IB Paper 8: Photo Editing

Lecture 5: Filtering and Enhancing

N G Kingsbury (given by J Lasenby in 2016)

Signal Processing Group,
Engineering Department,
Cambridge, UK

Easter 2016

Filtering the image: `ph_filter`

- This script filters the image using lowpass or highpass filters or some combination of the two.
- Lowpass filters pass the lower frequency components of the image, corresponding to smooth regions and
- ...reject higher frequency components corresponding to fine details.
- Highpass filters are filters with a gain which is greater for high frequencies than for low frequencies.
- Lowpass filters therefore remove noise and blur, highpass filters enhance edges.

Lowpass filters using Gaussians

- Consider just one type of lowpass filter here – the Gaussian filter. For unit variance (and in the time domain) it takes the form:

$$g(t) = \frac{1}{\sqrt{2\pi}} \exp\left(\frac{-t^2}{2}\right)$$

- The Gaussian is unique in that its Fourier Transform is also a Gaussian:

$$G(\omega) = \int_{-\infty}^{\infty} g(t) \exp(-j\omega t)\, dt = \exp\left(\frac{-\omega^2}{2}\right)$$

- Same shape in the time and freq domains (uniquely)
- $\int_{-\infty}^{\infty} g(t)\, dt = 1$
- As $\omega \longrightarrow 0$, $G(\omega) \longrightarrow 1$ and as $\omega \longrightarrow \infty$, $G(\omega) \longrightarrow 0$. It is therefore a lowpass filter.

2-D filtering

- In 2-D we can (under some circumstances) apply a 1-D filter first to the columns and then to the rows (or vice versa).
- If the image has intensity $I(x, y)$, and our column filter is $g_c(y)$ our 'column filtered' image is formed by convolution

$$I_c(x, y) = \int I(x, y - v)\ g_c(v)\ dv$$

- If row filter is $g_r(x)$, we then convolve this with $I_c(x, y)$:

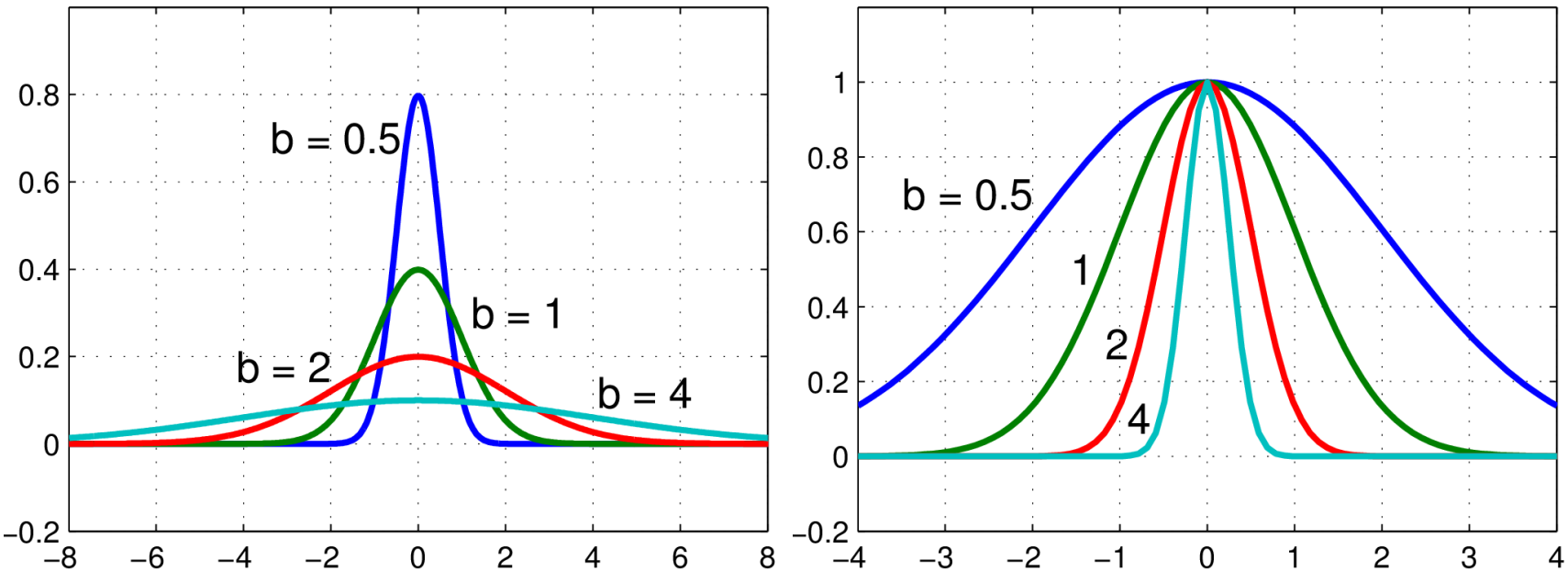
$$\begin{aligned} I_{r,c}(x, y) &= \int I_c(x - u, y)\ g_r(u)\ du \\ &= \int \int I(x - u, y - v)\ g_r(u)\ g_c(v)\ du\ dv \end{aligned}$$

