propria soluzione e cliccare su **Valuta** in Moodle (**se si è nella rete UFABC**) per registrare il proprio voto ufficiale.

§ Istruzioni Passo a Passo

In un ambiente di esecuzione (come VSCode, Jupyter o Colab), seguire l'ordine seguente per configurare l'ambiente e validare i propri esercizi:

Preparazione dell'Ambiente

Eseguire la cella di codice seguente per scaricare `morph.py` e `testsuite.py` dal repository della disciplina — solo se non sono già presenti nella directory locale. Con entrambi in `./`, il *notebook* e i sottoprocessi del `TestSuite` trovano il modulo senza configurazioni aggiuntive di percorso.

Nota: Lo script `testsuite.py` cercherà automaticamente i casi di test in `all/{cap}/cases` su GitHub.

Scrittura del Codice

Salvare la propria soluzione in una cella di codice utilizzando il comando magico `%%writefile`. Il nome del file deve seguire il pattern `EPX_Y.*`, dove `X` è il capitolo, `Y` è l'esercizio e `*` è l'estensione del linguaggio.

Esempio: `%%writefile EP01_01.py`

Download

Scaricare `morph.py` e `testsuite.py` eseguendo la cella seguente:

```
import os, urllib.request

os.makedirs("tmp/state", exist_ok=True)  # artefatti di build della pista C++ (.cpp, binario,
PNG)

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")

# Il kernel è comunque Python anche nella pista C++: `mm` (morph.py) è usato dai
# simulatori, dalla visualizzazione delle figure che il binario C++ genera e dallo
# stato mm::Image tra le celle. cpp=True scarica anche la pista compilata
# (morph.hpp + stb_image*.h), usata nell'#include delle celle %%writefile *.cpp.
import config
config.setup(testsuite=True, cpp=True)
from morph import mm
from testsuite import TestSuite
```

```
Ambiente pronto. Morph: 1.1.9 | OpenCV: 5.0.0 | TestSuite: 1.1.2
```

Esecuzione dei Test

Dopo aver salvato il file con la tua soluzione, esegui il comando seguente (in una nuova cella) per valutare i test automatici:

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

Sostituisci l'estensione in base al linguaggio utilizzato:

Linguaggio	Estensione
Python	`.py`
Java	`.java`
C	`.c`
C++	`.cpp`
JavaScript	`.js`
R	`.r`

Come funziona: Il `TestSuite` scarica i casi di test da GitHub, esegue il tuo programma con ciascun input e confronta l'output con quello atteso – calcolando automaticamente il tuo voto.

Per testare codice Python direttamente, senza salvare file, usa `run_code(codice)` passando il codice come *stringa* in una variabile `codice`:

```
codice = """
from morph import mm
# ... il tuo codice qui ...
"""
TestSuite("EP04_01").run_code(codice)
```

⚠️ Importante: Regole e Buone Pratiche

🔹 **Sull'Inserimento dei Dati**

Il tuo programma deve leggere l'input standard (tastiera).

- **Python:** Utilizza `input()`.
- **Altri linguaggi:** Utilizza il comando di lettura standard equivalente (`cin`, `Scanner`, ecc.).

🔹 **Configurazione dell'IA su Colab**

Per un migliore apprendimento, si consiglia di disattivare il completamento automatico del codice tramite IA, poiché non sarà disponibile durante le valutazioni. Ad esempio, nel browser Chrome:

- Vai su: **Strumenti > Impostazioni > IA generativa**
- Deseleziona: **Abilita generazione di codice**

🔹 **Integrità Accademica (Plagio)**

Questa risorsa di test locali si applica a ES **senza variazioni**. Tuttavia:

- **Individualità:** Ogni studente deve sviluppare la propria soluzione.
- **Rilevamento di Similarità:** Il professore utilizza strumenti che rilevano copie, **anche con modifica di nomi di variabili o spazi bianchi**.

1.17.1 EP01_01 📏 Tre metriche di distanza nella PDA (Processamento Digitale delle Immagini)

In questa attività, devi scrivere un programma che calcoli le tre distanze classiche nella PDA: **Euclidea (L2)**, **City-Block (L1)** e ***Chessboard*** **(L∞)**.

- Leggi **4 numeri reali** che rappresentano le coordinate: A_x, A_y, B_x, B_y.
- Calcola le tre distanze utilizzando le formule:

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

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

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

- Stampa i tre risultati, ciascuno su una riga, formattati con **due cifre decimali**, nell'ordine: **Euclidea**, ***City-block***, ***Chessboard***.

📌 **Importante**:

- Utilizza le funzioni matematiche standard del tuo linguaggio: `math.sqrt`, `abs` (o `fabs`) e `max`.
- L'output deve contenere **solo i numeri** (uno per riga), senza testi aggiuntivi.
- Consulta un simulatore interattivo per questo esercizio nella **Figura 1.11** (grafico con trascinamento dei punti e visualizzazione delle tre metriche).

1.17.1.1 🖼️ Perché è importante? – Costo computazionale

In un'immagine **1000×1000 pixel** (1 milione di pixel), calcolare la distanza di ogni pixel da un punto di riferimento richiede **1 milione di operazioni**. La scelta della metrica influisce sulle prestazioni:

Metrica	Operazioni per pixel	Costo relativo (1M pixel)	Quando utilizzarla
Euclidea (L2)	2 sottrazioni, 2 moltiplicazioni, 1 somma, 1 `sqrt`	🔴 Più costosa – `sqrt` è dispendiosa	Distanza "reale" nello spazio continuo
***City-block* (L1)**	2 sottrazioni, 2 `abs`, 1 somma	🟡 Moderata – senza radice quadrata	Griglie, robotica, immagini binarie
***Chessboard* (L∞)**	2 sottrazioni, 2 `abs`, 1 `max`	🟢 Più efficiente	Movimenti di pezzi, morfologia

La funzione `sqrt` è computazionalmente più costosa rispetto ad operazioni come addizione, sottrazione, moltiplicazione e valore assoluto. Nelle CPU moderne, la differenza può essere piccola (circa 1,5× a 3×), ma nei sistemi embedded o nei cicli di milioni di iterazioni, qualsiasi guadagno conta. Per questo motivo, **quando l'obiettivo è solo confrontare le distanze** (es.: trovare il punto più vicino), utilizza la **distanza euclidea al quadrato**.

1.17.1.2 📋 Compito (specifica per VPL)

Input:
Un'unica riga con quattro numeri reali: `Ax Ay Bx By`

Output:
Tre righe, ciascuna con un numero reale con due cifre decimali (Euclidea, City-block, Chessboard).

1.17.1.3 📌 Esempi

Input	Output	Osservazione
0 0 3 4	5.00 7.00 4.00	Triangolo 3-4-5
0 0 1 1	1.41 2.00 1.00	Diagonale unitaria

Esempio di test di `sqrt` in Python, con `timeit` che isola ogni operazione:

```
%%writefile tmp/mm_out_1.cpp
// Benchmark: soma simples vs soma + sqrt
#include <chrono>
#include <cmath>
#include <iostream>

int main() {
    const long long N = 50000000;  // 50 milhões

    // Mide o tempo apenas da soma
    auto t0 = std::chrono::high_resolution_clock::now();
    for (long long i = 0; i < N; ++i) {
        double a = 3.0, b = 4.0;
        volatile double resultado = a + b;  // evita otimização
    }
    auto t1 = std::chrono::high_resolution_clock::now();
    double t_soma = std::chrono::duration<double>(t1 - t0).count();
```

```
    // Mide o tempo da soma + sqrt
    t0 = std::chrono::high_resolution_clock::now();
    for (long long i = 0; i < N; ++i) {
        double a = 3.0, b = 4.0;
        volatile double resultado = std::sqrt(a*a + b*b);  // evita otimização
    }
    t1 = std::chrono::high_resolution_clock::now();
    double t_sqrt = std::chrono::duration<double>(t1 - t0).count();

    std::cout << "Soma simples       : " << t_soma << " s\n";
    std::cout << "Soma + sqrt        : " << t_sqrt << " s\n";
    std::cout << "Razão (sqrt/soma)  : " << (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 simples       : 0.155514 s
Soma + sqrt        : 0.415125 s
Razão (sqrt/soma)  : 2.66938x
```

1.17.1.4 🐍 Python

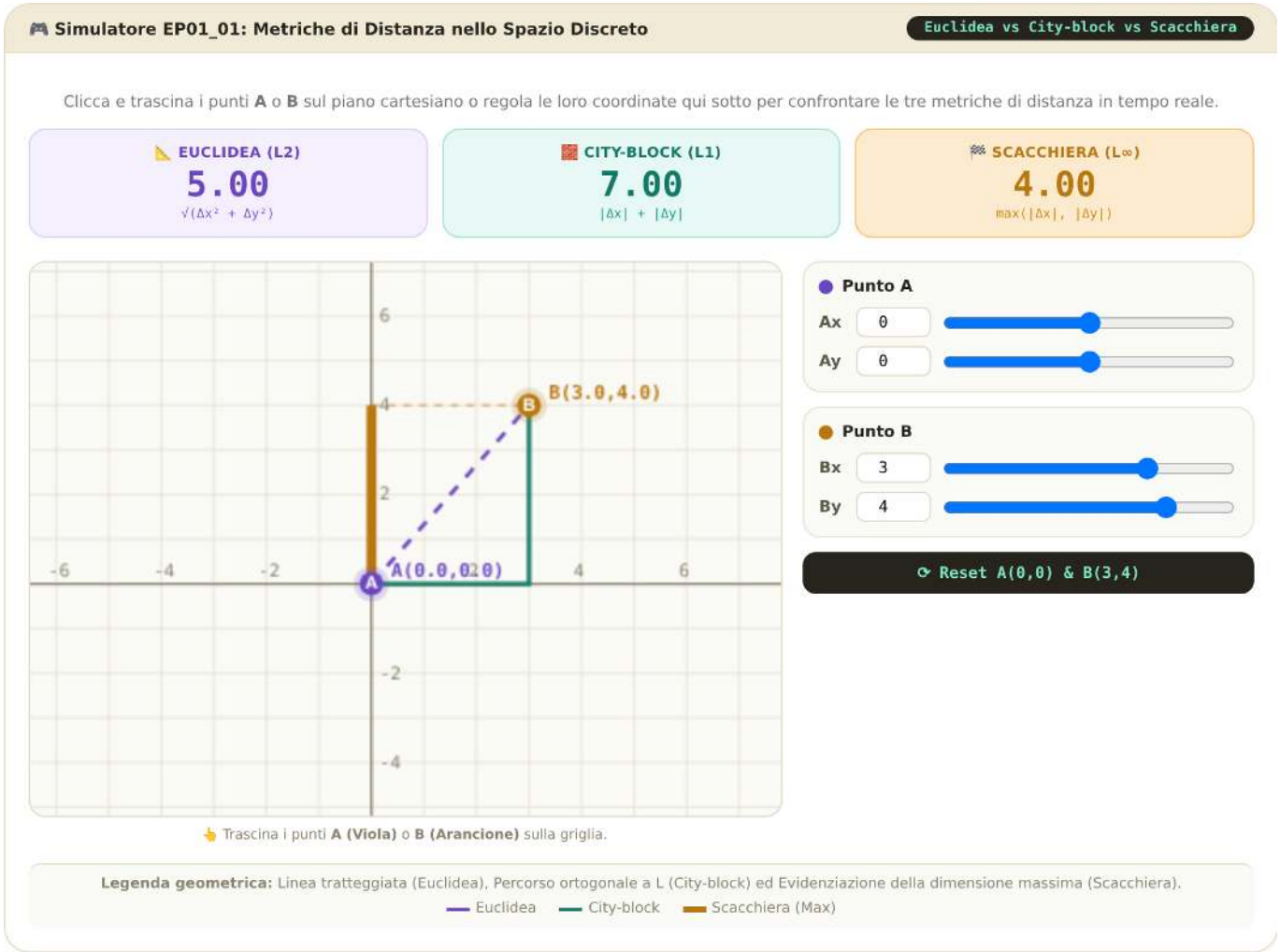
Basta creare una normale cella di codice e inserire il codice Python. L'input può essere simulato usando `input()`, che funziona normalmente.

Esempio di cella:

```
%%writefile EP01_01.py
# Codice Python
x1,y1,x2,y2 = int(input()), int(input()), int(input()), int(input())
# Calcolo delle differenze
dx = abs(x2 - x1)
dy = abs(y2 - y1)

# 1. Distanza Euclidea (L2)
dist_euclidiana = (dx**2 + dy**2)**0.5

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

# 3. Distanza Scacchiera / Chebyshev (Linf)
dist_chessboard = max(dx, dy)

# Output formattato secondo i casi di test
print(f"{dist_euclidiana:.2f}")
print(f"{dist_city_block:.2f}")
print(f"{dist_chessboard:.2f}")
```

```
Overwriting EP01_01.py
```

```
# Aspetta che tu inserisca 4 numeri interi eseguendo questa cella.
# In Jupyter o Google Colab, la magia %run -i permette allo script di leggere dalla tastiera.
# In un terminale comune, useresti: python3 EP01_01.py (senza il '!' e senza '%run').
```



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

Figura 1.11: Simulatore EP01_01: Distanze Euclidea, City-block e Chessboard

```
# %run -i EP01_01.py
```

```
# Invia 4 interi come input standard (stdin) allo script EP01_01.py usando una 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

Per eseguire Java in Colab, è necessario utilizzare una cella con il prefisso `%%writefile` per salvare il codice in un file, compilarlo ed eseguirlo.

```
%%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

Analogamente, usa `%%writefile` per salvare il codice, poi compila ed esegui. Per C, utilizziamo il compilatore **GCC**.

```
# Installare il compilatore GCC e gli strumenti di build
# build-essential include gcc, g++, make, ecc.
import platform, shutil, subprocess

def instalar_gcc():
    if shutil.which("gcc"):
        print(" GCC già disponibile."); return
    if platform.system() != "Linux":
        print(" Mac: xcode-select --install | Windows: WSL o 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 installato!")

instalar_gcc()
```

```
 GCC già disponibile.
```

```
%%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
```

```
# Compila il file .c generando l'eseguibile EP01_01
# -lm viene usato per linkare la libreria matematica (math.h) se necessario

!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++

Analogamente al Java, usa `%%writefile` per salvare il codice, poi compila ed esegui. Ricorda che, in Colab, è necessario installare anche quanto segue:

```
# Installa il compilatore G++ per C++
# Il build-essential include g++, make, ecc.
import platform, shutil, subprocess

def instalar_gpp():
    if shutil.which("g++"):
        print(" G++ già disponibile."); return
    if platform.system() != "Linux":
        if platform.system() == "Darwin":
            print(" Mac: xcode-select --install")
        else:
            print(" Windows: usa WSL o 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(" Compilatore C++ pronto.")

instalar_gpp()
```

```
 G++ già disponibile.
```

```
%%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)

Per JavaScript, usa `%%writefile` per creare il file ed eseguilo con 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

In Colab è possibile eseguire codice R direttamente utilizzando la magia `%%R`.
Il programma deve leggere **quattro numeri** (x1, y1, x2, y2) e mostrare la distanza euclidea con **due cifre decimali**.

Esempio di cella:

```
%%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))
```

```
# Installare R e Rscript
# Il pacchetto r-base installa l'ambiente R completo, inclusi Rscript
import platform, shutil, subprocess

def instalar_r():
    if shutil.which("R"):
        print("  R già disponibile."); 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("  Ambiente R pronto.")
```

```
instalar_r()
```

```
R già disponibile.
```

Para testare nello stesso modo degli esempi precedenti, si deve utilizzare l'input standard (stdin) nel terminale o adattare il codice come segue:

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

# Output: Euclidea, City-block e Scacchiera
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 📊 Prestazioni Predittive — Metriche di ML in VC

In questa attività entrerai nel mondo dell'**Apprendimento Automatico** (*Machine Learning*). Il tuo obiettivo è valutare le prestazioni di un classificatore binario calcolando le metriche a partire da una **Matrice di Confusione**.

1.17.2.1 🧠 Perché la metrica giusta è importante?

Immagina un rilevatore di **monete da R$ 1,00**. L'impatto dell'errore definisce la metrica prioritaria:

Metrica	Esempio Pratico	Importanza in PDI/VC
Accuratezza	**Conteggio dei Grani**	Utile quando le classi sono bilanciate (es: metà dei grani difettosi, metà sani).
Precisione	**Sicurezza/Biometria**	Fondamentale per evitare **Falsi Positivi** (es: non consentire a un impostore di accedere a un sistema per un errore di riconoscimento).
Sensibilità	**Salute (Tumori)**	Fondamentale per evitare **Falsi Negativi** (es: non lasciare che un tumore passi inosservato in una radiografia).

Metrica	Esempio Pratico	Importanza in PDI/VC
F1-score	**Banconote**	Ideale per un **equilibrio** tra non rifiutare banconote autentiche e non accettare banconote false.

1.17.2.2 📊 La Matrice di Confusione

	Predetta Positiva	**Predetta Negativa**
Reale Positiva	**VP** (Vero Positivo)	**FN** (Falso Negativo)
Reale Negativa	**FP** (Falso Positivo)	**VN** (Vero Negativo)

Compito:

1. Leggi **4 valori interi** nell'ordine: `VP`, `FN`, `FP`, `VN`.
2. Calcola le metriche utilizzando le formule:

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

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

$$\text{Sensibilità (Recall)} = \frac{VP}{VP + FN}$$

$$\text{F1-score} = \frac{2 \times \text{Precisione} \times \text{Sensibilità}}{\text{Precisione} + \text{Sensibilità}}$$

3. Stampa i risultati formattati con **due cifre decimali**, uno per riga.

📌 **Importante**:

- Usa la divisione in virgola mobile per evitare risultati troncati.
- L'ordine di uscita deve essere: **Accuratezza**, **Precisione**, **Sensibilità** e **F1-score**.
- Consulta la simulazione interattiva di questo EP alla Figura 1.12.

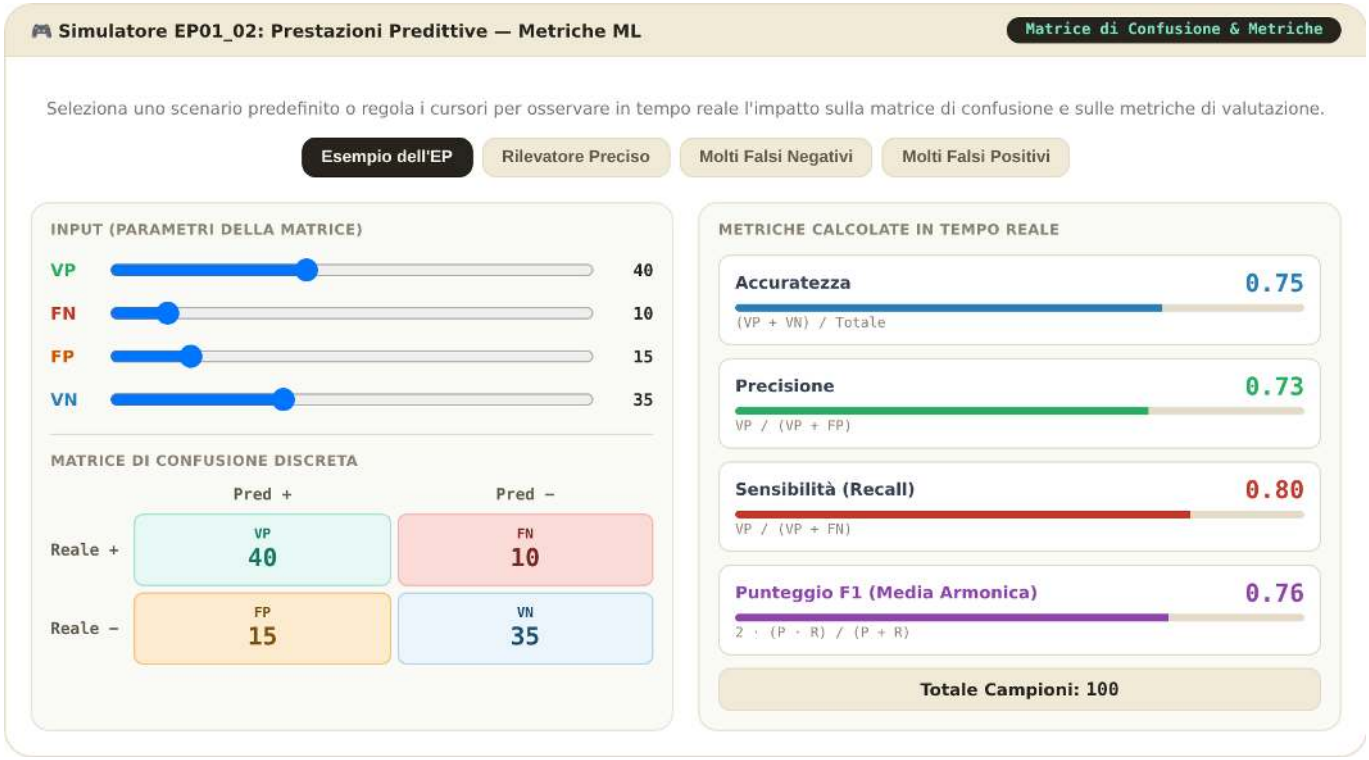
1.17.2.3 📌 Esempio di Esecuzione

Input	Output	Osservazione
`40`	`0.75`	Accuratezza
`10`	`0.73`	Precisione
`15`	`0.80`	Sensibilità
`35`	`0.76`	F1-score

(Nota: VP=40, FN=10, FP=15, VN=35. Totale dei casi = 100)

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

```
Overwriting EP01_02.cpp
```



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

Figura 1.12: Simulatore EP01_02: Prestazioni Predittive — Metriche di ML

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

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

1.17.3 EP01_03 ↗ *Mean Average Precision* (mAP) — Curva Precisione-Sensibilità

In questa attività valuterai un classificatore binario (es.: rilevamento della deforestazione in immagini satellitari, vedi dgi.inpe.br) tramite la **curva Precisione-Sensibilità** e la metrica **mAP (*Mean Average Precision*)**. L'mAP è uno standard in competizioni come COCO (*Common Objects in Context*) e PASCAL VOC (*Visual Object Classes*) e nei modelli YOLO (*You Only Look Once*).

1.17.3.1 🧠 Perché l'mAP è la metrica standard?

Nella EP01_02 hai visto che la scelta della soglia altera significativamente Precisione e Sensibilità. L'**mAP (*Mean Average Precision*)** risolve questo problema: valuta il modello su **più soglie** (ogni soglia deve generare una matrice di confusione diversa) e riassume le prestazioni tramite l'**area sotto la curva Precisione-Sensibilità (P-S)**.

Mentre l'F1-*Score* considera un singolo punto di equilibrio, l'mAP considera l'intera curva. Più il valore è vicino a **1,0**, migliore è il rilevatore su tutte le soglie e classi (es.: monete da 25, 50 centesimi e 1 euro).

Metrica	Cosa riassume	Limitazione
F1-*Score*	Equilibrio P × S su una singola soglia	Dipende dalla soglia scelta
AP	Area sotto la curva P-S di una classe	Valida solo per una classe
mAP	Media delle AP su tutte le classi	Più complesso da implementare

Riferimenti: Roboflow — mAP · Video esplicativo

1.17.3.2 Come viene calcolato l'mAP — passo dopo passo

1. **Soglie fisse** (usa sempre questa lista):

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

2. Per ogni soglia (t), classifica i campioni: `predito = 1 se confiança  t, altrimenti 0`. Calcola VP, FP, FN, VN e ottieni Precisione((t)) e Sensibilità((t)).
3. Costruisci la curva P-S: coppie (Sensibilità((t)), Precisione((t))), ordinate per Sensibilità crescente.
4. **Rendi monotona la Precisione**:

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

5. Calcola l'**AP** (area sotto la curva monotona) usando la **regola del trapezio** (approssimazione più accurata rispetto alla semplice somma di 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** = media delle AP di tutte le classi. In questa EP c'è solo **1 classe**, quindi mAP = AP.

> **Nota**
>
> 📐 **Differenza riassunta:**
> La *somma di Riemann* approssima l'area tramite rettangoli, potendo sottostimare o sovrastimare. La **regola del trapezio** usa trapezi, riducendo l'errore considerando la media tra i valori agli estremi dell'intervallo, risultando generalmente più precisa per funzioni continue a tratti, come la curva Precisione-Sensibilità.

1.17.3.3 📋 Compito

Leggi un intero `n` (quantità di campioni). Poi leggi `n` righe, ciascuna con: `verdade` (0 o 1) e `confiança` (float 0.0–1.0).

Calcola e stampa, per la **soglia 0.85** (indice 9 della lista):

- Matrice di Confusione (VP, FN, FP, VN)
- Accuratezza, Precisione, Sensibilità e F1-Score

Successivamente, per **tutte le soglie**, stampa:

- Precisioni grezze, Precisioni monotone e Sensibilità, separate da `,`
- mAP finale

1.17.3.4 📌 Importante

- Soglia fissa per le metriche individuali: **0.85**
- Divisione sicura: se il denominatore è zero, usa 0
- Formattazione: due cifre decimali
- Rendi monotona da destra verso sinistra
- La Figura 1.13 presenta una simulazione di questa domanda

1.17.3.5 📌 Esempio di Esecuzione

Input	Output Atteso
7 0 0.94 1 0.80 1 0.69 0 0.67 1 0.30 1 0.15 1 0.15	# MÉTRICHE PER LA SOGLIA 0.85 # Matrice di Confusione: VP = 0, FN = 5 FP = 1, VN = 1 Metriche di Valutazione: Accuratezza: 0.14 Precisione: 0.00 Sensibilità: 0.00 F1-Score: 0.00 # MÉTRICHE PER TUTTE LE SOGLIE # Precisioni: 0.00, 0.00, 0.00, 0.50, 0.50, 0.50, 0.50, 0.50, 0.60, 0.71, 0.71 Precisioni mon.: 0.71, 0.71, 0.71, 0.71, 0.71, 0.71, 0.71, 0.71, 0.71, 0.71, 0.71 Sensibilità: 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 🐍 Suggerimento per calcolare l'AP (con regola del trapezio)

```python
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):
        # Regola del trapezio: media delle altezze per 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
```

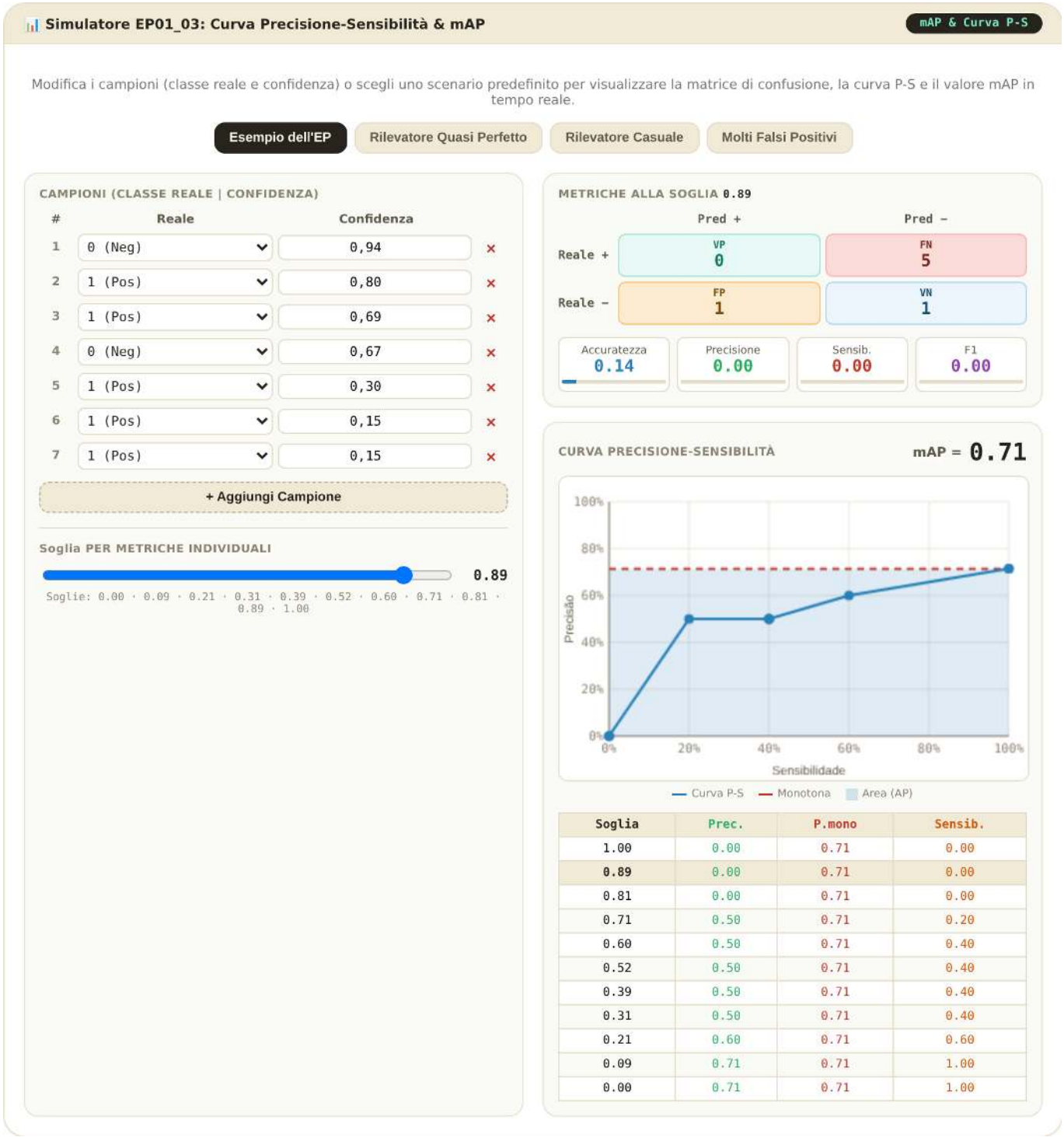
```
TestSuite("EP01_03.cpp").run()
```

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

1.17.4 EP01_04 🖼️ Lettura e Informazioni di un'Immagine in Matrice

In questa attività, devi scrivere un programma che elabori un'immagine digitale rappresentata come una matrice di pixel in scala di grigi.

- Leggi due interi **L** e **C**, che rappresentano il numero di righe e di colonne.



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

Figura 1.13: Simulatore EP01_03: Mean Average Precision (mAP) e Curva P-S

- Leggi i **L * C** valori interi che compongono la matrice dell'immagine (ogni valore tra 0 e 255).
- Calcola e stampa le seguenti informazioni:

1. Il numero di righe.
2. Il numero di colonne.
3. Il valore del pixel più grande (**Massimo**).
4. Il valore del pixel più piccolo (**Minimo**).
5. La **Media** aritmetica di tutti i pixel.

📌 **Importante**:

- L'output deve seguire esattamente il formato etichettato (es: `Righe: X`).
- Il valore della media deve essere formattato con **due cifre decimali**.
- Vedi un simulatore interattivo per questa domanda nella **Figura 1.14** (griglia interattiva per la visualizzazione delle intensità e dei calcoli in tempo reale).

1.17.4.1 🧠 Perché è importante? – L'Immagine come Dato

Ogni immagine digitale è, in fondo, una struttura dati. In scala di grigi a 8 bit, ogni pixel è un valore scalare. Estrarre statistiche di base è il primo passo per:

Operazione	Utilità Pratica
Massimo/Minimo	Identificare se l'immagine è "slavata" (basso contrasto) o saturata.
Media	Calcolare la luminosità globale della scena per regolazioni dell'esposizione.
Normalizzazione	Ridimensionare i valori in intervalli come $[0, 1]$ nelle reti neurali.

1.17.4.2 📋 Attività (specifica per VPL)

Input:

La prima riga contiene l'intero `L` (righe).

La seconda riga contiene l'intero `C` (colonne).

Le righe successive contengono gli elementi della matrice.

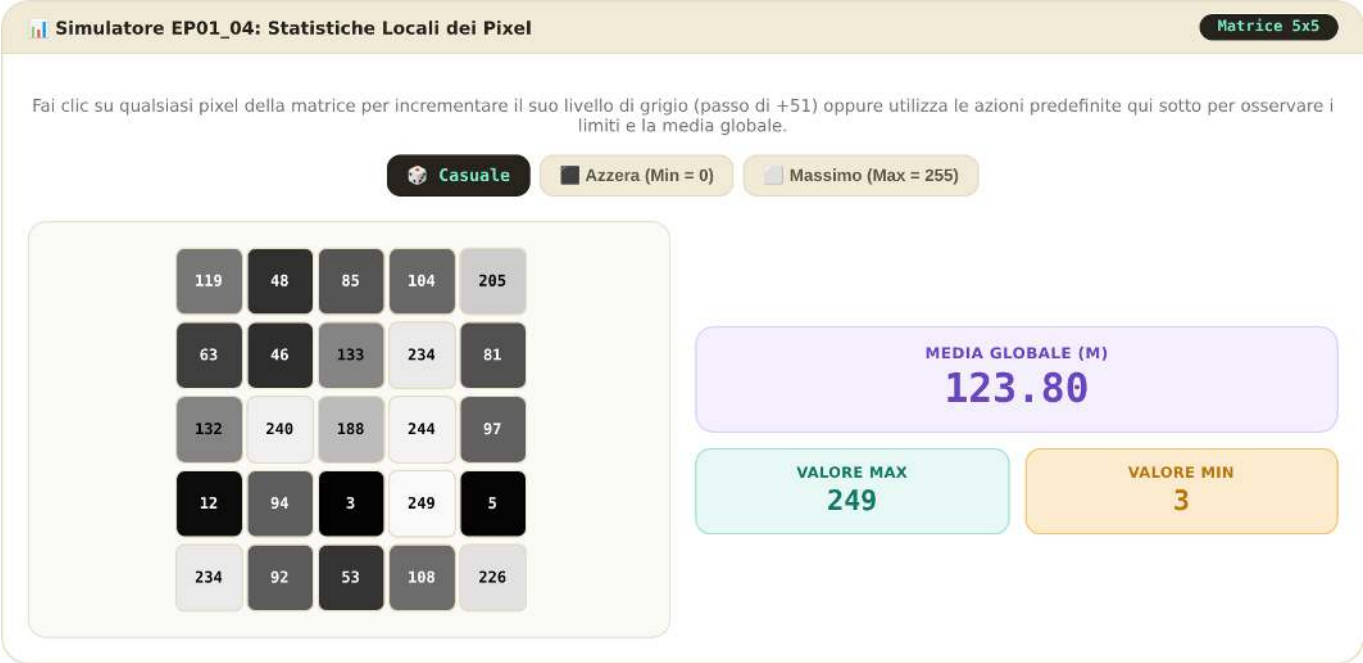
Output:

Cinque righe formattate secondo l'esempio:

```
Righe: L

Colonne: C

Max: V

Min: V

Media: V.VV
```

1.17.4.3 📌 Esempi

Input	Output	Osservazione
2 3 0 128 255 50 100 200	Righe: 2 Colonne: 3 Max: 255 Min: 0 Media: 122.17	Immagine piccola ad alto contrasto



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

Figura 1.14: Simulatore EP01_04: Statistiche dei pixel in matrice discreta 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 🔄 Negativo di un'Immagine in Ton di Grigio

In questa attività, si deve scrivere un programma che calcoli il negativo di un'immagine digitale.

- Leggere due interi **L** e **C**, che rappresentano il numero di righe e colonne.
- Leggere i valori interi che compongono la matrice dell'immagine.
- Per ogni pixel, applicare la trasformazione di inversione:

$$pixel_{negativo} = 255 - pixel_{originale}$$

- Stampare la matrice risultante, mantenendo il formato originale (L righe e C colonne).

📌 **Importante**:

- I valori di ciascuna riga nell'output devono essere separati da uno **spazio bianco**.
- L'output deve contenere **solo i numeri** della matrice risultante.

- Vedere un simulatore interattivo per questa questione nella **Figura 1.15** (confronto in tempo reale tra la matrice originale e il suo negativo).

1.17.5.1 🧠 Perché è importante? – Inversione di Intensità

Il **negativo** è una trasformazione lineare di base che inverte la scala di luminosità. È uno strumento essenziale per l'occhio umano per identificare dettagli chiari "nascosti" in sfondi più scuri, ed è ampiamente utilizzato in:

Applicazione	Utilità
Immagini Mediche	Migliora la visualizzazione di anomalie in tessuti densi (es. Radiografie).
Astronomia	Evidenziare galassie e nebulose deboli contro il vuoto dello spazio.
Arti Digitali	Effetti estetici e preparazione di maschere di selezione.

1.17.5.2 📋 Compito (specifica per VPL)

Input:

La prima riga contiene l'intero `L`.

La seconda riga contiene l'intero `C`.

Le righe successive contengono gli elementi della matrice.

Output:

La matrice invertita con `L` righe e `C` colonne.

1.17.5.3 📌 Esempi

Input	Output	Osservazione
2 3 0 128 255 50 100 255	255 127 0 205 155 55	Dove c'era 0 (nero) diventa 255 (bianco)

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

```
Overwriting EP01_05.cpp
```

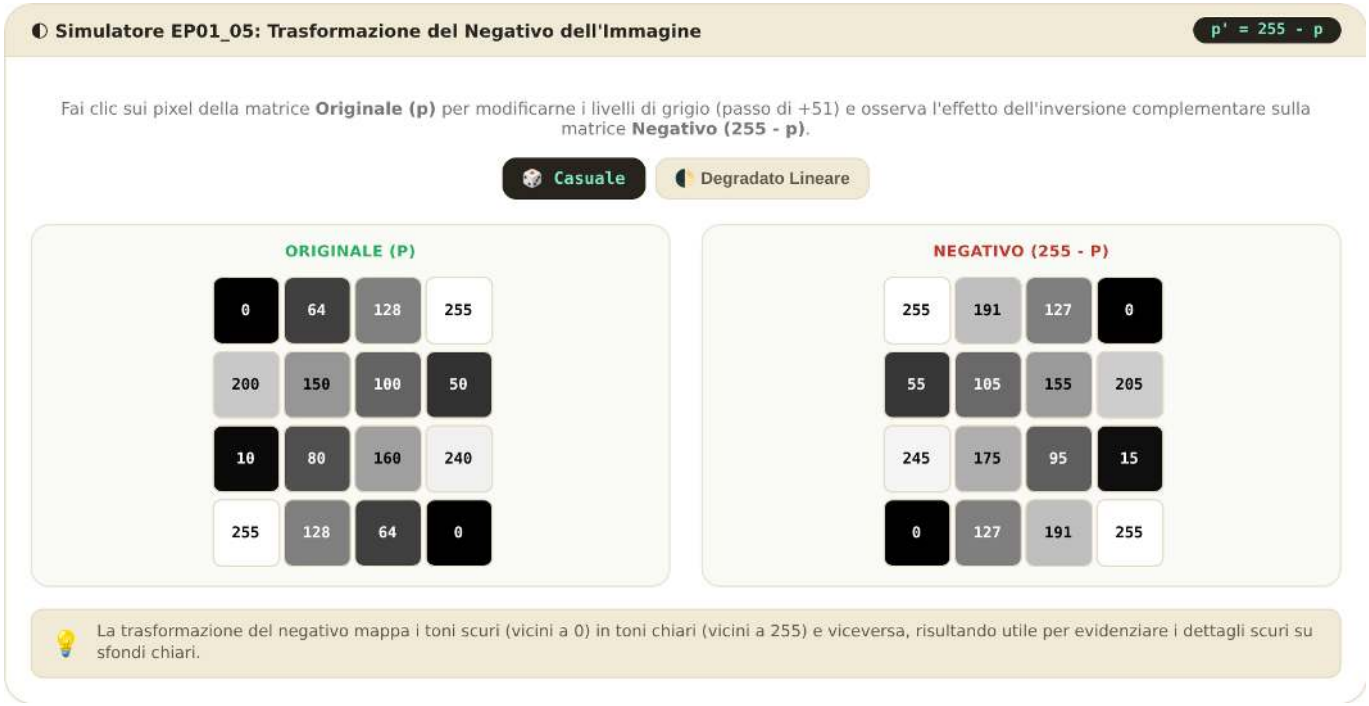
```
TestSuite("EP01_05.cpp").run()
```

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

1.17.6 EP01_06 🎨 Conversione RGB → Toni di Grigio (ITU-R BT.601)

In questa attività, devi scrivere un programma che converta pixel colorati (RGB) in toni di grigio utilizzando la ponderazione fisiologica dello standard ITU-R BT.601.

- Leggi due interi **L** e **C**, che rappresentano il numero di righe e colonne.



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

Figura 1.15: Simulatore EP01_05: Trasformazione del Negativo dell'Immagine (Inversione di Intensità)

- Leggi **L** × **C** triple di interi, dove ogni terna rappresenta i canali **R** (Rosso), **G** (Verde) e **B** (Blu) di un pixel.
- Per ogni pixel, calcola il valore di grigio (g) usando la formula:

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

- Stampa la matrice risultante (L righe e C colonne) contenente i valori interi convertiti.

📌 **Importante**:

- Utilizza la funzione `round()` del tuo linguaggio per garantire l'arrotondamento corretto all'intero più vicino.
- L'output deve contenere solo i valori di grigio, mantenendo la struttura a matrice (separati da spazio sulla riga).
- Vedi un simulatore interattivo per questa domanda nella **Figura 1.16** (regola gli slider per osservare come ogni colore contribuisce alla luminosità finale).

1.17.6.1 🧠 Perché non usare semplicemente la media?

L'occhio umano non percepisce tutti i colori con la stessa intensità. Siamo molto più sensibili al **Verde** che al **Blu** a causa della nostra evoluzione biologica. Lo standard ITU-R BT.601 utilizza pesi specifici per creare un'immagine in grigio che appaia naturalmente corretta alla nostra visione:

Canale	Peso	Percezione Umana
🟢 **Verde**	**58.7%**	Massima sensibilità (distinzione del fogliame).
🔴 **Rosso**	**29.9%**	Sensibilità media.
🔵 **Blu**	**11.4%**	Bassa sensibilità (toni più scuri).

1.17.6.2 📋 Compito (specifica per VPL)

Input:

La prima riga contiene l'intero `L`.

La seconda riga contiene l'intero `C`.

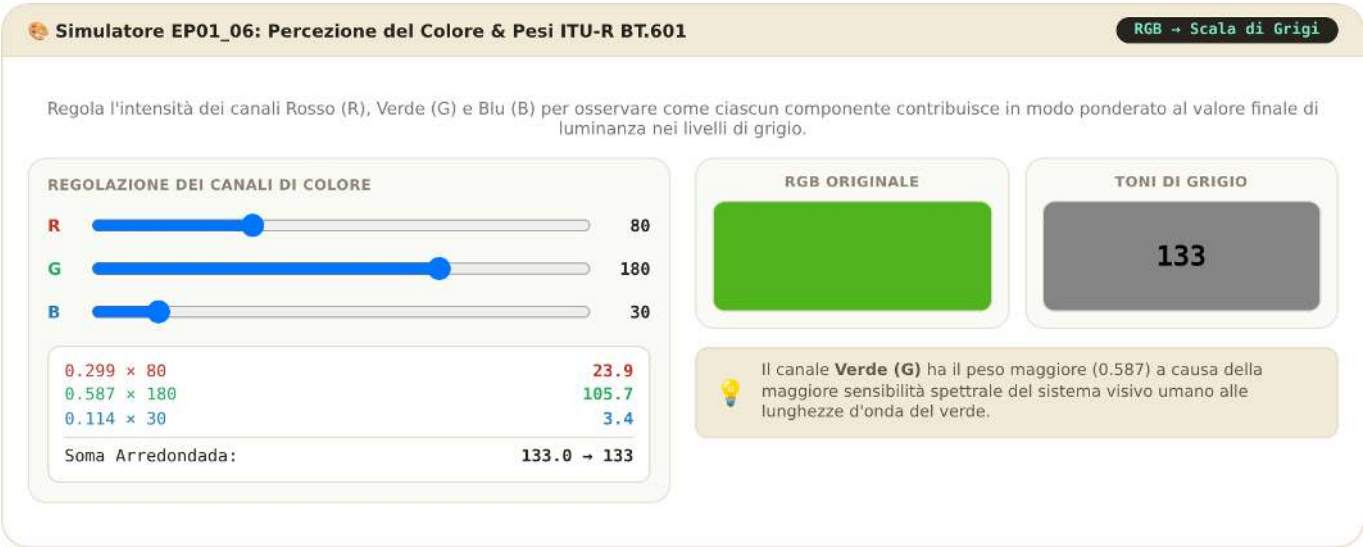
Le righe successive contengono triple di interi `R G B` per ogni pixel.

Output:

La matrice di grigi con `L` righe e `C` colonne.

1.17.6.3 📌 Esempi

Input	Output	Osservazione
1 3 255 0 0 0 255 0 0 0 255	76 150 29	Nota come il Verde (150) sia più luminoso del Blu (29)



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

Figura 1.16: Simulatore EP01_06: Conversione RGB in Scala di Grigi (Ponderazione Percettiva ITU-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 ⚫ Soglia Manuale: Immagine Binaria

In questa attività, devi scrivere un programma che esegua la segmentazione di un'immagine tramite sogliatura (*thresholding*).

- Leggi due interi **L** e **C**, che rappresentano le dimensioni della matrice.

- Leggi un intero **T**, che sarà il valore della soglia (taglio).
- Leggi i valori interi della matrice.
- Per ogni pixel p, applica la seguente regola di binarizzazione:

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

- Stampa la matrice risultante contenente solo i valori 0 o 255.

📌 **Importante**:

- Presta attenzione all'operatore: il pixel diventa bianco (255) solo se è **strettamente maggiore** di T.
- L'output deve mantenere la struttura della matrice (L righe e C colonne).
- Consulta un simulatore interattivo per questo problema su **Figura 1.17** (regola il cursore di T per osservare come gli oggetti vengono isolati dallo sfondo).

1.17.7.1 🧠 Cos'è la Segmentazione?

La sogliatura è il metodo più semplice per separare gli oggetti di interesse dallo sfondo dell'immagine. Trasformando i toni di grigio in bianco e nero puro, creiamo una mappa binaria che facilita il conteggio degli oggetti o l'identificazione delle forme:

Valore del Pixel (p)	Condizione	Risultato Finale
Scuro ($p \leq T$)	Sfondo/Rumore	**0** (Nero)
Chiaro ($p > T$)	Oggetto/In evidenza	**255** (Bianco)

1.17.7.2 📋 Compito (specifica per VPL)

Input:

La prima riga contiene l'intero `L`.

La seconda riga contiene l'intero `C`.

La terza riga contiene l'intero `T` (soglia).

Le righe successive contengono gli elementi della matrice.

Output:

La matrice binarizzata (0 o 255) con `L` righe e `C` colonne.

1.17.7.3 📌 Esempi

Input	Output	Osservazione
2 4 128 0 100 128 200 50 129 255 64	0 0 0 255 0 255 255 0	Nota che il valore 128 è diventato 0 (poiché $128 \leq 128$)

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

```
Overwriting EP01_07.cpp
```

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

Figura 1.17: Simulatore EP01_07: Soglia Globale Interattiva (Binarizzazione p > T)

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

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

1.17.8 EP01_08 🎨 Rimappatura per Intervallo di Intensità

In questa attività, devi scrivere un programma che applichi trasformazioni lineari distinte a diverse regioni di intensità dell'immagine.

- Leggi due interi **L** e **C**, che rappresentano le dimensioni della matrice.
- Leggi gli interi **T** (soglia), (delta 1) e (delta 2).
- Leggi i valori interi della matrice.
- Per ogni pixel p, applica la regola di rimappatura condizionale:

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

- Stampa la matrice risultante con i nuovi valori di intensità.

📌 **Importante**:

- è l'offset applicato ai pixel scuri (al di sotto della soglia).
- è l'offset applicato ai pixel chiari (maggiori o uguali alla soglia).
- I casi di test garantiscono che il risultato sia sempre nell'intervallo valido da **0 a 255**, quindi non è necessario gestire saturazione o arrotondamenti.
- L'output deve mantenere la struttura di matrice (**L** righe e **C** colonne).

1.17.8.1 🧠 Trasformazione Condizionale dei Pixel

Nell'elaborazione digitale delle immagini (PDI), spesso dobbiamo trattare le regioni in modo indipendente. Questa tecnica consente, ad esempio, di schiarire solo le ombre di una fotografia (aumentando i pixel scuri) senza bruciare la luminosità delle aree già chiare, o viceversa.

Intervallo di Intensità	Condizione	Operazione
Pixel scuri	$p < T$	$p + \delta_1$
Pixel chiari	$p \geq T$	$p + \delta_2$

1.17.8.2 📋 Compito (specifica per VPL)

Input:

La prima riga contiene gli interi `L` e `C`.

La seconda riga contiene gli interi `T`, e .

Le righe successive contengono gli elementi della matrice.

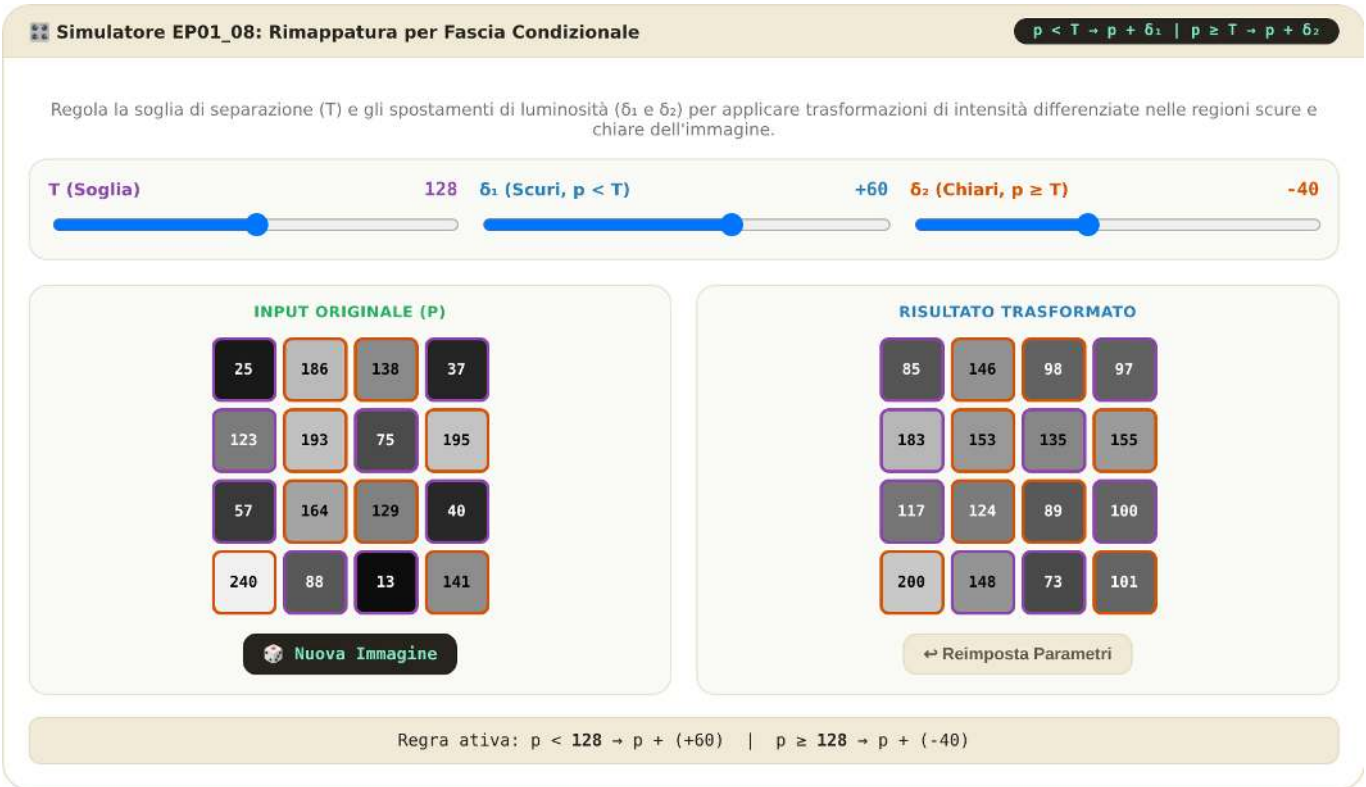
Output:

La matrice trasformata con `L` righe e `C` colonne, con valori separati da spazi.

1.17.8.3 📌 Esempi

Input	Output	Osservazione

`2 4`
`128 60 -40`
`0 100 150 255`
`80 128 200 30`
	`60 160 110 215`
	`140 88 160 90`
		I pixel < 128 sommano 60. I pixel $\geq$ 128 sottraggono 40.



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

Figura 1.18: Simulatore EP01_08: Rimappatura per Intervallo Condizionale (Luminosità e Contrasto per Soglia)

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

```
Overwriting EP01_08.cpp
```

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

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

1.17.9 EP01_09 🏁 Schema a Scacchiera: Matrice a Quadretti

In questa attività, devi scrivere un programma che generi un'immagine sintetica con il motivo di una scacchiera.

- Leggi due interi **L** (righe) e **C** (colonne).
- Genera una matrice in cui i valori alternano tra **0** (nero) e **1** (bianco).
- La logica di riempimento deve seguire la regola di parità:
- L'elemento nella posizione $(0, 0)$ è sempre **0**.
- Un pixel nella posizione (i, j) sarà **1** se la somma degli indici $(i + j)$ è **dispari**.
- Un pixel nella posizione (i, j) sarà **0** se la somma degli indici $(i + j)$ è **pari**.

📌 **Importante**:

- I colori devono alternarsi correttamente sia orizzontalmente che verticalmente.
- L'output deve essere la matrice stampata riga per riga, con gli elementi separati da uno spazio.
- Consulta un simulatore interattivo per questo problema su **Figura 1.19** (regola le dimensioni per visualizzare la costruzione della griglia e l'output testuale corrispondente).

1.17.9.1 🧠 Motivi Sintetici

Creare motivi geometrici è un esercizio fondamentale per padroneggiare la **logica degli indici** nelle matrici. Nell'elaborazione digitale delle immagini, il motivo a scacchiera non è solo estetico; è ampiamente utilizzato per:

Applicazione	Utilità
Calibrazione della fotocamera	Stimare i parametri intrinseci ed estrinseci dell'obiettivo.
Correzione della distorsione	Identificare e correggere l'effetto "a barile" o "a cuscino" negli obiettivi grandangolari.
Mappatura 3D	Proiettare motivi noti per ricostruire superfici in sistemi a luce strutturata.

1.17.9.2 📋 Compito (specifica per VPL)

Input:

Una riga contenente l'intero `L` (righe).

Una riga contenente l'intero `C` (colonne).

Output:

La matrice a scacchiera con `L` righe e `C` colonne, stampata con spazi tra gli elementi.

1.17.9.3 📌 Esempi

Input	Output	Osservazione
3 4	0 1 0 1 1 0 1 0 0 1 0 1	Nota che ogni riga inizia con l'inverso della precedente

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

Figura 1.19: Simulatore EP01_09: Generatore di Motivo a Scacchiera (Logica di Parità (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 📄 Metadati: Lettura di File PGM

In questa attività, devi leggere un file immagine nel formato **PGM (Portable Gray Map)** ed estrarne le dimensioni dall'intestazione.

- Il formato **PGM (P2)** è un file di testo semplice (ASCII) che memorizza immagini in scala di grigi.
- Il file possiede un'**intestazione** strutturata nel seguente modo:

1. **Versione:** L'identificatore `P2`.
2. **Commenti:** Righe opzionali che iniziano con `#` (devono essere ignorate).
3. **Dimensioni:** Due interi che rappresentano **Larghezza** e **Altezza**.
4. **Massimo:** Un intero che rappresenta l'intensità massima (generalmente 255).

- Dopo l'intestazione, seguono i dati dei pixel.

📌 **Importante**:

- **Lettura del File:** Devi aprire il file indicato nell'esempio utilizzando la funzione `open()` di Python.

- **Ordine di Uscita:** Contrariamente all'ordine presente nel file, l'output atteso deve essere nel formato di tupla: `(Altezza, Larghezza, Canali)`.
- Poiché i file PGM sono in scala di grigi, il numero di **Canali** è sempre **1**.
- Vedi un simulatore interattivo per questa domanda alla **Figura 1.20** (regola le dimensioni per vedere come viene generata l'intestazione ASCII).

1.17.10.1 🧠 Comprendere il formato PGM

Il formato PGM è uno dei più semplici per l'elaborazione delle immagini. Essendo in testo puro, consente di visualizzare i metadati e persino i valori dei pixel aprendo il file in un blocco note:

Componente	Esempio	Significato
Numero Magico	`P2`	Identifica che è un PGM in formato testo (ASCII).
Commento	`# CREATOR...`	Riga informativa ignorata dal processore.
Dimensioni	`397 343`	**397** colonne (Larghezza) e **343** righe (Altezza).
Intensità	`255`	Definisce il valore del bianco puro (scala da 0 a 255).

1.17.10.2 📋 Compito (specifica per VPL)

Input:

Nessun input da tastiera. Il programma deve leggere il file `"aula01fig03b.pgm"` presente nella directory di esecuzione.

Output:

Una tupla contenente `(Altezza, Larghezza, 1)`.

1.17.10.3 📌 Esempi

Nome del File	Output Atteso	Osservazione
"aula01fig03b.pgm"	(343, 397, 1)	Nota l'inversione dell'ordine: Altezza prima

> **Nota**
>
> Il file necessario per questo EP verrà scaricato automaticamente dal repository tramite il seguente codice:

```
%%writefile tmp/mm_out_2.cpp
// Compile with: g++ -std=c++17 -O2 program.cpp -o program $(pkg-config --cflags --libs
opencv4) -lm -lstdc++fs

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

int main() {
    const std::string BASE_URL =
"https://raw.githubusercontent.com/fzampirolli/pdi-vc/master/all/cap01/imagens";
```

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

Figura 1.20: Simulatore EP01_10: Struttura del file PGM (Intestazione ASCII P2 e mappatura a tupla Python)

```
    const std::string file = "aula01fig03b.pgm";

    const std::string url = BASE_URL + "/" + file;

    // Scarica il file direttamente
    mm::Image img = mm::read(url);
    std::cout << "  File scaricato: " << file << "\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
```

```
  File scaricato: aula01fig03b.pgm
```

```
%%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 ↗ Analisi di Vicinato: Filtro di Massimo 1D

In questa attività, devi implementare un semplice filtro morfologico di massimo che opera su un segnale unidimensionale (vettore).

- Leggi un intero **n**, che rappresenta la dimensione del vettore.
- Leggi i **n** elementi interi che compongono il vettore originale **v1**.
- Crea un nuovo vettore **v2**, in cui ogni posizione i è il risultato del confronto tra l'elemento corrente e i suoi vicini immediati:

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

📌 **Importante**:

- **Bordi:** Alle estremità del vettore (indici 0 e $n-1$), il vicinato ha solo due elementi (quello stesso e l'unico vicino disponibile). All'indice 0, confronta solo $v1[0]$ e $v1[1]$. All'ultimo indice, confronta solo $v1[n-2]$ e $v1[n-1]$.
- **Output:** Stampa l'intestazione "v2:" seguita dai valori del vettore risultante, uno per riga.
- Per un simulatore interattivo su questo problema, vedi **Figura 1.21** (passa il mouse sui risultati per visualizzare la finestra di vicinato utilizzata nel calcolo).

1.17.11.1 🧠 Perché analizzare i vicini?

Nell'elaborazione delle immagini, il valore di un pixel raramente è isolato; dipende dal contesto circostante. Il **Filtro di Massimo** è la base dell'operazione di **Dilatazione** nella morfologia matematica e serve a:

Funzione	Effetto Visivo
Enfatizzazione	Espande strutture luminose e "ingrossa" oggetti chiari.
Rimozione del Rumore	Elimina piccoli punti neri (rumore "sale e pepe" scuro).
Riempimento	Chiude piccoli buchi o lacune in forme binarie.

1.17.11.2 📋 Compito (specifica per VPL)

Input:

Un intero `n`.

Nelle righe successive, i `n` elementi interi del vettore.

Output:

La *stringa* `v2:` sulla prima riga.

Nelle righe successive, ogni elemento di `v2` (uno per riga).

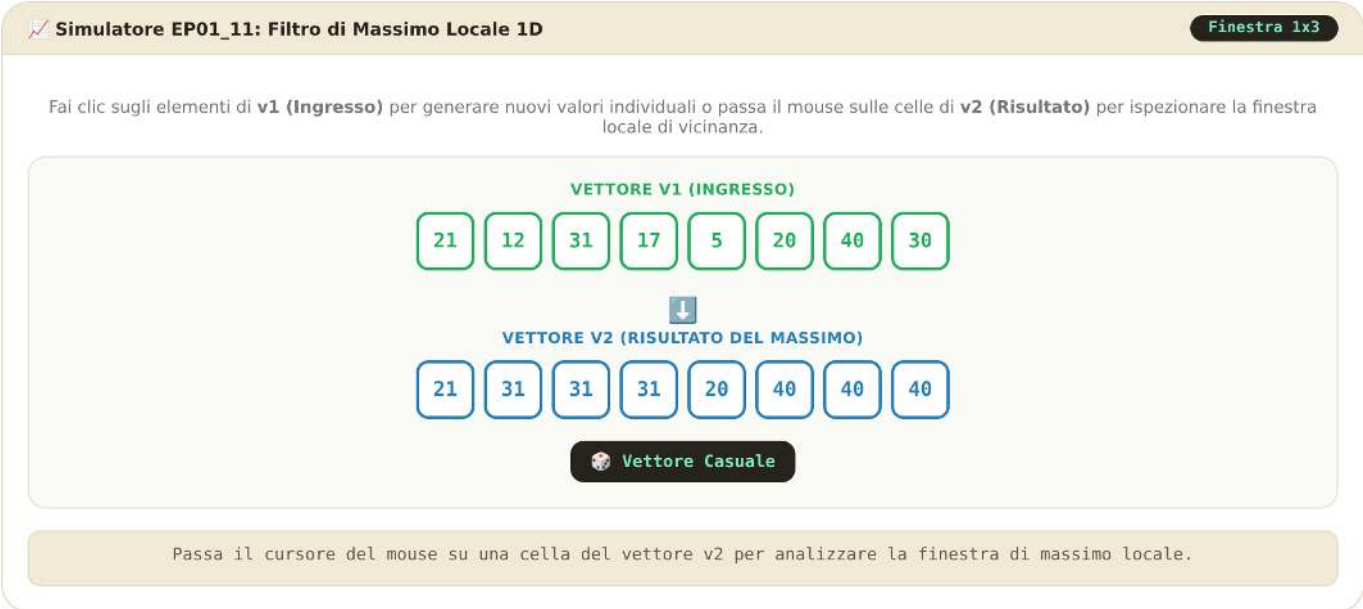
1.17.11.3 📌 Esempi

Input	Output	Osservazione
5	v2:	All'indice 1: max(10, 20, 5) = 20
10	20	
20	20	
5	30	
30	30	
15	30	

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

```
Overwriting EP01_11.cpp
```

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



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

Figura 1.21: Simulatore EP01_11: Filtro di Massimo Locale 1D (Vicinato 1x3 con Condizione di Bordo)

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

Capitolo 2

Dalla Cattura al Pixel - Campionamento, Quantizzazione e Connettività

Executar Colab Abrir GitHub

Questo capitolo approfondisce la comprensione dell'immagine digitale, passando dalla natura fisica della cattura alla sua rappresentazione matematica discreta. Indaghiamo come la luce diventa dato e come l'organizzazione spaziale dei pixel definisce le relazioni di vicinanza e connettività essenziali per algoritmi avanzati di Visione Artificiale (VC).

2.1 Obiettivi

Al termine di questo capitolo, sarai in grado di:

- Spiegare il modello fisico della formazione dell'immagine basato su illuminazione e riflettanza.
- Distinguere i meccanismi della visione umana da quelli dei sensori digitali.
- Comprendere i processi di **campionamento** (discretizzazione dello spazio) e **quantizzazione** (discretizzazione dell'intensità).
- Descrivere le relazioni topologiche tra pixel: vicinanza, adiacenza, connettività e distanze.
- Eseguire trasformazioni geometriche di base (traslazione, rotazione, scala) preservando la qualità.
- Visualizzare in pratica gli effetti della variazione della risoluzione spaziale e della profondità di bit.

2.2 L'Occhio e la Fotocamera - Elementi della Percezione Visiva

La formazione di un'immagine digitale inizia con la cattura della luce riflessa dagli oggetti. Come illustrato nella Figura 2.1, sia l'occhio umano che le fotocamere digitali seguono principi ottici simili per mettere a fuoco la luce su una superficie sensibile, sebbene utilizzino meccanismi biologici ed elettronici distinti per la trasduzione del segnale.

2.2.1 Visione umana

L'occhio funziona come un sistema ottico complesso: la luce attraversa la cornea, l'umor acqueo, la pupilla (controllata dall'iride) e il cristallino — che regola dinamicamente la messa a fuoco — fino a raggiungere la retina. Sulla retina si trovano i fotorecettori: i **coni** (6 milioni), concentrati nella fovea, sono responsabili della visione dei colori e dei dettagli, mentre i **bastoncelli** (120 milioni) garantiscono la visione in condizioni di bassa luminosità (visione scotopica), rilevando solo intensità di grigio. Il punto cieco è la regione da cui origina il nervo ottico, priva di recettori.

2.2.2 Sensori digitali

Nelle fotocamere, i sensori di immagine svolgono il ruolo della retina. I due tipi più comuni sono il **CCD** (*Charge-Coupled Device* - Dispositivo a Carica Accoppiata) e il **CMOS** (*Complementary Metal-Oxide-Semiconductor* - Semiconduttore a Ossido Metallico Complementare). Il sensore è composto da una matrice di fotositi (pixel) che accumulano carica elettrica proporzionale alla luce incidente.

Per la ricostruzione dei colori, si utilizza il **Filtro di Bayer**, una matrice di filtri colorati che consente a ciascun pixel di catturare una sola componente cromatica: rosso, verde o blu (RGGB - *Red, Green, Green, Blue*). Successivamente, un **ADC** (*Analog-to-Digital Converter* - Convertitore Analogico-Digitale) quantizza questa carica in valori numerici, definiti da una profondità di bit (es.: 8 bit, che producono 256 livelli di intensità).

> **Curiosità**
>
> Sebbene l'occhio umano abbia milioni di recettori, la risoluzione ad alta definizione è limitata alla fovea (visione centrale), equivalente a circa 120×120 pixel. La percezione di una scena completa in alta risoluzione è il risultato di un intenso post-elaborazione svolto dal cervello.

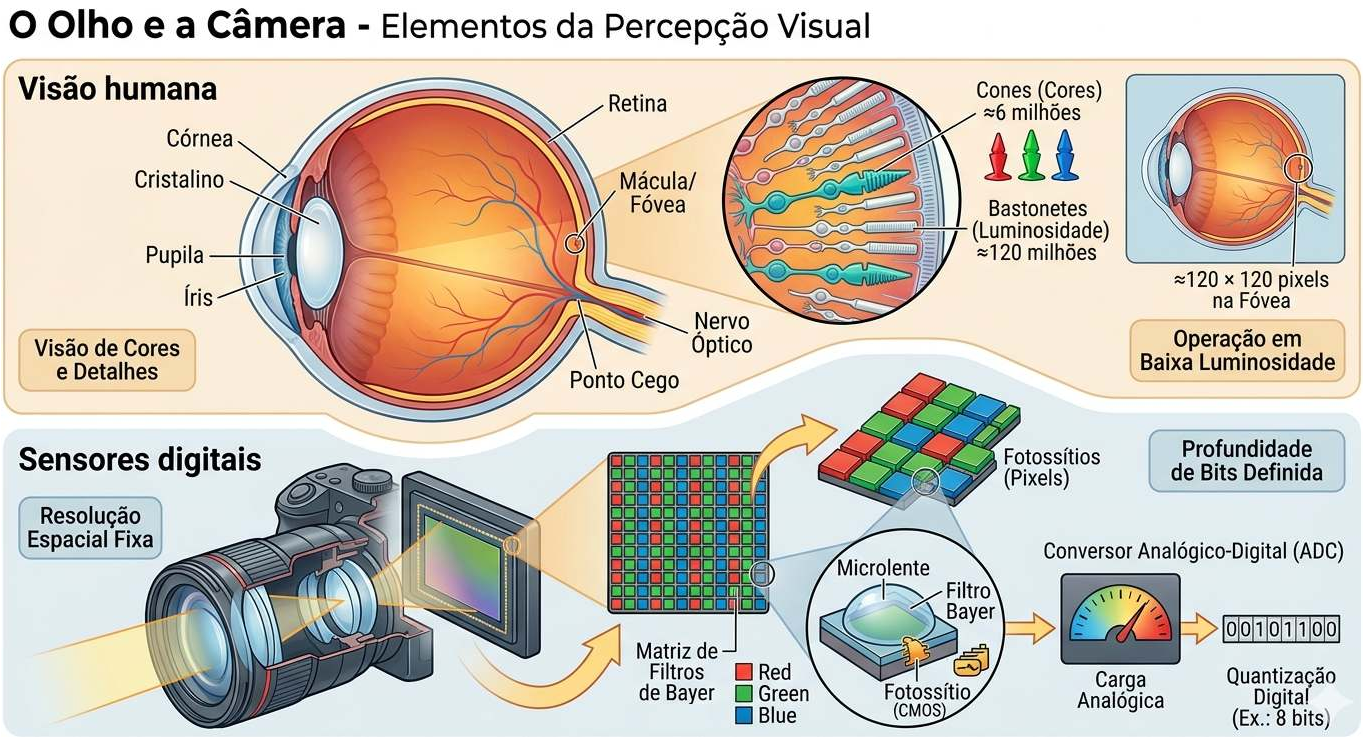


Figura 2.1: Confronto didattico tra il sistema visivo biologico e quello elettronico: in alto, l'anatomia dell'occhio umano che evidenzia la retina e i fotorecettori (coni e bastoncelli); in basso, la struttura di una fotocamera digitale che dettaglia il sensore CMOS, la matrice di filtri di Bayer (RGGB) e il processo di quantizzazione digitale eseguito dall'ADC.

2.3 Illusioni Ottiche: Le Sfide della Percezione Visiva

Mentre i sensori digitali catturano l'intensità della luce in modo lineare e oggettivo, il sistema visivo umano interpreta la scena basandosi su contesto, esperienze pregresse e meccanismi biologici di sopravvivenza. Le illusioni ottiche non sono "errori" dell'occhio, ma evidenze dell'intenso **post-elaborazione cerebrale** svolto nella corteccia visiva.

2.3.1 Ambiguità e Contesto

Il cervello cerca costantemente di dare senso a pattern ambigui. Nell'esempio del **Vaso di Rubin** (vedi Figura 2.2), la percezione alterna tra la figura (vaso) e lo sfondo (due volti), dimostrando che non riusciamo a elaborare entrambe le interpretazioni simultaneamente. L'illusione dell'**Elefante di Shepard**, invece, gioca con la nostra incapacità di riconciliare linee di contorno che suggeriscono volume in posizioni logicamente impossibili.

2.3.2 Luminosità e Contrasto Locale

Molte illusioni derivano dall'**inibizione laterale**, meccanismo mediante il quale neuroni vicini nella retina competono tra loro per evidenziare i bordi. Nella **Griglia Scintillante**, punti scuri "fantasma" sembrano apparire alle intersezioni bianche a causa di questa elaborazione locale del contrasto.

L'illusione dell'**Ombra sulla Scacchiera di Adelson** è forse la più impressionante per la VC: il quadrato "A" e il quadrato "B" hanno esattamente lo stesso valore di grigio nel sensore (o nel file digitale), ma il cervello "corregge" la luminosità di "B" comprendendo che si trova sotto un'ombra proiettata, percependo quest'ultimo come più chiaro.

2.3.3 Geometria e Prospettiva

La percezione della profondità può essere ingannata da costruzioni geometriche che sfidano la logica tridimensionale da un particolare angolo di osservazione. La **Scala di Schröder** sfrutta l'ambiguità della prospettiva per creare un oggetto che sembra salire o scendere a seconda di come viene osservato, evidenziando come la nostra interpretazione di "sopra" e "sotto" dipenda dal punto di fuga.

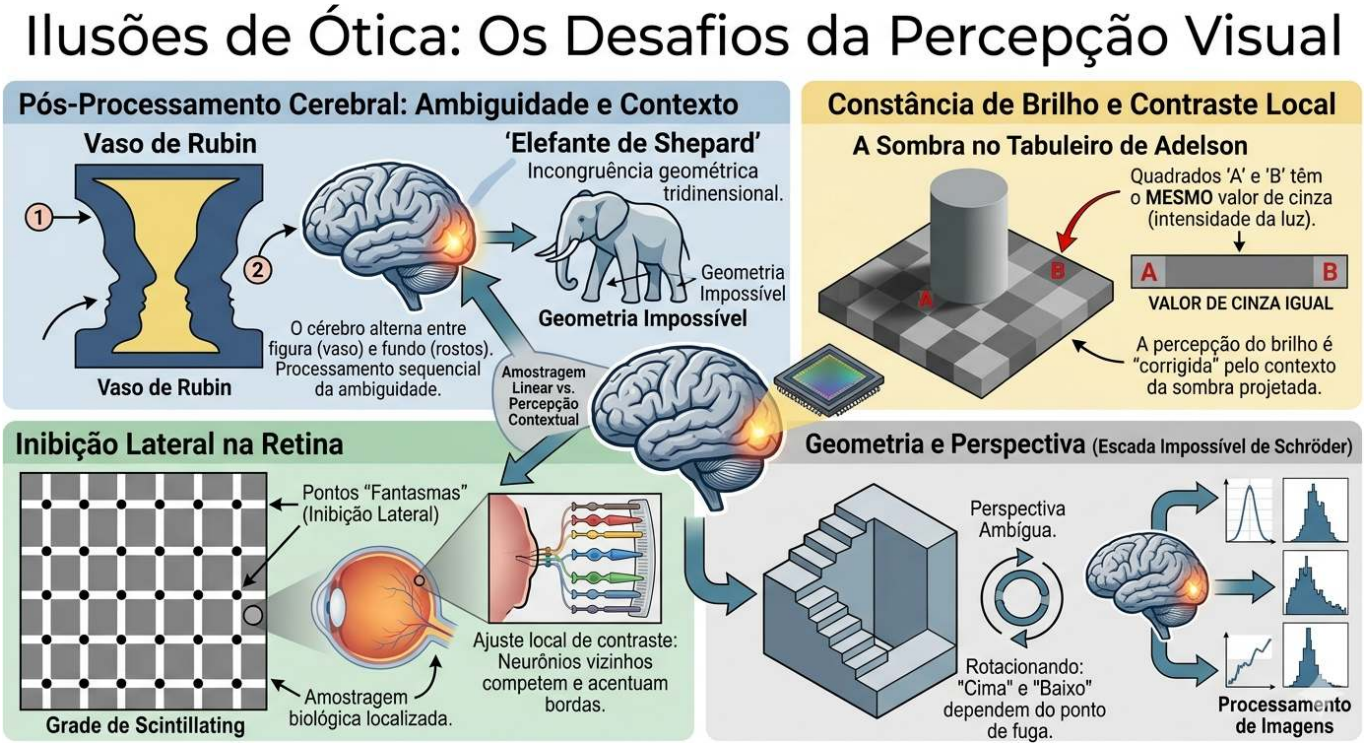


Figura 2.2: Raccolta di sfide percettive: (in alto a sinistra) Vaso di Rubin — ambiguità figura-sfondo; (in alto al centro) Elefante di Shepard — incongruenza geometrica; (in alto a destra) Ombra di Adelson — costanza della luminosità basata sul contesto; (in basso a sinistra) Griglia di punti — inibizione laterale; (in basso a destra) Scala di Schröder.

2.4 Il Modello Matematico della Formazione dell'Immagine

Un'immagine può essere modellata come il prodotto di due funzioni:

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

dove:

- $i(x, y)$ è l'**illuminazione** incidente sulla scena (energia luminosa per unità di area), determinata dalla sorgente di luce;

- $r(x, y)$ è la **riflettanza** dell'oggetto (frazione di luce riflessa), determinata dalle proprietà ottiche della superficie, con $0 < r(x, y) < 1$.

Nella pratica, le due componenti variano su scale spaziali distinte: $i(x, y)$ tende a variare lentamente nello spazio, mentre $r(x, y)$ può presentare variazioni brusche associate a trame, bordi e dettagli fini. Le tecniche di elaborazione delle immagini digitali (PDI) spesso cercano di separare o compensare queste componenti, come nei metodi di correzione dell'illuminazione non uniforme.

La Figura 2.3 illustra come questo modello si manifesti nelle immagini digitali a colori, rappresentate da molteplici canali spettrali (RGB) e, opzionalmente, da un canale aggiuntivo di trasparenza (RGBA).

2.4.1 Rappresentazione Digitale e Canali Spettrali

Nelle immagini digitali a colori, la funzione $f(x, y)$ della Equazione 2.1 è rappresentata da molteplici canali spettrali. Nello standard **RGB**, ogni pixel memorizza tre campioni indipendenti:

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

corrispondenti alle intensità delle componenti rossa (*Red*), verde (*Green*) e blu (*Blue*). Nelle immagini di tipo **RGBA**, si aggiunge un quarto canale:

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

dove $A(x, y)$ rappresenta il canale **alfa** (*Alpha*), responsabile di codificare l'opacità del pixel. Per convenzione, il valore $A = 0$ indica un pixel completamente trasparente, mentre $A = 255$ (o 1) rappresenta un pixel completamente opaco.

Pertanto, un'immagine digitale a colori è strutturata come una matrice multidimensionale di dimensioni:

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

in cui ogni posizione (x, y) memorizza i campioni associati alle proprietà ottiche di quella coordinata spaziale.

La conversione tra intervalli di intensità è diretta ed è data da:

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

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

Convenzione di rappresentazione in `morph.hpp`: la libreria di questo libro adotta una convenzione unica (Tabella 2.1), senza le ambiguità di intervallo e ordine dei canali che emergono combinando diverse librerie Python.

Tabella 2.1: Convenzione di rappresentazione delle immagini in morph.hpp — intervallo, ordine delle bande, numero di canali e layout di memoria.

Aspetto	`morph.hpp`
Tipo di campione	`unsigned char` (`uint8`), intervallo $[0, 255]$
Ordine delle bande	**RGB** (tramite `stb_image`; senza inversione BGR)
Numero di canali	`1` (toni di grigio) o `3` (RGB)
Layout in memoria	$H \times W \times C$ interleaved (`data[(y*w + x)*channels + c]`)

La `struct Image` memorizza i pixel in un `std::vector<unsigned char>` lineare, con i campi `h`, `w` e `channels`. La funzione `mm::read` decodifica con `stb_image` forzando 3 canali (o 1, quando `grayscale=true`); pertanto **un eventuale canale alfa del file viene scartato in lettura**. Per lavorare in virgola mobile, vale la medesima relazione $f_{\text{norm}} = f/255$.

Nella cella di esempio seguente, la versione RGBA ($M \times N \times 4$) viene costruita sovrapponendo un canale alfa sintetico all'array letto — il kernel del notebook è Python e la visualizzazione riceve l'`ndarray` in questo layout $H \times W \times C$.

2.4.2 Preparazione dell'Ambiente Pratico

Il blocco seguente carica il modulo `morph.py` dal repository e dimostra, per un pixel sintetico, le diverse convenzioni di scala e ordine dei canali adottate dalle principali librerie di elaborazione delle immagini (PDI).

```
import os, urllib.request

os.makedirs("tmp/state", exist_ok=True)  # artefatti di build del percorso C++ (.cpp, binario,
PNG)

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")

# Il kernel è Python anche nel percorso C++: `mm` (morph.py) è usato dagli
# simulatori, dalla visualizzazione delle figure che il binario C++ genera e dallo
# stato mm::Image tra celle. cpp=True scarica anche il percorso compilato
# (morph.hpp + stb_image*.h), usato nel #include delle celle %%writefile *.cpp.
import config
config.setup(demo=True, cpp=True)
from morph import mm
```

```
Ambiente pronto. Morph: 1.1.9 | OpenCV: 5.0.0

Convenzioni di scala per libreria
NumPy / Pillow / OpenCV (uint8): [200 100  50] → [0, 255]
scikit-image / PyTorch (float):  [0.78431373 0.39215686 0.19607843] → [0.0, 1.0]
OpenCV: attenzione - legge in BGR: [ 50 100 200] (canali invertiti)
```

2.4.3 Implementazione della Lettura di Immagini in `morph.hpp`

Il seguente estratto mostra il codice sorgente della funzione `mm::read`, consentendo di verificare direttamente come la libreria `morph.hpp` implementa la lettura delle immagini.

```
# La morph.hpp è header-only; sotto, il corpo della funzione 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 e RGBA: Esempio Pratico con l'Immagine Mandrill

L'esempio seguente carica l'immagine Mandrill in formato RGB, costruisce artificialmente un canale alfa con gradiente orizzontale e visualizza entrambe le rappresentazioni, illustrando concretamente le strutture matriciali $M \times N \times 3$ e $M \times N \times 4$.

```
%%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 <string>
#include <filesystem>

int main() {
    // https://commons.wikimedia.org/wiki/File:Mandrill-k-means.png
    std::string url =
"https://upload.wikimedia.org/wikipedia/commons/a/ab/Mandrill-k-means.png";
    std::string caminho = "imagens/mandrill.png";

    mm::Image img_rgb = mm::read(url);  // (M, N, 3), uint8

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

    // 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 c = 0; c < 3; c++) {
                img_rgba.at(y, x, c) = img_rgb.at(y, x, c);
```

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

    std::cout << "RGB   -> shape=" << h << " x " << w << " x 3\n";
    std::cout << "RGBA  -> shape=" << h << " x " << w << " x 4\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)")
```

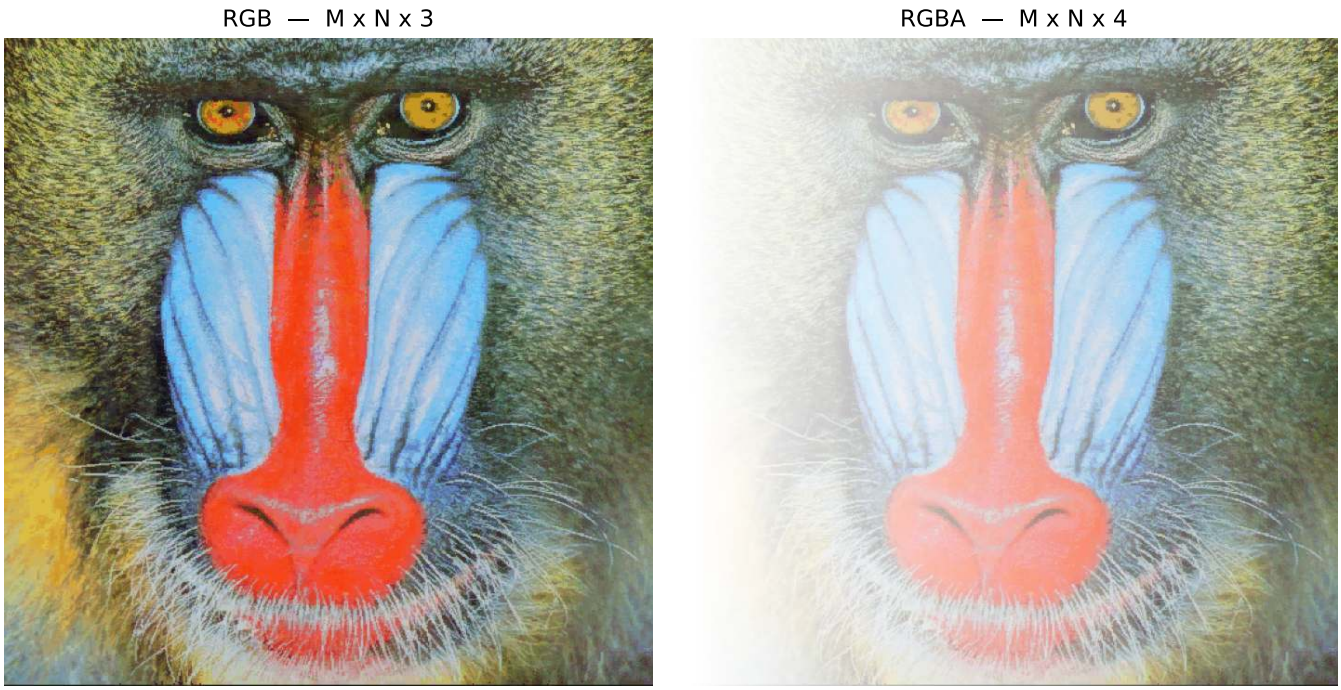


Figura 2.3: Imagem RGB (3 canais, $M \times N \times 3$) e versão RGBA (4 canais, $M \times N \times 4$) com transparência alfa gradual da esquerda para a direita. Imagem Mandrill — USC SIPI Image Database (domínio público).

2.5 Digitalizzazione: Campionamento e Quantizzazione

Per trasformare una scena continua in un'immagine digitale, sono necessari due processi: campionamento e quantizzazione.

2.5.1 Campionamento - Discretizzazione dello spazio

Il campionamento consiste nel misurare il valore della funzione $f(x, y)$ in punti equispaziati, formando una matrice di M righe (altezza) e N colonne (larghezza). Ciascun elemento di questa matrice è un **pixel**. La **risoluzione spaziale** è data da $M \times N$. Maggiore è la risoluzione, più dettagli spaziali vengono preservati, ma maggiore è anche il costo computazionale e di archiviazione.

2.5.2 Quantizzazione - Discretizzazione dell'intensità

La quantizzazione associa a ciascun pixel un valore numerico discreto, generalmente rappresentato da un intero di b bit. La **profondità di bit** definisce il numero di livelli di intensità: 2^b. Le immagini in scala di grigi utilizzano solitamente 8 bit (256 livelli). Le immagini a colori utilizzano tre canali da 8 bit (24 bit in totale).

Illustrazione: Se si utilizza solo 1 bit per pixel (bianco e nero), si perdono tutte le tonalità intermedie. Con 2 bit (4 livelli) si percepiscono già degradé grossolani. Con 8 bit, l'occhio umano difficilmente percepisce la discretizzazione (visione continua).

> ⚠ Errore di quantizzazione
>
> **L'errore di quantizzazione** è la differenza tra il valore analogico reale e il valore discreto attribuito. Si manifesta come rumore di quantizzazione, visibile in regioni con gradiente uniforme quando si utilizzano pochi bit.

2.5.3 Effetti del campionamento e della quantizzazione

Gli esperimenti che seguono mostrano come la riduzione della risoluzione spaziale (sottocampionamento) e della profondità di bit degradino la qualità visiva. Utilizzare il codice per esplorare diversi fattori e livelli di grigio.

```
%%writefile tmp/mm_out_1.cpp
//| quarto-raw: true

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

int main() {
    // Imagem de exemplo (barbudo-rajado) - base das figuras de amostragem,
    // quantização e transformações geométricas deste capítulo.
    mm::Image img_color =
mm::read("https://upload.wikimedia.org/wikipedia/commons/c/c5/Area_de_Prote%C3%A7%C3%A3o_Ambiental_Quilo
    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 << "x" << img_color.w << "x" <<
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: 3067x2047x3
```

L'esperimento presentato in Figura 2.4 illustra il compromesso tra la **risoluzione spaziale** e il **costo di archiviazione** in memoria. Il codice utilizza la tecnica di sottocampionamento per fette (*slicing*) per ridurre la matrice originale di pixel secondo un fattore f, risultando in un risparmio di memoria — ad esempio, un fattore $f = 8$ riduce la dimensione del dato di 64 volte (8^2). Per fini di confronto visivo, le immagini ridotte vengono ripristinate alle dimensioni originali (512×512) tramite l'interpolazione per **vicino più prossimo** (`nearest`). Questo processo non recupera l'informazione persa, ma rende evidente l'effetto di ***aliasing*** e la struttura a blocchi (pixelizzazione) generata dalla bassa densità di dati della matrice campionata.

```
%%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

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

// Função equivalente a subsample_simple
```

```cpp
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
    int mem_kb = (reduced.h * reduced.w * reduced.channels) / 1024;
    std::string label = std::to_string(reduced.w) + "x" + std::to_string(reduced.h) + ", " +
std::to_string(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 já está inicializado

    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 result = subsample_simple(img_gray, f);
        imgs_list.push_back(result.first);
        titles_list.push_back(result.second);
    }

    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)")
```

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

L'esperimento in Figura 2.5 si concentra sulla **quantizzazione dell'intensità**, il processo di discretizzazione dell'ampiezza della funzione $f(x, y)$. Mentre il sottocampionamento influisce sulla griglia spaziale, la riduzione della profondità di bit limita la quantità di livelli di grigio disponibili per rappresentare la luminosità.

Riducendo la profondità da 8 bit (256 livelli) a valori inferiori, emerge l'effetto di **posterizzazione**, in cui i gradienti morbidi di una scena vengono sostituiti da transizioni brusche. Al limite di 1 bit, l'immagine diventa strettamente binaria, preservando solo la silhouette e perdendo i dettagli di texture e volume.

```
%%writefile tmp/fig_02_quantizacao.cpp
#define MM_OUT "tmp/fig_02_quantizacao.png"
// g++ -std=c++17 -o quantizacao quantizacao.cpp -I. -L. -lmorph

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

struct QuantizedResult {
    mm::Image img;
    std::string label;
};

// Função que reduz a profundidade de bits e calcula metadados de memória
QuantizedResult quantize_simple(const mm::Image& image, int bits) {
    int levels = 1 << bits;  // 2 ** bits

    // Normalização e quantização uniforme
    mm::Image quantized(image.h, image.w);
    for (int y = 0; y < image.h; y++) {
        for (int x = 0; x < image.w; x++) {
            double val = std::floor((double)image.at(y, x) / 256.0 * levels) / levels * 255.0;
            quantized.at(y, x) = (unsigned char)val;
        }
    }

    // Cálculo de memória em KB (1 byte por amostra)
    int mem_kb = (quantized.h * quantized.w * quantized.channels) / 1024;
    std::string label = std::to_string(bits) + " bits (" + std::to_string(levels) + "
níveis)\n" +
                        std::to_string(mem_kb) + " KB";

    return {quantized, 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]

    // Lista de bits para teste (8 é o padrão, 1 é o binário)
    std::vector<int> bits_test = {8, 4, 2, 1};

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

    for (int b : bits_test) {
        QuantizedResult r = quantize_simple(img_gray, b);
        imgs_q.push_back(r.img);
        titles_q.push_back(r.label);
    }

    // Exibe as imagens quantizadas lado a lado
    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)")
```

Figura 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 Analisi Tecnica

- **Dominio vs. Codominio**: Si noti che la risoluzione spaziale (dimensioni della matrice) rimane costante a 512x512; ciò che cambia è solo il codominio della funzione dell'immagine.
- **Costanza della Memoria**: Si osservi nei titoli che la dimensione in KB non diminuisce. Ciò accade perché NumPy memorizza ogni pixel quantizzato in un contenitore a 8 bit (`uint8`), indipendentemente dal fatto che il valore reale sia solo 0 o 1.
- **Percezione**: Il degrado visivo diventa critico al di sotto di 4 bit, dove l'occhio umano inizia a percepire i "confini" artificiali creati dalla mancanza di tonalità intermedie.

La limitazione dei tipi di dati più piccoli di un byte nell'ecosistema Python/NumPy è dovuta all'architettura hardware, che indirizza la memoria in blocchi di 8 bit (byte). Per mantenere la compatibilità con OpenCV e garantire l'efficienza, anche gli elementi binari vengono mappati in contenitori da 1 byte (`uint8` o `bool8`).

Sebbene linguaggi come l'ANSI C consentano la compressione di 8 pixel per byte (*bit-packing*), tale approccio richiede una decompressione costante per i calcoli e impone un'elevata complessità nella gestione dei puntatori. Come indicato in Tabella 2.2, si privilegia l'uso di `uint8` per la facilità di accesso ai vicini e per la versatilità nelle trasformazioni geometriche. Inoltre, i metodi nativi di NumPy e OpenCV eseguono l'elaborazione internamente a basso livello (C/C++), rendendo le operazioni vettorizzate più rapide rispetto alle implementazioni manuali con cicli annidati in Python.

Tabella 2.2: Confronto tra strategie di compressione ed efficienza di elaborazione.

Caratteristica	Python (NumPy/OpenCV)	ANSI C (Bit-packing)
Unità minima	1 Byte (8 bit)	1 Bit
Memoria (binaria)	256 KB (per 512x512)	32 KB (per 512x512)
Velocità	Elevata (Vettorizzazione in C)	Variabile (Lenta in caso di bit-shift)
Complessità	Bassa: Metodi pronti	Alta: Puntatori e Maschere

2.6 Relazioni tra Pixel - Topologia dell'Immagine

I pixel non sono elementi isolati; le loro posizioni relative definiscono concetti importanti per l'elaborazione.

2.6.1 Vicinato

Dato un pixel di coordinate (x, y), si definiscono due tipi principali di vicinato (per immagini su *grid* rettangolare):

- **Vicinato-4** (von Neumann): include i pixel nelle posizioni $(x-1, y)$, $(x+1, y)$, $(x, y-1)$, $(x, y+1)$.
- **Vicinato-8** (Moore): include tutti gli otto pixel adiacenti (aggiunge i quattro diagonali).

La scelta del vicinato influenza operazioni come il rilevamento dei bordi, il calcolo dei gradienti e a connettività.

```
%%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"
#include <iostream>

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)")
```

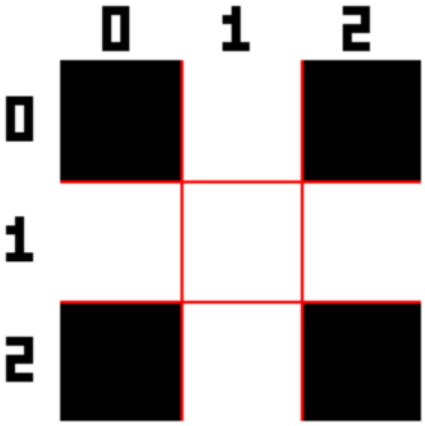


Figura 2.6: Ilustração de vizinhanças 4 em uma matriz 3x3. No centro (1,1), o pixel de interesse.

2.6.2 Adiacenza, connettività e percorsi

Due pixel sono **adiacenti** se sono in contatto secondo un intorno definito **e** soddisfano un criterio di valore (es.: stesso livello di intensità). Una **connettività** definisce una relazione di equivalenza tra pixel che formano una regione connessa. Un **percorso** è una sequenza di pixel adiacenti.

La **connettività-4** (N4) e la **connettività-8** (N8) possono produrre risultati differenti nella segmentazione e nel calcolo delle componenti connesse (etichettatura). Ad esempio, un motivo a scacchiera può essere completamente sconnesso in N4, ma totalmente connesso in N8.

2.6.3 Distanze tra pixel

Le metriche di distanza sono fondamentali per quantificare la prossimità fisica e la connettività tra gli elementi che compongono la griglia digitale. Come dimostrato nella Tabella 2.3, la scelta della metrica definisce il costo di spostamento tra pixel e altera il comportamento degli algoritmi di segmentazione e di analisi morfologica.

Per misurare la distanza tra due pixel $p(x_1, y_1)$ e $q(x_2, y_2)$, si utilizzano diverse funzioni metriche che impongono vincoli di movimento distinti sulla *griglia*:

Tabella 2.3: Confronto tra metriche di distanza applicate alla maglia di pixel.

Metrica	Definizione	Interpretazione
Euclidea	$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$	Distanza **esatta** in linea retta (continua)
Manhattan (*City block*)	$\lvert x_1 - x_2\rvert + \lvert y_1 - y_2\rvert$	Movimenti orizzontali + verticali
Chebyshev (*Scacchiera*)	$\max(\lvert x_1 - x_2\rvert, \lvert y_1 - y_2\rvert)$	Maggiore spostamento tra gli assi

Queste distanze sono applicate in diversi contesti di PDI, inclusi algoritmi di interpolazione geometrica, trasformate della distanza, crescita di regioni e analisi delle forme.

i Esempio pratico

Considerando due pixel con spostamenti relativi $\Delta x = 3$ e $\Delta y = 4$:

- **Euclidea**: $\sqrt{3^2 + 4^2} = 5$ (ipotenusa del triangolo rettangolo).
- ***Manhattan***: $3 + 4 = 7$ (somma dei cateti).
- ***Chebyshev***: $\max(3, 4) = 4$ (predominio dello spostamento maggiore).

2.7 Archiviazione delle Immagini

La scelta del formato di file rappresenta un passo decisivo nel flusso di elaborazione, poiché determina come i dati di campionamento e quantizzazione verranno preservati o scartati. Come presentato in Tabella 2.4, ciascuna estensione bilancia in modo distinto la fedeltà dei dati e l'efficienza di archiviazione.

Tabella 2.4: Principali formati di archiviazione delle immagini digitali e le loro applicazioni nell'elaborazione digitale delle immagini (PDI).

Formato	Caratteristiche	Uso tipico
PGM	Formato semplice di mappa dei grigi (testo o binario).	Ricerca accademica e strumenti Unix.
BMP	Non compresso (o compressione semplice).	Windows, applicazioni legacy.
PNG	Compressione **senza perdita** (*lossless*).	Web, immagini con trasparenza.
JPEG	Compressione **con perdita** (*lossy*), ideale per fotografie.	Foto, fotocamere digitali.
TIFF	Supporta più livelli e compressione variabile.	Editoria, archiviazione.
RAW	Dati grezzi del sensore, senza elaborazione.	Fotografia professionale.
DICOM	Standard medico con metadati clinici incorporati (paziente, apparecchiatura, protocollo).	Radiologia, tomografia, risonanza magnetica.

I metadati di un'immagine includono parametri quali larghezza, altezza, profondità di bit e codifica del colore. Nei formati scientifici vengono preservate anche le informazioni di calibrazione e i dettagli dell'acquisizione. Utilizzando la funzione `mm::read()`, la libreria `morph` preserva automaticamente questi dati affinché vengano rispettate le proprietà originali dell'immagine.

In contesti scientifici e ospedalieri, si privilegia lo standard **DICOM** (*Digital Imaging and Communications in Medicine*) per garantire l'assenza di perdita di precisione diagnostica. Repository pubblici come **The Cancer Imaging Archive (TCIA)**, la **Alzheimer's Disease Neuroimaging Initiative (ADNI)** e **PhysioNet** mettono a disposizione vasti set di dati in questo formato, inclusi metadati clinici anonimizzati essenziali per la ricerca scientifica.

2.7.1 Esempio: Estrazione di Metadati e Localizzazione GPS

A differenza della matrice di pixel pura ottenuta dalla lettura convenzionale — come nell'immagine dell'uccello presentata all'inizio di questo capitolo —, l'uso dell'argomento `pil=True` nel metodo `mm::read()` altera la natura dell'oggetto restituito (vedere Figura 2.7). Mentre il comportamento predefinito (`pil=False`) restituisce un `numpy.ndarray` RGB, la lettura con `pil=True` restituisce un oggetto specializzato della libreria Pillow, in grado di interpretare l'intestazione **EXIF**.

L'intestazione **EXIF** (*Exchangeable Image File Format*) funziona come un archivio tecnico della cattura, consentendo all'oggetto Pillow di interpretare un'ampia gamma di informazioni che vanno ben oltre le coordinate GPS. Utilizzando `pil=True`, il sistema acquisisce accesso al "DNA" dell'immagine, inclusi metadati hardware (marca e modello della fotocamera), impostazioni ottiche (apertura, lunghezza focale e tempo di esposizione) e parametri di illuminazione (uso del flash e bilanciamento del bianco).

Questa distinzione è importante nell'elaborazione digitale delle immagini (PDI), poiché trasforma la matrice di campionamento in un insieme di dati contestualizzato, dove le caratteristiche fisiche del sensore e dell'obiettivo possono essere utilizzate per normalizzare le luminosità o correggere distorsioni geometriche.

i Percorso C++: perché l'estrazione EXIF rimane in Python

L'estrazione dei metadati è **parsing del contenitore** — decodifica dell'intestazione EXIF incorporata nel file — e non elaborazione di pixel: è ortogonale alla pipeline di campionamento e quantizzazione. `morph.hpp` decodifica le immagini con `stb_image`, che restituisce solo la matrice dei pixel e scarta l'intestazione EXIF. Poiché il kernel di questo notebook è Python anche nel percorso C++, questo esempio utilizza la lettura in modalità Pillow (`pil=True`) in entrambi i percorsi. In un programma C++ autonomo, il percorso equivalente sarebbe quello di integrare una libreria dedicata: `easyexif` (lettura di EXIF/GPS in JPEG, header singolo, zero dipendenze) o `exiv2` (lettura **e** scrittura, inclusi IPTC/XMP).

```
%%writefile tmp/fig_01_natureza.cpp
#define MM_OUT "tmp/fig_01_natureza.png"
//| label: fig-01-natureza
//| fig-cap: "Area de Proteção Ambiental Quilombos do Médio Ribeira - Barbudo-rajado
(Malacoptila striata). Crédito: Thomas Fuhrmann (CC BY-SA 4.0)."
//| echo: true

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

int main() {
    // 1. Leitura direta da URL (mm::read aceita URLs diretamente)
    mm::Image img_obj =
mm::read("https://upload.wikimedia.org/wikipedia/commons/c/c5/Area_de_Prote%C3%A7%C3%A3o_Ambiental_Quilo

    // A conversão para NumPy não é necessária - mm::Image já armazena os dados
    mm::Image img_numpy = img_obj;

    // 2. Extração de EXIF e GPS não é suportada por mm::Image
    // (mm::read não preserva metadados EXIF)

    // 3. Diagnóstico de tipos, dimensões e acesso a pixels
    std::cout << "\nTipo PIL  : mm::Image   | Dimensões (x,y): " << img_obj.w << ", " <<
img_obj.h << "\n";
    std::cout << "Pillow (0,0): ";
    if (img_obj.channels >= 3) {
        std::cout << "(" << (int)img_obj.at(0, 0, 0) << ", " << (int)img_obj.at(0, 0, 1) << ",
" << (int)img_obj.at(0, 0, 2) << ")";
    } else {
        std::cout << (int)img_obj.at(0, 0);
    }
    std::cout << "\n";
    std::cout << "Tipo NumPy: mm::Image | Dimensões [y,x,c]: " << img_numpy.h << ", " <<
img_numpy.w << ", " << img_numpy.channels << "\n";
    std::cout << "NumPy [0,0]: ";
    if (img_numpy.channels >= 3) {
        std::cout << "(" << (int)img_numpy.at(0, 0, 0) << ", " << (int)img_numpy.at(0, 0, 1)
<< ", " << (int)img_numpy.at(0, 0, 2) << ")";
    } else {
        std::cout << (int)img_numpy.at(0, 0);
    }
    std::cout << "\n";

    // 4. Exibição
    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"
```

```

Tipo PIL  : mm::Image   | Dimensões (x,y): 2047, 3067
Pillow (0,0): (58, 95, 2)
Tipo NumPy: mm::Image | Dimensões [y,x,c]: 3067, 2047, 3
NumPy [0,0]: (58, 95, 2)
```

```
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)")
```

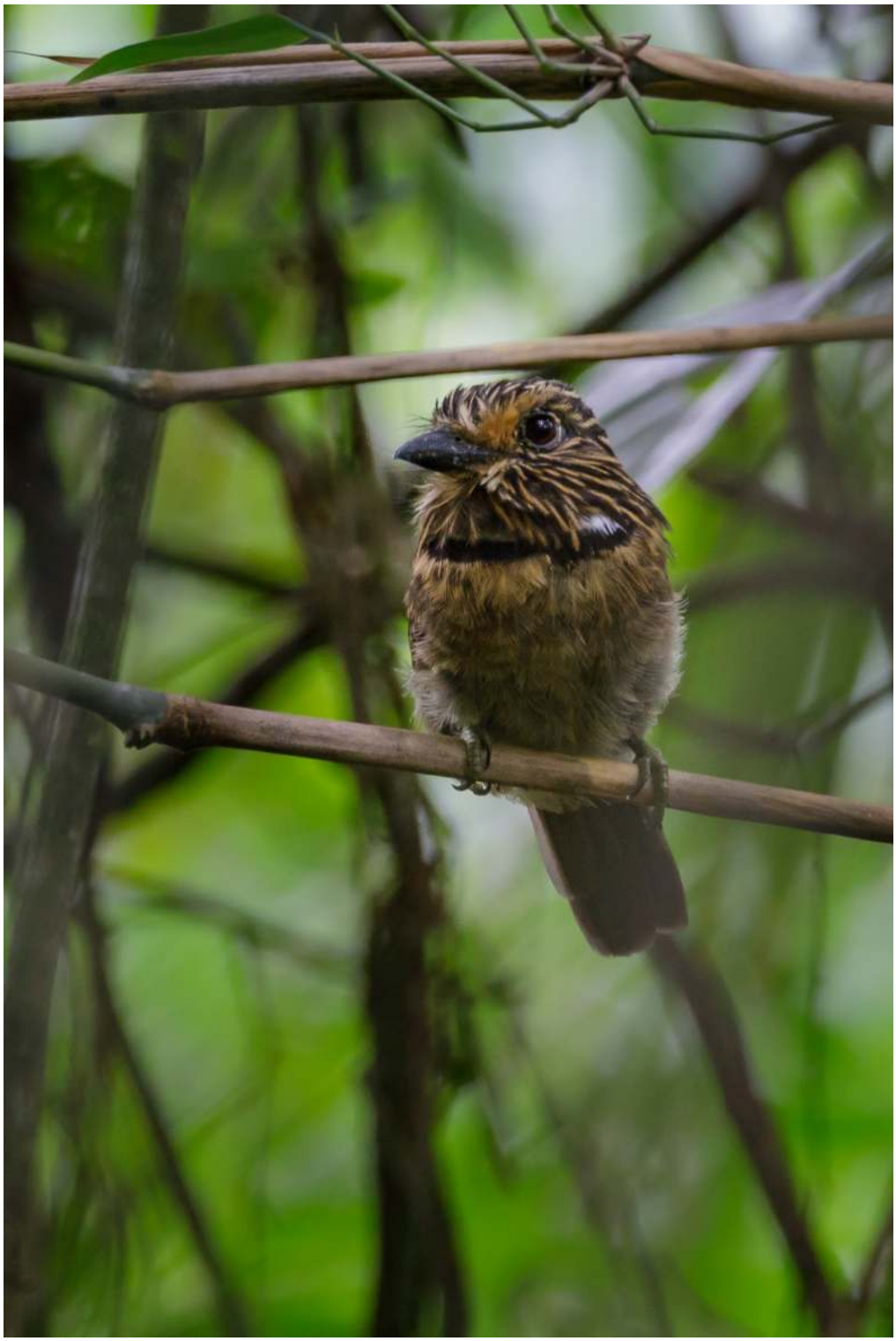


Figura 2.7: Area de Proteção Ambiental Quilombos do Médio Ribeira - Barbudo-rajado (Malacoptila striata). Crédito: Thomas Fuhrmann (CC BY-SA 4.0).

Nota Pedagogica: La sottile differenza delle dimensioni

Si noti che la rappresentazione delle dimensioni cambia a seconda della struttura dati utilizzata:

- **In Pillow (`.size`):** Restituisce `(Larghezza, Altezza)` — nell'esempio: `(2047, 3067)`. È una prospettiva orientata al file immagine.
- **In NumPy (`.shape`):** Segue la convenzione matematica delle matrici: `(Righe/Altezza,`

`Colonne/Larghezza, Canali)` — nell'esempio: `(3067, 2047, 3)`.
Questa distinzione è fondamentale per evitare errori di indicizzazione nell'implementazione di filtri manuali. Mentre l'oggetto Pillow trasporta il **"dove"** e il **"quando"** (contesto), l'array NumPy trasporta il **"quanto"** di luce (intensità) presente in ciascun punto dell'immagine.

2.7.2 Perché questa separazione è importante?

Caricando un'immagine tramite il percorso convenzionale (`pil=False`), il risultato è un `numpy.ndarray`, che contiene strettamente i valori numerici risultanti dalla **quantizzazione** e dal **campionamento**. Tuttavia, utilizzando `pil=True`, `mm::read()` restituisce un oggetto della classe `PIL.JpegImagePlugin.JpegImageFile`.

Questa classe mantiene il file "aperto" per consentire l'accesso al contesto dell'acquisizione prima che i dati vengano convertiti in una matrice grezza. Si noti che il pixel `(0, 0)` è identico nelle due rappresentazioni — `(58, 96, 0)` in Pillow e `[58, 96, 0]` in NumPy —, confermando che entrambe descrivono gli stessi dati, solo con interfacce diverse. Questa separazione è fondamentale: i pixel servono agli algoritmi; i metadati servono per la georeferenziazione, la catalogazione scientifica e le correzioni basate sull'hardware di acquisizione.

Per ispezionare tutti i metadati EXIF di un oggetto 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 Trasformazioni Geometriche Fondamentali

Le trasformazioni geometriche modificano la posizione dei pixel, mantenendo i valori di intensità. Sono fondamentali per l'allineamento, la correzione delle distorsioni e l'aumento dei dati (*data augmentation*) nell'apprendimento automatico.

Una **trasformazione affine** è qualsiasi mappatura che preservi la collinearità (punti su una retta rimangono su una retta) e i rapporti di distanza tra punti collineari. In 2D, ogni trasformazione affine può essere espressa in **coordinate omogenee** tramite una 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 sottomatrice 2×2 in alto a sinistra $\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$ controlla rotazione, scala e taglio; il vettore $(t_x, t_y)^\top$ controlla la traslazione. Le trasformazioni geometriche più utilizzate nell'elaborazione digitale delle immagini — traslazione, rotazione e scala — sono casi particolari di $\mathbf{T}$, e possono essere **composte** tramite moltiplicazione di matrici, nell'ordine $\mathbf{T} = \mathbf{T}_n \cdots \mathbf{T}_2 \mathbf{T}_1$.

i Trasformazione inversa e interpolazione

Nell'implementazione pratica (`cv2.warpAffine`), si applica la **trasformazione inversa**: per ogni pixel (x', y') dell'immagine di destinazione, si calcola la posizione di origine $(x, y) = \mathbf{T}^{-1}(x', y')$ e si interpola il valore. Ciò evita i buchi nell'immagine risultante causati dalla mappatura diretta di pixel interi su

posizioni non intere.

2.8.1 Traslazione

La traslazione è la trasformazione affine più semplice: sposta tutti i pixel di un vettore (t_x, t_y). In coordinate omogenee, è espressa dalla 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 terza riga della matrice garantisce che l'operazione rimanga nello spazio affine, consentendo di combinare traslazione, rotazione e scala tramite semplice moltiplicazione di matrici. In pratica, `cv2.warpAffine` utilizza solo le prime due righe (matrice 2×3), poiché la terza è sempre $[0, 0, 1]$.

I pixel spostati oltre l'area originale vengono scartati; le aree scoperte vengono riempite con 0 (nero). Si veda un esempio nella Figura 2.8.

