PDI &VC

Procesamiento Digital de Imágenes y Visión por Computadora

Fundamentos, Algoritmos y Aplicaciones Integradas

AUTOR: FRANCISCO DE ASSIS ZAMPIROLLI

Procesamiento Digital de Imágenes y Visión por Computador

Libro interactivo con C++

Francisco de Assis Zampirolli

Universidade Federal do ABC

20 de septiembre de 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]

Índice

Lista de Figuras

Lista de Tablas

PDI y VC — C++

Bienvenido al libro PDI y Visión por Computador — versión C++ / Español.

Organización

- **Parte I — Fundamentos de PDI** — Representación, histogramas, filtrado, morfología
- **Parte II — Visión por Computador** — Segmentación, descriptores, detección, aprendizaje profundo

---

Prefacio

Este libro constituye una **obra en permanente actualización** en las áreas de **Procesamiento Digital de Imágenes (PDI)** y **Visión Computacional (VC)**, concebida como material didáctico interactivo para cursos de grado y posgrado en Computación, Ingeniería y áreas afines.

> **!** Destacado
>
> La obra se fundamenta en la metodología descrita en Zampirolli et al. (2025) — extensión de Zampirolli et al. (2024), trabajo premiado en la línea **Recursos y Ambientes Educativos** de la **EduComp 2024**. El contenido integra la biblioteca `morph.py`, desarrollada por el autor.

Este no es un libro estático. Su contenido evoluciona continuamente a medida que los ejemplos se mejoran, se incorporan nuevas secciones y los enfoques pedagógicos se refinan con base en la experiencia de uso y en la retroalimentación de estudiantes y docentes. De esta manera, la obra se trata como un *proyecto en constante evolución*, buscando acompañar tanto los avances tecnológicos como las mejores prácticas de enseñanza en PDI y VC.

Contexto de la primera edición

El contenido de esta edición fue elaborado entre mayo y agosto de 2026, durante la primera oferta de la asignatura de Procesamiento Digital de Imágenes basada en este material didáctico. La asignatura se impartió en dos grupos de grado —mayoritariamente compuestos por estudiantes de Ciencias de la Computación, en el período matutino— y en un grupo de posgrado en Ciencias de la Computación, todas en la UFABC. Esta primera edición representa el resultado de esa oferta inicial, consolidando el contenido desarrollado, probado y continuamente perfeccionado a lo largo del período lectivo.

La metodología de enseñanza adoptada siguió una secuencia estructurada en cada clase. Inicialmente, se proyectaba un video corto, con una duración media de 7 minutos, generado en **Gemini Notebook**, alternando entre la presentación de la parte conceptual del capítulo y la resolución de los Ejercicios de Programación (EP). En este segundo caso, casi todos los EP se resolvían durante la propia clase. A continuación, se presentaba un conjunto de aproximadamente 12 *diapositivas*, también generadas en **Gemini Notebook**, teniendo como fuentes el PDF completo del libro y el archivo `morph.py`. Estas *diapositivas* se producían a partir de un *prompt* específico para la generación del contenido teórico y práctico de cada capítulo.

Tras esta etapa inicial, quedaban aproximadamente 100 minutos de clase para la exploración de los dos *cuadernos* Colab del capítulo, uno teórico y otro práctico, con énfasis en los simuladores interactivos y en la ejecución de los bloques de código, permitiendo a los estudiantes modificar parámetros y observar sus efectos. En las clases realizadas en laboratorio, los estudiantes transferían las respuestas de los EP desarrolladas en Colab directamente a las actividades VPL de Moodle, en las cuales se sometían a la evaluación automática. Este proceso de publicación y utilización de los EP se presenta al final de este prefacio.

La evaluación continua incluyó simulacros quincenales, aplicados con **SEB (*Safe Exam Browser*)** en Moodle, que contenían EP similares a los de las listas de ejercicios. Cada simulacro otorgaba un bono del 5% sobre la nota final. Las dos pruebas de la asignatura también utilizaron SEB, pero estuvieron compuestas por preguntas parametrizadas generadas en **MCTest**, cuya descripción combina LaTeX y parámetros en el formato `[[code:variable]]`, definidos en fragmentos de Python delimitados por `[[def: ... ]]` en la propia pregunta. Este proceso permitió generar 110 variaciones de examen, sorteadas individualmente entre los estudiantes.

El Apéndice A detalla la creación de estas preguntas en MCTest y su exportación a Moodle; el Apéndice B describe la configuración de las actividades VPL, incluido el proceso de publicación de los EP; el Apéndice C presenta la configuración de SEB para los simulacros y los exámenes; el Apéndice D describe la generación de los videos y *diapositivas* utilizados en las clases con **Gemini Notebook**; y el Apéndice E detalla el uso de **Inteligencia Artificial (IA)** en la generación de *retroalimentación* socrática para actividades formativas y evaluativas, complementando la corrección automática realizada por VPL.

Cómo se produce este libro

El contenido de este libro se desarrolla en **Quarto** y se almacena en la carpeta `all` del repositorio **github.com/fzampirolli/pdi-vc**. A partir de un único código fuente, el libro se genera automáticamente y se publica en diferentes formatos, presentados en Tabla 1.

Aunque la edición en PDF registra el estado de la obra al término de la primera oferta de la disciplina en 2026, el desarrollo del libro continúa de forma permanente. Con cada actualización del repositorio de GitHub, se generan automáticamente nuevas versiones de las páginas HTML, del PDF y de los notebooks, poniendo las mejoras inmediatamente a disposición de los lectores.

Tabla 1: Diferentes formatos de publicación generados automáticamente a partir del mismo código fuente.

Formato	Descripción
HTML	Versión para *web*, con simuladores interactivos y navegación entre capítulos: fzampirolli.github.io/pdi-vc/
PDF	Versión adecuada para impresión o lectura *offline*: livro.pt.py.pdf
Notebooks (`.ipynb`)	Compatibles con **Jupyter** y **Google Colab**, que permiten ejecutar, modificar y experimentar con los ejemplos de código y los Ejercicios de Programación (EPs). Un filtro personalizado mantiene las referencias cruzadas, la numeración de figuras y tablas, además de formatear automáticamente las citas según el estándar ABNT.

La obra se publica en **diez combinaciones** — cada ruta de código (**Python** y **C++**) en cinco idiomas (portugués, inglés, francés, español e italiano). Con la caché de traducción ya poblada, una regeneración completa (traducción, reejecución de los *notebooks*, renderizado de HTML y PDF y publicación) tarda unos **72 minutos**, con un paralelismo real medio de ~5 procesos (pico de 8); la **primera** generación de un idioma nuevo, con la caché vacía, es mucho más lenta — unos **80 minutos** por combinación, como se observó en las primeras generaciones de español e italiano. Tabla 2 resume cada combinación (ya con la caché poblada): número de páginas del PDF y tiempo de renderizado. Este tiempo total real es mucho menor que la suma de los tiempos de cada combinación por separado (más de **5 horas** si se ejecutaran una por una): en una CPU con muchos núcleos, varias combinaciones se procesan al mismo tiempo, por lo que el tiempo total no es la simple suma de los tiempos individuales.

Tabla 2: Número de páginas del PDF y tiempo de renderizado por combinación (paralelismo real medio de ~5 procesos, pico de 8). La combinación base en Python/Portugués solo renderiza las salidas ya registradas en los *notebooks*-fuente; las demás reejecutan todo el código.

Combinación	Ruta	Idioma	Páginas	Tiempo de render.
`py.pt`	Python	Portugués (base)	654	~7 min
`py.en`	Python	Inglés	644	~31 min
`py.fr`	Python	Francés	666	~31 min
`py.es`	Python	Español	666	~31 min
`py.it`	Python	Italiano	658	~31 min
`cpp.pt`	C++	Portugués	434	~26 min
`cpp.en`	C++	Inglés	426	~34 min

Combinación	Ruta	Idioma	Páginas	Tiempo de render.
`cpp.fr`	C++	Francés	434	~38 min
`cpp.es`	C++	Español	430	~37 min
`cpp.it`	C++	Italiano	428	~35 min

Estado de validación. Solo la combinación `py.pt` (Python, portugués) es la fuente curada: es donde el autor escribe, revisa y valida todo el contenido. Las otras nueve combinaciones — las rutas en **C++** y las traducciones al inglés, francés, español e italiano — se **generan automáticamente**, mediante traducción con un modelo de lenguaje y transpilación de código, y todavía **requieren una revisión detallada**. El punto más sensible es el **texto incrustado en algunas figuras**: donde aún no se ha producido la versión traducida, la imagen aparece con texto en portugués (cada figura traducida sigue el mismo sufijo de idioma que los demás archivos generados, por ejemplo `imagen_en.png`).

Las páginas HTML, el PDF y los *notebooks* disponibles en el proyecto reflejan siempre el estado más reciente del desarrollo. Cuando se publica una **edición oficial**, esta queda identificada mediante una *etiqueta* (tag) en el repositorio de GitHub, lo que permite reproducir exactamente el contenido correspondiente a esa edición. Así, el lector puede seguir tanto la evolución continua del proyecto como recuperar cualquier edición oficial publicada anteriormente.

```
git clone https://github.com/fzampirolli/pdi-vc.git
cd pdi-vc
git checkout <tag-de-la-version>
```

Cada capítulo dispone de dos botones **Ejecutar en Colab** (Figura 1), que dirigen a entornos complementarios:

1. **Parte Teórica**, ubicada al inicio de cada capítulo, que contiene los conceptos presentados y ejemplos de código ejecutables;
2. **Parte Práctica**, en la sección **Parte Práctica con Ejercicios de Programación** (EPs), dedicada a los ejercicios y a sus simuladores interactivos.

Figura 1: Botón clicable **Ejecutar en Colab**, disponible al inicio de las partes **Teórica** y **Práctica** de cada capítulo, que permite la ejecución interactiva de los ejemplos y ejercicios. En la versión PDF, existe un *enlace* HTML directo entre la imagen del simulador y su leyenda.

La generación de los *notebooks* está totalmente automatizada mediante el *script* `gerar_notebooks_alunos.py`, que adapta el contenido a entornos interactivos, permitiendo su ejecución sin necesidad de instalar Quarto.

Uso de herramientas de Inteligencia Artificial

La concepción, el proyecto pedagógico, la estructura conceptual y el contenido fundamental de este libro son de **autoría exclusiva del autor**.

En el proceso de edición y apoyo al desarrollo, se utilizaron, en sus versiones gratuitas, herramientas de **Inteligencia Artificial (IA)**, como **ChatGPT**, **Claude**, **DeepSeek** y **Gemini**, empleadas estrictamente como recursos auxiliares. Su uso se limitó al apoyo en la revisión y mejora del estilo textual, a la optimización de la sintaxis de códigos y a la generación asistida de ilustraciones conceptuales y ejemplos.

Todas las respuestas y contenidos sugeridos por estas herramientas fueron sometidos a **análisis crítico, verificación, validación técnica, adaptación e integración por parte del autor**, quien asume la responsabilidad por el texto, los códigos y los recursos pedagógicos presentados. Las herramientas de IA **no se consideran autoras ni coautoras de la obra**, ni responsables de las decisiones intelectuales, técnicas o pedagógicas que fundamentan su contenido.

Se destaca, además, que los **simuladores** y recursos interactivos presentes en el libro —disponibles en las versiones HTML y Jupyter Notebook (IPYNB)— fueron diseñados e implementados como parte del enfoque pedagógico de la obra, con el objetivo de proporcionar experimentación directa de los conceptos presentados y favorecer la autonomía de aprendizaje del lector.

A diferencia del uso editorial descrito anteriormente, el *pipeline* de generación multilingüe (ver "Cómo se produce este libro") emplea un modelo de lenguaje como **componente de ingeniería del sistema**: la traducción automática del contenido entre portugués y otros idiomas, con caché incremental que evita retraducir fragmentos sin cambios. Esta etapa utiliza la API de pago de **DeepSeek** (modelo `deepseek-v4-flash`); el libro completo ya ha sido traducido del portugués al **inglés**, al **francés**, al **español** y al **italiano**, y los **Capítulos 1 a 5** cuentan además con una ruta en **C++** que compila y ejecuta de verdad — a un costo acumulado del orden de unos pocos dólares, gracias a la caché incremental.

La biblioteca `morph.hpp`

La biblioteca **`morph.hpp`** (Zampirolli et al., 2025) acompaña toda la obra como una herramienta de apoyo a la enseñanza. Su objetivo no es sustituir bibliotecas consolidadas, como **OpenCV**, ni competir con ellas en rendimiento o alcance. En cambio, busca **hacer transparentes los algoritmos fundamentales de Procesamiento Digital de Imágenes y Visión Computacional (PDI-VC)**, permitiendo que el estudiante comprenda su implementación, modifique el código y desarrolle nuevas funcionalidades.

Es el equivalente en C++ de `morph.py`: *header-only* (basta un `#include "morph.hpp"`), con los **mismos nombres de función** en el *namespace* `mm::` — quien ya conoce `mm.dil()`, `mm.dil0()` o `mm.gradm()` en Python reconoce de inmediato `mm::dil()`, `mm::dil0()` y `mm::gradm()` en C++. Una divergencia de nombre entre las dos versiones se considera un defecto de la biblioteca, no una decisión de diseño.

> **Herencia Técnica**
>
> La estructura de `morph.hpp` (igual que su contraparte `morph.py`) tiene como base herramientas de Morfología Matemática desarrolladas en Brasil, como **MMachLib** (Lotufo et al., 1997) y **MMach** (Barrera et al., 1998), además de la **mmorph**, utilizada en la obra de Dougherty; Lotufo (2003). Estas bibliotecas sirvieron de referencia para la enseñanza del área durante décadas.

Esta filosofía puede observarse, por ejemplo, en los operadores morfológicos de dilatación:

- `mm::dil0`: implementación didáctica de la dilatación para **elementos estructurantes planares**, siguiendo directamente la definición clásica de la Morfología Matemática;
- `mm::dil1`: implementación didáctica de la dilatación para **funciones estructurantes (*kernels* no planares)**, siguiendo rigurosamente su formulación matemática;
- `mm::dil`: por defecto, delega en `mm::dil0()` — la versión didáctica planar, numéricamente equivalente a `cv::dilate` para ese tipo de elemento estructurante. La implementación optimizada, basada en **OpenCV** (`cv::dilate`), solo entra en acción si el programa se compila con la opción `-DMM_USE_OPENCV`.

> **Una diferencia deliberada respecto a `morph.py`**
>
> En Python, `mm.dil()` (sin sufijo) **ya es** la implementación optimizada por defecto. En C++, `mm::dil()` (mismo nombre, misma firma) solo se convierte en la versión optimizada si OpenCV se enlaza explícitamente en la compilación; sin eso —que es justamente el caso por defecto, incluso en Moodle/VPL— `mm::dil()` se comporta como `mm::dil0()`. Esta elección evita que la ruta en C++ dependa de OpenCV solo para compilar y ejecutar un ejercicio simple: `g++ archivo.cpp -o archivo` ya es suficiente. Quien quiera la versión acelerada localmente puede compilar con `g++ -DMM_USE_OPENCV archivo.cpp -o archivo $(pkg-config --cflags --libs opencv4)`.

La biblioteca también ofrece funciones para lectura, visualización, conversión de colores, filtrado y morfología matemática a través de una interfaz simple:

```cpp
#include "morph.hpp"

int main() {
    mm::Image img = mm::gray(mm::read("lena.jpg"));  // lectura y conversión a niveles de gris
    mm::Image grad = mm::gradm(img);                 // gradiente morfológico
    mm::show(grad, "salida.png");                    // visualización (escribe en un archivo)
}
```

A diferencia de la versión Python, `mm::show()` exige una ruta de salida explícita: cada celda de código C++ del libro se ejecuta como un proceso aislado (compilado y ejecutado mediante `%%writefile` + `!g++ ...` en Colab), sin el contador global de figuras que solo tiene sentido dentro de un único proceso Jupyter.

Además, todos los proyectos del **Gemini Notebook** de la ruta C++ incluyen el archivo `morph.hpp`, permitiendo consultar y discutir directamente la implementación de los algoritmos presentados en el libro. De esta forma, el estudiante puede utilizar el propio Gemini Notebook como un asistente de estudios, haciendo preguntas como:

> Explique cómo se implementó la función `mm::dil0` de `morph.hpp`, mostrando el código y comentando cada etapa de forma didáctica. Además, explique la diferencia entre `mm::dil0()` y `mm::dil()` sin la opción `-DMM_USE_OPENCV`.

! Implementaciones didácticas e implementaciones optimizadas

En varios capítulos, un mismo algoritmo se presenta en dos versiones. Las funciones con sufijo `0`, `1`, etc. (por ejemplo, `mm::dil0()` y `mm::dil1()`) son implementaciones didácticas, desarrolladas para facilitar la comprensión de los algoritmos y disponibles independientemente de la opción de compilación. En cambio, las funciones sin sufijo (como `mm::dil()`) usan la implementación optimizada basada en OpenCV **solo** cuando se compilan con `-DMM_USE_OPENCV`; en caso contrario, se comportan como la versión didáctica correspondiente.
En las actividades evaluadas por el **VPL de Moodle**, la compilación sigue el modo por defecto sin `-DMM_USE_OPENCV` —no por limitaciones de memoria (como ocurre con scikit-learn/scikit-image en la ruta Python), sino para que ningún EP en C++ dependa de que OpenCV esté instalado en el entorno de ejecución. En la práctica, esto significa que, en el VPL, `mm::dil()` y `mm::dil0()` producen exactamente el mismo resultado. Localmente —por ejemplo, en **VS Code** o **Google Colab**—, el estudiante puede optar por compilar con `-DMM_USE_OPENCV` para comparar con la versión optimizada.

El código fuente de la biblioteca está disponible en:

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

Ejercicios de Programación (EPs) y Validación Automática

Cada unidad del libro incluye **Ejercicios de Programación (EPs)** prácticos y de complejidad creciente, diseñados para consolidar los conceptos presentados a lo largo del capítulo. Cada EP está acompañado por un **simulador interactivo**, disponible en las versiones HTML e IPYNB, que permite al estudiante manipular parámetros, visualizar el comportamiento de los algoritmos y desarrollar una comprensión intuitiva del problema antes de iniciar su implementación. De esta forma, el proceso de aprendizaje combina experimentación, programación y validación automática.

La validación de las soluciones se realiza localmente mediante la clase `TestSuite` (`testsuite.py`), que compara la salida del programa con los archivos de casos de prueba (`.cases`):

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

El sistema soporta múltiples lenguajes (Python, Java, C++, C, JavaScript y R) y puede integrarse directamente en Moodle. Para docentes interesados en el paso a paso completo de publicación de los EPs como actividades VPL, consulte el **Guía del Profesor**, al final de este prefacio, y el Apéndice B.

Código abierto

El proyecto se rige por principios de código abierto. El repositorio público reúne el texto, los códigos, las imágenes y los *scripts*: **github.com/fzampirolli/pdi-vc**

Cada versión del libro se archiva permanentemente en **Zenodo** con un *DOI citable*. Para citar este material en trabajos académicos:

> ZAMPIROLLI, Francisco de Assis. **PDI+VC — Procesamiento Digital de Imágenes y Visión Computacional**. UFABC, 2026. DOI: 10.5281/zenodo.20784605

Cabe registrar, al respecto, que el contenido expuesto en esta obra refleja la mirada crítica, la propuesta pedagógica y la experiencia docente de su autor. Así, los análisis, enfoques y opiniones expresados a lo largo de este libro representan únicamente el entendimiento de su creador, no constituyendo ni reflejando un posicionamiento oficial o institucional de la Universidad Federal del ABC (UFABC).

Antes de comenzar: *Notebooks* en Python

El concepto de ***Literate Programming*** (Programación Literaria), propuesto por Donald Knuth (Knuth, 1984), fundamenta la estructura de este material. La lógica invierte el paradigma tradicional: el programa se escribe para la lectura humana, asemejándose a un ensayo, mientras que el código se extrae por separado para la ejecución computacional.

El contenido está estructurado en *notebooks* — documentos que intercalan **células de texto** (en Markdown) y **células de código** (en Python).

- **Ejecución:** las células de código se identifican con `[ ]`. La ejecución puede realizarse con `Shift + Enter` o mediante el botón ▷ de la interfaz.
- **Entornos:** los *notebooks* pueden ejecutarse localmente, mediante Jupyter o Visual Studio Code (VS Code), así como en entornos de nube, como Google Colab.

> Nota sobre el formato
>
> En entornos interactivos, el código puede modificarse y ejecutarse. En las versiones estáticas (HTML o PDF), los bloques de código tienen finalidad de lectura y referencia, sin perjuicio de la integridad de las explicaciones.

Guía del Profesor: Publicación de EPs en Moodle

*Esta sección está destinada a docentes que deseen integrar los Ejercicios de Programación (EPs) a las actividades **VPL (Virtual Programming Lab)** de Moodle, complementando el Apéndice B.*

Integración con Moodle (VPL)

Los EPs de los *notebooks* pueden convertirse en actividades **VPL** en Moodle. El *script* `ep_tools.py` (ejecutado mediante `make build`) automatiza la exportación de los EPs del final de cada capítulo en dos etapas:

1. **Extracción** (`make eps`): genera un HTML interactivo independiente por EP, con enunciado, ejemplos y simulador, en `gen/book/eps/<versao>/`. Vea la lista completa en: https://fzampirolli.github.io/pdi-vc/eps/py.pt/index.html
2. **Conversión** (`make moodle`): transforma el HTML en un fragmento autocontenido, sin dependencia de CSS externo, listo para pegarse en el editor de Moodle, en `gen/book/eps/<versao>_moodle/`. Vea un ejemplo en: https://fzampirolli.github.io/pdi-vc/eps/py.pt_moodle/EP01_01.html

Adicionalmente, todos los simuladores interactivos desarrollados a lo largo del libro pueden accederse directamente en https://fzampirolli.github.io/pdi-vc/simuladores/py.pt/.

Otras rutas e idiomas

Los enlaces anteriores usan `py.pt` como ejemplo. Para acceder a los EPs o simuladores de otra combinación lenguaje/idioma, basta con cambiar ese segmento en la URL — por ejemplo, `cpp.it` para la ruta C++ en italiano: `https://fzampirolli.github.io/pdi-vc/eps/cpp.it/index.html`. La estructura es la misma para todas las combinaciones disponibles, en `/eps/<versión>/`, `/eps/<versión>_moodle/` y `/simuladores/<versión>/`.

Al publicar con `make publish`, estas **dos versiones** de cada EP quedan disponibles en los *enlaces* indicados. El profesor puede combinar las dos estrategias: pegar la versión Moodle en el editor VPL (con simulador funcional) e incluir en el enunciado un *enlace* a la versión completa, donde las fórmulas se renderizan correctamente mediante MathJax.

Revisión obligatoria antes de publicar en Moodle

Aunque automatizada, la conversión exige revisión del profesor debido a las limitaciones del editor TinyMCE/HTML Purifier de Moodle:

- **Fórmulas matemáticas** — TinyMCE descarta barras invertidas (\), corrompiendo ecuaciones LaTeX. Por ello, la versión Moodle convierte fórmulas simples a HTML puro (entidades como `&ge;`, `&times;`). Las fórmulas complejas (`\begin{cases}`, matrices, integrales) pueden salir incompletas — revise y, si es necesario, reescríbalas en texto o indique la versión completa mediante *enlace*.
- **Figuras externas** — Las imágenes complementarias deben insertarse manualmente en la actividad VPL.
- **Simuladores** — Funcionan perfectamente siempre que utilicen JavaScript estándar con estilos *inline* y manipulación directa del DOM (`getElementById`). **Atención:** si incluye fórmulas en LaTeX que contengan el carácter \ en Moodle, los simuladores pueden dejar de funcionar.

Cómo Publicar en Moodle

1. Acceda a la versión Moodle del EP deseado:

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

2. Copie el código fuente completo de la página (`Ctrl+U` → `Ctrl+A` → `Ctrl+C`).
3. En Moodle, cree la actividad VPL, acceda al editor HTML, pegue el contenido (`Ctrl+V`) y guarde.
4. Importe los archivos `.cases` de la carpeta `all/capXX/casos/` en las configuraciones de pruebas del VPL.

Reutilización de los Casos de Prueba

Los archivos `.cases` son compatibles con el VPL, permitiendo usar el mismo conjunto de pruebas tanto en el desarrollo local como en la corrección automatizada en Moodle.

Parte I

Parte I — Fundamentos de PDI

Capítulo 1

Fundamentos y Primeros Pasos

Este capítulo inaugura la **Parte 1** del libro, dedicada a los fundamentos del Procesamiento Digital de Imágenes (PDI). Se presentan la representación matemática de imágenes digitales y los principales métodos para su manipulación.

Se utiliza el lenguaje C++ y la biblioteca `morph.hpp`, un puerto didáctico de `morph.py` (ZAMPIROLLI, 2025).

1.1 Objetivos

Al final de este capítulo, usted será capaz de:

- Comprender la naturaleza física y matemática de la imagen digital $f(x, y)$.
- Identificar las bandas del espectro electromagnético relevantes para PDI.
- Realizar operaciones básicas: lectura, visualización y guardado de imágenes.
- Acceder y modificar intensidades de píxeles individualmente.
- Aplicar umbralización manual.
- Configurar el entorno de desarrollo en C++.
- Manipular estructuras de matrices (`std::vector`) sin caer en trampas de copia/referencia.

1.2 Antes de comenzar: *Notebooks* interactivos

Este material fue construido bajo el concepto de ***Literate Programming*** (Programación Literaria), ideado por Donald Knuth en la década de 1980 (KNUTH, 1984). Knuth —también creador del sistema **TeX** para tipografía digital— propuso que los programas se escribieran como una narrativa lógica para los seres humanos, intercalando código y documentación.

Para ejecutar una celda, presione Shift + Enter o haga clic en el botón ▷.

> **Nota sobre el formato**
>
> En las versiones renderizadas (PDF o HTML), el código se presenta en bloques estáticos con fines de lectura y referencia. La ejecución interactiva requiere el acceso a través de Google Colab (disponible en la parte superior de la página) o en un entorno local mediante VSCode o Jupyter Notebook.

1.3 Fundamentos

El estudio de sistemas basados en imágenes comprende un ecosistema de disciplinas integradas que transforman datos visuales brutos en conocimiento estructurado. Mientras algunas áreas se centran en la generación

de representaciones, otras se dedican al tratamiento y análisis de estos datos para respaldar aplicaciones tecnológicas complejas.

El diagrama presentado en el Figura 1.1 establece la distinción y complementariedad entre el Procesamiento Digital de Imágenes (PDI) y la Visión por Computador (VC). El PDI, destacado en **verde**, se centra en la transformación de imagen a imagen, con el objetivo de mejorar la calidad o realizar un preprocesamiento, como la eliminación de ruidos y el realce de contraste.

En contraste, la VC, señalada en **azul**, se centra en la interpretación del contenido visual para extraer modelos o información, como el reconocimiento de objetos y gestos. La región de intersección ilustra la sinergia entre las áreas, donde el PDI prepara los datos visuales para la interpretación por parte de la VC. El mapa también demuestra las interconexiones de ambas disciplinas con áreas como Robótica, Computación Gráfica, Inteligencia Artificial (IA) y Neurociencia.

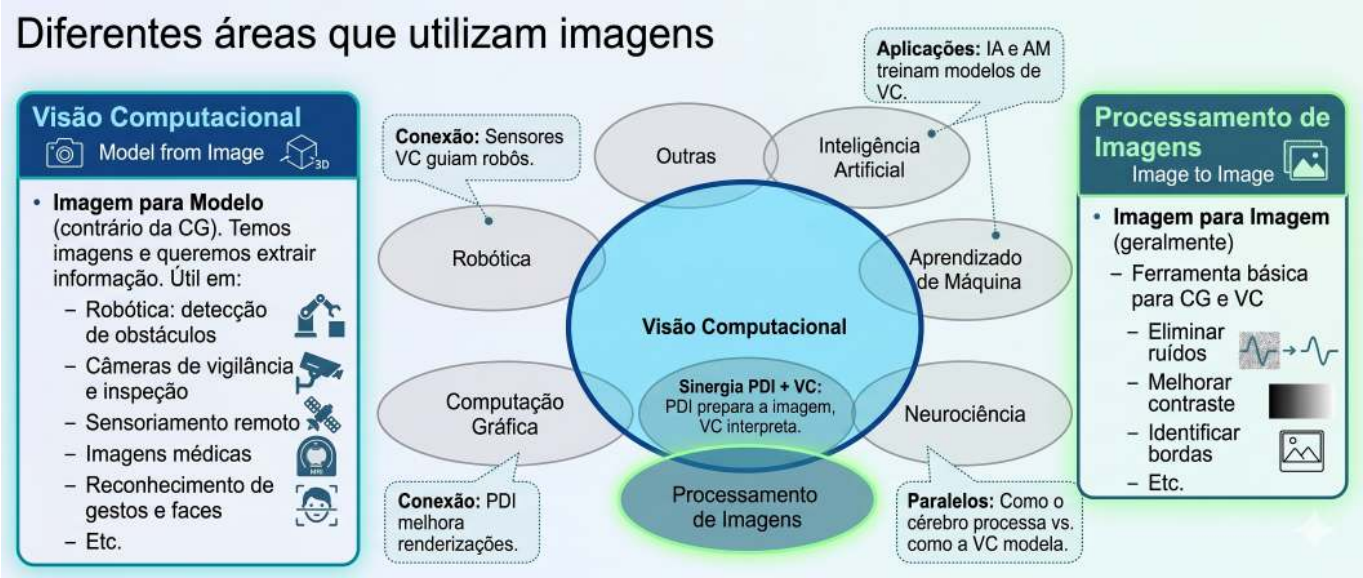


Figura 1.1: Diagrama relacional que detalla las distinciones fundamentales, sinergias e interconexiones entre PDI y VC en el contexto de sistemas basados en imágenes.

1.3.1 👁 Visión por Computador

- **Enfoque:** *Imagen* → *Modelo* (camino inverso de la Computación Gráfica).
- **Objetivo:** Extraer información de alto nivel a partir de imágenes o videos.
- **Aplicaciones típicas:**
  - Robótica: detección de obstáculos, localización y navegación autónoma.
  - Vigilancia e inspección: reconocimiento de eventos, lectura de placas, control de calidad.
  - Teledetección: análisis de imágenes satelitales, cartografía ambiental.
  - Imágenes médicas: detección de tumores, segmentación de órganos, ayuda al diagnóstico.
  - Interacción humano-computador: reconocimiento de gestos, expresiones faciales, seguimiento ocular.
- **Relación con otras áreas:** utiliza técnicas de Aprendizaje Automático e IA para clasificar e interpretar escenas; sirve como “ojos” de la Robótica.

1.3.2 🖼 Procesamiento Digital de Imágenes (PDI)

- **Enfoque:** *Imagen* → *Imagen* (generalmente: transformación de una imagen en otra).
- **Objetivo:** Mejorar la calidad visual o extraer características de bajo nivel.

- **Aplicaciones comunes:**
  - Eliminación de ruido (filtros de media, mediana, gaussiano).
  - Mejora del contraste (ecualización de histograma, ajuste gamma).
  - Detección de bordes (Sobel, Canny, Laplaciano).
  - Segmentación (umbralización, crecimiento de regiones, *watershed*).
  - Transformaciones geométricas (redimensionamiento, rotación, corrección de perspectiva).
- **Relación con otras áreas (ver Tabla 1.1):**
  - Es la **base** de la mayoría de los sistemas de VC (preprocesamiento).
  - La Computación Gráfica aplica frecuentemente PDI para posprocesamiento (ej.: suavizado, realce).
  - Las técnicas de IA pueden optimizar parámetros de procesamiento (ej.: aprendizaje de filtros).

Tabla 1.1: Conexión entre PDI, VC y otras áreas de la ciencia.

Área	Relación con PDI y VC
Inteligencia Artificial	Proporciona modelos (redes neuronales, SVM) que interpretan salidas de la VC.
Robótica	Consume datos de VC para tomar decisiones (navegación, manipulación).
Aprendizaje Automático	Utiliza descriptores extraídos por PDI/VC para entrenar clasificadores.
Computación Gráfica	Camino inverso: *modelo* → *imagen*; muchas veces aplica PDI para renderizado realista.
Neurociencia	Inspira modelos de PDI (ej.: filtros similares a células ganglionares de la retina).

1.4 Etapas del PDI

Las etapas del PDI se presentan en la Figura 1.2, que pueden comprenderse como una cadena de transformaciones que reduce la redundancia de los datos en busca de significado:

- **Bajo Nivel:** Actúa directamente sobre los píxeles de la imagen ruidosa para realizar mejoras y filtrados, generando como salida una imagen limpia o realzada.
- **Nivel Medio:** Recibe la imagen tratada y realiza la segmentación y descripción, transformando la matriz de píxeles en atributos estructurados (forma, tamaño y textura).
- **Alto Nivel:** Utiliza la tabla de atributos para alimentar procesos de lógica e inteligencia artificial, resultando en la decisión o reconocimiento final (como el diagnóstico médico).

A Figura 1.3 detalla la secuencia completa del PDI, desde la captura hasta la interpretación. El flujo comienza en la Adquisición de la Imagen (1) y prosigue con el Mejoramiento (2) y la Restauración (3). A continuación, el contenido se aísla mediante la Segmentación (4) y se refina con la Morfología (5). La transición crucial ocurre en la Representación y Descripción (6), donde los objetos visuales se convierten en datos matemáticos (área, perímetro, etc.), lo que permite el Reconocimiento (7). Los procesos auxiliares incluyen el Procesamiento de Imagen en Color y la Compresión, que contribuyen a la eficiencia del almacenamiento y del análisis.

1.5 Formación de la Imagen y el Espectro

El proceso de formación de una imagen se fundamenta en la interacción entre la materia y la energía radiante. Esencialmente, una imagen se concibe cuando un **sensor registra la radiación** resultante de la interacción con un objeto físico. En el contexto de la visión humana y de la fotografía convencional, este fenómeno depende de una fuente de luz que ilumine la escena; las características de los objetos se codifican entonces a través de las **variaciones de intensidad y color** de la luz que alcanza el sensor, tal como se ilustra en la Figura 1.4.

La **luz visible** ocupa solo una pequeña franja del espectro electromagnético — entre **380 nm (violeta)** y **750 nm (rojo)** — según se ilustra en Figura 1.5.. Los sensores digitales convencionales operan en esa

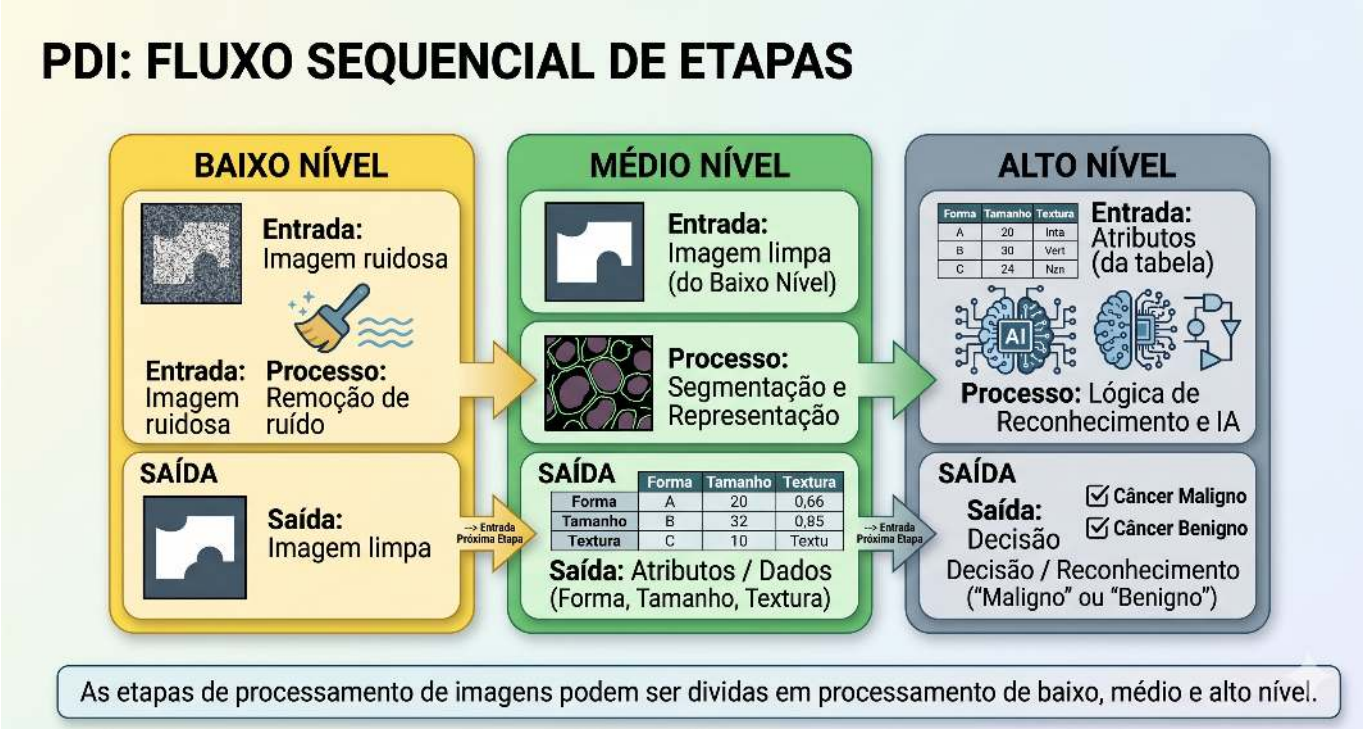


Figura 1.2: Representación del flujo secuencial de procesamiento: la salida de cada nivel se convierte en la entrada del nivel subsiguiente.

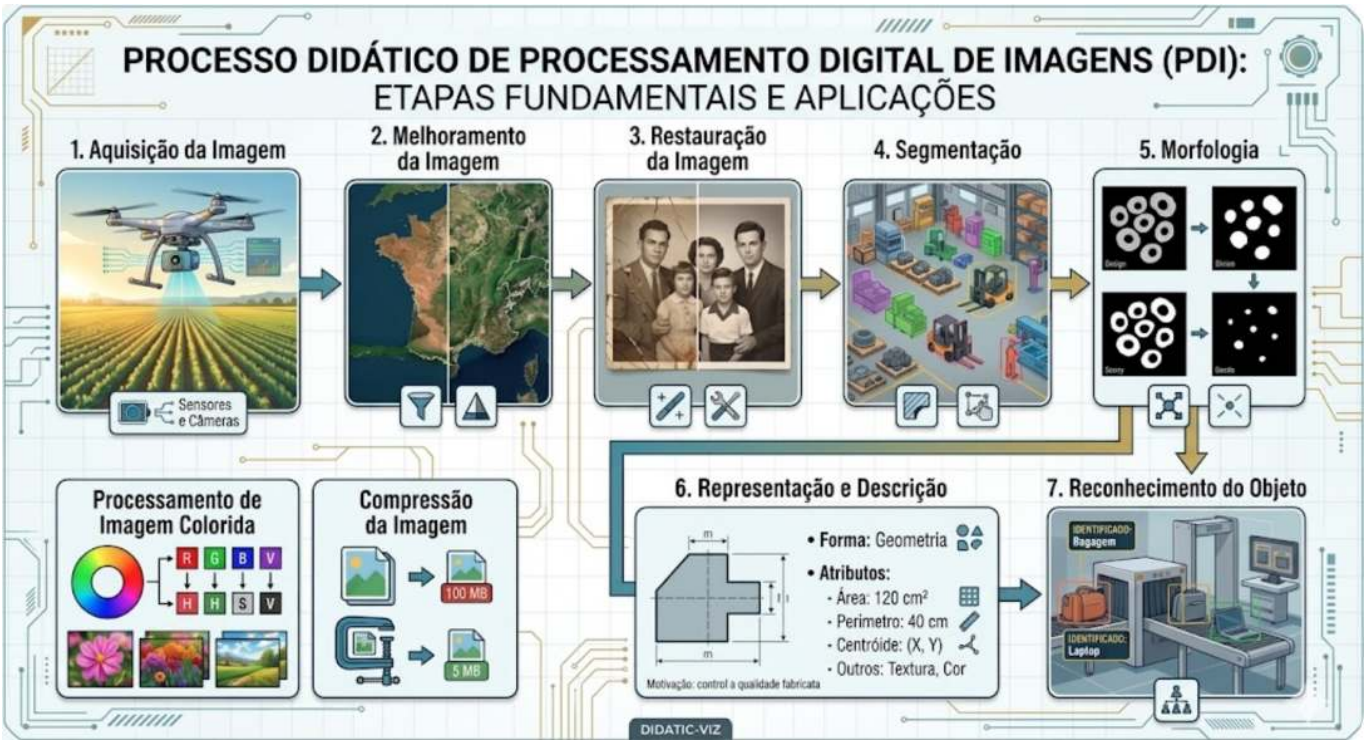


Figura 1.3: Flujo detallado del PDI: desde la adquisición sensorial hasta la extracción de atributos y el reconocimiento automatizado, incluyendo el procesamiento en color y la compresión.

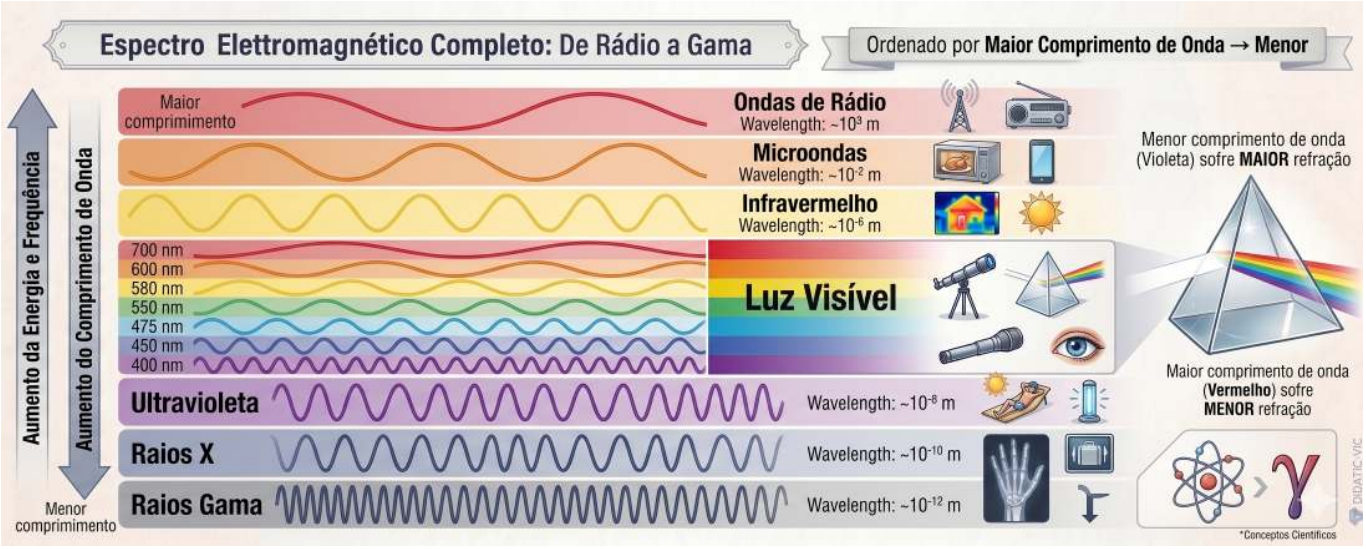


Figura 1.4: Representación del espectro visible y su posición en relación con las demás radiaciones electromagnéticas, destacando la variación de las longitudes de onda de 380 nm a 750 nm.

misma ventana, pero los equipos especializados pueden captar radiaciones invisibles al ojo humano, como el infrarrojo y los rayos X. En PDI, la imagen formada depende directamente de la sensibilidad espectral del sensor utilizado.

A partir de ese proceso físico de adquisición, se vuelve posible modelar matemáticamente la imagen digital como una función bidimensional discreta, en la cual cada punto de la escena está representado por muestras numéricas de intensidad luminosa, formalizando así los conceptos de píxel y de imagen digital presentados en la siguiente sección.

1.6 ¿Qué es una Imagen Digital?

Una **imagen digital** está formada por una cuadrícula de **píxeles** (*Picture Elements*), donde cada píxel es la unidad elemental más pequeña de la imagen.

> ¿Qué es un Píxel?
>
> Un **píxel** es la unidad direccionable más pequeña que compone una imagen digital. Cada píxel ocupa una posición única en la cuadrícula y almacena uno o más valores numéricos que representan su intensidad o color.

Representación Matemática

A diferencia de una función continua, el dominio de una imagen digital es un plano rectangular finito $\mathbb{E} \subset \mathbb{Z}^2$, que representa la cuadrícula de muestreo. Este dominio está indexado por coordenadas enteras:

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

Donde:

- L: representa el ancho de la imagen (número de columnas).
- H: representa la altura de la imagen (número de filas).

La imagen digital es una función que asocia cada par de coordenadas (x, y) a uno o más valores que describen la apariencia del píxel.

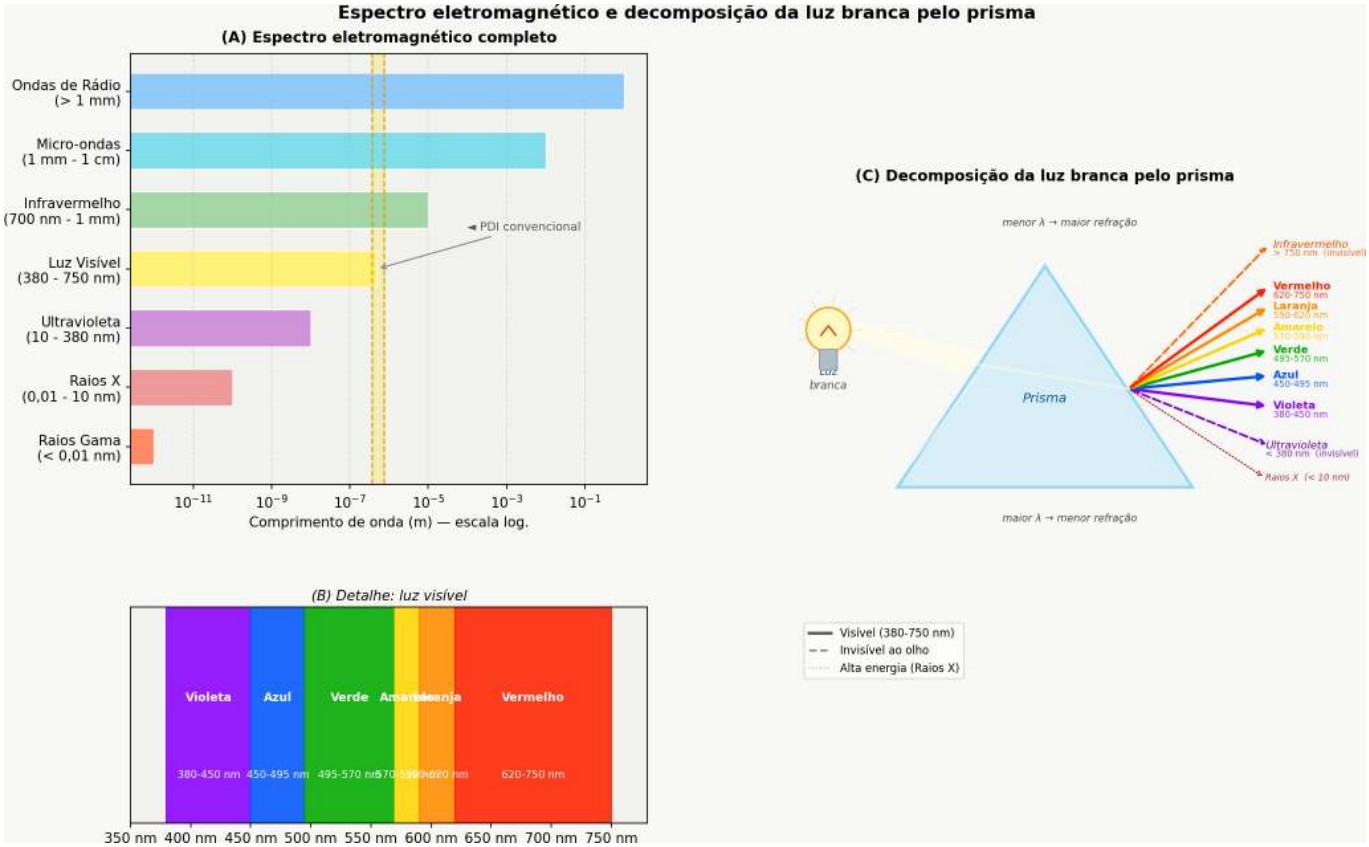


Figura 1.5: (A) Espectro electromagnético completo en escala logarítmica, con énfasis en el rango visible. (B) Detalle de la luz visible (380-750 nm) y sus colores. (C) Descomposición de la luz blanca por el prisma: la menor longitud de onda sufre mayor refracción, separando UV, visible e infrarrojo.

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

El conjunto $\mathcal{V}$ define los valores posibles para el píxel (codominio), variando según el tipo de imagen, como se demuestra en la Tabla 1.2.

Tabla 1.2: Principales tipos de imagen digital y sus respectivos conjuntos de valores posibles para cada píxel.

Tipo de imagen	$\mathcal{V}$ (valores del píxel)	Representación
Binaria	$\{0, 1\}$ o $\{0, 255\}$	■□
Escala de grises	$\{0, 1, \ldots, 255\}$	`[ ][=][#]`■
Color (RGB)	$\{0, \ldots, 255\}^3$ (triplas ordenadas de valores)	■■■

Ejemplo práctico: Una imagen a color en el modelo RGB puede representarse matemáticamente mediante una función que asocia tres valores de intensidad a cada píxel. Computacionalmente, esta representación corresponde a tres matrices superpuestas — los canales rojo (*Red*), verde (*Green*) y azul (*Blue*) — en las cuales cada elemento almacena la intensidad luminosa del respectivo canal en una posición determinada de la imagen.

Para viabilizar os experimentos prácticos de PDI e VC, este libro utiliza C++17 y un conjunto mínimo de bibliotecas orientadas a la manipulación matricial, al procesamiento de imágenes y a la visualización de resultados, presentadas a continuación.

1.7 Configuração do Ambiente

Este material utiliza **C++17** e a biblioteca de cabeçalho único **morph.hpp**, originalmente proposta para Python em Zampirolli (2025). Aqui, é utilizada uma versão mínima e didática da `morph.py`, adaptada para compilação simples com `g++`, conforme apresentado na Tabla 1.3.

Cada célula de código é compilada e executada como um processo isolado (`%%writefile arquivo.cpp` seguido de `g++`). Diferentemente do *kernel* Python, não há estado persistente entre as células: as imagens produzidas por uma célula são salvas em arquivos para serem lidas pela próxima.

Os **Ejercicios de Programación (EPs)** apresentados al final de los capítulos pueden ser validados por el mismo módulo `testsuite.py` utilizado en la ruta Python, que compila la solución con `g++` y compara su salida con los casos de prueba. Así, los mismos casos de prueba (`.cases`) pueden validar soluciones en C++, Python y otros lenguajes, tanto en los *notebooks* como en el **Moodle/VPL**.

Tabla 1.3: Principales bibliotecas y herramientas utilizadas en la ruta C++ de este libro.

Biblioteca / Herramienta	Función principal
`g++` (C++17)	Compilación de cada celda de código
`morph.hpp`	Abstracción didáctica de operaciones de PDI
`stb_image.h` / `stb_image_write.h`	Lectura/escritura de PNG/JPEG (sin OpenCV)
`testsuite.py`	Ejecución y validación automática de los EPs

1.8 Sobre el `morph.hpp`

La `morph.hpp` es una versión C++ mínima y didáctica de la `morph.py`: implementa las funciones usadas en este capítulo (`read`, `gray`, `randomImage`, `show`, `write`, `threshold`, `drawImg`), además de operaciones de morfología (`dil`/`ero` y variantes `dil0`/`ero0`/`dil1`/`ero1`) preparadas para los capítulos siguientes. No hay

paridad numérica garantizada con la implementación Python — el criterio es "compila y produce una imagen plausible", no "resultado bit a bit idéntico al Python".

Las bibliotecas `stb_image.h/stb_image_write.h` (dominio público/MIT) están **vendorizadas** junto al repositorio — es decir, su código fuente ya viene copiado dentro del propio proyecto, en lugar de instalado separadamente mediante `apt install` — lo que evita depender de paquetes del sistema durante la compilación en Colab. Las operaciones `dil()/ero()` usan `cv::dilate/cv::erode` cuando se compilan con `-DMM_USE_OPENCV`; sin esa *flag* (por defecto, incluso en Moodle/VPL), se recurre a la versión didáctica equivalente (`dil1/ero1`) sin requerir OpenCV.

```
import os, urllib.request

os.makedirs("tmp/state", exist_ok=True)  # artefatos de build del trayecto C++ (.cpp, binario,
PNGs)

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

# El kernel es Python incluso en el trayecto C++: `mm` (morph.py) es usado por los
# simuladores, por la exhibición de las figuras que el binario C++ genera y por el
# estado mm::Image entre celdas. cpp=True descarga también el trayecto compilado
# (morph.hpp + stb_image*.h), usado en el #include de las celdas %%writefile *.cpp.
import config
config.setup(testsuite=True, cpp=True)
from morph import mm
from testsuite import TestSuite
```

```
Entorno listo. Morph: 1.1.9 | OpenCV: 5.0.0 | TestSuite: 1.1.2
```

1.9 Fundamentos de Matrices — atención a la copia de referencias

Como una imagen digital puede representarse mediante una matriz, es importante comprender cómo crear y manipular matrices correctamente. En C++, `std::vector` tiene **semántica de valor**: copiar el vector copia sus datos. Por ello, `std::vector<std::vector<int>> m(3, std::vector<int>(2, 0))` crea de hecho tres filas **independientes** — el constructor de relleno copia la fila modelo tres veces.

> ⚠ Atención a la copia de referencias
>
> La trampa aparece cuando se usan **punteros** o **referencias** en lugar de copias. En `std::vector<int> linha(2, 0); std::vector<std::vector<int>*> m(3, &linha);`, los tres elementos de `m` apuntan a la **misma** fila, y modificar `(*m[0])[0]` altera todas. Lo mismo ocurre con `auto& linha = m[0];` (referencia — modifica la matriz), en contraste con `auto linha = m[0];` (copia — deja la matriz intacta).

Para visualizar este comportamiento, se puede ejecutar el código en Python Tutor (que también ejecuta C++ paso a paso) y comparar el efecto de copias y de referencias.

En la práctica, para el procesamiento digital de imágenes (PDI), se utiliza la estructura `mm::Image` de la `morph.hpp`, que también tiene semántica de valor: `mm::Image b = a;` copia los píxeles, mientras que `mm::Image& b = a;` crea solo un alias para la misma imagen. El siguiente código presenta diferentes formas de creación de imágenes sintéticas, cuyos resultados se muestran en la Figura 1.6..

```
%%writefile tmp/fig_imagens_sinteticas.cpp
#define MM_OUT "tmp/fig_imagens_sinteticas.png"
// Compile: g++ -std=c++17 -O2 program.cpp -o program $(pkg-config --cflags --libs opencv4)
-lm
```

```
#include "morph.hpp"
#include <iostream>
#include <vector>
#include <string>
#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:
 47 221  26  29 195 117
 68  33 148 205 188  35
 41 181 152 184 173 203
```

```
105 161 176  85 213 132

[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)
```

```
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)")
```

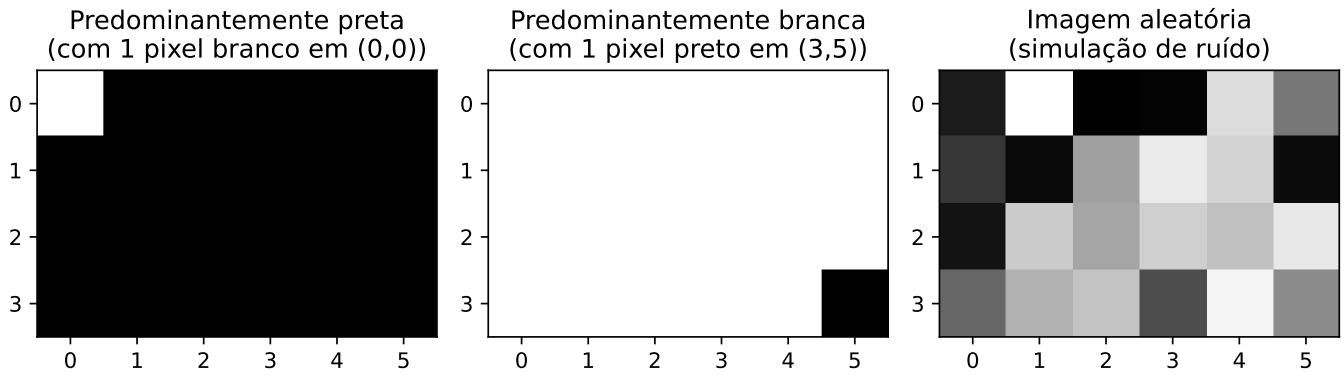


Figura 1.6: Exemplos de imagens sintéticas representadas matricialmente.

1.10 Leyendo y Mostrando Imágenes

En bibliotecas como OpenCV (`cv::Mat`), las imágenes digitales se representan computacionalmente como matrices multidimensionales. En `morph.hpp`, la estructura `mm::Image` adopta un enfoque más simple: un buffer lineal de bytes (`std::vector<unsigned char>`), con altura, ancho y número de canales explícitos.

Una de las operaciones fundamentales en PDI es la lectura de imágenes.

En `morph.hpp`, la función `mm::read()` permite cargar imágenes tanto de archivos locales como de URLs, como se ilustra en Figura 2.7..

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

```cpp
#include <iostream>
#include <string>
#include <filesystem>

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

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

    mm::Image img = mm::read(url);

    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
```

```python
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: *Descarga* de la Imagen para Almacenamiento Local

En entornos donde la lectura directa de URLs no esté disponible —debido a restricciones de red, políticas de *firewall* o ausencia de conectividad— una alternativa consiste en descargar previamente la imagen al sistema de archivos local y luego cargarla con `mm::read()`. En la ruta de C++, `mm::read` también acepta URLs directamente (descarga interna vía `curl`/`wget`); esta alternativa cubre el caso de descargar una vez y reutilizar el archivo local. En el siguiente ejemplo, el archivo se guarda como `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");   // lee del archivo local
    mm::show(img, MM_OUT);                      // Ejemplo: Captura RGB (archivo local)
    return 0;
}
```

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

Explicación

- `wget -O mandrill.png <URL>` realiza la descarga de la imagen y la almacena localmente con el nombre especificado;
- `mm::read("mandrill.png")` efectúa la lectura del archivo directamente desde el sistema de archivos, sin necesidad de solicitudes HTTP adicionales;
- esta estrategia reduce la dependencia de conectividad durante la ejecución de los experimentos y evita descargas repetidas de la misma imagen.

Nota

El prefijo `!` se utiliza en entornos basados en *notebooks*, como Jupyter Notebook, JupyterLab y Google Colab, para ejecutar comandos del sistema operativo directamente en celdas de código. En terminales convencionales, el comando debe utilizarse sin el prefijo `!`.
Si la utilidad `wget` no está instalada, se puede utilizar alternativamente:

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

1.11 Conversión de Tipos y Umbralización

Como vimos, una imagen a color en el espacio RGB se representa mediante la función:

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

Es decir, para cada píxel (x, y), tenemos tres valores (R, G, B) que definen su color.

Conversión a Escala de Grises

Para convertir una imagen RGB a escala de grises (*grayscale*), es necesario combinar los tres canales en un único valor de intensidad g, que representa el brillo percibido. Como el ojo humano no es igualmente sensible al rojo, verde y azul, se utiliza una **media ponderada**. El estándar ITU-R BT.601 ({ITU-R}, 2011) define los siguientes pesos:

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

Tras el cálculo, el valor g se redondea al entero más cercano y se ajusta al intervalo $[0, 255]$. El resultado es una nueva imagen, ahora en escala de grises, representada por:

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

Umbralización (*Thresholding*)

A partir de la imagen en escala de grises $f_{\text{gris}}(x, y)$, una operación fundamental es la **umbralización**, que produce una imagen **binaria** (solo blanco y negro). Para ello, se elige un valor de corte T (generalmente en el intervalo $[0, 255]$) y se define:

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

Ejemplo: Con $T = 128$, los píxeles con intensidad superior a 128 se vuelven blancos (255); los demás se vuelven negros (0).

La umbralización se usa ampliamente para **segmentar objetos del fondo**, extraer bordes o crear máscaras binarias para el procesamiento posterior.

> **Nota:** El valor 255 representa el blanco máximo en imágenes de 8 bits, mientras que 0 representa el negro absoluto.

Ejemplo práctico de conversión y umbralización

La Figura 1.8 ilustra los principales pasos para transformar una imagen a color en escala de grises y, posteriormente, convertirla en una imagen binaria mediante umbralización. El siguiente código implementa estas etapas:

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

    // 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=" +
std::to_string(limiar) + ")"},
        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 Umbralización por el método de Otsu

Tal como se presentó en la Ecuación 1.4, la umbralización convierte una imagen en tonos de gris a binaria usando un valor de corte T. Hasta ahora fijamos $T = 128$ manualmente.

```
mm::Image img_bin_fijo = mm::threshold(img_gris, 128);
```

Sin embargo, la elección manual de T no siempre es trivial. La biblioteca `mm` ofrece una alternativa automática: cuando el umbral **no se proporciona**, la función `mm::threshold(img_gris)` calcula el valor de T mediante el **método de Otsu** (OTSU, 1979). Este método, que se detallará en capítulos futuros, maximiza la varianza entre clases del histograma (frecuencia de cada tono de gris), separando automáticamente los píxeles de objeto y fondo.

El código siguiente compara la umbralización manual ($T = 128$) con la automática (Otsu). La binarización por Otsu se realiza con `mm::threshold(img_gris)`; dado que la API mínima de `morph.hpp` no devuelve el T calculado, el valor mostrado en la etiqueta de la figura se recupera con `cv2.threshold` en la etapa de visualización (mismo algoritmo, mismo 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 <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_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, Otsu quando o argumento é
omitido
    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)")
```

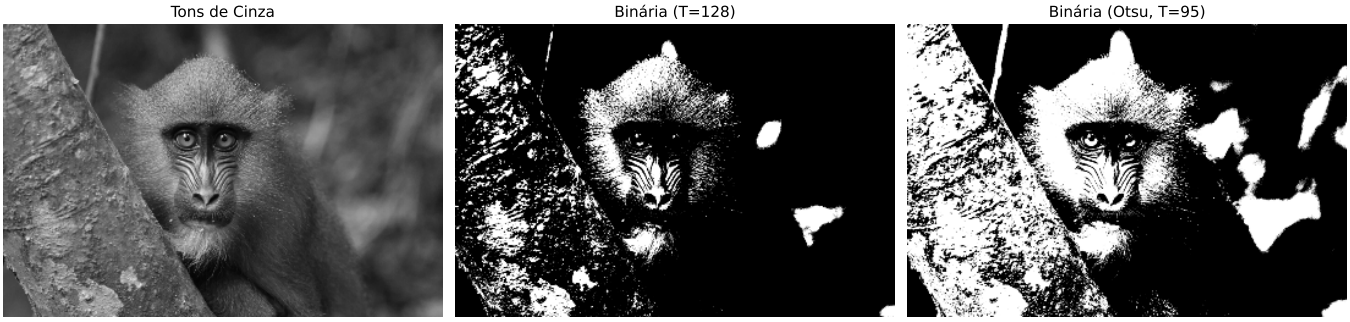


Figura 1.9: Comparação entre limiarização manual (T=128) e automática (Otsu) sobre a imagem em tons de cinza.

La Figura 1.9 muestra que el umbral obtenido por Otsu se adapta automáticamente a la imagen, resultando en una binarización más eficiente que un valor fijo, especialmente cuando las intensidades del objeto y del fondo están bien separadas en el histograma. Esta técnica es ampliamente utilizada en sistemas de VC para la binarización de documentos, detección de objetos y preprocesamiento de imágenes.

Simplicidad de la biblioteca `morph.hpp`

Mientras que OpenCV exige la llamada completa:

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

`morph.hpp` abstrae toda esa complejidad: basta con llamar a `mm::threshold(img_gray)`. El umbral de Otsu se calcula automáticamente y la imagen binaria se devuelve directamente. Este enfoque permite concentrarse en el concepto, no en los detalles de implementación.

1.13 Acceso a Píxeles

En `morph.hpp`, la imagen es un `mm::Image` con un buffer lineal `data` (fila tras fila, canales intercalados) y el método `at(y, x, c)` para acceso individual, siguiendo la convención matricial **fila** (eje Y) y **columna** (eje X): `img.at(fila, columna)` para tonos de gris y `img.at(fila, columna, canal)` para cada canal de una imagen RGB.

El código de la Figura 1.10 demuestra cómo extraer esos valores en imágenes de color (RGB) y en tonos de gris, además de aislar la vecindad inmediata del punto de interés.

Acceso a píxeles en C++ (`mm::Image`):

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


# 1. Coordenadas del píxel objetivo (fila, columna)
r, c = 600, 800


# 2. Acceso directo a los valores del píxel

pixel_cinza = img_gray[r, c]
print(f"Píxel objetivo ({r}, {c}):")

print(f"  Tons de gris (escalar) : {pixel_cinza}")
print(f"  Color (canales RGB)    : R={img[r, c, 0]}, G={img[r, c, 1]}, B={img[r, c, 2]}")


# 3. Vecindario 3x3 alrededor de (r, c); el píxel central va entre corchetes

print("Matriz de vecindario 3x3 (tons de gris):")
for dy in range(-1, 2):

    linha = ""
    for dx in range(-1, 2):

        v = img_gray[r + dy, c + dx]
        if dy == 0 and dx == 0:

            linha += f"[{v:3d}]"
        else:

            linha += f" {v:3d} "
    print("  " + linha)

# 4. Marca la posición del píxel con un cuadrado rojo hueco (borde de
#    3 px) en una copia de la imagen y muestra (equivalente al punto de matplotlib).

marcada = img.copy()
lado = 20   # medio-lado del cuadrado, en píxeles

esp = 5     # grosor del borde


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"Ubicación del píxel ({r}, {c})")
```


Figura 1.10

- **Escala de grises:** `img_gray.at(r, c)` devuelve un `unsigned char` (0 a 255) con la intensidad de gris en el píxel.
- **RGB:** `img.at(r, c, 0)`, `img.at(r, c, 1)`, `img.at(r, c, 2)` acceden a los canales **R**, **G** y **B** — un índice de canal a la vez; no hay vector `[R, G, B]`.
- **Vecindario** 3×3**:** se recorre mediante dos bucles anidados sobre `img_gray.at(r + dy, c + dx)`, con `dy, dx` en $\{-1, 0, 1\}$ — no hay segmentación (*slicing*) como en NumPy. Este barrido es la base para filtros espaciales y convoluciones.
- No hay gráficos interactivos (equivalente a `plt.plot`): para marcar la posición del píxel, la celda de código siguiente **dibuja un cuadrado rojo hueco** alrededor de él en una copia de la imagen (`marcada = img.copy()`, luego `marcada.at(...)`) y la muestra con `mm::show`.

La indexación es *basada en cero*: `(0,0)` es la esquina superior izquierda. La primera dimensión controla la **altura** (filas/Y) y la segunda la **anchura** (columnas/X).

1.14 Resumen

En este capítulo se presentaron los fundamentos de la representación de imágenes digitales: la definición de píxel, la estructuración de imágenes en matrices y el impacto del muestreo y la cuantificación en la calidad final:

- **Imagen digital** = función $f(x, y)$ que mapea coordenadas a intensidades (escalares o vectoriales).
- **Dominio**: conjunto finito $\mathbb{E} = \{(x, y) \in \mathbb{Z}^2 \mid 0 \leq x < L,\ 0 \leq y < H\}$.
- **Tipos principales**: binaria ($\mathcal{V} = \{0, 255\}$), escala de grises ($\mathcal{V} = [0, 255]$) y RGB ($\mathcal{V} = [0, 255]^3$).
- **Umbralización** convierte la escala de grises en binaria; el **método de Otsu** determina el valor de corte automáticamente maximizando la varianza entre clases.
- A `morph.hpp` (o `mm::`) ofrece funciones didácticas para operaciones básicas de PDI, como `mm::gray()`, `mm::threshold()`, `mm::show()` (sobrecargada para 1 imagen o varias).
- **Trampa de copia**: `std::vector` tiene semántica de valor, pero punteros/referencias compartidos entre "filas" de una matriz reintroducen el mismo problema del NumPy — prefiera `mm::Image`, que también copia por valor.
- Acceso a píxeles mediante `img.at(fila, columna)`, con indexación ***basada en cero***.

El Capítulo 2 abordará **histogramas y ecualización de contraste**.

1.15 🤖 Uso del Gemini Notebook como Tutor Complementario

En esta edición, además de los *notebooks* interactivos en Google Colab, el **Gemini Notebook** está disponible como herramienta complementaria de estudio. La plataforma utiliza exclusivamente los documentos proporcionados por el autor como base de conocimiento, garantizando respuestas coherentes con el contenido del libro.

🎓 Estudia con el Tutor Inteligente

Accede al entorno del capítulo a través del *enlace* a continuación y explora especialmente las opciones **Guía de Estudio** y **Conversación** para profundizar tu comprensión.
🚀 ACCEDER A GEMINI NOTEBOOK: CAPÍTULO 01

🌐 **Idioma y Lenguaje de Programación**

El proyecto de este capítulo en Gemini Notebook fue construido únicamente con el texto en **portugués** y los ejemplos de código en **Python**. Si estás estudiando con la edición en inglés o francés, o siguiendo

la ruta en C++, las respuestas del tutor pueden no corresponder exactamente a la versión que estás leyendo.

⚠️ **Aviso sobre Contenido Generado por IA**

La IA es una poderosa aliada en los estudios, pero el contenido generado puede contener **errores o imprecisiones**. Consulta siempre **libros, artículos científicos y otras fuentes académicas confiables** para validar la información. Siempre que sea posible, ejecuta los ejemplos prácticos proporcionados en este capítulo para verificar los resultados.

Funcionalidades Disponibles en la Plataforma

Gemini Notebook ofrece una suite avanzada de herramientas basadas en IA para transformar el contenido estático del libro en una experiencia de aprendizaje dinámica y multimedia. La plataforma utiliza técnicas de **RAG** (*Retrieval-Augmented Generation*), fundamentadas en el trabajo de Lewis (2020), para basar las respuestas estrictamente en los documentos proporcionados y minimizar la ocurrencia de alucinaciones.

Las principales funcionalidades incluyen:

- **Resúmenes Multimodales (Audio y Video)**: Generación de conversaciones naturales entre expertos en el formato de **Resumen de Audio** (estilo *podcast*) y **Resumen de Video**, discutiendo los temas centrales del capítulo, como las diferencias entre PDI y VC, o la interpretación de transformaciones como la umbralización y el método de Otsu.
- **Visualización de Estructuras (Mapa Mental e Infografía)**: Creación automática de diagramas que conectan visualmente los conceptos, por ejemplo, el flujo de procesamiento desde la captura de la imagen digital, pasando por la conversión a tonos de gris, umbralización y segmentación binaria.
- **Herramientas de Evaluación (Prueba y Tarjetas Didácticas)**: Generación de **Pruebas** de opción múltiple y **Tarjetas Didácticas** (*flashcards*) para la fijación de conocimientos, basadas en el texto autoral (ej.: preguntas sobre la fórmula de conversión RGB→gris o sobre el funcionamiento del umbral global y de Otsu).
- **Apoyo a la Presentación (*Diapositivas* e Informes)**: Ayuda en la estructuración de **Presentaciones de *Diapositivas*** y en la redacción de **Informes** técnicos, facilitando la comunicación de resultados de experimentos con imágenes.
- **Análisis de Datos (Tabla de Datos)**: Organización de datos extraídos del texto en tablas estructuradas, ayudando en la comprensión de ejemplos prácticos, como la comparación entre diferentes valores de umbral.
- ***Chat* Contextualizado**: Permite el cuestionamiento directo sobre el código y la teoría, como: *"¿Cómo implementar la conversión de RGB a tonos de gris usando los pesos del estándar ITU-R BT.601?"* o *"¿Qué sucede con la imagen binaria si elijo un umbral T=200 en lugar de T=128?"*.

1.16 Lista de Ejercicios

1. (15%) Con sus propias palabras, defina **imagen digital** y **píxel**. Dé un ejemplo concreto de cómo una imagen a color (RGB) se representa matricialmente en el computador.

2. (15%) Explique las diferencias entre **imagen binaria**, **escala de grises (8 bits)** y **color RGB**, indicando el rango de valores posibles para cada píxel en cada tipo.

3. (20%) Considerando la fórmula de conversión RGB → escala de grises del estándar ITU-R BT.601:

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

   Calcule el valor del píxel en escala de grises para $(R, G, B) = (80, 180, 30)$. Redondee al entero más cercano.

4. (20%) ¿Qué es **umbralización** (*thresholding*)? Explique la diferencia entre elegir un umbral T fijo (ej.: $T = 128$) y utilizar el **método de Otsu** para la determinación automática del umbral. En pocas palabras, ¿cómo elige el umbral el método de Otsu?

5. (15%) En el contexto de la biblioteca didáctica `mm` discutida en el capítulo, responda:

   - a) (7,5%) ¿Cómo se accede al valor del píxel en la posición (fila=50, columna=60) de una imagen en tonos de gris `img_gray`?
   - b) (7,5%) ¿Cuál es la ventaja de usar `mm::threshold(img_gray)` sin pasar el umbral? Compare con la llamada equivalente en OpenCV (`cv::threshold`).

6. (15%) ¿Qué representan los campos `h`, `w` y `channels` de un `mm::Image` para una imagen RGB? Dé un ejemplo concreto con una imagen de 640×480 píxeles.

Referencias del Capítulo

La fundamentación teórica de este capítulo comprende las siguientes obras de PDI y VC:

- Gonzalez (2018) para los fundamentos de **Procesamiento Digital de Imágenes** (PDI).
- Singh (2019) para la implementación práctica de métodos de **procesamiento y análisis de imágenes**.
- Szeliski (2022) para el estudio de VC y algoritmos fundamentales.
- Bradski (2008) para la aplicación de la biblioteca **OpenCV** en el entorno Python.
- Lewis (2020) para el concepto de **generación aumentada por recuperación (RAG)**, utilizado en el soporte al procesamiento de la información de este material.

Executar Colab Abrir GitHub

```
# Aquí va la traducción del contenido en portugués al español
# (Este bloque de código se mantiene sin cambios según las reglas)
```

1.17 💻 Parte Práctica con Ejercicios de Programación

🎯 Objetivo de este Cuaderno

Los **Ejercicios de Programación (EPs)** presentados a continuación también pueden enviarse en actividades del Moodle (actividades VPL) que proporcionan *feedback* automático.

Este cuaderno fue desarrollado para superar limitaciones de uso del Moodle. Con él, se debe:

1. **Desarrollar:** Escribir y editar tu solución directamente en el entorno Colab.
2. **Validar:** Probar tu código localmente utilizando los **mismos casos de prueba** del Moodle.
3. **Organizar:** Guardar tus códigos de las actividades VPL de forma segura.
4. **Evaluar:** Cuando estés conectado al Moodle, basta con copiar tu solución y hacer clic en **Evaluar** en el Moodle (**si estás en la red de la UFABC**) para registrar tu nota oficial.

§ Instrucciones Paso a Paso

En un entorno de ejecución (como VSCode, Jupyter o Colab), siga el orden a continuación para configurar el entorno y validar sus ejercicios:

Preparación del Entorno

Ejecute la celda de código a continuación para descargar `morph.py` y `testsuite.py` del repositorio de la disciplina — solo si aún no existen en el directorio local. Con ambos en `./`, el *notebook* y los subprocesos del `TestSuite` encuentran el módulo sin configuraciones adicionales de ruta.

Nota: El script `testsuite.py` buscará automáticamente los casos de prueba en `all/{cap}/cases` en GitHub.

Escribiendo el Código

Guarde su solución en una celda de código usando el comando mágico `%%writefile`. El nombre del archivo debe seguir el patrón `EPX_Y.*`, donde `X` es el capítulo, `Y` es el ejercicio y `*` es la extensión del lenguaje.

> **Ejemplo:** `%%writefile EP01_01.py`

Descarga

Descargue `morph.py` y `testsuite.py` ejecutando la celda a continuación:

```
import os, urllib.request

os.makedirs("tmp/state", exist_ok=True)  # artefatos de build del trayecto C++ (.cpp, binario,
PNGs)

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

# El kernel es Python incluso en el trayecto C++: `mm` (morph.py) es usado por los
# simuladores, por la exhibición de las figuras que el binario C++ genera y por el
# estado mm::Image entre celdas. cpp=True descarga también el trayecto compilado
# (morph.hpp + stb_image*.h), usado en el #include de las celdas %%writefile *.cpp.
import config
config.setup(testsuite=True, cpp=True)
from morph import mm
from testsuite import TestSuite
```

```
Entorno listo. Morph: 1.1.9 | OpenCV: 5.0.0 | TestSuite: 1.1.2
```

Ejecutando los Testes

Después de guardar el archivo con tu solución, ejecuta el comando a continuación (en una nueva celda) para evaluar los testes automáticos:

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

Reemplaza la extensión según el lenguaje usado:

Lenguaje	Extensión
Python	`.py`
Java	`.java`
C	`.c`
C++	`.cpp`
JavaScript	`.js`
R	`.r`

> **Cómo funciona:** El `TestSuite` descarga los casos de prueba de GitHub, ejecuta tu programa con cada entrada y compara la salida con la esperada – calculando automáticamente tu nota.

Para probar código Python directamente, sin guardar archivo, usa `run_code(codigo)` pasando el código como *string* en una variable `codigo`:

```
codigo = """
from morph import mm
```

```
# ... tu código aquí ...
"""
TestSuite("EP04_01").run_code(codigo)
```

⚠️ Importante: Reglas y Buenas Prácticas

🔹 Sobre la Entrada de Datos

Su programa debe leer la entrada estándar (teclado).

- **Python:** Utilice `input()`.
- **Otros lenguajes:** Utilice el comando de lectura estándar equivalente (`cin`, `Scanner`, etc.).

🔹 Configuración de IA en Colab

Para un mejor aprendizaje, se recomienda desactivar el autocompletado de código por IA, ya que no estará disponible durante las evaluaciones. Por ejemplo, en el navegador Chrome:

- Vaya a: **Herramientas > Configuración > IA generativa**
- Desmarque: **Habilitar generación de código**

🔹 Integridad Académica (Plagio)

Este recurso de pruebas locales se aplica a EPs **sin variaciones**. Sin embargo:

- **Individualidad:** Cada estudiante debe desarrollar su propia solución.
- **Detección de Similitud:** El profesor utiliza herramientas que detectan copias, **incluso con cambios de nombres de variables o espacios en blanco**.

1.17.1 EP01_01 📏 Tres métricas de distancia en PDI

En esta actividad, debes escribir un programa que calcule las tres distancias clásicas en PDI: **Euclidiana (L2)**, **City-Block (L1)** y ***Chessboard*** **(L∞)**.

- Lee **4 números reales** que representan las coordenadas: A_x, A_y, B_x, B_y.
- Calcula las tres distancias utilizando las fórmulas:

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

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

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

- Imprime los tres resultados, cada uno en una línea, formateados con **dos cifras decimales**, en el orden: **Euclidiana**, ***City-block***, ***Chessboard***.

📌 **Importante**:

- Utiliza las funciones matemáticas estándar de tu lenguaje: `math.sqrt`, `abs` (o `fabs`) y `max`.
- La salida debe contener **solo los números** (uno por línea), sin textos adicionales.
- Consulta un simulador interactivo para esta cuestión en la **Figura 1.11** (gráfico con arrastre de los puntos y visualización de las tres métricas).

1.17.1.1 🖼️ ¿Por qué es importante? – Costo computacional

En una imagen **1000×1000 píxeles** (1 millón de píxeles), calcular la distancia de cada píxel a un punto de referencia exige **1 millón de operaciones**. La elección de la métrica afecta el rendimiento:

Métrica	Operaciones por píxel	Costo relativo (1M píxeles)	Cuándo usar
Euclidiana (L2)	2 restas, 2 multiplicaciones, 1 suma, 1 `sqrt`	🔴 Más costosa – `sqrt` es cara	Distancia "real" en el espacio continuo
***City-block* (L1)**	2 restas, 2 `abs`, 1 suma	🟡 Moderada – sin raíz cuadrada	Cuadrículas, robótica, imágenes binarias
***Chessboard* (L∞)**	2 restas, 2 `abs`, 1 `max`	🟢 Más eficiente	Movimientos de piezas, morfología

La función `sqrt` es computacionalmente más cara que operaciones como suma, resta, multiplicación y valor absoluto. En CPUs modernas, la diferencia puede ser pequeña (alrededor de 1,5× a 3×), pero en sistemas embebidos o en bucles de millones de iteraciones, cualquier ganancia importa. Por eso, **cuando el objetivo es solo comparar distancias** (p. ej.: encontrar el punto más cercano), usa la **distancia euclidiana al cuadrado**.

1.17.1.2 📋 Tarea (especificación para VPL)

Entrada:
Una única línea con cuatro números reales: `Ax Ay Bx By`

Salida:
Tres líneas, cada una con un número real de dos cifras decimales (Euclidiana, City-block, Chessboard).

1.17.1.3 📌 Ejemplos

Entrada	Salida	Observación
0 0 3 4	5.00 7.00 4.00	Triángulo 3-4-5
0 0 1 1	1.41 2.00 1.00	Diagonal unitaria

Ejemplo de prueba de `sqrt` en Python, con `timeit` aislando cada operación:

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

// Tiempo en segundos de una lambda
template <typename F>
double timeit(F&& f, int N) {
    auto start = std::chrono::high_resolution_clock::now();
    for (int i = 0; i < N; ++i) f();
    auto end = std::chrono::high_resolution_clock::now();
```

```
    return std::chrono::duration<double>(end - start).count();
}

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

    // Solo suma
    auto apenas_soma = []() -> double {
        double a = 3.0, b = 4.0;
        return a + b;
    };

    // Suma y raíz cuadrada
    auto soma_e_sqrt = []() -> double {
        double a = 3.0, b = 4.0;
        return std::sqrt(a*a + b*b);
    };

    double t_soma = timeit(apenas_soma, N);
    double t_sqrt = timeit(soma_e_sqrt, N);

    std::cout.precision(3);
    std::cout << std::fixed;
    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
```

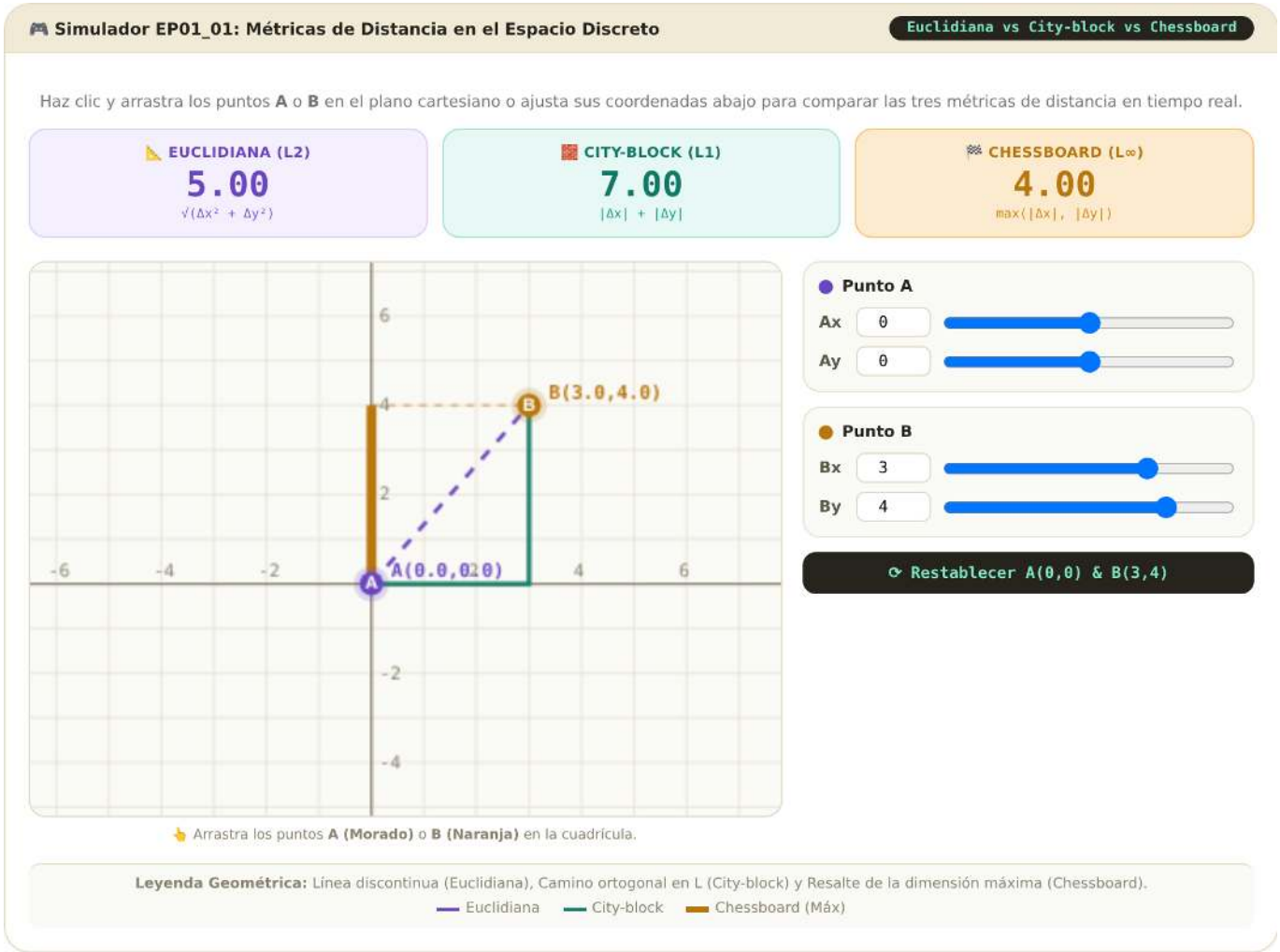
```
Soma simples       : 0.205 s
Soma + sqrt        : 0.444 s
Razão (sqrt/soma)  : 2.166x
```

1.17.1.4 🐍 Python

Basta crear una celda de código normal e insertar el código Python. La entrada se puede simular usando `input()`, que funciona normalmente.

Ejemplo de celda:

```
%%writefile EP01_01.py
# Código Python
x1,y1,x2,y2 = int(input()), int(input()), int(input()), int(input())
# Cálculo de las diferencias
dx = abs(x2 - x1)
dy = abs(y2 - y1)

# 1. Distancia Euclidiana (L2)
dist_euclidiana = (dx**2 + dy**2)**0.5

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

# 3. Distancia Chebyshev / Tablero de ajedrez (Linf)
```



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

Figura 1.11: Simulador EP01_01: Distancias Euclidiana, City-block y Chessboard

```
dist_chessboard = max(dx, dy)

# Salida formateada según los casos de prueba
print(f"{dist_euclidiana:.2f}")
print(f"{dist_city_block:.2f}")
print(f"{dist_chessboard:.2f}")
```

```
Overwriting EP01_01.py
```

```
# Espera que usted escriba 4 números enteros al ejecutar esta celda.
# En Jupyter o Google Colab, la magia %run -i permite que el script lea del teclado.
# En la terminal común, usted usaría: python3 EP01_01.py (sin el '!' y sin '%run').

# %run -i EP01_01.py
```

```
# Envía 4 enteros como entrada estándar (stdin) al script EP01_01.py usando un pipe
!echo -e "0\n0\n4\n4" | python3 EP01_01.py
```

```
5.66
8.00
4.00
```

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

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

1.17.1.5 ☕ Java

Para ejecutar Java en Colab, debes usar una celda con el prefijo `%%writefile` para guardar el código en un archivo, compilar y ejecutar.

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

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

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

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

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

```
Overwriting EP01_01.java
```

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

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

```
4,00
```

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

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

1.17.1.6 💻 C

De forma similar, use `%%writefile` para guardar el código, luego compile y ejecute. Para C, utilizamos el compilador **GCC**.

```
# Instalar el compilador GCC y herramientas de build
# El build-essential incluye gcc, g++, make, etc.
import platform, shutil, subprocess

def instalar_gcc():
    if shutil.which("gcc"):
        print("  GCC ya disponible."); 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 instalado!")

instalar_gcc()
```

```
  GCC ya disponible.
```

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

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

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

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

    return 0;
}
```

```
Overwriting EP01_01.c
```

```
# Compila el archivo .c generando el ejecutable EP01_01
# -lm se usa para enlazar la biblioteca matemática (math.h) si es necesario

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

```
5.66
8.00
```

```
4.00
```

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

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

1.17.1.7 💻 C++

De manera similar a Java, usa `%%writefile` para guardar el código, luego compila y ejecuta. Recuerda que, en Colab, también es necesario instalar lo siguiente:

```
# Instalar el compilador G++ para C++
# El build-essential incluye el g++, make, etc.
import platform, shutil, subprocess

def instalar_gpp():
    if shutil.which("g++"):
        print("  G++ ya disponible."); return
    if platform.system() != "Linux":
        if platform.system() == "Darwin":
            print("  Mac: xcode-select --install")
        else:
            print("  Windows: use 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("  Compilador C++ listo.")

instalar_gpp()
```

```
  G++ ya disponible.
```

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

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

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

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

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

    return 0;
}
```

```
Overwriting EP01_01.cpp
```

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

```
5.66
8.00
4.00
```

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

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

1.17.1.8 🌐 JavaScript (Node.js)

Para JavaScript, use `%%writefile` para crear el archivo y ejecútalo 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

En Colab, se puede ejecutar código R directamente utilizando la magia `%%R`.
El programa debe leer **cuatro números** (x1, y1, x2, y2) y mostrar la distancia euclidiana con **dos decimales**.

Ejemplo de celda:

```
%%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))
```

```
# Instalar R y Rscript
# r-base instala el entorno R completo, incluido Rscript
import platform, shutil, subprocess

def instalar_r():
```

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

instalar_r()
```

```
  R ya disponible.
```

Para probar de la misma manera que en los ejemplos anteriores, se debe usar la entrada estándar (stdin) en la terminal o adaptar el código según lo siguiente:

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

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

```
Overwriting EP01_01.r
```

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

```
5.66
8.00
4.00
```

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

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

1.17.2 EP01_02 📊 Desempeño Predictivo — Métricas de ML en VC

En esta actividad, entrarás al mundo del **Aprendizaje Automático** (*Machine Learning*). Tu objetivo es evaluar el desempeño de un clasificador binario calculando métricas a partir de una **Matriz de Confusión**.

1.17.2.1 🧠 ¿Por qué importa la métrica correcta?

Imagina un detector de **monedas de R$ 1,00**. El impacto del error define la métrica prioritaria:

Métrica	Ejemplo Práctico	Importancia en PDI/VC
Exactitud	**Conteo de Granos**	Útil cuando las clases están equilibradas (ej: la mitad de los granos con defecto, la mitad sanos).

Métrica	Ejemplo Práctico	Importancia en PDI/VC
Precisión	**Seguridad/Biometría**	Crucial para evitar **Falsos Positivos** (ej: no permitir que un impostor acceda a un sistema por error de reconocimiento).
Sensibilidad	**Salud (Tumores)**	Crucial para evitar **Falsos Negativos** (ej: no dejar que un tumor pase desapercibido en un examen de Rayos-X).
Puntuación F1	**Billetes de Dinero**	Ideal para un **equilibrio** entre no rechazar billetes verdaderos y no aceptar billetes falsos.

1.17.2.2 📊 La Matriz de Confusión

	Predicha Positiva	**Predicha Negativa**
Real Positiva	**VP** (Verdadero Positivo)	**FN** (Falso Negativo)
Real Negativa	**FP** (Falso Positivo)	**VN** (Verdadero Negativo)

Tarea:

1. Lee **4 valores enteros** en el orden: `VP`, `FN`, `FP`, `VN`.
2. Calcula las métricas utilizando las fórmulas:

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

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

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

$$\text{Puntuación F1} = \frac{2 \times \text{Precisión} \times \text{Sensibilidad}}{\text{Precisión} + \text{Sensibilidad}}$$

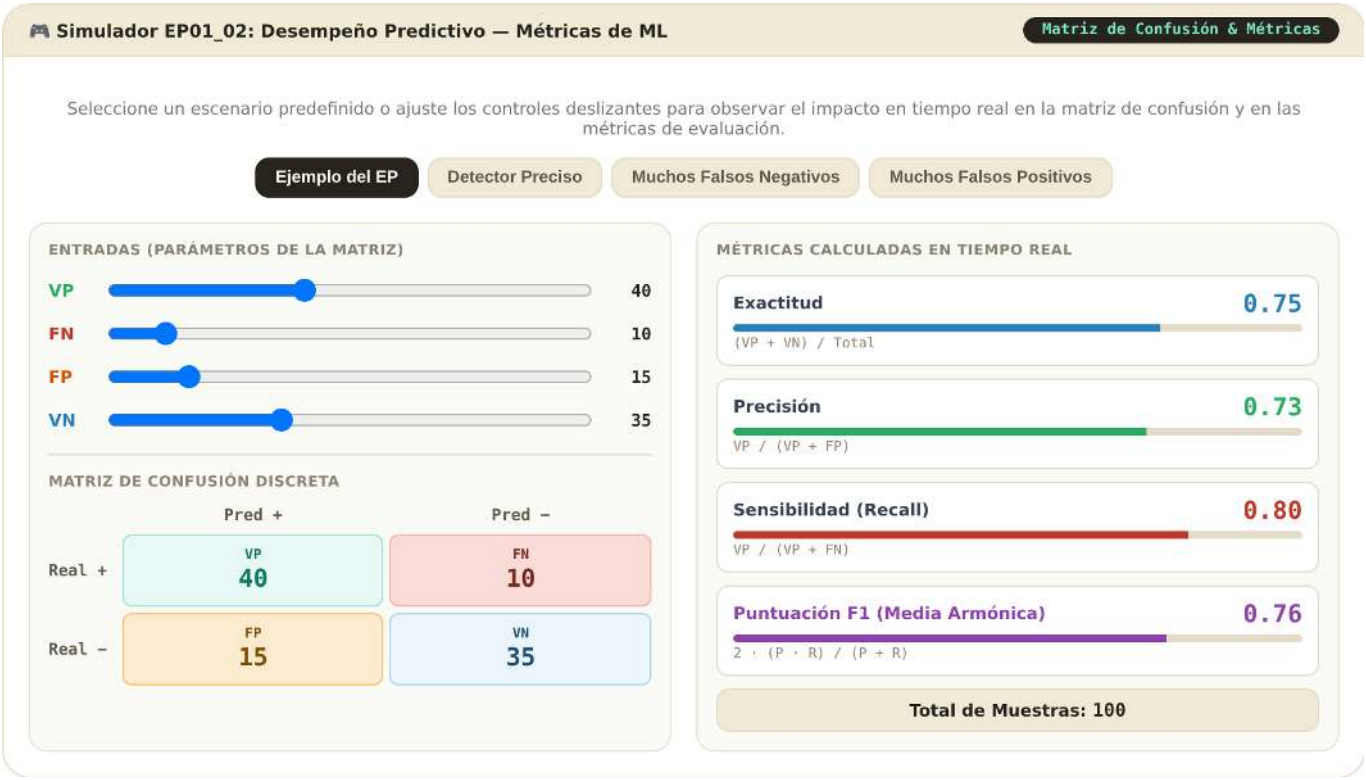
3. Imprime los resultados formateados con **dos decimales**, uno por línea.

📌 **Importante**:

- Usa división en punto flotante para evitar resultados truncados.
- El orden de salida debe ser: **Exactitud**, **Precisión**, **Sensibilidad** y **Puntuación F1**.
- Consulta la simulación interactiva de este EP en la Figura 1.12.

1.17.2.3 📌 Ejemplo de Ejecución

Entrada	Salida	Observación
`40`	`0.75`	Exactitud
`10`	`0.73`	Precisión
`15`	`0.80`	Sensibilidad
`35`	`0.76`	Puntuación F1



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

Figura 1.12: Simulador EP01_02: Rendimiento Predictivo — Métricas de ML

(Nota: VP=40, FN=10, FP=15, VN=35. Total de casos = 100)

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

```
Overwriting EP01_02.cpp
```

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

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

1.17.3 EP01_03 ↗ *Mean Average Precision* (mAP) — Curva Precisión-Sensibilidad

En esta actividad, evaluará un clasificador binario (p. ej., detección de deforestación en imágenes de satélite, ver dgi.inpe.br) mediante la **curva Precisión-Sensibilidad** y la métrica **mAP (*Mean Average Precision*)**. El mAP es estándar en competencias como COCO (*Common Objects in Context*) y PASCAL VOC (*Visual Object Classes*) y de los modelos YOLO (*You Only Look Once*).

1.17.3.1 🧠 ¿Por qué el mAP es la métrica-estándar?

En la EP01_02 vio que la elección del umbral altera significativamente la Precisión y la Sensibilidad. El **mAP (*Mean Average Precision*)** resuelve esto: evalúa el modelo en **varios umbrales** (cada umbral debe generar una matriz de confusión diferente) y resume el desempeño mediante el **área bajo la curva Precisión-Sensibilidad (P-S)**.

Mientras que el F1-*Score* observa un único punto de equilibrio, el mAP considera la curva completa. Cuanto más cercano a **1,0**, mejor es el detector en todos los umbrales y clases (p. ej., monedas de 25, 50 y 1 real).

Métrica	Qué resume	Limitación
F1-*Score*	Equilibrio P × S en un único umbral	Depende del umbral elegido
AP	Área bajo la curva P-S de una clase	Válida solo para una clase
mAP	Promedio de las APs sobre todas las clases	Más complejo de implementar

Referencias: Roboflow — mAP · Vídeo explicativo

1.17.3.2 Cómo se calcula el mAP — paso a paso

1. **Umbrales fijos** (use siempre esta lista):

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

2. Para cada umbral (t), clasifique las muestras: `predicho = 1 si confianza  t, sino 0`. Calcule VP, FP, FN, VN y obtenga Precisión((t)) y Sensibilidad((t)).

3. Construya la curva P-S: pares (Sensibilidad((t)), Precisión((t))), ordenados por Sensibilidad creciente.

4. **Monotonice la Precisión**:

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

5. Calcule la **AP** (área bajo la curva monotónica) usando la **regla del trapecio** (aproximación más precisa que la simple suma de Riemann):

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

6. **mAP** = promedio de las APs de todas las clases. En este EP hay solo **1 clase**, por lo tanto mAP = AP.

> **Nota**
>
> 📐 **Diferencia resumida:**
> La *suma de Riemann* aproxima el área mediante rectángulos, pudiendo subestimar o sobrestimar. La **regla del trapecio** usa trapecios, reduciendo el error al considerar el promedio entre los valores en los extremos del intervalo, siendo generalmente más precisa para funciones suaves por partes, como la curva Precisión-Sensibilidad.

1.17.3.3 📋 Tarea

Lea un entero `n` (cantidad de muestras). Luego lea `n` líneas, cada una con: `verdad` (0 o 1) y `confianza` (float 0.0–1.0).

Calcule e imprima, para el **umbral 0.85** (índice 9 de la lista):

- Matriz de Confusión (VP, FN, FP, VN)
- Exactitud, Precisión, Sensibilidad y F1-Score

A continuación, para **todos los umbrales**, imprima:

- Precisiones crudas, Precisiones monotónicas y Sensibilidades, separadas por `,`
- mAP final

1.17.3.4 📌 Importante

- Umbral fijo para las métricas individuales: **0.85**
- División segura: si el denominador es cero, use 0
- Formato: dos decimales
- Monotonice de atrás hacia adelante
- La Figura 1.13 presenta una simulación de esta cuestión

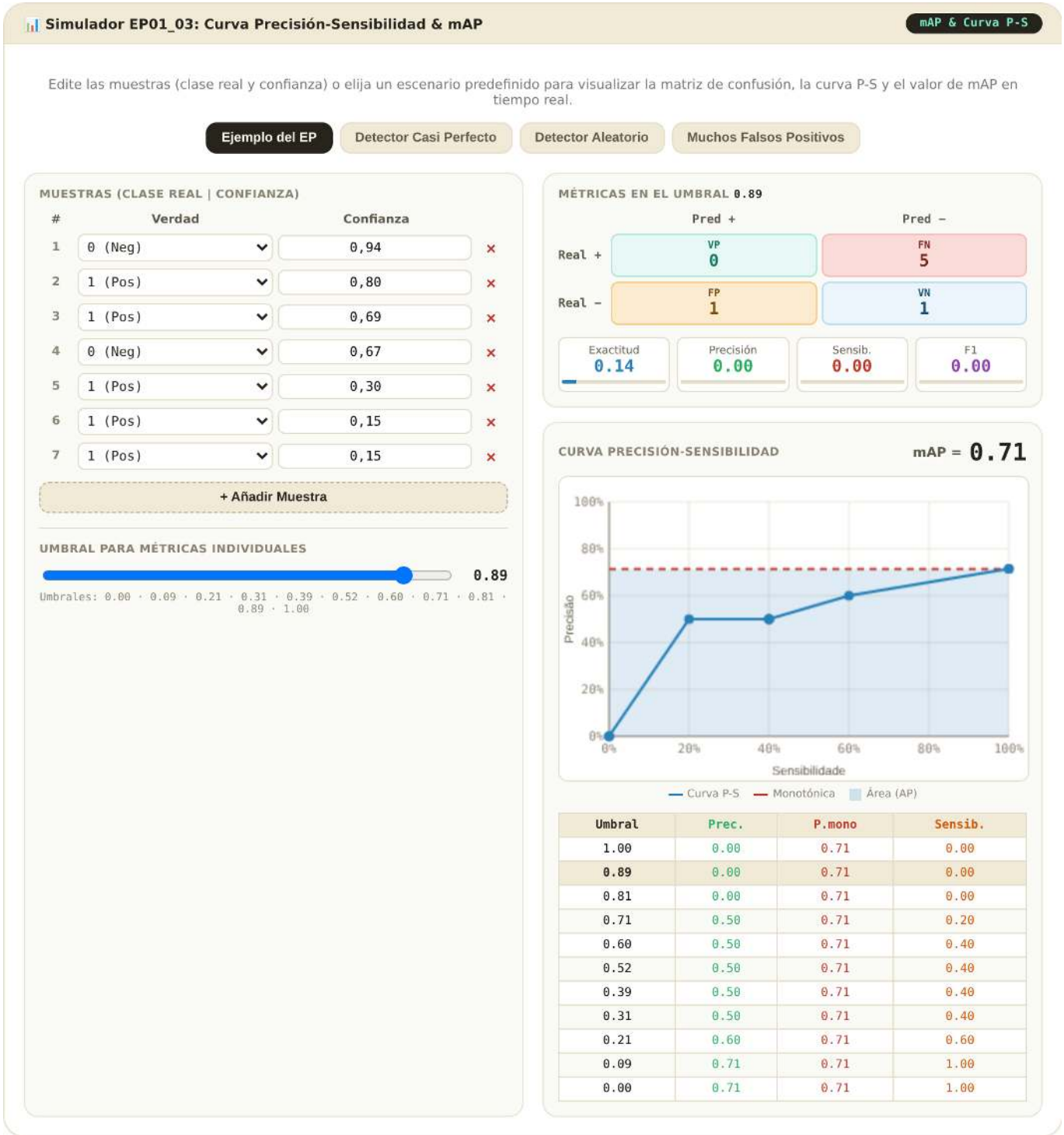
1.17.3.5 📌 Ejemplo de Ejecución

Entrada	Salida Esperada
7 0 0.94 1 0.80 1 0.69 0 0.67 1 0.30 1 0.15 1 0.15	# MÉTRICAS PARA EL UMBRAL 0.85 # Matriz de Confusión: VP = 0, FN = 5 FP = 1, VN = 1 Métricas de Evaluación: Exactitud: 0.14 Precisión: 0.00 Sensibilidad: 0.00 F1-Score: 0.00 # MÉTRICAS PARA TODOS LOS UMBRALES # Precisiones: 0.00, 0.00, 0.00, 0.50, 0.50, 0.50, 0.50, 0.50, 0.60, 0.71, 0.71 Precisiones mon.: 0.71, 0.71, 0.71, 0.71, 0.71, 0.71, 0.71, 0.71, 0.71, 0.71, 0.71 Sensibilidades: 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 🐍 Consejo para calcular el AP (con regla del trapecio)

```python
def calcular_AP(verdades, confianzas, umbrales):
    m = len(umbrales)
    precisiones = [0.0] * m
    sensibilidades = [0.0] * m
    for i in range(m):
        p, s = calcular_metricas(verdades, confianzas, umbrales[i])
        precisiones[m-1-i] = p
        sensibilidades[m-1-i] = s
    prec_mono = precisiones.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):
        # Regla del trapecio: promedio de las alturas por la base
        area_trapecio = (prec_mono[i-1] + prec_mono[i]) / 2.0
        AP += area_trapecio * (sensibilidades[i] - sensibilidades[i-1])
    return precisiones, prec_mono, sensibilidades, AP
```

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



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

Figura 1.13: Simulador EP01_03: Precisión Media Promedio (mAP) y Curva P-S

```
Overwriting EP01_03.cpp
```

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

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

1.17.4 EP01_04 🖼️ Lectura e Información de una Imagen en Matriz

En esta actividad, debes escribir un programa que procese una imagen digital representada como una matriz de píxeles en tonos de gris.

- Lee dos enteros **L** y **C**, que representan el número de filas y columnas.
- Lee los **L * C** valores enteros que componen la matriz de la imagen (cada valor entre 0 y 255).
- Calcula e imprime la siguiente información:

1. El número de filas.
2. El número de columnas.
3. El valor del píxel más grande (**Máximo**).
4. El valor del píxel más pequeño (**Mínimo**).
5. La **Media** aritmética de todos los píxeles.

📌 **Importante**:

- La salida debe seguir exactamente el formato etiquetado (ej.: `Filas: X`).
- El valor de la media debe formatearse con **dos decimales**.
- Consulta un simulador interactivo para esta pregunta en la **Figura 1.14** (cuadrícula interactiva para visualización de intensidades y cálculos en tiempo real).

1.17.4.1 🧠 ¿Por qué es importante? – La Imagen como Datos

Toda imagen digital es, en el fondo, una estructura de datos. En tonos de gris de 8 bits, cada píxel es un valor escalar. Extraer estadísticas básicas es el primer paso para:

Operación	Utilidad Práctica
Máximo/Mínimo	Identificar si la imagen está “lavada” (bajo contraste) o saturada.
Media	Calcular el brillo global de la escena para ajustes de exposición.
Normalización	Redimensionar los valores a intervalos como $[0, 1]$ en redes neuronales.

1.17.4.2 📋 Tarea (especificación para VPL)

Entrada:

La primera línea contiene el entero `L` (filas).

La segunda línea contiene el entero `C` (columnas).

Las líneas siguientes contienen los elementos de la matriz.

Salida:

Cinco líneas formateadas según el ejemplo:

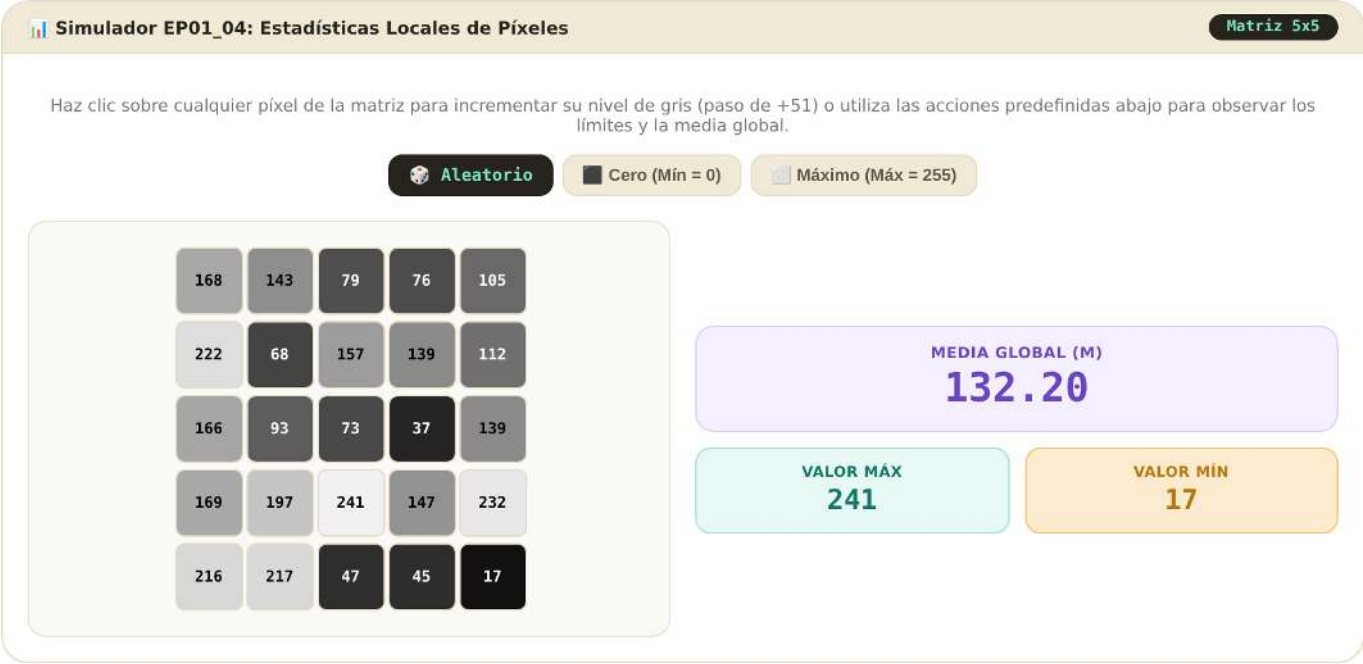
`Filas: L`

`Columnas: C`

```
Max: V

Min: V

Media: V.VV
```

1.17.4.3 📌 Ejemplos

Entrada	Salida	Observación
2 3 0 128 255 50 100 200	Filas: 2 Columnas: 3 Max: 255 Min: 0 Media: 122.17	Imagen pequeña de alto contraste



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

Figura 1.14: Simulador EP01_04: Estadísticas de Píxeles en Matriz 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 de una Imagen en Tonos de Gris

En esta actividad, se debe escribir un programa que calcule el negativo de una imagen digital.

- Lea dos enteros **L** y **C**, que representan el número de filas y columnas.
- Lea los valores enteros que componen la matriz de la imagen.
- Para cada píxel, aplique la transformación de inversión:

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

- Imprima la matriz resultante, manteniendo el formato original (L filas y C columnas).

📌 **Importante**:

- Los valores de cada fila en la salida deben estar separados por un **espacio en blanco**.
- La salida debe contener **solo los números** de la matriz resultante.
- Consulte un simulador interactivo para esta cuestión en la **Figura 1.15** (comparación en tiempo real entre la matriz original y su negativo).

1.17.5.1 🧠 ¿Por qué es importante? – Inversión de Intensidad

El **negativo** es una transformación lineal básica que invierte la escala de brillo. Es una herramienta esencial para que el ojo humano identifique detalles claros que están "ocultos" en fondos más oscuros, siendo ampliamente utilizada en:

Aplicación	Utilidad
Imágenes Médicas	Mejora la visualización de anomalías en tejidos densos (ej: Rayos-X).
Astronomía	Destacar galaxias y nebulosas tenues contra el vacío del espacio.
Artes Digitales	Efectos estéticos y preparación de máscaras de selección.

1.17.5.2 📋 Tarea (especificación para VPL)

Entrada:

La primera línea contiene el entero `L`.

La segunda línea contiene el entero `C`.

Las líneas siguientes contienen los elementos de la matriz.

Salida:

La matriz invertida con `L` filas y `C` columnas.

1.17.5.3 📌 Ejemplos

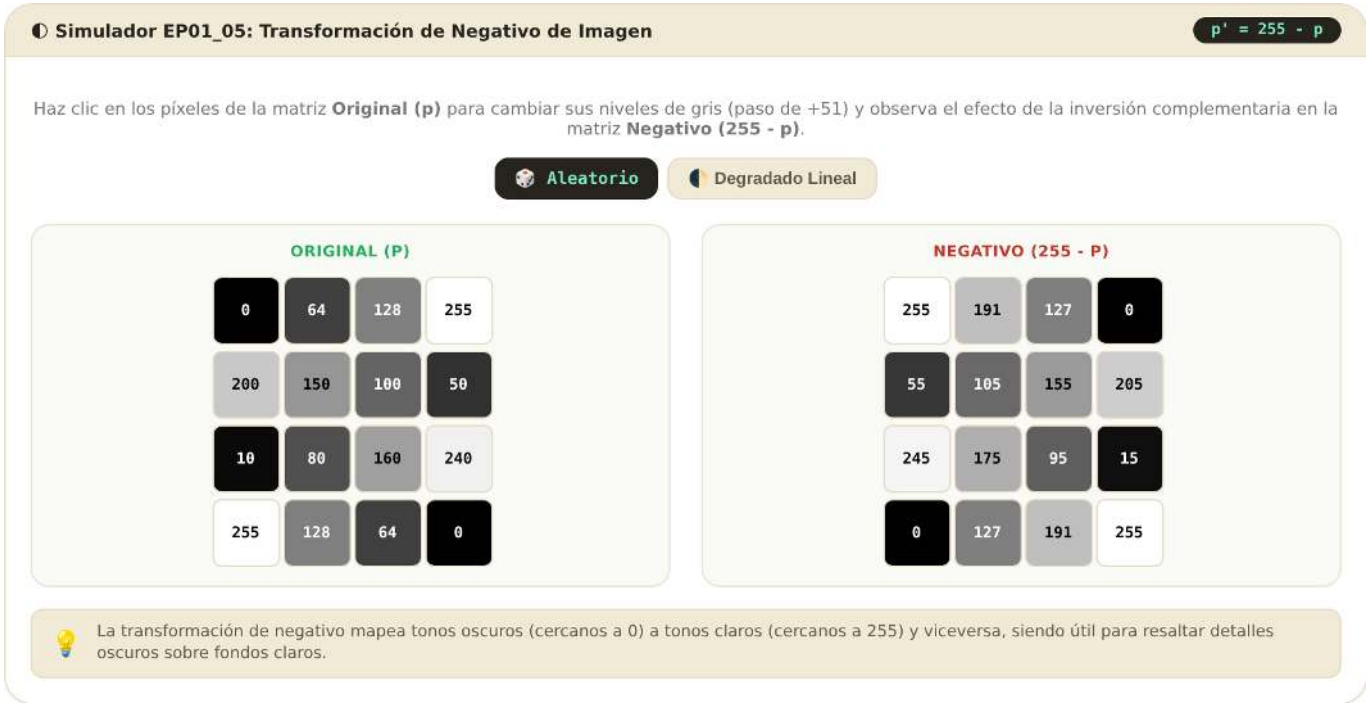
Entrada	Salida	Observación
2 3 0 128 255 50 100 255	255 127 0 205 155 55	Donde era 0 (negro) se convierte en 255 (blanco)

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

```
Overwriting EP01_05.cpp
```

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

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



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

Figura 1.15: Simulador EP01_05: Transformación de Negativo de Imagen (Inversión de Intensidad)

1.17.6 EP01_06 🎨 Conversión RGB → Escala de Grises (ITU-R BT.601)

En esta actividad, debes escribir un programa que convierta píxeles de color (RGB) a escala de grises utilizando la ponderación fisiológica del estándar ITU-R BT.601.

- Lee dos enteros **L** y **C**, que representan el número de filas y columnas.
- Lee **L** × **C** tripletas de enteros, donde cada tripleta representa los canales **R** (Red), **G** (Green) y **B** (Blue) de un píxel.
- Para cada píxel, calcula el valor de gris (g) usando la fórmula:

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

- Imprime la matriz resultante (L filas y C columnas) que contenga los valores enteros convertidos.

📌 **Importante**:

- Utiliza la función `round()` de tu lenguaje para garantizar el redondeo correcto al entero más cercano.
- La salida debe contener únicamente los valores de gris, manteniendo la estructura de matriz (separados por espacios en la línea).
- Consulta un simulador interactivo para esta pregunta en la **Figura 1.16** (ajusta los deslizadores para ver cómo contribuye cada color al brillo final).

1.17.6.1 🧠 ¿Por qué no usar solo el promedio?

El ojo humano no percibe todos los colores con la misma intensidad. Somos mucho más sensibles al **Verde** que al **Azul** debido a nuestra evolución biológica. El estándar ITU-R BT.601 utiliza pesos específicos para crear una imagen en gris que parezca naturalmente correcta para nuestra visión:

Canal	Peso	Percepción Humana
🟢 **Verde**	**58.7%**	Máxima sensibilidad (distinción de follajes).
🔴 **Rojo**	**29.9%**	Sensibilidad media.

Canal	Peso	Percepción Humana
🔵 **Azul**	**11.4%**	Baja sensibilidad (tonos más oscuros).

1.17.6.2 📋 Tarea (especificación para VPL)

Entrada:

La primera línea contiene el entero `L`.

La segunda línea contiene el entero `C`.

Las líneas siguientes contienen tripletas de enteros `R G B` para cada píxel.

Salida:

La matriz de grises con `L` filas y `C` columnas.

1.17.6.3 📌 Ejemplos

Entrada	Salida	Observación
1 3 255 0 0 0 255 0 0 0 255	76 150 29	Observa cómo el Verde (150) es más brillante que el Azul (29)

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

Figura 1.16: Simulador EP01_06: Conversión RGB a Escala de Grises (Ponderación Perceptual 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 ⚫ Umbralización Manual: Imagen Binaria

En esta actividad, debes escribir un programa que realice la segmentación de una imagen mediante la umbralización (*thresholding*).

- Lee dos enteros **L** y **C**, que representan las dimensiones de la matriz.
- Lee un entero **T**, que será el valor del umbral (corte).
- Lee los valores enteros de la matriz.
- Para cada píxel p, aplica la siguiente regla de binarización:

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

- Imprime la matriz resultante que contenga solo los valores 0 o 255.

📌 **Importante**:

- Presta atención al operador: el píxel solo se vuelve blanco (255) si es **estrictamente mayor** que T.
- La salida debe mantener la estructura de matriz (L filas y C columnas).
- Consulta un simulador interactivo para esta pregunta en la **Figura 1.17** (ajusta el control deslizante de T para observar cómo los objetos se aíslan del fondo).

1.17.7.1 🧠 ¿Qué es la Segmentación?

La umbralización es el método más simple para separar objetos de interés del fondo de la imagen. Al transformar tonos de gris en blanco y negro puro, creamos un mapa binario que facilita el conteo de objetos o la identificación de formas:

Valor del Píxel (p)	Condición	Resultado Final
Oscuro ($p \leq T$)	Fondo/Ruido	**0** (Negro)
Claro ($p > T$)	Objeto/Destacado	**255** (Blanco)

1.17.7.2 📋 Tarea (especificación para VPL)

Entrada:

La primera línea contiene el entero `L`.

La segunda línea contiene el entero `C`.

La tercera línea contiene el entero `T` (umbral).

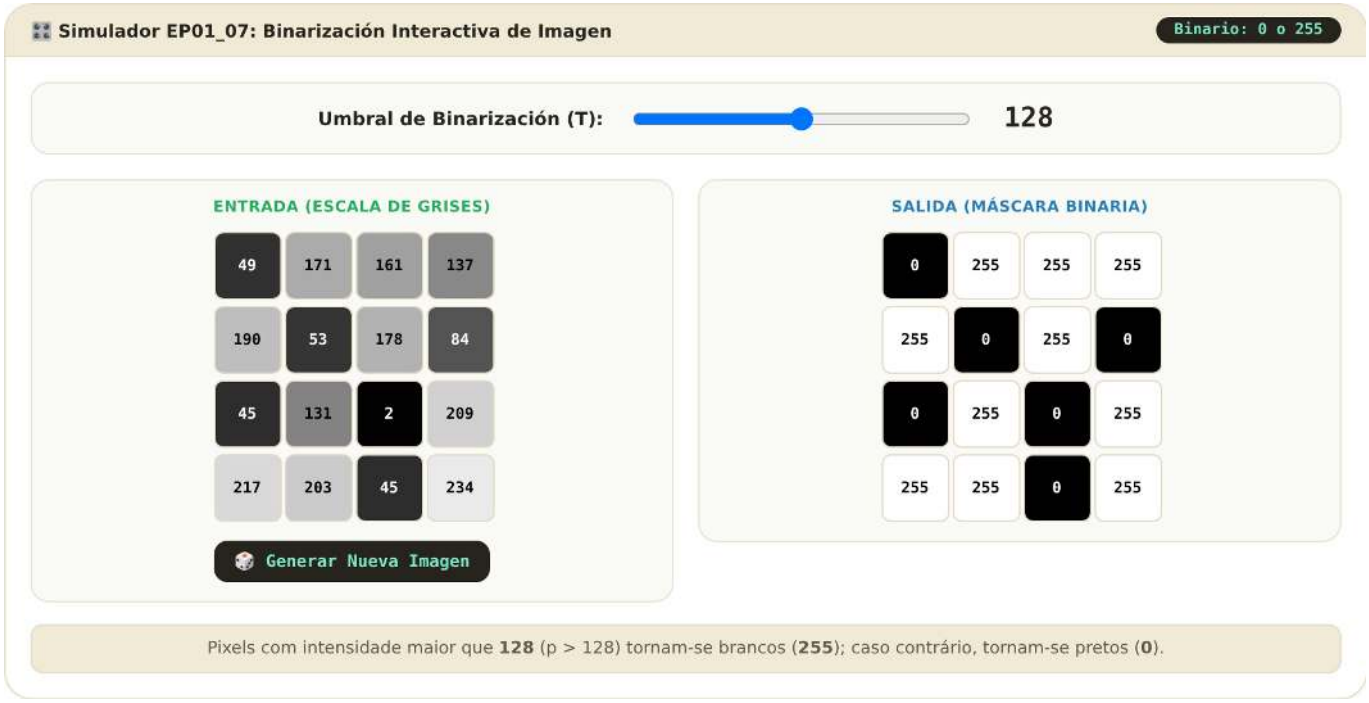
Las líneas siguientes contienen los elementos de la matriz.

Salida:

La matriz binarizada (0 o 255) con `L` filas y `C` columnas.

1.17.7.3 📌 Ejemplos

Entrada	Salida	Observación
2 4 128 0 100 128 200 50 129 255 64	0 0 0 255 0 255 255 0	Observa que el valor 128 se convirtió en 0 (pues $128 \leq 128$)



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

Figura 1.17: Simulador EP01_07: Umbralización Global Interactiva (Binarización p > T)

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

```
Overwriting EP01_07.cpp
```

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

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

1.17.8 EP01_08 🎨 Remapeamento por Rango de Intensidad

En esta actividad, debes escribir un programa que aplique transformaciones lineales distintas a diferentes regiones de intensidad de la imagen.

- Lee dos enteros **L** y **C**, que representan las dimensiones de la matriz.
- Lee los enteros **T** (umbral), (delta 1) y (delta 2).
- Lee los valores enteros de la matriz.
- Para cada píxel p, aplica la regla de remapeo condicional:

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

- Imprime la matriz resultante con los nuevos valores de intensidad.

📌 **Importante**:

- es el offset aplicado a los píxeles oscuros (por debajo del umbral).
- es el offset aplicado a los píxeles claros (mayores o iguales al umbral).
- Los casos de prueba garantizan que el resultado siempre estará en el rango válido de **0 a 255**, por lo tanto, no es necesario manejar saturación ni redondeos.
- La salida debe mantener la estructura de matriz (**L** filas y **C** columnas).

1.17.8.1 🧠 Transformación Condicional de Píxeles

En Procesamiento Digital de Imágenes (PDI), con frecuencia necesitamos tratar regiones de forma independiente. Esta técnica permite, por ejemplo, aclarar solo las sombras de una fotografía (aumentando los píxeles oscuros) sin estallar el brillo de las áreas ya claras, o viceversa.

Rango de Intensidad	Condición	Operación
Píxeles Oscuros	$p < T$	$p + \delta_1$
Píxeles Claros	$p \geq T$	$p + \delta_2$

1.17.8.2 📋 Tarea (especificación para VPL)

Entrada:

La primera línea contiene los enteros `L` y `C`.

La segunda línea contiene los enteros `T`, y .

Las líneas siguientes contienen los elementos de la matriz.

Salida:

La matriz transformada con `L` filas y `C` columnas, con valores separados por espacios.

1.17.8.3 📌 Ejemplos

Entrada	Salida	Observación

`2 4`
`128 60 -40`
`0 100 150 255`
`80 128 200 30`
	`60 160 110 215`
	`140 88 160 90`
		Píxeles < 128 suman 60. Píxeles 128 restan 40.

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

```
Overwriting EP01_08.cpp
```

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

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

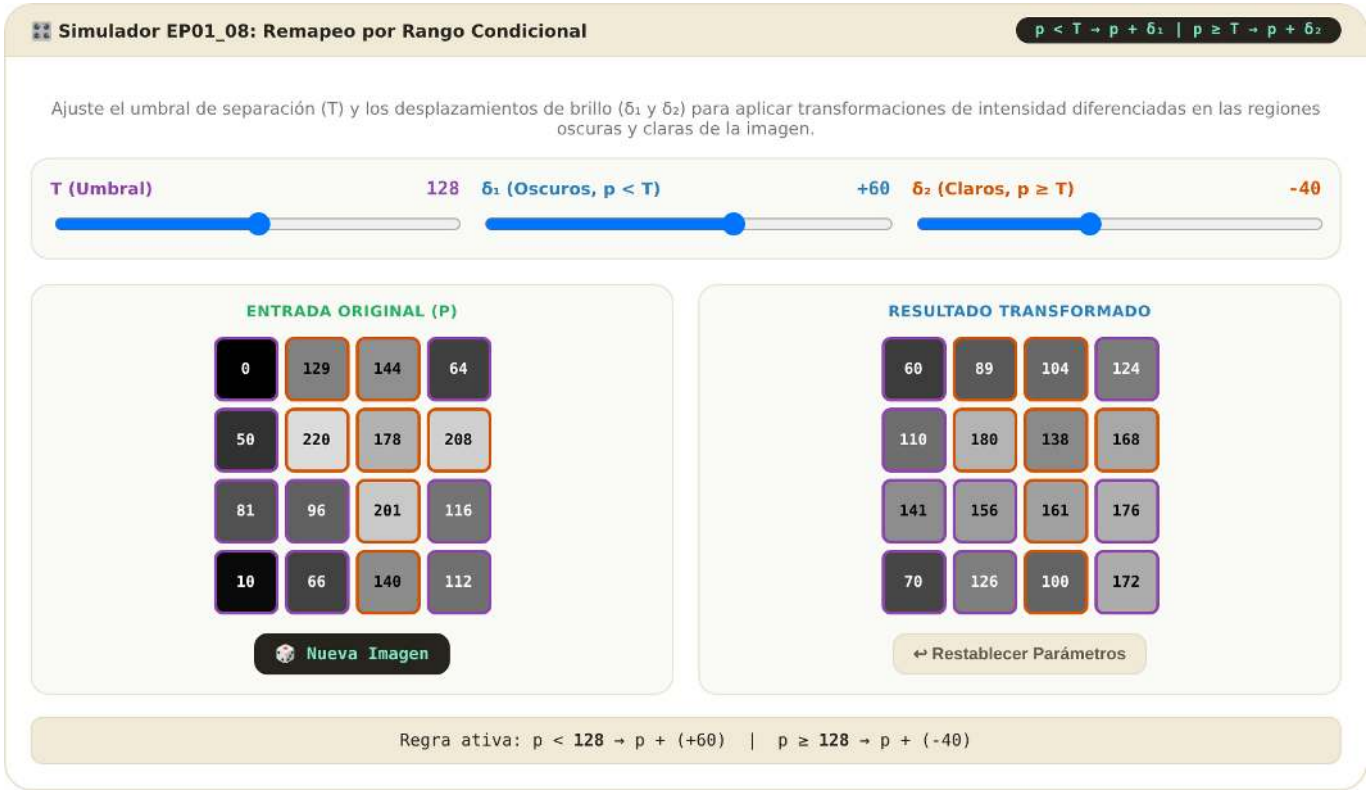
1.17.9 EP01_09 🏁 Patrón de Tablero: Matriz de Ajedrez

En esta actividad, debes escribir un programa que genere una imagen sintética con el patrón de un tablero de ajedrez.

- Lee dos enteros **L** (filas) y **C** (columnas).
- Genera una matriz donde los valores alternan entre **0** (negro) y **1** (blanco).
- La lógica de relleno debe seguir la regla de paridad:
- El elemento en la posición $(0, 0)$ siempre es **0**.
- Un píxel en la posición (i, j) será **1** si la suma de los índices $(i + j)$ es **impar**.
- Un píxel en la posición (i, j) será **0** si la suma de los índices $(i + j)$ es **par**.

📌 **Importante**:

- Los colores deben alternar correctamente tanto horizontal como verticalmente.



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

Figura 1.18: Simulador EP01_08: Remapeo por Rango Condicional (Brillo y Contraste por Umbral)

- La salida debe ser la matriz impresa línea por línea, con los elementos separados por un espacio.
- Consulta un simulador interactivo para esta cuestión en la **Figura 1.19** (ajusta las dimensiones para visualizar la construcción de la malla y la salida textual correspondiente).

1.17.9.1 🧠 Patrones Sintéticos

Crear patrones geométricos es un ejercicio fundamental para dominar la **lógica de índices** en matrices. En Procesamiento Digital de Imágenes, el patrón de ajedrez no solo es estético; también se utiliza ampliamente para:

Aplicación	Utilidad
Calibración de Cámara	Estimar parámetros intrínsecos y extrínsecos de la lente.
Corrección de Distorsión	Identificar y corregir el efecto de “barril” o “cojín” en lentes gran angulares.
Mapeo 3D	Proyectar patrones conocidos para reconstruir superficies en sistemas de luz estructurada.

1.17.9.2 📋 Tarea (especificación para VPL)

Entrada:

Una línea que contiene el entero `L` (filas).

Una línea que contiene el entero `C` (columnas).

Salida:

La matriz de ajedrez con `L` filas y `C` columnas, impresa con espacios entre los elementos.

1.17.9.3 📌 Ejemplos

Entrada	Salida	Observación
3 4	0 1 0 1 1 0 1 0 0 1 0 1	Observa que cada fila comienza con el inverso de la anterior

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

Figura 1.19: Simulador EP01_09: Generador de Patrón de Cuadros (Lógica de Paridad (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 📄 Metadatos: Lectura de Archivo PGM

En esta actividad, debes leer un archivo de imagen en el formato **PGM (Portable Gray Map)** y extraer sus dimensiones a partir del encabezado.

- El formato **PGM (P2)** es un archivo de texto simple (ASCII) que almacena imágenes en tonos de gris.
- El archivo posee un **encabezado** estructurado de la siguiente forma:

1. **Versión:** El identificador `P2`.
2. **Comentarios:** Líneas opcionales que comienzan con `#` (deben ignorarse).
3. **Dimensiones:** Dos enteros que representan **Ancho** y **Alto**.
4. **Máximo:** Un entero que representa la intensidad máxima (generalmente 255).

- Después del encabezado, siguen los datos de los píxeles.

📌 **Importante**:

- **Lectura de Archivo:** Debes abrir el archivo indicado en el ejemplo utilizando la función `open()` de Python.
- **Orden de Salida:** A diferencia del orden presente en el archivo, la salida esperada debe estar en el formato de tupla: `(Altura, Ancho, Canales)`.
- Como los archivos PGM son en tonos de gris, el número de **Canales** es siempre **1**.
- Ver un simulador interactivo para esta cuestión en la **Figura 1.20** (ajusta las dimensiones para ver cómo se genera el encabezado ASCII).

1.17.10.1 🧠 Entendiendo el formato PGM

El formato PGM es uno de los más simples para el procesamiento de imágenes. Por ser en texto puro, permite visualizar los metadatos e incluso los valores de los píxeles abriendo el archivo en un bloc de notas:

Componente	Ejemplo	Significado
Número Mágico	`P2`	Identifica que es un PGM en formato de texto (ASCII).
Comentario	`# CREATOR...`	Línea informativa ignorada por el procesador.
Dimensiones	`397 343`	**397** columnas (Ancho) y **343** filas (Altura).
Intensidad	`255`	Define el valor del blanco puro (escala de 0 a 255).

1.17.10.2 📋 Tarea (especificación para VPL)

Entrada:

Ninguna entrada mediante teclado. El programa debe leer el archivo `"aula01fig03b.pgm"` presente en el directorio de ejecución.

Salida:

Una tupla que contenga `(Altura, Ancho, 1)`.

1.17.10.3 📌 Ejemplos

Nombre del Archivo	Salida Esperada	Observación
“aula01fig03b.pgm”	(343, 397, 1)	Nota la inversión del orden: Altura primero

> **Nota**
>
> El archivo necesario para este EP se descargará automáticamente del repositorio mediante el siguiente código:

```
%%writefile tmp/mm_out_2.cpp
#include <iostream>
#include <string>
// No matter how much we try, C++ does not have a direct equivalent for
// downloading files with urllib. We'll simulate the structure.

int main() {
    // Constants
```

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

Figura 1.20: Simulador EP01_10: Estructura de Archivo PGM (Encabezado ASCII P2 y Mapeo a Tupla Python)

```
    const std::string BASE_URL =
"https://raw.githubusercontent.com/fzampirolli/pdi-vc/master/all/cap01/imagens";
    const std::string file = "aula01fig03b.pgm";

    // Simulating the download process (no actual download in C++ standard library)
    std::cout << "Simulating download of: " << file << "\n";

    // Simulating the loop over files
    std::string url = BASE_URL + "/" + file;
    std::cout << "URL: " << url << "\n";
    std::cout << "  Arquivo baixado: " << 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
```

```
Simulating download of: aula01fig03b.pgm
URL:
https://raw.githubusercontent.com/fzampirolli/pdi-vc/master/all/cap01/imagens/aula01fig03b.pgm
  Arquivo baixado: 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 ↗ Análisis de Vecindad: Filtro de Máximo 1D

En esta actividad, debes implementar un filtro morfológico simple de máximo que opera sobre una señal unidimensional (vector).

