SREENIVASA INSTITUTE OF TECHNOLOGY AND MANAGEMENT STUDIES

(AUTONOMOUS)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

COMPUTER VISION

Course Code: 23CSE364A

LECTURE NOTES

Units I – V

Prepared by

Karunia Krishna Priya & A. Ajitha

Table of Contents

LECTURE NOTES

UNIT – I

LINEAR FILTERS

Topics Covered:

1.1 Introduction to Computer Vision

1.2 Linear Filters and Convolution

1.3 Shift-Invariant Linear Systems

1.4 Spatial Frequency and Fourier Transforms

1.5 Sampling and Aliasing

1.6 Filters as Templates

1.7 Technique: Normalized Correlation and Finding Patterns

1.8 Technique: Scale and Image Pyramids

1.9 Unit Summary and Review Questions

Textbook: David A. Forsyth, Jean Ponce, "Computer Vision – A Modern Approach", PHI, 2003

1.1 Introduction to Computer Vision

Definition. Computer Vision is the scientific field concerned with enabling machines to extract meaningful, high-level information from digital images or video, in a manner that mimics — and in some tasks exceeds — human visual perception. It sits at the intersection of image processing, pattern recognition, machine learning, and geometry.

Why is vision hard? An image is a two-dimensional array of numbers (pixel intensities). The "meaning" of the scene — objects, depth, motion, identity — must be *inferred* from this array. This inference problem is ill-posed: the same 3-D scene can produce infinitely many 2-D images (depending on lighting, viewpoint, and camera parameters), and, conversely, a single 2-D image is consistent with infinitely many 3-D scenes. Vision algorithms must therefore use priors, models, and statistical regularities of the visual world to resolve this ambiguity.

Goals of Computer Vision

- **Image formation** — understanding how a scene becomes an image (radiometry, optics, sampling).
- **Low-level vision** — filtering, edge detection, feature extraction.
- **Mid-level vision** — segmentation, grouping, texture analysis, stereo, motion.
- **High-level vision** — object recognition, scene understanding, activity recognition.

Applications

Domain	Example Tasks

Domain	Example Tasks
Industrial	Defect inspection, robotic bin-picking
Medical	Tumor detection, organ segmentation, image-guided surgery
Security	Face recognition, surveillance, biometrics
Automotive	Lane detection, pedestrian detection, autonomous driving
Entertainment	Motion capture, augmented/virtual reality
Document analysis	OCR, handwriting recognition

Image as a function. A grayscale digital image is modeled as a function

$$I: \Omega \subset \mathbb{Z}^2 \to \mathbb{R}, \qquad I(x, y) = \text{intensity at pixel } (x, y)$$

For color images, $I(x, y)$ is a vector (R, G, B). Video adds a time dimension: $I(x, y, t)$.

The human visual system (brief preview). Human vision performs remarkable feats of grouping, constancy, and recognition using a retina (photoreceptors: rods for low-light scotopic vision, cones for color/photopic vision), the optic nerve, and hierarchical cortical processing (V1 → V2 → V4 → IT). Computer vision borrows inspiration from these principles — e.g., receptive fields in V1 resemble oriented Gaussian-derivative filters, motivating the linear filters studied in this unit; Gestalt grouping principles motivate segmentation (Unit IV).

1.1.1 Image Formation: A Brief Primer

Before any filtering or analysis can occur, a physical scene must be converted into a digital image. Two simplified models are useful background for everything that follows in this unit.

The pinhole camera model. Light from a 3-D scene point (X, Y, Z) (in camera-centered coordinates, with Z the depth along the optical axis) passes through an idealized infinitesimal aperture and projects onto an image plane at distance f (the focal length) behind the pinhole, landing at image coordinates:

$$x = f\,\frac{X}{Z}, \qquad y = f\,\frac{Y}{Z}$$

This is **perspective projection**: note that image position depends on the *ratio* X/Z, so scene points along the same ray from the camera center project to the same image point — the fundamental reason a single 2-D image cannot, by itself, determine 3-D depth (the ill-posedness noted above). It also explains why objects farther away (Z large) appear smaller — a fact used directly in Unit III's shape-from-texture analysis.

Sampling and quantization. The continuous irradiance pattern falling on the image plane is measured by a discrete grid of sensor elements (pixels), each integrating light over a small area and a short exposure time, and the resulting continuous voltage is **quantized** to a finite set of digital levels (typically 8 bits, i.e., 256 gray levels, per channel for standard images). This double discretization — spatial sampling (governed by the sampling theorem, §1.5) and intensity quantization — is what converts a continuous physical signal into the discrete array $I(x, y)$ that all subsequent processing operates on.

Color sensing. Most digital cameras use a single sensor array overlaid with a **color filter mosaic** (commonly the Bayer pattern: alternating rows of red-green and green-blue filtered pixels), and interpolate ("demosaic") the missing color channels at each pixel location — itself a filtering operation closely related to the interpolation/upsampling step used when expanding pyramid levels (§1.8.3).

Why this matters for linear filtering. Every linear filter studied in this unit is applied to the *already-sampled, already-quantized* discrete image $I(x, y)$ — but the sampling process itself (§1.5) is governed by exactly the same convolution and frequency-domain theory used to design filters, which is why sampling/aliasing is treated as a core topic of this unit rather than a separate concern.

Roadmap of this unit. We begin with the foundational operation of this entire unit — convolution — and its role in defining linear, shift-invariant systems (§1.2–1.3). We then develop the frequency-domain (Fourier) view of these same systems (§1.4), which is indispensable both for understanding *why* filters behave as they do and for efficient computation. This frequency-domain machinery immediately explains the sampling theorem and the phenomenon of aliasing (§1.5), a direct practical consequence of viewing an image as a signal with finite bandwidth. We then reinterpret filters as pattern-matching templates (§1.6), leading to normalized cross-correlation as a principled, brightness-invariant matching technique (§1.7). Finally, we introduce image pyramids (§1.8) as the standard tool for handling the unavoidable problem of unknown scale in real images — a representation that recurs throughout the remainder of the course, from oriented pyramids for texture (Unit III) to coarse-to-fine search strategies used broadly across computer vision.

1.2 Linear Filters and Convolution

1.2.1 Motivation

Most image-processing operations, at their core, replace each pixel with some combination of its neighbors. This local, neighborhood-based combination is the essence of **filtering**. When the combination rule is a fixed *weighted sum* — independent of position and image content — the operation is a **linear filter**, and it is mathematically expressed as **convolution**.

1.2.2 Linear Operators

An operator $\mathcal{H}$ acting on images is **linear** if, for any two images f_1, f_2 and scalars a, b:

$$\mathcal{H}(af_1 + bf_2) = a\,\mathcal{H}(f_1) + b\,\mathcal{H}(f_2)$$

This is the **superposition principle**. Linearity allows complex operations to be decomposed into simpler, additive pieces — a property exploited throughout signal and image processing.

1.2.3 Discrete Convolution

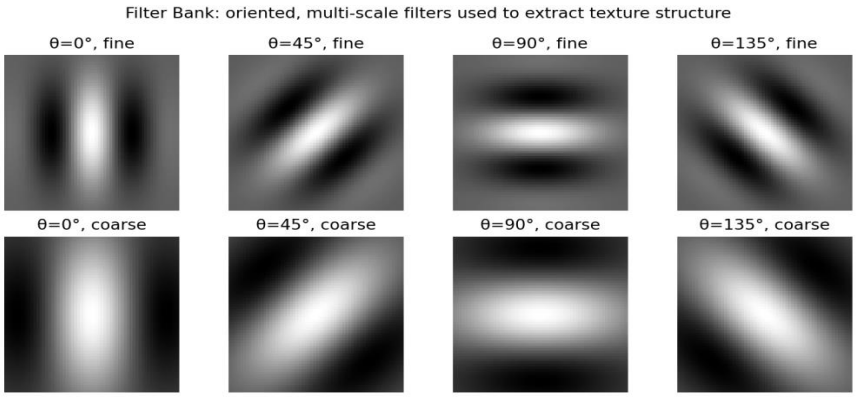
Given an image $I(x, y)$ and a kernel (also called a *mask*, *filter*, or *point-spread function*) $h(x, y)$ of size $(2k + 1) \times (2k + 1)$, the **2-D discrete convolution** is:

$$(I * h)(x,y) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} h\,(i,j)\, I(x-i,\, y-j)$$

Note the *kernel is flipped* (both axes reversed) relative to **correlation**:

$$(I \otimes h)(x,y) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} h\,(i,j)\, I(x+i,\, y+j)$$

For symmetric kernels (e.g., the Gaussian), convolution and correlation coincide.



Convolution: kernel slides over the image, computing a weighted sum at each position

1.2.4 Properties of Convolution

1. **Commutativity:** $f * g = g * f$
2. **Associativity:** $(f * g) * h = f * (g * h)$
3. **Distributivity over addition:** $f * (g + h) = f * g + f * h$
4. **Shift invariance:** if $f'(x) = f(x - x_0)$ then $(f' * g)(x) = (f * g)(x - x_0)$
5. **Linearity** in each argument.
6. **Separability:** if $h(x,y) = h_x(x)\, h_y(y)$, the 2-D convolution can be computed as two 1-D convolutions (row pass then column pass), reducing complexity from $O(N^2k^2)$ to $O(N^2k)$ for an $N \times N$ image and $k \times k$ kernel.

1.2.5 Common Kernels

Box filter (mean filter) — averages a neighborhood, blurring the image and suppressing noise:

$$h = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Gaussian filter — weights fall off smoothly with distance, giving a more natural blur with no ringing:

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

Sharpening kernel:

$$h = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Identity kernel: a single 1 at the center, all else zero — leaves the image unchanged.

1.2.5(c) Unsharp Masking: Sharpening as a Linear Combination of Filters

A classic sharpening technique, **unsharp masking**, is easily understood as a combination of the identity and Gaussian smoothing kernels already introduced:

$$I_{sharp} = I + \alpha\,(I - I * G_\sigma) = (1+\alpha)I - \alpha\,(I * G_\sigma)$$

The term $(I - I * G_\sigma)$ is the **high-frequency residual** — everything removed by Gaussian blurring — and adding a scaled copy of it back to the original amplifies fine detail and edges (high spatial frequencies) while leaving smooth regions largely unchanged. This shows explicitly that the "sharpening kernel" shown earlier is not a fundamentally different tool from Gaussian smoothing, but rather a specific linear combination of the identity filter and a low-pass filter — reinforcing the linear-systems perspective (§1.2.2) that complex-seeming operations are frequently built from simple linear building blocks.

Worked Problem 1.1(d)

Using unsharp masking with $\alpha = 1$, express the equivalent single convolution kernel in terms of a 3×3 approximate Gaussian $G \approx \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$.

Solution.

$$h_{sharp} = (1+\alpha)\,\delta - \alpha\,G = 2\delta - G$$

where δ is the identity kernel (1 at center, 0 elsewhere). Substituting:

$$h_{sharp} = 2\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} = \frac{1}{16}\begin{bmatrix} -1 & -2 & -1 \\ -2 & 28 & -2 \\ -1 & -2 & -1 \end{bmatrix}$$

This derived kernel is recognizable as a standard sharpening kernel (a strong positive center weight surrounded by negative weights), confirming that unsharp masking and the "sharpening kernel" presented earlier are the same operation viewed from two different but equivalent perspectives.

1.2.5(b) Choosing Kernel Support Size for a Given σ

In practice, the (theoretically infinite-extent) Gaussian must be truncated to a finite kernel. Since the Gaussian decays rapidly, truncating at a radius of about 3σ retains more than 99.7% of the kernel's total weight (by the standard normal distribution's three-sigma rule), so a common rule of thumb sizes the kernel as:

$$k = 2\lceil 3\sigma \rceil + 1$$

ensuring an odd kernel size with a well-defined center pixel and negligible truncation error. For example, $\sigma = 2$ pixels gives $k = 2(6) + 1 = 13$, i.e., a 13×13 kernel.

Worked Problem 1.1(c)

What kernel size should be used for a Gaussian smoothing filter with $\sigma = 4$ pixels, using the 3σ rule? What fraction of the kernel's total weight would be excluded if a smaller 9×9 kernel were used instead?

Solution.

Recommended size: $k = 2\lceil 3(4) \rceil + 1 = 2(12) + 1 = 25$, i.e., a 25×25 kernel.

A 9×9 kernel only extends to radius 4 pixels $= 1\sigma$ from the center. By the standard normal distribution, the fraction of a Gaussian's mass lying within $\pm 1\sigma$ is approximately 68.3% (1-D); in 2-D (a circularly symmetric Gaussian truncated to a square window is slightly more complex, but as an order-of-magnitude estimate using the 1-D marginal in each axis), a substantial fraction of the kernel's weight — easily over 30% — would be excluded, meaning the filter's effective smoothing behavior would deviate noticeably from the intended $\sigma = 4$ Gaussian, effectively behaving like a smaller, sharper filter than specified. This is why under-sized truncation is a common practical bug in filter implementations.

1.2.6 Worked Problem 1.1

Compute the convolution output at the center pixel for the 3×3 image patch and kernel below.

$$I = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}, \qquad h = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \text{ (discrete Laplacian)}$$

Solution. For convolution the kernel must be flipped by 180°. Since this Laplacian kernel is symmetric under 180° rotation (it is unchanged), flipping has no effect here. We align the kernel center on $I(1,1) = 5$ (using 0-indexing) and compute:

$$\text{output} = 0(1) + 1(2) + 0(3) + 1(4) + (-4)(5) + 1(6) + 0(7) + 1(8) + 0(9)$$

$$= 2 + 4 - 20 + 6 + 8 = 0$$

The Laplacian response is 0, meaning the intensity at the center varies linearly across the neighborhood (no curvature) — the discrete Laplacian is a curvature/edge detector, developed further in Unit II.

1.2.7 Border Handling

Convolution near image boundaries requires a padding convention:

- **Zero padding** — pad with zeros; introduces dark artificial edges.
- **Replicate (clamp)** — repeat edge pixel values.
- **Reflect (mirror)** — mirror the image across the border.
- **Wrap (circular)** — treat the image as periodic (natural for FFT-based convolution).

1.2.8 Computational Cost of Convolution

For an $N \times N$ image and a $k \times k$ kernel, direct spatial-domain convolution requires, at every output pixel, k^2 multiplications and $k^2 - 1$ additions, giving total cost:

$$\text{Cost}_{direct} = O(N^2 k^2)$$

Two standard strategies reduce this cost substantially:

1. **Separable kernels** (§1.2.4, property 6): if $h(x, y) = h_x(x) h_y(y)$, convolution can be performed as one 1-D pass along rows followed by one 1-D pass along columns, reducing the cost to:

$$\text{Cost}_{separable} = O(N^2 k)$$

which for a $k = 15$ kernel is already an order-of-magnitude saving, and grows more favorable as k increases.

2. **Frequency-domain convolution** (justified fully in §1.4.3): transform the image via the Fast Fourier Transform ($O(N^2 \log N)$), multiply point-wise by the kernel's transform ($O(N^2)$), and inverse-transform ($O(N^2 \log N)$):

$$\text{Cost}_{FFT} = O(N^2 \log N)$$

FFT-based convolution outperforms direct spatial convolution whenever the kernel is large relative to $\log N$ — precisely the regime encountered with the large-support filters used in scale-space and texture analysis (Unit III).

1.2.9 Worked Problem 1.1(b)

An image is 1024×1024 *pixels. A non-separable Gaussian-like kernel of size* 21×21 *is to be applied. Compare the number of multiplications required by (a) direct convolution and (b) separable convolution (assuming the kernel can be well-approximated as separable).*

Solution.

(a) Direct: $\text{Cost} = N^2 k^2 = (1024)^2 \times (21)^2 = 1{,}048{,}576 \times 441 \approx 4.62 \times 10^8$ multiplications.

(b) Separable: $\text{Cost} = N^2 \times 2k = 1{,}048{,}576 \times 42 \approx 4.40 \times 10^7$ multiplications.

The separable implementation requires roughly **10.5 times fewer multiplications** — this is exactly why the Gaussian's separability (§1.6, and its rotational symmetry, discussed further in Unit II) is repeatedly emphasized as a major practical, and not merely theoretical, advantage.

1.2.10 Convolution vs. Correlation in Practice

Despite the mathematical distinction drawn in §1.2.3, most computer vision libraries and much of the applied literature use the term "convolution" loosely to refer to what is technically correlation (i.e., without flipping the kernel), since:

1. For the overwhelmingly common **symmetric kernels** (Gaussian, box, Laplacian, LoG) used in smoothing and isotropic edge/blob detection, flipping has no effect at all — convolution and correlation give identical results.
2. Even for asymmetric kernels (e.g., a directional derivative filter), whether one calls the operation "convolution" or "correlation" is immaterial as long as the *same* convention is used consistently when designing and applying the filter — the kernel can simply be pre-flipped once, at design time, to compensate.
3. Deep learning frameworks (e.g., convolutional neural networks) almost universally implement **correlation**, not true convolution, despite the "convolution" name — a naming convention now so entrenched in the field that it is important to be aware of, even though it does not affect network training (since the kernel weights are learned, the network simply learns the flipped version of whatever a true convolution would have learned).

Practical guideline: when implementing a filter from a known, published formula (e.g., a Gaussian derivative formula), always verify whether the source defines the operation as convolution or correlation, and flip the kernel accordingly if the two conventions must match exactly (this matters most when directly comparing numerical output against a reference implementation or textbook worked example).

Worked Problem 1.1(e)

A 1-D kernel $h = [1,2,3]$ *(asymmetric) is applied to signal* $I = [0,0,1,0,0]$ *using (a) true convolution and (b) correlation, at the position where the kernel is centered on the '1' in the signal. Show the two results differ.*

Solution.

Center the kernel (length 3, so 1 pixel on each side of center) at the '1' (index 2, 0-indexed): the overlapping signal window is $I = [0,1,0]$ (indices 1,2,3).

Convolution flips the kernel first: flipped h is [3,2,1].

$$(I * h) = 0(3) + 1(2) + 0(1) = 2$$

Correlation uses the kernel as-is: $h = [1,2,3]$.

$$(I \otimes h) = 0(1) + 1(2) + 0(3) = 2$$

In this particular symmetric-signal case the two happen to coincide (both give 2), because the signal window itself is symmetric ($[0,1,0]$) even though the kernel is not. To see a genuine difference, shift the window to $I = [1,0,0]$ (kernel centered one position to the left, at index 1): convolution gives $1(3) + 0(2) + 0(1) = 3$, while correlation gives $1(1) + 0(2) + 0(3) = 1$ — confirming the two operations are generally distinct for asymmetric kernels, and that care must be taken about which convention is intended.

1.3 Shift-Invariant Linear Systems

A system is **shift-invariant** (also *space-invariant* or *time-invariant* in 1-D signals) if shifting the input shifts the output identically, without otherwise altering it:

$$\text{if } \mathcal{H}[f(x,y)] = g(x,y), \text{ then } \mathcal{H}[f(x-x_0, y-y_0)] = g(x-x_0, y-y_0)$$

Fundamental theorem. Every linear, shift-invariant (LSI) system can be completely characterized by its response to a unit impulse — its **impulse response** (or *point-spread function*) $h(x,y)$. The output for *any* input is then given by convolving the input with h:

$$g = f * h$$

This is the single most important fact in linear filtering theory: instead of characterizing a system for every possible input, we need only measure its impulse response, and convolution predicts the output for all inputs.

Impulse (Dirac delta) in the discrete case:

$$\delta(x,y) = \begin{cases} 1 & (x,y) = (0,0) \\ 0 & \text{otherwise} \end{cases}, \qquad \mathcal{H}[\delta] = h$$

Cascading LSI systems. If an image passes through two LSI systems h_1 then h_2 in series, the overall system is equivalent to a single filter $h = h_1 * h_2$ (by associativity of convolution). This justifies building complex filters (e.g., a smoothed derivative) by convolving simpler filters together.

Worked Problem 1.2

Two LSI systems with impulse responses $h_1 = [1,1]$ and $h_2 = [1,-1]$ (1-D) are cascaded. Find the equivalent single impulse response, and comment on what operation it performs.

Solution.

$$h = h_1 * h_2$$

Convolving $[1,1]$ with $[1,-1]$:

$$h[n] = \sum_k h_1[k] h_2[n-k]$$

Index $h_1 = \{h_1[0] = 1, h_1[1] = 1\}$, $h_2 = \{h_2[0] = 1, h_2[1] = -1\}$.

$h[0] = h_1[0]h_2[0] = 1 \quad h[1] = h_1[0]h_2[1] + h_1[1]h_2[0] = -1 + 1 = 0 \quad h[2] = h_1[1]h_2[1] = -1$

So $h = [1,0,-1]$ — this is exactly the **central-difference derivative operator**, confirming that a running-sum smoother cascaded with a first-difference produces a smoothed derivative filter, a key idea used in Unit II (derivative-of-Gaussian filters).

1.3.1 Characterizing a System via Its Frequency Response

Because convolution in the spatial domain is multiplication in the frequency domain (proved formally in §1.4.3), an LSI system's impulse response h and its **frequency response** $H(u,v) = \mathcal{F}\{h\}$ are two equivalent, fully-informative descriptions of the same system — the choice of which to use is a matter of convenience for the task at hand. Spatial-domain impulse responses are natural for describing *what a filter does to a local neighborhood* (as in the template-matching view, §1.6); frequency responses are natural for describing *what frequencies a filter passes or blocks* (as in the low/high/band-pass classification, §1.4.3).

Worked Problem 1.2(b) — 2-D Cascade of Separable Systems

A 2-D LSI system is built by cascading a horizontal averaging filter $h_x = \frac{1}{3}[1,1,1]$ *with a vertical derivative filter* $h_y = [1,0,-1]^T$ *(applied as separate 1-D passes, first along rows then along columns). Write the overall equivalent 2-D impulse response as an explicit* 3×3 *matrix, and identify what kind of operator this is.*

Solution.

Since the two 1-D filters are applied along orthogonal axes, the overall 2-D impulse response is their **outer product**:

$$h = h_y\, h_x^T = \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix} \begin{bmatrix} \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \end{bmatrix} = \begin{bmatrix} \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ 0 & 0 & 0 \\ -\frac{1}{3} & -\frac{1}{3} & -\frac{1}{3} \end{bmatrix}$$

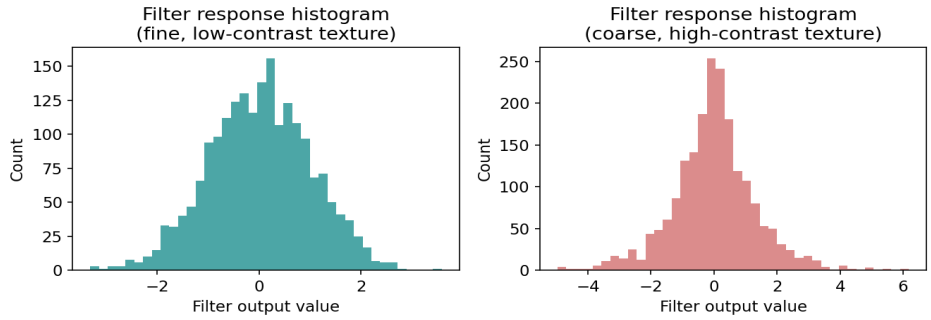
Multiplying through by 3 for a cleaner integer kernel: $\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$

This is precisely the (unnormalized) **Prewitt vertical-edge operator** G_y studied further as a gradient-based edge detector in Unit II (§2.8.2) — confirming that classic edge-detection kernels are themselves simply cascades of a smoothing pass (here, horizontal averaging, which suppresses noise along the edge direction) and a differencing pass (here, vertical differencing, which detects the edge itself), exactly the smoothing+differentiation principle developed at length in Unit II.

1.4 Spatial Frequency and Fourier Transforms

1.4.1 Why frequency matters

Any image can be expressed as a weighted sum of sinusoidal gratings of different **spatial frequencies** (cycles per unit distance) and orientations. Slowly-varying regions (smooth shading) are dominated by low spatial frequencies; sharp edges and fine texture contain high spatial frequencies. Understanding an image (and a filter) in the frequency domain often makes its behavior far more transparent than in the spatial domain.



Left: a spatial-domain signal built from two sinusoids. Right: its frequency-domain representation showing the two frequency components.

1.4.2 The Fourier Transform

The continuous 2-D Fourier Transform of an image $f(x, y)$ is:

$$F(u, v) = \iint_{-\infty}^{\infty} f(x, y)\, e^{-j2\pi(ux+vy)}\, dx\, dy$$

and the inverse transform reconstructs the image from its frequency content:

$$f(x, y) = \iint_{-\infty}^{\infty} F(u, v)\, e^{j2\pi(ux+vy)}\, du\, dv$$

Here u, v are spatial frequencies (cycles/unit length) along x and y. $F(u, v)$ is complex-valued in general; its magnitude $|F(u, v)|$ is the **amplitude spectrum** and its argument is the **phase spectrum**.

Discrete Fourier Transform (DFT) for an $N \times N$ image:

$$F(u, v) = \frac{1}{N^2} \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} I(x, y)\, e^{-j2\pi(ux+vy)/N}$$

1.4.3 The Convolution Theorem

The single most powerful property linking filtering to frequency analysis:

$$f * h \;\Leftrightarrow\; F(u,v) \cdot H(u,v)$$

Convolution in the spatial domain equals point-wise multiplication in the frequency domain, and vice-versa. This is why filters are often described by what frequencies they pass or block:

- **Low-pass filters** (e.g., Gaussian, box) attenuate high frequencies → smoothing/blurring.
- **High-pass filters** amplify high frequencies → sharpening/edge enhancement.
- **Band-pass filters** (e.g., Gabor, oriented pyramids) respond to a specific frequency band and orientation → used for texture analysis (Unit III).

$H(u,v)$, the Fourier transform of the kernel h, is called the filter's **transfer function** or **frequency response**.

1.4.4 Properties of the Fourier Transform

Property	Spatial domain	Frequency domain
Linearity	$af_1 + bf_2$	$aF_1 + bF_2$
Shift	$f(x - x_0, y - y_0)$	$F(u,v)e^{-j2\pi(ux_0+vy_0)}$
Scaling	$f(ax, by)$	$\frac{1}{\lvert ab\rvert}F(u/a, v/b)$
Convolution	$f * h$	$F \cdot H$
Multiplication	$f \cdot h$	$F * H$
Differentiation	$\partial f / \partial x$	$j2\pi u\, F(u,v)$
Parseval's theorem	$\sum\lvert f\rvert^2$	$\sum\lvert F\rvert^2$ (energy preserved)

1.4.4(b) The Relative Importance of Phase and Magnitude

An instructive and often surprising property of the 2-D Fourier transform is that the **phase spectrum**, not the magnitude spectrum, carries most of the perceptually important structural information in a natural image. A classic demonstration: if the magnitude spectra of two unrelated images are swapped while keeping each image's own phase spectrum, the reconstructed images remain recognizably close to their *original* (phase-donor) content, not the magnitude-donor content — object boundaries, shapes, and edges are encoded overwhelmingly in phase relationships between frequency components, while magnitude mainly affects overall contrast and texture "roughness." This is a useful conceptual check when reasoning about frequency-domain filters: operations that manipulate magnitude only (as most simple low-pass/high-pass filters do, since they multiply $F(u,v)$ by a real-valued, phase-preserving transfer function $H(u,v)$) preserve the essential spatial structure of the image, whereas operations that corrupt phase (e.g., certain non-linear frequency-domain manipulations) can produce dramatic, spatially incoherent distortions even with magnitude left completely unchanged.

1.4.5 Worked Problem 1.3

A Gaussian filter has transfer function $H(u) = e^{-2\pi^2\sigma^2u^2}$ (1-D). Explain, using this formula, why increasing σ in the spatial domain produces heavier smoothing.

Solution. As σ increases, the exponent $-2\pi^2\sigma^2u^2$ becomes more negative for any fixed non-zero u, so $H(u) \to 0$ faster as u grows — i.e., the transfer function's effective bandwidth **shrinks**. A narrower frequency response means more high-frequency content is attenuated, which manifests spatially as stronger blurring. This is a specific instance of the general **uncertainty principle**: a Gaussian's spatial width and its frequency-domain width are inversely related ($\sigma_x \cdot \sigma_u \geq$ constant) — a filter cannot be made arbitrarily narrow (sharp) in both domains simultaneously.

1.4.6 Worked Problem 1.3(b) — Computing a Small DFT by Hand

Compute the 1-D DFT of the length-4 sequence $I = [1,2,1,0]$ *using* $F(u) = \sum_{x=0}^{3} I\,(x)e^{-j2\pi ux/4}$.

Solution.

For $u = 0$: $F(0) = \sum I(x) = 1 + 2 + 1 + 0 = 4$ (the DC term — the sum/average intensity).

For $u = 1$: $F(1) = 1 \cdot e^{0} + 2e^{-j\pi/2} + 1 \cdot e^{-j\pi} + 0 \cdot e^{-j3\pi/2} = 1 + 2(-j) + 1(-1) + 0 = 1 - 2j - 1 = -2j$

For $u = 2$: $F(2) = 1 \cdot e^{0} + 2e^{-j\pi} + 1e^{-j2\pi} + 0e^{-j3\pi} = 1 - 2 + 1 - 0 = 0$

For $u = 3$: $F(3) = 1 + 2e^{-j3\pi/2} + 1e^{-j3\pi} + 0 = 1 + 2(j) + (-1) = 2j$

So $F = [4,\ -2j,\ 0,\ 2j]$. Note $F(1)$ and $F(3)$ are complex conjugates ($-2j$ and $+2j$) — a general property of the DFT of a real-valued signal (**conjugate symmetry**, $F(-u) = F(u)^*$), which is why only half the frequency components of a real image need be stored/considered independently. The zero magnitude at $u = 2$ (the Nyquist frequency of this length-4 sequence) indicates no energy at the fastest-alternating pattern $[+,-,+,-]$ within this particular signal.

1.4.7 Separability of the 2-D Fourier Transform

Just as many spatial-domain filters are separable (§1.2.4), the 2-D Fourier transform itself is separable into two independent 1-D transforms when the underlying image (or kernel) is separable, or in general can always be *computed* as a 1-D transform along rows followed by a 1-D transform along columns, regardless of whether the image itself is separable:

$$F(u,v) = \sum_{x} e^{-j2\pi ux/N} \left[\sum_{y} I\,(x,y) e^{-j2\pi vy/N} \right]$$

This row-then-column decomposition is exactly how the 2-D Fast Fourier Transform is implemented in practice, reducing the cost from a direct $O(N^4)$ 2-D DFT computation to $O(N^2 \log N)$ by applying the 1-D FFT ($O(N \log N)$) to each of the N rows and then each of the N columns.

1.4.8 Worked Problem 1.3(c)

Estimate the speedup of using row-column FFT decomposition over a direct (brute-force) 2-D DFT for a 256×256 *image.*

Solution.

Direct 2-D DFT cost: $O(N^4) = 256^4 \approx 4.29 \times 10^9$ operations.

Row-column FFT cost: $O(N^2 \log_2 N) = 256^2 \times 8 = 65{,}536 \times 8 = 524{,}288$ operations.

Speedup factor $\approx 4.29 \times 10^9 / 5.24 \times 10^5 \approx 8{,}192\times$ — an enormous, practically essential speedup that is the reason FFT-based methods (rather than the definitional DFT sum) are used for every real image-processing frequency-domain computation, including the FFT-based convolution introduced in §1.2.8.

1.5 Sampling and Aliasing

1.5.1 From continuous scene to discrete image

A camera sensor measures a continuous light signal at discrete spatial locations (pixels) — this is **sampling**. Sampling theory tells us exactly when this discretization loses information.

1.5.2 The Sampling Theorem (Nyquist–Shannon)

If a continuous signal $f(x)$ is **band-limited**, i.e., its Fourier transform is zero outside $|u| \leq u_{\max}$, then $f(x)$ can be **exactly reconstructed** from samples taken at a rate

$$f_s \geq 2\, u_{\max} \quad \text{(the Nyquist rate)}$$

Sampling below this rate causes **aliasing**: high frequencies masquerade as (are folded down to) lower, incorrect frequencies in the sampled signal, corrupting it irrecoverably.

Mathematical view. Sampling multiplies the continuous signal by a train of impulses (a "comb" function) spaced at interval $T = 1/f_s$. By the convolution theorem, multiplication in the spatial domain corresponds to convolution in the frequency domain — sampling therefore replicates the signal's spectrum $F(u)$ at intervals of f_s:

$$F_{sampled}(u) = \sum_{k=-\infty}^{\infty} F\,(u - k f_s)$$

If $f_s < 2u_{\max}$, adjacent spectral copies **overlap**, and the overlapping high-frequency energy is added into the low-frequency band — this overlap *is* aliasing.

1.5.3 Aliasing in Images

Classic visual symptoms of aliasing:

- **Moiré patterns** when photographing fine repetitive textures (e.g., striped fabric, window screens).
- **Jagged (staircase) edges** on diagonal lines — a spatial-domain manifestation of high-frequency edge content being under-sampled.
- **Wagon-wheel effect** in video — temporal aliasing of rotating spokes.

1.5.4 Anti-Aliasing

Because a sensor's sampling rate (pixel pitch) is fixed by hardware, the practical remedy is to **low-pass filter the signal before sampling** (an *anti-aliasing filter*), removing frequencies above the Nyquist limit so nothing is left to fold over. In digital image resizing (down-sampling), this means blurring (e.g., with a Gaussian) before subsampling — precisely the procedure used to build image pyramids (Section 1.7).

1.5.5 Worked Problem 1.4

A camera sensor has pixels spaced 4 microns apart. What is the highest spatial frequency (in cycles/mm) that can be captured without aliasing?

Solution. Sampling interval $T = 4\ \mu m = 0.004\ mm$. Sampling rate $f_s = 1/T = 250$ cycles/mm. By Nyquist, $u_{\max} = f_s/2 = 125$ cycles/mm.

Any scene detail finer than $1/125\ mm \approx 8\ \mu m$ per cycle (i.e., higher than 125 cycles/mm) will alias. This is why camera systems place an optical low-pass (anti-aliasing) filter in front of the sensor.

1.5.6 Worked Problem 1.4(b) — Aliasing in Image Downsampling

A 1200×1200 *image is to be reduced to* 300×300 *by simple pixel-dropping (keeping every 4th pixel, with no pre-filtering). Explain, quantitatively, why this will alias, and state the correct procedure.*

Solution.

Dropping every 4th pixel reduces the sampling rate by a factor of 4, so the new Nyquist frequency is $u_{\max}{}' = u_{\max}/4$ — any spatial frequency in the original image between $u_{\max}/4$ and $u_{\max}$ (three-quarters of the original representable frequency band) will violate the new, lower Nyquist limit and **fold back (alias)** into the visible low-frequency range of the downsampled image, typically appearing as false, jagged, or moiré-like patterns bearing no resemblance to the true (locally smooth) original content.

Correct procedure: first low-pass filter the 1200×1200 image with a filter whose cutoff is at (or below) the *new* Nyquist limit $u_{\max}/4$ (e.g., a Gaussian blur with an appropriately chosen σ, per the pyramid construction of §1.8.2), and only then subsample by keeping every 4th pixel. This is precisely the "smooth-then-subsample" pattern used to build every level of a Gaussian pyramid, and the general justification for why naive/nearest-neighbor image resizing tools frequently produce visibly aliased results compared to those using a proper anti-aliasing (area-averaging or Gaussian pre-filter) resize algorithm.

