Academic year
2025-2026

Faculty of Applied Engineering

Digital Image Processing

Text book

Walter Daems

Bachelor of Science in de industriële wetenschappen - elektronica-ICT

1509FTIABV 5-Digital Image Processing
1517FTIDSB 5-Digitale beeldverwerking

This document has been typeset using LaTeX and the `uantwerpendocs` package.
Calculations have been performed using Matlab/Octave and generic programming and script languages (C/C++/Perl/Python).
Graphics have been composed using PGF, TiKZ and InkScape.
All this material has been prepared on a GNU/Linux workstation.

Typesetting of this document was enabled by:

DIP-2024-3.11

Contents

Preface

The scenery

Image processing focuses on a part of the world of signal processing, i.e. systems that treat signals that are images or sequences of images. This allows for adding a number of mathematical tools to our toolbox that are targeted to discover the spatial patterns in the data. From a mathematical point of view, the art of clustering data gets a very prominent place in image processing.

Of course, the profound world of mathematics that we started to discover in the previous course (on digital signal processing) is applicable to images as well. However, we will have to acquire skills known as spatial transformations, mathematical morphology and based on the latter: image segmentation.

The available literature on image processing is vast. Many good books exist on the subject. This work in particular was inspired by some of them you can find in the reference list at the end [Jäh02, GW07, GWE09, Wal08, Mal09]. So, why write another introductory-level text on the very same subject? It allows treating the subjects at a level appropriate to undergraduates in Applied Engineering. There is almost no textbook available that has the correct mix of mathematics and engineering. Writing my own course material also allows elaborating difficult subjects based on teaching experience and updating evolving subjects swiftly.
It avoids also having to purchase multiple expensive books to cover the subject.
Besides, our time is limited, so we cannot go through the entire domain of image processing. Notably concepts like pattern recognition, classification are not treated in this text. The interested reader is referred to the general literature on the topic.

The script

This book is the second in a series of three books on DSP:

1. Digital Signal Processing — Signals and Transformations
2. Digital Image Processing
3. Digital Signal Processing — Signal Processing Systems

The first volume focuses on signals and signal transformations. This second volume focuses on image processing. The third focuses on building digital signal processing systems.

On the language used in this textbook: I tried to write this book from the perspective of a tutor guiding his tutees. I hope you like this style. Where appropriate, I left out some mathematical

derivations in order not to clutter the overall picture. I tried to add as much hints as needed to allow you working your way through the (sometimes difficult) material on your own if you like. The "he-him-his" formulation that has been used is not to emphasize the lack of women in engineering. This wording (instead of the more elaborate he/she, him/her and his/hers) has been chosen to keep this text simple.

A lot of effort has been put into this edition.
This edition has been equipped with a solution book containing the solutions to the exercises. Making exercises is thé way to make sure you understand the theory. Exercises marked with (*) are a bit harder than the standard non-marked exercises. Those marked with (**) are for the enthousiastic reader (to fill any rare rainy days).

Though I try to avoid any errors, human erring is of all times. Do not hesitate to check with me if you find any errors. Even when you think there is an error and there's not, you will gain my appreciation for taking the exercises seriously.

Most of the material in this textbook is my original work. Some of it has been taken from other (free/open) sources. In case you notice a copyright infringement (or a reference that is not clear), please contact me. It is my firm desire to be 100% in line with the copyright legislation. If there happens to be an infringment, I appologize for it even right now. I will do all that is reasonably possible to overcome that issue as soon as possible when it occurs.

The crew

Finally, I would like to thank many people who contributed directly and indirectly to this text. First, a special thanks to my editor, Paul Levrie, for helping me by reviewing this text and supporting me with references, his profound experience, joyful humor and music.
Thanks to Maggy Goossens, Eric Paillet and Jan Steckel for bringing interesting background material to my attention.
Finally, my deepest gratitude goes to my beloved wife and children for enduring me devoting my time to writing this text, instead of spending my time with them.

Any contribution to this work is welcome. You won't get any money for it. I can only offer you to be on my list of favorite people. You can contact me by e-mail to
`walter.daems@uantwerpen.be`.

I hope you enjoy discovering digital image processing!

Walter Daems
Summer 2025
Jordan Green, Norfolk (UK)

Symbol Table

Symbol	Meaning
Number sets	
$\mathbb{N}$	set of natural numbers (positive integer numbers)
$\mathbb{Z}$	set of integer numbers
$\mathbb{Q}$	set of rational numbers
$\mathbb{I}$	set of irrational numbers (real numbers that are not rational)
$\mathbb{R}$	set of real numbers
$\mathbb{C}$	set of complex real numbers
$\mathbb{X}^+$	set $\mathbb{X}$ restricted to positive numbers (not for $\mathbb{C}$)
$\mathbb{X}^-$	set $\mathbb{X}$ restricted to negative numbers (not for $\mathbb{C}$)
$\mathbb{X}_0$	set $\mathbb{X}$ with 0 excluded
Morphological operations	
$x[n] \star y[n]$	convolution of $x[n]$ and $y[n]$
$x[n] \star\!\!\!\star y[n]$	correlation of $x[n]$ and $y[n]$
$A \ominus B$	erosion of binary image A by structuring element B
$A \oplus B$	dilation of binary image A by structuring element B
$A \circ B$	opening of binary image A by structuring element B
$A \bullet B$	closing of binary image A by structuring element B
$f[x,y] \ominus b[x,y]$	erosion of intensity image $f[x,y]$ by structuring element $b[x,y]$
$f[x,y] \oplus b[x,y]$	dilation of intensity image $f[x,y]$ by structuring element $b[x,y]$
$f[x,y] \circ b[x,y]$	opening of intensity image $f[x,y]$ by structuring element $b[x,y]$
$f[x,y] \bullet b[x,y]$	closing of intensity image $f[x,y]$ by structuring element $b[x,y]$
$A \cap B$	intersection of binary images A and B
$A \cup B$	union of binary images A and B
$f[x,y] \wedge g[x,y]$	intersection of intensity images f and g
$f[x,y] \vee g[x,y]$	union of intensity images f and g

Chapter 1

Introduction

For a broader introduction on signal processing, please take a look at the introduction chapter of the previous course on Digital Signal Processing.

1.1 Image processing

Flat images are very natural means to hold complex information. Starting in the stone age, where man started making drawings of his surroundings, or his imagination, over early technical drawings (in the Middle or Far East), to written documents and books: man stored both visual as abstract information on flat surfaces. During a very long period, these information surfaces were hand-made and processed just by viewing them and reasoning on them.

However, many advancements have opened up the nature of things around us. Many technology enhancements allowed for making better images: photographic plates, powerful lenses, fast shutters, a.o. have opened up extreme scales (from microscopy to telescopy) and extreme observation speeds (below microseconds). This has helped our understanding of physics (both on a nano as on a universe scale) to a great extent.

It is only very recently (since a few decades) that the generation and processing of images using digital techniques have become feasible: making CAD drawings, digital image recording, processing images, computer vision. The silicon revolution has opened up a vast amount of pathways to continue using surfaces (flat images) as information source and storage medium, and even to start measuring/processing/generating 3D volumes of information in a digital way.

1.2 Why digital image processing?

Processing images in the analog domain, has been standard practice for many decades. What are the advantages of processing images in the digital domain?
The key fact is that we are able to implement a perfect memory element. This opens the path to

- programmable circuits, ranging from microprocessor-like products (allowing different algorithms to run on the same hardware) to reconfigurable hardware

(FPGAs, a.o.)[1];

- adaptive systems (that change behavior based on the data they process);
- spatial analysis (as images are no longer a concatenated line-scanning signal);
- processing signals using signal transformations (e.g., 2D-FFT convolution);

The latter allows performing spatial and motion analysis techniques that are very hard to implement in the analog domain.

Because of the continual improvements in image sensors, computing hardware and image generators, digital image has become the de facto standard way to process images. The main application areas are:

- particle physics,
- astronomy,
- medical imaging,
- industrial control applications,
- multimedia.

The end result is that while DIP used to be a specialist's graduate course, the introductory level has become a standard undergraduate course...and you are — whether you like it or not — taking it right now. Don't be mistaken, the state-of-the-art of DIP is still on the graduate level. It requires knowledge of more sophisticated mathematical concepts from the range of *functional analysis* (a.k.a. topological vector spaces) and *statistical process theory*.

1.3 Conclusion

Digital Image Processing beats analog image processing for very good reasons: programmability, performance that is (almost) infeasible in the analog domain based on spatial techniques, leading to improved compression and analysis opportunities.

However, concluding that analog signal processing will disappear in the future, is one step too far: in the area of image sensing (from photons to pixel intensities) and image generation (from pixel intensities to photons), analog devices, such as lenses and analog electronics (e.g., amplifiers) will remain to be (inevitably) the preferred technology. High-speed analog and digital will go hand-in-hand in the mixed-signal systems of the future. This also holds for digital image processing.

[1]Note that programmability allows reducing the time-to-market (the second major urge in electronics, next to scaling) for a dedicated signal processing related application. This often is referred to as "rapid prototyping".

Chapter 2

Discrete Calculus

In this chapter, you will learn about:

- discrete first-order derivatives and gradients
- discrete second-order derivatives and Laplacians
- how they can be implemented using convolution/correlation
- the common variants that are used

After having read/studied this chapter, you are expected to be able to

- calculate discrete first-order derivatives and gradients
- calculate discrete second-order derivatives and Laplacians
- explain why the different variants exist and use them

2.1 Introduction

First and higher-order derivatives of real functions of a real variable are well known concepts from calculus.

Consider a function $f : \mathbb{R} \to \mathbb{R} : x \mapsto f(x)$. To keep things simple, let's assume the function and it's first and second derivative to be continuous.

The derivative of our function f can be defined as:

$$\frac{\mathrm{d}f}{\mathrm{d}x}(x) = \lim_{\Delta x \to 0} \frac{f(x + \Delta x) - f(x)}{\Delta x}$$

The value of the derivative gives us the rate of increase of the function. If it is positive, the function increases with x; if it is negative, it decreases with x.

Likewise, using recursion, we can define the second-order derivative as:

$$\frac{\mathrm{d}^2 f}{\mathrm{d}x^2}(x) = \lim_{\Delta x \to 0} \frac{\frac{\mathrm{d}f}{\mathrm{d}x}(x + \Delta x) - \frac{\mathrm{d}f}{\mathrm{d}x}(x)}{\Delta x}$$

The value of the second-order derivative corresponds to the rate of increase of the first-order derivative. Seen from a different perspective, it tells us something about the curvature of the

function. If the second-order derivative is positive, the function is *convex* (i.e. it has the lip curvature of a happy face); if it is negative, it is *concave* (i.e. it has the lip curvature of a sad face).

Of course, one can continue the recursion for higher-order derivatives.

However, note that these concepts rely on the fact that one can get infinitely close to the particular x-value of interest, i.e. one can make Δx infinitesimally small and still $f(x + \Delta x)$ will be properly defined.

When considering functions on discrete domains, e.g. the set of integers, this property no longer holds. However, the rate of increase and the curvature are still appropriate characteristics of such functions.

In the next sections, we will provide new definitions for (first- and higher-order) derivatives of such functions. We will restrict ourselves to functions on an integer domain.

2.2 The discrete first-order derivative

Consider a discrete function f:

$$f : \mathbb{Z} \to \mathbb{R} : x \mapsto f[x]$$

Hence, in the rest of this chapter x is assumed to be an integer.

2.2.1 Asymmetric variant

Let's define the (asymmetric) first-order derivative of $f[x]$ as:

$$\frac{\mathrm{d}f}{\mathrm{d}x}[x] = \lim_{n \to 1} \frac{f[x+n] - f[x]}{n} = f[x+1] - f[x] \tag{2.1}$$

Note that we didn't set $n = 0$, as we then obtain a division of zero by zero. In addition, note that the definition above is asymmetrical: it is right-biased. Indeed, why not use the definition $\frac{\mathrm{d}f}{\mathrm{d}x}[x] = f[x] - f[x-1]$?

To overcome this issue, we provide a second (symmetric) variant, that is more commonly used.

2.2.2 Symmetric variant

Let's define the symmetric first-order derivative of $f[x]$ as:

$$\frac{\mathrm{d}f}{\mathrm{d}x}[x] = \lim_{n \to 1} \frac{f[x+n] - f[x-n]}{2n} = \frac{f[x+1] - f[x-1]}{2} \tag{2.2}$$

Exercises

Exercise 2.2.2-1: Calculate the values of the asymmetrical derivative for the one-dimensional function f. The value for $x = 0$ has been circled.

$$f[x] = \Big[7,\ \ 2,\ \ -3,\ \ 0,\ \ \textcircled{-2},\ \ -9,\ \ -6,\ \ -8,\ \ -7,\ \ -6\Big]$$

Assume the function f to be zero outside the indicated domain range.

Exercise 2.2.2-2: Calculate the values of the symmetrical derivative for the one-dimensional function f. The value for $x = 0$ has been circled.

$$f[x] = \Big[7,\ \ 2,\ \ -3,\ \ 0,\ \ \textcircled{-2},\ \ -9,\ \ -6,\ \ -8,\ \ -7,\ \ -6\Big]$$

Assume the function f to be zero outside the indicated domain range.

2.3 The discrete second-order derivative

Consider a discrete function f:

$$f : \mathbb{Z} \to \mathbb{R} : x \mapsto f[x]$$

Again, note that x represents an integer.

2.3.1 Asymmetric variant

Recursion on (2.1) leads to the following definition for the second-order derivative:

$$\begin{aligned}\frac{\mathrm{d}^2 f}{\mathrm{d}x^2}[x] &= \lim_{n\to 1} \frac{\frac{\mathrm{d}f}{\mathrm{d}x}[x+n] - \frac{\mathrm{d}f}{\mathrm{d}x}[x]}{n} \\ &= \frac{\mathrm{d}f}{\mathrm{d}x}[x+1] - \frac{\mathrm{d}f}{\mathrm{d}x}[x] \\ &= (f[x+2] - f[x+1]) - (f[x+1] - f[x]) \\ &= f[x+2] - 2f[x+1] + f[x] \end{aligned} \tag{2.3}$$

Note that, again, the result is biased to the right. To overcome this, a more commonly used symmetric variant is provided.

2.3.2 Symmetric variant

Let's define the second-order derivative of $f[x]$ as:

$$\frac{\mathrm{d}^2 f}{\mathrm{d}x^2}[x] = f[x+1] - 2f[x] + f[x-1] \tag{2.4}$$

Remarks

- Note that this definition did not result from recursion on (2.2), but was obtained by shifting the asymmetric variant one unit to the left.
- Check that a symmetric recursion on (2.2) does not yield a proper second-order derivative concept.
- Because of the fact that the commonly used definition of the first-order derivative of (2.2) and the second-order derivative (2.4) are not consistent, one cannot use theorems that apply in the real calculus case, e.g. Taylor series expansions.
  When one combines (2.1) and (2.3) the theory is consistent. You might want to try and prove that recursion on these finite differences yields a coefficient structure that is very related to Pascal's triangle.

Exercises

Exercise 2.3.2-1: Calculate the values of the asymmetrical second-order derivative for the one-dimensional function f. The value for $x = 0$ has been circled.

$$f[x] = \Big[7, \quad 2, \quad -3, \quad 0, \quad \textcircled{-2}, \quad -9, \quad -6, \quad -8, \quad -7, \quad -6\Big]$$

Assume the function f to be zero outside the indicated domain range.

Exercise 2.3.2-2: Calculate the values of the symmetrical second-order derivative for the one-dimensional function f. The value for $x = 0$ has been circled.

$$f[x] = \Big[7, \quad 2, \quad -3, \quad 0, \quad \textcircled{-2}, \quad -9, \quad -6, \quad -8, \quad -7, \quad -6\Big]$$

Assume the function f to be zero outside the indicated domain range.

2.4 Discrete correlation and convolution

2.4.1 Correlation

Consider two functions f and g:

$$f : \mathbb{Z} \to \mathbb{R} : x \mapsto f[x]$$
$$g : \mathbb{Z} \to \mathbb{R} : x \mapsto g[x]$$

The correlation of these two functions is a discrete function itself, denoted by $f \star g$ and defined as:

$$f \star g : \mathbb{Z} \to \mathbb{R} : x \mapsto \sum_{n=-\infty}^{+\infty} f[n] \cdot g[n - x]$$

The correlation of a function with itself, is called *auto-correlation*, and defined as:

$$f \star f : \mathbb{Z} \to \mathbb{R} : x \mapsto \sum_{n=-\infty}^{+\infty} f[n] \cdot f[n - x]$$

One can prove that $f \star f$ attains its maximum value for $x = 0$. This is the one of the basic properties on which *matched filtering* is based.

2.4.2 Convolution

Consider two functions f and g:

$$f : \mathbb{Z} \to \mathbb{R} : x \mapsto f[x]$$
$$g : \mathbb{Z} \to \mathbb{R} : x \mapsto g[x]$$

The convolution of these two functions is a discrete function itself, denoted by $f \star g$ and defined as:

$$f \star g : \mathbb{Z} \to \mathbb{R} : x \mapsto \sum_{n=-\infty}^{+\infty} f[n] \cdot g[x-n]$$

Remarks Convolution and correlation are closely related. If we define the reflection h^r of a function h as

$$h^r : \mathbb{Z} \to \mathbb{R} : x \mapsto h[-x]$$

then the relationship can be found to be:

$$f \star g = f \star g^r$$

This is the second basic property on which *matched filtering* relies. To learn more about this technique, we refer to other courses on signal processing.

Exercises

Exercise 2.4.2-1: Consider the following two signals:

$$f[x] = \Big[6.8,\ -4.9,\ \textcircled{6.3},\ -5.1,\ 8.6,\ -3.0,\ -6.1,\ -5.0,\ 2.3,\ -0.5\Big]$$
$$g[x] = \Big[-3.0,\ \textcircled{6.6},\ 1.7,\ 1.0\Big]$$

The values for $x = 0$ have been circled.
Calculate the correlation of the two signals: $(f \star g)[x]$.

Exercise 2.4.2-2: Consider the following two signals:

$$f[x] = \Big[8.3,\ -4.3,\ 5.1,\ \textcircled{5.1},\ -2.4,\ 1.4,\ -8.5,\ -8.9,\ 0.6,\ 5.6\Big]$$
$$g[x] = \Big[\textcircled{8.7},\ -7.4,\ 1.4,\ -0.6000\Big]$$

The values for $x = 0$ have been circled.
Calculate the convolution of the two signals: $(f \star g)[x]$.

2.5 Discrete derivative as correlation / convolution

Writing the discrete derivative as a correlation or convolution is straightforward.

2.5.1 First-order derivative

Defining γ_r as:

$$\gamma_r[x] = \begin{bmatrix} -1/2 & \textcircled{0} & 1/2 \end{bmatrix}$$

allows writing the first-order discrete derivative as:

$$\frac{\mathrm{d}f}{\mathrm{d}x}[x] = (f \star\!\!\star \gamma_r)[x]$$

We commonly call the function γ_r the derivative *correlation kernel*.

Likewise, defining γ_c as:

$$\gamma_c[x] = \begin{bmatrix} 1/2 & \textcircled{0} & -1/2 \end{bmatrix}$$

allows writing the first-order discrete derivative as:

$$\frac{\mathrm{d}f}{\mathrm{d}x}[x] = (f \star \gamma_c)[x]$$

We commonly call the function γ_c the derivative *convolution kernel*.

Remarks For simplicity reasons (to save computations) often the kernels are defined to be:

$$\gamma_r[x] = \begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$$
$$\gamma_c[x] = \begin{bmatrix} 1 & 0 & -1 \end{bmatrix}$$

2.5.2 Second-order derivative

Defining λ as:

$$\lambda[x] = \begin{bmatrix} 1 & \textcircled{-2} & 1 \end{bmatrix}$$

allows writing the second-order discrete derivative as

$$\frac{\mathrm{d}^2 f}{\mathrm{d}x^2}[x] = (f \star\!\!\star \lambda)[x]$$

or in view of the symmetry of the kernel $\lambda[x]$ as

$$\frac{\mathrm{d}^2 f}{\mathrm{d}x^2}[x] = (f \star \lambda)[x]$$

We commonly call the function λ the second-order derivative *correlation or convolution kernel*.

2.6 Discrete gradient

Consider a function f of two discrete variables x and y:

$$f : \mathbb{Z}^2 \to \mathbb{R} : [x, y] \mapsto f[x, y]$$

As in the continuous case, we can define partial derivatives, w.r.t. one variable, considering the other one to be constant. Restricting ourselves to the symmetrical case:

$$\frac{f}{x}[x,y] = \frac{f[x+1,y] - f[x-1,y]}{2}$$
$$\frac{f}{y}[x,y] = \frac{f[x,y+1] - f[x,y-1]}{2}$$

This allows us to define a discrete gradient $\vec{\nabla} f$ in the same way as can be done for the continuous case:

$$\vec{\nabla} f[x,y] = \frac{f}{x}[x,y]\vec{e}_x + \frac{f}{y}[x,y]\vec{e}_y$$

The equation is most easily remembered if one uses the nabla notation. The nabla vector $\vec{\nabla}$ is to be considered an operator (or operator *vector*) that transforms a two-dimensional function into a vector:

$$\vec{\nabla} = \frac{-}{x}\vec{e}_x + \frac{-}{y}\vec{e}_y$$

The fact that the operator is a vector and yields a vector, justifies writing the vector arrow above the nabla symbol (∇).

Remarks

- The gradient as described above can be expressed in Cartesian coordinates:

$$\begin{aligned}\vec{\nabla} f[x,y] &= \frac{f[x+1,y] - f[x-1,y]}{2}\vec{e}_x + \frac{f[x,y+1] - f[x,y-1]}{2}\vec{e}_y \\ &= g_x[x,y]\vec{e}_x + g_y[x,y]\vec{e}_y\end{aligned}$$

  with

$$g_x[x,y] = \frac{f[x+1,y] - f[x-1,y]}{2}$$
$$g_y[x,y] = \frac{f[x,y+1] - f[x,y-1]}{2}$$

  It can also be expressed in polar coordinates:

$$\vec{\nabla} f[x,y] = M[x,y] \angle \phi[x,y]$$

  with

$$M[x,y] = \sqrt{g_x^2[x,y] + g_y^2[x,y]}$$
$$\phi[x,y] = \text{Arctan2}\left(g_y[x,y], g_x[x,y]\right)$$

2.7 Discrete Laplacian

Consider a function f of two discrete variables x and y:

$$f : \mathbb{Z}^2 \to \mathbb{R} : [x,y] \mapsto f[x,y]$$

Using the second-order partial derivatives

$$\frac{^2f}{x^2}[x,y] = f[x+1,y] - 2f[x,y] + f[x-1,y]$$
$$\frac{^2f}{y^2}[x,y] = f[x,y+1] - 2f[x,y] + f[x,y-1]$$

we can define the discrete Laplacian in a similar fashion as for the continuous case:

$$\nabla^2 f(x,y) = \frac{^2f}{x^2}[x,y] + \frac{^2f}{y^2}[x,y]$$

Using the nabla notation, we define ∇^2 to be:

$$\begin{aligned}\nabla^2 &= \left\langle \vec{\nabla}, \vec{\nabla} \right\rangle \\ &= \left\langle \left(\frac{}{x}\vec{e}_x + \frac{}{y}\vec{e}_y \right), \left(\frac{}{x}\vec{e}_x + \frac{}{y}\vec{e}_y \right) \right\rangle\end{aligned}$$

Because we assume $\vec{e}_x \perp \vec{e}_y$ and $\left\|\vec{e}_x\right\| = \left\|\vec{e}_y\right\| = 1$ this reduces to:

$$\nabla^2 = \frac{^2}{x^2} + \frac{^2}{y^2}$$

Therefore, applying this operator to a function $f[x,y]$ yields:

$$\begin{aligned}\nabla^2 f[x,y] &= \left(\frac{^2}{x^2} + \frac{^2}{y^2} \right) f[x,y] \\ &= \frac{^2f}{x^2}[x,y] + \frac{^2f}{y^2}[x,y]\end{aligned}$$

2.8 Gradients and Laplacians applied to images

2.8.1 Axis convention

The two-dimensional discrete functions that we will consider are often (grayscale) images. These images can be considered to be matrices with pixel intensity values. Consider as an example the image f:

$$f = \begin{bmatrix} \textcircled{10} & 23 & 48 & 223 \\ 189 & 200 & 3 & 0 \\ 145 & 76 & 110 & 255 \end{bmatrix}$$

It is very common to attach the x and y axis to this image in a very specific way, such that the row index corresponds to the x-axis and the column index corresponds to the y-axis. This leads to:

$f[x,y]$ y

x

(10)	23	48	223
189	200	3	0
145	76	110	255

The origin $[x,y] = [0,0]$ is located a the top left pixel and has been indicated with a circle.

2.8.2 Correlation and convolution in two dimensions

The notions correlation and convolution can easily be extended to two dimensions.

Consider two functions f and g:

$$f : \mathbb{Z}^2 \to \mathbb{R} : [x,y] \mapsto f[x,y]$$
$$g : \mathbb{Z}^2 \to \mathbb{R} : [x,y] \mapsto g[x,y]$$

The correlation of f and g is a discrete function itself, denoted by $f \star g$ and defined as:

$$f \star g : \mathbb{Z}^2 \to \mathbb{R} : [x,y] \mapsto \sum_{n=-\infty}^{+\infty} \sum_{m=\infty}^{+\infty} f[n,m] \cdot g[n-x, m-y]$$

The convolution of f and g is a discrete function itself, denoted by $f \star g$ and defined as:

$$f \star g : \mathbb{Z}^2 \to \mathbb{R} : [x,y] \mapsto \sum_{n=-\infty}^{+\infty} \sum_{m=\infty}^{+\infty} f[n,m] \cdot g[x-n, y-m]$$

These simple definitions allow writing gradients and Laplacians as a correlation (or convolution).

Gradient Consider the following *gradient* kernels (with the origins $[x,y] = [0,0]$ indicated by the circle):

$$\gamma_x = \begin{bmatrix} -\frac{1}{2} \\ \textcircled{0} \\ \frac{1}{2} \end{bmatrix} \qquad \gamma_y = \begin{bmatrix} -\frac{1}{2} & \textcircled{0} & \frac{1}{2} \end{bmatrix}$$

This allows writing the gradient $\vec{\nabla} f$ as:

$$\vec{\nabla} f = (f \star \gamma_x)\vec{e}_x + (f \star \gamma_y)\vec{e}_y$$

Laplacian Consider the following *Laplacian* kernel (with the origin $[x,y] = [0,0]$ indicated by the circle):

$$\lambda = \begin{bmatrix} 0 & 1 & 0 \\ 1 & \textcircled{-4} & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

This allows writing the Laplacian $\nabla^2 f$ as:

$$\nabla^2 f = (f \star \lambda)$$

Challenge Write the definitions above using convolution instead of correlation.

2.8.3 Watch out for the border patrol!

In the setting of image processing, one can consider the gradient to be an operator that translates an image into two component images (one for the x-component and one for the y-component of the gradient). Likewise, one can consider the Laplacian to be an operator that translates an image into a new image.

However, a new problem arises due to the fact that the nature of images implies that they are domain limited: how do you calculate gradients or Laplacians at the border pixel locations?

An obvious solution is not calculating gradients and Laplacians at the border pixel locations. In that scenario, the resulting image is shrunk by two pixels in both height and width.

If the shrinkage is not an acceptable phenomenon, one needs to make an assumption about the pixel values outside the image in order to be able to calculate the gradient or the Laplacian at the border pixels.

Some common choices are:

- Consider the pixels outside the image to have an intensity value of 0. This is called *zero padding* the image.
- Consider the pixels outside the image to have a constant intensity value. This is called *constant padding* the image.
- Consider the pixels outside the image to take on the intensity value of the image pixel that is closest. This is called *border replication*.
- Consider the pixels outside the image to be a reflection of the image pixels as if one puts a mirror along the border and looks in that mirror. This is called *border mirroring*.
- Consider the image to be periodic.

Exercises

Exercise 2.8.3-1: Consider the image $f[x, y]$ below:

$f[x, y]$ y x

0.5	1.0	0.2	0.8	0.6	0.5	0.9	0.7
0.9	0.8	0.8	0.9	0.4	0.5	0.8	0.3
0.2	0.7	0.7	0.4	0.7	0.7	1.0	0.3
0.4	0.4	0.9	0.6	0.4	1.0	0.1	0.8
1.0	1.0	0.4	1.0	0.8	0.4	0.4	0.7

Calculate the discrete gradient $\vec{\nabla} f$ of this image, assuming border pixel replication.

Exercise 2.8.3-2: Consider the image $f[x, y]$ below:

$f[x,y]$ y

x

0.6	0.3	0.2	0.9	0.5	0.7	0.9	0.3
0.7	0.8	0.9	0.6	1.0	0.6	0.1	0.5
0.1	0.9	0.2	0.5	0.1	0.7	0.1	0.2
0.4	0.3	0.6	0.9	1.0	0.4	1.0	0.3
0.5	0.8	0.4	0.4	0.2	0.7	0.7	0.3

Calculate the discrete gradient $\vec{\nabla} f$ of this image, assuming zero padding.

Exercise 2.8.3-3: Consider the image $f[x,y]$ below:

$f[x,y]$ y

x

0.5	1.0	0.2	0.8	0.6	0.5	0.9	0.7
0.9	0.8	0.8	0.9	0.4	0.5	0.8	0.3
0.2	0.7	0.7	0.4	0.7	0.7	1.0	0.3
0.4	0.4	0.9	0.6	0.4	1.0	0.1	0.8
1.0	1.0	0.4	1.0	0.8	0.4	0.4	0.7

Calculate the discrete Laplacian $\nabla^2 f$ of this image assuming border pixel replication.

Exercise 2.8.3-4: Consider the image $f[x,y]$ below:

$f[x,y]$ y

x

0.6	0.3	0.2	0.9	0.5	0.7	0.9	0.3
0.7	0.8	0.9	0.6	1.0	0.6	0.1	0.5
0.1	0.9	0.2	0.5	0.1	0.7	0.1	0.2
0.4	0.3	0.6	0.9	1.0	0.4	1.0	0.3
0.5	0.8	0.4	0.4	0.2	0.7	0.7	0.3

Calculate the discrete Laplacian $\nabla^2 f$ of this image assuming zero padding.

2.9 Variants

2.9.1 Gradients

Many variant gradient definitions have been proposed. We list these variants using the corresponding correlation kernels. Therefore, if the gradient kernels listed for the x and y directions are labeled γ_x and γ_y, the gradient can be calculated as:

$$\vec{\nabla} f = (f \star \gamma_x)\vec{e}_x + (f \star \gamma_y)\vec{e}_y$$

To make comparison easy, one can find the kernels we already have seen above in Figure 2.1a and Figure 2.1b. For the latter, note that we discarded the factor $1/2$ for the simple reason that most often we are only interested in relative differences in gradients throughout an image. The scaling is therefore irrelevant and only costs extra clock cycles on a digital signal processor.

Prewitt To avoid influence of noise, Prewitt proposed to provide some averaging by using information of the adjacent rows and columns. The definition of the corresponding kernels can be found in Figure 2.1c.

Sobel Sobel elaborated on Prewitt's idea, but added an emphasis for the center row and column. you can find its definition in Figure 2.1d.

Roberts Sometimes the gradients of interest are directed along the diagonals of a two-dimensional figure (or image). Therefore, Roberts rotated the common discrete gradient kernels over $\pi/4$. One can find the corresponding kernels in Figure 2.1e (the asymmetrical version) and Figure 2.1f (the symmetrical version).
In this case, the gradient can be decomposed as:

$$\vec{\nabla} f = (f \star \gamma)\frac{\vec{e}_y + \vec{e}_x}{2} + (f \star \gamma)\frac{\vec{e}_y - \vec{e}_x}{2}$$

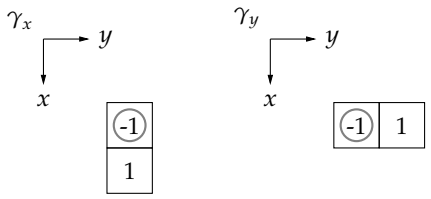
Diagonal Prewitt Even Prewitt's idea has been adapted to diagonal gradients. One can find the kernel in Figure 2.1g.

Diagonal Sobel And finally, also Sobel's idea has been adapted to diagonal gradients. The resulting kernel can be found in Figure 2.1h.

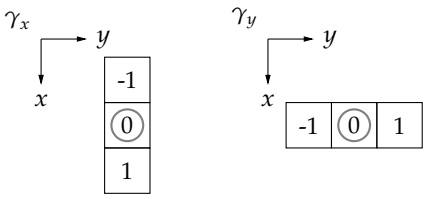
2.9.2 Laplacians

The range of variant Laplacian definitions is not so large when compared to the variant gradient definitions. Again, we specify the Laplacians using their correlation kernels. Therefore, if the Laplacian kernel listed is labeled λ, the Laplacian can be calculated as:

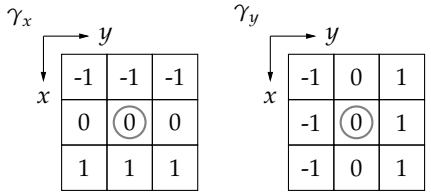
$$\nabla^2 f = (f \star \lambda)$$



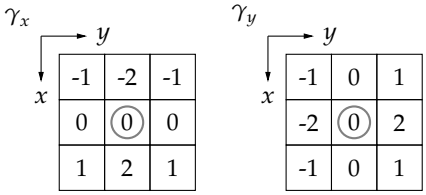
(a) Asymmetrical gradient



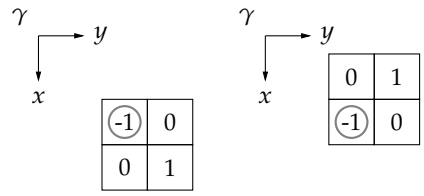
(b) Symmetrical gradient



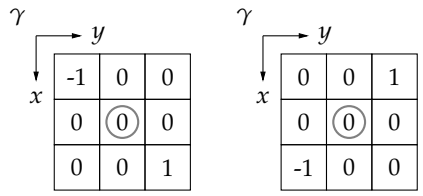
(c) Prewitt gradient



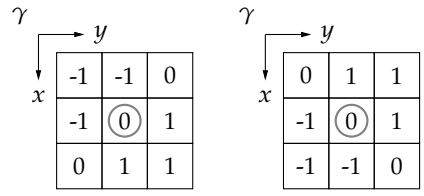
(d) Sobel gradient



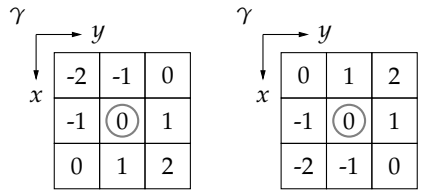
(e) Asymmetrical Roberts gradient



(f) Symmetrical Roberts gradient



(g) Diagonal Prewitt



(h) Diagonal Sobel

Figure 2.1: Common gradient kernels (the centers of the kernels have been indicated by a circle)

In essence, Prewitt's idea of noise reduction by averaging has been applied to the original (4-)Laplacian, leading to the 8-Laplacian. One can find these kernels in Figure 2.2a and Figure 2.2b.

(a) Four Laplacian (b) Eight Laplacian

Figure 2.2: Common Laplacian kernels (the centers of the kernels have been indicated by a circle); the negative Laplacians have been drawn on the right, because often these are (erroneously) taken as the Laplacian kernel.

Remarks

- Note that the sum of all kernel entries always is equal to zero. This is required to make sure that the derivatives/gradients/Laplacians are not dependent on the absolute function values.
- Strangely the negative Laplacian kernel $-\lambda$ is often mistaken for the positive one λ. The source of this mistake is unclear.

2.10 Calculating gradients/Laplacians of images using MATLAB/OCTAVE

In MATLAB or OCTAVE a kernel is easily defined as a matrix. E.g., consider the Sobel gradient kernel, applied to an arbitrary image.

```
image = rand(20);                          % create a random image
sobelx = [ 1 2 1; 0 0 0; -1 -2 -1 ];
sobely = sobelx';
gradx = imfilter( image, sobelx, 'replicate', 'same' );
grady = imfilter( image, sobely, 'replicate', 'same' );
```

The optional parameter '`replicate`' causes border pixel replication. The optional parameter '`same`' was specified to the `imfilter` function to ensure that the result has the same size as the original.

Check the documentation of '`imfilter`' to learn all the ins and outs:

```
help imfilter
```

Alternatively, one might use the '`conv2`' function as well.

Exercises

Repeat the exercises of this chapter, but this time using MATLAB/OCTAVE.

Chapter 3

Image Processing Systems

In this chapter, you will learn:

- what a generic image processing system looks like
- the five different classes of image processing systems and their specific nature.

After having read/studied this chapter, you are expected to be able to

- sketch the overall structure of an image processing system, and identify each of its components
- understand the function of each of those components
- classify a given system into one of the five standard classes.

3.1 Introduction

An image processing system is a specific kind of signal processing system, in the sense that it measures images as physical quantities and operates on digital representations of those images.

Just like every specific kind of signal processing system, image processing systems exploit the very specific nature of the signals at hand. The specific nature of images allows to apply specific mathematical techniques, like geometry transformations, morphological analysis, object recognition, a.s.o. Some of these techniques also appear in ordinary signal processing (e.g., object recognition becomes feature recognition), but in images they appear in their most beloved habitat. Prepare yourself for a challenging journey in the world of image processing systems.

3.2 Generic structure of image processing systems

Consider the generic structure of an image processing depicted in Figure 3.1 on the following page. Let's take a moment to browse through its individual elements.

We start at the left-hand side of the picture. A light source with power P illuminates an object O (e.g., a tree) that reflects part of the incident light energy through a lense system L onto an

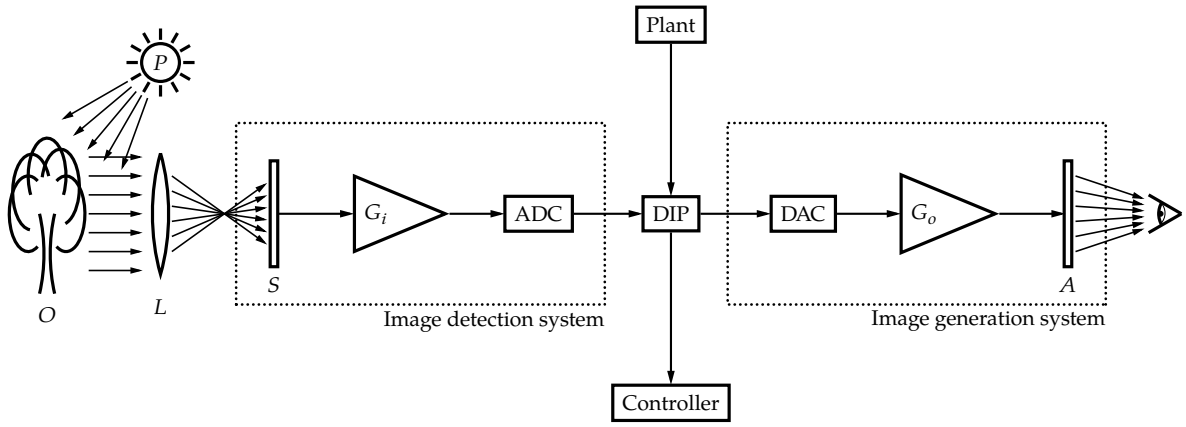


Figure 3.1: Generic structure of an image processing system

imaging sensor S. This detector samples the incident light flux and converts those samples to electrical signals using an amplifier G_i. These electrical signals subsequently are quantized by an *ADC* to an image that can be treated by the Digital Image Processor (DIP). Very often the chain from sensor to ADC is integrated into a stand-alone image detection system (e.g., a CMOS-camera).

At this moment we can take different routes (that we will elaborate on in the next section).

Either the objective is to manipulate the incoming images to produce new images and display them to a user. In that case, we proceed further to the right, converting our digital images again to the analog domain using a DAC converter and an amplifier G_o to drive the image actuator A that displays the resulting image as seen by our eye. Very often these last parts (starting from the DAC) are integrated into a stand-alone image generation system (e.g., an LCD monitor).

Either the objective is to have the DIP draw some conclusions from the incoming images and pass them on to a controller that can take appropriate action (e.g., sending a positive identification for the species oak to the Tree/GPS database when an oak has been identified).

A third route starts from other sensors that measure quantities in a plant (at the top of the picture) and use those measurements to generate new images, either from scratch, or from existing images coming in through the front end.

Browsing through this picture teaches us that understanding digital image processing — next to image processing algorithms — requires studying light, illumination, reflection, image sensors, signal conversions and image detectors.

3.3 Classification of image processing systems

Very often image processing systems are classified depending on their nature. We distinguish five classes:

1. *Imaging systems:* systems that process incoming sensor data with the objective to generate images that are suited for human interpretation (e.g., CAT-scanners);

2. *Image processing systems:* systems that process incoming images to create new (enhanced) images (e.g., removing salt-and-pepper noise from old photographs);
3. *Image analysis systems:* systems that analyze some specific properties or characteristics of the image (e.g., determining the biperietal diameter of a foetus' head);
4. *Image recognition/interpretation:* systems that analyze images to such a level that a predefinied set of meaningful structures can be isolated (located) in the image (the system *understands* the image; e.g., detecting metallic particles on a conveyor belt carrying food);
5. *Image synthesis systems:* systems that generate new or augment existing images, starting from a specific set of data (e.g., adding route information on the pictures taken with a smart phone).

Regarding the in- and outflow of information the following table summarizes the nature of these systems well.

System	In	Out
Imaging system	Sensor data	Images
Image processing system	Images	Images
Image analysis system	Images	Data
Image recognition system	Images	Recognized objects
Image synthesis system	Data	Images

Remarks

- Computer vision is the term used for a specific kind of image recognition/interpretation system in which no human intervention is required. The computer draws conclusions and acts upon them autonomously.
- One could consider imaging systems and image synthesis systems to be flavors of the same kind. We make the difference to stress the fact that image synthesis systems not necessarily strive to generate images that correspond to a physical reality, wheras imaging systems do.
- One might argue as to whether image analysis systems are more intelligent than image recognition systems. Actually, it depends on the application at hand. Sometimes object recognition is required before one can deduce properties/characteristics of those objects. Sometimes one can deduce properties/characteristics (e.g., a multispot light level measurement) without object recognition. We will consistently use the term *image analysis system* only for the latter kind.

Example - Imaging system

Figure 3.2 shows a sketch of a CAT (computed axial tomography) scanner. A rotatable ring (6), fixed by a number of bearings (8a, 8b, 8c), is driven by an electric motor (11) on a thoothed bar (22). The ring contains an X-ray source (12) that can irradiate an object in the scanner (7) under various angles. The signals picked up by the detector (13) are treated by a digital signal processing system (17-21) in order to generate planar image slices of the object under observation.
This is an example of an imaging system: sensor data in, images out.

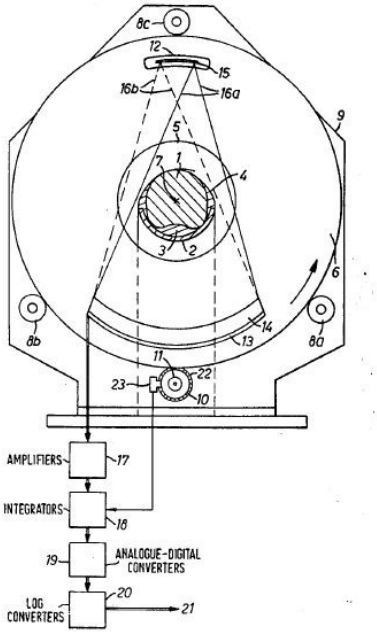


Figure 3.2: Illustration of a computed axial tomography (CAT) scan system [*image source: patent application US4115698 by Dr. Godfrey Newbold Hounsfield*]

Example - Image processing system

Consider the setup of Figure 3.3. On the left-thand side we start with a photograph containing a significant amount of salt-and-pepper noise. Applying a median filter can remove this type of noise to a great extent, as can been seen on the right.
This is an example of an image processing system: images in, images out.

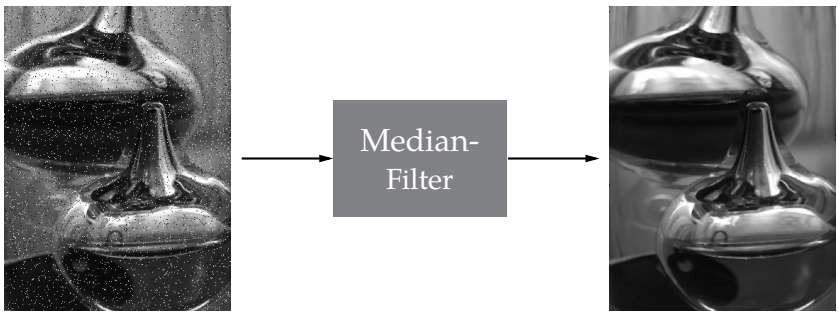
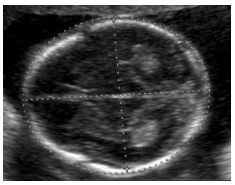


Figure 3.3: Illustration of an image processing system that filters salt-and-pepper noise from a noisy image [*image source: Wikimedia commons, user Marko Meza*]

Example - Image analysis system

Consider the setup of Figure 3.4. An obstetric digital image processing unit analyzes the ultrasound image of a foetus'head. It analyzes several parameters, such as the biperietal diameter (BPD, i.e. the distance between the ears), the front-to-back diameter (FBD) and the circumference of the head (HC).
This is an example of an image analysis system: images in, data out.



Metric	Value
BPD	5.45 cm
FBD	7.26 cm
HC	20.78 cm

Figure 3.4: Illustration of an image analysis system that determines several parameters related to a foetus' head [*image source: http://www.ultrasoundpeadia.com*]

Example - image recognition/interpretation system

Consider the setup of Figure 3.5. On the left we see an X-ray image of food passing by on a conveyor belt. The digital image processor recognizes 4 foreign objects on the belt and alerts the operator of the food processing plant.
This is an example of an image recognition/interpretation system: images in, recognized (located) objects out.

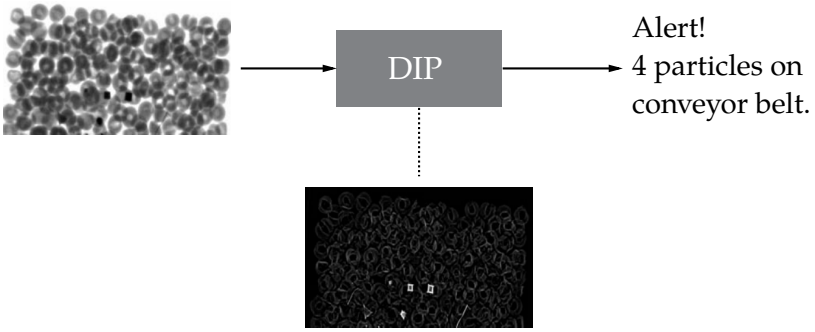


Figure 3.5: Illustration of an image recognition system that recognizes foreign (metallic) objects on a conveyor belt [*image source: Fraunhofer Institute for Integrated Circuits, Department EZRT*]

Example - image synthesis system

Consider the setup of Figure 3.6. A smart-phone camera takes images of the environment. Route information is fetched from a server in the cloud. The augmented reality application on the smart phone superimposes the route information on the pictures taken with the smart phone.
This is an example of an image synthesis system: data in, images out.

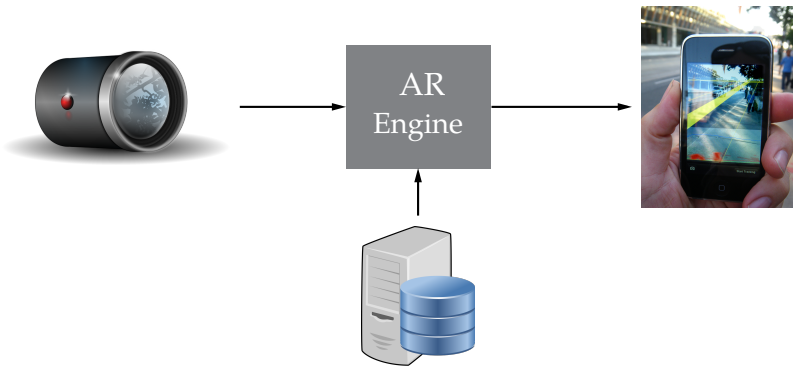


Figure 3.6: Illustration of an augmented reality application (image synthesis sytem) that adds route information to an image taken with a smart phone [*image source: Glogger*]

Chapter 4

Sampling and Resampling

In this chapter, you will learn:

- How and why images are sampled;
- How and why images are resampled;
- How and why images are interpolated.

After having read/studied this chapter, you are expected to be able to

- explain why we sample/resample/interpolate images,
- apply a specific image sampling strategy,scolor,
- apply a specific resampling strategy,
- derive interpolation equations for 0^{th}, 1^{st}, and 3^{rd} order interpolation,
- apply these interpolation equations on images,
- explain and apply geometric transformations,
- explain what image registration means.

4.1 Introduction

The world surrounding us is — on a macroscopic level — analog in nature. The physical quantities have a continuous value range. So do the objects that we can 'measure' with a camera. Those objects have dimensions of continuous lengths.

For reasons of convenience (see section section 4.2.2 on page 28) we discretize these continuous quantities to digital quantities. You probably have seen this matter in a basic course on signal processing. Figure 4.1, illustrates the options for discretizing a physical quantity: (1) discretizing its domain, and (2) discretizing its range.

In this chapter we will concentrate on discretizing the domain of images. This is different from discretizing time. However, the underlying principles are very similar.

The discetization can take place at different stages of the image processing chain. It can take place in the image detector. It also can take place at a later stage as part of the Digital Image Processing (DIP) itself.

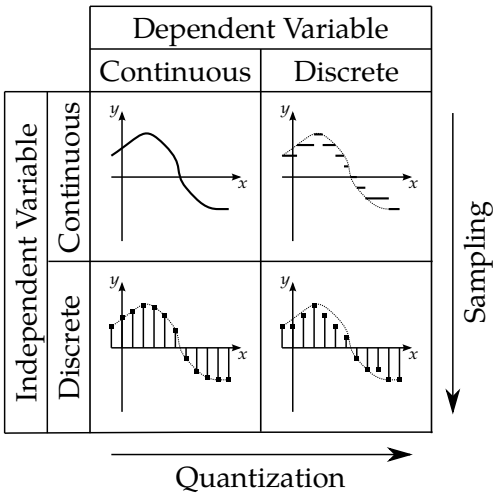


Figure 4.1: Sampling and quantization render an analog signal (top left) into a digital signal (bottom right)

In the former case, we *sample* a continuous image, to obtain an image that consists of a grid of pixels. This is a physical process. In the latter case, we *resample* a discrete-domain image, to obtain a new image with a new grid of pixels. The need for this resampling is dictated by our desire to look at images from a different point of view, requiring *domain transformations*. This is a mathematical process.

Therefore, you will not be surprised to find out that the major sections in this chapter are:

- sampling of images
- resampling
- domain transformations

4.2 Sampling of images

Images are a specific subcategory of signals with a two-dimensional domain. For now, we'll restrict ourselves to grayscale images. Later on we will add color to the picture.

4.2.1 Definition

To illustrate the nature of sampling images, we will use an example.

Consider a 2-dimensional photon flux density function $f(x, y)$, also denoted as *intensity function*:

$$f(x,y) = \sin(8\pi x - 2) \cdot e^{(-20y-0.5)^2} + e^{30(-x-0.5)^2-(y-0.25)^2-(x-0.5)(y-0.25)} \tag{4.0}$$

Don't be stupefied by the complexity of the function above. It is just *an example*.

The function has been graphed on the detector's domain $(x, y) = ([0 \ldots 1], [0 \ldots 1])$ in Figure 4.2.

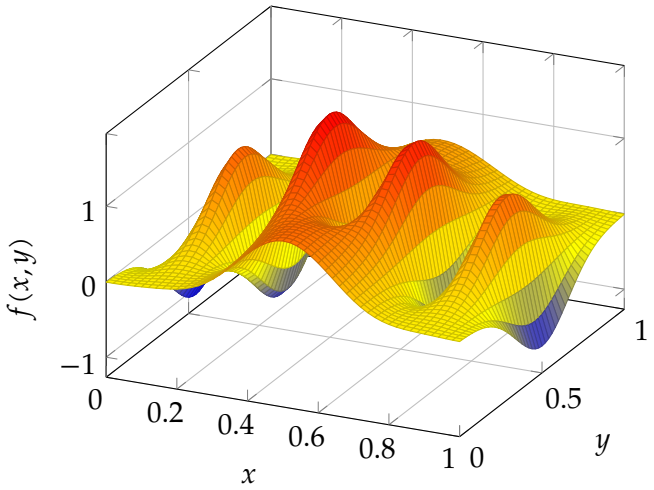


Figure 4.2: Graph of the intensity function of (4.2.1)

The intensity function reprents the photon flux density reflecting from an object and hitting our planar image detector. To allow for a mathematical treatment, we equipped the planar detector with an x- and y-axis.

Now, let's assume that the minimum of the given flux density corresponds to black and the maximum of the given flux density corresponds to white. Then we can reduce our intensity function by displaying the continuous gray-scale image that hits the detector. This image is shown in Figure 4.3 on the following page. It is important to realize that any value for x and y can be chosen within the domain of detector.

The show stopper in working with these continuous intensity functions is that light consists of particles: photons. Therefore, the chance that a photon hits a particular location spot on, is zero. The conclusion is simple: we cannot measure the photon flux density pointwise. We need a small but finite area to integrate the photons, such that we can measure the photon flux density. To this end, we will divide the detector area in small cells.

In selecting these cells we have many options. The simplest by far (and most used) is to use a grid of squares, as indicated in Figure 4.4 on the next page. The idea is to represent the intensity of every cell by a single intensity value, centered in the cell.

This intensity value can be obtained

1. by averaging the flux density in the cell,
2. by taking the flux density at the center of the cell.

The former will probably be used in a physical detector, the latter is easier mathematically. It has been illustrated in Figure 4.5 on page 27.

You can see the corresponding image, where every square has been filled with the intensity of its center, in Figure 4.6 on page 27. This is a sampled intensity image. I guess this is not new for you.

Now, we're ready to define *image sampling*.

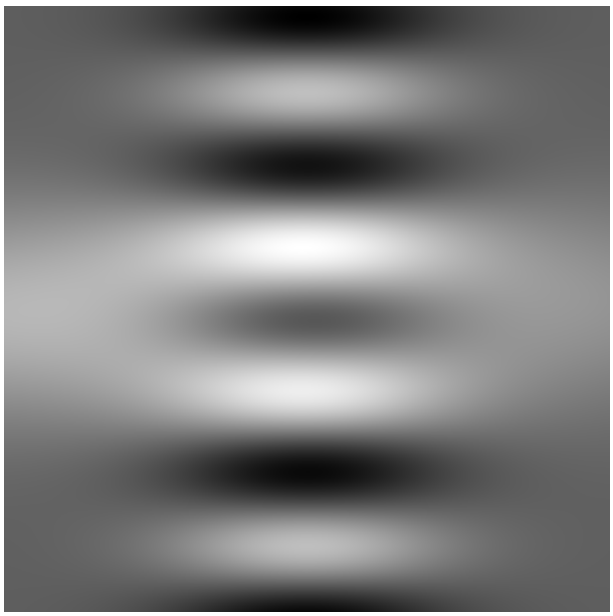


Figure 4.3: Continuous domain image corresponding to the intensity function of (4.2.1)

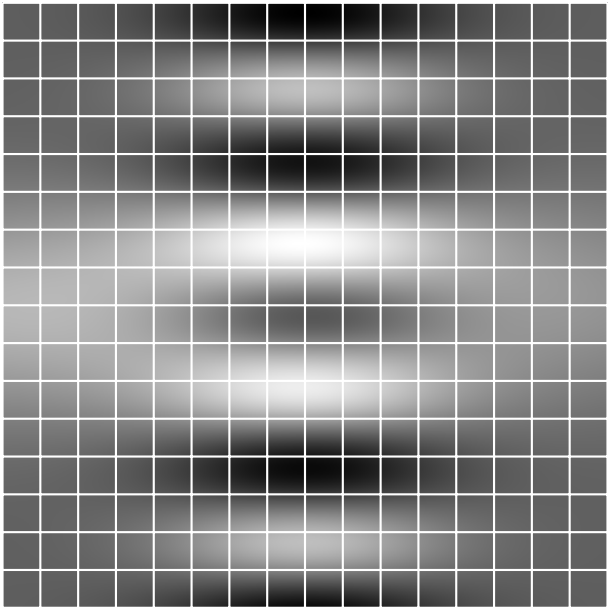


Figure 4.4: A grid of squares superimposed on the image of Figure 4.3

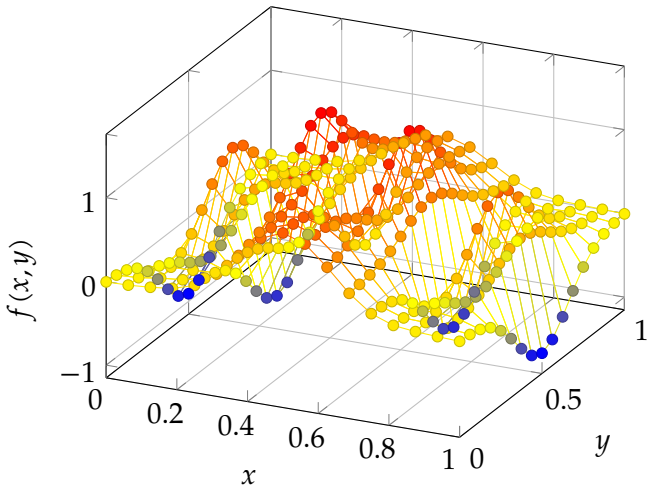


Figure 4.5: Graph of the discretized intensity function of (4.2.1)

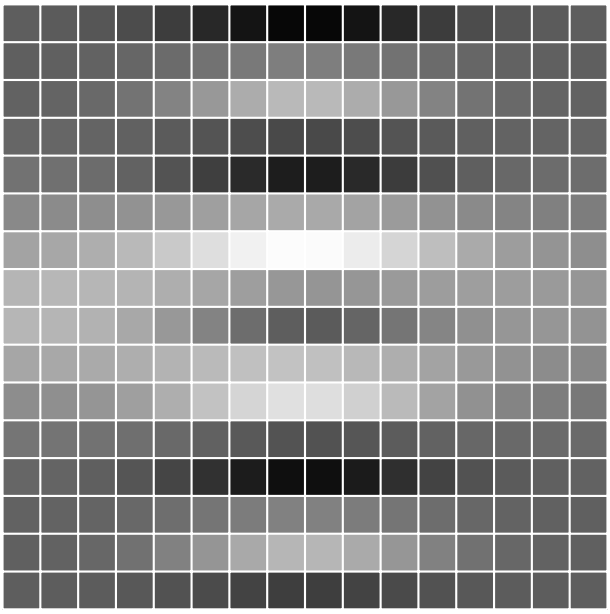


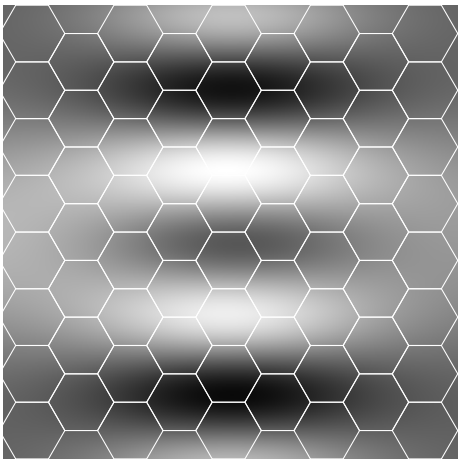
Figure 4.6: The sampled image of Figure 4.3 on the preceding page where every cell has been filled with its center intensity

Definition: image sampling Sampling an image on a rectangular grid with pitch Δ means replacing a continuous intensity function $f(x, y)$ by a discrete-domain intensity function defined as:

$$f[i, j] = f(i\Delta, j\Delta)$$

Remarks

- There are alternatives to sampling on rectangular grid (e.g., sampling using a hexagonal grid (a honeycomb structure) as indicated below). However in most cases we use a rectangular grid, or convert to a rectangular grid by interpolation.



- Note the subtle change in the definition above, replacing round brackets to square brackets to indicate the discrete domain nature of the new function.

4.2.2 Why do we have to sample images?

Why do we need to sample images? The major reason has been elaborated on in the previous section: light comes in particles, photons, and therefore, we need a minimal surface to integrate those photons in order to estimate the photon flux density.

However, the major secondary reason is that treating analytically defined functions (required for continuous intensity functions) is difficult.

Discrete mathematics are more tractable for a digital computer. The number formats that are supported natively all are discrete: integers, but also floats/doubles!

By discretizing images, we transform them to the domain where digital computers are at their best: number crunching.

4.2.3 Terminology

In order to describe the sampling of images, we need a new vocabulary. Whereas for one-dimensional signals the sampling process can be described with the unquestioned terms *sampling period* and *sampling frequency*, the vocabulary related to sampling images is less straightforward and often abused/questioned.

The central key word is *resolution*. As we will see, it comes in many flavors.

Resolution The resolution R of an image is obtained by counting the number of horizontal pixels multiplied by the number of vertical pixels. Often the number is left uncalculated. Therefore, both the following statements are valid: a picture in full HD1080

1. has a resolution of $1920 \times 1080\,\text{px}$,
2. has a resolution of about 2 Mpx.

The unit of resolution is the pixel.

Pixel resolution / pixel density The pixel resolution/density of an image, an image sensor (camera) or an image actuator (screen) is the average number of pixels per unit of length. It is obtained by measuring the distance between two neighboring pixels (the pitch of the sampling grid) and taking its reciprocal. The value is most conveniently found by dividing the width or height of an image by the number of pixels along that dimension. E.g., a full HD1080 screen with resolution $1920 \times 1080\,\text{px}$ with a screen diagonal of 15 in has a pixel resolution of:

$$R_p = \frac{\sqrt{1920^2 + 1080^2}\text{px}}{15\,\text{in}} = 146.86\,\text{ppi}$$

in which we conveniently used the Pythagorean theorem.

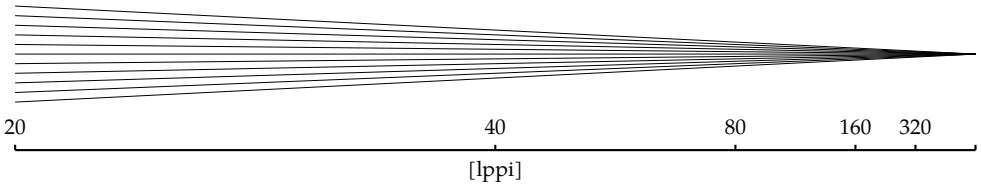
The unit of pixel resolution is the *pixel per meter* (ppm)[1] or the *pixel per inch* (ppi). In the printing industry, a more common unit is the *dots per inch* (dpi).

Spatial resolution The spatial resolution of an image, an image sensor (camera) or an image actuator (screen) indicates the size of the smallest feature that can be recognized in the image. The basic feature to assess the discrimination of the finest detail in an image is a *line pair*, i.e. two adjacent parallel lines, one white, the other black. The highest number of line pairs per unit of length that we can still tell apart is the spatial resolution.

The unit of spatial resolution is the *line pairs per meter* (lppm) or *line pairs per inch* (lppi).

In theory, a detector with a pixel resolution of 100 ppi could reach a spatial resolution of 50 lppi. However the detector (through lense aberrations and false illumination) will blur the detected image, probably leading to a spatial resolution that is smaller than the theoretical upper bound.

Spatial resolution is often checked using a radial test line pattern, like the one below. Can you determine the spatial resolution of the printed version of this course text?



[1]The shorthand ppm is often used to denote parts per million. Make sure no confusion can arise.

Remarks

- Many standards ask not to speak about resolution as the total number of pixels in an image, insisting that the word resolution should have a 1/m unit, or a 1/s unit.
  Given the fact that many manufacturers and practitioners use the term resolution in the broad sense, we chose to use the specific term pixel resolution or pixel density when speaking about a resolution in the strict sense.
- Note that many of the common units in the digital image industry are no SI units. Very common are: dpi (dots per inch), ppi (pixel per inch), lppi (line pairs per inch).
- Note that the unit point (pt) cannot be used interchangeably with the unit pixel (px). A point has a specific meaning in the printing industry: 1 pt = 1/72.27 in = 0.351 460 mm.
- Be aware that the spatial resolution is also limited by the capabilities of your eye.

4.2.4 Representation

Given the fact that most images are rectangular and are sampled using an equidistant square grid, it seems logical to represent the discretized intensity function as an array of numbers: a table.

	f[x,y]					
$x \backslash y$	0	1	2	3	4	5
0	3.61	3.64	-0.03	0.51	3.79	4.39
1	-2.00	-0.63	0.57	-6.01	2.17	3.09
2	-0.97	-6.69	4.05	-4.66	-0.53	4.01
3	-6.84	-6.75	0.11	1.82	0.78	0.69

In software, tables are commonly stored as 0-indexed 2D-arrays. E.g., in C++ using the boost library:

```
boost::multi_array<double, 3> a(boost::extents[4][6]);
a[0][0] = 3.61;
a[1][2] = 0.57;
```

In mathematical terms, arrays are matrices. We therefore also use the following notation to describe images:

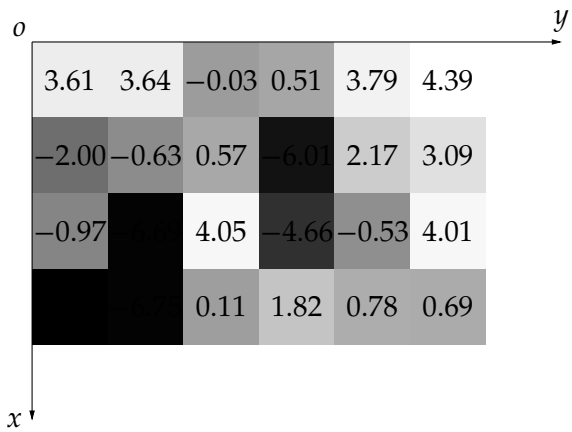
$$f[x,y] = \left[A_{x,y}\right] = \begin{bmatrix} 3.61 & 3.64 & -0.03 & 0.51 & 3.79 & 4.39 \\ -2.00 & -0.63 & 0.57 & -6.01 & 2.17 & 3.09 \\ -0.97 & -6.69 & 4.05 & -4.66 & -0.53 & 4.01 \\ -6.84 & -6.75 & 0.11 & 1.82 & 0.78 & 0.69 \end{bmatrix}$$

In MATLAB/OCTAVE, images are stored as 1-indexed matrices.

```
a = zeros(4,6);
a(1,1) = 3.61;
a(2,3) = 0.57;
```

Note that in the matrix representation the x-index is the row index and the y-index is the column index.

This is the reason why in most image processing literature the axes are commonly rotated over $-90°$, such that the x-axis points downwards and the y-axis points to the right. We will also use this convention in this text as illustrated below.



4.2.5 Memory requirements

Calculating the memory requirements of a picture is quite easy. We only spend some time on it, to stress that images are memory intensive.

We will discuss quantization in the next chapter, so for now let's assume that we need 1 B/px to store the gray-scale intensity of a gray scale image and 3 B/px to store the intensity of a full color image.

This brings us to the following table, illustrating the memory-hungry nature of gray scale images at typical resolutions. The last column shows the data rate for gray-scale motion pictures. For full-color imagery, multiply the numbers by three. The table also introduces you to the many acronyms in the display industry.

Format	Resolution [px]	Gray scale size [MB]	25 fps data rate [MB/s]
VGA	640 ×480	0.3072	7.6800
NTSC	720 ×480	0.3456	8.6400
WVGA	800 ×480	0.3840	9.6000
PAL	768 ×576	0.4423	11.0592
SVGA	800 ×600	0.4800	12.0000
XGA	1 024 ×768	0.7864	19.6608
HD720	1 280 ×720	0.9216	23.0400
WXGA	1 280 ×800	1.0240	25.6000

continued on next page

continued from previous page

Format	Resolution [px]	Gray scale size [MB]	25 fps data rate [MB/s]
SXGA	1 280 ×1 024	1.3107	32.7680
SXGA+	1 400 ×1 050	1.4700	36.7500
WSXGA	1 680 ×1 050	1.7640	44.1000
UXGA	1 600 ×1 200	1.9200	48.0000
HD1080	1 920 ×1 080	2.0736	51.8400
WUXGA	1 920 ×1 200	2.3040	57.6000
QXGA	2 048 ×1 536	3.1457	78.6432
WQXGA	2 560 ×1 600	4.0960	102.4000
QSXGA	2 560 ×2 048	5.2428	131.0720
UHD	3 840 ×2 160	8.2944	207.3600
4K	4 096 ×2 160	8.8473	221.1840

Imagine storing an uncompressed full-color 4K-movie on your brand-new 4 TB harddrive. It's done in less than 2 hours. We definitely need some compression techniques for digital video!

4.3 Resampling

4.3.1 The need for resampling

Given the fact that we have a sampled version of the image we want to consider, one might wonder: why do we need resampling?

Many valid reasons can be thought of. A few of them:

- It may be that the sampling frequency (or de pixel resolution) that has been used is too low or too high for the application at hand. To illustrate this, consider having to display an HD1080 image on an old SXGA monitor, or a picture taken with a VGA surveillance camera full screen on an HD1080 monitor.
- Memory or throughput limitations may impose an upper bound on the resolution a system can treat. E.g., the graphical chip of your smart phone will probably not be capable to perform real-time histogram equalization on a 4K movie.
- The grid that was used may have been inappopriate. To illustrate this, consider having to print a photograph on a color ink jet printer, rotated with an angle of 30°.

Though we will focus on two-dimensional signals in this chapter, we will start with one-dimensional illustrations first. This eases the treatment.

4.3.2 Overview

Depending on how the amount of samples changes, we can consider

- down sampling: reducing the amount of samples, or
- up sampling: increasing the amount of samples.

Depending on whether (some of) the existing samples are preserved or not, we can consider

- On-grid resampling: keeping the existing samples
- Off-grid resampling: not keeping the existing samples

We will use the latter subdivision as the major one in the following treatment.

4.3.3 On-grid resampling

On-grid resampling resamples the signal maintaining (some of) the orginal data points.

When considering on-grid down sampling, we speak about *decimation*, when considering on-grid up sampling, we speak about *interpolation*. Later on, when discussing off-grid resampling, we will also encounter interpolation. Therefore, we distinguish between the two by calling the latter wide-sense interpolation. The interpolation we are considering here, is called *strict-sense interpolation*, but very often the strict-sense is omitted.

4.3.3.1 On-grid resampling in one dimension

To take a smooth start, we will first consider one-dimensional on-grid resampling. Remember decimation means sampling down, interpolation means sampling up.

Definition: one-dimensional decimation The signal x_{dec} is a decimation by a factor of k (with $k \in \mathbb{N}_0$) of the signal x_{org} iff

$$x_{dec}[n] = x_{org}[kn], \forall n \in \mathbb{Z}$$

Simply stated: keep 1 sample, throw away $k-1$, keep 1 sample, throw away $k-1$, a.s.o.

Definition: one-dimensional interpolation (strict-sense interpolation) The signal x_{int} is an interpolation by a factor of k (with $k \in \mathbb{N}_0$) of x_{org} iff

$$x_{int}[kn] = x_{org}[n], \forall n \in \mathbb{Z} \tag{4.1}$$

Simply stated: keep 1 sample, insert $k-1$ new samples, keep 1 sample, insert $k-1$ new ones, a.s.o.

Remarks

- Note that $k = 1$ results in a rather trivial case where interpolation and decimation meet in the 'no-operation' of on-grid resampling.
- You might wonder if (4.1) tells you something about how to pick the intermediate samples? Well... it doesn't. We'll have to deal with this later on.

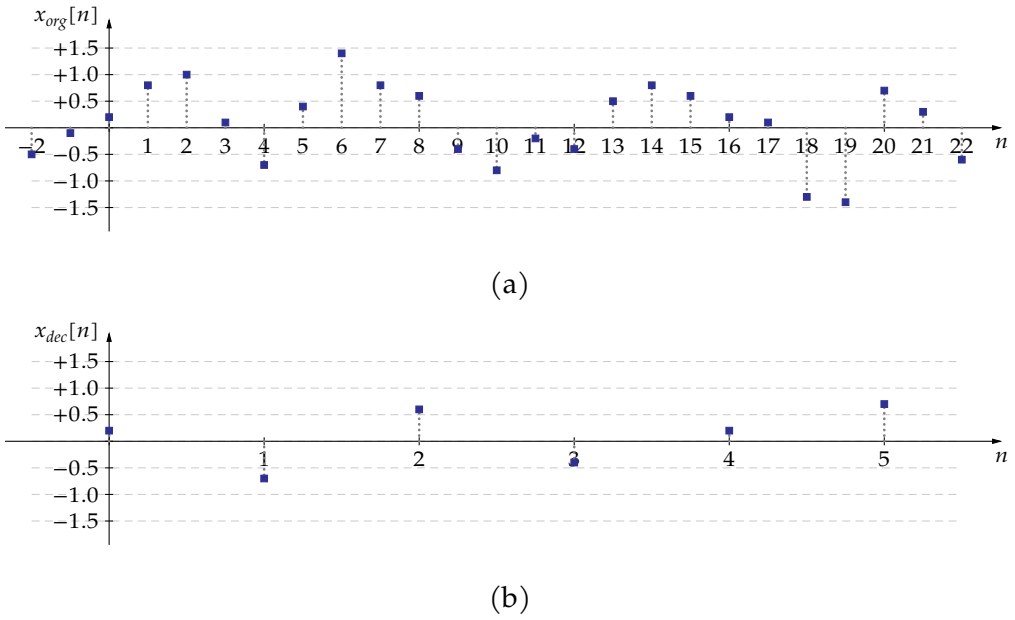


Figure 4.7: Illustration of the principle of decimation: (a) original signal, (b) resulting signal after decimation by a factor of 4

Example – decimation

Consider a one-dimensional signal x_{org}

$$x_{org}[n] = \ldots, -0.5, -0.1, \underbrace{0.2}_{n=0}, 0.8, 1.0, 0.1, -0.7, 0.4, 1.4, 0.8, 0.6, -0.4, -0.8, -0.2,$$
$$-0.4, 0.5, 0.8, 0.6, 0.2, 0.1, -1.3, -1.4, 0.7, 0.3, -0.6, \ldots$$

Decimation of this signal by a factor of 4 results in:

$$x_{dec}[n] = \ldots, \underbrace{0.2}_{n=0}, -0.7, 0.6, -0.4, 0.2, 0.7, \ldots$$

The process has been illustrated in Figure 4.7.

Example – interpolation

Consider a one-dimensional signal x_{org}

$$x_{org} = \ldots, \underbrace{0.2}_{n=0}, 0.1, 1.4, -0.4, -0.4, 0.6, -1.3, 0.3, \ldots$$

Interpolation of this signal by a factor of 3 results in:

$$x_{int} = \ldots, -0.5, -0.1, \underbrace{0.2}_{n=0}, 0.8, 1.0, 0.1, -0.7, 0.4, 1.4, 0.8, 0.6, -0.4, -0.8, -0.2,$$
$$-0.4, 0.5, 0.8, 0.6, 0.2, 0.1, -1.3, -1.4, 0.7, 0.3, -0.6, \ldots$$

The process has been illustrated in Figure 4.8 on the next page.

4.3.3.2 On-grid resampling in two dimensions

The two-dimensional versions are trivial as soon as you understand the one-dimensional versions of section 4.3.3.1 on the preceding page.

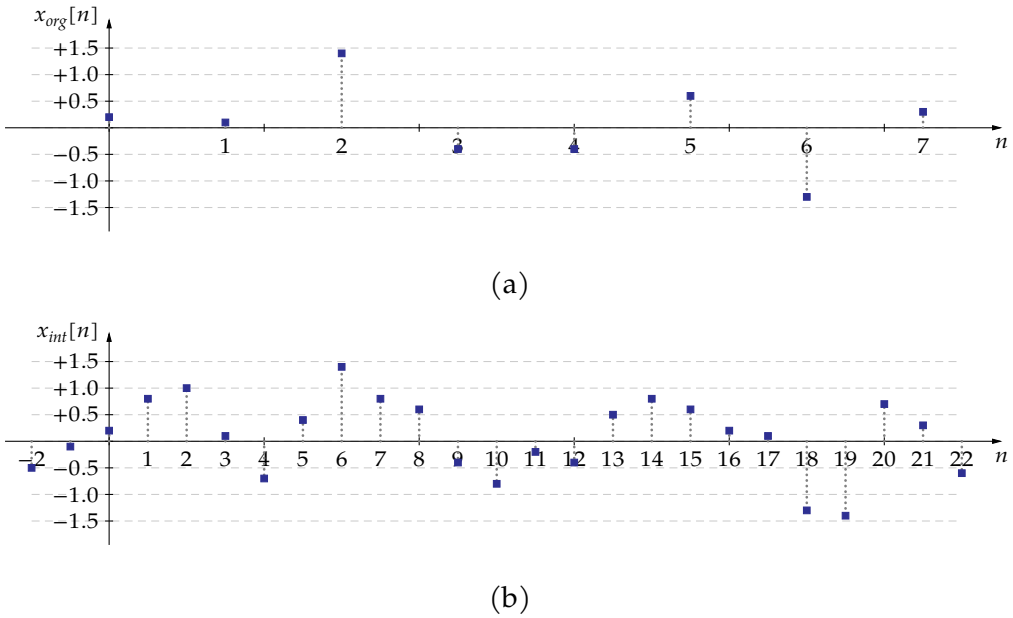


Figure 4.8: Illustration of the principe of interpolation: (a) original signal, (b) resulting signal after interpolation by a factor of 3

Definition: two-dimensional decimation The discrete intensity function g_{dec} is a decimation by factors (k_m, k_n) (with $k_m, k_n \in \mathbb{N}_0$) of g_{org} iff

$$g_{dec}[m,n] = g_{org}[k_m m, k_n n], \forall m,n \in \mathbb{Z}$$

Definition: two-dimensional interpolation (strict-sense interpolation) The discrete intensity function g_{int} is an interpolation by factors (k_m, k_n) (with $k_m, k_n \in \mathbb{N}_0$) of g_{org} iff

$$g_{int}[k_m m, k_n n] = g_{org}[m,n], \forall n \in \mathbb{Z} \tag{4.2}$$

Remarks

- Note that k_m does not *have to* equal k_n.
- Note that $k_m = k_n = 1$ results in a rather trivial case where interpolation and decimation meet in the 'no-operation' of on-grid resampling.
- Again: you might wonder if (4.2) tells you something about how to pick the intermediate samples? Well... it doesn't. We'll have to deal with this later on.

