

Module 2

1. More neighborhood operators :

Linear filters can perform a wide variety of image transformations. However non-linear filters, such as edge-preserving median or bilateral filters, can sometimes perform even better. Other examples of neighborhood operators include morphological operators that operate on binary images, as well as semi-global operators that compute distance transforms and find connected components in binary images.

1.1.Non-linear filtering

The filters we have looked at so far have all been linear, i.e., their response to a sum of two signals is the same as the sum of the individual responses. This is equivalent to saying that each output pixel is a weighted summation of some number of input pixels. Linear filters are easier to compose and are amenable to frequency response analysis. In many cases, however, better performance can be obtained by using a non-linear combination of neighboring pixels.

1.1.1. Median filtering

A better filter to use in this case is the median filter, which selects the median value from each pixel's neighborhood. Median values can be computed in expected linear time using a randomized select algorithm (Cormen 2001) and incremental variants have also been developed as well as a constant time algorithm that is independent of window size. Since the shot noise value usually lies well outside the true values in the neighborhood, the median filter is able to filter away such bad pixels. One downside of the median filter, in addition to its moderate computational cost, is that because it selects only one input pixel value to replace each output pixel, it is not as efficient at averaging away regular Gaussian noise. A better choice may be the alpha-trimmed mean which averages together all of the pixels except for the α fraction that are the smallest and the largest.

Another possibility is to compute a weighted median, in which each pixel is used a number of times depending on its distance from the center. This turns out to be equivalent to minimizing the weighted objective function

$$\sum_{k,l} w(k,l) |f(i+k, j+l) - g(i,j)|^p,$$

where $g(i,j)$ is the desired output value and $p = 1$ for the weighted median. The value $p = 2$ is the usual weighted mean, which is equivalent to correlation (3.12) after normalizing by the sum of the weights. The weighted mean also has deep connections to other methods in robust statistics such as influence functions. Non-linear smoothing has another, perhaps even more important property, especially as shot noise is rare in today's cameras. Such filtering is more edge preserving, i.e., it has less tendency to soften edges while filtering away high-frequency noise.

Consider the noisy image. In order to remove most of the noise, the Gaussian filter is forced to smooth away high-frequency detail, which is most noticeable near strong edges. Median

filtering does better but, as mentioned before, does not do as well at smoothing away from discontinuities. While we could try to use the α -trimmed mean or weighted median, these techniques still have a tendency to round sharp corners, since the majority of pixels in the smoothing area come from the background distribution.

1.2. Bilateral filtering :

In the bilateral filter, the output pixel value depends on a weighted combination of neighboring pixel values

$$g(i, j) = \frac{\sum_{k,l} f(k, l) w(i, j, k, l)}{\sum_{k,l} w(i, j, k, l)}. \quad (3.34)$$

The weighting coefficient $w(i, j, k, l)$ depends on the product of a *domain kernel*, (Figure 3.19c),

$$d(i, j, k, l) = \exp \left(-\frac{(i-k)^2 + (j-l)^2}{2\sigma_d^2} \right), \quad (3.35)$$

and a data-dependent *range kernel* (Figure 3.19d),

$$r(i, j, k, l) = \exp \left(-\frac{\|f(i, j) - f(k, l)\|^2}{2\sigma_r^2} \right). \quad (3.36)$$

When multiplied together, these yield the data-dependent *bilateral weight function*

$$w(i, j, k, l) = \exp \left(-\frac{(i-k)^2 + (j-l)^2}{2\sigma_d^2} - \frac{\|f(i, j) - f(k, l)\|^2}{2\sigma_r^2} \right). \quad (3.37)$$

Since bilateral filtering is quite slow compared to regular separable filtering, a number of acceleration techniques have been developed. In particular, the bilateral which subsamples the higher dimensional color/position space on a uniform grid, continues to be widely used, including the application of the bilateral solver. An even faster implementation of bilateral filtering can be obtained using the permutohedral lattice approach.

1.2.1. Iterated adaptive smoothing and anisotropic diffusion

Bilateral (and other) filters can also be applied in an iterative fashion, especially if an appearance more like a “cartoon” is desired. When iterated filtering is applied, a much smaller neighborhood can often be used.

Consider, for example, using only the four nearest neighbors, i.e., restricting $|k - i| + |l - j| \leq 1$ in (3.34). Observe that

$$d(i, j, k, l) = \exp \left(-\frac{(i - k)^2 + (j - l)^2}{2\sigma_d^2} \right) \quad (3.38)$$

$$= \begin{cases} 1, & |k - i| + |l - j| = 0, \\ e^{-1/2\sigma_d^2}, & |k - i| + |l - j| = 1. \end{cases} \quad (3.39)$$

We can thus re-write (3.34) as

$$\begin{aligned} f^{(t+1)}(i, j) &= \frac{f^{(t)}(i, j) + \eta \sum_{k,l} f^{(t)}(k, l) r(i, j, k, l)}{1 + \eta \sum_{k,l} r(i, j, k, l)} \\ &= f^{(t)}(i, j) + \frac{\eta}{1 + \eta R} \sum_{k,l} r(i, j, k, l) [f^{(t)}(k, l) - f^{(t)}(i, j)], \end{aligned} \quad (3.40)$$

where $R = \sum_{(k,l)} r(i, j, k, l)$, (k, l) are the $\mathcal{N}_4$ (nearest four) neighbors of (i, j) , and we have made the iterative nature of the filtering explicit.

Since its original introduction, anisotropic diffusion has been extended and applied to a wide range of problems. It has also been shown to be closely related to other adaptive smoothing techniques as well as Bayesian regularization with a non-linear smoothness term that can be derived from image statistics.

In its general form, the range kernel $r(i, j; k, l) = r(\|f(i, j) - f(k, l)\|)$, which is usually called the gain or edge-stopping function, or diffusion coefficient, can be any monotonically increasing function with $r(x) \rightarrow 0$ as $x \rightarrow \infty$. Black, Sapiro et al. (1998) show how anisotropic diffusion is equivalent to minimizing a robust penalty function on the image gradients.

They also extend the diffusion neighborhood from $\mathcal{N}_4$ to $\mathcal{N}_8$, which allows them to create a diffusion operator that is both rotationally invariant and incorporates information about the eigenvalues of the local structure tensor.

Note that, without a bias term towards the original image, anisotropic diffusion and iterative adaptive smoothing converge to a constant image. Unless a small number of iterations is used (e.g., for speed), it is usually preferable to formulate the smoothing problem as a joint minimization of a smoothness term and a data fidelity term, which introduce such a bias in a principled manner.

1.2.2. Guided image filtering

It is also possible to use a different guide image to adaptively filter a noisy input. An example of this is using a flash image, which has strong edges but poor color, to adaptively filter a low-light non-flash color image, which has large amounts of noise. Guided Image Filtering is a technique used to **enhance an image adaptively** based on a different image.

This is useful when the input image is noisy, blurry, or lacks details, and we want to improve it by referencing a clearer image.

The filtering process follows this equation:

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

Where:

- $g(i, j)$ is the output pixel
- $f(k, l)$ is the noisy input image
- $h(i, j)$ is the guide image
- $w(h; i, j, k, l)$ is a weight computed from the guide image

The relationship between the input and guide image follows an **affine transformation**:

$$g(i, j) = A_{k,l}h(i, j) + B_{k,l}$$

where:

- $g(i, j)$ is the filtered output.
- $h(i, j)$ is the guide image.
- $A_{k,l}$ and $B_{k,l}$ are local parameters computed from the guide image.

Using **ridge regression**, solve for $A_{k,l}$ and $B_{k,l}$:

$$A_{k,l} = \frac{\sum_{(i,j) \in N_{k,l}} (h(i, j) - \mu_h)(f(i, j) - \mu_f)}{\sigma_h^2 + \epsilon}$$

$$B_{k,l} = \mu_f - A_{k,l}\mu_h$$

Instead of just taking the predicted value of the filtered pixel $g(i, j)$ from the window centered on that pixel, an average across all windows that cover the pixel is used. The resulting algorithm consists of a series of local mean image and image moment filters, a per-pixel linear system solve (which reduces to a division if the guide image is scalar), and another set of filtering steps.

1.3.Binary image processing

While non-linear filters are often used to enhance grayscale and color images, they are also used extensively to process binary images. Such images often occur after a thresholding operation,

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

e.g., converting a scanned grayscale document into a binary image for further processing, such as optical character recognition.

1.3.1. Morphology

The most common binary image operations are called morphological operations, because they change the shape of the underlying binary objects.