1.5.7 Aliasing and Color Sensing

The Bayer color-filter mosaic introduced in §1.1.1 effectively samples each color channel (red, green, blue) at a *lower* spatial sampling rate than the full pixel grid (e.g., red and blue are each sampled at only 1/4 the pixel density, green at 1/2), since each physical sensor site captures only one color. This makes color images especially prone to color-fringing aliasing artifacts near

sharp, high-contrast edges unless the camera's optical anti-aliasing filter and demosaicing algorithm are carefully designed — a direct, practical consequence of the sampling theorem covered in this section.

1.6 Filters as Templates

1.6.1 Correlation as pattern matching

Convolving (or correlating) an image with a small kernel can be interpreted as **asking, at every pixel, "how much does the local neighborhood look like this template?"** The kernel acts as a **template**; the filter response is large wherever the local image patch is similar (well-aligned) to the template, and small/negative where it is dissimilar.

This interpretation unifies many operations:

- A Gaussian-derivative kernel used as a template responds strongly to step-edges of matching orientation and scale (Unit II).
- An oriented Gabor filter used as a template responds strongly to texture of matching frequency/orientation (Unit III).
- A grayscale patch of an object (a "chip") used as a template locates instances of that object via correlation (Section 1.7).

1.6.1(b) Limits of Simple Template Matching

While powerful, plain template matching (whether raw correlation or NCC) has well-understood limitations that motivate more sophisticated recognition techniques developed later in the course (Unit V): it assumes the object appears with a fixed, known appearance (no deformation, no significant lighting change beyond a global affine brightness shift, no perspective distortion), and it requires an exhaustive search over all candidate positions (and, per §1.6.3, scales and orientations), which becomes expensive for large images or large numbers of object classes. Unit V's relational and probabilistic recognition methods (voting on relations between templates, Hidden Markov Models) build directly on the template-matching concept introduced here, but relax the rigid, single-fixed-template assumption by allowing flexible spatial relationships between multiple smaller templates/parts.

1.6.2 Response as a Dot Product

For a kernel h of size $k \times k$ and image patch p of the same size, correlation at that location is:

$$\text{response} = h \cdot p = \sum_{i,j} h(i,j)\, p(i,j) = \| h \| \, \| p \| \, cos\theta$$

where θ is the angle between the kernel and patch, viewed as vectors in $\mathbb{R}^{k^2}$. This shows correlation is maximized (for fixed patch energy $\| p \|$) exactly when the patch is proportional to the template (i.e., $\theta = 0$) — confirming the "template matching" interpretation geometrically.

1.6.3 Templates at Multiple Scales and Orientations

A single fixed template can only match patterns of the same size and orientation as the template itself. In practice, patterns of interest appear scaled and rotated, so template-matching systems typically:

- Build a small **bank** of rotated copies of the template (e.g., every 15°–30°) and correlate with each, keeping the best-scoring orientation at every pixel — an early precursor to the oriented filter banks used for texture (Unit III).
- Search across an **image pyramid** (§1.8) using a single fixed-size template, so that a pattern appearing at half the original scale is matched at the pyramid level where it once again matches the template's native size.

Confound	Effect on raw correlation	Standard remedy
Brightness/contrast changes	Score changes even for identical shape	Normalized cross-correlation (§1.7)
Object scale changes	Template no longer matches proportions	Search across an image pyramid (§1.8)
Object rotation	Template no longer aligns with structure	Bank of rotated templates / oriented filters
Partial occlusion	Score reduced even for a true match	Robust matching (e.g., matching sub-patches)

1.6.4 Matched Filtering and Optimal Detection

A deeper justification for the "template as filter" idea comes from **matched-filter theory**: if a known pattern t is corrupted by additive white Gaussian noise (§2.2, developed fully in Unit II), the linear filter that **maximizes the output signal-to-noise ratio (SNR)** at the true pattern location is exactly the correlation filter using t itself (or its time/space-reversed version, for convolution) as the kernel:

$$SNR_{out} = \frac{|\text{ response at true location }|^2}{\text{noise variance at output}}$$

This result — that correlation with the template *is* the provably optimal linear detector for a known pattern in white noise — is why template matching (and its generalizations, such as matched filters in radar/communications and correlation-based feature detectors in vision) is not merely a heuristic but a principled, optimal strategy under this specific noise model. It also explains why NCC (§1.7), by removing brightness/contrast confounds while preserving the correlation structure, retains this near-optimality property in practical (non-ideal) imaging conditions.

1.7 Technique: Normalized Correlation and Finding Patterns

1.7.1 The problem with raw correlation

Raw correlation (or convolution) scores are sensitive to two confounds:

1. **Contrast/brightness** — a bright patch that is a poor match can still produce a large raw correlation score simply due to high pixel magnitudes.
2. **Template energy** — templates with larger overall magnitude generate larger scores regardless of matching quality.

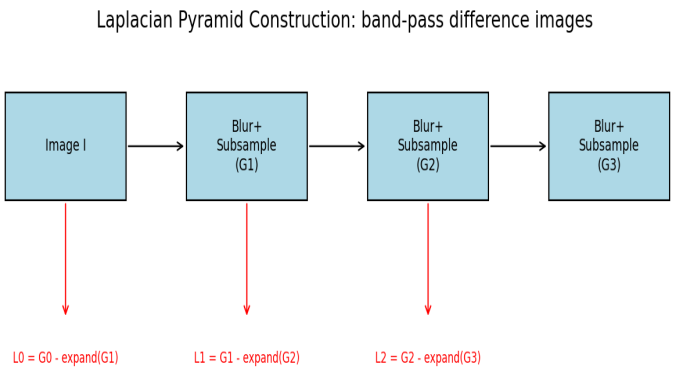
Normalized cross-correlation (NCC) fixes both problems by normalizing for local mean and standard deviation, producing a score in $[-1,1]$, invariant to affine changes in brightness/contrast.

1.7.2 Normalized Cross-Correlation formula

For template t and image window I (both size $k \times k$), with means $\bar{t}, \bar{I}$ over the window:

$$NCC(x,y) = \frac{\sum_{i,j}\left(I(x+i,y+j)-\bar{I}\right)\left(t(i,j)-\bar{t}\right)}{\sqrt{\sum_{i,j}\left(I(x+i,y+j)-\bar{I}\right)^2}\ \sqrt{\sum_{i,j}\left(t(i,j)-\bar{t}\right)^2}}$$

- $NCC = 1$: perfect positive linear match.
- $NCC = 0$: no linear correlation.
- $NCC = -1$: perfect inverse match.



Normalized Cross-Correlation: the template (red box) slides over the image; the location with the highest correlation score is the best match.

1.7.3 Finding Patterns — the sliding-window algorithm

1. Slide the template over every location (x, y) in the search image.
2. Compute $NCC(x, y)$ at each location.
3. The location(s) where $NCC(x, y)$ exceeds a threshold (or is the global maximum) are declared matches.
4. Optionally, run at multiple scales/rotations of the template to handle scale/orientation changes (see image pyramids next).

1.7.4 Worked Problem 1.5

Given a 1×3 template $t = [2,4,6]$ *and an image window* $I = [1,2,3]$*, compute the normalized cross-correlation.*

Solution.

$\bar{t} = (2+4+6)/3 = 4, \bar{I} = (1+2+3)/3 = 2$

Numerator: $\sum(I_i - \bar{I})(t_i - \bar{t}) = (1-2)(2-4) + (2-2)(4-4) + (3-2)(6-4) = (-1)(-2) + (0)(0) + (1)(2) = 2 + 0 + 2 = 4$

Denominator: $\sqrt{\sum(I_i - \bar{I})^2} \cdot \sqrt{\sum(t_i - \bar{t})^2}$ $\sum(I_i - \bar{I})^2 = 1 + 0 + 1 = 2 \Rightarrow \sqrt{2}$ $\sum(t_i - \bar{t})^2 = 4 + 0 + 4 = 8 \Rightarrow \sqrt{8} = 2\sqrt{2}$ Denominator $= \sqrt{2} \times 2\sqrt{2} = 2 \times 2 = 4$

$$NCC = \frac{4}{4} = 1.0$$

A perfect match (NCC = 1) confirms that I is an exact positive affine (linear brightness) transform of t — indeed $I_i = t_i/2$ exactly, a pure contrast scaling, which NCC correctly identifies as a perfect match, whereas raw correlation would have scored this incorrectly lower than a same-brightness mismatch.

1.7.5 Worked Problem 1.5(b) — Practical Application

A quality-control system searches a 640×480 *grayscale image for a* 40×40 *reference "chip" template (e.g., a printed logo) using NCC. (a) How many candidate window positions must be evaluated (ignoring borders)? (b) If the true logo appears in the image scaled to 70% of the template's size due to a slightly farther camera placement, will NCC at the original scale reliably find it? Explain and propose a fix.*

Solution.

(a) Valid top-left positions: $(640 - 40 + 1) \times (480 - 40 + 1) = 601 \times 441 = 265{,}041$ candidate positions — every one of these requires computing an NCC score, illustrating why efficient implementations (FFT-based cross-correlation, exploiting the convolution theorem of §1.4.3) are used in practice rather than a naive nested loop.

(b) No — NCC as defined compares patches of identical size; a logo appearing at 70% scale would have different internal spatial-frequency content and would **not** align well pixel-for-pixel with the full-size template, so its NCC score would likely fall below the detection threshold even at the correct location (a partial, imperfect match at best, from residual similarity of coarse structure). The correct fix is precisely the multi-scale search strategy of §1.8: build an image pyramid (or a scaled set of templates) and repeat the NCC search at each scale, reporting the best match found across all scales — the general solution to the scale-sensitivity problem noted in the confound table of §1.6.3.

1.8 Technique: Scale and Image Pyramids

1.8.1 Motivation — the problem of scale

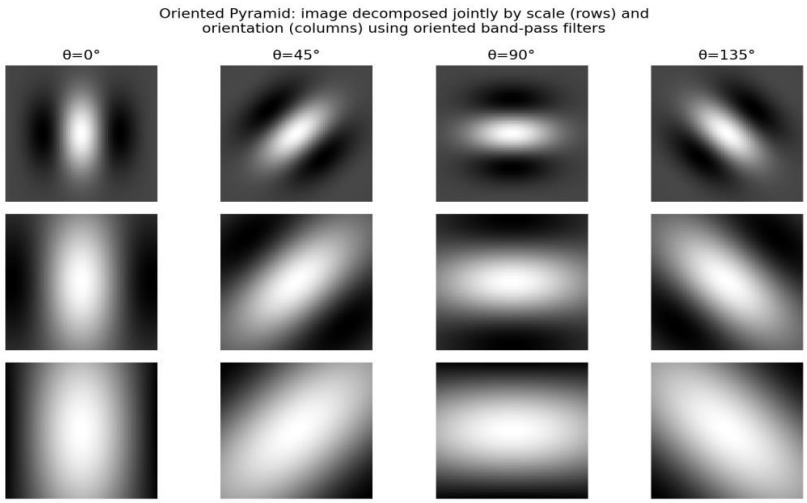
Objects and patterns of interest appear at different sizes in an image depending on distance from the camera. A single, fixed-size template or filter cannot detect a pattern at all possible scales efficiently. **Image pyramids** provide a systematic multi-resolution representation of an image, enabling search and analysis across scales.

1.8.2 The Gaussian Pyramid

Built by repeatedly **smoothing** (with a Gaussian filter, to satisfy the sampling theorem and avoid aliasing) and **subsampling** (typically by a factor of 2) the image:

$$G_0 = I \qquad G_l = \downarrow_2 (G_{l-1} * G_\sigma)$$

where $\downarrow_2$ denotes taking every second pixel in each dimension. Each level G_l has roughly a quarter the pixels of G_{l-1} and represents the image at half the linear resolution / spatial frequency band.



Gaussian / Image Pyramid: repeated smoothing and subsampling produces a multi-resolution image stack.

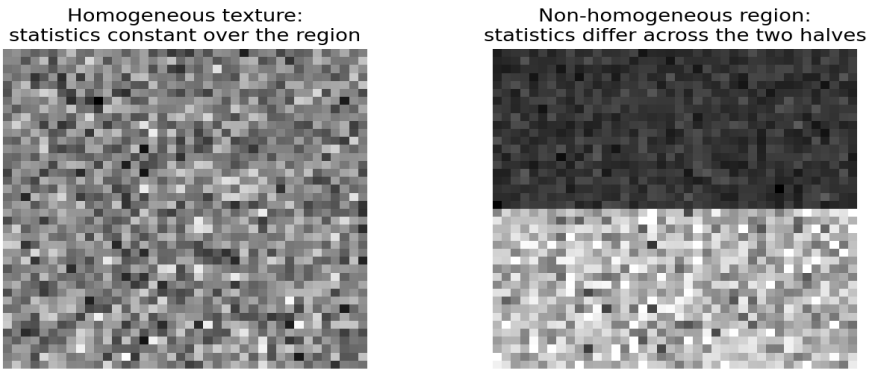
Why smoothing before subsampling is essential. Subsampling by 2 halves the effective sampling rate. By the Nyquist theorem, any content above half the previous Nyquist frequency will alias unless removed first — this is precisely why a low-pass (Gaussian) filter must precede subsampling at every pyramid level.

1.8.3 The Laplacian Pyramid

While the Gaussian pyramid is a smoothing/multi-resolution representation, the **Laplacian pyramid** captures the *band-pass* detail lost between successive Gaussian levels — it is effectively a multi-scale edge/detail representation and permits exact reconstruction of the original image.

$$L_l = G_l - \text{Expand}(G_{l+1})$$

where Expand$(\cdot)$ upsamples G_{l+1} back to the resolution of G_l (e.g., via interpolation) before subtraction.



Laplacian Pyramid construction: each level stores the band-pass detail lost when moving down to the next Gaussian level.

Reconstruction. The original image can be exactly recovered by reversing the process:

$$G_l = L_l + \text{Expand}(G_{l+1})$$

applied recursively from the coarsest level back up to $G_0 = I$.

1.8.4 Uses of Pyramids

- **Scale-invariant template matching / pattern finding** — search each pyramid level with the same-size template to detect the pattern at different real-world scales.
- **Coarse-to-fine search** (e.g., in optical flow, stereo matching) — solve at a coarse level first for a rough estimate, then refine at finer levels; dramatically reduces search cost.
- **Image blending / compositing** — blend corresponding Laplacian pyramid levels of two images and reconstruct, giving seamless transitions.
- **Compression** — the Laplacian pyramid is related to sub-band coding schemes.
- **Texture analysis** — oriented pyramids (Unit III) generalize this idea with multiple orientations per scale.

1.8.4(b) Detailed Walkthrough: Multi-Resolution Image Blending

A classic application (Burt & Adelson, 1983) illustrates why the Laplacian pyramid, rather than the raw image, is the right representation for combining two images seamlessly (e.g., blending a picture of an apple and an orange along a vertical seam):

1. Build Laplacian pyramids $L_0^A, \ldots, L_n^A$ and $L_0^B, \ldots, L_n^B$ for images A and B.
2. Build a Gaussian pyramid of a blending **mask** M (e.g., 1 on the left half, 0 on the right half, smoothly transitioning), giving a smoothed mask M_l at every pyramid level.
3. At each level l, blend the detail images using the level-appropriate mask:

$$L_l^{blend} = M_l \cdot L_l^A + (1 - M_l) \cdot L_l^B$$

4. Reconstruct the final image from the blended Laplacian pyramid L^{blend} by the usual reconstruction recursion (§1.8.3).

Why this avoids visible seams. Blending directly in the pixel domain (simply copying pixels from A on one side of a hard boundary and from B on the other) produces a sharp, visible discontinuity, because it does not respect the different characteristic scales of image content: fine texture detail should transition abruptly (over just a few pixels) for a natural look, while coarse, low-frequency shading/illumination should transition very gradually (over a wide region) to avoid a visible "seam" in overall brightness. Blending independently at each pyramid level — with the mask itself smoothed to match each level's spatial scale — automatically achieves exactly this scale-appropriate transition width at every band, which is precisely why Laplacian-pyramid blending produces such convincingly seamless composites compared to naive pixel-domain blending.

1.8.5 Worked Problem 1.6

A Gaussian pyramid is built from a 512×512 image with a downsampling factor of 2 per level. How many levels exist before the image reaches 8×8, and what is the total pixel count of the whole pyramid (including level 0)?

Solution.

$$512 \to 256 \to 128 \to 64 \to 32 \to 16 \to 8$$

That is $512/2^n = 8 \Rightarrow 2^n = 64 \Rightarrow n = 6$. So there are 7 levels total (levels 0 through 6), with level 6 being 8×8.

Total pixels $= \sum_{n=0}^{6} (512/2^n)^2 = 512^2 \sum_{n=0}^{6} \frac{1}{4^n}$

$$\sum_{n=0}^{6} (1/4)^n$$

$$= 1 + 0.25 + 0.0625 + 0.015625 + 0.00390625 + 0.0009765625 + 0.000244140625 \approx 1.3333$$

Total pixels $\approx 262144 \times 1.3333 \approx 349525$ pixels — only about **33% more** than the original image, illustrating the classic result that a Gaussian/Laplacian pyramid costs only $4/3$ the storage of the base image.

1.8.6 Worked Problem 1.7 — Pyramid Search Cost

A template of size 32×32 *must be found somewhere in a* 1024×1024 *image, but its true scale in the image is unknown and could be anywhere from 100% down to 12.5% of its original size. Using an image pyramid with octave (factor-of-2) spacing, how many pyramid levels must be searched, and what is the combined search cost (in pixel-comparisons, ignoring constant*

factors) relative to a single full-resolution search, assuming brute-force template matching costs $O(N^2)$ per level for an $N \times N$ level?

Solution.

Scale range 100% down to $12.5\% = 1/8$, which spans $\log_2(8) = 3$ octaves, so **4 pyramid levels** must be searched (level 0 at full scale, down to level 3 at $1/8$ scale).

Level sizes: 1024^2, 512^2, 256^2, 128^2.

Combined cost $\propto 1024^2 + 512^2 + 256^2 + 128^2 = 1{,}048{,}576 + 262{,}144 + 65{,}536 + 16{,}384 = 1{,}392{,}640$

Relative to a single full-resolution search cost of $1{,}048{,}576$, the total 4-level pyramid search costs about **1.33×** the cost of one full-resolution search — again the same $4/3$-type overhead factor seen in Worked Problem 1.6, confirming that **searching an entire pyramid of scales costs only marginally more than a single-scale search**, making pyramid-based scale search extremely efficient compared to, say, resampling and re-correlating the template itself at many finely-spaced scales.

1.8.6(b) Worked Problem 1.7(b) — Coarse-to-Fine Optical Flow Motivation

A motion-estimation algorithm must search for the displacement of a patch between two frames, over a maximum possible displacement of ±64 pixels in each direction. Using a 4-level Gaussian pyramid (levels 0–3, each subsequent level at half resolution), estimate the maximum displacement that must be searched at the coarsest level, and explain why this dramatically reduces total search cost compared to searching ±64 pixels directly at full resolution.

Solution.

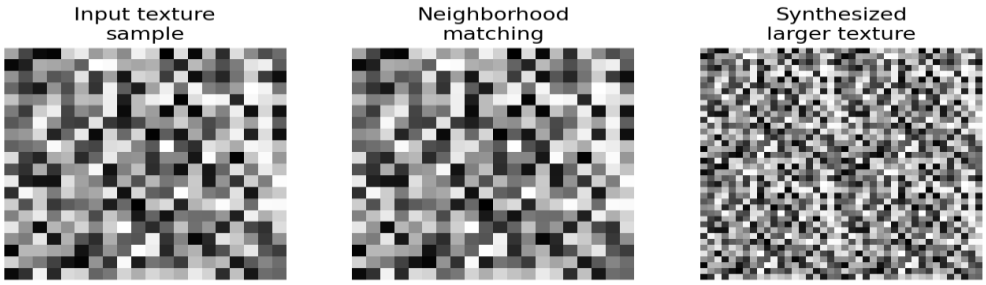
At pyramid level 3 (after 3 halvings, i.e., $2^3 = 8\times$ downsampling), a full-resolution displacement of ± 64 pixels corresponds to only $\pm 64/8 = \pm 8$ pixels at the coarse level. A coarse-level search over a ± 8 pixel range is far cheaper than a full-resolution ± 64 pixel search — for a simple 2-D block-matching search, cost scales roughly as the *square* of the search radius, so this represents a $(64/8)^2 = 64\times$ reduction in search cost at the coarsest level alone.

The standard **coarse-to-fine strategy**: estimate a rough displacement at the coarsest level (cheap, small search range), then propagate this estimate to the next finer level (doubling it, since displacements scale with resolution) and refine it with only a small local search (e.g., ±2 pixels) around the propagated estimate — repeating this refinement down to full resolution. This avoids ever performing a large-radius search at full image resolution, which is the single largest cost driver in motion estimation, and is precisely why coarse-to-fine, pyramid-based search is standard in optical flow, stereo correspondence, and image registration algorithms.

1.8.7 Bridge to Unit II: Multi-Scale Derivative Filters

The scale-space idea introduced here through image pyramids reappears, in a more refined form, in Unit II: rather than only smoothing at each pyramid level, we can compute **derivative-of-Gaussian filters at multiple values of** σ directly (without necessarily subsampling), giving a

continuous scale-space of edge and feature responses. The key formulas — the Gaussian, its first derivative (an edge detector), and its second derivative (the Laplacian of Gaussian, a blob/zero-crossing detector) — are previewed below and developed fully in Unit II.



Left: Gaussian G(x). Center: its first derivative G'(x), an edge-detecting filter. Right: its second derivative G''(x), the Laplacian of Gaussian used for zero-crossing edge detection (Unit II).

$$G_\sigma(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-x^2/2\sigma^2}, \qquad G'_\sigma(x) = -\frac{x}{\sigma^2} G_\sigma(x), \qquad G''_\sigma(x) = \left(\frac{x^2}{\sigma^4} - \frac{1}{\sigma^2}\right) G_\sigma(x)$$

Just as a Gaussian pyramid gives a discrete set of smoothing scales, this family of filters (parametrized continuously by σ) gives a discrete or continuous **scale-space of derivative filters**, unifying the "scale" concept of §1.8 with the "differentiation for edge detection" concept developed next in Unit II.

1.9 Unit I Summary

- Linear, shift-invariant systems are completely described by convolution with their impulse response.
- The Fourier transform reveals filtering as multiplication in the frequency domain (convolution theorem); this explains low-pass, high-pass, and band-pass filter behavior.
- The sampling theorem sets the Nyquist limit; violating it causes aliasing, motivating pre-filtering before subsampling.
- Filters can be viewed as templates; normalized cross-correlation gives a contrast/brightness-invariant matching score, and matched-filter theory shows this is the provably optimal linear detector under additive white noise.
- Image pyramids (Gaussian and Laplacian) give an efficient multi-scale representation used throughout vision for scale-invariant search, coarse-to-fine algorithms, and blending.

1.9.1 Key Formulas at a Glance

Concept	Formula
2-D convolution	$(I * h)(x,y) = \sum_{i,j} h(i,j) I(x-i, y-j)$
Gaussian kernel	$G_\sigma(x,y) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2}$
Convolution theorem	$f * h \Leftrightarrow F(u,v)H(u,v)$

Concept	Formula
Nyquist rate	$f_s \geq 2u_{\max}$
Output noise variance	$\sigma_{out}^2 = \sigma_n^2 \sum h(i,j)^2$
Normalized cross-correlation	$NCC = \frac{\sum (I - \bar{I})(t - \bar{t})}{\sqrt{\sum (I - \bar{I})^2} \sqrt{\sum (t - \bar{t})^2}}$
Gaussian pyramid	$G_l = \downarrow_2 (G_{l-1} * G_\sigma)$
Laplacian pyramid	$L_l = G_l - \text{Expand}(G_{l+1})$
Kernel size rule	$k = 2\lceil 3\sigma \rceil + 1$

1.9.2 How Unit I Connects to Later Units

This unit's concept	Where it is used next
Derivative-of-Gaussian filter preview (§1.8.7)	Full edge-detection theory, Unit II
Gaussian's separability and isotropy	Justification for Gaussian smoothing before differentiation, Unit II
Oriented/rotated template banks (§1.6.3)	Oriented filter banks for texture, Unit III
Laplacian pyramid	Oriented pyramids for texture analysis, Unit III
Normalized correlation / template matching	Finding objects by voting on template relations, Unit V
Scale-space via image pyramids	Multi-scale segmentation and feature detection, Units IV–V

1.10 Review Questions

1. Prove that convolution is commutative and associative.
2. Derive the impulse response of a cascade of two box filters of widths 3 and 5.
3. State and explain the convolution theorem with an example filter.
4. Why must an anti-aliasing filter be applied before down-sampling an image? Illustrate with the Nyquist rate calculation for a sensor of your choice.
5. Derive the normalized cross-correlation formula and explain why it is invariant to affine brightness changes.
6. Explain, with a diagram, how a Laplacian pyramid enables exact image reconstruction.
7. A Gaussian filter has $\sigma = 2$ pixels. Sketch (qualitatively) its frequency response and explain the effect of doubling σ.
8. Compute (by hand) the 2-D convolution of a 4×4 image with a 3×3 averaging kernel at two chosen locations, using zero-padding at the border.
9. Explain separable filtering and derive the computational savings for an $N \times N$ image and $k \times k$ separable kernel.
10. Discuss two real-world applications where scale-space / image pyramids are essential, and explain why a single-scale approach would fail.

11. Derive the matched-filter result that correlation with a known template maximizes output SNR under additive white noise, and explain its relevance to normalized cross-correlation.
12. Explain the difference between true convolution and correlation, and describe a scenario in which the distinction changes the numerical result of a filtering operation.
13. Using the 3σ truncation rule, determine the minimum kernel size for a Gaussian filter with $\sigma = 3.5$ pixels, and explain the consequence of using a smaller kernel than recommended.
14. Derive the unsharp-masking sharpening kernel from a given 3×3 approximate Gaussian smoothing kernel, for a general sharpening strength α.
15. Explain the coarse-to-fine strategy for motion/displacement search using an image pyramid, and estimate the computational savings for a given search radius and number of pyramid levels of your choice.
16. Describe, step by step, the multi-resolution (Laplacian pyramid) image blending algorithm, and explain why blending directly in the pixel domain produces visible seams while pyramid-based blending does not.

UNIT – II

EDGE DETECTION

Comprehensive Lecture Notes

(Lecture Hours: 9)

Prescribed Textbook:

David A. Forsyth & Jean Ponce, "Computer Vision — A Modern Approach", PHI, 2003
(Chapter 8: Edge Detection)

Table of Contents

Unit II Syllabus Coverage

This unit corresponds to Unit-II of the course 23CSE364A (Computer Vision), Lecture Hours: 9. The topics prescribed in the syllabus and covered in these notes are listed below.

- Noise — Additive Stationary Gaussian Noise
- Why Finite Differences Respond to Noise
- Estimating Derivatives — Derivative of Gaussian Filters
- Why Smoothing Helps
- Choosing a Smoothing Filter
- Why Smooth with a Gaussian?
- Detecting Edges — Using the Laplacian to Detect Edges
- Gradient-Based Edge Detectors
- Technique: Orientation Representations and Corners

Learning Outcomes

On completing this unit, the student will be able to:

1. Explain the sources and statistical model of noise in digital images.
2. Analyse why simple finite-difference operators amplify noise.
3. Derive and apply Gaussian derivative filters for robust differentiation.
4. Justify the choice of the Gaussian as the preferred smoothing kernel.
5. Apply the Laplacian-of-Gaussian (LoG) operator and interpret zero-crossings.
6. Implement and compare gradient-based edge detectors (Roberts, Prewitt, Sobel, Canny).
7. Represent local orientation and detect corners using structure-tensor (Harris) analysis.

2.1 Introduction to Edge Detection

An edge in an image is a place where the intensity function changes rapidly, usually because it corresponds to a physical discontinuity in the scene — an object boundary, a change in surface orientation, a change in albedo/reflectance, or a shadow boundary. Edge detection is the process of identifying and locating these sharp discontinuities so that the amount of data to be processed is reduced while the important structural information of the image is preserved.

Formally, if I(x, y) denotes the image intensity function, an edge is a location where the gradient magnitude $|\nabla I|$ attains a local maximum along the gradient direction, or equivalently, a location where the second derivative (Laplacian) crosses zero.

Why Edge Detection Matters

- It is the first step in most higher-level vision tasks: segmentation, object recognition, stereo correspondence, motion estimation, and shape-from-X techniques.

- It converts a 2-D array of intensities into a compact set of curves that summarise the geometric and photometric content of the scene.
- Human vision itself appears to rely heavily on edge- and contour-like representations, which motivates edge-based computational models.

Sources of Intensity Discontinuities

- Depth discontinuities — object boundaries / occlusion edges
- Surface orientation discontinuities — creases, corners of polyhedra
- Reflectance discontinuities — change of material or albedo
- Illumination discontinuities — shadow boundaries, specularities

Because real images are always corrupted by noise arising from the sensor and quantisation, the central theoretical challenge of this unit is: how do we estimate derivatives of the (unknown, noisy) image reliably, so that true edges are detected and spurious ones are suppressed?

2.2 Noise — Additive Stationary Gaussian Noise

Every image acquisition process (CCD/CMOS sensing, quantisation, transmission) introduces noise. The simplest and most widely used mathematical model for this noise is additive stationary Gaussian noise.

2.2.1 The Noise Model

Let I(x, y) be the true (noise-free) image and let n(x, y) be a random noise process. The observed, measured image J(x, y) is modelled as:

$$J(x,\ y)\ =\ I(x,\ y)\ +\ n(x,\ y)$$

Eq. (2.1) — Additive noise model

The noise n(x, y) is assumed to be:

- Additive — it is simply added to the true signal, independent of the signal value.
- Stationary — its statistical properties (mean, variance) do not depend on position (x, y); the noise looks statistically the same everywhere in the image.
- Gaussian — at each pixel, n(x, y) is a random variable drawn from a normal (Gaussian) distribution with zero mean and variance σ_n^2:

$$n(x,\ y)\ \sim\ N(0,\ \sigma^2),\qquad p(n)\ =\ \left(1\ /\ \left(\sigma\sqrt{(2\pi)}\right)\right)\ \cdot\ exp(\ -n^2\ /\ (2\sigma^2)\ )$$

Eq. (2.2) — Gaussian probability density of the noise

- Independent and identically distributed (i.i.d.) — the noise value at one pixel is statistically independent of the noise value at any other pixel.

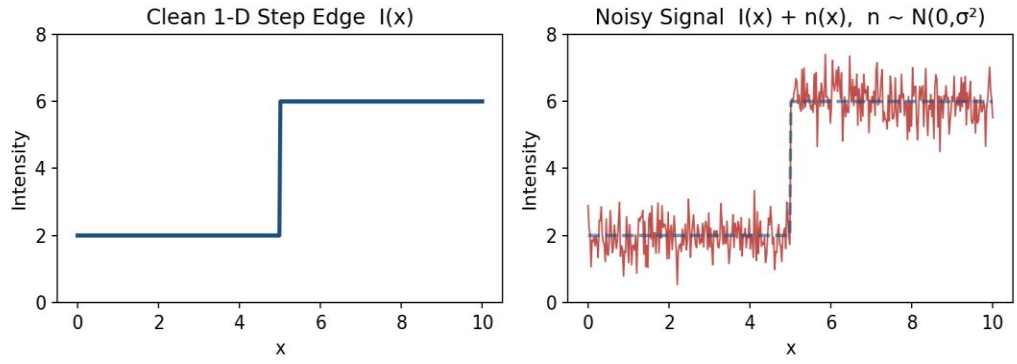


Fig. 2.1 — A clean 1-D step-edge signal (left) and the same signal corrupted by additive stationary Gaussian noise (right).

2.2.2 Why the Gaussian Model is Convenient

- The Central Limit Theorem justifies it: sensor noise is the sum of many small, independent random effects (thermal noise, shot noise, amplifier noise), and the sum of many independent random variables tends toward a Gaussian distribution.
- It is fully described by only two parameters — mean (assumed 0) and variance σ_n^2 — making it analytically tractable.
- Linear operations (filtering) on Gaussian noise produce Gaussian output, which keeps the analysis of filters mathematically closed-form.

2.2.3 Effect of Noise on Derivative Estimation

Because edge detectors rely on estimating derivatives of the image, and differentiation is a high-pass operation, noise — which has energy spread across all spatial frequencies — is amplified disproportionately by differentiation relative to the (usually band-limited) true signal. This motivates the discussion in the next section.

2.3 Why Finite Differences Respond to Noise

The simplest way to estimate the derivative of a discrete image is to use a finite-difference approximation. This section shows, mathematically, why such simple operators are extremely sensitive to noise.

2.3.1 Finite Difference Approximation

For a continuous function I(x), the derivative is defined as the limit:

$$I'(x) = lim(h \to 0) \, [\, I(x+h) - I(x) \,] \, / \, h$$

Eq. (2.3) — Definition of the derivative

For discrete images, h is fixed at the pixel spacing (h = 1), giving the forward difference, backward difference, and central difference operators:

$$Forward: \quad I'(x) \approx I(x+1) - I(x)$$

$$Backward: \quad I'(x) \approx I(x) - I(x-1)$$

$$Central: \quad I'(x) \approx [\, I(x+1) - I(x-1) \,] / 2$$

Eq. (2.4) — Finite difference operators

These correspond to convolving the image with the small kernels [−1, 1], [1, −1] and [−1, 0, 1]/2 respectively.

2.3.2 Noise Amplification — Mathematical Analysis

Let the observed signal be J(x) = I(x) + n(x), where n(x) is zero-mean noise with variance σ_n^2. Applying the central-difference operator:

$$J'(x) = I'(x) + [\, n(x+1) - n(x-1) \,] / 2$$

Eq. (2.5)

Since n(x+1) and n(x−1) are independent, zero-mean random variables each with variance σ_n^2, the variance of the derivative-noise term is:

$$Var\{\, [n(x+1) - n(x-1)] / 2 \,\} = (\, \sigma_n^2 + \sigma_n^2 \,) / 4 = \sigma_n^2 / 2$$

Eq. (2.6)

More generally, for a difference filter with kernel coefficients $c_1, c_2, \ldots, c_k$ ($\Sigma c_i = 0$, since a derivative filter must reject the DC/constant component), the output noise variance is:

$$\sigma\_out^2 = \sigma_n^2 \cdot \Sigma\, c_i^2$$

Eq. (2.7) — Noise variance after linear filtering

The key insight is this: differentiation is a high-pass filtering operation. Random noise has power spread uniformly across all spatial frequencies (it is approximately "white"), whereas the useful edge signal is typically concentrated at lower and mid frequencies. A difference operator boosts high frequencies, and therefore boosts the noise power far more than it boosts the (smoother)

true-signal power. Consequently, computing derivatives directly on raw, noisy image data produces a derivative image that is dominated by noise rather than by true edges.