Example – decimation

The principle has been illustrated in Figure 4.9 on the next page.

Example – interpolation

The principle has been illustrated in Figure 4.10 on the following page.

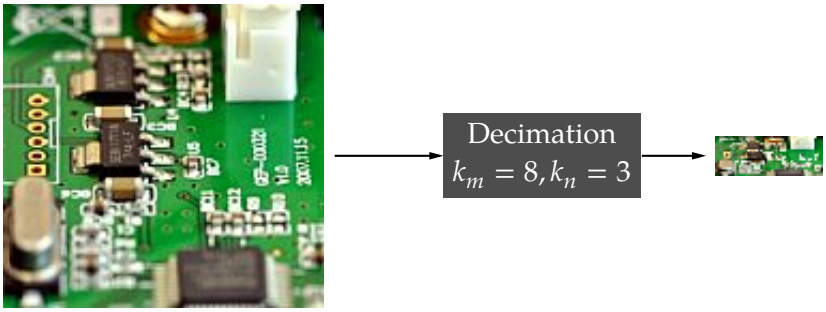


Figure 4.9: Illustration of the principle of two-dimensional on-grid decimation

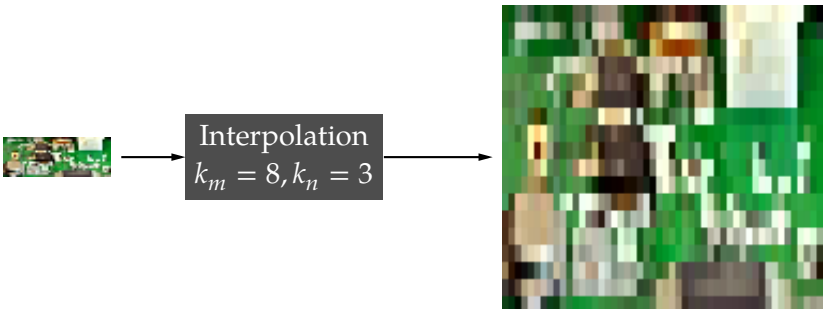


Figure 4.10: Illustration of the principle of two-dimensional on-grid interpolation

4.3.4 Off-grid resampling (wide-sense interpolation) in one dimension

When discussing on-grid interpolation, we still left one key question unanswered: how do you find the intermediate samples? The very same question gets even more to the point when considering off-grid resampling where we no longer keep any of the original datapoints. The fact whether we're sampling up or sampling down actually is no longer relevant.

Again, to take a smooth start, we will first consider one-dimensional interpolation.

For that case, the issue has been illustrated in Figure 4.11 on the next page. Subfigure (a) shows a discrete-time signal that we'd like to resample. The desired off-grid new sample locations have been indicated in subfigure (b). The key question is: "how do we find sensible values for the samples at these new locations?" It would have been nice, if we knew the original continuous-time function (see subfigure (c)) that was sampled in the first place. However, we don't know that function. The key in solving our problem will be trying to make a model of that original function based on the available samples. In this way, resampling is reduced to resampling that model.

Actually, two-dimensional interpolation is no more complex than the above. We need to find a function $f_m(x, y)$ that can act as a model of the original continuous-domain image.

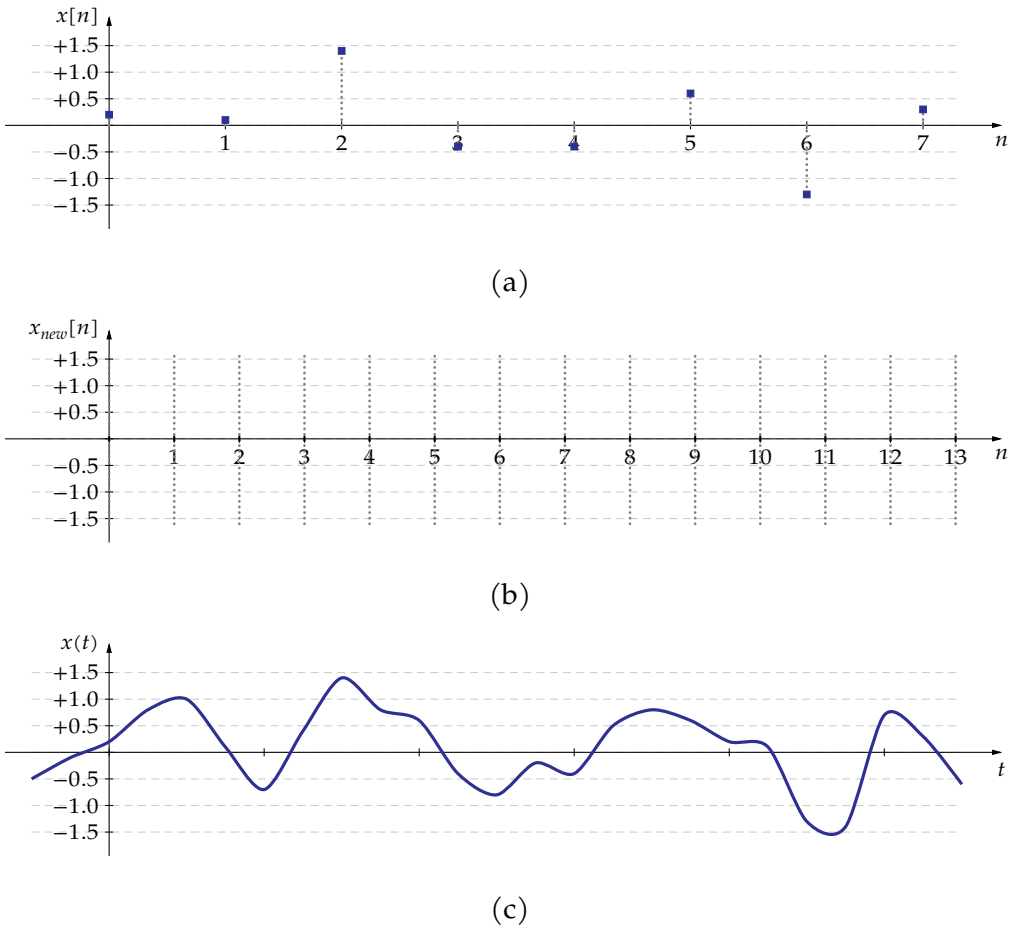


Figure 4.11: Illustration of the problem of wide-sense interpolation: (a) a discrete-domain signal, (b) the time points on which we want to generate new data, (c) the original signal that we would like to have known

4.3.5 Interpolation techniques

4.3.5.1 Overview

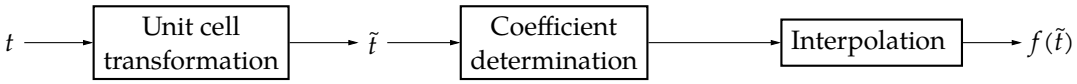
Several interpolation techniques can be considered. The list below is not exhaustive, but it can stand the test.

- Zeroth-order hold interpolation
- First-order (linear) interpolation
- Third-order (cubic spline) interpolation
- Interpolation by convolution
- Interpolation by polyphase filtering
- Zero stuffing

For now, we will restrict ourselves to the first three of them.

4.3.5.2 Basic Principle

To enable a sound and simple mathematical treatment, we will use a standard method for all three methods. It has been illustrated below.

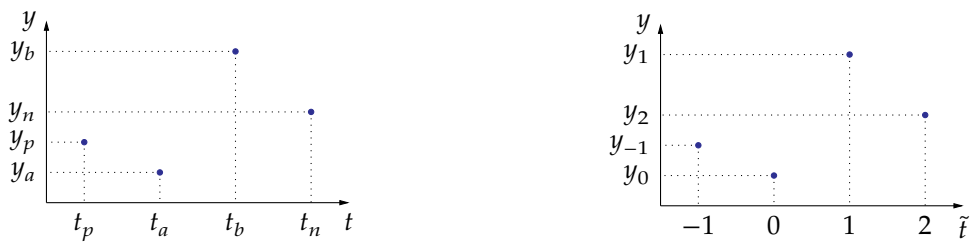


The reason for this procedure stems from the fact that we will use surrounding samples to create a local model $f_m(t)$ or $f_m(x, y)$. Developing the mathematical foundations for arbitrarily located samples hinders a simple treatment.

The first step is to perform a linear transformation of the domain, such that the sample right before the interpolated-location-to-be is mapped onto 0 and the sample just after is mapped onto 1. Assuming that the sample before is located at t_n, the sample-to-be at t and the sample right after is located at t_{n+1}, then the transformation is easily calculated as:

$$\tilde{t} = \frac{t - t_n}{t_{n+1} - t_n}$$

This principle has been illustrated below. The original situation has been depicted on the left, the transformed situation on the right. We will call this first step *unit cell transformation*. The unit cell is the $\tilde{t}$-domain ranging from 0 to 1. The sample at $\tilde{t} = 0$ is the y_0 *sample*, the sample at $\tilde{t} = 1$ is the so-called y_1 *sample*.

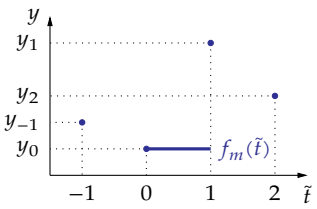


After this transformation, we will determine the coefficients of the model that we chose (zeroth-order hold, linear or cubic interpolation).

Finally, we will carry out the interpolation itself by evaluating the model.

4.3.5.3 Zeroth-order interpolation

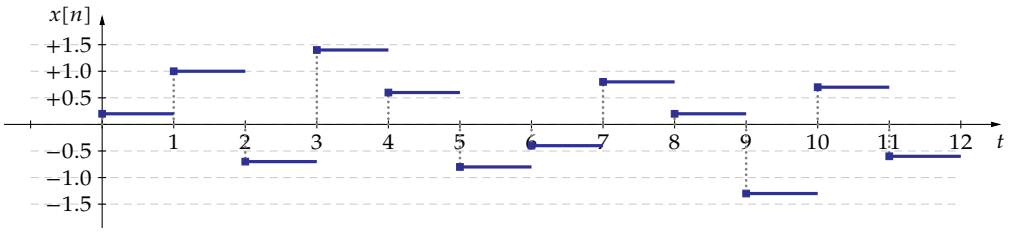
Zeroth-order interpolation is the most simple of methods. The idea is to maintain the value of the y_0 sample throughout the entire unit cell. This has been illustrated below.



The model therefore is:

$$0 \leq \tilde{t} < 1 : f_m(\tilde{t}) = y_0$$

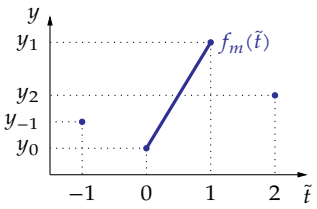
The result when applying this technique has been illustrated in the graph below.



The main advantage of this model is its simplicity. The main disadvantage of this model is that it is discontinuous.

4.3.5.4 First-order interpolation (linear interpolation)

A second attempt tries to make the model continuous, by using a linear model. This has been illustrated below.



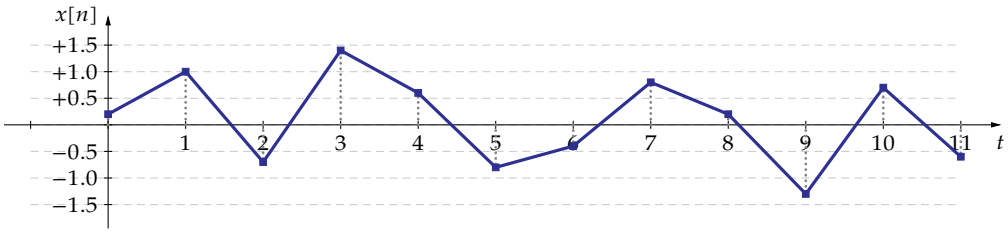
The model can be derived to be:

$$0 \leq \tilde{t} < 1 : f_m(\tilde{t}) = a_1\tilde{t} + a_0$$

The coefficients a_1 and a_0 can be determined by filling out the sample values of the y_0 and the y_1 sample into the model's equation:

$$\left.\begin{aligned} y_0 &= a_1 0 + a_0 \\ y_1 &= a_1 1 + a_0 \end{aligned}\right\} \Rightarrow \begin{cases} a_0 = y_0 \\ a_1 = y_1 - y_0 \end{cases}$$

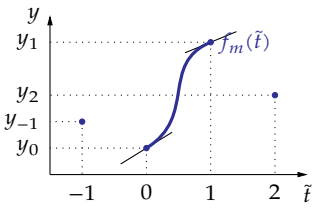
The result when applying this technique has been illustrated in the graph below.



The model is still quite simple. It is continuous. However, its main disadvantage is that is doesn't exhibit a continuous first derivative.

4.3.5.5 Third-order (spline) interpolation (cubic interpolation)

Our next (and for now final) attempt is to create a model with a continuous first-order derivative. This idea has been illustrated below. The derivatives at the cell boundaries ($\tilde{t} = 0$ and $\tilde{t} = 1$) are imposed. The model bends the curve in between the boundaries, such that it has the imposed derivatives at the cell borders.



An appropriate model to this end, is the model below:

$$0 \leq \tilde{t} < 1 : f_m(\tilde{t}) = \sum_{i=0}^{3} a_i \tilde{t}^i$$

As we need to make sure the derivatives of this model are continuous, we also calculate its derivative:

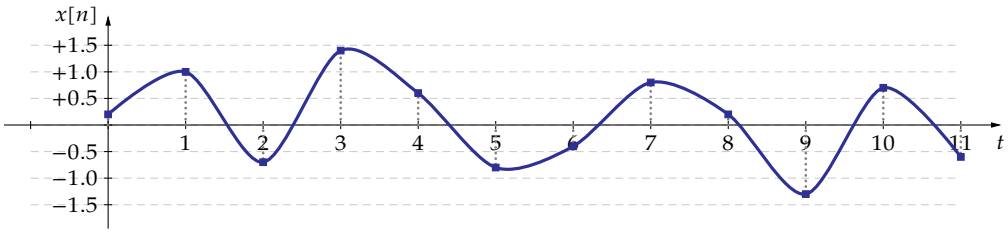
$$f'_m(\tilde{t}) = \sum_{i=1}^{3} i a_i \tilde{t}^{i-1} = 3a_3\tilde{t}^2 + 2a_2\tilde{t} + a_1$$

The coefficients can be found by filling out the values of the y_0 and y_1 samples and their derivative values into the model equation. By using consistent derivatives at subsequent unit cells, we can guarantee continuity in the first derivative.

$$\left.\begin{array}{l} f_m(0) = y_0 = a_0 \\ f_m(1) = y_1 = a_3 + a_2 + a_1 + a_0 \\ f'_m(0) = y'_0 = a_1 \\ f'_m(1) = y'_1 = 3a_3 + 2a_2 + a_1 \end{array}\right\} \Rightarrow \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -3 & 3 & -2 & -1 \\ 2 & -2 & 1 & 1 \end{bmatrix} \begin{bmatrix} y_0 \\ y_1 \\ y'_0 \\ y'_1 \end{bmatrix} \tag{4.3}$$

Careful consideration of the latter equations will reveal that after some optimization, it takes 3 multiplications and 4 additions to execute them.

The result when applying this technique has been illustrated in the graph below.



This is a top notch model at the expense of requiring a significant amount of computing power.

However, the attentive reader will have noticed one secret we didn't reveal yet. How are the values of y'_0 and y'_1 determined? We will discuss this in the next section.

4.3.5.6 Calculation of the unit-cell boundary derivatives

The values of y'_0 and y'_1 are just like y_0 and y_1 inputs when it comes to solving (4.3). The values of y_0 and y_1 are readily available, but how do we determine y'_0 and y'_1?

To this end, we need an estimator that fulfills one important condition. The imposed derivative on the upper boundary of a unit cell (y'_1) needs to be the same as the derivative imposed on the lower boundary of the neighboring cell (that abuts to the right). Likewise the imposed derivative on the lower boundary (y'_0) needs to be the same as the imposed derivative on the upper boundary of the neighboring cell (that abuts to the left).

We will use a parabole to provide our estimate, and luckily, parabola's have a peculiar property, that is expressed in the following lemma. To understand the lemma, take a look in parallel at Figure 4.12 on the following page.

Lemma: the derivative of a parabola based on a symmetrical sampling The calculation of a parabola's derivative f' in an arbitrary point x_0 can be easily determined by symmetrical sampling the parabola. Given

$$f(x) = ax^2 + bx + c$$

It's derivative in x_0 can be calculated as:

$$\forall x_0 \in \mathbb{R}, \forall \delta \in \mathbb{R}_0^+ : f'(x_0) = \frac{f(x_0+\delta) - f(x_0-\delta)}{2\delta},$$

The parabolic model together with this lemma ensures that our estimate for $y_1' = (y_2 - y_0)/2$ is consistent with the estimate of the dervitive at the lower bound of the right-abutting cell.

There we are: we found a way to estimate the derivatives at the boundaries of a unit cell that ensures a continuous first derivative over all cells.

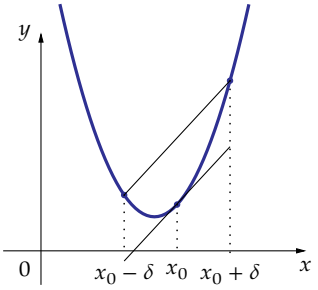


Figure 4.12: Illustration of the equidistant-sampled parabola derivative lemma

4.3.5.7 Computational comparison of the one-dimensional interpolation methods

The table below compares the amount of multiplications and additions required to perform the discussed interpolation methods. The time required for an addition is denoted by T_{ADD}, the time required for a multiplication by T_{MULT}.

	Unit cell transformation	Coefficient determination	Interpolation	Total
ZOH	0	0	0	0
Linear	$T_{MULT}+T_{ADD}$	T_{ADD}	$T_{MULT}+ T_{ADD}$	$2T_{MULT}+ 3T_{ADD}$
Bicubic	$T_{MULT}+T_{ADD}$	$3T_{MULT}+5T_{ADD}$	$3T_{MULT}+ 3\ T_{ADD}$	$7\ T_{MULT}+ 9T_{ADD}$

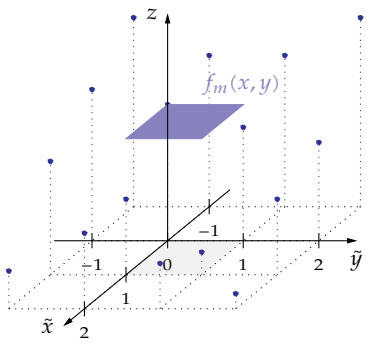
4.3.6 Off-grid resampling (wide-sense interpolation) in two dimensions

Again, we apply the same procedure as for the one-dimensional case:

4.3.6.1 Zeroth-order interpolation

Again, zeroth-order interpolation is the most simple of techniques we can apply. The idea is to maintain the value of the z_{00} sample throughout the entire unit cell. This has been illustrated

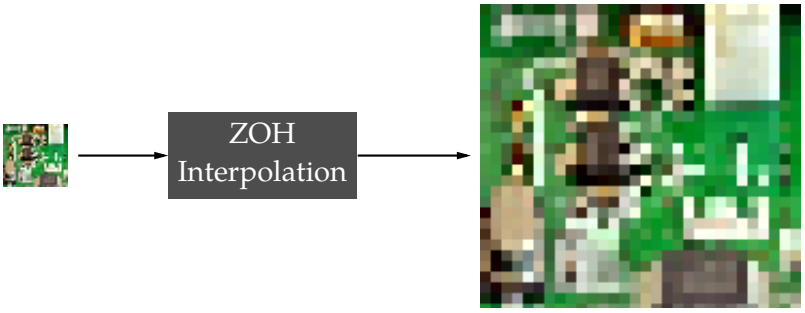
below.



The corresponding model is:

$$0 \leq \tilde{x}, \tilde{y} < 1 : f_m(\tilde{x}, \tilde{y}) = z_{00}$$

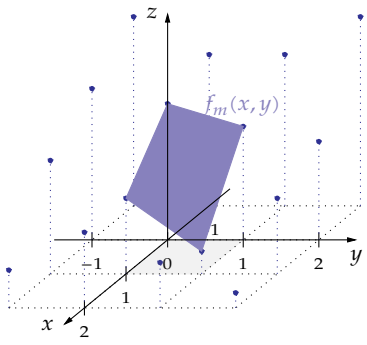
The result when applying this technique has been illustrated in the graph below.



The main advantage of this model is its simplicity. The main disadvantage of this model is that it is discontinuous.

4.3.6.2 First-order interpolation (bilinear interpolation)

A second attempt tries to make the model continuous, by using a linear model. This has been illustrated below.



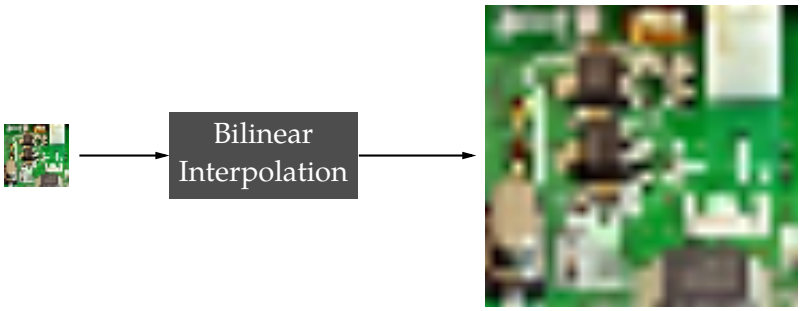
An appropriate model to this end, is the model below:

$$0 \leq \tilde{x}, \tilde{y} < 1 : f_m(\tilde{x}, \tilde{y}) = \sum_{i=0}^{1} \sum_{j=0}^{1} a_{ij} \tilde{x}^i \tilde{y}^j$$

By filling out the boundary conditions, we can determine the coefficients of the model:

$$\left.\begin{array}{l} z_{00} = a_{00} \\ z_{10} = a_{00} + a_{10} \\ z_{01} = a_{00} + a_{01} \\ z_{11} = a_{00} + a_{10} + a_{01} + a_{11} \end{array}\right\} \Rightarrow \begin{bmatrix} a_{00} \\ a_{10} \\ a_{01} \\ a_{11} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ +1 & -1 & -1 & 1 \end{bmatrix} \cdot \begin{bmatrix} z_{00} \\ z_{10} \\ z_{01} \\ z_{11} \end{bmatrix}$$

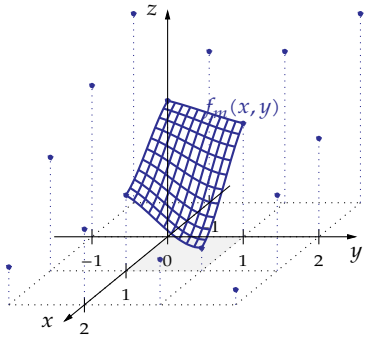
The result when applying this technique has been illustrated in the graph below.



The model is still quite simple. It is continuous, therefore it is quite superior to the zeroth-order hold model. However, its main disadvantage is that is doesn't exhibit a continuous first derivative.

4.3.6.3 Third-order (spline) interpolation (bicubic interpolation)

Our next (and for now final) attempt is to create a model with a continuous first-order derivative. This idea has been illustrated below.



An appropriate model to this end, is the model below:

$$0 \leq \tilde{x}, \tilde{y} < 1 : f_m(\tilde{x}, \tilde{y}) = \sum_{i=0}^{3} \sum_{j=0}^{3} a_{ij} \tilde{x}^i \tilde{y}^j$$

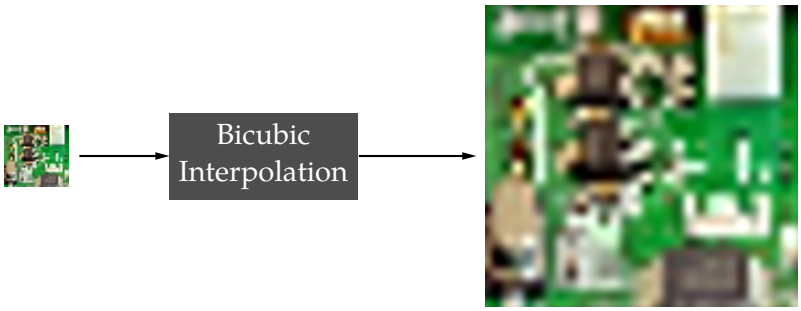
As we need to make sure the derivatives of this model are continuous, we also calculate its partial derivatives:

$$\begin{cases} \dfrac{\partial f}{\partial \tilde{x}} = \displaystyle\sum_{i=1}^{3}\sum_{j=0}^{3} i a_{ij} \tilde{x}^{i-1}\tilde{y}^{j} \\ \dfrac{\partial f}{\partial \tilde{y}} = \displaystyle\sum_{i=0}^{3}\sum_{j=1}^{3} j a_{ij} \tilde{x}^{i}\tilde{y}^{j-1} \\ \dfrac{\partial^2 f}{\partial \tilde{x}\partial \tilde{y}} = \displaystyle\sum_{i=1}^{3}\sum_{j=1}^{3} ij a_{ij} \tilde{x}^{i-1}\tilde{y}^{j-1} \end{cases}$$

The coefficients can be found by filling out the values of the z_{ij} samples and their derivative values into the model equation. By using consistent derivatives at subsequent unit cells, we can guarantee continuity in the first derivative.

We won't go through the maths here. Four nominal boundary conditions and twelve derivative boundary conditions allow determining the 16 parameters a_{ij} of the model.

The result when applying this technique has been illustrated in the graph below.



This is a top notch model at the expense of requiring a significant amount of computing power. It is marginally better than the bilinear model. However, with geometric transformations involving rotation, the bicubic interpolation shows its true value.

4.4 Domain transformations

4.4.1 Definition

A domain transformation maps the points of the domain of a signal into new locations for these points. As the domain of a signal very often can be modelled as a vector space (in which geometries make sense), we commonly denote these transformations as geometric transformations.

The principle has been illustrated in two dimensions on Figure 4.13. A point p in the planar space with coordinates (x, y) is mapped onto a new point p' with coordinates (u, v). Note that the domain and the image of the mapping is the same vector space.

As we're focusing on image processing in this part, we limit ourselves to the two-dimensional case (planar geometric transformations).

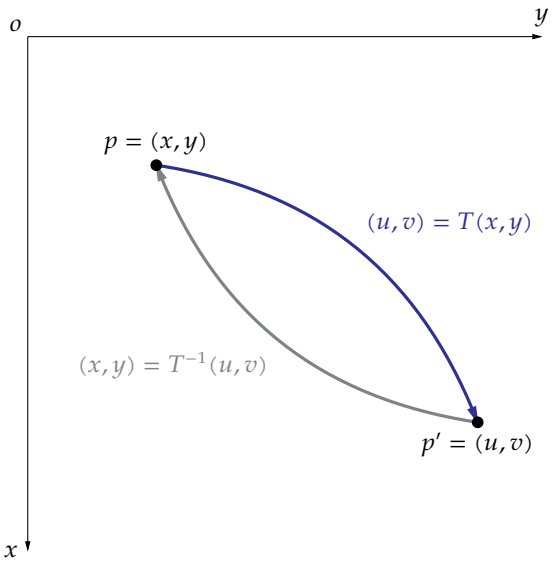


Figure 4.13: The principle of geometric transformations: a point p with coordinates (x, y) is mapped onto a new point p' with coordinates $(u, v) = T(x, y)$. The inverse mapping (assuming it exists) has been indicated with a gray arrow.

Definition: Geometric domain transformation (2D) A planar geometric domain transformation T maps every point (x, y) of the vector space onto a point (u, v) of the same vector space:

$$T : (x, y) \in \mathbb{R}^2 : (x, y) \mapsto (u, v) = T(x, y) \in \mathbb{R}^2$$

If the domain transformation is injective, the inverse transformation exists and is denoted by T^{-1}:

$$(x, y) = T^{-1}(u, v)$$

4.4.2 Rationale

Why would we need domain transformations in image processing? Many good reasons hold. A small selection to illustrate this:

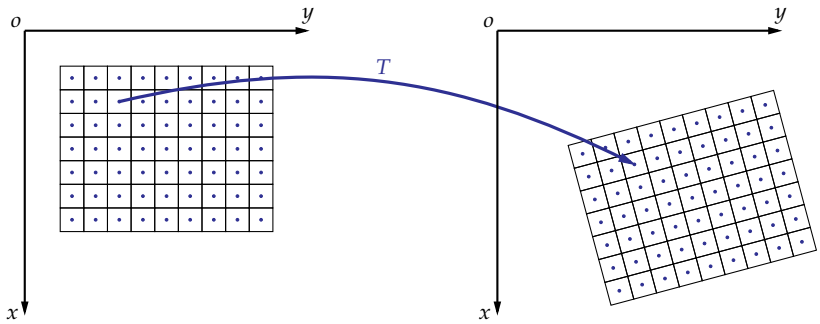
- to zoom-in or zoom-out on a specific image (e.g., a 3D representation of an object in a CAD software package);
- to rescale and rotate photographs to make them ready for printing;
- to remove unwanted deformation caused by the lenses of the image detection system;
- to project a three-dimensional image to a two-dimensional image;
- to perform image registration.

In fact, geometric transformations are one of the most applied image processing techniques in general.

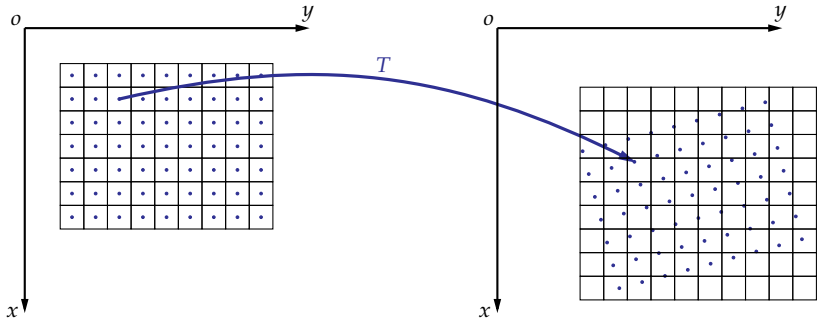
4.4.3 Interpolation after transformation

Let's take a detailed look into what happens when we perform a planar transformation on a sampled image with square pixels. Let's denote the original image as the source image and the resulting image as the target image.

Step 1: Transformation of the source image into the target image domain
The domain of the image consists of a number of pixel locations. Every one of these pixel locations is transformed into a new pixel location. This has been illustrated below. To avoid cluttering up the figure, the situation before transformation has been depicted on the left, the situation after transformation on the right. The pixel locations have been indicated as an array of dots. The arrow illustrates how one of the source pixels is mapped to the corresponding target pixel.



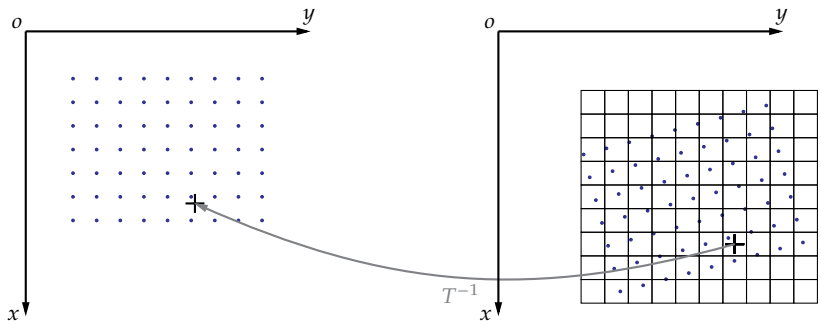
Step 2: Impose the pixel locations of the target medium
However, the target pixel grid as displayed above is not the one that the target medium offers. It offers a straight grid, just as the source medium did. This has been depicted below.



Our goal is to determine pixel intensities belonging to the center positions of the target's straight grid.

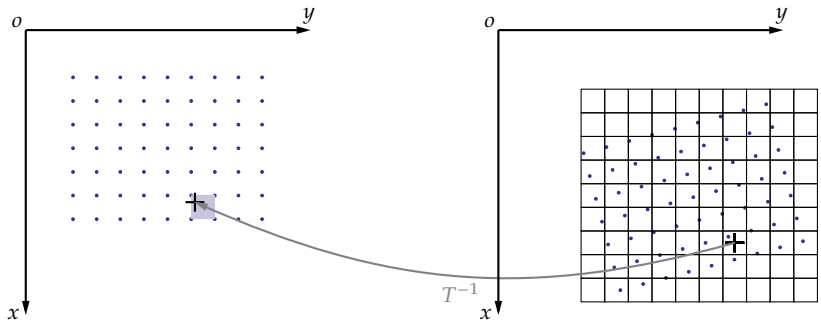
Step 3: Determine the inverse transformation of a target pixel

Consider the center position of a pixel of the target medium and perform the inverse transformation T^{-1} on this pixel position.



Step 4: Interpolate in the surrounding cell

The inverse transformation yielded a coordinate in the source picture. Find its surrounding cell (indicated in gray on the picture below) and perform interpolation using your interpolation model of choice.



4.4.4 Common geometric transformations

In this section, we will discuss a number of common geometric transformations. We will limit ourselves to transformations that maintain lines. We will consider:

- Linear transformations
- Affine transformations
- Projective transformations

The former two maintain the parallelism of lines. The latter does not. Instead, it allows to correct (or invoke) perspective in the image.

4.4.4.1 Linear transformations

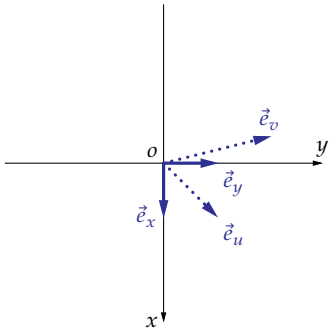
Forward transformation

A linear transformation can be fully defined by defining the transformation of the basis vectors.

Definition: Linear transformation A linear transform T_{lin} maps the basis vectors $(\vec{e}_x, \vec{e}_y)$ onto a new set of basis vectors $(\vec{e}_u, \vec{e}_v)$ using a linear relationship:

$$T_{\text{lin}} : (\vec{e}_x, \vec{e}_y) \mapsto (\vec{e}_u, \vec{e}_v) = (a_{11}\vec{e}_x + a_{21}\vec{e}_y, a_{12}\vec{e}_x + a_{22}\vec{e}_y)$$

This has been illustrated in the picture below:



Writing the coordinates of the mapped basis vectors as columns of a matrix A, i.e.

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$$

allows very convenient mathematics for applying the transformation or its inverse to a specific point with coordinates (x, y). We denote A as the *transformation matrix*.

Indeed, given (x, y) the transformed coordinates $(u, v) = T(x, y)$ can be easily determined to be:

$$\begin{bmatrix} u \\ v \end{bmatrix} = A \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

Inverse transformation

If A can be inverted (i.e. it is nonsingular, equivalently $\det(A) \neq 0$), the following also holds:

$$\begin{bmatrix} x \\ y \end{bmatrix} = A^{-1} \cdot \begin{bmatrix} u \\ v \end{bmatrix}$$

with

$$A^{-1} = \frac{1}{\det(A)} \text{Adj}\,(A) = \frac{1}{a_{11}a_{22} - a_{21}a_{12}} \begin{bmatrix} a_{22} & -a_{12} \\ -a_{21} & a_{11} \end{bmatrix}$$

Remarks Note that a linear transformation maps the origin to itself.

Exercises

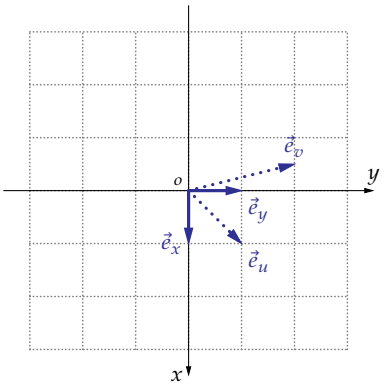
Some exercises on linear transformations.

Exercise 4.4.4.1-1: Determine the transformation matrix corresponding to the following linear transformation T:

$$T : \begin{cases} \vec{e}_u = 0.3\vec{e}_x - 0.8\vec{e}_y \\ \vec{e}_v = -2\vec{e}_x + 3\vec{e}_y \end{cases}$$

Then determine the coordinates of $T(3,2)$. Determine the transformation matrix corresponding to the inverse transform and use it to verify that $\left(T^{-1} \circ T\right)(3,2) = (3,2)$.

Exercise 4.4.4.1-2: Determine the transformation matrix corresponding to the linear transformation depicted below.



Using this matrix, determine the coordinates of $T(-0.5,-1)$.

4.4.4.2 Affine transformations

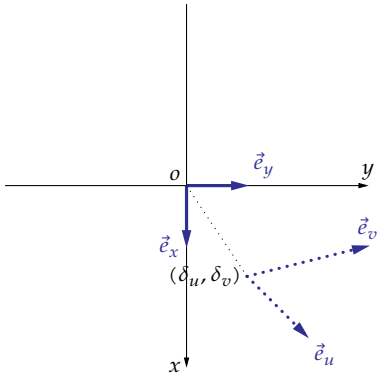
Forward transformation

An affine transformation is an extension of a linear transformation.

Definition: Affine transformation An affine transformation is a linear transformation followed by a translation.

$$\begin{bmatrix} u \\ v \end{bmatrix} = A \cdot \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} \delta_u \\ \delta_v \end{bmatrix} \tag{4.4}$$

This has been illustrated in the picture below:



It is very common to rewrite this transformation using *homogeneous coordinates*. This new type of coordinates is easily obtained by extending the coordinate column vector $[x\ y]^T$ with an extra row containing the constant value 1, so that it becomes $[x\ y\ 1]^T$.

In this way, (4.4) can be rewritten as:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} a_{11} & a_{12} & \delta_u \\ a_{21} & a_{22} & \delta_v \\ 0 & 0 & 1 \end{bmatrix}}_{T} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Inverse transformation

If A can be inverted (i.e. it is nonsingular, equivalently $\det(A) \neq 0$), then T is also nonsingular and can be inverted:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = T^{-1} \cdot \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

with

$$\begin{aligned} T^{-1} &= \frac{1}{\det(T)} \mathrm{Adj}\,(T) \\ &= \frac{1}{a_{11}a_{22} - a_{21}a_{12}} \begin{bmatrix} a_{22} & -a_{12} & a_{12}\delta_v - a_{22}\delta_u \\ -a_{21} & a_{11} & a_{21}\delta_u - a_{11}\delta_v \\ 0 & 0 & a_{11}a_{22} - a_{21}a_{12} \end{bmatrix} \end{aligned}$$

Remarks

- Note that a linear transformation is a special case of an affine transformation (with $\delta_u = \delta_v = 0$).
- Note that the inverse of an affine transformation is also an affine transformation.

4.4.4.3 Projective transformations

Going one step further, we meet the projective transformation. This transformation no longer keeps lines parallel.

This transformation (also known as *homography*) is intended to add or remove perspective to or from an image. It is required to be able to perform measurements on an image (e.g. planar angles or length of line sections in the same plane). An alternative use is to stitch together images taken from the same point, but under different angles.

Definition: Projective transformation A projective transformation consists of a linear transformation of the plane $z = 1$ in three dimensions:

$$\begin{bmatrix} u' \\ v' \\ w \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

followed by a homothetic projection on the plane $z = 1$:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \frac{1}{w} \begin{bmatrix} u' \\ v' \\ w \end{bmatrix}$$

Remarks

- Note that an affine transformation is a special case of a projective transformation.
- The *image processing toolbox* of MATLAB offers facilities to execute these transforms by means of the commands `maketform`, `tformfd` and `tforminv`. Be aware that the conventions MATLAB uses are aberrant w.r.t. our discussion.
  MATLAB uses row-vectors and T^T and interchanges x- and y-axis. One can solve this by interchanging rows and columns before ánd after applying `tformfd`. Alternatively, one can start thinking in the framework of the image processing toolbox.

4.4.5 Special cases of linear/affine transformations

Many special cases of the linear/affine transformation make sense. Table 4.1 lists most of them. We will illustrate all of them in the subsequent paragraphs.

Make sure that you are able to compose them yourself. Start by making a drawing of the transformation of the basis vectors.

Naam	T	Illustrated in
Scaling	$\begin{bmatrix} k_x & 0 & 0 \\ 0 & k_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$	Figure 4.14
x-axis shear	$\begin{bmatrix} 1 & s_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	Figure 4.15
y-axis shear	$\begin{bmatrix} 1 & 0 & 0 \\ s_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	Figure 4.16
Rotation	$\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$	Figure 4.17
Translation	$\begin{bmatrix} 1 & 0 & \delta_u \\ 0 & 1 & \delta_v \\ 0 & 0 & 1 \end{bmatrix}$	Figure 4.18
x-axis reflection	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	Figure 4.19
y-axis reflection	$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	Figure 4.21

Table 4.1: Special cases of linear/affine transformations

Scaling

This transformation stretches the image along the x-axis by a factor k_x and along the y-axis by a factor k_y.

To illustrate this transform, consider the following specific case.

$$T_s = \begin{bmatrix} k_x & 0 & 0 \\ 0 & k_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{with } k_x = 1.5 \text{ and } k_y = 1.25 \tag{4.5}$$

The effect has been depicted in Figure 4.14 below.

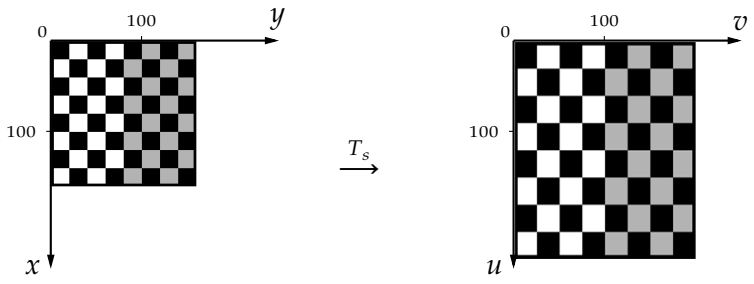


Figure 4.14: Illustration of scaling according to (4.5)

X-axis shear

This transformation shears the image along the x-axis with an increasing amount s_x per unit of distance w.r.t. the x-axis.

To illustrate this transform, consider the following specific case.

$$T_{s,x} = \begin{bmatrix} 1 & s_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{with } s_x = 0.3 \tag{4.6}$$

The effect has been depicted in Figure 4.15 below.

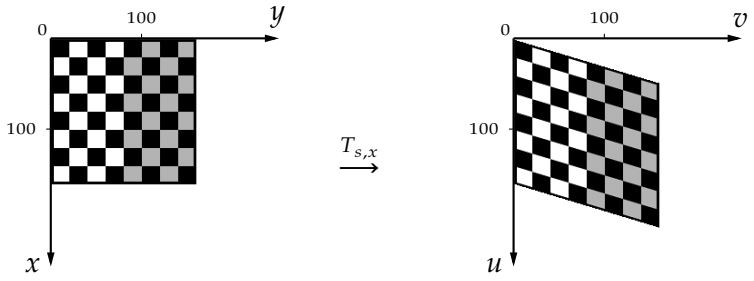


Figure 4.15: Illustration of x-shear according to (4.6)

Y-axis shear

This transformation shears the image along the y-axis with an increasing amount s_y per unit of distance w.r.t. the y-axis.

To illustrate this transform, consider the following specific case.

$$T_{s,y} = \begin{bmatrix} 1 & 0 & 0 \\ s_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{with } s_y = 0.3 \tag{4.7}$$

The effect has been depicted in Figure 4.16 below.

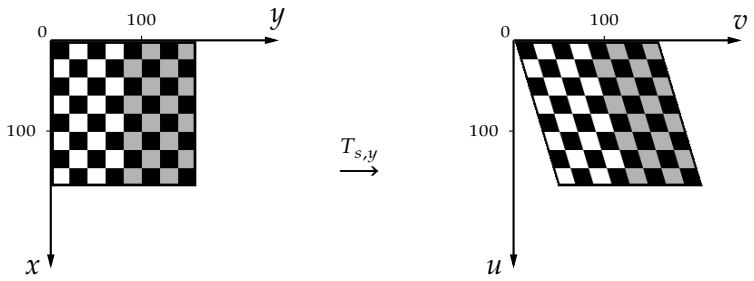


Figure 4.16: Illustration of y-shear according to (4.7)

Rotation

This transformation rotates the image around the z-axis (perpendicular to the x, y-plane pointing upward). Be aware of the ambiguity of angle specification in ° and rad.

To illustrate this transform, consider the following specific case.

$$T_r = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{with } \theta = 30° \tag{4.8}$$

The effect has been depicted in Figure 4.17 below.

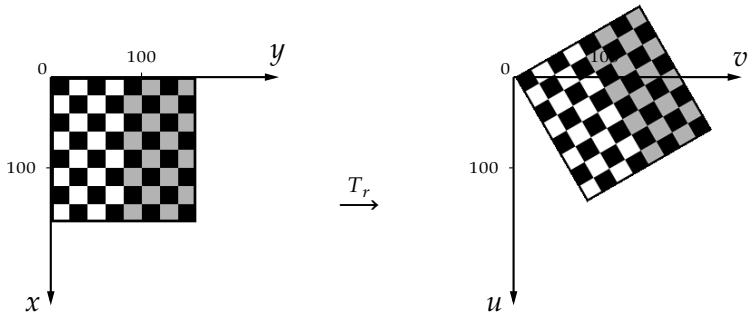


Figure 4.17: Illustration of rotation according to (4.8)

Translation

This transformation translates the image over an amount (δ_u, δ_v).

To illustrate this transform, consider the following specific case.

$$T_t = \begin{bmatrix} 1 & 0 & \delta_u \\ 0 & 1 & \delta_v \\ 0 & 0 & 1 \end{bmatrix} \quad \text{with } (\delta_u, \delta_v) = (20, -50) \tag{4.9}$$

The effect has been depicted in Figure 4.18 below.

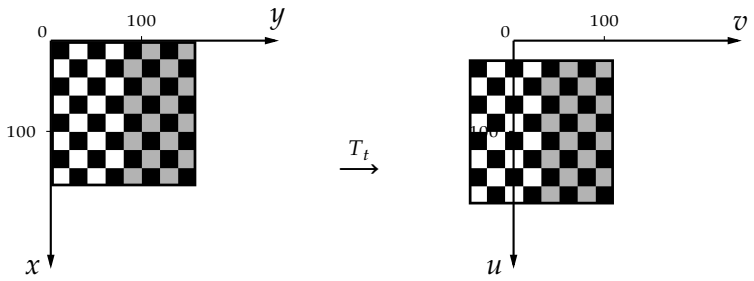


Figure 4.18: Illustration of translation according to (4.9)

X-axis reflection

This transformation reflects the image w.r.t. the x-axis.

To illustrate this transform, consider the following specific case.

$$T_{m,x} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.10}$$

The effect has been depicted in Figure 4.19 below.

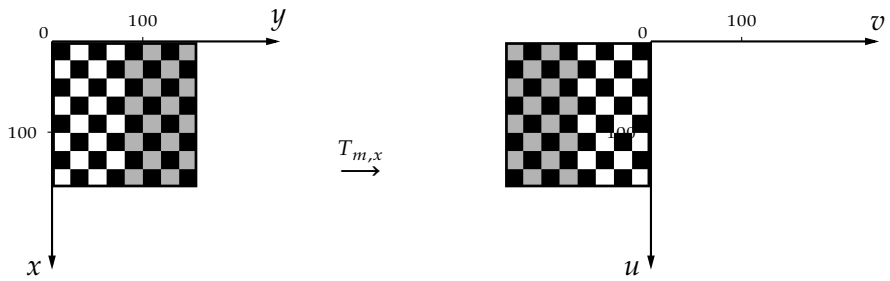


Figure 4.19: Illustration of reflection w.r.t. the x-axis according to (4.10)

Y-axis reflection

This transformation reflects the image w.r.t. the y-axis.

To illustrate this transform, consider the following specific case.

$$T_{m,y} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.11}$$

The effect has been depicted in Figure 4.21 below.

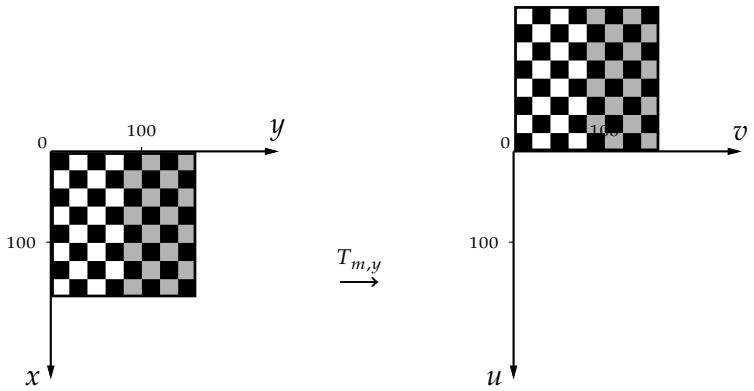


Figure 4.20: Illustration of reflection w.r.t. the y-axis according to (4.11)

4.4.6 Combining multiple linear/affine transformations

Combining linear and/or affine transformations is most easy. Just multiply their individual transformation matrices in order from right to left, to obtain the combined transformation matrix.

Symbolically[2]: If $(u,v) = (T_n \circ T_{n-1} \circ \cdots \circ T_2 \circ T_1)\,(x,y)$ then equivalently:

$$\begin{aligned} \begin{bmatrix} u \\ v \end{bmatrix} &= T_n \cdot \left(T_{n-1} \cdot \left(\cdots \left(T_2 \cdot \left(T_1 \cdot \begin{bmatrix} x \\ y \end{bmatrix} \right) \right) \right) \right) \\ &= (T_n \cdot T_{n-1} \cdot \cdots \cdot T_2 \cdot T_1) \begin{bmatrix} x \\ y \end{bmatrix} \end{aligned}$$

To illustrate this transform, consider the following specific case.

$$T = T_t \cdot T_r \cdot T_{s,x} \tag{4.12}$$

[2]The $\circ$ operation denotes a 'comes after' operation in this case, not a morphological opening.

The effect has been depicted in Figure 4.21 below.

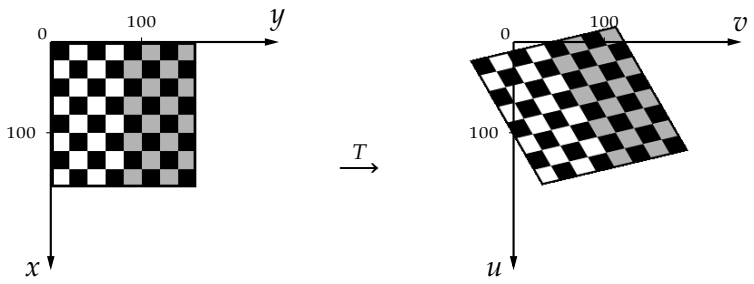
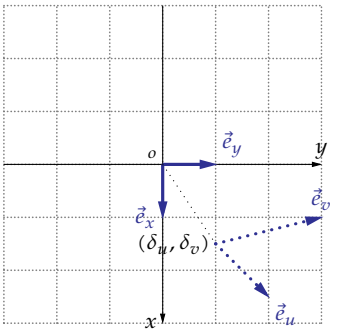


Figure 4.21: Illustration of a sequence of transformations according to (4.12)

Exercises

Some exercises on affine transformations.

Exercise 4.4.6-1: Determine the transformation matrix corresponding to the mapping of the basis vectors $(\vec{e}_x, \vec{e}_y)$ to $(\vec{e}_u, \vec{e}_v)$, according to the picture below:



Determine the transformation of the point with coordinates $[x\ y]^T = [-0.5\ \ -1]^T$.

Exercise 4.4.6-2: Compose the transformation matrix of a rotation over $-\pi/3$ rad, followed by an x-shear with a distance factor s_x of 2 and a translation over $(\delta_u, \delta_v) = (1, 2)$. Apply this domain transformation to the points $(0, 0)$ and $(-1, 4)$.

4.4.7 Registration

The term *registration* has a very specific meaning in the context of digital image processing.

If we want to compare an acquired image (e.g., a photograph of a scenery) with a reference image (e.g., an old photograph of the same scenery), we need to remove any acquisition distortion (e.g., scaling, rotation, a perspective change) before we can really compare the two images on a pixel per pixel basis. This process is called *image registration*.

Definition: registration Registration means aligning an acquired image with a reference image.

Remarks

- This is typically done by finding corresponding control points in the two images, followed by the quest for a transformation that relates the corresponding control points.
- In many cases special features are built into an image to facilitate identifying control points (e.g., the big squares in a QR-code).

Chapter 5

Quantization and Requantization

In this chapter, you will learn:

- why we quantize signals;
- how we quantize signals;
- how we quantize images;
- how images can be requantized using intensity transformations.

After having read/studied this chapter, you are expected to be able to

- understand and explain what quantization is, how it is used and why it is used;
- apply quantization to signals and images;
- understand, explain and apply pointwise and spatial intensity transformations;
- understand, explain and apply the image operations correlation and convolution and explain how they are related tot intensity transformations.

5.1 Introduction

On a macroscopic level, the physical quantities of our world can be considered to be *continuous in nature*. For example, when considering the temperature range between water becoming ice (273.15 K) and boiling water (373.15 K), we can halve that range and halve the first half again. We can continue this process endlessly. Not taking physical limitations into account, any positive real temperature can be considered. This is what we mean by saying that temperature is continuous in nature.

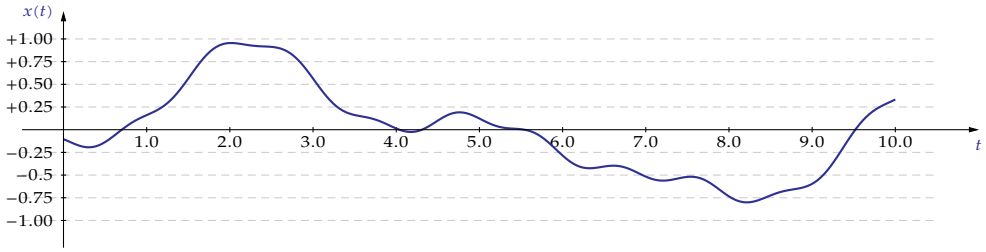
Though this infinite resolution of our temperature scale is fascinating, it is not very useful for practical purposes. Usually a coarse measurement is sufficient to achieve good results in a particular application. In addition, often the measurement sensor or the measurement setup given the available budget limits the accuracy that can be achieved. Therefore, limiting the representation accuracy of a physical quantity is no big deal.

This is often done by only considering values on a specific grid. In signal processing jargon, the rounding process to translate arbitrary values to the available grid values is called *quantization*.

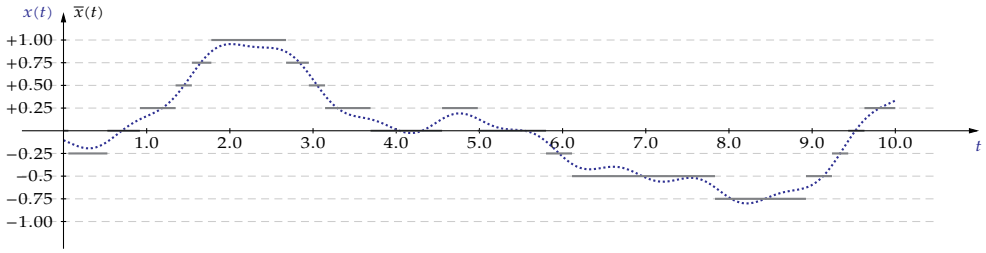
Though quantization is no big deal from a practical point of view, its effect is all but negligible. Therefore, we need to spend some time to understand its nature and its consequences.

5.2 Quantization

Consider the signal $x(t)$ below.



Suppose that we need to quantize this signal to the quantization grid indicated by the horizontal dashed lines. Though several options are available, an obvious one would be to round the signal to the nearest grid value. This is called *scientific rounding*. In this way we obtain the quantized signal $\overline{x}(t)$, that has been drawn below.



We formulate this operation mathematically as:

$$\overline{x}(t) = Q(x(t))$$

with Q the quantization function.

5.2.1 Rationale

In the introduction to this chapter, we already summarized why we need quantization. Let's elaborate on this a little further.

The *first observation* is that often we are only interested in a limited range of values. E.g., in a domestic heating application, measuring the room temperature in between -10 and $40\,°C$ is probably more than sufficient.

The *second observation* is that in the framework of digital signal processing we tend to measure physical quantities using a two-step process.

The first step is that we like to make sensors that translate the physical quantity to be measured into an electrical signal (a voltage or a current). The range of electrical signals that our measurement system (e.g. an embedded sensor system) can cope with is also limited (e.g., by the system's power supply).

The second step is to translate that voltage or current in a number that can be processed by our signal processor. Typically, numbers represented by a digital computer are limited in range and discrete by nature.

Measurement takes time and power Taking the two steps mentioned above takes time. As it turns out, the more accurate you want the measurement to be, the more time you need to devote to assessing the measured value, and the more power you will need per measurement. Therefore measuring physical quantities will require you to explore the accuracy vs. speed and power trade-off surface.

Limited number representation The same power-speed-accuracy trade-off is present inside any computing system. To obtain acceptable speeds, we need to accept the fact that we perform calculations with a limited precision. Any common number format ((u)int8, (u)int16, (u)int32, ..., float, double, ...) has a limited precision. You should be aware that representing $\sqrt{2}$ using a computer number format, requires you to round this value to the closest by binary floating point value, before you can store or manipulate it.

Though it is possible to perform infinite precision arithmetic using a computer, it is rarely used in digital signal or image processing applications.

5.2.2 Definition

In view of the above, we define quantization as follows:

Quantization Consider a signal with a limited range:

$$x_{\min} \leq x(t) \leq x_{\max}.$$

. Quantization consists of two elements:

- subdividing the range in (numbered) intervals $l_i = [x_i, x_{i+1}]$ with $L = \cup_i l_i$ covering the range[a]
- mapping every (numbered) interval to a *reconstruction level*

$$Q : x(t) \mapsto L \mapsto \overline{x}(t) = Q\left(x(t)\right) \tag{5.1}$$

[a] We may consider *'out of range'* intervals such that our DSP can take into account clipping when it occurs.

This has been illustrated graphically in Figure 5.1.

5.2.3 Example

Consider a DSP processor that has a 12-bit AD converter that quantizes the voltage range on an analog input pin uniformly. The reference voltage is 3.8V (i.e. the range is 0 V to 3.8 V).

Two questions to illustrate some basic math on the principles of quantization:

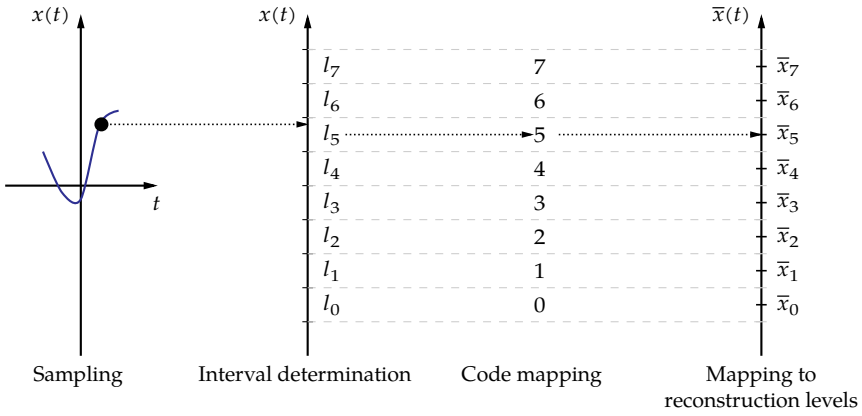


Figure 5.1: Illustration of the process of quantization

1. To what code is an input voltage of 2 V mapped?
2. If the code produced by the ADC equals 1000, then what voltage corresponds to that value?

Answer to question 1 Let's call the input signal x and start with calculating the interval size:

$$\Delta x = \frac{x_{\max} - x_{\min}}{2^N} = \frac{3.8\,\mathrm{V} - 0\,\mathrm{V}}{2^{12}} = \frac{3.8\,\mathrm{V}}{4096} = 927.74\,\mu\mathrm{V}.$$

Let's now map the actual input signal voltage to a specific interval. To this end, we divide the input voltage by the interval size to obtain a decimal number v

$$v = \frac{x - x_{\min}}{\Delta x} = \frac{2\,\mathrm{V} - 0\,\mathrm{V}}{927.74\,\mu\mathrm{V}} = 2155.8.$$

By truncating v one obtains the interval number. Therefore $x \in l_{2155}$. The DSP will read a value of 2155 from the ADC value register. This indicates that:

$$2155\Delta x \leq x \leq 2156\Delta x.$$

In a 16-bit fixed-point DSP, we would continue our calculations with the signal value 2155. In a floating-point DSP, this value is more appropriately converted to a reconstruction value

$$\bar{x}_{2155} = 2155.5\Delta x = 2155.5 \cdot 927.74\,\mu\mathrm{V} = 1.999\,7\,\mathrm{V}.$$

Of course, it may make sense to scale this value to maximize the range usage in the DSP and to avoid internal clipping.

Answer to question 2 The interval size Δx has been calculated in the previous paragraph. This allows us to translate the ADC code 1000 to an interval:

$$1000\Delta x \leq x < 1001\Delta x.$$

Therefore, an appropriate estimate of x is:

$$\begin{aligned} x &= 1000.5\Delta x \pm 0.5\Delta x \\ &= 0.928\,20\,\mathrm{V} \pm 0.000\,47\,\mathrm{V}. \end{aligned}$$

Exercises

Exercise 5.2.3-1: A microcontroller converts analog input voltages between −0.2 V and 1.2 V using an 8-bit ADC.
Two questions:

1. To which binary code is a value of 0.6 V mapped?
2. What voltage corresponds to an interval number of 100?

Exercise 5.2.3-2: A DSP is capable of converting analog voltages in between ±1.5 V using a 10-bit converter.
Two questions:

1. In which interval is a voltage of 0 V quantized? To which reconstruction value is it mapped under the assumption of no scaling?
2. What voltage corresponds to an interval number of 314?

Exercise 5.2.3-3: (*) Find the datasheet of the ADC of a TMS320x2080x DSP. Make a drawing of how the input voltages are mapped to input codes and calculate the interval size. If the readout value of the ADC register is 2000, then what voltage has been sampled?

Exercise 5.2.3-4: (*) Find the datasheet of the ADC of a DSP or microcontroller of your choice. Make a drawing of how the input voltages are mapped to input codes and calculate the interval size. Pick an arbitrary readout value and calculate with what voltage it corresponds.

5.2.4 Classification

Depending on when we determine the intervals and their relative size, we can distinguish multiple quantization variants.

Fixed vs. variable interval quantization Fixing the intervals at design time, is called *fixed interval quantization*. If the intervals are only determined at run time we are using *variable interval* or *adaptive quantization*. In this scenario we could decrease the interval size for slowly varying signals ($\mathrm{d}x/\mathrm{d}t$ small) and increase the interval size if the signals start to change more rapidly ($\mathrm{d}x/\mathrm{d}t$ large).

Uniform vs. graded quantization If all intervals are equal in size, we are using so-called *uniform quantization*. Using intervals that are not of equal size, is called nonuniform or *graded quantization*. A-law or μ-law quantization (employed in cell-phone communications) are examples of the latter.

That said, *fixed uniform interval quantization* is predominantly used, for its simplicity.

5.3 Quantization of images

5.3.1 Principle

Quantization of images is by no means different from quantization of one-dimensional signals. In the case of images, the detector of every pixel of the imaging device measures the light or color intensity, leading to a grayscale value (or color value) for the pixel. The range of available grayscales most often is divided in 2^N intervals. In many cases $N = 8$. For grayscale images, $N = 12, 16, 24, 32$ is not uncommon. For color images values larger than $N = 16$ per color channel are very rare. The amount of grayscales or colors that can be represented in this way is — given the limited capabilities of our vision — in most cases more than sufficient.

Grayscale		Color	
N (nr. of bits)	2^N (nr. of scales)	N (nr. of bits/channel)	2^{3N} (nr. of colors)
8	256	8	16M
12	4K	12	69G
16	65K	16	281T
24	16M		
32	4G		

The only difference with the generic case is the fact that the function that is quantized now is a two-dimensional function. Hence, assuming that the spatial variables x and y are discretized and the function $f[x, y]$ is quantized into an interval collection L, (5.1) can be rewritten as:

$$Q : f[x, y] \mapsto L \mapsto \bar{f}[x, y] = Q\left(f[x, y]\right).$$

5.3.2 What can go wrong?

What can go wrong with image quantization? Many things. But they all result in one phenomenon: images with a suboptimal use of the quantization range.

Lighting problems The lighting of a scene that we want to image may be insufficient. This causes overly dark images. The lighting may also be overabundant, causing overly white images.

Exposure problems We mentioned earlier that measuring takes time. An image sensor counts the number of photons hitting the sensor during a settable exposure time. When this exposure time is too small, overly dark images arise; when it is too large, overly white images are our problem.

Mapping problems Even when a scene is properly lighted and the exposure is globally correct, the distribution of all intensities may not be optimal. It might happen that part of the

range of the sensor is underused and other parts of the sensor are overused. In that case, the discriminating power of the sensor is not optimally exploited.

We treat these problems using intensity transformations and requantization, the topic of the following section.

5.4 Requantization / Intensity transformations

5.4.1 Definition

Requantization Consider an $M \times N$ image represented using monochromatic intensity values (grayscales or intensities of a color channel). We assume the intensity values lie within the range 0 (black/dark) and $L-1$ (white/bright).

We denote the intensity level of an arbitrary pixel of the image by $l[x, y]$, with x and y the coordinates of the pixel.

A *requantization* Q_2 is an image operation that maps *all* of the original intensity levels $l[x, y]$ to new intensity levels $m[x, y]$, that also lie within the range 0 and $L-1$:

$$m[x, y] = Q_2(l[x, y]). \tag{5.2}$$

Remarks

- Note that the transformation Q_2 must be defined for every original intensity level!
- Also note that both $l[x, y]$ and $m[x, y]$ are discrete in nature. Still, in many cases it makes sense to study the relationship between a non-quantized version of $m[x, y]$ and a non-quantized ('unrounded') version of $l[x, y]$. In that case, we call this stripped down definition of the process of requantization an *intensity transformation*.
- In addition, the quite common functional notation $l[x, y]$ and $m[x, y]$ risks leading us to an overly narrow interpretation of (5.2). Indeed, the notation might lead to the faulty conclusion that the new intensity for a specific pixel (x_1, y_1) can only determined by the old intensity level for the same pixel. That is not true. To be more correct, we'd better write:

$$m[x, y] = Q_2(l, x, y).$$

In fact, we denote intensity transformations

- in the more strict sense, as *pointwise intensity transformations*,
- in the broader sense, as *spatial intensity transformations*.

Let's take a detailed look at both of them.

5.4.2 Pointwise intensity transformations

As engineers, we like to divide the world in halves, and continue to divide the halves in halves until the entire world is binary classified. Let's succumb to this weakness and split the world

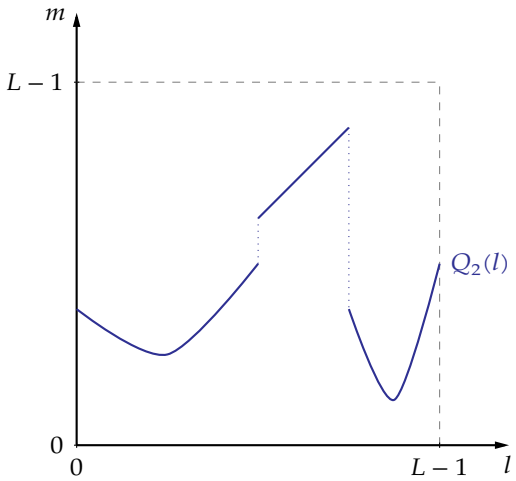


Figure 5.2: A generic intensity transformation

Figure 5.3: Test image: Ciudad de las Artes y las Ciencias, Valencia, Spain (source: W. Daems)

of pointwise intensity transformations into two alternative flavors:

- fixed deterministic transformations,
- deterministic transformations based on image statistics.

We explicitly added the word 'deterministic' to avoid the common confusion that the latter is random in nature (due to the word 'statistics'). It is not: it is a purely deterministic transformation as well. Often this word is omitted.

Let's go into more detail.

5.4.2.1 Fixed deterministic transformations

These are intensity transformations that do not depend on the image that is being transformed. An example of a generic intensity transformation is given in Figure 5.2. Note that the transformation is defined for the entire domain of l, i.e. from 0 tot $L-1$.

Let's take a look at a few examples. They have been illustrated in Figure 5.4 on page 70. The transformations have been applied to the custom test image of Figure 5.3.

Image negative The image negative inverts the grayscale of the image. It is defined by:

$$Q_{neg} : m = L - 1 - l.$$

Threshold transformation This transformation maps all intensities below a given value to black (dark) and the rest to white (bright). It is often used in image segmentation to transform a grayscale image to a binary image.

$$Q_{thr} : m = \begin{cases} L-1, & \text{if } l > a \\ 0 & \text{if } l \leq a \end{cases}$$

Intensity slicing This is a generalization of the threshold transformation. It is often used in image segmentation.

Intermezzo - reversibility Let's take a look back at the three intensity transformations we've treated so far. The first one, the image negative, allows restoring the original image by applying the intensity transformation a second time. An intensity transformation that allows restoring the original is a so-called *reversible* intensity transformation.

In contrast, after applying one of the latter two, it is not possible to restore the original image. These intensity transformations cannot be undone. These are so-called *irreversible* intensity transformations.

Let's continue discovering some more of these intensity transformations. They have been illustrated in Figure 5.4 and Figure 5.5.

Log transformation This transformation makes the image brighter.

$$Q_{\ln} : m = \frac{L-1}{\ln L} \ln(1 + l).$$

Inverse log transformation This transformation makes the image darker.

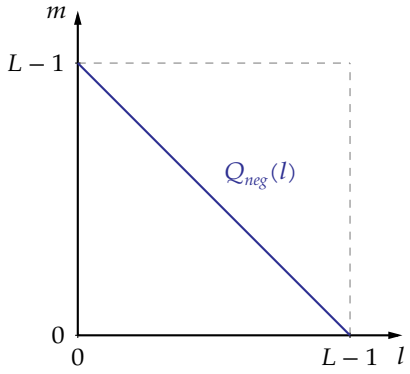
$$Q_{\text{invln}} : m = L^{\frac{l}{L-1}} - 1.$$

Gamma transformation This transformation makes the image darker or brighter depending on the value of the parameter γ. It is also known as the *power law transformation*.

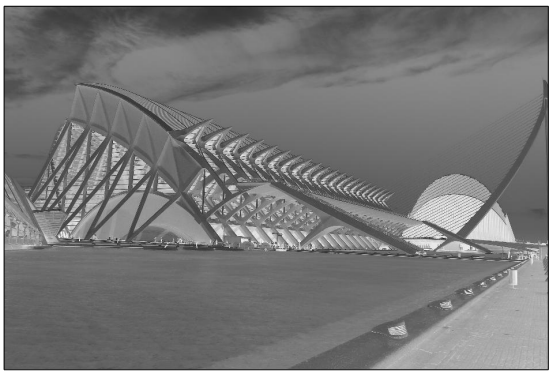
$$Q_{\gamma} : m = \frac{L-1}{(L-1)^{\gamma}} l^{\gamma}$$

Contrast stretching This transformation allows devoting the majority of the intensity range to a specific portion of the intensities.

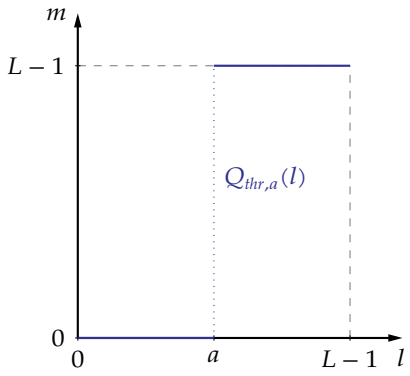
$$Q_{cs,c,S} : m = (L-1)\frac{l^S}{l^S + c^S}$$



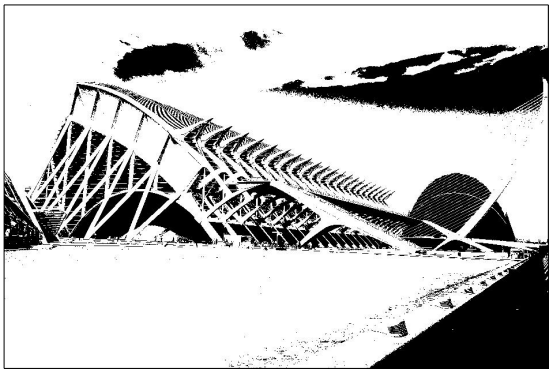
(a) Negative transformation



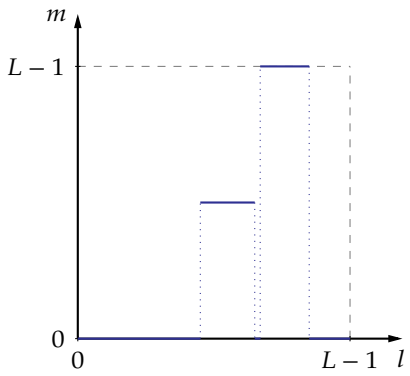
(b) Negative transformation result



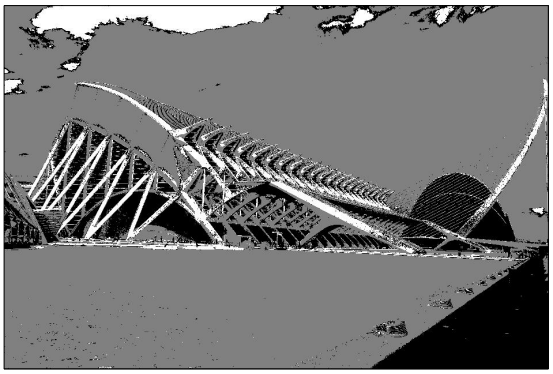
(c) Threshold transformation



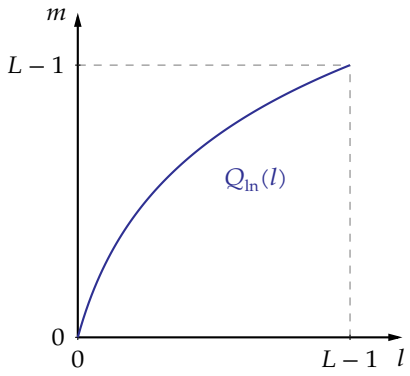
(d) Threshold transformation result



(e) Intensity slicing



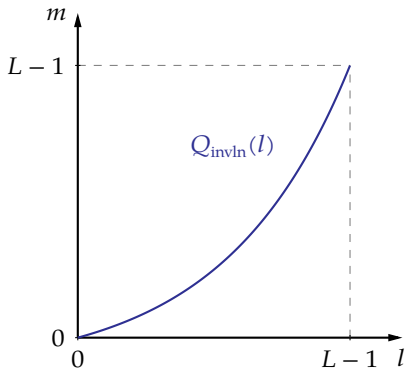
(f) Intensity slicing result



(g) Log transformation

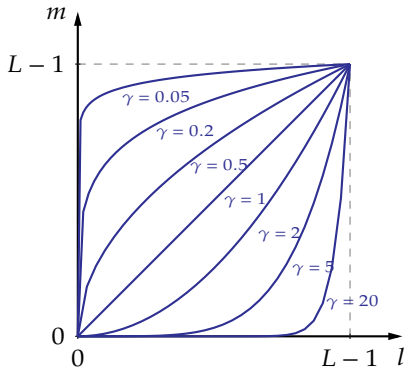
(h) Log transformation result

Figure 5.4: Fixed (deterministic) intensity transformations



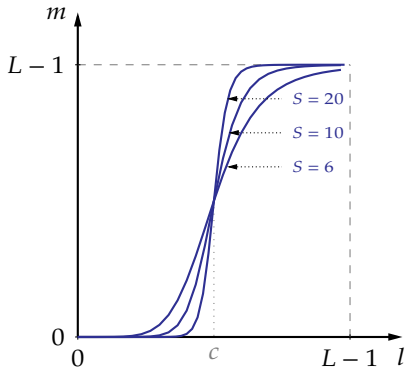
(a) Inverse log transformation

(b) Inverse log transformation result



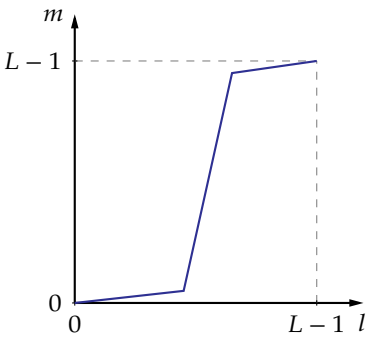
(c) Gamma transformation

(d) Gamma transformation result (for $\gamma = 0.2$)



(e) Contrast stretching transformation

(f) Contrast stretching transformation result



(g) Piecewise linear transformation

(h) Piecewise linear transformation result

Figure 5.5: Fixed (deterministic) intensity transformations (cont'd)

The transformations above only allow for a limited number of degrees of freedom. However, many more (more creative) mappings are possible. The piecewise-linear transformation is just an example of the more exotic options.

Piecewise-linear transformation In the example of Figure 5.5g on the preceding page only three linear pieces compose the PWL graph. Of course, many more complex examples can easily be conceived.

Exercises

Exercise 5.4.2.1-1: Consider the following pixel canvas, containing intensity values of a grayscale image quantized with 8-bit.

$f[x,y]$ y x

165	53	155	255	215	148
173	181	0	89	212	138
162	60	117	169	65	255
0	30	169	106	156	68

Apply the following transformations to this canvas and requantize:

- image negative
- threshold transformation

$$m = \begin{cases} 0 & \text{if } l < 100 \\ 255 & \text{if } l \geq 100 \end{cases}$$

- log transformation
- inverse log transformation
- gamma transformation for $\gamma = 4$
- contrast stretching transformation (with $c = 0.45(L-1)$ and $S = 4$

Exercise 5.4.2.1-2: Consider the following pixel canvas, containing intensity values of a grayscale image quantized with 8-bit.

$f[x,y]$ y x

81	122	139	56	103	160
30	163	184	27	114	197
240	139	133	28	93	238
165	165	253	16	195	248

Apply the following transformations to this canvas and requantize:

- image negative
- threshold transformation

$$m = \begin{cases} 0 & \text{if } l < 140 \\ 255 & \text{if } l \geq 140 \end{cases}$$

- log transformation
- inverse log transformation
- gamma transformation for $\gamma = 0.4$
- contrast stretching transformation (with $c = 0.55(L-1)$ and $S = 2$

Exercise 5.4.2.1-3: (*) Rework the equations of the intensity transformations, such that an $L-$bit picture is transformed to an $M-$bit picture.