To perform such an operation, we first convolve the binary image with a binary structuring element and then select a binary output value depending on the thresholded result of the convolution.

The structuring element can be any shape, from a simple 3×3 box filter, to more complicated disc structures. It can even correspond to a particular shape that is being sought for in the image.

The standard operations used in binary morphology include:

- **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:** $\text{close}(f, s) = \text{erode}(\text{dilate}(f, s), s)$.

dilation grows (thickens) objects consisting of 1s, while erosion shrinks (thins) them. The opening and closing operations tend to leave large regions and smooth boundaries unaffected, while removing small objects or holes and smoothing boundaries.

1.3.2. Distance transforms

The distance transform is useful in quickly precomputing the distance to a curve or set of points using a two-pass raster algorithm. It has many applications, including level sets, fast chamfer matching (binary image alignment), feathering in image stitching and blending, and nearest point alignment.

The distance transform $D(i, j)$ of a binary image $b(i, j)$ is defined as follows. Let $d(k, l)$ be some *distance metric* between pixel offsets. Two commonly used metrics include the *city block* or *Manhattan* distance

$$d_1(k, l) = |k| + |l| \quad (3.46)$$

and the *Euclidean* distance

$$d_2(k, l) = \sqrt{k^2 + l^2}. \quad (3.47)$$

The distance transform is then defined as

$$D(i, j) = \min_{k, l: b(k, l)=0} d(i - k, j - l), \quad (3.48)$$

A useful extension of the basic distance transform is the signed distance transform, which computes distances to boundary pixels for all the pixels. The simplest way to

create this is to compute the distance transforms for both the original binary image and its complement and to negate one of them before combining. Because such distance fields tend to be smooth, it is possible to store them more compactly using a spline defined over a quadtree or octree data structure. Such precomputed signed distance transforms can be extremely useful in efficiently aligning and merging 2D curves and 3D surfaces, especially if the vectorial version of the distance transform, i.e., a pointer from each pixel or voxel to the nearest boundary or surface element, is stored and interpolated. Signed distance fields are also an essential component of level set evolution, they are called characteristic functions.

1.3.3. Connected components

Another useful semi-global image operation is finding connected components, which are defined as regions of adjacent pixels that have the same input value or label. Pixels are said to be N4 adjacent if they are immediately horizontally or vertically adjacent, and N8 if they can also be diagonally adjacent. Both variants of connected components are widely used in a variety of applications, such as finding individual letters in a scanned document or finding objects (say, cells) in a thresholded image and computing their area statistics.

Once a binary or multi-valued image has been segmented into its connected components, it is often useful to compute the area statistics for each individual region R . Such statistics include:

- the area (number of pixels);
- the perimeter (number of boundary pixels);
- the centroid (average x and y values);
- the second moments,

$$\mathbf{M} = \sum_{(x,y) \in \mathcal{R}} \begin{bmatrix} x - \bar{x} \\ y - \bar{y} \end{bmatrix} \begin{bmatrix} x - \bar{x} & y - \bar{y} \end{bmatrix}, \quad (3.49)$$

from which the major and minor axis orientation and lengths can be computed using eigenvalue analysis.

These statistics can then be used for further processing, e.g., for sorting the regions by the area size (to consider the largest regions first) or for preliminary matching of regions in different images.

2. Fourier transforms

Fourier analysis is a mathematical tool that helps us study the frequency characteristics of signals and images. In image processing, it allows us to analyze how different filters affect various frequency components, such as high, medium, and low frequencies. When we apply a filter to an image or signal, it modifies the frequency content. To analyze what a filter does to different frequency components, we pass a sinusoidal wave of a known frequency through the filter and observe its effect.

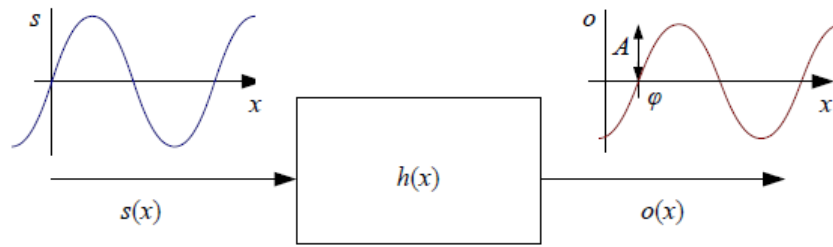


Figure 3.24 The Fourier Transform as the response of a filter $h(x)$ to an input sinusoid $s(x) = e^{j\omega x}$ yielding an output sinusoid $o(x) = h(x) * s(x) = Ae^{j(\omega x + \phi)}$.

A sinusoidal wave can be represented as:

A sinusoidal wave can be represented as:

$$s(x) = \sin(2\pi f x + \phi_i) = \sin(\omega x + \phi_i)$$

where:

- f is the frequency,
- $\omega = 2\pi f$ is the angular frequency,
- ϕ_i is the phase shift.

If we convolve this sinusoidal signal $s(x)$ with a filter $h(x)$, the output is another sinusoidal signal with the same frequency but a different magnitude and phase:

$$o(x) = h(x) * s(x) = A \sin(\omega x + \phi_o)$$

where:

- A is the new magnitude (gain),
- ϕ_o is the new phase shift.

The difference in phase, $\Delta\phi = \phi_o - \phi_i$, represents the phase shift introduced by the filter.

A sinusoidal function can also be expressed using Euler's formula:

$$s(x) = e^{j\omega x} = \cos(\omega x) + j \sin(\omega x)$$

Using this representation, the filtered output is:

$$o(x) = h(x) * s(x) = Ae^{j(\omega x + \phi)}$$

where $Ae^{j\phi}$ represents both the magnitude and phase change introduced by the filter.

The **Fourier transform** expresses how a filter or signal behaves across different frequencies. The Fourier transform of a filter $h(x)$ is given by:

$$H(\omega) = \mathcal{F}\{h(x)\} = Ae^{j\phi}$$

This means that the filter's effect on a frequency component is described by its magnitude A and phase shift ϕ .

The Fourier transform is often represented using a notation like:

$$h(x) \xleftrightarrow{\mathcal{F}} H(\omega)$$

where the arrow signifies that $H(\omega)$ is the Fourier transform of $h(x)$.

In the **continuous domain**, the Fourier transform is defined as:

$$H(\omega) = \int_{-\infty}^{\infty} h(x)e^{-j\omega x} dx$$

In the **discrete domain**, the Discrete Fourier Transform (DFT) is given by:

$$H(k) = \frac{1}{N} \sum_{x=0}^{N-1} h(x)e^{-j2\pi kx/N}$$

where N is the number of samples.

The DFT has a computational complexity of $O(N^2)$, which is inefficient for large images or signals. The **Fast Fourier Transform (FFT)** is an optimized algorithm that reduces the complexity to $O(N \log_2 N)$, making it much faster.

The FFT works by:

1. Breaking the computation into smaller parts (divide and conquer).
2. Using a series of small 2×2 transforms, known as "butterfly operations."
3. Rearranging data in a way that minimizes redundant calculations.

Most modern programming libraries, such as NumPy in Python, provide built-in FFT functions for efficient computation.

The Fourier transform has several useful properties that help in signal and image processing:

1. **Linearity:** $\mathcal{F}\{af(x) + bg(x)\} = aF(\omega) + bG(\omega)$.
2. **Shifting:** $\mathcal{F}\{f(x - x_0)\} = e^{-j\omega x_0} F(\omega)$.
3. **Convolution Theorem:** Convolution in the spatial domain corresponds to multiplication in the frequency domain:

$$\mathcal{F}\{f(x) * g(x)\} = F(\omega)G(\omega)$$

4. **Correlation Theorem:** The Fourier transform of the correlation function is the conjugate product of the transforms:

$$\mathcal{F}\{f(x) \star g(x)\} = F(\omega)G^*(\omega)$$

5. **Parseval's Theorem:** The total energy of a signal remains the same in both time and frequency domains:

$$\sum |f(x)|^2 = \sum |F(\omega)|^2$$

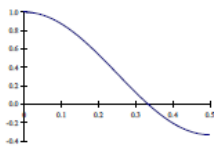
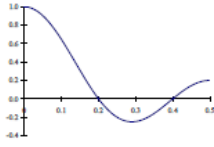
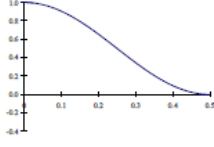
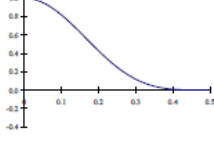
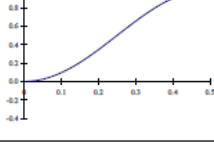
Name	Kernel	Transform	Plot
box-3	$\frac{1}{3} \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}$	$\frac{1}{3}(1 + 2 \cos \omega)$	
box-5	$\frac{1}{5} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \end{bmatrix}$	$\frac{1}{5}(1 + 2 \cos \omega + 2 \cos 2\omega)$	
linear	$\frac{1}{4} \begin{bmatrix} 1 & 2 & 1 \end{bmatrix}$	$\frac{1}{2}(1 + \cos \omega)$	
binomial	$\frac{1}{16} \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \end{bmatrix}$	$\frac{1}{4}(1 + \cos \omega)^2$	
Sobel	$\frac{1}{2} \begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$	$\sin \omega$	
corner	$\frac{1}{2} \begin{bmatrix} -1 & 2 & -1 \end{bmatrix}$	$\frac{1}{2}(1 - \cos \omega)$	

Table 3.1 Fourier transforms of the separable kernels shown in Figure 3.14, obtained by evaluating $\sum_k h(k)e^{-jk\omega}$.

