Summary of the lecture
'Image Processing'
by Prof. Christian Bauckhage
(WiSe 2021/2022)

Fabrice Beaumont
Matrikel-Nr: 2747609
Rheinische Friedrich-Wilhelms-Universität Bonn

January 7, 2022

Chapter 2

Digital Photography

2.1 Image acquisition and digital photography

(Digital) photography is the practice of visualizing, recording and storing light.

Definition 1.1: Light

Light can be described as both

- as electromagnetic radiation where energy propagates in form of electromagnetic waves and
- as a particle or quantum phenomenon where photons carry the electromagnetic force.

Light has a constant *speed* of $c := 299\,792\,458\,\mathrm{m/s}$. Light can have different *frequency* ν ($[\mathrm{s}^{-1}]$) and *wavelength* λ ($[\mathrm{m}]$), but these properties are reciprocal:

$$c = \nu\lambda \qquad \left[\frac{\mathrm{m}}{\mathrm{s}}\right] \tag{Eq. 1}$$

The *energy* E of light is defined as

$$E = h\nu = h\frac{c}{\lambda} \qquad \Big[\mathrm{J}\Big] \tag{Eq. 2}$$

where $h = 6.6260689633 \times 10^{-34}[\mathrm{J\,s}]$ is the **Planck's constant**.

From these definitions we can derive the relation that light with a high frequency and small wavelength carries more energy.

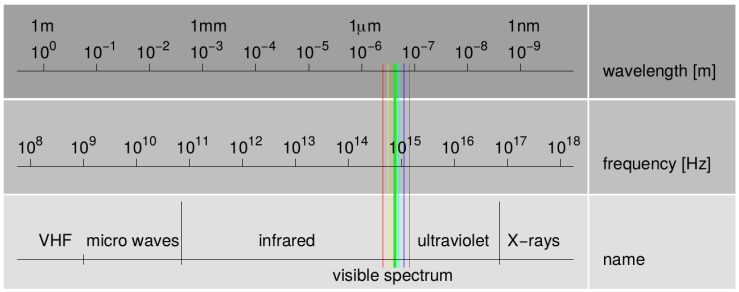

Figure 2.1 – Overview over the electromagnetic spectrum.

There are three ways to visualize light. It can be:

- **reflected** (e.g. photographic images or scanned images),
- **absorbed** (e.g. X-ray images) or
- **emitted** (infrared images).

There are several ways to record light. Digital (color-)photography relies on CCD cameras. CCD stands for **charged-coupled device** and describes a semi-conductor device consisting of photosensitive elements. Modern consumer cameras have at least 1600×1200 such elements.
Rays of light incident on the camera lens hit the CCD matrix. These photons generate a charge on the CCD elements by freeing electrons. The more light there is, the more electrons are freed. Depending on the camera quality, a single CCD element stores 100,000 to 350,000 electrons before saturating. An electric field gathers electrons into packets which are read and quantized into intensity levels. Thus every CCD element records a value of light intensity.
Since the amount of represented intensity levels relates to the needed memory bits per CCD, and since the perception of the human visual system is limited in this regard, typical images typically only store $2^8 = 256$ intensity levels (one byte) per color channel (intensity quantization).

Definition 1.2: Intensity image

Intensity images (**grayscale images**) do not contain color information and only display recorded light intensity (luminescence) per pixel in terms of shades of gray.

Definition 1.3: Color image

Color images do contain color information and split the recorded light into basic colors (commonly red, green and blue) from which the appropriate colors can be reconstructed on a display device later. To split the light one can use either a **trichroic prism** and record each basic color in a separate CCD array, or superimpose a mosaic of color filters (**Bayer filter**) over a single CCD array. Since the latter technique uses three times less CCD elements it is much cheaper (but also less accurate).

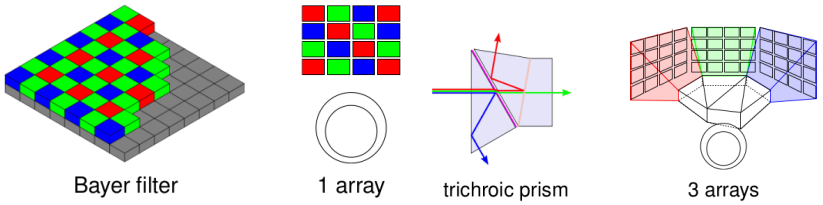

Figure 2.2 – Two techniques of CCD cameras to split recoreded light into three basic colors.

There are several problems with the CCD technology:

- **Geometric distortions**. CCD elements are typically not perfectly square. This may necessitate computational correction to compensate for geometric distortions.
- **Blooming**. Overexposure may free to many electrons, which may overflow to neighboring elements and cause highlight effects.
- **Dark noise**. CCD sensors are sensitive to parts of the invisible electromagnetic spectrum (UV or IR radiation) and may thus record invisible apparitions.

There are numerous file formats for storing digital images. But these can be separated into two main types of image file formats:

- vector graphics formats (e.g. SVG, FIG) and
- pixmaps (bitmap, raster graphics; e.g. JPG, GIF, PNG, TIFF).

Definition 1.4: PPM format

The **PPM** (**portable pixmap**) **format** stores image data in the following way:
The first four lines in the file format contain:

1. a magic number (indicating ASCII [P1, P2, P3] or binary [P4, P5, P6] format),
2. comments,
3. image width and image height and
4. the maximum color value (which is the same for all color channels).

After these lines, the image data is stored in red-green-blue-triples of hexadecimal values.

```
P6
# CREATOR: XV Version 3.10a  Rev: 12/29/94 (PNG patch 1.2)
74 82
255
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff fe fe fe fe fe fe ff ff ff fe fe fe fe fe fc
ff ff fd c7 c6 c4 19 15 ...
```

Figure 2.3 – Example header of a PPM file.

Keep in mind, that there are many different ways of image compression. In general these annihilate information and thus reduce memory requirements.

2.2 Representations of digital images

As explained, a digital image represents digitized and quantized information. The basic element of digital images are called **pixels** (picture elements), which are (commonly) arranged on a rectangular grid called a **pixmap**. The **resolution** of a digital image is expressed in terms of width times height of its pixmap. Digital images often have an aspect ratio of 4:3 (e.g. 1024×768) or 16:9 (e.g. 1920×1080).

In memory, intensity images are stored in a single pixmap. Color images are stored in three to four pixmaps. One per color channel and a possible α-layer. Accordingly, these images can be stored in two-, three- or four-dimensional arrays. Rows and columns of a pixel array are counted from zero and the pixel $p_{0,0}$ is located in the *upper left corner* of the pixel array.