```
%%writefile tmp/fig_02_translacao.cpp
#define MM_OUT "tmp/fig_02_translacao.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-translacao
    //| fig-cap: "Exemplo de translação da imagem do pássaro com deslocamentos (50,50) e
    (100,50)."
    //| echo: true
    //| output: true
    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)")
```

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

2.8.2 Rotazione

La rotazione di un angolo θ attorno a un punto centrale (c_x, c_y) è composta da tre trasformazioni affini: traslazione verso l'origine, rotazione pura e traslazione di ritorno. La matrice risultante è:

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

Nell'implementazione, `cv2.getRotationMatrix2D` genera direttamente le prime due righe di $\mathbf{T}_{\text{rot}}$ (matrice

2×3 per `warpAffine`), accettando anche un fattore de escala s que multiplica $\cos\theta$ e $\sin\theta$. Si veda l'esempio nella Figura 2.9.

Poiché la rotazione sposta i pixel in nuove posizioni non intere, `cv2.warpAffine` deve stimare il colore di ogni pixel di destinazione a partire dai vicini — processo chiamato **interpolazione**. Il parametro `interp` controlla questo comportamento:

- **`nearest`** (`INTER_NEAREST`): assegna il valore del pixel più vicino. Veloce, ma produce un effetto a scalini (*aliasing*) sui bordi diagonali.
- **`bilinear`** (`INTER_LINEAR`, predefinito): media ponderata dei 4 vicini più prossimi. Bilancia qualità e prestazioni — adatto alla maggior parte dei casi.
- **`bicubic`** (`INTER_CUBIC`): considera i 16 vicini su una superficie cubica. Produce bordi più morbidi, al costo di un maggiore carico di elaborazione.

```
%%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]

    // img_gray is already available

    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)")
```

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

2.8.3 Scala (Ridimensionamento)

Il ridimensionamento mediante fattori (s_x, s_y) è una trasformazione affine con matrice:

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

Quando $s > 1$ (ingrandimento), i pixel dell'immagine di destinazione mappano su posizioni non intere nell'origine — richiedendo interpolazione per stimare il valore. Quando $s < 1$ (riduzione), più pixel di origine contribuiscono a un singolo pixel di destinazione — richiedendo **decimazione**. Gli stessi tre metodi descritti nella rotazione sono disponibili in `mm::resize`, con la differenza che qui l'impatto visivo è più percettibile: nell'ingrandimento, `nearest` produce un effetto a blocchi (*pixelation*), mentre `bicubic` preserva meglio la nitidezza dei bordi, come indicato nella Tabella 2.5:

Tabella 2.5: Metodi di interpolazione disponibili in mm.resize e i rispettivi numeri di vicini utilizzati nel calcolo.

Metodo	Vicini utilizzati	Caratteristica
`'nearest'`	1	Veloce; produce effetto a blocchi (*pixelation*)
`'bilinear'`	4	Buon compromesso qualità/costo; bordi morbidi
`'bicubic'`	16	Maggiore nitidezza; preferito nei software professionali

Il parametro `size_or_factor` accetta sia uno scalare (fattore uniforme, es. `0.5` per ridurre della metà) sia una tupla `(larghezza, altezza)` per dimensioni assolute.

```
%%writefile tmp/fig_02_escala_detalhe.cpp
#define MM_OUT "tmp/fig_02_escala_detalhe.png"
//| 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

#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]

    // img_gray is already available

    // 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)")
```

Figura 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 Taglio (*Shear*)

Il taglio è una trasformazione affine che distorce l'immagine spostando ogni pixel proporzionalmente alla sua posizione su un asse. La matrice generale combina taglio orizzontale (sh_x) e verticale (sh_y):

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

Per $sh_x \neq 0$ e $sh_y = 0$, ogni riga viene spostata orizzontalmente in modo proporzionale alla sua posição verticale — producendo l'effetto di "inclinazione" caratteristico. Si veda l'esempio nella Figura 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 <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, 0.0);
    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)")
```

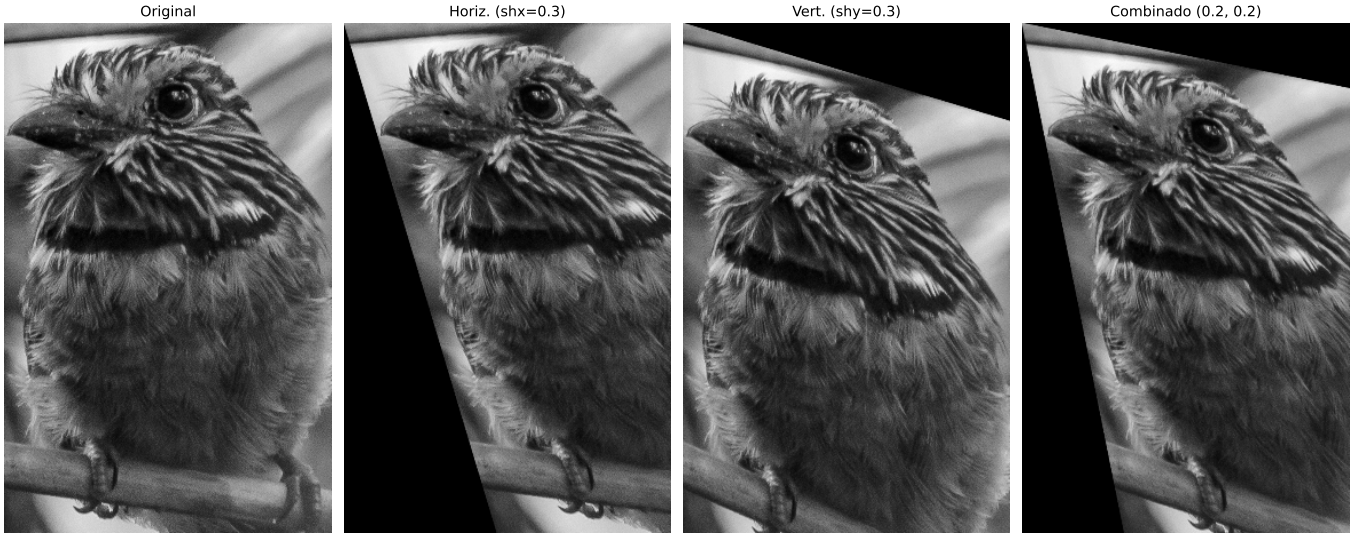


Figura 2.11: Exemplo de cisalhamento da imagem do pássaro: horizontal (shx=0.3), vertical (shy=0.3) e combinado (shx=0.2, shy=0.2).

2.9 Riassunto

In questo capitolo sono stati presentati i fondamenti della digitalizzazione e della topologia delle immagini:

- **Formazione dell'immagine:** $f(x, y) = i(x, y) \cdot r(x, y)$.
- **Campionamento:** discretizzazione dello spazio → risoluzione spaziale $M \times N$.
- **Quantizzazione:** discretizzazione dell'intensità → profondità di bit b.
- **Relazioni topologiche:** vicinato-4, vicinato-8, connettività, distanze (Euclidea, Manhattan, Chebyshev).
- **Trasformazioni geometriche:** traslazione, rotazione, scala (con interpolazione bilineare o vicino più prossimo).
- **Formati di file:** BMP, PNG, JPEG, TIFF, RAW; ciascuno con diversi compromessi tra qualità e dimensione.

Il Capitolo 3 tratterà **operazioni spaziali** come convoluzione, filtraggio e morfologia matematica (erosione, dilatazione).

2.10 🤖 Uso del Gemini Notebook come Tutor Complementare

In questa edizione, incoraggiamo l'uso del **Gemini Notebook** come strumento complementare di apprendimento. Questo strumento di IA utilizza esclusivamente i documenti forniti dall'autore come base di conoscenza, garantendo risposte coerenti con il contenuto del libro.

Per ogni capitolo, abbiamo preparato un progetto specifico sulla piattaforma. Per un'esperienza di studio ampliata, utilizza il seguente accesso:

🎓 Studia con il Tutor Intelligente

Per interagire con il contenuto di questo capitolo, accedi al seguente link. L'ambiente contiene materiali didattici in diversi formati, generati a partire dal **PDF** del capitolo. Sulla piattaforma, esplora in particolare le opzioni **Guida allo Studio** e **Conversazione** per approfondire la tua comprensione.
🚀 ACCEDI AL GEMINI NOTEBOOK: CAPITOLO 02

🌐 Lingua e Linguaggio di Programmazione

Il progetto di questo capitolo nel Gemini Notebook è stato costruito esclusivamente con il testo in **portoghese** e gli esempi di codice in **Python**. Se stai studiando dall'edizione in inglese o francese, oppure seguendo il percorso in C++, le risposte del tutor potrebbero non corrispondere esattamente alla versione che stai leggendo.

⚠️ Avviso sul Contenuto Generato dall'IA

L'IA è una potente alleata nello studio, ma il contenuto generato può contenere **errori o imprecisioni**. Consulta sempre **libri, articoli scientifici e altre fonti accademiche affidabili** per validare le informazioni. Quando possibile, esegui gli esempi pratici forniti in questo capitolo per verificarne i risultati.

2.11 Elenco di Esercizi

1. **(15%)** Spiega, con le tue parole, la differenza tra **campionamento** e **quantizzazione**. Fornisci un esempio concreto di ciascuno nel contesto di un'immagine digitale.
2. **(15%)** Considera un'immagine con risoluzione spaziale di 1024×768 pixel e profondità di 24 bit (8 bit per canale RGB). Calcola la dimensione totale non compressa dell'immagine in byte e in megabyte.
3. **(20%)** Utilizzando il codice del laboratorio, modifica il fattore di sottocampionamento a 3 e a 6. Descrivi visivamente cosa accade ai bordi degli oggetti. Che cos'è l'effetto di *aliasing*?
4. **(20%)** Per l'immagine in toni di grigio, applica la quantizzazione con 3 bit (8 livelli) e 5 bit (32 livelli). Confronta i risultati e spiega perché 5 bit possono già essere considerati sufficienti per molte applicazioni.
5. **(15%)** Dati due pixel $A = (10, 20)$ e $B = (15, 25)$, calcola le distanze Euclidea, di Manhattan e di Chebyshev tra di essi.
6. **(15%)** Usando la funzione `mm::rotate`, ruota l'immagine dell'uccello ad angoli di 90°, 180° e 270° con interpolazione bilineare. Confronta con la rotazione usando `method='nearest'`. In quali situazioni l'interpolazione del vicino più prossimo è ancora utile?

Riferimenti del Capitolo

La base teorica di questo capitolo si fonda sulle seguenti opere:

- Gonzalez (2018) per i concetti di campionamento, quantizzazione e relazioni tra pixel.

- Szeliski (2022) per trasformazioni geometriche e connettività.
- Bradski (2008) per l'implementazione pratica con OpenCV e `morph.py`.

2.12 💻 Parte pratica con esercizi di programmazione

🎯 Obiettivo di questo Quaderno

Il quaderno consente di sviluppare, validare, organizzare e testare soluzioni di **Esercizi di Programmazione (EP)** in ambienti interattivi, come Colab, con gli stessi casi di test di Moodle, copiandoli lì solo al momento di registrare il voto ufficiale.

Download

Scarica `morph.py` e `testsuite.py` eseguendo la cella qui sotto:

```
import os, urllib.request

os.makedirs("tmp/state", exist_ok=True)  # artefatti di build del percorso C++ (.cpp, binario,
PNG)

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")

# Il kernel è comunque Python nel percorso C++: `mm` (morph.py) è usato dai
# simulatori, dalla visualizzazione delle figure che il binario C++ genera e dallo
# stato mm::Image tra le celle. cpp=True scarica anche il percorso compilato
# (morph.hpp + stb_image*.h), usato nell'#include delle celle %%writefile *.cpp.
import config
config.setup(testsuite=True, cpp=True)
from morph import mm
from testsuite import TestSuite
```

```
Ambiente pronto. Morph: 1.1.9 | OpenCV: 5.0.0 | TestSuite: 1.1.2
```

Esecuzione dei Test

Per valutare i test, eseguire `TestSuite("EP04_01.extensão").run()` in una nuova cella, sostituendo l'estensione con quella del linguaggio utilizzato (`.py`, `.java`, `.c`, `.cpp`, `.js` o `.r`). Il sistema scarica i casi di test da GitHub, esegue il programma e calcola automaticamente il voto.

Per testare direttamente codice Python, senza salvare un file, utilizzare `run_code(codigo)` passando il codice come *stringa* in una variabile `codigo`:

```
codigo = """
from morph import mm
# ... tuo codice qui ...
"""
TestSuite("EP04_01").run_code(codigo)
```

2.12.1 EP02_01 ⊙ Regolazione di Luminosità e Contrasto

In questa attività, l'obiettivo è implementare un operatore puntuale per la **trasformazione lineare di intensità**, applicando la regolazione dinamica di luminosità e contrasto su un'immagine digitale.

2.12.1.1 📋 Linee Guida di Implementazione

L'algoritmo deve seguire il flusso di esecuzione di seguito:

1. **Dimensioni:** Leggere gli interi L (righe) e C (colonne) della matrice.
2. **Parametri:** Leggere il valore reale α (fattore di contrasto) e l'intero β (fattore di luminosità).
3. **Dati:** Leggere i valori interi della matrice originale.
4. **Mappatura:** Per ogni pixel p, calcolare il nuovo valore p' tramite l'equazione:

$$p' = \text{clip}(\text{round}(\alpha \cdot p + \beta))$$

5. **Output:** Visualizzare la matrice risultante con le dimensioni $L \times C$.

2.12.1.2 📌 Vincoli Computazionali

- **Arrotondamento (*Round*):** Si applica l'arrotondamento matematico all'intero più vicino prima della conversione di tipo.
- **Saturazione (*Clipping*):** I valori devono essere limitati all'intervallo $[0, 255]$ per preservare lo standard a 8 bit:

$$\text{clip}(x) = \max(0, \min(255, x))$$

- **Simulazione:** L'effetto dei parametri α e β sulla correzione istogrammatica può essere osservato nella Figura 2.12.

2.12.1.3 🧠 Fondamenti Teorici

Le modifiche alterano l'istogramma dell'immagine per regolare il profilo di illuminazione e la distinzione tonale.

Parametro	Funzione	Impatto Visivo
α (***Alpha***)	Scalare	Modula il **Contrasto**. Se $\alpha > 1$, espande l'istogramma; se $0 \leq \alpha < 1$, lo comprime.
β (***Beta***)	Additiva	Modula la **Luminosità**. Se positivo, trasla l'istogramma verso destra; se negativo, verso sinistra.
clip	Limitatore	Restringe la gamma dinamica, prevenendo errori di *underflow* e *overflow*.

2.12.1.4 📦 Specifica di Input e Output (VPL)

Input:

- Riga 1: Intero L.
- Riga 2: Intero C.
- Riga 3: Valori di `alpha` (α) e `beta` (β).
- Righe successive: Elementi numerici della matrice originale.

Output:

- Matrice trasformata strutturata in L righe e C colonne.

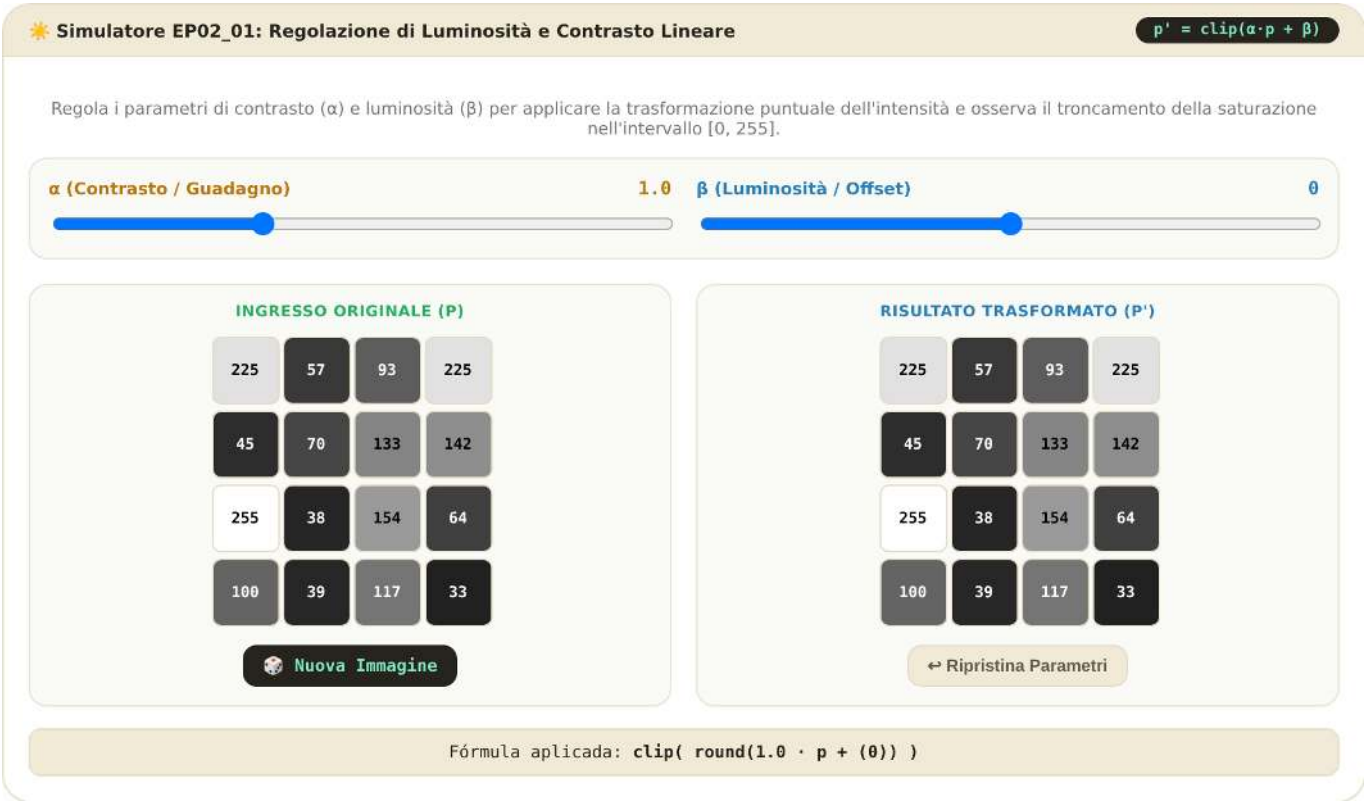
2.12.1.5 📌 Esempi

La tabella seguente presenta un esempio pratico del comportamento atteso dell'algoritmo, evidenziando l'azione degli operatori di arrotondamento e saturazione.

Input	Output	Osservazione
1 4 1.5 -30 0 100 180 255	0 120 240 255	Si noti l'effetto di saturazione sull'ultimo pixel

Esecuzione dei Test

Per valutare i test, eseguire `TestSuite("EP02_01.extensão").run()` in una nuova cella, sostituendo l'estensione con quella del linguaggio utilizzato (`.py`, `.java`, `.c`, `.cpp`, `.js` o `.r`). Il sistema scarica i casi di test da GitHub, esegue il programma e calcola automaticamente il voto.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.it/cap02/sim-ep0201-brilho-contraste.html

Figura 2.12: Simulatore EP02_01: Regolazione di Luminosità e Contrasto Lineare (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 🔬 Sottocampionamento Spaziale

In questa attività, devi implementare la riduzione della risoluzione spaziale di un'immagine attraverso il processo di sottocampionamento.

- Leggi due interi **L** e **C**, che rappresentano le dimensioni della matrice originale.
- Leggi un valore intero f ($f \geq 1$), che rappresenta il fattore di campionamento.
- Leggi i valori interi della matrice originale.
- La nuova immagine deve essere costruita selezionando il pixel nella posizione $(f \cdot i, f \cdot j)$ dell'immagine originale.
- Stampa la matrice risultante con le nuove dimensioni.
- Vedi una simulazione di questo EP in Figura 2.13.

📌 **Importante**:

- **Dimensioni Finali:** L'immagine campionata avrà dimensioni $\lceil L/f \rceil \times \lceil C/f \rceil$. Nel contesto della programmazione, ciò equivale alla dimensione risultante da uno slicing con passo f.
- **Implementazione:** Non utilizzare funzioni pronte di librerie di elaborazione delle immagini (come OpenCV o PIL) per il ridimensionamento. Implementa la logica di selezione dei pixel manualmente o tramite slicing di matrici.
- **Aliasing:** Nota che questo processo può causare l'effetto di *aliasing* (effetto a scaletta), dove i dettagli fini vengono persi o compaiono pattern indesiderati.

2.12.2.1 🧠 Discretizzazione dello Spazio

Il sottocampionamento riduce la risoluzione spaziale di un'immagine, selezionando solo un pixel ogni f pixel in ciascuna direzione. È il processo inverso dell'interpolazione:

Parametro	Funzione	Effetto
Fattore f	Passo di campionamento	Definisce l'intervallo di selezione. Un fattore 2 riduce larghezza e altezza della metà.
Risoluzione	Densità dei pixel	Riduce la quantità totale di informazione spaziale dell'immagine.
Aliasing	Effetto collaterale	Comparsa di pattern a scaletta o a blocchi dovuti alla perdita di dettagli fini.

2.12.2.2 📋 Compito (specifica per VPL)

Input:

La prima riga contiene `L`.

La seconda riga contiene `C`.

La terza riga contiene il fattore `f`.

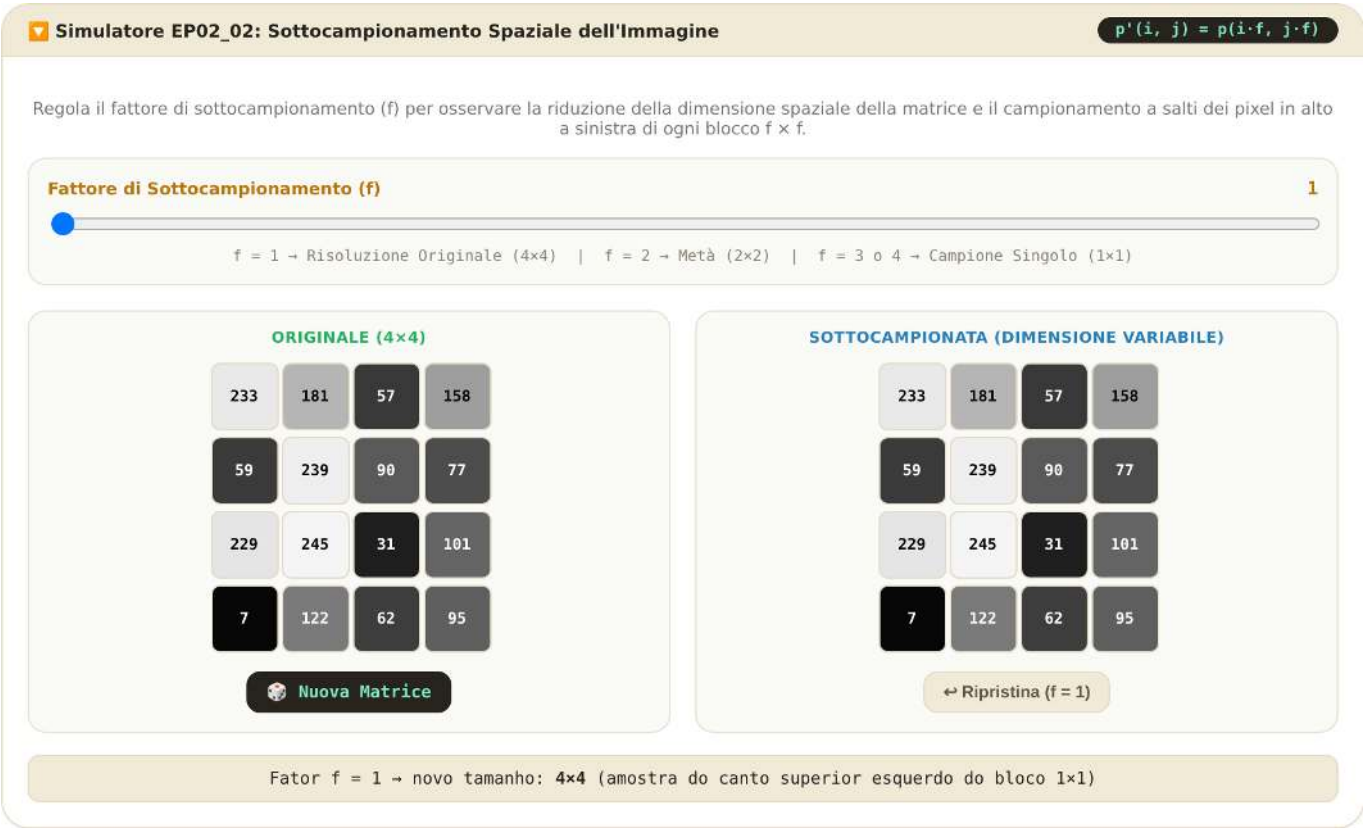
Le righe successive contengono gli elementi della matrice $L \times C$.

Output:

La matrice ridotta con le dimensioni corrispondenti allo slicing per `f`.

2.12.2.3 📌 Esempi

Input	Output	Osservazione
2 4 2 10 20 30 40 50 60 70 80	10 30	Il fattore 2 seleziona i pixel (0,0) e (0,2) della prima riga. La seconda riga viene ignorata.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.it/cap02/sim-ep0202-subamostragem.html

Figura 2.13: Simulatore EP02_02: Sottocampionamento Spaziale (Riduzione di Risoluzione per Salto 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 🎨 Quantizzazione dei Livelli di Grigio

In questa attività, devi implementare la quantizzazione uniforme di un'immagine, riducendo il numero di livelli di intensità di grigio originali a una nuova scala basata su un numero inferiore di bit.

- Leggi due interi **L** e **C**, che rappresentano le dimensioni della matrice.
- Leggi un intero k ($1 \leq k \leq 8$), che rappresenta il nuovo numero di bit dell'immagine.
- Calcola il numero di livelli ($N = 2^k$) e la dimensione dell'intervallo (passo).
- Per ogni pixel p, calcola il nuovo valore p' mappandolo all'indice del livello discretizzato corrispondente (variabile da 0 a $2^k - 1$).

- Stampa la matrice risultante con gli stessi valori delle dimensioni originali.
- Vedi in Figura 2.14 una simulazione di questo EP.

📌 **Importante**:

- **Posterizzazione:** Riducendo drasticamente i livelli (es: $k = 2$), noterai che le sfumature morbide si trasformano in bande cromatiche nette a causa della perdita di risoluzione dell'ampiezza.
- **Calcolo del Passo:** L'intervallo tra ogni livello è definito da $passo = 256/2^k$.
- **Mappatura:** Il metodo di quantizzazione uniforme per troncamento che mappa il pixel all'indice del rispettivo livello discretizzato è dato da:

$$p' = \left\lfloor \frac{p}{passo} \right\rfloor$$

In termini di implementazione (come in Python), ciò equivale alla divisione intera: `p' = p // passo`.

2.12.3.1 🧠 Discretizzazione dell'Ampiezza

Mentre il sottocampionamento gestisce la risoluzione spaziale, la quantizzazione si concentra sulla precisione del colore (ampiezza). Ridurre i bit significa semplificare l'informazione cromatica:

Parametro	Funzione	Effetto
Bit (k)	Profondità di colore	Definisce quanti toni diversi può avere l'immagine (2^k).
Passo	Intervallo di tono	Spaziatura tra i livelli di grigio consentiti.
Posterizzazione	Fenomeno visivo	Trasformazione di variazioni continue in blocchi di colore solido.

2.12.3.2 📋 Compito (specifica per VPL)

Input:

La prima riga contiene `L`.

La seconda riga contiene `C`.

La terza riga contiene il numero di bit `k`.

Le righe seguenti contengono gli elementi della matrice $L \times C$.

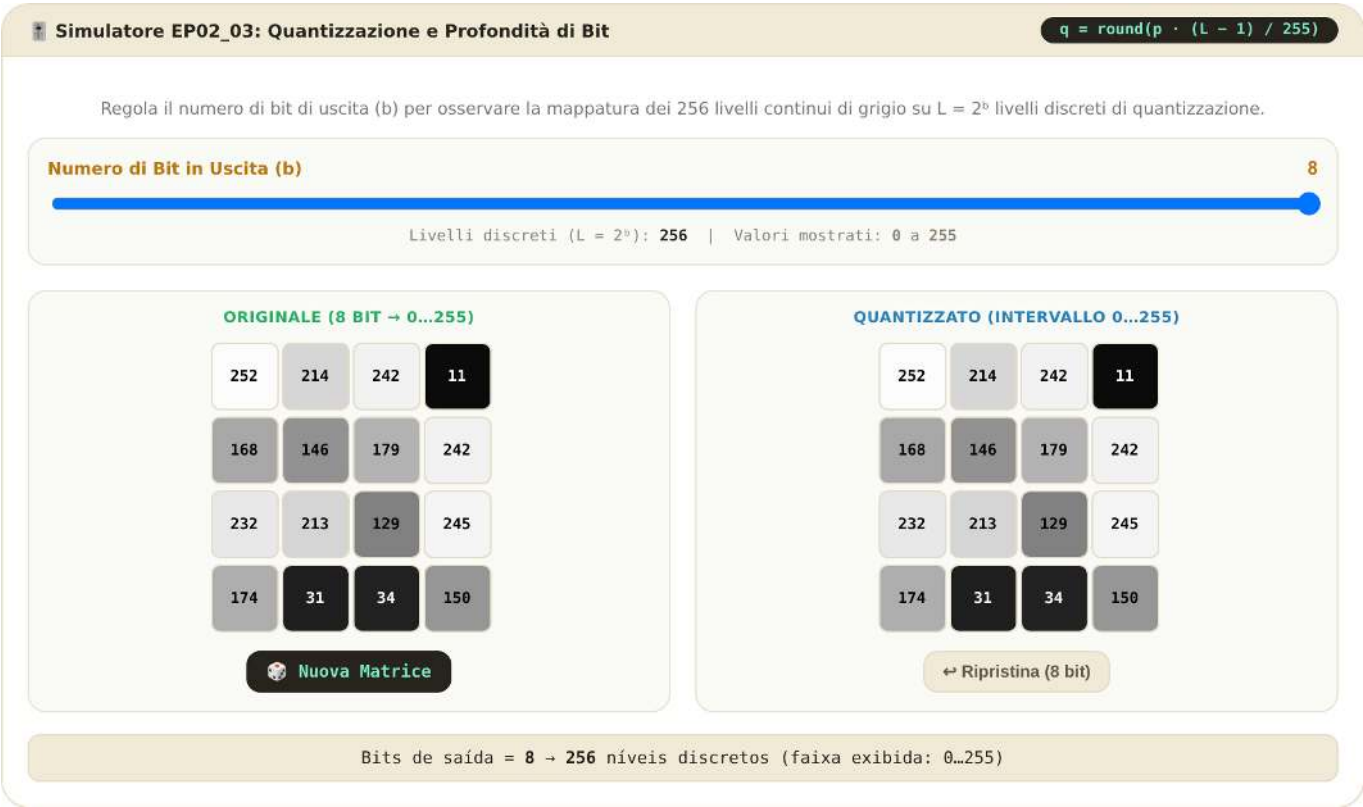
Output:

La matrice trasformata con gli indici dei livelli quantizzati, mantenendo la dimensione originale $L \times C$.

2.12.3.3 📌 Esempi

Input	Output	Osservazione
1 4 2 0 80 170 255	0 1 2 3	Con $k = 2$, abbiamo $2^2 = 4$ livelli discreti disponibili $(0, 1, 2, 3)$. Il passo è $256/4 = 64$. Applicando la divisione intera per elemento: $0//64 = 0$, $80//64 = 1$, $170//64 = 2$, $255//64 = 3$.

Input	Output	Osservazione
1 5 1 10 50 120 200 250	0 0 0 1 1	Con $k = 1$, abbiamo $2^1 = 2$ livelli (0 e 1). Passo $= 256/2 = 128$. I pixel minori di 128 risultano in 0, mentre i pixel maggiori o uguali a 128 risultano in 1.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.it/cap02/sim-ep0203-quantizacao.html

Figura 2.14: Simulatore EP02_03: Quantizzazione e Profondità di Bit (Riduzione del Numero di Livelli di Grigio)

```
%%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 📐 Trasformata di Distanza in Immagine Binaria

Data un'immagine binaria in cui i pixel di valore **1** rappresentano l'*oggetto* e i pixel **0** rappresentano lo *sfondo*, la distanza di un pixel di sfondo riceve la **minore distanza al pixel dell'oggetto più vicino**. I pixel dell'oggetto ricevono distanza **0**. Per semplicità, si consideri che l'immagine ha **un solo oggetto con un singolo pixel di valore 1**.

Problema: Leggere un'immagine binaria $L \times C$ e una metrica, e calcolare questa distanza semplificata applicando una delle tre formule:

$$d_{\text{Euclidea}} = \sqrt{(\Delta r)^2 + (\Delta c)^2}$$

$$d_{\text{City-block}} = |\Delta r| + |\Delta c|$$

$$d_{\text{Scacchiera}} = \max(|\Delta r|,\ |\Delta c|)$$

dove Δr è la differenza di righe e Δc la differenza di colonne tra due pixel.

2.12.4.1 🖼️ Perché è importante? - Applicazioni della DT

La Trasformata di Distanza (DT) compare in decine di pipeline di visione artificiale:

Metrica	Complessità	Applicazione tipica
Euclidea	🔴 $O(n^2)$ ingenuo	Scheletrizzazione, matching di forme
City-block	🟡 $O(n)$ con 2 passaggi	Morfologia, dilatazione/erosione
Scacchiera	🟢 $O(n)$ con 2 passaggi	Morfologia, dilatazione/erosione

2.12.4.2 📌 Requisiti Tecnici

- **Input:** * Prima riga: L e C (interi).
  - Seconda riga: nome della metrica (`euclidean`, `cityblock` o `chessboard`).
  - Successivamente, la matrice binaria $L \times C$ (valori 0 o 1).
- **Pixel dell'oggetto (1):** distanza = 0 (o 0.00 per euclidea).
- **Pixel di sfondo (0):** distanza all'unico pixel dell'oggetto nell'immagine.
- **Arrotondamento (euclidea):** stampare con **2 cifre decimali** (formato `:.2f`). City-block e Scacchiera producono interi — stampare senza decimali.
- **Output:** valori separati da spazio, una riga per ogni riga della matrice.
- Vedere in Figura 2.15 una simulazione di questo EP.

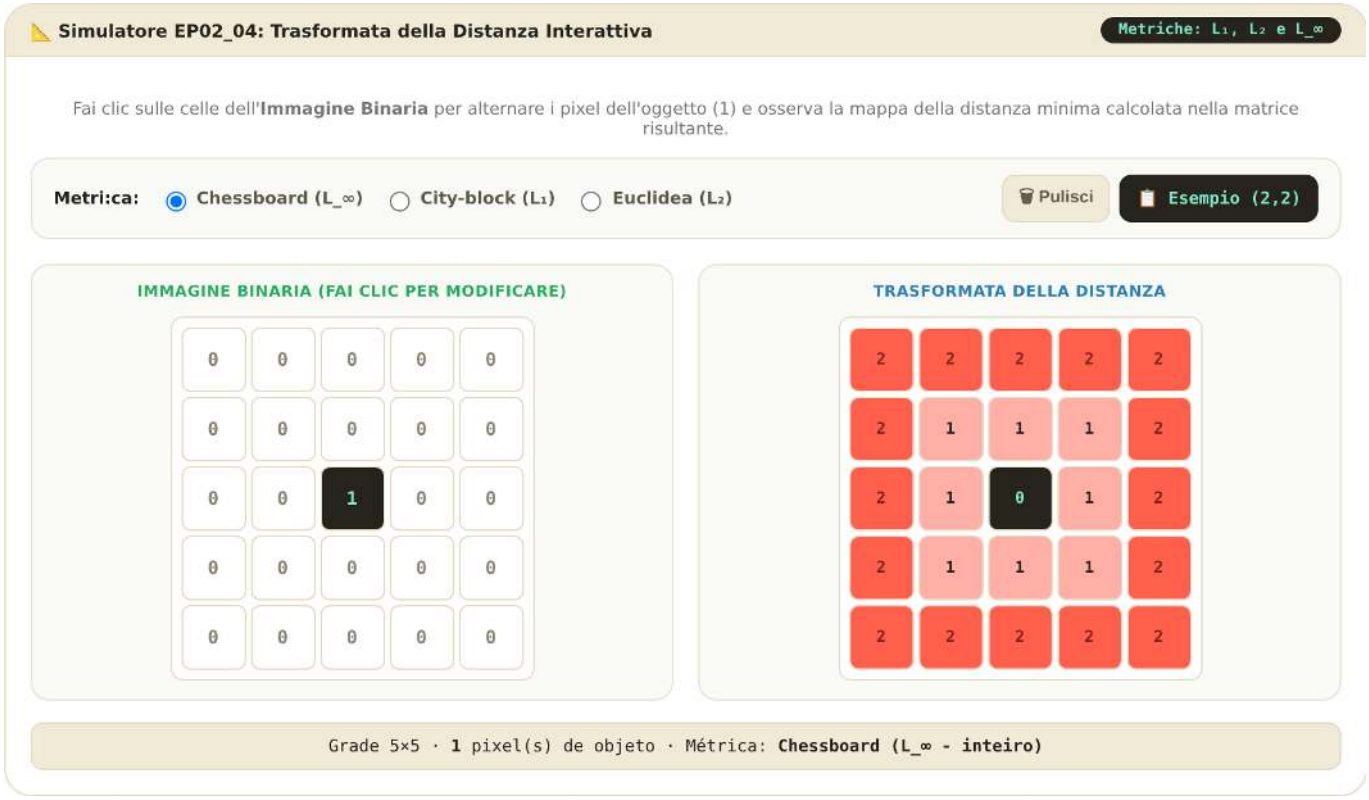
2.12.4.3 📌 Esempi

Input	Output	Osservazione
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 distanza Scacchiera è $\max(\|dx\|, \|dy\|)$. L'unico pixel oggetto è $(2, 2) = 0$; gli altri memorizzano la loro distanza minima fino ad esso.

2.12.4.4 📌 Osservazioni finali

- Poiché l'immagine ha **un solo oggetto di un pixel**, la distanza di ogni pixel di sfondo è semplicemente la distanza di quel pixel all'unico punto dell'oggetto.
- L'implementazione può usare la forza bruta (percorrere tutti i pixel dell'immagine e calcolare la distanza direttamente), poiché L e C sono piccoli nei casi di test.

- Questo problema è un riscaldamento per la Trasformata di Distanza generale, che verrà affrontata nei capitoli successivi con più oggetti e algoritmi ottimizzati.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.it/cap02/sim-ep0204-distancia.html

Figura 2.15: Simulatore EP02_04: Trasformata della Distanza in Immagine Binaria (Scacchiera, City-block ed Euclidea)

```
%%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 Traslazione dell'Immagine

In questa attività, devi implementare lo spostamento spaziale di un'immagine. La traslazione sposta ogni pixel dell'immagine originale in una nuova posizione basata su un vettore di spostamento.

- Leggi due interi **L** e **C**, che rappresentano le dimensioni della matrice.
- Leggi due interi t_x (spostamento orizzontale) e t_y (spostamento verticale).
- Leggi i valori interi della matrice originale.
- Calcola la nuova posizione (x', y') per ogni pixel originale (x, y).
- Stampa la matrice risultante con le stesse dimensioni dell'originale.
- Vedi una simulazione di questo EP in Figura 2.16.

📌 **Importante**:

- **Riempimento:** I pixel che "entrano" nell'immagine a causa dello spostamento e non hanno un corrispondente nell'originale devono essere riempiti con **0** (nero).

- **Scarto:** I pixel che, dopo la traslazione, escono dai limiti della matrice ($0 \dots L-1$ o $0 \dots C-1$) devono essere ignorati.
- **Coordinate:** Considera x come indice di riga e y come indice di colonna.

2.12.5.1 🧠 Spostamento Spaziale

Traslare un'immagine significa spostare tutti i suoi punti di una distanza fissa in direzioni specificate. Matematicamente, usando coordinate omogenee, l'operazione è descritta come:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Che risulta nelle semplici equazioni:

- $x' = x + t_x$
- $y' = y + t_y$

2.12.5.2 📋 Compito (specifica per VPL)

Input:

La prima riga contiene `L`.

La seconda riga contiene `C`.

La terza riga contiene gli interi `tx` e `ty`.

Le righe successive contengono gli elementi della matrice $L \times C$.

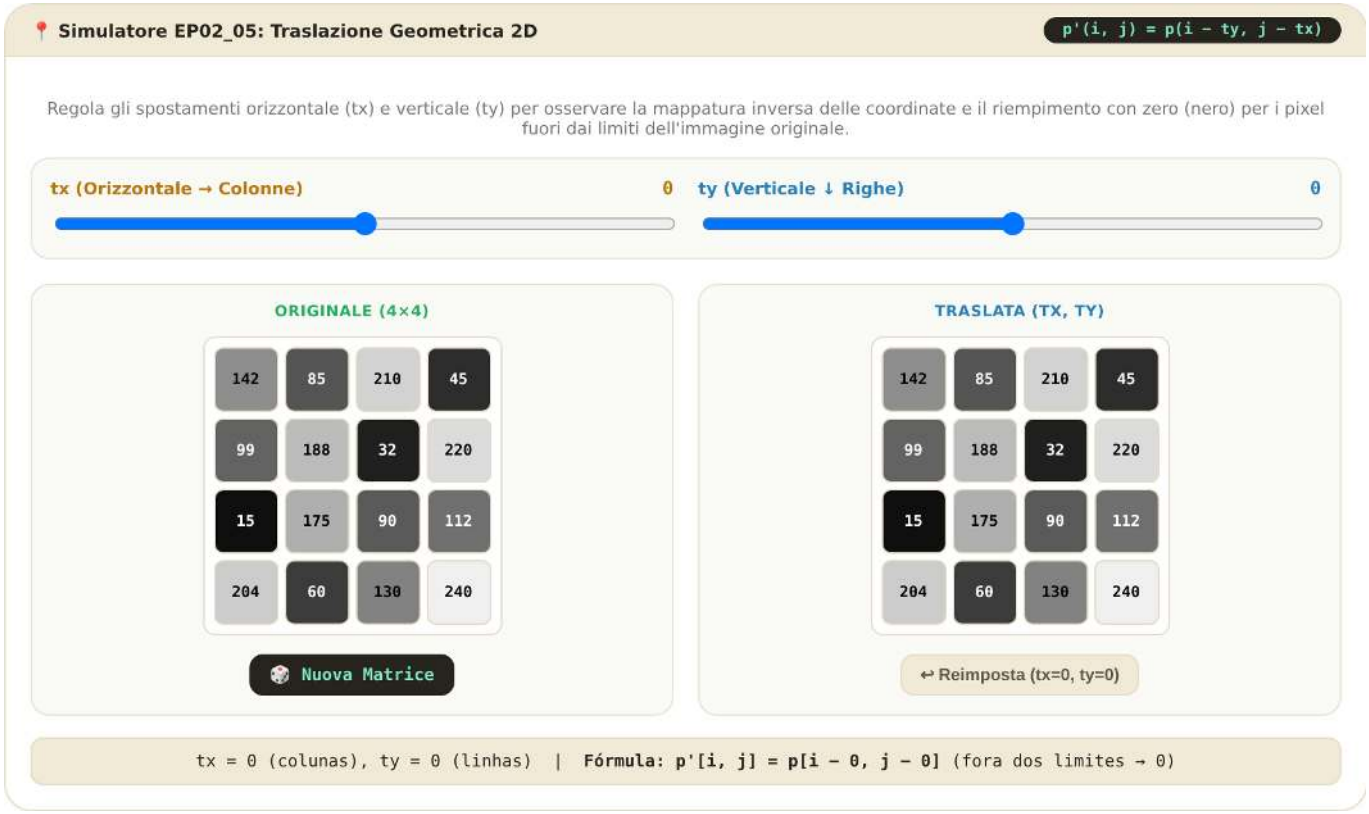
Output:

La matrice risultante con le stesse dimensioni $L \times C$ dopo lo spostamento.

2.12.5.3 📌 Esempi

Input	Output	Osservazione
2 2 1 1 10 20 30 40	0 0 0 10	**Spostamento ($t_x = 1, t_y = 1$):** Ogni pixel si sposta di una posizione a destra (orizzontale) e una in basso (verticale). Il pixel $(0, 0) = 10$ va alla destinazione $(1, 1)$ (angolo inferiore destro). Le posizioni vuote sono riempite con 0.
3 3 -1 0 1 2 3 4 5 6 7 8 9	2 3 0 5 6 0 8 9 0	**Spostamento ($t_x = -1, t_y = 0$):** Ogni pixel si sposta di una posizione a sinistra (orizzontale). La prima colonna originale (`1`, `4`, `7`) viene scartata, le altre colonne si spostano a sinistra, e l'ultima colonna risultante è riempita con zeri (0).

```
%%writefile EP02_05.cpp
// your solution
```



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.it/cap02/sim-ep0205-translacao.html

Figura 2.16: Simulatore EP02_05: Traslazione Geometrica dell'Immagine (Spostamento tx e ty con Riempimento del Bordo)

```
Overwriting EP02_05.cpp
```

```
TestSuite("EP02_05.cpp").run()
```

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

2.12.6 EP02_06 🔄 Rotazione dell'Immagine

In questa attività, devi implementare la rotazione di un'immagine attorno al suo centro geometrico. Questa operazione richiede il mapping delle coordinate e l'uso di tecniche di interpolazione per determinare i nuovi valori dei pixel.

- Leggi due interi **L** e **C**, che rappresentano le dimensioni della matrice.
- Leggi un valore reale θ (angolo in gradi) e una *stringa* che rappresenta il **metodo** di interpolazione (`nearest` o `bilinear`).
- Leggi i valori interi della matrice originale.
- Esegui la rotazione attorno al centro dell'immagine $(L/2, C/2)$.
- Stampa la matrice risultante con le stesse dimensioni dell'originale.
- Vedi in Figura 2.17 una simulazione di questo EP.

📌 **Importante**:

- **Mapping Inverso:** Per evitare "buchi" nell'immagine finale, percorri ogni pixel (x', y') dell'immagine di destinazione e calcola la sua posizione corrispondente (x, y) nell'immagine originale usando la matrice di rotazione inversa.
- **Interpolazione:**
- `nearest`: Assegna il valore del pixel più vicino alla coordinata calcolata.

- `bilinear`: Calcola una media ponderata basata sui 4 vicini più prossimi.
- **Bordi:** I pixel la cui origine (x, y) cade fuori dai limiti dell'immagine originale devono essere riempiti con **0**.

2.12.6.1 🧠 Trasformazione per Angolo

La rotazione di un punto (x, y) rispetto all'origine di un angolo θ è data dalla matrice di trasformazione. Per ruotare attorno a un centro (x_c, y_c), prima trasliamo il centro nell'origine, ruotiamo e trasliamo di nuovo:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & x_c \\ \sin\theta & \cos\theta & y_c \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x - x_c \\ y - y_c \\ 1 \end{bmatrix}$$

Suggerimento: Usa il mapping inverso per garantire che tutti i pixel dell'immagine di uscita siano riempiti correttamente.

2.12.6.2 📋 Compito (specifica per VPL)

Input:

La prima riga contiene `L`.

La seconda riga contiene `C`.

La terza riga contiene l'angolo `theta` (in gradi) e il metodo `interp` (`nearest` o `bilinear`).

Le righe successive contengono gli elementi della matrice $L \times C$.

Output:

La matrice ruotata con `L` righe e `C` colonne.

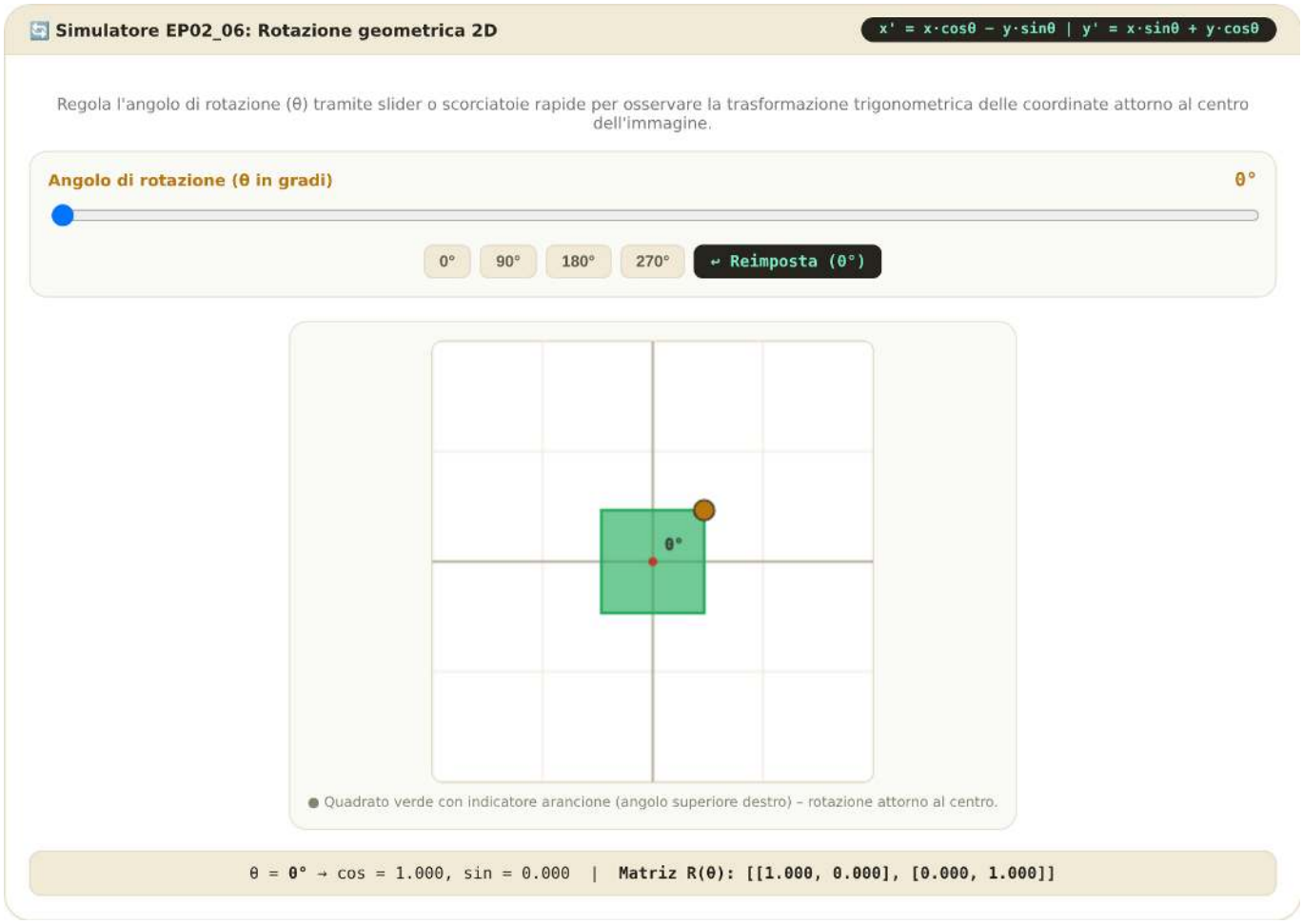
2.12.6.3 📌 Esempi

Input	Output	Osservazione
2 2 90 nearest 1 2 3 4	3 1 4 2	Rotazione di 90° in senso orario: la colonna 0 diventa la riga 0 (dal basso verso l'alto). $(0,0) = 1 \to (1,0)$, $(1,0) = 3 \to (0,0)$, $(0,1) = 2 \to (1,1)$, $(1,1) = 4 \to (0,1)$.
3 3 45 bilinear 0 0 0 0 255 0 0 0 0	0 180 0 180 255 180 0 180 0	Rotazione di 45°: il pixel centrale rimane 255; i vicini diretti ricevono un valore interpolato ≈ 180 tramite bilineare; gli angoli rimangono 0.

```
%%writefile EP02_06.cpp
// your solution
```

```
Overwriting EP02_06.cpp
```

```
TestSuite("EP02_06.cpp").run()
```



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.it/cap02/sim-ep0206-rotacao.html

Figura 2.17: Simulatore EP02_06: Rotazione dell'immagine attorno all'origine di un angolo

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

2.12.7 EP02_07 🔍 Ridimensionamento (Scala)

In questa attività, devi implementare il ridimensionamento di un'immagine utilizzando fattori di scala. Diversamente dal semplice sottocampionamento, qui useremo tecniche di interpolazione per consentire sia l'ingrandimento che la riduzione dell'immagine.

- Leggi due interi **L** e **C**, che rappresentano le dimensioni della matrice originale.
- Leggi due valori reali s_x (scala sulle righe) e s_y (scala sulle colonne).
- Leggi una *stringa* che rappresenta il **metodo** di interpolazione (`nearest` o `bilinear`).
- Leggi i valori interi della matrice originale.
- Calcola le nuove dimensioni: $L' = \text{round}(L \times s_x)$ e $C' = \text{round}(C \times s_y)$.
- Stampa la matrice risultante con le nuove dimensioni.
- Vedi in Figura 2.18 una simulazione di questo EP.

📌 **Importante**:

- **Mappatura inversa:** Per ogni pixel (x', y') dell'immagine di destinazione, trova la posizione corrispondente nell'origine usando $(x, y) = (x'/s_x, y'/s_y)$.
- **Interpolazione:**
- `nearest`: Seleziona il valore del pixel più vicino (arrotondamento delle coordinate).
- `bilinear`: Esegue un'interpolazione lineare doppia tra i quattro pixel vicini più prossimi nell'immagine originale.
- **Bordi:** Assicurati che la mappatura non tenti di accedere a indici al di fuori dell'intervallo $[0, L-1]$ e $[0, C-1]$.

2.12.7.1 🧠 Interpolazione per Ingrandimento/Riduzione

Ridimensionare un'immagine mediante fattori (s_x, s_y) richiede il riempimento dei vuoti (nell'ingrandimento) o la fusione delle informazioni (nella riduzione). Il metodo di interpolazione definisce la qualità visiva del risultato:

Metodo	Funzionamento	Effetto Visivo
Nearest	Prende il valore del vicino più prossimo.	Veloce, ma genera un effetto "pixelato" o a blocchi.
Bilineare	Media ponderata dei 4 vicini (2×2).	Ammorbidisce l'immagine, riducendo l'effetto scalettato (aliasing).

2.12.7.2 📋 Compito (specifica per VPL)

Input:

La prima riga contiene `L`.

La seconda riga contiene `C`.

La terza riga contiene i fattori `sx` e `sy`.

La quarta riga contiene il metodo `interp` (`nearest` o `bilinear`).

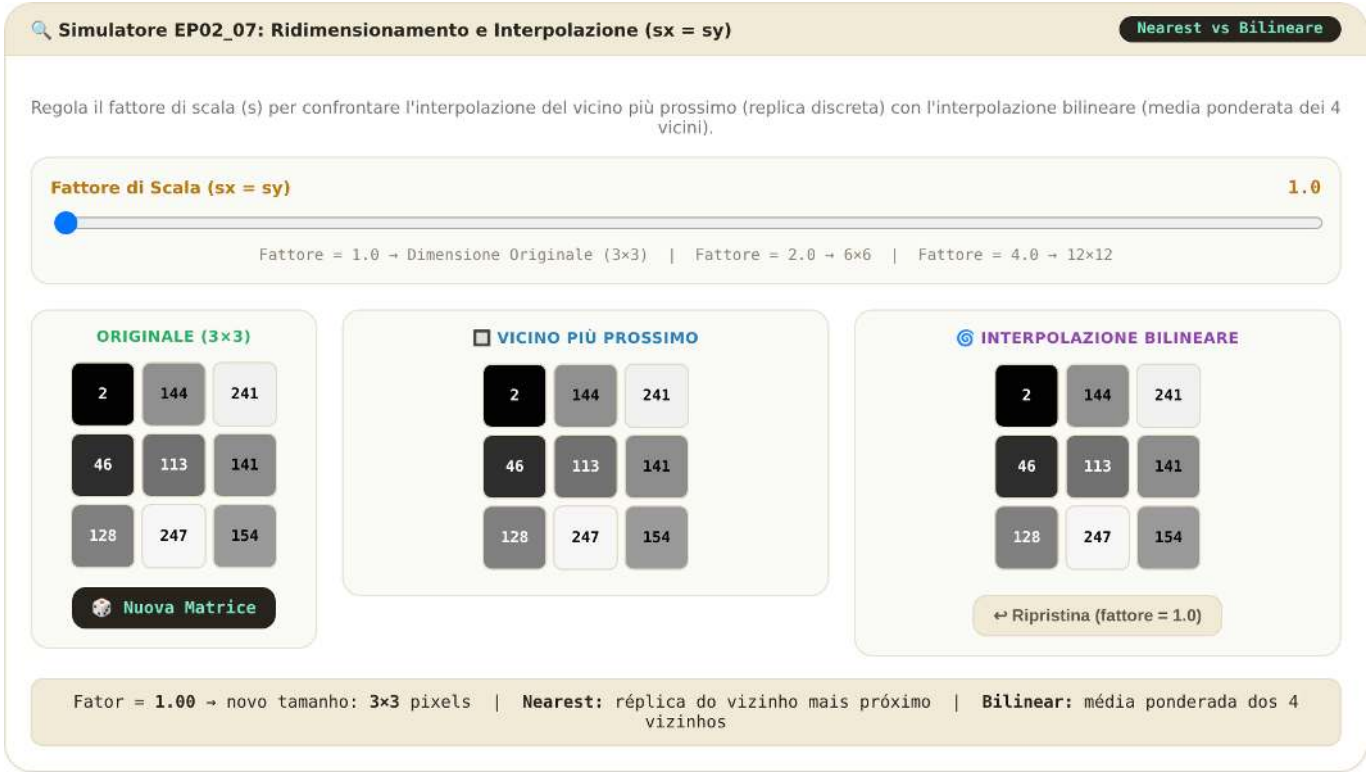
Le righe successive contengono gli elementi della matrice $L \times C$.

Output:

La matrice ridimensionata con dimensioni $L' \times C'$.

2.12.7.3 📌 Esempi

Input	Output	Osservazione
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	Ingrandimento 2×: ogni pixel originale viene replicato in un blocco 2×2. L'immagine 2×2 diventa 4×4.
2 2 0.5 0.5 nearest 10 20 30 40	10	Riduzione 0.5×: l'immagine 2×2 diventa 1×1. Con nearest, l'unico pixel di output campiona la posizione $(0, 0) = 10$.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.it/cap02/sim-ep0207-escala.html

Figura 2.18: Simulatore EP02_07: Ridimensionamento Spaziale e Interpolazione (Nearest Neighbor vs Bilineare)

```
# Not yet ported to this language in this version - conceptual reference in Python.
%%writefile EP02_07.py
# Codice Python
import numpy as np
from morph import mm

# 1. Lettura delle dimensioni, dei fattori e del metodo
l = int(input())
c = int(input())
sx, sy = map(float, input().split())
interp = input().strip()
```

```
# 2. Lettura dell'immagine originale
img = mm.readImg(l, c)

# 3. Nuove dimensioni
l_new = round(l * sx)
c_new = round(c * sy)

# 4. Ridimensionamento usando mm.resize
# cv2.resize usa (larghezza, altezza) = (colonne, righe)
resultado = mm.resize(img, (c_new, l_new), method=interp)

# 5. Visualizzazione
print(mm.drawImg(resultado))
```

```
TestSuite("EP02_07.cpp").run()
```

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

2.12.8 EP02_08 🔀 Taglio (Shear)

In questa attività, devi implementare la trasformazione di taglio su un'immagine. Il taglio è una trasformazione affine che sposta ogni punto in una direzione fissa, di un valore proporzionale alla sua distanza da una retta parallela a tale direzione, producendo un effetto di inclinazione.

- Leggi due interi **L** e **C**, che rappresentano le dimensioni della matrice.
- Leggi due valori reali sh_x (taglio orizzontale) e sh_y (taglio verticale).
- Leggi una *stringa* che rappresenta il **metodo** di interpolazione (`nearest` o `bilinear`).
- Leggi i valori interi della matrice originale.
- Applica la trasformazione mantenendo le dimensioni originali dell'immagine (tagliando ciò che supera i limiti).
- Stampa la matrice risultante con le dimensioni $L \times C$.
- Vedi in Figura 2.19 una simulazione di questo EP.

📌 **Importante**:

- **Mapping inverso:** Per ogni pixel (x', y') dell'immagine di destinazione, calcola la posizione corrispondente nell'origine (x, y) utilizzando la matrice di taglio inversa.
- **Riempimento:** Le coordinate che risultano in posizioni fuori dalla matrice originale devono essere riempite con **0**.
- **Coordinate:** Ai fini di questa implementazione, considera x come indice di riga e y come indice di colonna.

2.12.8.1 🧠 Distorsione affine

Il taglio altera la geometria dell'immagine inclinandone gli assi. La relazione tra le coordinate originali (x, y) e quelle trasformate (x', y') è data da:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & sh_x & 0 \\ sh_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Questo produce le equazioni:

- $x' = x + sh_x \cdot y$
- $y' = y + sh_y \cdot x$

2.12.8.2 📋 Compito (specifica per VPL)

Input:

La prima riga contiene `L`.

La seconda riga contiene `C`.

La terza riga contiene i fattori `shx` e `shy`.

La quarta riga contiene il metodo `interp` (`nearest` o `bilinear`).

Le righe successive contengono gli elementi della matrice $L \times C$.

Output:

La matrice trasformata con le stesse dimensioni $L \times C$.

2.12.8.3 📌 Esempi

Input	Output	Osservazione
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	Taglio orizzontale: la riga i si sposta di $\lfloor i \cdot 0.5 \rfloor$ pixel. Riga $0 \to 0$px, riga $1 \to 0$px, riga $2 \to 1$px. I pixel spostati fuori dai limiti vengono scartati e le posizioni vuote vengono riempite con 0.
2 2 0.0 1.0 nearest 10 20 30 40	10 0 30 20	Taglio verticale: la colonna j si sposta di $\lfloor j \cdot 1.0 \rfloor$ pixel verso il basso. Colonna $0 \to 0$px (invariata), colonna $1 \to 1$px: 20 scende a $(1, 1)$ e $(0, 1)$ diventa 0.

```
%%writefile EP02_08.cpp
// your solution
```

```
Overwriting EP02_08.cpp
```

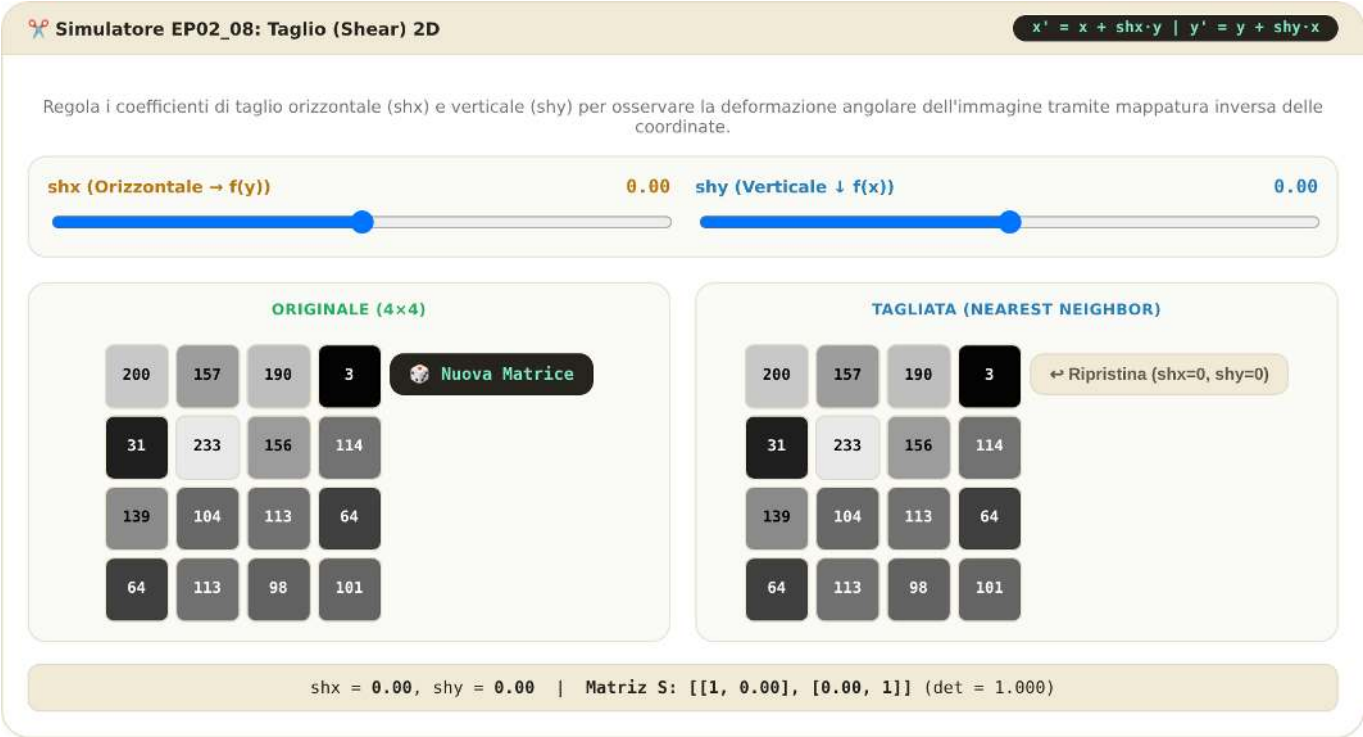
```
TestSuite("EP02_08.cpp").run()
```

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

2.12.9 EP02_09 🧩 Trasformazione Affine Generica

In questa attività, devi implementare una trasformazione affine arbitraria su un'immagine. Questa operazione è la generalizzazione di tutte le trasformazioni lineari (scala, rotazione, taglio) combinate con la traslazione, consentendo manipolazioni geometriche complesse attraverso un'unica matrice.

- Leggi due interi **L** e **C**, che rappresentano le dimensioni della matrice.
- Leggi **sei valori reali** (a, b, t_x, c, d, t_y) che compongono la matrice di trasformazione affine 2×3.
- Leggi una *stringa* che rappresenta il **metodo** di interpolazione (`nearest` o `bilinear`).
- Leggi i valori interi della matrice originale.
- Applica la trasformazione mantenendo la dimensione originale $L \times C$.
- Stampa la matrice risultante.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.it/cap02/sim-ep0208-cisalhamento.html

Figura 2.19: Simulatore EP02_08: Trasformazione Geometrica di Taglio 2D (Shear Orizzontale e Verticale)

- Vedi in Figura 2.20 una simulazione di questo EP.

📌 **Importante**:

- **Mappatura inversa:** Per calcolare il valore di ogni pixel nell'immagine di destinazione, devi utilizzare l'**inversa** della matrice di trasformazione affine fornita per trovare la coordinata corrispondente nell'immagine originale.
- **Riempimento:** Le coordinate calcolate che cadono al di fuori dei limiti $[0, L-1]$ e $[0, C-1]$ dell'immagine originale devono risultare in un pixel di valore **0**.
- **Flessibilità:** Questa implementazione deve essere in grado di eseguire uno qualsiasi dei compiti precedenti (traslazione, rotazione, ecc.) semplicemente modificando i parametri della matrice.

Suggerimento:

```
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 🧠 Combinazione di Operazioni

La trasformazione affine preserva punti, rette e piani. Nell'elaborazione delle immagini, mappa la posizione (x, y) in (x', y') seguendo il sistema:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

Oppure, in forma compatta in coordinate omogenee:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & t_x \\ c & d & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

2.12.9.2 📋 Compito (specifica per VPL)

Input:

La prima riga contiene `L`.

La seconda riga contiene `C`.

La terza riga contiene sei float: `a b tx c d ty`.

La quarta riga contiene il metodo `interp` (`nearest` o `bilinear`).

Le righe successive contengono gli elementi della matrice $L \times C$.

Output:

La matrice trasformata con le dimensioni originali $L \times C$.

2.12.9.3 📌 Esempi

Input	Output	Osservazione
2 2 1.0 0.0 0.5 0.0 1.0 0.5 bilinear 10 20 30 40	15 20 25 30	Traslazione frazionaria ($t_x = 0.5, t_y = 0.5$): ogni pixel di output (i, j) campiona la posizione $(i + 0.5,\ j + 0.5)$ dell'input tramite bilineare. Es: $(0, 0)$ interpola i quattro vicini $\rightarrow$ 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	Scala 2× tramite matrice affine ($a = 2, d = 2$): ogni pixel di output (i, j) campiona la posizione $(2i, 2j)$ dell'input con nearest. Es: $(0, 2) \rightarrow (0, 4)$ fuori dall'immagine $\rightarrow$ nearest blocca a $(0, 2) = 3$… in attesa di conferma della logica di bordo.

```
%%writefile EP02_09.cpp
// your solution
```

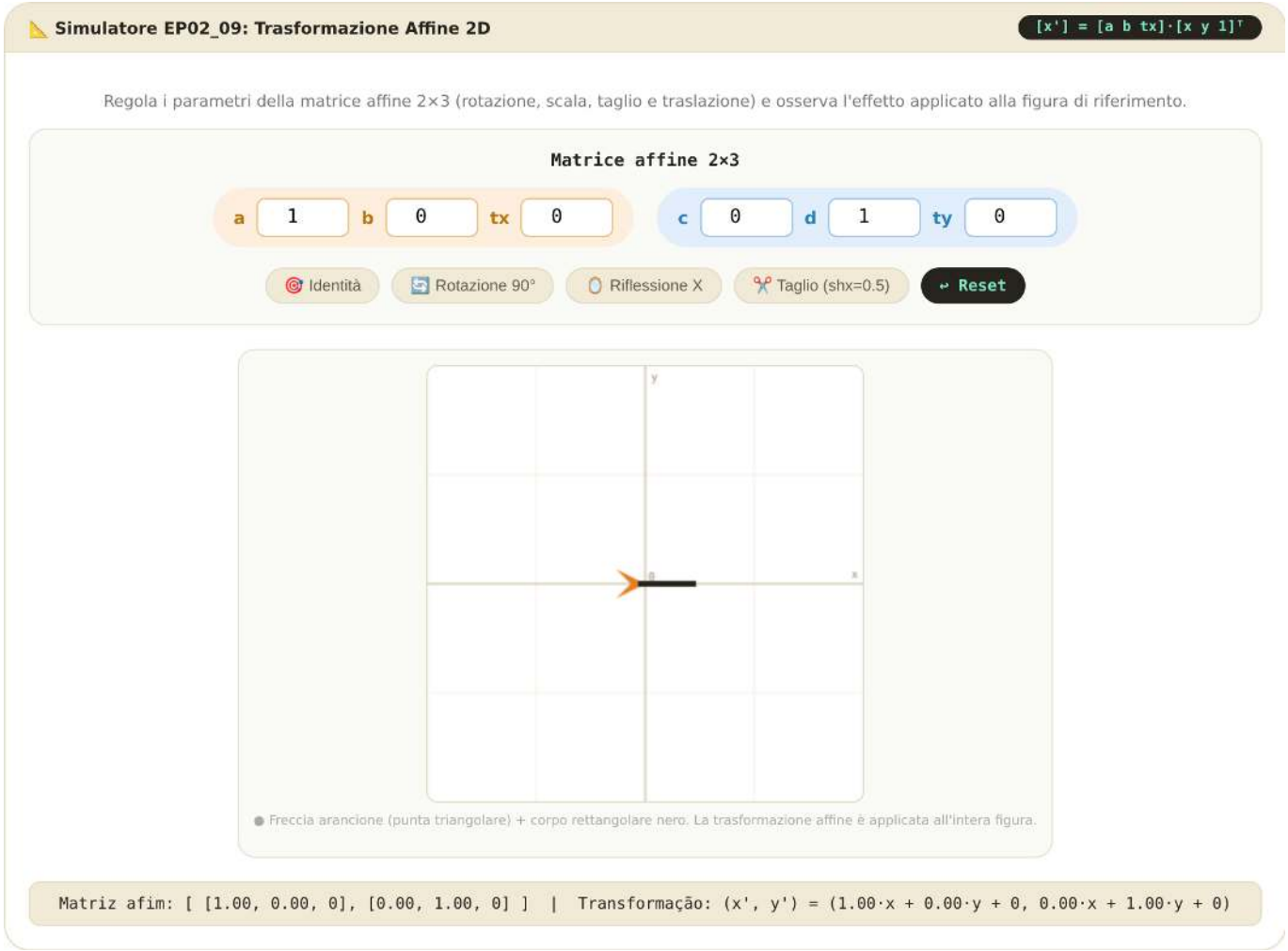
```
Overwriting EP02_09.cpp
```

```
TestSuite("EP02_09.cpp").run()
```

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

2.12.10 EP02_10 🎯 Correzione della Prospettiva (Omografia)

In questa attività, devi implementare la trasformazione di prospettiva, nota anche come omografia. A differenza delle trasformazioni affini, la prospettiva non preserva il parallelismo, consentendo di "rettificare" oggetti inclinati, come documenti o targhe catturati da angolazioni oblique.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.it/cap02/sim-ep0209-afim.html

Figura 2.20: Simulatore EP02_09: Trasformazione Affine 2D (Matrice 2×3)

- Leggi due interi **L** e **C**, che rappresentano le dimensioni della matrice originale.
- Leggi **quattro coppie di coordinate** (x, y) che rappresentano i vertici del quadrilatero di origine (oggetto distorto).
- Leggi **quattro coppie di coordinate** (x, y) che rappresentano i vertici del quadrilatero di destinazione (dove l'oggetto deve essere mappato).
- Leggi i valori della matrice originale.
- Calcola la matrice di omografia 3×3 e applica la trasformazione.
- Stampa la matrice risultante con le dimensioni di uscita specificate.
- Vedi in Figura 2.21 una simulazione di questo EP.

📌 **Importante**:

- **Gradi di libertà:** L'omografia possiede 8 gradi di libertà (il nono elemento della matrice 3×3 è una costante di normalizzazione, generalmente 1), richiedendo almeno 4 punti corrispondenti per essere calcolata.
- **Proiezione:** Dopo aver moltiplicato le coordinate per la matrice, è necessario dividere i risultati x' e y' per la componente omogenea w per tornare al piano 2D.
- **Uso di librerie:** Per questo compito, puoi utilizzare le funzioni `cv2.getPerspectiveTransform` per ottenere la matrice e `cv2.warpPerspective` per applicare la trasformazione, oppure implementare manualmente il sistema lineare e la mappatura inversa per una sfida extra.

```
# Dimensioni di uscita: bounding box dei punti di destinazione + 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))
# oppure
dst = mm.perspective_transform(img, pts1, pts2, size=(w, h))
```

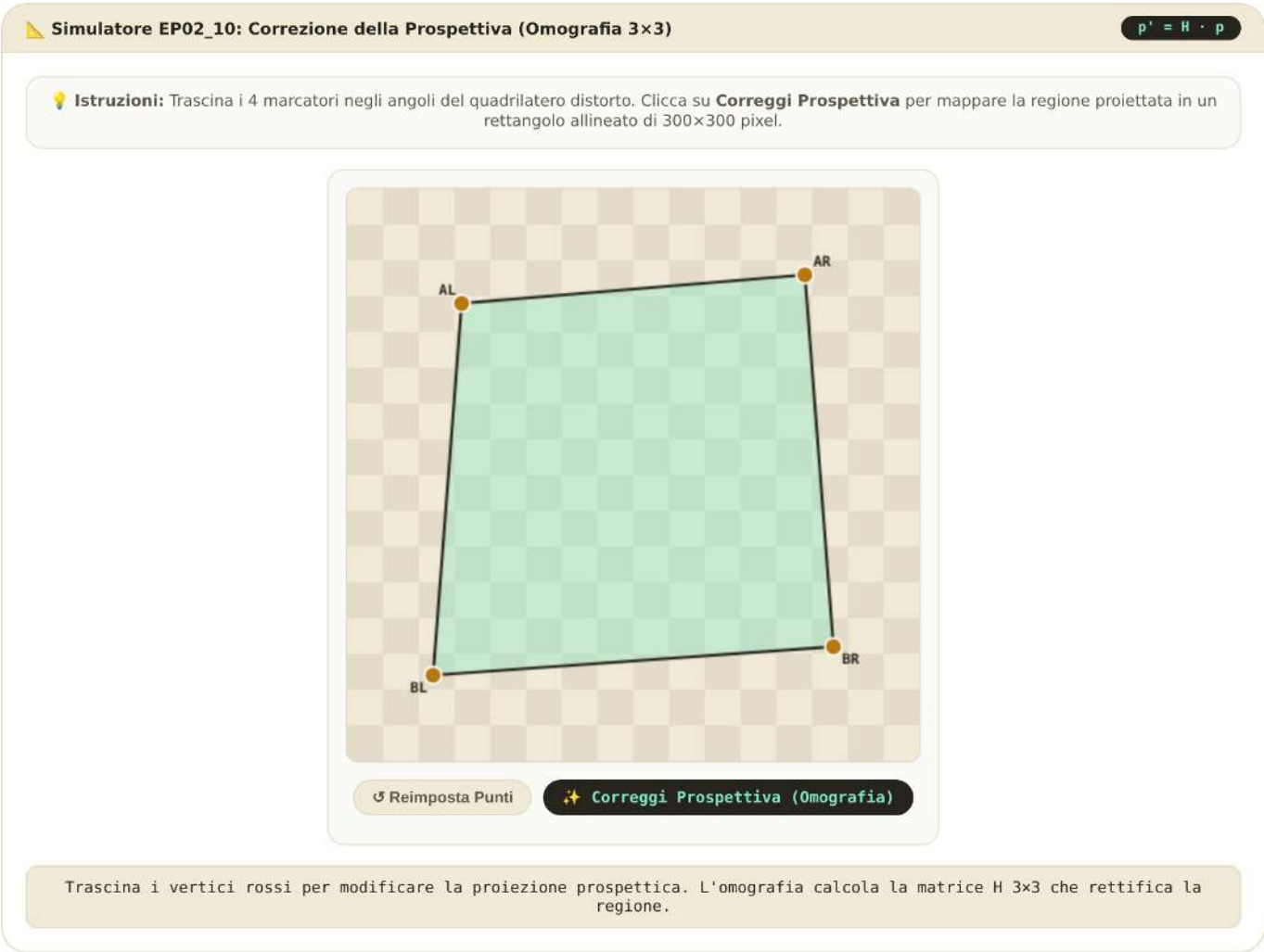
2.12.10.1 🧠 Deformazione non affine

Mentre le trasformazioni affini mappano parallelogrammi in parallelogrammi, l'omografia mappa qualsiasi quadrilatero in un altro quadrilatero. Questo è essenziale per la visione artificiale:

Operazione	Caratteristica	Applicazione Tipica
Omografia	Proiezione su piano	Correzione di documenti, scansione di targhe.
Punto di fuga	Convergenza di linee	Ricostruzione 3D da immagini 2D.
Warping	Deformazione di mesh	Stabilizzazione video e panoramiche (stitching).

2.12.10.2 📌 Esempi

Input	Output	Osservazione
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	Le prime 4 righe dopo le dimensioni sono i punti di origine; le 4 successive sono le destinazioni. Con punti identici, la trasformazione di prospettiva è l'identità e l'immagine viene preservata.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.it/cap02/sim-ep0210-perspectiva.html

Figura 2.21: Simulatore EP02_10: Correzione della Prospettiva (Trasformazione di Omografia 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 🏆 Correzione della Prospettiva (Omografia) su Immagine Reale

In questa attività, l'obiettivo è applicare la trasformazione prospettica (omografia) per "raddrizzare" un oggetto inclinato in una fotografia reale. Lavorerai con l'immagine di un giornale, dove la griglia di un gioco di Sudoku è distorta a causa dell'angolo con cui è stata scattata la foto.

Il tuo programma deve leggere i parametri di input dal terminale, caricare l'immagine, calcolare la matrice di omografia 3×3, applicare la trasformazione geometrica e visualizzare un indicatore globale di validazione.

- Leggi due interi **L** e **C**, che rappresentano le dimensioni di righe e colonne (altezza e larghezza) che l'immagine rettificata di uscita deve avere.
- Leggi **quattro coppie di coordinate** (x, y) dal terminale, che rappresentano i quattro angoli del quadrilatero di origine (il Sudoku distorto nell'immagine originale).
- Calcola automaticamente le **quattro coppie di coordinate di destinazione** utilizzando le dimensioni L e C fornite, mappando gli angoli agli estremi della nuova immagine: $(0, 0)$, $(C-1, 0)$, $(0, L-1)$ e $(C-1, L-1)$.
- Carica l'immagine locale `sudoku.png` e convertila in scala di grigi (grayscale).
- Calcola la matrice di omografia e applica la trasformazione spaziale all'immagine.
- **Output:** Calcola e stampa la **somma di tutti i pixel** dell'immagine risultante.

📌 **Importante**:

- **File di input:** L'immagine `sudoku.png` deve trovarsi nella stessa cartella dello script. Il programma deve leggerla direttamente dal disco (es: usando `mm::read("sudoku.png")` o `cv2.imread`).
- **Ordine dei Punti:** Assicurati che la lettura dei 4 punti di origine e la generazione dei 4 punti di destinazione seguano rigorosamente lo stesso ordine degli angoli: Superiore-Sinistro (TL), Superiore-Destro (TR), Inferiore-Sinistro (BL) e Inferiore-Destro (BR).
- **Dimensioni in OpenCV:** Ricorda che funzioni come `cv2.warpPerspective` si aspettano la dimensione dell'immagine di uscita nel formato `(larghezza, altezza)`, che equivale a `(C, L)`.
- **Interpolazione:** Per garantire la coerenza matematica della somma dei pixel con il correttore automatico, utilizza l'interpolazione bilineare standard (`flags=cv2.INTER_LINEAR`).
- **Crediti:** L'immagine utilizzata è **"Sudoku en periódico"** di Héctor Rodríguez, sotto licenza **CC BY 2.0**.

2.12.11.1 🧠 Contesto del Problema

L'omografia ha 8 gradi di libertà, richiedendo almeno 4 corrispondenze di punti per essere calcolata. A differenza delle trasformazioni affini, mappa qualsiasi quadrilatero in un altro quadrilatero, permettendo che le linee che convergono verso punti di fuga tornino ad essere parallele:

Operazione	Caratteristica	Applicazione Tipica
Omografia	Proiezione tra piani	Rettifica di documenti, scansione di targhe e codici QR.
Mappatura Inversa	Scansione dalla destinazione all'origine	Evita "buchi" o pixel vuoti nell'immagine finale rettificata.
Warping	Ricampionamento spaziale	Correzione della distorsione delle lenti e montaggio di panorami (*stitching*).

2.12.11.2 📌 Esempi

Input	Output	Osservazione
500 500 100 120 420 95 80 440 450 460	32982820	I primi due input sono le dimensioni di uscita (L e C). Le 4 righe successive sono le coordinate (x, y) degli angoli del Sudoku nell'immagine originale + PAD. L'output è la somma totale dei pixel dell'immagine rettificata.
200 200 100 120 420 95 80 440 450 460	5277150	Stessi punti di origine dell'esempio precedente, ma generando un'immagine di uscita più piccola (200×200). La somma dei pixel si riduce proporzionalmente a causa della scala.

2.12.11.3 Acquisizione dell'immagine del sudoku e conversione in livelli di grigio

La Figura 2.22 mostra la lettura dell'immagine originale seguita dalla conversione in tonalità di grigio e dal ridimensionamento a una matrice di 500×500 pixel, preparando i dati per la fase successiva.

La correzione di prospettiva, applicata nella Figura 2.23 tramite matrice di omografia, elimina le deformazioni causate dall'angolo della telecamera e produce una visione frontale e regolare della griglia del Sudoku.

```
%%writefile tmp/fig_02_sudoku_original.cpp
#define MM_OUT "tmp/fig_02_sudoku_original.png"
//| 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

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

int main() {
    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)")
```

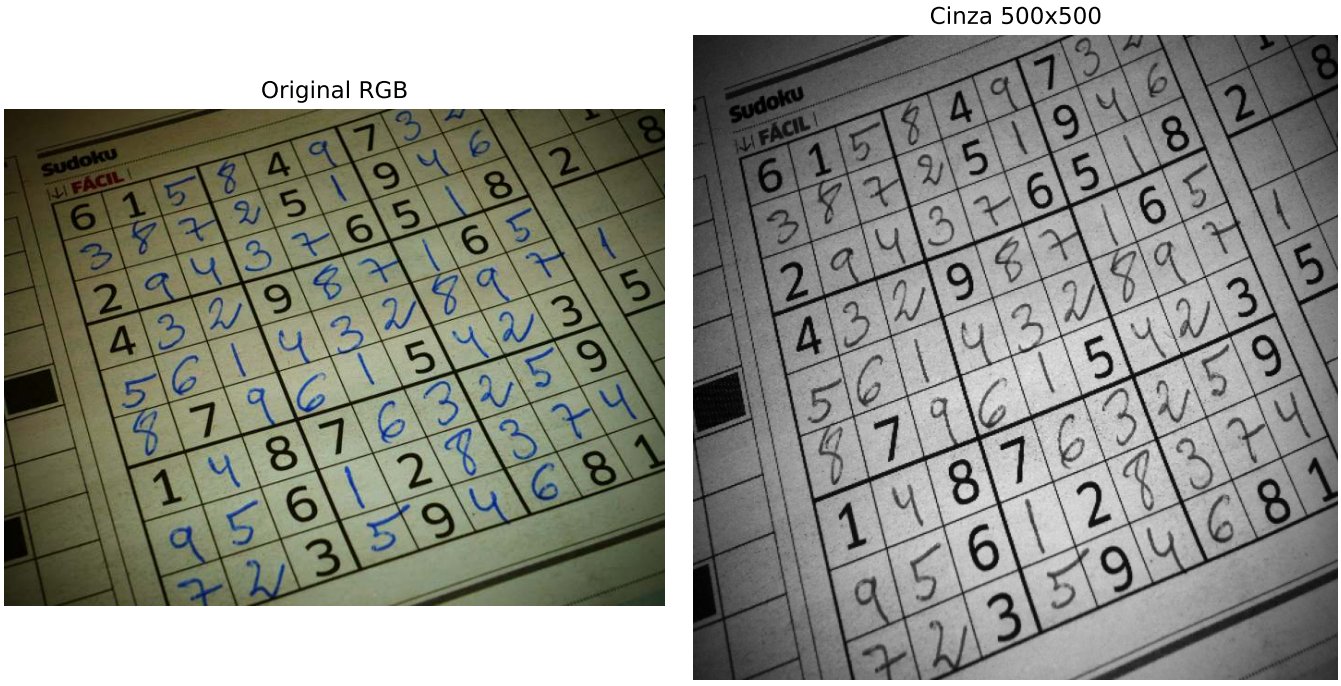


Figura 2.22: Aquisição da imagem de um Sudoku à esquerda. À direita, conversão para tons de cinza e redimensionamento. Crédito: Héctor Rodríguez de Guardamar, Espanha (CC BY 2.0).

```
%%writefile tmp/fig_02_sudoku2.cpp
#define MM_OUT "tmp/fig_02_sudoku2.png"
// 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
#include "morph.hpp"
```

```cpp
#include <vector>
#include <filesystem>

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)
    int PAD = 60;
    mm::Image img_pad = mm::pad(img, PAD);

    // 3. Cantos da grade na imagem expandida - TL, TR, BL, BR
    const double pts1[4][2] = {{100, 160}, {390, 45}, {200, 580}, {570, 420}};

    // 4. Cantos de destino: vista frontal SIZE×SIZE
    int SIZE = 500;
    const double pts2[4][2] = {{0, 0}, {SIZE, 0}, {0, SIZE}, {SIZE, SIZE}};

    // 5. Homografia pts1 -> pts2 e retificação
    mm::Image img_rect = mm::perspective_transform(img_pad, pts1, pts2, SIZE, SIZE);

    // 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"
```

```
tmp/fig_02_sudoku2.cpp: In function ‘int main()’:
tmp/fig_02_sudoku2.cpp:23:41: warning: narrowing conversion of ‘SIZE’ from ‘int’ to ‘double’
[-Wnarrowing]
   23 |     const double pts2[4][2] = {{0, 0}, {SIZE, 0}, {0, SIZE}, {SIZE, SIZE}};
      |                                         ^~~~
tmp/fig_02_sudoku2.cpp:23:55: warning: narrowing conversion of ‘SIZE’ from ‘int’ to ‘double’
[-Wnarrowing]
   23 |     const double pts2[4][2] = {{0, 0}, {SIZE, 0}, {0, SIZE}, {SIZE, SIZE}};
      |                                                       ^~~~
tmp/fig_02_sudoku2.cpp:23:63: warning: narrowing conversion of ‘SIZE’ from ‘int’ to ‘double’
[-Wnarrowing]
   23 |   const double pts2[4][2] = {{0, 0}, {SIZE, 0}, {0, SIZE}, {SIZE, SIZE}};
      |                                                               ^~~~
```

```
tmp/fig_02_sudoku2.cpp:23:69: warning: narrowing conversion of ‘SIZE’ from ‘int’ to ‘double’
[-Wnarrowing]
   23 | const double pts2[4][2] = {{0, 0}, {SIZE, 0}, {0, SIZE}, {SIZE, SIZE}};
      |                                                                     ^~~~

[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)")
```

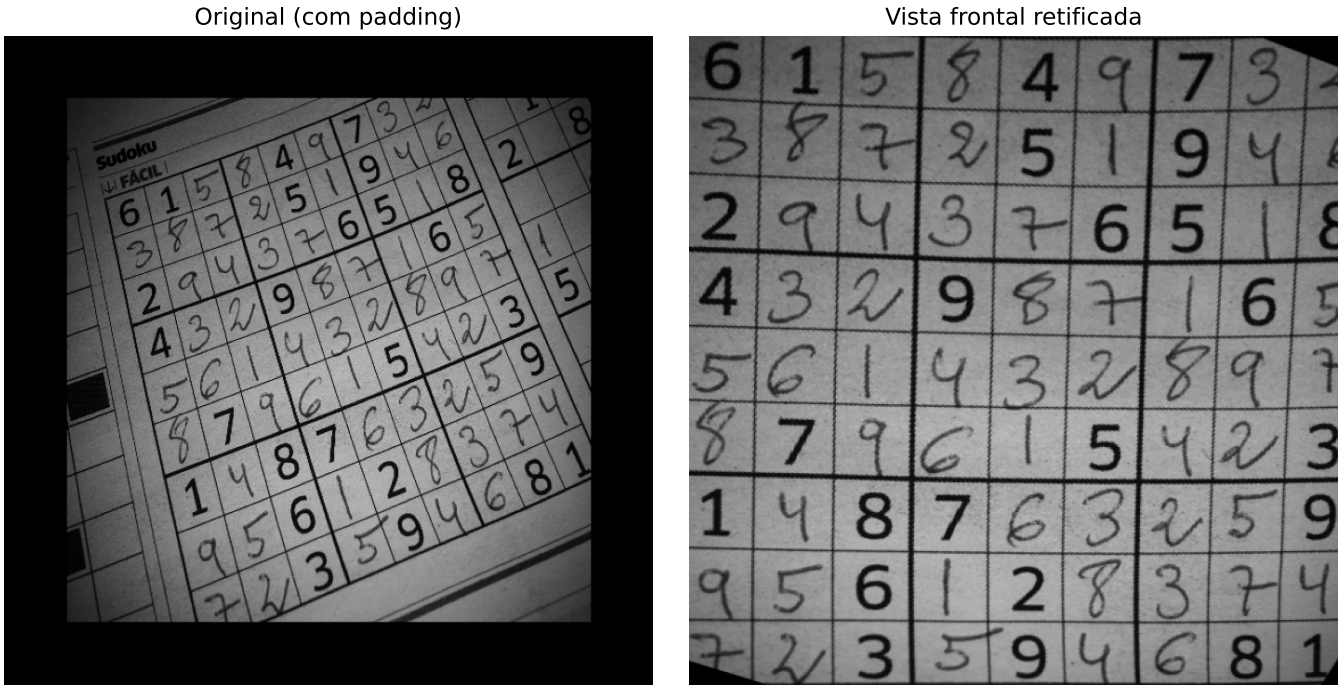
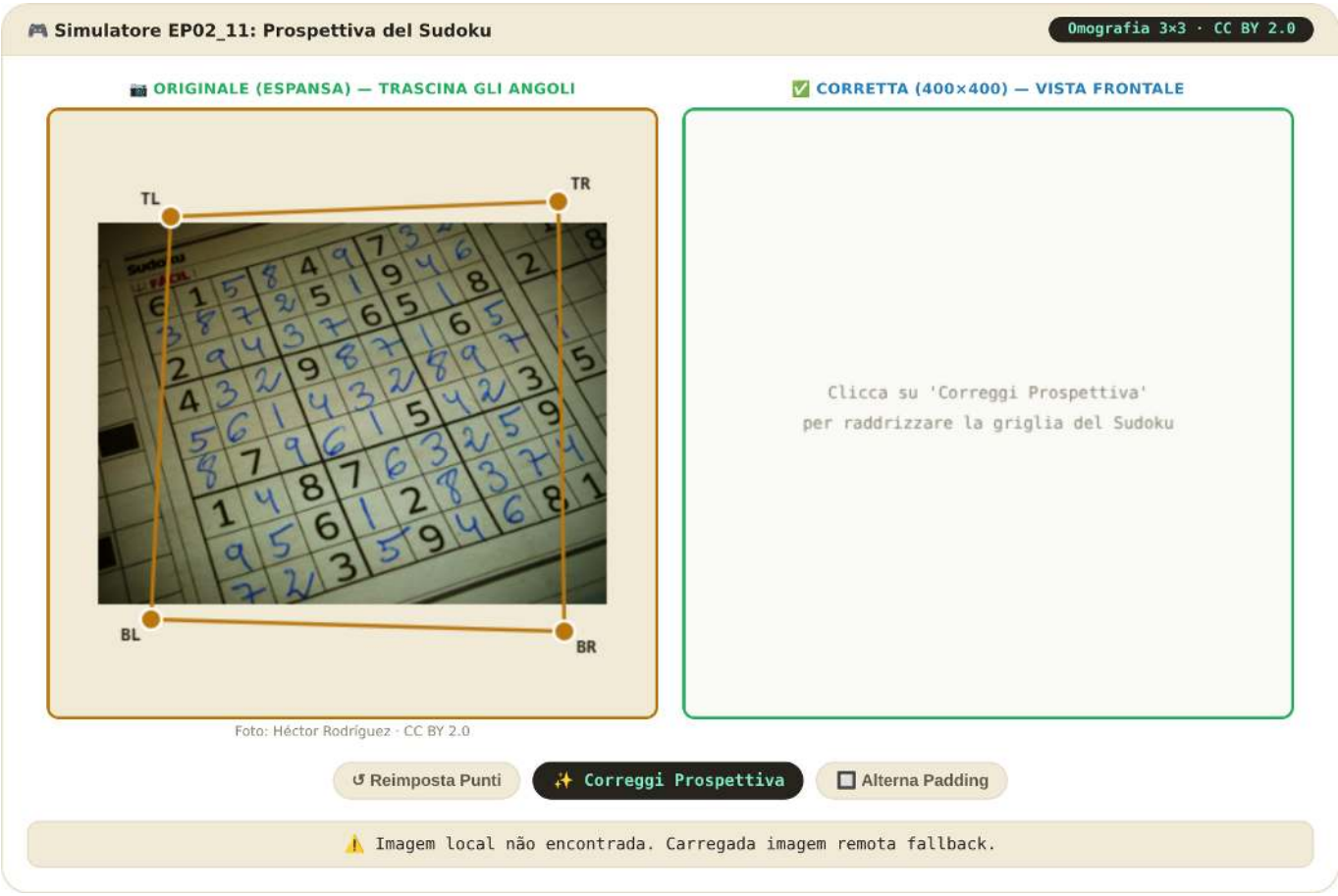


Figura 2.23: Correção de perspectiva por homografia 3×3: imagem com *padding* e a vista frontal retificada, via *mm::perspective_transform* (solve 8×8 dos 4 pontos + mapeamento inverso projetivo).

```
%%writefile EP02_11.cpp
// your solution
```

```
Overwriting EP02_11.cpp
```

```
TestSuite("EP02_11.cpp").run()
```



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.it/cap02/sim-ep0211-sudoku.html

Figura 2.24: Simulatore EP02_11: Correzione della Prospettiva del Sudoku (Omografia 3×3 con Ricampionamento Bilineare)

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

Capitolo 3

Operazioni Spaziali: Intensità, Istogramma e Filtraggio

Questo capitolo approfondisce l'elaborazione delle immagini nel dominio spaziale, partendo dalla manipolazione diretta dei pixel e degli istogrammi per l'enfatizzazione del contrasto, fino all'applicazione di filtri locali tramite convoluzione per la smussatura, la riduzione del rumore e il rilevamento dei bordi. L'obiettivo è sviluppare l'intuizione matematica e computazionale alla base di gran parte degli algoritmi moderni della Visione Artificiale.

3.1 Obiettivi

Al termine di questo capitolo, sarai in grado di:

- **Gestire intensità e pixel:** Eseguire operazioni aritmetiche sature (`mm::addm`, `mm::subm`) e logiche bit a bit (`mm::band`, `mm::bor`, `mm::bnot`) per la combinazione e la selezione di regioni di interesse (ROI), e applicare l'*alpha blending* (`mm::blend`) per la fusione ponderata di immagini;
- **Elaborare istogrammi:** Interpretare l'istogramma come diagnosi tonale e applicare l'equalizzazione globale tramite CDF (`mm::equalize`); nel percorso Python, anche l'equalizzazione adattiva (CLAHE) e la specifica dell'istogramma per il trasferimento del profilo tonale tra immagini;
- **Comprendere i fondamenti spaziali:** Capire il concetto di vicinato, il *padding* dei bordi (`mm::pad`) e la differenza tra correlazione incrociata (`mm::conv`) e convoluzione — inclusa la ragione per cui *kernel* asimmetrici come quello di Sobel producono risultati diversi nelle due operazioni;
- **Applicare il filtraggio di smoothing:** Usare il filtro della media (`mm::blur`, oppure `mm::conv` con *kernel* uniforme) e il filtro Gaussiano (`mm::gaussian`) per la riduzione del rumore, comprendendo il vantaggio della ponderazione radiale e della separabilità Gaussiana;
- **Applicare il filtraggio di enhancement:** Usare il Laplaciano w_4 e w_8 (`mm::laplacian`) per l'accentuazione isotropica dei bordi, l'operatore di Sobel (`mm::sobel`) per la magnitudine del gradiente — e, nel percorso Python, la decomposizione direzionale G_x, G_y e l'angolo —, e l'*Unsharp Masking* (`mm::usm`) per l'amplificazione delle alte frequenze controllata dal parametro k;
- **Utilizzare filtri d'ordine:** Applicare il filtro mediano (`mm::median`) per la rimozione del rumore sale e pepe, comprendendo perché la sua natura non lineare e la robustezza agli *outlier* lo rendano superiore ai filtri lineari in questo scenario;
- **Risolvere problemi pratici:** Combinare tecniche in *pipeline* di pre-processing (equalizzazione → Gaussiano → Canny; con CLAHE al posto dell'equalizzazione nel percorso Python) e utilizzare le funzioni di `morph` (`mm::conv`, `mm::histImg`, `mm::equalize`, `mm::drawImgKernel`) per l'analisi e la visualizzazione didattica di ogni fase.

3.2 Operazioni a Livello di Intensità

Il livello più elementare di elaborazione delle immagini agisce direttamente sui valori dei pixel, senza considerare il vicinato. Queste operazioni — chiamate **trasformazioni puntuali** (*point operations*) — sono

le più veloci dal punto di vista computazionale e costituiscono la base per tecniche più complesse.

Formalmente, una trasformazione puntuale può essere descritta come:

$$g(x, y) = T[f(x, y)] \tag{3.1}$$

dove $f(x, y)$ è l'immagine di ingresso, $g(x, y)$ è l'uscita e T è una funzione applicata a ciascun pixel individualmente.

3.2.1 Preparazione dell'Ambiente Pratico

Il blocco seguente carica la libreria `morph` dal repository (il modulo `morph.py` e, nel percorso C++, anche la `morph.hpp` utilizzata nel `#include` delle celle compilate).

```
import os, urllib.request

os.makedirs("tmp/state", exist_ok=True)  # artefatti di build del percorso C++ (.cpp, binario,
PNG)

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")

# Il kernel è Python anche nel percorso C++: `mm` (morph.py) viene usato dai
# simulatori, dalla visualizzazione delle figure che il binario C++ genera e dallo
# stato mm::Image tra le celle. cpp=True scarica anche il percorso compilato
# (morph.hpp + stb_image*.h), usato nell'#include delle celle %%writefile *.cpp.
import config
config.setup(cpp=True)
from morph import mm
import numpy as np
```

```
Ambiente pronto. Morph: 1.1.9 | OpenCV: 5.0.0
```

Come oggetto di studio lungo questo capitolo, utilizzeremo le immagini di vita selvatica presentate nelle Figura 3.1 e Figura 3.3. A partire da esse, esploreremo operazioni spaziali sull'intensità, istogrammi e filtraggio, analizzando i loro effetti sull'amplificazione, sulla levigatura, sulla riduzione del rumore e sulla rilevazione dei bordi, in modo da comprendere i fondamenti matematici e computazionali dell'elaborazione digitale delle immagini (PDI).

```
%%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 <iostream>
#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)")
```

Figura 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 Operazioni Aritmetiche

Le operazioni aritmetiche tra immagini sono ampiamente utilizzate nell'elaborazione digitale delle immagini (PDI) per combinare, confrontare o migliorare le informazioni. La **sottrazione di immagini** è particolarmente efficace per rilevare differenze tra due fotogrammi — ad esempio, nella rimozione dello sfondo statico nelle telecamere di sorveglianza:

$$g(x, y) = f_1(x, y) - f_2(x, y) \tag{3.2}$$

L'**addizione saturata** limita il risultato all'intervallo $[0, 255]$: i valori superiori a 255 vengono fissati a 255, evitando l'*overflow* silenzioso del tipo `uint8` (es.: $200 + 100 = 44$ anziché 300). La **sottrazione saturata** applica lo stesso principio sul lato inferiore: i valori negativi vengono fissati a 0.

⚠ Saturazione e *overflow*

Le operazioni aritmetiche su `uint8` subiscono un *overflow* silenzioso: $200+100 = 44$ (non 300). `mm::addm` e `mm::subm` eseguono la **saturazione automatica**, fissando il risultato in $[0, 255]$. Il *blending* utilizza pesi frazionari: `mm::blend` opera internamente in virgola mobile e solo successivamente arrotonda e satura a `uint8`.

La Figura 3.2 dimostra l'addizione di una costante (schiarimento) e la sottrazione di una costante (scurimento con saturazione a 0).

```
%%writefile tmp/fig_03_aritmetica.cpp
#define MM_OUT "tmp/fig_03_aritmetica.png"
//| label: fig-03-aritmetica
//| fig-cap: "Operações aritméticas saturadas: adição de constante (clareamento) e subtração
de constante (escurecimento com saturação em 0)."
//| 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_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)")
```

Figura 3.2: Operações aritméticas saturadas: adição de constante (clareamento) e subtração de constante (escurecimento com saturação em 0).

3.2.3 Miscelazione Ponderata (*Alpha Blending*)

La **miscelazione ponderata** (*alpha blending*) combina due immagini utilizzando pesi complementari α e $(1-\alpha)$:

$$g(x,y) = \alpha\, f_1(x,y) + (1-\alpha)\, f_2(x,y), \quad \alpha \in [0,1] \tag{3.3}$$

Quando $\alpha = 1$, si ottiene soltanto l'immagine f_1; quando $\alpha = 0$, soltanto f_2. I valori intermedi producono una transizione fluida tra le due, essendo ampiamente utilizzati nella composizione di immagini, sovrapposizione di livelli, filigrane ed effetti di fusione visiva.

Affinché la combinazione produca un risultato coerente, è necessario allineare preventivamente le regioni di interesse. Nella Figura 3.4, si ritaglia il volto del leopardo con `mm::crop(img_leop_gray, 250, H-300, 100, W-200)` e la regione facciale del mandrillo con `mm::crop(img_gray, 100, 400, 380, 530)`, in modo che occhi e struttura facciale risultino approssimativamente allineati. Il ritaglio del leopardo viene quindi ridimensionato (`mm::resize`) alle dimensioni del mandrillo prima della miscelazione.

`mm::blend` esegue l'operazione in virgola mobile — evitando *overflow* nei calcoli con pesi frazionari — e solo successivamente arrotonda e satura il risultato in `uint8`.

```
%%writefile tmp/fig_03_leopardo.cpp
#define MM_OUT "tmp/fig_03_leopardo.png"
// Compile: g++ -std=c++17 -o program program.cpp -I. -L. -lmorph `pkg-config --cflags --libs
opencv4` 2>/dev/null || g++ -std=c++17 -o program program.cpp morph.cpp
//| 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

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

```
int main() {
    std::string base = "https://upload.wikimedia.org/wikipedia/commons";
    std::string arquivo = "Leopard_%28Panthera_pardus%29_portrait.jpg";
    std::string url = base + "/9/92/" + arquivo;
    std::string caminho = "imagens/leopardo.jpg";

    mm::Image img_leop = mm::read(url);
    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)")
```

Figura 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"
//| label: fig-03-blend
//| fig-cap: "*Alpha blending* entre recortes alinhados de mandrill e do leopardo
(@fig-03-leopardo) para diferentes valores de  . Em  =1 vê-se apenas mandrill; em  =0, apenas o
leopardo; valores intermediários fundem os olhares das duas imagens proporcionalmente."
//| 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");
mm::Image img_leop_gray = mm::_read_state("tmp/state/img_leop_gray_12.png");
// [pdi:state-io:end]

    // 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(
        {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,
        {" =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
```

```
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)")
```

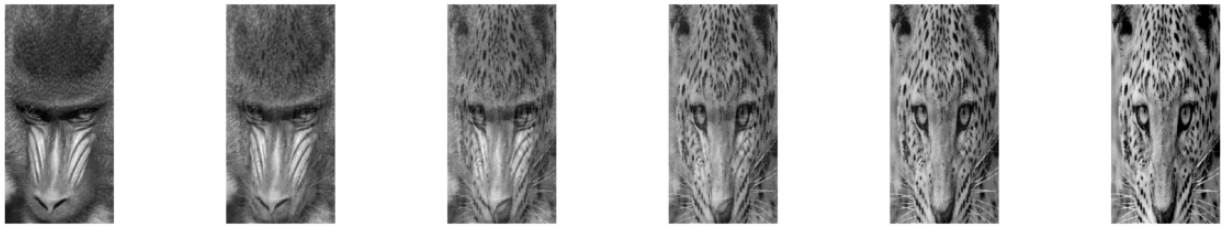


Figura 3.4: *Alpha blending* entre recortes alinhados de mandrill e do leopardo (Figura 3.3) para diferentes valores de . Em =1 vê-se apenas mandrill; em =0, apenas o leopardo; valores intermediários fundem os olhares das duas imagens proporcionalmente.

3.2.4 Operazioni Logiche e Maschere Bit a Bit

Le operazioni logiche bit a bit (AND, OR e NOT) agiscono direttamente sui bit di ciascun pixel e costituiscono la base per la creazione e l'applicazione di **maschere** (*masks*) — immagini binarie con solo 0 (nero) e 255 (bianco) utilizzate per isolare **Regioni di Interesse** (**ROI**).

Il comportamento di ciascuna operazione deriva dalla rappresentazione binaria del 255 (`11111111`) e dello 0 (`00000000`):

- **AND** con la maschera: dove $m = 255$, i bit originali vengono preservati; dove $m = 0$, il pixel viene azzerato. Risultato: ritaglio della ROI.

$$g(x, y) = f(x, y) \text{ AND } m(x, y) \tag{3.4}$$

- **OR** con la maschera: dove $m = 255$, il pixel viene forzato al bianco; dove $m = 0$, il valore originale viene mantenuto. Risultato: illuminazione della ROI.
- **NOT** (senza maschera): inverte tutti i bit ($g = 255 - f$), producendo il negativo fotografico dell'immagine.

La Figura 3.5 illustra le tre operazioni applicate all'immagine del mandrillo con una maschera circolare.