Fourier Transform of Common Filters

The Fourier transform allows us to analyze how different filters affect an image's frequency components. Here are some examples:

1. Moving Average Filter

A moving average filter smooths an image but does not uniformly suppress high frequencies, causing **ringing artifacts**.

2. Gaussian Filter

A Gaussian filter is often used for blurring and does a better job at preserving frequency components compared to the moving average filter.

3. Sobel Edge Detector

The Sobel filter enhances edges but weakens very high frequencies, which means it struggles to detect fine details.

Applications of Fourier Transform in Image Processing

1. **Image Filtering:** Applying filters in the frequency domain is often more efficient than in the spatial domain.
2. **Compression:** The Fourier transform helps in JPEG compression by removing less important frequency components.
3. **Feature Extraction:** Many computer vision algorithms analyze frequency characteristics of images.
4. **Image Enhancement:** Frequency-based techniques help in noise removal and sharpening.
5. **Pattern Recognition:** Fourier descriptors are used to recognize shapes in images.

Fourier analysis provides a powerful way to study the frequency characteristics of images and filters. The **Fourier transform** breaks down an image into its frequency components, while the **Fast Fourier Transform (FFT)** provides an efficient way to compute it. By analyzing the Fourier transform of filters, we can understand their effects on different frequency ranges, helping us design better image processing techniques.

2.1. Two-Dimensional Fourier Transforms

The formulas and insights developed for one-dimensional signals and their transforms translate directly to two-dimensional images. Instead of specifying a horizontal or vertical frequency ω_x or ω_y , we can create an oriented sinusoid of frequency (ω_x, ω_y) :

$$s(x, y) = \sin(\omega_x x + \omega_y y)$$

The corresponding two-dimensional Fourier transforms are defined as follows:

$$H(\omega_x, \omega_y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} h(x, y) e^{-j(\omega_x x + \omega_y y)} dx dy$$

In the discrete domain, the Discrete Fourier Transform (DFT) is given by:

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

where M and N are the width and height of the image. All Fourier transform properties from the one-dimensional case extend to two dimensions by replacing scalar variables x, ω, x_0, a with their two-dimensional counterparts $(x, y), (\omega_x, \omega_y), (x_0, y_0), (a_x, a_y)$, and using vector inner products instead of multiplications.

2.1.1. Wiener Filtering

The Fourier transform is not only useful for analyzing the frequency characteristics of a filter kernel or image, but also for studying the frequency spectrum of a whole class of images.

A simple model assumes that images are random noise fields with an expected magnitude at each frequency given by the power spectrum $P_s(\omega_x, \omega_y)$:

$$\mathbb{E}[|S(\omega_x, \omega_y)|^2] = P_s(\omega_x, \omega_y)$$

where $\mathbb{E}[\cdot]$ denotes the expected (mean) value. To generate such an image, we create a random Gaussian noise image $S(\omega_x, \omega_y)$ where each pixel follows a zero-mean Gaussian distribution with variance $P_s(\omega_x, \omega_y)$, and then take its inverse FFT.

This observation—that signal spectra provide a first-order description of spatial statistics—is widely used in signal and image processing. Assuming that an image is a sample from a correlated Gaussian random noise field, combined with a statistical model of the measurement process, yields an optimal restoration filter known as the **Wiener filter**.

2.1.2. Discrete Cosine Transform (DCT)

The Discrete Cosine Transform (DCT) is a variant of the Fourier transform particularly well-suited for compressing images in a block-wise manner. The one-dimensional DCT is computed by taking the dot product of each N -wide block of pixels with a set of cosines of different frequencies:

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

where k is the frequency coefficient index. The $\frac{1}{2}$ -pixel offset ensures the basis coefficients remain symmetric (Wallace, 1991).

Some discrete cosine basis functions illustrate this concept. The first basis function (a constant function) encodes the average DC value in the pixel block, while the second captures a low-frequency variation (a slightly curvy slope).

The DCT closely approximates the optimal **Karhunen–Loève transform (KLT)** for natural images over small patches. The KLT, obtained through Principal Component Analysis (PCA), optimally decorrelates signals assuming the signal is well-described by its spectrum, making it theoretically ideal for compression.

The two-dimensional version of the DCT follows a similar formulation:

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

Like the 2D Fast Fourier Transform (FFT), the 2D DCT can be computed separably: first by performing a DCT on each row and then applying the transform to each column. The computational complexity is $O(N \log N)$, similar to the FFT.

Applications of DCT in Image and Video Compression

The DCT is widely used in image and video compression standards such as JPEG. However, newer methods like **wavelet transforms** and **overlapped DCT variants** have been integrated into **JPEG2000** and **JPEG XR** standards.

These newer techniques suffer less from **blocking artifacts**, which occur due to independent transformation and quantization of pixel blocks (typically 8×8).

Two-dimensional Fourier transforms extend the principles of one-dimensional analysis to images, enabling frequency-based filtering and analysis. The Wiener filter, derived from statistical models, provides optimal restoration under Gaussian noise assumptions but has largely been replaced by modern non-linear techniques. The Discrete Cosine Transform (DCT) remains a cornerstone of image compression, effectively decorrelating image data and enabling efficient encoding in standards like JPEG. Despite advancements in wavelet-based compression methods, the DCT remains widely used due to its balance between computational efficiency and compression performance.

2.1.3. Application: Sharpening, Blur, and Noise Removal

A common application of image processing is the enhancement of images through sharpening and noise removal operations, which require neighborhood processing. Traditionally, these operations were performed using **linear filtering**. However, modern techniques now favor **non-linear filters**, such as the **weighted median** or **bilateral filter**, **anisotropic diffusion**, or **non-local means**. Additionally, **variational methods**, especially those using **non-quadratic (robust) norms**, such as the **L1 norm** (known as **total variation**), are widely used. Recently, **deep neural networks** have taken over the field of **image denoising**.

Linear vs. Non-Linear Filtering

Linear filters, such as Gaussian smoothing or Laplacian sharpening, work by convolving the image with a predefined kernel. Non-linear methods, however, adapt based on local pixel statistics, preserving important structures while reducing noise.

- **Weighted Median Filter:** Uses local pixel values with adaptive weights to preserve edges while reducing noise.
- **Bilateral Filter:** Smooths while maintaining edges by weighting pixels based on spatial and intensity differences.
- **Anisotropic Diffusion:** Reduces noise while enhancing edges by allowing controlled blurring along certain directions.
- **Non-Local Means:** Estimates the true intensity of a pixel by averaging over similar patches from the entire image.
- **Variational Methods:** Solve an optimization problem to enhance image quality while minimizing distortion.

Measuring the Effectiveness of Image Denoising

To evaluate the effectiveness of denoising algorithms, researchers typically use the **Peak Signal-to-Noise Ratio (PSNR)** (Equation 2.120). Given a clean image $I(x)$ and a denoised image $\hat{I}(x)$, PSNR is computed as:

$$PSNR = 10 \log_{10} \left(\frac{MAX^2}{MSE} \right)$$

where MAX is the maximum possible pixel value and MSE (Mean Squared Error) is given by:

$$MSE = \frac{1}{N} \sum_x \left(I(x) - \hat{I}(x) \right)^2$$

However, PSNR does not always correlate well with human perception. Thus, **perceptual similarity metrics** such as the **Structural Similarity Index (SSIM)** (Wang, Bovik et al. 2004; Wang, Bovik, and Simoncelli 2005) and the **FLIP image difference evaluator** (Andersson, Nilsson et al. 2020) are preferred.

