Advanced Computer Vision

Chapter 3

Image processing

Presenter: 林保羅 (Paulo L.)

指導教授: 傅楸善 博士

E-mail: d12922028@ntu.edu.tw

Outline

1 Point Operators

2 Linear Filtering

3 More Neighborhood Operators

4 Fourier Transforms

5 Pyramids and Wavelets

6 Geometric Transformations

Image processing operators

- Image processing operators h **generate an image** g from another image f.
- Where $\vec{\mathbf{x}} \in \mathbb{N}^2$ is the **domain** of the image, and $f(\vec{\mathbf{x}})$ is the **range** of the image

Intensity operator

The operator h aims to modify the **pixels' intensities** of an image f or a group of images f_k.

$$g(\vec{\mathbf{x}}) = h\left(\{f_k(\vec{\mathbf{x}})\}_{k=0}^{K-1}\right) \quad \text{Processing multiple images}$$

$$g(\vec{\mathbf{x}}) = h(f(\vec{\mathbf{x}})) \quad \text{Processing a single image}$$

Geometric operator

The operator h aims to modify the **pixels' locations** within the image.

$$g(\vec{\mathbf{x}}) = f(h(\vec{\mathbf{x}}))$$

Point Operators

"Each pixel is modified independently within an image"

Point Operators

Manipulate the intensity of each pixel independently of its neighbors.

- **Pixel transforms:** A predefined function is applied to the intensity of each pixel.
- **Color transforms:** Change the color space to process chromacity and luminosity independently. (RGB $\rightarrow$ YUV)
- **Compositing and matting:** Combining Foreground and Background images regarding their opacity .
- **Histogram equalization:** Generate a pixel transformation function from the image histogram.

Point Operators (cont.)

Pixel transforms

- **Problem:** The objects are poorly visible within the image or their appearance (brightness, contrast) are not good enough.
- **Solution:** Define an intensity mapping function m and use it to the modify pixels' intensities in a straighforward manner.

Intensity mapping function

Assuming intensity values in the range $[0, 1]$:

$$m : [0, 1] \rightarrow [0, 1]$$

$$s = m(k)$$

- k: Original intensity
- s: New intensity

Point Operators (cont.)

Gain and bias (Linear transformation)

$$\text{Constant gain, bias: } g(\vec{\mathbf{x}}) = af(\vec{\mathbf{x}}) + b$$

$$\text{Spatially varying gain, bias: } g(\vec{\mathbf{x}}) = a(\vec{\mathbf{x}})f(\vec{\mathbf{x}}) + b(\vec{\mathbf{x}})$$



(a) **(b)** **(c)**

Figure 1: *Example of a gain and bias transformation: a) Original image; b) Results of applying constant gain and bias; c) Applying spatially dependent gain and bias.* *The lantern looks too bright and lacks contrast. Gain and bias is applied only to the lantern to preserve the appearance of other objects in the scene.*

Point Operators (cont.)

Gamma correction (Power transformation)

$$g(\vec{\mathbf{x}}) = f(\vec{\mathbf{x}})^{\gamma}$$

- **Problem:** *"There is a mismatch between image sensor and human eye luminance responses"*

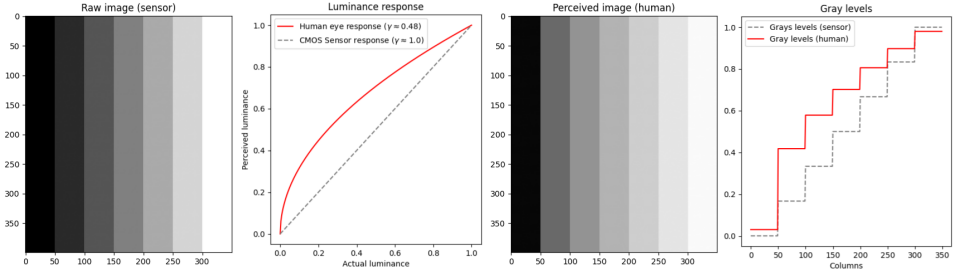


Figure 2: *Image sensor and human eye luminance responses.*

Point Operators (cont.)

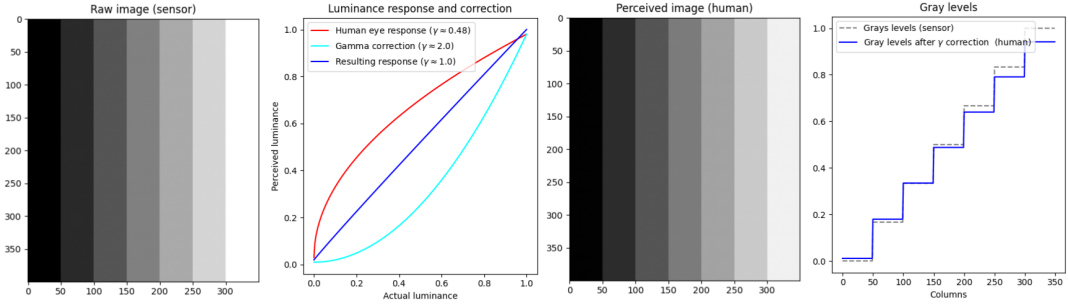


Figure 3: *Application of gamma correction to meet the human eye response.*

Point Operators (cont.)

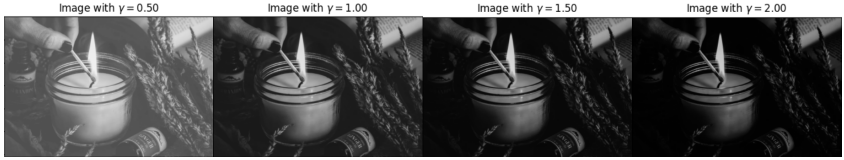


Figure 4: *Gamma correction applied to a sample image for different γ values.*

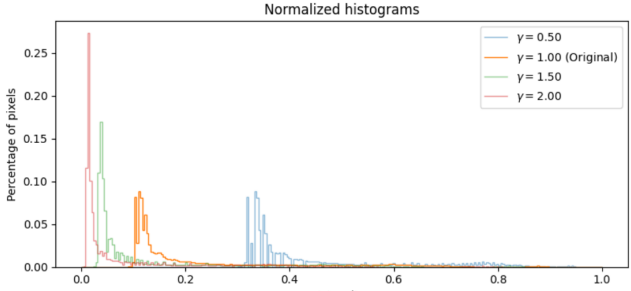


Figure 5: *Normalized histograms of the image after gamma correction.*
Histograms depict changes in brightness and contrast.

Point Operators (cont.)

Linear blend (Convex combination of multiple images)

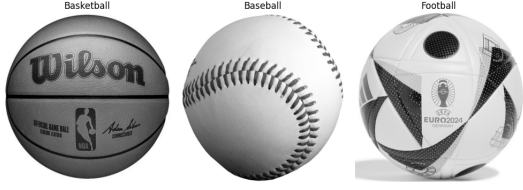
Two images

$$g(\vec{\mathbf{x}}) = \alpha f_1(\vec{\mathbf{x}}) + (1-\alpha) f_2(\vec{\mathbf{x}})$$

Multiple images

$$g(\vec{\mathbf{x}}) = \sum_k \alpha_k f_k(\vec{\mathbf{x}}) \quad s.t. \quad \sum_k \alpha_k = 1 \ ;$$

- α_k: blending coefficients
- $f_k(\vec{\mathbf{x}})$: images to be blended.



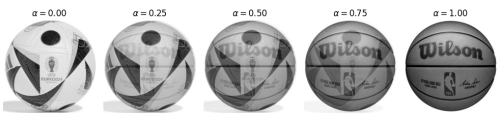
Point Operators (cont.)