- Lee un entero **n**, que representa el tamaño del vector.
- Lee los **n** elementos enteros que componen el vector original **v1**.
- Crea un nuevo vector **v2**, donde cada posición i es el resultado de la comparación entre el elemento actual y sus vecinos inmediatos:

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

📌 **Importante**:

- **Bordes:** En los extremos del vector (índices 0 y $n-1$), la vecindad posee solo dos elementos (el propio y el único vecino disponible). En el índice 0, compara únicamente $v1[0]$ y $v1[1]$. En el último índice, compara solo $v1[n-2]$ y $v1[n-1]$.
- **Salida:** Imprime el encabezado "v2:" seguido de los valores del vector resultante, uno por línea.
- Consulta un simulador interactivo para esta pregunta en la **Figura 1.21** (pasa el mouse sobre los resultados para visualizar la ventana de vecindad utilizada en el cálculo).

1.17.11.1 🧠 ¿Por qué analizar vecinos?

En procesamiento de imágenes, el valor de un píxel rara vez es aislado; depende del contexto que lo rodea. El **Filtro de Máximo** es la base de la operación de **Dilatación** en morfología matemática, y sirve para:

Función	Efecto Visual
Realce	Expande estructuras brillantes y "engorda" objetos claros.
Eliminación de Ruido	Elimina pequeños puntos negros (ruido de "sal y pimienta" oscuro).
Relleno	Cierra pequeños agujeros o huecos en formas binarias.

1.17.11.2 📋 Tarea (especificación para VPL)

Entrada:

Un entero `n`.

En las líneas siguientes, los `n` elementos enteros del vector.

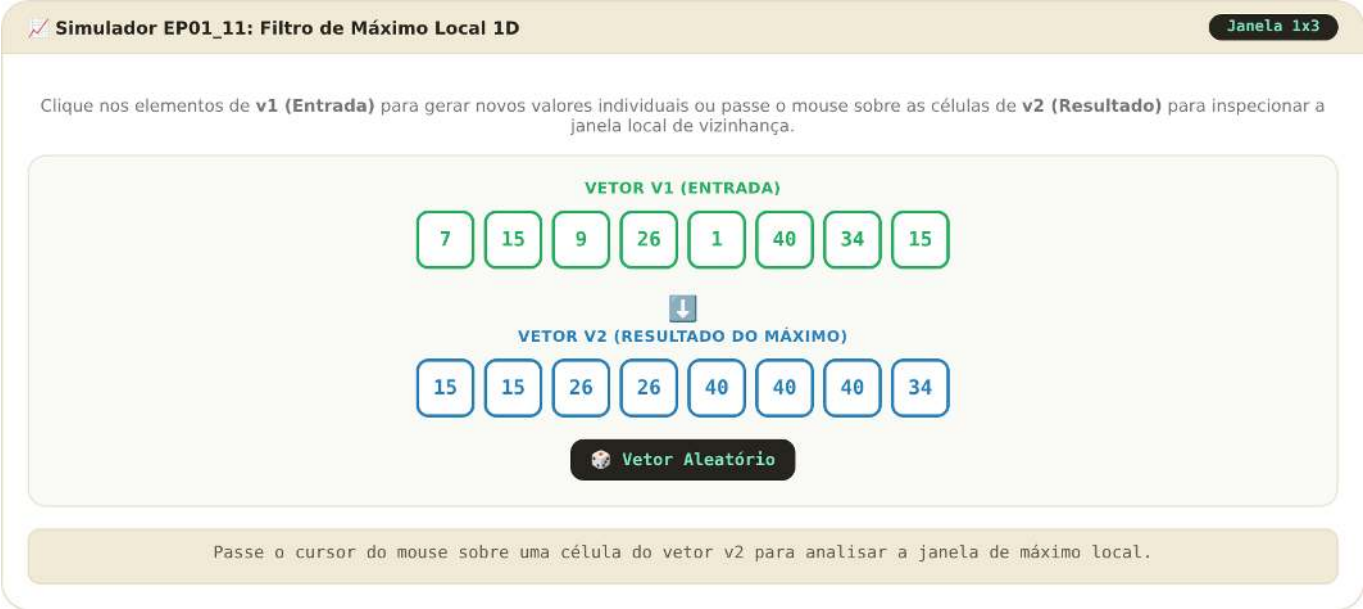
Salida:

La *cadena* `v2:` en la primera línea.

En las líneas siguientes, cada elemento de `v2` (uno por línea).

1.17.11.3 📌 Ejemplos

Entrada	Salida	Observación
5	v2:	En el índice 1: max(10, 20, 5) = 20
10	20	
20	20	
5	30	
30	30	
15	30	



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

Figura 1.21: Simulador EP01_11: Filtro de Máximo Local 1D (Vecindad 1x3 con Condición de Borde)

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

```
Overwriting EP01_11.cpp
```

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

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

Capítulo 2

De la Captura al Píxel - Muestreo, Cuantización y Conectividad

Executar Colab Abrir GitHub

Este capítulo profundiza en la comprensión de la imagen digital, transitando desde la naturaleza física de la captura hasta su representación matemática discreta. Investigamos cómo la luz se convierte en dato y cómo la organización espacial de los píxeles define las relaciones de vecindad y conectividad esenciales para algoritmos avanzados de Visión por Computador (VC).

2.1 Objetivos

Al final de este capítulo, podrá:

- Explicar el modelo físico de formación de la imagen basado en iluminación y reflectancia.
- Diferenciar los mecanismos de la visión humana y de los sensores digitales.
- Comprender los procesos de **muestreo** (discretización del espacio) y **cuantización** (discretización de la intensidad).
- Describir las relaciones topológicas entre píxeles: vecindad, adyacencia, conectividad y distancias.
- Realizar transformaciones geométricas básicas (traslación, rotación, escala) preservando la calidad.
- Visualizar en la práctica los efectos de la variación de la resolución espacial y de la profundidad de bits.

2.2 El Ojo y la Cámara - Elementos de la Percepción Visual

La formación de una imagen digital comienza con la captura de la luz reflejada por los objetos. Como se ilustra en Figura 2.1, tanto el ojo humano como las cámaras digitales siguen principios ópticos similares para enfocar la luz sobre una superficie sensible, aunque utilizan mecanismos biológicos y electrónicos distintos para la transducción de la señal.

2.2.1 Visión humana

El ojo funciona como un sistema óptico complejo: la luz atraviesa la córnea, el humor acuoso, la pupila (controlada por el iris) y el cristalino —que ajusta el enfoque dinámicamente— hasta llegar a la retina. En la retina se encuentran los fotorreceptores: los **conos** (6 millones), concentrados en la fóvea, son responsables de la visión de colores y detalles, mientras que los **bastones** (120 millones) garantizan la visión en condiciones de baja luminosidad (visión escotópica), detectando únicamente intensidades de gris. El punto ciego es la región de donde parte el nervio óptico, carente de receptores.

2.2.2 Sensores digitales

En las cámaras, los sensores de imagen desempeñan el papel de la retina. Los dos tipos más comunes son el **CCD** (*Charge-Coupled Device* - Dispositivo de Carga Acoplada) y el **CMOS** (*Complementary Metal-Oxide-Semiconductor* - Semiconductor de Óxido Metálico Complementario). El sensor está compuesto por una matriz de fotositios (píxeles) que acumulan carga eléctrica proporcional a la luz incidente.

Para la reconstrucción de colores, se utiliza el **Filtro de Bayer**, una matriz de filtros de color que permite que cada píxel capture únicamente un componente de color: rojo, verde o azul (RGGB - *Red, Green, Green, Blue*). Posteriormente, un **ADC** (*Analog-to-Digital Converter* - Convertidor Analógico-Digital) cuantifica esa carga en valores numéricos, definidos por una profundidad de bits (p. ej.: 8 bits, lo que resulta en 256 niveles de intensidad).

> **Curiosidad**
>
> Aunque el ojo humano tiene millones de receptores, la resolución de alta definición se restringe a la fóvea (visión central), equivalente a aproximadamente 120×120 píxeles. La percepción de una escena completa en alta resolución es el resultado de un intenso posprocesamiento realizado por el cerebro.

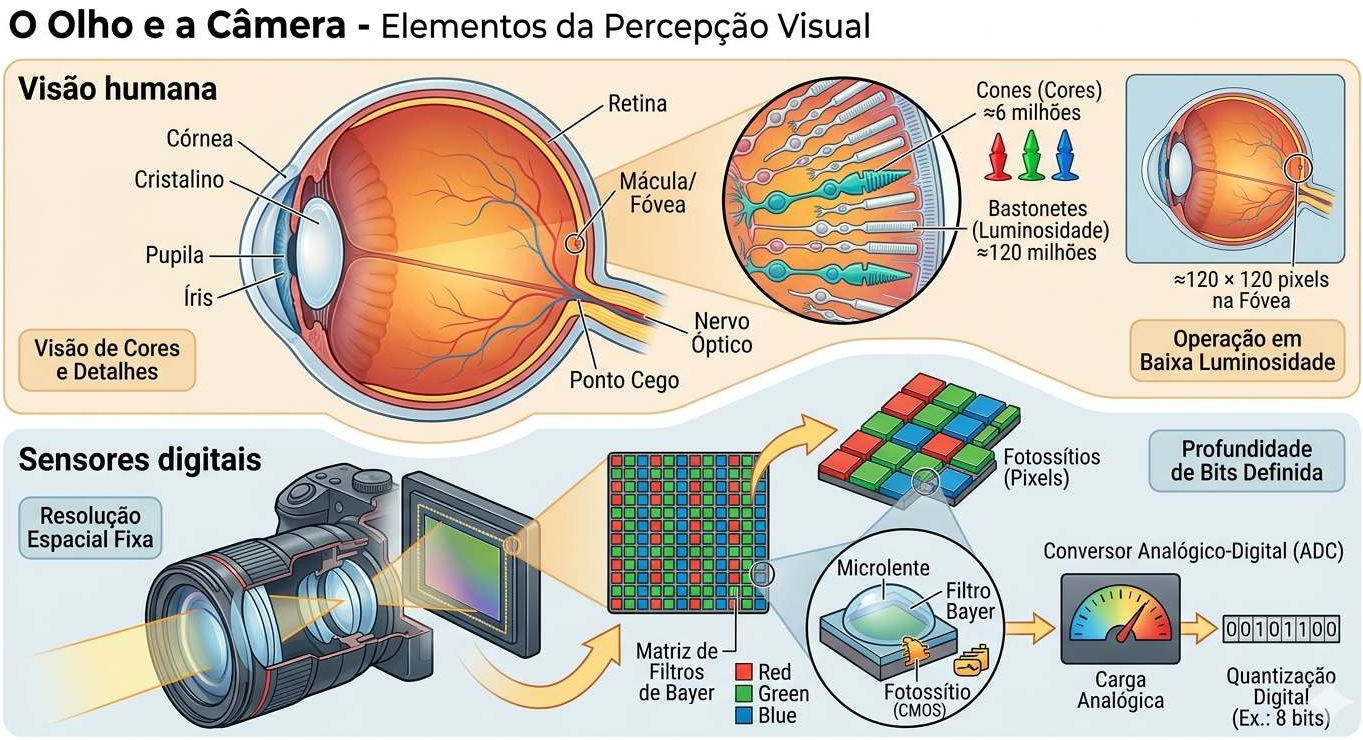


Figura 2.1: Comparativo didáctico entre el sistema visual biológico y el electrónico: en la parte superior, la anatomía del ojo humano destacando la retina y los fotorreceptores (conos y bastones); debajo, la estructura de una cámara digital detallando el sensor CMOS, la matriz de filtros de Bayer (RGGB) y el proceso de cuantización digital realizado por el ADC.

2.3 Ilusiones Ópticas: Los Desafíos de la Percepción Visual

Mientras que los sensores digitales capturan la intensidad de la luz de forma lineal y objetiva, el sistema visual humano interpreta la escena basándose en el contexto, las experiencias previas y los mecanismos biológicos de supervivencia. Las ilusiones ópticas no son "errores" del ojo, sino evidencias del intenso **posprocesamiento cerebral** realizado en la corteza visual.

2.3.1 Ambigüedad y Contexto

El cerebro busca constantemente dar sentido a patrones ambiguos. En el ejemplo del **Vaso de Rubin** (ver Figura 2.2), la percepción alterna entre la figura (el vaso) y el fondo (dos rostros), demostrando que no podemos procesar ambas interpretaciones simultáneamente. Por su parte, la ilusión del **Elefante de Shepard** juega con nuestra incapacidad de reconciliar líneas de contorno que sugieren volumen en posiciones lógicamente imposibles.

2.3.2 Brillo y Contraste Local

Muchas ilusiones derivan de la **inhibición lateral**, mecanismo mediante el cual las neuronas vecinas en la retina compiten entre sí para resaltar bordes. En la **Cuadrícula Scintillating**, puntos oscuros "fantasma" parecen aparecer en las intersecciones blancas debido a este procesamiento local de contraste.

La ilusión de la **Sombra del Tablero de Adelson** es quizás la más impactante para la VC: el cuadrado "A" y el cuadrado "B" poseen exactamente el mismo valor de gris en el sensor (o archivo digital), pero el cerebro "corrige" el brillo de "B" al comprender que se encuentra bajo una sombra proyectada, percibiéndolo como más claro.

2.3.3 Geometría y Perspectiva

La percepción de profundidad puede ser engañada por construcciones geométricas que desafían la lógica tridimensional desde un ángulo de visión específico. La **Escalera de Schröder** utiliza la ambigüedad de la perspectiva para crear un objeto que parece ascender o descender dependiendo de cómo se observe, evidenciando cómo nuestra interpretación de "arriba" y "abajo" depende del punto de fuga.

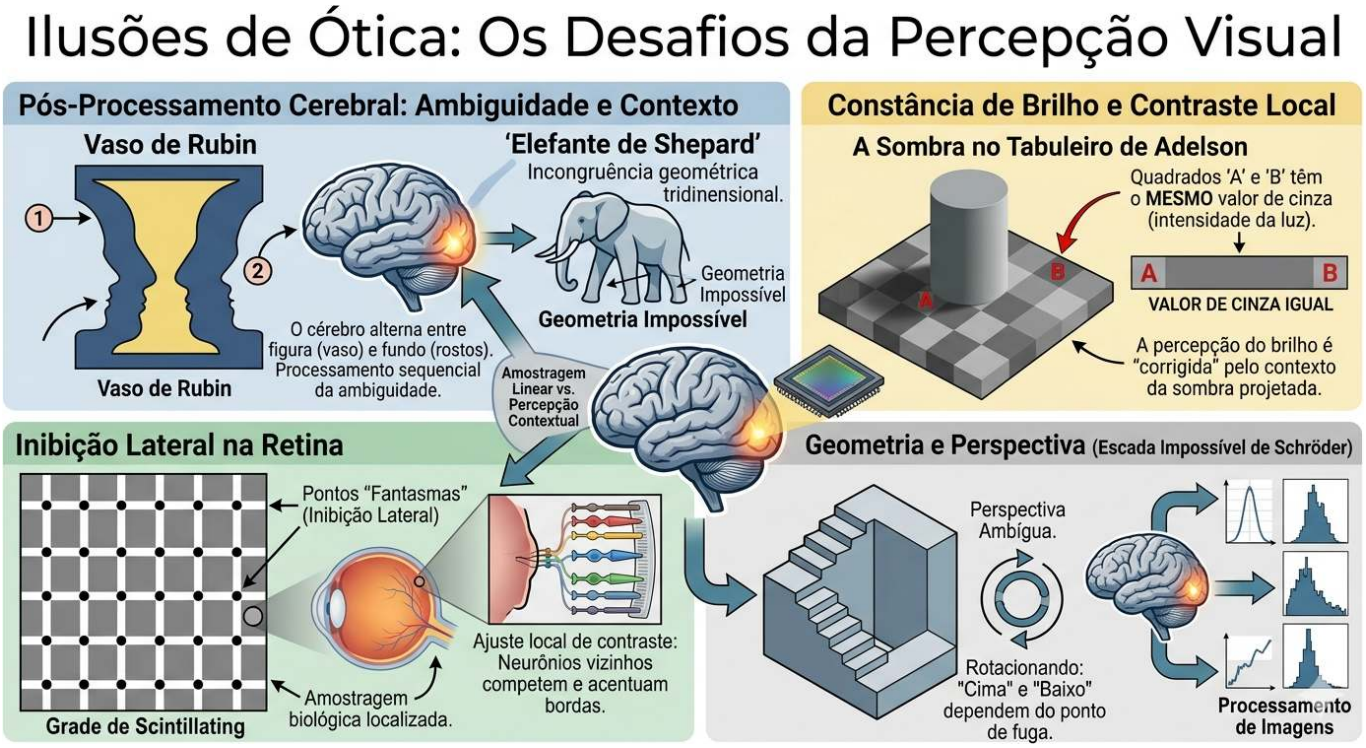


Figura 2.2: Colección de desafíos perceptivos: (superior izquierda) Vaso de Rubin — ambigüedad figura-fondo; (superior central) Elefante de Shepard — incongruencia geométrica; (superior derecha) Sombra de Adelson — constancia de brillo basada en el contexto; (inferior izquierda) Cuadrícula de puntos — inhibición lateral; (inferior derecha) Escalera de Schröder.

2.4 El Modelo Matemático de la Formación de la Imagen

Una imagen puede modelarse como el producto de dos funciones:

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

donde:

- $i(x, y)$ es la **iluminación** incidente sobre la escena (energía luminosa por unidad de área), determinada por la fuente de luz;

- $r(x, y)$ es la **reflectancia** del objeto (fracción de la luz reflejada), determinada por las propiedades ópticas de la superficie, con $0 < r(x, y) < 1$.

En la práctica, los dos componentes varían en escalas espaciales distintas: $i(x, y)$ tiende a variar lentamente en el espacio, mientras que $r(x, y)$ puede presentar variaciones abruptas asociadas a texturas, bordes y detalles finos. Las técnicas de PDI frecuentemente buscan separar o compensar estos componentes, como en métodos de corrección de iluminación no uniforme.

La Figura 2.3 ilustra cómo este modelo se manifiesta en imágenes digitales en color, representadas por múltiples canales espectrales (RGB) y, opcionalmente, por un canal adicional de transparencia (RGBA).

2.4.1 Representación Digital y Canales Espectrales

En imágenes digitales de color, la función $f(x, y)$ de la Ecuación 2.1 se representa mediante múltiples canales espectrales. En el estándar **RGB**, cada píxel almacena tres muestras independientes:

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

correspondientes a las intensidades de los componentes rojo (*Red*), verde (*Green*) y azul (*Blue*). En imágenes del tipo **RGBA**, se añade un cuarto canal:

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

donde $A(x, y)$ representa el canal **alfa** (*Alpha*), responsable de codificar la opacidad del píxel. Por convención, el valor $A = 0$ indica un píxel totalmente transparente, mientras que $A = 255$ (o 1) representa un píxel totalmente opaco.

Así, una imagen digital de color se estructura como una matriz multidimensional de dimensiones:

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

en la que cada posición (x, y) almacena las muestras asociadas a las propiedades ópticas de esa coordenada espacial.

La conversión entre rangos de intensidad es directa y se expresa como:

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

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

Convención de representación en `morph.hpp`: la biblioteca de este libro adopta una convención única (Tabla 2.1), sin las ambigüedades de rango y orden de canales que surgen al combinar varias bibliotecas de Python.

Tabla 2.1: Convención de representación de imágenes en morph.hpp — rango, orden de bandas, nº de canales y disposición de memoria.

Aspecto	`morph.hpp`
Tipo de muestra	`unsigned char` (`uint8`), rango $[0, 255]$
Orden de las bandas	**RGB** (vía `stb_image`; sin inversión BGR)
Nº de canales	`1` (escala de grises) o `3` (RGB)
Disposición en memoria	$H \times W \times C$ intercalado (`data[(y*w + x)*channels + c]`)

La `struct Image` guarda los píxeles en un `std::vector<unsigned char>` lineal, con los campos `h`, `w` y `channels`. La función `mm::read` decodifica con `stb_image` forzando 3 canales (o 1, cuando `grayscale=true`); por lo tanto, **un posible canal alfa del archivo se descarta en la lectura**. Para trabajar en coma flotante, vale la misma relación $f_{\text{norm}} = f/255$.

En la celda de ejemplo a continuación, la versión RGBA ($M \times N \times 4$) se construye apilando un canal alfa sintético sobre el array leído — el kernel del notebook es Python, y la visualización recibe el `ndarray` en ese diseño $H \times W \times C$.

2.4.2 Preparando el Entorno Práctico

El siguiente bloque carga el módulo `morph.py` del repositorio y demuestra, para un píxel sintético, las diferentes convenciones de escala y orden de canales adoptadas por las principales bibliotecas de PDI.

```
import os, urllib.request

os.makedirs("tmp/state", exist_ok=True)  # artefatos de build del camino C++ (.cpp, binario,
PNGs)

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

# El kernel es Python incluso en el camino C++: `mm` (morph.py) es usado por los
# simuladores, por la exhibición de las figuras que el binario C++ genera y por el
# estado mm::Image entre celdas. cpp=True descarga también el camino compilado
# (morph.hpp + stb_image*.h), usado en el #include de las celdas %%writefile *.cpp.
import config
config.setup(demo=True, cpp=True)
from morph import mm
```

```
Entorno listo. Morph: 1.1.9 | OpenCV: 5.0.0

Convenciones de escala por biblioteca
NumPy / Pillow / OpenCV (uint8): [200 100  50] → [0, 255]
scikit-image / PyTorch (float):  [0.78431373 0.39215686 0.19607843] → [0.0, 1.0]
OpenCV: atención - lee en BGR:   [ 50 100 200] (canales invertidos)
```

2.4.3 Implementación de la Lectura de Imágenes en `morph.hpp`

El siguiente fragmento muestra el código fuente de la función `mm::read`, permitiendo verificar directamente cómo la biblioteca `morph.hpp` implementa la lectura de imágenes.

```
# A morph.hpp es header-only; a continuación, el cuerpo de la función 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 y RGBA: Ejemplo Práctico con la Imagen Mandrill

El siguiente ejemplo carga la imagen Mandrill en formato RGB, construye artificialmente un canal alfa con gradiente horizontal y muestra ambas representaciones, ilustrando concretamente las estructuras matriciales $M \times N \times 3$ y $M \times N \times 4$.

```
%%writefile tmp/fig_02_rgb_rgba.cpp
#define MM_OUT "tmp/fig_02_rgb_rgba.png"
// Compile with: g++ -std=c++17 -o program program.cpp morph.hpp

//| 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 <vector>
#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";

    // La descarga y caché se simplifica: mm::read acepta URL directamente
    mm::Image img_rgb = mm::read(url);                      // (M, N, 3), uint8

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

    // Empila 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 " << img_rgb.channels <<
std::endl;
    std::cout << "RGBA  -> shape=" << h << " x " << w << " x " << img_rgba.channels <<
std::endl;

    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)")
```

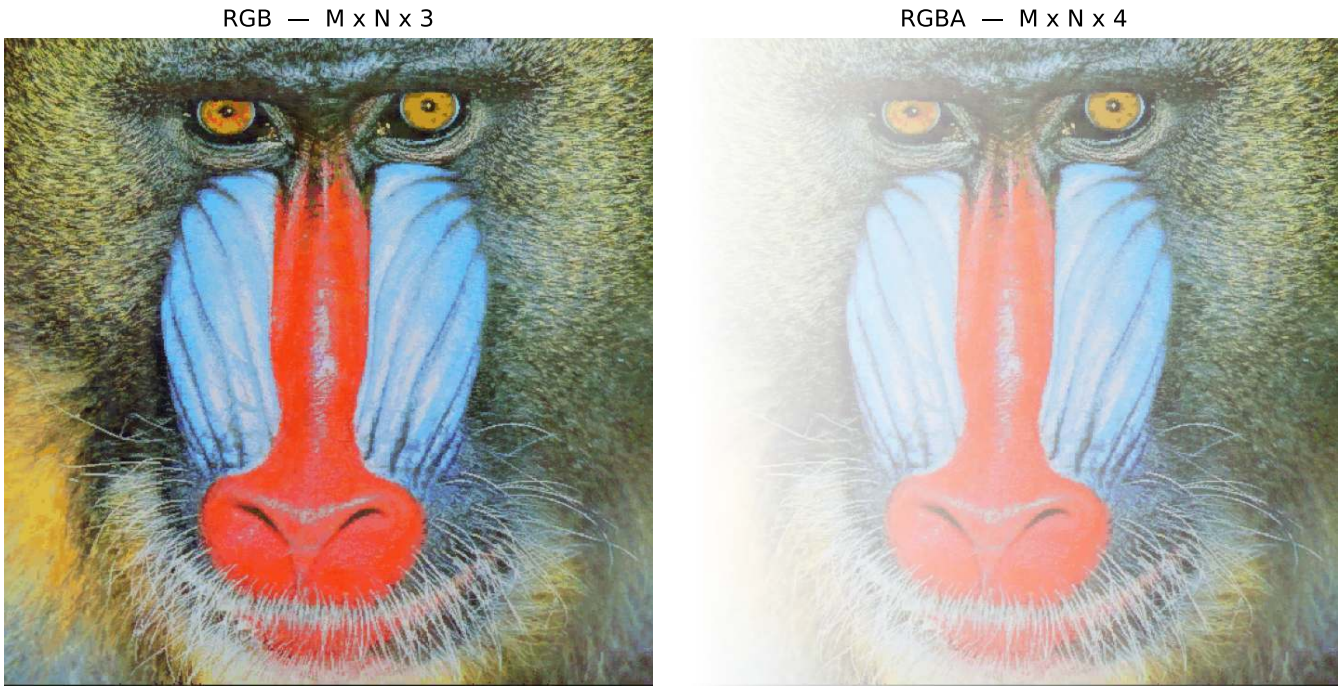


Figura 2.3: Imagem RGB (3 canais, $M \times N \times 3$) e versão RGBA (4 canais, $M \times N \times 4$) com transparência alfa gradual da esquerda para a direita. Imagem Mandrill — USC SIPI Image Database (domínio público).

2.5 Digitalización: Muestreo y Cuantización

Para transformar una escena continua en una imagen digital, son necesarios dos procesos: muestreo y cuantización.

2.5.1 Muestreo - Discretización del espacio

El muestreo consiste en medir el valor de la función $f(x, y)$ en puntos igualmente espaciados, formando una matriz de M filas (altura) y N columnas (ancho). Cada elemento de esta matriz es un **píxel**. La **resolución espacial** está dada por $M \times N$. Cuanto mayor es la resolución, más detalles espaciales se preservan, pero también mayor es el costo computacional y de almacenamiento.

2.5.2 Cuantización - Discretización de la intensidad

La cuantización asocia a cada píxel un valor numérico discreto, generalmente representado por un entero de b bits. La **profundidad de bits** define el número de niveles de intensidad: 2^b. Las imágenes en tonos de gris suelen usar 8 bits (256 niveles). Las imágenes en color usan tres canales de 8 bits (24 bits en total).

Ilustración: Si usamos solo 1 bit por píxel (blanco y negro), perdemos todos los tonos intermedios. Con 2 bits (4 niveles) ya se perciben degradados gruesos. Con 8 bits, el ojo humano difícilmente percibe la discretización (visión continua).

⚠ Error de cuantización

Error de cuantización es la diferencia entre el valor analógico real y el valor discreto asignado. Se manifiesta como ruido de cuantización, visible en regiones con gradiente suave cuando se usan pocos bits.

2.5.3 Efectos del muestreo y la cuantización

Los siguientes experimentos muestran cómo la reducción de la resolución espacial (submuestreo) y de la profundidad de bits degradan la calidad visual. Utilice el código para explorar diferentes factores y niveles de gris.

```
%%writefile tmp/mm_out_1.cpp
// Compile: g++ -std=c++17 -O2 -o programa programa.cpp -lm
#include "morph.hpp"
#include <iostream>
#include <vector>
#include <string>
#include <filesystem>

//| quarto-raw: true

int main() {
    // Imagem de exemplo (barbudo-rajado) - base das figuras de amostragem,
    // quantização e transformações geométricas deste capítulo.
    std::string base    = "https://upload.wikimedia.org/wikipedia/commons";
    std::string arquivo = (
        "Area_de_Prote%C3%A7%C3%A3o_Ambiental_"
        "Quilombos_do_M%C3%A9dio_Ribeira_-_Thomas-"
        "Fuhrmann_%282023-_02%29_Malacoptila_striata.jpg"
    );
    std::string url  = base + "/c/c5/" + arquivo;
    std::string caminho = "imagens/barbudo-rajado.jpg";

    // El patrón de descarga/cache se colapsa a una sola llamada mm::read con la URL.
    mm::Image img_color = mm::read(url);
    mm::Image img_gray0 = mm::gray(img_color);
    mm::Image img_gray  = mm::crop(img_gray0, 820, 1850, 890, 1550);  // recorte p/ ver
detalhes
    std::cout << "Imagem original: [" << img_color.h << ", " << img_color.w << ", " <<
img_color.channels << "]\n";

    // img_color, img_gray0, img_gray, base, arquivo, url, caminho siguen en alcance hasta el
    final de main,
    // pero solo las mm::Image se conservan automáticamente.

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

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

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

```
Imagem original: [3067, 2047, 3]
```

El experimento presentado en la Figura 2.4 ilustra el compromiso entre la **resolución espacial** y el **costo de almacenamiento** en memoria. El código utiliza la técnica de submuestreo por rebanado (*slicing*) para reducir la matriz original de píxeles según un factor f, lo que resulta en un ahorro de memoria — por ejemplo, un factor $f = 8$ reduce el tamaño del dato en 64 veces (8^2). Para fines de comparación visual, las imágenes reducidas se restauran a las dimensiones originales (512×512) mediante la interpolación por **vecino más cercano** (`nearest`). Este proceso no recupera la información perdida, pero hace evidente el

efecto de ***aliasing*** y la estructura de bloques (pixelización) generada por la baja densidad de datos de la matriz muestreada.

```
%%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 <iostream>
#include <vector>
#include <string>

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

    // Cálculo de memória em KB
    double mem_kb = (reduced.h * reduced.w * reduced.channels) / 1024.0;
    std::string label = std::to_string(reduced.w) + "x" + std::to_string(reduced.h) + ", " +
                        std::to_string((int)mem_kb) + " KB\n(Fator " + std::to_string(f) +
")";

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

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

    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).

El experimento en la Figura 2.5 se centra en la **cuantización de intensidad**, el proceso de discretización de la amplitud de la función $f(x, y)$. Mientras que el submuestreo afecta la rejilla espacial, la reducción de la profundidad de bits limita la cantidad de tonos de gris disponibles para representar el brillo.

Al reducir la profundidad de 8 bits (256 niveles) a valores menores, surge el efecto de **posterización**, donde los gradientes suaves de una escena se sustituyen por transiciones abruptas. En el límite de 1 bit, la imagen se vuelve estrictamente binaria, preservando únicamente la silueta y perdiendo detalles de textura y volumen.

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

// Quantiza uma imagem para o número de bits especificado
mm::Image quantize_simple(const mm::Image& image, int bits, std::string& label) {
    int levels = std::pow(2, bits);

    // Normalização e quantização uniforme
    mm::Image quantized(image.h, image.w);
    for (int i = 0; i < image.h * image.w; i++) {
        quantized.data[i] = static_cast<unsigned char>(
            std::floor(image.data[i] / 256.0 * levels) / levels * 255
```

```
        );
    }

    // Cálculo de memória em KB
    double mem_kb = (quantized.h * quantized.w * quantized.channels) / 1024.0;
    label = std::to_string(bits) + " bits (" + std::to_string(levels) + " níveis)\n" +
            std::to_string(static_cast<int>(mem_kb)) + " KB";

    return quantized;
}

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) {
        std::string label;
        mm::Image quantized = quantize_simple(img_gray, b, label);
        imgs_q.push_back(quantized);
        titles_q.push_back(label);
    }

    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 Análisis Técnico

- **Dominio vs. Codominio**: Nótese que la resolución espacial (dimensiones de la matriz) permanece constante en 512x512; lo que se altera es únicamente el codominio de la función de la imagen.
- **Constancia de Memoria**: Obsérvese en los títulos que el tamaño en KB no disminuye. Esto ocurre porque NumPy almacena cada píxel cuantizado en un contenedor de 8 bits (`uint8`), independientemente de que el valor real sea solo 0 o 1.
- **Percepción**: La degradación visual se vuelve crítica por debajo de 4 bits, donde el ojo humano comienza a percibir las "fronteras" artificiales creadas por la falta de tonos intermedios.

La limitación de tipos de datos menores que un byte en el ecosistema Python/NumPy se debe a la arquitectura del hardware, que direcciona la memoria en bloques de 8 bits (Bytes). Para mantener la compatibilidad con OpenCV y garantizar la eficiencia, incluso los elementos binarios se mapean a contenedores de 1 byte (`uint8` o `bool8`).

Aunque lenguajes como ANSI C permiten la compactación de 8 píxeles por byte (*bit-packing*), este enfoque exige la descompactación constante para los cálculos e impone una alta complejidad en la manipulación de punteros. Según la Tabla 2.2, se privilegia el uso de `uint8` por la facilidad en el acceso a vecinos y por la versatilidad en transformaciones geométricas. Además, los métodos nativos de NumPy y OpenCV ejecutan el procesamiento internamente a bajo nivel (C/C++), lo que hace que las operaciones vectorizadas sean más rápidas que las implementaciones manuales con bucles anidados en Python.

Tabla 2.2: Comparativo entre estrategias de compactación y eficiencia de procesamiento.

Característica	Python (NumPy/OpenCV)	ANSI C (Bit-packing)
Unidad más pequeña	1 Byte (8 bits)	1 Bit
Memoria (Binaria)	256 KB (para 512x512)	32 KB (para 512x512)
Velocidad	Alta (Vectorización en C)	Variable (Lenta si hay bit-shift)
Complejidad	Baja: Métodos listos	Alta: Punteros y Máscaras

2.6 Relaciones entre Píxeles - Topología de la Imagen

Los píxeles no son elementos aislados; sus posiciones relativas definen conceptos importantes para el procesamiento.

2.6.1 Vecindad

Dado un píxel de coordenadas (x, y), se definen dos tipos principales de vecindad (para imágenes en *grid* rectangular):

- **Vecindad-4** (von Neumann): incluye los píxeles en las posiciones $(x-1, y)$, $(x+1, y)$, $(x, y-1)$, $(x, y+1)$.
- **Vecindad-8** (Moore): incluye los ocho píxeles adyacentes (añade los cuatro diagonales).

La elección de la vecindad influye en operaciones como detección de bordes, cálculo de gradientes y conectividad.

```
%%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)")
```

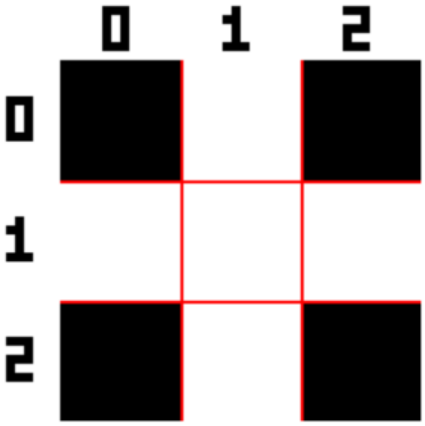


Figura 2.6: Ilustração de vizinhanças 4 em uma matriz 3x3. No centro (1,1), o pixel de interesse.

2.6.2 Adyacencia, conectividad y caminos

Dos píxeles son **adyacentes** si están en contacto según una vecindad definida **y** satisfacen un criterio de valor (ej.: mismo nivel de intensidad). Una **conectividad** define una relación de equivalencia entre píxeles que forman una región conexa. Un **camino** es una secuencia de píxeles adyacentes.

La **conectividad-4** (N4) y la **conectividad-8** (N8) pueden producir resultados diferentes en la segmentación y en el cálculo de componentes conexas (etiquetado). Por ejemplo, un patrón de tablero de ajedrez puede ser completamente disconexo en N4, pero totalmente conexo en N8.

2.6.3 Distancias entre píxeles

Las métricas de distancia son fundamentales para cuantificar la proximidad física y la conectividad entre los elementos que componen la rejilla digital. Como se demuestra en la Tabla 2.3, la elección de la métrica define el coste de desplazamiento entre píxeles y altera el comportamiento de los algoritmos de segmentación y análisis morfológico.

Para medir la distancia entre dos píxeles $p(x_1, y_1)$ y $q(x_2, y_2)$, se utilizan diferentes funciones métricas que imponen restricciones de movimiento distintas sobre la *cuadrícula*:

Tabla 2.3: Comparativa de métricas de distancia aplicadas a la malla de píxeles.

Métrica	Definición	Interpretación
Euclidiana	$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$	Distancia **exacta** en línea recta (continua)
Manhattan (*City block*)	$\lvert x_1 - x_2\rvert + \lvert y_1 - y_2\rvert$	Movimientos horizontales + verticales
Chebyshev (*Tablero*)	$\max(\lvert x_1 - x_2\rvert, \lvert y_1 - y_2\rvert)$	Mayor desplazamiento entre los ejes

Estas distancias se aplican en diversos contextos de PDI, incluyendo algoritmos de interpolación geométrica, transformadas de distancia, crecimiento de regiones y análisis de formas.

i Ejemplo práctico

Considerando dos píxeles con desplazamientos relativos $\Delta x = 3$ y $\Delta y = 4$:

- **Euclidiana**: $\sqrt{3^2 + 4^2} = 5$ (hipotenusa del triángulo rectángulo).
- ***Manhattan***: $3 + 4 = 7$ (suma de los catetos).
- ***Chebyshev***: $\max(3, 4) = 4$ (predominio del mayor desplazamiento).

2.7 Almacenamiento de Imágenes

La elección del formato de archivo es un paso decisivo en el flujo de procesamiento, ya que determina cómo se preservarán o descartarán los datos de muestreo y cuantización. Como se presenta en la Tabla 2.4, cada extensión equilibra de forma distinta la fidelidad de los datos y la eficiencia de almacenamiento.

Tabla 2.4: Principales formatos de almacenamiento de imágenes digitales y sus aplicaciones en PDI.

Formato	Características	Uso típico
PGM	Formato simple de mapa de grises (texto o binario).	Investigación académica y herramientas Unix.
BMP	Sin compresión (o compresión simple).	Windows, aplicaciones heredadas.
PNG	Compresión **sin pérdidas** (*lossless*).	Web, imágenes con transparencia.
JPEG	Compresión **con pérdidas** (*lossy*), ideal para fotografías.	Fotos, cámaras digitales.
TIFF	Admite múltiples capas y compresión variada.	Edición, archivado.
RAW	Datos brutos del sensor, sin procesamiento.	Fotografía profesional.
DICOM	Estándar médico con metadatos clínicos integrados (paciente, equipo, protocolo).	Radiología, tomografía, resonancia magnética.

Los metadatos de una imagen incluyen parámetros como ancho, alto, profundidad de bits y codificación de color. En formatos científicos, también se preservan información de calibración y detalles de la captura. Al utilizar la función `mm::read()`, la biblioteca `morph` preserva automáticamente estos datos para que se respeten las propiedades originales de la imagen.

En contextos científicos y hospitalarios, se privilegia el estándar **DICOM** (*Digital Imaging and Communications in Medicine*) para garantizar que no haya pérdida de precisión diagnóstica. Repositorios públicos como **The Cancer Imaging Archive (TCIA)**, la **Alzheimer's Disease Neuroimaging Initiative (ADNI)** y **PhysioNet** ponen a disposición vastos conjuntos de datos en este formato, incluidos metadatos clínicos anonimizados que son esenciales para la investigación científica.

2.7.1 Ejemplo: Extracción de Metadatos y Localización GPS

A diferencia de la matriz de píxeles pura obtenida mediante la lectura convencional —como en la imagen del pájaro presentada al inicio de este capítulo—, el uso del argumento `pil=True` en el método `mm::read()` altera la naturaleza del objeto devuelto (ver Figura 2.7). Mientras que el estándar (`pil=False`) devuelve un `numpy.ndarray` RGB, la lectura con `pil=True` devuelve un objeto especializado de la biblioteca Pillow, capaz de interpretar el encabezado **EXIF**.

El encabezado **EXIF** (*Exchangeable Image File Format*) funciona como un repositorio técnico de la captura, permitiendo que el objeto Pillow interprete una amplia gama de información que va mucho más allá de las coordenadas GPS. Al utilizar `pil=True`, el sistema obtiene acceso al "ADN" de la imagen, incluyendo metadatos de hardware (marca y modelo de la cámara), configuraciones ópticas (apertura, distancia focal y tiempo de exposición) y parámetros de iluminación (uso de flash y balance de blancos).

Esta distinción es importante en el PDI, ya que transforma la matriz de muestreo en un conjunto de datos contextualizado, donde las características físicas del sensor y del lente pueden utilizarse para normalizar brillos o corregir distorsiones geométricas.

i Ruta C++: por qué la extracción de EXIF permanece en Python

La extracción de metadatos es **análisis de contenedor** — decodificar el encabezado EXIF incrustado en el archivo — y no procesamiento de píxeles: es ortogonal al pipeline de muestreo y cuantización. La `morph.hpp` decodifica imágenes con `stb_image`, que entrega solo la matriz de píxeles y descarta el encabezado EXIF. Como el kernel de este notebook es Python incluso en la ruta C++, este ejemplo utiliza la lectura en modo Pillow (`pil=True`) en ambas rutas. En un programa C++ autónomo, el camino equivalente sería vendorizar una biblioteca dedicada: `easyexif` (lectura de EXIF/GPS en JPEG, header único, cero dependencias) o `exiv2` (lectura **y** escritura, incluyendo 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>
#include <string>
#include <cmath>

int main() {
    // 1. Leitura - preserva objeto PIL com EXIF (nota: C++ não preserva EXIF, apenas lê a
    imagem)
    std::string url = "https://upload.wikimedia.org/wikipedia/commons/c/c5/"
                      "Area_de_Prote%C3%A7%C3%A3o_Ambiental_"
                      "Quilombos_do_M%C3%A9dio_Ribeira_-_Thomas-"
                      "Fuhrmann_%282023-_02%29_Malacoptila_striata.jpg";
    mm::Image img = mm::read(url);

    // Nota: EXIF e GPS não estão disponíveis na API mm::Image
    // 2. Extração e conversão de GPS (Tag 34853) - não aplicável em C++

    // 3. Diagnóstico de tipos, dimensões e acesso a pixels
    std::cout << "\nDimensões (x,y): " << img.w << " x " << img.h << "\n";
    std::cout << "Pixel (0,0): (" << (int)img.at(0, 0, 0) << ", "
              << (int)img.at(0, 0, 1) << ", " << (int)img.at(0, 0, 2) << ")\n";
    std::cout << "Canais: " << img.channels << "\n";

    // 4. Exibição
    mm::show(img, 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"
```

```
Dimensões (x,y): 2047 x 3067
Pixel (0,0): (58, 95, 2)
Canais: 3
```

```
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)")
```

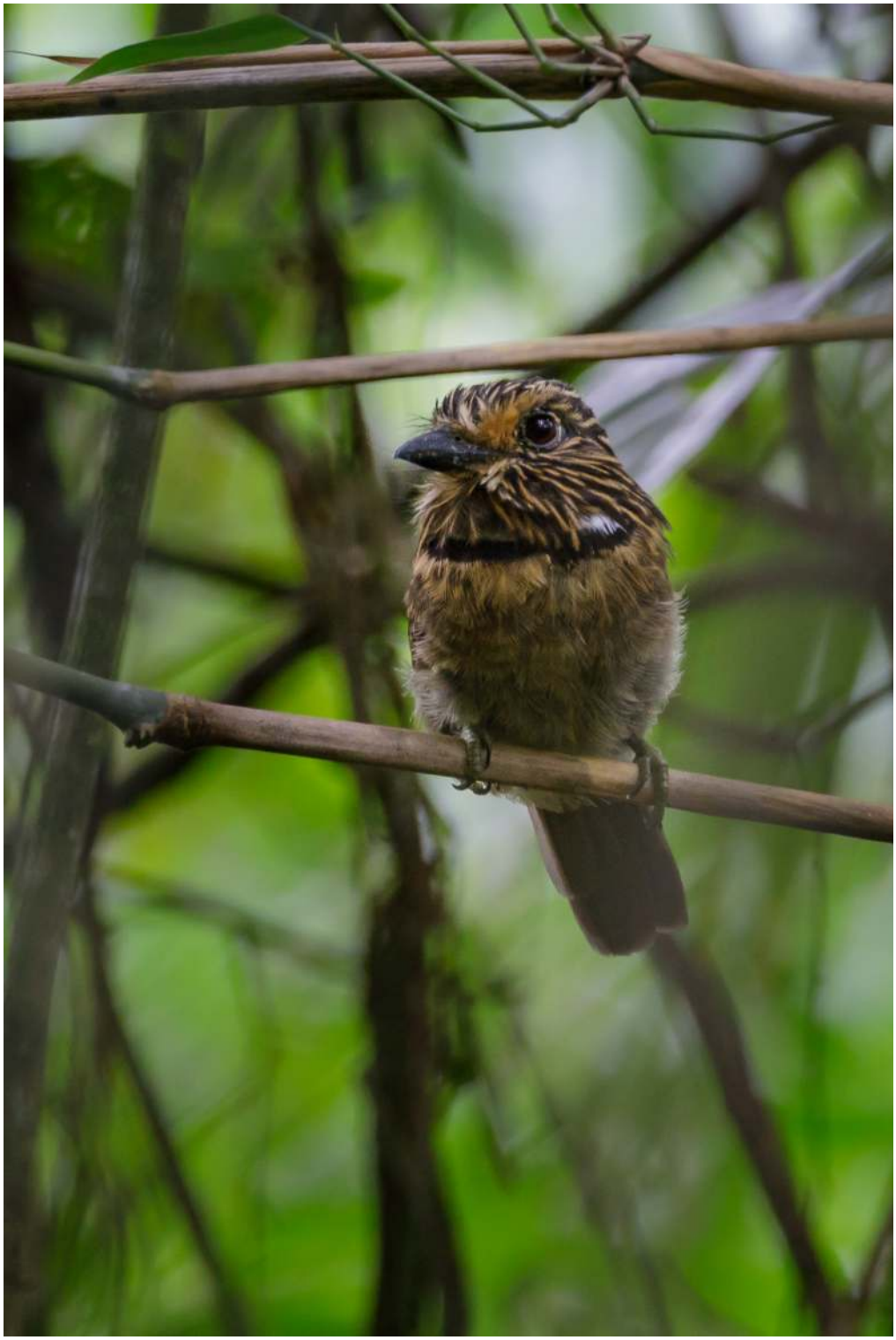


Figura 2.7: Area de Proteção Ambiental Quilombos do Médio Ribeira - Barbudo-rajado (Malacoptila striata). Crédito: Thomas Fuhrmann (CC BY-SA 4.0).

i Nota Pedagógica: La sutil diferencia de las dimensiones

Observe que la representación de las dimensiones cambia según la estructura de datos utilizada:

- **En Pillow (`.size`):** Devuelve `(Ancho, Alto)` — en el ejemplo: `(2047, 3067)`. Es una visión orientada al archivo de imagen.
- **En NumPy (`.shape`):** Sigue la convención matemática de matrices: `(Filas/Alto, Columnas/Ancho, Canales)` — en el ejemplo: `(3067, 2047, 3)`.

Esta distinción es fundamental para evitar errores de indexación al implementar filtros manuales. Mientras el objeto Pillow carga el **"dónde"** y el **"cuándo"** (contexto), el array NumPy carga el **"cuánto"** de luz (intensidad) que existe en cada punto de la imagen.

2.7.2 ¿Por qué es importante esta separación?

Al cargar una imagen por el camino convencional (`pil=False`), el resultado es un `numpy.ndarray`, que contiene estrictamente los valores numéricos resultantes de la **cuantización** y **muestreo**. Sin embargo, al utilizar `pil=True`, el `mm::read()` devuelve un objeto de la clase `PIL.JpegImagePlugin.JpegImageFile`.

Esta clase mantiene el archivo "abierto" para permitir el acceso al contexto de la captura antes de que los datos se conviertan en una matriz bruta. Note que el píxel `(0, 0)` es idéntico en las dos representaciones — `(58, 96, 0)` en Pillow y `[58, 96, 0]` en NumPy —, confirmando que ambos describen los mismos datos, solo con interfaces distintas. Esta separación es vital: los píxeles sirven para algoritmos; los metadatos sirven para georreferenciación, catalogación científica y correcciones basadas en el hardware de adquisición.

Para inspeccionar todos los metadatos EXIF de un objeto 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 Transformaciones Geométricas Básicas

Las transformaciones geométricas alteran la posición de los píxeles, manteniendo los valores de intensidad. Son fundamentales para la alineación, la corrección de distorsiones y el aumento de datos (*data augmentation*) en el aprendizaje automático.

Una **transformación afín** es cualquier mapeo que preserve la colinealidad (puntos sobre una recta permanecen sobre una recta) y las razones de distancias entre puntos colineales. En 2D, toda transformación afín puede expresarse en **coordenadas homogéneas** mediante una matriz 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 submatriz 2×2 superior izquierda $\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$ controla la rotación, la escala y el cizallamiento; el vector $(t_x, t_y)^\top$ controla la traslación. Las transformaciones geométricas más utilizadas en PDI —traslación, rotación y escala— son casos particulares de $\mathbf{T}$, y pueden **componerse** mediante la multiplicación de matrices, en el orden $\mathbf{T} = \mathbf{T}_n \cdots \mathbf{T}_2 \mathbf{T}_1$.

> **i** Transformación inversa e interpolación
>
> En la implementación práctica (`cv2.warpAffine`), se aplica la **transformación inversa**: para cada píxel (x', y') de la imagen destino, se calcula la posición de origen $(x, y) = \mathbf{T}^{-1}(x', y')$ y se interpola el valor. Esto evita huecos en la imagen resultante causados por el mapeo directo de píxeles enteros a posiciones no enteras.

2.8.1 Traslación

La traslación es la transformación afín más simple: desplaza todos los píxeles mediante un vector (t_x, t_y). En coordenadas homogéneas, se expresa mediante la matriz:

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

La tercera fila de la matriz garantiza que la operación permanezca en el espacio afín, permitiendo que la traslación, la rotación y la escala se combinen mediante una simple multiplicación de matrices. En la práctica, `cv2.warpAffine` utiliza solo las dos primeras filas (matriz 2×3), ya que la tercera es siempre $[0, 0, 1]$.

Los píxeles desplazados más allá del área original se descartan; las áreas descubiertas se rellenan con 0 (negro). Ver un ejemplo en la 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 <string>
#include <filesystem>

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

    //| 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 Rotación

La rotación por un ángulo θ alrededor de un punto central (c_x, c_y) se compone de tres transformaciones afines: traslación al origen, rotación pura y traslación de regreso. La matriz resultante es:

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

En la implementación, `cv2.getRotationMatrix2D` genera directamente las dos primeras filas de $\mathbf{T}_{\text{rot}}$ (matriz 2×3 para `warpAffine`), aceptando también un factor de escala s que multiplica $\cos\theta$ y $\sin\theta$. Ver ejemplo en la Figura 2.9.

Dado que la rotación desplaza píxeles a nuevas posiciones no enteras, `cv2.warpAffine` necesita estimar el color de cada píxel de destino a partir de los vecinos — proceso llamado **interpolación**. El parámetro `interp` controla este comportamiento:

- **`nearest`** (`INTER_NEAREST`): asigna el valor del píxel más cercano. Rápido, pero produce bordes dentados (*aliasing*) en bordes diagonales.
- **`bilinear`** (`INTER_LINEAR`, predeterminado): promedio ponderado de los 4 vecinos más cercanos. Equilibra calidad y rendimiento — adecuado para la mayoría de los casos.
- **`bicubic`** (`INTER_CUBIC`): considera los 16 vecinos en una superficie cúbica. Produce bordes más suaves, al costo de mayor procesamiento.

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

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

    mm::Image img_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 Escala (Redimensionamiento)

El redimensionamiento por factores (s_x, s_y) es una transformación afín con matriz:

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

Cuando $s > 1$ (ampliación), los píxeles de la imagen destino se mapean a posiciones no enteras en el origen — lo que exige interpolación para estimar el valor. Cuando $s < 1$ (reducción), múltiples píxeles de origen contribuyen a un único píxel destino — lo que exige **decimación**. Los mismos tres métodos descritos en la rotación están disponibles en `mm::resize`, con la diferencia de que aquí el impacto visual es más perceptible: en la ampliación, `nearest` produce efecto de bloques (*pixelation*), mientras que `bicubic` preserva mejor la nitidez de los bordes, conforme a la Tabla 2.5:

Tabla 2.5: Métodos de interpolación disponibles en `mm::resize` y sus respectivos números de vecinos utilizados en el cálculo.

Método	Vecinos utilizados	Característica
`'nearest'`	1	Rápido; produce efecto de bloques (*pixelation*)
`'bilinear'`	4	Buen compromiso calidad/costo; bordes suaves
`'bicubic'`	16	Mayor nitidez; preferido en software profesional

El parámetro `size_or_factor` acepta tanto un escalar (factor uniforme, p. ej., `0.5` para reducir a la mitad) como una tupla `(ancho, alto)` para dimensiones absolutas.

```
%%writefile tmp/fig_02_escala_detalhe.cpp
#define MM_OUT "tmp/fig_02_escala_detalhe.png"
// Compile with: g++ -std=c++17 -O2 -I. -o programa programa.cpp -lm && ./programa
#include "morph.hpp"
#include <iostream>
#include <vector>
#include <string>
#include <filesystem>

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

    // Variables img_gray are already initialized

    // 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 Cisallamiento (*Shear*)

El cisallamiento es una transformación afín que distorsiona la imagen al desplazar cada píxel proporcionalmente a su posición en un eje. La matriz general combina el cisallamiento horizontal (sh_x) y vertical (sh_y):

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

Para $sh_x \neq 0$ y $sh_y = 0$, cada fila se desplaza horizontalmente de manera proporcional a su posición vertical, produciendo el efecto de “inclinación” característico. Ver ejemplo en la 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 <vector>
#include <string>
#include <filesystem>

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

    mm::Image img_shx = mm::shear(img_gray, 0.3, 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)")
```

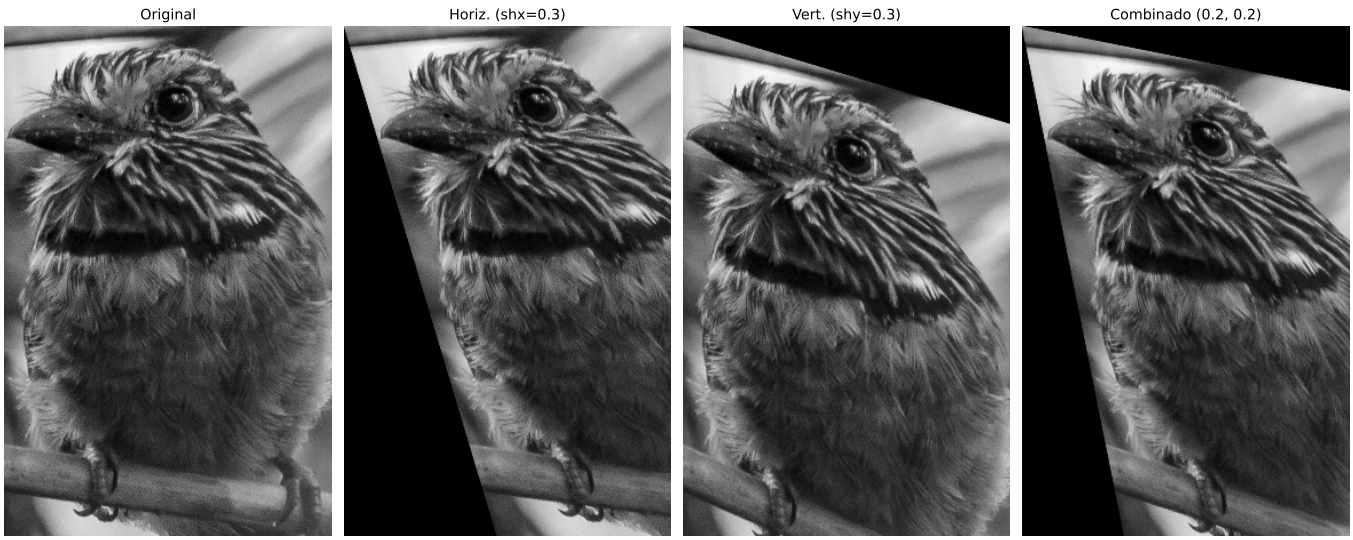


Figura 2.11: Exemplo de cisalhamento da imagem do pássaro: horizontal (shx=0.3), vertical (shy=0.3) e combinado (shx=0.2, shy=0.2).

2.9 Resumen

En este capítulo se presentaron los fundamentos de digitalización y topología de imágenes:

- **Formación de la imagen:** $f(x, y) = i(x, y) \cdot r(x, y)$.
- **Muestreo:** discretización del espacio → resolución espacial $M \times N$.
- **Cuantización:** discretización de la intensidad → profundidad de bits b.
- **Relaciones topológicas:** vecindad-4, vecindad-8, conectividad, distancias (Euclidiana, Manhattan, Chebyshev).
- **Transformaciones geométricas:** traslación, rotación, escala (con interpolación bilineal o vecino más próximo).
- **Formatos de archivo:** BMP, PNG, JPEG, TIFF, RAW; cada uno con diferentes compromisos entre calidad y tamaño.

El Capítulo 3 abordará **operaciones espaciales** como convolución, filtrado y morfología matemática (erosión, dilatación).

2.10 🤖 Uso del Gemini Notebook como Tutor Complementario

En esta edición, incentivamos el uso del **Gemini Notebook** como herramienta complementaria de aprendizaje. Esta herramienta de IA utiliza exclusivamente los documentos proporcionados por el autor como base de conocimiento, garantizando respuestas coherentes con el contenido del libro.

Para cada capítulo, hemos preparado un proyecto específico en la plataforma. Para una experiencia de estudio ampliada, utilice el siguiente acceso:

🎓 Estudie con el Tutor Inteligente

Para interactuar con el contenido de este capítulo, acceda al siguiente enlace. El entorno contiene materiales didácticos en diferentes formatos, generados a partir del **PDF** del capítulo. En la plataforma, explore especialmente las opciones **Guía de Estudio** y **Conversación** para profundizar su comprensión.
🚀 ACCEDER A GEMINI NOTEBOOK: CAPÍTULO 02

🌐 Idioma y Lenguaje de Programación

El proyecto de este capítulo en el Gemini Notebook fue construido únicamente con el texto en **portugués** y los ejemplos de código en **Python**. Si está estudiando la edición en inglés o francés, o siguiendo la ruta en C++, las respuestas del tutor pueden no corresponder exactamente a la versión que está leyendo.

⚠️ Aviso sobre Contenido Generado por IA

La IA es una poderosa aliada en los estudios, pero el contenido generado puede contener **errores o imprecisiones**. Consulte siempre **libros, artículos científicos y otras fuentes académicas confiables** para validar la información. Siempre que sea posible, ejecute los ejemplos prácticos proporcionados en este capítulo para verificar los resultados.

2.11 Lista de Ejercicios

1. **(15%)** Explique, con sus propias palabras, la diferencia entre **muestreo** y **cuantización**. Dé un ejemplo concreto de cada uno en el contexto de una imagen digital.
2. **(15%)** Considere una imagen con resolución espacial de 1024×768 píxeles y profundidad de 24 bits (8 bits por canal RGB). Calcule el tamaño total no comprimido de la imagen en bytes y en megabytes.
3. **(20%)** Utilizando el código del laboratorio, modifique el factor de submuestreo a 3 y a 6. Describa visualmente qué ocurre con los bordes de los objetos. ¿Qué es el efecto de *aliasing*?
4. **(20%)** Para la imagen en escala de grises, aplique cuantización con 3 bits (8 niveles) y 5 bits (32 niveles). Compare los resultados y explique por qué 5 bits ya puede considerarse suficiente para muchas aplicaciones.
5. **(15%)** Dados dos píxeles $A = (10, 20)$ y $B = (15, 25)$, calcule las distancias Euclidiana, Manhattan y Chebyshev entre ellos.
6. **(15%)** Usando la función `mm::rotate`, gire la imagen del pájaro en ángulos de 90°, 180° y 270° con interpolación bilineal. Compare con la rotación usando `method='nearest'`. ¿En qué situaciones la interpolación de vecino más cercano sigue siendo útil?

Referencias del Capítulo

La fundamentación teórica de este capítulo se basa en las siguientes obras:

- Gonzalez (2018) para los conceptos de muestreo, cuantización y relaciones entre píxeles.
- Szeliski (2022) para transformaciones geométricas y conectividad.
- Bradski (2008) para la implementación práctica con OpenCV y `morph.py`.

Executar Colab Abrir GitHub

2.12 💻 Parte Práctica con Ejercicios de Programación

🎯 Objetivo de este Cuaderno

El cuaderno permite desarrollar, validar, organizar y probar soluciones de **Ejercicios de Programación (EPs)** en entornos interactivos, como Colab, con los mismos casos de prueba de Moodle, copiando allí únicamente al momento de registrar la nota oficial.

Download

Descargue `morph.py` y `testsuite.py` ejecutando la celda a continuación:

```
import os, urllib.request

os.makedirs("tmp/state", exist_ok=True)  # artefactos de build del camino C++ (.cpp, binario,
PNGs)

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

# El kernel es Python incluso en el camino C++: `mm` (morph.py) es usado por los
# simuladores, por la visualización de las figuras que el binario C++ genera y por el
# estado mm::Image entre celdas. cpp=True baja también el camino compilado
# (morph.hpp + stb_image*.h), usado en el #include de las celdas %%writefile *.cpp.
import config
config.setup(testsuite=True, cpp=True)
from morph import mm
from testsuite import TestSuite
```

```
Entorno listo. Morph: 1.1.9 | OpenCV: 5.0.0 | TestSuite: 1.1.2
```

Ejecutando las pruebas

Para evaluar las pruebas, ejecute `TestSuite("EP04_01.extensión").run()` en una nueva celda, reemplazando la extensión por la del lenguaje utilizado (`.py`, `.java`, `.c`, `.cpp`, `.js` o `.r`). El sistema descarga los casos de prueba de GitHub, ejecuta el programa y calcula la nota automáticamente.

Para probar código Python directamente, sin guardar archivo, use `run_code(codigo)` pasando el código como *string* en una variable `codigo`:

```
codigo = """
from morph import mm
# ... su código aquí ...
"""
TestSuite("EP04_01").run_code(codigo)
```

2.12.1 EP02_01 ⊙ Ajuste de Brillo y Contraste

En esta actividad, el objetivo es implementar un operador de punto para **transformación lineal de intensidad**, aplicando el ajuste dinámico de brillo y contraste en una imagen digital.

2.12.1.1 📋 Directrices de Implementación

El algoritmo debe seguir el flujo de ejecución siguiente:

1. **Dimensiones:** Leer los enteros L (filas) y C (columnas) de la matriz.
2. **Parámetros:** Leer el valor real α (factor de contraste) y el entero β (factor de brillo).
3. **Datos:** Leer los valores enteros de la matriz original.
4. **Mapeo:** Para cada píxel p, calcular el nuevo valor p' mediante la ecuación:

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

5. **Salida:** Mostrar la matriz resultante con las dimensiones $L \times C$.

2.12.1.2 📌 Restricciones Computacionales

- **Redondeo (*Round*):** Se aplica el redondeo matemático al entero más próximo antes de la conversión de tipo.
- **Saturación (*Clipping*):** Los valores deben limitarse al intervalo $[0, 255]$ para preservar el estándar de 8 bits:

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

- **Simulación:** El efecto de los parámetros α y β en la corrección histogramática puede observarse en la Figura 2.12.

2.12.1.3 🧠 Fundamentación Teórica

Las alteraciones modifican el histograma de la imagen para ajustar el perfil de iluminación y la distinción tonal.

Parámetro	Función	Impacto Visual
α (***Alpha***)	Escalar	Modula el **Contraste**. Si $\alpha > 1$, expande el histograma; si $0 \leq \alpha < 1$, lo comprime.
β (***Beta***)	Aditiva	Modula el **Brillo**. Si es positivo, desplaza el histograma hacia la derecha; si es negativo, hacia la izquierda.
clip	Limitador	Restringe el rango dinámico, evitando errores de *underflow* y *overflow*.

2.12.1.4 📦 Especificación de Entrada y Salida (VPL)

Entrada:

- Línea 1: Entero L.
- Línea 2: Entero C.
- Línea 3: Valores de `alpha` (α) y `beta` (β).
- Líneas siguientes: Elementos numéricos de la matriz original.

Salida:

- Matriz transformada estructurada en L filas y C columnas.

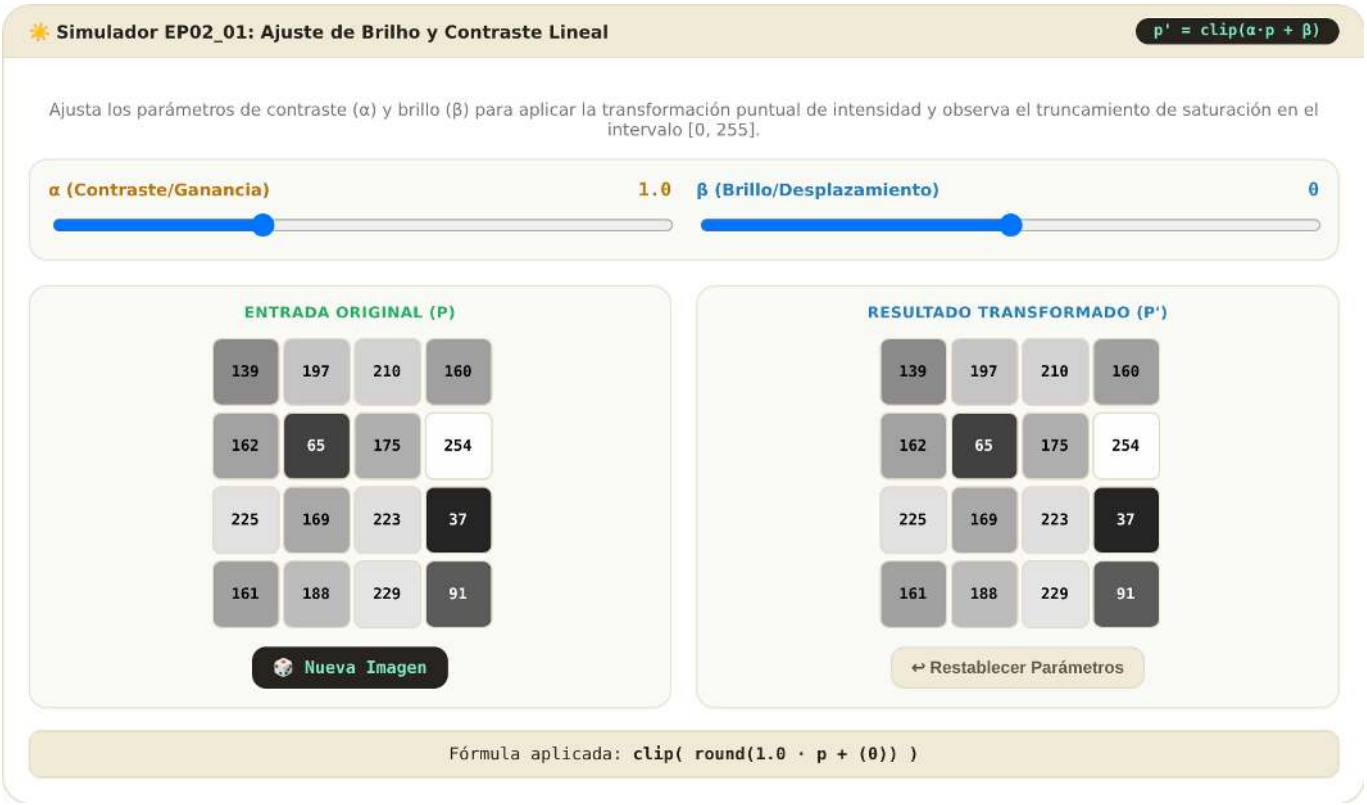
2.12.1.5 📌 Ejemplos

La tabla siguiente presenta un ejemplo práctico del comportamiento esperado del algoritmo, destacando la actuación de los operadores de redondeo y saturación.

Entrada	Salida	Observación
1 4 1.5 -30 0 100 180 255	0 120 240 255	Nótese el efecto de saturación en el último píxel

Ejecutando los Pruebas

Para evaluar las pruebas, ejecute `TestSuite("EP02_01.extensión").run()` en una nueva celda, reemplazando la extensión por la del lenguaje utilizado (`.py`, `.java`, `.c`, `.cpp`, `.js` o `.r`). El sistema descarga los casos de prueba desde GitHub, ejecuta el programa y calcula la nota automáticamente.



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

Figura 2.12: Simulador EP02_01: Ajuste de Brilho y Contraste Lineal (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 🔬 Submuestreo Espacial

En esta actividad, debes implementar la reducción de la resolución espacial de una imagen mediante el proceso de submuestreo.

- Lee dos enteros **L** y **C**, que representan las dimensiones de la matriz original.
- Lee un valor entero f ($f \geq 1$), que representa el factor de muestreo.
- Lee los valores enteros de la matriz original.
- La nueva imagen debe construirse seleccionando el píxel de la posición $(f \cdot i, f \cdot j)$ de la imagen original.
- Imprime la matriz resultante con las nuevas dimensiones.
- Ver en la Figura 2.13 una simulación de este EP.

📌 **Importante**:

- **Dimensiones Finales:** La imagen muestreada tendrá dimensiones $\lceil L/f \rceil \times \lceil C/f \rceil$. En el contexto de programación, esto equivale al tamaño resultante de un segmentado (slicing) con paso f.
- **Implementación:** No utilices funciones predefinidas de bibliotecas de procesamiento de imágenes (como OpenCV o PIL) para el redimensionamiento. Implementa la lógica de selección de píxeles manualmente o mediante segmentado de matrices.
- **Aliasing:** Ten en cuenta que este proceso puede causar el efecto de *aliasing* (dientes de sierra), donde se pierden detalles finos o aparecen patrones no deseados.

2.12.2.1 🧠 Discretización del Espacio

El submuestreo reduce la resolución espacial de una imagen, seleccionando solo un píxel de cada f píxeles en cada dirección. Es el proceso inverso de la interpolación:

Parámetro	Función	Efecto
Factor f	Salto de muestreo	Define el intervalo de selección. Un factor 2 reduce el ancho y la altura a la mitad.
Resolución	Densidad de píxeles	Disminuye la cantidad total de información espacial de la imagen.
Aliasing	Efecto secundario	Aparición de patrones en escalera o bloques debido a la pérdida de detalles finos.

2.12.2.2 📋 Tarea (especificación para VPL)

Entrada:

La primera línea contiene `L`.

La segunda línea contiene `C`.

La tercera línea contiene el factor `f`.

Las líneas siguientes contienen los elementos de la matriz $L \times C$.

Salida:

La matriz reducida con las dimensiones correspondientes al segmentado por `f`.

2.12.2.3 📌 Ejemplos

Entrada	Salida	Observación
2 4 2 10 20 30 40 50 60 70 80	10 30	El factor 2 selecciona los píxeles (0,0) y (0,2) de la primera fila. La segunda fila se ignora.

```
%%writefile EP02_02.cpp
// your solution
```

```
Overwriting EP02_02.cpp
```

```
TestSuite("EP02_02.cpp").run()
```

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

Figura 2.13: Simulador EP02_02: Submuestreo Espacial (Reducción de Resolución por Salto f)

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

2.12.3 EP02_03 🎨 Cuantización de Niveles de Gris

En esta actividad, debes implementar la cuantización uniforme de una imagen, reduciendo la cantidad de niveles de intensidad de gris originales a una nueva escala basada en un número menor de bits.

- Lee dos enteros **L** y **C**, que representan las dimensiones de la matriz.
- Lee un entero k ($1 \leq k \leq 8$), que representa el nuevo número de bits de la imagen.
- Calcula el número de niveles ($N = 2^k$) y el tamaño del intervalo (paso).
- Para cada píxel p, calcula el nuevo valor p' mapeándolo al índice del nivel discretizado correspondiente (que varía de 0 a $2^k - 1$).
- Imprime la matriz resultante con los mismos valores de dimensiones originales.
- Consulta en Figura 2.14 una simulación de este EP.

📌 **Importante**:

- **Posterización:** Al reducir drásticamente los niveles (ej: $k = 2$), notarás que los degradados suaves se transforman en bandas abruptas de color debido a la pérdida de resolución de amplitud.
- **Cálculo del Paso:** El intervalo entre cada nivel se define como $paso = 256/2^k$.
- **Mapeo:** El método de cuantización uniforme por truncamiento que mapea el píxel al índice de su respectivo nivel discretizado está dado por:

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

En términos de implementación (como en Python), esto equivale a la división entera: `p' = p // paso`.

2.12.3.1 🧠 Discretización de la Amplitud

Mientras que el submuestreo trata con la resolución espacial, la cuantización se centra en la precisión del color (amplitud). Reducir los bits significa simplificar la información cromática:

Parámetro	Función	Efecto
Bits (k)	Profundidad de color	Define cuántos tonos diferentes puede tener la imagen (2^k).
Paso	Intervalo de tono	Espaciado entre los niveles de gris permitidos.
Posterización	Fenómeno visual	Transformación de variaciones continuas en bloques de color sólido.

2.12.3.2 📋 Tarea (especificación para VPL)

Entrada:

La primera línea contiene `L`.

La segunda línea contiene `C`.

La tercera línea contiene el número de bits `k`.

Las líneas siguientes contienen los elementos de la matriz $L \times C$.

Salida:

La matriz transformada con los índices de los niveles cuantizados, manteniendo el tamaño original $L \times C$.

2.12.3.3 📌 Ejemplos

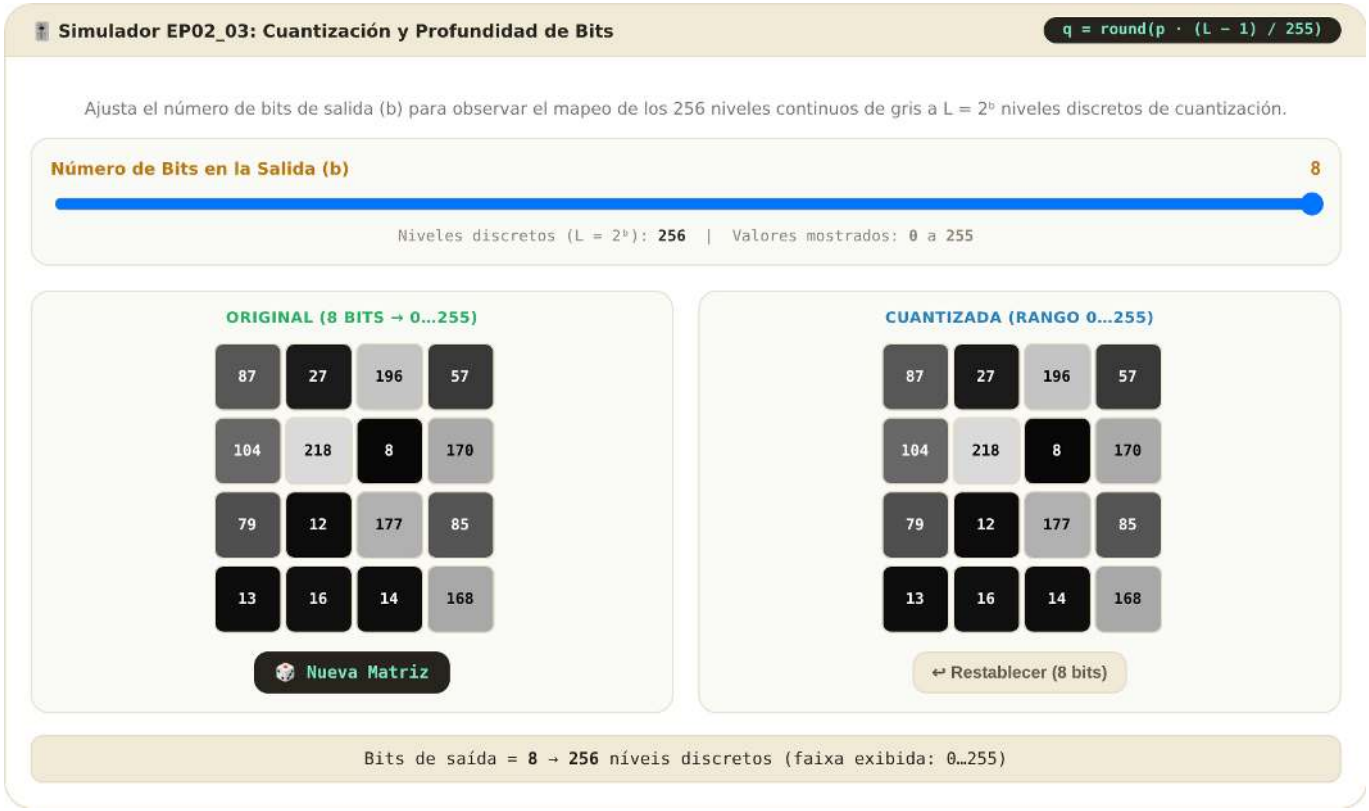
Entrada	Salida	Observación
1 4 2 0 80 170 255	0 1 2 3	Con $k = 2$, tenemos $2^2 = 4$ niveles discretos disponibles $(0, 1, 2, 3)$. El paso es $256/4 = 64$. Aplicando la división entera por elemento: $0//64 = 0$, $80//64 = 1$, $170//64 = 2$, $255//64 = 3$.
1 5 1 10 50 120 200 250	0 0 0 1 1	Con $k = 1$, tenemos $2^1 = 2$ niveles (0 y 1). Paso $= 256/2 = 128$. Los píxeles menores que 128 resultan en 0, y los píxeles mayores o iguales a 128 resultan en 1.

```
%%writefile EP02_03.cpp
// your solution
```

```
Overwriting EP02_03.cpp
```

```
TestSuite("EP02_03.cpp").run()
```

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



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

Figura 2.14: Simulador EP02_03: Cuantización y Profundidad de Bits (Reducción del Número de Niveles de Gris)

2.12.4 EP02_04 📐 Transformada de Distancia en Imagen Binaria

Dada una imagen binaria donde los píxeles de valor **1** representan el *objeto* y los píxeles **0** representan el *fondo*, la distancia de un píxel de fondo recibe la **menor distancia al píxel de objeto más cercano**. Los píxeles de objeto reciben distancia **0**. Para simplificar, considere que la imagen tiene **solo un único objeto con un píxel de valor 1**.

Problema: Lea una imagen binaria $L \times C$ y una métrica, y calcule esa distancia simplificada aplicando una de las tres fórmulas:

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

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

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

donde Δr es la diferencia de filas y Δc la diferencia de columnas entre dos píxeles.

2.12.4.1 🖼️ ¿Por qué es importante? - Aplicaciones de la DT

La Transformada de Distancia (DT) aparece en decenas de pipelines de visión por computadora:

Métrica	Complejidad	Aplicación típica
Euclidiana	🔴 $O(n^2)$ ingenuo	Esqueletización, emparejamiento de formas
City-block	🟡 $O(n)$ con 2 pasadas	Morfología, dilatación/erosión

Métrica	Complejidad	Aplicación típica
Chessboard	🟢 $O(n)$ con 2 pasadas	Morfología, dilatación/erosión

2.12.4.2 📌 Requisitos Técnicos

- **Entrada:** * Primera línea: L y C (enteros).
  - Segunda línea: nombre de la métrica (`euclidean`, `cityblock` o `chessboard`).
  - A continuación, la matriz binaria $L \times C$ (valores 0 o 1).
- **Píxeles de objeto (1):** distancia = 0 (o 0.00 para euclidiana).
- **Píxeles de fondo (0):** distancia al único píxel de objeto en la imagen.
- **Redondeo (euclidiana):** imprimir con **2 decimales** (formato `:.2f`). City-block y Chessboard producen enteros — imprimir sin decimales.
- **Salida:** valores separados por espacio, una línea por fila de la matriz.
- Ver en Figura 2.15 una simulación de este EP.

2.12.4.3 📌 Ejemplos

Entrada	Salida	Observación
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 distancia Chessboard es $\max(\|dx\|, \|dy\|)$. El único píxel objeto es $(2, 2) = 0$; los demás almacenan su distancia mínima hasta él.

2.12.4.4 📌 Observaciones finales

- Como la imagen tiene **solo un objeto de un píxel**, la distancia de cada píxel de fondo es simplemente la distancia de ese píxel al único punto objeto.
- La implementación puede usar fuerza bruta (recorrer todos los píxeles de la imagen y calcular la distancia directamente), ya que L y C son pequeños en los casos de prueba.
- Este problema es un calentamiento para la Transformada de Distancia general, que se trabajará en capítulos posteriores con múltiples objetos y algoritmos optimizados.

```
%%writefile EP02_04.cpp
// your solution
```

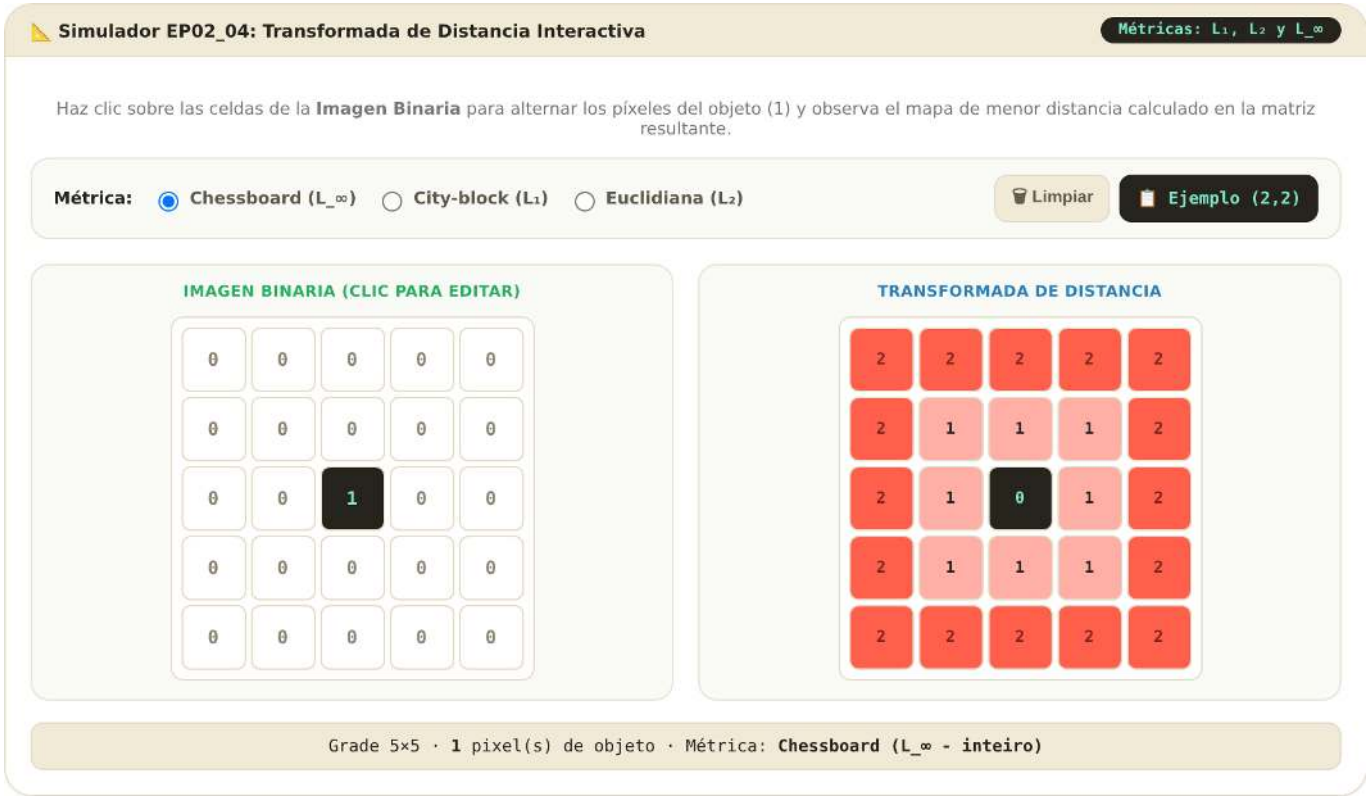
```
Overwriting EP02_04.cpp
```

```
TestSuite("EP02_04.cpp").run()
```

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

2.12.5 EP02_05 Transposición de Imagen

En esta actividad, debes implementar el desplazamiento espacial de una imagen. La traslación mueve cada píxel de la imagen original a una nueva posición basándose en un vector de desplazamiento.



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

Figura 2.15: Simulador EP02_04: Transformada de Distância em Imagem Binária (Chessboard, City-block y Euclidiana)

- Lee dos enteros **L** y **C**, que representan las dimensiones de la matriz.
- Lee dos enteros t_x (desplazamiento horizontal) y t_y (desplazamiento vertical).
- Lee los valores enteros de la matriz original.
- Calcula la nueva posición (x', y') para cada píxel (x, y) original.
- Imprime la matriz resultante con las mismas dimensiones que la original.
- Consulta en Figura 2.16 una simulación de este EP.

📌 **Importante**:

- **Relleno:** Los píxeles que "entran" en la imagen debido al desplazamiento y no tienen un correspondiente en la original deben rellenarse con **0** (negro).
- **Descarte:** Los píxeles que, después de la traslación, queden fuera de los límites de la matriz ($0 \ldots L-1$ o $0 \ldots C-1$) deben ignorarse.
- **Coordenadas:** Considera x como el índice de la fila e y como el índice de la columna.

2.12.5.1 🧠 Desplazamiento Espacial

Trasladar una imagen significa mover todos sus puntos una distancia fija en direcciones especificadas. Matemáticamente, usando coordenadas homogéneas, la operación se describe como:

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

Lo que resulta en las ecuaciones simples:

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

2.12.5.2 📋 Tarea (especificación para VPL)

Entrada:

La primera línea contiene `L`.

La segunda línea contiene `C`.

La tercera línea contiene los enteros `tx` y `ty`.

Las líneas siguientes contienen los elementos de la matriz $L \times C$.

Salida:

La matriz resultante con las mismas dimensiones $L \times C$ después del desplazamiento.

2.12.5.3 📌 Ejemplos

Entrada	Salida	Observación
2 2 1 1 10 20 30 40	0 0 0 10	**Desplazamiento ($t_x = 1, t_y = 1$):** Cada píxel se mueve una posición hacia la derecha (horizontal) y una hacia abajo (vertical). El píxel $(0, 0) = 10$ va al destino $(1, 1)$ (esquina inferior derecha). Las posiciones vacías se rellenan con 0.
3 3 -1 0 1 2 3 4 5 6 7 8 9	2 3 0 5 6 0 8 9 0	**Desplazamiento ($t_x = -1, t_y = 0$):** Cada píxel se mueve una posición hacia la izquierda (horizontal). La primera columna original (`1`, `4`, `7`) se descarta, las demás columnas se mueven hacia la izquierda y la última columna resultante se rellena con ceros (0).

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

```
Overwriting EP02_05.cpp
```

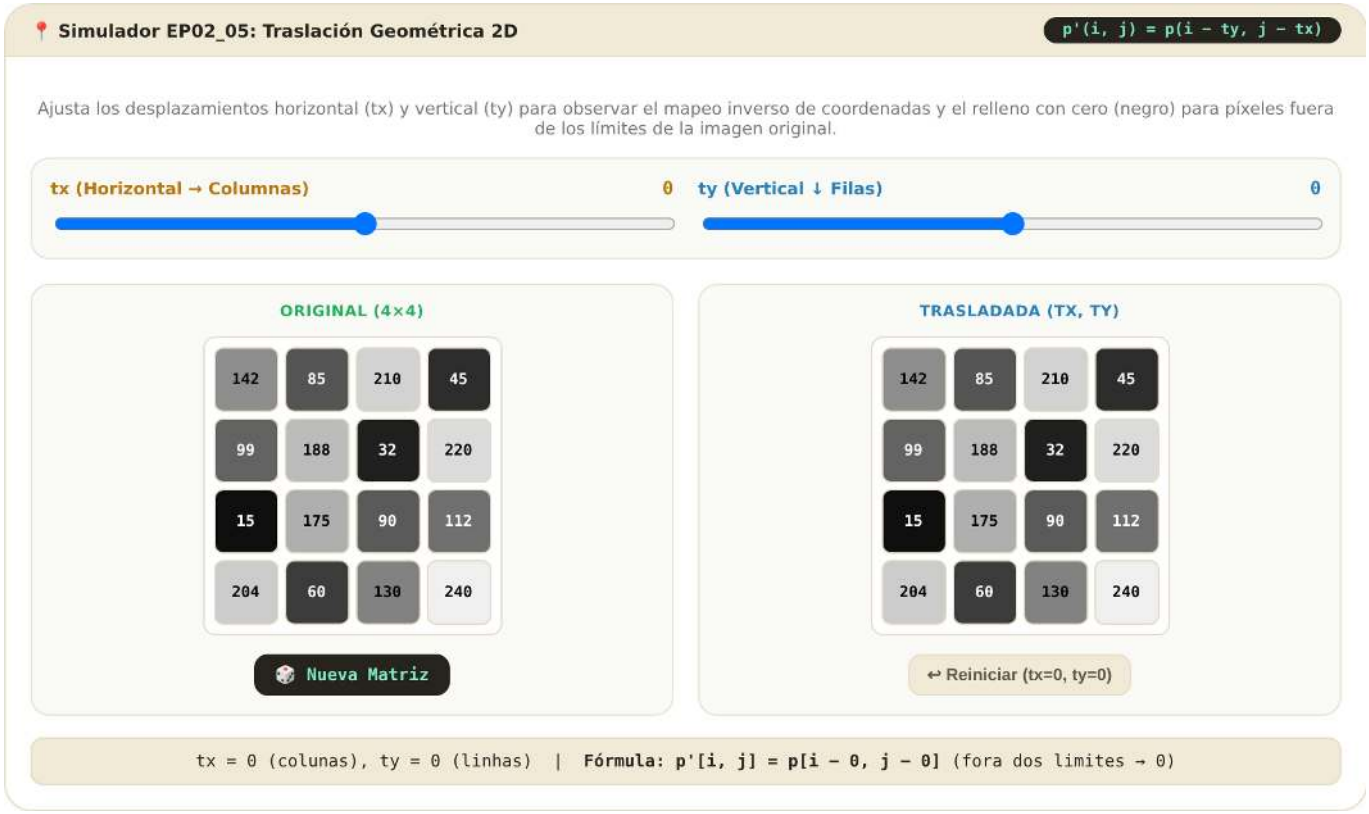
```
TestSuite("EP02_05.cpp").run()
```

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

2.12.6 EP02_06 🔄 Rotación de Imagen

En esta actividad, debes implementar la rotación de una imagen alrededor de su centro geométrico. Esta operación requiere el mapeo de coordenadas y el uso de técnicas de interpolación para determinar los nuevos valores de los píxeles.

- Lea dos enteros **L** y **C**, que representan las dimensiones de la matriz.
- Lea un valor real θ (ángulo en grados) y una *cadena* que representa el **método** de interpolación (`nearest` o `bilinear`).
- Lea los valores enteros de la matriz original.



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

Figura 2.16: Simulador EP02_05: Traslación Geométrica de Imagen (Desplazamiento tx y ty con Relleno de Borde)

- Realice la rotación alrededor del centro de la imagen $(L/2, C/2)$.
- Imprima la matriz resultante con las mismas dimensiones que la original.
- Ver en Figura 2.17 una simulación de este EP.

📌 **Importante**:

- **Mapeo Inverso:** Para evitar "huecos" en la imagen final, recorra cada píxel (x', y') de la imagen de destino y calcule su posición correspondiente (x, y) en la imagen original usando la matriz de rotación inversa.
- **Interpolación:**
- `nearest`: Asigna el valor del píxel más cercano a la coordenada calculada.
- `bilinear`: Calcula un promedio ponderado basado en los 4 vecinos más cercanos.
- **Bordes:** Los píxeles cuyo origen (x, y) caiga fuera de los límites de la imagen original deben rellenarse con **0**.

2.12.6.1 🧠 Transformación por Ángulo

La rotación de un punto (x, y) con respecto al origen por un ángulo θ se da mediante la matriz de transformación. Para rotar alrededor de un centro (x_c, y_c), primero trasladamos el centro al origen, rotamos y trasladamos de vuelta:

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

Consejo: Use el mapeo inverso para garantizar que todos los píxeles de la imagen de salida se rellenen correctamente.

2.12.6.2 📋 Tarea (especificación para VPL)

Entrada:

La primera línea contiene `L`.

La segunda línea contiene `C`.

La tercera línea contiene el ángulo `theta` (en grados) y el método `interp` (`nearest` o `bilinear`).

Las líneas siguientes contienen los elementos de la matriz $L \times C$.

Salida:

La matriz rotada con `L` filas y `C` columnas.

2.12.6.3 📌 Ejemplos

Entrada	Salida	Observación
2 2 90 nearest 1 2 3 4	3 1 4 2	Rotación de 90° horaria: la columna 0 se convierte en la fila 0 (de abajo hacia arriba). $(0,0) = 1 \to (1,0)$, $(1,0) = 3 \to (0,0)$, $(0,1) = 2 \to (1,1)$, $(1,1) = 4 \to (0,1)$.
3 3 45 bilinear 0 0 0 0 255 0 0 0 0	0 180 0 180 255 180 0 180 0	Rotación de 45°: el píxel central permanece 255; los vecinos directos reciben un valor interpolado ≈ 180 por bilinear; las esquinas permanecen 0.

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

```
Overwriting EP02_06.cpp
```

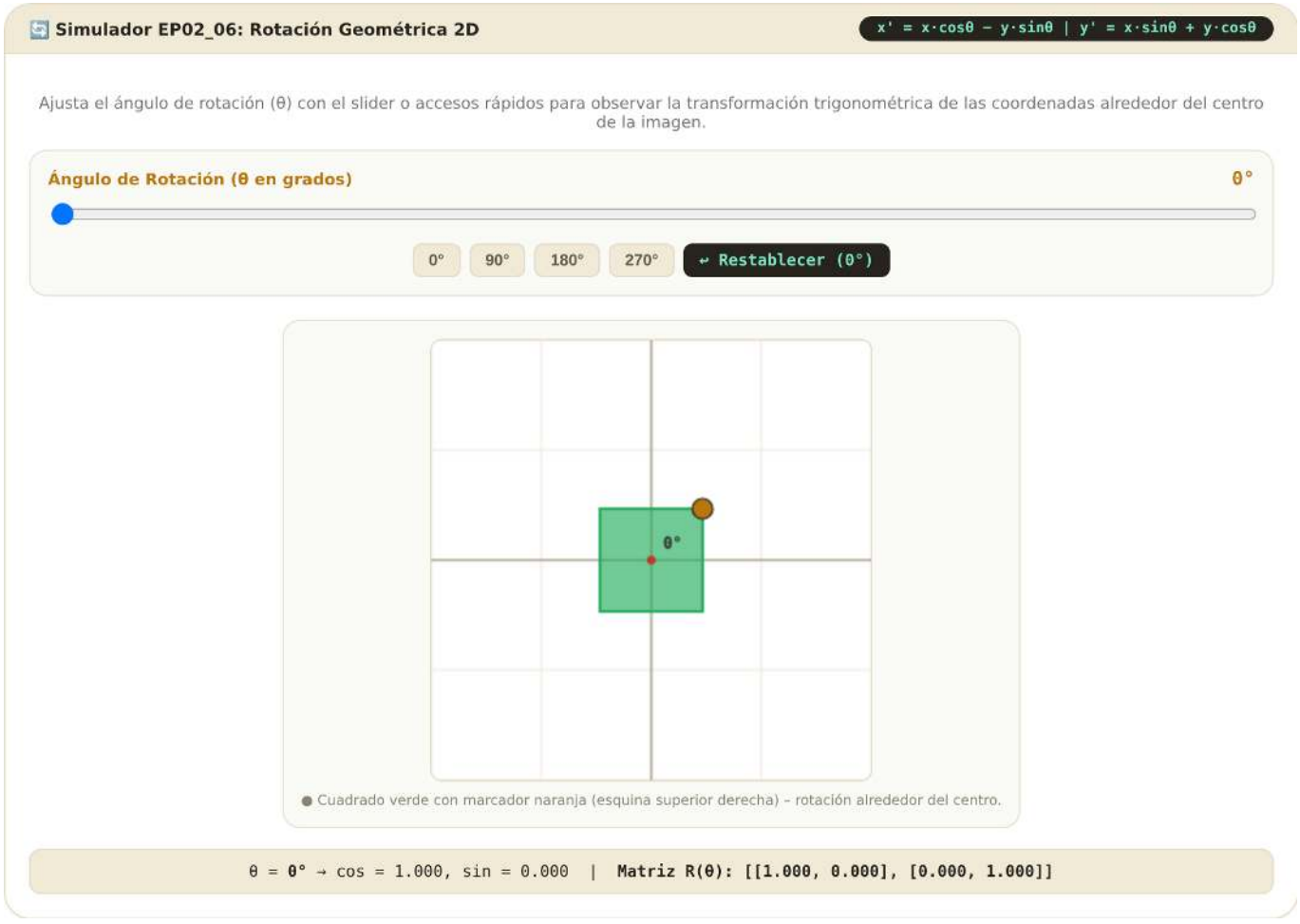
```
TestSuite("EP02_06.cpp").run()
```

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

2.12.7 EP02_07 🔍 Redimensionamiento (Escala)

En esta actividad, debes implementar el redimensionamiento de una imagen utilizando factores de escala. A diferencia del submuestreo simple, aquí utilizaremos técnicas de interpolación para permitir tanto la ampliación como la reducción de la imagen.

- Lee dos enteros **L** y **C**, que representan las dimensiones de la matriz original.
- Lee dos valores reales s_x (escala en las filas) y s_y (escala en las columnas).
- Lee una *cadena* que representa el **método** de interpolación (`nearest` o `bilinear`).
- Lee los valores enteros de la matriz original.
- Calcula las nuevas dimensiones: $L' = \text{round}(L \times s_x)$ y $C' = \text{round}(C \times s_y)$.



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

Figura 2.17: Simulador EP02_06: Rotación de Imagen en Torno al Origen por Ángulo

- Imprime la matriz resultante con las nuevas dimensiones.
- Ver en Figura 2.18 una simulación de este EP.

📌 **Importante**:

- **Mapeo inverso:** Para cada píxel (x', y') de la imagen de destino, encuentra la posición correspondiente en el origen usando $(x, y) = (x'/s_x, y'/s_y)$.
- **Interpolación:**
- `nearest`: Selecciona el valor del píxel más cercano (redondeo de las coordenadas).
- `bilinear`: Realiza una interpolación lineal doble entre los cuatro píxeles vecinos más cercanos en la imagen original.
- **Bordes:** Asegúrate de que el mapeo no intente acceder a índices fuera del intervalo $[0, L-1]$ y $[0, C-1]$.

2.12.7.1 🧠 Interpolación para Ampliación/Reducción

Redimensionar una imagen por factores (s_x, s_y) exige el relleno de vacíos (en la ampliación) o la fusión de información (en la reducción). El método de interpolación define la calidad visual del resultado:

Método	Funcionamiento	Efecto Visual
Nearest	Toma el valor del vecino más cercano.	Rápido, pero genera efecto "pixelado" o bloques.
Bilineal	Promedio ponderado de los 4 vecinos (2×2).	Suaviza la imagen, reduciendo el aliasing.

2.12.7.2 📋 Tarea (especificación para VPL)

Entrada:

La primera línea contiene `L`.

La segunda línea contiene `C`.

La tercera línea contiene los factores `sx` y `sy`.

La cuarta línea contiene el método `interp` (`nearest` o `bilinear`).

Las líneas siguientes contienen los elementos de la matriz $L \times C$.

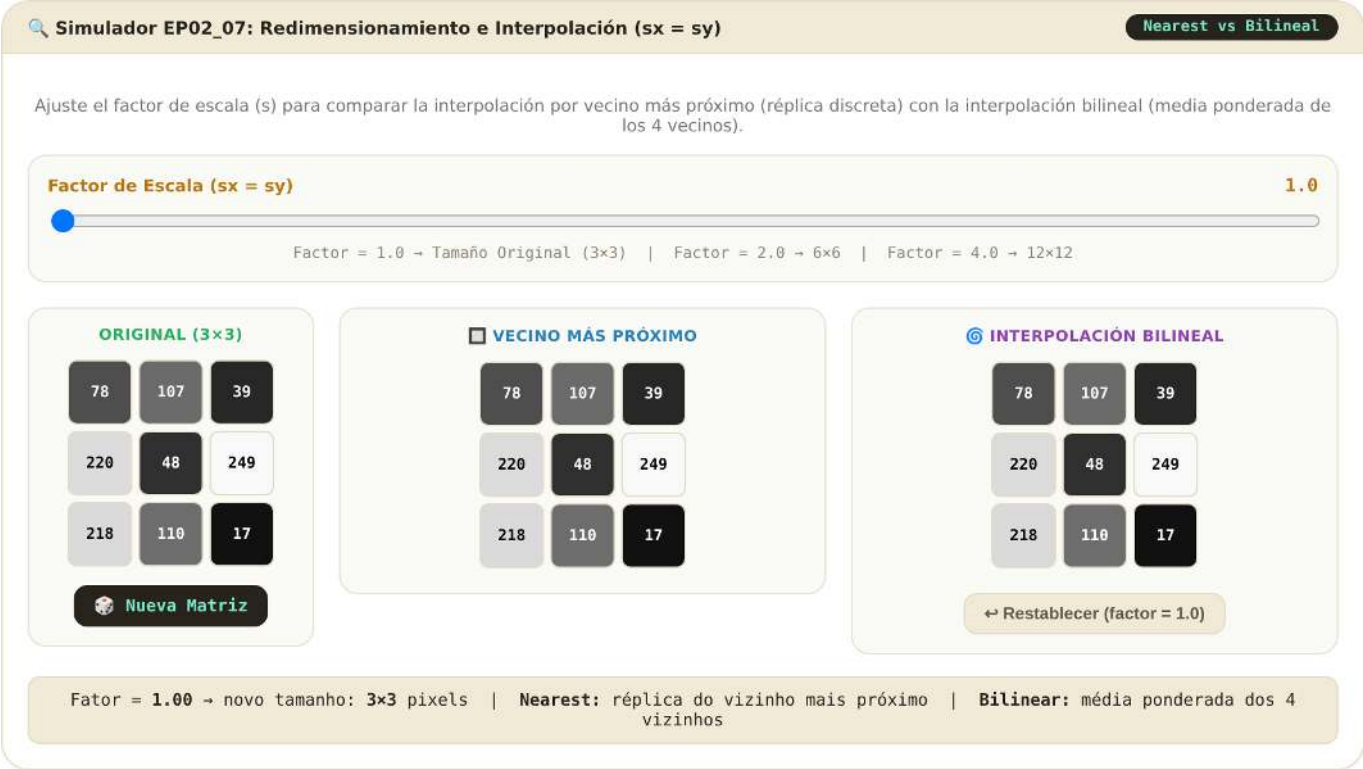
Salida:

La matriz redimensionada con dimensiones $L' \times C'$.

2.12.7.3 📌 Ejemplos

Entrada	Salida	Observación
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	Ampliación 2×: cada píxel original se replica en un bloque 2×2. La imagen 2×2 se convierte en 4×4.

Entrada	Salida	Observación
2 2 0.5 0.5 nearest 10 20 30 40	10	Reducción 0.5×: la imagen 2×2 se convierte en 1×1. Con nearest, el único píxel de salida muestrea la posición $(0, 0) = 10$.



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

Figura 2.18: Simulador EP02_07: Redimensionamiento Espacial e Interpolación (Vecino más cercano vs Bilineal)

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

# 1. Lectura de las dimensiones, factores y método
l = int(input())
c = int(input())
sx, sy = map(float, input().split())
interp = input().strip()

# 2. Lectura de la imagen original
img = mm.readImg(l, c)

# 3. Nuevas dimensiones
l_new = round(l * sx)
c_new = round(c * sy)

# 4. Redimensionamiento usando mm.resize
# cv2.resize usa (ancho, alto) = (columnas, filas)
```

```
resultado = mm.resize(img, (c_new, l_new), method=interp)

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

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

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

2.12.8 EP02_08 🔀 Cortante (Shear)

En esta actividad, debes implementar la transformación de cortante en una imagen. El cortante es una transformación afín que desplaza cada punto en una dirección fija, por un valor proporcional a su distancia de una recta paralela a esa dirección, resultando en un efecto de inclinación.