```
%%writefile tmp/fig_03_logica.cpp
#define MM_OUT "tmp/fig_03_logica.png"
//| label: fig-03-logica
//| fig-cap: "Operações lógicas bit a bit com máscara circular: AND (isolamento da ROI), OR
(iluminação da ROI) e NOT (negativo)."
//| echo: true
//| output: true

#include "morph.hpp"
#include <algorithm>
#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");
// [pdi:state-io:end]

    // img_gray is provided as input

    int h = img_gray.h;
    int w = img_gray.w;

    // Máscara circular preenchida, centrada na imagem (mm.circle desenha o
    // disco; a versão didática, teste de raio pixel a pixel, é 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);
```

```
    // Operações via morph
    mm::Image img_not = mm::bnot(img_gray);            // NOT: negativo fotográfico
    mm::Image img_and = mm::band(img_gray, mask_circ); // preserva apenas a ROI circular
    mm::Image img_or  = mm::bor(img_gray, mask_circ);  // ilumina a região da máscara

    mm::show(
        std::vector<mm::Image>{img_gray, img_and, img_or, img_not},
        MM_OUT,
        std::vector<std::string>{"Original", "AND (ROI circular)", "OR (ilumina ROI)", "NOT
(negativo)"},
        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 circular)
[3] OR (ilumina ROI)
[4] NOT (negativo)
```

```
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)")
```

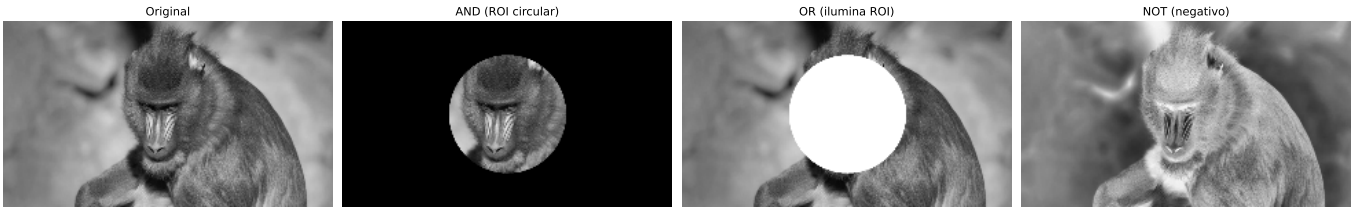


Figura 3.5: Operações lógicas bit a bit com máscara circular: AND (isolamento da ROI), OR (iluminação da ROI) e NOT (negativo).

3.3 Istogramma delle Immagini

L'**istogramma** di un'immagine in scala di grigi è una funzione discreta che descrive la distribuzione delle frequenze delle intensità:

$$h(r_k) = n_k, \quad k = 0, 1, \ldots, L-1 \tag{3.5}$$

dove r_k è il k-esimo livello di intensità, n_k è il numero di pixel con tale intensità e L è il totale dei livelli (tipicamente 256 per 8 bit). L'istogramma normalizzato stima la probabilità di ciascun livello:

$$p(r_k) = \frac{n_k}{MN} \tag{3.6}$$

dove MN è il totale dei pixel. Essendo una **statistica globale**, l'istogramma non contiene informazioni posizionali, ma rivela caratteristiche essenziali come luminosità media, contrasto e distribuzione tonale. In pratica, `mm::hist(img)` restituisce il vettore delle occorrenze $h(r_k)$, che serve sia per la visualizzazione (tramite `mm::histImg`) sia per calcoli come la **funzione di distribuzione cumulativa (CDF)** e l'equalizzazione.

i Interpretazione dell'Istogramma

- **Stretto a sinistra:** immagine sottoesposta (scura).
- **Stretto a destra:** immagine sovraesposta (chiara).
- **Concentrato al centro:** basso contrasto.
- **Distribuito su tutta la gamma:** alto contrasto, buon utilizzo dei toni disponibili.

La Figura 3.6 presenta l'istogramma dell'immagine del mandrillo, nonché versioni scurita (`mm::subm`) e schiarita (`mm::addm`). Si osserva lo spostamento della distribuzione delle intensità rispettivamente verso sinistra e verso destra. Si noti che l'intervallo rappresentato sull'asse x non corrisponde necessariamente all'intera gamma da 0 a 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 <string>
#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_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)")
```

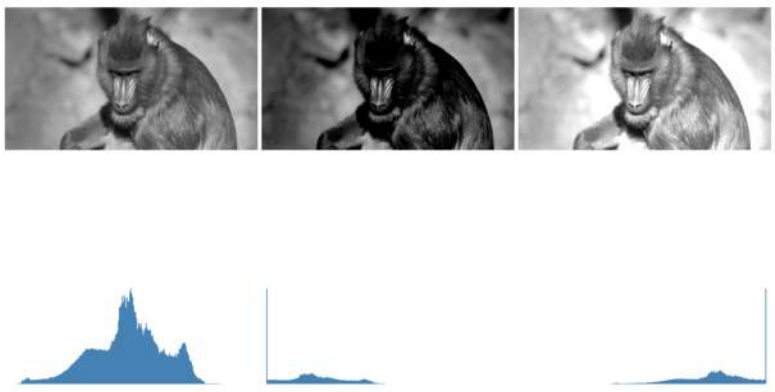


Figura 3.6: Histogramas da imagem original, de uma versão escurecida (−80) e de uma clareada (+80). A subtração/adição satura em 0 e 255.

3.3.1 Equalizzazione dell'Istogramma

L'**equalizzazione dell'istogramma** ridistribuisce le intensità affinché l'istogramma risultante sia il più uniforme possibile. La mappatura è data dalla **funzione di distribuzione cumulativa (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 trasformazione è monotòna: i livelli frequenti ricevono intervalli più ampi nel dominio di uscita (maggiore separazione → maggiore contrasto), mentre i livelli rari vengono compressi.

L'algoritmo completo, in cinque fasi, è presentato nella Tabella 3.1.

Tabella 3.1: Algoritmo di equalizzazione dell'istogramma.

Fase	Operazione	Formula
1	**Istogramma**	$h[k] \leftarrow$ numero di pixel con intensità k, $k = 0 \dots L-1$
2	**Probabilità**	$p[k] \leftarrow h[k]/MN$
3	**CDF**	$\text{cdf}[k] \leftarrow \sum_{j=0}^{k} p[j]$ (somma cumulativa)
4	***Look-Up Table* (mappatura)**	$\text{lut}[k] \leftarrow \text{round}(\text{cdf}[k] \times (L-1))$
5	**Applicazione**	$g[i,j] \leftarrow \text{lut}[f[i,j]]$ (per ogni pixel)

Si noti nella Figura 3.7 che l'equalizzazione **ridistribuisce** i toni esistenti in posizioni più distanziate lungo l'intervallo $[0, L-1]$, ma non crea nuovi toni — l'immagine equalizzata continua ad avere esattamente 3 toni distinti, ora in $\{1, 5, 7\}$ invece di $\{2, 3, 4\}$.

```
%%writefile tmp/fig_03_equalizacao_didatica.cpp
#define MM_OUT "tmp/fig_03_equalizacao_didatica.png"
// Compile with: g++ -std=c++17 -o program program.cpp -I/path/to/morph.hpp/include
#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);
    unsigned char data5[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) = data5[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)")
```

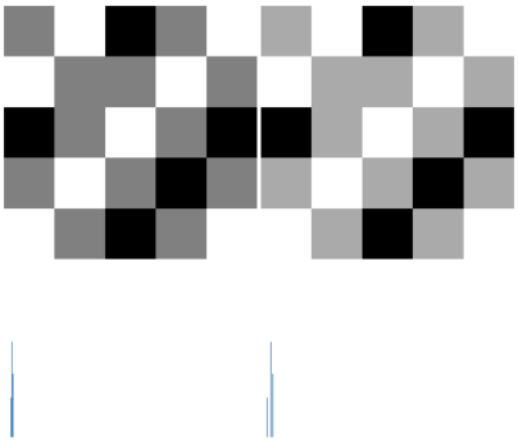


Figura 3.7: Equalização de histograma numa imagem 5×5 de 3 bits (L=8): tons concentrados em {2,3,4} são redistribuídos pela CDF. *mm::equalize(img, 3)* faz o mapeamento.

Limitazione: l'equalizzazione globale può enfatizzare eccessivamente il rumore e produrre contrasto eccessivo nelle regioni omogenee. Il **CLAHE** (*Contrast Limited Adaptive Histogram Equalization*) riduce questo problema applicando l'equalizzazione a blocchi locali (*tiles*) e limitando l'altezza dei picchi dell'istogramma prima dell'equalizzazione.

La Figura 3.8 confronta l'immagine originale, l'equalizzazione globale tramite `mm::equalize` e il CLAHE di OpenCV, mostrando anche gli istogrammi risultanti. Diversamente dall'equalizzazione globale, che utilizza una singola trasformazione basata sulla CDF dell'intera immagine, il CLAHE adatta il contrasto a ciascuna regione, essendo particolarmente utile in immagini con illuminazione non uniforme.

Nell'esempio sono stati utilizzati `clipLimit=2.0` e `tileGridSize=(32,32)`. Il parametro `clipLimit` definisce quanto i picchi dell'istogramma locale possono crescere prima di essere tagliati (*clipped*). In OpenCV, questo valore è un fattore relativo: il limite effettivo è approssimativamente calcolato come `clipLimit × (numero di pixel del blocco / numero di livelli di grigio)`. Ad esempio, in un blocco con 4096 pixel e un'immagine a 8 bit (256 livelli di grigio), la frequenza media per livello è $4096/256 = 16$. Pertanto, `clipLimit=2.0` consente picchi di circa $2 \times 16 = 32$ occorrenze prima del taglio. Le occorrenze eccedenti non vengono scartate: vengono ridistribuite tra gli altri livelli di grigio dell'istogramma, riducendo la concentrazione eccessiva in pochi livelli ed evitando un'amplificazione esagerata del contrasto locale. Valori minori limitano maggiormente il contrasto e riducono l'amplificazione del rumore, mentre valori maggiori consentono un miglioramento più intenso ma possono introdurre artefatti.

- `clipLimit=1.0`: miglioramento morbido e conservativo;
- `clipLimit=2.0`: buon equilibrio tra contrasto e naturalezza;
- `clipLimit=4.0`: maggiore evidenziazione dei dettagli locali;
- `clipLimit=8.0`: contrasto aggressivo, con possibile amplificazione del rumore.

Pertanto, il CLAHE tende a produrre risultati più naturali rispetto all'equalizzazione globale, specialmente in immagini con ombre, riflessi o illuminazione disomogenea.

```
%%writefile tmp/fig_03_equalizacao.cpp
#define MM_OUT "tmp/fig_03_equalizacao.png"
//| label: fig-03-equalizacao
//| fig-cap: "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."
//| echo: true
//| output: true
```

```cpp
#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_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)", "Histograma - original",
"Histograma - equalizado"},
        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] Histograma - original
[4] Histograma - equalizado
```

```python
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)")
```

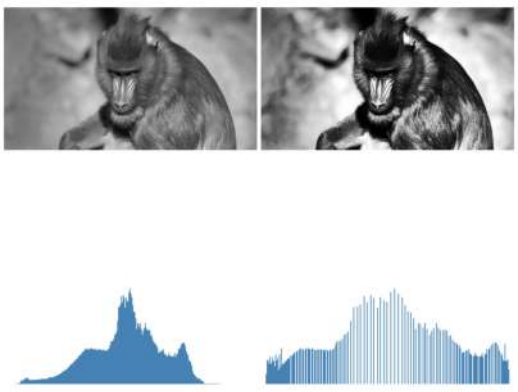


Figura 3.8: Equalização de histograma global (mm::equalize, via CDF) e os histogramas antes/depois. CLAHE (adaptativa) fica só na trilha Python — não tem equivalente em morph.hpp.

3.3.2 Specifica dell'istogramma

Mentre l'equalizzazione impone una distribuzione uniforme, la **specifica dell'istogramma** (*histogram matching*) consente che l'istogramma dell'immagine di uscita segua una distribuzione **arbitraria** — ad esempio, l'istogramma di un'altra immagine di riferimento.

La procedura prevede tre fasi:

1. Calcolare la CDF dell'immagine di ingresso: $P_r(r_k)$.
2. Calcolare la CDF dell'immagine di riferimento: $P_z(z_k)$.
3. Per ogni livello r_k, trovare il livello z che minimizza $|P_z(z) - P_r(r_k)|$.

$$T(r_k) = \arg\min_z |P_z(z) - P_r(r_k)| \tag{3.8}$$

Nella Figura 3.9, trasferiamo il profilo tonale del leopardo (Figura 3.3) all'immagine del mandrillo — un'applicazione diretta del concetto visto nel *blending*: invece de fondere i pixel, qui fondiamo le distribuzioni tonali.

```
%%writefile tmp/fig_03_especificacao.cpp
#define MM_OUT "tmp/fig_03_especificacao.png"
//| 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

#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]

    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)")
```

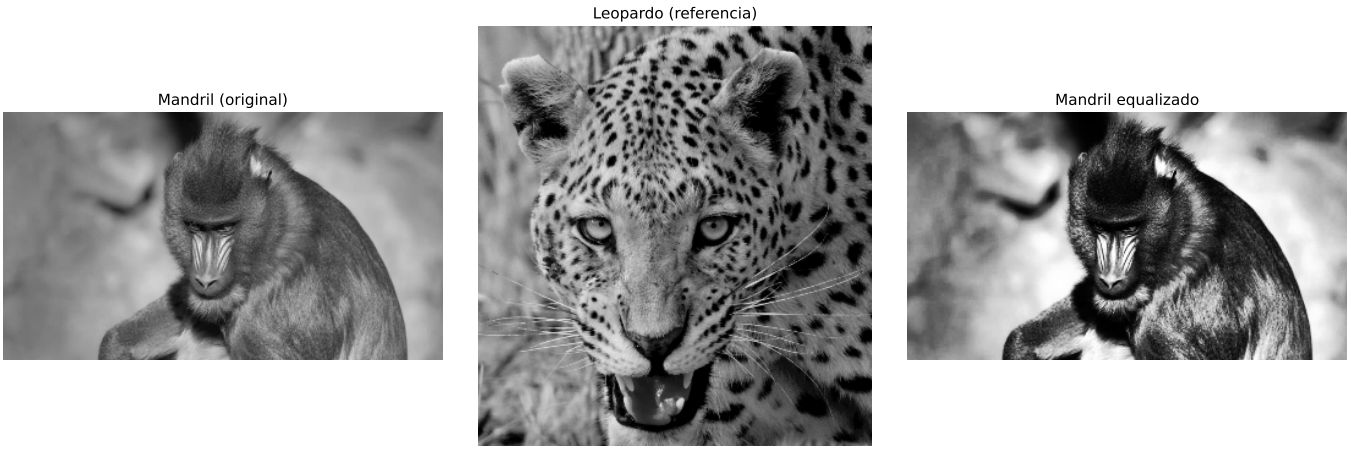


Figura 3.9: Especificação de histograma (mapear o mandril para o perfil tonal do leopardo) precisa da CDF inversa da referência — fica só na trilha Python. Aqui, a equalização global do mandril, como comparação.

3.4 Fondamenti Spaziali: Intorno, Convoluzione e *Kernel*

Le operazioni di filtraggio spaziale non agiscono su un singolo pixel isolato, ma su un **intorno** che lo circonda. A tale scopo, si utilizza una piccola matrice di coefficienti denominata **kernel** (o maschera), che percorre l'intera immagine attraverso una **finestra scorrevole** (*sliding window*).

Le finestre più comuni sono 3×3, 5×5 e 7×7. In una finestra 3×3, ad esempio, il pixel centrale viene elaborato insieme ai suoi otto vicini immediati. In ogni posizione della finestra, i valori dei pixel vengono combinati con i coefficienti del *kernel*, producendo un nuovo valore per il pixel centrale.

3.4.1 Intorno

Si consideri una finestra 3×3 centrata sul pixel (x, y):

$$\begin{bmatrix} (x-1,y-1) & (x,y-1) & (x+1,y-1) \\ (x-1,y) & (x,y) & (x+1,y) \\ (x-1,y+1) & (x,y+1) & (x+1,y+1) \end{bmatrix} \tag{3.9}$$

In generale, una finestra di dimensioni $(2a+1) \times (2b+1)$ comprende tutti i pixel situati fino a a posizioni in orizzontale e fino a b posizioni in verticale rispetto al pixel centrale. Pertanto, una finestra 3×3 corrisponde a $a = b = 1$, una finestra 5×5 a $a = b = 2$, e così via.

Matematicamente, l'intorno è definito da

$$\mathcal{V}(x,y) = \{(x+s,\, y+t) : -a \leq s \leq a,\, -b \leq t \leq b\} \tag{3.10}$$

3.4.2 Trattamento dei Bordi

I pixel vicini ai bordi hanno parte del loro intorno al di fuori dell'immagine. Per applicare filtri in queste regioni, è necessario definire come verranno ottenuti i valori esterni. Le tre strategie più comuni (con la costante equivalente di OpenCV tra parentesi) sono:

- **Zero-padding** (`BORDER_CONSTANT`): completa la regione esterna con zeri.
- **Replicazione** (`BORDER_REPLICATE`): ripete il valore del pixel del bordo.
- **Riflessione** (`BORDER_REFLECT_101`): rispecchia i pixel vicini, senza ripetere quello del bordo.

`mm::conv` utilizza la riflessione per impostazione predefinita, poiché preserva meglio la continuità dei livelli di grigio e riduce gli artefatti nel trattamento dei bordi.

L'esempio seguente confronta le tre strategie con `mm::pad` su una matrice 3×3. Osserva come ciascuna riempie i pixel esterni necessari per applicare un filtro 3×3 anche agli angoli.

```
%%writefile tmp/mm_out_1.cpp
// Compile with: g++ -std=c++17 main.cpp -o main
#include "morph.hpp"
#include <iostream>
#include <vector>
#include <string>

int main() {
    mm::Image img(3, 3);
    img.at(0, 0) = 1; img.at(0, 1) = 2; img.at(0, 2) = 3;
    img.at(1, 0) = 4; img.at(1, 1) = 5; img.at(1, 2) = 6;
    img.at(2, 0) = 7; img.at(2, 1) = 8; img.at(2, 2) = 9;

    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 std::string& nome : bordas) {
            std::cout << nome << std::endl;
            mm::Border border;
            if (nome == "constant") border = mm::Border::CONSTANT;
            else if (nome == "replicate") border = mm::Border::REPLICATE;
            else border = mm::Border::REFLECT101;
            std::cout << mm::drawImg(mm::pad(img, b, border)) << 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
```

Si noti che il risultato di un filtro può variare significativamente a seconda del trattamento adottato per i bordi dell'immagine.

Nella `morph.hpp`, le funzioni di filtraggio (`mm::conv`, `mm::blur`, `mm::gaussian`, `mm::laplacian`, `mm::usm`) applicano *padding* per **riflessione** (`mm::Border::REFLECT101`) per impostazione predefinita — lo stesso comportamento di `cv2.filter2D`. La variante didattica `mm::conv0` utilizza `mm::Border::KEEP`: i pixel del bordo mantengono il valore originale, senza il filtro. Invece, `mm::sobel` e `mm::prewitt` lasciano il bordo a zero (calcolano solo l'interno).

3.4.3 Correlazione vs. Convoluzione

Esistono due meccanismi matematicamente correlati.

Correlazione incrociata (*cross-correlation*) — il *kernel* viene applicato direttamente:

$$g(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t)\, f(x+s,\, y+t) \tag{3.11}$$

Convoluzione bidimensionale — il *kernel* viene ruotato di 180° prima dell'applicazione:

$$g(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t)\, f(x-s,\, y-t) \tag{3.12}$$

Per *kernel* simmetrici (Gaussiano, Laplaciano, media) le due operazioni producono risultati identici. Per *kernel* asimmetrici (Sobel, Prewitt) la differenza è significativa, come mostrano gli esempi seguenti.

3.4.3.1 Correlazione (`mm::conv`)

```cpp
%%writefile tmp/mm_out_2.cpp
// Convert image 4x4 (uint8) and asymmetric 3x3 kernel to Python to C++
#include "morph.hpp"
#include <iostream>

int main() {
    // 4x4 image (uint8) and asymmetric 3x3 kernel
    mm::Image img(4, 4);
    for (int y = 0; y < 4; y++) {
        for (int x = 0; x < 4; x++) {
            img.at(y, x) = static_cast<unsigned char>(y * 4 + x + 1);
        }
    }

    mm::Kernel w{{0, 1, 2}, {0, 0, 0}, {0, 0, 0}};
    mm::Image corr = mm::conv(img, w, mm::Border::CONSTANT); // zero outside the image

    std::cout << "Original image:" << std::endl;          std::cout << mm::drawImg(img) <<
std::endl;
    std::cout << "Kernel:" << std::endl;                  std::cout << mm::drawImg(w) <<
std::endl;
    std::cout << "Correlation result:" << std::endl;      std::cout << mm::drawImg(corr) <<
std::endl;

    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
```

```
Original image:
 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

Correlation result:
 0  0  0  0
 5  8 11  4
17 20 23  8
29 32 35 12
```

`mm::conv` esegue la correlazione, cioè applica il *kernel* esattamente nell'orientamento fornito.

3.4.3.2 Convoluzione

```
%%writefile tmp/mm_out_3.cpp
// Compile with: g++ -std=c++17 -I. programa.cpp -o programa -lboost_filesystem -lboost_system

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

int main() {
    // repetidos aqui para a célula ser independente
    mm::Image img(4, 4);  // criando imagem 4x4
    // Preenchendo valores manualmente
    int vals[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) = vals[y][x];

    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 convolução utilizza il *kernel* ruotato di 180°. Per riprodurre la definizione matematica di convolução, si ruota il *kernel* (qui, `[[0,1,2],[0,0,0],[0,0,0]]` → `[[0,0,0],[0,0,0],[2,1,0]]`) prima di applicare `mm::conv`.

3.4.4 Il Ruolo del *Kernel*

I coefficienti del *kernel* determinano completamente l'effetto prodotto dal filtro, come riassunto nella Tabella 3.2.

Tabella 3.2: Interpretazione tipica dei coefficienti del *kernel*.

Caratteristica	Effetto tipico
Coefficienti positivi con somma 1	Attenuazione (*passa-basso*)
Somma pari a 0, con valori positivi e negativi	Rilevamento dei bordi (*passa-alto*)
Coefficiente centrale positivo dominante e vicini negativi	Enfasi della nitidezza
Coefficienti asimmetrici	Gradiente direzionale

Esempi:

Attenuazione:

$$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$$

Rilevamento dei bordi:

$$\begin{bmatrix}-1 & -1 & -1\\ -1 & 8 & -1\\ -1 & -1 & -1\end{bmatrix}$$

Enfasi della nitidezza:

$$\begin{bmatrix}0 & -1 & 0\\ -1 & 5 & -1\\ 0 & -1 & 0\end{bmatrix}$$

Gradiente direzionale (Sobel):

$$\begin{bmatrix}-1 & 0 & 1\\ -2 & 0 & 2\\ -1 & 0 & 1\end{bmatrix}$$

La Figura 3.10 dimostra il meccanismo passo dopo passo: per ogni posizione della finestra, si moltiplica ciascun coefficiente del *kernel* per il pixel corrispondente dell'intorno e si sommano i prodotti ottenuti. Il risultato è esattamente il valore definito dalla Equazione 3.11 per quella posizione dell'immagine. Sebbene le immagini prodotte da `mm::conv0` e `cv2.filter2D` (o `mm::conv`) siano visivamente molto simili, l'implementazione basata su OpenCV è migliaia di volte più rapida, come mostrato di seguito.

Prestazioni: cicli Python vs. operazioni vettorializzate

La funzione `mm::conv0` implementa la correlazione direttamente in Python tramite cicli annidati. Sebbene questo approccio sia adeguato a fini didattici, esegue un numero elevato di operazioni e diventa lento per immagini di dimensioni maggiori.
Invece, `mm::conv` utilizza `cv2.filter2D`, implementato in C++ e ottimizzato per operazioni matriciali. Nell'esempio presentato, la versione vettorializzata è risultata oltre **3000 volte più rapida** rispetto all'implementazione didattica, producendo un risultato visivamente equivalente.
Le differenze numeriche osservate si concentrano principalmente sui bordi dell'immagine. In `mm::conv0`, i pixel del bordo rimangono invariati, mentre `mm::conv` adotta una strategia di riflessione dei bordi (`cv2.BORDER_REFLECT_101`, predefinita di `cv2.filter2D`).
Pertanto, `mm::conv0` dovrebbe essere utilizzata per comprendere l'algoritmo, mentre `mm::conv` è l'opzione consigliata per applicazioni pratiche.

```
%%writefile tmp/fig_03_convolucao_passo.cpp
#define MM_OUT "tmp/fig_03_convolucao_passo.png"
//| label: fig-03-convolucao-passo
//| fig-cap: "Correlação com *kernel* de média 3×3: versão didática mm::conv0 (bordas
preservadas) vs. mm::conv (borda refletida). A diferença se concentra nas bordas."
//| 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_leop_gray = mm::_read_state("tmp/state/img_leop_gray_12.png");
// [pdi:state-io:end]
```

```
    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]:" << std::endl;
    std::cout << "  original = " << (int)img_gray.at(251, 251) << std::endl;
    std::cout << "  conv0    = " << (int)img_conv0.at(251, 251) << std::endl;
    std::cout << "  conv     = " << (int)img_conv.at(251, 251) << std::endl;

    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]
}
```

```
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)")
```

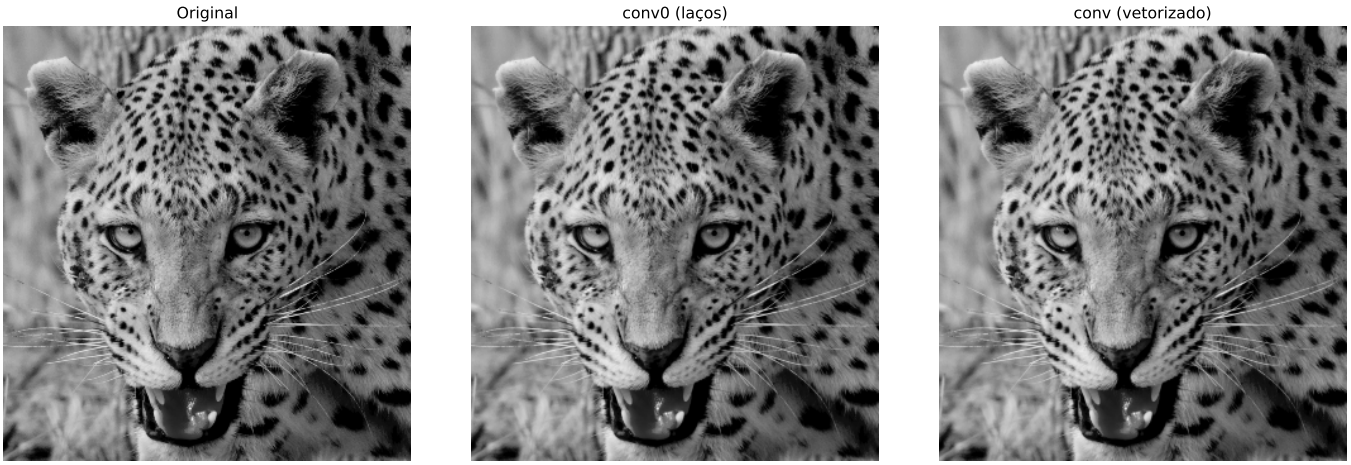


Figura 3.10: Correlação com *kernel* de média 3×3: versão didática mm::conv0 (bordas preservadas) vs. mm::conv (borda refletida). A diferença se concentra nas bordas.

```
%%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 Esempio Numerico: Correlazione Passo per Passo

Per rendere concreto il meccanismo della Equazione 3.11, si consideri il *kernel* di media 3×3 ($a = b = 1$, tutti i coefficienti $= 1/9 \approx 0{,}111$) applicato al *patch* 5×5 estratto dall'immagine del leopardo. La Figura 3.11 mostra il *patch* con griglia ed evidenzia in giallo la finestra 3×3 centrata sul pixel $[1, 1]$:

```
%%writefile tmp/fig_03_patch.cpp
#define MM_OUT "tmp/fig_03_patch.png"
//| 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
```

```cpp
#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]

    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) << "\n";

    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)")
```

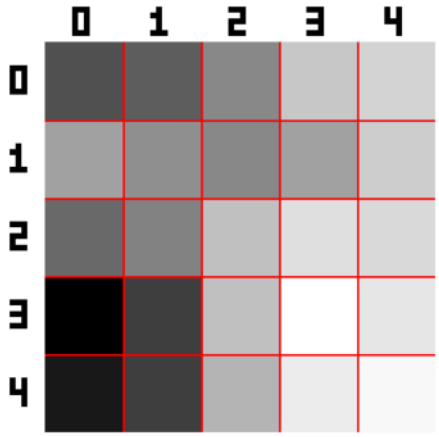


Figura 3.11: *Patch* 5×5 extraído da imagem do leopardo (posição [250:255, 250:255]). A janela amarela destaca a vizinhança 3×3 centrada no pixel [1,1] onde a correlação será calculada.

Per illustrare il calcolo della correlazione, si consideri il pixel nella posizione $[1, 1]$ del *patch* 5×5 mostrato nella Figura 3.11.. Tale posizione è stata scelta solo per comodità didattica, poiché possiede un intorno 3×3 completo attorno a sé.

I valori di tale intorno corrispondono alla sottomatrice in alto a sinistra del *patch*:

$$\text{vicinato} = \begin{bmatrix} 91 & 95 & 108 \\ 106 & 107 & 108 \\ 102 & 103 & 107 \end{bmatrix}$$

Applicando la Equazione 3.11 con il kernel di media:

$$g[1,1] = \frac{91 + 95 + 108 + 106 + 107 + 108 + 102 + 103 + 107}{9} = \frac{927}{9} = 103$$

Il risultato (103) è leggermente inferiore al valore originale del pixel centrale (107), poiché la media incorpora vicini di minore intensità, producendo l'effetto di smussamento. In pratica, l'algoritmo avvia l'elaborazione in $[0, 0]$ e ripete lo stesso calcolo per ogni posizione dell'immagine, spostando la finestra fino a coprire l'intero dominio.

3.5 Filtraggio Spaziale di Levigatura

I filtri di levigatura (*smoothing filters*) attenuano le variazioni brusche di intensità, riducendo rumore e dettagli ad alta frequenza. Sono filtri **passa-basso** — preservano le componenti a bassa frequenza (strutture grandi) e attenuano quelle ad alta frequenza (rumore, bordi).

3.5.1 Filtro a Media (*Box Filter*)

Il filtro a media utilizza un *kernel* uniforme di dimensione $n \times n$, dove tutti i coefficienti valgono $1/n^2$:

$$w_{\text{media}} = \frac{1}{n^2} \begin{bmatrix} 1 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 1 \end{bmatrix}_{n \times n} \tag{3.13}$$

Ogni pixel di uscita è la media aritmetica degli n^2 pixel del suo intorno. Si noti che la somma dei coefficienti è sempre 1 — la luminosità media dell'immagine viene preservata. *Kernel* più grandi producono una levigatura più aggressiva, ma sfumano progressivamente i bordi.

La Figura 3.12 mostra l'effetto del filtro di media con *kernel* 3×3, 7×7 e 15×15 su un dettaglio dell'immagine del leopardo. I risultati sono stati ottenuti con `mm::blur`, che implementa il filtro di media tramite la funzione `cv2.blur`, equivalente alla convoluzione dell'immagine con un *kernel* uniforme i cui coefficienti sono $h(x, y) = 1/N^2$; in modo equivalente, lo stesso risultato può essere ottenuto con `mm::conv`, calcolando $(g = f * h)$. Man mano che il *kernel* aumenta, più pixel contribuiscono a ciascun valore di uscita, intensificando la levigatura, riducendo il rumore e rendendo i dettagli fini e i bordi progressivamente più sfocati.

```
%%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 <string>
#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

    // Detalhe da região do olho
    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));
    }

    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)")
```

Figura 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 Filtro Gaussiano

Il filtro gaussiano pesa i pixel dell'intorno secondo una funzione gaussiana bidimensionale:

$$G(s,t) = \frac{1}{2\pi\sigma^2}\, e^{-\frac{s^2+t^2}{2\sigma^2}} \tag{3.14}$$

dove σ è la deviazione standard e controlla il raggio di influenza. I pixel più vicini al centro hanno peso maggiore; i pixel distanti vengono progressivamente ignorati.

A Figura 3.13 apresenta o *kernel* Gaussiano 5×5 gerado para $\sigma = 1$. O *kernel* foi costruído a partir do produto externo de due vettori Gaussiani unidimensionali e successivamente normalizzato affinché la somma dei suoi coefficienti sia uguale a 1. Si osserva che i pesi maggiori si concentrano al centro della matrice, decrescendo radialmente verso i bordi. Questa distribuzione fa sì che i pixel centrali abbiano maggiore influenza sul risultato del filtraggio, contribuendo a una smussatura più naturale e con una migliore conservazione dei bordi rispetto al filtro della media.

```
%%writefile tmp/fig_03_gauss_kernel.cpp
#define MM_OUT "tmp/fig_03_gauss_kernel.png"
// Compile with: g++ -std=c++17 program.cpp -o program $(pkg-config --cflags --libs opencv4)
-lmorph
#include "morph.hpp"
#include <iostream>
#include <iomanip>
#include <vector>

//| 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

int main() {
```

```
    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) << "  ";
        }
        std::cout << "\n";
    }
    std::cout << std::fixed << std::setprecision(4)
              << "Peso central [2,2] = " << w.at(2, 2)
              << "   |   canto [0,0] = " << 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)")
```

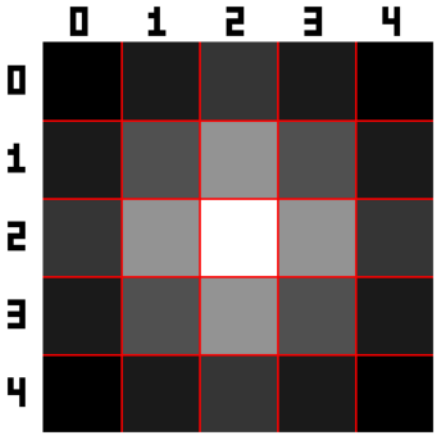


Figura 3.13: *Kernel* Gaussiano 5×5 (=1): resposta ao impulso do filtro — pesos maiores no centro, decrescendo radialmente.

i Vantaggio computazionale della separabilità

Si consideri un *kernel* quadrato di dimensione $n \times n$. Se tale filtro è separabile (come quello gaussiano), la convoluzione 2D può essere decomposta in due convoluzioni 1D: una orizzontale e una verticale. In questo caso, il costo per pixel passa da circa $O(n^2)$ operazioni (convoluzione 2D diretta) a $O(2n)$ operazioni (due convoluzioni 1D). La complessità viene quindi ridotta in modo significativo, rendendo l'elaborazione più efficiente.

Rispetto al filtro medio, il filtro gaussiano:

- **Preserva meglio i bordi** — la ponderazione radiale smussa senza creare transizioni brusche;
- **Non introduce anelli** (*ringing*) nel dominio della frequenza, poiché la gaussiana è la propria trasformata di Fourier (Capitolo 5);
- **È controllato da** σ — aumentare σ equivale ad aumentare il raggio di smussamento in modo continuo e prevedibile.

La Figura 3.14 confronta i filtri media e gaussiano applicati all'immagine del leopardo utilizzando una finestra 9×9. Il filtro media è stato implementato tramite convoluzione con un *kernel* uniforme, in cui tutti gli 81 pixel dell'intorno possiedono lo stesso peso (1/81), mentre il filtro gaussiano è stato ottenuto con `cv2.GaussianBlur`, utilizzando pesi definiti da una distribuzione gaussiana. Entrambi riducono il rumore e levigano l'immagine, ma il filtro gaussiano preserva meglio i bordi e i dettagli locali, come si può osservare nella regione ingrandita dell'occhio.

La Figura 3.14 confronta i filtri media e gaussiano applicati a un dettaglio dell'immagine del leopardo con *kernel* 9×9. Il filtro media è stato ottenuto con `mm::blur`, equivalente alla convoluzione con un *kernel* uniforme i cui coefficienti valgono 1/81, mentre il filtro gaussiano è stato ottenuto con `mm::gaussian`, equivalente alla convoluzione con un *kernel* generato a partire da una distribuzione gaussiana. Entrambi promuovono levigatura e riduzione del rumore, ma il filtro gaussiano attribuisce un peso maggiore ai pixel centrali dell'intorno, preservando meglio i bordi e i dettagli locali, come si può osservare nella regione ingrandita dell'occhio.

```
%%writefile tmp/fig_03_gauss.cpp
#define MM_OUT "tmp/fig_03_gauss.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_gray = mm::_read_state("tmp/state/img_gray_45.png");
// [pdi:state-io:end]

    //| label: fig-03-gauss
    //| fig-cap: "Comparação 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

    mm::Image img_media9 = mm::blur(img_gray, 9);
    mm::Image img_gauss9 = mm::gaussian(img_gray, 9, 0);

    // Detalhe da região do olho
    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>{"Detalhe: Original", "Média 9×9", "Gaussiano 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] Detalhe: Original
[2] Média 9×9
[3] Gaussiano 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)")
```

Figura 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 Filtraggio Spaziale di Enfasi

I filtri di enfasi (*sharpening filters*) enfatizzano le transizioni brusche di intensità, aumentando la nitidezza e la visibilità dei bordi. Sono filtri **passa-alto** — amplificano le componenti ad alta frequenza (bordi, texture) e sopprimono quelle a bassa frequenza (regioni uniformi).

L'intuizione è semplice: se sottraiamo da un'immagine la sua versione attenuata (che contiene solo le basse frequenze), ciò che resta sono le alte frequenze — bordi e dettagli. Sommando questo residuo all'immagine originale, il contrasto locale aumenta:

$$g = f + k\,(f - f_{\text{suave}}), \quad k > 0 \tag{3.15}$$

La Figura 3.15 illustra questo processo su un segnale 1D sintetico con tre strutture distinte: un gradino largo, un picco sottile e una rampa dolce. Nel pannello , il segnale originale $f(x)$; nel , la versione attenuata $f_{\text{suave}}(x)$ ottenuta mediante media mobile — si noti come il picco sottile venga attenuato. Il pannello mostra il residuo $f - f_{\text{suave}}$, che conserva solo le transizioni brusche. Infine, il pannello mostra $g(x)$: il picco, prima attenuato, viene ripristinato e amplificato rispetto all'originale. Regolate k e la dimensione della finestra per osservare il *trade-off* tra nitidezza e amplificazione del rumore.

I filtri di enfasi formalizzano questa idea direttamente nel *kernel*, senza necessità di due fasi separate.

3.6.1 Laplaciano

Il Laplaciano è un operatore di **derivata seconda** isotropico, cioè risponde in modo uguale alle variazioni in tutte le direzioni, a differenza degli operatori di derivata prima, come Sobel e Prewitt, che sono direzionali:

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \tag{3.16}$$

Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.it/cap03/sim-03-filtragem1d.html

Figura 3.15: Simulatore: Filtraggio Spaziale di Enfasi 1D (Unsharp Masking e High-Boost)

Una proprietà importante della derivata seconda è che il suo valore è **prossimo allo zero nelle regioni uniformi** ed elevato nelle transizioni di intensità. Così, sottraendo il Laplaciano dall'immagine originale, si rafforzano bordi e dettagli, aumentando il contrasto locale:

$$g(x,y) = f(x,y) - \nabla^2 f(x,y) \tag{3.17}$$

Nella forma discreta, la derivata seconda in x è approssimata da $f(x+1,y) - 2f(x,y) + f(x-1,y)$, e analogamente in y. Sommando le due direzioni, si ottiene il *kernel* w_4 (4-vicini) o w_8 (8-vicini, includendo le diagonali):

$$w_4 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \qquad w_8 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{3.18}$$

i Somma zero e centro negativo

Entrambi i *kernel* hanno **somma dei coefficienti pari a zero**: nelle regioni uniformi, l'uscita è 0 — il Laplaciano non modifica la luminosità media, ma rileva solo le variazioni. Il **centro negativo** indica che il pixel viene confrontato con i suoi vicini: quanto più esso si distingue (verso l'alto o verso il basso), tanto maggiore è il valore assoluto del Laplaciano in quel punto.

Nell'esempio seguente, il pixel centrale $[1,1] = 107$ possiede vicini $\{95, 106, 108, 103\}$. Poiché questi valori sono tra loro prossimi, la regione è quasi uniforme e il Laplaciano restituisce un valore basso, producendo poco miglioramento. Nelle regioni di bordo, dove vi sono differenze maggiori tra il pixel centrale e i suoi vicini, il Laplaciano assume valori più elevati (positivi o negativi), e l'operazione di Equazione 3.17 intensifica queste transizioni.

La Figura 3.16 illustra il calcolo del Laplaciano con il *kernel* w_4 in un intorno 3×3 evidenziato all'interno di un *patch* 5×5. L'esempio mostra il valore ottenuto dall'operatore e il corrispondente pixel migliorato nell'immagine di uscita, che passa da 107 a 123.

```
%%writefile tmp/fig_03_laplaciano_patch.cpp
#define MM_OUT "tmp/fig_03_laplaciano_patch.png"
//| 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

#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]

    // As variáveis img_gray são pré-inicializadas aqui

    mm::Image patch = mm::crop(img_gray, 250, 255, 250, 255);
    mm::Kernel B{{1,1,1},{1,1,1},{1,1,1}};

    std::cout << "Patch 5x5 (intensidades):\n";
    std::cout << mm::drawImg(patch) << "\n";

    mm::drawImgKernel(patch, B, 1, 1, MM_OUT);
```

```
    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)")
```

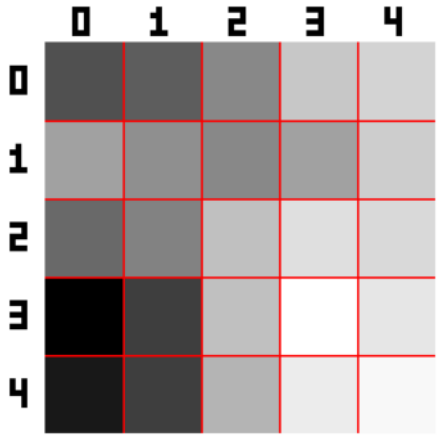


Figura 3.16: *Kernel* Laplaciano w4 sobre o *patch* 5×5: a janela amarela destaca a vizinhança 3×3 onde o operador de segunda derivada é calculado.

Figura 3.17 confronta l'applicazione dei *kernel* laplaciani w_4 e w_8 su un ritaglio più ampio dell'immagine del leopardo. Per ciascun caso, vengono mostrate la risposta grezza dell'operatore, che evidenzia i bordi e le transizioni di intensità, e l'immagine ottenuta dopo l'enfatizzazione mediante sottrazione del laplaciano. Si osserva che il *kernel* w_8, considerando anche i vicini diagonali, produce una risposta più intensa e rileva variazioni in più direzioni, risultando in un'enfatizzazione lievemente più marcata.

```
%%writefile tmp/fig_03_laplaciano.cpp
#define MM_OUT "tmp/fig_03_laplaciano.png"
//| label: fig-03-laplaciano
//| fig-cap: "Laplaciano applicato all'immagine del leopardo: risposta grezza (bordi) com w4 e
w8, e immagini migliorate mediante sottrazione del Laplaciano. w8 è più sensibile alle
diagonali."
//| echo: true
//| output: true

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

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]

    // Le variabili img_gray_crop sono già inizializzate qui

    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)")
```

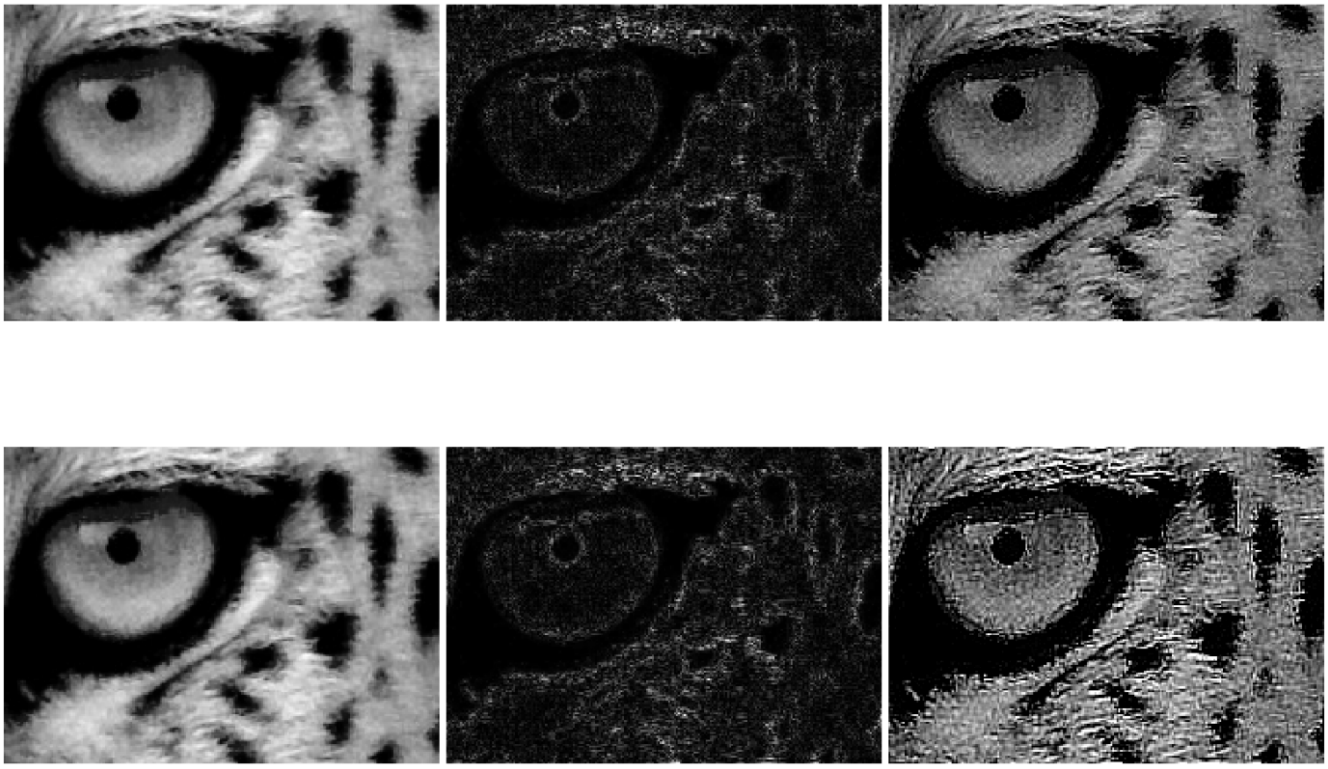


Figura 3.17: Laplaciano aplicado à imagem do leopardo: resposta bruta (bordas) com w4 e w8, e imagens realçadas pela subtração do Laplaciano. w8 é mais sensível às diagonais.

3.6.2 Operatore di Sobel

L'operatore di Sobel stima le **derivate parziali del primo ordine** nelle direzioni orizzontale e verticale. A differenza del Laplaciano (derivata seconda), il Sobel è direzionale e più robusto al rumore, poiché ogni *kernel* combina una derivata con un smoothing gaussiano perpendicolare:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * f, \qquad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} * f \tag{3.19}$$

G_x rileva i bordi **verticali** (variazione nella direzione x); G_y rileva i bordi **orizzontali** (variazione nella direzione y). I pesi $\{1, 2, 1\}$ nella direzione perpendicolare corrispondono allo smoothing gaussiano 1D, che riduce la sensibilità al rumore.

i Sobel è correlazione, non convoluzione

I *kernel* di Sobel sono **asimmetrici** — la rotazione di 180° modifica il risultato. `cv2.Sobel` implementa la correlazione incrociata (come `cv2.filter2D`). Per ottenere la derivata direzionale corretta, i segni sono già definiti per la correlazione: G_x restituisce valori positivi dove l'intensità cresce da sinistra a destra.

La magnitudine del **gradiente** combina le duas componenti, rappresentando la forza del bordo indipendentemente dalla direzione:

$$|\nabla f| = \sqrt{G_x^2 + G_y^2} \tag{3.20}$$

E la **direzione** del gradiente (perpendicolare al bordo) è:

$$\theta = \arctan\left(\frac{G_y}{G_x}\right) \tag{3.21}$$

Per illustrare numericamente, si calcolano G_x e G_y manualmente sul pixel centrale $[1, 1]$ del *patch* 5×5:

Il valore ridotto di $|\nabla f|$ in questo *patch* conferma che la regione è quasi uniforme, poiché il gradiente assume valori elevati solo dove vi sono cambiamenti significativi di intensità. La Figura 3.18 applica l'operatore di Sobel a un ritaglio più ampio dell'immagine del leopardo. Vengono presentate le risposte orizzontale (G_x) e verticale (G_y), ottenute per convoluzione con i rispettivi *kernel* di Sobel, oltre alla magnitudine $|\nabla f|$, calcolata dalla combinazione di entrambe. Mentre G_x evidenzia i bordi verticali e G_y quelli orizzontali, la magnitudine mette in risalto i bordi in qualsiasi direzione.

```
%%writefile tmp/fig_03_sobel.cpp
#define MM_OUT "tmp/fig_03_sobel.png"
#include "morph.hpp"
#include <iostream>
#include <vector>
#include <filesystem>

//| label: fig-03-sobel
//| fig-cap: "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."
//| 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)")
```

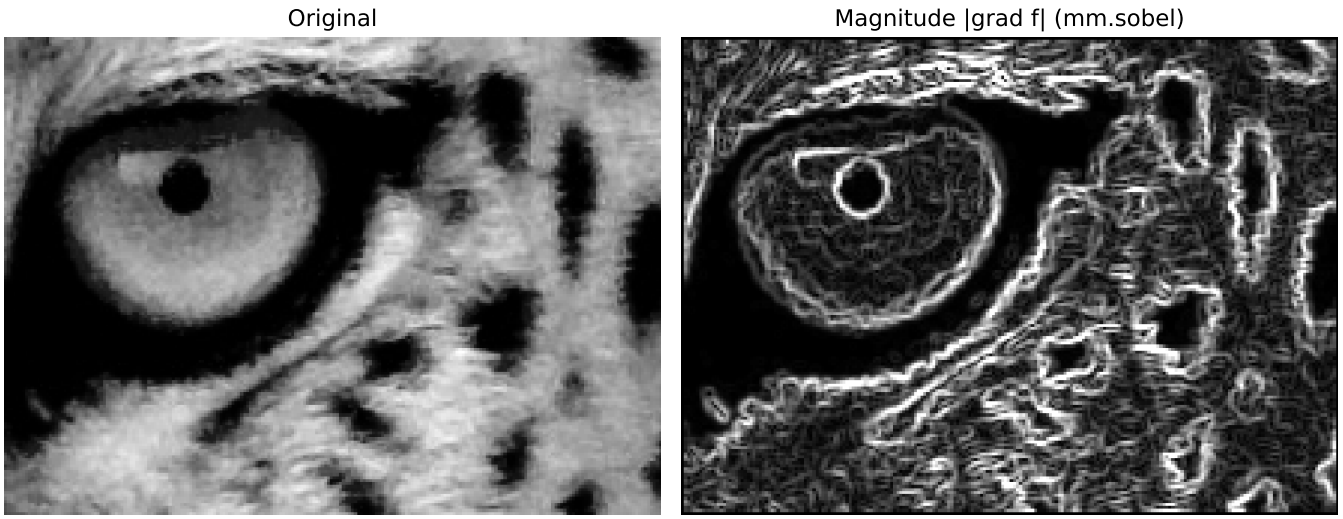


Figura 3.18: Operador de Sobel na imagem do leopardo: a magnitude | f| combina os gradientes horizontal e vertical, revelando todas as bordas. A decomposição Gx/Gy com sinal fica na trilha Python — mm::sobel devolve a magnitude já com clip.

3.6.3 Operatore di Prewitt

L'operatore di Prewitt è strutturalmente identico a Sobel, ma sostituisce la ponderazione gaussiana $\{1, 2, 1\}$ con pesi uniformi $\{1, 1, 1\}$:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} * f, \qquad G_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} * f \tag{3.22}$$

La magnitudine e la direzione del gradiente seguono le stesse equazioni di Sobel (Equazione 3.20 e Equazione 3.21). La differenza pratica è che Prewitt è leggermente più sensibile al rumore — l'appianamento perpendicolare uniforme pondera meno il pixel centrale della linea — ma è computazionalmente più semplice. In immagini a basso rumore i risultati sono equivalenti.

```
%%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>

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 è già disponibile come variabile valida
    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)")
```

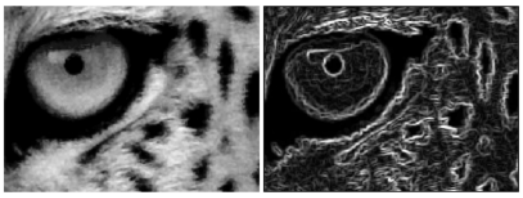


Figura 3.19: Operador de Prewitt: magnitude do gradiente, comparável ao Sobel mas sem a ponderação central. mm::prewitt devolve |f| com clip.

3.6.4 *Unsharp Masking* (USM)

L'*Unsharp Masking* è una tecnica classica di miglioramento della nitidezza, originaria della fotografia analogica, oggi ampiamente utilizzata nei software di editing delle immagini. L'idea centrale è estrarre le **componenti ad alta frequenza** dell'immagine (bordi e dettagli) e sommarle nuovamente all'originale con un peso k:

Tabella 3.3: Fasi dell'*Unsharp Masking*.

Fase	Operazione	Descrizione
1	$\bar{f} = f * G_\sigma$	Smussa con una gaussiana — mantiene le basse frequenze
2	$m = f - \bar{f}$	Maschera: differenza = alte frequenze (bordi)
3	$g = f + k \cdot m$	Somma ponderata della maschera all'originale

Sostituendo la fase 2 nella fase 3, si ottiene l'espressione compatta:

$$g = f + k\,(f - f * G_\sigma) = (1 + k)\,f - k\,(f * G_\sigma) \tag{3.23}$$

Il parametro k controlla l'intensità del miglioramento:

- $k = 0$: nessun miglioramento ($g = f$);
- $k = 1$: USM classico — raddoppia il contributo delle alte frequenze;
- $k > 1$: *High Boost Filtering* — amplificazione oltre il doppio, utile per immagini molto sfocate.

> ⚠ Amplificazione del rumore
>
> L'USM non distingue i bordi dal rumore — entrambi sono componenti ad alta frequenza. Per valori elevati di k, il rumore presente nell'immagine viene amplificato insieme ai bordi. Per questo motivo, è consigliabile applicare una leggera smussatura prima dell'USM su immagini rumorose, oppure utilizzare un σ piccolo nella gaussiana.

Per illustrare le fasi dell'USM, la Figura 3.20 applica il metodo a una *patch* 30×30 dell'immagine del leopardo, utilizzando $\sigma = 1$ e $k = 1$. Inizialmente, l'immagine viene smussata da un filtro gaussiano. Successivamente, la maschera ad alta frequenza si ottiene dalla differenza tra l'immagine originale e quella smussata. Infine, tale maschera viene sommata all'immagine originale, rafforzando bordi e dettagli. La figura presenta le tre fasi del processo e il resultado finale dell'enhancement.

```
%%writefile tmp/fig_03_usm2.cpp
#define MM_OUT "tmp/fig_03_usm2.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-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

    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)")
```

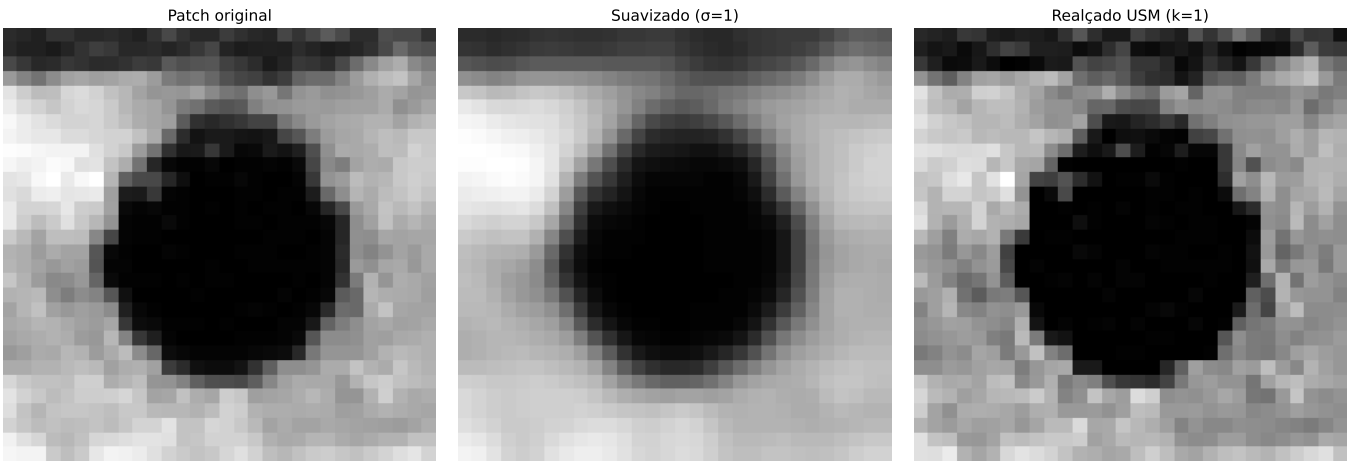


Figura 3.20: Realce por *Unsharp Masking* num *patch* do leopardo: mm::usm faz suavização Gaussiana, subtrai da original (máscara de alta frequência) e reintroduz a máscara realçada.

La Figura 3.21 applica il metodo USM a un ritaglio più ampio dell'immagine del leopardo utilizzando $\sigma = 1$ e diversi valori del fattore di guadagno k. In tutti i casi, la maschera ad alta frequenza è ottenuta dalla differenza tra l'immagine originale e la sua versione smussata tramite filtro gaussiano. Il parametro k controlla l'intensità del miglioramento: valori più piccoli producono un aumento sottile della nitidezza, mentre valori più grandi rafforzano progressivamente bordi e dettagli. Si osserva che, per valori elevati di k, compaiono aloni attorno ai bordi e il rumore presente nell'immagine viene amplificato.

```
%%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>

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 is pre-initialized

    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)")
```

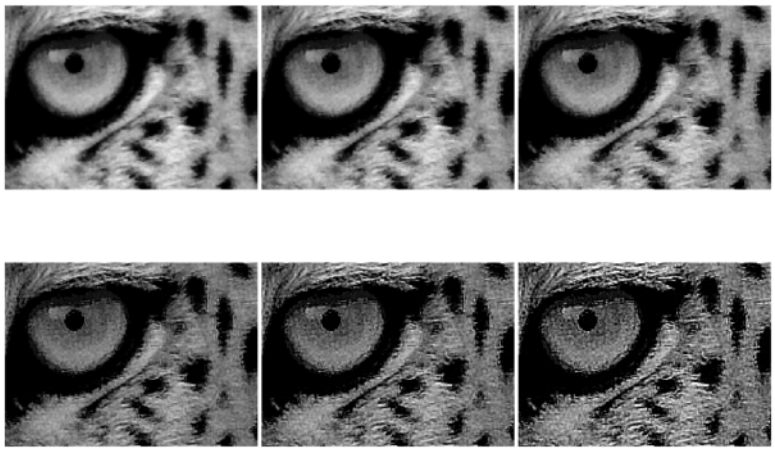


Figura 3.21: *Unsharp Masking* na imagem do leopardo com =1 e k de 0.5 a 8.0. Para k>2 surgem halos nas bordas e o ruído de fundo aparece.

3.6.5 Rilevatore di Canny

Il Canny combina quattro fasi in sequenza — smoothing gaussiano, gradiente di Sobel, soppressione dei non-massimi e isteresi con doppia soglia — per produrre bordi **sottili, binari e connessi**. A differenza di Sobel e Prewitt, il risultato non è una mappa di gradiente continua, ma una maschera in cui ogni pixel è bordi o non lo è.

Il parametro centrale è la coppia di soglie (T_{low}, T_{high}). I pixel con gradiente superiore a T_{high} sono bordi certi; al di sotto di T_{low}, vengono scartati. I pixel ambigui — tra le due soglie — vengono decisi mediante **isteresi**: diventano bordi se sono connessi a un bordo certo, e vengono scartati in caso contrario. Questo evita sia la perdita di tratti deboli di bordi reali, sia l'inclusione di rumore isolato. Un'euristica comune è $T_{high} = 3 \times T_{low}$.

```
%%writefile tmp/fig_03_canny.cpp
#define MM_OUT "tmp/fig_03_canny.png"
//| label: fig-03-canny
```

```
//| fig-cap: "Detector de Canny com diferentes pares de limiar: limiares baixos capturam mais
bordas (inclusive ruído); limiares altos retêm apenas as bordas mais fortes."
//| 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]

    // img_gray_crop is provided as a valid mm::Image variable

    mm::show(
        std::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,
        std::vector<std::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)")
```

Figura 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.

Scelta delle soglie

Un'euristica comune è $T_{high} = 3 \times T_{low}$. I valori tipici dipendono dall'intervallo di gradiente dell'immagine — `cv2.Canny` accetta valori assoluti in $[0, 255]$. Per immagini con contrasto variabile, calcolare le soglie a partire dai percentili della magnitudine di Sobel è più robusto rispetto a valori fissi.

3.7 Filtri d'Ordine: Filtro Mediano

I filtri d'ordine (*order-statistic filters*) sostituiscono il pixel centrale con il valore di un **percentile** della distribuzione delle intensità del vicinato — a differenza dei filtri lineari, che calcolano combinazioni ponderate. Il più importante è il **filtro mediano**.

3.7.1 Rumore Impulsivo: Sale e Pepe

Il rumore **sale e pepe** (*salt-and-pepper noise*) sostituisce pixel casuali con valori estremi: 0 (pepe, nero) o 255 (sale, bianco). È comune nella trasmissione di immagini con errori di bit e in fotocamere con sensori difettosi.

Per capire perché i filtri lineari falliscono, si consideri un vicinato 3×3 in cui un singolo pixel è stato corrotto a 255:

$$\text{vizinhança} = \begin{bmatrix} 102 & 98 & 105 \\ 100 & \mathbf{255} & 97 \\ 103 & 99 & 101 \end{bmatrix}$$

Tabella 3.4: Media vs. mediana con un pixel corrotto. La mediana ignora il valore anomalo; la media è spostata di ~40 livelli.

Metodo	Calcolo	Risultato
Media	(102+98+…+255+…+101)/9	140
Mediana	{97,98,99,100,**101**,102,103,105,255}	101

Perché i filtri di media falliscono con il rumore impulsivo?

La media è sensibile ai ***valori anomali*** — un singolo pixel con valore 255 in un vicinato di valore 100 innalza l'uscita a 140, diffondendo il rumore nell'immagine. La mediana, essendo uno **stimatore robusto**, seleziona il valore centrale della distribuzione ordinata, scartando naturalmente gli estremi senza alcun aggiustamento speciale.

L'esempio seguente illustra il comportamento della media e della mediana in presenza de um pixel corrotto da rumore impulsivo. Si osserva che la media è fortemente influenzata dal valore estremo (255), producendo una stima lontana dai valori predominanti del vicinato. La mediana, invece, rimane vicina al valore originale della regione, evidenziando la sua maggiore robustezza ai *valori anomali* e giustificando il suo utilizzo nella rimozione del rumore sale e pepe.

La Figura 3.23 presenta l'effetto del rumore sale e pepe a diverse densità. Il rumore è stato generato sostituendo casualmente una frazione dei pixel con valori minimi (0, pepe) e massimi (255, sale). All'aumentare della densità dal 2% al 10%, cresce la quantità di pixel corrotti, rendendo la degradazione visiva più evidente e ostacolando la percezione dei dettagli dell'immagine.

```
%%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;  // cópia
    int h = out.h, 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_rand(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_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 aqui

    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({img_gray_crop, n2, n5, n10},
             MM_OUT,
             {"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)")
```

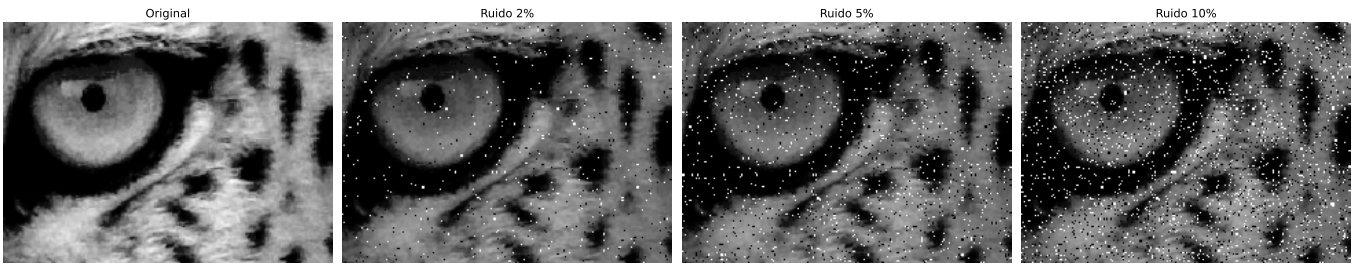


Figura 3.23: Ruído sal e pimenta com densidades crescentes (2%, 5%, 10%): metade dos pixels corrompidos vira sal (255), metade pimenta (0).

3.7.2 Filtro da Mediana

Il filtro della mediana sostituisce ogni pixel con il **valore mediano** dei pixel del suo intorno $n \times n$:

$$g(x,y) = \operatorname{med}_{(s,t)\in\mathcal{V}_n}\{f(x+s, y+t)\} \tag{3.24}$$

Il valore mediano è quello che occupa la posizione centrale quando gli n^2 valori dell'intorno vengono ordinati. Per una finestra 3×3 ($n^2 = 9$ pixel), la mediana è il 5° valore della sequenza ordinata.

Per illustrare, si consideri lo stesso *patch* 5×5 con un pixel corrotto artificialmente in $[1, 1]$:

L'esempio conferma: anche con il pixel corrotto a 255, la mediana restituisce il valore centrale corretto — il *valore anomalo* occupa l'ultima posizione nell'ordinamento e viene scartato naturalmente.

Poiché si basa sull'ordinamento e non sulla somma, la mediana possiede tre proprietà fondamentali che la differenziano dai filtri lineari:

- **Robusta** al rumore impulsivo — i valori anomali finiscono alle estremità della sequenza ordinata e non influenzano il valore centrale;
- **Preservatrice dei bordi** — le transizioni brusche di intensità vengono mantenute, poiché la mediana seleziona un valore che esiste già nell'intorno, senza creare nuovi livelli intermedi;
- **Non lineare** — non può essere espressa come convoluzione, quindi `mm::conv` non si applica; si usa `cv2.medianBlur`.

Figura 3.24 confronta diverse tecniche di rimozione del rumore sale e pepe applicate a un'immagine con il 10% dei pixel corrotti. Sono stati valutati i filtri Gaussiano, Media, Mediana, Bilaterale e Morfologico (prossimo capitolo), consentendo di osservare il compromesso tra rimozione del rumore e preservazione dei dettagli. In generale, i filtri di media e Gaussiano riducono il rumore, ma tendono a sfocare i bordi, mentre la mediana presenta prestazioni migliori per il rumore impulsivo. Il filtro bilaterale preserva meglio i bordi, e il filtro morfologico rimuove gran parte dei pixel corrotti senza degradare eccessivamente la struttura dell'immagine.

```
%%writefile tmp/fig_03_ruido_filtros.cpp
#define MM_OUT "tmp/fig_03_ruido_filtros.png"
#include <iostream>
#include <random>
#include "morph.hpp"
#include <filesystem>

//| 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

mm::Image salt_pepper(const mm::Image& img, double prob) {
    mm::Image out = img;
    int h = out.h, 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 noisy = salt_pepper(img_gray_crop, 0.10);

    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);

    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)")
```

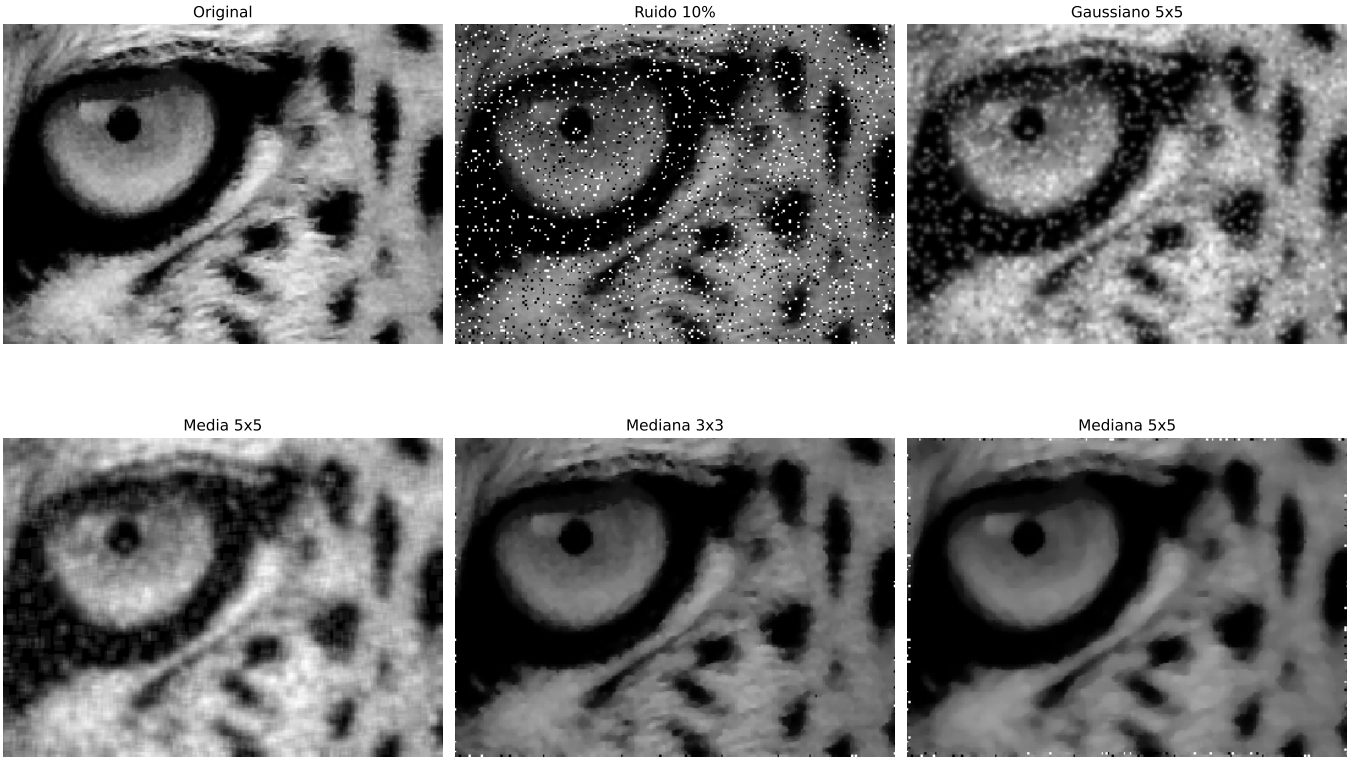


Figura 3.24: Filtros para ruído sal e pimenta (10%): Gaussiano, Média e Mediana. Bilateral e morfológico (open+close) ficam só na trilha Python — bilateral não tem equivalente em morph.hpp e morfologia é do próximo capítulo.

3.8 Applicazione Pratica: Pre-elaborazione per la Segmentazione

Nella pratica, le tecniche di questo capitolo raramente vengono utilizzate singolarmente. Una ***pipeline* di pre-elaborazione** tipica combina più fasi in sequenza, adattandosi al tipo di immagine e all'applicazione. La Figura 3.25 illustra una *pipeline* completa:

1. **Equalizzazione dell'istogramma (CLAHE):** normalizza il contrasto indipendentemente dalle condizioni di illuminazione;
2. **Filtro Gaussiano:** attenua il rumore di acquisizione senza distruggere i bordi;
3. **Rilevamento dei bordi (Sobel/Canny):** estrae strutture rilevanti per la segmentazione.

> **i** L'ordine è importante
>
> L'ordine delle operazioni influisce sul risultato finale. In generale: **(1) normalizzazione dell'intensità → (2) riduzione del rumore → (3) miglioramento/segmentazione**. Invertire l'ordine può amplificare il rumore o perdere i bordi prima di rilevarli.

```
%%writefile tmp/fig_03_pipeline.cpp
#define MM_OUT "tmp/fig_03_pipeline.png"
//| label: fig-03-pipeline
//| fig-cap: "*Pipeline* di pre-elaborazione: equalizzazione → Gaussiano → Canny. Il percorso
Python usa CLAHE al posto dell'equalizzazione globale; CLAHE non ha equivalente in morph.hpp."
```

```
//| 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]

    // img_gray_crop è già disponibile (assegnato automaticamente)

    mm::Image img_eq    = mm::equalize(img_gray_crop);     // Fase 1: equalizzazione globale
    mm::Image img_gauss = mm::gaussian(img_eq, 5, 0);      // Fase 2: Gaussiano
    mm::Image edges     = mm::canny(img_gauss, 50, 150);   // Fase 3: Canny

    mm::Image edges_direct = mm::canny(img_gray_crop, 50, 150);   // Canny diretto, senza
pre-elaborazione

    mm::show(
        std::vector<mm::Image>{img_gray_crop, img_eq, img_gauss, edges, edges_direct},
        MM_OUT,
        std::vector<std::string>{"Originale", "1. Equalizzato", "2. Gaussiano", "3. Canny
(pipeline)", "Canny (diretto)"},
        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] Originale
[2] 1. Equalizzato
[3] 2. Gaussiano
[4] 3. Canny (pipeline)
[5] Canny (diretto)
```

```
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)")
```

Figura 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 Riassunto

In questo capitolo sono state presentate le principali tecniche di elaborazione nel dominio spaziale, dalla manipolazione diretta dei pixel fino al filtraggio per vicinato:

- **Operazioni puntuali:** aritmetiche sature (`mm::addm`, `mm::subm`) e logiche bit a bit (`mm::band`, `mm::bor`, `mm::bnot`) per il ritaglio di ROI e la combinazione di immagini; *alpha blending* (`mm::blend`) per la fusione pesata con peso $\alpha \in [0, 1]$.
- **Istogramma:** funzione discreta di distribuzione delle intensità; visualizzato con `mm::histImg` e calcolato con `mm::hist`; base per la diagnosi tonale e per le tecniche di equalizzazione e specificazione.
- **Equalizzazione:** redistribuzione automatica delle intensità tramite la CDF (`mm::equalize`), con variante adattativa CLAHE per il controllo locale del contrasto.
- **Specificazione dell'istogramma:** trasferimento del profilo tonale di un'immagine di riferimento mediante mappatura inversa della CDF — generalizzazione dell'equalizzazione per distribuzioni arbitrarie.
- **Correlazione e convoluzione:** meccanismo di finestra scorrevole implementato in `mm::conv` (`cv2.filter2D`); differenziati dalla rotazione di 180° del *kernel* — rilevante solo per kernel asimmetrici.
- **Filtri di smoothing:** media (*kernel* uniforme, sfuma i bordi proporzionalmente alla dimensione) e Gaussiano (ponderazione radiale, separabile, senza *ringing*, preserva meglio i bordi).
- **Filtri di enhancement:** Laplaciano (w_4/w_8, seconda derivata isotropica), Sobel (gradiente direzionale del primo ordine, con magnitudine $|\nabla f|$ e direzione θ) e *Unsharp Masking* (amplificazione delle alte frequenze con parametro k).
- **Filtro mediano:** non lineare, robusto agli outlier, preserva i bordi — superiore ai filtri lineari per il rumore sale e pepe.
- **Pipeline pratica:** concatenazione CLAHE → Gaussiano → Canny come strategia di pre-elaborazione; `mm::drawImgKernel` per la visualizzazione didattica della finestra scorrevole.

Il Capitolo 4 tratterà la **morfologia matematica** (erosione, dilatazione, apertura e chiusura), esplorando in

profondità le funzioni `mm::ero` e `mm::dil` della libreria `morph.py`. Successivamente, il Capitolo 5 presenterà l'**elaborazione nel dominio della frequenza**, con focus sulla Trasformata di Fourier e sulle tecniche di filtraggio spettrale.

3.10 🤖 Uso di Gemini Notebook come Tutor Complementare

In questa edizione, incoraggiamo l'uso di **Gemini Notebook** come strumento complementare di apprendimento. Questo strumento di IA utilizza esclusivamente i documenti forniti dall'autore come base di conoscenza, garantendo risposte coerenti con il contenuto del libro — incluse le funzioni della libreria `morph.py` e gli esperimenti realizzati in questo capitolo.

Per ogni capitolo, abbiamo preparato un progetto specifico sulla piattaforma con il PDF del capitolo, i *notebook* e i materiali ausiliari. Suggeriamo di esplorare in particolare:

- **Guida allo Studio:** riassunto strutturato dei concetti, ideale per il ripasso prima delle verifiche;
- **Conversazione:** chiarisci dubbi su equalizzazione, convoluzione, filtri e pipeline direttamente con il tutor;
- **Domande frequenti:** quesiti tipici sulla differenza tra media e mediana, USM, Laplaciano vs. Sobel.

🎓 Studia con il Tutor Intelligente

Per interagire con il contenuto di questo capitolo, accedi al *link* seguente. L'ambiente contiene materiali didattici in diversi formati, generati dal **PDF** del capitolo. Sulla piattaforma, esplora in particolare le opzioni **Guida allo Studio** e **Conversazione** per approfondire la tua comprensione.
🚀 ACCEDI A Gemini Notebook: CAPITOLO 03

🌐 Lingua e Linguaggio di Programmazione

Il progetto di questo capitolo su Gemini Notebook è stato costruito solo con il testo in **portoghese** e gli esempi di codice in **Python**. Se stai studiando dall'edizione in inglese o francese, o seguendo il percorso in C++, le risposte del tutor potrebbero non corrispondere esattamente alla versione che stai leggendo.

⚠️ Avviso sui Contenuti Generati dall'IA

L'IA è una potente alleata nello studio, ma il contenuto generato può contenere **errori o imprecisioni**. Consulta sempre **libri, articoli scientifici e altre fonti accademiche affidabili** per validare le informazioni. Quando possibile, esegui gli esempi pratici forniti in questo capitolo per verificare i risultati.

3.11 Lista di Esercizi

1. **(10%)** Spiegare la differenza tra **convoluzione** e **correlazione incrociata**. Per quali tipi di *kernel* i risultati sono identici? Fornire un esempio di *kernel* asimmetrico (come Sobel G_x) e mostrare numericamente che i risultati differiscono applicandolo al patch 5×5 del capitolo in entrambi i modi.

2. **(15%)** Considerare un'immagine 5×5 con intensità concentrate tra i livelli 3 e 5 (basso contrasto, 3 bit). Applicare manualmente l'algoritmo di equalizzazione della Tabella 3.1, completando tutte le colonne della tabella (k, $h[k]$, $p[k]$, cdf$[k]$, lut$[k]$). Verificare il risultato con `mm::equalize`.

3. **(15%)** Utilizzando `mm::conv`, applicare il filtro media con kernel di dimensione 3×3, 9×9 e 21×21 all'immagine del mandrillo. Per ciascuna versione, calcolare il **PSNR** (*Peak Signal-to-Noise Ratio*) rispetto all'originale:

$$\text{PSNR} = 10\log_{10}\left(\frac{255^2}{\text{MSE}}\right), \quad \text{MSE} = \frac{1}{MN}\sum_{i,j}(f-g)^2$$

Tracciare il PSNR in funzione della dimensione del kernel e spiegare cosa indica la diminuzione progressiva riguardo alla relazione tra smoothing e perdita di informazione.

4. **(15%)** Utilizzando `add_salt_pepper` con densità del 5%, applicare e confrontare: (a) `mm::conv` con media 3×3, (b) `cv2.GaussianBlur` con $\sigma = 1$, (c) `cv2.medianBlur` con finestra 3×3 e (d) `cv2.medianBlur` con finestra 5×5. Visualizzare le immagini con `mm::show` in una griglia 2×4 (riga 1: immagini, riga 2: istogrammi tramite `mm::histImg`). Spiegare perché la mediana supera i filtri lineari utilizzando l'argomento della Tabella 3.4..

5. **(15%)** Implementare `mm::conv0` utilizzando solo operazioni NumPy vettorizzate — senza cicli Python e senza `cv2.filter2D` — con l'operatore di *stride tricks* (`np.lib.stride_tricks.sliding_window_view`). Confrontare il risultato e il tempo di esecuzione con `mm::conv0` (cicli) e `mm::conv` (cv2) per kernel 3×3 e 15×15 sull'immagine del mandrillo.

6. **(15%)** Applicare l'*Unsharp Masking* con $\sigma = 1$ e $k \in \{0.5, 1.0, 2.0, 4.0\}$ utilizzando la funzione `usm` del capitolo. Per ciascun valore di k: (a) calcolare la differenza assoluta $|g - f|$, (b) visualizzare le immagini e le differenze con `mm::show`, e (c) tracciare l'istogramma delle differenze con `mm::histImg`. Identificare a partire da quale k gli artefatti (aloni e amplificazione del rumore) diventano visivamente inaccettabili.

7. **(15%)** Scegliere un'immagine radiografica o tomografica disponibile pubblicamente (es.: tramite `mm::read` da URL) e progettare una pipeline di pre-elaborazione con almeno 4 fasi sequenziali, giustificando ogni scelta sulla base dei concetti del capitolo. Visualizzare con `mm::show` in griglia: l'immagine originale, ogni fase intermedia e il risultato finale con i rispettivi istogrammi (`mm::histImg`).

Riferimenti del Capitolo

La base teorica di questo capitolo si fonda sulle seguenti opere:

- Gonzalez (2018) per i concetti di operazioni di intensità, istogramma, convoluzione e filtraggio spaziale.
- Szeliski (2022) per la visione computazionale e le applicazioni pratiche del filtraggio.
- Bradski (2008) per l'implementazione pratica con OpenCV e `morph.py`.

3.12 💻 Parte pratica con esercizi di programmazione

🎯 Obiettivo di questo Quaderno

Il quaderno consente di sviluppare, validare, organizzare e testare soluzioni di **Esercizi di Programmazione (EP)** in ambienti interattivi, come Colab, con gli stessi casi di test di Moodle, copiandoli lì solo al momento di registrare il voto ufficiale.

Download

Scarica `morph.py` e `testsuite.py` eseguendo la cella qui sotto:

```
import os, urllib.request

os.makedirs("tmp/state", exist_ok=True)  # artefatti di build della pista C++ (.cpp, binario,
PNG)

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")
```

```
# Il kernel è comunque Python anche nella pista C++: `mm` (morph.py) è usato dai
# simulatori, dalla visualizzazione delle figure che il binario C++ genera e dallo
# stato mm::Image tra le celle. cpp=True scarica anche la pista compilata
# (morph.hpp + stb_image*.h), usata nell'#include delle celle %%writefile *.cpp.
import config
config.setup(testsuite=True, cpp=True)
from morph import mm
from testsuite import TestSuite
```

```
Ambiente pronto. Morph: 1.1.9 | OpenCV: 5.0.0 | TestSuite: 1.1.2
```

Esecuzione dei Test

Per valutare i test, eseguire `TestSuite("EP03_01.extensão").run()` in una nuova cella, sostituendo l'estensione con quella del linguaggio utilizzato (`.py`, `.java`, `.c`, `.cpp`, `.js` o `.r`). Il sistema scarica i casi di test da GitHub, esegue il programma e calcola automaticamente il voto.

Per testare direttamente codice Python, senza salvare file, utilizzare `run_code(codigo)` passando il codice come *stringa* in una variabile `codigo`:

```
codigo = """
from morph import mm
# ... il tuo codice qui ...
"""
TestSuite("EP03_01").run_code(codigo)
```

3.12.1 EP03_01 ➕ Addizione Saturata di una Costante

Nei sistemi di videosorveglianza, le telecamere in ambienti con illuminazione variabile producono immagini sottoesposte. La regolazione della luminosità mediante **addizione saturata di una costante** è l'operazione più semplice per una correzione immediata, applicata in tempo reale sui *chip* delle telecamere integrate e nelle *pipeline* di pre-elaborazione dei robot mobili.

Vedi in Figura 3.26 una simulazione di questo EP.

3.12.1.1 📋 Linee Guida per l'Implementazione

1. **Dimensioni:** Leggere gli interi L (righe) e C (colonne).
2. **Costante:** Leggere l'intero k (valore da sommare).
3. **Dati:** Leggere i valori interi della matrice originale riga per riga.
4. **Mappatura:** Per ogni pixel p, calcolare il nuovo valore tramite l'equazione:

$$p' = \text{clip}(p + k)$$

5. **Output:** Visualizzare la matrice risultante con dimensioni $L \times C$.

3.12.1.2 📌 Vincoli Computazionali

- **Saturazione (*Clipping*):** I valori devono essere confinati nell'intervallo $[0, 255]$:

$$\text{clip}(x) = \max(0, \min(255, x))$$

- **Tipo:** Il risultato finale deve essere intero (senza cifre decimali).
- k **può essere negativo:** i valori negativi scuriscono l'immagine; quelli positivi la schiariscono.

3.12.1.3 🧠 Fondamento Teorico

Parametro	Tipo	Impatto Visivo
$k > 0$	Intero	Schiarisce l'immagine; i pixel vicini a 255 saturano in bianco
$k < 0$	Intero	Scurisce l'immagine; i pixel vicini a 0 saturano in nero
$k = 0$	Intero	Immagine invariata

3.12.1.4 📦 Specifica di Input e Output (VPL)

Input:

- Riga 1: Intero L.
- Riga 2: Intero C.
- Riga 3: Intero k.
- Righe successive: Elementi interi della matrice originale.

Output:

- Matrice trasformata in L righe e C colonne, valori interi separati da spazi.

3.12.1.5 📌 Esempi

Input	Output	Osservazione
2 3 50 0 100 200 210 240 255	50 150 250 255 255 255	Saturazione a 255 nei pixel alti
1 4 -30 0 20 200 255	0 0 170 225	Saturazione a 0 nei pixel bassi

```
%%writefile EP03_01.cpp
// your solution
```

```
Overwriting EP03_01.cpp
```

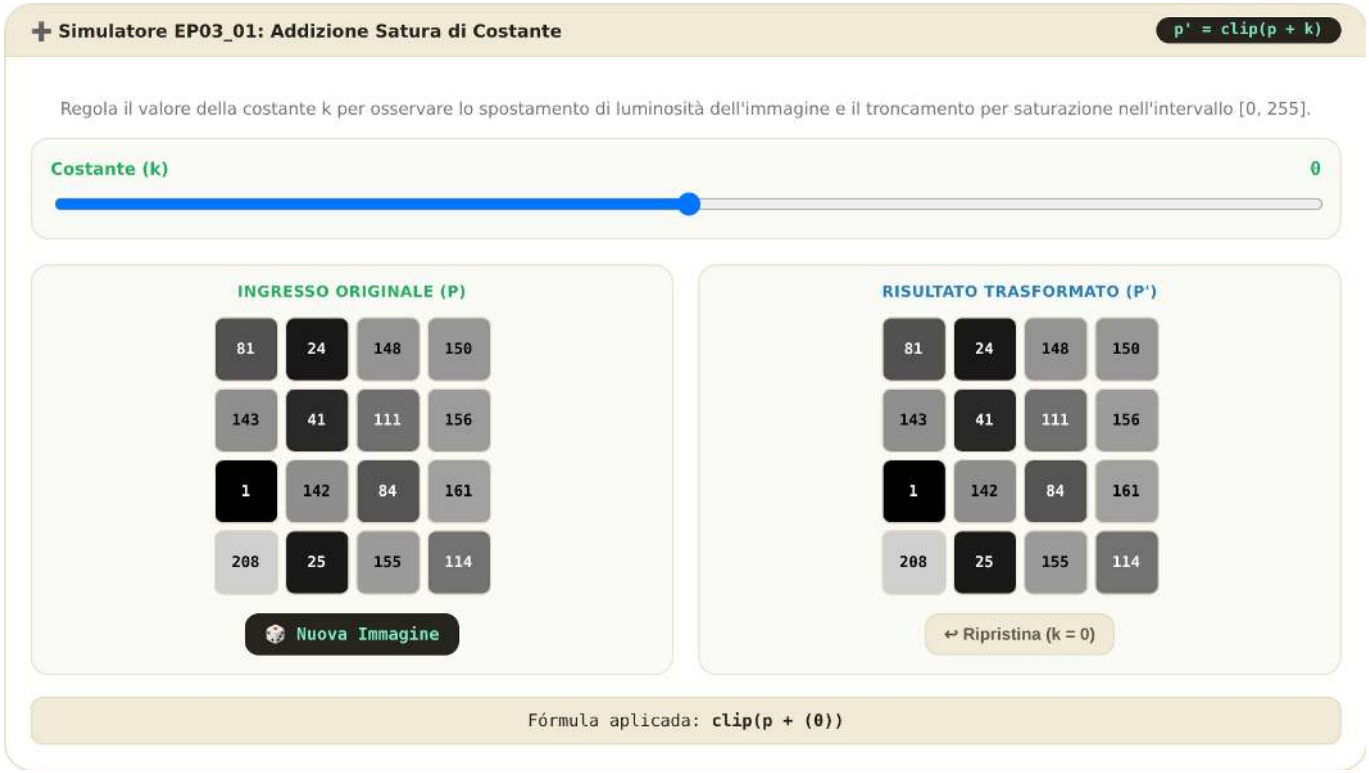
```
TestSuite("EP03_01.cpp").run()
```

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

3.12.2 EP03_02 🔀 Alpha *Blending* di Due Immagini

In medicina nucleare, immagini di diverse modalità (tomografia computerizzata e risonanza magnetica) vengono fuse per supportare la diagnosi. La **miscela ponderata** (*alpha blending*) è l'operazione fondamentale di questo processo, consentendo al radiologo di controllare interattivamente il peso di ciascuna modalità nell'immagine visualizzata.

Vedi in Figura 3.27 una simulazione di questo EP.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.it/cap03/sim-ep0301-adicao.html

Figura 3.26: Simulatore EP03_01: Addizione Saturata di Costante (p' = clip(p + k))

3.12.2.1 📋 Linee Guida di Implementazione

1. **Dimensioni:** Leggere gli interi L (righe) e C (colonne).
2. **Parametro:** Leggere il valore reale $\alpha \in [0, 1]$.
3. **Dati:** Leggere i valori interi della matrice f_1 (immagine 1) e successivamente della matrice f_2 (immagine 2).
4. **Mappatura:** Per ogni posizione (i, j), calcolare:

$$g(i,j) = \text{clip}\left(\text{round}\left(\alpha \cdot f_1(i,j) + (1-\alpha) \cdot f_2(i,j)\right)\right)$$

5. **Uscita:** Visualizzare la matrice risultante $L \times C$.

3.12.2.2 📌 Vincoli Computazionali

- **Arrotondamento:** Applicare `round` prima della conversione a intero.
- **Saturazione:** Confinare all'intervallo $[0, 255]$ con $\text{clip}(x) = \max(0, \min(255, x))$.
- **Operazione in virgola mobile:** Eseguire l'operazione in virgola mobile prima di arrotondare.

3.12.2.3 🧠 Fondamenti Teorici

Valore di α	Risultato
$\alpha = 1.0$	Solo f_1
$\alpha = 0.5$	Media aritmetica di f_1 e f_2
$\alpha = 0.0$	Solo f_2

3.12.2.4 📦 Specifica di Input e Output (VPL)

Input:

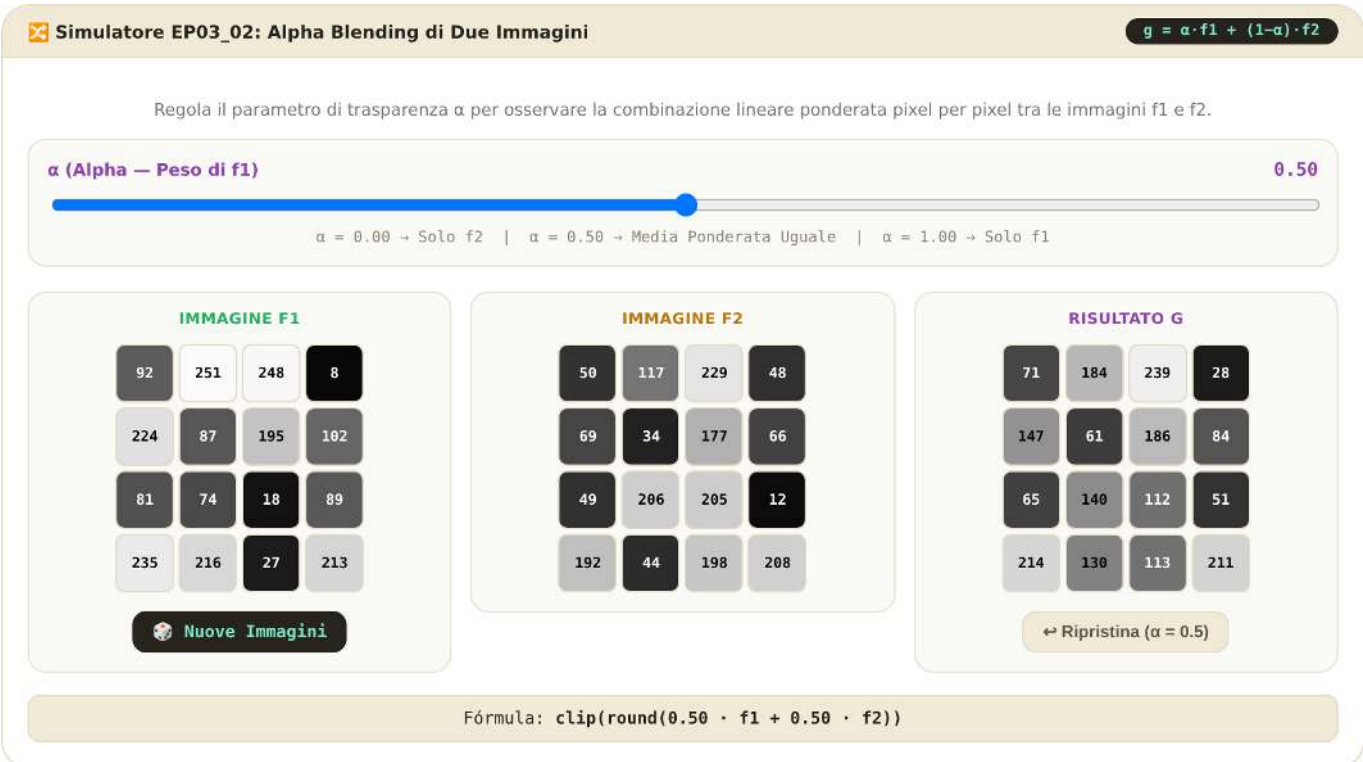
- Riga 1: Intero L.
- Riga 2: Intero C.
- Riga 3: Reale α.
- Righe successive: Elementi di f_1 (L righe con C valori ciascuna).
- Righe successive: Elementi di f_2 (L righe con C valori ciascuna).

Output:

- Matrice risultante $L \times C$.

3.12.2.5 📌 Esempi

Input	Output	Osservazione
1 3 0.5 0 100 200 100 200 50	50 150 125	Media tra le due immagini
1 3 1.0 10 20 30 90 80 70	10 20 30	Solo f_1 (alpha=1)



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.it/cap03/sim-ep0302-blending.html

Figura 3.27: Simulatore EP03_02: Alpha Blending di Due Immagini (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 🎭 Inversione dell'Immagine (Negativo Fotografico)

In radiologia, le immagini ai raggi X sono tradizionalmente visualizzate in negativo: le ossa appaiono in nero su sfondo bianco. L'operazione di **negativo fotografico** viene applicata di routine nei PACS (*Picture Archiving and Communication Systems*) per facilitare la rilevazione di fratture e densità ossee.

Vedere nella Figura 3.28 una simulazione di questo EP.

3.12.3.1 📋 Linee Guida di Implementazione

1. **Dimensioni:** Leggere gli interi L (righe) e C (colonne).
2. **Dati:** Leggere i valori interi della matrice originale.
3. **Mappatura:** Per ogni pixel p, calcolare il negativo:

$$p' = 255 - p$$

4. **Output:** Visualizzare la matrice risultante $L \times C$.

3.12.3.2 📌 Vincoli Computazionali

- **Nessun *clipping* necessario:** Il risultato di $255 - p$ con $p \in [0, 255]$ è sempre $\in [0, 255]$.
- **Tipo intero:** L'output deve essere composto da valori interi.
- **Equivalenza logica:** L'operazione è identica al `NOT` bit a bit (`mm::bnot`) su immagini a 8 bit.

3.12.3.3 🧠 Fondamenti Teorici

Pixel Originale p	Pixel Negativo p'	Osservazione
0 (nero)	255 (bianco)	Inversione totale
128 (grigio medio)	127 (grigio medio)	Valore centrale
255 (bianco)	0 (nero)	Inversione totale

3.12.3.4 📦 Specifica di Input e Output (VPL)

Input:

- Riga 1: Intero L.
- Riga 2: Intero C.
- Righe successive: Elementi interi della matrice originale.

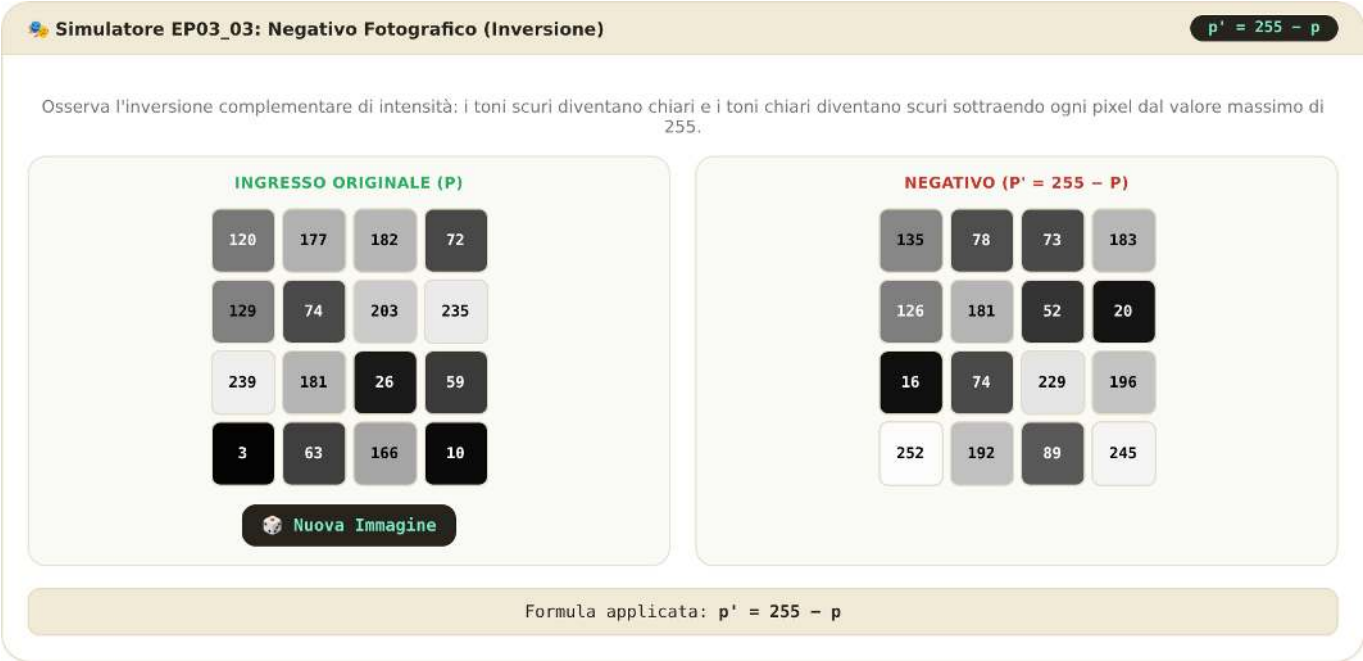
Output:

- Matrice negativa in L righe e C colonne.

3.12.3.5 📌 Esempi

Input	Output	Osservazione
1 4 0 128 200 255	255 127 55 0	Inversione di ogni pixel

Input	Output	Osservazione
2 2 10 20 30 40	245 235 225 215	Matrice 2x2 invertita



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.it/cap03/sim-ep0303-inversao.html

Figura 3.28: Simulatore EP03_03: Inversione dell'immagine — Negativo fotografico (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 📊 Equalizzazione dell'istogramma (L bit)

Nelle immagini satellitari di telerilevamento, la variazione dell'illuminazione durante il giorno produce immagini a basso contrasto. L'**equalizzazione dell'istogramma** viene applicata automaticamente su satelliti come il Landsat per redistribuire i toni, rivelando dettagli di vegetazione, rilievo e aree urbane invisibili nell'immagine originale.

Vedere in Figura 3.29 una simulazione di questo EP.

3.12.4.1 📋 Linee guida di implementazione

1. **Dimensioni:** Leggere gli interi L (righe), C (colonne) e B (numero di bit, con $L_{\max} = 2^B$).
2. **Dati:** Leggere la matrice di pixel f con valori in $[0, 2^B - 1]$.
3. **Istogramma:** Calcolare $h[k]$ = numero di pixel con intensità k, per $k = 0 \ldots 2^B - 1$.
4. **Probabilità:** $p[k] = h[k]/(L \cdot C)$.
5. **CDF:** $\text{cdf}[k] = \sum_{j=0}^{k} p[j]$; funzione di distribuzione cumulativa.

6. **LUT:** $\text{lut}[k] = \text{round}\,(\text{cdf}[k] \cdot (2^B - 1))$; *Look-Up Table* (tabella di consultazione).
7. **Applicazione:** $g[i, j] = \text{lut}[f[i, j]]$.
8. **Uscita:** Visualizzare la matrice equalizzata $L \times C$.

3.12.4.2 📌 Vincoli computazionais

- **Arrotondamento:** Usare l'arrotondamento matematico (`round`) nella LUT.
- **Bit:** Il numero di livelli è 2^B (es.: $B = 3 \Rightarrow 8$ livelli, $B = 8 \Rightarrow 256$ livelli).
- **CDF cumulativa:** $\text{cdf}[k] = \sum_{j=0}^{k} p[j]$, con $\text{cdf}[2^B - 1] = 1.0$.

3.12.4.3 🧠 Fondamenti teorici

Fase	Operazione	Formula
1	Istogramma	$h[k] \leftarrow$ n. pixel con intensità k
2	Probabilità	$p[k] = h[k]/(L \cdot C)$
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	Applicazione	$g[i, j] = \text{lut}[f[i, j]]$

3.12.4.4 📦 Specifica di input e output (VPL)

Input:

- Riga 1: Intero L.
- Riga 2: Intero C.
- Riga 3: Intero B (numero di bit).
- Righe successive: Elementi interi della matrice.

Output:

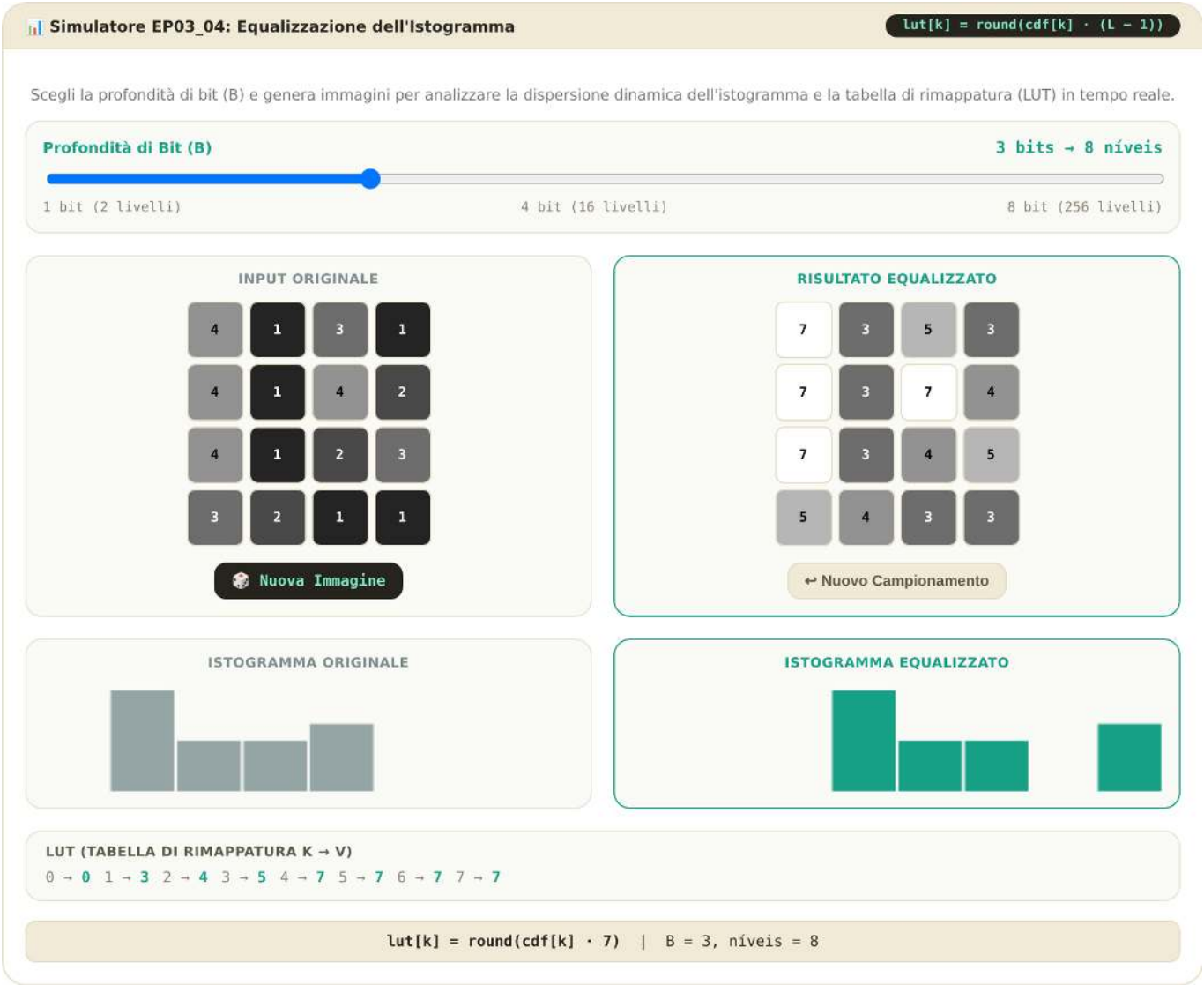
- Matrice equalizzata in L righe e C colonne.

3.12.4.5 📌 Esempi

Input	Output	Osservazione
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	Esempio a 3 bit del capitolo
1 4 3 0 0 7 7	0 0 7 7	Istogramma bimodale estremo

```
%%writefile EP03_04.cpp
// your solution
```

```
Overwriting EP03_04.cpp
```



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.it/cap03/sim-ep0304-equalizacao.html

Figura 3.29: Simulatore EP03_04: Equalizzazione dell'istogramma (Livelli L = 2^B)

```
TestSuite("EP03_04.cpp").run()
```

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

3.12.5 EP03_05 🔲 Applicazione della Maschera AND Binaria

Nei sistemi di ispezione industriale tramite visione artificiale, è necessario isolare le regioni di interesse (ROI) nelle immagini dei pezzi per verificare difetti di fabbricazione. L'operazione **AND bit a bit con una maschera binaria** è il meccanismo fondamentale per ritagliare esattamente l'area di ispezione, azzerando tutti i pixel al di fuori di essa.

Vedi in Figura 3.30 una simulazione di questo EP.

3.12.5.1 📋 Linee Guida di Implementazione

1. **Dimensioni:** Leggere gli interi L (righe) e C (colonne).
2. **Dati:** Leggere la matrice di pixel f (valori $\in [0, 255]$).
3. **Maschera:** Leggere la matrice binaria m (valori: solo 0 o 255).
4. **Mappatura:** Per ogni pixel (i, j), applicare l'AND bit a bit:

$$g(i, j) = f(i, j) \text{ AND } m(i, j)$$

dove $255 =$ `11111111` e $0 =$ `00000000` in binario.

5. **Output:** Visualizzare la matrice risultante $L \times C$.

3.12.5.2 📌 Vincoli Computazionali

- **AND con 255:** p AND $255 = p$ (tutti i bit preservati).
- **AND con 0:** p AND $0 = 0$ (tutti i bit azzerati).
- **Maschera:** Gli unici valori possibili nella maschera sono 0 e 255.
- **Implementazione:** In Python, l'AND bit a bit tra interi utilizza l'operatore `&`.

3.12.5.3 🧠 Fondamento Teorico

Pixel f	Maschera m	Risultato f AND m
qualsiasi v	255 (`11111111`)	v (preservato)
qualsiasi v	0 (`00000000`)	0 (azzerato)

3.12.5.4 📦 Specifica di Input e Output (VPL)

Input:

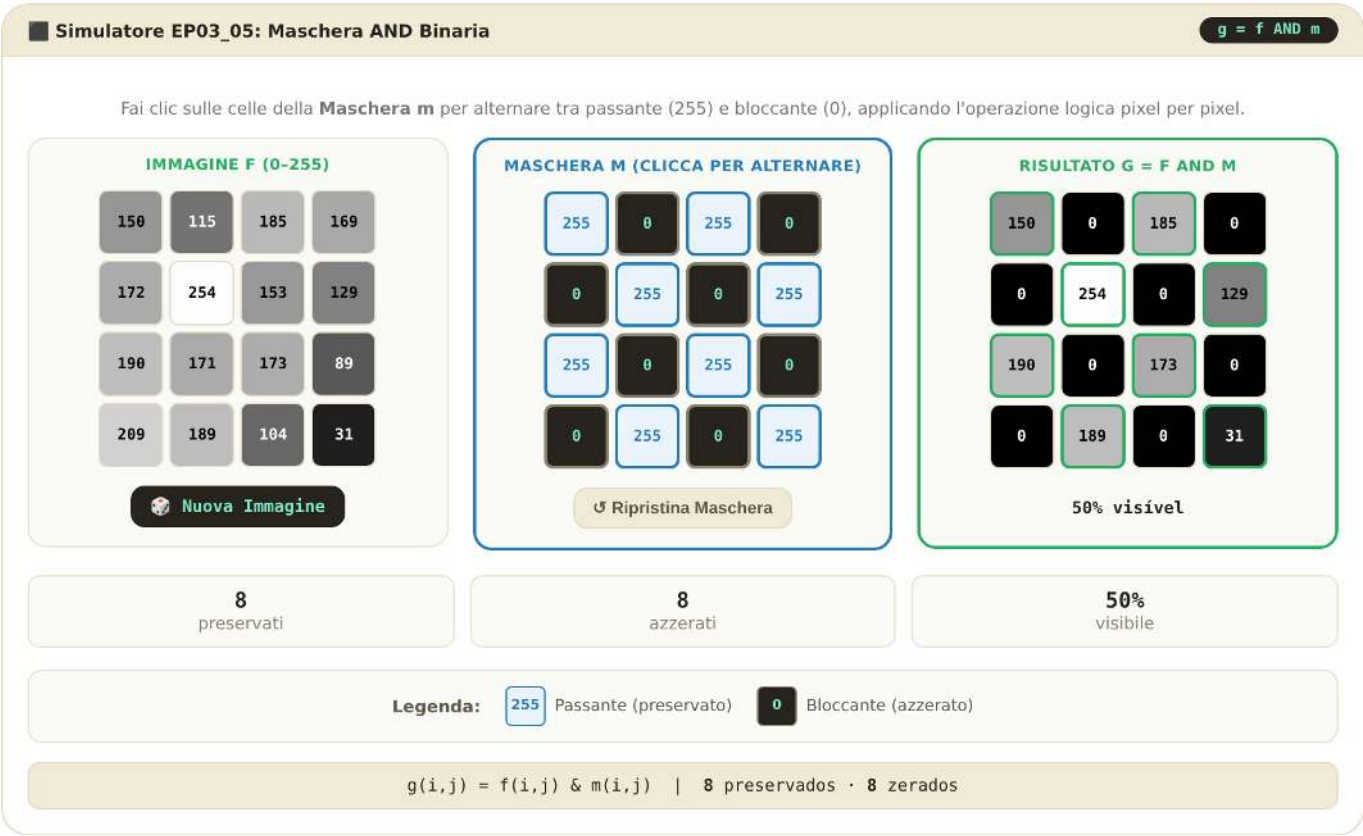
- Riga 1: Intero L.
- Riga 2: Intero C.
- Righe successive: Elementi di f (L righe).
- Righe successive: Elementi di m (L righe con valori 0 o 255).

Output:

- Matrice risultante $L \times C$.