(a)



(b)



(c)

Figure 6: *Examples of linear blending between two images.*



Figure 7: *Examples of linear blending between three images.*

Point Operators (cont.)

Color transforms

- In digital images, **color information** is represented in a 3D **color space** (cartesian and non-cartesian).
- RGB[1] is the mostly used color space in image processing, but:
  - All its channels convey **luminance and chromaticity information**.
  - If we want to process luminance and chromaticity separately we should process the image in other color spaces (YUV, CIELAB, HSV[2]).
- **Color transforms**: Change the space used to represent color information by applying linear or non-linear transformations.

$$I_{rgb} \leftrightarrow I_{yuv} : \text{Linear}$$

$$I_{rgb} \leftrightarrow I_{hsv} : \text{non-linear}$$

[1] RGB: Red, Green, Blue

[2] HSV: Hue, Saturation, Value

Point Operators (cont.)



- **Problem:** Applying pixel transforms (e.g. gain-and-bias) to an RGB image changes the appearance of the scene in the image.
- **Solution:** Convert the RGB image to another color space, apply pixel transformations, convert back to RGB.

$$I_{rgb} \xrightarrow{\text{convert}} I_{yuv} \xrightarrow{\text{transform}} I'_{yuv} \xrightarrow{\text{convert}} I'_{rgb}$$

Point Operators (cont.)

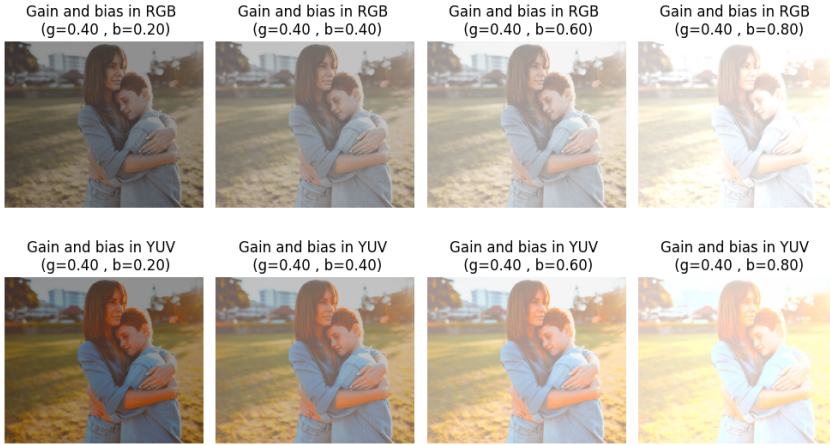


Figure 8: *Gain and bias transformation applied to a color image. Top: Using RGB; Bottom: Using YUV.*
The original image turns grayish at some parts when applying gain and bias in RGB.

Point Operators (cont.)

Compositing and matting

- **Opacity** (α): Additional channel that is multiplied to an image to attenuate objects in the background. A region with $\alpha = 1.0$ is **fully "visible"**, while $\alpha = 0.0$ makes the region **"invisible"**.
- **Compositing** : Process of inserting **foreground regions** from a image F into a **background image** B without visible artifacts
- **Matting** : The process of extracting the α channel from the original image.

$$C = \alpha F + (1 - \alpha)B$$

Computing the opacity (matting) from raw images requires a more complex process than compositing.

Point Operators (cont.)

Compositing when the alpha channel is available (RGBA image).



Figure 9: *Example of compositing when opacity information is available: (a) Foreground image; (b) Foreground opacity channel; (c) Background image; (d) Final composed image.*

Point Operators (cont.)

Straightforward matting using a green screen.

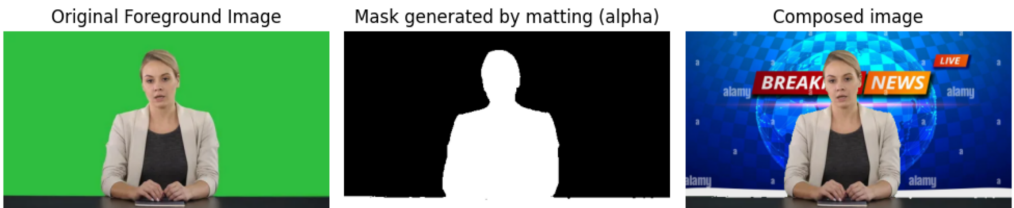


Figure 10: *Example of using a green screen to perform matting from a regular image.*

Point Operators (cont.)

Histogram equalization (HE)

Finds an **intensity mapping function** $s = m(k)$ such that the histogram of the resulting image is flat.

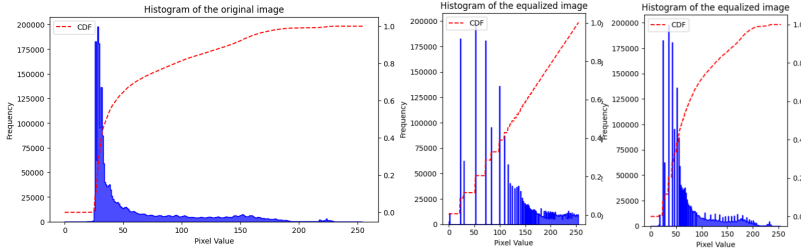
- **Problem**: We have a lot of images and don't know which intensity mapping function may be suitable for each one.
- **Solution**: Generate a intensity mapping function m by processing the image intensity statistics (histogram).

Point Operators (cont.)



(a) (b) (c)

Figure 11: *Application of histogram equalization: (a) Original image; (b) Using conventional histogram equalization; (c) Using histogram equalization and mapping blending.*



(a) (b) (c)

Figure 12: *Histograms of the images in Fig. 11.*

Point Operators (cont.)

Algorithm 1: Histogram Equalization (Global)

Input: Source image $\mathbf{I_s} \in \mathbb{N}^{W \times H}$
Result: Equalized Image $\mathbf{I_{eq}}$

▷ Step 1: CDF computation
for $k = 0, 1, \ldots, K$ **do**
  $H(k) = \sum_{i=0}^{k} \underbrace{\sum_{\vec{x}} \mathbb{1}_{\mathbf{I_s}(\vec{x})==i}}_{\text{histogram}}$
end
$\tilde{H}(k) := H(k)/HW$
▷ Step 2: Intensity mapping function
$m(k) := \lfloor K\tilde{H}(k) \rfloor$
▷ mapping blending (optional)
$m(k) := \lfloor K \left(\alpha \tilde{H}(k) + (1-\alpha)k \right) \rfloor$
▷ Step 3: Transform intensities
foreach $\vec{x}, k$ **do**
  $\mathbf{I_{eq}}(\vec{x}) = m(k)\mathbb{1}_{\mathbf{I_s}(\vec{x})==k}$
end

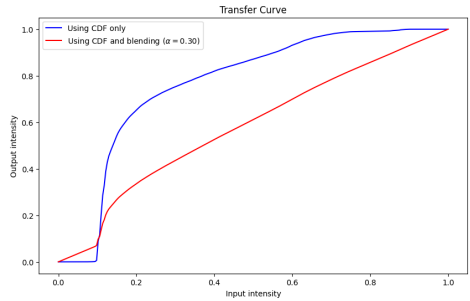


Figure 13: *Intensity mappings functions generated with histogram equalization.* *The transfer curve for HE changes abruptly near 0.1, while HE+Blending softens the change.*

k: Original intensity; K: Number of intensity levels.
$\tilde{H}(k)$: Cumulative Distribution Function (CDF).
$m(k)$: Intensity mapping function.

Point Operators (cont.)