Definition 2.1: 4-Neighborhood

The **4-neighborhood** of a pixel is the set of the two vertically and the two horizontally adjacent pixels.

Definition 2.2: 8-Neighborhood

The **8-neighborhood** of a pixel is the set of the 4-neighborhood and the four diagonally adjacent pixels.

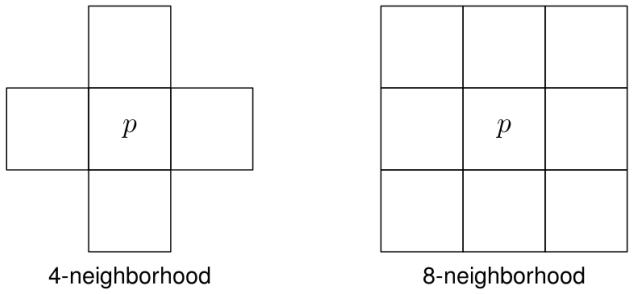

Figure 2.4

Note that the Euclidean distance yields different distances between the pixels in the 8-neighborhood. It is possible to use another metric or another arrangement of the pixelmap (e.g. a hexagonal-grid), but the technically simplest approach with respect to computer memory is the default rectangular grid.

2.2.1 Images as functions

In a digital intensity image, array coordinates $[i, j] \in \{0, \ldots, M-1\} \times \{0, \ldots, N-1\}$ are mapped to intensity values $p_{i,j} \in \{0, \ldots, L-1\}$. Thus one may think of such an image as a **bi-variate discrete function with finite domain** $f : \mathbb{R}_+^2 \to \mathbb{R}_+$.

Since the pixels are enumerated from the upper left corner, mapping the planar points $[x_j, y_i]$ to the images array coordinates $[i, j]$ is done as

$$[x_i, y_j] \leftrightarrow [j, M - 1 - i] \tag{Eq. 3}$$

Also, we assume that the two-dimensional point at which an image function is defined are equally spaced. More precise, the horizontal and vertical distances between neighboring point amounts to one:

$$\begin{aligned} \delta_x &= |x_{j\pm1} - x_j| = |x \pm 1 - x| = 1 \\ \delta_y &= |y_{i\pm1} - y_j| = |y \pm 1 - y| = 1 \end{aligned} \tag{Eq. 4}$$

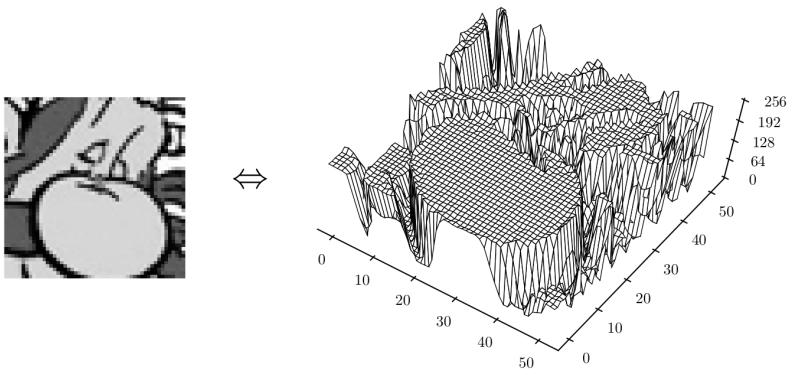

Figure 2.5

Notation: To signify that f is continuous, write $f(x)$. To signify that f is discrete (partial), write $f[x]$ (with $x \in \{\mathbb{R}, \mathbb{R}^2\}$).

Definition 2.3: Derivatives

Let f be a continuous, uni-variate function $f : \mathbb{R} \to \mathbb{R}$ where $x \mapsto f(x)$ and $x_0 \in \mathbb{R}$.
At x_0, the **derivative of f from above** is

$$f'_+(x_0) = \lim_{\epsilon \to 0} \frac{f(x_0 + \epsilon) - f(x_0)}{\epsilon} \tag{Eq. 5}$$

At x_0, the **derivative of f from below** is

$$f'_-(x_0) = \lim_{\epsilon \to 0} \frac{f(x_0) - f(x_0 - \epsilon)}{\epsilon} \tag{Eq. 6}$$

f is **differentiable** at x_0, if

$$f'_+(x_0) = f'_-(x_0) = f'(x_0) = \frac{\partial}{\partial x} f(x_0) \tag{Eq. 7}$$

In this case, $f'(x_0)$ indicates the *rate of change* of f at x_0. That is the *slope of the tangent* of f at x_0.
f is differentiable, if it is differentiable at any $x \in \mathbb{R}$.
Note that the derivation is a **linear operator** [a]:

$$\frac{\partial}{\partial x}\Big(af(x) + bg(x)\Big) = a\frac{\partial}{\partial x} f(x) + b\frac{\partial}{\partial x} g(x) \tag{Eq. 8}$$

[a] An **operator** is a function that maps functions to functions.

For the discrete case set for simplicity $x_{i+1} - x_i = 1 = \epsilon$. Then, the definitions are analogue:

- **Backward difference**:

$$f'_-[x] = \frac{f[x] - f[x-1]}{x - (x-1)} = f[x] - f[x-1] \tag{Eq. 9}$$

- **Forward difference**:

$$f'_+[x] = \frac{f[x+1] - f[x]}{(x+1) - x} = f[x+1] - f[x] \tag{Eq. 10}$$

- **Central difference**:

$$f'[x] = \frac{f'_+[x] - f'_-[x]}{2} = \frac{f[x+1] - f[x-1]}{2} \tag{Eq. 11}$$

Note that if $f[x]$ has been sampled from $f(x)$, then the central difference of $f[x]$ provides a provably better approximation of the derivative of $f(x)$ than the backward- or forward differences.
Using an operator notation, we will also write the central difference as $f'[x] = d_x f[x]$.