Thus our 2-D separable filter is $g_{r,c}(x, y) = g_r(x)\ g_c(y)$.
This is also often known as the point spread function.

Scaling the Gaussian

- Know from IB Signal and Data Analysis that if $g(x) \leftrightarrow G(\omega)$ then $\frac{1}{\alpha} g\left(\frac{x}{\alpha}\right) \leftrightarrow G(\alpha\omega)$
- ie expand in the frequency domain $\Rightarrow$ shrink in the spatial domain, and vice versa.
- Thus if $g_r(x) = \frac{1}{b}\, g\left(\frac{x}{b}\right)$ and $g_c(y) = \frac{1}{b}\, g\left(\frac{y}{b}\right)$, we know $G_r(\omega_x) = G(b\omega_x)$ and $G_c(\omega_y) = G(b\omega_y)$.
- Thus we retain unit gain at low frequencies and if we have a large uniform patch, we do not introduce lots of extraneous frequencies.

The scaled Gaussian and its Fourier transform



Thus, the larger the value of b the smaller the bandwidth of the filter and the more blurring is produced.

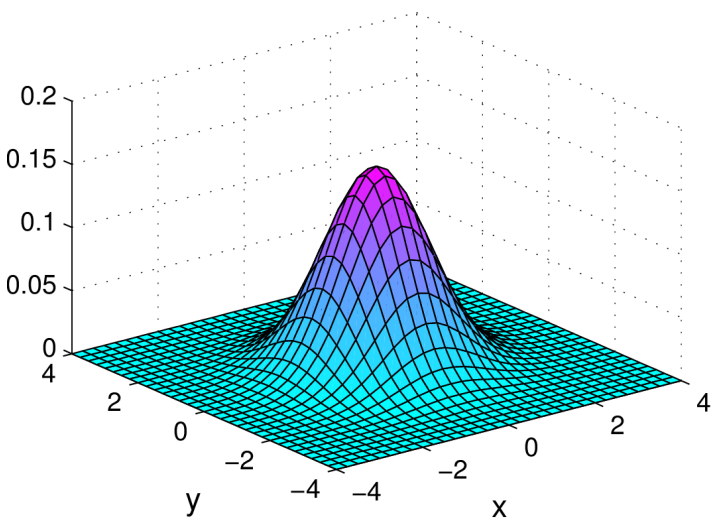
The Gaussian – an isotropic response.

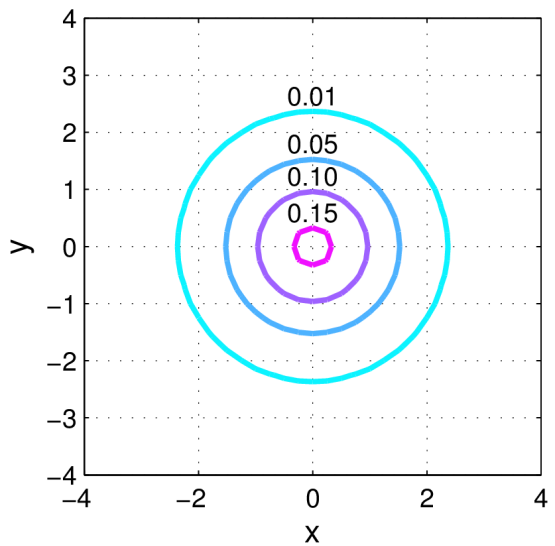
- One of the most important properties of Gaussian filters is that they are isotropic
- It is easy to see that

$$g_{r,c}(x,y) = \frac{1}{2\pi\, b^2}\ \exp\left(\frac{-(x^2+y^2)}{2b^2}\right) = \frac{1}{2\pi\, b^2}\ \exp\left(\frac{-r^2}{2b^2}\right)$$

with $r^2 = x^2 + y^2$, independent of polar angle θ.

- Can show that frequency response/FT is also isotropic.





Why separable filters?

