Computer Vision - Exam Preparation Q&A

Computer Vision Students

Check updates here

July 13, 2026

Contents

Lecture 1-3: Introduction and Image Processing Basics

In a CV systems pipeline, how do low-level tasks compare versus mid and high level tasks?

Low-level processing focuses on image filtering, gray-level/color processing, and image enhancement (e.g., single-pixel transforms, DFT, local operators). Examples include single pixel and histogram-based transform, linear filters, non-linear filters.

Mid-level processing is based on extracting more complex models from the image, such as line detection, blob detection, salient features/keypoints, and edge detection algorithms. Examples include derivate, corner detection, hough transform, segmentation, feature detection (Keypoints detection and feature descriptor calculation, feature matching).

High-level processing includes object detection and recognition, image (semantic) segmentation, ML for CV, projective geometry, image formation and cameras, camera calibration, DL for CV.

Lecture 4-5: Single pixel ops, img histogram, spatial filtering

Define point operations and explain the difference between Logarithmic and Power-Law (Gamma) transformations

Point ops are spatial domain image processing techniques where the new value of a pixel depends strictly on the intensity value of that same pixel in the original image, without considering its neighbors. This is expressed as s = T(r), where r is the original intensity, s is the new intensity, and T is the transformation function.

Logarithmic transform $s = c\log(1 + r)$, where $c = \frac{L-1}{\log(L)}$ expands the values of dark pixels while compressing higher-level values. It is primarily used to compress the dynamic range of an image with huge variations in pixel values, such as displaying the FT spectrum.

Power-law / gamma transform $s = cr^{\gamma}$, where $c = (L-1)^{1-\gamma}$ is more versatile than the log transform as by altering the gamma value, it can both expand dark regions (when $\gamma < 1$) or expand bright regions (when $\gamma > 1$). It is most commonly used for gamma correction to adjust an image's contrast to match the display specs.

Compare and contrast "Contrast Stretching" and "Thresholding".

Both techniques use piecewise-linear transformations (mappings composed of connected linear segments that apply different linear scalings to distinct intensity ranges) to manipulate image contrast, but they serve different goals.

Contrast stretching: expands a narrow range of intensity levels in an image to span the full available dynamic range. It is used to correct low-contrast images by making the darks darker and brights brighter, while preserving the relative order of pixel intensities. It is represented as three segments, in which typically the one in middle has a slope coefficient > 1 resulting in a stretching of those intensity levels (contrast expansion), while the other two have slope coefficient < 1 resulting in an constrast compression.

Thresholding: extreme form of contrast stretching used for image segmentation. It maps all pixel intensities below a specified threshold value to a single dark value (like 0) and all intensities above the threshold to a single bright value (like L-1). Unlike stretching, thresholding irreversibly destroys the original intensity variations, resulting in a binary image.

What is the main limitation of Histogram Equalization, and why might we use Histogram Specification (Matching) instead?

The primary limitation of Histogram Equalization is that it is a fully automatic, global operation that aggressively forces the image's histogram to be as flat as possible. While this generally increases contrast, it cannot be controlled and may artificially enhance background noise or wash out details in uniform areas.

Histogram Specification (Matching) addresses this by giving the user control over the output intensity distribution. Unlike Equalization, Specification transforms the input image to match a specific, user-defined target histogram rather than forcing a flat one. This is useful for standardizing images from different sensors, for preventing the over-enhancement caused by standard equalization or for highlighting specific intensity ranges.

The histogram specification algorithm is a **global point operation**, meaning the output pixel value depends strictly on the input pixel intensity and the global transformation function (derived from CDFs), not on the neighborhood of pixels. Therefore, it differs from spatial filtering or edge detection, which rely on local operations.

The mathematical procedure consists of three steps:

1. **Equalize Input:** Compute the Cumulative Distribution Function (CDF) of the original image to map original intensities r to intermediate equalized values s.
2. **Equalize Target:** Compute the CDF of the specified target histogram to map desired output intensities z to the same intermediate equalized values v (where $v \approx s$).
3. **Inverse Transform:** Apply the inverse of the target transformation, $z = G^{-1}(s)$, to map the intermediate values s to the final target intensities z, resulting in an

image with the user-defined histogram shape (neglecting rounding effects).

A) Define an image histogram. B) What is the goal of histogram equalization and why is it useful? C) Outline the procedure for deriving the equalization function, providing an example.

A) An image histogram is a graphical representation of the distribution of pixel intensities in an image. Specifically, it is a bar graph where the x-axis represents the possible intensity values (0 to L-1) and the y-axis represents the frequency (number of pixels) that assume each specific value. It provides a global description of image's appearance, showing if image is well contrasted, under/overexposed (histogram heavily concentrated on left/right side, respectively). It doesn't contain any spatial information about pixels location in the image.

B) Histogram Equalization is an image enhancement technique used to improve the global contrast of an image. It's useful when an image's pixels intensities are concentrated in a narrow range (e.g. under/overexposed images). The goal is to redistribute the pixel intensities so that they are as evenly spread as possible across the entire available spectrum, resulting in a flatter histogram and higher contrast.

C) That's achieved by computing PDF of original image so to find frequencies for all L intensity levels, then deriving CDF, then map original pixel values r_k to new equalized values. By replacing every pixel with a newly mapped value, we generate an image utilizing the full dynamic range.

Let's assume now to have a 3-bit image ($L = 8$), currently very dark (total 10 pixels):

Original intensity r_k	**Pixel count** n_k	**PDF** $(n_k/10)$	**CDF**	$(L-1)$**CDF**	s_k
0	2	0.2	0.2	1.4	1
1	4	0.4	0.6	4.2	4
2	3	0.3	0.9	6.3	6
3	1	0.1	1.0	7.0	7
4 to 7	0	0	1.0	7.0	7

Resulting in a brighter image, considering that the new equalized intensities s_k are greater than the original values r_k.

Describe the mathematical procedure for performing Histogram Specification.

Histogram Specification involves transforming an input image so its intensity distribution matches a specified target PDF. The procedure consists of 3 main steps: 1. Equalize the original image: compute the CDF of the original image to find the transformation

$s = T(r)$. This maps the original pixels r to equalized values s. 2. Equalize the target histogram: compute CDF of the specified target histogram to find the transformation $v = G(z)$, $z =$ desired output pixel value. This maps the target pixels to equalized values v. 3. Apply the inverse transformation: Since both s, v are equalized (flat) distributions, let's assume $s \approx v$ and compute the inverse of the target transformation, $z = G^{-1}(s)$, and map the equalized pixels s from the first step to their final specified values z.

Lecture 6: Non-linear filters and frequency domain

Explain the principle of the median filter and its primary advantage over linear smoothing filters.

The median filter is a non-linear spatial filter that replaces the center pixel of a neighborhood with the median intensity value of the pixels within that window. Its primary advantage over linear filters (like the mean filter) is its superior ability to remove impulse noise (salt & pepper) without blurring the image excessively. While a mean filter averages the extreme noise values into the surrounding pixels, creating a blur, the median filter sorts the neighborhood values and picks the exact middle one, effectively discarding extreme outliers and preserving sharp edges (it doesn't compute any mean!).

How does the Bilateral Filter achieve edge-preserving smoothing, and how does it differ from a standard Gaussian filter?

A standard Gaussian filter smooths an image by computing a weighted average of neighboring pixels based solely on their spatial distance from the center. This reduces noise but blurs edges. The Bilateral Filter is a non-linear, edge-preserving technique that solves this by combining two Gaussian kernels: a spatial (domain) kernel and a range (intensity) kernel. The spatial kernel weights pixels based on physical proximity, while the range kernel weights them based on intensity similarity. Consequently, the bilateral filter only averages pixels that are both physically close and similar in brightness, ensuring flat regions are smoothed and preserving sharp intensity transitions / edges.

In the context of 2D FT, how do low and high frequency translate to the visual components of an image?

When an image is transformed into the frequency domain, its spatial intensity variations are represented as sinusoidal waves of varying frequencies. Low frequencies correspond to areas of the image where pixel intensities change slowly / are constant, i.e. smooth gradients, uniform background / illumination. They tipically contain the vast majority

of the image's energy and appearance. High frequencies correspond to areas of rapid intensity transitions i.e. edges, fine details, random noise. Removing high frequencies results in blurring, while emphasizing them sharpens the image.

Compare ideal LPF and Butterworth LPF (BLPF) in the frequency domain. Why is BLPF generally preferred in practice?

Both filters are used in the frequency domain to attenuate high frequencies and smooth an image, but they differ in their cutoff characteristics. The ideal LPF has an infinitely sharp cutoff radius, passing all frequencies below the threshold perfectly and cutting off all the higher frequencies entirely. This truncation causes severe ringing artifacts (concentric ripples) in the resulting spatial image. The BLPF introduces a smooth transition between the passed and attenuated frequencies, controlled by the filter's order. Because it avoids a sudden cutoff, a low-order BLPF significantly minimizes or completely eliminates the ringing effect, making it generally preferred for practical image processing and restoration. Note that as BLPF order grows to infinity, the BLPF approaches the ideal LPF, and ringing artifacts will return.

How are geometric spatial transformations mathematically represented and applied to an image?

Geometric transformations modify the spatial relationship between pixels (e.g. translating, rotating, scaling, shearing them). They are typically represented using affine transformations via transformation matrices and homogeneous coordinates. It allows us to combine multiple sequential operations, such as scaling an image and then rotating it, into a single composite transformation matrix. This matrix is then multiplied by the original pixel coordinates (x,y) to compute the new spatial locations.

Lecture 7: Filtering in frequency and geometrical transformations

Analyze the two steps required by a geometric trasform.

First is Coordinate transform which is a mathematical operation that works purely on geometrical points to determine their new spatial locations (represented as $(x', y') = T(x, y)$). The subsequent step is Image resampling that maps the actual pixel intensity values to the new discrete coordinate grid created by the transform.

How is a point represented in standard 2D coordinates vs homogeneous coordinates? And what is the structural advantage of using homogeneous coordinates vs standard cartesian ones when performing geometric translations?

In standard 2D coordinates a point is represented by two values (x, y). In homogeneous coordinates, an N-dimensional point is mapped into N+1 dimensions. For example, a 2D point (x, y) becomes a three-coordinate vector $(\tilde{x}, \tilde{y}, \tilde{w})$. You can always convert it back to standard coordinates by dividing by the scaling factor $\tilde{w}$, resulting in $x = \tilde{x}/\tilde{w}$ and $y = \tilde{y}/\tilde{w}$. In standard coordinates, an affine transformation requires separate linear operations and additions (e.g. multiplying by matrix A and adding translation vector b). By switching to homogeneous coordinates, multiple operations (including translation) can be easily combined into a single matrix multiplication.

What makes an affine transform different from a basic linear transform, and what specific properties does it preserve?

An Affine transform is a more generic planar transformation that consists of a linear transform followed by a translation. While it moves and scales points, an Affine transform specifically preserves point collinearity (points on a line stay on a line) and distance ratios along a line.

When analyzing an image for edges, what is the conceptual difference between using a derivative filter versus a true edge detection algorithm?

Derivative filters are a low-level operation; they simply highlight pixels where intensity changes rapidly. The resulting filtered image still lacks the actual concept of what an edge is. Edge detection algorithms, on the other hand, belong to mid-level processing because they mathematically embed the concept or "model" of an edge to actively detect and classify those structural features.

Lecture 8: Detecting edges and lines

How do first-order and second-order derivatives differ in their approach to edge detection?

Derivatives are used to detect rapid spatial changes in pixel intensity, which define edges. First-order derivatives (like the Sobel operator) compute the gradient of the image. An edge is identified by locating the local maxima (the highest peaks) in the gradient magnitude. The gradient also provides the direction of the edge. Second-order derivatives

(like the Laplacian) compute the rate of change of the gradient. An edge is identified by locating zero-crossing: the exact point where the second derivative passes through zero. While second-order derivatives are better at identifying the exact center of an edge, they are exponentially more sensitive to high-frequency noise. Both derivatives amplify noise. For this reason, it's strictly necessary to smooth the image before applying edge detection. The most appropriate and used choice for this is a Gaussian Low-Pass Filter, which has a smooth derivative and it's also the foundation of Canny Edge Detector.