Histogram equalization (Global)

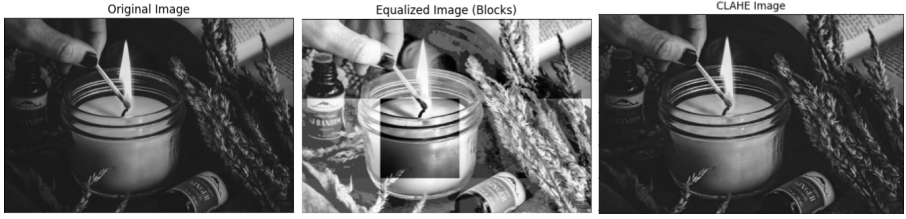
- **Problem**: Since the background of the image is too dark, after applying histogram equalization:
  - The objects in the foreground look quite bright.
  - Dark objects depict "saturation" artifacts.
- **Solution** : Adopt a local approach to perform histogram equalization.



Point Operators (cont.)

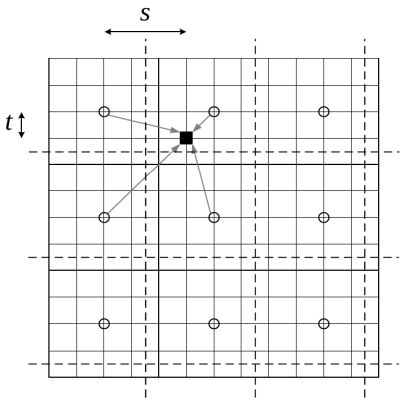
Histogram equalization (Local)

- **Naive solution**: Divide the image into blocks and apply histogram equalization within each block
- **Results**: The resulting image exhibits a lot of block-shaped artifacts.
- **Better approach**: Divide the image into blocks, compute histograms and mapping functions for each block, combine the mapping functions of the blocks in a linear way (e.g. AHE, CLAHE).



AHE: Adaptive Histogram Equalization
CLAHE: Contrast Limited Adaptive Histogram Equalization

Point Operators (cont.)



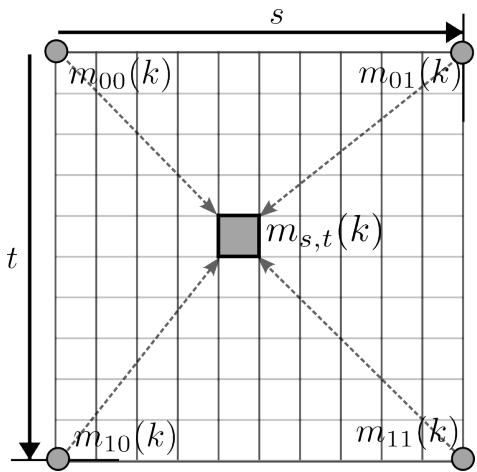
One-way approach (block based)

1. Divide the image into blocks and compute $m(k)$ for each block as done for global histogram equalization (steps 1, 2).
2. Apply **bilinear blending** from 4 adjacent mapping functions and modify pixel intensities.

"What is bilinear blending?"
"How it is applied to mapping functions?"

$m(k)$ is the mapping function from histogram equalization
(s, t) are relative coordinates between 4 adjacent block centers.

Point Operators (cont.)

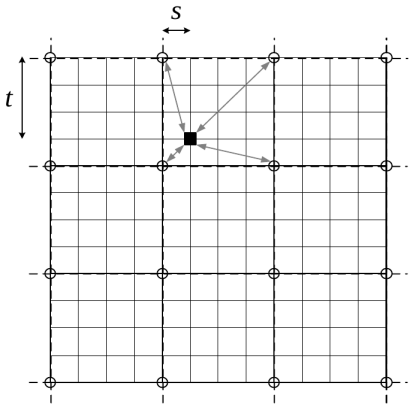


Mapping bilinear blending

$$m_{s,t}(k) = (1-s)(1-t)m_{00}(k) + s(1-t)m_{10}(k) + (1-s)tm_{01}(k) + stm_{11}(k)$$

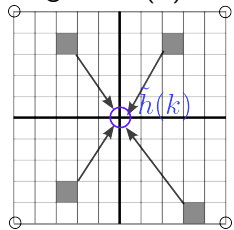
- $s, t \in [0, 1]$ are relative coordinates within each block.
- $m_{00}, m_{01}, m_{10}, m_{11}$ are 4 adjacent mapping functions obtained from blocks' histograms.

Point Operators (cont.)



Two-way approach (corner based)

1. Divide the image into blocks and compute normalized histograms $\tilde{h}(k)$ as follows:



2. Compute $m(k)$ for each corner based on $\tilde{h}(k)$.
3. Apply bilinear blending to the mapping functions and modify pixel intensities.

Note: In this approach, corners' mapping functions are obtained by computing histogram from the 4 blocks surrounding a corner.

Linear Filtering

"Convolution and correlation: A pixel's intensity is determined by its neighbors"

Linear Filtering

- Convolution and correlation
- Separable filtering
- Band-pass and steerable filtering

Linear Filtering

Linear Shift Invariant (LSI) operators

- **Linear operator**: $h\left(\alpha f_a(\vec{x}) + \beta f_b(\vec{x})\right) = \alpha h\left(f_a(\vec{x})\right) + \beta h\left(f_b(\vec{x})\right)$
- **Shift invariant operator**:
  - Shift operation: $g(\vec{x}) = f(\vec{x} + \vec{x_0})$
  - Shift invariant condition: $(h \circ g)(\vec{x}) = (h \circ f)(\vec{x} + \vec{x_0})$
  - An operator h is **shift invariant** if applying the operator to the **shifted image** $g(\vec{x})$ gives the same results as applying h to the **original image** $f(\vec{x})$ and then shifting the resulting image.

$f_a(\vec{x}), f_b(\vec{x})$ are two different images.
h is an operator for images.
α, β are scalars.

Linear Filtering (cont.)

Convolution and Correlation

Convolution and correlation are LSI operators

Let's assume an image $f(i,j) \in \mathbb{R}^{H \times W}$, and a kernel $k(i,j) \in \mathbb{R}^{h \times w}$:

Correlation ($g = f \otimes h$)

$$g(i,j) = \sum_{(u,v)} = f(i+u, j+v)k(u,v)$$

Convolution ($g = f * h$)

$$g(i,j) = \sum_{u,v} \underbrace{f(i-u, j-v)}_{\text{shifting the image}} k(u,v) = \sum_{u,v} f(u,v) \underbrace{k(i-u, j-v)}_{\text{shifting the kernel}}$$

When shifting the image: The range of (u,v) goes from $(\lceil \frac{-h}{2} \rceil, \lceil \frac{-w}{2} \rceil)$ to $(\lfloor \frac{h}{2} \rfloor, \lfloor \frac{w}{2} \rfloor)$

When shifting the kernel: The range of (u,v) goes from $(i - \lceil \frac{-h}{2} \rceil, j - \lceil \frac{-w}{2} \rceil)$ to $(i - \lfloor \frac{h}{2} \rfloor, j - \lfloor \frac{w}{2} \rfloor)$

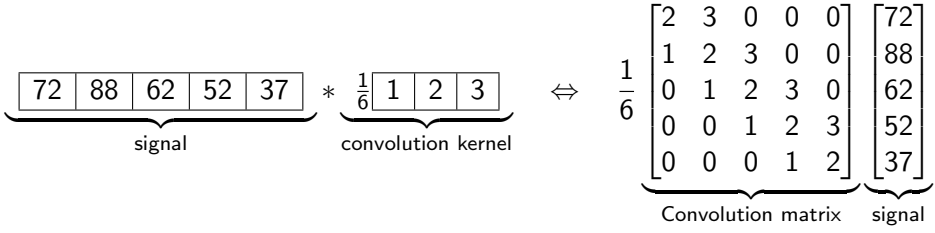
Linear Filtering (cont.)

Other representations of convolution

Shifted impulse responses

$$g(i,j) = \sum_{u,v} f(i-u, j-v) \sum_{\tau,\rho} \underbrace{\delta(u-\tau, v-\rho)}_{\text{shifted impulse}} k(\tau,\rho)$$

Sparse matrix multiplication



$\delta(\cdot)$ is the impulse function (Kronecker delta).