- We can make arbitrary 2D filters which are circularly symmetric but only the Gaussian is separable.
- 1 2D convolution is expensive, whereas 2 1D convolutions (rows then columns) are much cheaper.
- If our 1D filter has length N a convolution requires N mults and adds per pixel. Therefore two applications will give $2N$. Whereas a full 2D convolution with an $N \times N$ filter will need N^2 operations per pixel. The difference can be very considerable!
- Functions which implement these lowpass filtering operations are `im_lowpass`, `gaussian` and `colfilter` (see notes for implementation details, how to deal with boundaries etc).

Highpass filtering with Gaussians

- The most straightforward way of generating a highpass filter is to subtract a lowpass filtered signal from the unfiltered signal. e.g in 1D

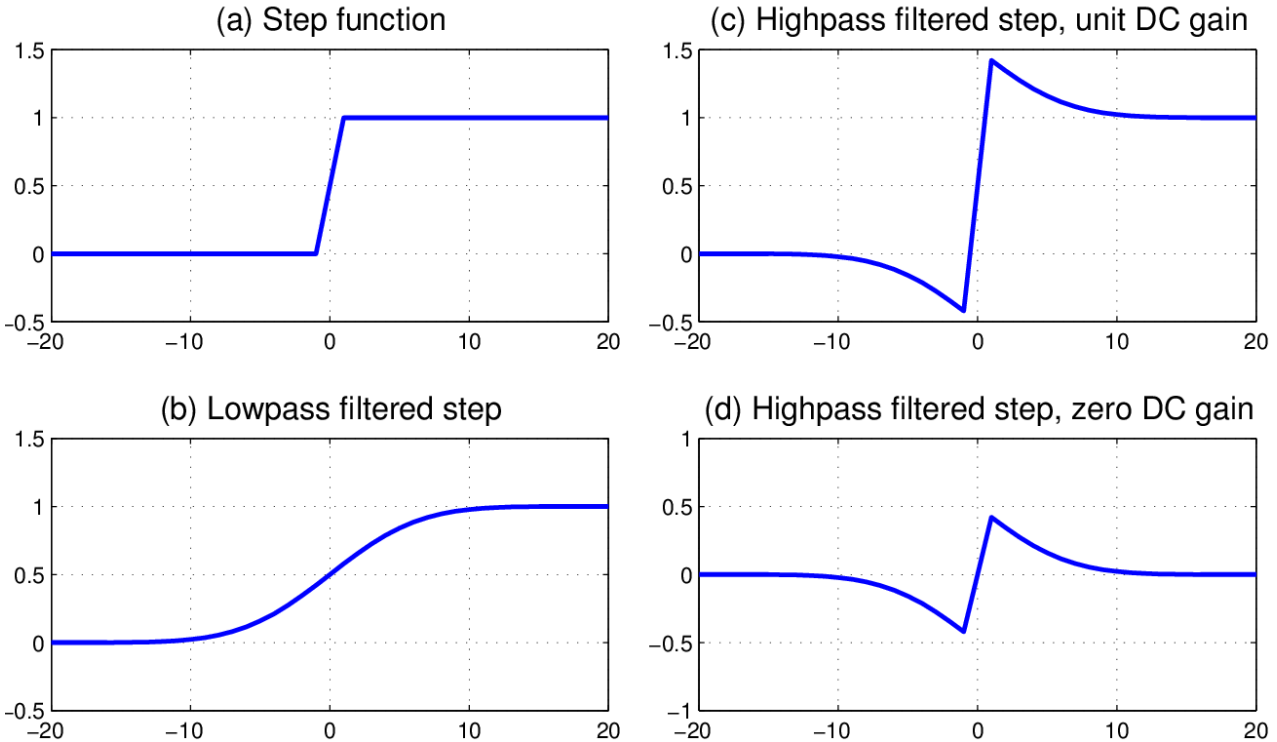
$$Y(\omega) = \alpha X(\omega) - \beta G(\omega) X(\omega) = X(\omega)[\alpha - \beta G(\omega)]$$

- which therefore has a frequency response, $H(\omega)$ of

$$H(\omega) = \frac{Y(\omega)}{X(\omega)} = \alpha - \beta G(\omega)$$

- As $\omega \to 0$, $H(\omega) \to (\alpha - \beta)$ and as $\omega \to \infty$, $H(\omega) \to \alpha$. eg we can have say $(\alpha - \beta) = 1$ and $\alpha = 2$.