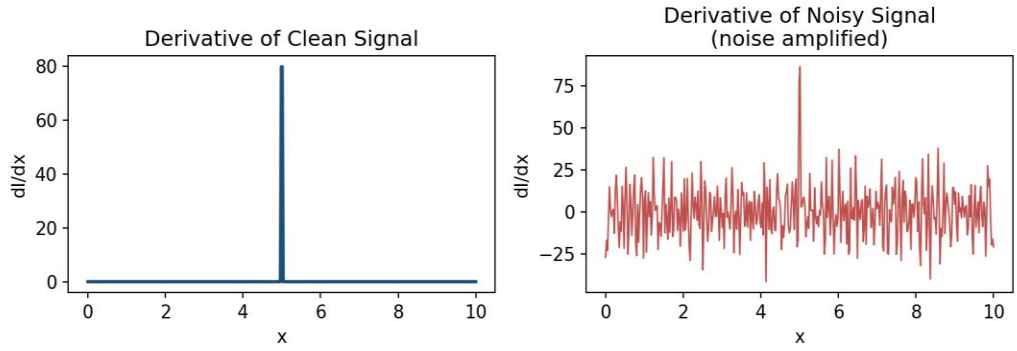


Fig. 2.2 — Differentiating a clean signal gives a clean spike at the edge (left); differentiating the same signal after it is corrupted by noise produces a derivative that is swamped by high-frequency noise (right).

2.3.3 Consequence

Because raw finite differencing is unusable on real (noisy) images, practical edge detectors never differentiate the raw image directly. Instead they combine differentiation with smoothing — this combined operation is the subject of the next two sections.

2.4 Estimating Derivatives — Derivative of Gaussian Filters

The standard solution to the noise-amplification problem is to smooth the image before (or while) differentiating it. Because both smoothing and differentiation are linear operations, they can be combined into a single filter by exploiting the associativity/commutativity of convolution and differentiation.

2.4.1 Combining Smoothing and Differentiation

Let G(x, σ) denote the 1-D Gaussian smoothing kernel with standard deviation σ:

$$G(x,\ \sigma)\ =\ 1\ /\ \left(\sigma\sqrt{(2\pi)}\right)\ \cdot\ exp(\ -x^2\ /\ (2\sigma^2)\ )$$

Eq. (2.8) — 1-D Gaussian kernel

Suppose we first smooth the noisy signal J(x) with G(x, σ), then differentiate the result. Because convolution and differentiation commute:

$$d/dx\ [\ J(x)\ *\ G(x,\ \sigma)\ ]\ \ =\ \ J(x)\ *\ [\ dG(x,\ \sigma)/dx\ ]$$

Eq. (2.9) — Derivative-of-Gaussian identity

This is one of the most important identities in edge detection: instead of smoothing and then differentiating in two passes, we can pre-compute the derivative of the Gaussian kernel itself and

convolve the noisy image with it in a single pass. This combined kernel is called the Derivative-of-Gaussian (DoG) filter.

2.4.2 The First Derivative of the Gaussian

$$G'(x,\ \sigma)\ =\ dG/dx\ =\ -(x\ /\ \sigma^2)\ \cdot\ G(x,\ \sigma)\ =\ -x\ /\ \left(\sigma^3\sqrt{(2\pi)}\right)\ \cdot\ exp(\ -x^2\ /\ (2\sigma^2)\ )$$

Eq. (2.10) — First derivative of Gaussian

2.4.3 The Second Derivative of the Gaussian

$$G''(x,\ \sigma)\ =\ (x^2\ -\ \sigma^2)\ /\ \sigma^4\ \cdot\ G(x,\ \sigma)$$

Eq. (2.11) — Second derivative of Gaussian

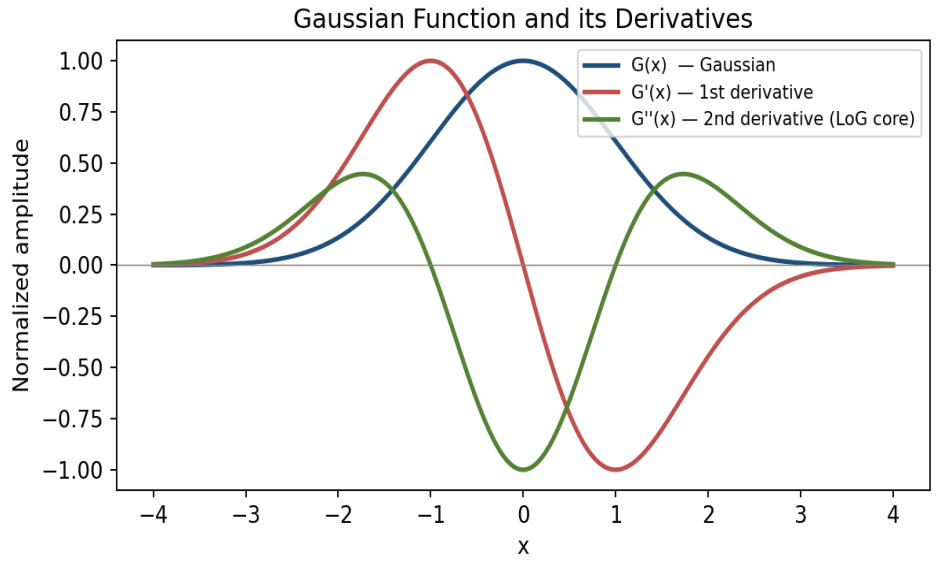


Fig. 2.3 — The Gaussian G(x), its first derivative G'(x) (used for gradient/edge detectors), and its second derivative G''(x) (the core of the Laplacian-of-Gaussian operator).

2.4.4 2-D Extension

For a 2-D image I(x, y), the 2-D Gaussian is separable, G(x, y, σ) = G(x, σ)·G(y, σ), and the partial derivatives used to build the gradient are:

$$\partial/\ \partial x\ [\ I\ *\ G\ ]\ =\ I\ *\ \partial G/\ \partial x\ ,\qquad \partial/\ \partial y\ [\ I\ *\ G\ ]\ =\ I\ *\ \partial G/\ \partial y$$

Eq. (2.12) — 2-D derivative-of-Gaussian filters

Because the Gaussian is separable, these 2-D convolutions can each be computed as two 1-D convolutions (one along rows, one along columns), which is far more efficient than direct 2-D convolution.

2.4.5 Properties of the DoG Filter

- It performs smoothing and differentiation simultaneously in one linear convolution — computationally efficient.
- The scale parameter σ controls the trade-off between noise suppression and localisation accuracy of the detected edge (see Section 2.5).
- Because the Gaussian and its derivatives decay smoothly to zero, the filter has no ringing artefacts, unlike an ideal low-pass filter.

2.5 Why Smoothing Helps

Smoothing (low-pass filtering) suppresses high spatial-frequency content. Since noise is broadband (spread across all frequencies) while the true edge signal is concentrated at lower frequencies, smoothing reduces the noise power far more than it reduces the useful signal power, improving the signal-to-noise ratio (SNR) before differentiation is applied.

2.5.1 Quantitative Argument

Consider smoothing the noisy image $J = I + n$ with a filter having coefficients w_i ($\Sigma w_i = 1$, so that a constant region is unaffected). The noise variance after smoothing is:

$$\sigma\_smoothed^2 = \sigma^2 \cdot \Sigma\, w_i^2$$

Eq. (2.13)

Since $\Sigma w_i = 1$ and there are N non-zero taps, $\Sigma w_i^2 \leq 1$ with equality only for a single-tap (no smoothing) filter. As the number of averaging taps N grows (wider smoothing kernel), $\Sigma w_i^2 \rightarrow 1/N$, so the noise variance is reduced roughly in proportion to 1/N — i.e. averaging over more pixels reduces noise standard deviation by a factor of $\sqrt{N}$.

2.5.2 The Trade-off: Smoothing vs Localisation

Smoothing is not "free": a larger smoothing scale σ:

- Reduces noise sensitivity and produces fewer false edges — improves detection.
- But blurs the location of true edges and can merge nearby edges together — worsens localisation.
- Can also completely suppress fine texture and weak/small edges that are genuine features.

This is often summarised as the fundamental trade-off in edge detection between good detection (few missed edges / false positives) and good localisation (precise position of the detected edge) — a trade-off formalised in Canny's criteria for an "optimal" edge detector (see Section 2.9).

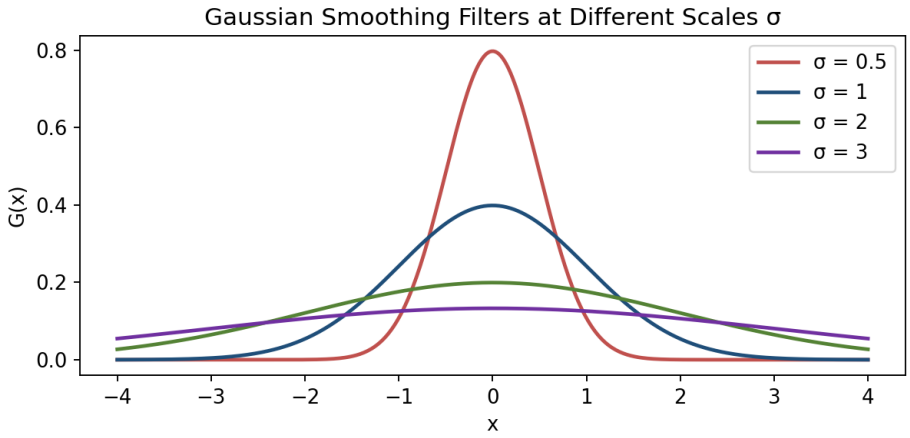


Fig. 2.4 — Gaussian smoothing kernels at increasing scale σ. Larger σ averages over a wider neighbourhood, suppressing more noise but blurring fine detail.

2.6 Choosing a Smoothing Filter

Several smoothing kernels could, in principle, be used before differentiation. The desirable properties of a good smoothing filter for edge detection are listed below, followed by a comparison of common choices.

2.6.1 Desirable Properties

1. Effective noise suppression — should behave as a good low-pass filter.
2. No new features introduced — the filter's own oscillations should not create spurious zero-crossings or extra edges (no ringing).
3. Compact support in space — for computational efficiency.
4. Smooth (differentiable) — so that it can be differentiated analytically to build the derivative-of-Gaussian filters described in Section 2.4.
5. Rotational symmetry (isotropy) in 2-D — smoothing should not bias the estimate toward any particular orientation.

2.6.2 Common Candidate Filters

Filter	Frequency Response	Ringing?	Suitability
Box (mean) filter	sinc-shaped, poor roll-off	Yes — significant side-lobes	Poor: introduces ringing / spurious edges
Ideal low-pass	Perfect cut-off in	Yes — infinite spatial	Poor: not realisable,

Filter	Frequency Response	Ringing?	Suitability
(sinc)	frequency	extent, severe ringing	strong ringing
Triangular / Bartlett	Better roll-off than box	Reduced but present	Moderate
Gaussian	Smooth Gaussian roll-off, no side-lobes	No ringing (Gaussian has a Gaussian Fourier transform)	Best overall choice

The comparison shows that the box filter and the ideal low-pass filter, despite being computationally simple or theoretically "perfect" in the frequency domain, both introduce ringing artefacts (extra oscillations near sharp edges) because of their sharp cut-offs or discontinuities. The Gaussian filter avoids these problems, which is why it is used almost universally as the smoothing kernel of choice — the detailed justification is given in the next section.

2.7 Why Smooth with a Gaussian?

The Gaussian kernel is the preferred smoothing filter in edge detection for several precise mathematical reasons, summarised below.

2.7.1 Uncertainty-Principle Optimality

Among all possible functions, the Gaussian is the unique function that minimises the product of its spread in the spatial domain (Δx) and its spread in the frequency domain (Δω) — i.e. it achieves the theoretical minimum of the uncertainty relation:

$$\Delta x \cdot \Delta \omega \geq 1/2 \quad (equality\ holds\ only\ for\ the\ Gaussian)$$

Eq. (2.14) — Space–frequency uncertainty principle

This means the Gaussian gives the best possible simultaneous localisation in both space and frequency — it smooths (localises in frequency, suppressing high-frequency noise) while disturbing spatial position as little as mathematically possible.

2.7.2 No Spurious Zero-Crossings / Extrema

A key theoretical result (Babaud, Witkin, Baudin and Duda; Yuille and Poggio) states that as the smoothing scale σ of a Gaussian filter is increased, the number of zero-crossings (or local extrema) in the smoothed signal never increases — new spurious edges/features cannot be created by additional Gaussian smoothing. This makes the Gaussian scale-space causal: coarse-scale features can always be traced back to fine-scale ones, giving a well-behaved multi-scale (scale-space) representation of the image. Filters other than the Gaussian do not, in general, have this property and can create new, spurious extrema as σ increases.

2.7.3 Separability

$$G(x, y, \sigma) = G(x, \sigma) \cdot G(y, \sigma)$$

Eq. (2.15) — 2-D Gaussian is separable into 1-D Gaussians

This allows an expensive 2-D convolution ($O(N^2)$ per pixel for an N×N kernel) to be replaced by two cheap 1-D convolutions (O(2N) per pixel), giving a large computational speed-up.

2.7.4 Self-Similarity Under Convolution

The convolution of two Gaussians is itself a Gaussian, with variances adding:

$$G(x, \sigma_1) * G(x, \sigma_2) = G\left(x, \sqrt{({\sigma_1}^2 + {\sigma_2}^2)}\right)$$

Eq. (2.16)

This property allows smoothing to be applied incrementally / cascaded efficiently, and underlies the construction of Gaussian and Laplacian image pyramids.

2.7.5 Rotational Symmetry (Isotropy)

The 2-D Gaussian is circularly symmetric, $G(x, y, \sigma) = G(\sqrt{(x^2+y^2)}, \sigma)$. Smoothing therefore treats every orientation identically, avoiding any directional bias in the subsequent gradient/edge estimate.

2.7.6 Summary of Reasons

- Optimal joint space–frequency localisation (uncertainty principle).
- Guarantees no new zero-crossings are created — well-behaved scale-space.
- Separable → computationally efficient.
- Self-similar under convolution → smooth multi-scale cascading.
- Rotationally symmetric → no directional bias.
- Infinitely differentiable → analytic derivative-of-Gaussian filters can be pre-computed (Section 2.4).

2.8 Detecting Edges — Using the Laplacian to Detect Edges

An alternative to searching for maxima of the first derivative (gradient) is to search for zero-crossings of the second derivative. This is the basis of the Laplacian and Laplacian-of-Gaussian (LoG / "Marr–Hildreth") edge detector.

2.8.1 The Laplacian Operator

The Laplacian of a 2-D function I(x, y) is the sum of its unmixed second partial derivatives:

$$\nabla^2 I = \partial^2 I / \partial x^2 + \partial^2 I / \partial y^2$$

Eq. (2.17) — Laplacian operator

The Laplacian is a linear, isotropic (rotation-invariant) operator — it responds equally to edges of any orientation, which is an advantage over directional gradient operators when only edge presence (not orientation) is required.

2.8.2 Zero-Crossings Mark Edge Locations

Consider a 1-D step edge that has been blurred (as every real edge is, by the imaging system and by any pre-smoothing). Its intensity profile I(x) is monotonic through the transition. Its first derivative I'(x) has a single maximum exactly at the centre of the transition, and its second derivative I''(x) crosses zero at exactly that same location — changing sign from positive to negative (or vice-versa).

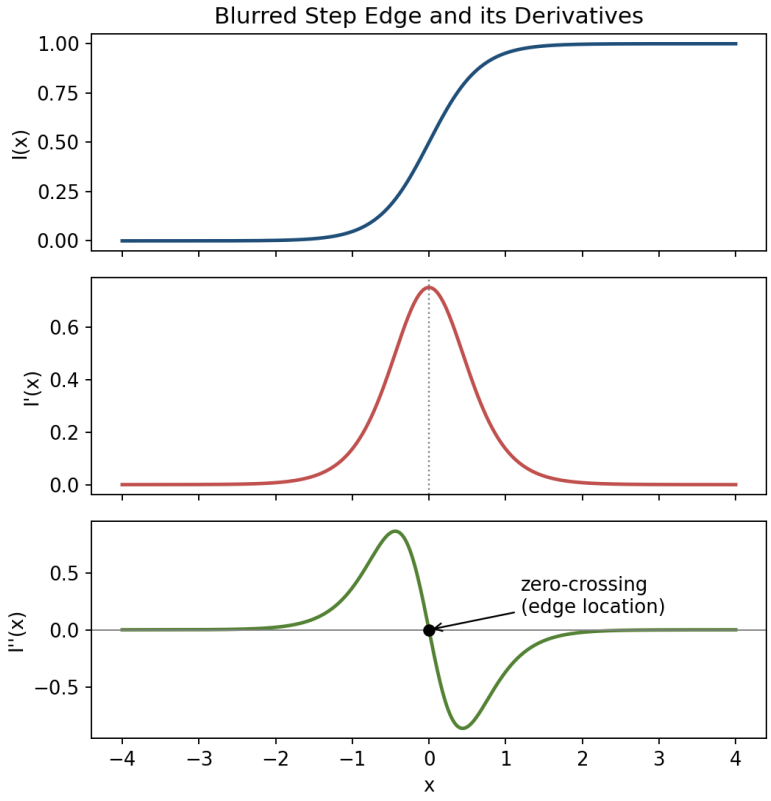


Fig. 2.5 — A blurred step edge I(x) (top), its first derivative I'(x) which peaks at the edge (middle), and its second derivative I''(x) which crosses zero exactly at the edge location (bottom).

This gives the zero-crossing criterion for edge detection:

$$Edge\ at\ x_0 \quad \Leftrightarrow \quad \nabla^2 I(x_0) = 0 \ \ and \ \ \nabla^2 I \ changes \ sign \ across \ x_0$$

Eq. (2.18) — Zero-crossing edge criterion

2.8.3 The Laplacian-of-Gaussian (LoG) Operator

Since the Laplacian is a second-derivative (high-pass) operator, applying it directly to a noisy image is even more noise-sensitive than first-derivative operators (Section 2.3). The image is therefore smoothed with a Gaussian before the Laplacian is applied. As with the first derivative (Section 2.4), linearity lets us pre-combine the two operations into a single kernel:

$$\nabla^2(I * G(\sigma)) = I * \nabla^2 G(\sigma) = I * LoG(x, y, \sigma)$$

Eq. (2.19) — Laplacian-of-Gaussian identity

The 2-D LoG kernel has the closed form (with $r^2 = x^2 + y^2$):

$$LoG(x, y, \sigma) = -1/(\pi\sigma^4) \cdot [\, 1 - r^2/(2\sigma^2) \,] \cdot exp(-r^2 / (2\sigma^2))$$

Eq. (2.20) — Laplacian-of-Gaussian kernel ("Mexican hat")

Because of its characteristic profile (a central positive peak surrounded by a negative annulus, or vice-versa) this kernel is popularly called the "Mexican-hat" operator.

2.8.4 The Marr–Hildreth Edge-Detection Algorithm

1. Convolve the image with the LoG(x, y, σ) kernel (equivalently, Gaussian-smooth then apply the Laplacian).
2. Locate the zero-crossings of the filtered image — pixels where the sign changes between neighbours, above a chosen gradient-magnitude threshold to reject weak (noise-driven) crossings.
3. Mark these zero-crossing locations as edge pixels.

2.8.5 Advantages and Disadvantages of the LoG Approach

- Advantage: isotropic — a single (rotation-invariant) filter detects edges of every orientation.
- Advantage: gives closed edge contours more readily since zero-crossings form continuous curves.
- Disadvantage: at large σ, edges from different nearby features can merge or their exact position can shift (localisation error grows with σ).
- Disadvantage: produces some zero-crossings that do not correspond to significant edges (need for gradient-magnitude thresholding at the zero-crossing).
- Disadvantage: no explicit edge-orientation information is produced directly (unlike gradient methods).

2.9 Gradient-Based Edge Detectors

Gradient-based methods locate edges by finding local maxima of the first-derivative (gradient) magnitude of the (smoothed) image, in the direction of the gradient.

2.9.1 The Image Gradient

For a 2-D image I(x, y), the gradient is the vector of partial derivatives:

$$\nabla I = (\partial I/\partial x,\ \partial I/\partial y) = (Gx,\ Gy)$$

Eq. (2.21) — Image gradient vector

The gradient magnitude and direction are:

$$|\nabla I| = \sqrt{(Gx^2 + Gy^2)} \qquad (often\ approximated\ as\ |Gx| + |Gy|\ for\ speed)$$

Eq. (2.22) — Gradient magnitude

$$\theta(x, y) = tan^{-1}(Gy / Gx)$$

Eq. (2.23) — Gradient (edge-normal) direction

The gradient vector points in the direction of steepest increase of intensity, i.e. perpendicular to the local edge orientation; the edge itself runs tangent to (perpendicular to) the gradient direction.

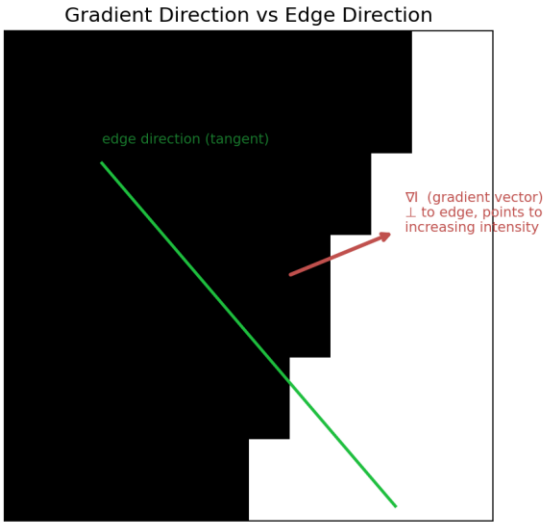


Fig. 2.6 — At an edge, the gradient vector ∇I is perpendicular to the local edge (tangent) direction and points toward increasing intensity.

2.9.2 Roberts Cross Operator

One of the earliest gradient operators, using 2×2 diagonal kernels:

$$Gx = [[1, 0], [0, -1]], \quad Gy = [[0, 1], [-1, 0]]$$

Eq. (2.24) — Roberts cross kernels

Simple and fast, but highly sensitive to noise because it uses only a 2×2 neighbourhood with no smoothing.

2.9.3 Prewitt and Sobel Operators

Prewitt and Sobel operators approximate the derivative-of-Gaussian idea using small integer masks that combine differencing (in one direction) with simple averaging/smoothing (in the perpendicular direction), giving much better noise immunity than Roberts.

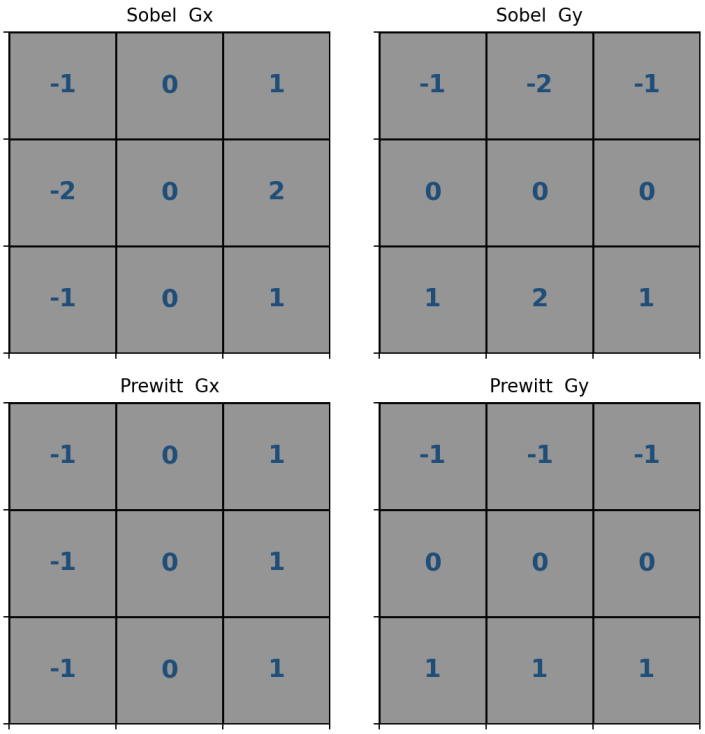


Fig. 2.7 — 3×3 Sobel and Prewitt kernels for the horizontal (Gx) and vertical (Gy) gradient components. Sobel weights the central row/column by 2, giving mild additional smoothing.

The Sobel gradient magnitude and direction are computed from the two convolution outputs exactly as in Eq. (2.22)–(2.23), with Gx and Gy obtained by convolving the image with the Sobel Gx and Gy masks respectively.

2.9.4 The Canny Edge Detector

John Canny (1986) formalised three optimality criteria for an edge detector, and derived an operator that (approximately) satisfies them:

1. Good detection — the detector should respond to true edges and not respond to non-edges (maximise SNR of the response, minimise false positives / false negatives).
2. Good localisation — the detected edge points should be as close as possible to the true edge location (minimise positional error).
3. Single response — the detector should return only one point per true edge, not multiple responses to the same edge (minimise the number of local maxima around the true edge).

Canny showed that the first derivative of a Gaussian gives a good practical approximation to the operator that jointly optimises these three (partly conflicting) criteria. The complete Canny algorithm pipeline is:

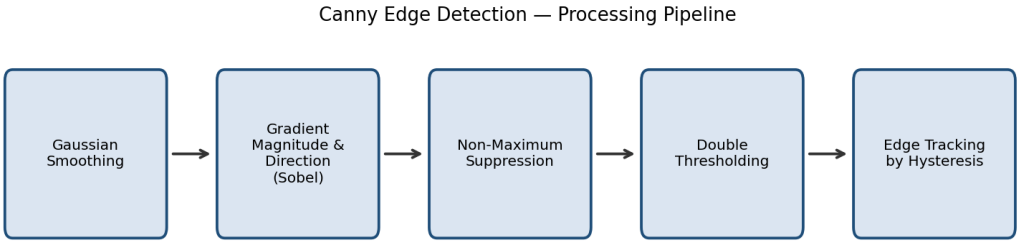


Fig. 2.8 — The five stages of the Canny edge-detection pipeline.

Stage 1 — Gaussian Smoothing

Convolve the input image with a Gaussian G(x, y, σ) to suppress noise; σ is chosen according to the desired detection/localisation trade-off (Section 2.5).

Stage 2 — Gradient Computation

Compute Gx, Gy (typically with Sobel masks) on the smoothed image, and derive magnitude $|\nabla I|$ and direction θ using Eq. (2.22)–(2.23).

Stage 3 — Non-Maximum Suppression (NMS)

At each pixel, the gradient direction θ is quantised to one of four principal directions (0°, 45°, 90°, 135°). The pixel's gradient magnitude is compared with the magnitudes of its two neighbours along that direction; the pixel is retained only if its magnitude is a local maximum, otherwise it is suppressed (set to zero). This thins wide edge responses down to single-pixel-wide ridges.

Stage 4 — Double Thresholding

Two thresholds, a high threshold T_h and a low threshold T_l (with T_h typically 2–3× T_l), are applied to the suppressed gradient magnitude image:

- Magnitude > T_h → classified as a strong (definite) edge pixel.
- T_l < Magnitude ≤ T_h → classified as a weak (candidate) edge pixel.

- Magnitude ≤ T_l → suppressed (not an edge).

Stage 5 — Edge Tracking by Hysteresis

A weak edge pixel is retained (promoted to a final edge) only if it is 8-connected to at least one strong edge pixel (directly or via a chain of other weak pixels); otherwise it is discarded as noise. This hysteresis mechanism produces continuous, thin edge contours while suppressing isolated noise responses.

2.9.5 Comparison of Gradient-Based Detectors

Detector	Kernel Size	Noise Immunity	Localisation	Remarks
Roberts	2×2	Poor	Good (small support)	Very fast, obsolete for noisy images
Prewitt	3×3	Moderate	Moderate	Uniform averaging in perpendicular direction
Sobel	3×3	Moderate–Good	Moderate	Weighted averaging gives slightly better smoothing than Prewitt
Laplacian of Gaussian	Depends on σ	Good (tunable via σ)	Degrades as σ increases	Isotropic; zero-crossing based
Canny	Depends on σ	Very good	Very good (satisfies 3 optimality criteria)	Industry standard; adds NMS + hysteresis

2.10 Technique: Orientation Representations and Corners

2.10.1 Representing Local Orientation

At every pixel, the gradient direction θ(x, y) from Eq. (2.23) gives a local estimate of edge orientation. However, orientation is inherently a mod-180° (not mod-360°) quantity for edges, since an edge running at angle θ is indistinguishable from one running at θ + 180° — only the perpendicular gradient direction (which does span the full 360°, since it also encodes the sign / direction of the intensity change) is a full-circle quantity.

A convenient way to represent orientation that respects this ambiguity is the double-angle representation:

$$v(x,\ y)\ =\ (\ cos\ 2\theta\ ,\ \ sin\ 2\theta\ )$$

Eq. (2.25) — Double-angle orientation vector

This maps the 180°-periodic orientation onto a full 360° vector, so that orientations differing by 180° map to the same vector, and averaging orientation vectors (e.g. over a neighbourhood, to get a smoothed orientation field) becomes a simple, well-defined vector average.

A common way to obtain a robust orientation estimate at a pixel — used as the basis of the structure tensor / second-moment matrix below — is to accumulate the outer product of the gradient vector over a small neighbourhood, weighted by a Gaussian window.

2.10.2 Corners and the Structure Tensor

A corner is a point where the local image structure varies in more than one direction — i.e. two (or more) edges meet. Corners are of special interest because, unlike points along a straight edge, they can be localised precisely in both coordinates and are therefore excellent features for matching, tracking, and stereo correspondence.

2.10.2.1 The Auto-Correlation / Second-Moment Matrix

Consider a small window W around a pixel (x, y), and consider shifting that window by a small displacement (Δx, Δy). The sum of squared intensity differences between the original and shifted windows is:

$$E(\Delta x,\ \Delta y)\ =\ \Sigma\_\{(u,v) \in W\}\ w(u,v)\ \cdot\ [\ I(u+\Delta x,\ v+\Delta y)\ -\ I(u,\ v)\ ]^2$$

Eq. (2.26) — Auto-correlation (SSD) function

---

where w(u, v) is a weighting window (often Gaussian). Using a first-order (Taylor series) approximation I(u+Δx, v+Δy) ≈ I(u,v) + Ix·Δx + Iy·Δy, this becomes a quadratic form:

$$E(\Delta x,\ \Delta y)\ \ \approx\ \ [\Delta x\ \ \Delta y]\ \cdot\ M\ \cdot\ [\Delta x\ \ \Delta y]^{\mathrm{T}}$$

Eq. (2.27)

---

where M is the 2×2 structure tensor (second-moment matrix), computed from the smoothed products of image gradients:

$$M\ =\ [\ \Sigma w \cdot Ix^2\quad \Sigma w \cdot Ix \cdot Iy\ ;\ \ \Sigma w \cdot Ix \cdot Iy\quad \Sigma w \cdot Iy^2\ ]$$

Eq. (2.28) — Structure tensor M

2.10.2.2 Eigenvalue Interpretation

Let λ_1 and λ_2 ($\lambda_1 \geq \lambda_2 \geq 0$) be the eigenvalues of M. Because M is a real symmetric matrix, its eigenvalues directly describe how E(Δx, Δy) — and hence local intensity variation — behaves along the two principal (eigenvector) directions:

- $\lambda_1 \approx \lambda_2 \approx 0$ → flat region: intensity is nearly constant in every direction (no structure).
- λ_1 large, $\lambda_2 \approx 0$ → edge: intensity varies strongly in one direction (across the edge) and hardly at all along the other (along the edge).
- λ_1 and λ_2 both large → corner: intensity varies strongly in every direction — this is the signature of a corner.

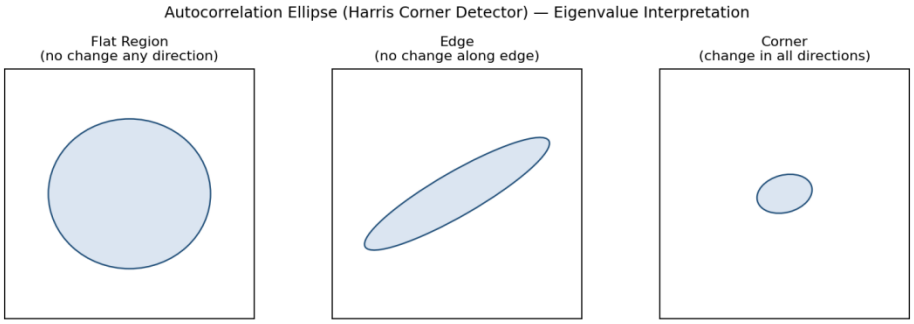


Fig. 2.9 — The auto-correlation function E(Δx,Δy), visualised as an ellipse (level curve), for a flat region, an edge, and a corner respectively. Ellipse shape reflects the eigenvalues of the structure tensor M.

2.10.2.3 The Harris Corner Response

Computing eigenvalues explicitly at every pixel is expensive, so Harris and Stephens (1988) proposed a corner-response function that avoids explicit eigen-decomposition, using the trace and determinant of M (which equal the sum and product of the eigenvalues respectively):

$$R = det(M) - k \cdot [\, trace(M) \,]^2 = (\lambda_1\lambda_2) - k \cdot (\lambda_1 + \lambda_2)^2$$

Eq. (2.29) — Harris corner response ($k \approx 0.04 – 0.06$)

Interpretation of R:

- R large and positive → corner (both eigenvalues large and comparable).
- R large and negative → edge (one eigenvalue much larger than the other).
- |R| small → flat region (both eigenvalues small).

Corners are finally selected as local maxima of R that exceed a chosen threshold, typically followed by non-maximum suppression to retain one detection per physical corner.

2.10.2.4 Properties and Applications of Corner Detection

- Rotation invariant — eigenvalues of M do not change under image rotation.
- Partially invariant to affine changes in illumination and to small viewpoint changes.
- Not scale invariant in its basic form — extensions (e.g. Harris–Laplace, SIFT keypoints) add automatic scale selection.
- Applications: feature-based image matching and stereo correspondence, motion tracking (KLT tracker), 3-D reconstruction and structure-from-motion, panorama stitching, object recognition.

2.11 Worked Examples

Example 1 — Applying the Sobel Operator

Given the 3×3 image neighbourhood I below, compute the horizontal and vertical Sobel gradients and the gradient magnitude at the centre pixel.

$$I = [\,10\ \ 10\ \ 10\,;\ \ 10\ \ 50\ \ 90\,;\ \ 10\ \ 90\ \ 90\,]$$

Given 3×3 neighbourhood

Step 1 — Apply the Sobel Gx mask [[−1,0,1],[−2,0,2],[−1,0,1]] by element-wise multiplication and summation:

$$Gx = (-1\cdot 10 + 0\cdot 10 + 1\cdot 10) + (-2\cdot 10 + 0\cdot 50 + 2\cdot 90) + (-1\cdot 10 + 0\cdot 90 + 1\cdot 90) = 0 + 160 + 80 = 240$$

Step 2 — Apply the Sobel Gy mask [[−1,−2,−1],[0,0,0],[1,2,1]]:

$$Gy = (-1\cdot 10 - 2\cdot 10 - 1\cdot 10) + (0 + 0 + 0) + (1\cdot 10 + 2\cdot 90 + 1\cdot 90) = -40 + 0 + 280 = 240$$

Step 3 — Gradient magnitude and direction:

$$|\nabla I| = \sqrt{(240^2 + 240^2)} \approx 339.4, \qquad \theta = tan^{-1}(240/240) = 45°$$

Result

Interpretation: the large, equal Gx and Gy values indicate a strong edge running diagonally (perpendicular to the 45° gradient direction), consistent with the bright region occupying the bottom-right of the neighbourhood.