Linear Filtering (cont.)

Image Padding

- **Problem:** The convolution and correlation operations yield invalid values near the borders. To avoid invalid values, the size of the resulting image is shrinked.
- **Solution:** Apply padding (extend the image over the borders) to the original image before applying convolution.

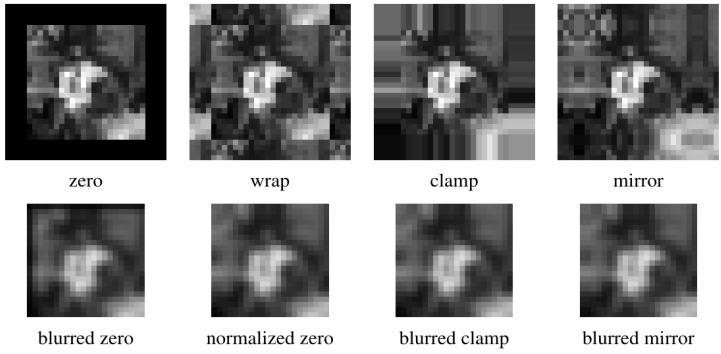


Figure 14: *Image padding techniques.*

Linear Filtering (cont.)

Separable filters

- **Problem:** Applying 2D convolution with a large kernel ($N = 100$) to the frames of a 4K video (3840 x 2160 pixels) takes too much time. Performing a 2D convolution requires N^2 (multiply-add) operations per pixel.
- **Solution:** Decompose the 2D filter kernel into two 1D kernels. Performing a 1D convolution requires N operations per pixel (total of $2N$ operations/pixel).

Linear Filtering (cont.)

- The two-dimensional kernel K corresponding to successive convolution with a horizontal kernel $\vec{\mathbf{h}}$ and a vertical kernel $\vec{\mathbf{v}}$ is the outer product of the two kernels.

$$K = \vec{\mathbf{v}}\vec{\mathbf{h}}^{\mathsf{T}}$$

- For kernel with shape $N \times N$ the number of multiply-add operations per pixel is reduced from N^2 to $2N$.

Linear Filtering (cont.)

Examples of separable filters

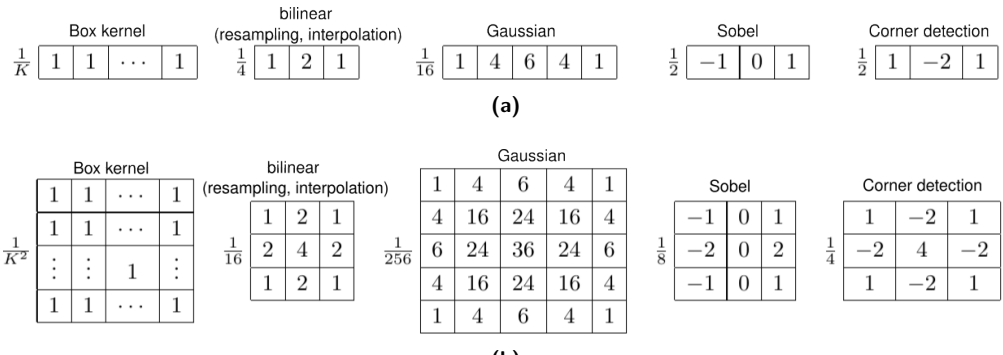


Figure 15: *Examples of separable filters: (a) 1D kernels; (b) 2D kernels.*

Linear Filtering (cont.)

Separability criterion

- Use SVD to **decompose the kernel** $K = \sum_i \sigma_i \vec{\mathbf{u}}_i \vec{\mathbf{v}}_i^\top$. Where σ_i are the singular values, $\vec{\mathbf{u}}_i$ are the left singular vectors, $\vec{\mathbf{v}}_i$ are the right singular vectors.
- The **kernel is separable** if it has only **one non-zero singular value** σ_0.

Separable filter approximation (non-separable $\rightarrow$ separable)

1. Decompose the kernel using SVD $K = \sum_{r=1}^{R} \sigma_r \vec{\mathbf{u}}_r \vec{\mathbf{v}}_r^\top$. $R = \text{rank}(K)$
2. Take the T highest non-zero singular values (e.g. 1, 2).
3. Approximation of the original as $K' = \sum_{t=1}^{T} \sigma_t \vec{\mathbf{u}}_t \vec{\mathbf{v}}_t^\top \quad s.t. \quad T \leq R$
4. Convolution is linear: Convolve the image with each component $\sigma_t \vec{\mathbf{u}}_t \vec{\mathbf{v}}_t^\top$ and sum the results.

SVD: Singular Value Decomposition

Linear Filtering (cont.)

Summed area table (aka Integral image)

- An image is going to be **repeatedly convolved** with different **box filters** (different sizes N) $\rightarrow$ we might need N^2 or $2N$ adding operations per pixel.
- We can reduce the number of operations to just 4.
- Summed area table $s(i,j) = \sum_{k=0}^{i} \sum_{l=0}^{j} f(k,l)$

Remember: "Convolution with a box filter requires computing the sum of box-shaped image region."

Sum of a box-shaped image region:

$$S(i_0 : i_1, j_0 : j_1) = \sum_{k=i_0}^{i_1} \sum_{l=j_0}^{j_1} f(k,l) = s(i_1,j_1) - s(i_1,j_0-1) - s(i_0-1,j_1) + s(i_0-1,j_0-1)$$

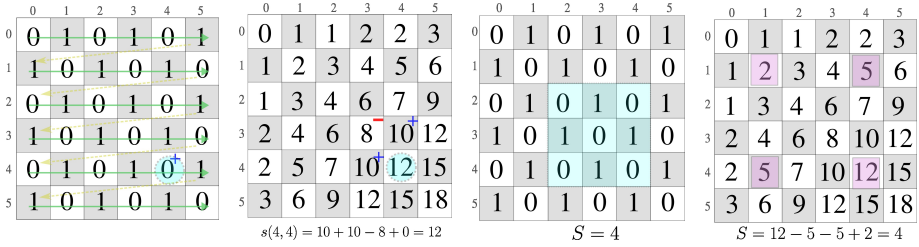
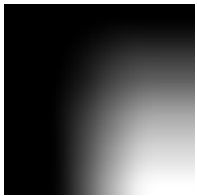


Figure 16: *Summed area tables: (a) original image; (b) summed area table; (c) Area sum using the original image; (d) Area sum using the summed area table.*

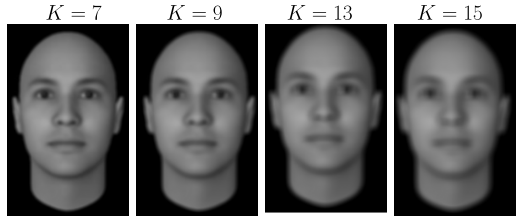


(a)

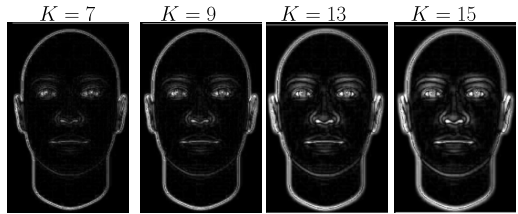


(b)

Figure 17: *Sample image summed area table: (a) Original image; (b) Summed area table.*



(a)



(b)

Figure 18: *Filtered images using summed table: (a) Original image convolved with different box filters; (b) Absolute difference between filtered images and the original one (simple edge detector).*

Linear Filtering (cont.)

Filtering may be confused with blurring – or denoising – an image (low-pass filter), but filters can be:

- **Band-pass**: Filter low and high frequencies (e.g. edges, corners)
- **Oriented**: Respond differently to the same structures (e.g edges) but at different angles.