- Lee dos enteros **L** y **C**, que representan las dimensiones de la matriz.
- Lee dos valores reales sh_x (cortante horizontal) y sh_y (cortante vertical).
- Lee una *cadena* que representa el **método** de interpolación (`nearest` o `bilinear`).
- Lee los valores enteros de la matriz original.
- Aplica la transformación manteniendo el tamaño original de la imagen (recortando lo que exceda los límites).
- Imprime la matriz resultante con las dimensiones $L \times C$.
- Ver en la Figura 2.19 una simulación de este EP.

📌 **Importante**:

- **Mapeo Inverso:** Para cada píxel (x', y') de la imagen de destino, calcula la posición correspondiente en el origen (x, y) utilizando la matriz de cortante inversa.
- **Relleno:** Las coordenadas que resulten en posiciones fuera de la matriz original deben rellenarse con **0**.
- **Coordenadas:** Para fines de esta implementación, considera x como el índice de la fila e y como el índice de la columna.

2.12.8.1 🧠 Distorsión Afín

El cortante altera la geometría de la imagen inclinando sus ejes. La relación entre las coordenadas originales (x, y) y las transformadas (x', y') está dada por:

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

Esto resulta en las ecuaciones:

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

2.12.8.2 📋 Tarea (especificación para VPL)

Entrada:

La primera línea contiene `L`.

La segunda línea contiene `C`.

La tercera línea contiene los factores `shx` y `shy`.

La cuarta línea contiene el método `interp` (`nearest` o `bilinear`).

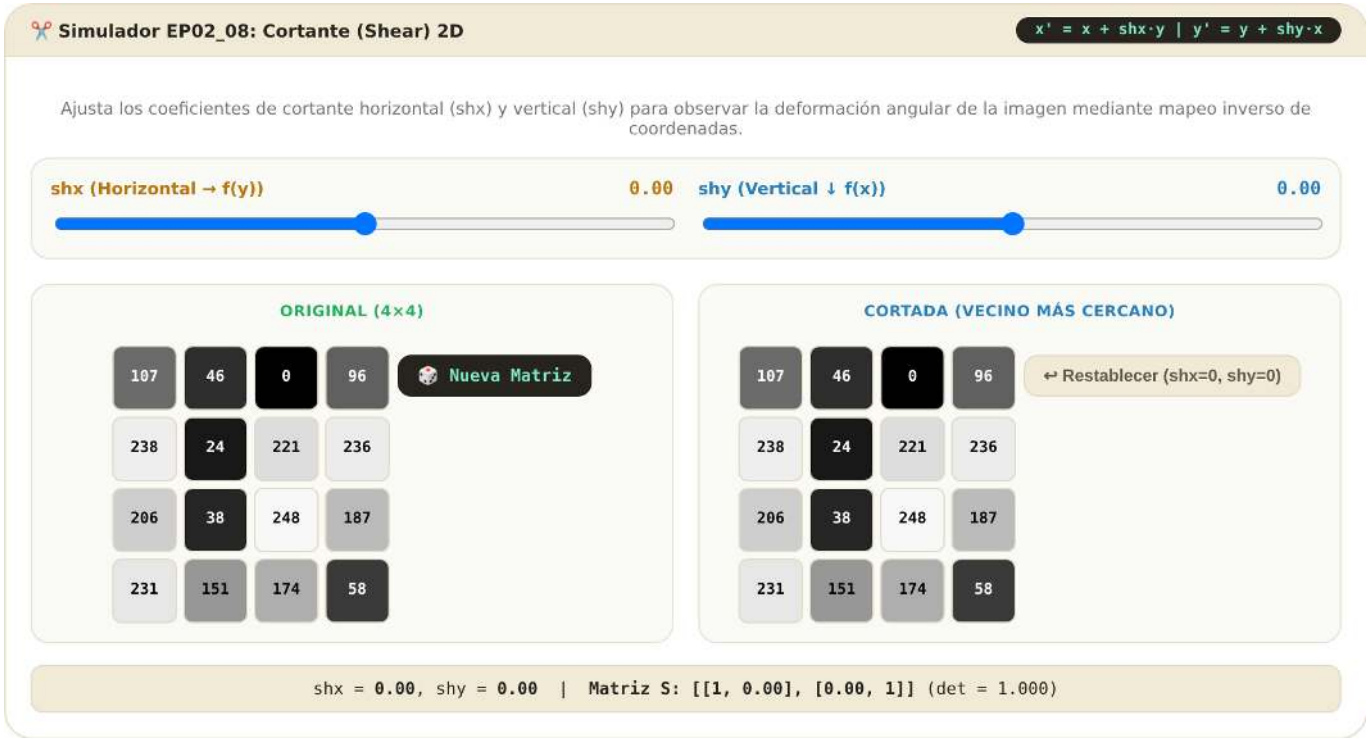
Las líneas siguientes contienen los elementos de la matriz $L \times C$.

Salida:

La matriz transformada con las mismas dimensiones $L \times C$.

2.12.8.3 📌 Ejemplos

Entrada	Salida	Observación
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	Cortante horizontal: fila i se desplaza $\lfloor i \cdot 0.5 \rfloor$ píxeles. Fila 0 → 0px, fila 1 → 0px, fila 2 → 1px. Los píxeles desplazados hacia fuera se descartan y las posiciones vacías se rellenan con 0.
2 2 0.0 1.0 nearest 10 20 30 40	10 0 30 20	Cortante vertical: columna j se desplaza $\lfloor j \cdot 1.0 \rfloor$ píxeles hacia abajo. Columna 0 → 0px (sin cambios), columna 1 → 1px: 20 baja a $(1, 1)$ y $(0, 1)$ queda 0.



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

Figura 2.19: Simulador EP02_08: Transformación Geométrica de Cortante 2D (Cizallamiento Horizontal y Vertical)

```
%%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 🧩 Transformación Afín Genérica

En esta actividad, debes implementar una transformación afín arbitraria sobre una imagen. Esta operación es la generalización de todas las transformaciones lineales (escala, rotación, cizallamiento) combinadas con la traslación, permitiendo manipulaciones geométricas complejas mediante una única matriz.

- Lee dos enteros **L** y **C**, que representan las dimensiones de la matriz.
- Lee **seis valores reales** (a, b, t_x, c, d, t_y) que componen la matriz de transformación afín 2×3.
- Lee una *cadena* que representa el **método** de interpolación (`nearest` o `bilinear`).
- Lee los valores enteros de la matriz original.
- Aplica la transformación manteniendo el tamaño original $L \times C$.
- Imprime la matriz resultante.
- Ver en Figura 2.20 una simulación de este EP.

📌 **Importante**:

- **Mapeo Inverso:** Para calcular el valor de cada píxel en la imagen de destino, debes utilizar la **inversa** de la matriz de transformación afín proporcionada para encontrar la coordenada correspondiente en la imagen original.
- **Relleno:** Las coordenadas calculadas que caigan fuera de los límites $[0, L-1]$ y $[0, C-1]$ de la imagen original deben resultar en un píxel de valor **0**.
- **Flexibilidad:** Esta implementación debe ser capaz de realizar cualquiera de las tareas anteriores (traslación, rotación, etc.) bastando con alterar los parámetros de la matriz.

Consejo:

```
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 🧠 Combinación de Operaciones

La transformación afín preserva puntos, rectas y planos. En el procesamiento de imágenes, mapea la posición (x, y) a (x', y') siguiendo el sistema:

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

O, de forma compacta en coordenadas homogéneas:

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

2.12.9.2 📋 Tarea (especificación para VPL)

Entrada:

La primera línea contiene `L`.

La segunda línea contiene `C`.

La tercera línea contiene seis flotantes: `a b tx c d ty`.

La cuarta línea contiene el método `interp` (`nearest` o `bilinear`).

Las líneas siguientes contienen los elementos de la matriz $L \times C$.

Salida:

La matriz transformada con las dimensiones originales $L \times C$.

2.12.9.3 📌 Ejemplos

Entrada	Salida	Observación
2 2 1.0 0.0 0.5 0.0 1.0 0.5 bilinear 10 20 30 40	15 20 25 30	Traslación fraccionaria ($t_x = 0.5, t_y = 0.5$): cada píxel de salida (i, j) muestrea la posición $(i + 0.5,\ j + 0.5)$ de la entrada mediante bilinear. Ej: $(0, 0)$ interpola los cuatro vecinos → 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	Escala 2× mediante matriz afín ($a = 2, d = 2$): cada píxel de salida (i, j) muestrea la posición $(2i, 2j)$ de la entrada con nearest. Ej: $(0, 2) \rightarrow (0, 4)$ fuera de la imagen → nearest recorta a $(0, 2) = 3$… espera confirmación de la lógica de borde.

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

```
Overwriting EP02_09.cpp
```

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

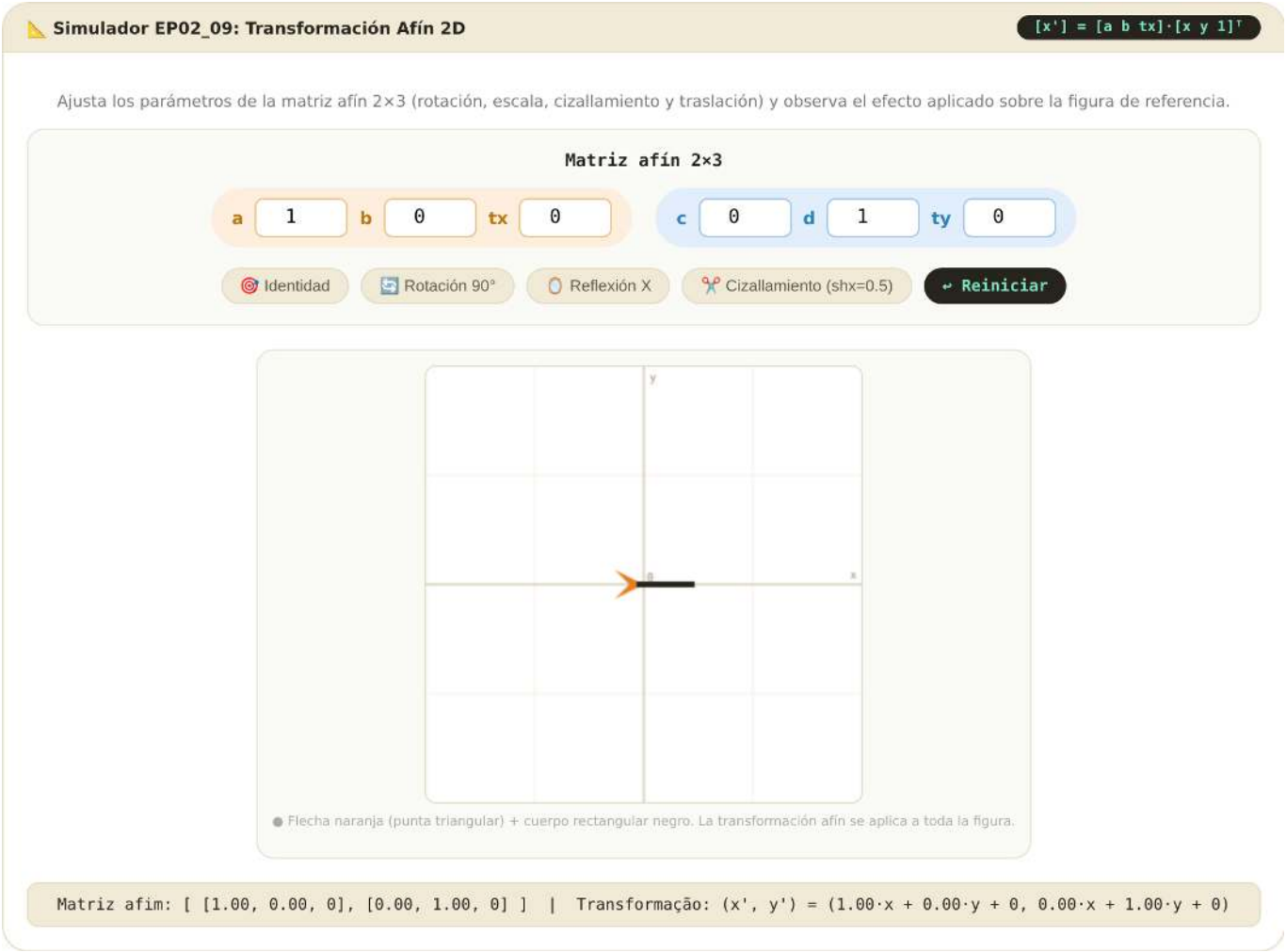
```
<IPython.core.display.HTML object>
```

2.12.10 EP02_10 🎯 Corrección de Perspectiva (Homografía)

En esta actividad, debes implementar la transformación de perspectiva, también conocida como homografía. A diferencia de las transformaciones afines, la perspectiva no preserva el paralelismo, permitiendo "rectificar" objetos inclinados, como documentos o placas capturados en ángulos oblicuos.

- Lee dos enteros **L** y **C**, que representan las dimensiones de la matriz original.
- Lee **cuatro pares de coordenadas** (x, y) que representan las esquinas del cuadrilátero de origen (objeto distorsionado).
- Lee **cuatro pares de coordenadas** (x, y) que representan las esquinas del cuadrilátero de destino (donde se debe mapear el objeto).
- Lee los valores de la matriz original.
- Calcula la matriz de homografía 3×3 y aplica la transformación.
- Imprime la matriz resultante con las dimensiones de salida especificadas.
- Consulta en Figura 2.21 una simulación de este EP.

📌 **Importante**:



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

Figura 2.20: Simulador EP02_09: Transformación Afín 2D (Matriz 2×3)

- **Grados de libertad:** La homografía posee 8 grados de libertad (el noveno elemento de la matriz 3×3 es una constante de normalización, generalmente 1), lo que requiere al menos 4 puntos correspondientes para calcularse.
- **Proyección:** Después de multiplicar las coordenadas por la matriz, es necesario dividir los resultados x' e y' por la componente homogénea w para regresar al plano 2D.
- **Uso de bibliotecas:** Para esta tarea, puedes utilizar las funciones `cv2.getPerspectiveTransform` para obtener la matriz y `cv2.warpPerspective` para aplicar la transformación, o implementar el sistema lineal y el mapeo inverso manualmente para un desafío adicional.

```
# Dimensiones de salida: bounding box de los puntos destino + 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))
# o
dst = mm.perspective_transform(img, pts1, pts2, size=(w, h))
```

2.12.10.1 🧠 Deformación no afín

Mientras que las transformaciones afines mapean paralelogramos en paralelogramos, la homografía mapea cualquier cuadrilátero en otro cuadrilátero. Esto es esencial para la visión por computadora:

Operación	Característica	Aplicación típica
Homografía	Proyección en plano	Corrección de documentos, escaneo de placas.
Punto de fuga	Convergencia de líneas	Reconstrucción 3D a partir de imágenes 2D.
Warping	Deformación de malla	Estabilización de video y panoramas (stitching).

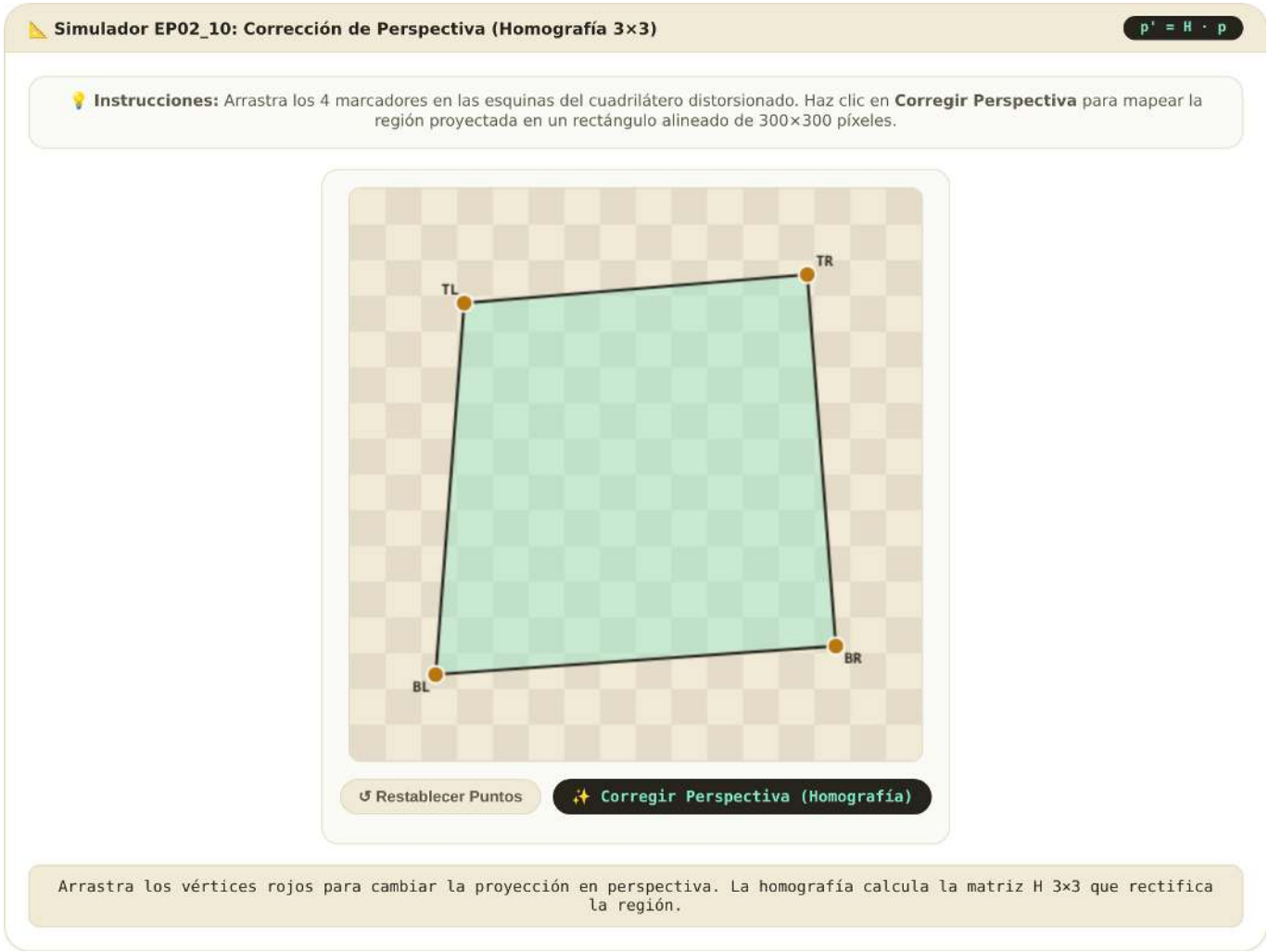
2.12.10.2 📌 Ejemplos

Entrada	Salida	Observación
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	Las 4 primeras líneas después de las dimensiones son los puntos de origen; las 4 siguientes son los destinos. Con puntos idénticos, la transformación de perspectiva es la identidad y la imagen se preserva.

```
%%writefile EP02_10.cpp
// your solution
```

```
Overwriting EP02_10.cpp
```

```
TestSuite("EP02_10.cpp").run()
```



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

Figura 2.21: Simulador EP02_10: Corrección de Perspectiva (Transformación de Homografía 3×3)

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

2.12.11 EP02_11 🏆 Corrección de Perspectiva (Homografía) en Imagen Real

En esta actividad, el objetivo es aplicar la transformación de perspectiva (homografía) para "rectificar" un objeto inclinado en una fotografía real. Trabajarás con la imagen de un periódico, donde la cuadrícula de un juego de Sudoku está distorsionada debido al ángulo en que se tomó la foto.

Tu programa debe leer los parámetros de entrada desde el terminal, cargar la imagen, calcular la matriz de homografía 3×3, aplicar la transformación geométrica y mostrar un indicador global de validación.

- Lee dos enteros **L** y **C**, que representan las dimensiones de filas y columnas (alto y ancho) que debe tener la imagen rectificada de salida.
- Lee **cuatro pares de coordenadas** (x, y) desde el terminal, que representan las cuatro esquinas del cuadrilátero de origen (el Sudoku distorsionado en la imagen original).
- Calcula automáticamente los **cuatro pares de coordenadas de destino** utilizando las dimensiones L y C proporcionadas, mapeando las esquinas a los extremos de la nueva imagen: $(0, 0)$, $(C-1, 0)$, $(0, L-1)$ y $(C-1, L-1)$.
- Carga la imagen local `sudoku.png` y conviértela a escala de grises.
- Calcula la matriz de homografía y aplica la transformación espacial en la imagen.
- **Salida:** Calcula e imprime la **suma de todos los píxeles** de la imagen resultante.

📌 **Importante**:

- **Archivo de entrada:** La imagen `sudoku.png` debe estar en la misma carpeta que el script. El programa debe leerla directamente del disco (por ejemplo, usando `mm::read("sudoku.png")` o `cv2.imread`).
- **Orden de los puntos:** Asegúrate de que la lectura de los 4 puntos de origen y la generación de los 4 puntos de destino sigan rigurosamente el mismo orden de las esquinas: Superior-Izquierda (TL), Superior-Derecha (TR), Inferior-Izquierda (BL) e Inferior-Derecha (BR).
- **Dimensiones en OpenCV:** Recuerda que funciones como `cv2.warpPerspective` esperan el tamaño de la imagen de salida en el formato `(ancho, alto)`, lo que equivale a `(C, L)`.
- **Interpolación:** Para garantizar la consistencia matemática de la suma de píxeles con el corrector automático, utiliza la interpolación bilineal estándar (`flags=cv2.INTER_LINEAR`).
- **Créditos:** La imagen utilizada es **"Sudoku en periódico"** de Héctor Rodríguez, bajo licencia **CC BY 2.0**.

2.12.11.1 🧠 Contexto del Problema

La homografía tiene 8 grados de libertad, lo que requiere al menos 4 correspondencias de puntos para calcularse. A diferencia de las transformaciones afines, mapea cualquier cuadrilátero en otro cuadrilátero, permitiendo que las líneas que convergen en puntos de fuga vuelvan a ser paralelas:

Operación	Característica	Aplicación Típica
Homografía	Proyección entre planos	Rectificación de documentos, escaneo de placas y códigos QR.
Mapeo Inverso	Recorrido del destino al origen	Evita "agujeros" o píxeles vacíos en la imagen final rectificada.
Warping	Remuestreo espacial	Corrección de distorsión de lentes y montaje de panoramas (*stitching*).

2.12.11.2 📌 Ejemplos

Entrada	Salida	Observación
500 500 100 120 420 95 80 440 450 460	32982820	Las dos primeras entradas son las dimensiones de salida (L y C). Las 4 líneas siguientes son las coordenadas (x, y) de las esquinas del Sudoku en la imagen original + PAD. La salida es la suma total de los píxeles de la imagen rectificada.
200 200 100 120 420 95 80 440 450 460	5277150	Mismos puntos de origen que el ejemplo anterior, pero generando una imagen de salida más pequeña (200×200). La suma de píxeles se reduce proporcionalmente debido a la escala.

2.12.11.3 Adquisición de la imagen del sudoku y conversión a niveles de gris

La Figura 2.22 muestra la lectura de la imagen original seguida de la conversión a tonos de gris y el redimensionamiento a una matriz de 500×500 píxeles, preparando los datos para la etapa siguiente.

La corrección de perspectiva, aplicada en la Figura 2.23 mediante la matriz de homografía, elimina las deformaciones causadas por el ángulo de la cámara y produce una vista frontal y regular de la cuadrícula 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
#include "morph.hpp"
#include <iostream>
#include <filesystem>

int main() {
    //| output: true

    std::string url =
"https://upload.wikimedia.org/wikipedia/commons/e/e7/Sudoku_en_peri%C3%B3dico.jpg";
    mm::Image sudoku_rgb = mm::read(url);
    mm::Image sudoku_gray = mm::gray(sudoku_rgb);
    mm::Image sudoku_small = mm::resize(sudoku_gray, 500, 500);
    mm::write(sudoku_small, "sudoku.png");   // consumida pela célula fig-02-sudoku2

    mm::show(
        std::vector<mm::Image>{sudoku_rgb, sudoku_small},
        MM_OUT,
        std::vector<std::string>{"Original RGB", "Cinza 500x500"},
        2
    );


// [pdi:panel-io] auto-generated - do not edit by hand
std::filesystem::create_directories("tmp");
mm::write(sudoku_rgb, "tmp/fig_02_sudoku_original_0.png");
mm::write(sudoku_small, "tmp/fig_02_sudoku_original_1.png");
```

```
// [pdi:panel-io:end]
return 0;
}
```

```
Overwriting tmp/fig_02_sudoku_original.cpp
```

```
!g++ -I. -std=c++17 tmp/fig_02_sudoku_original.cpp -o tmp/fig_02_sudoku_original \
  && ./tmp/fig_02_sudoku_original \
  && test -f "tmp/fig_02_sudoku_original.png" \
  || echo "  mm::show não gravou tmp/fig_02_sudoku_original.png"
```

```
[1] Original RGB
[2] Cinza 500x500
```

```
try:
    mm.show(
        [
            mm.read("tmp/fig_02_sudoku_original_0.png"),
            mm.read("tmp/fig_02_sudoku_original_1.png"),
        ],
        titles=[
            'Original RGB',
            'Cinza 500x500',
        ],
        cols=2,
    )
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + "
    tmp/fig_02_sudoku_original_0.png (ver a versao Python)")
```

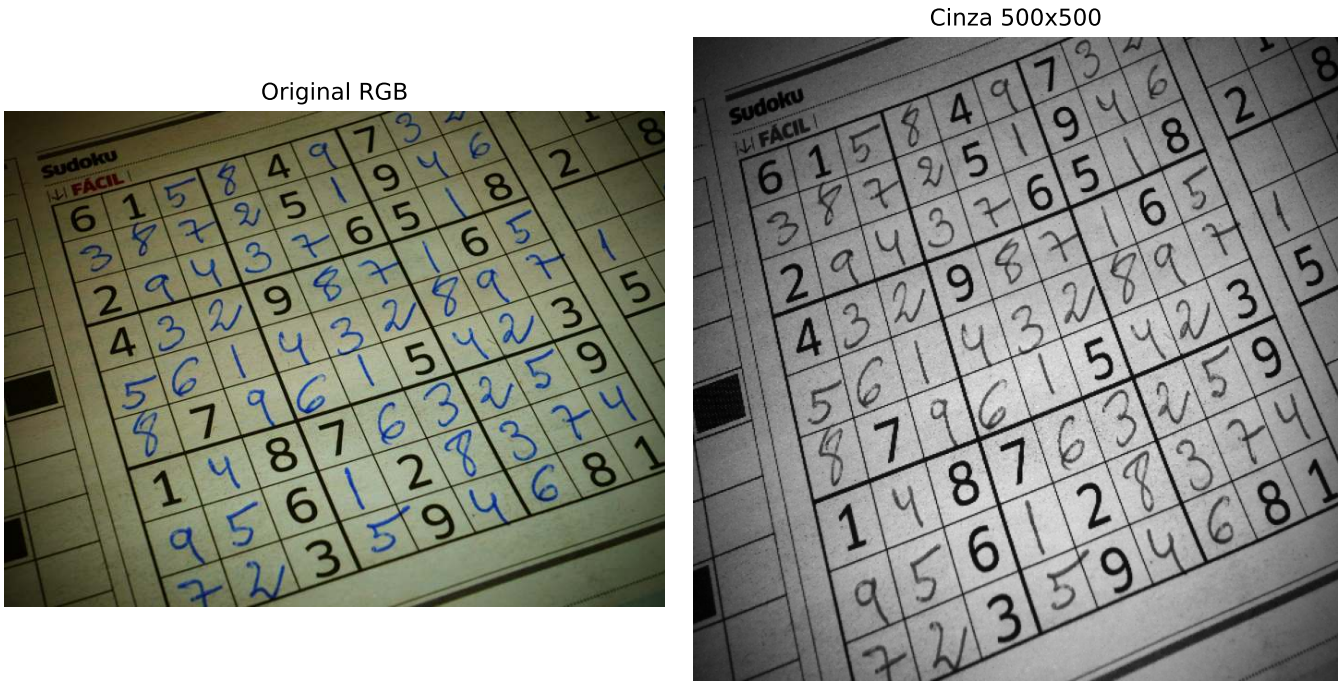


Figura 2.22: Aquisição da imagem de um Sudoku à esquerda. À direita, conversão para tons de cinza e redimensionamento. Crédito: Héctor Rodríguez de Guardamar, Espanha (CC BY 2.0).

```
%%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"
#include <vector>
#include <string>
#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)
    const int PAD = 60;
    mm::Image img_pad = mm::pad(img, PAD);

    // 3. Cantos da grade na imagem expandida - TL, TR, BL, BR
    // Nota: pts1 (4 pontos) como std::vector de pares (x, y)
    std::vector<std::pair<int,int>> pts1 = {{100, 160}, {390, 45}, {200, 580}, {570, 420}};

    // 4. Cantos de destino: vista frontal SIZE×SIZE
    const int SIZE = 500;
    std::vector<std::pair<int,int>> pts2 = {{0, 0}, {SIZE, 0}, {0, SIZE}, {SIZE, SIZE}};

    // 5. Homografia pts1 -> pts2 e retificação
    // (Nota: mm::perspective_transform nesta versão pode não estar disponível;
    //  este é um placeholder - veja comentários para implementação completa)
    // mm::Image img_rect = mm::perspective_transform(img_pad, pts1, pts2, SIZE, SIZE);
    mm::Image img_rect;
    // TODO: implementar homografia aqui ou usar a API correspondente se disponível

    // 6. Exibição
    mm::show(
        std::vector<mm::Image>{img_pad, img_rect},
        MM_OUT,
        std::vector<std::string>{"Original (com padding)", "Vista frontal retificada"},
        2
    );


// [pdi:panel-io] auto-generated - do not edit by hand
std::filesystem::create_directories("tmp");
mm::write(img_pad, "tmp/fig_02_sudoku2_0.png");
mm::write(img_rect, "tmp/fig_02_sudoku2_1.png");
// [pdi:panel-io:end]
return 0;
}
```

```
Overwriting tmp/fig_02_sudoku2.cpp
```

```
!g++ -I. -std=c++17 tmp/fig_02_sudoku2.cpp -o tmp/fig_02_sudoku2 \
  && ./tmp/fig_02_sudoku2 \
  && test -f "tmp/fig_02_sudoku2.png" \
  || echo "  mm::show não gravou tmp/fig_02_sudoku2.png"
```

```
[1] Original (com padding)
[2] Vista frontal retificada
```

```
try:
    mm.show(
        [
            mm.read("tmp/fig_02_sudoku2_0.png"),
            mm.read("tmp/fig_02_sudoku2_1.png"),
        ],
        titles=[
            'Original (com padding)',
            'Vista frontal retificada',
        ],
        cols=2,
    )
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + " tmp/fig_02_sudoku2_0.png
    (ver a versao Python)")
```

```
figura indisponivel nesta trilha (C++): UnidentifiedImageError("cannot identify image file
'tmp/fig_02_sudoku2_1.png'") tmp/fig_02_sudoku2_0.png (ver a versao Python)
```

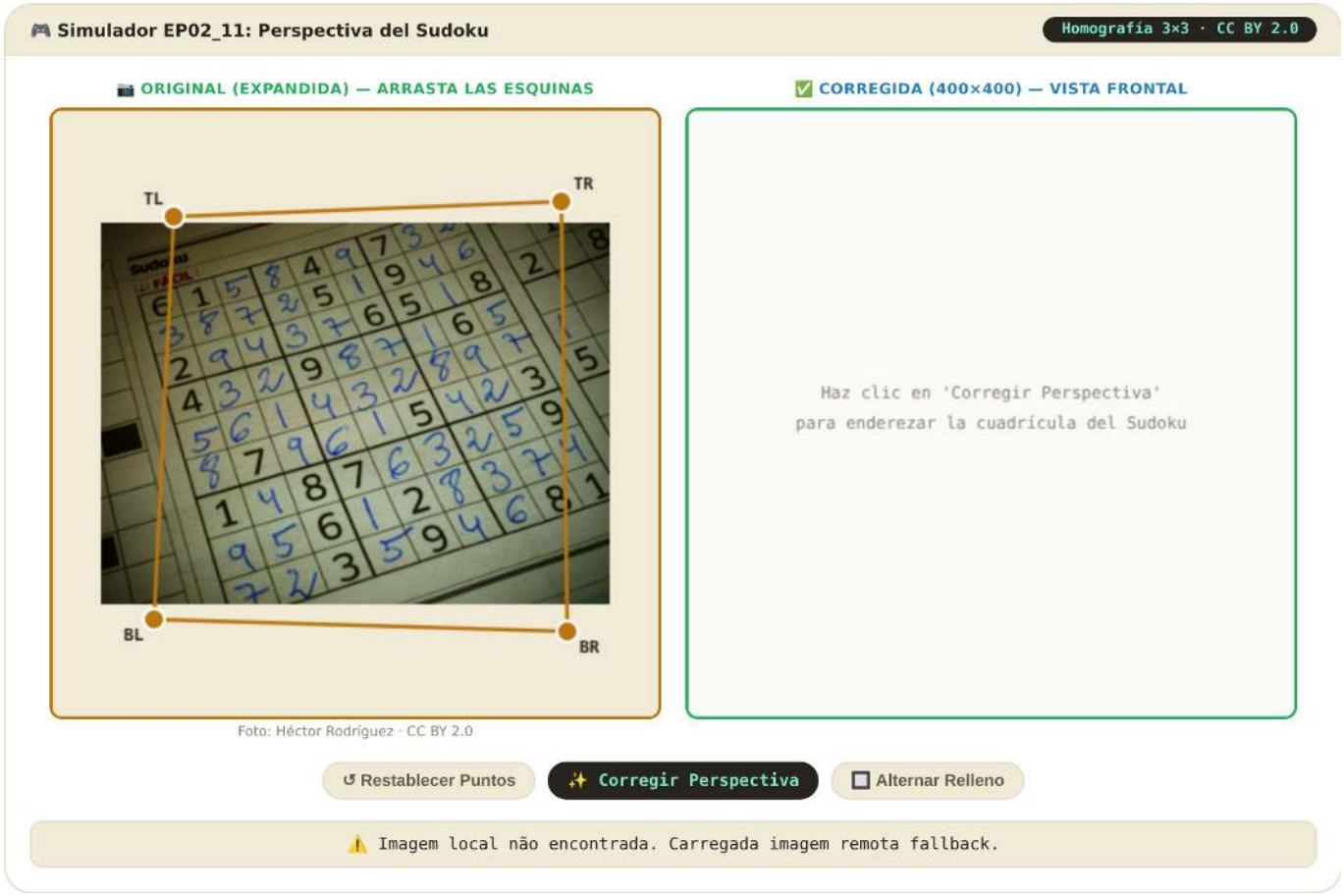
Figura 2.23

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

```
Overwriting EP02_11.cpp
```

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

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



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

Figura 2.24: Simulador EP02_11: Corrección de Perspectiva del Sudoku (Homografía 3×3 con Remuestreo Bilineal)

Capítulo 3

Operaciones Espaciales: Intensidad, Histograma y Filtrado

Este capítulo profundiza en el procesamiento de imágenes en el dominio espacial, partiendo de la manipulación directa de píxeles e histogramas para el realce de contraste, hasta la aplicación de filtros locales por convolución para suavizado, reducción de ruido y detección de bordes. El objetivo es desarrollar la intuición matemática y computacional que sustenta gran parte de los algoritmos modernos de Visión Computacional.

3.1 Objetivos

Al final de este capítulo, usted será capaz de:

- **Manipular intensidad y píxeles:** Ejecutar operaciones aritméticas saturadas (`mm::addm`, `mm::subm`) y lógicas bit a bit (`mm::band`, `mm::bor`, `mm::bnot`) para combinación y selección de regiones de interés (ROI), y aplicar *alpha blending* (`mm::blend`) para fusión ponderada de imágenes;
- **Procesar histogramas:** Interpretar el histograma como diagnóstico tonal y aplicar ecualización global mediante CDF (`mm::equalize`); en la ruta Python, también la ecualización adaptativa (CLAHE) y la especificación de histograma para transferencia de perfil tonal entre imágenes;
- **Comprender fundamentos espaciales:** Entender vecindad, *padding* de borde (`mm::pad`) y la diferencia entre correlación cruzada (`mm::conv`) y convolución — incluyendo por qué *kernels* asimétricos como el de Sobel producen resultados distintos en las dos operaciones;
- **Aplicar filtrado de suavizado:** Usar el filtro de media (`mm::blur`, o `mm::conv` con *kernel* uniforme) y el filtro Gaussiano (`mm::gaussian`) para reducción de ruido, comprendiendo la ventaja de la ponderación radial y de la separabilidad Gaussiana;
- **Aplicar filtrado de realce:** Usar el Laplaciano w_4 y w_8 (`mm::laplacian`) para realce isotrópico de bordes, el operador de Sobel (`mm::sobel`) para la magnitud del gradiente — y, en la ruta Python, la descomposición direccional G_x, G_y y ángulo —, y el *Unsharp Masking* (`mm::usm`) para amplificación de alta frecuencia controlada por el parámetro k;
- **Utilizar filtros de orden:** Aplicar el filtro de la mediana (`mm::median`) para eliminación de ruido sal y pimienta, comprendiendo por qué su naturaleza no lineal y la robustez a *outliers* lo hacen superior a los filtros lineales en ese escenario;
- **Resolver problemas prácticos:** Encadenar técnicas en *pipelines* de preprocesamiento (ecualización → Gaussiano → Canny; con CLAHE en lugar de la ecualización en la ruta Python) y usar las funciones de `morph` (`mm::conv`, `mm::histImg`, `mm::equalize`, `mm::drawImgKernel`) para análisis y visualización didáctica de cada etapa.

3.2 Operaciones a Nivel de Intensidad

El nivel más elemental de procesamiento de imágenes actúa directamente sobre los valores de los píxeles, sin considerar la vecindad. Estas operaciones —llamadas **transformaciones de punto** (*point operations*)— son las más rápidas computacionalmente y forman la base para técnicas más complejas.

Formalmente, una transformación de punto puede describirse como:

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

donde $f(x, y)$ es la imagen de entrada, $g(x, y)$ es la salida y T es una función aplicada a cada píxel individualmente.

3.2.1 Preparando el Entorno Práctico

El siguiente bloque carga la biblioteca `morph` del repositorio (el módulo `morph.py` y, en la ruta C++, también el `morph.hpp` utilizado en el `#include` de las celdas compiladas).

```
import os, urllib.request

os.makedirs("tmp/state", exist_ok=True)  # artefatos de build de la pista C++ (.cpp, binario,
PNGs)

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

# El kernel es Python incluso en la pista C++: `mm` (morph.py) es usado por los
# simuladores, por la visualización de las figuras que el binario C++ genera y por el
# estado mm::Image entre celdas. cpp=True también descarga la pista compilada
# (morph.hpp + stb_image*.h), usada en el #include de las celdas %%writefile *.cpp.
import config
config.setup(cpp=True)
from morph import mm
import numpy as np
```

```
Entorno listo. Morph: 1.1.9 | OpenCV: 5.0.0
```

Como objeto de estudio a lo largo de este capítulo, utilizaremos las imágenes de vida silvestre presentadas en las Figura 3.1 y Figura 3.3.. A partir de ellas, exploraremos operaciones espaciales sobre intensidad, histogramas y filtrado, analizando sus efectos en el realce, la suavización, la reducción de ruido y la detección de bordes, con el fin de comprender los fundamentos matemáticos y computacionales del 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 Operaciones Aritméticas

Las operaciones aritméticas entre imágenes se utilizan ampliamente en PDI para combinar, comparar o realzar información. La **resta de imágenes** es especialmente poderosa para detectar diferencias entre dos cuadros — por ejemplo, en la eliminación del fondo estático en cámaras de vigilancia:

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

La **suma saturada** limita el resultado al intervalo $[0, 255]$: los valores superiores a 255 se fijan en 255, evitando el *overflow* silencioso del tipo `uint8` (p. ej., $200 + 100 = 44$ en lugar de 300). La **resta saturada** aplica el mismo principio por el lado inferior: los valores negativos se fijan en 0.

> ⚠ Saturación y *overflow*
>
> Las operaciones aritméticas en `uint8` sufren *overflow* silencioso: $200 + 100 = 44$ (no 300). `mm::addm` y `mm::subm` realizan la **saturación automática**, fijando el resultado en $[0, 255]$. El *blending* utiliza pesos fraccionarios: `mm::blend` opera internamente en punto flotante y solo entonces redondea y satura a `uint8`.

La Figura 3.2 demuestra la suma de una constante (aclarado) y la resta de una constante (oscurecimiento con saturación en 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 Mezcla Ponderada (*Alpha Blending*)

La **mezcla ponderada** (*alpha blending*) combina dos imágenes utilizando pesos complementarios α y $(1-\alpha)$:

$$g(x,y) = \alpha\, f_1(x,y) + (1-\alpha)\, f_2(x,y), \quad \alpha \in [0,1] \tag{3.3}$$

Cuando $\alpha = 1$, se obtiene únicamente la imagen f_1; cuando $\alpha = 0$, solo f_2. Los valores intermedios producen una transición suave entre ambas, siendo ampliamente utilizados en composición de imágenes, superposición de capas, marcas de agua y efectos de fusión visual.

Para que la combinación produzca un resultado coherente, es necesario alinear previamente las regiones de interés. En la Figura 3.4, se recorta el rostro del leopardo con `mm::crop(img_leop_gray, 250, H-300, 100, W-200)` y la región facial del mandril con `mm::crop(img_gray, 100, 400, 380, 530)`, de modo que los ojos y la estructura facial queden aproximadamente alineados. El recorte del leopardo se redimensiona entonces (`mm::resize`) a las dimensiones del mandril antes de la mezcla.

`mm::blend` realiza la operación en punto flotante — evitando *overflow* en los cálculos con pesos fraccionarios — y solo entonces redondea y satura el resultado a `uint8`.

```
%%writefile tmp/fig_03_leopardo.cpp
#define MM_OUT "tmp/fig_03_leopardo.png"
// Compile: g++ -std=c++17 -o programa programa.cpp -I. -lcurl -lpng -ljpeg

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

int main() {
    //| 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

    mm::Image img_leop =
mm::read("https://upload.wikimedia.org/wikipedia/commons/9/92/Leopard_%28Panthera_pardus%29_portrait.jpg
```

```
    mm::Image img_leop_gray = mm::gray(img_leop);

    mm::show(img_leop, MM_OUT);


// [pdi:state-io] auto-generated - do not edit by hand
std::filesystem::create_directories("tmp/state");
mm::write(img_leop, "tmp/state/img_leop_12.png");
mm::write(img_leop_gray, "tmp/state/img_leop_gray_12.png");
// [pdi:state-io:end]
return 0;
}
```

```
Overwriting tmp/fig_03_leopardo.cpp
```

```
!g++ -I. -std=c++17 tmp/fig_03_leopardo.cpp -o tmp/fig_03_leopardo \
  && ./tmp/fig_03_leopardo \
  && test -f "tmp/fig_03_leopardo.png" \
  || echo "  mm::show não gravou tmp/fig_03_leopardo.png"
```

```
try:
    mm.show(mm.read("tmp/fig_03_leopardo.png"))
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + " tmp/fig_03_leopardo.png
    (ver a versao Python)")
```

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(
        std::vector<mm::Image>{mm::blend(mandrill, leo_r, 1.0), mm::blend(mandrill, leo_r,
0.8),
                               mm::blend(mandrill, leo_r, 0.6), mm::blend(mandrill, leo_r,
0.4),
                               mm::blend(mandrill, leo_r, 0.2), mm::blend(mandrill, leo_r,
0.0)},
        MM_OUT,
        std::vector<std::string>{" =1.0", " =0.8", " =0.6", " =0.4", " =0.2", " =0.0"},
        6
    );

    return 0;
}
```

```
Overwriting tmp/fig_03_blend.cpp
```

```
!g++ -I. -std=c++17 tmp/fig_03_blend.cpp -o tmp/fig_03_blend \
  && ./tmp/fig_03_blend \
  && test -f "tmp/fig_03_blend.png" \
  || echo "  mm::show não gravou tmp/fig_03_blend.png"
```

```
[1]  =1.0
[2]  =0.8
[3]  =0.6
[4]  =0.4
[5]  =0.2
[6]  =0.0
```

```
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)")
```

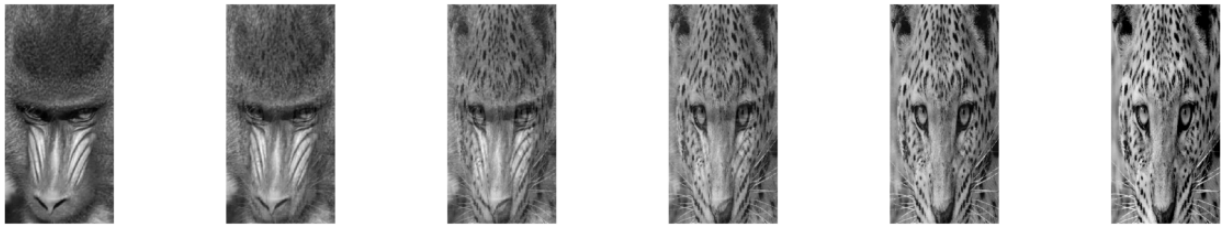


Figura 3.4: *Alpha blending* entre recortes alinhados de mandrill e do leopardo (Figura 3.3) para diferentes valores de . Em =1 vê-se apenas mandrill; em =0, apenas o leopardo; valores intermediários fundem os olhares das duas imagens proporcionalmente.

3.2.4 Operaciones Lógicas y Máscaras Bit a Bit

Las operaciones lógicas bit a bit (AND, OR y NOT) actúan directamente sobre los bits de cada píxel y son la base para la creación y aplicación de **máscaras** (*masks*) —imágenes binarias con solo 0 (negro) y 255 (blanco) utilizadas para aislar **Regiones de Interés** (**ROI**).

El comportamiento de cada operación se deriva de la representación binaria del 255 (`11111111`) y del 0 (`00000000`):

- **AND** con la máscara: donde $m = 255$, los bits originales se preservan; donde $m = 0$, el píxel se pone a cero. Resultado: recorte de la ROI.

$$g(x, y) = f(x, y) \text{ AND } m(x, y) \tag{3.4}$$

- **OR** con la máscara: donde $m = 255$, el píxel se fuerza a blanco; donde $m = 0$, se mantiene el valor original. Resultado: iluminación de la ROI.
- **NOT** (sin máscara): invierte todos los bits ($g = 255 - f$), produciendo el negativo fotográfico de la imagen.

La Figura 3.5 ilustra las tres operaciones aplicadas a la imagen del mandril con una máscara circular.

```
%%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 <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]

    // img_gray ya está inicializado (proporcionado automáticamente)

    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);
```

```
    int radius = std::min(h, w) / 3 - 10;
    mask_circ = mm::circle(mask_circ, w / 2, h / 2, radius, 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)")
```

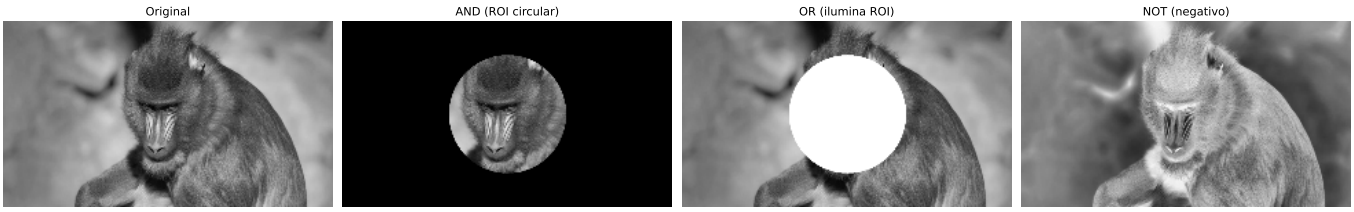


Figura 3.5: Operações lógicas bit a bit com máscara circular: AND (isolamento da ROI), OR (iluminação da ROI) e NOT (negativo).

3.3 Histograma de Imágenes

El **histograma** de una imagen en tonos de gris es una función discreta que describe la distribución de frecuencias de las intensidades:

$$h(r_k) = n_k, \quad k = 0, 1, \ldots, L-1 \tag{3.5}$$

donde r_k es el k-ésimo nivel de intensidad, n_k es el número de píxeles con esa intensidad y L es el total de niveles (típicamente 256 para 8 bits). El histograma normalizado estima la probabilidad de cada nivel:

$$p(r_k) = \frac{n_k}{MN} \tag{3.6}$$

donde MN es el total de píxeles. Por ser una **estadística global**, el histograma no contiene información posicional, pero revela características esenciales como brillo medio, contraste y distribución tonal. En la práctica, `mm::hist(img)` devuelve el vector de conteos $h(r_k)$, que sirve tanto para visualización (mediante `mm::histImg`) como para cálculos como **función de distribución acumulada (CDF)** y ecualización.

> **i** Interpretación del Histograma
>
> - **Estrecho a la izquierda:** imagen subexpuesta (oscura).
> - **Estrecho a la derecha:** imagen sobreexpuesta (clara).
> - **Concentrado en el centro:** bajo contraste.
> - **Distribuido por todo el rango:** alto contraste, buena utilización de los tonos disponibles.

La Figura 3.6 presenta el histograma de la imagen del mandril, así como versiones oscurecida (`mm::subm`) y aclarada (`mm::addm`). Se observa el desplazamiento de la distribución de intensidades hacia la izquierda y hacia la derecha, respectivamente. Nótese que el intervalo representado en el eje x no corresponde necesariamente a todo el rango de 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 <iostream>
#include <vector>
#include <string>

int main() {
// [pdi:state-io] auto-generated - do not edit by hand
```

```
mm::Image img_gray = mm::_read_state("tmp/state/img_gray_8.png");
// [pdi:state-io:end]

    // img_gray is provided already initialized

    mm::Image img_dark = mm::subm(img_gray, 80);
    mm::Image img_high = mm::addm(img_gray, 80);

    mm::show(
        std::vector<mm::Image>{img_gray, img_dark, img_high,
             mm::histImg(img_gray), mm::histImg(img_dark), mm::histImg(img_high)},
        MM_OUT,
        std::vector<std::string>{"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)")
```

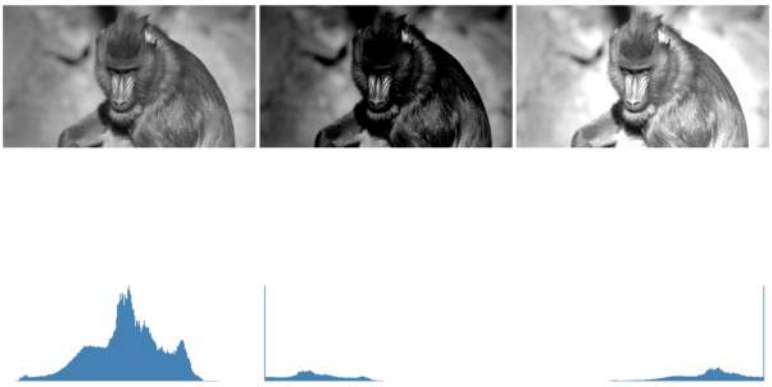


Figura 3.6: Histogramas da imagem original, de uma versão escurecida (−80) e de uma clareada (+80). A subtração/adição satura em 0 e 255.

3.3.1 Ecualización de Histograma

La **ecualización de histograma** redistribuye las intensidades para que el histograma resultante sea lo más uniforme posible. El mapeo está dado por la **función de distribución acumulada (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 transformación es monótona: los niveles frecuentes reciben intervalos mayores en el dominio de salida (mayor separación → más contraste), mientras que los niveles raros se comprimen.

El algoritmo completo, en cinco etapas, se presenta en la Tabla 3.1.

Tabla 3.1: Algoritmo de ecualización de histograma.

Etapa	Operación	Fórmula
1	**Histograma**	$h[k] \leftarrow$ número de píxeles con intensidad k, $k = 0 \dots L-1$
2	**Probabilidad**	$p[k] \leftarrow h[k]/MN$
3	**CDF**	$\text{cdf}[k] \leftarrow \sum_{j=0}^{k} p[j]$ (suma acumulada)
4	***Look-Up Table* (mapeo)**	$\text{lut}[k] \leftarrow \text{round}(\text{cdf}[k] \times (L-1))$
5	**Aplicación**	$g[i,j] \leftarrow \text{lut}[f[i,j]]$ (para todo píxel)

Observe en la Figura 3.7 que la ecualización **redistribuye** los tonos existentes a posiciones más espaciadas en el rango $[0, L-1]$, pero no crea tonos nuevos — la imagen ecualizada continúa con exactamente 3 tonos distintos, ahora en $\{1, 5, 7\}$ en lugar de $\{2, 3, 4\}$.

```
%%writefile tmp/fig_03_equalizacao_didatica.cpp
#define MM_OUT "tmp/fig_03_equalizacao_didatica.png"
//| 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
#include "morph.hpp"
#include <iostream>
#include <vector>

int main() {
    mm::Image img5(5, 5);
    int vals[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) = vals[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)")
```

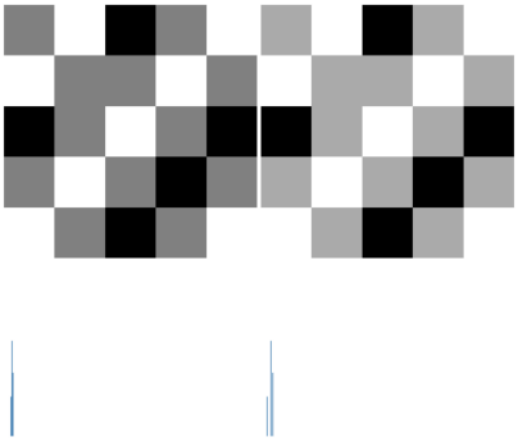


Figura 3.7: Equalização de histograma numa imagem 5×5 de 3 bits (L=8): tons concentrados em {2,3,4} são redistribuídos pela CDF. *mm::equalize(img, 3)* faz o mapeamento.

Limitación: la ecualización global puede realzar en exceso los ruidos y producir contraste excesivo en regiones homogéneas. El **CLAHE** (*Contrast Limited Adaptive Histogram Equalization*) reduce este problema al aplicar la ecualización en bloques locales (*tiles*) y limitar la altura de los picos del histograma antes de la ecualización.

La Figura 3.8 compara la imagen original, la ecualización global mediante `mm::equalize` y el CLAHE de OpenCV, mostrando también los histogramas resultantes. A diferencia de la ecualización global, que utiliza una única transformación basada en la CDF de toda la imagen, el CLAHE adapta el contraste a cada región, siendo particularmente útil en imágenes con iluminación no uniforme.

En el ejemplo, se utilizó `clipLimit=2.0` y `tileGridSize=(32,32)`. El parámetro `clipLimit` define cuánto pueden crecer los picos del histograma local antes de ser recortados (*clipped*). En OpenCV, este valor es un factor relativo: el límite real se calcula aproximadamente como `clipLimit × (número de píxeles del bloque / número de niveles de gris)`. Por ejemplo, en un bloque con 4096 píxeles y una imagen de 8 bits (256 niveles de gris), la frecuencia media por nivel es $4096/256 = 16$. Así, `clipLimit=2.0` permite picos de aproximadamente $2 \times 16 = 32$ ocurrencias antes del recorte. Las ocurrencias excedentes no se descartan: se redistribuyen entre los demás niveles de gris del histograma, reduciendo la concentración excesiva en pocos niveles y evitando una amplificación exagerada del contraste local. Valores menores limitan más el contraste y reducen la amplificación del ruido, mientras que valores mayores permiten un realce más intenso, pero pueden introducir artefactos.

- `clipLimit=1.0`: realce suave y conservador;
- `clipLimit=2.0`: buen equilibrio entre contraste y naturalidad;
- `clipLimit=4.0`: mayor énfasis en los detalles locales;
- `clipLimit=8.0`: contraste agresivo, con posible amplificación del ruido.

Así, el CLAHE suele producir resultados más naturales que la ecualización global, especialmente en imágenes con sombras, reflejos o iluminación desigual.

```
%%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

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

int main() {
// [pdi:state-io] auto-generated - do not edit by hand
mm::Image img_gray = mm::_read_state("tmp/state/img_gray_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
```

```
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)")
```

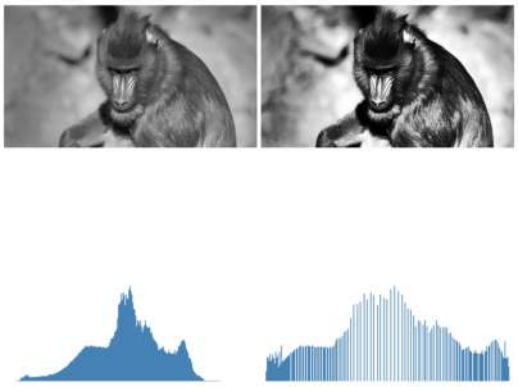


Figura 3.8: Equalização de histograma global (mm::equalize, via CDF) e os histogramas antes/depois. CLAHE (adaptativa) fica só na trilha Python — não tem equivalente em morph.hpp.

3.3.2 Especificación de Histograma

Mientras que la ecualización impone una distribución uniforme, la **especificación de histograma** (*histogram matching*) permite que el histograma de la imagen de salida siga una distribución **arbitraria** — por ejemplo, el histograma de otra imagen de referencia.

El procedimiento involucra tres etapas:

1. Calcular la CDF de la imagen de entrada: $P_r(r_k)$.
2. Calcular la CDF de la imagen de referencia: $P_z(z_k)$.
3. Para cada nivel r_k, encontrar el nivel z que minimiza $|P_z(z) - P_r(r_k)|$.

$$T(r_k) = \arg\min_z |P_z(z) - P_r(r_k)| \tag{3.8}$$

En la Figura 3.9, transferimos el perfil tonal del leopardo (Figura 3.3) a la imagen del mandril — una aplicación directa del concepto visto en el *blending*: en lugar de fusionar píxeles, aquí fusionamos distribuciones tonales.

```
%%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 <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");
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 Fundamentos Espaciales: Vecindad, Convolución y *Kernels*

Las operaciones de filtrado espacial no actúan sobre un píxel aislado, sino sobre una **vecindad** que lo rodea. Para ello, se utiliza una pequeña matriz de coeficientes denominada **kernel** (o máscara), que recorre toda la imagen mediante una **ventana deslizante** (*sliding window*).

Las ventanas más comunes son de 3×3, 5×5 y 7×7. En una ventana de 3×3, por ejemplo, el píxel central se procesa junto con sus ocho vecinos inmediatos. En cada posición de la ventana, los valores de los píxeles se combinan con los coeficientes del *kernel*, produciendo un nuevo valor para el píxel central.

3.4.1 Vecindad

Considere una ventana de 3×3 centrada en el píxel (x, y):

$$\begin{bmatrix} (x-1, y-1) & (x, y-1) & (x+1, y-1) \\ (x-1, y) & (x, y) & (x+1, y) \\ (x-1, y+1) & (x, y+1) & (x+1, y+1) \end{bmatrix} \tag{3.9}$$

De forma general, una ventana de tamaño $(2a+1) \times (2b+1)$ abarca todos los píxeles situados hasta a posiciones en horizontal y hasta b posiciones en vertical con respecto al píxel central. Así, una ventana de 3×3 corresponde a $a = b = 1$, una ventana de 5×5 a $a = b = 2$, y así sucesivamente.

Matemáticamente, la vecindad se define como

$$\mathcal{V}(x, y) = \{(x+s,\, y+t) : -a \leq s \leq a,\, -b \leq t \leq b\} \tag{3.10}$$

3.4.2 Tratamiento de Bordes

Los píxeles cercanos a los bordes tienen parte de su vecindad fuera de la imagen. Para aplicar filtros en estas regiones, es necesario definir cómo se obtendrán los valores externos. Las tres estrategias más comunes (con la constante equivalente de OpenCV entre paréntesis) son:

- **Zero-padding** (`BORDER_CONSTANT`): completa la región externa con ceros.
- **Replicación** (`BORDER_REPLICATE`): repite el valor del píxel del borde.
- **Reflexión** (`BORDER_REFLECT_101`): refleja los píxeles vecinos, sin repetir el del borde.

`mm::conv` usa la reflexión por defecto, ya que preserva mejor la continuidad de los niveles de gris y reduce artefactos en el tratamiento de los bordes.

El siguiente ejemplo compara las tres estrategias con `mm::pad` en una matriz 3×3. Observa cómo cada una rellena los píxeles externos necesarios para aplicar un filtro 3×3 también en las esquinas.

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

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

int main() {
    // Construir la imagen 3x3
    mm::Image img(3, 3);
    int valores[3][3] = {{1,2,3},{4,5,6},{7,8,9}};
    for (int y = 0; y < 3; y++)
        for (int x = 0; x < 3; x++)
            img.at(y, x) = valores[y][x];

    std::vector<std::string> bordas = {"constant", "replicate", "reflect101"};

    std::cout << "Imagem original:\n";
    std::cout << mm::drawImg(img) << "\n";

    for (int k : {3, 5}) {
        int b = k / 2;
        std::cout << "=== Kernel " << k << "x" << k << " (padding b=" << b << ") ===\n";
        for (const auto& nome : bordas) {
            std::cout << nome << "\n";
            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)) << "\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
```

```
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
```

Note que el resultado de un filtro puede variar significativamente según el tratamiento adoptado para los bordes de la imagen.

En `morph.hpp`, las funciones de filtrado (`mm::conv`, `mm::blur`, `mm::gaussian`, `mm::laplacian`, `mm::usm`) aplican *padding* por **reflexión** (`mm::Border::REFLECT101`) por defecto — el mismo comportamiento que `cv2.filter2D`. La variante didáctica `mm::conv0` usa `mm::Border::KEEP`: los píxeles del borde mantienen el valor original, sin el filtro. En cambio, `mm::sobel` y `mm::prewitt` dejan el borde en cero (calculan solo el interior).

3.4.3 Correlación vs. Convolución

Existen dos mecanismos matemáticamente relacionados.

Correlación cruzada (*cross-correlation*) — el *kernel* se aplica directamente:

$$g(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t)\, f(x+s,\, y+t) \tag{3.11}$$

Convolución bidimensional — el *kernel* se rota 180° antes de su aplicación:

$$g(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t)\, f(x-s,\, y-t) \tag{3.12}$$

Para *kernels* simétricos (Gaussiano, Laplaciano, media) las dos operaciones producen resultados idénticos. Para *kernels* asimétricos (Sobel, Prewitt) la diferencia es significativa, como lo muestran los ejemplos a continuación.

3.4.3.1 Correlación (`mm::conv`)

```
%%writefile tmp/mm_out_2.cpp
// Converte código Python para C++ (usando morph.hpp)
// Compile com: g++ -std=c++17 programa.cpp -o programa $(pkg-config --cflags --libs opencv4)

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

int main() {
    // imagem 4x4 (uint8) e kernel assimétrico 3x3
    mm::Image img(4, 4);
    // Preenche com os valores da matriz
    {
        int valores[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) = static_cast<unsigned char>(valores[y][x]);
            }
        }
    }

    mm::Kernel w{{0,1,2},{0,0,0},{0,0,0}};

    mm::Image corr = mm::conv(img, w, mm::Border::CONSTANT);  // zero fora da imagem

    std::cout << "Imagem original:\n";         std::cout << mm::drawImg(img);
    std::cout << "Kernel:\n";                  std::cout << mm::drawImg(w);
    std::cout << "Resultado da correlação:\n"; std::cout << mm::drawImg(corr);

    return 0;
}
```

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

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

```
Imagem original:
 1  2  3  4
 5  6  7  8
 9 10 11 12
13 14 15 16
Kernel:
   0    1    2
   0    0    0
   0    0    0
Resultado da correlação:
 0  0  0  0
```

```
 5  8 11  4
17 20 23  8
29 32 35 12
```

`mm::conv` realiza correlación, es decir, aplica el *kernel* exactamente en la orientación proporcionada.

3.4.3.2 Convolución

```
%%writefile tmp/mm_out_3.cpp
// Compile: g++ -std=c++17 -o program program.cpp -I. && ./program
#include "morph.hpp"
#include <iostream>

int main() {
    // repetidos aqui para a célula ser independente
    mm::Image img(4, 4);
    img.at(0,0) = 1;   img.at(0,1) = 2;   img.at(0,2) = 3;   img.at(0,3) = 4;
    img.at(1,0) = 5;   img.at(1,1) = 6;   img.at(1,2) = 7;   img.at(1,3) = 8;
    img.at(2,0) = 9;   img.at(2,1) = 10;  img.at(2,2) = 11;  img.at(2,3) = 12;
    img.at(3,0) = 13;  img.at(3,1) = 14;  img.at(3,2) = 15;  img.at(3,3) = 16;

    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::cout << mm::drawImg(img) << "\n";
    std::cout << "Kernel original:";            std::cout << mm::drawImg(w) << "\n";
    std::cout << "Kernel rotacionado 180°:";    std::cout << mm::drawImg(w_conv) << "\n";
    std::cout << "Resultado da convolução:";    std::cout << mm::drawImg(conv) << "\n";

    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 convolución utiliza el *kernel* rotado 180°. Para reproducir la definición matemática de convolución, se rota

el *kernel* (aquí, `[[0,1,2],[0,0,0],[0,0,0]]` → `[[0,0,0],[0,0,0],[2,1,0]]`) antes de aplicar `mm::conv`.

3.4.4 El Papel del *Kernel*

Los coeficientes del *kernel* determinan completamente el efecto producido por el filtro, tal como se resume en la Tabla 3.2.

Tabla 3.2: Interpretación típica de los coeficientes del *kernel*.

Característica	Efecto típico
Coeficientes positivos con suma 1	Suavizado (*pasa-baja*)
Suma igual a 0, con valores positivos y negativos	Detección de bordes (*pasa-alta*)
Coeficiente central positivo dominante y vecinos negativos	Realce de nitidez
Coeficientes asimétricos	Gradiente direccional

Ejemplos:

Suavizado:

$$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$$

Detección de bordes:

$$\begin{bmatrix}-1 & -1 & -1\\ -1 & 8 & -1\\ -1 & -1 & -1\end{bmatrix}$$

Realce de nitidez:

$$\begin{bmatrix}0 & -1 & 0\\ -1 & 5 & -1\\ 0 & -1 & 0\end{bmatrix}$$

Gradiente direccional (Sobel):

$$\begin{bmatrix}-1 & 0 & 1\\ -2 & 0 & 2\\ -1 & 0 & 1\end{bmatrix}$$

La Figura 3.10 demuestra el mecanismo paso a paso: para cada posición de la ventana, se multiplica cada coeficiente del *kernel* por el píxel correspondiente de la vecindad y se suman los productos obtenidos. El resultado es exactamente el valor definido por la Ecuación 3.11 para esa posición de la imagen. Aunque las imágenes producidas por `mm::conv0` y `cv2.filter2D` (o `mm::conv`) sean visualmente muy similares, la implementación basada en OpenCV es miles de veces más rápida, como se muestra a continuación.

⚠ Rendimiento: bucles de Python vs. operaciones vectorizadas

La función `mm::conv0` implementa la correlación directamente en Python mediante bucles anidados. Aunque este enfoque es adecuado para fines didácticos, ejecuta un gran número de operaciones y se vuelve lento para imágenes más grandes.
En cambio, `mm::conv` utiliza `cv2.filter2D`, implementado en C++ y optimizado para operaciones matriciales. En el ejemplo presentado, la versión vectorizada fue más de **3000 veces más rápida** que la implementación didáctica, produciendo un resultado visualmente equivalente.
Las diferencias numéricas observadas se concentran principalmente en los bordes de la imagen. En `mm::conv0`, los píxeles del borde permanecen inalterados, mientras que `mm::conv` utiliza una estrategia de reflexión de bordes (`cv2.BORDER_REFLECT_101`, estándar de `cv2.filter2D`).

Por ello, `mm::conv0` debe utilizarse para comprender el algoritmo, mientras que `mm::conv` es la opción recomendada para aplicaciones prácticas.

```
%%writefile tmp/fig_03_convolucao_passo.cpp
#define MM_OUT "tmp/fig_03_convolucao_passo.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_leop_gray = mm::_read_state("tmp/state/img_leop_gray_12.png");
// [pdi:state-io:end]

    //| 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

    mm::Kernel w_mean = mm::Kernel::mean(3);
    mm::Image img_gray = img_leop_gray;

    mm::Image img_conv0 = mm::conv0(img_gray, w_mean);   // laços, bordas preservadas
    mm::Image img_conv  = mm::conv(img_gray, w_mean);    // borda refletida

    std::cout << "Correlacao no pixel central [251,251]:" << "\n";
    std::cout << "  original = " << (int)img_gray.at(251, 251) << "\n";
    std::cout << "  conv0    = " << (int)img_conv0.at(251, 251) << "\n";
    std::cout << "  conv     = " << (int)img_conv.at(251, 251) << "\n";

    mm::show(std::vector<mm::Image>{img_gray, img_conv0, img_conv},
            MM_OUT,
            std::vector<std::string>{"Original", "conv0 (laços)", "conv (vetorizado)"}, 3);


// [pdi:panel-io] auto-generated - do not edit by hand
std::filesystem::create_directories("tmp");
mm::write(img_gray, "tmp/fig_03_convolucao_passo_0.png");
mm::write(img_conv0, "tmp/fig_03_convolucao_passo_1.png");
mm::write(img_conv, "tmp/fig_03_convolucao_passo_2.png");
// [pdi:panel-io:end]
return 0;
}
```

```
Overwriting tmp/fig_03_convolucao_passo.cpp
```

```
!g++ -I. -std=c++17 tmp/fig_03_convolucao_passo.cpp -o tmp/fig_03_convolucao_passo \
  && ./tmp/fig_03_convolucao_passo \
  && test -f "tmp/fig_03_convolucao_passo.png" \
  || echo "  mm::show não gravou tmp/fig_03_convolucao_passo.png"
```

```
Correlacao no pixel central [251,251]:
  original = 104
  conv0    = 102
  conv     = 102
[1] Original
[2] conv0 (laços)
```

```
[3] conv (vetorizado)
```

```
try:
    mm.show(
        [
            mm.read("tmp/fig_03_convolucao_passo_0.png"),
            mm.read("tmp/fig_03_convolucao_passo_1.png"),
            mm.read("tmp/fig_03_convolucao_passo_2.png"),
        ],
        titles=[
            'Original',
            'conv0 (laços)',
            'conv (vetorizado)',
        ],
        cols=3,
    )
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + "
    tmp/fig_03_convolucao_passo_0.png (ver a versao Python)")
```

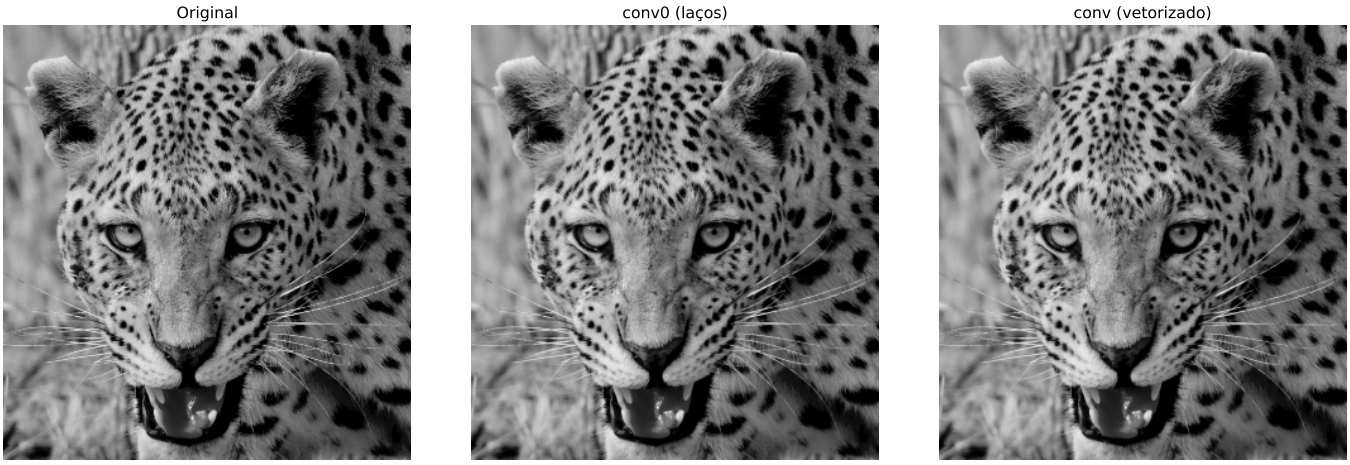


Figura 3.10: Correlação com *kernel* de média 3×3: versão didática mm::conv0 (bordas preservadas) vs. mm::conv (borda refletida). A diferença se concentra nas bordas.

```
%%writefile tmp/mm_out_4.cpp
#include "morph.hpp"
#include <iostream>
#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]

    //| echo: false
    // A partir daqui o "sujeito" dos exemplos de filtragem passa a ser o
    // leopardo (mais textura e bordas que o mandril).
    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 Ejemplo Numérico: Correlación Paso a Paso

Para hacer concreto el mecanismo de la Ecuación 3.11, considere el *kernel* de media 3×3 ($a = b = 1$, todos los coeficientes $= 1/9 \approx 0{,}111$) aplicado al *patch* 5×5 extraído de la imagen del leopardo. La Figura 3.11 muestra el *patch* con la cuadrícula y resalta en amarillo la ventana 3×3 centrada en el píxel $[1, 1]$:

```
%%writefile tmp/fig_03_patch.cpp
#define MM_OUT "tmp/fig_03_patch.png"
#include "morph.hpp"
#include <iostream>

int main() {
// [pdi:state-io] auto-generated - do not edit by hand
mm::Image img_gray = mm::_read_state("tmp/state/img_gray_45.png");
// [pdi:state-io:end]

    //| label: fig-03-patch
    //| fig-cap: "*Patch* 5×5 extraído da imagem do leopardo (posição [250:255, 250:255]). A
    janela amarela destaca a vizinhança 3×3 centrada no pixel [1,1] onde a correlação será
    calculada."
    //| echo: true
    //| output: true

    mm::Image patch = mm::crop(img_gray, 250, 255, 250, 255);
    mm::Kernel B = mm::Kernel::ones(3);

    std::cout << "Patch 5×5 (intensidades):" << std::endl;
    std::cout << mm::drawImg(patch);

    mm::drawImgKernel(patch, B, 1, 1, MM_OUT, 40);

    return 0;
}
```

```
Overwriting tmp/fig_03_patch.cpp
```

```
!g++ -I. -std=c++17 tmp/fig_03_patch.cpp -o tmp/fig_03_patch \
  && ./tmp/fig_03_patch \
  && test -f "tmp/fig_03_patch.png" \
  || echo "  mm::show não gravou tmp/fig_03_patch.png"
```

```
Patch 5×5 (intensidades):
 94  96 103 113 115
107 104 103 107 114
 98 102 112 117 116
 81  91 112 122 118
 85  91 110 119 121
Processando pixel (x,y)=(1,1)  |  janela do kernel 3x3
 94  96 103 113 115
107 104 103 107 114
```

```
 98 102 112 117 116
 81  91 112 122 118
 85  91 110 119 121
```

```
try:
    mm.show(mm.read("tmp/fig_03_patch.png"))
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + " tmp/fig_03_patch.png (ver
    a versao Python)")
```

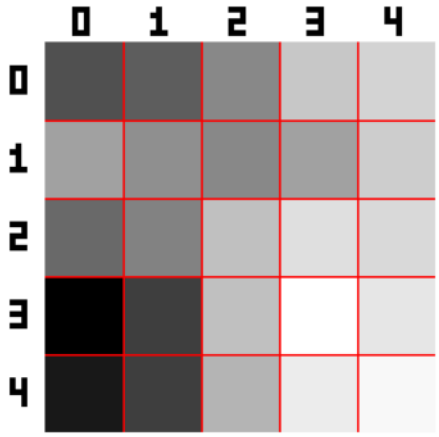


Figura 3.11: *Patch* 5×5 extraído da imagem do leopardo (posição [250:255, 250:255]). A janela amarela destaca a vizinhança 3×3 centrada no pixel [1,1] onde a correlação será calculada.

Para ilustrar el cálculo de la correlación, considere el píxel en la posición $[1, 1]$ del *patch* 5×5 mostrado en la Figura 3.11.. Esa posición fue elegida solo por conveniencia didáctica, ya que posee una vecindad 3×3 completa a su alrededor.

Los valores de esa vecindad corresponden a la submatriz superior izquierda del *patch*:

$$\text{vecindad} = \begin{bmatrix} 91 & 95 & 108 \\ 106 & 107 & 108 \\ 102 & 103 & 107 \end{bmatrix}$$

Aplicando la Ecuación 3.11 con el kernel de media:

$$g[1,1] = \frac{91 + 95 + 108 + 106 + 107 + 108 + 102 + 103 + 107}{9} = \frac{927}{9} = 103$$

El resultado (103) es ligeramente menor que el valor original del píxel central (107), pues la media incorpora vecinos de menor intensidad, produciendo el efecto de suavizado. En la práctica, el algoritmo inicia el procesamiento en $[0, 0]$ y repite ese mismo cálculo para cada posición de la imagen, desplazando la ventana hasta cubrir todo el dominio.

3.5 Filtrado Espacial de Suavizado

Los filtros de suavizado (*smoothing filters*) atenúan variaciones bruscas de intensidad, reduciendo ruido y detalles de alta frecuencia. Son filtros **paso-bajo** — preservan las componentes de baja frecuencia (estructuras grandes) y atenúan las de alta frecuencia (ruido, bordes).

3.5.1 Filtro de Media (*Box Filter*)

El filtro de media utiliza un *kernel* uniforme de tamaño $n \times n$, donde todos los coeficientes valen $1/n^2$:

$$w_{\text{media}} = \frac{1}{n^2} \begin{bmatrix} 1 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 1 \end{bmatrix}_{n \times n} \tag{3.13}$$

Cada píxel de salida es la media aritmética de los n^2 píxeles de su vecindario. Nótese que la suma de los coeficientes es siempre 1 — el brillo medio de la imagen se preserva. Los *kernels* más grandes producen un suavizado más agresivo, pero desenfocan progresivamente los bordes.

Figura 3.12 muestra el efecto del filtro de media con *kernels* 3×3, 7×7 y 15×15 sobre un detalle de la imagen del leopardo. Los resultados se obtuvieron con `mm::blur`, que implementa el filtro de media mediante la función `cv2.blur`, equivalente a la convolución de la imagen con un *kernel* uniforme cuyos coeficientes son $h(x, y) = 1/N^2$; de forma equivalente, el mismo resultado puede obtenerse con `mm::conv`, calculando $(g = f * h)$. A medida que el *kernel* aumenta, más píxeles contribuyen a cada valor de salida, intensificando el suavizado, reduciendo el ruido y haciendo que los detalles finos y los bordes se vuelvan progresivamente más borrosos.

```
%%writefile tmp/fig_03_media.cpp
#define MM_OUT "tmp/fig_03_media.png"
#include "morph.hpp"
#include <iostream>
#include <vector>
#include <string>

int main() {
// [pdi:state-io] auto-generated - do not edit by hand
mm::Image img_gray = mm::_read_state("tmp/state/img_gray_45.png");
// [pdi:state-io:end]

    // 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)); // ou
        // mm::Kernel kernel = mm::Kernel::mean(k);
        // imgs.push_back(mm::conv(img_gray_crop, kernel));
    }
    std::vector<std::string> titles = {"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

El filtro Gaussiano pondera los píxeles de la vecindad de acuerdo con una función Gaussiana bidimensional:

$$G(s,t) = \frac{1}{2\pi\sigma^2}\, e^{-\frac{s^2+t^2}{2\sigma^2}} \tag{3.14}$$

donde σ es la desviación estándar y controla el radio de influencia. Los píxeles más cercanos al centro tienen un peso mayor; los píxeles distantes se ignoran progresivamente.

La Figura 3.13 presenta el *kernel* Gaussiano 5×5 generado para $\sigma = 1$. El *kernel* fue construido a partir del producto externo de dos vectores Gaussianos unidimensionales y posteriormente normalizado para que la suma de sus coeficientes sea igual a 1. Se observa que los mayores pesos se concentran en el centro de la matriz, decreciendo radialmente hacia los bordes. Esta distribución hace que los píxeles centrales tengan mayor influencia en el resultado del filtrado, contribuyendo a una suavización más natural y con mejor preservación de bordes que el filtro de media.

```
%%writefile tmp/fig_03_gauss_kernel.cpp
#define MM_OUT "tmp/fig_03_gauss_kernel.png"
#include "morph.hpp"
#include <iostream>
#include <vector>
#include <string>
#include <numeric>
#include <iomanip>

int main() {
    //| label: fig-03-gauss-kernel
    //| fig-cap: "*Kernel* Gaussiano 5×5 (σ=1): resposta ao impulso do filtro - pesos maiores
    no centro, decrescendo radialmente."
```

```
    //| echo: true
    //| output: true

    mm::Kernel w = mm::Kernel::gaussian(5, 1.0);   // mm::Kernel::gaussian(5, 1.0) na
morph.hpp

    std::cout << "Kernel Gaussiano 5x5 (s=1), normalizado:" << "\n";
    for (int y = 0; y < 5; y++) {
        std::cout << "  ";
        for (int x = 0; x < 5; x++) {
            std::cout << std::fixed << std::setprecision(4) << w.at(y, x);
            if (x < 4) std::cout << "  ";
        }
        std::cout << "\n";
    }
    std::cout << "Peso central [2,2] = " << std::fixed << std::setprecision(4) << w.at(2, 2)
<< "   |   canto [0,0] = " << 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);

    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)")
```

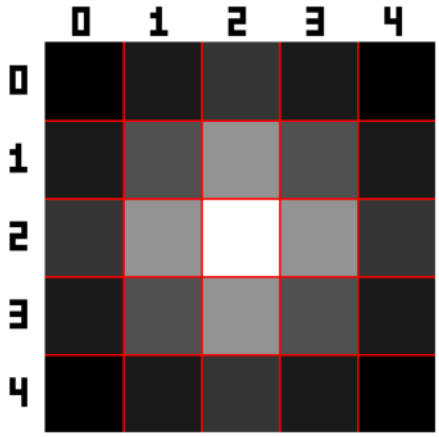


Figura 3.13: *Kernel* Gaussiano 5×5 (=1): resposta ao impulso do filtro — pesos maiores no centro, decrescendo radialmente.

i Ventaja computacional de la separabilidad

Considere un *kernel* cuadrado de tamaño $n \times n$. Si este filtro es separable (como el Gaussiano), la convolución 2D puede descomponerse en dos convoluciones 1D: una horizontal y otra vertical.
En ese caso, el costo por píxel pasa de aproximadamente $O(n^2)$ operaciones (convolución 2D directa) a $O(2n)$ operaciones (dos convoluciones 1D). Así, la complejidad se reduce de forma significativa, haciendo el procesamiento más eficiente.

En comparación con el filtro de media, el Gaussiano:

- **Preserva mejor los bordes** — la ponderación radial suaviza sin crear transiciones abruptas;
- **No introduce anillos** (*ringing*) en el dominio de la frecuencia, pues la Gaussiana es su propia transformada de Fourier (Capítulo 5);
- **Está controlado por** σ — aumentar σ equivale a aumentar el radio de suavizado de forma continua y predecible.

La Figura 3.14 compara los filtros de media y gaussiano aplicados a la imagen del leopardo usando una ventana 9×9. El filtro de media se implementó mediante convolución con un *kernel* uniforme, donde todos los 81 píxeles de la vecindad poseen el mismo peso (1/81), mientras que el filtro gaussiano se obtuvo con `cv2.GaussianBlur`, utilizando pesos definidos por una distribución gaussiana. Ambos reducen ruido y suavizan la imagen, pero el filtro gaussiano preserva mejor los bordes y los detalles locales, como puede observarse en la región ampliada del ojo.

La Figura 3.14 compara los filtros de media y gaussiano aplicados a un detalle de la imagen del leopardo con *kernels* 9×9. El filtro de media se obtuvo con `mm::blur`, equivalente a la convolución con un *kernel* uniforme cuyos coeficientes valen 1/81, mientras que el filtro gaussiano se obtuvo con `mm::gaussian`, equivalente a la convolución con un *kernel* generado a partir de una distribución gaussiana. Ambos promueven suavización y reducción de ruido, pero el filtro gaussiano asigna mayor peso a los píxeles centrales de la vecindad, preservando mejor los bordes y los detalles locales, como puede observarse en la región ampliada del ojo.

```
%%writefile tmp/fig_03_gauss.cpp
#define MM_OUT "tmp/fig_03_gauss.png"
// Compile: g++ -std=c++17 -O2 -o prog prog.cpp -lm && ./prog
#include "morph.hpp"
#include <iostream>
#include <vector>
#include <string>
#include <filesystem>
```

```
//| 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

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 é fornecido automaticamente (não declarar)

    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 Filtrado Espacial de Realce

Los filtros de realce (*sharpening filters*) enfatizan transiciones abruptas de intensidad, aumentando la nitidez y la visibilidad de bordes. Son filtros **paso-alto** — amplifican los componentes de alta frecuencia (bordes, textura) y suprimen los de baja frecuencia (regiones uniformes).

La intuición es simple: si restamos de una imagen su versión suavizada (que contiene solo las bajas frecuencias), lo que queda son las altas frecuencias — bordes y detalles. Sumando ese residuo de vuelta a la imagen original, el contraste local aumenta:

$$g = f + k\,(f - f_{\text{suave}}), \quad k > 0 \tag{3.15}$$

La Figura 3.15 ilustra este proceso en una señal 1D sintética con tres estructuras distintas: un escalón ancho, un pico fino y una rampa suave. En el panel , la señal original $f(x)$; en el , la versión suavizada $f_{\text{suave}}(x)$ obtenida por media móvil — nótese cómo el pico fino se atenúa. El panel muestra el residuo $f - f_{\text{suave}}$, que retiene solo las transiciones abruptas. Finalmente, el panel muestra $g(x)$: el pico, antes atenuado, se restaura y amplifica en relación con el original. Ajuste k y el tamaño de la ventana para observar el *trade-off* entre nitidez y amplificación de ruido.

Los filtros de realce formalizan esta idea directamente en el *kernel*, sin necesidad de dos etapas separadas.

3.6.1 Laplaciano

El Laplaciano es un operador de **segunda derivada** isotrópico, es decir, responde de igual manera a las variaciones en todas las direcciones, a diferencia de los operadores de primera derivada, como Sobel y Prewitt, que son direccionales:

Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.es/cap03/sim-03-filtragem1d.html

Figura 3.15: Simulador: Filtrado Espacial de Realce 1D (Unsharp Masking y High-Boost)

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \tag{3.16}$$

Una propiedad importante de la segunda derivada es que su valor es **cercano a cero en regiones uniformes** y elevado en las transiciones de intensidad. Así, al restar el Laplaciano de la imagen original, se refuerzan los bordes y detalles, aumentando el contraste local:

$$g(x, y) = f(x, y) - \nabla^2 f(x, y) \tag{3.17}$$

En la forma discreta, la segunda derivada en x se aproxima por $f(x+1, y) - 2f(x, y) + f(x-1, y)$, y de manera análoga en y. Sumando ambas direcciones, se obtiene el *kernel* w_4 (4-vecinos) o w_8 (8-vecinos, incluyendo diagonales):

$$w_4 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \qquad w_8 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{3.18}$$

i Suma cero y centro negativo

Ambos *kernels* tienen **suma de coeficientes igual a cero**: en regiones uniformes, la salida es 0 — el Laplaciano no altera el brillo medio, solo detecta variaciones. El **centro negativo** indica que el píxel se compara con sus vecinos: cuanto más se destaque (hacia arriba o hacia abajo), mayor será el valor absoluto del Laplaciano en ese punto.

En el siguiente ejemplo, el píxel central $[1, 1] = 107$ posee vecinos $\{95, 106, 108, 103\}$. Como estos valores son cercanos entre sí, la región es casi uniforme y el Laplaciano devuelve un valor bajo, produciendo poco realce. En regiones de borde, donde hay diferencias mayores entre el píxel central y sus vecinos, el Laplaciano asume valores más elevados (positivos o negativos), y la operación de Ecuación 3.17 intensifica esas transiciones.

La Figura 3.16 ilustra el cálculo del Laplaciano con el *kernel* w_4 en una vecindad 3×3 destacada dentro de un *patch* 5×5. El ejemplo muestra el valor obtenido por el operador y el correspondiente píxel realzado en la imagen de salida, que pasa de 107 a 123.

```
%%writefile tmp/fig_03_laplaciano_patch.cpp
#define MM_OUT "tmp/fig_03_laplaciano_patch.png"
// Compile with: g++ -std=c++17 -o program program.cpp -lmorph
#include "morph.hpp"
#include <iostream>

//| label: fig-03-laplaciano-patch
//| fig-cap: "*Kernel* Laplaciano w4 sobre el *patch* 5×5: la ventana amarilla destaca la
vecindad 3×3 donde el operador de segunda derivada se calcula."
//| echo: true
//| output: true

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 5x5 (intensidades):\n";
    std::cout << mm::drawImg(patch) << "\n";
```

```
    mm::drawImgKernel(patch, B, 1, 1, MM_OUT, 40);

    return 0;
}
```

```
Overwriting tmp/fig_03_laplaciano_patch.cpp
```

```
!g++ -I. -std=c++17 tmp/fig_03_laplaciano_patch.cpp -o tmp/fig_03_laplaciano_patch \
  && ./tmp/fig_03_laplaciano_patch \
  && test -f "tmp/fig_03_laplaciano_patch.png" \
  || echo "  mm::show não gravou tmp/fig_03_laplaciano_patch.png"
```

```
Patch 5x5 (intensidades):
 94  96 103 113 115
107 104 103 107 114
 98 102 112 117 116
 81  91 112 122 118
 85  91 110 119 121

Processando pixel (x,y)=(1,1)  |  janela do kernel 3x3
 94  96 103 113 115
107 104 103 107 114
 98 102 112 117 116
 81  91 112 122 118
 85  91 110 119 121
```

```
try:
    mm.show(mm.read("tmp/fig_03_laplaciano_patch.png"))
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + "
    tmp/fig_03_laplaciano_patch.png (ver a versao Python)")
```

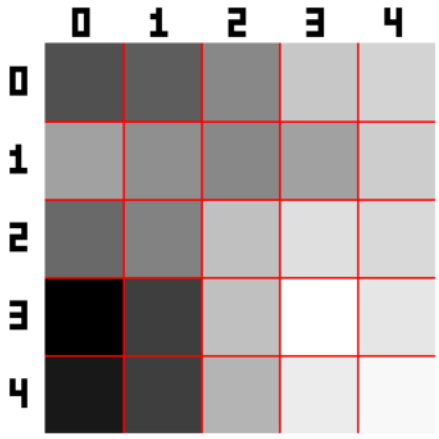


Figura 3.16: *Kernel* Laplaciano w4 sobre o *patch* 5×5: a janela amarela destaca a vizinhança 3×3 onde o operador de segunda derivada é calculado.

A Figura 3.17 compara a aplicação de los *kernels* laplacianos w_4 y w_8 en un recorte más grande de la imagen del leopardo. Para cada caso, se muestran la respuesta bruta del operador, que evidencia los bordes y las transiciones de intensidad, y la imagen obtenida tras el realce por sustracción del laplaciano. Se observa que el *kernel* w_8, al considerar también los vecinos diagonales, produce una respuesta más intensa y detecta variaciones en más direcciones, resultando en un realce ligeramente más acentuado.

```
%%writefile tmp/fig_03_laplaciano.cpp
#define MM_OUT "tmp/fig_03_laplaciano.png"
//| label: fig-03-laplaciano
//| fig-cap: "Laplaciano aplicado à imagem do leopardo: resposta bruta (bordas) com w4 e w8, e
imagens realçadas pela subtração do Laplaciano. w8 é mais sensível às diagonais."
//| echo: true
//| output: true

#include "morph.hpp"
#include <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 automatically

    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(
        {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,
        {"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)")
```

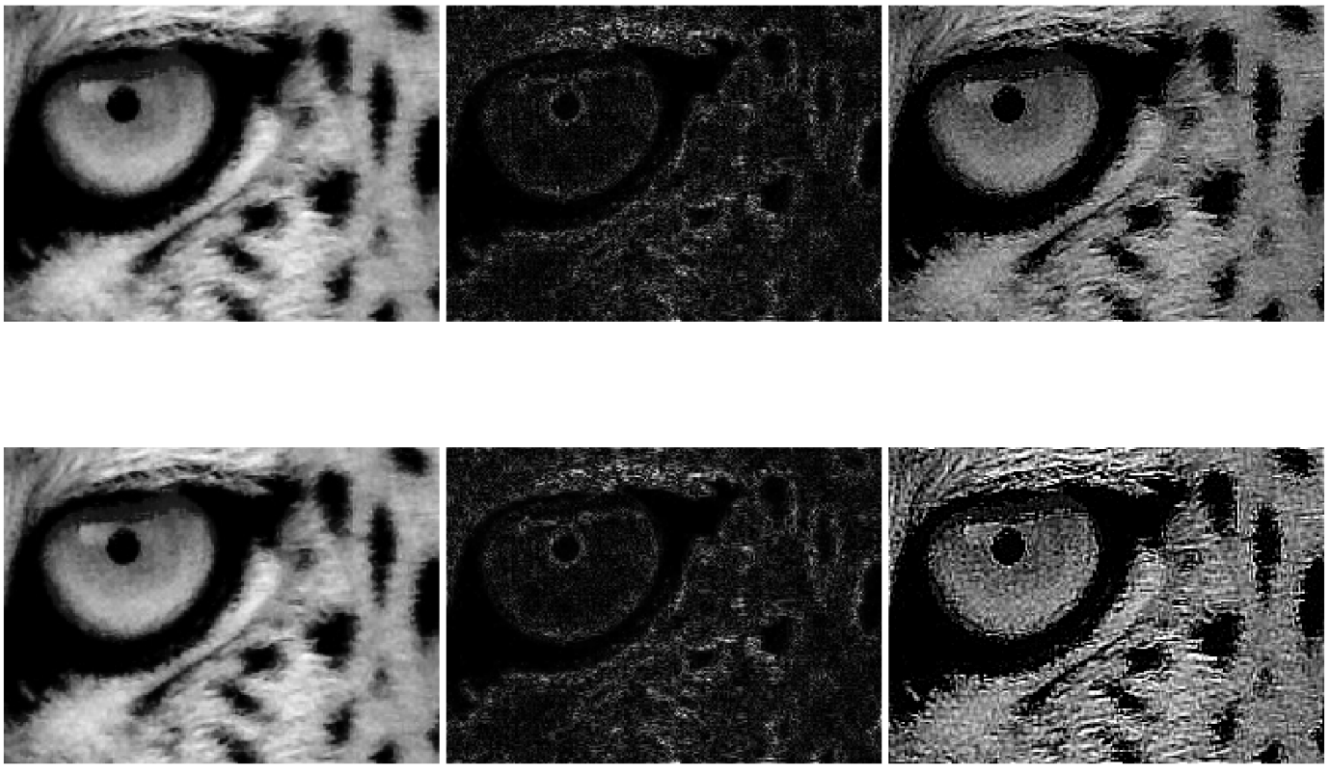


Figura 3.17: Laplaciano aplicado à imagem do leopardo: resposta bruta (bordas) com w4 e w8, e imagens realçadas pela subtração do Laplaciano. w8 é mais sensível às diagonais.

3.6.2 Operador de Sobel

El operador de Sobel estima las **derivadas parciales de primer orden** en las direcciones horizontal y vertical. A diferencia del Laplaciano (segunda derivada), el Sobel es direccional y más robusto al ruido, pues cada *kernel* combina una derivada con un suavizado Gaussiano perpendicular:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * f, \qquad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} * f \tag{3.19}$$

G_x detecta bordes **verticales** (variación en la dirección x); G_y detecta bordes **horizontales** (variación en la dirección y). Los pesos $\{1, 2, 1\}$ en la dirección perpendicular corresponden al suavizado Gaussiano 1D, que reduce la sensibilidad al ruido.

> **i** Sobel es correlación, no convolución
>
> Los *kernels* de Sobel son **asimétricos** — la rotación de 180° altera el resultado. `cv2.Sobel` implementa correlación cruzada (como `cv2.filter2D`). Para obtener la derivada direccional correcta, las señales ya están definidas para correlación: G_x devuelve valores positivos donde la intensidad crece de izquierda a derecha.

La magnitud del **gradiente** combina los dos componentes, representando la fuerza del borde independientemente de la dirección:

$$|\nabla f| = \sqrt{G_x^2 + G_y^2} \tag{3.20}$$

Y la **dirección** del gradiente (perpendicular al borde) es:

$$\theta = \arctan\left(\frac{G_y}{G_x}\right) \tag{3.21}$$

Para ilustrar numéricamente, se calculan G_x y G_y manualmente en el píxel central $[1, 1]$ del *patch* de 5×5:

El valor reducido de $|\nabla f|$ en ese *patch* confirma que la región es casi uniforme, pues el gradiente asume valores elevados solo donde hay cambios significativos de intensidad. La Figura 3.18 aplica el operador de Sobel a un recorte mayor de la imagen del leopardo. Se presentan las respuestas horizontal (G_x) y vertical (G_y), obtenidas por convolución con los respectivos *kernels* de Sobel, además de la magnitud $|\nabla f|$, calculada a partir de la combinación de ambas. Mientras que G_x destaca bordes verticales y G_y bordes horizontales, la magnitud evidencia bordes en cualquier dirección.

```
%%writefile tmp/fig_03_sobel.cpp
#define MM_OUT "tmp/fig_03_sobel.png"
#include "morph.hpp"
#include <iostream>
#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-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

    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)")
```

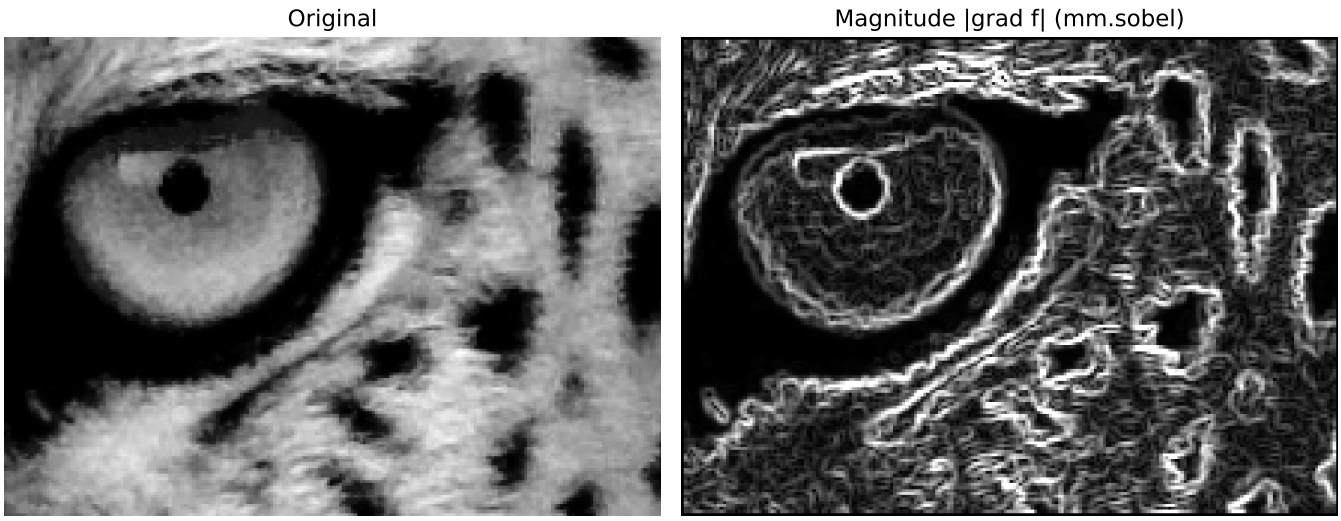


Figura 3.18: Operador de Sobel na imagem do leopardo: a magnitude | f| combina os gradientes horizontal e vertical, revelando todas as bordas. A decomposição Gx/Gy com sinal fica na trilha Python — mm::sobel devolve a magnitude já com clip.

3.6.3 Operador de Prewitt

El operador de Prewitt es estructuralmente idéntico al de Sobel, pero sustituye la ponderación gaussiana $\{1, 2, 1\}$ por pesos uniformes $\{1, 1, 1\}$:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} * f, \qquad G_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} * f \tag{3.22}$$

La magnitud y la dirección del gradiente siguen las mismas ecuaciones que las de Sobel (Ecuación 3.20 y Ecuación 3.21). La diferencia práctica es que Prewitt es ligeramente más sensible al ruido — el suavizado perpendicular uniforme pondera menos el píxel central de la línea — pero computacionalmente más simple. En imágenes con bajo ruido los resultados son equivalentes.

```
%%writefile tmp/fig_03_prewitt.cpp
#define MM_OUT "tmp/fig_03_prewitt.png"
// Compile with: g++ -std=c++17 -o program program.cpp morph.hpp
```

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

//| 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

int main() {
// [pdi:state-io] auto-generated - do not edit by hand
mm::Image img_gray_crop = mm::_read_state("tmp/state/img_gray_crop_58.png");
// [pdi:state-io:end]

    mm::show(std::vector<mm::Image>{img_gray_crop, mm::prewitt(img_gray_crop)},
             MM_OUT,
             std::vector<std::string>{"Original", "Magnitude |grad f| (mm.prewitt)"}, 2);
    return 0;
}
```

```
Overwriting tmp/fig_03_prewitt.cpp
```

```
!g++ -I. -std=c++17 tmp/fig_03_prewitt.cpp -o tmp/fig_03_prewitt \
  && ./tmp/fig_03_prewitt \
  && test -f "tmp/fig_03_prewitt.png" \
  || echo "  mm::show não gravou tmp/fig_03_prewitt.png"
```

```
[1] Original
[2] Magnitude |grad f| (mm.prewitt)
```

```
try:
    mm.show(mm.read("tmp/fig_03_prewitt.png"))
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + " tmp/fig_03_prewitt.png
    (ver a versao Python)")
```

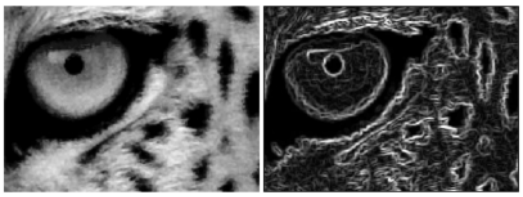


Figura 3.19: Operador de Prewitt: magnitude do gradiente, comparável ao Sobel mas sem a ponderação central. mm::prewitt devolve |f| com clip.

3.6.4 *Unsharp Masking* (USM)

El *Unsharp Masking* es una técnica clásica de realce de nitidez originaria de la fotografía analógica, hoy ampliamente utilizada en software de edición de imágenes. La idea central es extraer los **componentes de alta frecuencia** de la imagen (bordes y detalles) y sumarlos de vuelta a la original con un peso k:

Tabla 3.3: Etapas del *Unsharp Masking*.

Etapa	Operación	Descripción
1	$\bar{f} = f * G_\sigma$	Suaviza con gaussiana — retiene bajas frecuencias
2	$m = f - \bar{f}$	Máscara: diferencia = altas frecuencias (bordes)
3	$g = f + k \cdot m$	Suma ponderada de la máscara a la original

Sustituyendo la etapa 2 en la etapa 3, se obtiene la expresión compacta:

$$g = f + k\,(f - f * G_\sigma) = (1 + k)\,f - k\,(f * G_\sigma) \tag{3.23}$$

El parámetro k controla la intensidad del realce:

- $k = 0$: sin realce ($g = f$);
- $k = 1$: USM clásico — duplica la contribución de las altas frecuencias;
- $k > 1$: *High Boost Filtering* — amplificación más allá del doble, útil para imágenes muy borrosas.

> **Amplificación de ruido**
>
> El USM no distingue bordes de ruido — ambos son componentes de alta frecuencia. Para k elevado, el ruido presente en la imagen se amplifica junto con los bordes. Por ello, se recomienda aplicar una leve suavización antes del USM en imágenes ruidosas, o usar un σ pequeño en la gaussiana.

Para ilustrar las etapas del USM, la Figura 3.20 aplica el método a un *parche* 30×30 de la imagen del leopardo, utilizando $\sigma = 1$ y $k = 1$. Inicialmente, la imagen se suaviza mediante un filtro gaussiano. A continuación, la máscara de alta frecuencia se obtiene como la diferencia entre la imagen original y la suavizada. Finalmente, esta máscara se suma a la imagen original, reforzando bordes y detalles. La figura presenta las tres etapas del proceso y el resultado final del realce.

```
%%writefile tmp/fig_03_usm2.cpp
#define MM_OUT "tmp/fig_03_usm2.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_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)")
```

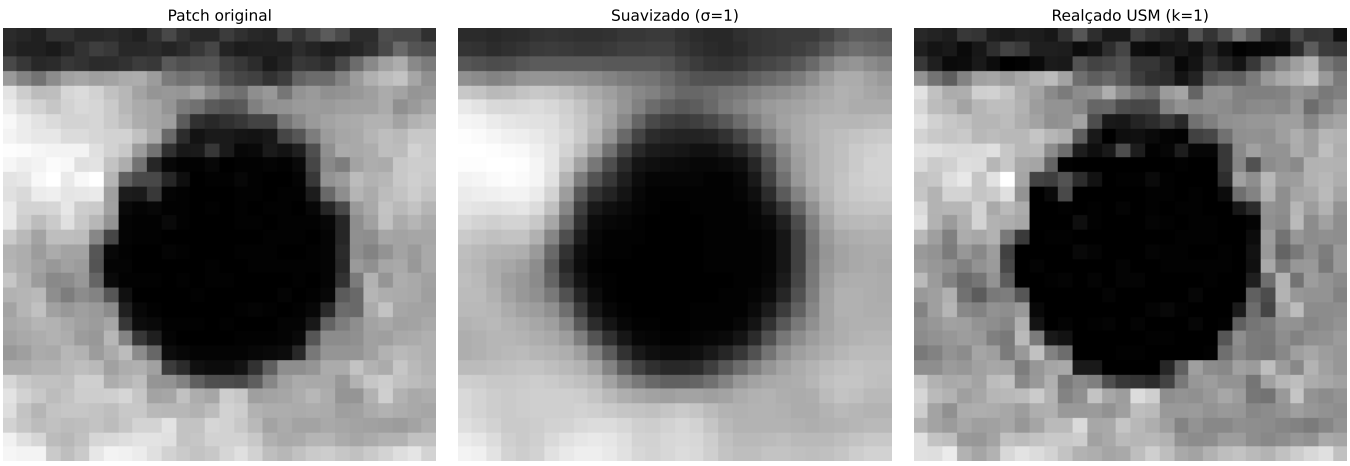


Figura 3.20: Realce por *Unsharp Masking* num *patch* do leopardo: mm::usm faz suavização Gaussiana, subtrai da original (máscara de alta frequência) e reintroduz a máscara realçada.

La Figura 3.21 aplica el método USM a un recorte más grande de la imagen del leopardo utilizando $\sigma = 1$ y diferentes valores del factor de ganancia k. En todos los casos, la máscara de alta frecuencia se obtiene mediante la diferencia entre la imagen original y su versión suavizada por filtro gaussiano. El parámetro k controla la intensidad del realce: valores menores producen un aumento sutil de nitidez, mientras que valores mayores refuerzan progresivamente bordes y detalles. Se observa que, para valores elevados de k, surgen halos alrededor de los bordes y el ruido presente en la imagen comienza a amplificarse.

```
%%writefile tmp/fig_03_usm.cpp
#define MM_OUT "tmp/fig_03_usm.png"
// Compile with: g++ -std=c++17 -o program program.cpp -lmorph
#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]

    //| 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

    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)")
```

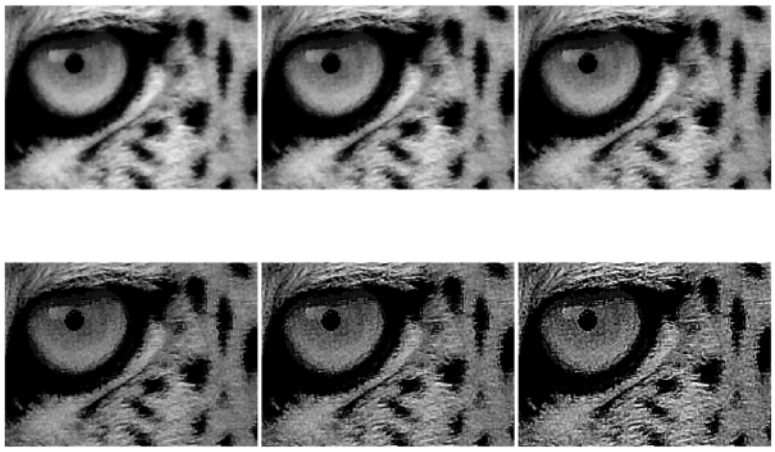


Figura 3.21: *Unsharp Masking* na imagem do leopardo com =1 e k de 0.5 a 8.0. Para k>2 surgem halos nas bordas e o ruído de fundo aparece.

3.6.5 Detector de Canny

Canny combina cuatro etapas en secuencia — suavizado Gaussiano, gradiente de Sobel, supresión de no-máximos e histéresis por doble umbral — para producir bordes **finos, binarios y conectados**. A diferencia de Sobel y Prewitt, el resultado no es un mapa de gradiente continuo, sino una máscara donde cada píxel es borde o no.

El parámetro central es el par de umbrales (T_{low}, T_{high}). Los píxeles con gradiente por encima de T_{high} son bordes seguros; por debajo de T_{low}, se descartan. Los píxeles ambiguos — entre los dos umbrales — se deciden mediante **histéresis**: se convierten en borde si están conectados a un borde seguro, y se descartan en caso contrario. Esto evita tanto la pérdida de tramos débiles de bordes reales como la inclusión de ruido aislado. Una heurística común es $T_{high} = 3 \times T_{low}$.

```
%%writefile tmp/fig_03_canny.cpp
#define MM_OUT "tmp/fig_03_canny.png"
// Compile with: g++ -std=c++17 -o output code.cpp -I<include_path>
```

```
#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]

    //| label: fig-03-canny
    //| fig-cap: "Detector de Canny com diferentes pares de umbral: umbrales bajos capturan
    más bordes (incluso ruido); umbrales altos retienen solo los bordes más fuertes."
    //| echo: true
    //| output: true

    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.

i Elección de los umbrales

Una heurística común es $T_{high} = 3 \times T_{low}$. Los valores típicos dependen del rango de gradiente de la imagen — `cv2.Canny` acepta valores absolutos en $[0, 255]$. Para imágenes con contraste variable, calcular los umbrales a partir de percentiles de la magnitud de Sobel es más robusto que usar valores fijos.

3.7 Filtros de Orden: Filtro de la Mediana

Los filtros de orden (*order-statistic filters*) reemplazan el píxel central por el valor de un **percentil** de la distribución de intensidades de la vecindad — a diferencia de los filtros lineales, que calculan combinaciones ponderadas. El más importante es el **filtro de la mediana**.

3.7.1 Ruido Impulsivo: Sal y Pimienta

El ruido **sal y pimienta** (*salt-and-pepper noise*) reemplaza píxeles aleatorios por valores extremos: 0 (pimienta, negro) o 255 (sal, blanco). Es común en la transmisión de imágenes con errores de bit y en cámaras con sensores defectuosos.

Para entender por qué fallan los filtros lineales, considere una vecindad 3×3 donde un único píxel ha sido corrompido a 255:

$$\text{vecindad} = \begin{bmatrix} 102 & 98 & 105 \\ 100 & \mathbf{255} & 97 \\ 103 & 99 & 101 \end{bmatrix}$$

Tabla 3.4: Media vs. mediana con un píxel corrompido. La mediana ignora el valor atípico; la media se desplaza ~40 niveles.

Método	Cálculo	Resultado
Media	(102+98+…+255+…+101)/9	140
Mediana	{97,98,99,100,**101**,102,103,105,255}	101

⚠ ¿Por qué fallan los filtros de media con ruido impulsivo?

La media es sensible a los ***valores atípicos*** — un único píxel con valor 255 en una vecindad de valor 100 eleva la salida a 140, propagando el ruido por la imagen. La mediana, por ser un **estimador robusto**, selecciona el valor central de la distribución ordenada, descartando naturalmente los extremos sin ningún ajuste especial.

El siguiente ejemplo ilustra el comportamiento de la media y de la mediana en presencia de un píxel corrupto por ruido impulsivo. Se observa que la media está fuertemente influenciada por el valor extremo (255), produciendo una estimación distante de los valores predominantes de la vecindad. En cambio, la mediana permanece cercana al valor original de la región, evidenciando su mayor robustez frente a *valores atípicos* y justificando su uso en la eliminación de ruido de sal y pimienta.

La Figura 3.23 presenta el efecto del ruido sal y pimienta en diferentes densidades. El ruido fue generado reemplazando aleatoriamente una fracción de los píxeles por valores mínimos (0, pimienta) y máximos (255, sal). A medida que la densidad aumenta del 2% al 10%, crece la cantidad de píxeles corruptos, haciendo que la degradación visual sea más evidente y dificultando la percepción de los detalles de la imagen.

```
%%writefile tmp/fig_03_ruido.cpp
#define MM_OUT "tmp/fig_03_ruido.png"
#include "morph.hpp"
#include <iostream>
#include <vector>
#include <string>
#include <random>
#include <algorithm>
#include <filesystem>

//| 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

mm::Image salt_pepper(const mm::Image& img, double prob) {
    mm::Image out = img;
    int h = out.h;
    int w = out.w;
    std::mt19937 rng(42);
    std::uniform_int_distribution<int> dist_y(0, h - 1);
    std::uniform_int_distribution<int> dist_x(0, w - 1);
    std::uniform_real_distribution<double> dist_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);
        if (dist_rand(rng) < 0.5)
            out.at(y, x) = 0;
        else
            out.at(y, x) = 255;
    }
    return out;
}

int main() {
// [pdi:state-io] auto-generated - do not edit by hand
mm::Image img_gray_crop = mm::_read_state("tmp/state/img_gray_crop_58.png");
// [pdi:state-io:end]

    mm::Image n2 = salt_pepper(img_gray_crop, 0.02);
    mm::Image n5 = salt_pepper(img_gray_crop, 0.05);
    mm::Image n10 = salt_pepper(img_gray_crop, 0.10);

    mm::show(std::vector<mm::Image>{img_gray_crop, n2, n5, n10},
             MM_OUT,
             std::vector<std::string>{"Original", "Ruido 2%", "Ruido 5%", "Ruido 10%"},
             4);
```

```
// [pdi:panel-io] auto-generated - do not edit by hand
std::filesystem::create_directories("tmp");
mm::write(img_gray_crop, "tmp/fig_03_ruido_0.png");
mm::write(n2, "tmp/fig_03_ruido_1.png");
mm::write(n5, "tmp/fig_03_ruido_2.png");
mm::write(n10, "tmp/fig_03_ruido_3.png");
// [pdi:panel-io:end]
return 0;
}
```

```
Overwriting tmp/fig_03_ruido.cpp
```

```
!g++ -I. -std=c++17 tmp/fig_03_ruido.cpp -o tmp/fig_03_ruido \
  && ./tmp/fig_03_ruido \
  && test -f "tmp/fig_03_ruido.png" \
  || echo "  mm::show não gravou tmp/fig_03_ruido.png"
```

```
[1] Original
[2] Ruido 2%
[3] Ruido 5%
[4] Ruido 10%
```

```
try:
    mm.show(
        [
            mm.read("tmp/fig_03_ruido_0.png"),
            mm.read("tmp/fig_03_ruido_1.png"),
            mm.read("tmp/fig_03_ruido_2.png"),
            mm.read("tmp/fig_03_ruido_3.png"),
        ],
        titles=[
            'Original',
            'Ruido 2%',
            'Ruido 5%',
            'Ruido 10%',
        ],
        cols=4,
    )
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + " tmp/fig_03_ruido_0.png
    (ver a versao Python)")
```

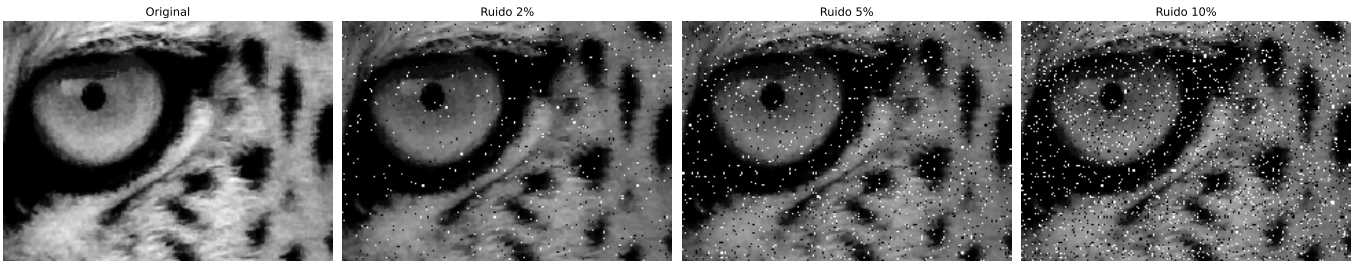


Figura 3.23: Ruído sal e pimenta com densidades crescentes (2%, 5%, 10%): metade dos pixels corrompidos vira sal (255), metade pimenta (0).

3.7.2 Filtro de la Mediana

El filtro de la mediana sustituye cada píxel por el **valor mediano** de los píxeles de su vecindad $n \times n$:

$$g(x,y) = \text{med}_{(s,t)\in\mathcal{V}_n}\{f(x+s, y+t)\} \tag{3.24}$$

El valor mediano es aquel que ocupa la posición central cuando los n^2 valores de la vecindad se ordenan. Para una ventana 3×3 ($n^2 = 9$ píxeles), la mediana es el 5º valor de la secuencia ordenada.

Para ilustrar, considere el mismo *patch* 5×5 con un píxel corrompido artificialmente en $[1, 1]$:

El ejemplo confirma: incluso con el píxel corrompido a 255, la mediana devuelve el valor central correcto — el *outlier* ocupa la última posición en la ordenación y se descarta de forma natural.

Por basarse en la ordenación y no en la suma, la mediana posee tres propiedades fundamentales que la diferencian de los filtros lineales:

- **Robusta** ante el ruido impulsivo — los *outliers* van a los extremos de la secuencia ordenada y no afectan el valor central;
- **Preservadora de bordes** — las transiciones abruptas de intensidad se mantienen, pues la mediana selecciona un valor que ya existe en la vecindad, sin crear nuevos niveles intermedios;
- **No lineal** — no puede expresarse como convolución, por lo que `mm::conv` no se aplica; se utiliza `cv2.medianBlur`.

A Figura 3.24 compara diferentes técnicas de remoção de ruido sal y pimienta aplicadas a una imagen con un 10% de píxeles corruptos. Se evaluaron los filtros Gaussiano, Media, Mediana, Bilateral y Morfológico (próximo capítulo), lo que permite observar la relación entre la eliminación de ruido y la preservación de detalles. En general, los filtros de media y Gaussiano reducen el ruido, pero tienden a desenfocar los bordes, mientras que la mediana presenta un mejor rendimiento para ruido impulsivo. El filtro bilateral preserva mejor los bordes, y el filtro morfológico elimina gran parte de los píxeles corruptos sin degradar excesivamente la estructura de la imagen.