Example 2 — Noise Variance After Averaging

An image is corrupted by additive Gaussian noise with σ_n = 20 grey levels. If a 3×3 box averaging filter (all weights = 1/9) is applied, find the standard deviation of the noise in the output.

$$\sigma\_out^2 = \sigma^2 \cdot \Sigma w_i^2 = 400 \cdot (9 \times (1/9)^2) = 400 \times (1/9) = 44.44 \Rightarrow \sigma\_out \approx 6.67$$

Solution

The noise standard deviation is reduced from 20 to about 6.67 (a factor of 3 = √9), confirming the 1/√N noise-reduction rule of Section 2.5.

Example 3 — Choosing σ for a Gaussian Derivative Filter

Qualitatively explain what happens to a detected edge map if σ in a Canny detector is increased from 1 to 5 pixels.

- Noise sensitivity decreases — fewer spurious edge fragments from sensor noise.
- Edge localisation degrades — the exact edge position may shift by an amount proportional to σ.
- Weak, fine-scale edges (e.g. texture, thin lines) may be smoothed away and missed entirely.
- Nearby parallel edges may merge into a single detected edge, since they are no longer resolved after wide smoothing.

Example 4 — Harris Response Classification

At a pixel, the structure tensor M has eigenvalues $\lambda_1 = 8000$ and $\lambda_2 = 7500$. Using $k = 0.05$, compute the Harris response R and classify the pixel.

$$R = \lambda_1\lambda_2 - k(\lambda_1 + \lambda_2)^2 = (8000 \times 7500) - 0.05 \times (15500)^2$$
$$= 6.0 \times 10^7 - 1.20 \times 10^7 = 4.80 \times 10^7$$

Solution

Since R is large and positive, and $\lambda_1 \approx \lambda_2$ (both large), the pixel is classified as a corner.

2.12 Unit Summary

This unit developed the theory of edge detection starting from the statistical model of image noise, through to state-of-the-art gradient- and Laplacian-based detectors and corner detection. The key ideas are summarised below.

- Real images always contain noise, commonly modelled as additive, stationary, zero-mean Gaussian noise $n(x,y) \sim N(0, \sigma_n^2)$.
- Direct finite-difference derivatives amplify noise because differentiation is a high-pass operation, and noise power is spread across all frequencies while edge signal power is concentrated at lower frequencies.
- Smoothing before differentiating improves SNR; the two operations can be fused, using linearity, into a single derivative-of-Gaussian (DoG) filter: $d/dx(I*G) = I*(dG/dx)$.
- The Gaussian is the preferred smoothing kernel because it is optimal under the space–frequency uncertainty principle, never introduces new zero-crossings as σ increases, is separable, self-similar under convolution, and rotationally symmetric.
- The Laplacian-of-Gaussian (LoG) operator detects edges as zero-crossings of the second derivative of the smoothed image; it is isotropic but loses localisation accuracy as σ grows.

- Gradient-based detectors (Roberts, Prewitt, Sobel, Canny) locate edges as local maxima of gradient magnitude. Canny's method — Gaussian smoothing, gradient computation, non-maximum suppression, double thresholding, and hysteresis tracking — remains the benchmark algorithm.
- Local orientation can be represented using a double-angle vector to handle the inherent 180° ambiguity of edge direction.
- Corners are located using the structure tensor (second-moment matrix) M built from smoothed products of image gradients; the Harris response $R = \det(M) - k \cdot \text{trace}(M)^2$ classifies a pixel as flat, edge, or corner without explicit eigen-decomposition.

2.13 Two-Mark Questions

1. Define an edge in the context of digital image processing.
2. What is meant by additive stationary Gaussian noise?
3. Give the mathematical model relating a noisy image J(x,y) to the true image I(x,y) and noise n(x,y).
4. Why does simple finite differencing amplify noise?
5. State the derivative-of-Gaussian identity that combines smoothing and differentiation.
6. Write the formula for the 1-D Gaussian smoothing kernel $G(x, \sigma)$.
7. What is the trade-off involved in choosing the smoothing scale σ?
8. List any two desirable properties of a good smoothing filter for edge detection.
9. State the space–frequency uncertainty principle and its relevance to the Gaussian filter.
10. Define the Laplacian operator $\nabla^2 I$ for a 2-D image.
11. What is a zero-crossing, and how is it used to detect edges?
12. Write the closed-form expression for the Laplacian-of-Gaussian (LoG) kernel.
13. Give the 3×3 Sobel masks for Gx and Gy.
14. Define gradient magnitude and gradient direction of an image.
15. List Canny's three criteria for an optimal edge detector.
16. What is the purpose of non-maximum suppression in the Canny algorithm?
17. Explain briefly the purpose of hysteresis thresholding.
18. What is the structure tensor (second-moment matrix) used for?

19. Write the Harris corner response formula.
20. Differentiate between an edge point and a corner point in terms of eigenvalues of M.

2.14 Five-Mark Questions

1. Explain the additive stationary Gaussian noise model and discuss why it is a reasonable approximation for real image sensors.
2. Derive an expression for the noise variance after applying a linear finite-difference filter, and explain why this shows that differentiation amplifies noise.
3. Derive the derivative-of-Gaussian identity d/dx(I*G) = I*(dG/dx) and explain its practical significance in edge detection.
4. Explain, with the help of a diagram, why smoothing improves the signal-to-noise ratio before differentiation.
5. Compare the box filter, ideal low-pass filter, and Gaussian filter as candidate smoothing kernels for edge detection.
6. Explain any three mathematical reasons why the Gaussian is preferred over other smoothing kernels.
7. Describe the Marr–Hildreth (Laplacian-of-Gaussian) edge detection algorithm, with its advantages and limitations.
8. Explain the working of the Sobel edge operator with its kernels and an example computation.
9. Describe the five stages of the Canny edge detection algorithm.
10. Explain the structure tensor (second-moment matrix) and how its eigenvalues are used to classify a pixel as flat, edge, or corner.
11. Explain the double-angle representation of edge orientation and why it is needed.
12. Compare Roberts, Prewitt, Sobel, and Canny edge detectors on the basis of noise immunity and localisation accuracy.

2.15 Ten-Mark Questions

1. Discuss in detail the statistical model of noise in images. Derive the effect of a linear smoothing/difference filter on noise variance, and explain the fundamental trade-off between noise suppression and edge localisation with the help of suitable diagrams.
2. Derive the first and second derivatives of the 1-D Gaussian function. Explain how these derivative-of-Gaussian filters are used to build the gradient and Laplacian-of-Gaussian edge detectors, extending the discussion to 2-D using separability.

3. With neat diagrams, explain why the Gaussian is the optimal choice of smoothing filter for edge detection, covering the uncertainty principle, the non-creation of new zero-crossings, separability, and rotational symmetry.

4. Explain the Marr–Hildreth edge detection algorithm based on the Laplacian-of-Gaussian operator. Derive the LoG kernel and describe, with a diagram, how zero-crossings are used to localise edges. Discuss its merits and demerits compared to gradient-based methods.

5. Describe the Canny edge detection algorithm in complete detail, covering Gaussian smoothing, gradient computation, non-maximum suppression, double thresholding, and hysteresis edge tracking. Justify how the algorithm satisfies Canny's three optimality criteria.

6. Explain the concept of the structure tensor (second-moment matrix) for corner detection. Derive the Harris corner response function and explain, using eigenvalue analysis and suitable diagrams, how flat regions, edges, and corners are distinguished. Discuss applications of corner detection in computer vision.

7. Compare and contrast the Laplacian-based and gradient-based approaches to edge detection, covering their mathematical basis, computational cost, noise sensitivity, localisation accuracy, and typical use cases.

References

- David A. Forsyth, Jean Ponce, "Computer Vision — A Modern Approach", 2nd Edition, Pearson/PHI, 2003 — Chapter 8: Edge Detection.
- R. C. Gonzalez, R. E. Woods, "Digital Image Processing", 4th Edition, Pearson, 2018 — Chapter 10: Image Segmentation.
- J. Canny, "A Computational Approach to Edge Detection", IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. PAMI-8, no. 6, pp. 679–698, 1986.
- C. Harris, M. Stephens, "A Combined Corner and Edge Detector", Proceedings of the 4th Alvey Vision Conference, pp. 147–151, 1988.
- D. Marr, E. Hildreth, "Theory of Edge Detection", Proceedings of the Royal Society of London B, vol. 207, pp. 187–217, 1980.

COMPUTER VISION — UNIT III 1

TEXTURE 1

Table of Contents

COMPUTER VISION — UNIT III

TEXTURE

Topics covered: Representing Texture — Extracting Image Structure with Filter Banks; Representing Texture using the Statistics of Filter Outputs; Analysis (and Synthesis) Using Oriented Pyramids — The Laplacian Pyramid; Filters in the Spatial Frequency Domain; Oriented Pyramids; Application — Synthesizing Textures for Rendering; Homogeneity; Synthesis by Sampling Local Models; Shape from Texture; Shape from Texture for Planes.

Textbook reference: David A. Forsyth, Jean Ponce, *Computer Vision — A Modern Approach*, PHI, 2003.

Table of Contents

3.1 Introduction — What Is Texture?

Texture refers to the visual and tactile quality of a surface arising from the repetition, with variation, of small structural elements ("textons" or "texels") — such as the weave of fabric, the grain of wood, the pattern of bricks in a wall, gravel, foliage, or the ripples on water. Formally, texture in an image is characterized not by the intensity value at a single pixel, but by the **statistical or structural regularities of intensity variation over a neighborhood**.

Why study texture in computer vision?

1. **Segmentation cue** — regions of an image can often be told apart by their texture even when average brightness or color is similar (e.g., distinguishing grass from a paved path).
2. **Recognition cue** — many materials and object categories (fur, bark, brick, fabric) are most easily recognized by their texture.

3. **Shape cue** — the systematic distortion of a *regular* texture pattern under perspective projection ("texture gradient") reveals the 3-D orientation and shape of the surface it lies on — this is **shape from texture**, covered in §3.10–3.11.
4. **Synthesis and rendering** — realistic computer graphics requires the ability to generate large amounts of natural-looking texture from a small example (§3.7, §3.9).

Texture vs. a single edge. A single edge is a local, oriented, one-dimensional discontinuity. Texture is inherently a **neighborhood** (patch-level) property — no single pixel or simple edge measurement is "textured"; texture emerges only when we look at the statistics of many local measurements over a region. This is the central reason texture representation methods are built from **banks of filters** applied over a window, rather than a single filter applied at a point (as with edge detection in Unit II).

Two broad approaches to representing texture:

- **Filter-bank / statistical approaches** — decompose the image using many filters tuned to different scales and orientations, then summarize each region by the statistics (e.g., energy, histograms) of the filter responses. (§3.2, §3.3)
- **Structural / local-model approaches** — treat texture as small repeating primitives placed according to some placement rule, and represent/synthesize texture by directly manipulating these local samples. (§3.9)

Roadmap of this unit. We begin with the filter-bank machinery for extracting local texture structure (§3.2) and the statistics used to summarize it into a compact descriptor (§3.3). We then examine the frequency-domain view of these filters (§3.4), which motivates the joint scale-orientation organization of the oriented pyramid, building on the (purely scale-organized) Laplacian pyramid introduced in Unit I (§3.5–3.6). With a texture representation in hand, we turn to two important applications: synthesizing novel texture for computer graphics rendering (§3.7, §3.9), for which the notion of a **homogeneous** texture region (§3.8) is the key underlying assumption; and recovering 3-D surface shape from the geometric distortion of a known or regular texture pattern (§3.10–3.11), a classical monocular shape-recovery technique that complements the stereo- and motion-based methods studied elsewhere in a full computer vision course.

3.2 Representing Texture: Extracting Image Structure with Filter Banks

3.2.1 The core idea

The human visual system's primary visual cortex (V1) contains neurons selectively responsive to intensity variation at a **particular orientation and spatial frequency (scale)** within a small receptive field. Computer vision mimics this by convolving an image with a **filter bank** — a large, fixed collection of filters, each tuned to a different combination of:

- **Scale** (coarse vs. fine spatial frequency), and
- **Orientation** (the direction along which the filter is most sensitive to intensity change).

At every pixel, the *vector* of responses from all filters in the bank forms a rich local descriptor of the texture at that location — this is dramatically more informative than a single grayscale value, and it is the input to essentially all statistical texture representations.

3.2.2 Types of filters used in a texture filter bank

Oriented Derivative-of-Gaussian filters (from Unit II, §2.4) — tuned to a scale σ and orientation θ; respond strongly to oriented edges/bars in the texture.

Oriented Laplacian-of-Gaussian / difference-of-Gaussian filters — isotropic, blob-detecting; useful for capturing "spot"-like texture elements.

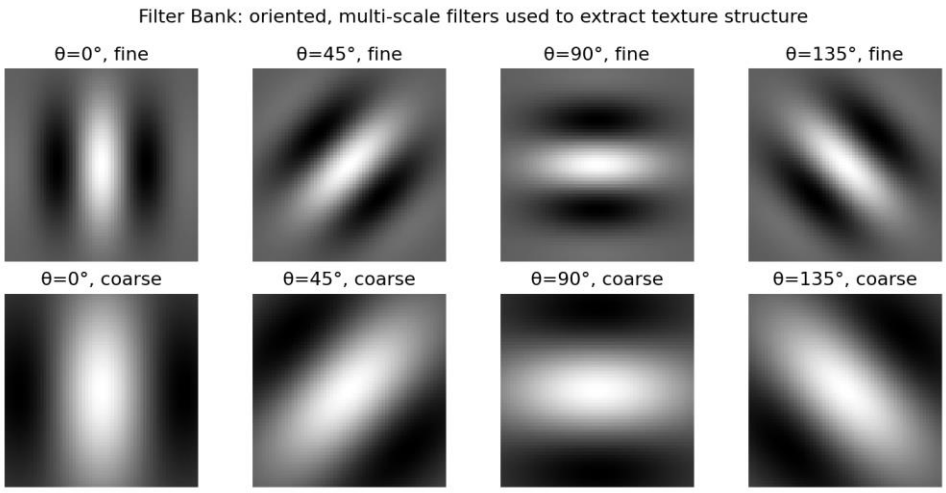
Gabor filters — a sinusoidal grating windowed by a Gaussian envelope, giving a filter tuned jointly to a spatial frequency, orientation, and scale (spatial extent):

$$G(x, y; \theta, f, \sigma) = exp\left(-\frac{x_\theta^2 + \gamma^2 y_\theta^2}{2\sigma^2}\right)\cos(2\pi f\, x_\theta + \phi)$$

where

$$x_\theta = x\cos\theta + y\sin\theta, \qquad y_\theta = -x\sin\theta + y\cos\theta$$

Here f is the sinusoid's spatial frequency, θ its orientation, σ the Gaussian envelope width, γ the spatial aspect ratio, and ϕ the phase offset.



Filter bank: a set of oriented, multi-scale filters (e.g., Gabor-like) used to decompose texture. Rows = scale, columns = orientation.

3.2.3 The MR8 / LM Filter Banks (representative examples used in practice)

Two influential, standard filter banks in the texture-recognition literature:

- **Leung–Malik (LM) filter bank** — 48 filters: first and second derivative-of-Gaussian filters at 6 orientations and 3 scales, plus 8 LoG filters and 4 Gaussian filters at different scales.
- **MR8 (Maximum Response 8) filter bank** — derived from a larger rotationally-varying filter set, but the *maximum* response across orientations is kept for each scale, reducing dimensionality to 8 responses per pixel while remaining approximately rotation-invariant.

3.2.4 Filtering as a Convolution Pipeline

Given filter bank $\{h_1, h_2, \ldots, h_N\}$ (each tuned to a distinct scale/orientation combination), the response image for filter k is:

$$R_k(x, y) = (I * h_k)(x, y)$$

Stacking these gives, at every pixel, an N-dimensional **filter response vector**:

$$\mathbf{r}(x, y) = [R_1(x, y),\ R_2(x, y),\ \ldots,\ R_N(x, y)]^T$$

This vector is the fundamental unit from which texture is statistically characterized (§3.3).

3.2.5 Computational Considerations

Applying a full filter bank of N filters, each of spatial support $k \times k$, to an $M \times M$ image by direct convolution costs:

$$\text{Cost}_{direct} = O(N \cdot M^2 \cdot k^2)$$

For a realistic bank ($N = 48$ filters as in LM, $k = 49$ for the largest-scale filters, $M = 512$) this is prohibitively expensive if computed naively. Two standard optimizations are used in practice:

1. **Separable filter design** — where possible (e.g., for Gaussian-derivative filters, which factor into 1-D x- and y-components), reduce each 2-D convolution to two 1-D passes (Unit I, §1.2.4), cutting the k^2 factor to $2k$.
2. **Frequency-domain (FFT-based) filtering** — using the convolution theorem (Unit I, §1.4.3), transform the image once via FFT ($O(M^2 \log M)$), multiply by each filter's precomputed frequency response ($O(M^2)$ per filter), and inverse-transform: total cost $O(N \cdot M^2 + M^2 \log M)$, which is dramatically cheaper than direct spatial convolution whenever k is large relative to $\log M$.
3. **Pyramid-based computation** — since texture filter banks are applied across many scales, computing them on a Gaussian/Laplacian pyramid (Unit I, §1.8) rather than at full resolution for every scale directly reduces the *effective* image size M^2 at coarser scales geometrically (each octave has a quarter the pixels), giving an overall cost across all scales that is only a small constant multiple of the cost at the finest scale alone — this is one of the strongest practical motivations for building texture analysis on top of a pyramid representation rather than computing every scale densely at full image resolution.

3.2.6 Worked Problem 3.1

A filter bank has 6 orientations $\{0°, 30°, 60°, 90°, 120°, 150°\}$ *and 3 scales* $\{\sigma = 1,2,4\}$*, plus 4 isotropic (rotation-independent) filters. How many total filter responses exist per pixel, and what is the dimensionality of the local texture descriptor at each pixel?*

Solution.

Oriented filters: 6 orientations $\times$ 3 scales $= 18$ filters.

Isotropic filters: 4 filters.

Total filters $= 18 + 4 = 22$.

Therefore, the local texture descriptor (filter-response vector) at each pixel has dimensionality **22** — i.e., each pixel is now represented not by 1 grayscale number, but by a 22-dimensional feature vector summarizing multi-scale, multi-orientation local structure.

3.3 Representing Texture Using the Statistics of Filter Outputs

3.3.1 From per-pixel filter vectors to a region descriptor

A single pixel's filter response vector is still a *local* measurement (dominated by whatever texture element happens to sit under the filter support at that exact location). Texture, however, is a *regional* property. We therefore compute the **statistics of filter responses over a neighborhood or region**, discarding exact spatial arrangement in favor of an aggregate, translation-invariant summary.

3.3.2 Energy Statistics

The simplest, most common summary is the **mean squared response ("energy")** of each filter over a window W:

$$E_k = \frac{1}{|W|} \sum_{(x,y)\in W} R_k\,(x,y)^2$$

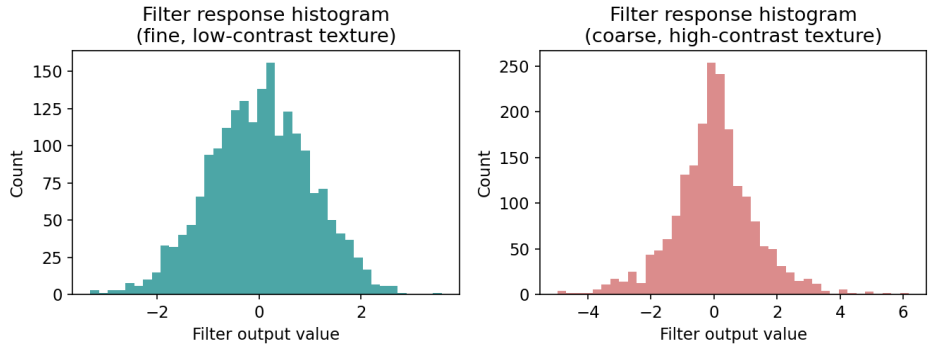
Stacking the E_k for all N filters in the bank gives an N-dimensional **texture energy vector** describing the region — this is a compact, powerful descriptor widely used for texture classification and texture-based segmentation.

3.3.3 Histogram Statistics

A richer representation retains the full (or binned) **histogram** of each filter's responses over the region, rather than collapsing to a single energy number:

$$h_k(b) = \#\{(x,y) \in W : R_k(x,y) \in \text{bin } b\}$$

This captures higher-order statistics (skewness, multi-modality) that energy alone discards — important when a texture has, e.g., two very different types of edges mixed together (bimodal response).



Filter-response histograms differ characteristically between fine, low-contrast textures (left) and coarse, high-contrast textures (right).

3.3.4 Textons and Vector Quantization

The **textons** approach clusters the N-dimensional filter response vectors (pooled across many training images) into K cluster centers ("textons") using an algorithm such as K-means (introduced formally in Unit IV). Each pixel's filter vector is then replaced by the identity of its nearest texton — this converts the continuous, high-dimensional descriptor into a compact discrete label. A texture region is then represented by the **histogram of texton labels** occurring within it:

$$T(x,y) = arg\min_{k\in\{1,\dots,K\}} \| \mathbf{r}(x,y) - \boldsymbol{\mu}_k \|$$

where $\boldsymbol{\mu}_k$ are the learned texton cluster centers. Texton histograms are the basis of many classical texture-classification pipelines (e.g., the Leung–Malik / MR8 + textons pipeline).

3.3.5 Statistical Properties Commonly Computed

Beyond energy and histograms, other statistics are frequently used to summarize filter outputs or raw intensity co-occurrence within a region:

- **Mean** $\mu_k = \frac{1}{|W|}\sum R_k$
- **Variance** $\sigma_k^2 = \frac{1}{|W|}\sum (R_k - \mu_k)^2$
- **Skewness** (asymmetry of the response distribution)
- **Kurtosis** (peakedness/tail-heaviness of the response distribution)
- **Co-occurrence statistics** (Gray-Level Co-occurrence Matrix, GLCM) — the joint probability of two intensity values occurring at a fixed relative displacement $(\Delta x, \Delta y)$:

$$P(i,j \mid \Delta x, \Delta y) = Pr(I(x,y) = i,\ I(x+\Delta x, y+\Delta y) = j)$$

From the GLCM, well-known scalar texture measures are derived, e.g.:

$$\text{Contrast} = \sum_{i,j} (i-j)^2 \; P(i,j), \qquad \text{Energy (Angular Second Moment)} = \sum_{i,j} P\,(i,j)^2$$

$$\text{Homogeneity} = \sum_{i,j} \frac{P(i,j)}{1+|i-j|}, \qquad \text{Entropy} = -\sum_{i,j} P\,(i,j)\log P(i,j)$$

3.3.6 Worked Problem 3.2

A 4×4 image region has only two gray levels, 0 and 1, arranged as:

$$\begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{bmatrix}$$

Compute the GLCM for horizontal displacement $(\Delta x, \Delta y) = (1,0)$*, and then compute the Energy and Contrast measures.*

Solution.

Count all horizontally-adjacent pixel pairs $\big(I(x,y), I(x+1,y)\big)$ across the 4 rows (3 pairs per row × 4 rows = 12 pairs total):

Row 1: (0,0)=0-0, (0,1)=0-1, (1,1)=1-1 → pairs: (0,0),(0,1),(1,1) Row 2: same as row 1 → (0,0),(0,1),(1,1) Row 3: (1,1),(1,0),(0,0) → (1,1),(1,0),(0,0) Row 4: same as row 3 → (1,1),(1,0),(0,0)

Tally: $(0,0)$: 4 times; $(0,1)$: 2 times; $(1,1)$: 4 times; $(1,0)$: 2 times. Total = 12.

$$P(0,0) = 4/12 = 0.333, \quad P(0,1) = 2/12 = 0.167, \quad P(1,0) = 2/12 = 0.167, \quad P(1,1) = 4/12 = 0.333$$

Energy: $\sum P(i,j)^2 = 0.333^2 + 0.167^2 + 0.167^2 + 0.333^2 = 0.111 + 0.0278 + 0.0278 + 0.111 = 0.278$

Contrast: $\sum (i-j)^2 P(i,j) = (0)^2(0.333) + (1)^2(0.167) + (1)^2(0.167) + (0)^2(0.333) = 0.167 + 0.167 = 0.333$

The moderate energy (0.278, versus a maximum of 1.0 for a completely uniform region) and non-zero contrast (0.333) numerically confirm this checkerboard-like region is neither perfectly homogeneous nor maximally random — consistent with its visibly blocky, alternating structure.

3.3.7 Rotation and Scale Invariance in Statistical Descriptors

A practical difficulty with filter-based texture descriptors is that the *raw* energy vector (§3.3.2) is **not invariant** to rotation of the texture or to changes in viewing scale — rotating a fabric sample

by 45° changes which oriented filters respond strongly, even though a human observer would call it "the same texture." Two standard remedies are used:

1. **Rotation invariance via orientation pooling** — instead of retaining all K orientation channels at a given scale separately, retain only the **maximum** (or the sum) response over all orientations at that scale (this is exactly the strategy used by the MR8 filter bank, §3.2.3). This discards orientation information but yields a descriptor unaffected by in-plane rotation of the texture.
2. **Scale invariance via pyramid normalization** — because a texture's dominant spatial frequency shifts predictably with viewing scale (Worked Problem 3.3), descriptors can be made scale-invariant by first estimating the dominant scale (e.g., the pyramid level of maximum energy, §3.5.3) and re-centering the filter bank's scale range on that dominant scale before extracting the final descriptor.

These invariance strategies are essential in practice: a texture classifier trained only on fronto-parallel, fixed-scale samples will otherwise fail badly whenever the same material is viewed from a different distance or in-plane rotation — an extremely common occurrence in real images.

3.3.8 Worked Problem 3.2(b) — Rotation-Invariant Descriptor

A texture is measured with an oriented filter bank at one scale and 4 orientations $\{0°, 45°, 90°, 135°\}$*, giving energies* $E = [12,30,45,18]$*. The same physical texture, rotated by 45° in the world, gives energies* $E' = [18,12,30,45]$ *at the same four filter orientations (i.e., a cyclic shift). Show that the ordinary energy vector changes with rotation, but the MR8-style "max energy" descriptor does not.*

Solution.

The raw vectors are clearly different: $E = [12,30,45,18] \neq E' = [18,12,30,45]$ — a classifier comparing these two vectors directly (e.g., by Euclidean distance) would judge them as substantially different textures, even though they arise from the same physical material.

$$D_{Euclidean}(E, E') = \sqrt{(12-18)^2 + (30-12)^2 + (45-30)^2 + (18-45)^2}$$
$$= \sqrt{36 + 324 + 225 + 729} = \sqrt{1314} \approx 36.2$$

This large distance would be a **false negative** for texture matching.

Now consider the rotation-invariant descriptor $E_{\max} = \max_\theta E_\theta$:

$$E_{\max} = max(12,30,45,18) = 45, \qquad E'_{\max} = max(18,12,30,45) = 45$$

$$D(E_{\max}, E'_{\max}) = |45 - 45| = 0$$

The max-pooled descriptor correctly identifies the two measurements as **identical**, confirming that orientation-pooling successfully removes the effect of the 45° rotation, at the cost of discarding the information about which orientation was dominant.

3.4 Filters in the Spatial Frequency Domain

3.4.1 Why view texture filters in frequency space

Texture is fundamentally about *repetition at some characteristic spacing* — and repetition at a fixed spacing corresponds exactly to **energy concentrated at a specific spatial frequency** (recall Unit I, §1.4). Viewing filters (and textures) in the frequency domain therefore gives the clearest possible account of what a texture filter bank is doing: **each filter isolates a narrow band of orientation and frequency, and the image's local frequency content — its "texture" — is read off from how much energy falls into each band.**

3.4.2 Frequency Response of a Gabor Filter

The Fourier transform of a Gabor filter (§3.2.2) is itself a Gaussian, shifted to be centered at the tuned frequency $\pm f$ along orientation θ:

$$\mathcal{F}\{G(x, y; \theta, f, \sigma)\} \approx exp\left(-\frac{\sigma^2\left((u_\theta - f)^2 + \gamma^{-2}v_\theta^2\right)}{2}\right)$$

where (u_θ, v_θ) are frequency coordinates rotated into the filter's own reference frame. This shows explicitly that a Gabor filter is a **band-pass filter** localized around frequency f and orientation θ — precisely the "frequency + orientation tuned receptive field" model motivated in §3.2.1.

3.4.3 Tiling the Frequency Plane

A well-designed filter bank should have its individual filters' frequency responses **tile the 2-D frequency plane** — covering all frequencies and orientations of interest with some overlap, but without large gaps (which would make the bank blind to certain textures) or excessive redundancy (which wastes computation). This tiling principle directly motivates the **oriented pyramid** structure of §3.6, in which scale is handled by an octave-spaced pyramid and orientation is handled by multiple oriented filters at each pyramid level.

3.4.4 Worked Problem 3.3

A Gabor filter is tuned to spatial frequency $f = 0.25$ cycles/pixel and orientation $\theta = 0°$. What is the smallest repeating spacing (in pixels) of a texture pattern to which this filter will respond most strongly? If the same texture is viewed from twice the distance (linear size halved), what new frequency tuning would best match it?

Solution.

Spacing $= 1/f = 1/0.25 = 4$ pixels — the filter responds maximally to a periodic pattern that repeats every 4 pixels along the x-direction ($\theta = 0°$).

If viewed from twice the distance, the pattern's linear size in the image halves, so its repeat spacing halves to 2 pixels, meaning its spatial frequency **doubles** to $f' = 0.5$ cycles/pixel. This illustrates precisely why texture analysis must be performed **across multiple scales** (a pyramid

of filters, §3.5–3.6) — the same physical texture produces different image spatial frequencies depending on viewing distance/scale.

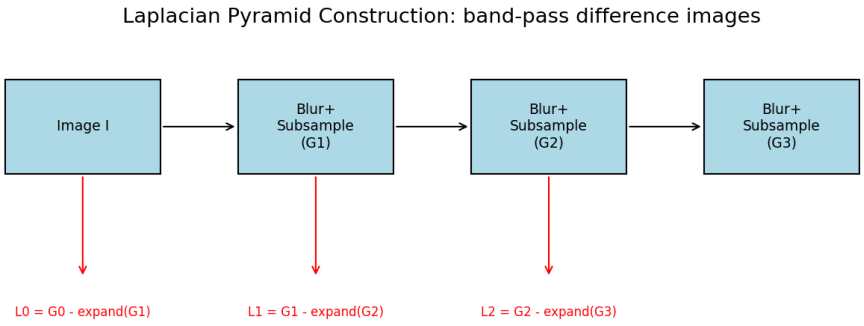
3.5 Analysis Using Oriented Pyramids: The Laplacian Pyramid

3.5.1 Recap: The Laplacian Pyramid (from Unit I)

Recall from Unit I that the **Laplacian pyramid** decomposes an image into a set of band-pass "detail" images $L_0, L_1, \ldots, L_n$ plus a final low-pass residual G_n:

$$L_l = G_l - \text{Expand}(G_{l+1}), \qquad G_l = \downarrow_2 (G_{l-1} * G_\sigma)$$

Each level L_l isolates the image structure that exists at a particular **scale (octave) band** — but critically, **the ordinary Laplacian pyramid has no orientation selectivity**: L_l responds to structure of the right scale in *any* orientation simultaneously, since it is built from an isotropic Gaussian/Laplacian.



The Laplacian Pyramid isolates band-pass "detail" images at successive octave scales — the scale axis of a texture filter bank.

3.5.2 Why This Matters for Texture

Texture generally has structure that is both **scale-specific** (e.g., "coarse-grained" vs. "fine-grained") **and orientation-specific** (e.g., vertical stripes vs. diagonal weave). The Laplacian pyramid alone handles only the scale axis. To fully characterize texture, we need a decomposition that is selective in **both scale and orientation simultaneously** — leading directly to the **oriented pyramid** (§3.6), which augments each level of the scale pyramid with a bank of oriented filters.

3.5.3 Analysis via Pyramid Sub-band Energies

A simple but effective texture descriptor is obtained by computing the **energy** (mean squared value) within each Laplacian pyramid level:

$$E_l = \frac{1}{|L_l|} \sum_{(x,y)} L_l\,(x,y)^2$$

A texture dominated by fine detail will have high energy concentrated in the finer (higher-resolution, l small) levels; a coarse, blobby texture will show energy concentrated in coarser levels. Plotting E_l versus level l gives a compact, rotation-blind "scale signature" of the texture.

3.5.4 Worked Problem 3.4

A 4-level Laplacian pyramid (L_0, L_1, L_2, L_3) is computed for two texture patches, A (fine sand) and B (large pebbles). The energies are:

$$E^A = [50,30,12,4], \qquad E^B = [5,15,40,60]$$

Normalize each energy profile and interpret the results.

Solution.

Sum for A: $50 + 30 + 12 + 4 = 96$. Normalized: $[0.521,0.313,0.125,0.042]$

Sum for B: $5 + 15 + 40 + 60 = 120$. Normalized: $[0.042,0.125,0.333,0.5]$

Texture A's energy is concentrated at **level 0** (the finest scale) — consistent with fine sand, whose grain size is small and therefore contains predominantly high-frequency (fine-scale) structure. Texture B's energy is concentrated at **level 3** (the coarsest scale) — consistent with large pebbles, whose structure repeats at a much larger spatial period, placing its energy in low-frequency/coarse pyramid levels. This confirms that pyramid sub-band energy is a simple, effective discriminator of texture "coarseness."

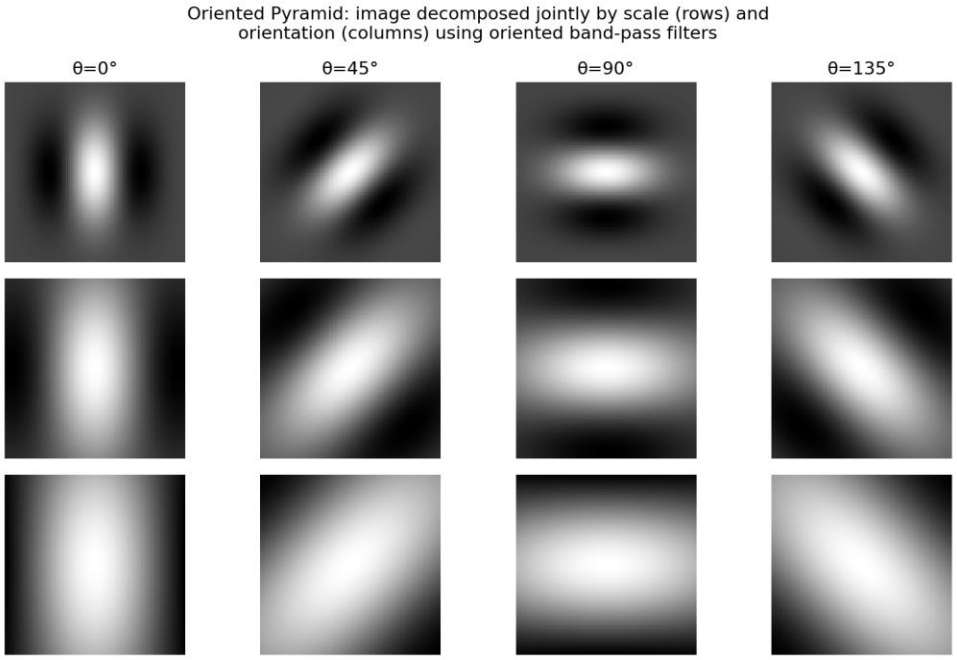
3.6 Oriented Pyramids

3.6.1 Definition

An **oriented pyramid** (also called a *steerable pyramid* in its most refined form, Simoncelli & Freeman) generalizes the Laplacian pyramid by decomposing each octave/scale band **further into several orientation sub-bands**, using a set of oriented band-pass filters at each pyramid level:

$$B_{l,\theta}(x,y) = (G_l * h_\theta)(x,y), \qquad \theta \in \{\theta_1, \theta_2, \ldots, \theta_K\}$$

where h_θ is an oriented band-pass filter (e.g., a directional derivative or Gabor-like kernel) tuned to orientation θ, applied at pyramid level l (i.e., at that level's spatial scale).