3.12.5.5 📌 Esempi

Input	Output	Osservazione
2 3 100 150 200 50 80 120 255 255 0 0 255 255	100 150 0 0 80 120	La maschera seleziona la regione
1 4 10 20 30 40 255 0 255 0	10 0 30 0	Alternato preservato/azzerato



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.it/cap03/sim-ep0305-mascara.html

Figura 3.30: Simulatore EP03_05: Applicazione della Maschera AND Binaria

```
%%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 Filtro di Media con *Kernel* N×N

Nelle telecamere dei veicoli autonomi, le immagini acquisite sotto pioggia o nebbia presentano rumore gaussiano. Il **filtro di media** è ampiamente utilizzato per la sua riduzione in tempo reale, essendo

implementato direttamente nell'**ISP** (*Image Signal Processor*) dei sensori **CMOS** (*Complementary Metal-Oxide-Semiconductor*).

I sensori CMOS sono i sensori di immagine utilizzati nella maggior parte delle fotocamere moderne (*smartphone*, *webcam*, fotocamere automobilistiche, ecc.). Essi convertono la luce in segnali elettrici, e l'ISP elabora questi segnali in tempo reale — applicando operazioni come la riduzione del rumore, il bilanciamento del bianco e altre regolazioni dell'immagine.

Vedere in Figura 3.31 una simulazione di questo EP.

3.12.6.1 📋 Linee Guida di Implementazione

1. **Dimensioni:** Leggere gli interi L (righe), C (colonne) e N (dimensione del *kernel*, sempre dispari).
2. **Dati:** Leggere la matrice di pixel f.
3. **Filtro di Media:** Per ogni pixel (i, j) **interno** (senza bordi), calcolare:

$$g(i,j) = \text{round}\left(\frac{1}{N^2}\sum_{s=-(r)}^{r}\sum_{t=-(r)}^{r} f(i+s, j+t)\right), \quad r = \lfloor N/2 \rfloor$$

4. **Trattamento dei Bordi:** I pixel sul bordo (dove la finestra $N \times N$ supera i limiti) devono essere **copiati direttamente** dall'originale senza modifiche.
5. **Uscita:** Visualizzare la matrice risultante $L \times C$.

3.12.6.2 📌 Vincoli Computazionali

- **Raggio:** $r = \lfloor N/2 \rfloor$ (metà del *kernel*, intero).
- **Pixel interni:** (i, j) con $r \leq i < L - r$ e $r \leq j < C - r$.
- **Arrotondamento:** Usare l'arrotondamento matematico prima di convertire in intero.
- **Senza *clipping*:** La media dei valori $\in [0, 255]$ rimane in $[0, 255]$.

3.12.6.3 🧠 Fondamenti Teorici

Dimensione N	Coefficiente	Pixel nella finestra	Effetto
3	$1/9 \approx 0.111$	9	Morbido
5	$1/25 = 0.04$	25	Medio
7	$1/49 \approx 0.020$	49	Forte

3.12.6.4 📦 Specifica di Input e Output (VPL)

Input:

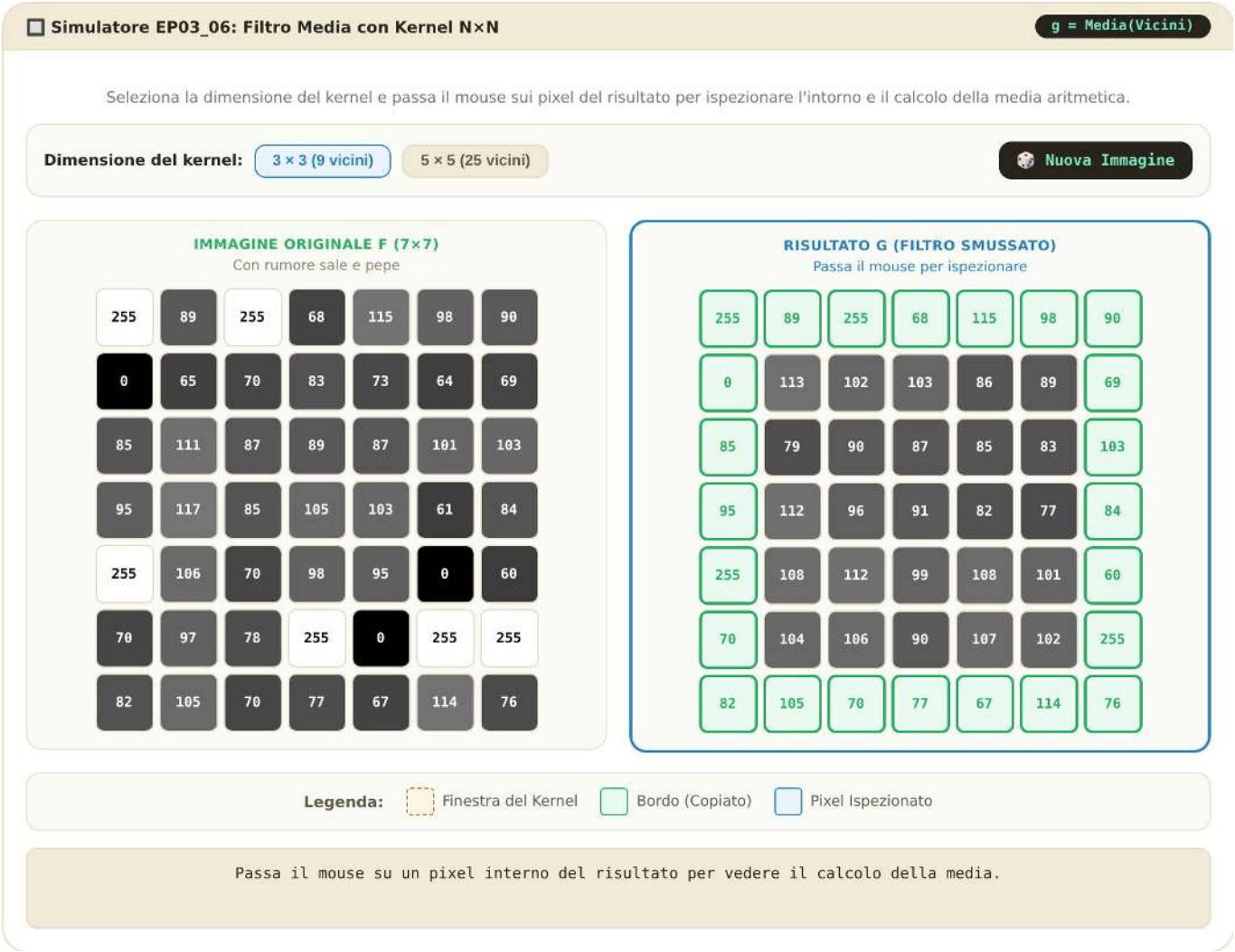
- Riga 1: Intero L.
- Riga 2: Intero C.
- Riga 3: Intero N (dispari, $N \geq 3$).
- Righe successive: Elementi della matrice originale.

Output:

- Matrice filtrata $L \times C$.

3.12.6.5 📌 Esempi

Input	Output	Osservazione
3 3 3 10 20 30 40 50 60 70 80 90	10 20 30 40 50 60 70 80 90	Solo bordo (3×3 = bordo 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 isolado: tutti i 9 pixel interni la cui finestra 3×3 include il valore 100 ricevono round(100/9)=11



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.it/cap03/sim-ep0306-media.html

Figura 3.31: Simulatore EP03_06: Filtro Medio con Kernel 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 🔍 Operatore Laplaciano (w4) per l'Enfatizzazione dei Bordi

Nelle tomografie ad alta risoluzione, la nitidezza dei bordi tra i tessuti è critica per la diagnosi. L'**operatore Laplaciano** è ampiamente utilizzato nei *pipeline* di pre-elaborazione delle immagini mediche per enfatizzare automaticamente i contorni anatomici prima della segmentazione, evitando l'intervento manuale del radiologo.

Vedi in Figura 3.32 una simulazione di questo EP.

3.12.7.1 📋 Linee Guida di Implementazione

1. **Dimensioni:** Leggere gli interi L (righe) e C (colonne).
2. **Dati:** Leggere la matrice di pixel f.
3. **Laplaciano (w4):** Per ogni pixel **interno** (i, j) con $1 \leq i < L-1$, $1 \leq j < C-1$, calcolare:

$$\nabla^2 f(i,j) = f(i-1,j) + f(i+1,j) + f(i,j-1) + f(i,j+1) - 4 \cdot f(i,j)$$

4. **Enfatizzazione:** Calcolare l'immagine enfatizzata:

$$g(i,j) = \text{clip}(f(i,j) - \nabla^2 f(i,j))$$

5. **Bordo:** I pixel sul bordo vengono copiati direttamente: $g(i,j) = f(i,j)$.
6. **Uscita:** Visualizzare la matrice enfatizzata $L \times C$.

3.12.7.2 📌 Vincoli Computazionali

- ***Kernel* w4:** $\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$ — solo 4-vicini.
- **Saturazione:** $\text{clip}(x) = \max(0, \min(255, x))$ applicato al risultato dell'enfatizzazione.
- **Senza arrotondamento:** Il Laplaciano usa solo somme/sottrazioni di interi.

3.12.7.3 🧠 Fondamenti Teorici

Regione	$\nabla^2 f$	Effetto dell'Enfatizzazione
Uniforme	≈ 0	Nessuna modifica
Bordo crescente	< 0	Pixel schiarito
Bordo decrescente	> 0	Pixel scurito

3.12.7.4 📦 Specifica di Ingresso e Uscita (VPL)

Ingresso:

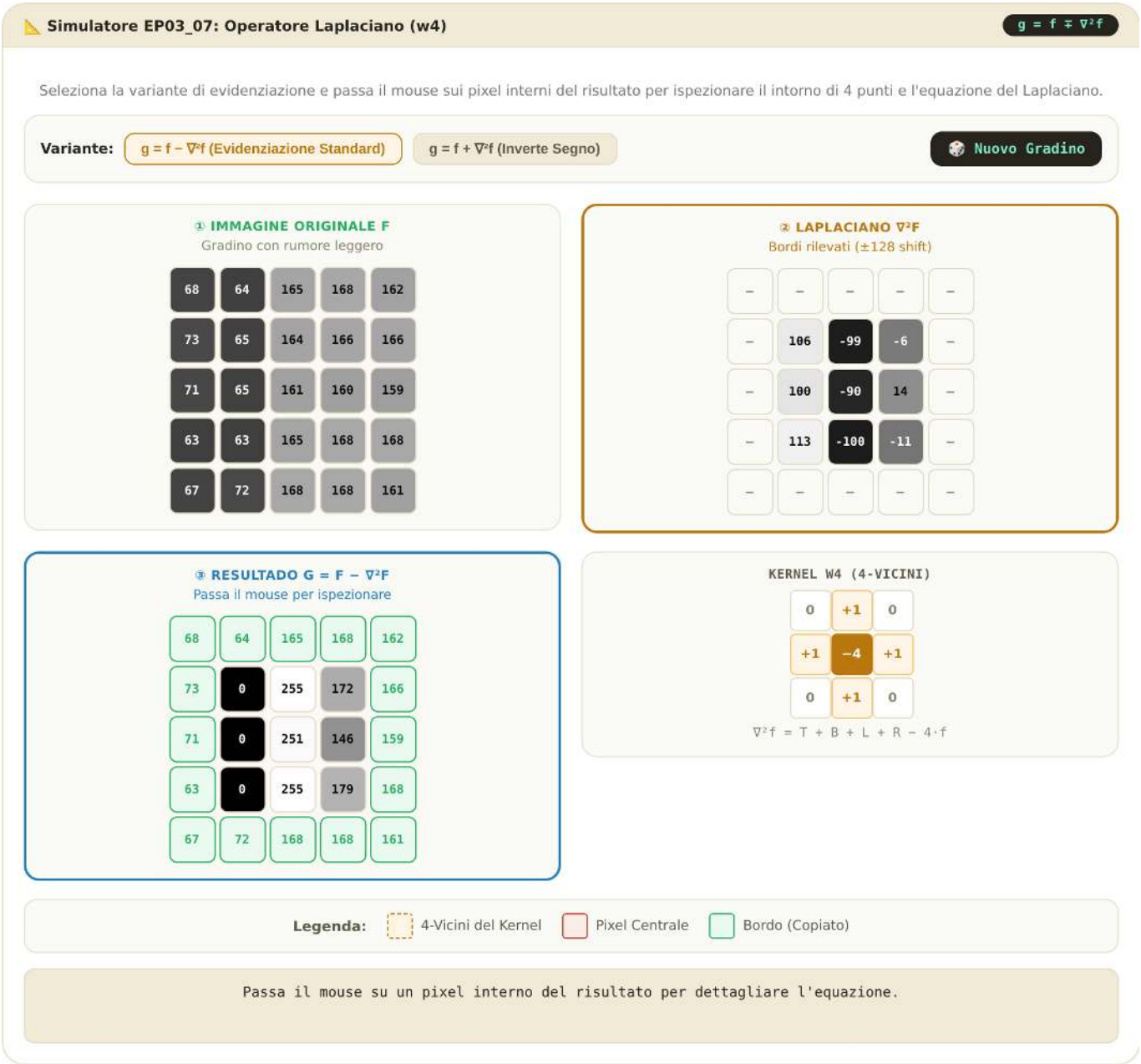
- Riga 1: Intero L.
- Riga 2: Intero C.
- Righe successive: Elementi della matrice originale.

Uscita:

- Matrice enfatizzata $L \times C$.

3.12.7.5 📌 Esempi

Ingresso	Uscita	Osservazione
3 3 0 0 0 0 100 0 0 0 0	0 0 0 0 255 0 0 0 0	Picco isolato: 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	Regione uniforme: Laplaciano=0, nessuna modifica



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.it/cap03/sim-ep0307-laplaciano.html

Figura 3.32: Simulador EP03_07: Operador Laplaciano (w4) para Realce de Bordas

```
%%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 🧭 Gradiente di Sobel: Gx e Gy

Nei robot esploratori di Marte (come il Perseverance), il rilevamento degli ostacoli viene eseguito in tempo reale da telecamere stereoscopiche. L'**operatore di Sobel** calcola il gradiente direzionale della scena e viene utilizzato nell'algoritmo di rilevamento dei bordi per identificare rocce, crepe e dislivelli del terreno che potrebbero compromettere la navigazione.

Vedi in Figura 3.33 una simulazione di questo EP.

3.12.8.1 📋 Linee guida di implementazione

1. **Dimensioni:** Leggere gli interi L (righe) e C (colonne).
2. **Dati:** Leggere la matrice f.
3. **Gx e Gy:** Per ogni pixel **interno** (i, j) con $1 \leq i < L-1$, $1 \leq j < C-1$:

$$G_x(i,j) = [f(i-1,j+1) + 2f(i,j+1) + f(i+1,j+1)] - [f(i-1,j-1) + 2f(i,j-1) + f(i+1,j-1)]$$

$$G_y(i,j) = [f(i+1,j-1) + 2f(i+1,j) + f(i+1,j+1)] - [f(i-1,j-1) + 2f(i-1,j) + f(i-1,j+1)]$$

4. **Magnitudo:** $|\nabla f(i,j)| = \text{clip}(\text{round}(\sqrt{G_x^2 + G_y^2}))$.
5. **Bordo:** I pixel di bordo ricevono magnitudo 0.
6. **Output:** Visualizzare il magnitudo $L \times C$.

3.12.8.2 📌 Vincoli computazionali

- **Arrotondamento:** Applicare `round` prima di convertire in intero.
- **Saturazione:** $\text{clip}(x) = \max(0, \min(255, x))$.
- **Radice quadrata:** Usare $\sqrt{G_x^2 + G_y^2}$ (non l'approssimazione $|G_x| + |G_y|$).

3.12.8.3 🧠 Fondamenti teorici

Operatore	Rileva	Coefficienti diagonali
G_x	Bordi verticali	± 1
G_y	Bordi orizzontali	± 1
$\lvert\nabla f\rvert$	Tutti i bordi	Combinato

3.12.8.4 📦 Specifica di input e output (VPL)

Input:

- Riga 1: Intero L.

- Riga 2: Intero C.
- Righe successive: Elementi della matrice.

Output:

- Magnitudo del gradiente, matrice $L \times C$.

3.12.8.5 📌 Esempi

Input	Output	Osservazione
3 3 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0	Immagine nulla: gradiente zero
3 3 0 0 255 0 0 255 0 0 255	0 0 0 0 255 0 0 0 0	Bordo verticale centrale: Gx alto

```
%%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 📡 Filtro da Mediana 3×3

Le immagini radar ad apertura sintetica (SAR) utilizzate nel monitoraggio ambientale e militare sono soggette a un tipo specifico di rumore chiamato *speckle*, che presenta caratteristiche simili al rumore sale e pepe. Il **filtro della mediana** è il metodo standard per la rimozione di tale rumore perché preserva i bordi delle strutture eliminando al contempo i punti spuri.

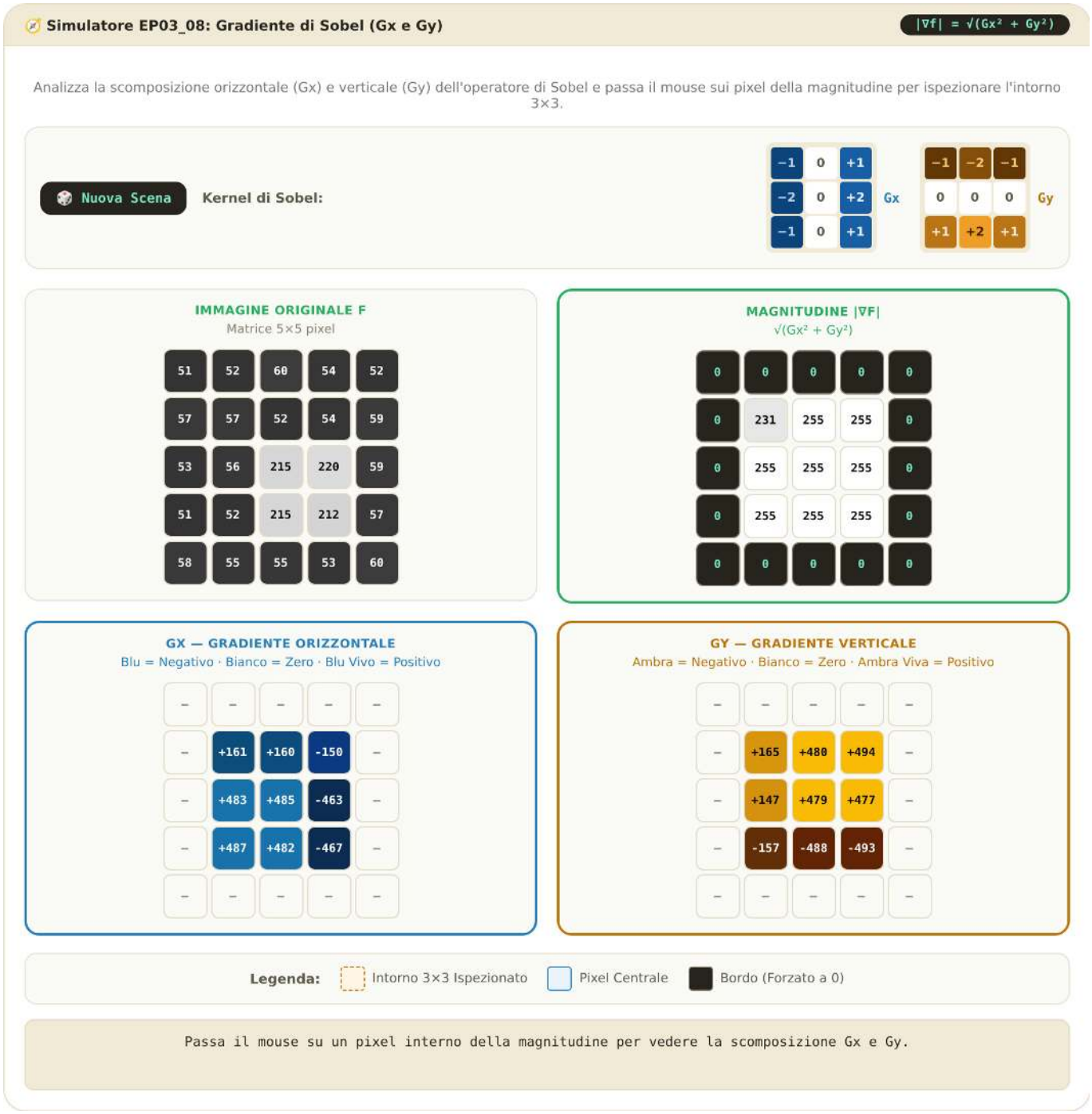
Vedere in Figura 3.34 una simulazione di questo EP.

3.12.9.1 📋 Linee Guida di Implementazione

1. **Dimensioni:** Leggere gli interi L (righe) e C (colonne).
2. **Dati:** Leggere la matrice dei pixel f.
3. **Filtro della Mediana 3×3:** Per ogni pixel **interno** (i, j) con $1 \leq i < L-1$, $1 \leq j < C-1$:
   - Raccogliere i 9 pixel del vicinato 3×3: $\{f(i+s, j+t) : s, t \in \{-1, 0, 1\}\}$.
   - Ordinare i 9 valori in ordine crescente.
   - Assegnare a $g(i, j)$ il valore centrale (posizione indice 4, considerando indice 0).

$$g(i,j) = \text{mediana}\{f(i+s, j+t) : s, t \in \{-1, 0, 1\}\}$$

4. **Bordo:** Copiare direttamente: $g(i, j) = f(i, j)$.
5. **Uscita:** Visualizzare la matrice filtrata $L \times C$.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.it/cap03/sim-ep0308-sobel.html

Figura 3.33: Simulatore EP03_08: Gradiente di Sobel (Gx e Gy)

3.12.9.2 📌 Vincoli Computazionali

- **Finestra:** Sempre $3 \times 3 = 9$ elementi.
- **Mediana:** L'elemento centrale della sequenza ordinata (indice 4 da 0 a 8).
- **Nessun clipping:** La mediana di valori in $[0, 255]$ rimane in $[0, 255]$.
- **Non lineare:** Il filtro mediana non può essere espresso come convoluzione lineare.

3.12.9.3 🧠 Fondamenti Teorici

Rumore	Filtro della Media	Filtro della Mediana
Sale e pepe (0 o 255)	Diffonde il rumore	Rimuove senza distorcere i bordi
Gaussiano	Riduce efficacemente	Riduce parzialmente

3.12.9.4 📦 Specifica di Ingresso e Uscita (VPL)

Ingresso:

- Riga 1: Intero L.
- Riga 2: Intero C.
- Righe successive: Elementi della matrice.

Uscita:

- Matrice filtrata $L \times C$.

3.12.9.5 📌 Esempi

Ingresso	Uscita	Osservazione
3 3 100 100 100 100 0 100 100 100 100	100 100 100 100 100 100 100 100 100	Punto nero eliminato: mediana di $8\times100+1\times0 = 100$
3 3 50 50 50 50 255 50 50 50 50	50 50 50 50 50 50 50 50 50	Punto bianco (sale) eliminato

```
%%writefile EP03_09.cpp
// your solution
```

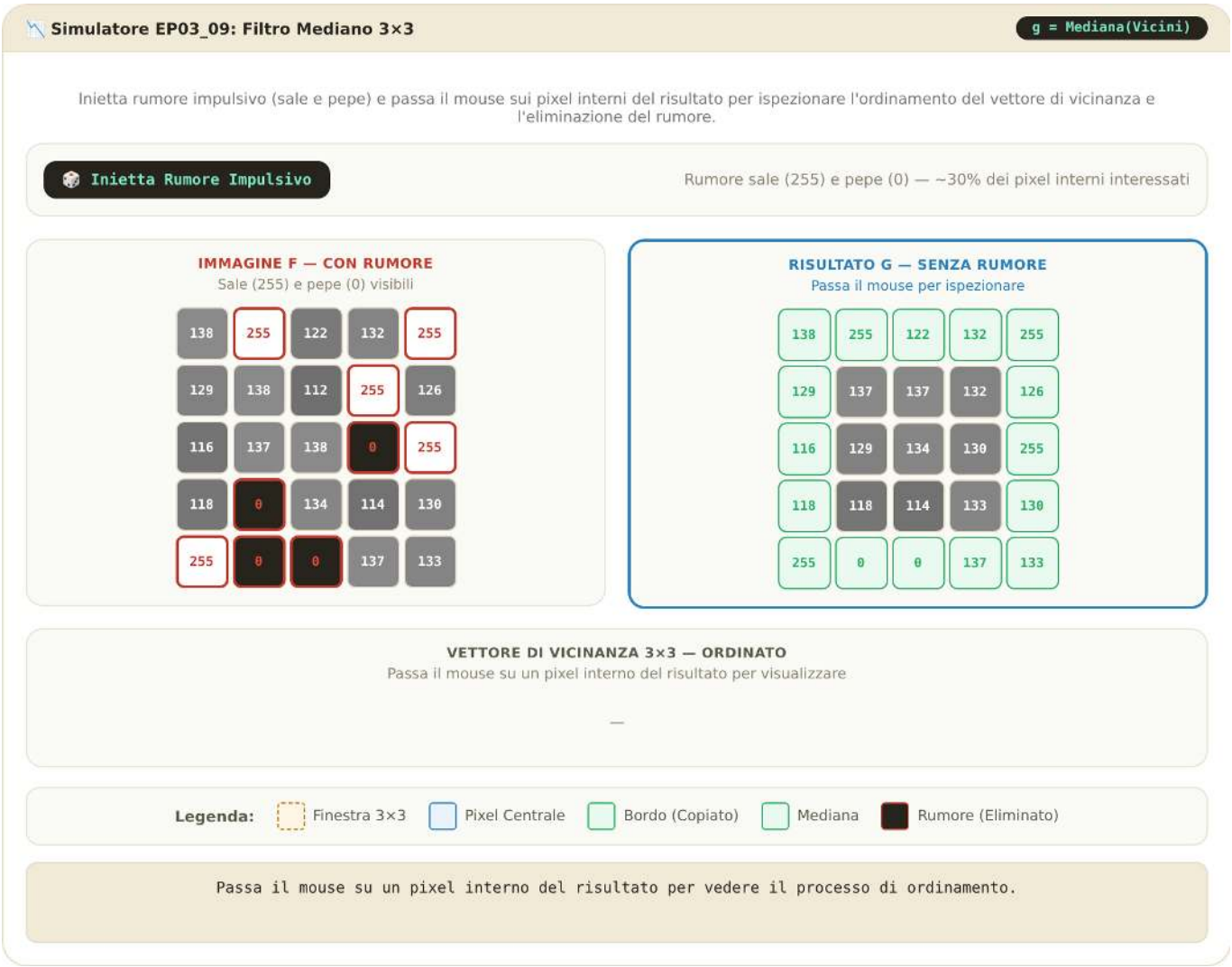
```
Overwriting EP03_09.cpp
```

```
TestSuite("EP03_09.cpp").run()
```

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

3.12.10 EP03_10 ✨ *Unsharp Masking* (USM)

Nei sistemi di digitalizzazione di documenti storici e opere d'arte, la nitidezza delle immagini è fondamentale per la lettura di testi manoscritti e dettagli ornamentali. Lo ***Unsharp Masking*** **(USM)** è l'algoritmo



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.it/cap03/sim-ep0309-mediana.html

Figura 3.34: Simulatore EP03_09: Filtro della Mediana 3×3

di miglioramento della nitidezza standard utilizzato negli *scanner* professionali e in software come Adobe Photoshop, controllato dal parametro k che determina l'intensità del miglioramento.

Vedi in Figura 3.35 una simulazione di questo EP.

3.12.10.1 📋 Linee Guida di Implementazione

1. **Dimensioni:** Leggere gli interi L (righe) e C (colonne).
2. **Parametro:** Leggere il valore reale k (intensità del miglioramento, $k \geq 0$).
3. **Dati:** Leggere la matrice di pixel f.
4. **Smussamento:** Calcolare $\bar{f}$ con filtro di media 3×3 (solo pixel interni; bordi mantenuti):

$$\bar{f}(i,j) = \frac{1}{9} \sum_{s=-1}^{1} \sum_{t=-1}^{1} f(i+s, j+t)$$

5. **Maschera ad alta frequenza:** $m(i,j) = f(i,j) - \bar{f}(i,j)$.
6. **Miglioramento USM:** Per ogni pixel interno:

$$g(i,j) = \text{clip}\left(\text{round}\left(f(i,j) + k \cdot m(i,j)\right)\right)$$

7. **Bordo:** $g(i,j) = f(i,j)$ (copia diretta).
8. **Output:** Visualizzare la matrice migliorata $L \times C$.

3.12.10.2 📌 Vincoli Computazionali

- **Arrotondamento:** Applicare `round` prima del clipping.
- **Saturazione:** $\text{clip}(x) = \max(0, \min(255, x))$.
- **Operazioni in virgola mobile:** Calcolare $\bar{f}$ e m in virgola mobile prima di arrotondare il risultato finale.
- $k = 0$**:** Nessun miglioramento — l'output è identico all'input (eccetto per i bordi).

3.12.10.3 🧠 Fondamento Teorico

Fase	Operazione	Descrizione
1	$\bar{f} = f * \frac{1}{9}\mathbf{1}_{3\times3}$	Smussamento (basse frequenze)
2	$m = f - \bar{f}$	Maschera (alte frequenze)
3	$g = \text{clip}(\text{round}(f + k \cdot m))$	Miglioramento ponderato

3.12.10.4 📦 Specifica di Input e Output (VPL)

Input:

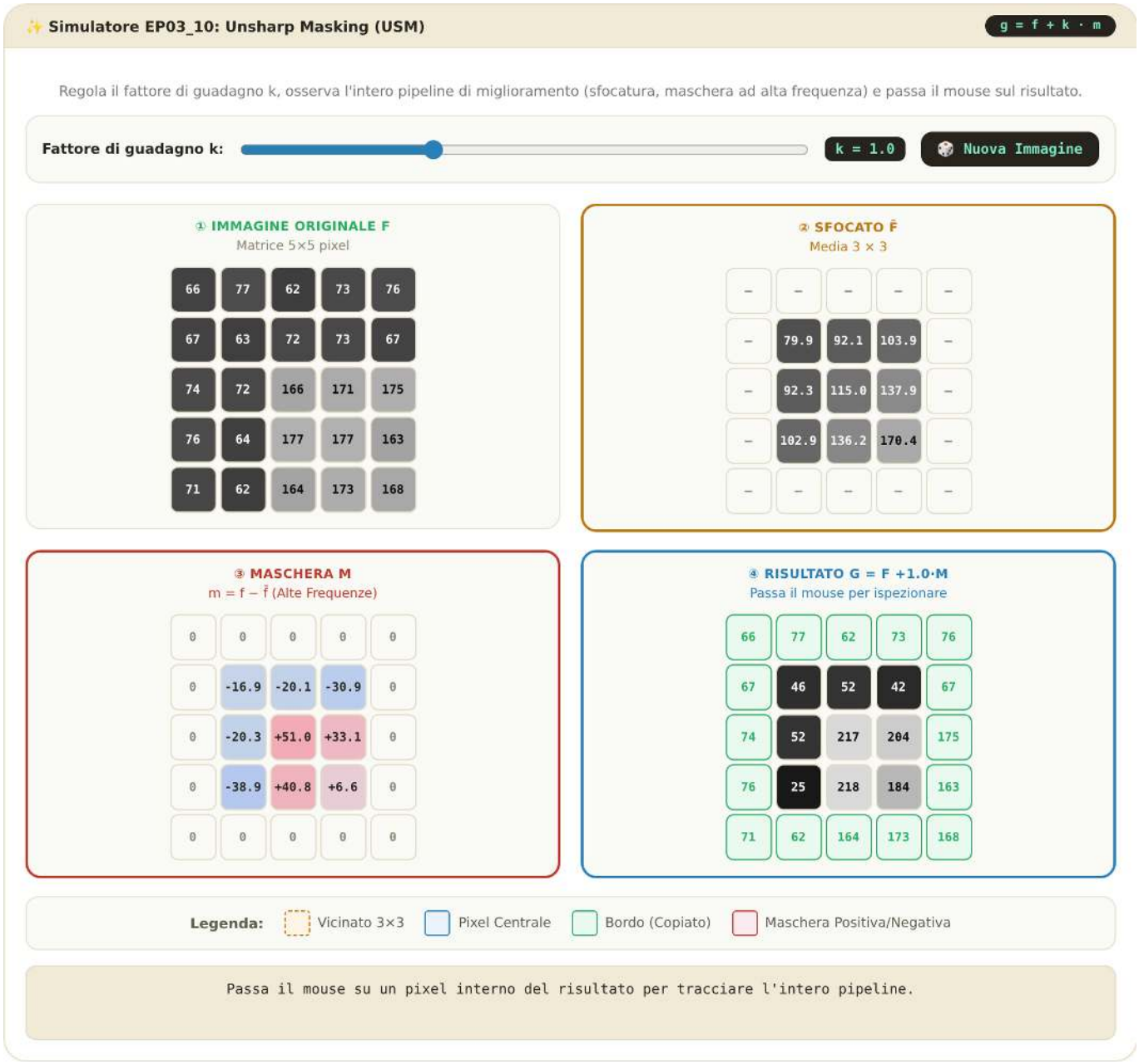
- Riga 1: Intero L.
- Riga 2: Intero C.
- Riga 3: Reale k.
- Righe successive: Elementi della matrice originale.

Output:

- Matrice migliorata $L \times C$.

3.12.10.5 📌 Esempi

Input	Output	Osservazione
3 3 0.0 100 100 100 100 100 100 100 100 100	100 100 100 100 100 100 100 100 100	k=0: nessun miglioramento
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 centrale migliorato e saturato



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.it/cap03/sim-ep0310-unsharp.html

Figura 3.35: Simulatore 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>
```

Capitolo 4

Morfologia Matematica e Segmentazione delle Immagini

Questo capitolo presenta due temi fondamentali dell'Elaborazione Digitale delle Immagini (EDI): la **morfologia matematica** e la **segmentazione delle immagini**. La morfologia matematica fornisce un quadro teorico basato sulla teoria degli insiemi per analizzare, raffinare e quantificare la forma degli oggetti in immagini binarie e in scala di grigi, mediante operatori fondamentali come l'**erosione** e la **dilatazione**. La segmentazione, a sua volta, ha l'obiettivo di partizionare l'immagine in regioni di interesse, separando gli oggetti dallo sfondo e producendo rappresentazioni adeguate per l'analisi e l'interpretazione.

Il capitolo inizia con la **sogliatura**, una delle tecniche più importanti di segmentazione, introducendo il metodo automatico di **Otsu** e riprendendo l'analisi degli istogrammi tramite la varianza interclasse, presentata nel Capitolo 1. Successivamente, vengono studiati i principali operatori della morfologia matematica, inclusi erosione, dilatazione, apertura, chiusura e ricostruzione morfologica, che consentono di raffinare le maschere binarie e preservare le strutture rilevanti degli oggetti. Infine, vengono presentate tecniche di segmentazione basata su regioni, come l'**etichettatura delle componenti connesse**, la **trasformata della distanza** e l'algoritmo ***watershed*** basato su marcatori, culminando nell'estrazione di descrittori geometrici e nella generazione di *bounding boxes* compatibili con i moderni sistemi di rilevamento degli oggetti.

4.1 Obiettivi

Al termine di questo capitolo, sarai in grado di:

- **Applicare la sogliatura:** Comprendere il criterio automatico di Otsu basato sulla massimizzazione della varianza tra le classi (σ_B^2) e selezionare strategie adeguate di pre-elaborazione per facilitare la segmentazione;
- **Padroneggiare la morfologia binaria:** Comprendere e applicare l'erosione ($A \ominus B$) e la dilatazione ($A \oplus B$) come operatori fondamentali, derivando l'apertura ($A \circ B$), la chiusura ($A \bullet B$) e le operazioni basate sulla ricostruzione morfologica, come `mm::clohole` e `mm::edgeoff`;
- **Applicare la morfologia in toni di grigio:** Utilizzare il gradiente morfologico e i filtri *top-hat* per l'enhancement e l'analisi di strutture locali;
- **Etichettare i componenti connessi:** Identificare e separare regioni connesse in immagini binarie mediante algoritmi di etichettatura;
- **Applicare la trasformata della distanza:** Interpretare e calcolare le distanze dallo sfondo utilizzando approcci morfologici e metriche geometriche;
- **Segmentare per regioni:** Costruire *pipeline* di segmentazione basate su marcatori utilizzando la Trasformata della Distanza e l'algoritmo ***watershed***;
- **Estrarre descrittori geometrici:** Calcolare proprietà come area, perimetro, centroide, circolarità e *bounding box* tramite `mm::label0` ed estrazione dei contorni;
- **Mettere in relazione l'elaborazione delle immagini digitali e la visione artificiale:** Comprendere come i descrittori estratti tramite segmentazione possano essere convertiti in formati utilizzati da rilevatori moderni, come YOLO.

```
import os, urllib.request

os.makedirs("tmp/state", exist_ok=True)  # artefatti di build del percorso C++ (.cpp, binario,
PNG)

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")

# Il kernel è Python anche nel percorso C++: `mm` (morph.py) è usato dai
# simulatori, dalla visualizzazione delle figure che il binario C++ genera e dallo
# stato mm::Image tra le celle. cpp=True scarica anche il percorso compilato
# (morph.hpp + stb_image*.h), usato nell'#include delle celle %%writefile *.cpp.
# Le celle C++ di questo capitolo compilano CON OpenCV (-DMM_USE_OPENCV +
# pkg-config opencv4): mm::dil/ero → cv::dilate/erode, quindi open/close/asf/
# gradm/tophat/blackhat girano a piena risoluzione e coincidono bit a bit con il percorso py.
import config
config.setup(cpp=True)
from morph import mm
import numpy as np
```

```
Ambiente pronto. Morph: 1.1.9 | OpenCV: 5.0.0
```

4.2 Soglie

La **sogliatura** (*thresholding*) è una delle forme più semplici ed efficaci di segmentazione delle immagini. Il suo obiettivo è classificare ogni pixel in due classi di intensità, normalmente associate a *oggetto* e *sfondo*:

$$g(x,y) = \begin{cases} 255, & \text{se } f(x,y) > T \\ 0, & \text{caso contrário} \end{cases} \tag{4.1}$$

dove $f(x,y)$ rappresenta l'intensità del pixel nell'immagine originale e $g(x,y)$ l'immagine binaria risultante.

La scelta della soglia T è importante per la qualità della segmentazione. Il metodo di **Otsu** determina automaticamente la soglia ottimale massimizzando la **varianza tra le classi** σ_B^2, definita come:

$$\sigma_B^2(T) = w_0(T)\, w_1(T)\, \left[\mu_0(T) - \mu_1(T)\right]^2 \tag{4.2}$$

dove:

- $w_0(T)$ e $w_1(T)$ sono le probabilità cumulative delle classi sfondo e oggetto;
- $\mu_0(T)$ e $\mu_1(T)$ sono le medie di intensità di queste classi;
- $\sigma_B^2(T)$ rappresenta la varianza tra le classi per una data soglia T.

Il metodo funziona meglio quando l'istogramma presenta due gruppi di intensità relativamente separati. A tal fine, l'algoritmo valuta tutte le possibili soglie dell'immagine — tipicamente nell'intervallo $[0, 255]$ per immagini a 8 bit — e seleziona il valore che massimizza la varianza tra le classi, denotata da σ_B^2:

$$T^* = \arg\max_{T \in [0,255]} \sigma_B^2(T)$$

i Otsu presuppone istogrammi bimodali

Il metodo di Otsu produce risultati migliori quando l'istogramma presenta due picchi ben definiti (*bimodalità*), corrispondenti allo sfondo e all'oggetto. Quanto maggiore è la separazione tra questi picchi

e quanto più pronunciato è il massimo di σ_B^2, tanto più affidabile tende a essere la soglia ottenuta.
In immagini con illuminazione non uniforme o con multiple regioni di intensità, le tecniche di **sogliatura adattiva** — nelle quali la soglia viene calcolata localmente — tendono a produrre segmentazioni più robuste.
L'indice B in σ_B^2 significa ***between classes*** (*tra le classi*). Pertanto, σ_B^2 rappresenta la **varianza tra le classi** (*between-class variance*).

4.2.1 Immagine di Monete

L'immagine utilizzata per esercitarsi nella segmentazione è una fotografia di una collezione di monete di diversi paesi ed epoche (Figura 4.1). Crediti: GAZI.MD.AHAD (CC BY-SA 4.0). Essa presenta oggetti circolari con bordi ben definiti, risultando ideale per dimostrare la sogliatura, gli operatori morfologici, la trasformata della distanza, il *watershed* e i descrittori di forma.

```
%%writefile tmp/fig_04_coins.cpp
#define MM_OUT "tmp/fig_04_coins.png"
// Compile with: g++ -std=c++17 -o program program.cpp -I<path-to-morph.hpp> -L<path-to-libs>
-lmorph -lOpenCV

#include "morph.hpp"
#include <opencv2/opencv.hpp>
#include <iostream>
#include <filesystem>

int main() {
    // 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 << "]" << std::endl;
    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)")
```

Figura 4.1: Imagem com moedas de vários tipos. Crédito: GAZI.MD.AHAD (CC BY-SA 4.0).

4.2.2 Pre-processing per l'Otsu

Il metodo di Otsu dipende da un istogramma ben **bimodale**. L'immagine delle monete presenta un'illuminazione non uniforme e monete scure vicine allo sfondo, il che lo ostacola. Nel percorso C++ si applica **l'equalizzazione globale dell'istogramma** (`mm::equalize`) prima della sogliatura — il confronto CLAHE × Gaussiano e le curve $\sigma_B^2(T)$ sono trattati nel percorso Python (Figura 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 otsu_clahe otsu_clahe.cpp -I. -lopencv_core -lopencv_imgproc
-lopencv_highgui $(pkg-config --cflags --libs opencv4) -lm

#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 provided as input

    // Realce adaptativo com limite de contraste (CLAHE)
    mm::Image img_clahe = mm::clahe(img_coins_gray, 2.0, 8);

    // Suavização Gaussiana opcional (preservada do código original, embora não exibida)
    mm::Image img_gauss = mm::gaussian(img_clahe, 5, 0);

    // Exibição: original, CLAHE, histograma do CLAHE e binarização por Otsu
    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)")
```

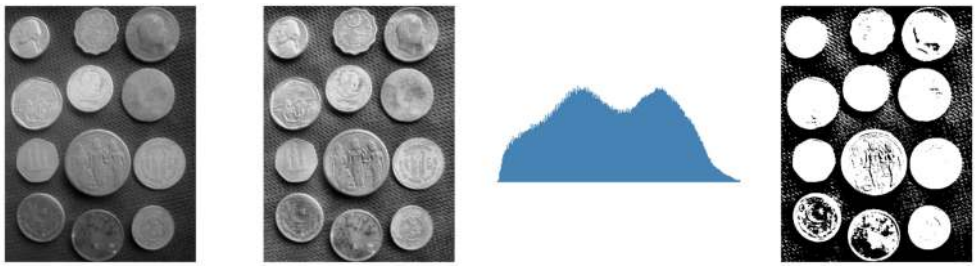


Figura 4.2: CLAHE (realce adaptativo com limite de contraste) antes da limiarização de Otsu: original, realçado, histograma e binarização. (A trilha Python também traça as curvas de $\sigma_B^2(T)$.)

4.2.3 Risultato: CLAHE come Miglior Preprocessamento

L'analisi della Figura 4.2 indica che il **CLAHE** ha ottenuto il valore più alto della varianza inter-classi ($\sigma_B^2 \approx 2{,}47 \times 10^3$), con soglia ottimale $T^* = 122$. Sebbene la combinazione **CLAHE+Gaussiano** abbia prodotto un risultato molto simile ($\sigma_B^2 \approx 2{,}43 \times 10^3$, $T^* = 123$), il criterio quantitativo del metodo di Otsu favorisce leggermente l'uso del CLAHE da solo.

In termini visivi, le immagini binarizzate ottenute con CLAHE e CLAHE+Gaussiano sono praticamente equivalenti. La differenza tra i due approcci diventa più evidente nell'analisi degli istogrammi e dei valori di $\sigma_B^2(T)$ che nell'ispezione diretta delle segmentazioni risultanti. Pertanto, la scelta del CLAHE si basa principalmente sulla massimizzazione della separazione statistica tra le classi di sfondo e oggetto.

Interpretazione dei risultati

Si noti che i preprocessamenti con CLAHE e CLAHE+Gaussiano producono istogrammi e soglie ottimali molto vicini ($T^* = 122$ e $T^* = 123$). Di conseguenza, le immagini binarizzate risultanti sono anch'esse piuttosto simili. In questo caso, la decisione non si basa su differenze visive marcate, ma sul criterio oggettivo del metodo di Otsu: il valore più alto di σ_B^2 indica la migliore separazione tra le classi.

4.3 Morfologia Matematica

La **morfologia matematica** è una teoria basata su insiemi utilizzata per analizzare la forma e la struttura degli oggetti nelle immagini. A differenza dei filtri lineari presentati nel Capitolo 3, gli operatori morfologici sono **non lineari**, poiché si basano su operazioni di minimo, massimo e inclusione spaziale, anziché su combinazioni lineari di intensità. Questi operatori agiscono sul vicinato di ciascun pixel tramite un **elemento strutturante** $\mathbb{B}$, responsabile di definire la forma e la dimensione della regione analizzata.

Nelle immagini binarie e in scala di grigi con elementi piatti, l'elemento strutturante traslato nella posizione x è definito spazialmente come:

$$\mathbb{B}_x = \{x + b \mid b \in \mathbb{B}\}$$

Nelle regioni di bordo dell'immagine, parte dell'insieme $\mathbb{B}_x$ può estendersi oltre il dominio fisico della scena ($\mathbb{E}$). Per garantire la coerenza matematica degli operatori primitivi in questi confini, si assume teoricamente che lo spazio esterno al dominio dell'immagine sia riempito con l'**elemento neutro** dell'operazione corrispondente (infinito positivo per l'erosione e infinito negativo per la dilatazione), impedendo che l'ambiente esterno corrompa le strutture interne dell'oggetto.

Quando l'elemento strutturante associa pesi ai suoi elementi — cioè $b : \mathbb{B} \to \mathbb{Z}$ — esso è denominato **funzione strutturante** o **elemento strutturante non piano**.

Sviluppata da Matheron e Serra negli anni Sessanta per immagini binarie e successivamente estesa alla scala di grigi, la morfologia matematica fonda operatori come il gradiente morfologico, il *top-hat*, il *watershed* e la

trasformata della distanza, tutti derivati da due primitivi: **erosione** e **dilatazione** [Matheron (1975); Serra (1982)].

4.3.1 Erosione e Dilatazione

I due operatori primitivi sono definiti in modo unificato per immagini in toni di grigio ($f : \mathbb{E} \to \mathbb{Z}$) e, per restrizione al dominio $\{0, 1\}$, anche per immagini binarie.

4.3.1.1 Erosione

L'**Erosione** di un'immagine f mediante una funzione strutturante $b : \mathbb{B} \to \mathbb{Z}$ è definita formalmente da:

$$\varepsilon_b(f)(x) = (f \ominus b)(x) = \min_{z \in \mathbb{B}} \{ f(x+z) - b(z) \}, \quad \forall\, x \in \mathbb{E} \tag{4.3}$$

Nella pratica, l'erosione sostituisce l'intensità del pixel x con il valore minimo risultante dalla differenza tra l'immagine e l'elemento strutturante nell'intorno definito dal dominio $\mathbb{B}$. Valori positivi nei pesi di $b(z)$ spingono il risultato locale verso il basso, "scavando" il rilievo dell'immagine più profondamente e intensificando l'erosione.

Nel caso **piano** (dove i pesi sono nulli all'interno del dominio, cioè $b \equiv 0$), l'espressione si semplifica al minimo locale puro:

$$\varepsilon_B(f)(x) = \min\{ f(y) : y \in \mathbb{B}_x \}$$

Nelle immagini binarie, questa operazione equivale a richiedere che l'insieme $\mathbb{B}$, traslato alla coordinata x, sia **completamente contenuto** nell'oggetto A:

$$A \ominus \mathbb{B} = \{ z \in \mathbb{E} \mid \mathbb{B}_z \subseteq A \}$$

Effetto Visivo: *Rimpicciolisce* oggetti e strutture chiare, eliminando protuberanze, picchi luminosi o rumori che siano geometricamente più piccoli del dominio $\mathbb{B}$.

4.3.1.2 Implementazione dell'erosione

La versione didattica `mm::ero0` implementa il caso particolare di erosione con **elemento strutturante piano**. Per ogni pixel (y, x), la funzione esamina i vicini spaziali consentiti da B e memorizza il valore minimo trovato nell'immagine di ingresso f, riproducendo direttamente l'operazione di minimo locale descritta nella Equazione 4.3 per $b \equiv 0$.

Si noti che i valori dei vicini vengono sempre letti in modo statico dall'immagine originale f; la matrice di uscita g viene utilizzata esclusivamente per registrare il minimo accumulato dell'intorno corrente. In questo modo, il risultato finale è invariante rispetto all'ordine di scansione dei pixel (sia per righe che per colonne).

La funzione ausiliaria `_viz` calcola le coordinate dei vicini validi entro i limiti fisici dell'immagine. Ai bordi, l'inizializzazione dell'accumulatore a `255` emula esattamente il riempimento con elemento neutro richiesto dalla teoria. La funzione di interfaccia `mm::ero` ricorre invece all'implementazione nativa e ottimizzata di OpenCV (`mm::ero`) quando l'elemento strutturante è piano, passando alla routine generale `mm::ero1` qualora l'elemento presenti pesi topografici.

L'esempio computazionale seguente illustra l'applicazione di un elemento strutturante a croce (`mm::secross()`) evidenziato nella Figura 4.3, confrontando l'esecuzione della variante didattica a cicli (`mm::ero0`) con il motore computazionale di 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"

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)")
```

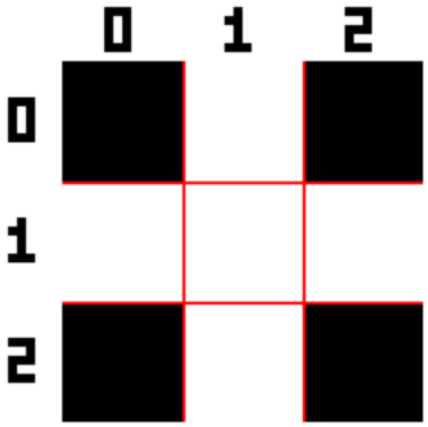


Figura 4.3: Elemento estruturante em formato de cruz (B_{cruz}) utilizado para conectividade-4.

```
# La morph.hpp è header-only; sotto, l'helper _viz e il corpo di 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 Dilatazione

La **Dilatazione** di un'immagine f mediante una funzione strutturante $b : \mathbb{B} \to \mathbb{Z}$ è definita formalmente da:

$$\delta_b(f)(x) = (f \oplus b)(x) = \max_{z \in \mathbb{B}} \{ f(x - z) + b(z) \}, \quad \forall\, x \in \mathbb{E} \tag{4.4}$$

In pratica, la dilatazione sostituisce l'intensità del pixel x con il valore più elevato risultante dalla somma tra l'immagine e l'elemento strutturante nell'intorno definito. L'argomento di inversione spaziale $(x - z)$ indica che la dilatazione valuta implicitamente l'elemento trasposto (riflesso) $\hat{b}$, proprietà fondamentale per garantire la dualità matematica rispetto all'erosione.

Nel caso **piano** (dove i pesi sono nulli all'interno del dominio, ovvero $b \equiv 0$), l'espressione si riduce al massimo locale puro:

$$\delta_B(f)(x) = \max\{ f(y) : y \in \mathbb{B}_x \}$$

Nelle immagini binarie, tale operazione equivale a richiedere che l'insieme riflesso $\hat{\mathbb{B}}$, traslato alla coordinata x, possieda **intersezione non vuota** con l'oggetto A:

$$A \oplus \mathbb{B} = \{ z \in \mathbb{E} \mid \hat{\mathbb{B}}_z \cap A \neq \emptyset \}$$

Effetto visivo: *Espande* le strutture chiare dell'immagine, aumentando il riempimento degli oggetti, collegando componenti vicine ed eliminando canali, cavità scure o valli che siano geometricamente più piccoli del dominio $\mathbb{B}$.

4.3.1.4 Implementazione della dilatazione

La versione didattica `mm::dil0` implementa il caso particolare di dilatazione con **elemento strutturante piatto**. Per ogni pixel (y, x), la funzione esamina i vicini spaziali consentiti da B e memorizza il valore massimo trovato nell'immagine di ingresso f, riproducendo direttamente l'operazione di massimo locale per $b \equiv 0$.

Prima di iniziare la scansione spaziale, l'elemento strutturante subisce una riflessione geometrica tramite l'istruzione `np.flip(Bc)` per costruire esplicitamente la matrice trasposta $\hat{B}$ richiesta dalla teoria. In maschere perfettamente simmetriche (come croci, quadrati e dischi centrati nell'origine), tale riflessione non altera la disposizione dei pixel; tuttavia, per elementi asimmetrici, questo passaggio è strettamente necessario per garantire l'equivalenza con le definizioni formali e salvaguardare le leggi di dualità.

Come verificato nell'operatore di erosione, i valori dei vicini vengono sempre letti in modo statico dalla matrice originale f, mentre la matrice di uscita g agisce puramente come registratore del massimo accumulato dell'intorno. Ai bordi dell'immagine, l'inizializzazione dell'accumulatore a `0` emula esattamente il riempimento esterno con elemento neutro ($-\infty$, o zero in rappresentazioni a 8 bit), garantendo che i bordi fisici della scena vengano dilatati in perfetta conformità con lo standard adottato da OpenCV.

L'esempio computazionale seguente illustra l'applicazione pratica di un elemento a croce (`mm::secross()`), validando la coerenza tra la logica a cicli (`mm::dil0`) e il metodo nativo industriale (`mm::dil`).

```
# Corpo di mm::dil0() in morph.hpp (riflette il SE prima di scansionare, come 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 Nota: Il Confronto dei Segni ($f(x+z)$ vs $f(x-z)$)

Confronta le definizioni formali dell'erosione (Equazione 4.3) e della dilatazione (Equazione 4.4). Considera un elemento strutturante asimmetrico a destra $\mathbb{B} = \{0, 1\}$ (origine e un pixel a destra) applicato alla posizione $x = 10$.

1. **Nell'Erosione** (Equazione 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'operatore interroga il pixel corrente (10) e il pixel a destra (11), preservando l'orientamento originale di $\mathbb{B}$.
2. **Nella Dilatazione** (Equazione 4.4):

$$\max\{f(x-z)+b(z)\}$$

- $z=0 \Rightarrow f(10-0)=\mathbf{f(10)}$
- $z=1 \Rightarrow f(10-1)=\mathbf{f(9)}$

A causa del segno negativo $(-z)$, avanzare nell'elemento strutturante corrisponde a retrocedere nell'immagine, facendo sì che la dilatazione interroghi il pixel a sinistra (9).

La funzione `_viz`, utilizzata in `morph.py`, genera i vicini tramite spostamenti additivi della forma $x+z$. Per questo motivo, l'implementazione di `mm::dil0` riflette preventivamente l'elemento strutturante tramite `np.flip(B)`. Dopo la riflessione, la scansione basata su $x+z$ accede esattamente agli stessi punti definiti dall'espressione teorica $f(x-z)$ della dilatazione in Equazione 4.4.

Per elementi strutturanti simmetrici (come dischi, quadrati e croci centrate), la riflessione non altera la maschera. Per elementi asimmetrici, invece, questo passaggio è indispensabile affinché l'implementazione riproduca correttamente la definizione matematica della dilatazione e preservi la dualità erosione–dilatazione.

Dualità erosione–dilatazione

L'erosione e la dilatazione sono **duali rispetto al complemento**. Ciò significa che un operatore può essere completamente ottenuto dall'altro, purché si agisca sul complemento dell'immagine utilizzando l'elemento strutturante riflesso $\hat{B}$:

$$(A \ominus B)^c = A^c \oplus \hat{B} \quad \Longleftrightarrow \quad A \ominus B = (A^c \oplus \hat{B})^c$$

Analogamente, anche la **dilatazione può essere ottenuta dall'erosione**:

$$(A \oplus B)^c = A^c \ominus \hat{B} \quad \Longleftrightarrow \quad A \oplus B = (A^c \ominus \hat{B})^c$$

In termini pratici, l'erosione di un oggetto può essere ottenuta tramite la dilatazione del suo complemento, seguita dalla complementazione del risultato (e viceversa). Nell'implementazione del pacchetto `morph.py`, le versioni didattiche `mm::ero0` e `mm::dil0` rendono esplicita questa struttura mediante cicli (*loop*), mentre `mm::ero` e `mm::dil` delegano le operazioni a OpenCV per una maggiore efficienza computazionale.

Condizioni al contorno e immagini finite

Nella morfologia matematica classica, definita su un dominio infinito (tipicamente $\mathbb{Z}^2$), questa dualità è esatta. Nelle immagini digitali, tuttavia, si lavora con matrici finite, e il risultato dipende dal modo in cui vengono trattati i pixel situati al di fuori dell'immagine.

Affinché le identità di dualità rimangano valide, il complemento deve essere definito rispetto allo stesso universo e le condizioni al contorno adottate per l'erosione e per la dilatazione devono essere complementari tra loro. Ad esempio, se l'erosione assume che i pixel esterni appartengano all'oggetto (255), allora la dilatazione applicata al complemento deve assumere che questi stessi pixel esterni appartengano allo sfondo (0).

Quando vengono utilizzate diverse strategie di riempimento (replicazione, riflessione, valore costante, ecc.), la dualità teorica può cessare di essere soddisfatta esattamente nelle regioni prossime ai bordi dell'immagine.

Per illustrare numericamente gli operatori morfologici e la dualità erosione–dilatazione, la Figura 4.4 presenta un'immagine binaria 10×10 elaborata con un elemento strutturante a forma di "L". Nell'implementazione di `morph.py`, l'origine di B è fissata nel centro geometrico della maschera — posizione $(1,1)$ per un *kernel* 3×3 — e deve corrispondere a un elemento attivo affinché l'erosione si comporti correttamente (come discusso in precedenza). L'elemento strutturante B_L definito di seguito soddisfa questa condizione. La Figura 4.5 completa l'analisi con un simulatore interattivo dell'erosione, consentendo di visualizzare lo spostamento dell'elemento strutturante sull'immagine e di identificare le posizioni in cui esso rimane completamente contenuto nell'oggetto.

```
%%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() {
    // Criar imagem A 10x10
    mm::Image A(10, 10);

    // Preencher A com os valores do array numpy
    int vals[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) = vals[y][x] * 255;
        }
    }

    // B_L = 'L', origem no centro
    mm::SE B_L{{1,0,0},{1,1,0},{1,1,0}};
    // B_hat = B_L girado 180° (B)
    mm::SE B_hat{{0,1,1},{0,1,1},{0,0,1}};

    std::cout << "Elemento estruturante B_L:" << std::endl;
    // Criar imagens para exibir os elementos estruturantes
    mm::Image B_L_img(3, 3);
    int b_l_vals[3][3] = {{1,0,0},{1,1,0},{1,1,0}};
    for (int y = 0; y < 3; y++)
        for (int x = 0; x < 3; x++)
            B_L_img.at(y, x) = b_l_vals[y][x] * 255;
    std::cout << mm::drawImg(B_L_img) << std::endl;

    std::cout << "Elemento estruturante refletido B:" << std::endl;
    mm::Image B_hat_img(3, 3);
    int b_hat_vals[3][3] = {{0,1,1},{0,1,1},{0,0,1}};
    for (int y = 0; y < 3; y++)
        for (int x = 0; x < 3; x++)
            B_hat_img.at(y, x) = b_hat_vals[y][x] * 255;
    std::cout << mm::drawImg(B_hat_img) << std::endl;

    mm::Image img_ero0 = mm::ero0(A, B_L);
```

```
    // Dualidade: (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);

    // Verificar igualdade pixel a pixel
    bool equal = true;
    for (int i = 0; i < (int)A.data.size(); i++) {
        if (ero_c.data[i] != dil_Ac.data[i]) {
            equal = false;
            break;
        }
    }
    std::cout << "Dualidade (A   B)  == A    B : " << (equal ? "True" : "False") << std::endl;

    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

Dualidade (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)")
```

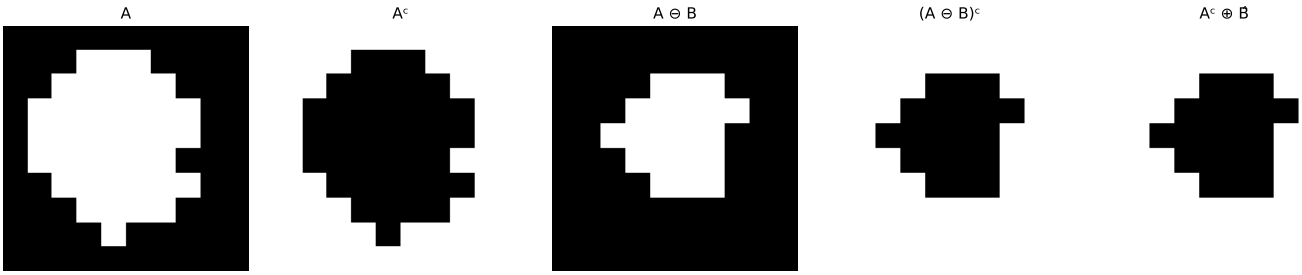


Figura 4.4: Erosão e dilatação em imagem binária 10×10 com elemento estruturante 'L' assimétrico 3×3. Validação da dualidade erosão–dilatação.

4.3.2 Apertura e Chiusura

Combinando erosione e dilatazione si ottengono due operatori di grande utilità pratica: l'**apertura** e la **chiusura**, definiti dalle Equazioni Equazione 4.5 e Equazione 4.6. I loro principali effetti sono riassunti nella Tabella 4.1.

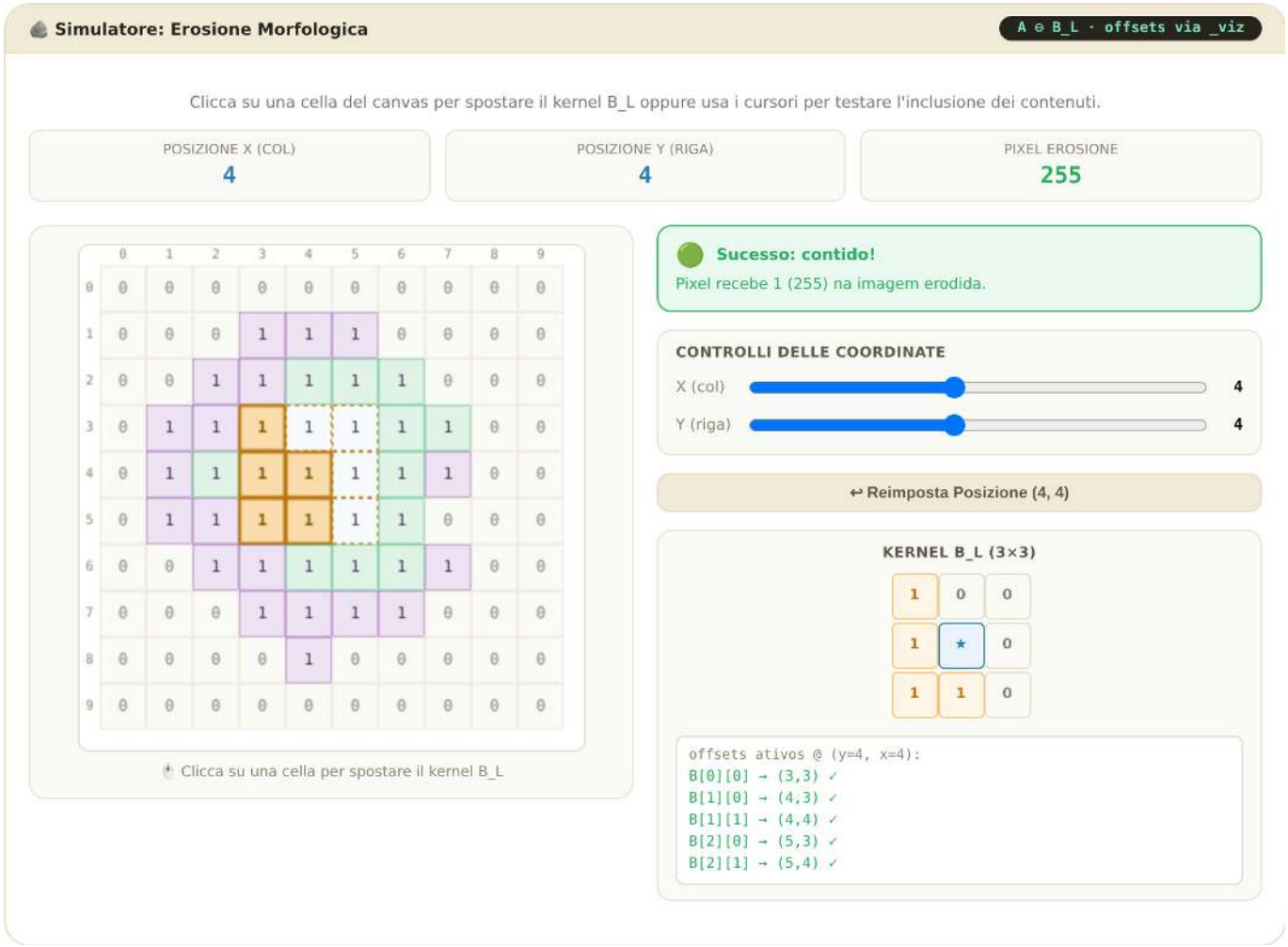
Apertura (*opening*) — erosione seguita da dilatazione con lo stesso B:

$$A \circ B = (A \ominus B) \oplus B \tag{4.5}$$

Chiusura (*closing*) — dilatazione seguita da erosione con lo stesso B:

$$A \bullet B = (A \oplus B) \ominus B \tag{4.6}$$

In pratica, `mm::open` e `mm::close` applicano lo stesso elemento strutturante nelle due fasi. Per elementi strutturanti simmetrici (i più comuni), questa implementazione coincide con la definizione matematica presentata sopra.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.it/cap04/sim-04-erosao.html

Figura 4.5: Simulatore: Erosione Morfologica (A B_L)

Tabella 4.1: Proprietà di apertura e chiusura.

Operatore	Sequenza	Effetto principale
Apertura $A \circ B$	erosione → dilatazione	Rimuove strutture incapaci di contenere l'elemento strutturante; smussa i contorni esterni
Chiusura $A \bullet B$	dilatazione → erosione	Riempie buchi più piccoli di B; smussa i contorni interni

Proprietà importante: entrambi sono **idempotenti**. Per esempio,

$$(A \circ B) \circ B = A \circ B,$$

cioè, dopo la prima applicazione, nuove applicazioni dello stesso operatore non modificano più il risultato.

4.3.2.1 Implementazione dell'apertura e della chiusura

A differenza dell'erosione e della dilatazione, l'apertura e la chiusura non introducono nuovi meccanismi computazionali. Entrambe sono ottenute tramite la composizione sequenziale degli operatori primitivi già presentati:

```
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 funzione `mm::open` delega l'operazione a `mm::open(f, B)`, mentre `mm::close` utilizza `mm::close(f, B)`, producendo lo stesso risultato in modo più efficiente.

L'apertura eredita dall'erosione la capacità di rimuovere strutture più piccole dell' elemento strutturante e dalla dilatazione il ripristino parziale delle regioni conservate. La chiusura realizza il processo inverso: prima espande gli oggetti e poi ripristina le loro dimensioni originali, colmando lacune e buchi più piccoli dell'elemento strutturante.

4.3.2.2 Filtro Sequenziale Alternato

Nella pratica, l'apertura e la chiusura vengono spesso applicate in sequenza per rimuovere simultaneamente il rumore esterno e riempire i fori interni. La funzione `mm::asf` (*Alternating Sequential Filter*) generalizza questa strategia applicando aperture e chiusure alternativamente con elementi strutturanti progressivamente più grandi. Le sequenze disponibili sono presentate nella Tabella 4.2.

Tabella 4.2: Sequenze del filtro sequenziale alternato mm.asf.

Sequenza	Ordine	Uso tipico
`'OC'`	apertura → chiusura	rimuove il rumore esterno prima di riempire piccoli fori
`'CO'`	chiusura → apertura	riempie piccoli fori prima di rimuovere il rumore esterno
`'OCO'`	apertura → chiusura → apertura	enfatizza la rimozione del rumore esterno
`'COC'`	chiusura → apertura → chiusura	enfatizza il riempimento di fori e lacune

Il parametro `n` controlla il numero di scale utilizzate dal filtro. In ogni iterazione i, l'elemento strutturante viene ampliato mediante somma di Minkowski (`mm::sesum(b, i)`), producendo una sequenza di filtri morfologici sempre più comprensivi. A differenza di una singola apertura o chiusura con un elemento strutturante grande,

l'ASF realizza una levigatura progressiva su più scale, preservando meglio la geometria degli oggetti rilevanti mentre elimina le strutture più piccole. La Figura 4.6 presenta un esempio di applicazione dell'apertura, della chiusura e dell'ASF sull'immagine delle monete.

```
%%writefile tmp/fig_04_open_close.cpp
#define MM_OUT "tmp/fig_04_open_close.png"
//| label: fig-04-open-close
//| fig-cap: "Abertura, fechamento, composicao e filtro sequencial alternado aplicados a
binarizacao Otsu das moedas (pre-processadas por realce de contraste). Elemento estruturante:
disco 13×13."
//| echo: true
//| output: true

#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_clahe = mm::_read_state("tmp/state/img_clahe_12.png");
// [pdi:state-io:end]

    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)")
```

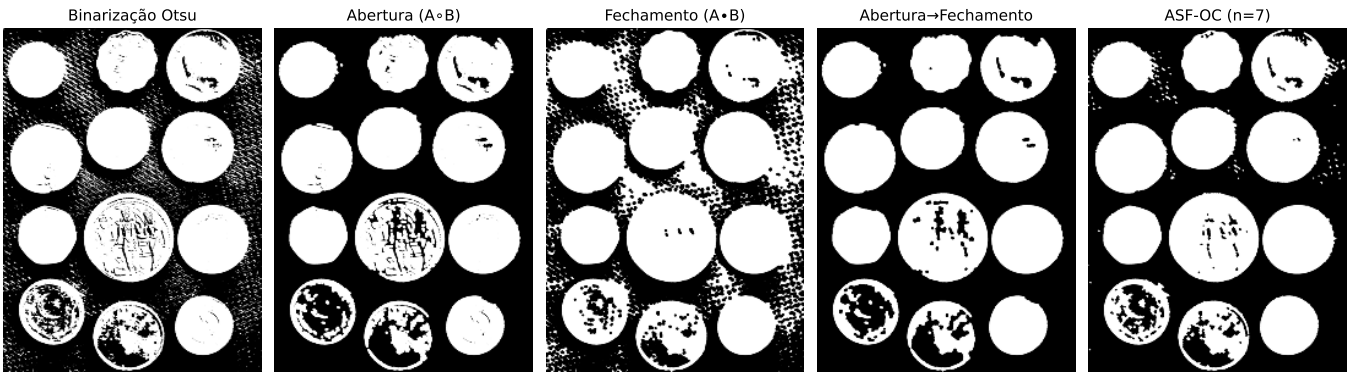


Figura 4.6: Abertura, fechamento, composição e filtro sequencial alternado aplicados à binarização Otsu das moedas (pré-processadas por realce de contraste). Elemento estruturante: disco 13×13.

4.3.3 Operatori Geodetici

Gli operatori geodetici introducono un vincolo aggiuntivo agli operatori morfologici classici tramite un'immagine di controllo denominata **maschera** g. Invece di consentire che l'erosione o la dilatazione si propaghino liberamente attraverso l'immagine, il risultato di ciascuna iterazione è limitato punto per punto dai valori della maschera, restringendo l'evoluzione dell'operazione alle regioni consentite.

4.3.3.1 Dilatazione Geodetica

La **dilatazione geodetica** di un'immagine marcatore f sotto un'immagine maschera g, utilizzando un elemento strutturante piano b, è definita da:

$$f \oplus_g b = (f \oplus b) \wedge g, \tag{4.7}$$

dove $\wedge$ rappresenta il minimo punto a punto.

In altre parole, si esegue inizialmente una dilatazione convenzionale sul marcatore e, successivamente, il risultato viene limitato dalla maschera g. In questo modo, la propagazione non può mai superare le regioni consentite dalla maschera.

La formulazione classica della dilatazione geodetica presuppone che il marcatore sia contenuto nella maschera, cioè $f \leq g$, garantendo che l'evoluzione dell'operazione rimanga sempre limitata dalla maschera.

4.3.3.2 Implementazione della dilatazione geodetica

La funzione `mm::cdil` implementa direttamente questo operatore e consente di eseguire più iterazioni consecutive:

```
def cdil(f, g, b=np.zeros((3,3),dtype='uint8'), n=1):
    """Dilatazione geodetica del marcatore f sotto la maschera g."""
    y = f.copy()
    for _ in range(n):
        y = np.minimum(mm.dil(y, b), g)
    return y
```

L'istruzione `np.minimum(mm::dil(y, b), g)` implementa esattamente la definizione matematica della dilatazione geodetica, cioè $(y \oplus b) \wedge g$.

Quando $n = 1$, la funzione esegue una singola dilatazione geodetica. Per $n > 1$, il risultato di ogni fase diventa il marcatore della fase successiva, producendo una propagazione progressiva controllata dalla maschera.

Il simulatore interattivo Figura 4.7 permette di seguire, passo dopo passo, la propagazione del marcatore f lungo i corridoi del labirinto. A ogni iterazione di `mm::cdil`, il fronte di dilatazione avanza verso le celle vicine libere — quelle in cui $g = 1$ —, mentre le pareti ($g = 0$) rimangono invalicabili. Il numero di passi necessari affinché il marcatore raggiunga l'uscita corrisponde esattamente alla lunghezza geodetica del percorso più breve all'interno della maschera, evidenziando la connessione diretta tra la dilatazione geodetica iterata e la nozione di distanza nei grafi.

4.3.3.3 Erosione Geodetica

In modo duale, l'**erosione geodetica** di un'immagine marcatore f sotto un'immagine maschera g è definita da:

$$f \ominus_g b = (f \ominus b) \vee g, \tag{4.8}$$

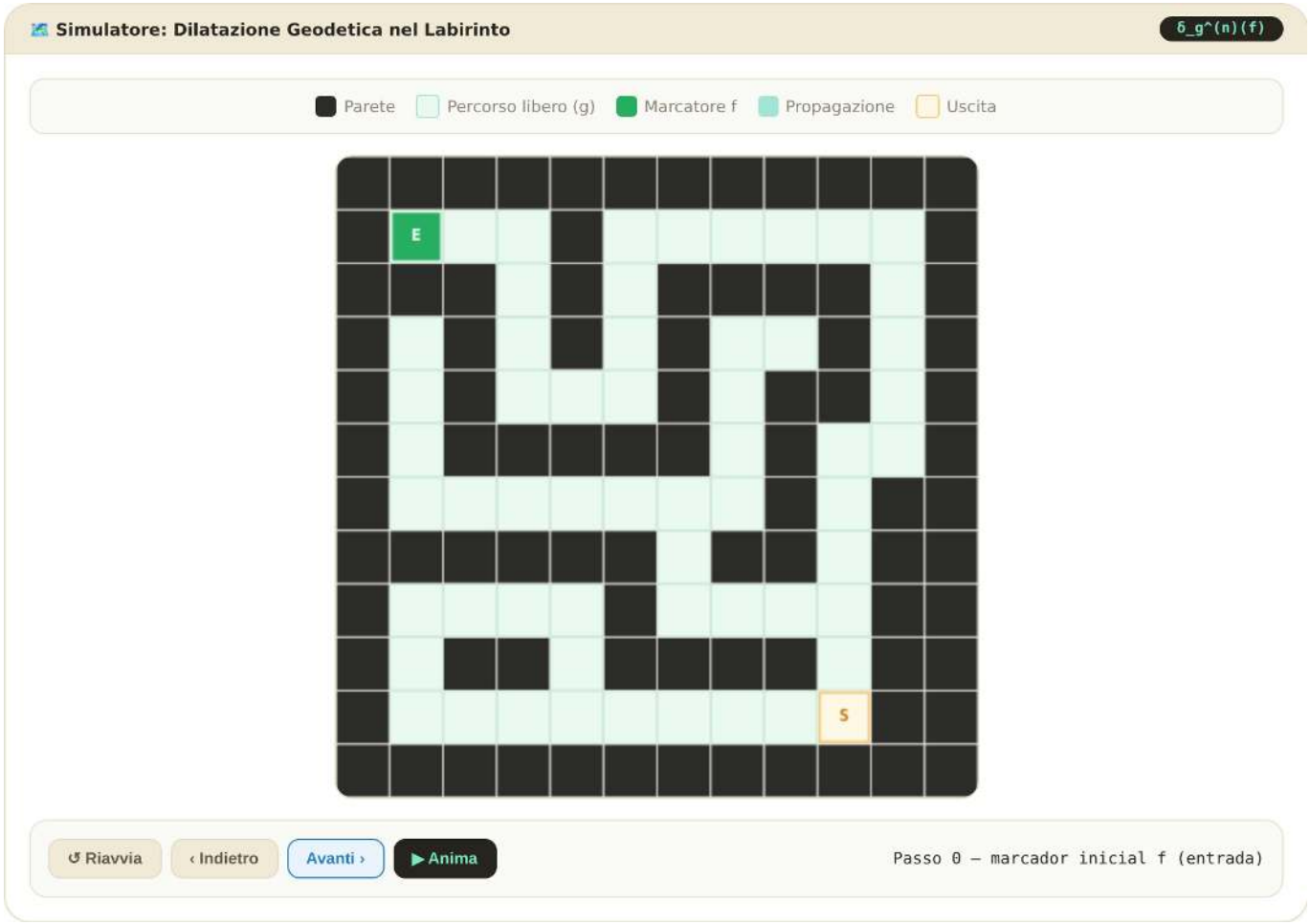
dove $\vee$ rappresenta l'operatore di massimo punto a punto.

In questo caso, l'erosione convenzionale del marcatore è seguita da una restrizione inferiore imposta dalla maschera. Così, nessun pixel del risultato può assumere un valore inferiore al corrispondente pixel della maschera.

La formulazione classica dell'erosione geodetica presuppone la condizione duale

$$f \geq g,$$

cosicché la maschera agisca come limite inferiore durante tutto il processo.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.it/cap04/sim-04-cdil.html

Figura 4.7: Simulatore interattivo di dilatazione geodetica: il marker f (verde) si propaga passo dopo passo attraverso i percorsi liberi della maschera g, senza attraversare pareti.

4.3.3.4 Implementazione dell'erosione geodetica

La funzione statica `mm::cero` materializza questo operatore:

```
staticmethod
def cero(f, g, b=np.zeros((3,3),dtype='uint8'), n=1):
    """Erosione geodetica del marcatore f sotto la maschera g."""
    y = f.copy()
    for _ in range(n):
        y = np.maximum(mm.ero(y, b), g)
    return y
```

L'istruzione `np.maximum(mm::ero(y, b), g)` implementa direttamente l'espressione $(y \ominus b) \vee g$.

Così come per la dilatazione geodetica, il parametro n definisce quante erosioni geodetiche successive saranno calcolate prima di restituire l'immagine finale.

> **i** Relazione con la ricostruzione morfologica
>
> La ricostruzione morfologica presentata nella prossima sezione è ottenuta mediante l'applicazione iterativa della dilatazione geodetica (`mm::cdil`) fino a raggiungere un punto fisso, cioè fino a quando nessuna cella cambia valore tra due iterazioni consecutive. In altre parole, la ricostruzione consiste in una sequenza di dilatazioni geodetiche successive che si propagano all'interno della maschera fino a quando non ci sono più modifiche.
> In modo duale, è anche possibile definire ricostruzioni basate sull'erosione geodetica tramite applicazioni successive di `mm::cero`.

4.3.3.5 Esempio: propagazione in un labirinto tramite dualità

La Figura 4.8 illustra la risoluzione del problema di connettività di un labirinto utilizzando la dualità morfologica tramite le funzioni `mm::cero` e `mm::suprec`.

Invece di propagare un marcatore attraverso i corridoi liberi mediante dilatazioni geodetiche, il problema viene formulato nel dominio complementare. Inizialmente, la maschera originale viene invertita,

$$g = 1 - g_{orig},$$

così che le pareti assumano valore 1 e i corridoi valore 0. In modo analogo, il marcatore viene costruito in questo stesso dominio complementare, contenente un unico valore 0 nella posizione di ingresso del labirinto e valore 1 nei restanti pixel.

Utilizzando l'elemento strutturante a croce (`mm::secross()`), l'erosione geodetica agisce sul marcatore complementato. A ogni iterazione di `mm::cero`, la regione connessa al marcatore iniziale subisce erosioni successive, mentre la maschera impone un limite inferiore che impedisce la propagazione attraverso le pareti del labirinto.

Le immagini intermedie mostrano stati dell'evoluzione dopo diversi numeri di iterazioni (`n=5`, `n=12` e `n=22`). Il risultato finale si ottiene mediante la ricostruzione geodetica per erosione (`mm::suprec`), che applica erosioni geodetiche successive fino a raggiungere un punto fisso, cioè una situazione in cui non si verifica alcuna ulteriore modifica tra due iterazioni consecutive.

Nel dominio complementare, la regione ricostruita corrisponde esattamente all'insieme di corridoi connessi all'ingresso del labirinto. Pertanto, la connettività tra ingresso e uscita può essere determinata direttamente a partire dall'immagine ricostruita.

```
%%writefile tmp/fig_04_cero_labirinto.cpp
#define MM_OUT "tmp/fig_04_cero_labirinto.png"
#include "morph.hpp"
```

```
#include <iostream>
#include <vector>
#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) = 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)")
```

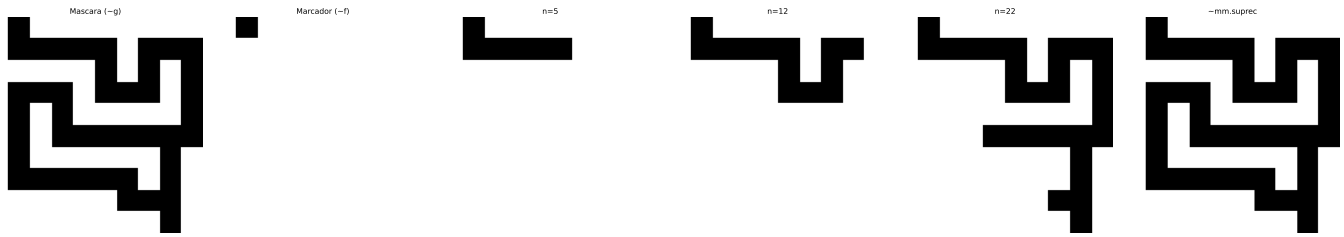


Figura 4.8: Resolução de labirinto no domínio complementar: com máscara e marcador invertidos, *mm::cero* e *mm::suprec* propagam a onda geodésica pelos corredores.

4.3.4 Ricostruzione Morfologica

La **ricostruzione morfologica** propaga un'immagine **marcatore** f all'interno di un'immagine **maschera** g, garantendo che il risultato non superi mai i valori di intensità imposti dalla maschera. L'operatore fondamentale che rende possibile questa propagazione contenuta è la **dilatazione geodetica**, definita dalla Equazione 4.7.

La ricostruzione si ottiene applicando in modo iterativo questa dilatazione condizionata. Inizialmente, il marcatore è limitato dalla maschera per stabilire lo stato iniziale:

$$X^{(0)} = f \wedge g,$$

e le iterazioni successive sono definite in modo ricorsivo da:

$$X^{(k)} = (X^{(k-1)} \oplus b) \wedge g.$$

La sequenza cresce monotonamente fino a raggiungere un punto fisso, producendo la **ricostruzione morfologica per dilatazione** (nota anche in letteratura come *inf-ricostruzione*):

$$R_g^{\delta}(f) = \lim_{k \to \infty} X^{(k)} = X^{(k)} \quad \text{quando} \quad X^{(k)} = X^{(k-1)}. \tag{4.9}$$

La salita iterativa viene interrotta non appena si raggiunge la stabilità, cioè quando due iterazioni consecutive producono matrici con valori assolutamente identici.

4.3.4.1 Implementazione della ricostruzione morfologica

La routine didattica `mm::infrec` implementa direttamente l'algoritmo iterativo a punto fisso. Inizialmente, il marker effettivo iniziale $X^{(0)}$ è determinato tramite l'operazione `np.minimum(f, g)`. Per garantire che il ciclo di verifica esegua la prima iterazione senza innescare false convergenze premature, la variabile di controllo dell'iterazione precedente (`y1`) è inizializzata con un valore sentinella al di fuori del dominio dei dati (o semplicemente con una matrice che forzi la prima esecuzione).

```
def infrec(f, g, b=np.zeros((3,3), dtype='uint8')):
    """Inf-ricostruzione: dilata il marker (f   g) fino a convergenza sotto la maschera g."""
    y = np.minimum(f, g)
    # Inizializza y1 con valori impossibili per forzare l'ingresso nel ciclo
    y1 = np.full_like(f, 256, dtype=np.int16)
    while not np.array_equal(y, y1):
        y1 = y.copy()
        # Applica la dilatazione geodetica: (y   b)   g
        y = np.minimum(mm.dil(y, b), g)
    return y.astype('uint8')
```

All'interno del ciclo `while`, la variabile `y` memorizza la stima corrente della ricostruzione $X^{(k)}$, mentre `y1` preserva l'immagine dello stadio immediatamente precedente $X^{(k-1)}$. L'istruzione di controllo condizionale `np.minimum(mm::dil(y, b), g)` traduce fedelmente la dilatazione geodetica teorica, in cui l'espansione morfologica convenzionale comandata da OpenCV viene immediatamente "potata" e limitata dalle barriere di intensità della maschera g. Il ciclo termina quando nessuna modifica dei pixel viene registrata tra i passi.

4.3.4.2 Vantaggi della ricostruzione morfologica

La ricostruzione morfologica è significativamente più robusta dell'apertura convenzionale perché è in grado di rimuovere strutture indesiderate senza distorcere o alterare la morfologia degli oggetti che devono essere preservati.

Mentre l'apertura classica smussa gli angoli, elimina le punte e deforma i contorni a causa dell'imposizione geometrica rigida dell'elemento strutturante, la ricostruzione geodesica utilizza la maschera per recuperare con precisione i limiti e le forme originali degli oggetti che presentano connettività con il marcatore iniziale.

In termini intuitivi, il marcatore agisce come un seme di contagio che si espande progressivamente, ma solo percorrendo le regioni consentite dalla maschera. I componenti che non presentano alcuna intersezione con il marcatore non verranno mai ricostruiti (venendo eliminati), mentre i componenti toccati dal seme si espandono fino a ripristinare integralmente la loro geometria originale.

Questo comportamento discriminatorio e conservativo è illustrato nella Figura 4.9, utilizzando l'elemento strutturante a croce presentato nella Figura 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>

int main() {
    mm::Image g(10, 10);
    int vals_g[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) = (unsigned char)(vals_g[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);

    // Verifica estabilidade comparando pixel a pixel
    bool igual = true;
    for (int i = 0; i < (int)img_reconstruida.data.size(); i++) {
        if (img_reconstruida.data[i] != iteracoes[iteracoes.size()-1].data[i]) {
            igual = false;
            break;
        }
    }
    std::cout << "  Estabilidade na iteração 5: " << (igual ? "True" : "False") << "\n";

    std::vector<mm::Image> todas = {g, f};
    todas.insert(todas.end(), iteracoes.begin(), iteracoes.end());
    todas.push_back(img_reconstruida);

    std::vector<std::string> nomes = {"Máscara (g)", "Marcador (f)"};
    nomes.insert(nomes.end(), titulos.begin(), titulos.end());
    nomes.push_back("Reconstrução R_g(f)");

    mm::show(todas, MM_OUT, nomes, 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)")
```

Figura 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 Riempimento di Buchi e Rimozione dei Bordi

Due operatori basati sulla ricostruzione morfologica completano la *pipeline* di pulizia binaria. Le loro caratteristiche principali sono riassunte nella Tabella 4.3.

Riempimento di buchi (`mm::clohole`) rimuove cavità completamente circondate dall'oggetto, indipendentemente dalla dimensione, senza alterare i contorni esterni. La procedura agisce sul complemento dell'immagine, utilizzando come marcatore una restrizione della cornice (*frame*) allo sfondo:

$$\text{clohole}(f) = \left(R^{\delta}_{f^c}(\text{frame}(f) \wedge f^c)\right)^c \tag{4.10}$$

In termini operativi, si ricostruisce prima lo sfondo esterno e, successivamente, si applica la complementazione per recuperare gli oggetti con i buchi riempiti.

Rimozione degli oggetti di bordo (`mm::edgeoff`) elimina tutti gli oggetti che toccano il bordo dell'immagine, preservando solo i componenti completamente interni. Il marcatore è ottenuto dall'intersezione tra la cornice (*frame*) e gli oggetti dell'immagine:

$$\text{edgeoff}(f) = f \setminus R^{\delta}_{f}(\text{frame}(f) \wedge f) \tag{4.11}$$

Tabella 4.3: Confronto tra gli operatori *clohole* e *edgeoff*.

Operatore	Marcatore	Maschera	Effetto
`mm::clohole`	*frame* ristretto allo sfondo (f^c)	f^c	Riempie i buchi interni
`mm::edgeoff`	*frame* ristretto all'oggetto (f)	f	Rimuove gli oggetti connessi al bordo

L'evoluzione passo passo di queste trasformazioni geodetiche può essere seguita nelle figure successive. La Figura 4.10 illustra il meccanismo di inondazione controllata dell'operatore `mm::clohole`, nel quale la ricostruzione avviene a partire dallo sfondo esterno e impedisce la propagazione verso regioni interne non connesse all'esterno, risultando nel riempimento consistente delle cavità interne. Al contrario, la Figura 4.11 dettaglia la dinamica dell'operatore `mm::edgeoff`, in cui solo i componenti connessi al bordo vengono ricostruiti e successivamente rimossi, preservando esclusivamente gli oggetti completamente contenuti all'interno dell'immagine.

La connettività della propagazione geodetica è controllata dall'elemento strutturante: `mm::sebox()` (vicinanza di 8) include connessioni diagonali, mentre `mm::secross()` (vicinanza di 4) le esclude. Di conseguenza, la scelta dell'elemento strutturante influisce su quali componenti vengono raggiunti dalla ricostruzione e, quindi, su quali verranno preservati o rimossi.

4.3.5.1 Conformità con l'implementazione

Le definizioni sopra sono direttamente allineate con l'implementazione in `morph.py`, riprodotta di seguito:

```
staticmethod
def clohole(f, b=np.ones((3,3),dtype='uint8')):
    # marcador restrito ao fundo da imagem
    marcador = mm.frame(f, border=1) & mm.neg(f)
    return mm.neg(mm.infrec(marcador, mm.neg(f), b))

staticmethod
def edgeoff(f, b=np.ones((3,3),dtype='uint8')):
    # marcador restrito aos objetos da imagem
    marcador = mm.frame(f, border=1) & f
    return mm.subm(f, mm.infrec(marcador, f, b))
```

Queste implementazioni rendono esplicito che entrambi gli operatori sono istanze dirette di ricostruzione morfologica per dilatazione geodetica con `mm::infrec`, differendo solo nella scelta del marcatore e della maschera: `clohole` opera sul complemento dell'immagine, mentre `edgeoff` opera direttamente sul dominio degli oggetti.

```
%%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>
#include <vector>

int main() {
    mm::Image f(10, 10);
```

```
    // Initialize f with the given pattern * 255
    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));
    std::cout << "  Validação clohole: " << (std::equal(f.data.begin(), f.data.end(),
mm::clohole(f).data.begin()) ? "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: False
[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)
```

```python
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)")
```

Figura 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"
// Compile: g++ -std=c++17 -o program program.cpp -I. $(pkg-config --cflags --libs opencv4)
-DMM_USE_OPENCV -lmm

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

int main() {
    //#| label: fig-04-edgeoff-didatico
    //#| fig-cap: "*Pipeline* di eliminazione delle strutture di bordo (*edgeoff*): il marker
    cattura le radici connesse alle estremità, la ricostruzione delimita questi elementi e la
    sottrazione preserva solo gli oggetti totalmente interni."
    //#| echo: true
    //#| output: true

    mm::Image f(10, 10);
    unsigned char f_data[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) = f_data[y][x] * 255;
        }
    }
```

```
    mm::Image B_box = mm::sebox();
    mm::Image marcador_eo = mm::band(mm::frame(f, 1), f);   // bordo   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 originale", "Marcatore (bordo)"};
    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 originale
[2] Marcatore (bordo)
[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)")
```

Figura 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* di Pulizia Binaria con CLAHE

Sulla base dell'analisi precedente — in cui il CLAHE ha prodotto il valore più alto della varianza interclassi ($\sigma_B^2 \approx 2{,}47 \times 10^3$), vedi Figura 4.2, e l'operatore `mm::clohole` si è mostrato efficace nel riempimento delle cavità interne —, il *pipeline* finale di segmentazione, illustrato in Figura 4.12, è strutturato dal seguente flusso computazionale:

$$\text{gray} \xrightarrow{\text{CLAHE}} \xrightarrow{\text{Otsu}} \xrightarrow{\text{open}} \xrightarrow{\text{clohole}} \xrightarrow{\text{open}} \xrightarrow{\text{edgeoff}} \text{segmentazione}$$

Dopo la fase di `mm::clohole`, si applica una seconda apertura morfologica con un elemento strutturante più grande (`mm::sedisk(33)`, disco di diametro 33). Questa operazione rimuove piccole regioni residue e artefatti che potrebbero essere rimasti dopo la segmentazione. In particolare, il riempimento geodetico può trasformare piccole cavità isolate in componenti connesse all'oggetto, rendendo conveniente un'ulteriore fase di filtraggio basata sulla dimensione. Il diametro è stato scelto in modo che le monete rimangano in grado di contenere l'elemento strutturante, mentre componenti significativamente più piccole vengano eliminate.

In questa immagine, nessuna moneta è connessa al bordo della matrice. Di conseguenza, l'applicazione di `mm::edgeoff` non altera il risultato ottenuto dopo la seconda apertura. Tuttavia, questa fase è mantenuta nel *pipeline* per robustezza, poiché in altre immagini possono esistere oggetti parzialmente visibili o connessi ai bordi, che devono essere rimossi prima della fase di analisi.

> **Perché l'apertura dopo il clohole?**
>
> L'operatore `mm::clohole` riempie tutte le cavità chiuse presenti negli oggetti segmentati. In alcune situazioni, piccole regioni indesiderate possono rimanere dopo questa fase o diventare connesse agli oggetti principali. L'apertura morfologica successiva rimuove componenti più piccole dell'elemento strutturante, preservando le monete grazie alla loro dimensione significativamente maggiore.

```
%%writefile tmp/fig_04_pipeline_clahe.cpp
#define MM_OUT "tmp/fig_04_pipeline_clahe.png"
#include "morph.hpp"
#include <iostream>
#include <filesystem>

//| 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

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)
```

```
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)")
```

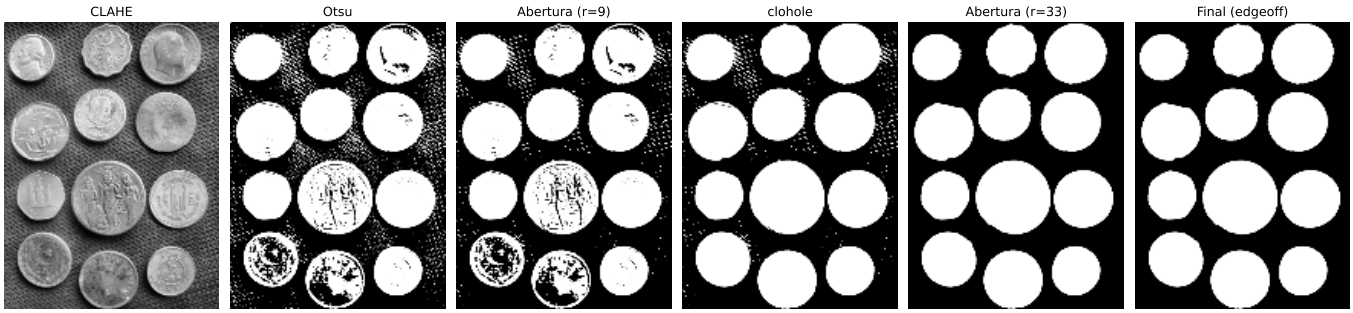


Figura 4.12: *Pipeline* completo de segmentação com CLAHE: Otsu → abertura (r=9) → clohole → abertura (r=33) → edgeoff.

4.3.7 Morfologia in Toni di Grigio

Gli operatori morfologici si estendono naturalmente alle immagini in toni di grigio. In questa formulazione, l'erosione e la dilatazione agiscono direttamente sui livelli di intensità dell'immagine. Per elementi strutturanti piatti ($b \equiv 0$), l'erosione corrisponde al **minimo locale** e la dilatazione al **massimo locale** all'interno del vicinato definito dall'elemento strutturante.

L'interpretazione intuitiva è semplice: l'erosione **scurisce** le regioni sostituendo ogni pixel con il valore più piccolo presente nel suo vicinato, mentre la dilatazione **schiarisce** le regioni utilizzando il valore più grande disponibile. La combinazione di questi operatori consente di costruire trasformazioni in grado di evidenziare i bordi, rimuovere le tendenze di illuminazione e mettere in risalto le strutture locali.

Tre operatori derivati sono particolarmente utili:

Gradiente morfologico — evidenzia i bordi come differenza tra dilatazione ed erosione:

$$\mathrm{grad}_B(f) = (f \oplus B) - (f \ominus B) \tag{4.12}$$

Top-hat — evidenzia le strutture brillanti più piccole dell'elemento strutturante (differenza tra l'immagine originale e la sua apertura):

$$\text{top-hat}_B(f) = f - (f \circ B) \tag{4.13}$$

Black-hat — evidenzia le strutture scure più piccole dell'elemento strutturante (differenza tra la chiusura e l'immagine originale):

$$\text{black-hat}_B(f) = (f \bullet B) - f \tag{4.14}$$

Il *Top-hat* estrae i dettagli brillanti che non sopravvivono all'apertura, mentre il *Black-hat* evidenzia i dettagli scuri rimossi dalla chiusura. Il gradiente morfologico, invece, mette in risalto le transizioni brusche di intensità, producendo una rappresentazione simile a quella di un rivelatore di bordi.

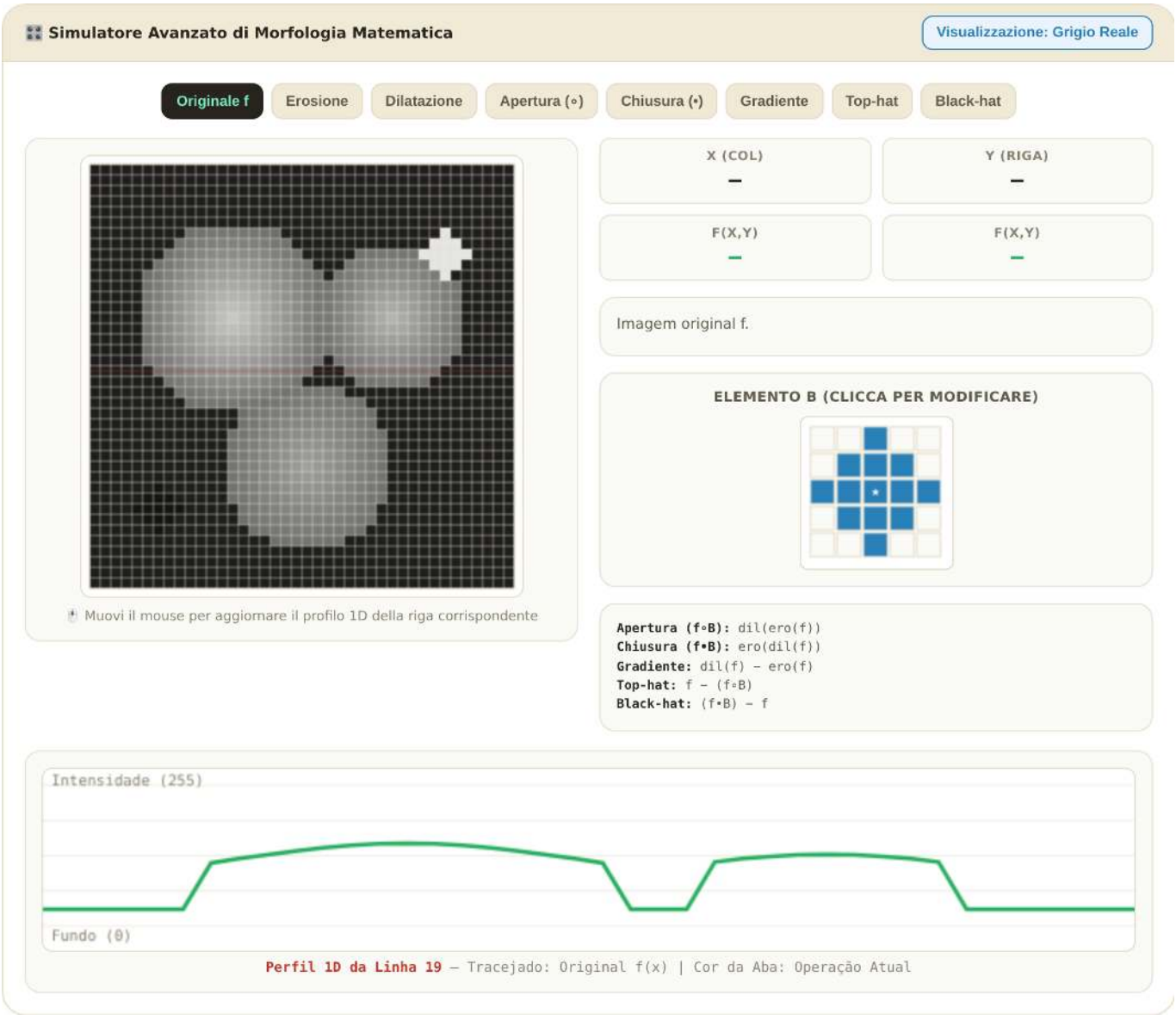
Per comprendere il meccanismo di questi operatori a livello locale, la Figura 4.13 presenta un simulatore interattivo di morfologia in toni di grigio. Il simulatore consente di modificare liberamente l'elemento strutturante, visualizzare il suo spostamento sull'immagine e seguire simultaneamente il profilo unidimensionale delle intensità. In questo modo, diventa possibile osservare direttamente come l'erosione seleziona i minimi locali, come la dilatazione seleziona i massimi locali e come il gradiente morfologico emerge dalla differenza tra questi due operatori.

Il pulsante situato nell'angolo superiore destro consente di alternare tra la visualizzazione originale in toni di grigio e una rappresentazione pseudocolorata (*colormap*) esclusivamente per i tre tipi di gradiente. La versione colorata facilita la percezione visiva delle variazioni di intensità, rendendo più evidente l'azione degli operatori morfologici su massimi, minimi e transizioni locali dell'immagine.

Gli operatori presentati sono disponibili in `morph.py` tramite le funzioni `mm::gradm`, `mm::tophat` e `mm::blackhat`.

La Figura 4.14 illustra gli effetti di questi operatori sull'immagine delle monete e sui rispettivi istogrammi. Si osservi che l'erosione sposta la distribuzione verso intensità più basse, mentre la dilatazione la sposta verso intensità più alte. Il gradiente concentra i valori nelle regioni di contorno, e gli operatori *Top-hat* e *Black-hat* producono istogrammi fortemente concentrati su bassi livelli di intensità, poiché vengono evidenziate solo piccole strutture locali.

```
%%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>

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::SE B = mm::sedisk(19);  // Elemento estruturante: disco de diâmetro 19

    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);

    std::vector<mm::Image> imagens = {
```



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.it/cap04/sim-04-operadores-av.html

Figura 4.13: Simulatore interattivo avanzato di morfologia con elemento strutturante modificabile e profilo 1D.