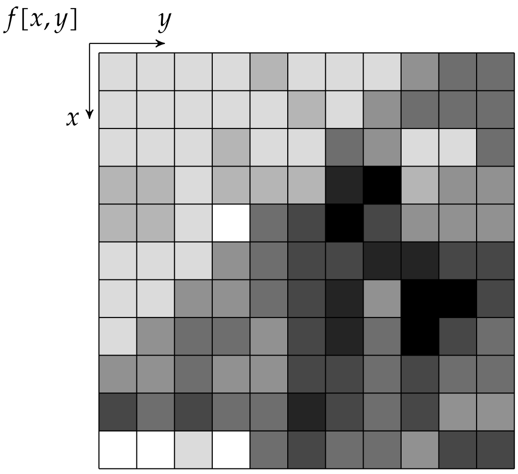
Exercise 5.4.2.1-4: (*) MATLAB supports image transformations quite well. Take a picture from your own photo collection. A few basic steps can help you in processing your image:

1. you can import it in MATLAB using the `imread` function;
2. you can convert it to a grayscale version using the `rgb2gray` function;
3. you can convert the integer representation to a floating point representation using `im2single` or `im2double`;
4. then you can process your image using all the MATLAB trickery you can imagine;
5. you can reconvert the floating point representation to an integer version suited for writing to disk, using the `im2uint8` function; this will also requantize it;
6. finally, you can write the image back to disk using `imwrite`.

Now, perform the following transformations to your image and check the results using `imshow`:

- image negative (the `imcomplement` function may be handy)
- threshold transformation
- log transformation
- inverse log transformation
- gamma transformation (for $\gamma = 0.4$) (check out the `imadjust` function)
- contrast stretching transformation
- intensity slicing transformation (the `mat2gray` may help you); pick some arbirary slicing pattern
- piecewise-linear transformation (the `interp1` function may be useful); pick some arbitrary line pattern

Tip: don't use the MATLAB command line. Write a script instead.



(a) Grayscale view

$f[x,y]$ (rows: x, columns: y)

6	6	6	6	5	6	6	6	4	3	3
6	6	6	6	6	5	6	4	3	3	3
6	6	6	5	6	6	3	4	6	6	3
5	5	6	5	5	5	1	0	5	4	4
5	5	6	7	3	2	0	2	4	4	4
6	6	6	4	3	2	2	1	1	2	2
6	6	4	4	3	2	1	4	0	0	2
6	4	3	3	4	2	1	3	0	2	3
4	4	3	4	4	2	2	3	2	3	3
2	3	2	3	3	1	2	3	2	4	4
7	7	6	7	3	2	3	3	4	2	2

(b) Pixel intensity view

f	$n(f)$	$c(f)$	$n(f)/N$	$c(f)/N$
0	5	5	0.0413	0.0413
1	6	11	0.0496	0.0909
2	20	31	0.1653	0.2562
3	25	56	0.2066	0.4628
4	21	77	0.1736	0.6364
5	11	88	0.0909	0.7273
6	29	117	0.2397	0.9669
7	4	121	0.0331	1.0000

(c) Frequency table

Figure 5.6: Example 11 × 11 image quantized using 3 bits

5.4.2.2 Deterministic transformations based on image statistics

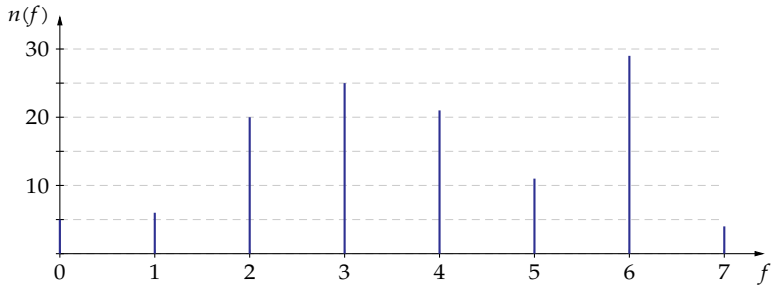
So far, we've considered transformations that do not depend on the contents of the image. In this section, we will encounter transformations that take the actual image into account. To this end, we need to take a more detailed look at images.

In essence images are a bunch of pixel intensities positioned on a regular grid. If we consider any of these pixel intensities as the result of a random variable, we can consider the statistics of this random variable.[1]

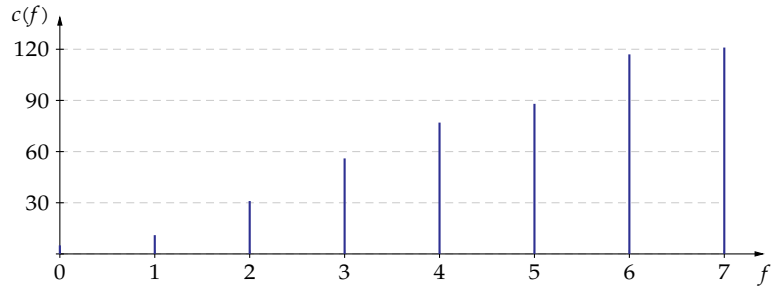
A common way to analyze the statistics of random variables is to draw a histogram. We can do the same for an image. As an example, consider the 3-bit quantized 11×11 image $f[x,y]$ of Figure 5.6. The image has been drafted on the left using grayscales. On the right side you can find the actual pixel intensity values.

The frequency table corresponding to this image is shown in Figure 5.6c. The corresponding histogram can be found in Figure 5.7. The number of pixels N equals 121. The frequency of the

[1]For a brief refresher on random variables, consult appendix A on page 263.



(a) Frequency histogram



(b) Cumulative frequency histogram

Figure 5.7: Histogram of the image of Figure 5.6

.

pixel intensities f is denoted by $n(f)$. The cumulative sum of the frequencies is denoted by $c(f)$.

In fact, the histogram gives us a coarse idea of how well the intensity information is spread over all available intensity values. The basic principle of the next two intensity transformations is that we can use the histogram information to optimize this distribution.

In the following subsections we will treat two possible techniques:

- histogram equalization
- histogram matching

Histogram equalization The goal of histogram equalization is to find a transformation that makes the histogram of the image as uniform as possible.

This has been illustrated in Figure 5.8. The graph at the bottom shows the histogram of a particular image with intensities ranging from 0 to $L-1$. Now, the intensity variable l is transformed to a new intensity variable m using the function of the top left graph. The result of this is that the new intensity variable m is uniformly distributed over the range 0 to $L-1$ as indicated in the top right graph. The key questions is: how can we find an appropriate transformation function?

Foundation The basis for this technique is the *equalization theorem* that is well known from statistics.

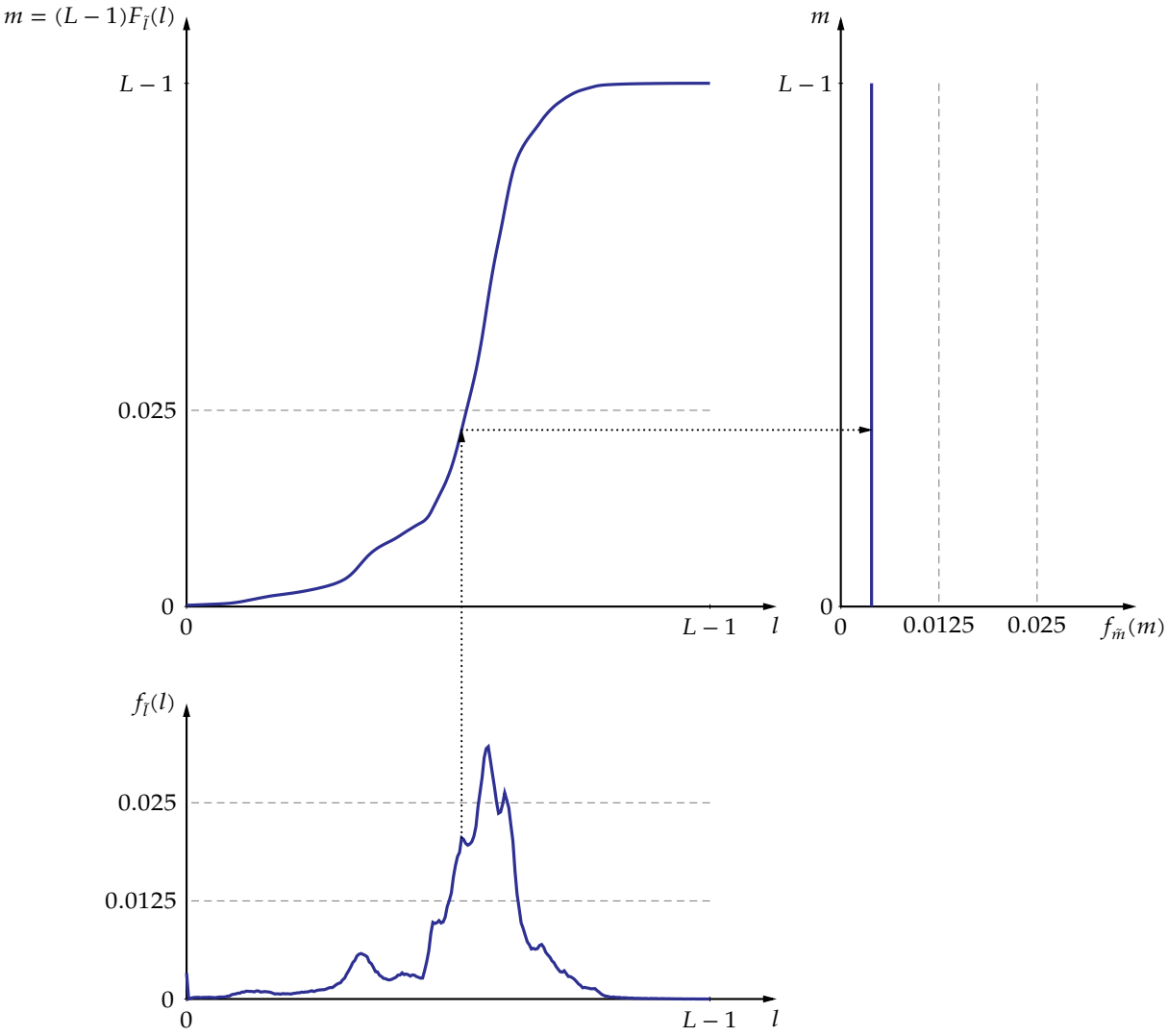


Figure 5.8: Histogram equalization - principle: the variable l (distributed as given in bottom graph) can be transformed into a uniformly distributed variable m (distribution in the top right graph, sideways) using the cumulative distribution of l times $L-1$ as a mapping function (the top left graph)

Equalization theorem Given a random variable X

- corresponding to experiments with a continuous outcome set,
- with a range limited between a and b, and
- with a probability density function $f_X(x)$

the transformed random variable

$$Y = g(X)$$

such that $c = g(a)$ and $d = g(b)$ will have a uniform probability density function $f_Y(y) = \frac{1}{d-c}$ if:

$$g(x) = (d-c)\int_a^x f_X(u)\,\mathrm{d}u = (d-c)F_X(x).$$

Again, check Figure 5.8 that illustrates the principle to an example histogram. Now, let's prove this theorem.

Proof:
The transformation property for (continuous) distribution functions states that

$$\begin{aligned} f_Y(y) &= \frac{1}{\frac{\partial g(x)}{\partial x}} f_X(x) \\ \downarrow \quad & \frac{\partial g(x)}{\partial x} = (d-c) f_X(x) \quad \text{(due to Leibniz's rule)} \\ &= \frac{1}{(d-c)f_X(x)} f_X(x) = \frac{1}{d-c} \end{aligned} \tag{5.3}$$

■

Conclusion: *the (scaled) cumulative distribution function is the proper transformation function to make the histogram uniform.*

Unfortunately, there is no discrete equivalent for this (continuous) equalization theorem. The discrete version is what we need to be able to treat images. Luckily it turns out that for typical images, the number of grayscales is so big, that the intensity of the pixels (our random variable) can be considered to be near-continuous. This is one of the cases in which our engineering pedigree surfaces: "We know that the technique is theoretically incorrect, but who cares? It works!"

However, the traditional method often found in literature (e.g. in [GW07]) and that uses (5.3) literally, is suboptimal. We will therefore not treat it, but use a better one. Luckily it turns out to be (1) easier and (2) also more generic. Life can be sweet.

Example Let's show how it works using an example. Consider the image of Figure 5.3 on page 68. It looks quite dull. The reason for this is the fact that its histogram (see Figure 5.9a) is not well balanced.

WDSC

Let's apply a histogram equalization transform to this image. The basis for this

i.e. applying the following transformation:

$$Q_{HE} : m = (L-1) \sum_{\lambda=0}^{l} f_{\tilde{I}}(\lambda) = (L-1) F_{\tilde{I}}(l).$$

The resulting image can be found in Figure 5.10 on page 80. The histogram of this new image can be found in Figure 5.9b. At first sight, this histogram seems not uniform at all. However, this is due to the bin size that was chosen too small. When chosing a larger bin size, we get the result of Figure 5.9c, which looks a lot more uniform.

Illustrating the mechanics of this technique on the example above, would be quite cumbersome. Let's treat a simpler example. Consider the image of Figure 5.11, i.e. a 3-bit quantized version of the original image of Figure 5.3 on page 68.

The (normalized) histogram data of this image can be found below in columns 1 and 2:

l	$f_{\tilde{I}}(l)$	$F_{\tilde{I}}(l)$	$(L-1)F_{\tilde{I}}(l)$	m
0	0.0054	0.0054	0.0380	0
1	0.0218	0.0272	0.1907	0
2	0.0788	0.1061	0.7425	1
3	0.1386	0.2447	1.7130	2
4	0.6280	0.8727	6.1091	6
5	0.1223	0.9950	6.9651	7
6	0.0049	0.9999	6.9991	7
7	0.0001	1.0000	7.0000	7

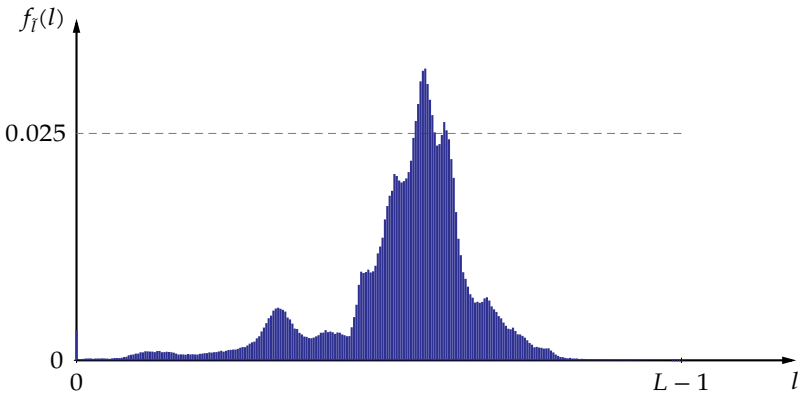
The histogram itself has been plotted in Figure 5.12.

The cumulative distribution and the scaled version have been calculated in columns 3 and 4. Column 4 contains the results of applying the proper transformation function g (i.e. the scaled cumulative distribution function) to the discrete intensity levels of the original image. The last column is just a snap-to-grid of those values to the discrete intensity levels of the new image (i.e. they have been rounded). The table can now be used as a mapping table by combining the first and the last column. A grayscale value of the first column is mapped to a new grayscale value of the last column.

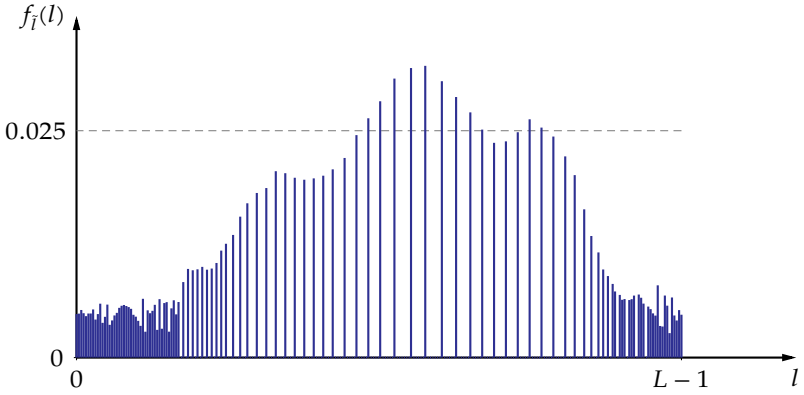
The resulting histogram after transformation can be found in Figure 5.13. The resulting image is shown in Figure 5.14. As you can see, the resulting histogram is not uniform at all. This is due to the fact that for discrete random variables, the equalization property does not hold. However, the histogram is more widely spread over the entire range and in that sense 'more uniform'. For more fine-grained quantizations, in general, the approximation gets better.

Histogram equalization is a cheap transformation. It requires a considerable amount of computations, but they're all very cheap: only addition and scaling is required.

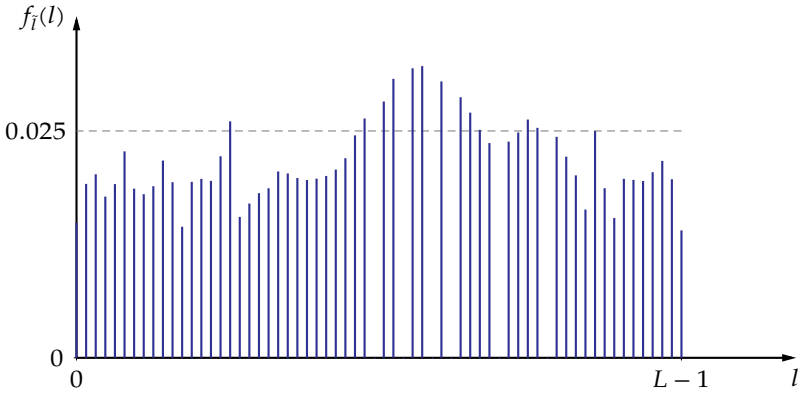
Histogram matching The goal of histogram matching is to make the histogram of the image resemble as closely as possible a given histogram (e.g. of another image). To this end, we will use the equalization theorem again. To make this possible, we will use an intermediate step, i.e. we will first equalize the histogram and then map it to the desired histogram.



(a) Histogram of the original image



(b) Histogram after equalization



(c) Histogram after equalization with an increased bin size

Figure 5.9: Histograms of the image of Figure 5.3 on page 68

Figure 5.10: Result of histogram equalization applied to the image of Figure 5.3 on page 68

Figure 5.11: 3-bit quantized version of the image of Figure 5.3 on page 68

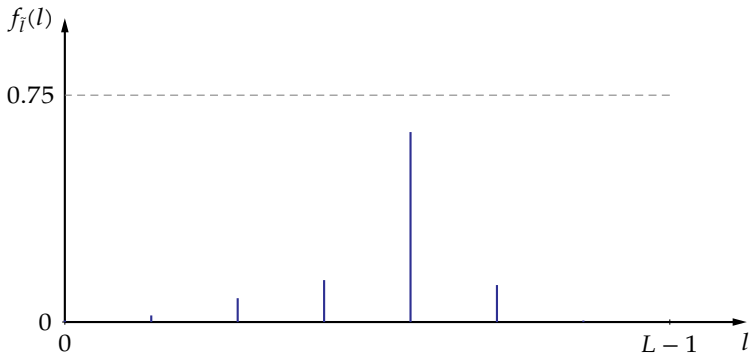


Figure 5.12: Histogram of the image of Figure 5.11

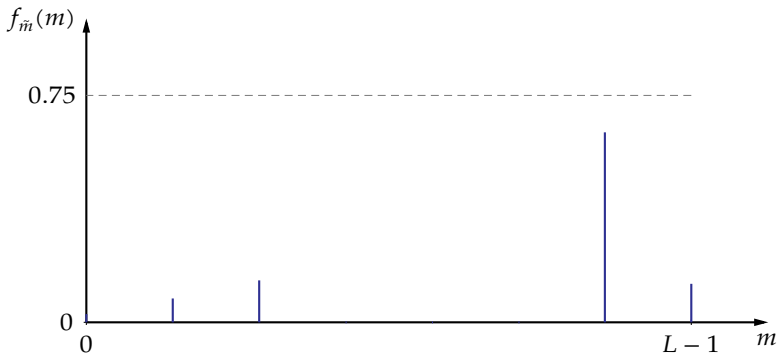


Figure 5.13: Histogram of the image of Figure 5.11 on the facing page after equalization

Figure 5.14: Result of histogram equalization applied to the image of Figure 5.11

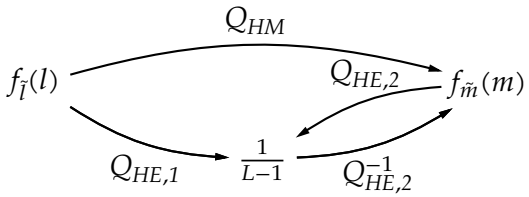


Figure 5.15: Principle of histogram matching

Foundation The principle has been depicted in Figure 5.15. Our goal is to transform a given distribution $f_{\tilde{l}}(l)$ to a desired distribution $f_{\tilde{m}}(m)$ using a transform Q_{HM}. Given our knowledge from the previous section on histogram equalization, we are perfectly capable of transforming both distributions to a uniform one $(1/(L-1))$ using $Q_{HE,1}$ and $Q_{HE,2}$. The trick is now to invert the transformation $Q_{HE,2}$, such that the chain can be completed:

$$m = Q_{HE,2}^{-1}\left(Q_{HE,1}\left(l\right)\right).$$

Example As an example, consider the histogram data below. It belongs to two simple 11×11 test images A and B, quantized using 3-bit intensity values l. Image A again corresponds to Figure 5.3 on page 68. Image B is in fact nonexistent. The histogram is just a linear decreasing histogram for testing purposes.

l	n_A	n_B
0	5	27
1	6	24
2	20	20
3	25	17
4	21	13
5	11	10
6	29	7
7	4	3

Let's begin with mapping these distributions to the uniform distribution. The result of the calculations (whose principles have been explained in section 5.4.2.2 on page 75) can be found in Table 5.1. Our goal is now to create the mapping from l to m, by ensuring that:

$$Q_{HE,1}(l) = Q_{HE,2}(m)$$

However, given the discrete nature of our intensity values, we will never be able to realize a total identity. Instead, we need to match pairs that are the closest to eachother.

To this end, we construct a distance table (see Table 5.2), with the uniformized intensity levels $Q_{HE,1}(l)$ vs. l in the leftmost columns and $Q_{HE,2}(m)$ vs. m in the top rows.

It is to be read as follows: suppose one wants to transform the intensity of the pixels with intensity level $l = 5$. Find the row containing $l = 5$, go down the row until you find the column that contains the minimal distance value (i.e. in this case 0.04, indicated in boldface for your

l	n	$f_{\tilde{I}}(l)$	$F_{\tilde{I}}(l)$	$7 \cdot F_{\tilde{I}}(l)$
0	5	0.04	0.04	0.29
1	6	0.05	0.09	0.64
2	20	0.17	0.26	1.79
3	25	0.21	0.46	3.24
4	21	0.17	0.64	4.45
5	11	0.09	0.73	5.09
6	29	0.24	0.97	6.77
7	4	0.03	1.00	7.00

(a) Calculations for $Q_{HE,1}$

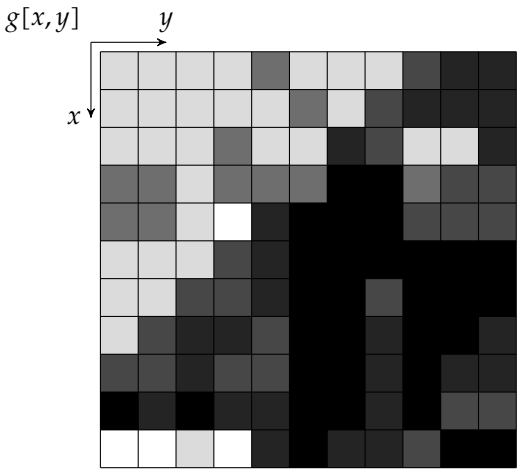
m	n	$f_{\tilde{m}}(m)$	$F_{\tilde{m}}(m)$	$7 \cdot F_{\tilde{m}}(m)$
0	27	0.22	0.22	1.55
1	24	0.20	0.42	2.93
2	20	0.16	0.58	4.07
3	17	0.14	0.72	5.05
4	13	0.11	0.83	5.80
5	10	0.08	0.91	6.37
6	7	0.06	0.97	6.77
7	3	0.02	1.00	7.00

(b) Calculations for $Q_{HE,2}$

Table 5.1: Calculation results for histogram equalization

		m							
		0	1	2	3	4	5	6	7
l		1.55	2.93	4.07	5.80	6.37	6.77	7.00	
0	0.29	**-1.26**	−2.64	−3.78	−4.76	−5.51	−6.08	−6.48	−6.71
1	0.64	**-0.91**	−2.29	−3.44	−4.41	−5.16	−5.73	−6.13	−6.36
2	1.79	**0.24**	−1.13	−2.28	−3.26	−4.00	−4.58	−4.98	−5.21
3	3.24	1.69	**0.31**	−0.83	−1.81	−2.56	−3.13	−3.53	−3.76
4	4.45	2.91	1.53	**0.38**	−0.59	−1.34	−1.91	−2.32	−2.55
5	5.09	3.54	2.16	1.02	**0.04**	−0.70	−1.28	−1.68	−1.91
6	6.77	5.22	3.84	2.69	1.72	0.97	0.40	**0.00**	−0.23
7	7.00	5.45	4.07	2.93	1.95	1.20	0.63	0.23	**0.00**

Table 5.2: Level distance table for histogram matching of test image A to test image B; minimum distance positions have been indicated in boldface.



(a) Gray scale view

g[x, y] y x

6	6	6	6	3	6	6	6	2	1	1
6	6	6	6	6	3	6	2	1	1	1
6	6	6	3	6	6	1	2	6	6	1
3	3	6	3	3	3	0	0	3	2	2
3	3	6	7	1	0	0	0	2	2	2
6	6	6	2	1	0	0	0	0	0	0
6	6	2	2	1	0	0	2	0	0	0
6	2	1	1	2	0	0	1	0	0	1
2	2	1	2	2	0	0	1	0	1	1
0	1	0	1	1	0	0	1	0	2	2
7	7	6	7	1	0	1	1	2	0	0

(b) Pixel intensity view

Figure 5.16: Resulting image obtained by histogram matching the image of Figure 5.6a on page 74 to the linearly declining histogram of image B

convenience). Then go up that column to find the intensity value $m = 3$ to which l should be mapped. This can be done for any value of l, leading to the following mapping table indicating how to map l to m.

l	m	$n(m)$
0	0	31
1	0	
2	0	
3	1	25
4	2	21
5	3	11
	4	0
	5	0
6	6	29
7	7	4

The third column indicates the histogram of the resulting image. As one can see the matching is — though not perfect — acceptable given the coarse quantization. For your visual amusement the transformed image has been depicted in Figure 5.16.

Exercises

Exercise 5.4.2.2-1: Consider a 20 × 20 picture, quantized using 4 bit. The histogram data of the intensity l can be found below:

l	n_l	l	n_l	l	n_l	l	n_l
0	36	4	17	8	18	12	37
1	32	5	14	9	19	13	30
2	25	6	16	10	23	14	25
3	20	7	17	11	57	15	14

Determine the intensity mapping required to uniformize the histogram of this picture.

Exercise 5.4.2.2-2: Consider a 20×20 picture, quantized using 4 bit. The histogram data of the intensity l can be found below:

l	n_l	l	n_l	l	n_l	l	n_l
0	1	4	41	8	11	12	31
1	11	5	35	9	1	13	41
2	21	6	31	10	11	14	51
3	31	7	21	11	21	15	41

Determine the intensity mapping required to uniformize the histogram of this picture.

Exercise 5.4.2.2-3: Take the histogram data of the two previous exercises and create a mapping table to match the histogram of the first picture to the histogram of the second picture and vice versa.

Exercise 5.4.2.2-4: (*) Take one of your favorite holiday pictures.

1. create a grayscale version of it (in MATLAB: `im2gray`),
2. calculate its intensity histogram,
3. calculate the mapping table to map it onto a uniform histogram, and
4. execute the mapping and check your result.

Exercise 5.4.2.2-5: (*) Take a picture of your favorite pet.

1. create a grayscale version of it (in MATLAB: `im2gray`),
2. calculate its intensity histogram,
3. calculate the mapping table to map it onto a uniform histogram, and
4. execute the mapping and check your result.

Exercise 5.4.2.2-6: (*) Take the histogram data of the two previous exercises and create a mapping table to match the histogram of the first picture to the histogram of the second picture.

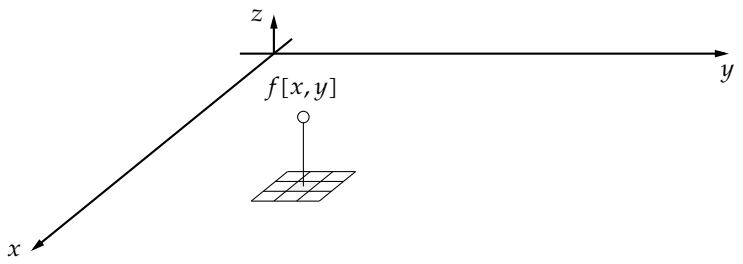
5.4.3 Spatial intensity transformations

So far, in section 5.4.2 on page 67, we discussed *pointwise intensity transformations*. This means that the new grayscale of a particular pixel (obtained by transformation) is only function of the original grayscale value of the pixel itself. We'll now go further and broaden the concept of intensity transformations to *spatial* intensity transformations.

This way of processing images is a very common one.

5.4.3.1 Principle

Consider a pixel with coordinates $[x, y]$ in the figure below. The intensity of the image to which this pixel belongs can be described using an intensity function $f[x, y]$.



Spatial intensity transformation A spatial intensity transformation F calculates a new intensity value $g[x, y]$ for a pixel with coordinates $[x, y]$, based on the intensity values f of the corresponding pixel and its neighboring pixels:

$$g[x, y] = F\left(\{f[x + u, y + v] \mid u \in [-a : a], v \in [-b : b]\}\right).$$

In theory the neighborhood that is taken does not need to be symmetrical, but in practice it often is. Therefore we used the symmetrical ranges $[-a, a]$ and $[-b, b]$ in the definition above.

5.4.3.2 Nonlinear spatial transformations and order-statistic filters

If F is a nonlinear function of its arguments, we label the transformation as *nonlinear*. As the function F may assume very exotic forms, it does not make a lot of sense to treat these kinds of filters in more detail from a theoretical point of view.

However, we want to mention a very common set of filters in this category: the *order-statistic filters*. We distinguish three flavors:

- a min filter: F is the minimum-function
- a max filter: F is the maximum-function
- a median filter: F is the median-function

These filters are very often used to filter out salt-and-pepper noise in an image. The min-filter suppresses salt noise (white spots on a grayscale image) but amplifies pepper noise (black

spots on a grayscale image), the max-filter suppresses pepper noise but amplifies salt noise and the median filter suppresses both. Figure 5.17 illustrates this.

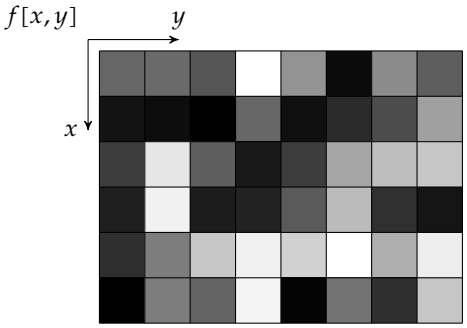
Keep in mind that these filters not only filter away the noise, but they actually alter the image. So, use with care.

Exercises

Exercise 5.4.3.2-1: Consider the image fragment below. Apply a 3 × 3 Min-, Median-, and Max-filter to it.

$f[x,y]$ (y →, x ↓)

102	106	86	255	147	11	139	94
19	13	0	103	15	43	76	160
61	230	94	25	60	166	190	199
31	241	28	34	90	187	48	21
47	125	199	240	209	255	175	237
0	125	99	244	4	115	47	198



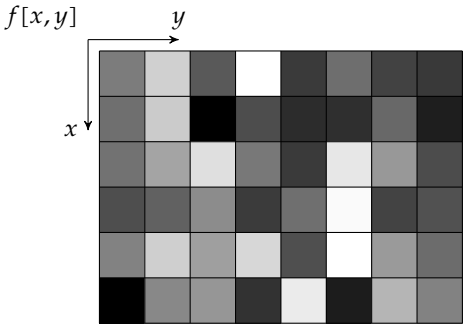
Assume that the image fragment extends symmetrically beyond its borders. This has been illustrated for the top-left edge of the image fragment:

135	120	120	135	0
79	145	145	79	176
79	145	145	79	176
135	120	120	135	0
42	3	3	42	115

Exercise 5.4.3.2-2: Consider the image fragment below. Apply a 3 × 3 Min-, Median-, and Max-filter to it.

$f[x,y]$ (y →, x ↓)

124	208	89	255	58	110	66	57
111	203	0	77	44	47	104	30
114	164	223	120	58	231	152	76
78	97	140	59	111	250	67	81
130	207	159	215	79	255	154	108
0	136	150	50	235	28	181	130



Assume the image fragment to be zero padded beyond its borders.

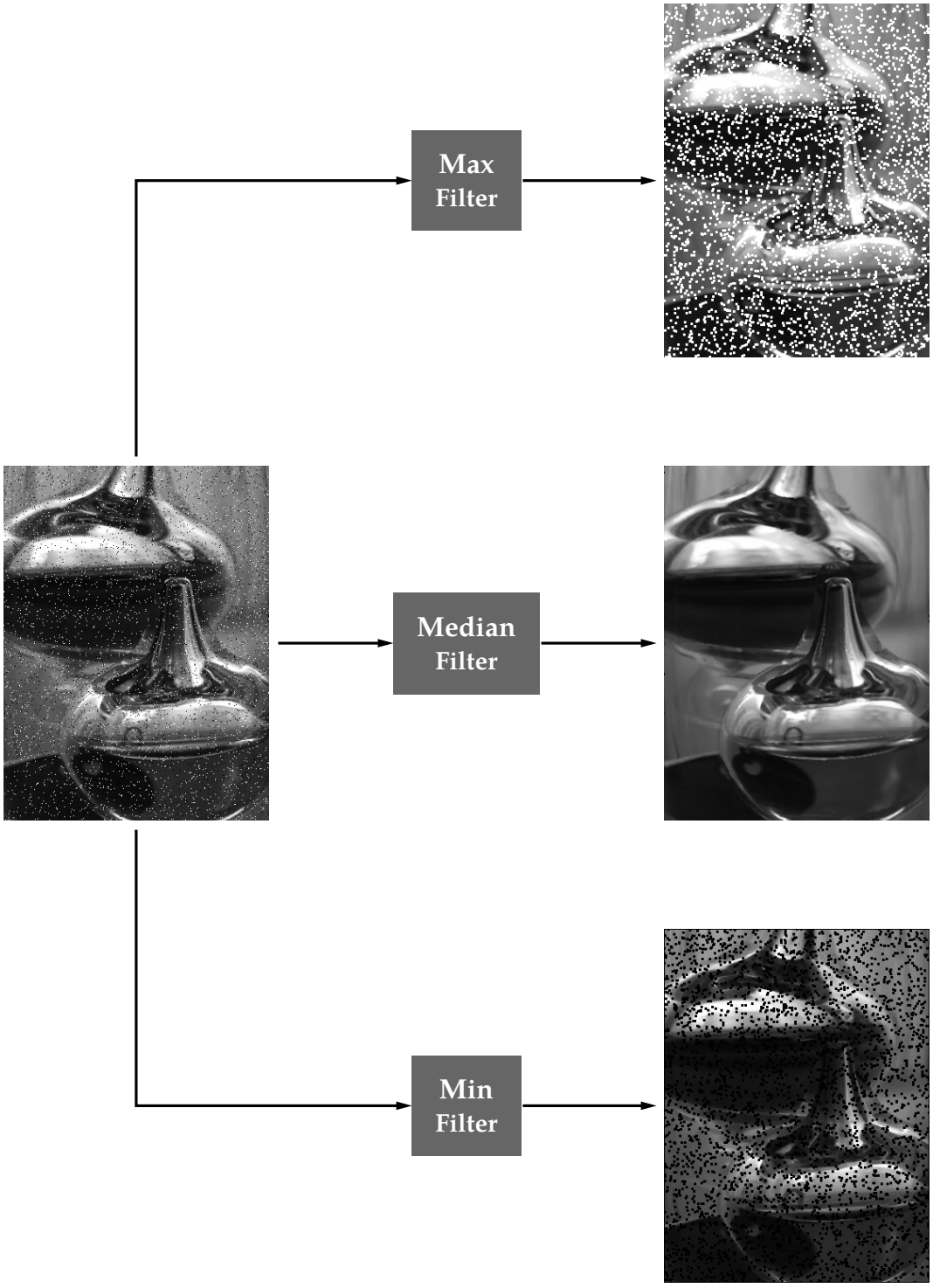


Figure 5.17: Illustration of the action of a 3 × 3 min-filter, max filter and median filter on a 320 × 428 test image (source: Wikimedia commons, user Marko Meza) containing salt-and-pepper noise.

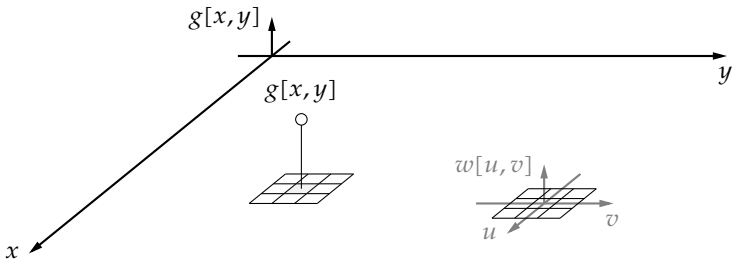
Exercise 5.4.3.2-3: Use MATLAB to apply the order-statistics filters on the image fragment above. To this end, use the `ordfilt2` function.

Exercise 5.4.3.2-4: (*) Search for some images containing salt-n-pepper noise. Apply a Min-, Median- and Max-filter to explore the results. Try varying the size of the filter's footprint.

5.4.3.3 Linear spatial transformations

If F is a linear function of its arguments, we label the transformation as *linear*. This means that the new pixel intensity is a linear combination of the original intensities of the pixel and its neighbors.

To formally describe this new situation, we define a *kernel* $w[u,v]$ that is an image itself containing the desired weighing factors as pixels intensities. This has been illustrated below:



The transformation can then be described as:

$$g[x,y] = \sum_{u=-a}^{a} \sum_{v=-b}^{b} w[u,v] f[x+u, y+v].$$

In theory the neighborhood that is taken does not need to be symmetrical, but in practice it often is. Therefore, we used the symmetrical ranges $[-a,a]$ and $[-b,b]$ in the definition above.

This corresponds to overlaying the original image in the pixel under consideration with the kernel and multiplying the intensities cell by cell and adding them.

In fact, this way of working has been formalized in two image operators: correlation and convolution.

Correlation

Definition

Correlation of two images Given two images described by their intensity functions $w[x,y]$ and $f[x,y]$ we define the correlation $f \star w$ of these two images to be a new image g defined as

$$g[x,y] = (f \star w)[x,y] = \sum_{u=-a}^{a} \sum_{v=-b}^{b} f[x+u, y+v] \cdot w[u,v].$$

The notation $g[x,y] = f[x,y] \star w[x,y]$ is also used in practice. If f is an original image, then w is often called *the (correlation) kernel*.

Calculation example Take a look at Figure 5.18a. Consider an image f that has to be correlated with a kernel w and let's assume that we focus on a specific pixel $[x_1, y_1]$. In the figure the correlation kernel w is shown on the right. On the left we show a selection of the entire image f that is as big as the correlation kernel w, centered around the pixel $[x_1, y_1]$. We need to multiply the corresponding cells (two corresponding cells have been indicated using a dashed arc) and add all these values together to obtain the result of the correlation for pixel $[x_1, y_1]$. We need to repeat this procedure for every pixel.

Therefore:

$$\begin{aligned} g[x_1,y_1] &= \sum_{u=-a}^{a} \sum_{v=-b}^{b} f[x_1+u, y_1+v] \cdot w[u,v] \\ &= (-2)\cdot 1 + 0 \cdot 2 + (-1)\cdot 3 + 3 \cdot 4 + (-4)\cdot 1 + 1 \cdot (-2) + 2 \cdot 0 + (-1)\cdot 3 + 2 \cdot (-1) \\ &= -4 \end{aligned}$$

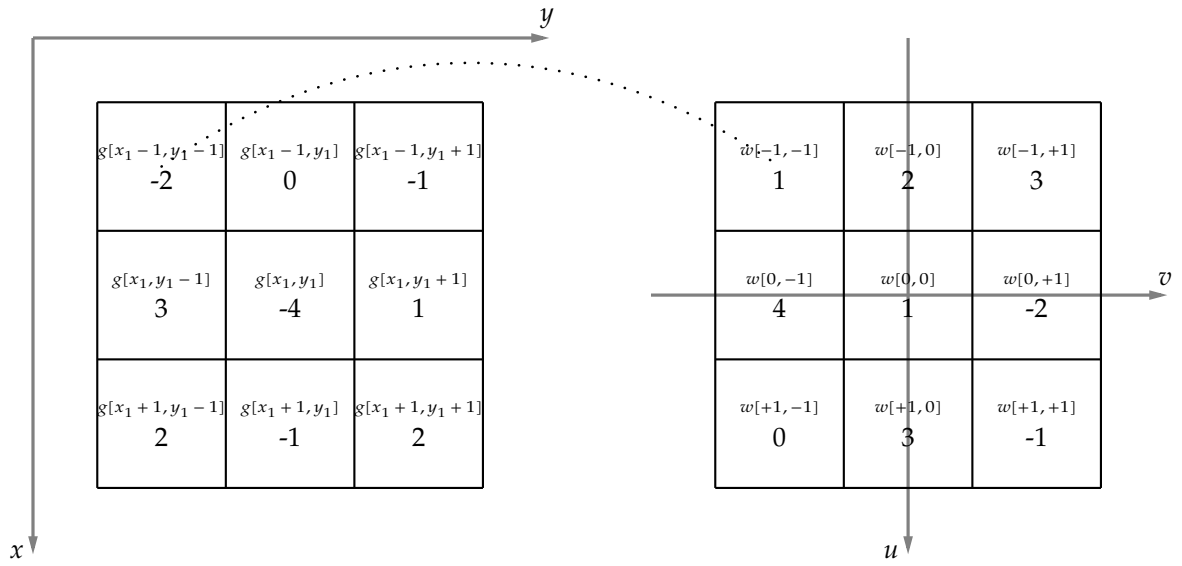
Computational complexity How many calculations do we need to schedule on our processor to calculate the correlation of an image with a specific kernel?

If we make abstraction of memory issues (cache performance may be an issue), and restrict ourselves to calculating the mathematical operations, an approximate closed formula can be given for an $M \times N$ image that is being correlated with a kernel containing K nonzero elements:

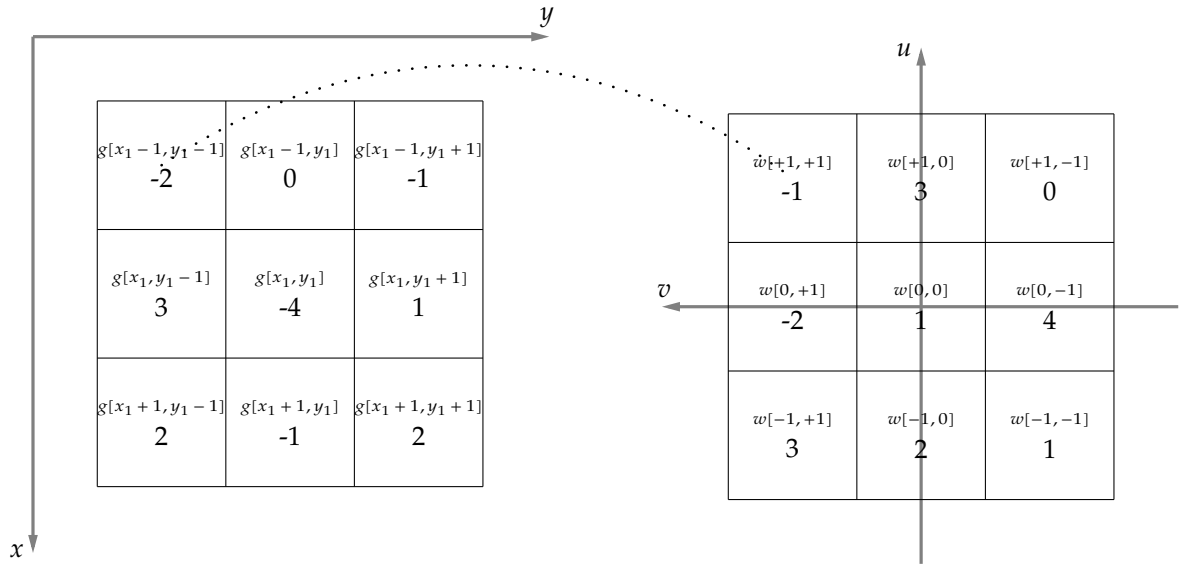
$$T_{CORR} = M \cdot N \cdot K \cdot (T_{MULT} + T_{ADD})$$

with T_{MULT} and T_{ADD} the respective times required to execute a multiplication and an addition.

Why 'approximate'? The perimeter of the image requires special consideration. The section with remarks given below, goes into more detail. Also per pixel, one actually needs one addition less.



(a) Correlation



(b) Convolution

Figure 5.18: Illustration of how a two-dimensional correlation/convolution is calculated

Convolution

Definition

Convolution of two images Given two images described by their intensity functions $w[x,y]$ and $f[x,y]$ we define the convolution $f \star w$ of these two images to be a new image g defined as

$$g[x,y] = (f \star w)[x,y] = \sum_{u=-a}^{a} \sum_{v=-b}^{b} f[x-u, y-v] \cdot w[u,v]. \tag{5.4}$$

Note the subtle sign difference with the correlation operation. The notation $g[x,y] = f[x,y] \star w[x,y]$ is also used in practice. If f is an original image, then w is often called *the (correlation) kernel*.

Calculation example Take a look at Figure 5.18b on the preceding page. Consider an image f that has to be convoluted with a kernel w and let's assume that we focus on a specific pixel $[x_1, y_1]$. In the figure the convolution kernel w is shown on the right. Note that it has been rotated over 180°! This corresponds to the effect of the minus signs in (5.4). On the left we show a selection of the entire image f that is as big as the convolution kernel w, centered around the pixel $[x_1, y_1]$. We need to multiply the corresponding cells (two corresponding cells have been indicated using a dashed arc) and add all these values together to obtain the result of the convolution for pixel $[x_1, y_1]$. We need to repeat this procedure for every pixel.

$$\begin{aligned} g(x_1,y_1) &= \sum_{u=-a}^{a} \sum_{v=-b}^{b} f[x_1-u, y_1-v] \cdot w[u,v] \\ &= (-2)\cdot(-1) + 0\cdot 3 + (-1)\cdot 0 + 3\cdot(-2) + (-4)\cdot 1 + 1\cdot 4 + 2\cdot 3 + (-1)\cdot 2 + 2\cdot 1 \\ &= 2 \end{aligned}$$

Computational complexity How many calculations do we need to schedule on our processor to calculate the convolution of an image with a specific kernel?

For this, we refer to the correlation case, which is identical.

Remarks

- Strictly speaking, the summation boundaries do not need to be symmetrical, but this is often the case; the dimensions of the kernels may vary, the kernel does not even have to be rectangular in shape.
- For the pixels on the perimeter of an image, the problem arises that out-of-domain pixels are referenced; in this case
  - the image is extended (e.g., by duplicating perimeter pixels, or by adding pixels with intensity value zero), or
  - the resulting image is a few pixels smaller than the original.
- Note that in the case of a symmetrical kernel, the correlation and convolution operations give identical results.
- Though we introduce the concept of two-dimensional correlation and convolution in the frame of linear spatial transformations, these concepts are often seen as *filtering operations*. Therefore the kernels are also referred to as *filter kernels*.

Examples As examples of linear spatial transformations (or filtering), let's take a look at two examples:

1. a moving average filter,
2. a Laplacian filter.

Moving average filter A very simple filter is the moving average filter (a smoothing filter). The corresponding filter kernel is:

$$w[u,v] = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}.$$

The result of convolving or correlating the image with this kernel is given in Figure 5.19.

Laplacian Another simple filter is the Laplacian filter (an edge detection filter).

$$w[u,v] = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

The result of convolving or correlating the image with this kernel is given in Figure 5.19.

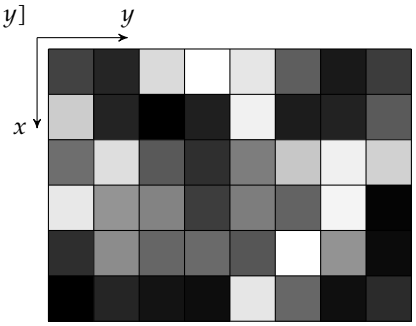
Exercises

Exercise 5.4.3.3-1: Consider the image fragment below. Apply a 3 × 3 moving average filter to it

$f[x,y]$ y

66	37	218	255	230	94	25	60
204	35	0	31	241	28	34	90
110	222	89	47	125	199	240	209
232	148	131	61	125	99	244	4
46	140	102	106	86	255	147	11
0	37	19	13	230	103	15	43

x



Assume the image fragment to (a) extend symmetrically and, as a separate exercise, (b) to be zero padded beyond its borders. Check Exercise 5.4.3.2-1 if you are unsure how to extend an image fragment symmetrically.

Exercise 5.4.3.3-2: Consider the image fragment of the previous exercise. Apply a Laplacian filter to it. Assume the image fragment to (a) extend symmetrically and, as a separate exercise, (b) to be zero padded beyond its borders. Check Exercise 5.4.3.2-1 if you are unsure how to extend an image fragment symmetrically.

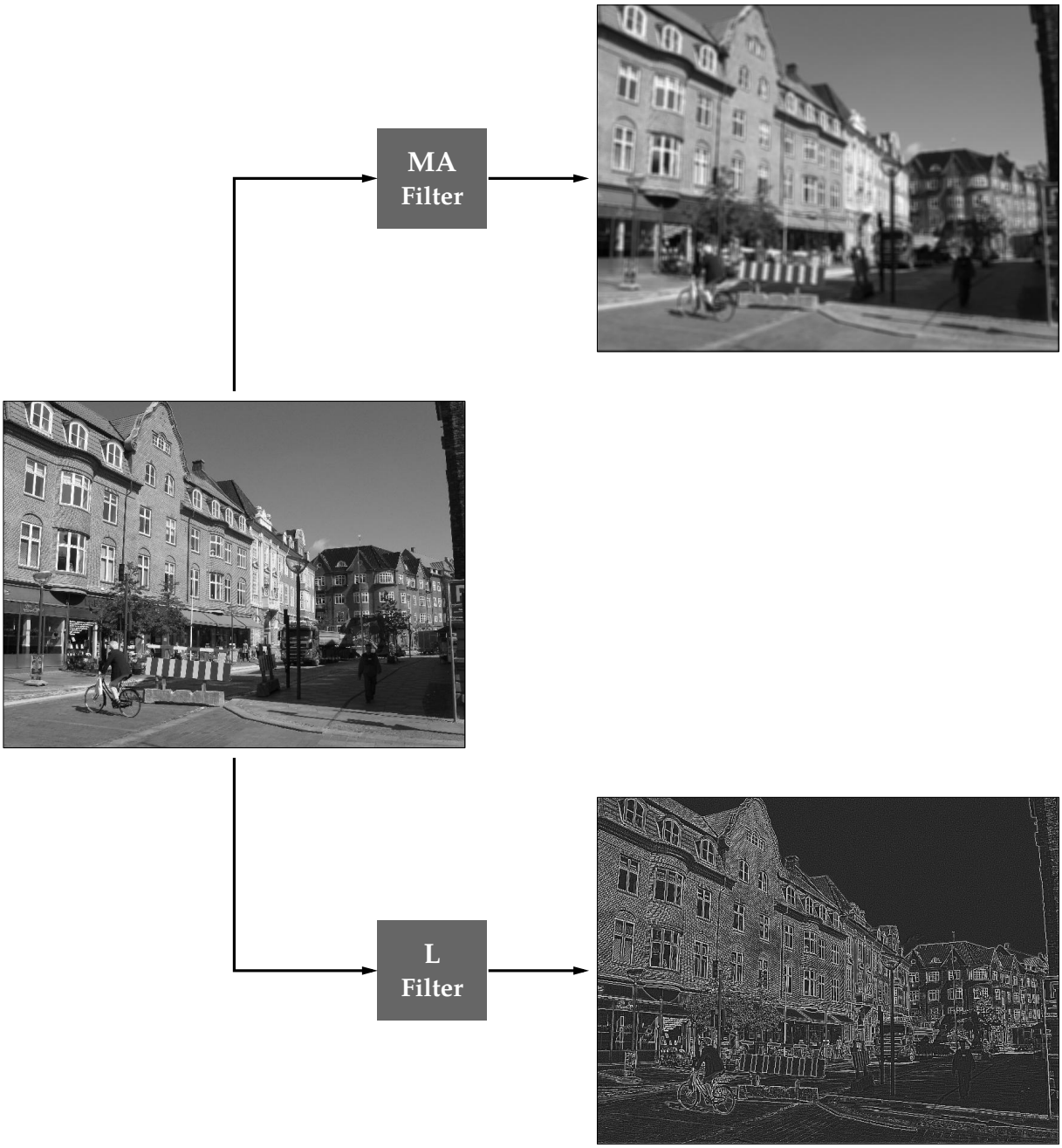


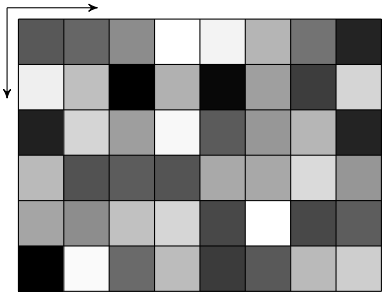
Figure 5.19: Illustration of the effect of some practical linear spatial transformations on a test image (source: Walter Daems): a moving average filter (top) and a Laplacian filter (bottom)

Exercise 5.4.3.3-3: Use MATLAB/OCTAVE to apply the filter kernel $h[x,y]$ to the image $f[x,y]$, (a) using correlation, and (b) using convolution.

$f[x,y]$ ($y \rightarrow$, $x \downarrow$)

88	102	140	255	243	181	115	35
239	191	0	177	8	159	61	213
32	213	158	248	91	151	182	35
186	82	92	84	169	168	218	150
165	141	193	214	72	255	72	93
0	250	106	188	59	89	186	206

$f[x,y]$ ($y \rightarrow$, $x \downarrow$)



$$h = \begin{bmatrix} 0.24 & -0.31 & 0.50 & -1.00 & 0.44 \\ -0.23 & 0.50 & \boxed{1.00} & 0.24 & -0.45 \\ -1.00 & 0.23 & 0.00 & 0.71 & -0.12 \end{bmatrix}$$

Assume the image fragment to extend symmetrically beyond its borders. Check Exercise 5.4.3.2-1 if you are unsure how to extend an image fragment symmetrically.

Chapter 6

Light and Color Modeling

In this chapter, you will learn:

- what light is,
- what color is,
- how we model both, and
- that we can transform color components to our benefit.

After having read/studied this chapter, you are expected to be able to

- understand and explain the nature of light,
- perform calculations with light quantities,
- understand and explain the nature of color and how we model it, and
- work with the available color models.

6.1 Light

6.1.1 What is light?

Light is a *quantized transversal electromagnetic wave*. Stated differently: light consists of small energy packets, so-called *photons* that behave like waves whose electromagnetic and electric field vector define a plane that is perpendicular to the direction of motion.

The energy content of a photon is related to its wave properties according to the *Planck-Einstein relation*:

$$E = h\nu$$

in which E represents the energy content, ν represents the frequency of the wave and h is Planck's constant defined to be:

$$h = 6.626 \times 10^{-34}\,\mathrm{J\,s}$$

In most conditions — while traveling through space — a photon will not be altered with respect to its wave properties, it will not even lose energy on its trajectory through space. Such a trajectory is called a ray. The process of calculating how light travels across space is called *ray tracing*. In many cases light travels along straight lines, making ray tracing a feasible job.

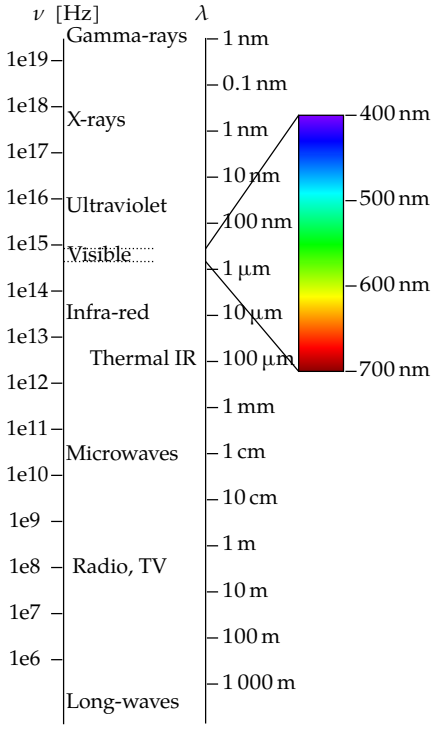


Figure 6.1: Electromagnetic spectrum (image source: Victor Blacus)

When traveling, a photon has wavelike behavior and therefore like every wave exhibits a wavelength λ, related to its frequency and the speed of light. From wave theory we know that:

$$\lambda = \frac{c}{\nu}$$

in which c represents the speed of light. In vacuum the value of c amounts to 299 792 458 m/s, commonly approximated as 3×10^8 m/s. This value is also a good approximation for the speed of light in our atmosphere.

Of course, photons rarely travel solitary. They usually come in bundles. A fraction of such a bundle might get caught in a medium (absorbed), i.e. the energy of the caught photon is converted to another energy form, e.g., its energy may be converted to heat or increase the energy state of other particles (e.g., electrons).

If we categorize electromagnetic waves (in vacuum) according to their frequency or wavelength, we obtain the spectrum of Figure 6.1. Light is only a small subset of that entire spectrum. Visible light ranges from 400 nm (violet) to 700 nm (red). The neighbors of that range (ultra violet, below 400 nm, and infra red, above 700 nm are often also considered to be (non visible) light. It might be obvious to you that we use colors to name the different wavelengths. However, if you're stupefied by that fact, bear a little longer: it will become clear when discussing the human eye in section 6.2 on page 110.

We will not dive into relativistic quantum mechanics or in Maxwell's laws to study light from a theoretical perspective. Instead, we will take a practical approach in order to get the understanding we need for lighting scenes. A good lighting is required to make decent images.

To this end, consider the scene of Figure 6.2: a lighting device L produces light to illuminate an object or a surface S that reflects the light diffusely to a sensor.

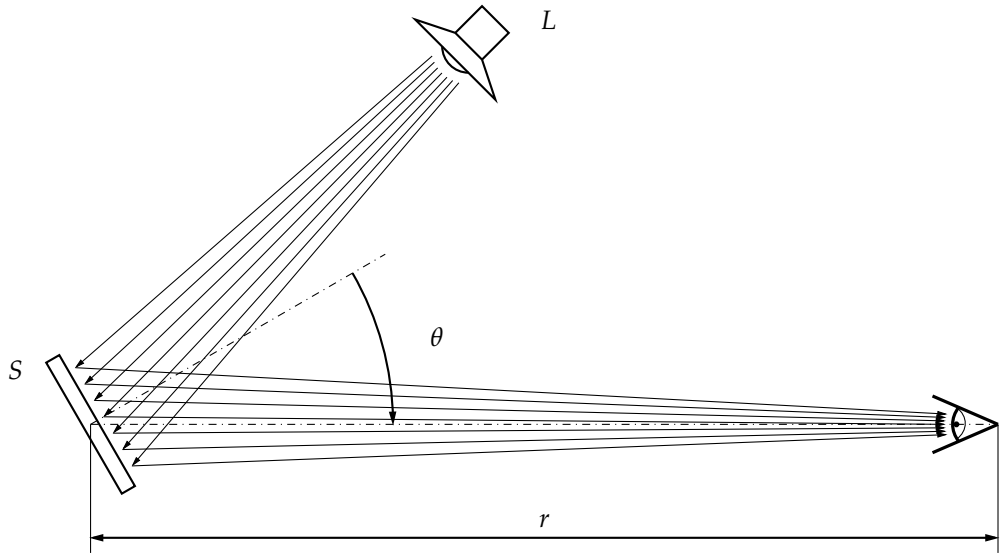


Figure 6.2: Lighting scene to illustrate the path the light travels from lighting device to sensor

The path the light travels can be described in two sets of terms:

1. from the perspective of the physics: using *radiometric quantities*,
2. from the perspective of how we perceive light: using *photometric quantities*.

Because of the fact that light quantities are often misnamed and misunderstood, it makes sense to pay special attention to units, as they can be your guideline in remembering the correct equations and checking whether your calculations make sense. We will therefore note the units of the equations next to them.

6.1.2 Describing light in radiometric quantities

Light is radiated by a *light source*. This source converts a different form of energy (fuel, electricity, ...) to light energy. In response to being fed with a power P[W], the source emits a stream of light Φ_e[W], the so-called *light flux*.

This conversion process is not perfect, i.e. not all incoming energy is radiated as light. Some of it gets lost in other forms of energy (e.g., heat). Therefore:

$$\Phi_e = P_{eff} = \eta_e P \qquad [\mathrm{W}]$$

with η_e *radiant efficiency*. Energy conservation dictates that $0 \leq \eta_e \leq 1$.

The source may emit light over a range of wavelengths. Therefore, it makes sense to consider the distribution of the radiant flux over the different wavelengths. The total flux is the integral of the detailed distribution $\partial\Phi_e/\partial\lambda$:

$$\Phi_e = \int_0^\infty \frac{\partial \Phi_e}{\partial \lambda}\, \mathrm{d}\lambda \qquad [\mathrm{W}] \tag{6.1}$$

Why would we bother considering the detailed wavelength distribution of a source? Well, we see different wavelengths as different colors. In that way, the flux versus wavelength distribution is also a color distribution.

6.1.3 Describing light in photometric quantities

Luminous flux and efficacy Light is a special electromagnetic wave in the sense that we have a set of particularly good sensors for it: our eyes. Therefore it makes sense to describe light in terms of how we *see* it.

The starting point of this is the fact that our sensor is not independent of the wavelength: we see some wavelengths better than others, i.e. our eyes have a sensitivity that depends on the wavelength of the light.

This can be embedded into (6.1), using a sensitivity function $S(\lambda)$ that gives a weight to every wavelength. This gives birth to a new quantity, the perceived light flux, or *luminous flux*:[1]

$$\Phi_v = K_m \int_0^\infty \frac{\partial \Phi}{\partial \lambda} S(\lambda)\, \mathrm{d}\lambda \qquad [\mathrm{lm}]$$

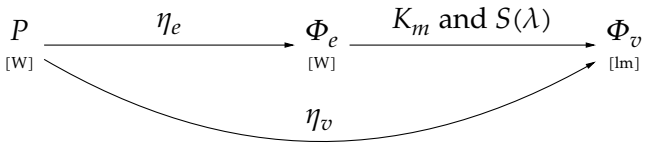
in which K_m is a conversion factor that depends on how we use the sensors in our eyes:

$$K_m = \begin{cases} 683\,\mathrm{lm/W} & \text{(photopic vision)} \\ 1\,699\,\mathrm{lm/W} & \text{(scotopic vision)} \end{cases}$$

We will reveal the meaning of the terms *photopic* and *scotopic vision* in section 6.2 on page 110.

Note the new unit *lumen* (symbol: lm), which is nothing more than a converted version of the unit Watt (symbol: W). Whereas two light sources of the same radiant flux (in Watt) can give us a different impression of light intensity, depending on their color distribution, two light sources of the same luminous flux (in lumen), will be perceived as equally intense.

An oversimplified view on things can be found below.



Instead of walking the straight path from P over Φ_e to Φ_v, one can also describe the relationship between those two quantities using a direct factor, the so-called luminous efficacy η_v:[2]

$$\eta_v = \frac{\Phi_v}{P} \qquad [\mathrm{lm/W}]$$

Note that this factor covers both the effect of K_m, the sensitivity function $S(\lambda)$ and the actual wavelength (color) distribution of the light source. It describes how effective the source is at producing visible light. This makes η_v a very useful factor to compare, e.g., different light bulbs or LEDs.

Typical values are:

[1] The subscript 'v' stands for 'vision'.

[2] Note that the term efficiency is not used here, because the quantity is not dimensionless, and not bounded to the interval $[0, 1]$.

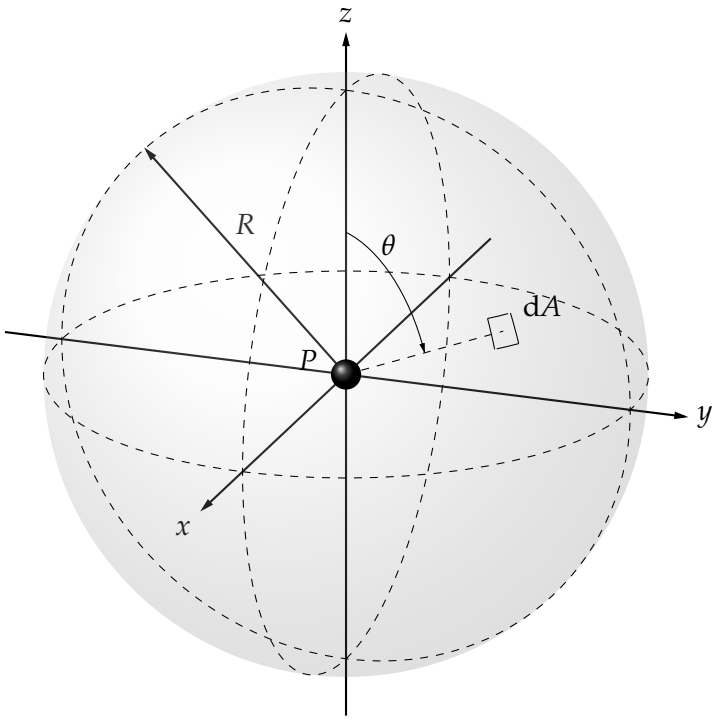


Figure 6.3: Illustration of a point radiator P sending out its energy radially

Device	η_v [lm/W]
Conventional light bulb	15
Halogen light bulb	15 – 25
Arc lamp	30 – 80
LED lamp	25 – 100
Fluorescent lamp	45 – 100
GAS discharge lamp	50 – 200

The table above might be worth considering when you plan to create a plan for the lighting of your new house or office building.

Luminous intensity A light source is often small when compared to the distance to the observer. Therefore, we often model a source as a *point source* (infinitely small) that emits its energy radially. This has been illustrated in Figure 6.3.

The radiated light flux distributes itself over an imaginary ball surface centered at the point source. The distribution of the flux over this surface leads to the idea of flux density, the density of the elementary amount of flux $\partial\Phi_v$ that flows through an elementary amount of area ∂A:

$$E_v = \frac{\partial\Phi_v}{\partial A}$$

One can view this as the number of flux lines that punch through the sphere per unit of area. However, the issue with this metric is that it is dependent on the radius of the sphere. If its radius R doubles, the flux density diminishes by a factor of 4.

Therefore — as for every problem that has radial properties — it makes sense to normalize w.r.t. the radius of the sphere. The key here is to use the concept of solid angle Ω, i.e.:

$$\Omega = \frac{A}{R^2}$$

If one considers a unit sphere (i.e. $R = 1$) then the concepts of area and solid angle are the same.[3]

This leads to studying the flux per unit of solid angle Ω, the so-called *luminous intensity*:

$$I_v = \frac{\partial \Phi_v}{\partial \Omega} \qquad [\text{cd}] = [\text{lm/sr}]$$

The unit lm/sr is commonly abbreviated as the *candela* (symbol: cd).

For a uniformly radiating source this becomes:

$$I_v = \frac{\Phi_v}{4\pi} \qquad [\text{cd}] = [\text{lm/sr}]$$

since the surface area of a unit sphere is 4π.

Claiming that no source exists that radiates uniformly would be an overstatement (e.g., stars do so more or less). However, it must be said that human-made lighting sources are rarely uniform. In many cases we even add reflectors to guide the light in a specific direction!

One could therefore assume that every technical lighting source comes with a complex formula describing the I_v as a function of your position w.r.t. the lighting source. However, for real-world light sources and the fixtures they come in, these formulae would be overly complex. Luckily in many cases the directional radiation pattern has a certain symmetry. This allows us to describe I_v as a one-dimensional graph. An example of such a graph can be found in Figure 6.4. We will learn how to use these diagrams in section 6.1.4.

Illuminance Now that we know how our electricity is converted to light and how the light of a point source is radiated onto our scene, we might wonder, how well is our scene lit? To light a scene, we're interested in the light density, i.e. the amount of light flux per square meter that reaches a surface that we want to light. We call this quantity the *illuminance*:[4]

$$E_v = \frac{\partial \Phi_v}{\partial A} \qquad [\text{lx}] = \left[\text{lm/m}^2\right]$$

Note that the illuminance describes the incident light, but not how the light is reflected from the surface we want to be lit. Therefore, the material properties of the surface are of no consequence when discussing illuminance! The surface might even be imaginary.

Typical illumination values can be found in the table below:

[3] This is true except for the units: solid angle is dimensionless, often referred to as *steradians* (symbol: sr), whereas the dimension of area is the square meter (symbol: m^2).

[4] If you wonder why the lighting symbols are so illogical (E for illumination?), think French: éclairage.

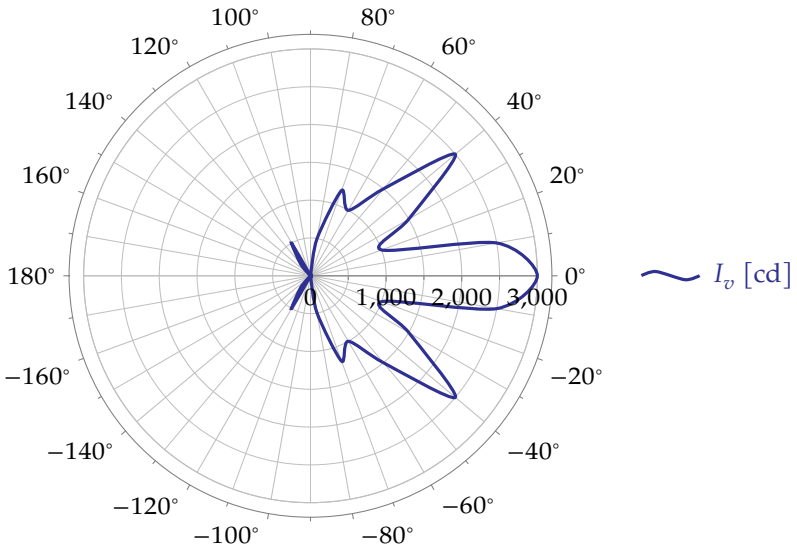


Figure 6.4: Example of a luminous intensity distribution pattern

Condition	Illuminance E_v [lx]
Bright sun	100 000
Full daylight (not direct sun)	10 000
Overcast day	1 000
Very dark overcast day	100
Twilight	10
Dark Twilight	1
Full moon	0.1
First/third quarter moon	0.01
New moon	0.001
overcast night	0.000 1
Living area	100
Office area	500
Workshops	1 000

It makes sense to learn some of these values by heart such that you have some reasonable idea about the values you may encounter in different situations.

Reflectivity - Luminous exitance So far, so good, our surface is lit. The next question is: how much light per unit of area is reflected? We call this quantity the *luminous exitance* and denote it with M_v. Of course, this is a function of the reflectivity of the surface. We can describe this using a reflectivity degree ρ:

$$M_v = \rho E_v \qquad \left[\mathrm{lm/m^2}\right]$$

with $0 \leq \rho \leq 1$.

The parameter ρ clearly is a surface material parameter tied to a particular material and the way its surface has been machined. Though one could abbreviate $\mathrm{lm/m^2}$ to lx, this is *never*

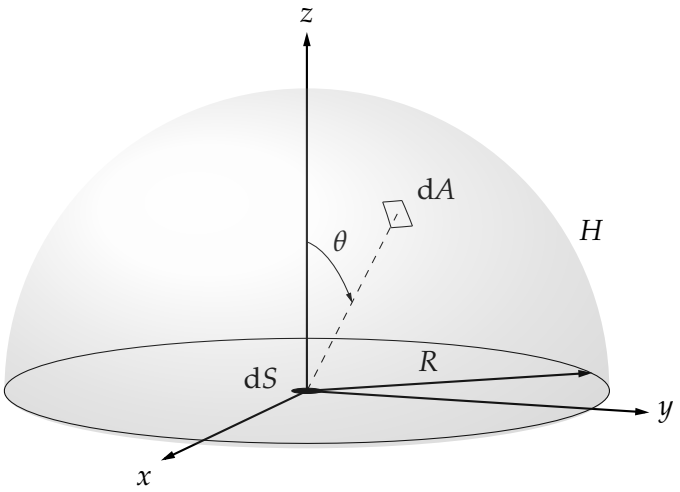


Figure 6.5: Illustration of a point radiator $\mathrm{d}S$ radiating through the upper hemisphere and generating a Lambertian flux density through $\mathrm{d}A$

done in the case of the luminous exitance.

It should be noted that ρ can be a function of wavelength and therefore the exitance could be more accurately defined as an integral quantity.

It may also occur that the surface is producing light itself. Examples of this are *displays* ((like LCD, (O)LED or plasma displays). In that case we also call M_v the *luminous emittance*.

Lambert's law Finally, as key background information before tackling the next quantity, it is important to understand diffuse reflection/emission.

For surfaces that are not highly polished and have no regular relief pattern, the reflection is not related to the direction of the incident light. Every point on the surface behaves like a point radiator. Due to the symmetry of the setup, the radiation pattern must have a circular symmetry. In many cases a Lambertian emission pattern is observed. Let's see what that means.

Take a look at Figure 6.5. Consider a point (or an infinitesimally small circle) $\mathrm{d}S$ on a surface, and consider an imaginary hemisphere H with radius R centered in that point. Given the emittance of the surface, the point with surface $\mathrm{d}S$ generates a flux

$$\phi_t = M_v \, \mathrm{d}S. \tag{6.2}$$

That flux cannot but go through the hemisphere above. However, according to *Lambert's law* the flux $\mathrm{d}\phi$ going through an arbitrary infinitesimally small area $\mathrm{d}A$ on the hemisphere H is dependent on the polar angle θ:

$$\mathrm{d}\phi = \phi_t \frac{\cos\theta}{\pi R^2} \, \mathrm{d}A$$

and therefore the flux density equals:

$$E_v = \frac{\partial \phi}{\partial A} = \phi_t \frac{\cos\theta}{\pi R^2}$$

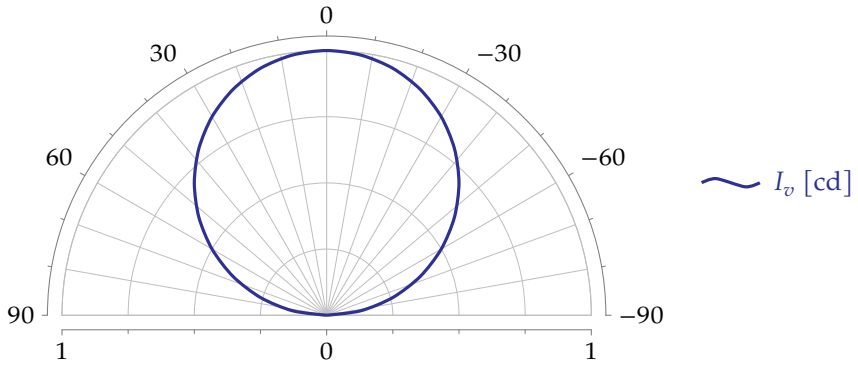


Figure 6.6: Illustration of the Lambertian luminous intensity pattern

One can easily prove by integration that the integral of this flux density over the hemisphere H equals ϕ_t.

In the equations above, we can exchange spherical area for solid angle if we set $R = 1$. This leads to:

$$\mathrm{d}\phi = \phi_t \frac{\cos\theta}{\pi}\,\mathrm{d}\Omega$$

and therefore the luminous intensity I_v equals:

$$I_v = \frac{\partial\phi}{\partial\Omega} = \phi_t \frac{\cos\theta}{\pi} \tag{6.3}$$

Note that the latter equation is the polar equation of a circle. This means that the luminous intensity is maximal for $\theta = 0$ and decreases to 0 for $\theta = \pi/2$. The maximal luminous intensity when $\theta = 0$ is $I_{v,\max} = \phi_t/\pi$. This Lambertian luminous intensity pattern has been illustrated in Figure 6.6. The luminous intensity axis has been normalized w.r.t $I_{v,\max}$. Real (non-ideal Lambertian) surfaces will have an intensity pattern that deviates from the perfect circular shape of Figure 6.6, but in many cases the circular shape will still be recognizable.

In case the surface is polished and starts behaving like a mirror, a full 3D radiation pattern, dependent on the actual direction of the incident light, will need to be considered.

Luminance We're almost there. Once again, take a look at Figure 6.2 on page 99. Our light source has illuminated a surface. In our setting the surface is considered to be a Lambertian reflecting surface and emits a specific exitance. The final question is: how much of that light will reach my sensor?

Of course, this depends on a lot of factors:

1. the angle θ under which we see the surface: the more we deviate from looking perpendicularly, the less light will reach our eye (due to Lambert's law);
2. the area of the surface: the larger the surface, the bigger the flux it will emit[5];
3. the distance of our eye with respect to the surface: the farther away, the less flux will reach our sensor, because of the fact that our sensor will cover a smaller solid angle when seen from the reflective/emitting surface.

[5]This is under the assumption of a constant flux density.

The idea behind the quantity luminance is to define a metric that is independent of all these factors. The basis to start from is the luminous intensity as described by (6.3) in combination with (6.2). This leads to:

$$I_v = \frac{\partial \phi}{\partial \Omega} = \frac{M_v \, \mathrm{d}S \cos\theta}{\pi}$$

However, though I_v is no function of the distance (factor 3, as listed above), it is still dependent on the emitting area $\mathrm{d}S$ and the observation angle θ.

To avoid this dependence, we define a new quantity L_v, the so-called *luminance* as:

$$L_v = \frac{1}{\cos\theta} \frac{I_v}{\partial S} = \frac{M_v}{\pi} \qquad \left[\mathrm{cd/m^2}\right]$$

Note that this definition is based on a Lambertian radiating surface. Still, the notion of *luminance* is also used for non-Lambertian radiators.

The nice thing about luminance is that it is constant in ideal optical lens systems. Note the unit $\mathrm{cd/m^2}$. Sometimes it is also referred to as the unit *nits*. In theory the unit lx/sr is correct, but it is *never* used.