2.2.2 Image gradient

By considering images as functions, we can compute their the gradient. For intensity images $f[x, y]$ it is defined as

$$\nabla f[x, y] = \begin{bmatrix} d_x f[x, y] \\ d_y f[x, y] \end{bmatrix} = \frac{1}{2} \begin{bmatrix} f[x+1, y] - f[x-1, y] \\ f[x, y+1] - f[x, y-1] \end{bmatrix} \quad \text{(Eq. 12)}$$

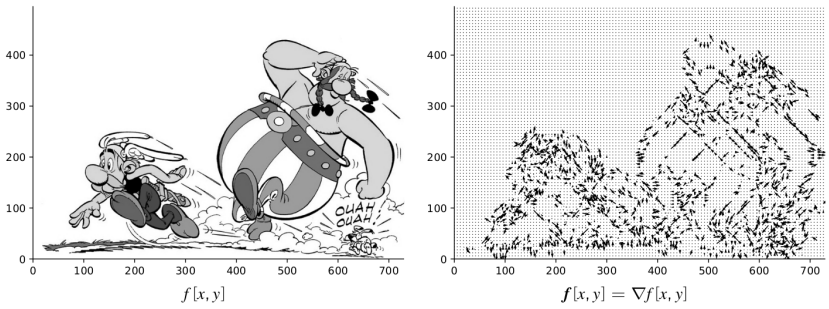

Figure 2.6 – Example intensity image and its gradient image.

Gradients of intensity images are typically visualized by only considering its magnitude

$$\|\nabla f[x, y]\| = \sqrt{\langle \nabla f[x, y], \nabla f[x, y] \rangle} = \sqrt{\left(\nabla f[x, y]\right)^2} \quad \text{(Eq. 13)}$$

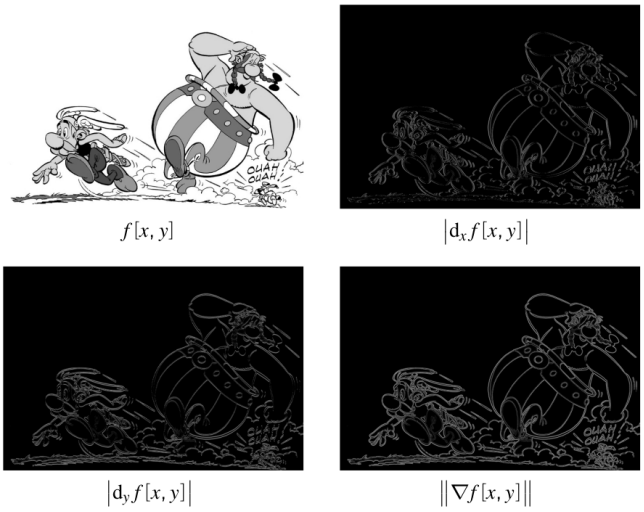

Figure 2.7 – Example of an intensity image, the norms of the directed gradients and the magnitude of the total gradient.

As one can see, the gradient (and its magnitude) can be used as an **edge detector**.

2.2.3 Emboss effect

The emboss effect transforms an image such that it resembles a copper engraving or an etching. It can be achieved by applying the following operator:

$$g[x,y] = \max\Big(0,\ \min\big(255,\ 128 + f[x+1,y-1] - f[x-1,y+1]\big)\Big) \quad \text{(Eq. 14)}$$

Note that by definition it is $0 \leq g[x,y] \leq 255$. Furthermore, the resulting pixels are computed as 128 plus the difference in intensity between the lower left and upper right neighboring pixel (diagonal gradient).

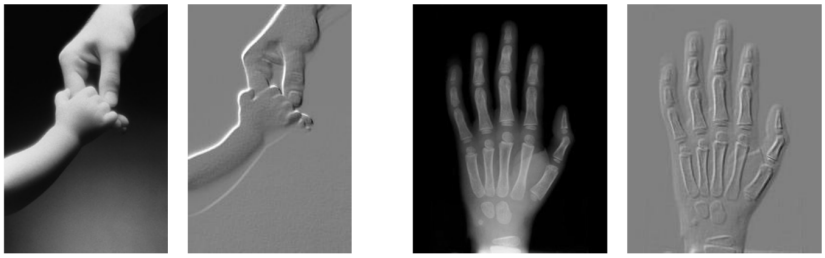

Figure 2.8 – Two examples of the emboss effect. Left the original intensity image, right an emboss effect on it.

2.3 Linear filters

Examples of linear filters are low-pass filters, Gaussian filters and mean filters.

Definition 3.1: Linear filters

Convolution filters

$$h(x) = f(x) * g(x)$$

(where $f(x)$ is a signal and $g(x)$ is a filter) are called **convolution filters** because the convolution is a linear operations.

2.3.1 Gaussian filter

Space domain image filtering is done by convolution of the image with a filter mask:

$$h[x,y] = (f * g)[x,y] = \sum_{u=-\frac{n}{2}}^{\frac{n}{2}} \sum_{v=-\frac{m}{2}}^{\frac{m}{2}} f[x-u, y-v]g[u,v]$$

A 2D convolution with a Gaussian filter mask G_σ is a separable operation and thus can be computed in terms of two 1D convolutions with two 1D Gaussians. This reduces the per pixel effort from $\mathcal{O}(m^2)$ to $\mathcal{O}(m)$:

$$
\begin{aligned}
(f * G_\sigma)(x,y) &= \frac{1}{2\pi\sigma^2} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(\xi, v) \exp\Big(- \frac{(x-\xi)^2 + (y-\nu)^2}{2\sigma^2} \Big) d\xi d\nu \\
&= \frac{1}{2\pi\sigma^2} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(\xi, v) e^{-\frac{(x-\xi)^2}{2\sigma^2}} e^{\frac{-(y-\nu)^2}{2\sigma^2}} d\xi d\nu \\
&= \frac{1}{2\pi\sigma^2} \int_{-\infty}^{\infty} \Big(\frac{1}{\sqrt{2\pi}\sigma} \int_{-\infty}^{\infty} f(\xi, \nu) e^{-\frac{(x-\xi)^2}{2\sigma^2}} d\xi \Big) e^{\frac{-(y-\nu)^2}{2\sigma^2}} d\nu
\end{aligned}
$$

Figure 2.9 – Illustration of the separability of the convolution with a Gaussian filter mask

Note that the convolution with a bi-variate Gaussian is separable, because the bi-variate Gaussian itself is separable. For the *continuous*, isotropic case, we have:

- Uni-variate Gaussian:

$$g_\sigma(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}}$$

- Bi-variate Gaussian:

$$G_\sigma(x,y) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2+y^2}{2\sigma^2}} = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}} e^{-\frac{y^2}{2\sigma^2}} = g_\sigma(x) \cdot g_\sigma(y)$$

For the *discrete*, isotropic case, we have:

- Uni-variate Gaussian:

$$g_\sigma[x_i] = \frac{e^{-\frac{x_i^2}{2\sigma^2}}}{\sum_i e^{-\frac{x_i^2}{2\sigma^2}}}$$

- Bi-variate Gaussian:

$$G_\sigma[x_i, y_j] = \frac{e^{-\frac{x_i^2+y_j^2}{2\sigma^2}}}{\sum_{i,j} e^{-\frac{x_i^2+y_j^2}{2\sigma^2}}} = g_\sigma[x_i] \cdot g_\sigma[y_j]$$

If we understand the discrete, uni-variate Gaussian as a finite dimensional vector $g \in \mathbb{R}^m$ with entries

$$g = \left[g_\sigma\left[x_{-\frac{m}{2}}\right], \ldots, g_\sigma[x_0], \ldots, g_\sigma\left[x_{+\frac{m}{2}}\right] \right]^\mathsf{T}$$

we can think of the discrete, bi-variate Gaussian as a finite dimensional outer product matrix $G \in \mathbb{R}^{m \times m}$: $G = gg^\mathsf{T}$.