Band-pass steerable filters

- The Sobel and corner operators are simple examples of **band-pass** and **oriented filters**.
- Change the response of a filter to different orientations $\rightarrow$ **steerable filter**.
- More **sophisticated kernels** can be created by first smoothing the image with a Gaussian filter, and then taking the first or second derivatives.

Linear Filtering (cont.)

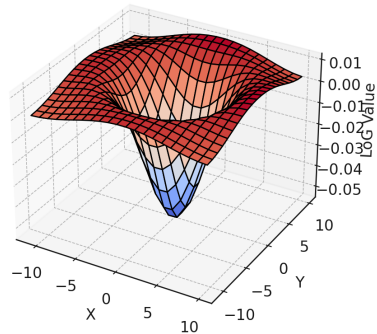
Laplacian of Gaussian (isotropic band-pass filter)

$$G(x, y; \sigma) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right)$$

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

$$\nabla^2 G(x, y; \sigma) = \left(\frac{x^2 + y^2 - 2\sigma^2}{\sigma^4}\right) G(x, y; \sigma)$$

Laplacian of Gaussian (LoG) Kernel



- $\nabla^2(G * f)$ is equivalent to $\nabla^2 G * f$
- It can detect edges and blobs in an image (band-pass).
- We cannot steer its response.

Linear Filtering (cont.)

Steerable band-pass filter

Steerable filtering (Filtering the image and steering the response) :

$$\hat{\mathbf{u}} \cdot \nabla(G * f), \quad \hat{\mathbf{u}} = (\cos\theta, \sin\theta)$$

Steerable kernel (Steering the kernel) :

$$G_{\hat{u}} = uG_x + vG_y = u\frac{\partial G}{\partial x} + v\frac{\partial G}{\partial y}$$

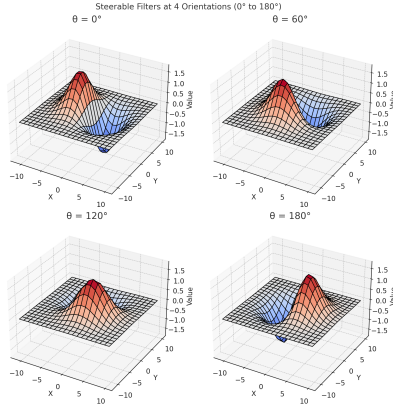


Figure 19: *Example of steerable kernels for 4 different orientations (0°, 60°, 120°, 180°). G is the Gaussian kernel.*

Linear Filtering (cont.)

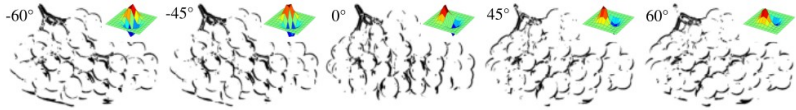


Figure 20: *Example of responses to steerable kernels with 5 different orientations* $(-60^o, -45^o, 0^o, 0^o, 45^o, 60^0)$



(a) **(b)** **(c)**

Figure 21: *Image sharpening using the 5 responses above: (a) Original image; (b) Edges obtained with the 5 filtered images above; (c) Sharpened image by blending the original image with edges.*

More Neighborhood Operators

"Non-linear filters, Morphological operations over binary images"

TOC2

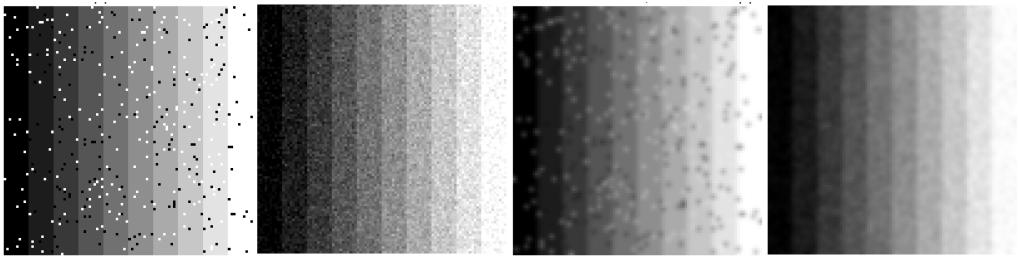
More Neighborhood Operators

1. Non-linear filtering
2. Bilateral filtering
3. Binary Image processing

More Neighborhood Operators

Image Noise

Unwanted random variations in pixel intensity that degrade image quality. It is typically introduced during image acquisition, transmission, or processing.



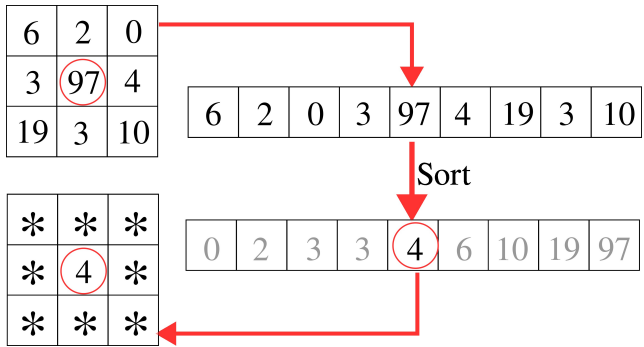
(a) **(b)** **(c)** **(d)**

Figure 22: *Example of the application of the median filter: (a) Image with salt-pepper noise; (b) Image with additive gaussian noise; (c) Box filter applied to Fig. 22a; (c) Box filter applied to Fig. 22b*

More Neighborhood Operators (cont.)

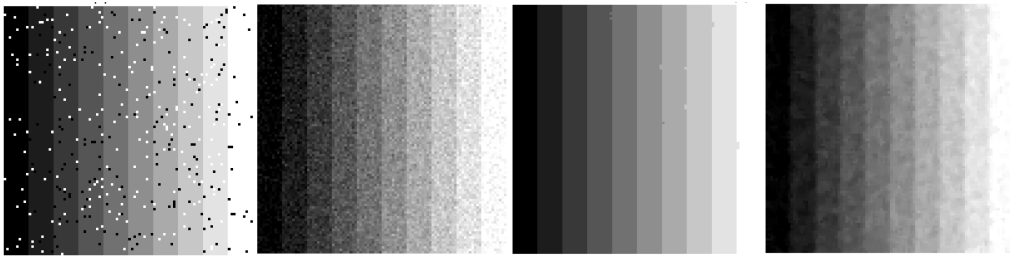
Median filtering

Non-linear edge-preserving filter. It selects the median value from each pixel's neighborhood.



Neighborhood: Pixels surrounding the pixel to be modified.

More Neighborhood Operators (cont.)



(a) *w. salt-pepper noise* **(b)** *w. Gaussian noise* **(c)** *median filter to (a)* **(d)** *median filter to (b)*

Figure 23: *Example of the application of the median filter: (a) Image with salt-pepper noise; (b) Image with additive gaussian noise; (c) Median filter applied to Fig. 23a; (c) Median filter applied to Fig. 23b*

More Neighborhood Operators (cont.)

α-trimmed mean

Sort the entries of a rectangular region $S_{i,j}$. Average the region's entries **disregarding** the **lowest** and **highest** values according to the α fraction.

$$g(i,j) = \frac{1}{|S_{ij}| - 2\lfloor \alpha |S_{ij}| \rfloor} \sum_{(s,t) \in S_{ij}} f(s,t)$$

- $\alpha = 0.5 \rightarrow$ Median filter
- $\alpha = 0.0 \rightarrow$ Mean (box) filter
- $0 < \alpha < 0.5 \rightarrow$ Hybrid mean-median

13	28	33
15	76	32
17	30	42

={13,15,17,28,30,32,33,42,76}

if $\alpha = 0.25 \rightarrow \lfloor \alpha |S_{ij}| \rfloor = 2$

Mean({13,15,17,28,30,32,33,42,76})=28

More Neighborhood Operators (cont.)