In the Canny algorithm pipeline, what distinct structural problems do Non-Maxima Suppression and Hysteresis Thresholding solve?

- **Non-Maxima Suppression** solves the problem of thick, blurred edge responses. It ensures precise edge localization by thinning the gradient down to a single-pixel line. It achieves this by suppressing (setting to 0) any pixel that is not the local maximum along its quantized gradient direction.
- **Hysteresis Thresholding** solves the problem of fragmented edges and noise. By using two thresholds (T_H and T_L), it keeps strong edges, completely rejects isolated weak responses (noise), and most importantly, preserves weak edge pixels *only* if they are spatially connected to a strong edge, thus closing gaps.

How does the choice of the Gaussian kernel size (σ) in Step 1 dictate the final output of the Canny edge detector?

- **A small** σ applies minimal smoothing. This allows the detector to capture finer details and highly localized edges, though it makes the output more susceptible to keeping small noise components.
- **A large** σ applies aggressive smoothing. This blurs out fine textures and noise, restricting the detector to finding only coarse, large-scale, and highly prominent structural edges in the image.

Why is Canny considered a complete "edge detector algorithm" rather than just a simple derivative filter?

- A **simple derivative filter** (e.g., computing the gradient magnitude and applying a basic threshold) merely highlights areas of high intensity change. This typically results in thick, noisy, and broken boundary responses.

- The **Canny edge detector** embeds a complex, multi-step mathematical model of what an edge should be. It actively enforces three strict design criteria: a low error rate (via Gaussian smoothing), precise single-point localization (via non-maxima suppression), and continuous contour extraction (via hysteresis thresholding).

Does the Canny algorithm use block filters to speed up edge orientation evaluation?

No. The algorithm computes gradients g_x and g_y using standard derivative masks. The true computational optimization in this step comes from angle quantization. The continuous angle θ is snapped to one of four discrete directional bins $(0°, 45°, 90°, 135°)$. This allows the subsequent Non-Maxima Suppression to only perform scalar comparisons with predefined nearest neighbors in a 3×3 grid, avoiding complex floating-point geometry.

What are the 3 hyperparameters of the Canny edge detector?

Note that Canny is a deterministic algorithm, not a machine learning model, so it does not have trainable weights. Instead, it possesses three configurable hyperparameters that must be manually tuned based on the specific application:

- σ (Gaussian kernel size): controls the scale of edges detected and noise smoothing.
- T_H (High Threshold): determines the minimum gradient magnitude for a strong, guaranteed edge pixel.
- T_L (Low Threshold): determines the tolerance for connecting weak edge pixels to strong ones during hysteresis.

When building the parameter space for the Hough Transform, why is the normal representation ($x\cos\theta + y\sin\theta = \rho$) preferred over the standard Cartesian equation ($y = ax + b$)?

- **Cartesian Representation**: Maps an image point (x, y) to a line $b = -xa + y$ in the ab-parameter space. It fails to represent vertical lines because their slope (a) approaches infinity.
- **Normal Representation**: Solves the vertical line issue by mapping an image point to a bounded sinusoidal curve in the $\rho\theta$-parameter space. This allows the algorithm to gracefully detect lines at any angle.

In the Hough Transform algorithm, the parameter space (ρ, θ) must be quantized into accumulation cells. What is the engineering trade-off between using few cells versus many cells?

- **Few Cells (Coarse Quantization)**: Makes the algorithm robust to noise, allowing it to handle pixels that are not perfectly aligned along a mathematical line. The drawback is poor line localization.
- **Many Cells (Fine Quantization)**: Yields very accurate line localization, but requires the edge pixels in the image to be precisely aligned. If they are slightly scattered, they won't accumulate in the same cell, and the line might be missed.

How does the Generalized Hough Transform differ conceptually from the standard version, and what is its main computational drawback?

- **Standard Hough Transform**: Detects straight lines by finding intersections in a 2D parameter space (either ab or $\rho\theta$).
- **Generalized Hough Transform**: Extends the concept to find more complex shapes (like circles) using a general equation $g(\boldsymbol{v}, \boldsymbol{c}) = 0$, where $\boldsymbol{v}$ are coordinates and $\boldsymbol{c}$ are coefficients. For example, a circle uses $(x-c_1)^2+(y-c_2)^2 = c_3^2$. The main drawback is that complex shapes require a parameter space with **high dimensionality** (e.g., 3 dimensions for a circle), drastically increasing the computational and memory cost.

Why do we prefer the Hough Transform over a naive approach to find lines? (Computational Complexity)

- A **naive approach** would consider all pairs of edge points, evaluate the line passing through them, and count the points on that line. This yields a heavy computational complexity of $O(n^3)$ (considering $O(n^2)$ distinct pairs of points multiplied by a linear check to count the remaining points on the line).
- The **Hough Transform** avoids this combinatorics explosion by mapping each edge pixel independently into a parameter space and simply voting (accumulating) in a discrete grid, making line detection highly efficient.

Lecture 9: Morphological Operations and Thresholding

In morphological operations, how does Erosion differ from Dilation in terms of their physical effect on foreground shapes and component connectivity?

- **Erosion** shrinks or thins foreground objects. It is used to separate weakly connected components, as a pixel is kept only if the structuring element is *fully included* in the object set ($B_z \subseteq A$).
- **Dilation** expands or thickens foreground objects. It is used to merge close and unconnected components, as a pixel is kept if the structuring element has *at least one pixel overlapping* with the object set ($B_z \cap A \neq \emptyset$).

Structurally and mathematically, what is the exact difference between an Opening operation versus a Closing operation?

- **Opening** is defined mathematically as an Erosion followed by a Dilation: $A \circ B = (A \ominus B) \oplus B$. Structurally, it smooths the contour of an object, breaks narrow isthmuses, and eliminates thin protrusions.
- **Closing** is defined mathematically as a Dilation followed by an Erosion: $A \bullet B = (A \oplus B) \ominus B$. Structurally, it also smooths contours but does so by fusing narrow breaks and long thin gulfs, eliminating small holes, and filling gaps.

When dealing with imperfect binary images, how does Opening perform versus Closing in terms of noise removal and shape repair?

- **Opening** is best for "opening spaces" in the background: it removes small, bright (white) details or "salt" noise that are smaller than the structuring element, smoothing contours without reducing the main object's size.
- **Closing** is best for "closing gaps" in the foreground: it fills in small, dark (black) holes, narrow breaks, or "pepper" noise within the white objects, repairing the foreground without increasing the main object's overall size.

Since morphological operations are not cumulative, when is it appropriate to use [Opening then Closing] versus [Closing then Opening]?

- Use **Opening then Closing** when the background noise is severe but the main object is relatively sturdy (e.g., a noisy fingerprint). The Opening clears the background "salt" noise first so it doesn't get fused to the object, then the Closing patches cracks in the ridges.
- Use **Closing then Opening** when the main object is highly fragmented or full of holes (e.g., poorly printed dot-matrix text). You must Close first to fuse the broken pieces together; if you Opened first, the initial erosion step would completely erase the fragile object.

Is it mathematically valid to increase the strength of a morphological opening operator by simply applying it twice consecutively?

- **No, this is a fundamental conceptual error.** Morphological opening (erosion followed by dilation) is strictly an **idempotent** mathematical operation.
- **Idempotence ($A \circ B \circ B = A \circ B$):** Once an image is opened with a specific structuring element B, all structures smaller than the geometry of B are completely eradicated. Applying the exact same operator a second time will have absolutely zero additional effect because there is nothing left at that specific spatial scale to remove.
- **Correct Engineering Approach:** To aggressively increase the "strength" of an opening operation (e.g., to filter out larger noise blobs or disconnect thicker bridges), you must physically **increase the spatial dimensions** or change the shape of the structuring element B (e.g., scaling a 3×3 kernel up to a 5×5 kernel), rather than repeating the operation.

In the context of morphological descriptions, what is the conceptual difference between the image set versus the structuring element?

- The **image set** is the mathematical representation of the image itself. In the simplest case of a binary image, the image is defined as the set of coordinates containing the foreground pixels. Morphological operators act on this set by adding or removing pixels to/from it.
- The **structuring element** is the predefined "probe" or shape applied to the image. Its specific shape determines the direction(s) and extent to which pixels are added to or removed from the image set.

When segmenting an image, why might variable (local) thresholding be preferred over global thresholding?

- **Global thresholding** applies one single threshold value to the entire image. It works well only if the image has consistent lighting and a uniform background. It fails completely when the image suffers from uneven illumination, shadows, or gradients, because no single threshold can separate the foreground from the background everywhere.
- **Variable (or local) thresholding** divides the image into smaller sub-regions and calculates a distinct threshold for each specific area based on its local neighborhood. This allows the algorithm to dynamically adapt to local lighting changes, successfully segmenting objects even when the background intensity drifts across the image.

How does Otsu's method use inter-class variance versus intra-class variance to find the optimal threshold?

Otsu's method evaluates every possible threshold k, computes class probabilities and means, and picks the k that maximizes the inter-class variance σ_B^2. Since the global variance is fixed ($\sigma_G^2 = \sigma_{in}^2 + \sigma_B^2$), maximizing σ_B^2 is exactly equivalent to minimizing the intra-class variance σ_{in}^2, giving the optimal separation between the two classes.

Does Otsu's method fail when foreground and background regions have very different areas (sizes), and why?

No, it does not fail solely due to area differences. Otsu's method calculates the threshold based on minimizing intra-class variance. The class probabilities (P_0 and P_1) in the variance formula naturally account for the number of pixels (area) in each class.

As long as there is a significant contrast between the two regions (a distinct valley between the peaks in the histogram), Otsu will find the optimal threshold regardless of whether the foreground is small (e.g., a few pixels) or large (e.g., half the image).

The method *does* struggle if the histogram is unimodal or if the valley between the two modes is very shallow (low contrast), but this is an issue of contrast/noise, not a difference in the relative areas of the objects.

In the context of Otsu's algorithm, what is the conceptual difference between the cumulative mean $m(k)$ and the mean intensity value $m_1(k)$ for Class 1?

- The **cumulative mean** $m(k) = \sum_{i=0}^{k} ip_i$ calculates the mean up to intensity level k, but it is normalized over the *whole* image.
- The **mean intensity value** for Class 1:

$$m_1(k) = \frac{1}{P_1(k)} \sum_{i=0}^{k} ip_i$$

takes that cumulative mean and divides it by $P_1(k)$. This normalizes the value specifically to the number of pixels actually belonging to Class 1, giving the true average intensity of that segment.

How does image noise and object size affect the standard Otsu's method versus an approach that incorporates image smoothing?

- **Basic Otsu's method** struggles when dealing with noisy images or when target regions are very small (meaning they have a very limited impact on the histogram). Noise blurs the separation between histogram peaks.
- **Otsu's method with Smoothing** applies a spatial smoothing filter before calculating the histogram. This reduces noise, creates deeper valleys and clearer peaks in the histogram, and allows the algorithm to successfully segment smaller or noisier patterns.

Lecture 10: Segmentation by clustering

What is the primary goal of image segmentation, and how does Region Growing achieve it compared to standard thresholding?

The primary objective of image segmentation is to partition an image into distinct spatial regions that exhibit statistically homogeneous behavior. This means we group pixels based on their intensity (how bright/dark they are, i.e. grayscale) or spectral features (their specific color or wavelength profile, e.g. RGB). While standard thresholding groups pixels globally based purely on intensity without considering spatial relationships, **region growing** is an iterative, bottom-up spatial approach. It groups spatially contiguous pixels that share specific characteristics into larger regions. Algorithm Workflow:

1. **Initialization (Seeds):** Define the input image and a seed array containing 1s at seed locations and 0s elsewhere. If initial masks are obtained via rough heuristics,

each connected component is eroded to a single seed point, ensuring the final region is governed strictly by the growing predicate rather than the initial shape.

2. **Growing Predicate (Q) & Homogeneity Metric:** For a pixel to be merged, it must satisfy two strict conditions: **spatial connectivity** AND **similarity**. Connectivity is inherently enforced by the algorithm's search path (it only evaluates immediate neighbors). Similarity is evaluated by a logical predicate Q, which mathematically quantifies pixel similarity using a distance metric in an N-dimensional feature space (e.g., Euclidean distance in 3D RGB color space). The merge operation is governed by a user-defined scalar threshold T:

$$Q(x_i, y_i) = ||\vec{f}(x_i, y_i) - \vec{f}(x_s, y_s)|| \leq T$$

   If the neighbor's feature distance to the seed (or region mean) is smaller than T, Q evaluates to True and the pixel is merged. If False, the region stops growing in that direction.