```
%%writefile tmp/fig_03_ruido_filtros.cpp
#define MM_OUT "tmp/fig_03_ruido_filtros.png"
#include "morph.hpp"
#include <random>
#include <vector>
#include <string>
#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

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]

    // Função para adicionar ruído sal e pimenta
    auto 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_p(0.0, 1.0);

        for (int i = 0; i < static_cast<int>(prob * h * w); ++i) {
            int y = dist_y(rng);
```

```
            int x = dist_x(rng);
            out.at(y, x) = (dist_p(rng) < 0.5) ? 0 : 255;
        }
        return out;
    };

    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)")
```

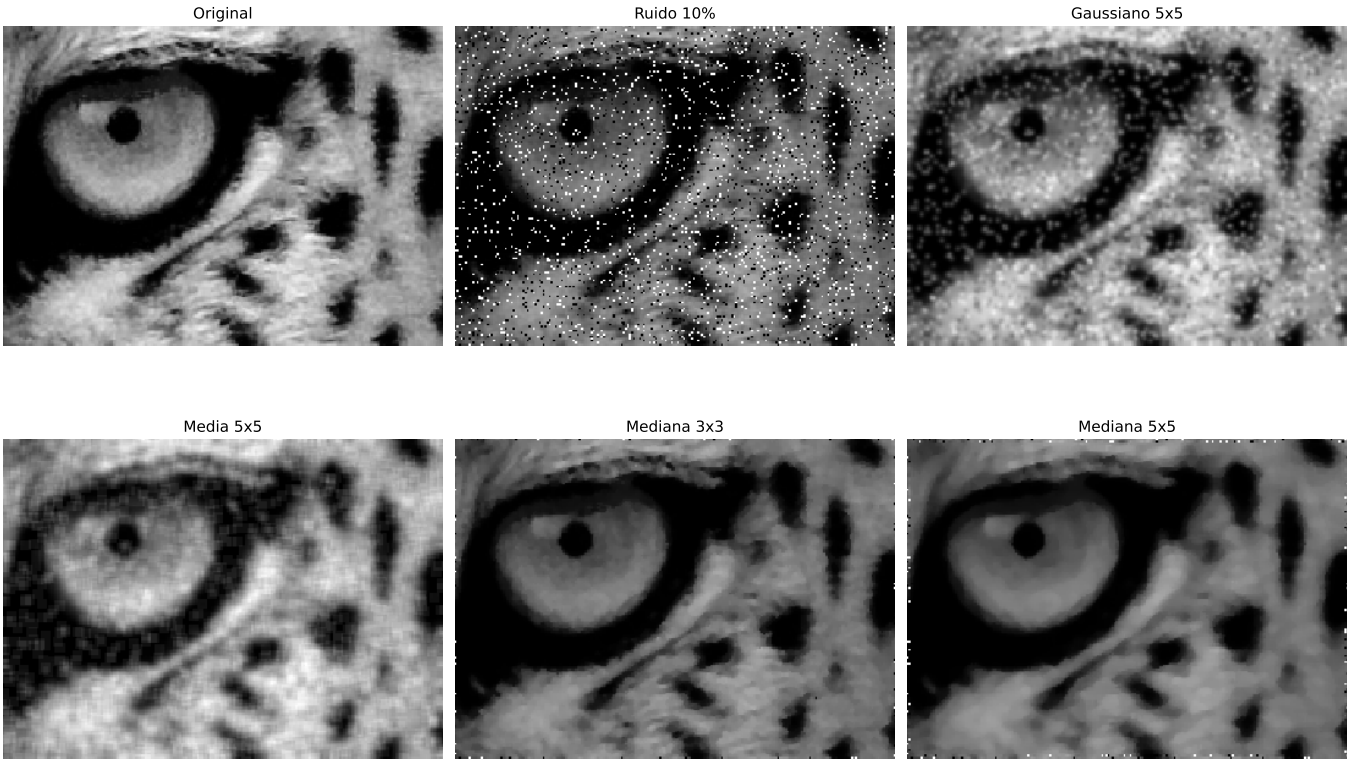


Figura 3.24: Filtros para ruído sal e pimenta (10%): Gaussiano, Média e Mediana. Bilateral e morfológico (open+close) ficam só na trilha Python — bilateral não tem equivalente em morph.hpp e morfologia é do próximo capítulo.

3.8 Aplicación Práctica: Preprocesamiento para Segmentación

En la práctica, las técnicas de este capítulo rara vez se utilizan de forma aislada. Un ***pipeline* de preprocesamiento** típico combina varias etapas en secuencia, adaptándose al tipo de imagen y a la aplicación. La Figura 3.25 ilustra un *pipeline* completo:

1. **Ecualización de histograma (CLAHE):** normaliza el contraste independientemente de las condiciones de iluminación;
2. **Filtro Gaussiano:** suaviza el ruido de adquisición sin destruir bordes;
3. **Detección de bordes (Sobel/Canny):** extrae estructuras relevantes para la segmentación.

i El orden importa

El orden de las operaciones afecta el resultado final. En general: **(1) normalización de intensidad → (2) reducción de ruido → (3) realce/segmentación**. Invertir el orden puede amplificar el ruido o perder bordes antes de detectarlos.

```
%%writefile tmp/fig_03_pipeline.cpp
#define MM_OUT "tmp/fig_03_pipeline.png"
#include "morph.hpp"
#include <iostream>
#include <filesystem>

int main() {
// [pdi:state-io] auto-generated - do not edit by hand
mm::Image img_gray_crop = mm::_read_state("tmp/state/img_gray_crop_58.png");
// [pdi:state-io:end]

    //| label: fig-03-pipeline
    //| fig-cap: "*Pipeline* de pré-processamento: equalização → Gaussiano → Canny. A trilha
    Python usa CLAHE no lugar da equalização global; CLAHE não tem equivalente em morph.hpp."
    //| echo: true
    //| output: true

    mm::Image img_eq    = mm::equalize(img_gray_crop);     // Etapa 1: equalização global
    mm::Image img_gauss = mm::gaussian(img_eq, 5, 0);      // Etapa 2: Gaussiano
    mm::Image edges     = mm::canny(img_gauss, 50, 150);   // Etapa 3: Canny

    mm::Image edges_direct = mm::canny(img_gray_crop, 50, 150);   // Canny direto, sem
pré-processo

    mm::show(
        std::vector<mm::Image>{img_gray_crop, img_eq, img_gauss, edges, edges_direct},
        MM_OUT,
        std::vector<std::string>{"Original", "1. Equalizado", "2. Gaussiano", "3. Canny
(pipeline)", "Canny (direto)"},
        5
    );


// [pdi:panel-io] auto-generated - do not edit by hand
std::filesystem::create_directories("tmp");
mm::write(img_gray_crop, "tmp/fig_03_pipeline_0.png");
mm::write(img_eq, "tmp/fig_03_pipeline_1.png");
mm::write(img_gauss, "tmp/fig_03_pipeline_2.png");
mm::write(edges, "tmp/fig_03_pipeline_3.png");
mm::write(edges_direct, "tmp/fig_03_pipeline_4.png");
// [pdi:panel-io:end]
return 0;
}
```

```
Overwriting tmp/fig_03_pipeline.cpp
```

```
!g++ -I. -std=c++17 tmp/fig_03_pipeline.cpp -o tmp/fig_03_pipeline \
  && ./tmp/fig_03_pipeline \
  && test -f "tmp/fig_03_pipeline.png" \
  || echo "  mm::show não gravou tmp/fig_03_pipeline.png"
```

```
[1] Original
[2] 1. Equalizado
[3] 2. Gaussiano
[4] 3. Canny (pipeline)
[5] Canny (direto)
```

```
try:
    mm.show(
        [
```

```
            mm.read("tmp/fig_03_pipeline_0.png"),
            mm.read("tmp/fig_03_pipeline_1.png"),
            mm.read("tmp/fig_03_pipeline_2.png"),
            mm.read("tmp/fig_03_pipeline_3.png"),
            mm.read("tmp/fig_03_pipeline_4.png"),
        ],
        titles=[
            'Original',
            '1. Equalizado',
            '2. Gaussiano',
            '3. Canny (pipeline)',
            'Canny (direto)',
        ],
        cols=5,
    )
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + " tmp/fig_03_pipeline_0.png
    (ver a versao Python)")
```

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 Resumen

En este capítulo se presentaron las principales técnicas de procesamiento en el dominio espacial, desde la manipulación directa de píxeles hasta el filtrado por vecindad:

- **Operaciones de punto:** aritméticas saturadas (`mm::addm`, `mm::subm`) y lógicas bit a bit (`mm::band`, `mm::bor`, `mm::bnot`) para recorte de ROI y combinación de imágenes; *alpha blending* (`mm::blend`) para fusión ponderada con peso $\alpha \in [0, 1]$.
- **Histograma:** función discreta de distribución de intensidades; visualizado con `mm::histImg` y calculado con `mm::hist`; base para el diagnóstico tonal y para las técnicas de ecualización y especificación.
- **Ecualización:** redistribución automática de las intensidades mediante la CDF (`mm::equalize`), con variante adaptativa CLAHE para el control local del contraste.
- **Especificación de histograma:** transferencia del perfil tonal de una imagen de referencia mediante mapeo inverso de la CDF — generalización de la ecualización para distribuciones arbitrarias.
- **Correlación y convolución:** mecanismo de ventana deslizante implementado en `mm::conv` (`cv2.filter2D`); diferenciados por la rotación de 180° del *kernel* — relevante solo para kernels asimétricos.
- **Filtros de suavizado:** media (*kernel* uniforme, desenfoca bordes proporcionalmente al tamaño) y Gaussiano (ponderación radial, separable, sin *ringing*, conserva mejor los bordes).
- **Filtros de realce:** Laplaciano (w_4/w_8, segunda derivada isotrópica), Sobel (gradiente direccional de primer orden, con magnitud $|\nabla f|$ y dirección θ) y *Unsharp Masking* (amplificación de las altas frecuencias con parámetro k).
- **Filtro de la mediana:** no lineal, robusto a valores atípicos, conserva bordes — superior a los filtros lineales para ruido sal y pimienta.
- **Pipeline práctico:** encadenamiento CLAHE → Gaussiano → Canny como estrategia de preprocesamiento; `mm::drawImgKernel` para visualización didáctica de la ventana deslizante.

El Capítulo 4 abordará la **morfología matemática** (erosión, dilatación, apertura y cierre), explorando en profundidad las funciones `mm::ero` y `mm::dil` de la biblioteca `morph.py`. A continuación, el Capítulo 5 presentará el **procesamiento en el dominio de la frecuencia**, con enfoque en la Transformada de Fourier y en técnicas de filtrado espectral.

3.10 🤖 Uso del Gemini Notebook como Tutor Complementario

En esta edición, incentivamos el uso del **Gemini Notebook** como herramienta complementaria de aprendizaje. Esta herramienta de IA utiliza exclusivamente los documentos proporcionados por el autor como base de conocimiento, garantizando respuestas coherentes con el contenido del libro — incluyendo las funciones de la biblioteca `morph.py` y los experimentos realizados en este capítulo.

Para cada capítulo, hemos preparado un proyecto específico en la plataforma con el PDF del capítulo, los *notebooks* y materiales auxiliares. Sugerimos explorar especialmente:

- **Guía de Estudio:** resumen estructurado de los conceptos, ideal para repasar antes de los exámenes;
- **Conversación:** resuelve dudas sobre ecualización, convolución, filtros y pipelines directamente con el tutor;
- **Preguntas frecuentes:** cuestiones típicas sobre la diferencia entre media y mediana, USM, Lapaciano vs. Sobel.

🎓 Estudia con el Tutor Inteligente

Para interactuar con el contenido de este capítulo, accede al siguiente *enlace*. El entorno contiene materiales didácticos en diferentes formatos, generados a partir del **PDF** del capítulo. En la plataforma, explora especialmente las opciones **Guía de Estudio** y **Conversación** para profundizar tu comprensión.
🚀 ACCEDER A GEMINI NOTEBOOK: CAPÍTULO 03

🌐 Idioma y Lenguaje de Programación

El proyecto de este capítulo en Gemini Notebook fue construido únicamente con el texto en **portugués** y los ejemplos de código en **Python**. Si estás estudiando con la edición en inglés o francés, o siguiendo la ruta en C++, las respuestas del tutor pueden no corresponder exactamente con la versión que estás leyendo.

⚠️ Aviso sobre Contenido Generado por IA

La IA es una poderosa aliada en los estudios, pero el contenido generado puede contener **errores o imprecisiones**. Consulta siempre **libros, artículos científicos y otras fuentes académicas confiables** para validar la información. Siempre que sea posible, ejecuta los ejemplos prácticos proporcionados en este capítulo para verificar los resultados.

3.11 Lista de Ejercicios

1. **(10%)** Explique la diferencia entre **convolución** y **correlación cruzada**. ¿Para qué tipos de *kernel* los resultados son idénticos? Dé un ejemplo de *kernel* asimétrico (como Sobel G_x) y muestre numéricamente que los resultados difieren aplicándolo al parche 5×5 del capítulo de las dos formas.

2. **(15%)** Considere una imagen 5×5 con intensidades concentradas entre los niveles 3 y 5 (bajo contraste, 3 bits). Aplique manualmente el algoritmo de ecualización de la Tabla 3.1, completando todas las columnas de la tabla (k, $h[k]$, $p[k]$, cdf$[k]$, lut$[k]$). Verifique el resultado con `mm::equalize`.

3. **(15%)** Usando `mm::conv`, aplique el filtro de media con kernels de tamaño 3×3, 9×9 y 21×21 a la imagen del mandril. Para cada versión, calcule el **PSNR** (*Peak Signal-to-Noise Ratio*) con respecto a

la original:

$$\text{PSNR} = 10\log_{10}\left(\frac{255^2}{\text{MSE}}\right), \quad \text{MSE} = \frac{1}{MN}\sum_{i,j}(f-g)^2$$

Grafique el PSNR en función del tamaño del kernel y explique qué indica la caída progresiva sobre la relación entre suavizado y pérdida de información.

4. **(15%)** Usando `add_salt_pepper` con densidad del 5%, aplique y compare: (a) `mm::conv` con media 3×3, (b) `cv2.GaussianBlur` con $\sigma = 1$, (c) `cv2.medianBlur` con ventana 3×3 y (d) `cv2.medianBlur` con ventana 5×5. Muestre las imágenes con `mm::show` en una cuadrícula 2×4 (fila 1: imágenes, fila 2: histogramas mediante `mm::histImg`). Explique por qué la mediana supera a los filtros lineales usando el argumento de la Tabla 3.4.

5. **(15%)** Implemente `mm::conv0` usando solo operaciones NumPy vectorizadas — sin bucles en Python y sin `cv2.filter2D` — con el operador de *stride tricks* (`np.lib.stride_tricks.sliding_window_view`). Compare el resultado y el tiempo de ejecución con `mm::conv0` (bucles) y `mm::conv` (cv2) para kernels 3×3 y 15×15 en la imagen del mandril.

6. **(15%)** Aplique el *Unsharp Masking* con $\sigma = 1$ y $k \in \{0.5, 1.0, 2.0, 4.0\}$ usando la función `usm` del capítulo. Para cada valor de k: (a) calcule la diferencia absoluta $|g - f|$, (b) muestre las imágenes y las diferencias con `mm::show`, y (c) grafique el histograma de las diferencias con `mm::histImg`. Identifique a partir de cuál k los artefactos (halos y amplificación de ruido) se vuelven visualmente inaceptables.

7. **(15%)** Elija una imagen de rayos X o tomografía disponible públicamente (ej.: mediante `mm::read` de URL) y diseñe un pipeline de preprocesamiento con al menos 4 etapas secuenciales, justificando cada elección con base en los conceptos del capítulo. Muestre con `mm::show` en una cuadrícula: imagen original, cada etapa intermedia y el resultado final con sus histogramas (`mm::histImg`).

Referencias del Capítulo

La fundamentación teórica de este capítulo se basa en las siguientes obras:

- Gonzalez (2018) para los conceptos de operaciones de intensidad, histograma, convolución y filtrado espacial.
- Szeliski (2022) para la visión por computadora y aplicaciones prácticas de filtrado.
- Bradski (2008) para la implementación práctica con OpenCV y `morph.py`.

3.12 💻 Parte Práctica con Ejercicios de Programación

🎯 Objetivo de este Cuaderno

El cuaderno permite desarrollar, validar, organizar y probar soluciones de **Ejercicios de Programación (EPs)** en entornos interactivos, como Colab, con los mismos casos de prueba de Moodle, copiándolos allí solo al momento de registrar la nota oficial.

Download

Descargue `morph.py` y `testsuite.py` ejecutando la celda a continuación:

```
import os, urllib.request

os.makedirs("tmp/state", exist_ok=True)  # artefatos de build del trayecto C++ (.cpp, binario,
PNGs)
```

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

# El kernel es Python incluso en el trayecto C++: `mm` (morph.py) es usado por los
# simuladores, por la exhibición de las figuras que el binario C++ genera y por el
# estado mm::Image entre celdas. cpp=True descarga también el trayecto compilado
# (morph.hpp + stb_image*.h), usado en el #include de las celdas %%writefile *.cpp.
import config
config.setup(testsuite=True, cpp=True)
from morph import mm
from testsuite import TestSuite
```

```
Entorno listo. Morph: 1.1.9 | OpenCV: 5.0.0 | TestSuite: 1.1.2
```

Ejecutando los Tests

Para evaluar los tests, ejecuta `TestSuite("EP03_01.extensión").run()` en una nueva celda, reemplazando la extensión por la del lenguaje utilizado (`.py`, `.java`, `.c`, `.cpp`, `.js` o `.r`). El sistema descarga los casos de prueba de GitHub, ejecuta el programa y calcula la nota automáticamente.

Para probar código Python directamente, sin guardar un archivo, usa `run_code(codigo)` pasando el código como *string* en una variable `codigo`:

```
codigo = """
from morph import mm
# ... tu código aquí ...
"""
TestSuite("EP03_01").run_code(codigo)
```

3.12.1 EP03_01 ➕ Adición Saturada de Constante

En sistemas de vigilancia por video, las cámaras en entornos con iluminación variable producen imágenes subexpuestas. El ajuste de brillo mediante **adición saturada de una constante** es la operación más simple para la corrección inmediata, aplicándose en tiempo real en los *chips* de cámaras embebidas y en *pipelines* de preprocesamiento de robots móviles.

Ver en Figura 3.26 una simulación de este EP.

3.12.1.1 📋 Directrices de Implementación

1. **Dimensiones:** Leer los enteros L (filas) y C (columnas).
2. **Constante:** Leer el entero k (valor a sumar).
3. **Datos:** Leer los valores enteros de la matriz original fila por fila.
4. **Mapeo:** Para cada píxel p, calcular el nuevo valor mediante la ecuación:

$$p' = \text{clip}(p + k)$$

5. **Salida:** Mostrar la matriz resultante con dimensiones $L \times C$.

3.12.1.2 📌 Restricciones Computacionales

- **Saturación (*Clipping*):** Los valores deben confinarse al intervalo $[0, 255]$:

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

- **Tipo:** El resultado final debe ser entero (sin decimales).
- **k puede ser negativo:** los valores negativos oscurecen la imagen; los positivos la aclaran.

3.12.1.3 🧠 Fundamentación Teórica

Parámetro	Tipo	Impacto Visual
$k > 0$	Entero	Aclara la imagen; los píxeles cercanos a 255 saturan en blanco
$k < 0$	Entero	Oscurece la imagen; los píxeles cercanos a 0 saturan en negro
$k = 0$	Entero	Imagen sin cambios

3.12.1.4 📦 Especificación de Entrada y Salida (VPL)

Entrada:

- Línea 1: Entero L.
- Línea 2: Entero C.
- Línea 3: Entero k.
- Líneas siguientes: Elementos enteros de la matriz original.

Salida:

- Matriz transformada en L filas y C columnas, valores enteros separados por espacios.

3.12.1.5 📌 Ejemplos

Entrada	Salida	Observación
2 3 50 0 100 200 210 240 255	50 150 250 255 255 255	Saturación en 255 en los píxeles altos
1 4 -30 0 20 200 255	0 0 170 225	Saturación en 0 en los píxeles bajos

```
%%writefile EP03_01.cpp
// your solution
```

```
Overwriting EP03_01.cpp
```

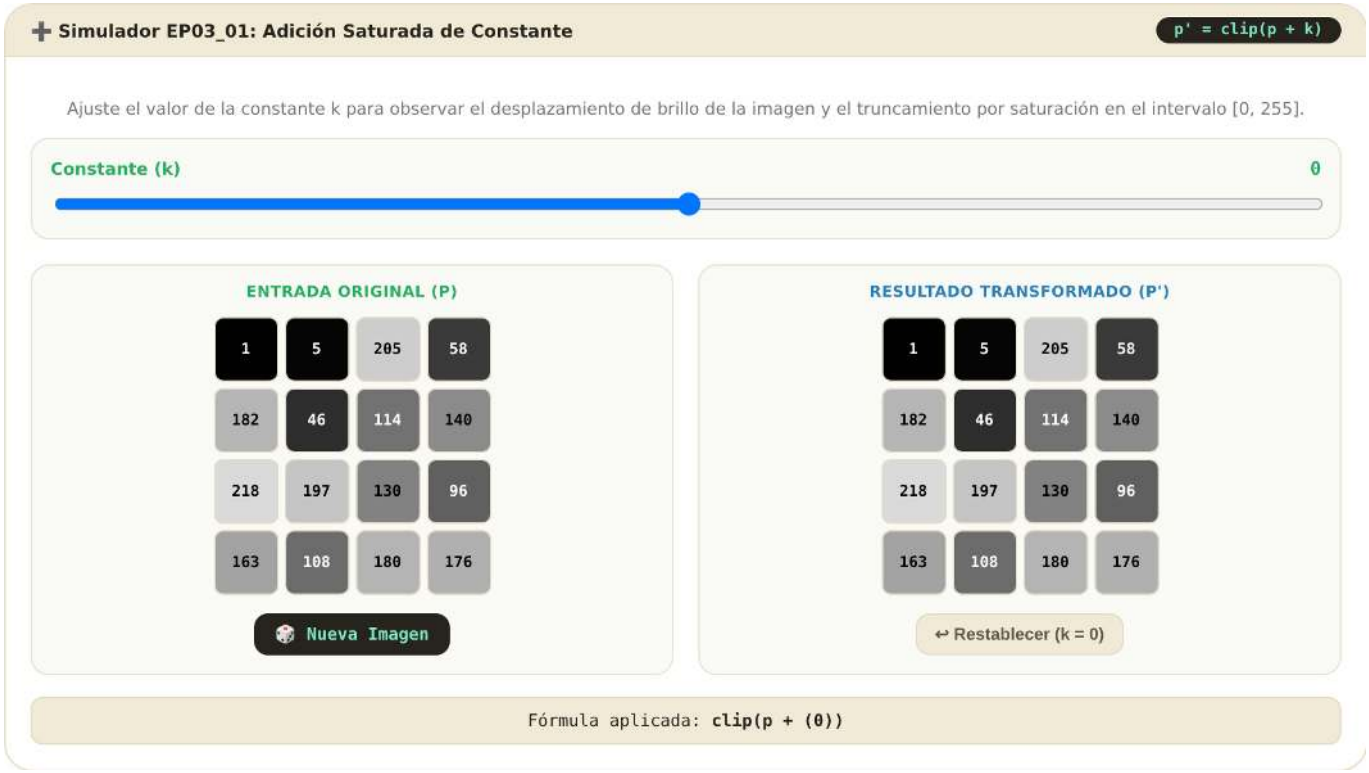
```
TestSuite("EP03_01.cpp").run()
```

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

3.12.2 EP03_02 🔀 Alpha *Blending* de Dos Imágenes

En medicina nuclear, imágenes de diferentes modalidades (tomografía computarizada y resonancia magnética) se fusionan para ayudar en el diagnóstico. La **mezcla ponderada** (*alpha blending*) es la operación fundamental de este proceso, permitiendo al radiólogo controlar interactivamente el peso de cada modalidad en la imagen mostrada.

Ver en Figura 3.27 una simulación de este EP.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.es/cap03/sim-ep0301-adicao.html

Figura 3.26: Simulador EP03_01: Adición Saturada de Constante (p' = clip(p + k))

3.12.2.1 📋 Directrices de Implementación

1. **Dimensiones:** Leer los enteros L (filas) y C (columnas).
2. **Parámetro:** Leer el valor real $\alpha \in [0, 1]$.
3. **Datos:** Leer los valores enteros de la matriz f_1 (imagen 1) y luego de la matriz f_2 (imagen 2).
4. **Mapeo:** Para cada posición (i, j), calcular:

$$g(i, j) = \text{clip}\left(\text{round}\left(\alpha \cdot f_1(i, j) + (1 - \alpha) \cdot f_2(i, j)\right)\right)$$

5. **Salida:** Mostrar la matriz resultante $L \times C$.

3.12.2.2 📌 Restricciones Computacionales

- **Redondeo:** Aplicar `round` antes de la conversión a entero.
- **Saturación:** Confinar al intervalo $[0, 255]$ con $\text{clip}(x) = \max(0, \min(255, x))$.
- **Operación en float:** Realizar la operación en punto flotante antes de redondear.

3.12.2.3 🧠 Fundamentación Teórica

Valor de α	Resultado
$\alpha = 1.0$	Solo f_1
$\alpha = 0.5$	Media aritmética de f_1 y f_2
$\alpha = 0.0$	Solo f_2

3.12.2.4 📦 Especificación de Entrada y Salida (VPL)

Entrada:

- Línea 1: Entero L.

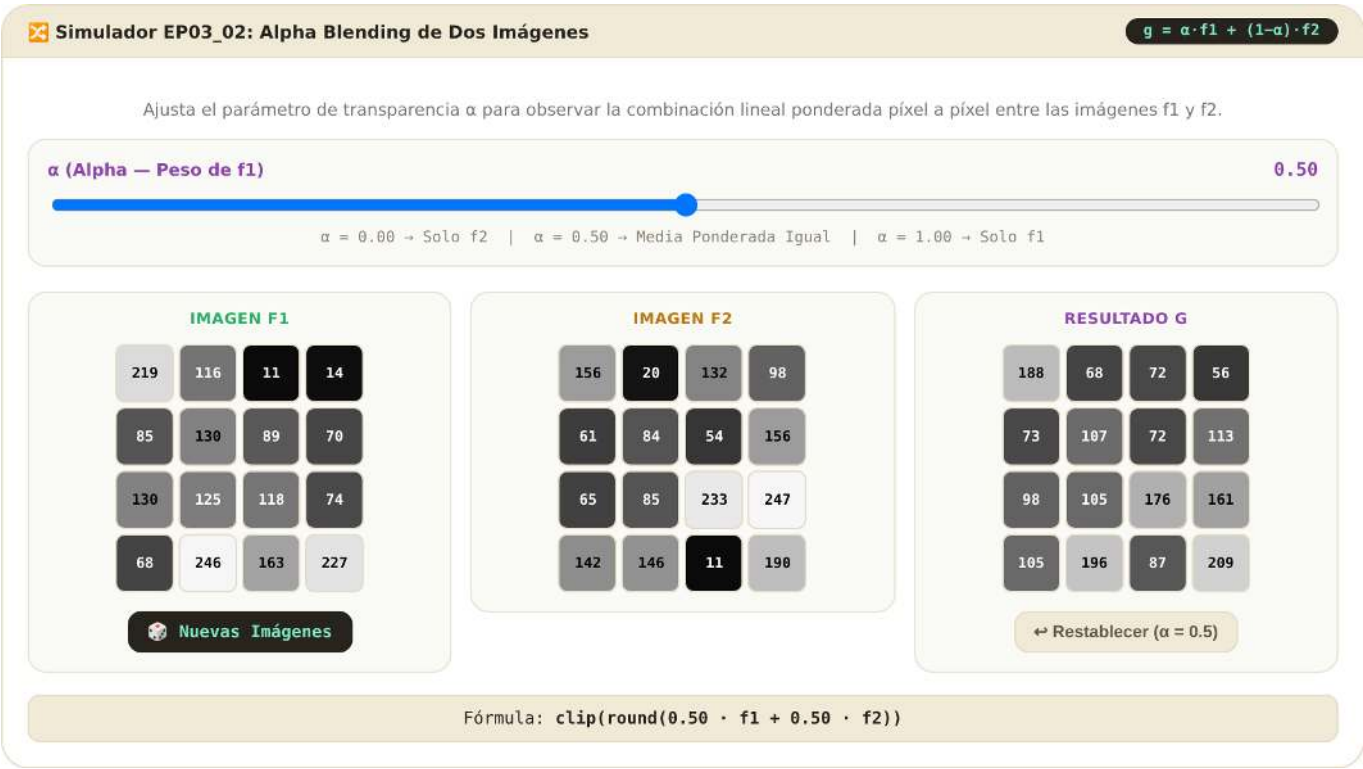
- Línea 2: Entero C.
- Línea 3: Real α.
- Líneas siguientes: Elementos de f_1 (L líneas con C valores cada una).
- Líneas siguientes: Elementos de f_2 (L líneas con C valores cada una).

Salida:

- Matriz resultante $L \times C$.

3.12.2.5 📌 Ejemplos

Entrada	Salida	Observación
1 3 0.5 0 100 200 100 200 50	50 150 125	Media entre las dos imágenes
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.es/cap03/sim-ep0302-blending.html

Figura 3.27: Simulador EP03_02: Mezcla alfa de dos imágenes (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 🎭 Inversión de Imagen (Negativo Fotográfico)

En radiología, las imágenes de rayos X se visualizan tradicionalmente en negativo: los huesos aparecen en negro sobre fondo blanco. La operación de **negativo fotográfico** se aplica rutinariamente en PACS (*Picture Archiving and Communication Systems*) para facilitar la detección de fracturas y densidades óseas.

Ver en Figura 3.28 una simulación de este EP.

3.12.3.1 📋 Directrices de Implementación

1. **Dimensiones:** Leer los enteros L (filas) y C (columnas).
2. **Datos:** Leer los valores enteros de la matriz original.
3. **Mapeo:** Para cada píxel p, calcular el negativo:

$$p' = 255 - p$$

4. **Salida:** Mostrar la matriz resultante de $L \times C$.

3.12.3.2 📌 Restricciones Computacionales

- **Sin necesidad de *recorte*:** El resultado de $255 - p$ con $p \in [0, 255]$ siempre está $\in [0, 255]$.
- **Tipo entero:** La salida debe ser valores enteros.
- **Equivalencia lógica:** La operación es idéntica al `NOT` bit a bit (`mm::bnot`) en imágenes de 8 bits.

3.12.3.3 🧠 Fundamentación Teórica

Píxel Original p	Píxel Negativo p'	Observación
0 (negro)	255 (blanco)	Inversión total
128 (gris medio)	127 (gris medio)	Valor central
255 (blanco)	0 (negro)	Inversión total

3.12.3.4 📦 Especificación de Entrada y Salida (VPL)

Entrada:

- Línea 1: Entero L.
- Línea 2: Entero C.
- Líneas siguientes: Elementos enteros de la matriz original.

Salida:

- Matriz negativa en L filas y C columnas.

3.12.3.5 📌 Ejemplos

Entrada	Salida	Observación
140 128 200 255	255 127 55 0	Inversión de cada píxel
2 2 10 20 30 40	245 235225 215	Matriz 2x2 invertida

Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.es/cap03/sim-ep0303-inversao.html

Figura 3.28: Simulador EP03_03: Inversión de Imagen — Negativo Fotográfico (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 📊 Equalización de Histograma (L bits)

En imágenes de satélite de teleobservación, la variación de iluminación a lo largo del día produce imágenes de bajo contraste. La **equalización de histograma** se aplica automáticamente en satélites como el Landsat para redistribuir los tonos, revelando detalles de vegetación, relieve y zonas urbanas invisibles en la imagen original.

Ver en la Figura 3.29 una simulación de este EP.

3.12.4.1 📋 Directrices de Implementación

1. **Dimensiones:** Leer los enteros L (líneas), C (columnas) y B (número de bits, con $L_{\max} = 2^B$).
2. **Datos:** Leer la matriz de píxeles f con valores en $[0, 2^B - 1]$.
3. **Histograma:** Calcular $h[k]$ = número de píxeles con intensidad k, para $k = 0 \ldots 2^B - 1$.
4. **Probabilidad:** $p[k] = h[k]/(L \cdot C)$.
5. **CDF:** $\text{cdf}[k] = \sum_{j=0}^{k} p[j]$; función de distribución acumulada.
6. **LUT:** $\text{lut}[k] = \text{round}\,(\text{cdf}[k] \cdot (2^B - 1))$; *Look-Up Table* (tabla de consulta).
7. **Aplicación:** $g[i, j] = \text{lut}[f[i, j]]$.
8. **Salida:** Mostrar la matriz equalizada $L \times C$.

3.12.4.2 📌 Restricciones Computacionales

- **Redondeo:** Usar redondeo matemático (`round`) en la LUT.

- **Bits:** El número de niveles es 2^B (ej.: $B = 3 \Rightarrow 8$ niveles, $B = 8 \Rightarrow 256$ niveles).
- **CDF acumulada:** $\text{cdf}[k] = \sum_{j=0}^{k} p[j]$, con $\text{cdf}[2^B - 1] = 1.0$.

3.12.4.3 🧠 Fundamentación Teórica

Etapa	Operación	Fórmula
1	Histograma	$h[k] \leftarrow$ nº píxeles con intensidad k
2	Probabilidad	$p[k] = h[k]/(L \cdot C)$
3	CDF	$\text{cdf}[k] = \sum_{j=0}^{k} p[j]$
4	LUT	$\text{lut}[Look - UpTable(tabladeconsulta)k] = \text{round}(\text{cdf}[k] \cdot (2^B - 1))$
5	Aplicación	$g[i, j] = \text{lut}[f[i, j]]$

3.12.4.4 📦 Especificación de Entrada y Salida (VPL)

Entrada:

- Línea 1: Entero L.
- Línea 2: Entero C.
- Línea 3: Entero B (número de bits).
- Líneas siguientes: Elementos enteros de la matriz.

Salida:

- Matriz equalizada en L líneas y C columnas.

3.12.4.5 📌 Ejemplos

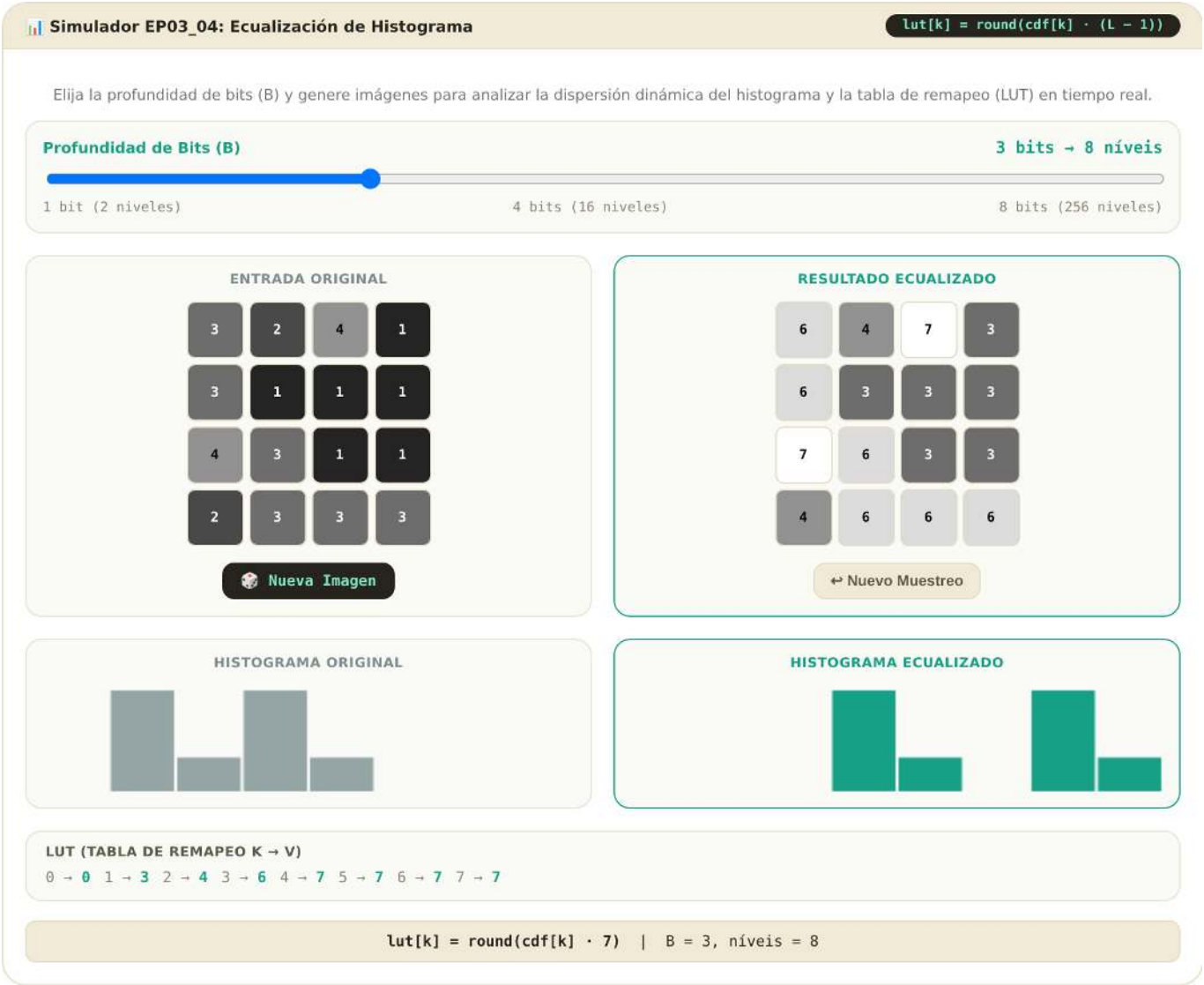
Entrada	Salida	Observación
5 5 3 3 4 2 3 4 4 3 3 4 3 2 3 4 3 2 3 4 3 2 3 4 3 2 3 4	5 7 1 5 7 7 5 5 7 5 1 5 7 5 1 5 7 5 1 5 7 5 1 5 7	Ejemplo 3 bits del capítulo
1 4 3 0 0 7 7	0 0 7 7	Histograma bimodal extremo

```
%%writefile EP03_04.cpp
// your solution
```

```
Overwriting EP03_04.cpp
```

```
TestSuite("EP03_04.cpp").run()
```

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



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.es/cap03/sim-ep0304-equalizacao.html

Figura 3.29: Simulador EP03_04: Ecualización de Histograma (Niveles L = 2^B)

3.12.5 EP03_05 🔲 Aplicación de Máscara AND Binaria

En sistemas de inspección industrial por visión por computadora, es necesario aislar regiones de interés (ROI) en imágenes de piezas para verificar defectos de fabricación. La operación **AND bit a bit con una máscara binaria** es el mecanismo fundamental para recortar exactamente el área de inspección, poniendo a cero todos los píxeles fuera de ella.

Ver en Figura 3.30 una simulación de este EP.

3.12.5.1 📋 Directrices de Implementación

1. **Dimensiones:** Leer los enteros L (filas) y C (columnas).
2. **Datos:** Leer la matriz de píxeles f (valores $\in [0, 255]$).
3. **Máscara:** Leer la matriz binaria m (valores: solo 0 o 255).
4. **Mapeo:** Para cada píxel (i, j), aplicar el AND bit a bit:

$$g(i, j) = f(i, j) \text{ AND } m(i, j)$$

donde $255 =$ `11111111` y $0 =$ `00000000` en binario.

5. **Salida:** Mostrar la matriz resultante $L \times C$.

3.12.5.2 📌 Restricciones Computacionales

- **AND con 255:** p AND $255 = p$ (todos los bits preservados).
- **AND con 0:** p AND $0 = 0$ (todos los bits puestos a cero).
- **Máscara:** Los únicos valores posibles en la máscara son 0 y 255.
- **Implementación:** En Python, el AND bit a bit entre enteros usa el operador `&`.

3.12.5.3 🧠 Fundamentación Teórica

Píxel f	Máscara m	Resultado f AND m
cualquier v	255 (`11111111`)	v (preservado)
cualquier v	0 (`00000000`)	0 (puesto a cero)

3.12.5.4 📦 Especificación de Entrada y Salida (VPL)

Entrada:

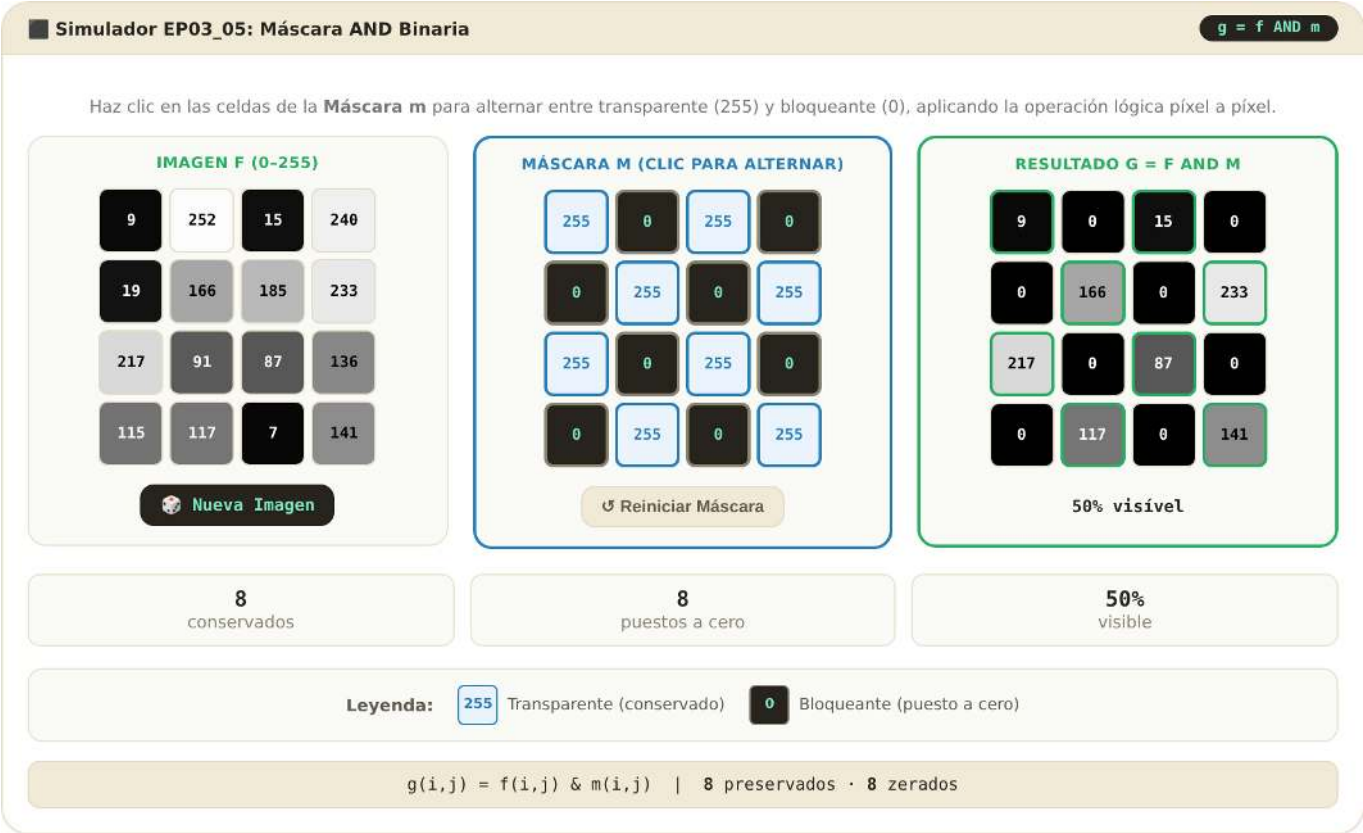
- Línea 1: Entero L.
- Línea 2: Entero C.
- Líneas siguientes: Elementos de f (L líneas).
- Líneas siguientes: Elementos de m (L líneas con valores 0 o 255).

Salida:

- Matriz resultante $L \times C$.

3.12.5.5 📌 Ejemplos

Entrada	Salida	Observación
2 3 100 150 200 50 80 120 255 255 0 0 255 255	100 150 0 0 80 120	La máscara selecciona la región
1 4 10 20 30 40 255 0 255 0	10 0 30 0	Alternado preservado/puesto a cero



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.es/cap03/sim-ep0305-mascara.html

Figura 3.30: Simulador EP03_05: Aplicación de Máscara 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 de Media con *Kernel* N×N

En cámaras de vehículos autónomos, las imágenes capturadas bajo lluvia o niebla presentan ruido gaussiano. El **filtro de media** se utiliza ampliamente para su reducción en tiempo real, implementándose directamente en el **ISP** (*Image Signal Processor*) de los sensores **CMOS** (*Complementary Metal-Oxide-Semiconductor*).

Los sensores CMOS son los sensores de imagen utilizados en la mayoría de las cámaras modernas (*smartphones*, *webcams*, cámaras automotrices, etc.). Convierten la luz en señales eléctricas, y el ISP procesa estas señales en tiempo real — aplicando operaciones como reducción de ruido, balance de blancos y otros ajustes de imagen.

Ver en la Figura 3.31 una simulación de este EP.

3.12.6.1 📋 Directrices de Implementación

1. **Dimensiones:** Leer los enteros L (filas), C (columnas) y N (tamaño del *kernel*, siempre impar).
2. **Datos:** Leer la matriz de píxeles f.
3. **Filtro de Media:** Para cada píxel (i, j) **interno** (sin bordes), calcular:

$$g(i,j) = \text{round}\left(\frac{1}{N^2}\sum_{s=-(r)}^{r}\sum_{t=-(r)}^{r} f(i+s, j+t)\right), \quad r = \lfloor N/2 \rfloor$$

4. **Tratamiento de Bordes:** Los píxeles en el borde (donde la ventana $N \times N$ sobrepasa los límites) deben **copiarse directamente** del original sin modificación.
5. **Salida:** Mostrar la matriz resultante $L \times C$.

3.12.6.2 📌 Restricciones Computacionales

- **Radio:** $r = \lfloor N/2 \rfloor$ (mitad del *kernel*, entero).
- **Píxeles internos:** (i, j) con $r \leq i < L - r$ y $r \leq j < C - r$.
- **Redondeo:** Usar redondeo matemático antes de convertir a entero.
- **Sin *clipping*:** El promedio de valores $\in [0, 255]$ permanece en $[0, 255]$.

3.12.6.3 🧠 Fundamentación Teórica

Tamaño N	Coeficiente	Píxeles en la ventana	Efecto
3	$1/9 \approx 0.111$	9	Suave
5	$1/25 = 0.04$	25	Medio
7	$1/49 \approx 0.020$	49	Fuerte

3.12.6.4 📦 Especificación de Entrada y Salida (VPL)

Entrada:

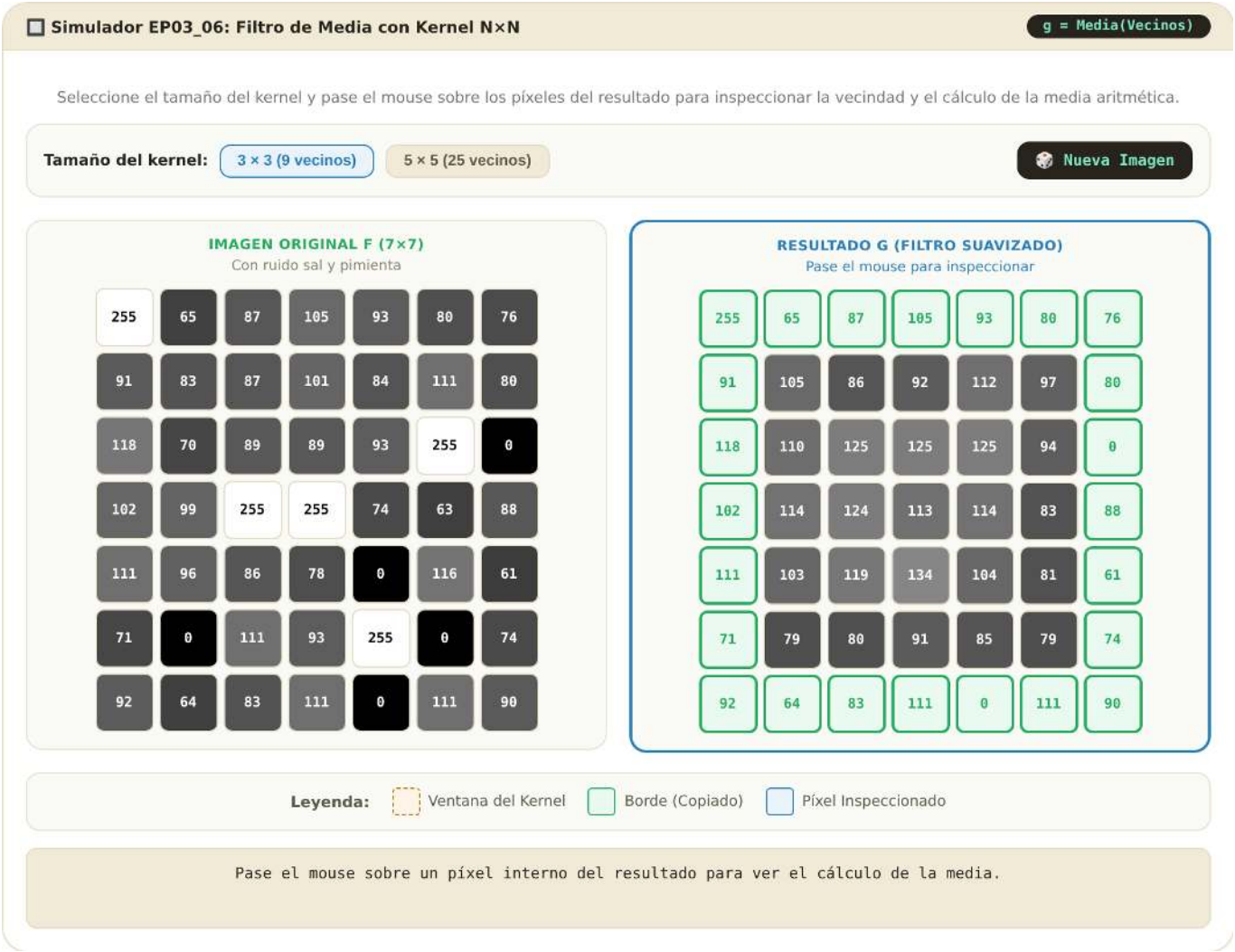
- Línea 1: Entero L.
- Línea 2: Entero C.
- Línea 3: Entero N (impar, $N \geq 3$).
- Líneas siguientes: Elementos de la matriz original.

Salida:

- Matriz filtrada $L \times C$.

3.12.6.5 📌 Ejemplos

Entrada	Salida	Observación
3 3 3 10 20 30 40 50 60 70 80 90	10 20 30 40 50 60 70 80 90	Solo borde (3×3 = borde total)
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	Píxel aislado: todos los 9 píxeles internos cuya ventana 3×3 incluye el valor 100 reciben round(100/9)=11



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.es/cap03/sim-ep0306-media.html

Figura 3.31: Simulador EP03_06: Filtro de Media 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 🔍 Operador Laplaciano (w4) para Realce de Bordas

En tomografías de alta resolución, la nitidez de los bordes entre tejidos es crítica para el diagnóstico. El **operador Laplaciano** se utiliza ampliamente en *pipelines* de preprocesamiento de imágenes médicas para resaltar automáticamente los contornos anatómicos antes de la segmentación, evitando la intervención manual del radiólogo.

Ver en Figura 3.32 una simulación de este EP.

3.12.7.1 📋 Directrices de Implementación

1. **Dimensiones:** Leer los enteros L (filas) y C (columnas).
2. **Datos:** Leer la matriz de píxeles f.
3. **Laplaciano (w4):** Para cada píxel **interno** (i, j) con $1 \leq i < L-1$, $1 \leq j < C-1$, calcular:

$$\nabla^2 f(i,j) = f(i-1,j) + f(i+1,j) + f(i,j-1) + f(i,j+1) - 4 \cdot f(i,j)$$

4. **Realce:** Calcular la imagen realzada:

$$g(i,j) = \text{clip}(f(i,j) - \nabla^2 f(i,j))$$

5. **Borde:** Los píxeles en el borde se copian directamente: $g(i,j) = f(i,j)$.
6. **Salida:** Mostrar la matriz realzada $L \times C$.

3.12.7.2 📌 Restricciones Computacionales

- ***Kernel* w4:** $\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$ — solo vecinos-4.
- **Saturación:** $\text{clip}(x) = \max(0, \min(255, x))$ aplicado al resultado del realce.
- **Sin redondeo:** El Laplaciano utiliza solo sumas/restas de enteros.

3.12.7.3 🧠 Fundamentación Teórica

Región	$\nabla^2 f$	Efecto del Realce
Uniforme	≈ 0	Sin alteración
Borde creciente	< 0	Píxel aclarado
Borde decreciente	> 0	Píxel oscurecido

3.12.7.4 📦 Especificación de Entrada y Salida (VPL)

Entrada:

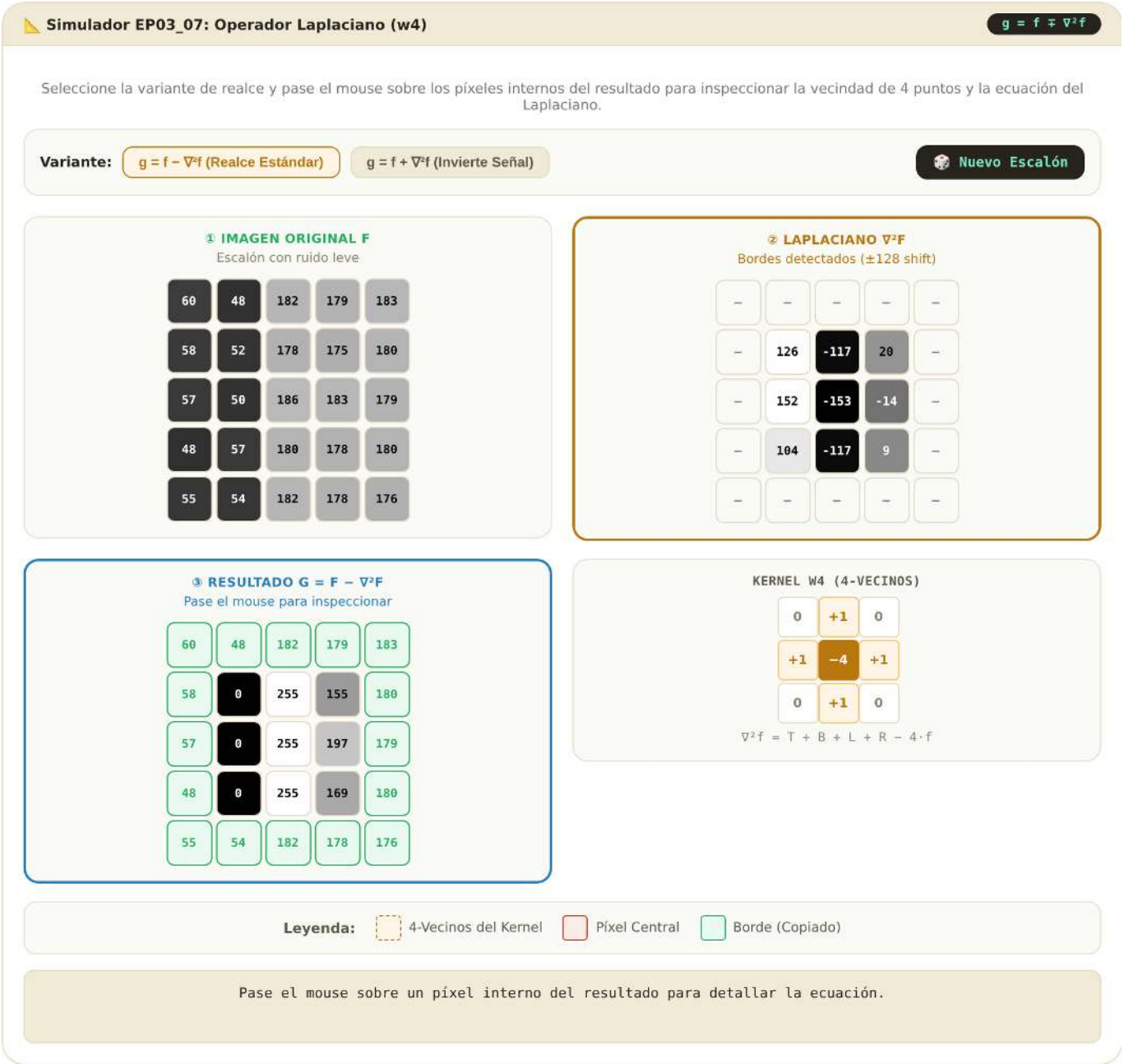
- Línea 1: Entero L.
- Línea 2: Entero C.
- Líneas siguientes: Elementos de la matriz original.

Salida:

- Matriz realzada $L \times C$.

3.12.7.5 📌 Ejemplos

Entrada	Salida	Observación
3 3 0 0 0 0 100 0 0 0 0	0 0 0 0 255 0 0 0 0	Pico aislado: 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	Región uniforme: Laplaciano=0, sin alteración



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.es/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 de Sobel: Gx y Gy

En robots exploradores de Marte (como el Perseverance), la detección de obstáculos se realiza en tiempo real mediante cámaras estereoscópicas. El **operador de Sobel** calcula el gradiente direccional de la escena y se utiliza en el algoritmo de detección de bordes para identificar rocas, fisuras y desniveles del terreno que puedan comprometer la navegación.

Ver en Figura 3.33 una simulación de este EP.

3.12.8.1 📋 Directrices de Implementación

1. **Dimensiones:** Leer los enteros L (filas) y C (columnas).
2. **Datos:** Leer la matriz f.
3. **Gx y Gy:** Para cada píxel **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. **Magnitud:** $|\nabla f(i,j)| = \text{clip}(\text{round}(\sqrt{G_x^2 + G_y^2}))$.
5. **Borde:** Los píxeles de borde reciben magnitud 0.
6. **Salida:** Mostrar la magnitud $L \times C$.

3.12.8.2 📌 Restricciones Computacionales

- **Redondeo:** Aplicar `round` antes de convertir a entero.
- **Saturación:** $\text{clip}(x) = \max(0, \min(255, x))$.
- **Raíz cuadrada:** Usar $\sqrt{G_x^2 + G_y^2}$ (no la aproximación $|G_x| + |G_y|$).

3.12.8.3 🧠 Fundamentación Teórica

Operador	Detecta	Coeficientes diagonales
G_x	Bordes verticales	± 1
G_y	Bordes horizontales	± 1
$\lvert\nabla f\rvert$	Todos los bordes	Combinado

3.12.8.4 📦 Especificación de Entrada y Salida (VPL)

Entrada:

- Línea 1: Entero L.

- Línea 2: Entero C.
- Líneas siguientes: Elementos de la matriz.

Salida:

- Magnitud del gradiente, matriz $L \times C$.

3.12.8.5 📌 Ejemplos

Entrada	Salida	Observación
3 3 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0	Imagen nula: gradiente cero
3 3 0 0 255 0 0 255 0 0 255	0 0 0 0 255 0 0 0 0	Borde vertical central: 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 de la Mediana 3×3

Las imágenes de radar de apertura sintética (SAR) utilizadas en monitoreo ambiental y militar sufren de un tipo específico de ruido llamado *speckle*, que posee características similares al ruido sal y pimienta. El **filtro de la mediana** es el método estándar para eliminar este ruido porque preserva los bordes de las estructuras mientras elimina los puntos espurios.

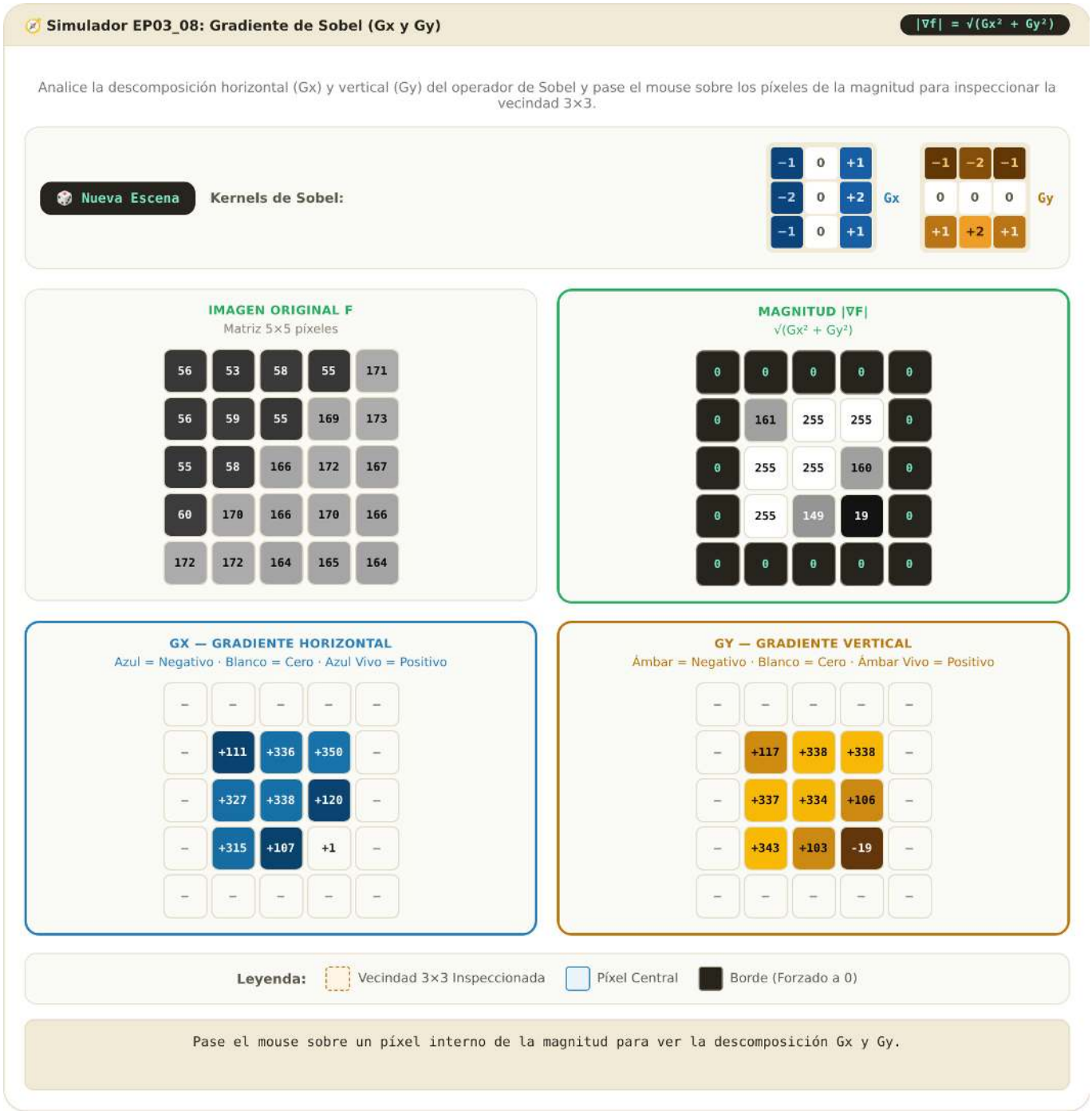
Ver en Figura 3.34 una simulación de este EP.

3.12.9.1 📋 Directrices de Implementación

1. **Dimensiones:** Leer los enteros L (filas) y C (columnas).
2. **Datos:** Leer la matriz de píxeles f.
3. **Filtro de la Mediana 3×3:** Para cada píxel **interno** (i, j) con $1 \leq i < L-1$, $1 \leq j < C-1$:
   - Recolectar los 9 píxeles de la vecindad 3×3: $\{f(i+s, j+t) : s, t \in \{-1, 0, 1\}\}$.
   - Ordenar los 9 valores en orden creciente.
   - Asignar $g(i, j)$ al valor central (posición índice 4, considerando índice 0).

$$g(i,j) = \text{mediana}\{f(i+s, j+t) : s, t \in \{-1, 0, 1\}\}$$

4. **Borde:** Copiar directamente: $g(i, j) = f(i, j)$.
5. **Salida:** Mostrar la matriz filtrada $L \times C$.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.es/cap03/sim-ep0308-sobel.html

Figura 3.33: Simulador EP03_08: Gradiente de Sobel (Gx y Gy)

3.12.9.2 📌 Restricciones Computacionales

- **Ventana:** Siempre $3 \times 3 = 9$ elementos.
- **Mediana:** El elemento central de la secuencia ordenada (índice 4 de 0 a 8).
- **Sin recorte:** La mediana de valores en $[0, 255]$ permanece en $[0, 255]$.
- **No lineal:** El filtro de mediana no puede expresarse como convolución lineal.

3.12.9.3 🧠 Fundamentación Teórica

Ruido	Filtro de Media	Filtro de Mediana
Sal y pimienta (0 o 255)	Dispersa el ruido	Elimina sin distorsionar bordes
Gaussiano	Reduce eficazmente	Reduce parcialmente

3.12.9.4 📦 Especificación de Entrada y Salida (VPL)

Entrada:

- Línea 1: Entero L.
- Línea 2: Entero C.
- Líneas siguientes: Elementos de la matriz.

Salida:

- Matriz filtrada $L \times C$.

3.12.9.5 📌 Ejemplos

Entrada	Salida	Observación
3 3 100 100 100 100 0 100 100 100 100	100 100 100 100 100 100 100 100 100	Punto negro eliminado: mediana de $8{\times}100{+}1{\times}0 = 100$
3 3 50 50 50 50 255 50 50 50 50	50 50 50 50 50 50 50 50 50	Punto blanco (sal) eliminado

```
%%writefile EP03_09.cpp
// your solution
```

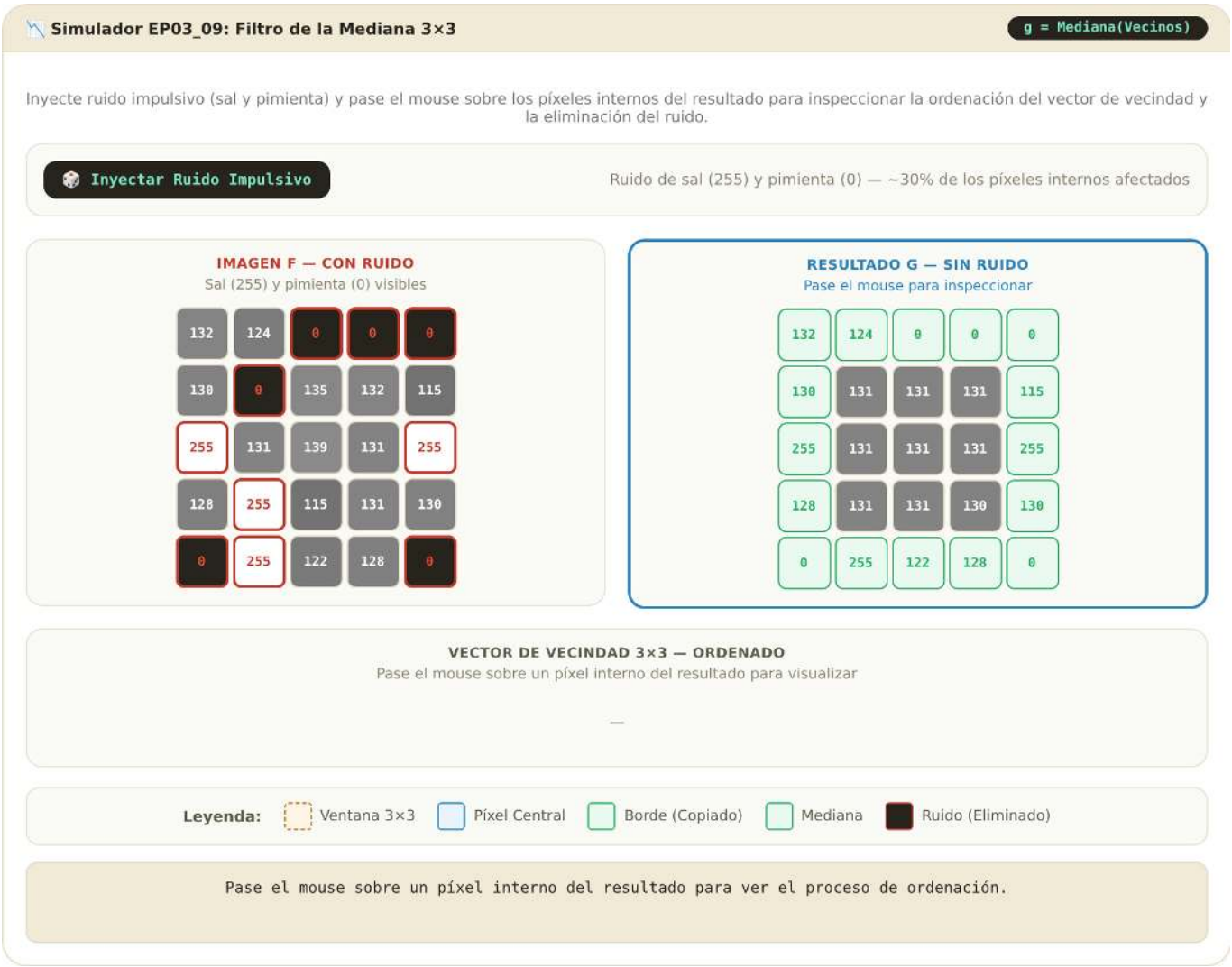
```
Overwriting EP03_09.cpp
```

```
TestSuite("EP03_09.cpp").run()
```

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

3.12.10 EP03_10 ✨ *Unsharp Masking* (USM)

En los sistemas de digitalización de documentos históricos y obras de arte, la nitidez de las imágenes es fundamental para la lectura de textos manuscritos y detalles ornamentales. El ***Unsharp Masking*** **(USM)**



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.es/cap03/sim-ep0309-mediana.html

Figura 3.34: Simulador EP03_09: Filtro de la Mediana 3×3

es el algoritmo de realce de nitidez estándar utilizado en *escáneres* profesionales y software como Adobe Photoshop, controlado por el parámetro k que determina la intensidad del realce.

Ver en la Figura 3.35 una simulación de este EP.

3.12.10.1 📋 Directrices de Implementación

1. **Dimensiones:** Leer los enteros L (líneas) y C (columnas).
2. **Parámetro:** Leer el valor real k (intensidad del realce, $k \geq 0$).
3. **Datos:** Leer la matriz de píxeles f.
4. **Suavizado:** Calcular $\bar{f}$ con el filtro de promedio 3×3 (solo píxeles internos; bordes mantenidos):

$$\bar{f}(i,j) = \frac{1}{9} \sum_{s=-1}^{1} \sum_{t=-1}^{1} f(i+s, j+t)$$

5. **Máscara de alta frecuencia:** $m(i,j) = f(i,j) - \bar{f}(i,j)$.
6. **Realce USM:** Para cada píxel interno:

$$g(i,j) = \text{clip}\left(\text{round}\left(f(i,j) + k \cdot m(i,j)\right)\right)$$

7. **Borde:** $g(i,j) = f(i,j)$ (copia directa).
8. **Salida:** Mostrar la matriz realzada $L \times C$.

3.12.10.2 📌 Restricciones Computacionales

- **Redondeo:** Aplicar `round` antes del clip.
- **Saturación:** $\text{clip}(x) = \max(0, \min(255, x))$.
- **Operaciones en float:** Calcular $\bar{f}$ y m en punto flotante antes de redondear el resultado final.
- $k = 0$**:** Sin realce — la salida es idéntica a la entrada (excepto en los bordes).

3.12.10.3 🧠 Fundamentación Teórica

Etapa	Operación	Descripción
1	$\bar{f} = f * \frac{1}{9}\mathbf{1}_{3\times3}$	Suavizado (bajas frecuencias)
2	$m = f - \bar{f}$	Máscara (altas frecuencias)
3	$g = \text{clip}(\text{round}(f + k \cdot m))$	Realce ponderado

3.12.10.4 📦 Especificación de Entrada y Salida (VPL)

Entrada:

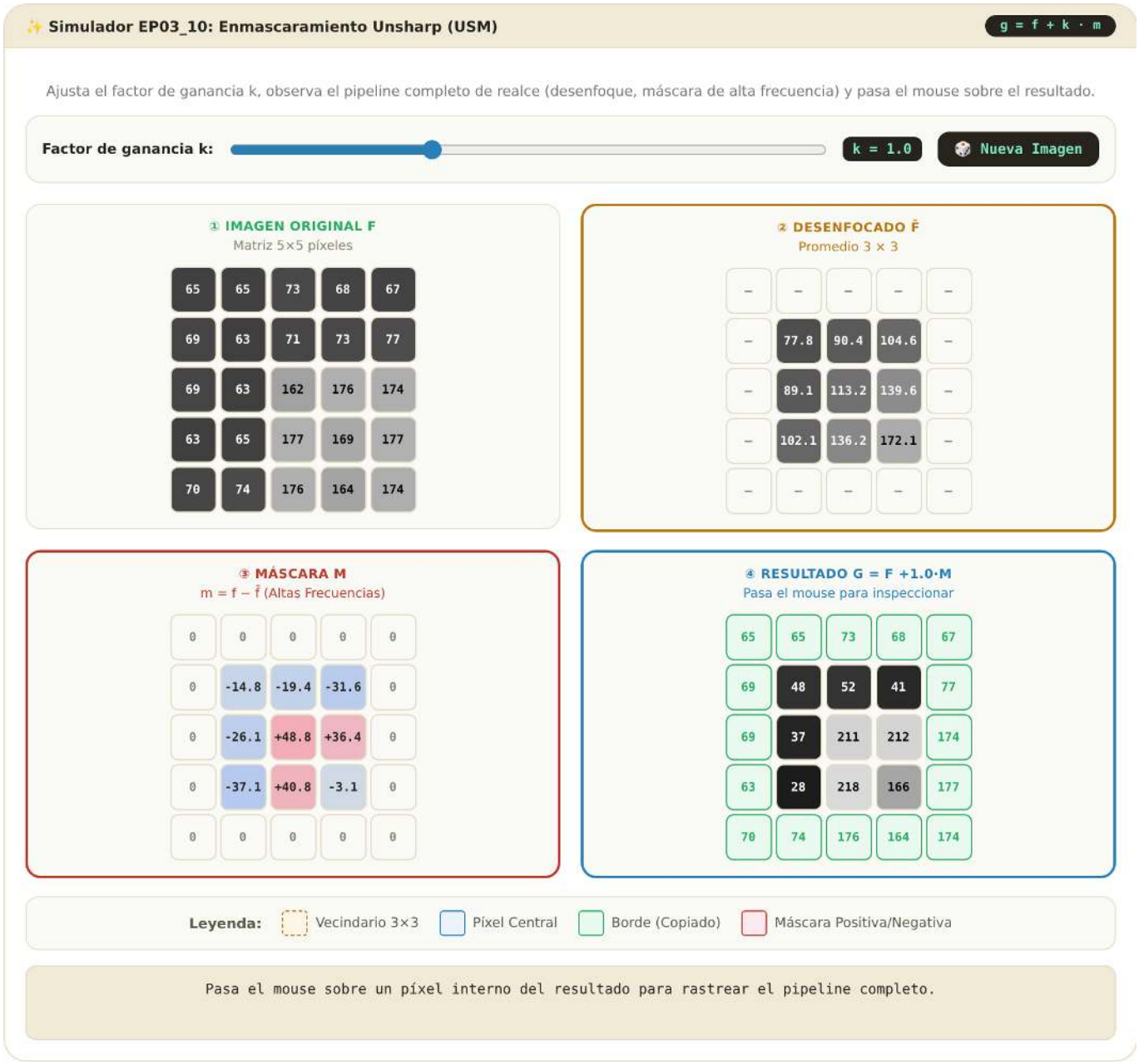
- Línea 1: Entero L.
- Línea 2: Entero C.
- Línea 3: Real k.
- Líneas siguientes: Elementos de la matriz original.

Salida:

- Matriz realzada $L \times C$.

3.12.10.5 📌 Ejemplos

Entrada	Salida	Observación
3 3 0.0 100 100 100 100 100 100 100 100 100	100 100 100 100 100 100 100 100 100	k=0: sin realce
3 3 1.0 50 50 50 50 200 50 50 50 50	50 50 50 50 255 50 50 50 50	k=1: píxel central realzado y saturado



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.es/cap03/sim-ep0310-unsharp.html

Figura 3.35: Simulador EP03_10: *Máscara de enfoque* (USM)

```
%%writefile EP03_10.cpp
// your solution
```

```
Overwriting EP03_10.cpp
```

```
TestSuite("EP03_10.cpp").run()
```

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

Capítulo 4

Morfología Matemática y Segmentación de Imágenes

Este capítulo presenta dos temas fundamentales del Procesamiento Digital de Imágenes (PDI): la **morfología matemática** y la **segmentación de imágenes**. La morfología matemática proporciona un marco teórico basado en la teoría de conjuntos para analizar, refinar y cuantificar la forma de objetos en imágenes binarias y en tonos de gris, mediante operadores fundamentales como la **erosión** y la **dilatación**. La segmentación, por su parte, tiene como objetivo particionar la imagen en regiones de interés, separando objetos del fondo y produciendo representaciones adecuadas para el análisis y la interpretación.

El capítulo comienza con la **umbralización**, una de las técnicas más importantes de segmentación, introduciendo el método automático de **Otsu** y revisitando el análisis de histogramas mediante la varianza interclases, presentada en el Capítulo 1. A continuación, se estudian los principales operadores de la morfología matemática, incluyendo erosión, dilatación, apertura, cierre y reconstrucción morfológica, que permiten refinar máscaras binarias y preservar estructuras relevantes de los objetos. Finalmente, se presentan técnicas de segmentación basada en regiones, como el **etiquetado de componentes conexos**, la **transformada de distancia** y el algoritmo ***watershed*** basado en marcadores, culminando en la extracción de descriptores geométricos y en la generación de *bounding boxes* compatibles con sistemas modernos de detección de objetos.

4.1 Objetivos

Al final de este capítulo, usted será capaz de:

- **Aplicar umbralización:** Comprender el criterio automático de Otsu por maximización de la varianza interclases (σ_B^2) y seleccionar estrategias adecuadas de preprocesamiento para facilitar la segmentación;
- **Dominar la morfología binaria:** Comprender y aplicar erosión ($A \ominus B$) y dilatación ($A \oplus B$) como operadores fundamentales, derivando apertura ($A \circ B$), cierre ($A \bullet B$) y operaciones basadas en reconstrucción morfológica, como `mm::clohole` y `mm::edgeoff`;
- **Aplicar morfología en tonos de gris:** Utilizar gradiente morfológico y filtros *top-hat* para realce y análisis de estructuras locales;
- **Etiquetar componentes conexos:** Identificar y separar regiones conectadas en imágenes binarias mediante algoritmos de etiquetado;
- **Aplicar transformada de distancia:** Interpretar y calcular distancias al fondo utilizando enfoques morfológicos y métricas geométricas;
- **Segmentar por regiones:** Construir *pipelines* de segmentación basados en marcadores utilizando Transformada de Distancia y el algoritmo ***watershed***;
- **Extraer descriptores geométricos:** Calcular propiedades como área, perímetro, centroide, circularidad y *bounding boxes* mediante `mm::label0` y extracción de contornos;
- **Relacionar PDI y visión computacional:** Comprender cómo los descriptores extraídos por segmentación pueden convertirse a formatos utilizados por detectores modernos, como YOLO.

```
import os, urllib.request

os.makedirs("tmp/state", exist_ok=True)  # artefactos de build de la trilha C++ (.cpp,
binário, PNGs)

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

# O kernel é Python mesmo na trilha C++: `mm` (morph.py) é usado pelos
# simuladores, pela exibição das figuras que o binário C++ gera e pelo
# estado mm::Image entre células. cpp=True baixa também a trilha compilada
# (morph.hpp + stb_image*.h), usada no #include das células %%writefile *.cpp.
# As células C++ deste capítulo compilam COM OpenCV (-DMM_USE_OPENCV +
# pkg-config opencv4): mm::dil/ero → cv::dilate/erode, então open/close/asf/
# gradm/tophat/blackhat rodam full-res e batem bit a bit com a trilha py.
import config
config.setup(cpp=True)
from morph import mm
import numpy as np
```

```
Entorno listo. Morph: 1.1.9 | OpenCV: 5.0.0
```

4.2 Umbralización

La **umbralización** (*thresholding*) es una de las formas más simples y eficientes de segmentación de imágenes. Su objetivo es clasificar cada píxel en dos clases de intensidad, normalmente asociadas a *objeto* y *fondo*:

$$g(x,y) = \begin{cases} 255, & \text{si } f(x,y) > T \\ 0, & \text{en caso contrario} \end{cases} \tag{4.1}$$

donde $f(x,y)$ representa la intensidad del píxel en la imagen original y $g(x,y)$ la imagen binaria resultante.

La elección del umbral T es importante para la calidad de la segmentación. El método de **Otsu** determina automáticamente el umbral óptimo al maximizar la **varianza entre clases** σ_B^2 definida por:

$$\sigma_B^2(T) = w_0(T)\, w_1(T)\, \left[\mu_0(T) - \mu_1(T)\right]^2 \tag{4.2}$$

donde:

- $w_0(T)$ y $w_1(T)$ son las probabilidades acumuladas de las clases fondo y objeto;
- $\mu_0(T)$ y $\mu_1(T)$ son las medias de intensidad de esas clases;
- $\sigma_B^2(T)$ representa la varianza entre clases para un umbral dado T.

El método funciona mejor cuando el histograma presenta dos grupos de intensidades relativamente separados. Para ello, el algoritmo evalúa todos los umbrales posibles de la imagen — típicamente en el intervalo $[0, 255]$ para imágenes de 8 bits — y selecciona el valor que maximiza la varianza entre clases, denotada por σ_B^2:

$$T^* = \arg\max_{T \in [0,255]} \sigma_B^2(T)$$

> **i** Otsu asume histogramas bimodales
>
> El método de Otsu produce mejores resultados cuando el histograma presenta dos picos bien definidos (*bimodalidad*), correspondientes al fondo y al objeto. Cuanto mayor sea la separación entre estos picos y

más pronunciado sea el máximo de σ_B^2, más confiable tiende a ser el umbral obtenido.
En imágenes con iluminación no uniforme o múltiples regiones de intensidad, las técnicas de **umbralización adaptativa** — en las cuales el umbral se calcula localmente — suelen producir segmentaciones más robustas.
El subíndice B en σ_B^2 significa ***between classes*** (*entre clases*). Así, σ_B^2 representa la **varianza entre las clases** (*between-class variance*).

4.2.1 Imagen de Monedas

La imagen utilizada para practicar la segmentación es una fotografía de una colección de monedas de diferentes países y épocas (Figura 4.1). Crédito: GAZI.MD.AHAD (CC BY-SA 4.0). Presenta objetos circulares con bordes bien definidos, siendo ideal para demostrar umbralización, operadores morfológicos, transformada de distancia, *watershed* y descriptores de forma.

```
%%writefile tmp/fig_04_coins.cpp
#define MM_OUT "tmp/fig_04_coins.png"
//| label: fig-04-coins
//| fig-cap: "Imagem com moedas de vários tipos. Crédito: GAZI.MD.AHAD (CC BY-SA 4.0)."
//| echo: true

#include "morph.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 Preprocesamiento para Otsu

El método de Otsu depende de un histograma bien **bimodal**. La imagen de las monedas tiene iluminación no uniforme y monedas oscuras cerca del fondo, lo cual dificulta el proceso. En la ruta C++ se aplica **ecualización global del histograma** (`mm::equalize`) antes de la umbralización — la comparación CLAHE × Gaussiano y las curvas $\sigma_B^2(T)$ se encuentran en la ruta Python (Figura 4.2).

```
%%writefile tmp/fig_04_otsu_comparacao_histogramas.cpp
#define MM_OUT "tmp/fig_04_otsu_comparacao_histogramas.png"
//| label: fig-04-otsu-comparacao-histogramas
//| fig-cap: "CLAHE (realce adaptativo com limite de contraste) antes da limiarização de Otsu:
original, realçado, histograma e binarização. (A trilha Python também traça as curvas de
$\\sigma^2_B(T)$.)"
//| echo: true
//| output: true
```

```
#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]

    mm::Image img_clahe = mm::clahe(img_coins_gray, 2.0, 8);   // realce adaptativo com limite
de contraste
    mm::Image img_gauss = mm::gaussian(img_clahe, 5, 0);

    mm::show(
        std::vector<mm::Image>{img_coins_gray, img_clahe, mm::histImg(img_clahe),
mm::threshold(img_clahe)},
        MM_OUT,
        std::vector<std::string>{"Original", "CLAHE", "Histograma (CLAHE)", "Otsu"},
        4
    );


// [pdi:state-io] auto-generated - do not edit by hand
std::filesystem::create_directories("tmp/state");
mm::write(img_clahe, "tmp/state/img_clahe_12.png");
// [pdi:state-io:end]
return 0;
}
```

```
Overwriting tmp/fig_04_otsu_comparacao_histogramas.cpp
```

```
!g++ -I. -std=c++17 -DMM_USE_OPENCV -I/usr/include/opencv4
tmp/fig_04_otsu_comparacao_histogramas.cpp -o tmp/fig_04_otsu_comparacao_histogramas
-lopencv_imgproc -lopencv_imgcodecs -lopencv_core \
  && ./tmp/fig_04_otsu_comparacao_histogramas \
  && test -f "tmp/fig_04_otsu_comparacao_histogramas.png" \
  || echo "  mm::show não gravou tmp/fig_04_otsu_comparacao_histogramas.png"
```

```
[1] Original
[2] CLAHE
[3] Histograma (CLAHE)
[4] Otsu
```

```
try:
    mm.show(mm.read("tmp/fig_04_otsu_comparacao_histogramas.png"))
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + "
    tmp/fig_04_otsu_comparacao_histogramas.png (ver a versao Python)")
```

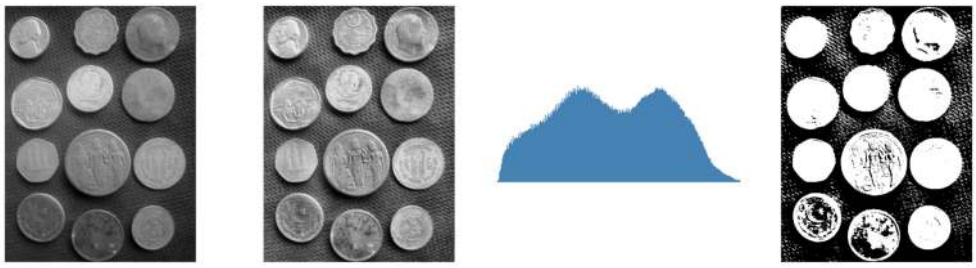


Figura 4.2: CLAHE (realce adaptativo com limite de contraste) antes da limiarização de Otsu: original, realçado, histograma e binarização. (A trilha Python também traça as curvas de $\sigma_B^2(T)$.)

4.2.3 Resultado: CLAHE como Mejor Preprocesamiento

El análisis de la Figura 4.2 indica que el **CLAHE** obtuvo el mayor valor de la varianza entre clases ($\sigma_B^2 \approx 2{,}47 \times 10^3$), con umbral óptimo $T^* = 122$. Aunque la combinación **CLAHE+Gaussiano** haya producido un resultado muy similar ($\sigma_B^2 \approx 2{,}43 \times 10^3$, $T^* = 123$), el criterio cuantitativo del método de Otsu favorece ligeramente el uso del CLAHE de forma aislada.

En términos visuales, las imágenes binarizadas obtenidas con CLAHE y CLAHE+Gaussiano son prácticamente equivalentes. La diferencia entre ambos enfoques se hace más evidente en el análisis de los histogramas y de los valores de $\sigma_B^2(T)$ que en la inspección directa de las segmentaciones resultantes. Así, la elección del CLAHE se basa principalmente en la maximización de la separación estadística entre las clases de fondo y objeto.

Interpretación de los resultados

Observe que los preprocesamientos con CLAHE y CLAHE+Gaussiano producen histogramas y umbrales óptimos muy próximos ($T^* = 122$ y $T^* = 123$). En consecuencia, las imágenes binarizadas resultantes también son bastante similares. En este caso, la decisión no se basa en diferencias visuales marcadas, sino en el criterio objetivo del método de Otsu: el mayor valor de σ_B^2 indica la mejor separación entre las clases.

4.3 Morfología Matemática

La **morfología matemática** es una teoría basada en conjuntos utilizada para analizar la forma y la estructura de objetos en imágenes. A diferencia de los filtros lineales presentados en el Capítulo 3, los operadores morfológicos son **no lineales**, pues se basan en operaciones de mínimo, máximo e inclusión espacial, en lugar de combinaciones lineales de intensidades. Estos operadores actúan sobre la vecindad de cada píxel mediante un **elemento estructurante** $\mathbb{B}$, responsable de definir la forma y el tamaño de la región analizada.

En imágenes binarias y en tonos de gris con elementos planos, el elemento estructurante trasladado a la posición x se define espacialmente como:

$$\mathbb{B}_x = \{x + b \mid b \in \mathbb{B}\}$$

En las regiones de borde de la imagen, parte del conjunto $\mathbb{B}_x$ puede extrapolar el dominio físico de la escena ($\mathbb{E}$). Para garantizar la consistencia matemática de los operadores primitivos en esas fronteras, se asume teóricamente que el espacio exterior al dominio de la imagen se rellena con el **elemento neutro** de la operación correspondiente (infinito positivo para la erosión e infinito negativo para la dilatación), impidiendo que el entorno externo corrompa las estructuras internas del objeto.

Cuando el elemento estructurante asocia pesos a sus elementos — es decir, $b : \mathbb{B} \to \mathbb{Z}$ — se denomina **función estructurante** o **elemento estructurante no plano**.

Desarrollada por Matheron y Serra en la década de 1960 para imágenes binarias y posteriormente extendida a tonos de gris, la morfología matemática fundamenta operadores como el gradiente morfológico, el *top-hat*, el *watershed* y la transformada de distancia, todos derivados de dos primitivos: la **erosión** y la **dilatación** [Matheron (1975); Serra (1982)].

4.3.1 Erosión y Dilatación

Los dos operadores primitivos se definen de manera unificada para imágenes en tonos de gris ($f : \mathbb{E} \to \mathbb{Z}$) y, por restricción al dominio $\{0, 1\}$, también para imágenes binarias.

4.3.1.1 Erosión

La **Erosión** de una imagen f por una función estructurante $b : \mathbb{B} \to \mathbb{Z}$ se define formalmente por:

$$\varepsilon_b(f)(x) = (f \ominus b)(x) = \min_{z \in \mathbb{B}} \{\, f(x+z) - b(z) \,\}, \quad \forall\, x \in \mathbb{E} \tag{4.3}$$

En la práctica, la erosión sustituye la intensidad del píxel x por el valor mínimo resultante de la diferencia entre la imagen y el elemento estructurante en la vecindad definida por el dominio $\mathbb{B}$. Los valores positivos en los pesos de $b(z)$ fuerzan el resultado local hacia abajo, "excavando" el relieve de la imagen más profundamente e intensificando la erosión.

En el caso **plano** (donde los pesos son nulos dentro del dominio, es decir, $b \equiv 0$), la expresión se simplifica al mínimo local puro:

$$\varepsilon_B(f)(x) = \min\{\, f(y) : y \in \mathbb{B}_x \,\}$$

En imágenes binarias, esta operación equivale a exigir que el conjunto $\mathbb{B}$, trasladado a la coordenada x, esté **completamente contenido** en el objeto A:

$$A \ominus \mathbb{B} = \{\, z \in \mathbb{E} \mid \mathbb{B}_z \subseteq A \,\}$$

Efecto Visual: *Encoge* objetos y estructuras claras, eliminando protuberancias, picos brillantes o ruidos que sean geométricamente menores que el dominio $\mathbb{B}$.

4.3.1.2 Implementación de la erosión

La versión didáctica `mm::ero0` implementa el caso particular de erosión con **elemento estructurante plano**. Para cada píxel (y, x), la función recorre los vecinos espaciales permitidos por B y almacena el menor valor encontrado en la imagen de entrada f, reproduciendo directamente la operación de mínimo local descrita en la Ecuación 4.3 para $b \equiv 0$.

Observe que los valores de los vecinos siempre se leen de forma estática de la imagen original f; la matriz de salida g se utiliza exclusivamente para registrar el mínimo acumulado de la vecindad actual. De esta forma, el resultado final es invariante respecto al orden de barrido de los píxeles (ya sea por filas o columnas).

La función auxiliar `_viz` calcula las coordenadas de los vecinos válidos dentro de los límites físicos de la imagen. En los bordes, la inicialización del acumulador en `255` emula con exactitud el relleno por elemento neutro exigido por la teoría. Por su parte, la función de interfaz `mm::ero` recurre a la implementación nativa y optimizada de OpenCV (`mm::ero`) cuando el elemento estructurante es plano, cambiando a la rutina general `mm::ero1` en caso de que el elemento posea pesos topográficos.

El ejemplo computacional siguiente ilustra la aplicación de un elemento estructurante en cruz (`mm::secross()`) destacado en la Figura 4.3, comparando la ejecución de la variante didáctica en bucle (`mm::ero0`) con el motor computacional de 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)")
```

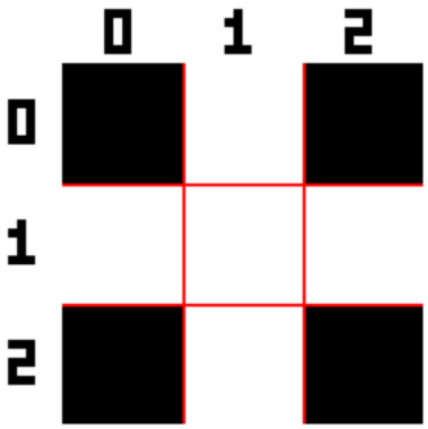


Figura 4.3: Elemento estruturante em formato de cruz (B_{cruz}) utilizado para conectividade-4.

```
# La morph.hpp es header-only; a continuación, el helper _viz y el cuerpo de mm::ero0().
import re, pathlib
```

```
hpp = pathlib.Path("morph.hpp").read_text()
for nome, pat in [("_viz", r"template <class F>\ninline void _viz\(.*?\n\}"),
                  ("ero0", r"inline Image ero0\(.*?\n\}")]:
    m = re.search(pat, hpp, re.S)
    print(f"// ---- mm::{nome} ----")
    print(m.group(0) if m else f"({nome} não encontrada)")
    print()
```

```
// ---- mm::_viz ----
template <class F>
inline void _viz(const Image& f, const SE& B, int y, int x, F&& cb) {
    double oh = -B.h / 2.0 + 0.5;
    double ow = -B.w / 2.0 + 0.5;
    for (int by = 0; by < B.h; ++by)
        for (int bx = 0; bx < B.w; ++bx) {
            int vy = (int)(y + by + oh);
            int vx = (int)(x + bx + ow);
            if (vy >= 0 && vy < f.h && vx >= 0 && vx < f.w)
                cb(vy, vx, B.at(by, bx));
        }
}

// ---- mm::ero0 ----
inline Image ero0(const Image& f, SE Bc = SE::box(3)) {
    _require_gray(f, "ero0");
    Image g(f.h, f.w, 1);
    for (int y = 0; y < f.h; ++y)
        for (int x = 0; x < f.w; ++x) {
            int mn = 255;
            _viz(f, Bc, y, x, [&](int vy, int vx, int bv) {
                if (bv != 0 && (int)f.at(vy, vx) < mn) mn = f.at(vy, vx);
            });
            g.at(y, x) = (unsigned char)mn;
        }
    return g;
}
```

4.3.1.3 Dilatación

La **Dilatación** de una imagen f mediante una función estructurante $b : \mathbb{B} \to \mathbb{Z}$ se define formalmente como:

$$\delta_b(f)(x) = (f \oplus b)(x) = \max_{z \in \mathbb{B}} \{ f(x - z) + b(z) \}, \quad \forall\, x \in \mathbb{E} \tag{4.4}$$

En la práctica, la dilatación reemplaza la intensidad del píxel x por el mayor valor resultante de la suma entre la imagen y el elemento estructurante en la vecindad definida. El argumento de inversión espacial $(x - z)$ indica que la dilatación evalúa implícitamente el elemento transpuesto (reflejado) $\hat{b}$, propiedad fundamental para asegurar la dualidad matemática respecto a la erosión.

En el caso **plano** (donde los pesos son nulos dentro del dominio, es decir, $b \equiv 0$), la expresión se reduce al máximo local puro:

$$\delta_B(f)(x) = \max\{ f(y) : y \in \mathbb{B}_x \}$$

En imágenes binarias, esta operación equivale a exigir que el conjunto reflejado $\hat{\mathbb{B}}$, trasladado a la coordenada x, tenga una **intersección no vacía** con el objeto A:

$$A \oplus \mathbb{B} = \{ z \in \mathbb{E} \mid \hat{\mathbb{B}}_z \cap A \neq \emptyset \}$$

Efecto Visual: *Expande* las estructuras claras de la imagen, aumentando el relleno de objetos, conectando componentes cercanos y eliminando canales, fosas oscuras o valles que sean geométricamente menores que el dominio $\mathbb{B}$.

4.3.1.4 Implementación de la dilatación

La versión didáctica `mm::dil0` implementa el caso particular de dilatación con **elemento estructurante plano**. Para cada píxel (y, x), la función recorre los vecinos espaciales permitidos por B y almacena el mayor valor encontrado en la imagen de entrada f, reproduciendo directamente la operación de máximo local para $b \equiv 0$.

Antes de iniciar el barrido espacial, el elemento estructurante sufre una reflexión geométrica mediante la instrucción `np.flip(Bc)` para construir explícitamente la matriz transpuesta $\hat{B}$ exigida por la teoría. En máscaras perfectamente simétricas (como cruces, cuadrados y discos centrados en el origen), esta reflexión no altera la disposición de los píxeles; no obstante, para elementos asimétricos, tal etapa es estrictamente necesaria para garantizar la equivalencia con las definiciones formales y salvaguardar las leyes de dualidad.

Así como se verificó en el operador de erosión, los valores de los vecinos siempre se leen de forma estática a partir de la matriz original f, mientras que la matriz de salida g actúa puramente como el registrador del máximo acumulado de la vecindad. En las fronteras de la imagen, la inicialización del acumulador en `0` emula con exactitud el relleno externo por elemento neutro ($-\infty$, o cero en representaciones de 8 bits), garantizando que los bordes físicos de la escena sean dilatados en perfecta conformidad con el estándar adoptado por OpenCV.

El ejemplo computacional siguiente ilustra la aplicación práctica de un elemento en cruz (`mm::secross()`), validando la consistencia entre la lógica en bucles (`mm::dil0`) y el método nativo industrial (`mm::dil`).

```python
# Cuerpo de mm::dil0() en morph.hpp (refleja el SE antes de barrer, como 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;
}
```

Nota: El Enfrentamiento de los Signos ($f(x+z)$ vs $f(x-z)$)

Compare las definiciones formales de la erosión (Ecuación 4.3) y de la dilatación (Ecuación 4.4). Considere un elemento estructurante asimétrico a la derecha $\mathbb{B} = \{0, 1\}$ (origen y un píxel a la derecha) aplicado en la posición $x = 10$.

1. **En la Erosión** (Ecuación 4.3):

$$\min\{f(x+z) - b(z)\}$$

- $z = 0 \Rightarrow f(10+0) = \mathbf{f(10)}$

- $z = 1 \Rightarrow f(10 + 1) = \mathbf{f(11)}$

El operador consulta el píxel actual (10) y el píxel a la derecha (11), preservando la orientación original de $\mathbb{B}$.

2. **En la Dilatación** (Ecuación 4.4):

$$\max\{f(x - z) + b(z)\}$$

- $z = 0 \Rightarrow f(10 - 0) = \mathbf{f(10)}$
- $z = 1 \Rightarrow f(10 - 1) = \mathbf{f(9)}$

Debido al signo negativo $(-z)$, avanzar en el elemento estructurante corresponde a retroceder en la imagen, haciendo que la dilatación consulte el píxel a la izquierda (9).

La función `_viz`, utilizada en `morph.py`, genera vecinos mediante desplazamientos aditivos de la forma $x + z$. Por ese motivo, la implementación de `mm::dil0` refleja previamente el elemento estructurante mediante `np.flip(B)`. Tras la reflexión, el barrido basado en $x + z$ pasa a acceder exactamente a los mismos puntos definidos por la expresión teórica $f(x - z)$ de la dilatación en Ecuación 4.4.

Para elementos estructurantes simétricos (como discos, cuadrados y cruces centradas), la reflexión no altera la máscara. En cambio, para elementos asimétricos, esta etapa es indispensable para que la implementación reproduzca correctamente la definición matemática de la dilatación y preserve la dualidad erosión–dilatación.

i Dualidad erosión–dilatación

La erosión y la dilatación son **duales por complemento**. Esto significa que un operador puede obtenerse completamente a partir del otro, siempre que se actúe sobre el complemento de la imagen utilizando el elemento estructurante reflejado $\hat{B}$:

$$(A \ominus B)^c = A^c \oplus \hat{B} \quad \Longleftrightarrow \quad A \ominus B = (A^c \oplus \hat{B})^c$$

De manera análoga, la **dilatación también puede obtenerse a partir de la erosión**:

$$(A \oplus B)^c = A^c \ominus \hat{B} \quad \Longleftrightarrow \quad A \oplus B = (A^c \ominus \hat{B})^c$$

En términos prácticos, la erosión de un objeto puede obtenerse mediante la dilatación de su complemento, seguida de la complementación del resultado (y viceversa). En la implementación del paquete `morph.py`, las versiones didácticas `mm::ero0` y `mm::dil0` hacen explícita esta estructura mediante bucles (*loops*), mientras que `mm::ero` y `mm::dil` delegan las operaciones a OpenCV buscando una mayor eficiencia computacional.

i Condiciones de contorno e imágenes finitas

En la morfología matemática clásica, definida sobre un dominio infinito (típicamente $\mathbb{Z}^2$), esta dualidad es exacta. En imágenes digitales, sin embargo, se trabaja con matrices finitas, y el resultado pasa a depender de la forma en que se tratan los píxeles ubicados fuera de la imagen.

Para que las identidades de dualidad permanezcan válidas, el complemento debe definirse con respecto al mismo universo y las condiciones de contorno adoptadas para la erosión y para la dilatación deben ser complementarias entre sí. Por ejemplo, si la erosión asume que los píxeles externos pertenecen al objeto (255), entonces la dilatación aplicada al complemento debe asumir que esos mismos píxeles externos pertenecen al fondo (0).

Cuando se utilizan diferentes estrategias de relleno (replicación, reflexión, valor constante, etc.), la dualidad teórica puede dejar de satisfacerse exactamente en las regiones cercanas a los bordes de la imagen.

Para ilustrar numéricamente los operadores morfológicos y la dualidad erosión–dilatación, la Figura 4.4 presenta una imagen binaria de 10×10 procesada con un elemento estructurante en forma de “L”. En

la implementación de `morph.py`, el origen de B se fija en el centro geométrico de la máscara —posición $(1, 1)$ para un *kernel* de 3×3— y debe corresponder a un elemento activo para que la erosión se comporte correctamente (según lo discutido anteriormente). El elemento estructurante B_L definido a continuación satisface esa condición. La Figura 4.5 complementa el análisis con un simulador interactivo de la erosión, permitiendo visualizar el desplazamiento del elemento estructurante sobre la imagen e identificar las posiciones en las que este permanece completamente contenido en el objeto.

```
%%writefile tmp/fig_04_ero_dil_didatico.cpp
#define MM_OUT "tmp/fig_04_ero_dil_didatico.png"
// Compile with: g++ -std=c++17 -O2 snippet.cpp -o snippet $(pkg-config --cflags --libs
opencv4) -DMM_USE_OPENCV -I../morph/cpp && ./snippet
#include "morph.hpp"
#include <opencv2/opencv.hpp>
#include <iostream>
#include <vector>
#include <string>
#include <filesystem>

int main() {
    //| 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

    // A = np.array([...], dtype=np.uint8) * 255
    mm::Image A(10, 10, 1);
    int A_data[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) = A_data[y][x] * 255;
        }
    }

    // B_L = np.array([[1,0,0],[1,1,0],[1,1,0]], dtype=np.uint8)   # 'L', origem no centro
    mm::SE B_L{{1,0,0},{1,1,0},{1,1,0}};
    // B_hat = np.array([[0,1,1],[0,1,1],[0,0,1]], dtype=np.uint8)   # B_L girado 180° (B)
    mm::SE B_hat{{0,1,1},{0,1,1},{0,0,1}};

    std::cout << "Elemento estruturante B_L:\n";
    // B_L is an SE; convert to Image for drawing
    mm::Image B_L_img = mm::sebox(0); // placeholder, will fix below
    B_L_img = mm::dil0(mm::Image(3, 3, 1), B_L); // get binary SE shape
    std::cout << mm::drawImg(B_L_img);
    std::cout << "Elemento estruturante refletido B:\n";
    mm::Image B_hat_img = mm::dil0(mm::Image(3, 3, 1), B_hat); // get binary SE shape
    std::cout << mm::drawImg(B_hat_img);

    mm::Image img_ero0 = mm::ero0(A, B_L);
```

```
    // Dualidade: (A ⊖ B)ᶜ == Aᶜ ⊕ 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);

    // Check if ero_c and dil_Ac are equal
    bool equal = true;
    for (int i = 0; i < (int)ero_c.data.size() && equal; i++) {
        if (ero_c.data[i] != dil_Ac.data[i]) equal = false;
    }
    std::cout << "Dualidade (A ⊖ B)ᶜ == Aᶜ ⊕ B : " << (equal ? "True" : "False") << "\n";

    mm::show(
        std::vector<mm::Image>{A, A_c, img_ero0, ero_c, dil_Ac},
        MM_OUT,
        std::vector<std::string>{"A", "Aᶜ", "A ⊖ B", "(A ⊖ B)ᶜ", "Aᶜ ⊕ 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:
0 0 0
0 0 0
0 0 0
Elemento estruturante refletido B:
0 0 0
0 0 0
0 0 0
Dualidade (A ⊖ B)ᶜ == Aᶜ ⊕ B : True
[1] Aᶜ
[2] A
[3] A ⊖ B
[4] (A ⊖ B)ᶜ
[5] Aᶜ ⊕ 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)")
```

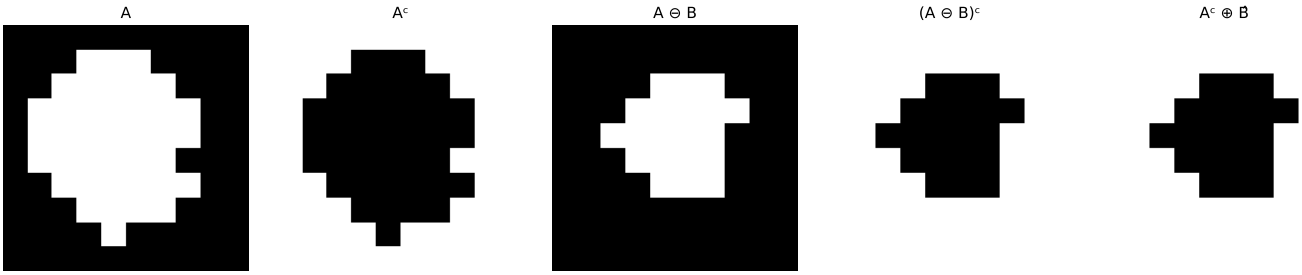


Figura 4.4: Erosão e dilatação em imagem binária 10×10 com elemento estruturante 'L' assimétrico 3×3. Validação da dualidade erosão–dilatação.

4.3.2 Apertura y Cierre

Combinando erosión y dilatación se obtienen dos operadores de gran utilidad práctica: la **apertura** y el **cierre**, definidos por las Ecuaciones Ecuación 4.5 y Ecuación 4.6. Sus principales efectos se resumen en la Tabla 4.1.

Apertura (*opening*) — erosión seguida de dilatación por el mismo B:

$$A \circ B = (A \ominus B) \oplus B \tag{4.5}$$

Cierre (*closing*) — dilatación seguida de erosión por el mismo B:

$$A \bullet B = (A \oplus B) \ominus B \tag{4.6}$$

En la práctica, `mm::open` y `mm::close` aplican el mismo elemento estructurante en las dos etapas. Para elementos estructurantes simétricos (los más comunes), esta implementación coincide con la definición matemática presentada anteriormente.

Tabla 4.1: Propiedades de apertura y cierre.

Operador	Secuencia	Efecto principal
Apertura $A \circ B$	erosión → dilatación	Elimina estructuras incapaces de contener el elemento estructurante; suaviza contornos externos

Operador	Secuencia	Efecto principal
Cierre $A \bullet B$	dilatación → erosión	Rellena agujeros más pequeños que B; suaviza contornos internos

Propiedad importante: ambos son **idempotentes**. Por ejemplo,

$$(A \circ B) \circ B = A \circ B,$$

es decir, después de la primera aplicación, nuevas aplicaciones del mismo operador no alteran más el resultado.

4.3.2.1 Implementación de la apertura y el cierre

A diferencia de la erosión y la dilatación, la apertura y el cierre no introducen nuevos mecanismos computacionales. Ambos se obtienen mediante la composición secuencial de los operadores primitivos ya presentados:

```
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 función `mm::open` delega la operación a `mm::open(f, B)`, mientras que `mm::close` utiliza `mm::close(f, B)`, produciendo el mismo resultado de forma más eficiente.

La apertura hereda de la erosión la capacidad de eliminar estructuras más pequeñas que el elemento estructurante y de la dilatación la restauración parcial de las regiones preservadas. El cierre realiza el proceso inverso: primero expande los objetos y luego restaura sus dimensiones originales, rellenando huecos y agujeros más pequeños que el elemento estructurante.

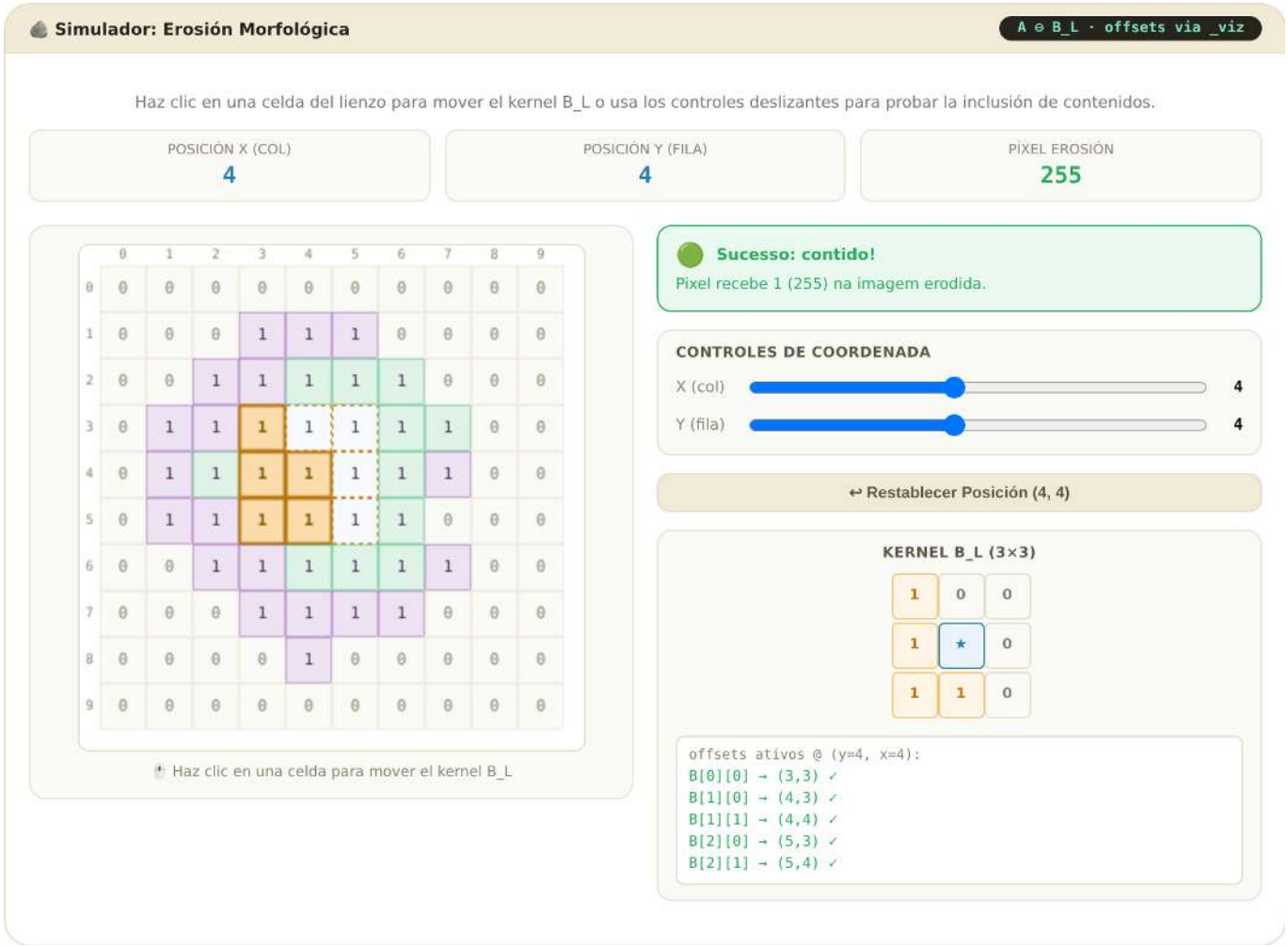
4.3.2.2 Filtro Secuencial Alternado

En la práctica, la apertura y el cierre se aplican frecuentemente en secuencia para eliminar simultáneamente el ruido externo y rellenar los huecos internos. La función `mm::asf` (*Filtro Secuencial Alternado*) generaliza esta estrategia al aplicar aperturas y cierres alternadamente con elementos estructurantes progresivamente más grandes. Las secuencias disponibles se presentan en la Tabla 4.2.

Tabla 4.2: Secuencias del filtro secuencial alternado mm.asf.

Secuencia	Orden	Uso típico
`'OC'`	apertura → cierre	elimina el ruido externo antes de rellenar pequeños huecos
`'CO'`	cierre → apertura	rellena pequeños huecos antes de eliminar el ruido externo
`'OCO'`	apertura → cierre → apertura	enfatiza la eliminación del ruido externo
`'COC'`	cierre → apertura → cierre	enfatiza el relleno de huecos y lagunas

El parámetro `n` controla el número de escalas utilizadas por el filtro. En cada iteración i, el elemento estructurante se amplía mediante la suma de Minkowski (`mm::sesum(b, i)`), produciendo una secuencia de filtros morfológicos cada vez más abarcativos. A diferencia de una única apertura o cierre con un elemento estructurante grande, el ASF realiza una suavización progresiva en múltiples escalas, preservando mejor la geometría de los objetos relevantes mientras elimina estructuras menores. La Figura 4.6 presenta un ejemplo de aplicación de la apertura, el cierre y el ASF en la imagen de las monedas.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.es/cap04/sim-04-erosao.html

Figura 4.5: Simulador: Erosión Morfológica (A B_L)

```
%%writefile tmp/fig_04_open_close.cpp
#define MM_OUT "tmp/fig_04_open_close.png"
#include "morph.hpp"
#include <iostream>
#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)")
```

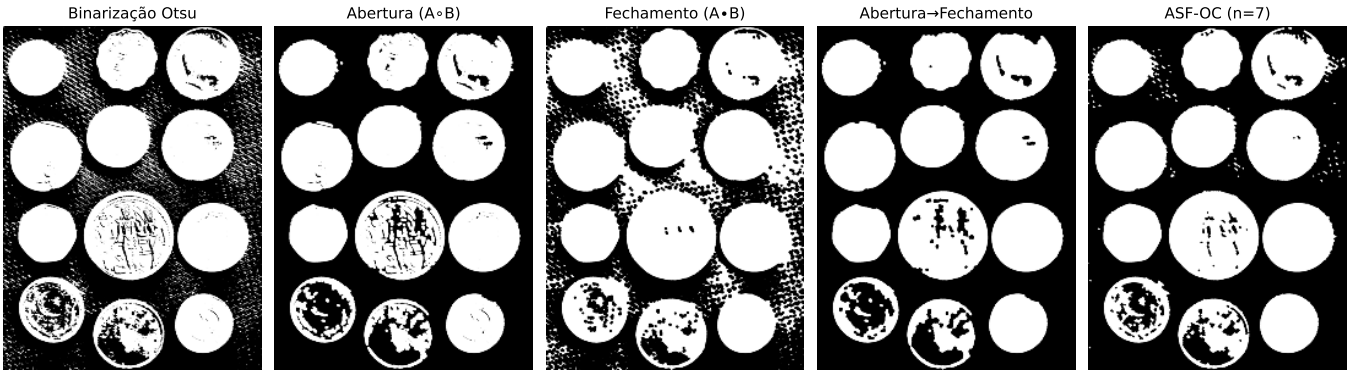


Figura 4.6: Abertura, fechamento, composição e filtro sequencial alternado aplicados à binarização Otsu das moedas (pré-processadas por realce de contraste). Elemento estruturante: disco 13×13.

4.3.3 Operadores Geodésicos

Los operadores geodésicos introducen una restricción adicional a los operadores morfológicos clásicos mediante una imagen de control denominada **máscara** g. En lugar de permitir que la erosión o la dilatación se propaguen libremente por la imagen, el resultado de cada iteración se limita punto a punto por los valores de la máscara, restringiendo la evolución de la operación a las regiones permitidas.

4.3.3.1 Dilatación Geodésica

La **dilatación geodésica** de una imagen marcador f bajo una imagen máscara g, utilizando un elemento estructurante plano b, se define por:

$$f \oplus_g b = (f \oplus b) \wedge g, \tag{4.7}$$

donde $\wedge$ representa el mínimo punto a punto.

En otras palabras, se realiza inicialmente una dilatación convencional sobre el marcador y, a continuación, el resultado se restringe mediante la máscara g. De esta manera, la propagación nunca puede superar las regiones permitidas por la máscara.

La formulación clásica de la dilatación geodésica presupone que el marcador esté contenido en la máscara, es decir, $f \leq g$, garantizando que la evolución de la operación permanezca siempre limitada por la máscara.

4.3.3.2 Implementación de la dilatación geodésica

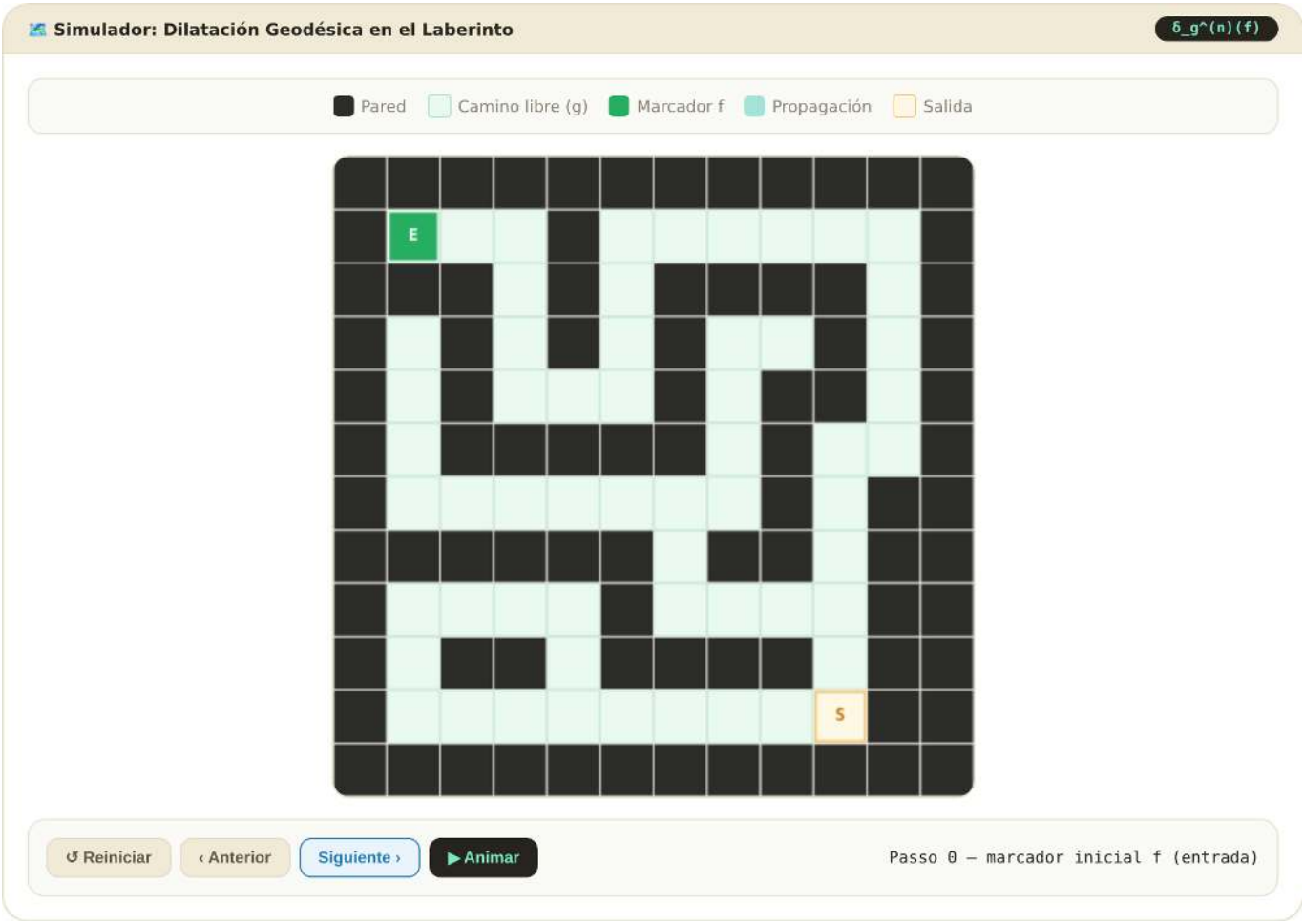
La función `mm::cdil` implementa directamente este operador y permite ejecutar múltiples iteraciones consecutivas:

```
def cdil(f, g, b=np.zeros((3,3),dtype='uint8'), n=1):
    """Dilatação geodésica do marcador f sob a máscara g."""
    y = f.copy()
    for _ in range(n):
        y = np.minimum(mm.dil(y, b), g)
    return y
```

La instrucción `np.minimum(mm::dil(y, b), g)` implementa exactamente la definición matemática de la dilatación geodésica, es decir, $(y \oplus b) \wedge g$.

Cuando $n = 1$, la función ejecuta una única dilatación geodésica. Para $n > 1$, el resultado de cada etapa se convierte en el marcador de la etapa siguiente, produciendo una propagación progresiva controlada por la máscara.

El simulador interactivo Figura 4.7 permite seguir, paso a paso, la propagación del marcador f a lo largo de los corredores del laberinto. En cada iteración de `mm::cdil`, el frente de dilatación avanza hacia las celdas vecinas libres —aquellas en las que $g = 1$—, mientras que las paredes ($g = 0$) permanecen intransitables. El número de pasos necesarios para que el marcador alcance la salida corresponde exactamente a la longitud geodésica del camino más corto dentro de la máscara, evidenciando la conexión directa entre la dilatación geodésica iterada y la noción de distancia en grafos.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.es/cap04/sim-04-cdil.html

Figura 4.7: Simulador interactivo de dilatación geodésica: el marcador f (verde) se propaga paso a paso por los caminos libres de la máscara g, sin atravesar paredes.

4.3.3.3 Erosión Geodésica

De forma dual, la **erosión geodésica** de una imagen marcador f bajo una imagen máscara g se define por:

$$f \ominus_g b = (f \ominus b) \vee g, \tag{4.8}$$

donde $\vee$ representa el operador de máximo punto a punto.

En este caso, la erosión convencional del marcador es seguida por una restricción inferior impuesta por la máscara. Así, ningún píxel del resultado puede asumir un valor inferior al correspondiente píxel de la máscara.

La formulación clásica de la erosión geodésica presupone la condición dual

$$f \geq g,$$

de modo que la máscara actúe como límite inferior durante todo el proceso.

4.3.3.4 Implementación de la erosión geodésica

La función estática `mm::cero` materializa este operador:

```
staticmethod
def cero(f, g, b=np.zeros((3,3),dtype='uint8'), n=1):
    """Erosión geodésica del marcador f bajo la máscara g."""
    y = f.copy()
    for _ in range(n):
        y = np.maximum(mm.ero(y, b), g)
    return y
```

La instrucción `np.maximum(mm::ero(y, b), g)` implementa directamente la expresión $(y \ominus b) \vee g$.

Al igual que en la dilatación geodésica, el parámetro n define cuántas erosiones geodésicas sucesivas se calcularán antes de devolver la imagen final.

> **i** Relación con la reconstrucción morfológica
>
> La reconstrucción morfológica presentada en la siguiente sección se obtiene mediante la aplicación iterativa de la dilatación geodésica (`mm::cdil`) hasta alcanzar un punto fijo, es decir, hasta que ninguna celda cambie de valor entre dos iteraciones consecutivas. En otras palabras, la reconstrucción consiste en una secuencia de dilataciones geodésicas sucesivas que se propagan dentro de la máscara hasta que no haya más alteraciones.
> De forma dual, también es posible definir reconstrucciones basadas en erosión geodésica mediante aplicaciones sucesivas de `mm::cero`.

4.3.3.5 Ejemplo: propagación en un laberinto mediante dualidad

La Figura 4.8 ilustra la resolución del problema de conectividad de un laberinto utilizando la dualidad morfológica por medio de las funciones `mm::cero` y `mm::suprec`.

En lugar de propagar un marcador por los corredores libres mediante dilataciones geodésicas, el problema se formula en el dominio complementario. Inicialmente, la máscara original se invierte,

$$g = 1 - g_{orig},$$

de modo que las paredes pasan a tomar el valor 1 y los corredores el valor 0. De manera análoga, el marcador se construye en ese mismo dominio complementario, conteniendo un único valor 0 en la posición de entrada del laberinto y valor 1 en los demás píxeles.

Utilizando el elemento estructurante en cruz (`mm::secross()`), la erosión geodésica actúa sobre el marcador complementado. En cada iteración de `mm::cero`, la región conectada al marcador inicial sufre erosiones sucesivas, mientras que la máscara impone un límite inferior que impide la propagación a través de las paredes del laberinto.

Las imágenes intermediarias muestran estados de la evolución después de diferentes números de iteraciones (`n=5`, `n=12` y `n=22`). El resultado final se obtiene mediante la reconstrucción geodésica por erosión (`mm::suprec`), que aplica erosiones geodésicas sucesivas hasta alcanzar un punto fijo, es decir, una situación en la que no se produce ninguna alteración adicional entre dos iteraciones consecutivas.

En el dominio complementario, la región reconstruida corresponde exactamente al conjunto de corredores conectados a la entrada del laberinto. Así, la conectividad entre la entrada y la salida puede determinarse directamente a partir de la imagen reconstruida.