```
        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)
    };

    std::vector<std::string> titulos = {
        "Original", "Hist", "Erosão", "Hist", "Dilatação", "Hist",
        "Gradiente", "Hist", "Top-hat", "Hist", "Black-hat", "Hist"
    };

    mm::show(imagens, MM_OUT, titulos, 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)")
```

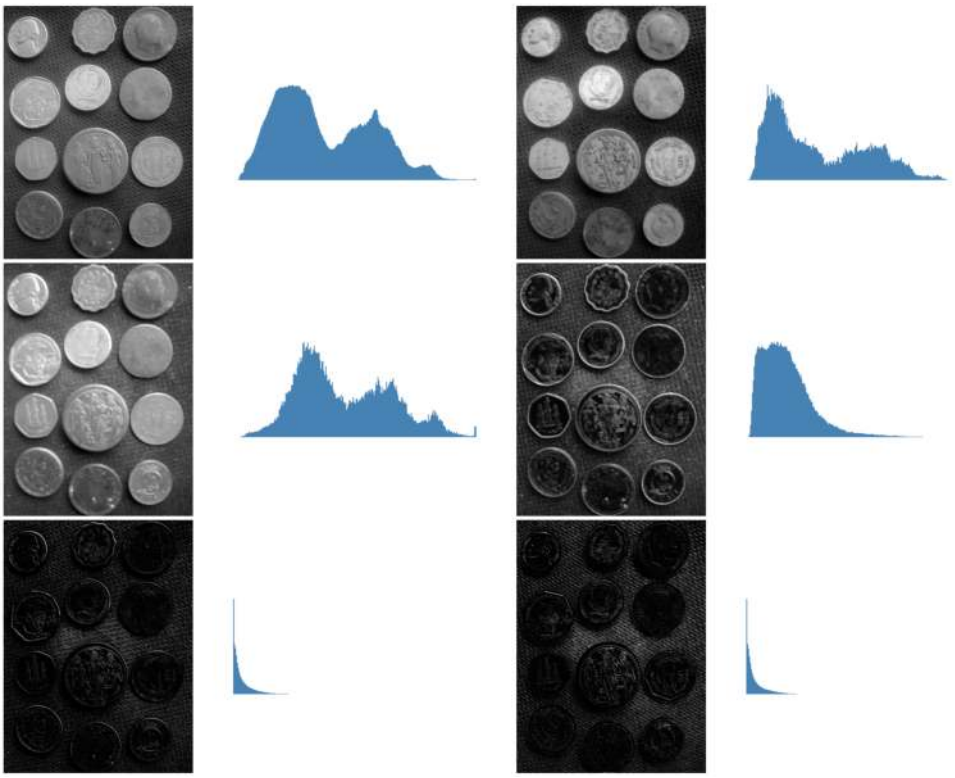


Figura 4.14: Morfologia em tons de cinza e seus histogramas: erosão, dilatação, gradiente, top-hat e black-hat. Elemento estruturante: disco de diâmetro 19.

Gli operatori morfologici presentati in precedenza saranno ora utilizzati come strumenti di raffinamento e generazione di marcatori per metodi di segmentazione più avanzati, presentati di seguito.

4.4 Segmentazione delle Immagini: Fondamenti e Tassonomia

La **segmentazione delle immagini** consiste nel dividere l'immagine in regioni associate a oggetti o strutture di interesse. In elaborazione digitale delle immagini (EDI), essa rappresenta la transizione tra l'elaborazione di basso livello — come filtraggio e miglioramento — e fasi di analisi più avanzate, come l'estrazione di caratteristiche, il riconoscimento e l'interpretazione della scena.

Formalmente, l'obiettivo della segmentazione consiste nel decomporre il dominio spaziale completo di un'immagine, denotato da Ω, in una partizione di sottoinsiemi $\{R_1, R_2, \dots, R_n\}$ che soddisfi simultaneamente i criteri di **completezza** e **disgiunzione**:

$$\bigcup_{i=1}^{n} R_i = \Omega, \qquad R_i \cap R_j = \emptyset \quad \forall\, i \neq j \tag{4.15}$$

Oltre alle proprietà di completezza e disgiunzione espresse nella Equazione 4.15, ciascuna sotto-regione R_i deve costituire un dominio **omogeneo** secondo un predicato di similarità definito su proprietà locali — intensità, colore o texture — e, simultaneamente, essere **distinta** dalle regioni adiacenti.

Le tecniche di segmentazione possono essere organizzate in diverse famiglie. In questo capitolo verranno enfatizzati gli approcci riassunti nella Tabella 4.4, fondati principalmente su criteri di intensità, connettività e prossimità spaziale.

Tabella 4.4: Tassonomia semplificata dei principali approcci di segmentazione e raffinamento studiati in questo capitolo.

Approccio	Criterio di Segmentazione	Operatori di Riferimento
Sogliatura	Partizionamento dello spazio delle intensità	Criterio di Otsu, sogliatura globale e locale
Morfologia Matematica	Relazioni spaziali definite da elementi/funzioni strutturanti	Erosione, dilatazione, apertura, chiusura e ricostruzione
Basata su Regioni	Omogeneità locale e connettività spaziale	Etichettatura delle componenti connesse, Trasformata della Distanza e *Watershed*

Fino a questo punto, lo sviluppo pratico si è concentrato sulla **sogliatura**, mediante la combinazione tra equalizzazione adattiva CLAHE e il metodo globale di Otsu. Questa fase è stata integrata da operatori di **ricostruzione morfologica** basati su dilatazioni geodetiche, implementati dalle funzioni `mm::infrec`, `mm::clohole` e `mm::edgeoff`, producendo una maschera binaria pulita e adeguata all'analisi.

Tuttavia, in scenari in cui oggetti distinti appaiono connessi nella maschera binaria — sia per contatto fisico, sovrapposizione parziale o per stretti ponti di pixel prodotti dalla segmentazione —, la sogliatura non è più sufficiente per individualizzare ciascun oggetto. In tali casi, molteplici oggetti finiscono per comporre un'unica componente connessa, rendendo difficili le fasi successive di misurazione e interpretazione.

Per superare questa limitazione, le prossime sezioni introducono tre strumenti complementari: l'**Etichettatura delle Componenti Connesse**, la **Trasformata della Distanza** e l'algoritmo di segmentazione ***Watershed*** basato su marcatori. Nell'insieme, queste tecniche consentono di separare oggetti adiacenti, identificare le regioni individualmente ed estrarre descrittori geometrici coerenti per l'analisi quantitativa.

4.4.1 Etichettatura

L'**etichettatura delle componenti connesse** (*connected component labeling*) è l'operatore che assegna un identificatore intero unico a ciascun insieme di pixel appartenenti alla stessa componente connessa in un'immagine binaria.

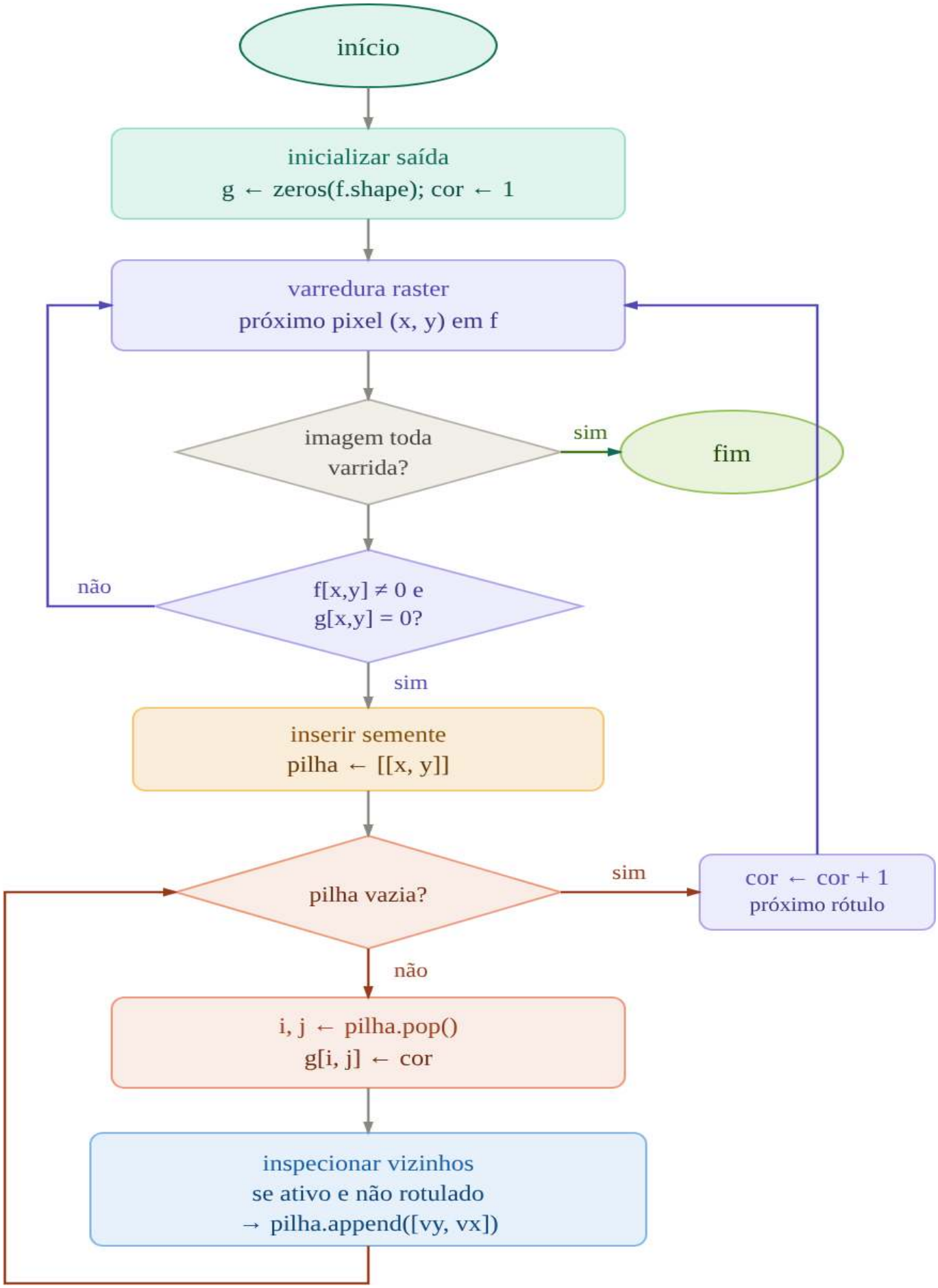
> **i** Definizione formale
>
> Data un'immagine binaria f e una relazione di connettività definita da un elemento strutturante B (tipicamente connettività-4 o connettività-8), l'algoritmo di etichettatura della Figura 4.15 produce un'immagine g nella quale tutti i pixel appartenenti alla stessa componente connessa ricevono la stessa etichetta intera positiva, mentre pixel appartenenti a componenti distinte ricevono etichette diverse.

La connettività definisce quali pixel sono considerati vicini diretti di un pixel (x, y). Le definizioni più utilizzate sono:

- **Connettività-4:** considera solo i quattro vicini ortogonali (nord, sud, est e ovest).
- **Connettività-8:** considera i quattro vicini ortogonali e i quattro diagonali, per un totale di otto vicini.

La scelta della connettività influenza direttamente la formazione delle componenti connesse e, di conseguenza, il risultato dell'etichettatura, come illustrato nella Figura 4.17. Un esempio aggiuntivo può essere esplorato interattivamente nel simulatore presentato nella Figura 4.16.

L'implementazione in `morph.py` fornisce due versioni di questo operatore. La funzione `mm::label0` riproduce esplicitamente l'algoritmo di *flood-fill* utilizzando una pila e consente di controllare la connettività tramite l'elemento strutturante adottato. Invece, `mm::label` delega l'operazione all'implementazione ottimizzata di OpenCV (`mm::label0`). In entrambi i casi, il risultato è un'immagine etichettata nella quale ciascuna componente connessa riceve un identificatore intero distinto.

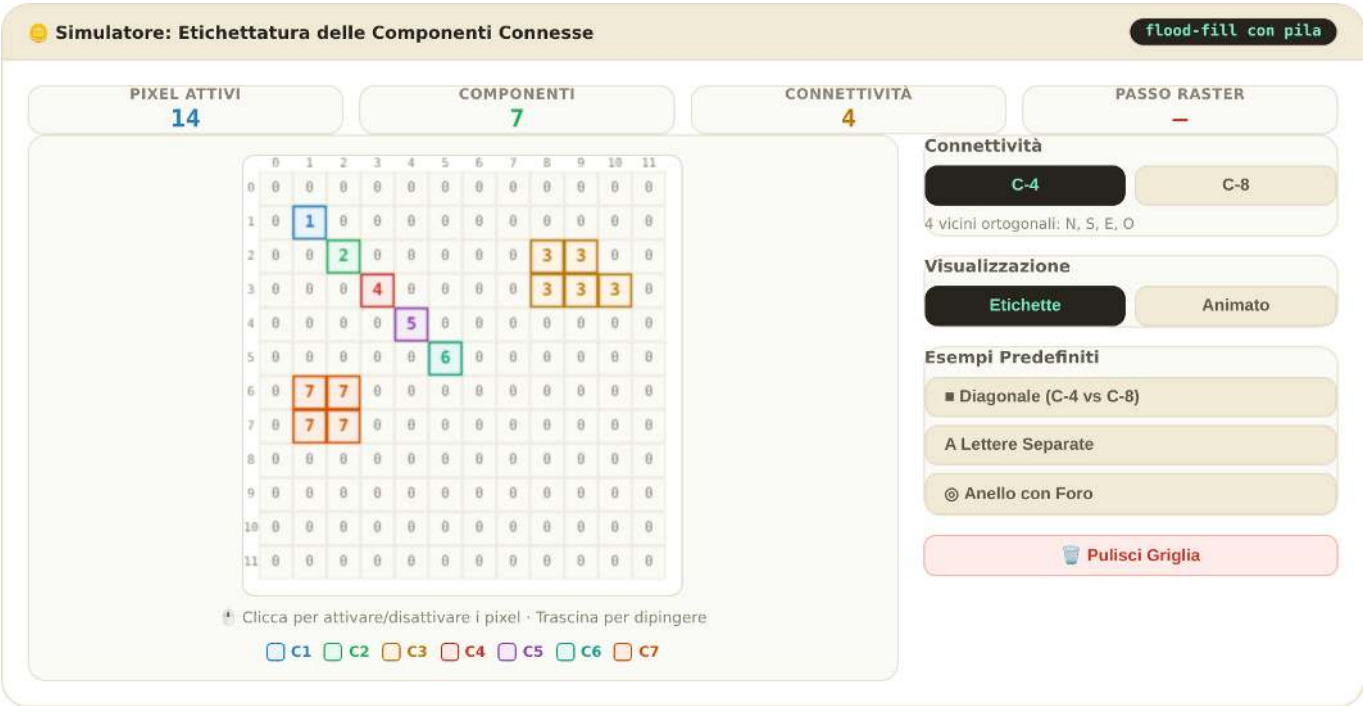


Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.it/cap04/sim-alg-rotulagem2.html

Figura 4.15: Algoritmo di etichettatura per *flood-fill* con pila.

L'helper `_viz` itera sulla finestra strutturante `b` centrata in (i, j), generando solo i vicini validi entro i limiti dell'immagine — la connettività desiderata è interamente determinata dalla forma di `b` passata all'algoritmo.

Esempio didattico — effetto della connettività:



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.it/cap04/sim-04-rotulacao.html

Figura 4.16: Simulatore interattivo di etichettatura delle componenti connesse (*connected component labeling*): visualizzazione dell'espansione flood-fill, connettività-4 e connettività-8.

```
%%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>

int main() {
    mm::Image f(10, 10);
    // Initialize f with the pattern (scaled by 255)
    int pattern[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) = pattern[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);

    // Find maximum values
    int n4 = 0, n8 = 0;
    for (int i = 0; i < lbl4.data.size(); i++) {
        if ((int)lbl4.data[i] > n4) n4 = (int)lbl4.data[i];
        if ((int)lbl8.data[i] > n8) n8 = (int)lbl8.data[i];
    }
    std::cout << "Componentes C4: " << n4 << "  |  C8: " << n8 << "\n";

    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) = (int)(lbl.at(y, x) * 255 / mx);
            }
        }
        return out;
    };

    std::vector<mm::Image> imgs = {
        f,
        norm_label(lbl4, std::max(n4, 1)),
        norm_label(lbl8, std::max(n8, 1))
    };
    std::vector<std::string> titles = {
        "f original",
        "C4 (" + std::to_string(n4) + " comp.)",
        "C8 (" + std::to_string(n8) + " comp.)"
    };
    mm::show(imgs, MM_OUT, titles, 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)")
```

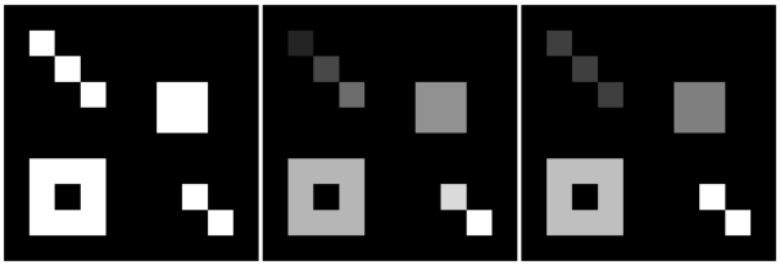


Figura 4.17: Efeito da conectividade na rotulação (*mm::label0*): pixels diagonalmente adjacentes formam componentes distintas em conectividade-4 e se fundem em conectividade-8.

4.4.2 Trasformata della Distanza

La **Trasformata della Distanza** (TD) è un operatore che, applicato a un'immagine binaria f, produce un'immagine in livelli di grigio D nella quale a ciascun pixel appartenente all'oggetto ($f(x, y) \neq 0$) viene assegnato come valore la distanza geometrica fino al pixel di sfondo ($f(x', y') = 0$) più vicino:

$$D(x, y) = \min_{(x', y') \,:\, f(x', y') = 0} d((x, y), (x', y'))$$

dove $d(\cdot, \cdot)$ è una metrica di distanza — tipicamente la distanza Euclidea (L_2). Il risultato è una rappresentazione topografica degli oggetti: i pixel situati all'interno assumono valori elevati, mentre i pixel vicini ai bordi presentano bassi valori di distanza. I **massimi locali** di D corrispondono ai punti più lontani dal bordo dell'oggetto, spesso vicini ai suoi centri geometrici o ai centri di massima inscrizione — proprietà particolarmente utile per la generazione automatica di marcatori nell'algoritmo *watershed*.

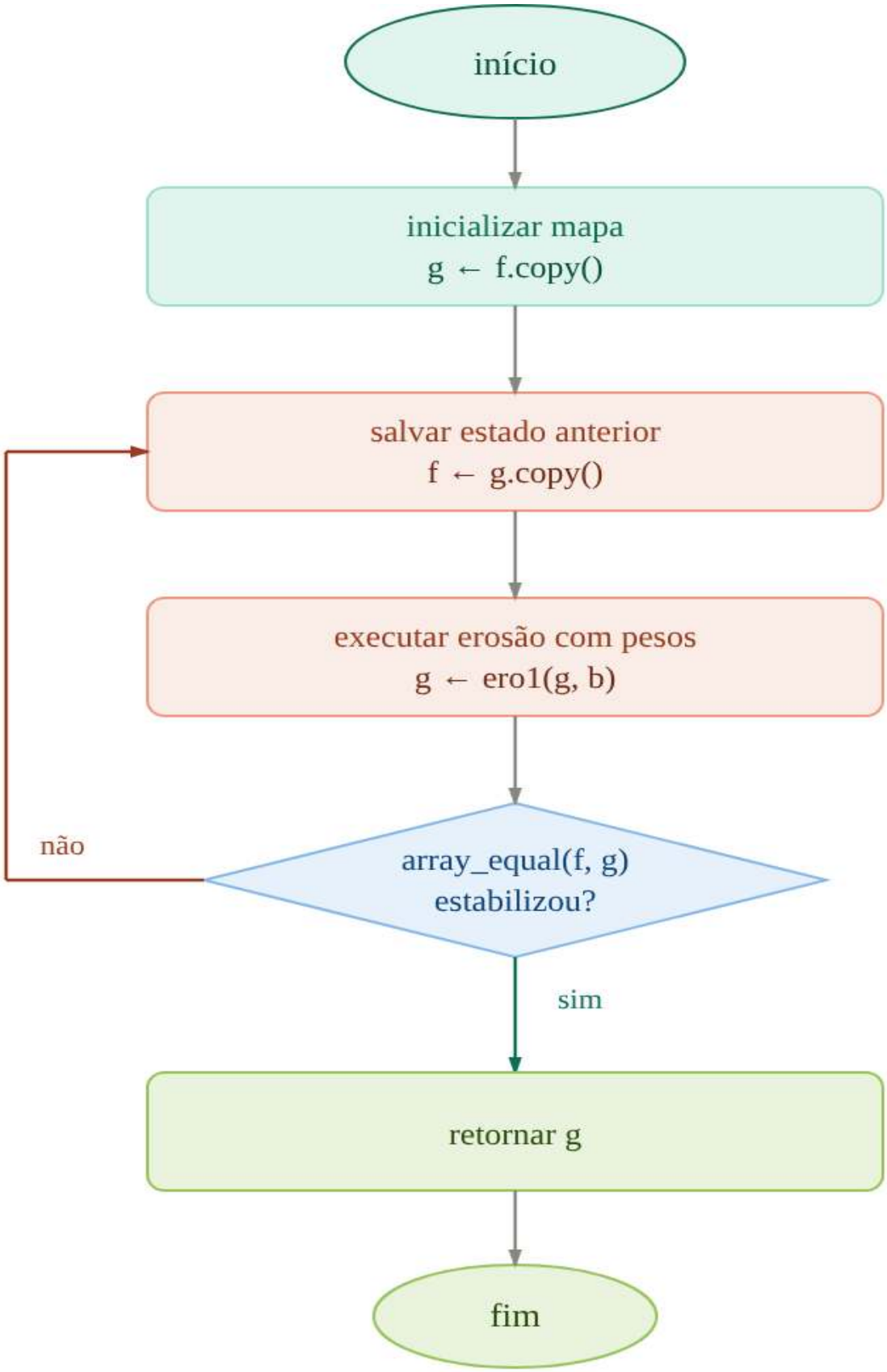
> **i** Definizione formale tramite erosioni
>
> La TD ammette inoltre un'interpretazione morfologica iterativa, come nell'algoritmo della Figura 4.18. Si consideri una funzione strutturante b il cui valore centrale è nullo e i cui vicini possiedono costi negativi associati allo spostamento. Applicando erosioni successive con questa particolare funzione strutturante, i valori dei pixel degli oggetti (che devono assumere la distanza massima possibile dall'immagine) vengono progressivamente ridotti secondo i costi definiti da b. Il valore accumulato di questa propagazione rappresenta quindi la distanza dallo sfondo secondo la metrica indotta dalla funzione strutturante.

Questa interpretazione è implementata in `mm::dist1()`, che accumula erosioni successive utilizzando l'operazione `mm::ero1()`. Invece `mm::dist()` delega il calcolo della distanza Euclidea all'operatore ottimizzato di OpenCV `mm::dist(f)`, dove `f` è l'immagine binaria di ingresso, la distanza L2 specifica la metrica Euclidea (L_2) e `5` indica l'uso di una maschera 5×5 per approssimare la distanza con elevata precisione.

La funzione `dist1` produce una trasformata della distanza discreta la cui metrica è determinata dalla geometria e dai pesi della funzione strutturante utilizzata. Ad esempio, utilizzando una funzione strutturante a croce con costo unitario per i quattro vicini ortogonali, si ottiene la distanza di ***Manhattan*** (L_1). Altre scelte di vicinanza e pesi inducono metriche differenti. Invece `mm::dist()` calcola un'approssimazione efficiente della distanza Euclidea (L_2).

Poiché richiede successive erosioni su tutta l'immagine, l'approccio `dist1` possiede un costo computazionale significativamente maggiore rispetto a `mm::dist()`, essendo impiegato in questo libro principalmente a fini didattici e per evidenziare la relazione tra morfologia matematica e trasformate della distanza.

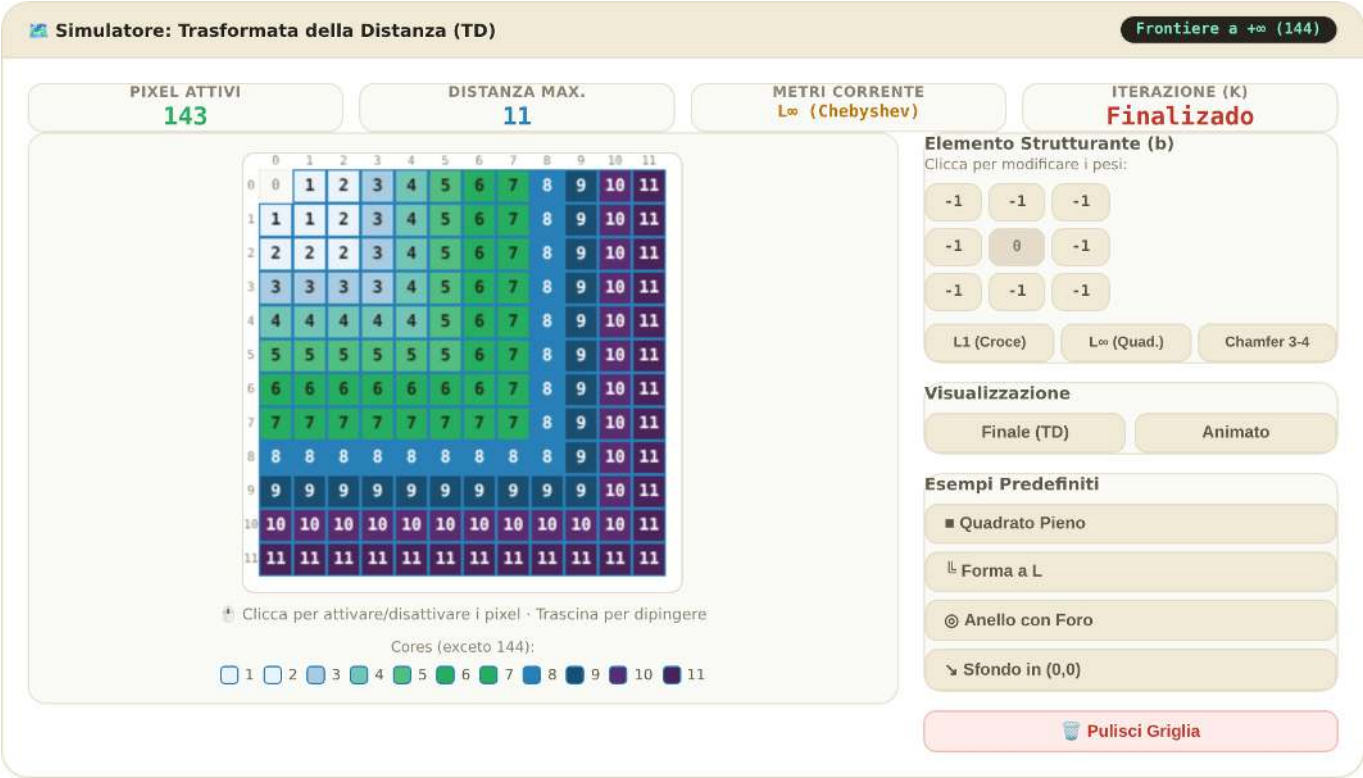
A Figura 4.19 presenta un simulatore interattivo della TD: è possibile posizionare il cursore su diversi pixel dell'oggetto e osservare, in tempo reale, il valore della distanza associato a quella posizione, cioè la distanza fino al pixel di sfondo più vicino. La Figura 4.20 presenta un esempio pratico di questa esecuzione in ambiente Python.



Ponto Fixo Numérico: A distância emerge da propagação matemática do valor zero.

Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.it/cap04/sim-alg-distancia2.html

Figura 4.18: Algoritmo della Trasformata della Distanza.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.it/cap04/sim-04-distancia.html

Figura 4.19: Simulatore interattivo della Trasformata della Distanza (TD) iterativa tramite erosione in toni di grigio. I pixel fuori dall'immagine assumono il valore massimo (144), propagando i costi a partire dallo sfondo interno.

```
%%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 <algorithm>
#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);

    // Find max values
    auto max_iter = *std::max_element(d_iter.data.begin(), d_iter.data.end());
    auto max_l2 = *std::max_element(d_l2.data.begin(), d_l2.data.end());

    std::cout << "Máx. dist1 (erosões) : " << (int)max_iter << " px\n";
    std::cout << "Máx. dist  (L2)      : " << (int)max_l2 << " px\n";
```

```
    mm::show(
        std::vector<mm::Image>{f, d_iter, d_l2},
        MM_OUT,
        std::vector<std::string>{"f original", "mm.dist1 (erosões com cruz)", "mm.dist (L2)"},
        3
    );


// [pdi:panel-io] auto-generated - do not edit by hand
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)")
```

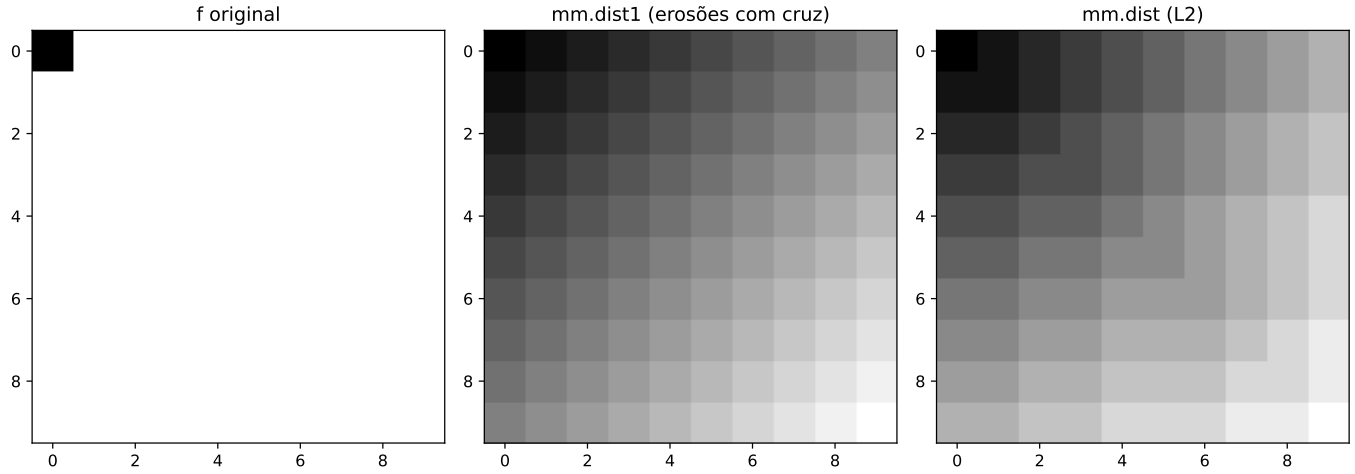


Figura 4.20: Transformada de Distância em imagem binária 10×10. Esquerda: original (*foreground* = 255). Centro: mm.dist1 iterativa (erosões com cruz). Direita: mm.dist (L2).

L'annotazione dei valori numerici direttamente sui pixel consente di verificare come `dist1` propaghi le distanze secondo la metrica indotta dalla funzione strutturante utilizzata. Nel caso dell'elemento a croce con costo unitario, i valori ottenuti corrispondono alla distanza di *Manhattan* (L_1). Sebbene `dist1` e `mm::dist` producano valori numerici distinti poiché adottano metriche differenti, entrambe le trasformate preservano la struttura topografica degli oggetti, facendo sì che i loro massimi si verifichino in regioni centrali simili. Questa proprietà giustifica l'uso di `mm::dist` nelle applicazioni pratiche, grazie alla sua elevata efficienza computazionale.

4.4.3 Trasformata della Distanza Euclidea in quattro passaggi

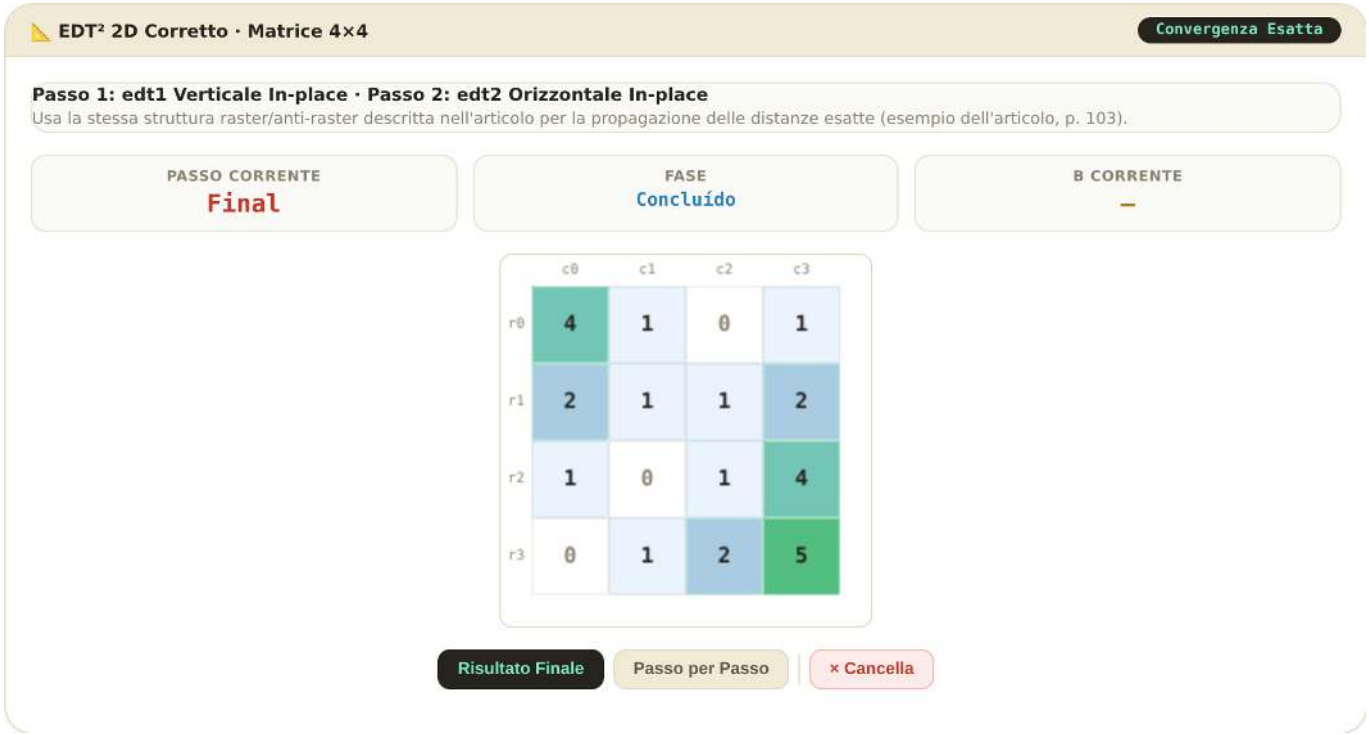
Il simulatore della Figura 4.21 implementa l'algoritmo della TDE di Lotufo (2001) in due fasi. Nella prima fase, la funzione `edt1` esegue una trasformazione unidimensionale verticale in modo sequenziale (*in-place*), percorrendo ciascuna colonna in *raster* (↓) e *anti-raster* (↑) per calcolare le distanze nella direzione verticale (due passaggi: Sud e Nord). Nella seconda fase, la funzione `edt2` utilizza questo risultato come input ed esegue una propagazione orizzontale tramite code: per ogni riga della matrice, due code di priorità, `Eq` e `Wq`, vengono inizializzate scorrendo gli indici di colonna in direzioni opposte (`Eq` da `W-1` a `1`, `Wq` da `2` a `W`), così che ogni pixel aggiornato accodi immediatamente i suoi vicini per un riprocessamento all'interno dello stesso ciclo (altri due passaggi: Est e Ovest). Questo meccanismo a coda consente di applicare erosioni successive con pesi dispari crescenti (`b = 1, 3, 5, ...`, incrementati a ogni iterazione del ciclo esterno) senza che la propagazione rimanga bloccata su valori obsoleti, poiché ogni ciclo risolve completamente la catena di dipendenze orizzontali prima del successivo incremento di `b`. La convergenza di questa propagazione produce la Trasformata della Distanza Euclidea sull'intera matrice, combinando l'informazione verticale ottenuta in `edt1` con la propagazione orizzontale a coda eseguita in `edt2`.

4.4.3.1 Trasformata di Distanza Geodetica

La trasformata di distanza geodetica associa a ogni pixel la distanza minima fino a un marcatore, sotto il vincolo imposto da una maschera. In questo modo, la propagazione avviene esclusivamente attraverso i pixel consentiti, preservando la connettività del dominio.

La Figura 4.22 illustra questo processo in un labirinto: (a) la maschera `g`; (b) la distanza geodetica `D1` calcolata dall'ingresso; (c) la distanza `D2` calcolata dall'uscita; e (d) il percorso minimo ottenuto da queste due trasformate.

Il percorso ottimale è determinato dalla somma delle distanze (`D1 + D2`). I pixel appartenenti alla traiettoria minima sono quelli per cui tale somma assume il suo valore più basso, definendo una connessione tra ingresso e uscita con lunghezza geodetica minima.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.it/cap04/sim-04-tde.html

Figura 4.21: Simulatore interattivo 2D (4x4) con sincronizzazione rigorosa delle code di propagazione orizzontale (b) per ottenere la convergenza esatta descritta nell'articolo.

Questo principio consente di risolvere labirinti senza la necessità di esplorare esplicitamente tutte le possibilità di percorso. La soluzione emerge direttamente dalla propagazione delle distanze in un dominio ristretto. Questo approccio è particolarmente rilevante in labirinti di elevata complessità, come quelli costruiti a partire da strutture quasi-cristalline e cicli hamiltoniani descritti da Singh (2024). In Zampirolli (2025), questo stesso formalismo viene impiegato per risolvere un labirinto complesso; successivamente, il metodo è illustrato in una versione semplificata del problema.

```
%%writefile tmp/fig_04_gdist_menor_caminho.cpp
#define MM_OUT "tmp/fig_04_gdist_menor_caminho.png"
// Compile with: g++ -std=c++17 -o program program.cpp -lopencv_core -lopencv_imgproc
-lopencv_highgui -I/usr/include/opencv4 && ./program
#include "morph.hpp"
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>

//| 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

int main() {
    // 1 = corredor, 0 = parede
    mm::Image g(10, 10, 1);
    int g_data[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) = g_data[y][x];

    // marcador da entrada
    mm::Image entrada(10, 10, 1);
    entrada.at(0, 1) = 1;

    // marcador da saída
    mm::Image saida(10, 10, 1);
    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 = mm::addm(D1, D2);

    // menor valor válido da soma
    int dmin = 255;
    for (int i = 0; i < S.h * S.w * S.channels; 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, 1);
    for (int y = 0; y < 10; y++) {
        for (int x = 0; x < 10; x++) {
            caminho.at(y, x) = (S.at(y, x) == dmin) ? 255 : 0;
        }
    }

    std::cout << "Distância geodésica mínima: " << dmin << "\n";

    std::vector<std::string> titles = {
        "Labirinto",
        "Distância da Entrada",
        "Distância da Saída",
        "Menor Caminho\n(d=" + std::to_string(dmin) + ")"
    };

    mm::show(
        std::vector<mm::Image>{g, D1, D2, caminho},
        MM_OUT,
        titles,
        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)")
```

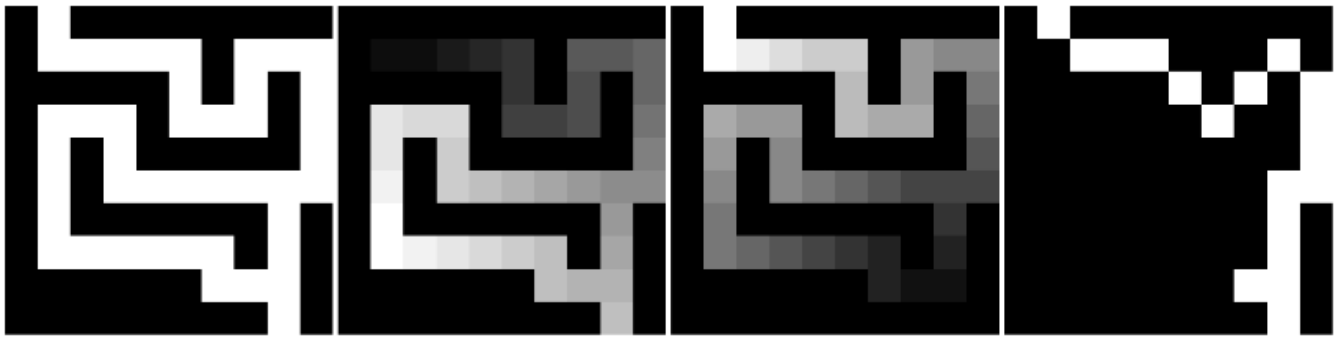


Figura 4.22: Menor caminho geodésico em um labirinto. As distâncias geodésicas são calculadas a partir da entrada e da saída usando mm.gdist. Os pixels cujo somatório das duas distâncias é igual à distância mínima entre os marcadores pertencem a um caminho ótimo.

4.4.4 Segmentação por *Watershed*

L'algoritmo ***Watershed*** interpreta un'immagine in livelli di grigio come una superficie topografica, in cui valori elevati corrispondono a montagne e valori bassi corrispondono a valli o bacini di drenaggio (*catchment basins*). Nel contesto della segmentazione basata su marcatori, i massimi della Trasformata della Distanza sono spesso utilizzati per identificare regioni interne agli oggetti, fornendo semi affidabili per il processo di inondazione.

La segmentazione viene quindi realizzata tramite una simulazione concettuale di inondazione progressiva a partire da questi marcatori. Man mano che i bacini associati a semi diversi si espandono, le regioni vicine entrano eventualmente in contatto. In quel momento vengono costruite barriere virtuali, denominate *watershed lines*, che delimitano gli oggetti della scena. Questo meccanismo consente di separare oggetti adiacenti o parzialmente sovrapposti, anche quando essi formano una singola componente connessa dopo la sogliatura.

L'implementazione didattica presentata in questo capitolo esplora inizialmente il concetto di crescita delle regioni (*region growing*) confinato da una maschera binaria, come dettagliato nell'algoritmo interattivo della Figura 4.23.

i Versione didattica versus implementazione classica

La funzione `mm::watershed0` non implementa l'algoritmo *watershed* classico. Il suo scopo è illustrare, in modo semplificato, la propagazione dei marcatori tramite crescita delle regioni (*region growing*), consentendo di visualizzare come diversi semi competono per l'occupazione dello spazio disponibile. La crescita è delimitata da una maschera binaria di supporto e monitorata da un controllo di stagnazione, generando un risultato simile a una partizione di Voronoi limitata alla geometria degli oggetti di input. La funzione `mm::watershed`, invece, utilizza l'implementazione ottimizzata di OpenCV (`mm::watershed`), che esegue l'inondazione su una superficie topografica definita dall'immagine di input. In questo caso, la propagazione dei marcatori è influenzata dai valori dei pixel, facendo sì che le linee di separazione si formino naturalmente sulle creste del rilievo.

Figura 4.24 presenta un simulatore iterativo che illustra la propagazione dei marcatori nella regione di interesse. Ogni marcatore agisce come una sorgente di allagamento che espande la propria area di influenza fino a incontrare regioni provenienti da altri semi. Nell'algoritmo di OpenCV, i pixel appartenenti alle linee di divisione sono identificati dal valore `-1`, che rappresenta i confini tra bacini adiacenti.

Pipeline Morfologico del *Watershed*

Il *watershed* basato su marcatori integra normalmente un flusso più ampio di segmentazione. Nelle immagini reali, fasi di pre-elaborazione sono spesso necessarie per migliorare il contrasto, ridurre il rumore e generare marcatori affidabili. Questo flusso completo è riassunto nella Tabella 4.5.

Tabella 4.5: *Pipeline* completo del *watershed* basato su marcatori per immagini reali.

Fase	Operazione	Finalità
1	CLAHE + Smussamento	Miglioramento del contrasto e riduzione del rumore
2	Sogliatura	Separazione iniziale tra oggetto e sfondo
3	Apertura/Chiusura	Rimozione di rumori e piccole imperfezioni
4	Dilatazione della Maschera	Identificazione dello Sfondo Certo (*Sure Background*)
5	Trasformata della Distanza + Soglia	Identificazione dell'Oggetto Certo (*Sure Foreground*)
6	Regione Incerta	Differenza tra Sfondo Certo e Oggetto Certo
7	`mm::watershed`	Propagazione dei marcatori attraverso la regione incerta

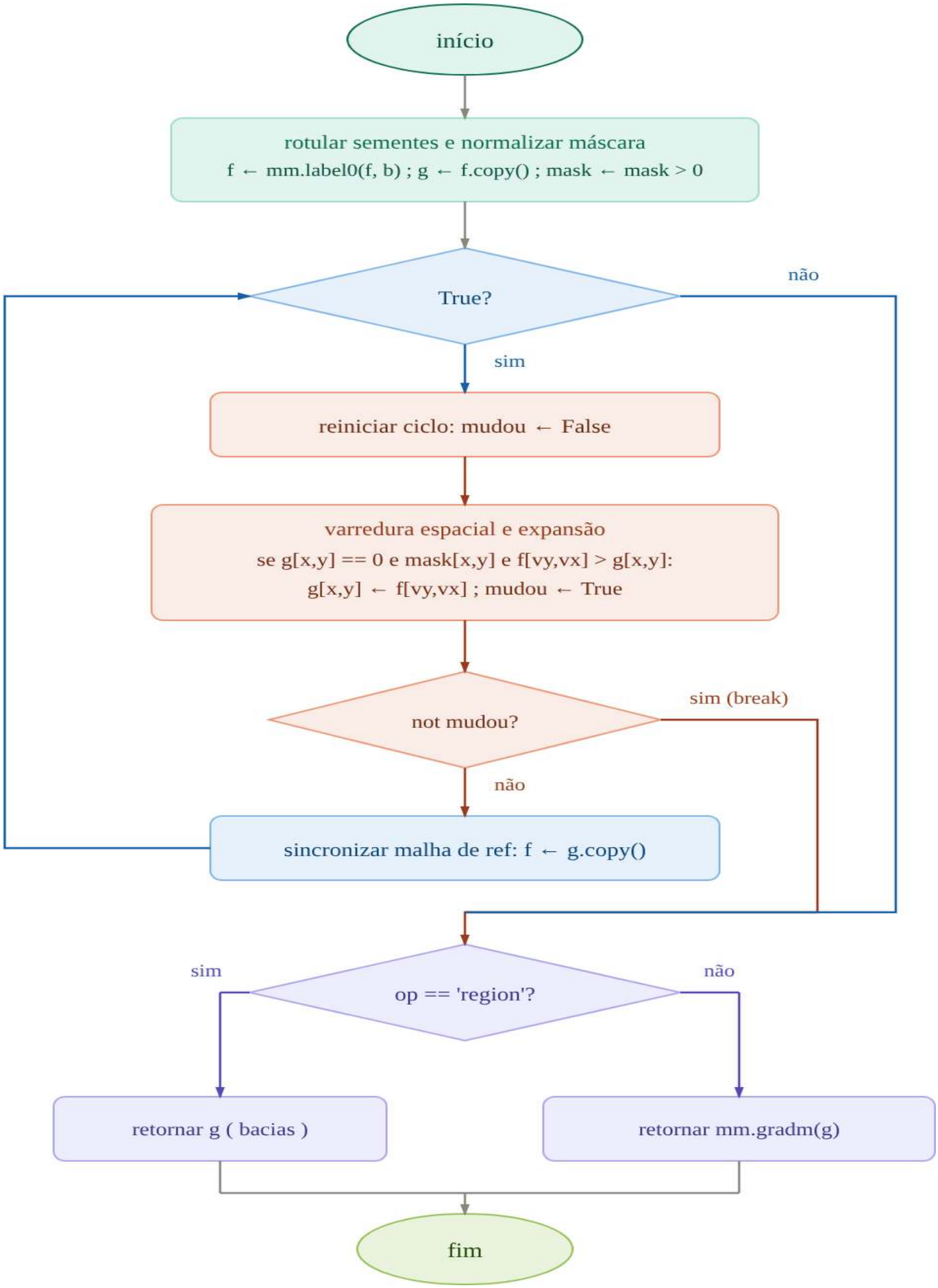
Per enfatizzare esclusivamente i concetti di Trasformata della Distanza, marcatori e inondazione topografica, l'esempio della Figura 4.25 utilizza un'immagine binaria sintetica e adotta un flusso semplificato, riassunto nella Tabella 4.6.

Tabella 4.6: *Pipeline* semplificato utilizzato nell'esempio didattico della Figura 4.25.

Fase	Operazione	Finalità
1	Trasformata della Distanza	Costruzione della superficie topografica
2	Soglia della TD	Estrazione dei marcatori (*Sure Foreground*)

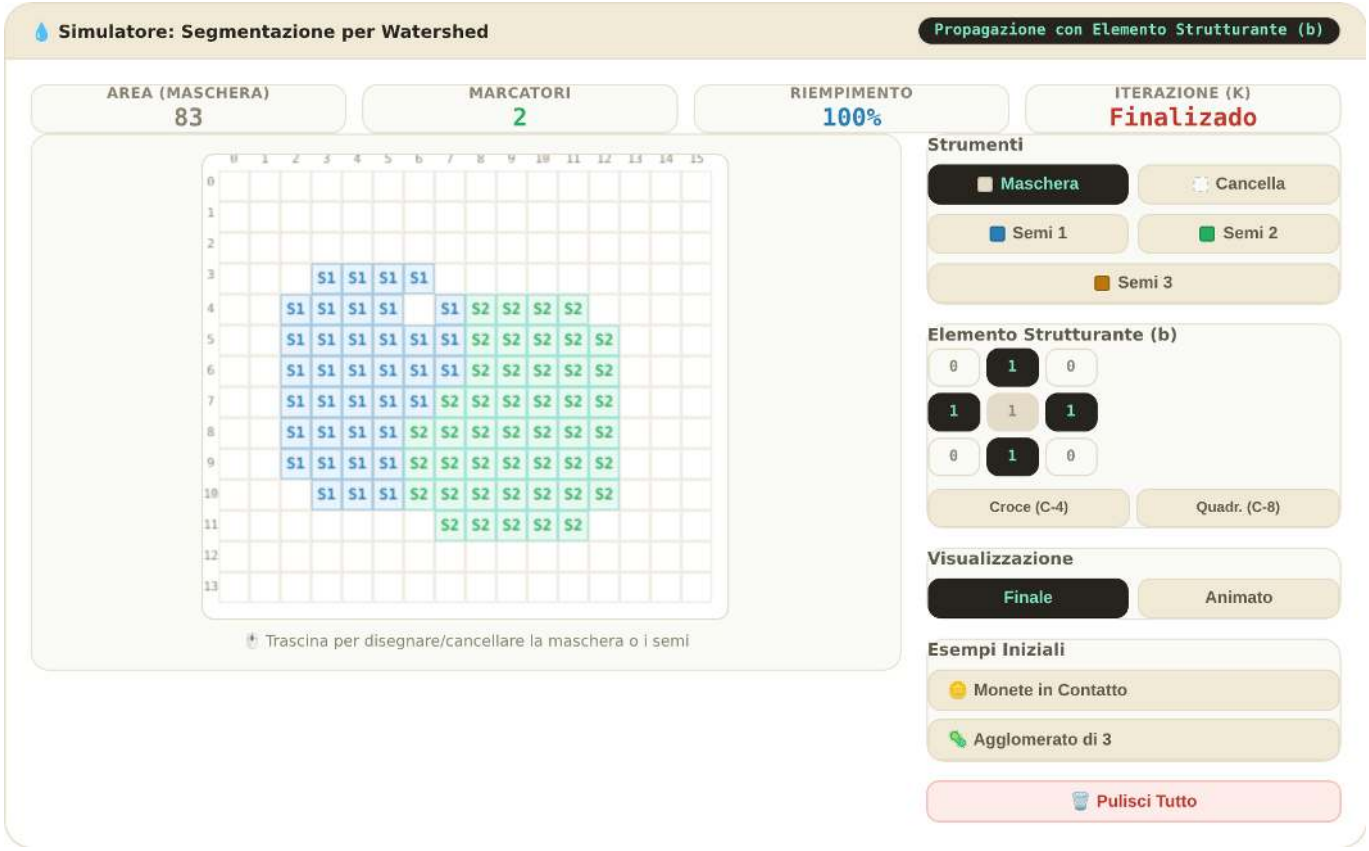
Fase	Operazione	Finalità
3	Dilatazione	Determinazione dello Sfondo Certo (*Sure Background*)
4	Regione Incerta	Differenza tra sfondo e marcatori
5	`mm::watershed`	Propagazione dei marcatori e generazione delle frontiere

```
%%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

// Compile: g++ -std=c++17 -o programa programa.cpp -lmorph -lopencv_core -lopencv_imgproc
-lopencv_highgui

#include "morph.hpp"
#include <iostream>
#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)
    mm::Image m(20, 20);
    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 * 255) ? 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");

    mm::Image w_reg2  = mm::watershed(m, f_sint, "region");
    mm::Image w_line2 = mm::watershed(m, f_sint, "line");

    // 4. Exibição dos resultados
    mm::show(std::vector<mm::Image>{f_sint, dist, m, w_reg2, w_line2}, 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;
```



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.it/cap04/sim-alg-watershed2.html

Figura 4.23: Algoritmo Didático di *Watershed* per Crescita di Regioni Limitato da Maschera.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.it/cap04/sim-04-watershed.html

Figura 4.24: Simulatore interattivo dell'Algoritmo *Watershed* per propagazione morfologica. Disegna la maschera, posiziona i marcatori e regola l'Elemento Strutturante per osservare l'inondazione. Quando i bacini si incontrano simultaneamente, il pareggio viene risolto assumendo una delle regioni in modo casuale.

```
}
```

```
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)")
```

Figura 4.25: *Pipeline watershed* delimitado por máscara em imagem binária 20×20.

Applicazione su monete sovrapposte:

```
%%writefile tmp/fig_04_watershed_moedas.cpp
#define MM_OUT "tmp/fig_04_watershed_moedas.png"
#include "morph.hpp"
#include <opencv2/opencv.hpp>
#include <iostream>
```

```
#include <vector>
#include <set>
#include <algorithm>
#include <cstring>
#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]

    // img_coins_gray and img_final are provided
    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);

    // Encontrar o valor máximo de dist
    int max_dist = 0;
    for (unsigned int i = 0; i < dist.data.size(); i++) {
        if (dist.data[i] > max_dist) max_dist = dist.data[i];
    }

    mm::Image dist_vis(dist.h, dist.w);
    if (max_dist > 0) {
        for (unsigned int i = 0; i < dist.data.size(); i++) {
            dist_vis.data[i] = static_cast<unsigned char>(255.0 * dist.data[i] / max_dist);
        }
    } else {
        dist_vis = dist;
    }

    // 4. Picos seguros (Marcadores das moedas)
    mm::Image picos(dist.h, dist.w);
    for (unsigned int i = 0; i < dist.data.size(); i++) {
        picos.data[i] = (dist.data[i] > 0.5 * max_dist) ? 255 : 0;
    }

    // 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<unsigned char> unique_labels_set;
    for (unsigned int i = 0; i < ws_region.data.size(); i++) {
        if (ws_region.data[i] > 0) {
            unique_labels_set.insert(ws_region.data[i]);
        }
    }
    std::vector<unsigned char> labels(unique_labels_set.begin(), unique_labels_set.end());
    std::cout << "Objetos detectados: " << labels.size() << "\n";

    // 7. Anotação Final
```

```
    // Converter mm::Image para cv::Mat
    cv::Mat img_base_cv(img_base.h, img_base.w, CV_8UC1, img_base.data.data());
    cv::Mat img_annotated_cv;
    cv::cvtColor(img_base_cv, img_annotated_cv, cv::COLOR_GRAY2BGR);

    // Converter ws_region para cv::Mat para comparação
    cv::Mat ws_region_cv(ws_region.h, ws_region.w, CV_8UC1, ws_region.data.data());

    int label_counter = 0;
    for (unsigned char label_id : labels) {
        // Criar máscara para este label
        cv::Mat mask_reg(ws_region.h, ws_region.w, CV_8UC1, cv::Scalar(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) {
                    mask_reg.at<unsigned char>(y, x) = 255;
                }
            }
        }

        // Verificar tamanho
        int area = cv::countNonZero(mask_reg);
        if (area < 500) continue;

        // Calcular centroide
        cv::Moments m = cv::moments(mask_reg, true);
        if (m.m00 == 0) continue;
        int cx = static_cast<int>(m.m10 / m.m00);
        int cy = static_cast<int>(m.m01 / m.m00);

        label_counter++;
        cv::putText(img_annotated_cv, std::to_string(label_counter),
                    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
    // Dilatar ws_line com um kernel 11x11 de uns
    mm::Image ws_line_dilated = mm::dil(ws_line, mm::SE::box(11));

    // Aplicar cor vermelha onde máscara dilatada > 0
    for (int y = 0; y < img_annotated_cv.rows; y++) {
        for (int x = 0; x < img_annotated_cv.cols; x++) {
            if (ws_line_dilated.at(y, x) > 0) {
                img_annotated_cv.at<cv::Vec3b>(y, x) = cv::Vec3b(0, 0, 255);
            }
        }
    }

    // Converter de volta para mm::Image
    mm::Image img_annotated(img_annotated_cv.rows, img_annotated_cv.cols, 3);
    std::memcpy(img_annotated.data.data(), img_annotated_cv.data, img_annotated.data.size());

    // Exibição
    mm::show(
        std::vector<mm::Image>{img_base, img_sobrepostas, dist_vis, picos, ws_region,
img_annotated},
        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)")
```

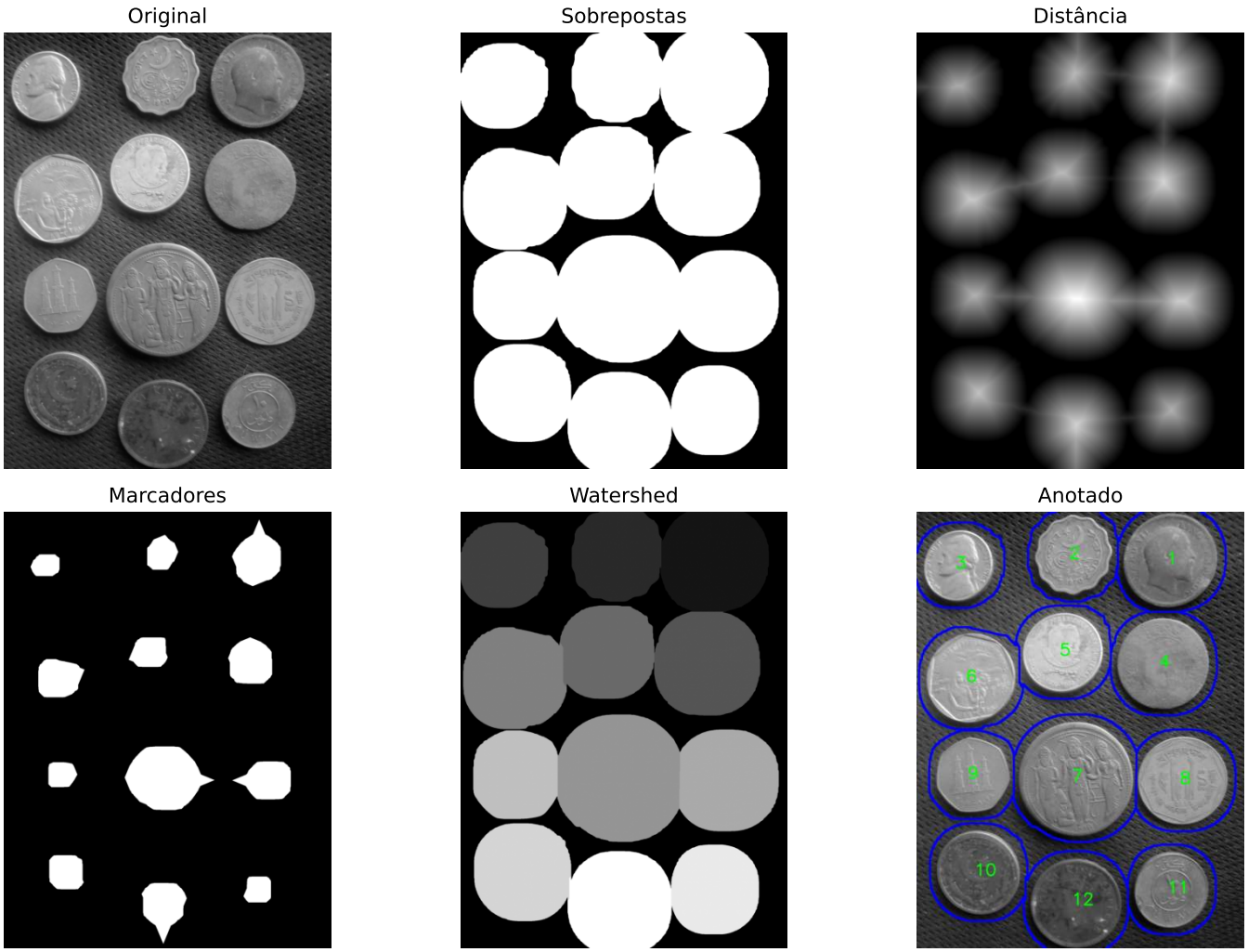


Figura 4.26: *Pipeline Watershed* para separação de moedas sobrepostas: da máscara binária dilatada até os contornos finais anotados sobre a imagem original.

4.5 Estrazione di Componenti e Descrittori di Forma

Dopo la segmentazione e il raffinamento morfologico, il passo successivo consiste nell'identificare individualmente ciascun oggetto presente nell'immagine ed estrarne le proprietà geometriche. Questa fase è fondamentale per compiti di misurazione, classificazione e riconoscimento di pattern.

Un **componente connesso** è un insieme massimale di pixel appartenenti all'oggetto che rimangono mutuamente connessi secondo una relazione di connettività precedentemente definita (connettività a 4 o a 8). Dopo l'etichettatura, ogni componente riceve un identificatore univoco, consentendo che le sue caratteristiche vengano analizzate singolarmente.

OpenCV offre due approcci complementari per questa analisi, riassunti nella Tabella 4.7.

Tabella 4.7: Confronto tra gli approcci basati su componenti connessi e su contorni.

	`connectedComponentsWithStats`	`findContours`
Restituisce	etichetta per pixel e statistiche per componente	sequenza di punti che descrive il contorno
Descrittori diretti	area, *bounding box* e centroide	perimetro, forma e gerarchia
Oggetti a contatto	tende a fondere regioni connesse	tende a produrre un unico contorno esterno

	`connectedComponentsWithStats`	`findContours`
Uso tipico	conteggio, filtraggio ed etichettatura	analisi geometrica e descrittori di forma

4.5.1 Etichettatura e Statistiche dei Componenti

`mm::label0` assegna un'etichetta a ogni componente connesso. Dall'immagine etichettata si ottengono, tramite scansione, l'**area** (conteggio dei pixel) e il **riquadro di delimitazione** di ciascun oggetto (Figura 4.27). Il `connectedComponentsWithStats` di OpenCV, che restituisce queste statistiche già pronte, e le annotazioni colorate sull'immagine si trovano nel percorso Python.

```
%%writefile tmp/fig_04_componentes.cpp
#define MM_OUT "tmp/fig_04_componentes.png"
#include "morph.hpp"
#include <opencv2/opencv.hpp>
#include <iostream>
#include <vector>
#include <string>
#include <cstring>
#include <iomanip>
#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
    // Bridge mm::Image -> cv::Mat (sem cópia)
    cv::Mat img_final_mat(img_final.h, img_final.w, CV_8UC1, img_final.data.data());

    // cv::connectedComponentsWithStats requer 7 argumentos (com algoritmo)
    cv::Mat labels_mat, stats_mat, centroids_mat;
    int n = cv::connectedComponentsWithStats(img_final_mat, labels_mat, stats_mat,
    centroids_mat, 8, CV_32S, cv::CCL_DEFAULT);

    // 2. Coloração dos componentes - paleta FIXA para a trilha C++ reproduzir cor a cor
    std::vector<std::vector<unsigned char>> 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}};

    // Criar imagem colorida
    mm::Image img_colored(img_final.h, img_final.w, 3);
    for (int y = 0; y < img_final.h; y++) {
        for (int x = 0; x < img_final.w; x++) {
            int label = labels_mat.at<int>(y, x);
            if (label >= 0 && label < n) {
                const auto& cor = PALETA[label % PALETA.size()];
                img_colored.at(y, x, 0) = cor[0];
                img_colored.at(y, x, 1) = cor[1];
                img_colored.at(y, x, 2) = cor[2];
            }
        }
    }

    // Fundo preto (label 0)
```

```
    for (int y = 0; y < img_final.h; y++) {
        for (int x = 0; x < img_final.w; x++) {
            if (labels_mat.at<int>(y, x) == 0) {
                img_colored.at(y, x, 0) = 0;
                img_colored.at(y, x, 1) = 0;
                img_colored.at(y, x, 2) = 0;
            }
        }
    }

    // Converter para cv::Mat para anotações
    cv::Mat img_annotated_mat(img_colored.h, img_colored.w, CV_8UC3, img_colored.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_mat.at<int>(i, cv::CC_STAT_AREA);
        int cx = (int)centroids_mat.at<double>(i, 0);
        int cy = (int)centroids_mat.at<double>(i, 1);
        int w = stats_mat.at<int>(i, cv::CC_STAT_WIDTH);
        int h = stats_mat.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";

        // Anotar com dois contornos (preto grosso, vermelho fino)
        cv::putText(img_annotated_mat, std::to_string(i) + ": " + std::to_string(area),
                    cv::Point(cx - 150, cy + 18),
                    cv::FONT_HERSHEY_SIMPLEX, 2.0, cv::Scalar(0, 0, 0), 10, cv::LINE_AA);
        cv::putText(img_annotated_mat, std::to_string(i) + ": " + std::to_string(area),
                    cv::Point(cx - 150, cy + 18),
                    cv::FONT_HERSHEY_SIMPLEX, 2.0, cv::Scalar(0, 0, 255), 5, cv::LINE_AA);
    }

    // Copiar de volta para mm::Image
    mm::Image img_annotated(img_annotated_mat.rows, img_annotated_mat.cols,
img_annotated_mat.channels());
    std::memcpy(img_annotated.data.data(), img_annotated_mat.data, img_annotated.data.size());

    // Exibir resultados
    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)")
```

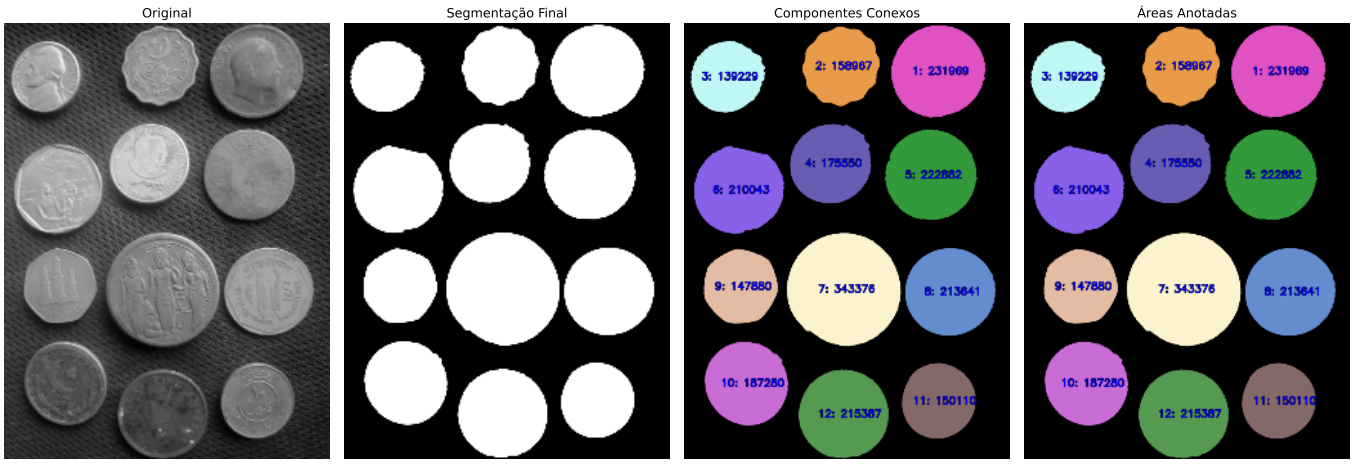


Figura 4.27: Componentes conexos extraídos após o *pipeline* CLAHE → Otsu → limpeza morfológica. Cada objeto é colorido com cor distinta e anotado com sua área em pixels.

4.5.2 Descritores de Forma

I descritores basati su **contorno vettoriale** — perimetro (`arcLength`), area del poligono (`contourArea`), circolarità, momenti e approssimazione poligonale (`approxPolyDP`) — dipendono dal tracciamento dei bordi (`findContours`), assente nella `morph.hpp`. Restano solo nel percorso Python. Nel percorso C++, il **gradiente morfologico** (`mm::gradm`) evidenzia il contorno di ciascun oggetto (Figura 4.28).

```
%%writefile tmp/fig_04_contornos.cpp
#define MM_OUT "tmp/fig_04_contornos.png"
//| 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

#include <opencv2/opencv.hpp>
#include <iostream>
#include <vector>
#include <cmath>
#include <iomanip>
#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]

    // img_coins_gray, img_coins_gray e img_final já estão inicializados aqui

    // Converter mm::Image para cv::Mat para usar as funções do OpenCV
    cv::Mat final_mat(img_final.h, img_final.w,
                      img_final.channels == 1 ? CV_8UC1 : CV_8UC3,
                      img_final.data.data());

    cv::Mat gray_coins_mat(img_coins_gray.h, img_coins_gray.w,
                           img_coins_gray.channels == 1 ? CV_8UC1 : CV_8UC3,
                           img_coins_gray.data.data());

    std::vector<std::vector<cv::Point>> contornos;
```

```
    cv::findContours(final_mat, contornos, cv::RETR_EXTERNAL, cv::CHAIN_APPROX_SIMPLE);

    cv::Mat img_contornos_mat;
    cv::cvtColor(gray_coins_mat, img_contornos_mat, cv::COLOR_GRAY2BGR);
    cv::Mat img_circulares_mat = img_contornos_mat.clone();

    std::cout << "Contornos detectados: " << contornos.size() << std::endl;
    std::cout << "\n" << std::setw(4) << "ID" << std::setw(8) << "Área"
              << std::setw(10) << "Perímetro" << std::setw(14) << "Circularidade" <<
              std::endl;
    std::cout << std::string(42, '-') << std::endl;

    int i = 1;
    for (const auto& cnt : contornos) {
        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) << i << std::setw(8) << std::fixed << std::setprecision(0)
<< area
                  << std::setw(10) << std::setprecision(1) << perim
                  << std::setw(14) << std::setprecision(3) << circ << std::endl;

        cv::drawContours(img_contornos_mat, std::vector<std::vector<cv::Point>>{cnt}, -1,
                         cv::Scalar(0, 255, 0), 3);
        cv::drawContours(img_circulares_mat, std::vector<std::vector<cv::Point>>{cnt}, -1,
                         cv::Scalar(0, 255, 0), 3);

        // Desenhar texto com contorno para melhor legibilidade
        for (const auto& [cor, esp] : std::vector<std::pair<cv::Scalar, int>>{
                {cv::Scalar(0, 0, 0), 8}, {cv::Scalar(255, 0, 0), 3}}) {
            cv::putText(img_circulares_mat, cv::format("%.2f", circ),
                        cv::Point(cx - 80, cy + 15),
                        cv::FONT_HERSHEY_SIMPLEX, 3.6, cor, esp, cv::LINE_AA);
        }
        i++;
    }

    // Converter de volta para mm::Image para exibir com mm::show
    mm::Image img_contornos(img_contornos_mat.rows, img_contornos_mat.cols,
                            img_contornos_mat.channels());
    std::memcpy(img_contornos.data.data(), img_contornos_mat.data, img_contornos.data.size());

    mm::Image img_circulares(img_circulares_mat.rows, img_circulares_mat.cols,
                             img_circulares_mat.channels());
    std::memcpy(img_circulares.data.data(), img_circulares_mat.data,
img_circulares.data.size());

    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  212886    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)")
```

Figura 4.28: Contornos extraídos com *findContours*. Cada moeda é anotada com sua circularidade — valores próximos de 1 confirmam forma circular.

4.5.3 Collegamento con la Rilevazione Moderna degli Oggetti

I descrittori estratti nelle sezioni precedenti — in particolare *bounding boxes*, centroidi, aree e misure di forma — stabiliscono un ponte naturale tra la segmentazione morfologica classica e i sistemi moderni di rilevazione degli oggetti. Sebbene le tecniche studiate in questo capitolo utilizzino operazioni su pixel e regioni segmentate, molte delle rappresentazioni prodotte sono direttamente compatibili con i formati impiegati nei modelli contemporanei di visione artificiale.

I rilevatori basati su apprendimento profondo, come la famiglia YOLO (*You Only Look Once*) (REDMON, 2016), operano direttamente su immagini a colori e producono, per ciascun oggetto rilevato, una *bounding box* descritta dal centro (cx, cy) e dalle dimensioni (w, h), oltre a una classe e a un punteggio di confidenza. Questa rappresentazione condivide la stessa struttura geometrica di base ottenuta tramite `connectedComponentsWithStats`, sebbene sia prodotta da un modello appreso e non da una segmentazione esplicita.

La Figura 4.29 illustra come le *bounding boxes* ottenute per morfologia possano essere esportate nel formato YOLO per comporre insiemi di dati utilizzati nell'addestramento o nella valutazione dei rilevatori.

```
%%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 "morph.hpp"
#include <iostream>
#include <sstream>
#include <string>
#include <vector>
#include <cmath>
#include <iomanip>
#include <fstream>
#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]

    // Recalcula rótulos/estatísticas a partir da segmentação final
    cv::Mat final_mat(img_final.h, img_final.w, CV_8UC1, img_final.data.data());

    cv::Mat labels, stats, centroids;
    int n = cv::connectedComponentsWithStats(final_mat, labels, stats, centroids, 8);

    int H_img = img_coins_gray.h;
    int W_img = img_coins_gray.w;

    cv::Mat gray_mat(img_coins_gray.h, img_coins_gray.w, CV_8UC1, img_coins_gray.data.data());
    cv::Mat img_bbox;
    cv::cvtColor(gray_mat, img_bbox, cv::COLOR_GRAY2BGR);

    int CLASSE = 0;   // 0 = moeda (única categoria neste exemplo)

    std::cout << std::setw(4) << "cls" << " "
              << std::setw(8) << "cx_n" << " "
              << std::setw(8) << "cy_n" << " "
              << std::setw(8) << "w_n" << " "
              << std::setw(8) << "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  = (double)w / W_img;
        double h_n  = (double)h / H_img;

        std::ostringstream oss;
        oss << CLASSE << " "
            << std::fixed << std::setprecision(4) << cx_n << " "
            << cy_n << " " << w_n << " " << h_n;
        yolo_linhas.push_back(oss.str());

        std::cout << std::setw(4) << CLASSE << " "
                  << std::fixed << std::setprecision(4)
                  << std::setw(8) << cx_n << " "
                  << std::setw(8) << cy_n << " "
                  << std::setw(8) << w_n << " "
                  << std::setw(8) << 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 j = 0; j < yolo_linhas.size(); j++) {
        if (j > 0) f << "\n";
        f << yolo_linhas[j];
    }
    f.close();
    std::cout << "\nAnotação salva em moedas.txt\n";

    // Copy result back to mm::Image
    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)")
```

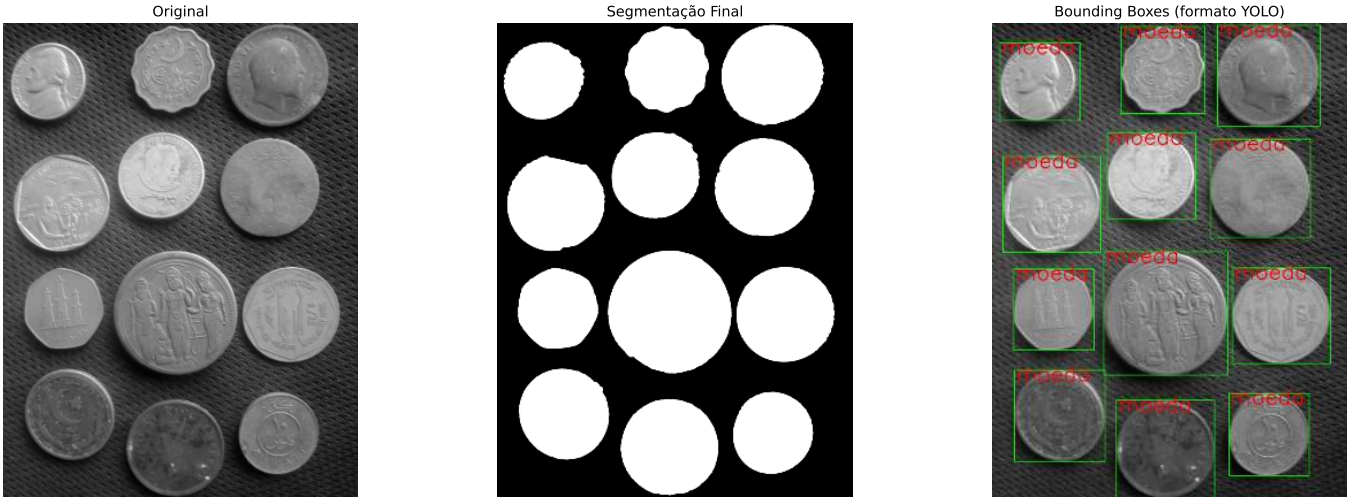


Figura 4.29: *Bounding boxes* derivadas dos componentes conexos sobrepostas à imagem original. As anotações são exportadas no formato YOLO (*classe cx cy w h*), com coordenadas normalizadas para o intervalo [0,1].

Il formato YOLO memorizza ogni oggetto in una riga contenente cinque campi:

```
classe cx cy w h
```

dove (cx, cy) rappresenta il centro della *bounding box* e (w, h) le sue dimensioni. Tutti i valori geometrici sono normalizzati nell'intervallo $[0, 1]$ rispetto alla larghezza e all'altezza dell'immagine. La **classe** è un identificatore intero associato a una categoria definita dal dataset (ad esempio, `0 → moneta`). Quando vi sono più categorie — come moneta d'oro (`0`), moneta d'argento (`1`) e disco di plastica (`2`) — è sufficiente assegnare l'identificatore corrispondente a ciascun oggetto prima dell'esportazione, mantenendo esattamente lo stesso formato di annotazione. Nell'esempio precedente, queste informazioni sono state memorizzate nel file `moedas.txt`.

Il flusso presentato in questo capitolo — segmentazione → etichettatura → estrazione delle *bounding box* — corrisponde concettualmente alla fase di **annotazione** (*labeling*) impiegata nella costruzione dei set di addestramento per i moderni detector. Strumenti specializzati, come Label Studio e Roboflow, automatizzano questo processo in immagini complesse, ma la logica fondamentale rimane la stessa: associare a ciascun oggetto una regione di interesse e una classe. In scenari controllati, con sfondo uniforme e oggetti ben separati, le tecniche morfologiche possono persino generare annotazioni automaticamente o fungere da punto di partenza per l'etichettatura manuale, riducendo significativamente lo sforzo di costruzione del dataset. Nelle applicazioni reali più complesse, tuttavia, la validazione umana rimane necessaria per garantire la qualità delle annotazioni.

Valutazione: IoU (*Intersection over Union*)

Un modo semplice per valutare la qualità di una segmentazione consiste nel confrontarla con una maschera di riferimento (*ground truth*). La metrica più utilizzata a questo scopo è la **IoU** (*Intersection over Union*):

$$\text{IoU} = \frac{|A \cap B|}{|A \cup B|} \tag{4.16}$$

dove A rappresenta la segmentazione prodotta dall'algoritmo e B la segmentazione di riferimento.
Il valore della IoU varia tra 0 e 1. Quanto più alto è il valore, tanto maggiore è la sovrapposizione tra le maschere. Una IoU pari a 1 indica una corrispondenza perfetta tra la segmentazione ottenuta e il riferimento.
La stessa metrica è ampiamente utilizzata anche nel rilevamento degli oggetti, applicandola alle *bounding boxes* previste e annotate. In questo ambito, valori di IoU superiori a 0,5 sono spesso adottati come criterio minimo per considerare corretta una rilevazione.

Oltre la Morfologia

Le tecniche studiate in questo capitolo segmentano gli oggetti sfruttando la connettività spaziale, le operazioni morfologiche e il rilievo topografico. Esistono, tuttavia, approcci alternativi basati sul raggruppamento di caratteristiche, come l'algoritmo *k-means*, i modelli di miscela gaussiana (GMM) e metodi più recenti basati sull'apprendimento profondo. Queste tecniche verranno riprese nella Parte II del libro, dedicata alla Visione Artificiale.

4.6 Riassunto

In questo capitolo sono state presentate le principali tecniche di segmentazione e morfologia matematica, concludendo lo studio dell'elaborazione delle immagini nel dominio spaziale.

- **Pre-elaborazione e sogliatura:** La combinazione tra equalizzazione adattiva CLAHE e il metodo di Otsu si è dimostrata efficace per indurre una separazione bimodale nell'istogramma e semplificare la binarizzazione di immagini con illuminazione non uniforme.
- **Erosione e dilatazione:** Operatori morfologici fondamentali basati sulla ricerca di minimi e massimi locali in un intorno definito dall'elemento strutturante B. Sono operatori duali rispetto al complemento e sono stati implementati tramite le funzioni `mm::ero` e `mm::dil`.
- **Apertura e chiusura:** Composizioni di erosione e dilatazione che consentono di rimuovere rumori, smussare i contorni e colmare piccole lacune, preservando la struttura globale degli oggetti.
- **Ricostruzione morfologica:** Processo geodetico iterativo che propaga un marcatore all'interno dei limiti imposti da una maschera, costituendo la base di operatori come `mm::clohole` e `mm::edgeoff`.
- **Pipeline di pulizia binaria:** Flusso consolidato composto da CLAHE → Otsu → apertura → `mm::clohole` → apertura ristretta → `mm::edgeoff`, che produce maschere adatte all'analisi quantitativa.
- **Morfologia in scala di grigi:** Estensione algebrica basata su minimi e massimi ponderati, che consente operatori come il gradiente morfologico e i filtri *top-hat* per l'enfatizzazione di strutture locali.
- **Trasformata della Distanza e Watershed:** La Trasformata della Distanza ha permesso di generare marcatori automatici per l'algoritmo *watershed*, consentendo la separazione di oggetti adiacenti o parzialmente sovrapposti.
- **Componenti connesse e descrittori:** L'etichettatura delle regioni (`mm::label0`) e l'estrazione dei contorni hanno permesso di calcolare descrittori geometrici come area, centroide, perimetro, circolarità e *bounding boxes*.
- **Collegamento con la Visione Artificiale Moderna:** Le *bounding boxes* estratte tramite morfologia sono state esportate nel formato YOLO, evidenziando il legame tra tecniche classiche di segmentazione e sistemi moderni di rilevamento degli oggetti.

Il Capitolo 5 introdurrà tecniche di elaborazione nel **dominio della frequenza**, affrontando la **Trasformata di Fourier**, il filtraggio spettrale e i fondamenti della **compressione delle immagini**, inclusi DCT, JPEG e *wavelets*.

4.7 🤖 Uso del Gemini Notebook come Tutor Complementare

In questa edizione, l'uso del **Gemini Notebook** è incentivato come strumento complementare di apprendimento. Basato sull'intelligenza artificiale, il sistema utilizza esclusivamente i documenti forniti dall'autore come fonte di conoscenza, producendo risposte allineate al contenuto e all'approccio adottato nel corso di questo capitolo.

🎓 Studia con il Tutor Intelligente

🚀 ACCEDI AL GEMINI NOTEBOOK: CAPITOLO 04

🌐 Lingua e Linguaggio di Programmazione

Il progetto di questo capitolo nel Gemini Notebook è stato costruito esclusivamente con il testo in **portoghese** e gli esempi di codice in **Python**. Se stai studiando dall'edizione in inglese o francese, o seguendo il percorso in C++, le risposte del tutor potrebbero non corrispondere esattamente alla versione che stai leggendo.

⚠️ Avviso sui Contenuti Generati dall'IA

Sebbene sia uno strumento prezioso di supporto allo studio, il Gemini Notebook può occasionalmente produrre risposte incomplete, imprecise o errate. Si raccomanda di validare le informazioni consultando il materiale del capitolo, libri, articoli scientifici e altre fonti accademiche affidabili. Quando possibile, esegui e sperimenta gli esempi pratici presentati nel corso del testo per consolidare la comprensione dei concetti.

4.8 Elenco di Esercizi

1. **(10%)** Implementare manualmente il criterio di Otsu senza utilizzare `mm::threshold`. Calcolare la varianza tra le classi $\sigma_B^2(T)$ per tutte le soglie $T \in [0, 255]$ usando `mm::hist`, identificare la soglia ottimale T^* e confrontare il risultato con il valore ottenuto da OpenCV. Tracciare σ_B^2 in funzione di T ed evidenziare il punto di massimo.

2. **(15%)** Applicare la sogliatura adattativa con blocchi di dimensione 11, 31 e 51 a un'immagine contenente illuminazione non uniforme. Confrontare i risultati con la sogliatura globale di Otsu e discutere i vantaggi e i limiti di ciascun approccio.

3. **(15%)** Eseguire il *pipeline* watershed sull'immagine delle monete variando la soglia applicata alla Trasformata della Distanza (0.3, 0.5 e 0.7 volte il valore massimo). Spiegare come questo parametro influenzi la generazione dei marcatori, la separazione di oggetti adiacenti e l'occorrenza di sovra-segmentazione.

4. **(15%)** Utilizzando `mm::drawImg`, costruire una dimostrazione visiva passo passo dell'erosione di un'immagine binaria 7×7 con elemento strutturante quadrato 3×3. Per ogni posizione analizzata, indicare se l'elemento strutturante è completamente contenuto nell'oggetto e giustificare il valore attribuito al pixel di uscita.

5. **(15%)** Dimostrare sperimentalmente la dualità tra erosione e dilatazione verificando l'identità $(A \ominus B)^c = A^c \oplus \hat{B}$ utilizzando `mm::ero`, `mm::dil` e `mm::bnot`. Calcolare la differenza pixel per pixel tra i due membri dell'equazione e presentare il risultato utilizzando `mm::histImg` o una visualizzazione equivalente.

6. **(15%)** Implementare manualmente il gradiente morfologico utilizzando solo `mm::ero` e `mm::dil`, confrontando il risultato con `mm::gradm(img, B)`. Valutare l'effetto di diversi elementi strutturanti (quadrato 3×3, disco 5×5 e linea 1×9) sulla rilevazione dei bordi.

7. **(15%)** Costruire un *pipeline* completo per il conteggio e la classificazione di monete per dimensione (piccola, media e grande) utilizzando area e circolarità come descrittori. Generare manualmente una maschera di riferimento (*ground truth*) e calcolare la metrica IoU (*Intersection over Union*) per valutare la qualità della segmentazione. Presentare i risultati in tabella e tramite visualizzazioni prodotte con `mm::show`.

Riferimenti del Capitolo

La base teorica di questo capitolo si fonda sulle seguenti opere:

- Gonzalez (2018) per i concetti di segmentazione, sogliatura di Otsu, Trasformata della Distanza, *watershed*, morfologia matematica e descrittori di forma.
- Matheron (1975) e Serra (1982) per la fondazione teorica originale, la formulazione algebrica e lo sviluppo della Morfologia Matematica.
- Szeliski (2022) per la segmentazione basata su regioni, l'etichettatura delle componenti connesse, il *watershed* basato su marcatori e la valutazione della segmentazione tramite la metrica IoU.
- Bradski (2008) per l'utilizzo pratico della libreria OpenCV, incluse funzioni come `mm::dist`, `mm::watershed`, `mm::label0` e l'estrazione dei contorni.
- Redmon (2016) per l'introduzione ai moderni rilevatori della famiglia YOLO e la loro relazione con descrittori geometrici come le *bounding boxes* estratte mediante segmentazione.
- Singh (2024) per la costruzione di labirinti complessi basati su cicli hamiltoniani su mosaici quasicristallini, utilizzati come esempio applicativo della Trasformata della Distanza Geodetica e degli algoritmi di ricerca di percorsi.
- Zampirolli (2025) per l'implementazione degli operatori morfologici, delle trasformate geodetiche e della risoluzione di labirinti tramite propagazione delle distanze in domini ristretti.

4.9 💻 Parte Pratica con Esercizi di Programmazione

Questa lista trasforma i concetti del Capitolo 4 in un percorso pratico di segmentazione e morfologia matematica. Gli EP iniziano con la sogliatura e avanzano fino all'etichettatura e ai descrittori delle componenti, utilizzando sempre matrici di piccole dimensioni affinché ogni pixel possa essere verificato a mano.

> ❗ Regola comune degli EP morfologici
>
> Nelle operazioni con vicinato, **non eseguire il padding**. Per ogni pixel, valutare solo le posizioni dell'elemento strutturante che ricadono all'interno del dominio dell'immagine. Questa è la stessa idea delle implementazioni didattiche in `morph.py`, come `mm::dil0`, `mm::ero0`, `mm::dil1` e `mm::label0`: il vicinato viene ritagliato dal dominio valido dell'immagine.

🎯 Obiettivo di questo Quaderno

Il quaderno consente di sviluppare, validare, organizzare e testare soluzioni di **Esercizi di Programmazione (EP)** in ambienti interattivi, come Colab, con gli stessi casi di test di Moodle, copiandoli lì solo al momento di registrare il voto ufficiale.

Download

Scarica `morph.py` e `testsuite.py` eseguendo la cella qui sotto:

```
import os, urllib.request

os.makedirs("tmp/state", exist_ok=True)  # artefatti di build della pista C++ (.cpp, binario,
PNG)

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")

# Il kernel è comunque Python anche nella pista C++: `mm` (morph.py) è usato dai
# simulatori, dalla visualizzazione delle figure che il binario C++ genera e dallo
# stato mm::Image tra le celle. cpp=True scarica anche la pista compilata
# (morph.hpp + stb_image*.h), usata nell'#include delle celle %%writefile *.cpp.
import config
config.setup(testsuite=True, cpp=True)
from morph import mm
from testsuite import TestSuite
```

```
Ambiente pronto. Morph: 1.1.9 | OpenCV: 5.0.0 | TestSuite: 1.1.2
```

Esecuzione dei Test

Per valutare i test, eseguire `TestSuite("EP04_01.extensão").run()` in una nuova cella, sostituendo l'estensione con quella del linguaggio utilizzato (`.py`, `.java`, `.c`, `.cpp`, `.js` o `.r`). Il sistema scarica i casi di test da GitHub, esegue il programma e calcola automaticamente il voto.

Per testare direttamente codice Python, senza salvare un file, utilizzare `run_code(codigo)` passando il codice come *stringa* in una variabile `codigo`:

```
codigo = """
from morph import mm
# ... tuo codice qui ...
"""
TestSuite("EP04_01").run_code(codigo)
```

4.9.1 EP04_01 Soglia Globale con Soglia Fissa

Negli **scanner di documenti** e nei **sistemi di lettura di codici a barre**, la prima fase dell'elaborazione consiste sempre nel separare ciò che è "oggetto" (inchiostro, testo, barre) da ciò che è "sfondo" (carta, imballaggio). La **soglia globale** fa esattamente questo: confronta ogni pixel con un'unica soglia T e decide, in tempo reale, se esso appartiene alla classe chiara o alla classe scura. È l'operatore di segmentazione più semplice — eppure è alla base di gran parte dei *pipeline* industriali di ispezione visiva. Vedi in Figura 4.30 una simulazione di questo EP.

4.9.1.1 📋 Linee Guida di Implementazione

1. **Dimensioni:** Leggere gli interi L (righe) e C (colonne).
2. **Soglia:** Leggere l'intero T (soglia di decisione).
3. **Dati:** Leggere i valori interi della matrice originale riga per riga.
4. **Mappatura:** Per ogni pixel p, calcolare il nuovo valore tramite l'equazione:

$$p' = \begin{cases} 255, & \text{se } p > T \\ 0, & \text{se } p \leq T \end{cases}$$

5. **Output:** Visualizzare la matrice binarizzata con dimensioni $L \times C$.

4.9.1.2 📌 Vincoli Computazionali

- **Binarizzazione:** L'output contiene **solo** i valori 0 o 255.
- **Confronto rigoroso:** Il criterio usa $> T$ (i pixel uguali a T diventano sfondo).
- **Tipo:** Il risultato finale deve essere intero.
- **Osservazione:** Questo EP segue la convenzione di OpenCV (`cv2.THRESH_BINARY`): solo i pixel con valore **maggiore di** T diventano bianchi (`255`); i pixel con valore **uguale a** T restano neri (`0`).

4.9.1.3 🧠 Fondamenti Teorici

Parametro	Tipo	Impatto Visivo
T **piccolo**	Intero	La maggior parte dei pixel diventa bianca
T **grande**	Intero	La maggior parte dei pixel diventa nera
T **ben scelto**	Intero	Separa nettamente oggetto e sfondo

4.9.1.4 📦 Specifica di Input e Output (VPL)

Input:

- Riga 1: Intero L.
- Riga 2: Intero C.
- Riga 3: Intero T.
- Righe successive: Elementi interi della matrice originale.

Output:

- Matrice binarizzata in L righe e C colonne, valori 0 o 255 separati da spazi.

4.9.1.5 📌 Esempi

Input	Output	Osservazione
2 4 100 0 99 100 180 255 30 120 80	0 0 0 255 255 0 255 0	$T = 100$: solo i pixel con valore maggiore di 100 diventano bianchi; pertanto, 99 e 100 diventano neri.
1 3 0 0 50 255	0 255 255	$T = 0$: solo i pixel con valore strettamente maggiore di 0 diventano bianchi.

```
%%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.it/cap04/sim-ep0401-limiar.html

Figura 4.30: Simulatore EP04_01: Soglia Globale con Soglia Fissa (p' = (p > T) ? 255 : 0)

4.9.2 EP04_02 📊 Soglia Automatica di Otsu

Scegliere manualmente la soglia T funziona quando l'illuminazione è stabile, ma in **microscopia digitale** e nell'**ispezione di strisci di sangue**, ogni campione ha un contrasto diverso — una soglia fissa fallirebbe da un'immagine all'altra. Il **metodo di Otsu** risolve questo problema trovando, da solo, la soglia che **massimizza la separazione statistica** tra le due classi di pixel, rendendo la segmentazione automatica e adattiva. Vedi in Figura 4.31 una simulazione di questo EP.

4.9.2.1 📋 Linee Guida di Implementazione

1. **Dimensioni:** Leggere gli interi L (righe) e C (colonne).
2. **Dati:** Leggere i valori interi della matrice originale riga per riga.
3. **Istogramma:** Costruire l'istogramma $h[i]$, $i = 0, \ldots, 255$, contando quanti pixel hanno valore i.
4. **Ricerca della soglia:** Per ogni candidato T da 1 a 255, calcolare la **varianza tra le classi**:

$$\sigma_B^2(T) = \frac{n_0 \cdot n_1}{N^2} (m_0 - m_1)^2$$

   dove n_0, n_1 sono le quantità di pixel con valore $< T$ e $\geq T$, m_0, m_1 sono le loro medie, e $N = L \times C$.
5. **Scelta:** La soglia ottimale T^* è quella che massimizza $\sigma_B^2(T)$ (in caso di pareggio, mantenere la **prima** trovata).
6. Applicazione: Binarizzare l'immagine usando T*, applicando:

$$p' = \begin{cases} 255, & \text{se } p > T^* \\ 0, & \text{se } p \leq T^* \end{cases}$$

4.9.2.2 📌 Vincoli Computazionali

- **Candidati validi:** Ignorare T che lasci $n_0 = 0$ o $n_1 = 0$ (classe vuota).

- **Pareggio:** Mantenere sempre la **prima** T che ha raggiunto il valore massimo di σ_B^2.
- **Tipo:** T^* e la matrice di uscita devono essere interi.
- **Convenzione OpenCV:** La binarizzazione segue `cv2.THRESL_BINARY`; i pixel con valore esattamente uguale a T^* diventano neri.

4.9.2.3 🧠 Fondamenti Teorici

Concetto	Significato	Impatto
$\sigma_B^2(T)$ **alta**	Classi ben separate in T	T è un buon candidato come soglia
Istogramma bimodale	Due "colline" distinte	Otsu trova la valle tra di esse
Istogramma unimodale	Una sola "collina"	Otsu sceglie comunque *qualche* T, ma la segmentazione è poco affidabile

4.9.2.4 📦 Specifica di Input e Output (VPL)

Input:

- Riga 1: Intero L.
- Riga 2: Intero C.
- Righe successive: Elementi interi della matrice originale.

Output:

- Matrice binarizzata in L righe e C colonne, valori 0 o 255.

4.9.2.5 📌 Esempi

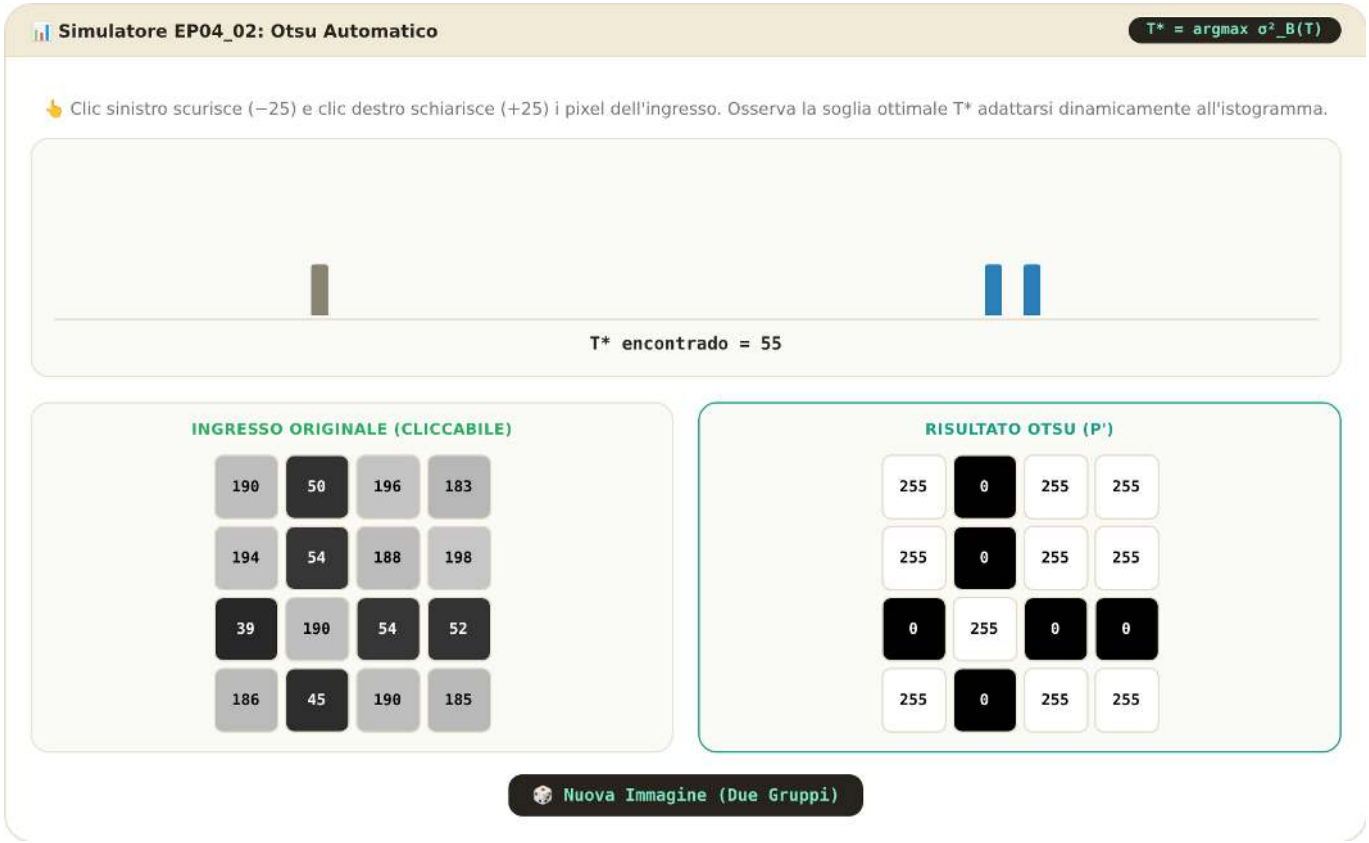
Input	Output	Osservazione
4 4 12 12 12 200 12 12 200 200 12 200 200 200 200 200 200 200	0 0 0 255 0 0 255 255 0 255 255 255 255 255 255 255	Istogramma bimodale chiaro: 12 e 200
1 2 10 250	0 250	Solo due valori: T^* si colloca sul maggiore

```
%%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.it/cap04/sim-ep0402-otsu.html

Figura 4.31: Simulatore EP04_02: Sogliatura Automatica di Otsu (T* = argmax ²_B(T))

4.9.3 EP04_03 🌱 Dilatazione Binaria Piana (mm.dil0)

Nella **microscopia di particelle** e nell'**OCR di targhe automobilistiche usurate**, tratti sottili o discontinui devono essere "ingrossati" affinché il riconoscimento funzioni. La **dilatazione morfologica** fa esattamente questo: espande le regioni chiare utilizzando un elemento strutturante B — la stessa operazione implementata in `morph.py` come `mm::dil0(f, B)`, utilizzata quando B è **piano** (senza pesi, solo 0/1). Vedi in Figura 4.32 una simulazione di questo EP.

4.9.3.1 📋 Linee Guida di Implementazione

1. **Dimensioni dell'immagine:** Leggere gli interi L (righe) e C (colonne) di f.
2. **Dimensioni di B:** Leggere gli interi L_B (righe) e C_B (colonne) dell'elemento strutturante.
3. **Elemento strutturante:** Leggere la matrice B con valori 0 o 1, riga per riga.
4. **Dati:** Leggere la matrice f (l'immagine originale), riga per riga.
5. **Riflessione:** Costruire B_{ref}, la versione di B riflessa di 180° (righe e colonne invertite) — esattamente come fa `mm::dil0` internamente.
6. **Vicinato senza padding:** Per ogni pixel (y, x), percorrere le posizioni (by, bx) di B_{ref} centrate su (y, x), usando lo spostamento

$$v_y = y + by + o_y, \quad v_x = x + bx + o_x, \quad o_y = -\frac{L_B}{2} + 0{,}5, \quad o_x = -\frac{C_B}{2} + 0{,}5$$

   Scartare ogni (v_y, v_x) al di fuori di $[0, L) \times [0, C)$ — **non riempire con zeri**.
7. **Mappatura:** Calcolare ogni pixel di uscita come il **massimo** tra $f(y, x)$ e tutti gli $f(v_y, v_x)$ validi la cui posizione corrispondente in B_{ref} vale 1:

$$g(y, x) = \max\left(f(y, x), \max_{\substack{(v_y, v_x) \text{ valido} \\ B_{ref}(by, bx)=1}} f(v_y, v_x)\right)$$

8. **Uscita:** Visualizzare la matrice g con dimensioni $L \times C$.

4.9.3.2 📌 Vincoli Computazionali

- **Senza padding:** Non inventare mai vicini al di fuori dell'immagine; utilizzare solo quelli che esistono realmente.
- **Riflessione obbligatoria:** B deve essere riflesso prima dell'applicazione (è ciò che distingue `mm::dil0` da una semplice ricerca del massimo).
- **Robustezza ai bordi:** Se nessuna posizione valida di $B_{ref} = 1$ ricade all'interno del dominio per un dato pixel, questo **mantiene il suo valore originale**.

4.9.3.3 🧠 Fondamento Teorico

Concetto	Significato	Impatto Visivo
Dilatazione	$g \geq f$ sempre (estensiva)	Le regioni chiare crescono, i buchi scuri si restringono
B **più grande**	Vicinato più ampio	Crescita più aggressiva
Riflessione di B	$B_{ref}(y, x) = B(-y, -x)$	Garantisce la definizione formale di Minkowski della dilatazione

4.9.3.4 📦 Specifica di Input e Output (VPL)

Input:

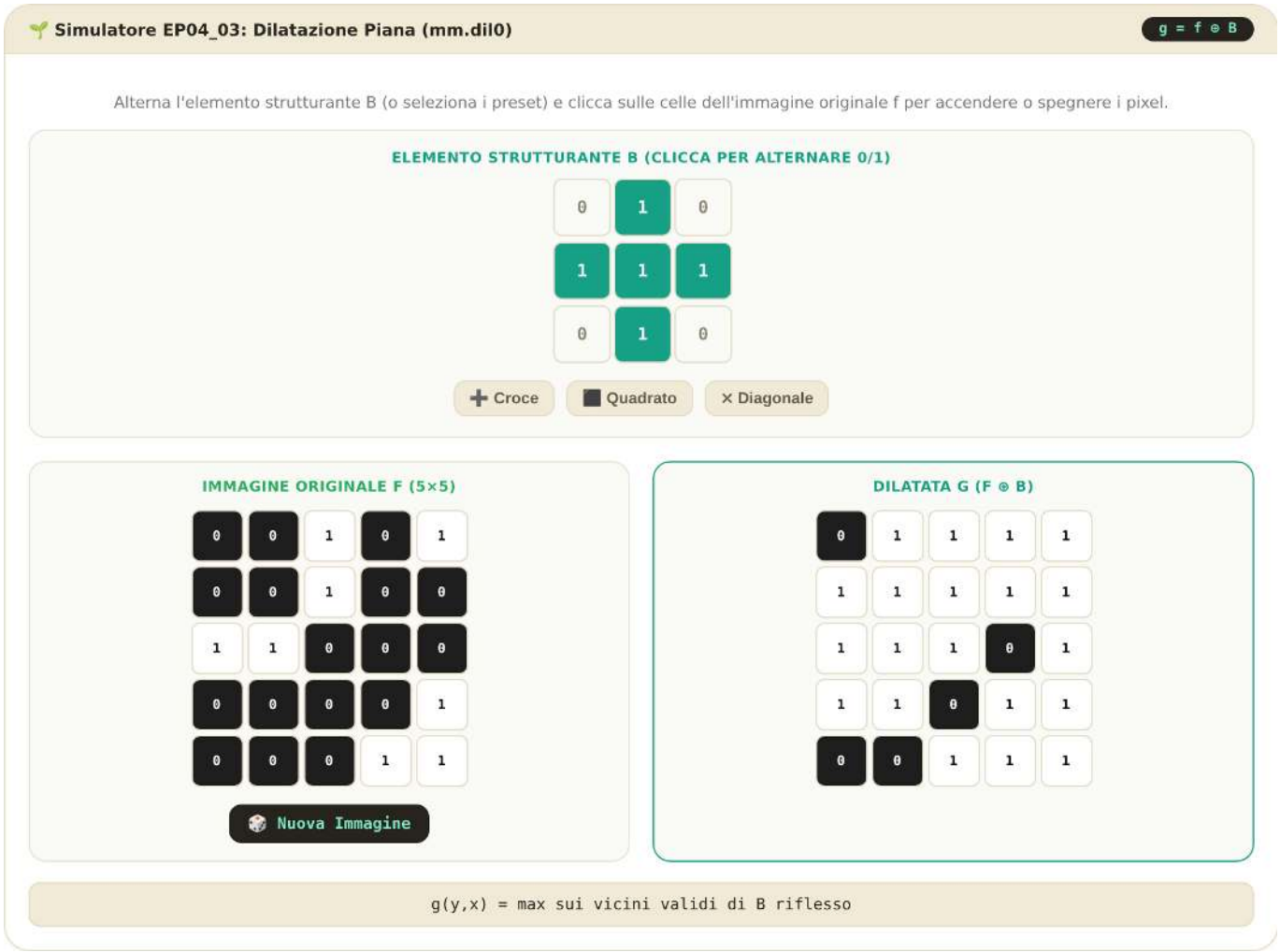
- Riga 1: Intero L.
- Riga 2: Intero C.
- Riga 3: Intero L_B.
- Riga 4: Intero C_B.
- Successive L_B righe: elementi interi (0 o 1) della matrice B.
- Successive L righe: elementi interi della matrice f.

Output:

- Matrice g in L righe e C colonne, valori interi separati da spazio.

4.9.3.5 📌 Esempi

Input	Output	Osservazione
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 a croce simmetrico: punto isolato si espande a croce
1 4 1 3 1 1 1 10 200 5 80	200 200 200 80	B orizzontale: ogni pixel "attira" il massimo dei vicini della riga



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.it/cap04/sim-ep0403-dilatacao.html

Figura 4.32: Simulatore EP04_03: Dilatazione Binaria Piana (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 🪨 Erosione Binaria Piatta (mm.ero0)

Se la dilatazione ingrossa, l'**erosione** assottiglia. Nei **sistemi di conteggio cellulare**, viene utilizzata per **separare cellule che si toccano**: "consumando" i bordi di ogni regione, le connessioni sottili tra gli oggetti scompaiono prima ancora che venga effettuato qualsiasi conteggio. In `morph.py`, questa è l'operazione `mm::ero0(f, B)` — il **duale** esatto della dilatazione, e l'unica delle due che **non** riflette l'elemento strutturante. Vedi nella Figura 4.33 una simulazione di questo EP.

4.9.4.1 📋 Linee Guida di Implementazione

1. **Dimensioni dell'immagine:** Leggere gli interi L (righe) e C (colonne) da f.
2. **Dimensioni di B:** Leggere gli interi L_B (righe) e C_B (colonne) dell'elemento strutturante.
3. **Elemento strutturante:** Leggere la matrice B con valori 0 o 1, riga per riga.
4. **Dati:** Leggere la matrice f (l'immagine originale), riga per riga.
5. **Vicinato senza padding (senza riflessione!):** Per ogni pixel (y, x), percorrere le posizioni (by, bx) di B **nell'ordine originale** (senza riflettere), usando lo stesso spostamento dell'EP04_03:

$$v_y = y + by + o_y, \quad v_x = x + bx + o_x, \quad o_y = -\tfrac{L_B}{2} + 0{,}5, \quad o_x = -\tfrac{C_B}{2} + 0{,}5$$

Scartare ogni (v_y, v_x) al di fuori di $[0, L) \times [0, C)$. 6. **Mappatura:** Calcolare ogni pixel di uscita come il **minimo** tra $f(y, x)$ e tutti gli $f(v_y, v_x)$ validi la cui posizione corrispondente in B vale 1:

$$g(y,x) = \min\Big(f(y,x),\ \min_{\substack{(v_y,v_x)\ \text{valido}\\ B(by,bx)=1}} f(v_y,v_x)\Big)$$

7. **Uscita:** Visualizzare la matrice g con dimensioni $L \times C$.

4.9.4.2 📌 Vincoli Computazionali

- **Senza riflessione:** Diversamente dalla dilatazione, B viene utilizzato **esattamente come letto** — riflettere qui sarebbe un errore concettuale grave.
- **Senza padding:** I vicini al di fuori dell'immagine vengono semplicemente ignorati, mai trattati come 0.
- **Robustezza ai bordi:** Se nessuna posizione valida di $B = 1$ ricade all'interno del dominio, il pixel mantiene il suo valore originale.

4.9.4.3 🧠 Fondamenti Teorici

Concetto	Significato	Impatto Visivo
Erosione	$g \leq f$ sempre (anti-estensiva)	Le regioni chiare si restringono, il rumore puntuale scompare
Dualità	$\text{ero}(f, B) = -\text{dil}(-f, B_{ref})$	Erosione e dilatazione sono "specchi" matematici
B **più grande**	Erosione più aggressiva	Gli oggetti sottili scompaiono completamente

4.9.4.4 📦 Specifica di Ingresso e Uscita (VPL)

Ingresso:

- Riga 1: Intero L.
- Riga 2: Intero C.
- Riga 3: Intero L_B.
- Riga 4: Intero C_B.
- Prossime L_B righe: elementi interi (0 o 1) della matrice B.
- Prossime L righe: elementi interi della matrice f.

Uscita:

- Matrice g in L righe e C colonne, valori interi separati da spazio.

4.9.4.5 📌 Esempi

Ingresso	Uscita	Osservazione
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	Il “buco” centrale (0) si propaga a croce
1 4 1 3 1 1 1 10 200 5 80	10 5 5 80	B orizzontale: ogni pixel “attira” il minimo dei vicini della riga

```
%%writefile EP04_04.cpp
// your solution
```

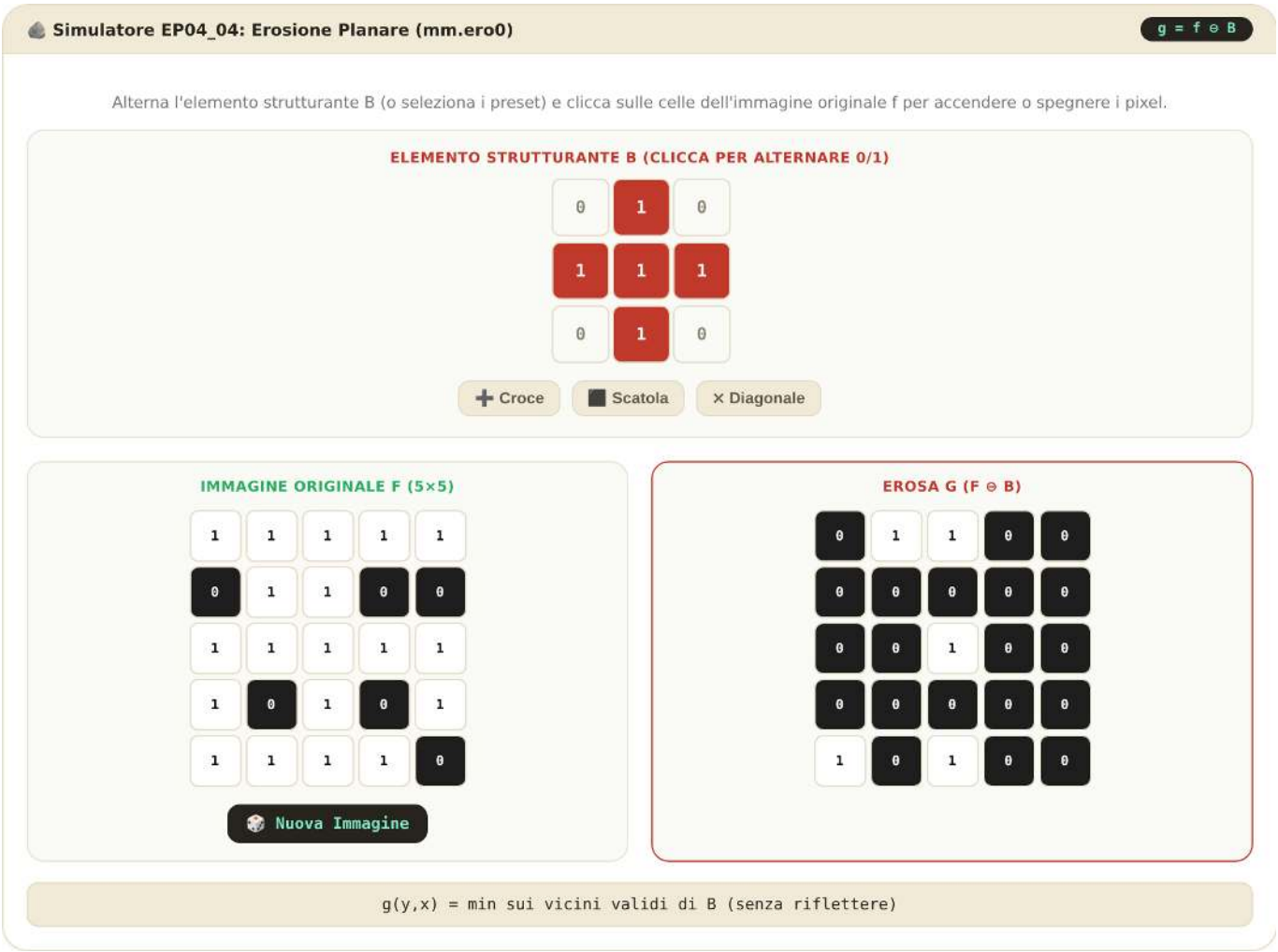
```
Overwriting EP04_04.cpp
```