Advanced Similarity Metrics

More recent approaches use **neural perceptual similarity metrics** to assess image quality:

- **Feature-based Perceptual Metrics:** Evaluate image similarity using deep feature representations (Johnson, Alahi, and Fei-Fei 2016; Dosovitskiy and Brox 2016; Zhang, Isola et al. 2018; Tariq, Tursun et al. 2020; Czolbe, Krause et al. 2020).
- **Texture Preservation Metrics:** Unlike PSNR or L1 metrics that encourage smooth results, these metrics retain texture similarity (Cho, Joshi et al. 2012).

No-Reference Image Quality Assessment

When a clean reference image is not available, **no-reference image quality assessment** methods are used:

- **NIQE (Mittal, Moorthy, and Bovik 2012)**: Uses natural image statistics to estimate quality.
- **Deep Learning-Based Assessments (Talebi and Milanfar 2018)**: Train neural networks to predict perceived image quality without a reference.

3. Pyramids and Wavelets

In this section, we explore techniques for modifying image resolution to match specific needs. While previous transformations have maintained the same image size, real-world applications often require resizing images. For instance, when displaying an image on a high-resolution screen or printing it, we may need to interpolate a smaller image to a higher resolution. Conversely, downsampling an image can reduce storage requirements, transmission time, or computation costs.

In some cases, the appropriate resolution may be unknown. For example, in face detection, we do not initially know the scale at which a face appears. A common approach is to construct a multi-resolution pyramid, where we generate images at various scales and scan each for faces. This strategy is inspired by biological vision systems, which process images at multiple scales (Marr 1982). Similarly, pyramids assist object detection by first identifying a coarse version of the object and then refining the search in higher-resolution images. They also enable multi-scale image editing, such as seamless blending of images while preserving fine details.

3.1. Interpolation

In order to interpolate (or upsample) an image to a higher resolution, we need to select some interpolation kernel with which to convolve the image,

$$g(i, j) = \sum_{k, l} f(k, l) h(i - rk, j - rl).$$

This formula extends the concept of **discrete convolution** by modifying the input sample positions using the **upsampling rate rrr**. Instead of directly using k and l in the filter function $h()$, we replace them with rk and rl . This effectively spaces out the original input samples by a factor of r , creating an upsampled image.

The effectiveness of an interpolation kernel depends on both the application and computational complexity. Several smoothing kernels can be adapted for interpolation by appropriately rescaling them.

- **Linear Interpolation (Tent Kernel)**: This produces **piecewise linear curves**, which are continuous but introduce **visible creases** in images due to sharp transitions.
- **Cubic B-Spline Interpolation**: This is a **smoother, approximating kernel**, meaning the interpolated image does **not** pass through the exact input points. Instead, it generates **soft, visually pleasing results** but reduces high-frequency details (sharp edges). The

cubic B-spline function can be derived by convolving the **tent function (linear B-spline) with itself**, effectively smoothing out transitions further.

Each kernel has trade-offs: linear interpolation is computationally cheaper but introduces artifacts, while cubic B-splines provide smoother results but may blur fine details.

While most graphics cards use the bilinear kernel, most photo editing packages use bicubic interpolation. The cubic interpolant is a C1 (derivative-continuous) piecewise-cubic spline whose equation is

$$h(x) = \begin{cases} 1 - (a + 3)x^2 + (a + 2)|x|^3 & \text{if } |x| < 1 \\ a(|x| - 1)(|x| - 2)^2 & \text{if } 1 \leq |x| < 2 \\ 0 & \text{otherwise,} \end{cases}$$

where a specifies the derivative at $x = 1$. The value of a is often set to -1 , since this best matches the frequency characteristics of a sinc function. It also introduces a small amount of sharpening, which can be visually appealing. Unfortunately, this choice does not linearly interpolate straight lines (intensity ramps), so some visible ringing may occur. A better choice for large amounts of interpolation is probably $a = -0.5$, which produces a quadratic reproducing spline; it interpolates linear and quadratic functions exactly.

In **computer vision**, splines play a crucial role in various tasks, including:

- **Elastic image deformations**, where images are smoothly warped using spline-based transformations.
 - **Scattered data interpolation**, which reconstructs missing data points.
 - **Motion estimation**, where splines help model smooth object trajectories.
 - **Surface interpolation**, useful in reconstructing 3D shapes from sparse data.
- More broadly, many **image processing operations** can be performed by representing images as splines and **manipulating them in a multi-resolution framework**.

Among interpolation methods, the **windowed sinc function** is considered the **highest quality interpolator** because it effectively:

1. **Preserves details** in lower-resolution images.
2. **Minimizes aliasing**, preventing high-frequency artifacts.

It is even possible to construct a **C¹ piecewise-cubic approximation** to the windowed sinc by matching its **derivatives at zero crossings**.

However, despite its advantages, some practitioners find the **windowed sinc undesirable** due to:

- **Excessive ringing artifacts** (visible oscillations around edges).
- **Repetitive ringing frequencies**, which can create artificial patterns in images

To mitigate these issues, some photographers and image processors prefer **alternative interpolation techniques**, such as:

- **Repeated small fractional interpolation**, which **decorrelates the original pixel grid from the final image**, reducing structured artifacts.
- **Using a bilateral filter as an interpolator**, which adapts to image content and avoids over-smoothing.

- **Global optimization techniques**, which minimize artifacts through an energy-based approach.
 - **Hallucinating details**, where **high-frequency information is synthesized** to improve perceived sharpness in upsampled images.
- Each technique offers trade-offs between **quality, computational efficiency, and artifact suppression**, making the choice of interpolator highly application-dependent.

3.2. Decimation

While interpolation can be used to increase the resolution of an image, decimation (downsampling) is required to reduce the resolution.¹³ To perform decimation, we first (conceptually) convolve the image with a low-pass filter (to avoid aliasing) and then keep every r th sample. In practice, we usually only evaluate the convolution at every r th sample,

$$g(i, j) = \sum_{k, l} f(k, l) h(ri - k, rj - l),$$

Note that the smoothing kernel $h(k, l)$, in this case, is often a stretched and re-scaled version of an interpolation kernel. Alternatively, we can write

$$g(i, j) = \frac{1}{r} \sum_{k, l} f(k, l) h(i - k/r, j - l/r)$$

and keep the same kernel $h(k, l)$ for both interpolation and decimation.

A widely used **$r=2$ decimation filter** is the **binomial filter**. This filter is effective at **separating high and low frequencies** but still allows some **high-frequency detail** to remain. This residual detail can lead to **aliasing** when the image is **downsampled**. However, in applications like **image blending**, this aliasing is generally **not a major concern**.

For cases where the **downsampled images** will be **displayed directly to users** or blended with multiple resolutions, a **higher-quality filter** is preferred. When dealing with **high downsampling rates**, the **windowed sinc prefilter** is a good option. However, for **lower downsampling rates**, such as **$r=2$** , a more **carefully designed filter** is required.

Several **commonly used $r=2$ downsampling filters**, These include:

- **The linear filter [1; 2; 1]**: Offers a **relatively poor response** with weak frequency separation.
- **The binomial filter [1; 4; 6; 4; 1]**: Effectively removes high frequencies and is **widely used in computer vision analysis pyramids**.
- **The cubic filters from (3.65)**: The **$a=-1$ filter** has a **sharper fall-off** than the **$a=-0.5$ filter**, making it more effective at reducing aliasing.
- **A cosine-windowed sinc function**: Provides a **smoother frequency response** for better image quality.
- **The QMF-9 filter**: Commonly used for **wavelet denoising**. It allows a **fair amount of aliasing**, but its **original coefficients are normalized to a $\sqrt{2}$ gain**, making it **self-inverting**.
- **The 9/7 analysis filter from JPEG 2000** : A high-quality filter designed for **wavelet-based compression**, reducing artifacts while maintaining detail.

For **full-precision** values of some filter coefficients, refer to the **original papers**. Each of these filters has **trade-offs** in terms of **computational complexity**, **aliasing suppression**, and **image sharpness**, making the choice dependent on the specific **downsampling application**.

3.3 Multi-resolution representations

The **Burt and Adelson Laplacian Pyramid** is one of the most widely used **pyramid representations** in **computer vision**. It is constructed by repeatedly **blurring** and **subsampling** the original image by a factor of **two**, storing the result in successive levels of the pyramid.