3. **Growing Process:** Iteratively evaluate immediate spatial neighbors of the current active region. If predicate Q is true, the pixel is appended to the region.

4. **Termination Condition:** The spatial traversal algorithm continues until the entire image space is exhausted, terminating only when every pixel has been analyzed and assigned to a specific region (or marked as background).

5. **Output:** The final image is generated, where each connected component is assigned a distinct region label.

What are the primary algorithmic weaknesses and trade-offs of the Region Growing approach?

- **Seed Dependency (Non-Determinism):** Using random pixels as initialization seeds is a major weak assumption that leads to highly unstable, non-deterministic segmentation outputs. Robust, industry-level implementations require deterministic seed generation strategies (e.g., utilizing local histogram maxima, distance transforms, or regular spatial grids).

- **Threshold Brittleness:** Relying on a rigid, global threshold (like Euclidean distance in standard RGB space) assumes perfectly uniform illumination. This logic structurally fails when confronted with gradients, shadows, or specular highlights, often resulting in under-segmentation.

- **Greedy Local Decisions (The "Leakage" Problem):** Region growing relies on local similarity criteria and therefore lacks global geometric awareness. Weak or discontinuous boundaries may allow the growing process to leak across object

borders, causing distinct regions to merge. To reduce this problem, edge maps are often dilated before region growing so that small breaks in detected boundaries are closed, creating more reliable stopping barriers.

Explain the core principle of Watershed segmentation.

The algorithm models a grayscale image (or its gradient magnitude) as a 3D topographic surface, where pixel intensity equals elevation.

- Flooding begins from local minima, forming expanding **catchment basins**.
- When water from distinct basins threatens to merge, a "dam" is mathematically constructed.
- The set of all dams constitutes the **watershed lines**, which serve as the final and fully connected segmentation boundaries.

What is the primary failure mode of Watershed segmentation, and what is the standard algorithmic enhancement used to resolve it?

Direct application on real-world images almost always causes severe oversegmentation. Image noise and minor local surface irregularities generate numerous spurious local minima, resulting in a highly fractured segmentation map comprised of tiny, useless regions. The standard solution is to abandon raw local minima and instead initiate flooding exclusively from a robust set of predefined **markers**.

- **Preprocessing:** The image is typically smoothed to suppress noise-induced variations.
- **Internal Markers:** Confirmed regions representing the actual objects of interest (e.g., prominent local minima surrounded by lighter pixels).
- **External Markers:** Regions firmly classified as the background.

By restricting the flooding seed points to these curated markers, the algorithm outputs a controlled number of meaningful, macroscopic regions.

What is the primary target application of Watershed segmentation, and what strict pre-processing pipeline does it require to succeed?

Watershed is specifically deployed for separating objects that are physically touching or overlapping (e.g., clustered cells or tree canopies), a scenario where standard global

thresholding fundamentally fails. However, Watershed is not a standalone operation; it requires a strict, sequential pre-processing pipeline to artificially construct the "valleys" (markers) it will flood:

- **Thresholding:** Generates the initial binary mask to separate foreground from background.
- **Morphological Operations (e.g., Dilation):** Eliminates internal noise (like holes within foreground objects) which would otherwise create false, unwanted basins that over-segment the image.
- **Distance Transform:** Computes the Euclidean distance from every foreground pixel to the nearest background pixel. This is critical for isolating the structural "cores" or centers of touching objects.
- **Connected Components:** Assigns discrete numerical seed labels to these isolated centers, providing the Watershed algorithm with the exact number and location of distinct objects to expand from.

What are the main drawbacks of the deterministic Watershed pipeline that have driven the industry shift toward AI?

- **Parameter Brittleness:** The method is highly rigid. It relies on manually tuned, hardcoded scalar values (e.g., threshold limits, morphological kernel sizes, distance mask sizes). These parameters are often found via trial and error and do not generalize well to images with varying scale or illumination.
- **Complex Boundaries:** Deterministic Watershed struggles significantly with fuzzy, complex, or low-contrast boundaries. Because the pre-processing pipeline is brittle and prone to generating artifacts on difficult images, industry practice has largely shifted to Machine Learning (like Convolutional Neural Networks) for robust, generalized segmentation without manual parameter tuning.

Define the objective function of the K-means clustering algorithm and outline the standard iterative optimization approach (Lloyd's algorithm).

K-means aims to partition pixel feature vectors into k disjoint sets $C_1, \ldots, C_k$ with centroids μ_i, minimizing the intra-cluster variance (Sum of Squared Errors) made by approximating points with their cluster centers:

$$\min \left(\sum_{i=1}^{k} \sum_{\vec{x} \in C_i} d(\vec{x}, \mu_i) \right)$$

where $d(\cdot)$ is an appropriate distance metric. Because an exhaustive search is computationally unfeasible, a heuristic approach is required. Lloyd's Algorithm does iterative optimization:

1. **Initialization:** Select k initial centroids. Common heuristics include the *Forgy method* (randomly select k actual data points, ensuring centroids start within the dataset distribution) or *Random partition* (randomly assign all points to clusters then compute initial centroids).
2. **Assignment:** Associate each feature vector $\vec{x}$ to the closest centroid:
$$C_i = \{\vec{x} : i = \text{argmin}_j d(\vec{x}, \mu_j)\}$$
3. **Update:** Compute the new centroids as the center of mass (mean) of the associated points:
$$\mu_i = \frac{1}{|C_i|} \sum_{\vec{x} \in C_i} \vec{x}$$
4. **Termination:** Repeat assignment and update steps until a termination condition is achieved (centroids do not sensibly move, or a maximum step limit is reached).
5. **Output Visualization:** In the final segmented image, every pixel in a given cluster is typically reassigned the mean intensity/color of its respective cluster centroid.

Critically evaluate the structural assumptions of K-means segmentation, and analyze the trade-offs of including spatial coordinates in the feature vector versus using color/intensity alone.

Algorithmic Weaknesses & Assumptions:

- **Non-Optimality:** The heuristic guarantees fast convergence, but only to a local minimum. The output is strictly dependent on the initialization.
- **Rigid Parameters:** The exact number of clusters (k) must be known/provided in advance.
- **Geometric Bias (Spherical Symmetry):** The algorithm assumes data is evenly distributed around the centroid within a certain radius. It is mathematically incapable of accurately capturing real-world data structures that form elongated, non-convex, or highly irregular shapes in the feature space.

Feature Vector Composition Trade-offs:

- **Color/Intensity Only:** Segmented clusters will contain pixels that are **not necessarily spatially connected components** (e.g., similar objects on opposite sides

of an image group together). It also fails critically on textured objects with high local variance (like cabbage), resulting in meaningless, highly fragmented clusters.

- **Color + Position (x, y):** Appending spatial coordinates enforces local compactness, vastly improving the separation of physically distinct objects. **Trade-off:** Large, homogeneous background regions will be artificially fractured into multiple distinct clusters simply because the spatial distance from a single centroid grows too large to minimize the objective function efficiently.

Explain the conceptual foundation of density-based clustering using Kernel Density Estimation and outline how it theoretically addresses K-means' limitations.

While K-means forms clusters based on the shortest distance to a calculated centroid (forcing spherical symmetry and requiring a predefined k), density-based clustering defines clusters as areas of high data point density separated by areas of lower density.

To find these areas, the algorithm constructs a continuous Probability Density Function (PDF) from the discrete pixel data using **Kernel Density Estimation** (Parzen Window Technique):

1. **Kernel Placement:** A chosen kernel function (e.g., Uniform, Gaussian/Normal, Epanechnikov) of radius r is centered on every single sample point $\vec{x}_i$ in the dataset.
2. **Summation (Convolution):** The individual kernel contributions are summed across the feature space, blending the distributions to create a single, smooth global density estimate curve.
3. **Mode Seeking:** The algorithm then identifies the **modes** (the major peaks or local maxima) of this synthesized PDF. Each mode corresponds to a cluster center. All regions of the input space that "climb" toward the same peak are assigned to that specific cluster.

This approach theoretically solves major K-means flaws because it does not force arbitrary spherical shapes (clusters conform to the natural data topography), and the number of clusters is discovered organically based on the number of peaks found, rather than being guessed in advance.

Formally define the mathematical construction of a Probability Density Function using Kernel Density Estimation, and explain the primary computational drawback of this method.

Given N input samples $\vec{x}_i$ in an n-dimensional feature space, the estimated density function $f(\vec{x})$ evaluated at a generic point $\vec{x}$ is defined as the sum of translated kernels:

$$f(\vec{x}) = \frac{1}{Nr^n} \sum_{i=1}^{N} K(\vec{x} - \vec{x}_i) = \frac{c_k}{Nr^n} \sum_{i=1}^{N} k\left(\frac{||\vec{x} - \vec{x}_i||^2}{r^2} \right)$$

Where:

- $K(\cdot)$ is the specific kernel function (which must integrate to 1).
- $k(\cdot)$ is the 1-dimensional profile generating the kernel.
- r is the kernel radius (bandwidth), controlling the smoothness of the estimation.
- The factor r^n normalizes the equation by the number of dimensions to ensure the resulting PDF integrates to 1.

While theoretically robust, explicitly constructing and evaluating this global density function $f(\vec{x})$ everywhere across an n-dimensional feature space is **computationally highly complex and inefficient**. Calculating the contribution of every data point to every other coordinate in the space makes direct density function processing generally impractical for high-resolution image segmentation, necessitating alternative methods for finding peaks (like Mean Shift).

Lecture 11: Mean Shift and introduction to features

How does Mean Shift differ from standard classification or regression in real-world ML systems?

- **Different Objective:** Classification and regression are **Supervised Learning** tasks that predict labels or values. Mean Shift is an **Unsupervised Learning** clustering algorithm used to discover hidden structures (high-density modes) in unlabeled data.
- **Complementary Role:** It is not a replacement for supervised models. Real-world CV systems use complex pipelines that layer unsupervised methods (like Mean Shift for segmentation) together with supervised classifiers and feature engineering.

Why does the Mean Shift algorithm evaluate the density gradient rather than computing the full PDF?

Calculating the full density function across a high-dimensional feature space is computationally intractable and fundamentally unnecessary if the goal is only to find peaks.

Mean Shift acts as a steepest-ascent optimization method. By computing only the gradient of the density field, the algorithm directly obtains the vector pointing to the local maximum (the mode). It bypasses the global PDF computation entirely.

Extra: Both Mean Shift and kernel density estimation (KDE) are closely related: Mean Shift is derived from KDE but does not build a global density estimate. Instead it repeatedly computes a local, weighted centroid (the mean shift vector), which is proportional to the KDE gradient, and follows that vector to climb toward modes. This avoids evaluating the full PDF everywhere, but each iteration still requires evaluating the kernel over the local window and many iterations may be needed, so Mean Shift can remain computationally expensive in practice (per-step cost $\mathcal{O}(W)$, where W is the number of points in the window).

What is the mathematical decomposition of the density gradient $\nabla f_k(x)$, and what is its structural purpose in the Mean Shift algorithm?

In the formal derivation of Mean Shift, the kernel density gradient $\nabla f_k(x)$ is algebraically factored into three distinct terms, as $\nabla f_k(x) = A \cdot f_g(x) \cdot m_g(x)$, where:

1. $A = \frac{2c_k}{r^2 c_g}$ is the **constant term** establishing the scaling factor based on kernel profiles.
2. $f_g(x) = \frac{c_g}{Nr^n} \sum_{i=1}^{N} g(y_i)$ is the **kernel density estimator** evaluated at x, measuring the absolute magnitude of the local data density.
3. $m_g(x) = \left[\frac{\sum_{i=1}^{N} x_i g(y_i)}{\sum_{i=1}^{N} g(y_i)} - x \right]$ is the **mean shift vector**.

Structural Purpose: This factorization directly isolates the spatial translation vector $m_g(x)$. By demonstrating that the gradient $\nabla f_k(x)$ is strictly proportional to $m_g(x)$ (since A and $f_g(x)$ are both strictly positive scalars), it mathematically proves that calculating the local weighted center of mass inherently yields a vector pointing in the steepest-ascent direction. This elegantly eliminates the need to explicitly compute complex derivatives at runtime.