A few typical luminance values have been listed in the table below:

Condition/Object/Device	Luminance L_v $[\mathrm{cd/m^2}]$
Night sky	0.001
Scene at sunrise or sunset	25
Peak luminance LCD monitor	300
Scene on overcast day	700
Scene in full sunlight	5 000
Clear sky	7 000
Solar disk at horizon	600 000
Solar disk at noon	1 600 000 000

It makes sense to learn some of these values by heart such that you have some reasonable idea about the values you may encounter in different situations.

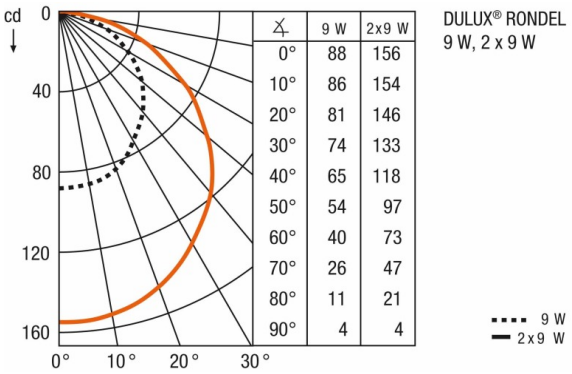
A typical display has a luminance between 50 and 600 $\mathrm{cd/m^2}$.

6.1.4 Calculating the lighting of a scene

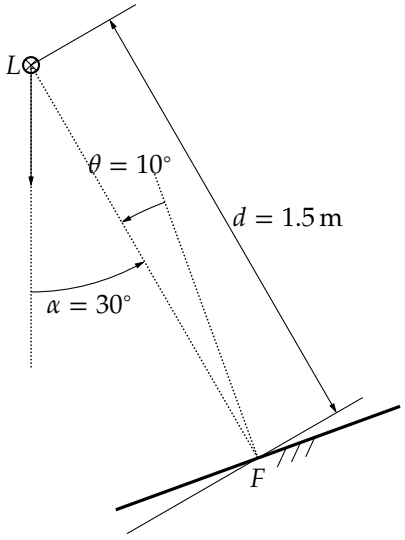
Now that we have some basic understanding of lighting a scene, let's demonstrate our skills on calculating the lighting conditions.

From lamp to scene A lighting system (see Figure 6.7a) to mount on a ceiling generates an intensity profile according to the graph of Figure 6.7b.

(a) Light fixture



(b) Luminous intensity profile of the fixture



(c) Lighting situation

Figure 6.7: An example lighting problem using an OSRAM fixture (source: www.osram.com)

Let's calculate the illuminance in a point F on a distance of $d = 1.5\,\text{m}$ of the lamp body L under an angle of $\alpha = 30°$ for a surface whose normal is angled $\theta = 10°$ with the incident light rays. We assume using a (double) lamp of $2 \times 9\,\text{W}$. The situation has been illustrated in Figure 6.7c.

Let's consider an infinitely small unit of area ∂A_t in the point F on the tilted surface and start with the definition of illuminance:

$$E_v = \frac{\partial \Phi_v}{\partial A_t} \tag{6.4}$$

The relationships

$$I_v = \frac{\partial \Phi_v}{\partial \Omega}$$
$$A_t = \frac{A}{\cos\theta}$$
$$\Omega = \frac{A}{d^2}$$

allow us to rework (6.4) and write:

$$\begin{aligned}
E_v &= \frac{\partial \Phi_v}{\partial A_t} \\
&\downarrow \quad \Rightarrow \partial \Phi_v = I_v \partial \Omega \\
&= \frac{I_v \partial \Omega}{\partial A_t} \\
&\downarrow \quad \Rightarrow \partial A_t = \frac{1}{\cos\theta}\partial A \\
&= \frac{I_v \partial \Omega}{\frac{1}{\cos\theta}\partial A} \\
&\downarrow \quad \Rightarrow \partial A = d^2 \partial \Omega \\
&= \frac{I_v \partial \Omega}{\frac{1}{\cos\theta} d^2 \partial \Omega} \\
&= \frac{I_v}{d^2}\cos\theta
\end{aligned}$$

The luminous intensity I_v can be read from the diagram of Figure 6.7b and determined to be

$$I_v = 133\,\text{cd} = 133\,\text{lm/sr}$$

Therefore:

$$E_v = \frac{I_v}{d^2}\cos\theta = \frac{133\,\text{lm/sr}}{2.25\,\text{m}^2/\text{sr}}\cos 10° = 58.213\,\text{lx}$$

From scene to sensor Consider our scene surface of $1\,\text{m}^2$ that is illuminated at 58.213 lx and has a diffuse reflectivity of $\rho = 0.5$. What is the luminance level of that surface?

Let's start using the definition of *luminance*:

$$\begin{aligned} L_v &= \frac{M_v}{\pi} \\ &= \frac{\rho E_v}{\pi} \\ &= \frac{0.5 \cdot 58.213\,\mathrm{lx}}{\pi} \\ &= 9.265\,\mathrm{cd/m^2} \end{aligned}$$

What will be the luminance level when the surface is observed by a sensor under an angle $\theta = 40°$?

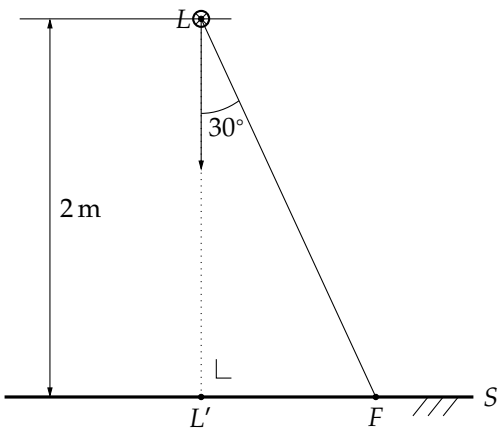
Well, actually, that is not a very well formulated question. The concept of luminance removed the observation angle, therefore the answer is the same: the luminance is $9.265\,\mathrm{cd/m^2}$. The angle is irrelevant for the notion of luminance.

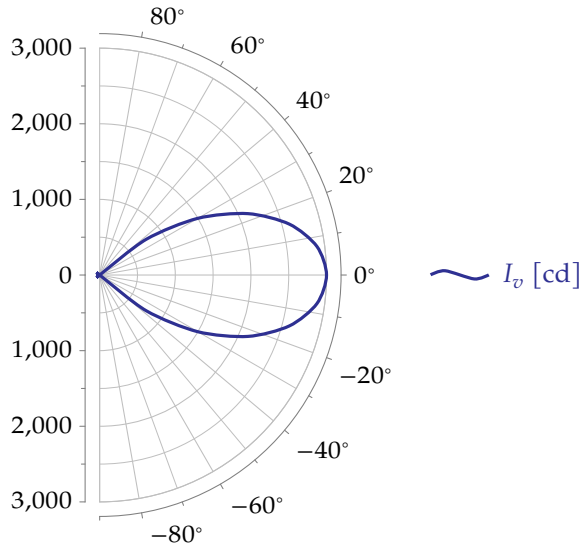
A more correct question could be: what is the luminous intensity when the surface is observed by a sensor under an angle $\theta = 40°$?[6] To be able to answer this question, we assume that the area is sufficiently distant from the observer, such that the observation angle can be considered to be constant. Under that assumption:

$$\begin{aligned} I_v &= AL_v \cos\theta \\ &\downarrow \quad A = 1\,\mathrm{m^2}, \theta = 40° \\ &= 7.097\,\mathrm{cd} \end{aligned}$$

Exercises

Exercise 6.1.4-1: A halogen lamp in a downwards oriented directional reflector, positioned in L, illuminates a surface S. The perpendicular line through L onto S, intersects with S in the point L'.

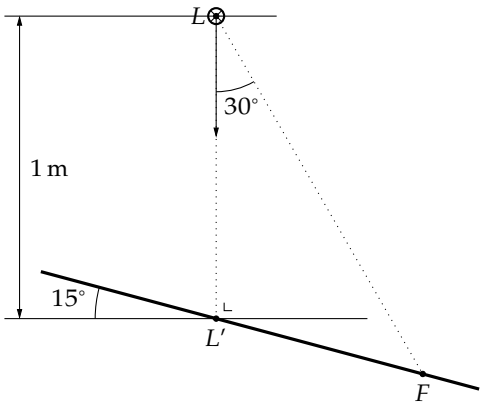


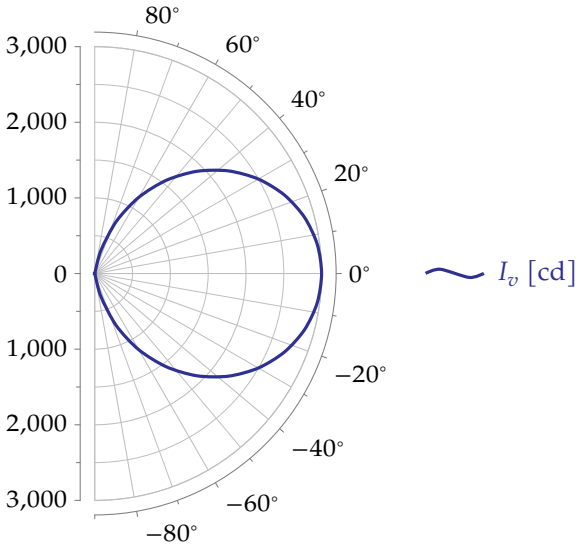


[6] The observation angle is defined as the angle between the incident light rays and the normal on the surface that is to be lit.

The distance between L and L' equals 2 m. The angle between LL' and LF amounts to 30°. The lamp and the reflector cause a luminous intensity I_v as indicated in the figure above. Calculate the illumination of the surface S at the point F.

Exercise 6.1.4-2: A LED lamp with a downwards oriented lens, positioned in L, illuminates a surface S. The downwards line from L intersects with S at the point L'.





The distance between L and L' equals 1 m. The angles are indicated on the figure above. The lamp and the lens cause a luminous intensity I_v as indicated in the figure above on the right. Calculate the illumination of the surface S at the point F.

Exercise 6.1.4-3: A surface is illuminated at 250 lx. Calculate the luminance of the surface assuming perfect diffuse reflection with a reflectivity $\rho = 0.23$.

Exercise 6.1.4-4: An LC Display of 4 in^2 has a maximal luminance of 350 cd/m^2. What is the radial flux density at a distance $R = 1.5$ m if the viewing angle $\theta = 57°$ assuming the display is a Lambertian radiator.

6.2 The human eye

The human eye is a great sensor. It is not our ambition to give a thorough anatomical and sensory treatment of the human eye. That would go far beyond your and my expertise. However, a basic understanding of how it is built and what elements determine its operation is required to understand basic image processing aspects, e.g., color.

6.2.1 Composition

A cross section of the human eye can be found in Figure 6.8. The eye consists of two parts, the *anterior chamber* (spherical in nature, with a smaller diameter) and the *posterior chamber* (also spherical in nature, but with a larger diameter).

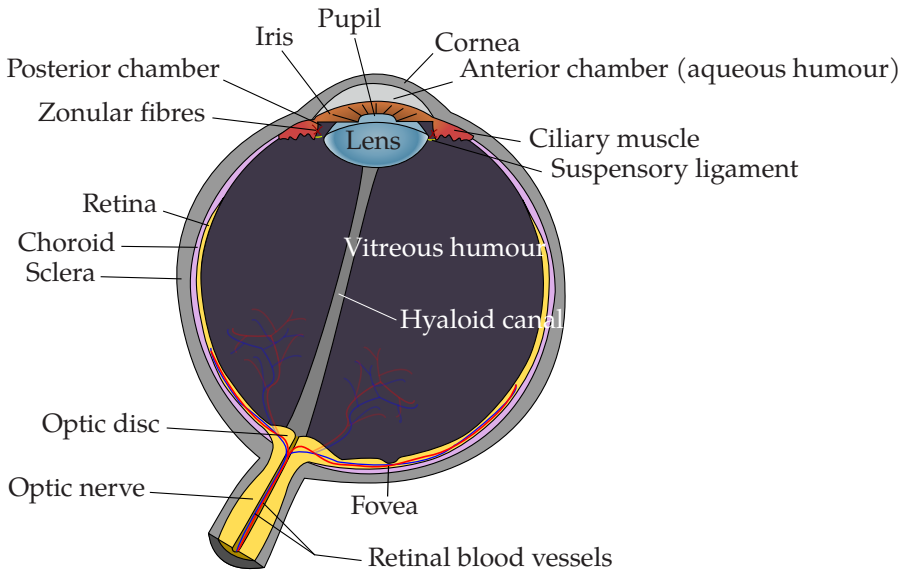


Figure 6.8: Cross section of the human eye (source: Rodrigo Castilhos)

Light enters the eye through the transparent *cornea*, through the *aqueous humour* and the opening of the *pupil* into the *lens*. The lens focuses the light of the image we see through the *vitreous humour* onto our *retina*.

The amount of light that is passing through the lens can be regulated using the *iris*. This is a kind of diaphragm that can be made wider and smaller using a set of iris muscles. The lens can be adjusted for different focal lengths using the ciliary muscles in the eye.

The retina is filled with different kinds of light sensitive nerve endings connected with the optic nerve. The central spot of the retina is the so-called fovea. The optic nerve enters the eye at the back side at a position that we call the *optic disc*. On this disc (part of the retina), no light sensors are located.

6.2.2 Its light sensors

The retina is filled with light sensitive nerve endings. We discern two types of endings: rods and cones.

Rods The rods behave like a wide-wavelength-range integrator. They are sensitive to intensity but cannot distinguish colors. They are positioned in a ring-shape around the fovea. They contribute to our scotopic *and* photopic vision. The optic disc is located right in the ring of rods and causes a blind spot in our vision as it is not equipped with rods.[7]

Cones As opposed to the wide-wavelength-range nature of the rods, the cones are only sensitive to a smaller range of wavelengths. They come in different variants. Usually, we classify them into three categories: cones sensitive to blue light, sensitive to green light and

[7]There exist many good test sheets and even web applets that can help you to discover your own blind spots. Do a web search for 'eye blind spot' and have fun!

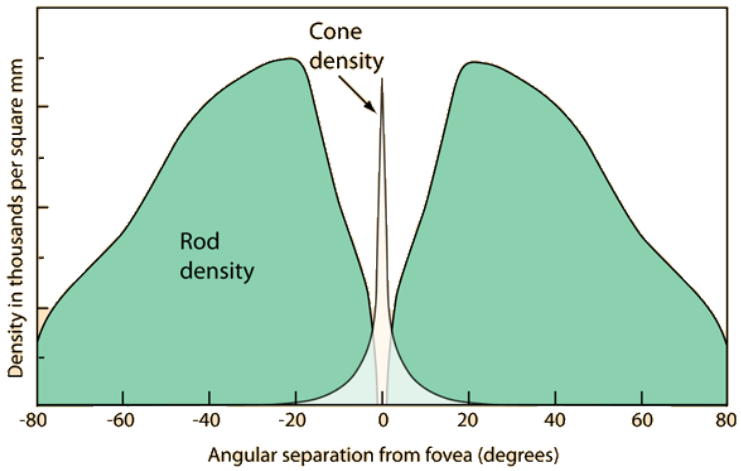


Figure 6.9: Rod and cone density as a function of the angular separation from the fovea (source: http://hyperphysics.phy-astr.gsu.edu)

sensitive to red light. The cones are positioned in a circle on the fovea. They only contribute to our photopic vision.

Photopic vs. scotopic vision What's the deal about photopic and scotopic vision? Well, it turns out that though the rods are not color sensitive, they are very light sensitive, while the cones are color sensitive, but they require a lot of light to function. In addition, there are many more rods than cones (the density of rods and cones as a function of the angular separation from the fovea can be seen in Figure 6.9).

Therefore, for low light levels, the signal coming from the cones does not get above the noise-level, and the only useful signals our brain receives are the signals from the rods. Therefore, in low light conditions, we cannot discern any colors. No doubt that you've experienced it yourself, when walking in a forest at night, the leaves look the same as the bark of the trees. You only see shades of gray. This type of vision is called *scotopic vision*.

Under sufficient lighting levels, the cones come into action, and together with the rods, they bring us our full *photopic vision*.

Speed of the cones and the rods Discerning colors comes at a price: the time needed for a sufficient amount of photons to excite a cone is significant and definitely larger than for a rod. Therefore the integration time of cones is larger than the integration time of rods. Stated differently: the cones are slower than the rods.

6.2.3 Our brain

One might wonder: how is it possible that such a diverse sensor, with cones, rods, with different sensitivities and different integration times and a whopping blind spot in the middle of the rod ring, gives us such a good and sharp vision? The answer is that our brain is a magnificent signal processor that keeps a model of our environment and even fills in details of

the scenery that we did not see. It may also block details that we do sense, but are not aware of. We need to remember at all times: our brain is a tricky little devil, that is capable of doing great things for our vision, but it may also fool us with optical illusions.[8]

6.2.4 Various facts and figures

Some facts and figures:

- Color perception is only adequate near the fovea: our color-perception of a larger area is largely due to scanning and keeping color information in our memory.
- Our eye is capable of focusing our lens in the order of 400 ms.
- Our eye has a gigantic dynamic range:
  - with a fixed pupil size, we can distinguish approx. 20 intensity levels;
  - with varying pupil sizes, the range is many times bigger.
- Our capability to discern colors is even better.
- Photopic vision is active for incident light levels above 10 lx; in that case our pupil's opening is normal and we have a good color perception.
- Scotopic vision kicks in for incident light level below 10 lx; in that case our pupil is wide open and we have bad (or no) color perception.
- Our eye can move in different modes. Two important modes are:
  - Saccadic movements: when scanning a scene, our eyeball moves in a jerky, jumpy fashion, trying to maximize the time our eyeball stands perfectly still (to allow for a sharp image); at that moment your eye reaches angular speeds up until 900 °/s in time spans of about 20 ms.
  - Servo-motoric movements: when following an object, our eyeball moves continuously to make the moving object stand maximally still in our image frame; in this mode, your eye can reach angular speeds of up to 100 °/s.

To continue a bit further on the last issue regarding modes of movement: without help it is impossible to make your eyeballs perform a perfect circular trajectory. Our tendency to move our eyes in a saccadic way ruins every attempt. Try doing so, while a friend of yours watches your eyes. He/she will clearly see the saccadic motions. On the contrary: Move your finger in a big vertical circle in front of you and follow it with your eyes. Your friend will be able to see a continuous eye movement. Therefore, it can be concluded that it is not some muscularly limitation, but just one of the peculiarities of the big eye movement controller: our brain. It is trained in the very first months of our life and in a Darwinian way naturally selected over the past millions of years to create still images of the things we are interested in. When man evolved from hunting behavior to a more sedentary behavior, no doubt, we lost some of our servo-motoric capabilities.

[8] The Wikipedia page on 'optical illusions' is a true source of amazement and amusement. Check it out!

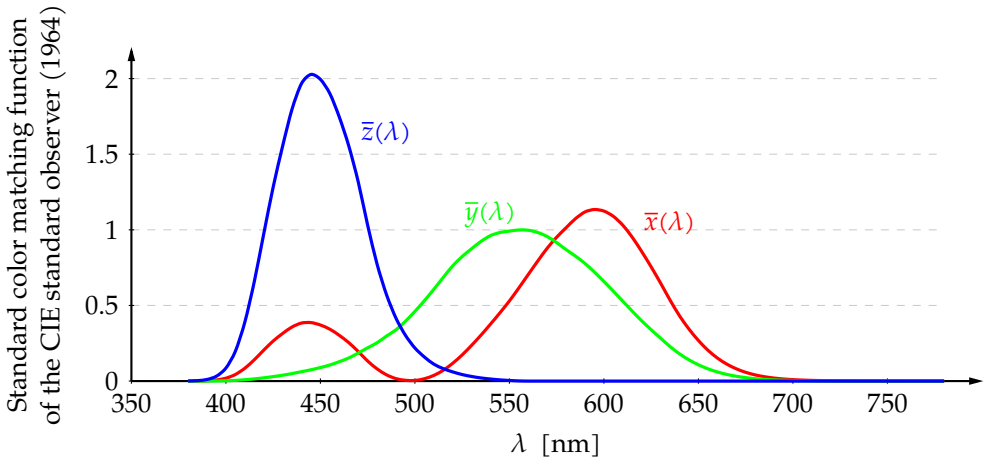


Figure 6.10: The $\overline{x}/\overline{y}/\overline{z}$ color mix required to match monochromatic light of wavelength λ for the 1964 CIE standard observer ($\overline{x}$ red, $\overline{y}$ green and $\overline{z}$ blue)

6.3 Color

In the previous section, we gave a basic overview of the operation of the human eyesight. If you did not read that section, it makes sense to spend a few minutes reading it, before continuing with this section.

6.3.1 The color sensitivity of our eye

Our eyes see colors through three different kinds of cones, located in a circular shape on the fovea in our eye. The excitation of these different cone types gives us different brain sensations and we have learned to describe these sensations as colors. We even agreed upon names for these colors.

Still, it is important to realize that color is not a physical concept. It is our impression of the effect waves with different wavelengths have on the cones in our eyes. This makes color a difficult (and partly an artificial) concept. Our eyes cannot measure wavelength. Color is how our brain translates wavelengths in sensations.

Early research in order to quantify our sensitivity to light of different wavelengths was gathered by the international commission on illumination (CIE, commission internationale de l'éclairage) and synthesized into the *1931 standard observer model* that describes the average mix of standard colors red, green and blue an observer would have to mix in order to match monochromatic light (i.e. light containing only a single wavelength). Later, in 1964, that effort was repeated, leading to the *1964 standard observer model*. The key resulting sensitivity graph can be found in Figure 6.10.

It is important to note that our brain is not aware of this diagram. It just receives per nerve spot in the eye three intensity signals, one for blue, one for green and one for red. Real-world objects emit light that has a complex composition over the entire spectrum of visible light. In our eye, that spectrum is sampled using these functions and then integrated to result in the three mentioned signal values. In our brain, that trio is mapped to a specific color sensation. A

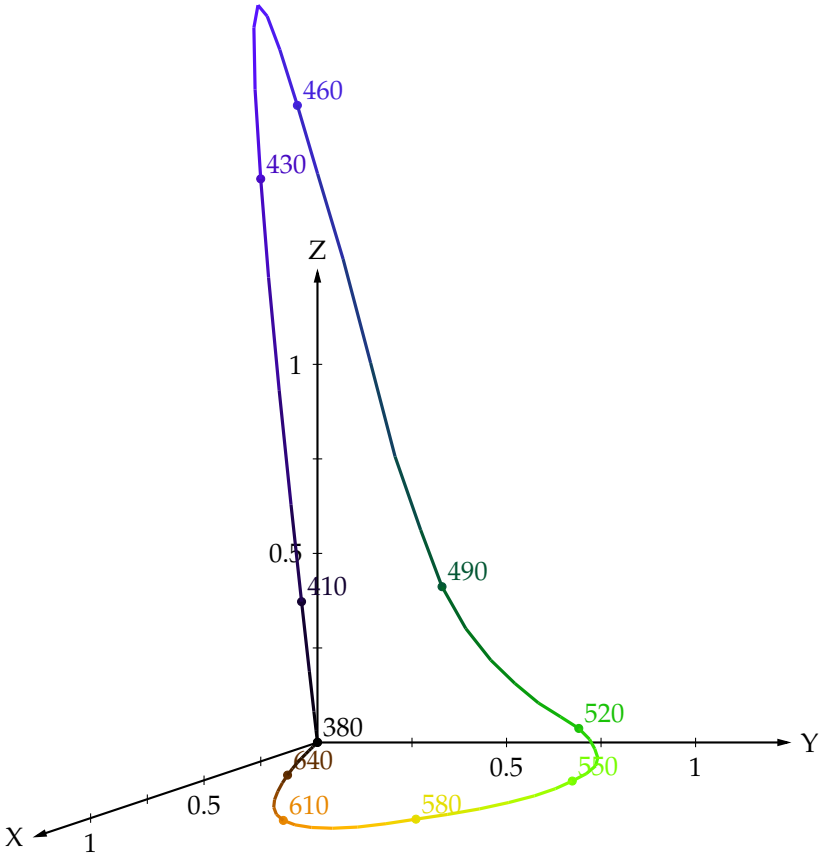


Figure 6.11: The trajectory corresponding to monochromatic light in the tristimulus vector space

uniform spectrum is perceived by our brain as a shade of gray.

6.3.1.1 The tristimulus values

We can model the effect any arbitrary spectrum has in our brain, by considering the three intensity signals of the primary colors as points in the three-dimensional space. This leads to the well known *tristimulus values*:

$$\text{Color} = \begin{cases} X & (\text{red}) \\ Y & (\text{green}) \\ Z & (\text{blue}) \end{cases}$$

In a first attempt, let's check how monochromatic light (light containing only a single wavelength) is mapped in the tristimulus vector space. This is done by scanning through the graph of Figure 6.10 and for every value of λ writing down the tristimulus values. We depicted these values as a trajectory in the tristimulus vector space in Figure 6.11.

Every specific tristimulus vector corresponds to a specific color perception. That said, it must be noted that not all tristimulus vectors correspond to valid perceptions. For example, the tristimulus values of the vectors located on the individual axes for red, green or blue will

never occur in our brain. It's not a matter of not seeing them. It is a matter that they do not correspond to the color of real world objects.[9]

6.3.1.2 The intensity

A specific vector in the tristimulus space corresponds to a specific color sensation. Scaling that vector happens to give us (almost) the same color perception, but with a different brightness. This illustrates that our tristimulus color perception is close to linear. In that sense the brightness is related to the length of the tristimulus vector.

For practical reasons, we pick the 1-norm as a brightness measure, and will refer to it as the *intensity*:

$$\begin{aligned} I &= |X| + |Y| + |Z| \\ &\downarrow \quad X, Y, Z \geq 0 \\ &= X + Y + Z \end{aligned}$$

6.3.1.3 The trichromatic coefficients

Our new brightness measure allows us to normalize the tristimulus values, thus obtaining the *trichromatic coefficients*:

$$\begin{cases} x = \dfrac{X}{I} \\ y = \dfrac{Y}{I} \\ z = \dfrac{Z}{I} \end{cases}$$

Every triplet of these coefficients is considered to correspond to a particular color. In addition, since $x + y + z = 1$, we can describe *any* (normalized) color using only x and y.

6.3.2 Color composition - towards the color gamut

The idea of normalizing colors using the intensity suggests to normalize the entire tristimulus space, by projecting it onto the plane described by $X + Y + Z = 1$ (the so-called trichromatic plane). To illustrate this, this plane has been added to the monochromatic trajectory graph in Figure 6.12. The projection is not an orthogonal, but a homothetic projection (with the origin as homothetic center).

The idea of taking into account that $x + y + z = 1$ can also be introduced into the picture. Let's take a look at the projection plane in the tristimulus space positioning ourselves in the point $(0, 0, +\infty)$. In fact, this corresponds to projecting the trichromatic plane onto the plane formed by the X and the Y axis and relabeling these as x and y.

The result (including the monochromatic trajectory) is called the *color gamut* and it has been depicted in Figure 6.13. The points on the inside of the gamut represent valid color

[9] Never say never. Using psychedelic drugs, probably any of the tristimulus signals can occur in our brain, leading to sensations that are described as 'unreal'. You are advised not to try any of these forbidden substances, but to take my word for it.

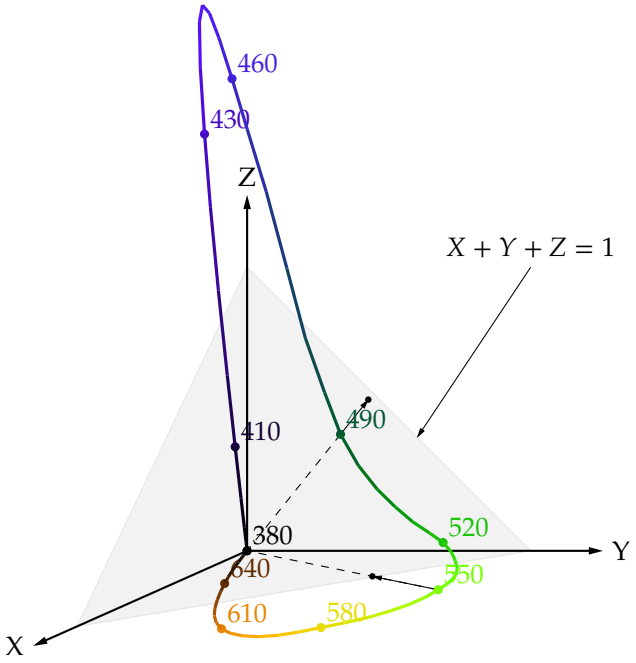


Figure 6.12: The trichromatic normalization plane added to the monochromatic trajectory graph of Figure 6.11 on page 115

perceptions, as any of these points can be created by a weighted combination of pure monochromatic colors. This is based on the assumption that our vision is linear with respect to color perception. Points outside of the color gamut don't correspond to tristimulus signals that will ever occur in a normal brain.

6.3.3 Generating colors

Now that we know how we perceive colors, a key question remains: how are colors being generated?

Two basic mechanisms are in play:

- photon emission,
- photon absorption.

Photon emitting materials Many objects emit photons. Consider for example the sun. Radioactive processes convert matter into energy, energy that is partially radiated as visible light. The light emitted from the sun contains energy at all wavelengths (from ultra-violet to infra-red) (see Figure 6.14). By the time the sunlight reaches the earth's surface, of course, some of the energy has been absorbed amongst others by the water molecules in the air, leading to an energyscape that has some dips in it.

Other examples of radiating color generators are: light bulbs, electrofluorescent lamps, gas discharge lamps, (O)LEDs, fire, ...

The spectrum they emit, determines our color sensation of them.

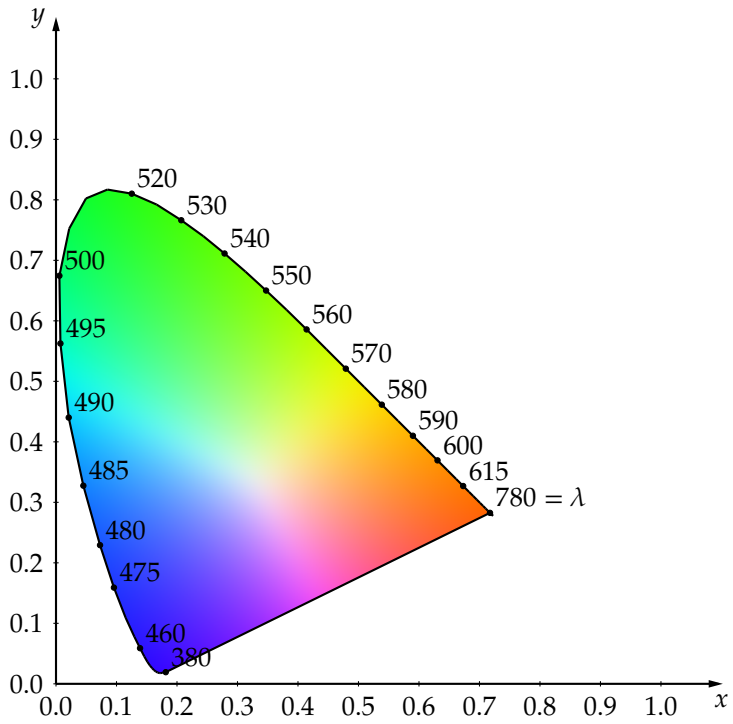


Figure 6.13: The color gamut

Photon absorbing materials Some objects do not generate photons, but they absorb them. In fact, any object (even a mirror, and even a photon emitter) absorbs some of the photons that hit the object. It turns out that this photon catching ability of materials is wavelength dependent. For example, the leaves of trees are very good at catching all photons except for the photons with a wavelength of around 530 nm. These are reflected and we perceive them to be green.

Man has gone to great length in perfecting the photon catching process in paint and ink technology.

6.3.4 Mixing colors

We can describe mixing colors using two different mixing models, depending on the way the colors have been made. We discern:

- additive mixing,
- subtractive mixing.

The former is used for modeling the mixing of colors coming from a photon emitting material (light sources like, for example, the RGB LEDs in a display). The latter is used to model the mixing of colors coming from a photon absorbing material (filters like, for example, paint).

Additive mixing Because our eye basically samples red, green and blue, it makes sense to use these as basic colors. If we join the light coming from these light sources, we can make any color within that color triangle in the color gamut. Picking red, green and blue ensures a large triangle. That has been illustrated in Figure 6.15. Mixing two colors adds the properties of

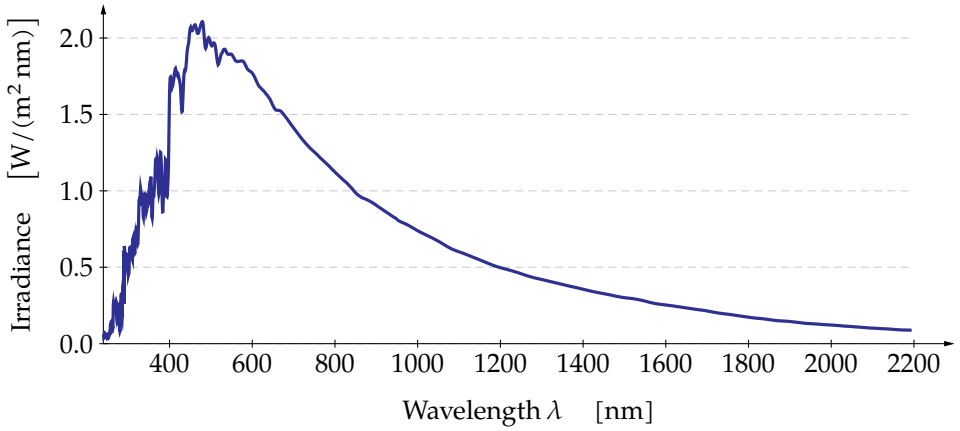


Figure 6.14: Irradiation of the sun as a function of wavelength, on July 1st 2015, measured on an orbital satellite by the *SOlar Radiation and Climate Experiment* (*SORCE*) of NASA; data provided by University of Colorado at Boulder

both colors to the final color. If you mix the three basic colors in equal proportions and then keep increasing its intensity, you will obtain white.

Often a basic circle diagram is considered to illustrate additive mixing. It can be found in Figure 6.16a. This Venn-diagram-like structure shows us that mixing light from a red and a blue source gives magenta. Mixing red and green gives yellow and green and blue yields cyan.

Subtractive mixing Understanding subtractive mixing takes a bit more reasoning. Remember, photon absorbing materials absorb a particular range of wavelengths that get absorbed, while leaving the others untouched. Therefore, mixing two of these materials, creates a material that only re-emits the intersection of what they each emit separately. This is the kind of color mixing that you probably will remember from kindergarten: "mixing yellow and light blue gives green". You will also remember that when you keep mixing different paints, you end up with a sludge that is very close to black.

The three basic colors for additive mixing — red, green and blue — are not the most optimal for subtractive mixing. Their complements: cyan (the color that is obtained by only filtering away red), magenta (obtained by only filtering green) and yellow (obtained by only filtering blue) are much more suited for this. Therefore, these are the inks you will find in any common color ink-jet printer to produce the vivid colors on your photo prints.

Subtractive mixing is often illustrated using a similar circle diagram that can be found in Figure 6.16b.

6.3.5 Color models

Fiddling with colors is fun, but in the end, our goal is to use the wavelength or color properties of the objects in an image to allow for a better way of image processing than we can do with just grayscales.

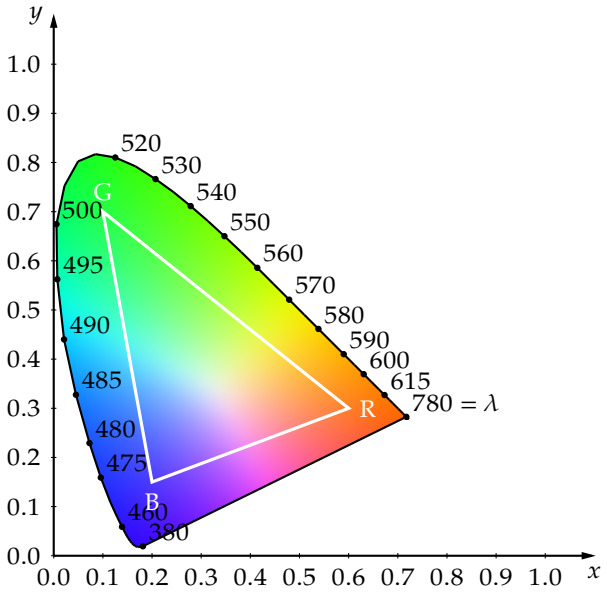


Figure 6.15: The color gamut with the attainable color triangle, given three colors R, G and B

(a) Additive color mixing

(b) Subtractive color mixing

Figure 6.16: Color mixing

Therefore, we devote quite a bit of attention to the color models that we use to perform calculations on.

6.3.6 Simple color models

A first category are models that are a rather straightforward implementation of our first insight into how our vision works. Their advantage is simplicity, their drawback is that processing them in a sensible way is not so obvious.

6.3.6.1 RGB

The RGB color model is most suited for describing colors in additive situations (e.g., light emitting displays).

In the RGB model, we just pick three colors out of the color gamut, a particular tone of red, a particular tone of green and a particular tone of blue. With these basic colors we can create any normalized color in the gamut triangle and even to a limited extent more and less bright versions of them.

The model consists of considering three coefficients R, G and B within the range $[0, 1]$ and using them as weight coefficients to mix our three chosen basic colors:

$$C_{RGB} = \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

with $R, G, B \in [0, 1]$.

In this way we can make any color in the RGB cube. This has been illustrated in Figure 6.17. A lot of colors have been indicated on this cube. Often these colors are abbreviated using a single character. The following coding scheme is generally accepted: red - R, green - G, blue - B, cyan - C, magenta - M, yellow - Y, white - W, black - K.

6.3.6.2 CMY(K)

The CMY color model is most suited for describing colors in a subtractive situation (e.g., mixing inks).

The basic idea is to use the complements of the basic colors of the RGB model, respectively cyan (C), magenta (M) and yellow (Y). Therefore:

$$C_{CMY} = \begin{bmatrix} C \\ M \\ Y \end{bmatrix} = 1 - C_{RGB} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

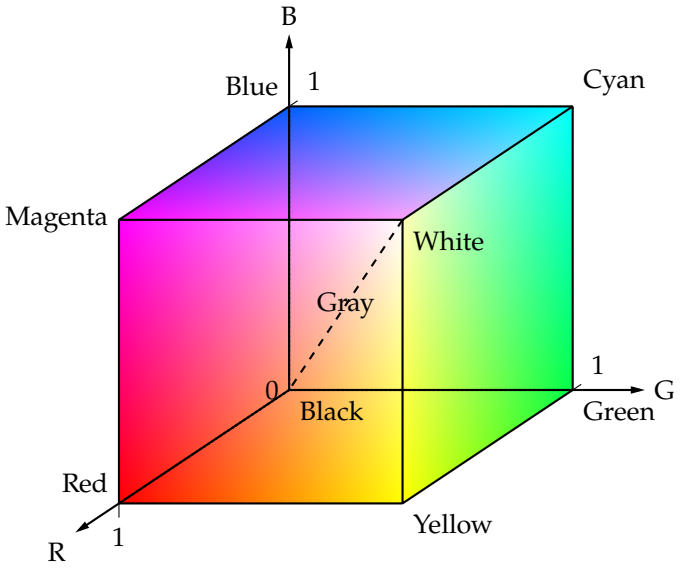


Figure 6.17: The RGB cube of the RGB color model

Mixing cyan, magenta and yellow inks or paints often leads to a poor-quality black, more resembling dark brown than black. No doubt you have experienced this yourself when mixing paints when you were in kindergarten. Mixing too many different colors, gives you a dark brown sludge, but it never becomes true black. To overcome this, one replaces the gray-amount in the color by black ink, leading to the well-known CMYK model.

$$C_{CMYK} = \begin{bmatrix} C - \min(C_{CMY}) \\ M - \min(C_{CMY}) \\ Y - \min(C_{CMY}) \\ \min(C_{CMY}) \end{bmatrix} \tag{6.23}$$

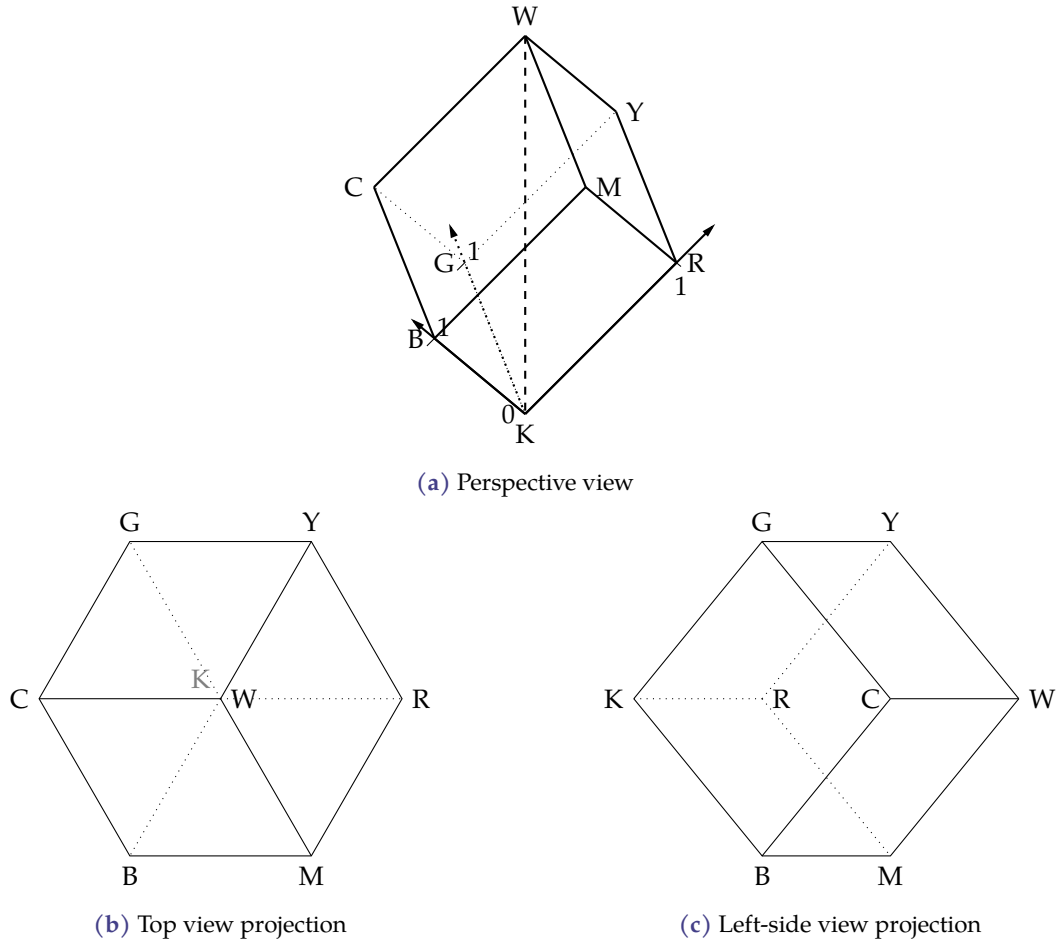
In printing jargon one describes this as 'four color printing'. Now you understand why your photo printer has four color cartridges: cyan, magenta, yellow and black.

The definitions of the CMY and CMYK models also show a very basic operation that we'll have to carry out when doing digital image processing: converting one color model to another. Why would we want a different color model when doing our DIP calculations? Because some operations are easier with a particular color model then with another. In this way, the color model of your sensor (most probably RGB) might not be the one of your preference.

The simple equation of (6.23) often still results in poor quality results. Therefore, more complex *color management systems* are used that take into account the specific inks used and the particular target printing process. This is also a matter of converting color models. We will get to that when discussing device independent color models and ICC color profiles.

6.3.7 Conceptual color models

A disadvantage of the RGB, CMY and CMYK color models is that we don't perceive them as very natural. Humans seem to have a natural tendency to describe colors using basic concepts



(a) Perspective view

(b) Top view projection

(c) Left-side view projection

Figure 6.18: The RGB cube put on its black corner and erected along its black to white diagonal

as the basic color ('red'), its color intensity ('deep, intense') and its lightness ('bright').[10]

The HSI model attempts to fill in the need for a model in which colors are described in this conceptual manner.

6.3.7.1 HSI

The HSI model is derived from the RGB model by putting the RGB cube on its black corner and erecting it along the black to white diagonal (the grayscale line). By convention we draw the red axis directed to the right. This has been illustrated in Figure 6.18.

Now let's consider cross-sectional planes perpendicular to the black to white diagonal (grayscale line). We will use the projection views of Figure 6.18b and Figure 6.18c to show what these planes look like. We use the relative position I of the intersection of the plane with the grayscale line to label the situations under consideration. The parameter I represents the intensity and it ranges from $I = 0$ (in position K) to $I = 1$ (in position W). Knowing this, take a

[10] No doubt these concepts are relevant to our survival during ancient times, to keep us from eating poisonous berries, or to keep us from touching hot, glowing surfaces. Our nowadays DIP needs, are still focused on similar jobs. Therefore, these concepts are still relevant in a computer vision environment.

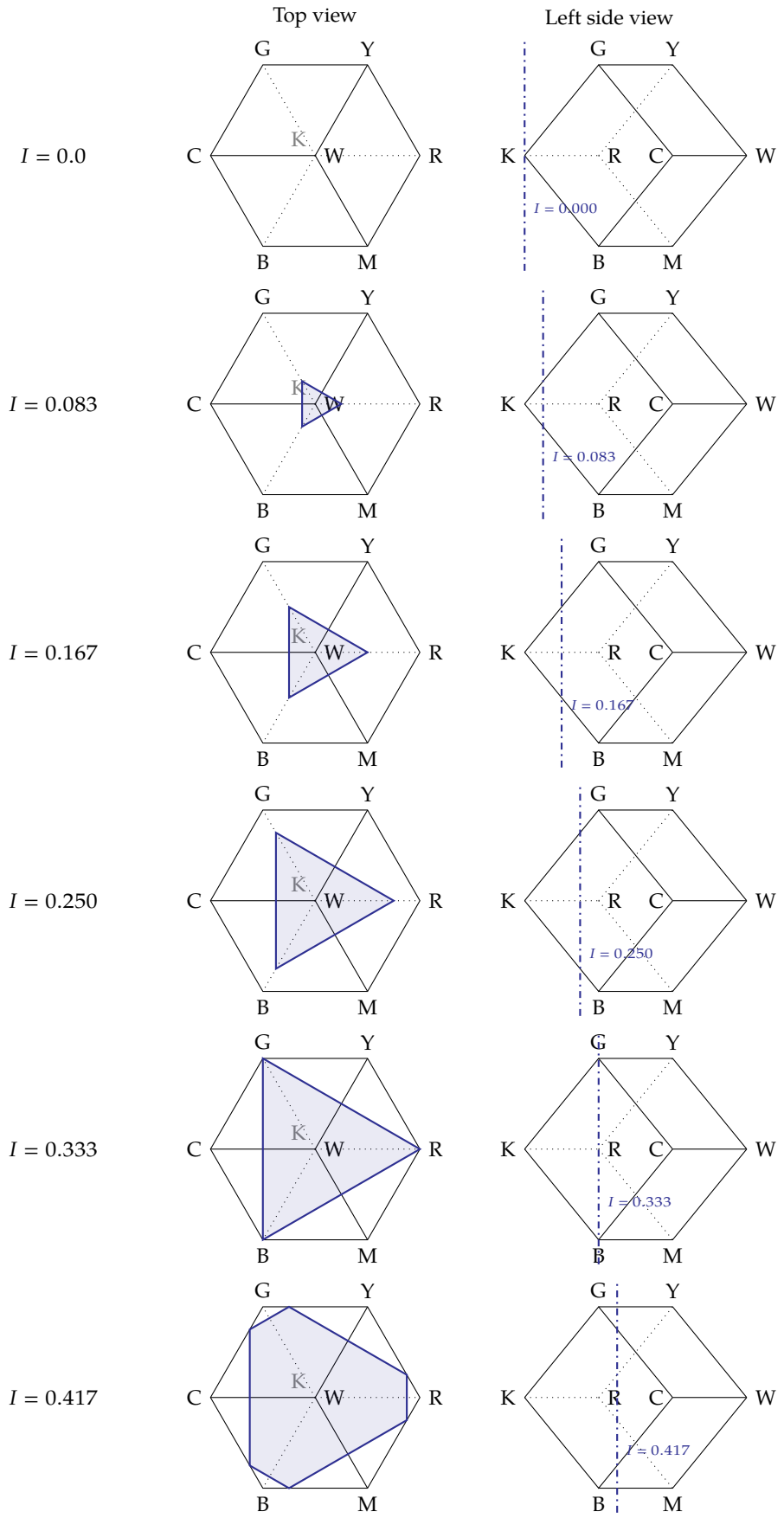


Figure 6.19: Cross sections of the erected RGB cube with planes perpendicular to the grayscale line for different values of I

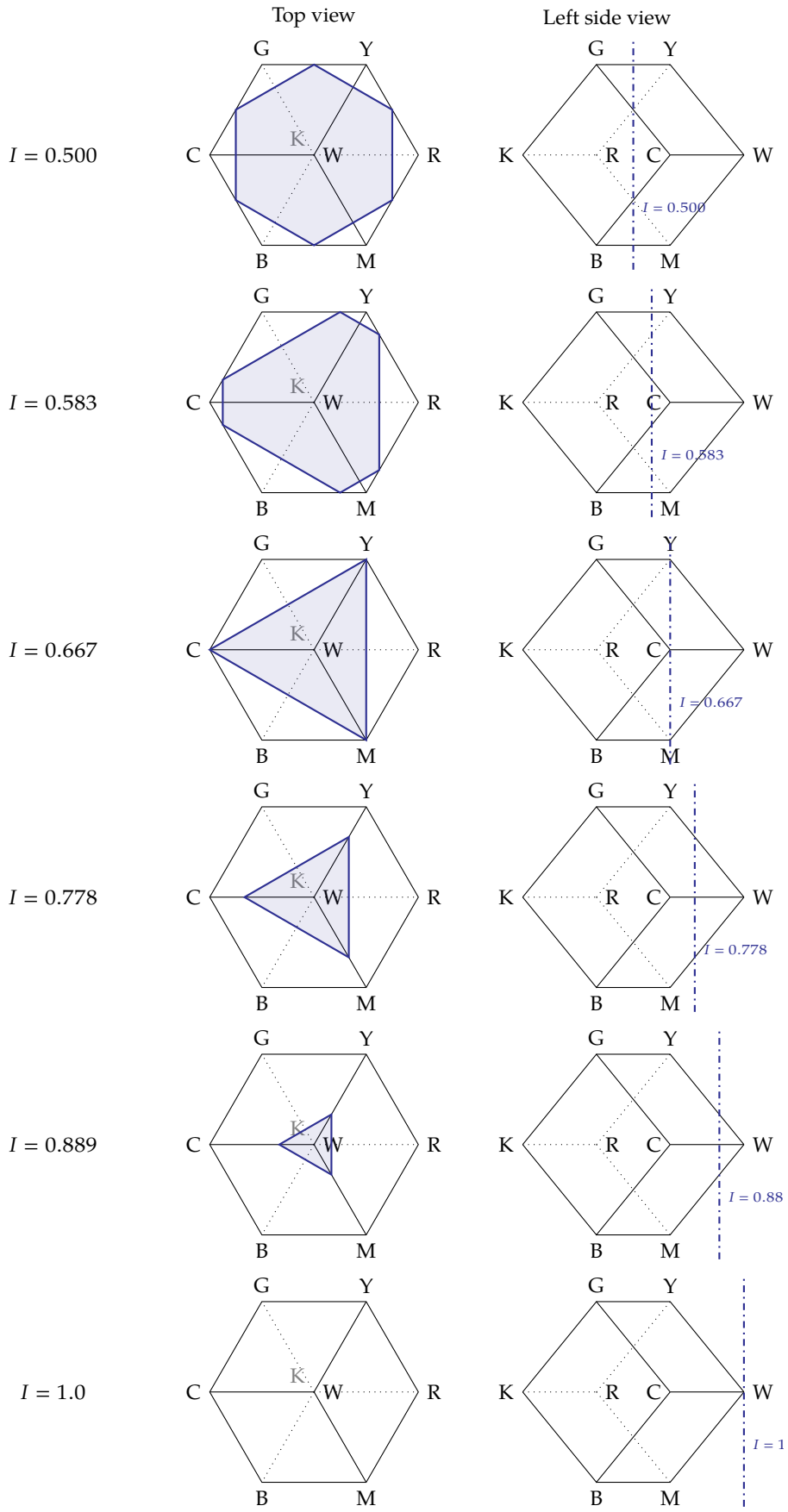
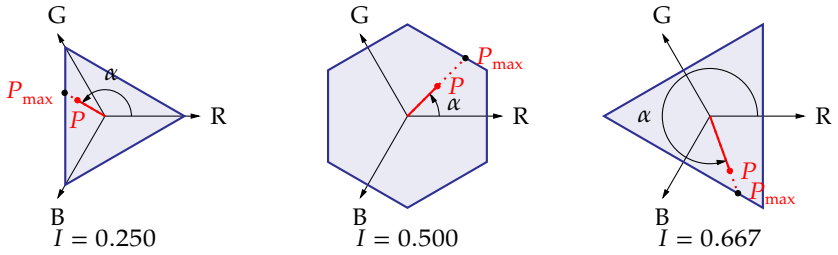


Figure 6.20: Cross sections of the erected RGB cube with planes perpendicular to the grayscale line for different values of I (cont'd)

look at Figure 6.19. and Figure 6.20. Using these cross-sectional planes, we can describe any point P within the RGB cube in two steps, using three new parameters:

1. The position I of the plane containing P. As mentioned before, we call this parameter the *intensity* I. As extreme values the value $I = 0$ represents black, the value $I = 1$ represents white.
2. The position of the point P within the plane. This has been illustrated below for three values of I:



- The angle α between the vector P and the R-axis determines the color, and is called the *hue* (H):

$$H = \alpha$$

(in radians!)

- The relative length of the vector P w.r.t. $P_{\max}$, is the *saturation* (S):

$$S = \|P\| / \|P_{\max}\|$$

That completes our HSI color model.

Conversion from RGB to HSI and vice versa Using the following conversion formulae, you can convert RGB to HSI colors:

$$I = \frac{R + G + B}{3}$$

$$S = 1 - \frac{\min(R, G, B)}{I}$$

$$H = \begin{cases} \alpha' & \text{if } B \leq G \\ 2\pi - \alpha' & \text{if } B > G \end{cases}$$

with

$$\alpha' = \arccos\left(\frac{R - 0.5G - 0.5B}{\sqrt{(R-G)^2 + (R-B)(G-B)}}\right)$$

You can also convert HSI to RGB, using the following formulae. We need to discern three ranges of H:

- $0 \leq H < 2\pi/3$:

$$B = I(1 - S)$$

$$R = I\left(1 + \frac{S \cos H}{\cos(\pi/3 - H)}\right)$$

$$G = 3I - R - B$$

- $2\pi/3 \leq H < 4\pi/3$:

$$R = I(1-S)$$
$$G = I\left(1 + \frac{S\cos(H - 2\pi/3)}{\cos(-\pi/3 - H)}\right)$$
$$B = 3I - R - G$$

- $4\pi/3 \leq H < 2\pi$:

$$G = I(1-S)$$
$$B = I\left(1 + \frac{S\cos(H - 4\pi/3)}{\cos(-\pi - H)}\right)$$
$$R = 3I - G - B$$

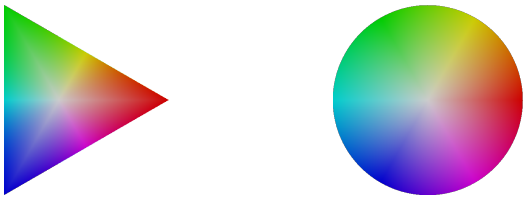
Learning these equations by heart does not make sense. They are available as conversion routines in most image processing packages (like MATLAB for example) and for pen-and-paper calculus, you can consult a formula collection that lists them.

Remarks

- H is often normalized to the range $[0, 1]$:

$$\overline{H} = \frac{H}{2\pi}$$

- As the shape of the color planes varies with the intensity I, often the color planes are mapped (independently of the intensity) to triangles or circles:



- A lot of variants exist of the HSI model with two notable examples that are commonly used: HSV and HSL. Their hue definition is different, as are their definitions for saturation and the third parameter (V or L). They both use a common intermediate parameter C denoted as *chroma*, with

$$C = M - m$$

with $M = \max(R, G, B)$ and $m = \min(R, G, B)$.
The hue definition in HSV and HSL tries to avoid the arccos operation, by defining hue (in radians) as:

$$H = \begin{cases} \frac{\pi}{3}\frac{G-B}{C} \mod 6 & \text{if } M = R \\ \frac{\pi}{3}\frac{B-R}{C} + 2 & \text{if } M = G \\ \frac{\pi}{3}\frac{R-G}{C} + 4 & \text{if } M = B \end{cases} \tag{6.24}$$

This corresponds to linear interpolation in between the grid points $3k\pi$, with $k = 0, 1, 2, 3, 4, 5$.
Specifically for HSV:

$$V = \max(R, G, B)$$
$$S = \frac{C}{V}$$

For HSL:

$$L = \frac{\max(R, G, B) + \min(R, G, B)}{2}$$
$$S = \frac{C}{1 - |2L - 1|}$$

Insight in the origin of these equations requires quite a bit of geometric understanding and goes beyond the scope of this course.

6.3.7.2 NTSC

A disadvantage of the HSI model is that one needs trigonometric functions to convert to and from RGB. The HSL and HSV already mitigate this by defining hue in a different way, but they still require 'case'-equations.[11] In the early days of television, hardware was still mainly analog and therefore calculating the sine, cosine or using case definitions of a signal was not easy.

An attempt to make a more conceptual color model that relates more easily to RGB is the mode created by the National Television System Committee (NTSC). It is a linear model:

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.274 & -0.322 \\ 0.211 & -0.523 & 0.312 \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Y describes the intensity (black to white) and is called the *luma component*; I and Q describe the color information and are therefore called the *chroma components*.

One can go back using:

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1.000 & 0.956 & 0.621 \\ 1.000 & -0.272 & -0.647 \\ 1.000 & -1.106 & 1.703 \end{bmatrix} \cdot \begin{bmatrix} Y \\ I \\ Q \end{bmatrix}$$

6.3.7.3 YCbCr

The YCbCr color model is a model for digital video/television. In its SD version it converts the normal RGB values to 8-bit Y, Cb, Cr values using an affine transformation, followed by a quantization step (in this case, rounding to integer values in between 0 and 255).

$$\begin{bmatrix} Y' \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 65.48 & 128.55 & 24.97 \\ -37.98 & -74.20 & 112.00 \\ 112.00 & -93.79 & -18.21 \end{bmatrix} \cdot \begin{bmatrix} R' \\ G' \\ B' \end{bmatrix} + \begin{bmatrix} 16 \\ 128 \\ 128 \end{bmatrix}$$

Y is the luma component and describes the intensity (black to white). Cb (the blue difference) and Cr (the red difference) are the chroma components and they describe the color information. Did you note the tick marks next to the R, G and B? This is to indicate that these are the gamma corrected RGB values. Why are these gamma corrected? In the old days it was to correct for nonlinearities in imaging vacuum tubes (vidicons and cathode ray tubes). Now we don't use these anymore, but we still use gamma correction for a good reason: our human vision is more sensitive to darker differences in intensity than to lighter differences in intensity. Therefore, before storing images, the intensities are scaled using a gamma transformation with

[11] In more formal mathematics, case definitions are denoted as: function definitions with multiple members; you recognize them by the accolade in front. See, e.g., equation (6.24).

often $\gamma = 0.45$ (see section 5.4.2.1 on page 69, for more information on the gamma transform). Before displaying the images, the original intensity values are restored using a gamma transformation with $\gamma = 2.2$. This allows coding using less bits for a similar result, or alternatively obtaining better results given a number of bits. To indicate that the luma component is derived from gamma corrected RGB values, it is tick marked also.

This model has been derived from an analog mapping, the so-called *YPbPr* color model, that helps explaining the terms blue difference and red difference:

$$\begin{aligned} Y' &= K_R \cdot R' + (1 - K_R - K_B) \cdot G' + K_B \cdot B' \\ P_B &= \frac{1}{2} \cdot \frac{B' - Y'}{1 - K_B} \\ P_R &= \frac{1}{2} \cdot \frac{R' - Y'}{1 - K_R} \end{aligned}$$

with K_b and K_r well chosen constants.

Now you can clearly see that the P_B signal is built from the difference between the blue component and the luma, and the P_R signal is likewise built on the difference between the red component and the luma signal.

Often the name *YCbCr* is mixed up with the name *YUV* (the name of the color model used in the PAL television standard). Nowadays in most of the cases, when people mention YUV, they actually mean YCbCr. The old standards like NTSC and PAL are almost history.

6.3.8 Indexed colors

Indexed colors are the result of a computer programmer's approach to color. The starting point is that there are a lot of colors, so enumerating (numbering) them (a typical computer programmer's approach) will result in huge color codes. However, we may assume that we will not need all colors. Therefore, let's only number the colors we really need and store these in a color dictionary, the so-called color index or color map.

Now, we can build images whose pixels are encoded using pseudo grayscale intensities, every intensity level mapping to a different color as listed in the index.

Examples of such image standards are:

- the Graphics Interchange Format (GIF),
- the indexed-mode of the Portable Network Graphics (PNG) format.

Remarks

- Without the color map, the image cannot be truly reproduced with respect to color. So, care should be taken to avoid corruption of the color map.
- Sometimes the color map is fixed for all images: e.g., the 64 'excel standard colors'.

Indexed colors are not very useful for Digital Image Processing.

6.3.9 Device independent color models

So far, our color creation model has been based on the principle of picking three basic colors for red, green and blue. and mixing them. This allows us to create any color within the RGB color triangle that's part of the entire gamut. Picking colors red, green and blue that cause a triangle that covers as much as possible of the gamut is key for making a good display system. Computer display manufacturers go to great lengths in trying to create displays with maximal color coverage. Yet, that illustrates the fundamental problem: when the industry would have agreed on using standard red, green and blue color filters in their LCD screens, or standardized red, green and blue LEDs, then a particular image would look exactly the same on every computer display or television set. However, that's not the case. Every monitor or television set (type) creates its own version of a particular image. This might be no big deal when programming computers or doing office tasks. However, when dealing with images for desktop publishing or photo processing, this poses a huge problem.

This calls for a new type of color model. One that does not base itself on a particular choice of red, green, or blue, but that is *device independent*.

Without even trying to be exhaustive, we will take a brief look at:

- the 1931 CIE XYZ color space
- the 1931 CIE xyY color space
- the sRGB color space
- the CIE L*a*b* space

Before starting off: the most brute force approach would be to store the entire spectrum of visible light for every pixel. However, there are two problems attached to that approach:

1. This would require an awful amount of data to store for every pixel; you'd better start inventing new, larger memories if you want to take that route.
2. We don't have any sensors that are capable of analyzing the full spectrum for every pixel. A new technology challenge?

No, it doesn't make sense to follow this approach because, finally, the light beam will have to enter our eye and will be converted to three internal signals in our brain. Therefore, you will see that all device independent models in some way are based on the tristimulus model.

A remaining flaw might be the inaccuracy of the standard observer model, but that turns out to be quite accurate.

6.3.9.1 1931 CIE XYZ - the tristimulus model

In this color model, we just agree to use the tristimulus model of the 1931 standard observer. The color of every pixel is encoded using the three values of the parameters X, Y and Z. The range of these parameters (in case you want to quantize them) depends on the brightness of the image under consideration. In many cases, floating point values mitigate this problem sufficiently.

6.3.9.2 1931 CIE xyY - the chromatic model

In an attempt to get a bit closer to the luma/chroma idea, this color model uses the trichromatic coefficients:

$$\begin{cases} x = \frac{X}{I} \\ y = \frac{Y}{I} \\ z = \frac{Z}{I} \end{cases}$$

with

$$I = X + Y + Z$$

This allows us to use I as luma, and x and y as chroma parameters. Indeed, we can derive $z = 1 - x - y$.

However, for simplicity's sake the chromatic model uses Y as intensity and it codes every point in the tristimulus space using x, y and Y, therefore its name: the xyY model.

It's quite easy to convert from tristimulus values to xyY:

$$\begin{cases} x = \frac{X}{X + Y + Z} \\ y = \frac{Y}{X + Y + Z} \\ Y = Y \end{cases}$$

and vice versa:

$$\begin{cases} X = \frac{x}{y} Y \\ Y = Y \\ Z = \frac{Y}{y}(1 - x - y) \end{cases}$$

6.3.9.3 sRGB - keep the ship afloat

Big IT companies, like HP and Microsoft, were not so keen on adopting their software and operating system to a totally new device independent color model. The amount of data and programs that were available for RGB was huge. Throwing that all away would indeed have been a waste or would have taken a considerable effort.

Therefore, they proposed to adapt the RGB model such that it becomes device independent. The created the *sRGB* model.

It is based on

- standardizing the colors one takes for red, green and blue in the RGB model:

	1936 CIE xyY		
Basic color	x	y	Y
R	0.6400	0.3300	0.2126
G	0.3000	0.6000	0.7153
B	0.1500	0.0600	0.0721
W	0.3127	0.3290	1.0000

- a nonlinear color transformation from XYZ to RGB, including gamma transformation for optimal use of the quantization range.

Transformation from the tristimulus values to the sRGB values is straightforward:
We start by computing intermediary values for R, G and B, using

$$\begin{bmatrix} R_{int} \\ G_{int} \\ B_{int} \end{bmatrix} = \begin{bmatrix} 3.2406 & -1.5372 & -0.4986 \\ -0.9689 & 1.8758 & 0.0415 \\ 0.0557 & -0.2040 & 1.0570 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

If needed X, Y and Z need to be scaled to guarantee that R_{int}, G_{int} and B_{int} are all in the range 0 to 1.
Subsequently, we perform a gamma transformation

$$\begin{cases} R = f(R_{int}) \\ G = f(G_{int}) \\ B = f(B_{int}) \end{cases}$$

with

$$f(t) = \begin{cases} 12.92t & \text{if } t \leq 0.0031308 \\ (1+a)t^{1/2.4} - a & \text{otherwise} \end{cases}$$

and $a = 0.055$.

Inverse transformation goes along the same lines and can easily be derived as:

$$\begin{cases} R_{int} = g(R) \\ G_{int} = g(G) \\ B_{int} = g(B) \end{cases}$$

with

$$g(t) = \begin{cases} \frac{t}{12.92} & \text{if } t \leq 0.04045 \\ \left(\frac{t+a}{1+a}\right)^{2.4} & \text{otherwise} \end{cases}$$

Subsequently, we perform a linear conversion to the tristimulus values:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0.4124 & 0.3576 & 0.1805 \\ 0.2126 & 0.7152 & 0.0722 \\ 0.0193 & 0.1192 & 0.9505 \end{bmatrix} \begin{bmatrix} R_{int} \\ G_{int} \\ B_{int} \end{bmatrix}$$

The sRGB standard has been adopted by many companies and organizations and is widely used.

6.3.9.4 CIE L*a*b* - the industrial model

A final model that we will discuss is the CIE L*a*b* model. It builds on the idea of using Y as luma and using the red difference and blue difference as chroma signals. Therefore, it's quite similar to the YCbCr model. However, this one is device independent and has the advantage of exhibiting a very true color distance (thanks to a nonlinear transformation characteristic): equal distances in the L*a*b* space are perceived as true equal differences in color perception.

In addition the model offers a settable white point (X_w, Y_w, Z_w), allowing for a proper white balance correction.

Transforming from the tristimulus values to L*a*b* is straightforward:

$$\begin{cases} L_* = 116 f(Y/Y_w) - 16 \\ a_* = 500\,(f(X/X_w) - f(Y/Y_w)) \\ b_* = 200\,(f(Y/Y_w) - f(Z/Z_w)) \end{cases}$$

with

$$f(t) = \begin{cases} t^{1/3} & \text{if } t > (6/29)^3 \\ 1/3(29/6)^2 t + 4/29 & \text{otherwise} \end{cases}$$

Inverse transformation is also very straightforward:

$$\begin{cases} X = X_w f^{-1}\left(\dfrac{L_* + 16}{116} + \dfrac{a_*}{500}\right) \\ Y = Y_w f^{-1}\left(\dfrac{L_* + 16}{116}\right) \\ Z = Z_w f^{-1}\left(\dfrac{L_* + 16}{116} - \dfrac{b_*}{200}\right) \end{cases}$$

with

$$f^{-1}(t) = \begin{cases} t^3 & \text{if } t > 6/29 \\ 3\,(6/29)^2\,(t - 4/29) & \text{otherwise} \end{cases}$$

Of course, the price one pays for its good properties is computation time. Evaluating $f(t)$ takes time and power.

6.3.9.5 Color profiles

In theory, the mapping of image colors from one device to another requires a forward and inverse mapping function for every pair of devices.

This can be avoided by using a device-independent color model as an intermediary format (the so-called *profile connection space*). This has been standardized in *ICC color profiles*. Every device has its profile, containing a

- forward transformation function mapping the device-independent model to the internal device-dependent model, and an
- inverse transformation function performing the opposite mapping.

The models that have been chosen as PCS models for the ICC color profiles are:

- 1931 CIE XYZ
- CIE L*a*b*

The mapping function can be a function or algorithm, or a numeric table that can be used for interpolation.

In that way true colors can be preserved throughout the entire image acquisition, processing and rendering chain.

Exercises

Exercise 6.3.9.5-1: Given the following color:

$$C_{RGB} = \begin{bmatrix} 0.3 \\ 0.7 \\ 0.2 \end{bmatrix}.$$

Calculate the corresponding C_{CMY}, $C_{CMYK}$and C_{HSI}.

Exercise 6.3.9.5-2: Try converting the following color vectors to RGB:

- $C_{CMY} = \begin{bmatrix} 0.7 \\ 0.3 \\ 0.8 \end{bmatrix}$
- $C_{CMYK} = \begin{bmatrix} 0.4 \\ 0.0 \\ 0.5 \\ 0.3 \end{bmatrix}$
- $C_{HSI} = \begin{bmatrix} 1.9043 \\ 0.5 \\ 0.4 \end{bmatrix}$

Exercise 6.3.9.5-3: (*) Select some arbitrary RGB colors, convert them to some of the common industrial models, like e.g., HSV, NTSC, YCbCr, L*a*b* and check your results using the appropriate functions in MATLAB: `rgb2hsv`, `rgb2ntsc`, `rgb2ycbcr` and `rgb2lab`. Also try to understand the resulting values by relating them to the color you converted.

6.4 Color transformations

In Chapter 5 we applied intensity transformations to grayscale images. We can also apply transformations to the color space. We'll discuss two types of transformations:

- pseudo color transformations, and
- full color transformations.

6.4.0.1 Pseudo color transformations

Rationale Our eye is better equipped to discern color nuances than grayscale nuances. Therefore, to use our human sensor optimally when inspecting grayscale images, it makes sense to map the grayscales to color.

Definition This is the essence of a pseudo color transformation: we map an image's grayscale intensities to the color components of a color image, e.g., gray $\rightarrow (R, G, B)$. All imaginable intensity transformations can be used (often intensity slices).

Remarks

- This is a transformation that can be considered to be an *image enhancement*. You'd better be cautious for artifacts that may arise.
- Don't use rainbow sequences for sequences without order.

Example Consider the infrared image coming from the Meteosat satellite of the National Oceanic and Atmospheric Administration (NOAA) of the Department of Commerce of the United States (see Figure 6.21). This infrared satellite imager produces grayscale images. Let's bring this picture a bit more alive, by using the pseudo color transformation of Figure 6.22. The result can be observed in Figure 6.23. If you look carefully, you will see the artifacts in the resulting image at the edge of the earth's disc.

6.4.0.2 Full color transformations

Rationale Color correction is often needed because of non-ideal image sensors or displays. However, it is also often used to visually improve images. This may be for a decent purpose, e.g. to allow a better interpretation of the images, but it may also be for less noble reasons, e.g. to create an image that is more pleasing to the eye.

Principle This is the essence of a full color transformation: we map an image's color component values to new color component values. Usually, we stay in the same color space (i.e. we keep using the same color model), however, switching color spaces may make the transformation more easy.
All transformation tricks we have on our sleeve for grayscale images may be reused on the individual color components.

Remarks

- As with grayscale images, we also may use spatial transformations.
- We can go one step further and consider all color components together.

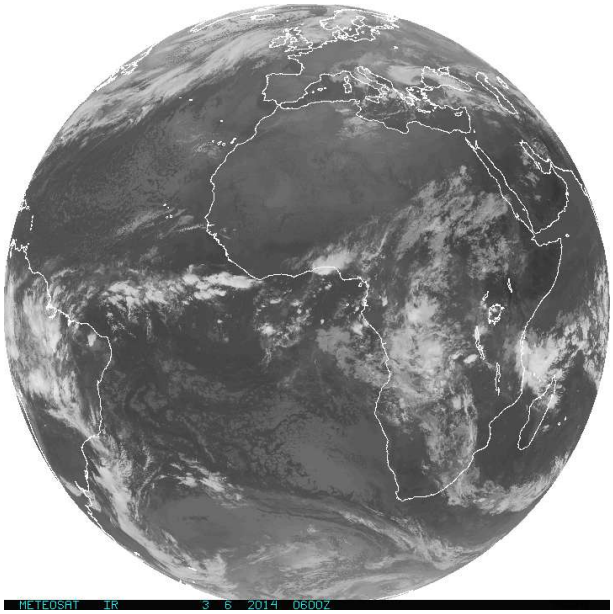


Figure 6.21: Original infrared grayscale image of the earth (image source: NOAA Geostationary
Satellite Server)

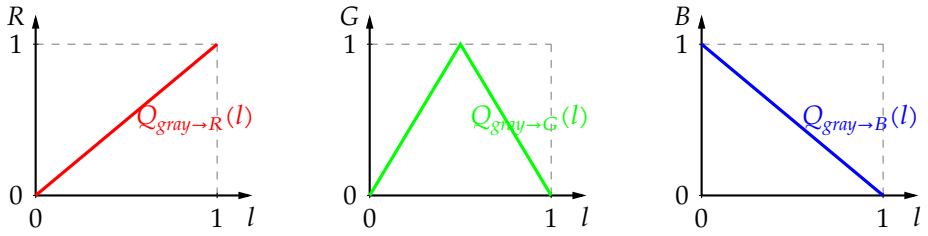


Figure 6.22: Pseudo color transformation to be used on a grayscale image

Example Consider the image of the *Ciudad de las Artes y las Ciencias* in Valencia, Spain (see Figure 6.24). The image was taken on a summer's day with an almost clear sky. Though the image looks good, probably it won't qualify for appearance in a fancy tourist magazine of the city of Valencia.

The photo camera produced a JPEG image in sRGB color space. We took the raw image, transformed it to the HSV space using MATLAB, and applied histogram equalization on the saturation component (the S value of the HSV model). Then we transformed the resulting image back to the sRGB space and produced a JPEG file of it. The result can be seen in Figure 6.25. It clearly looks much more appealing to the eye. Some of you might even perceive it as overdone. Whether it qualifies for appearance in a magazine remains to be seen. However, be warned that a lot of the imagery that appears in holiday magazines has been treated by people who know this cheap photoshop trick well.

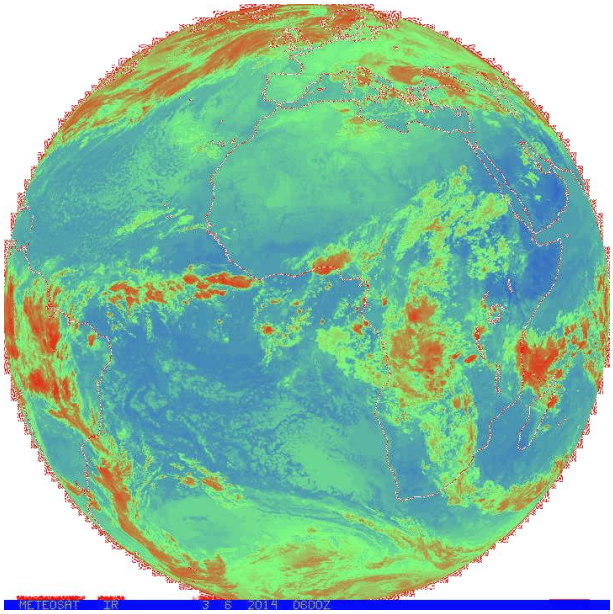


Figure 6.23: The image of Figure 6.21 after a pseudo color transformation using the transformation profiles of Figure 6.22

Figure 6.24: Test image of the Ciudad de las Artes y Ciencias in Valencia, Spain (source: Walter Daems)

Figure 6.25: Resulting image from applying histogram equalization on the S-channel of the test image of Figure 6.24

Chapter 7

Mathematical Morphology

In this chapter, you will learn:

- again how to add and subtract, but instead of subtracting numbers, you will be operating on images instead,
- how to use these basic operations erosion and dilation to define more complex operations,
- what the mathematical basis is for these operations,
- how to use these more advanced operations to analyze what objects or shapes are present in an image.

We'll discuss both binary and grayscale images.

After having read/studied this chapter, you are expected to be able to

- define and explain any of the operations,
- use them on simple examples,
- use them to build simple image analysis algorithms.

7.1 Introduction

Morphology is the science of shapes and forms. It exists in many science domains. In biology morphology is the science of the composition of living organisms. In linguistics it is the science of the composition of words and sentences. In the field of digital image processing, it is the science of describing how images are composed using basic elements or objects. It is the science of deciding what pixels belong to the background, what pixels belong to the foreground and what interesting objects can be enumerated on the foreground.

In that sense, this is the first chapter in which we will perform *image analysis* rather than *image processing*. Remember, the former means determining characteristics of an image while the latter means processing one image into another (see section 3.3 on page 18).

As with any positive science, it makes sense to perform this morphology in a rigorous manner, and the fundamentals of our treatment will be the concept of *sets*. Therefore, what we are about to perform is *mathematical morphology*.

Though this term is almost exclusively used for image processing, it does not mean that the other morphological science domains are not based on a rigorous treatment. Consider e.g. determination charts in biology or syntax/grammar descriptions in linguistics. These are also very rigorous treatments of the science of shapes and forms. So there's no reason to be posh about the fact that we will be doing this in a mathematical manner.

The second reason why the term *mathematical* is appropriate, is that our treatment will define a number of operations on images. Just like in primary school when you were taught to add and subtract numbers, we will learn how to 'add' (the so-called *dilation* of) and 'subtract' (the so-called *erosion* of) images. You will even recognize the basic plus and minus symbols again. In fact, we will be making a new algebra on images. In that sense, you will definitely experience that even *image analysis* is still based on a lot of *image processing*.