```
%%writefile tmp/fig_04_cero_labirinto.cpp
#define MM_OUT "tmp/fig_04_cero_labirinto.png"
#include "morph.hpp"
#include <iostream>
#include <filesystem>

int main() {
    // Máscara já invertida (255 = parede, 0 = corredor)
    mm::Image g(10, 10);
    unsigned char 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)")
```

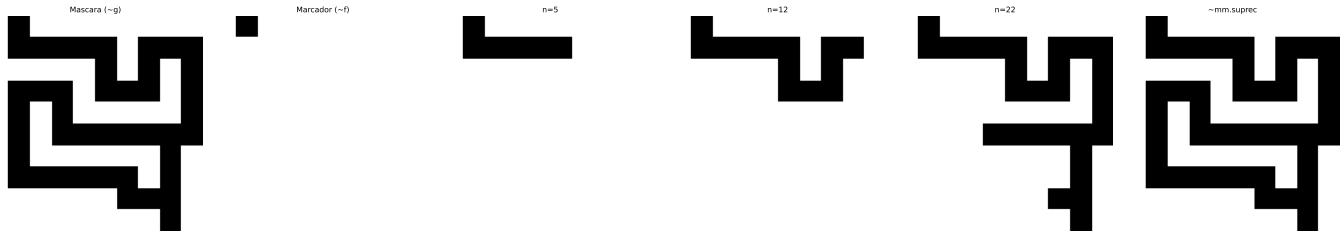


Figura 4.8: Resolução de labirinto no domínio complementar: com máscara e marcador invertidos, *mm::cero* e *mm::suprec* propagam a onda geodésica pelos corredores.

4.3.4 Reconstrucción Morfológica

La **reconstrucción morfológica** propaga una imagen **marcadora** f dentro de una imagen **máscara** g, garantizando que el resultado nunca supere los valores de intensidad impuestos por la máscara. El operador fundamental que posibilita esta propagación contenida es la **dilatación geodésica**, definida por la Ecuación 4.7.

La reconstrucción se obtiene mediante la aplicación iterativa de esta dilatación condicionada. Inicialmente, el marcador está limitado por la máscara para establecer el estado inicial:

$$X^{(0)} = f \wedge g,$$

y las iteraciones subsiguientes se definen de forma recursiva mediante:

$$X^{(k)} = (X^{(k-1)} \oplus b) \wedge g.$$

La secuencia crece de manera monótona hasta alcanzar un punto fijo, produciendo la **reconstrucción morfológica por dilatación** (también conocida en la literatura como *inf-reconstrucción*):

$$R_g^{\delta}(f) = \lim_{k\to\infty} X^{(k)} = X^{(k)} \quad \text{cuando} \quad X^{(k)} = X^{(k-1)}. \tag{4.9}$$

El ascenso iterativo se detiene tan pronto como se logra la estabilidad, es decir, cuando dos iteraciones consecutivas producen matrices con valores absolutamente idénticos.

4.3.4.1 Implementação da reconstrucción morfológica

La rutina didáctica `mm::infrec` implementa directamente el algoritmo iterativo de punto fijo. Inicialmente, el marcador efectivo inicial $X^{(0)}$ se determina mediante la operación `np.minimum(f, g)`. Para garantizar que el bucle de verificación ejecute la primera pasada sin disparar falsas convergencias prematuras, la variable de control de la iteración anterior (`y1`) se inicializa rellenada con un valor centinela fuera del dominio de los datos (o simplemente con una matriz que fuerce la primera ejecución).

```
def infrec(f, g, b=np.zeros((3,3), dtype='uint8')):
    """Inf-reconstrucción: dilata el marcador (f   g) hasta converger bajo la máscara g."""
    y = np.minimum(f, g)
    # Inicializa y1 con valores imposibles para forzar la entrada en el bucle
    y1 = np.full_like(f, 256, dtype=np.int16)
    while not np.array_equal(y, y1):
        y1 = y.copy()
        # Aplica la dilatación geodésica: (y   b)   g
        y = np.minimum(mm.dil(y, b), g)
    return y.astype('uint8')
```

En el interior del bucle `while`, la variable `y` almacena la estimación corriente de la reconstrucción $X^{(k)}$, mientras que `y1` preserva la imagen del estadio inmediatamente anterior $X^{(k-1)}$. La instrucción de control condicional `np.minimum(mm::dil(y, b), g)` traduce fielmente la dilatación geodésica teórica, donde la expansión morfológica convencional comandada por OpenCV se "poda" inmediatamente y queda limitada por las barreras de intensidad de la máscara g. El bucle cesa cuando no se registra ninguna modificación de píxel entre los pasos.

4.3.4.2 Ventajas de la reconstrucción morfológica

La reconstrucción morfológica es significativamente más robusta que la apertura convencional porque es capaz de eliminar estructuras no deseadas sin distorsionar ni alterar la morfología de los objetos que deben conservarse.

Mientras que la apertura clásica suaviza esquinas, elimina puntas y deforma contornos debido a la imposición geométrica rígida del elemento estructurante, la reconstrucción geodésica utiliza la máscara para recuperar con exactitud los límites y formatos originales de los objetos que poseen conectividad con el marcador original.

En términos intuitivos, el marcador actúa como una semilla de contagio que se expande progresivamente, pero solo transitando por las regiones permitidas por la máscara. Los componentes que no poseen ninguna intersección con el marcador jamás serán reconstruidos (siendo eliminados), mientras que los componentes tocados por la semilla se expanden hasta restaurar integralmente su geometría original.

Este comportamiento discriminatorio y conservativo se ilustra en la Figura 4.9, utilizando el elemento estructurante en cruz presentado en la Figura 4.3..

```
%%writefile tmp/fig_04_reconstrucao_didatica.cpp
#define MM_OUT "tmp/fig_04_reconstrucao_didatica.png"
// g++ -std=c++17 -DMORPH_USE_OPENCV snippet.cpp -o snippet $(pkg-config --cflags --libs
opencv4) -lm

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

int main() {
    //| 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

    mm::Image g(10, 10);
    // Initialize g with the 0/1 pattern scaled by 255
    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(0));          // 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);

    // Check if img_reconstruida equals iteracoes[-1]
    bool igual = true;
    if (img_reconstruida.h != iteracoes.back().h || img_reconstruida.w != iteracoes.back().w)
    {
        igual = false;
    } else {
        for (size_t i = 0; i < img_reconstruida.data.size(); i++) {
            if (img_reconstruida.data[i] != iteracoes.back().data[i]) {
                igual = false;
                break;
            }
        }
    }
    std::cout << "  Estabilidade na iteração 5: " << (igual ? "True" : "False") << std::endl;

    std::vector<mm::Image> todas_imagens;
    todas_imagens.push_back(g);
    todas_imagens.push_back(f);
    for (auto& img : iteracoes) {
        todas_imagens.push_back(img);
    }
    todas_imagens.push_back(img_reconstruida);

    std::vector<std::string> todos_titulos = {"Máscara (g)", "Marcador (f)"};
    for (auto& t : titulos) {
        todos_titulos.push_back(t);
    }
    todos_titulos.push_back("Reconstrução R_g(f)");

    mm::show(todas_imagens, MM_OUT, todos_titulos, 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)")
```

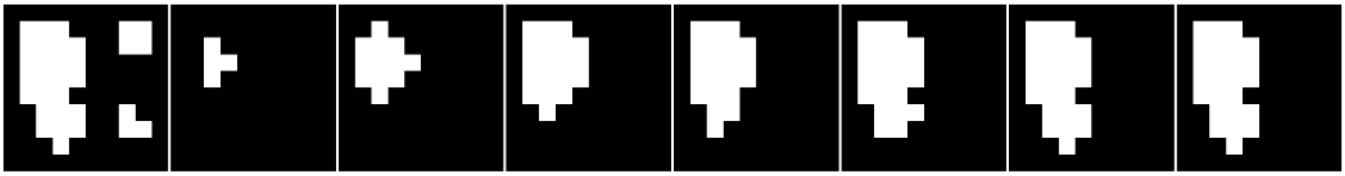


Figura 4.9: *Pipeline* de Reconstrução Morfológica por Dilatação Condicionada: a máscara contém dois objetos, o marcador isola apenas o núcleo do objeto principal, e as iterações reconstroem sua forma exata até a convergência.

4.3.5 Relleno de Agujeros y Eliminación de Bordes

Dos operadores basados en reconstrucción morfológica completan el *pipeline* de limpieza binaria. Sus características principales se resumen en la Tabla 4.3.

Relleno de agujeros (`mm::clohole`) elimina cavidades completamente rodeadas por el objeto, independientemente del tamaño, sin alterar los contornos externos. El procedimiento actúa sobre el complemento de la imagen, utilizando como marcador una restricción del marco (*frame*) al fondo:

$$\text{clohole}(f) = \left(R^{\delta}_{f^c}(\text{frame}(f) \wedge f^c)\right)^c \tag{4.10}$$

En términos operativos, primero se reconstruye el fondo externo y, a continuación, se aplica la complementación para recuperar los objetos con los agujeros rellenos.

Eliminación de objetos de borde (`mm::edgeoff`) elimina todos los objetos que tocan el borde de la imagen, preservando únicamente los componentes totalmente internos. El marcador se obtiene mediante la intersección entre el marco (*frame*) y los objetos de la imagen:

$$\text{edgeoff}(f) = f \setminus R^{\delta}_{f}(\text{frame}(f) \wedge f) \tag{4.11}$$

Tabla 4.3: Comparación entre los operadores *clohole* y *edgeoff*.

Operador	Marcador	Máscara	Efecto
`mm::clohole`	*frame* restringido al fondo (f^c)	f^c	Rellena agujeros internos
`mm::edgeoff`	*frame* restringido al objeto (f)	f	Elimina objetos conectados al borde

La evolución paso a paso de estas transformaciones geodésicas puede seguirse en las figuras siguientes. La Figura 4.10 ilustra el mecanismo de inundación controlada del operador `mm::clohole`, en el cual la reconstrucción se produce a partir del fondo externo e impide la propagación hacia regiones internas no conectadas al exterior, dando como resultado el relleno consistente de las cavidades internas. En cambio, la Figura 4.11 detalla la dinámica del operador `mm::edgeoff`, en la que solo los componentes conectados al borde se reconstruyen y posteriormente se eliminan, preservando exclusivamente los objetos totalmente contenidos en el interior de la imagen.

La conectividad de la propagación geodésica se controla mediante el elemento estructurante: `mm::sebox()` (vecindad de 8) incluye conexiones diagonales, mientras que `mm::secross()` (vecindad de 4) las excluye. En consecuencia, la elección del elemento estructurante afecta a qué componentes se alcanzan mediante la reconstrucción y, por tanto, cuáles se preservarán o eliminarán.

4.3.5.1 Conformidad con la implementación

Las definiciones anteriores están directamente alineadas con la implementación en `morph.py`, reproducida a continuación:

```
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))
```

Estas implementaciones dejan explícito que ambos operadores son instancias directas de reconstrucción morfológica por dilatación geodésica con `mm::infrec`, diferenciándose únicamente en la elección del marcador y de la máscara: `clohole` actúa sobre el complemento de la imagen, mientras que `edgeoff` actúa directamente en el dominio de los objetos.

```
%%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);
    // Matriz binária definida manualmente
    int dados[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) = dados[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));

    // Verificação
    mm::Image esperado = mm::clohole(f);
    bool igual = true;
    for (int y = 0; y < f.h && igual; y++)
        for (int x = 0; x < f.w; x++)
            if (img_clohole.at(y,x) != esperado.at(y,x)) { igual = false; break; }
    std::cout << "  Validação clohole: " << (igual ? "true" : "false") << "\n";

    std::vector<mm::Image> todas = {f, marcador_ch};
    todas.insert(todas.end(), iteracoes.begin(), iteracoes.end());
    todas.push_back(img_clohole);

    std::vector<std::string> nomes = {"f original", "Marcador (borda)"};
    nomes.insert(nomes.end(), titulos.begin(), titulos.end());
    nomes.push_back("clohole(f)");

    mm::show(todas, MM_OUT, nomes, 8);
    return 0;
}
```

```
Overwriting tmp/fig_04_clohole_didatico.cpp
```

```
!g++ -I. -std=c++17 -DMM_USE_OPENCV -I/usr/include/opencv4 tmp/fig_04_clohole_didatico.cpp -o
tmp/fig_04_clohole_didatico -lopencv_imgproc -lopencv_imgcodecs -lopencv_core \
  && ./tmp/fig_04_clohole_didatico \
  && test -f "tmp/fig_04_clohole_didatico.png" \
  || echo "  mm::show não gravou tmp/fig_04_clohole_didatico.png"
```

```
  Validação clohole: true
[1] f original
[2] Marcador (borda)
[3] Iter. (n=1)
[4] Iter. (n=2)
[5] Iter. (n=3)
[6] Iter. (n=4)
[7] Iter. (n=5)
[8] clohole(f)
```

```
try:
    mm.show(mm.read("tmp/fig_04_clohole_didatico.png"), figsize=(18, 3))
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + "
    tmp/fig_04_clohole_didatico.png (ver a versao Python)")
```

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"
//| label: fig-04-edgeoff-didatico
//| fig-cap: "*Pipeline* de eliminação de estruturas de borda (*edgeoff*): o marcador captura
as raízes conectadas às extremidades, a reconstrução delimita esses elementos e a subtração
preserva só os objetos totalmente internos."
//| echo: true
//| output: true

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

int main() {
    mm::Image f(10, 10);
    int 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) = data[y][x] * 255;

    mm::Image B_box = mm::sebox();
    mm::Image marcador_eo = mm::band(mm::frame(f, 1), f);   // borda   f

    std::vector<mm::Image> iteracoes;
    std::vector<std::string> titulos;
    mm::Image img_atual = marcador_eo;
    for (int i = 1; i <= 5; i++) {
        img_atual = mm::cdil(img_atual, f, B_box);
        iteracoes.push_back(img_atual);
        titulos.push_back("Iter. (n=" + std::to_string(i) + ")");
    }
```

```
    mm::Image img_edgeoff = mm::edgeoff(f, B_box);

    std::vector<mm::Image> imgs = {f, marcador_eo};
    imgs.insert(imgs.end(), iteracoes.begin(), iteracoes.end());
    imgs.push_back(img_edgeoff);

    std::vector<std::string> titles = {"f original", "Marcador (borda)"};
    titles.insert(titles.end(), titulos.begin(), titulos.end());
    titles.push_back("edgeoff(f)");

    mm::show(imgs, MM_OUT, titles, 8);
    return 0;
}
```

```
Overwriting tmp/fig_04_edgeoff_didatico.cpp
```

```
!g++ -I. -std=c++17 -DMM_USE_OPENCV -I/usr/include/opencv4 tmp/fig_04_edgeoff_didatico.cpp -o
tmp/fig_04_edgeoff_didatico -lopencv_imgproc -lopencv_imgcodecs -lopencv_core \
  && ./tmp/fig_04_edgeoff_didatico \
  && test -f "tmp/fig_04_edgeoff_didatico.png" \
  || echo "  mm::show não gravou tmp/fig_04_edgeoff_didatico.png"
```

```
[1] f original
[2] Marcador (borda)
[3] Iter. (n=1)
[4] Iter. (n=2)
[5] Iter. (n=3)
[6] Iter. (n=4)
[7] Iter. (n=5)
[8] edgeoff(f)
```

```
try:
    mm.show(mm.read("tmp/fig_04_edgeoff_didatico.png"), figsize=(18, 3))
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + "
    tmp/fig_04_edgeoff_didatico.png (ver a versao Python)")
```

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* de Limpieza Binaria con CLAHE

Con base en el análisis anterior —en el cual el CLAHE produjo el mayor valor de la varianza interclases ($\sigma_B^2 \approx 2{,}47 \times 10^3$), ver Figura 4.2, y el operador `mm::clohole` mostró ser eficaz en el relleno de las cavidades internas—, el *pipeline* final de segmentación, ilustrado en la Figura 4.12, se estructura mediante el siguiente flujo computacional:

$$\text{gray} \xrightarrow{\text{CLAHE}} \xrightarrow{\text{Otsu}} \xrightarrow{\text{open}} \xrightarrow{\text{clohole}} \xrightarrow{\text{open}} \xrightarrow{\text{edgeoff}} \text{segmentación}$$

Tras la etapa de `mm::clohole`, se aplica una segunda apertura morfológica con un elemento estructurante mayor (`mm::sedisk(33)`, disco de diámetro 33). Esta operación elimina pequeñas regiones residuales y artefactos que podrían haber permanecido después de la segmentación. En particular, el relleno geodésico puede transformar pequeñas cavidades aisladas en componentes conectados al objeto, haciendo conveniente una etapa adicional de filtrado basada en tamaño. El diámetro se eligió de modo que las monedas sigan siendo capaces de contener el elemento estructurante, mientras que los componentes significativamente más pequeños sean eliminados.

En esta imagen, ninguna moneda está conectada al borde de la matriz. En consecuencia, la aplicación de `mm::edgeoff` no altera el resultado obtenido tras la segunda apertura. Aun así, esta etapa se mantiene en el *pipeline* por robustez, pues en otras imágenes pueden existir objetos parcialmente visibles o conectados a los bordes, que deben eliminarse antes de la etapa de análisis.

¿Por qué la apertura después del clohole?

El operador `mm::clohole` rellena todas las cavidades cerradas presentes en los objetos segmentados. En algunas situaciones, pequeñas regiones no deseadas pueden permanecer después de esta etapa o volverse conectadas a los objetos principales. La apertura morfológica subsiguiente elimina componentes más pequeños que el elemento estructurante, preservando las monedas debido a su tamaño significativamente mayor.

```
%%writefile tmp/fig_04_pipeline_clahe.cpp
#define MM_OUT "tmp/fig_04_pipeline_clahe.png"
// Compile: g++ -std=c++17 -O2 -o program program.cpp -I/usr/include/opencv4 $(pkg-config
--cflags --libs opencv4) -DMM_USE_OPENCV -DMM_OUT="\"output.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_coins_gray = mm::_read_state("tmp/state/img_coins_gray_8.png");
// [pdi:state-io:end]

    mm::Image img_clahe0 = mm::clahe(img_coins_gray, 2.0, 8);
    mm::Image img_bin = mm::threshold(img_clahe0);
    mm::Image img_open = mm::open(img_bin, mm::sedisk(9));
    mm::Image img_hole = mm::clohole(img_open);
    mm::Image img_limpo = mm::open(img_hole, mm::sedisk(33));
    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)")
```

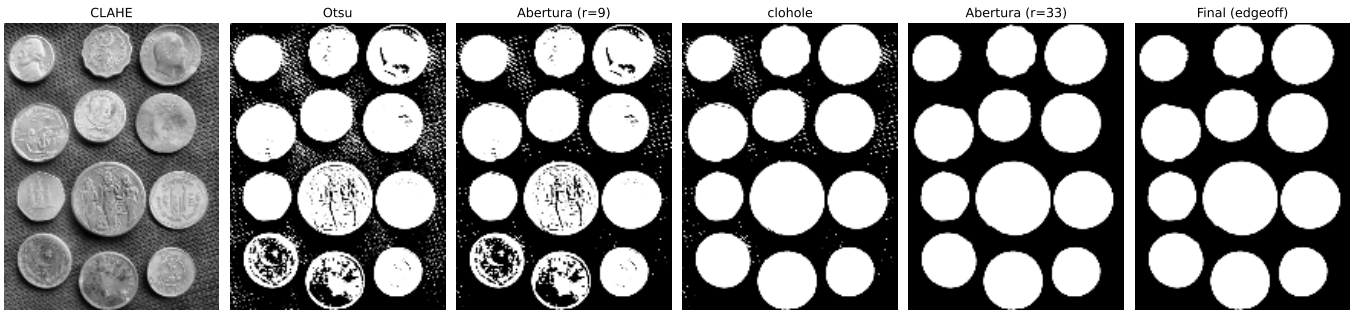


Figura 4.12: *Pipeline* completo de segmentação com CLAHE: Otsu → abertura (r=9) → clohole → abertura (r=33) → edgeoff.

4.3.7 Morfología en Tonos de Gris

Los operadores morfológicos se extienden naturalmente a imágenes en tonos de gris. En esta formulación, la erosión y la dilatación pasan a actuar directamente sobre los niveles de intensidad de la imagen. Para elementos estructurantes planos ($b \equiv 0$), la erosión corresponde al **mínimo local** y la dilatación al **máximo local** dentro de la vecindad definida por el elemento estructurante.

La interpretación intuitiva es sencilla: la erosión **oscurece** regiones al reemplazar cada píxel por el menor valor presente en su vecindad, mientras que la dilatación **aclara** regiones al utilizar el mayor valor disponible. La combinación de estos operadores permite construir transformaciones capaces de resaltar bordes, eliminar tendencias de iluminación y destacar estructuras locales.

Tres operadores derivados son especialmente útiles:

Gradiente morfológico — resalta bordes como la diferencia entre dilatación y erosión:

$$\text{grad}_B(f) = (f \oplus B) - (f \ominus B) \tag{4.12}$$

Top-hat — resalta estructuras brillantes más pequeñas que el elemento estructurante (diferencia entre la imagen original y su apertura):

$$\text{top-hat}_B(f) = f - (f \circ B) \tag{4.13}$$

Black-hat — resalta estructuras oscuras más pequeñas que el elemento estructurante (diferencia entre el cierre y la imagen original):

$$\text{black-hat}_B(f) = (f \bullet B) - f \tag{4.14}$$

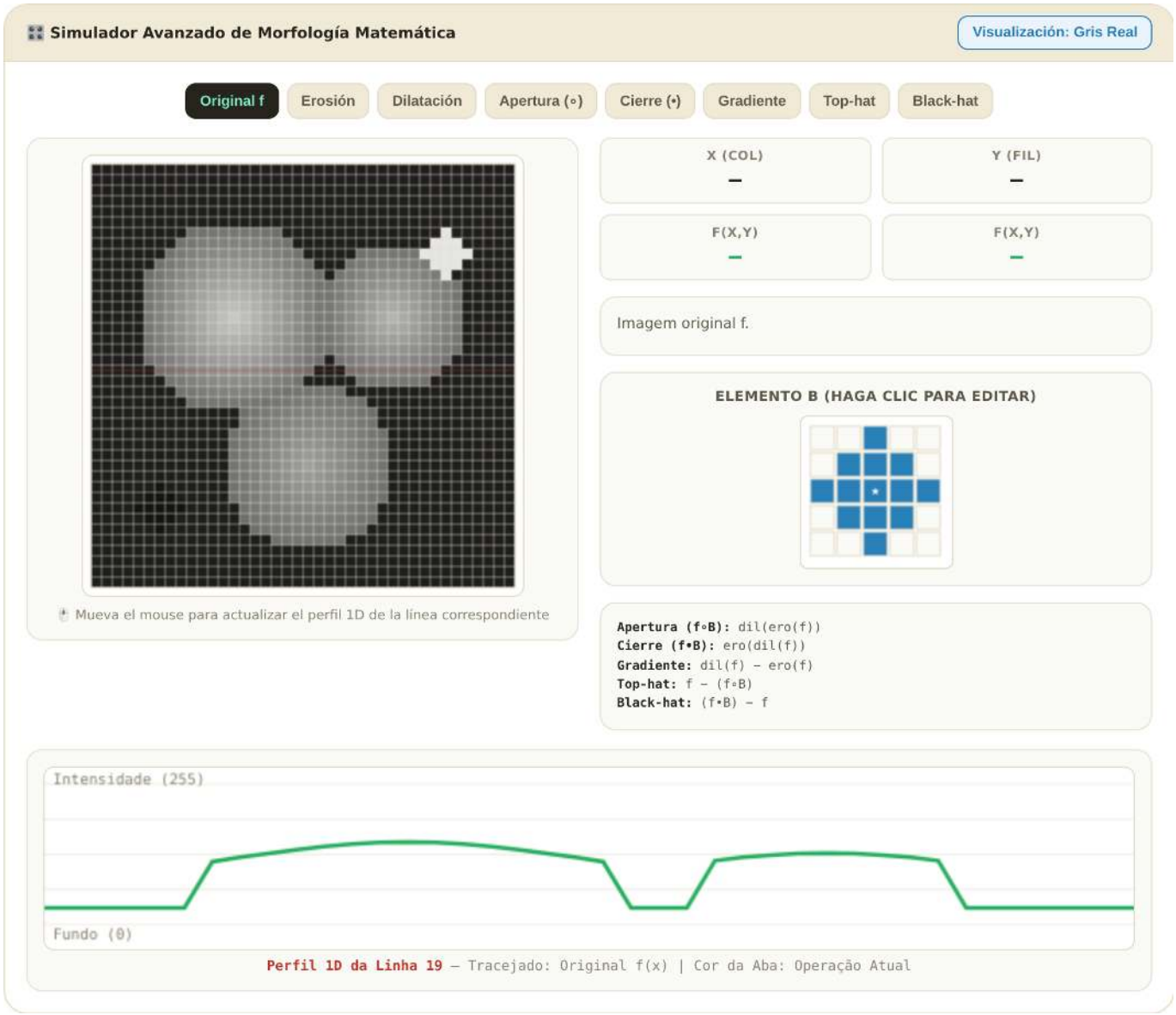
El *Top-hat* extrae detalles brillantes que no sobreviven a la apertura, mientras que el *Black-hat* evidencia detalles oscuros eliminados por el cierre. Por su parte, el gradiente morfológico resalta transiciones abruptas de intensidad, produciendo una representación similar a la de un detector de bordes.

Para comprender el mecanismo de estos operadores a nivel local, la Figura 4.13 presenta un simulador interactivo de morfología en tonos de gris. El simulador permite editar libremente el elemento estructurante, visualizar su desplazamiento sobre la imagen y acompañar simultáneamente el perfil unidimensional de las intensidades. De esta forma, se hace posible observar directamente cómo la erosión selecciona mínimos locales, cómo la dilatación selecciona máximos locales y cómo el gradiente morfológico emerge de la diferencia entre estos dos operadores.

El botón ubicado en la esquina superior derecha permite alternar entre la visualización original en tonos de gris y una representación pseudocoloreada (*colormap*) solo en los tres tipos de gradientes. La versión

coloreada facilita la percepción visual de las variaciones de intensidad, haciendo más evidente la acción de los operadores morfológicos sobre máximos, mínimos y transiciones locales de la imagen.

Los operadores presentados están disponibles en `morph.py` mediante las funciones `mm::gradm`, `mm::tophat` y `mm::blackhat`.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.es/cap04/sim-04-operadores-av.html

Figura 4.13: Simulador interactivo avanzado de morfología con elemento estructurante editable y perfil 1D.

La Figura 4.14 ilustra los efectos de estos operadores sobre la imagen de las monedas y sus respectivos histogramas. Observe que la erosión desplaza la distribución hacia intensidades más bajas, mientras que la dilatación la desplaza hacia intensidades más altas. El gradiente concentra valores en las regiones de contorno, y los operadores *Top-hat* y *Black-hat* producen histogramas fuertemente concentrados en bajos niveles de intensidad, ya que solo se resaltan pequeñas estructuras locales.

```
%%writefile tmp/fig_04_morf_gc_histogramas.cpp
#define MM_OUT "tmp/fig_04_morf_gc_histogramas.png"
//| label: fig-04-morf-gc-histogramas
//| fig-cap: "Morfologia em tons de cinza e seus histogramas: erosão, dilatação, gradiente,
top-hat e black-hat. Elemento estruturante: disco de diâmetro 19."
//| echo: true
//| output: true
```

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

int main() {
// [pdi:state-io] auto-generated - do not edit by hand
mm::Image img_coins_gray = mm::_read_state("tmp/state/img_coins_gray_8.png");
// [pdi:state-io:end]

    // img_coins_gray is already available as mm::Image

    mm::Image B = mm::sedisk(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);

    mm::show(
        std::vector<mm::Image>{img_coins_gray, mm::histImg(img_coins_gray), img_ero,
mm::histImg(img_ero),
                               img_dil, mm::histImg(img_dil), img_grad, mm::histImg(img_grad),
                               img_th, mm::histImg(img_th), img_bh, mm::histImg(img_bh)},
        MM_OUT,
        std::vector<std::string>{"Original", "Hist", "Erosão", "Hist", "Dilatação", "Hist",
                                 "Gradiente", "Hist", "Top-hat", "Hist", "Black-hat", "Hist"},
        4
    );

    return 0;
}
```

```
Overwriting tmp/fig_04_morf_gc_histogramas.cpp
```

```
!g++ -I. -std=c++17 -DMM_USE_OPENCV -I/usr/include/opencv4 tmp/fig_04_morf_gc_histogramas.cpp
-o tmp/fig_04_morf_gc_histogramas -lopencv_imgproc -lopencv_imgcodecs -lopencv_core \
  && ./tmp/fig_04_morf_gc_histogramas \
  && test -f "tmp/fig_04_morf_gc_histogramas.png" \
  || echo "  mm::show não gravou tmp/fig_04_morf_gc_histogramas.png"
```

```
[1] Original
[2] Hist
[3] Erosão
[4] Hist
[5] Dilatação
[6] Hist
[7] Gradiente
[8] Hist
[9] Top-hat
[10] Hist
[11] Black-hat
[12] Hist
```

```
try:
    mm.show(mm.read("tmp/fig_04_morf_gc_histogramas.png"))
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + "
    tmp/fig_04_morf_gc_histogramas.png (ver a versao Python)")
```

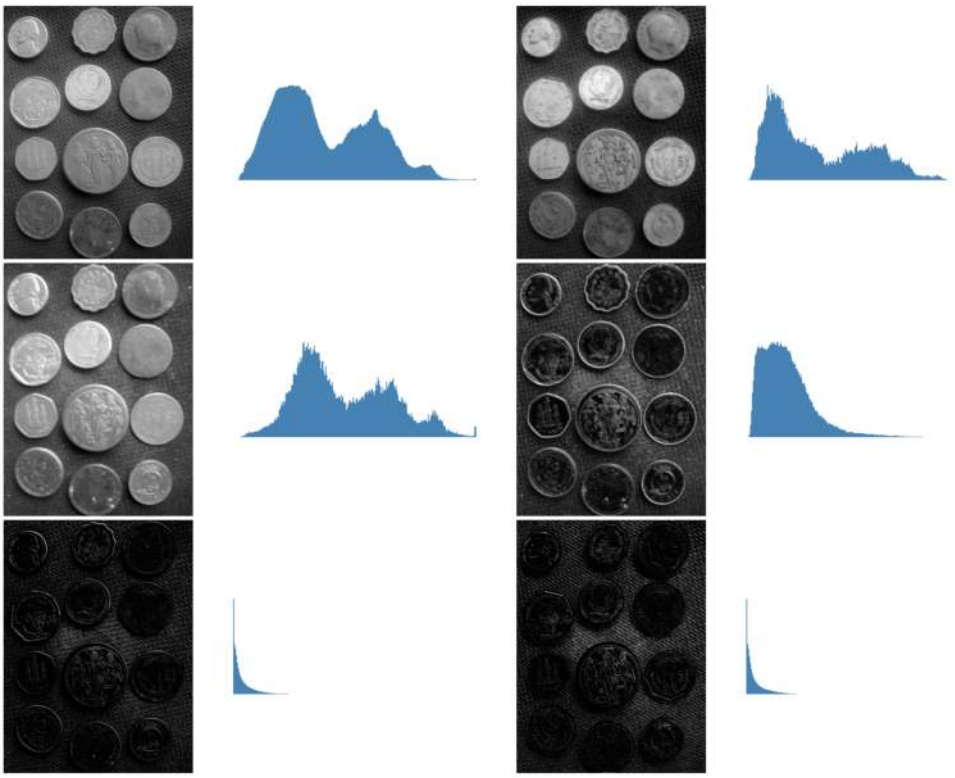


Figura 4.14: Morfologia em tons de cinza e seus histogramas: erosão, dilatação, gradiente, top-hat e black-hat. Elemento estruturante: disco de diâmetro 19.

Los operadores morfológicos presentados anteriormente se utilizarán ahora como herramientas de refinamiento y generación de marcadores para métodos de segmentación más avanzados, que se presentan a continuación.

4.4 Segmentación de Imágenes: Fundamentación y Taxonomía

La **segmentación de imágenes** consiste en dividir la imagen en regiones asociadas a objetos o estructuras de interés. En PDI, representa la transición entre el procesamiento de bajo nivel —como filtrado y realce— y etapas de análisis más avanzadas, como extracción de características, reconocimiento e interpretación de la escena.

Formalmente, el objetivo de la segmentación consiste en descomponer el dominio espacial completo de una imagen, denotado por Ω, en una partición de subconjuntos $\{R_1, R_2, \dots, R_n\}$ que satisfaga simultáneamente los criterios de **completitud** y **disyunción**:

$$\bigcup_{i=1}^{n} R_i = \Omega, \qquad R_i \cap R_j = \emptyset \quad \forall\, i \neq j \tag{4.15}$$

Además de las propiedades de completitud y disyunción expresadas en la Ecuación 4.15, cada subregión R_i debe constituir un dominio **homogéneo** según un predicado de similitud definido sobre propiedades locales —intensidad, color o textura— y, simultáneamente, ser **distinta** de las regiones adyacentes.

Las técnicas de segmentación pueden organizarse en diferentes familias. En este capítulo se enfatizarán los enfoques resumidos en la Tabla 4.4, fundamentados principalmente en criterios de intensidad, conectividad y proximidad espacial.

Tabla 4.4: Taxonomía simplificada de los principales enfoques de segmentación y refinamiento estudiados en este capítulo.

Enfoque	Criterio de Segmentación	Operadores de Referencia
Umbralización	Particionamiento del espacio de intensidades	Criterio de Otsu, umbralización global y local
Morfología Matemática	Relaciones espaciales definidas por elementos/funciones estructurantes	Erosión, dilatación, apertura, cierre y reconstrucción
Basada en Regiones	Homogeneidad local y conectividad espacial	Etiquetado de componentes conexos, Transformada de Distancia y *Watershed*

Hasta este punto, el desarrollo práctico se ha centrado en la **umbralización**, mediante la combinación entre ecualización adaptativa CLAHE y el método global de Otsu. Esta etapa se complementó con operadores de **reconstrucción morfológica** basados en dilataciones geodésicas, implementados por las funciones `mm::infrec`, `mm::clohole` y `mm::edgeoff`, produciendo una máscara binaria limpia y adecuada para el análisis.

Sin embargo, en escenarios donde objetos distintos aparecen conectados en la máscara binaria —ya sea por contacto físico, superposición parcial o por puentes estrechos de píxeles producidos por la segmentación—, la umbralización deja de ser suficiente para individualizar cada objeto. En estos casos, múltiples objetos pasan a componer un único componente conexo, dificultando etapas posteriores de medición e interpretación.

Para superar esta limitación, las próximas secciones introducen tres herramientas complementarias: el **Etiquetado de Componentes Conexos**, la **Transformada de Distancia** y el algoritmo de segmentación por ***Watershed*** basado en marcadores. En conjunto, estas técnicas permiten separar objetos adyacentes, identificar regiones individualmente y extraer descriptores geométricos consistentes para el análisis cuantitativo.

4.4.1 Etiquetado

El **etiquetado de componentes conexas** (*connected component labeling*) es el operador que asigna un identificador entero único a cada conjunto de píxeles pertenecientes a la misma componente conexa en una imagen binaria.

> **Definición formal**
>
> Dada una imagen binaria f y una relación de conectividad definida por un elemento estructurante B (típicamente conectividad-4 o conectividad-8), el algoritmo de etiquetado de la Figura 4.15 produce una imagen g en la cual todos los píxeles pertenecientes a la misma componente conexa reciben la misma etiqueta entera positiva, mientras que los píxeles pertenecientes a componentes distintas reciben etiquetas diferentes.

La conectividad define qué píxeles se consideran vecinos directos de un píxel (x, y). Las definiciones más utilizadas son:

- **Conectividad-4:** considera solo los cuatro vecinos ortogonales (norte, sur, este y oeste).
- **Conectividad-8:** considera los cuatro vecinos ortogonales y los cuatro diagonales, totalizando ocho vecinos.

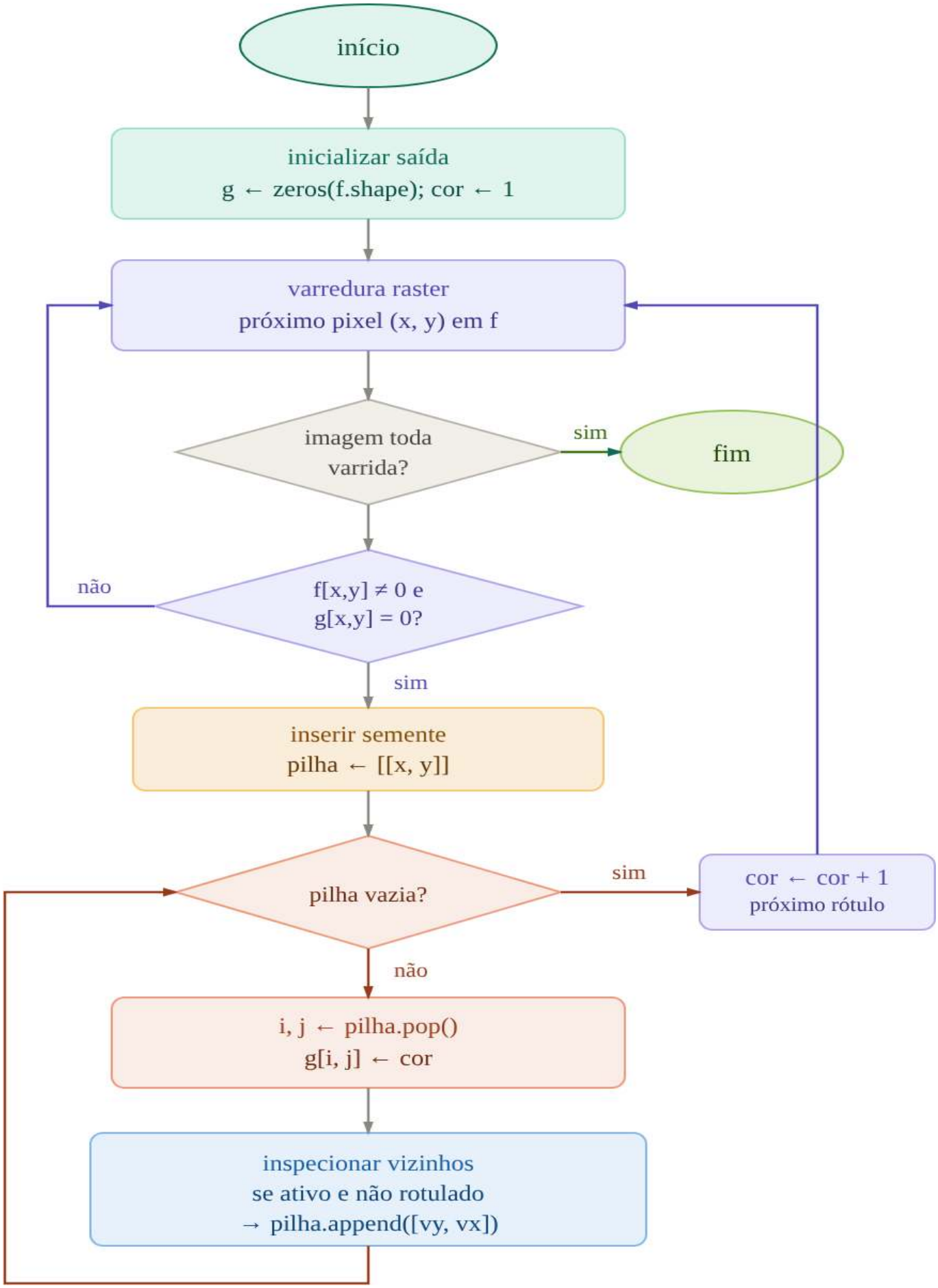
La elección de la conectividad influye directamente en la formación de las componentes conexas y, consecuentemente, en el resultado del etiquetado, como se ilustra en la Figura 4.17. Un ejemplo adicional puede explorarse de forma interactiva en el simulador presentado en la Figura 4.16.

La implementación en `morph.py` proporciona dos versiones de este operador. La función `mm::label0` reproduce explícitamente el algoritmo de *flood-fill* utilizando una pila y permite controlar la conectividad mediante el elemento estructurante adoptado. En cambio, `mm::label` delega la operación a la implementación optimizada de OpenCV (`mm::label0`). En ambos casos, el resultado es una imagen etiquetada en la cual cada componente conexa recibe un identificador entero distinto.

El auxiliar `_viz` itera sobre la ventana estructurante `b` centrada en (i, j), generando únicamente los vecinos válidos dentro de los límites de la imagen — la conectividad deseada está enteramente determinada por la forma de `b` pasada al algoritmo.

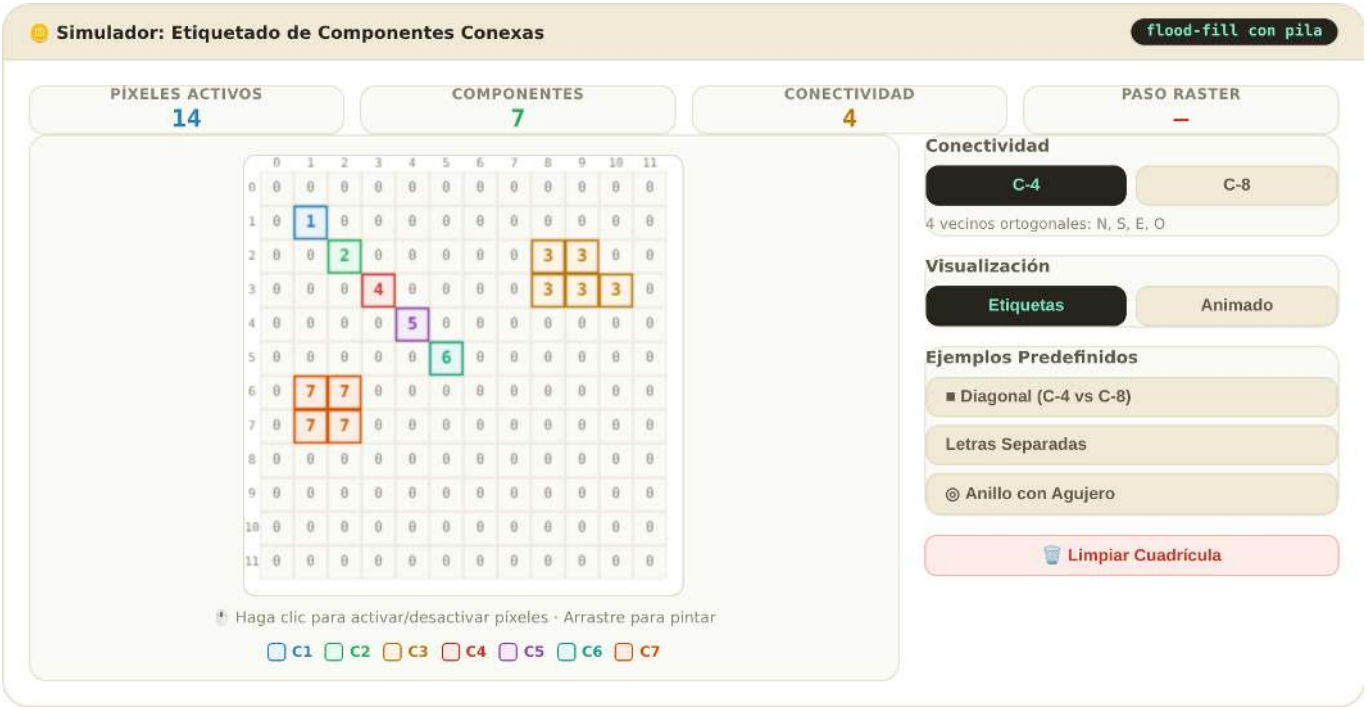
Ejemplo didáctico — efecto de la conectividad:

```
%%writefile tmp/fig_04_rotulacao_didatico.cpp
#define MM_OUT "tmp/fig_04_rotulacao_didatico.png"
//| label: fig-04-rotulacao-didatico
//| fig-cap: "Efeito da conectividade na rotulação (*mm::label0*): pixels diagonalmente
adjacentes formam componentes distintas em conectividade-4 e se fundem em conectividade-8."
//| echo: true
//| output: true

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

int main() {
    mm::Image f(10, 10);
    unsigned char vals[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) = vals[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);
    int n4 = 0, n8 = 0;
    for (int i = 0; i < (int)lbl4.data.size(); i++)
        n4 = std::max(n4, (int)lbl4.data[i]);
    for (int i = 0; i < (int)lbl8.data.size(); i++)
        n8 = std::max(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);
```



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.es/cap04/sim-alg-rotulagem2.html

Figura 4.15: Algoritmo de etiquetado por *flood-fill* con pila.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.es/cap04/sim-04-rotulacao.html

Figura 4.16: Simulador interactivo de etiquetado de componentes conexas (*connected component labeling*): visualización de la expansión flood-fill, conectividad-4 y conectividad-8.

```
            }
        }
        return out;
    };

    mm::show(
        std::vector<mm::Image>{f, norm_label(lbl4, std::max(n4, 1)), norm_label(lbl8,
std::max(n8, 1))},
        MM_OUT,
        std::vector<std::string>{"f original", "C4 (" + std::to_string(n4) + " comp.)", "C8 ("
+ std::to_string(n8) + " comp.)"},
        3
    );

    return 0;
}
```

```
Overwriting tmp/fig_04_rotulacao_didatico.cpp
```

```
!g++ -I. -std=c++17 -DMM_USE_OPENCV -I/usr/include/opencv4 tmp/fig_04_rotulacao_didatico.cpp
-o tmp/fig_04_rotulacao_didatico -lopencv_imgproc -lopencv_imgcodecs -lopencv_core \
  && ./tmp/fig_04_rotulacao_didatico \
  && test -f "tmp/fig_04_rotulacao_didatico.png" \
  || echo "  mm::show não gravou tmp/fig_04_rotulacao_didatico.png"
```

```
Componentes C4: 7  |  C8: 4
[1] f original
[2] C4 (7 comp.)
[3] C8 (4 comp.)
```

```
try:
    mm.show(mm.read("tmp/fig_04_rotulacao_didatico.png"))
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + "
    tmp/fig_04_rotulacao_didatico.png (ver a versao Python)")
```

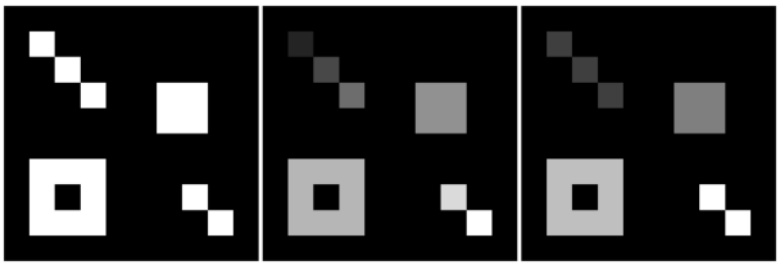


Figura 4.17: Efeito da conectividade na rotulação (*mm::label0*): pixels diagonalmente adjacentes formam componentes distintas em conectividade-4 e se fundem em conectividade-8.

4.4.2 Transformada de Distancia

La **Transformada de Distancia** (TD) es un operador que, aplicado a una imagen binaria f, produce una imagen en niveles de gris D en la cual cada píxel perteneciente al objeto ($f(x, y) \neq 0$) recibe como valor la distancia geométrica hasta el píxel de fondo ($f(x', y') = 0$) más cercano:

$$D(x, y) = \min_{(x', y') \,:\, f(x', y') = 0} d((x, y),\, (x', y'))$$

donde $d(\cdot, \cdot)$ es una métrica de distancia — típicamente la distancia Euclidiana (L_2). El resultado es una representación topográfica de los objetos: los píxeles ubicados en el interior asumen valores elevados, mientras que los píxeles próximos a los bordes presentan bajos valores de distancia. Los **máximos locales** de D corresponden a los puntos más alejados del borde del objeto, frecuentemente próximos a sus centros geométricos o centros de máxima inscripción — propiedad particularmente útil para la generación automática de marcadores en el algoritmo *watershed*.

> **i** Definición formal mediante erosiones
>
> La TD admite además una interpretación morfológica iterativa, conforme al algoritmo de la Figura 4.18. Considérese una función estructurante b cuyo valor central es nulo y cuyos vecinos poseen costos negativos asociados al desplazamiento. Al aplicar erosiones sucesivas con esta función estructurante particular, los valores de los píxeles de los objetos (que deben asumir la distancia máxima posible de la imagen) se reducen progresivamente según los costos definidos por b. El valor acumulado de esta propagación pasa entonces a representar la distancia al fondo según la métrica inducida por la función estructurante.

Esta interpretación se implementa en `mm::dist1()`, que acumula erosiones sucesivas utilizando la operación `mm::ero1()`. Por su parte, `mm::dist()` delega el cálculo de la distancia Euclidiana al operador optimizado de OpenCV `mm::dist(f)`, donde `f` es la imagen binaria de entrada, la distancia L2 especifica la métrica Euclidiana (L_2) y `5` indica el uso de una máscara 5×5 para aproximar la distancia con elevada precisión.

La función `dist1` produce una transformada de distancia discreta cuya métrica está determinada por la geometría y los pesos de la función estructurante utilizada. Por ejemplo, utilizando una función estructurante en cruz con costo unitario para los cuatro vecinos ortogonales, se obtiene la distancia de ***Manhattan*** (L_1). Otras elecciones de vecindad y pesos inducen métricas diferentes. En cambio, `mm::dist()` calcula una aproximación eficiente de la distancia Euclidiana (L_2).

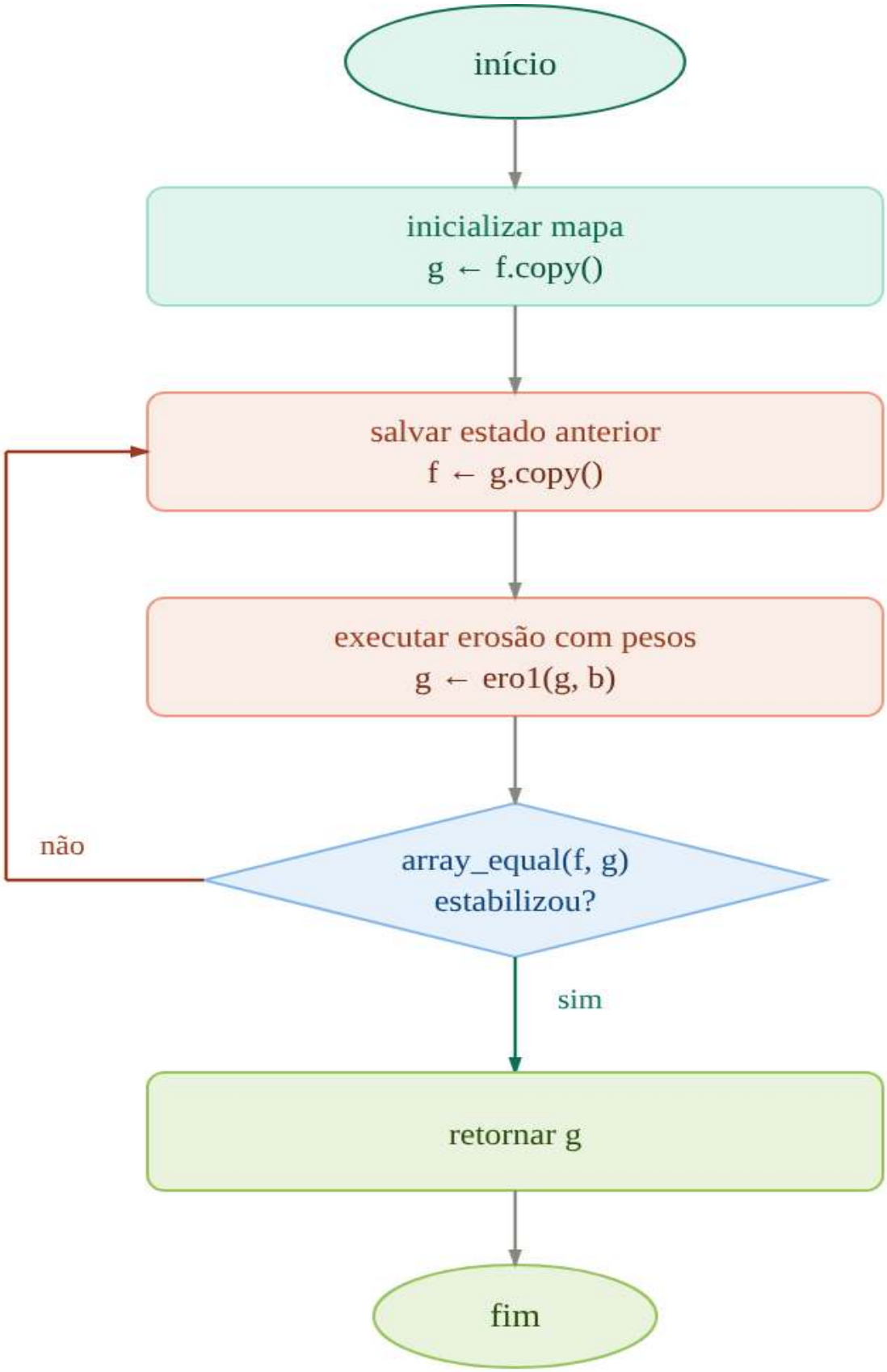
Por exigir sucesivas erosiones sobre toda la imagen, el enfoque `dist1` posee un costo computacional significativamente mayor que `mm::dist()`, siendo empleado en este libro principalmente con fines didácticos y para evidenciar la relación entre morfología matemática y transformadas de distancia.

La Figura 4.19 presenta un simulador interactivo de la TD: es posible posicionar el cursor sobre diferentes píxeles del objeto y observar, en tiempo real, el valor de la distancia asociado a esa posición, es decir, la distancia hasta el píxel de fondo más cercano. La Figura 4.20 presenta un ejemplo práctico de esta ejecución en entorno Python.

```
%%writefile tmp/fig_04_distancia_didatico.cpp
#define MM_OUT "tmp/fig_04_distancia_didatico.png"
//| label: fig-04-distancia-didatico
//| fig-cap: "Transformada de Distância em imagem binária 10×10. Esquerda: original
(*foreground* = 255). Centro: mm.dist1 iterativa (erosões com cruz). Direita: mm.dist (L2)."
//| echo: true
//| output: true

#include "morph.hpp"
#include <iostream>
#include <algorithm>
#include <vector>
#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);

    std::cout << "Máx. dist1 (erosões) : " << (int)*std::max_element(d_iter.data.begin(),
d_iter.data.end()) << " px\n";
    std::cout << "Máx. dist  (L2)      : " << (int)*std::max_element(d_l2.data.begin(),
d_l2.data.end()) << " 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
```

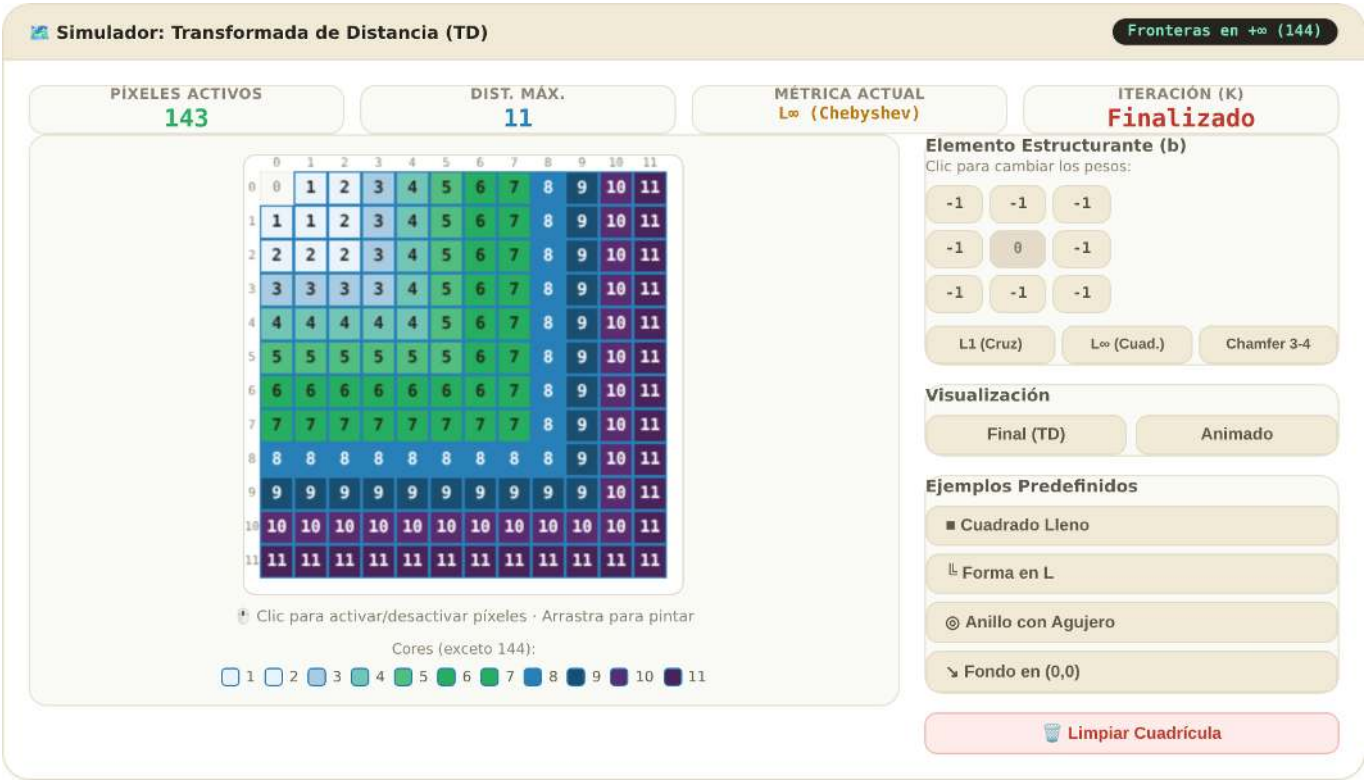
```
!g++ -I. -std=c++17 -DMM_USE_OPENCV -I/usr/include/opencv4 tmp/fig_04_distancia_didatico.cpp
-o tmp/fig_04_distancia_didatico -lopencv_imgproc -lopencv_imgcodecs -lopencv_core \
  && ./tmp/fig_04_distancia_didatico \
```



Ponto Fixo Numérico: A distância emerge da propagação matemática do valor zero.

Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.es/cap04/sim-alg-distancia2.html

Figura 4.18: Algoritmo de la Transformada de Distancia.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.es/cap04/sim-04-distancia.html

Figura 4.19: Simulador interactivo de la Transformada de Distancia (TD) iterativa mediante erosión en tonos de gris. Los píxeles fuera de la imagen asumen el valor máximo (144), propagando los costos a partir del fondo interno.

```
  && 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)")
```

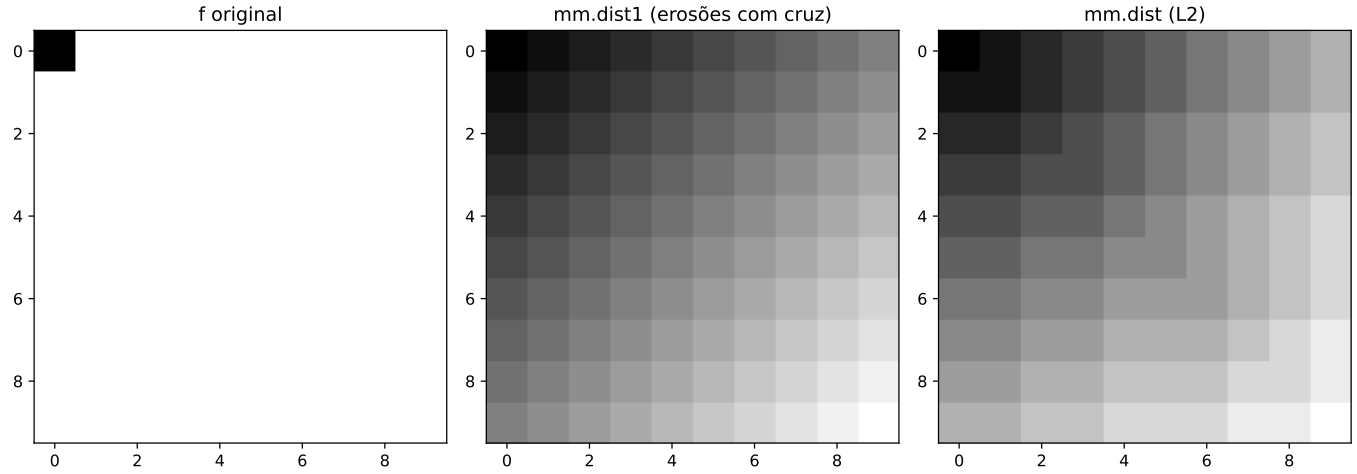


Figura 4.20: Transformada de Distância em imagem binária 10×10. Esquerda: original (*foreground* = 255). Centro: mm.dist1 iterativa (erosões com cruz). Direita: mm.dist (L2).

La anotación de los valores numéricos directamente sobre los píxeles permite verificar cómo `dist1` propaga las distancias según la métrica inducida por la función estructural utilizada. En el caso del elemento cruz con costo unitario, los valores obtenidos corresponden a la distancia de *Manhattan* (L_1). Aunque `dist1` y `mm::dist` producen valores numéricos distintos por adoptar métricas diferentes, ambas transformadas preservan la estructura topográfica de los objetos, haciendo que sus máximos ocurran en regiones centrales similares. Esta propiedad justifica el uso de `mm::dist` en aplicaciones prácticas, debido a su elevada eficiencia computacional.

4.4.3 Transformada de Distancia Euclidiana en cuatro pasos

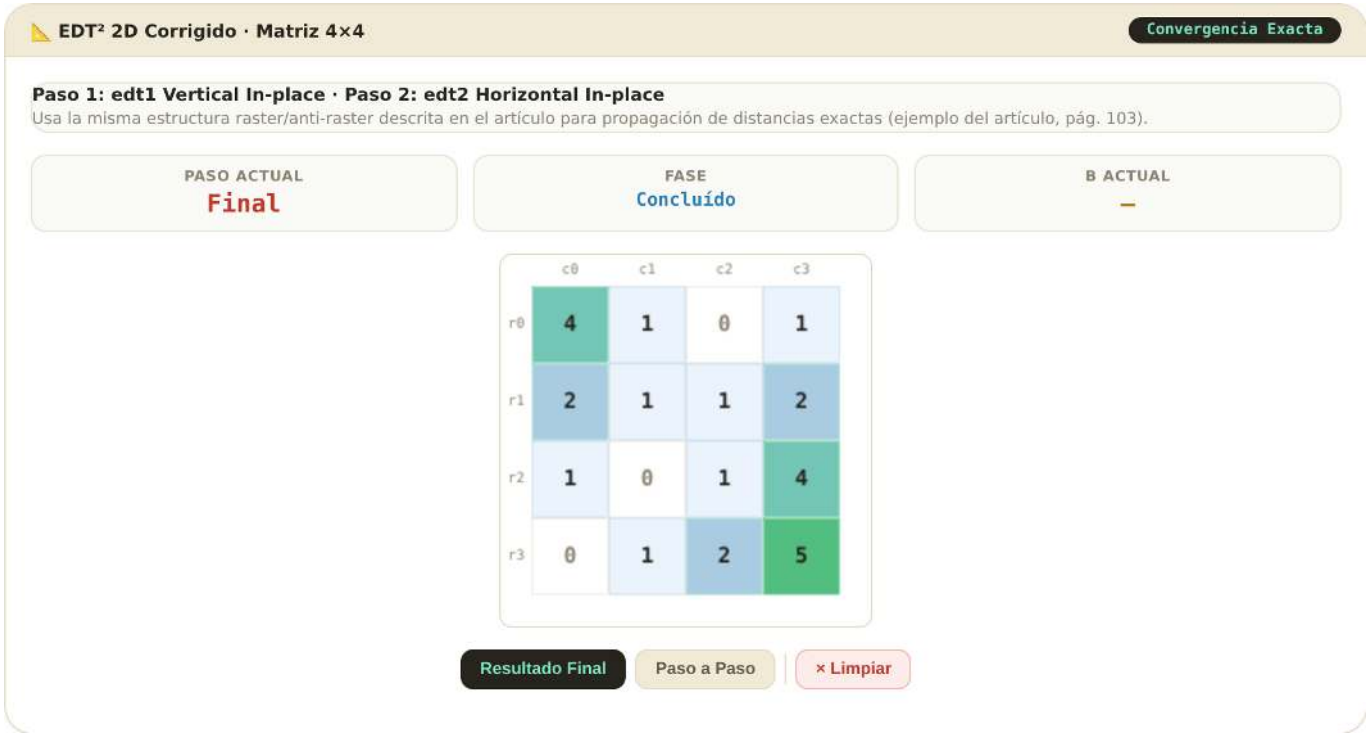
El simulador de la Figura 4.21 implementa el algoritmo de la TDE de Lotufo (2001) en dos etapas. En la primera etapa, la función `edt1` realiza una transformación unidimensional vertical de forma secuencial (*in-place*), recorriendo cada columna en *raster* (↓) y *anti-raster* (↑) para calcular las distancias en la dirección vertical (dos pasos: Sur y Norte). En la segunda etapa, la función `edt2` utiliza ese resultado como entrada y realiza una propagación horizontal por colas: para cada fila de la matriz, dos colas de prioridad, `Eq` y `Wq`, se inicializan recorriendo los índices de columna en sentidos opuestos (`Eq` de `W-1` hasta `1`, `Wq` de `2` hasta `W`), de modo que cada píxel actualizado encola inmediatamente a sus vecinos para reprocesamiento dentro de la misma ronda (dos pasos más: Este y Oeste). Este mecanismo de cola permite aplicar erosiones sucesivas con pesos impares crecientes (`b = 1, 3, 5, ...`, incrementado en cada iteración del bucle externo) sin que la propagación quede atrapada en valores desactualizados, ya que cada ronda resuelve por completo la cadena de dependencias horizontales antes del siguiente incremento de `b`. La convergencia de esta propagación produce la Transformada de Distancia Euclidiana en toda la matriz, combinando la información vertical obtenida en `edt1` con la propagación horizontal en cola realizada en `edt2`.

4.4.3.1 Transformada de Distancia Geodésica

La transformada de distancia geodésica asocia a cada píxel la menor distancia hasta un marcador, bajo la restricción impuesta por una máscara. De esta forma, la propagación ocurre exclusivamente por los píxeles permitidos, preservando la conectividad del dominio.

La Figura 4.22 ilustra este proceso en un laberinto: (a) la máscara `g`; (b) la distancia geodésica `D1` calculada a partir de la entrada; (c) la distancia `D2` calculada a partir de la salida; y (d) el camino mínimo obtenido a partir de estas dos transformadas.

El camino óptimo se determina mediante la suma de las distancias (`D1 + D2`). Los píxeles pertenecientes a la trayectoria mínima son aquellos para los cuales esta suma asume su menor valor, definiendo una conexión entre entrada y salida con longitud geodésica mínima.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.es/cap04/sim-04-tde.html

Figura 4.21: Simulador interactivo 2D (4x4) con sincronización estricta de colas de propagación horizontal (b) para obtener la convergencia exacta descrita en el artículo.

Este principio permite resolver laberintos sin la necesidad de explorar explícitamente todas las posibilidades de recorrido. La solución emerge directamente de la propagación de distancias en un dominio restringido. Este enfoque es particularmente relevante en laberintos de elevada complejidad, como los construidos a partir de estructuras cuasicristalinas y ciclos hamiltonianos descritos por Singh (2024). En Zampirolli (2025), este mismo formalismo se emplea para resolver un laberinto complejo; a continuación, el método se ilustra en una versión simplificada 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 -I. program.cpp -o program $(pkg-config --cflags --libs
opencv4) -DMM_USE_OPENCV
//| label: fig-04-gdist-menor-caminho
//| fig-cap: "Menor caminho geodésico em um labirinto. As distâncias geodésicas são calculadas
a partir da entrada e da saída usando mm.gdist. Os pixels cujo somatório das duas distâncias é
igual à distância mínima entre os marcadores pertencem a um caminho ótimo."
//| echo: true
//| output: true
#include "morph.hpp"
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>

int main() {
    // 1 = corredor, 0 = parede
    mm::Image g(10, 10);
    unsigned char 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);
    entrada.at(0, 1) = 1;

    // marcador da saída
    mm::Image saida(10, 10);
    saida.at(9, 8) = 1;

    // distâncias geodésicas
    mm::Image D1 = mm::gdist(g, entrada);
    mm::Image D2 = mm::gdist(g, saida);

    // soma das distâncias
    mm::Image S = mm::addm(D1, D2);

    // menor valor válido da soma
    int dmin = 255;
    for (int i = 0; i < S.h * S.w; i++)
        if (S.data[i] > 0 && S.data[i] < dmin)
            dmin = S.data[i];

    // pixels pertencentes a um caminho ótimo
    mm::Image caminho(S.h, S.w);
    for (int i = 0; i < S.h * S.w; i++)
        caminho.data[i] = (S.data[i] == dmin) ? 255 : 0;

    std::cout << "Distância geodésica mínima: " << dmin << "\n";

    mm::show(
        std::vector<mm::Image>{g, D1, D2, caminho},
        MM_OUT,
        std::vector<std::string>{
            "Labirinto",
            "Distância da Entrada",
            "Distância da Saída",
            "Menor Caminho\n(d=" + std::to_string(dmin) + ")"
        },
        4
    );

    return 0;
}
```

```
Overwriting tmp/fig_04_gdist_menor_caminho.cpp
```

```
!g++ -I. -std=c++17 -DMM_USE_OPENCV -I/usr/include/opencv4 tmp/fig_04_gdist_menor_caminho.cpp
-o tmp/fig_04_gdist_menor_caminho -lopencv_imgproc -lopencv_imgcodecs -lopencv_core \
  && ./tmp/fig_04_gdist_menor_caminho \
  && test -f "tmp/fig_04_gdist_menor_caminho.png" \
```

```
  || echo "  mm::show não gravou tmp/fig_04_gdist_menor_caminho.png"
```

```
Distância geodésica mínima: 15
[1] Labirinto
[2] Distância da Entrada
[3] Distância da Saída
[4] Menor Caminho
(d=15)
```

```
try:
    mm.show(mm.read("tmp/fig_04_gdist_menor_caminho.png"), figsize=(14, 4))
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + "
    tmp/fig_04_gdist_menor_caminho.png (ver a versao Python)")
```

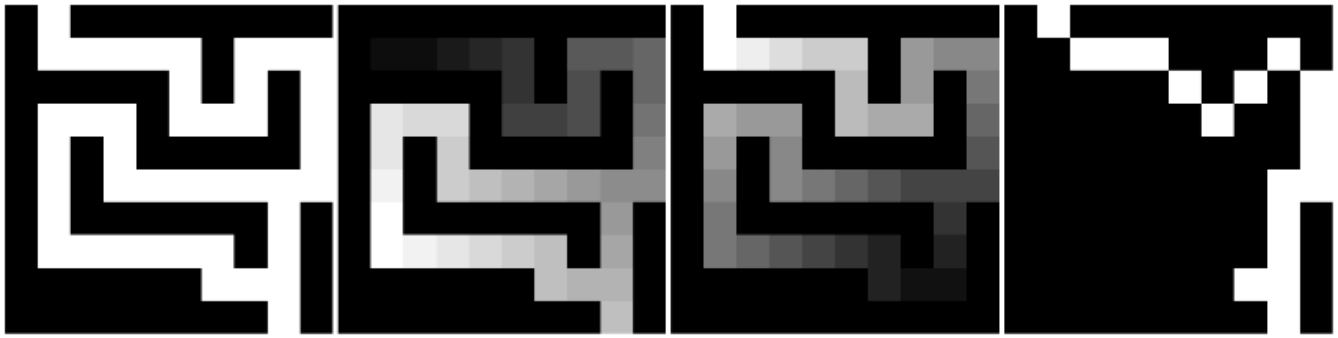


Figura 4.22: Menor caminho geodésico em um labirinto. As distâncias geodésicas são calculadas a partir da entrada e da saída usando mm.gdist. Os pixels cujo somatório das duas distâncias é igual à distância mínima entre os marcadores pertencem a um caminho ótimo.

4.4.4 Segmentación por *Watershed*

El algoritmo ***Watershed*** interpreta una imagen en niveles de gris como una superficie topográfica, en la cual valores elevados corresponden a montañas y valores bajos corresponden a valles o cuencas de drenaje (*catchment basins*). En el contexto de la segmentación basada en marcadores, los máximos de la Transformada de Distancia se utilizan frecuentemente para identificar regiones internas de los objetos, proporcionando semillas confiables para el proceso de inundación.

La segmentación se realiza entonces mediante una simulación conceptual de inundación progresiva a partir de estos marcadores. A medida que las cuencas asociadas a diferentes semillas se expanden, las regiones vecinas eventualmente entran en contacto. En ese instante se construyen barreras virtuales, denominadas *watershed lines*, que pasan a delimitar los objetos de la escena. Este mecanismo permite separar objetos adyacentes o parcialmente superpuestos, incluso cuando forman una única componente conexa tras la umbralización.

La implementación didáctica presentada en este capítulo explora inicialmente el concepto de crecimiento de regiones (*region growing*) confinado por una máscara binaria, según se detalla en el algoritmo interactivo de la Figura 4.23.

i Versión didáctica versus implementación clásica

La función `mm::watershed0` no implementa el algoritmo *watershed* clásico. Su objetivo es ilustrar, de forma simplificada, la propagación de marcadores por crecimiento de regiones (*region growing*), permitiendo visualizar cómo diferentes semillas compiten por la ocupación del espacio disponible. El crecimiento está delimitado por una máscara binaria de soporte y monitoreado por un control de estancamiento, generando un resultado similar a una partición de Voronoi restringida a la geometría de

los objetos de entrada.
En cambio, la función `mm::watershed` utiliza la implementación optimizada de OpenCV (`mm::watershed`), que realiza la inundación sobre una superficie topográfica definida por la imagen de entrada. En este caso, la propagación de los marcadores está influenciada por los valores de los píxeles, haciendo que las líneas de separación se formen naturalmente sobre las crestas del relieve.

A Figura 4.24 presenta un simulador iterativo que ilustra la propagación de los marcadores por la región de interés. Cada marcador actúa como una fuente de inundación que expande su área de influencia hasta encontrar regiones provenientes de otras semillas. En el algoritmo de OpenCV, los píxeles pertenecientes a las líneas divisorias se identifican por el valor `-1`, representando las fronteras entre cuencas adyacentes.

Pipeline Morfológico del *Watershed*

El *watershed* basado en marcadores normalmente integra un flujo más amplio de segmentación. En imágenes reales, etapas de preprocesamiento son frecuentemente necesarias para mejorar el contraste, reducir ruidos y generar marcadores confiables. Este flujo completo está resumido en la Tabla 4.5.

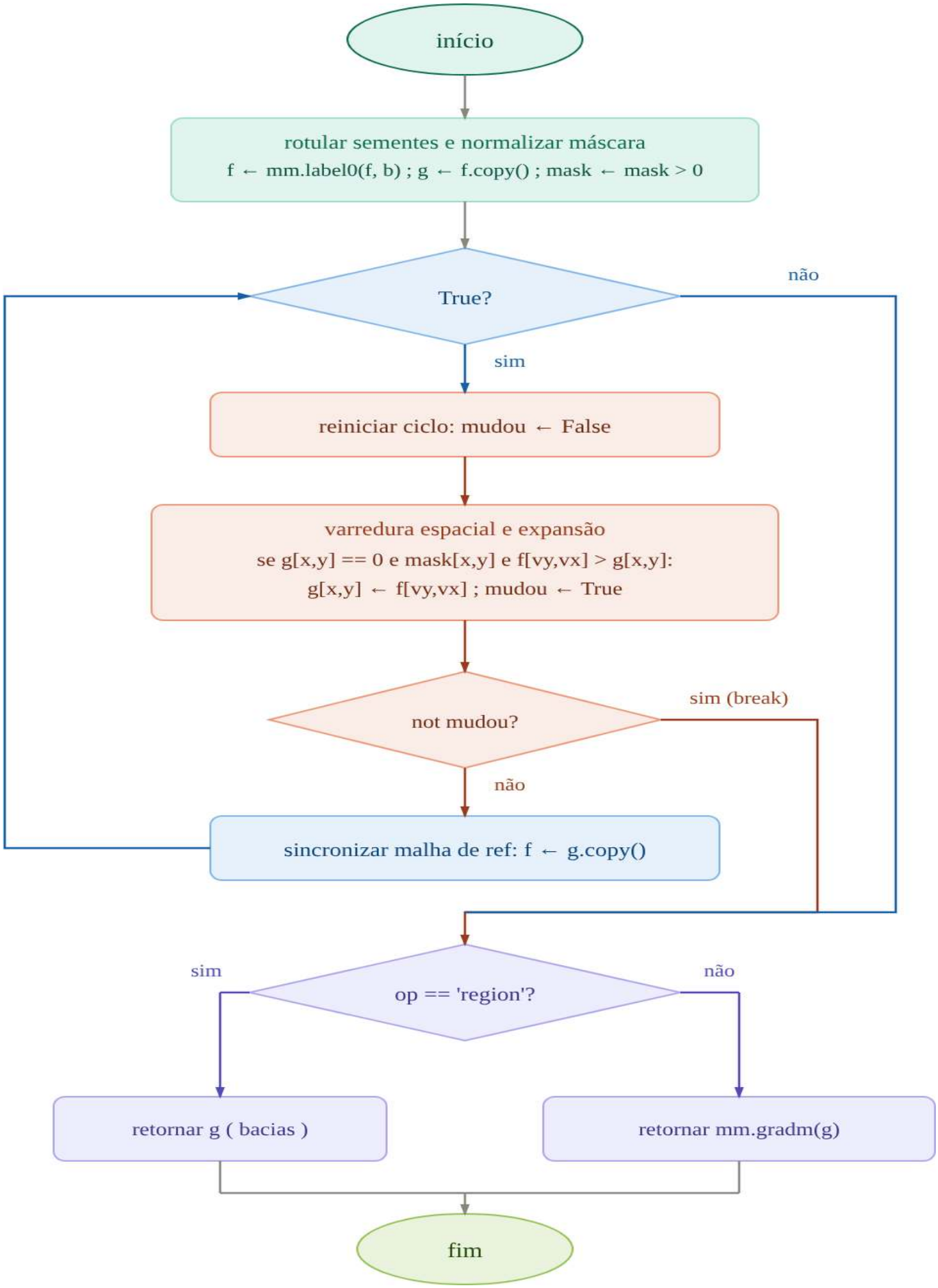
Tabla 4.5: *Pipeline* completo del *watershed* basado en marcadores para imágenes reales.

Etapa	Operación	Finalidad
1	CLAHE + Suavizado	Realce de contraste y reducción de ruido
2	Umbralización	Separación inicial entre objeto y fondo
3	Apertura/Cierre	Eliminación de ruidos y pequeñas imperfecciones
4	Dilatación de la Máscara	Identificación del Fondo Seguro (*Sure Background*)
5	Transformada de Distancia + Umbral	Identificación del Objeto Seguro (*Sure Foreground*)
6	Región Incierta	Diferencia entre Fondo Seguro y Objeto Seguro
7	`mm::watershed`	Propagación de los marcadores por la región incierta

Para enfatizar exclusivamente los conceptos de Transformada de Distancia, marcadores e inundación topográfica, el ejemplo de la Figura 4.25 utiliza una imagen binaria sintética y adopta un flujo simplificado, resumido en la Tabla 4.6.

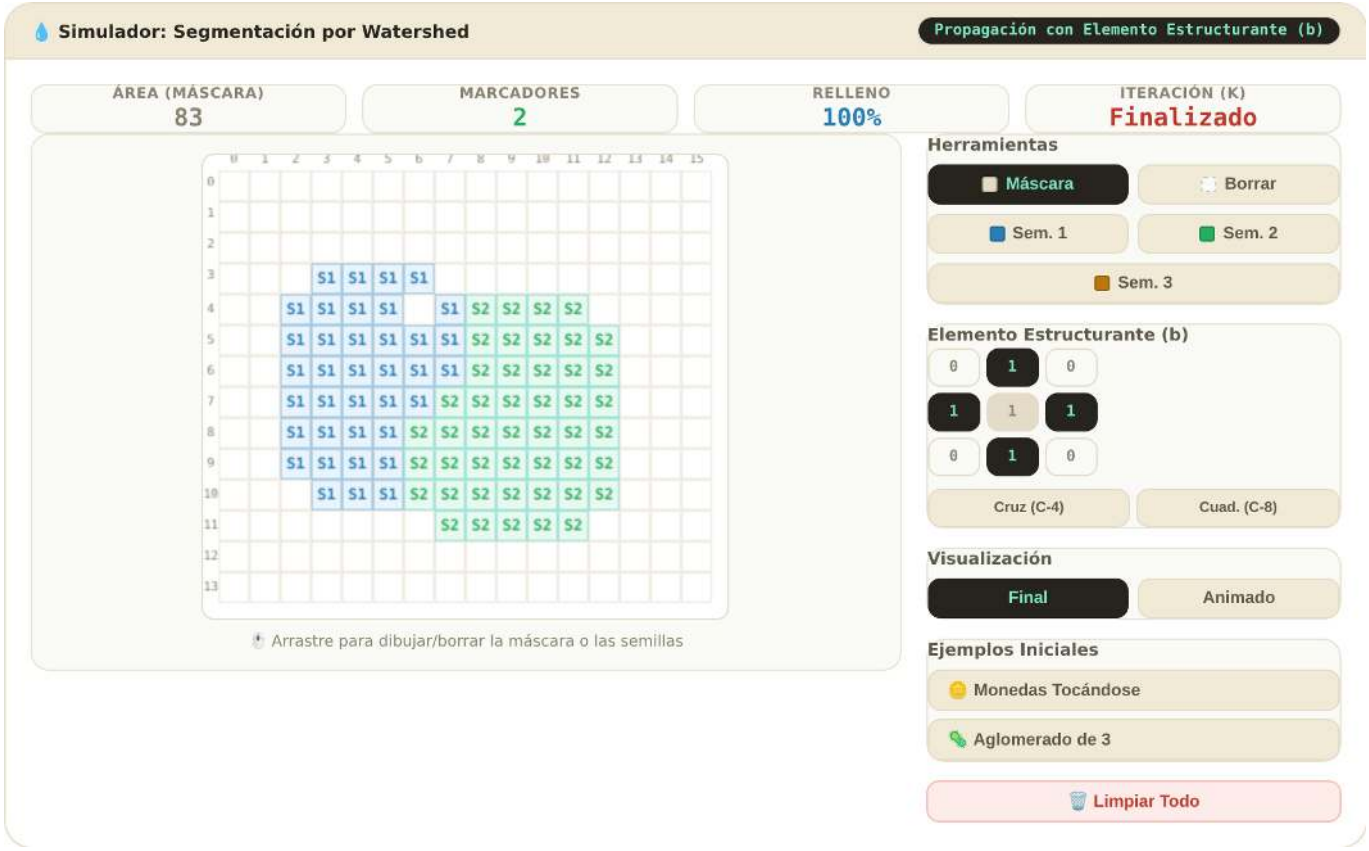
Tabla 4.6: *Pipeline* simplificado utilizado en el ejemplo didáctico de la Figura 4.25.

Etapa	Operación	Finalidad
1	Transformada de Distancia	Construcción de la superficie topográfica
2	Umbral de la TD	Extracción de los marcadores (*Sure Foreground*)
3	Dilatación	Determinación del Fondo Seguro (*Sure Background*)
4	Región Incierta	Diferencia entre fondo y marcadores
5	`mm::watershed`	Propagación de los marcadores y generación de las fronteras



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.es/cap04/sim-alg-watershed2.html

Figura 4.23: Algoritmo Didáctico de *Watershed* por Crecimiento de Regiones Limitado por Máscara.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.es/cap04/sim-04-watershed.html

Figura 4.24: Simulador interactivo del Algoritmo *Watershed* por propagación morfológica. Dibuja la máscara, coloca los marcadores y ajusta el Elemento Estructurante para observar la inundación. Cuando las cuencas se encuentran simultáneamente, el empate se resuelve asumiendo una de las regiones de manera aleatoria.

```
%%writefile tmp/fig_04_watershed_didatico.cpp
#define MM_OUT "tmp/fig_04_watershed_didatico.png"
// g++ -std=c++17 snippet.cpp -o snippet $(pkg-config --cflags --libs opencv4) -DMM_USE_OPENCV
#include "morph.hpp"
#include <vector>
#include <string>
#include <algorithm>
#include <filesystem>

int main() {
    //| label: fig-04-watershed-didatico
    //| fig-cap: "*Pipeline* *watershed* delimitado por máscara em imagem binária 20×20."
    //| echo: true
    //| output: true

    // 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);
    unsigned char max_dist = 0;
    for (int y = 0; y < dist.h; y++) {
        for (int x = 0; x < dist.w; x++) {
            if (dist.at(y, x) > max_dist) {
                max_dist = dist.at(y, x);
            }
        }
    }
    for (int y = 0; y < dist.h; y++) {
        for (int x = 0; x < dist.w; x++) {
            m.at(y, x) = (dist.at(y, x) > 0.8 * max_dist) ? 255 : 0;
        }
    }

    // 3. Execução do Watershed0 condicional (m=marcadores primeiro, mask=f_sint)
    mm::Image w_reg = mm::watershed(m, f_sint, "region");
    mm::Image w_line = mm::watershed(m, f_sint, "line");

    //w_reg  = mm.watershedB(m, mask=f_sint, op='region')
    //w_line = mm.watershedB(m, mask=f_sint, op='line')

    // 4. Exibição dos resultados
    mm::show(std::vector<mm::Image>{f_sint, dist, m, w_reg, w_line}, MM_OUT,
             std::vector<std::string>{"Original", "Distância", "Marcadores", "Regiões",
"Linhas"}, 5);


// [pdi:panel-io] auto-generated - do not edit by hand
std::filesystem::create_directories("tmp");
mm::write(f_sint, "tmp/fig_04_watershed_didatico_0.png");
mm::write(dist, "tmp/fig_04_watershed_didatico_1.png");
mm::write(m, "tmp/fig_04_watershed_didatico_2.png");
mm::write(w_reg, "tmp/fig_04_watershed_didatico_3.png");
mm::write(w_line, "tmp/fig_04_watershed_didatico_4.png");
// [pdi:panel-io:end]
return 0;
}
```

```
Overwriting tmp/fig_04_watershed_didatico.cpp
```

```
!g++ -I. -std=c++17 -DMM_USE_OPENCV -I/usr/include/opencv4 tmp/fig_04_watershed_didatico.cpp
-o tmp/fig_04_watershed_didatico -lopencv_imgproc -lopencv_imgcodecs -lopencv_core \
  && ./tmp/fig_04_watershed_didatico \
  && test -f "tmp/fig_04_watershed_didatico.png" \
  || echo "  mm::show não gravou tmp/fig_04_watershed_didatico.png"
```

```
[1] Original
[2] Distância
[3] Marcadores
[4] Regiões
[5] Linhas
```

```
try:
    mm.show(
        [
            mm.read("tmp/fig_04_watershed_didatico_0.png"),
            mm.read("tmp/fig_04_watershed_didatico_1.png"),
            mm.read("tmp/fig_04_watershed_didatico_2.png"),
            mm.read("tmp/fig_04_watershed_didatico_3.png"),
            mm.read("tmp/fig_04_watershed_didatico_4.png"),
        ],
        titles=[
            'Original',
            'Distância',
            'Marcadores',
            'Regiões',
            'Linhas',
        ],
        cols=5,
        figsize=(16, 4),
    )
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + "
    tmp/fig_04_watershed_didatico_0.png (ver a versao Python)")
```

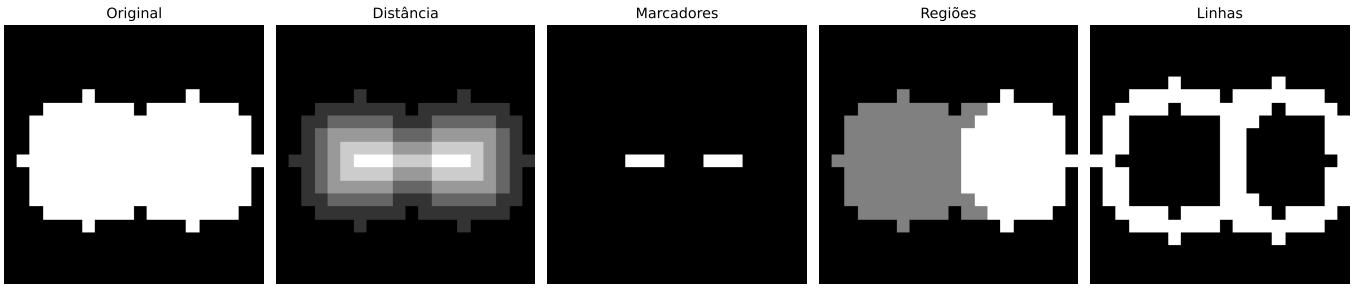


Figura 4.25: *Pipeline watershed* delimitado por máscara em imagem binária 20×20.

Aplicación en monedas superpuestas:

```
%%writefile tmp/fig_04_watershed_moedas.cpp
#define MM_OUT "tmp/fig_04_watershed_moedas.png"
#include "morph.hpp"
#include <opencv2/opencv.hpp>
#include <vector>
#include <string>
#include <iostream>
```

```
#include <set>
#include <algorithm>
#include <numeric>
#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]

    // Variables img_coins_gray (mm::Image) and img_final (mm::Image) 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);

    // Normalize dist to [0, 255] for visualization
    mm::Image dist_vis(dist.h, dist.w);
    int max_val = 0;
    for (int i = 0; i < (int)dist.data.size(); i++) {
        max_val = std::max(max_val, (int)dist.data[i]);
    }
    if (max_val > 0) {
        for (int i = 0; i < (int)dist.data.size(); i++) {
            dist_vis.data[i] = (unsigned char)(255 * (double)dist.data[i] / max_val);
        }
    } else {
        dist_vis = dist;
    }

    // 4. Picos seguros (Marcadores das moedas)
    mm::Image picos(dist.h, dist.w);
    for (int i = 0; i < (int)dist.data.size(); i++) {
        if ((double)dist.data[i] > 0.5 * max_val) {
            picos.data[i] = 255;
        }
    }

    // 5. Execução do Watershed (Ajustado para usar a nova assinatura)
    // Passamos 'opening' direto em mask, pois ela delimita o escopo de expansão das moedas
    mm::Image ws_region = mm::watershed(picos, opening, "region");
    mm::Image ws_line = mm::watershed(picos, opening, "line");

    // 6. Contagem de objetos (Ignora fundo 0)
    std::set<int> unique_labels;
    for (int i = 0; i < (int)ws_region.data.size(); i++) {
        int val = ws_region.data[i];
        if (val > 0) {
            unique_labels.insert(val);
        }
    }
    std::vector<int> labels(unique_labels.begin(), unique_labels.end());
    std::cout << "Objetos detectados: " << labels.size() << "\n";
```

```
    // 7. Anotação Final
    // Convert img_base (1-channel) to BGR for annotation
    cv::Mat img_base_mat(img_base.h, img_base.w, CV_8UC1, img_base.data.data());
    cv::Mat img_annotated_mat;
    cv::cvtColor(img_base_mat, img_annotated_mat, cv::COLOR_GRAY2BGR);

    for (size_t idx = 0; idx < labels.size(); idx++) {
        int label_id = labels[idx];

        // Build mask for this label
        cv::Mat mask_reg(img_base.h, img_base.w, CV_8UC1, cv::Scalar(0));
        int mask_sum = 0;
        for (int y = 0; y < img_base.h; y++) {
            for (int x = 0; x < img_base.w; x++) {
                if (ws_region.at(y, x) == label_id) {
                    mask_reg.at<unsigned char>(y, x) = 255;
                    mask_sum++;
                }
            }
        }
        if (mask_sum < 500) continue;

        // Compute centroid
        int sum_x = 0, sum_y = 0;
        for (int y = 0; y < img_base.h; y++) {
            for (int x = 0; x < img_base.w; x++) {
                if (ws_region.at(y, x) == label_id) {
                    sum_x += x;
                    sum_y += y;
                }
            }
        }
        int cx = sum_x / mask_sum;
        int cy = sum_y / mask_sum;

        cv::putText(img_annotated_mat, std::to_string(idx + 1), cv::Point(cx - 25, cy + 20),
                    cv::FONT_HERSHEY_SIMPLEX, 3.2, cv::Scalar(0, 255, 0), 8, cv::LINE_AA);
    }

    // Copy back to 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());

    // Desenha as linhas de separação em vermelho
    // Build 11x11 ones kernel as SE for dilation
    mm::SE kernel_11 = mm::SE::box(11);
    mm::Image mask_ann = mm::dil(ws_line, kernel_11);

    // Apply red color where mask has values > 0
    for (int y = 0; y < img_base.h; y++) {
        for (int x = 0; x < img_base.w; x++) {
            if (mask_ann.at(y, x) > 0) {
                img_annotated.at(y, x, 0) = 255;
                img_annotated.at(y, x, 1) = 0;
                img_annotated.at(y, x, 2) = 0;
            }
        }
    }
```

```
    // 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)")
```

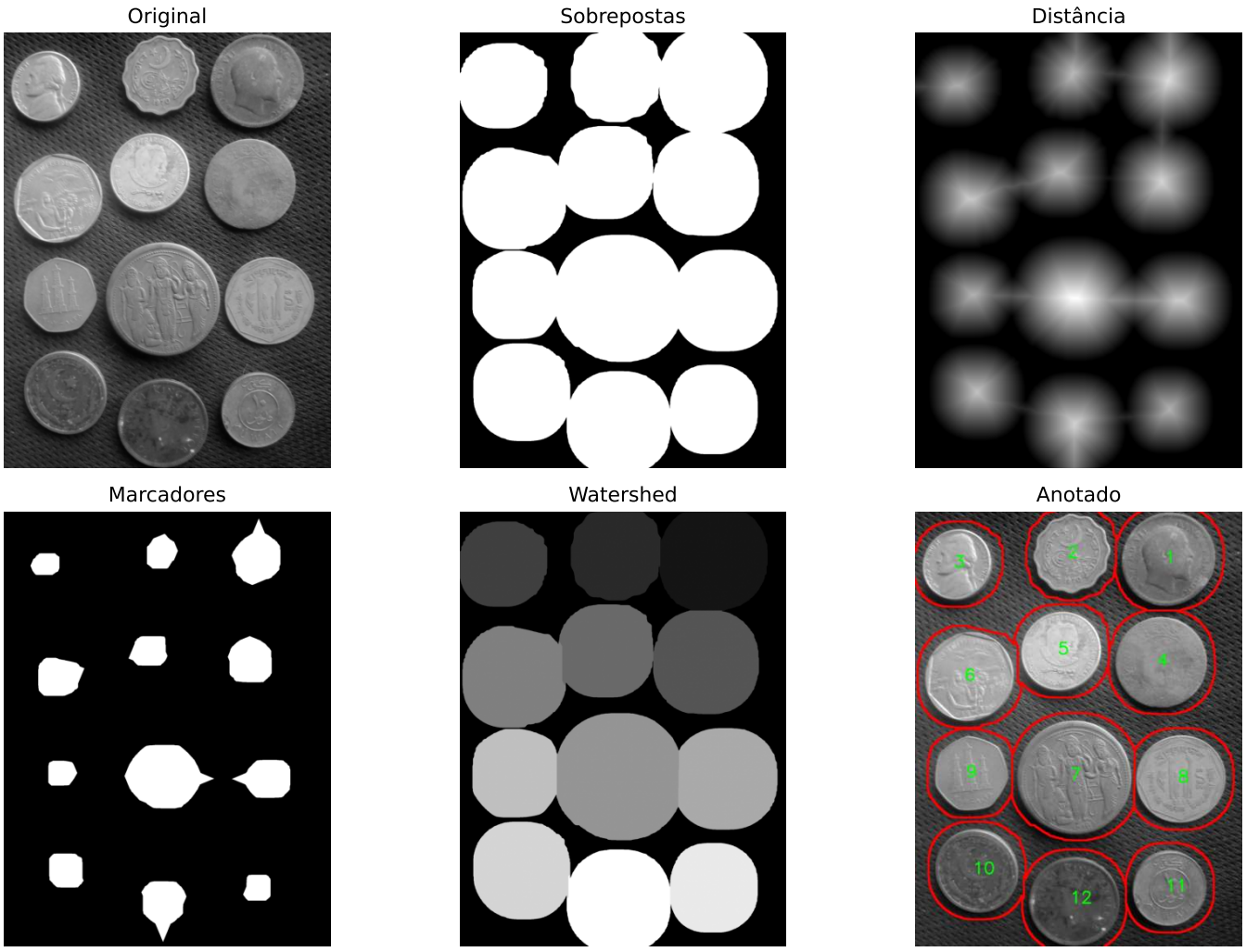


Figura 4.26: *Pipeline Watershed* para separação de moedas sobrepostas: da máscara binária dilatada até os contornos finais anotados sobre a imagem original.

4.5 Extracción de Componentes y Descriptores de Forma

Tras la segmentación y el refinamiento morfológico, el siguiente paso consiste en identificar individualmente cada objeto presente en la imagen y extraer sus propiedades geométricas. Esta etapa es fundamental para tareas de medición, clasificación y reconocimiento de patrones.

Un **componente conexo** es un conjunto maximal de píxeles pertenecientes al objeto que permanecen mutuamente conectados según una relación de conectividad previamente definida (conectividad 4 u 8). Tras la rotulación, cada componente recibe un identificador único, permitiendo que sus características sean analizadas individualmente.

OpenCV ofrece dos enfoques complementarios para este análisis, resumidos en la Tabla 4.7.

Tabla 4.7: Comparación entre los enfoques basados en componentes conexos y en contornos.

	`connectedComponentsWithStats`	`findContours`
Devuelve	etiqueta por píxel y estadísticas por componente	secuencia de puntos que describe el borde
Descriptores directos	área, *bounding box* y centroide	perímetro, forma y jerarquía
Objetos en contacto	tiende a fusionar regiones conectadas	tiende a producir un único contorno externo
Uso típico	conteo, filtrado y rotulación	análisis geométrico y descriptores de forma

4.5.1 Etiquetado y Estadísticas de Componentes

`mm::label0` asigna una etiqueta a cada componente conexo. A partir de la imagen etiquetada se obtienen, mediante un barrido, el **área** (conteo de píxeles) y la **caja delimitadora** de cada objeto (Figura 4.27). El `connectedComponentsWithStats` de OpenCV, que devuelve estas estadísticas ya calculadas, y las anotaciones coloreadas sobre la imagen quedan en el rastro de Python.

```
%%writefile tmp/fig_04_componentes.cpp
#define MM_OUT "tmp/fig_04_componentes.png"
//| label: fig-04-componentes
//| fig-cap: "Componentes conexos extraídos após o *pipeline* CLAHE → Otsu → limpeza
morfológica. Cada objeto é colorido com cor distinta e anotado com sua área em pixels."
//| echo: true
//| output: true
#include "morph.hpp"
#include <opencv2/opencv.hpp>
#include <iostream>
#include <vector>
#include <iomanip>
#include <cstring>
#include <numeric>
#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 e img_final são fornecidos automaticamente

    // 1. Rotulagem e estatísticas
    cv::Mat mask_final(img_final.h, img_final.w, CV_8UC1, img_final.data.data());
    cv::Mat labels_mat, stats_mat, centroids_mat;
    int n = cv::connectedComponentsWithStats(mask_final, labels_mat, stats_mat, centroids_mat,
    8, CV_32S);

    // Converter labels para mm::Image
    mm::Image labels_img(labels_mat.rows, labels_mat.cols);
    std::memcpy(labels_img.data.data(), labels_mat.data, labels_img.data.size());

    // 2. Coloração dos componentes - paleta FIXA (gerada uma vez com
    // np.random.seed(4)) para a trilha C++ reproduzir cor a cor sem
    // depender do gerador do numpy. Se n exceder len(PALETA), cicla.
    const std::vector<std::array<unsigned char, 3>> PALETA = {
        {0, 0, 0}, {224, 82, 193}, {233, 155, 73}, {190, 247, 244},
        {103, 94, 179}, {51, 154, 59}, {137, 96, 232}, {250, 243, 205},
```

```
        {100, 141, 208}, {228, 187, 163}, {202, 108, 214}, {131, 105, 104},
        {86, 153, 81}
    };

    mm::Image img_colored(img_final.h, img_final.w, 3);
    mm::Image img_annotated = img_colored;

    for (int y = 0; y < img_final.h; y++) {
        for (int x = 0; x < img_final.w; x++) {
            int label = labels_img.at(y, x);
            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];
        }
    }
    img_annotated = img_colored;

    // 3. Tabela de descritores
    std::cout << "Componentes detectados (excluindo fundo): " << (n - 1) << "\n";
    std::cout << 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 = static_cast<int>(centroids_mat.at<double>(i, 0));
        int cy = static_cast<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 no image
        cv::Mat ann_mat(img_annotated.h, img_annotated.w, CV_8UC3, img_annotated.data.data());
        std::string texto = std::to_string(i) + ": " + std::to_string(area);
        for (const auto& [cor, esp] : std::vector<std::pair<cv::Scalar, int>>{
            {cv::Scalar(0, 0, 0), 10}, {cv::Scalar(255, 0, 0), 5}}) {
            cv::putText(ann_mat, texto, cv::Point(cx - 150, cy + 18),
                        cv::FONT_HERSHEY_SIMPLEX, 2.0, cor, esp, cv::LINE_AA);
        }
        std::memcpy(img_annotated.data.data(), ann_mat.data, img_annotated.data.size());
    }

    // Conversões cv::Mat → mm::Image para exibição
    mm::Image img_colored_mm(img_colored.h, img_colored.w, 3);
    std::memcpy(img_colored_mm.data.data(), img_colored.data.data(), img_colored.data.size());

    mm::show(std::vector<mm::Image>{img_coins_gray, img_final, img_colored_mm, 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)")
```

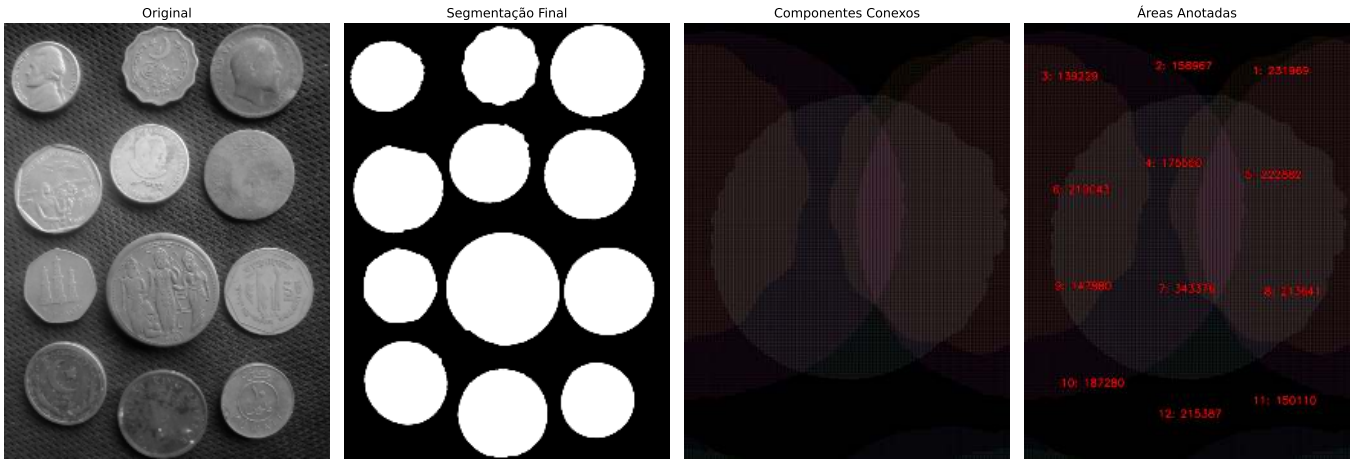


Figura 4.27: Componentes conexos extraídos após o *pipeline* CLAHE → Otsu → limpeza morfológica. Cada objeto é colorido com cor distinta e anotado com sua área em pixels.

4.5.2 Descriptores de Forma

Los descriptores basados en **contorno vectorial** — perímetro (`arcLength`), área del polígono (`contourArea`), circularidad, momentos y aproximación poligonal (`approxPolyDP`) — dependen del rastreo de frontera (`findContours`), ausente en `morph.hpp`. Solo permanecen en la ruta de Python. En la ruta de C++, el **gradiente morfológico** (`mm::gradm`) resalta el contorno de cada objeto (Figura 4.28).

```
%%writefile tmp/fig_04_contornos.cpp
#define MM_OUT "tmp/fig_04_contornos.png"
// Compile: g++ -std=c++17 -o programa programa.cpp -DMM_USE_OPENCV `pkg-config --cflags
--libs opencv4`
#include "morph.hpp"
#include <opencv2/opencv.hpp>
#include <iostream>
#include <vector>
#include <cmath>
#include <iomanip>

int main() {
// [pdi:state-io] auto-generated - do not edit by hand
mm::Image img_coins_gray = mm::_read_state("tmp/state/img_coins_gray_8.png");
mm::Image img_final = mm::_read_state("tmp/state/img_final_55.png");
// [pdi:state-io:end]

    //| label: fig-04-contornos
    //| fig-cap: "Contornos extraídos com *findContours*. Cada moeda é anotada com sua
    circularidade - valores próximos de 1 confirmam forma circular."
    //| echo: true
    //| output: true

    // Bridge mm::Image to cv::Mat for OpenCV operations
    cv::Mat img_final_mat(img_final.h, img_final.w,
                          img_final.channels == 1 ? CV_8UC1 : CV_8UC3,
                          img_final.data.data());

    std::vector<std::vector<cv::Point>> contornos;
    std::vector<cv::Vec4i> hierarquia;
    cv::findContours(img_final_mat, contornos, cv::RETR_EXTERNAL, cv::CHAIN_APPROX_SIMPLE);

    // Bridge mm::Image to cv::Mat for OpenCV operations
    cv::Mat img_coins_gray_mat(img_coins_gray.h, img_coins_gray.w,
```

```
                            img_coins_gray.channels == 1 ? CV_8UC1 : CV_8UC3,
                            img_coins_gray.data.data());

    cv::Mat img_contornos_mat;
    cv::cvtColor(img_coins_gray_mat, img_contornos_mat, cv::COLOR_GRAY2BGR);
    cv::Mat img_circulares_mat = img_contornos_mat.clone();

    std::cout << "Contornos detectados: " << contornos.size() << "\n";
    std::cout << std::setw(4) << "ID" << std::setw(8) << "Área"
              << std::setw(10) << "Perímetro" << std::setw(14) << "Circularidade" << "\n";
    std::cout << std::string(42, '-') << "\n";

    int id = 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) << id << std::setw(8) << static_cast<int>(area)
                  << std::setw(10) << std::fixed << std::setprecision(1) << perim
                  << std::setw(14) << std::setprecision(3) << circ << "\n";

        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);

        std::vector<std::pair<cv::Scalar, int>> cor_esp = {
            {cv::Scalar(0, 0, 0), 8}, {cv::Scalar(255, 0, 0), 3}
        };
        for (const auto& ce : cor_esp) {
            cv::putText(img_circulares_mat, std::to_string(circ).substr(0,
std::to_string(circ).find('.') + 3),
                        cv::Point(cx - 80, cy + 15),
                        cv::FONT_HERSHEY_SIMPLEX, 3.6, ce.first, ce.second, cv::LINE_AA);
        }
        id++;
    }

    // Build result images and copy back to mm::Image
    mm::Image out_contornos(img_contornos_mat.rows, img_contornos_mat.cols,
img_contornos_mat.channels());
    std::memcpy(out_contornos.data.data(), img_contornos_mat.data, out_contornos.data.size());

    mm::Image out_circulares(img_circulares_mat.rows, img_circulares_mat.cols,
img_circulares_mat.channels());
    std::memcpy(out_circulares.data.data(), img_circulares_mat.data,
out_circulares.data.size());

    mm::show(
        std::vector<mm::Image>{img_final, out_contornos, out_circulares},
        MM_OUT,
        std::vector<std::string>{"Segmentação Final", "Contornos", "Circularidade"},
        3
    );

    return 0;
}
```

```
Overwriting tmp/fig_04_contornos.cpp
```

```
!g++ -I. -std=c++17 -DMM_USE_OPENCV -I/usr/include/opencv4 tmp/fig_04_contornos.cpp -o
tmp/fig_04_contornos -lopencv_imgproc -lopencv_imgcodecs -lopencv_core \
  && ./tmp/fig_04_contornos \
  && test -f "tmp/fig_04_contornos.png" \
  || echo "  mm::show não gravou tmp/fig_04_contornos.png"
```

```
Contornos detectados: 12
  ID   ÁreaPerímetro Circularidade
----------------------------------------
   1  214626    1769.6         0.861
   2  149480    1465.1         0.875
   3  186570    1654.9         0.856
   4  147252    1458.6         0.870
   5  212885    1756.3         0.867
   6  342424    2232.5         0.863
   7  209282    1767.7         0.842
   8  222108    1809.7         0.852
   9  174854    1616.4         0.841
  10  138602    1471.5         0.804
  11  158306    1538.7         0.840
  12  231181    1847.4         0.851
[1] Segmentação Final
[2] Contornos
[3] Circularidade
```

```
try:
    mm.show(mm.read("tmp/fig_04_contornos.png"), figsize=(18, 6))
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + " tmp/fig_04_contornos.png
    (ver a versao Python)")
```

Figura 4.28: Contornos extraídos con *findContours*. Cada moeda é anotada con sua circularidade — valores próximos de 1 confirmam forma circular.

4.5.3 Conexión con la Detección de Objetos Moderna

Los descriptores extraídos en las secciones anteriores — especialmente *bounding boxes*, centroides, áreas y medidas de forma — establecen un puente natural entre la segmentación morfológica clásica y los sistemas modernos de detección de objetos. Aunque las técnicas estudiadas en este capítulo utilizan operaciones sobre píxeles y regiones segmentadas, muchas de las representaciones producidas son directamente compatibles con los formatos empleados en modelos contemporáneos de visión por computadora.

Los detectores basados en aprendizaje profundo, como la familia YOLO (*You Only Look Once*) (REDMON, 2016), operan directamente sobre imágenes en color y producen, para cada objeto detectado, una *bounding box* descrita por el centro (cx, cy) y las dimensiones (w, h), además de una clase y una puntuación de confianza. Esta representación comparte la misma estructura geométrica básica obtenida por `connectedComponentsWithStats`, aunque es producida por un modelo aprendido y no mediante segmentación explícita.