Since each level of the pyramid is related by a **sampling rate $r=2$** , this structure is known as an **octave pyramid**. The key to this method is the choice of a **five-tap kernel** for downsampling.

Binomial Kernel and Gaussian Pyramid Formation

The binomial kernel, derived from:

$$[c \quad b \quad a \quad b \quad c]$$

where:

- $b = \frac{1}{4}$
- $c = \frac{1}{4} - \frac{a}{2}$
- The commonly used value for a is $3/8$

This results in the familiar **binomial filter**:

$$\frac{1}{16} [1 \quad 4 \quad 6 \quad 4 \quad 1]$$

This filter is computationally efficient, as it can be implemented using simple **shifts and adds**, which was crucial in earlier computing when **multipliers were expensive**.

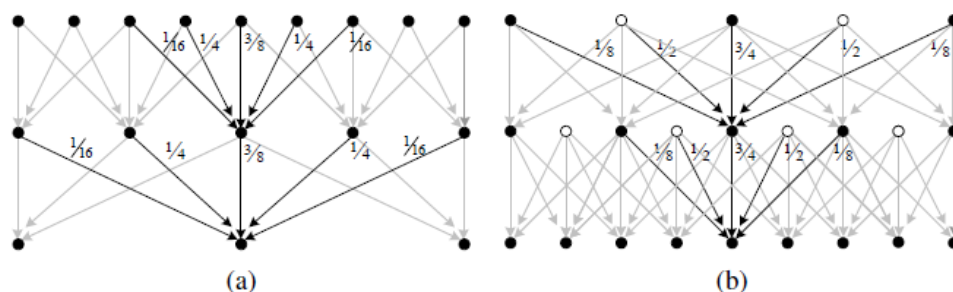


Figure 3.32 The Gaussian pyramid shown as a signal processing diagram: The (a) analysis and (b) re-synthesis stages are shown as using similar computations. The white circles indicate zero values inserted by the $\uparrow 2$ upsampling operation. Notice how the reconstruction filter coefficients are twice the analysis coefficients. The computation is shown as flowing down the page, regardless of whether we are going from coarse to fine or vice versa.

Repeated **convolutions** with this binomial kernel cause the pyramid to **converge to a Gaussian**, which is why it is called a **Gaussian Pyramid**. This multi-resolution structure allows for efficient:

- **Coarse-to-fine search algorithms**, enabling detection of objects or patterns at different scales.
- **Multi-resolution blending**, where images are merged seamlessly without visible seams.
- **Fractional-level decimation** using **MIP-mapping**, commonly applied in **computer graphics hardware and software**.

The **Laplacian Pyramid**, introduced by **Burt and Adelson**, is derived from the **Gaussian Pyramid** by computing **differences between adjacent levels**, effectively capturing high-frequency details. This makes it particularly useful for **image compression, feature detection, and seamless blending**.

To compute the **Laplacian Pyramid**, **Burt and Adelson** first perform **interpolation** on a **lower-resolution image** to obtain a **reconstructed low-pass version** of the original image. This reconstructed image is then **subtracted** from the original image to generate the **band-pass “Laplacian” image**, which can be stored for further processing.

A key property of the **Laplacian Pyramid** is **perfect reconstruction**—the **Laplacian images** combined with the **base-level Gaussian** are sufficient to **exactly reconstruct** the original image. **Figure 3.32** illustrates this process using a **signal processing diagram**, which effectively captures the **analysis and re-synthesis stages**.

Variants of the Laplacian Pyramid

Burt and Adelson describe an alternative approach where the **low-pass image** is taken **directly from the original blurred image** rather than the **reconstructed pyramid** (by piping the output of the **L box** directly into the **subtraction step** in **Figure 3.33**). This variant introduces **less aliasing**, since it avoids an additional **downsampling and upsampling** round-trip. However, it is **not self-inverting**, as the **Laplacian images** alone are **not sufficient to reconstruct** the original image.

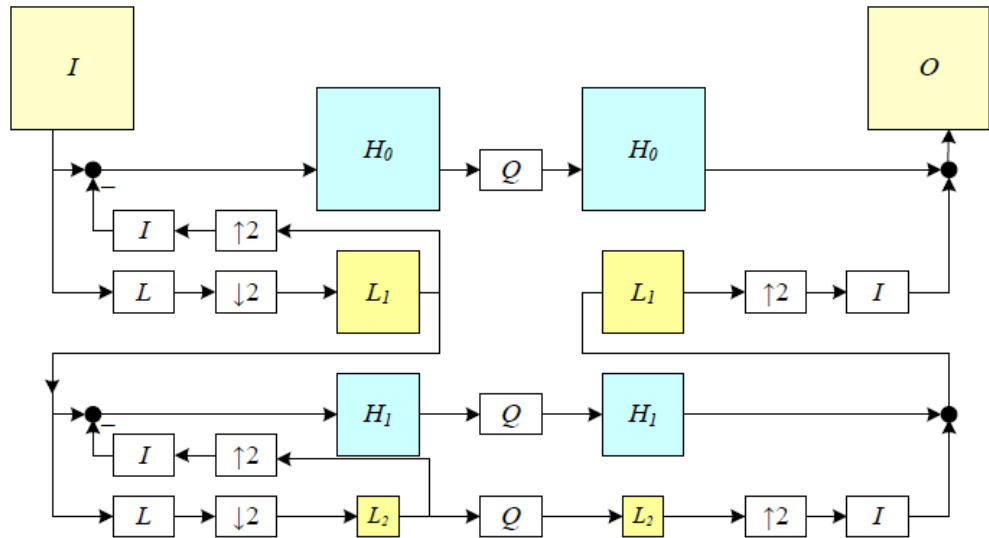


Figure 3.33 The Laplacian pyramid. The yellow images form the Gaussian pyramid, which is obtained by successively low-pass filtering and downsampling the input image. The blue images, together with the smallest low-pass image, which is needed for reconstruction, form the Laplacian pyramid. Each band-pass (blue) image is computed by upsampling and interpolating the lower-resolution Gaussian pyramid image, resulting in a blurred version of that level's low-pass image, which is subtracted from the low-pass to yield the blue band-pass image. During reconstruction, the interpolated images and the (optionally filtered) high-pass images are added back together starting with the coarsest level. The Q box indicates quantization or some other pyramid processing, e.g., noise removal by coring (setting small wavelet values to 0).

Relation to Differences of Gaussians (DoG) and Laplacians of Gaussians (LoG)

Similar to the **Gaussian Pyramid**, the **Laplacian Pyramid** is named somewhat **imprecisely**, since the computed **band-pass images** are actually **Differences of Gaussians (DoG)**:

$$DoG\{I, \sigma_1, \sigma_2\} = G_{\sigma_1} * I - G_{\sigma_2} * I = (G_{\sigma_1} - G_{\sigma_2}) * I$$

where G_{σ} is a Gaussian filter with scale σ , and $*$ represents convolution.

In contrast, the **Laplacian of Gaussian (LoG)** is computed as the **second derivative** of a Gaussian-smoothed image:

$$LoG\{I, \sigma\} = \nabla^2(G_{\sigma} * I) = (\nabla^2 G_{\sigma}) * I$$

where the **Laplacian operator** is given by:

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

Applications of the Laplacian Pyramid

The **Laplacian Pyramid** has many useful applications in **computer vision** and **image processing**:

- **Edge Detection**
- **Stereo Matching**
- **Image Enhancement**
- **Local Contrast Manipulation and Tone Mapping for High Dynamic Range (HDR) Images**
- **Local Laplacian Filters**
 - These use **local range clipping** and **modified Laplacian pyramids** for **edge-preserving filtering** and **tone mapping**.
- **Style Transfer**

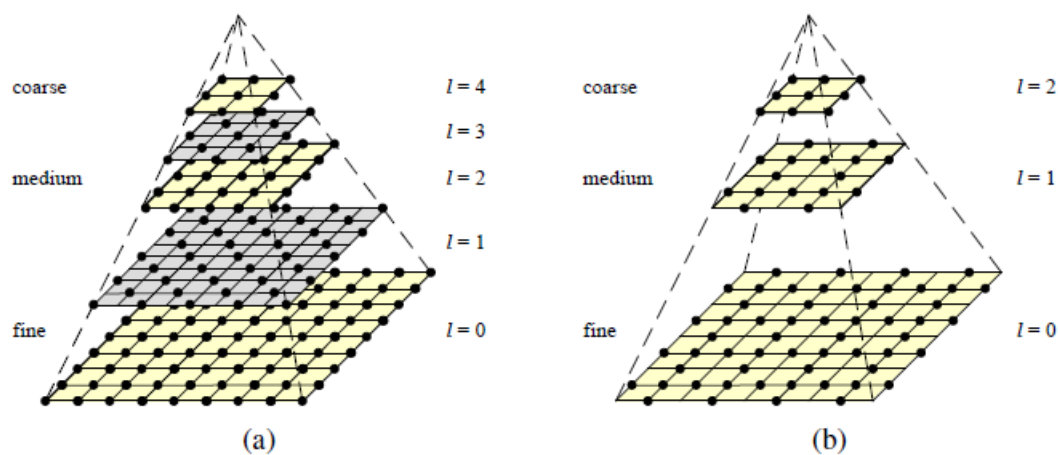


Figure 3.35 Multiresolution pyramids: (a) pyramid with half-octave (quincunx) sampling (odd levels are colored gray for clarity). (b) wavelet pyramid—each wavelet level stores $3/4$ of the original pixels (usually the horizontal, vertical, and mixed gradients), so that the total number of wavelet coefficients and original pixels is the same.