```
TestSuite("EP04_04.cpp").run()
```

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

4.9.5 EP04_05 🧹 Apertura Morfologica (Rimozione del Rumore)

Le immagini acquisite da **sensori a basso costo**, come quelli dei droni agricoli, sono spesso disseminate di piccoli punti di rumore — pixel isolati che non rappresentano nulla di reale. Applicare l’erosione seguita dalla dilatazione con lo **stesso** elemento strutturante produce l’**apertura**: essa “pulisce” punti e sottili protuberanze, ma restituisce all’oggetto principale praticamente la sua dimensione originale. È la combinazione classica utilizzata nella **pre-elaborazione di immagini satellitari** prima di qualsiasi conteggio dell’area coltivata. Vedi in Figura 4.34 una simulazione di questo EP.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.it/cap04/sim-ep0404-erosao.html

Figura 4.33: Simulatore EP04_04: Erosione Binaria Piana (g = f B)

4.9.5.1 📋 Linee Guida di Implementazione

1. **Dimensioni dell'immagine:** Leggere gli interi L (righe) e C (colonne) da f.
2. **Dimensioni di B:** Leggere gli interi L_B (righe) e C_B (colonne) dell'elemento strutturante.
3. **Elemento strutturante:** Leggere la matrice B con valori 0 o 1, riga per riga.
4. **Dati:** Leggere la matrice binaria f (valori 0 o 1), riga per riga.
5. **Erosione:** Calcolare $e = f \ominus B$, usando esattamente l'algoritmo di EP04_04 (senza riflettere B, senza padding).
6. **Dilatazione:** Calcolare $g = e \oplus B$, usando esattamente l'algoritmo di EP04_03 (riflettendo B, senza padding) — ma ora applicato su e, non su f.
7. **Output:** Visualizzare la matrice risultante g (l'**apertura** di f tramite B) con dimensioni $L \times C$.

4.9.5.2 📌 Vincoli Computazionali

- **Ordine fisso:** È **sempre** prima erosione, poi dilatazione — l'ordine inverso definisce un altro operatore (chiusura, nel prossimo EP).
- **Stesso B:** L'elemento strutturante utilizzato nell'erosione e nella dilatazione deve essere identico.
- **Nessun padding in nessuna delle due fasi.**

4.9.5.3 🧠 Fondamenti Teorici

Concetto	Significato	Impatto Visivo
Anti-estensività	$g \subseteq f$ sempre	L'apertura non crea mai nuovi pixel, li rimuove soltanto
Idempotenza	apertura(apertura(f)) = apertura(f)	Applicarla di nuovo non cambia più nulla
Punti isolati	Più piccoli di B	Vengono completamente eliminati
Nucleo dell'oggetto	Più grande di B	Viene recuperato quasi intatto dalla dilatazione finale

4.9.5.4 📦 Specifica di Input e Output (VPL)

Input:

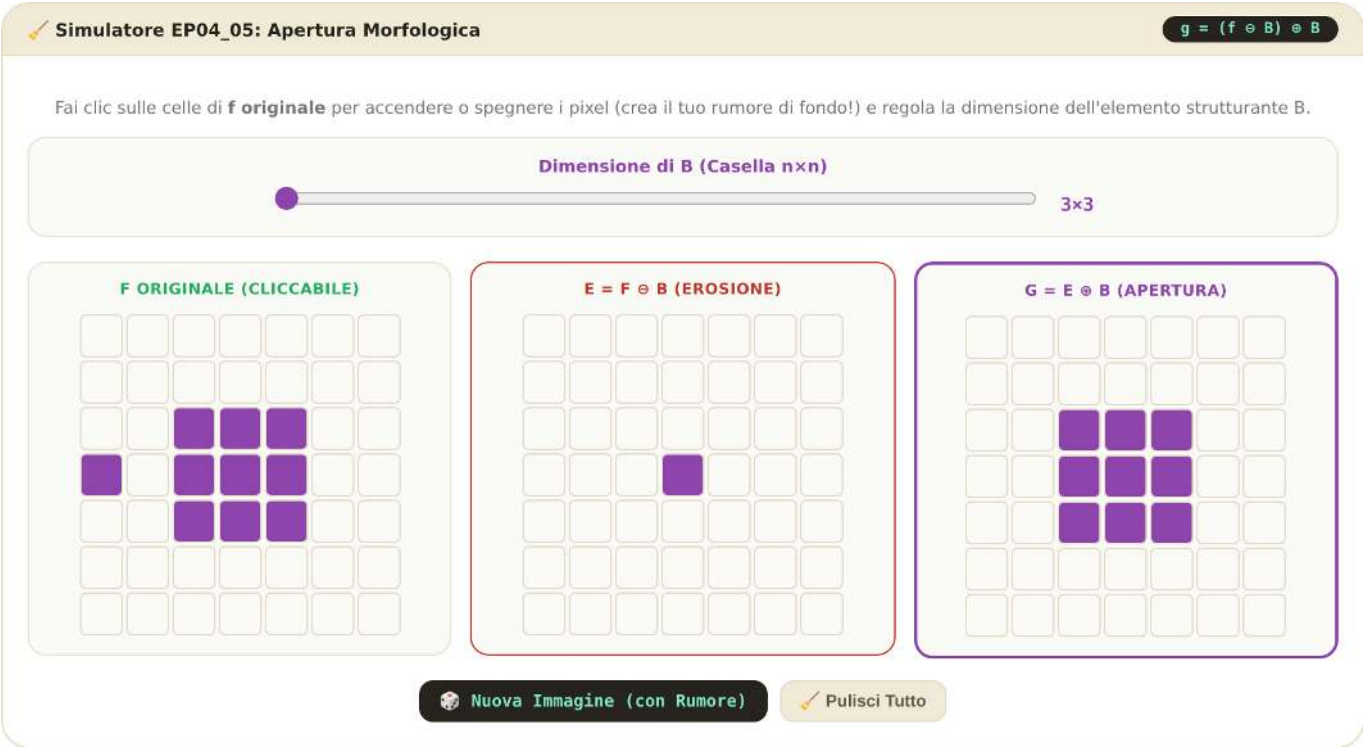
- Riga 1: Intero L.
- Riga 2: Intero C.
- Riga 3: Intero L_B.
- Riga 4: Intero C_B.
- Prossime L_B righe: elementi interi (0 o 1) della matrice B.
- Prossime L righe: elementi interi (0 o 1) della matrice f.

Output:

- Matrice risultante in L righe e C colonne, valori 0 o 1.

4.9.5.5 📌 Esempi

Input	Output	Osservazione
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	Punti isolati e la sottile protuberanza scompaiono; il quadrato centrale sopravvive



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.it/cap04/sim-ep0405-abertura.html

Figura 4.34: Simulatore EP04_05: Apertura Morfologica (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 🧩 Chiusura Morfologica (Riempimento delle Lacune)

Nella **digitalizzazione delle impronte digitali**, i solchi della pelle talvolta vengono interrotti da sporco o secchezza, creando piccole lacune nella curva continua che dovrebbe esistere. La **chiusura** — dilatazione seguita da erosione con lo stesso elemento strutturante — è l'operatore duale dell'apertura: **riempie piccoli buchi e rientranze strette**, senza alterare significativamente il contorno esterno dell'oggetto. È la fase standard prima di estrarre lo scheletro di un'impronta digitale. Vedi una simulazione di questo EP in Figura 4.35.

4.9.6.1 📋 Linee Guida di Implementazione

1. **Dimensioni dell'immagine:** Leggere gli interi L (righe) e C (colonne) da f.
2. **Dimensioni di B:** Leggere gli interi L_B (righe) e C_B (colonne) dell'elemento strutturante.
3. **Elemento strutturante:** Leggere la matrice B con valori 0 o 1, riga per riga.
4. **Dati:** Leggere la matrice binaria f (valori 0 o 1), riga per riga.
5. **Dilatazione:** Calcolare $d = f \oplus B$, usando esattamente l'algoritmo del EP04_03 (riflettendo B, senza padding).
6. **Erosione:** Calcolare $g = d \ominus B$, usando esattamente l'algoritmo del EP04_04 (senza riflettere B, senza padding) — ora applicato su d, non su f.
7. **Uscita:** Visualizzare la matrice risultante g (la **chiusura** di f per B) con dimensioni $L \times C$.

4.9.6.2 📌 Vincoli Computazionali

- **Ordine fisso:** È **sempre** prima la dilatazione, poi l'erosione — l'ordine inverso è l'apertura del EP04_05.
- **Stesso B:** L'elemento strutturante usato nella dilatazione e nell'erosione deve essere identico.
- **Nessun padding in nessuna delle due fasi.**

4.9.6.3 🧠 Fondamento Teorico

Concetto	Significato	Impatto Visivo
Estensività	$g \supseteq f$ sempre	La chiusura non rimuove mai pixel, solo aggiunge
Idempotenza	chiudi(chiudi(f)) = chiudi(f)	Applicare di nuovo non cambia più nulla
Piccoli buchi	Minori di B	Vengono completamente riempiti
Dualità	chiudi(f) = $\overline{\text{apri}(\bar{f})}$	È l'apertura applicata al "negativo" dell'immagine

4.9.6.4 📦 Specifica di Input e Output (VPL)

Input:

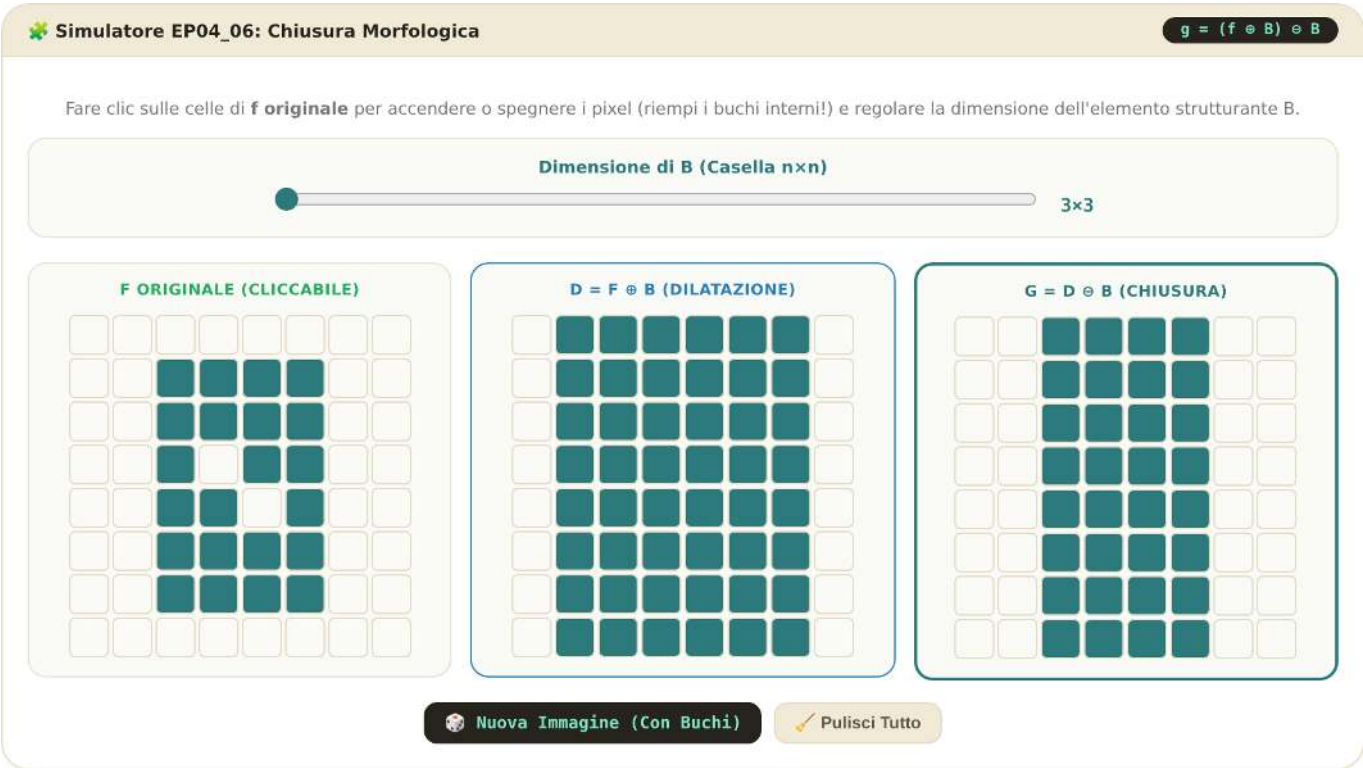
- Riga 1: Intero L.
- Riga 2: Intero C.
- Riga 3: Intero L_B.
- Riga 4: Intero C_B.
- Prossime L_B righe: elementi interi (0 o 1) della matrice B.
- Prossime L righe: elementi interi (0 o 1) della matrice f.

Output:

- Matrice risultante in L righe e C colonne, valori 0 o 1.

4.9.6.5 📌 Esempi

Input	Output	Osservazione
8 8 3 3 1 1 1 1 1 1 1 1 1 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	I due buchi interni non adiacenti vengono completamente riempiti



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.it/cap04/sim-ep0406-fechamento.html

Figura 4.35: Simulatore EP04_06: Chiusura Morfologica (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 Dilatazione ed Erosione con Pesi (mm.dil1 / mm.ero1)

Finora, l'elemento strutturante diceva solo "questo vicino conta" o "non conta" — ma nei **modelli digitali di elevazione** (usati nei GIS e nella pianificazione del drenaggio urbano), ogni vicino dovrebbe avere un **peso diverso** a seconda della distanza o della direzione del rilievo. Le versioni **ponderate** della dilatazione e dell'erosione, implementate in `morph.py` come `mm::dil1(f, b)` e `mm::ero1(f, b)`, sommano (o sottraggono) il peso di ciascun vicino prima di prendere il massimo (o il minimo) — generalizzando tutto ciò che è stato fatto negli EP precedenti. Vedere in Figura 4.36 una simulazione di questo EP.

4.9.7.1 📋 Linee Guida di Implementazione

1. **Dimensioni dell'immagine:** Leggere gli interi L (righe) e C (colonne) di f.
2. **Dimensioni di b:** Leggere gli interi L_B (righe) e C_B (colonne) dell'elemento strutturante ponderato.
3. **Pesi:** Leggere la matrice b di pesi **interi** (possono essere negativi, zero o positivi), riga per riga.
4. **Dati:** Leggere la matrice f (l'immagine originale), riga per riga.
5. **Vicinato senza padding:** Per ogni pixel (y, x), percorrere **tutte** le posizioni (by, bx) di b (non solo dove varrebbe 1 — qui **tutto** il peso partecipa), usando lo stesso spostamento degli EP precedenti:

$$v_y = y + by + o_y, \quad v_x = x + bx + o_x, \quad o_y = -\tfrac{L_B}{2} + 0{,}5, \quad o_x = -\tfrac{C_B}{2} + 0{,}5$$

   Scartare ogni (v_y, v_x) fuori da $[0, L) \times [0, C)$.
6. **Dilatazione ponderata:** Calcolare

$$g_{dil}(y, x) = \max\Big(f(y, x), \max_{(v_y, v_x)\ \text{valido}} \big(f(v_y, v_x) + b(by, bx)\big)\Big)$$

7. **Erosione ponderata:** Calcolare, **usando lo stesso b e senza riflessione**:

$$g_{ero}(y, x) = \min\Big(f(y, x), \min_{(v_y, v_x)\ \text{valido}} \big(f(v_y, v_x) - b(by, bx)\big)\Big)$$

8. **Uscita:** Mostrare **prima** la matrice completa g_{dil}, e **poi** la matrice completa g_{ero}.

4.9.7.2 📌 Vincoli Computazionali

- **Nessuna delle due riflette** b — la versione ponderata non usa riflessione, nemmeno nella dilatazione (diversamente da `mm::dil0`).
- **Tutti i pesi partecipano:** Non esiste qui il filtro "$B = 1$"; anche il peso 0 entra nel calcolo.
- **Senza padding:** i vicini fuori dall'immagine sono ignorati, mai virtualmente riempiti.
- **Tipo:** L'uscita può contenere valori negativi o maggiori di 255 — **non** c'è *clipping* in questo EP.
- **Suggerimento:** Per rimuovere i messaggi di overflow quando si superano i limiti del tipo uint8, includere all'inizio del codice:

```
import warnings
warnings.filterwarnings("ignore")
```

4.9.7.3 🧠 Fondamenti Teorici

Concetto	Significato	Impatto Visivo
Peso positivo	"Spinge" il valore del vicino verso l'alto nella dilatazione	Simula un rilievo che sale in quella direzione
Peso negativo	Riduce il contributo del vicino	Simula distanza o attenuazione direzionale
Dualità ponderata	$\text{ero1}(f, b) = -\text{dil1}(-f, b)$	La simmetria tra le due operazioni si mantiene anche con i pesi

4.9.7.4 📦 Specifica di Input e Output (VPL)

Input:

- Riga 1: Intero L.
- Riga 2: Intero C.
- Riga 3: Intero L_B.
- Riga 4: Intero C_B.
- Prossime L_B righe: elementi interi (possono essere negativi) della matrice b.
- Prossime L righe: elementi interi della matrice f.

Output:

- Prima la matrice g_{dil} in L righe e C colonne.
- Successivamente la matrice g_{ero} in L righe e C colonne.

4.9.7.5 📌 Esempi

Input	Output	Osservazione
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	Il peso centrale 2 accelera la crescita nella dilatazione e il restringimento nell'erosione

```
%%writefile EP04_07.cpp
// your solution
```

```
Overwriting EP04_07.cpp
```

```
TestSuite("EP04_07.cpp").run()
```

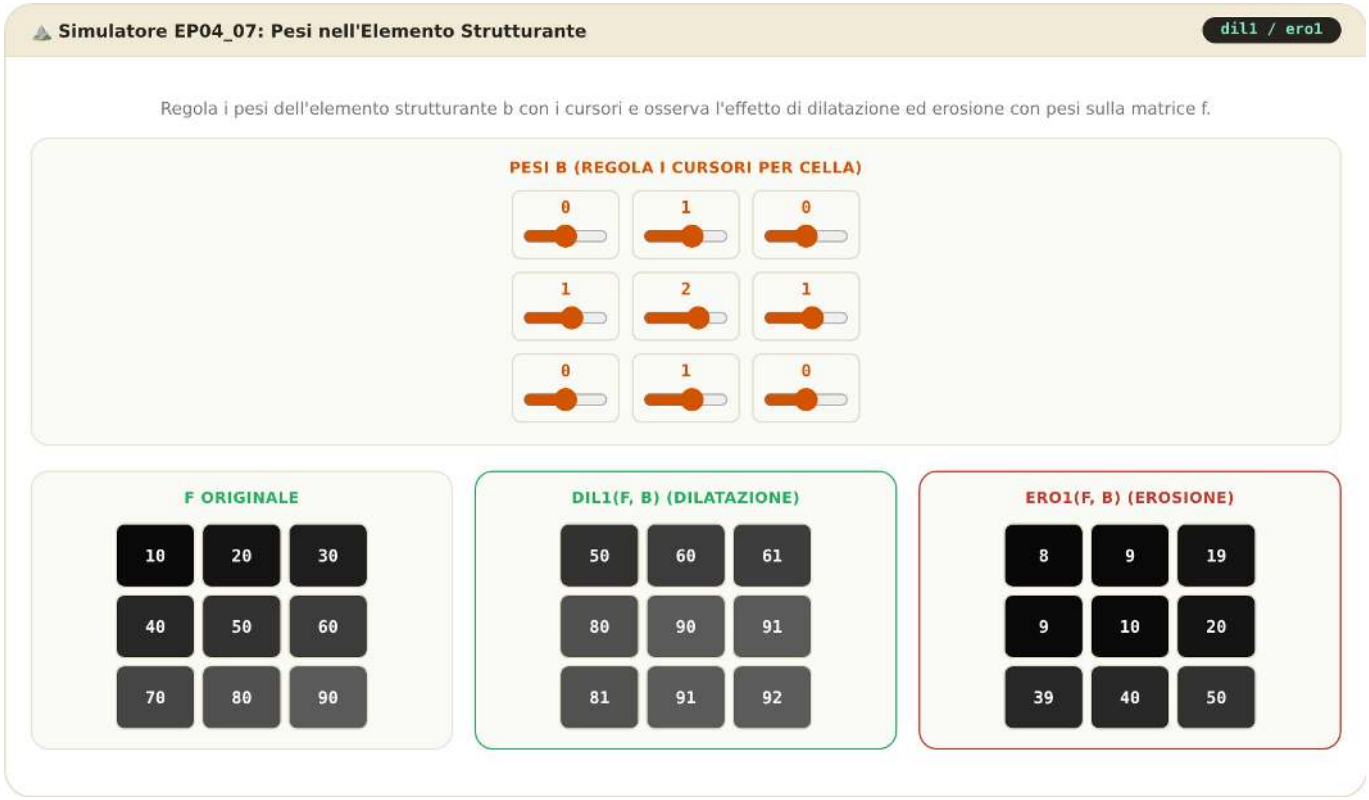
```
<IPython.core.display.HTML object>
```

4.9.8 EP04_08 🌋 Gradiente Morfologico, Top-hat e Black-hat

Nell'**ispezione automatica dei circuiti stampati**, tre domande ricorrono continuamente: dove sono i **bordi** dei componenti? Quali **dettagli chiari e piccoli** (come i punti di saldatura) si distinguono dallo sfondo? Quali **incavi scuri** (come le cricche) lo sfondo nasconde? Una singola coppia erosione/dilatazione risponde a tutte e tre: il **gradiente morfologico** evidenzia i contorni, il **top-hat** rivela i picchi stretti, e il **black-hat** rivela le valli strette — tre strumenti, un solo intorno. Vedi in Figura 4.37 una simulazione di questo EP.

4.9.8.1 📋 Linee Guida di Implementazione

1. **Dimensioni dell'immagine:** Leggere gli interi L (righe) e C (colonne) di f.
2. **Dimensioni di B:** Leggere gli interi L_B (righe) e C_B (colonne) dell'elemento strutturante.
3. **Elemento strutturante:** Leggere la matrice B con valori 0 o 1, riga per riga.
4. **Dati:** Leggere la matrice f (l'immagine originale, in scala di grigi), riga per riga.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.it/cap04/sim-ep0407-pesos.html

Figura 4.36: Simulatore EP04_07: Dilatazione ed Erosione con Pesi (mm.dil1 / mm.ero1)

5. **Operatori di base:** Calcolare, esattamente come negli EP 04_03 fino a 04_06:
   - $d = f \oplus B$ (dilatazione),
   - $e = f \ominus B$ (erosione),
   - apertura $= e \oplus B$,
   - chiusura $= d \ominus B$.
6. **Gradiente morfologico:** $\text{grad}(y, x) = d(y, x) - e(y, x)$.
7. **Top-hat:** $\text{tophat}(y, x) = f(y, x) - \text{apertura}(y, x)$.
8. **Black-hat:** $\text{blackhat}(y, x) = \text{chiusura}(y, x) - f(y, x)$.
9. **Output:** Mostrare, **in questo ordine**, le tre matrici complete: gradiente, top-hat, black-hat.

4.9.8.2 📌 Vincoli Computazionali

- **Nessun padding in nessuna fase intermedia** — dilatazione, erosione, apertura e chiusura seguono le stesse regole di vicinato degli EP precedenti.
- **Nessun *clipping*:** le tre uscite possono contenere qualsiasi valore intero (il gradiente è sempre ≥ 0, ma anche top-hat e black-hat lo sono).
- **Riutilizzo:** d e e devono essere calcolati **una sola volta** e riutilizzati per costruire apertura, chiusura e gradiente.

4.9.8.3 🧠 Fondamenti Teorici

Operatore	Formula	Cosa rivela
Gradiente	$d - e$	Bordi: zero in regioni piatte, alto nelle transizioni
Top-hat	$f - \text{apertura}(f)$	Elementi **chiari e sottili**, più piccoli di B

Operatore	Formula	Cosa rivela
Black-hat	chiusura$(f) - f$	Elementi **scuri e sottili**, più piccoli di B

4.9.8.4 📦 Specifica di Input e Output (VPL)

Input:

- Riga 1: Intero L.
- Riga 2: Intero C.
- Riga 3: Intero L_B.
- Riga 4: Intero C_B.
- Prossime L_B righe: elementi interi (0 o 1) della matrice B.
- Prossime L righe: elementi interi della matrice f.

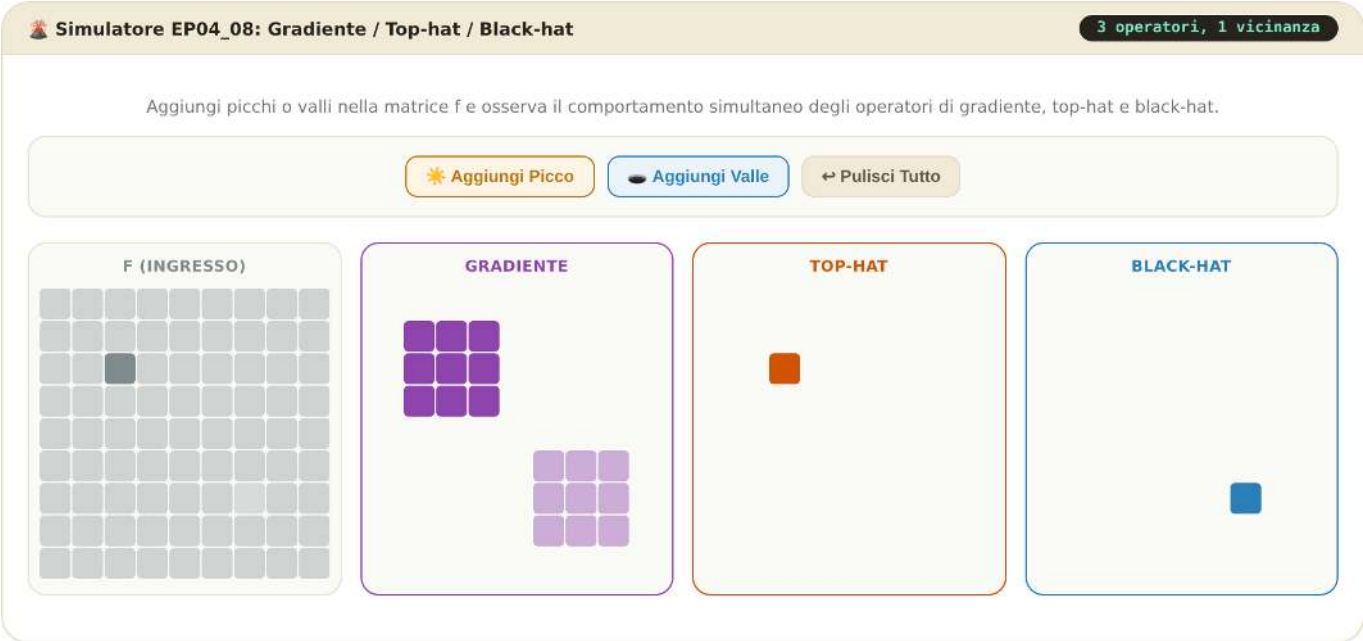
Output:

- Matrice gradiente in L righe e C colonne.
- Matrice top-hat in L righe e C colonne.
- Matrice black-hat in L righe e C colonne.

4.9.8.5 📌 Esempi

Input	Output	Osservazione
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	(gradiente: alone $3 \times 3 = 70$ attorno a $(2, 2)$ e alone $3 \times 3 = 8$ attorno a $(6, 6)$, resto 0) (top-hat: unico 70 in $(2, 2)$, resto 0) (black-hat: unico 8 in $(6, 6)$, resto 0)	Picco isolato diventa top-hat; valle isolata diventa black-hat; entrambi appaiono nel gradiente

```
%%writefile EP04_08.cpp
// your solution
```



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.it/cap04/sim-ep0408-gradiente.html

Figura 4.37: Simulatore EP04_08: Gradiente morfologico, Top-hat e Black-hat

```
Overwriting EP04_08.cpp
```

```
TestSuite("EP04_08.cpp").run()
```

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

4.9.9 EP04_09 Trasformata della Distanza e il "Nucleo" dell'Oggetto

Nella **robotica mobile**, quando si pianifica un percorso all'interno di un corridoio, il robot vuole sapere non solo *dove* c'è spazio libero, ma anche **quanto distante** ogni punto libero sia dalla parete più vicina. I percorsi più sicuri tendono a passare attraverso il "nucleo" del corridoio, lontano dagli ostacoli.

La **trasformata della distanza morfologica** assegna a ogni pixel un valore che rappresenta la sua distanza dal bordo più vicino, secondo la metrica definita dall'elemento strutturante. I pixel vicini al bordo ricevono valori bassi, mentre i pixel più interni ricevono valori più alti. Il pixel con il valore massimo corrisponde alla regione più protetta dell'oggetto, spesso associata al suo centro morfologico.

Vedere in Figura 4.38 una simulazione di questo EP.

4.9.9.1 📋 Linee Guida di Implementazione

1. **Dimensioni dell'immagine:** leggere gli interi L (righe) e C (colonne) dell'immagine f.
2. **Dimensioni di B:** leggere gli interi L_B (righe) e C_B (colonne) dell'elemento strutturante.
3. **Elemento strutturante:** leggere la matrice b, contenente valore 0 al centro e valori negativi nelle altre posizioni.
4. **Immagine:** leggere la matrice binaria f (valori 0 o 1), riga per riga.
5. **Preparazione:** moltiplicare l'immagine per $L \times C$, garantendo che i pixel interni abbiano un valore iniziale sufficientemente alto per la propagazione delle distanze.
6. **Trasformata della distanza:** calcolare la matrice delle distanze utilizzando il metodo `mm::dist1(f,b)`.
7. **Output:** visualizzare la matrice risultante dalla trasformata della distanza.

4.9.9.2 📌 Vincoli Computazionali

- Utilizzare l'implementazione dell'erosione ponderata fornita dalla libreria.
- L'elemento strutturante può contenere valori negativi arbitrari.
- La trasformata deve essere ottenuta applicando iterativamente erosioni ponderate fino a raggiungere un punto fisso.

⚠️ **Nota Cruciale sulla Lettura delle Matrici:** Poiché l'elemento strutturante può contenere valori interi negativi (ad esempio, `-1` e `-99`), **non utilizzare la funzione `mm::readImg` per leggere la matrice** b. Questa funzione converte i dati nel tipo `uint8`, causando *underflow* e corrompendo i valori negativi. Leggere le L_B righe di b manualmente utilizzando il tipo predefinito `int`. L'immagine f può continuare a essere letta normalmente con `mm::readImg`.

4.9.9.3 🧠 Fondamenti Teorici

Concetto	Significato	Impatto Visivo
dist(y, x)	Distanza morfologica fino al bordo più vicino secondo la metrica definita da b	I pixel più interni ricevono valori più alti
Valore massimo	Pixel più distante dal bordo	Approssima il centro morfologico dell'oggetto
Elemento strutturante ponderato	Definisce i costi di spostamento tra pixel vicini	Determina la metrica della distanza utilizzata
Oggetti sottili	Regioni strette dell'oggetto	Producono valori bassi di distanza

4.9.9.4 📦 Specifica di Input e Output (VPL)

Input:

- Riga 1: intero L.
- Riga 2: intero C.
- Riga 3: intero L_B.
- Riga 4: intero C_B.
- Prossime L_B righe: elementi interi della matrice b.
- Prossime L righe: elementi binari (0 o 1) della matrice f.

⚠️ **Nota di implementazione:** Gli elementi della matrice f (0 o 1) devono essere moltiplicati per **255** per generare un'immagine binaria adeguata (0 e 255) prima di applicare la Trasformata della Distanza (TD).

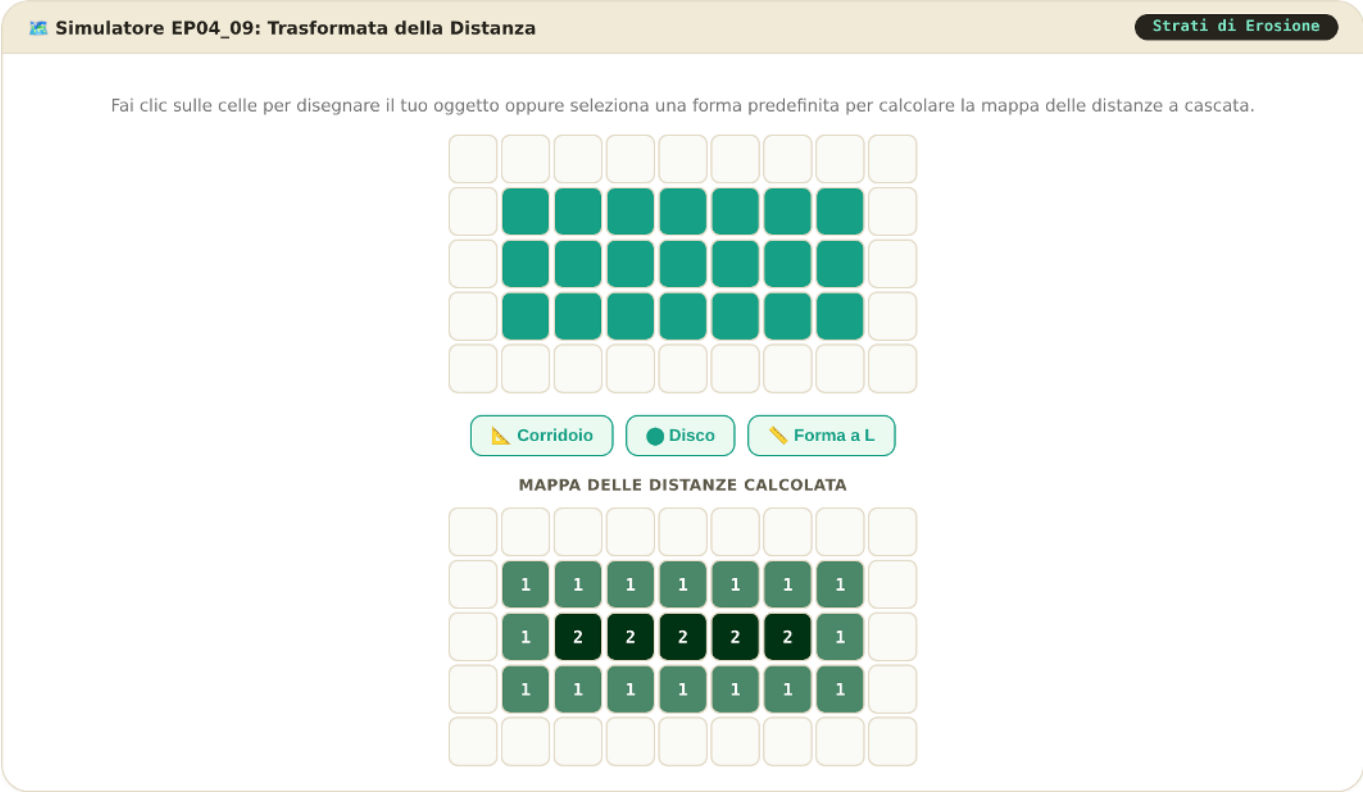
Output:

- Matrice della trasformata della distanza in L righe e C colonne.

4.9.9.5 📌 Esempio

Input	Output	Osservazione
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	Risultato della trasformata della distanza.

Nota: il valore `-99` agisce come un'approssimazione pratica di $-\infty$, impedendo la propagazione attraverso le diagonali. In questo modo, solo i vicini orizzontali e verticali contribuiscono alla distanza, producendo la distanza di Manhattan.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.it/cap04/sim-ep0409-distancia.html

Figura 4.38: Simulatore EP04_09: Trasformata della Distanza (Livelli di Erosione)

```
%%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 🪙 Separazione dei *Blob*, Etichettatura e Descrittori

In una **linea di produzione di monete**, è comune che i pezzi si tocchino l'un l'altro sul nastro trasportatore, formando un'unica macchia connessa nell'immagine — un conteggio ingenuo sbaglierebbe il totale. La soluzione classica combina operazioni morfologiche e analisi di connettività: prima un'**erosione** riduce o spezza le connessioni fragili tra gli oggetti, e poi l'**etichettatura delle componenti connesse** separa ciascun oggetto in una regione distinta. Infine, **descrittori geometrici** (area e bounding box) riassumono ciascuna componente trovata.

Vedi in Figura 4.39 una simulazione di questo EP.

4.9.10.1 📋 Linee Guida di Implementazione

1. **Dimensioni dell'immagine:** leggere gli interi L (righe) e C (colonne) da f.
2. **Dimensioni di B:** leggere gli interi L_B (righe) e C_B (colonne) dell'elemento strutturante.
3. **Elemento strutturante:** leggere la matrice B, contenente valori 0 o 1, riga per riga.
4. **Dati:** leggere la matrice binaria f (valori 0 o 1), riga per riga.
5. **Separazione:** calcolare
$$f_{ero} = f \ominus B$$
usando erosione binaria piatta (come nell'EP04_04), eliminando connessioni fragili tra gli oggetti.
6. **Etichettatura:** su f_{ero}, identificare le componenti connesse usando la connettività definita dall'intorno B. L'etichettatura deve seguire una scansione *raster*: quando si trova un pixel 1 non ancora etichettato, assegnare una nuova etichetta intera crescente a partire da 1 e propagare tale etichetta a tutta la regione connessa.
7. **Descrittori:** per ogni etichetta k, calcolare:
   - **Area:** numero di pixel appartenenti all'etichetta;
   - **Bounding box:**
   $$(y_{min}, x_{min}, y_{max}, x_{max})$$
8. **Output:** mostrare il numero totale di etichette e, successivamente, una riga per etichetta nel formato:
$$k,\ \text{area},\ y_{min},\ x_{min},\ y_{max},\ x_{max}$$

4.9.10.2 📌 Vincoli Computazionali

- L'erosione deve essere applicata prima dell'etichettatura.
- La connettività è fissa e definita dall'intorno sopra descritto.
- L'elemento strutturante B non interferisce con la connettività dell'etichettatura.
- Nessun padding in alcuna fase.
- L'ordine delle etichette segue la prima scoperta durante la scansione *raster*.

4.9.10.3 🧠 Fondamenti Teorici

Concetto	Significato	Impatto
Ponte sottile	Connessione stretta tra oggetti	Può essere rimosso dall'erosione morfologica
Connettività	Definita dall'insieme $\mathcal{N}(y, x)$	Determina quali pixel appartengono alla stessa componente
Area	Numero di pixel per componente	Stima diretta della dimensione dell'oggetto
Bounding box	Estensione spaziale dell'etichetta	Riassunto geometrico della componente

4.9.10.4 📦 Specifica di Input e Output (VPL)

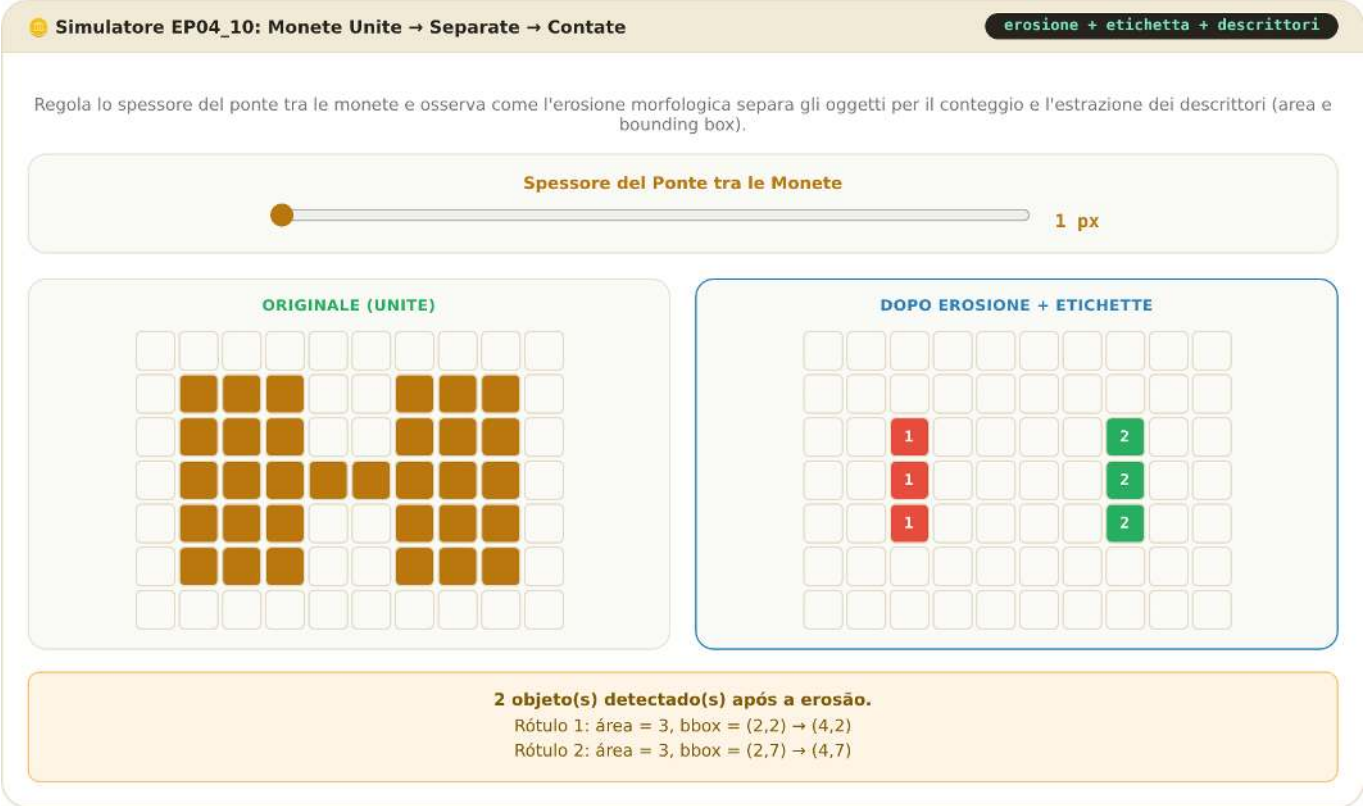
Input:

- Riga 1: intero L
- Riga 2: intero C
- Riga 3: intero L_B
- Riga 4: intero C_B
- Prossime L_B righe: matrice B
- Prossime L righe: matrice f

Output:

- Riga 1: numero totale di etichette trovate
- Righe successive:

$$k,\ \text{area},\ y_{min},\ x_{min},\ y_{max},\ x_{max}$$



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.it/cap04/sim-ep0410-rotulacao.html

Figura 4.39: Simulatore EP04_10: Separazione di Blob, Etichettatura e Descrittori

```
%%writefile EP04_10.cpp
// your solution
```

```
Overwriting EP04_10.cpp
```

```
TestSuite("EP04_10.cpp").run()
```

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

Capitolo 5

Trasformate e Compressione

Nei capitoli precedenti, tutte le operazioni sono state eseguite **nel dominio spaziale**, in cui gli algoritmi agiscono direttamente sui valori di intensità dei pixel.

In questo capitolo verrà presentato un approccio complementare: il **dominio della frequenza**, nel quale l'immagine è rappresentata dalle variazioni spaziali di intensità, e non solo dai valori individuali dei pixel.

Il concetto di **frequenza spaziale** descrive la rapidità con cui l'intensità varia lungo l'immagine. Le variazioni lente corrispondono a **basse frequenze**, mentre bordi, dettagli fini e rumori corrispondono ad **alte frequenze**.

Questa rappresentazione si basa sul fatto che qualsiasi immagine digitale discreta può essere decomposta in una combinazione di **funzioni ortogonali**. La **Trasformata di Fourier** utilizza una **base di esponenziali complesse bidimensionali** (equivalenti a sinusoidi con orientamento e frequenza specifici). Altre trasformate, come la **Trasformata del Coseno (DCT)** e la **Trasformata *Wavelet* (DWT)**, utilizzano diverse famiglie di funzioni di base — coseni bidimensionali nel caso della DCT, e funzioni con supporto compatto nel caso delle *wavelet*.

Tra le principali applicazioni di questa rappresentazione si evidenziano:

1. **Filtraggio nel dominio della frequenza**, per attenuare o enfatizzare determinate bande di frequenza;
2. **Analisi multirisoluzione tramite trasformate *wavelet***, che rappresenta strutture a diverse scale;
3. **Compressione delle immagini**, mediante la riduzione del numero di coefficienti necessari per rappresentare l'immagine.

5.1 Obiettivi

Al termine di questo capitolo, sarai in grado di:

- **Interpretare lo spettro di Fourier** di un'immagine, distinguendo magnitudine, fase e componenti di frequenza;
- **Applicare il Teorema della Convoluzione** per effettuare filtraggi nel dominio della frequenza utilizzando la Trasformata Veloce di Fourier (FFT);
- **Progettare e analizzare filtri nel dominio della frequenza**, comprendendo il funzionamento di filtri passa-basso, passa-alto e *notch*;
- **Comprendere l'analisi multirisoluzione tramite trasformate *wavelet*** e la loro applicazione nella rappresentazione gerarchica delle immagini;
- **Descrivere il processo di compressione delle immagini**, inclusa la Trasformata Discreta del Coseno (DCT) e la quantizzazione dei coefficienti;
- **Selezionare formati di archiviazione delle immagini**, come JPEG, PNG e WebP, in base ai requisiti dell'applicazione.

5.2 Configurazione dell'Ambiente

```
import os, urllib.request

os.makedirs("tmp/state", exist_ok=True)  # artefatti di build della traccia 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")

# Kernel Python anche nella traccia C++. cpp=True scarica morph.hpp + stb; le
# celle %%writefile *.cpp di questo capitolo compilano CON OpenCV
# (-DMM_USE_OPENCV + pkg-config opencv4).
import config
config.setup(cpp=True)
from morph import mm
import numpy as np
```

```
Ambiente pronto. Morph: 1.1.9 | OpenCV: 5.0.0
```

5.3 Trasformata di Fourier Discreta 2D

L'analisi di Fourier si basa sul principio secondo cui qualsiasi segnale periodico può essere rappresentato come una somma di funzioni sinusoidali con differenti frequenze, ampiezze e fasi. Questo concetto si applica anche alle immagini digitali, consentendo di rappresentarle nel **dominio della frequenza** invece che nel dominio spaziale.

La Figura 5.1 illustra questa decomposizione per un segnale unidimensionale. Nel caso di un'immagine, la Trasformata Discreta di Fourier (DFT) converte la matrice delle intensità $f(x, y)$ in un insieme di coefficienti che descrive il contributo delle diverse frequenze spaziali presenti nell'immagine.

```
%%writefile tmp/fig_decomposicao_1d.cpp
#define MM_OUT "tmp/fig_decomposicao_1d.png"
//| label: fig-decomposicao-1d
//| fig-cap: "Decomposição de Fourier 1D: uma onda quadrada (linha tracejada) é aproximada
pela soma das primeiras senoides (linhas coloridas). Quanto mais termos, melhor a
aproximação."
//| echo: false
//| output: true

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

int main() {
    // Cria um vetor com 400 pontos entre 0 e 2*pi
    std::vector<double> x(400);
    for (int i = 0; i < 400; i++) {
        x[i] = i * 2.0 * M_PI / 399.0;  // equivalente a np.linspace(0, 2*pi, 400)
    }

    // Onda quadrada: 1 para x < pi, -1 caso contrário
    std::vector<double> square(400);
    for (int i = 0; i < 400; i++) {
        square[i] = (x[i] < M_PI) ? 1.0 : -1.0;
```

```
    }

    // Soma das harmônicas
    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 = 0; n < 3; n++) {
        std::vector<double> h(400);
        for (int i = 0; i < 400; i++) {
            h[i] = (4.0 / M_PI) * (1.0 / (2 * n + 1)) * std::sin((2 * n + 1) * x[i]);
            soma[i] += h[i];
        }
        if (n == 0) h1 = h;
        if (n == 1) h2 = h;
        if (n == 2) h3 = h;
    }

    ys.push_back(square);
    ys.push_back(h1);
    ys.push_back(h2);
    ys.push_back(h3);
    ys.push_back(soma);

    std::vector<std::string> labels = {
        "Onda quadrada ideal", "1a harmonica", "3a harmonica", "5a harmonica", "Soma (3
        primeiras)"
    };
    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)
    };

    mm::Image chart = mm::lineChart(
        x, ys, labels, colors,
        "Sintese de Fourier: de senos a uma onda quadrada",
        "Posicao", "Intensidade"
    );

    std::vector<mm::Image> charts = {chart};
    mm::show(charts, MM_OUT, {"Decomposicao 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] Decomposicao 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)")
```

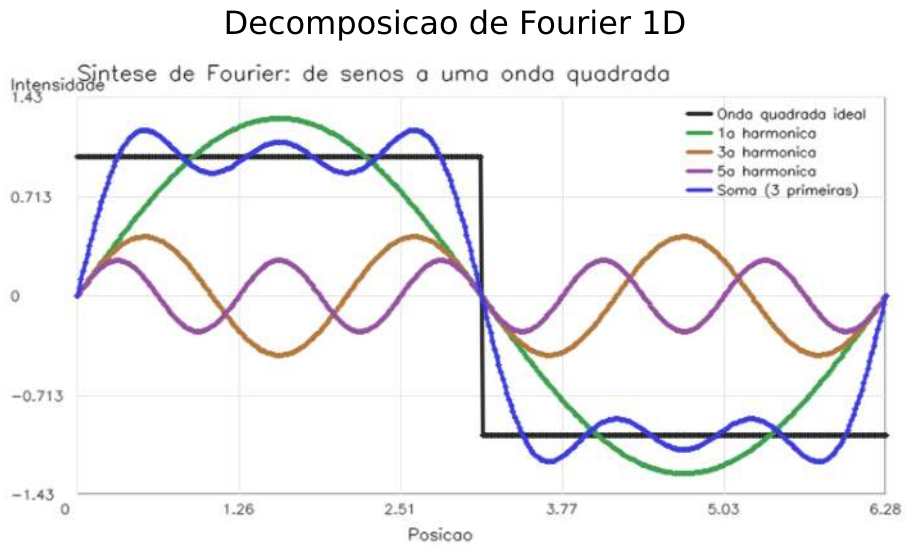


Figura 5.1: Decomposição de Fourier 1D: uma onda quadrada (linha tracejada) é aproximada pela soma das primeiras senoides (linhas coloridas). Quanto mais termos, melhor a aproximação.

5.3.1 Simulatore: Ricostruire Segnali con Sinusoidi

Prima di studiare le immagini bidimensionali, il simulatore della Figura 5.2 illustra il principio dell'analisi di Fourier per segnali unidimensionali: **una forma d'onda può essere approssimata dalla somma di sinusoidi con diverse frequenze e ampiezze**.

Man mano che si aggiungono nuovi termini, la somma delle sinusoidi (curva nera) si avvicina alla forma d'onda di riferimento (tratteggiata). Il grafico inferiore mostra lo spettro delle ampiezze, indicando il contributo di ciascuna frequenza alla ricostruzione del segnale.

Attività

Esplora il simulatore e rispondi:

1. Quanti termini sono necessari per ottenere una buona approssimazione dell'onda quadra?
2. Quale delle tre forme d'onda converge più rapidamente? Giustifica la tua risposta.
3. Come cambia lo spettro delle ampiezze passando dall'onda quadra a quella triangolare?

Risposte

1. Quanti termini sono necessari per una buona approssimazione dell'onda quadra?
Con circa 15-20 termini, la forma dell'onda si avvicina già bene al riferimento. Tuttavia, in prossimità delle discontinuità permane una piccola oscillazione, nota come **fenomeno di Gibbs**, che non scompare nemmeno aggiungendo più termini.
2. Quale forma converge più rapidamente? Perché?
L'**onda triangolare** converge più rapidamente, poiché le ampiezze delle sue armoniche decadono più velocemente di quelle dell'onda quadra e dell'onda a dente di sega. Di conseguenza, pochi termini già producono una buona approssimazione.
3. Come cambia lo spettro tra l'onda quadra e quella triangolare?
Entrambe presentano solo **armoniche dispari**, ma nell'onda triangolare le ampiezze diminuiscono molto più rapidamente. Pertanto, poche armoniche sono sufficienti per ricostruire il segnale con buona precisione.

5.3.2 Interpretazione dello spettro di frequenza

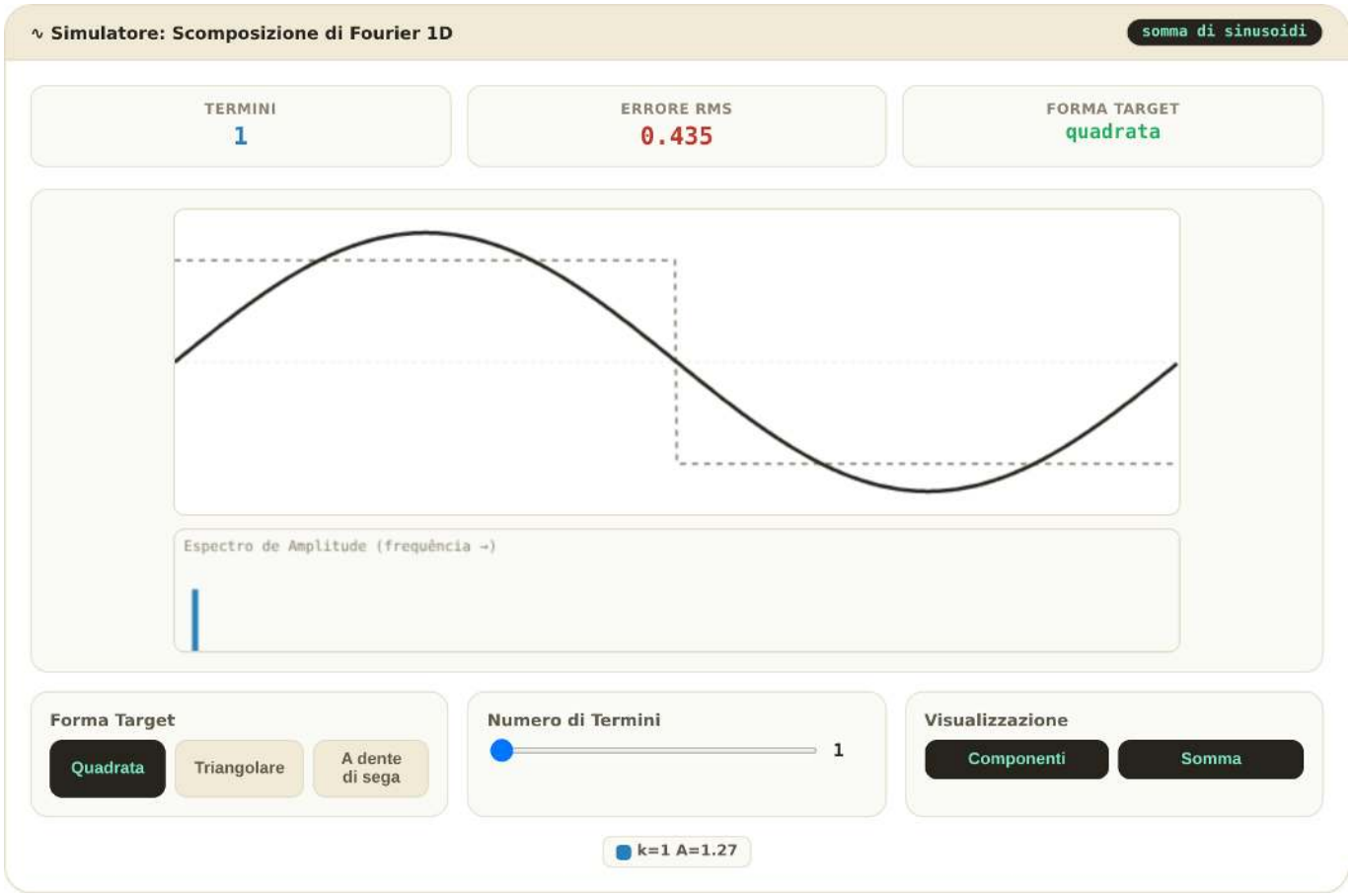
Applicando la Trasformata Discreta di Fourier (DFT) a un'immagine e visualizzando il modulo dei suoi coefficienti (vedi Figura 5.5), si ottiene lo **spettro di ampiezza**, che mostra la distribuzione delle frequenze spaziali presenti nell'immagine.

Il coefficiente situato all'origine della DFT, denominato **componente DC** (*Direct Current*), corrisponde alla frequenza nulla e rappresenta l'intensità media dell'immagine. Per convenzione, tale coefficiente è memorizzato nell'angolo superiore sinistro dello spettro. Per facilitarne l'interpretazione, si applica l'operazione **FFT Shift**, che sposta la componente DC al centro dell'immagine. Dopo questo spostamento, le basse frequenze si concentrano nella regione centrale, mentre le alte frequenze si trovano in prossimità dei bordi, come riassunto nella Tabella 5.1.

Tabella 5.1: Corrispondenza tra le regioni dello spettro di ampiezza dopo l'applicazione dell'FFT Shift.

Regione dello spettro	Componenti predominanti	Esempi nell'immagine
Centro (basse frequenze)	Variazioni spaziali lente	Illuminazione, regioni omogenee e forme globali
Regione intermedia (medie frequenze)	Variazioni di scala intermedia	Trame e pattern ripetitivi
Bordi (alte frequenze)	Variazioni spaziali rapide	Contorni, dettagli fini e rumore

Questa organizzazione facilita l'interpretazione dello spettro e la progettazione di filtri. L'attenuazione delle basse frequenze riduce le variazioni globali di intensità, mentre l'attenuazione delle alte frequenze smussa l'immagine riducendo i dettagli fini e parte del rumore.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.it/cap05/sim-05-freq.html

Figura 5.2: Simulatore interattivo della decomposizione di Fourier 1D: visualizzazione della somma di sinusoidi con diverse frequenze, ampiezze e fasi. Aggiungi termini e osserva la convergenza verso forme d'onda arbitrarie.

5.3.3 L'Esperimento della Griglia: Costruire un'Immagine da un Singolo Coefficiente

Prima di presentare la formulazione matematica della Trasformata Discreta di Fourier (DFT), è utile analizzare la sua inversa, denominata Trasformata Discreta Inversa di Fourier (IDFT). Si consideri uno spettro in cui tutti i coefficienti siano nulli, eccetto uno. Un esempio di questa costruzione è presentato nel codice della Figura 5.3 e può essere esplorato interattivamente nel simulatore della Figura 5.4..

L'immagine ricostruita è una sinusoide bidimensionale. La posizione del coefficiente nello spettro determina la sua **orientazione** e la sua **frequenza spaziale**, mentre la sua magnitudine e la sua fase definiscono, rispettivamente, la sua ampiezza e il suo spostamento spaziale. Pertanto, ciascun coefficiente della DFT rappresenta una componente sinusoidale, e l'immagine originale può essere ricostruita mediante la somma di tutte queste componenti.

```
%%writefile tmp/fig_05_grade_2d.cpp
#define MM_OUT "tmp/fig_05_grade_2d.png"
//| label: fig-05-grade-2d
//| fig-cap: "Toda frequência no espectro (ponto isolado) corresponde a uma onda senoidal 2D
rotacionada no domínio espacial."
//| echo: true
//| output: true

#include <opencv2/opencv.hpp>
#include <vector>
#include <string>
#include <complex>
#include "morph.hpp"
#include <filesystem>

int main() {
    int N_grid = 100;
    cv::Mat espectro_vazio(N_grid, N_grid, CV_64FC2, cv::Scalar(0, 0));

    // Acendendo um único ponto (frequência) fora do centro
    int u0 = 10, v0 = 5;
    espectro_vazio.at<cv::Vec2d>(N_grid/2 - v0, N_grid/2 - u0) = cv::Vec2d(1000, 0);

    // Retornando para o domínio espacial (IDFT)
    cv::Mat espectro_shifted;
    cv::Mat planes_shifted[2];
    cv::split(espectro_vazio, planes_shifted);

    // Manual ifftshift (swap quadrants)
    cv::Mat re_shifted = planes_shifted[0].clone();
    cv::Mat im_shifted = planes_shifted[1].clone();
    int cx = N_grid / 2;
    int cy = N_grid / 2;
    cv::Mat q0_re(re_shifted, cv::Rect(0, 0, cx, cy));
    cv::Mat q1_re(re_shifted, cv::Rect(cx, 0, cx, cy));
    cv::Mat q2_re(re_shifted, cv::Rect(0, cy, cx, cy));
    cv::Mat q3_re(re_shifted, cv::Rect(cx, cy, cx, cy));
    cv::Mat tmp_re;
    q0_re.copyTo(tmp_re);
    q3_re.copyTo(q0_re);
    tmp_re.copyTo(q3_re);
    q1_re.copyTo(tmp_re);
    q2_re.copyTo(q1_re);
    tmp_re.copyTo(q2_re);

    cv::Mat q0_im(im_shifted, cv::Rect(0, 0, cx, cy));
    cv::Mat q1_im(im_shifted, cv::Rect(cx, 0, cx, cy));
    cv::Mat q2_im(im_shifted, cv::Rect(0, cy, cx, cy));
```

```
    cv::Mat q3_im(im_shifted, cv::Rect(cx, cy, cx, cy));
    cv::Mat tmp_im;
    q0_im.copyTo(tmp_im);
    q3_im.copyTo(q0_im);
    tmp_im.copyTo(q3_im);
    q1_im.copyTo(tmp_im);
    q2_im.copyTo(q1_im);
    tmp_im.copyTo(q2_im);

    std::vector<cv::Mat> planes_shifted_vec = {re_shifted, im_shifted};
    cv::Mat espectro_shifted_complex;
    cv::merge(planes_shifted_vec, espectro_shifted_complex);

    cv::Mat onda_2d_complex;
    cv::idft(espectro_shifted_complex, onda_2d_complex, cv::DFT_SCALE |
cv::DFT_COMPLEX_OUTPUT);

    std::vector<cv::Mat> onda_planes;
    cv::split(onda_2d_complex, onda_planes);
    cv::Mat onda_2d = onda_planes[0];  // parte real

    cv::Mat onda_vis;
    cv::normalize(onda_2d, onda_vis, 0, 255, cv::NORM_MINMAX, CV_8U);

    cv::Mat espectro_mag;
    cv::magnitude(planes_shifted[0], planes_shifted[1], espectro_mag);
    cv::Mat espectro_vis;
    cv::normalize(espectro_mag, espectro_vis, 0, 255, cv::NORM_MINMAX, CV_8U);

    // Destaque visual do ponto
    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::Image img1(espectro_color);
    mm::Image img2(onda_vis);
    std::vector<mm::Image> imgs = {img1, img2};
    std::vector<std::string> titles = {"Espectro (1 ponto ativo)", "Onda 2D Resultante
(IDFT)"};
    mm::show(imgs, MM_OUT, titles, 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] Espectro (1 ponto ativo)
[2] Onda 2D Resultante (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)")
```

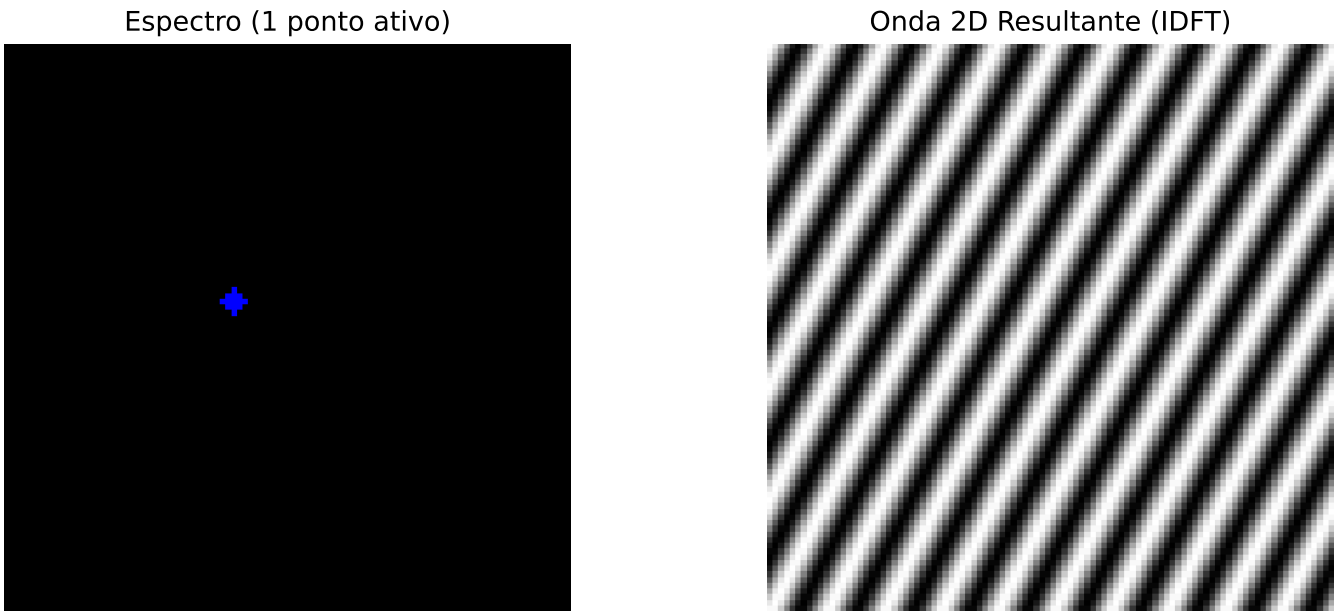
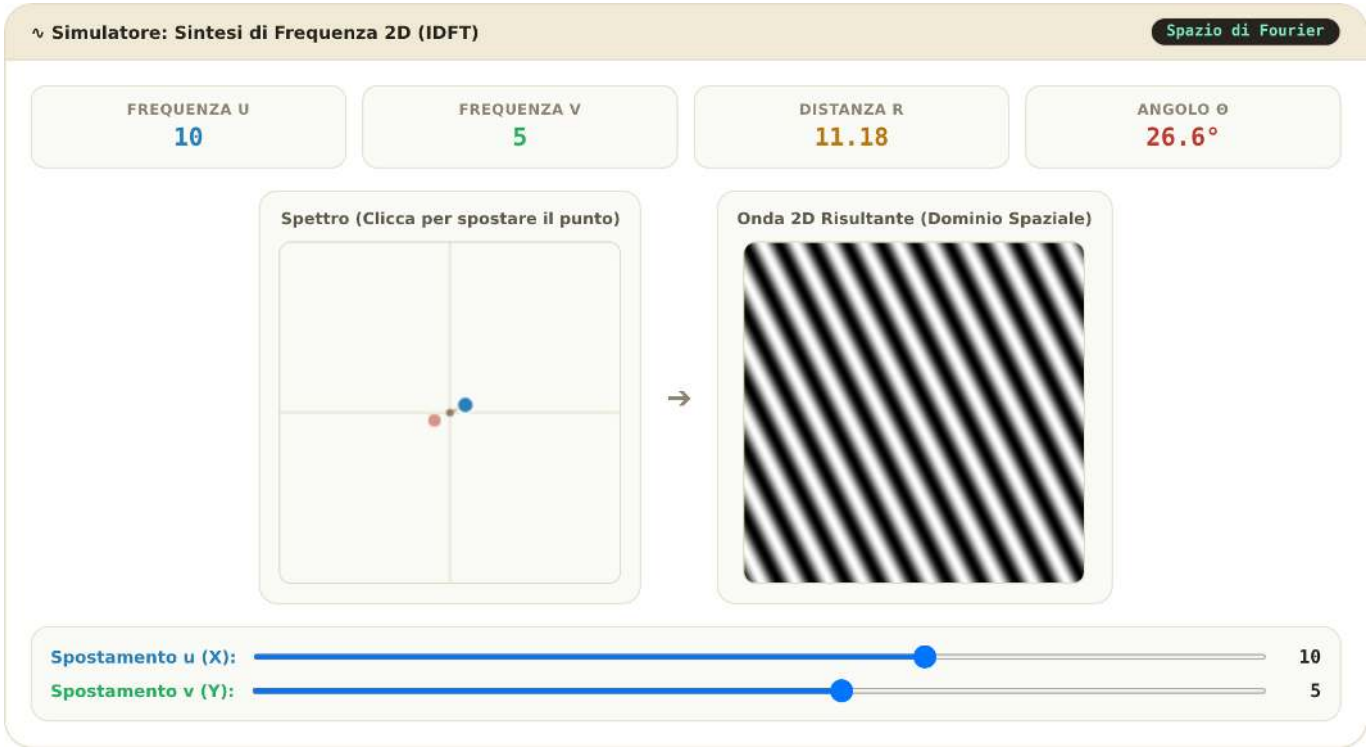


Figura 5.3: Toda frequência no espectro (ponto isolado) corresponde a uma onda senoidal 2D rotacionada no domínio espacial.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.it/cap05/sim-05-grade-2d.html

Figura 5.4: Simulatore interattivo della sintesi di Fourier 2D. Modifica la posizione orizzontale (u) e verticale (v) del coefficiente nello spettro di frequenze centrato e osserva come la distanza dal centro determina la frequenza spaziale (spessore) e l'angolo determina l'orientamento dell'onda sinusoidale generata.

5.3.4 Definizione Matematica

Si consideri un'immagine $f(x, y)$ con dimensioni $M \times N$. La sua **Trasformata Discreta di Fourier 2D** (DFT) è definita da:

$$F(u,v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y)\, e^{-j2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)} \tag{5.1}$$

dove $u = 0, 1, \ldots, M-1$ e $v = 0, 1, \ldots, N-1$ rappresentano le frequenze discrete nelle direzioni orizzontale e verticale, rispettivamente. Il termine esponenziale corrisponde a una sinusoide bidimensionale, la cui frequenza e orientazione sono determinate dagli indici (u, v).

La **Trasformata Discreta Inversa di Fourier 2D** (IDFT) ricostruisce l'immagine originale a partire dai suoi coefficienti:

$$f(x,y) = \frac{1}{MN} \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} F(u,v)\, e^{j2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)} \tag{5.2}$$

Le Equazioni Equazione 5.1 e Equazione 5.2 mostrano che la DFT e la IDFT formano una coppia di trasformazioni: la prima converte l'immagine nel dominio della frequenza, mentre la seconda ricostruisce esattamente l'immagine originale a partire dai suoi coefficienti.

> **i** Sul simbolo j
>
> Il termine j denota l'**unità immaginaria**, definita da $j^2 = -1$. In ingegneria e nell'elaborazione dei segnali, si adotta j invece di i per evitare conflitti con la notazione della corrente elettrica. Il suo utilizzo nell'esponenziale complessa, governato dalla formula di Eulero ($e^{j\theta} = \cos\theta + j\sin\theta$),

consente di rappresentare in modo compatto l'ampiezza e la fase di ciascuna frequenza spaziale presente nell'immagine.

Che cos'è il componente DC?

Il coefficiente $F(0,0)$, denominato **componente DC** (*Direct Current*), è uguale alla somma delle intensità di tutti i pixel dell'immagine (vedi Figura 5.5):

$$F(0,0) = MN\,\bar{f},$$

dove $\bar{f}$ è l'intensità media dell'immagine. Per questo motivo, il componente DC rappresenta il livello medio di intensità e, nella maggior parte delle immagini naturali, possiede la maggiore ampiezza dello spettro.
Gli altri coefficienti rappresentano variazioni attorno a tale media. Dopo l'applicazione del **FFT Shift**, il componente DC viene spostato al centro dello spettro, concentrando le basse frequenze nella regione centrale e le alte frequenze ai bordi.

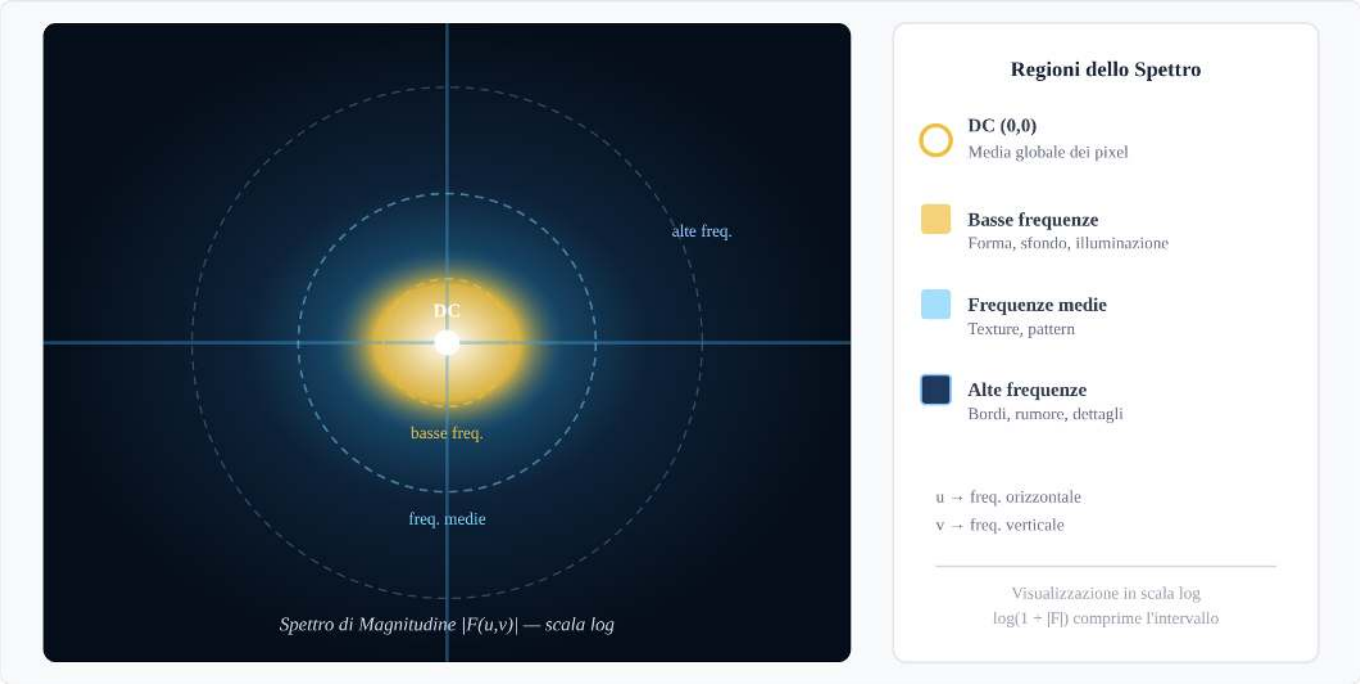


Figura 5.5: Diagramma concettuale dello spettro di Fourier 2D centrato.

5.3.5 Magnitudine e Fase

Ogni coefficiente della Trasformata Discreta di Fourier (DFT) è un numero complesso e può essere scritto come

$$F(u,v) = R(u,v) + j\,I(u,v),$$

dove $R(u,v)$ e $I(u,v)$ corrispondono, rispettivamente, alle parti **reale** e **immaginaria** del coefficiente. Dall'Equazione Equazione 5.1 si ottiene

$$R(u,v) = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)\cos\left(2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)\right),$$

e

$$I(u,v) = -\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)\sin\Big(2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)\Big)\,.$$

Da questa rappresentazione si definiscono due grandezze fondamentali:

- **Magnitudine**, che indica l'intensità della componente di frequenza,

$$|F(u,v)| = \sqrt{R(u,v)^2 + I(u,v)^2};$$

- **Fase**, che determina l'allineamento (o lo spostamento) spaziale della componente,

$$\phi(u,v) = \text{atan2}(I(u,v),\, R(u,v))\,.$$

Pertanto, ogni coefficiente può anche essere scritto nella sua forma polare,

$$F(u,v) = |F(u,v)|\, e^{j\phi(u,v)}.$$

Lo spettro di Fourier può quindi essere visualizzato mediante due immagini distinte: lo **spettro di magnitudine**, generalmente utilizzato per analizzare la distribuzione delle frequenze, e lo **spettro di fase**, che descrive l'organizzazione spaziale delle componenti sinusoidali.

Sebbene lo spettro di magnitudine sia il più utilizzato per l'ispezione visiva, la fase contiene gran parte delle informazioni strutturali dell'immagine. La combinazione di magnitudine e fase consente di ricostruire esattamente l'immagine originale tramite la IDFT.

5.3.6 Cosa trasportano l'ampiezza e la fase?

Una dimostrazione classica consiste nel combinare l'ampiezza di un'immagine con la fase di un'altra e ricostruire il risultato. Questo esperimento evidenzia che:

- **La fase** preserva la struttura spaziale dell'immagine, inclusa la posizione degli oggetti, i loro contorni e la loro geometria. Piccole alterazioni nella fase possono provocare grandi cambiamenti visivi.
- **L'ampiezza** controlla come l'energia è distribuita tra le frequenze spaziali, influenzando principalmente il contrasto e la tessitura.

Quando un'immagine viene ricostruita con l'ampiezza di A e la fase di B, il risultato tende a **somigliare più a B che ad A**, evidenziando che la fase è il componente principale responsabile dell'organizzazione spaziale della scena. Tuttavia, l'ampiezza rimane importante, poiché modula il contrasto delle strutture ricostruite. Pertanto, una ricostruzione fedele dipende dalla combinazione coerente tra ampiezza e fase.

Un esempio di questo comportamento è presentato in Figura 5.6.

i Analogia con l'Audio: Limitazioni e Precauzioni

La fase di un segnale svolge ruoli distinti nell'audio e nelle immagini:

- **Audio stereo o multicanale:** la fase relativa tra i canali è fondamentale per la percezione della posizione delle sorgenti sonore, tramite le differenze interaurali di tempo (ITD, *Interaural Time Differences*).
- **Audio monaurale:** la fase assoluta esercita una scarsa influenza percettiva diretta.
- **Immagini (DFT):** la fase è il fattore principale responsabile dell'organizzazione spaziale della scena, mentre l'ampiezza modula il contrasto e la distribuzione dell'energia tra le frequenze.

In entrambi i domini, l'**ampiezza** è legata all'intensità delle componenti di frequenza: nell'audio, influenza il timbro e l'intensità percepita; nelle immagini, influenza il contrasto e la tessitura.

```
import numpy as np, cv2, os  # [pdi] passthrough: garante imports desta trilha
#   Esperimento: L'Importanza della Fase
#   Caricamento dell'immagine
url     = "https://upload.wikimedia.org/wikipedia/commons/2/25/GAZI.MD.AHAD_11.jpg"
caminho = "imagens/coins.jpg"

if not os.path.exists(caminho):
    os.makedirs("imagens", exist_ok=True)
    img_obj = mm.read(url, pil=True)
    mm.write(img_obj, caminho)
else:
    img_obj = mm.read(caminho, pil=True)

img_color = np.array(img_obj)
img_gray  = mm.gray(img_color)

img_a = cv2.resize(img_gray, (400, 400))

# Creare un'immagine B sintetica (motivo geometrico)
img_b = np.zeros((400, 400), dtype=np.uint8)
cv2.rectangle(img_b, (100, 100), (300, 300), 255, -1)
cv2.circle(img_b, (200, 200), 150, 128, 10)

FA = np.fft.fft2(img_a)
FB = np.fft.fft2(img_b)

# Scambio di Fase
rec_A_mag_B_fase = np.real(np.fft.ifft2(np.abs(FA) * np.exp(1j * np.angle(FB))))
rec_B_mag_A_fase = np.real(np.fft.ifft2(np.abs(FB) * np.exp(1j * np.angle(FA))))

mm.show(
    [img_a, img_b, rec_A_mag_B_fase, rec_B_mag_A_fase],
    titles=["Immagine A", "Immagine B", "Mag(A) + Fase(B)", "Mag(B) + Fase(A)"],
    cols=4, figsize=(16, 4)
)

print("  La fase preserva bordi e contorni; la magnitudine controlla contrasto e")
print("texture. Nell'audio stereo, la fase influisce sulla localizzazione spaziale; nelle")
print("immagini, determina l'organizzazione della scena.")
# [pdi:state-io] auto-gerado - não editar à mão
import os as _pdi_os; _pdi_os.makedirs("tmp/state", exist_ok=True)
mm.write(img_gray, "tmp/state/img_gray_20.png")
# [pdi:state-io:end]
```

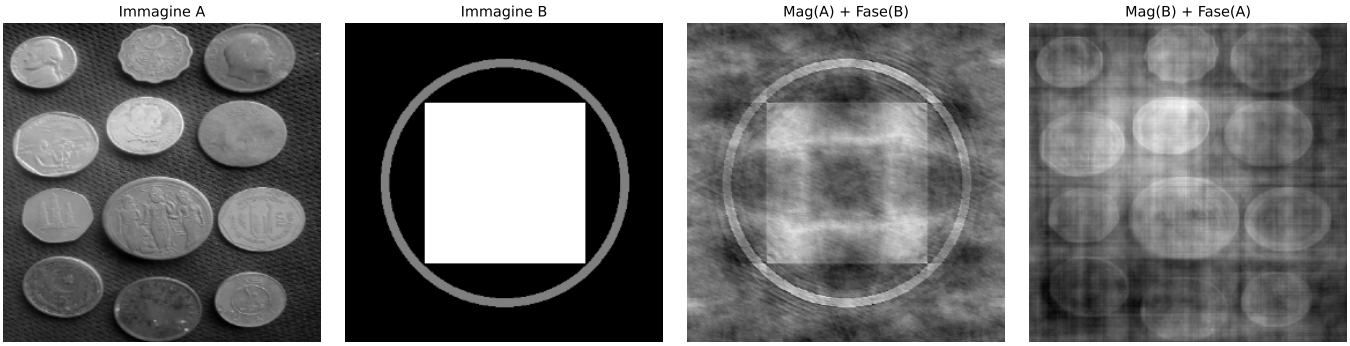


Figura 5.6: Esperimento di cambio di fase: Immagine A (monete) e Immagine B (motivo geometrico) ricostruite con magnitudini e fasi scambiate. Il risultato mostra che la struttura visiva è **molto più sensibile alla fase** che alla magnitudine: quando la fase di B viene mantenuta, l'immagine risultante preserva l'organizzazione spaziale di B, anche con la magnitudine di A. La magnitudine, a sua volta, influenza principalmente il contrasto e la texture. Si noti che la qualità della ricostruzione non è perfetta — ci sono artefatti visibili —, evidenziando l'interdipendenza tra fase e magnitudine per una rappresentazione fedele dell'immagine.

```
 La fase preserva bordi e contorni; la magnitudine controlla contrasto e
texture. Nell'audio stereo, la fase influisce sulla localizzazione spaziale; nelle
immagini, determina l'organizzazione della scena.
```

5.4 Teorema della Convoluzione e Strategie di Filtraggio

Il **Teorema della Convoluzione** stabilisce una relazione fondamentale tra il dominio spaziale e quello delle frequenze:

$$f(x,y) \circledast h(x,y) \overset{\mathcal{F}}{\longleftrightarrow} F(u,v)\,H(u,v) \tag{5.3}$$

dove $\circledast$ rappresenta la **convoluzione circolare discreta**. Pertanto, la convoluzione tra un'immagine $f(x,y)$ e un filtro $h(x,y)$ può essere sostituita dalla moltiplicazione dei loro spettri.

In pratica, per ottenere lo stesso risultato della convoluzione lineare eseguita nel dominio spaziale, si applica il **zero-padding** prima della Trasformata Rapida di Fourier (FFT), evitando artefatti ai bordi dell'immagine.

Tuttavia, la filtrazione nel dominio delle frequenze non è sempre l'alternativa più efficiente. Per filtri come quello gaussiano e il filtro della media (*Box Filter*), la proprietà di **separabilità** consente di ridurre significativamente il costo computazionale della convoluzione nel dominio spaziale.

5.4.1 *Kernel* Separabile vs. Non Separabile

Un ***kernel* separabile** può essere scritto come il prodotto esterno di due vettori unidimensionali,

$$H = v\,h^T,$$

consentendo di sostituire la convoluzione bidimensionale con due convoluzioni unidimensionali consecutive: una nella direzione orizzontale e una in quella verticale.

Un ***kernel* non separabile**, invece, non ammette tale decomposizione e, pertanto, la sua convoluzione deve essere eseguita direttamente sull'intorno bidimensionale.

In pratica, per un *kernel* di dimensione $K \times K$, la convoluzione diretta richiede K^2 moltiplicazioni per pixel, mentre un *kernel* separabile ne richiede solo $2K$, riducendo significativamente il costo computazionale.

5.4.2 Analisi dell'Efficienza Computazionale

Si consideri un'immagine di dimensioni $M \times N$ e un filtro quadrato di dimensione $K \times K$. La Tabella 5.2 confronta la complessità delle principali strategie di filtraggio.

Tabella 5.2: Confronto della complessità della convoluzione diretta, separabile e tramite Trasformata Rapida di Fourier (FFT).

Metodo di filtraggio	Complessità asintotica	Dipendenza da K	Applicazione tipica
Spatiale non separabile	$\mathcal{O}(MNK^2)$	Quadratica	*Kernel* piccoli e non separabili
Spatiale separabile	$\mathcal{O}(MNK)$	Lineare	Filtri Gaussiano e della media
Tramite FFT	$\mathcal{O}(MN\log(MN))$	Indipendente da K	*Kernel* grandi

Per *kernel* piccoli, la convoluzione spaziale, specialmente quando il filtro è separabile, risulta solitamente più efficiente grazie al basso costo delle operazioni. All'aumentare della dimensione del *kernel*, il filtraggio tramite FFT diventa più vantaggioso, poiché il suo costo è praticamente indipendente dalla dimensione del filtro.

5.4.3 Discussione dei risultati sperimentali

Il grafico ottenuto nella prova con l'immagine delle monete (2560×1920), presentato nella Figura 5.7, conferma il comportamento previsto dall'analisi della complessità computazionale.

1. **Convoluzione non separabile ($\mathcal{O}(MNK^2)$)**
   La convoluzione diretta presenta una crescita quadratica con la dimensione del *kernel*. Per valori piccoli di K, il costo è basso, ma aumenta rapidamente man mano che il *kernel* cresce, diventando impraticabile per applicazioni in tempo reale.
2. **Filtraggio tramite FFT ($\mathcal{O}(MN\log(MN))$)**
   Il costo della FFT dipende solo dalla dimensione dell'immagine, essendo indipendente da K. Pertanto, le sue prestazioni rimangono approssimativamente costanti al variare del *kernel*, rendendola vantaggiosa per filtri grandi o non separabili.
3. **Convoluzione separabile ($\mathcal{O}(MNK)$)**
   La scomposizione del *kernel* in due filtri monodimensionali riduce significativamente il costo computazionale. In pratica, questo approccio tende ad essere il più efficiente per filtri separabili, specialmente in implementazioni ottimizzate.

In generale, la scelta del metodo dipende dalla dimensione e dalla struttura del *kernel*. I filtri separabili sono più efficienti nel dominio spaziale, mentre la FFT diventa più vantaggiosa per *kernel* grandi o per convoluzioni multiple nel dominio della frequenza.

$$g = \mathcal{F}^{-1}[\mathcal{F}(f)\cdot\mathcal{F}(h)] \quad \text{(FFT)} \qquad g = f \circledast h \quad \text{(convoluzione diretta)} \qquad g = (f \circledast v) \circledast h^T \quad \text{(separabile)} \tag{5.4}$$

dove:

- $f(x,y)$ rappresenta l'immagine di ingresso;
- $h(x,y)$ è il *kernel* bidimensionale del filtro;
- v e h^T sono, rispettivamente, i vettori verticale e orizzontale che compongono il *kernel* separabile.

```
%%writefile tmp/fig_05_conv_eficiencia.cpp
#define MM_OUT "tmp/fig_05_conv_eficiencia.png"
// Compile: g++ -std=c++17 -o program program.cpp $(pkg-config --cflags --libs opencv4)
#include <vector>
```

```cpp
#include <string>
#include <cmath>
#include "morph.hpp"
#include <filesystem>

#ifndef MM_OUT
#endif

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;

    std::vector<double> t_nao_sep;
    std::vector<double> t_sep;
    std::vector<double> t_fft;

    for (double k : K) {
        t_nao_sep.push_back(N2 * k * k / 1e6);
        t_sep.push_back(N2 * 2.0 * k / 1e6);
        t_fft.push_back(N2 * std::log2(N2) / 1e6);
    }

    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,
        {}, // cores vazias - usa padrão
        "Custo relativo: Espacial vs Frequencia",
        "Tamanho do kernel  K",
        "Operacoes  (x10^6)"
    );

    mm::show(std::vector<mm::Image>{chart}, MM_OUT,
             std::vector<std::string>{"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)")
```

Figura 5.7: Comparação de eficiência: Convolução Não Separável (Espacial 2D), Separável (Espacial 1D) e via FFT.

! Il problema della convoluzione circolare (*wrap-around*)

La Trasformata Discreta di Fourier (DFT) assume che l'immagine sia **estesa periodicamente nello spazio**, cioè che i suoi bordi si ripetano indefinitamente.
In questa condizione, la moltiplicazione nel dominio della frequenza corrisponde a una **convoluzione circolare** nel dominio spaziale. Di conseguenza, regioni opposte dell'immagine (alto e basso, sinistra e destra) iniziano a interagire artificialmente, come illustrato nella Figura 5.8..
L'applicazione del *zero-padding* prima della FFT riduce questo effetto estendendo l'immagine con valori nulli ai bordi, avvicinando il risultato alla convoluzione lineare. Questo comportamento può essere interpretato alla luce del Teorema della Convoluzione, presentato nella Figura 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 <cmath>
#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]

    //   Definir M e N
    int M = img_gray.h;  // linha adicionada
    int N = img_gray.w;

    // 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 phase = -2.0 * M_PI * (u * 120.0 / M + v * 120.0 / N);
            H_shift.at<cv::Vec2d>(u, v) = {std::cos(phase), std::sin(phase)};
        }
    }

    // Filtragem SEM padding (causa o wrap-around)
    // F_img = np.fft.fft2(img_gray) → FFT da imagem em ponto flutuante
    cv::Mat img_gray_f;
    cv::Mat img_gray_cv = img_gray;  // conversão implícita
    img_gray_cv.convertTo(img_gray_f, CV_64F);
    cv::Mat F_img;
    cv::dft(img_gray_f, F_img, cv::DFT_COMPLEX_OUTPUT);

    // Multiplicação no domínio da frequência
    cv::Mat F_product;
    cv::mulSpectrums(F_img, H_shift, F_product, 0);

    // Inversa com escala e saída real
    cv::Mat img_vazada;
```

```
    cv::idft(F_product, img_vazada, cv::DFT_SCALE | cv::DFT_REAL_OUTPUT);

    // Normalização para visualização
    cv::Mat img_vazada_vis;
    cv::normalize(img_vazada, img_vazada_vis, 0, 255, cv::NORM_MINMAX, CV_8U);

    // Exibição
    mm::Image img_vazada_vis_img = img_vazada_vis;
    mm::show({img_gray, img_vazada_vis_img},
             MM_OUT,
             {"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)")
```

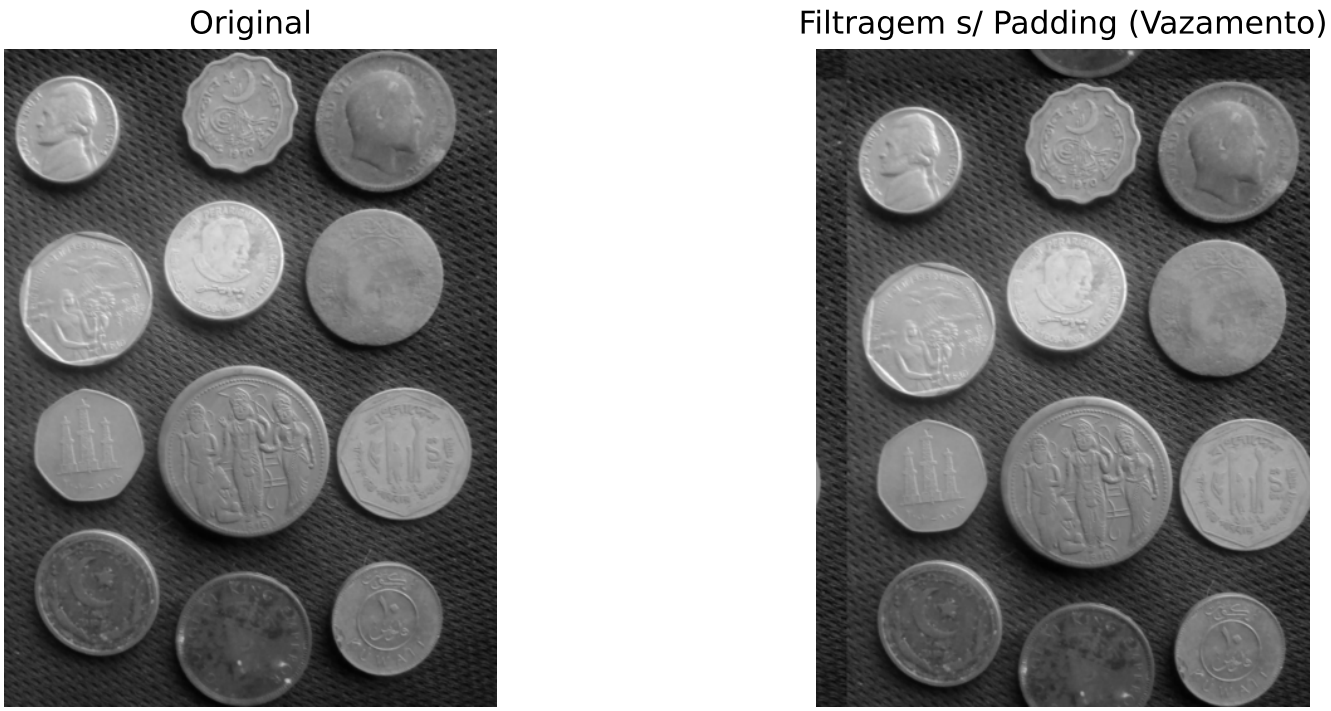


Figura 5.8: Sem *padding*, um deslocamento severo faz a imagem vazar para o lado oposto (convolução circular).

```
%%writefile tmp/fig_05_conv_teorema.cpp
#define MM_OUT "tmp/fig_05_conv_teorema.png"
//| label: fig-05-conv-teorema
//| fig-cap: "Teorema della Convoluzione: filtrare nel dominio spaziale (Gaussiana) equivale a
moltiplicare lo spettro per $H(u,v)$ in frequenza. Le uscite coincidono visivamente."
//| 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]

    // Stessa sfocatura attraverso due percorsi: spazio vs. frequenza.
    int M = img_gray.h;
    int N = img_gray.w;

    cv::Mat H = mm::gaussFilter(M, N, 30);      // H(u,v): trasferimento passa-basso gaussiano
    mm::Image f_freq = mm::freqFilter(img_gray, H);    // convoluzione tramite moltiplicazione
in frequenza

    // Applica la stessa sfocatura spaziale (Gaussiana 31x31 con sigma=5)
    cv::Mat img_mat(img_gray.h, img_gray.w, CV_8UC1, img_gray.data.data());
    cv::Mat blurred;
    cv::GaussianBlur(img_mat, blurred, cv::Size(31, 31), 5);
    mm::Image f_esp(blurred);
```

```
    mm::show(
        std::vector<mm::Image>{img_gray, f_esp, f_freq},
        MM_OUT,
        {"Originale", "Spazio: Gaussiana", "Frequenza: 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] Originale
[2] Spazio: Gaussiana
[3] Frequenza: 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)")
```

Figura 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 Sulla differenza numerica

La differenza residua dell'ordine di 10^{-13} non viola il Teorema della Convoluzione, ma riflette **limitazioni computazionali** inerenti all'aritmetica a virgola mobile (doppia precisione, $\sim 10^{-16}$) e all'**ordine delle operazioni** tra i due metodi:

- **Convoluzione spaziale:** somma ponderata di vicini con arrotondamenti successivi.
- **Convoluzione in frequenza:** coinvolge tre trasformate FFT e una moltiplicazione complessa, soggetta a errori di troncamento e quantizzazione.

Pertanto, l'uguaglianza teorica è esatta, ma l'implementazione numerica produce una differenza praticamente nulla (errore relativo $< 10^{-12}$), confermando il teorema entro la precisione della macchina.

5.5 Filtri nel Dominio delle Frequenze

Un filtro nel dominio delle frequenze può essere interpretato come una **funzione di trasferimento applicata allo spettro dell'immagine**. In questa rappresentazione, ogni coefficiente di frequenza viene moltiplicato per un valore compreso tra 0 e 1, che determina la sua attenuazione o preservazione. La forma di questa funzione definisce l'effetto visivo del filtro.

Taglio netto e *ringing*. I filtri ideali con transizione istantanea a una frequenza di taglio D_0 producono discontinuità nel dominio delle frequenze. Questa discontinuità si riflette nel dominio spaziale come oscillazioni in prossimità dei bordi, note come *ringing*. Questo effetto è associato alla convoluzione con funzioni di supporto infinito nello spazio, come la funzione *sinc*, come illustrato nella Figura 5.10.

Filtri con transizione graduale. Alternative come i filtri Gaussiano e Butterworth attenuano la transizione tra regioni di passaggio e di reiezione, riducendo il *ringing*. Di contro, tale attenuazione implica un confine di separazione meno definito tra frequenze preservate e attenuate.

```
%%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
```

```
// Filtro Ideal na frequência (cilindro) e sua resposta espacial (sinc 2D).
#include <opencv2/opencv.hpp>
#include "morph.hpp"
#include <vector>
#include <string>
#include <filesystem>

int main() {
    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::show(
        {H_freq, h_space},
        MM_OUT,
        {
            "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)")
```

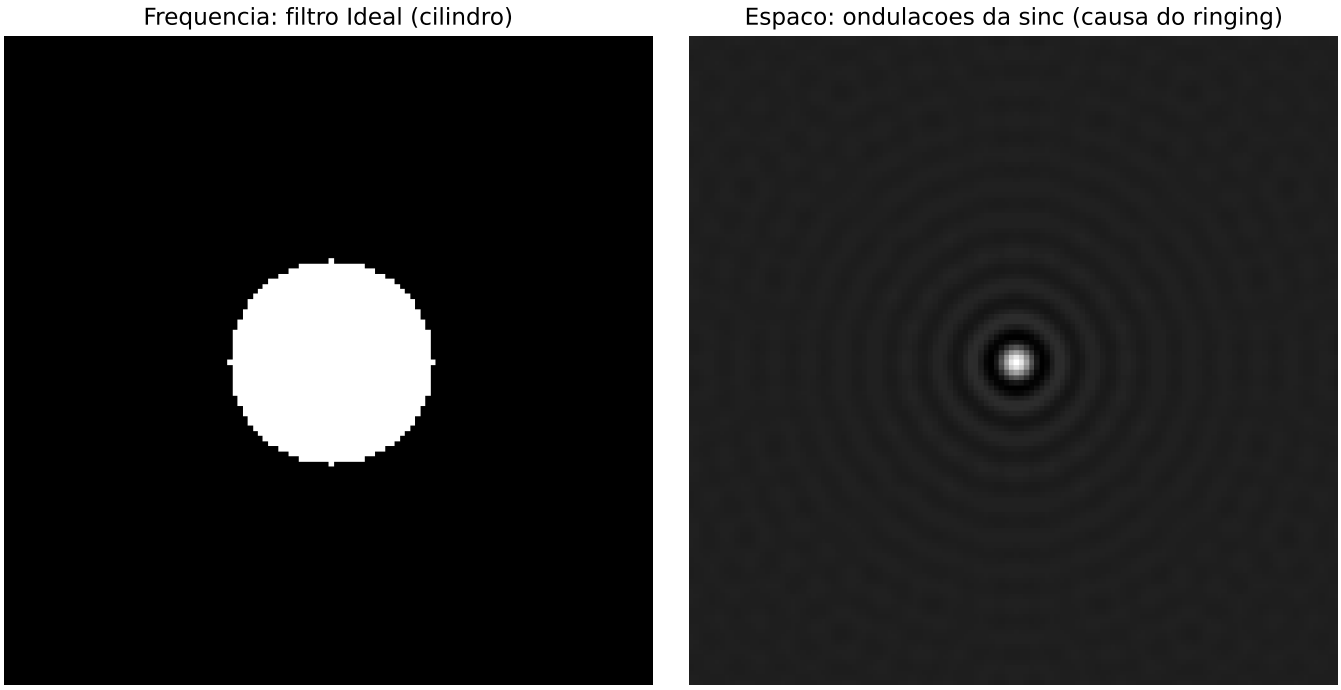


Figura 5.10: A Dualidade Perigosa: o corte abrupto na Frequência (cilindro Ideal) vira obrigatoriamente uma *sinc* no espaço. Suas ondulações causam o *ringing* fantasma nas bordas da imagem.

5.5.1 Filtri Passa-Basso

I filtri passa-basso attenuano le componenti ad alta frequenza, producendo un appiattimento dell'immagine e una riduzione del rumore. Dopo la centralizzazione dello spettro (FFT Shift), la distanza di ciascun punto dal centro è data da:

$$D(u,v) = \sqrt{\left(u - \tfrac{M}{2}\right)^2 + \left(v - \tfrac{N}{2}\right)^2} \tag{5.5}$$

Filtro Ideale (LPFI):

$$H_{\text{ideal}}(u,v) = \begin{cases} 1, & D(u,v) \le D_0 \\ 0, & D(u,v) > D_0 \end{cases} \tag{5.6}$$

Il taglio netto a D_0 introduce discontinuità nel dominio delle frequenze, causando oscillazioni nel dominio spaziale note come *ringing*. Questo effetto è associato alla convoluzione con funzioni a supporto infinito.

Filtro Gaussiano (LPFG):

$$H_{\text{gauss}}(u,v) = e^{-D^2(u,v)/(2\sigma^2)} \tag{5.7}$$

La regolarità della funzione gaussiana nel dominio delle frequenze evita discontinuità, eliminando il *ringing* e producendo una transizione graduale tra frequenze preservate e attenuate.

Filtro di Butterworth (LPFB) di ordine n:

$$H_{\mathrm{BW}}(u,v) = \frac{1}{1 + [D(u,v)/D_0]^{2n}} \tag{5.8}$$

Il parametro n controla la gradualità della transizione tra l'attenuazione e il passaggio delle frequenze. Valori piccoli producono transizioni morbide, mentre valori grandi avvicinano il comportamento al filtro ideale, con un maggiore rischio di *ringing*. Un esempio comparativo è mostrato in Figura 5.11.

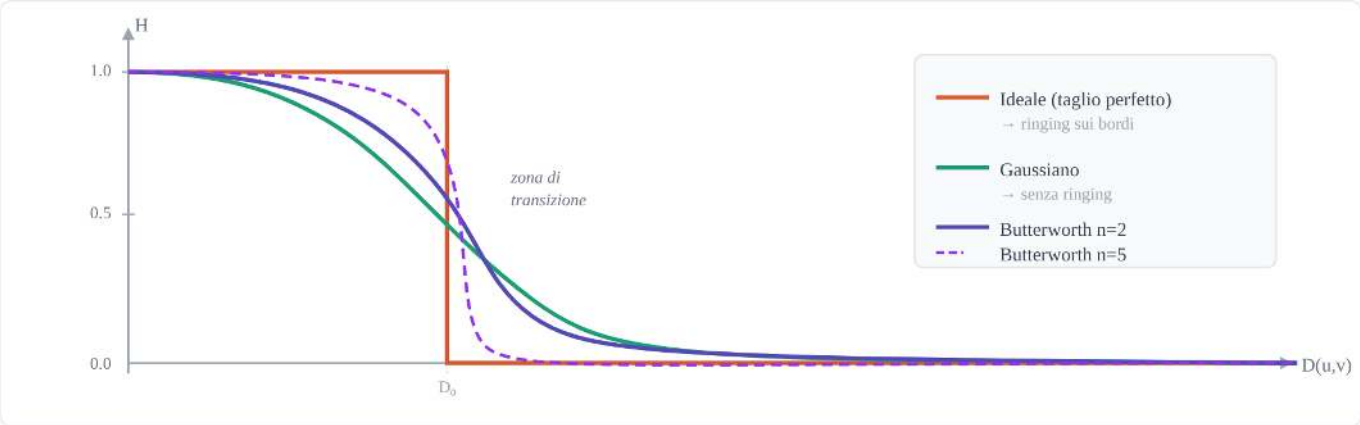


Figura 5.11: Filtri passa-basso.

5.5.2 Filtri Passa-Alto e Passa-Banda

I filtri passa-alto possono essere ottenuti da un filtro passa-basso complementare, definito come:

$$H_{\mathrm{HP}}(u,v) = 1 - H_{\mathrm{LP}}(u,v)$$

Questo tipo di filtro preserva le componenti ad alta frequenza, evidenziando bordi e dettagli, mentre attenua le regioni a variazione graduale.

I filtri passa-banda preservano solo una banda intermedia di frequenze, limitata da due raggi D_L e D_H:

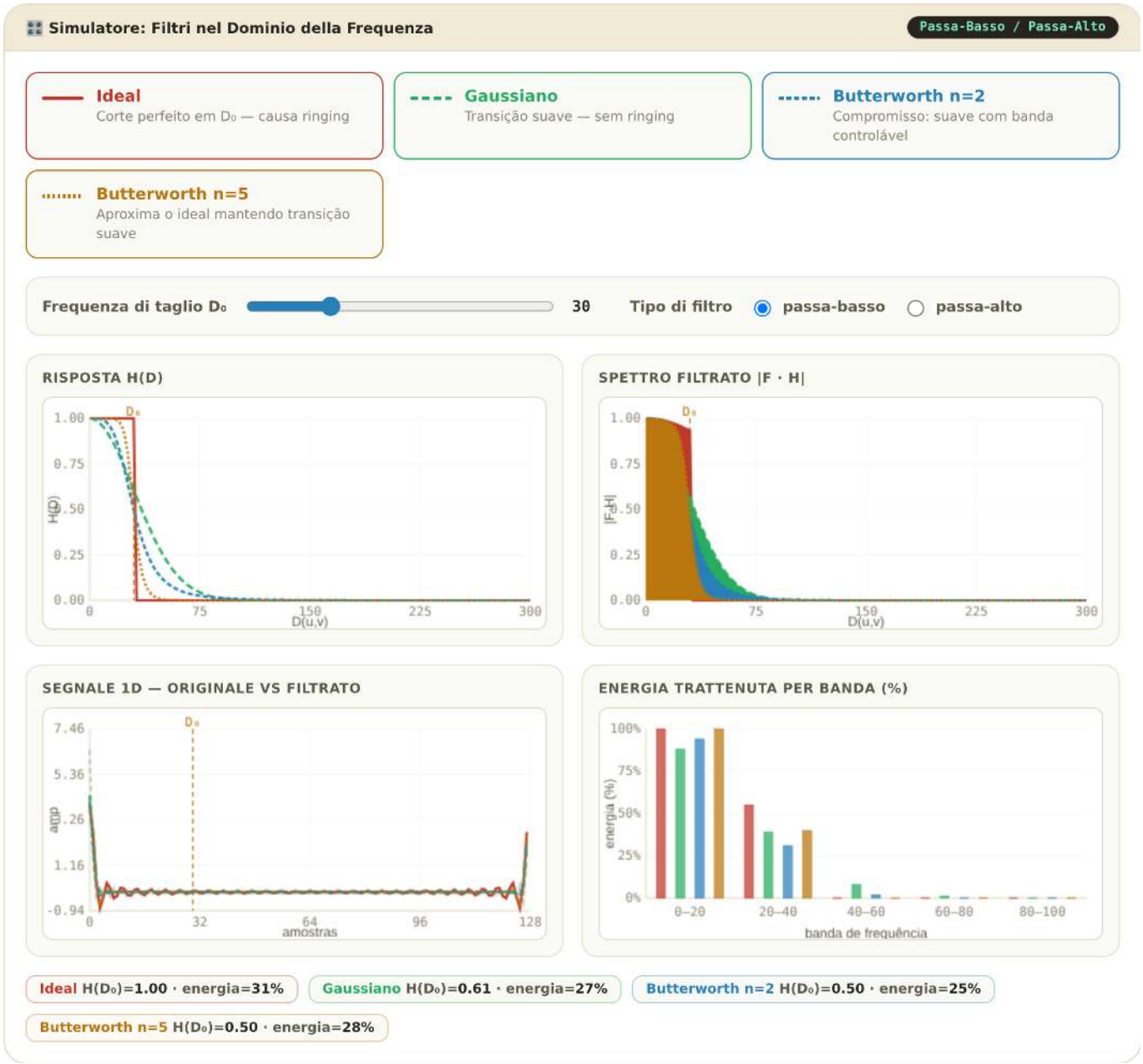
$$H_{\mathrm{BP}}(u,v) = H_{\mathrm{LP}}^{(D_H)}(u,v) \cdot \left[1 - H_{\mathrm{LP}}^{(D_L)}(u,v)\right]$$

Questo tipo di filtraggio è utile quando si desidera rimuovere simultaneamente le componenti a bassa e ad alta frequenza, preservando solo le strutture di scala intermedia.

Un'applicazione importante è la rimozione del **rumore periodico**, in cui i pattern regolari appaiono come picchi localizzati nello spettro di magnitudine. Questi picchi possono essere attenuati mediante filtri *notch* (reietta-banda), posizionati specificamente sulle frequenze indesiderate.

Esempi di filtri nel dominio della frequenza sono presentati nel simulatore della Figura 5.12, Figura 5.13 e Figura 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.it/cap05/sim-05-filtros.html

Figura 5.12: Simulatore interattivo di filtri nel dominio della frequenza.