La Figura 4.29 ilustra cómo las *bounding boxes* obtenidas por morfología pueden exportarse en el formato YOLO para componer conjuntos de datos utilizados en el entrenamiento o la evaluación de detectores.

```
%%writefile tmp/fig_04_bbox.cpp
#define MM_OUT "tmp/fig_04_bbox.png"
// Compile with: g++ -std=c++17 -o program program.cpp -I. -L. -lmorph -lopencv_core
-lopencv_imgproc -lopencv_imgcodecs -lopencv_highgui
#include "morph.hpp"
#include <opencv2/opencv.hpp>
#include <iostream>
#include <iomanip>
#include <string>
#include <vector>
#include <fstream>
#include <sstream>
#include <cstring>
#include <filesystem>

//| 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

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 labels, stats, centroids;
    int n = cv::connectedComponentsWithStats(
        cv::Mat(img_final.h, img_final.w, CV_8UC1, img_final.data.data()),
        labels, stats, centroids, 8, CV_32S
    );

    int H_img = img_coins_gray.h;
    int W_img = img_coins_gray.w;

    // Convert mm::Image to cv::Mat for operations
    cv::Mat gray_mat(H_img, W_img, 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(9) << "cx_n" << std::setw(9) << "cy_n"
              << std::setw(9) << "w_n" << std::setw(9) << "h_n" << "  ← formato YOLO\n";
    std::cout << std::string(54, '-') << "\n";

    std::vector<std::string> yolo_linhas;
```

```
    for (int i = 1; i < n; i++) {
        int x0 = stats.at<int>(i, cv::CC_STAT_LEFT);
        int y0 = stats.at<int>(i, cv::CC_STAT_TOP);
        int w  = stats.at<int>(i, cv::CC_STAT_WIDTH);
        int h  = stats.at<int>(i, cv::CC_STAT_HEIGHT);

        double cx_n = (x0 + w / 2.0) / W_img;
        double cy_n = (y0 + h / 2.0) / H_img;
        double w_n  = w / (double)W_img;
        double h_n  = h / (double)H_img;

        std::ostringstream 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::setw(9) << std::fixed <<
std::setprecision(4)
                  << cx_n << std::setw(9) << cy_n << std::setw(9) << w_n << std::setw(9) <<
                  h_n << "\n";

        cv::rectangle(img_bbox, cv::Point(x0, y0), cv::Point(x0 + w, y0 + h),
                      cv::Scalar(0, 255, 0), 4);
        cv::putText(img_bbox, "moeda", cv::Point(x0 + 8, y0 + 60),
                    cv::FONT_HERSHEY_SIMPLEX, 3.8, cv::Scalar(255, 0, 0), 5, cv::LINE_AA);
    }

    // Exportar arquivo de anotação no formato YOLO
    std::ofstream f("moedas.txt");
    for (size_t 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";

    // Convert 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)")
```

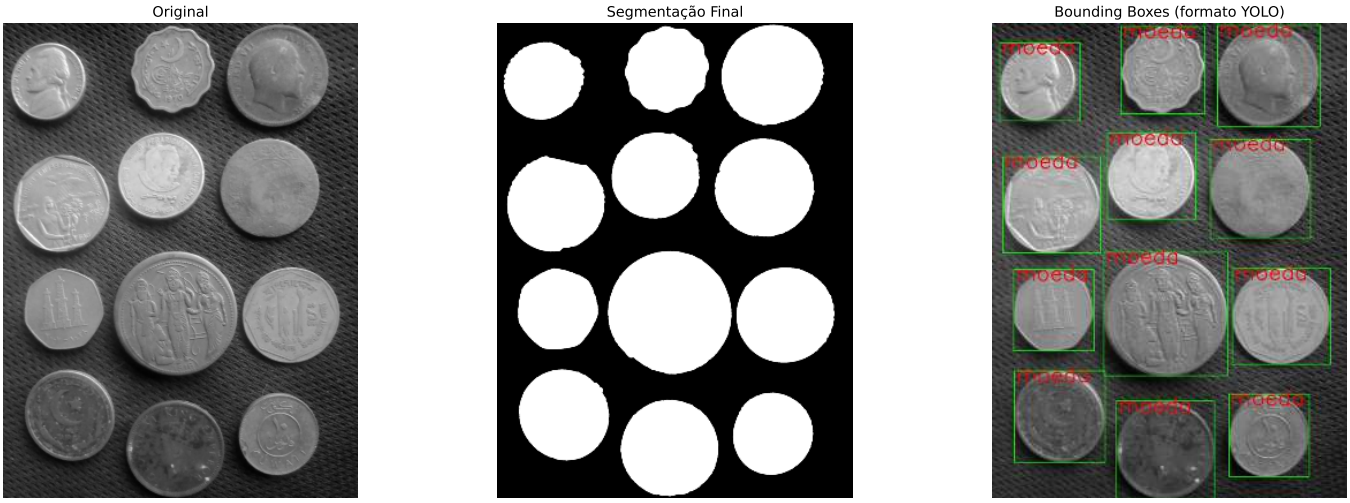


Figura 4.29: *Bounding boxes* derivadas dos componentes conexos sobrepostas à imagem original. As anotações são exportadas no formato YOLO (*classe cx cy w h*), com coordenadas normalizadas para o intervalo [0,1].

El formato YOLO almacena cada objeto en una línea que contiene cinco campos:

```
clase cx cy w h
```

donde (cx, cy) representa el centro de la *bounding box* y (w, h) sus dimensiones. Todos los valores geométricos se normalizan al intervalo $[0, 1]$ con respecto al ancho y al alto de la imagen. La **clase** es un identificador entero asociado a una categoría definida por el conjunto de datos (por ejemplo, `0 → moneda`). Cuando hay múltiples categorías —como moneda de oro (`0`), moneda de plata (`1`) y disco plástico (`2`)— basta con asignar el identificador correspondiente a cada objeto antes de la exportación, manteniendo exactamente el mismo formato de anotación. En el ejemplo anterior, esta información se almacenó en el archivo `monedas.txt`.

El flujo presentado en este capítulo —segmentación → etiquetado → extracción de *bounding boxes*— corresponde conceptualmente a la etapa de **anotación** (*labeling*) empleada en la construcción de conjuntos de entrenamiento para detectores modernos. Herramientas especializadas, como Label Studio y Roboflow, automatizan este proceso en imágenes complejas, pero la lógica fundamental sigue siendo la misma: asociar a cada objeto una región de interés y una clase. En escenarios controlados, con fondo uniforme y objetos bien separados, las técnicas morfológicas pueden incluso generar anotaciones automáticamente o servir como punto de partida para el etiquetado manual, reduciendo significativamente el esfuerzo de construcción del conjunto de datos. En aplicaciones reales más complejas, sin embargo, la validación humana sigue siendo necesaria para garantizar la calidad de las anotaciones.

i Evaluación: IoU (*Intersection over Union*)

Una forma sencilla de evaluar la calidad de una segmentación consiste en compararla con una máscara de referencia (*ground truth*). La métrica más utilizada para este propósito es la **IoU** (*Intersection over Union*):

$$\text{IoU} = \frac{|A \cap B|}{|A \cup B|} \tag{4.16}$$

donde A representa la segmentación producida por el algoritmo y B la segmentación de referencia.
El valor de la IoU varía entre 0 y 1. Cuanto mayor sea el valor, mayor será la superposición entre las máscaras. Una IoU igual a 1 indica una correspondencia perfecta entre la segmentación obtenida y la referencia.
La misma métrica también se utiliza ampliamente en la detección de objetos, aplicándose a las *bounding boxes* previstas y anotadas. En esta área, valores de IoU superiores a 0,5 se adoptan frecuentemente

como criterio mínimo para considerar una detección correcta.

Más Allá de la Morfología

Las técnicas estudiadas en este capítulo segmentan objetos explorando la conectividad espacial, las operaciones morfológicas y el relieve topográfico. Existen, sin embargo, enfoques alternativos basados en la agrupación de características, como el algoritmo *k-means*, los modelos de mezcla Gaussiana (GMM) y métodos más recientes basados en aprendizaje profundo. Estas técnicas se retomarán en la Parte II del libro, dedicada a la Visión Computacional.

4.6 Resumen

En este capítulo se presentaron las principales técnicas de segmentación y morfología matemática, concluyendo el estudio del PDI en el dominio espacial.

- **Preprocesamiento y umbralización:** La combinación entre ecualización adaptativa CLAHE y el método de Otsu mostró ser eficaz para inducir separación bimodal en el histograma y simplificar la binarización de imágenes con iluminación no uniforme.
- **Erosión y dilatación:** Operadores morfológicos fundamentales basados en la búsqueda de mínimos y máximos locales en una vecindad definida por el elemento estructurante B. Son operadores duales por el complemento y fueron implementados mediante las funciones `mm::ero` y `mm::dil`.
- **Apertura y cierre:** Composiciones de erosión y dilatación que permiten eliminar ruidos, suavizar contornos y rellenar pequeñas lagunas, preservando la estructura global de los objetos.
- **Reconstrucción morfológica:** Proceso geodésico iterativo que propaga un marcador dentro de los límites impuestos por una máscara, constituyendo la base de operadores como `mm::clohole` y `mm::edgeoff`.
- **Pipeline de limpieza binaria:** Flujo consolidado compuesto por CLAHE → Otsu → apertura → `mm::clohole` → apertura restringida → `mm::edgeoff`, produciendo máscaras adecuadas para análisis cuantitativo.
- **Morfología en tonos de gris:** Extensión algebraica basada en mínimos y máximos ponderados, viabilizando operadores como gradiente morfológico y filtros *top-hat* para realce de estructuras locales.
- **Transformada de Distancia y Watershed:** La Transformada de Distancia permitió generar marcadores automáticos para el algoritmo *watershed*, posibilitando la separación de objetos adyacentes o parcialmente superpuestos.
- **Componentes conexos y descriptores:** La rotulación de regiones (`mm::label0`) y la extracción de contornos permitieron calcular descriptores geométricos como área, centroide, perímetro, circularidad y *bounding boxes*.
- **Conexión con Visión Computacional Moderna:** Las *bounding boxes* extraídas por morfología fueron exportadas en el formato YOLO, evidenciando el vínculo entre técnicas clásicas de segmentación y sistemas modernos de detección de objetos.

El Capítulo 5 introducirá técnicas de procesamiento en el **dominio de la frecuencia**, abordando la **Transformada de Fourier**, filtrado espectral y los fundamentos de la **compresión de imágenes**, incluyendo DCT, JPEG y *wavelets*.

4.7 🤖 Uso de Gemini Notebook como Tutor Complementario

En esta edición, se incentiva el uso de **Gemini Notebook** como herramienta complementaria de aprendizaje. Basado en inteligencia artificial, el sistema utiliza exclusivamente los documentos proporcionados por el autor como fuente de conocimiento, produciendo respuestas alineadas con el contenido y el enfoque adoptado a lo largo de este capítulo.

🎓 Estudia con el Tutor Inteligente

🚀 ACCEDER A Gemini Notebook: CAPÍTULO 04

🌐 Idioma y Lenguaje de Programación

El proyecto de este capítulo en Gemini Notebook fue construido únicamente con el texto en **portugués** y los ejemplos de código en **Python**. Si estás estudiando con la edición en inglés o francés, o siguiendo la ruta en C++, las respuestas del tutor pueden no corresponder exactamente a la versión que estás leyendo.

⚠️ Aviso sobre Contenido Generado por IA

Aunque es una herramienta valiosa de apoyo al estudio, Gemini Notebook puede eventualmente producir respuestas incompletas, imprecisas o incorrectas. Se recomienda validar la información consultando el material del capítulo, libros, artículos científicos y otras fuentes académicas confiables. Siempre que sea posible, ejecuta y experimenta con los ejemplos prácticos presentados a lo largo del texto para consolidar la comprensión de los conceptos.

4.8 Lista de Ejercicios

1. **(10%)** Implemente manualmente el criterio de Otsu sin utilizar `mm::threshold`. Calcule la varianza entre clases $\sigma_B^2(T)$ para todos los umbrales $T \in [0, 255]$ usando `mm::hist`, identifique el umbral óptimo T^* y compare el resultado con el valor obtenido por OpenCV. Grafique σ_B^2 en función de T y resalte el punto de máximo.

2. **(15%)** Aplique umbralización adaptativa con bloques de tamaño 11, 31 y 51 a una imagen que contenga iluminación no uniforme. Compare los resultados con la umbralización global de Otsu y discuta las ventajas y limitaciones de cada enfoque.

3. **(15%)** Ejecute el *pipeline* de watershed de la imagen de monedas variando el umbral aplicado a la Transformada de Distancia (0.3, 0.5 y 0.7 veces el valor máximo). Explique cómo este parámetro influye en la generación de los marcadores, la separación de objetos adyacentes y la ocurrencia de sobre-segmentación.

4. **(15%)** Utilizando `mm::drawImg`, construya una demostración visual paso a paso de la erosión de una imagen binaria 7×7 con elemento estructurante cuadrado 3×3. Para cada posición analizada, indique si el elemento estructurante está completamente contenido en el objeto y justifique el valor asignado al píxel de salida.

5. **(15%)** Demuestre experimentalmente la dualidad entre erosión y dilatación verificando la identidad $(A \ominus B)^c = A^c \oplus \hat{B}$ utilizando `mm::ero`, `mm::dil` y `mm::bnot`. Calcule la diferencia píxel a píxel entre los dos lados de la ecuación y presente el resultado utilizando `mm::histImg` o una visualización equivalente.

6. **(15%)** Implemente manualmente el gradiente morfológico utilizando únicamente `mm::ero` y `mm::dil`, comparando el resultado con `mm::gradm(img, B)`. Evalúe el efecto de diferentes elementos estructurantes (cuadrado 3×3, disco 5×5 y línea 1×9) sobre la detección de bordes.

7. **(15%)** Construya un *pipeline* completo para el conteo y clasificación de monedas por tamaño (pequeña, mediana y grande) utilizando área y circularidad como descriptores. Genere una máscara de referencia (*ground truth*) manualmente y calcule la métrica IoU (*Intersection over Union*) para evaluar la calidad de la segmentación. Presente los resultados en una tabla y mediante visualizaciones producidas con `mm::show`.

Referencias del Capítulo

La fundamentación teórica de este capítulo se basa en las siguientes obras:

- Gonzalez (2018) para los conceptos de segmentación, umbralización de Otsu, Transformada de Distancia, *watershed*, morfología matemática y descriptores de forma.
- Matheron (1975) y Serra (1982) para la fundamentación teórica original, formulación algebraica y desarrollo de la Morfología Matemática.
- Szeliski (2022) para segmentación basada en regiones, etiquetado de componentes conexos, *watershed* basado en marcadores y evaluación de segmentación mediante la métrica IoU.
- Bradski (2008) para la utilización práctica de la biblioteca OpenCV, incluyendo funciones como `mm::dist`, `mm::watershed`, `mm::label0` y extracción de contornos.
- Redmon (2016) para la introducción a los detectores modernos de la familia YOLO y su relación con descriptores geométricos como *bounding boxes* extraídas por segmentación.
- Singh (2024) para la construcción de laberintos complejos basados en ciclos hamiltonianos sobre mosaicos cuasicristalinos, utilizados como ejemplo de aplicación de la Transformada de Distancia Geodésica y de algoritmos de búsqueda de caminos.
- Zampirolli (2025) para la implementación de los operadores morfológicos, transformadas geodésicas y resolución de laberintos por propagación de distancias en dominios restringidos.

4.9 💻 Parte Práctica con Ejercicios de Programación

Esta lista transforma los conceptos del Capítulo 4 en una ruta práctica de segmentación y morfología matemática. Los EPs comienzan con umbralización y avanzan hasta etiquetado y descriptores de componentes, siempre con matrices pequeñas para que cada píxel pueda verificarse a mano.

> ❗ Regla común de los EPs morfológicos
>
> En las operaciones con vecindad, **no haga padding**. Para cada píxel, evalúe únicamente las posiciones del elemento estructurante que caen dentro del dominio de la imagen. Esta es la misma idea de las implementaciones didácticas en `morph.py`, como `mm::dil0`, `mm::ero0`, `mm::dil1` y `mm::label0`: la vecindad se recorta por el dominio válido de la imagen.

🎯 Objetivo de este cuaderno

El cuaderno permite desarrollar, validar, organizar y probar soluciones de **Ejercicios de Programación (EPs)** en entornos interactivos, como Colab, con los mismos casos de prueba de Moodle, copiándolos allí solo al momento de registrar la nota oficial.

Download

Descargue `morph.py` y `testsuite.py` ejecutando la celda a continuación:

```
import os, urllib.request

os.makedirs("tmp/state", exist_ok=True)  # artefatos de build del trayecto C++ (.cpp, binario,
PNGs)

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

```
# El kernel es Python incluso en el trayecto C++: `mm` (morph.py) es usado por los
# simuladores, por la exhibición de las figuras que el binario C++ genera y por el
# estado mm::Image entre celdas. cpp=True descarga también el trayecto compilado
# (morph.hpp + stb_image*.h), usado en el #include de las celdas %%writefile *.cpp.
import config
config.setup(testsuite=True, cpp=True)
from morph import mm
from testsuite import TestSuite
```

```
Entorno listo. Morph: 1.1.9 | OpenCV: 5.0.0 | TestSuite: 1.1.2
```

Ejecutando las pruebas

Para evaluar las pruebas, ejecute `TestSuite("EP04_01.extensión").run()` en una nueva celda, reemplazando la extensión por la del lenguaje utilizado (`.py`, `.java`, `.c`, `.cpp`, `.js` o `.r`). El sistema descarga los casos de prueba de GitHub, ejecuta el programa y calcula la nota automáticamente.

Para probar código Python directamente, sin guardar archivo, use `run_code(codigo)` pasando el código como *string* en una variable `codigo`:

```
codigo = """
from morph import mm
# ... su código aquí ...
"""
TestSuite("EP04_01").run_code(codigo)
```

4.9.1 EP04_01 Umbralización Global por Umbral Fijo

En los **escáneres de documentos** y en los **sistemas de lectura de códigos de barras**, la primera etapa del procesamiento siempre consiste en separar lo que es "objeto" (tinta, texto, barras) de lo que es "fondo" (papel, embalaje). La **umbralización global** hace exactamente esto: compara cada píxel con un único umbral T y decide, en tiempo real, si pertenece a la clase clara o a la clase oscura. Es el operador de segmentación más simple — y aun así, está detrás de buena parte de los *pipelines* industriales de inspección visual. Ver en Figura 4.30 una simulación de este EP.

4.9.1.1 📋 Directrices de Implementación

1. **Dimensiones:** Leer los enteros L (filas) y C (columnas).
2. **Umbral:** Leer el entero T (umbral de decisión).
3. **Datos:** Leer los valores enteros de la matriz original fila a fila.
4. **Mapeo:** Para cada píxel p, calcular el nuevo valor mediante la ecuación:

$$p' = \begin{cases} 255, & \text{si } p > T \\ 0, & \text{si } p \leq T \end{cases}$$

5. **Salida:** Mostrar la matriz binarizada con dimensiones $L \times C$.

4.9.1.2 📌 Restricciones Computacionales

- **Binarización:** La salida contiene **solo** los valores 0 o 255.
- **Comparación estricta:** El criterio usa $> T$ (los píxeles iguales a T se convierten en fondo).
- **Tipo:** El resultado final debe ser entero.
- **Observación:** Este EP sigue la convención de OpenCV (`cv2.THRESL_BINARY`): solo los píxeles con valor **mayor que** T se convierten en blancos (`255`); los píxeles con valor **igual a** T permanecen negros (`0`).

4.9.1.3 🧠 Fundamentación Teórica

Parámetro	Tipo	Impacto Visual
T **pequeño**	Entero	La mayoría de los píxeles se vuelven blancos
T **grande**	Entero	La mayoría de los píxeles se vuelven negros
T **bien elegido**	Entero	Separa nítidamente objeto y fondo

4.9.1.4 📦 Especificación de Entrada y Salida (VPL)

Entrada:

- Línea 1: Entero L.
- Línea 2: Entero C.
- Línea 3: Entero T.
- Líneas siguientes: Elementos enteros de la matriz original.

Salida:

- Matriz binarizada en L filas y C columnas, valores 0 o 255 separados por espacio.

4.9.1.5 📌 Ejemplos

Entrada	Salida	Observación
2 4 100 0 99 100 180 255 30 120 80	0 0 0 255 255 0 255 0	$T = 100$: solo los píxeles con valor mayor que 100 se vuelven blancos; por eso, 99 y 100 se vuelven negros.
1 3 0 0 50 255	0 255 255	$T = 0$: solo los píxeles con valor estrictamente mayor que 0 se vuelven blancos.

```
%%writefile EP04_01.cpp
// your solution
```

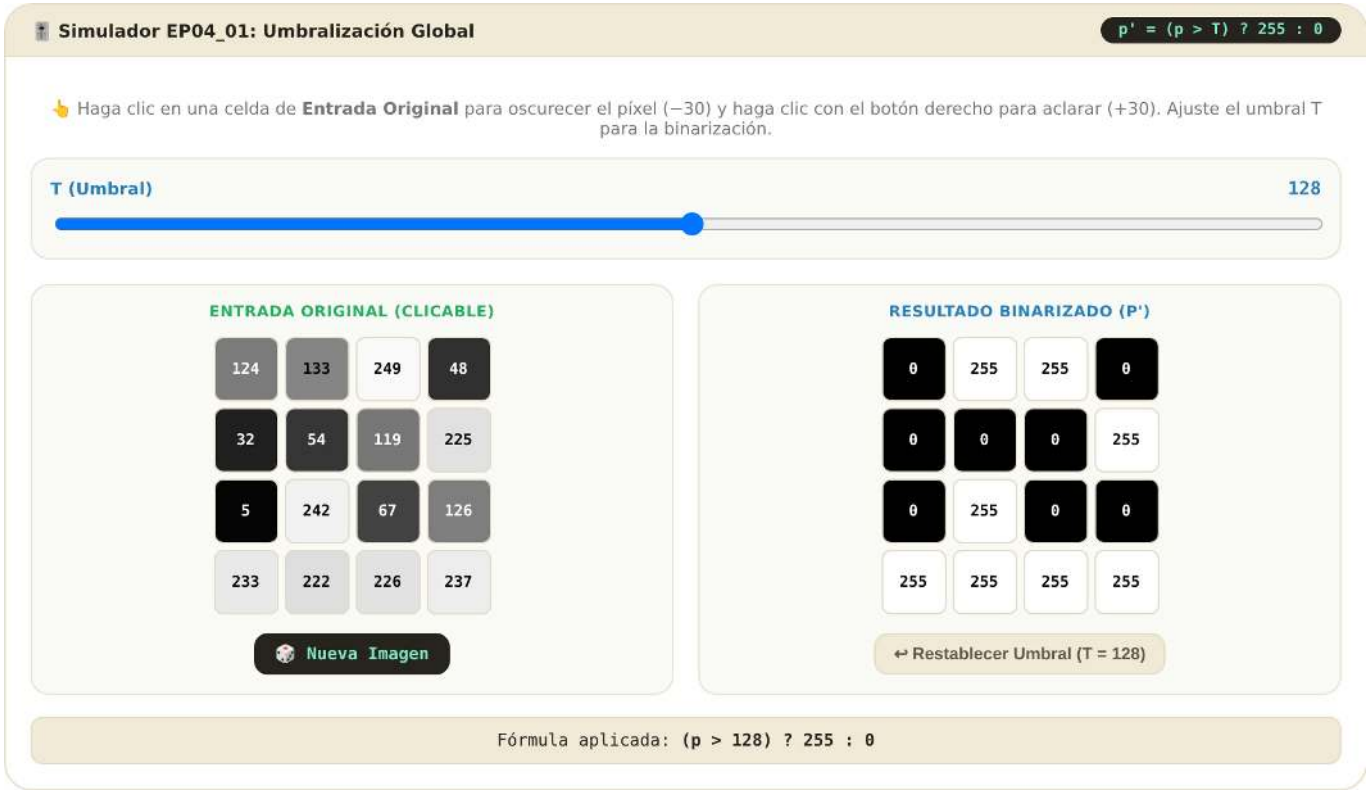
```
Overwriting EP04_01.cpp
```

```
TestSuite("EP04_01.cpp").run()
```

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

4.9.2 EP04_02 📊 Umbralización Automática de Otsu

Elegir manualmente el umbral T funciona cuando la iluminación es estable, pero en **microscopía digital** y en **inspección de láminas de sangre**, cada muestra tiene un contraste diferente — un umbral fijo fallaría de imagen en imagen. El **método de Otsu** resuelve esto encontrando, por sí solo, el umbral que **maximiza la separación estadística** entre las dos clases de píxeles, haciendo la segmentación automática y adaptativa. Ver en Figura 4.31 una simulación de este EP.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.es/cap04/sim-ep0401-limiar.html

Figura 4.30: Simulador EP04_01: Umbralización Global por Umbral Fijo (p' = (p > T) ? 255 : 0)

4.9.2.1 📋 Directrices de Implementación

1. **Dimensiones:** Leer los enteros L (filas) y C (columnas).
2. **Datos:** Leer los valores enteros de la matriz original fila por fila.
3. **Histograma:** Construir el histograma $h[i]$, $i = 0, \ldots, 255$, contando cuántos píxeles tienen valor i.
4. **Búsqueda del umbral:** Para cada candidato T de 1 a 255, calcular la **varianza entre clases**:

$$\sigma_B^2(T) = \frac{n_0 \cdot n_1}{N^2} (m_0 - m_1)^2$$

   donde n_0, n_1 son las cantidades de píxeles con valor $< T$ y $\geq T$, m_0, m_1 son sus medias, y $N = L \times C$.
5. **Elección:** El umbral óptimo T^* es el que maximiza $\sigma_B^2(T)$ (en caso de empate, mantener el **primero** encontrado).
6. **Aplicación:** Binarizar la imagen usando T^*, aplicando:

$$p' = \begin{cases} 255, & \text{si } p > T^* \\ 0, & \text{si } p \leq T^* \end{cases}$$

4.9.2.2 📌 Restricciones Computacionales

- **Candidatos válidos:** Ignorar T que deje $n_0 = 0$ o $n_1 = 0$ (clase vacía).
- **Empate:** Mantener siempre el **primer** T que alcanzó el valor máximo de σ_B^2.
- **Tipo:** T^* y la matriz de salida deben ser enteros.
- **Convención OpenCV:** La binarización sigue `cv2.THRESL_BINARY`; los píxeles con valor exactamente igual a T^* se vuelven negros.

4.9.2.3 🧠 Fundamentación Teórica

Concepto	Significado	Impacto
$\sigma_B^2(T)$ **alta**	Clases bien separadas en T	T es un buen candidato a umbral
Histograma bimodal	Dos "picos" distintos	Otsu encuentra el valle entre ellos
Histograma unimodal	Un único "pico"	Otsu aún elige *algún* T, pero la segmentación es poco fiable

4.9.2.4 📦 Especificación de Entrada y Salida (VPL)

Entrada:

- Línea 1: Entero L.
- Línea 2: Entero C.
- Líneas siguientes: Elementos enteros de la matriz original.

Salida:

- Matriz binarizada en L filas y C columnas, valores 0 o 255.

4.9.2.5 📌 Ejemplos

Entrada	Salida	Observación
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	Histograma bimodal claro: 12 y 200
1 2 10 250	0 250	Solo dos valores: T^* queda en el mayor

```
%%writefile EP04_02.cpp
// your solution
```

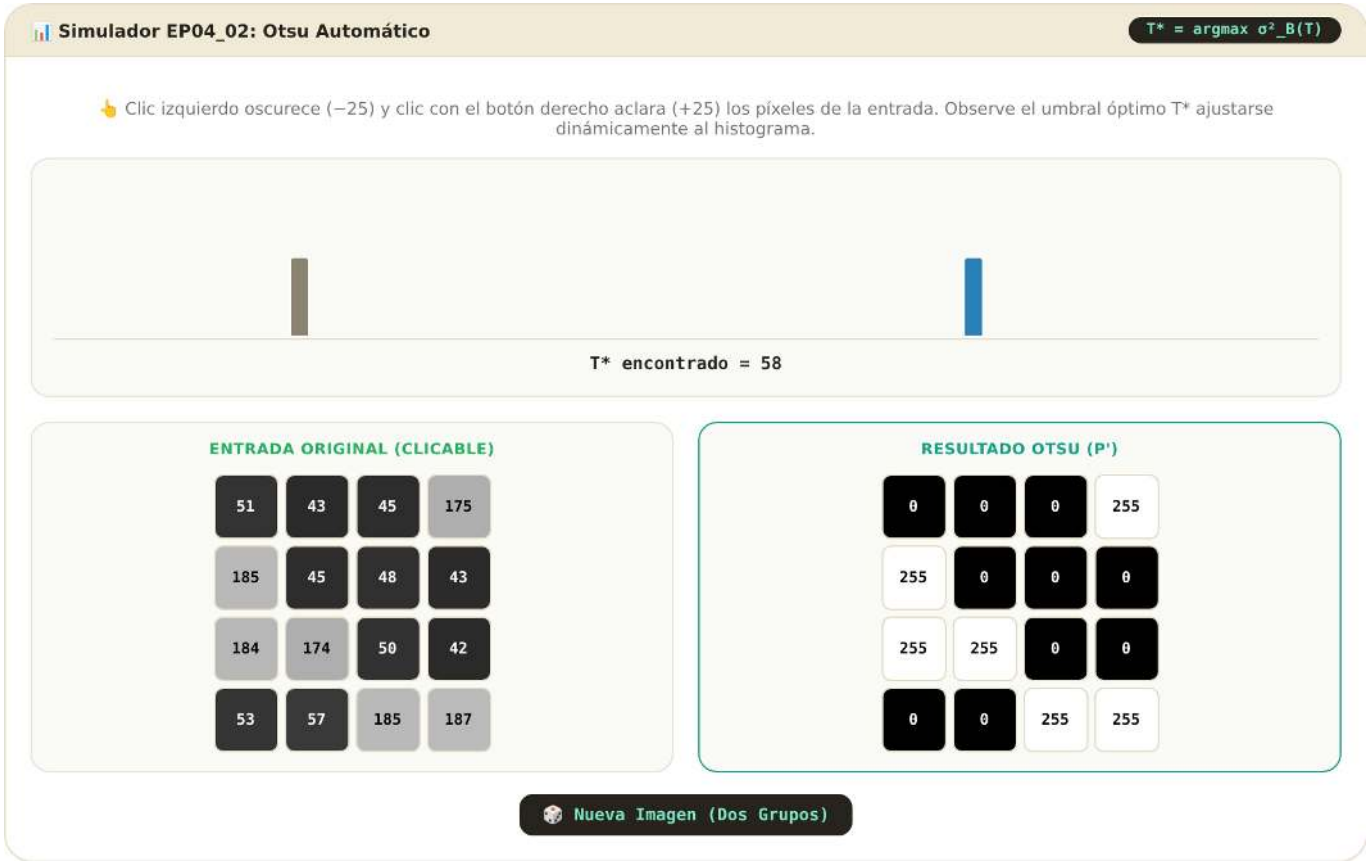
```
Overwriting EP04_02.cpp
```

```
TestSuite("EP04_02.cpp").run()
```

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

4.9.3 EP04_03 🌱 Dilatación Binaria Plana (mm.dil0)

En **microscopía de partículas** y en **OCR de placas de automóvil desgastadas**, los trazos finos o discontinuos deben "engrosarse" para que el reconocimiento funcione. La **dilatación morfológica** hace exactamente eso: expande regiones claras usando un elemento estructurante B — la misma operación implementada en `morph.py` como `mm::dil0(f, B)`, usada cuando B es **plano** (sin pesos, solo 0/1). Ver en Figura 4.32 una simulación de este EP.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.es/cap04/sim-ep0402-otsu.html

Figura 4.31: Simulador EP04_02: Limiarización Automática de Otsu (T* = argmax ²_B(T))

4.9.3.1 📋 Directrices de Implementación

1. **Dimensiones de la imagen:** Leer los enteros L (filas) y C (columnas) de f.
2. **Dimensiones de B:** Leer los enteros L_B (filas) y C_B (columnas) del elemento estructurante.
3. **Elemento estructurante:** Leer la matriz B con valores 0 o 1, fila a fila.
4. **Datos:** Leer la matriz f (la imagen original), fila a fila.
5. **Reflexión:** Construir B_{ref}, la versión de B reflejada en 180° (filas y columnas invertidas) — exactamente como hace `mm::dil0` internamente.
6. **Vecindad sin padding:** Para cada píxel (y, x), recorrer las posiciones (by, bx) de B_{ref} centradas en (y, x), usando el desplazamiento

$$v_y = y + by + o_y, \quad v_x = x + bx + o_x, \quad o_y = -\frac{L_B}{2} + 0{,}5, \quad o_x = -\frac{C_B}{2} + 0{,}5$$

   Descartar todo (v_y, v_x) fuera de $[0, L) \times [0, C)$ — **no rellenar con ceros**.
7. **Mapeo:** Calcular cada píxel de salida como el **máximo** entre $f(y, x)$ y todos los $f(v_y, v_x)$ válidos cuya posición correspondiente en B_{ref} vale 1:

$$g(y, x) = \max\left(f(y, x), \max_{\substack{(v_y, v_x) \text{ válido} \\ B_{ref}(by, bx)=1}} f(v_y, v_x)\right)$$

8. **Salida:** Mostrar la matriz g con dimensiones $L \times C$.

4.9.3.2 📌 Restricciones Computacionales

- **Sin padding:** Nunca inventar vecinos fuera de la imagen; usar solo los que existen realmente.
- **Reflexión obligatoria:** B debe reflejarse antes de aplicarse (es lo que diferencia `mm::dil0` de una simple búsqueda de máximo).
- **Robustez de borde:** Si ninguna posición válida de $B_{ref} = 1$ cae dentro del dominio para un píxel dado, este **mantiene su valor original**.

4.9.3.3 🧠 Fundamentación Teórica

Concepto	Significado	Impacto Visual
Dilatación	$g \geq f$ siempre (extensiva)	Las regiones claras crecen, los huecos oscuros se encogen
B **mayor**	Vecindad más amplia	Crecimiento más agresivo
Reflexión de B	$B_{ref}(y,x) = B(-y,-x)$	Garantiza la definición formal de Minkowski de la dilatación

4.9.3.4 📦 Especificación de Entrada y Salida (VPL)

Entrada:

- Línea 1: Entero L.
- Línea 2: Entero C.
- Línea 3: Entero L_B.
- Línea 4: Entero C_B.
- Siguientes L_B líneas: elementos enteros (0 o 1) de la matriz B.
- Siguientes L líneas: elementos enteros de la matriz f.

Salida:

- Matriz g en L filas y C columnas, valores enteros separados por espacio.

4.9.3.5 📌 Ejemplos

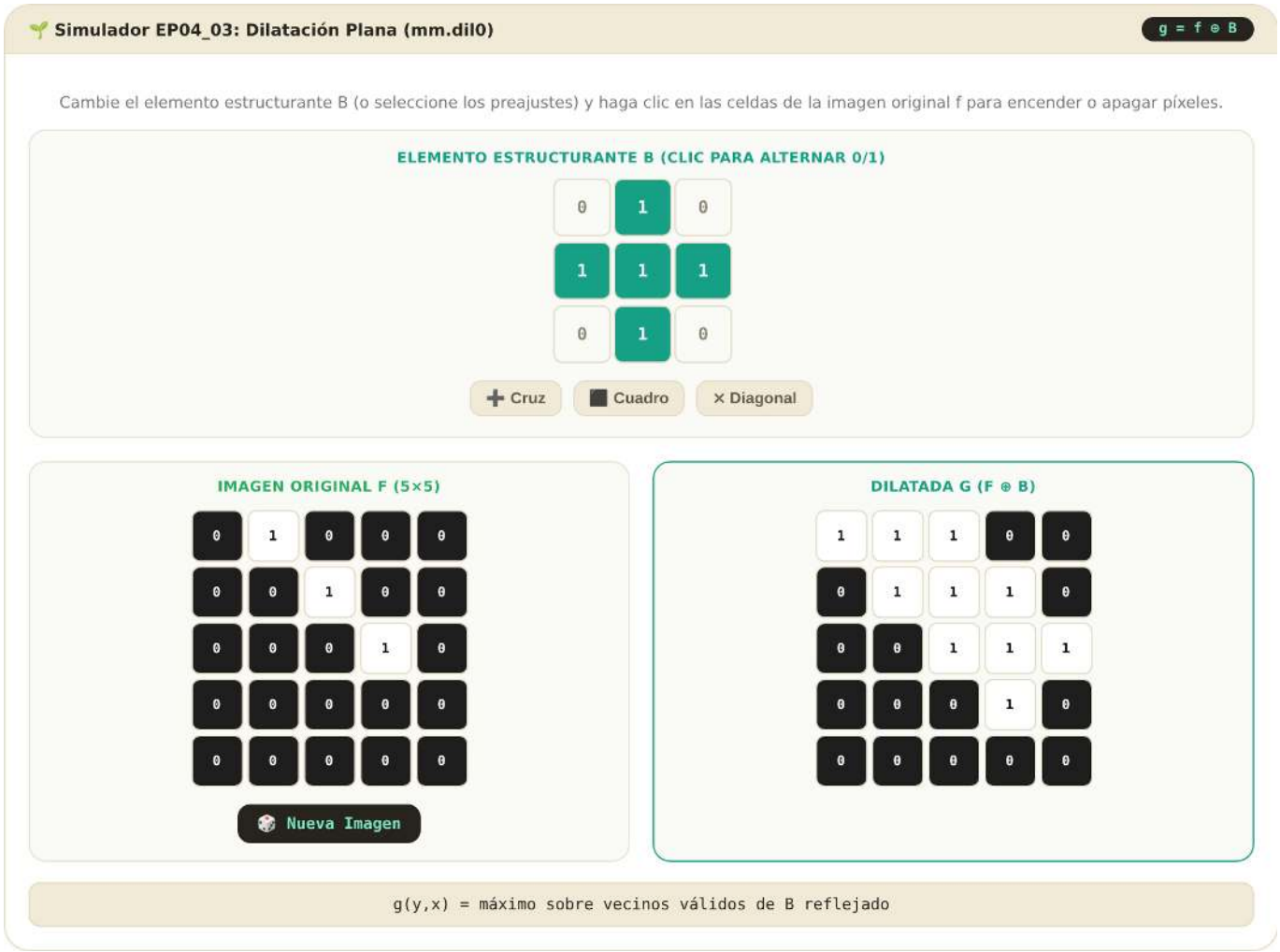
Entrada	Salida	Observación
3 3 3 3 0 1 0 1 1 1 0 1 0 0 0 0 0 9 0 0 0 0	0 9 0 9 9 9 0 9 0	B en cruz simétrico: punto aislado se expande en cruz
1 4 1 3 1 1 1 10 200 5 80	200 200 200 80	B horizontal: cada píxel “atrae” el máximo de los vecinos de la fila

```
%%writefile EP04_03.cpp
// your solution
```

```
Overwriting EP04_03.cpp
```

```
TestSuite("EP04_03.cpp").run()
```

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



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.es/cap04/sim-ep0403-dilatacao.html

Figura 4.32: Simulador EP04_03: Dilatación Binaria Plana (g = f B)

4.9.4 EP04_04 🪨 Erosión Binaria Plana (mm.ero0)

Si la dilatación engrosa, la **erosión** afina. En **sistemas de conteo de células**, se utiliza para **separar células que se tocan**: al "comer" los bordes de cada región, las conexiones finas entre objetos desaparecen incluso antes de que se realice cualquier conteo. En `morph.py`, esta es la operación `mm::ero0(f, B)` — la **dual** exacta de la dilatación, y la única de las dos que **no** refleja el elemento estructurante. Ver en Figura 4.33 una simulación de este EP.

4.9.4.1 📋 Directrices de Implementación

1. **Dimensiones de la imagen:** Leer los enteros L (filas) y C (columnas) de f.
2. **Dimensiones de B:** Leer los enteros L_B (filas) y C_B (columnas) del elemento estructurante.
3. **Elemento estructurante:** Leer la matriz B con valores 0 o 1, fila a fila.
4. **Datos:** Leer la matriz f (la imagen original), fila a fila.
5. **Vecindad sin padding (¡sin reflexión!):** Para cada píxel (y, x), recorrer las posiciones (by, bx) de B **en el orden original** (sin reflejar), usando el mismo desplazamiento del EP04_03:

$$v_y = y + by + o_y, \quad v_x = x + bx + o_x, \quad o_y = -\tfrac{L_B}{2} + 0{,}5, \quad o_x = -\tfrac{C_B}{2} + 0{,}5$$

Descartar todo (v_y, v_x) fuera de $[0, L) \times [0, C)$. 6. **Mapeo:** Calcular cada píxel de salida como el **mínimo** entre $f(y, x)$ y todos los $f(v_y, v_x)$ válidos cuya posición correspondiente en B vale 1:

$$g(y, x) = \min\left(f(y, x), \min_{\substack{(v_y, v_x)\ \text{válido} \\ B(by, bx)=1}} f(v_y, v_x)\right)$$

7. **Salida:** Mostrar la matriz g con dimensiones $L \times C$.

4.9.4.2 📌 Restricciones Computacionales

- **Sin reflexión:** A diferencia de la dilatación, B se usa **exactamente como se lee** — reflejarlo aquí sería un error conceptual grave.
- **Sin padding:** Los vecinos fuera de la imagen simplemente se ignoran, nunca se tratan como 0.
- **Robustez de borde:** Si ninguna posición válida de $B = 1$ cae dentro del dominio, el píxel mantiene su valor original.

4.9.4.3 🧠 Fundamentación Teórica

Concepto	Significado	Impacto Visual
Erosión	$g \leq f$ siempre (anti-extensiva)	Las regiones claras se encogen, el ruido puntual desaparece
Dualidad	$\text{ero}(f, B) = -\text{dil}(-f, B_{ref})$	La erosión y la dilatación son "espejos" matemáticos
B **más grande**	Erosión más agresiva	Los objetos finos desaparecen por completo

4.9.4.4 📦 Especificación de Entrada y Salida (VPL)

Entrada:

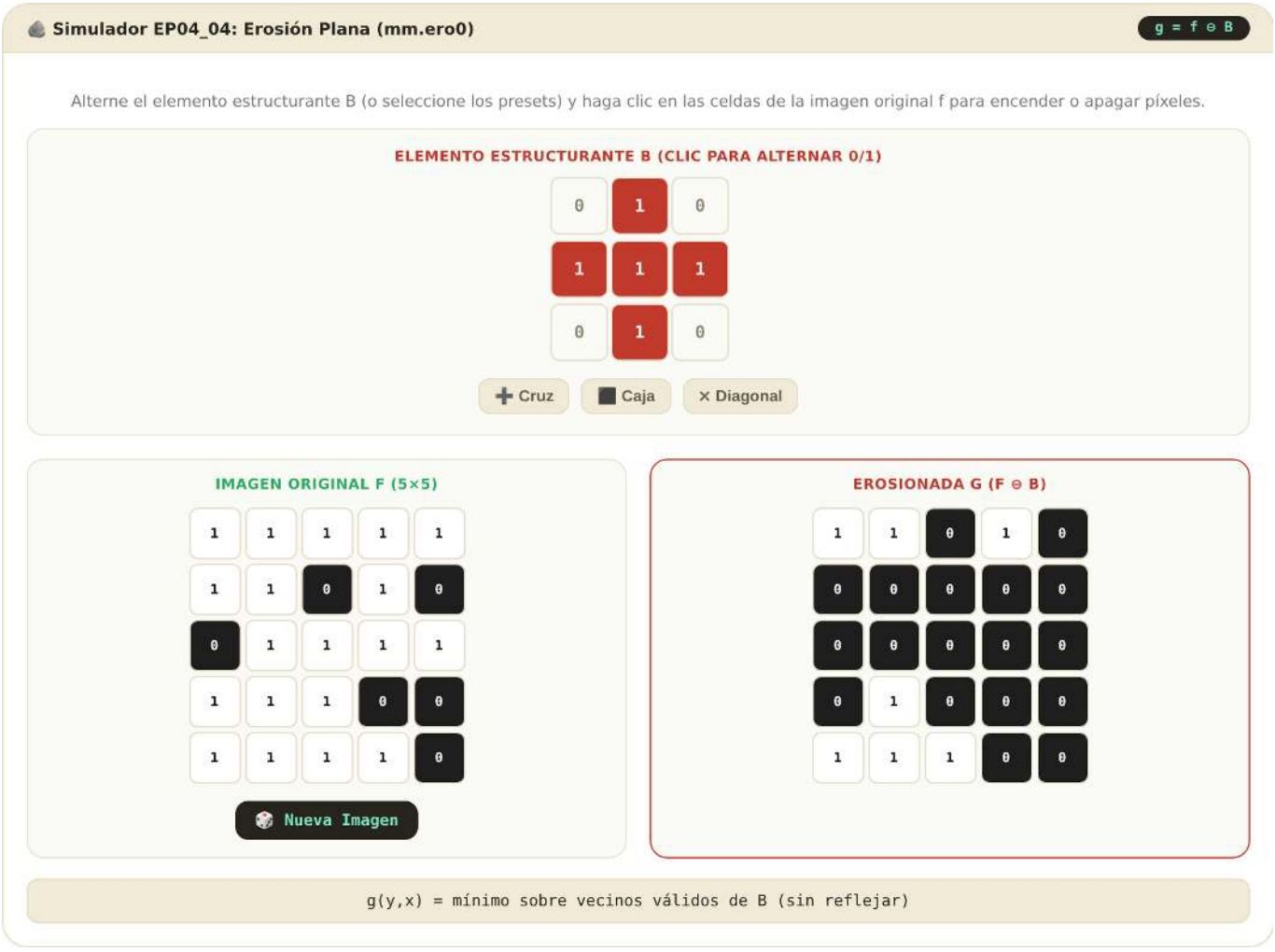
- Línea 1: Entero L.
- Línea 2: Entero C.
- Línea 3: Entero L_B.
- Línea 4: Entero C_B.
- Siguientes L_B líneas: elementos enteros (0 o 1) de la matriz B.
- Siguientes L líneas: elementos enteros de la matriz f.

Salida:

- Matriz g en L filas y C columnas, valores enteros separados por espacios.

4.9.4.5 📌 Ejemplos

Entrada	Salida	Observación
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	El “agujero” central (0) se propaga en cruz
1 4 1 3 1 1 1 10 200 5 80	10 5 5 80	B horizontal: cada píxel “extrae” el mínimo de los vecinos de la fila



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.es/cap04/sim-ep0404-erosao.html

Figura 4.33: Simulador EP04_04: Erosión Binaria Plana (g = f B)

```
%%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 Morfológica (Eliminación de Ruido)

Las imágenes capturadas por **sensores de bajo costo**, como los de drones agrícolas, suelen venir salpicadas de pequeños puntos de ruido — píxeles aislados que no representan nada real. Aplicar erosión seguida de dilatación con el **mismo** elemento estructurante produce la **apertura**: esta "limpia" puntos y protuberancias finas, pero devuelve al objeto principal prácticamente su tamaño original. Es la combinación clásica utilizada en **preprocesamiento de imágenes de satélite** antes de cualquier conteo de área plantada. Ver en Figura 4.34 una simulación de este EP.

4.9.5.1 📋 Directrices de Implementación

1. **Dimensiones de la imagen:** Leer los enteros L (filas) y C (columnas) de f.
2. **Dimensiones de B:** Leer los enteros L_B (filas) y C_B (columnas) del elemento estructurante.
3. **Elemento estructurante:** Leer la matriz B con valores 0 o 1, fila a fila.
4. **Datos:** Leer la matriz binaria f (valores 0 o 1), fila a fila.
5. **Erosión:** Calcular $e = f \ominus B$, usando exactamente el algoritmo del EP04_04 (sin reflejar B, sin padding).
6. **Dilatación:** Calcular $g = e \oplus B$, usando exactamente el algoritmo del EP04_03 (reflejando B, sin padding) — pero ahora aplicado sobre e, no sobre f.
7. **Salida:** Mostrar la matriz resultante g (la **apertura** de f por B) con dimensiones $L \times C$.

4.9.5.2 📌 Restricciones Computacionales

- **Orden fijo:** Es **siempre** erosión primero, luego dilatación — el orden inverso define otro operador (cierre, del próximo EP).
- **Mismo B:** El elemento estructurante utilizado en la erosión y en la dilatación debe ser idéntico.
- **Sin padding en ninguna de las dos etapas.**

4.9.5.3 🧠 Fundamentación Teórica

Concepto	Significado	Impacto Visual
Antiextensividad	$g \subseteq f$ siempre	La apertura nunca crea un píxel nuevo, solo elimina
Idempotencia	apertura(apertura(f)) = apertura(f)	Aplicar de nuevo no cambia nada más
Puntos aislados	Menores que B	Son completamente eliminados
Núcleo del objeto	Mayor que B	Se recupera casi intacto mediante la dilatación final

4.9.5.4 📦 Especificación de Entrada y Salida (VPL)

Entrada:

- Línea 1: Entero L.
- Línea 2: Entero C.

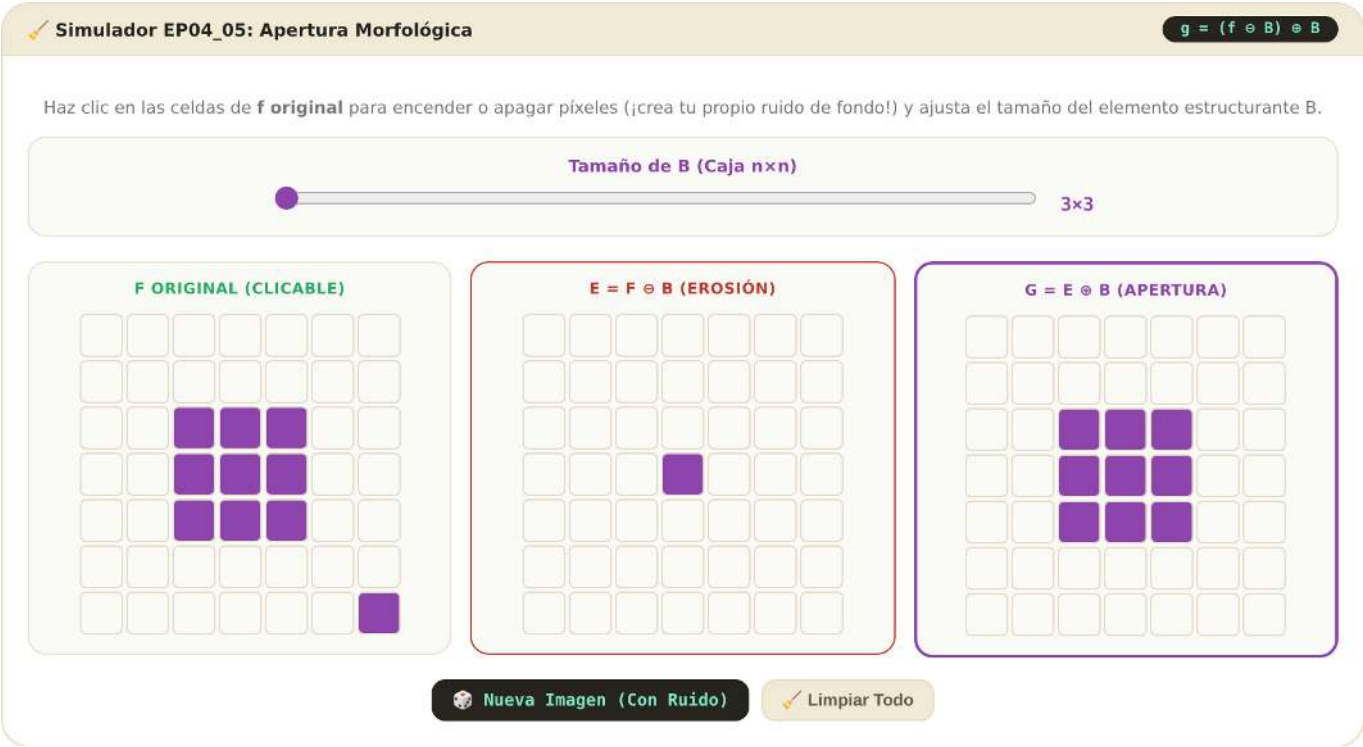
- Línea 3: Entero L_B.
- Línea 4: Entero C_B.
- Siguientes L_B líneas: elementos enteros (0 o 1) de la matriz B.
- Siguientes L líneas: elementos enteros (0 o 1) de la matriz f.

Salida:

- Matriz resultante en L filas y C columnas, valores 0 o 1.

4.9.5.5 📌 Ejemplos

Entrada	Salida	Observación
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	Los puntos aislados y la protuberancia fina desaparecen; el cuadrado central sobrevive



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.es/cap04/sim-ep0405-abertura.html

Figura 4.34: Simulador EP04_05: Apertura Morfológica (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 🧩 Cierre Morfológico (Relleno de Huecos)

En la **digitalización de huellas dactilares**, los surcos de la piel a veces se ven interrumpidos por suciedad o sequedad, creando pequeñas fallas en la curva continua que debería existir. El **cierre** —dilatación seguida de erosión con el mismo elemento estructurante— es el operador dual de la apertura: **rellena huecos pequeños y entrantes estrechos**, sin alterar significativamente el contorno externo del objeto. Es el paso estándar antes de extraer el esqueleto de una huella dactilar. Ver en Figura 4.35 una simulación de este EP.

4.9.6.1 📋 Directrices de Implementación

1. **Dimensiones de la imagen:** Leer los enteros L (líneas) y C (columnas) de f.
2. **Dimensiones de B:** Leer los enteros L_B (líneas) y C_B (columnas) del elemento estructurante.
3. **Elemento estructurante:** Leer la matriz B con valores 0 o 1, línea por línea.
4. **Datos:** Leer la matriz binaria f (valores 0 o 1), línea por línea.
5. **Dilatación:** Calcular $d = f \oplus B$, usando exactamente el algoritmo del EP04_03 (reflejando B, sin padding).
6. **Erosión:** Calcular $g = d \ominus B$, usando exactamente el algoritmo del EP04_04 (sin reflejar B, sin padding) — ahora aplicado sobre d, no sobre f.
7. **Salida:** Mostrar la matriz resultante g (el **cierre** de f por B) con dimensiones $L \times C$.

4.9.6.2 📌 Restricciones Computacionales

- **Orden fijo:** Es **siempre** dilatación primero, luego erosión — el orden inverso es la apertura del EP04_05.
- **Mismo B:** El elemento estructurante usado en la dilatación y en la erosión debe ser idéntico.
- **Sin padding en ninguna de las dos etapas.**

4.9.6.3 🧠 Fundamentación Teórica

Concepto	Significado	Impacto Visual
Extensividad	$g \supseteq f$ siempre	El cierre nunca elimina píxeles, solo añade
Idempotencia	$\text{cierre}(\text{cierre}(f)) = \text{cierre}(f)$	Aplicarlo de nuevo no cambia nada más
Huecos pequeños	Menores que B	Se rellenan completamente
Dualidad	$\text{cierre}(f) = \overline{\text{apertura}(\bar{f})}$	Es la apertura aplicada al "negativo" de la imagen

4.9.6.4 📦 Especificación de Entrada y Salida (VPL)

Entrada:

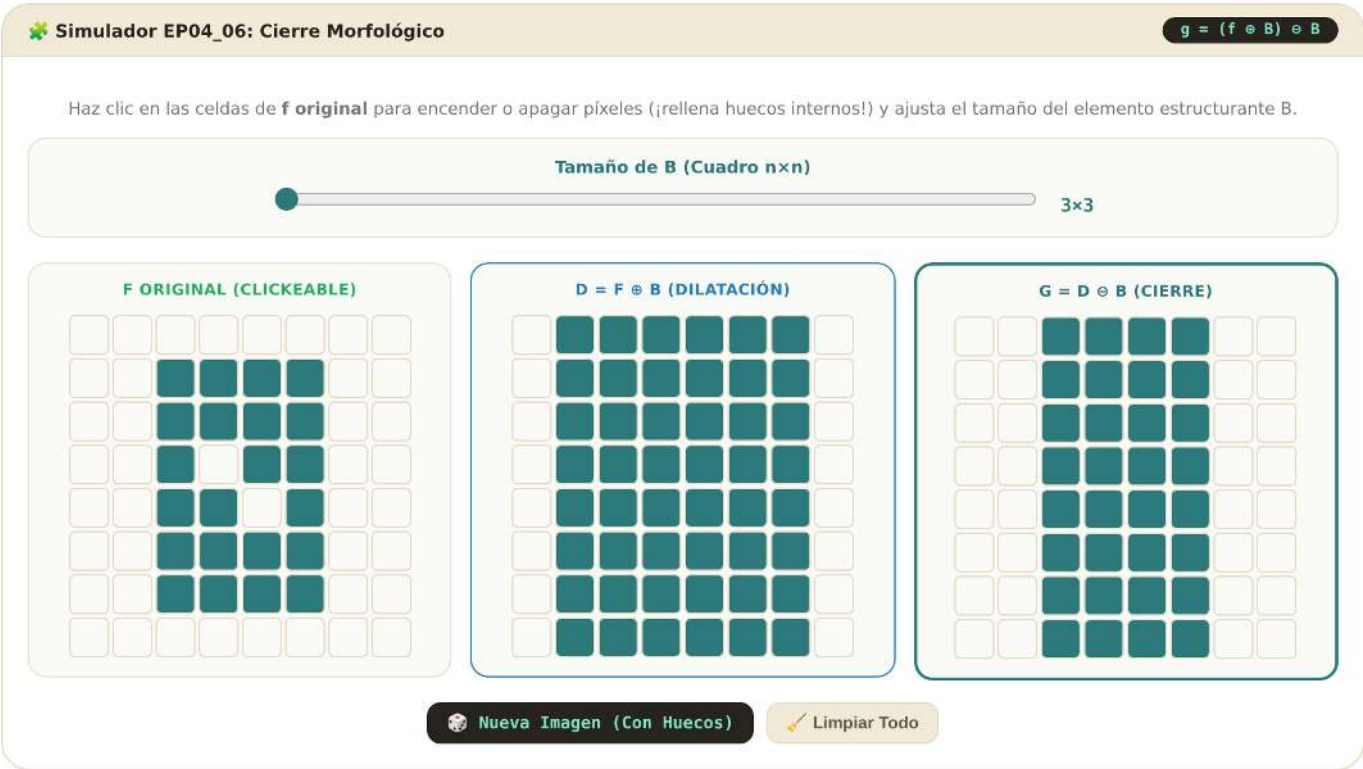
- Línea 1: Entero L.
- Línea 2: Entero C.
- Línea 3: Entero L_B.
- Línea 4: Entero C_B.
- Siguientes L_B líneas: elementos enteros (0 o 1) de la matriz B.
- Siguientes L líneas: elementos enteros (0 o 1) de la matriz f.

Salida:

- Matriz resultante en L líneas y C columnas, valores 0 o 1.

4.9.6.5 📌 Ejemplos

Entrada	Salida	Observación
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	Los dos huecos internos no adyacentes se rellenan totalmente



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.es/cap04/sim-ep0406-fechamento.html

Figura 4.35: Simulador EP04_06: Cierre Morfológico (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 Dilatación y Erosión con Pesos (mm.dil1 / mm.ero1)

Hasta ahora, el elemento estructurante solo decía "este vecino cuenta" o "no cuenta" — pero en **modelos digitales de elevación** (usados en SIG y en planificación de drenaje urbano), cada vecino debería tener un **peso diferente** dependiendo de la distancia o de la dirección del relieve. Las versiones **ponderadas** de la dilatación y de la erosión, implementadas en `morph.py` como `mm::dil1(f, b)` y `mm::ero1(f, b)`, suman (o restan) el peso de cada vecino antes de tomar el máximo (o mínimo) — generalizando todo lo realizado en los EPs anteriores. Ver en Figura 4.36 una simulación de este EP.

4.9.7.1 📋 Directrices de Implementación

1. **Dimensiones de la imagen:** Leer los enteros L (filas) y C (columnas) de f.
2. **Dimensiones de b:** Leer los enteros L_B (filas) y C_B (columnas) del elemento estructurante ponderado.
3. **Pesos:** Leer la matriz b de pesos **enteros** (pueden ser negativos, cero o positivos), fila a fila.
4. **Datos:** Leer la matriz f (la imagen original), fila a fila.
5. **Vecindario sin padding:** Para cada píxel (y, x), recorrer **todas** las posiciones (by, bx) de b (no solo donde valdría 1 — aquí **todo** peso participa), usando el mismo desplazamiento de los EPs anteriores:

$$v_y = y + by + o_y, \quad v_x = x + bx + o_x, \quad o_y = -\tfrac{L_B}{2} + 0{,}5, \quad o_x = -\tfrac{C_B}{2} + 0{,}5$$

   Descartar todo (v_y, v_x) fuera de $[0, L) \times [0, C)$.
6. **Dilatación ponderada:** Calcular

$$g_{dil}(y, x) = \max\left(f(y, x), \max_{(v_y, v_x)\ \text{válido}} \left(f(v_y, v_x) + b(by, bx)\right)\right)$$

7. **Erosión ponderada:** Calcular, **usando el mismo b y sin reflejar**:

$$g_{ero}(y, x) = \min\left(f(y, x), \min_{(v_y, v_x)\ \text{válido}} \left(f(v_y, v_x) - b(by, bx)\right)\right)$$

8. **Salida:** Mostrar **primero** la matriz g_{dil} completa, y **después** la matriz g_{ero} completa.

4.9.7.2 📌 Restricciones Computacionales

- **Ninguna de las dos refleja** b — la versión ponderada no usa reflexión, incluso en la dilatación (diferente de `mm::dil0`).
- **Todos los pesos participan:** No existe aquí el filtro "$B = 1$"; incluso el peso 0 entra en la cuenta.
- **Sin padding:** los vecinos fuera de la imagen se ignoran, nunca se rellenan virtualmente.
- **Tipo:** La salida puede contener valores negativos o mayores que 255 — **no** hay *clipping* en este EP.
- **Consejo:** Para eliminar mensajes de desbordamiento al superar los límites del tipo uint8, incluir al inicio del código:

```
import warnings
warnings.filterwarnings("ignore")
```

4.9.7.3 🧠 Fundamentación Teórica

Concepto	Significado	Impacto Visual
Peso positivo	"Empuja" el valor del vecino hacia arriba en la dilatación	Simula relieve que asciende en esa dirección
Peso negativo	Reduce la contribución del vecino	Simula distancia o atenuación direccional
Dualidad ponderada	$\text{ero1}(f, b) = -\text{dil1}(-f, b)$	La simetría entre las dos operaciones se mantiene incluso con pesos

4.9.7.4 📦 Especificación de Entrada y Salida (VPL)

Entrada:

- Línea 1: Entero L.
- Línea 2: Entero C.
- Línea 3: Entero L_B.
- Línea 4: Entero C_B.
- Siguientes L_B líneas: elementos enteros (pueden ser negativos) de la matriz b.
- Siguientes L líneas: elementos enteros de la matriz f.

Salida:

- Primero la matriz g_{dil} en L líneas y C columnas.
- A continuación la matriz g_{ero} en L líneas y C columnas.

4.9.7.5 📌 Ejemplos

Entrada	Salida	Observación
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	Peso central 2 acelera el crecimiento en la dilatación y la contracción en la erosión

```
%%writefile EP04_07.cpp
// your solution
```

```
Overwriting EP04_07.cpp
```

```
TestSuite("EP04_07.cpp").run()
```

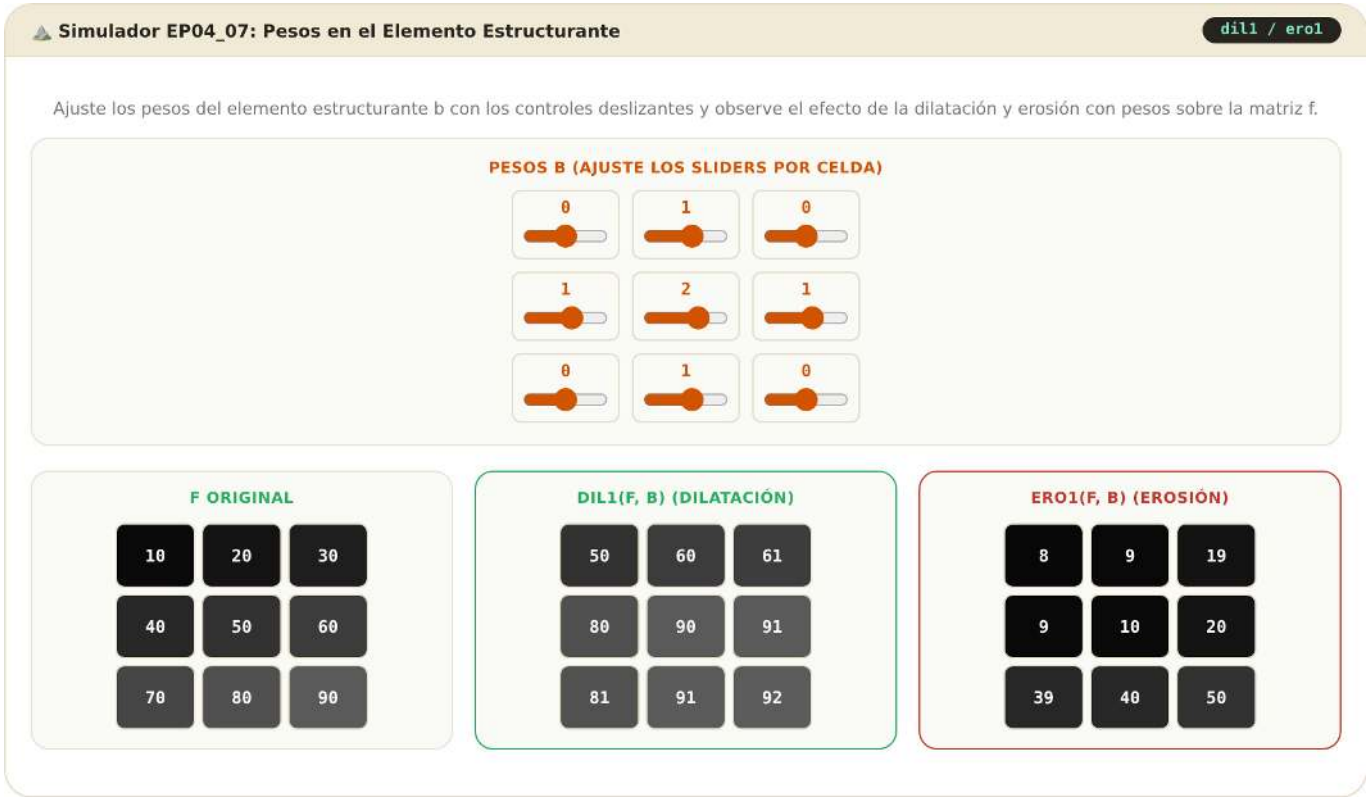
```
<IPython.core.display.HTML object>
```

4.9.8 EP04_08 🌋 Gradiente morfológico, Top-hat y Black-hat

En la **inspección automática de placas de circuito**, tres preguntas aparecen todo el tiempo: ¿dónde están los **bordes** de los componentes? ¿Qué **detalles claros y pequeños** (como puntos de soldadura) se destacan del fondo? ¿Qué **reentrancias oscuras** (como fisuras) esconde el fondo? Un único par erosión/dilatación responde a las tres: el **gradiente morfológico** evidencia contornos, el **top-hat** revela picos estrechos, y el **black-hat** revela valles estrechos — tres herramientas, una sola vecindad. Ver en Figura 4.37 una simulación de este EP.

4.9.8.1 📋 Directrices de Implementación

1. **Dimensiones de la imagen:** Leer los enteros L (filas) y C (columnas) de f.
2. **Dimensiones de B:** Leer los enteros L_B (filas) y C_B (columnas) del elemento estructurante.
3. **Elemento estructurante:** Leer la matriz B con valores 0 o 1, línea a línea.
4. **Datos:** Leer la matriz f (la imagen original, en tonos de gris), línea a línea.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.es/cap04/sim-ep0407-pesos.html

Figura 4.36: Simulador EP04_07: Dilatación y Erosión con Pesos (mm.dil1 / mm.ero1)

5. **Operadores de base:** Calcular, exactamente como en los EPs 04_03 a 04_06:
   - $d = f \oplus B$ (dilatación),
   - $e = f \ominus B$ (erosión),
   - abertura $= e \oplus B$,
   - fechamento $= d \ominus B$.
6. **Gradiente morfológico:** $\text{grad}(y, x) = d(y, x) - e(y, x)$.
7. **Top-hat:** $\text{tophat}(y, x) = f(y, x) - \text{abertura}(y, x)$.
8. **Black-hat:** $\text{blackhat}(y, x) = \text{fechamento}(y, x) - f(y, x)$.
9. **Salida:** Mostrar, **en este orden**, las tres matrices completas: gradiente, top-hat, black-hat.

4.9.8.2 📌 Restricciones Computacionales

- **Sin padding en ninguna etapa intermedia** — dilatación, erosión, apertura y cierre siguen las mismas reglas de vecindad de los EPs anteriores.
- **No hay *clipping*:** las tres salidas pueden contener cualquier valor entero (el gradiente es siempre ≥ 0, pero top-hat y black-hat también).
- **Reutilización:** d y e deben calcularse **una única vez** y reutilizarse para montar apertura, cierre y gradiente.

4.9.8.3 🧠 Fundamentación Teórica

Operador	Fórmula	Qué revela
Gradiente	$d - e$	Bordes: cero en regiones planas, alto en las transiciones
Top-hat	$f - \text{abertura}(f)$	Elementos **claros y finos**, más pequeños que B

Operador	Fórmula	Qué revela
Black-hat	fechamento$(f) - f$	Elementos **oscuros y finos**, más pequeños que B

4.9.8.4 📦 Especificación de Entrada y Salida (VPL)

Entrada:

- Línea 1: Entero L.
- Línea 2: Entero C.
- Línea 3: Entero L_B.
- Línea 4: Entero C_B.
- Siguientes L_B líneas: elementos enteros (0 o 1) de la matriz B.
- Siguientes L líneas: elementos enteros de la matriz f.

Salida:

- Matriz gradiente en L filas y C columnas.
- Matriz top-hat en L filas y C columnas.
- Matriz black-hat en L filas y C columnas.

4.9.8.5 📌 Ejemplos

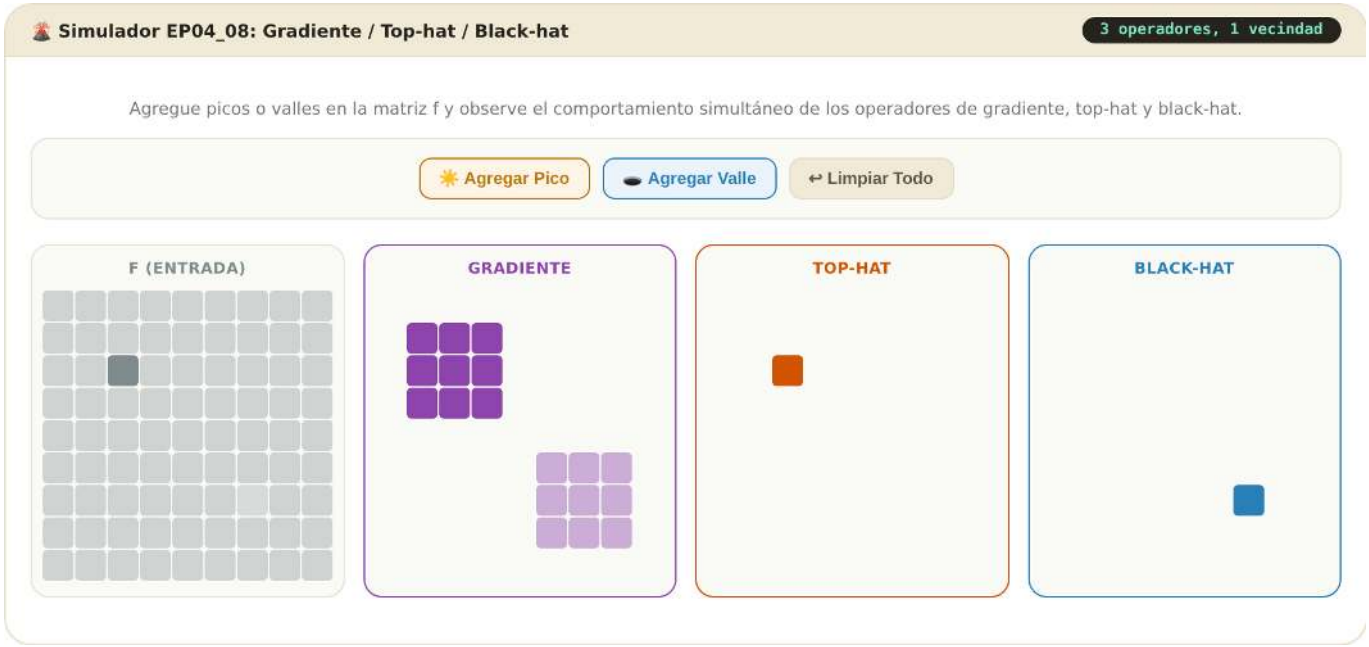
Entrada	Salida	Observación
9 9 3 3 1 1 1 1 1 1 1 1 1 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 80 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 2 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10	(gradiente: halo $3 \times 3 = 70$ en torno a $(2, 2)$ y halo $3 \times 3 = 8$ en torno a $(6, 6)$, resto 0) (top-hat: único 70 en $(2, 2)$, resto 0) (black-hat: único 8 en $(6, 6)$, resto 0)	Pico aislado se convierte en top-hat; valle aislado se convierte en black-hat; ambos aparecen en el gradiente

```
%%writefile EP04_08.cpp
// your solution
```

```
Overwriting EP04_08.cpp
```

```
TestSuite("EP04_08.cpp").run()
```

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



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.es/cap04/sim-ep0408-gradiente.html

Figura 4.37: Simulador EP04_08: Gradiente Morfológico, Top-hat y Black-hat

4.9.9 EP04_09 Transformada de Distancia y el "Centro" del Objeto

En **robótica móvil**, al planificar una ruta dentro de un pasillo, el robot quiere saber no solo *dónde* hay espacio libre, sino también **qué tan lejos** está cada punto libre de la pared más cercana. Las rutas más seguras tienden a pasar por el "centro" del pasillo, lejos de los obstáculos.

La **transformada de distancia morfológica** asigna a cada píxel un valor que representa su distancia hasta el borde más cercano, según la métrica definida por el elemento estructurante. Los píxeles cercanos al borde reciben valores bajos, mientras que los píxeles más internos reciben valores mayores. El píxel de valor máximo corresponde a la región más protegida del objeto, frecuentemente asociada a su centro morfológico.

Ver en Figura 4.38 una simulación de este EP.

4.9.9.1 📋 Directrices de Implementación

1. **Dimensiones de la imagen:** leer los enteros L (filas) y C (columnas) de la imagen f.
2. **Dimensiones de B:** leer los enteros L_B (filas) y C_B (columnas) del elemento estructurante.
3. **Elemento estructurante:** leer la matriz b, que contiene el valor 0 en el centro y valores negativos en las demás posiciones.
4. **Imagen:** leer la matriz binaria f (valores 0 o 1), fila a fila.
5. **Preparación:** multiplicar la imagen por $L \times C$, asegurando que los píxeles internos tengan un valor inicial suficientemente alto para la propagación de las distancias.
6. **Transformada de distancia:** calcular la matriz de distancias utilizando el método `mm::dist1(f,b)`.
7. **Salida:** mostrar la matriz resultante de la transformada de distancia.

4.9.9.2 📌 Restricciones Computacionales

- Utilizar la implementación de erosión ponderada proporcionada por la biblioteca.
- El elemento estructurante puede contener valores negativos arbitrarios.
- La transformada debe obtenerse mediante la aplicación iterativa de erosiones ponderadas hasta alcanzar un punto fijo.

⚠️ **Nota Crucial sobre Lectura de Matrices:** Como el elemento estructurante puede contener valores enteros negativos (por ejemplo, `-1` y `-99`), **no utilice la función `mm::readImg` para leer la matriz** b. Esa función convierte los datos al tipo `uint8`, provocando *underflow* y corrompiendo los valores negativos. Lea las

L_B filas de b manualmente utilizando el tipo estándar `int`. La imagen f puede seguir leyéndose normalmente con `mm::readImg`.

4.9.9.3 🧠 Fundamentación Teórica

Concepto	Significado	Impacto Visual
dist(y, x)	Distancia morfológica hasta el borde más cercano según la métrica definida por b	Los píxeles más internos reciben valores mayores
Valor máximo	Píxel más distante del borde	Aproxima el centro morfológico del objeto
Elemento estructurante ponderado	Define los costos de desplazamiento entre píxeles vecinos	Determina la métrica de distancia utilizada
Objetos finos	Regiones estrechas del objeto	Producen valores bajos de distancia

4.9.9.4 📦 Especificación de Entrada y Salida (VPL)

Entrada:

- Línea 1: entero L.
- Línea 2: entero C.
- Línea 3: entero L_B.
- Línea 4: entero C_B.
- Siguientes L_B líneas: elementos enteros de la matriz b.
- Siguientes L líneas: elementos binarios (0 o 1) de la matriz f.

⚠️ **Nota de implementación:** Los elementos de la matriz f (0 o 1) deben multiplicarse por **255** para generar una imagen binaria adecuada (0 y 255) antes de aplicar la Transformada de Distancia (TD).

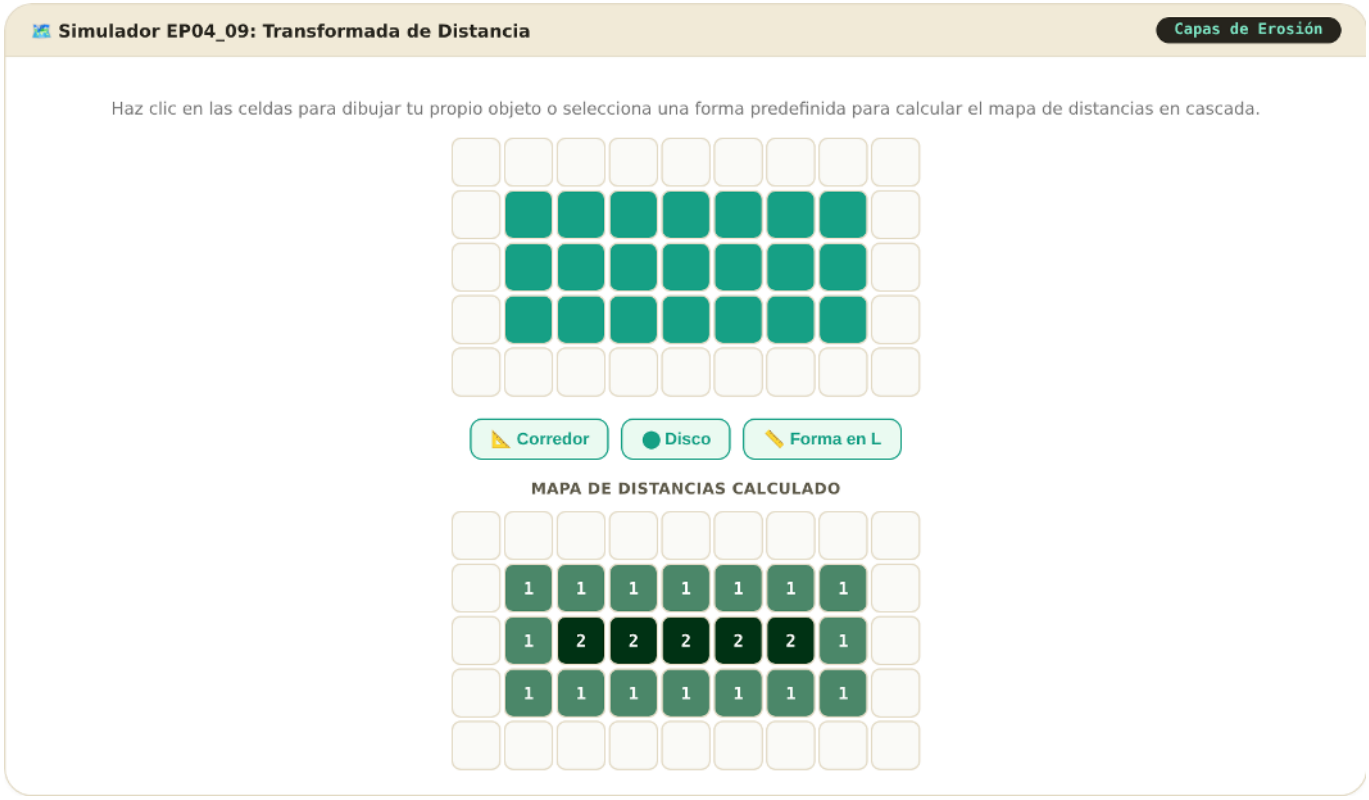
Salida:

- Matriz de la transformada de distancia en L filas y C columnas.

4.9.9.5 📌 Ejemplo

Entrada	Salida	Observación
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	Resultado de la transformada de distancia.

Nota: el valor `-99` actúa como una aproximación práctica de $-\infty$, impidiendo la propagación por las diagonales. De esta forma, solo los vecinos horizontal y vertical contribuyen a la distancia, produciendo la distancia de Manhattan.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.es/cap04/sim-ep0409-distancia.html

Figura 4.38: Simulador EP04_09: Transformada de Distância (Camadas de Erosión)

```
%%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 🪙 Separación de *Blobs*, Etiquetado y Descriptores

En una **línea de producción de monedas**, es común que las piezas se toquen entre sí en la cinta transportadora, formando una única mancha conectada en la imagen — un conteo ingenuo erraría el total. La solución clásica combina operaciones morfológicas y análisis de conectividad: primero una **erosión** reduce o rompe conexiones frágiles entre objetos, y luego el **etiquetado de componentes conectados** separa cada objeto en una región distinta. Finalmente, **descriptores geométricos** (área y caja delimitadora) resumen cada componente encontrado.

Ver en Figura 4.39 una simulación de este EP.

4.9.10.1 📋 Directrices de Implementación

1. **Dimensiones de la imagen:** leer los enteros L (filas) y C (columnas) de f.
2. **Dimensiones de B:** leer los enteros L_B (filas) y C_B (columnas) del elemento estructurante.
3. **Elemento estructurante:** leer la matriz B, que contiene valores 0 o 1, fila a fila.
4. **Datos:** leer la matriz binaria f (valores 0 o 1), fila a fila.

5. **Separación:** calcular
$$f_{ero} = f \ominus B$$
usando erosión binaria plana (como en el EP04_04), eliminando conexiones frágiles entre objetos.

6. **Etiquetado:** sobre f_{ero}, identificar componentes conectados usando conectividad definida por la vecindad B. El etiquetado debe seguir un barrido *raster*: al encontrar un píxel 1 aún no etiquetado, asignar una nueva etiqueta entera creciente a partir de 1 y propagar esa etiqueta a toda la región conectada.

7. **Descriptores:** para cada etiqueta k, calcular:
   - **Área:** número de píxeles pertenecientes a la etiqueta;
   - **Caja delimitadora:**
$$(y_{min}, x_{min}, y_{max}, x_{max})$$

8. **Salida:** mostrar el número total de etiquetas y, a continuación, una línea por etiqueta en el formato:
$$k,\ \text{área},\ y_{min},\ x_{min},\ y_{max},\ x_{max}$$

4.9.10.2 📌 Restricciones Computacionales

- La erosión debe aplicarse antes del etiquetado.
- La conectividad es fija y está definida por la vecindad anterior.
- El elemento estructurante B no interfiere en la conectividad del etiquetado.
- Sin *padding* en ninguna etapa.
- El orden de las etiquetas sigue la primera detección en el barrido *raster*.

4.9.10.3 🧠 Fundamentación Teórica

Concepto	Significado	Impacto
Puente fino	Conexión estrecha entre objetos	Puede ser eliminado por la erosión morfológica
Conectividad	Definida por el conjunto $\mathcal{N}(y, x)$	Determina qué píxeles pertenecen al mismo componente
Área	Número de píxeles por componente	Estimación directa del tamaño del objeto
Caja delimitadora	Extensión espacial de la etiqueta	Resumen geométrico del componente

4.9.10.4 📦 Especificación de Entrada y Salida (VPL)

Entrada:

- Línea 1: entero L
- Línea 2: entero C
- Línea 3: entero L_B
- Línea 4: entero C_B
- Siguientes L_B líneas: matriz B
- Siguientes L líneas: matriz f

Salida:

- Línea 1: número total de etiquetas encontradas
- Líneas siguientes:
$$k,\ \text{área},\ y_{min},\ x_{min},\ y_{max},\ x_{max}$$

```
%%writefile EP04_10.cpp
// your solution
```

Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.es/cap04/sim-ep0410-rotulacao.html

Figura 4.39: Simulador EP04_10: Separación de Blobs, Etiquetado y Descriptores

```
Overwriting EP04_10.cpp
```

```
TestSuite("EP04_10.cpp").run()
```

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

Capítulo 5

Transformadas y Compresión

En los capítulos anteriores, todas las operaciones se realizaron **en el dominio espacial**, en el que los algoritmos actúan directamente sobre los valores de intensidad de los píxeles.

En este capítulo se presentará un enfoque complementario: el **dominio de la frecuencia**, en el cual la imagen se representa mediante las variaciones espaciales de intensidad, y no solo por los valores individuales de los píxeles.

El concepto de **frecuencia espacial** describe la rapidez con la que la intensidad varía a lo largo de la imagen. Las variaciones lentas corresponden a **bajas frecuencias**, mientras que los bordes, los detalles finos y los ruidos corresponden a **altas frecuencias**.

Esta representación se basa en el hecho de que cualquier imagen digital discreta puede descomponerse en una combinación de **funciones ortogonales**. La **Transformada de Fourier** utiliza una **base de exponenciales complejas bidimensionales** (equivalentes a senoides con orientación y frecuencia específicas). Otras transformadas, como la **Transformada de Cosenos (DCT)** y la **Transformada *Wavelet* (DWT)**, utilizan diferentes familias de funciones de base — cosenos bidimensionales en el caso de la DCT, y funciones con soporte compacto en el caso de las *wavelets*.

Entre las principales aplicaciones de esta representación se destacan:

1. **Filtrado en el dominio de la frecuencia**, para atenuar o realzar determinadas bandas de frecuencia;
2. **Análisis multirresolución mediante transformadas *wavelet***, que representa estructuras en diferentes escalas;
3. **Compresión de imágenes**, mediante la reducción del número de coeficientes necesarios para representar la imagen.

5.1 Objetivos

Al concluir este capítulo, usted será capaz de:

- **Interpretar el espectro de Fourier** de una imagen, distinguiendo magnitud, fase y componentes de frecuencia;
- **Aplicar el Teorema de la Convolución** para realizar filtrado en el dominio de la frecuencia utilizando la Transformada Rápida de Fourier (FFT);
- **Diseñar y analizar filtros en el dominio de la frecuencia**, comprendiendo el funcionamiento de filtros pasa-baja, pasa-alta y *notch*;
- **Comprender el análisis multirresolución mediante transformadas *wavelet*** y su aplicación en la representación jerárquica de imágenes;
- **Describir el proceso de compresión de imágenes**, incluyendo la Transformada Discreta del Coseno (DCT) y la cuantización de los coeficientes;
- **Seleccionar formatos de almacenamiento de imágenes**, como JPEG, PNG y WebP, de acuerdo con los requisitos de la aplicación.

5.2 Configuración del Entorno

```
import os, urllib.request

os.makedirs("tmp/state", exist_ok=True)  # artefactos de build de la ruta 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 incluso en la ruta C++. cpp=True descarga morph.hpp + stb; las
# celdas %%writefile *.cpp de este capítulo compilan CON OpenCV
# (-DMM_USE_OPENCV + pkg-config opencv4).
import config
config.setup(cpp=True)
from morph import mm
import numpy as np
```

```
Entorno listo. Morph: 1.1.9 | OpenCV: 5.0.0
```

5.3 Transformada de Fourier Discreta 2D

El análisis de Fourier se basa en el principio de que cualquier señal periódica puede representarse como una suma de funciones senoidales con diferentes frecuencias, amplitudes y fases. Este concepto también se aplica a las imágenes digitales, permitiendo representarlas en el **dominio de la frecuencia** en lugar del dominio espacial.

La Figura 5.1 ilustra esta descomposición para una señal unidimensional. En el caso de una imagen, la Transformada Discreta de Fourier (DFT) convierte la matriz de intensidades $f(x, y)$ en un conjunto de coeficientes que describe la contribución de las diferentes frecuencias espaciales presentes en la imagen.

```
%%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 <opencv2/opencv.hpp>
#include "morph.hpp"
#include <vector>
#include <string>
#include <cmath>
#include <filesystem>

int main() {
    // Crear vector x: 400 puntos de 0 a 2*pi
    std::vector<double> x(400);
    for (int i = 0; i < 400; ++i) {
        x[i] = 2.0 * M_PI * i / 399.0;
    }

    // Onda cuadrada: 1 para x < pi, -1 en caso contrario
    std::vector<double> square(400);
    for (int i = 0; i < 400; ++i) {
        square[i] = (x[i] < M_PI) ? 1.0 : -1.0;
```

```
    }

    // Suma acumulada y armónicos
    std::vector<double> soma(400, 0.0);
    std::vector<std::vector<double>> harms;
    for (int n = 1; 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];
        }
        harms.push_back(h);
    }

    // Curvas y etiquetas
    std::vector<std::vector<double>> ys = {square, harms[0], harms[1], harms[2], 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)
    };

    // Generar gráfico
    mm::Image chart = mm::lineChart(
        x, ys, labels, colors,
        "Sintese de Fourier: de senos a uma onda quadrada",
        "Posicao", "Intensidade"
    );

    // Mostrar resultado
    std::vector<mm::Image> images = {chart};
    std::vector<std::string> titles = {"Decomposicao de Fourier 1D"};
    mm::show(images, MM_OUT, titles, 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)")
```

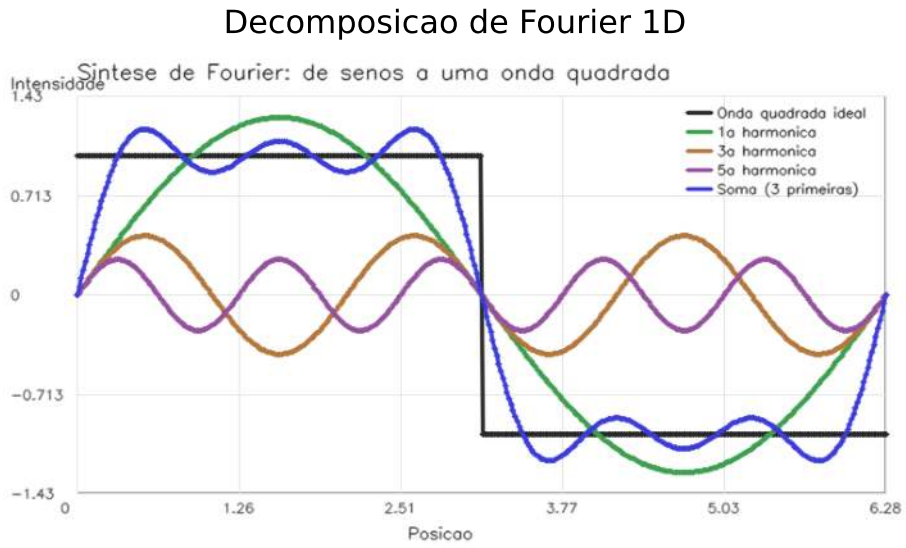


Figura 5.1: Decomposição de Fourier 1D: uma onda quadrada (linha tracejada) é aproximada pela soma das primeiras senoides (linhas coloridas). Quanto mais termos, melhor a aproximação.

5.3.1 Simulador: Reconstruyendo Señales con Senoides

Antes de estudiar imágenes bidimensionales, el simulador de la Figura 5.2 ilustra el principio del análisis de Fourier para señales unidimensionales: **una forma de onda puede aproximarse mediante la suma de senoides con diferentes frecuencias y amplitudes**.

A medida que se añaden nuevos términos, la suma de las senoides (curva negra) se aproxima a la forma de onda de referencia (trazada). El gráfico inferior presenta el espectro de amplitudes, indicando la contribución de cada frecuencia a la reconstrucción de la señal.

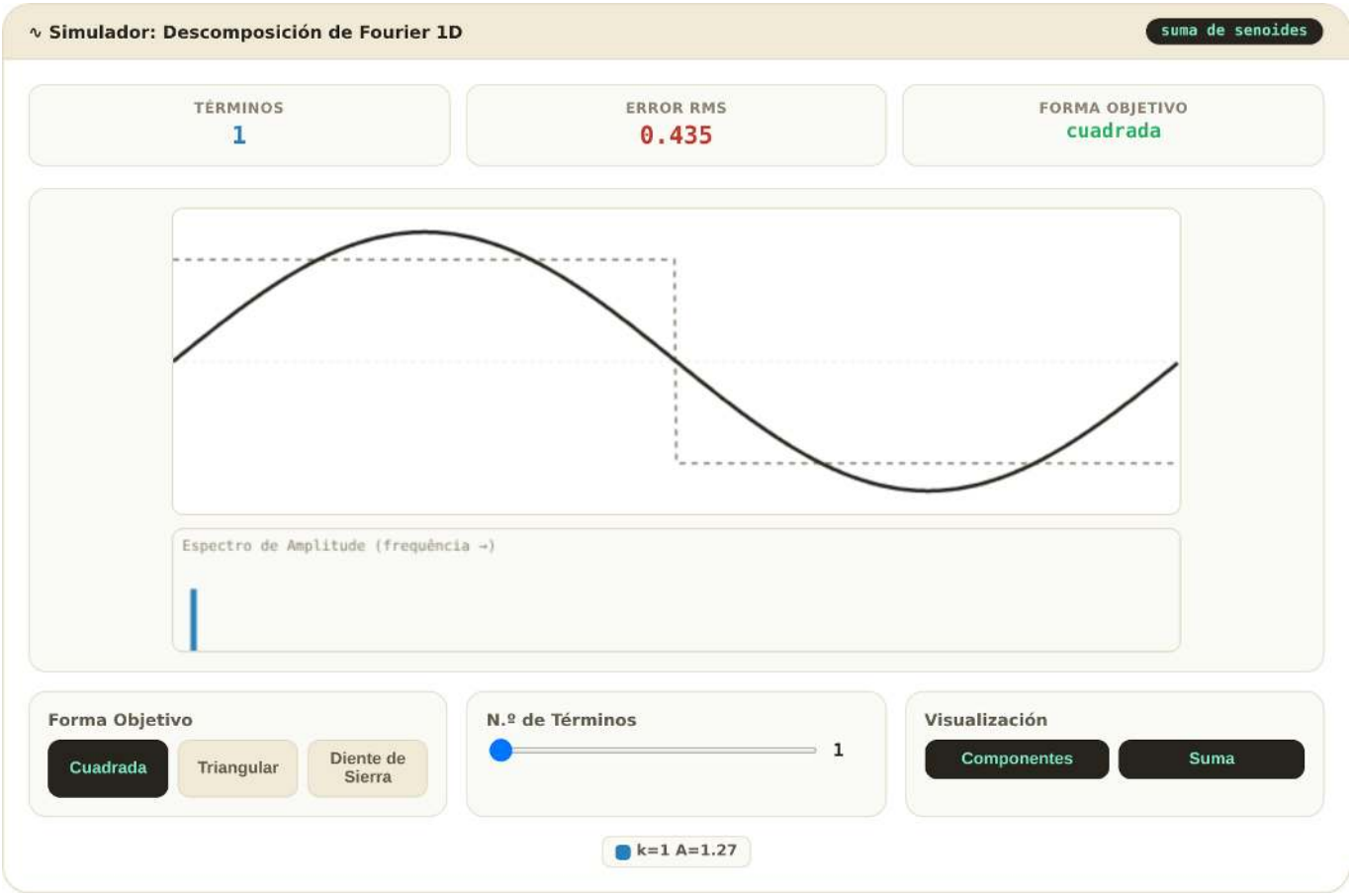
Actividad

Explore el simulador y responda:

1. ¿Cuántos términos son necesarios para obtener una buena aproximación de la onda cuadrada?
2. ¿Cuál de las tres formas de onda converge más rápidamente? Justifique su respuesta.
3. ¿Cómo se altera el espectro de amplitudes al cambiar la onda cuadrada por la triangular?

Respuestas

1. ¿Cuántos términos son necesarios para una buena aproximación de la onda cuadrada?
Con aproximadamente 15 a 20 términos, la forma de la onda ya se aproxima bien a la referencia. Sin embargo, cerca de las discontinuidades permanece una pequeña oscilación, conocida como **fenómeno de Gibbs**, que no desaparece incluso con la adición de más términos.
2. ¿Cuál forma converge más rápidamente? ¿Por qué?
La **onda triangular** converge más rápidamente, pues las amplitudes de sus armónicos decaen más rápido que las de la onda cuadrada y la onda de diente de sierra. Como consecuencia, pocos términos ya producen una buena aproximación.
3. ¿Cómo cambia el espectro entre la onda cuadrada y la triangular?
Ambas poseen únicamente **armónicos impares**, pero, en la onda triangular, las amplitudes disminuyen mucho más rápidamente. Así, pocos armónicos son suficientes para reconstruir la señal con buena precisión.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.es/cap05/sim-05-freq.html

Figura 5.2: Simulador interactivo de la descomposición de Fourier 1D: visualización de la suma de senoides con diferentes frecuencias, amplitudes y fases. Añada términos y observe la convergencia hacia formas de onda arbitrarias.

5.3.2 Interpretación del espectro de frecuencia

Al aplicar la Transformada Discreta de Fourier (DFT) a una imagen y visualizar el módulo de sus coeficientes (ver Figura 5.5), se obtiene el **espectro de magnitud**, que muestra la distribución de las frecuencias espaciales presentes en la imagen.

El coeficiente ubicado en el origen de la DFT, denominado **componente DC** (*Direct Current*), corresponde a la frecuencia nula y representa la intensidad media de la imagen. Por convención, dicho coeficiente se

almacena en la esquina superior izquierda del espectro. Para facilitar su interpretación, se aplica la operación **FFT Shift**, que desplaza la componente DC al centro de la imagen. Tras este desplazamiento, las bajas frecuencias se concentran en la región central, mientras que las altas frecuencias se sitúan cerca de los bordes, tal como resume la Tabla 5.1.

Tabla 5.1: Correspondencia entre las regiones del espectro de magnitud tras la aplicación del FFT Shift.

Región del espectro	Componentes predominantes	Ejemplos en la imagen
Centro (bajas frecuencias)	Variaciones espaciales lentas	Iluminación, regiones homogéneas y formas globales
Región intermedia (frecuencias medias)	Variaciones de escala intermedia	Texturas y patrones repetitivos
Bordes (altas frecuencias)	Variaciones espaciales rápidas	Contornos, detalles finos y ruido

Esta organización facilita la interpretación del espectro y el diseño de filtros. La atenuación de las bajas frecuencias reduce las variaciones globales de intensidad, mientras que la atenuación de las altas frecuencias suaviza la imagen al disminuir los detalles finos y parte del ruido.

5.3.3 El Experimento de la Rejilla: Construyendo una Imagen a partir de un Único Coeficiente

Antes de presentar la formulación matemática de la Transformada Discreta de Fourier (DFT), es útil analizar su inversa, denominada Transformada Discreta Inversa de Fourier (IDFT). Considere un espectro en el que todos los coeficientes sean nulos, excepto uno. Un ejemplo de esta construcción se presenta en el código de la Figura 5.3 y puede explorarse interactivamente en el simulador de la Figura 5.4..

La imagen reconstruida es una senoide bidimensional. La posición del coeficiente en el espectro determina su **orientación** y su **frecuencia espacial**, mientras que su magnitud y su fase definen, respectivamente, su amplitud y su desplazamiento espacial. Así, cada coeficiente de la DFT representa una componente senoidal, y la imagen original puede reconstruirse mediante la suma de todas estas componentes.

```
%%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 "morph.hpp"
#include <vector>
#include <string>
#include <filesystem>

int main() {
    int N_grid = 100;
    cv::Mat espectro_vazio = cv::Mat::zeros(N_grid, N_grid, CV_64FC2);

    // 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[2];
    cv::split(espectro_vazio, planes);
```

```
    // Manual fftshift
    int cx = N_grid / 2;
    int cy = N_grid / 2;
    cv::Mat q0(espectro_vazio, cv::Rect(0, 0, cx, cy));
    cv::Mat q1(espectro_vazio, cv::Rect(cx, 0, cx, cy));
    cv::Mat q2(espectro_vazio, cv::Rect(0, cy, cx, cy));
    cv::Mat q3(espectro_vazio, cv::Rect(cx, cy, cx, cy));
    cv::Mat tmp;
    q0.copyTo(tmp);
    q3.copyTo(q0);
    tmp.copyTo(q3);
    q1.copyTo(tmp);
    q2.copyTo(q1);
    tmp.copyTo(q2);

    cv::Mat espectro_ifft;
    cv::idft(espectro_vazio, espectro_ifft, cv::DFT_SCALE | cv::DFT_REAL_OUTPUT);

    // Manual ifftshift (swap again)
    cv::Mat q0b(espectro_ifft, cv::Rect(0, 0, cx, cy));
    cv::Mat q1b(espectro_ifft, cv::Rect(cx, 0, cx, cy));
    cv::Mat q2b(espectro_ifft, cv::Rect(0, cy, cx, cy));
    cv::Mat q3b(espectro_ifft, cv::Rect(cx, cy, cx, cy));
    q0b.copyTo(tmp);
    q3b.copyTo(q0b);
    tmp.copyTo(q3b);
    q1b.copyTo(tmp);
    q2b.copyTo(q1b);
    tmp.copyTo(q2b);

    cv::Mat onda_2d = espectro_ifft;

    cv::Mat onda_vis, espectro_vis;
    cv::normalize(onda_2d, onda_vis, 0, 255, cv::NORM_MINMAX, CV_8U);
    cv::Mat espectro_mag;
    cv::split(espectro_vazio, planes);
    cv::magnitude(planes[0], planes[1], espectro_mag);
    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::show(std::vector<mm::Image>{mm::Image(espectro_color), mm::Image(onda_vis)},
             MM_OUT,
             {"Espectro (1 ponto ativo)", "Onda 2D Resultante (IDFT)"},
             2);


// [pdi:panel-io] auto-generated - do not edit by hand
std::filesystem::create_directories("tmp");
mm::write(espectro_color, "tmp/fig_05_grade_2d_0.png");
mm::write(onda_vis, "tmp/fig_05_grade_2d_1.png");
// [pdi:panel-io:end]
return 0;
}
```

```
Overwriting tmp/fig_05_grade_2d.cpp
```

```
!g++ -I. -std=c++17 -DMM_USE_OPENCV -I/usr/include/opencv4 tmp/fig_05_grade_2d.cpp -o
tmp/fig_05_grade_2d -lopencv_stitching -lopencv_alphamat -lopencv_aruco -lopencv_barcode
-lopencv_bgsegm -lopencv_bioinspired -lopencv_ccalib -lopencv_dnn_objdetect
-lopencv_dnn_superres -lopencv_dpm -lopencv_face -lopencv_freetype -lopencv_fuzzy -lopencv_hdf
-lopencv_hfs -lopencv_img_hash -lopencv_intensity_transform -lopencv_line_descriptor
-lopencv_mcc -lopencv_quality -lopencv_rapid -lopencv_reg -lopencv_rgbd -lopencv_saliency
-lopencv_shape -lopencv_stereo -lopencv_structured_light -lopencv_phase_unwrapping
-lopencv_superres -lopencv_optflow -lopencv_surface_matching -lopencv_tracking
-lopencv_highgui -lopencv_datasets -lopencv_text -lopencv_plot -lopencv_ml -lopencv_videostab
-lopencv_videoio -lopencv_viz -lopencv_wechat_qrcode -lopencv_ximgproc -lopencv_video
-lopencv_xobjdetect -lopencv_objdetect -lopencv_calib3d -lopencv_imgcodecs -lopencv_features2d
-lopencv_dnn -lopencv_flann -lopencv_xphoto -lopencv_photo -lopencv_imgproc -lopencv_core \
  && ./tmp/fig_05_grade_2d \
  && test -f "tmp/fig_05_grade_2d.png" \
  || echo "  mm::show não gravou tmp/fig_05_grade_2d.png"
```

```
[1] 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)")
```

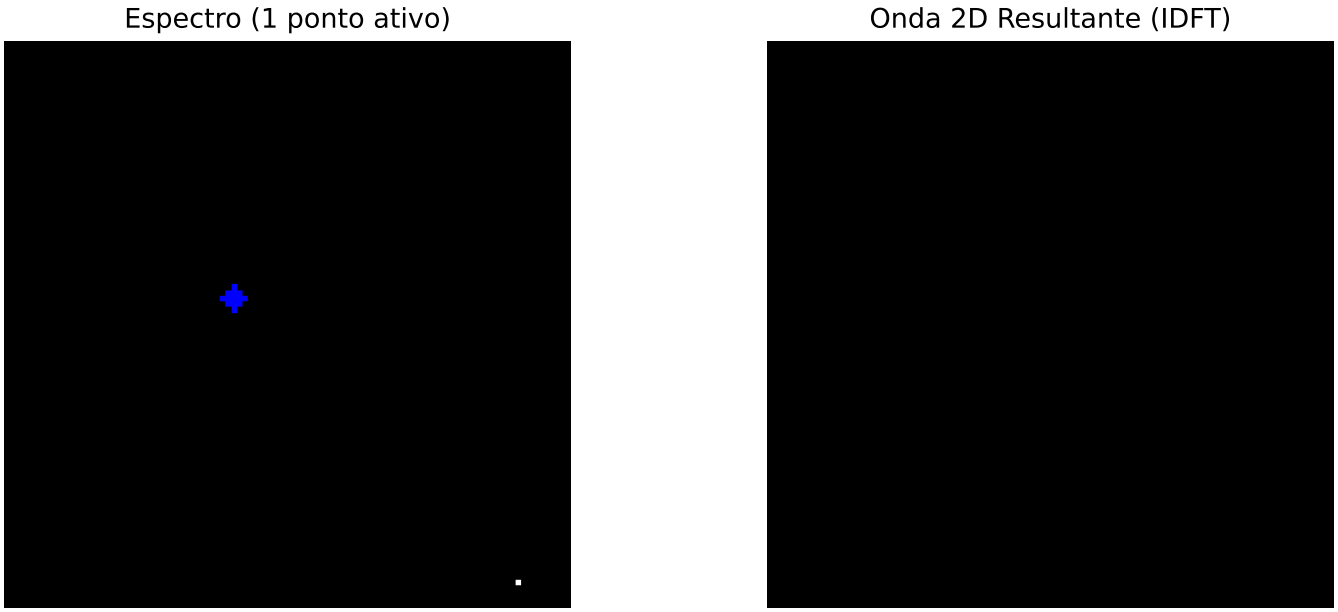
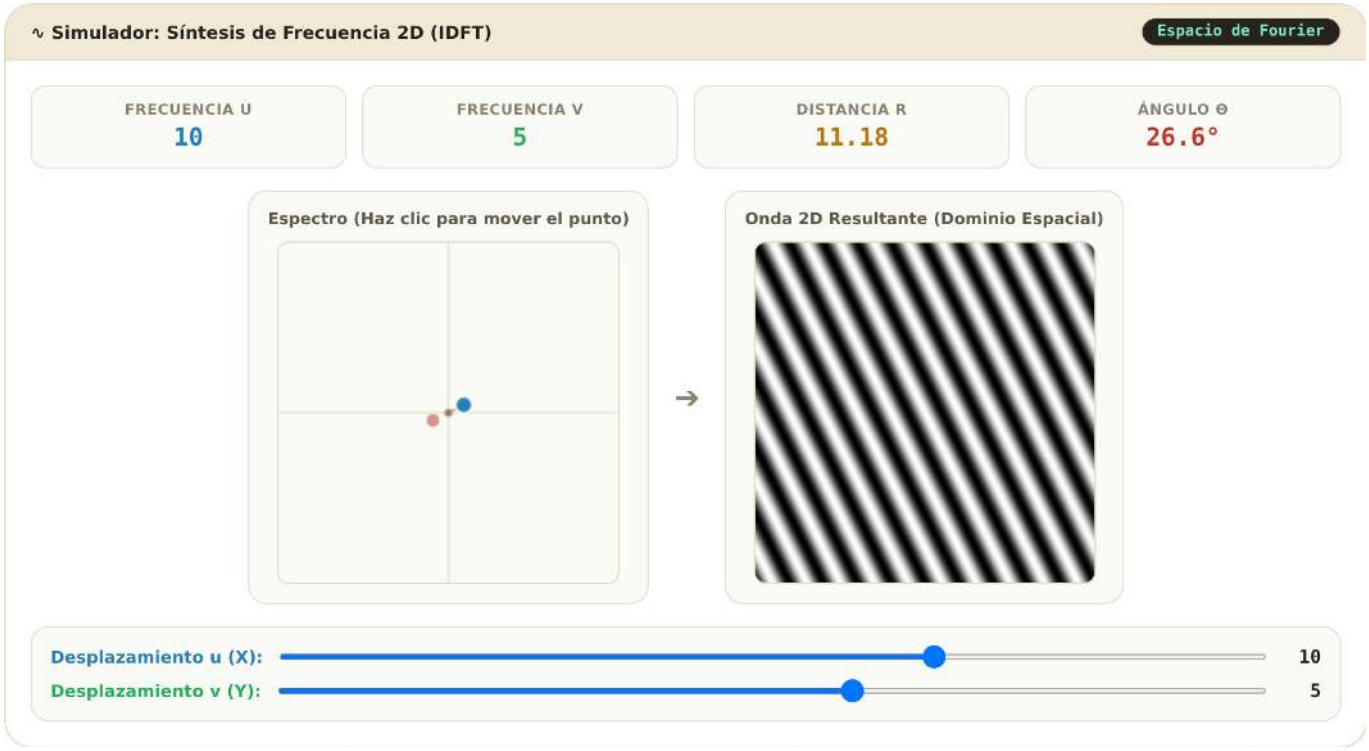


Figura 5.3: Toda frequência no espectro (ponto isolado) corresponde a uma onda senoidal 2D rotacionada no domínio espacial.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.es/cap05/sim-05-grade-2d.html

Figura 5.4: Simulador interactivo de la síntesis de Fourier 2D. Cambie la posición horizontal (u) y vertical (v) del coeficiente en el espectro de frecuencias centrado y observe cómo la distancia respecto al centro determina la frecuencia espacial (grosor) y el ángulo determina la orientación de la onda sinusoidal generada.

5.3.4 Definición Matemática

Considere una imagen $f(x, y)$ con dimensiones $M \times N$. Su **Transformada Discreta de Fourier 2D** (DFT) se define por:

$$F(u, v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x, y)\, e^{-j2\pi\left(\frac{ux}{M} + \frac{vy}{N}\right)} \tag{5.1}$$

donde $u = 0, 1, \ldots, M-1$ y $v = 0, 1, \ldots, N-1$ representan las frecuencias discretas en las direcciones horizontal y vertical, respectivamente. El término exponencial corresponde a una senoide bidimensional, cuya frecuencia y orientación están determinadas por los índices (u, v).

La **Transformada Discreta Inversa de Fourier 2D** (IDFT) reconstruye la imagen original a partir de sus coeficientes:

$$f(x, y) = \frac{1}{MN} \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} F(u, v)\, e^{j2\pi\left(\frac{ux}{M} + \frac{vy}{N}\right)} \tag{5.2}$$

Las Ecuaciones Ecuación 5.1 y Ecuación 5.2 muestran que la DFT y la IDFT forman un par de transformaciones: la primera convierte la imagen al dominio de la frecuencia, mientras que la segunda reconstruye exactamente la imagen original a partir de sus coeficientes.

> **i** Sobre el símbolo j
>
> El término j denota la **unidad imaginaria**, definida por $j^2 = -1$. En ingeniería y procesamiento de señales, se adopta j en lugar de i para evitar conflictos con la notación de corriente eléctrica. Su uso en la exponencial compleja, regida por la fórmula de Euler ($e^{j\theta} = \cos\theta + j\sin\theta$), permite representar de

forma compacta la amplitud y la fase de cada frecuencia espacial presente en la imagen.

i ¿Qué es el componente DC?

El coeficiente $F(0,0)$, denominado **componente DC** (*Direct Current*), es igual a la suma de las intensidades de todos los píxeles de la imagen (ver Figura 5.5):

$$F(0,0) = MN\,\bar{f},$$

donde $\bar{f}$ es la intensidad media de la imagen. Por ello, el componente DC representa el nivel medio de intensidad y, en la mayoría de las imágenes naturales, posee la mayor magnitud del espectro.
Los demás coeficientes representan variaciones en torno a esa media. Tras la aplicación del **FFT Shift**, el componente DC se desplaza al centro del espectro, concentrando las bajas frecuencias en la región central y las altas frecuencias en los bordes.

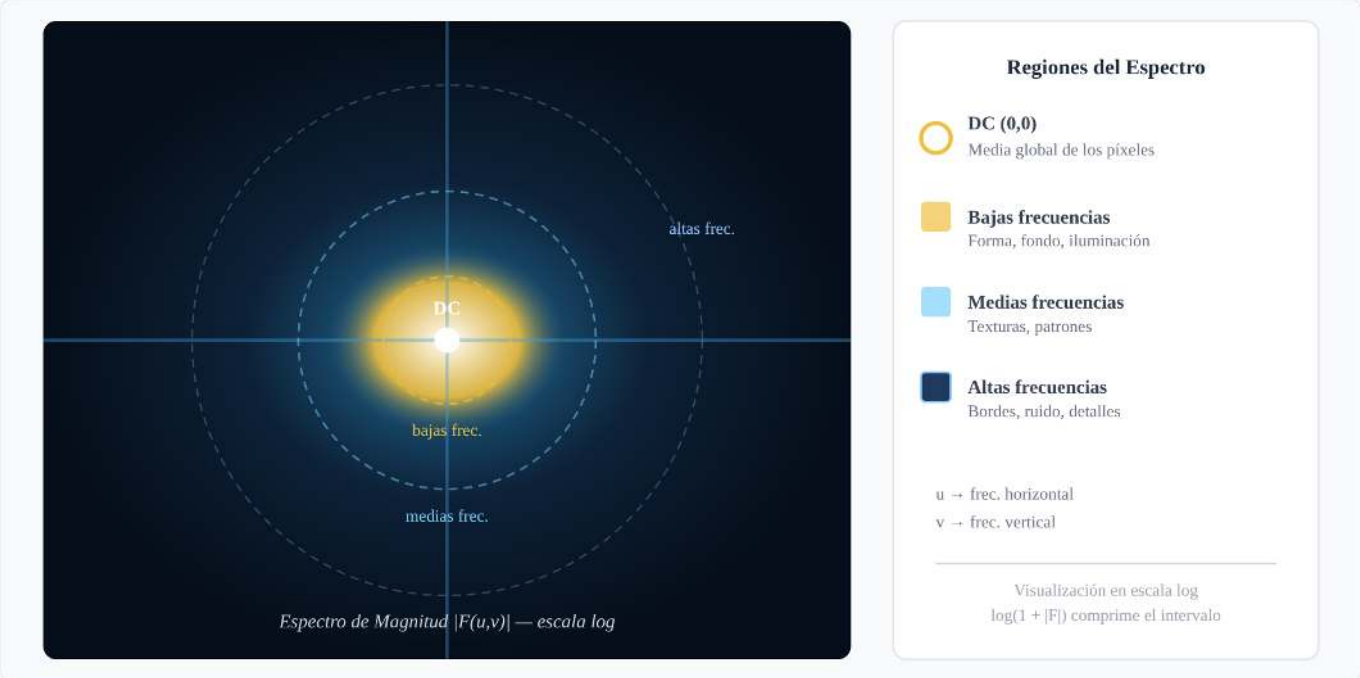


Figura 5.5: Diagrama conceptual del espectro de Fourier 2D centrado.

5.3.5 Magnitud y Fase

Cada coeficiente de la Transformada Discreta de Fourier (DFT) es un número complejo y puede escribirse como

$$F(u,v) = R(u,v) + j\,I(u,v),$$

donde $R(u,v)$ e $I(u,v)$ corresponden, respectivamente, a las partes **real** e **imaginaria** del coeficiente. De la Ecuación Ecuación 5.1, se obtienen

$$R(u,v) = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)\cos\left(2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)\right),$$

y

$$I(u,v) = -\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)\sin\Big(2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)\Big)\,.$$

A partir de esta representación, se definen dos magnitudes fundamentales:

- **Magnitud**, que indica la intensidad de la componente de frecuencia,

$$|F(u,v)| = \sqrt{R(u,v)^2 + I(u,v)^2};$$

- **Fase**, que determina la alineación (o desplazamiento) espacial de la componente,

$$\phi(u,v) = \operatorname{atan2}(I(u,v),\, R(u,v))\,.$$

Así, cada coeficiente también puede escribirse en su forma polar,

$$F(u,v) = |F(u,v)|\, e^{j\phi(u,v)}.$$

El espectro de Fourier puede, por tanto, visualizarse mediante dos imágenes distintas: el **espectro de magnitud**, normalmente utilizado para analizar la distribución de las frecuencias, y el **espectro de fase**, que describe la organización espacial de las componentes senoidales.

Aunque el espectro de magnitud sea el más utilizado para la inspección visual, la fase contiene gran parte de la información estructural de la imagen. La combinación de magnitud y fase permite reconstruir exactamente la imagen original mediante la IDFT.

5.3.6 ¿Qué transportan la magnitud y la fase?

Una demostración clásica consiste en combinar la magnitud de una imagen con la fase de otra y reconstruir el resultado. Este experimento evidencia que:

- **La fase** preserva la estructura espacial de la imagen, incluida la posición de los objetos, sus contornos y su geometría. Pequeñas alteraciones en la fase pueden provocar grandes cambios visuales.
- **La magnitud** controla cómo se distribuye la energía entre las frecuencias espaciales, influyendo principalmente en el contraste y la textura.

Cuando una imagen se reconstruye con la magnitud de A y la fase de B, el resultado tiende a **parecerse más a B que a A**, evidenciando que la fase es el principal componente responsable de la organización espacial de la escena. Sin embargo, la magnitud sigue siendo importante, ya que modula el contraste de las estructuras reconstruidas. Así, una reconstrucción fiel depende de la combinación coherente entre magnitud y fase.

Un ejemplo de este comportamiento se presenta en la Figura 5.6.

> **i** Analogía con el Audio: Limitaciones y Precauciones
>
> La fase de una señal desempeña papeles distintos en audio e imágenes:
> - **Audio estéreo o multicanal:** la fase relativa entre los canales es fundamental para la percepción de la posición de las fuentes sonoras, mediante las diferencias interaurales de tiempo (ITD, *Interaural Time Differences*).
> - **Audio monoaural:** la fase absoluta ejerce poca influencia perceptual directa.
> - **Imágenes (DFT):** la fase es el principal factor responsable de la organización espacial de la escena, mientras que la magnitud modula el contraste y la distribución de la energía entre las frecuencias.
>
> En ambos dominios, la **magnitud** está relacionada con la intensidad de los componentes de frecuencia: en audio, influye en el timbre y la intensidad percibida; en imágenes, influye en el contraste y la textura.