Half-Octave Pyramids and Quincunx Sampling

A less common variant of the Laplacian Pyramid is the **Half-Octave Pyramid** (Figure 3.35a), introduced by **Crowley and Stern (1984)**. They refer to it as the **Difference of Low-Pass (DOLP) transform**. Since this method involves **smaller scale changes** between levels, it is claimed to improve **coarse-to-fine algorithms**.

In **image processing**, a related concept is **Quincunx Sampling**, which combines **half-octave pyramids** with a **checkerboard sampling grid**. This method is useful for **multi-scale feature detection**, where it is common to use:

- **Half-octave pyramids**
- **Quarter-octave pyramids**

In this case, **subsampling** only occurs at **octave levels**, and the image is repeatedly **blurred** with wider **Gaussian kernels** until a **full octave of resolution change** is achieved.

3.4.Wavelets

While **pyramids** are widely used in **computer vision**, some researchers use **wavelet decompositions** as an alternative. **Wavelets** are **filters** that **localize** a signal in both **space and frequency** (similar to the **Gabor filter**) and are defined across a **hierarchy of scales**. Unlike pyramids, wavelets **decompose a signal into frequency components** without introducing **blocking artifacts**.

Wavelets originated in **applied mathematics and signal processing** and were introduced to **computer vision**.

Applications of Wavelets in Computer Graphics and Vision

Wavelets are extensively used in the **computer graphics community** for **multi-resolution geometric processing** and have been applied in **computer vision** for various tasks, including:

- **Multi-resolution image analysis**
- **Feature detection**
- **Denoising**
- **Multi-scale oriented filtering**

Wavelets vs. Image Pyramids

Both **wavelets** and **image pyramids** decompose an image into **multi-resolution representations**, capturing details localized in both **space and frequency**. However, the key difference is:

- **Traditional pyramids are overcomplete**, meaning they use **more pixels** than the original image to represent the decomposition.
- **Wavelets provide a tight frame**, maintaining the same number of coefficients as the original image.

Some **wavelet families** are **overcomplete**, offering better **shiftability** and **orientation selectivity**, which is an advantage over standard **band-pass pyramids**.

Wavelet Construction in Two Dimensions

Two-dimensional **wavelets** are constructed using a **recursive coarse-to-fine approach**, as illustrated in **Figure 3.37a**. This process consists of:

1. **Analysis (decomposition):**
 - A **high-pass filter** is applied, followed by **decimation**, preserving $\frac{3}{4}$ of the original pixels.
 - $\frac{1}{4}$ of the **low-frequency coefficients** are passed to the next stage for further processing.
2. **Synthesis (reconstruction):**
 - The original signal is **reconstructed** by reversing the decomposition process.

In practice, filtering is usually performed in **two separable sub-stages**, as shown in **Figure 3.37b**. This results in three **wavelet subbands**, often referred to as:

- **High–High (HH)**: Captures mixed derivative information.
- **High–Low (HL)**: Emphasizes **horizontal edges** and gradients.
- **Low–High (LH)**: Highlights **vertical edges** and gradients.

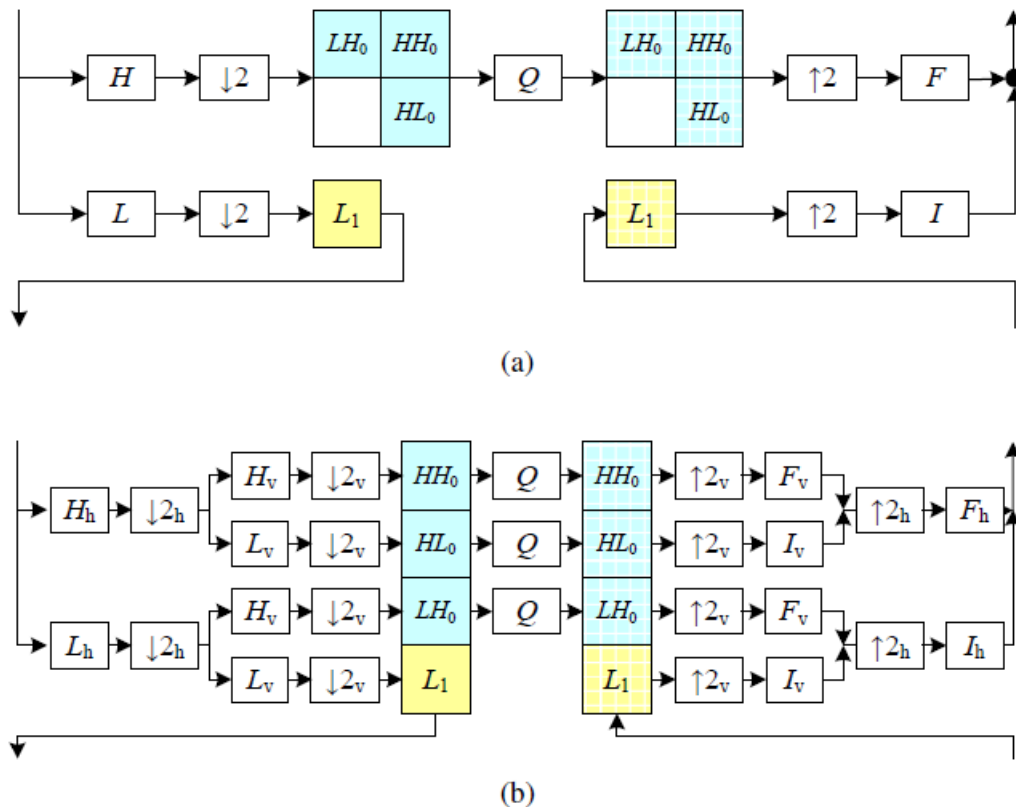


Figure 3.37 Two-dimensional wavelet decomposition: (a) high-level diagram showing the low-pass and high-pass transforms as single boxes; (b) separable implementation, which involves first performing the wavelet transform horizontally and then vertically. The I and F boxes are the interpolation and filtering boxes required to re-synthesize the image from its wavelet components.

Choice of Wavelet Filters

The **high-pass (H)** and **low-pass (L)** filters used in **Figure 3.37b** must be carefully chosen to ensure optimal performance in applications such as **compression, feature extraction, and denoising**. The corresponding **reconstruction filters (I and F)** must also be designed for **perfect reconstruction**.

Simoncelli and Adelson provide **odd-length quadrature mirror filter (QMF) coefficients** that work well in practice.

The Lifting Scheme for Wavelets

Since **wavelet filter design** is **complex**, a more straightforward approach called the **lifting scheme** has been introduced. Instead of designing complex wavelet bases, this method involves:

1. **Splitting the signal into even and odd components**
2. **Applying simple, reversible filtering operations** on each sequence
3. **Reconstructing the transformed signal**

This approach produces what are called **lifted wavelets**, which are easier to compute while maintaining **perfect reconstruction**. Sweldens (1996, 1997) provides a clear introduction to this method, which has led to the development of **second-generation wavelets**. (Figures 3.38 and 3.39).