From an algorithmic perspective, why is the calculation of the mean shift vector restricted to a local window rather than processing all N data points in the feature space?

The algorithm utilizes kernel functions (e.g., Epanechnikov or truncated Gaussian) that possess a finite extension governed by the window size parameter r.

Mathematically, any data point outside this radial distance yields a weight of exactly zero in the profile derivative $g(y_i)$. Therefore, restricting the calculation to only the points within the window is not an approximation. It yields the exact local gradient while drastically dropping the computational cost of a single step from $O(N)$ (evaluating the whole dataset) to $O(W)$, where W is the number of points inside the local neighborhood. W is on average way lower than N.

How does Mean Shift fundamentally differ from K-Means in defining clusters and structural assumptions?

- **K-Means**: Requires a strictly predefined hyperparameter k (the number of clusters), structurally assumes that clusters are spherical, and is highly sensitive to both its random initialization and spatial outliers (since it minimizes global variance).
- **Mean Shift**: Resolves these flaws. It does not assume spherical clusters, does not require knowing k a priori, and avoids initialization instability. Furthermore, because it relies on local density estimation rather than global variance, it is highly robust to outliers; it does not distort cluster boundaries to accommodate scattered noise.
- **Cluster Definition (Topological):** Mean Shift conceptualizes the feature distribution as a density function where clusters are defined by local maxima ("hills"). It defines an **attraction basin**: all data points whose gradient trajectories (hill-climbing) terminate at the exact same local mode constitute a single cluster. The number of clusters found depends entirely on the underlying data distribution and the chosen kernel window size r.

The iterative procedure stops when the gradient is close to 0. How does Mean Shift differentiate between a true mode and a saddle point, given both have a zero-gradient?

Saddle Points are zero-gradient locations, but they are structurally unstable. To prevent the algorithm from halting at a saddle point instead of a true peak, Mean Shift introduces a small mathematical **perturbation**. Because the saddle point is an unstable equilibrium, a microscopic random shift will easily push the kernel window off the plateau, causing the steepest-ascent gradient to become non-zero and allowing the trajectory to resume moving freely toward a true mode.

What are the failure modes of choosing an inappropriate kernel window size (r) in Mean Shift, and what are its general computational drawbacks?

Mean Shift relies heavily on a single primary hyperparameter: the kernel extension or window size r (bandwidth).

- **Window too small (r is small)**: The algorithm becomes highly sensitive to local noise and outliers, resulting in extreme over-segmentation (finding vastly more modes than actual objects).
- **Window too large (r is large)**: The kernel over-smooths the density space. Distinct peaks merge into single, broad attraction basins, causing distinct objects/modes to be erroneously clustered together (under-segmentation).
- **Computational Complexity**: Regardless of r, the algorithm is computationally expensive compared to standard K-Means. It requires repeatedly calculating distances and evaluating the kernel function across points, and it does not scale well with the dimensionality of the feature space or massive dataset sizes.

In Mean Shift image segmentation, how does the joint spatial-color kernel govern region coherence, edge preservation, and texture flattening?

Mean Shift operates in a joint feature space by multiplying two independent kernels:

$$K(x) = c_k \cdot k_s \left(\frac{x_s}{r_s} \right) \cdot k_r \left(\frac{x_r}{r_r} \right)$$

- **Region Coherence (spatial + range VS range only):** Segmenting solely on intensity/color results in spatially disconnected clusters (e.g., pixels on opposite sides of an image merging into the same segment just because they share the same dark shadow intensity). Incorporating the **spatial kernel** (k_s) enforces strict local coherence. During the hill-climbing process, a pixel's trajectory is only influenced by neighboring pixels that are close *both* physically (within r_s) *and* chromatically (within r_r).
- **Edge Preservation (discontinuities):** Structural boundaries are exceptionally well-preserved compared to standard linear filters (like a basic Gaussian). If a pixel sits exactly on a sharp edge, the spatial window overlaps both sides, but the **range kernel** (k_r) acts as a hard mathematical filter assigning zero weight to pixels across the boundary (if their color difference exceeds r_r). Therefore, the shift vector is only pulled by pixels on the *same side* of the edge. The trajectory effectively runs parallel to the discontinuity, aggressively smoothing interior textures (painterly effect) while keeping the boundary contrast intact.
- **Parameter Trade-offs (r_s vs. r_r):**
  - Increasing the **color window** (r_r) aggressively flattens distinct textures, merging varied chromatic gradients into solid, uniform blocks.

– Increasing the **spatial window** (r_s) forces the algorithm to bridge larger physical distances, generating broader geometric regions. However, the boundaries between these large regions will remain perfectly crisp as long as the color tolerance (r_r) is kept strictly small. A small r_r ensures the algorithm still refuses to blur different colors together. If r_r is also increased, the algorithm will merge regions across color boundaries, destroying the edges.

How is the standard feature extraction and matching pipeline structured conceptually?

The pipeline operates as a sequence of strictly dependent stages:

1. **Feature Detection (Upstream):** The initial step that identifies stable, geometric points of interest (keypoints).
2. **Feature Description (Middle):** Computes mathematical signatures (feature vectors) for the previously detected points.
3. **Feature Matching or Feature Tracking (Downstream):** These are downstream tasks because they cannot function without the descriptor vectors generated by the previous steps. They establish relationships by comparing those descriptors.

What is the structural difference between feature detection and feature description in the computer vision pipeline?

- **Feature Detection:** The process of identifying locally distinct, structurally stable geometric points (keypoints) across an image (*where* to look).
- **Feature Description:** The extraction of a high-dimensional mathematical vector (a signature) representing the visual properties of the local patch surrounding the keypoint (*how to identify* the point later).

What are the distinct engineering requirements for an ideal keypoint (detector) versus an ideal descriptor?

- An **ideal keypoint** must be geometrically reliable. It requires precise localization, high repeatability, **invariance** to geometric transformations (scale, rotation), and insensitivity to photometric (illumination) changes.

- An **ideal descriptor** must be highly distinctive yet compact. It requires mathematically **robustness** against:
  - **Occlusions:** Partial obstruction of the target patch.
  - **Clutter:** Dense, distracting backgrounds or overlapping structures.
  - **Degradations:** Sensor noise, motion blur, and compression artifacts.

How is the stability (Repeatability Score) of a keypoint detector mathematically quantified when comparing a pair of images?

- Stability is measured using a **repeatability score**. Given two images of the same scene, it is the ratio between the successfully matched ground-truth point pairs and the theoretical maximum possible matches:

$$\text{Repeatability} = \frac{\text{Number of corresponding keypoints detected in both images}}{\min(N_1, N_2)}$$

  where N_1 and N_2 are the total number of keypoints detected in Image 1 and Image 2, respectively. A higher score indicates a more stable and reliable detector.

According to the feature pipeline, what distinguishes feature matching from feature tracking?

Both processes are subsequent stages in the pipeline that rely fundamentally on **feature description** to find similar descriptors, differing primarily in their search domains:

- **Feature Matching (Global Search):** Establishes correspondences between vastly different images by globally comparing descriptors, typically minimizing a distance metric (e.g., Euclidean distance) in the high-dimensional feature space.
- **Feature Tracking (Local Search):** Establishes temporal correspondences across consecutive video frames. It exploits temporal continuity and local spatial proximity, heavily restricting the search area to a small neighborhood around the keypoint's previous location.

What are the primary higher-level CV tasks that rely on feature detection and matching/tracking pipelines?

- **Motion Estimation:** Using feature tracking matrices across frames to compute camera ego-motion or optical flow.
- **Image Stitching:** Using feature matching to compute homographies, aligning overlapping photos to construct panoramic mosaics.
- **3D Reconstruction:** Triangulating matched features across multiple 2D viewpoints (epipolar geometry) to calculate depth maps and construct 3D models.

Lecture 12: Harris corners and SIFT

Consider the Harris corner detector, where corner detection is mainly based on the autocorrelation matrix A: a) Describe how the autocorrelation matrix is mathematically derived for a grayscale image $I(x, y)$. b) What kind of information about the content of the patch can be obtained from the autocorrelation matrix? Describe the relation between the autocorrelation matrix and the possible salient points to be found in an image. c) Describe the various types of transformations that are invariant for the Harris corner detector, providing also some examples.

a) Mathematical derivation of the autocorrelation matrix A:
The detector analyzes the change in intensity for a small spatial shift (u, v) using the Sum of Squared Differences (SSD) as an auto-correlation function:

$$E(u, v) = \sum_{x,y} w(x, y)[I(x + u, y + v) - I(x, y)]^2$$

where $w(x, y)$ is a window function (e.g., uniform or a Gaussian weighting). Using a first-order Taylor expansion, the shifted image intensity is approximated as:

$$I(x + u, y + v) \approx I(x, y) + I_x u + I_y v$$

where I_x and I_y are the image gradients in the x and y directions. Substituting this back gives:

$$E(u, v) \approx \sum_{x,y} w(x, y)[I_x u + I_y v]^2 = \begin{bmatrix} u & v \end{bmatrix} A \begin{bmatrix} u \\ v \end{bmatrix}$$

where A is the autocorrelation matrix (or structure tensor):

$$A = \sum_{x,y} w(x, y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

b) Information obtained and relation to salient points:

The matrix A describes the variance of spatial gradients (data spread) within the local patch along its principal directions. By analyzing its eigenvalues λ_0 and λ_1 (which represent the magnitude of the maximum and minimum variance of the gradients), we classify the local image content based on spatial displacement $(\Delta x, \Delta y)$:

- **Flat/Uniform Region:** Both λ_0 and λ_1 are small. Moving the patch in any spatial direction produces little to no change in the local intensity (no salient point).
- **Edge:** One eigenvalue is large, and the other is small. Displacement perpendicular to the edge causes a massive change, but displacement along the edge causes almost no change.
- **Corner:** Both λ_0 and λ_1 are large. Any spatial displacement in any direction produces a significant intensity change. The junction of these gradients makes the point highly well-localized and uniquely identifiable, thus an ideal salient point.

c) Invariant transformations for the Harris detector:

- **Translation invariance:** Since the detector is based on numerical spatial derivatives, image gradients simply shift with translation. A translated corner will be perfectly re-detected (e.g., tracking a static object when a camera pans left/right).
- **Rotation invariance:** The eigenvalues of A are intrinsically rotation-invariant. The eigenvectors will rotate with the corner, but the magnitude of the principal curvatures remains identical (e.g., detecting corners on a square that gets rotated by 45 degrees).
- **Additive Brightness Offset (Partial Illumination Invariance):** Taking numerical spatial derivatives $(\nabla I_x, \nabla I_y)$ means adding a uniform brightness constant $I(x, y) + c$ identically vanishes during derivation (e.g., matching points in a slightly brighter room).
- **Not Invariant to Spatial Scaling (Failure Mode):** If an image is heavily magnified, a sharp geometric angle will stretch out and appear as a localized smooth edge or a flat curve to the detector. Since the algorithm evaluates a fixed-size analysis window, it fundamentally fails to recognize scaled-up corners.

Why does the Harris algorithm evaluate the response function $R = \det(A) - \alpha \cdot \text{trace}(A)^2$ instead of directly extracting the exact eigenvalues λ_0 and λ_1?

Computing exact eigenvalues involves solving a characteristic polynomial, which requires computationally expensive square root operations for every single pixel in the image.

The Harris response R mathematically bypasses this by exploiting properties of matrix invariants: $\det(A) = \lambda_0\lambda_1$ and $\text{trace}(A) = \lambda_0 + \lambda_1$. This allows for highly efficient classification:

- R is **significantly positive**: Indicates a corner (the product of the two large eigenvalues dominates).
- R is **significantly negative**: Indicates an edge (one large eigenvalue drives up the trace penalty squared, pushing the overall value negative).
- $|R|$ is **close to zero**: Indicates a flat region (both eigenvalues are negligible).

How does the USAN/SUSAN detector fundamentally differ from the Harris detector regarding computational logic and noise robustness?