Our treatment will be split in two. We will start by discussing the morphology for binary images (pure black and white) and then we will discuss grayscale images. However, before we can take off, we will need to introduce a number of basic concepts, related to points (pixels) and regions (sets of pixels) in the next section.

7.2 Basic concepts

There are a number of new concepts that we need to define. A few related to points or pixels:

- neighborhood,
- adjacency,
- paths,
- components;

and a few other related to regions or sets of pixels:

- reflection and translation,
- foreground and background, and
- set calculus.

You will see that for some of the set concepts, we have to make a distinction between binary and grayscale images. We will restrict ourselves to images that have been sampled on a rectangular grid, though extending the concepts to more exotic samplings should be straightforward.

7.2.1 Concepts related to points/pixels

7.2.1.1 Neighboring pixels / neighborhood

Consider a particular pixel of an image, located at position $[x, y]$. The concept of neighbors of this pixel is an intuitive one. However, we explicitly define a number of neighborhood concepts in the following rigorous manner.

Neighborhood of a pixel

$\boldsymbol{N_4(p)}$ — The N_4 neighborhood of a pixel p at $[x, y]$ consists of the horizontally and vertically neighboring pixels.
$\boldsymbol{N_D(p)}$ — The N_D neighborhood of a pixel p at $[x, y]$ consists of the diagonally neighboring pixels.
$\boldsymbol{N_8(p)}$ — The N_8 neighborhood of a pixel p at $[x, y]$ consists of all 8 neighboring pixels, in short: $N_8(p) = N_4(p) \cup N_D(p)$.

These definitions have been illustrated in Figure 7.1. Note that the definition of neighboring pixels is not dependent on the value/intensity of the pixels.

7.2.1.2 V-adjacency

V-adjacency

Consider a set of intensity values V. Two pixels p and q are V-adjacent if they are neighboring pixels and have intensity values that belong to V.

As this definition relies on the concept neighborhood, we also distinguish multiple variants of V-adjacency:

- 4-V-adjacency — q is 4-V-adjacent to p if they both have a value from V and $q \in N_4(p)$ (or equivalently: $p \in N_4(q)$).
- 8-V-adjacency — q is 8-V-adjacent to p if they both have a value from V and $q \in N_8(p)$.
- m-V-adjacency — q is m-V-adjacent to p if they both have a value from V and
  - $q \in N_4(p)$, or
  - $q \in N_D(p)$ without common 4-V-adjacent pixels.

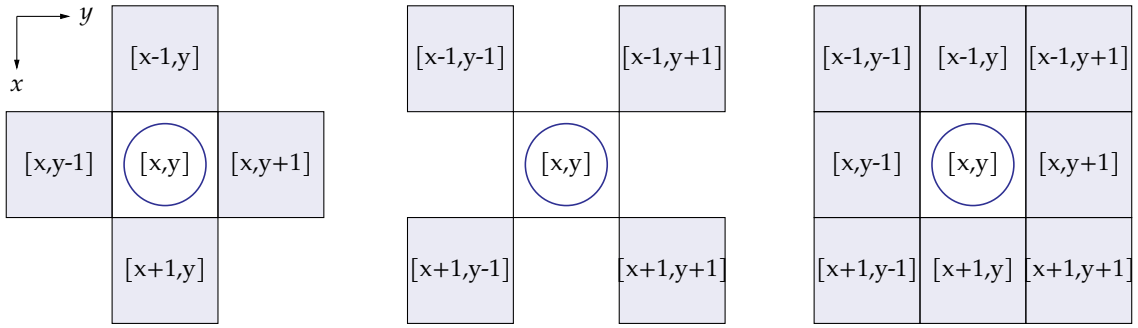
This has been illustrated for binary images in Figure 7.2. Note that the definition of adjacent pixels is dependent on the value/intensity of the pixels! For this reason — in contrast to the common literature — we will always refer to this concept as 'V-adjacency' instead of just the ordinary 'adjacency'.

7.2.2 V-paths

V-path

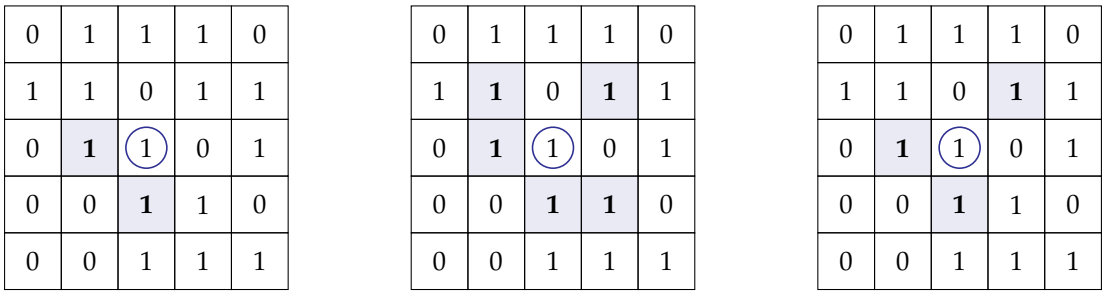
Consider a set of intensity values V. A V-path exists between two pixels p and q if it is possible to progress from p to q through V-adjacent pixels. The pixels are said to be *V-path connected*.

Again, we distinguish three flavors of V-paths based on their corresponding adjacency definitions: 4-V-paths, 8-V-paths and m-V-paths.



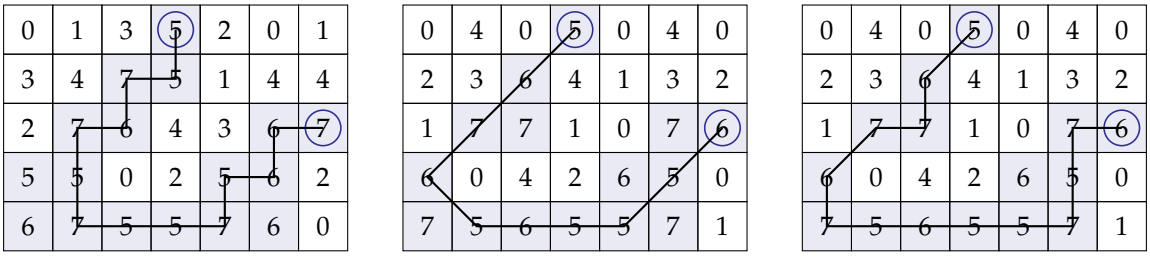
(a) $N_4(p)$ (b) $N_D(p)$ (c) $N_8(p)$

Figure 7.1: Illustration of the neighborhood concept; neighborhood pixels of p indicated in gray; p has been circled.



(a) 4-V-adjacency (b) 8-V-adjacency (c) m-V-adjacency

Figure 7.2: Illustration of the V-adjacency concept with $V = \{1\}$; the central pixel p has been circled, the V-adjacent pixels of p are indicated in gray



(a) 4-V-path and component (b) 8-V-path and component (c) m-V-path and component

Figure 7.3: Illustration of a 4-V-path, an 8-V-path and an m-V-path in between the two indicated pixels with $V = \{5, 6, 7\}$; path lengths are 13, 9 and 13 respectively; these are also the distances as path lengths are minimal; the components in which the paths reside have been indicated with a gray background.

The *length* of a V-path is the number of steps taken to progress from p to q.
The *distance* between a pixel p and q is equal to the length of the shortest V-path between the pixels.
A V-path is *closed* if $p = q$.

These concepts have been illustrated in Figure 7.3.

Remarks

- We consider a pixel to be connected to itself.
- The V-connectedness relationship is an equivalence relationship as it is reflexive, symmetrical and transitive.

7.2.3 V-Components

V-component

The union of all pixels q that can be connected through V-paths with pixel p is said to be an *image V-component*.

The components belonging to the circled pixels have been indicated in Figure 7.3.

Exercises

Exercise 7.2.3-1: Consider the binary image below. Find the $N_4(p)$, $N_D(p)$ and $N_8(p)$ neighborhoods of the pixel p that has been circled.

0	0	0	0	0	0	1
1	0	1	1	0	0	1
0	0	1	(1)	0	1	0
0	0	0	0	1	0	0
0	1	1	0	0	1	1

Exercise 7.2.3-2: Consider the binary image below. Find the 4-V-adjacent, 8-V-adjacent and m-V-adjacent pixels given that $V = \{1\}$.

0	0	0	0	0	0	1
1	0	1	1	0	0	1
0	0	1	(1)	0	1	0
0	0	0	0	1	0	0
0	1	1	0	0	1	1

Exercise 7.2.3-3: Consider the 3-bit quantized grayscale image below. Find the 4-V-, 8-V- and m-V-adjacency, given $V = \{4, 6, 7\}$.

1	2	7	0	6	5	6
3	0	3	4	7	0	4
0	1	2	(6)	1	5	1
5	2	4	0	6	3	2
0	0	3	7	4	3	1

Exercise 7.2.3-4: Consider the binary image below. Find the shortest V-path between the circled pixels assuming $V = \{1\}$. If it exists, determine the path's length. Determine the V-component in which the pixels reside. Do this for the concept of 4-V, 8-V and m-V-paths.

0	1	1	0	0	0	1
(1)	0	1	1	0	0	1
0	0	1	1	1	1	0
0	0	0	0	1	0	0
0	1	1	0	0	1	(1)

Exercise 7.2.3-5: Consider the 3-bit quantized grayscale image below. Find the 4-V-adjacent, 8-V-adjacent and m-V-adjacent paths given that $V = \{0, 2, 4, 6\}$.

1	2	7	0	6	5	6
3	0	3	4	7	0	(4)
0	1	2	6	1	5	1
5	2	4	0	6	3	2
0	(0)	3	7	4	3	4

7.2.4 Concepts related to regions/sets of pixels

7.2.4.1 Reflection and translation

Consider a set D of pixels p. We can consider these pixels to be vectors $\vec{p}$ (with coordinates $[x, y]$).

$$D = \{\vec{p}\}$$

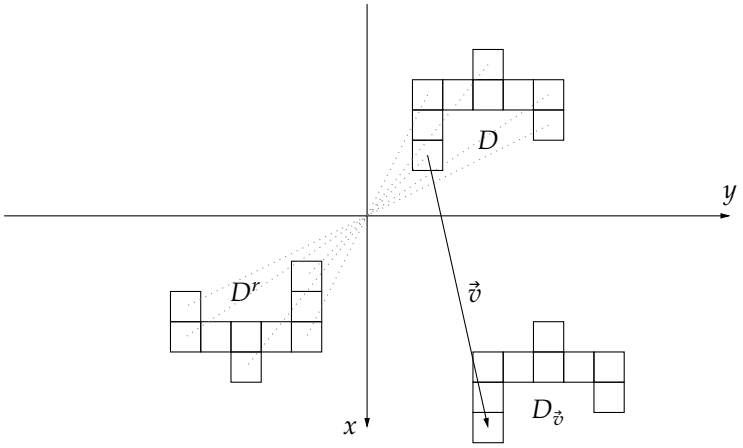


Figure 7.4: Illustration of the concept of reflection and translation of a set of pixels D (indicated with squares in the second quadrant) into D^r and $D_{\vec{v}}$ respectively.

Reflection

The reflection of a set of pixels D is denoted by D^r and defined as:

$$D^r = \{\vec{q} \mid -\vec{q} \in D\}$$

In mathematical jargon this is a so-called *point reflection with the origin as reflection center*.

Translation

The translation of a set of pixels D by a (fixed) vector $\vec{v}$ is denoted as $D_{\vec{v}}$ and defined as:

$$D_{\vec{v}} = \{\vec{q} \mid \vec{q} - \vec{v} \in D\}\,.$$

Note: the vector arrows on top of the symbols are often omitted.

The concepts of translation and reflection have been illustrated in Figure 7.4.

7.2.4.2 Foreground and background

For binary images Binary images only have two types of pixels. The pixels with value 1 compose the foreground. The pixels with value 0 compose the background.

Formally, for an image $f(\vec{p})$:

- Foreground: $A = \{\vec{p} \mid f(\vec{p}) = 1\}$
- Background: $B = \{\vec{p} \mid f(\vec{p}) = 0\}$

For grayscale images For grayscale images the concepts of foreground and background are a bit more *fuzzy*. The latter is pun intended, as we will be borrowing the concept of fuzzy sets

from the theory of fuzzy logic.

We consider the value of the pixel normalized to the range 0 to 1 as a membership function describing the pixel's degree of membership to the foreground. We call these normalized intensity functions *intensity membership functions*. The higher the value, the more the pixel belongs to the foreground. A white pixel (with a value of 1) will be fully member of the foreground. The lower the intensity value, the less it belongs to the foreground. A black pixel (with a value of 0) will be in no way member of the foreground.

7.2.4.3 Set calculus

For binary images If pixels are gathered in sets, we can perform traditional set calculus on them. Consider two pixel sets A and B, then the following set operations can be defined. The set operations operate on sets and give a set as result.

Union $C = A \cup B = \{\vec{p} \mid \vec{p} \in A \text{ or } \vec{p} \in B\}$

Intersection $C = A \cap B = \{\vec{p} \mid \vec{p} \in A \text{ and } \vec{p} \in B\}$

Complement $C = A^c = \{\vec{p} \mid \vec{p} \notin A\}$

Note the danger in the notation: the symbol C denotes a set while the superscript ^{c} denotes the complement!

Set difference $C = A \setminus B = \{\vec{p} \mid \vec{p} \in A \text{ but } \vec{p} \notin B\}$

For grayscale images The situation for grayscale images is a bit more complex. The sets are no longer binary (an element belongs to a set or not), but we have to deal with partial membership of a set.

Luckily the basic set operation definitions (for the binary case) can be extended in a consistent way to the notion of fuzzy sets. To stress the fact that these are no ordinary set operations, we call them pointwise operations and give them some dedicated symbols.[1] The pointwise operations operate on intensity functions and yield new intensity functions as result. Consider two intensity membership functions $f_A(\vec{p})$ and $f_B(\vec{p})$, then we can define:

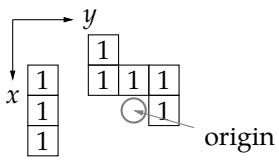
Pointwise union $f_C(\vec{p}) = f_A(\vec{p}) \vee f_B(\vec{p}) = \max(f_A(\vec{p}), f_B(\vec{p}))$

Pointwise intersection $f_C(\vec{p}) = f_A(\vec{p}) \wedge f_b(\vec{p}) = \min(f_A(\vec{p}), f_B(\vec{p}))$

Pointwise complement $f_C(\vec{p}) = f_A^c(\vec{p}) = 1 - f_A(\vec{p})$

Again, note the danger in the notation: the subscript C denotes a particulare image while the superscript ^{c} denotes the complement!

[1] The specific adjective 'pointwise' also suggests that it is possible to create a set of spatial definitions as well. We will not treat this possibility.



(a) Graphical representation

$$SE = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & \mathbf{0} & 1 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

(b) Matrix representation

Figure 7.5: Example of a structuring element for binary images

Pointwise difference $f_C(\vec{p}) = f_A(\vec{p}) \setminus f_B(\vec{p}) = \max(0, f_A(\vec{p}) - f_B(\vec{p}))$

7.2.4.4 Structuring Elements

A basic ingredient for the morphological operations are so-called *structuring elements*. These are image fragments that help obtain a specific effect on an image or to discover a specific structure in the image.

For binary images A structuring element (SE) for binary images is an image fragment (a set of pixels) that

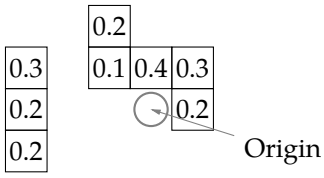
- is usually of limited size,
- is not necessarily connected,
- has an origin, that does not have to be part of the SE.

The origin is the spot where you 'grab' the structuring element to later translate it, or to rotate it 180° to reflect it. An example of a structuring element can be found in Figure 7.5. The example given is quite uncommon, but still a valid structuring element. Often structuring elements are symmetrical around the origin. In the graphical representation, the origin is indicated using a circle. In the matrix representation on the right, a value of 1 means the pixel is in the SE, a value of 0 means it is not in the SE. It is also assumed one can indicate the origin (e.g., using boldface or with a circle). If not, the SE is extended with extra zero columns or rows such that the origin lies in the middle. E.g., MATLAB does not allow boldface or circles around values, so there extra zeros are added, and the origin is the center pixel.

For grayscale images A structuring element (SE) for grayscale images is an image fragment (a set of pixels) that

- is usually of limited size,
- is not necessarily connected,
- has an origin, that does not have to be part of the SE,
- has an intensity value per pixel contained in the SE.

An example of a structuring element can be found in Figure 7.6. The example given is quite uncommon, but still a valid structuring element. Often structuring elements are symmetrical



(a) Graphical representation

$$SE = \begin{bmatrix} - & - & 0.2 & - & - \\ 0.3 & - & 0.1 & 0.4 & 0.3 \\ 0.2 & - & - & - & 0.2 \\ 0.2 & - & - & - & - \end{bmatrix}$$

(b) Matrix representation

Figure 7.6: Example of a structuring element for grayscale images

around the origin. In the graphical representation, the origin is indicated using a circle. In the matrix representation on the right pixels that are in the SE are indicated by an intensity value (that may be negative or zero!), empty matrix positions correspond to pixels that are not part of the SE. Again, it is assumed that one can indicate the origin (e.g., using boldface or with a circle). If not, the SE is extended with extra empty columns or rows such that the origin lies in the middle.

When every pixel of the SE has the same intensity value, we obtain a so-called *flat structuring element*. If not, we have a so-called *umbrella structuring element*. In most cases we will restrict ourselves to flat structuring elements.

7.3 Morphology for binary images

In this section, we will discuss the morphology for binary images. Binary images are often generated from grayscale images using a threshold operation. Therefore, binary images are no kindergarten toy we use to study morphology, but contribute to real life image analysis.

7.3.1 Basic operations

Let's start with the 'addition and subtraction' of mathematical morphology, i.e. dilation and erosion:

- dilation will put an extra layer to a foreground object
- erosion will remove a layer from a foreground object

Let's crisply define these operations.

7.3.1.1 Dilation

Use The dilation adds an extra layer to a foreground object, making it a little fatter.

Definition

Dilation of binary images

The dilation of a binary image A with a structuring element B yields a new image C and is denoted as

$$C = A \oplus B$$

and defined as

$$A \oplus B = \left\{ \vec{v} \mid B^r_{\vec{v}} \cap A \neq \emptyset \right\}. \tag{7.6}$$

In words: take the structuring element B, reflect it and translate it over all pixels of the image A. For every position you hold the SE at, for which the intersection of the SE and the (foreground) image A is not empty, the pixel is part of the foreground of C. If there is no overlap the pixel is part of the background of C.

If you wonder why the SE has been reflected, read on. The explanation will come.

This has been illustrated in Figure 7.7.

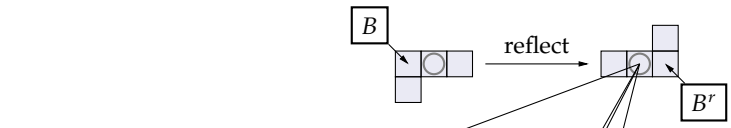
Properties The dilation is:

- commutative
$$A \oplus B = B \oplus A$$
- associative
$$(A \oplus B) \oplus C = A \oplus (B \oplus C)$$
- distributive w.r.t. the union operation
$$A \oplus (B \cup C) = (A \oplus B) \cup (A \oplus C)$$
- monotonic w.r.t. the subset operation
$$A \subset B \quad \Rightarrow \quad A \oplus C \subset B \oplus C$$

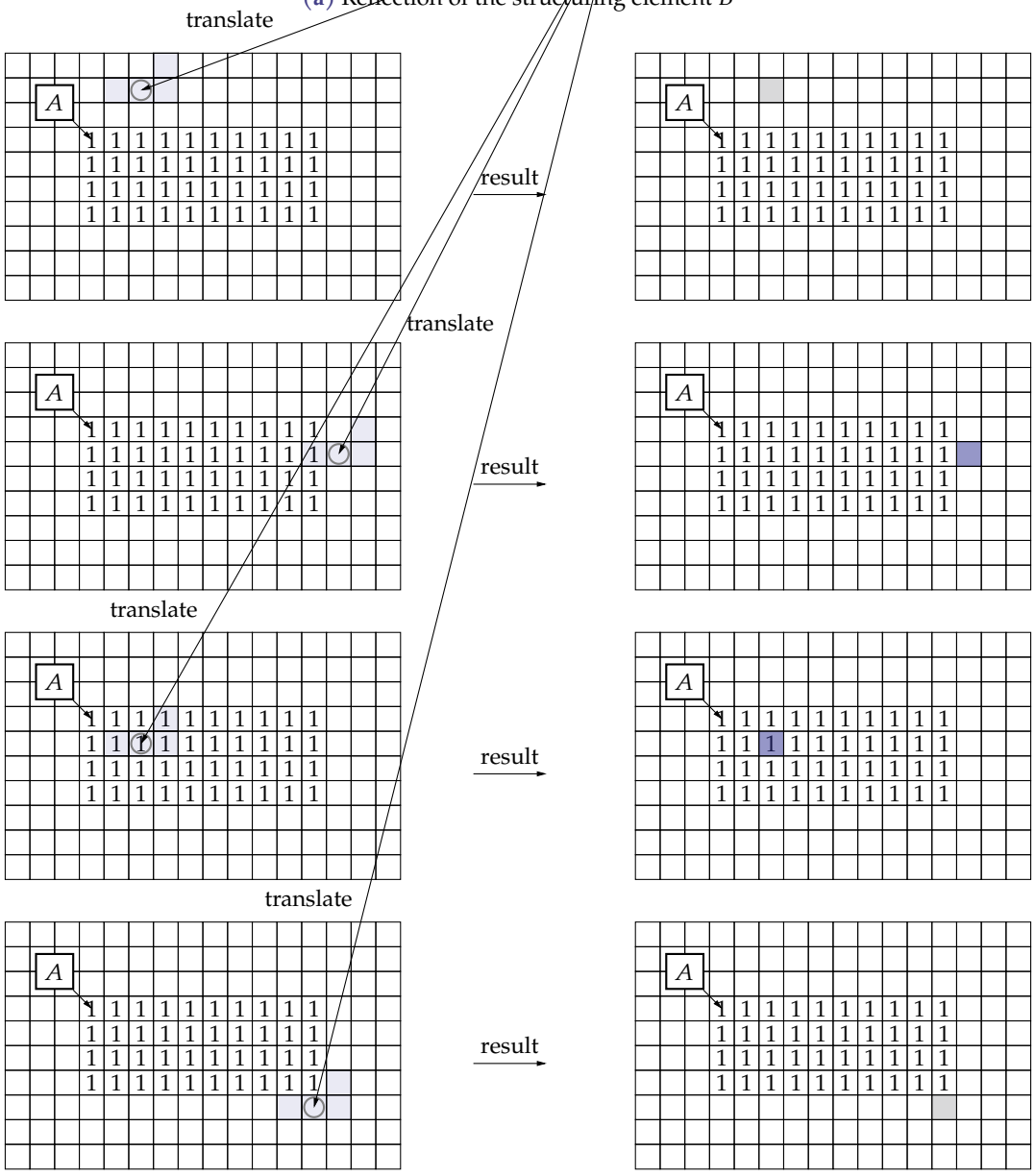
Alternative definition It is possible to give an equivalent definition of dilation, that simplifies executing it.

$$A \oplus B = \bigcup \{ B_{\vec{v}} \mid \vec{v} \in A \}$$

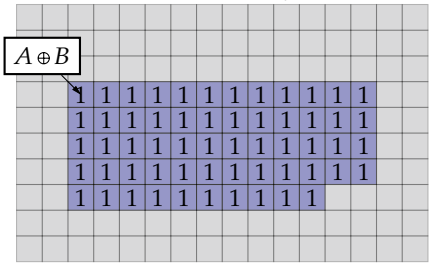
In words: do *not* reflect the structuring element, translate it to every pixel of A. All pixels that were ever covered by $B_{\vec{v}}$ belong to the dilated result.
Without having proof of this, I suspect that this was how dilation was conceived at first. The alternative definition is simpler, more intuitive and it doesn't require a reflection. However, this way of thinking is not easily extended to grayscale images. The original definition of (7.3.1.1) is easily extended to grayscale images.



(a) Reflection of the structuring element B



(b) Left column: translated reflected SE $B^r_{\vec{v}}$, to be checked for overlap with A; right column: resulting pixel (light gray: $B^r_{\vec{v}} \cap A = \emptyset$, therefore not part of the resulting set; blue: $B^r_{\vec{v}} \cap A \neq \emptyset$ and therefore part of the resulting set)



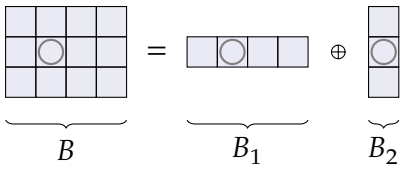
(c) Resulting dilated image (in blue)

Figure 7.7: Dilation of binary images: example

Decomposition of structuring elements The computational effort of the dilation-operation is proportional to the number of pixels in the SE. Associativity allows for the decomposition of structuring elements enabling sequential dilation:

$$A \oplus B = A \oplus (B_1 \oplus B_2) = (A \oplus B_1) \oplus B_2$$

For example, consider the following decomposition:



This allows reducing the computational effort from 12 (= 4×3) to 7 (= 1×4 + 3×1).

7.3.1.2 Erosion

Use The erosion removes a layer from a foreground object, making it a little slimmer.

Definition

Erosion of binary images

The erosion of a binary image A with a structuring element B yields a new image C and is denoted as

$$C = A \ominus B$$

and defined as

$$A \ominus B = \{\vec{v} \mid B_{\vec{v}} \subseteq A\}.$$

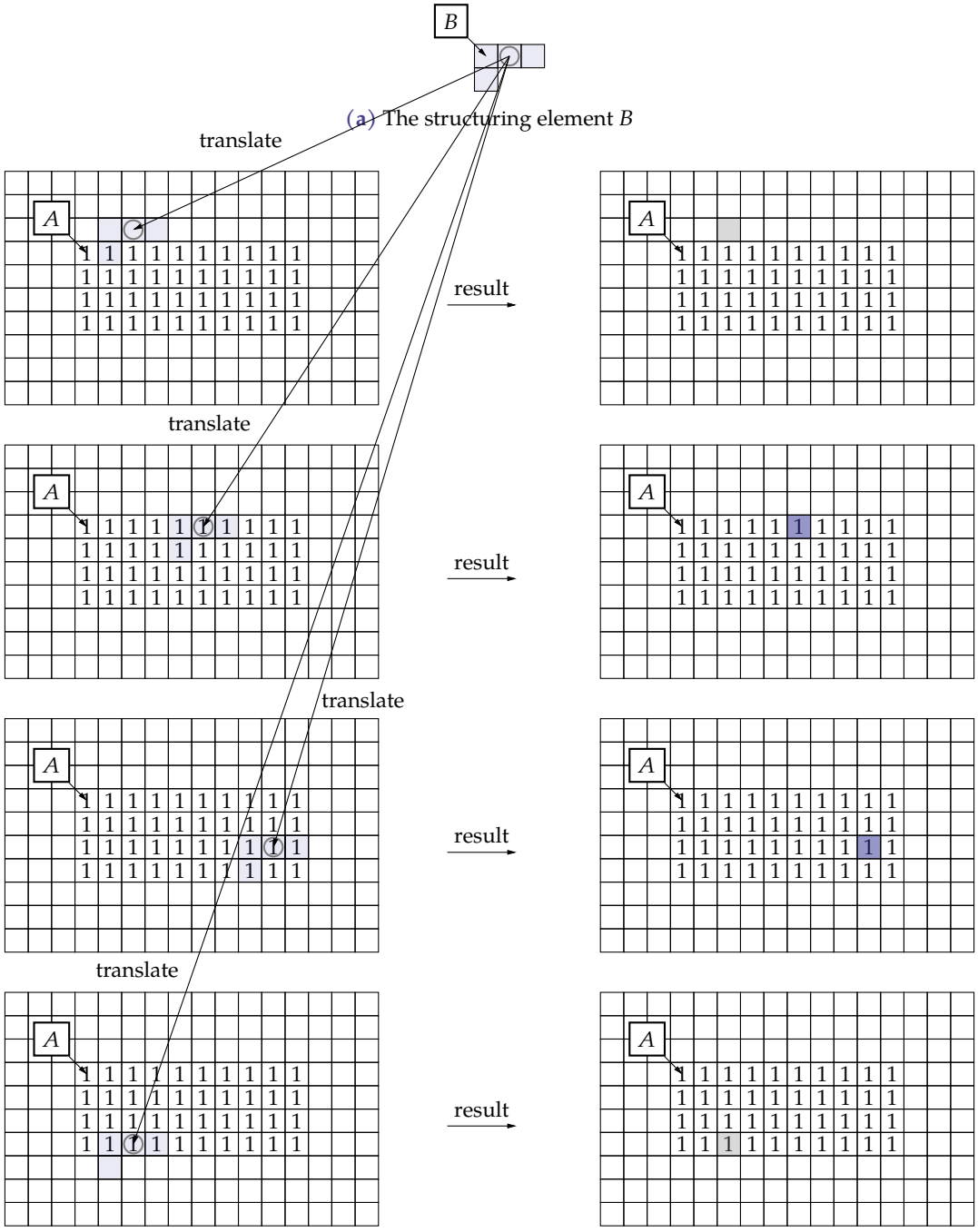
In words: take the structuring element B by it's origin and translate it over all pixels of the image A. For every position you hold the SE at, for which the SE is fully contained in the (foreground) image A, the pixel is part of the foreground of C. If not, the pixel is part of the background of C.

This has been illustrated in Figure 7.8.

The erosion in fact is also a basic tool for pattern recognition. It shows us the pixel positions for which the structuring element fully fits within the foreground objects. This property will be used later in more complex operations like for example the hit/miss transformation and the opening by reconstruction.

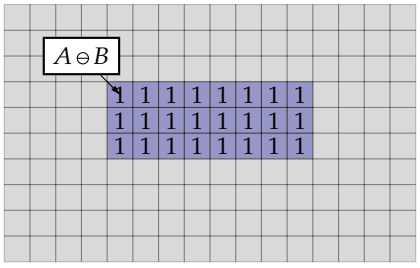
Properties The erosion is:

- not commutative!
- not associative!



(a) The structuring element B

(b) Left column: translated SE $B_{\vec{v}}$, to be checked for containment in A; right column: resulting pixel (light gray: $B_{\vec{v}} \not\subseteq A$, therefore not part of the resulting set; blue: $B_{\vec{v}} \subseteq A$ and therefore part of the resulting set)



(c) Resulting eroded image (in blue)

Figure 7.8: Erosion of binary images: example

- distributive w.r.t. the intersection operation

$$A \ominus (B \cap C) = (A \ominus B) \cap (A \ominus C)$$

- monotonic w.r.t. the subset operation

$$A \subset B \quad \Rightarrow \quad A \ominus C \subset B \ominus C$$

7.3.1.3 Duality of erosion and dilation

Duality of erosion and dilation

The erosion and dilation are dual operations, i.e.

$$A \oplus B = (A^c \ominus B^r)^c \qquad A \ominus B = (A^c \oplus B^r)^c$$

In words: dilation of the foreground corresponds to erosion of the background with the reflected structuring element.

Proof
Starting from the dilation:

$$\begin{aligned} A \oplus B &= \left((A \oplus B)^c\right)^c \\ &= \left(\left(\{\vec{v} \mid B^r_{\vec{v}} \cap A \neq \emptyset\}\right)^c\right)^c \\ &= \left(\{\vec{v} \mid B^r_{\vec{v}} \subseteq A^c\}\right)^c \\ &= (A^c \ominus B^r)^c \end{aligned}$$

Starting from the erosion:

$$\begin{aligned} A \ominus B &= \left((A \ominus B)^c\right)^c \\ &= \left(\left(\{\vec{v} \mid B_{\vec{v}} \subseteq A\}\right)^c\right)^c \\ &= \left(\{\vec{v} \mid B_{\vec{v}} \cap A^c \neq \emptyset\}\right)^c \\ &= (A^c \oplus B^r)^c \end{aligned}$$

■

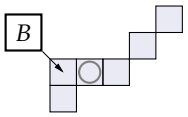
Duality is a very common property. It may allow simplifying complex morphological operations, in a similar manner as De Morgan's laws do for Boolean algebra. Note how striking the similarity is:

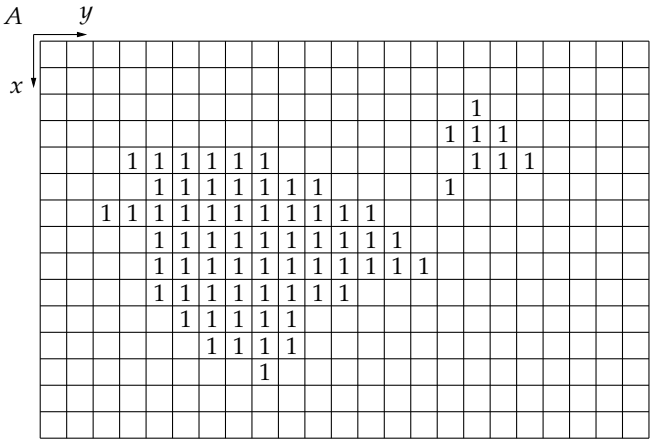
$$A \,\&\, B = \overline{\overline{A} \mid \overline{B}} \qquad A \mid B = \overline{\overline{A} \,\&\, \overline{B}}$$

with & denoting the logical AND operation, | denoting the logical OR operation, and the line on top of the symbols the NOT operation.

Exercises

Exercise 7.3.1.3-1: Consider the image A below. Erode and dilate A using the structuring element B.





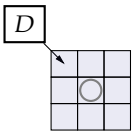
Exercise 7.3.1.3-2: (*) Consider the dilated result of the previous exercise and erode that using the same structuring element B. Do you regain the original?
In symbols:

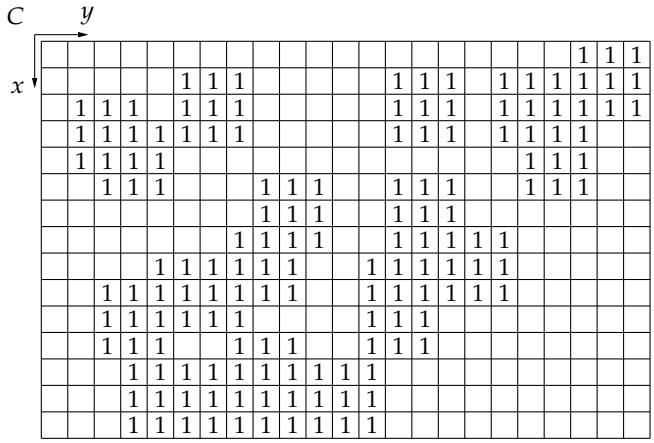
$$(A \oplus B) \ominus B \stackrel{?}{=} A$$

Likewise, consider the eroded result of the previous exercise and dilate that using the same structuring element B. Do you regain the original?
In symbols:

$$(A \ominus B) \oplus B \stackrel{?}{=} A$$

Exercise 7.3.1.3-3: Consider the image C below. Erode and dilate C using the structuring element D.





Exercise 7.3.1.3-4: Read the documentation of the MATLAB function `strel`. Learn how to use it to compose structuring elements.

Read the documentation of the MATLAB functions `imdilate` and `imerode`. Learn how to use them.
Redo the previous exercises using these MATLAB functions.

7.3.2 Derived operations

Based on the basic dilation and erosion for binary images, we can define a number of derived operations:

- opening and closing
- thinning and thickening
- hit/miss transformation

7.3.2.1 Opening

Use The opening

- removes hairline protuberances, and
- grows hairline cracks and trenches.

Definition

Opening of binary images

The opening of a binary image A with a structuring element B yields a new image C and is denoted as

$$C = A \circ B$$

and defined as

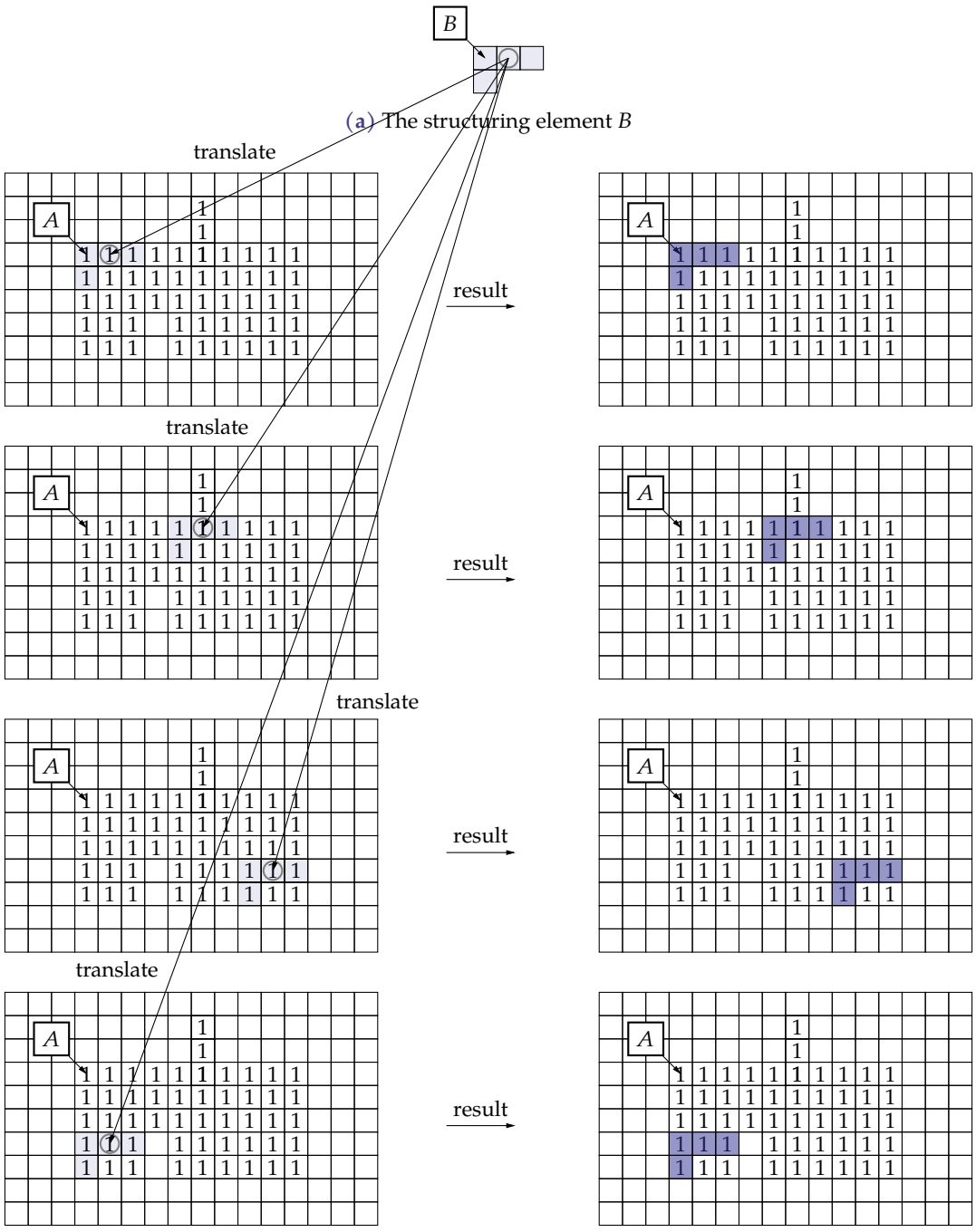
$$A \circ B = (A \ominus B) \oplus B.$$

Alternative definition An alternative definition of the opening will allow us to determine the opening in a single step process.

$$\begin{aligned} A \circ B &= (A \ominus B) \oplus B \\ &= \{\vec{v} \mid B_{\vec{v}} \subseteq A\} \oplus B \\ &= \bigcup \{B_{\vec{v}} \mid \vec{v} \in \{\vec{v} \mid B_{\vec{v}} \subseteq A\}\} \qquad \text{(using)} \\ &= \bigcup \{B_{\vec{v}} \mid B_{\vec{v}} \subseteq A\} \end{aligned}$$

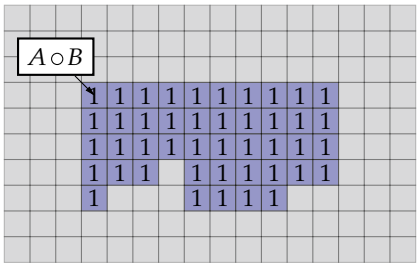
In words: take the structuring element B and translate it over all pixels for which the structuring element is fully contained within the image A. All pixels the structuring element covered, are part of the resulting opening.

This has been illustrated in Figure 7.9.



(a) The structuring element B

(b) Left column: translated SE $B_{\vec{v}}$ such that it is fully contained in A; right column: resulting pixels covered by the SE in blue



(c) Resulting opened image (in blue)

Figure 7.9: Opening of binary images: example

7.3.2.2 Closing

Use The closing

- removes hairline cracks and trenches, and
- grows hairline protuberances.

Definition

Closing of binary images

The closing of a binary image A with a structuring element B yields a new image C and is denoted as

$$C = A \bullet B$$

and defined as

$$A \bullet B = (A \oplus B) \ominus B.$$

Alternative definition An alternative definition of the closing will allow us to determine the closing in a single step process.

$$\begin{aligned} A \bullet B &= \left(((A \oplus B) \ominus B)^c\right)^c \\ &= \left((A \oplus B)^c \oplus B^r\right)^c \\ &= \left((A^c \ominus B^r) \oplus B^r\right)^c \\ &= (A^c \circ B^r)^c \end{aligned}$$

In words: take the structuring element B, reflect it and use it to open the background. Then invert the background to obtain the foreground.

This has been illustrated in Figure 7.10.

7.3.2.3 Hit/miss transformation

Use The hit/miss-transformation is used to recognize objects with a specific shape.

Definition

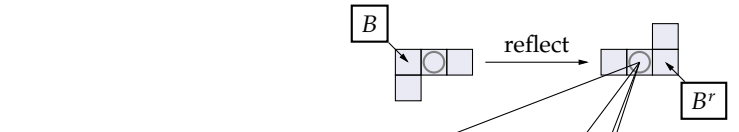
Hit/miss transformation for binary images The hit/miss transformation of a binary image A with a pair of structuring elements (B_f, B_b) yields a new image C and is denoted as

$$C = A \circledast (B_f, B_b)$$

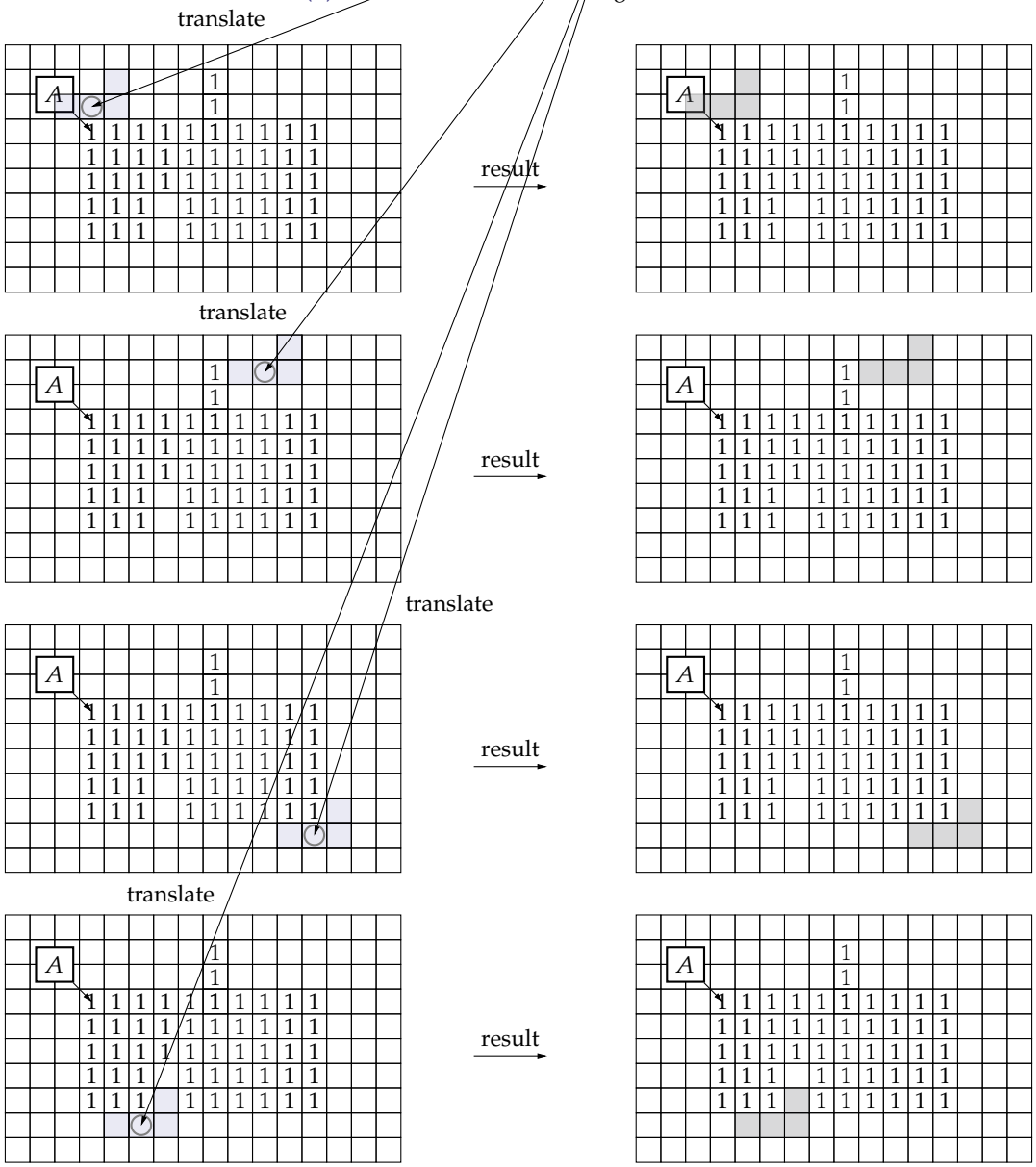
and defined as

$$A \circledast (B_f, B_b) = \left(A \ominus B_f\right) \cap \left(A^c \ominus B_b\right).$$

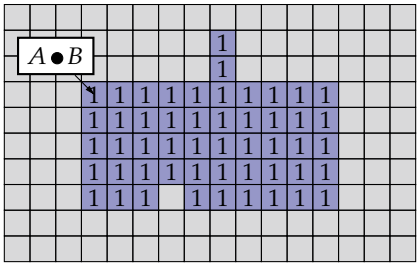
The choice of the pair of structuring elements (B_f, B_b) is key:



(a) Reflection of the structuring element B

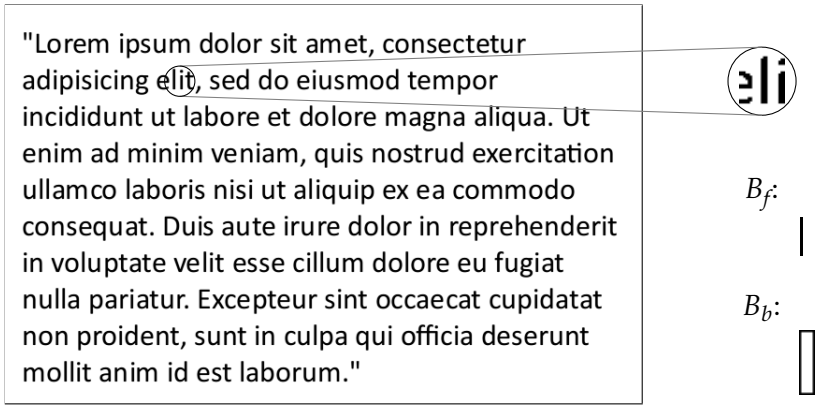


(b) Left column: translated and reflected SE $B^r_{\vec{v}}$ such that it is fully contained in A^c; right column: resulting pixels covered by the SE in light gray

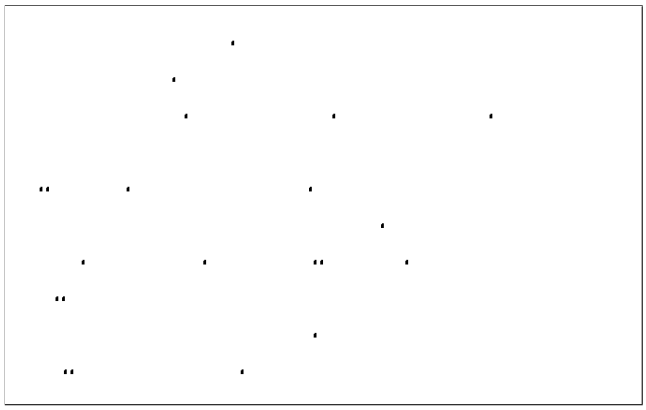


(c) Resulting closed image (in blue)

Figure 7.10: Closing of binary images: example



(a) Text fragment (as a binary image with white background and black foreground) in which we'd like to detect the character 'l', using an appropriate set of structuring elements (B_f, B_b). The foreground SE B_f fits in the letter l, but also in the letters p, d, ...The background SE B_b only fits around an l and not around the letters, p, d, ...Therefore the combination of the two effectively only selects the letter l.



(b) Result of applying the hit/miss transformation; the dots mark the pixels at which the set of structuring elements found a match

Figure 7.11: Illustration of the hit/miss transformation

- B_f is the SE that needs to fit within the foreground of the object sought
- B_b is the SE that needs to fit within the background of the object sought

Alternative definition An alternative definition of the hit/miss transformation is commonly found in literature, but does not help in gaining more insight.

$$\begin{aligned} A \circledast (B_f, B_b) &= \left(A \ominus B_f\right) \cap \left(A^c \ominus B_b\right) \\ &= \left(A \ominus B_f\right) \setminus \left(A^c \ominus B_b\right)^c \\ &= \left(A \ominus B_f\right) \setminus \left(A \oplus B_b^r\right) \end{aligned}$$

As an example, consider an arbitrary text fragment (see Figure 7.11a).

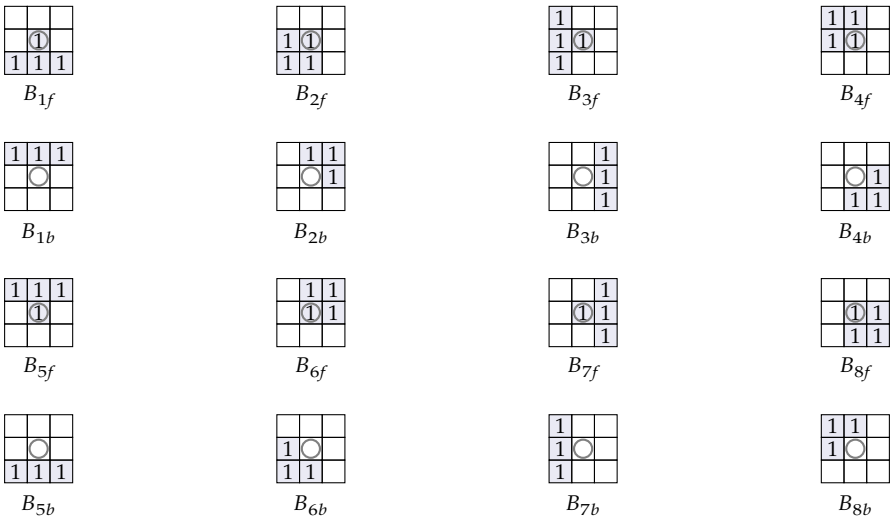


Figure 7.12: An appropriate set of thinning elements for skeletonization

7.3.2.4 Thinning

Use Thinning is often used to reduce an object to a wire frame representing the object.

Definition

Thinning of binary images

The thinning of a binary image A using a pair of structuring elements (B_f, B_b) yields a new image C and is denoted as

$$C = A \otimes (B_f, B_b)$$

and defined as

$$A \otimes (B_f, B_b) = A \setminus \left(A \circledast (B_f, B_b) \right).$$

In words: we remove the specific shapes from A recognized by (B_f, B_b): this makes A thinner.

Sequential thinning It makes sense to apply a sequence of thinning operations using a set of SE pairs. This is called *sequential thinning*.

$$A \otimes \left\{ (B_{if}, B_{ib}) \right\} = (\cdots ((A \otimes (B_{1f}, B_{1b})) \otimes (B_{2f}, B_{2b})) \cdots) \otimes (B_{nf}, B_{nb})$$

Sequential thinning converges to a stable result. If one picks the right set of SE pairs, it is possible to skeletonize A. An appropriate choice can be found in Figure 7.12.

An example of a thinning operation can be found in Figure 7.13. Note that skeletonization generates a substantial amount of hairline protuberances: these are usually trimmed. The generated loops can be avoided by filling the holes first. The obtained skeletons can be used to measure characteristic angles and lengths to ease the matching of the objects to a database.

(a) Original image of a finger print (black = foreground, white = background)

(b) Image after thinning (black = foreground, white = background

(c) Original radiographic image of a weapon (white = foreground, black = background)

(d) Image after thinning (black = foreground, white = foreground)

Figure 7.13: Illustration of sequentially thinning images until convergence (skeletonization)

7.3.2.5 Thickening

Use The operation 'thickening' is not very often used in practice. Either the background is thinned instead, or the cheaper dilation operation is used. Still, for completeness, we provide its definition.

Definition

Thickening of binary images

The thickening of a binary image A using a pair of structuring elements (B_f, B_b) yields a new image C and is denoted as

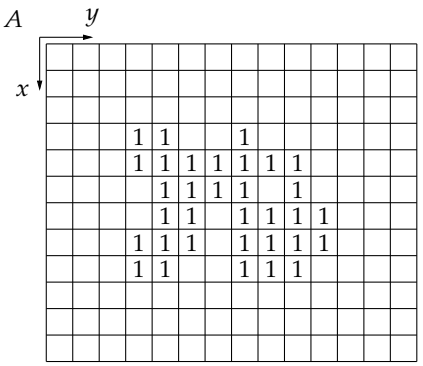
$$C = A \odot (B_f, B_b)$$

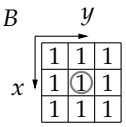
and defined as

$$A \odot (B_f, B_b) = A \cup \left(A \circledast (B_f, B_b) \right).$$

Exercises

Exercise 7.3.2.5-1: Consider the image A and the structuring element B below.

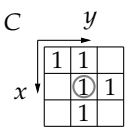




Now:

- calculate $A \circ B$.
- calculate $A \bullet B$.

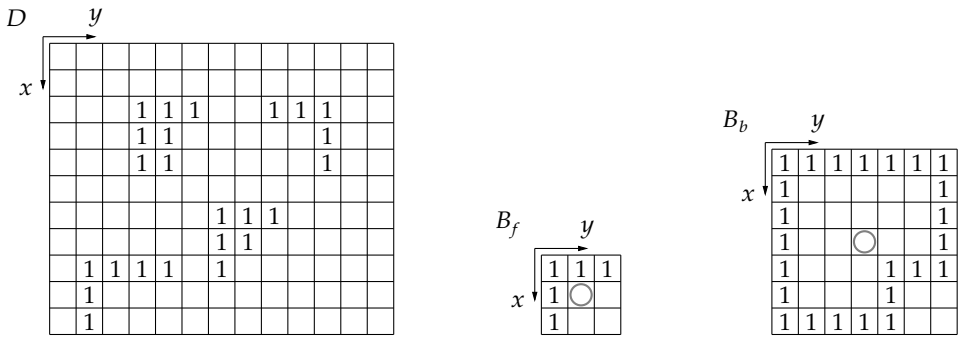
Exercise 7.3.2.5-2: Consider the image A of the previous exercise and the structuring element C below.



Now:

- calculate $A \circ C$.
- calculate $A \bullet C$.

Exercise 7.3.2.5-3: Consider the image D below, together with a pair of structuring elements (B_f, B_b).



Calculate $E = D \circledast (B_f, B_b)$.

Exercise 7.3.2.5-4: Try to find a set of structuring elements (B_f, B_g) to detect the letter i in a text using a sans-serif font.

Exercise 7.3.2.5-5: Try to find a set of structuring elements (B_f, B_g) to detect double spaces in a text using a sans-serif font.

Exercise 7.3.2.5-6: Try to find a set of structuring elements (B_f, B_g) to detect the letter o in a text using a sans-serif font.

Exercise 7.3.2.5-7: Read the documentation of the MATLAB functions `strel`, `imopen`, `imclose`, `bwhitmiss` and `bwmorph`. Learn how to use them.
Redo the previous exercises using these MATLAB functions.

7.3.3 Geodesic operations

In the operations (erosion, dilation, opening, closing, ...) we've treated so far, the structuring element has a tremendous influence on the end result. The structuring element is kind of like the paint brush to the painter. One can't paint fine strokes using a fat brush.

The *geodesic* operations tackle this issue, by adding a masking operation to the basic operations to correct for the boldness or the specific shape of the brush. To continue on the painter's metaphor: it is like using masking tape when painting a decoration on a wall; the masking tape avoids painting areas that should not carry any paint. It diminishes the influence of the brush on the end result.

The geodesic operations will enable us to fully reconstruct parts of an image.

7.3.3.1 Geodesic dilation

Use The geodesic dilation will allow dilating an object without growing beyond the borders imposed by a chosen mask. Its true value will become apparent when going one step further, i.e. performing *iterative* geodesic dilation.

Definition

Geodesic dilation of binary images

Geodesic dilation of an image X using a structuring element B, with respect to a masking image M yields a new image C and is denoted as

$$C = D_M(X)$$

and is defined as

$$D_M(X) = (X \oplus B) \cap M.$$

Note the absence of the structuring element B in the basic notation $D_M(X)$. This stresses the smaller importance of the structuring element. Very often a $N \times N$ square or disc-shaped SE is used.

Iterative geodesic dilation Often the operation is used iteratively. Therefore we agree on the notation:

$$D_M^{(n)}(X) = D_M\left(D_M^{(n-1)}(X)\right)$$

with

$$D_M^{(1)}(X) = D_M(X).$$

The iterative application of the geodesic dilation converges to a stable image.[2] Applying it until convergence occurs, is the so-called *reconstruction by dilation* and is denoted as:

$$R_{D,M}(X) = D_M^{(k)}(X)$$

with k such that $D_M^{(k+1)}(X) = D_M^{(k)}(X)$.

This has been illustrated in Figure 7.14. Note how reconstruction by dilation allows marking one of the objects in the image and fully reconstructing it. This also explains the specific name for the argument of the geodesic dilation operation: the marker.[3]

7.3.3.2 Geodesic erosion

The dual of geodesic dilation is geodesic erosion.

[2] In general, this statement is incorrect. Limit cycles may appear if the center pixel is no part of the structuring element!

[3] And as always: x marks the spot!

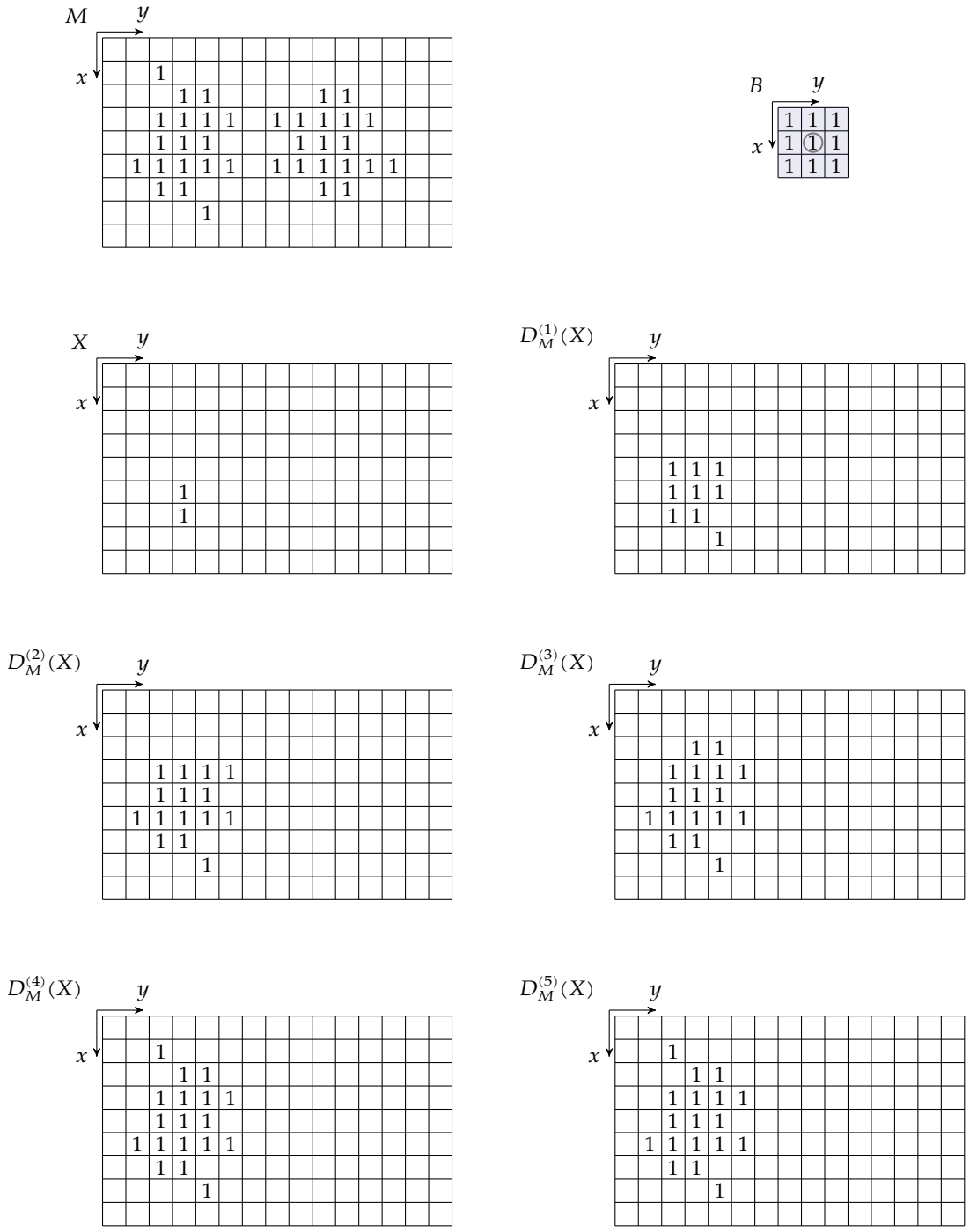


Figure 7.14: Illustration of reconstruction by dilation of marker X, using structuring element B and mask M; note that convergence appears after 5 iterations.

Use The geodesic erosion will allow eroding an object without shrinking below the borders imposed by a chosen mask. Again, this operation is only really useful when performed iteratively. In addition, most often it is seen as a dilation of the background with the reflected structuring element.

Definition

Geodesic erosion of binary images

Geodesic erosion of an image X using a structuring element B, with respect to a masking image M, yields a new image C and is denoted as

$$C = E_M(X)$$

and is defined as

$$E_M(X) = (X \ominus B) \cup M.$$

Note the absence of the structuring element B in the basic notation $R_M(X)$. This stresses the smaller importance of the structuring element. Very often an $N \times N$ square or disc-shaped SE is used.

Iterative geodesic erosion Often the operation is used iteratively. Therefore we agree on the notation:

$$E_M^{(n)}(X) = E_M\left(E_M^{(n-1)}(X)\right)$$

with

$$E_M^{(1)}(X) = E_M(X).$$

The iterative application of the geodesic erosion converges to a stable image. Applying it until convergence occurs is the so-called *reconstruction by erosion* and is denoted as:

$$R_{E,M}(X) = E_M^{(k)}(X)$$

with k such that $E_M^{(k+1)}(X) = E_M^{(k)}(X)$.

7.3.3.3 Opening by reconstruction

Use In a classical *opening*, the erosion performs a pattern recognition (it finds the pixel positions at which the structuring element fully fits within the foreground objects). Then the subsequent dilation recomposes the areas the structuring element touched when a match was found. However, the resulting shape is dependent on the shape of the structuring element (remember the story of the painter and his brush).

Let's replace the dilation with a reconstruction by dilation, such that the original objects are fully restored without letting the brush's shape interfere.[4]

[4]Of course the structuring element of the dilation should be versatile enough to allow reconstruction in all desired directions.

Definition

Opening by reconstruction of binary images

Opening by reconstruction of an image A using an erosion structuring element B_e and a geodesic dilation structuring element B_d yields a new image C and is denoted as

$$C = O_R^{(n)}(A)$$

and is defined as

$$O_R^{(n)}(A) = R_{D,A}(A \ominus nB_e)$$

in which $A \ominus nB_e$ means eroding the image A multiple (n) times by a specific B_e:

$$A \ominus nB_e = (\cdots((A \underbrace{\ominus B_e) \ominus B_e) \cdots) \ominus B_e}_{\text{n times}}$$

Usually, a different structuring element B_d is used for the geodesic dilation.

Very often a specific B_d is used depending on the adjacency definition one wants to use as a basis for distinguishing components:

- a cross in case of 4-V-adjacency,
- a square in case of 8-V-adjacency.

The operation has been illustrated in Figure 7.15

7.3.3.4 Closing by reconstruction

Use The *closing by reconstruction* is most easily understood by considering it as the *opening by reconstruction* of the background (with a reflected structuring element).

Definition

Closing by reconstruction of binary images

Closing by reconstruction of an image A using a dilation structuring element B_d and a geodesic erosion structuring element B_e yields a new image D and is denoted as

$$D = C_R^{(n)}(A)$$

and is defined as

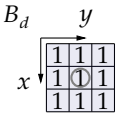
$$C_R^{(n)}(A) = R_{E,A}(A \oplus nB_e)$$

in which $A \oplus nB_e$ means dilating the image A multiple (n) times by a specific B_e:

$$A \oplus nB_e = (\cdots((A \underbrace{\oplus B_e) \oplus B_e) \cdots) \oplus B_e}_{\text{n times}}$$

Usually, a different structuring element B_e is used for the geodesic erosion.

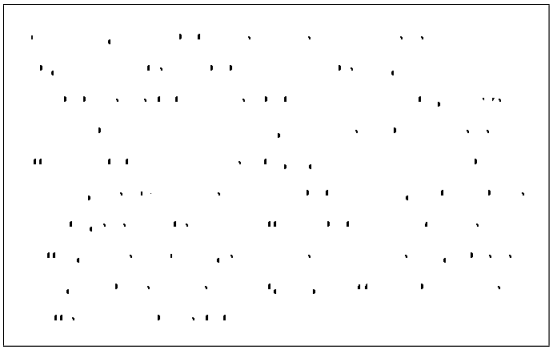
(a) Structuring element B_e used for the erosion (a vertical bar, to scale). This element was chosen, because it fits in all letters exhibiting a long vertical bar (such as the letters b,d,f,h,l,p,q,t)



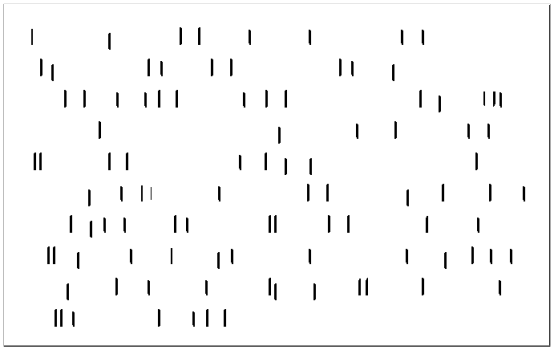
(b) Structuring element B_d used for the geodesic dilation (a 3 × 3 square, not to scale)

"Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum."

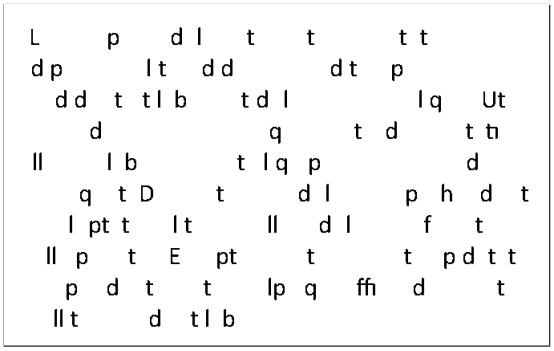
(c) Original image of the text fragment (black = foreground, white = background)



(d) Intermediate result after one erosion with B_e



(e) Result of an ordinary opening



(f) Final result of opening by reconstruction

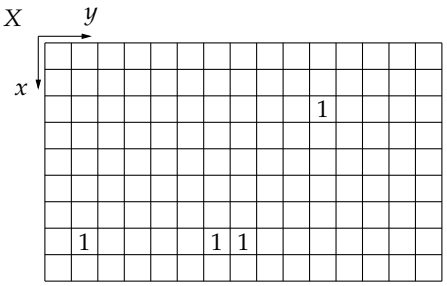
Figure 7.15: Illustration of opening by reconstruction on a text fragment to reconstruct only the characters that contain a long vertical bar (black = foreground, white = background). The frames around the images are not part of the image.

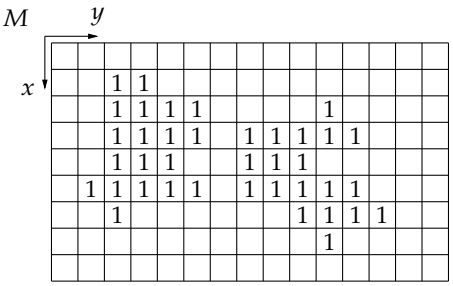
Very often a specific B_e is used depending on the adjacency definition one wants to use as a basis for distinguishing components:

- a cross in case of 4-V-adjacency,
- a square in case of 8-V-adjacency.

Exercises

Exercise 7.3.3.4-1: Consider the marker image X and the mask M below.

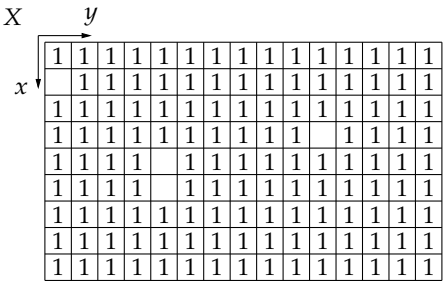


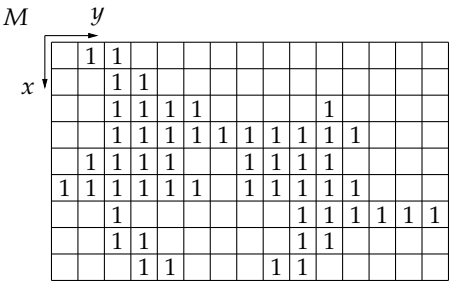


Calculate the reconstruction by dilation of X w.r.t. the mask M

- using a 3×3 disc-shaped structuring element with radius 1,
- using a 3×3 square structuring.

Exercise 7.3.3.4-2: Consider the marker image X and the mask M below.

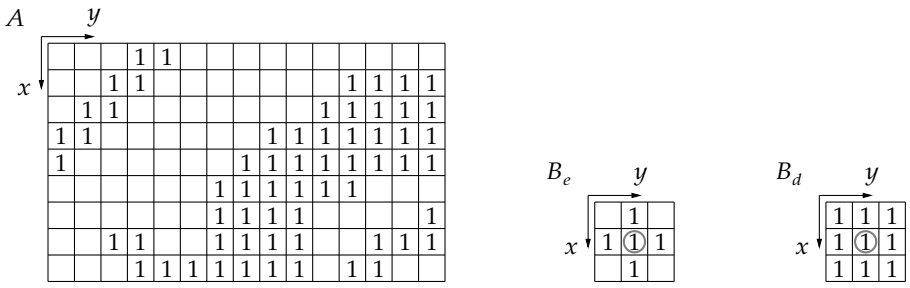




Calculate the reconstruction by erosion of X w.r.t. the mask M

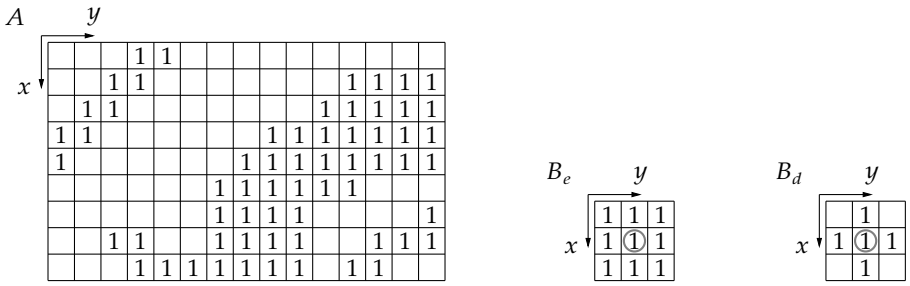
- using a 3×3 disc-shaped structuring element with radius 1,
- using a 3×3 square structuring.

Exercise 7.3.3.4-3: Consider the binary image A and two structuring elements B_e and B_d.



Calculate the opening by reconstruction with $n = 2$, i.e. $O_R^{(2)}(A) = R_{D,A}(A \ominus 2B_e)$ using the structuring elements B_e and B_d for erosion and geodesic dilation respectively.

Exercise 7.3.3.4-4: Consider the binary image A and two structuring elements B_e and B_d.



Calculate the closing by reconstruction with $n = 2$, i.e. $C_R^{(2)}(A) = R_{E,A}(A \oplus 2B_d)$ using the structuring elements B_d and B_e for dilation and geodesic erosion respectively.

Exercise 7.3.3.4-5: (*) Try repeating the previous exercises in MATLAB using the functions `strel`, `imdilate` and `imerode`.

Exercise 7.3.3.4-6: (*) Compose a procedure to label all connected regions in an image using geodesic dilation.

Exercise 7.3.3.4-7: (**) Compose a procedure to label all connected regions in an image without using the concept geodesic dilation. Compare your solution to the previous solution.

Exercise 7.3.3.4-8: (*) Compose a procedure to find a foreground-path between two arbitrary foreground pixels using 4-V-paths, 8-V-paths and m-V-paths.

Exercise 7.3.3.4-9: (*) Extend the procedure of the previous exercise to calculate the length of the path using 4-V-paths, 8-V-paths and m-V-paths.

7.3.4 Overview

Table 7.1 gives a convenient overview of the operations we've treated so far. The relationships between the most elementary operations have been indicated graphically in Figure 7.16. It may help you memorize these basic operations.

Name	Notation	Definition
Erosion	$A \ominus B$	$\{\vec{v} \mid B_{\vec{v}} \subseteq A\}$
Dilation	$A \oplus B$	$\left\{\vec{v} \mid B^r_{\vec{v}} \cap A \neq \emptyset\right\}$
Opening	$A \circ B$	$(A \ominus B) \oplus B$
Closing	$A \bullet B$	$(A \oplus B) \ominus B$
Hit-miss	$A \circledast (B_f, B_b)$	$\left(A \ominus B_f\right) \cap (A^c \ominus B_b)$
Thinning	$A \otimes (B_f, B_b)$	$A \setminus \left(A \circledast (B_f, B_b)\right)$
Thickening	$A \odot (B_f, B_b)$	$A \cup \left(A \circledast (B_f, B_b)\right)$
Geodesic erosion	$E_M(X)$	$(X \ominus B) \cup M$
Geodesic dilation	$D_M(X)$	$(X \oplus B) \cap M$
Reconstruction by erosion	$R_{E,M}(X)$	$E_M^{(k)}(X)$
Reconstruction by dilation	$R_{D,M}(X)$	$D_M^{(k)}(X)$
Opening by reconstruction	$O_{R,A}(A)$	$R_{D,A}(A \ominus nB)$
Closing by reconstruction	$C_{R,A}(A)$	$R_{E,A}(A \oplus nB)$

Table 7.1: Overview of the morphological operations on binary images

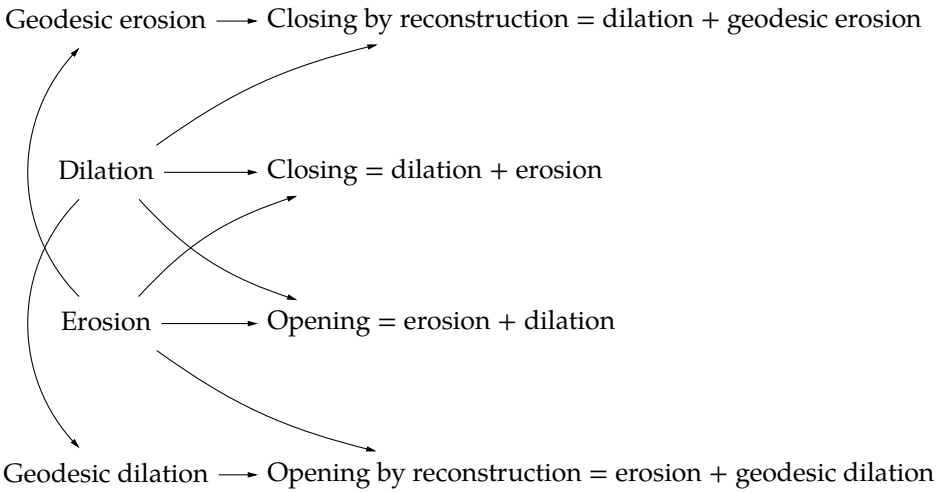


Figure 7.16: Relationships between the morphological operations on binary images

7.3.5 Applications

Now, the fun starts. Let's try to use the basic morphological operations to get some things done. We will treat

- edge detection,
- hole filling,
- border evacuation, and
- convex hull determination.

7.3.5.1 Edge detection

Getting to know the edge of an object is the basis for many image processing algorithm.

The basic idea of edge detection is that:

- the erosion removes a layer of the object,
- the dilation adds an extra layer to the object.

Therefore, the layer itself can be obtained using:

$$C = A \setminus (A \ominus B) \tag{7.7}$$

or

$$C = (A \oplus B) \setminus A$$

The former will result in a layer that is part of the foreground. The latter will yield a layer that is part of the background.