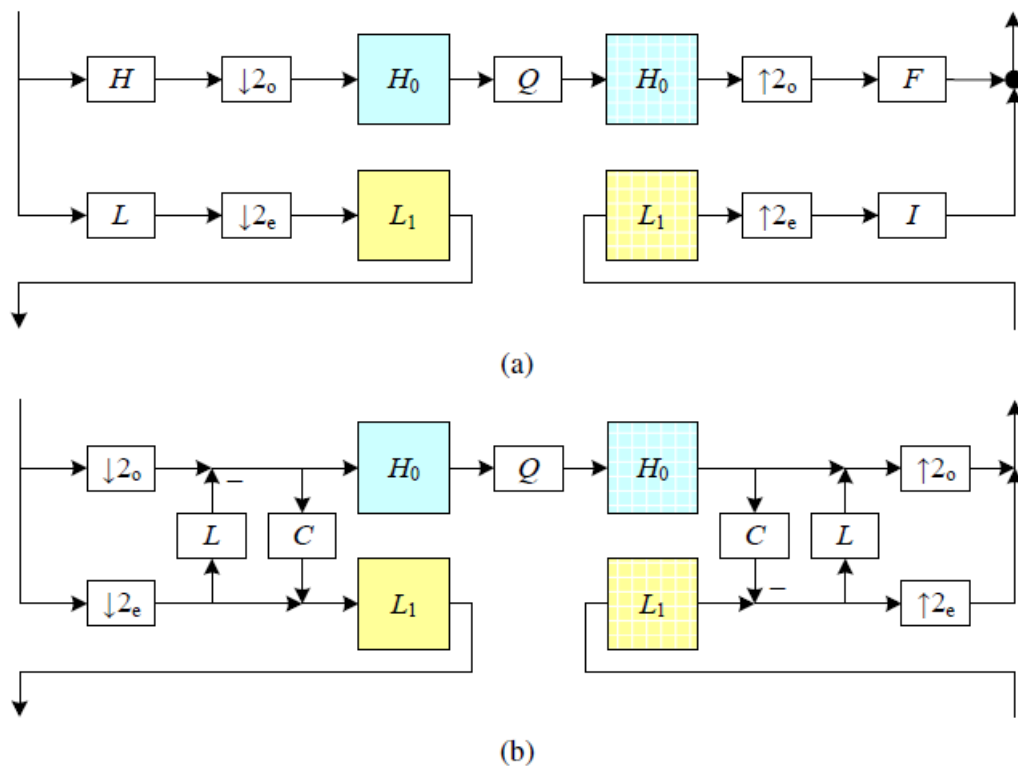


Figure 3.38 One-dimensional wavelet transform: (a) usual high-pass + low-pass filters followed by odd ($\downarrow 2_o$) and even ($\downarrow 2_e$) downsampling; (b) lifted version, which first selects the odd and even subsequences and then applies a low-pass prediction stage L and a high-pass correction stage C in an easily reversible manner.

3.5.Application: Image Blending

One of the most **engaging** and widely used applications of the **Laplacian pyramid** is the creation of **blended composite images**. A simple **splicing** of two images, such as an **apple** and an **orange**, along a **hard boundary** results in a **noticeable cut**. However, by **splining** them together, the result is a **seamless hybrid fruit**, where the transition between the **red apple** and **orange** appears natural.

Key Idea: Smooth Frequency Blending

The essential concept behind this **pyramid-based blending** is that:

- **Low-frequency color variations** (such as the transition between red and orange) are blended **smoothly**.
- **High-frequency textures** (such as the fine details on the apple's skin and the roughness of the orange peel) are blended **quickly** to avoid **ghosting artifacts** when overlaid.

Steps for Laplacian Pyramid Blending

To create a **blended image**, the process involves the following steps:

1. **Decomposing each input image** into its **own Laplacian pyramid**
2. **Generating a smooth weighting function**, which defines the contribution of each image at different scales.
 - The **simplest approach** is to use a **binary mask**, which marks the regions where each image should contribute.
 - A **Gaussian pyramid** is then computed from this mask to create **smooth transitions**.
3. **Multiplying each Laplacian level** of the source images by the corresponding level of the **Gaussian mask pyramid**.
4. **Summing the weighted Laplacian pyramids** to obtain the **final blended image**.

Flexible Masking and Arbitrary Image Blending

This technique is highly **flexible**, allowing for:

- **Arbitrary mask shapes**, leading to **interesting and artistic blending results**.
- **Multi-image blending**, where an **integer-valued label image** specifies the contribution of multiple source images at different locations.
- **Application to image stitching and compositing**, where exposures vary across images.

More **recent blending techniques**, such as **Poisson blending** and **gradient-domain blending**, provide alternative ways to achieve **seamless image composites**, but the **Laplacian pyramid method** remains a **widely used and effective approach** in **computer vision and image processing**.

4. Geometric Transformations

Unlike brightness or contrast changes (which affect pixel colors), geometric transformations change where pixels are placed in the image.

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

here we look at functions that transform the *domain*,

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

4.1. Parametric Transformations

Parametric transformations apply a global deformation to an image, where the behavior of the transformation is controlled by a small number of parameters.

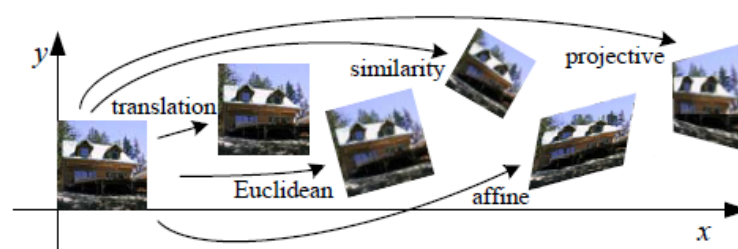


Figure 3.44 Basic set of 2D geometric image transformations.

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	

Table 3.3 Hierarchy of 2D coordinate transformations. Each transformation also preserves the properties listed in the rows below it, i.e., similarity preserves not only angles but also parallelism and straight lines. The 2×3 matrices are extended with a third $[0^T \ 1]$ row to form a full 3×3 matrix for homogeneous coordinate transformations.

To get the new image $g(x')$, we need to **map pixels from the original image $f(x)$ to their new locations**.

- Approach 1: Forward Warping (Forward Mapping)
- Approach 2: Inverse Warping

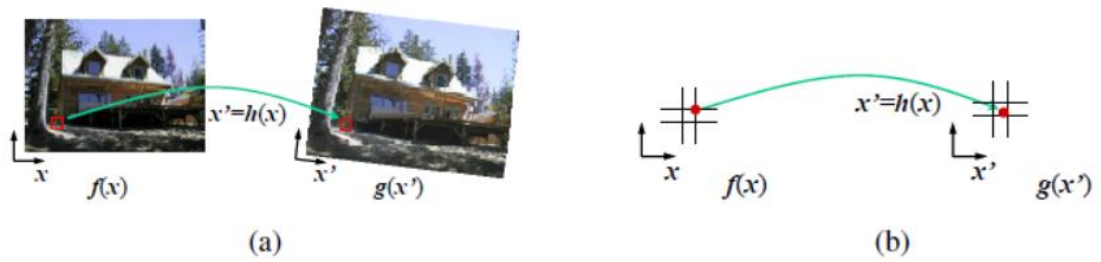


Figure 3.45 Forward warping algorithm: (a) a pixel $f(x)$ is copied to its corresponding location $x' = h(x)$ in image $g(x')$; (b) detail of the source and destination pixel locations.

```
procedure forwardWarp( $f, h, \text{out } g$ ):
```

```
  For every pixel  $x$  in  $f(x)$ 
```

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

In fact, this approach suffers from several limitations. The process of copying a pixel $f(x)$ to a location x' in g is not well defined when x' has a non-integer value. You can round the value of x' to the nearest integer coordinate and copy the pixel there, but the resulting image has severe aliasing and pixels that jump around a lot when animating the transformation. You can also “distribute” the value among its four nearest neighbors in a weighted (bilinear) fashion, keeping track of the per-pixel weights and normalizing at the end. This technique is called splatting and is sometimes used for volume rendering in the graphics community. Unfortunately, it suffers from both moderate amounts of aliasing and a fair amount of blur (loss of high-resolution detail). The second major problem with forward warping is the appearance of cracks and holes, especially when magnifying an image. Filling such holes with their nearby neighbors can lead to further aliasing and blurring.

A preferable solution is to use inverse warping.



Figure 3.46 Inverse warping algorithm: (a) a pixel $g(x')$ is sampled from its corresponding location $x = \hat{h}(x')$ in image $f(x)$; (b) detail of the source and destination pixel locations.

```
procedure inverseWarp( $f, h, out\ g$ ):
```

```
  For every pixel  $x'$  in  $g(x')$ 
```

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

Algorithm 3.2 Inverse warping algorithm for creating an image $g(x')$ from an image $f(x)$ using the parametric transform $x' = h(x)$.