- **Harris (Derivative-based):** Computes partial spatial derivatives to build an autocorrelation matrix. While mathematically rigorous, numerical differentiation inherently amplifies high-frequency image noise.
- **USAN/SUSAN (Derivative-free):** Evaluates a circular mask and simply counts the number of pixels possessing an intensity similar to the central "nucleus" pixel (i.e., calculating the USAN area).
  - If USAN area $\approx$ full mask: Uniform region.
  - If USAN area $\approx$ half mask: Edge.
  - If USAN area significantly $<$ half mask: Corner (Smallest USAN).

  Because SUSAN replaces numerical differentiation with thresholded integration (counting similar pixels), it is structurally far more robust to image noise.

What is the structural difference between a corner feature (e.g., Harris) and a blob feature like MSER (Maximally Stable Extremal Regions)?

- **Harris corners** focus on highly localized **specific 0D points** (pure coordinates, no spatial extension). Unlike an edge, an exact corner has zero dimensions: moving even a fraction of a pixel means you are no longer on the corner.
- Other features focus on **blobs**. A blob is a contiguous **2D region** where internal properties are (approximately) constant, yet strictly different from surrounding background regions. Because they possess measurable physical area, you can move locally within the boundaries of a blob and still remain inside the same feature.

MSER (Maximally Stable Extremal Regions) are specific connected areas characterized by almost uniform intensity, making the MSER feature detector function as a powerful blob detector. The core idea is based on two properties:

- **"Extremal"** refers to a **structural (order-based) property**: a region is extremal if all its pixels are strictly brighter or darker than its boundary. It does not measure how large the difference is, only that a consistent ordering exists.
- **"Maximally Stable"** refers to a **dynamic (multi-threshold) property**: among all extremal regions obtained by sweeping intensity thresholds, the algorithm selects those whose area remains nearly unchanged across thresholds. Stability is defined by minimal relative variation in region size.

In short:

- **Harris corners** → mathematical 0D points with no physical extension.
- **MSER blobs** → specialized, physically extensive 2D regions where:
  - **Extremal** → defines valid regions via intensity ordering (internal vs boundary).
  - **Maximally Stable** → selects regions that persist across threshold changes.

How does the SIFT algorithm construct the scale space, and what is the strict mathematical relationship governing the smoothing parameter σ within and across octaves?

- SIFT organizes the scale space into discrete blocks called **octaves**.
- **Within an octave:** The image resolution remains constant. The image is progressively blurred using a Gaussian filter. For an octave divided into s intervals, the smoothing parameter σ is multiplied by a constant factor $k = 2^{1/s}$ at each step.
- **Across octaves:** Instead of continuously increasing the filter size (which is computationally prohibitive), the image is **downsampled** (its resolution is halved). The next octave then re-applies the exact same base Gaussian filters to the smaller image, effectively doubling σ relative to the original image dimensions.

Why does SIFT rely on the Difference of Gaussians (DoG) rather than computing the true Laplacian of Gaussian (LoG) for feature detection?

- The **LoG filter** ($\nabla^2 G(x, y)$) optimally combines smoothing and second-order derivatives to isolate high spatial frequencies (edges and corners) across scales. However, explicitly computing the second-order derivative matrix for every pixel at every scale is computationally expensive.
- The **DoG filter** $D(x, y, \sigma)$ is calculated simply by subtracting two consecutively smoothed images within the same octave. In SIFT, images are already computed at multiple scales by Gaussian smoothing within each octave. The Difference of Gaussians naturally arises as the difference between adjacent levels in this scale space, making it an efficient way to approximate the scale-normalized Laplacian without explicitly computing second-order derivatives. Mathematically, the heat diffusion equation proves that DoG is a highly accurate, computationally cheap approximation of the scale-normalized LoG ($\sigma^2 \nabla^2 G$), achieving the same zero-crossing and isotropic detection capabilities.

What is the exact volumetric neighborhood evaluated to identify a candidate keypoint in the DoG scale space?

Keypoints in SIFT are not detected in a flat 2D image, but as extrema in a **3D scale-space volume** (x, y, σ), where x, y are spatial coordinates and σ is the scale (blur level). Think of the DoG as a 3D scalar field where keypoints are simply peaks or valleys.

A candidate pixel is tested by comparing its DoG intensity to a $3 \times 3 \times 3$ cube centered on it, resulting in exactly **26 spatial-scale neighbors** (the center pixel itself is excluded):

- **8 neighbors** in the current DoG layer (same scale σ).
- **9 neighbors** in the DoG layer immediately above (scale $\sigma + \Delta$).
- **9 neighbors** in the DoG layer immediately below (scale $\sigma - \Delta$).

Selection Criterion (3D Extremum Test): The pixel is selected if and only if its value is *strictly greater* (local maximum) or *strictly less* (local minimum) than all 26 neighbors.

This matters as it enforces simultaneous locality in both space and scale. It ensures the point is not only distinctive in its 2D spatial neighborhood, but also that the structure exists at a specific, characteristic scale. Detecting extrema in a scale-space function rather than just "corners in an image" is what fundamentally gives SIFT its **scale invariance** and robustness to zoom.

Since digital images are discrete grids, how does SIFT achieve the high positional accuracy required for precise feature matching?

- The discrete maximum found by the 26-neighbor check is an approximation. SIFT achieves **subpixel location accuracy** by applying a 3D Taylor series expansion to the DoG scale-space function.
- By calculating the derivative and setting it to zero, the algorithm interpolates the continuous peak of the 3D quadratic function, yielding the exact subpixel (and sub-scale) coordinates of the extremum.

After finding local extrema, SIFT aggressively discards certain points. How is the Hessian matrix utilized to reject false keypoints?

- Local extrema in the DoG space often trigger strongly along straight structural edges. These points are highly unstable to noise and suffer from the aperture problem (poor localization along the edge axis).
- To eliminate them, SIFT evaluates local principal curvatures using the **Hessian matrix** of the DoG image (computed using finite differences):

$$H = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{yx} & D_{yy} \end{bmatrix}$$

- Using the ratio of the eigenvalues (λ_1, λ_2), the algorithm tests the structural shape without explicitly computing the roots:
  - If one eigenvalue is much larger than the other ($\lambda_1 \gg \lambda_2$), the local curvature is flat in one direction (an **edge**). The point is discarded.
  - If both eigenvalues are large and comparable, the point represents a highly curved **corner**. The point is kept.
- *Note:* SIFT also independently discards low-contrast points where the absolute DoG response is below a threshold (e.g., $|D(\hat{x})| < 0.03$).

How does SIFT determine the dominant orientation of a keypoint to achieve rotation invariance?

- SIFT evaluates the gradient magnitude $M(x, y)$ and direction $\theta(x, y)$ for every pixel in a local circular neighborhood around the keypoint (the size of this neighborhood is strictly proportional to the keypoint's detected scale σ).
- A histogram of gradient orientations is constructed with 36 bins (10 degrees per bin). Crucially, each pixel's contribution is weighted by its gradient magnitude **and** a Gaussian weighting function (to give less importance to pixels far from the center).

- The highest peak in this histogram represents the dominant local direction. To achieve sub-bin accuracy, a parabola is fitted to the 3 bins closest to the peak to interpolate the exact orientation angle. The descriptor's coordinate system is subsequently rotated to align with this angle.

How does the SIFT algorithm handle local neighborhoods that exhibit multiple strong gradient directions?

- While building the orientation histogram, SIFT evaluates secondary peaks alongside the absolute maximum.
- If any secondary peak has a magnitude strictly greater than 80% of the highest peak, the algorithm generates a completely **new keypoint**.
- This newly created keypoint shares the exact same spatial coordinates (x, y) and scale (σ) as the original, but is assigned the orientation of the secondary peak. This prevents ambiguous descriptor matching at complex junctions or corners.

What is the exact geometric and mathematical breakdown used to construct the 128-dimensional SIFT descriptor?

- A 16×16 pixel neighborhood around the keypoint is extracted at the correct scale and rotated to match the dominant orientation computed previously.
- This entire 16×16 grid is overlaid with a Gaussian weighting window (to smoothly downweight edges) and then subdivided into sixteen smaller 4×4 sub-regions.
- For each 4×4 sub-region, an 8-bin histogram of gradient orientations is computed (grouping directions into 45-degree increments).
- Concatenating the histograms from all 16 sub-regions yields exactly $16 \times 8 = 128$ dimensions. This forms a highly distinctive, spatially-structured gradient signature.

Why is the final 128-dimensional descriptor vector both normalized and explicitly thresholded (capped) at a value of 0.2?

- **Normalization:** The entire 128-element vector is normalized to unit length. This mathematically cancels out linear illumination changes (e.g., uniform shifts in image contrast, since gradients already cancel out uniform brightness shifts).

- **Thresholding (Capping at 0.2):** Global normalization fails against non-linear illumination effects (e.g., harsh specular reflections, camera saturation, or stark shadows) which create localized, massive gradient spikes. By capping any single vector component at a maximum value of 0.2, and subsequently re-normalizing the vector, SIFT prevents these localized lighting artifacts from dominating the entire descriptor and skewing the matching distance.

Lecture 13: Feature overview and feature matching

How does PCA-SIFT alter the standard SIFT descriptor calculation, and why does the patch size strictly shrink from 41×41 to 39×39?

- PCA-SIFT retains the first three steps of SIFT (detection, scale, orientation) but fundamentally changes the descriptor extraction. It extracts a 41×41 pixel patch centered on the keypoint and rotated to the canonical orientation.
- Instead of building 8-bin orientation histograms, it computes the raw horizontal and vertical gradients for the patch.
- The bounding pixels are strictly discarded (reducing the patch to 39×39) because calculating a discrete spatial derivative requires comparing a pixel to its neighbors, which is mathematically impossible for pixels on the outermost edge.
- The horizontal and vertical gradients are concatenated into a massive vector ($2 \times 39 \times 39 = 3042$ dimensions). This vector is normalized and then projected onto a lower-dimensional linear space using **Principal Component Analysis (PCA)** to maximize variance while drastically reducing the descriptor's footprint.

How does SURF utilize integral images and box filters to achieve its significant speed advantage over SIFT's Hessian matrix computation?

- **Integral Image:** An intermediate representation $I_\Sigma(x, y)$ where each pixel value is the sum of all pixels in the rectangle defined by the origin and (x, y). This allows the sum of intensities inside *any* arbitrary rectangular image area to be computed in $O(1)$ time using just four array references (e.g., $A + D - B - C$).
- **Box Filters:** SURF approximates the computationally heavy second-order Gaussian derivatives (for the Hessian matrix) using simple rectangular "box filters" composed of discrete black (negative weight) and white (positive weight) rectangles.

- Because these box filters are purely rectangular sums, they are computed instantly using the integral image, completely decoupling the processing time from the actual size of the filter.

What is the structural difference in how the scale space pyramid is constructed in SURF compared to SIFT?

- **SIFT:** Iteratively smooths the image with a Gaussian filter and then physically downsamples (reduces the resolution of) the image to build the next octave.
- **SURF:** Never subsamples the original image. Instead of shrinking the image, SURF **up-scales the filter size**. It applies progressively larger box filters directly to the original-resolution integral image. Because integral image lookups are $O(1)$, applying a massive box filter costs the exact same computational time as a small one, eliminating the need for iterative image resizing.

How does SURF determine the dominant keypoint orientation without computing exact, continuous image gradients?

- SURF evaluates a circular neighborhood of radius $6s$ (s = scale).
- It approximates the x and y gradients extremely fast using **Haar wavelets** (simple binary black/white filters).
- The responses are plotted on a 2D (x, y) plane. A sliding wedge of $60°$ sweeps around the origin, summing the wavelet responses within it. The angle of the wedge that yields the highest total sum is assigned as the dominant keypoint orientation.

What is the mathematical composition of the final SURF descriptor, and what is its primary engineering trade-off compared to SIFT?

- **Composition:** SURF extracts a square region around the keypoint (sized $20s \times 20s$, where s is the scale) aligned with the dominant orientation, and divides it into a 4×4 spatial grid of sub-regions (16 total). For each sub-region, instead of 8-bin histograms, SURF computes a simple 4-dimensional vector using Haar wavelet responses: $v = (\sum d_x, \sum d_y, \sum |d_x|, \sum |d_y|)$.