Note that this also works for non-isotropic yet uncorrelated bi-variate Gaussians:

$$G_{\sigma_i,\sigma_j}[x_i, y_j] = \frac{\exp\left(-\frac{x_i^2}{2\sigma_x^2} - \frac{y_j^2}{2\sigma_y^2} \right)}{\sum_{i,j} \exp\left(-\frac{x_i^2}{2\sigma_x^2} - \frac{y_j^2}{2\sigma_y^2} \right)}$$

Recursive Gaussian filtering

Since the 2D Gaussian is separable, an efficient 1D Gaussian filter would yield a Gaussian filtering algorithm in $\mathcal{O}(1)$ time. This will be discussed in this section.

Intensity images are discrete bi-variate signals $f[x, y]$. Here, we will focus on discrete uni-variate signals $f[x]$. However, instead of thinking of them as functions of space x, we will consider them as functions of time t (to match the terminology used in signal processing).

The key insight is, that a discrete, uni-variate mean filter can be computed

recursively:

$$
\begin{aligned}
h[t] &= \frac{1}{m} \sum_{i=-\frac{m}{2}}^{\frac{m}{2}} f[t+i] \\
&= \frac{1}{m} \sum_{i=-\frac{m}{2}}^{\frac{m}{2}-1} f[t+i] + \frac{1}{m} f\big[t+\frac{m}{2}\big] \\
&= \frac{1}{m} \sum_{i=-\frac{m}{2}-1}^{\frac{m}{2}-1} f[t+i] - \frac{1}{m} f\big[t-\frac{m}{2}-1\big] + \frac{1}{m} f\big[t+\frac{m}{2}\big] \\
&= \frac{1}{m} \sum_{i=-\frac{m}{2}}^{\frac{m}{2}} f[t-1+i] - \frac{1}{m} f\big[t-1-\frac{m}{2}\big] + \frac{1}{m} f\big[t+\frac{m}{2}\big] \\
&= h[t-1] - \frac{1}{m} f\big[t-1-\frac{m}{2}\big] + \frac{1}{m} f\big[t+\frac{m}{2}\big]
\end{aligned}
$$

Definition 3.2: Causal and anti-causal filters

Let f be a discrete, uni-variate function and h a filter on it. The **causal** part of the filter are all terms that only use the past values $f[t-i]$ to compute the filter value at t.
The **anti-causal** part of the filter are all terms that only use the future values $f[t+i]$ to compute the filter value at t.

Example of causal and anti-causal filter components of mean filtering:

$$
h[t] = \frac{1}{m} \sum_{i=-\frac{m}{2}}^{\frac{m}{2}} f[t+i] = \underbrace{\frac{1}{m} \sum_{i=0}^{\frac{m}{2}} f[t-i]}_{\text{cuasal}} + \underbrace{\frac{1}{m} \sum_{i=1}^{\frac{m}{2}} f[t+i]}_{\text{anti-cuasal}}
$$

In signal processing, the convent is to denote the input to a filter as $x[t]$, and the output of a filter as $y[t]$.

Definition 3.3: Finite and infinite impulse response filter

A **Finite impulse response (FIR) filter** of order m is a causal filter where

$$
y[t] = a_0 x[t] + a_1 x[t-1] + \cdots + a_m x[t-m] \; = \sum_{i=0}^{m} a_i x[t-i]
$$

An **infinite impulse response (IIR) filter** is a causal filter where

$$y[t] = \sum_{i=0}^{\infty} a_i x[t-i]$$

IIR systems are often modeled in terms of difference equations or *recursive filters*:

$$y[t] = \sum_{i=0}^{m} a_i x[t-i] - \sum_{i=1}^{n} b_i y[t-i]$$

Lets use the notation

$$y[n] = \sum_{m=0}^{P} a_m x[n-m] - \sum_{m=1}^{Q} b_m y[n-m]$$

or, equivalently with $b_0 = 1$

$$\sum_{m=0}^{Q} b_m y[n-m] = \sum_{m=0}^{P} a_m x[n-m]$$

Such IIR systems have a particularly simple Z-transform or transfer functions. Using $\mathcal{Z}\{x[n-k]\} = z^{-k}X(z)$ we can write:

$$Y(z) = \sum_{m=0}^{P} a_m z^{-m} X[z] - \sum_{m=1}^{Q} b_m z^{-m} Y(z)$$

or, again equivalently

$$Y(z)\Big(1 + \sum_{m=1}^{Q} b_m z^{-m}\Big) = X(z) \sum_{m=0}^{P} a_m z^{-m}$$

Using this we can define a *transfer function*:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{m=0}^{P} a_m z^{-m}}{1 + \sum_{m=1}^{Q} b_m z^{-m}}$$

The transfer function of an IIR system is a rational function (and a ratio of two polynomials). The numerator $Y(z)$ is a convolution, the denominator $X(z)$ is a recursion. The goal is to determine $H(z)$ such that it realizes Gaussian filtering. This consists of two challenges:

1. Gaussian filtering is only convolutional (that is, there is no recursion). Thus we need to approximate the Z-transform of a Gaussian using two polynomials.

2. The transfer function is causal (only looks back in time), but Gaussian filtering is anti-causal (requires us to look ahead in time). Thus we need to express an anti-causal filter in terms of causal components only.