Oriented Pyramid: at each scale (row), the image is further decomposed into several orientation channels (columns), giving a joint scale–orientation representation.

3.6.2 Steerability

A particularly elegant property, exploited heavily in texture and edge analysis, is **steerability**: if the basis filters are chosen appropriately (e.g., directional derivatives of a Gaussian), the response at *any arbitrary orientation* θ can be computed as a **linear combination** of the responses of just a few basis filters, without explicitly convolving with a filter at every possible orientation:

$$R_\theta(x,y) = \sum_{k=1}^{K} c_k\,(\theta)\, R_{\theta_k}(x,y)$$

For example, the first derivative of a 2-D Gaussian is steerable using only $K = 2$ basis orientations (e.g., 0° and 90°):

$$G'_\theta = cos\theta\ G'_{0^\circ} + sin\theta\ G'_{90^\circ}$$

This dramatically reduces the computational cost of evaluating orientation-tuned filters at a dense set of orientations.

3.6.3 Advantages of Oriented Pyramids for Texture

1. **Joint scale-orientation selectivity** matches the true nature of most textures.
2. **Efficient, non-redundant tiling** of the frequency domain (§3.4.3).

3. **Rotation-covariant representation** — rotating the input image rotates the set of oriented responses in a predictable way, enabling rotation-invariant descriptors (e.g., via the MR8-style max-over-orientation trick, §3.2.3).
4. **Basis for texture synthesis** — the oriented pyramid coefficients of an example texture can be resampled/matched to synthesize novel texture with the same local statistical structure (§3.7, §3.9), since coefficients at each scale/orientation approximately capture the "texture look" independently of exact pixel position.

3.6.4 General Steering Formula and Higher-Order Filters

The steering result of §3.6.2 generalizes: an N^{th}-order directional derivative of a Gaussian requires exactly $N + 1$ basis filters to be steered to any orientation. For example, the **second** derivative of a Gaussian (closely related to the oriented, elongated edge/ridge detectors used in some texture filter banks) requires $N + 1 = 3$ basis orientations, e.g., $0°, 60°, 120°$, with steering coefficients:

$$G''_{\theta}(x, y) = \sum_{k=0}^{2} c_k(\theta)\, G''_{\theta_k}(x, y)$$

where the $c_k(\theta)$ are known trigonometric interpolation functions determined once for the chosen basis set. This generalization is important because texture filter banks frequently use second-derivative or higher-order oriented filters (which have narrower, more orientation-selective tuning curves than first-derivative filters) to better discriminate finely-oriented texture patterns such as woven fabric or wood grain.

Practical implication for filter-bank design. Because steerability lets us synthesize the response at *any* orientation cheaply from a small fixed basis, a texture-analysis system does not need to store or compute convolutions for every one of, say, 36 finely-spaced orientations — it needs only the 2–3 basis filters per scale and order, computing finer angular resolution on demand. This is one of the main practical reasons steerable/oriented pyramids became the standard tool for texture and oriented-structure analysis in preference to brute-force, densely-sampled filter banks.

3.6.5 Worked Problem 3.5

A steerable filter is built from 2 basis filters, $G'_{0°}$ and $G'_{90°}$, with measured responses at a pixel of $R_{0°} = 6$ and $R_{90°} = 8$. Compute the steered response at $\theta = 30°$, and also find the orientation at which the response is maximal.

Solution.

Steered response:

$$R_{30°} = cos(30°)\, R_{0°} + sin(30°)\, R_{90°} = (0.866)(6) + (0.5)(8) = 5.196 + 4 = 9.196$$

Maximal orientation: the response as a function of θ is $R(\theta) = R_{0°}\cos\theta + R_{90°}\sin\theta$, which can be written as $A\cos(\theta - \phi)$ with $A = \sqrt{R_{0°}^2 + R_{90°}^2}$ and $\phi = \tan^{-1}(R_{90°}/R_{0°})$:

$$A = \sqrt{6^2 + 8^2} = \sqrt{100} = 10, \qquad \phi = \tan^{-1}(8/6) = 53.13^\circ$$

So the response is maximal ($= 10$) at $\theta = 53.13^\circ$ — this is the **dominant local orientation** at the pixel, recovered from just two filter evaluations thanks to steerability.

3.7 Application: Synthesizing Textures for Rendering

3.7.1 Why synthesize texture?

Photorealistic computer graphics needs to cover large 3-D surfaces (terrain, fabric, walls) with texture, but capturing (or hand-painting) a full-resolution image for every surface is impractical. **Texture synthesis** algorithms take a small example texture patch and generate an arbitrarily large output that is perceptually similar — statistically indistinguishable in local structure — without simply tiling the source (which produces visible, repeating seams).

3.7.2 Filter-Bank / Pyramid-Based Synthesis (Heeger–Bergen style)

A classical approach (Heeger & Bergen, 1995) proceeds as follows:

1. Decompose the **example texture** into an oriented pyramid (§3.6), and record the histogram of coefficients at every scale/orientation sub-band.
2. Initialize the **output image** as random noise.
3. Decompose the noise image with the same oriented pyramid.
4. At each sub-band, **histogram-match** the noise image's coefficients to the example texture's coefficients at that sub-band (i.e., replace the noise coefficients' distribution with the target distribution, preserving rank order).
5. Reconstruct the (now-matched) pyramid back into an image.
6. **Iterate** steps 3–5 for several passes, since matching one sub-band's histogram slightly perturbs others upon reconstruction, and repeated iteration converges to an image that jointly matches all sub-band statistics.

This process, though iterative, is efficient and works well for "stochastic" (irregular, texture-like) patterns; it is less successful for highly regular/structured textures (e.g., a brick wall), which are better handled by structural/local-model methods (§3.9).

3.7.3 Requirements for Good Synthesis

- **Local coherence** — synthesized texture should look plausible in any local window, matching the example's local statistics.
- **No visible seams or periodic repetition** unless the source texture itself is periodic.
- **Boundary/scale consistency** — texture should tile seamlessly across large surfaces and remain plausible under the different apparent scales induced by perspective (relevant when texture is being applied to a 3-D surface for rendering, tying back to shape-from-texture considerations in §3.10–3.11 in reverse: correctly-scaled texture communicates the correct surface geometry to a viewer).

3.7.4 Worked Problem 3.6

A texture-synthesis algorithm processes a target output of 1024×1024 *pixels from an example texture of* 64×64 *pixels. If naive tiling were used instead, how many tile repetitions would appear along each dimension, and why is this visually undesirable?*

Solution.

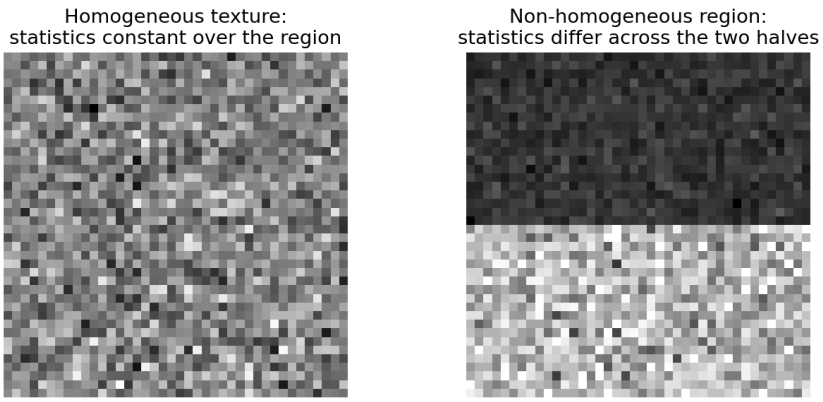
Repetitions per dimension $= 1024/64 = 16$.

Naive tiling would repeat the exact same 64×64 patch 16 times along each axis (256 total tiles), producing highly visible periodic repetition — any distinctive feature in the source patch (a particular fiber clump, a stray mark) would appear at a perfectly regular grid spacing of 64 pixels, an artifact immediately obvious to a human viewer and a signature the eye is highly tuned to detect (per Gestalt grouping principles, Unit IV). Statistical/structural synthesis methods avoid this by generating output that matches the *statistics* of the example without literally repeating it verbatim.

3.8 Homogeneity

3.8.1 Definition

A texture region is **homogeneous** if its defining statistical properties (filter-response energies, histograms, or local-model parameters) are **constant (stationary) across the region** — i.e., any sufficiently large sub-window of the region, regardless of position, yields approximately the same statistical descriptor.



Left: a homogeneous texture — local statistics remain constant across the region. Right: a non-homogeneous region — two visually and statistically distinct sub-regions.

3.8.2 Why Homogeneity Matters

- **Segmentation:** homogeneity is precisely the property that defines a "texture region" for segmentation purposes (Unit IV) — a segmentation algorithm using texture cues seeks to partition the image into maximal regions of homogeneous texture.

- **Sample sufficiency for synthesis:** homogeneity justifies using a small patch as representative of an entire (potentially much larger) surface's texture for synthesis purposes (§3.7, §3.9) — if the texture were not homogeneous, no small sample could correctly characterize the whole.
- **Statistical estimation validity:** many texture descriptors (energy, histograms, GLCM) implicitly assume the region being measured is homogeneous; applying them across a boundary between two different textures produces a meaningless "average" descriptor that matches neither texture.

3.8.3 Testing for Homogeneity

A simple practical test: split a candidate region into two (or more) sub-windows and compare their filter-response statistics using a suitable distance measure (e.g., a χ^2 distance between histograms, or Euclidean distance between energy vectors). If the distance is below a threshold for all splits, the region is treated as homogeneous; large distances indicate the region actually straddles a texture boundary and should be further segmented.

$$D_{\chi^2}(h_1, h_2) = \sum_b \frac{\big(h_1(b) - h_2(b)\big)^2}{h_1(b) + h_2(b)}$$

3.8.4 Worked Problem 3.7

Two halves of a candidate texture region give energy vectors (3 filters) $E_{left} = [10,4,2]$ *and* $E_{right} = [10.5,3.8,2.3]$. *A second candidate region gives* $E_{left} = [10,4,2]$ *and* $E_{right} = [40,1,15]$. *Using Euclidean distance and a homogeneity threshold of 2.0, classify each region.*

Solution.

Region 1:

$$D = \sqrt{(10-10.5)^2 + (4-3.8)^2 + (2-2.3)^2} = \sqrt{0.25 + 0.04 + 0.09} = \sqrt{0.38} \approx 0.616$$

$0.616 < 2.0 \rightarrow$ **homogeneous**.

Region 2:

$$D = \sqrt{(10-40)^2 + (4-1)^2 + (2-15)^2} = \sqrt{900 + 9 + 169} = \sqrt{1078} \approx 32.83$$

$32.83 \gg 2.0 \rightarrow$ **not homogeneous**; the region should be split/segmented, as its two halves have drastically different texture statistics.

3.9 Synthesis by Sampling Local Models

3.9.1 The structural/non-parametric alternative

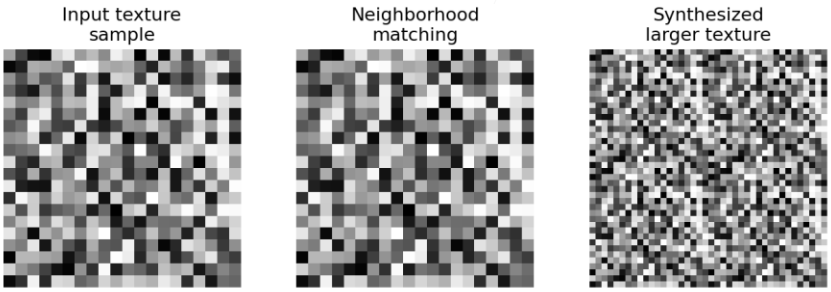
Rather than matching global filter-bank statistics (§3.7), an alternative and highly successful family of algorithms directly **copies small local neighborhoods** from the example texture into the output, choosing at each output location a source patch that best matches the (already-synthesized) surrounding context. This treats texture synthesis as a **non-parametric sampling problem**: the true generative distribution of the texture is unknown, so we sample directly from the empirical example instead of fitting a parametric statistical model.

3.9.2 The Efros–Leung Pixel-Growing Algorithm

1. Start with a small seed of already-known pixels (e.g., a small patch copied from the example texture, placed in a corner of the output).
2. Consider an unfilled output pixel adjacent to already-filled pixels; extract its (partially filled) neighborhood window.
3. Search the **example texture** for all windows whose already-filled portion closely matches this neighborhood (e.g., by sum of squared differences, restricted to filled pixels).
4. **Randomly select** one of the best-matching windows (not necessarily the single best match — this randomization is essential to avoid literally recreating the source verbatim and to introduce natural-looking variation) and copy its center pixel into the output.
5. Repeat until the entire output is filled, always growing outward from filled to unfilled regions.

3.9.3 Efros–Freeman Image Quilting

A faster variant copies whole **patches** (not single pixels) from the example into the output, then blends overlapping patch boundaries using a **minimum-error boundary cut** (a shortest-path / dynamic-programming search through the region of overlap that finds the seam with least visible discontinuity) — this preserves large-scale structure much better and is far faster than pixel-by-pixel growing.



Structural texture synthesis: local neighborhoods are matched and copied from an example sample to grow a larger, seamless output texture.

3.9.4 Choosing the Neighborhood/Patch Size

The neighborhood window size is a crucial parameter:

- **Too small:** insufficient context to capture the texture's characteristic large-scale structure (e.g., a small window cannot "see" an entire brick, only fragments of mortar lines), producing an unstructured, mushy result.
- **Too large:** the algorithm degenerates toward copying large verbatim chunks of the source (little novel variation), and computation cost grows substantially.

A common heuristic ties window size to the texture's dominant repeat period, estimable from the pyramid energy-peak scale (§3.5.3) or from autocorrelation of the example patch.

3.9.5 Worked Problem 3.8

In the Efros-Leung algorithm, a 9×9 neighborhood window is used, and the example texture is 80×80 pixels. How many candidate window positions exist for a full search (ignoring border effects), and if only the best-matching window (no randomization) were always chosen, what visual artifact would likely occur, and why?

Solution.

Valid top-left positions for a 9×9 window within an 80×80 image (no wraparound): $(80 - 9 + 1) \times (80 - 9 + 1) = 72 \times 72 = 5184$ candidate window positions.

If only the single best match were always chosen deterministically, the synthesis would tend to **repeatedly copy the same, most "generic"/frequently-matching region of the source** (a form of the well-known "garbage growing" or verbatim-copying artifact), producing visible repeated patterns or even a degenerate output that looks like small tiled copies of one favored source patch, rather than natural-looking novel variation. Randomly selecting among several near-equally-good matches (e.g., within a small tolerance of the best score) is precisely what avoids this artifact and gives the perceptually convincing, non-repetitive results the algorithm is known for.

3.9.6 Comparing Statistical and Structural Synthesis Approaches

Aspect	Statistical (pyramid histogram-matching, §3.7)	Structural (neighborhood copying, §3.9)
Best suited for	Stochastic, irregular textures (sand, clouds, foliage)	Structured/regular textures (bricks, weave patterns)
Preserves large-scale structure	Often weakly	Strongly (especially with patch-based quilting)
Computational cost	Iterative but bounded (fixed number of passes)	Search-dominated; can be slow for pixel-by-pixel growing
Risk of visible artifacts	Blurring / loss of sharp structure	Verbatim copying / mismatched seams if search is deterministic
Underlying	Texture is fully described by	Texture is well-approximated by

Aspect	Statistical (pyramid histogram-matching, §3.7)	Structural (neighborhood copying, §3.9)
assumption	marginal statistics of filter sub-bands	resampling local neighborhoods from an example

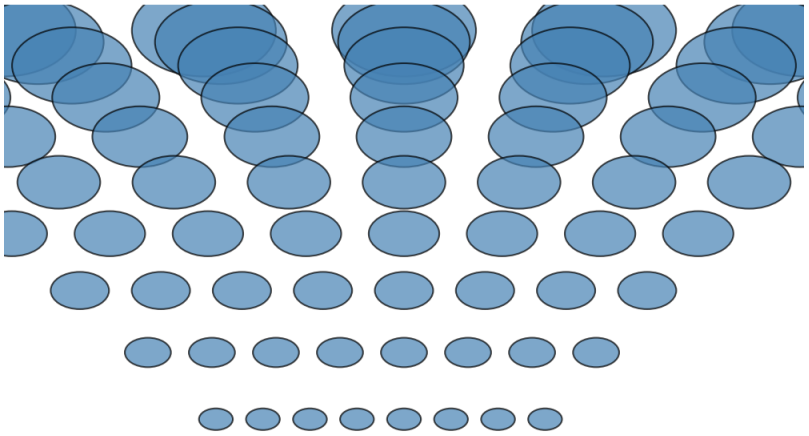
This comparison explains why practical texture-synthesis systems sometimes **hybridize** the two ideas — e.g., using patch-based structural copying for the dominant large-scale layout of a texture, while applying statistical sub-band matching as a refinement pass to correct any remaining local statistical mismatches at fine scales.

3.10 Shape from Texture

3.10.1 The Basic Idea

When a **regular or statistically homogeneous texture** (e.g., a tiled floor, a textured fabric) lies on a 3-D surface and is viewed under perspective (or even orthographic) projection, the *appearance* of the texture elements in the 2-D image is systematically distorted according to the surface's local orientation and the imaging geometry. This distortion — the **texture gradient** — can be measured from the image and *inverted* to recover the 3-D shape (local surface orientation) of the surface, without needing stereo, motion, or any other depth cue. This is the **shape-from-texture** problem, one of several classical "shape-from-X" monocular shape-recovery techniques.

Texture Gradient on a Slanted Plane:
t size and spacing shrink with distance — this gradient encodes surface orie



Texture gradient on a slanted plane: as the surface recedes from the viewer, texture elements systematically shrink in size and compress in spacing along the direction of maximum slant — this gradient directly encodes the surface's 3-D orientation.

3.10.2 Two Types of Texture Distortion Cues

1. **Foreshortening (compression) cue** — texture elements are compressed more in the direction of maximum surface slant than perpendicular to it; the *ratio* of apparent size along these two directions directly reveals the slant angle.
2. **Perspective scaling (size-gradient) cue** — under perspective projection, texture elements farther from the camera project to smaller image size purely due to increased distance (even on a fronto-parallel surface); this size gradient encodes the surface's depth gradient, and together with foreshortening reveals full 3-D orientation.

Real algorithms typically combine both cues, extracted from measurements of local texture element size, shape, spacing, and orientation across the image.

3.10.3 Statistical (Filter-Based) Approaches to Shape from Texture

Rather than explicitly detecting and measuring individual texture elements, statistical shape-from-texture methods analyze the **local spatial-frequency content** using an oriented filter bank (§3.6) at every image location:

- On a fronto-parallel (perpendicular-to-view) homogeneous texture, the dominant local spatial frequency and orientation from the filter bank is **constant** across the image.
- On a slanted/tilted surface, perspective/foreshortening cause the **dominant frequency to increase, and the dominant orientation to rotate, systematically across the image** — precisely tracking the direction and rate of surface slant.

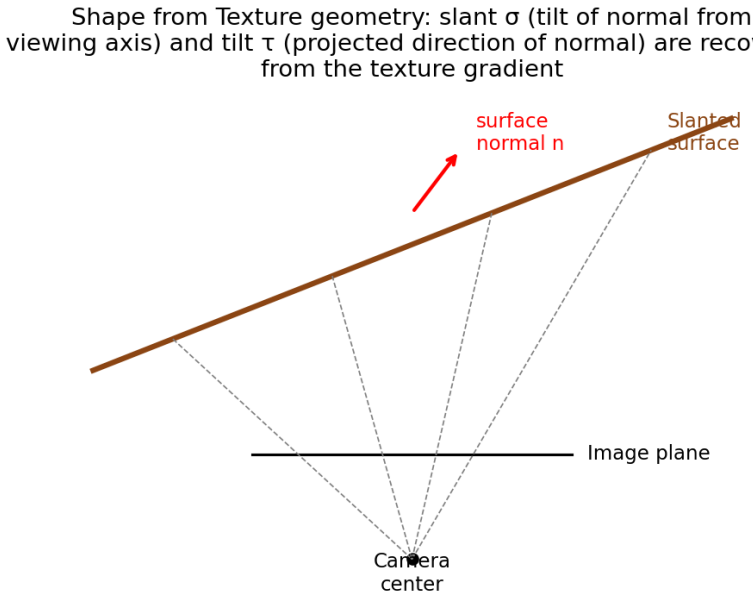
By fitting a model of how frequency and orientation should vary under a given surface shape hypothesis (e.g., planar, §3.11, or a smoothly curved surface) to the *measured* frequency/orientation field, the surface orientation at every point can be estimated.

3.11 Shape from Texture for Planes

3.11.1 Slant and Tilt

For the important special case of a **planar surface**, its orientation relative to the camera is completely described by two angles:

- **Slant** σ — the angle between the surface normal $\mathbf{n}$ and the camera's viewing axis (optical axis). $\sigma = 0$ means the surface is fronto-parallel (facing the camera directly); larger σ means more oblique.
- **Tilt** τ — the direction (angle in the image plane) along which the surface normal is tilted away from the viewing axis — i.e., the projected direction of steepest recession of the plane.



Shape-from-texture geometry for a planar surface: slant σ is the angle between the surface normal and the viewing axis; tilt τ is the projected direction of the normal in the image.

3.11.2 Recovering Tilt

Tilt is generally the easier parameter to recover: it is simply the image-plane direction along which the **texture gradient (rate of change of apparent element size/spacing) is greatest** — i.e., the direction of maximum foreshortening/compression. Equivalently, in the filter-bank formulation (§3.10.3), tilt is the orientation at which the dominant spatial frequency changes most rapidly across the image.

3.11.3 Recovering Slant

Slant is recovered from the **magnitude** of the foreshortening effect along the tilt direction. A standard result relates slant to the ratio of the apparent texture-element dimensions measured along the tilt direction versus perpendicular to it. If a locally-isotropic texture element (e.g., approximately circular) has apparent (projected) semi-axes a (along tilt direction) and b (perpendicular to tilt direction) in the image, then:

$$\cos\sigma = \frac{a}{b}$$

since foreshortening compresses the element's extent along the tilt direction by exactly $\cos\sigma$, while leaving its extent perpendicular to tilt unaffected. Equivalently, using the ratio of measured spatial frequencies from the filter-bank approach (frequency along tilt direction $f_{\parallel}$, versus perpendicular $f_{\perp}$):

$$\cos\sigma = \frac{f_{\perp}}{f_{\parallel}}$$

(a locally circular texture patch's spatial frequency increases along the direction of compression, since the same physical period now spans fewer image pixels).

3.11.4 Additional Refinement Using the Perspective Scaling Cue

For a full perspective (rather than orthographic) camera model, the systematic *increase* in apparent spatial frequency across the image (due to increasing distance, independent of slant) provides an additional constraint that, combined with the foreshortening-based slant estimate, improves accuracy and can help disambiguate the true 3-D distance scale up to the classical shape-from-texture depth/scale ambiguity (texture-based methods generally recover *shape*, i.e., orientation, reliably, but absolute scale/distance only up to an unknown constant, without an additional reference such as a known true texture-element size).

3.11.5 Worked Problem 3.9

A texture element that is circular in the real (undistorted, fronto-parallel) world projects into the image as an ellipse with semi-major axis (perpendicular to tilt) $b = 10$ pixels and semi-minor axis (along tilt direction) $a = 6$ pixels. Compute the surface slant σ.

Solution.

$$\cos\sigma = \frac{a}{b} = \frac{6}{10} = 0.6$$

$$\sigma = \cos^{-1}(0.6) \approx 53.13^\circ$$

The surface is tilted away from fronto-parallel by approximately 53.13° — a strongly oblique (steeply receding) surface, consistent with the significant 40% compression observed along the tilt direction.

3.11.6 Worked Problem 3.10

Using the filter-bank frequency method, the dominant spatial frequency measured perpendicular to the tilt direction is $f_\perp = 0.10$ cycles/pixel, and along the tilt direction it is $f_\parallel = 0.25$ cycles/pixel. (a) Compute the slant. (b) If the tilt direction was measured as $\tau = 40^\circ$ from the image horizontal, state the full recovered plane orientation.

Solution.

$$\cos\sigma = \frac{f_\perp}{f_\parallel} = \frac{0.10}{0.25} = 0.4$$

$$\sigma = \cos^{-1}(0.4) \approx 66.42^\circ$$

(a) The recovered planar surface orientation is fully specified by the pair:

$$(\sigma, \tau) = (66.42^\circ,\ 40^\circ)$$

meaning the plane's normal is tilted 66.42° away from the camera's viewing axis, with the direction of steepest recession (maximum compression) pointing at 40° from the image

horizontal. This is a very oblique surface (close to "edge-on" to the camera, since σ is close to 90°), such as a floor viewed at a shallow, grazing angle.

3.11.7 Worked Problem 3.11 — Comparing Two Surface Patches

Two different regions of a photographed textured floor give the following measurements:

Region	$f_{\perp}$ (cyc/px)	$f_{\parallel}$ (cyc/px)	Tilt τ
P	0.08	0.10	10°
Q	0.05	0.20	75°

Compute the slant for each region and comment on which region of the floor is more oblique relative to the camera, and what this implies about the floor's shape.

Solution.

Region P:

$$\cos\sigma_P = \frac{0.08}{0.10} = 0.8 \quad \Rightarrow \quad \sigma_P = \cos^{-1}(0.8) \approx 36.87^\circ$$

Region Q:

$$\cos\sigma_Q = \frac{0.05}{0.20} = 0.25 \quad \Rightarrow \quad \sigma_Q = \cos^{-1}(0.25) \approx 75.52^\circ$$

Region Q has a substantially larger slant (75.52° vs. 36.87°) — it is much more oblique (closer to edge-on) relative to the camera than region P. Since both regions were measured on the same physical floor, this systematic difference in recovered slant, together with the difference in recovered tilt direction (10° vs. 75°), is consistent with the floor being **non-planar** (e.g., a floor that curves or ramps away from the camera) — a single global (σ, τ) pair could not explain both measurements simultaneously if the surface were perfectly flat and the camera model were orthographic. This illustrates how shape-from-texture, applied locally across many small regions of an image, can recover *curved* surface shape (not just single flat planes) by piecing together many local slant/tilt estimates, each computed exactly as in §3.11.3.

3.11.8 Summary Comparison: Planar Shape-from-Texture Cues

Cue	What it measures	Recovers
Foreshortening (compression ratio)	Ratio of texture-element extent along vs. across tilt direction	Slant σ
Frequency gradient direction	Direction of maximum rate of change of dominant spatial frequency	Tilt τ
Perspective size gradient	Overall shrinkage of element size with image position, at fixed slant	Additional depth-scale constraint
Local frequency comparison across regions	Consistency (or inconsistency) of recovered (σ, τ) across the image	Detects planar vs. curved surfaces

3.12 Unit Summary

- Texture is a *regional* (neighborhood) property, best represented by decomposing the image with a **bank of multi-scale, multi-orientation filters** (Gabor, oriented derivative-of-Gaussian, LoG) rather than any single-pixel measurement.
- Local filter-response vectors are summarized by region-level **statistics** — energy, histograms, textons (via vector quantization), or gray-level co-occurrence measures — to give a compact, discriminative texture descriptor.
- Viewing filters in the **spatial-frequency domain** clarifies that texture filter banks are designed to **tile the frequency-orientation plane**, motivating the joint scale-orientation structure of the **oriented (steerable) pyramid**, an extension of the ordinary (isotropic) Laplacian pyramid.
- **Steerability** allows dense orientation tuning from just a few basis filter responses via simple linear combination.
- **Texture synthesis** for graphics/rendering can be approached either statistically (matching oriented-pyramid sub-band histograms, Heeger–Bergen) or structurally/non-parametrically (copying and blending local neighborhoods from an example, Efros–Leung / image quilting).
- **Homogeneity** — statistical stationarity of texture descriptors across a region — is the defining property exploited by both texture-based segmentation (Unit IV) and texture synthesis (a small homogeneous sample can represent an entire large surface).
- **Shape from texture** inverts the systematic geometric distortion (foreshortening/compression and perspective size-gradient) that a regular texture undergoes when imaged on a 3-D surface, recovering local surface orientation; for planar surfaces this reduces to estimating just two angles, **slant** (σ) and **tilt** (τ), from measured ratios of texture-element size or dominant spatial frequency along and across the tilt direction.

3.13 Review Questions

1. Explain why texture cannot be characterized by a single-pixel or single-filter measurement, and describe the role of a filter bank in addressing this.
2. Compare and contrast Gabor filters and oriented derivative-of-Gaussian filters as building blocks of a texture filter bank.
3. Derive the “energy” texture descriptor from a set of filter responses and explain the difference between energy statistics and histogram statistics.
4. Define textons and explain, step by step, how a texton-histogram descriptor is built for an image region.
5. Explain the Gray-Level Co-occurrence Matrix (GLCM) and derive at least two scalar texture measures from it.
6. Why does an ordinary (isotropic) Laplacian pyramid fail to fully capture texture information, and how does the oriented pyramid address this limitation?
7. Explain the concept of steerability and derive the steering formula for a first-derivative-of-Gaussian filter pair.

8. Describe the Heeger–Bergen texture synthesis algorithm and explain why it is applied iteratively across pyramid sub-bands.
9. Contrast statistical (filter-histogram-matching) texture synthesis with structural (local neighborhood copying) texture synthesis, giving one advantage and one disadvantage of each.
10. Define homogeneity in the context of texture and explain a practical statistical test for whether a candidate image region is homogeneous.
11. Explain, with a labeled diagram, the meaning of slant and tilt for a planar surface, and derive the formula relating slant to the ratio of apparent texture-element dimensions.
12. A circular texture element projects to an ellipse of semi-axes 8 and 5 pixels. Compute the surface slant. (Practice problem — solve independently using the method of Worked Problem 3.9.)
13. Explain why shape-from-texture methods generally recover surface *orientation* reliably but not absolute *scale/distance*, and what additional information would resolve this ambiguity.
14. Discuss two applications (outside of graphics rendering) where texture synthesis is a useful computer vision or image-processing tool.
15. Explain, using the frequency-domain view of filters, why the same physical texture produces different measured spatial frequencies at different viewing distances, and how this affects the design of a texture-analysis filter bank.

UNIT – IV

SEGMENTATION BY CLUSTERING

Topics Covered:

What is Segmentation

Human Vision: Grouping and Gestalt

Applications: Shot Boundary Detection and Background Subtraction

Image Segmentation by Clustering Pixels

Segmentation by Graph-Theoretic Clustering

The Hough Transform

Fitting Lines

Fitting Curves

Textbook: David A. Forsyth, Jean Ponce, "Computer Vision – A Modern Approach", PHI, 2003

4.1 What is Segmentation?

Image segmentation is the process of partitioning a digital image into multiple regions or sets of pixels, such that pixels within a region are homogeneous with respect to some characteristic (intensity, colour, texture, motion), while pixels in different regions are markedly different with respect to the same characteristic. Segmentation is one of the earliest and most important steps in a computer vision pipeline, because most later processes — recognition, description, measurement — operate not on individual pixels but on meaningful groups of pixels called regions or tokens.

Formally, if I is an image defined over a spatial domain Ω, segmentation seeks a partition of Ω into n disjoint regions R1, R2, …, Rn such that:

$$\Omega = R1 \cup R2 \cup \ldots \cup Rn\ ,\ Ri \cap Rj = \emptyset \text{ for } i \neq j$$

together with a uniformity predicate P, such that P(Ri) = TRUE for every region Ri (all pixels in a region satisfy the homogeneity criterion) and P(Ri ∪ Rj) = FALSE whenever Ri and Rj are adjacent regions (merging two different regions must break the homogeneity criterion).

Segmentation is fundamentally an ill-posed and perceptual problem — there is no single, universally correct segmentation of an image. What counts as a meaningful region depends heavily on the task at hand (e.g. segmenting a person from a background versus segmenting a shirt from skin). This is why the study of segmentation begins by drawing on principles of human perceptual grouping, discussed in Section 4.2, before moving on to computational algorithms.

Why Segment an Image?

- To reduce the amount of data that later stages of a vision system must process, by replacing pixels with a small number of tokens (regions) that carry equivalent information.
- To group pixels into perceptually meaningful units — objects, object parts, or surfaces — that correspond to structures in the real, three-dimensional world.
- To provide a mid-level representation of the image that acts as a bridge between low-level pixel processing (filtering, edge detection) and high-level scene understanding (recognition).
- To support applications such as medical image analysis, object tracking, background/foreground separation, image editing, and content-based retrieval.

General Approaches to Segmentation

- Region-based approaches – group pixels sharing common properties (e.g. clustering methods, region growing, split-and-merge).
- Boundary-based approaches – locate edges/boundaries that separate regions (e.g. edge detection followed by edge linking).
- Graph-based approaches – model the image as a weighted graph and segment by partitioning the graph (e.g. minimum cut, normalized cuts).
- Model/Fitting-based approaches – fit parametric shapes (lines, circles, curves) to grouped features, useful when objects have simple geometric structure (Hough transform, least-squares fitting).

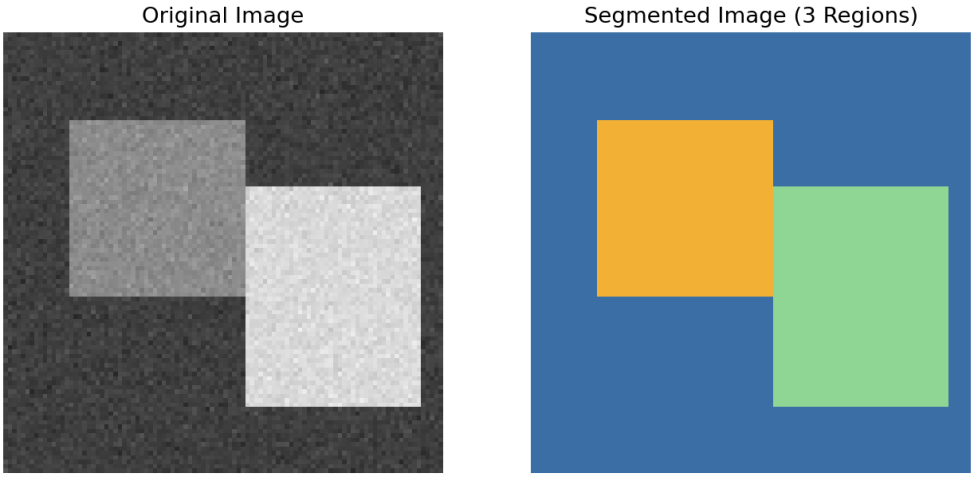


Fig. 4.1 — An image (left) segmented into three homogeneous regions (right) based on intensity similarity.