- Concatenating these 16 sub-regions yields a $16 \times 4 =$ **64-dimensional descriptor** (exactly half the size of standard SIFT).
- **Trade-off:** SURF achieves drastically faster computation and matching times due to integral images and a smaller descriptor vector. However, it is structurally **less robust** to extreme illumination variations and viewpoint (perspective) changes compared to SIFT.

How does the Gradient Location-Orientation Histogram (GLOH) alter SIFT's spatial grid, and how does it manage the resulting dimensionality explosion?

- GLOH modifies the descriptor calculation step by replacing SIFT's Cartesian 4×4 grid with a **log-polar location grid**. This grid consists of 3 radial distance bins and 8 angular bins (leaving the central base bin un-subdivided, resulting in $1+8+8 = 17$ spatial bins).
- Like SIFT, it computes 16 gradient orientations per spatial sub-region. This geometry creates $17 \times 16 = 272$ total orientation bins (a massive vector).
- To remain computationally competitive and remove redundant correlations, GLOH applies **Principal Component Analysis (PCA)** to strictly project the 272-dimensional vector down to a 128-dimensional space, matching SIFT's descriptor size but capturing radial spatial layouts more effectively.

What is the mathematical basis of the Shape Context descriptor, and what critical limitation led to its obsolescence?

- It quantizes relative point locations into a **2D** log-polar coordinate system (5 radial bins, 12 angular bins). **For clarity**: while the spatial layout is 2D, the descriptor builds a **3D** histogram by additionally accumulating the **edge orientations** (e.g., extracted via Canny) as the third dimension, counting the discrete number of edge points falling into each bin.
- Scale invariance is achieved by normalizing all radial distances by the mean distance between points.
- **Fatal Flaw:** While robust to noise and minor deformations, it is strictly **dependent on rotations**. It lacks native rotation invariance, making it highly brittle to viewpoint changes compared to modern descriptors.

How does LBP encode local image patches, and what is its primary architectural advantage?

- LBP compares a central pixel's intensity to its 8 immediate neighbors. Using a thresholding function: if the neighbor is strictly greater than the center, it outputs 1; otherwise, 0.
- This generates an 8-bit binary sequence (converted to a simple integer). Histograms of these values are concatenated across sub-regions.
- **Advantage:** It is incredibly **compact** and does not strictly require a standalone keypoint detector. It can be applied densely across an entire image or bounding box (originally designed for robust text and texture recognition).

How does BRIEF structurally differ from gradient-histogram descriptors like SIFT, and why is it vastly faster to match?

- Instead of accumulating continuous gradient vectors, BRIEF (Binary Robust Independent Elementary Features) applies Gaussian smoothing and directly compares the raw intensities of randomly sampled pixel pairs (x, y) inside a window.
- It utilizes a binary test: $\tau(p; x, y) = 1$ if $p(x) < p(y)$, otherwise 0. These binary results are concatenated into a single bitstring.
- **Speed Trade-off:** Because the descriptor is purely binary, feature matching relies on the **Hamming distance** (counting differing bits via a hardware-level XOR operation). This is computationally exponentially faster than calculating the Euclidean distance of floating-point vectors required by SIFT/SURF.

What components make up the ORB algorithm, and how does it geometrically fix the main limitation of the BRIEF descriptor?

- ORB (Oriented FAST and Rotated BRIEF) fuses the highly efficient **FAST corner detector** with the **BRIEF descriptor**.
- Standard BRIEF is entirely rotation-dependent; if the image rotates, the fixed pixel-pair sampling fails.
- ORB fixes this by computing the **intensity centroid** of the keypoint neighborhood. The vector from the geometric center to the intensity centroid defines a dominant orientation. ORB then explicitly rotates the BRIEF pixel-pair sampling coordinates to align with this axis, creating a rotation-invariant binary descriptor.

In feature space matching strategies, why is Nearest Neighbor Distance Ratio (NNDR) generally superior to simple Nearest Neighbor (NN)?

- **Simple NN** finds the mathematically closest descriptor in feature space but requires a hard, arbitrary global distance threshold to reject bad matches.
- **NNDR** evaluates the ratio between the distance to the absolute best match (d_1) and the second-best match (d_2). If $d_1 \approx d_2$, it indicates an ambiguous feature (e.g., repeating patterns like windows on a building). NNDR dynamically filters out these structurally ambiguous matches without relying on a rigid distance threshold, keeping only highly distinct matches where $d_1 \ll d_2$.

How do Precision and Recall mathematically and conceptually evaluate the quality of a feature matching pipeline?

- **Precision** ($TP/(TP + FP)$) measures *exactness*: out of all the matches the algorithm claimed were correct, what fraction were genuinely correct? It heavily penalizes False Positives (wrongly matched features).
- **Recall** ($TP/(TP + FN)$) measures *completeness*: out of all the true physical correspondences that actually exist in the scene, what fraction did the algorithm successfully detect? It penalizes False Negatives (missed true matches).

Lecture 14: Face Detection

Why is the sliding window approach computationally daunting for face detection, and what two absolute design imperatives must the Viola-Jones architecture satisfy to overcome this?

The sliding window must evaluate all spatial locations across multiple scales, resulting in millions of candidate patches per image.

Since the vast majority of these patches are non-faces, the architecture must achieve two things to be viable: 1) a **fast processing** to rapidly discard non-face background patches, and 2) a **very low false positive rate** to prevent the sheer volume of background windows from overwhelming the final output.

How are Haar features mathematically defined and evaluated, and what is the engineering trade-off regarding their structural coarseness?

- A Haar feature is fundamentally defined as the strictly coupled combination of a **function** and a **geometry**:
  - **Function:** The mathematical operation $f(x)$, which is the scalar difference between the sum of pixel intensities in adjacent "black" and "white" rectangular regions:
    $$f(x) = \sum_i p_{black}(i) - \sum_i p_{white}(i)$$
  - **Geometry:** The specific spatial layout (e.g., 2-rectangle, 3-rectangle), scale (width and height), and discrete position of those rectangles within the evaluation window.
- **Trade-off:** Haar features are structurally coarse; they can only capture simple gradients like horizontal/vertical edges or bars. However, because their geometry is strictly rectangular, they can be computed in $O(1)$ time using an **integral image**. This massive computational efficiency perfectly compensates for their lack of fine structural detail.

Given an intractable pool of ∼160,000 possible Haar features in a 24×24 window, how does AdaBoost construct a strong classifier and force the algorithm to learn difficult edge cases?

- AdaBoost sequentially selects the single best "weak learner" at each iteration. A weak classifier evaluates a single Haar feature $f_j(x)$, optimized with a threshold θ_j and a parity p_j (direction of the inequality):
$$h_j(x) = \begin{cases} 1 & \text{if } p_j f_j(x) > p_j \theta_j \\ -1 & \text{otherwise} \end{cases}$$
- **Re-weighting mechanism:** After each round, training examples that were misclassified by the current weak learner have their weights exponentially increased, while correctly classified examples are down-weighted. This forces the subsequent weak classifiers in the next round to mathematically prioritize correcting previous mistakes. The final strong classifier is the sign of the weighted sum of all chosen weak learners: $h(x) = \text{sign}\left(\sum_{j=1}^{T} \alpha_j h_j(x)\right)$.

What is the structural logic of the Viola-Jones cascade, and how does it enforce error asymmetry while drastically reducing computational cost?

- The cascade arranges weak classifiers into sequential, increasingly complex stages (e.g., Stage 1 uses 1 feature, Stage 2 uses 10 features). A sliding window must pass *every* stage to be classified as a face; if it fails any single stage, it is instantly rejected and computation halts entirely for that window.
- **Computational Efficiency (Early Rejection):** Because the vast majority of sliding windows in an image contain only background, the cascade rejects them almost immediately. Rejecting a background patch in Stage 1 requires evaluating only a single feature, saving the system from calculating thousands of complex features on empty space. The computationally heavy later stages are strictly reserved for rare, face-like regions.
- **Error Asymmetry:** The cascade intentionally treats false positives and false negatives differently. A **False Positive** at an early stage is highly acceptable because the stricter, more complex stages later in the pipeline are specifically designed to filter it out. Conversely, a **False Negative** is fatal: if a true face is rejected early, it vanishes from the pipeline forever. Therefore, early stages are aggressively tuned for near 100% recall (catching all possible faces) at the deliberate expense of precision.

What visual cues does the Viola-Jones detector capture, and how do local Haar patterns relate to human facial components?

- Haar features capture broad, high-contrast intensity differences between adjacent regions rather than fine textures. For human faces, these differences naturally arise from the 3D structure of the face casting predictable shadows under normal lighting.
- The original paper notes that AdaBoost autonomously selected these specific geometric cues as the most universally distinctive. The first two features are:
  1. A **vertical two-rectangle feature** measuring the difference between the **eye region** (which is darker due to **structural eye socket shadows**) and the **upper cheeks** (which protrude and catch light).
  2. A **horizontal three-rectangle feature** (side-by-side) contrasting the darker **eyes** on the exterior against the brighter, protruding **bridge of the nose** in the center.
- **Vulnerability & Bias:** Since these features rely on structural 3D shadows rather than textures, they assume standard top-down lighting. If a dark-skinned person with light eyes under flat lighting breaks this intensity gradient, the first stage

may output a negative response. Rejecting this at Stage 1 triggers an immediate, unrecoverable **False Negative** (causing the known racial and lighting biases in early systems).

Lecture 16: Object Recognition

In a cv pipeline, how do the structural outputs fundamentally differ between Classification + Localization, Object Detection, and Instance Segmentation?

- **Classification + Localization:** Assumes a single dominant object per image. It outputs one categorical label and one bounding box (4 coordinates: x, y, w, h).
- **Object Detection:** Relaxes the single-object assumption. It outputs an arbitrary number of bounding boxes, each coupled with a specific class label, allowing for multiple instances and clutter.
- **Instance Segmentation:** Drops the bounding box abstraction entirely. It computes exact pixel-level masks for every distinct object instance, effectively combining object detection with semantic segmentation boundaries.

Why is rigid template matching via a sliding window fundamentally brittle when deployed in unconstrained real-world environments?

A template is a fixed 2D array of pixels. It lacks native invariance to scale, rotation, translation, viewpoint changes, and illumination variations.

Even if geometric transforms are perfectly corrected, rigid templates fundamentally fail on intra-class variations (e.g., the structural differences between a wooden stool and a soft recliner, despite both being "chairs"). No simple affine transformation maps the geometry of one distinct object to another within the same semantic class.

Under what photometric conditions do SSD and SAD fail, and how does ZNCC mathematically mitigate this?

Both SSD and SAD evaluate raw intensity differences. If the target image undergoes a uniform linear illumination shift (e.g., adding a brightness constant), the absolute pixel values diverge drastically, causing the match to fail despite perfect structural alignment.

Zero-mean Normalized Cross-Correlation explicitly corrects this:

$$\phi(x,y) = \frac{\sum_{u,v \in T}(I(x+u,y+v) - \bar{I}(x,y))(T(u,v) - \bar{T})}{\sigma_I(x,y)\sigma_T}$$

By subtracting the local window mean ($\bar{I}$) and the template mean ($\bar{T}$), it centers the distributions at zero, canceling out uniform brightness shifts. By dividing by the standard deviations (σ_I, σ_T), it normalizes contrast.

This makes ZNCC stronger to illumination changes when compared with SSD and SAD, but it still might be problematic under particular lighting situations.

If ZNCC is too computationally expensive, how does a classical cv pipeline physically manipulate the data to cope with illumination, scale, and rotation changes in template matching?

- **Illumination:** Instead of matching raw intensities, convert both the image and the template into **edge maps** (or gradient orientations). Gradients measure local intensity changes, natively filtering out uniform, low-frequency illumination shifts.
- **Scale:** The system must brute-force the search space by either matching against a pre-computed bank of **several scaled versions** of the template, or by building an image pyramid and testing the fixed template across **multiple rescaled copies** of the target image.
- **Rotation:** Similarly requires matching the target against a generated bank of **several rotated versions** of the rigid template.

How does the Generalized Hough Transform conceptually act as a form of template matching, and what is its dimensionality trade-off?

Instead of sliding a dense matrix of pixels, the GHT mathematically models a complex shape using a general equation $g(\vec{v}, \vec{c}) = 0$, where $\vec{v}$ are spatial coordinates and $\vec{c}$ are shape coefficients. Edge pixels in the image "vote" in the parameter space to locate the template.