First, lets solve the second challenge. Consider causal and anti-causal components by requiring

$$h[n] = h_+[n] + h_-[n]$$

where

$$h_+[n] := \begin{cases} h[n] & \text{if } n \leq 0 \\ 0 & \text{if } n > 0 \end{cases}, \qquad h_-[n] := \begin{cases} 0 & \text{if } n \leq 0 \\ h[n] & \text{if } n > 0 \end{cases}$$

We will make use of the following equations (without proofing them):

$$\mathcal{Z}\{h_+[n]\} = H_+(z) = \frac{\sum_{m=0}^{P-1} a_m^+ z^{-m}}{1 + \sum_{m=1}^{Q} b_m^+ z^{-m}} \tag{Eq. 15}$$

$$\mathcal{Z}\{h_-[n]\} = H_-(z) = \frac{\sum_{m=1}^{P} a_m^- z^{m}}{1 + \sum_{m=1}^{Q} b_m^- z^{m}} \tag{Eq. 16}$$

Using these, we construct a recursive Gaussian filter as a sum of a causal- and an anti-causal system $h[n] = h_+[n] + h_-[n]$ or $H(z) = H_+(z) + H_-(z)$.

To implement the composite filter, we apply the causal- and the anti-causal filter to an input sequence $x[n]$ and accumulate their results in an output sequence $y[n]$.

It remains to determine the coefficients a_m^+, b_m^+, a_m^- and b_m^-. Therefore set $P = Q$. Without proof we claim that

- for symmetric, even filters, the coefficients are related as

$$\begin{aligned} b_m^- &= b_m^+ && \forall m \\ a_m^- &= a_m^+ \quad -b_m^+ a_0^+ && \forall m = 1, \ldots, Q-1 \\ a_Q^- &= \quad -b_Q^+ a_0^+ \end{aligned}$$

- for anti-symmetric, odd filters, we have

$$\begin{aligned} b_m^- &= b_m^+ && \forall m \\ a_m^- &= -a_m^+ \quad +b_m^+ a_0^+ && \forall m = 1, \ldots, Q-1 \\ a_Q^- &= \qquad\quad b_Q^+ a_0^+ \end{aligned}$$

Now only a_m^+ and b_m^+ have to be determined.

The solution to the first challenge, that is to approximate the Z-transform of a Gaussian using two polynomials, goes as follows: Approximate the causal part of

$$g_\sigma[n] = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{n^2}{2\sigma^2}}$$

as a linear combination of four simple exponential functions

$$h_\sigma^+[n] = \sum_{i=0}^{k=3} w_i e^{-\lambda_i \frac{n}{\sigma}}$$

because this model has a simple Z-transform. Using substitutions and Euler's identity, we can write

$$h_\sigma^+[n] = \sum_{i=1}^{2} \left(\alpha_i \cos\left(\frac{n\omega_i}{\sigma}\right) + \beta_i \sin\left(\frac{n\omega_i}{\sigma}\right) \right) e^{-\frac{n\gamma_i}{\sigma}}$$

Computing the Z-transform of $h_\sigma^+[n]$ and comparison to $h_+[n]$ in Eq. 15 yields:

$$
\begin{aligned}
a_0^+ &= \alpha_1 + \alpha_2 \\
a_1^+ &= e^{-\frac{\gamma_2}{\sigma}} \left(\beta_2 \sin\frac{\omega_2}{\sigma} - (\alpha_2 + 2\alpha_1)\cos\frac{\omega_2}{\sigma}\right) + e^{-\frac{\gamma_1}{\sigma}} \left(\beta_1 \sin\frac{\omega_1}{\sigma} - (2\alpha_2 + \alpha_1)\cos\frac{\omega_1}{\sigma}\right) \\
a_2^+ &= 2e^{-\frac{\gamma_1+\gamma_2}{\sigma}} \left((\alpha_1 + \alpha_2)\cos\frac{\omega_2}{\sigma}\cos\frac{\omega_1}{\sigma} - \cos\frac{\omega_2}{\sigma}\beta_1 \sin\frac{\omega_1}{\sigma} - \cos\frac{\omega_1}{\sigma}\beta_2 \sin\frac{\omega_2}{\sigma}\right) \\
&\quad + \alpha_2 e^{-2\frac{\gamma_1}{\sigma}} + \alpha_1 e^{-2\frac{\gamma_2}{\sigma}} \\
a_3^+ &= e^{-\frac{\gamma_2+2\gamma_1}{\sigma}} \left(\beta_2 \sin\frac{\omega_2}{\sigma} - \alpha_2 \cos\frac{\omega_2}{\sigma}\right) + e^{-\frac{\gamma_1+2\gamma_2}{\sigma}} \left(\beta_1 \sin\frac{\omega_1}{\sigma} - \alpha_1 \cos\frac{\omega_1}{\sigma}\right) \\
b_1^+ &= -2e^{-\frac{\gamma_2}{\sigma}} \cos\frac{\omega_2}{\sigma} - 2e^{-\frac{\gamma_1}{\sigma}} \cos\frac{\omega_1}{\sigma} \\
b_2^+ &= 4\cos\frac{\omega_2}{\sigma}\cos\frac{\omega_1}{\sigma} e^{-\frac{\gamma_1+\gamma_2}{\sigma}} + e^{-2\frac{\gamma_2}{\sigma}} + e^{-2\frac{\gamma_1}{\sigma}} \\
b_3^+ &= -2\cos\frac{\omega_1}{\sigma} e^{-\frac{\gamma_1+2\gamma_2}{\sigma}} - 2\cos\frac{\omega_2}{\sigma} e^{-\frac{\gamma_2+2\gamma_1}{\sigma}} \\
b_4^+ &= e^{-\frac{2\gamma_1+2\gamma_2}{\sigma}}
\end{aligned}
$$

To determine the coefficients α_i, β_i, ω_i and γ_i, Deriche (1992) resorts to the method of least squares by minimizing the loss function

$$E = \sum_{n=0}^{1000} \left[h_\sigma^+\left[\frac{n}{100}\right] - g_\sigma\left[\frac{n}{100}\right]\right]^2$$

His results are:

	α_i	β_i	γ_i	ω_i
$i = 1$	1.6800	3.7350	1.7830	0.6318
$i = 2$	−0.6803	−0.2598	1.7230	1.9970

Given $\alpha_i, \beta_i, \gamma_i, \omega_i$ and the standard deviation σ of the desired Gaussian filter, we can determine a_m^+, b_m^+, a_m^- and b_m^-. Using this method to filter a given signal $x[n]$ is then to compute

$$y[n] = \frac{1}{\sigma\sqrt{2\pi}}\left(y_+[n] + y_-[n]\right)$$

where

$$y_+[n] = \sum_{m=0}^{3} a_m^+ x[n-m] - \sum_{m=1}^{4} b_m^+ y^+[n-m]$$
$$y_-[n] = \sum_{m=1}^{4} a_m^- x[n+m] - \sum_{m=1}^{4} b_m^- y^-[n+m]$$

In other words, if $x[n]$ is a row (or a column) of an image, we compute e

$y_+[n]$ by iterating over $x[n]$ from left to right (top to bottom) and

$y_-[n]$ by iterating over $x[n]$ from right to left (bottom to top).