One might even consider using the following in order to obtain an edge that is balanced over the fore- and the background:

$$C = (A \oplus B) \setminus (A \ominus B)$$

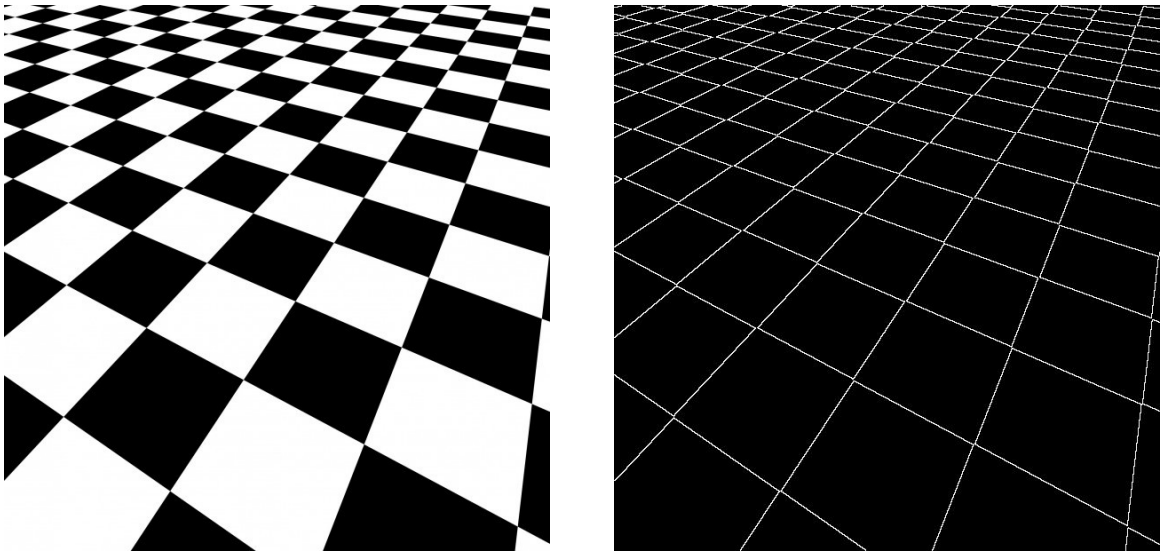
Of course, a sufficiently thick symmetrical structuring element B is appropriate. The size of B determines the size of the edge.

The principle has been illustrated in Figure 7.17.

7.3.5.2 Hole filling

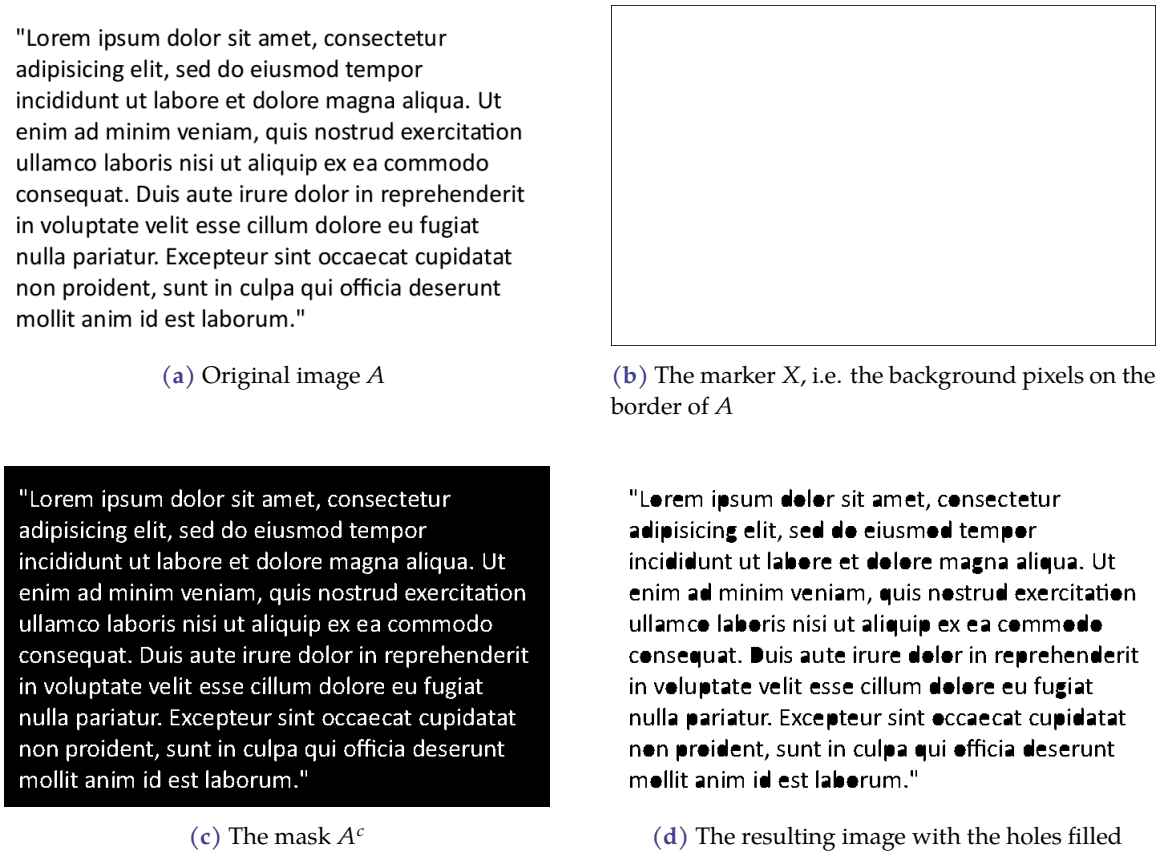
Very often after tresholding, holes are introduced in the foreground that do not correspond to physical holes in the objects the foreground represents. Therefore, we'd like to remove these hole artifacts.

Hole filling in a binary image is based on the fact that holes are disconnected parts of the background. We assume that the true background (not the holes) is the portion of the background that touches the border of the image.



(a) Original image (b) Resulting image

Figure 7.17: Edge detection using the principle of equation (7.7)



(a) Original image A (b) The marker X, i.e. the background pixels on the border of A

(c) The mask A^c (d) The resulting image with the holes filled

Figure 7.18: Hole filling according to the principle of equation (7.8); note: black = foreground, white = background

Therefore, let's rebuild the connected part of the background using geodesic dilation starting from the background pixels on the border as a marker and using the background as a mask.

In detail: let the marker X be:

$$X[x,y] = \begin{cases} 1 - A[x,y] & \text{if } [x,y] \text{ is on the border} \\ 0 & \text{otherwise} \end{cases}$$

then F is a reconstruction of A with all holes filled, when

$$F = \left(R_{D,A^c}(X)\right)^c \tag{7.8}$$

This has been illustrated in Figure 7.18.

7.3.5.3 Border evacuation

The goal is to remove objects that are on the border of an image. Objects that are on the border have only been images partially. Therefore analyzing their properties doesn't make any sense. We'd like to remove these border objects from the image before analyzing it further.

The idea is to reconstruct the foreground objects on the border using geodesic dilation (using the foreground as a mask), starting from the foreground pixels of the border as marker image. Afterwards we can subtract these reconstructed border objects from the full foreground.

In detail: Let the marker X be:

$$X[x,y] = \begin{cases} A[x,y] & \text{if } [x,y] \text{ on the border} \\ 0 & \text{otherwise} \end{cases}$$

then F is a reconstruction of A with an evacuated border:

$$F = A \setminus R_{D,A}(X) \tag{7.9}$$

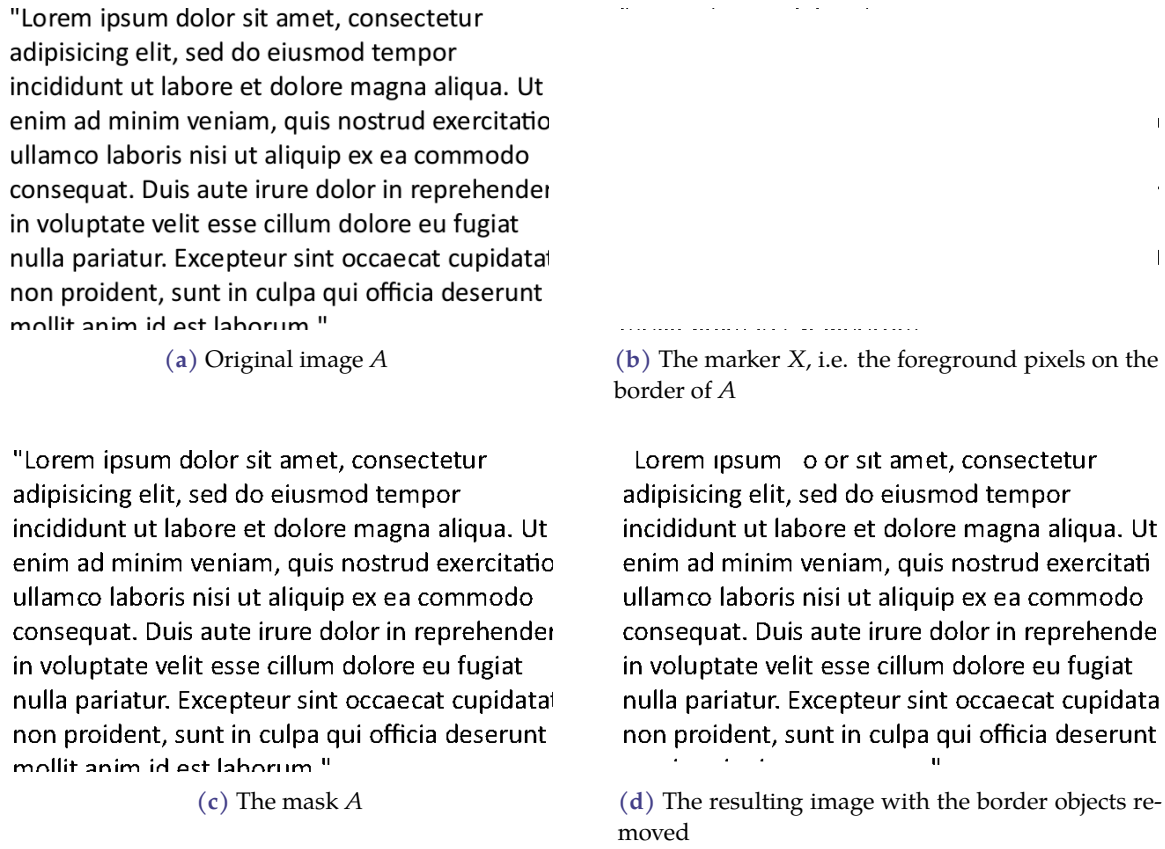
This has been illustrated in Figure 7.19.

7.3.5.4 Convex hull determination

In many cases processing a full image with complex imaging algorithms requires too much computing power. Determination of a region of interest is required to allow focusing the image analysis onto a specific part of the image. Usually the shape of the hull around the object of interest should not be overly complex. Demanding it to be rectangular or diamond-shaped might seem appropriate.

To this end, we're going to use the hit/miss transformation. Just apply the following algorithm using the structuring elements listed in Figure 7.20 for a rectangular-shaped convex hull and the ones in Figure 7.21 for a diamond-shaped convex hull.

The structuring elements for the rectangular-shaped case, find edges that are angled at odd multiples of 45° and flattens them out. The structuring elements for the diamond-shaped case, find edges that are angled at multiples of 90° and make them angled at odd multiples of 45°.



(a) Original image A

(b) The marker X, i.e. the foreground pixels on the border of A

(c) The mask A

(d) The resulting image with the border objects removed

Figure 7.19: Border evacuation according to the principle of equation (7.9); note: black = foreground, white = background

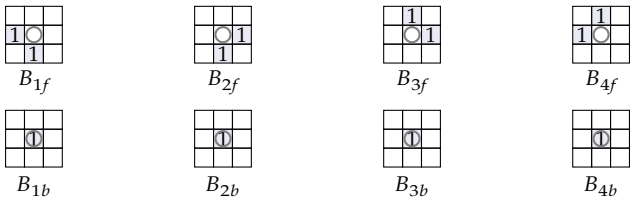


Figure 7.20: Structuring elements to be used for rectangle-shaped convex hull determination

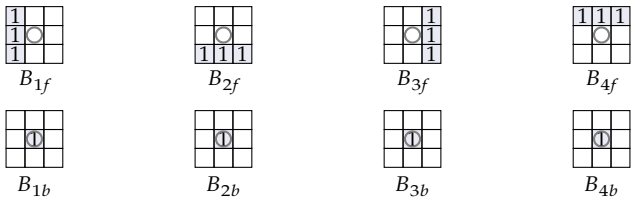
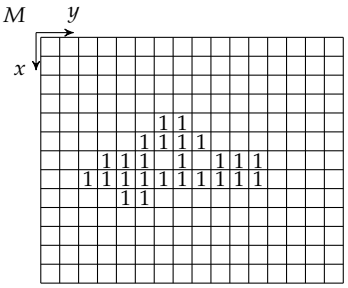
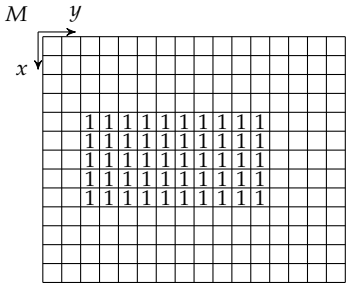


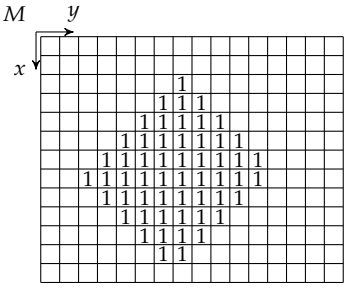
Figure 7.21: Structuring elements to be used for diamond-shaped convex hull determination



(a) Original image



(b) Rectangular convex hull



(c) Diamond shaped convex hull

Figure 7.22: Convex hull determination

Convex Hull Algorithm

1. $\forall i \in \{1,2,3,4\}$:
   1.1. $H_{i,0} = A$
   1.2. $\forall j \in \{1,2,\ldots k_i\}$:
   $$H_{i,j} = (H_{i,j-1} \circledast (B_{if}, B_{ib})) \cup A$$
   with k_i sufficiently high for convergence of $H_{i,j}$
2. $H(A) = \bigcup_{i=1}^{4} H_{i,k_i}$

The result of the algorithm has been illustrated in Figure 7.22. The intermediary steps can be found in Figure 7.23 for the rectangular case and in Figure 7.24 for the diamond-shaped case.

7.4 Morphology for grayscale images

In this section, we will discuss the morphology for grayscale images. For a good understanding, start studying the binary operations first. The treatment for grayscale images will be more terse. When reading this section after having studied the binary operations, you will no doubt experience a *déjà-vu* feeling more than once.

7.4.1 Basic operations

Let's again start with the 'addition and subtraction' of mathematical morphology, i.e. dilation and erosion.

7.4.1.1 Dilation

Use The dilation makes high intensity regions a bit wider and low intensity regions a bit smaller. This corresponds perfectly to the behavior of a binary dilation.

Definition

Dilation of grayscale images

The dilation of a grayscale image f with a structuring element b yields a new image g and is denoted as

$$g = f \oplus b$$

and defined as

$$(f \oplus b)[x,y] = \max_{[u,v] \in \mathrm{dom}\, b} \{f[x-u, y-v] + b[u,v]\}.$$

Frequently, we'll use a flat SE with intensity value 0, reducing the definition to

$$(f \oplus b)[x,y] = \max_{[u,v] \in \mathrm{dom}\, b} \{f[x-u, y-v]\}.$$

The operation has been illustrated in Figure 7.25.

Properties The dilation is:

- commutative
$$f \oplus g = g \oplus f$$
- associative
$$(f \oplus g) \oplus h = f \oplus (g \oplus h)$$
- distributive w.r.t. the pointwise union
$$f \oplus (g \vee h) = (f \oplus g) \vee (f \oplus h)$$
- monotonic
$$f < g \quad \Rightarrow \quad f \oplus h < g \oplus h$$

Border pixels While the basic definition of the *binary* dilation was robust w.r.t. border effects, the definition of the *grayscale* dilation runs into trouble. What about calculating the dilation for pixels on (or near) the border of the image? Some of the intensity values needed in the definition become undefined.

Commonly, for dilation, we will assume the intensity values of pixels beyond the border to be $-\infty$. In that way the end result is not influenced by them.

Decomposition of structuring elements The computational effort of the dilation-operation is proportional to the number of pixels in the SE. Associativity allows for the decomposition of structuring elements enabling sequential dilation.

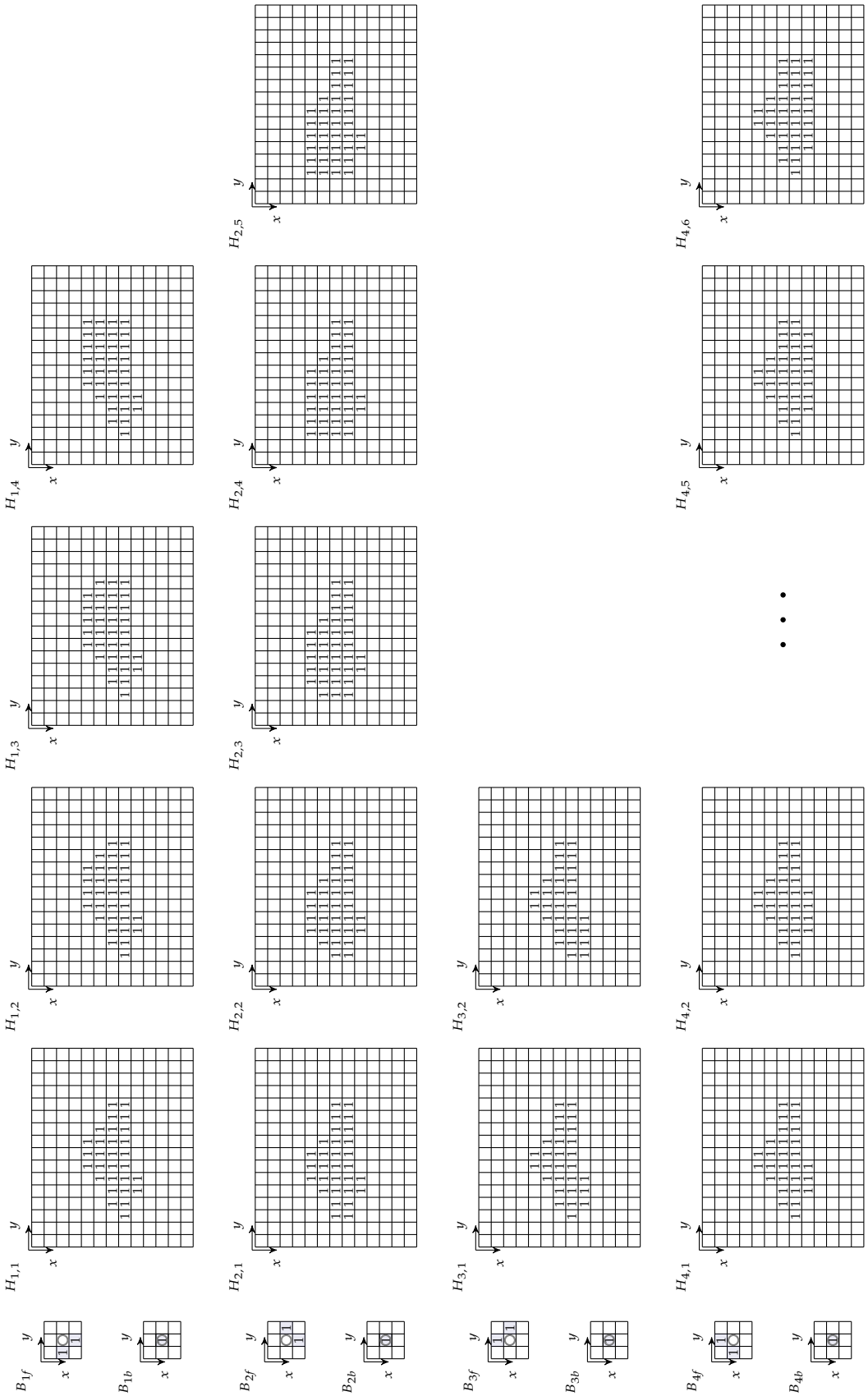


Figure 7.23: Illustration of the intermediary images obtained by the convex hull algorithm using the 'rectangular' structuring elements; every row contains the hit/miss extension of the original image using the structuring elements indicated in the first column.

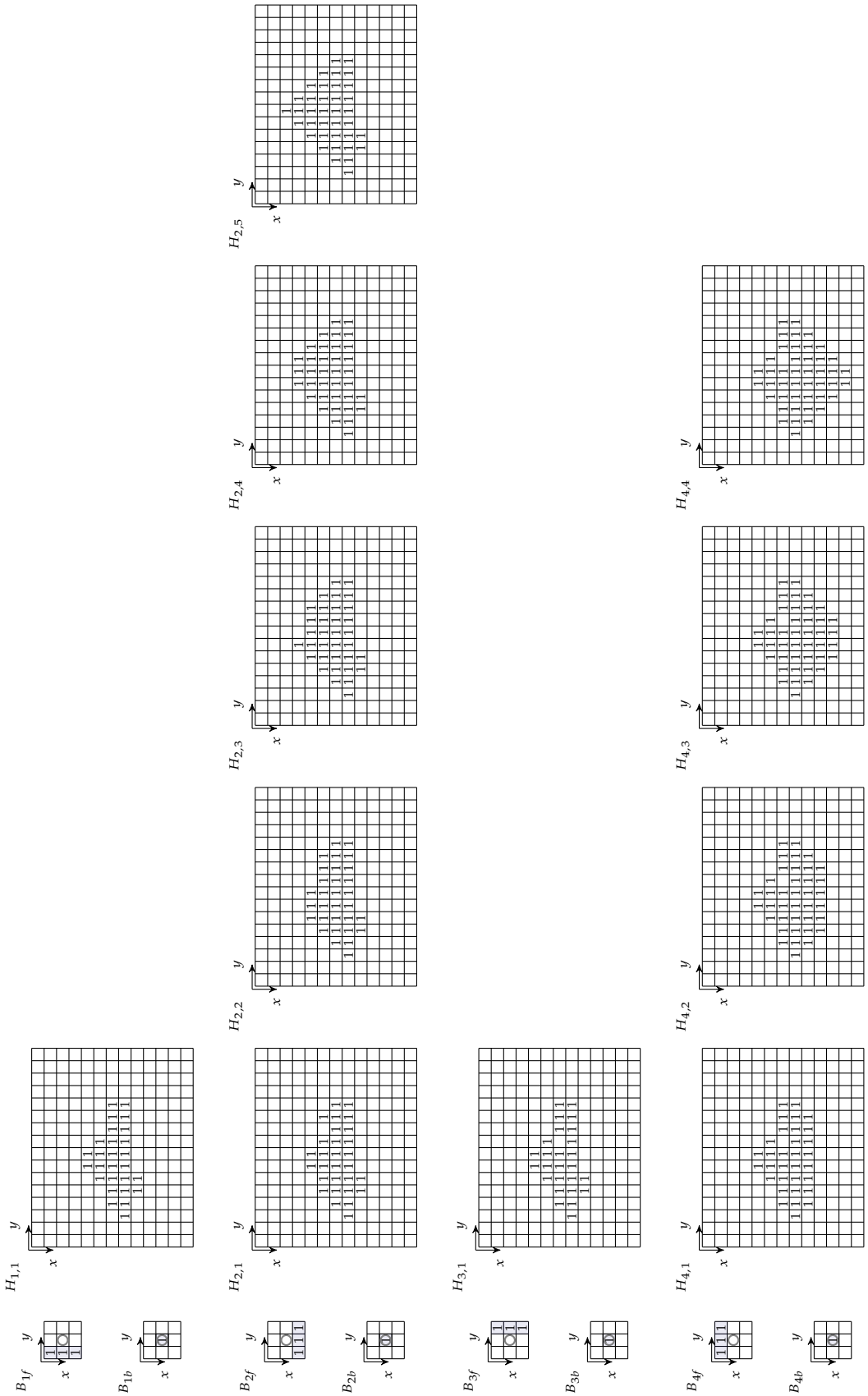
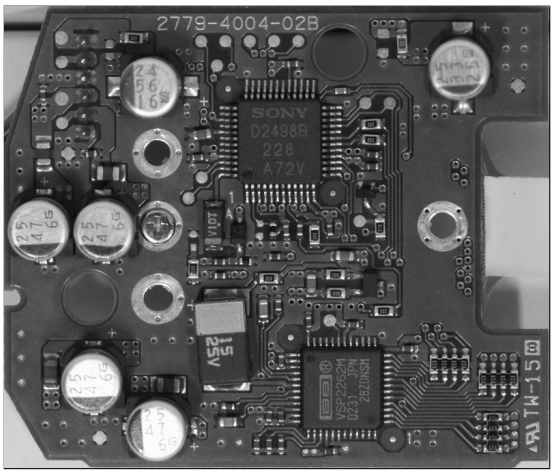


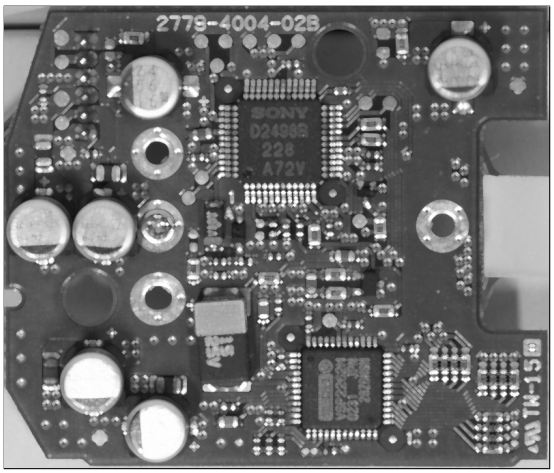
Figure 7.24: Illustration of the intermediary images obtained by the convex hull algorithm using the 'diamond' structuring elements; every row contains the hit/miss extension of the original image using the structuring elements indicated in the first column.

(a) Original image of Barbara (source: Allen Gersho)

(b) Resulting image



(c) Original image of a PCB (source: Wikimedia commons, user FuseOppl)



(d) Resulting image

Figure 7.25: Grayscale dilation of some images using a flat disc with a radius of 5 pixels

7.4.1.2 Erosion

Use The erosion makes high intensity regions a bit smaller and low intensity regions a bit wider. This corresponds perfectly to the behavior of a binary erosion.

Definition

Erosion of grayscale images

The erosion of a grayscale image f with a structuring element b yields a new image g and is denoted as

$$g = f \ominus b$$

and defined as

$$(f \ominus b)[x,y] = \min_{[u,v] \in \mathrm{dom}\, b} \{f[x+u, y+v] - b[u,v]\}$$

Frequently, we'll use a flat SE with intensity value 0, reducing the definition to

$$(f \ominus b)[x,y] = \min_{[u,v] \in \mathrm{dom}\, b} \{f[x+u, y+v]\}\,.$$

The operation has been illustrated in Figure 7.26.

Properties The erosion is:

- not commutative!
- not associative!
- distributive w.r.t. the pointwise intersection

$$(f \wedge g) \ominus (h \wedge i) = (f \ominus h) \wedge (f \ominus i) \wedge (g \ominus h) \wedge (g \ominus i)$$

- monotonic

$$f < g \quad \Rightarrow \quad f \ominus h \; < \; g \ominus h$$

Border pixels Again, while the basic definition of the *binary* erosion was robust w.r.t. border effects, the definition of the *grayscale* erosion runs into trouble for the same reasons as the dilation did.

Commonly, for erosion, we will assume the intensity values of pixels beyond the border to be $+\infty$. In that way the end result is not influenced by them.

7.4.1.3 Duality of erosion and dilation

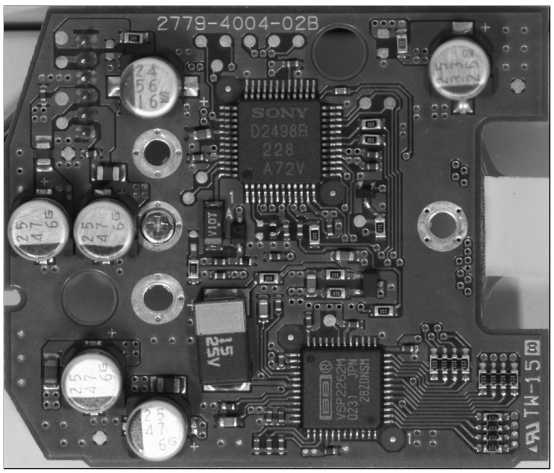
Duality of erosion and dilation

The erosion and dilation are dual operations, i.e.

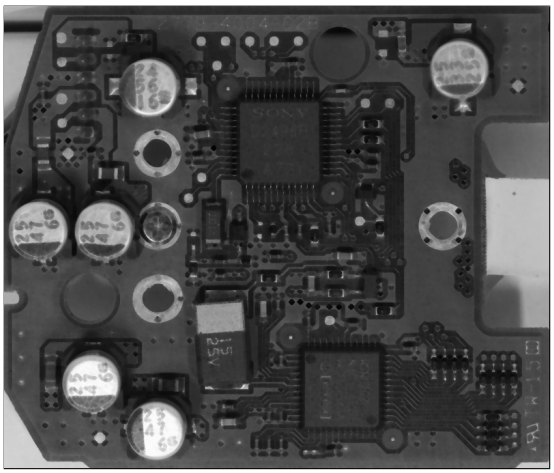
$$f \oplus g = (f^c \ominus g^r)^c \qquad f \ominus g = (f^c \oplus g^r)^c$$

(a) Original image of Barbara (source: Allen Gersho)

(b) Resulting image



(c) Original image of a PCB (source: Wikimedia commons, user FuseOppl)



(d) Resulting image

Figure 7.26: Grayscale erosion of some images using a flat disc with a radius of 5 pixels

In words: dilation of the foreground corresponds to erosion of the background with the reflected structuring element.

Proof
Starting from the dilation:

$$\begin{aligned}
(f \oplus g)[x,y] &= \left(\left((f \oplus g)[x,y]\right)^c\right)^c \\
&= (1 - (f \oplus g)[x,y])^c \\
&= \left(1 - \max_{[u,v] \in \mathrm{dom}\, g} \{f[x-u, y-v] + g[u,v]\}\right)^c \\
&= \left(\min_{[u,v] \in \mathrm{dom}\, g} \{1 - f[x-u, y-v] - g[u,v]\}\right)^c \\
&= \left(\min_{[u,v] \in \mathrm{dom}\, g} \{f^c[x-u, y-v] - g^r[-u,-v]\}\right)^c \\
&= \left(\min_{[-u,-v] \in \mathrm{dom}\, g^r} \{f^c[x-u, y-v] - g^r[-u,-v]\}\right)^c \\
&= (f^c \ominus g^r)^c\,[x,y]
\end{aligned}$$

Starting from the erosion:

$$\begin{aligned}
(f \ominus g)[x,y] &= \left(\left((f \ominus g)[x,y]\right)^c\right)^c \\
&= (1 - (f \ominus g)[x,y])^c \\
&= \left(1 - \min_{[u,v] \in \mathrm{dom}\, g} \{f[x+u, y+v] - g[u,v]\}\right)^c \\
&= \left(\max_{[u,v] \in \mathrm{dom}\, g} \{1 - f[x+u, y+v] + g[u,v]\}\right)^c \\
&= \left(\max_{[u,v] \in \mathrm{dom}\, g} \{f^c[x+u, y+v] + g^r[-u,-v]\}\right)^c \\
&= \left(\max_{[-u,-v] \in \mathrm{dom}\, g^r} \{f^c[x-(-u), y-(-v)] + g^r[-u,-v]\}\right)^c \\
&= (f^c \oplus g^r)^c\,[x,y]
\end{aligned}$$

■

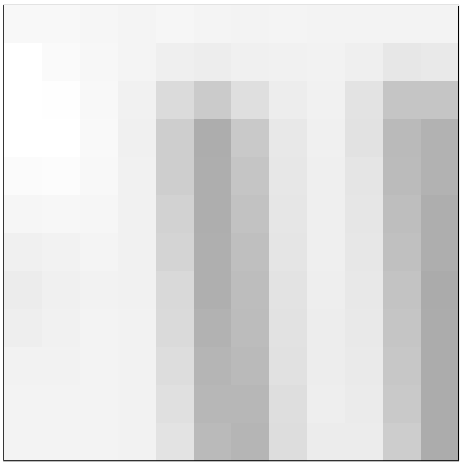
7.4.1.4 Calculations for flat structuring elements

Executing the definitions of grayscale dilation and erosion is not very difficult when restricting to flat structuring elements. Still it is worthwhile taking a good look into them, so that we can be sure you understand them in depth.

As an example, consider the image in Figure 7.27a. We indicated a small rectangular cutout on the image, that we will study in detail. The cutout is taken from the radiator grille of the toy car on the left. A magnified version of the cutout can be found in Figure 7.27b, the corresponding intensity values can be found in Figure 7.27c.

Dilation Consider the intensity matrix below on the left, together with the flat square 5×5 structuring element b that has been indicated using a gray overlay. Dilation means taking the

(a) Original image with cutout indicated by small square



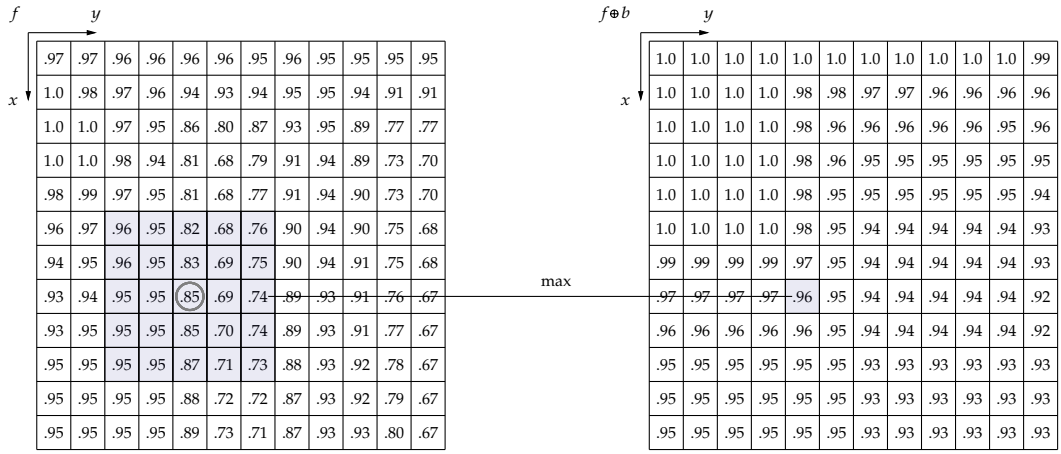
(b) Cutout magnified

x \ y												
	.97	.97	.96	.96	.96	.96	.95	.96	.95	.95	.95	.95
	1.0	.98	.97	.96	.94	.93	.94	.95	.95	.94	.91	.91
	1.0	1.0	.97	.95	.86	.80	.87	.93	.95	.89	.77	.77
	1.0	1.0	.98	.94	.81	.68	.79	.91	.94	.89	.73	.70
	.98	.99	.97	.95	.81	.68	.77	.91	.94	.90	.73	.70
	.96	.97	.96	.95	.82	.68	.76	.90	.94	.90	.75	.68
	.94	.95	.96	.95	.83	.69	.75	.90	.94	.91	.75	.68
	.93	.94	.95	.95	.85	.69	.74	.89	.93	.91	.76	.67
	.93	.95	.95	.95	.85	.70	.74	.89	.93	.91	.77	.67
	.95	.95	.95	.95	.87	.71	.73	.88	.93	.92	.78	.67
	.95	.95	.95	.95	.88	.72	.72	.87	.93	.92	.79	.67
	.95	.95	.95	.95	.89	.73	.71	.87	.93	.93	.80	.67

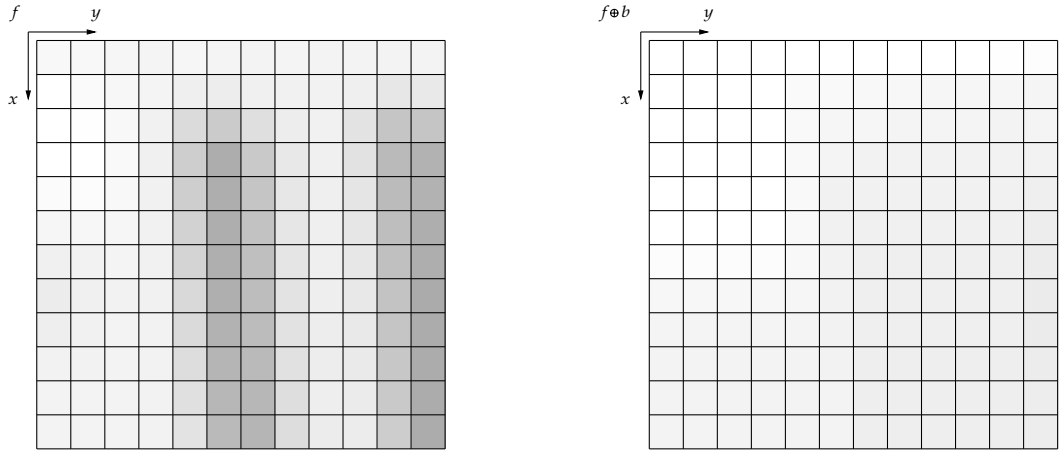
(c) Intensity values normalized to range 0 to 1

Figure 7.27: Test image from the University of Southern Carolina SIPI Image database

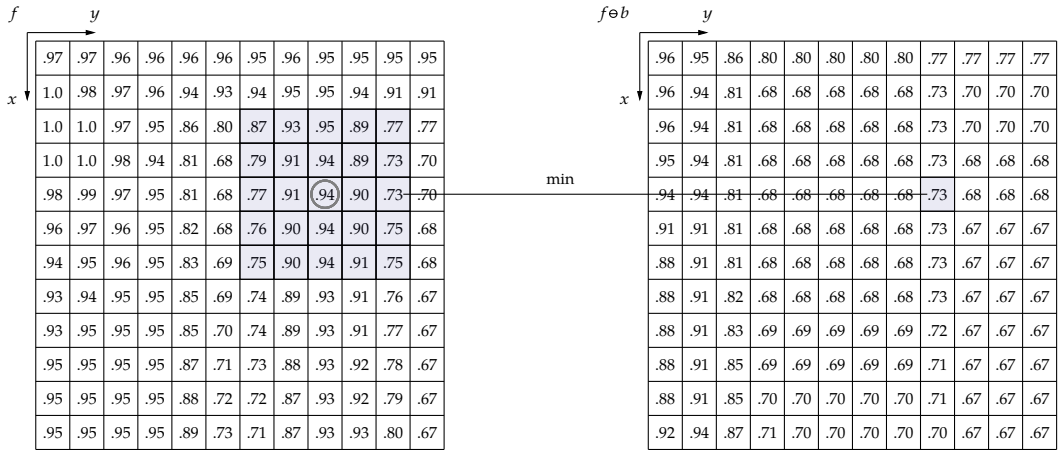
maximum value out of the indicated gray overlay, i.e. the central pixel gets the value of .96 as indicated on the right.



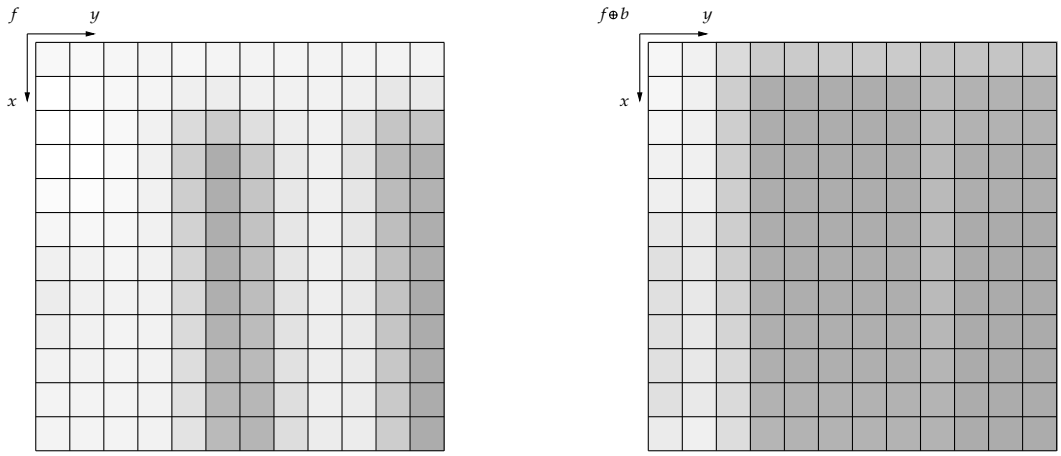
The result of the dilation operation can be seen in grayscale below:



Erosion Consider the intensity matrix below on the left, together with the flat square 5×5 structuring element b that has been indicated using a gray overlay. Erosion means taking the minimum value out of the indicated gray overlay, i.e. the central pixel gets the value of .73 as indicated on the right.



The result of the erosion operation can be seen in grayscale below:



The global effect of dilation and erosion on the test image can be seen in Figure 7.28.

Exercises

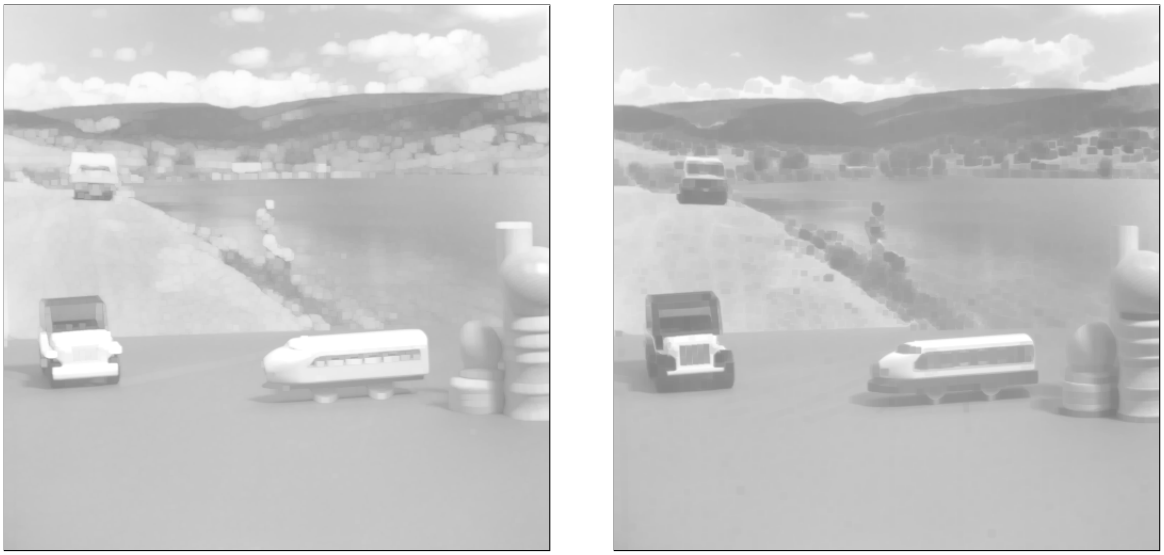
Exercise 7.4.1.4-1: Consider the intensity values of a 10 × 10 image f normalized to the range 0 to 1. Consider the flat structuring element b.

$f[x,y]$

	y									
x	0.5	0.7	0.4	0.1	0.7	0.0	0.1	0.5	0.9	0.9
	0.6	0.3	0.5	0.4	0.5	0.4	0.6	0.6	0.7	0.4
	0.8	0.5	0.4	0.6	0.2	0.8	0.0	0.8	0.5	0.1
	0.5	0.6	0.8	0.5	0.6	0.6	0.8	0.1	0.8	0.4
	0.2	0.2	0.7	0.1	0.1	0.5	0.8	0.7	0.5	0.3
	0.5	0.6	0.4	0.2	0.7	0.9	0.9	0.5	0.6	0.4
	0.4	0.7	0.7	0.8	0.6	0.1	1.0	0.2	0.7	0.8
	1.0	0.4	0.9	0.0	0.1	0.9	0.5	0.9	0.4	0.4
	0.6	0.8	0.8	0.9	0.1	0.1	0.3	0.6	0.2	0.4
	0.7	0.7	0.7	0.1	0.2	0.5	0.1	0.4	0.6	0.4

$g[x,y]$

	y			
x	0	0	0	0
	0	0	0	0
	0	0	0	0
	0	0	0	0



(a) Resulting image after dilation **(b)** Resulting image after erosion

Figure 7.28: Grayscale dilation and erosion applied to the test image of Figure 7.27a using a square 5×5 flat structuring element

Calculate:

- $f \oplus b$
- $f \ominus b$

Exercise 7.4.1.4-2: (*) Consider the dilated result of the previous exercise and erode that using the same structuring element g. Do you regain the original?
In symbols:

$$(f \oplus g) \ominus g \stackrel{?}{=} f$$

Likewise, consider the eroded result of the previous exercise and dilate that using the same structuring element g. Do you regain the original?
In symbols:

$$(f \ominus g) \oplus g \stackrel{?}{=} f$$

Exercise 7.4.1.4-3: Consider the intensity values of a 10×10 image r normalized to the range 0 to 1. Consider the flat structuring element s.

$r[x,y]$

x \ y	0	1	2	3	4	5	6	7	8	9
0	0.1	0.2	0.2	0.1	0.7	0.3	0.5	0.3	0.2	0.8
1	0.3	0.8	0.1	0.4	0.6	0.6	0.7	0.2	0.0	0.6
2	0.1	0.1	0.6	0.2	0.6	0.9	0.2	0.2	0.3	0.7
3	0.4	0.4	0.9	0.0	0.2	0.9	0.3	0.2	0.7	0.6
4	0.3	0.0	0.9	1.0	0.8	0.6	0.2	1.0	0.6	0.1
5	0.3	0.2	0.2	0.4	0.2	0.3	0.3	0.9	0.5	0.5
6	0.4	0.0	0.5	1.0	0.4	0.9	0.4	0.8	0.4	0.3
7	0.1	0.2	0.4	0.8	0.9	0.4	0.5	0.7	0.3	0.1
8	0.5	0.1	0.5	0.0	0.9	0.9	0.0	0.2	0.5	0.1
9	0.7	0.3	0.3	0.7	0.4	0.0	0.6	0.4	0.8	0.2

$s[x,y]$

x \ y	0	1	2
0	0	0	0
1	0	(0)	
2		0	0

Calculate:

- $r \oplus s$
- $r \ominus s$

Exercise 7.4.1.4-4: Read the documentation of the MATLAB function `strel`. Learn how to use it to compose structuring elements for grayscale images.
Read the documentation of the MATLAB functions `imdilate` and `imerode`. Learn how to use them.
Redo the previous exercises using these MATLAB functions.

7.4.2 Derived operations

7.4.2.1 Opening

Use The opening

- removes hairline protuberances, and
- grows hairline cracks and trenches.

Definition

Opening of grayscale images The opening of a grayscale image f with a structuring element b yields a new image g and is denoted as

$$g = f \circ b$$

and defined as

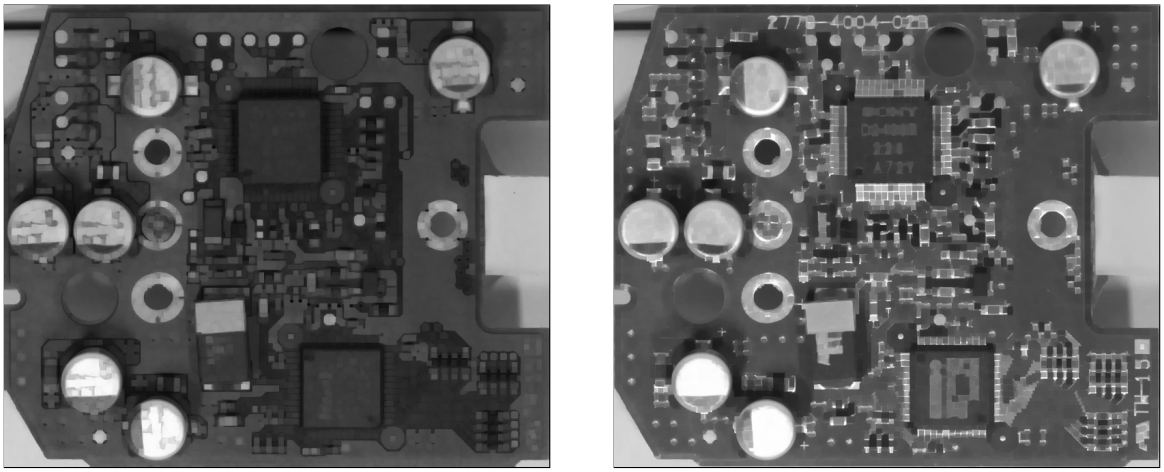
$$f \circ b = (f \ominus b) \oplus b.$$

This has been illustrated using a picture of a PCB in Figure 7.29a. Note how the solder work including the integrated circuit's leads has been totally removed by the opening.

7.4.2.2 Closing

Use The closing

- removes hairline cracks and trenches, and
- grows hairline protuberances.



(a) Result after opening (b) Result after closing

Figure 7.29: Grayscale opening and closing of the image of Figure 7.25c using a flat disc structuring element with radius 15

Definition

Closing of grayscale images

The closing of a grayscale image f with a structuring element b yields a new image g and is denoted as

$$g = f \bullet b$$

and defined as

$$f \bullet b = (f \oplus b) \ominus b.$$

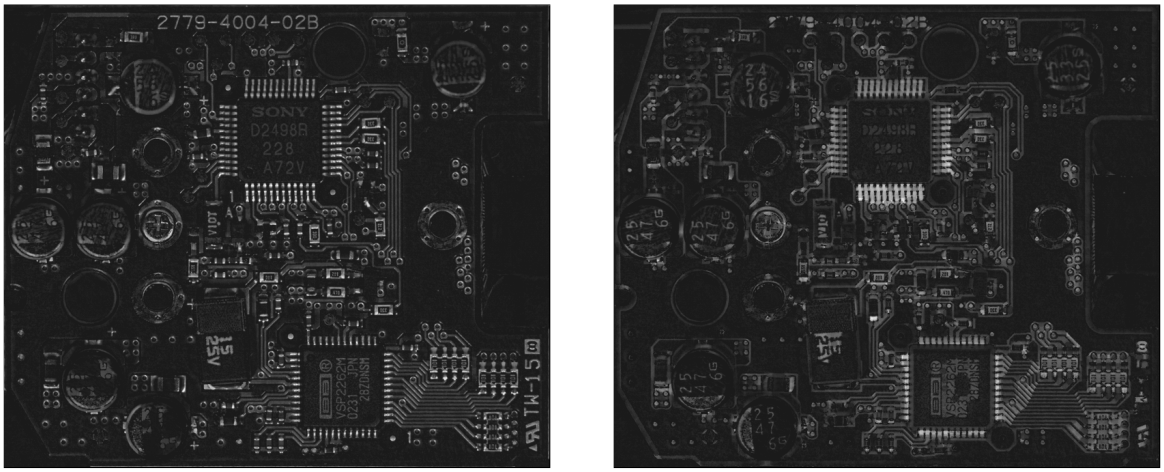
This has been illustrated using a picture of a PCB in Figure 7.29b. Note how the isolation between the tracks and between the integrated circuit's leads has been totally removed by the closing.

7.4.2.3 Top hat transformation

Use The top hat transformation is used to

- select white objects, and
- correct (equalize) lighting in dark parts.

The top hat transformation was not explicitly defined for binary images because it has no good use for binary images.



(a) Result after top hat transformation

(b) Result after bottom hat transformation

Figure 7.30: Grayscale top hat and bottom hat transformation of the image of Figure 7.25c using a flat disc structuring element with radius 5

Definition

Top hat transformation of grayscale images

The top hat transformation of a grayscale image f using a structuring element b yields a new image g and is denoted as

$$g = \hat{T}_b(f)$$

and is defined as

$$\hat{T}_b(f) = f - (f \circ b).$$

Opening removes the local intensity peaks. Therefore, subtracting the result of the opening from the original image yields the intensity peaks only.

This has been illustrated using a picture of a PCB in Figure 7.30a.

7.4.2.4 Bottom hat transformation

Use The bottom hat transformation is used to

- select black objects, and
- correct (equalize) lighting in light parts.

The bottom hat transformation was not explicitly defined for binary images because it has no good use for binary images.

Definition

Bottom hat transformation of grayscale images

The bottom hat transformation of a grayscale image f using a structuring element b yields a new image g and is denoted as

$$g = \hat{B}_b(f)$$

and is defined as

$$\hat{B}_b(f) = (f \bullet b) - f.$$

Closing removes the local intensity valleys. Therefore, subtracting the original image from this result yields the intensity valleys only.

This has been illustrated using a picture of a PCB in Figure 7.30b.

Exercises

Exercise 7.4.2.4-1: Consider the image intensity function $f[x, y]$ below together with the flat square 3×3 structuring element $g[x, y]$.

$f[x,y]$

x \ y	0	1	2	3	4	5	6	7	8	9
0	1.0	0.9	0.8	0.3	0.1	0.3	0.4	0.5	0.3	0.1
1	0.9	0.9	0.8	0.0	0.4	1.0	0.9	1.0	0.8	0.2
2	0.8	0.7	0.5	0.1	0.5	0.8	0.9	1.0	0.8	0.2
3	0.7	0.5	0.3	0.0	0.4	1.0	0.8	0.7	0.5	0.0
4	0.5	0.1	0.0	0.1	0.3	0.9	0.7	0.1	0.0	0.5
5	0.5	0.1	0.0	0.2	0.1	0.8	0.6	0.0	0.1	0.7
6	0.3	0.0	0.1	0.0	0.1	0.9	0.8	0.2	0.1	0.3
7	0.0	0.1	0.0	0.1	0.2	0.9	1.0	0.6	0.1	0.2
8	0.3	0.2	0.3	0.0	0.3	0.9	1.0	0.9	0.4	0.3
9	0.1	0.1	0.0	0.5	0.8	0.9	1.0	0.9	0.8	0.1

$g[x,y]$

x \ y	0	1	2
0		0	
1	0	(0)	0
2		0	

Calculate

- $f \circ g$
- $f \bullet g$
- $\hat{T}_g(f)$
- $\hat{B}_g(f)$

Exercise 7.4.2.4-2: Consider the image intensity function $f[x, y]$ below together with the flat square 3×3 structuring element $g[x, y]$.

$f[x,y]$ (x ↓, y →)

0.2	1.0	0.2	0.1	0.2	1.0	0.0	0.1	0.0	0.1
0.0	0.2	1.0	0.2	1.0	0.1	0.0	0.1	0.0	0.0
0.1	0.0	0.2	1.0	1.0	0.2	0.0	0.0	0.1	0.0
0.1	0.1	1.0	0.1	0.2	1.0	0.2	0.0	0.2	0.1
0.0	0.1	1.0	0.1	0.1	0.2	1.0	0.2	0.0	0.1
0.1	1.0	0.1	0.0	0.0	0.0	0.2	1.0	0.2	1.0
0.2	1.0	0.0	0.1	0.1	0.0	0.2	1.0	1.0	0.2
1.0	0.2	0.1	0.2	0.0	0.2	1.0	0.2	1.0	0.2
0.1	0.1	0.0	0.2	0.8	1.0	0.1	0.0	0.2	1.0
0.0	0.1	0.2	1.0	0.2	0.1	0.0	0.1	0.0	0.1

$g[x,y]$ (x ↓, y →)

0	0	0
0	0	0
0	0	0

Calculate

- $f \circ g$
- $f \bullet g$
- $\hat{T}_g(f)$
- $\hat{B}_g(f)$

Exercise 7.4.2.4-3: Read the documentation of the MATLAB function `strel`. Learn how to use it to compose structuring elements for grayscale images.
Read the documentation of the MATLAB functions `imopen`, `imclose`, `imtophat`, `imbothat` and `imreconstruct`. Learn how to use them.
Redo the previous exercises using these MATLAB functions.

7.4.3 Geodesic operations

For a rationale on why to use geodesic operations, see section 7.3.3 on page 165.

7.4.3.1 Geodesic dilation

Use Geodesic dilation will allow dilating an object without growing beyond the borders imposed by a chosen mask. Its true value will become apparent when going one step further, i.e. performing *iterative* geodesic dilation.

Definition

Geodesic dilation of grayscale images

Geodesic dilation of an image x using a structuring element b, with respect to a masking image m yields a new image y and is denoted as

$$y = D_m(x)$$

and is defined as

$$D_m(x) = (x \oplus b) \wedge m.$$

Note the absence of the structuring element b in the basic notation $D_m(x)$. This stresses the smaller importance of the structuring element. Very often a $N \times N$ square or disc-shaped SE is used.

Iterative geodesic dilation Often the operation is used iteratively. Therefore we agree on the notation:

$$D_m^{(n)}(x) = D_m\left(D_m^{(n-1)}(x)\right)$$

with

$$D_m^{(1)}(x) = D_m(x).$$

The iterative application of the geodesic dilation converges to a stable image. Applying it until convergence occurs is the so-called *reconstruction by dilation* and is denoted as:

$$R_{D,m}(x) = D_m^{(k)}(x)$$

with k such that $D_m^{(k+1)}(x) = D_m^{(k)}(x)$.

7.4.3.2 Geodesic erosion

The dual of geodesic dilation is geodesic erosion.

Use Geodesic erosion will allow eroding an object without shrinking below the borders imposed by a chosen mask. Again, this operation is only really useful when performed iteratively. In addition, most often it is seen as a dilation of the background with the reflected structuring element.

Definition

Geodesic erosion of grayscale images

Geodesic erosion of an image x using a structuring element b, with respect to a masking image m yields a new image y and is denoted as

$$y = E_m(x)$$

and is defined as

$$E_m(x) = (x \ominus b) \vee m.$$

Note the absence of the structuring element b in the basic notation $E_m(x)$. This stresses the smaller importance of the structuring element. Very often a $N \times N$ square or disc-shaped SE is used.

The simplest way to understand geodesic erosion is through duality, i.e. erosion corresponds to the dilation of the complement of the image with a reflected structuring element.

Iterative geodesic erosion Often the operation is used iteratively. Therefore we agree on the notation:

$$E_m^{(n)}(x) = E_m\left(E_m^{(n-1)}(x)\right)$$

with

$$E_m^{(1)}(x) = E_m(x).$$

The iterative application of the geodesic erosion converges to a stable image.[5] Applying it until convergence occurs is the so-called *reconstruction by erosion* and is denoted as:

$$R_{E,m}(x) = E_m^{(k)}(x)$$

with k such that $E_m^{(k+1)}(x) = E_m^{(k)}(x)$.

7.4.3.3 Opening by reconstruction

Use This corresponds to a classical opening, but with the advantage of removing the influence of the structuring element. Therefore, it can be used to replace the opening in other algorithms or operations, e.g. in the top hat transformation.

Definition

Opening by reconstruction of grayscale images

Opening by reconstruction of an image f using an erosion structuring element b_e and a geodesic dilation structuring element b_d yields a new image g and is denoted as

$$g = O_R^{(n)}(f)$$

and is defined as

$$O_R^{(n)}(f) = R_{D,f}(f \ominus nb_e)$$

in which $f \ominus nb_e$ means eroding the image f multiple (n) times by a specific b_e. Usually, a different structuring element b_d is used for the geodesic dilation.

7.4.3.4 Closing by reconstruction

Use This corresponds to a classical closing, but with the advantage of removing the influence of the structuring element. Therefore, it can be used to replace the closing in other algorithms or operations, e.g. in the bottom hat transformation.

[5] In general, this statement is incorrect. Limit cycles may appear if the center pixel is no part of the structuring element!

Definition

Closing by reconstruction of grayscale images

Closing by reconstruction of an image f using a dilation structuring element b_d and a geodesic erosion structuring element b_e yields a new image g and is denoted as

$$g = C_R^{(n)}(f)$$

and is defined as

$$C_R^{(n)}(f) = R_{E,f}(f \oplus nb_d)$$

in which $f \oplus nb_d$ means dilating the image f multiple (n) times by a specific b_d. Usually, a different structuring element b_e is used for the geodesic erosion.

7.4.3.5 Top hat and bottom hat by reconstruction

Replacing the opening and closing in the top and bottom hat transformation, yields two new operations:

Top hat by reconstruction

$$\begin{aligned}\hat{T}_R^{(n)}(f) &= f - O_{R,f}^{(n)}(f) \\ &= f - R_{D,f}(f \ominus nb) \\ &= f - D_f^{(k)}(f \ominus nb)\end{aligned}$$

with k sufficiently high to cause convergence to a stable image.

Bottom hat by reconstruction

$$\begin{aligned}\hat{B}_R^{(n)}(f) &= C_{R,f}^{(n)}(f) - f \\ &= R_{E,f}(f \oplus nb) - f \\ &= E_f^{(k)}(f \oplus nb) - f\end{aligned}$$

with k sufficiently high to cause convergence to a stable image.

Exercises

Exercise 7.4.3.5-1: Consider the image $f[x, y]$ below together with the flat square structuring element $g[x, y]$ and a marker $h[x, y]$.

$f[x,y]$ (x ↓, y →)

0.0	0.1	0.3	0.1	0.0	0.0	0.0	0.0	0.0	0.0
0.3	0.7	0.9	0.7	0.2	0.0	0.0	0.1	0.0	0.3
0.2	0.9	1.0	0.9	0.0	0.0	0.2	0.3	0.5	1.0
0.1	0.7	0.8	0.8	0.3	0.0	0.7	0.6	0.5	0.9
0.0	0.0	0.1	0.2	0.0	0.0	0.8	0.5	0.1	0.4
0.1	0.0	0.0	0.0	0.2	0.3	0.7	0.1	0.0	0.1
0.0	0.3	0.2	0.1	0.0	0.2	0.8	0.3	0.1	0.3
0.6	1.0	0.7	0.4	0.2	0.1	0.9	0.7	0.5	0.8
0.9	1.0	0.9	0.7	0.4	0.0	0.5	0.9	1.0	0.7
1.0	1.0	0.9	0.8	0.3	0.0	0.0	0.1	0.2	0.3

$g[x,y]$ (x ↓, y →)

0	0	0
0	(0)	0
0	0	0

$h[x,y]$ (x ↓, y →)

0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	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	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Calculate the reconstruction by dilation of marker h using the structuring element g w.r.t. mask f, i.e. calculate $R_{D,f}(h)$.

Exercise 7.4.3.5-2: Consider the image $f[x,y]$ below together with the flat square structuring element $g[x,y]$ and a marker $h[x,y]$.

$f[x,y]$ (x ↓, y →)

0.0	0.1	0.3	0.1	0.0	0.0	0.0	0.0	0.0	0.0
0.3	0.7	0.9	0.7	0.2	0.0	0.0	0.1	0.0	0.3
0.2	0.9	1.0	0.9	0.0	0.0	0.2	0.3	0.5	1.0
0.1	0.7	0.8	0.8	0.3	0.0	0.7	0.6	0.5	0.9
0.0	0.0	0.1	0.2	0.0	0.0	0.8	0.5	0.1	0.4
0.1	0.0	0.0	0.0	0.2	0.3	0.7	0.1	0.0	0.1
0.0	0.3	0.2	0.1	0.0	0.2	0.8	0.3	0.1	0.3
0.6	1.0	0.7	0.4	0.2	0.1	0.9	0.7	0.5	0.8
0.9	1.0	0.9	0.7	0.4	0.0	0.5	0.9	1.0	0.7
1.0	1.0	0.9	0.8	0.3	0.0	0.0	0.1	0.2	0.3

$g[x,y]$ (x ↓, y →)

0	0	0
0	(0)	0
0	0	0

$h[x,y]$ (x ↓, y →)

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	1.0	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	1.0	1.0	1.0	1.0	0.0	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	1.0	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

Calculate the reconstruction by erosion of marker h using the structuring element g w.r.t. mask f, i.e. calculate $R_{E,f}(h)$.

Exercise 7.4.3.5-3: Consider the image $f[x,y]$ below together with the flat square structuring elements $g_e[x,y]$, $g_d[x,y]$

$f[x,y]$ (x ↓, y →)

0.0	0.1	0.3	0.1	0.0	0.0	0.0	0.0	0.0	0.0
0.3	0.7	0.9	0.7	0.2	0.0	0.0	0.1	0.0	0.3
0.2	0.9	1.0	0.9	0.0	0.0	0.2	0.3	0.5	1.0
0.1	0.7	0.8	0.8	0.3	0.0	0.7	0.6	0.5	0.9
0.0	0.0	0.1	0.2	0.0	0.0	0.8	0.5	0.1	0.4
0.1	0.0	0.0	0.0	0.2	0.3	0.7	0.1	0.0	0.1
0.0	0.3	0.2	0.1	0.0	0.2	0.8	0.3	0.1	0.3
0.6	1.0	0.7	0.4	0.2	0.1	0.9	0.7	0.5	0.8
0.9	1.0	0.9	0.7	0.4	0.0	0.5	0.9	1.0	0.7
1.0	1.0	0.9	0.8	0.3	0.0	0.0	0.1	0.2	0.3

$g_e[x,y]$ (x ↓, y →)

	0	
0	(0)	0
	0	

$g_d[x,y]$ (x ↓, y →)

1	1	1
1	(1)	1
1	1	1

Calculate $O_R^{(2)}(f)$ using g_e and g_d as erosion and geodesic dilation structuring element respectively.

Exercise 7.4.3.5-4: Consider the image $f[x,y]$ below together with the flat square structuring elements $g_d[x,y]$, $g_e[x,y]$

$f[x,y]$ (x ↓, y →)

0.0	0.1	0.3	0.1	0.0	0.0	0.0	0.0	0.0	0.0
0.3	0.7	0.9	0.7	0.2	0.0	0.0	0.1	0.0	0.3
0.2	0.9	1.0	0.9	0.0	0.0	0.2	0.3	0.5	1.0
0.1	0.7	0.8	0.8	0.3	0.0	0.7	0.6	0.5	0.9
0.0	0.0	0.1	0.2	0.0	0.0	0.8	0.5	0.1	0.4
0.1	0.0	0.0	0.0	0.2	0.3	0.7	0.1	0.0	0.1
0.0	0.3	0.2	0.1	0.0	0.2	0.8	0.3	0.1	0.3
0.6	1.0	0.7	0.4	0.2	0.1	0.9	0.7	0.5	0.8
0.9	1.0	0.9	0.7	0.4	0.0	0.5	0.9	1.0	0.7
1.0	1.0	0.9	0.8	0.3	0.0	0.0	0.1	0.2	0.3

$g_d[x,y]$ (x ↓, y →)

	0	
0	(0)	0
	0	

$g_e[x,y]$ (x ↓, y →)

1	1	1
1	(1)	1
1	1	1

Calculate $C_R^{(2)}(f)$ using g_d and g_e as dilation and geodesic erosion structuring element respectively.

Exercise 7.4.3.5-5: Consider the image $f[x,y]$ below together with the flat square structuring elements $g_e[x,y]$, $g_d[x,y]$

$f[x,y]$ (x ↓, y →)

0.0	0.1	0.3	0.1	0.0	0.0	0.0	0.0	0.0	0.0
0.3	0.7	0.9	0.7	0.2	0.0	0.0	0.1	0.0	0.3
0.2	0.9	1.0	0.9	0.0	0.0	0.2	0.3	0.5	1.0
0.1	0.7	0.8	0.8	0.3	0.0	0.7	0.6	0.5	0.9
0.0	0.0	0.1	0.2	0.0	0.0	0.8	0.5	0.1	0.4
0.1	0.0	0.0	0.0	0.2	0.3	0.7	0.1	0.0	0.1
0.0	0.3	0.2	0.1	0.0	0.2	0.8	0.3	0.1	0.3
0.6	1.0	0.7	0.4	0.2	0.1	0.9	0.7	0.5	0.8
0.9	1.0	0.9	0.7	0.4	0.0	0.5	0.9	1.0	0.7
1.0	1.0	0.9	0.8	0.3	0.0	0.0	0.1	0.2	0.3

$g_e[x,y]$ (x ↓, y →)

	0	
0	(0)	0
	0	

$g_d[x,y]$ (x ↓, y →)

1	1	1
1	(1)	1
1	1	1

Calculate $\hat{T}_R^{(2)}(f)$ using g_e and g_d as erosion and geodesic dilation structuring element respectively.

Exercise 7.4.3.5-6: Consider the image $f[x,y]$ below together with the flat square structuring elements $g_d[x,y]$, $g_e[x,y]$

$f[x,y]$ (x ↓, y →)

0.0	0.1	0.3	0.1	0.0	0.0	0.0	0.0	0.0	0.0
0.3	0.7	0.9	0.7	0.2	0.0	0.0	0.1	0.0	0.3
0.2	0.9	1.0	0.9	0.0	0.0	0.2	0.3	0.5	1.0
0.1	0.7	0.8	0.8	0.3	0.0	0.7	0.6	0.5	0.9
0.0	0.0	0.1	0.2	0.0	0.0	0.8	0.5	0.1	0.4
0.1	0.0	0.0	0.0	0.2	0.3	0.7	0.1	0.0	0.1
0.0	0.3	0.2	0.1	0.0	0.2	0.8	0.3	0.1	0.3
0.6	1.0	0.7	0.4	0.2	0.1	0.9	0.7	0.5	0.8
0.9	1.0	0.9	0.7	0.4	0.0	0.5	0.9	1.0	0.7
1.0	1.0	0.9	0.8	0.3	0.0	0.0	0.1	0.2	0.3

$g_d[x,y]$ (x ↓, y →)

	0	
0	(0)	0
	0	

$g_e[x,y]$ (x ↓, y →)

1	1	1
1	(1)	1
1	1	1

Calculate $\hat{B}_R^{(2)}(f)$ using g_d and g_e as dilation and geodesic erosion structuring element respectively.

Exercise 7.4.3.5-7: Read the documentation of the MATLAB function `strel`. Learn how to use it to compose structuring elements for grayscale images.
Read the documentation of the MATLAB functions `imopen`, `imclose`, `imtophat`, `imbothat` and `imreconstruct`. Learn how to use them.
Redo the previous exercises using these MATLAB functions.

7.4.4 Overview

Table 7.2 gives a convenient overview of the operations we've treated so far. The relationships between the most elementary operations have been indicated graphically in Figure 7.31. It may help you memorize these basic operations.

Name	Notation	Definition
Erosion	$f \ominus g$	$\min_{[u,v] \in \mathrm{dom}\, g} \{f[x+u, y+v] - g[u,v]\}$
Dilation	$f \oplus g$	$\max_{[u,v] \in \mathrm{dom}\, g} \{f[x-u, y-v] + g[u,v]\}$
Opening	$f \circ g$	$(f \ominus g) \oplus g$
Closing	$f \bullet g$	$(f \oplus g) \ominus g$
Pointwise intersection	$(f \wedge g)(\vec{p})$	$\min(f(\vec{p}), g(\vec{p}))$
Pointwise union	$(f \vee g)(\vec{p})$	$\max(f(\vec{p}), g(\vec{p}))$
Geodesic erosion	$E_m(x)$	$(x \ominus b) \vee m$
Geodesic dilation	$D_m(x)$	$(x \oplus b) \wedge m$
Reconstruction by erosion	$R_{E,m}(x)$	$E_m^{(k)}(x)$
Reconstruction by dilation	$R_{D,m}(x)$	$D_m^{(k)}(x)$
Opening by reconstruction	$O_{R,f}(f)$	$R_{D,f}(f \ominus nb)$
Closing by reconstruction	$C_{R,f}(f)$	$R_{E,f}(f \oplus nb)$
Top hat	$\hat{T}_b(f)$	$f - (f \circ b)$
Bottom hat	$\hat{B}_b(f)$	$(f \bullet b) - f$
Top hat by reconstruction	$\hat{T}_R(f)$	$f - O_{R,f}(f)$
Bottom hat by reconstruction	$\hat{B}_R(f)$	$C_{R,f}(f) - f$

Table 7.2: Overview of the morphological operations on grayscale images

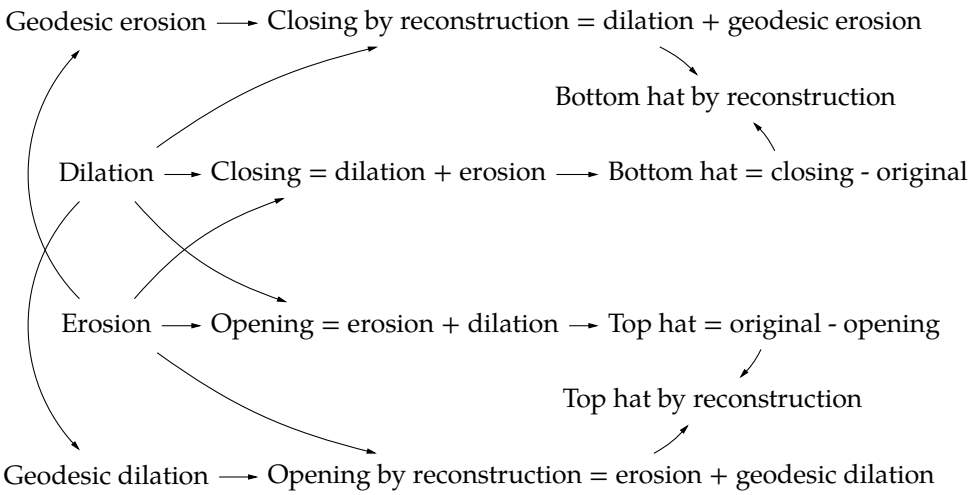


Figure 7.31: Relationships between the morphological operations on grayscale images

7.4.5 Applications

7.4.5.1 Lighting correction

Consider the number plate of a car shown in Figure 7.32a. The image is unevenly lit due to the shadow of the car itself. Before launching a number plate recognition algorithm, based on registration and the hit/miss transformation, it makes sense to perform some lighting correction. This kind of preconditioning step is common in image processing. As a first step, the image has been converted to grayscale (see Figure 7.32b). Then, a closing was performed to remove the text (see Figure 7.32c). This closing has been used in a bottom hat transformation ($\hat{B}_g(f) = (f \bullet g) - f$) and then inverted to obtain the result of Figure 7.32d. This is a version in which the influence of the uneven lighting has been removed pretty well. The result obtained is ready for a threshold transformation, to obtain the binary image of Figure 7.32e, that can be used for further processing.

7.4.5.2 Morphological smoothing

A basic observation of the operations opening and closing, results in the following conclusions:

- opening removes slight lighter noise elements, and
- closing removes slight darker noise elements.

In fact the action is very similar to the min and max filters of order-statistic filtering.

Removing noise from an image is often called *smoothing* and when using an opening or closing to this end, we call this *morphological smoothing*.

This has been illustrated in Figure 7.33 on the same salt-n-pepper test image as used in the previous chapters.

7.4.5.3 Morphological gradient

In regions of constant intensity, dilation and erosion have little impact on an image. In regions of strong fluctuation, the dilation will thicken light objects and the erosion will thin them, locally causing significant differences between the two.

Therefore it makes sense to use this property to do edge detection in a similar manner as discussed in section 7.3.5.1 on page 174.

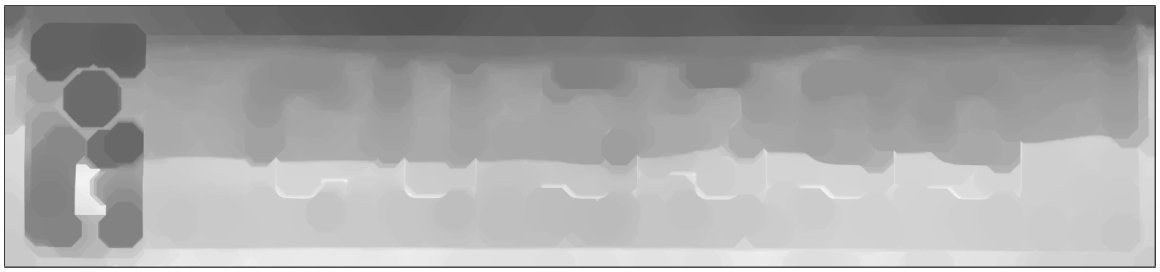
The principle is to subtract an eroded version of the image from a dilated version of the image. The result gives us the edges. This is commonly called the *morphological gradient*:

$$\left\|\vec{\nabla} f\right\| = (f \oplus b) - (f \ominus b)$$

Note, however, that only magnitude information is present, and no directional information. One might argue that the gradient is orthogonal to the edge, but still the ambiguity of the sense according to this orthogonal direction remains.

(a) Original 1572 × 356 image of a car number plate (source: Wikimedia commons, user Dantadd)

(b) Image converted to grayscale



(c) Image after closing using a disc of radius 5

(d) Image after bottom hat transformation and inversion

(e) Image converted to binary image using threshold transformation

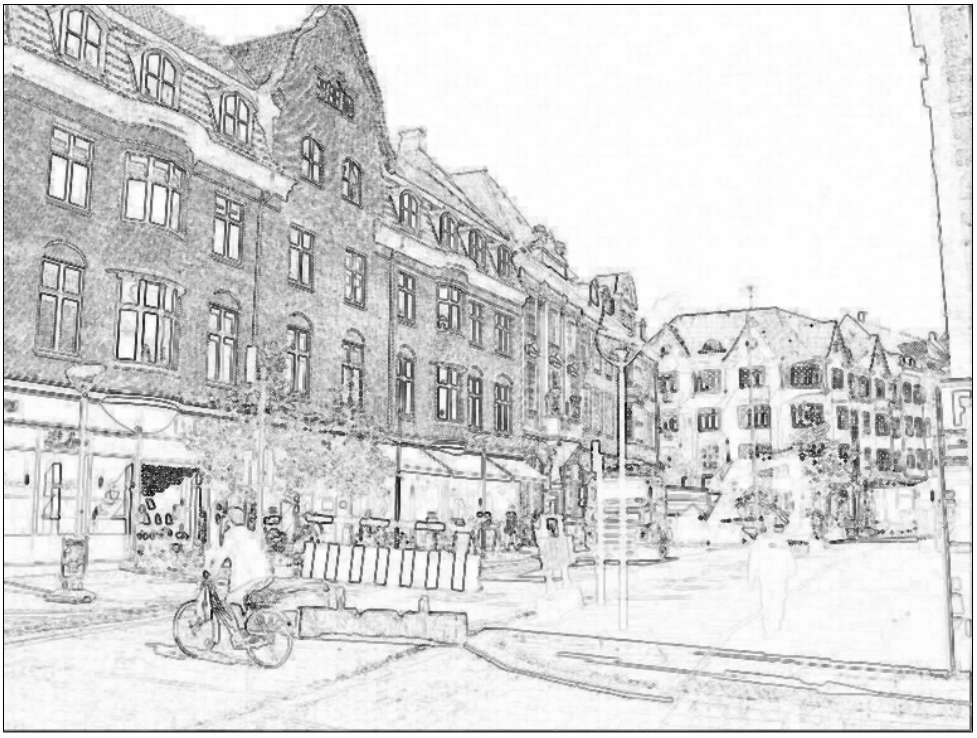
Figure 7.32: Lighting correction of light background using grayscale bottom hat transformation

(a) Original 320 × 428 image

(b) Image without salt noise by morphological smoothing by opening using a 7 × 7 square structuring element

Figure 7.33: Illustration of a morphological smoothing on a test image (source: Wikimedia commons, user Marko Meza)

(a) Original 769 × 577 image



(b) Morphological gradient image obtained by dilating and eroding using a disc-shaped structuring element with radius 1 (white = background, black = foreground)

Figure 7.34: Illustration of morphological gradient calculation on a test image of the town of Aalborg in Denmark (source: Walter Daems)

The principle has been illustrated in Figure 7.34.