Be aware of the trade-off, as while it removes the spatial sliding window, it suffers from the curse of dimensionality. The more complex the shape (requiring degrees of freedom for scale, rotation, etc.), the higher the dimensionality of the parameter vector $\vec{c}$, making the accumulator space exponentially heavier to compute and threshold.

What is the exact sequence of operations used to compute a Histogram of Oriented Gradients (HOG) descriptor?

1. **(Optional) Preprocessing / intensity normalization:** Convert the image to a suitable intensity representation (typically grayscale). Optionally apply mild smoothing or contrast normalization to reduce noise and global illumination effects.
2. **Gradient computation:** Compute spatial gradients (e.g., g_x, g_y) to obtain, for each pixel, the **gradient magnitude** and **orientation** (phase).
3. **Cell histograms:** Divide the image into small non-overlapping cells (commonly 8×8 pixels). Within each cell, accumulate a histogram of gradient orientations where each pixel casts a vote weighted by its gradient magnitude. Orientations are typically quantized into **9 bins** over 0°–180° (unsigned gradients).
4. **Overlapping blocks:** Group neighboring cells into larger, spatially overlapping blocks (commonly 2×2 cells per block).
5. **Block normalization:** Concatenate the histograms within each block and normalize the resulting vector (e.g., using ℓ_2-norm variants such as ℓ_2-Hys) to achieve robustness to local illumination and contrast changes. A typical block descriptor has 4 cells $\times$ 9 bins = 36 elements.
6. **Feature vector assembly:** Concatenate (serialize) all normalized block descriptors into a single final feature vector, which is then used by a downstream classifier (e.g., linear SVM).

HOG Descriptor Dimensionality Calculation

Given a standard 64×128 pixel detection window, how is the exact dimensionality of the HOG descriptor mathematically derived to be 3780?

- **Cells:** At 8×8 pixels per cell, the 64×128 window is divided into a grid of 8×16 cells.
- **Histograms:** Each cell computes a 9-bin orientation histogram.
- **Blocks (Overlapping):** The algorithm groups cells into 2×2 blocks. Because these blocks overlap by one cell in each direction, an 8×16 cell grid yields $(8-1) \times (16-1) = 7 \times 15 =$ **105 blocks**.
- **Elements per Block:** Each block contains 4 cells, and each cell has 9 bins, resulting in $4 \times 9 =$ **36 elements** per block.
- **Final Vector:** 105 blocks $\times$ 36 elements/block = **3780 elements** (dimensions) serialized for the classifier.

What are the strict structural requirements for applying HOG to bounding boxes, and how does the algorithm compensate for its lack of native scale invariance?

- **Standardization Requirement:** The HOG descriptor characterizes the entire holistic content of a bounding box, producing a vector destined for a classifier (like an SVM). Because standard classifiers require fixed-length input vectors, every bounding box must be physically resized (warped) to a strictly predetermined standard pixel size (e.g., 64×128) *before* the descriptor gradients are computed.
- **External Scale Management:** Unlike SIFT, the HOG mathematical formulation is not natively scale-invariant. To detect objects of varying sizes, the pipeline must rely on an external mechanism: evaluating sliding windows across an image pyramid. The system extracts bounding boxes of different physical dimensions and scales them all down to the standard size to generate comparable vectors.
- **Hyperparameter Trade-offs:** The specific architecture (8×8 cells, 2×2 blocks) represents just one possible implementation. These are tunable parameters that dictate the final vector size and spatial granularity. Using smaller cells captures finer, high-frequency structural details but drastically increases the final vector dimensionality and susceptibility to spatial noise. Larger cells provide greater spatial invariance but risk blurring distinct boundaries together.

What is the fundamental abstraction of the Bag of Words (BoW) model in computer vision, and what is its primary geometric trade-off?

BoW treats an image as an unstructured, unordered collection of independent "visual words" (local feature patches). The model explicitly **discards all spatial relationships** and global geometry between features. While this makes the algorithm highly invariant to severe viewpoint changes, occlusion, and non-rigid deformations (e.g., recognizing a highly articulated human body), it fundamentally blinds the system to structural context (e.g., it cannot distinguish between a face and a scrambled collage of facial features).

How is the continuous feature space mathematically discretized to form the visual "codebook" or dictionary?

- First, highly discriminative local features (specifically using extractors like **SIFT, ORB, or GLOH**) are densely extracted across a massive training dataset containing the objects of interest.

- Because these continuous descriptors map to an infinite feature space, an unsupervised clustering algorithm (typically **K-means**) is applied to partition the space into K distinct groups.
- The **centroids** (representative samples) of these K clusters become the discrete "visual words". The entire collection of centroids serves as the definitive codebook, providing a finite, compact vocabulary.

What is the exact processing pipeline to encode a novel test image into a Bag of Visual Words (BOVW) representation

- **1. Extraction:** Detect and extract local feature descriptors from the test image. This is done for each object or class to be recognized
- **2. Quantization (Vector Association):** For every extracted descriptor, compute the distance (e.g., Euclidean) against all centroids in the pre-computed codebook. The descriptor is strictly mapped to its nearest neighbor codeword.
- **3. Histogram Generation:** Tally the total occurrences of each visual word. The output is a 1D **frequency vector** (histogram). This brilliantly normalizes the data: regardless of whether the original image produced 500 or 5,000 keypoints, the final frequency vector will strictly have a length of K (the size of the dictionary).

Downstream Execution: Matching vs. Classification

Once the fixed-length frequency vector is generated, how is it operationally utilized for final object recognition?

- **Direct Matching:** The test vector can be directly compared against a database of known object vectors using standard similarity metrics (e.g., Euclidean distance or dot product).
- **Machine Learning Pipeline (Industry Standard):** More effectively, the frequency vector serves as a standardized, fixed-length input for a discriminative classifier (such as a Support Vector Machine or a Random Forest). The classifier learns to map specific distributions of visual words to semantic labels.

Lecture 17: Pinhole camera model

Why does the standard computer vision camera model mathematically place the image plane *in front* of the optical center, deviating from how a physical pinhole camera works?

A physical pinhole camera captures light rays passing through the aperture, projecting an inverted (upside-down and horizontally flipped) image on the sensor plane located *behind* the focal point.

Placing a "virtual" image plane at the exact same focal distance f but *in front* of the optical center preserves the identical geometric projection properties (via similar triangles) while mathematically stripping away the negative signs. This completely avoids the upside-down effect, streamlining all subsequent coordinate transformations and matrix algebra.

Pinhole camera model: how does a 3D object project onto the (virtual) image plane? (Intro + sketch)

The **pinhole camera model** idealizes an imaging system as a single point (the **optical centre**) through which all rays pass. Every 3D scene point P defines a ray from the optical centre O; the **image** is the intersection of these rays with an **image plane**. In computer vision we typically use a **virtual image plane** placed *in front* of O at distance f to avoid the sign flip caused by the real sensor plane behind the pinhole.

Geometrically: perspective projection is purely **ray casting** from O to the plane, so points farther away (large Z) project closer to the principal point, while closer points (small Z) appear larger and more spread out.

How is the camera's Field of View mathematically defined, and what is the strict physical trade-off required to widen it?

The maximum viewing angle (FoV) is derived via trigonometry based on the physical dimension of the sensor d (using sensor width for horizontal FoV, or height for vertical FoV) and the focal length f:

$$\text{FoV} = 2\varphi = 2\arctan\left(\frac{d}{2f}\right)$$

To capture a wider scene (increase FoV), you must either increase the physical sensor size d (which drastically increases hardware cost and lens complexity) or decrease the focal length f (which reduces optical magnification and often introduces severe non-linear radial distortions, such as the "fish-eye" effect).

Projection geometry: what is the math, how does object size relate to image size, and how do you compute the size in pixels?

Let the camera frame have origin at the optical centre O, and place the virtual image plane at $Z = f$ (same length units as the scene, e.g., mm). For a 3D point $P = (X, Y, Z)$ with $Z > 0$, similar triangles give the metric image-plane coordinates

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

Size relationship (constant-depth approximation). Consider an object segment of physical length H (e.g., vertical height) located at depth Z and approximately parallel to the image plane (so its endpoints have roughly the same Z). Its projected length on the image plane is

$$h = f\,\frac{H}{Z},$$

so the magnification is $m = \dfrac{h}{H} = \dfrac{f}{Z}$: doubling distance halves the projected size. (If the object spans a depth range, different parts have different Z, yielding perspective foreshortening; in that case you must treat points separately.)

From metric length to pixels (digital sensor). Let the sensor have physical size $W_{sens} \times H_{sens}$ (e.g., mm) and resolution $W_{px} \times H_{px}$ (pixels). Pixel pitch (mm/pixel) is

$$p_x = \frac{W_{sens}}{W_{px}}, \qquad p_y = \frac{H_{sens}}{H_{px}}.$$

If h is a vertical metric length on the image plane, the corresponding pixel length is

$$h_{px} = \frac{h}{p_y} = \frac{f\,H}{Z\,p_y} = \Big(\frac{f}{p_y}\Big)\frac{H}{Z}.$$

Define the focal length in pixels $f_x = \dfrac{f}{p_x}$ and $f_y = \dfrac{f}{p_y}$ (these are the f_x, f_y in K). Then, for an object whose relevant extent is approximately at depth Z,

$$\text{(horizontal pixels) } w_{px} = f_x\,\frac{W}{Z}, \qquad \text{(vertical pixels) } h_{px} = f_y\,\frac{H}{Z}.$$

Why are standard Cartesian coordinates insufficient for 3D-to-2D camera modeling, and how do homogeneous coordinates solve this?

Standard Cartesian representations cannot express the non-linear perspective division (the division by Z) as a simple linear matrix multiplication.

By augmenting the dimensionality of the vectors (e.g., mapping a 3D point to a 4D vector $[X, Y, Z, 1]^T$), homogeneous coordinates allow translations, rotations, and perspec-

tive scaling to be unified into a single, continuous linear matrix operation: $\tilde{m} \simeq P\tilde{M}$. The final division is cleanly delayed until the coordinates are converted back from homogeneous to Cartesian space.

Camera model recap (pinhole)

Intrinsics —

$$K = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix},$$

where f_x, f_y are focal lengths in pixels ($f_x = f \cdot \text{px/mm}$), c_x, c_y the principal point, and s the skew (≈ 0). Lens distortion is not in K; common models use radial (k_1, k_2, k_3) and tangential (p_1, p_2) terms.

For a camera-frame point $X_c = [X_c, Y_c, Z_c]^T$ define normalized coords $x = X_c/Z_c,\ y = Y_c/Z_c$ and $r^2 = x^2 + y^2$. Distorted coords:

$$\begin{aligned} x_d &= x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + 2p_1 xy + p_2(r^2 + 2x^2), \\ y_d &= y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + p_1(r^2 + 2y^2) + 2p_2 xy. \end{aligned}$$

Pixel coordinates:

$$u = f_x x_d + s\, y_d + c_x, \qquad v = f_y y_d + c_y.$$

Extrinsics — $X_c = R\, X_w + t$, with $R \in \mathrm{SO}(3),\ t \in \mathbb{R}^3$ (6 DoF).

Coordinate conventions — camera origin at optical centre; z axis forward along optical axis, x to the right, y down (so $x = X_c/Z_c,\ y = Y_c/Z_c$). Pixel frame origin top-left, u right, v down; c_x, c_y measured in that pixel frame.

Projection (compact) —

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K\,[R \mid t] \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}.$$

Assumptions / notes — $s \approx 0$; often $f_x \approx f_y$ (square pixels); undistort images or include distortion in reprojection; resizing/cropping scales K; essential decomposition gives t up to scale.

Estimation (brief) — Intrinsics: Zhang (planar) + BA; Extrinsics: PnP (RANSAC) or essential decomposition; joint: SfM/BA.

What are the four distinct reference systems and three sequential transformations that define the complete camera projection pipeline?