Per pixel, this requires only a fixed number of operations, regardless of our choice of σ. Thus the computational effort per pixel is $\mathcal{O}(1)$.
An improved (even faster) method for recursive Gaussian filtering can be found in [**young1995recursive**].

Summary

To summarize, Gaussian filtering can be done with a pixel effort of

- $\mathcal{O}(m^2)$ - using naive 2D convolutions and
- $\mathcal{O}(m)$ - exploiting separability.

2.3.2 Mean filter

Definition 3.4: Mean filters

A mean filter $g(x)$ can be defined as

$$\frac{1}{mn}\begin{bmatrix} 1 & 1 & \dots & 1 \\ 1 & 1 & \dots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \dots & 1 \end{bmatrix}$$

Convolution with it assigns every pixel the *average intensity value* of all pixels in its $m \times n$ neighborhood.

Notice that the brute-force effort per pixel of mean filtering is $\mathcal{O}(mn)$. Using intelligent pre-processing, the effort per pixel can be reduced to $\mathcal{O}(1)$!

Integrals of integrals

Given an $N \times M$ image $f[x, y]$, we often consider **integrals**

$$c[x, y, n, m] = \sum_{u=x-\frac{n}{2}}^{x+\frac{n}{2}} \sum_{v=y-\frac{n}{2}}^{y+\frac{n}{2}} h\Big[f[u, v]\Big]$$

The function h and thus c may be scalar- or vector-valued.

For the mean filter, we simply have

$$h\Big[f[u, v]\Big] = \frac{f[u, v]}{mn}$$

In this setting, it is often beneficial to consider **integrals of integrals**

$$C[x, y] = \sum_{u=0}^{x} \sum_{v=0}^{y} c[u, v]$$

where

$$c[u, v] = c[u, v, 1, 1] = h\Big[f[u, v]\Big]$$

us computed from only a single pixel.

Although these integrals of integrals necessitate an additional iteration over the image, they can ease computational burden. Because once an integral image $C[x, y]$ is available, its sub-integrals $c[x, y]$ can be computed in constant time.

Consider this:

$$\begin{aligned} c[x, y, n, m] = \quad & C\Big[x + \frac{n}{2}, y + \frac{m}{2}\Big] \\ & -C\Big[x + \frac{n}{2}, y - \frac{m}{2}\Big] \\ & -C\Big[x - \frac{n}{2}, y + \frac{m}{2}\Big] \\ & +C\Big[x - \frac{n}{2}, y - \frac{m}{2}\Big] \\ =: \quad & \delta + \alpha - \beta - \gamma \end{aligned}$$

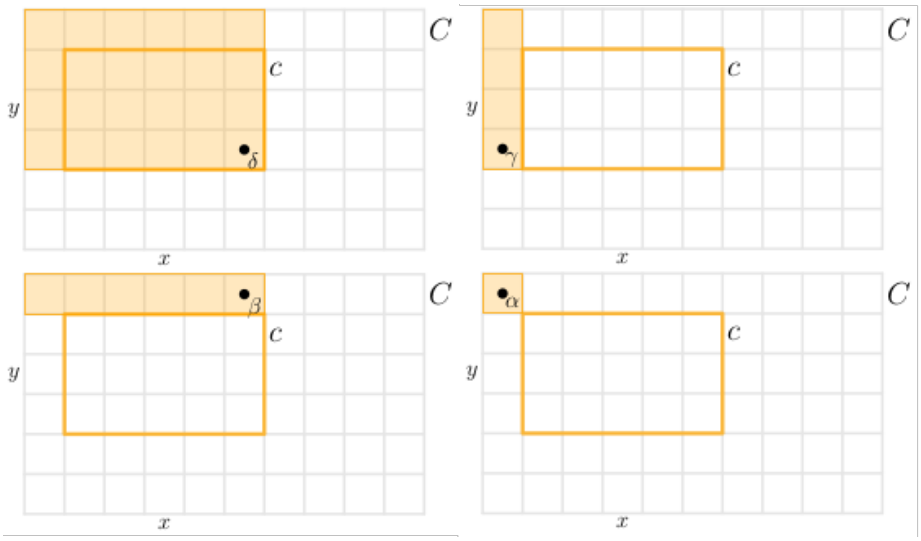

Figure 2.10 – Illustration of the computation using the sub-integrals.

The integrals of integrals can be computed efficiently with the following *dynamic program*:

$$C[x,y] = f[x,y] + C[x,y-1] + C[x-1,y] - C[x-1,y-1]$$

Integral images play a key role in rapid face detection algorithms [2]. The idea of integral images can be generalized, e.g. to *integral histograms*.

Summary

To summarize, mean filtering can be done with a pixel effort of

- $\mathcal{O}(m^2)$ - using naive 2D convolutions,
- $\mathcal{O}(m)$ - exploiting separability and
- $\mathcal{O}(1)$ - using integral images.

2.4 Non-linear filters

2.4.1 Bilateral filter

Bilateral filtering is a rather recent idea [1], [3]. Given an image $f[x,y]$ and an $n \times m$ neighborhood, compute the new image as

$$h[x,y] = \gamma[x,y] \sum_{u=-\frac{n}{2}}^{\frac{n}{2}} \sum_{v=-\frac{m}{2}}^{\frac{m}{2}} g_\rho\Big[f[x,y] - f[x-u,y-v]\Big] \cdot G_\sigma[u,v] \cdot f[x-u,y-v]$$
$$= \gamma[x,y] \sum_{u=-\frac{n}{2}}^{\frac{n}{2}} \sum_{v=-\frac{m}{2}}^{\frac{m}{2}} H[x,y,u,v] \cdot f[x-u,y-v]$$

where g_ρ and G_σ are Gaussians with standard deviations ρ and σ and γ is a location dependent normalization factor:

$$\gamma[x,y] = \frac{1}{\sum_{u,v} H[x,y,u,v]}$$

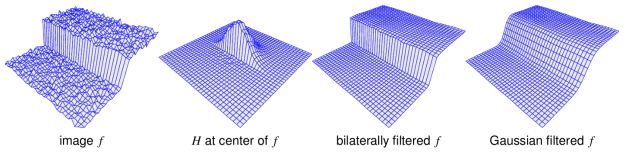

Figure 2.11 – Example of bilateral filtering.