```
%%writefile tmp/fig_05_dft_intro.cpp
#define MM_OUT "tmp/fig_05_dft_intro.png"
// Compile with: g++ -std=c++17 -O2 program.cpp -o program $(pkg-config --cflags --libs
opencv4)
#include <opencv2/opencv.hpp>
#include <opencv2/core.hpp>
#include <opencv2/imgproc.hpp>
#include <opencv2/highgui.hpp>
#include <iostream>
#include <string>
#include <cmath>
#include <vector>
#include <filesystem>
#include "morph.hpp"

// Helper: swap quadrants for fftshift
void fftshift(cv::Mat& mat) {
    int cx = mat.cols / 2;
    int cy = mat.rows / 2;
    cv::Mat q0(mat, cv::Rect(0, 0, cx, cy));
    cv::Mat q1(mat, cv::Rect(cx, 0, cx, cy));
    cv::Mat q2(mat, cv::Rect(0, cy, cx, cy));
    cv::Mat q3(mat, cv::Rect(cx, cy, cx, cy));
    cv::Mat tmp;
    q0.copyTo(tmp);
    q3.copyTo(q0);
    tmp.copyTo(q3);
    q1.copyTo(tmp);
    q2.copyTo(q1);
    tmp.copyTo(q2);
}

// Helper: compute magnitude spectrum in dB
cv::Mat spectrumMag(cv::Mat& img) {
    cv::Mat img32f;
    img.convertTo(img32f, CV_32F);
    cv::Mat planes[] = {img32f, cv::Mat::zeros(img32f.size(), CV_32F)};
    cv::Mat complexImg;
    cv::merge(planes, 2, complexImg);
    cv::dft(complexImg, complexImg);
    cv::Mat mag, phase;
    cv::magnitude(complexImg, complexImg, mag);

    // Shift quadrants
    fftshift(mag);

    // Compute log(1+|F|) and normalize
    mag += 1.0;
    cv::log(mag, mag);
    cv::normalize(mag, mag, 0, 255, cv::NORM_MINMAX, CV_8U);
    return mag;
}

int main() {
    //   Experimento: A Importância da Fase
    //   Carregamento da imagem
    std::string url =
"https://upload.wikimedia.org/wikipedia/commons/2/25/GAZI.MD.AHAD_11.jpg";
    std::string caminho = "imagens/coins.jpg";

    if (!std::filesystem::exists(caminho)) {
```

```
        std::filesystem::create_directories("imagens");
        cv::Mat img_obj = cv::imread(url, cv::IMREAD_COLOR);
        mm::write(img_obj, caminho);
    }

    cv::Mat img_color = cv::imread(caminho, cv::IMREAD_COLOR);
    cv::Mat img_gray_mat;
    cv::cvtColor(img_color, img_gray_mat, cv::COLOR_BGR2GRAY);
    mm::Image img_gray(img_gray_mat);

    cv::Mat img_a;
    cv::resize(img_gray_mat, img_a, cv::Size(400, 400));

    // Criar uma imagem B sintética (padrão geométrico)
    cv::Mat img_b = cv::Mat::zeros(400, 400, CV_8U);
    cv::rectangle(img_b, cv::Point(100, 100), cv::Point(300, 300), cv::Scalar(255), -1);
    cv::circle(img_b, cv::Point(200, 200), 150, cv::Scalar(128), 10);

    // FFT de A e B
    cv::Mat img_a_f;
    img_a.convertTo(img_a_f, CV_32F);
    cv::Mat planesA[] = {img_a_f, cv::Mat::zeros(img_a.size(), CV_32F)};
    cv::Mat complexA;
    cv::merge(planesA, 2, complexA);
    cv::dft(complexA, complexA);

    cv::Mat img_b_f;
    img_b.convertTo(img_b_f, CV_32F);
    cv::Mat planesB[] = {img_b_f, cv::Mat::zeros(img_b.size(), CV_32F)};
    cv::Mat complexB;
    cv::merge(planesB, 2, complexB);
    cv::dft(complexB, complexB);

    // Separar magnitude e fase
    cv::Mat magA[2], magB[2], phaseA[2], phaseB[2];
    cv::Mat planesA_out[2], planesB_out[2];
    cv::split(complexA, planesA_out);
    cv::split(complexB, planesB_out);
    cv::magnitude(planesA_out[0], planesA_out[1], magA[0]);
    cv::magnitude(planesB_out[0], planesB_out[1], magB[0]);
    cv::phase(planesA_out[0], planesA_out[1], phaseA[0]);
    cv::phase(planesB_out[0], planesB_out[1], phaseB[0]);

    // Troca de fase: reconstruir com magnitude de A e fase de B
    cv::Mat complex_recAB[2];
    cv::Mat realAB, imagAB;
    cv::polarToCart(magA[0], phaseB[0], realAB, imagAB);
    complex_recAB[0] = realAB;
    complex_recAB[1] = imagAB;
    cv::Mat rec_complexAB;
    cv::merge(complex_recAB, 2, rec_complexAB);
    cv::Mat rec_A_mag_B_fase_mat;
    cv::idft(rec_complexAB, rec_A_mag_B_fase_mat, cv::DFT_SCALE | cv::DFT_REAL_OUTPUT);

    // Troca de fase: reconstruir com magnitude de B e fase de A
    cv::Mat complex_recBA[2];
    cv::Mat realBA, imagBA;
    cv::polarToCart(magB[0], phaseA[0], realBA, imagBA);
    complex_recBA[0] = realBA;
    complex_recBA[1] = imagBA;
    cv::Mat rec_complexBA;
```

```
    cv::merge(complex_recBA, 2, rec_complexBA);
    cv::Mat rec_B_mag_A_fase_mat;
    cv::idft(rec_complexBA, rec_B_mag_A_fase_mat, cv::DFT_SCALE | cv::DFT_REAL_OUTPUT);

    // Converter para mm::Image
    cv::Mat rec_A_mag_B_fase_8u, rec_B_mag_A_fase_8u;
    rec_A_mag_B_fase_mat.convertTo(rec_A_mag_B_fase_8u, CV_8U);
    rec_B_mag_A_fase_mat.convertTo(rec_B_mag_A_fase_8u, CV_8U);

    mm::Image img_a_im(img_a);
    mm::Image img_b_im(img_b);
    mm::Image rec_A_mag_B_fase(rec_A_mag_B_fase_8u);
    mm::Image rec_B_mag_A_fase(rec_B_mag_A_fase_8u);

    mm::show({img_a_im, img_b_im, rec_A_mag_B_fase, rec_B_mag_A_fase},
             MM_OUT, {"Imagem A", "Imagem B", "Mag(A) + Fase(B)", "Mag(B) + Fase(A)"}, 4);

    std::cout << "  La fase preserva bordes y contornos; la magnitud controla contraste y" <<
std::endl;
    std::cout << "textura. En audio estéreo, la fase afecta la localización espacial; en" <<
std::endl;
    std::cout << "imágenes, determina la organización de la escena." << std::endl;

    // Variables to persist
    img_gray = img_gray;  // already set


// [pdi:state-io] auto-generated - do not edit by hand
std::filesystem::create_directories("tmp/state");
mm::write(img_gray, "tmp/state/img_gray_20.png");
// [pdi:state-io:end]

// [pdi:panel-io] auto-generated - do not edit by hand
std::filesystem::create_directories("tmp");
mm::write(img_a, "tmp/fig_05_dft_intro_0.png");
mm::write(img_b, "tmp/fig_05_dft_intro_1.png");
mm::write(rec_A_mag_B_fase, "tmp/fig_05_dft_intro_2.png");
mm::write(rec_B_mag_A_fase, "tmp/fig_05_dft_intro_3.png");
// [pdi:panel-io:end]
return 0;
}
```

```
Overwriting tmp/fig_05_dft_intro.cpp
```

```
!g++ -I. -std=c++17 -DMM_USE_OPENCV -I/usr/include/opencv4 tmp/fig_05_dft_intro.cpp -o
tmp/fig_05_dft_intro -lopencv_stitching -lopencv_alphamat -lopencv_aruco -lopencv_barcode
-lopencv_bgsegm -lopencv_bioinspired -lopencv_ccalib -lopencv_dnn_objdetect
-lopencv_dnn_superres -lopencv_dpm -lopencv_face -lopencv_freetype -lopencv_fuzzy -lopencv_hdf
-lopencv_hfs -lopencv_img_hash -lopencv_intensity_transform -lopencv_line_descriptor
-lopencv_mcc -lopencv_quality -lopencv_rapid -lopencv_reg -lopencv_rgbd -lopencv_saliency
-lopencv_shape -lopencv_stereo -lopencv_structured_light -lopencv_phase_unwrapping
-lopencv_superres -lopencv_optflow -lopencv_surface_matching -lopencv_tracking
-lopencv_highgui -lopencv_datasets -lopencv_text -lopencv_plot -lopencv_ml -lopencv_videostab
-lopencv_videoio -lopencv_viz -lopencv_wechat_qrcode -lopencv_ximgproc -lopencv_video
-lopencv_xobjdetect -lopencv_objdetect -lopencv_calib3d -lopencv_imgcodecs -lopencv_features2d
-lopencv_dnn -lopencv_flann -lopencv_xphoto -lopencv_photo -lopencv_imgproc -lopencv_core \
  && ./tmp/fig_05_dft_intro \
  && test -f "tmp/fig_05_dft_intro.png" \
  || echo "  mm::show não gravou tmp/fig_05_dft_intro.png"
```

```
[1] Imagem A
[2] Imagem B
[3] Mag(A) + Fase(B)
[4] Mag(B) + Fase(A)
 La fase preserva bordes y contornos; la magnitud controla contraste y
textura. En audio estéreo, la fase afecta la localización espacial; en
imágenes, determina la organización de la escena.
```

```
try:
    mm.show(
        [
            mm.read("tmp/fig_05_dft_intro_0.png"),
            mm.read("tmp/fig_05_dft_intro_1.png"),
            mm.read("tmp/fig_05_dft_intro_2.png"),
            mm.read("tmp/fig_05_dft_intro_3.png"),
        ],
        titles=[
            'Imagem A',
            'Imagem B',
            'Mag(A) + Fase(B)',
            'Mag(B) + Fase(A)',
        ],
        cols=4,
        figsize=(16, 4),
    )
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + " tmp/fig_05_dft_intro_0.png
    (ver a versao Python)")
```

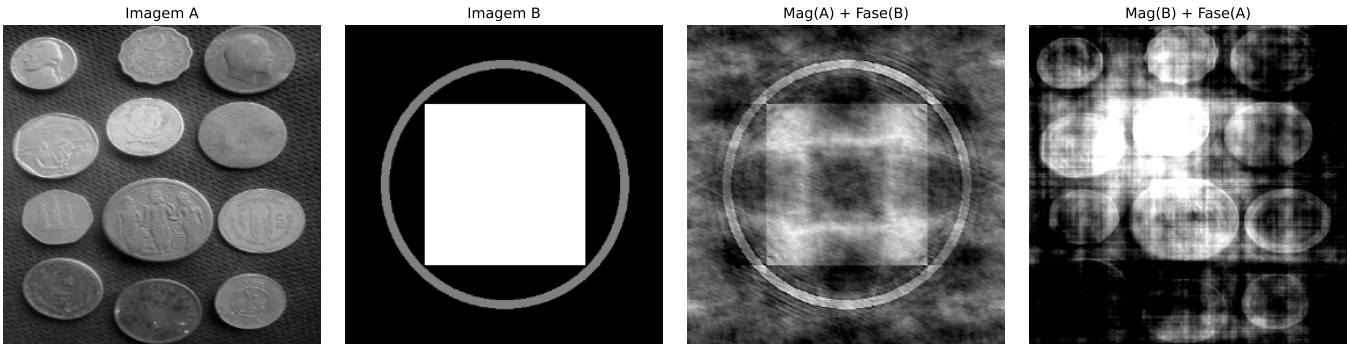


Figura 5.6: Experimento de troca de fase: Imagem A (moedas) e Imagem B (padrão geométrico) reconstruídas com magnitudes e fases trocadas. O resultado mostra que a estrutura visual é **muito mais sensível à fase** do que à magnitude: quando a fase de B é mantida, a imagem resultante preserva a organização espacial de B, mesmo com a magnitude de A. A magnitude, por sua vez, influencia principalmente o contraste e a textura. Observe que a qualidade da reconstrução não é perfeita — há artefatos visíveis —, evidenciando a interdependência entre fase e magnitude para uma representação fiel da imagem.

5.4 Teorema de la Convolución y Estrategias de Filtrado

El **Teorema de la Convolución** establece una relación fundamental entre los dominios espacial y de la frecuencia:

$$f(x,y) \circledast h(x,y) \overset{\mathcal{F}}{\longleftrightarrow} F(u,v)\,H(u,v) \tag{5.3}$$

donde $\circledast$ representa la **convolución circular discreta**. Así, la convolución entre una imagen $f(x,y)$ y un filtro $h(x,y)$ puede sustituirse por la multiplicación de sus espectros.

En la práctica, para obtener el mismo resultado de la convolución lineal realizada en el dominio espacial, se aplica **relleno con ceros** (*zero-padding*) antes de la Transformada Rápida de Fourier (FFT), evitando artefactos en los bordes de la imagen.

Sin embargo, no siempre el filtrado en el dominio de la frecuencia es la alternativa más eficiente. Para filtros como el gaussiano y el filtro de la media (*Box Filter*), la propiedad de **separabilidad** permite reducir significativamente el costo computacional de la convolución en el dominio espacial.

5.4.1 *Kernel* Separable vs. No separable

Un ***kernel* separable** puede escribirse como el producto externo de dos vectores unidimensionales,

$$H = v\, h^T,$$

lo que permite que la convolución bidimensional se reemplace por dos convoluciones unidimensionales consecutivas: una en la dirección horizontal y otra en la vertical.

En cambio, un ***kernel* no separable** no admite esta descomposición y, por lo tanto, su convolución debe realizarse directamente sobre la vecindad bidimensional.

En la práctica, para un *kernel* de dimensión $K \times K$, la convolución directa requiere K^2 multiplicaciones por píxel, mientras que un *kernel* separable requiere solo $2K$ multiplicaciones, reduciendo significativamente el costo computacional.

5.4.2 Análisis de Eficiencia Computacional

Considere una imagen de dimensiones $M \times N$ y un filtro cuadrado de tamaño $K \times K$. La Tabla 5.2 compara la complejidad de las principales estrategias de filtrado.

Tabla 5.2: Comparación de la complejidad de la convolución directa, separable y vía Transformada Rápida de Fourier (FFT).

Método de Filtrado	Complejidad Asintótica	Dependencia de K	Aplicación típica
Espacial no separable	$\mathcal{O}(MNK^2)$	Cuadrática	*Kernels* pequeños y no separables
Espacial separable	$\mathcal{O}(MNK)$	Lineal	Filtros Gaussiano y de la media
Vía FFT	$\mathcal{O}(MN\log(MN))$	Independiente de K	*Kernels* grandes

Para *kernels* pequeños, la convolución espacial, especialmente cuando el filtro es separable, suele ser más eficiente debido al bajo costo de las operaciones. A medida que el tamaño del *kernel* aumenta, el filtrado vía FFT se vuelve más ventajoso, ya que su costo prácticamente no depende de la dimensión del filtro.

5.4.3 Discusión de los resultados experimentales

El gráfico obtenido en el ensayo con la imagen de las monedas (2560×1920), presentado en la Figura 5.7, confirma el comportamiento previsto por el análisis de complejidad computacional.

1. **Convolución no separable ($\mathcal{O}(MNK^2)$)**
   La convolución directa presenta un crecimiento cuadrático con el tamaño del *kernel*. Para valores pequeños de K, el costo es bajo, pero aumenta rápidamente a medida que el *kernel* crece, volviéndose inviable para aplicaciones en tiempo real.
2. **Filtrado mediante FFT ($\mathcal{O}(MN\log(MN))$)**
   El costo de la FFT depende únicamente del tamaño de la imagen, siendo independiente de K. Por ello,

su rendimiento permanece aproximadamente constante al variar el *kernel*, lo que la hace ventajosa para filtros grandes o no separables.

3. **Convolución separable ($\mathcal{O}(MNK)$)**
   La descomposición del *kernel* en dos filtros unidimensionales reduce significativamente el costo computacional. En la práctica, este enfoque tiende a ser el más eficiente para filtros separables, especialmente en implementaciones optimizadas.

En general, la elección del método depende del tamaño y la estructura del *kernel*. Los filtros separables son más eficientes en el dominio espacial, mientras que la FFT resulta más ventajosa para *kernels* grandes o múltiples convoluciones en el dominio de la frecuencia.

$$g = \mathcal{F}^{-1}[\mathcal{F}(f) \cdot \mathcal{F}(h)] \quad \text{(FFT)} \qquad g = f \circledast h \quad \text{(convolución directa)} \qquad g = (f \circledast v) \circledast h^T \quad \text{(separable)} \tag{5.4}$$

donde:

- $f(x, y)$ representa la imagen de entrada;
- $h(x, y)$ es el *kernel* bidimensional del filtro;
- v y h^T son, respectivamente, los vectores vertical y horizontal que componen el *kernel* separable.

```
%%writefile tmp/fig_05_conv_eficiencia.cpp
#define MM_OUT "tmp/fig_05_conv_eficiencia.png"
//| label: fig-05-conv-eficiencia
//| fig-cap: "Comparação de eficiência: Convolução Não Separável (Espacial 2D), Separável
(Espacial 1D) e via FFT."
//| echo: false
//| output: true

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

int main() {
    // Trilha C++: curvas de CUSTO relativo (contagem de operacoes) - a forma das
    // curvas (K^2 vs 2K vs log N) e o que importa; a trilha py mede tempos reais.
    std::vector<double> K = {3, 7, 11, 15, 21, 31, 41, 51};
    double N2 = 256.0 * 256.0;
    std::vector<double> t_nao_sep, t_sep, 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, {},
        "Custo relativo: Espacial vs Frequencia",
        "Tamanho do kernel  K", "Operacoes  (x10^6)"
```

```
    );
    mm::show({chart}, MM_OUT, {"Comparacao de complexidade"}, 1);

// [pdi:panel-io] auto-generated - do not edit by hand
std::filesystem::create_directories("tmp");
mm::write(chart, "tmp/fig_05_conv_eficiencia_0.png");
// [pdi:panel-io:end]
return 0;
}
```

```
Overwriting tmp/fig_05_conv_eficiencia.cpp
```

```
!g++ -I. -std=c++17 -DMM_USE_OPENCV -I/usr/include/opencv4 tmp/fig_05_conv_eficiencia.cpp -o
tmp/fig_05_conv_eficiencia -lopencv_stitching -lopencv_alphamat -lopencv_aruco
-lopencv_barcode -lopencv_bgsegm -lopencv_bioinspired -lopencv_ccalib -lopencv_dnn_objdetect
-lopencv_dnn_superres -lopencv_dpm -lopencv_face -lopencv_freetype -lopencv_fuzzy -lopencv_hdf
-lopencv_hfs -lopencv_img_hash -lopencv_intensity_transform -lopencv_line_descriptor
-lopencv_mcc -lopencv_quality -lopencv_rapid -lopencv_reg -lopencv_rgbd -lopencv_saliency
-lopencv_shape -lopencv_stereo -lopencv_structured_light -lopencv_phase_unwrapping
-lopencv_superres -lopencv_optflow -lopencv_surface_matching -lopencv_tracking
-lopencv_highgui -lopencv_datasets -lopencv_text -lopencv_plot -lopencv_ml -lopencv_videostab
-lopencv_videoio -lopencv_viz -lopencv_wechat_qrcode -lopencv_ximgproc -lopencv_video
-lopencv_xobjdetect -lopencv_objdetect -lopencv_calib3d -lopencv_imgcodecs -lopencv_features2d
-lopencv_dnn -lopencv_flann -lopencv_xphoto -lopencv_photo -lopencv_imgproc -lopencv_core \
  && ./tmp/fig_05_conv_eficiencia \
  && test -f "tmp/fig_05_conv_eficiencia.png" \
  || echo "  mm::show não gravou tmp/fig_05_conv_eficiencia.png"
```

```
[1] Comparacao de complexidade
```

```
try:
    mm.show(
        [
            mm.read("tmp/fig_05_conv_eficiencia_0.png"),
        ],
        titles=[
            'Comparacao de complexidade',
        ],
        cols=1,
    )
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + "
    tmp/fig_05_conv_eficiencia_0.png (ver a versao Python)")
```

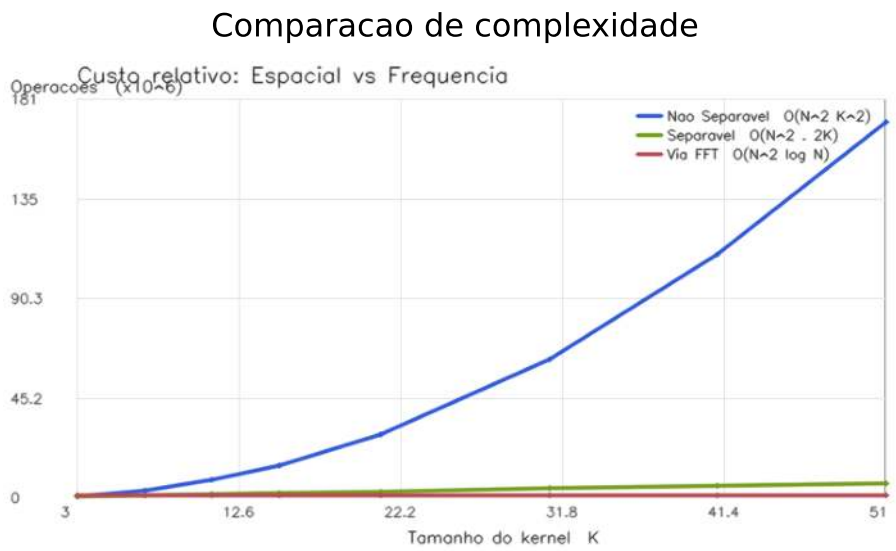


Figura 5.7: Comparação de eficiência: Convolução Não Separável (Espacial 2D), Separável (Espacial 1D) e via FFT.

! El problema de la convolución circular (*wrap-around*)

La Transformada Discreta de Fourier (DFT) asume que la imagen se **extiende periódicamente en el espacio**, es decir, que sus bordes se repiten indefinidamente.
En esta condición, la multiplicación en el dominio de la frecuencia corresponde a una **convolución circular** en el dominio espacial. Como consecuencia, regiones opuestas de la imagen (parte superior e inferior, izquierda y derecha) interactúan artificialmente, tal como se ilustra en Figura 5.8.
La aplicación de *zero-padding* antes de la FFT reduce este efecto al extender la imagen con valores nulos en los bordes, aproximando el resultado a la convolución lineal. Este comportamiento puede interpretarse a la luz del Teorema de la Convolución, presentado en 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 <vector>
#include <string>
#include <cmath>
#include <complex>
#include "morph.hpp"
#include <filesystem>


// Helpers para fftshift manual
static void fftshift_manual(const cv::Mat& src, cv::Mat& dst) {
    int cx = src.cols / 2;
    int cy = src.rows / 2;
    cv::Mat q0(src, cv::Rect(0, 0, cx, cy));
    cv::Mat q1(src, cv::Rect(cx, 0, src.cols - cx, cy));
    cv::Mat q2(src, cv::Rect(0, cy, cx, src.rows - cy));
    cv::Mat q3(src, cv::Rect(cx, cy, src.cols - cx, src.rows - cy));
    cv::Mat tmp;
```

```
    q0.copyTo(tmp);
    q3.copyTo(q0);
    tmp.copyTo(q3);
    q1.copyTo(tmp);
    q2.copyTo(q1);
    tmp.copyTo(q2);
    dst = src; // na prática, com q0..q3 cópias, pode precisar de clone
}

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 é válida (injetada automaticamente)

    //   Definir M e N
    int M = img_gray.h;
    int N = img_gray.w;

    // Simulação de um filtro de deslocamento brutal
    // H_shift[u,v] = exp(-1j * 2*pi * (u*delta_x/M + v*delta_y/N))
    cv::Mat img_gray_float;
    mm::Image img_gray_mm = img_gray;
    cv::Mat img_gray_mat = img_gray_mm;
    img_gray_mat.convertTo(img_gray_float, CV_64F);

    // FFT da imagem
    cv::Mat F_img_complex;
    cv::dft(img_gray_float, F_img_complex, cv::DFT_COMPLEX_OUTPUT);

    // Criar H_shift complexo (centrado no centro da imagem, como em numpy)
    // Em numpy, o loop i,j percorre a imagem sem fftshift - então faremos igual
    int delta = 120;
    cv::Mat H_shift_real(M, N, CV_64F, cv::Scalar(0));
    cv::Mat H_shift_imag(M, N, CV_64F, cv::Scalar(0));

    for (int u = 0; u < M; ++u) {
        for (int v = 0; v < N; ++v) {
            double theta = -2.0 * M_PI * (static_cast<double>(u) * delta / M +
static_cast<double>(v) * delta / N);
            H_shift_real.at<double>(u, v) = std::cos(theta);
            H_shift_imag.at<double>(u, v) = std::sin(theta);
        }
    }

    // Multiplicação no domínio da frequência: F_img * H_shift
    cv::Mat F_img_planes[2];
    cv::split(F_img_complex, F_img_planes);
    cv::Mat mult_real = F_img_planes[0].mul(H_shift_real) - F_img_planes[1].mul(H_shift_imag);
    cv::Mat mult_imag = F_img_planes[0].mul(H_shift_imag) + F_img_planes[1].mul(H_shift_real);

    std::vector<cv::Mat> planes = {mult_real, mult_imag};
    cv::Mat F_filtered;
    cv::merge(planes, F_filtered);

    // IFFT
    cv::Mat img_vazada_float;
    cv::idft(F_filtered, img_vazada_float, cv::DFT_SCALE | cv::DFT_REAL_OUTPUT);

    // Normalizar para visualização
```

```
    cv::Mat img_vazada_vis;
    cv::normalize(img_vazada_float, img_vazada_vis, 0, 255, cv::NORM_MINMAX, CV_8U);

    // Converter para mm::Image para mostrar
    mm::Image img_vazada_vis_img(img_vazada_vis);

    // Mostrar
    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)")
```

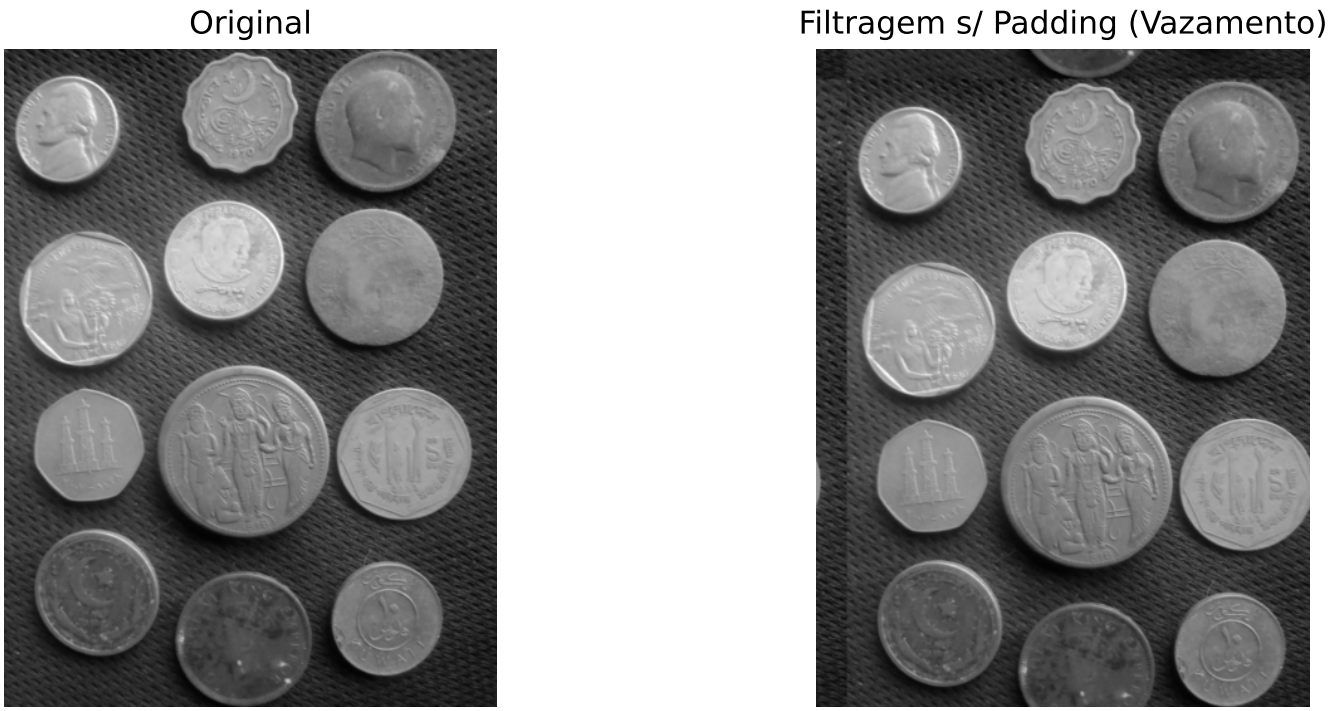


Figura 5.8: Sem *padding*, um deslocamento severo faz a imagem vazar para o lado oposto (convolução circular).

```
%%writefile tmp/fig_05_conv_teorema.cpp
#define MM_OUT "tmp/fig_05_conv_teorema.png"
//| label: fig-05-conv-teorema
//| fig-cap: "Teorema da Convolución: filtrar en el dominio del espacio (Gaussiana) equivale a
multiplicar el espectro por $H(u,v)$ en la frecuencia. Las salidas coinciden visualmente."
//| echo: true
//| output: true

#include <opencv2/opencv.hpp>
#include "morph.hpp"
#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_20.png");
// [pdi:state-io:end]

    // Misma suavización por dos caminos: espacio vs. frecuencia.
    int M = img_gray.h;
    int N = img_gray.w;
    cv::Mat H = mm::gaussFilter(M, N, 30);      // H(u,v): transferencia pasa-bajas gaussiana
    mm::Image f_freq = mm::freqFilter(img_gray, H);    // convolución vía multiplicación en
frecuencia
    mm::Image f_esp = mm::gaussian(img_gray, 31, 5);  // misma suavización hecha en el espacio

    mm::show(
        std::vector<mm::Image>{img_gray, f_esp, f_freq},
        MM_OUT,
        std::vector<std::string>{"Original", "Espacio: Gaussiana", "Frecuencia:
H(u,v).F(u,v)"},
        3
```

```
    );

// [pdi:panel-io] auto-generated - do not edit by hand
std::filesystem::create_directories("tmp");
mm::write(img_gray, "tmp/fig_05_conv_teorema_0.png");
mm::write(f_esp, "tmp/fig_05_conv_teorema_1.png");
mm::write(f_freq, "tmp/fig_05_conv_teorema_2.png");
// [pdi:panel-io:end]
return 0;
}
```

```
Overwriting tmp/fig_05_conv_teorema.cpp
```

```
!g++ -I. -std=c++17 -DMM_USE_OPENCV -I/usr/include/opencv4 tmp/fig_05_conv_teorema.cpp -o
tmp/fig_05_conv_teorema -lopencv_stitching -lopencv_alphamat -lopencv_aruco -lopencv_barcode
-lopencv_bgsegm -lopencv_bioinspired -lopencv_ccalib -lopencv_dnn_objdetect
-lopencv_dnn_superres -lopencv_dpm -lopencv_face -lopencv_freetype -lopencv_fuzzy -lopencv_hdf
-lopencv_hfs -lopencv_img_hash -lopencv_intensity_transform -lopencv_line_descriptor
-lopencv_mcc -lopencv_quality -lopencv_rapid -lopencv_reg -lopencv_rgbd -lopencv_saliency
-lopencv_shape -lopencv_stereo -lopencv_structured_light -lopencv_phase_unwrapping
-lopencv_superres -lopencv_optflow -lopencv_surface_matching -lopencv_tracking
-lopencv_highgui -lopencv_datasets -lopencv_text -lopencv_plot -lopencv_ml -lopencv_videostab
-lopencv_videoio -lopencv_viz -lopencv_wechat_qrcode -lopencv_ximgproc -lopencv_video
-lopencv_xobjdetect -lopencv_objdetect -lopencv_calib3d -lopencv_imgcodecs -lopencv_features2d
-lopencv_dnn -lopencv_flann -lopencv_xphoto -lopencv_photo -lopencv_imgproc -lopencv_core \
  && ./tmp/fig_05_conv_teorema \
  && test -f "tmp/fig_05_conv_teorema.png" \
  || echo "  mm::show não gravou tmp/fig_05_conv_teorema.png"
```

```
[1] Original
[2] Espacio: Gaussiana
[3] Frecuencia: 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 Sobre la diferencia numérica

La diferencia residual del orden de 10^{-13} no viola el Teorema de la Convolución, sino que refleja **limitaciones computacionales** inherentes a la aritmética de coma flotante (doble precisión, ~10^{-16}) y al **orden de las operaciones** entre los dos métodos:

- **Convolución espacial:** suma ponderada de vecinos con redondeos sucesivos.
- **Convolución en frecuencia:** involucra tres transformadas FFT y una multiplicación compleja, sujeta a errores de truncamiento y cuantización.

Por lo tanto, la igualdad teórica es exacta, pero la implementación numérica produce una diferencia prácticamente nula (error relativo $< 10^{-12}$), lo que confirma el teorema dentro de la precisión de la máquina.

5.5 Filtros en el dominio de la frecuencia

Un filtro en el dominio de la frecuencia puede interpretarse como una **función de transferencia aplicada al espectro de la imagen**. En esta representación, cada coeficiente de frecuencia se multiplica por un valor entre 0 y 1, que determina su atenuación o preservación. La forma de esta función define el efecto visual del filtro.

Corte abrupto y *ringing*. Los filtros ideales con transición instantánea en una frecuencia de corte D_0 producen discontinuidades en el dominio de la frecuencia. Esta discontinuidad se refleja en el dominio espacial como oscilaciones cercanas a los bordes, conocidas como *ringing*. Este efecto está asociado a la convolución con funciones de soporte infinito en el espacio, como la función *sinc*, tal como se ilustra en la Figura 5.10.

Filtros con transición suave. Alternativas como los filtros gaussiano y Butterworth suavizan la transición entre las regiones de paso y de rechazo, reduciendo el *ringing*. En contrapartida, esta suavización implica una frontera de separación menos definida entre las frecuencias preservadas y las atenuadas.

```
%%writefile tmp/fig_05_conv_teorema_zoom.cpp
#define MM_OUT "tmp/fig_05_conv_teorema_zoom.png"
#include <opencv2/opencv.hpp>
#include <vector>
#include <string>
#include "morph.hpp"
#include <filesystem>
```

```
int main() {
    //| 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).
    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(
        std::vector<mm::Image>{H_freq, h_space},
        MM_OUT,
        std::vector<std::string>{
            "Frequencia: filtro Ideal (cilindro)",
            "Espaco: ondulacoes da sinc (causa do ringing)"
        },
        2
    );


// [pdi:panel-io] auto-generated - do not edit by hand
std::filesystem::create_directories("tmp");
mm::write(H_freq, "tmp/fig_05_conv_teorema_zoom_0.png");
mm::write(h_space, "tmp/fig_05_conv_teorema_zoom_1.png");
// [pdi:panel-io:end]
return 0;
}
```

```
Overwriting tmp/fig_05_conv_teorema_zoom.cpp
```

```
!g++ -I. -std=c++17 -DMM_USE_OPENCV -I/usr/include/opencv4 tmp/fig_05_conv_teorema_zoom.cpp -o
tmp/fig_05_conv_teorema_zoom -lopencv_stitching -lopencv_alphamat -lopencv_aruco
-lopencv_barcode -lopencv_bgsegm -lopencv_bioinspired -lopencv_ccalib -lopencv_dnn_objdetect
-lopencv_dnn_superres -lopencv_dpm -lopencv_face -lopencv_freetype -lopencv_fuzzy -lopencv_hdf
-lopencv_hfs -lopencv_img_hash -lopencv_intensity_transform -lopencv_line_descriptor
-lopencv_mcc -lopencv_quality -lopencv_rapid -lopencv_reg -lopencv_rgbd -lopencv_saliency
-lopencv_shape -lopencv_stereo -lopencv_structured_light -lopencv_phase_unwrapping
-lopencv_superres -lopencv_optflow -lopencv_surface_matching -lopencv_tracking
-lopencv_highgui -lopencv_datasets -lopencv_text -lopencv_plot -lopencv_ml -lopencv_videostab
-lopencv_videoio -lopencv_viz -lopencv_wechat_qrcode -lopencv_ximgproc -lopencv_video
-lopencv_xobjdetect -lopencv_objdetect -lopencv_calib3d -lopencv_imgcodecs -lopencv_features2d
-lopencv_dnn -lopencv_flann -lopencv_xphoto -lopencv_photo -lopencv_imgproc -lopencv_core \
  && ./tmp/fig_05_conv_teorema_zoom \
  && test -f "tmp/fig_05_conv_teorema_zoom.png" \
  || echo "  mm::show não gravou tmp/fig_05_conv_teorema_zoom.png"
```

```
[1] Frequencia: filtro Ideal (cilindro)
[2] Espaco: ondulacoes da sinc (causa do ringing)
```

```
try:
    mm.show(
        [
            mm.read("tmp/fig_05_conv_teorema_zoom_0.png"),
```

```
            mm.read("tmp/fig_05_conv_teorema_zoom_1.png"),
        ],
        titles=[
            'Frequencia: filtro Ideal (cilindro)',
            'Espaco: ondulacoes da sinc (causa do ringing)',
        ],
        cols=2,
    )
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + "
    tmp/fig_05_conv_teorema_zoom_0.png (ver a versao Python)")
```

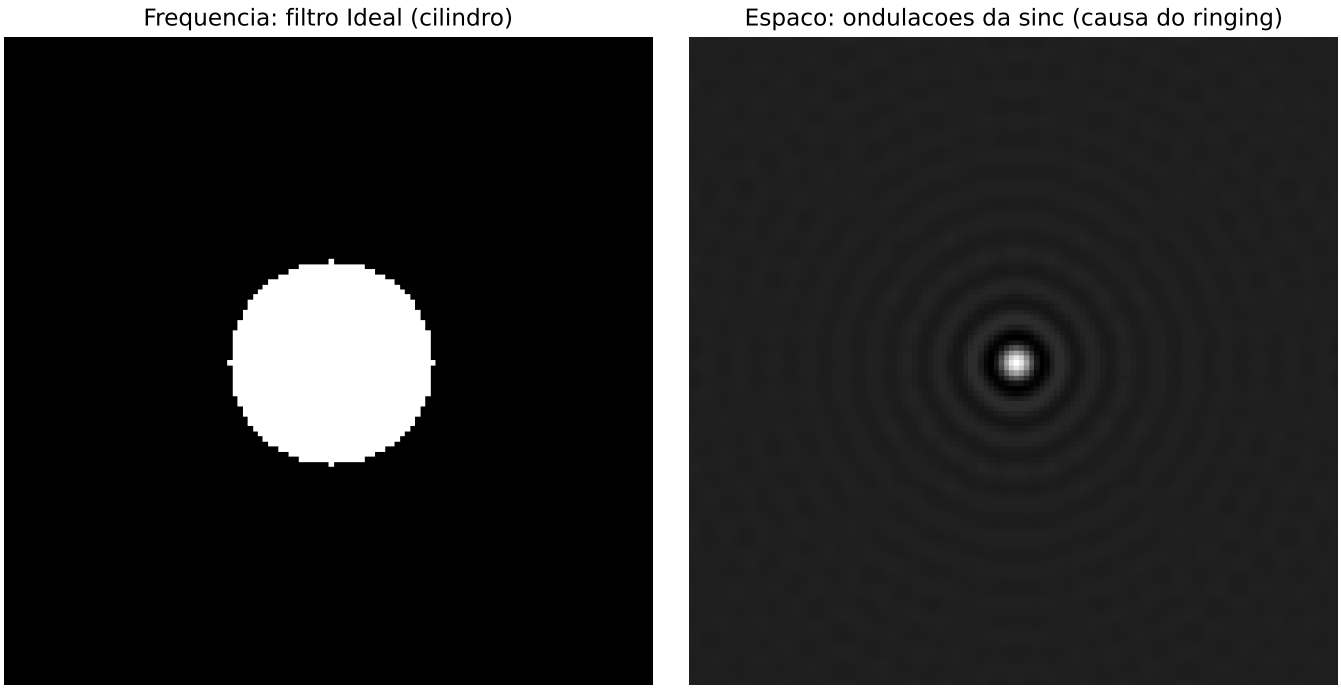


Figura 5.10: A Dualidade Perigosa: o corte abrupto na Frequência (cilindro Ideal) vira obrigatoriamente uma *sinc* no espaço. Suas ondulações causam o *ringing* fantasma nas bordas da imagem.

5.5.1 Filtros Pasa-Bajos

Los filtros pasa-bajos atenúan los componentes de alta frecuencia, lo que resulta en un suavizado de la imagen y una reducción del ruido. Tras la centralización del espectro (FFT Shift), la distancia de cada punto al centro viene dada por:

$$D(u,v) = \sqrt{\left(u - \tfrac{M}{2}\right)^2 + \left(v - \tfrac{N}{2}\right)^2} \tag{5.5}$$

Filtro Ideal (LPFI):

$$H_{\text{ideal}}(u,v) = \begin{cases} 1, & D(u,v) \le D_0 \\ 0, & D(u,v) > D_0 \end{cases} \tag{5.6}$$

El corte abrupto en D_0 introduce discontinuidades en el dominio de la frecuencia, lo que da lugar a oscilaciones en el dominio espacial conocidas como *ringing*. Este efecto está asociado con la convolución con funciones de soporte infinito.

Filtro Gaussiano (LPFG):

$$H_{\text{gauss}}(u,v) = e^{-D^2(u,v)/(2\sigma^2)} \tag{5.7}$$

La suavidad de la función gaussiana en el dominio de la frecuencia evita discontinuidades, lo que elimina el *ringing* y produce una transición gradual entre las frecuencias preservadas y las atenuadas.

Filtro de Butterworth (LPFB) de orden n:

$$H_{\mathrm{BW}}(u,v) = \frac{1}{1 + [D(u,v)/D_0]^{2n}} \tag{5.8}$$

El parámetro n controla la suavidad de la transición entre el paso y el rechazo de frecuencias. Los valores pequeños producen transiciones suaves, mientras que los valores grandes aproximan el comportamiento del filtro ideal, con un mayor riesgo de *ringing*. En la Figura 5.11. se presenta un ejemplo comparativo.

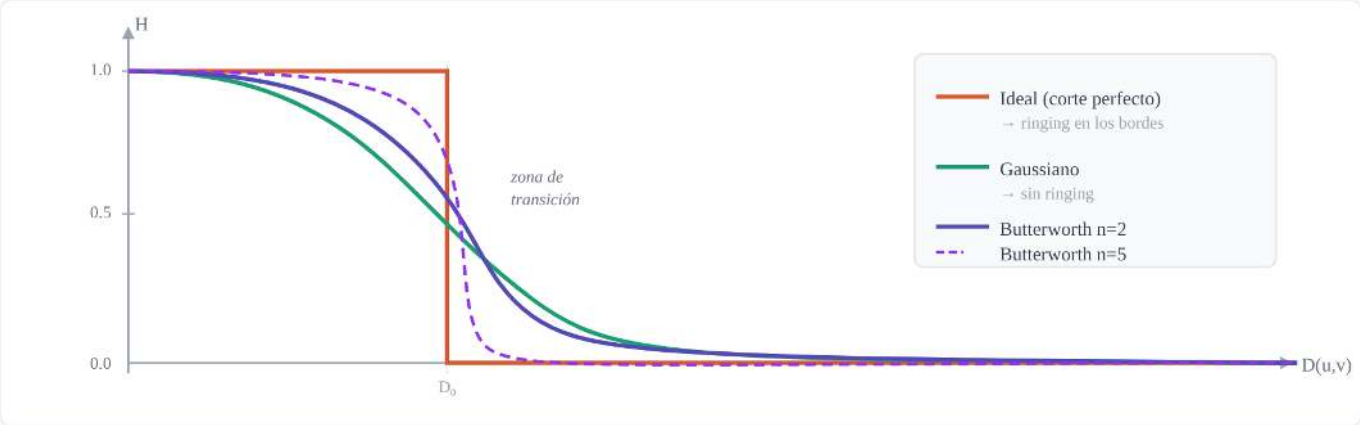


Figura 5.11: Filtros pasa-baja.

5.5.2 Filtros Pasa-Alto y Pasa-Banda

Los filtros pasa-alto pueden obtenerse a partir de un filtro pasa-bajo complementario, definido como:

$$H_{\mathrm{HP}}(u,v) = 1 - H_{\mathrm{LP}}(u,v)$$

Este tipo de filtro preserva componentes de alta frecuencia, resaltando bordes y detalles, mientras atenúa regiones de variación suave.

Los filtros pasa-banda preservan solo un rango intermedio de frecuencias, limitado por dos radios D_L y D_H:

$$H_{\mathrm{BP}}(u,v) = H_{\mathrm{LP}}^{(D_H)}(u,v) \cdot \left[1 - H_{\mathrm{LP}}^{(D_L)}(u,v)\right]$$

Este tipo de filtrado es útil cuando se desea eliminar simultáneamente componentes de baja y alta frecuencia, preservando únicamente estructuras de escala intermedia.

Una aplicación importante es la eliminación de **ruido periódico**, en la cual patrones regulares aparecen como picos localizados en el espectro de magnitud. Estos picos pueden atenuarse mediante filtros *notch* (rechaza-banda), posicionados específicamente en las frecuencias no deseadas.

Ejemplos de filtros en el dominio de la frecuencia se presentan en el simulador de la Figura 5.12, Figura 5.13 y Figura 5.14..

```
%%writefile tmp/fig_05_filtros_freq.cpp
#define MM_OUT "tmp/fig_05_filtros_freq.png"
//| label: fig-05-filtros-freq
```

Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.es/cap05/sim-05-filtros.html

Figura 5.12: Simulador interactivo de filtros en el dominio de la frecuencia.

```
//| fig-cap: "Comparación entre filtros pasa-bajos: Ideal (D =30), Gaussiano (D =30) y
Butterworth (D =30, n=2). Perfiles de H(u,v) a lo largo de una línea central e imágenes
filtradas correspondientes."
//| echo: true
//| output: true

#include <opencv2/opencv.hpp>
#include <vector>
#include <string>
#include <cmath>
#include "morph.hpp"

// Helper function to visualize H(u,v) as an image
static cv::Mat H_vis(const cv::Mat& H) {
    cv::Mat scaled;
    cv::Mat H_uint8;
    // Scale H to 0-255 range and convert to uint8
    H.convertTo(H_uint8, CV_8U, 255.0);
    cv::normalize(H_uint8, scaled, 0, 255, cv::NORM_MINMAX);
    return scaled;
}

// Helper function to draw a profile curve
static void traca(const cv::Mat& row, const cv::Scalar& cor, cv::Mat& perfil, int M, int N,
int Wp, int Hp, int linha) {
    cv::Point ant(-1, -1);
    for (int x = 0; x < N; x++) {
        int px = static_cast<int>(std::round(x * (Wp - 1) / (N - 1)));
        int py = static_cast<int>(std::round(10 + (1.0 - static_cast<float>(row.at<double>(0,
        x))) * (Hp - 20)));
        if (ant.x >= 0) {
            cv::line(perfil, ant, cv::Point(px, py), cor, 2, cv::LINE_AA);
        }
        ant = cv::Point(px, py);
    }
}

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]

    // Convert mm::Image to cv::Mat
    cv::Mat img_gray_cv = img_gray;

    int M = img_gray_cv.rows;
    int N = img_gray_cv.cols;
    double D0 = 30.0;
    int n_bw = 2;

    //   Funciones de transferencia (H centrado) vía constructores de morph.hpp
    //   Ideal:        1 si D<=D0, sino 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);

    //   Imágenes filtradas vía 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);

    //   Perfiles de H(u,v) en la línea central, dibujados con cv::line
    int linha = M / 2;
    int Wp = 512, Hp = 288;
    cv::Mat perfil(Hp, Wp, CV_8UC3, cv::Scalar(255, 255, 255));

    // Extraer la fila central de cada filtro H
    cv::Mat row_ideal = H_ideal.row(linha);
    cv::Mat row_gauss = H_gauss.row(linha);
    cv::Mat row_bw    = H_bw.row(linha);

    traca(row_ideal, cv::Scalar(48, 90, 216), perfil, M, N, Wp, Hp, linha);
    traca(row_gauss, cv::Scalar(117, 158, 29), perfil, M, N, Wp, Hp, linha);
    traca(row_bw,    cv::Scalar(183, 74, 83), perfil, M, N, Wp, Hp, linha);

    // Convert perfil from cv::Mat to mm::Image for display
    mm::Image perfil_img(perfil);

    mm::show(
        std::vector<mm::Image>{img_gray, img_ideal, img_gauss, img_bw,
                               H_vis(H_ideal), H_vis(H_gauss), H_vis(H_bw), perfil_img},
        MM_OUT,
        std::vector<std::string>{
            "Original", "LPF Ideal", "LPF Gaussiano", "LPF Butterworth (n=2)",
            "H Ideal", "H Gaussiano", "H Butterworth", "Perfiles H(u,v)",
        },
        4
    );

    return 0;
}
```

```
Overwriting tmp/fig_05_filtros_freq.cpp
```

```
!g++ -I. -std=c++17 -DMM_USE_OPENCV -I/usr/include/opencv4 tmp/fig_05_filtros_freq.cpp -o
tmp/fig_05_filtros_freq -lopencv_stitching -lopencv_alphamat -lopencv_aruco -lopencv_barcode
-lopencv_bgsegm -lopencv_bioinspired -lopencv_ccalib -lopencv_dnn_objdetect
-lopencv_dnn_superres -lopencv_dpm -lopencv_face -lopencv_freetype -lopencv_fuzzy -lopencv_hdf
-lopencv_hfs -lopencv_img_hash -lopencv_intensity_transform -lopencv_line_descriptor
-lopencv_mcc -lopencv_quality -lopencv_rapid -lopencv_reg -lopencv_rgbd -lopencv_saliency
-lopencv_shape -lopencv_stereo -lopencv_structured_light -lopencv_phase_unwrapping
-lopencv_superres -lopencv_optflow -lopencv_surface_matching -lopencv_tracking
-lopencv_highgui -lopencv_datasets -lopencv_text -lopencv_plot -lopencv_ml -lopencv_videostab
-lopencv_videoio -lopencv_viz -lopencv_wechat_qrcode -lopencv_ximgproc -lopencv_video
-lopencv_xobjdetect -lopencv_objdetect -lopencv_calib3d -lopencv_imgcodecs -lopencv_features2d
-lopencv_dnn -lopencv_flann -lopencv_xphoto -lopencv_photo -lopencv_imgproc -lopencv_core \
  && ./tmp/fig_05_filtros_freq \
  && test -f "tmp/fig_05_filtros_freq.png" \
  || echo "  mm::show não gravou tmp/fig_05_filtros_freq.png"
```

```
[1] Original
[2] LPF Ideal
[3] LPF Gaussiano
[4] LPF Butterworth (n=2)
[5] H Ideal
[6] H Gaussiano
[7] H Butterworth
[8] Perfiles 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)")
```

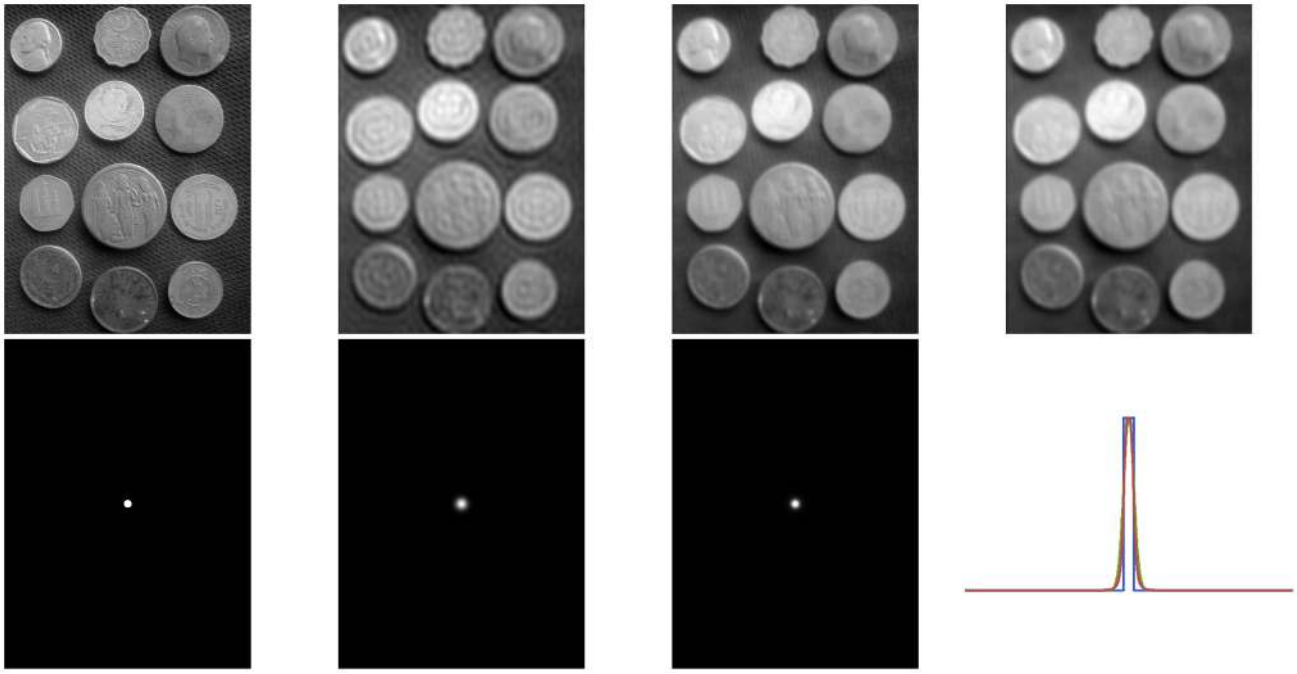


Figura 5.13: Comparação entre filtros passa-baixa: Ideal (D =30), Gaussiano (D =30) e Butterworth (D =30, n=2). Perfis de H(u,v) ao longo de uma linha central e imagens filtradas correspondentes.

```
%%writefile tmp/fig_05_filtros_passa_alta.cpp
#define MM_OUT "tmp/fig_05_filtros_passa_alta.png"
//| 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."

#include <opencv2/opencv.hpp>
#include <vector>
#include <string>
#include "morph.hpp"
#include <filesystem>

// Filtro passa-alta = complemento do passa-baixa Gaussiano (1 - H_gauss),
// construído com mm.gaussFilter(..., highpass=true) e aplicado via FFT.
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 é fornecido automaticamente (mm::Image)

    int M = img_gray.h;
    int N = img_gray.w;
    double D0 = 30;
    cv::Mat H_alta = mm::gaussFilter(M, N, D0, true);
    mm::Image img_alta = mm::freqFilter(img_gray, H_alta);
```

```
    mm::show(
        std::vector<mm::Image>{img_gray, img_alta},
        MM_OUT,
        {"Original", "Passa-alta Gaussiano ($D_0=30$)"},
        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)")
```

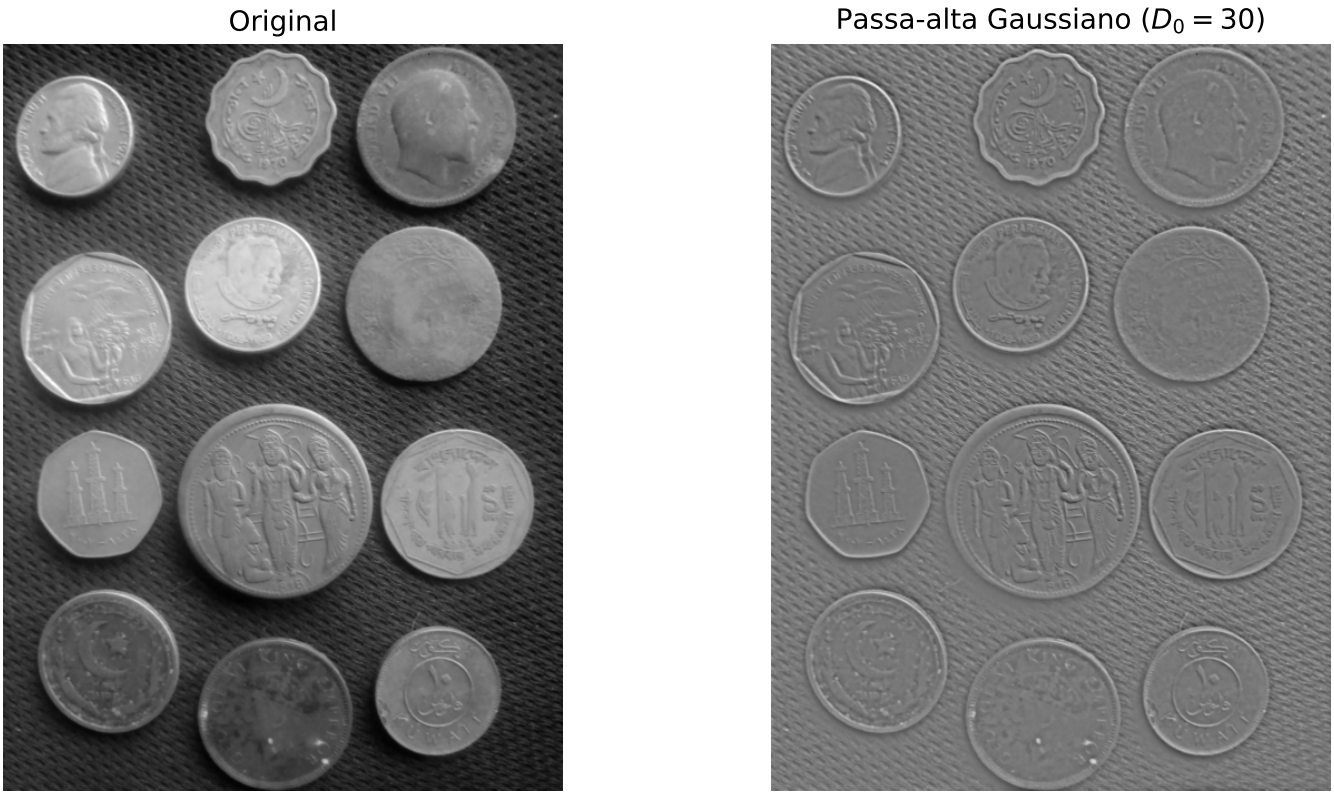


Figura 5.14: Filtro passa-alta Gaussiano. (a) Original; (b) Filtro passa-alta (D =30) - as bordas das moedas e fundo texturizado são realçados.

5.5.3 Eliminación de Ruido Periódico

El ruido periódico — asociado a interferencias eléctricas, patrones regulares de sensores o artefactos de digitalización — aparece en el espectro de Fourier como **picos puntuales simétricos alrededor del centro**.

El filtro **rechaza-banda (*notch*)** atenúa selectivamente esas frecuencias, preservando las demás componentes de la imagen. Un ejemplo de aplicación se presenta en la 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 <vector>
#include <cmath>
#include <string>
#include "morph.hpp"

int main() {
// [pdi:state-io] auto-generated - do not edit by hand
mm::Image img_gray = mm::_read_state("tmp/state/img_gray_20.png");
// [pdi:state-io:end]
```

```
    // Variables pre-loaded: img_gray (mm::Image)

    int h_img = img_gray.h;
    int w_img = img_gray.w;

    // Criar meshgrid equivalente em 2D
    cv::Mat X(h_img, w_img, CV_64F), Y(h_img, w_img, CV_64F);
    for(int i = 0; i < h_img; i++) {
        for(int j = 0; j < w_img; j++) {
            X.at<double>(i,j) = j;
            Y.at<double>(i,j) = i;
        }
    }

    double u0 = 20, v0 = 20;  // frequências exatas do ruído
    cv::Mat ruido(h_img, w_img, CV_64F);
    for(int i = 0; i < h_img; i++) {
        for(int j = 0; j < w_img; j++) {
            ruido.at<double>(i,j) = 40 * std::sin(2 * M_PI * (u0 * j / w_img + v0 * i /
h_img));
        }
    }

    // Converter imagem para double e adicionar ruído
    cv::Mat img_gray_cv = img_gray;  // conversão implícita
    cv::Mat img_float;
    img_gray_cv.convertTo(img_float, CV_64F);
    cv::Mat img_ruidosa_float = img_float + ruido;

    cv::Mat img_ruidosa_8u;
    cv::threshold(img_ruidosa_float, img_ruidosa_float, 0, 255, cv::THRESH_TRUNC);
    cv::threshold(img_ruidosa_float, img_ruidosa_float, 0, 255, cv::THRESH_TOZERO);
    img_ruidosa_float.convertTo(img_ruidosa_8u, CV_8U);
    mm::Image img_ruidosa(img_ruidosa_8u);

    // Espectro de magnitude (log) da imagem ruidosa
    mm::Image mag_vis = mm::spectrumMag(img_ruidosa);

    // Máscara notch: 1.0 em todo o plano, disco de 0.0 em cada um dos 4 picos
    cv::Mat mascara = cv::Mat::ones(h_img, w_img, CV_64F);
    int r_notch = 8;
    int cy = h_img / 2, cx = w_img / 2;

    // Gerar coordenadas de grade
    cv::Mat yy(h_img, w_img, CV_64F), xx(h_img, w_img, CV_64F);
    for(int i = 0; i < h_img; i++) {
        for(int j = 0; j < w_img; j++) {
            yy.at<double>(i,j) = i;
            xx.at<double>(i,j) = j;
        }
    }

    // Aplicar notches nos 4 picos
    int dys[] = {static_cast<int>(v0), -static_cast<int>(v0), static_cast<int>(v0),
    -static_cast<int>(v0)};
    int dxs[] = {static_cast<int>(u0), -static_cast<int>(u0), -static_cast<int>(u0),
    static_cast<int>(u0)};

    for(int k = 0; k < 4; k++) {
        int dy = dys[k], dx = dxs[k];
```

```
        cv::Mat dist;
        cv::Mat diff_y = yy - (cy + dy);
        cv::Mat diff_x = xx - (cx + dx);
        cv::magnitude(diff_x, diff_y, dist);

        for(int i = 0; i < h_img; i++) {
            for(int j = 0; j < w_img; j++) {
                if(dist.at<double>(i,j) <= r_notch) {
                    mascara.at<double>(i,j) = 0.0;
                }
            }
        }
    }

    cv::Mat mascara_8u;
    cv::Mat mascara_scaled = mascara * 255;
    mascara_scaled.convertTo(mascara_8u, CV_8U);
    mm::Image mascara_vis(mascara_8u);

    // Filtragem: aplica a máscara centrada como filtro no domínio da frequência
    mm::Image img_rest_vis = mm::freqFilter(img_ruidosa, mascara);

    // Calcular PSNR
    cv::Mat original_cv = img_gray;
    cv::Mat restaurada_cv = img_rest_vis;
    double psnr = cv::PSNR(original_cv, restaurada_cv);

    std::cout << "PSNR (original vs restaurada): " << psnr << " dB" << std::endl;

    // Construir títulos
    std::string titulo1 = "Com ruído periódico";
    std::string titulo2 = "Espectro (log)";
    std::string titulo3 = "Máscara notch";

    char buffer[100];
    snprintf(buffer, sizeof(buffer), "Restaurada (PSNR=%.1f dB)", psnr);
    std::string titulo4 = buffer;

    std::vector<mm::Image> imagens = {img_ruidosa, mag_vis, mascara_vis, img_rest_vis};
    std::vector<std::string> titulos = {titulo1, titulo2, titulo3, titulo4};

    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): 7.46158 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.

📌 Síntesis — Filtros Espectrales

Filtro	Efecto visual	Artefacto	Uso
Paso-bajo ideal	Suavizado intenso	*Ringing*	Ilustrativo
Paso-bajo gaussiano	Suavizado suave	No presenta *ringing*	Suavizado general
Paso-bajo Butterworth	Suavizado controlado	*Ringing* (órdenes altos)	Compromiso entre suavizado y selectividad
Paso-alto	Realce de bordes	Amplificación de ruido	Detección de contornos
Notch	Eliminación selectiva de frecuencias	Posibles distorsiones locales	Eliminación de ruido periódico

El diseño de filtros en el dominio de la frecuencia consiste en la definición de máscaras espectrales. Sin embargo, efectos en el dominio espacial, como *ringing* y desenfoque, emergen directamente de esas elecciones en el espectro.

5.6 *Wavelets* y Multirresolución

La Transformada de Fourier descompone la señal en frecuencias **globales**: cada coeficiente $F(u, v)$ recibe contribuciones de toda la imagen, sin información explícita sobre la ubicación espacial de esas frecuencias. Así, las estructuras localizadas, como los bordes, se representan de forma distribuida en el espectro.

Las ***wavelets* (ondículas)** superan esta limitación al utilizar funciones base **localizadas en el espacio**, que pueden ser desplazadas y escaladas. Estas funciones poseen **soporte compacto**, es decir, son diferentes

de cero solo en una región finita del dominio, permitiendo una representación simultánea en términos de **frecuencia y ubicación espacial**.

5.6.1 El Límite de la Transformada de Fourier: localización espacial

La Transformada de Fourier describe con precisión **qué frecuencias están presentes** en una señal, pero no representa explícitamente **dónde ocurren esas frecuencias en el espacio**.

En el experimento presentado en la Figura 5.16, dos imágenes con estructuras localizadas en posiciones diferentes producen espectros de magnitud prácticamente idénticos. Esto ocurre porque la representación de Fourier es global: cada coeficiente recibe contribución de toda la imagen.

Como consecuencia, el espectro de magnitud no representa explícitamente la localización de bordes u otras estructuras, solo la distribución de las frecuencias presentes. Esta limitación motivó el desarrollo de representaciones multirresolución, como la Transformada *Wavelet* Discreta (DWT), capaces de describir simultáneamente la frecuencia y la localización espacial de las estructuras de la imagen.

```
%%writefile tmp/fig_05_fracasso_fourier.cpp
#define MM_OUT "tmp/fig_05_fracasso_fourier.png"
//| label: fig-05-fracasso-fourier
//| fig-cap: "Fourier global es ciego a la posición. Los espectros no dicen dónde están los
bordes."
//| echo: true
//| output: true

#include <opencv2/opencv.hpp>
#include <vector>
#include <string>
#include "morph.hpp"
#include <filesystem>

int main() {
    // Crear img_sinal1: ceros de 128x128, columna 20-25 = 1, fila 100-105 = 1
    cv::Mat img_sinal1 = cv::Mat::zeros(128, 128, CV_8UC1);
    img_sinal1.colRange(20, 25).setTo(1);
    img_sinal1.rowRange(100, 105).setTo(1);

    // Crear img_sinal2: ceros de 128x128, columna 90-95 = 1, fila 30-35 = 1
    cv::Mat img_sinal2 = cv::Mat::zeros(128, 128, CV_8UC1);
    img_sinal2.colRange(90, 95).setTo(1);
    img_sinal2.rowRange(30, 35).setTo(1);

    // mag1 = log(1+|FFT shift|)
    cv::Mat img1_64f, img2_64f;
    img_sinal1.convertTo(img1_64f, CV_64F);
    img_sinal2.convertTo(img2_64f, CV_64F);

    cv::Mat fft1, fft2;
    cv::dft(img1_64f, fft1, cv::DFT_COMPLEX_OUTPUT);
    cv::dft(img2_64f, fft2, cv::DFT_COMPLEX_OUTPUT);

    // Función para fftshift de espectros
    auto fftshift = [](cv::Mat& spectrum) {
        cv::Mat shifted;
        int cx = spectrum.cols / 2;
        int cy = spectrum.rows / 2;

        // Cuadrantes para intercambiar
        cv::Mat q0(spectrum, cv::Rect(0, 0, cx, cy));
        cv::Mat q1(spectrum, cv::Rect(cx, 0, spectrum.cols - cx, cy));
```

```
        cv::Mat q2(spectrum, cv::Rect(0, cy, cx, spectrum.rows - cy));
        cv::Mat q3(spectrum, cv::Rect(cx, cy, spectrum.cols - cx, spectrum.rows - cy));

        cv::Mat tmp;
        q0.copyTo(tmp);
        q3.copyTo(q0);
        tmp.copyTo(q3);

        q1.copyTo(tmp);
        q2.copyTo(q1);
        tmp.copyTo(q2);
    };

    fftshift(fft1);
    fftshift(fft2);

    cv::Mat mag1, mag2;
    cv::Mat planes1[2], planes2[2];
    cv::split(fft1, planes1);
    cv::split(fft2, planes2);
    cv::magnitude(planes1[0], planes1[1], mag1);
    cv::magnitude(planes2[0], planes2[1], mag2);
    cv::log(1.0 + mag1, mag1);
    cv::log(1.0 + mag2, mag2);

    // Normalizar y convertir a 8-bit para visualización
    cv::Mat mag1_8u, mag2_8u;
    cv::normalize(mag1, mag1_8u, 0, 255, cv::NORM_MINMAX, CV_8U);
    cv::normalize(mag2, mag2_8u, 0, 255, cv::NORM_MINMAX, CV_8U);

    // Convertir imágenes para mostrar
    mm::Image img1_out(img_sinal1);
    mm::Image img2_out(img_sinal2);
    mm::Image mag1_out(mag1_8u);
    mm::Image mag2_out(mag2_8u);

    mm::show({img1_out, mag1_out, img2_out, mag2_out},
             MM_OUT,
             {"Sinal A", "Espectro A", "Sinal B (Desplazado)", "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 (Desplazado)
[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)")
```

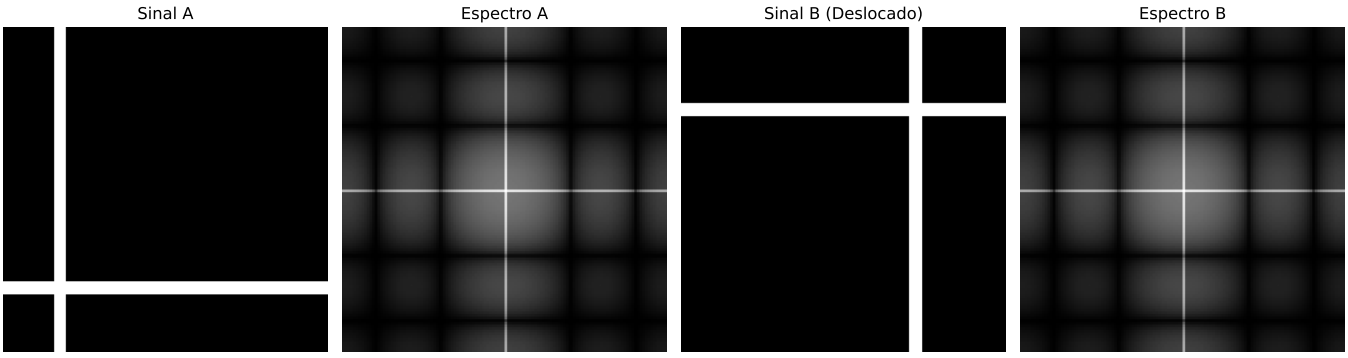


Figura 5.16: Fourier global é cego para a posição. Os espectros não dizem onde as bordas estão.

5.6.2 Transformada Wavelet Discreta 2D

La Transformada Wavelet Discreta (DWT) aplica, de manera separada en las direcciones horizontal y vertical, dos filtros complementarios: un **pasa-bajos** h (aproximación) y un **pasa-altos** g (detalles), seguidos de un submuestreo por un factor de 2 en cada dimensión. Este proceso produce cuatro subbandas, cuyos nombres indican la combinación de los filtros aplicados en cada dirección (L = *Low-pass*, pasa-bajos; H = *High-pass*, pasa-altos). Las características de cada subbanda se resumen en la Tabla 5.3.

$$\mathrm{DWT}(f) = \{\ \underbrace{\mathrm{LL}}_{\text{aprox.}}\ ,\quad \underbrace{\mathrm{LH}}_{\text{detalles horizontales}}\ ,\quad \underbrace{\mathrm{HL}}_{\text{detalles verticales}}\ ,\quad \underbrace{\mathrm{HH}}_{\text{detalles diagonales}}\ \}.$$

Tabla 5.3: Subbandas producidas por la Transformada Wavelet Discreta 2D (DWT), indicando los filtros aplicados en cada dirección y el contenido predominante de cada componente.

Subbanda	Filtros aplicados	Contenido visual
LL	baja × baja	Aproximación de la imagen (versión suavizada y reducida)
LH	baja × alta	Bordes horizontales y variaciones verticales
HL	alta × baja	Bordes verticales y variaciones horizontales
HH	alta × alta	Detalles diagonales y texturas

La descomposición puede aplicarse recursivamente sobre la subbanda LL, generando una representación multirresolución. Tras J niveles, se obtiene una estructura con $3J + 1$ subbandas, donde cada nuevo nivel reduce la resolución de la componente de aproximación.

> **Conexión con CNNs**
>
> La descomposición multirresolución de las *wavelets* posee una relación conceptual con las representaciones jerárquicas utilizadas en redes neuronales convolucionales (CNNs). En ambos casos, sucesivas etapas de filtrado y reducción de resolución producen descripciones cada vez más abstractas de la imagen. Sin embargo, las ***wavelets* utilizan filtros matemáticamente definidos y reconstruibles**, mientras que las **CNNs aprenden sus filtros durante el entrenamiento**.

5.6.3 Familias de *Wavelets*

Diferentes familias de *wavelets* presentan compromisos distintos entre **soporte espacial**, suavidad y capacidad de compresión. El **soporte** corresponde a la extensión de la función *wavelet* en el dominio espacial: cuanto menor es el soporte, más localizada está la función; cuanto mayor es, más suave tiende a ser su representación, aunque con un mayor costo computacional. La Tabla 5.4 compara algunas de las familias más utilizadas.

Tabla 5.4: Comparación entre familias de *wavelets*, destacando la longitud del soporte, el número de momentos nulos, la simetría y las aplicaciones típicas.

Wavelet	Longitud del soporte	Momentos nulos	Simetría	Uso típico
Haar	2	1	Asimétrica	Introducción y análisis básico
Daubechies db4	8	4	Asimétrica	Compresión y análisis general
Symlet sym4	8	4	Casi simétrica	Reconstrucción de señales

Wavelet	Longitud del soporte	Momentos nulos	Simetría	Uso típico
Biortogonal 5/3	5/3	2/2	Simétrica	JPEG 2000 sin pérdida
Biortogonal 9/7	9/7	4/4	Simétrica	JPEG 2000 con pérdida

Los **momentos nulos** miden la capacidad de la *wavelet* para representar regiones suaves de la imagen con pocos coeficientes distintos de cero. Una *wavelet* con p momentos nulos anula exactamente polinomios de grado hasta $p-1$. En consecuencia, cuanto mayor es el número de momentos nulos, mayor tiende a ser la eficiencia de compresión en regiones homogéneas, aunque esto generalmente implica funciones con un soporte más largo.

La Figura 5.17 presenta las funciones de base (*wavelets*) $\psi(t)$ en el dominio espacial. Estas funciones poseen **soporte compacto**, es decir, son distintas de cero solo en una región finita del dominio, a diferencia de las sinusoides de la Transformada de Fourier, que se extienden por todo el dominio.

```
%%writefile tmp/fig_05_wavelet_functions.cpp
#define MM_OUT "tmp/fig_05_wavelet_functions.png"
#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);

    mm::Image chart_h = mm::lineChart(
        std::vector<std::vector<double>>{x_h},
        std::vector<std::vector<double>>{psi_h},
        std::vector<std::string>{"psi Haar"},
        std::vector<cv::Scalar>{cv::Scalar(180, 60, 40)},
        "Ondaleta Haar (psi)", "t");

    mm::Image chart_d = mm::lineChart(
        std::vector<std::vector<double>>{x_d},
        std::vector<std::vector<double>>{psi_d},
        std::vector<std::string>{"psi Daubechies 4"},
        std::vector<cv::Scalar>{cv::Scalar(60, 140, 40)},
        "Ondaleta Daubechies 4 (psi)", "t");

    mm::show(std::vector<mm::Image>{chart_h, chart_d}, MM_OUT,
             std::vector<std::string>{"Haar", "Daubechies 4"}, 2);


// [pdi:panel-io] auto-generated - do not edit by hand
std::filesystem::create_directories("tmp");
mm::write(chart_h, "tmp/fig_05_wavelet_functions_0.png");
mm::write(chart_d, "tmp/fig_05_wavelet_functions_1.png");
// [pdi:panel-io:end]
return 0;
}
```

```
Overwriting tmp/fig_05_wavelet_functions.cpp
```

```
!g++ -I. -std=c++17 -DMM_USE_OPENCV -I/usr/include/opencv4 tmp/fig_05_wavelet_functions.cpp -o
tmp/fig_05_wavelet_functions -lopencv_stitching -lopencv_alphamat -lopencv_aruco
-lopencv_barcode -lopencv_bgsegm -lopencv_bioinspired -lopencv_ccalib -lopencv_dnn_objdetect
-lopencv_dnn_superres -lopencv_dpm -lopencv_face -lopencv_freetype -lopencv_fuzzy -lopencv_hdf
-lopencv_hfs -lopencv_img_hash -lopencv_intensity_transform -lopencv_line_descriptor
-lopencv_mcc -lopencv_quality -lopencv_rapid -lopencv_reg -lopencv_rgbd -lopencv_saliency
-lopencv_shape -lopencv_stereo -lopencv_structured_light -lopencv_phase_unwrapping
-lopencv_superres -lopencv_optflow -lopencv_surface_matching -lopencv_tracking
-lopencv_highgui -lopencv_datasets -lopencv_text -lopencv_plot -lopencv_ml -lopencv_videostab
-lopencv_videoio -lopencv_viz -lopencv_wechat_qrcode -lopencv_ximgproc -lopencv_video
-lopencv_xobjdetect -lopencv_objdetect -lopencv_calib3d -lopencv_imgcodecs -lopencv_features2d
-lopencv_dnn -lopencv_flann -lopencv_xphoto -lopencv_photo -lopencv_imgproc -lopencv_core \
  && ./tmp/fig_05_wavelet_functions \
  && test -f "tmp/fig_05_wavelet_functions.png" \
  || echo "  mm::show não gravou tmp/fig_05_wavelet_functions.png"
```

```
[1] Haar
[2] Daubechies 4
```

```
try:
    mm.show(
        [
            mm.read("tmp/fig_05_wavelet_functions_0.png"),
            mm.read("tmp/fig_05_wavelet_functions_1.png"),
        ],
        titles=[
            'Haar',
            'Daubechies 4',
        ],
        cols=2,
    )
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + "
    tmp/fig_05_wavelet_functions_0.png (ver a versao Python)")
```

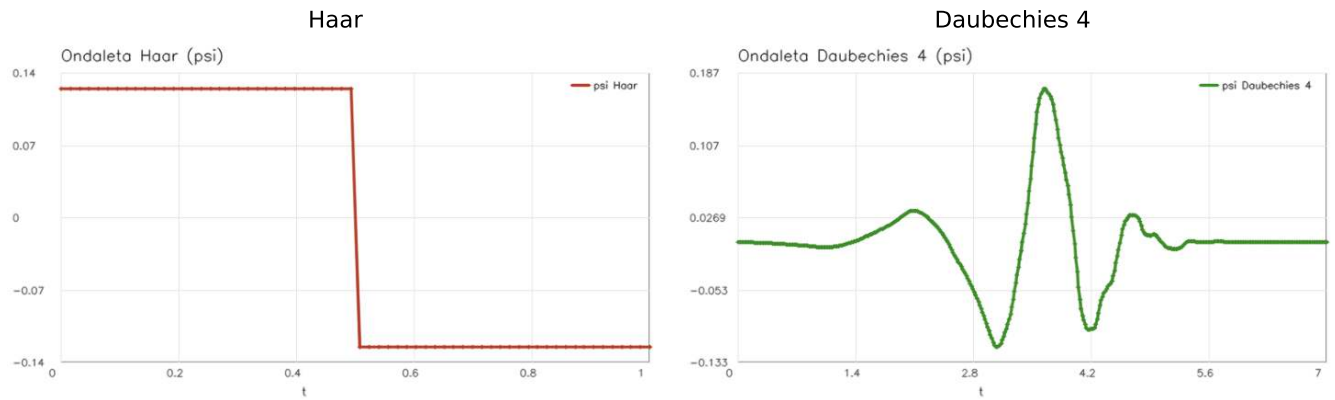


Figura 5.17: Funções da *Wavelet* ( ). Note como elas rapidamente decaem para zero (suporte compacto), ao contrário das senoides infinitas de Fourier.

El diagrama de la Figura 5.18 ilustra el análisis multirresolución realizado por la DWT, en el cual la subbanda de aproximación (LL) se descompone sucesivamente, formando una representación jerárquica con dos niveles.

El simulador de la Figura 5.19 permite explorar interactivamente la Transformada *Wavelet* Discreta 2D (DWT) utilizando la *wavelet* de Haar. La descomposición en subbandas evidencia la separación entre la componente de aproximación y las componentes de detalle de la imagen.

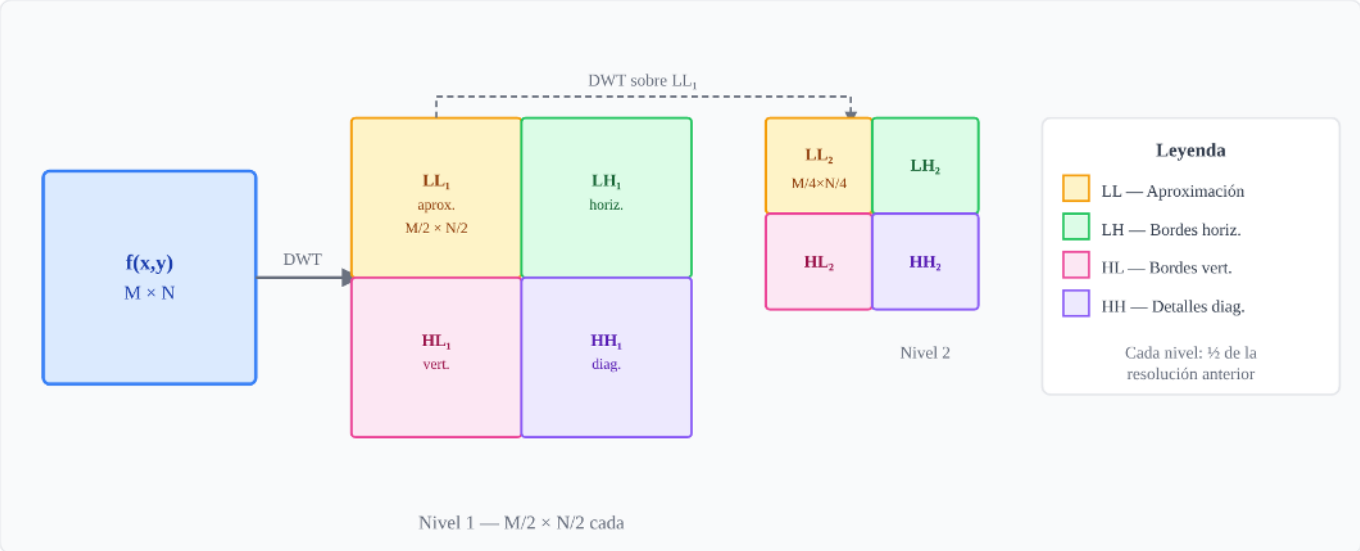


Figura 5.18: Diagrama de la descomposición *wavelet* 2D en dos niveles.

Los diferentes patrones de entrada permiten observar el comportamiento direccional de los filtros. En imágenes con bordes horizontales y verticales, las subbandas LH y HL destacan, respectivamente, las variaciones verticales y horizontales de la intensidad. En regiones de variación suave, la mayor parte de la energía se concentra en la subbanda de aproximación LL, mientras que las subbandas de detalle presentan coeficientes cercanos a cero.

El análisis multirresolución también se puede observar al aumentar el número de niveles de descomposición. En este caso, solo la subbanda LL_1 se descompone nuevamente, dando origen a las subbandas LL_2, LH_2, HL_2 y HH_2, que forman el segundo nivel de la representación jerárquica.

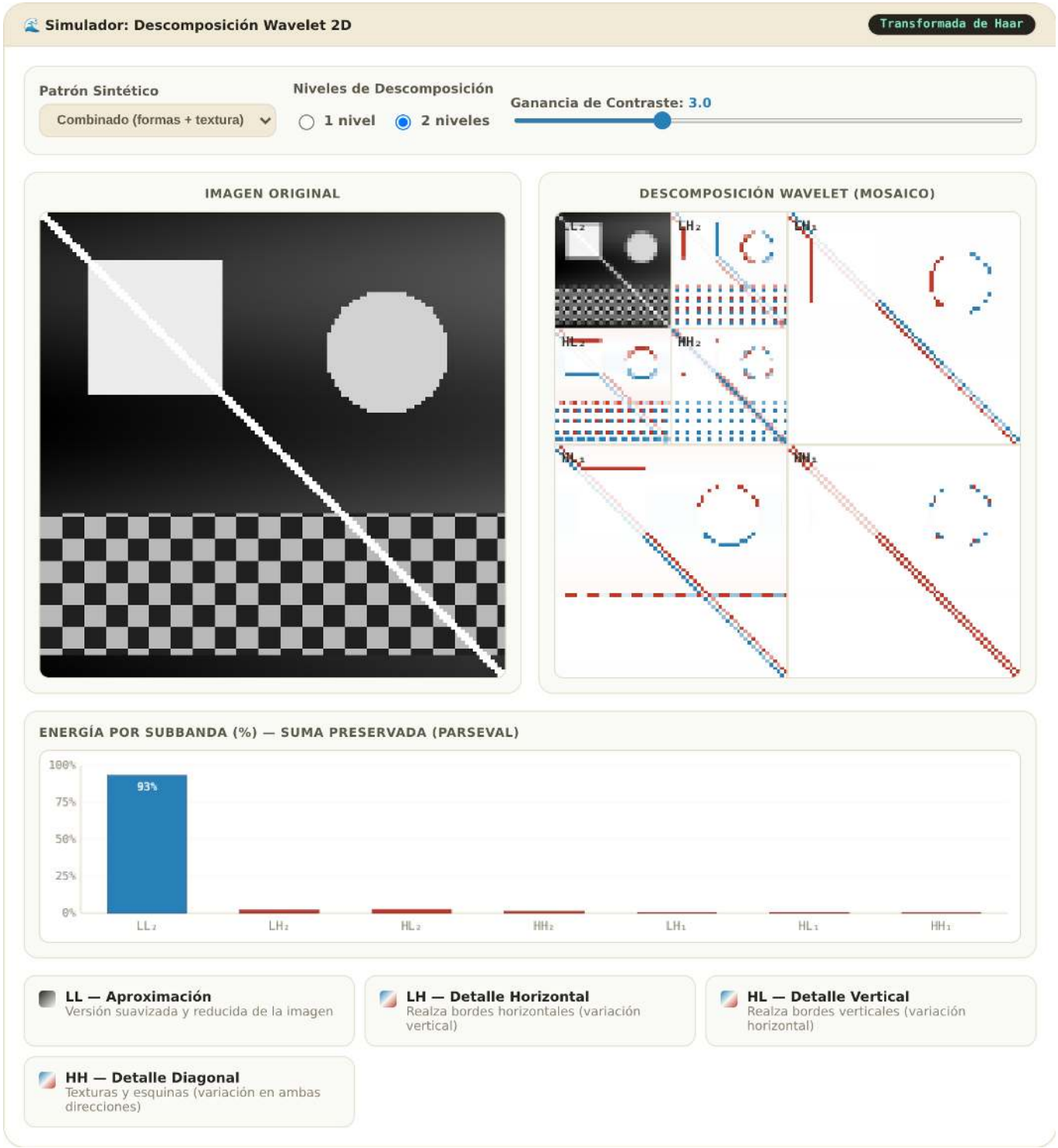
En patrones formados por regiones homogéneas de gran extensión, como un degradado suave o un tablero compuesto por bloques grandes, la energía permanece predominantemente concentrada en la subbanda LL. En el degradado, esto ocurre porque las diferencias entre píxeles vecinos son pequeñas. En el tablero, por su parte, los píxeles poseen prácticamente la misma intensidad en el interior de cada bloque, de modo que solo las fronteras entre bloques producen coeficientes no nulos en las subbandas de detalle. Como estas fronteras ocupan solo una pequeña fracción de la imagen, su contribución a la energía total permanece reducida.

Para posibilitar el análisis visual de estas variaciones sutiles, el simulador incorpora un control de ganancia de contraste de los detalles (que varía de 1 a 8). Este parámetro funciona como un factor de amplificación lineal aplicado exclusivamente a los coeficientes de las subbandas de detalle (LH, HL y HH) antes de su renderización en pantalla. En escenarios de transición suave (como el gradiente) o de uniformidad local (como el interior de los bloques del tablero), las diferencias numéricas calculadas por el filtro paso-alto de Haar resultan en coeficientes muy cercanos a cero, lo que haría que los cuadrantes correspondientes fueran oscuros e imperceptibles a simple vista. Al multiplicar estos valores por la ganancia, el simulador recupera visualmente las estructuras de alta frecuencia ocultas y resalta la orientación de los bordes remanentes.

El gráfico de energía por subbanda cuantifica esta distribución entre la componente de aproximación y las componentes de detalle, demostrando que la ganancia visual no altera la métrica original de la energía. En imágenes naturales, la mayor parte de la energía se concentra en la subbanda LL, mientras que las subbandas LH, HL y HH representan principalmente bordes, texturas y otras variaciones locales de la intensidad.

5.6.4 Análisis Multirresolución con la DWT 2D

La Transformada Wavelet Discreta 2D (DWT) descompone una imagen en componentes de aproximación y detalle, organizadas de forma jerárquica en diferentes escalas y orientaciones. Como las subbandas de detalle en imágenes naturales frecuentemente presentan coeficientes de bajo contraste, los ejemplos prácticos a continuación utilizan un patrón geométrico sintético generado en Python. Este enfoque replica



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.es/cap05/sim-05-wavelet.html

Figura 5.19: Simulación de la descomposición *wavelet* 2D.

el comportamiento del simulador de la Figura 5.19, haciendo visualmente explícitos los efectos del filtrado espacial y de la descomposición multirresolución.

5.6.4.1 Descomposición en Mosaico de Múltiples Niveles

La Figura 5.20 ilustra la estructura jerárquica de la DWT en dos niveles utilizando la *wavelet* de Haar. El proceso se basa en la aplicación combinada de filtros pasa-baja y pasa-alta en las direcciones horizontal y vertical, seguidos por submuestreo por un factor de 2.

En el primer nivel, la imagen original origina la subbanda de aproximación (LL_1) y las componentes de detalle horizontal (LH_1), vertical (HL_1) y diagonal (HH_1). En el análisis multirresolución, la subbanda LL_1 se filtra y submuestrea nuevamente, generando el segundo nivel de descomposición (LL_2, LH_2, HL_2 y HH_2).

Para viabilizar la interpretación visual de las componentes de detalle, el código extrae el valor absoluto de sus coeficientes y aplica una normalización lineal (*min-max*) para ocupar toda la gama dinámica de tonos de gris [0, 255]. Esta operación transforma regiones homogéneas (coeficientes nulos) en negro y destaca en blanco los bordes y texturas extraídas en cada escala y orientación.

```
%%writefile tmp/fig_05_dwt_subbandas.cpp
#define MM_OUT "tmp/fig_05_dwt_subbandas.png"
// Compile: g++ -std=c++17 -o snippet snippet.cpp $(pkg-config --cflags --libs opencv4)
#include <opencv2/opencv.hpp>
#include <iostream>
#include <vector>
#include <string>
#include <cmath>
#include "morph.hpp"

//| 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

//   Geração da Imagem Sintética (Mesmo padrão 'combined' do simulador)
cv::Mat gerar_imagem_sintetica(int N = 256) {
    cv::Mat img = cv::Mat::zeros(N, N, CV_64F);
    for (int y = 0; y < N; ++y) {
        for (int x = 0; x < N; ++x) {
            double v = 55 + 35 * (static_cast<double>(x) / N) + 15 *
std::sin(static_cast<double>(y) / 24);
            // Quadrado
            if (24 < x && x < 100 && 24 < y && y < 100) {
                v = 225;
            }
            // Círculo
            int cx = 190, cy = 76, r = 34;
            if ((x - cx) * (x - cx) + (y - cy) * (y - cy) < r * r) {
                v = 205;
            }
            // Textura periódica (inferior)
            if (y > 164 && y < 244) {
                int p = 12;
                v = (((x / p) + (y / p)) % 2 == 0) ? 185.0 : 65.0;
            }
            // Linha diagonal
            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 img8;
    img.convertTo(img8, CV_8U);
    return img8;
}

// Normaliza subbanda para visualização [0,255]
cv::Mat sb_vis(const cv::Mat& sb) {
    cv::Mat abs_sb;
    cv::absdiff(sb, cv::Scalar(0), abs_sb);
    cv::Mat normalized;
    cv::normalize(abs_sb, normalized, 0, 255, cv::NORM_MINMAX, CV_8U);
    return normalized;
}

int main() {
    // Substitui a imagem escura de moedas pelo padrão sintético claro
    cv::Mat img_gray = gerar_imagem_sintetica(256);

    //   Decomposição wavelet 2 níveis
    std::string wavelet = "haar";
    cv::Mat img_float;
    img_gray.convertTo(img_float, CV_64F);

    // Nível 1
    mm::Subbands coefs1 = mm::dwt2(img_float, wavelet);
    cv::Mat LL1 = coefs1.LL;
    cv::Mat LH1 = coefs1.LH;
    cv::Mat HL1 = coefs1.HL;
    cv::Mat HH1 = coefs1.HH;

    // Nível 2 (aplicado sobre LL1)
    mm::Subbands coefs2 = mm::dwt2(LL1, wavelet);
    cv::Mat LL2 = coefs2.LL;
    cv::Mat LH2 = coefs2.LH;
    cv::Mat HL2 = coefs2.HL;
    cv::Mat HH2 = coefs2.HH;

    std::cout << "Forma original     : " << img_gray.rows << "x" << img_gray.cols <<
std::endl;
    std::cout << "LL1 (nível 1)      : " << LL1.rows << "x" << LL1.cols << "  |  LH1/HL1/HH1:
" << LH1.rows << "x" << LH1.cols << std::endl;
    std::cout << "LL2 (nível 2)      : " << LL2.rows << "x" << LL2.cols << "    |
LH2/HL2/HH2: " << LH2.rows << "x" << LH2.cols << std::endl;

    std::vector<mm::Image> imgs_dwt = {mm::Image(img_gray),
                                       mm::Image(sb_vis(LL1)), mm::Image(sb_vis(LH1)),
                                       mm::Image(sb_vis(HL1)), mm::Image(sb_vis(HH1)),
                                       mm::Image(sb_vis(LL2)), mm::Image(sb_vis(LH2)),
                                       mm::Image(sb_vis(HL2)), mm::Image(sb_vis(HH2))};
    std::vector<std::string> titles_dwt = {"Original",
                                           "LL (aprox.)", "LH (horiz.)", "HL (vert.)", "HH
                                           (diag.)",
                                           "LL (aprox.)", "LH (horiz.)", "HL (vert.)", "HH
                                           (diag.)"};

    mm::show(imgs_dwt, MM_OUT, titles_dwt, 5);
```

```
    return 0;
}
```

```
Overwriting tmp/fig_05_dwt_subbandas.cpp
```

```
!g++ -I. -std=c++17 -DMM_USE_OPENCV -I/usr/include/opencv4 tmp/fig_05_dwt_subbandas.cpp -o
tmp/fig_05_dwt_subbandas -lopencv_stitching -lopencv_alphamat -lopencv_aruco -lopencv_barcode
-lopencv_bgsegm -lopencv_bioinspired -lopencv_ccalib -lopencv_dnn_objdetect
-lopencv_dnn_superres -lopencv_dpm -lopencv_face -lopencv_freetype -lopencv_fuzzy -lopencv_hdf
-lopencv_hfs -lopencv_img_hash -lopencv_intensity_transform -lopencv_line_descriptor
-lopencv_mcc -lopencv_quality -lopencv_rapid -lopencv_reg -lopencv_rgbd -lopencv_saliency
-lopencv_shape -lopencv_stereo -lopencv_structured_light -lopencv_phase_unwrapping
-lopencv_superres -lopencv_optflow -lopencv_surface_matching -lopencv_tracking
-lopencv_highgui -lopencv_datasets -lopencv_text -lopencv_plot -lopencv_ml -lopencv_videostab
-lopencv_videoio -lopencv_viz -lopencv_wechat_qrcode -lopencv_ximgproc -lopencv_video
-lopencv_xobjdetect -lopencv_objdetect -lopencv_calib3d -lopencv_imgcodecs -lopencv_features2d
-lopencv_dnn -lopencv_flann -lopencv_xphoto -lopencv_photo -lopencv_imgproc -lopencv_core \
  && ./tmp/fig_05_dwt_subbandas \
  && test -f "tmp/fig_05_dwt_subbandas.png" \
  || echo "  mm::show não gravou tmp/fig_05_dwt_subbandas.png"
```

```
Forma original     : 256x256
LL1 (nível 1)      : 128x128  |  LH1/HL1/HH1: 128x128
LL2 (nível 2)      : 64x64    |  LH2/HL2/HH2: 64x64
[1] Original
[2] LL  (aprox.)
[3] LH  (horiz.)
[4] HL  (vert.)
[5] HH  (diag.)
[6] LL  (aprox.)
[7] LH  (horiz.)
[8] HL  (vert.)
[9] HH  (diag.)
```

```
try:
    mm.show(mm.read("tmp/fig_05_dwt_subbandas.png"), figsize=(16, 7))
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + "
    tmp/fig_05_dwt_subbandas.png (ver a versao Python)")
```

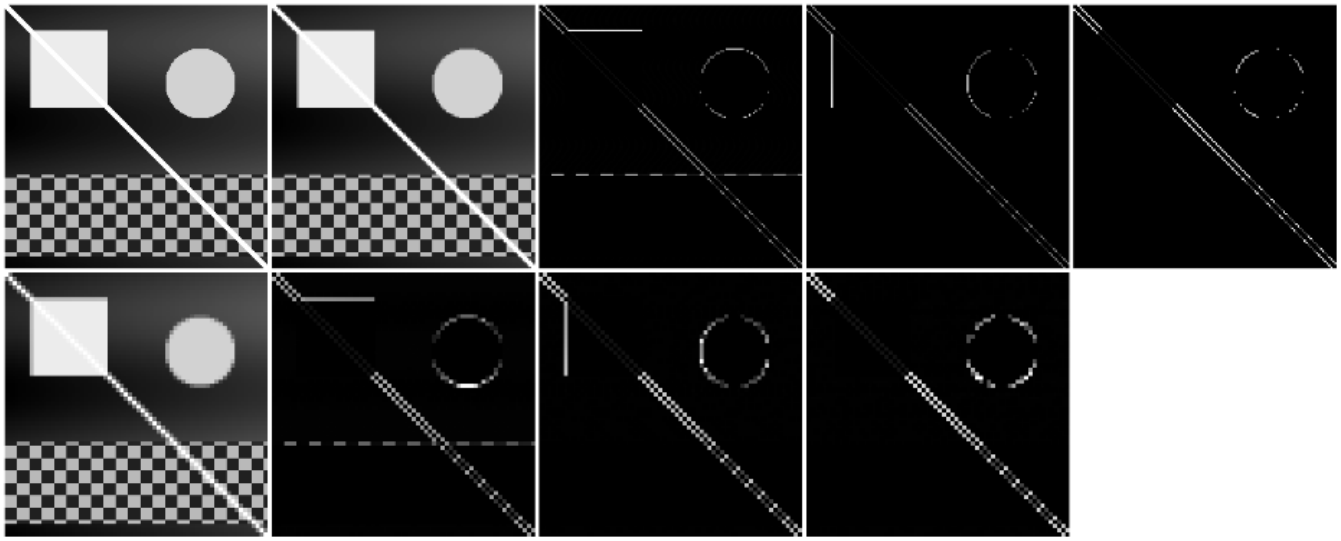


Figura 5.20: Decomposição *wavelet* 2D de 2 níveis com *wavelet* Haar: subbandas LL, LH, HL, HH em cada nível. As subbandas de detalhe revelam estruturas orientadas em diferentes escalas utilizando um padrão sintético.

5.6.4.2 El Compromiso entre Localización y Suavidad

La elección de la función de base (*wavelet*) influye directamente en la forma en que las características de la imagen se distribuyen y codifican mediante los coeficientes de la DWT. La Figura 5.21 compara los resultados prácticos obtenidos al aplicar cuatro familias distintas sobre el patrón geométrico sintético: `haar`, `db4`, `sym4` y `bior2.2`.

Por poseer soporte corto y formato de función escalón, la *wavelet* de Haar produce coeficientes altamente localizados en las discontinuidades espaciales, generando bordes finos y nítidos en las subbandas de detalle. En contrapartida, familias como Daubechies (`db4`) y Symlets (`sym4`), que presentan mayor soporte (filtros más largos) y mayor número de momentos nulos, generan respuestas más suaves y distribuidas alrededor de las transiciones, lo que puede introducir ligeras oscilaciones o desenfoques en las fronteras abruptas.

Este comportamiento evidencia el clásico compromiso (*trade-off*) del análisis de multirresolución: soportes menores favorecen la localización espacial exacta de los bordes, mientras que soportes mayores y un mayor número de momentos nulos tienden a producir representaciones más dispersas y suaves. Esa suavidad y capacidad de atenuación de altas frecuencias garantizan una mayor eficiencia en la compactación de la energía, características fundamentales para aplicaciones de compresión de datos y eliminación de ruido (*denoising*).

```
%%writefile tmp/fig_05_dwt_wavelets.cpp
#define MM_OUT "tmp/fig_05_dwt_wavelets.png"
//| label: fig-05-dwt-wavelets
//| fig-cap: "Comparação entre famílias de *wavelets*: Haar, db4, sym4 e bior2.2. Subbanda LL (aproximação) e HH  (diagonal) para cada escolha, ilustrando o compromisso entre compactação e suavidade com base no padrão sintético."
//| echo: true
//| output: true

#include <opencv2/opencv.hpp>
#include <vector>
#include <string>
#include <cmath>
#include "morph.hpp"

// Garante que img_gray e img_float utilizem o mesmo padrão sintético claro
// (Fallback caso o bloco anterior não tenha sido executado na mesma sessão)
```

```
static 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;
            img.at<double>(y, x) = std::max(0.0, std::min(255.0, v));
        }
    }
    cv::Mat img8;
    img.convertTo(img8, CV_8U);
    return img8;
}

// Visor de subbanda (assume que existe em morph.hpp ou é implementado aqui)
static mm::Image sb_vis(const cv::Mat& sb) {
    cv::Mat normalized;
    cv::normalize(sb, normalized, 0, 255, cv::NORM_MINMAX, CV_8U);
    return mm::Image(normalized);
}

int main() {
    cv::Mat img_gray;
    // Se a variável global estiver disponível (injetada por célula anterior), use-a;
    // caso contrário, gere o fallback.
    // (Na prática, como não há persistência real entre células neste contexto,
    //  sempre geramos o fallback.)
    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::Subbands via morph.hpp
        mm::Subbands s = mm::dwt2(img_float, wname);
        cv::Mat LL = s.LL;
        cv::Mat HH = s.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)")
```

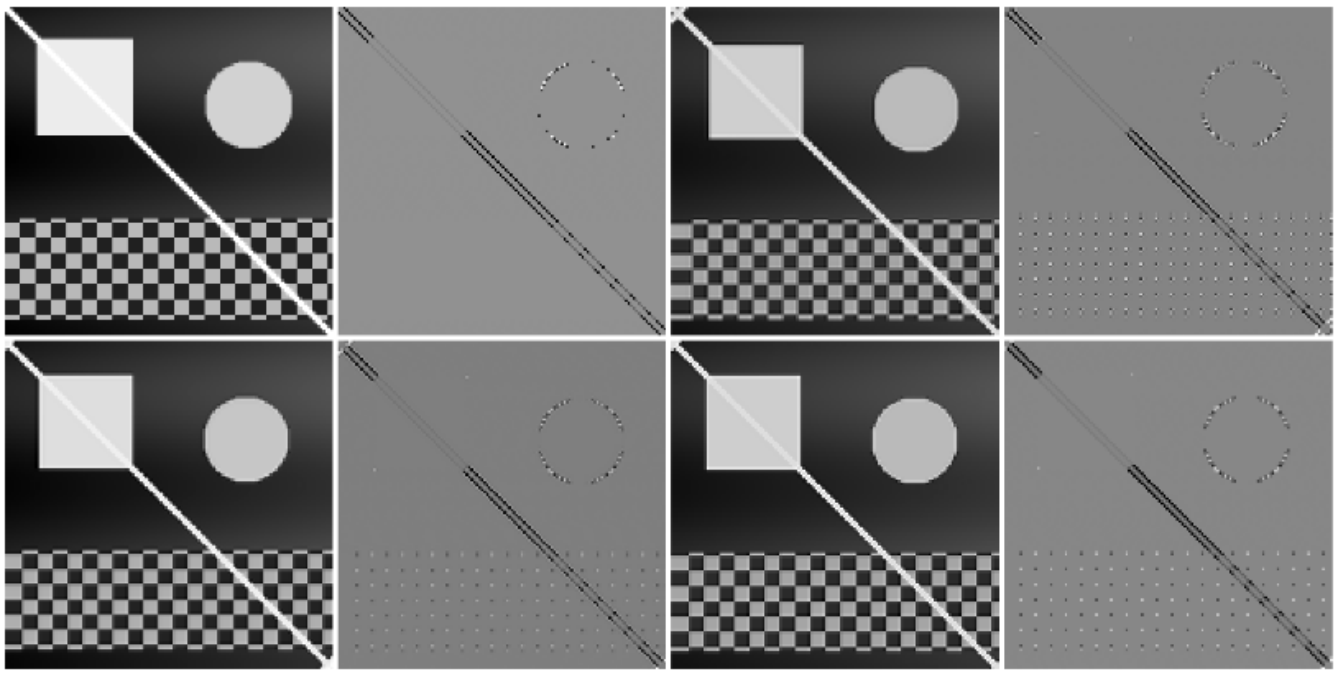


Figura 5.21: Comparação entre famílias de *wavelets*: Haar, db4, sym4 e bior2.2. Subbanda LL (aproximação) e HH (diagonal) para cada escolha, ilustrando o compromisso entre compactação e suavidade com base no padrão sintético.

5.6.4.3 Limiarización de Coeficientes y Compresión

Una de las principales aplicaciones de la Transformada *Wavelet* Discreta (DWT) es la compresión de datos, impulsada por la capacidad de representación **dispersa** de los coeficientes. La Figura 5.22 ilustra el efecto de la limiarización abrupta (*hard thresholding*), técnica en la cual los coeficientes de detalle con magnitud inferior a un umbral T se anulan por completo antes del proceso de síntesis realizado por la Transformada *Wavelet* Discreta Inversa (IDWT).

A medida que se eleva el umbral T, un volumen creciente de coeficientes de alta frecuencia se pone a cero. Al concentrar menor energía, la eliminación de estos componentes reduce considerablemente la cantidad de información necesaria para representar la imagen, manteniendo intacta la componente de aproximación global (la subbanda LL más profunda) para preservar la estructura macro. Visualmente, este descarte de coeficientes se manifiesta a través de la desaparición progresiva de texturas finas y el suavizado de transiciones abruptas de intensidad.

La fidelidad de la imagen reconstruida frente a la original se cuantifica mediante la métrica de **Pico de Relación Señal-Ruido (PSNR, *Peak Signal-to-Noise Ratio*)**, expresada en decibelios (dB). Valores más altos de PSNR indican menor distorsión y mayor proximidad matemática con la señal original. El experimento práctico evidencia la disminución gradual del PSNR conforme aumenta la agresividad de la limiarización, lo que permite evaluar numéricamente el umbral óptimo para el equilibrio entre compresión y degradación visual.

```
%%writefile tmp/fig_05_dwt_reconstrucao.cpp
#define MM_OUT "tmp/fig_05_dwt_reconstrucao.png"
// Compile: g++ -std=c++17 -O2 snippet.cpp -o snippet $(pkg-config --cflags --libs opencv4)
#include <opencv2/opencv.hpp>
#include <string>
#include <vector>
#include <cmath>
#include <iostream>
#include "morph.hpp"

//| label: fig-05-dwt-reconstrucao
//| fig-cap: "Reconstrução *wavelet* com limiarização de coeficientes (*hard thresholding*): à
medida que o limiar aumenta, mais detalhes são zerados, produzindo imagens progressivamente
mais suaves. Métrica PSNR quantifica a perda de qualidade sobre o padrão sintético."
//| echo: true
//| output: true

// Garante que img_gray utilize o mesmo padrão sintético claro
// (Função auxiliar para gerar o padrão sintético)
cv::Mat gerar_imagem_sintetica(int N = 256) {
    cv::Mat img = cv::Mat::zeros(N, N, CV_32F);
    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;
            double 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<float>(y, x) = static_cast<float>(std::max(0.0, std::min(v, 255.0)));
        }
    }
    cv::Mat out;
    img.convertTo(out, CV_8U);
    return out;
```

```
}

cv::Mat dwt_threshold_reconstruct(const cv::Mat& img, const std::string& wavelet = "db4", int
nivel = 2, double threshold = 0.0) {
    // Decompõe, aplica limiar e reconstrói via IDWT.
    cv::Mat img64;
    img.convertTo(img64, CV_64F);
    mm::WaveDec2 c = mm::wavedec2(img64, wavelet, nivel);

    // Copia e aplica hard thresholding em todos os detalhes
    // LL final não é limiarizado
    std::vector<cv::Mat> detail_t;
    // Reconstruir a estrutura de coeficientes
    // c_rec será construído no formato esperado por waverec2
    mm::WaveDec2 c_rec = c; // copia inicial

    // Aplicar limiar aos detalhes (nível mais grosseiro ao mais fino)
    for (int j = 0; j < nivel; ++j) {
        int idx = nivel - 1 - j; // ordem: from coarse to fine
        auto& detalhe = c.detail[idx];
        for (int k = 0; k < 3; ++k) {
            c_rec.detail[idx][k] = mm::wave_threshold(detalhe[k], threshold, "hard");
        }
    }

    cv::Mat rec = mm::waverec2(c_rec, wavelet);

    // Recorte para dimensão original
    cv::Mat rec_crop = rec(cv::Rect(0, 0, img.cols, img.rows)).clone();

    // Clip e converter para uint8
    cv::Mat rec8;
    cv::normalize(rec_crop, rec_crop, 0, 255, cv::NORM_MINMAX);
    rec_crop.convertTo(rec8, CV_8U);
    return rec8;
}

int main() {
    cv::Mat img_gray = gerar_imagem_sintetica(256);

    std::vector<int> thresholds = {0, 10, 30, 60, 100};
    std::vector<cv::Mat> imgs_thr = {img_gray};
    std::vector<std::string> titles_thr = {"Original"};

    for (int t : thresholds) {
        cv::Mat rec = dwt_threshold_reconstruct(img_gray, "db4", 2, static_cast<double>(t));
        double psnr = cv::PSNR(img_gray, rec);
        imgs_thr.push_back(rec);
        titles_thr.push_back("T=" + std::to_string(t) + "  PSNR=" +
std::to_string(psnr).substr(0, 5) + " dB");
    }

    // Convert vector<cv::Mat> to vector<mm::Image> for display
    std::vector<mm::Image> imgs_display;
    for (const auto& m : imgs_thr) {
        imgs_display.push_back(mm::Image(m));
    }
    mm::show(imgs_display, MM_OUT, titles_thr, 3);

    return 0;
}
```

```
Overwriting tmp/fig_05_dwt_reconstrucao.cpp
```

```
!g++ -I. -std=c++17 -DMM_USE_OPENCV -I/usr/include/opencv4 tmp/fig_05_dwt_reconstrucao.cpp -o
tmp/fig_05_dwt_reconstrucao -lopencv_stitching -lopencv_alphamat -lopencv_aruco
-lopencv_barcode -lopencv_bgsegm -lopencv_bioinspired -lopencv_ccalib -lopencv_dnn_objdetect
-lopencv_dnn_superres -lopencv_dpm -lopencv_face -lopencv_freetype -lopencv_fuzzy -lopencv_hdf
-lopencv_hfs -lopencv_img_hash -lopencv_intensity_transform -lopencv_line_descriptor
-lopencv_mcc -lopencv_quality -lopencv_rapid -lopencv_reg -lopencv_rgbd -lopencv_saliency
-lopencv_shape -lopencv_stereo -lopencv_structured_light -lopencv_phase_unwrapping
-lopencv_superres -lopencv_optflow -lopencv_surface_matching -lopencv_tracking
-lopencv_highgui -lopencv_datasets -lopencv_text -lopencv_plot -lopencv_ml -lopencv_videostab
-lopencv_videoio -lopencv_viz -lopencv_wechat_qrcode -lopencv_ximgproc -lopencv_video
-lopencv_xobjdetect -lopencv_objdetect -lopencv_calib3d -lopencv_imgcodecs -lopencv_features2d
-lopencv_dnn -lopencv_flann -lopencv_xphoto -lopencv_photo -lopencv_imgproc -lopencv_core \
  && ./tmp/fig_05_dwt_reconstrucao \
  && test -f "tmp/fig_05_dwt_reconstrucao.png" \
  || echo "  mm::show não gravou tmp/fig_05_dwt_reconstrucao.png"
```

```
[1] Original
[2] T=0  PSNR=19.54 dB
[3] T=10  PSNR=19.57 dB
[4] T=30  PSNR=22.94 dB
[5] T=60  PSNR=21.42 dB
[6] T=100  PSNR=18.46 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)")
```

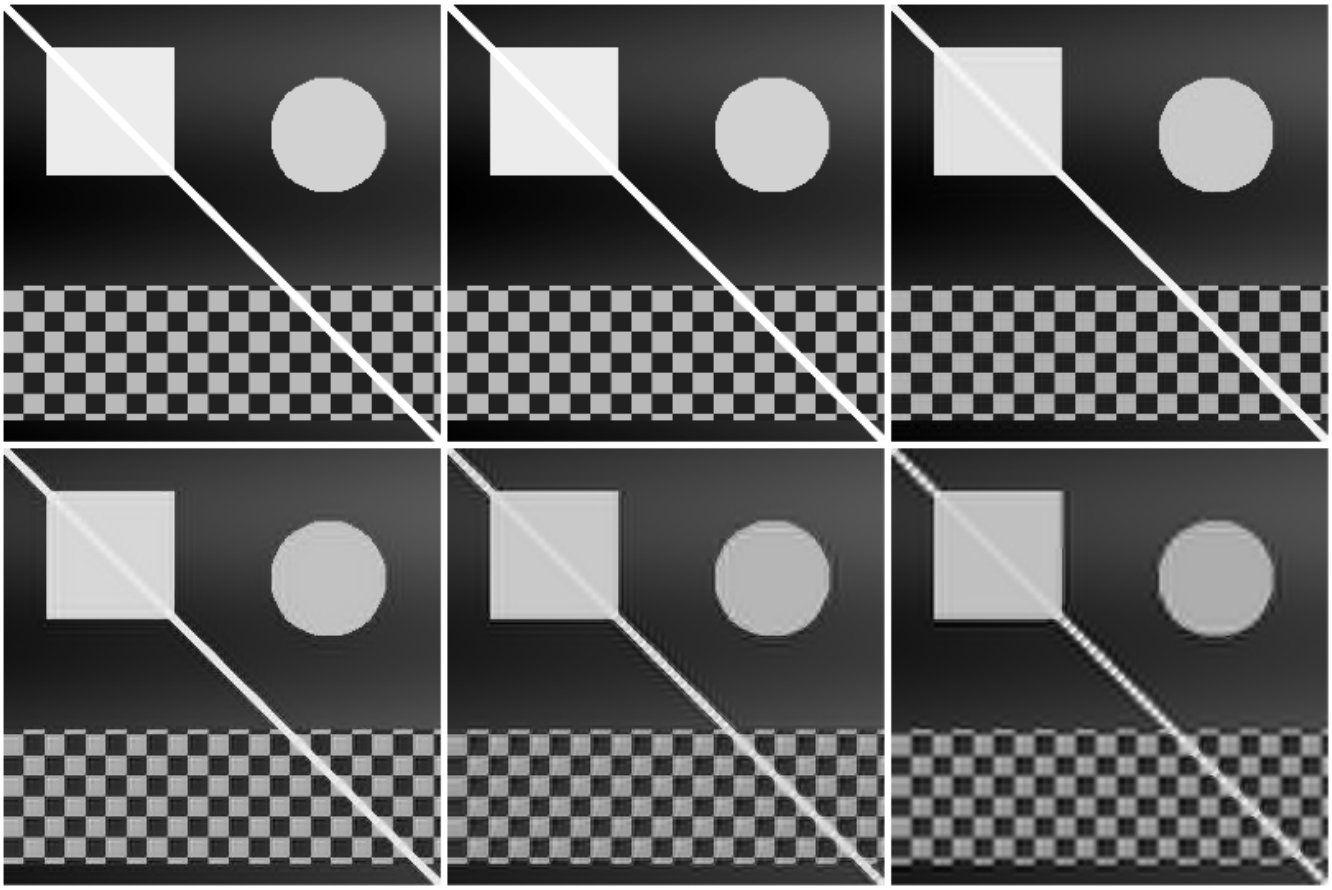


Figura 5.22: Reconstrução *wavelet* com limiarização de coeficientes (*hard thresholding*): à medida que o limiar aumenta, mais detalhes são zerados, produzindo imagens progressivamente mais suaves. Métrica PSNR quantifica a perda de qualidade sobre o padrão sintético.

Síntesis — Fourier vs. *Wavelets*: ¿cuándo utilizar cada enfoque?

La Tabla 5.5 sintetiza las principales diferencias estructurales y operativas entre la Transformada Discreta de Fourier (DFT) y la Transformada *Wavelet* Discreta (DWT).

Tabla 5.5: Comparación entre la Transformada Discreta de Fourier (DFT) y la Transformada *Wavelet* Discreta (DWT), destacando sus principales características y aplicaciones.

Criterio	Fourier (DFT)	*Wavelet* (DWT)
Funciones de base	Senoides de soporte infinito	Funciones de soporte compacto
Localización espacial	No explícita (global)	Explícita (local)
Filtrado espectral	Excelente para el control fino de frecuencias	Basado en subbandas (escalas)
Compresión de imágenes	Base de la DCT (JPEG tradicional)	Base de la DWT (JPEG 2000)
Análisis multiescala	No	Sí
Eliminación de ruido periódico	Altamente eficiente	Poco indicada
Señales no estacionarias	Limitada	Altamente eficiente

En términos prácticos, la DFT se consolida como la herramienta ideal para el análisis espectral puro, el diseño de filtros selectivos en el dominio de la frecuencia y la atenuación de ruidos periódicos y armónicos. Por otro lado, la DWT sobresale en escenarios que exigen la preservación rigurosa de la localización espacial

de las características asociada a su contenido frecuencial, destacándose en la compresión de datos, el análisis multirresolución y el procesamiento de transiciones abruptas. De este modo, ambas transformadas deben entenderse como técnicas perfectamente complementarias, que mapean caminos distintos y específicos para la resolución de problemas en PDI-VC.

i Analogías con Audio: Limitaciones y Precauciones

Al establecer analogías entre el procesamiento de imágenes y el audio, es importante considerar las diferencias fundamentales:

- En los sistemas de audio estéreo/multicanal, la fase entre canales es crucial para la percepción de la localización espacial (diferencias interaurales de fase y tiempo).
- En los sistemas monoaurales, la fase tiene una influencia perceptual limitada: el oído humano es relativamente insensible a la fase absoluta de componentes sinusoidales aislados.
- En las imágenes, la fase de la DFT siempre es fundamental para la localización espacial de las estructuras, independientemente de que se trate de una imagen monocromática o en color.

La analogía entre la fase en audio y la fase en imágenes debe emplearse con cautela, destacando que, aunque ambas transportan información sobre la organización espacial/temporal de la señal, los mecanismos perceptuales son fundamentalmente diferentes.

5.7 Compresión de Imágenes

Mientras que las *wavelets* establecen la base teórica del estándar JPEG 2000, el estándar JPEG tradicional se basa en la **Transformada Discreta de Cosenos (DCT, *Discrete Cosine Transform*)**. A pesar de las diferencias estructurales, ambos enfoques comparten el mismo principio fundamental: compactar la energía de la imagen en un número reducido de coeficientes y descartar los componentes de menor relevancia con un impacto visual mínimo.

El objetivo central de la compresión es reducir el volumen de datos necesario para el almacenamiento o transmisión de una imagen. Este proceso es posible gracias a la identificación y eliminación de **redundancias** estructurales y perceptuales.

5.7.1 Taxonomía de las Redundancias

El desarrollo de algoritmos de compresión se fundamenta en la identificación y eliminación de tres categorías principales de redundancia, sintetizadas en la Tabla 5.6.

Tabla 5.6: Categorías de redundancia en imágenes digitales y sus respectivos mecanismos de explotación.

Tipo	Definición	Enfoque de Explotación
Espacial (interpíxel)	Alta correlación y dependencia estadística entre píxeles vecinos.	DCT, DWT y codificación predictiva.
Espectral (intercanal)	Correlación estadística entre los canales de color de una misma imagen.	Transformaciones de espacio de color (ej: RGB a YC_bC_r).
Psicovisual	Insensibilidad del sistema visual humano (SVH) a variaciones de alta frecuencia y bajo contraste.	Procesos de cuantización selectiva de coeficientes.

Dependiendo de la preservación de la información original tras el proceso de decodificación, los métodos de compresión se dividen en dos clases fundamentales:

- **Sin pérdida (*lossless*):** Garantiza una reconstrucción bit a bit idéntica a la imagen original. Se emplea en escenarios donde la integridad de los datos es estrictamente crítica, como en imágenes médicas, diagnósticos por imagen y almacenamiento de documentos textuales.

- **Con pérdida (*lossy*):** Admite la introducción de una distorsión controlada en la señal a cambio de tasas de compresión sustancialmente más elevadas. Es el enfoque estándar para fotografías de consumo y *streaming* de vídeo, ecosistemas en los cuales el SVH tolera pequeñas atenuaciones de alta frecuencia sin percepción de degradación de la calidad visual.

5.7.2 Transformada de Cosenos Discreta (DCT-II 2D)

La **Transformada de Cosenos Discreta** (DCT) constituye la operación central del estándar JPEG. A diferencia de la DFT, que utiliza una base compleja, la DCT se basa en funciones trigonométricas puramente reales. Para un bloque de imagen $f(x, y)$ de dimensiones $N \times N$, la **DCT-II 2D** mapea la señal espacial al dominio de las frecuencias espaciales, generando la matriz de coeficientes $C(u, v)$ mediante:

$$C(u,v) = \alpha(u)\,\alpha(v) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x,y)\, \cos\left[\frac{\pi(2x+1)u}{2N}\right] \cos\left[\frac{\pi(2y+1)v}{2N}\right] \tag{5.9}$$

donde los factores de normalización ortogonal están dados por $\alpha(0) = \sqrt{1/N}$ y $\alpha(k) = \sqrt{2/N}$ para $k > 0$.

Cada coeficiente $C(u, v)$ cuantifica la contribución —o "peso"— de una frecuencia espacial específica dentro de ese bloque. El término $C(0, 0)$ se denomina **componente DC** y representa la intensidad media del bloque (frecuencia nula). Los demás coeficientes, llamados **componentes AC** (*Alternating Current*), corresponden a las frecuencias espaciales progresivamente mayores.

5.7.3 Las Funciones de Base de la DCT

Desde una perspectiva geométrica, la Ecuación 5.9 realiza la proyección del bloque de píxeles sobre un conjunto de funciones ortogonales. Para el caso estándar de JPEG ($N = 8$), el bloque espacial se descompone en una combinación lineal de **64 funciones de base** bidimensionales, denotadas por $B_{u,v}(x, y)$ y generadas por el producto de funciones cosenoidales:

$$B_{u,v}(x,y) = \cos\left[\frac{\pi(2x+1)u}{16}\right] \cos\left[\frac{\pi(2y+1)v}{16}\right]$$

De esta manera, la operación inversa puede interpretarse como la reconstrucción exacta del bloque original mediante la suma ponderada de estas 64 matrices de base, donde cada coeficiente $C(u, v)$ actúa como el peso analítico de su respectiva componente armónica.

La **frecuencia espacial** indicada por los índices (u, v) determina el número de ciclos de oscilación a lo largo de las dimensiones horizontales y verticales del bloque. Como se ilustra en la Figura 5.23 —cuyo código aísla cada base aplicando la transformación inversa sobre impulsos unitarios—, estas 64 funciones se organizan en una matriz 8×8. La esquina superior izquierda ($u = 0, v = 0$) muestra el patrón uniforme de frecuencia nula (DC), mientras que el avance hacia la derecha (eje u) o hacia abajo (eje v) mapea variaciones armónicas progresivamente mayores, representando transiciones rápidas, bordes y texturas en las orientaciones horizontales, verticales y diagonales.

> **i** DCT vs DFT: Ventaja de la Compactación de Energía
>
> Tanto la DCT como la DFT mapean un bloque espacial $N \times N$ en una matriz de coeficientes de la misma dimensión. Sin embargo, para imágenes naturales, la DCT presenta una mayor eficiencia en la **compactación de energía** en las bajas frecuencias. Esto ocurre porque la DCT asume implícitamente una simetría par de la señal en las fronteras del bloque, lo que equivale a una extensión periódica continua, minimizando el efecto de dispersión espectral (*ringing*). Como consecuencia, la mayoría de los coeficientes AC decae rápidamente hacia valores cercanos a cero, optimizando el *pipeline* de compresión sin introducir degradación visual perceptible.