To prevent aliasing, we need to prefilter the image $f(x)$ with a filter whose frequency response is the projection of the final desired spectrum through the A^{-T} transform. In general (for non-zoom transforms), this filter is nonseparable and hence is very slow to compute. Therefore, a number of approximations to this filter are used in practice, include MIP-mapping, elliptically weighted Gaussian averaging, and anisotropic filtering.

MIP-Mapping

MIP-mapping, first introduced by **Williams (1983)**, is a **pyramid-based** technique designed for **efficient texture filtering** in **computer graphics**. It is particularly useful in **texture mapping**, where it helps reduce **aliasing** and improve **rendering speed** by precomputing multiple downsampled versions of an image.

Structure of a MIP-Map

A **MIP-map** is essentially a **standard image pyramid**, similar to the **Gaussian pyramid**, but with a key difference:

- Each level is **prefiltered** using a **high-quality filter**, unlike **Burt and Adelson's five-tap binomial filter**, which provides a less accurate approximation.
- This results in **better texture quality** and **reduced aliasing artifacts** when textures are displayed at different resolutions.

Resampling an Image from a MIP-Map

When rendering a **textured surface**, the system first determines the **resampling rate rrr**, which represents how much the texture needs to be scaled:

- **Maximum resampling rate** → **Reduces aliasing** (suppresses high-frequency details).
- **Minimum resampling rate** → **Reduces blurring** (preserves more texture detail).

Using this resampling rate, the appropriate **pyramid level** l is calculated using the **base-2 logarithm**:

$$l = \log_2 r$$

This determines which **MIP-map level** to sample from.

Sampling Strategies in MIP-Mapping

There are different approaches to selecting the appropriate **MIP-map level** for texture resampling:

1. Nearest Level Sampling

- Selects the **closest** MIP-map level (higher or lower).
- **Fast**, but may cause visible transitions between levels.

2. Trilinear MIP-Mapping

- Resamples from **two adjacent levels** and **blends them linearly** based on the fractional part of l .
- This provides **smoother transitions** between texture levels and **reduces artifacts**.
- Most **graphics APIs** (e.g., **OpenGL**, **Direct3D**) support this method by default.

Trade-offs in MIP-Mapping

- **Speed vs. Quality:**
 - **Lower-quality filtering** (e.g., nearest level) is **faster** but may introduce **visible artifacts**.
 - **Higher-quality filtering** (e.g., trilinear MIP-mapping) produces **smoother textures** but requires **more computation**.
- **Aliasing vs. Blurring:**
 - Choosing the **wrong resampling rate** can either cause **aliasing** (sharp but **jagged textures**) or **excessive blurring** (**smooth but overly soft textures**).
- **Hardware Acceleration:**
 - Modern GPUs provide **hardware support** for MIP-mapping, allowing real-time implementation with **minimal performance cost**.

Elliptical Weighted Average (EWA) filter :

The **Elliptical Weighted Average (EWA) filter**, introduced by **Greene and Heckbert (1986)**, is a **high-quality resampling filter** that accounts for **anisotropic distortion** when mapping textures in **computer graphics**. It improves **texture filtering** by adapting to **skewed**

transformations in an image, making it more accurate than standard filtering techniques like **bilinear or trilinear filtering**.

Despite its reputation as a high-quality filter, that because a Gaussian kernel is used, the technique suffers simultaneously from both blurring and aliasing, compared to higher-quality filters. The EWA is also quite slow, although faster variants based on MIP-mapping have been proposed.

Anisotropic filtering

An alternative approach to filtering oriented textures, which is sometimes implemented in graphics hardware (GPUs), is to use anisotropic. In this approach, several samples at different resolutions (fractional levels in the MIP-map) are combined along the major axis of the EWA Gaussian.

4.2. Mesh-based Warping

Local deformations with more degrees of freedom are often required when different parts of an image need varying amounts of motion. For example, transforming a frown into a smile requires curving the corners of the mouth upwards while keeping the rest of the face intact. To achieve this, a dense displacement field must be interpolated from a sparse set of corresponding points.

One method is to triangulate the points in the source image and use an affine motion model within each triangle. After adjusting the vertex locations in the destination image, inverse warping can be applied. If the source image is used as a texture map, computer graphics rendering algorithms can reconstruct the transformed image, but careful handling of triangle edges is necessary to prevent aliasing.

Alternative interpolation methods involve moving quadrilateral mesh vertices or using variational interpolants like regularization. Another approach is employing locally weighted combinations of displacements, such as radial basis functions. When quadrilateral meshes are used, displacement interpolation to pixel values can be achieved through smooth functions like quadratic B-splines.

If a dense depth map is available, forward warping can be applied, though it may introduce cracks and aliasing. A more effective approach is to forward warp the displacement field, fill any small holes, and then use inverse warping for resampling. This technique works well because displacement fields are generally smoother than images, reducing aliasing artifacts.

A second method for defining displacements is using corresponding oriented line segments. Pixels along these segments are mapped directly from source to destination, while other pixels undergo smooth interpolation. Each segment induces a similarity transform—comprising translation, rotation, and scaling—that influences neighboring pixels based on their minimum distance to the segment.

Another approach is to use a mesh tailored to the image content. Although manually defining such meshes can be labor-intensive, interactive systems facilitate this process. Once

corresponding meshes are specified, intermediate warps can be produced through linear interpolation, with spline-based interpolation used to determine displacements at mesh nodes.

4.3. Application: Feature-based morphing

Warping can be used to modify the appearance of a single image or animate it, but even more powerful effects are achieved by warping and blending multiple images using morphing. Instead of simple cross-dissolving, which results in ghosting, each image is first warped toward the other before blending. When correspondences are correctly set up using techniques such as those in Figure 3.49, features align properly, preventing ghosting.

During morphing, this process is applied for each intermediate frame, where different blends and amounts of deformation are applied at each time step. The interpolation follows a time parameter $t \in [0,1]$, which controls the transformation sequence. The weighting functions for the two warped images are $(1 - t)$ and t , ensuring a smooth transition. The pixel movements determined by the correspondences are also linearly interpolated.

Special attention is needed when partially warping an image towards another, especially for complex nonlinear motion. In such cases, defining how deformation progresses is crucial to achieving natural results.

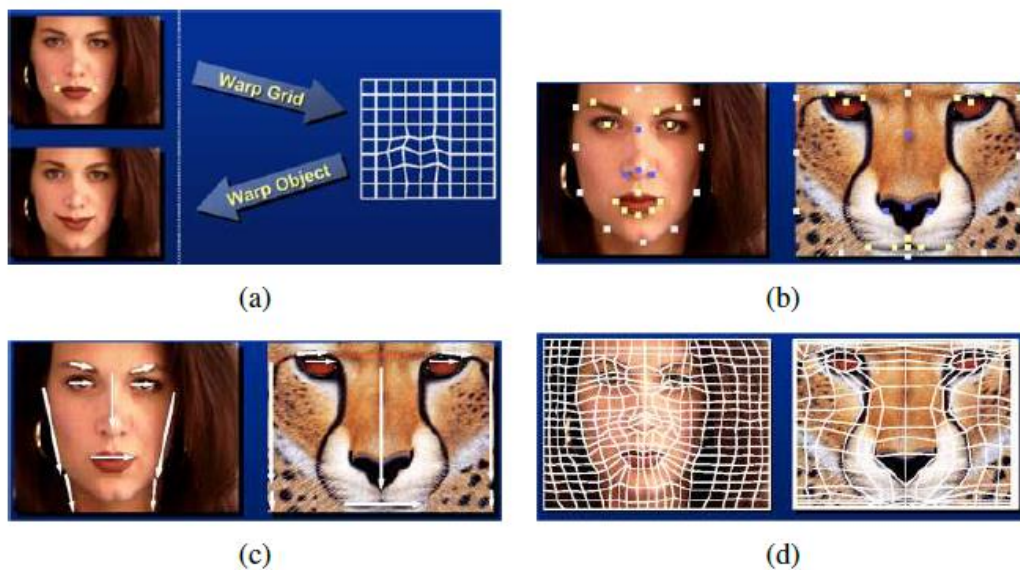


Figure 3.49 Image warping alternatives (Gomes, Darsa et al. 1999) © 1999 Morgan Kaufmann: (a) sparse control points $\rightarrow$ deformation grid; (b) denser set of control point correspondences; (c) oriented line correspondences; (d) uniform quadrilateral grid.