Desirable Properties of a Good Segmentation

- Regions should be uniform and homogeneous with respect to the chosen characteristic.
- Region interiors should be simple and free of small holes.
- Adjacent regions should have significantly different values with respect to the characteristic on which they are uniform.
- Boundaries of each region should be smooth and spatially accurate.

Note: Segmentation is generally treated as a clustering problem in an appropriate feature space (e.g. colour space, texture space, or the (x, y, intensity) space), which motivates the clustering-based techniques studied in this unit.

4.2 Human Vision: Grouping and Gestalt

Long before computational vision existed, psychologists studying human perception observed that the human visual system does not perceive a scene as an unstructured collection of independent points of light. Instead, the brain automatically organizes visual elements into coherent wholes — a phenomenon studied extensively by the Gestalt school of psychology in the early twentieth century (Wertheimer, Koffka, Köhler). The central tenet of Gestalt psychology is often summarized as: "the whole is different from the sum of its parts."

These observations are directly relevant to computer vision because the "grouping laws" discovered by Gestalt psychologists describe exactly the kind of low-level perceptual organization that segmentation algorithms attempt to replicate computationally. The most important Gestalt grouping laws are:

(a) Proximity

Elements that are close to one another in space tend to be perceptually grouped together into a single unit, even when the elements themselves are otherwise identical and unconnected.

(b) Similarity

Elements that share similar visual properties — colour, shape, size, orientation, or texture — tend to be grouped together, regardless of their spatial proximity.

(c) Closure

The human visual system tends to "close up" or complete figures that have gaps in their boundary, perceiving an incomplete shape as a whole, closed object.

(d) Continuation (Good Continuation)

Elements arranged along a smooth path — a straight line or a smoothly curving line — tend to be grouped as continuing that path, rather than being seen as sharply changing direction.

(e) Common Fate

Elements that move in the same direction, or at the same speed, are perceived as belonging together — this principle is especially important for motion-based grouping in video.

(f) Common Region

Elements that are enclosed within the same boundary (e.g. inside the same rectangle or region outline) are grouped together, even if they are otherwise dissimilar.

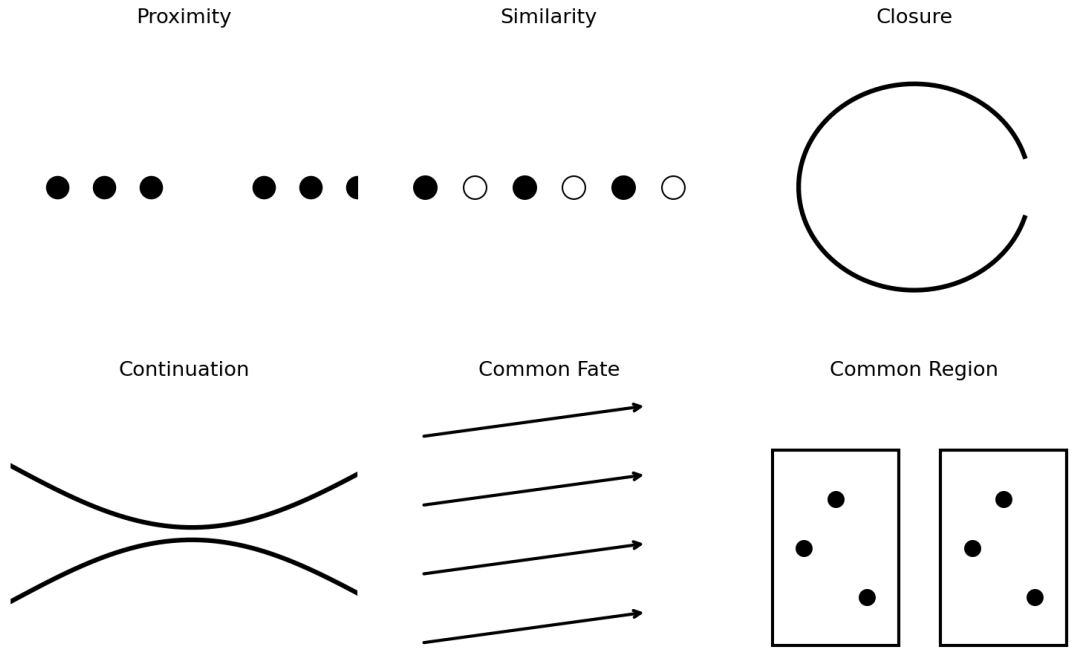


Fig. 4.2 — Illustration of the principal Gestalt grouping laws used to motivate perceptual grouping in vision algorithms.

Figure–Ground Organization

Another key Gestalt phenomenon is figure–ground segregation — the tendency of the visual system to divide a scene into a 'figure' (the object of interest, which appears to be in front and has a definite shape) and a 'ground' (the background, which appears to extend behind the figure and has no definite shape at that boundary). Classic ambiguous images (such as Rubin's vase) show that the same boundary can define two different figures depending on which side is interpreted as figure and which as ground.

Relevance to Computational Segmentation

- Clustering-based segmentation algorithms formalize similarity grouping by clustering pixels in a feature space.

- Graph-based methods formalize proximity and similarity together by defining edge weights between neighbouring, similar pixels.
- The Hough transform and curve-fitting methods formalize good continuation, by grouping edge points that lie along a common geometric curve despite gaps caused by noise or occlusion.
- Common fate is exploited in motion segmentation, where pixels/regions moving with a consistent optical flow are grouped as one object.

Note: Although Gestalt laws were discovered qualitatively, modern vision research treats them as inspiration for the design of the similarity/affinity measures used inside clustering and graph-partitioning algorithms.

4.3 Applications: Shot Boundary Detection and Background Subtraction

Segmentation ideas extend naturally from single still images to video sequences, where the additional dimension of time provides powerful cues for grouping. Two classical applications illustrate this: segmenting a video into individual camera shots (shot boundary detection), and segmenting each video frame into a moving foreground and a static background (background subtraction).

4.3.1 Shot Boundary Detection

A video is composed of a sequence of shots, where a shot is an unbroken sequence of frames captured continuously by a single camera. Shot boundary detection is the task of automatically identifying the points in a video where one shot ends and another begins, so that the video can be segmented (in time) into its constituent shots. This is a critical first step for video summarization, indexing, and content-based video retrieval.

Shot transitions are broadly of two kinds:

- Abrupt transitions (cuts) — an instantaneous change from one shot to the next between two consecutive frames.
- Gradual transitions — the change occurs over several frames through effects such as fades, dissolves, or wipes.

Frame Differencing Method

The simplest approach compares consecutive frames using a distance/difference metric D(t) between frame t and frame t+1. A commonly used pixel-based metric is the sum of absolute differences (SAD):

$$D(t) = (1 / N) \cdot \Sigma x \, \Sigma y \, | \, I\, t(x, y) - I\, t+1(x, y) \, |$$

where I t(x, y) is the intensity of pixel (x, y) in frame t, and N is the total number of pixels. A shot boundary is declared whenever D(t) exceeds a threshold T:

Shot Boundary at t $\Leftrightarrow D(t) > T$

Because raw pixel differencing is highly sensitive to camera motion, object motion, and illumination changes, more robust variants compare colour histograms of successive frames instead of raw pixel values:

$D\ hist(t) = \Sigma i \mid H\ t(i) - H\ t+1(i) \mid$

where H t(i) is the count of pixels in frame t falling into histogram bin i. Histogram-based differencing is far less sensitive to small local motions because it discards spatial information and compares only the global distribution of intensities/colours.

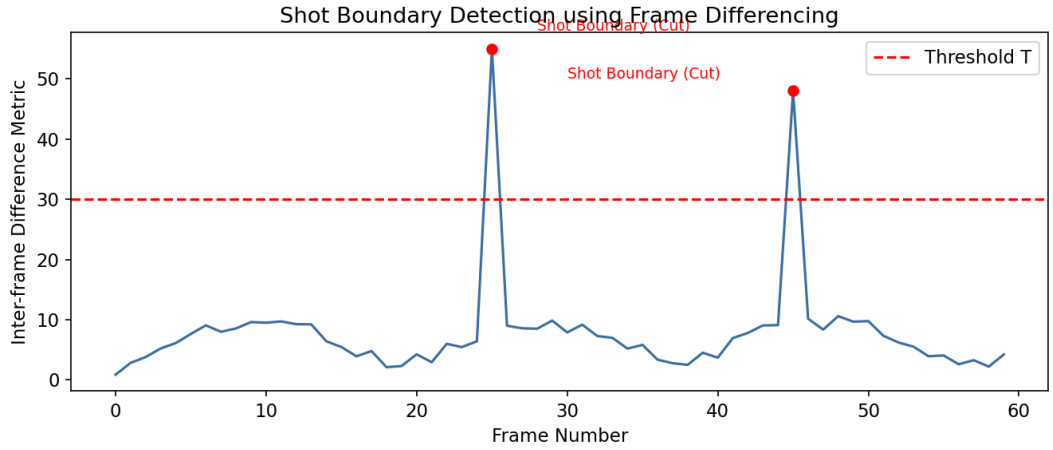


Fig. 4.3 — Frame-differencing curve showing two abrupt shot boundaries (cuts) as sharp peaks above threshold T.

Twin-Comparison Method for Gradual Transitions

For gradual transitions, a single high threshold misses the boundary because the frame-to-frame difference is small at every step even though the cumulative difference across the whole transition is large. The twin-comparison technique uses two thresholds: a high threshold Th to catch cuts, and a lower threshold Tl to detect the start of a potential gradual transition, after which the cumulative difference from the potential start frame is accumulated and compared against Th to confirm the end of the transition.

4.3.2 Background Subtraction

Background subtraction (also called foreground detection) is a widely used technique for segmenting the moving objects (foreground) in a video captured by a static camera from the static parts of the scene (background). It underlies applications such as visual surveillance, traffic monitoring, and human activity analysis.

The basic idea is to maintain a model B(x, y) of the background — an estimate of what the scene looks like when no foreground objects are present — and classify a pixel in the current frame I t(x, y) as foreground if it differs sufficiently from the background model:

Foreground Mask: M t(x, y) = 1 if | I t(x, y) − B(x, y) | > T, else 0

Building the Background Model

- Frame Differencing: use the previous frame as the background estimate — simple, but sensitive to noise and slow-moving objects.
- Running (Exponential) Average: continuously update the background estimate with a learning rate α ($0 < \alpha < 1$) so that the model adapts to slow illumination changes:

$$B_{t+1}(x, y) = \alpha \cdot I_t(x, y) + (1 - \alpha) \cdot B_t(x, y)$$

- Median Filtering over Time: maintain a buffer of the last n frames and set the background to the per-pixel median — robust to short-duration foreground occlusions.
- Mixture of Gaussians (MoG): model the intensity value of each pixel over time as a mixture of K Gaussian distributions, updated recursively; a pixel is classified as background if it matches one of the high-weight, low-variance Gaussians in its mixture:

$$P(I_t(x, y)) = \Sigma_{k=1..K} w_{k,t} \cdot \square(I_t(x, y); \mu_{k,t}, \sigma^2_{k,t})$$

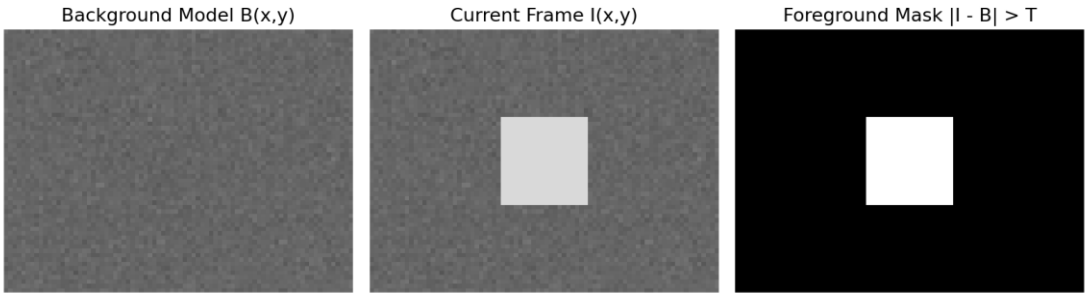


Fig. 4.4 — Background model, current frame containing a moving object, and the resulting binary foreground mask.

Challenges in Background Subtraction

- Illumination changes — gradual (daylight change) or sudden (lights switching on/off).
- Dynamic backgrounds — waving trees, rippling water, flickering monitors.
- Shadows cast by foreground objects being incorrectly classified as foreground.
- Camouflage — foreground objects whose colour closely matches the background.
- Bootstrapping — building an initial background model from a video that already contains moving foreground objects from the very first frame.

Note: Both shot boundary detection and background subtraction can be viewed as segmentation problems along the time axis (or across the space–time volume of a video), reinforcing that clustering-style reasoning generalizes naturally from spatial to spatio-temporal domains.

4.4 Image Segmentation by Clustering Pixels

A natural and very general way to formulate segmentation is as a clustering problem: represent each pixel by a feature vector (which might include its colour, position, or texture descriptors), and then partition the set of feature vectors into clusters such that vectors within a cluster are close to one another (by some distance measure) and vectors in different clusters are far apart. Each resulting cluster, when mapped back to image coordinates, corresponds to one segmented region.

A pixel at location (x, y) may, for instance, be represented in feature space by the vector:

$f(x, y) = [\ R(x,y), G(x,y), B(x,y), x, y\ ]^{T}$

combining colour and spatial position so that the resulting clusters are both colour-consistent and spatially compact.

4.4.1 K-Means Clustering

K-means is the most widely used clustering algorithm for segmentation, owing to its simplicity and efficiency. Given the number of clusters K, K-means seeks to partition n feature vectors {f1, f2, …, fn} into K clusters {C1, …, CK} so as to minimize the within-cluster sum of squared distances to the cluster mean (centroid):

$J = \Sigma k=1..K\ \Sigma\ (fi \in Ck)\ \| fi - \mu k \|^2$

where μk is the centroid (mean feature vector) of cluster Ck. This objective function J is often called the distortion or the sum-of-squared-error (SSE) criterion.

K-Means Algorithm

- 1. Choose the number of clusters K, and initialize K cluster centroids μ1, …, μK (e.g. randomly from the data points).
- 2. Assignment step: assign every feature vector fi to the nearest centroid: fi → Ck such that k = arg minj ‖ fi − μj ‖².
- 3. Update step: recompute each centroid as the mean of all feature vectors currently assigned to it: μk = (1 / |Ck|) Σ(fi ∈ Ck) fi .
- 4. Repeat steps 2 and 3 until the assignments no longer change (convergence), or a maximum number of iterations is reached.

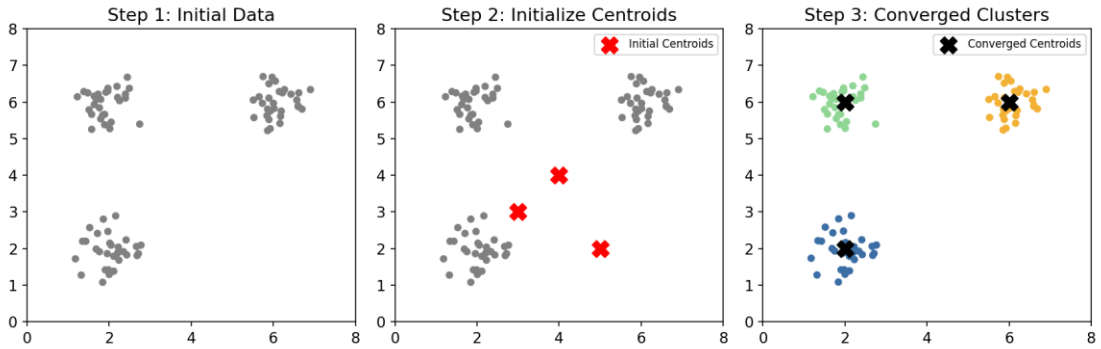


Fig. 4.5 — K-means clustering: (1) raw data, (2) randomly initialized centroids, (3) converged clusters after iterative assignment and update steps.

K-means is guaranteed to converge to a local minimum of J, but the result depends heavily on the initial choice of centroids; it is common practice to run K-means several times with different random initializations and keep the result with the lowest final J. K-means also implicitly assumes clusters are roughly spherical (isotropic) and of comparable size, which limits its ability to capture elongated or irregularly-shaped image regions.

4.4.2 Clustering by Mixture Models (Expectation–Maximization)

A probabilistic generalization of K-means models the feature vectors as being drawn from a mixture of K Gaussian distributions, where each Gaussian corresponds to one segment/cluster:

$P(f) = \Sigma k=1..K\ \pi k \cdot \square(f ; \mu k , \Sigma k)$

where πk is the mixing weight of component k (Σ πk = 1), and □(f ; μk , Σk) is a multivariate Gaussian with mean μk and covariance Σk. The parameters {πk, μk, Σk} are estimated using the Expectation–Maximization (EM) algorithm:

- E-step: compute the responsibility (posterior probability) that component k generated data point fi, using the current parameter estimates: γik = πk □(fi; μk, Σk) / Σj πj □(fi; μj, Σj).
- M-step: re-estimate πk, μk, Σk as weighted averages using the responsibilities γik as weights.
- Repeat E and M steps until the log-likelihood of the data converges.

Because it allows for elliptical (not just spherical) clusters and gives a soft (probabilistic) assignment of each pixel to each segment, the EM/mixture-of-Gaussians approach is generally more flexible than hard K-means, at the cost of higher computational complexity.

4.4.3 Agglomerative (Hierarchical) Clustering

Agglomerative clustering builds a hierarchy of clusters bottom-up: it begins by treating every feature vector as its own singleton cluster, and then repeatedly merges the pair of clusters that are closest together (by some inter-cluster distance measure, such as single-link, complete-link, or

average-link distance) until only one cluster remains or a stopping criterion is met. The sequence of merges can be displayed as a dendrogram, and a segmentation is obtained by cutting the dendrogram at an appropriate level.

4.4.4 Mean Shift Clustering

Mean shift is a non-parametric, mode-seeking clustering procedure. It treats the feature vectors as samples from an underlying probability density function and iteratively shifts each candidate point towards the nearest local maximum (mode) of the estimated density, using a kernel density estimate:

$$\hat{f}(x) = (1 / n\, h^{d})\, \Sigma i=1..n\, K((x - fi) / h)$$

where K is a kernel function (commonly Gaussian) and h is the bandwidth parameter that controls the scale of grouping. All points that converge to the same mode form a cluster. Mean shift does not require the number of clusters K to be specified in advance, and naturally produces clusters of arbitrary shape, unlike K-means.

4.4.5 Watershed Segmentation (Brief Note)

The watershed algorithm treats the image's gradient magnitude as a topographic surface, where high gradient values correspond to "ridges" (likely boundaries) and low, flat regions correspond to "catchment basins" (likely region interiors). Simulated flooding of this surface from local minima produces watershed lines exactly at the ridges, yielding a segmentation. Watershed segmentation is often applied after clustering or smoothing to avoid excessive over-segmentation caused by noise.

> **Note:** K-means is by far the most examination-relevant clustering technique for segmentation; be prepared to write out its objective function J and the four-step algorithm precisely.

4.5 Segmentation by Graph-Theoretic Clustering

Graph-theoretic methods formulate image segmentation as a graph-partitioning problem. The image is represented as a weighted, undirected graph G = (V, E), where each vertex vi ∈ V corresponds to a pixel (or a small group of pixels), and each edge (vi, vj) ∈ E connects two pixels — usually neighbouring pixels, or all pairs within some feature-space neighbourhood — with a weight w(vi, vj) that measures the similarity (affinity) between them. A common weight function combines spatial proximity and feature (colour/intensity) similarity:

$$w(vi, vj) = exp(- \| f i - f j \|^{2} / 2\sigma I^{2} - \| x i - x j \|^{2} / 2\sigma X^{2})$$

where fi is the feature vector (e.g. intensity or colour) at pixel i, xi is its spatial location, and σI, σX are scale parameters. High weights indicate that two pixels are likely to belong to the same region; low weights indicate they likely belong to different regions. Segmentation is then achieved by removing ("cutting") a set of low-weight edges so as to partition the graph into disjoint subgraphs, each subgraph corresponding to one segmented region.

4.5.1 Minimum Cut

Given a partition of the vertex set V into two disjoint subsets A and B ($A \cup B = V$, $A \cap B = \emptyset$), the cut between A and B is defined as the sum of the weights of edges that must be removed to separate A from B:

$$cut(A, B) = \Sigma\,(i \in A, j \in B)\; w(vi, vj)$$

The minimum cut criterion segments the graph by finding the partition (A, B) that minimizes cut(A, B). Efficient polynomial-time algorithms (e.g. the Ford–Fulkerson max-flow / min-cut algorithm) exist for computing the exact minimum cut given two terminal nodes.

Bias of the Minimum Cut Criterion

A well-known drawback of the plain minimum-cut criterion is that it is biased towards cutting off small, isolated sets of vertices, since the cut value grows with the number of edges crossing the boundary, not with the sizes of the two parts. In the worst case, minimum cut will separate a single outlier pixel from the rest of the image, which is rarely the desired segmentation.

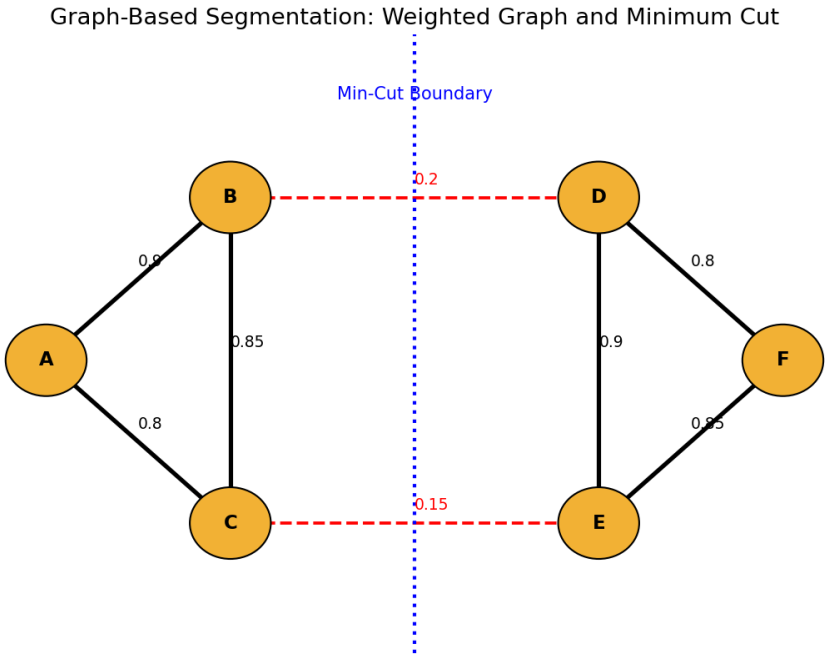


Fig. 4.6 — A weighted graph representing an image region; the minimum cut (blue dashed boundary) removes only the two lowest-weight edges to separate the graph into two segments.

4.5.2 Normalized Cuts (Ncut)

To overcome the small-set bias of the minimum cut, Shi and Malik proposed the normalized cut criterion, which normalizes the cut value by the total edge-weight connection of each partition to the entire graph, rather than looking at the cut in isolation:

$$Ncut(A, B) = cut(A, B) / assoc(A, V) + cut(A, B) / assoc(B, V)$$

where assoc(A, V) = Σ(i ∈ A, j ∈ V) w(vi, vj) is the total connection weight from set A to all vertices in the graph. This normalization penalizes partitions that isolate a small subset even if the cut value itself is small, because a small subset A will also have a small assoc(A, V), inflating the ratio.

Minimizing Ncut(A, B) exactly is an NP-hard combinatorial problem, but Shi and Malik showed that a good approximate solution can be obtained efficiently by relaxing the discrete partitioning problem into a continuous eigenvalue problem. Defining D as the diagonal degree matrix (with Dii = Σj w(vi, vj)) and W as the affinity/weight matrix, the second-smallest eigenvector of the generalized eigenvalue system:

$$(D - W)\, y = \lambda\, D\, y$$

provides a real-valued relaxation of the discrete partition indicator vector; thresholding this eigenvector (e.g. at zero, or at the value minimizing Ncut over all possible splits of the sorted values) yields the two-way segmentation. The procedure can be applied recursively to each resulting sub-region to obtain a segmentation into more than two parts.

Algorithm: Normalized Cuts Segmentation

- 1. Construct a weighted graph G = (V, E) from the image, with edge weights w(vi, vj) reflecting feature and spatial similarity.
- 2. Compute the affinity matrix W and the degree matrix D.
- 3. Solve the generalized eigenvalue problem (D − W) y = λ D y for the eigenvector y2 corresponding to the second-smallest eigenvalue.
- 4. Threshold y2 to bipartition the graph into two segments that (approximately) minimize Ncut.
- 5. Recursively apply steps 1–4 to each of the resulting sub-graphs until a stopping criterion (e.g. Ncut value exceeds a threshold, or a target number of segments is reached) is satisfied.

Note: Normalized cuts is one of the most frequently examined graph-based segmentation methods — remember that its key advantage over plain min-cut is the normalization term that discourages trivial small-set partitions.

4.6 The Hough Transform

Many objects in images are bounded, or partially bounded, by simple geometric shapes such as straight lines, circles, or ellipses (road lane markings, coins, wheels, the edges of buildings). Edge detectors produce a set of candidate edge pixels, but due to noise, occlusion, and broken contours, the edge points belonging to a single geometric shape are rarely perfectly connected. The Hough transform is a robust voting-based technique for detecting parametric shapes even when the supporting edge points are noisy, incomplete, or partly occluded.

4.6.1 Hough Transform for Lines

A line in image space can be written in the familiar slope-intercept form y = mx + c, but this representation cannot represent vertical lines (infinite slope) and behaves poorly numerically for near-vertical lines. Instead, the Hough transform commonly uses the normal (polar) representation of a line:

$$\rho = x \cdot \cos\theta + y \cdot \sin\theta$$

where ρ is the perpendicular distance from the origin to the line, and θ is the angle that this perpendicular makes with the x-axis. Every line in the image corresponds to exactly one point (ρ, θ) in this parameter space (also called Hough space).

Key Idea: Duality between Image Space and Parameter Space

- A single point (x0, y0) in image space, substituted into the line equation, generates a whole family of lines through it — this family traces out a sinusoidal curve ρ = x0 cos θ + y0 sin θ in (θ, ρ) parameter space.
- Collinear points in image space — that is, points lying on the same straight line — produce sinusoidal curves in parameter space that all intersect at one common point (θ0, ρ0), which are exactly the parameters of the line through those points.
- The Hough transform therefore converts the line-detection problem into a peak-detection problem: find the point(s) in (θ, ρ) space through which the greatest number of sinusoidal curves pass.

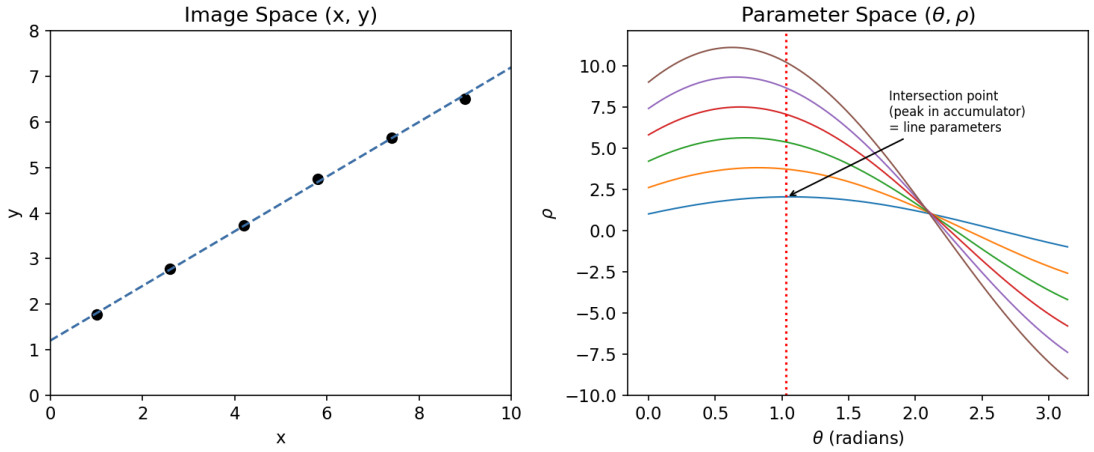


Fig. 4.7 — Left: collinear edge points in image space. Right: each point maps to a sinusoidal curve in (θ, ρ) parameter space; the curves intersect at the point representing the common line.

Hough Transform Algorithm

- 1. Quantize the parameter space (θ, ρ) into a discrete accumulator array A(θ, ρ), initialized to zero.

- 2. For every edge point (x, y) in the image, and for every discretized value of θ, compute ρ = x cos θ + y sin θ, and increment the corresponding accumulator cell: A(θ, ρ) ← A(θ, ρ) + 1.
- 3. After processing all edge points, search the accumulator array for local maxima (peaks). Each peak A(θ*, ρ*) that exceeds a threshold corresponds to a line detected in the image with parameters (θ*, ρ*).
- 4. Optionally, map the detected (θ*, ρ*) parameters back to image coordinates to draw or extract the actual line segment(s).

4.6.2 Hough Transform for Circles and the Generalized Hough Transform

The Hough transform extends naturally to any curve that can be described by a small number of parameters. A circle of radius r centred at (a, b) satisfies:

$$(x - a)^2 + (y - b)^2 = r^2$$

giving a three-dimensional parameter space (a, b, r); every edge point now votes for a surface (a cone) in this 3-D accumulator, and peaks correspond to detected circles. Because the accumulator grows in size and computational cost with the number of parameters, the Hough transform becomes progressively more expensive for shapes with more free parameters.

For shapes that cannot be described by a simple analytic equation, the Generalized Hough Transform (GHT) represents the shape by a lookup table (the R-table) that stores, for every possible edge orientation on the shape's boundary, the vector(s) from the boundary point to a fixed reference point. During detection, every edge point votes for the possible location(s) of the reference point implied by its orientation, and peaks in this vote space indicate the presence and position of the shape.

Advantages and Limitations of the Hough Transform

- Advantage: robust to noise and gaps, because it does not require an unbroken chain of edge points — any subset of points lying (approximately) on the shape contributes a vote.
- Advantage: can detect multiple instances of a shape simultaneously by locating multiple peaks in the accumulator.
- Limitation: computational and memory cost grows rapidly with the number of shape parameters (the "curse of dimensionality" of the accumulator array).
- Limitation: choice of accumulator quantization (bin size) trades off between detection accuracy and susceptibility to noise/split peaks.

Note: Be ready to derive ρ = x cos θ + y sin θ and explain, in words, why collinear image points map to concurrent sinusoids in Hough space — this is one of the most frequently asked derivations from this unit.

4.7 Fitting Lines

Once a set of edge points has been identified as (approximately) belonging to a straight line — whether via the Hough transform, edge linking, or clustering — it is usually necessary to fit an accurate line to those points. Line fitting is the process of estimating the parameters of the straight line that best matches a given set of 2-D points, typically by minimizing some measure of total fitting error.

4.7.1 Least Squares Line Fitting (Vertical / Ordinary Least Squares)

Given n data points (xi, yi), i = 1, …, n, ordinary least squares fits a line of the form y = mx + c by choosing the slope m and intercept c that minimize the sum of squared vertical distances between the observed points and the line:

$$E(m, c) = \Sigma i=1..n \, (yi - (m\, xi + c))^2$$

Setting the partial derivatives of E with respect to m and c to zero and solving the resulting normal equations yields the closed-form least squares estimates:

$$m = (n \Sigma xi\, yi - \Sigma xi \Sigma yi) / (n \Sigma xi^2 - (\Sigma xi)^2)$$

$$c = (\Sigma yi - m \Sigma xi) / n$$

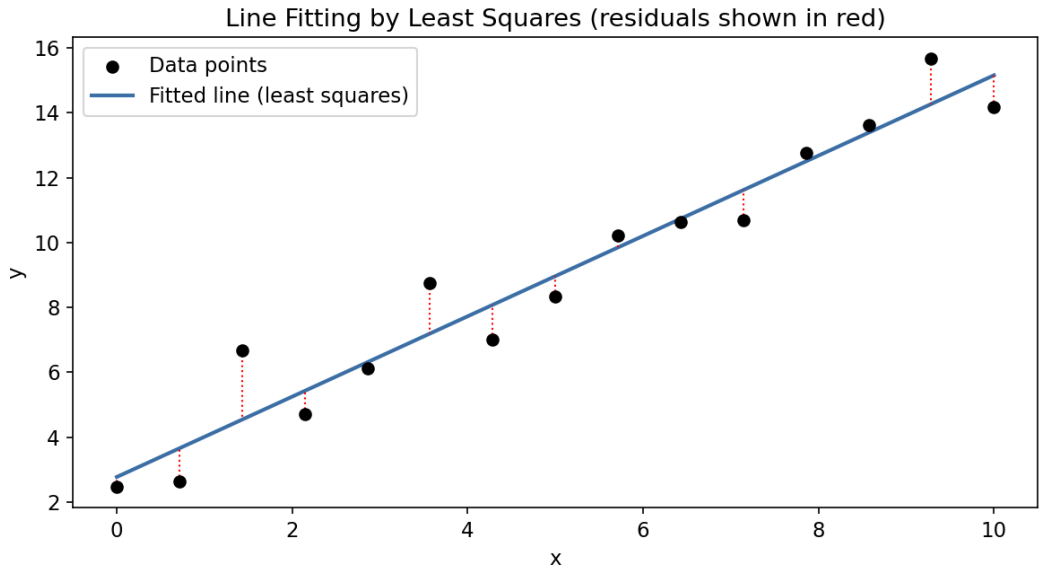


Fig. 4.8 — Least-squares line fit to a set of noisy points; red dotted segments show the vertical residual for each point.

Limitation of Ordinary Least Squares

Ordinary least squares minimizes only the vertical (y-direction) distance between points and the line. This makes the fit asymmetric — it performs poorly for near-vertical lines (where the model y = mx + c breaks down entirely) and is not invariant to a rotation of the coordinate axes.

4.7.2 Total Least Squares (Orthogonal Regression)

Total least squares fixes this asymmetry by minimizing the sum of squared perpendicular (orthogonal) distances from each point to the line, rather than vertical distances. Representing the line in normal form as x cos θ + y sin θ = ρ, the total least squares fit minimizes:

$$E(\theta, \rho) = \Sigma i=1..n \, (xi \cos \theta + yi \sin \theta - \rho)^2$$

The minimizing ρ is shown to equal $\bar{x} \cos \theta + \bar{y} \sin \theta$, where $(\bar{x}, \bar{y})$ is the centroid of the data points — that is, the fitted line must pass through the centroid of the points. Substituting this back gives θ in terms of the second-order central moments of the point set:

$$\tan(2\theta) = 2 \Sigma (xi - \bar{x})(yi - \bar{y}) / [\Sigma(xi - \bar{x})^2 - \Sigma(yi - \bar{y})^2]$$

Equivalently, total least squares can be computed by forming the 2×2 scatter (covariance) matrix of the centred data points and taking the eigenvector corresponding to the smallest eigenvalue as the direction perpendicular to the best-fit line; this is precisely the same computation used in Principal Component Analysis (PCA).