```
%%writefile tmp/fig_05_dct_basis.cpp
#define MM_OUT "tmp/fig_05_dct_basis.png"
//| label: fig-05-dct-basis
//| fig-cap: "O Alfabeto Visual do JPEG: As 64 funções de base da DCT-II. O coeficiente DC
fica no topo esquerdo (suave). Ao descer e avançar à direita, a oscilação espacial aumenta
drasticamente."
//| echo: true
//| output: true

#include <opencv2/opencv.hpp>
#include <vector>
#include <string>
#include "morph.hpp"
#include <filesystem>

int main() {
    // Cada base é a IDCT de um único coeficiente unitário - montadas num mosaico 8×8.
    int tile = 32;
    cv::Mat montagem = cv::Mat::zeros(8 * tile, 8 * tile, CV_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);
            cv::Mat base_norm;
            cv::normalize(base, base_norm, 0, 255, cv::NORM_MINMAX);
            base_norm.convertTo(base_norm, CV_8U);
            cv::Mat base_resized;
            cv::resize(base_norm, base_resized, cv::Size(tile, tile), 0, 0,
cv::INTER_NEAREST);

            cv::Rect roi(j * tile, i * tile, tile, tile);
            base_resized.copyTo(montagem(roi));
        }
    }

    mm::Image montagem_img(montagem);
    mm::show(std::vector<mm::Image>{montagem_img}, MM_OUT,
             std::vector<std::string>{"As 64 bases da DCT-II 8×8 (DC no topo-esquerdo)"}, 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)")
```

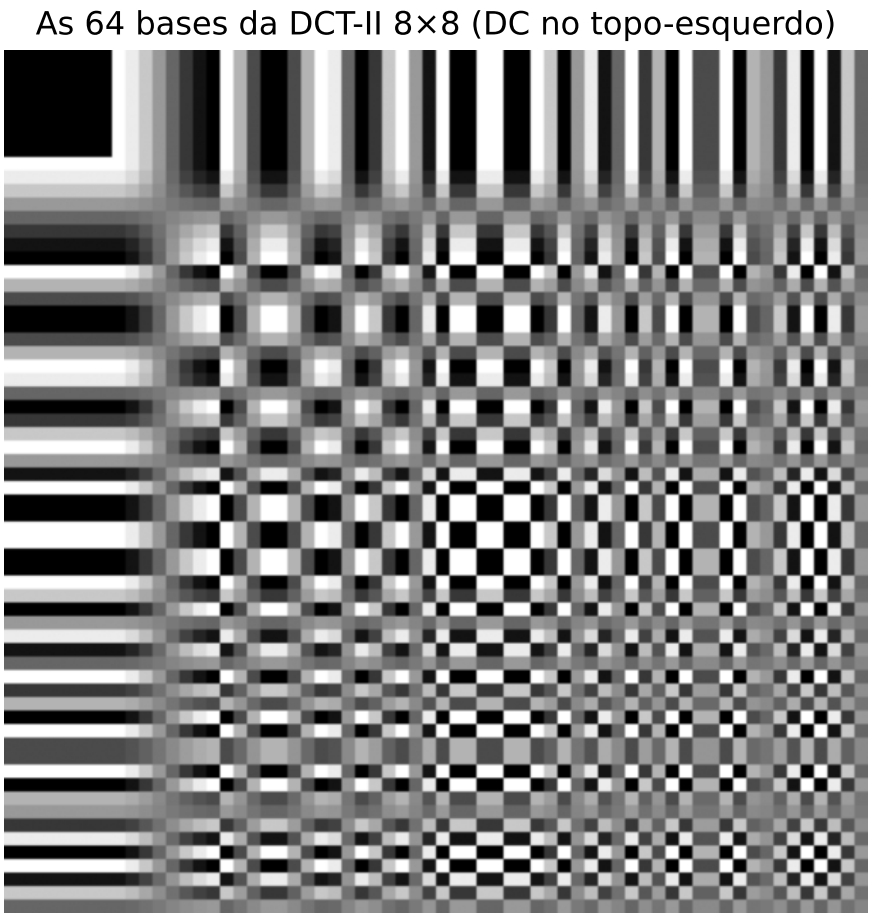


Figura 5.23: O Alfabeto Visual do JPEG: As 64 funções de base da DCT-II. O coeficiente DC fica no topo esquerdo (suave). Ao descer e avançar à direita, a oscilación espacial aumenta drasticamente.

5.7.4 Concentración de Energía y Reconstrucción Progresiva

Antes de la aplicación de la DCT, los píxeles del bloque de intensidad se trasladan rutinariamente (restando 128 para imágenes de 8 bits) con el fin de centrar la señal en torno a cero, eliminando componentes continuas innecesarias. Al calcular la DCT sobre el bloque resultante, la propiedad de **compactación de energía** se hace evidente: casi la totalidad de la varianza y de la información de la imagen original se concentra en el coeficiente DC ($C(0,0)$) y en los primeros armónicos AC de baja frecuencia.

La Figura 5.24 demuestra este fenómeno mediante una reconstrucción progresiva por truncamiento abrupto. En lugar de utilizar los 64 coeficientes, el algoritmo preserva únicamente los k primeros componentes —seleccionados con base en un barrido que prioriza las bajas frecuencias espaciales— y anula los restantes.

La síntesis inversa (**IDCT**) realizada con solo una fracción de los coeficientes (como 15% o 30%) ya es capaz de recuperar las estructuras y la iluminación macro del bloque original de píxeles. A medida que los armónicos de frecuencias más altas se reincorporan progresivamente, los detalles finos y las transiciones rápidas se restauran. Este comportamiento valida el principio de la compresión perceptual: las altas frecuencias descartadas poseen poca energía y su ausencia, en condiciones normales, genera un impacto visual secundario en la percepción del observador.

```
%%writefile tmp/fig_05_dct_bloco.cpp
#define MM_OUT "tmp/fig_05_dct_bloco.png"
#include <opencv2/opencv.hpp>
#include <vector>
#include <string>
#include <cmath>
```

```cpp
#include <iostream>
#include <algorithm>
#include "morph.hpp"

//| label: fig-05-dct-bloco
//| fig-cap: "DCT 2D em bloco 8×8: coeficientes e reconstrução progressiva."
//| echo: true
//| output: true

// DCT-II 2D ortogonal (separável) - usar mm::dct2 do morph.hpp
// IDCT-II 2D ortogonal - usar mm::idct2 do 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]

    //   Bloco 8×8 centralizado da imagem
    int cy = img_gray.h / 2;
    int cx = img_gray.w / 2;

    // Extrair bloco 8×8
    cv::Mat img_mat = img_gray;
    cv::Mat bloco_mat = img_mat(cv::Rect(cx, cy, 8, 8)).clone();

    // Converter para float64 e subtrair 128
    cv::Mat bloco_f;
    bloco_mat.convertTo(bloco_f, CV_64F);
    bloco_f -= 128.0;

    // Aplicar DCT2
    cv::Mat C = mm::dct2(bloco_f);

    std::cout << "Coeficientes DCT do bloco 8×8:" << std::endl;

    // Arredondar e imprimir
    cv::Mat C_rounded;
    cv::Mat C_abs, C_int;
    cv::normalize(C, C_abs, 0, 255, cv::NORM_MINMAX);
    C.convertTo(C_int, CV_32S, 1.0, 0.5); // arredondamento via conversão
    std::cout << C_int << std::endl;

    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);
        }
    }

    std::cout << "\nEnergia DC     : " << dc_energy << std::endl;
    std::cout << "Energia total  : " << total_energy << std::endl;
    std::cout << "Fração no DC   : " << (dc_energy / total_energy) * 100.0 << "% ←
concentração de energia" << std::endl;

    //   Reconstrução progressiva
    std::vector<mm::Image> imgs_rec;
    std::vector<std::string> titles_rec;

    // Bloco original
    cv::Mat bloco_original;
```

```
    bloco_mat.convertTo(bloco_original, CV_8U);
    imgs_rec.push_back(mm::Image(bloco_original));
    titles_rec.push_back("Bloco original\n(8×8 pixels)");

    // Diferentes quantidades de coeficientes
    int keeps[] = {1, 4, 10, 20, 40, 64};

    for (int keep : keeps) {
        cv::Mat C_trunc = cv::Mat::zeros(8, 8, CV_64F);

        // Zerar coeficientes mantendo apenas os "keep" primeiros por ordem zigzag
        // (ordem por soma u+v, que é uma aproximação da ordem zigzag)
        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});
            }
        }

        // Ordenar por soma u+v (frequência crescente)
        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 k = 0; k < keep && k < 64; k++) {
            int u = indices[k].first;
            int v = indices[k].second;
            C_trunc.at<double>(u, v) = C.at<double>(u, v);
        }

        // Reconstruir
        cv::Mat rec_f = mm::idct2(C_trunc);
        rec_f += 128.0;

        // Recortar para [0, 255] e converter para uint8
        cv::Mat rec;
        cv::threshold(rec_f, rec_f, 255, 255, cv::THRESH_TRUNC);
        cv::threshold(rec_f, rec_f, 0, 0, cv::THRESH_TOZERO);
        rec_f.convertTo(rec, CV_8U);

        imgs_rec.push_back(mm::Image(rec));

        char buf[64];
        snprintf(buf, sizeof(buf), "%d coef.\n(%.0f%% do total)",
                 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"
```

```
Coeficientes DCT do bloco 8×8:
[192, -47, 1, 9, 0, 0, 0, 0;
 -95, 55, 7, -10, 0, 1, 1, 0;
 14, -14, 9, 1, 1, 0, 1, 0;
 0, 1, -11, 1, 1, 1, 1, 0;
 10, -10, 0, 1, 1, 0, 1, 0;
 0, 1, 0, 1, 0, 0, 1, 1;
 0, 1, 0, 0, 0, 1, 1, 0;
 1, 0, 0, 0, 1, 1, 1, 1]

Energia DC     : 36816
Energia total  : 52253
Fração no DC   : 70.4572% ← concentração de energia
[1] Bloco original
(8×8 pixels)
[2] 1 coef.
(2% do total)
[3] 4 coef.
(6% do total)
[4] 10 coef.
(16% do total)
[5] 20 coef.
(31% do total)
[6] 40 coef.
(62% do total)
[7] 64 coef.
(100% do total)
```

```
try:
    mm.show(mm.read("tmp/fig_05_dct_bloco.png"), figsize=(12, 7))
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + " tmp/fig_05_dct_bloco.png
    (ver a versao Python)")
```

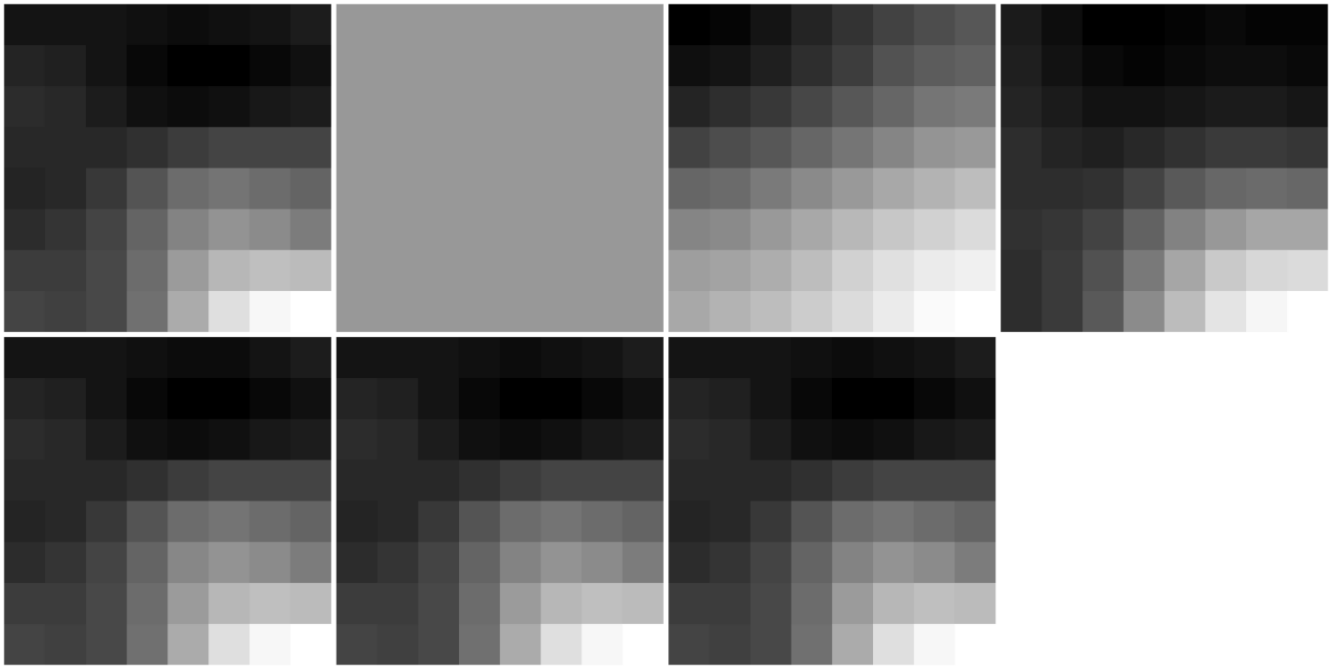


Figura 5.24: DCT 2D em bloco 8×8: coeficientes e reconstrução progressiva.

5.7.5 El *Pipeline* de Compresión JPEG

El estándar JPEG opera dividiendo la imagen en bloques disjuntos de 8×8 píxeles, procesados mediante una secuencia de transformaciones espaciales, perceptuales y estadísticas. El *pipeline* completo de codificación se estructura en seis etapas principales:

$$\text{RGB} \xrightarrow{\text{(1) } YC_bC_r} \xrightarrow{\text{(2) Submuestreo}} \xrightarrow{\text{(3) Bloques 8×8}} \xrightarrow{\text{(4) DCT}} \xrightarrow{\text{(5) Cuantificación}} \xrightarrow{\text{(6) Codificación Entrópica}}$$

La Tabla 5.7 detalla la función analítica y el fundamento perceptual que justifica cada una de estas etapas.

Tabla 5.7: Etapas del *pipeline* de compresión JPEG y sus respectivos fundamentos de diseño.

Etapa	Operación	Fundamento Perceptual y Estadístico
1	Conversión $RGB \to YC_bC_r$	Separa la luminancia (Y) de la crominancia (C_b, C_r). El sistema visual humano (SVH) presenta mayor sensibilidad a variaciones de brillo que de color.
2	Submuestreo de crominancia (ej: 4:2:0)	Reduce la resolución espacial de los canales de color a la mitad, descartando datos redundantes con impacto visual despreciable.
3–4	Centralización y aplicación de la DCT 8×8	Traslada los píxeles al intervalo $[-128, 127]$ y compacta la energía espectral del bloque en los coeficientes de baja frecuencia.

Etapa	Operación	Fundamento Perceptual y Estadístico
5	Cuantificación lineal selectiva	Divide cada coeficiente $C(u,v)$ por el elemento correspondiente de la matriz $Q(u,v)$, aplicando redondeo entero. Constituye la principal fuente de compresión con pérdida.
6	Barrido en zigzag y codificación	Ordena los coeficientes cuantificados para maximizar secuencias nulas consecutivas, optimizando la codificación por longitud de corrida (RLE) y la codificación de Huffman.

La **matriz de cuantificación** $Q(u,v)$ es el mecanismo central de control del compromiso entre tasa de compresión y calidad visual. En el algoritmo práctico de la Figura 5.25, el factor de calidad estipulado por el usuario (escala de 1 a 100) se convierte en un escalar que parametriza la severidad de la matriz Q. Valores reducidos de calidad expanden los divisores de $Q(u,v)$, forzando el truncamiento masivo de los coeficientes AC a cero. Cuando esta eliminación es excesiva, la discontinuidad en las fronteras de los bloques adyacentes no se atenúa en la reconstrucción, generando los denominados **artefactos de bloque** (*blocking artifacts*).

La Lógica del Barrido en Zigzag

La eficiencia del codificador entrópico posterior a la cuantificación depende directamente de la ordenación de los datos. Como la DCT concentra la energía vital en el vértice superior izquierdo de la matriz (bajas frecuencias) y empuja los coeficientes nulos hacia las extremidades opuestas, la lectura lineal por filas o columnas fragmentaría las secuencias de ceros.

La ordenación en zigzag soluciona esta limitación al recorrer la matriz diagonalmente en orden creciente de frecuencia espacial. Este mapeo agrupa los coeficientes significativos al inicio del vector y concentra los coeficientes nulos en una única secuencia continua al final del arreglo, permitiendo que el algoritmo RLE codifique grandes bloques de datos de forma compacta y eficiente.

> **i** ¿Qué es RLE?
>
> **RLE** (*Run-Length Encoding*) es una técnica de compresión sin pérdidas que codifica secuencias consecutivas de valores idénticos — especialmente **ceros** — como un par (recuento, valor). En JPEG, después del barrido en zigzag, los coeficientes cuantificados se organizan de modo que los ceros se concentren al final del vector. El RLE entonces comprime esa larga corrida de ceros con extrema eficiencia, optimizando el almacenamiento y la transmisión de la imagen comprimida.

```
%%writefile tmp/fig_05_jpeg_pipeline.cpp
#define MM_OUT "tmp/fig_05_jpeg_pipeline.png"
#include <opencv2/opencv.hpp>
#include <string>
#include <vector>
#include <cstdio>
#include "morph.hpp"

//| 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

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_mat = cv::imread("imagens/cameraman.png", cv::IMREAD_GRAYSCALE);
    if (img_src_mat.empty()) {
        return 1;
    }
    cv::Mat img_resized;
    cv::resize(img_src_mat, img_resized, cv::Size(256, 256));
    mm::Image img_src = mm::gray(img_resized);

    std::vector<mm::Image> imgs;
    imgs.push_back(img_src);
    std::vector<std::string> titles;
    titles.push_back("Original (Cameraman)");

    // Store PSNR values before pushing to titles
    std::vector<double> psnr_vals;
    for (int q : {10, 25, 50, 75, 90}) {
        mm::Image rec = mm::jpegCompress(img_src, q);
        imgs.push_back(rec);
        double psnr_val = mm::psnr(img_src, rec);
        psnr_vals.push_back(psnr_val);
        char title[128];
        std::snprintf(title, sizeof(title), "Q=%d (PSNR=%.1f dB)", q, psnr_val);
        titles.push_back(std::string(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 Simulador Interactivo: Cuantización DCT

El simulador de la Figura 5.26 permite explorar el impacto del proceso de cuantización sobre un bloque 8×8 extraído de una imagen real, sintetizando en tiempo real los siguientes componentes:

- **Bloque original y reconstruido:** Representación directa de los píxeles en el dominio espacial en escala de grises [0, 255].
- **Coeficientes DCT:** Distribución de la energía mapeada de forma logarítmica en un gradiente cromático, evidenciando la concentración de intensidad en el vértice superior izquierdo (bajas frecuencias).
- **Coeficientes cuantizados:** Exhibición de los valores enteros resultantes de la división por la matriz $Q(u, v)$, haciendo visualmente explícita la aparición masiva de coeficientes nulos (en tonos oscuros) conforme el factor de calidad se reduce.

- **Métricas de compresión:** Panel de monitoreo que cuantifica el Error Cuadrático Medio (MSE), el número de coeficientes preservados y el volumen de ceros generados para la codificación entrópica.

Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.es/cap05/sim-05-dct.html

Figura 5.26: Simulador interactivo de compresión DCT-JPEG: ajuste el factor de calidad y visualice en tiempo real los coeficientes anulados, el bloque reconstruido y el error de cuantización.

5.8 Comparación de Formatos de Imagen

La elección de un formato de almacenamiento digital impacta directamente en el compromiso entre calidad visual, tamaño de archivo y costo computacional de decodificación. Los tres formatos de mayor relevancia para arquitecturas *web* y sistemas de computación visual son JPEG, PNG y WebP.

5.8.1 Características de los Formatos

La Tabla 5.8 sintetiza las propiedades estructurales de los principales formatos de imagen rasterizados.

Tabla 5.8: Comparación estructural entre los principales formatos de imagen rasterizados.

Característica	JPEG	PNG	WebP
Compresión	Con pérdida	Sin pérdida	Con y sin pérdida.
Transparencia (canal alfa)	No	Sí	Sí.
Soporte de animación	No	Limitado (APNG)	Sí.
Algoritmo base	DCT + Huffman	DEFLATE (LZ77 + Huffman)	VP8 / VP8L.
Mejor para	Fotografía	Gráficos, texto e iconos	Uso universal en entorno Web.
Peor para	Texto y bordes nítidos	Imágenes fotográficas complejas	Compatibilidad heredada.

5.8.2 Métricas de Evaluación de Calidad

Dos métricas objetivas son ampliamente adoptadas para cuantificar la distorsión introducida por procesos de compresión:

Pico de la Relación Señal-Ruido (PSNR, *Peak Signal-to-Noise Ratio*):

$$\text{PSNR} = 10 \log_{10}\left(\frac{L^2}{\text{MSE}}\right) \quad [\text{dB}] \tag{5.10}$$

donde $L = 255$ para imágenes cuantizadas en 8 bits y MSE representa el **Error Cuadrático Medio** (*Mean Squared Error*). Valores de PSNR superiores a 40 dB indican excelente fidelidad; entre 30 dB y 40 dB representan buena calidad; y valores inferiores a 30 dB corresponden a degradaciones visuales fácilmente perceptibles.

Índice de Similitud Estructural (SSIM, *Structural Similarity Index*):

$$\text{SSIM}(f, g) = \frac{(2\mu_f\mu_g + c_1)(2\sigma_{fg} + c_2)}{(\mu_f^2 + \mu_g^2 + c_1)(\sigma_f^2 + \sigma_g^2 + c_2)} \tag{5.11}$$

El SSIM evalúa ventanas locales de la imagen basándose en tres componentes complementarios: **luminancia** (μ_f, μ_g), **contraste** (σ_f, σ_g) y **estructura** (σ_{fg}), ponderados por constantes de estabilidad c_1 y c_2. El índice varía en el intervalo $[-1, 1]$, donde la unidad representa la identidad perfecta. A diferencia del PSNR, el SSIM considera la organización espacial de los errores, alineándose con la percepción del sistema visual humano (SVH).

> **i** PSNR vs SSIM: Aplicación de Métricas Perceptuales
>
> El PSNR posee una formulación matemática simple y un bajo costo computacional; sin embargo, tiende a sobreestimar la calidad en imágenes con distorsiones localizadas o subestimarla en variaciones globales de brillo toleradas por el observador. El SSIM modela con mayor fidelidad la percepción biológica, pero exige un mayor esfuerzo de procesamiento. Para análisis rigurosos de codificadores, se recomienda reportar ambas métricas estadísticas en carácter complementario.

5.8.3 Inspección Visual: Naturaleza de los Artefactos de Compresión

La naturaleza matemática del codificador determina el tipo de degradación introducida en tasas de bits reducidas. Como se ilustra en la Figura 5.27, la compresión agresiva mediante DCT en el estándar JPEG segmenta la imagen en mallas rígidas, generando los **artefactos de bloque** (*blocking artifacts*). En contrapartida, los algoritmos basados en codificación predictiva o representaciones sometidas a transformadas espaciales avanzadas (como WebP y JPEG 2000) eliminan las discontinuidades de bloque, pero introducen pérdida de textura fina y desenfoques característicos alrededor de bordes de alto contraste.

```
%%writefile tmp/fig_05_zoom_artefatos.cpp
#define MM_OUT "tmp/fig_05_zoom_artefatos.png"
//| label: fig-05-zoom-artefatos
//| fig-cap: "Análise comparativa de artefatos de compressão sob fator de qualidade reduzido
($Q=10$). À esquerda, observa-se o artefato de bloco característico da discretização por DCT
no JPEG. À direita, evidencia-se o efeito de atenuação e suavização de bordas intrínseco ao
padrão WebP."
//| echo: true
//| output: true

#include <opencv2/opencv.hpp>
#include <vector>
#include <string>
#include "morph.hpp"
```

```cpp
#include <filesystem>

cv::Mat zoom(const cv::Mat& img) {
    // Recorta a região 120:200, 150:230 e redimensiona para 320x320 com interpolação mais
    próxima
    cv::Mat crop = img(cv::Rect(150, 120, 80, 80));
    cv::Mat resized;
    cv::resize(crop, resized, cv::Size(320, 320), 0, 0, cv::INTER_NEAREST);
    return resized;
}

int main() {
    // Lê a imagem, converte para escala de cinza e redimensiona para 256x256
    cv::Mat img_src_m = cv::imread("imagens/cameraman.png", cv::IMREAD_GRAYSCALE);
    cv::Mat img_src_resized;
    cv::resize(img_src_m, img_src_resized, cv::Size(256, 256));
    mm::Image img_src = img_src_resized;  // converte para mm::Image para usar mm::gray se
necessário, mas já é gray

    // Salva com qualidade JPEG 10 e WebP 10
    std::filesystem::create_directories("tmp");
    cv::imwrite("tmp/zoom_q10.jpg", img_src_resized, {cv::IMWRITE_JPEG_QUALITY, 10});
    cv::imwrite("tmp/zoom_q10.webp", img_src_resized, {cv::IMWRITE_WEBP_QUALITY, 10});

    // Aplica zoom na imagem original, JPEG e WebP
    cv::Mat z_orig = zoom(img_src_resized);
    cv::Mat z_jpeg = zoom(cv::imread("tmp/zoom_q10.jpg", cv::IMREAD_GRAYSCALE));
    cv::Mat z_webp = zoom(cv::imread("tmp/zoom_q10.webp", cv::IMREAD_GRAYSCALE));

    // Exibe as três imagens em uma grade
    mm::show(
        std::vector<mm::Image>{mm::Image(z_orig), mm::Image(z_jpeg), mm::Image(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)")
```

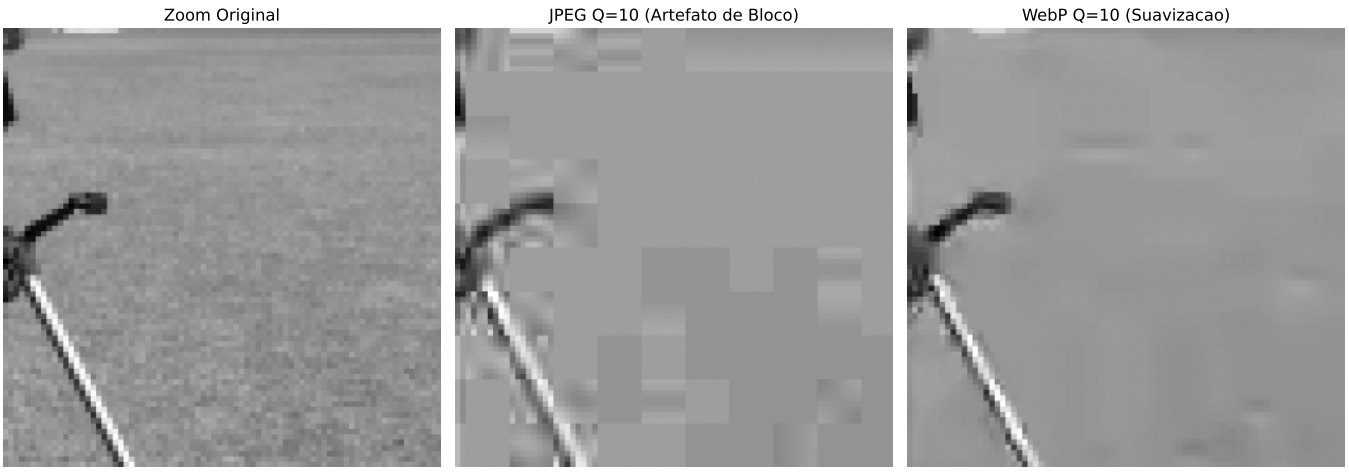


Figura 5.27: Análise comparativa de artefatos de compressão sob fator de qualidade reduzido ($Q = 10$). À esquerda, observa-se o artefato de bloco característico da discretização por DCT no JPEG. À direita, evidencia-se o efeito de atenuação e suavização de bordas intrínseco ao padrão WebP.

5.8.4 Evaluación Cuantitativa y Espacial de la Compresión

La validación de los algoritmos de compresión con pérdida exige un análisis que correlacione el costo de almacenamiento con la fidelidad de la señal reconstruida. Esta evaluación se realiza de manera complementaria mediante curvas de rendimiento global y mediante el mapeo local de las distorsiones inducidas por los codificadores.

5.8.4.1 Curvas de Tasa-Distorsión

La Figura 5.28 presenta la evaluación empírica del *pipeline* JPEG y WebP mediante **curvas de tasa-distorsión**, que monitorean la ganancia de compresión (tamaño del archivo en KB) en función del PSNR. El formato PNG actúa como línea de base ideal (PSNR $= \infty$), ya que su naturaleza *lossless* impide cualquier degradación, aunque requiere un volumen de datos sustancialmente mayor.

El análisis de las curvas demuestra la superioridad y la eficiencia del estándar WebP sobre el JPEG tradicional: para alcanzar un mismo nivel de fidelidad matemática (como el rango de calidad excelente, donde PSNR > 40 dB), el codificador WebP genera archivos significativamente más pequeños. Este comportamiento refleja el impacto práctico de la evolución de los algoritmos en la optimización de los sistemas de transmisión y almacenamiento digital.

```
%%writefile tmp/fig_05_formatos_comparacao.cpp
#define MM_OUT "tmp/fig_05_formatos_comparacao.png"
//| label: fig-05-formatos-comparacao
//| fig-cap: "Curva taxa-distorção: PSNR vs tamanho de arquivo para JPEG, WebP e PNG aplicada
à imagem do *Cameraman*."
//| echo: true
//| output: true

#include <opencv2/opencv.hpp>
#include <string>
#include <vector>
#include <filesystem>
#include <cstdio>
#include "morph.hpp"

int main() {
    // Criar diretório temporário se não existir
    std::filesystem::create_directories("tmp");

    // Carregar e redimensionar a imagem do Cameraman para 256x256
    cv::Mat img_full = cv::imread("imagens/cameraman.png", cv::IMREAD_GRAYSCALE);
    cv::Mat img_resized;
    cv::resize(img_full, img_resized, cv::Size(256, 256));
    mm::Image src = mm::gray(img_resized);

    // Loop para JPEG com diferentes qualidades
    std::vector<double> jpeg_kb, jpeg_psnr;
    std::vector<int> jpeg_qualities = {10, 20, 30, 40, 50, 60, 70, 80, 90, 95};
    for (int q : jpeg_qualities) {
        std::string p = "tmp/fmt_q" + std::to_string(q) + ".jpg";
        cv::Mat src_mat = src;  // converter mm::Image para cv::Mat
        std::vector<int> params = {cv::IMWRITE_JPEG_QUALITY, q};
        cv::imwrite(p, src_mat, params);
        cv::Mat rec = cv::imread(p, cv::IMREAD_GRAYSCALE);
        mm::Image rec_img = rec;
        jpeg_kb.push_back((double)std::filesystem::file_size(p) / 1024.0);
        jpeg_psnr.push_back(mm::psnr(src, rec_img));
    }

    // Loop para WebP com diferentes qualidades
```

```
    std::vector<double> webp_kb, webp_psnr;
    std::vector<int> webp_qualities = {30, 50, 70, 85, 95};
    for (int q : webp_qualities) {
        std::string p = "tmp/fmt_w" + std::to_string(q) + ".webp";
        cv::Mat src_mat = src;  // converter mm::Image para cv::Mat
        std::vector<int> params = {cv::IMWRITE_WEBP_QUALITY, q};
        cv::imwrite(p, src_mat, params);
        cv::Mat rec = cv::imread(p, cv::IMREAD_GRAYSCALE);
        mm::Image rec_img = rec;
        webp_kb.push_back((double)std::filesystem::file_size(p) / 1024.0);
        webp_psnr.push_back(mm::psnr(src, rec_img));
    }

    // PNG sem perda
    std::string p_png = "tmp/fmt.png";
    cv::Mat src_mat_png = src;  // converter mm::Image para cv::Mat
    std::vector<int> params_png = {cv::IMWRITE_PNG_COMPRESSION, 9};
    cv::imwrite(p_png, src_mat_png, params_png);
    double png_kb = (double)std::filesystem::file_size(p_png) / 1024.0;

    // Criar o gráfico de linha comparando JPEG e WebP
    std::string title = "Curva Taxa-Distorcao (PNG sem perda: " +
                        std::to_string(png_kb).substr(0, std::to_string(png_kb).find(".") + 2)
+
                        " KB)";

    // Preparar vetores como vetores de vetores (uma curva por elemento)
    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"};

    mm::Image chart = mm::lineChart(xs, ys, labels, {}, title,
                                    "Tamanho do arquivo (KB)", "PSNR (dB)");

    // Exibir o resultado
    std::vector<mm::Image> charts = {chart};
    std::vector<std::string> chart_titles = {"JPEG x WebP x PNG"};
    mm::show(charts, MM_OUT, chart_titles, 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)")
```

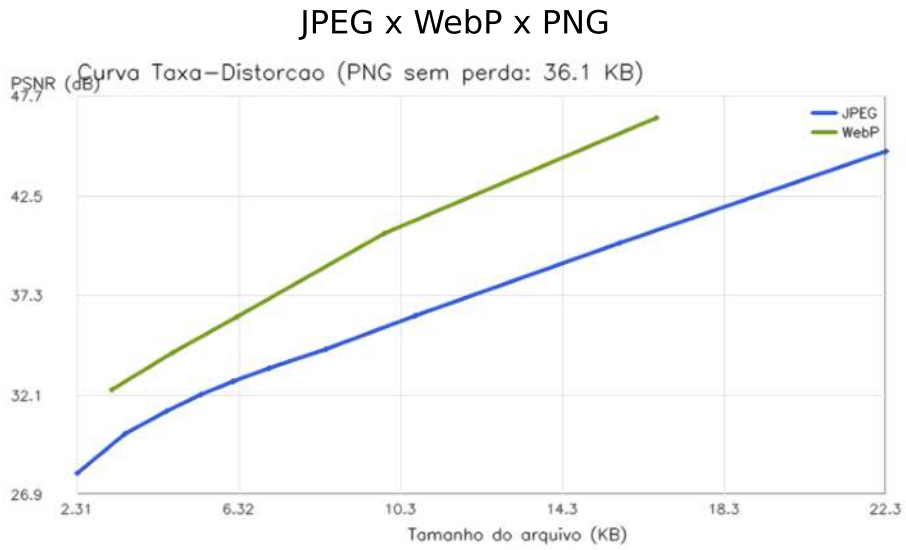


Figura 5.28: Curva taxa-distorção: PSNR vs tamanho de arquivo para JPEG, WebP e PNG aplicada à imagem do *Cameraman*.

Tamaño original de la imagen

La imagen *Cameraman* (256 × 256 píxeles en escala de grises) ocupa **64 KB** en formato bruto (sin compresión). Como referencia, el PNG *lossless* comprime ese volumen a **36,2 KB** — evidenciando que la compresión sin pérdidas ya reduce significativamente el almacenamiento para imágenes con regiones homogéneas. En contrapartida, los formatos con pérdida (JPEG y WebP) alcanzan tamaños aún menores: el JPEG con calidad 95 ocupa 22,3 KB (PSNR 45 dB), mientras que el WebP con calidad 90 alcanza 12,5 KB con PSNR equivalente, demostrando su superioridad en eficiencia de compresión.

5.8.4.2 Mapeo Espacial de Errores y Correlación Perceptual

Aunque el PSNR ofrece un indicativo numérico rápido, las métricas globales no logran discriminar cómo se distribuye geométricamente la pérdida de información sobre la imagen. La Figura 5.29 soluciona esta limitación al asociar las reconstrucciones en diferentes calidades con sus respectivos mapas de error absoluto y con el SSIM.

Los mapas residuales — obtenidos mediante la diferencia absoluta normalizada entre la imagen original y la comprimida — revelan la firma espacial intrínseca de cada arquitectura de codificación:

- **En calidades altas ($Q = 95$ a $Q = 75$):** Las distorsiones se concentran predominantemente alrededor de transiciones abruptas de intensidad (bordes), como resultado del reflejo espectral derivado del descarte de altas frecuencias. El índice SSIM permanece cercano a la unidad, atestiguando la integridad de las estructuras originales.
- **En calidades agresivas ($Q = 50$ a $Q = 25$):** El error adopta una estructura de malla ortogonal regularizada. Este patrón geométrico evidencia la aparición de los **artefactos de bloque** (*blocking artifacts*), indicando que la cuantización severa ha corrompido la correlación espacial entre bloques adyacentes de 8×8 píxeles.

El SSIM captura esta degradación morfológica de manera mucho más sensible que el PSNR, penalizando la puntuación final a medida que la organización estructural y las texturas finas — a las cuales el sistema visual humano es altamente receptivo — son eliminadas por el codificador.

```
%%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 <algorithm>
#include "morph.hpp"

// Implementación de SSIM (Wang et al.) con ventana gaussiana
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 mu1, mu2;
    cv::GaussianBlur(I1, mu1, cv::Size(11, 11), 1.5);
    cv::GaussianBlur(I2, mu2, cv::Size(11, 11), 1.5);
```

```
    cv::Mat mu1_sq = mu1.mul(mu1);
    cv::Mat mu2_sq = mu2.mul(mu2);
    cv::Mat mu1_mu2 = mu1.mul(mu2);

    cv::Mat sigma1_sq, sigma2_sq, sigma12;
    cv::GaussianBlur(I1.mul(I1), sigma1_sq, cv::Size(11, 11), 1.5);
    sigma1_sq -= mu1_sq;
    cv::GaussianBlur(I2.mul(I2), sigma2_sq, cv::Size(11, 11), 1.5);
    sigma2_sq -= mu2_sq;
    cv::GaussianBlur(I1.mul(I2), sigma12, cv::Size(11, 11), 1.5);
    sigma12 -= mu1_mu2;

    cv::Mat ssim_map;
    cv::Mat cs_map;
    cv::Mat temp1 = 2 * mu1_mu2 + C1;
    cv::Mat temp2 = 2 * sigma12 + C2;
    cv::Mat temp3 = mu1_sq + mu2_sq + C1;
    cv::Mat temp4 = sigma1_sq + sigma2_sq + C2;

    cv::divide(temp1.mul(temp2), temp3.mul(temp4), ssim_map);
    cv::divide(temp2, temp4, cs_map);

    cv::Scalar mssim = cv::mean(ssim_map);
    return mssim[0];
}

int main() {
    // Cameraman (activo del capítulo), 256x256 - misma imagen de la pista py.
    cv::Mat img_raw = cv::imread("imagens/cameraman.png", cv::IMREAD_GRAYSCALE);
    if (img_raw.empty()) {
        return 1;
    }
    cv::Mat img_resized;
    cv::resize(img_raw, img_resized, cv::Size(256, 256));
    mm::Image img_gray(img_resized);

    std::vector<mm::Image> imgs_ssim;
    std::vector<std::string> titles_ssim;
    imgs_ssim.push_back(img_gray);
    titles_ssim.push_back("Original");

    int qs[] = {25, 50, 75, 95};
    for (int idx = 0; idx < 4; ++idx) {
        int q = qs[idx];
        mm::Image rec = mm::jpegCompress(img_gray, q);  // DCT 8x8 -> quant -> IDCT
        double psnr_v = mm::psnr(img_gray, rec);

        cv::Mat gray_mat = img_gray;
        cv::Mat rec_mat = rec;

        double ssim_v = compute_ssim(gray_mat, rec_mat);

        cv::Mat diff = cv::abs(gray_mat - rec_mat);
        cv::Mat diff_vis;
        cv::normalize(diff, diff_vis, 0, 255, cv::NORM_MINMAX, CV_8U);

        imgs_ssim.push_back(rec);
        imgs_ssim.push_back(mm::Image(diff_vis));

        char title1[100];
        snprintf(title1, sizeof(title1), "Q=%d (PSNR=%.1fdB | SSIM=%.3f)", q, psnr_v, ssim_v);
```

```
        titles_ssim.push_back(title1);

        char title2[100];
        snprintf(title2, sizeof(title2), "Mapa de erro (Q=%d) - bordas e blocagem", q);
        titles_ssim.push_back(title2);
    }

    mm::show(imgs_ssim, MM_OUT, titles_ssim, 3);

    return 0;
}
```

```
Overwriting tmp/fig_05_ssim_artefatos.cpp
```

```
!g++ -I. -std=c++17 -DMM_USE_OPENCV -I/usr/include/opencv4 tmp/fig_05_ssim_artefatos.cpp -o
tmp/fig_05_ssim_artefatos -lopencv_stitching -lopencv_alphamat -lopencv_aruco -lopencv_barcode
-lopencv_bgsegm -lopencv_bioinspired -lopencv_ccalib -lopencv_dnn_objdetect
-lopencv_dnn_superres -lopencv_dpm -lopencv_face -lopencv_freetype -lopencv_fuzzy -lopencv_hdf
-lopencv_hfs -lopencv_img_hash -lopencv_intensity_transform -lopencv_line_descriptor
-lopencv_mcc -lopencv_quality -lopencv_rapid -lopencv_reg -lopencv_rgbd -lopencv_saliency
-lopencv_shape -lopencv_stereo -lopencv_structured_light -lopencv_phase_unwrapping
-lopencv_superres -lopencv_optflow -lopencv_surface_matching -lopencv_tracking
-lopencv_highgui -lopencv_datasets -lopencv_text -lopencv_plot -lopencv_ml -lopencv_videostab
-lopencv_videoio -lopencv_viz -lopencv_wechat_qrcode -lopencv_ximgproc -lopencv_video
-lopencv_xobjdetect -lopencv_objdetect -lopencv_calib3d -lopencv_imgcodecs -lopencv_features2d
-lopencv_dnn -lopencv_flann -lopencv_xphoto -lopencv_photo -lopencv_imgproc -lopencv_core \
  && ./tmp/fig_05_ssim_artefatos \
  && test -f "tmp/fig_05_ssim_artefatos.png" \
  || echo "  mm::show não gravou tmp/fig_05_ssim_artefatos.png"
```

```
[1] Original
[2] Q=25 (PSNR=30.7dB | SSIM=0.860)
[3] Mapa de erro (Q=25) - bordas e blocagem
[4] Q=50 (PSNR=32.8dB | SSIM=0.905)
[5] Mapa de erro (Q=50) - bordas e blocagem
[6] Q=75 (PSNR=35.2dB | SSIM=0.938)
[7] Mapa de erro (Q=75) - bordas e blocagem
[8] Q=95 (PSNR=44.8dB | SSIM=0.990)
[9] Mapa de erro (Q=95) - bordas e blocagem
```

```
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)")
```

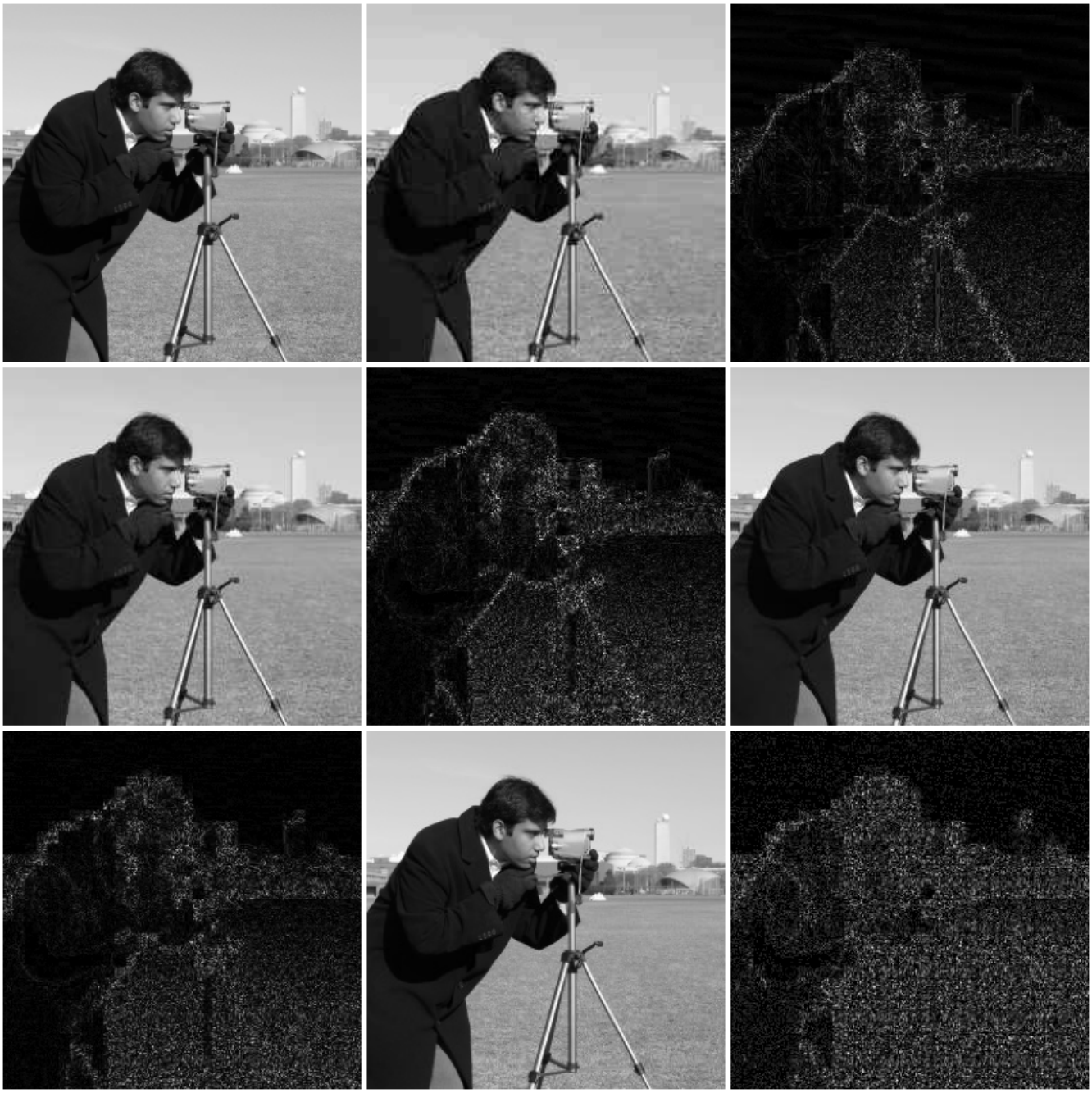


Figura 5.29: Análise espacial de degradação: imagens reconstruídas e respectivos mapas de erro absoluto normalizados para diferentes fatores de qualidade JPEG.

i Interpretando los mapas de error

Los mapas de error presentados fueron **normalizados individualmente** (`cv2.NORM_MINMAX`) para maximizar el contraste visual y revelar la estructura espacial de las distorsiones. Esto significa que:

- En **Q=95**, el error absoluto es del orden de **0.5–1.5 niveles de gris** (imperceptible visualmente), pero la normalización lo amplifica a blanco y negro para evidenciar su ubicación en bordes y transiciones.
- En **Q=25**, el error absoluto es **10–20 veces mayor** (5–15 niveles de gris), pero la normalización también lo lleva al mismo intervalo [0, 255].

Por lo tanto, **la intensidad del blanco en los mapas NO es comparable entre diferentes calidades** — los mapas sirven únicamente para revelar la **firma espacial** del error (bordes vs bloques),

no su magnitud. La magnitud correcta está dada por los valores de PSNR y SSIM, que muestran claramente que Q=95 tiene un error mucho menor que Q=25.

Síntesis — Compresión JPEG

El proceso de compresión en el estándar JPEG se basa en la aplicación combinada de transformaciones espaciales, perceptuales y estadísticas para reducir las redundancias de una imagen. La Tabla 5.9 resume el papel de cada etapa en el *pipeline* y su respectivo impacto en la reducción de datos.

Tabla 5.9: Síntesis de las etapas del *pipeline* de compresión JPEG y sus respectivos impactos.

Etapa	Operación Analítica	Mecanismo de Ganancia / Compresión
Conversión YC_bC_r	Aislamiento de los canales de luminancia y crominancia.	Modela la percepción del SVH, permitiendo tratar el color y el brillo de forma independiente.
Submuestreo 4:2:0	Reducción de la resolución espacial de los canales de color (C_b y C_r).	Elimina aproximadamente el 50% de los datos brutos con un impacto visual mínimo.
DCT 8×8	Mapeo del dominio espacial al dominio de frecuencias espaciales.	Compactación de energía, concentrando la información vital en los primeros coeficientes.
Cuantización Lineal	División entera de los coeficientes por una matriz de ponderación $Q(u, v)$.	Principal fuente de compresión con pérdida; elimina altas frecuencias imperceptibles.
Codificación Entrópica	Aplicación de algoritmos RLE y codificación de Huffman.	Compresión estadística sin pérdida, optimizada por las largas series de coeficientes nulos.

Artefactos de Degradación Característicos

La aplicación de tasas de compresión excesivamente agresivas (factores de calidad reducidos) introduce distorsiones predecibles en la imagen reconstruida, derivadas de las limitaciones matemáticas del modelo:

- **Artefactos de bloqueo (*blocking artifacts*):** Discontinuidades geométricas visibles en los límites de los bloques de 8×8 píxeles, causadas por la pérdida de correlación espacial tras la cuantización severa de los componentes de CA.
- **Efecto de dispersión (*ringing*):** Oscilaciones fantasma o distorsiones de "humo" alrededor de bordes nítidos y de alto contraste, provocadas por la eliminación abrupta de armónicos de alta frecuencia necesarios para reconstruir funciones escalón.
- **Pérdida de textura fina:** Atenuación de detalles de alta frecuencia y bajo contraste (como céspedes, tejidos o porosidad), lo que hace que regiones originalmente texturizadas adopten un aspecto excesivamente liso u homogeneizado.

5.9 Aplicación Práctica: Eliminación de Ruido mediante Filtrado Híbrido

Reuniendo las técnicas consolidadas a lo largo de este capítulo, se presenta un *pipeline* completo de **restauración de imágenes** que combina el análisis espectral en el dominio de la frecuencia con el filtrado adaptativo en el dominio espacial. El objetivo es atenuar un ruido mixto (compuesto por degradación gaussiana e interferencia periódica) preservando al máximo los detalles estructurales de la imagen original.

$$\text{Imagen Ruidosa} \xrightarrow{\text{FFT2}} \xrightarrow{\text{Filtro Notch Gaussiano}} \xrightarrow{\text{IFFT2}} \xrightarrow{\text{Filtro Bilateral}} \text{Imagen Restaurada}$$

i Evaluación Complementaria: PSNR vs. SSIM

El par de métricas estadísticas PSNR y SSIM proporciona una evaluación cualitativa y morfológica complementaria del proceso de restauración:

- **PSNR:** Penaliza uniformemente la desviación cuadrática media píxel a píxel.
- **SSIM:** Evalúa la preservación de estructuras locales perceptualmente relevantes (luminancia, contraste y contornos).

En la práctica, existe un compromiso analítico (*trade-off*) entre **reducción de ruido** y **preservación de detalles**: los filtros espaciales excesivamente agresivos atenúan bien el ruido de alta frecuencia, pero degradan texturas finas y suavizan bordes nítidos — lo que **reduce simultáneamente** tanto el PSNR como el SSIM en relación con la imagen original. El desafío del diseño de filtros es encontrar el punto de equilibrio que maximice ambas métricas, garantizando una restauración fiel y visualmente agradable.

5.9.1 Análisis de Rendimiento y Conclusión del Capítulo

Los resultados numéricos y visuales generados por la Figura 5.30 demuestran la relevancia práctica de asociar diferentes dominios de procesamiento. La inserción simultánea de ruido periódico y estocástico corrompe las propiedades morfológicas de la señal, reduciendo severamente los índices de similitud y la relación señal-ruido de la imagen de referencia.

El aislamiento y la supresión de los picos armónicos en el dominio de la frecuencia mediante la máscara *notch* eliminan las franjas de interferencia senoidales dispersas sobre el espacio bidimensional. Como se evidencia en los datos impresos de la Figura 5.30, este filtrado quirúrgico promueve un salto inmediato y sustancial en la métrica PSNR. No obstante, el ruido Gaussiano de alta frecuencia permanece activo de forma homogénea en el espectro, lo que exige un enfoque complementario.

La restauración final se consolida en el dominio espacial con la introducción del filtro bilateral. A diferencia de los operadores de paso bajo convencionales (como el Gaussiano o el de media), que suavizarían indiscriminadamente el ruido y los contornos estructurales, el filtrado bilateral calcula pesos ponderados por la proximidad geométrica y por la diferencia de intensidad radiométrica. Este comportamiento adaptativo atenúa las fluctuaciones estocásticas remanentes en las regiones de transición suave y preserva la nitidez de los bordes espaciales.

La convergencia de ambos enfoques resulta en una **mejora sustancial y simultánea** del PSNR y del SSIM en relación con la imagen ruidosa — aunque los valores finales permanecen inferiores a los de la imagen original (PSNR $= \infty$, SSIM $= 1{,}0$), debido a la pérdida inevitable de información espectral y de textura durante los procesos de filtrado. La atenuación suave (gaussiana) de los picos en el espectro evita los artefactos de *ringing*, mientras que el filtro bilateral elimina el ruido estocástico residual sin comprometer la nitidez de los bordes. Los resultados confirman la eficacia y la complementariedad práctica de las herramientas de análisis de frecuencia presentadas en este capítulo, demostrando que el filtrado híbrido (frecuencia + espacial) es superior a cualquier enfoque aislado para la restauración de imágenes degradadas por ruido mixto.

```
%%writefile tmp/fig_05_pipeline_denoising.cpp
#define MM_OUT "tmp/fig_05_pipeline_denoising.png"
//| label: fig-05-pipeline-denoising
//| fig-cap: "*Pipeline* completo de remoção de ruído misto: (1) adição de ruído gaussiano e
periódico; (2) identificação de picos de interferência no espectro de frequências; (3)
aplicação de máscara *notch* com atenuação gaussiana suave; (4) pós-processamento via filtro
bilateral para eliminação do ruído estocástico residual."
//| echo: true
//| output: true

#include <opencv2/opencv.hpp>
#include <vector>
#include <string>
#include <cmath>
```

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

// Función auxiliar: suprimir pico gaussiano en la máscara
static void suprimir_pico_gaussiano(cv::Mat& mask, int cy, int cx, double sigma = 3.0) {
    int H = mask.rows, W = mask.cols;
    double inv2sigma2 = 1.0 / (2.0 * sigma * sigma);
    for (int y = 0; y < H; ++y) {
        for (int x = 0; x < W; ++x) {
            double dy = y - cy;
            double dx = x - cx;
            double notch = std::exp(-(dy * dy + dx * dx) * inv2sigma2);
            mask.at<double>(y, x) *= (1.0 - notch);
        }
    }
}

int main() {
    // Cameraman (asset del capítulo), 256x256 - misma imagen de la trilha py.
    mm::Image img_orig = mm::read("imagens/cameraman.png");
    cv::Mat img_resized;
    cv::resize((cv::Mat)img_orig, img_resized, cv::Size(256, 256));
    mm::Image img_gray = mm::gray(img_resized);
    int h_img = img_gray.h;
    int w_img = img_gray.w;

    //   1. Ruido mixto: gaussiano + periódico
    std::mt19937 gen(42);
    std::normal_distribution<double> dist(0.0, 15.0);

    int u0 = 15, v0 = 10;

    // Ruido gaussiano
    cv::Mat ruido_gauss(h_img, w_img, CV_64F);
    for (int y = 0; y < h_img; ++y) {
        for (int x = 0; x < w_img; ++x) {
            ruido_gauss.at<double>(y, x) = dist(gen);
        }
    }

    // Ruido periódico
    cv::Mat ruido_period(h_img, w_img, CV_64F);
    for (int y = 0; y < h_img; ++y) {
        for (int x = 0; x < w_img; ++x) {
            ruido_period.at<double>(y, x) = 30.0 * std::sin(2.0 * M_PI * (u0 * (double)x /
w_img + v0 * (double)y / h_img));
        }
    }

    // Imagen ruidosa
    cv::Mat img_gray64f(h_img, w_img, CV_64F);
    for (int y = 0; y < h_img; ++y) {
        for (int x = 0; x < w_img; ++x) {
            double val = (double)img_gray.data[y * img_gray.w + x] + ruido_gauss.at<double>(y,
x) + ruido_period.at<double>(y, x);
            val = std::min(255.0, std::max(0.0, val));
            img_gray64f.at<double>(y, x) = val;
        }
    }
    cv::Mat img_noisy8u;
```

```
    img_gray64f.convertTo(img_noisy8u, CV_8U);
    mm::Image img_noisy(img_noisy8u);

    //   2. Espectro (log-magnitud) de la imagen ruidosa
    mm::Image mag_n = mm::spectrumMag(img_noisy);

    //   3. Máscara notch gaussiana en los 4 picos periódicos
    int cy0 = h_img / 2, cx0 = w_img / 2;
    cv::Mat mascara_notch(h_img, w_img, CV_64F, cv::Scalar(1.0));

    // Picos: (v0,u0), (-v0,-u0), (v0,-u0), (-v0,u0)
    int picos[4][2] = {{v0, u0}, {-v0, -u0}, {v0, -u0}, {-v0, u0}};
    for (int i = 0; i < 4; ++i) {
        suprimir_pico_gaussiano(mascara_notch, cy0 + picos[i][0], cx0 + picos[i][1], 3.0);
    }

    //   4. Filtrado: notch en el dominio de la frecuencia + bilateral
    mm::Image img_notch = mm::freqFilter(img_noisy, mascara_notch);
    cv::Mat img_notch_cv = img_notch; // convertir a cv::Mat
    cv::Mat img_den_cv;
    cv::bilateralFilter(img_notch_cv, img_den_cv, 7, 25, 7);
    mm::Image img_den(img_den_cv);

    // Máscara visible (0-255)
    cv::Mat mascara_vis64f;
    cv::multiply(mascara_notch, 255.0, mascara_vis64f);
    cv::Mat mascara_vis8u;
    mascara_vis64f.convertTo(mascara_vis8u, CV_8U);
    mm::Image mascara_vis(mascara_vis8u);

    // Calcular PSNR
    double psnr_n = mm::psnr(img_gray, img_noisy);
    double psnr_no = mm::psnr(img_gray, img_notch);
    double psnr_d = mm::psnr(img_gray, img_den);

    std::cout << "Ruidosa: PSNR=" << psnr_n << " dB | Apos notch: " << psnr_no << " dB |
Notch+bilateral: " << psnr_d << " dB" << std::endl;

    // Mostrar resultados
    mm::show(
        std::vector<mm::Image>{img_gray, img_noisy, mag_n, mascara_vis, img_notch, img_den},
        MM_OUT,
        std::vector<std::string>{
            "Original",
            "Ruidosa (PSNR=" + std::to_string(psnr_n).substr(0, 4) + " dB)",
            "Espectro (picos visiveis)",
            "Mascara notch (gaussiana)",
            "Apos notch (PSNR=" + std::to_string(psnr_no).substr(0, 4) + " dB)",
            "Notch + bilateral (PSNR=" + std::to_string(psnr_d).substr(0, 4) + " dB)"
        },
        6
    );

    return 0;
}
```

```
Overwriting tmp/fig_05_pipeline_denoising.cpp
```

```
!g++ -I. -std=c++17 -DMM_USE_OPENCV -I/usr/include/opencv4 tmp/fig_05_pipeline_denoising.cpp
-o tmp/fig_05_pipeline_denoising -lopencv_stitching -lopencv_alphamat -lopencv_aruco
-lopencv_barcode -lopencv_bgsegm -lopencv_bioinspired -lopencv_ccalib -lopencv_dnn_objdetect
-lopencv_dnn_superres -lopencv_dpm -lopencv_face -lopencv_freetype -lopencv_fuzzy -lopencv_hdf
-lopencv_hfs -lopencv_img_hash -lopencv_intensity_transform -lopencv_line_descriptor
-lopencv_mcc -lopencv_quality -lopencv_rapid -lopencv_reg -lopencv_rgbd -lopencv_saliency
-lopencv_shape -lopencv_stereo -lopencv_structured_light -lopencv_phase_unwrapping
-lopencv_superres -lopencv_optflow -lopencv_surface_matching -lopencv_tracking
-lopencv_highgui -lopencv_datasets -lopencv_text -lopencv_plot -lopencv_ml -lopencv_videostab
-lopencv_videoio -lopencv_viz -lopencv_wechat_qrcode -lopencv_ximgproc -lopencv_video
-lopencv_xobjdetect -lopencv_objdetect -lopencv_calib3d -lopencv_imgcodecs -lopencv_features2d
-lopencv_dnn -lopencv_flann -lopencv_xphoto -lopencv_photo -lopencv_imgproc -lopencv_core \
  && ./tmp/fig_05_pipeline_denoising \
  && test -f "tmp/fig_05_pipeline_denoising.png" \
  || echo "  mm::show não gravou tmp/fig_05_pipeline_denoising.png"
```

```
Ruidosa: PSNR=20.1968 dB | Apos notch: 22.3552 dB | Notch+bilateral: 23.5442 dB
[1] Original
[2] Ruidosa (PSNR=20.1 dB)
[3] Espectro (picos visiveis)
[4] Mascara notch (gaussiana)
[5] Apos notch (PSNR=22.3 dB)
[6] Notch + bilateral (PSNR=23.5 dB)
```

```
try:
    mm.show(mm.read("tmp/fig_05_pipeline_denoising.png"), figsize=(20, 4))
except Exception as _e:
    print("figura indisponivel nesta trilha (C++): " + repr(_e) + "
    tmp/fig_05_pipeline_denoising.png (ver a versao Python)")
```

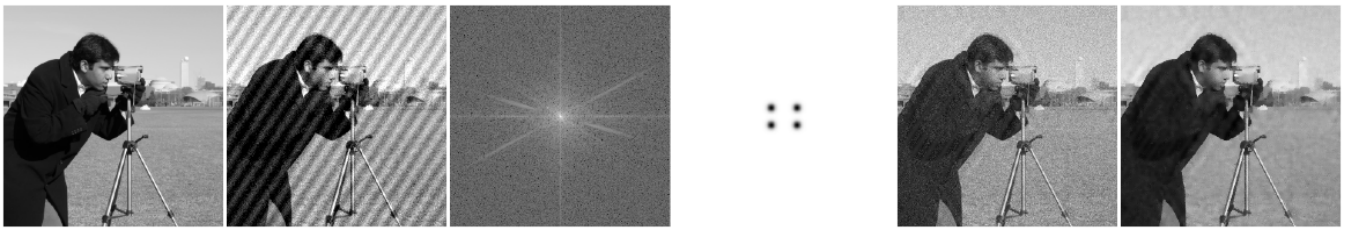


Figura 5.30: *Pipeline* completo de remoção de ruído misto: (1) adição de ruído gaussiano e periódico; (2) identificação de picos de interferência no espectro de frequências; (3) aplicação de máscara *notch* com atenuação gaussiana suave; (4) pós-processamento via filtro bilateral para eliminação do ruído estocástico residual.

5.10 Resumen del Capítulo

La transición del **dominio espacial** al **dominio de la frecuencia** revela la distribución espectral de energía de la imagen, estableciendo la base analítica para el filtrado avanzado, la restauración y la compresión de datos. La articulación estructural de estos conceptos se sintetiza en el mapa conceptual de la Figura 5.31.

Fundamentos Esenciales

- **DFT y Percepción Visual:** El espectro descompone la imagen en componentes armónicas. La **fase** retiene la inteligibilidad geométrica de la escena y la localización de contornos, mientras que la **magnitud** dicta la distribución de contraste y las amplitudes globales.
- **Eficiencia Algorítmica:** El Teorema de la Convolución hace viable el procesamiento de máscaras de gran escala en el dominio de la frecuencia mediante FFT, reduciendo la complejidad computacional asintótica de $O(N^2K^2)$ en el espacio a $O(N^2 \log N)$.

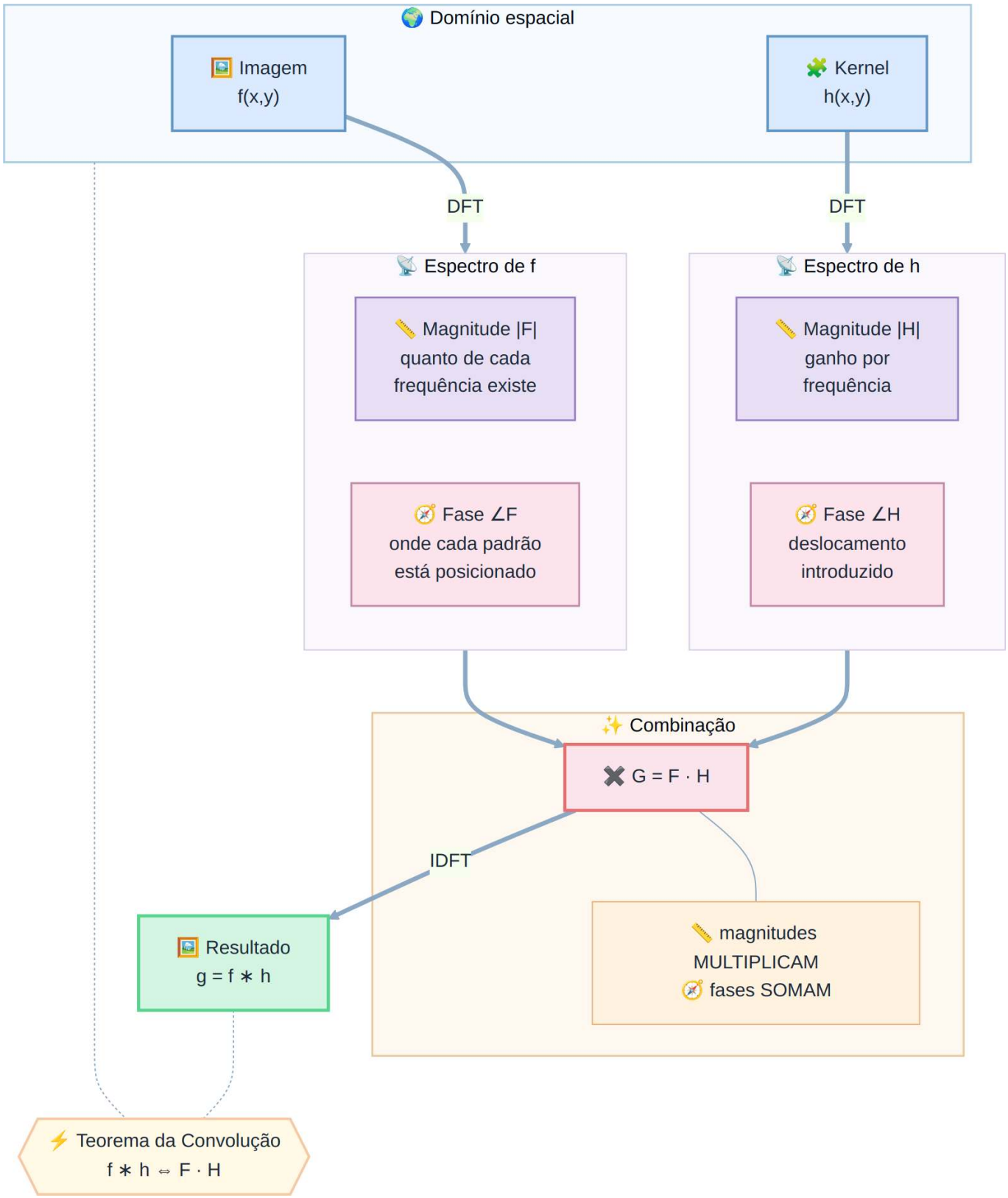


Figura 5.31: Mapa conceptual de las transformaciones y propiedades en el dominio de la frecuencia.

- **Fenómeno de *Ringing*:** Los cortes abruptos en el espectro (Filtros Ideales) generan oscilaciones espaciales no deseadas (fenómeno de Gibbs). La atenuación suave mediante filtros de **Butterworth** o **Gaussianos** elimina estas discontinuidades.
- **Análisis Multirresolución mediante *Wavelets*:** Superando el carácter puramente global de Fourier, la DWT captura la frecuencia y la localización espacial simultáneamente, fundamentando el estándar JPEG 2000 y subsidiando representaciones jerárquicas análogas a las extracciones de características en Redes Neuronales Convolucionales (CNNs).
- **Compresión Perceptual (DCT):** El *pipeline* JPEG explora las limitaciones de contraste del sistema visual humano en altas frecuencias espaciales. La DCT aísla la energía de bloques 8×8, permitiendo que la cuantización descarte coeficientes AC de detalles finos sin perjuicio perceptual severo.

Próximos Pasos: El **Capítulo 6** inaugura la Parte II de la obra, aplicando las herramientas de procesamiento de imágenes en la resolución de problemas reales de inspección industrial. Se explorarán técnicas de **segmentación y análisis de formas** para la detección automática de fallas en líneas de producción — desde la identificación de defectos superficiales en piezas hasta la lectura *QRCode* en pruebas, consolidando el puente entre la teoría presentada en la Parte I y las demandas prácticas de la visión computacional.

5.11 🤖 Uso de Gemini Notebook como Tutor Complementario

En esta edición, se incentiva el uso de la plataforma **Gemini Notebook** como herramienta complementaria de aprendizaje — **no como sustituta** de la lectura atenta, de la resolución de ejercicios o de la experimentación práctica. Basado en arquitecturas de inteligencia artificial, el sistema utiliza exclusivamente el material didáctico y los documentos proporcionados por el autor como base de conocimiento, asegurando que las respuestas generadas estén conceptualmente alineadas con el contenido programático y con el enfoque pedagógico adoptado a lo largo de esta obra.

Acceso al Tutor Inteligente

🚀 ACCEDER A Gemini Notebook: CAPÍTULO 05

🌐 Idioma y Lenguaje de Programación

El proyecto de este capítulo en Gemini Notebook fue construido únicamente con el texto en **portugués** y los ejemplos de código en **Python**. Si estás estudiando con la edición en inglés o francés, o siguiendo la ruta en C++, las respuestas del tutor pueden no corresponder exactamente con la versión que estás leyendo.

Directrices sobre el Contenido Generado por Inteligencia Artificial

Aunque las herramientas de inteligencia artificial constituyen aliados eficientes en el proceso de aprendizaje y revisión, el contenido generado está sujeto a inconsistencias o imprecisiones técnicas. Por ello, es indispensable la consulta sistemática de libros de texto, artículos científicos y fuentes académicas indexadas para la validación rigurosa de la información. Se recomienda encarecidamente la ejecución y la modificación de los ejemplos prácticos en Python proporcionados en este capítulo como método primario de verificación experimental de los resultados.

5.12 Lista de Ejercicios

1. **(10%) Implementación Directa de la DFT 2D:** Implemente analíticamente la Transformada Discreta de Fourier 2D (DFT) sin la ayuda de funciones nativas de bibliotecas (como `np.fft.fft2`), utilizando estrictamente la formulación matemática definida en la Ecuación 5.1 para una matriz de dimensiones 16×16. Realice la validación numérica comparando los coeficientes generados con los resultados de la función `np.fft.fft2`, asegurándose de que la desviación absoluta máxima sea inferior

a 10^{-8}. Mida los tiempos de ejecución de ambos métodos y presente una justificación teórica para la disparidad observada en términos de complejidad asintótica.

2. **(15%) Supresión de Ruido Periódico:** Agregue interferencias sinusoidales con frecuencias espaciales $(u_0, v_0) \in \{(5, 10), (20, 5), (30, 30)\}$ a la imagen de prueba del *Cameraman*. Para cada escenario de degradación, diseñe una máscara de filtrado *notch* específica en el dominio de la frecuencia para aislar y atenuar los picos armónicos no deseados. Evalúe cuantitativamente la eficacia del proceso de restauración mediante el cálculo de las métricas de PSNR y SSIM. Discuta analíticamente el compromiso (*trade-off*) entre la atenuación del ruido sinusoidal y la indeseada atenuación de características estructurales legítimas de la imagen.

3. **(15%) Análisis Comparativo de Operadores Pasa-Bajas:** Realice un estudio comparativo entre los filtros pasa-bajas Ideal, Gaussiano y Butterworth (con órdenes armónicas $n = 1, 2, 4$), parametrizados con frecuencias de corte $D_0 = 20, 40, 60$ píxeles. Para cada combinación estructural, calcule los índices PSNR y SSIM de la imagen resultante frente a la señal original de referencia. Organice los datos cuantitativos en una tabla estructurada y grafique las curvas unidimensionales de las funciones de transferencia correspondientes a lo largo del perfil horizontal $H(u, 0)$.

4. **(15%) Banco de Filtros Multirresolución de Haar:** Desarrolle un script para ejecutar manualmente la descomposición *wavelet* discreta 2D de primer nivel utilizando la familia Haar. El algoritmo debe calcular los coeficientes de los filtros correspondientes pasa-bajas (h) y pasa-altas (g), aplicándolos de forma separable sobre las filas y columnas de la matriz, seguidos por la operación de diezmado (submuestreo espacial por un factor de 2). Valide numéricamente la exactitud de su implementación contrastando las subbandas obtenidas con la salida de la función `pywt.dwt2(img, 'haar')`.

5. **(15%) Compresión Dispersa por Umbralización Wavelet:** Aplique la técnica de filtrado por umbralización abrupta (*hard thresholding*) sobre los coeficientes de detalle de la descomposición *wavelet*, adoptando los umbrales numéricos $T \in \{5, 10, 20, 40, 80\}$ para las familias Haar, Daubechies (`db4`) y Symlets (`sym4`). Después de realizar el proceso de síntesis mediante la transformada inversa (`pywt.waverec2`), calcule los valores de PSNR y SSIM de cada imagen reconstruida. Identifique y justifique qué combinación de familia *wavelet* y umbral T maximiza la similitud estructural.

6. **(15%) Construcción de Codificador JPEG Simplificado:** Implemente el flujo completo de compresión de datos simulando el estándar JPEG. El flujo debe abarcar: conversión espacial $RGB \to YC_bC_r$, submuestreo cromático en la proporción 4:2:0, segmentación de la luminancia en bloques disjuntos de 8×8 píxeles, aplicación de la DCT-II 2D ortogonal y cuantización lineal basada en la matriz normalizada de luminancia escalada por factores de calidad deseados. Realice la decodificación inversa y compare cuantitativamente las reconstrucciones con los archivos generados por la función `cv2.imencode` para los factores de calidad de 20, 50 y 80.

7. **(15%) Análisis Perceptual en Contenidos Heterogéneos:** Desarrolle una imagen sintética compuesta por tres regiones distintas y de características espectrales contrastantes: una textura fotográfica compleja (representando altas frecuencias estocásticas), un área de texto vectorizado con bordes nítidos (representando transiciones escalón puras) y un gradiente lineal continuo (representando bajas frecuencias homogéneas). Someta esta imagen mixta a los procesos de compresión bajo los formatos JPEG, PNG y WebP. Evalúe e interprete los resultados correlacionando el tamaño final del archivo en disco con las métricas PSNR y SSIM obtenidas, justificando qué formato exhibe el mejor desempeño para señales de naturaleza heterogénea y por qué ocurre esa ventaja en términos de compactación de energía y preservación perceptual.

Referencias del Capítulo

La fundamentación teórica y el desarrollo analítico de los conceptos tratados en este capítulo se basan en las siguientes obras de referencia:

- **Gonzalez (2018)** — Formulaciones clásicas de Transformadas Discretas de Fourier 2D (DFT), diseño de filtros analíticos en el dominio de la frecuencia, Transformada Discreta de Cosenos (DCT) y principios

fundamentales de sistemas de compresión de imágenes.

- **Oppenheim (2010)** — Teoría formal de señales y sistemas aplicados en el dominio discreto, cubriendo las propiedades matemáticas de la DFT y el modelado analítico del Teorema de la Convolución.
- **Mallat (1999)** — Fundamentación matemática de la teoría de *wavelets*, formalización del análisis multirresolución (MRA) y arquitectura de bancos de filtros diádicos.
- **Wallace (1991)** — Especificación original y aspectos de ingeniería del estándar de compresión ISO/IEC JPEG, con énfasis en los criterios psicovisuales para el diseño de matrices de cuantización DCT.
- **Szeliski (2022)** — Modelado computacional y caracterización de métricas modernas de fidelidad y calidad perceptual (PSNR y SSIM), así como el análisis comparativo de formatos de imagen rasterizados de alto rendimiento.

5.13 💻 Parte Práctica con Ejercicios de Programación

La presente lista de ejercicios de programación (EP) consolida las formulaciones teóricas presentadas a lo largo del Capítulo 5 — Transformadas y Compresión — mediante una ruta práctica aplicada. Los ejercicios se estructuran a partir de matrices de dimensiones reducidas, lo que permite la validación analítica y la inspección manual de cada coeficiente, manteniendo la consistencia metodológica adoptada en los capítulos anteriores.

El encadenamiento de los ejercicios reproduce rigurosamente el flujo conceptual del capítulo: se comienza con la implementación explícita de la Transformada Discreta de Fourier (DFT) a partir de su definición matemática fundamental; se avanza hacia el diseño de filtros pasa-baja y máscaras *notch* en el dominio de la frecuencia; se aplica la cuantización de coeficientes (núcleo de la compresión con pérdida); y se concluye con la integración de estas etapas en la construcción de un *pipeline* de compresión JPEG simplificado y en el análisis perceptual de formatos de imagen.

> ❗ Directrices para la Resolución de los Ejercicios de Programación
>
> En todos los ejercicios de este capítulo, las coordenadas del **centro del espectro** (origen de las frecuencias espaciales tras la aplicación del desplazamiento `fftshift`) deben determinarse mediante división entera. Para una matriz con L filas y C columnas, la componente de frecuencia nula se localiza en la posición:
>
> $$(c_y, c_x) = \left(\left\lfloor \frac{L}{2} \right\rfloor, \left\lfloor \frac{C}{2} \right\rfloor \right)$$
>
> Esta convención es rigurosamente idéntica a la adoptada por la función `np.fft.fftshift`. Además, en todas las etapas que requieran discretización o redondeo numérico (ya sea en la cuantización de coeficientes AC o en la reconstrucción final de píxeles), debe emplearse el redondeo estándar al entero más cercano (*round half away from zero*), mitigando ambigüedades en valores con fracción exactamente igual a 0.5.

🎯 Objetivo de este Cuaderno

El cuaderno permite desarrollar, validar, organizar y probar soluciones de **Ejercicios de Programación (EPs)** en entornos interactivos, como Colab, con los mismos casos de prueba de Moodle, copiándolos allí solo en el momento de registrar la nota oficial.

Download

Descargue `morph.py` y `testsuite.py` ejecutando la celda a continuación:

```
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
```

```
Entorno listo. Morph: 1.1.9 | OpenCV: 5.0.0 | TestSuite: 1.1.2
```

Ejecutando las Pruebas

Para evaluar las pruebas, ejecuta `TestSuite("EP05_01.extensión").run()` en una nueva celda, reemplazando la extensión por la del lenguaje utilizado (`.py`, `.java`, `.c`, `.cpp`, `.js` o `.r`). El sistema descarga los casos de prueba de GitHub, ejecuta el programa y calcula la nota automáticamente.

Para probar código Python directamente, sin guardar archivo, usa `run_code(codigo)` pasando el código como *cadena* en una variable `codigo`:

```
codigo = """
from morph import mm
# ... tu código aquí ...
"""
TestSuite("EP05_01").run_code(codigo)
```

5.13.1 EP05_01 🟢 Filtro Pasa-Bajas Ideal por Distancia en el Espectro

En un ***escáner* de documentos antiguo**, el sensor capta papel arrugado y textura de fibra junto con el texto — ruido de alta frecuencia que "contamina" el espectro en los bordes. El técnico de mantenimiento no tiene acceso a la imagen original, solo al **espectro de magnitud ya calculado** por el software del *escáner*. Su trabajo es simple y quirúrgico: mantener únicamente el **círculo central** de bajas frecuencias (la estructura global del documento) y borrar todo lo que esté fuera del radio D_0, eliminando la textura fina sin siquiera tocar la imagen espacial.

Este es el **Filtro Pasa-Bajas Ideal (LPFI)**: la operación espectral más directa del capítulo, pero también la que mejor revela la anatomía de un espectro centrado.

5.13.1.1 📋 Directrices de Implementación

1. **Dimensiones:** Leer los enteros L (filas) y C (columnas) del espectro de magnitud — ya proporcionado **centrado** (equivalente a la salida de `np.fft.fftshift`).
2. **Frecuencia de corte:** Leer el entero D_0.
3. **Datos:** Leer los valores enteros de la matriz de magnitud, fila por fila.
4. **Centro del espectro:** Calcular $(c_y, c_x) = (L \,//\, 2,\ C \,//\, 2)$.
5. **Distancia:** Para cada posición (u, v), calcular

$$D(u,v) = \sqrt{(u-c_y)^2 + (v-c_x)^2}$$

6. **Máscara ideal:** Aplicar

$$H(u,v) = \begin{cases} 1, & D(u,v) \le D_0 \\ 0, & D(u,v) > D_0 \end{cases}$$

7. **Filtrado:** El valor de salida es $\text{mag}'(u,v) = \text{mag}(u,v) \cdot H(u,v)$.

8. **Salida:** Mostrar la matriz filtrada con dimensiones $L \times C$.

5.13.1.2 📌 Restricciones Computacionales

- **Comparación no estricta:** el criterio usa $D(u,v) \leq D_0$ (la frontera pertenece al filtro, es decir, se mantiene).
- **Tipo:** todos los valores de entrada y salida son enteros; la distancia se calcula en punto flotante solo internamente.
- **Sin redondeo de magnitud:** como la entrada ya es entera y la máscara es binaria (0 o 1), la salida nunca necesita redondeo.

5.13.1.3 🧠 Fundamentación Teórica

Región	Distancia al centro	Efecto del filtro
Centro ($D \leq D_0$)	Bajas frecuencias	Preservadas — estructura global mantenida
Bordes ($D > D_0$)	Altas frecuencias	Puestas a cero — textura y ruido eliminados
D_0 **pequeño**	—	Imagen reconstruida quedaría muy borrosa
D_0 **grande**	—	Poca filtración; casi toda la energía preservada

5.13.1.4 📦 Especificación de Entrada y Salida (VPL)

Entrada:

- Línea 1: Entero L.
- Línea 2: Entero C.
- Línea 3: Entero D_0.
- Líneas siguientes: Elementos enteros de la matriz de magnitud (centrada).

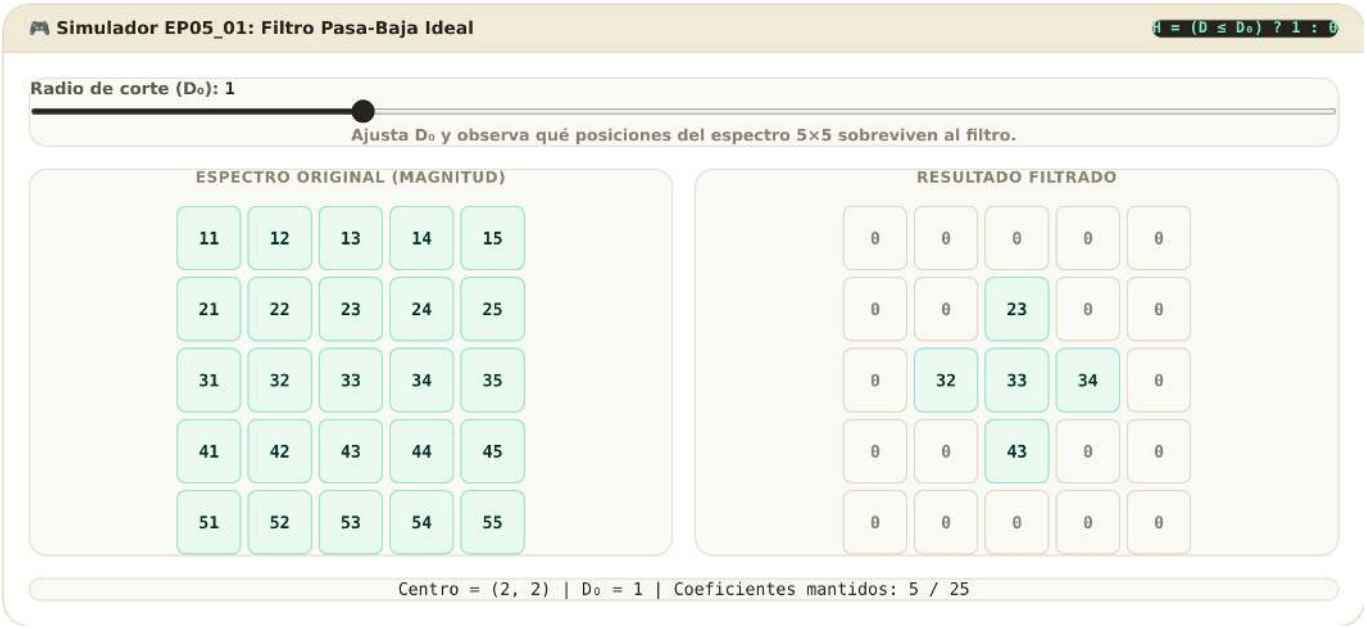
Salida:

- Matriz filtrada en L filas y C columnas, separados por espacio.

5.13.1.5 📌 Ejemplos

Entrada	Salida	Observación
3 3 1 10 20 30 40 50 60 70 80 90	0 20 0 40 50 60 0 80 0	Centro $(1,1)$. Las esquinas tienen $D = \sqrt{2} \approx 1.41 > 1$, por lo que se ponen a cero; los vecinos ortogonales tienen $D = 1 \leq 1$ y se mantienen.
1 3 0 5 9 7	0 9 0	$L = 1, C = 3$: centro en $(0,1)$. Solo la propia posición central ($D = 0$) sobrevive a $D_0 = 0$.

```
%%writefile EP05_01.cpp
// your solution
```



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.es/cap05/sim-ep0501.html

Figura 5.32: Simulador EP05_01: Filtro Pasabajas Ideal en el Espectro

```
Overwriting EP05_01.cpp
```

```
TestSuite("EP05_01.cpp").run()
```

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

5.13.2 EP05_02 🟡 Filtro *Notch*: Eliminando Picos Periódicos

Una cámara de **inspección industrial** captura imágenes de placas de circuito, pero la fuente de alimentación de la línea de producción introduce una **interferencia eléctrica periódica** — un patrón de franjas casi imperceptible a simple vista, pero que aparece en el espectro de Fourier como **pares de picos brillantes** simétricamente posicionados alrededor del centro. El equipo de visión por computadora no puede reprocesar la captura: necesita **localizar y borrar quirúrgicamente** esos pares de picos en el espectro, preservando todo el resto de la información útil de la imagen.

Ese es el papel del **filtro rechaza-banda *notch***: a diferencia del pasa-bajas (que afecta una región continua), ataca **puntos específicos y sus simétricos**, dejando el resto del espectro intacto.

5.13.2.1 📋 Directrices de Implementación

1. **Dimensiones:** Leer los enteros L (filas) y C (columnas) del espectro de magnitud centrado.
2. **Datos:** Leer los valores enteros de la matriz de magnitud, fila por fila.
3. **Picos:** Leer el entero K (cantidad de pares de picos a eliminar).
4. **Para cada uno de los K picos:** leer tres enteros Δv, Δu, r — desplazamiento vertical, desplazamiento horizontal y radio del *notch*.
5. **Centro del espectro:** $(c_y, c_x) = (L \,//\, 2,\ C \,//\, 2)$.
6. **Supresión simétrica:** para cada pico, poner a cero **todas** las posiciones (u, v) tales que la distancia al punto $(c_y + \Delta v,\ c_x + \Delta u)$ sea $\leq r$, **y también** todas las posiciones con distancia $\leq r$ al punto simétrico $(c_y - \Delta v,\ c_x - \Delta u)$.
7. **Salida:** Mostrar la matriz resultante con dimensiones $L \times C$.

5.13.2.2 📌 Restricciones Computacionales

- **Simetría obligatoria:** cada pico informado genera **dos** discos puestos a cero (el punto y su simétrico respecto al centro) — olvidar el simétrico es el error más común.
- **Superposición:** si dos discos se superponen, la posición permanece en cero (no hay "suma" ni restauración).
- **Comparación no estricta:** una posición se pone a cero si distancia $\leq r$.
- **Orden de lectura:** los K picos deben procesarse en el orden en que aparecen en la entrada, pero el resultado final no depende del orden (las operaciones de poner a cero son conmutativas).

5.13.2.3 🧠 Fundamentación Teórica

Concepto	Papel en el filtro *notch*
Pico en $(\Delta v, \Delta u)$	Frecuencia de la interferencia periódica detectada visualmente en el espectro
Punto simétrico $(-\Delta v, -\Delta u)$	Toda DFT de señal real es hermítica: los picos siempre aparecen en pares simétricos al centro
Radio r	Controla la "anchura" del rechazo — r grande elimina más energía alrededor del pico, pero también información útil

5.13.2.4 📦 Especificación de Entrada y Salida (VPL)

Entrada:

- Línea 1: Entero L.
- Línea 2: Entero C.
- Líneas siguientes: Elementos enteros de la matriz de magnitud (centrada), L filas.
- Siguiente línea: Entero K.
- K líneas siguientes: tres enteros Δv, Δu, r (separados por espacios).

Salida:

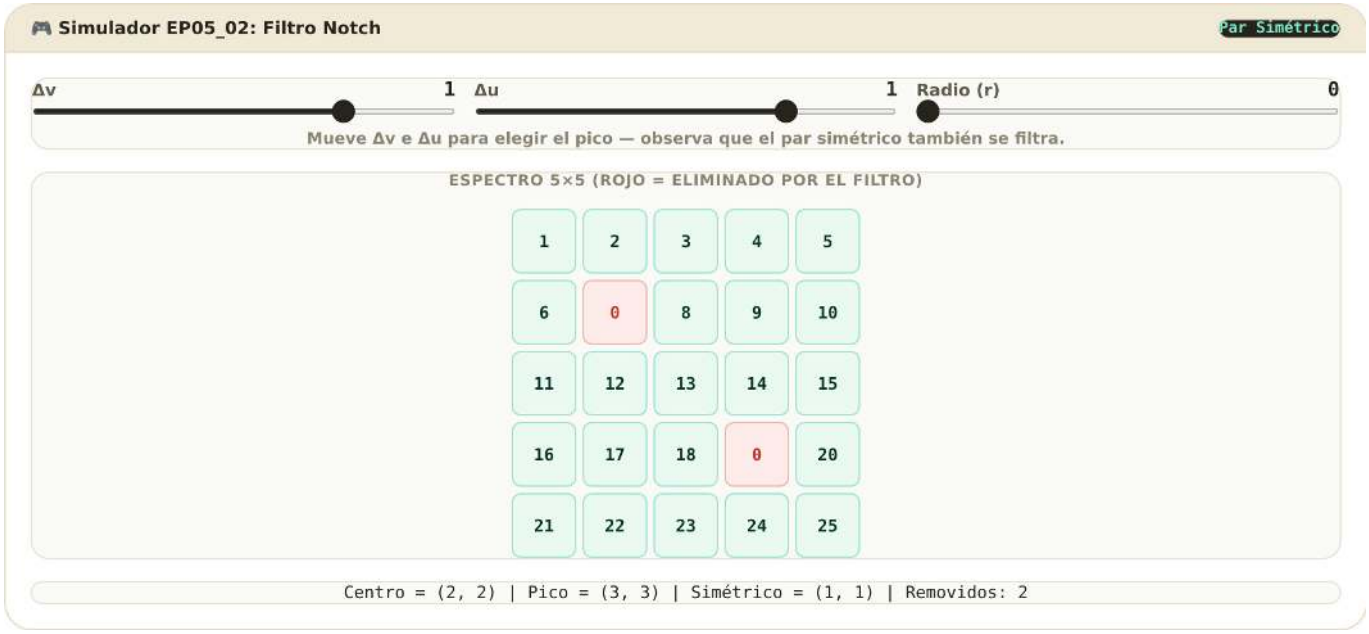
- Matriz resultante en L filas y C columnas, separadas por espacios.

5.13.2.5 📌 Ejemplos

Entrada	Salida	Observación
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 $(c_y, c_x) = (2, 2)$. El pico informado $(\Delta v, \Delta u) = (1, 1)$ genera el punto $(3, 3)$ (valor 19) y su simétrico $(1, 1)$ (valor 7), ambos puestos a cero con $r = 0$ (solo los puntos exactos).

```
%%writefile EP05_02.cpp
// your solution
```

```
Overwriting EP05_02.cpp
```



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.es/cap05/sim-ep0502.html

Figura 5.33: Simulador EP05_02: Filtro Notch

```
TestSuite("EP05_02.cpp").run()
```

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

5.13.3 EP05_03 🟠 Cuantización DCT: la Verdadera Fuente de Compresión

Una aplicación de **galería de fotos** necesita reducir el tamaño de miles de imágenes antes de hacer *upload* a la nube, sin recodificar todo desde cero. El ingeniero responsable ya tiene los **coeficientes DCT** de cada bloque 4×4 calculados (la etapa costosa computacionalmente ya se ha realizado) — solo falta aplicar la **tabla de cuantización**, la etapa que realmente descarta información y genera compresión. Los coeficientes de alta frecuencia, menos perceptibles al ojo humano, reciben divisores grandes y tienden a convertirse en **cero**; los coeficientes de baja frecuencia, más perceptibles, reciben divisores pequeños y sobreviven casi intactos.

Vas a implementar exactamente esta etapa: **cuantizar y descuantizar** (dividir, redondear, multiplicar de vuelta) — el corazón de la compresión *lossy* del JPEG.

5.13.3.1 📋 Directrices de Implementación

1. **Dimensión del bloque:** Leer el entero N (bloque $N \times N$).
2. **Coeficientes:** Leer la matriz C de coeficientes DCT, N filas con N enteros cada una (pueden ser negativos).
3. **Tabla de cuantización:** Leer la matriz Q, N filas con N enteros positivos cada una.
4. **Cuantización:** Para cada posición (u, v), calcular el índice cuantizado

$$\tilde{C}(u,v) = \text{round}\left(\frac{C(u,v)}{Q(u,v)}\right)$$

   usando redondeo estándar al entero más cercano (los valores intermedios `.5` nunca ocurren en los casos de prueba).
5. **Descuantización (reconstrucción):** Calcular

$$C'(u,v) = \tilde{C}(u,v) \times Q(u,v)$$

6. **Salida:** Mostrar la matriz reconstruida C', $N \times N$, enteros.

5.13.3.2 📌 Restricciones Computacionales

- ***Round-trip* completo:** la salida es el coeficiente **reconstruido** ($\tilde{C} \times Q$), no el índice cuantizado aislado.
- **División en punto flotante:** la división $C(u,v)/Q(u,v)$ debe realizarse en punto flotante antes del redondeo — la división entera truncada producirá un resultado incorrecto.
- **Signo preservado:** los coeficientes negativos mantienen el signo después de la cuantización y la reconstrucción.
- $Q(u,v) > 0$ **siempre:** no hay necesidad de tratar la división por cero.

5.13.3.3 🧠 Fundamentación Teórica

Coeficiente	Frecuencia	Valor típico de Q	Efecto de la cuantización
$C(0,0)$	DC (promedio del bloque)	Pequeño	Casi siempre sobrevive — domina la energía
$C(u,v)$ bajo $u+v$	Baja frecuencia	Pequeño/medio	Parcialmente preservado
$C(u,v)$ alto $u+v$	Alta frecuencia	Grande	Frecuentemente se convierte en cero — fuente de la compresión

5.13.3.4 📦 Especificación de Entrada y Salida (VPL)

Entrada:

- Línea 1: Entero N.
- N líneas siguientes: matriz C (coeficientes DCT, enteros, pueden ser negativos).
- N líneas siguientes: matriz Q (tabla de cuantización, enteros positivos).

Salida:

- Matriz reconstruida C', $N \times N$, enteros separados por espacio.

5.13.3.5 📌 Ejemplos

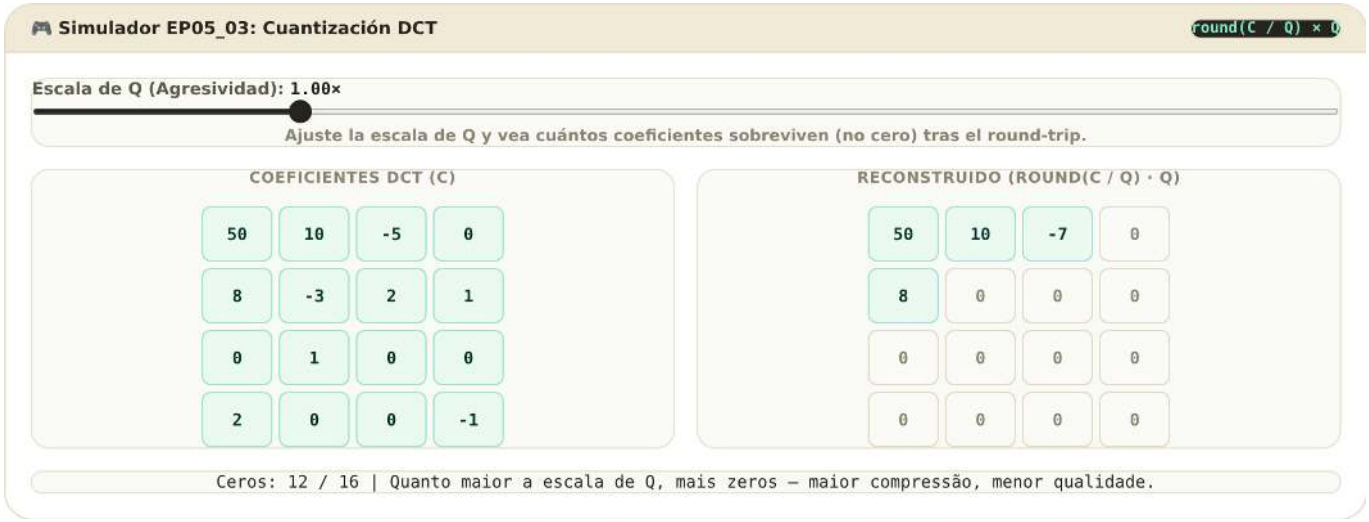
Entrada	Salida	Observación
4 50 10 -5 0 8 -3 2 1 0 1 0 0 2 0 0 -1 2 5 7 8 4 7 8 11 6 8 11 12 9 11 12 14	50 10 -7 0 8 0 0 0 0 0 0 0 0 0 0 0	$C(0,0) = 50/2 = 25 \to 25{\times}2 = 50$ (preservado). $C(0,2) = -5/7 \approx -0.71 \to -1 \to -1 \times 7 = -7$. Ya $C(1,1) = -3/7 \approx -0.43 \to 0$: anulado por la cuantización — la mayor parte del bloque se convierte en cero, ilustrando la compactación de energía en la esquina superior izquierda.

```
%%writefile EP05_03.cpp
// your solution
```

```
Overwriting EP05_03.cpp
```

```
TestSuite("EP05_03.cpp").run()
```

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



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.es/cap05/sim-ep0503.html

Figura 5.34: Simulador EP05_03: Cuantización DCT (*round-trip*)

5.13.4 EP05_04 🔴 Implementando la DFT 2D a partir de la definición

Un laboratorio de investigación en **astronomía computacional** recibió, de una misión antigua, un pequeño sensor experimental cuyos datos brutos no pueden ser procesados por bibliotecas modernas de FFT — el entorno de validación está aislado y solo permite operaciones aritméticas básicas. El equipo necesita **reimplementar la Transformada de Fourier Discreta 2D a partir de la propia definición matemática**, célula por célula, para después comparar bit a bit con `np.fft.fft2` en otro entorno.

Este es el ejercicio más conceptual de la lista: no hay atajos. Vas a implementar el doble sumatorio de la Ecuación 5.1 directamente, evidenciando *por qué* existe la FFT — y el costo computacional que evita.

5.13.4.1 📋 Directrices de implementación

1. **Dimensiones:** Leer los enteros M (filas) y N (columnas) de la imagen $f(x, y)$.
2. **Datos:** Leer los valores enteros de $f(x, y)$, fila a fila.
3. **DFT 2D:** Para cada par de frecuencias (u, v) con $u = 0, \ldots, M-1$ y $v = 0, \ldots, N-1$, calcular

$$F(u,v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y)\, e^{-j2\pi\left(\frac{ux}{M} + \frac{vy}{N}\right)}$$

   usando la identidad de Euler $e^{-j\theta} = \cos(\theta) - j\sin(\theta)$ para separar parte real e imaginaria — **no se debe utilizar ninguna función de FFT predefinida**.
4. **Magnitud:** Calcular $|F(u,v)| = \sqrt{\mathrm{Re}(F)^2 + \mathrm{Im}(F)^2}$ y redondear al entero más cercano.
5. **Salida:** Mostrar la matriz de magnitudes redondeadas, $M \times N$, en el mismo orden (sin `fftshift` — el DC permanece en $(0, 0)$).

5.13.4.2 📌 Restricciones computacionales

- **Prohibido usar bibliotecas de FFT:** la implementación debe calcular los sumatorios dobles explícitamente (bucles anidados), aunque sea más lenta.
- **Sin `fftshift`:** la salida mantiene la convención cruda de la DFT, con el componente DC en $F(0, 0)$ (esquina superior izquierda).
- **Redondeo:** la magnitud final debe redondearse al entero más cercano; en los casos de prueba no hay ambigüedad `.5`.
- **Precisión:** pequeños errores de punto flotante (del orden de 10^{-6}) antes del redondeo son esperados y no afectan al resultado entero final.

5.13.4.3 🧠 Fundamentación teórica

Elemento	Significado
$F(0,0)$	Componente DC — suma de todos los píxeles, $F(0,0) = \sum f(x,y)$
Parte real $\mathrm{Re}(F)$	Proyección de la señal sobre cosenos
Parte imaginaria $\mathrm{Im}(F)$	Proyección de la señal sobre senos
Complejidad de esta implementación	$\mathcal{O}((MN)^2)$ — por eso la FFT, con $\mathcal{O}(MN\log(MN))$, resulta indispensable en imágenes reales

5.13.4.4 📦 Especificación de entrada y salida (VPL)

Entrada:

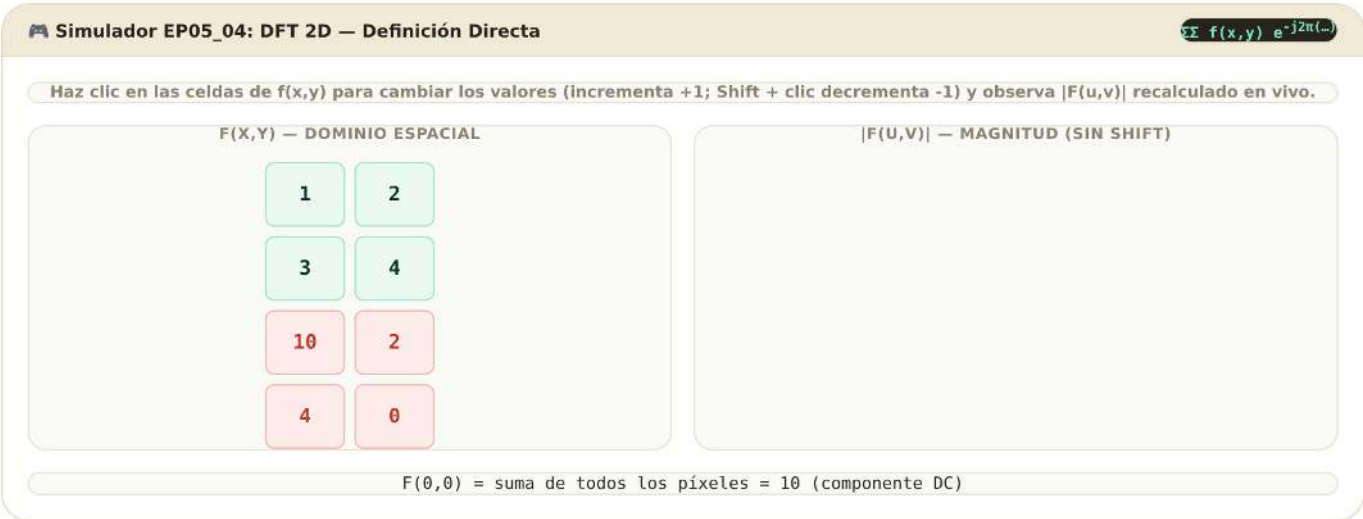
- Línea 1: Entero M.
- Línea 2: Entero N.
- Líneas siguientes: Elementos enteros de $f(x,y)$, M líneas.

Salida:

- Matriz de magnitudes $|F(u,v)|$ redondeadas, $M \times N$, separadas por espacios.

5.13.4.5 📌 Ejemplos

Entrada	Salida	Observación
2 2 1 2 3 4	10 2 4 0	$F(0,0) = 1+2+3+4 = 10$ (DC = suma total). $F(0,1) = (1-2)+(3-4) = -2 \to \lvert F\rvert = 2$. $F(1,0) = (1+2)-(3+4) = -4 \to \lvert F\rvert = 4$. $F(1,1) = (1-2)-(3-4) = 0$.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.es/cap05/sim-ep0504.html

Figura 5.35: Simulador EP05_04: DFT 2D manual

```
%%writefile EP05_04.cpp
// your solution
```

```
Overwriting EP05_04.cpp
```

```
TestSuite("EP05_04.cpp").run()
```

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

5.13.5 EP05_05 🏆 *Pipeline* JPEG Completo: DCT, Cuantización y Reconstrucción

Usted ha sido contratado para crear, desde cero, un **códec JPEG didáctico** en un entorno embebido, sin ninguna biblioteca de imágenes disponible — solo operaciones matemáticas básicas. El cliente quiere entender exactamente dónde se pierde la calidad y dónde se recupera, bloque por bloque. Este es el desafío final del capítulo: integrar **todo** lo estudiado — la DCT-II ortonormal, la cuantización perceptual y la reconstrucción vía IDCT — en un único *pipeline* de extremo a extremo, procesando un bloque $N \times N$ desde el inicio hasta el final, exactamente como el estándar JPEG lo hace internamente, 8×8 píxeles a la vez.

5.13.5.1 📋 Directrices de Implementación

1. **Dimensión del bloque:** Leer el entero N.
2. **Bloque original:** Leer la matriz de píxeles $f(x, y)$, N líneas con N enteros en $[0, 255]$.
3. **Tabla de cuantización:** Leer la matriz Q, $N \times N$ enteros positivos.
4. **Centralización:** Restar 128 de cada píxel: $g(x, y) = f(x, y) - 128$.
5. **DCT-II 2D ortonormal:** Calcular

$$C(u, v) = \alpha(u)\,\alpha(v) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} g(x, y) \cos\left[\frac{\pi(2x+1)u}{2N}\right] \cos\left[\frac{\pi(2y+1)v}{2N}\right]$$

   con $\alpha(0) = \sqrt{1/N}$ y $\alpha(k) = \sqrt{2/N}$ para $k > 0$.
6. **Cuantización:** $\tilde{C}(u, v) = \text{round}(C(u, v)/Q(u, v))$.
7. **Descuantización:** $C'(u, v) = \tilde{C}(u, v) \times Q(u, v)$.
8. **IDCT-II 2D (inversa ortonormal):** Calcular $g'(x, y)$ a partir de $C'(u, v)$ usando la transformada inversa correspondiente (misma base, sumatorio sobre u, v).
9. **Reversión de la centralización y redondeo:** $f'(x, y) = \text{round}(g'(x, y) + 128)$, restringido al intervalo $[0, 255]$ (*clipping*).
10. **Salida:** Mostrar el bloque reconstruido f', $N \times N$, enteros.

5.13.5.2 📌 Restricciones Computacionales

- ***Pipeline* completo obligatorio:** todas las seis etapas (centralizar, DCT, cuantizar, descuantizar, IDCT, revertir) deben implementarse — omitir la cuantización no pasa las pruebas, pues el resultado sería idéntico al original.
- ***Clipping*:** los valores reconstruidos fuera de $[0, 255]$ deben truncarse (0 si es negativo, 255 si es mayor que 255).
- **Redondeo:** tanto en la cuantización como en la reconstrucción final de los píxeles, use redondeo estándar; los casos de prueba evitan ambigüedad `.5`.
- **Base ortonormal:** la normalización $\alpha(u)$ y $\alpha(v)$ debe aplicarse exactamente como se especifica — sin ella, la IDCT no reconstruye correctamente.

5.13.5.3 🧠 Fundamentación Teórica

Etapa	Análoga en el estándar JPEG real	Dónde se pierde la calidad
Centralización	Misma — la DCT asume señal centrada en cero	Sin pérdida

Etapa	Análoga en el estándar JPEG real	Dónde se pierde la calidad
DCT-II	Etapas 3–4 del *pipeline* (Tabla 5.7)	Sin pérdida (transformación exacta y reversible)
Cuantización	Etapa 5 — división por $Q(u,v)$	**Principal fuente de pérdida** — los coeficientes de alta frecuencia se vuelven cero
IDCT	Reconstrucción final	Reconstruye exactamente los coeficientes *cuantizados*, no los originales

5.13.5.4 📦 Especificación de Entrada y Salida (VPL)

Entrada:

- Línea 1: Entero N.
- N líneas siguientes: bloque original $f(x,y)$, enteros en $[0,255]$.
- N líneas siguientes: tabla de cuantización Q, enteros positivos.

Salida:

- Bloque reconstruido $f'(x,y)$, $N \times N$, enteros en $[0,255]$, separados por espacios.

5.13.5.5 📌 Ejemplos

Entrada	Salida	Observación
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	Tras la DCT, cuantización agresiva en las altas frecuencias (valores grandes de Q en la esquina inferior derecha) y reconstrucción vía IDCT, el bloque queda **cercano** al original, pero no idéntico — la diferencia es el costo de la compresión *lossy*.

5.13.5.6 💡 Consejo de Depuración

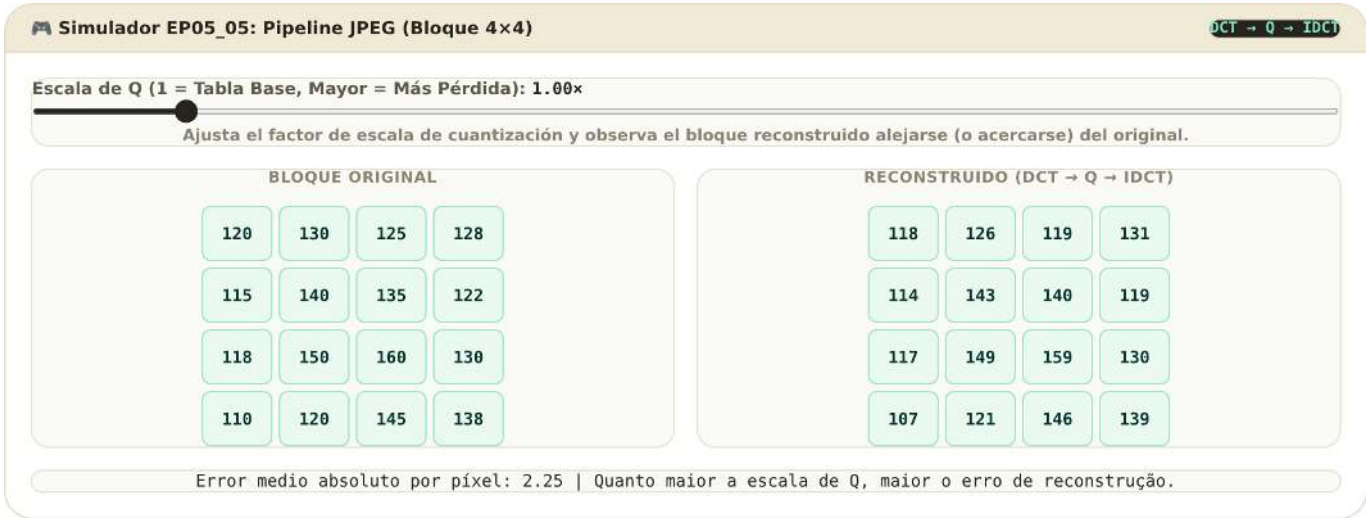
Si el resultado no coincide, verifique en este orden: (1) los coeficientes DCT brutos (antes de la cuantización) — deben reconstruir el original **exactamente** vía IDCT si omite las etapas 6–7; (2) la tabla $\alpha(u)$ — error común es aplicar $\sqrt{2/N}$ también para $u = 0$; (3) el redondeo de la cuantización, que debe ocurrir **antes** de multiplicar de vuelta por Q.

```
%%writefile EP05_05.cpp
// your solution
```

```
Overwriting EP05_05.cpp
```

```
TestSuite("EP05_05.cpp").run()
```

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



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/cpp.es/cap05/sim-ep0505.html

Figura 5.36: Simulador EP05_05: *Pipeline* JPEG completo en bloques

Referencias

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. *En*: 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. *En*: 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.

Apéndice 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.

Apéndice 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.

Apéndice 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 Tabla 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*

Tabla 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.

Apéndice 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.

Apéndice 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.

EL VIAJE DE PDI & VC

SISTEMATIZANDO EL CONOCIMIENTO, DESDE EL PÍXEL HASTA LA INTELIGENCIA

Parte I — Fundamentos de PDI

- 1. Fundamentos y primeros pasos
- 2. Muestreo, cuantización y conectividad
- 3. Operaciones espaciales y filtrado
- 4. Morfología matemática y segmentación
- 5. Transformadas y compresión

Parte II — Visión por Computadora

- 6. Análisis e inspección de documentos
- 7. Clasificación y reconocimiento de patrones
- 8. Comprensión de escenas y segmentación
- 9. Aprendizaje profundo

Apéndices esenciales

- **A:** MCTest
- **B:** Moodle/VPL
- **C:** SEB/Simuladores
- **D:** Gemini/Material didáctico
- **E:** IA en el vpl-ai-feedback
- **F:** Percepción y evaluación

IMAGEN DE ENTRADA (PÍXEL) → **PDI** (PROCESAMIENTO) → **VC** (VISIÓN) → SALIDA INTELIGENTE (SEMÁNTICA)

DOMINA LOS PÍXELES, CONSTRUYE EL FUTURO.

Desde histogramas simples hasta la comprensión compleja de escenas con aprendizaje profundo, este trabajo ofrece una hoja de ruta completa. Transforma imágenes en datos y datos en decisiones inteligentes. Tu capacidad para dar forma a lıa percepción computacional comienza aquí. ¡Mantente a la vanguardia, explora e innova!

Disponible en formato digital (HTML, PDF y Jupyter Notebook), con simuladores y ejercicios, incluidos casos de prueba con VPL y Moodle.

ESCANEA Y ACCEDE A CONTENIDO EXTRA

AUTOR: FRANCISCO DE ASSIS ZAMPIROLLI

https://fzampirolli.github.io/dip-cv/