```cpp
#include <opencv2/opencv.hpp>
#include <vector>
#include <string>
#include <cmath>
#include "morph.hpp"

// Função auxiliar para converter H (CV_64F) para visualização
mm::Image H_vis(const cv::Mat& H) {
    cv::Mat H_uint8;
    cv::normalize(H * 255.0, H_uint8, 0, 255, cv::NORM_MINMAX, CV_8U);
    return mm::Image(H_uint8);
}

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;
    double D0 = 30;
    int n_bw = 2;

    //   Funções de transferência (H centrado) via construtores de morph.hpp
    //   Ideal:       1 se D<=D0, senão 0
    //   Gaussiano:   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);

    //   Imagens filtradas 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);

    //   Perfis de H(u,v) na linha central, desenhados com cv::line
    int linha = M / 2;
    int Wp = 512, Hp = 288;
    cv::Mat perfil(Hp, Wp, CV_8UC3, cv::Scalar(255, 255, 255));

    // Função lambda para desenhar um perfil
    auto traca = [&](const cv::Mat& row, cv::Scalar cor) {
        cv::Point ant(-1, -1);
        bool first = true;
        for (int x = 0; x < N; ++x) {
            double val = row.at<double>(x);
            int px = (int)std::round(x * (double)(Wp - 1) / (N - 1));
            int py = (int)std::round(10 + (1.0 - val) * (Hp - 20));
            if (ant.x >= 0) {
                cv::line(perfil, ant, cv::Point(px, py), cor, 2, cv::LINE_AA);
            }
            ant = cv::Point(px, py);
        }
    };

    traca(H_ideal.row(linha), cv::Scalar(216, 90, 48));
    traca(H_gauss.row(linha), cv::Scalar(29, 158, 117));
    traca(H_bw.row(linha),    cv::Scalar(83, 74, 183));

    // Preparar imagens para exibição
```

```
    std::vector<mm::Image> imagens = {
        img_gray, img_ideal, img_gauss, img_bw,
        H_vis(H_ideal), H_vis(H_gauss), H_vis(H_bw),
        mm::Image(perfil)
    };

    std::vector<std::string> titulos = {
        "Original", "LPF Ideal", "LPF Gaussiano", "LPF Butterworth (n=2)",
        "H Ideal", "H Gaussiano", "H Butterworth", "Perfis H(u,v)"
    };

    mm::show(imagens, MM_OUT, titulos, 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 Gaussiano
[4] LPF Butterworth (n=2)
[5] H Ideal
[6] H Gaussiano
[7] H Butterworth
[8] Perfis 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)")
```

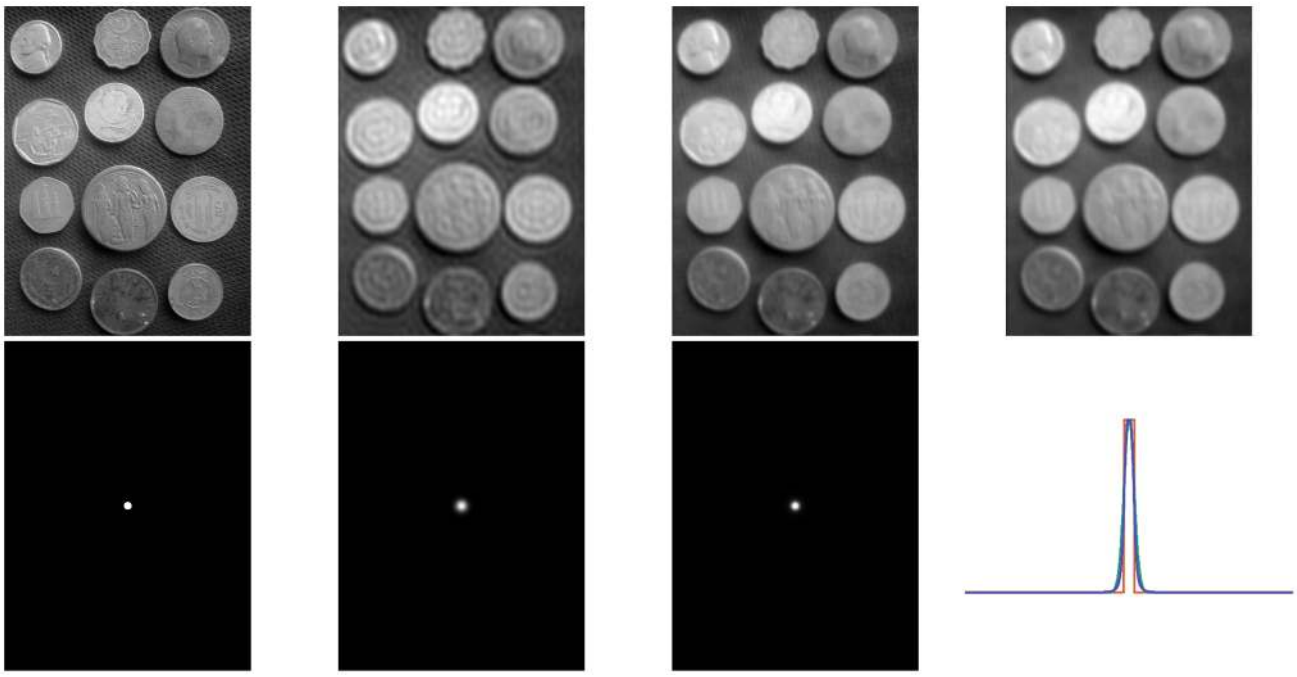


Figura 5.13: Comparação entre filtros passa-baixa: Ideal (D =30), Gaussiano (D =30) e Butterworth (D =30, n=2). Perfis de H(u,v) ao longo de uma linha central e imagens filtradas correspondentes.

```
%%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,
        std::vector<std::string>{"Original", "Passa-alta Gaussiano ($D_0=30$)"},
        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 ($D_0=30$)
```

```
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)")
```

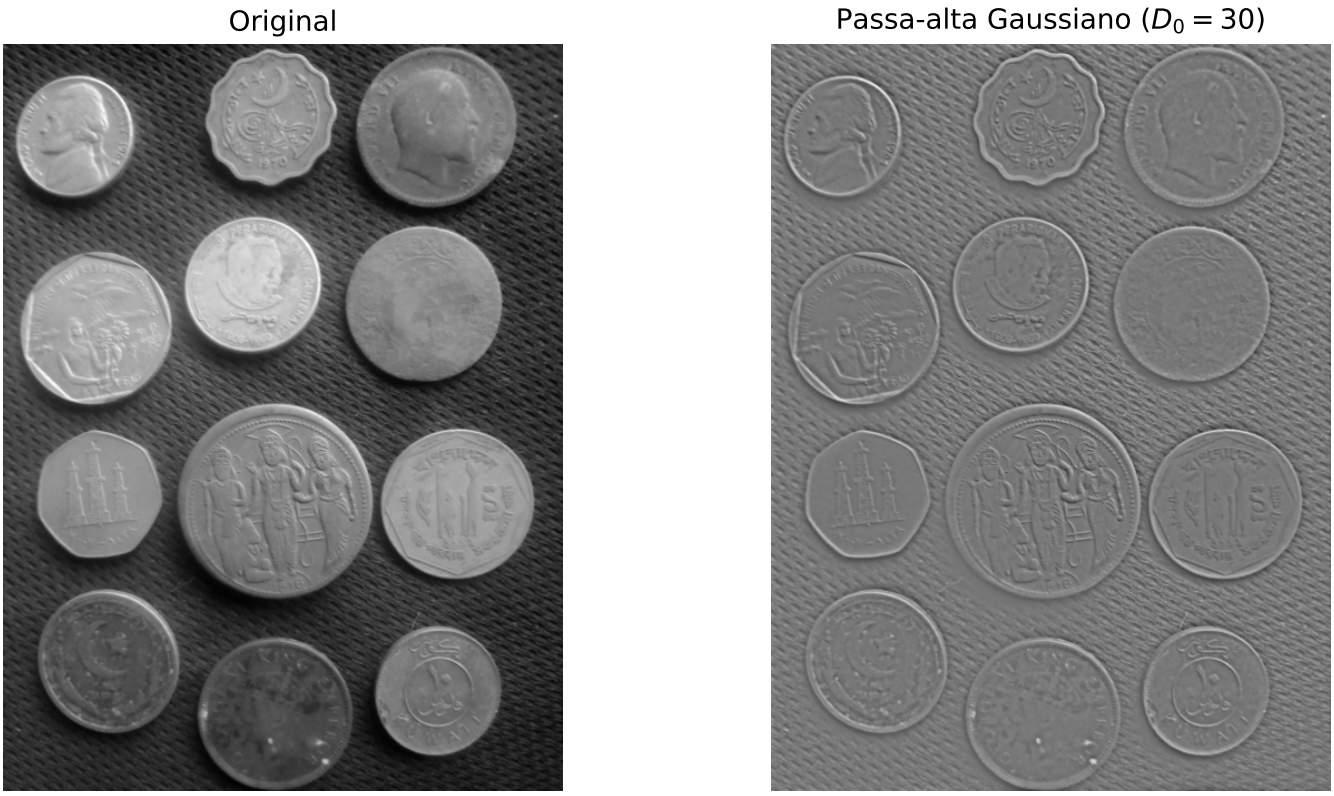


Figura 5.14: Filtro passa-alta Gaussiano. (a) Original; (b) Filtro passa-alta (D =30) - as bordas das moedas e fundo texturizado são realçados.

5.5.3 Rimozione del Rumore Periodico

Il rumore periodico — associato a interferenze elettriche, pattern regolari dei sensori o artefatti di scansione — appare nello spettro di Fourier come **picchi puntuali simmetrici rispetto al centro**.

Il filtro **reietta-banda (*notch*)** attenua selettivamente queste frequenze, preservando le altre componenti dell'immagine. Un esempio di applicazione è presentato nella Figura 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 <iostream>
#include <cmath>
#include <vector>
#include <string>
#include <cstdio>
#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 já está disponível (injetado automaticamente)

    int h_img = img_gray.h;
    int w_img = img_gray.w;

    // Criar meshgrid X, Y equivalente
    cv::Mat X(h_img, w_img, CV_64F), Y(h_img, w_img, CV_64F);
    for (int y = 0; y < h_img; y++) {
        for (int x = 0; x < w_img; x++) {
            X.at<double>(y, x) = x;
            Y.at<double>(y, x) = y;
        }
    }

    double u0 = 20, v0 = 20;  // frequências exatas do ruído

    // Converter img_gray para CV_64F
    cv::Mat img_gray64f;
    cv::Mat img_gray_mat = img_gray;  // conversão implícita para cv::Mat
    img_gray_mat.convertTo(img_gray64f, CV_64F);

    // Calcular ruído senoidal
    cv::Mat ruido(h_img, w_img, CV_64F);
    for (int y = 0; y < h_img; y++) {
        for (int x = 0; x < w_img; x++) {
            ruido.at<double>(y, x) = 40 * sin(2 * M_PI * (u0 * X.at<double>(y, x) / w_img + v0
* Y.at<double>(y, x) / h_img));
        }
    }

    // Imagem ruidosa = clip(img_gray + ruido, 0, 255)
    cv::Mat img_ruidosa64f = img_gray64f + ruido;
    cv::Mat img_ruidosa;
    cv::threshold(img_ruidosa64f, img_ruidosa, 0, 255, cv::THRESH_TRUNC);
    cv::threshold(img_ruidosa, img_ruidosa, 0, 0, cv::THRESH_TOZERO);
    img_ruidosa.convertTo(img_ruidosa, CV_8U);

    mm::Image img_ruidosa_mm = img_ruidosa;  // converter para mm::Image se necessário

    // Espectro de magnitude (log) da imagem 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);
    double r_notch = 8;
    int cy = h_img / 2, cx = w_img / 2;

    // Criar meshgrid yy, xx com indexing="ij"
    cv::Mat yy(h_img, w_img, CV_64F), xx(h_img, w_img, CV_64F);
    for (int y = 0; y < h_img; y++) {
        for (int x = 0; x < w_img; x++) {
            yy.at<double>(y, x) = y;
            xx.at<double>(y, x) = x;
        }
    }

    // Para cada par (dy, dx) nos 4 picos simétricos
    std::vector<std::pair<double, double>> picos = {{v0, u0}, {-v0, -u0}, {v0, -u0}, {-v0,
u0}};
    for (auto& p : picos) {
```

```
        double dy = p.first, dx = p.second;
        for (int y = 0; y < h_img; y++) {
            for (int x = 0; x < w_img; x++) {
                double dist = sqrt(pow(yy.at<double>(y, x) - (cy + dy), 2) +
pow(xx.at<double>(y, x) - (cx + dx), 2));
                if (dist <= r_notch) {
                    mascara.at<double>(y, x) = 0.0;
                }
            }
        }
    }

    // Visualização da máscara
    cv::Mat mascara_vis = mascara * 255;
    mascara_vis.convertTo(mascara_vis, CV_8U);

    // Filtragem: aplica a máscara centrada como filtro no domínio da frequência
    mm::Image img_rest_vis = mm::freqFilter(img_ruidosa_mm, mascara);

    // PSNR entre original e restaurada
    cv::Mat img_gray_mat2 = img_gray;  // re-converter para cv::Mat
    cv::Mat img_rest_mat = img_rest_vis;  // converter mm::Image para cv::Mat
    double psnr = cv::PSNR(img_gray_mat2, img_rest_mat);
    printf("PSNR (original vs restaurada): %.2f dB\n", psnr);

    // 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)"
    };

    // Corrigir título com formatação .1f
    char titulo_psnr[100];
    sprintf(titulo_psnr, "Restaurada (PSNR=%.1f dB)", psnr);
    titulos[3] = titulo_psnr;

    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): 7.46 dB
[1] Com ruído periódico
[2] Espectro (log)
[3] Máscara notch
[4] Restaurada (PSNR=7.5 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)")
```

Figura 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.

📌 Sintesi — Filtri Spettrali

Filtro	Effetto visivo	Artefatto	Uso
Passa-basso ideale	Attenuazione intensa	*Ringing*	Illustrativo
Passa-basso Gaussiano	Attenuazione graduale	Non presenta *ringing*	Attenuazione generale
Passa-basso Butterworth	Attenuazione controllata	*Ringing* (ordini elevati)	Compromesso tra attenuazione e selettività
Passa-alto	Enfatizzazione dei bordi	Amplificazione del rumore	Rilevamento dei contorni
Notch	Rimozione selettiva delle frequenze	Possibili distorsioni locali	Rimozione del rumore periodico

La progettazione dei filtri nel dominio della frequenza consiste nella definizione di maschere spettrali. Tuttavia, effetti nel dominio spaziale, come *ringing* e sfocatura, emergono direttamente da queste scelte nello spettro.

5.6 *Wavelet* e Multirisoluzione

La Trasformata di Fourier decompone il segnale in frequenze **globali**: ogni coefficiente $F(u, v)$ riceve contributi dall'intera immagine, senza informazioni esplicite sulla localizzazione spaziale di tali frequenze. Pertanto, strutture localizzate, come i bordi, sono rappresentate in modo distribuito nello spettro.

Le ***wavelet* (ondine)** superano questa limitazione utilizzando funzioni base **localizzate nello spazio**, che possono essere traslate e scalate. Queste funzioni possiedono **supporto compatto**, ossia sono diverse

da zero solo in una regione finita del dominio, consentendo una rappresentazione simultanea in termini di **frequenza e localizzazione spaziale**.

5.6.1 Il Limite della Trasformata di Fourier: localizzazione spaziale

La Trasformata di Fourier descrive con precisione **quali frequenze sono presenti** in un segnale, ma non rappresenta esplicitamente **dove queste frequenze si verificano nello spazio**.

Nell'esperimento presentato nella Figura 5.16, due immagini con strutture localizzate in posizioni diverse producono spettri di magnitudine praticamente identici. Ciò accade perché la rappresentazione di Fourier è globale: ogni coefficiente riceve un contributo dall'intera immagine.

Di conseguenza, lo spettro di magnitudine non rappresenta esplicitamente la localizzazione dei bordi o di altre strutture, ma solo la distribuzione delle frequenze presenti. Questa limitazione ha motivato lo sviluppo di rappresentazioni multirisoluzione, come la Trasformata *Wavelet* Discreta (DWT), in grado di descrivere simultaneamente la frequenza e la localizzazione spaziale delle strutture dell'immagine.

```
%%writefile tmp/fig_05_fracasso_fourier.cpp
#define MM_OUT "tmp/fig_05_fracasso_fourier.png"
#include <opencv2/opencv.hpp>
#include <vector>
#include <string>
#include "morph.hpp"
#include <filesystem>

//| 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

int main() {
    // Cria os dois sinais binários: cruzes em posições diferentes
    cv::Mat img_sinal1 = cv::Mat::zeros(128, 128, CV_8UC1);
    img_sinal1(cv::Rect(20, 0, 5, 128)).setTo(1);
    img_sinal1(cv::Rect(0, 100, 128, 5)).setTo(1);

    cv::Mat img_sinal2 = cv::Mat::zeros(128, 128, CV_8UC1);
    img_sinal2(cv::Rect(90, 0, 5, 128)).setTo(1);
    img_sinal2(cv::Rect(0, 30, 128, 5)).setTo(1);

    // Calcula a magnitude do espectro de Fourier (log(1+|F|)) para cada sinal
    mm::Image mag1 = mm::spectrumMag(mm::Image(img_sinal1));
    mm::Image mag2 = mm::spectrumMag(mm::Image(img_sinal2));

    // Exibe os sinais e seus espectros
    mm::show({mm::Image(img_sinal1), mag1, mm::Image(img_sinal2), mag2},
             MM_OUT,
             {"Sinal A", "Espectro A", "Sinal B (Deslocado)", "Espectro B"},
             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)")
```

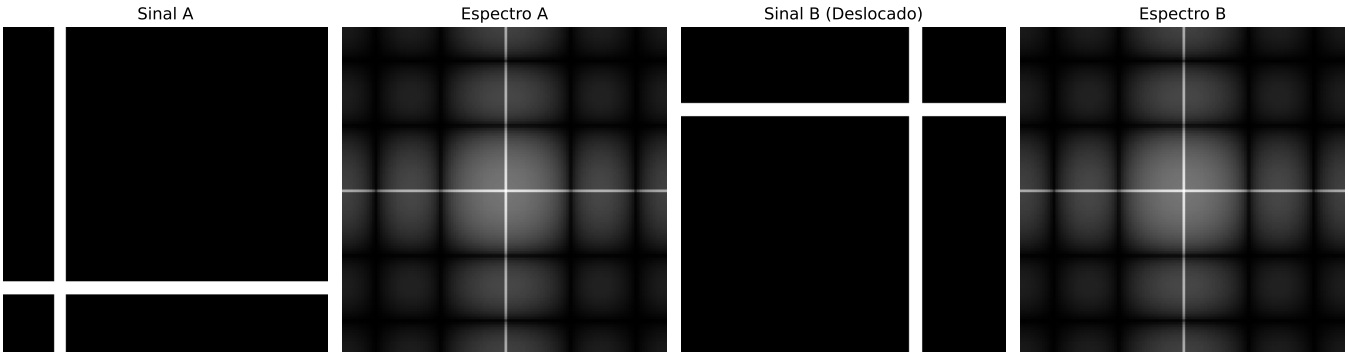


Figura 5.16: Fourier global é cego para a posição. Os espectros não dizem onde as bordas estão.

5.6.2 Trasformata *Wavelet* Discreta 2D

La Trasformata *Wavelet* Discreta (DWT) applica, separatamente nelle direzioni orizzontale e verticale, due filtri complementari: um **passa-basso** h (approssimazione) e um **passa-alto** g (dettagli), seguiti da sottocampionamento per um fattore de 2 in ciascuna dimensione. Questo processo produce quattro sottobande, i cui nomi indicano la combinazione dei filtri applicati in ciascuna direzione (L = *Low-pass*, passa-basso; H = *High-pass*, passa-alto). Le caratteristiche di ciascuna sottobanda sono riassunte nella Tabela 5.3.

$$\mathrm{DWT}(f) = \{ \underbrace{\mathrm{LL}}_{\text{approssimazione}}, \underbrace{\mathrm{LH}}_{\text{dettagli orizzontali}}, \underbrace{\mathrm{HL}}_{\text{dettagli verticali}}, \underbrace{\mathrm{HH}}_{\text{dettagli diagonali}} \}.$$

Tabela 5.3: Sottobande prodotte dalla Trasformata *Wavelet* Discreta 2D (DWT), indicando i filtri applicati in ciascuna direzione e il contenuto predominante de ciascuna componente.

Sottobanda	Filtri applicati	Contenuto visivo
LL	basso × basso	Approssimazione dell'immagine (versione smussata e ridotta)
LH	basso × alto	Bordi orizzontali e variazioni verticali
HL	alto × basso	Bordi verticali e variazioni orizzontali
HH	alto × alto	Dettagli diagonali e trame

La decomposizione può essere applicata ricorsivamente sulla sottobanda LL, generando uma rappresentazione multirisoluzione. Dopo J livelli, si ottiene uma struttura com $3J + 1$ sottobande, in cui ogni nuovo livello riduce la risoluzione della componente de approssimazione.

> **i** Collegamento com le CNN
>
> La decomposizione multirisoluzione delle *wavelet* possiede uma relazione concettuale com le rappresentazioni gerarchiche utilizzate nelle reti neurali convoluzionali (CNN). In entrambi i casi, successive fasi di filtraggio e riduzione della risoluzione producono descrizioni sempre più astratte dell'immagine. Tuttavia, le ***wavelet* utilizzano filtri matematicamente definiti e ricostruibili**, mentre le **CNN apprendono i propri filtri durante l'addestramento**.

5.6.3 Famiglie di *Wavelet*

Diverse famiglie di *wavelet* presentano compromessi distinti tra **supporto spaziale**, regolarità e capacità di compressione. Il **supporto** corrisponde all'estensione della funzione *wavelet* nel dominio spaziale: quanto più piccolo è il supporto, tanto più localizzata è la funzione; quanto più grande, tanto più liscia tende a essere la sua rappresentazione, sebbene com un costo computazionale maggiore. La Tabela 5.4 confronta alcune delle famiglie più utilizzate.

Tabela 5.4: Confronto tra famiglie di *wavelet*, evidenziando la lunghezza del supporto, il numero de momenti nulli, la simmetria e le applicazioni tipiche.

Wavelet	Lunghezza del supporto	Momenti nulli	Simmetria	Uso tipico
Haar	2	1	Asimmetrica	Introduzione e analisi de base
Daubechies db4	8	4	Asimmetrica	Compressione e analisi generale

Wavelet	Lunghezza del supporto	Momenti nulli	Simmetria	Uso tipico
Symlet sym4	8	4	Quasi simmetrica	Ricostruzione di segnali
Biortogonale 5/3	5/3	2/2	Simmetrica	JPEG 2000 senza perdita
Biortogonale 9/7	9/7	4/4	Simmetrica	JPEG 2000 con perdita

I **momenti nulli** misurano la capacità della *wavelet* di rappresentare regioni uniformi dell'immagine con pochi coefficienti diversi da zero. Una *wavelet* con p momenti nulli annulla esattamente i polinomi di grado fino a $p-1$. Di conseguenza, quanto maggiore è il numero di momenti nulli, tanto maggiore tende a essere l'efficienza di compressione nelle regioni omogenee, sebbene ciò implichi generalmente funzioni con supporto più esteso.

La Figura 5.17 presenta le funzioni di base (*wavelet*) $\psi(t)$ nel dominio spaziale. Queste funzioni possiedono **supporto compatto**, cioè sono diverse da zero solo in una regione finita del dominio, contrariamente alle sinusoidi della Trasformata di Fourier, che si estendono su tutto il dominio.

```
%%writefile tmp/fig_05_wavelet_functions.cpp
#define MM_OUT "tmp/fig_05_wavelet_functions.png"
//| 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

#include <opencv2/opencv.hpp>
#include <vector>
#include <string>
#include "morph.hpp"
#include <filesystem>


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::string> labels_h = {"psi Haar"};
    std::vector<cv::Scalar> colors_h = {cv::Scalar(180, 60, 40)};
    mm::Image chart_h = mm::lineChart(xs_h, ys_h, labels_h, colors_h,
                                      "Ondaleta Haar (psi)", "t");

    std::vector<std::vector<double>> xs_d = {x_d};
    std::vector<std::vector<double>> ys_d = {psi_d};
    std::vector<std::string> labels_d = {"psi Daubechies 4"};
    std::vector<cv::Scalar> colors_d = {cv::Scalar(60, 140, 40)};
    mm::Image chart_d = mm::lineChart(xs_d, ys_d, labels_d, colors_d,
                                      "Ondaleta Daubechies 4 (psi)", "t");

    std::vector<mm::Image> charts = {chart_h, chart_d};
    std::vector<std::string> titles = {"Haar", "Daubechies 4"};
    mm::show(charts, MM_OUT, titles, 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)")
```

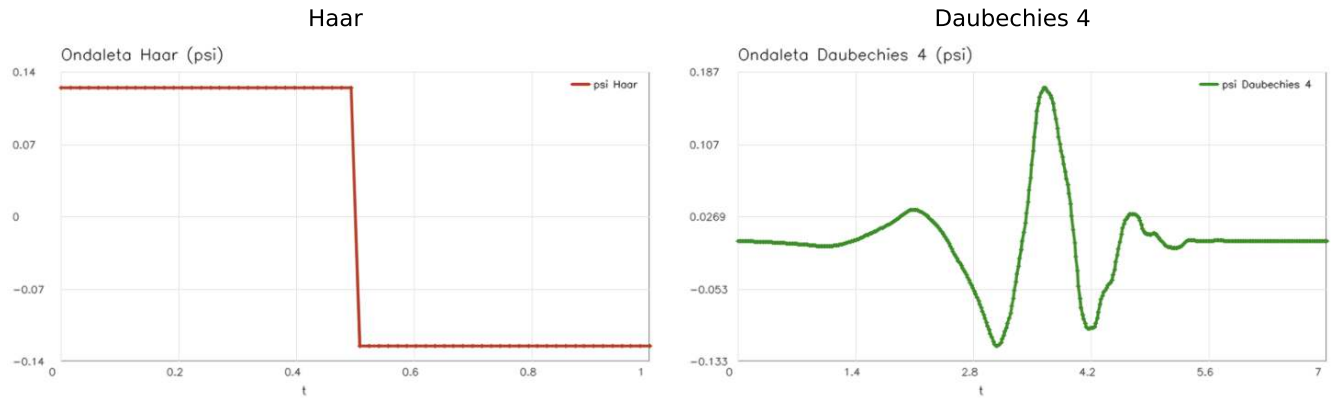


Figura 5.17: Funções da *Wavelet* ( ). Note como elas rapidamente decaem para zero (suporte compacto), ao contrário das senoides infinitas de Fourier.

Il diagrama della Figura 5.18 illustra l'analisi multirisoluzione eseguita dalla DWT, in cui la sottobanda di approssimazione (LL) viene successivamente decomposta, formando una rappresentazione gerarchica su due livelli.

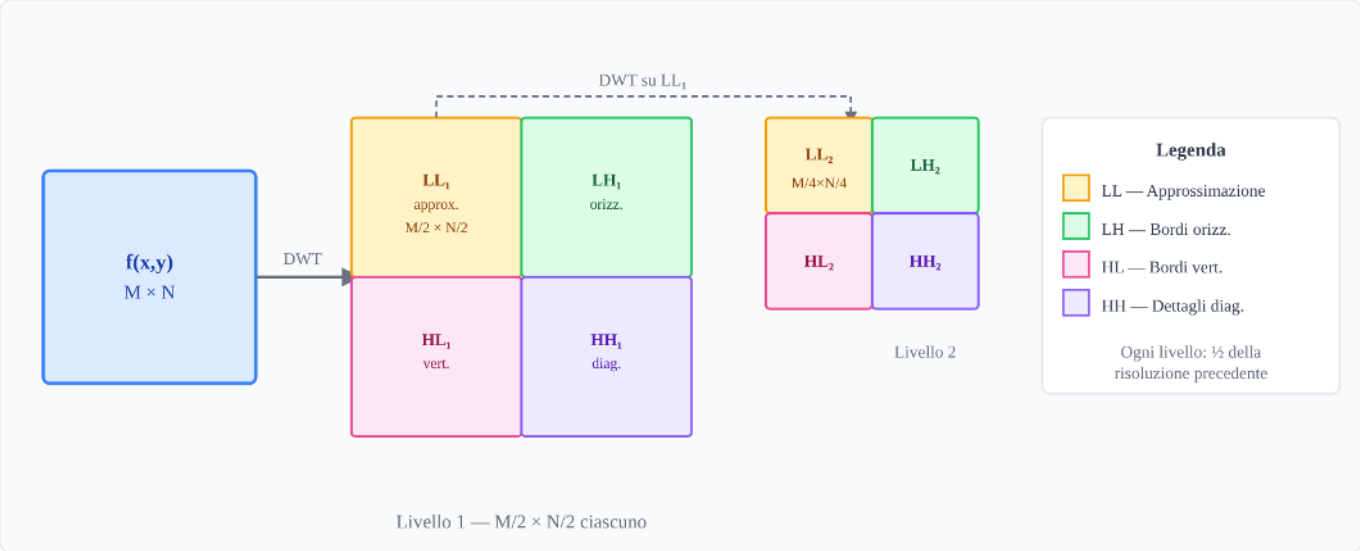


Figura 5.18: Diagramma della decomposizione *wavelet* 2D su due livelli.

Il simulatore Figura 5.19 consente di esplorare in modo interattivo la Trasformata *Wavelet* Discreta 2D (DWT) utilizzando la *wavelet* di Haar. La decomposizione in sottobande evidenzia la separazione tra la componente di approssimazione e le componenti di dettaglio dell'immagine.

I diversi modelli di ingresso permettono di osservare il comportamento direzionale dei filtri. In immagini con bordi orizzontali e verticali, le sottobande LH e HL evidenziano, rispettivamente, le variazioni verticali e orizzontali dell'intensità. Nelle regioni a variazione graduale, la maggior parte dell'energia si concentra nella sottobanda di approssimazione LL, mentre le sottobande di dettaglio presentano coefficienti prossimi allo zero.

L'analisi multirisoluzione può essere osservata anche aumentando il numero di livelli di decomposizione. In tal caso, solo la sottobanda LL_1 viene nuovamente decomposta, generando le sottobande LL_2, LH_2, HL_2 e HH_2, che formano il secondo livello della rappresentazione gerarchica.

In pattern costituiti da regioni omogenee di grande estensione, come una sfumatura graduale o una scacchiera composta da blocchi di grandi dimensioni, l'energia rimane prevalentemente concentrata nella sottobanda LL. Nella sfumatura, ciò avviene perché le differenze tra pixel adiacenti sono piccole. Nella scacchiera, invece, i pixel possiedono praticamente la stessa intensità all'interno di ciascun blocco, cosicché solo i confini tra

blocchi producono coefficienti non nulli nelle sottobande di dettaglio. Poiché tali confini occupano solo una piccola frazione dell'immagine, il loro contributo all'energia totale rimane ridotto.

Per rendere possibile l'analisi visiva di queste variazioni sottili, il simulatore incorpora un controllo del guadagno di contrasto dei dettagli (variabile da 1 a 8). Questo parametro funziona come un fattore di amplificazione lineare applicato esclusivamente ai coefficienti delle sottobande di dettaglio (LH, HL e HH) prima della loro visualizzazione a schermo. Negli scenari di transizione graduale (come il gradiente) o di uniformità locale (come l'interno dei blocchi della scacchiera), le differenze numeriche calcolate dal filtro passa-alto di Haar producono coefficienti molto prossimi allo zero, il che renderebbe i quadranti corrispondenti scuri e impercettibili a occhio nudo. Moltiplicando questi valori per il guadagno, il simulatore recupera visivamente le strutture ad alta frequenza nascoste ed enfatizza l'orientamento dei bordi residui.

Il grafico dell'energia per sottobanda quantifica tale distribuzione tra la componente di approssimazione e le componenti di dettaglio, dimostrando che il guadagno visivo non altera la metrica originale dell'energia. Nelle immagini naturali, la maggior parte dell'energia si concentra nella sottobanda LL, mentre le sottobande LH, HL e HH rappresentano principalmente bordi, texture e altre variazioni locali dell'intensità.

5.6.4 Analisi Multirisoluzione con la DWT 2D

La Trasformata *Wavelet* Discreta 2D (DWT) decompone un'immagine in componenti di approssimazione e dettaglio, organizzate in modo gerarchico su diverse scale e orientazioni. Poiché le sottobande di dettaglio in immagini naturali spesso presentano coefficienti a basso contrasto, gli esempi pratici che seguono utilizzano un pattern geometrico sintetico generato in Python. Questo approccio replica il comportamento del simulatore della Figura 5.19, rendendo visivamente espliciti gli effetti del filtraggio spaziale e della decomposizione multirisoluzione.

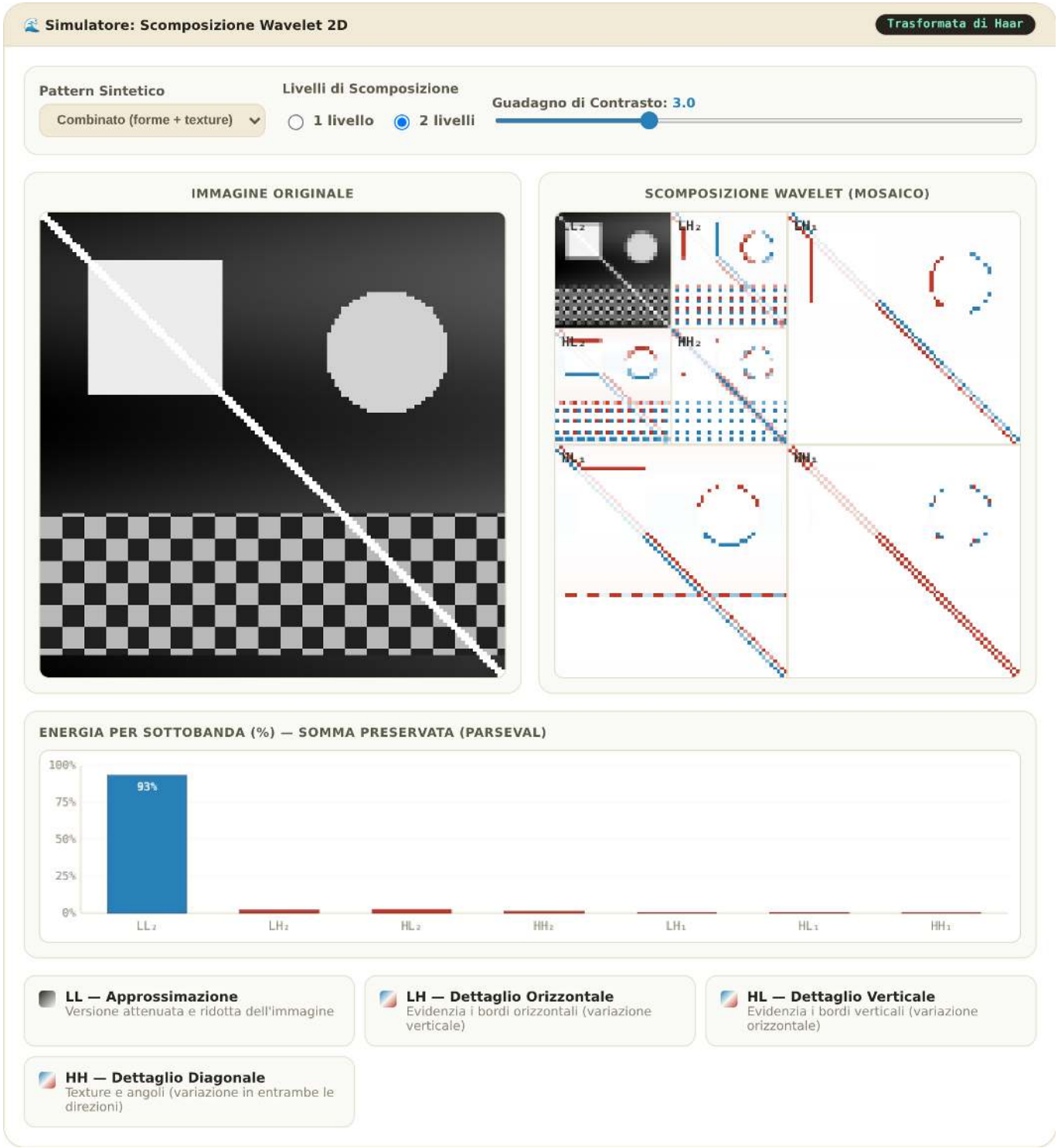
5.6.4.1 Scomposizione a Mosaico su Più Livelli

La Figura 5.20 illustra la struttura gerarchica della DWT su due livelli utilizzando la *wavelet* di Haar. Il processo si basa sull'applicazione combinata di filtri passa-basso e passa-alto nelle direzioni orizzontale e verticale, seguiti da un sottocampionamento con fattore 2.

Nel primo livello, l'immagine originale genera la sottobanda di approssimazione (LL_1) e le componenti di dettaglio orizzontale (LH_1), verticale (HL_1) e diagonale (HH_1). Nell'analisi multirisoluzione, la sottobanda LL_1 viene nuovamente filtrata e sottocampionata, producendo il secondo livello di scomposizione (LL_2, LH_2, HL_2 e HH_2).

Per consentire l'interpretazione visiva delle componenti di dettaglio, il codice estrae il valore assoluto dei loro coefficienti e applica una normalizzazione lineare (*min-max*) per occupare l'intera gamma dinamica dei toni di grigio [0, 255]. Questa operazione trasforma le regioni omogenee (coefficienti nulli) in nero ed evidenzia in bianco i bordi e le texture estratte a ciascuna scala e orientazione.

```
%%writefile tmp/fig_05_dwt_subbandas.cpp
#define MM_OUT "tmp/fig_05_dwt_subbandas.png"
//| 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 <iostream>
#include <cmath>
#include <vector>
#include <string>
#include "morph.hpp"
```



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.it/cap05/sim-05-wavelet.html

Figura 5.19: Simulazione della decomposizione *wavelet* 2D.

```
// Função para gerar a 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 * (static_cast<double>(x) / N) + 15 * std::sin(y / 24.0);
            // Quadrado
            if (24 < x && x < 100 && 24 < y && y < 100) {
                v = 225;
            }
            // Círculo
            double cx = 190, cy = 76, r = 34;
            if (std::pow(x - cx, 2) + std::pow(y - cy, 2) < 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(255.0, v));
        }
    }
    cv::Mat img_uint8;
    img.convertTo(img_uint8, CV_8U);
    return img_uint8;
}

// Normaliza subbanda para visualização [0,255]
cv::Mat sb_vis(const cv::Mat& sb) {
    cv::Mat abs_sb, norm_sb, uint8_sb;
    cv::absdiff(sb, cv::Scalar(0), abs_sb);
    cv::normalize(abs_sb, norm_sb, 0, 255, cv::NORM_MINMAX);
    norm_sb.convertTo(uint8_sb, CV_8U);
    return uint8_sb;
}

int main() {
    // Substitui a imagem escura de moedas pelo padrão sintético claro
    cv::Mat img_gray = gerar_imagem_sintetica(256);
    cv::Mat img_float;
    img_gray.convertTo(img_float, CV_64F);

    //   Decomposição wavelet 2 níveis
    std::string wavelet = "haar";

    // Nível 1
    mm::Subbands s1 = mm::dwt2(img_float, wavelet);
    cv::Mat LL1 = s1.LL, LH1 = s1.LH, HL1 = s1.HL, HH1 = s1.HH;

    // Nível 2 (aplicado sobre LL1)
    mm::Subbands s2 = mm::dwt2(LL1, wavelet);
    cv::Mat LL2 = s2.LL, LH2 = s2.LH, HL2 = s2.HL, HH2 = s2.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)")
```

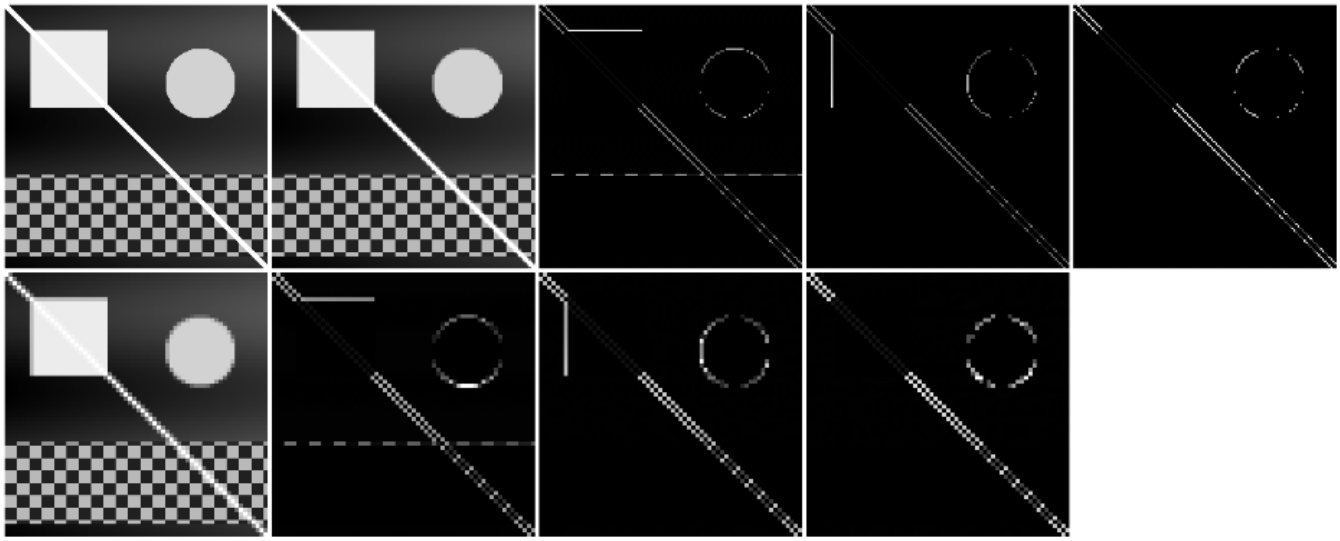


Figura 5.20: Decomposição *wavelet* 2D de 2 níveis com *wavelet* Haar: subbandas LL, LH, HL, HH em cada nível. As subbandas de detalhe revelam estruturas orientadas em diferentes escalas utilizando um padrão sintético.

5.6.4.2 Il Compromesso tra Localizzazione e Morbidezza

La scelta della funzione di base (*wavelet*) influenza direttamente il modo in cui le caratteristiche dell'immagine vengono distribuite e codificate dai coefficienti della DWT. La Figura 5.21 confronta i risultati pratici ottenuti applicando quattro famiglie distinte sul pattern geometrico sintetico: `haar`, `db4`, `sym4` e `bior2.2`.

Poiché possiede un supporto corto e una forma a funzione a gradino, la *wavelet* di Haar produce coefficienti altamente localizzati in corrispondenza delle discontinuità spaziali, generando bordi sottili e nitidi nelle sottobande di dettaglio. Al contrario, famiglie come Daubechies (`db4`) e Symlets (`sym4`), che presentano un supporto maggiore (filtri più lunghi) e un numero più elevato di momenti nulli, generano risposte più morbide e distribuite attorno alle transizioni, il che può introdurre lievi oscillazioni o effetti di sfocatura sui confini netti.

Questo comportamento evidenzia il classico compromesso (*trade-off*) dell'analisi multirisoluzione: supporti più piccoli favoriscono la localizzazione spaziale esatta dei bordi, mentre supporti più ampi e un maggior numero di momenti nulli tendono a produrre rappresentazioni più sparse e morbide. Questa morbidezza e la capacità di attenuazione delle alte frequenze garantiscono una maggiore efficienza nella compattazione dell'energia, caratteristiche fondamentali per applicazioni di compressione dei dati e rimozione del rumore (*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 <functional>
#include "morph.hpp"
#include <iostream>
```

```
// Função auxiliar para visualizar subbandas (normalizar para 0-255)
mm::Image sb_vis(const cv::Mat& sb) {
    cv::Mat normalized;
    // Normalizar para visualização
    double minVal, maxVal;
    cv::minMaxLoc(sb, &minVal, &maxVal);
    if (maxVal - minVal > 0) {
        sb.convertTo(normalized, CV_8U, 255.0 / (maxVal - minVal), -minVal * 255.0 / (maxVal -
minVal));
    } else {
        normalized = cv::Mat::zeros(sb.size(), CV_8U);
    }
    return mm::Image(normalized);
}

// Função para gerar imagem sintética (fallback)
cv::Mat gerar_imagem_sintetica(int N = 256) {
    cv::Mat img(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);
            if (24 < x && x < 100 && 24 < y && y < 100) v = 225;
            int cx = 190, cy = 76, r = 34;
            if ((x - cx) * (x - cx) + (y - cy) * (y - cy) < 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(255.0, v));
            img.at<double>(y, x) = v;
        }
    }
    cv::Mat img8u;
    img.convertTo(img8u, CV_8U);
    return img8u;
}

int main() {
    // Garante que img_gray e img_float utilizem o mesmo padrão sintético claro
    cv::Mat img_gray;

    // Fallback: gerar imagem sintética (não temos acesso a variáveis de outros blocos)
    img_gray = gerar_imagem_sintetica(256);

    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) {
        // pywt.dwt2 -> mm::dwt2
        mm::Subbands sub = mm::dwt2(img_float, wname);
        cv::Mat LL = sub.LL;
        cv::Mat LH = sub.LH;
        cv::Mat HL = sub.HL;
        cv::Mat HH = sub.HH;
```

```
        imgs_comp.push_back(sb_vis(LL));
        imgs_comp.push_back(sb_vis(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)")
```

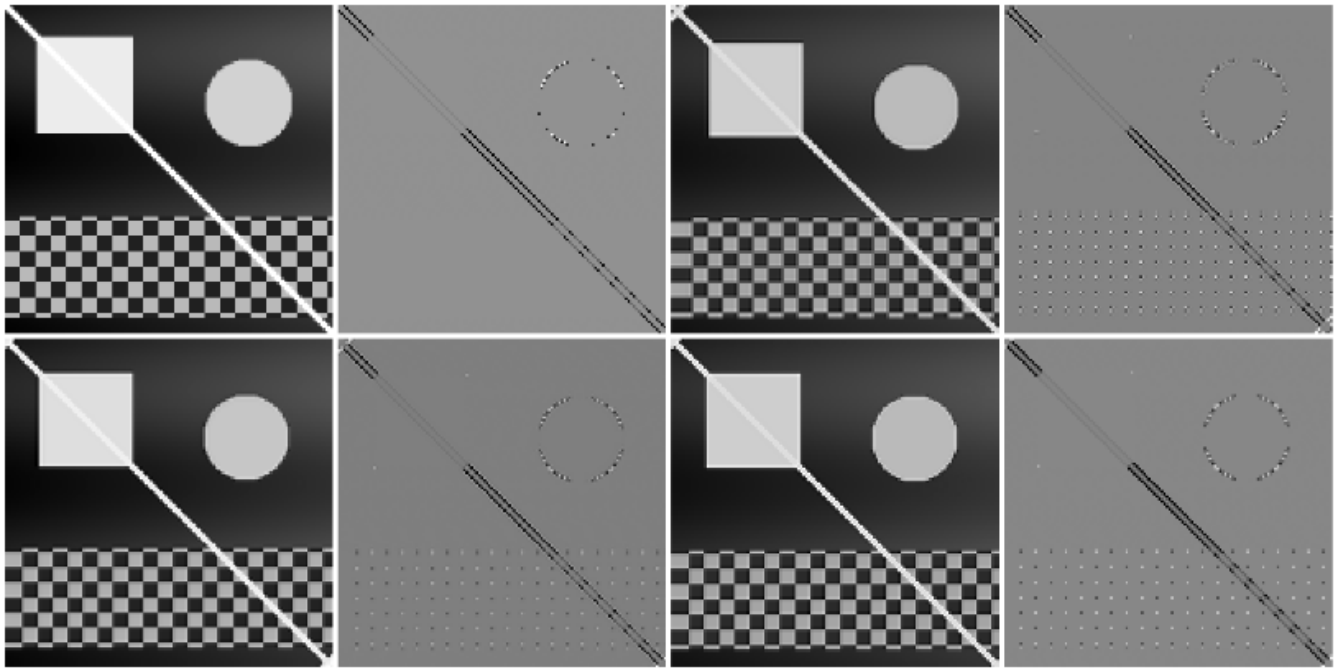


Figura 5.21: Comparação entre famílias de *wavelets*: Haar, db4, sym4 e bior2.2. Subbanda LL (aproximação) e HH (diagonal) para cada escolha, ilustrando o compromisso entre compactação e suavidade com base no padrão sintético.

5.6.4.3 Limiarizzazione dei Coefficienti e Compressione

Una delle principali applicazioni della Trasformata *Wavelet* Discreta (DWT) è la compressione dei dati, favorita dalla capacità di rappresentazione **sparsa** dei coefficienti. La Figura 5.22 illustra l'effetto della limiarizzazione netta (*hard thresholding*), tecnica in cui i coefficienti di dettaglio con magnitudine inferiore a una soglia T vengono integralmente azzerati prima del processo di sintesi eseguito dalla Trasformata *Wavelet* Discreta Inversa (IDWT).

Man mano che la soglia T viene aumentata, un volume crescente di coefficienti ad alta frequenza viene azzerato. Poiché concentrano meno energia, la rimozione di queste componenti riduce considerevolmente la quantità di informazione necessaria per rappresentare l'immagine, mantenendo intatta la componente di approssimazione globale (la sottobanda LL più profonda) per preservare la struttura macro. Visivamente, questo scarto di coefficienti si manifesta attraverso la scomparsa progressiva delle trame fini e la levigatura delle transizioni brusche di intensità.

La fedeltà dell'immagine ricostruita rispetto all'originale è quantificata dalla metrica del **Picco del Rapporto Segnale-Rumore (PSNR, *Peak Signal-to-Noise Ratio*)**, espressa in decibel (dB). Valori più elevati di PSNR indicano una distorsione minore e una maggiore prossimità matematica al segnale originale. L'esperimento pratico evidenzia il decadimento graduale del PSNR all'aumentare dell'aggressività della limiarizzazione, consentendo di valutare numericamente la soglia ottimale per il bilanciamento tra compressione e degrado visivo.

```
%%writefile tmp/fig_05_dwt_reconstrucao.cpp
#define MM_OUT "tmp/fig_05_dwt_reconstrucao.png"
// Compile with: g++ -std=c++17 -O2 snippet.cpp -o snippet $(pkg-config --cflags --libs
opencv4) -I/path/to/morph/include
#include <opencv2/opencv.hpp>
#include <vector>
#include <string>
#include <cmath>
#include <cstdint>
```

```
#include "morph.hpp"

// Helper function to generate synthetic image (same as Python fallback)
cv::Mat gerar_imagem_sintetica(int N = 256) {
    cv::Mat img(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(double(y) / 24);
            if (24 < x && x < 100 && 24 < y && y < 100) v = 225;
            int cx = 190, cy = 76, r = 34;
            if ((x - cx) * (x - cx) + (y - cy) * (y - cy) < 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;
            img.at<double>(y, x) = std::max(0.0, std::min(255.0, v));
        }
    }
    cv::Mat img8;
    img.convertTo(img8, CV_8U);
    return img8;
}

cv::Mat dwt_threshold_reconstruct(const cv::Mat& img, const std::string& wavelet = "db4", int
nivel = 2, double threshold = 0.0) {
    cv::Mat img64;
    img.convertTo(img64, CV_64F);

    // Perform decomposition using mm::wavedec2
    mm::WaveDec2 c = mm::wavedec2(img64, wavelet, nivel);

    // Apply thresholding to detail coefficients
    mm::WaveDec2 c_t;
    c_t.LL = c.LL.clone();  // LL not thresholded
    c_t.detail.resize(c.detail.size());

    for (size_t j = 0; j < c.detail.size(); ++j) {
        c_t.detail[j].resize(3);
        for (int k = 0; k < 3; ++k) {
            c_t.detail[j][k] = mm::wave_threshold(c.detail[j][k], threshold, "hard");
        }
    }

    // Reconstruct via mm::waverec2
    cv::Mat rec = mm::waverec2(c_t, wavelet);

    // Crop to original dimensions and clip to [0, 255]
    rec = rec(cv::Rect(0, 0, img.cols, img.rows)).clone();
    cv::Mat rec8;
    rec.convertTo(rec8, CV_8U);
    return rec8;
}

int main() {
    // Ensure img_gray uses the same clear synthetic pattern
    cv::Mat img_gray = gerar_imagem_sintetica(256);

    std::vector<int> thresholds = {0, 10, 30, 60, 100};
    std::vector<mm::Image> imgs_thr;
    std::vector<std::string> titles_thr;
```

```
    imgs_thr.push_back(mm::Image(img_gray));
    titles_thr.push_back("Original");

    for (int t : thresholds) {
        cv::Mat rec = dwt_threshold_reconstruct(img_gray, "db4", 2, double(t));
        double psnr = cv::PSNR(img_gray, rec);
        imgs_thr.push_back(mm::Image(rec));
        titles_thr.push_back("T=" + std::to_string(t) + "  PSNR=" +
                             std::to_string(psnr).substr(0, std::to_string(psnr).find('.') + 2)
+ " dB");
    }

    mm::show(imgs_thr, 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=361.2 dB
[3] T=10  PSNR=42.4 dB
[4] T=30  PSNR=32.5 dB
[5] T=60  PSNR=28.0 dB
[6] T=100  PSNR=23.1 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)")
```

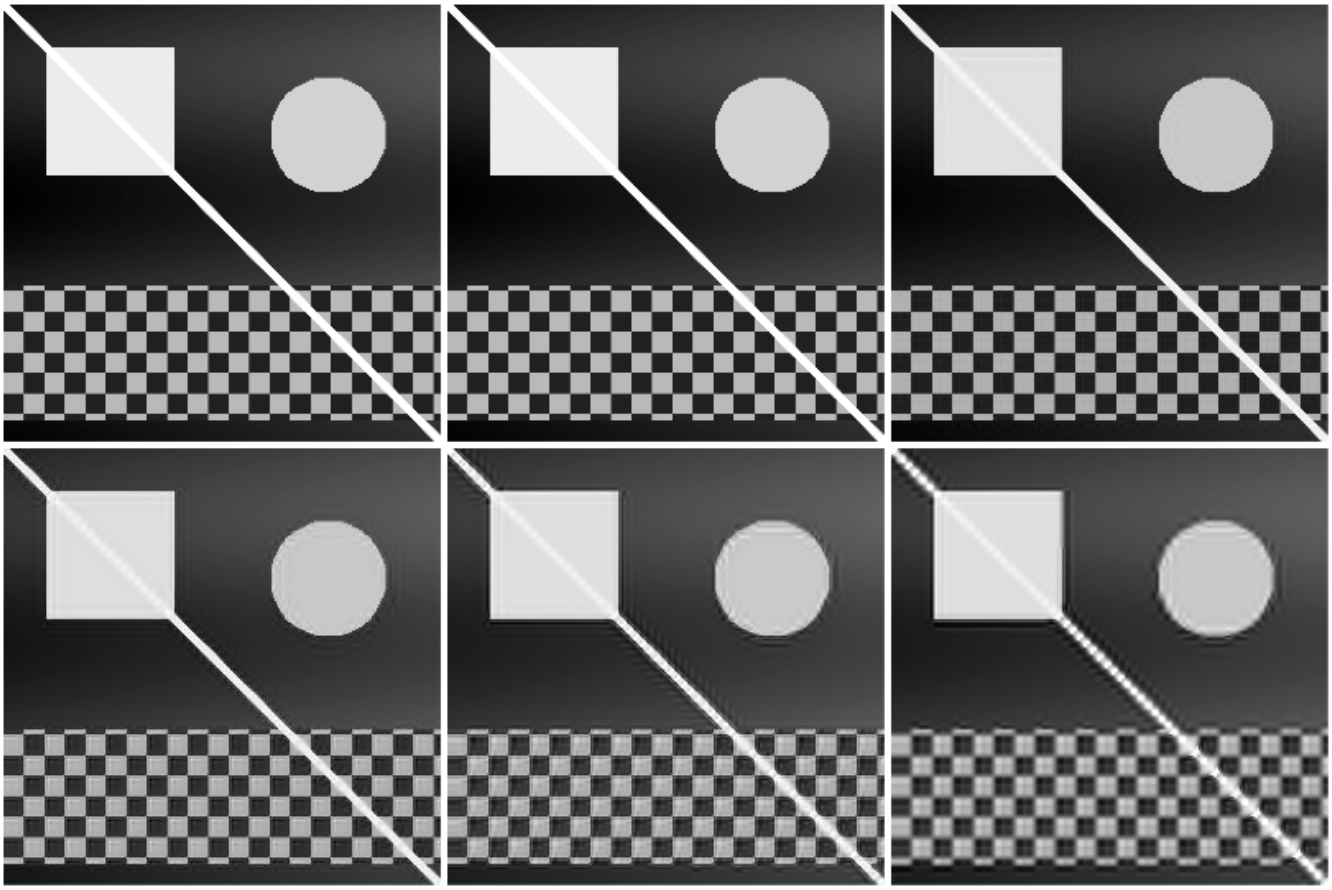


Figura 5.22: Reconstrução *wavelet* com limiarização de coeficientes (*hard thresholding*): à medida que o limiar aumenta, mais detalhes são zerados, produzindo imagens progressivamente mais suaves. Métrica PSNR quantifica a perda de qualidade sobre o padrão sintético.

Sintesi — Fourier vs. *Wavelet*: quando utilizzare ciascun approccio?

La Tabella 5.5 sintetizza le principali differenze strutturali e operative tra la Trasformata Discreta di Fourier (DFT) e la Trasformata *Wavelet* Discreta (DWT).

Tabella 5.5: Confronto tra la Trasformata Discreta di Fourier (DFT) e la Trasformata *Wavelet* Discreta (DWT), evidenziandone le principali caratteristiche e applicazioni.

Criterio	Fourier (DFT)	*Wavelet* (DWT)
Funzioni di base	Sinusoidi di supporto infinito	Funzioni di supporto compatto
Localizzazione spaziale	Non esplicita (globale)	Esplicita (locale)
Filtraggio spettrale	Eccellente per il controllo fine delle frequenze	Basata su sottobande (scale)
Compressione delle immagini	Base della DCT (JPEG tradizionale)	Base della DWT (JPEG 2000)
Analisi multiscala	No	Sì
Rimozione del rumore periodico	Altamente efficiente	Poco indicata
Segnali non stazionari	Limitata	Altamente efficiente

In termini pratici, la DFT si consolida come lo strumento ideale per l'analisi spettrale pura, la progettazione di filtri selettivi nel dominio della frequenza e l'attenuazione di rumori periodici e armonici. D'altro canto, la DWT eccelle in scenari che richiedono la rigorosa preservazione della localizzazione spaziale delle

caratteristiche associata al loro contenuto frequenziale, distinguendosi nella compressione dei dati, nell'analisi multirisoluzione e nell'elaborazione di transizioni brusche. Pertanto, entrambe le trasformate devono essere comprese come tecniche perfettamente complementari, che tracciano percorsi distinti e specifici per la risoluzione di problemi nell'ambito dell'elaborazione digitale di immagini e visione artificiale (PDI-VC).

i Analogie con l'Audio: Limitazioni e Precauzioni

Nel tracciare analogie tra l'elaborazione delle immagini e l'audio, è importante considerare le differenze fondamentali:

- Nei sistemi audio stereo/multicanale, la fase tra i canali è cruciale per la percezione della localizzazione spaziale (differenze interaurali di fase e di tempo).
- Nei sistemi monoaurali, la fase ha un'influenza percettiva limitata — l'orecchio umano è relativamente insensibile alla fase assoluta di componenti sinusoidali isolate.
- Nelle immagini, la fase della DFT è sempre fondamentale per la localizzazione spaziale delle strutture, indipendentemente dal fatto che l'immagine sia monocromatica o a colori.

L'analogia tra la fase nell'audio e la fase nelle immagini deve essere utilizzata con cautela, evidenziando che, sebbene entrambe trasportino informazioni sull'organizzazione spaziale/temporale del segnale, i meccanismi percettivi sono fondamentalmente differenti.

5.7 Compressione delle Immagini

Mentre le *wavelet* stabiliscono il fondamento teorico dello standard JPEG 2000, lo standard JPEG tradizionale si basa sulla **Trasformata Discreta dei Coseni (DCT, *Discrete Cosine Transform*)**. Nonostante le differenze strutturali, entrambi gli approcci condividono lo stesso principio fondamentale: compattare l'energia dell'immagine in un numero ridotto di coefficienti e scartare le componenti di minore rilevanza con impatto visivo minimo.

L'obiettivo centrale della compressione è ridurre il volume di dati necessario per l'archiviazione o la trasmissione di un'immagine. Tale processo è reso possibile dall'identificazione e dall'eliminazione di **ridondanze** strutturali e percettive.

5.7.1 Tassonomia delle Ridondanze

Lo sviluppo di algoritmi di compressione si fonda sull'identificazione e sull'eliminazione di tre categorie principali di ridondanza, sintetizzate nella Tabella 5.6.

Tabella 5.6: Categorie di ridondanza nelle immagini digitali e i rispettivi meccanismi di sfruttamento.

Tipo	Definizione	Approccio di Sfruttamento
Spaziale (interpixel)	Elevata correlazione e dipendenza statistica tra pixel adiacenti.	DCT, DWT e codifica predittiva.
Spettrale (intercanale)	Correlazione statistica tra i canali di colore di una stessa immagine.	Trasformazioni dello spazio colore (es: RGB in YC_bC_r).
Psicovisuale	Insensibilità del sistema visivo umano (SVH) alle variazioni ad alta frequenza e a basso contrasto.	Processi di quantizzazione selettiva dei coefficienti.

A seconda della preservazione dell'informazione originale dopo il processo di decodifica, i metodi di compressione si dividono in due classi fondamentali:

- **Senza perdita (*lossless*):** Garantisce una ricostruzione bit per bit identica all'immagine originale. Viene impiegata in scenari in cui l'integrità dei dati è strettamente critica, come nelle immagini mediche, nella diagnostica per immagini e nell'archiviazione di documenti testuali.

- **Con perdita (*lossy*):** Ammette l'introduzione di una distorsione controllata nel segnale in cambio di tassi di compressione sostanzialmente più elevati. È l'approccio standard per le fotografie di consumo e lo *streaming* video, ecosistemi nei quali il SVH tollera piccole attenuazioni dell'alta frequenza senza percezione di degrado della qualità visiva.

5.7.2 Trasformata del Coseno Discreta (DCT-II 2D)

La **Trasformata del Coseno Discreta** (DCT) costituisce l'operazione centrale dello standard JPEG. A differenza della DFT, che utilizza una base complessa, la DCT si basa su funzioni trigonometriche puramente reali. Per un blocco immagine $f(x, y)$ di dimensioni $N \times N$, la **DCT-II 2D** mappa il segnale spaziale nel dominio delle frequenze spaziali, generando la matrice dei coefficienti $C(u, v)$ tramite:

$$C(u,v) = \alpha(u)\,\alpha(v) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x,y)\, \cos\left[\frac{\pi(2x+1)u}{2N}\right] \cos\left[\frac{\pi(2y+1)v}{2N}\right] \tag{5.9}$$

dove i fattori di normalizzazione ortogonale sono dati da $\alpha(0) = \sqrt{1/N}$ e $\alpha(k) = \sqrt{2/N}$ per $k > 0$.

Ogni coefficiente $C(u, v)$ quantifica il contributo — o "peso" — di una specifica frequenza spaziale all'interno di quel blocco. Il termine $C(0, 0)$ è denominato **componente DC** e rappresenta l'intensità media del blocco (frequenza nulla). I restanti coefficienti, chiamati **componenti AC** (*Alternating Current*), corrispondono a frequenze spaziali progressivamente più elevate.

5.7.3 Le Funzioni di Base della DCT

Da una prospettiva geometrica, la Equazione 5.9 realizza la proiezione del blocco di pixel su un insieme di funzioni ortogonali. Per il caso standard del JPEG ($N = 8$), il blocco spaziale viene decomposto in una combinazione lineare di **64 funzioni di base** bidimensionali, denotate da $B_{u,v}(x, y)$ e generate dal prodotto di funzioni cosinusoidali:

$$B_{u,v}(x,y) = \cos\left[\frac{\pi(2x+1)u}{16}\right] \cos\left[\frac{\pi(2y+1)v}{16}\right]$$

In questo modo, l'operazione inversa può essere interpretata come la ricostruzione esatta del blocco originale tramite la somma ponderata di queste 64 matrici di base, dove ogni coefficiente $C(u, v)$ funge da peso analitico della rispettiva componente armonica.

La **frequenza spaziale** indicata dagli indici (u, v) determina il numero di cicli di oscillazione lungo le dimensioni orizzontali e verticali del blocco. Come illustrato nella Figura 5.23 — il cui codice isola ciascuna base applicando la trasformazione inversa su impulsi unitari —, queste 64 funzioni sono organizzate in una matrice 8×8. L'angolo superiore sinistro ($u = 0, v = 0$) mostra il pattern uniforme a frequenza nulla (DC), mentre il progredire verso destra (asse u) o verso il basso (asse v) mappa variazioni armoniche progressivamente maggiori, rappresentando transizioni rapide, bordi e trame nelle orientazioni orizzontali, verticali e diagonali.

i DCT vs DFT: Vantaggio della Compattazione dell'Energia

Sia la DCT che la DFT mappano un blocco spaziale $N \times N$ in una matrice di coefficienti della stessa dimensione. Tuttavia, per immagini naturali, la DCT presenta una maggiore efficienza nella **compattazione dell'energia** alle basse frequenze. Ciò avviene perché la DCT assume implicitamente una simmetria pari del segnale ai bordi del blocco, il che equivale a un'estensione periodica continua, minimizzando l'effetto di dispersione spettrale (*ringing*). Di conseguenza, la maggior parte dei coefficienti AC decade rapidamente verso valori prossimi allo zero, ottimizzando il *pipeline* di compressione senza introdurre una degradazione visiva percepibile.

```
%%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.
    const int tile = 32;
    cv::Mat montagem(8 * tile, 8 * tile, CV_8UC1);
    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);
            base = mm::idct2(coef);
            cv::normalize(base, base, 0, 255, cv::NORM_MINMAX, CV_8U);
            cv::Mat base_resized;
            cv::resize(base, base_resized, cv::Size(tile, tile), 0, 0, cv::INTER_NEAREST);
            base_resized.copyTo(montagem(cv::Rect(j * tile, i * tile, tile, tile)));
        }
    }

    mm::Image montagem_img = montagem;
    std::vector<mm::Image> images = {montagem_img};
    std::vector<std::string> titles = {"As 64 bases da DCT-II 8×8 (DC no topo-esquerdo)"};
    mm::show(images, MM_OUT, titles, 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)")
```

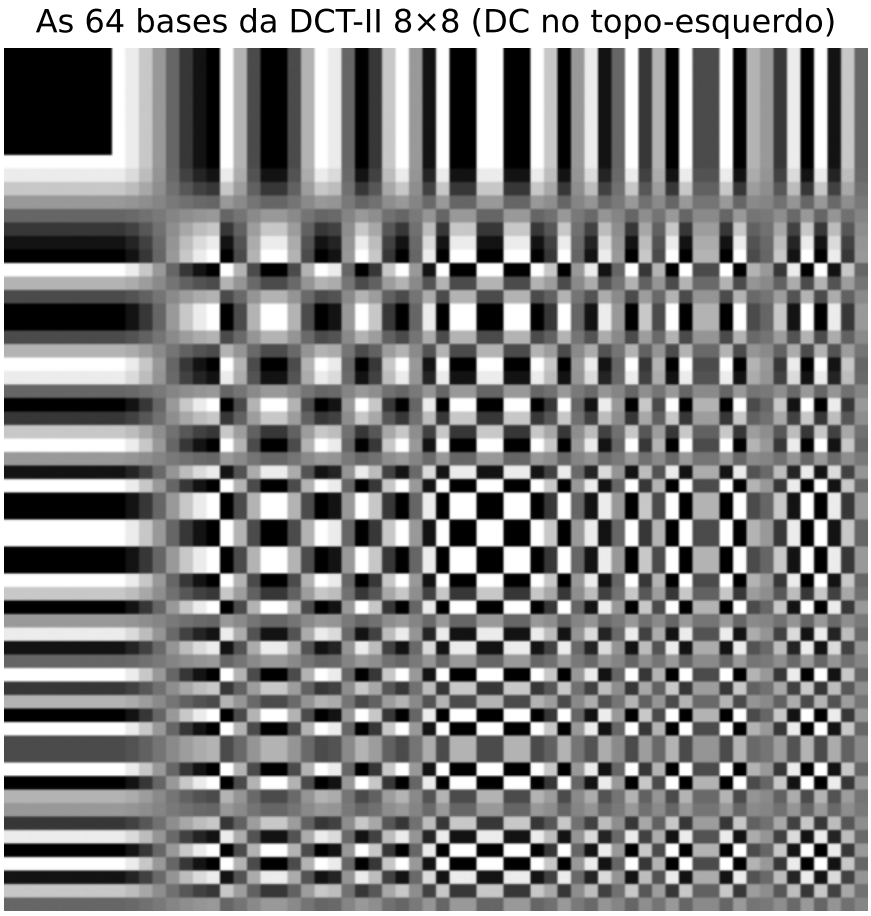


Figura 5.23: O Alfabeto Visual do JPEG: As 64 funções de base da DCT-II. O coeficiente DC fica no topo esquerdo (suave). Ao descer e avançar à direita, a oscilação espacial aumenta drasticamente.

5.7.4 Concentrazione di Energia e Ricostruzione Progressiva

Prima dell'applicazione della DCT, i pixel del blocco di intensità vengono di routine traslati (sottraendo 128 per immagini a 8 bit) al fine di centrare il segnale attorno allo zero, eliminando componenti continue superflue. Calcolando la DCT sul blocco risultante, la proprietà di **compattazione dell'energia** diventa evidente: la quasi totalità della varianza e dell'informazione dell'immagine originale si concentra nel coefficiente DC

($C(0,0)$) e nei primi armonici AC a bassa frequenza.

La Figura 5.24 dimostra questo fenomeno mediante una ricostruzione progressiva per troncamento brusco. Invece di utilizzare tutti i 64 coefficienti, l'algoritmo conserva solo i primi k componenti — selezionati in base a una scansione che privilegia le basse frequenze spaziali — e azzera i rimanenti.

La sintesi inversa (**IDCT**) eseguita con solo una frazione dei coefficienti (come il 15% o il 30%) è già in grado di recuperare le strutture e l'illuminazione macro del blocco originale di pixel. Man mano che gli armonici a frequenze più elevate vengono progressivamente reintegrati, i dettagli fini e le transizioni rapide vengono ripristinati. Questo comportamento valida il principio della compressione percettiva: le alte frequenze scartate possiedono poca energia e la loro assenza, in condizioni normali, genera un impatto visivo secondario sulla percezione dell'osservatore.

```
%%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 em bloco 8×8: coefficienti e ricostruzione progressiva."
//| echo: true
//| output: true

#include <opencv2/opencv.hpp>
#include <cmath>
#include <cstdio>
#include <vector>
#include <string>
#include <algorithm>
#include "morph.hpp"

static cv::Mat dct2_block(const cv::Mat& bloco) {
    // DCT-II 2D ortogonale (separabile).
    cv::Mat temp, result;
    cv::dct(bloco, temp);
    return temp.clone();
}

static cv::Mat idct2_block(const cv::Mat& coefs) {
    // IDCT-II 2D ortogonale.
    cv::Mat result;
    cv::idct(coefs, result);
    return result;
}

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]

    //   Blocco 8×8 centrato sull'immagine
    int cy = img_gray.h / 2;
    int cx = img_gray.w / 2;

    cv::Mat img_mat(img_gray.h, img_gray.w, CV_8UC1, img_gray.data.data());
    cv::Mat bloco_f = img_mat(cv::Rect(cx, cy, 8, 8)).clone();
    bloco_f.convertTo(bloco_f, CV_64F);
    bloco_f -= 128.0;

    // DCT 2D del blocco
    cv::Mat C = dct2_block(bloco_f);

    printf("Coefficienti DCT del blocco 8×8:\n");
    // Stampare i coefficienti arrotondati
```

```
    for (int i = 0; i < 8; i++) {
        for (int j = 0; j < 8; j++) {
            printf("%4d ", (int)std::round(C.at<double>(i, j)));
        }
        printf("\n");
    }

    double dc_energy = C.at<double>(0, 0) * C.at<double>(0, 0);
    double total_energy = 0.0;
    for (int i = 0; i < 8; i++)
        for (int j = 0; j < 8; j++)
            total_energy += C.at<double>(i, j) * C.at<double>(i, j);

    printf("\nEnergia DC     : %.1f\n", dc_energy);
    printf("Energia totale : %.1f\n", total_energy);
    printf("Frazione nel DC: %.1f%% ← concentrazione di energia\n",
           (dc_energy / total_energy) * 100.0);

    //   Ricostruzione progressiva
    std::vector<mm::Image> imgs_rec;
    std::vector<std::string> titles_rec;

    cv::Mat bloco_orig_f;
    bloco_f.convertTo(bloco_orig_f, CV_8U);
    bloco_orig_f += 128;
    cv::Mat bloco_show;
    cv::normalize(bloco_orig_f, bloco_show, 0, 255, cv::NORM_MINMAX, CV_8U);
    imgs_rec.push_back(mm::Image(bloco_show));
    titles_rec.push_back("Blocco originale\n(8×8 pixel)");

    std::vector<int> keeps = {1, 4, 10, 20, 40, 64};
    for (int keep : keeps) {
        cv::Mat C_trunc = cv::Mat::zeros(8, 8, CV_64F);

        // Costruire l'ordine zig-zag semplicemente usando la somma 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);
                  });

        for (int i = 0; i < keep && i < (int)indices.size(); i++) {
            int u = indices[i].first;
            int v = indices[i].second;
            C_trunc.at<double>(u, v) = C.at<double>(u, v);
        }

        cv::Mat rec = idct2_block(C_trunc);
        rec += 128.0;

        cv::Mat rec_8u;
        rec.convertTo(rec_8u, CV_8U);

        imgs_rec.push_back(mm::Image(rec_8u));
        char buf[64];
        snprintf(buf, sizeof(buf), "%d coef.\n(%.0f%% del totale)", keep,
                 (keep / 64.0) * 100.0);
```

```
        titles_rec.push_back(buf);
    }

    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"
```

```
Coefficienti DCT del blocco 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    1    0    0    0    0
   9  -11    0    0    1    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

Energia DC     : 36816.0
Energia totale : 52253.0
Frazione nel DC: 70.5% ← concentrazione di energia
[1] Blocco originale
(8×8 pixel)
[2] 1 coef.
(2% del totale)
[3] 4 coef.
(6% del totale)
[4] 10 coef.
(16% del totale)
[5] 20 coef.
(31% del totale)
[6] 40 coef.
(62% del totale)
[7] 64 coef.
(100% del totale)
```

```
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)")
```

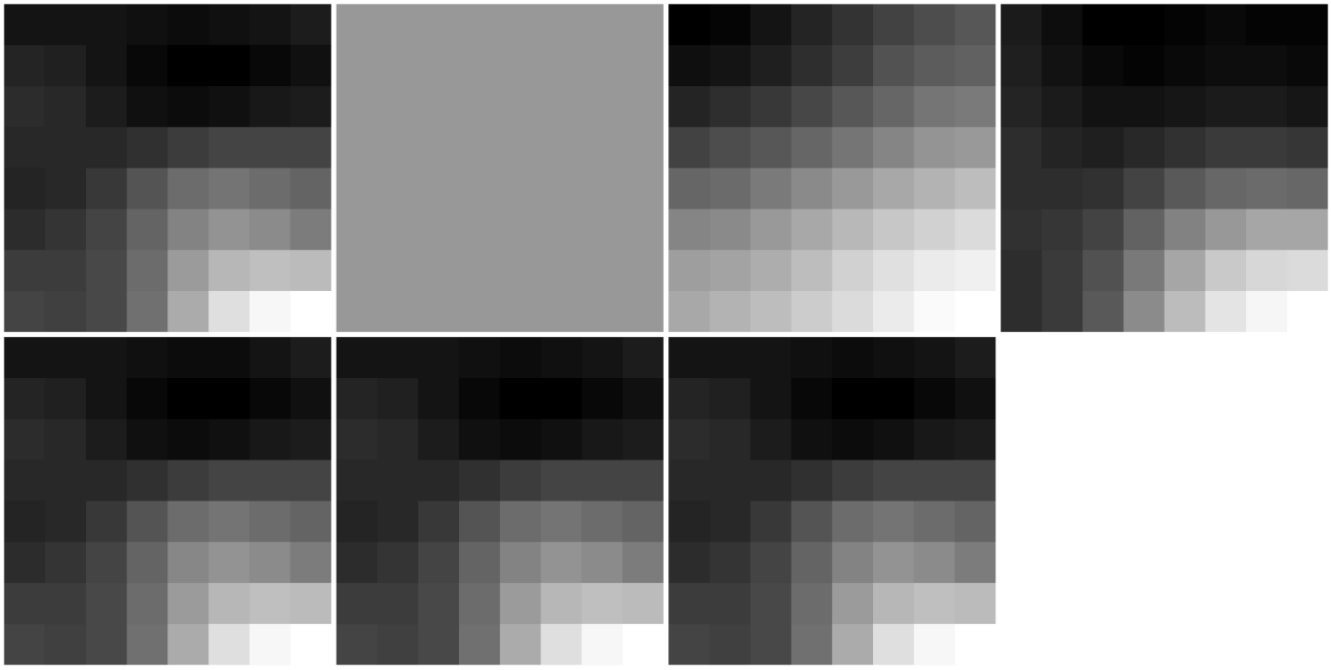


Figura 5.24: DCT 2D em bloco 8×8: coeficientes e reconstrução progressiva.

5.7.5 Il *Pipeline* di Compressione JPEG

Lo standard JPEG opera dividendo l'immagine in blocchi disgiunti di 8×8 pixel, elaborati tramite una sequenza di trasformazioni spaziali, percettive e statistiche. Il *pipeline* completo di codifica è strutturato in sei fasi principali:

$$\text{RGB} \xrightarrow{\text{(1) } YC_bC_r} \xrightarrow{\text{(2) Sottocampionamento}} \xrightarrow{\text{(3) Blocchi 8×8}} \xrightarrow{\text{(4) DCT}} \xrightarrow{\text{(5) Quantizzazione}} \xrightarrow{\text{(6) Codifica Entropica}}$$

La Tabella 5.7 dettaglia la funzione analitica e il fondamento percettivo che giustifica ciascuna di queste fasi.

Tabella 5.7: Fasi del *pipeline* di compressione JPEG e i rispettivi fondamenti di progetto.

Fase	Operazione	Fondamento Percettivo e Statistico
1	Conversione $RGB \to YC_bC_r$	Separa la luminanza (Y) dalla crominanza (C_b, C_r). Il sistema visivo umano (SVH) presenta maggiore sensibilità alle variazioni di luminosità che di colore.
2	Sottocampionamento della crominanza (es: 4:2:0)	Riduce la risoluzione spaziale dei canali colore della metà, scartando dati ridondanti con impatto visivo trascurabile.
3–4	Centratura e applicazione della DCT 8×8	Trasla i pixel nell'intervallo $[-128, 127]$ e compatta l'energia spettrale del blocco nei coefficienti a bassa frequenza.

Fase	Operazione	Fondamento Percettivo e Statistico
5	Quantizzazione lineare selettiva	Divide ciascun coefficiente $C(u,v)$ per l'elemento corrispondente della matrice $Q(u,v)$, applicando un arrotondamento a interi. Costituisce la principale fonte di compressione con perdita.
6	Scansione a zig-zag e codifica	Ordina i coefficienti quantizzati per massimizzare le sequenze nulle consecutive, ottimizzando la codifica a lunghezza di corsa (RLE) e la codifica di Huffman.

La **matrice di quantizzazione** $Q(u,v)$ è il meccanismo centrale di controllo del compromesso tra tasso di compressione e qualità visiva. Nell'algoritmo pratico della Figura 5.25, il fattore di qualità stabilito dall'utente (scala da 1 a 100) viene convertito in uno scalare che parametrizza la severità della matrice Q. Valori ridotti di qualità espandono i divisori di $Q(u,v)$, forzando il troncamento di massa dei coefficienti AC a zero. Quando questa eliminazione è eccessiva, la discontinuità ai confini dei blocchi adiacenti non viene attenuata nella ricostruzione, generando i cosiddetti **artefatti a blocchi** (*blocking artifacts*).

La Logica della Scansione a Zig-Zag

L'efficienza del codificatore entropico successivo alla quantizzazione dipende direttamente dall'ordinamento dei dati. Poiché la DCT concentra l'energia vitale nel vertice superiore sinistro della matrice (basse frequenze) e spinge i coefficienti nulli verso le estremità opposte, la lettura lineare per righe o colonne frammenterebbe le sequenze di zeri.

L'ordinamento a zig-zag risolve questa limitazione percorrendo la matrice diagonalmente in ordine crescente di frequenza spaziale. Questa mappatura raggruppa i coefficienti significativi all'inizio del vettore e concentra i coefficienti nulli in un'unica sequenza continua alla fine dell'arrangiamento, consentendo all'algoritmo RLE di codificare grandi blocchi di dati in modo compatto ed efficiente.

> **i** Cos'è l'RLE?
>
> **RLE** (*Run-Length Encoding*) è una tecnica di compressione senza perdita che codifica sequenze consecutive di valori identici — specialmente **zeri** — come una coppia (conteggio, valore). Nel JPEG, dopo la scansione a zig-zag, i coefficienti quantizzati sono organizzati in modo che gli zeri si concentrino alla fine del vettore. L'RLE comprime quindi questa lunga corsa di zeri con estrema efficienza, ottimizzando l'archiviazione e la trasmissione dell'immagine compressa.

```
%%writefile tmp/fig_05_jpeg_pipeline.cpp
#define MM_OUT "tmp/fig_05_jpeg_pipeline.png"
//| label: fig-05-jpeg-pipeline
//| fig-cap: "*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 $Q=25$)."
//| echo: true
//| output: true

#include <opencv2/opencv.hpp>
#include <vector>
```

```cpp
#include <string>
#include <cstdio>
#include "morph.hpp"


int main() {
    // Pipeline JPEG (DCT 8x8 -> quantizacao -> IDCT) via mm.jpegCompress,
    // na imagem classica do Cameraman (asset do capitulo) reduzida a 256x256.
    cv::Mat img_src_full = cv::imread("imagens/cameraman.png", cv::IMREAD_GRAYSCALE);
    cv::Mat img_src_resized;
    cv::resize(img_src_full, img_src_resized, cv::Size(256, 256));
    mm::Image img_src = mm::gray(img_src_resized);

    std::vector<mm::Image> imgs;
    std::vector<std::string> titles;
    imgs.push_back(img_src);
    titles.push_back("Original (Cameraman)");

    int qs[] = {10, 25, 50, 75, 90};
    for (int q : qs) {
        mm::Image rec = mm::jpegCompress(img_src, q);
        imgs.push_back(rec);
        char title[100];
        double psnr_val = mm::psnr(img_src, rec);
        std::snprintf(title, sizeof(title), "Q=%d (PSNR=%.1f dB)", q, psnr_val);
        titles.push_back(title);
    }

    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=30.7 dB)
[4] Q=50 (PSNR=32.8 dB)
[5] Q=75 (PSNR=35.2 dB)
[6] Q=90 (PSNR=40.0 dB)
```

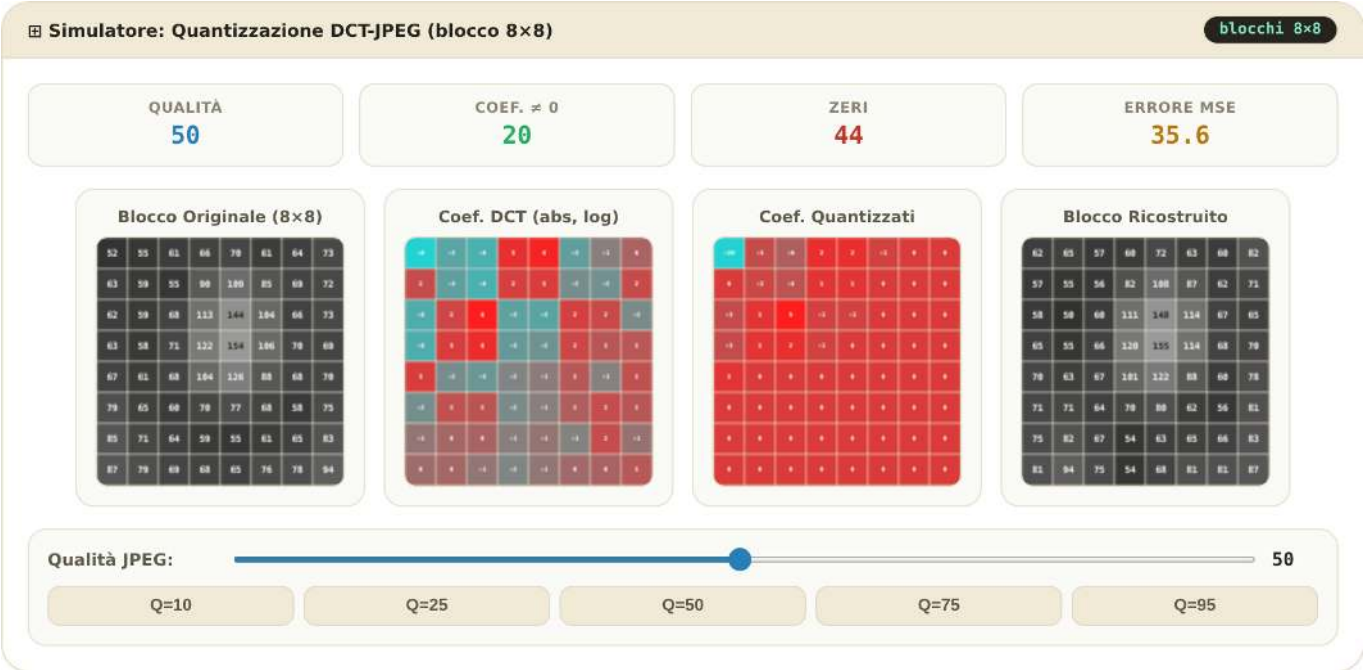
```
try:
    mm.show(mm.read("tmp/fig_05_jpeg_pipeline.png"), figsize=(14, 10))
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + "
    tmp/fig_05_jpeg_pipeline.png (ver a versao Python)")
```

Figura 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 Simulatore Interattivo: Quantizzazione DCT

Il simulatore della Figura 5.26 consente di esplorare l'impatto del processo di quantizzazione su un blocco 8×8 estratto da un'immagine reale, sintetizzando in tempo reale le seguenti componenti:

- **Blocco originale e ricostruito:** Rappresentazione diretta dei pixel nel dominio spaziale in scala di grigi [0, 255].
- **Coefficienti DCT:** Distribuzione dell'energia mappata in modo logaritmico su un gradiente cromatico, evidenziando la concentrazione di intensità nel vertice superiore sinistro (basse frequenze).
- **Coefficienti quantizzati:** Visualizzazione dei valori interi risultanti dalla divisione per la matrice $Q(u, v)$, rendendo visivamente esplicita l'emergenza in massa di coefficienti nulli (in toni scuri) al diminuire del fattore di qualità.
- **Metriche di compressione:** Pannello di monitoraggio che quantifica l'Errore Quadratico Medio (MSE), il numero di coefficienti preservati e il volume di zeri generati per la codifica entropica.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.it/cap05/sim-05-dct.html

Figura 5.26: Simulatore interattivo di compressione DCT-JPEG: regola il fattore di qualità e visualizza in tempo reale i coefficienti azzerati, il blocco ricostruito e l'errore di quantizzazione.

5.8 Confronto dei Formati Immagine

La scelta di un formato di memorizzazione digitale incide direttamente sul compromesso tra qualità visiva, dimensione del file e costo computazionale della decodifica. I tre formati di maggiore rilevanza per architetture *web* e sistemi di calcolo visivo sono JPEG, PNG e WebP.

5.8.1 Caratteristiche dei Formati

La Tabella 5.8 sintetizza le proprietà strutturali dei principali formati di immagine rasterizzati.

Tabella 5.8: Confronto strutturale tra i principali formati di immagine rasterizzati.

Caratteristica	JPEG	PNG	WebP
Compressione	Con perdita	Senza perdita	Con e senza perdita.
Trasparenza (canale alfa)	No	Sì	Sì.
Supporto animazioni	No	Limitato (APNG)	Sì.
Algoritmo di base	DCT + Huffman	DEFLATE (LZ77 + Huffman)	VP8 / VP8L.
Ideale per	Fotografia	Grafica, testo e icone	Uso universale in ambiente Web.
Inadeguato per	Testo e bordi netti	Immagini fotografiche complesse	Compatibilità legacy.

5.8.2 Metriche di Valutazione della Qualità

Due metriche oggettive sono ampiamente adottate per quantificare la distorsione introdotta dai processi di compressione:

Picco del Rapporto Segnale-Rumore (PSNR, *Peak Signal-to-Noise Ratio*):

$$\text{PSNR} = 10 \log_{10}\left(\frac{L^2}{\text{MSE}}\right) \quad [\text{dB}] \tag{5.10}$$

dove $L = 255$ per immagini quantizzate a 8 bit e MSE rappresenta l'**Errore Quadratico Medio** (*Mean Squared Error*). Valori di PSNR superiori a 40 dB indicano fedeltà eccellente; tra 30 dB e 40 dB rappresentano buona qualità; valori inferiori a 30 dB corrispondono a degradazioni visive facilmente percepibili.

Indice di Similarità Strutturale (SSIM, *Structural Similarity Index*):

$$\text{SSIM}(f, g) = \frac{(2\mu_f\mu_g + c_1)(2\sigma_{fg} + c_2)}{(\mu_f^2 + \mu_g^2 + c_1)(\sigma_f^2 + \sigma_g^2 + c_2)} \tag{5.11}$$

Lo SSIM valuta finestre locali dell'immagine basandosi su tre componenti complementari: **luminanza** (μ_f, μ_g), **contrasto** (σ_f, σ_g) e **struttura** (σ_{fg}), ponderate da costanti di stabilità c_1 e c_2. L'indice varia nell'intervallo $[-1, 1]$, dove l'unità rappresenta l'identità perfetta. A differenza del PSNR, lo SSIM considera l'organizzazione spaziale degli errori, allineandosi alla percezione del sistema visivo umano (SVH).

> **i** PSNR vs SSIM: Applicazione di Metriche Percettive
>
> Il PSNR possiede una formulazione matematica semplice e un basso costo computazionale; tuttavia tende a sovrastimare la qualità in immagini con distorsioni localizzate o a sottostimarla in variazioni globali di luminosità tollerate dall'osservatore. Lo SSIM modella con maggiore fedeltà la percezione biologica, ma richiede un maggiore sforzo di elaborazione. Per analisi rigorose dei codec, si raccomanda di riportare entrambe le metriche statistiche in carattere complementare.

5.8.3 Ispezione Visiva: Natura degli Artefatti di Compressione

La natura matematica del codificatore determina il tipo di degrado introdotto a bitrate ridotti. Come illustrato nella Figura 5.27, la compressione aggressiva tramite DCT nello standard JPEG segmenta l'immagine in griglie rigide, generando gli **artefatti a blocchi** (*blocking artifacts*). Al contrario, gli algoritmi basati su codifica predittiva o rappresentazioni sottoposte a trasformate spaziali avanzate (come WebP e JPEG 2000) eliminano le discontinuità di blocco, ma introducono perdita di texture fine e sfocature caratteristiche attorno ai bordi ad alto contrasto.

```
%%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"

// Função de zoom com interpolação NEAREST
cv::Mat zoom(const cv::Mat& img) {
    cv::Mat roi = img(cv::Rect(150, 120, 80, 80));  // img[120:200, 150:230]
    cv::Mat resized;
    cv::resize(roi, resized, cv::Size(320, 320), 0, 0, cv::INTER_NEAREST);
```

```
    return resized;
}

int main() {
    // Cria diretório temporário se não existir
    std::filesystem::create_directories("tmp");

    // Lê e redimensiona a imagem original
    cv::Mat img_src_full = cv::imread("imagens/cameraman.png", cv::IMREAD_GRAYSCALE);
    cv::Mat img_src_resized;
    cv::resize(img_src_full, img_src_resized, cv::Size(256, 256));
    mm::Image img_src = img_src_resized;  // Converte para mm::Image

    // Salva versões comprimidas
    cv::Mat img_src_mat = img_src;  // Converte de volta para cv::Mat
    cv::imwrite("tmp/zoom_q10.jpg", img_src_mat, {cv::IMWRITE_JPEG_QUALITY, 10});
    cv::imwrite("tmp/zoom_q10.webp", img_src_mat, {cv::IMWRITE_WEBP_QUALITY, 10});

    // Aplica zoom em cada versão
    cv::Mat z_orig = zoom(img_src_mat);
    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));

    // Mostra os resultados
    mm::show(
        {z_orig, z_jpeg, z_webp},
        MM_OUT,
        {"Zoom Original", "JPEG Q=10 (Artefato de Bloco)", "WebP Q=10 (Suavizacao)"},
        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)")
```

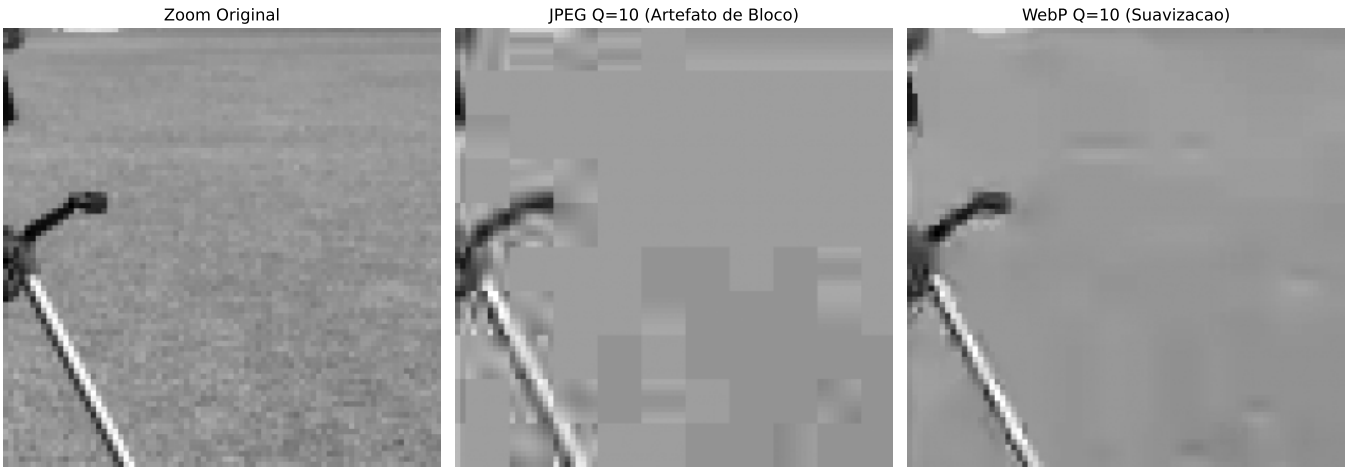


Figura 5.27: Análise comparativa de artefatos de compressão sob fator de qualidade reduzido ($Q = 10$). À esquerda, observa-se o artefato de bloco característico da discretização por DCT no JPEG. À direita, evidencia-se o efeito de atenuação e suavização de bordas intrínseco ao padrão WebP.

5.8.4 Valutazione Quantitativa e Spaziale della Compressione

La validazione degli algoritmi di compressione con perdita richiede un'analisi che correli il costo di archiviazione alla fedeltà del segnale ricostruito. Tale valutazione viene condotta in modo complementare attraverso curve di prestazione globale e mediante la mappatura locale delle distorsioni indotte dai codificatori.

5.8.4.1 Curve di Rateo-Distorsione

La Figura 5.28 presenta la valutazione empirica del *pipeline* JPEG e WebP tramite **curve di rateo-distorsione**, che monitorano il guadagno di compressione (dimensione del file in KB) in funzione del PSNR. Il formato PNG funge da linea di base ideale (PSNR $= \infty$), poiché la sua natura *lossless* impedisce qualsiasi degradazione, sebbene richieda un volume di dati notevolmente maggiore.

L'analisi delle curve dimostra la superiorità e l'efficienza dello standard WebP rispetto al JPEG tradizionale: per raggiungere lo stesso livello di fedeltà matematica (come la fascia di qualità eccellente, dove PSNR $>$ 40 dB), il codificatore WebP genera file significativamente più piccoli. Questo comportamento riflette

l'impatto pratico dell'evoluzione degli algoritmi nell'ottimizzazione dei sistemi di trasmissione e archiviazione digitale.

```
%%writefile tmp/fig_05_formatos_comparacao.cpp
#define MM_OUT "tmp/fig_05_formatos_comparacao.png"
// Compilar: g++ -std=c++17 -O2 snippet.cpp -o snippet $(pkg-config --cflags --libs opencv4)
// Executar: ./snippet

#include <opencv2/opencv.hpp>
#include <filesystem>
#include <vector>
#include <string>
#include <iostream>
#include "morph.hpp"

int main() {
    //| 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

    std::filesystem::create_directories("tmp");

    cv::Mat resized;
    cv::resize(cv::imread("imagens/cameraman.png", cv::IMREAD_GRAYSCALE), resized,
cv::Size(256, 256));
    mm::Image src = mm::gray(mm::Image(resized));

    std::vector<double> jpeg_kb, jpeg_psnr;
    std::vector<int> jpeg_q = {10, 20, 30, 40, 50, 60, 70, 80, 90, 95};
    for (int q : jpeg_q) {
        std::string p = "tmp/fmt_q" + std::to_string(q) + ".jpg";
        std::vector<int> params = {cv::IMWRITE_JPEG_QUALITY, q};
        cv::imwrite(p, cv::Mat(src), params);
        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)));
    }

    std::vector<double> webp_kb, webp_psnr;
    std::vector<int> webp_q = {30, 50, 70, 85, 95};
    for (int q : webp_q) {
        std::string p = "tmp/fmt_w" + std::to_string(q) + ".webp";
        std::vector<int> params = {cv::IMWRITE_WEBP_QUALITY, q};
        cv::imwrite(p, cv::Mat(src), params);
        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)));
    }

    std::string p_png = "tmp/fmt.png";
    std::vector<int> png_params = {cv::IMWRITE_PNG_COMPRESSION, 9};
    cv::imwrite(p_png, cv::Mat(src), png_params);
    double png_kb = (double)std::filesystem::file_size(p_png) / 1024.0;

    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_buf[256];
    std::snprintf(title_buf, sizeof(title_buf),
```

```
                "Curva Taxa-Distorcao (PNG sem perda: %.1f KB)", png_kb);
    std::string title = title_buf;

    mm::Image chart = mm::lineChart(
        xs, ys,
        labels,
        {},
        title,
        "Tamanho do arquivo (KB)", "PSNR (dB)"
    );
    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)")
```

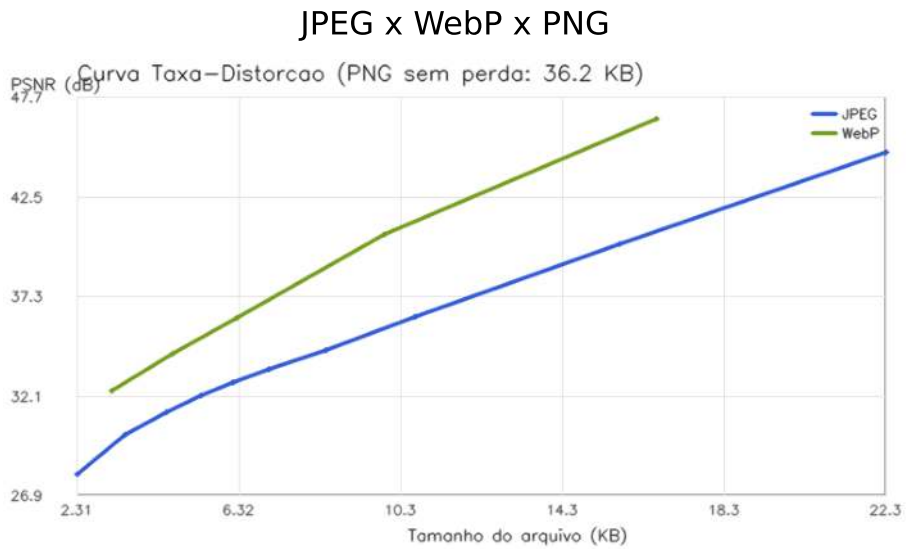


Figura 5.28: Curva taxa-distorção: PSNR vs tamanho de arquivo para JPEG, WebP e PNG aplicada à imagem do *Cameraman*.

i Dimensione originale dell'immagine

L'immagine *Cameraman* (256 × 256 pixel in scala di grigi) occupa **64 KB** in formato grezzo (senza compressione). Come riferimento, il PNG *lossless* comprime questo volume a **36,2 KB** — evidenziando che la compressione senza perdita riduce già significativamente lo spazio di archiviazione per immagini con regioni omogenee. In contrapposizione, i formati con perdita (JPEG e WebP) raggiungono dimensioni ancora minori: il JPEG con qualità 95 occupa 22,3 KB (PSNR 45 dB), mentre il WebP con qualità 90 raggiunge 12,5 KB con PSNR equivalente, dimostrando la sua superiorità in efficienza di compressione.

5.8.4.2 Mappatura Spaziale degli Errori e Correlazione Percettiva

Sebbene il PSNR offra un indicatore numerico rapido, le metriche globali non riescono a distinguere come la perdita di informazione si distribuisca geometricamente sull'immagine. La Figura 5.29 risolve questa limitazione associando le ricostruzioni a diverse qualità ai rispettivi mappe di errore assoluto e all'SSIM.

Le mappe residue — ottenute dalla differenza assoluta normalizzata tra l'immagine originale e quella compressa — rivelano la firma spaziale intrinseca di ciascuna architettura di codifica:

- **Ad alte qualità ($Q = 95$ a $Q = 75$):** Le distorsioni si concentrano prevalentemente attorno a transizioni brusche di intensità (bordi), a causa del ripiegamento spettrale derivante dallo scarto delle alte frequenze. L'indice SSIM rimane prossimo all'unità, attestando l'integrità delle strutture originali.
- **A qualità aggressive ($Q = 50$ a $Q = 25$):** L'errore assume una struttura a maglia ortogonale regolarizzata. Questo pattern geometrico evidenzia l'emergere degli **artefatti a blocchi** (*blocking artifacts*), indicando che la quantizzazione severa ha corrotto la correlazione spaziale tra blocchi adiacenti di 8×8 pixel.

L'SSIM cattura questa degradazione morfologica in modo molto più sensibile rispetto al PSNR, penalizzando il punteggio finale man mano che l'organizzazione strutturale e le trame fini — alle quali il sistema visivo umano è altamente reattivo — vengono eliminate dal codificatore.

```
%%writefile tmp/fig_05_ssim_artefatos.cpp
#define MM_OUT "tmp/fig_05_ssim_artefatos.png"
//| label: fig-05-ssim-artefatos
//| fig-cap: "Análise espacial de degradação: imagens reconstruídas e respectivos mapas de
erro absoluto normalizados para diferentes fatores de qualidade JPEG."
```

```
//| echo: true
//| output: true

#include <opencv2/opencv.hpp>
#include <vector>
#include <string>
#include <cmath>
#include <cstdio>
#include "morph.hpp"

// Implementazione SSIM (Wang et al.) con finestra gaussiana
static double compute_ssim(const cv::Mat& img1, const cv::Mat& img2) {
    const double C1 = 6.5025, C2 = 58.5225;
    cv::Mat I1, I2;
    img1.convertTo(I1, CV_64F);
    img2.convertTo(I2, CV_64F);

    cv::Mat kernel = cv::getGaussianKernel(11, 1.5);
    cv::Mat window = kernel * kernel.t();

    cv::Mat mu1, mu2, mu1_sq, mu2_sq, mu1_mu2;
    cv::filter2D(I1, mu1, -1, window);
    cv::filter2D(I2, mu2, -1, window);
    cv::pow(mu1, 2, mu1_sq);
    cv::pow(mu2, 2, mu2_sq);
    mu1_mu2 = mu1.mul(mu2);

    cv::Mat sigma1_sq, sigma2_sq, sigma12;
    cv::filter2D(I1.mul(I1), sigma1_sq, -1, window);
    sigma1_sq -= mu1_sq;
    cv::filter2D(I2.mul(I2), sigma2_sq, -1, window);
    sigma2_sq -= mu2_sq;
    cv::filter2D(I1.mul(I2), sigma12, -1, window);
    sigma12 -= mu1_mu2;

    cv::Mat ssim_map;
    cv::Mat cs_map;
    cv::Mat t1 = 2 * mu1_mu2 + C1;
    cv::Mat t2 = 2 * sigma12 + C2;
    cv::Mat t3 = t1.mul(t2);
    cv::divide(t3, (mu1_sq + mu2_sq + C1).mul(sigma1_sq + sigma2_sq + C2), ssim_map);

    cv::Scalar mssim = cv::mean(ssim_map);
    return mssim[0];
}

int main() {
    // Cameraman (asset del capitolo), 256x256 - stessa immagine del percorso py.
    cv::Mat img_tmp = cv::imread("imagens/cameraman.png", cv::IMREAD_GRAYSCALE);
    cv::Mat img_resized;
    cv::resize(img_tmp, img_resized, cv::Size(256, 256));
    mm::Image img_gray = mm::gray(mm::Image(img_resized));

    std::vector<mm::Image> imgs_ssim;
    std::vector<std::string> titles_ssim;
    imgs_ssim.push_back(img_gray);
    titles_ssim.push_back("Originale");

    int qs[] = {25, 50, 75, 95};
    for (int i = 0; i < 4; i++) {
        int q = qs[i];
```

```
        mm::Image rec = mm::jpegCompress(img_gray, q);  // DCT 8x8 -> quant -> IDCT

        cv::Mat img_gray_cv = img_gray;
        cv::Mat rec_cv = rec;

        double psnr_v = mm::psnr(img_gray, rec);

        // Calcola SSIM
        double ssim_v = compute_ssim(img_gray_cv, rec_cv);

        // Mappa di errore normalizzata
        cv::Mat img_float, rec_float, diff;
        img_gray_cv.convertTo(img_float, CV_64F);
        rec_cv.convertTo(rec_float, CV_64F);
        diff = cv::abs(img_float - rec_float);

        cv::Mat diff_vis_m;
        cv::normalize(diff, diff_vis_m, 0, 255, cv::NORM_MINMAX, CV_8U);
        mm::Image diff_vis(diff_vis_m);

        imgs_ssim.push_back(rec);
        imgs_ssim.push_back(diff_vis);

        char title_buf[256];
        std::snprintf(title_buf, sizeof(title_buf), "Q=%d (PSNR=%.1fdB | SSIM=%.3f)", q,
psnr_v, ssim_v);
        titles_ssim.push_back(std::string(title_buf));

        char err_buf[256];
        std::snprintf(err_buf, sizeof(err_buf), "Mappa di errore (Q=%d) - bordi e blocco", q);
        titles_ssim.push_back(std::string(err_buf));
    }

    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] Originale
[2] Q=25 (PSNR=30.7dB | SSIM=0.860)
[3] Mappa di errore (Q=25) - bordi e blocco
[4] Q=50 (PSNR=32.8dB | SSIM=0.905)
```

```
[5] Mappa di errore (Q=50) - bordi e blocco
[6] Q=75 (PSNR=35.2dB | SSIM=0.938)
[7] Mappa di errore (Q=75) - bordi e blocco
[8] Q=95 (PSNR=44.8dB | SSIM=0.990)
[9] Mappa di errore (Q=95) - bordi e blocco
```

```
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)")
```

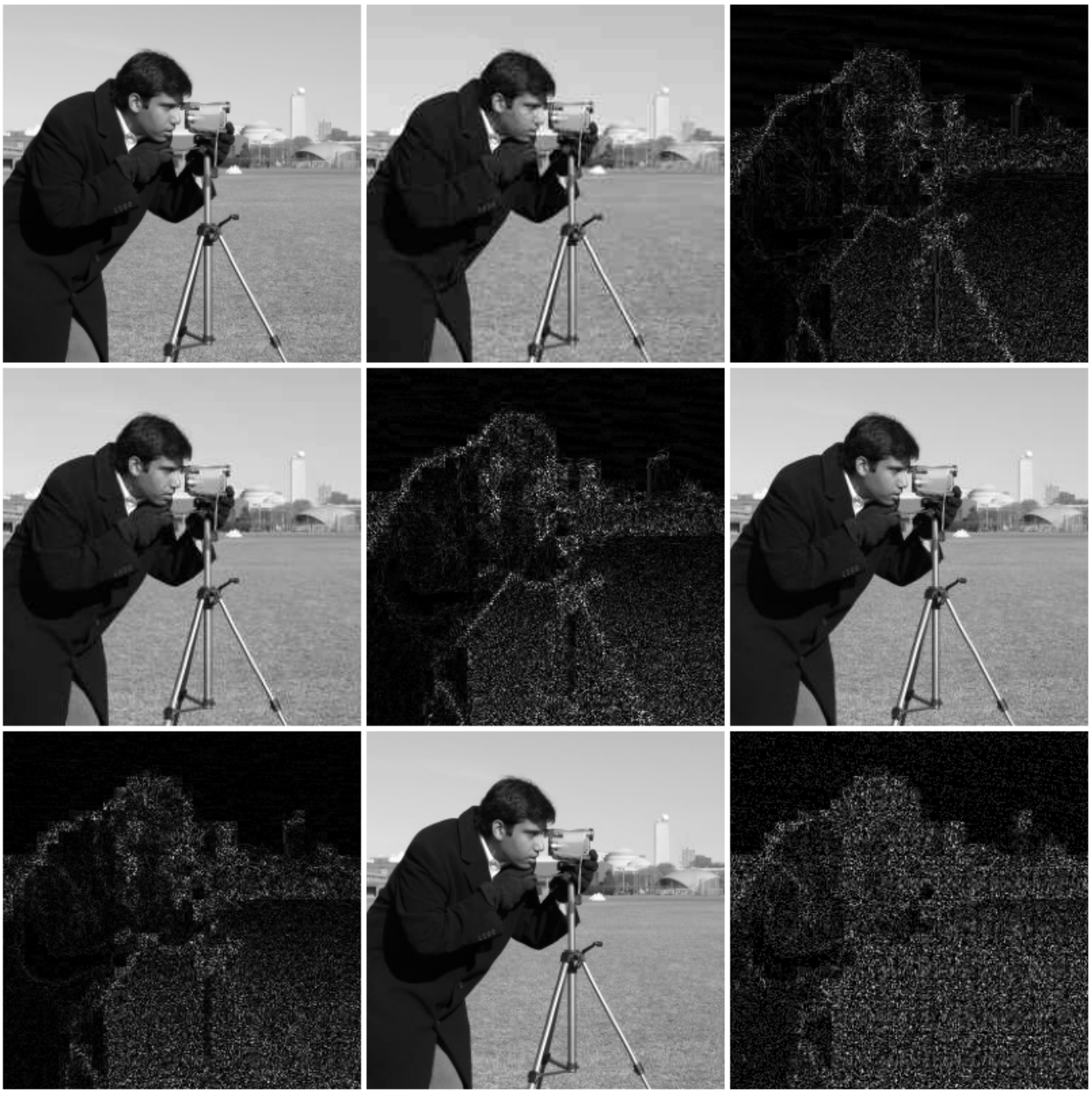


Figura 5.29: Análise espacial de degradação: imagens reconstruídas e respectivos mapas de erro absoluto normalizados para diferentes fatores de qualidade JPEG.

i Interpretando le mappe di errore

Le mappe di errore presentate sono state **normalizzate singolarmente** (`cv2.NORM_MINMAX`) per massimizzare il contrasto visivo e rivelare la struttura spaziale delle distorsioni. Ciò significa che:

- In **Q=95**, l'errore assoluto è dell'ordine di **0.5–1.5 livelli di grigio** (impercettibile visivamente), ma la normalizzazione lo amplifica in bianco e nero per evidenziarne la localizzazione su bordi e transizioni.
- In **Q=25**, l'errore assoluto è **10–20 volte maggiore** (5–15 livelli di grigio), ma la normalizzazione lo porta anch'esso allo stesso intervallo [0, 255].

Pertanto, **l'intensità del bianco nelle mappe NON è confrontabile tra diverse qualità** — le mappe servono solo a rivelare la **firma spaziale** dell'errore (bordi vs blocchi), non la sua ampiezza. L'ampiezza corretta è fornita dai valori di PSNR e SSIM, che mostrano chiaramente che Q=95 ha un errore molto minore rispetto a Q=25.

Sintesi — Compressione JPEG

Il processo di compressione nello standard JPEG si basa sull'applicazione combinata di trasformazioni spaziali, percettive e statistiche per ridurre le ridondanze di un'immagine. La Tabella 5.9 riassume il ruolo di ciascuna fase nel *pipeline* e il rispettivo impatto sulla riduzione dei dati.

Tabella 5.9: Sintesi delle fasi del *pipeline* di compressione JPEG e dei rispettivi impatti.

Fase	Operazione Analitica	Meccanismo di Guadagno / Compressione
Conversione YC_bC_r	Isolamento dei canali di luminanza e crominanza.	Modella la percezione del SVH, consentendo di trattare colore e luminosità in modo indipendente.
Sottocampionamento 4:2:0	Riduzione della risoluzione spaziale dei canali di colore (C_b e C_r).	Elimina circa il 50% dei dati grezzi con un impatto visivo minimo.
DCT 8×8	Mappatura dal dominio spaziale al dominio delle frequenze spaziali.	Compattazione dell'energia, concentrando l'informazione vitale nei primi coefficienti.
Quantizzazione Lineare	Divisione intera dei coefficienti per una matrice di ponderazione $Q(u, v)$.	Principale fonte di compressione con perdita; elimina le alte frequenze impercettibili.
Codifica Entropica	Applicazione di algoritmi RLE e codifica di Huffman.	Compressione statistica senza perdita, ottimizzata dalle lunghe sequenze di coefficienti nulli.

Artefatti di Degradazione Caratteristici

L'applicazione di tassi di compressione eccessivamente aggressivi (fattori di qualità ridotti) introduce distorsioni prevedibili nell'immagine ricostruita, derivanti dalle limitazioni matematiche del modello:

- **Artefatti a blocchi (*blocking artifacts*):** Discontinuità geometriche visibili ai confini dei blocchi di 8×8 pixel, causate dalla perdita di correlazione spaziale dopo la quantizzazione severa delle componenti AC.
- **Effetto di alone (*ringing*):** Oscillazioni fantasma o distorsioni "a fumo" attorno ai bordi netti e ad alto contrasto, provocate dall'eliminazione brusca delle armoniche ad alta frequenza necessarie per ricostruire funzioni a gradino.
- **Perdita di texture fine:** Attenuazione dei dettagli ad alta frequenza e a basso contrasto (come prati, tessuti o porosità), rendendo le regioni originariamente strutturate eccessivamente lisce o omogenee.

5.9 Applicazione Pratica: Rimozione del Rumore mediante Filtraggio Ibrido

Integrando le tecniche consolidate nel corso di questo capitolo, si presenta un *pipeline* completo di **restauro delle immagini** che combina l'analisi spettrale nel dominio della frequenza con il filtraggio adattivo nel dominio spaziale. L'obiettivo è attenuare un rumore misto (composto da degradazione gaussiana e interferenza periodica) preservando al massimo i dettagli strutturali dell'immagine originale.

$$\text{Immagine Rumorosa} \xrightarrow{\text{FFT2}} \xrightarrow{\text{Filtro Notch Gaussiano}} \xrightarrow{\text{IFFT2}} \xrightarrow{\text{Filtro Bilaterale}} \text{Immagine Restaurata}$$

Valutazione Complementare: PSNR vs. SSIM

La coppia di metriche statistiche PSNR e SSIM fornisce una valutazione qualitativa e morfologica complementare del processo di restauro:

- **PSNR:** Penalizza uniformemente lo scarto quadratico medio pixel per pixel.
- **SSIM:** Valuta la preservazione di strutture locali percettivamente rilevanti (luminanza, contrasto e contorni).

Nella pratica, esiste un compromesso analitico (*trade-off*) tra **riduzione del rumore** e **preservazione dei dettagli**: filtri spaziali eccessivamente aggressivi attenuano bene il rumore ad alta frequenza, ma degradano trame fini e smussano bordi netti — il che **riduce simultaneamente** sia il PSNR che lo SSIM rispetto all'immagine originale. La sfida nella progettazione dei filtri è trovare il punto di equilibrio che massimizzi entrambe le metriche, garantendo un restauro fedele e visivamente gradevole.

5.9.1 Analisi delle Prestazioni e Conclusione del Capitolo

I risultati numerici e visivi generati dalla Figura 5.30 dimostrano la rilevanza pratica di associare diversi domini di elaborazione. L'inserimento simultaneo di rumore periodico e stocastico corrompe le proprietà morfologiche del segnale, riducendo severamente gli indici di similarità e il rapporto segnale-rumore dell'immagine di riferimento.

L'isolamento e la soppressione dei picchi armonici nel dominio della frequenza tramite la maschera *notch* rimuovono le frange di interferenza sinusoidali sparse nello spazio bidimensionale. Come evidenziato nei dati stampati della Figura 5.30, questa filtrazione chirurgica promuove un salto immediato e sostanziale nella metrica PSNR. Tuttavia, il rumore gaussiano ad alta frequenza rimane attivo in modo omogeneo nello spettro, richiedendo un approccio complementare.

Il restauro finale è consolidato nel dominio spaziale con l'introduzione del filtro bilaterale. Diversamente dagli operatori passa-basso convenzionali (come quello gaussiano o di media), che smusserebbero indiscriminatamente rumore e contorni strutturali, la filtrazione bilaterale calcola pesi ponderati in base alla prossimità geometrica e alla differenza di intensità radiometrica. Questo comportamento adattivo attenua le fluttuazioni stocastiche residue nelle regioni di transizione graduale e preserva la nitidezza dei bordi spaziali.

La convergenza di entrambi gli approcci risulta in un **miglioramento sostanziale e simultaneo** di PSNR e SSIM rispetto all'immagine rumorosa — sebbene i valori finali rimangano inferiori a quelli dell'immagine originale (PSNR $= \infty$, SSIM $= 1{,}0$), a causa della perdita inevitabile di informazioni spettrali e testurali durante i processi di filtrazione. L'attenuazione graduale (gaussiana) dei picchi nello spettro evita artefatti di *ringing*, mentre il filtro bilaterale elimina il rumore stocastico residuo senza compromettere la nitidezza dei bordi. I risultati confermano l'efficacia e la complementarità pratica degli strumenti di analisi di frequenza presentati in questo capitolo, dimostrando che la filtrazione ibrida (frequenza + spaziale) è superiore a qualsiasi approccio isolato per il restauro di immagini degradate da rumore misto.