Bilateral filtering (leveraging spatial and intensity data)

Assign lower importance to pixels whose intensity values differ too much from the central pixel value and are distant.

$$g(i,j) = \frac{\sum_{k,l} f(k,l) w(i,j,k,j,l)}{\sum_{k,l} w(i,j,k,j,l)}$$

Bilateral weight function: $w(i,j,k,j,l) = d(i,j,k,l) r(i,j,k,l)$

Domain kernel (spatial information)

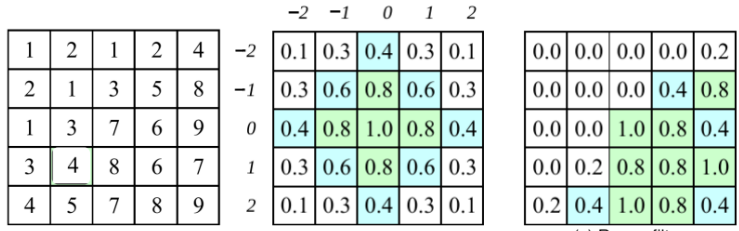
$$d(i,j,k,l) = \exp\left(-\frac{(i-k)^2 + (j-l)^2}{2\sigma_d^2}\right) \longleftarrow \text{ Same for every } (i,j) \text{ pair.}$$

Range kernel (intensity information)

$$r(i,j,k,l) = \exp\left(-\frac{(f(i,j) - f(k,l))^2}{2\sigma_r^2}\right) \longleftarrow \text{ Depends on } (i,j) \text{ pairs and } (k,l) \text{ pairs.}$$

σ_d^2: variance of the domain kernel. σ_r^2: variance of the range kernel.

More Neighborhood Operators (cont.)



1	2	1	2	4	
2	1	3	5	8	
1	3	7	6	9	
3	4	8	6	7	
4	5	7	8	9	

(a) Original image (region)

	−2	−1	0	1	2
−2	0.1	0.3	0.4	0.3	0.1
−1	0.3	0.6	0.8	0.6	0.3
0	0.4	0.8	1.0	0.8	0.4
1	0.3	0.6	0.8	0.6	0.3
2	0.1	0.3	0.4	0.3	0.1

(b) Domain filter

0.0	0.0	0.0	0.0	0.2
0.0	0.0	0.0	0.4	0.8
0.0	0.0	1.0	0.8	0.4
0.0	0.2	0.8	0.8	1.0
0.2	0.4	1.0	0.8	0.4

(c) Range filter

Figure 24: *Domain and range filters used for bilateral filtering: (a) Original image; (b) Domain filter; (c) Range filter.*

More Neighborhood Operators (cont.)

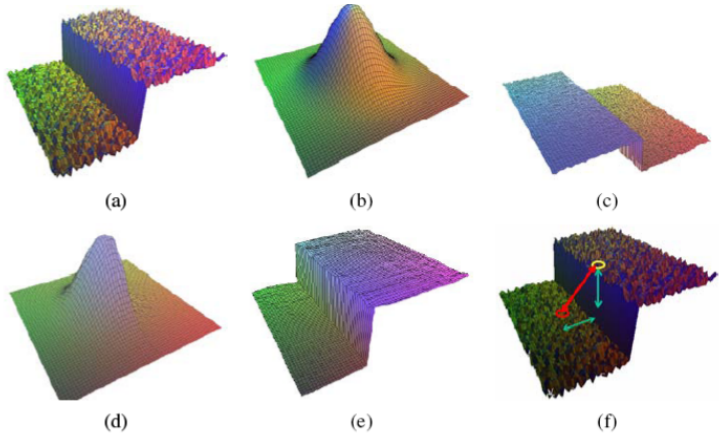
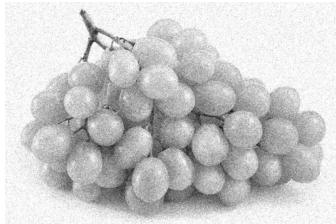


Figure 25: *Bilateral filtering: (a) noisy step edge input; (b) domain filter (Gaussian); (c) range filter (similarity to center pixel value); (d) bilateral filter; (e) filtered step edge output; (f) 3D distance between pixels.*

More Neighborhood Operators (cont.)



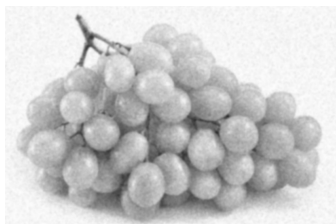
(a)



(b)



(c)



(d)

Figure 26: *Bilateral filtering example: (a) Original image; (b) Image with gaussian noise; (c) Filtering with a gaussian kernel; (d) Filtered image obtained via bilateral filtering.*

More Neighborhood Operators (cont.)

Binary images

- Images whose entries are 1 or 0.
- Such images often occur after a thresholding operation

$$\theta(f,t) = \begin{cases} 1, & \text{if } f \geq t, \\ 0, & \text{otherwise.} \end{cases}$$

- Besides used for thresholding a regular image, $\theta(f,t)$ operator is employed for **performing morphological operations**.

More Neighborhood Operators (cont.)

Morphological operations

- The most common **binary image operations** are called **morphological operations**, because they **change the shape** of the underlying binary objects.
- Morphological operations are **binary convolutions** (binary image, binary kernel) followed by a thresholding operation.

Binary convolution: $c = f \otimes s$

Dilation:	$\text{dilate}(f, s) = \theta(c, 1)$;
Erosion:	$\text{erode}(f, s) = \theta(c, S)$;
Majority:	$\text{maj}(f, s) = \theta(c, S/2)$;
Opening:	$\text{open}(f, s) = \text{dilate}(\text{erode}(f, s), s)$;
Closing:	$\text{close}(f, s) = \text{erode}(\text{dilate}(f, s), s)$.

- f: input binary image.
- s: binary kernel.
- c: Output image (non-binary).
- S: The number of 1-entries in the structuring element (kernel).

$c(i, j)$ counts how many 1-entries in the image f match the 1-entries in the kernel s.

More Neighborhood Operators (cont.)

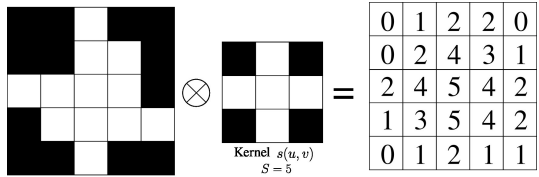
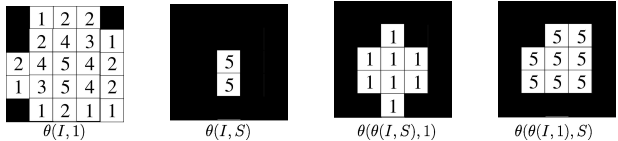


Figure 27: *Example of convolving a binary image with a binary kernel.* *The resulting image is not binary.*



(a) *Dilation* **(b)** *Erosion* **(c)** *Opening* **(d)** *Closing*

Figure 28: *Sample morphological operation results*

For Opening and Closing, binary convolution $c = I \otimes s$ is applied again before thresholding.

More Neighborhood Operators (cont.)