2.4.2 Median filter

Given an 2D image f, consider the $m \times n$ neighborhood of pixel p at coordinate $[x,y]$

$$\mathcal{N}_{xy} := \Big\{ f[i,j] |\, x - \frac{m}{2} \leq i \leq x + \frac{m}{2} \wedge y - \frac{n}{2} \leq j \leq y + \frac{n}{2} \Big\} := \{p_0, p_1, \ldots, p_{q-1}\}$$

where $q = mn$.
Now, assume that we order the set $\mathcal{N}_{xy}$ ascendingly:

$$\mathcal{R}_{xy} := \Big\{ r_0, r_1, \ldots, r_{q-1} |\, r_i \leq r_{i+1} \ \wedge \ r_i \in \mathcal{N}_{xy} \Big\}$$

Then the element $r_{\frac{q-1}{2}}$ is called the **median** of the neighborhood $\mathcal{N}_{xy}$.

If we compute the local median for every coordinate $[x, y]$ in f to create a new image g, the operation $g[x, y] \leftarrow r_{\frac{q-1}{2}}$ is called **median filtering**.

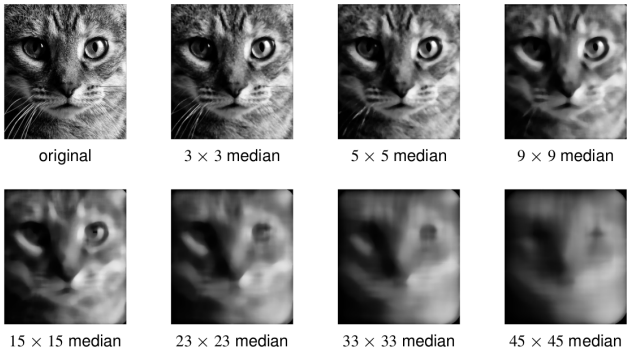

Figure 2.12 – Example of median filtering with different sizes of neighborhoods.

Algorithm 1 Naive Median filtering

Input: an image f of size $M \times N$,
Filter dimensions m and n (size of the considered pixel neighborhood).
Output: the median filtered image g of f.

1: **for** $y = \frac{m}{2}, \ldots, M - \frac{m}{2}$ **do**
2: **for** $y = \frac{n}{2}, \ldots, N - \frac{n}{2}$ **do**
3: $\mathcal{N}_{xy} = \emptyset$
4: **for** $j = -\frac{m}{2}, \ldots, \frac{m}{2}$ **do**
5: **for** $i = -\frac{n}{2}, \ldots, \frac{n}{2}$ **do**
6: $\mathcal{N}_{xy} = \mathcal{N}_{xy} \cup \left\{ f[y + i, x + i] \right\}$
7: $\mathcal{R}_{xy} = \text{qsort}(\mathcal{N}_{xy})$
8: $g[y, x] \leftarrow r_{\frac{q-1}{2}}$

Runtime: Notice that the neighborhoods at the borders of the image are smaller. The average effort per pixel is $\mathcal{O}\big(mn \log(mn)\big)$. If mn is large, this is rather slow.

A more efficient computation can make use of the fact, that the median is the 0.5 quantile of a given distribution. A histogram $H[i]$ is used to count objects, for example the number of occurrences of an intensity i in an image. When

normalized such that its entries sum to 1, a histogram becomes a *probability mass function*.

Algorithm 2 Huang Median filtering, 1979

Input: an image f of size $M \times N$,
Filter dimensions m and n (size of the considered pixel neighborhood).
Output: the median filtered image g of f.

1: **for** $y = \frac{m}{2}, \ldots, M - \frac{m}{2}$ **do**
2: **for** $y = \frac{n}{2}, \ldots, N - \frac{n}{2}$ **do**
3: **if** $[y, x]$ is in the upper left corner **then**
4: Initialize histogram H
5: **else**
6: **for** $k = -\frac{m}{2}, \ldots, \frac{m}{2}$ **do**
7: Subtract $f[y + k, x - \frac{n}{2} - 1]$ from H
8: Add $f[y + k, x + \frac{n}{2}]$ to H
9: $q = \sum_i H[i]$
10: $s = 0$
11: **for** $i = 0, \ldots,$ #colors **do**
12: $s = s + H[i]$
13: **if** $s \geq \frac{q}{2}$ **then**
14: median $= i$
15: **break**

Runtime: The effort per pixel is limited by the number of colors (available intensity values). For an 8-bit gray scale image it is:

$$\mathcal{O}(n) + 256 \text{ operations} = \mathcal{O}(n) + \mathcal{O}(1) = \backslash$$

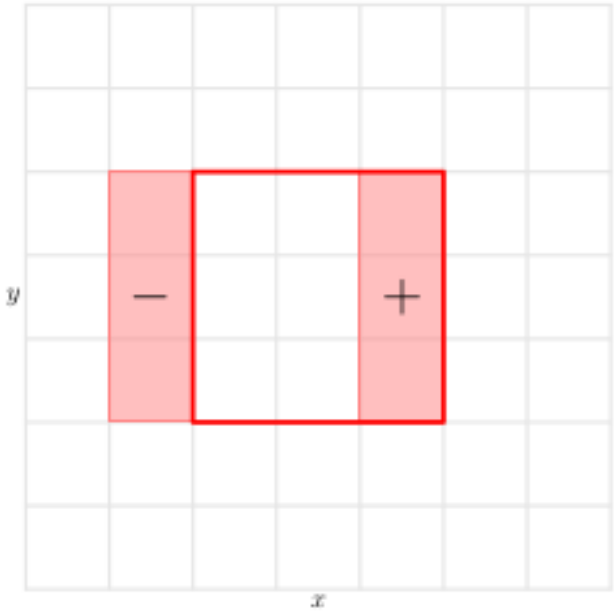

Figure 2.13 – Sketch of the operations in the Huang median filter (algorithm 2).

Furthermore we can make use of the fact, that the histogram of disjoint sets is the sum of their individual histograms:

$$\forall A, B: \qquad A \cap B = \emptyset \implies H[A \cup B] = H[A] + H[B]$$

Thus one may maintain a histogram h_x for each column x of f. This leads to the following algorithm:

Algorithm 3 Perreault & Hébert Median filtering, 2007

Input: an image f of size $M \times N$,
Filter dimensions m and n (size of the considered pixel neighborhood).
Output: the median filtered image g of f.