```
import numpy as np, cv2, os  # [pdi] passthrough: garante imports desta trilha
import numpy as np, cv2
```

```python
# Cameraman (asset del capitolo), 256x256 - stessa immagine della traccia py.
img_gray = mm.gray(cv2.resize(mm.read("imagens/cameraman.png"), (256, 256)))
h_img, w_img = img_gray.shape

#   1. Rumore misto: gaussiano + periodico
np.random.seed(42)
X2, Y2 = np.meshgrid(np.arange(w_img), np.arange(h_img))
u0, v0       = 15, 10
ruido_gauss  = np.random.normal(0, 15, img_gray.shape)
ruido_period = 30 * np.sin(2 * np.pi * (u0 * X2 / w_img + v0 * Y2 / h_img))
img_noisy    = np.clip(img_gray.astype(float) + ruido_gauss + ruido_period, 0,
255).astype(np.uint8)

#   2. Spettro (log-magnitudine) dell'immagine rumorosa
mag_n = mm.spectrumMag(img_noisy)

#   3. Maschera notch gaussiana sui 4 picchi periodici
def suprimir_pico_gaussiano(mask, cy, cx, sigma=3.0):
    yy, xx = np.meshgrid(np.arange(mask.shape[0]), np.arange(mask.shape[1]), indexing="ij")
    notch = np.exp(-((yy - cy)**2 + (xx - cx)**2) / (2 * sigma**2))
    return mask * (1.0 - notch)

cy0, cx0 = h_img // 2, w_img // 2
mascara_notch = np.ones((h_img, w_img), dtype=np.float64)
for dy, dx in [(v0, u0), (-v0, -u0), (v0, -u0), (-v0, u0)]:
    mascara_notch = suprimir_pico_gaussiano(mascara_notch, cy0 + dy, cx0 + dx, 3.0)

#   4. Filtraggio: notch nel dominio della frequenza + bilaterale
img_notch = mm.freqFilter(img_noisy, mascara_notch)
img_den   = cv2.bilateralFilter(img_notch, 7, 25, 7)
mascara_vis = (mascara_notch * 255).astype(np.uint8)

psnr_n  = mm.psnr(img_gray, img_noisy)
psnr_no = mm.psnr(img_gray, img_notch)
psnr_d  = mm.psnr(img_gray, img_den)
print(f"Rumorosa: PSNR={psnr_n:.2f} dB | Dopo notch: {psnr_no:.2f} dB | Notch+bilaterale:
{psnr_d:.2f} dB")

mm.show(
    [img_gray, img_noisy, mag_n, mascara_vis, img_notch, img_den],
    titles=[
        "Originale",
        f"Rumorosa (PSNR={psnr_n:.1f} dB)",
        "Spettro (picchi visibili)",
        "Maschera notch (gaussiana)",
        f"Dopo notch (PSNR={psnr_no:.1f} dB)",
        f"Notch + bilaterale (PSNR={psnr_d:.1f} dB)",
    ],
    cols=6, figsize=(20, 4)
)
```

```
Rumorosa: PSNR=20.19 dB | Dopo notch: 22.40 dB | Notch+bilaterale: 23.61 dB
```

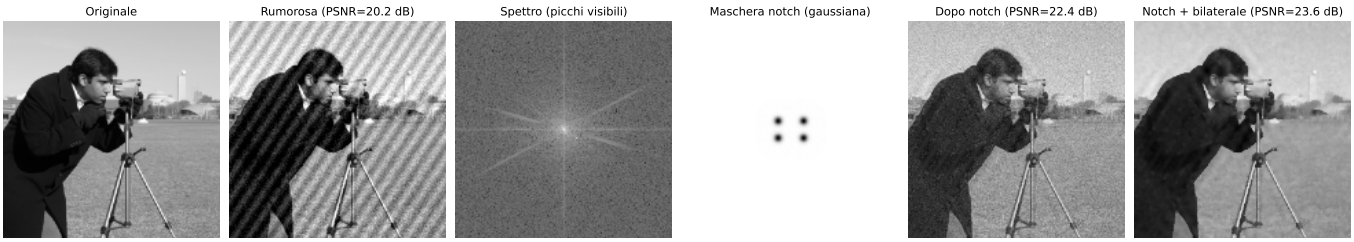


Figura 5.30: *Pipeline* completo di rimozione del rumore misto: (1) aggiunta di rumore gaussiano e periodico; (2) identificazione dei picchi di interferenza nello spettro delle frequenze; (3) applicazione di maschera *notch* con attenuazione gaussiana morbida; (4) post-elaborazione tramite filtro bilaterale per l'eliminazione del rumore stocastico residuo.

5.10 Riepilogo del Capitolo

La transizione dal **dominio spaziale** al **dominio delle frequenze** rivela la distribuzione spettrale dell'energia dell'immagine, stabilendo la base analitica per il filtraggio avanzato, il restauro e la compressione dei dati. L'articolazione strutturale di questi concetti è sintetizzata nella mappa concettuale della Figura 5.31.

Fondamenti Essenziali

- **DFT e Percezione Visiva:** Lo spettro scompone l'immagine in componenti armoniche. La **fase** mantiene l'intelligibilità geometrica della scena e la localizzazione dei contorni, mentre la **magnitudine** determina la distribuzione del contrasto e delle ampiezze globali.
- **Efficienza Algoritmica:** Il Teorema della Convoluzione rende possibile l'elaborazione di maschere su larga scala nel dominio della frequenza tramite FFT, riducendo la complessità computazionale asintotica da $O(N^2K^2)$ nello spazio a $O(N^2 \log N)$.
- **Fenomeno di *Ringing*:** Tagli netti nello spettro (filtri ideali) generano oscillazioni spaziali indesiderate (fenomeno di Gibbs). L'attenuazione graduale mediante filtri di **Butterworth** o **Gaussiani** elimina queste discontinuità.
- **Analisi Multirisoluzione tramite *Wavelets*:** Superando il carattere puramente globale di Fourier, la DWT cattura simultaneamente frequenza e localizzazione spaziale, costituendo la base dello standard JPEG 2000 e supportando rappresentazioni gerarchiche analoghe alle estrazioni di caratteristiche nelle Reti Neurali Convoluzionali (CNN).
- **Compressione Percettiva (DCT):** La pipeline JPEG sfrutta i limiti di contrasto del sistema visivo umano alle alte frequenze spaziali. La DCT isola l'energia di blocchi 8×8, consentendo alla quantizzazione di scartare i coefficienti AC di dettagli fini senza un danno percettivo severo.

Prossimi Passi: Il **Capitolo 6** inaugura la Parte II dell'opera, applicando gli strumenti di elaborazione delle immagini alla risoluzione di problemi reali di ispezione industriale. Verranno esplorate tecniche di **segmentazione e analisi delle forme** per il rilevamento automatico di difetti nelle linee di produzione — dall'identificazione di difetti superficiali nei pezzi alla lettura di *QRCode* nelle prove, consolidando il ponte tra la teoria presentata nella Parte I e le esigenze pratiche della visione computazionale.

5.11 🤖 Uso del Gemini Notebook come Tutore Complementare

In questa edizione, si incoraggia l'uso della piattaforma **Gemini Notebook** come strumento complementare di apprendimento — **non come sostituto** della lettura attenta, della risoluzione degli esercizi o della sperimentazione pratica. Basato su architetture di intelligenza artificiale, il sistema utilizza esclusivamente il materiale didattico e i documenti forniti dall'autore come base di conoscenza, garantendo che le risposte generate siano concettualmente allineate al contenuto programmatico e all'approccio pedagogico adottato nel corso di quest'opera.

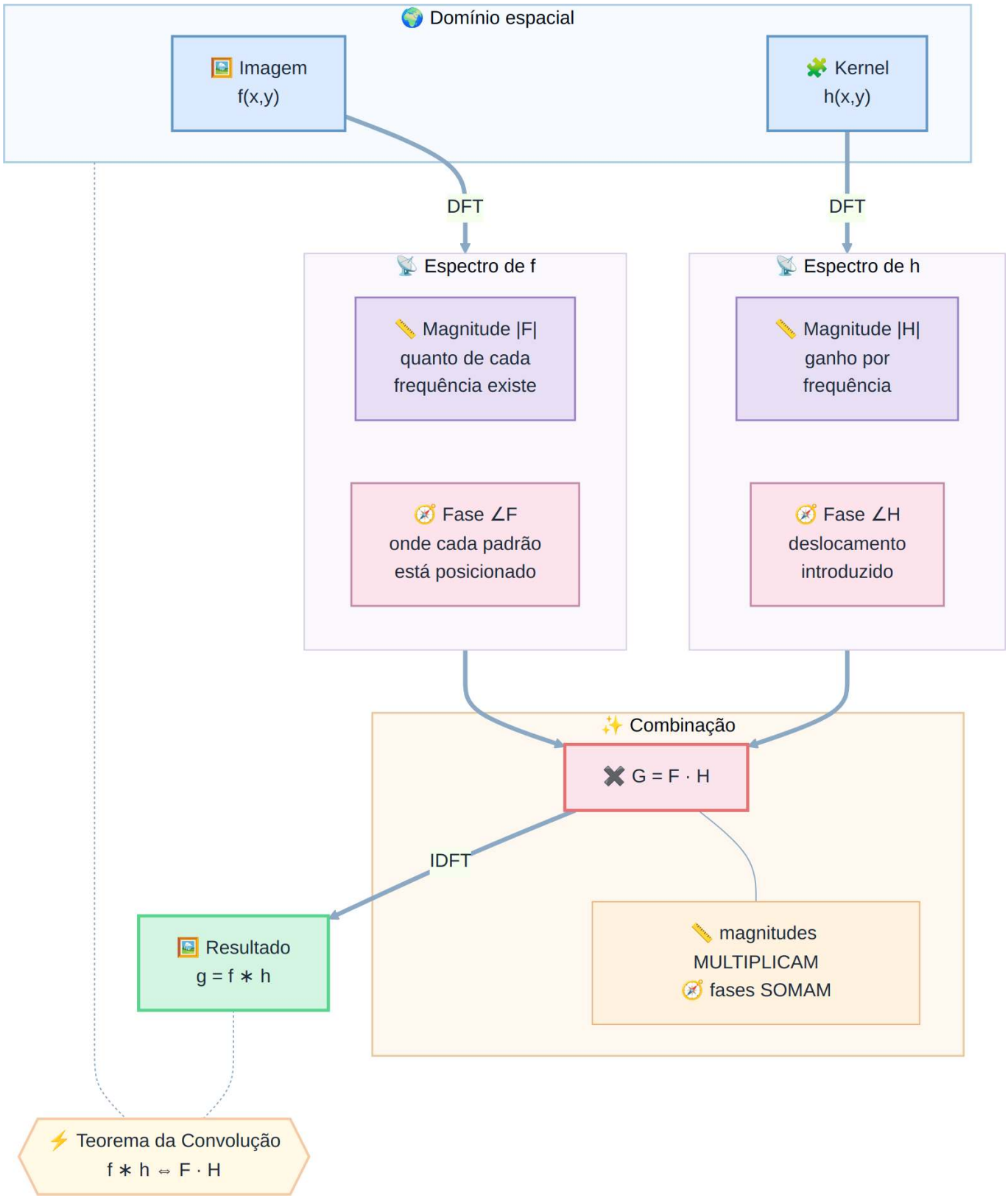


Figura 5.31: Mappa concettuale delle trasformazioni e delle proprietà nel dominio delle frequenze.

Accesso al Tutore Intelligente

🚀 ACCEDI A Gemini Notebook: CAPITOLO 05

🌐 Lingua e Linguaggio di Programmazione

Il progetto di questo capitolo nel Gemini Notebook è stato realizzato esclusivamente con il testo in **portoghese** e gli esempi di codice in **Python**. Se stai studiando dall'edizione in inglese o francese, oppure seguendo il percorso in C++, le risposte del tutore potrebbero non corrispondere esattamente alla versione che stai leggendo.

Linee Guida sul Contenuto Generato dall'Intelligenza Artificiale

Sebbene gli strumenti di intelligenza artificiale costituiscano alleati efficienti nel processo di apprendimento e revisione, il contenuto generato è soggetto a incongruenze o imprecisioni tecniche. Pertanto, è indispensabile la consultazione sistematica di libri di testo, articoli scientifici e fonti accademiche indicizzate per una validazione rigorosa delle informazioni. Si raccomanda vivamente l'esecuzione e la modifica degli esempi pratici in Python forniti in questo capitolo come metodo primario di verifica sperimentale dei risultati.

5.12 Elenco di Esercizi

1. **(10%) Implementazione Diretta della DFT 2D:** Implementare analiticamente la Trasformata Discreta di Fourier 2D (DFT) senza l'ausilio di funzioni native di librerie (come `np.fft.fft2`), utilizzando strettamente la formulazione matematica definita in Equazione 5.1 per una matrice di dimensioni 16×16. Eseguire la validazione numerica confrontando i coefficienti generati con i risultati della funzione `np.fft.fft2`, assicurandosi che la deviazione assoluta massima sia inferiore a 10^{-8}. Misurare i tempi di esecuzione di entrambi i metodi e presentare una giustificazione teorica per la disparità osservata in termini di complessità asintotica.

2. **(15%) Soppressione del Rumore Periodico:** Aggiungere interferenze sinusoidali con frequenze spaziali $(u_0, v_0) \in \{(5, 10), (20, 5), (30, 30)\}$ all'immagine di test del *Cameraman*. Per ogni scenario di degradazione, progettare una maschera di filtraggio *notch* specifica nel dominio della frequenza per isolare e attenuare i picchi armonici indesiderati. Valutare quantitativamente l'efficacia del processo di restauro mediante il calcolo delle metriche PSNR e SSIM. Discutere analiticamente il compromesso (*trade-off*) tra l'attenuazione del rumore sinusoidale e l'indesiderata attenuazione delle caratteristiche strutturali legittime dell'immagine.

3. **(15%) Analisi Comparativa degli Operatori Passa-Basso:** Condurre uno studio comparativo tra i filtri passa-basso Ideale, Gaussiano e Butterworth (con ordini armonici $n = 1, 2, 4$), parametrizzati con frequenze di taglio $D_0 = 20, 40, 60$ pixel. Per ogni combinazione strutturale, calcolare gli indici PSNR e SSIM dell'immagine risultante rispetto al segnale originale di riferimento. Organizzare i dati quantitativi in una tabella strutturata e tracciare i grafici unidimensionali delle funzioni di trasferimento corrispondenti lungo il profilo orizzontale $H(u, 0)$.

4. **(15%) Banco di Filtri Multirisoluzione di Haar:** Sviluppare uno script per eseguire manualmente la decomposizione *wavelet* discreta 2D di primo livello utilizzando la famiglia Haar. L'algoritmo deve calcolare i coefficienti dei filtri corrispondenti passa-basso (h) e passa-alto (g), applicandoli in modo separabile sulle righe e colonne della matrice, seguiti dall'operazione di decimazione (sottocampionamento spaziale per un fattore di 2). Validare numericamente l'accuratezza della propria implementazione confrontando le sottobande ottenute con l'output della funzione `pywt.dwt2(img, 'haar')`.

5. **(15%) Compressione Sparsa mediante Sogliatura Wavelet:** Applicare la tecnica di filtraggio per sogliatura netta (*hard thresholding*) sui coefficienti di dettaglio della decomposizione *wavelet*, adottando

le soglie numeriche $T \in \{5, 10, 20, 40, 80\}$ per le famiglie Haar, Daubechies (`db4`) e Symlets (`sym4`). Dopo aver eseguito il processo di sintesi mediante la trasformata inversa (`pywt.waverec2`), calcolare i valori di PSNR e SSIM di ciascuna immagine ricostruita. Identificare e giustificare quale combinazione di famiglia *wavelet* e soglia T massimizza la similarità strutturale.

6. **(15%) Costruzione di un Codificatore JPEG Semplificato:** Implementare la pipeline completa di compressione dei dati simulando lo standard JPEG. Il flusso deve comprendere: conversione spaziale $RGB \to YC_bC_r$, sottocampionamento cromatico nella proporzione 4:2:0, segmentazione della luminanza in blocchi disgiunti di 8×8 pixel, applicazione della DCT-II 2D ortogonale e quantizzazione lineare basata sulla matrice normalizzata di luminanza scalata per i fattori di qualità desiderati. Eseguire la decodifica inversa e confrontare quantitativamente le ricostruzioni con i file generati dalla funzione `cv2.imencode` per i fattori di qualità di 20, 50 e 80.

7. **(15%) Analisi Percettiva su Contenuti Eterogenei:** Sviluppare un'immagine sintetica composta da tre regioni distinte e di caratteristiche spettrali contrastanti: una texture fotografica complessa (che rappresenta alte frequenze stocastiche), un'area di testo vettorizzato con bordi netti (che rappresenta transizioni a gradino pure) e un gradiente lineare continuo (che rappresenta basse frequenze omogenee). Sottoporre questa immagine mista ai processi di compressione nei formati JPEG, PNG e WebP. Valutare e interpretare i risultati correlando la dimensione finale del file su disco alle metriche PSNR e SSIM ottenute, giustificando quale formato mostra le prestazioni migliori per segnali di natura eterogenea e perché tale vantaggio si verifica in termini di compattazione dell'energia e preservazione percettiva.

Riferimenti del Capitolo

La base teorica e lo sviluppo analitico dei concetti trattati in questo capitolo si fondano sulle seguenti opere di riferimento:

- **Gonzalez (2018)** — Formulazioni classiche delle Trasformate Discrete di Fourier 2D (DFT), progettazione di filtri analitici nel dominio della frequenza, Trasformata Discreta del Coseno (DCT) e principi fondamentali dei sistemi di compressione delle immagini.
- **Oppenheim (2010)** — Teoria formale di segnali e sistemi applicati nel dominio discreto, che copre le proprietà matematiche della DFT e la modellazione analitica del Teorema della Convoluzione.
- **Mallat (1999)** — Fondamento matematico della teoria delle *wavelet*, formalizzazione dell'analisi multirisoluzione (MRA) e architettura dei banchi di filtri diadici.
- **Wallace (1991)** — Specifica originale e aspetti ingegneristici dello standard di compressione ISO/IEC JPEG, con particolare enfasi sui criteri psicovisuali per la progettazione delle matrici di quantizzazione DCT.
- **Szeliski (2022)** — Modellazione computazionale e caratterizzazione delle metriche moderne di fedeltà e qualità percettiva (PSNR e SSIM), nonché l'analisi comparativa dei formati di immagine rasterizzati ad alte prestazioni.

5.13 5.1 Introduzione

Questo capitolo tratta della **elaborazione di immagini con Python**, concentrandosi sulla applicazione di filtri e tecniche di manipolazione dei pixel. Verranno presentati esempi pratici e esercizi proposti (EP) per consolidare l'apprendimento.

5.14 5.2 Filtri spaziali

I filtri spaziali operano direttamente sui pixel di un'immagine. Le operazioni più comuni includono:

- **Filtro medio**: sostituisce ogni pixel con la media dei pixel vicini, riducendo il rumore.
- **Filtro di Sobel**: evidenzia i contorni calcolando il gradiente dell'intensità.
- **Filtro gaussiano**: applica una convoluzione con una funzione gaussiana per sfocare l'immagine.

5.15 5.3 Esempi di codici

Di seguito alcuni frammenti di codice per illustrare le tecniche.

```
import numpy as np
from scipy import ndimage

# Applicazione di un filtro medio
def filtro_medio(immagine, dimensione=3):
    kernel = np.ones((dimensione, dimensione)) / (dimensione ** 2)
    return ndimage.convolve(immagine, kernel)
```

```
# Filtro di Sobel per il rilevamento dei contorni
def filtro_sobel(immagine):
    sobel_x = np.array([[-1, 0, 1],
                        [-2, 0, 2],
                        [-1, 0, 1]])
    sobel_y = np.array([[-1, -2, -1],
                        [0, 0, 0],
                        [1, 2, 1]])
    grad_x = ndimage.convolve(immagine, sobel_x)
    grad_y = ndimage.convolve(immagine, sobel_y)
    return np.sqrt(grad_x**2 + grad_y**2)
```

5.16 5.4 Esercizi proposti

5.16.1 EP1: Soglia adattiva

Implementare una funzione che applica una soglia adattiva a un'immagine in scala di grigi, utilizzando la media locale dei pixel.

5.16.2 EP2: Morfologia matematica

Applicare operazioni di erosione e dilatazione a un'immagine binaria utilizzando elementi strutturanti di diversa forma.

5.16.3 EP3: Trasformata di Hough

Utilizzare la trasformata di Hough per rilevare linee in un'immagine e visualizzarle sovrapposte all'originale.

5.17 5.5 Considerazioni finali

Le tecniche presentate costituiscono la base per lo sviluppo di sistemi di visione artificiale più complessi. Si consiglia di sperimentare variando i parametri e analizzando gli effetti sulle immagini.

ZQREF123Z ZQREF456Z

5.18 💻 Parte Pratica con Esercizi di Programmazione

La presente lista di esercizi di programmazione (EP) consolida le formulazioni teoriche presentate nel corso del Capitolo 5 — Trasformate e Compressione — attraverso un percorso pratico applicato. Gli esercizi

sono strutturati a partire da matrici di dimensioni ridotte, consentendo la validazione analitica e l'ispezione manuale di ciascun coefficiente, mantenendo la coerenza metodologica adottata nei capitoli precedenti.

L'incatenamento degli esercizi riproduce rigorosamente il flusso concettuale del capitolo: si inizia con l'implementazione esplicita della Trasformata Discreta di Fourier (DFT) a partire dalla sua definizione matematica fondamentale; si prosegue con la progettazione di filtri passa-basso e maschere *notch* nel dominio della frequenza; si applica la quantizzazione dei coefficienti (nucleo della compressione con perdita); e si conclude con l'integrazione di queste fasi nella costruzione di un *pipeline* di compressione JPEG semplificato e nell'analisi percettiva dei formati immagine.

! Linee Guida per la Risoluzione degli Esercizi di Programmazione

In tutti gli esercizi di questo capitolo, le coordinate del **centro dello spettro** (origine delle frequenze spaziali dopo l'applicazione dello spostamento `fftshift`) devono essere determinate tramite divisione intera. Per una matrice con L righe e C colonne, la componente di frequenza nulla si trova nella posizione:

$$(c_y, c_x) = \left(\left\lfloor \frac{L}{2} \right\rfloor, \left\lfloor \frac{C}{2} \right\rfloor \right)$$

Questa convenzione è rigorosamente identica a quella adottata dalla funzione `np.fft.fftshift`. Inoltre, in tutte le fasi che richiedono discretizzazione o arrotondamento numerico (sia nella quantizzazione dei coefficienti AC sia nella ricostruzione finale dei pixel), si deve impiegare l'arrotondamento standard al numero intero più vicino (*round half away from zero*), mitigando ambiguità in valori con frazione esattamente pari a 0.5.

🎯 Obiettivo di questo Quaderno

Il quaderno consente di sviluppare, validare, organizzare e testare soluzioni di **Esercizi di Programmazione (EPs)** in ambienti interattivi, come Colab, con gli stessi casi di test di Moodle, copiandoli lì solo al momento di registrare il voto ufficiale.

Download

Scarica `morph.py` e `testsuite.py` eseguendo la cella qui sotto:

```
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
```

```
Ambiente pronto. Morph: 1.1.9 | OpenCV: 5.0.0 | TestSuite: 1.1.2
```

Esecuzione dei Test

Per valutare i test, esegui `TestSuite("EP05_01.estensione").run()` in una nuova cella, sostituendo l'estensione con quella del linguaggio utilizzato (`.py`, `.java`, `.c`, `.cpp`, `.js` o `.r`). Il sistema scarica i casi di test da GitHub, esegue il programma e calcola automaticamente il voto.

Per testare il codice Python direttamente, senza salvare un file, usa `run_code(codice)` passando il codice come *stringa* in una variabile `codice`:

```
codice = """
from morph import mm
# ... il tuo codice qui ...
"""
TestSuite("EP05_01").run_code(codice)
```

5.18.1 EP05_01 🟢 Filtro Passa-Basso Ideale per Distanza nello Spettro

In uno ***scanner* di documenti antico**, il sensore cattura carta stropicciata e la trama delle fibre insieme al testo — rumore ad alta frequenza che "inquina" lo spettro ai bordi. Il tecnico della manutenzione non ha accesso all'immagine originale, ma solo allo **spettro di magnitudo già calcolato** dal software dello *scanner*. Il suo compito è semplice e chirurgico: mantenere solo il **cerchio centrale** delle basse frequenze (la struttura globale del documento) ed eliminare tutto ciò che si trova al di fuori del raggio D_0, rimuovendo la trama fine senza nemmeno dover toccare l'immagine spaziale.

Questo è il **Filtro Passa-Basso Ideale (LPFI)**: l'operazione spettrale più diretta del capitolo, ma anche quella che meglio rivela l'anatomia di uno spettro centrato.

5.18.1.1 📋 Linee Guida di Implementazione

1. **Dimensioni:** Leggere gli interi L (righe) e C (colonne) dello spettro di magnitudo — già fornito **centrato** (equivalente all'uscita di `np.fft.fftshift`).
2. **Frequenza di taglio:** Leggere l'intero D_0.
3. **Dati:** Leggere i valori interi della matrice di magnitudo, riga per riga.
4. **Centro dello spettro:** Calcolare $(c_y, c_x) = (L \,//\, 2,\ C \,//\, 2)$.
5. **Distanza:** Per ogni posizione (u, v), calcolare

$$D(u,v) = \sqrt{(u - c_y)^2 + (v - c_x)^2}$$

6. **Maschera ideale:** Applicare

$$H(u,v) = \begin{cases} 1, & D(u,v) \le D_0 \\ 0, & D(u,v) > D_0 \end{cases}$$

7. **Filtraggio:** Il valore di uscita è $\text{mag}'(u,v) = \text{mag}(u,v) \cdot H(u,v)$.
8. **Uscita:** Visualizzare la matrice filtrata con dimensioni $L \times C$.

5.18.1.2 📌 Vincoli Computazionali

- **Confronto non stretto:** il criterio usa $D(u,v) \le D_0$ (il confine appartiene al filtro, cioè viene mantenuto).
- **Tipo:** tutti i valori di ingresso e uscita sono interi; la distanza è calcolata in virgola mobile solo internamente.
- **Nessun arrotondamento della magnitudo:** poiché l'ingresso è già intero e la maschera è binaria (0 o 1), l'uscita non richiede mai arrotondamento.

5.18.1.3 🧠 Fondamenti Teorici

Regione	Distanza dal centro	Effetto del filtro
Centro ($D \le D_0$)	Basse frequenze	Preservate — struttura globale mantenuta
Bordi ($D > D_0$)	Alte frequenze	Azzerate — trama e rumore rimossi

Regione	Distanza dal centro	Effetto del filtro
D_0 **piccolo**	—	L'immagine ricostruita sarebbe molto sfocata
D_0 **grande**	—	Poca filtrazione; quasi tutta l'energia preservata

5.18.1.4 📦 Specifica di Ingresso e Uscita (VPL)

Ingresso:

- Riga 1: Intero L.
- Riga 2: Intero C.
- Riga 3: Intero D_0.
- Righe successive: Elementi interi della matrice di magnitudo (centrata).

Uscita:

- Matrice filtrata con L righe e C colonne, separati da spazi.

5.18.1.5 📌 Esempi

Ingresso	Uscita	Osservazione
3 3 1 10 20 30 40 50 60 70 80 90	0 20 0 40 50 60 0 80 0	Centro $(1,1)$. Gli angoli hanno $D = \sqrt{2} \approx 1.41 > 1$, quindi vengono azzerati; i vicini ortogonali hanno $D = 1 \leq 1$ e sono mantenuti.
1 3 0 5 9 7	0 9 0	$L = 1, C = 3$: centro in $(0,1)$. Solo la posizione centrale stessa $(D = 0)$ sopravvive a $D_0 = 0$.

```
%%writefile EP05_01.cpp
// your solution
```

```
Overwriting EP05_01.cpp
```

```
TestSuite("EP05_01.cpp").run()
```

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

5.18.2 EP05_02 🟡 Filtro *Notch*: Rimozione dei Picchi Periodici

Una telecamera di **ispezione industriale** acquisisce immagini di circuiti stampati, ma l'alimentazione della linea di produzione introduce un'**interferenza elettrica periodica** — un pattern di strisce quasi impercettibile a occhio nudo, che però appare nello spettro di Fourier come **coppie di picchi luminosi** posizionati simmetricamente attorno al centro. Il team di visione artificiale non può rielaborare l'acquisizione: deve **localizzare e cancellare chirurgicamente** queste coppie di picchi nello spettro, preservando tutta l'altra informazione utile dell'immagine.

Questo è il ruolo del **filtro rigetta-banda *notch***: a differenza del passa-basso (che interessa una regione continua), esso agisce su **punti specifici e sui loro simmetrici**, lasciando intatto il resto dello spettro.

Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.it/cap05/sim-ep0501.html

Figura 5.32: Simulatore EP05_01: Filtro Passa-Basso Ideale nello Spettro

5.18.2.1 📋 Linee Guida di Implementazione

1. **Dimensioni:** Leggere gli interi L (righe) e C (colonne) dello spettro di magnitudine centrato.
2. **Dati:** Leggere i valori interi della matrice di magnitudine, riga per riga.
3. **Picchi:** Leggere l'intero K (numero di coppie di picchi da rimuovere).
4. **Per ciascuno dei K picchi:** leggere tre interi Δv, Δu, r — spostamento verticale, spostamento orizzontale e raggio del *notch*.
5. **Centro dello spettro:** $(c_y, c_x) = (L \,//\, 2,\ C \,//\, 2)$.
6. **Soppressione simmetrica:** per ogni picco, azzerare **tutte** le posizioni (u, v) tali che la distanza dal punto $(c_y + \Delta v,\ c_x + \Delta u)$ sia $\leq r$, **e anche** tutte le posizioni con distanza $\leq r$ dal punto simmetrico $(c_y - \Delta v,\ c_x - \Delta u)$.
7. **Uscita:** Visualizzare la matrice risultante con dimensioni $L \times C$.

5.18.2.2 📌 Vincoli Computazionali

- **Simmetria obbligatoria:** ogni picco indicato genera **due** dischi azzerati (il punto e il suo simmetrico rispetto al centro) — dimenticare il simmetrico è l'errore più comune.
- **Sovrapposizione:** se due dischi si sovrappongono, la posizione rimane azzerata (non c'è "somma" o ripristino).
- **Confronto non stretto:** una posizione viene azzerata se distanza $\leq r$.
- **Ordine di lettura:** i K picchi devono essere elaborati nell'ordine in cui compaiono nell'input, ma il risultato finale non dipende dall'ordine (le operazioni di azzeramento sono commutative).

5.18.2.3 🧠 Fondamenti Teorici

Concetto	Ruolo nel filtro *notch*
Picco in $(\Delta v, \Delta u)$	Frequenza dell'interferenza periodica rilevata visivamente nello spettro
Punto simmetrico $(-\Delta v, -\Delta u)$	Ogni DFT di segnale reale è hermitiana: i picchi compaiono sempre in coppie simmetriche rispetto al centro

Concetto	Ruolo nel filtro *notch*
Raggio r	Controlla la "larghezza" della reiezione — un r grande rimuove più energia attorno al picco, ma anche informazione utile

5.18.2.4 📦 Specifica di Input e Output (VPL)

Input:

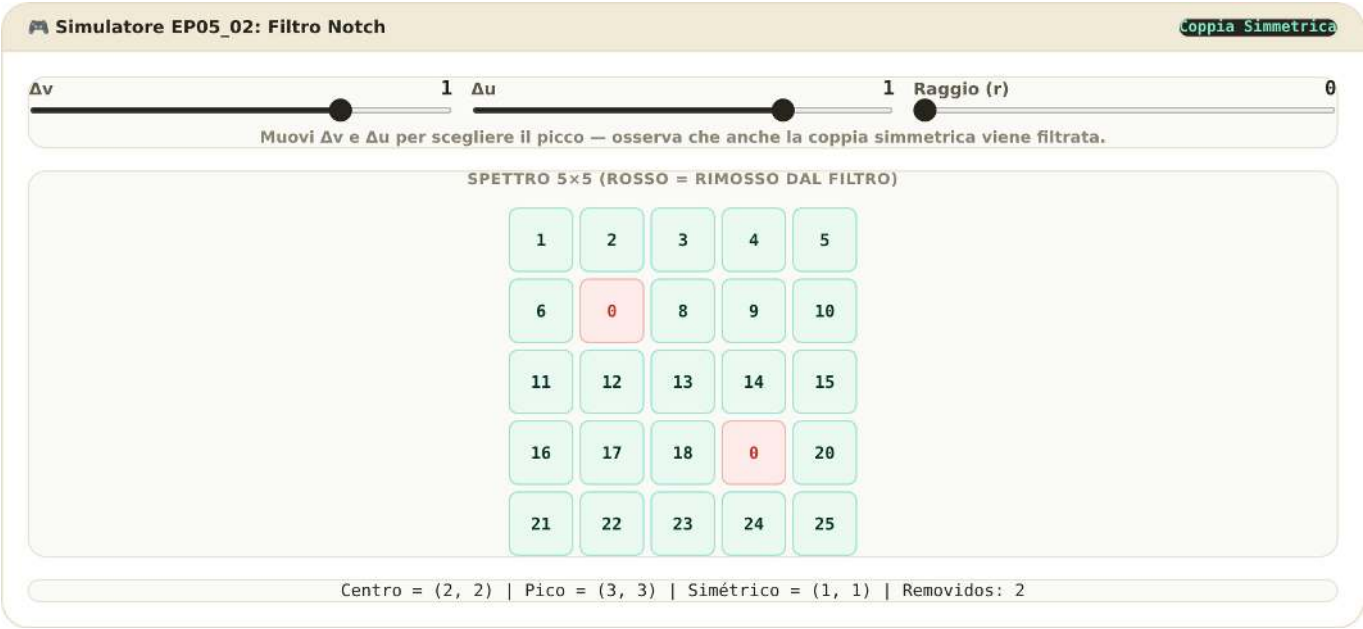
- Riga 1: Intero L.
- Riga 2: Intero C.
- Righe successive: Elementi interi della matrice di magnitudine (centrata), L righe.
- Riga successiva: Intero K.
- K righe successive: tre interi Δv, Δu, r (separati da spazi).

Output:

- Matrice risultante in L righe e C colonne, separati da spazi.

5.18.2.5 📌 Esempi

Input	Output	Osservazione
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	Centro $(c_y, c_x) = (2, 2)$. Il picco indicato $(\Delta v, \Delta u) = (1, 1)$ genera il punto $(3, 3)$ (valore 19) e il suo simmetrico $(1, 1)$ (valore 7), entrambi azzerati con $r = 0$ (solo i punti esatti).



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.it/cap05/sim-ep0502.html

Figura 5.33: Simulatore EP05_02: Filtro Notch

```
%%writefile EP05_02.cpp
// your solution
```

```
Overwriting EP05_02.cpp
```

```
TestSuite("EP05_02.cpp").run()
```

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

5.18.3 EP05_03 🟠 Quantizzazione DCT: la Vera Fonte di Compressione

Un'applicazione di **galleria fotografica** deve ridurre le dimensioni di migliaia di immagini prima di caricarle sul cloud, senza ricodificare tutto da zero. L'ingegnere responsabile ha già i **coefficienti DCT** di ogni blocco 4×4 calcolati (la fase computazionalmente onerosa è già stata eseguita) — manca solo applicare la **tabella di quantizzazione**, la fase che scarta realmente informazioni e genera compressione. I coefficienti ad alta frequenza, meno percettibili all'occhio umano, ricevono divisori grandi e tendono a diventare **zero**; i coefficienti a bassa frequenza, più percettibili, ricevono divisori piccoli e sopravvivono quasi intatti.

Dovrai implementare esattamente questa fase: **quantizzare e dequantizzare** (dividere, arrotondare, moltiplicare di nuovo) — il cuore della compressione *lossy* del JPEG.

5.18.3.1 📋 Linee Guida di Implementazione

1. **Dimensione del blocco:** Leggere l'intero N (blocco $N \times N$).
2. **Coefficienti:** Leggere la matrice C dei coefficienti DCT, N righe con N interi ciascuna (possono essere negativi).
3. **Tabella di quantizzazione:** Leggere la matrice Q, N righe con N interi positivi ciascuna.
4. **Quantizzazione:** Per ogni posizione (u, v), calcolare l'indice quantizzato

$$\tilde{C}(u,v) = \text{round}\left(\frac{C(u,v)}{Q(u,v)}\right)$$

   usando l'arrotondamento standard all'intero più vicino (i valori intermedi `.5` non si verificano mai nei casi di test).
5. **Dequantizzazione (ricostruzione):** Calcolare

$$C'(u,v) = \tilde{C}(u,v) \times Q(u,v)$$

6. **Output:** Visualizzare la matrice ricostruita C', $N \times N$, di interi.

5.18.3.2 📌 Vincoli Computazionali

- ***Round-trip* completo:** l'output è il coefficiente **ricostruito** ($\tilde{C} \times Q$), non l'indice quantizzato isolato.
- **Divisione in virgola mobile:** la divisione $C(u,v)/Q(u,v)$ deve essere eseguita in virgola mobile prima dell'arrotondamento — la divisione intera troncata produrrà un risultato errato.
- **Segno preservato:** i coefficienti negativi mantengono il segno dopo la quantizzazione e la ricostruzione.
- $Q(u,v) > 0$ **sempre:** non è necessario gestire la divisione per zero.

5.18.3.3 🧠 Fondamenti Teorici

Coefficiente	Frequenza	Valore tipico di Q	Effetto della quantizzazione
$C(0,0)$	DC (media del blocco)	Piccolo	Quasi sempre sopravvive — domina l'energia
$C(u,v)$ con $u+v$ basso	Bassa frequenza	Piccolo/medio	Parzialmente preservato

Coefficiente	Frequenza	Valore tipico di Q	Effetto della quantizzazione
$C(u,v)$ con $u+v$ alto	Alta frequenza	Grande	Spesso diventa zero — fonte della compressione

5.18.3.4 📦 Specifica di Input e Output (VPL)

Input:

- Riga 1: Intero N.
- N righe successive: matrice C (coefficienti DCT, interi, possono essere negativi).
- N righe successive: matrice Q (tabella di quantizzazione, interi positivi).

Output:

- Matrice ricostruita C', $N \times N$, interi separati da spazio.

5.18.3.5 📌 Esempi

Input	Output	Osservazione
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	$C(0,0) = 50/2 = 25 \to 25\times 2 = 50$ (preservato). $C(0,2) = -5/7 \approx -0.71 \to -1 \to -1 \times 7 = -7$. Invece $C(1,1) = -3/7 \approx -0.43 \to 0$: azzerato dalla quantizzazione — la maggior parte del blocco diventa zero, illustrando la compattazione dell'energia nell'angolo superiore sinistro.

Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.it/cap05/sim-ep0503.html

Figura 5.34: Simulatore EP05_03: Quantizzazione DCT (*round-trip*)

```
%%writefile EP05_03.cpp
// your solution
```

```
Overwriting EP05_03.cpp
```

```
TestSuite("EP05_03.cpp").run()
```

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

5.18.4 EP05_04 🔴 Implementazione della DFT 2D a partire dalla definizione

Un laboratorio di ricerca in **astronomia computazionale** ha ricevuto, da una missione remota, un piccolo sensore sperimentale i cui dati grezzi non possono essere elaborati tramite librerie moderne di FFT — l'ambiente di validazione è isolato e consente solo operazioni aritmetiche di base. Il team deve **reimplementare la Trasformata di Fourier Discreta 2D a partire dalla definizione matematica stessa**, cella per cella, per poi confrontare bit per bit con `np.fft.fft2` in un altro ambiente.

Questo è l'esercizio più concettuale della lista: non ci sono scorciatoie. Dovrai implementare direttamente la doppia sommatoria della Equazione 5.1, evidenziando *perché* la FFT esiste — e il costo computazionale che essa evita.

5.18.4.1 📋 Linee guida per l'implementazione

1. **Dimensioni:** Leggere i numeri interi M (righe) e N (colonne) dell'immagine $f(x, y)$.
2. **Dati:** Leggere i valori interi di $f(x, y)$, riga per riga.
3. **DFT 2D:** Per ogni coppia di frequenze (u, v) con $u = 0, \ldots, M-1$ e $v = 0, \ldots, N-1$, calcolare

$$F(u,v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y)\, e^{-j2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)}$$

   usando l'identità di Eulero $e^{-j\theta} = \cos(\theta) - j\sin(\theta)$ per separare la parte reale e quella immaginaria — **non utilizzare alcuna funzione FFT predefinita**.
4. **Magnitudine:** Calcolare $|F(u,v)| = \sqrt{\mathrm{Re}(F)^2 + \mathrm{Im}(F)^2}$ e arrotondare all'intero più vicino.
5. **Output:** Visualizzare la matrice delle magnitudini arrotondate, $M \times N$, nello stesso ordine (senza `fftshift` — il componente DC rimane in $(0, 0)$).

5.18.4.2 📌 Vincoli computazionali

- **Vietato l'uso di librerie FFT:** l'implementazione deve calcolare esplicitamente le doppie sommatorie (cicli annidati), anche se più lenta.
- **Senza `fftshift`:** l'output mantiene la convenzione grezza della DFT, con il componente DC in $F(0, 0)$ (angolo superiore sinistro).
- **Arrotondamento:** la magnitudine finale deve essere arrotondata all'intero più vicino; nei casi di test non vi è ambiguità `.5`.
- **Precisione:** piccoli errori di virgola mobile (ordine di 10^{-6}) prima dell'arrotondamento sono previsti e non influenzano il risultato intero finale.

5.18.4.3 🧠 Fondamenti teorici

Elemento	Significato
$F(0, 0)$	Componente DC — somma di tutti i pixel, $F(0,0) = \sum f(x,y)$
Parte reale $\mathrm{Re}(F)$	Proiezione del segnale sui coseni
Parte immaginaria $\mathrm{Im}(F)$	Proiezione del segnale sui seni
Complessità di questa implementazione	$\mathcal{O}((MN)^2)$ — ecco perché la FFT, con $\mathcal{O}(MN \log(MN))$, è indispensabile nelle immagini reali

5.18.4.4 📦 Specifica di input e output (VPL)

Input:

- Riga 1: intero M.
- Riga 2: intero N.
- Righe successive: elementi interi di $f(x, y)$, M righe.

Output:

- Matrice delle magnitudini $|F(u, v)|$ arrotondate, $M \times N$, separate da spazio.

5.18.4.5 📌 Esempi

Input	Output	Osservazione
2 2 1 2 3 4	10 2 4 0	$F(0,0) = 1+2+3+4 = 10$ (DC = somma totale). $F(0,1) = (1-2)+(3-4) = -2 \to \|F\| = 2$. $F(1,0) = (1+2)-(3+4) = -4 \to \|F\| = 4$. $F(1,1) = (1-2)-(3-4) = 0$.

Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.it/cap05/sim-ep0504.html

Figura 5.35: Simulatore EP05_04: DFT 2D manuale

```
%%writefile EP05_04.cpp
// your solution
```

```
Overwriting EP05_04.cpp
```

```
TestSuite("EP05_04.cpp").run()
```

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

5.18.5 EP05_05 🏆 *Pipeline* JPEG Completo: DCT, Quantizzazione e Ricostruzione

Sei stato incaricato di creare, da zero, un **codec JPEG didattico** in un ambiente embedded, senza alcuna libreria di immagini disponibile — solo operazioni matematiche di base. Il cliente vuole capire esattamente dove la qualità viene persa e dove viene recuperata, blocco per blocco. Questa è la sfida finale

del capitolo: integrare **tutto** ciò che è stato studiato — la DCT-II ortonormale, la quantizzazione percettiva e la ricostruzione tramite IDCT — in un unico *pipeline* end-to-end, elaborando un blocco $N \times N$ dall'inizio alla fine, esattamente come fa internamente lo standard JPEG, 8×8 pixel alla volta.

5.18.5.1 📋 Linee Guida di Implementazione

1. **Dimensione del blocco:** Leggere il numero intero N.
2. **Blocco originale:** Leggere la matrice di pixel $f(x, y)$, N righe con N interi in $[0, 255]$.
3. **Tabella di quantizzazione:** Leggere la matrice Q, $N \times N$ interi positivi.
4. **Centratura:** Sottrarre 128 da ogni pixel: $g(x, y) = f(x, y) - 128$.
5. **DCT-II 2D ortonormale:** Calcolare

$$C(u, v) = \alpha(u)\,\alpha(v) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} g(x, y)\, \cos\left[\frac{\pi(2x+1)u}{2N}\right] \cos\left[\frac{\pi(2y+1)v}{2N}\right]$$

   con $\alpha(0) = \sqrt{1/N}$ e $\alpha(k) = \sqrt{2/N}$ per $k > 0$.
6. **Quantizzazione:** $\tilde{C}(u, v) = \text{round}(C(u, v)/Q(u, v))$.
7. **Dequantizzazione:** $C'(u, v) = \tilde{C}(u, v) \times Q(u, v)$.
8. **IDCT-II 2D (inversa ortonormale):** Calcolare $g'(x, y)$ da $C'(u, v)$ usando la trasformata inversa corrispondente (stessa base, sommatoria su u, v).
9. **Inversione della centratura e arrotondamento:** $f'(x, y) = \text{round}(g'(x, y) + 128)$, limitato all'intervallo $[0, 255]$ (*clipping*).
10. **Output:** Visualizzare il blocco ricostruito f', $N \times N$, interi.

5.18.5.2 📌 Vincoli Computazionali

- ***Pipeline* completo obbligatorio:** tutte e sei le fasi (centrare, DCT, quantizzare, dequantizzare, IDCT, invertire) devono essere implementate — saltare la quantizzazione non supera i test, poiché il risultato sarebbe identico all'originale.
- ***Clipping*:** i valori ricostruiti al di fuori di $[0, 255]$ devono essere troncati (0 se negativi, 255 se maggiori di 255).
- **Arrotondamento:** sia nella quantizzazione che nella ricostruzione finale dei pixel, usare l'arrotondamento standard; i casi di test evitano ambiguità `.5`.
- **Base ortonormale:** la normalizzazione $\alpha(u)$ e $\alpha(v)$ deve essere applicata esattamente come specificato — senza di essa, la IDCT non ricostruisce correttamente.

5.18.5.3 🧠 Fondamenti Teorici

Fase	Analoga nel vero standard JPEG	Dove la qualità viene persa
Centratura	Stessa — la DCT presuppone un segnale centrato su zero	Nessuna perdita
DCT-II	Fasi 3–4 del *pipeline* (Tabella 5.7)	Nessuna perdita (trasformazione esatta e reversibile)
Quantizzazione	Fase 5 — divisione per $Q(u, v)$	**Principale fonte di perdita** — i coefficienti ad alta frequenza diventano zero
IDCT	Ricostruzione finale	Ricostruisce esattamente i coefficienti *quantizzati*, non quelli originali

5.18.5.4 📦 Specifica di Input e Output (VPL)

Input:

- Riga 1: Intero N.
- N righe successive: blocco originale $f(x, y)$, interi in $[0, 255]$.
- N righe successive: tabela de quantização Q, inteiros positivos.

Output:

- Bloco reconstruído $f'(x, y)$, $N \times N$, inteiros em $[0, 255]$, separados por espaços.

5.18.5.5 📌 Exemplos

Input	Output	Observação
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	Após a DCT, a quantização agressiva nas altas frequências (valores grandes de Q no canto inferior direito) e a reconstrução via IDCT fazem o bloco resultar **próximo** do original, mas não idêntico — a diferença é o custo da compressão *lossy*.

5.18.5.6 💡 Dica de Depuração

Se o resultado não corresponder, verifique nesta ordem: (1) os coeficientes DCT brutos (antes da quantização) — eles devem reconstruir o original **exatamente** via IDCT se você pular as fases 6–7; (2) a tabela $\alpha(u)$ — um erro comum é aplicar $\sqrt{2/N}$ também para $u = 0$; (3) o arredondamento da quantização, que deve ocorrer **antes** de multiplicar de volta por Q.

Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.it/cap05/sim-ep0505.html

Figura 5.36: Simulatore EP05_05: *Pipeline* JPEG completo a blocchi

```
%%writefile EP05_05.cpp
// your solution
```

```
Overwriting EP05_05.cpp
```

```
TestSuite("EP05_05.cpp").run()
```

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

Riferimenti

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.

Appendice A

Sobre o MCTest

O **MCTest** é um sistema de código aberto para criação e correção automática de exames parametrizados (Zampirolli, 2023). Ele oferece suporte a questões de múltipla escolha, dissertativas e de programação, estas últimas integradas ao VPL do Moodle, descrito no Apêndice B.

A versão do MCTest utilizada neste livro é a **5.4**, disponível em https://github.com/fzampirolli/mctest. A subpasta `book` do repositório contém as versões **5.3**, descrita em (Zampirolli, 2023), e **5.4**, utilizada para gerar as provas apresentadas neste livro e atualmente em produção na UFABC (https://mctest.ufabc.edu.br).

A.1 Instalação do MCTest

A instalação recomendada utiliza o VirtualBox com Ubuntu 22.04. No terminal do Ubuntu, como *root*, execute:

```
wget https://raw.githubusercontent.com/fzampirolli/mctest/master/_setup-all.sh
```

Em seguida, substitua `yourLogin` pelo nome do usuário local e execute:

```
sed -i 's/\/home\/fz\//\/home\/yourLogin\//g' _setup-all.sh
source _setup-all.sh
pip install mysqlclient
```

Após alguns minutos, o MCTest estará configurado. Para executá-lo:

```
source /home/yourLogin/PycharmProjects/runDjango.sh
```

O sistema fica então acessível em `http://127.0.0.1:8000`.

A.2 Criando um exame no MCTest

No MCTest, todo exame é configurado em uma tela de **Exame**, que reúne Turmas, Tópicos (associados a uma disciplina, como PDI-VC) e Questões — cada questão pertence a um único tópico. Além dos atributos de banco de dados, o exame possui parâmetros próprios, como o número de questões sorteadas e o número de variações geradas.

No caso das duas provas descritas neste apêndice, foi definido um conjunto de questões parametrizadas por prova, das quais um subconjunto é sorteado aleatoriamente para cada variação, totalizando **110 variações** distintas — uma por estudante matriculado nas turmas.

O botão **Create-Variations** gera um novo conjunto de variações a cada acionamento. Quando as questões estão associadas a atividades VPL, o professor recebe por e-mail um arquivo `*linker.json` contendo todos os casos de teste das variações geradas. O botão **Create-PDF** gera, para cada turma, um PDF com as provas sorteadas por estudante, além de um arquivo `*students_variations.csv` com nome, e-mail e variação sorteada de cada estudante.

> **!** Atenção
>
> Cada acionamento do botão **Create-Variations** gera um novo conjunto de variações de exames. Após imprimir o PDF, é fundamental **não alterar os atributos do exame**, sob risco de invalidar a correção automática das provas já impressas ou aplicadas.

A.3 Criando uma questão paramétrica

Uma questão paramétrica no MCTest combina três elementos:

1. **Descrição** — o enunciado da questão, escrito em LaTeX, contendo variáveis entre `[[code:variavel]]`, que são substituídas pelo valor sorteado para cada estudante;
2. **Bloco de definição** (`[[def: ... ]]`) — um trecho de código Python, embutido na própria questão, responsável por sortear os parâmetros, calcular a solução de referência e montar os casos de teste;
3. **Casos de teste para o Moodle** (bloco `moodle_cases`) — dicionário serializado em JSON contendo as entradas e saídas esperadas, consumido pela atividade VPL.

Um exemplo simplificado de definição de questão (adaptado da geração de um tabuleiro de xadrez `h × w`) é:

```
[[code:texto]]

\vspace{2mm}\noindent\textbf{Exemplo de Entrada:}\vspace{-2mm}\
\begin{verbatim}
[[code:caso0_inp]]
\end{verbatim}
\vspace{-2mm}\noindent\textbf{Exemplo de Saída:}\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 # TEM QUE INCLUIR "topic." no 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

# PARÂMETROS USADOS NA DESCRIÇÃO DA QUESTÃO
height = int(np.random.randint(5, 15))

# ENUNCIADO COMPLETO, COM O VALOR DE height JÁ INTERPOLADO
texto = (
    r"\vspace{2mm}\noindent\textbf{Descrição:}\vspace{-2mm} "
    r"Escreva um programa que leia um número inteiro \texttt{W}, representando a "
    r"largura (número de colunas) de um tabuleiro, e imprima esse tabuleiro no "
    r"formato de um tabuleiro de xadrez, usando os dígitos \texttt{0} e \texttt{1}. "
    r"O tabuleiro impresso terá sempre altura (número de linhas) igual a "
    f"\\textbf{{{height}}} "
    r"e largura igual ao valor \texttt{W} lido da entrada. "
```

```
    r"Considerando a posição $(i, j)$ do tabuleiro (indexada a partir de 0, com "
    r"$i$ representando a linha e $j$ a coluna), o valor impresso nessa posição "
    r"deve ser \texttt{1} se $i + j$ for ímpar, e \texttt{0} caso contrário. Cada "
    r"linha do tabuleiro deve ser impressa em uma linha separada, com os valores "
    r"de cada coluna separados por um espaço."
)

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']      = ["matrizes", "laços de repetição", "formatação de saída"]
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]
]]
```

O valor de `height` é sorteado uma única vez por variação (para aparecer no enunciado), enquanto `width` é sorteado a cada caso de teste, aumentando a robustez da correção automática.

Vale destacar o papel duplo da variável `texto` nesse mecanismo. Ela é ao mesmo tempo:

- **conteúdo do enunciado impresso**, inserida na descrição LaTeX da questão através da tag `[[code:texto]]`, aparecendo no PDF gerado pelo botão **Create-PDF**; e
- **entrada do dicionário exportado para o Moodle**, através da chave `cases['description']`, que passa a compor o JSON `moodle_cases`. É justamente esse campo que a atividade VPL exibe ao estudante quando ele abre a questão no Moodle para avaliá-la ou submeter uma solução.

Há, porém, uma diferença importante entre esses dois usos: o PDF é composto em LaTeX e, portanto, interpreta normalmente comandos como `\textbf{}`, `\texttt{}` ou `$...$`. Já o VPL do Moodle **não** processa LaTeX — ele espera texto puro. Por isso, ao montar `cases['description']`, `texto` não é inserido diretamente, mas sim passado pela função utilitária `latex_to_text()`, própria do MCTest, que remove (ou converte) os comandos e símbolos LaTeX do enunciado, produzindo uma versão em texto simples equivalente. Assim, `[[code:texto]]` no PDF continua recebendo o `texto` original, com toda a formatação LaTeX, enquanto `cases['description']` recebe `latex_to_text(texto)`, garantindo que o enunciado exibido no Moodle seja legível mesmo sem suporte a LaTeX.

Como `texto` é montada dentro do próprio bloco `[[def: ...]]` — no mesmo escopo em que `height`, `width` e os demais parâmetros são sorteados —, ela é recalculada a cada variação. Isso garante que o enunciado visto pelo estudante no Moodle seja **sempre idêntico** ao que foi impresso na prova em papel, mesmo quando o texto muda de estudante para estudante junto com os valores sorteados. A Seção A.4 retoma esse ponto em um caso real, no qual o enunciado é significativamente mais longo e descreve, além dos parâmetros numéricos, o próprio cenário visual gerado para cada variação.

A Figura A.1 mostra a questão do tabuleiro de xadrez efetivamente gerada pelo MCTest para uma das variações, com o enunciado já contendo o valor sorteado de `height` e o exemplo de entrada/saída correspondente ao primeiro caso de teste:

Ao acionar **Create-PDF** na tela da questão, uma nova variação é gerada a cada clique, permitindo conferir o enunciado antes de publicá-lo.

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
```

Figura A.1: Questão do tabuleiro de xadrez gerada pelo MCTest, ilustrando o enunciado parametrizado com o valor de `height` sorteado para a variação e o exemplo de entrada/saída do primeiro caso de teste.

Para as duas provas da disciplina, esse mecanismo foi utilizado de forma mais ampla: cada questão paramétrica define não apenas os valores numéricos, mas também, em alguns casos, pequenas variações estruturais no enunciado (por exemplo, qual direção cardinal — Norte, Sul, Leste, Oeste — deve ser avaliada), dificultando a busca por soluções prontas na internet ou em ferramentas de IA generativa.

A.4 Exemplo real: uma questão do Simulado 4

Para ilustrar o mecanismo descrito na seção anterior com um caso concreto, apresenta-se a seguir uma questão efetivamente aplicada no **Simulado 4** (tópico *im4-Operadores Morfológicos*, dificuldade 1, taxonomia de Bloom "remember"), do tipo questão de programação parametrizada com integração Moodle+VPL.

A questão pede ao estudante que escreva um programa capaz de:

- ler uma imagem binária, corrompida por ruído sal e pimenta, contendo ao menos dois objetos geométricos isolados;
- aplicar filtragem morfológica (abertura seguida de fechamento) para eliminar o ruído;
- extrair, a partir da imagem filtrada, as medidas geométricas de cada objeto (área, perímetro, centro, bounding box, circularidade, solidez e número de vértices), usando a função auxiliar `mm.measure`;
- ordenar os objetos por posição (coordenada X do bounding box, com Y como critério de desempate) e reatribuir os identificadores sequencialmente;
- imprimir a matriz limpa e a tabela de medidas no formato esperado pelo corretor automático.

No bloco `[[def: ... ]]` correspondente, um gerador de cena (`gerarCena`) sorteia aleatoriamente a altura e a largura da imagem, o tipo, o tamanho e a posição de cada objeto (quadrado, retângulo, triângulo ou bloco), garantindo que eles não se sobreponham, e em seguida acrescenta ruído sal e pimenta com 3% de probabilidade por pixel. Dez casos de teste distintos são gerados dessa forma para cada variação da prova, e o par entrada/saída do primeiro caso é reaproveitado no próprio enunciado como exemplo para o estudante. A biblioteca de morfologia (`mm`) é importada diretamente de `morph.py`, também disponível como módulo utilitário do próprio MCTest.

Em resumo, o enunciado pede ao estudante um programa que:

1. leia, na primeira linha, dois inteiros H e W (altura e largura da imagem);
2. leia as H linhas seguintes da matriz binária (0s e 1s separados por espaço), usando `mm.readImg(H, W)`;
3. aplique filtragem morfológica para remover o ruído sal e pimenta;
4. imprima a matriz já filtrada, em 0s e 1s separados por espaço;
5. extraia as medidas geométricas dos objetos com `mm.measure(imgLimpa)`;
6. ordene os objetos e imprima a tabela de medidas no formato esperado.

O enunciado ainda traz duas observações importantes para a correção automática: (i) a área calculada

pelo OpenCV (`cv2.contourArea`) corresponde ao polígono contínuo delimitado pelos centros dos pixels de borda, sendo por isso menor do que a simples contagem de pixels 1 (`np.sum`); e (ii) a lista de objetos deve ser ordenada em ordem crescente pela coordenada X do bounding box (`bbox[0]`), usando a coordenada Y (`bbox[1]`) como critério de desempate, com os IDs reatribuídos sequencialmente de 1 a N após a ordenação.

Assim como no exemplo da Seção A.3, o texto do enunciado aqui também não é fixo: ele é montado dentro do próprio bloco `[[def: ... ]]`, na variável Python `texto`, e desempenha o mesmo papel duplo já descrito — alimenta o PDF via `[[code:texto]]` e é exportado, já convertido por `latex_to_text()`, em `cases['description']` dentro de `moodle_cases`, sendo essa a versão exibida ao estudante no VPL do Moodle. A diferença é que, neste caso real, é o **cenário visual** (imagem ruidosa, número e posição dos objetos) que muda de estudante para estudante a cada variação, enquanto a redação de `texto` permanece fixa entre variações, já que apenas os dados numéricos (imagem e casos de teste) são sorteados. O ideal seria que a redação também variasse a cada geração, como no exemplo anterior, em que o valor de `height` era interpolado diretamente no texto da descrição. A estrutura real utilizada (com trechos de sorteio de cenário omitidos por brevidade) é:

```
[[code:texto]]

\vspace{2mm}\noindent\textbf{Exemplo de Entrada:}\vspace{-2mm}\
\begin{verbatim}
[[code:caso0_inp]]
\end{verbatim}
\vspace{-2mm}\noindent\textbf{Exemplo de Saída:}\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 # TEM QUE INCLUIR "topic." no MCTest

# ... geração da cena, do ruído e dos 10 casos de teste (inplist/outlist) ...

# ENUNCIADO COMPLETO, COM EXPLICAÇÃO DE ÁREA E DE ORDENAÇÃO
texto = (
    "Uma imagem binária corrompida por ruído sal e pimenta contém no mínimo dois objetos
    geométricos isolados.\n\n"
    "Escreva um programa que:\n"
    "1. Leia dois inteiros ..."
)

cases = {}
cases['skills']      = ["morfologia matematica", "remocao de ruido", "mm.measure",
"tabulacao"]
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]
]]
```

Abaixo, a fotografia da questão efetivamente gerada e impressa para uma das 110 variações do Simulado

4, mostrando o enunciado, a imagem de entrada ruidosa e o exemplo de saída esperado (imagem filtrada e tabela de medidas):

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
```

Figura A.2: Questão real gerada pelo MCTest para o Simulado 4 — tópico "Operadores Morfológicos".

A.5 Considerações finais

Este apêndice resumiu os passos para instalar o MCTest, criar exames e construir questões parametrizadas com integração ao VPL do Moodle. O uso combinado de variáveis sorteadas na descrição (`[[code:...]]`) e de código Python embutido na própria questão (`[[def: ... ]]`) permite gerar centenas de variações de prova a partir de um único modelo, cada uma corrigida automaticamente e de forma consistente — com a variável `texto`, em particular, garantindo que o mesmo enunciado seja exibido tanto no PDF impresso quanto na avaliação da questão pelo estudante no Moodle. O exemplo real do Simulado 4 (Seção A.4) ilustra como esse mecanismo é usado na prática para gerar questões de programação com cenários visuais distintos por estudante. O Apêndice B detalha como essas variações são publicadas e corrigidas no Moodle, e o Apêndice C descreve como restringir o acesso às provas por meio do SEB.

Appendice B

Sobre o Moodle (VPL)

Este apêndice descreve a configuração de uma atividade **VPL (*Virtual Programming Lab*)** no Moodle, *plugin* utilizado para a submissão e a correção automática dos Exercícios de Programação (EPs), dos simulados quinzenais e das provas descritas neste livro. A versão do Moodle utilizada foi a **4.5**.

B.1 Configurando uma atividade VPL

A criação de um exame no MCTest com integração ao VPL (ver Apêndice A) gera dois arquivos que devem ser renomeados para `linker.json` e `students_variations.csv`. Esses arquivos, juntamente com o `morph.py`, devem ser adicionados aos **Arquivos de Execução** da atividade VPL, além de todos os arquivos contidos em `VPL_modification/V14-AI-feedback.zip`, disponível no repositório do MCTest em https://github.com/fzampirolli/mctest. Todos eles devem ser marcados com a opção **"*Files to keep when running*"**.

Em síntese, a atividade VPL passa a conhecer, para cada estudante, qual variação da questão foi sorteada (por meio de `students_variations.csv`) e qual é o conjunto de casos de teste correspondente (por meio de `linker.json`). Isso permite que a correção seja individualizada, mesmo quando dezenas de estudantes resolvem a mesma atividade com variações distintas.

B.2 Publicando os EPs como atividades VPL

Os Exercícios de Programação (EPs) apresentados ao final de cada capítulo deste livro também podem ser publicados como atividades VPL, seguindo o mesmo mecanismo. O fluxo típico de uso em sala de aula foi:

1. Abertura dos dois *notebooks* Colab do capítulo (teórico e prático), com destaque para os simuladores interativos;
2. Execução dos blocos de código, alterando parâmetros para reforçar a compreensão dos conceitos;
3. Nas aulas em laboratório, submissão das respostas dos EPs diretamente na atividade VPL correspondente, com retorno imediato sobre a aprovação ou reprovação nos casos de teste.

Caso a atividade VPL não seja gerada pelo MCTest, os arquivos `linker.json` e `students_variations.csv` não serão necessários. Entretanto, para utilizar o *feedback* por IA (ver Apêndice D), devem ser adicionados os arquivos contidos em `VPL_modification/V14-EP0_VPL-TEMPLATE.zip`, disponível no mesmo repositório (https://github.com/fzampirolli/mctest), seguindo o procedimento descrito na Seção B.1.

> **Reaproveitamento dos casos de teste**
>
> Os arquivos `.cases` utilizados na validação local (classe `TestSuite`, descrita no corpo deste livro) são compatíveis com o formato esperado pelo VPL, permitindo reaproveitar o mesmo conjunto de testes tanto no desenvolvimento do EP quanto na correção automatizada no Moodle.

Nos simulados quinzenais — aplicados com o SEB (ver Apêndice C) e com bônus de 5% sobre a nota final —, foram utilizadas atividades VPL com EPs semelhantes aos das listas regulares, porém corrigidas sob restrição de acesso à internet, reforçando o caráter avaliativo da atividade.

B.3 Considerações finais

Este apêndice descreveu a configuração de atividades VPL no Moodle para a correção automática de EPs, simulados e provas parametrizadas. A combinação entre MCTest (geração de variações e casos de teste), VPL (submissão e correção) e SEB (restrição de acesso) forma o tripé sobre o qual se apoia a avaliação automatizada e individualizada utilizada ao longo da disciplina.

Appendice C

Uso do SEB nos simulados quinzenais e avaliações

Além das duas provas com questões parametrizadas (ver Apêndice A), o ***Safe Exam Browser*** **(SEB)** também foi utilizado para restringir o acesso aos **simulados quinzenais** — atividades VPL (*Virtual Programming Lab*) com Exercícios de Programação (EPs) semelhantes aos das listas regulares, valendo um bônus de até 5% sobre a nota final —, bem como às demais avaliações práticas.

O SEB garante um ambiente seguro de avaliação ao bloquear o acesso a recursos externos não autorizados nas máquinas do laboratório. Para a implementação adotada neste trabalho, utilizou-se o sistema operacional **Windows 10** e a versão **3.7.1.704** do SEB, disponível em https://safeexambrowser.org/, que deve estar previamente instalada em todas as estações do laboratório.

A seguir, descreve-se o procedimento para configurar o arquivo de ambiente do SEB e vinculá-lo à atividade VPL no Moodle.

C.1 Configurando a aplicação no *SEB Tool*

Para gerar o arquivo de configuração `.seb` que será distribuído aos estudantes, abra o utilitário *SEB Tool* e siga os passos descritos a seguir.

1. ***General***
   - No campo ***Start URL***, insira a URL direta da atividade VPL no Moodle.
   - *Nota:* caso seja necessário navegar por mais de uma atividade dentro do mesmo curso, insira a URL principal do curso e, na aba ***Browser***, marque a opção ***Allow navigation back/forward in exam***.
2. ***Network***
   - Inclua as URLs permitidas para navegação. Para restringir o acesso à atividade desejada, utilize uma expressão de filtro no padrão `/mod/vpl/*?id=4624*`, em que o número final corresponde ao identificador da atividade VPL no Moodle. Na Tabella C.1, a última linha contém a URL da página principal do curso (identificador `475`), mas ela também pode ser substituída pela URL de uma seção que contenha várias atividades. Essa configuração é opcional e útil quando há mais de uma atividade VPL. Nesse caso, a primeira linha da tabela deve ser repetida para cada atividade avaliativa que deverá ficar acessível durante o *exame*.
   - **Configurações gerais da aba *Filter***:
     ☒ *Activate URL filtering*
     ☐ *Filter also embedded content*

Tabella C.1: Expressões de filtro de URL utilizadas na configuração do 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*
[x]	[ ]	`https://moodle.ufabc.edu.br/course/switchrole.php*`	*Allow*

Active	*Regex*	*Expression*	*Action*
[x]	[ ]	`https://mctest.ufabc.edu.br/compare`	*Allow*
[x]	[ ]	`https://moodle.ufabc.edu.br/enrol/index.php?id=475`	*Allow*

3. ***Security***
   - Caso o laboratório utilize um projetor ou monitor secundário, ajuste o campo ***Maximum allowed number of connected displays*** para um valor maior que 1, evitando que o SEB bloqueie a exibição.
4. ***Config File***
   - Clique em ***Save Settings As...*** e salve o arquivo com a extensão `.seb`.
5. ***Exam***
   - Marque a opção ***Use Browser Exam Key and Configuration Key***.
   - Copie o código gerado no campo ***Browser Exam Key***. *Importante:* copie essa chave somente **após** salvar o arquivo no passo anterior.

C.2 Vinculando a chave do SEB à atividade VPL no Moodle

Após gerar o arquivo e obter a *Browser Exam Key*, acesse o Moodle para aplicar as restrições.

1. Na página da atividade VPL, clique no ícone de engrenagem (**Configurações**).
2. Acesse ***Configuration*** → ***Submission restrictions*** e clique em ***Show more***.
3. Altere a opção ***SEB browser required*** para ***Yes***.
4. Cole a chave copiada no campo ***SEB exam key(s)***.

C.3 Restrição por endereço IP (opcional)

Para garantir que as submissões ocorram exclusivamente a partir dos computadores do laboratório, é possível combinar o SEB com o filtro de rede do Moodle, preenchendo o campo ***Allowed submission from net***.

- **Faixas consecutivas:** utilize um hífen para definir o intervalo. Exemplo: `111.11.11.1-62` (permite os IPs de `111.11.11.1` a `111.11.11.62`).
- **IPs não consecutivos:** informe os endereços individuais separados por vírgulas.

C.4 Considerações finais

A integração do SEB com o VPL no Moodle cria um fluxo simples para o estudante: basta dar um duplo clique no arquivo `.seb` disponibilizado pelo docente para que o navegador seguro seja aberto diretamente na atividade configurada.

A combinação da *Browser Exam Key* com a restrição por faixa de endereços IP das máquinas do laboratório fornece uma solução robusta para avaliações de programação *online*, dificultando acessos indevidos e contribuindo para a integridade do processo de avaliação.

Appendice D

Geração de material didático com o *Gemini Notebook*

Este apêndice descreve o uso do ***Gemini Notebook*** **(*NotebookLM*)** como ferramenta de apoio à preparação do material didático da disciplina, complementando o uso de IA na revisão de textos e códigos, mencionado no prefácio deste livro.

D.1 Vídeos conceituais e de resolução de EPs

Para cada capítulo, foi criado um projeto no *Gemini Notebook* tendo como fontes o PDF completo do livro e o arquivo `morph.py`. A partir dessas fontes, o *Gemini Notebook* gerou automaticamente um vídeo curto (em média, 7 minutos), utilizado na abertura das aulas. Os capítulos alternaram dois tipos de vídeo:

- **Vídeo conceitual**, apresentando os fundamentos teóricos do capítulo, geralmente exibido na primeira aula dedicada ao tema;
- **Vídeo de resolução de EPs**, apresentando uma estratégia de solução para os Exercícios de Programação, exibido na aula seguinte, quando a maior parte dos EPs era resolvida em conjunto com a turma.

i Papel do vídeo na aula

O vídeo gerado pelo *Gemini Notebook* não substitui a explicação do professor. Ele funciona como uma breve introdução e motivação ao tema, contextualizando o estudante antes da apresentação dos *slides* e da abertura dos *notebooks Colab*.

D.2 *Slides* de apoio

Complementando os vídeos, o *Gemini Notebook* também gerou, a partir das mesmas fontes (o PDF do livro e o arquivo `morph.py`), um conjunto de aproximadamente 12 *slides* por capítulo, abrangendo tanto os conceitos teóricos quanto a parte prática. A geração utilizou um *prompt* específico para cada capítulo, delimitando o escopo do conteúdo a ser sintetizado e evitando tanto a antecipação de assuntos de capítulos posteriores quanto a reprodução de trechos extensos do livro sem adaptação didática.

Após a apresentação dos *slides*, a aula prosseguia com a abertura dos dois *notebooks Colab* do capítulo (teórico e prático), enfatizando os simuladores interativos e a execução dos blocos de código com alteração de parâmetros.

D.3 Material principal e considerações finais

A geração de vídeos e *slides* com o *Gemini Notebook*, a partir do próprio livro e da biblioteca `morph.py`, permitiu padronizar a abertura das aulas e reduzir o tempo de preparação do material didático, sem prescindir da curadoria do professor na elaboração dos *prompts* e na condução das atividades em sala de aula.

Os vídeos e *slides* produzidos pelo *Gemini Notebook* têm caráter **motivacional** e servem como ponto de partida para a discussão dos conceitos apresentados em cada capítulo. Como qualquer conteúdo gerado por IA, eles podem conter imprecisões ou erros ocasionais. Em alguns casos, essas imprecisões são exploradas intencionalmente durante a aula para verificar se os estudantes estão acompanhando criticamente a apresentação e conseguem identificar inconsistências conceituais.

O conteúdo oficial da disciplina, entretanto, é o material produzido pelo método descrito no prefácio deste livro, disponibilizado em três formatos complementares: **HTML**, para navegação com simuladores interativos; **PDF**, para leitura e impressão; e ***notebooks Jupyter*** (`.ipynb`), para execução e experimentação dos códigos. Todo o material gerado pelo *Gemini Notebook* deve ser entendido como um recurso complementar a esse conteúdo principal.

Assim como nos demais usos de IA generativa descritos neste livro, a responsabilidade pela qualidade técnica, pela correção conceitual e pela adequação pedagógica do material permanece com o professor.

Appendice E

Uso de IA na Avaliação de Estudantes: o Projeto vpl-ai-feedback

Este apêndice descreve o **vpl-ai-feedback**, sistema de código aberto desenvolvido para apoiar a correção de submissões de código enviadas ao VPL (Moodle) por meio de *feedback* gerado por Inteligência Artificial (IA). Diferentemente de um corretor automático tradicional — que apenas compara saídas com casos de teste —, o **vpl-ai-feedback** envia o código do estudante, junto com a rubrica da questão, a um modelo de linguagem (LLM), que devolve uma análise qualitativa por critério, no estilo de um ***feedback* socrático**: o estudante recebe comentários que apontam o que foi feito corretamente, o que pode ser melhorado e por quê, tanto em atividades **formativas** (exercícios de prática) quanto em atividades **avaliativas** (simulados) do VPL.

Esse mecanismo de *feedback* socrático em atividades VPL foi proposto por Zampirolli et al. (2026). O estudo descreve a integração de IA ao processo de avaliação de exercícios de programação parametrizados no VPL/Moodle, utilizando um conjunto de sete LLMs adotados para analisar o código dos estudantes e gerar *feedback* automatizado a cada solicitação de correção, com resultados preliminares indicando percepção positiva dos estudantes quanto à geração de ideias, clareza das explicações e autonomia de aprendizagem.

O restante deste apêndice detalha a implementação prática dessas ideias no projeto **vpl-ai-feedback** — cujo código está disponível em https://github.com/fzampirolli/vpl-ai-feedback — e descreve especificamente seu uso nas três turmas de PDI-VC entre maio e agosto de 2026.

> ⚠ Atenção: IA não substitui o julgamento docente
>
> O uso de IA para **avaliar** estudantes **não é recomendável como fonte única de nota**, pois modelos de linguagem podem **alucinar** — isto é, produzir avaliações, critérios ou justificativas plausíveis, porém incorretas, mesmo diante de um código correto ou incorreto. Um LLM pode atribuir nota alta a um código com erro sutil, ou nota baixa a um código correto mas escrito de forma incomum, simplesmente porque a explicação gerada "parece" coerente. **A IA deve ser usada apenas como um complemento à avaliação**, nunca como substituta do professor. O Apêndice A e o Apêndice B descrevem os mecanismos de correção objetiva (casos de teste) que continuam sendo a base da nota oficial; este apêndice trata exclusivamente do uso da IA como camada adicional de *feedback* qualitativo.

E.1 Contexto de uso

O **vpl-ai-feedback** foi utilizado em três turmas da disciplina **PDI-VC**, ministradas entre maio e agosto de 2026, totalizando aproximadamente uma centena de estudantes. O *feedback* por IA foi empregado de três formas complementares:

1. ***Feedback* contínuo em atividades formativas** — exercícios de prática enviados ao VPL ao longo do curso, nos quais o estudante podia reenviar o código quantas vezes desejasse. A cada submissão, recebia um *feedback* qualitativo gerado por IA, além da correção objetiva realizada pelo VPL, descrita em Zampirolli et al. (2026).
2. ***Feedback* complementar em simulados (atividades avaliativas bônus)** — quatro simulados realizados ao longo do quadrimestre, previstos no plano de ensino como atividades bônus, cada um

valendo até 5% da nota final. Aplicados aproximadamente 30 minutos antes do término das aulas práticas quinzenais em laboratório, os simulados utilizavam o **vpl-ai-feedback** para gerar uma rubrica individual de avaliação, enviada por e-mail juntamente com um resumo comparativo entre a nota atribuída pelo Moodle/VPL e a avaliação produzida pela IA.

3. ***Feedback* complementar nas Provas 1 e 2 (avaliações oficiais)** — as duas provas formais do quadrimestre, cada uma valendo uma parcela maior da nota final da disciplina, também passaram a utilizar o **vpl-ai-feedback** após sua aplicação. Diferentemente dos simulados (bônus), aqui a nota oficial já é definida pela correção objetiva do VPL e/ou avaliação manual do professor antes da divulgação; a rubrica gerada pela IA é enviada ao estudante apenas posteriormente, como material de apoio para revisão do próprio desempenho — reforçando, também nas avaliações de maior peso, a separação entre *nota oficial* e *feedback* complementar detalhada na Seção D.7.

Uma pesquisa de opinião, respondida por 102 estudantes antes da adoção do sistema, indicou elevada aceitação da proposta. Apenas 2 estudantes atribuíram nota 1 (rejeição) e 7 atribuíram nota 2 ao interesse em receber *feedback* automático de código por IA. Outros 19 declararam-se indiferentes (nota 3), enquanto a maioria — 38 e 37 estudantes — atribuiu notas 4 e 5 (alta aprovação), respectivamente, totalizando os 102 respondentes. Esse resultado motivou a adoção do sistema como complemento ao processo formativo, e não como substituto da correção humana.

! A nota oficial nunca foi a nota da IA

É fundamental destacar que, conforme definido no plano de ensino, **a nota utilizada tanto para o cálculo do bônus de cada simulado quanto para a nota oficial das Provas 1 e 2 foi sempre a nota atribuída pelo VPL** (baseada nos casos de teste) e/ou pela avaliação manual do professor — e **não** a nota sugerida pela IA. A rubrica gerada pelo **vpl-ai-feedback** foi enviada ao estudante apenas como material de apoio ao aprendizado — nunca como critério de atribuição de nota, seja em atividades bônus ou em avaliações oficiais. Essa separação entre *nota oficial* (VPL/professor) e *feedback complementar* (IA) é o princípio central deste apêndice e foi retomada na Seção D.8.

E.2 Arquitetura do projeto

O **vpl-ai-feedback** é um *script* Python assíncrono organizado da seguinte forma:

```
.
  config.yaml              ← configuração (credenciais, provedor, pesos) - NUNCA versionado
  config.yaml.example      ← template público de configuração
  main.py                  ← script principal (async), orquestra todo o processo
  run.sh                   ← wrapper de execução (bash run.sh config.yaml)
  gerar_relatorio.py       ← converte o consolidado *_ALL.txt em CSV
  enviar_email.py          ← envia as rubricas por e-mail a cada estudante
  providers/                ← clientes das APIs de LLM
      __init__.py          ← fábrica de clientes (Factory)
      base.py               ← lógica de retry, backoff e fallback entre modelos
      groq.py
      deepseek.py
      gemini.py
  core/                     ← lógica de negócio
      grader.py            ← identifica arquivos por questão, monta o prompt, chama a LLM
      utils.py
  <pasta_da_turma>/         ← submissões baixadas do Moodle VPL
      Nome estudante - login/
          <timestamp>/               ← última submissão do estudante
              Q1.py                  ← padrão para provas com múltiplas questões
              Q2.py
              rubrica.txt            ← gerado pela LLM (somente se avaliação válida)
```

```
        <timestamp>.ceg/
            execution.txt            ← nota/objetiva atribuída pelo VPL
    ...
```

O sistema aceita três provedores de LLM — **Groq**, **DeepSeek** e **Gemini** — configuráveis em `config.yaml` por meio da chave `llm.provider`. Cada provedor pode ter **múltiplos modelos** cadastrados; a cada chamada, a lista é **embaralhada**, distribuindo a carga entre os estudantes processados em paralelo e reduzindo a chance de erros de limite de requisições (HTTP 429). Quando um modelo falha, o sistema tenta até três vezes antes de passar ao próximo da lista, respeitando o tempo de espera sugerido pelo provedor. Erros de autenticação (401) ou de saldo insuficiente (402) interrompem imediatamente a tentativa naquele provedor.

Um aspecto importante do projeto diz respeito à privacidade dos dados. **Nome, e-mail, login, RA e CPF do estudante nunca são enviados à API do LLM**. São transmitidos para processamento apenas: o nome do arquivo contendo o código-fonte; o próprio código (com os comentários removidos); o *prompt* da rubrica — que não contém dados pessoais —; e, quando disponíveis no `execution.txt` gerado pelo VPL, o enunciado oficial da questão sorteada para aquele estudante e o resultado bruto dos casos de teste executados (entrada, saída produzida pelo código e saída esperada). Esses dois últimos itens, embora extraídos de um arquivo específico de cada estudante, também não contêm identificação pessoal — apenas o texto da questão e os valores de entrada/saída dos testes automáticos —, e são usados como evidência objetiva para reduzir o risco de a IA identificar incorretamente o tipo de questão ou avaliar a lógica do código apenas por leitura estática (Seção D.3).

Ainda assim, é importante reconhecer que os três provedores atualmente suportados — Groq, DeepSeek e Gemini — são serviços comerciais operados por empresas estrangeiras, cujos modelos são executados em infraestrutura fora do país e sujeitos a políticas de uso, disponibilidade e custo que fogem ao controle da instituição de ensino. Essa dependência de provedores externos traz riscos que vão além da correção de código pontual: mudanças de preço, descontinuação de modelos, indisponibilidade temporária do serviço, alterações unilaterais nos termos de uso e, em casos mais sensíveis, restrições geopolíticas de acesso podem comprometer a continuidade de um sistema de avaliação que passe a depender estruturalmente dessas APIs. Por isso, é estratégico que Instituições de Ensino Superior (IES) como a UFABC invistam no desenvolvimento e na hospedagem de **provedores próprios de IA** — por exemplo, modelos abertos (*open-weight*) executados em infraestrutura local ou em nuvem soberana —, reduzindo a dependência de serviços externos, especialmente de outros países, e ampliando o controle institucional sobre custos, disponibilidade, privacidade dos dados de estudantes e continuidade pedagógica de longo prazo. O desenho do **vpl-ai-feedback** já favorece esse caminho: a arquitetura de `providers/` foi construída como uma fábrica (*factory*) extensível, na qual um novo provedor — inclusive um modelo hospedado localmente pela própria instituição, com interface compatível — pode ser adicionado com baixo esforço de integração.

E.3 O *prompt* universal e a rubrica em duas etapas

Cada tipo de prova é descrito em um arquivo de *prompt* (por exemplo, `Simulado4.txt`), referenciado em `grading.prompt_file` no `config.yaml`. Esse arquivo instrui a LLM a realizar a avaliação em **duas etapas**:

1. **Identificação da questão** — a LLM determina o tipo específico de operação sorteado para aquele estudante (útil quando há questões paramétricas sorteadas e embaralhadas por estudante, como descrito no Apêndice A);
2. **Aplicação da rubrica correspondente** — a LLM avalia o código segundo critérios pontuados, cada um com uma faixa máxima de pontos, retornando ao final uma nota no formato `NOTA FINAL: X/PESO`.

Em provas parametrizadas, o arquivo de *prompt* costuma conter **várias rubricas paralelas**, uma para cada tipo de questão possível, delimitadas por marcadores como `[START_RUBRICA_TIPO_A]` / `[END_RUBRICA_TIPO_A]`. A LLM identifica primeiro qual tipo foi sorteado para aquele estudante e aplica **apenas** a rubrica correspondente, ignorando as demais — o que evita que critérios de um tipo diferente contaminem a nota.

Quando a prova possui mais de uma questão por atividade VPL, os arquivos de código devem seguir o

padrão `Q1.*`, `Q2.*` etc. — um arquivo por questão, com extensão livre conforme a linguagem escolhida pelo estudante —, e cada questão tem peso configurável individualmente em `grading.weights` no `config.yaml`. Quando a prova tem uma única questão e não foi gerada pelo MCTest, o sistema aceita qualquer nome de arquivo com extensão suportada, desde que seja o único arquivo de código presente na pasta de submissão.

No caso do Simulado 4 (tópico *Operadores Morfológicos*), por exemplo, um dos tipos possíveis definia três critérios, com peso total de 100 pontos para aquela questão:

Critério	Descrição	Pontuação máxima
1 — Entrada de dados	Leitura de `H`, `W` e da matriz binária (`mm.readImg`)	25 pts
2 — Processamento morfológico	Abertura + Fechamento (`mm.sebox()`), `mm.measure`, ordenação por `bbox[0]`/`bbox[1]` e reatribuição de IDs	50 pts
3 — Saída e formatação	Exibição da imagem limpa (`mm.drawImg`) e tabela de medidas formatada	25 pts

O *prompt* exige ainda um **formato de resposta obrigatório** (largura de linha, ordem das seções, marcação `NOTA FINAL:`), o que torna a extração automática da nota e a montagem do relatório consolidado mais confiáveis — embora, como discutido na Seção D.8, isso não elimine o risco de erro semântico na avaliação do conteúdo.

E.4 Duas camadas adicionais de evidência

Para reduzir o risco de a LLM alucinar o tipo de questão ou a corretude do código — o risco central discutido na Seção D.7 —, o **vpl-ai-feedback** passou a fornecer à IA duas fontes de evidência objetiva, extraídas automaticamente do próprio `execution.txt` gerado pelo VPL, além do código-fonte:

- **Enunciado oficial da questão.** Como as questões costumam ser sorteadas e embaralhadas por estudante no MCTest (Apêndice A), o `execution.txt` de cada estudante já traz, no campo "Descrição resumida da questão", o texto exato da questão que caiu para ele. O sistema extrai esse trecho automaticamente e o envia à LLM como evidência **primária** para identificar o tipo/variação da questão — mais confiável do que inferir apenas pela leitura estática do código, especialmente em submissões incompletas ou ambíguas entre dois tipos próximos (por exemplo, erosão "plana" *versus* "com pesos"). Quando o código entregue diverge do que o enunciado pede, o sistema não ignora o enunciado nem "corrige" a leitura silenciosamente: mantém a identificação pelo enunciado, reduz a confiança da avaliação para "Baixa" e explicita a divergência no *feedback* enviado ao estudante.
- **Resultado real da execução no VPL.** Quando disponível, o sistema também envia à LLM os casos de teste efetivamente executados sobre aquele código — entrada, saída produzida e saída esperada —, como evidência objetiva de corretude, a ser usada para confirmar ou refutar a leitura da lógica antes de pontuar os critérios de processamento e de saída, em vez de a IA depender apenas de simular mentalmente o código (o que é especialmente falho em laços com `min`/`max`, deslocamento de índice ou cálculo de limiar de Otsu, onde erros sutis passam despercebidos numa leitura estática).

Além disso, a LLM é instruída a declarar explicitamente seu **nível de confiança** (Alta, Média ou Baixa) na identificação de cada tipo de questão, com justificativa quando a confiança não for Alta. Esse valor é extraído automaticamente e alimenta uma coluna `Revisar_Manualmente` no relatório CSV consolidado (Seção D.4), permitindo ao professor priorizar exatamente os casos em que a própria IA reconhece maior incerteza — em vez de tratar todas as avaliações de uma turma como igualmente confiáveis.

E.5 Fluxo de execução

A execução típica, feita por `bash run.sh config.yaml`, segue os passos:

1. Carrega `config.yaml` e localiza a pasta de submissões (`paths.student_base_dir`);
2. Para cada estudante, identifica a **última submissão** (pasta com o *timestamp* mais recente);
3. Extrai a nota objetiva do Moodle a partir de `*.ceg/execution.txt`;
4. Para cada questão da prova, localiza o arquivo de código (`Q1.*`, `Q2.*`, ou o único arquivo, no caso de prova que não foi gerada pelo MCTest), monta o *prompt* (código + rubrica) e envia a um modelo sorteado da lista configurada;
5. Processa todos os estudantes **em paralelo**, com controle de concorrência;
6. Gera, por estudante, o arquivo `rubrica.txt` — **somente se a IA retornar ao menos uma avaliação válida** para as questões que possuem código; caso contrário, o arquivo não é salvo, forçando nova tentativa na próxima execução;
7. Consolida todos os `rubrica.txt` em um único `*_ALL.txt` e gera um relatório `*_relatorio.csv`, comparando nota do Moodle, nota da IA e a diferença entre ambas, por questão e no total.

Situação	`rubrica.txt` é salvo?
Todas as questões com código avaliadas com sucesso	✅ Sim
IA falhou em ao menos uma questão com código	❌ Não — reprocessa na próxima execução
estudante sem nenhum arquivo de código enviado	❌ Não
Tipo de questão identificado como DESCONHECIDO	❌ Não

E.6 Exemplo ilustrativo: rubrica de uma prova com três questões

i Sobre a origem deste exemplo

O trecho a seguir **não reproduz o código de nenhum estudante específico**. Trata-se de um exemplo reconstruído pelo autor, que reproduz um *padrão* de erro observado repetidamente ao longo das turmas de PDI-VC — a confusão entre uma função morfológica 2D (`mm.ero`/`mm.ero0`) e uma operação 1D sobre sinais — sem citar ou identificar qualquer submissão real. Essa opção evita qualquer risco de violação do direito autoral do estudante sobre seu próprio código-fonte, ainda que anonimizado, já que a titularidade da obra permanece do autor original mesmo quando nome e *login* são removidos.

O trecho a seguir ilustra a saída gerada pelo **vpl-ai-feedback** para um estudante de uma prova com **três questões** (Q1, Q2, Q3), usando o modelo `deepseek-chat`. O resumo comparativo aparece no topo do arquivo, seguido do enunciado oficial de cada questão, do código submetido e da avaliação por critério.

```
RESUMO - IA (deepseek-chat)  x  MOODLE

Peso 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

Diferença (IA - Moodle): -23.7 pts

Q1: tipo identificado = B  | confiança = Baixa (código não
```

```
  corresponde ao enunciado da questão)  REVISAR
Q2: tipo identificado = A  | confiança = Alta (enunciado oficial
  confirma o tipo e a variação)
Q3: tipo identificado = C1 | confiança = Alta (enunciado oficial
  confirma erosão 2D com OpenCV integrada)
```

Q1 — código não corresponde ao enunciado (confiança baixa): o enunciado oficial da questão, extraído do `execution.txt`, pedia uma erosão morfológica **1D** sobre um sinal (leitura de N, M, o sinal e o elemento estruturante B, seguida do mínimo dos vizinhos ativos). O código submetido, no entanto, lê apenas dois inteiros e invoca `mm.ero0` — uma função de erosão **2D** para imagens, não relacionada à operação pedida.

```
Tipo identificado: TIPO B (Erosão 1D - plana)
Confiança: Baixa - código não corresponde ao enunciado da questão (usa
mm.ero0, função de erosão 2D, em vez de implementar erosão 1D plana
sobre um sinal)

Critério 1 - [3.33/10.00 pts]:
- O código lê dois inteiros em uma única linha (que corresponderiam a
  N e M), mas não lê o sinal nem o elemento estruturante B como vetores
  1D. Em vez disso, trata ambos como imagens via mm.readImg, o que não
  corresponde à leitura esperada de um sinal 1D e de B em linhas
  separadas.

Critério 2 - [0.00/16.67 pts]:
- A operação aplicada é mm.ero0(img, element), erosão 2D para imagens,
  não a erosão 1D de sinais pedida no enunciado. A lógica de mínimo dos
  vizinhos ativos com bordas por recorte não está implementada.

Critério 3 - [0.00/6.67 pts]:
- A saída é uma imagem 2D via mm.drawImg, não o sinal 1D resultante em
  uma única linha com valores inteiros separados por espaço, como
  exigido.

NOTA FINAL: 3.33/33.33

Comentários para o estudante: O código apresentado não corresponde ao que o
enunciado pede. A questão sorteada solicita uma erosão morfológica 1D
sobre um sinal unidimensional; o código usa uma função de erosão 2D
para imagens. Releia atentamente o enunciado sorteado antes de escolher
a implementação.
```

Q2 — erro de sintaxe impede a execução (confiança alta, nota baixa por falha real): aqui a IA identificou corretamente o tipo (limiarização com Otsu), com apoio do enunciado oficial, mas o código continha um erro de indentação que impedia a execução — evidenciado tanto pela leitura estática quanto pela ausência de saída válida nos testes do VPL.

```
Tipo identificado: TIPO A (Limiarização - T manual + Otsu)
Confiança: Alta - enunciado oficial confirma o tipo e a variação

Critério 1 - [6.67/10.00 pts]:
- Lê corretamente os três parâmetros L, C e T em linhas separadas e a
  imagem de entrada. Porém, a definição de uma função auxiliar de
  limiarização está indentada incorretamente dentro do escopo global,
  causando um erro de indentação que impede a execução completa do
  programa.

Critério 2 - [0.00/16.67 pts]:
- A binarização com limiar fixo T está corretamente estruturada, mas o
  cálculo do limiar de Otsu depende da função auxiliar malformada, que
```

```
  referencia uma variável fora de escopo. O erro de indentação impede
  qualquer processamento.

Critério 3 - [0.00/6.67 pts]:
- Devido ao erro de indentação, nenhuma saída é produzida.

NOTA FINAL: 6.67/33.33

Comentários para o estudante: A estrutura geral está correta (leitura de L,
C, T e da imagem), mas há um erro de indentação que impede a execução.
Revise a sintaxe Python antes de submeter e teste localmente o código
antes do envio ao VPL.
```

Q3 — divergência de função, mas equivalência funcional confirmada por evidência real (confiança alta, nota justa): o enunciado pedia explicitamente o uso de `mm.ero` (versão integrada ao OpenCV), mas o código usa `mm.ero0` (versão plana implementada manualmente). Como o `execution.txt` mostrava os 10 casos de teste do VPL passando integralmente, a IA não penalizou a divergência de função — reconheceu a equivalência funcional demonstrada pelos testes reais, mas ainda assim registrou a observação no *feedback*.

```
Tipo identificado: TIPO C1 (Erosão 2D - plana sem pesos)
Confiança: Alta - enunciado oficial confirma erosão 2D com OpenCV
integrada (mm.ero), e o código implementa erosão plana via mm.ero0

Critério 1 - [10.00/10.00 pts]:
- Lê corretamente altura, largura, a matriz da imagem e as dimensões e
  valores do elemento estruturante B.

Critério 2 - [16.67/16.67 pts]:
- O enunciado pedia o uso de mm.ero (OpenCV integrada), mas o código
  usa mm.ero0 (plana sem pesos). Apesar da divergência de função, a
  operação foi implementada corretamente com o elemento estruturante
  lido: os 10 testes executados no VPL passaram integralmente (10/10),
  confirmando equivalência funcional para os casos testados.

Critério 3 - [6.67/6.67 pts]:
- Imprime a imagem resultante no formato correto; os testes confirmaram
  a saída correta.

NOTA FINAL: 33.34/33.33

Comentários para o estudante: O código está correto e funcional, com todos
os 10 testes passando. Observação: o enunciado pedia explicitamente
mm.ero (versão OpenCV), mas você usou mm.ero0 (versão plana). Embora os
resultados tenham sido equivalentes nos testes, siga exatamente a
função solicitada no enunciado em provas futuras.
```

Note que, nesse caso composto, a IA atribuiu **23,7 pontos a menos** do que a nota objetiva do VPL, puxada principalmente pela Q1. A diferença agregada por si só já seria um sinal para investigação (Seção D.7), mas o próprio sistema entrega ao professor a causa provável de cada questão: a coluna `Revisar_Manualmente` do CSV é marcada "SIM" sempre que ao menos uma questão do estudante recebe confiança baixa, com o motivo específico registrado na coluna de confiança correspondente — reduzindo o tempo que o professor precisa gastar para localizar, entre uma centena de estudantes, os poucos casos que realmente merecem atenção manual antes da atribuição da nota oficial. O caso da Q3, por sua vez, ilustra o cuidado inverso: a divergência entre enunciado e código **não** resultou em penalização indevida, porque a evidência real de execução confirmou a equivalência funcional — exatamente o comportamento que a Seção D.3 descreve para essas duas camadas de evidência.

E.7 Envio do *feedback* por e-mail

Após a geração das rubricas, o *script* `enviar_email.py` envia a cada estudante um e-mail com o `rubrica.txt` em anexo. O corpo do e-mail é definido em `config.yaml`, em `templates.corpo`, e é interpolado com `{nome_pasta}` e `{login}`:

```yaml
email:
  smtp_server: smtp.ufabc.edu.br
  smtp_port: 587
  from_address: professor@ufabc.edu.br
  password: "SUA_SENHA_AQUI"     # nunca versionar credenciais reais
  use_tls: true

templates:
  assunto: "Rubricas e Correções geradas por LLM - Simulado - {login}@aluno.ufabc.edu.br"
  corpo: |
    Prezado(a) {nome_pasta},
    ...
```

! Segurança de credenciais

O `config.yaml` contém segredos reais (senha de SMTP, chaves de API). Ele deve constar do `.gitignore` do projeto e **jamais** ser publicado, compartilhado ou versionado — inclusive em anexos de e-mail, capturas de tela ou repositórios públicos.

Um exemplo real de e-mail enviado a um estudante (dados **anonimizados**, com nome e login substituídos por um caso fictício) é reproduzido a seguir, para ilustrar o tom didático e as ressalvas explicitadas ao estudante:

! Exemplo de mensagem individual (anonimizada)

Prezado(a) [Nome do Estudante] - [login],
A sua nota do Simulado 4 está disponível no Moodle.
Envio abaixo as competências avaliadas e uma correção detalhada do seu código, gerada automaticamente por Inteligência Artificial (IA).
No anexo "rubrica.txt" está o retorno completo da IA (recomendo baixar e abrir com um editor de texto ou bloco de notas).
Ressalto que essa correção gerada por IA PODE conter imprecisões ou erros, mas serve como um excelente apoio ao seu processo de aprendizagem na disciplina.
Uma sugestão pedagógica é submeter os seus códigos (contidos neste arquivo), juntamente com a RUBRICA abaixo, a outros modelos de LLM para comparação. Isso pode ajudar a identificar possíveis divergências, além de oferecer diferentes perspectivas sobre o seu código e sobre os critérios de avaliação.
Caso tenha dúvidas ou note alguma inconsistência gritante na correção apresentada no Moodle, estou à disposição para esclarecimentos.
Atenciosamente,
Prof. Francisco Zampirolli
PS.: Explicando o processo: a última versão do seu código salva no Moodle foi anexada ao prompt e enviada para um dos LLMs escolhidos de forma integrada ao nosso sistema de avaliação (modelos = ("deepseek-chat")). O processo é repetido até que seja obtida uma resposta com nota válida (entre 0 e 100).

Três elementos didáticos merecem destaque nesse texto: (i) o alerta explícito de que a correção **pode conter erros**; (ii) o convite para que o próprio estudante **contraste a avaliação com outros LLMs**, transformando a IA em ferramenta de estudo ativo em vez de veredito passivo; e (iii) a explicação transparente do processo automatizado, incluindo o(s) modelo(s) usado(s) e a política de reprocessamento até obter uma nota válida.

E.8 Riscos, limites éticos e responsabilidade docente

! O professor não pode simplesmente adotar a nota da IA

Este é o ponto central deste apêndice: **em nenhuma hipótese a nota sugerida pela IA deve ser atribuída automaticamente ao estudante como nota oficial**. Modelos de linguagem podem alucinar — atribuir pontos a critérios não atendidos, "inventar" que uma função foi chamada corretamente quando não foi, ou penalizar um código correto por má interpretação da rubrica. A nota final de qualquer atividade avaliativa deve **sempre** resultar de avaliação manual do professor (ou da correção objetiva por casos de teste, como no VPL/MCTest), com a IA cumprindo, no máximo, o papel de **primeira leitura** ou de **material de apoio ao estudante**, nunca de instância decisória.

Alguns cuidados práticos adotados no uso do **vpl-ai-feedback**, e recomendados a qualquer professor que deseje adaptar o sistema:

- **Separação clara entre nota oficial e *feedback* de IA.** No caso relatado, a nota do bônus dos simulados sempre veio do VPL, nunca da IA (Seção D.1). A rubrica de IA foi comunicada ao estudante como material de apoio, com aviso explícito de que pode conter erros.
- **Transparência com o estudante.** O e-mail enviado (Seção D.6) explicita que a avaliação foi gerada por IA, informa o(s) modelo(s) utilizados e convida o estudante a comparar com outros LLMs — reduzindo a assimetria de informação e incentivando o pensamento crítico sobre a própria correção recebida.
- **Reprocessamento em vez de *cache* inválido.** O sistema só grava `rubrica.txt` quando obtém avaliação válida (Seção D.4); isso evita que uma resposta malformada, incompleta ou visivelmente inconsistente da IA seja apresentada ao estudante como se fosse definitiva.
- **Monitoramento das divergências.** O relatório CSV consolidado (coluna `Diferenca = Total_IA - Total_Moodle`) permite ao professor identificar rapidamente os casos em que IA e correção objetiva mais discordam — como no exemplo da Seção D.5 (+20 pontos) —, priorizando esses casos para revisão manual.
- **Canal aberto para contestação.** O e-mail final oferece explicitamente ao estudante a possibilidade de reportar inconsistências, mantendo o professor como autoridade final sobre a nota.
- **Dados pessoais fora do *prompt*.** Como descrito na Seção D.2, nome, e-mail, login, RA e CPF nunca são enviados à API do LLM, reduzindo riscos de privacidade no uso de provedores externos de IA.

E.9 Considerações finais

O **vpl-ai-feedback** mostra como a IA pode enriquecer o ciclo de *feedback* em disciplinas de programação com alto volume de submissões, oferecendo a cada estudante uma leitura qualitativa e individualizada do próprio código — algo inviável de se fazer manualmente para uma centena de estudantes, em múltiplas questões e provas ao longo do quadrimestre. A pesquisa de opinião citada na Seção D.1 sugere que esse tipo de retorno é bem recebido pelos estudantes. Ainda assim, o uso responsável do sistema depende de um princípio inegociável, reiterado ao longo deste apêndice: **a IA complementa, mas não substitui, a avaliação do professor**, e a nota oficial de qualquer atividade avaliativa deve continuar sendo definida por correção objetiva (VPL/MCTest, Apêndices A e B) e/ou julgamento humano — nunca pela nota bruta devolvida por um modelo de linguagem.

IL PERCORSO DI EDI & VC

SISTEMATIZZARE LA CONOSCENZA, DAL PIXEL ALL'INTELLIGENZA

Parte I — Fondamenti di EDI

- 1. Fondamenti e Primi Passi
- 2. Campionamento, Quantizzazione e Connettività
- 3. Operazioni Spaziali e Filtraggio
- 4. Morfologia Matematica e Segmentazione
- 5. Trasformate e Compressione

Parte II — Visione Computazionale

- 6. Analisi e Ispezione dei Documenti
- 7. Classificazione e Riconoscimento dei Pattern
- 8. Comprensione delle Scene e Segmentazione
- 9. Deep Learning (Apprendimento Profondo)

Appendici Essenziali

- **A:** MCTest
- **B:** Moodle/VPL
- **C:** SEB/Simulatore
- **D:** Materiale Didattico su Gemini
- **E:** IA nel feedback del VPL
- **F:** Percezione e Valutazione

IMMAGINE DI INPUT (PIXEL) → **EDI** (ELABORAZIONE) → **VC** (VISIONE) → OUTPUT INTELLIGENTE (SEMANTICO)

DOMINARE I PIXEL, COSTRUIRE IL FUTURO.

Dai semplici istogrammi alla comprensione complessa delle scene con il deep learning, questo lavoro offre una roadmap completa.

Trasforma le immagini in dati e i dati in decisioni intelligenti.

La capacità di plasmare la percezione computazionale inizia qui. Rimani al passo, esplora e innova!

Disponibile in formato digitale (HTML, PDF e Notebook Jupyter), con simulazioni ed esercizi, inclusi casi di test con VPL e Moodle.

SCANSIONA E ACCEDI AI CONTENUTI EXTRA

AUTORE: FRANCESCO DE ASSIS ZAMPIROLLI

https://fzampirolli.github.io/edi-vc/