Effects of a highpass filter on edges



(c) illustrates $(\alpha - \beta) = 1$, ie unit gain at low frequencies, while
(d) illustrates $(\alpha - \beta) = 0$, ie zero gain at low frequencies.
(c) would be most appropriate for enhancing edges without losing potentially important low frequency information.
(d) would be suitable for detecting edges and ignoring other smoother image regions.

2D highpass filters

- NOTE: while we could form a highpass filter form a pair of 1D highpass filters, this would not be isotropic.
- Instead, we must form the 2D filter by subtracting the 2D lowpass filtered image from the original. This will then be isotropic.
- In the script `ph_filter`, the two parameters that we change are b, the breadth of the 1D Gaussian filters, and the low frequency gain (called DC) G_0 of the highpass filter (not α and β).
- In addition the code introduces an offset term of the form $128(1 - G_0)$ in order to prevent the image from becoming too dark.

Enhancing the image: `ph_enhance`

- Here we attempt to spatially adapt our lowpass and highpass filters.
- In particular, we would like to reduce noise in smooth areas while simultaneously sharpening edges in other areas.
- We will do this via detecting vertical and horizontal edges and using these to spatially adapt the filters.

Detecting Vertical and Horizontal Edges

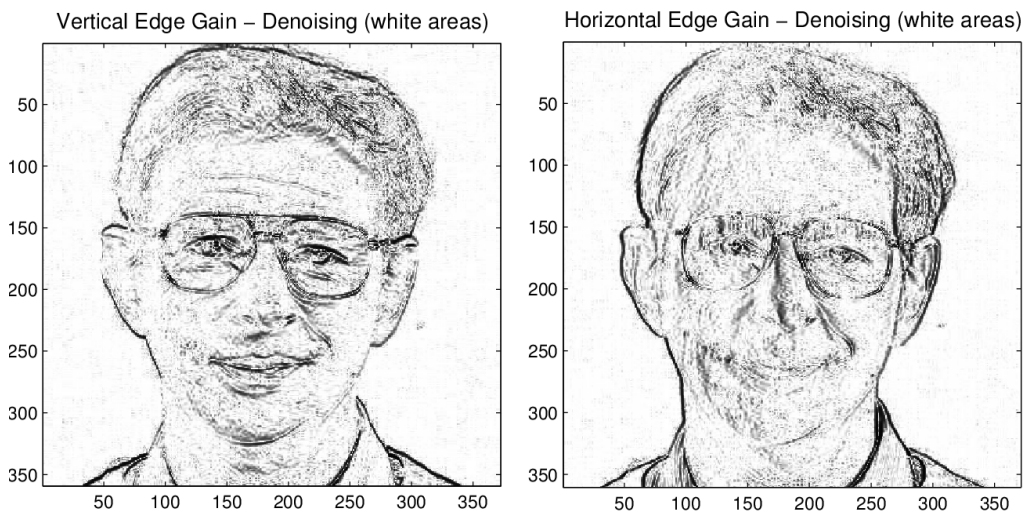
- Since our filtering is separable for maximum efficiency, we detect horizontal and vertical edges separately.
- For horizontal edges we need to measure vertical gradients. First calculate the luminance image, $L1$, and approximate the vertical gradient at a given pixel by smoothing and taking the difference between pixels (vertically adjacent):
  `edv = diff(colfilter(L1,gaussian(0.8)))`
- Similarly for vertical edges:
  `edh = diff(colfilter(L1.',gaussian(0.8))).'`
- From `edv` and `edu` are then used to create two pairs of matrices `gainv` and `gainh` which control the filtering.

Forming the denoising and sharpening gain matrices

Matrices are formed via

$$gain_k = \frac{1}{1 + (ed / thresh_k)^2}$$

where ed is the gradient (either edv or edu), $k = 1$ for denoising and $k = 2$ for sharpening, $thresh_k$ are chosen thresholds.



A similar pair of gain matrices are created for sharpening – these are used to determine where filters are applied.

Adaptive filters for denoising and sharpening

- To do adaptive filtering we can no longer use `colfilter` or `conv2`, as these apply the same filter to the whole image.
- `PhotoEditor` gets around this by designing discrete filters which vary across the image – the variations are controlled by the gain matrices (for denoising and sharpening).
- As before, the adaptive filters for denoising are lowpass filters and the adaptive filters for sharpening are highpass filters.
- Notes give the details (not trivial) of these processes.

Summary

- Section 8 covered filtering of images – lowpass and highpass filters using Gaussians.
- Section 9 covered enhancement of images (in scant detail!) – basically consisting of adaptive low- and highpass filtering.

J. Lasenby (Easter 2016)