4.7.3 Robust Line Fitting

Both ordinary and total least squares are highly sensitive to outliers, because the squared-error criterion allows a single distant point to dominate the sum. Two standard remedies are used in computer vision:

(a) M-Estimators

M-estimators replace the squared-error term with a robust loss function ρ(·) that grows more slowly than a square for large residuals, thereby reducing the influence of outliers:

$$E = \Sigma i=1..n \, \rho(ri), \; ri = \text{residual of point } i \text{ from the fitted line}$$

Common choices for ρ(·) include the Huber loss (quadratic for small residuals, linear for large ones) and the Tukey biweight (which caps the influence of very large residuals to zero beyond a threshold).

(b) RANSAC (Random Sample Consensus)

RANSAC is a general-purpose, highly effective robust-fitting framework that tolerates a very large proportion of outliers. For line fitting, the RANSAC algorithm proceeds as follows:

- 1. Randomly select the minimum number of points needed to define a line (2 points).
- 2. Fit a candidate line through the selected points.
- 3. Count the number of inliers — all remaining data points that lie within a small distance threshold of the candidate line.
- 4. Repeat steps 1–3 for a large number of trials, and keep the candidate line that attracted the largest number of inliers.

- 5. (Optional refinement) Re-fit the final line to all its inliers using ordinary or total least squares for improved accuracy.

The number of trials N required to guarantee, with probability p, that at least one sample of size s is free of outliers, given an outlier proportion e, is given by:

$N = log(1 - p) / log(1 - (1 - e)^s)$

> **Note:** Total least squares (orthogonal regression) is the mathematically preferred line-fitting method in computer vision because it treats x and y symmetrically and is invariant to rotation of the image axes — this is a common exam distinction to draw against ordinary least squares.

4.8 Fitting Curves

Many objects of interest are bounded by curves more general than straight lines — circles, ellipses, or arbitrary smooth contours. Curve fitting generalizes the line-fitting methods of Section 4.7 to these richer shape models.

4.8.1 Fitting a Circle

A circle with centre (a, b) and radius r satisfies the implicit equation:

$(x - a)^2 + (y - b)^2 - r^2 = 0$

Given a set of n candidate points (xi, yi) believed to lie on a circle, the parameters (a, b, r) can be estimated by minimizing the sum of squared algebraic (or geometric) distances of the points from the circle:

$E(a, b, r) = \Sigma i=1..n \, [(xi - a)^2 + (yi - b)^2 - r^2]^2$

This algebraic form can be rearranged into a linear least-squares problem in the unknowns (a, b, c) where $c = a^2 + b^2 - r^2$, since expanding the square yields a linear equation in a, b, and c for each data point — allowing an efficient closed-form solution, though it is biased in the presence of noise compared with minimizing true geometric (perpendicular) distance.

4.8.2 Fitting General Conics

Lines and circles are special cases of the general conic section, described implicitly by:

$A x^2 + B xy + C y^2 + D x + E y + F = 0$

Given a set of points, the conic parameters (A, B, C, D, E, F) — defined up to an overall scale — can be estimated by minimizing the sum of squared algebraic distances $\Sigma i \, (A \, xi^2 + B \, xi \, yi + C \, yi^2 + D \, xi + E \, yi + F)^2$, subject to a normalization constraint (e.g. $A^2 + B^2 + C^2 + D^2 + E^2 + F^2 = 1$) to avoid the trivial all-zero solution; this reduces to an eigenvector problem, solvable by Singular Value Decomposition (SVD).

4.8.3 Robust Curve Fitting

As with lines, curve fitting is highly sensitive to outlier points that do not actually belong to the curve. The same robust strategies used for line fitting apply directly to curves:

- M-estimators — replace the squared algebraic/geometric distance with a robust loss function that limits the influence of outliers.
- RANSAC — repeatedly sample the minimum number of points required to define the curve (e.g. 3 points determine a circle), fit a candidate curve, count inliers within a distance threshold, and retain the candidate with the largest inlier support, followed by a final refit on the inlier set.

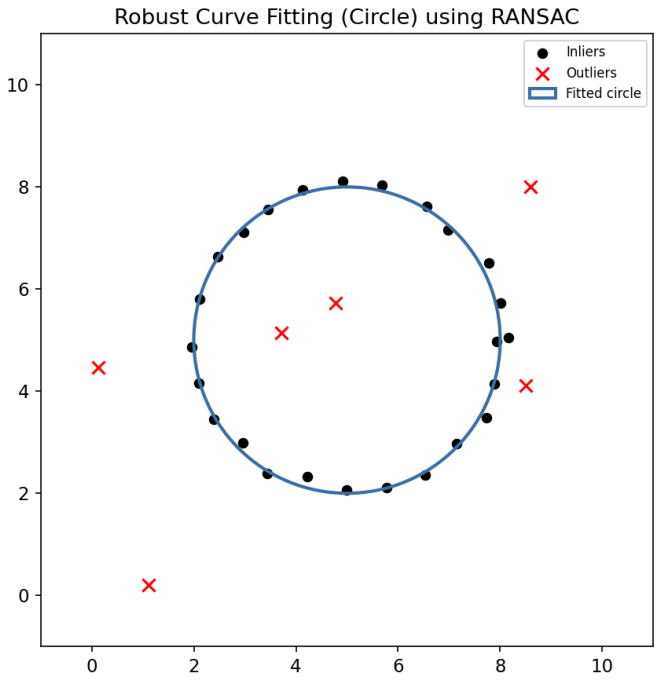


Fig. 4.9 — Robust circle fitting using RANSAC: black points are inliers used for the final fit; red crosses are outliers correctly excluded from the fitted circle.

4.8.4 Shape from Fitted Curves

Once a reliable curve (or set of curves) has been fitted, higher-level geometric properties — object boundaries, symmetry axes, vanishing points, or 3-D pose (for known-shape objects such as coins or wheels imaged under perspective projection) — can be recovered from the fitted parameters. This links curve fitting, discussed in this final section of Unit IV, to the recognition-by-relations and shape-analysis techniques developed later in the course.

> **Note:** Whenever a fitting problem can be written as a linear least-squares problem (as circle fitting can, via the substitution $c = a^2 + b^2 - r^2$), prefer the closed-form solution

for speed, but remember it minimizes algebraic, not geometric, error — mention this trade-off in descriptive answers.

4.11 Solved Examples

The following worked examples illustrate how the formulae developed in this unit are applied to small numerical problems — useful both for building intuition and for examination practice.

Example 1 — One Iteration of K-Means Clustering

Consider the 1-D data points {2, 4, 10, 12, 3, 20} to be grouped into K = 2 clusters. Let the initial centroids be chosen as $\mu 1 = 2$ and $\mu 2 = 4$.

Step 1: Assignment

Compute the distance of every point from each centroid and assign it to the nearer one:

- Point 2: $|2-2| = 0$, $|2-4| = 2$ → assigned to cluster 1
- Point 4: $|4-2| = 2$, $|4-4| = 0$ → assigned to cluster 2
- Point 10: $|10-2| = 8$, $|10-4| = 6$ → assigned to cluster 2
- Point 12: $|12-2| = 10$, $|12-4| = 8$ → assigned to cluster 2
- Point 3: $|3-2| = 1$, $|3-4| = 1$ → assigned to cluster 1 (tie broken arbitrarily)
- Point 20: $|20-2| = 18$, $|20-4| = 16$ → assigned to cluster 2

Step 2: Update

Recompute each centroid as the mean of the points currently assigned to it:

$$\mu 1 = (2 + 3) / 2 = 2.5 \quad \mu 2 = (4 + 10 + 12 + 20) / 4 = 11.5$$

The assignment and update steps are then repeated with the new centroids $\mu 1 = 2.5$ and $\mu 2 = 11.5$ until the cluster memberships stop changing. In this example, a second iteration reassigns the same points to the same two clusters, so the algorithm has converged after just one update, giving the final clusters {2, 3} and {4, 10, 12, 20}.

Example 2 — Hough Transform Vote for a Single Edge Point

Suppose an edge point is located at (x, y) = (4, 3). Using the normal form $\rho = x \cos\theta + y \sin\theta$, the votes cast by this point into the accumulator, sampled at $\theta = 0°, 30°, 60°, 90°$, are:

$$\theta = 0°: \rho = 4\cos 0° + 3\sin 0° = 4(1) + 3(0) = 4$$

$$\theta = 30°: \rho = 4\cos 30° + 3\sin 30° = 4(0.866) + 3(0.5) = 4.96$$

$$\theta = 60°: \rho = 4\cos 60° + 3\sin 60° = 4(0.5) + 3(0.866) = 4.60$$

θ = 90°: ρ = 4 cos90° + 3 sin90° = 4(0) + 3(1) = 3

Each computed (θ, ρ) pair increments the corresponding accumulator cell by one vote. If a second edge point, say (8, 1), is also processed and happens to produce the same (θ, ρ) pair as (4, 3) at some angle θ*, the accumulator cell at (θ*, ρ*) accumulates two votes — indicating that both points are consistent with a single straight line having those parameters. As more collinear points are processed, the accumulator cell corresponding to the true line parameters accumulates the highest vote count and stands out as a clear peak.

Example 3 — Least Squares Line Fit

Fit a line y = mx + c to the four points (1, 2), (2, 3), (3, 5), (4, 6) using ordinary least squares.

First compute the required sums:

n = 4, Σx = 10, Σy = 16, Σxy = (1·2 + 2·3 + 3·5 + 4·6) = 2+6+15+24 = 47, Σx² = 1+4+9+16 = 30

Substituting into the normal-equation formulae:

m = (nΣxy − ΣxΣy) / (nΣx² − (Σx)²) = (4·47 − 10·16) / (4·30 − 10²) = (188 − 160) / (120 − 100) = 28/20 = 1.4

c = (Σy − mΣx) / n = (16 − 1.4·10) / 4 = (16 − 14) / 4 = 0.5

The least-squares fitted line is therefore y = 1.4x + 0.5. This matches the type of computation shown graphically in Fig. 4.8, and the same normal-equation approach extends directly to larger data sets.

Example 4 — Normalized Cut on a Small Graph

Consider a graph split into sets A and B with cut(A, B) = 5, assoc(A, V) = 20, and assoc(B, V) = 15. The normalized cut value is:

Ncut(A, B) = 5/20 + 5/15 = 0.25 + 0.333 = 0.583

If, instead, a different partition isolates a very small set A′ with cut(A′, B′) = 2 but assoc(A′, V) = 2 (A′ contains almost no other connections), then:

Ncut(A', B') = 2/2 + 2/33 = 1.0 + 0.061 = 1.061

Even though the raw cut value (2) for the second partition is smaller than the first (5), its normalized cut value (1.061) is much larger, correctly penalizing the trivial small-set partition and confirming that the first, more balanced partition is preferable under the Ncut criterion.

> **Note:** Worked numerical examples of this kind — particularly K-means iterations, Hough-space voting, and least-squares line fitting — are common short-answer and problem-type examination questions for this unit.

4.9 Summary of Unit IV

- Segmentation partitions an image into homogeneous, disjoint regions; it is inherently perceptual and task-dependent, motivating the study of human grouping (Gestalt) principles.
- Gestalt laws — proximity, similarity, closure, continuation, common fate, and common region — describe how the human visual system groups elements, and inspire the design of similarity/affinity measures in algorithms.
- Shot boundary detection segments video in time using frame-difference or histogram-difference metrics compared against a threshold; twin-comparison handles gradual transitions.
- Background subtraction segments each frame into foreground and background by comparing the current frame with an adaptively maintained background model (running average, median, or Gaussian mixture models).
- K-means clustering partitions pixels/feature vectors into K clusters by iteratively minimizing the within-cluster sum-of-squares objective J; EM/mixture-of-Gaussians and mean shift provide more flexible probabilistic and non-parametric alternatives.
- Graph-theoretic segmentation represents an image as a weighted graph and partitions it via minimum cut or, preferably, normalized cuts, which corrects the small-set bias of plain minimum cut through a normalization term.
- The Hough transform detects parametric shapes (lines, circles, arbitrary shapes via the Generalized Hough Transform) by having each edge point vote in a parameter (accumulator) space, converting shape detection into peak detection.
- Line fitting uses ordinary least squares (vertical error), total least squares / orthogonal regression (perpendicular error, rotation-invariant), and robust methods (M-estimators, RANSAC) to handle outliers.
- Curve fitting generalizes line fitting to circles and general conics using implicit equations solved by linear least squares or SVD, with RANSAC providing robustness to outliers.

4.10 Review Questions

- 1. Define image segmentation formally in terms of a partition of the image domain, and state the two conditions a valid segmentation partition must satisfy.
- 2. List and explain, with a diagram, the six major Gestalt grouping laws relevant to computer vision.
- 3. Explain the frame-differencing method for shot boundary detection. Why is histogram-based differencing generally preferred over pixel-based differencing?
- 4. Describe the twin-comparison method for detecting gradual transitions in video.

- 5. Explain background subtraction using an adaptively updated background model. Derive the running-average update equation.
- 6. State the objective function minimized by K-means clustering, and write out the complete K-means algorithm.
- 7. Compare K-means clustering with Expectation-Maximization clustering using Gaussian mixture models.
- 8. Define the minimum cut of a graph partition, and explain why the plain minimum cut criterion is biased towards isolating small sets of vertices.
- 9. Define the normalized cut (Ncut) criterion and explain how it overcomes the bias of the minimum cut. Outline the eigenvector-based algorithm used to approximate the optimal normalized cut.
- 10. Derive the polar (normal) form of a line, $\rho = x \cos \theta + y \sin \theta$, and explain how the Hough transform uses this representation to detect lines robustly.
- 11. Describe the Hough transform algorithm for detecting circles, and explain how the Generalized Hough Transform extends the idea to arbitrary shapes.
- 12. Derive the normal equations for ordinary least-squares line fitting, and explain its main limitation.
- 13. Explain total least squares (orthogonal regression) for line fitting and why it is preferred over ordinary least squares in computer vision.
- 14. Explain the RANSAC algorithm for robust line/curve fitting, and derive the expression for the required number of trials N.
- 15. Derive the implicit equation of a circle and explain how circle fitting can be posed as a linear least-squares problem.

UNIT – V

RECOGNITION BY RELATIONS BETWEEN TEMPLATES

Topics Covered:

Finding Objects by Voting on Relations between Templates

Relational Reasoning Using Probabilistic Models and Search

Using Classifiers to Prune Search

Hidden Markov Models

Application: HMM and Sign Language Understanding

Finding People with HMM

Audio: Physics of Sound, Physiology of Hearing, Auditory Perception & Rendering

Textbook: David A. Forsyth, Jean Ponce, "Computer Vision – A Modern Approach", PHI, 2003 (Audio topics from Text Book 2)

5.1 Introduction: Recognition by Relations between Templates

Earlier units treated recognition largely in terms of matching individual templates or classifying individual feature vectors. However, most real objects are not rigid, single-piece patterns — they are composed of multiple parts held together in a characteristic spatial arrangement. A face, for example, consists of eyes, a nose, and a mouth held in a fairly consistent geometric configuration; a walking person consists of a torso, head, and four limbs connected by joints that move in constrained ways.

Recognition by relations between templates is the approach of modelling an object as a collection of simple part-templates together with the geometric or temporal relations that must hold between them. Rather than searching for one large, complex template that matches an entire object exactly — which is brittle under pose change, articulation, and occlusion — the vision system searches for the individual, simpler part-templates and then checks whether the relations expected between them are satisfied. This unit studies three complementary strategies for this kind of relational recognition:

- Voting schemes, in which each matched part casts a vote for the location/pose of the whole object, and consistent votes reinforce each other (Section 5.2).
- Probabilistic relational reasoning combined with search, in which the correspondence between image features and model parts is treated as a search problem guided by a probabilistic scoring function (Section 5.3).
- Classifier-based pruning, in which fast classifiers eliminate implausible correspondences early, making the search computationally tractable (Section 5.4).

The unit then studies Hidden Markov Models (HMMs) in depth (Section 5.5) — a probabilistic sequence model that is naturally suited to recognizing patterns that unfold over time or over an ordered sequence of parts, such as gestures, sign language, or human body configurations (Sections 5.6 and 5.7). The unit closes with an introduction to audio: the physics of sound, the physiology of hearing, auditory perception, and auditory rendering, which parallel the visual material by studying how meaningful auditory "objects" are represented, transmitted, and perceived (Sections 5.8–5.11).

> **Note:** The unifying theme of this unit is structural / relational matching: complex, deformable, or temporally extended patterns are recognized by combining evidence from simple parts using geometric, probabilistic, or sequential relations, rather than by matching a single rigid template.

5.2 Finding Objects by Voting on Relations between Templates

Voting-based recognition generalizes the Hough transform idea (Unit IV) from simple geometric shapes to entire objects composed of multiple parts. The key insight is that if a part template matches the image at some location, and we know the expected geometric offset of that part from the object's reference point (e.g. its centre), then the matched part can "vote" for where the object's reference point must be. When several independent parts all vote for approximately the same reference location, this consensus is strong evidence that the whole object is present at that location.

5.2.1 The Voting Procedure

Let an object model consist of a set of part templates T1, T2, …, Tm, and let each part Ti have a learned offset vector di from the object's reference point (established during training, e.g. by recording the displacement of each part from the object centre in training images). Given a new image, the voting procedure is:

- 1. Match each part template Ti against the image (e.g. by normalized correlation, or by matching local features), producing a set of candidate detections, each with an image location pk and a confidence score.
- 2. For every candidate detection of part Ti at location pk, cast a vote for the object reference location:

$$v = pk - di$$

into an accumulator array indexed by candidate object-reference locations (and, if scale or orientation is also being searched, by scale/orientation as well).

- 3. After all parts have voted, search the accumulator for peaks. A peak that has received votes from a sufficient number of distinct parts, exceeding a threshold, is declared a detected object instance at that location.

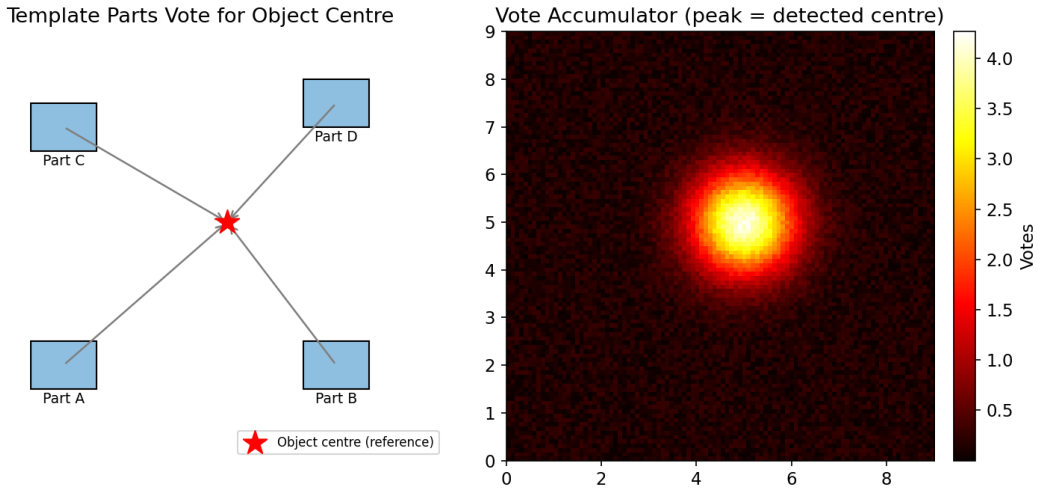


Fig. 5.1 — Left: template parts vote for the object's reference location using their learned offsets. Right: the resulting vote accumulator, with a clear peak marking the detected object centre.

5.2.2 Why Voting Is Robust

- Occlusion tolerance: even if some parts are occluded or fail to match, the remaining visible parts can still generate a strong peak, since votes accumulate independently.
- Noise tolerance: spurious false matches to background clutter cast votes that are scattered randomly across the accumulator, and therefore rarely reinforce each other, whereas true part matches consistently vote for the same location.
- Simplicity: each part-detection step is independent and can be run in parallel; combining evidence is a simple accumulation step, exactly as in the Hough transform.

5.2.3 Generalized Hough Transform for Object Detection

This voting procedure is, in fact, a direct generalization of the Generalized Hough Transform (GHT) introduced in Unit IV for arbitrary shape detection. There, edge points with a given local orientation voted for a reference point using a precomputed R-table; here, the "edge points" are replaced by matched part templates, and the R-table is replaced by the learned set of offsets {di}. Both techniques convert a difficult correspondence/detection problem into a straightforward peak-detection problem in an accumulator space.

5.2.4 Limitations

- The accumulator space grows in dimensionality if pose parameters such as scale and rotation must also be searched, increasing memory and computation (the same curse of dimensionality noted for the Hough transform).
- Voting alone ignores higher-order relations between parts (e.g. whether the relative angle between two parts is plausible) — it only checks that each part is consistent with a common reference point, not that parts are mutually consistent with each other. This motivates the richer relational reasoning discussed next.

Note: Voting-based object detection is often called the 'implicit shape model' approach in later literature; remember its three defining steps — part matching, offset-based voting, and peak detection — as they mirror the Hough transform almost exactly.

5.3 Relational Reasoning Using Probabilistic Models and Search

Voting handles independent part-to-reference relations well, but many objects require reasoning about relations between pairs (or larger groups) of parts directly — for instance, the relative angle between the upper and lower arm at the elbow, or the expected distance between the two eyes relative to the width of the face. This richer form of reasoning is naturally formulated as a search over correspondences between image features and model parts, guided by a probabilistic scoring function that captures both individual part appearance and pairwise geometric relations.

5.3.1 Formulating Correspondence as Search

Let the object model consist of parts P1, …, Pn, and let the image contain a set of candidate feature detections F1, …, Fm (typically m ≫ n, since many candidate detections are false positives from clutter). A correspondence (or matching) assigns each model part Pi to some image feature Fσ(i), or marks it as unmatched (occluded/missing). The space of all possible correspondences σ can be organized as a search tree: at the first level, decide the match for P1; at the second level, decide the match for P2, and so on.

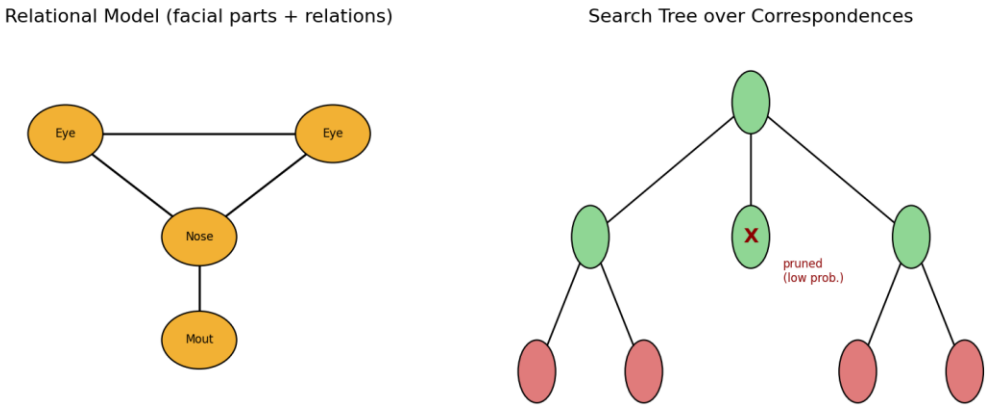


Fig. 5.2 — Left: a relational model of facial parts with pairwise geometric relations. Right: the corresponding search tree over part-to-feature correspondences, with an implausible branch pruned.

5.3.2 Probabilistic Scoring of a Correspondence

A natural way to score a candidate correspondence σ is to define a joint probability (or, equivalently, an energy/cost function) that factors into unary terms (how well each matched feature resembles the expected appearance of its part) and binary/pairwise terms (how well the geometric relation between each pair of matched parts agrees with the model's expected relation):

$$P(\sigma \mid Image) \propto \prod_{i=1..n} \varphi(Pi, F\sigma(i)) \cdot \prod_{(i,j) \in E} \psi(Pi, Pj, F\sigma(i), F\sigma(j))$$

where φ(Pi, Fσ(i)) is the unary appearance compatibility (e.g. a template-matching or classifier score for part Pi at feature Fσ(i)), ψ(Pi, Pj, Fσ(i), Fσ(j)) is the pairwise relational compatibility (e.g. a Gaussian penalty on the deviation of the observed relative position/angle/scale between Fσ(i) and Fσ(j) from the model's learned relation), and E is the set of part pairs for which a relation is modelled (often just the edges of a tree-structured or star-structured model, for computational tractability).

Recognition then reduces to searching for the correspondence σ* that maximizes this probability (or minimizes the corresponding negative-log-probability "energy"):

$\sigma^* = arg\ max_\sigma\ P(\sigma \mid Image)$

5.3.3 Search Strategies

- Exhaustive search — try every possible assignment of parts to features; correct but combinatorially explosive, since the number of possible correspondences grows as $O(m^n)$.
- Branch-and-bound search — extend partial correspondences level by level in the search tree, and prune (discard) any partial correspondence whose best-possible completion score cannot exceed the score of the best complete correspondence found so far; this can be dramatically faster than exhaustive search while still guaranteeing the optimal solution.
- Greedy / best-first search — always extend the most promising partial correspondence first, according to the probabilistic score; fast, but does not guarantee the globally optimal correspondence.
- Dynamic programming — when the relational model is restricted to a tree or a chain structure (as with many articulated body or gesture models), the optimal correspondence can be found efficiently and exactly using dynamic programming, exploiting the fact that the joint probability factors along the tree/chain.

5.3.4 Tree-Structured and Star-Structured Models

Because a fully connected relational graph (with a pairwise term between every pair of parts) makes exact search intractable for anything but very small n, practical relational models typically restrict E to a tree (each part has one parent, e.g. a kinematic chain for a body) or a star (all parts related only to a single central reference part, e.g. the object centroid). Both restrictions permit exact and efficient inference (dynamic programming for a tree, direct accumulation for a star), at the cost of ignoring some higher-order relations that a fully connected model could represent.

> **Note:** The probabilistic formulation in this section is precisely the pictorial structures model widely used in articulated pose estimation — remember that restricting the relational graph to a tree is what makes exact inference by dynamic programming possible.

5.4 Using Classifiers to Prune Search

Even with branch-and-bound or dynamic programming, relational search can remain expensive if the number of candidate image features m is large, as is typical in cluttered real-world images. A very effective practical strategy is to use fast, simple classifiers to eliminate large numbers of implausible partial correspondences before any expensive relational scoring is performed — turning an otherwise intractable search into a tractable one.

5.4.1 The Role of a Pruning Classifier

A pruning classifier is typically a lightweight binary classifier (for example, a decision stump, a small decision tree, or a cascade of simple feature tests, as used in the Viola–Jones face detector) trained to quickly reject image locations/features that are clearly inconsistent with a given model part, without needing to evaluate the full, expensive appearance or relational score. Because the classifier only needs to be fast and have a low false-negative rate (it must not throw away correct matches), while a higher false-positive rate is tolerable (surviving false positives will be filtered out later by the full relational score), simple and efficient classifiers are well suited to this role.

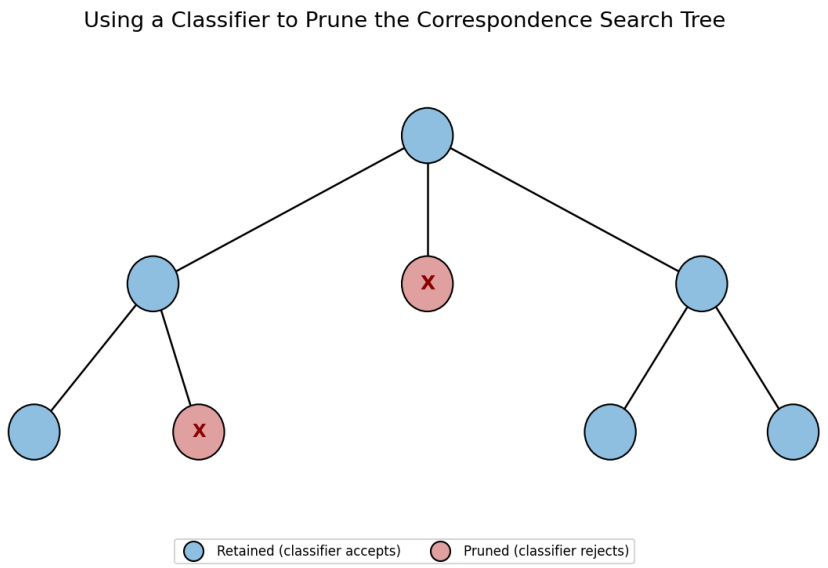


Fig. 5.3 — A classifier prunes implausible branches of the correspondence search tree early (marked with ✕), leaving only a small number of promising branches for expensive relational scoring.

5.4.2 Cascade of Classifiers

A particularly effective architecture is the classifier cascade: a sequence of classifiers of increasing complexity and discriminative power is applied in order. Early stages use very cheap features and reject the vast majority of clearly implausible candidates (e.g. regions with no edges at all cannot contain a face). Only candidates that pass an early stage are forwarded to the next, more expensive and more discriminative stage. Because most candidates are rejected at the earliest, cheapest stages, the average computational cost per image location is very low, even though later stages may be individually expensive.

If stage k of the cascade has a false-positive rate fk and a detection (true-positive) rate dk, and there are K stages, the overall detection performance of the cascade is:

$D = \prod k=1..K\ dk\ F = \prod k=1..K\ fk$

This shows why even stages with a fairly high individual false-positive rate can be combined into a cascade with an overall very low false-positive rate F, provided that each stage's detection rate dk is kept close to 1 so that the overall detection rate D does not degrade too much.

5.4.3 Integrating Pruning with Relational Search

In a relational search framework (Section 5.3), classifier-based pruning is naturally applied at each level of the search tree: before extending a partial correspondence by matching the next model part Pi to a candidate feature Fk, a fast classifier first tests whether Fk is even a plausible instance of part Pi's appearance, and/or whether the candidate is geometrically plausible given the parts already matched. Only candidates surviving this fast test are explored further using the full, more expensive relational scoring function φ, ψ described in Section 5.3.2.

5.4.4 Benefits and Trade-offs

- Benefit: dramatically reduces the effective branching factor of the search tree, since most candidate features at each level are eliminated cheaply.
- Benefit: allows relational recognition systems to scale to real-time or near-real-time operation on cluttered, high-resolution images.
- Trade-off: an overly aggressive pruning classifier risks discarding correct matches (increasing false negatives), so pruning thresholds must be tuned conservatively, especially in early stages.
- Trade-off: training effective cascades or pruning classifiers requires substantial labelled training data covering the range of appearance variation expected for each part.

> **Note:** The classifier-cascade idea (fast, low-cost stages first; expensive, discriminative stages last) is one of the most important practical engineering techniques in real-time detection, and its detection/false-positive rate formulae are common examination items.

5.5 Hidden Markov Models (HMM)

Many recognition problems involve patterns that unfold over an ordered sequence — over time (a spoken word, a hand gesture, a walking gait) or over an ordered set of parts (the sequence of joints along a limb). A Hidden Markov Model (HMM) is a probabilistic model perfectly suited to this setting: it assumes that the system being observed moves through a sequence of hidden (unobserved) states, and that at each time step the current hidden state produces an observable output according to some probability distribution. The word 'hidden' refers to the fact that only the outputs are directly observed — the underlying state sequence must be inferred.

5.5.1 Formal Definition and Elements of an HMM

An HMM is completely specified by the tuple λ = (A, B, π), defined over:

- A finite set of N hidden states, S = {S1, S2, …, SN}.
- A finite set of possible observation symbols, V = {v1, v2, …, vM} (for discrete HMMs; continuous HMMs instead use continuous emission densities such as Gaussian mixtures).
- The state transition probability matrix A = {aij}, where aij = P(qt+1 = Sj | qt = Si) is the probability of moving to state Sj at time t+1 given that the system was in state Si at time t. Each row of A sums to 1.
- The observation (emission) probability distribution B = {bj(k)}, where bj(k) = P(ot = vk | qt = Sj) is the probability of emitting symbol vk while in state Sj.
- The initial state distribution π = {πi}, where πi = P(q1 = Si) is the probability that the model starts in state Si.

λ = (A, B, π)

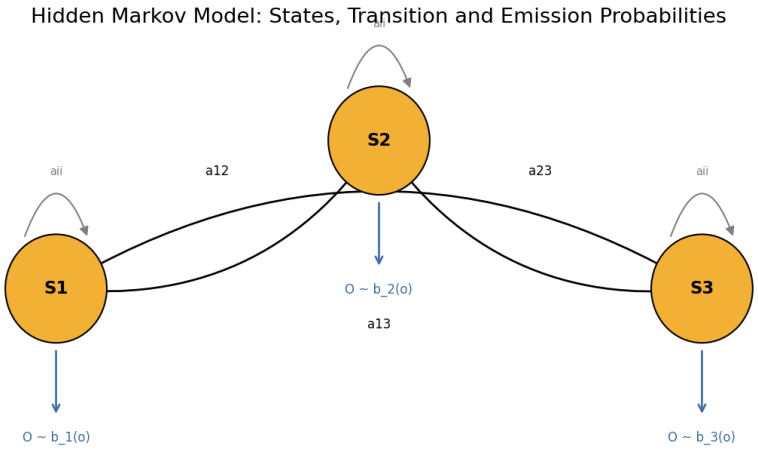


Fig. 5.4 — An HMM with three states, showing transition probabilities aij between states and emission probabilities bj(o) governing the observation generated by each state.

5.5.2 The Markov Assumption and Output Independence Assumption

An HMM relies on two simplifying assumptions that make inference and learning tractable:

- First-order Markov assumption: the probability of transitioning to the next state depends only on the current state, not on the entire history of previous states: P(qt+1 | qt, qt−1, …, q1) = P(qt+1 | qt).
- Output independence assumption: the probability of the current observation depends only on the current state, not on any previous states or observations: P(ot | qt, qt−1, …, ot−1, …) = P(ot | qt).

5.5.3 The Three Canonical Problems of HMMs

Given the definition of an HMM, three fundamental computational problems arise, each with an efficient dynamic-programming solution:

Problem 1 — Evaluation: The Forward Algorithm

Given a model λ and an observation sequence O = (o1, o2, …, oT), compute the probability that the model generated this sequence, P(O | λ). A naive computation would sum over all NT possible hidden state sequences, which is computationally infeasible for even moderate T. Instead, the forward algorithm defines the forward variable:

$\alpha t(j) = P(o1, o2, ..., ot, qt = Sj \mid \lambda)$

computed efficiently by the recursion:

$\alpha 1(j) = \pi j \; bj(o1)$

$\alpha t+1(j) = [\, \Sigma i=1..N \; \alpha t(i) \; aij \,] \; bj(ot+1)$

$P(O \mid \lambda) = \Sigma j=1..N \; \alpha T(j)$

This recursion reduces the computational complexity from exponential, O(NT · T), to a manageable O(N²T), by reusing partial computations, exactly as illustrated by the trellis structure below.