Distance transforms (minimum distance to a zero pixel)

The distance transform $D(i,j)$ of a binary image $b(i,j)$ is defined as:

$$D(i,j) = \min_{k,l\,:\,b(k,l)=0} d(i-k, j-l);$$

Where, $d(k,l)$ is some distance between pixel offsets (e.g. Manhattan, Euclidean)

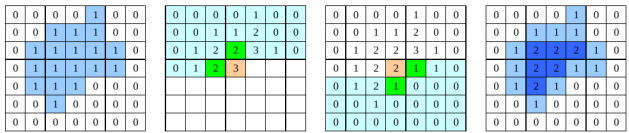


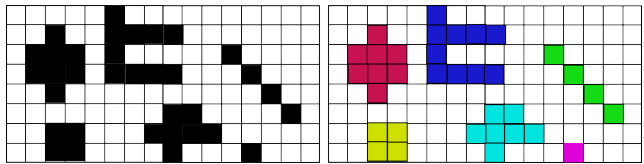
Figure 29: *City block distance transform: (a) original binary image; (b) top to bottom (forward) raster sweep: green values are used to compute the orange value; (c) bottom to top (backward) raster sweep: green values are merged with old orange value; (d) final distance transform.*

Application: Improve the construction of panoramas: blending images of the same scene to generate a bigger picture – Section 8.

More Neighborhood Operators (cont.)

Connected components and component properties

Regions of adjacent pixels that have the same input value or label.



- Area: the number of pixels in the region.
- Perimeter: the number of boundary pixels.
- Centroid: the average x and y values,

$$\bar{x} = \frac{1}{N} \sum_{(x,y) \in R} x, \quad \bar{y} = \frac{1}{N} \sum_{(x,y) \in R} y.$$

- Second moments:

$$M = \sum_{(x,y) \in R} \begin{bmatrix} x - \bar{x} \\ y - \bar{y} \end{bmatrix} \begin{bmatrix} x - \bar{x} & y - \bar{y} \end{bmatrix}$$

Second moments allow to determine main axes lengths and orientation of a connected component.

Fourier Transforms

"Processing an image in the frequency domain and its applications"

TOC2

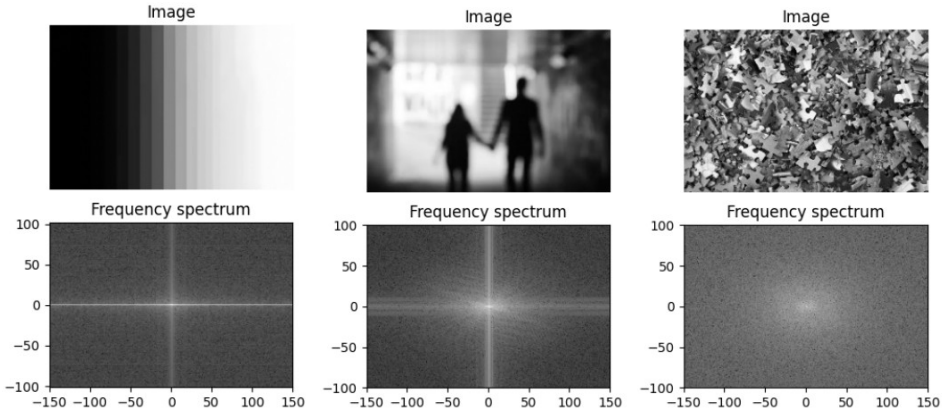
Fourier Transforms

- Fourier Transform and Analysis in the frequency domain
- Two-dimensional Fourier Transforms
- Discrete Cosine Transform (DCT) and data compression

Fourier Transforms

Fourier transform and 2D Fourier transform

- Fourier analysis: Any image (signal) can be represented as series of weighted periodic components (sine, cosine) with specific frequency values. The weights are determined by the Fourier transform.
- Fourier Transforms usage:
  - To **analyze** what a given filter in the spatial domain does to high, medium, and low frequency components (analysis in the frequency domain).
  - Filtering in the frequency domain.
  - **Image compresion**: Block-wise transform + Quantization
- Fourier transform is **reversible** without information loss. FTT $\leftrightarrow$ IFFT


Image
Image
Image
Frequency spectrum
Frequency spectrum
Frequency spectrum
100
50
0
−50
−100
−150 −100 −50 0 50 100 150
(a)
(b)
(c)

Fourier Transforms (cont.)

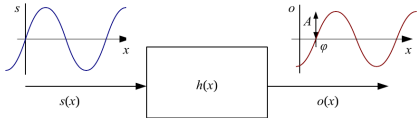


Figure 30: *The Fourier Transform as the response of a filter $h(x)$ to an input sinusoid $s(x) = e^{j\omega x}$ yielding an output sinusoid $o(x) = h(x) * s(x) = Ae^{j(\omega x+\phi)}$*

1D Fourier transform

$$H(k) = \sum_{x=0}^{N-1} h(x) e^{-j\frac{2\pi kx}{N}} \longleftarrow \text{ Complex-valued function}$$

2D Fourier transform (Image with shape $N \times M$)

$$H(k_x, k_y) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} h(x, y) e^{-j2\pi\left(\frac{k_x x}{M} + \frac{k_y y}{N}\right)}$$

$H(k), H(k_x, k_y)$ correspond to the FFT, whereas $h(x), h(x, y)$ are the signals in its original domain. k, k_x, k_y are the frequency indexes.

Fourier Transforms (cont.)

Discrete Cosine Transform (DCT)

1D Discrete Cosine Transform

$$F(k) = \sum_{i=0}^{N-1} \cos\left(\frac{\pi}{N}\left(i + \frac{1}{2}\right)k\right) f(i) \longleftarrow \text{Real-valued function}$$

2D Discrete Cosine Transform (Image with shape $W \times H$)

$$F(k,l) = \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} \cos\left(\frac{\pi}{N}\left(i + \frac{1}{2}\right)k\right) \cos\left(\frac{\pi}{N}\left(j + \frac{1}{2}\right)l\right) f(i,j)$$

$$f(i,j) \xrightarrow{\text{DCT + high-freq. removal}} F(k,l) \xrightarrow{\text{quant.}} F_q[k,l] \xrightarrow{\text{de-quant.}} F'(k,l) \xrightarrow{\text{IDCT}} f'(i,j)$$

The DCT is widely used in today's image and video compression algorithms (e.g. JPEG2000).

Pyramids and Wavelets

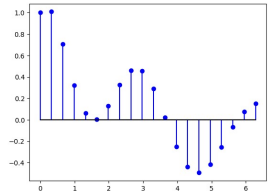
"Multiresolution processing, Interpolation, Decimation"

TOC2

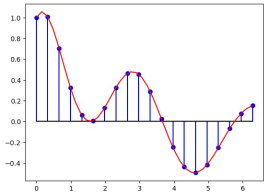
Pyramids and Wavelets

- Image Interpolation (upsampling)
- Image Decimation (downsampling)
- Multi-resolution representations (image pyramids)
- Wavelets

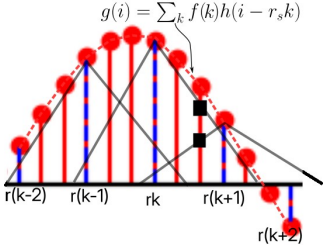
Pyramids and Wavelets (cont.)



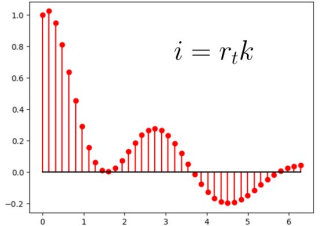
(a)



(b)



(c)



(d)

Figure 31: *Signal interpolation $g(i) = \sum_k f(k)h(i - rk)$: (a) Original signal; (b) Approximate continuous representation; (c) Signal interpolation using a tent kernel $h(\cdot)$; (d) Discrete resampled signal.*
r_s: original sampling rate;
r_t: target sampling rate.

Pyramids and Wavelets (cont.)