7.5 Conclusion

A lot of morphological operations have been treated in this chapter. Some of them are suited for preconditioning images, some of them are capable of analyzing the various objects in the images. The key remains to be creative using them to obtain a specific effect.

Chapter 8

Image Segmentation

In this chapter, you will learn about:

- what image segmentation is,
- why we perform it,
- how to perform segmentation based on edge detection and region growing,
- how to use gradients and Laplacians to detect edges, in combination with
  - thresholding,
  - zero crossing detection,
- how noise influences the detection of images,
- using simple and advanced algorithms to perform segmentation.

After having read/studied this chapter, you are expected to be able to

- understand and explain the basic concepts of image segmentation,
- apply these basics concepts,
- mitigate noise in edge detection,
- understand and explain the common algorithms in segmentation,
- apply these algorithms to simple example images.

8.1 Introduction

8.1.1 Goal

An image can be considered as a big set of pixels. Yet, the individual pixels do not carry much information. It is the grouping of pixels in objects or into what is considered background (unimportant pixels) that makes images useful. We call this process of attributing pixels to individual sets *segmentation*. These segments will be the basis for further image analysis.

From a mathematical point of view, segmentation corresponds to partitioning.

8.1.2 Definition

Segmentation Segmentation is the process of splitting the domain of an image into useful parts that are suited for further image processing.

This means that we'll *partition* the domain of an image $S \subset \mathbb{Z}^2$ into segments S_i, such that:

1. $\bigcup_{i=1}^{N} S_i = S$
2. $S_i \cap S_j = \emptyset, \forall i \neq j$
3. S_i is a connected region

8.1.3 Classification

Partitioning the domain of an image can be done in many ways. The two most common ones are:

1. by finding the borders separating groups of pixels that do not belong together; this is called *edge based segmentation*,
2. by finding the regions of pixels that belong together; this is called *region-based segmentation*.

The former searches for the discontinuities in the intensity/color profile of an image, the latter for the similarities in the intensity/color values of an image.

8.2 Thresholding

Before diving into image segmentation, we need to take a look at an auxiliary technique, that can also be considered to be a basic form of segmentation: thresholding.

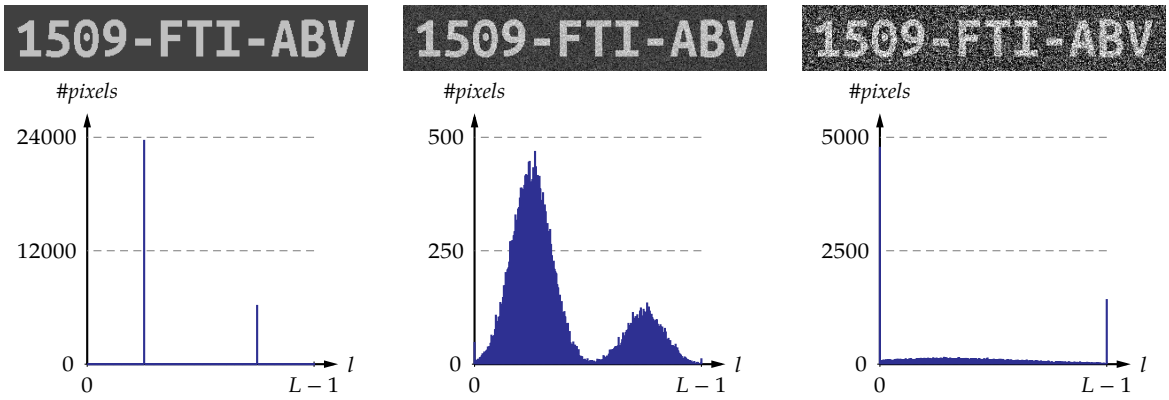
The peculiar thing about thresholding is that it does not take into account the constellation of an image. It treats all pixels just by considering their individual intensity value, not their location.

We will discuss three options:

- Average mean thresholding
- Optimal thresholding
- Multivariable thresholding

The basis for these three methods is common: defining sets/ranges of intensity values that belong together into a segment.

To simplify the setting, let's consider it our goal to discern the foreground and background pixels in an image, based on their intensity values, assuming foreground pixels have a high intensity value and background pixels a low intensity. In that way, we create a binary partition of the image, labeling all values above the intensity threshold l_T as belonging to foreground pixels, and all values below the intensity threshold l_T as belonging to background pixels. In that sense thresholding converts a grayscale image to a binary image.



(a) Without noise (b) With a limited amount of noise (c) With a lot of noise

Figure 8.1: Some images of a number plate and the corresponding intensity histograms

The key question is of course: how do we determine a good threshold value l_T?

Consider as an example the image of the number plate in Figure 8.1a. This is a version of the image without noise. Finding a proper threshold value l_T is easy in this case: any value in between the two peaks does a perfect job.

Then, take a look at Figure 8.1b. This is a version of the image with a limited amount of Gaussian white noise added to it. The noise results in the spreading of the two intensity peaks into two hills separated by a clean valley. Selecting a proper threshold value l_T is still a reasonable task: set the value right in the middle of the value, and we will obtain a reasonable separation of foreground and background pixels.

Finally, take a look at Figure 8.1c. This is a version of the image with a considerable amount of Gaussian white noise added to it. The intensity histogram is flat except for some peaks at the intensity values of 0 and $L - 1$ due to clipping of the intensity values after the noise has been added. Selecting a proper threshold is no easy task in this case.

A number of conclusions are imminent:

1. Without noise, selecting a threshold is easy. Even writing an algorithm to do the job for you should pose no problem to you.
2. With a limited amount of noise, the problem is still solvable, even writing an algorithm should pose no problem: the threshold is to be put at the minimum of the curve that is in the middle part of the image. However, what will you do if the noise is slightly more complexly shaped and there are more valleys, possibly not positioned in the middle? The extreme of that situation corresponds to the case with a significant amount of noise.
3. Note that you can still read the numbers and letters on the plate easily. Our brain is an excellent segmentation engine. The reason is obvious: we are not restricted to the intensity information, but we can also use the configuration of the pixels, given our knowledge of the alphabet and Arabic numbers.

Of course, thresholding can be more than a purely binary partitioning. Thresholding schemes that use more classes are viable as well.

8.2.1 Average mean thresholding

The first method that we will treat is basically a trial-and-error method. It is well appreciated for its simplicity.

The goal of this thresholding method is to divide the pixels into two classes: foreground pixels (class C_1) and background pixels (class C_2). The starting point is that we pick an arbitrary value l_T, such that neither C_1, nor C_2 is empty. Then we calculate the mean of both classes and take the average of those means as new threshold. Then, we iterate on that principle.

In algorithmic form, this becomes:

Algorithm: Average mean thresholding

1. Choose an initial threshold l_T and a stopping criterion parameter Δl
2. Do:
   1. Determine two classes: $\begin{cases} C_1 = \{[x,y] \mid l[x,y] < l_T\} \\ C_2 = \{[x,y] \mid l[x,y] \geq l_T\} \end{cases}$
   2. Determine the average intensity of both classes: μ_1 and μ_2.
   3. Let $l_o = l_T$ and $l_T = \dfrac{\mu_1 + \mu_2}{2}$

   while $|l_T - l_o| > \Delta l$
3. The threshold is l_T

This algorithm performs well if the two pixel classes C_1 and C_2 are well separated. The algorithm may end up in a limit cycle. In that case some additional stopping criterion harness code is required. We did not add that in order not to distract the attention from the basic idea.

Exercises

Exercise 8.2.1-1: Consider the following histogram table corresponding to a 20 × 20 image, showing the distribution of the pixel intensity l versus the number of times n the pixel intensity occurs in the picture.

l	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
n	4	12	20	21	20	19	13	?	13	8	26	38	46	65	49	34

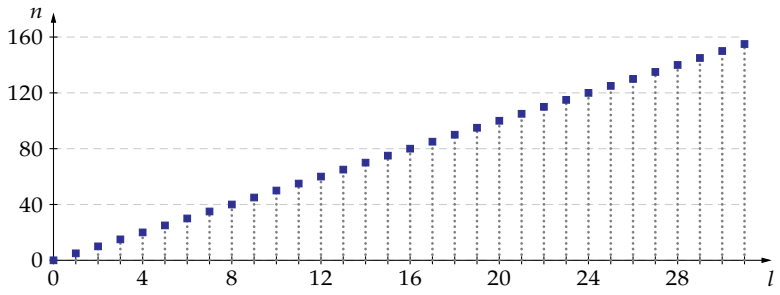
First fill in the missing value in the histogram. Then, execute average mean thresholding to determine a proper threshold value.
To gain insight in the calculations, first make an attempt in a spreadsheet, using the functions `sumproduct()` and `sum()` should get you a long way.
Then, give it a try in MATLAB/OCTAVE. Do not try to implement the full algorithm yet, just make sequential calculations.

Exercise 8.2.1-2: Consider the following histogram table corresponding to a 30 × 20 image, showing the distribution of the pixel intensity l versus the number of times n the pixel intensity occurs in the picture.

l	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
n	139	104	94	79	33	20	13	6	18	27	27	11	14	10	3	2

Execute average mean thresholding to determine a proper threshold value.
First continue adapting your sequential MATLAB/OCTAVE solution of the previous exercise to obtain proper results. You should obtain: $l_T = 5.65$.
Then, write a fully executable algorithm as a function, taking a histogram vector as input. Let's assume that the leftmost position in the vector corresponds to intensity value 0, the next value corresponds intensity value 1, and so on.

Exercise 8.2.1-3: Consider the following histogram corresponding to a 62×40 image, showing the distribution of the pixel intensity l versus the number of times n the pixel intensity occurs in the picture.



Determine a proper threshold using average mean tresholding. Reuse your algorithm composed for the previous exercise.

8.2.2 Optimum thresholding - Otsu's method

Goal

As we've seen in the previous method, a binary threshold transformation corresponds to subdividing all pixels in two classes. Otsu's method tries to optimize that process, by maximizing the *inter-class variance*. When considering two classes C_1 and C_2, the inter-class variance σ_{12} is defined as:

$$\sigma_{12}^2 = P_1(\mu_1 - \mu_G)^2 + P_2(\mu_2 - \mu_G)^2$$

with μ_i the mean of C_i, μ_G the global mean and P_i the relative class population counts, i.e. the number of pixels in the class divided by the total number of pixels. This also implies $P_1 + P_2 = 1$.

Often the inter-class variance is divided by the global variance σ_G^2 to obtain the class separability:

$$\eta = \frac{\sigma_{12}^2}{\sigma_G^2}$$

with $0 \leq \eta \leq 1$. Separabilities below 0.6 are bad. The closer to 1, the better the separability (and the thresholding) will be.

Theoretical foundation

Let's develop Otsu's method on an $M \times N$ image that has been quantized with L levels (from 0 to $L-1$). We start by considering the normalized histogram. Assuming that there are n_l pixels with intensity value l, the histogram function f is defined as:

$$f(l) = \frac{n_l}{MN}$$

In view of this definition, of course $\sum_{l=0}^{L-1} f(l) = 1$.

An arbitrary threshold value l_T, partitions the pixels into two classes

$$C_1 = \{[x,y] \mid l[x,y] < l_T\},$$
$$C_2 = \{[x,y] \mid l[x,y] \geq l_T\}$$

that each have a fraction P_i of all pixels in them, with:

$$P_1 = \sum_{l=0}^{l_T-1} f(l),$$
$$P_2 = \sum_{l=l_T}^{L-1} f(l).$$

Evidently, the class means can be calculated as

$$\mu_1 = \frac{\sum_{l=0}^{l_T-1} l f(l)}{P_1},$$
$$\mu_2 = \frac{\sum_{l=l_T}^{L-1} l f(l)}{P_2},$$

and the global mean can be calculated using the class means:

$$\mu_G = \sum_{l=0}^{L-1} l \cdot f(l) = P_1\mu_1 + P_2\mu_2.$$

The inter-class variance can be reworked, to allow for easy computation, only requiring the computation of μ_G (once), and P_1 and μ_1 for every value of l_T.

$$\begin{aligned}
\sigma_{12}^2 &= P_1(\mu_1 - \mu_G)^2 + P_2(\mu_2 - \mu_G)^2 \\
&\downarrow \quad \mu_2 = \frac{\mu_G - P_1\mu_1}{P_2} \\
&= P_1(\mu_1 - \mu_G)^2 + P_2\left(\frac{\mu_G - P_1\mu_1}{P_2} - \mu_G\right)^2 \\
&= P_1(\mu_1 - \mu_G)^2 + \frac{1}{P_2}(\mu_G - P_1\mu_1 - P_2\mu_G)^2 \\
&= P_1(\mu_1 - \mu_G)^2 + \frac{P_1^2}{P_2}(\mu_G - \mu_1)^2 \\
&= \left(P_1 + \frac{P_1^2}{P_2}\right)(\mu_1 - \mu_G)^2 \\
&= \frac{P_1(P_1 + P_2)}{P_2}(\mu_1 - \mu_G)^2 \\
&= \frac{P_1}{1 - P_1}(\mu_1 - \mu_G)^2
\end{aligned}$$

Method

The method itself is a brute force method. We just try every possible value of l_T and keep track of the achieved inter-class variances.

Algorithm: Optimum thresholding - Otsu's method

1. Calculate the normalized histogram of the image
2. Calculate μ_G and σ_G^2
3. Let $l_T = 0$, $\sigma_*^2 = 0$, $M_1 = 0$ and $P_1 = 0$
4. For all $l = 0, 1, 2, \ldots, L-2$:
   - 4.1. Calculate $P_1 = P_1 + f(l)$
   - 4.2. Calculate $M_1 = M_1 + l * f(l)$
   - 4.3. Calculate $\sigma_{12}^2 = \frac{P_1}{1-P_1}\left(\frac{M_1}{P_1} - \mu_G\right)^2$
   - 4.4. If $\sigma_{12}^2 > \sigma_*^2$, then:
     - 4.4.1. $l_T = l + 1$
     - 4.4.2. $\sigma_*^2 = \sigma_{12}^2$
5. l_T is the optimal threshold value with separability $\eta(l_T) = \sigma_*^2/\sigma_G^2$

Remarks

- Filtering the noise improves the separability
- When multiple maxima σ_*^2 occur for neighboring thresholds, we might select their average. This has not been implemented in the algorithm.
- When a small foreground object drowns in the multitude of background pixels, we only incorporate pixels near the edge in the normalized histogram of Otsu's method, or we subdivide the image in subimages.

Example

Let's apply Otsu's method to a small example. Consider a 13 × 17-image (221 pixels) quantized using 16 levels. One can find the frequency table below:

l	n_l	l	n_l	l	n_l	l	n_l
0	1	4	10	8	4	12	47
1	2	5	10	9	10	13	28
2	4	6	7	10	28	14	10
3	7	7	4	11	47	15	2

To be able to execute Otsu's method, we calculate $f(l)$, P_1, and M_1 for every l_T ranging from 0 to 15. This allows calculating the corresponding σ_{12}^2.

The result can be found in Table 8.1. The optimal σ_{12}^2 is obtained for $l_T = 8$, and amounts to 8.44. Note that step 4.4.1 in the algorithm sets the optimal l_T equal to $l + 1$, with l the value for which the highest σ_{12}^2 has been obtained.

The separation has been indicated in Figure 8.2.

l	n_l	$f(l)$	P_1	M_1	σ_{12}^2
0	1	0.0045	0.004525	0.000000	0.453723
1	2	0.0090	0.013575	0.009050	1.196453
2	4	0.0181	0.031674	0.045249	2.398132
3	7	0.0317	0.063348	0.140271	4.090199
4	10	0.0452	0.108597	0.321267	6.025304
5	10	0.0452	0.153846	0.547511	7.522228
6	7	0.0317	0.185520	0.737557	8.241984
7	4	0.0181	0.203620	0.864253	8.443209
8	4	0.0181	0.221719	1.009050	8.430517
9	10	0.0452	0.266968	1.416290	7.996807
10	28	0.1267	0.393665	2.683258	6.544299
11	47	0.2127	0.606335	5.022624	4.489933
12	47	0.2127	0.819005	7.574661	2.493528
13	28	0.1267	0.945701	9.221719	1.001145
14	10	0.0452	0.990950	9.855204	0.229138
15	2	0.0090	1.000000	9.990950	

Table 8.1: Summary of the intermediate results of applying Otsu's method to the example image for every value of l

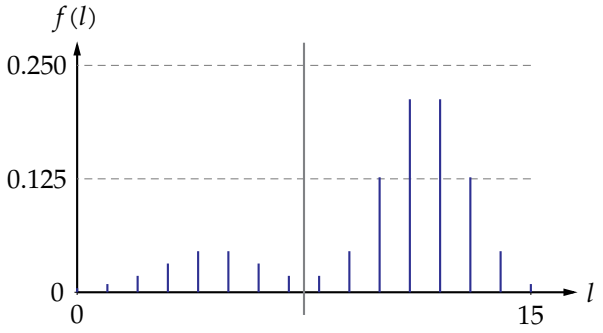


Figure 8.2: Normalized histogram, with indicated optimal class separation obtained by applying Otsu's method, of the example image

Exercises

We will treat the same exercises as for average thresholding.

Exercise 8.2.2-1: Consider the following histogram table corresponding to a 20 × 20 image, showing the distribution of the pixel intensity l versus the number of times n the pixel intensity occurs in the picture.

l	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
n	4	12	20	21	20	19	13	12	13	8	26	38	46	65	49	34

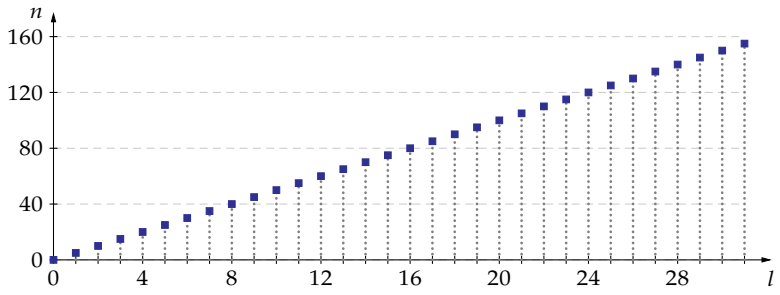
Execute Otsu's algorithm to determine a proper threshold value.
To gain insight in the calculations, do this in a spreadsheet implementing the calculations.

Exercise 8.2.2-2: Consider the following histogram table corresponding to a 30 × 20 image, showing the distribution of the pixel intensity l versus the number of times n the pixel intensity occurs in the picture.

l	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
n	139	104	94	79	33	20	13	6	18	27	27	11	14	10	3	2

Execute Otsu's method to determine a proper threshold value.
First, use your spreadsheet of the previous exercise. You should obtain: $l_T = 9$ (remember this is one higher than the value of l for which the maximal interclass variance occurs).
Now, write Otsu's algorithm in a MATLAB/OCTAVE function, taking a histogram vector as input. Let's assume that the leftmost position in the vector corresponds to intensity value 0, the next value corresponds intensity value 1, and so on.

Exercise 8.2.2-3: Consider the following histogram corresponding to a 62 × 40 image, showing the distribution of the pixel intensity l versus the number of times n the pixel intensity occurs in the picture.



Execute Otsu's method to determine a proper threshold value. Reuse your algorithm of the previous exercise.

Exercise 8.2.2-4: Explore the MATLAB/OCTAVE function `graythresh()`. It performs thresholding according to Otsu's method. Try it on the previous exercises. Can you apply it directly on the given data?

8.2.3 Multivariable thresholding

So far, we have only applied thresholding to grayscale images. What if we want to apply thresholding to a color image? Of course, we might apply the grayscale thresholding to any of the color values. Choosing the right color model is important to obtain good results.

We also may use a distance metric that's valid in the chosen color model.

8.2.3.1 Principle

For a color image the color values of the pixels are represented using vectors (using one dimension per color variable, e.g., R, G, B or H, S, I). We denote the color vector by $\vec{c}$.

We can apply any grayscale method by mapping the color vector to a pseudo grayscale value.

To this end, we use the distance d between the pixel's color vector $\vec{c}[x,y]$ and a target color vector $\vec{c}_0$.

$$d(\vec{c}[x,y], \vec{c}_0)$$

8.2.3.2 Distance metric

Multiple distance metrics can be used. We just list two commonly used ones below.

Euclidean distance

$$\begin{aligned} d(\vec{c}[x,y], \vec{c}_0) &= \left\| \vec{c}[x,y] - \vec{c}_0 \right\|_2 \\ &= \sqrt{\left(\vec{c}[x,y] - \vec{c}_0\right)^T \left(\vec{c}[x,y] - \vec{c}_0\right)} \end{aligned}$$

Mahalanobis distance

$$\begin{aligned} d(\vec{c}[x,y], \vec{c}_0) &= \left\| \vec{c}[x,y] - \vec{c}_0 \right\|_M \\ &= \sqrt{\left(\vec{c}[x,y] - \vec{c}_0\right)^T A \left(\vec{c}[x,y] - \vec{c}_0\right)} \end{aligned}$$

with A the inverse of the covariance matrix of the pixel set. If the pixels don't exhibit a specific covariance, i.e. $A = I$, then the *Mahalanobis*-distance corresponds to the Euclidean distance.

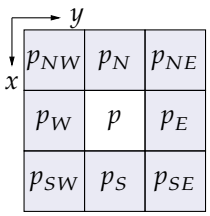
8.3 Detecting zero crossings

Another auxiliary technique we need to investigate before committing ourselves to edge-based segmentation is the detection of zero crossings. A zero crossing of a one-dimensional function is a clear-cut concept. We could also define an exact analytical equivalent for two or more dimensions. However, we will restrict ourselves to two possible definitions for discrete, two-dimensional intensity functions.

8.3.1 Definition

Before taking off, remember the concept of *neighborhoods* of a pixel (see section 7.2.1.1 on page 142). To ease the definitions below, we will introduce a new notation.

Consider a pixel p and its 8-neighborhood ($N_8(p)$). Let's start by denoting the neighboring pixels using the cardinal and intercardinal wind directions as subscripts.



In view of this new notation, we could write:

$$N_4(p) = \{p_N, p_E, p_S, p_W\}$$
$$N_8(p) = \{p_N, p_E, p_S, p_W, p_{NE}, p_{SE}, p_{SW}, p_{NW}\}$$

Now, let's make a new notation for the concept 'the opposite pixel in the neighborhood'. We will denote the opposite pixel of a neighboring pixel by writing a superscripted r next to it (r for 'reflection'). We denote explicitly the origin p of the reflection by adding a subscripted p to the r.

Therefore:

$$p_N^{r_p} = p_S \qquad p_{NE}^{r_p} = p_{SW}$$
$$p_S^{r_p} = p_N \qquad p_{SW}^{r_p} = p_{NE}$$
$$p_E^{r_p} = p_W \qquad p_{SE}^{r_p} = p_{NW}$$
$$p_W^{r_p} = p_E \qquad p_{NW}^{r_p} = p_{SE}$$

Now you are ready to understand the definition of zero crossing.

N_4 zero crossing

Definition of discrete N_4 zero crossing

Consider an image f with pixels $p \in \text{dom}\, f$.
The image has a zero crossing in p if it has a strictly positive *zero crossing strength* S at pixel p, with

$$S(p) = \max_{q \in N_4(p)} S_q(p)$$

and S_q defined depending on the value of $f(p)$:

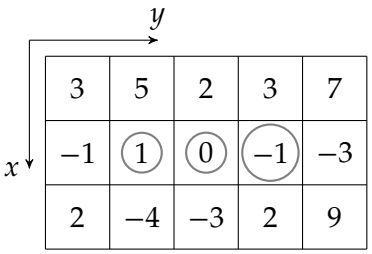
- if $f(p) > 0$, then
$$S_q(p) = 0$$
- if $f(p) < 0$, then
$$S_q(p) = \begin{cases} f(q) - f(p) & \text{if } f(q) > 0 \\ 0 & \text{otherwise} \end{cases}$$
- if $f(p) = 0$:
$$S_q(p) = \begin{cases} \left| \frac{f(q) - f(q^{r_p})}{2} \right) \Big| & \text{if } f(q) \cdot f(q^{r_p}) < 0 \\ 0 & \text{otherwise} \end{cases}$$

N_8 zero crossing

Replacing N_4 with N_8 in the definition for N_4 zero crossing above and introducing a correction factor of $1/\sqrt{2}$ for the diagonal directions, yields a proper definition for N_8 zero crossing. Try to write it down, yourself!

Example

Consider the circled pixels below. Let's determine their N_4 zero strength.



Let's start with the pixel of the leftmost circle. This is the most simple case as the pixel intensity is positive. So:

$$f(p) > 0 \Rightarrow S(p) = 0$$

Conclusion: there is no N_4 zero crossing at that position.

The rightmost circled pixel has a negative intensity and therefore needs investigation w.r.t. it's N_4 neighboring pixels.

$$f(p) < 0 \Rightarrow S(p) = \max\left(\underbrace{3 - (-1)}_{q=p_N}, \underbrace{0}_{q=p_E}, \underbrace{2 - (-1)}_{q=p_S}, \underbrace{0}_{q=p_W} \right) = 4$$

Conclusion: there is an N_4 zero crossing at that position, with strength 4.

The middle case is the most complex one:

$$f(p) = 0 \Rightarrow S(p) = \max\left(\underbrace{\left|\frac{2-(-3)}{2}\right|}_{q=p_N}, \underbrace{\left|\frac{-1-1}{2}\right|}_{q=p_E}, \underbrace{\left|\frac{-3-2)}{2}\right|}_{q=p_S}, \underbrace{\left|\frac{1-(-1)}{2}\right|}_{q=p_W}\right) = \frac{5}{2}$$

Conclusion: there is also an N_4 zero crossing at the middle pixel, with strength 2.5.

Can you search for N_8 zero crossings as well?

8.3.2 Zero crossing detection as an image operation

It would be nice to be able to obtain a binary image g, corresponding to an image f, that contains ones for every pixel with a zero-crossing and zeros for the other pixels.

Given the fact that $S(p)$ is defined for every pixel p, it composes a new grayscale image (possibly out of the quantization bounds, but that is not an issue), the so-called zero crossing image S.

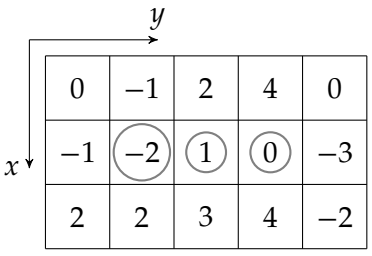
Thresholding this image yields a new binary image h

$$h = (S > T)$$

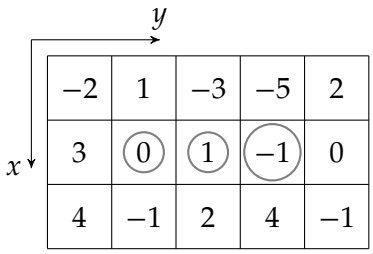
in which we assumed the $>$-operator to be a matrix operator that yields a binary matrix with ones for pixels where the inequality holds and zeros elsewhere.

Exercises

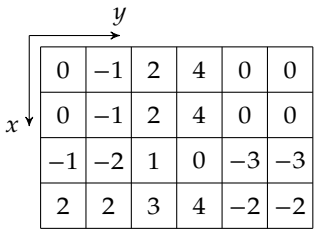
Exercise 8.3.2-1: Determine the N_4 and the N_8 zero-crossing strenght of the pixels that have been circled. Based on the value found, determine whether a zero-crossing appears at the pixel according to the corresponding criterion.



Exercise 8.3.2-2: Determine the N_4 and the N_8 zero-crossing strenght of the pixels that have been circled. Based on the value found, determine whether a zero-crossing appears at the pixel according to the corresponding criterion.



Exercise 8.3.2-3: Apply the N_4 zero-crossing detection as an image operation on the image below. Treat border pixels by not looking beyond the border.



Exercise 8.3.2-4: Apply the N_8 zero-crossing detction as an image operation on the image of the previous exercise. Treat border pixels by not looking beyond the border.

8.4 Edge-based segmentation

Edge-based segmentation is mostly based on analyzing gradients and Laplacians of images. If you have no clue on what these are, it makes sense to study Chapter 2. Without a thorough understanding of discrete calculus, you'll get lost in the wilderness of segmentation.

8.4.1 Foundations

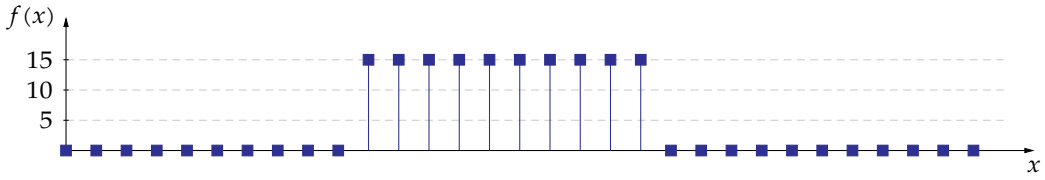
Our goal is to detect edges using gradients (first-order derivative) and Laplacians (second-order derivative). So let's start by analyzing how edges influence gradients and Laplacians. We will consider different types of edges to get a complete view. To keep things simple let's consider the first- and second-order derivative of an intensity function seen along a straight line (cut). When this line is parallel to one of the axes, in fact, we are analyzing the first and second order partial derivatives of the intensity function $f[x, y]$ along that specific axis.

In order to calculate the first-order and second-order derivative, we will use convolution kernels g_x and l_x respectively:

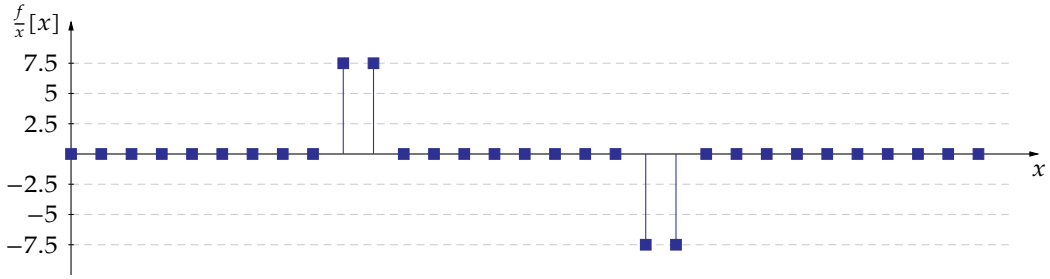
$$g_x = \begin{bmatrix} \frac{1}{2} & 0 & -\frac{1}{2} \end{bmatrix}$$
$$l_x = \begin{bmatrix} 1 & -2 & 1 \end{bmatrix}$$

A step edge

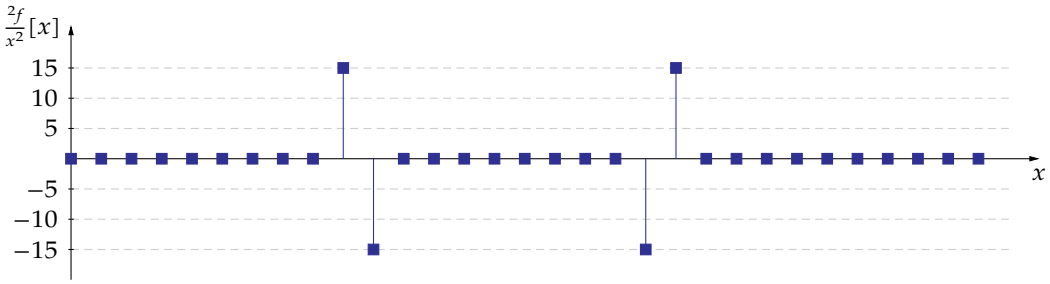
Consider a linear cut through an image that exhibits two step edges:



The first-order derivative looks like:



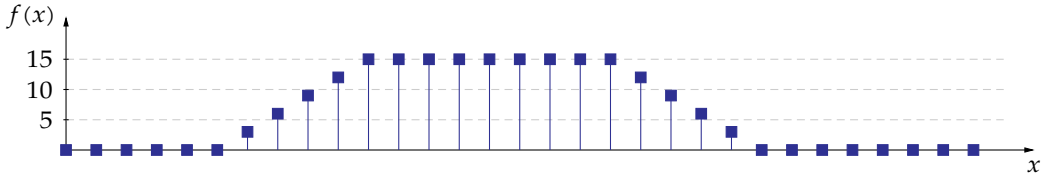
The second-order derivative looks like:



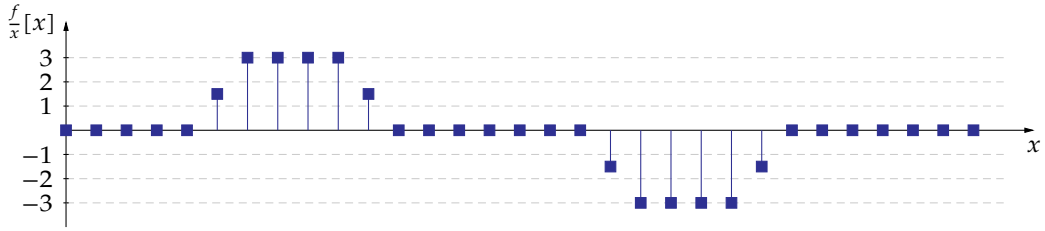
Conclusion: at the position of the step, the first-order derivative is nonzero and the second-order derivative makes a single alternation (including a zero crossing).

A ramp

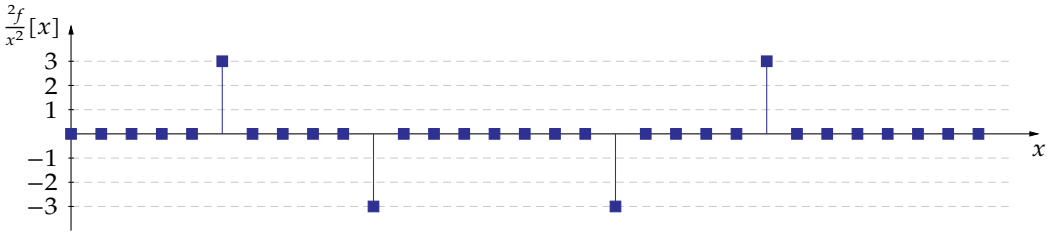
Consider a linear cut through an image that exhibits two ramp edges:



The first-order derivative looks like:



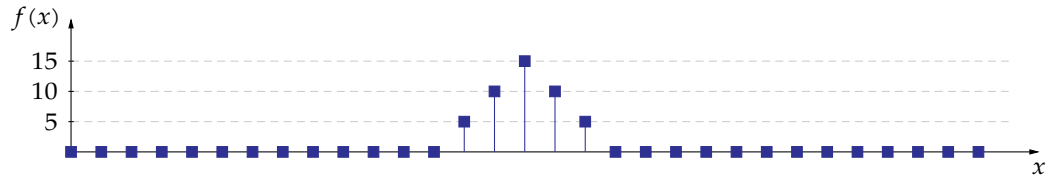
The second-order derivative looks like:



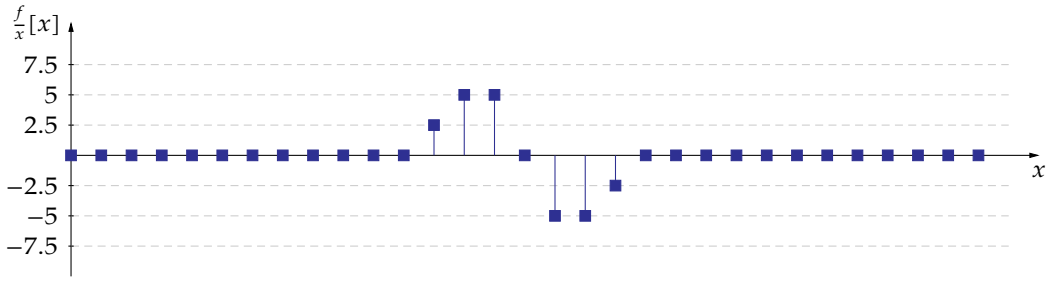
Conclusion: at the position of the ramp, the first-order derivative is nonzero throughout the length of the ramp and the second-order derivative makes a single alternation (including a zero crossing) spread over the length of the ramp.

A roof

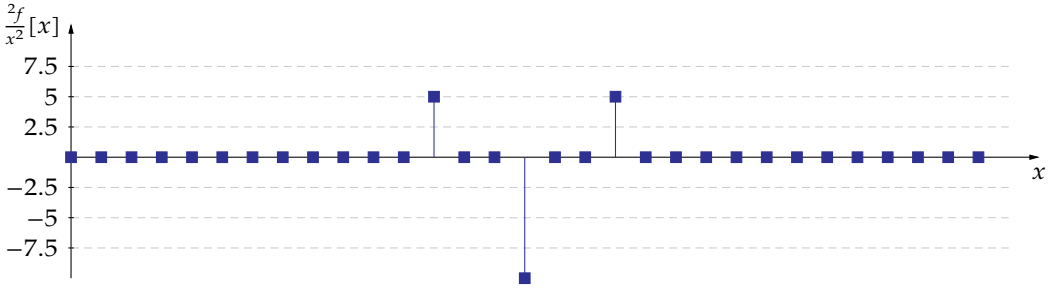
Consider a linear cut through an image that exhibits a roof:



The first-order derivative looks like:



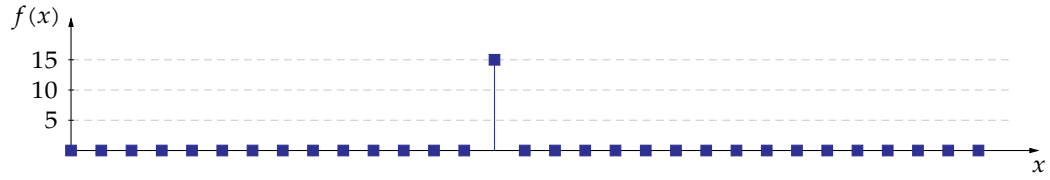
The second-order derivative looks like:



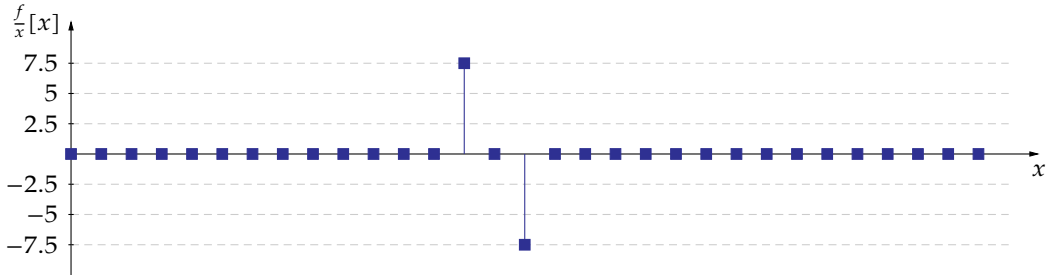
Conclusion: the first-order derivative is nonzero throughout the entire length of the roof (except for the occasional point where it may be zero when the roof is perfectly symmetrical). The second-order derivative makes two alternations that share a common (middle) point.

A spike

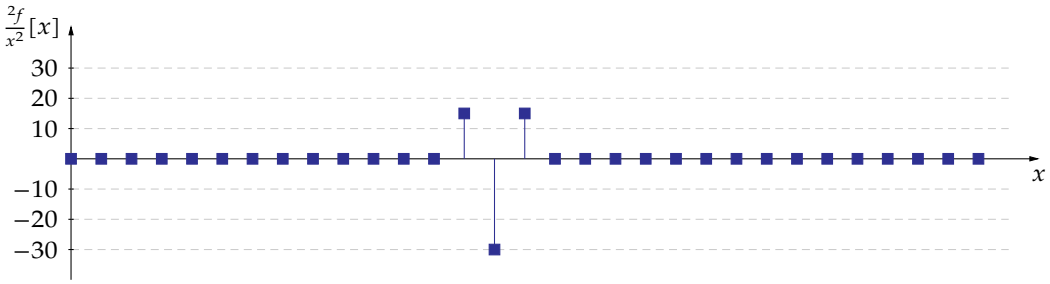
Consider a linear cut through an image that exhibits a spike:



The first-order derivative looks like:



The second-order derivative looks like:



Conclusion: The first-order derivative alternates during a spike and the second-order derivative makes two alternations that share a common (middle) point.

Overall conclusion

When edges occur:

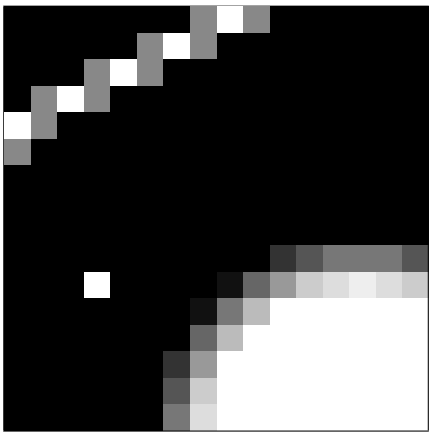
- the magnitude of the gradient is large
- the magnitude of the Laplacian is large and exhibits a significant alternation (with zero crossing)

Therefore, our main tool to detect edges will be to perform thresholding operations on the gradient image or Laplacian image or to investigate zero crossings in the latter.

8.4.2 Simple (theoretical) algorithms

8.4.2.1 Edge detection using the gradient

Based on the fact that when an edge occurs, the absolute value of the gradient is nonzero, deriving the following algorithm is quite straightforward:



(a) Test image

f (y →, x ↓)

0	0	0	0	0	0	0	8	15	8	0	0	0	0	0	0
0	0	0	0	0	0	8	15	8	0	0	0	0	0	0	0
0	0	0	8	15	8	0	0	0	0	0	0	0	0	0	0
0	8	15	8	0	0	0	0	0	0	0	0	0	0	0	0
15	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	3	5	7	7	7	5
0	0	0	15	0	0	0	0	1	6	9	12	13	14	13	12
0	0	0	0	0	0	0	1	7	11	15	16	16	16	16	16
0	0	0	0	0	0	0	6	11	16	16	16	16	16	16	16
0	0	0	0	0	0	3	9	15	16	16	16	16	16	16	16
0	0	0	0	0	0	5	12	16	16	16	16	16	16	16	16
0	0	0	0	0	0	7	13	16	16	16	16	16	16	16	16

(b) Intensity function f of the test image

Figure 8.3: 16 × 16 test image, quantized using 17 levels

Algorithm: Gradient edge detection

1. $g_x = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$ and $g_y = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$
2. $\vec{\nabla} f[x,y] = \vec{e}_x(f \star g_x) + \vec{e}_y(f \star g_y)$
3. Convert to a binary image R using a threshold transformation:
   $R = \left(\left|\vec{\nabla} f[x,y]\right| > T\right)$
   The resulting binary image R contains the edge pixels in its foreground.

Simple example To illustrate this algorithm, let's apply it to a simple example. Consider the 16 × 16 4-bit quantized test image of Figure 8.3.

Executing step one and two of the algorithm yields the following two partial derivative images, constituting the gradient together:

$f \star g_x$ (y →, x ↓)

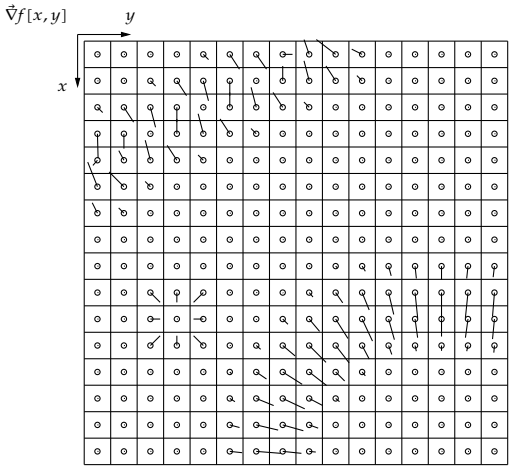
0	0	0	0	8	23	23	0	-23	-23	-8	0	0	0	0	0
0	0	8	23	31	23	0	-23	-31	-23	-8	0	0	0	0	0
8	23	31	23	0	-23	-31	-23	-8	0	0	0	0	0	0	0
38	23	0	-23	-31	-23	-8	0	0	0	0	0	0	0	0	0
8	-15	-31	-23	-8	0	0	0	0	0	0	0	0	0	0	0
-38	-23	-8	0	0	0	0	0	0	0	0	0	0	0	0	0
-16	-8	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	3	8	15	19	21	19	17
0	0	15	15	15	0	0	1	7	16	27	34	39	40	39	37
0	0	0	0	0	0	1	8	19	30	34	32	29	27	29	31
0	0	-15	-15	-15	0	6	16	26	27	21	14	9	8	9	11
0	0	0	0	0	3	11	19	21	14	6	1	0	0	0	0
0	0	0	0	0	5	11	16	11	5	0	0	0	0	0	0
0	0	0	0	0	4	8	9	5	1	0	0	0	0	0	0
0	0	0	0	0	2	3	3	1	0	0	0	0	0	0	0

$f \star g_y$ (y →, x ↓)

0	0	0	0	8	15	16	15	-8	-30	-16	0	0	0	0	0
0	0	8	15	8	0	0	0	-8	-15	-8	0	0	0	0	0
8	15	8	0	0	0	-8	-15	-8	0	0	0	0	0	0	0
1	0	0	0	-8	-15	-8	0	0	0	0	0	0	0	0	0
-7	-8	-8	-15	-8	0	0	0	0	0	0	0	0	0	0	0
-15	-23	-8	0	0	0	0	0	0	0	0	0	0	0	0	0
-8	-8	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	3	5	4	2	0	-2	-2
0	0	15	0	-15	0	0	1	6	11	11	8	4	0	-4	-3
0	0	15	0	-15	0	1	8	16	19	16	9	4	0	-4	-3
0	0	15	0	-15	0	7	19	26	21	11	5	2	0	-2	-1
0	0	0	0	0	3	16	30	27	14	5	1	0	0	0	0
0	0	0	0	0	8	27	34	21	6	0	0	0	0	0	0
0	0	0	0	0	15	34	32	14	1	0	0	0	0	0	0
0	0	0	0	0	19	38	29	10	0	0	0	0	0	0	0

Combining these two partial derivative images, yields the following gradient image on the left

below, in which the size and direction of the gradient has been indicated with small lines. The norm of the gradient image can be found on the right.



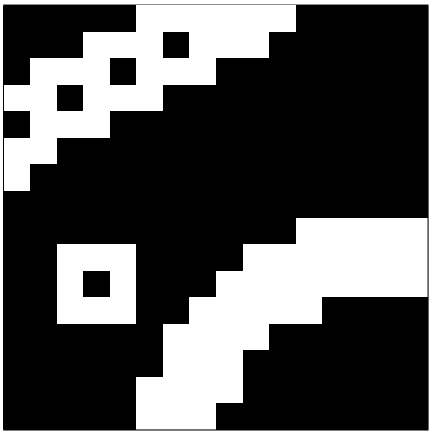
$\|\vec{\nabla} f[x,y]\|_2$

y \ x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	0	0	0	0	11	27	28	15	24	38	18	0	0	0	0
1	0	0	11	27	32	23	0	23	32	27	11	0	0	0	0
2	11	27	32	23	0	23	32	27	11	0	0	0	0	0	0
3	38	23	0	23	32	27	11	0	0	0	0	0	0	0	0
4	11	17	32	27	11	0	0	0	0	0	0	0	0	0	0
5	41	33	11	0	0	0	0	0	0	0	0	0	0	0	0
6	18	11	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	4	9	16	19	21	19	17
9	0	0	21	15	21	0	0	1	9	19	29	35	39	40	39	37
10	0	0	15	0	15	0	1	11	25	36	38	33	29	27	29	31
11	0	0	21	15	21	0	9	25	37	34	24	15	9	8	9	11
12	0	0	0	0	0	4	19	36	34	20	8	1	0	0	0	0
13	0	0	0	0	0	9	29	38	24	8	0	0	0	0	0	0
14	0	0	0	0	0	16	35	33	15	1	0	0	0	0	0	0
15	0	0	0	0	0	19	38	29	10	0	0	0	0	0	0	0

Finally, thresholding this picture with $T = 14$ yields:

R

y \ x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	0	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0
1	0	0	0	1	1	1	0	1	1	1	0	0	0	0	0	0
2	0	1	1	1	0	1	1	1	0	0	0	0	0	0	0	0
3	1	1	0	1	1	1	0	0	0	0	0	0	0	0	0	0
4	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
5	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1
9	0	0	1	1	1	0	0	0	0	1	1	1	1	1	1	1
10	0	0	1	0	1	0	0	0	1	1	1	1	1	1	1	1
11	0	0	1	1	1	0	0	1	1	1	1	1	0	0	0	0
12	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0
13	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0
14	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0
15	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0



Of course, in a real-world application, one would use Otsu's method to determine the optimal threshold value. The one above has been chosen arbitrarily.

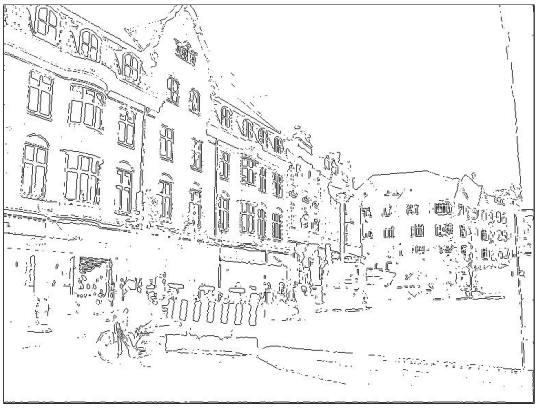
In hindsight, one of the problems of gradient-based edge detection surfaces: there are holes in the edge detected at the top left.

Real-world example Figure 8.4 illustrates the result of applying this technique on a real-world example image.

Exercises

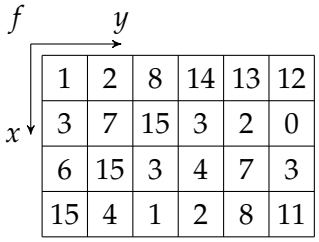
Exercise 8.4.2.1-1: Consider the 4-bit quantized 6 × 4-image f below. Apply the simple gradient-based edge-detection algorithm to this image. Assume border replication.

(a) Image of the town of Aalborg, Denmark (source: Walter Daems)



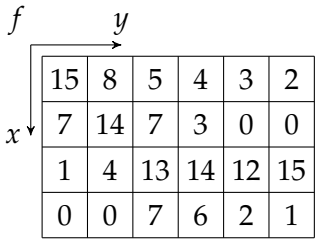
(b) Result after edge detection (white = background, black = foreground)

Figure 8.4: Illustration of the simple gradient edge detection algorithm on a real-world example



1	2	8	14	13	12
3	7	15	3	2	0
6	15	3	4	7	3
15	4	1	2	8	11

Exercise 8.4.2.1-2: Consider the 4-bit quantized 6 × 4-image h below. Apply the simple gradient-based edge detection algorithm to this image. Assume border replication.



15	8	5	4	3	2
7	14	7	3	0	0
1	4	13	14	12	15
0	0	7	6	2	1

Exercise 8.4.2.1-3: Write a MATLAB/OCTAVE filter to perform the operations above in a more automated fashion. Use the MATLAB/OCTAVE-function `imfilter()` to calculate gradients. You can use `graythresh()` to perform Otsu's method. You may need to load the image package in OCTAVE manually.

8.4.2.2 Edge detection using the Laplacian

Based on the fact that when an edge occurs, the absolute value of the Laplacian is nonzero and it shows an alternation, deriving the following algorithm is quite straightforward:

Algorithm: Laplacian edge detection

1. $l = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}$
2. $\nabla^2 f[x,y] = f \star l = f \star l$
3. Convert to a binary image R using a threshold transformation:
   $R = \left(|\nabla^2 f[x,y]| > T\right)$
   or
   find the zero crossings larger than T.

Simple example To illustrate this algorithm, let's apply it to a simple example. Consider the 16 × 16 17-levels quantized test image of Figure 8.3 on page 224.

Executing step one and two of the algorithm yields the following Laplacian image:

$f \star \lambda_8$ (x: ↓, y: →)

0	0	0	0	8	23	47	-3	-65	-26	16	0	0	0	0	0
0	0	8	23	39	-26	-88	-26	39	23	8	0	0	0	0	0
8	23	39	-26	-88	-26	39	23	8	0	0	0	0	0	0	0
46	-26	-88	-26	39	23	8	0	0	0	0	0	0	0	0	0
-73	-18	39	23	8	0	0	0	0	0	0	0	0	0	0	0
-18	31	8	0	0	0	0	0	0	0	0	0	0	0	0	0
16	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	3	8	15	19	21	19	17
0	0	15	15	15	0	0	1	7	19	8	4	-5	-2	-5	9
0	0	15	-120	15	0	1	9	17	-2	-4	-12	-11	-17	-11	-6
0	0	15	15	15	0	7	17	-4	-7	-18	-15	-9	-8	-9	-11
0	0	0	0	0	3	19	-2	-7	-21	-6	-1	0	0	0	0
0	0	0	0	0	8	8	-4	-18	-6	0	0	0	0	0	0
0	0	0	0	0	15	4	-12	-15	-1	0	0	0	0	0	0
0	0	0	0	0	19	-6	-12	-10	0	0	0	0	0	0	0

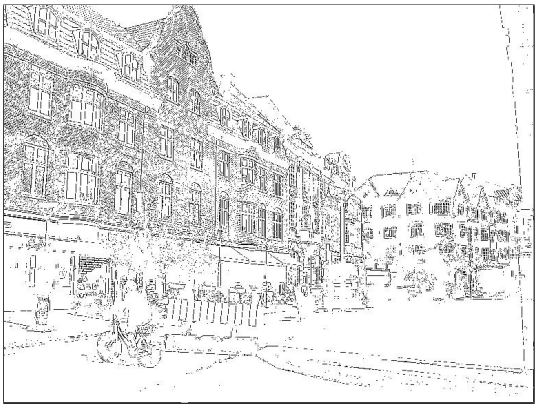
Finally, thresholding this picture with $T = 10$ yields:

R (x: ↓, y: →)

0	0	0	0	0	1	1	0	1	1	1	0	0	0	0	0
0	0	0	1	1	1	1	1	1	1	0	0	0	0	0	0
0	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
1	1	1	1	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
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	1	1	1	1
0	0	1	1	1	0	0	0	0	1	0	0	0	0	0	0
0	0	1	1	1	0	0	0	1	0	0	1	1	1	1	0
0	0	1	1	1	0	0	1	0	0	1	1	0	0	0	1
0	0	0	0	0	0	1	0	0	1	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	1	0	1	1	0	0	0	0	0	0	0
0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0

Of course, in a real-world application, one would use Otsu's method to determine the optimal threshold value, or use zero crossing detection. The threshold value above has been chosen arbitrarily.

(a) Image of the town of Aalborg, Denmark (source: Walter Daems)



(b) Result after edge detection (white = background, black = foreground)

Figure 8.5: Illustration of the simple Laplacian edge detection algorithm on a real-world example

In hindsight, some typical problems of edge detection using a Laplacian surface: there is still a hole in the border at the top and we see some double and yet incomplete edges on the bottom right.

Real-world example Figure 8.5 illustrates the result of applying this technique on a real-world example image.

Exercises

Exercise 8.4.2.2-1: Consider once again the image of exercise 8.4.2.1-1. Apply the simple Laplacian-based edge-detection algorithm to this image. Assume border replication.

Exercise 8.4.2.2-2: Consider once again the image of exercise 8.4.2.1-2. Apply the simple Laplacian-based edge-detection algorithm to this image. Assume border replication.

Exercise 8.4.2.2-3: Write a MATLAB/OCTAVE filter to perform the operations above in a more automated fashion. Use the MATLAB-function `imfilter` to calculate Laplacians. You can use `graythresh()` to perform Otsu's method. You may need to load the image package in OCTAVE manually.

8.4.2.3 Variants

Many variations on these two themes are possible:

- use a different gradient/Laplacian operator,
- use different norms (e.g., 1-norm instead of a 2-norm),
- for the gradient-operator, select specific directions:

- along the main axes,
- along the diagonals,
- along arbitrary directions.

8.4.3 More robust (practical) algorithms

The simple algorithms of the previous section show quite some deficiencies. Let's improve on them.

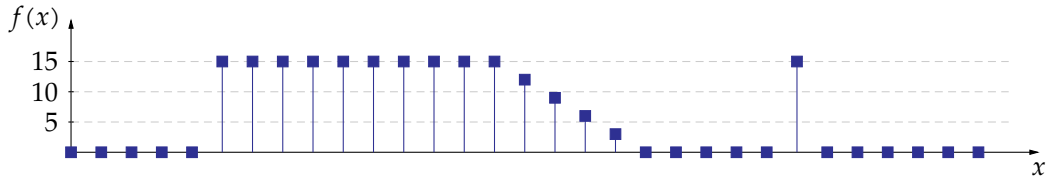
8.4.3.1 Problems with the simple algorithms and how to improve them

Let's start by listing the problems. Some we have seen, some did not yet surface, but are easy to understand.

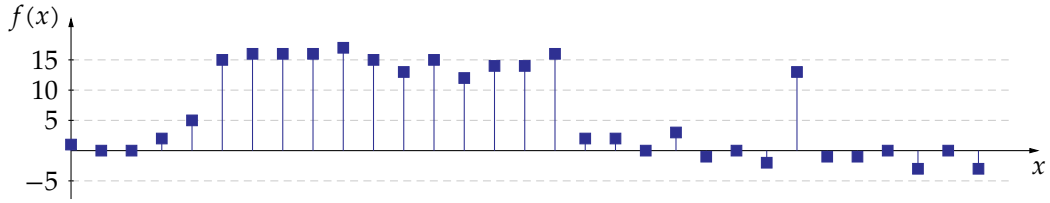
Noise

Noise is a real burden when it comes to edge detection. Let's investigate its influence by considering a small example.

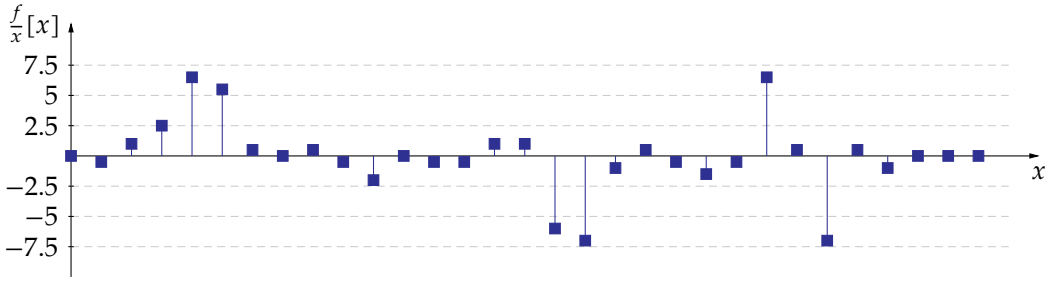
Consider a linear cut through an image that exhibits a number of edges:



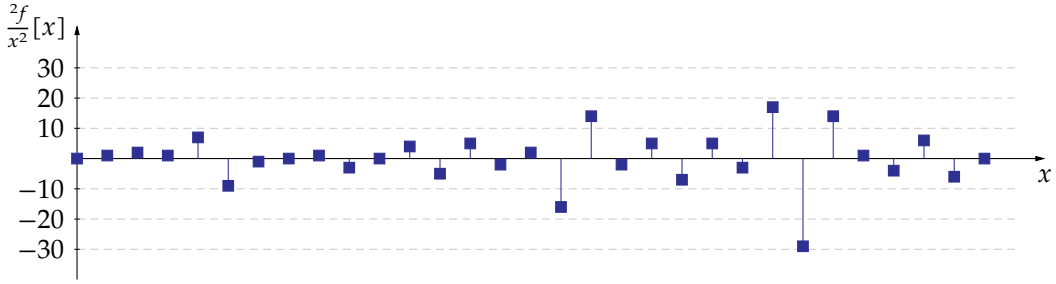
Now, let's add a little bit of noise to this line cut:



The first-order derivative looks like:



The second-order derivative looks like:



From these graphs, it's obvious that edge detection is not that simple in these cases. Noise impedes edge detection

- when using gradient information, and
- even more when using Laplacian information.

The core of the problem is that (discrete) derivatives are very sensitive to noise. We'd better take care of this before detecting edges.

Defective edges
In many cases, the edges that result contain holes, protuberances (dangling ends) or are incomplete.

We will treat holes and protuberances using a post-processing step. Sometimes it is possible to avoid holes or double edges in the edge detection algorithm itself.

8.4.3.2 DoG / LoG filtering

A common way to filter noise before calculating a gradient or a Laplacian, is to perform Gaussian filtering, i.e. filtering using a Gaussian kernel.

Gaussian filtering Given an image $f[x,y]$, one can suppress noise by applying a filter with a Gaussian kernel, i.e. the filtered image $g[x,y]$ is defined as

$$g[x,y] = G[x,y] \star f[x,y]$$

with

$$G[x,y] = \mathrm{e}^{-\frac{x^2+y^2}{2\sigma^2}}.$$

The advantage of this filter is that it

- is isotropic,
- has a limited support size in space and frequency, i.e. it will have limited artifacts,
- allows for a smooth combination with derivatives, leading to Derivative of Gaussian filters and Laplacian of Gaussian filters.

Derivative of Gaussian (DoG)
The process of calculating the Derivative of an image $f[x,y]$ that has been filtered with a

Gaussian, can be optimized:

$$\begin{aligned} h[x,y] &= \vec{\nabla}\left(G[x,y] \star f[x,y]\right) \\ &\downarrow \quad \text{associativity of linear operators} \\ &= \vec{\nabla} G[x,y] \star f[x,y] \end{aligned}$$

The gradient of the Gaussian can be elaborated further, as follows:

$$\begin{aligned} \vec{\nabla} G[x,y] &= \vec{e}_x \frac{G}{x}[x,y] + \vec{e}_y \frac{G}{y}[x,y] \\ &= \vec{e}_x \underbrace{\left(\mathrm{e}^{-\frac{x^2+y^2}{2\sigma^2}} \cdot \frac{-x}{\sigma^2} \right)}_{G'_x} + \vec{e}_y \underbrace{\left(\mathrm{e}^{-\frac{x^2+y^2}{2\sigma^2}} \cdot \frac{-y}{\sigma^2} \right)}_{G'_y} \\ &= G'_x \vec{e}_x + G'_y \vec{e}_y \end{aligned}$$

The kernels $G_x[x,y]$ and $G_y[x,y]$ can be precomputed and then applied in a single filtering operation (per direction) to calculate the Gaussian filtering and the gradient simultaneously. Usually, the kernel is considered to be zero at a distance 3σ from its center point.

The kernels have been illustrated in Figure 8.6.

Laplacian of Gaussian (LoG)

The process of calculating the Laplacian of an image $f[x,y]$ that has been filtered with a Gaussian, can be optimized:

$$\begin{aligned} h[x,y] &= \nabla^2\left(G[x,y] \star f[x,y]\right) \\ &\downarrow \quad \text{associativity of linear operators} \\ &= \nabla^2 G[x,y] \star f[x,y] \end{aligned}$$

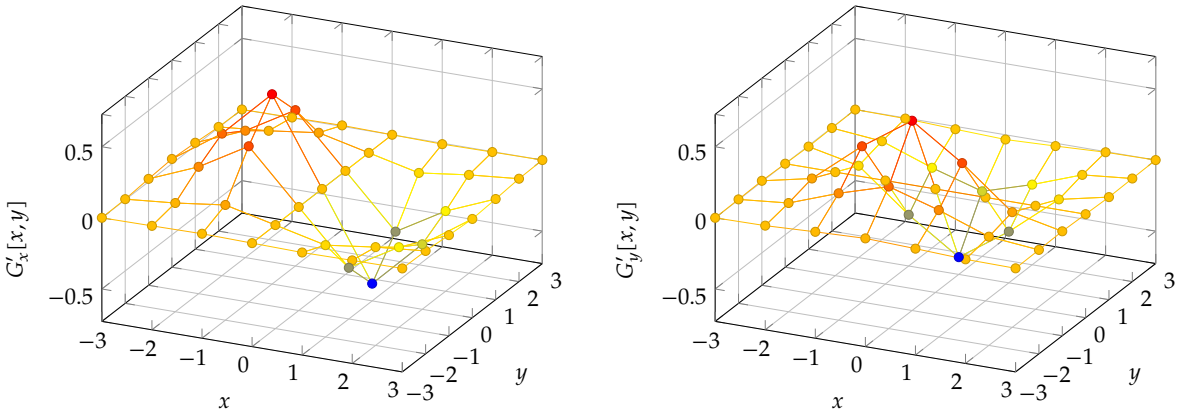
The Laplacian of the Gaussian can be further elaborated, as follows:

$$\begin{aligned} \nabla^2 G[x,y] &= \frac{{}^2G}{x^2}[x,y] + \frac{{}^2G}{y^2}[x,y] \\ &= \frac{}{x}\left(\mathrm{e}^{-\frac{x^2+y^2}{2\sigma^2}} \cdot \frac{-x}{\sigma^2} \right) + \frac{}{y}\left(\mathrm{e}^{-\frac{x^2+y^2}{2\sigma^2}} \cdot \frac{-y}{\sigma^2} \right) \\ &= \mathrm{e}^{-\frac{x^2+y^2}{2\sigma^2}} \cdot \frac{-x}{\sigma^2} \cdot \frac{-x}{\sigma^2} + \mathrm{e}^{-\frac{x^2+y^2}{2\sigma^2}} \cdot \frac{-1}{\sigma^2} + \mathrm{e}^{-\frac{x^2+y^2}{2\sigma^2}} \cdot \frac{-y}{\sigma^2} \cdot \frac{-y}{\sigma^2} + \mathrm{e}^{-\frac{x^2+y^2}{2\sigma^2}} \cdot \frac{-1}{\sigma^2} \\ &= \left(\frac{x^2+y^2-2\sigma^2}{\sigma^4} \right) \mathrm{e}^{-\frac{x^2+y^2}{2\sigma^2}} \end{aligned}$$

This kernel $\nabla^2 G[x,y]$ can be precomputed and then applied in a single filtering operation to calculate the Gaussian filtering and the Laplacian simultaneously. Usually, the kernel is considered to be zero at a distance 3σ from its center point.

The kernel has been illustrated in Figure 8.7.

Exercises



(a) G'_x **(b)** G'_y

Figure 8.6: Graphical representation of the DoG kernel, for $\sigma = 1$

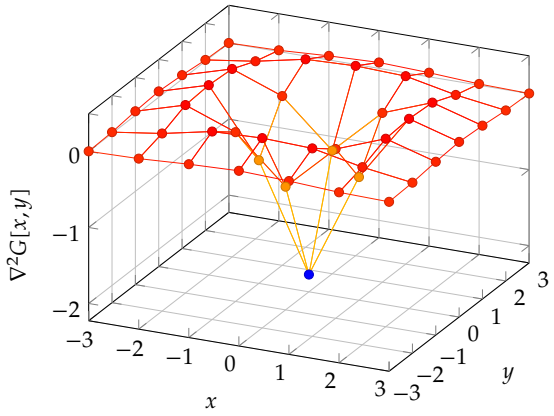


Figure 8.7: Graphical representation of the LoG kernel, for $\sigma = 1$

Exercise 8.4.3.2-1: Calculate the DoG kernel according to the x-axis and the y-axis, using Prewitt's gradient kernels. Take $\sigma = 1$.

Exercise 8.4.3.2-2: Calculate the LoG kernel, using the 4-Laplacian. Take $\sigma = 0.6$.

Exercise 8.4.3.2-3: Calculate the LoG kernel, using the 8-Laplacian. Take $\sigma = 0.75$.

8.4.3.3 The Marr-Hildreth algorithm

The Marr-Hildreth algorithm is Laplacian based. It uses LoG filtering to mitigate the influence of noise.

Algorithm: Marr-Hildreth edge detection

1. Filter the image $f[x,y]$ using a $n \times n$ filter with $n = 2\lceil 3\sigma \rceil + 1$ with a kernel that is obtained by sampling $G[x,y] = e^{-\frac{x^2+y^2}{2\sigma^2}}$
2. Calculate the Laplacian of the filtered image: $\nabla^2 \left(G[x,y] \star f[x,y]\right)$
3. Determine the zero crossings of the image using an appropriate zero crossing criterion (with threshold T), to obtain the resulting image $h[x,y]$

Remarks

- Often the Marr-Hildreth detector is called the *log-detector*.
- In practice, steps one and two are frequently bundled as elaborated in section 8.4.3.2. For reasons of clarity, we kept the two steps separated here.

Simple example To illustrate this algorithm, let's apply it to a simple example. Consider the 16×16 4-bit quantized test image of Figure 8.3 on page 224.

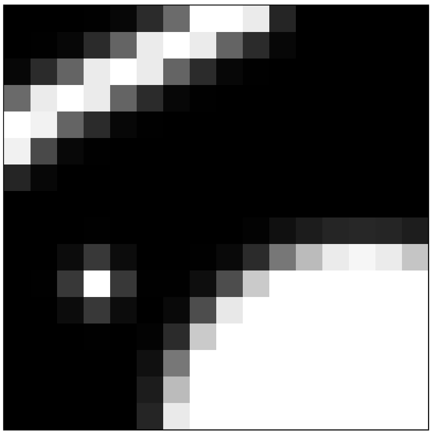
The Gaussian kernel (with $\sigma = 0.5745$, arbitrarily chosen) can be determined to be:

$$G[x,y] = \begin{bmatrix} 0.000 & 0.001 & 0.002 & 0.001 & 0.000 \\ 0.001 & 0.048 & 0.220 & 0.048 & 0.001 \\ 0.002 & 0.220 & 1.000 & 0.220 & 0.002 \\ 0.001 & 0.048 & 0.220 & 0.048 & 0.001 \\ 0.000 & 0.001 & 0.002 & 0.001 & 0.000 \end{bmatrix} \tag{8.1}$$

After Gaussian filtering, the image becomes:

$G[x,y] \star f[x,y]$ (rows: x, columns: y)

0.0	0.0	0.0	0.0	0.0	0.2	0.4	1.1	1.5	0.9	0.1	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.2	0.4	0.9	1.3	0.9	0.4	0.2	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.2	0.4	0.9	1.3	0.9	0.4	0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.4	0.9	1.3	0.9	0.4	0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1.5	0.9	0.4	0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.9	0.3	0.0	0.0	0.0	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	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.1	0.1	0.2	0.1	0.1
0.0	0.0	0.0	0.2	0.0	0.0	0.0	0.0	0.0	0.2	0.5	0.7	0.9	1.0	0.9	0.8
0.0	0.0	0.2	1.0	0.2	0.0	0.0	0.1	0.3	0.8	1.2	1.6	1.7	1.8	1.7	1.6
0.0	0.0	0.0	0.2	0.0	0.0	0.0	0.3	0.9	1.5	1.9	2.0	2.0	2.1	2.0	2.0
0.0	0.0	0.0	0.0	0.0	0.0	0.2	0.8	1.5	1.9	2.1	2.1	2.1	2.1	2.1	2.1
0.0	0.0	0.0	0.0	0.0	0.1	0.5	1.2	1.9	2.1	2.1	2.1	2.1	2.1	2.1	2.1
0.0	0.0	0.0	0.0	0.0	0.1	0.7	1.6	2.0	2.1	2.1	2.1	2.1	2.1	2.1	2.1
0.0	0.0	0.0	0.0	0.0	0.1	0.9	1.7	2.0	2.1	2.1	2.1	2.1	2.1	2.1	2.1



Applying the Laplacian kernel to this image yields:

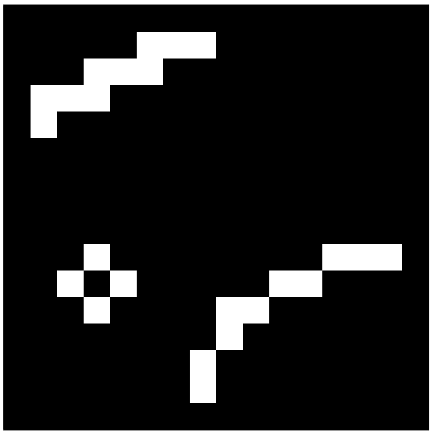
$\nabla^2 (G[x,y] \star f[x,y])$ (rows: x, columns: y)

0.0	0.0	0.2	0.6	1.6	2.3	2.7	-1.1	-5.2	-2.5	1.0	0.3	0.0	0.0	0.0	0.0
0.2	0.6	1.4	1.7	1.3	-2.5	-5.3	-2.1	1.7	1.7	1.0	0.2	0.0	0.0	0.0	0.0
1.7	1.7	1.3	-2.5	-5.5	-2.5	1.3	1.7	1.4	0.6	0.2	0.0	0.0	0.0	0.0	0.0
2.2	-2.2	-5.5	-2.5	1.3	1.7	1.4	0.6	0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0
-5.7	-1.8	1.4	1.7	1.4	0.6	0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
-2.1	1.7	1.6	0.6	0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1.2	1.2	0.3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.3	0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.2	0.3	0.4	0.4	0.4	0.4
0.0	0.1	0.3	0.3	0.3	0.0	0.0	0.0	0.2	0.6	1.0	1.5	1.7	1.9	1.8	1.8
0.0	0.3	1.1	-0.2	1.1	0.3	0.1	0.4	1.0	1.6	0.9	0.4	-0.2	-0.2	-0.1	0.8
0.0	0.3	-0.2	-6.9	-0.2	0.3	0.4	1.2	1.4	0.1	-0.8	-1.4	-1.6	-1.9	-1.5	-1.0
0.0	0.3	1.1	-0.2	1.1	0.5	1.0	1.4	-0.1	-1.3	-2.0	-1.4	-1.0	-0.9	-0.9	-0.9
0.0	0.0	0.3	0.3	0.3	0.7	1.6	0.1	-1.3	-1.6	-0.9	-0.3	-0.1	-0.1	-0.1	-0.2
0.0	0.0	0.0	0.0	0.2	1.0	0.9	-0.8	-2.0	-0.9	-0.2	-0.0	-0.0	-0.0	-0.0	-0.0
0.0	0.0	0.0	0.0	0.3	1.5	0.3	-1.4	-1.4	-0.3	-0.0	-0.0	-0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.4	1.7	-0.4	-1.7	-1.1	-0.2	-0.0	-0.0	0.0	0.0	0.0	0.0

After an N_4 zero crossing determination with $T = 0.52$, we obtain:

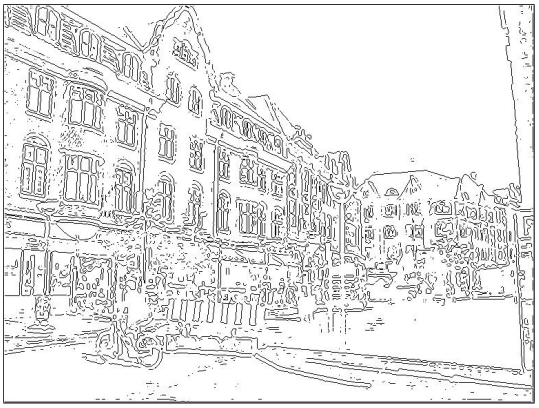
$h[x,y]$ (rows: x, columns: y)

0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	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	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	1.0	1.0	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	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	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0	0.0
0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	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	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	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	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



Real-world example Figure 8.8 illustrates the result of applying this technique on a real-world example image.

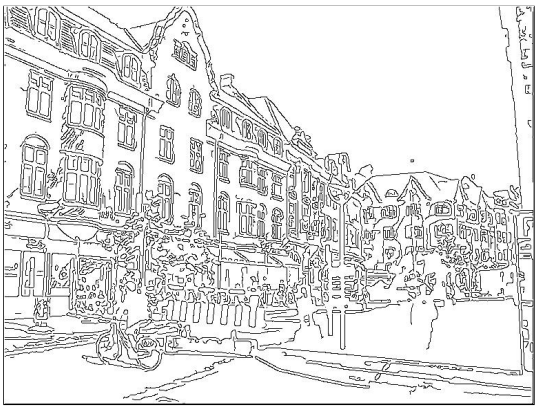
(a) Image of the town of Aalborg, Denmark (source: Walter Daems)



(b) Result after edge detection (white = background, black = foreground)

Figure 8.8: Illustration of the Marr-Hildreth edge detection algorithm on a real-world example, with $\sigma = 2$ and N_4 zero crossing detection with $T = 6.510 \times 10^{-3}$

(a) Image of the town of Aalborg, Denmark (source: Walter Daems)



(b) Result after edge detection (white = background, black = foreground)

Figure 8.9: Illustration of the Canny edge detection algorithm on a real-world example, with $\sigma = 1$ and $T_L = 65.3 \times 10^{-3}$ and $T_H = 0.140\,6$

Exercises
We will not make exercises on the Marr-Hildreth algorithm. Find a good library that implements the algorithm if you need it. Consider e.g. MATLAB/OCTAVE's `edge()` function.

8.4.3.4 The Canny algorithm

The Canny algorithm is gradient based. It uses DoG filtering to mitigate the influence of noise. It uses a special form of thinning to minimize the thickness of edges, and improves the quality of the resulting edge by promotion/demotion of (candidate) edge pixels.

Algorithm: Canny edge detection

1. Filter the image f using an $n \times n$ filter with $n = 2\lceil 3\sigma \rceil + 1$ with a kernel that is obtained by sampling $G[x,y] = e^{-\frac{x^2+y^2}{2\sigma^2}}$.
   Result: filtered image $g[x,y] = G[x,y] \star f[x,y]$.
2. Calculate the gradient of the filtered image: $\vec{\nabla} g$
3. Thin the gradient image by suppressing false maxima in the gradient-direction; this results in an image $(\vec{\nabla} g)_{thinned}$.
4. Transform the image $|(\vec{\nabla} g)_{thinned}|$ into a ternary image h using two threshold values T_L and T_H
   - background pixels: $|h[x,y]| < T_L$
   - candidate edge pixels: $T_L < |h[x,y]| < T_H$
   - edge pixels: $T_H < |h[x,y]|$
5. Promote the candidate edge pixels to edge pixels if there is a V-path to an edge pixel, demote the rest of the candidate edge pixels to background pixels. This results in the final image $h[x,y]$.

Remarks

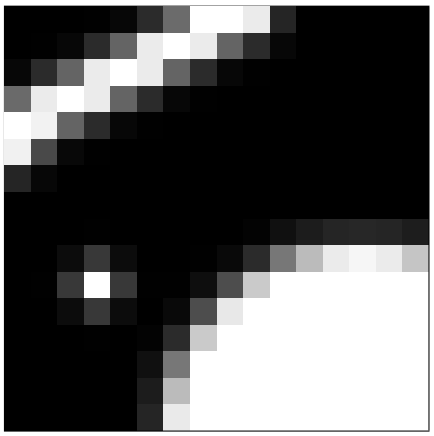
- Canny is the name of a person and therefore no mediocre adjective. The Canny edge detector is doing a very good job!
- In practice steps one and two are frequently bundled as elaborated in section 8.4.3.2. For reasons of clarity, we kept the two steps separated here.