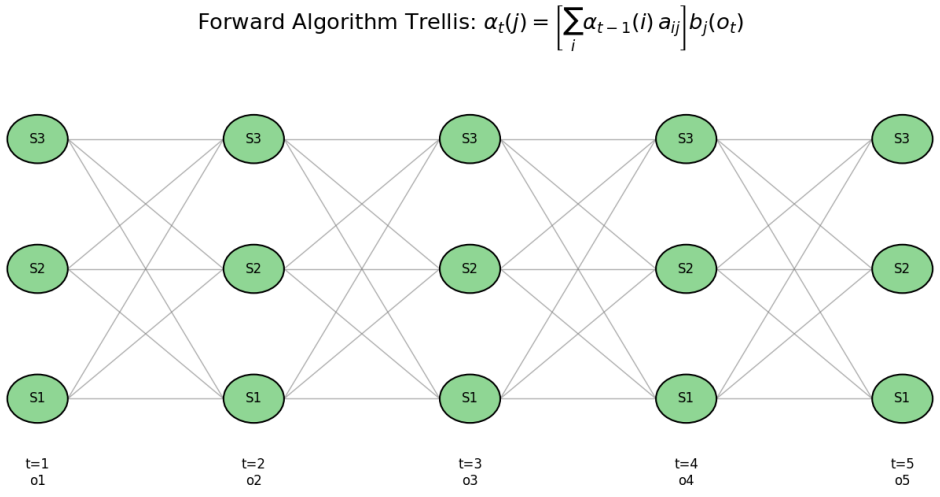


Fig. 5.5 — The forward algorithm trellis: each node αt(j) is computed from all αt−1(i) at the previous time step, giving an efficient O(N²T) evaluation.

Problem 2 — Decoding: The Viterbi Algorithm

Given a model λ and an observation sequence O, find the single most likely hidden state sequence Q* = (q1, q2, …, qT) that produced O. This is solved by the Viterbi algorithm, which

is structurally identical to the forward algorithm except that the summation over previous states is replaced by a maximization, since we now want the single best path rather than the total probability over all paths:

$\delta 1(j) = \pi j \; bj(o1)$

$\delta t(j) = max \; i \; [\; \delta t-1(i) \; aij \;] \; bj(ot)$

At each step, the algorithm also records the arg max, i.e. the state $\psi t(j)$ that achieved the maximum, so that after reaching t = T the best final state can be found and the optimal state sequence recovered by backtracking through the stored $\psi t(j)$ pointers:

$qT^* = arg \; maxj \; \delta T(j), \; qt^* = \psi t+1(qt+1^*) \; for \; t = T-1, \; ..., \; 1$

Problem 3 — Learning: The Baum–Welch (EM) Algorithm

Given only a set of observation sequences (and the number of states N), estimate the model parameters $\lambda = (A, B, \pi)$ that maximize the likelihood of the observed data. Since the hidden state sequence is unknown, this is solved using the Baum–Welch algorithm, a special case of the Expectation–Maximization (EM) algorithm for HMMs. Defining the backward variable analogously to the forward variable:

$\beta t(i) = P(ot+1, \; ot+2, \; ..., \; oT \mid qt = Si, \; \lambda)$

the probability of being in state Si at time t and state Sj at time t+1, given the current model and the full observation sequence, is:

$\xi t(i, \; j) = [\; \alpha t(i) \; aij \; bj(ot+1) \; \beta t+1(j) \;] \; / \; P(O \mid \lambda)$

and the probability of being in state Si at time t is $\gamma t(i) = \Sigma j \; \xi t(i, j)$. These quantities are used in the E-step; the M-step then re-estimates the parameters as:

$\hat{\pi i} = \gamma 1(i)$

$\hat{a} \; ij = (\; \Sigma t=1..T-1 \; \xi t(i,j) \;) \; / \; (\; \Sigma t=1..T-1 \; \gamma t(i) \;)$

$\hat{b} \; j(k) = (\; \Sigma t: \; ot=vk \; \gamma t(j) \;) \; / \; (\; \Sigma t=1..T \; \gamma t(j) \;)$

These E-step and M-step computations are repeated until the model parameters converge, i.e. until the increase in the observation-sequence likelihood $P(O \mid \lambda)$ between successive iterations falls below a small threshold.

5.5.4 Summary Table of the Three Problems

- Evaluation — "How well does model λ explain observation sequence O?" → solved by the Forward Algorithm → complexity $O(N^2T)$.
- Decoding — "What is the most likely hidden state sequence given O?" → solved by the Viterbi Algorithm → complexity $O(N^2T)$.

- Learning — "What model parameters λ best explain a set of training sequences?" → solved by the Baum–Welch (EM) Algorithm → iterative, each iteration $O(N^2T)$.

Note: These three problems, and the fact that each is solved efficiently using dynamic programming (forward, Viterbi) or EM (Baum-Welch) rather than brute-force enumeration over NT sequences, are the single most important set of facts to remember from this section.

5.6 Application: HMM and Sign Language Understanding

Sign language recognition is a natural and widely studied application of HMMs, because a signed word or phrase is exactly the kind of temporally structured pattern HMMs are designed to model: it consists of a sequence of hand shapes, positions, and movements that unfold in a characteristic order, with natural variation in speed and exact trajectory between different signers or repetitions.

5.6.1 Feature Extraction for Sign Language

Before an HMM can be applied, each video frame of the signer must be converted into an observation vector ot capturing the information relevant to the sign being performed. Commonly used features include:

- Hand position (x, y, and sometimes depth z) relative to the body, often tracked using colour, motion, or skeleton-tracking cues.
- Hand shape descriptors (e.g. the outline or a shape-context descriptor of the segmented hand region, encoding whether the hand is open, closed, pointing, etc.).
- Hand orientation and velocity, which are especially important for distinguishing signs that share the same hand shape but differ in motion direction.
- For two-handed signs, separate feature vectors for the dominant and non-dominant hand, sometimes modelled with two coupled HMMs.

5.6.2 Modelling a Sign as an HMM

Each distinct sign (or sub-word unit) w is modelled by its own HMM $\lambda w = (Aw, Bw, \pi w)$, trained on multiple example videos of that sign being performed. The hidden states of the HMM correspond to sub-gesture phases of the sign — for example, the preparatory hand position, the characteristic movement, and the final hold position — while the emission probabilities bj(ot) model the expected range of feature vectors observed during each phase.

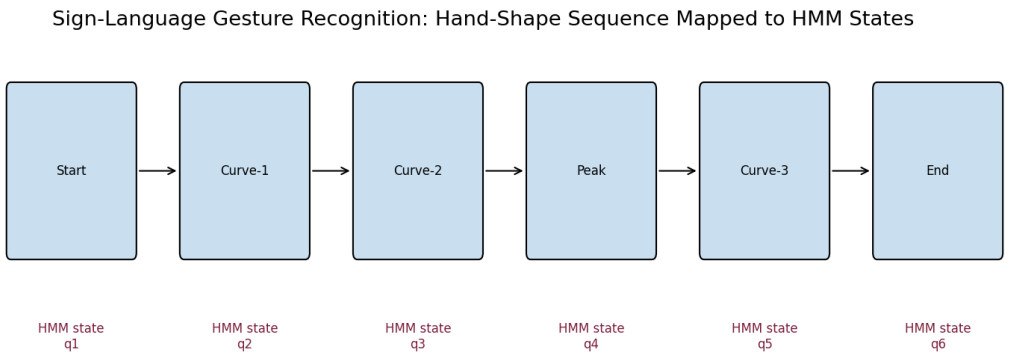


Fig. 5.6 — A signed gesture is segmented into a sequence of hand-shape/position observations, each generated by a corresponding hidden state of the sign's HMM.

5.6.3 Recognition of an Unknown Sign

Given an unknown video sequence, producing an observation sequence O, recognition proceeds by evaluating O against every trained sign model λw in the vocabulary using the forward algorithm (Section 5.5.3), and selecting the sign whose model assigns the highest likelihood to the observed sequence:

$w^* = arg\ max_w\ P(O \mid \lambda w)$

For continuous sign-language sentences (rather than isolated signs), individual sign HMMs are typically concatenated into a larger network (analogous to connected-word speech recognition), and the Viterbi algorithm is used to simultaneously segment the continuous observation sequence into sign boundaries and identify the most likely sequence of signs.

5.6.4 Practical Challenges

- Coarticulation — the shape and trajectory of one sign is influenced by the signs immediately before and after it, similar to coarticulation effects in speech.
- Signer-dependent variation — different signers perform the same sign with different speed, amplitude, and style; HMMs trained across multiple signers must be flexible enough to capture this variability without losing discriminative power between different signs.
- Non-manual features — facial expression, mouth shape, and head movement often carry grammatical information (e.g. negation, questions) alongside the hand gestures, requiring multi-channel or multi-stream HMM extensions.
- Vocabulary scaling — as the number of distinct signs grows, evaluating every sign model against the input becomes expensive, motivating hierarchical or pruned search strategies analogous to those in Section 5.4.

> **Note:** Sign-language recognition exemplifies a general pattern-recognition-by-HMM pipeline: extract per-frame features, train one HMM per class, and classify a new sequence by picking the class HMM with the maximum likelihood — the same pipeline underlies isolated-word speech recognition and many gesture-recognition systems.

5.7 Finding People with HMM

Detecting and tracking human figures in images and video is a classic and challenging recognition problem, because the human body is highly articulated (it has many degrees of freedom at the joints), and can appear in a huge variety of poses, viewpoints, and clothing. A powerful strategy, building directly on the relational and sequential ideas of this unit, is to model the human body as a set of parts connected in a kinematic chain, and to use an HMM (or a closely related structured probabilistic model) to encode both the appearance of each part and the spatial or temporal relations between adjacent parts.

5.7.1 Body Parts as States

In this formulation, each body part (head, torso, upper arm, lower arm, upper leg, lower leg, and so on) is treated as a state in a structured probabilistic model. The chain (or tree) structure of the human skeleton naturally defines which parts are directly related: the lower arm connects to the upper arm at the elbow, the upper arm connects to the torso at the shoulder, and so on. Unlike the fully general graphical model of Section 5.3, this kinematic chain is tree-structured, which — exactly as noted in Section 5.3.4 — permits exact and efficient inference using dynamic programming.

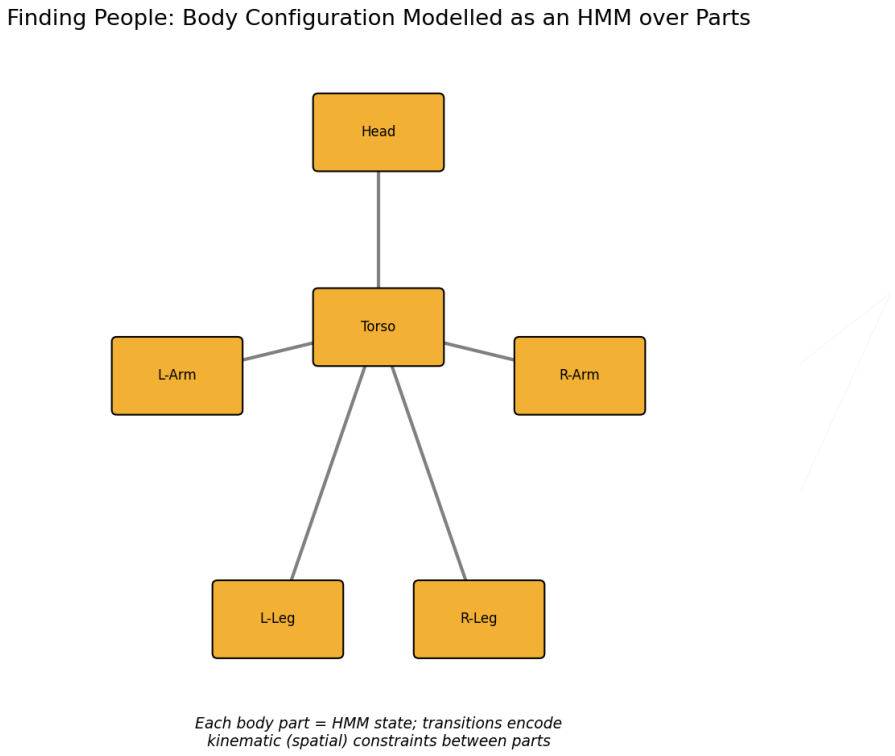


Fig. 5.7 — The human body modelled as a chain/tree of parts; each part is a 'state', and the relations between connected parts encode plausible kinematic configurations (e.g. elbow angle range, expected part-to-part distance).

5.7.2 Spatial HMM / Pictorial Structure Formulation

Where a temporal HMM has states indexed by time, a spatial version used for body-part finding has states indexed by position in the kinematic chain (e.g. state 1 = head, state 2 = torso, state 3 = upper arm, …). The 'observation' at each state is the image evidence for that part (e.g. an edge-based or appearance-based part detector's response at a candidate image location), and the 'transition probability' between adjacent parts encodes the learned distribution of relative position, orientation, and scale between connected parts (e.g. how far the elbow can plausibly be from the shoulder, and at what range of angles). The joint configuration L = (l1, l2, …, ln) of part locations is scored by a product of unary (appearance) and pairwise (kinematic) terms, exactly analogous to Section 5.3.2:

$$P(L \mid Image) \propto \prod_{i=1..n} \varphi(li) \cdot \prod_{(i,j) \in Tree} \psi(li, lj)$$

and the optimal configuration L* = arg maxL P(L | Image) is found efficiently by dynamic programming over the tree — passing messages from leaf parts (e.g. hands, feet) inward to the torso and back out again, in a manner directly analogous to the forward/backward recursions used for temporal HMMs in Section 5.5.

5.7.3 Tracking People over Time

For video, the spatial body-part model at each individual frame can be combined with a temporal HMM (or a more general dynamic Bayesian network) across frames, so that the body configuration estimated at frame t constrains the search for the body configuration at frame t+1 — exploiting the fact that human motion is smooth and that body parts cannot move arbitrarily far between consecutive frames. This combined spatio-temporal model allows a person-finding system to maintain a consistent track of body pose through a video sequence, recovering gracefully from brief occlusions of individual parts using the same dynamic-programming machinery.

5.7.4 Practical Difficulties

- Self-occlusion — one body part can occlude another (e.g. an arm crossing in front of the torso), violating the simple independent-appearance assumption for the occluded part.
- Clothing and appearance variability — the same body part can have very different appearance across individuals and clothing, requiring appearance models that generalize well.
- Background clutter — many candidate image locations may spuriously resemble a body part (e.g. vertical edges resembling a limb), inflating the search space, which again motivates the classifier-pruning strategy of Section 5.4.
- Multiple interacting people — when several people are close together or overlapping, correctly associating detected parts with the correct individual becomes a much harder combinatorial problem.

> **Note:** Finding people with a (spatial) HMM / tree-structured model is a direct application of the general relational-reasoning framework of Section 5.3, specialized to the case where the relational graph is exactly the kinematic tree of the human skeleton.

5.8 Audio: The Physics of Sound

Although this course is primarily concerned with vision, the perception and processing of audio share many underlying principles with vision — both involve sensing a physical signal, transforming it through a biological (or artificial) sensor, and interpreting the result perceptually. This closing section introduces the physics of sound, the physiology of hearing, auditory perception, and auditory rendering, drawing the parallel with the image-formation, sensing, and perception pipeline studied for vision.

5.8.1 Sound as a Pressure Wave

Sound is a mechanical, longitudinal pressure wave that propagates through an elastic medium (typically air) as a series of compressions and rarefactions of air molecules. At any point in space, the sound pressure varies as a function of time, p(t), oscillating about the ambient atmospheric pressure. For a pure tone, this variation is sinusoidal:

$$p(t) = A \sin(2\pi f t + \varphi)$$

where A is the amplitude (related to loudness), f is the frequency in Hertz (related to pitch), and φ is the phase. The period T of the wave and its frequency f are reciprocally related, and the wave travels through the medium at the speed of sound v, related to wavelength λ by:

$$T = 1 / f \quad v = f \lambda$$

The speed of sound in air at room temperature is approximately 343 m/s, though it depends on the medium's density and elasticity (sound travels faster in water and faster still in solids).

5.8.2 Frequency, Amplitude, and Timbre

- Frequency (Hz) determines the perceived pitch of a sound: higher frequency is perceived as higher pitch. The typical human audible range is approximately 20 Hz to 20,000 Hz.
- Amplitude determines the perceived loudness of a sound, though loudness perception is non-linear in amplitude (see Section 5.10).
- Timbre is the quality that distinguishes two sounds of the same pitch and loudness (e.g. a violin versus a flute playing the same note), and arises from the relative strengths of the harmonics (integer multiples of the fundamental frequency) present in a complex tone, as revealed by its Fourier spectrum.

5.8.3 Sound Pressure Level (Decibels)

Because the human ear is sensitive over an enormous range of pressure amplitudes (a ratio of about 10 million to 1), sound intensity is conventionally expressed on a logarithmic decibel (dB) scale relative to a standard reference pressure p0 = 20 μPa (approximately the threshold of human hearing at 1 kHz):

$$L = 20 \cdot \log10(p / p0) \text{ dB SPL}$$

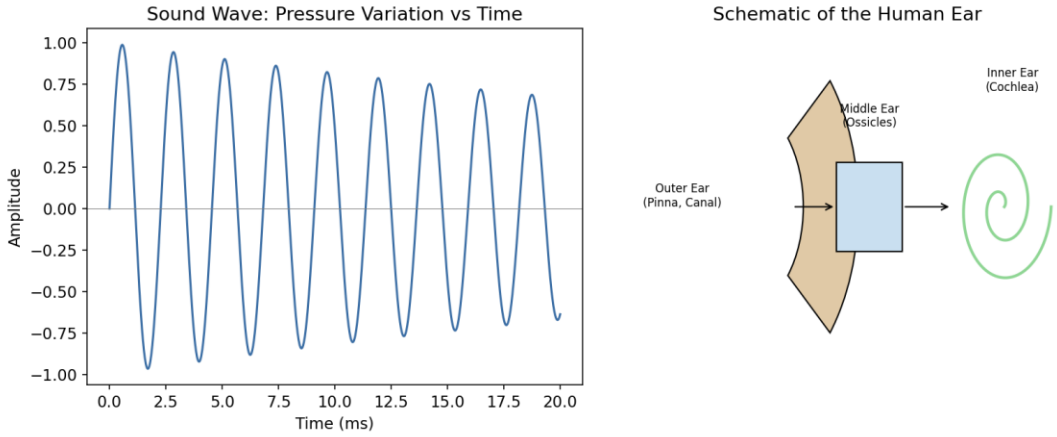


Fig. 5.8 — Left: a pure-tone sound wave (damped sinusoid) showing pressure amplitude versus time. Right: schematic anatomy of the human ear, showing the outer, middle, and inner ear.

5.9 The Physiology of Human Hearing

The human ear converts incoming sound pressure waves into neural signals through three anatomically distinct sections, each performing a different stage of the transduction process.

5.9.1 The Outer Ear

The outer ear consists of the pinna (the visible, folded cartilage structure) and the ear canal. The pinna's shape helps to collect sound and provides direction-dependent spectral cues (particularly for distinguishing sounds coming from above/below and front/behind), while the ear canal acts as a resonant tube that amplifies frequencies in the range important for speech (roughly 2–5 kHz), before the sound reaches the eardrum (tympanic membrane) at its inner end.

5.9.2 The Middle Ear

The middle ear contains three tiny bones — the ossicles (malleus, incus, and stapes) — which mechanically couple the vibration of the eardrum to the oval window of the cochlea in the inner ear. Because the inner ear is fluid-filled and therefore has a much higher acoustic impedance than air, the ossicles perform crucial impedance matching, using a lever mechanism and the area difference between the eardrum and the oval window to amplify the pressure of the vibration substantially, minimizing the energy that would otherwise be reflected at an air-to-fluid boundary.

5.9.3 The Inner Ear (Cochlea)

The cochlea is a fluid-filled, spiral-shaped structure containing the basilar membrane, which runs along its length. Vibrations entering through the oval window set up travelling waves along the basilar membrane, and — critically — different frequencies of sound cause the point of maximum vibration amplitude to occur at different positions along the membrane: high frequencies peak near the base (close to the oval window), and low frequencies peak near the apex. This spatial separation of frequency, known as tonotopic organization, is the physiological basis of frequency (pitch) discrimination. Thousands of hair cells along the basilar membrane transduce this mechanical vibration into electrical neural impulses, which are transmitted to the brain via the auditory nerve.

5.9.4 Frequency Selectivity and Critical Bands

Because the basilar membrane's response to a pure tone is not infinitely sharp, the ear's frequency resolution is described in terms of critical bands — narrow frequency ranges within which sounds interact strongly (e.g. one tone can mask another within the same critical band, but has much less masking effect on tones in a distant critical band). Critical bandwidth increases with centre frequency and underlies many practical audio-perception phenomena, including simultaneous masking, which is exploited heavily in perceptual audio compression (e.g. MP3).

> **Note:** The impedance-matching function of the middle-ear ossicles, and the tonotopic (frequency-to-place) organization of the cochlea's basilar membrane, are the two physiological facts most frequently asked about in this section.

5.10 Auditory Perception

Auditory perception refers to how the physical properties of a sound wave (frequency, amplitude, spectral content, timing) are transformed by the auditory system into subjective perceptual experiences (pitch, loudness, timbre, and spatial location). Several important non-linearities and interactions characterize this transformation.

5.10.1 Loudness and Equal-Loudness Contours

Loudness is the subjective perception of sound intensity, and it depends not only on the physical sound pressure level (dB SPL) but also, importantly, on frequency — the ear is far more sensitive to some frequencies (particularly 2–5 kHz) than to others (particularly very low or very high frequencies), for the same physical sound pressure level. This frequency-dependent sensitivity is captured by equal-loudness contours (historically known as Fletcher–Munson curves), which plot the sound pressure level required, at each frequency, to produce a constant perceived loudness, measured in phons (by definition, a sound has a loudness of N phons if it is perceived as equally loud as a 1 kHz tone at N dB SPL).

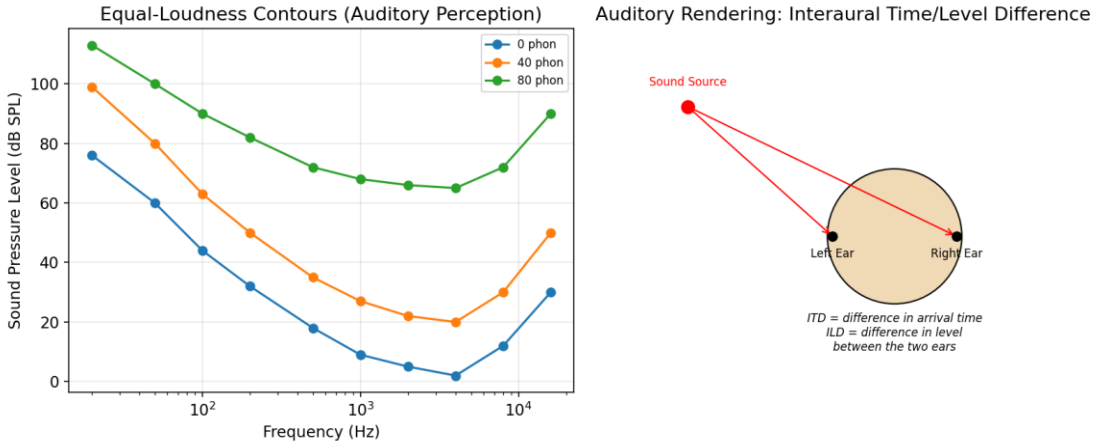


Fig. 5.9 — Left: equal-loudness contours, showing that the sound pressure level needed for a given perceived loudness (phon value) varies strongly with frequency. Right: interaural time and level differences used by the auditory system to localize a sound source.

5.10.2 Pitch Perception

Pitch perception is primarily driven by the tonotopic place of maximum excitation on the basilar membrane (place theory), and, for lower frequencies, additionally by the timing pattern of neural firing that follows the period of the stimulating waveform (temporal/periodicity theory). Together these mechanisms allow the ear to perceive pitch accurately across its full audible range.

5.10.3 Masking

Masking is the phenomenon whereby the presence of one sound (the masker) raises the threshold of audibility for another sound (the target), particularly when the two sounds share similar frequency content (within the same critical band, Section 5.9.4) or occur close together in time (temporal masking). Masking is central to lossy perceptual audio coding: components of a signal

that would be masked by other, louder components can be encoded with less precision (or discarded entirely) without a perceptible change in the decoded sound.

5.10.4 Sound Localization Cues

The auditory system localizes the direction of a sound source primarily using two binaural (two-eared) cues, which are described further in the context of auditory rendering below:

- Interaural Time Difference (ITD) — the difference in arrival time of a sound at the two ears, most useful for localizing lower-frequency sounds.
- Interaural Level Difference (ILD) — the difference in sound intensity at the two ears, caused by the acoustic shadow of the head, most useful for localizing higher-frequency sounds.
- Spectral (pinna) cues — direction-dependent filtering by the shape of the pinna, which helps resolve front-back and up-down ambiguities that ITD and ILD alone cannot distinguish.

5.11 Auditory Rendering

Auditory rendering is the process of synthesizing an audio signal (or pair of signals, for stereo/binaural playback) so that a listener perceives sound as originating from a desired virtual position in space — the auditory analogue of rendering a synthetic image of a 3-D scene from a desired camera viewpoint in computer graphics.

5.11.1 Interaural Time Difference (ITD) Model

For a sound source at azimuth angle θ relative to the listener's forward direction, and assuming a simple spherical-head model of radius r and speed of sound v, the interaural time difference can be approximated (Woodworth's formula) as:

$$ITD(\theta) \approx (r / v) \cdot (\theta + \sin \theta)$$

Synthetic audio rendering systems reproduce this delay by delaying the signal sent to one ear relative to the other by the appropriate ITD(θ), causing the auditory system to perceive the sound as arriving from angle θ.

5.11.2 Interaural Level Difference (ILD) Model

The head acts as an acoustic obstacle that attenuates sound reaching the far ear, particularly at higher frequencies where the wavelength is small relative to the head size. This attenuation is frequency- and angle-dependent, and is typically captured empirically via Head-Related Transfer Functions (HRTFs), denoted HL(f, θ) and HR(f, θ) for the left and right ear respectively; the interaural level difference is:

$$ILD(f, \theta) = 20 \cdot \log10 (|HL(f, \theta)| / |HR(f, \theta)|)$$

5.11.3 Head-Related Transfer Functions (HRTF) and Binaural Rendering

A Head-Related Transfer Function fully captures how sound from a given direction is filtered by the listener's head, torso, and pinnae before reaching the eardrum, encoding ITD, ILD, and the spectral (pinna) cues simultaneously as a function of frequency and direction. Convolving a monaural (single-channel) source signal with a pair of measured or modelled HRTFs (one for each ear) for the desired virtual direction produces a binaural signal that, when played over headphones, recreates a convincing perception of the sound arriving from that direction — this is the basis of modern virtual/augmented reality spatial audio rendering.

5.11.4 Reverberation and Distance Cues

Beyond direction, the perceived distance and the acoustic character of the surrounding space are rendered by simulating reverberation — the pattern of early reflections and the later, denser reverberant tail caused by sound bouncing off surfaces in an environment. The ratio of direct (unreflected) sound energy to reverberant sound energy is a strong cue for perceived distance: closer sources are perceived as having relatively more direct sound energy, while more distant sources are perceived as more reverberant relative to their direct sound.

> **Note:** Just as image rendering in computer graphics simulates the physics of light transport to produce a desired 2-D projection, auditory rendering simulates the physics of sound propagation and the physiology of binaural hearing (ITD, ILD, HRTF) to produce a desired spatial audio percept — keep this vision/audio rendering analogy in mind when answering comparative questions.

5.12 Solved Examples

The following worked examples illustrate how the HMM and voting formulae developed in this unit are applied to small numerical problems.

Example 1 — Vote Accumulation for Object Detection

Suppose an object model has two parts, Part A with learned offset dA = (2, 1) from the object centre, and Part B with learned offset dB = (−2, 1). In a test image, Part A is detected at pA = (7, 6), and Part B is detected at pB = (3, 6). The votes cast for the object centre are:

$$vA = pA - dA = (7,6) - (2,1) = (5, 5)$$

$$vB = pB - dB = (3,6) - (-2,1) = (5, 5)$$

Both parts vote for the same location (5, 5), producing a strong peak in the accumulator at that point and confirming a detected object centred at (5, 5) — exactly the voting mechanism illustrated in Fig. 5.1.

Example 2 — Forward Algorithm Computation

Consider an HMM with two states S1, S2, initial probabilities π = (0.6, 0.4), transition matrix a11 = 0.7, a12 = 0.3, a21 = 0.4, a22 = 0.6, and emission probabilities for observation o1 = 'A': b1(A) = 0.5, b2(A) = 0.1. The forward variable at t = 1 is initialized as:

$$\alpha 1(1) = \pi 1 \cdot b1(A) = 0.6 \times 0.5 = 0.30$$

$\alpha 1(2) = \pi 2 \cdot b2(A) = 0.4 \times 0.1 = 0.04$

Suppose the next observation is o2 = 'B' with b1(B) = 0.2 and b2(B) = 0.6. The forward variable at t = 2 is then computed using the recursion:

$\alpha 2(1) = [\alpha 1(1)a11 + \alpha 1(2)a21] \cdot b1(B) = [0.30(0.7) + 0.04(0.4)] \times 0.2 = [0.21+0.016] \times 0.2 = 0.0452$

$\alpha 2(2) = [\alpha 1(1)a12 + \alpha 1(2)a22] \cdot b2(B) = [0.30(0.3) + 0.04(0.6)] \times 0.6 = [0.09+0.024] \times 0.6 = 0.0684$

The total probability of observing the sequence O = (A, B) given this model is therefore:

$P(O \mid \lambda) = \alpha 2(1) + \alpha 2(2) = 0.0452 + 0.0684 = 0.1136$

Example 3 — Viterbi Decoding (continuing Example 2)

Using the same model and observations, the Viterbi recursion replaces the sum with a maximum. At t = 1, $\delta 1(1) = 0.30$ and $\delta 1(2) = 0.04$, identical to the forward algorithm's initialization. At t = 2:

$\delta 2(1) = \max[\ \delta 1(1)a11,\ \delta 1(2)a21\ ] \cdot b1(B) = \max[0.30 \times 0.7,\ 0.04 \times 0.4] \times 0.2 = \max[0.21,\ 0.016] \times 0.2 = 0.042$

$\delta 2(2) = \max[\ \delta 1(1)a12,\ \delta 1(2)a22\ ] \cdot b2(B) = \max[0.30 \times 0.3,\ 0.04 \times 0.6] \times 0.6 = \max[0.09,\ 0.024] \times 0.6 = 0.054$

Since $\delta 2(2) = 0.054$ is the larger value, the most likely state at t = 2 is S2, and backtracking through the arg max recorded at each step (state S1 was the arg max source for both $\delta 2(1)$ and $\delta 2(2)$ in this example) recovers the most likely state sequence Q* = (S1, S2).

Example 4 — Interaural Time Difference

Using Woodworth's formula with a head radius r = 8.75 cm = 0.0875 m, speed of sound v = 343 m/s, and a sound source at azimuth $\theta = 30° = 0.524$ rad, the interaural time difference is:

$ITD = (r/v)(\theta + \sin\theta) = (0.0875/343)(0.524 + 0.5) = (2.551 \times 10^{-4})(1.024) \approx 2.61 \times 10^{-4}\ s$

This corresponds to a delay of approximately 0.261 milliseconds between the two ears — a delay well within the range the human auditory system can detect and use for sound localization.

> **Note:** Numerical HMM computations (forward probability and Viterbi path) and simple offset-vector voting calculations are common short-answer / problem-based examination questions for this unit.

5.13 Summary of Unit V

- Recognition by relations between templates models complex or articulated objects as a collection of simple part-templates together with geometric or temporal relations between them, rather than a single rigid template.

- Voting-based recognition generalizes the Hough transform: each matched part votes for the object's reference location using a learned offset, and consistent votes from multiple parts reinforce each other at a peak in the accumulator.
- Relational reasoning formulates correspondence between model parts and image features as a search problem, scored by a probabilistic model combining unary appearance terms and pairwise (or tree-structured) geometric relation terms.
- Classifier-based pruning uses fast, cheap classifiers (often organized in a cascade) to eliminate implausible correspondences early, making relational search computationally tractable on cluttered images.
- A Hidden Markov Model $\lambda = (A, B, \pi)$ models a sequence generated by an underlying (hidden) Markov chain of states, each emitting an observable symbol; its three canonical problems — evaluation, decoding, and learning — are solved efficiently by the Forward algorithm, the Viterbi algorithm, and the Baum–Welch algorithm respectively.
- Sign language understanding trains one HMM per sign on extracted per-frame hand-shape/position features, and recognizes an unknown sequence by selecting the sign model with maximum likelihood (or via Viterbi decoding for continuous signing).
- Finding people applies the same relational/HMM machinery spatially, treating the kinematic tree of body parts as a tree-structured relational model solvable by dynamic programming, optionally extended temporally for video tracking.
- Sound is a pressure wave characterized by frequency, amplitude, and phase; the ear transduces this wave through the outer, middle, and inner ear, with the cochlea's tonotopic organization underlying frequency (pitch) discrimination.
- Auditory perception (loudness, pitch, masking, equal-loudness contours) and auditory rendering (ITD, ILD, HRTF-based binaural synthesis, reverberation) parallel the visual perception and rendering pipeline studied elsewhere in the course.

5.14 Review Questions

- 1. Explain the voting procedure for finding objects using relations between templates, and describe why it is robust to occlusion and background clutter.
- 2. How does object-detection voting relate to the Generalized Hough Transform studied in Unit IV?
- 3. Formulate object recognition as a search over correspondences between model parts and image features. Write the probabilistic scoring function combining unary and pairwise terms.
- 4. Compare exhaustive search, branch-and-bound search, and dynamic programming as strategies for solving the correspondence search problem.

- 5. Explain why restricting a relational model to a tree or star structure permits efficient exact inference.
- 6. Explain the role of a classifier cascade in pruning search, and derive the expressions for the overall detection rate and false-positive rate of a K-stage cascade.
- 7. Define a Hidden Markov Model formally, listing all of its constituent parameters.
- 8. State the Markov assumption and the output independence assumption used in HMMs.
- 9. Derive the forward algorithm recursion for computing $P(O \mid \lambda)$, and explain why it is more efficient than direct enumeration over all state sequences.
- 10. Explain the Viterbi algorithm and how it differs from the forward algorithm. What quantity does each algorithm compute?
- 11. Outline the Baum–Welch algorithm for learning HMM parameters from unlabelled observation sequences.
- 12. Describe how an HMM is used to recognize an isolated sign-language gesture. What features are typically used as observations?
- 13. Explain how a person's body configuration can be modelled and estimated using a tree-structured (spatial) HMM over body parts.
- 14. Define sound pressure level in decibels, and explain the function of the outer, middle, and inner ear in the transduction of sound.
- 15. Explain equal-loudness contours, and describe the interaural time difference (ITD) and interaural level difference (ILD) cues used for sound localization and auditory rendering.