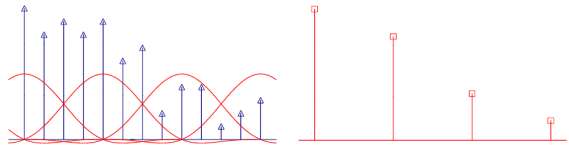


Figure 32: *Signal decimation: (a) the original samples are (b) convolved with a low-pass filter before being downsampled.*

Image pyramid

- Image pyramids are extremely useful for performing **multi-scale editing operations** such as blending images while maintaining details.
- For example, the task of finding a face in an image. Since we do not know the scale at which the face will appear, we need to **generate a whole pyramid of differently sized images** and scan each one for possible faces.
- Changing image resolution to construct pyramids requires: **upsampling** (interpolation) and **downsampling** (decimation).

Pyramids and Wavelets (cont.)

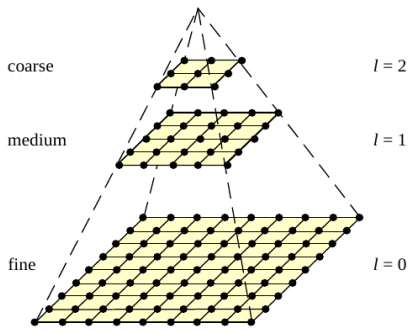


Figure 33: *A traditional image pyramid: each level has half the resolution (width and height), and hence a quarter of the pixels, of its parent level*

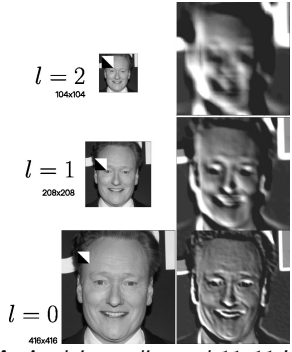


Figure 34: *Applying a diagonal 11x11 Haar filter to images at multiple scales.* *We want to find left side of the subject's jaw.*

Pyramids and Wavelets (cont.)

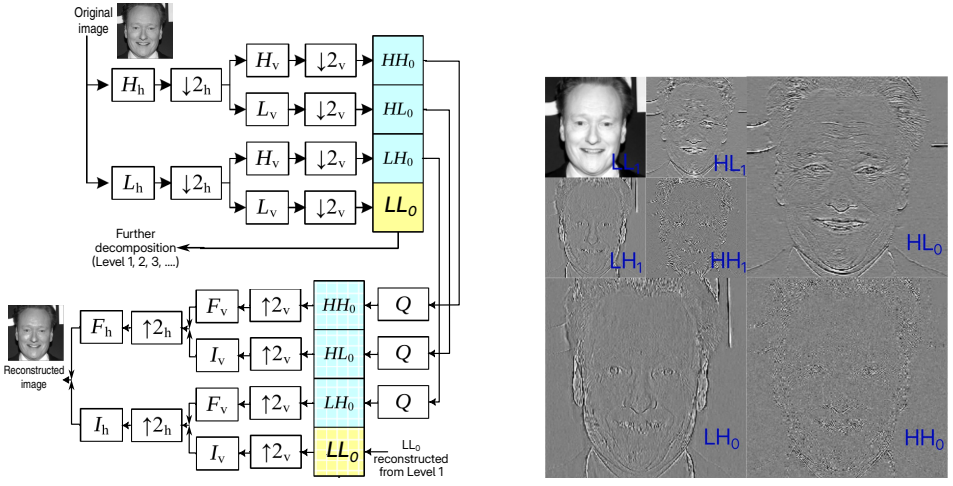


Figure 35: *Example of a wavelet decomposition: (a) Original image, (b) Wavelet decomposition.*

Pyramids and Wavelets (cont.)

Wavelets vs. Pyramids

- Wavelets are filters that localize a signal in both space and frequency (like the Gabor filter) and are defined over a hierarchy of scales.
- Wavelets provide a smooth way to decompose a signal into frequency components without blocking and are closely related to pyramids.
- Traditional pyramids are overcomplete (i.e. they use more pixels than the original image to represent the decomposition), whereas wavelets provide a tight frame (i.e., they keep the size of the decomposition the same as the image)

Geometric Transformations

"Aims to modify a pixel's position (image's domain) rather than its intensity (image's range)"

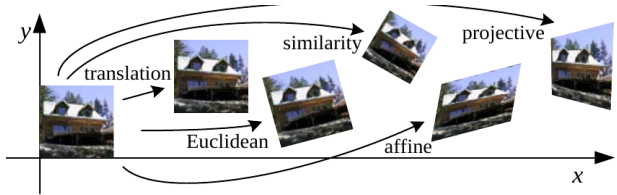
Geometric Transformations

- Parametric transformations
- Warping algorithms
- Image warping as a Resampling problem
- Mesh-based warping

Geometric Transformations

Parametric transformations (affines and projection)

Parametric transformations apply a global deformation to an image. Deformation is controlled by a small number of parameters.



Geometric operator

The operator h aims to modify the pixels' locations within the image.

$$g(\vec{\mathbf{x}}) = f(h(\vec{\mathbf{x}}))$$

Geometric Transformations (cont.)

Transformation	Matrix	# DoF	Preserves	Icon
translation	$\begin{bmatrix} \mathbf{I} & \mathbf{t} \end{bmatrix}_{2\times 3}$	2	orientation	
rigid (Euclidean)	$\begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix}_{2\times 3}$	3	lengths	
similarity	$\begin{bmatrix} s\mathbf{R} & \mathbf{t} \end{bmatrix}_{2\times 3}$	4	angles	
affine	$\begin{bmatrix} \mathbf{A} \end{bmatrix}_{2\times 3}$	6	parallelism	
projective	$\begin{bmatrix} \tilde{\mathbf{H}} \end{bmatrix}_{3\times 3}$	8	straight lines	

Figure 36: *Hierarchy of 2D coordinate transformations*

Geometric Transformations (cont.)

Image warping algorithms

procedure *forwardWarp*$(f, \mathbf{h}, \textbf{out}\ g)$:

For every pixel $\mathbf{x}$ in $f(\mathbf{x})$

1. Compute the destination location $\mathbf{x}' = \mathbf{h}(\mathbf{x})$.
2. Copy the pixel $f(\mathbf{x})$ to $g(\mathbf{x}')$.

Figure 37: *Forward warping*

procedure *inverseWarp*$(f, \mathbf{h}, \textbf{out}\ g)$:

For every pixel $\mathbf{x}'$ in $g(\mathbf{x}')$

1. Compute the source location $\mathbf{x} = \hat{\mathbf{h}}(\mathbf{x}')$
2. Resample $f(\mathbf{x})$ at location $\mathbf{x}$ and copy to $g(\mathbf{x}')$

Figure 38: *Inverse warping*

Geometric Transformations (cont.)



Figure 39: *Forward warping*

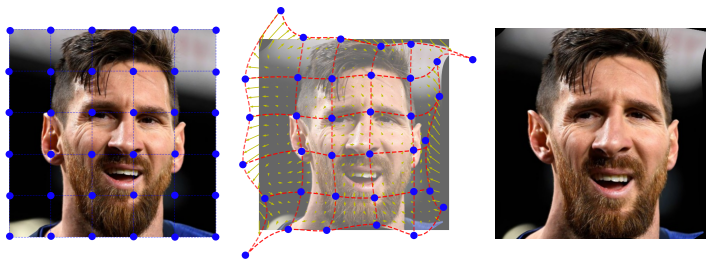


Figure 40: *Inverse warping*

Geometric Transformations (cont.)

Mesh based warping

- While parametric transforms specified by a small number of global parameters have many uses, local deformations with more degrees of freedom are often required.
- Different amounts of transformations are required in different parts of the image.



(a) (b) (c)

Figure 41: *Mesh based warping: (a) Original image with deformation grid; (b) Displacement of the grid's points; (c) Warped image.* *Free-form deformation (FFD) was employed for warping. Arrows in yellow depict deformation fields.*