1: **for** $y = \frac{m}{2}, \ldots, M - \frac{m}{2}$ **do**
2: **for** $y = \frac{n}{2}, \ldots, N - \frac{n}{2}$ **do**
3: **if** y is in the uppermost row **then**
4: Initialize a histogram h_x for the column x
5: **if** $[y, x]$ is in the upper left corner **then**
6: Initialize H
7: **else**
8: Subtract $f[y - \frac{m}{2} - 1, x + \frac{n}{2}]$ from $h_{x+\frac{n}{2}}$
9: Add $f[y + \frac{m}{2}, x + \frac{n}{2}]$ to $h_{x+\frac{n}{2}}$
10: $H = H - h_{x-\frac{n}{2}-1} + h_{x+\frac{n}{2}}$
11: $s = 0$
12: **for** $i = 0, \ldots,$ #colors **do**
13: $s = s + H[i]$
14: **if** $s \geq \frac{q}{2}$ **then**
15: median $= i$
16: **break**

Runtime: The effort per pixel is limited by the number of colors (available intensity values). For an 8-bit gray scale image it is:

- One addition and one subtraction for $h_{x+\frac{n}{2}}$
- 256 additions and 256 subtractions for $H = H - h_{x-\frac{n}{2}-1} + h_{x+\frac{n}{2}}$
- At most 256 operations for computing s.

Thus the runtime is of constant complexity: $\mathcal{O}(1)$.
However note, that there is a large constant factor. Yet, additional tricks keep the efforts low, e.g. multilevel histograms and additions and subtraction by means of bit shifting.

A corresponding implementation is available with OpenCV.

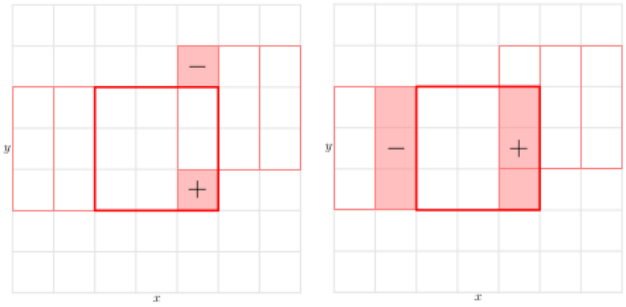

Figure 2.14 – Sketch of the operations in the Perreault & Hébert median filter (algorithm 3).

To summarize, mean filtering can be done with a pixel effort of

- $\mathcal{O}(m^2 \log(m^2))$ - using naive 2D convolutions (with qsort),
- $\mathcal{O}(m)$ - using local histogram updates and
- $\mathcal{O}(1)$ - using clever histogram updates.

2.5 Morphological operations

Definition 5.1: Erosion and dilation

Given $f[x,y]$ and an ordered set $\mathcal{R}_{xy} = \{r_i \in \mathcal{N}_{xy} | \; r_i \leq r_{i+1}\}$, the operation

- $e : g[x,y] \leftarrow r_0$ is called **erosion** (spread of small values) and
- $d : g[x,y] \leftarrow r_{q-1}$ is called **dialation** (spread of large values).

Erosion and dilation are useful in binary image processing. They shrink (erosion) or expand (dilation) shapes. This can also be useful to extract the boundary of a shape. See image 2.15 for example.

Definition 5.2: Opening and closing

Given $f[x,y]$ and an ordered set $\mathcal{R}_{xy} = \{r_i \in \mathcal{N}_{xy} | \; r_i \leq r_{i+1}\}$. Using erosion e and dialation d one can define the operations

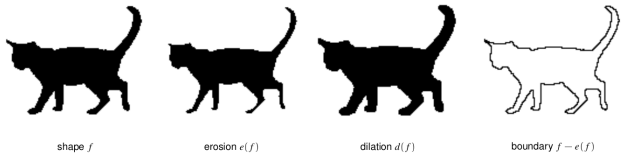

Figure 2.15 – Example of erosion and dilation on a binary image of the shape of a cat.

- **opening** $d \circ e$ and
- **closing** $e \circ d$.

Figure 2.16 – Example of the opening and closing operation on a binary image of the shape of a cat.

Morphological operations are important in areas such as industrial computer vision (machine vision) and in medical image processing. See **[gonzalez2009digital]** and **[jahne2005applications]** for more details.

Chapter 3

GOTO END

Chapter 4

Exercises

4.1 Sheet 0 - Questions in the lecture slides

4.1.1 Q1 - lecture 02

Assume you have an RGB image of resolution 1024×768 with 256 intensity levels per color channel. How many bytes of memory does such an image require when loaded into a computer?

Solution: To store $256 = 2^8$ intensity levels exactly eight bits (one byte) is required. Thus to store $1024 \times 768 = 786,432$ pixels (intensity levels), 786,432 bytes are required.

4.1.2 Q2 - lecture 03

The **divergence** of a continuous multivariate function $f(x_1, x_2, \ldots, x_n)$ is defined as

$$\Delta f = \nabla^2 f = \langle \nabla, \nabla f \rangle = \sum_{i=1}^{n} \frac{\partial^2 f}{\partial x_i^2} \qquad \text{(Eq. 1)}$$

where Δ is called the **Laplace operator**.
Write down an expression for the divergence of a discrete function $g[x, y]$.

Solution:

4.1.3 Q3 - lecture 03

Assume a discrete uni-variate function $g[x]$. What does the following operator do?

$$\text{AM}g[x] = \frac{1}{3}\big(g[x-1] + g[x] + g[x+1]\big) \qquad \text{(Eq. 2)}$$

Solution:

4.1.4 Q4 - lecture 03

Assume a discrete uni-variate function $g[x]$. What does the following operator do?

$$\text{WM}g[x] = \frac{1}{4}\big(g[x-1] + 2g[x] + g[x+1]\big) \qquad \text{(Eq. 3)}$$

Solution:

4.2 Sheet 1

4.2.1 Assignment 1a - Bias of an estimator

...

Solution:

Bibliography

[1] Carlo Tomasi and Roberto Manduchi. "Bilateral filtering for gray and color images". In: Sixth international conference on computer vision (IEEE Cat. No. 98CH36271). IEEE. 1998, pp. 839–846.

[2] Paul Viola and Michael Jones. "Rapid object detection using a boosted cascade of simple features". In: Proceedings of the 2001 IEEE computer society conference on computer vision and Vol. 1. Ieee. 2001, pp. I–I.

[3] Sylvain Paris et al. Bilateral filtering: Theory and applications. Now Publishers Inc, 2009.