Simple example To illustrate this algorithm, let's apply it to a simple example. Consider the 16×16 4-bit quantized test image of Figure 8.3 on page 224.

The Gaussian kernel (with $\sigma = 2/3$, arbitrarily chosen) can be found in (8.1). After Gaussian filtering, the image becomes (as calculated earlier):

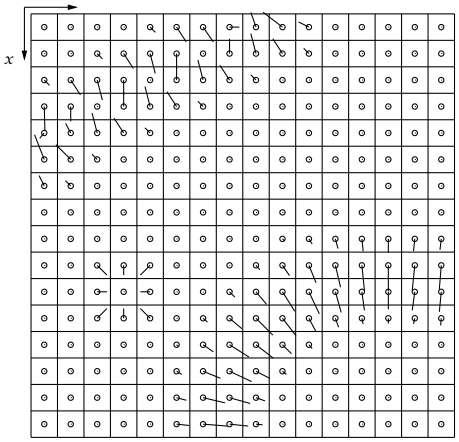
$G[x,y] \star f[x,y]$ (y →, x ↓)

0.0	0.0	0.0	0.0	0.0	0.2	0.4	1.1	1.5	0.9	0.1	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.2	0.4	0.9	1.3	0.9	0.4	0.2	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.2	0.4	0.9	1.3	0.9	0.4	0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.4	0.9	1.3	0.9	0.4	0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1.5	0.9	0.4	0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.9	0.3	0.0	0.0	0.0	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	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.1	0.1	0.2	0.1	0.1
0.0	0.0	0.0	0.2	0.0	0.0	0.0	0.0	0.0	0.2	0.5	0.7	0.9	1.0	0.9	0.8
0.0	0.0	0.2	1.0	0.2	0.0	0.0	0.1	0.3	0.8	1.2	1.6	1.7	1.8	1.7	1.6
0.0	0.0	0.0	0.2	0.0	0.0	0.0	0.3	0.9	1.5	1.9	2.0	2.0	2.1	2.0	2.0
0.0	0.0	0.0	0.0	0.0	0.0	0.2	0.8	1.5	1.9	2.1	2.1	2.1	2.1	2.1	2.1
0.0	0.0	0.0	0.0	0.0	0.1	0.5	1.2	1.9	2.1	2.1	2.1	2.1	2.1	2.1	2.1
0.0	0.0	0.0	0.0	0.0	0.1	0.7	1.6	2.0	2.1	2.1	2.1	2.1	2.1	2.1	2.1
0.0	0.0	0.0	0.0	0.0	0.1	0.9	1.7	2.0	2.1	2.1	2.1	2.1	2.1	2.1	2.1



Calculating the discrete gradient of this image yields:

$\vec{\nabla}(G[x,y] \star g[x,y])$ (y →, x ↓)

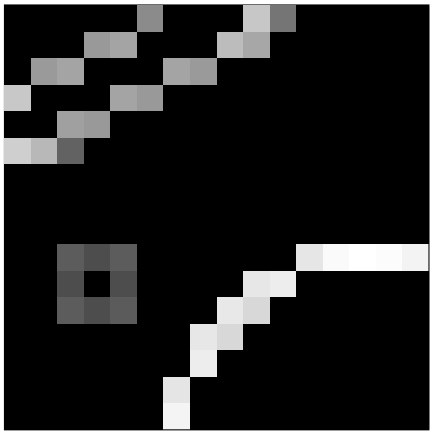


Then we perform step number 3, the suppression of false maxima.

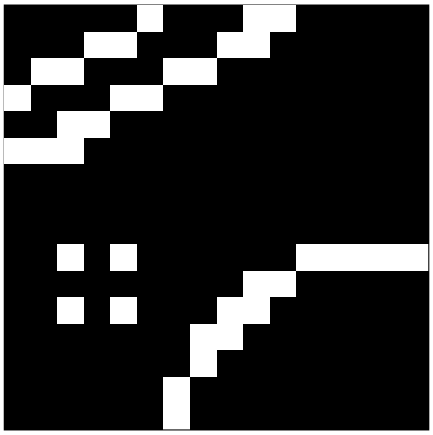
After zero crossing determination with $T = 0.52$, we obtain:

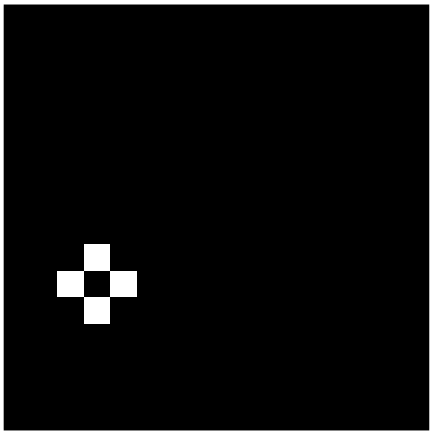
$h[x,y]$ (y →, x ↓)

0.0	0.0	0.0	0.0	0.0	2.6	0.0	0.0	0.0	3.7	2.2	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	2.9	3.1	0.0	0.0	0.0	3.5	3.1	0.0	0.0	0.0	0.0	0.0	0.0
0.0	2.9	3.1	0.0	0.0	0.0	3.1	2.9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3.7	0.0	0.0	0.0	3.1	2.9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	3.0	2.9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3.9	3.4	1.8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	1.7	1.4	1.7	0.0	0.0	0.0	0.0	0.0	0.0	4.3	4.7	4.8	4.7	4.6
0.0	0.0	1.4	0.0	1.4	0.0	0.0	0.0	0.0	4.3	4.4	0.0	0.0	0.0	0.0	0.0
0.0	0.0	1.7	1.4	1.7	0.0	0.0	0.0	4.3	4.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	4.3	4.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	4.4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	4.3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	4.6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0



Using a lower threshold of $T_L = 1.0$ and $T_H = 1.5$, we obtain the following edge pixels (on the left) and candidate edge pixels (on the right):

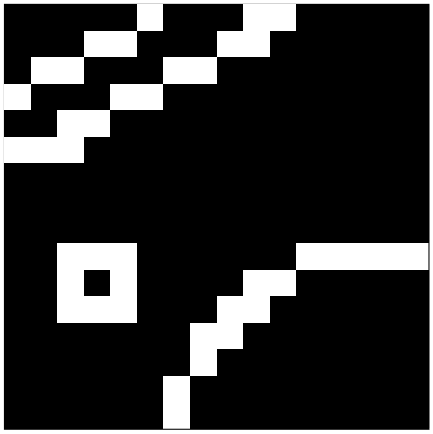




Promotion and demotion using 8-V-adjacency (8-V-paths) leads to the following result:

$h[x, y]$ y / x

0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	
0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	
0.0	1.0	1.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
1.0	0.0	0.0	0.0	1.0	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	1.0	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	
1.0	1.0	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	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
0.0	0.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0	1.0	1.0	
0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	
0.0	0.0	1.0	1.0	1.0	0.0	0.0	0.0	1.0	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	1.0	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	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	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	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	



Real-world example Figure 8.9 illustrates the result of applying this technique on a real-world example image.

Exercises

We will not make exercises on the Canny algorithm. Find a good library that implements the algorithm if you need it. Consider e.g. MATLAB/OCTAVE's `edge()` function.

8.4.4 Modeling the edges

Why creating models for the edges? In many cases the detected edges are not complete (missing pixels, i.e. false negatives) or overcomplete (pixels that are no edge pixels, i.e. false positives). Besides that, the multitude of pixels is not the most efficient way to describe an edge.

Deriving simpler models from the available edge pixels is not only more efficient from an information point of view, but will also solve the false negatives and positives problem.

We will treat three methods, a local one, a regional one and a global one. While the local method is not a very good one - we added it for completeness - the latter two are good

performers.

There are many criteria to compare the efficiency of edge modeling methods. Three good ones are:

- the simplicity of the model (the fewer model parameters, the better),
- the accuracy of the model,
- the computational effort to derive the model.

A good compromise between the three criteria is important.

8.4.4.1 Local analysis - edge extension

The basic idea is to consider all non-edge neighbors of edge pixels and consider them for promotion. The criterion for deciding on promotion is the similarity of the gradient.

Algorithm: Edge extension algorithm

1. Determine for the entire image $f[x,y]$, the magnitude $M[x,y]$ and the direction $\alpha[x,y]$ of the gradient.
2. For all edge pixels $[x,y]$:
   2.1. For all neighboring pixels $[u,v]$ of $[x,y]$ that aren't edge pixels:
      2.1.1. if $|M[u,v] - M[x,y]| < T_M$ and $|\alpha[u,v] - \alpha[x,y]| < T_\alpha$:
      mark $[u,v]$ as edge pixel

This algorithm is computationally very expensive and therefore not often used in practice. We will not investigate it further.

8.4.4.2 Regional analysis - piecewise linear edge modeling

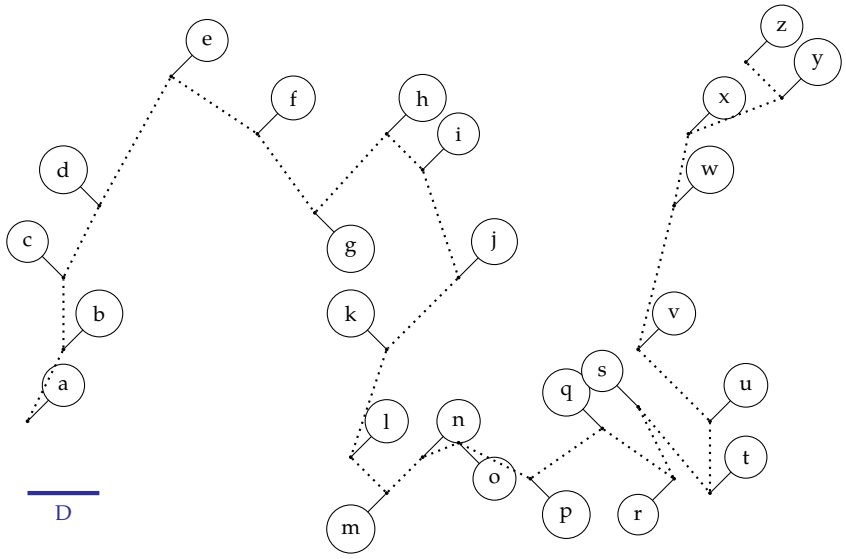
We will restrict ourselves to one particular regional method, i.e. creating a poly-linear model, based on an ordered set of edge pixels. A second prerequisite is that we need to know if the edge is a closed or an open path. If it is an open path, we can apply the algorithm straight away. If it a closed path, we need to split it into two open half paths.

Algorithm: Double stack algorithm for piecewise linear edge modeling

1. Make two stacks `TP` (to process) and `PD` (processed).
   We denote the top of the stack `X` by $t_{\texttt{X}}$. Choose a distance D.
2. Put the first point in the `TP` and the last point in `PD`
3. While the stack `TP` is not empty:
   3.1. Determine the line L through $t_{\texttt{TP}}$ and $t_{\texttt{PD}}$
   3.2. Determine the points that are ordered in between $t_{\texttt{TP}}$ and $t_{\texttt{PD}}$. Denote the point with the biggest distance $d(x,L)$ to the line L by x.
   3.3. If $d(x,L) > D$: place x on top of the stack `TP`
   otherwise: move $t_{\texttt{TP}}$ to the top of the stack `PD`.
4. The points of the poly-linear approximation are listed in the stack `PD`

Example

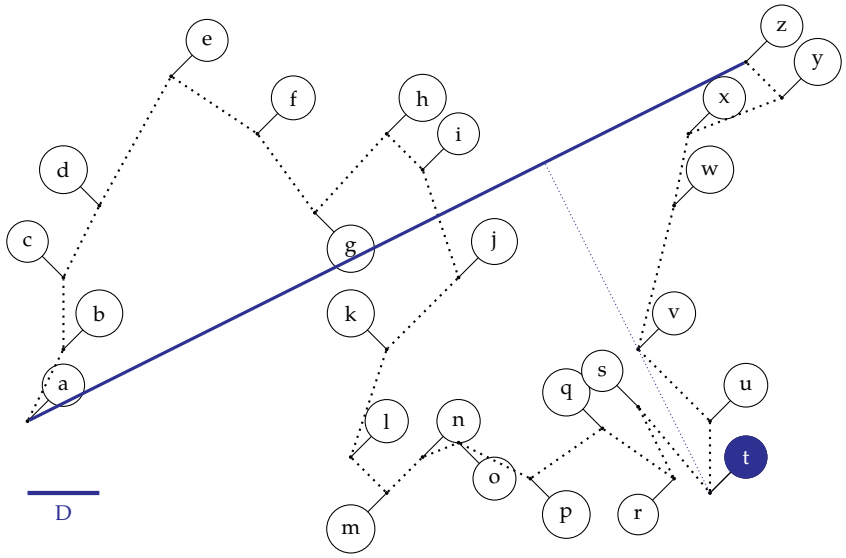
Consider the configuration of ordered edge pixels a to z below. Find a poly-linear approximation of the edge described by these pixels, guaranteeing that the maximal distance between any edge pixel and the curve is smaller than or equal to D.



Let's create two stacks according to step one and initialize them according to step two:

a	z
TP	PD

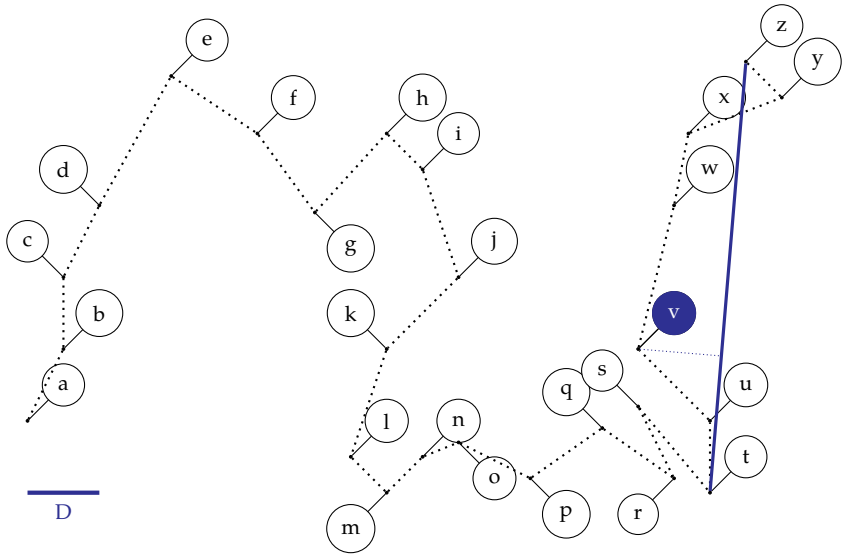
Then we proceed with step 3. Given the fact that the TP stack is not empty, we determine the line L joining the stack tops a and z, and determine t to be the most distant point in between.



Given the fact that the $d(L,t) > D$, we put the point t on the TP stack.

$$\begin{array}{cc} \multicolumn{2}{c}{\text{t}} \\ \text{a} & \text{z} \\ \hline \text{TP} & \text{PD} \end{array}$$

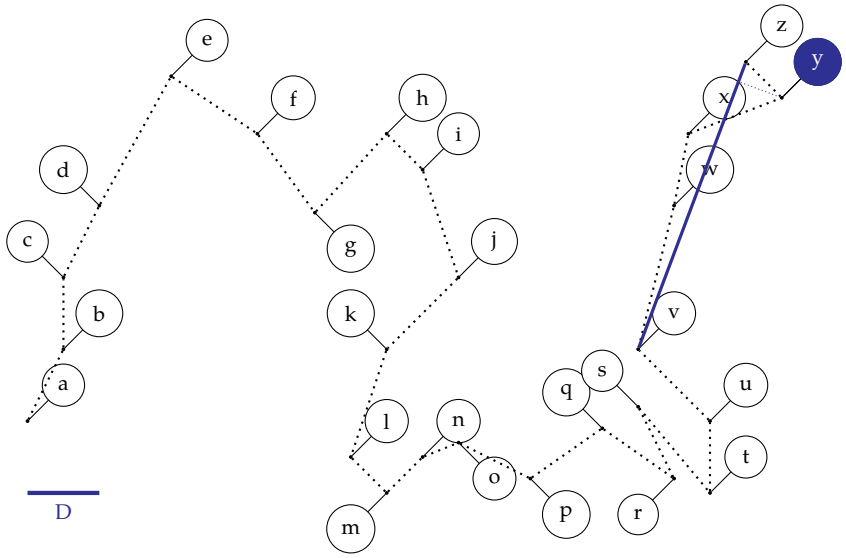
Given the fact that the TP stack is not empty, we determine the line L joining the stack tops t and z, and determine v to be the most distant point in between.



As $d(L, v) > D$, we put the point v on the TP stack.

$$\begin{array}{cc} \multicolumn{2}{c}{\text{v}} \\ \multicolumn{2}{c}{\text{t}} \\ \text{a} & \text{z} \\ \hline \text{TP} & \text{PD} \end{array}$$

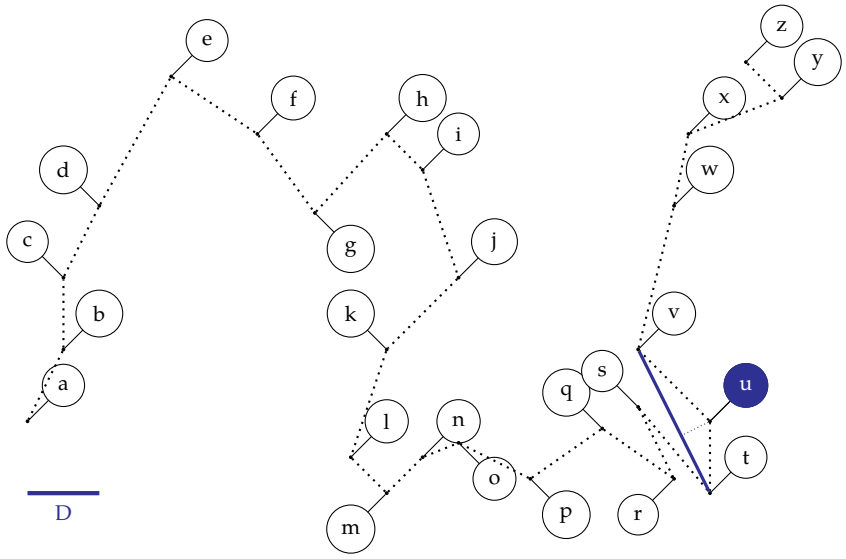
Given the fact that the TP stack is not empty, we determine the line L joining the stack tops v and z, and determine y to be the most distant point in between.



As $d(L, y) \leq D$, we discard the point y and move the top of the TP stack (v) to the PD stack, i.e. the line joining v and z is a good approximation for the edge pixels v until z.

t	v
a	z
TP	PD

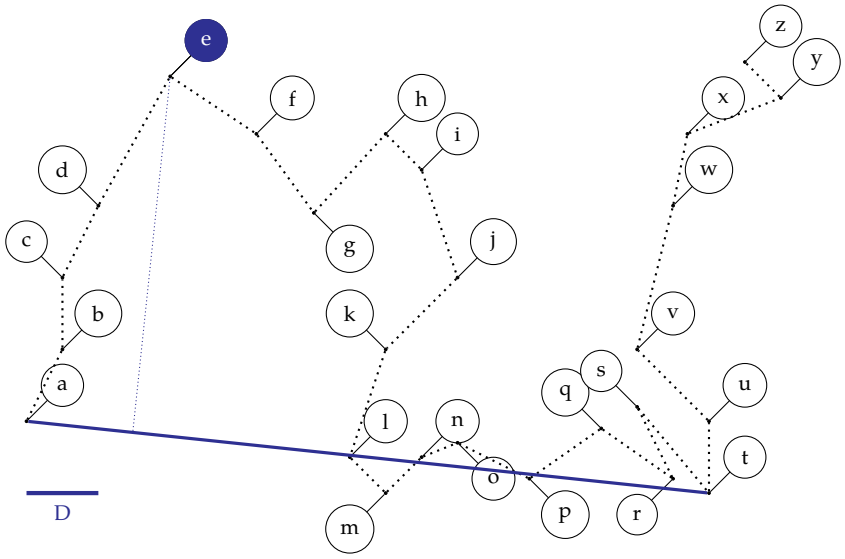
Given the fact that the TP stack is not empty, we determine the line L joining the stack tops t and v, and determine u to be the most distant point in between.



As $d(L, u) \leq D$, we discard the point u and move the top of the TP stack (t) to the PD stack, i.e. the line joining t and v is a good approximation for the edge pixels t until v.

	t
	v
a	z
TP	PD

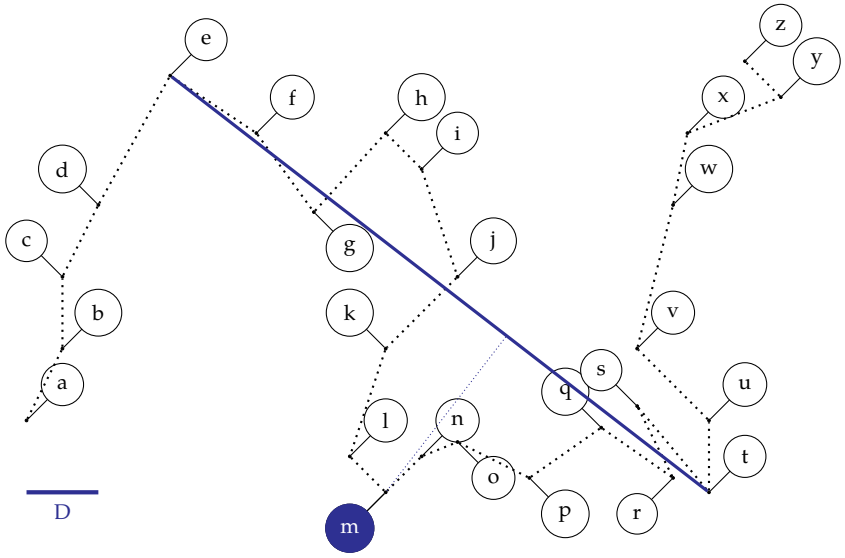
Given the fact that the TP stack is not empty, we determine the line L joining the stack tops a and t, and determine e to be the most distant point in between.



As $d(L, e) > D$, we put the point e on the TP stack.

	t
e	v
a	z
TP	PD

Given the fact that the TP stack is not empty, we determine the line L joining the stack tops e and t, and determine m to be the most distant point in between.



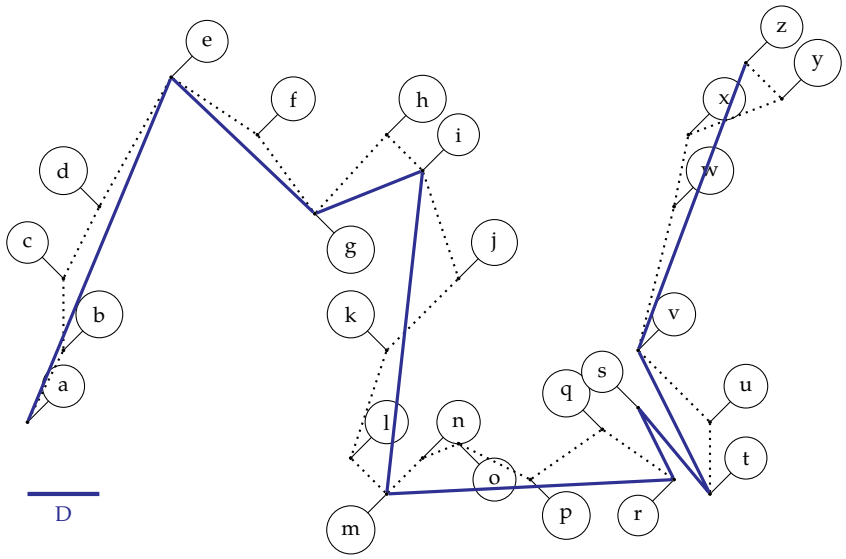
As $d(L, m) > D$, we put the point m on the TP stack.

m	t
e	v
a	z
TP	PD

We can continue this process (please, try this yourself!) until we obtain the following stack image:

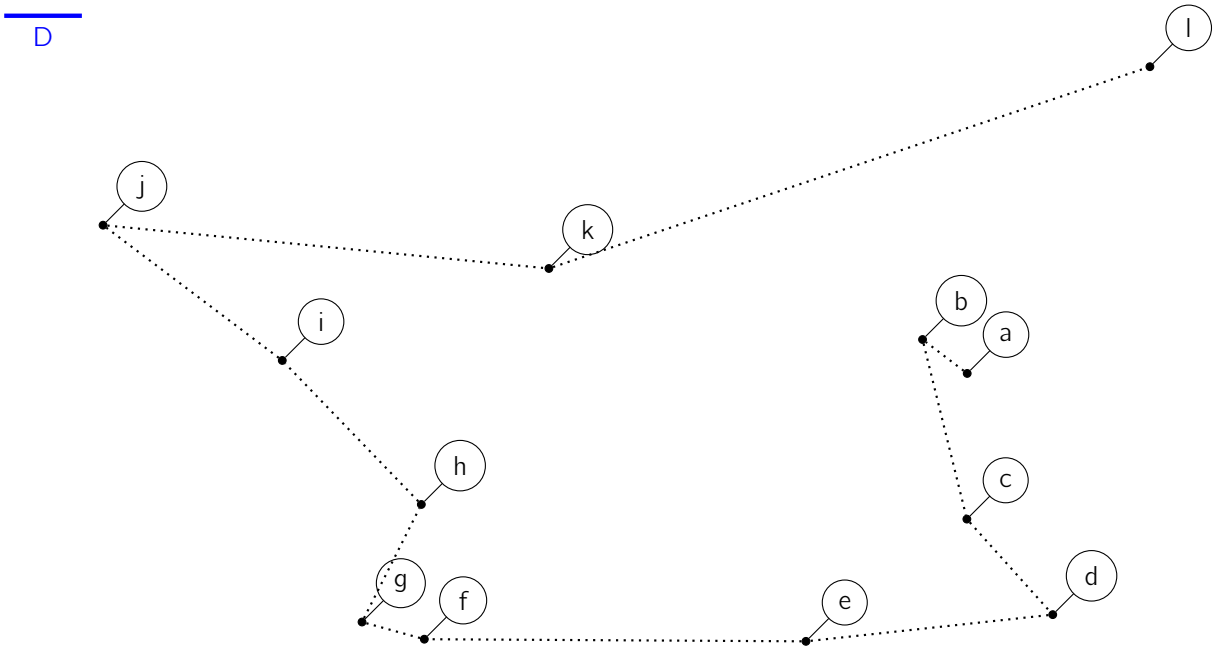
TP	PD
	a
	e
	g
	i
	m
	r
	s
	t
	v
	z

Now the TP stack is empty and the algorithm stops. The PD stack shows the corner points of the best piecewise linear approximation, given a desired accuracy D. This result has been shown below:

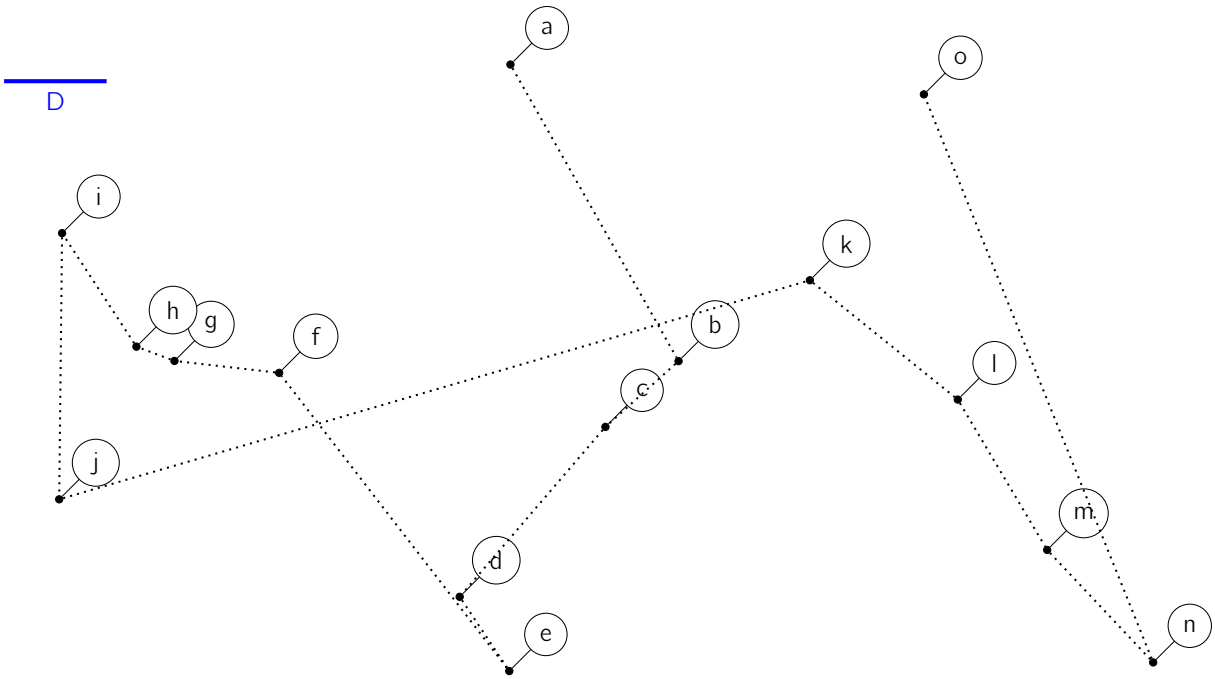


Exercises

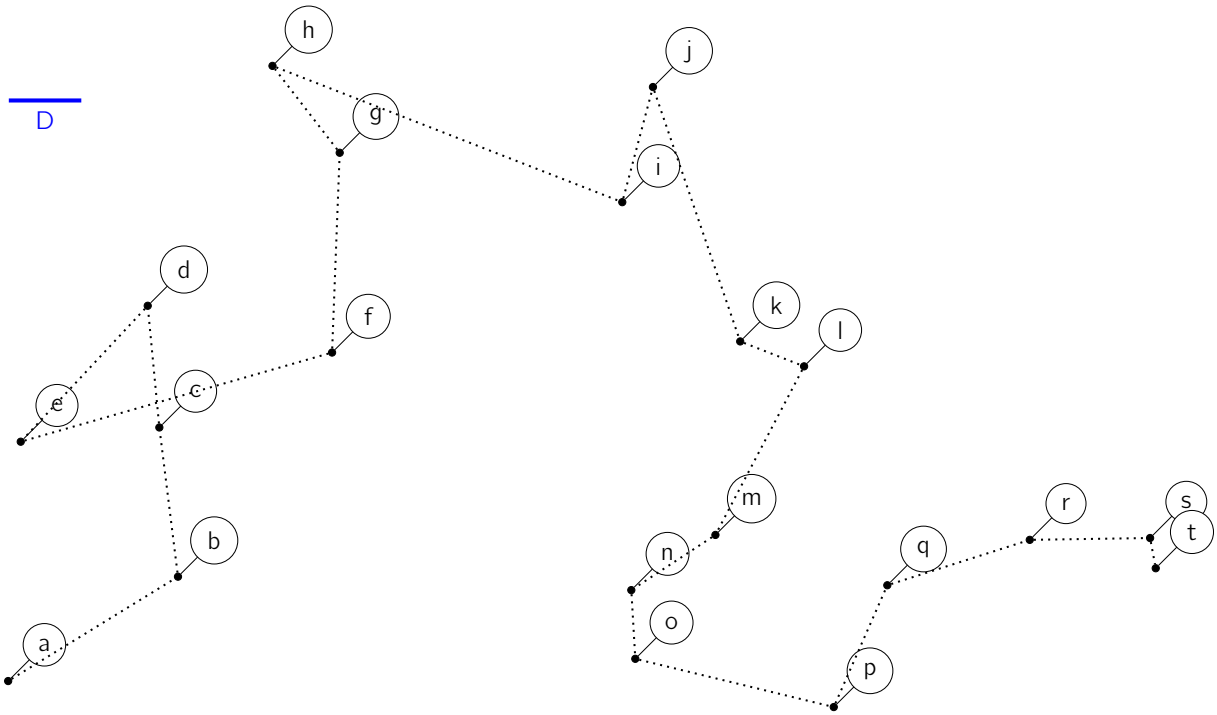
Exercise 8.4.4.2-1: Find the best polylinear approximation of the ordered set of points a to l, using a critical distance D, indicated below.



Exercise 8.4.4.2-2: Find the best polylinear approximation of the ordered set of points *a* to *o*, using a critical distance *D*, indicated below.



Exercise 8.4.4.2-3: Find the best polylinear approximation of the ordered set of points *a* to *t*, using a critical distance *D*, indicated below.



8.4.4.3 Global analysis - Hough transformation

The Hough transformation is a global edge modeling algorithm. In its simplest form, it finds edge points that are collinear. The points don't need to be ordered. In fact, the Hough transformation is a more general transformation that allows recognizing predefined shapes in a set of points. For more complex shapes the data structure used in the algorithm quickly becomes too large to fit in a modern computer. In that case special tricks to reduce the amount of memory required (not described in this text) can save the day.

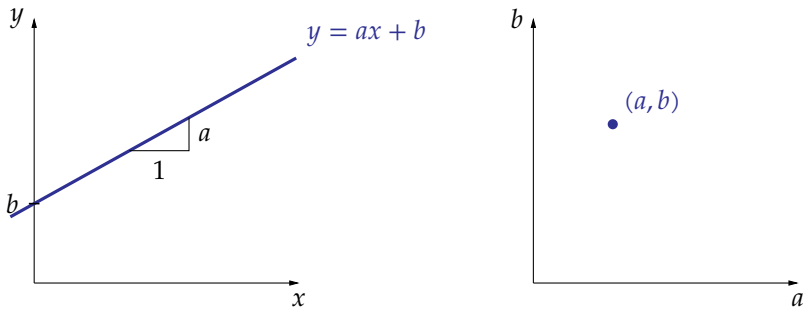
We will just treat the technique for finding collinear points.

Principle

Consider a line in a two dimensional Cartesian space, the xy-space or so-called *data space*. Such a line can be described using two parameters a and b:

$$y = ax + b$$

The parameter a denotes the slope, b denotes the y-axis intercept point (as indicated below on the left). The same line can be described in the ab-space or so-called *parameter space*. In this space, the line is described by a single point (a, b) (as indicated below on the right).



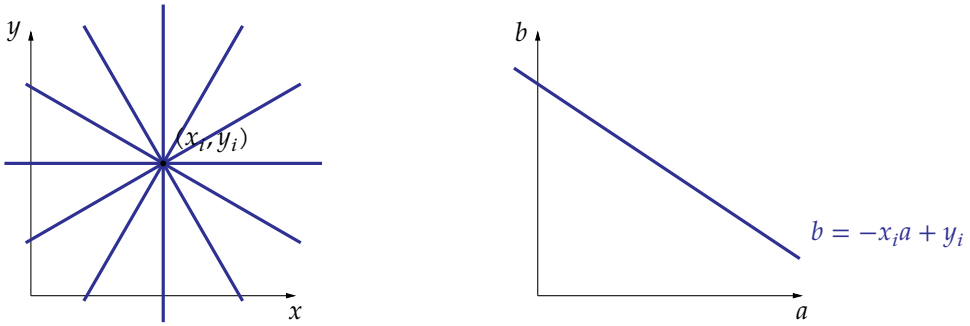
Now, let's generalize the problem and consider all lines that contain a specific point of the data space (x_i, y_i). The parameters a and b of these lines must obey

$$y_i = ax_i + b.$$

We can write this explicitly, by considering b as a function of a, and x_i and y_i to be the parameters:

$$b = -x_i a + y_i$$

Graphically, in the data space (on the left, below) the set of lines going through (x_i, y_i) is represented by a bundle of lines through (x_i, y_i). In the parameter space (on the right below) this corresponds to a single line defined by the 'parameters' x_i and y_i.

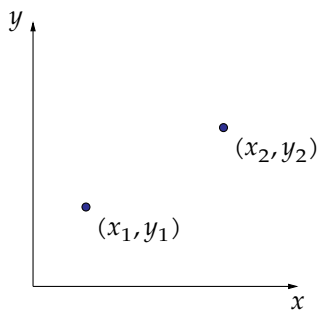


This mapping of the data space onto the parameter space is the so-called *Hough transformation*.

Finding the line that contains two points

Searching for line patterns turns out to be easier in the parameter space than in the data space. That is why the Hough transformation is such a good tool for this purpose.

Let's start by considering only two points (x_1, y_1) and (x_2, y_2). Our goal is to find the line $y = a_{12}x + b_{12}$ that contains these two points.



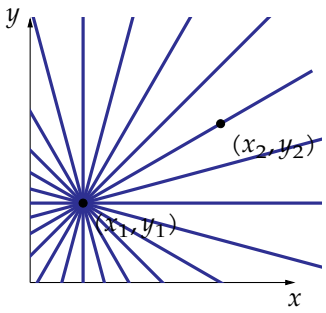
Obviously, in the data space, this is still a manageable job. Nevertheless, let's solve it using the Hough transformation.

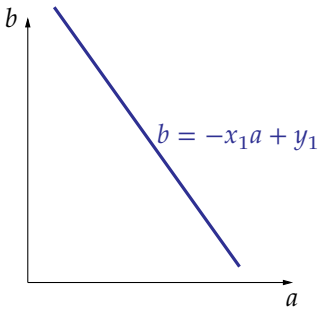
The procedure to follow is:

Algorithm: Finding the line that contains two points (using the Hough transformation)
Consider in the parameter space (using the Hough transformation):

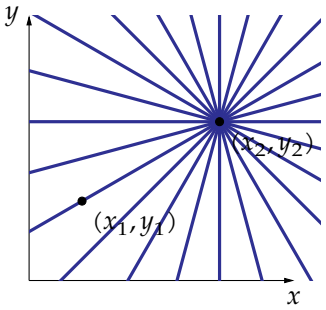
1. all lines containing (x_1, y_1)
2. all lines containing (x_2, y_2)
3. the common point(s) of these two sets

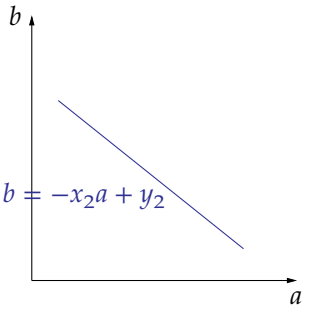
The set of lines of step 1 have been indicated in the data space on the left below. Its parameter representation can be found on the right below.



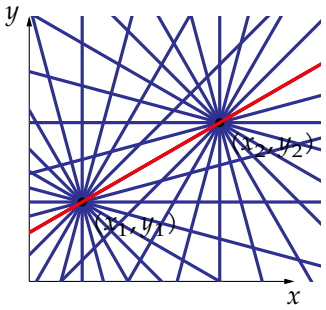


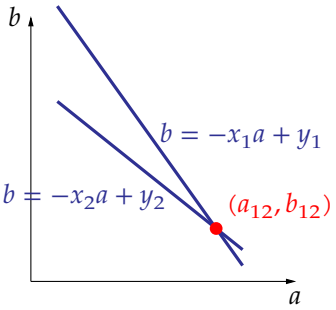
The set of lines of step 2 have been similarly indicated below:





To execute step 3, both sets have been indicated below and their common points have been indicated as well. In the data space, it is the line that contains both (x_1, y_1) and (x_2, y_2). In the parameter space, it is the intersection of the two lines, describing both line bundles.





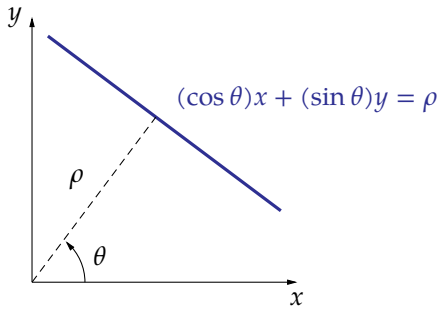
The problems of Descartes

Cartesian coordinates have two big problems:

1. for vertical lines the slope parameter and the intercept point become infinite,
2. related to this, but put more mildly, we need to consider an infinitely large range of a and b values to describe all possible lines in an image (that only has a limited domain).

However, this is easily solved by switching to polar coordinates.

In that representation, we describe a line by its angle w.r.t. the x-axis and its distance from the origin:



Analytically this can be summarized in one equation:

$$(\cos\theta)x + (\sin\theta)y = \rho \tag{8.2}$$

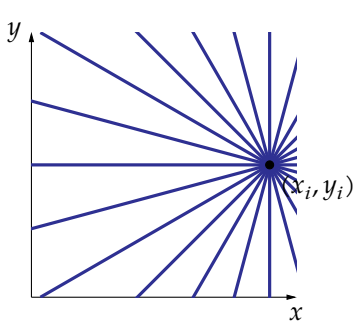
This avoids infinite parameter values, but it also allows representing all possible lines using a limited parameter range. We can represent any arbitrary line by $\theta \in [-\pi/2, \pi/2]$ and $\rho \in [-\rho_{\max}, +\rho_{\max}]$ with $2\rho_{\max}$ the Euclidean length of the image diagonal.

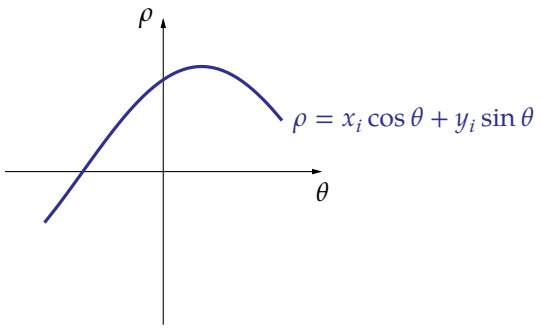
The Hough transformation using polar coordinates

Writing (8.2) to allow calculating ρ as a function of θ, yields:

$$\rho = x\cos\theta + y\sin\theta$$

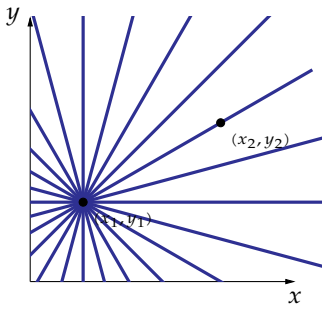
This equation reveals that lines in data space containing (x_i, y_i), are represented in the parameter space by sinusoids.

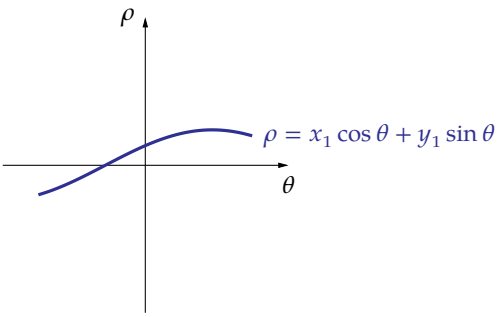




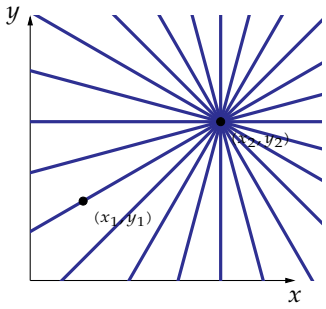
Applied to the problem of finding collinear points, this yields the following (using the same algorithm as for the Cartesian case).

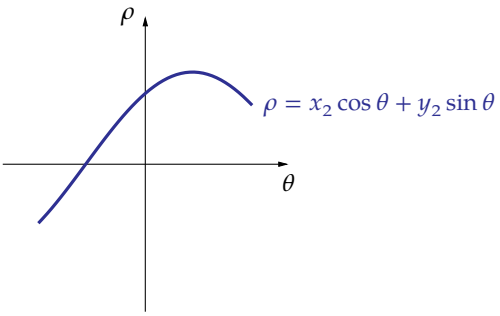
The set of lines of step 1 have been indicated in the data space on the left below. Its parameter representation can be found on the right below.



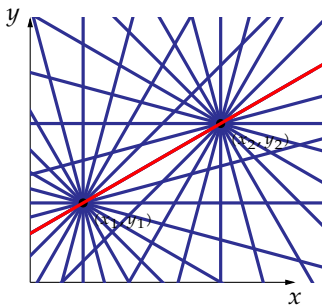


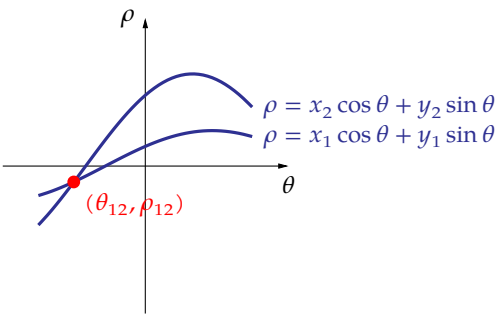
Similarly, the set of lines of step 2 have been indicated below:





To execute step 3, both sets have been indicated below and their common points have been indicated as well. In the data space, it is the line that contains both (x_1, y_1) and (x_2, y_2). In the parameter space, it is the intersection of the two lines, describing both line bundles.





The keystone: the accumulator-cell algorithm

A final issue remains, how do we use the Hough transformation in practice? To this end, we need to transform the continuous transformation to a discrete one. During this conversion, we will also solve the mystery of how to treat more than two points.

The main idea is to discretize the parameter space. This leads to the accumulator-cell algorithm.

Algorithm: Accumulator-cell algorithm

1. Discretize the θ, ρ-plane using $M \times N$ sample points (θ_i, ρ_j); Make an $M \times N$ matrix `AC` of integers (the accumulator cell matrix) and initialize it to [0].
2. For every edge pixel $[x_k, y_k]$:
   2.1. For every θ_i:
      2.1.1. calculate the rounded ρ_j value, i.e. $\rho_j \approx x_k \cos\theta_i + y_k \sin\theta_i$
      2.1.2. increment `AC`$[i, j]$
3. The accumulator cells `AC`$[i, j]$ that contain big values represent parameters (θ_i, ρ_j) of lines containing many edge pixels.

Example

Let's apply this algorithm to the image of Figure 8.10a. The first step is to determine the edge pixels using the Canny edge detection algorithm. This has been done in Figure 8.10b. The key question is now: on which lines do we find most pixels? The accumulator-cell algorithm can find them for us. It counts per cell how many points there are having a line bundle parameter representation that passes through the cell. The result has been displayed in Figure 8.10c. The whiter the cell, the more sinusoidal parameter curves are passing through it. Selecting the cells with the largest count (indicated with small squares on the image), results in a few lines that contain a lot of points in the image. They have been indicated in Figure 8.10d.

Exercises

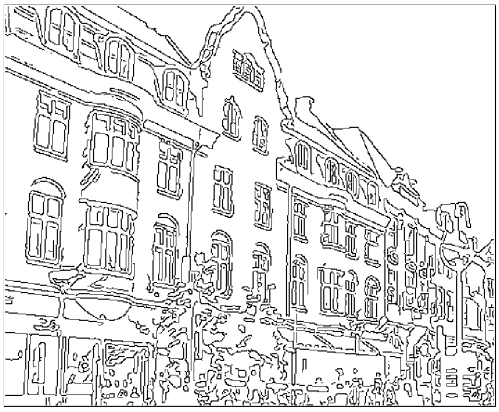
Exercise 8.4.4.3-1: Find the approximate line that contains as many of the points of the list below as possible. Do this using the Hough transformation and the accumulator-cell algorithm, using as values for $\theta = -\pi/2, -3\pi/8, -\pi/4, -\pi/8, 0, \pi/8, \pi/4, 3\pi/8, \pi/2$ and as values for $\rho = -3, -2, -1, 0, 1, 2, 3$.

i	$p_i = (x_i, y_i)$
1	$(-1, 1)$
2	$(-2, 0)$
3	$(2, 0)$
4	$(0, -1)$
5	$(0, 2)$

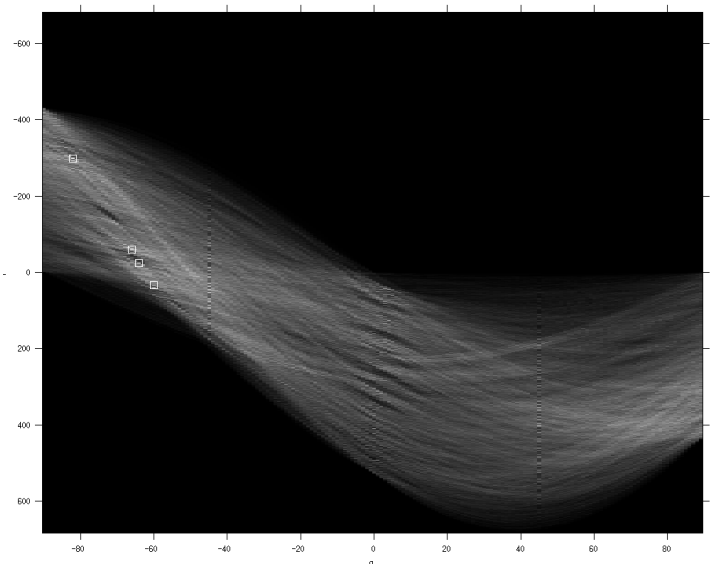
Make a Cartesian drawing of the points and the resulting line that contains most of the points.

Exercise 8.4.4.3-2: Find the approximate lines that contains as many of the points of the list below as possible. Do this using the Hough transformation and the accumulator-cell algorithm, using as values for
$\theta = -\pi/2, -2\pi/5, -3\pi/10, -\pi/5, -\pi/10, 0, \pi/10, \pi/5, 3\pi/10, 2\pi/5, \pi/2$ and as values for
$\rho = -3, -2.25, -1.5, -0.75, 0, 0.75, 1.5, 2.25, 3$.

(a) Image of the town of Aalborg, Denmark (source: Walter Daems)



(b) Edge pixels derived using the Canny edge detection algorithm (white = background, black = foreground)



(c) Accumulator-cell matrix

(d) Detected lines indicated on the original image

Figure 8.10: Illustration of the accumulator-cell algorithm on a grayscale test image

i	$p_i = (x_i, y_i)$
1	$(-1, -1)$
2	$(0, -1.3)$
3	$(1, -1.6)$
4	$(1, 1)$
5	$(2, 2.5)$

Make a Cartesian drawing of the points and the resulting line that contains most of the points.

Exercise 8.4.4.3-3: Find the approximate lines that contains as many of the points of the list below as possible. Do this using the Hough transformation and the accumulator-cell algorithm, using 7 appropriate values for θ and 9 appropriate values for ρ.

i	$p_i = (x_i, y_i)$
1	$(-3, 2.5)$
2	$(1, 1.5)$
3	$(-1, -3)$
4	$(0.85, -0.85)$
5	$(2, 1)$

Make a Cartesian drawing of the points and the resulting line that contains most of the points.

8.5 Region-based segmentation

So far, we've segmented images using edge detection. Once we have found a closed edge path, we have found a particular segment of the image. Now, let's reverse the strategy and try to find segments by starting with markers indicating the regions of interest and growing those markers into full segments.

You will notice that our knowledge of morphological operations is of good use now. If you have no clue about what morphological operations are, you'd better review Chapter 7.

8.5.1 Region growing

Principle

We'd like to group pixels that

- fulfill a specific criterion (a 'predicate' P), and
- are connected (with an 8-P-path, with P the predicate).

Predicate

The predicate is a function $P[x, y]$ that (based on the pixel's value and the values of the neighboring pixels), results in a binary output:

$$P[x,y] = \begin{cases} 1 & \text{if the pixel fulfills the criterion} \\ 0 & \text{otherwise} \end{cases}$$

Region growing is very related to 'reconstruction by dilation' and 'opening by reconstruction' (see Chapter 7).

Algorithms

Consider the following algorithm. It shows the basic principle of region growing. Many variants are possible.

Algorithm: Region growing

1. Choose a marker image X with default pixel value 0 and a pixel value of 1 for spots that mark the regions of interest.
2. Optional: erode the marker-image until the spots of the marker image are reduced to single pixel marks and label these remaining pixels with incremental integer values.
3. Grow the marker image with 8-P-adjacent pixels, where
   - P is the predicate;
   - the grown pixels inherit the label of the marker pixel.

   If two growing regions touch, then they are joined.
4. If the growth saturates, the segmentation is complete.

A variant of this algorithm can be found below:

Algorithm: Region growing (variant)

1. Derive a marker image $X = P_X(f)$ from the original image f using a predicate P_X with a default pixel value of 0 and a pixel value of 1 for the regions of interest.
2. Derive a mask image $M = P_M(f)$ from the original image f using a predicate P_M that envelopes the segments we are looking for.
3. Execute reconstruction by dilation, using the marker X and the mask M:

   $$g = R_{D,M}(X) = D_M^{(k)}(X)$$

   with k such that $D_M^{(k+1)}(X) = D_M^{(k)}(X)$.
   The resulting g is the segmented image.

Example

We will illustrate region growing using the image of Figure 8.11, in which we'd like to determine the segment containing the gravel path.

First, we will show that segmentation, based on thresholding does not yield a satisfactory result. An attempt using this approach has been made below. The image on the left shows the

Figure 8.11: Image of a gravel path (source: E. Dronkert, Creative Commons 2.0)

image in a grayscale version, using:

$$I = \frac{R + G + B}{3}$$

with R, G and B all in the range 0 to 1. The image on the right shows the result after thresholding using

$$f[x, y] = \begin{cases} 1 & \text{if } 0.55 < I < 0.66 \\ 0 & \text{otherwise} \end{cases}$$

and then is inverted. The inversion was added to save (black) ink. The result is, mildly put, not satisfactory, we detected a lot of features that do not belong to the gravel path.

Second, let's try to apply the principle of region growing. It's not uncommon to apply a preconditioning step before starting the region growing process. We will do so with the goal of stressing the green colors in the image, because we definitely know that the gravel path is not green. The transformation that has been used is:

$$I = G - R - B$$

The result can be found below:

Before we proceed, let's define two new auxiliary functions:

- $m[x, y]$ is the median of a 15×15 window around $[x, y]$.
- $\sigma^2[x, y]$ is the variance of a 15×15 window around $[x, y]$.

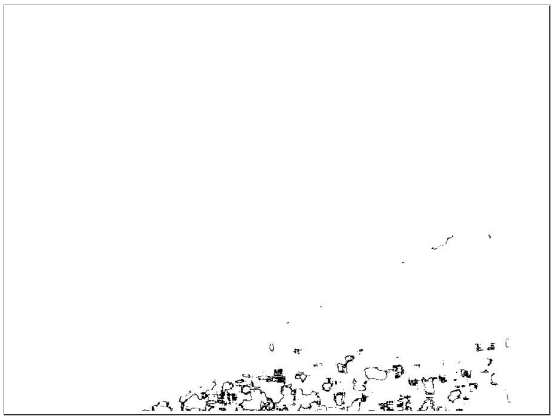
We will use the variant algorithm. Therefore, we need to define predicates for the marker and the mask. We have chosen the predicate for the mask to be:

$$P_M[x, y] = \begin{cases} 1 & \text{if } m[x, y] < 0.65 \\ 0 & \text{otherwise} \end{cases}$$

We have chosen the predicate for the marker to be:

$$P_X[x, y] = \begin{cases} 1 & \text{if } 0.25 < m[x, y] < 0.65 \text{ and } 0.1 < \sigma^2[x, y] < 0.11 \\ 0 & \text{otherwise} \end{cases}$$

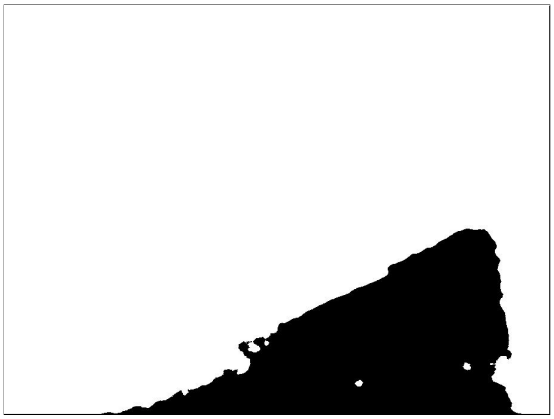
Executing step 1, yields the following marker image X on the left below. Executing step 2, yields the mask image M on the right below. We inverted the images in order to save ink.



Next, we execute step 3, i.e. we reconstruct by dilation

$$g = R_{D,M}(X)$$

to obtain the following result. Again, we inverted the image to save ink.



A justified comment to the example above, is that quite a lot of constants were chosen. In a real application, the determination of these constants will have to be automated. This is the subject we will not treat in this course.

Exercises
We will not make exercises on region growing algorithms. Find a good library that implements some if you need them.

8.5.2 Morphological watershed segmentation

As a final example of region-based segmentation we will discuss the morphological watershed segmentation. Before confronting you with the algorithm, let's illustrate the principle.

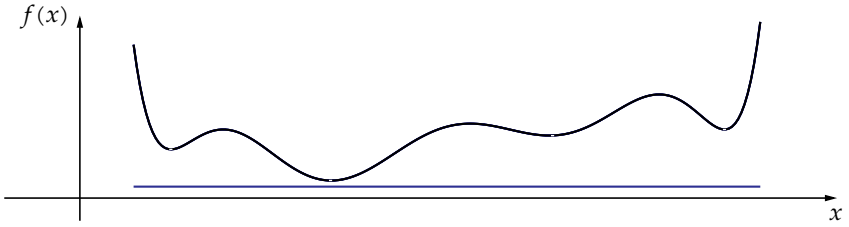
Principle
The basic idea is to consider the image $f[x, y]$ to be a landscape. The gray-scale value of a pixel indicates the height of the landscape. We'll assume the soil not to be permeable, i.e. if it rains, every rain drop rolls towards the local valley in the landscape. For every point in the landscape it is clear to which valley the drop will roll, except for the points on the *watershed lines*. The goal is to determine these watershed lines. Indeed, these are the lines that segment the individual basins.

Now, execute the following steps:

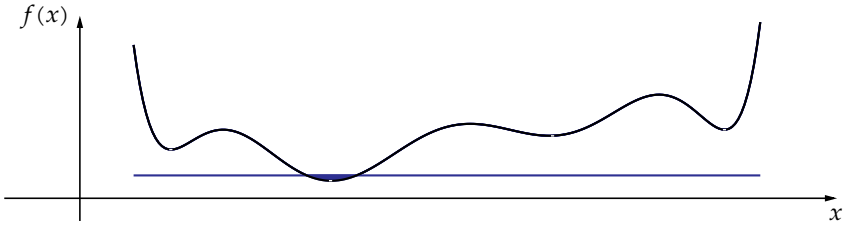
1. Drill a hole in the minimum of each basin such that the ground water can flow through.
2. Then, raise the ground water level gradually. At the lowest minima, small pools will arise that grow as the ground water level increases.
3. If two basins are about to flow together, build a dam in between them.

This has been illustrated in Figure 8.12 on a one-dimensional intensity function $f[x, y]$.

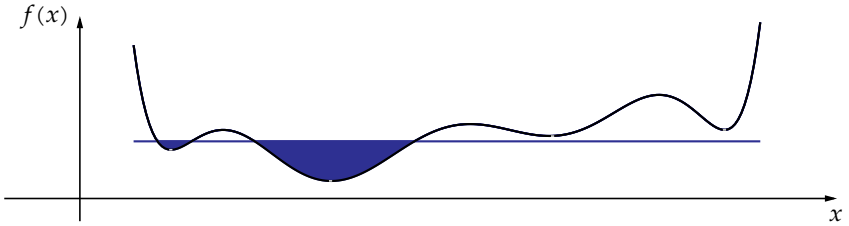
Pro: the dams will form closed loops such that no post-processing steps are required.



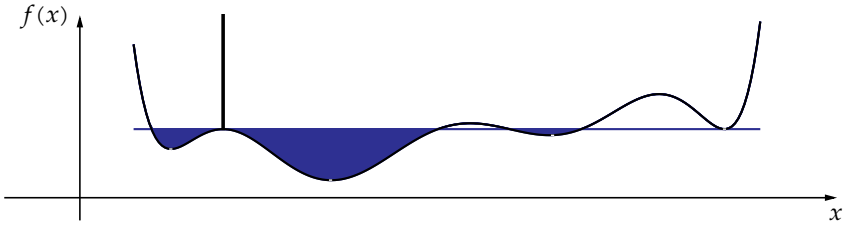
(a) Original intensity function; holes drilled in the basins; water level below lowest point.



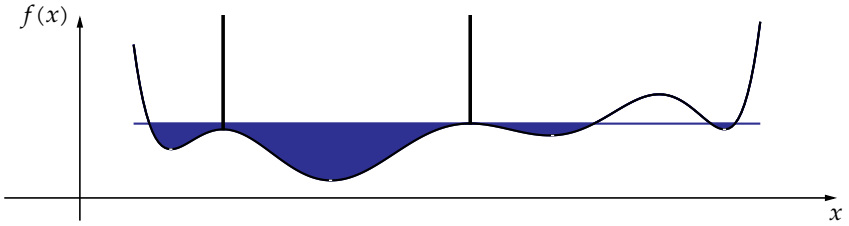
(b) The water level rises; the first basin fills itself.



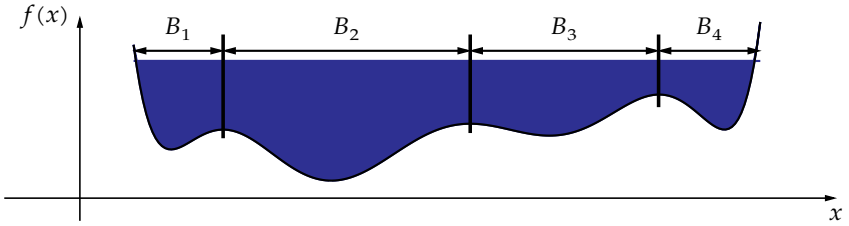
(c) Water level keeps rising; the second basin fill itself.



(d) Third basin fills itself; The first basins are about to flow together. Therefore, build a dam.



(e) The third basin fills itself; two more basins are about to flow together, therefore, build another dam.



(f) Final view, with four recognized segments B_1 to B_4

Figure 8.12: Illustration of the principle of watershed segmentation on a one-dimensional intensity function

Contra: too many local minima will result in oversegmentation; two possible solutions to this:

- remove the local minima by averaging or consecutive morphological openings and closings
- select the water holes manually and wisely (manually placed markers)

Algorithm

The algorithm is not that complicated, but is computationally quite intensive.

Algorithm: Morphological Watershed segmentation

1. Initialize all basins B_i by populating them with their local minima
   $B_i = \{[x,y] | f[x,y]$ is a minimum belonging to basin $i\}$
2. For all intensity levels $l \in 0, 1, 2, \ldots, L-1$:
   2.1. Determine $T_l = \{[x,y] \mid f[x,y] \geq l\}$
   2.2. While regions grow:
      2.2.1. For every basin B_i:
         2.2.1.1. $B_i = D_{T_l}^{(1)}(B_i)$
         2.2.1.2. If the resulting B_i overlaps with another basin B_j then let $f[x,y] = L$ for the overlapping pixels
3. The pixels with intensity values L are the segment boundaries.

Remarks

- Sometimes the watershed segmentation is applied to a gradient image; in this way, the edges will become the mountains and the regions of near-constant intensity will become the valleys. Watershed segmentation thus results in edge detection.
- The watershed segmentation is not an easy technique. It is difficult to obtain good results. Manual tuning is required taking into account the specificity of the application.
- Frequently the image is converted to a binary image first, followed by a distance transformation, before it is subjected to watershed segmentation.

Example

Let's apply this technique to detect the individual cells on the image of an analysis sample containing the Zygosaccharomyces Bailii yeast (a well known food spoiler), see Figure 8.13.

As a preconditioning technique, we filter the image using a moving average filter.

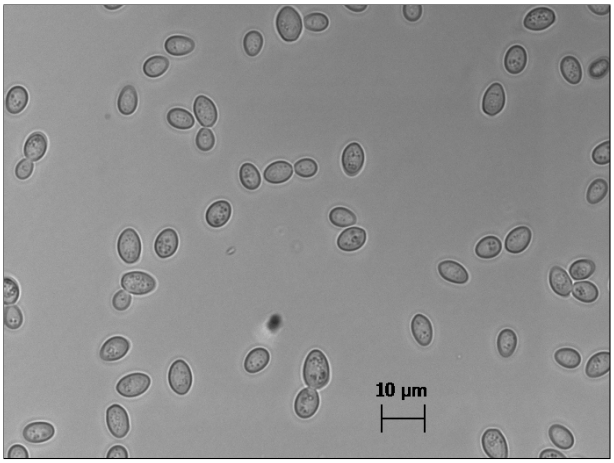
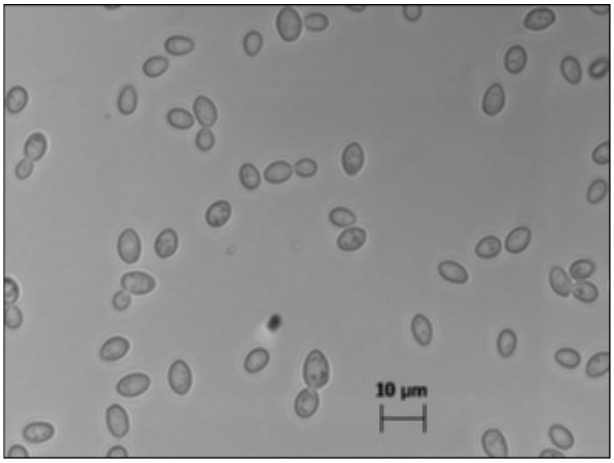
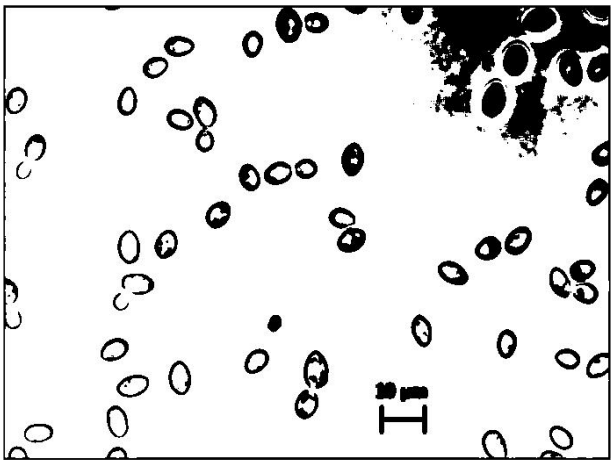


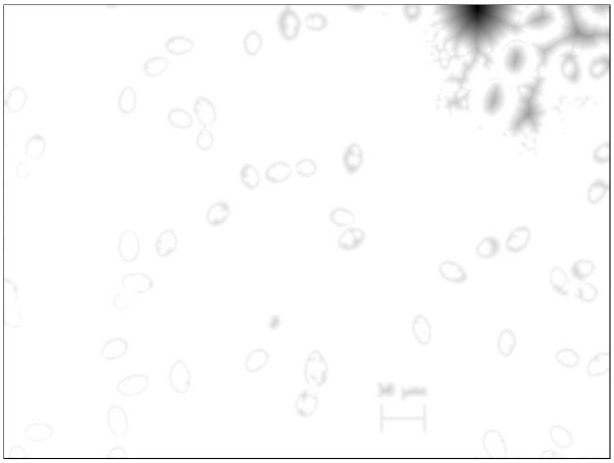
Figure 8.13: Image of an analysis sample containing the Zygosaccharomyces Bailii yeast sample (source http://en.wikipedia.org/wiki/Zygosaccharomyces_bailii (Creative Commons 2.0))



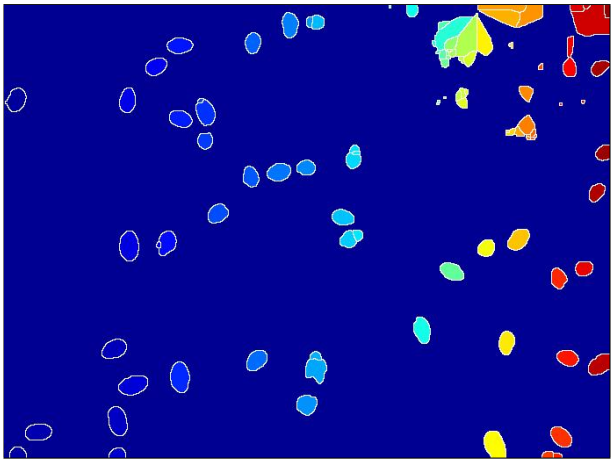
After Otsu thresholding, this becomes:



Then, we apply a distance transformation. To save on ink, we inverted the image.



After a watershed transformation, this becomes:



In this image, a separate color has been used for every basin. You'll still notice some artifacts at the top right of the image. Artifacts when using the watershed transformation very often occur. Special attention has to be devoted to this issue.

Exercises

We will not make exercises on the watershed algorithm. Find a good library that implements the algorithm if you need it. Consider e.g. MATLAB/OCTAVE's `watershed()` function i.c.w. the `bwdist()` function.

Appendix A

Stochastic Theory

In daily life, we often encounter phenomena that seem unpredictable. We often call them *random* phenomena. The term 'unpredictable' in fact is more correct than the term 'random'. In many cases the physics of of the phenomenon at hand is well understood and therefore not random at all. The only problem is that the physics governing the phenomenon is often very complex and therefore is influenced by a lot of factors. In addition, the sensitivity of the outcome on these factors is so big, that predicting that outcome becomes (almost) impossible.

As an example, consider throwing a dice. Throwing the dice can be accurately modeled using Newton's laws and fluid dynamics. However, the number of factors influencing the experiment of throwing a dice (the specific contraction of every muscle in your body when throwing, the humidity, the wind speed, the temperature, the pressure of the air, the smoothness of the table surface, the roughness of the dice edges, ...) is so big, that it is impossible to predict its outcome.

Where, in general, predicting the outcome of a single complex experiment is often impossible, luckily, in many cases we can make good predictions about executing a (big) number of experiments. In fact, the bigger the number of experiments, the more accurate the statements we can make about it.

Again, consider the example of a dice. When throwing a dice a thousand times, the number of cases in which the outcome will be a six, will be close to 1000/6 (in case of a fair dice). When summing all outcomes of throwing a thousand times, we expect the total value to be close to 3500.

In signal and image processing, stochastic theory is more and more gaining in importance. Therefore, in this appendix, we review the basic theory of stochastic phenomena.

A.1 Experiments and outcomes

The very basic concept of stochastic theory is the *experiment*. Consider it to be a particular action that

- one can take *repeatedly*, and
- leads to a specific *outcome* (a result).

In the process of throwing a dice, throwing a single time is the experiment, and the outcome is the number of eyes on the side facing up, i.e. one of 1, 2, 3, 4, 5 or 6.

The set of all outcomes is denoted by Ω, the outcome set. The outcome set can be discrete (e.g., when throwing a dice $\Omega = \{1, 2, 3, 4, 5, 6\}$), or can be continuous (e.g., the length of a person, when selecting a human being from the earth's population is the experiment).

A.2 Events

An event is a subset of the outcome set. If the outcome of an experiment is part of this subset, then we say *the event happened*. If it is not a part of the event subset, then we say *the event did not happen*.

In the process of throwing a dice, 'throwing an odd number', 'throwing a number less than or equal to two, would be events.

Note that the empty set $\emptyset$ and the outcome set Ω are both events themselves. The Ω-event always happens, the $\emptyset$-event never happens.

To keep things simple, we define the complement E_C of an event to be:

$$E^C = \Omega \setminus E$$

The set of all events is called the *event set* $\mathcal{E}$:

$$\mathcal{E} = \{E \mid E \subset \Omega\}$$

A.3 Probability function

We can define a probability function P that maps every event to a probability value:

$$P : \mathcal{E} \mapsto [0, 1]$$

The probability function expresses our estimate of the likelihood that an event occurs at the next experiment. If the probability function is dependent on time, we call the phenomenon under study a *stochastic process*. In the scope of this appendix, we will not consider this time dependence.

In order to be useful, a probability function should obey a number of rules:

1. $\forall E \in \mathcal{E} : 0 \leq P(E) \leq 1$
2. $P(\emptyset) = 0$
3. $P(\Omega) = 1$
4. $\forall E_1, E2 \in \mathcal{E} : E_1 \cap E_1 = \emptyset \quad \Rightarrow \quad P(E_1 \cup E_2) = P(E_1) + P(E_2)$

These *axiomas* enforce that our probability function makes sense.

A number of properties can be proven for a proper probability function:

1. $\forall E \in \mathcal{E} : P(E) + P(E^C) = 1$

2. $\forall E_1, E_2 \in \mathcal{E} : P(E_1 \cup E_2) = P(E_1) + P(E_2) - P(E_1 \cap E_2)$

We can also define a *conditional probability* $P(E \mid C)$ that expresses our estimate of an event E happening, knowing that event C happened for the experiment under study:

$$P(E \mid C) = \frac{P(E \cap C)}{P(C)}$$

in which we assumed that $C \neq \emptyset$. We call $P(E)$ the probability *a priori* and $P(E|C)$ the probability *a posteriori*. In plain English, we read $P(E|C)$ as the probability E happens, *given* C *happened* or short the probability of E *given* C.

This easily leads us to the well known *Bayes' rule*:

$$P(E|C) = \frac{P(C|E)P(E)}{P(C)} = \frac{P(C|E)P(E)}{P(C|E)P(E) + P(C|E^C)P(E^C)}$$

A.4 Stochastic or random variables

A stochastic variable is a mapping of the outcome set to a real number that is compatible with the event set.

$$X : \Omega \mapsto \mathbb{R}$$

The compatibility means that for every $a \in \mathbb{R}$ its corresponding event E_a, defined as:

$$E_a = \{\omega \mid X(\omega) \leq a\}$$

should be a proper event, i.e.:

$$E_a \in \mathcal{E}$$

When throwing a pair of dice, the sum of the eyes of a single throw would be an appropriate stochastic variable.

A.5 Describing stochastic variables

A stochastic variable can be described using its distribution. We distinguish two flavors of probability distributions.

A.5.1 Cumulative probability distribution

The *cumulative probability distribution* of a stochastic variable is a function $F_X(x)$ that is defined as:

$$F_X(x) = P(X \leq x)$$

with P the probability function corresponding to the experiments.

A.5.2 Ordinary probability distribution

Experiments with discrete outcome sets The *ordinary probability distribution* is defined as:

$$f_X(x) = P(X = x)$$

It is also known as the *probability mass function*.

Experiments with continuous outcome sets The *ordinary probability distribution* is defined as:

$$f_X(x) = \frac{\mathrm{d}}{\mathrm{d}x} F_X(x)$$

It is also known as the *probability density function*.

A.5.3 Properties

Again, a number of properties can be derived:

1. $F_X(x) = \int_{-\infty}^{x} f_X(u)\,\mathrm{d}u$
2. $0 \leq F_X(x) \leq 1$
3. $a \leq b \Rightarrow F_X(a) \leq F_X(b)$
4. $P(a < X \leq b) = F_X(b) - F_X(a)$

A.6 Describing distributions using moments

Distributions can have the most exotic shapes. Often the particular shape is not so important, but some basic properties are. This is the basis for describing distributions using *moments*. We distinguish *raw* moments and *centralized* moments.

A.6.1 Expected value

The basis for these moments is the *expectancy operator* E.

Experiments with discrete outcome sets

$$E\{X\} = \sum_i x_i f_X(x_i)$$

As the expectancy operator is an integral operator, it is linear (assuming the sum is related to the same random variable, and therefore governed by a single (joint) probability density function).

Experiments with continuous outcome sets

$$E\{X\} = \int_{-\infty}^{+\infty} x f_X(x)\,\mathrm{d}x$$

A.6.2 Raw moments

In general the i-th order raw moment $\alpha_{i,X}$ are defined as:

$$\alpha_{i,X} = E\left\{X^i\right\}$$

A.6.3 Centralized moments

In general the i-th order centralized moment $\mu_{i,X}$ is defined as:

$$\mu_{i,X} = E\left\{\left(X - \alpha_{1,X}\right)^i\right\}$$

It is easy to check that $\mu_{1,X} = 0$.

A.6.4 Characteristic values

Even more than the raw or centralized moments, the characteristic values are used. The symbols used below are quite typical and widely used:

The mean The mean indicates where the center of gravity of the distribution is located.

$$\mu_X = \alpha_{1,X} = E\left\{X\right\}$$

Note the very confusing symbol that is being used!

The variance The variance indicates how wide the distribution extends from its mean.

$$\sigma_X^2 = \mu_{2,X} = E\left\{\left(X - \mu_X\right)^2\right\}$$

Often, its square root is being used, the so-called *standard deviation* σ_X.

The skewness The skewness indicates if the distribution is assymetric.

$$\xi_X^3 = \frac{\mu_{3,X}}{\sigma_X^3}$$

If ξ_X^3 is positive, the distribution extends more to the right, if it is negative, it extends more to the left.

The kurtosis The kurtosis indicates the flatness of the distribution.

$$\kappa_X^4 = \frac{\mu_{4,x}}{\sigma_X^4} - 3$$

If $\kappa_X^4 > 0$ then it is flatter than the normal distribution. If it is smaller than zero, it is more peaked than the normal distribution.

A.7 Transformations of stochastic variables

Often, derived versions of stochastic variables are useful in some calculations. We consider two specific cases.

Affine transformation Given a stochastic variable X distributed as described by $F_X(x)$, we can derive a new stochastic variable Y by affine transformation:

$$Y = Ax + b \text{ with } a > 0.$$

This transformation is distributed according to $F_Y(y)$ with

$$F_Y(y) = F_X\left(\frac{y-b}{a}\right)$$

Nonlinear transformation Given a stochastic variable X distributed as described by $f_X(x)$, we can derive a new stochastic variable Y by nonlinear transformation g:

$$Y = g(X)$$

In case g is monotonically strictly increasing or decreasing, the following holds:

$$f_Y(y) = \frac{1}{\frac{\partial g(x)}{\partial x}} f_X(x)$$

Bibliography

[GW07] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing*. Prentice Hall, 3rd edition, 2007.

[GWE09] Rafael C. Gonzalez, Richard E. Woods, and Steven L. Eddins. *Digital Image Processing Using MATLAB*. Prentice Hall, 2nd edition, 2009.

[Jäh02] Bernd Jähne. *Digital Image Processing*. Springer–Verlag, 5th edition, 2002.

[Mal09] Stéphane Mallat. *A Wavelet Tour of Signal Processing* (*The Sparse Way*). Elsevier, Academic Press, 2009.

[Wal08] James S. Walker. *A primer on wavelets and their scientific applications*. Chapman & Hall/CRC, 2nd edition, 2008.

Index