- Four reference frames (global → sensor):
  1. **World frame:** Absolute 3D coordinates.
  2. **Camera frame:** 3D coordinates relative to the camera optical centre (pose given by $T = [R \mid t]$).
  3. **Image plane (metric):** Continuous 2D coordinates on the projection surface (often modeled as a virtual plane at focal distance f).
  4. **Pixel frame:** Discrete 2D integer coordinates on the digital sensor (origin typically top-left).
- Three sequential transformations:
  1. **World → Camera (3D → 3D):** Apply extrinsics $T = [R \mid t]$: $X_c = R\,X_w + t$.
  2. **Camera → Image plane (3D → 2D continuous):** Perspective projection (similar triangles): $(x, y) = \left(\frac{fX_c}{Z_c}, \frac{fY_c}{Z_c}\right)$ (or normalized $(X_c/Z_c,\, Y_c/Z_c)$).
  3. **Image plane → Pixels (2D continuous → 2D discrete):** Apply intrinsics K:
  $$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}, \quad K = \begin{bmatrix} f_u & s & u_0 \\ 0 & f_v & v_0 \\ 0 & 0 & 1 \end{bmatrix},$$
  where s (skew) is usually 0, and f_u, f_v are focal lengths in pixels.
- **Compact homogeneous form:**
$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K\,[R \mid t] \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}.$$

What are the mathematical operations that make camera projection strictly non-invertible, and what compromises are required to partially invert the system?

- **1. The 3D → 2D Projection:** The division by Z permanently collapses the depth dimension.
- **2. Pixel Quantization:** Converting continuous metric measurements into discrete integer pixel bins inherently destroys sub-pixel accuracy.

To invert the projection (map a pixel back to the real world), we must mathematically accept a **3D direction vector (a ray)** instead of a single point, provide external structural constraints (e.g., assuming the point lies on a known floor plane), and strictly neglect the quantization error (acceptable only for high-resolution sensors).

Lecture 18: Thin lens model and image mapping

What is the primary physical limitation of the pure pinhole camera model, and why do real cameras introduce a physical lens?

A pure pinhole camera forces a severe physical trade-off between **image sharpness and light intensity**. A tiny pinhole creates a perfectly sharp image but lets in almost no light, while a larger pinhole lets in enough light but severely blurs the image.

Introducing a lens solves this by allowing the camera to capture a massive amount of light (large aperture) while actively bending the rays to converge and maintain sharp focus.

Note: the **aperture size** dictates the light wave diffraction that causes image blur. Smaller apertures increase diffraction spread and thus image blur; it is not caused by the lens thickness. To reduce diffraction, use a wider aperture (lower f-number).

In the thin lens model, what mathematical condition determines whether an object is in focus, and what geometric phenomenon creates a "circle of confusion"?

Focus is strictly governed by the **thin lens equation**:

$$\frac{1}{d_0} + \frac{1}{d_1} = \frac{1}{f}$$

where d_0 is object distance, d_1 is image (sensor) distance, and f is the focal length. Only light rays originating from the exact spatial distance d_0 will converge perfectly to a single point on the sensor. Light rays from objects at any other distance will converge either in front of or behind the sensor. When these unconverged rays hit the sensor plane, they form a blurred, circular patch of light known as a **circle of confusion**. Introducing a physical barrier (stopping down the aperture) artificially narrows the light cone, reducing the diameter of this circle and expanding the depth of field.

If the lens focuses light at a plane that does not coincide with the sensor, rays from a single object point will meet either in front or behing the sensor. When these unconveyed rays hit the sensor they form a blurred disk, which diameter grows with the distance between the actual convergence plane and the sensor (and with aperture size).

How is radial distortion mathematically modeled and corrected in standard computer vision pipelines (like OpenCV)?

Radial distortion is modeled as a polinomial in $r^2 = x^2 + y^2$ applied to normalized image coordinates, often augmented with additional tangential distorion terms. This polynomial form arises from a Taylor expansion approximation of the non-linear distortion as a function of the radial distance r from the optical center.

The forward distortion model (ideal $\rightarrow$ distorted) is:

$$x_{corr} = x \cdot (1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$
$$y_{corr} = y \cdot (1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

(In the full model, tangential distotion with coefficients p_1, p_2 is also included.) In practice, undistortion (distorted $\rightarrow$ ideal) is not obtained by analytically inverting this polynomial. Instead, inverse mapping is used, typically via iterative methods or precomputed lookup maps.

What is the physical optical mechanism behind chromatic aberration, and how does it manifest in the final image?

Chromatic aberration is caused by **optical dispersion**. The refractive index of the glass lens is not a constant; it depends on the wavelength of the incoming light.

As a result, the focal length $f(\lambda)$ varies slightly for different colors (e.g., blue light bends at a steeper angle than red light). This failure to focus all color wavelengths at the exact same convergence point results in noticeable **color fringes** (unnatural purple or green halos) along high-contrast structural edges, particularly near the outer boundaries of the image.

When performing inverse projection (mapping a 2D pixel back to a 3D physical ray) with a non-ideal camera, how must the computational pipeline be sequenced?

Inverse projection operates on undistorted, normalized image coordinates. The correct pipeline is:

$$\text{pixel space} \rightarrow \text{undistorted pixel space} \rightarrow \text{normalized coordinates} \rightarrow \text{3D ray}$$

1. **Undistortion:** Map the observed (distorted) pixel (u_d, v_d) to the undistorted pixel (u, v) using the camera distortion model (radial and tangential).

2. **Normalization / Intrinsics removal:** Convert to normalized image coordinates:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = K^{-1} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

3. **Ray construction:** The direction $(x, y, 1)$ defines the 3D ray in camera coordinates; it can be transformed to world coordinates via the extrinsic parameters.

Note: Do not apply K^{-1} directly to distorted pixels. Undistortion (typically via inverse mapping or lookup tables) must be performed first.

What are the two fundamental steps involved in applying a geometric transformation to an image?

- **Coordinate transformation:** This mathematical step works on ideal geometrical points, defining exactly how the spatial coordinates change (e.g., $(x', y') = T\{(x, y)\}$).
- **Mapping / Resampling:** Because digital images are discrete grids, this step goes back to the pixels to determine how to correctly assign actual pixel intensity values to the new discrete grid locations after the geometrical points have been mathematically moved.

How do forward mapping and backward (inverse) mapping fundamentally differ in their approach to generating a transformed image?

- **Forward Mapping:** This is the direct application of the transformation. It maps points from the *source* image grid directly onto the destination grid.
- **Backward (Inverse) Mapping:** This inverts the process by iterating through every pixel in the *destination* image and mapping it backwards to find its corresponding original location in the source image.

Why is inverse mapping preferred over forward mapping when applying geometric transformations to digital images?

Since forward mapping (source $\rightarrow$ destination) is continuous but the image grid is discrete, it introduces two issues after discretization:

- **Holes:** some destination pixels receive no value because mapped source coordinates do not cover every integer grid location.
- **Collisions (overwrites):** multiple source pixels can map to the same destination pixel due to rounding of the continuos coordinates, causing loss of information.

These artifacts arise because continous coordinates must be assigned to integer pixel indices.

Inverse mapping (destination $\rightarrow$ source) avoids both issues by iterating over every destination pixel and sampling its corresponding (generally non-integer) location in the source image. This guarantees full coverage and eliminates overwriting by avoiding collisions, with values computed by interpolation (e.g., bilinear or bicubic).

Both issues can be solved also by using advanced forward-mapping schemes: each source pixel is distributed over nearby destination pixels using a kernel, with contributions accumulated and normalized. This is called splatting and is complex and computationally heavier.

In the context of inverse mapping, why is interpolation necessary and what are some common techniques?

When a destination pixel is mathematically mapped backward, its computed coordinates usually fall between discrete pixel centers in the source image (e.g., floating-point values instead of integers). Interpolation is required to mathematically mix or estimate the pixel intensity values based on the surrounding area in the source image.

Standard interpolation techniques include **Nearest Neighbor**, **Bilinear**, and **Bicubic** interpolation.

How can Look-Up Tables (LUTs) optimize the geometric mapping process for large datasets or video streams?

If the exact same spatial mapping is applied to a large number of images, it is computationally meaningless to repeatedly evaluate the transformation matrix for every single pixel.

Instead, the exact floating-point source locations for every destination pixel can be precomputed once and saved into a Look-Up Table (typically one map for X coordinates and one map for Y coordinates). The mapping process then becomes an extremely fast memory read operation.

Lecture 19: Real cameras

What geometric properties are strictly preserved and which are lost during a standard 3D to 2D perspective projection?

Perspective projection preserves incidence (collinearity of points on a line) and the cross-ratio of collinear points. It maps straight 3D lines to straight lines in the image plane, but does not preserve angles, Euclidean distances, or parallelism. Projections of parallel 3D lines may intersect in the image at a vanishing point.

In the hierarchy of geometric transformations, what defines a projective transformation in terms of matrix dimensions and degrees of freedom?

A 2D projective transformation (homography) is represented by a 3×3 matrix $\tilde{H}$ defined up to scale, resulting in 8 degrees of freedom. In 3D, projective transformations are represented by 4×4 matrices defined up to scale, resulting in 15 degrees of freedom.

Why cannot lengths be trusted in an image acquired via perspective projection?

Because image coordinates scale inversely with depth. Under the pinhole model,

$$x = \frac{fX}{Z},$$

so identical physical lengths project to different image lengths depending on their distance Z from the camera. Recovering metric lengths therefore requires depth information or camera calibration.

What are the inherent trade-offs when optimizing the aperture size of a purely pinhole camera?

A smaller aperture reduces geometric blur but increases diffraction effects. A larger aperture increases light throughput but introduces greater geometric blur (circles of confusion). The optimal pinhole diameter approximately satisfies $d \approx 1.9\sqrt{f\lambda}$, where λ is the wavelength of light (e.g., $\lambda \approx 550$ nm for green light).

How does changing the focal length and the camera viewpoint simultaneously to maintain the same subject framing affect perspective?

Maintaining the same framing requires adjusting camera distance when focal length changes. A short focal length combined with a close viewpoint exaggerates depth differences, while a long focal length combined with a distant viewpoint compresses depth, reducing apparent separation between foreground and background structures.

Why does a telephoto lens make it easier to isolate a background behind a subject?

Primarily due to increased magnification and a narrower field of view. For a given aperture and sensor size, telephoto lenses also produce a shallower depth of field, which enhances background blur and improves subject-background separation.

How does the thin lens model differ mechanically from a pinhole model regarding depth of field?

A pinhole camera provides an extremely large depth of field, limited mainly by diffraction. The thin-lens model introduces a finite aperture and focusing mechanism, enabling control over the focal plane and therefore the depth of field.

What is the correct mathematical relationship between f-stop values, aperture diameter, and depth of field?

The f-number N is defined as

$$N = \frac{f}{D},$$

where f is the focal length and D is the aperture diameter. A lower f-number corresponds to a larger aperture and generally a shallower depth of field. However, depth of field also depends on focal length, subject distance, and the acceptable circle of confusion.

At a fundamental physical level, how do digital camera sensors capture light, and why are they inherently incapable of sensing color on their own?

Digital sensors operate by converting incoming photons into electrons. The photodiodes measure the total energy (the total count of electrons generated) but fundamentally cannot differentiate between the specific wavelengths (photon energy) of the incoming light. Because they strictly measure light intensity regardless of wavelength, base CCD and CMOS sensors are inherently grayscale devices.

What are the primary architectural approaches used to extract full color (RGB) information from inherently grayscale sensors?

- **3-chip color sensors:** These use a trichroic prism to physically split the incident polychromatic ("white") light into its red, green, and blue components. Each color beam is directed to its own dedicated sensor, ensuring every spatial pixel receives full RGB data without any need for interpolation.
- **Single-chip with filters (Bayer Pattern):** A Color Filter Array (CFA) or mosaic is placed over a single sensor, where each individual pixel is covered by only one color filter (Red, Green, or Blue). Because each pixel only records a single color, this method requires mathematical interpolation to estimate the missing two color channels for each pixel to provide complete color info.
- **Single-chip with wavelength penetration (Direct Image Sensors):** These exploit the physical property of silicon where different light wavelengths penetrate to different depths. This allows stacked sensors to capture full RGB data at every single pixel location simultaneously.

How do direct image sensors (like Foveon) structurally bypass the limitations of the Bayer pattern without requiring the bulky prism optics of a 3-chip system?

Instead of placing a mosaic of color filters side-by-side horizontally (which causes spatial color information loss), direct image sensors stack the color-sensitive photodiodes vertically at each pixel location. This design leverages the physical principle that light penetrates silicon at depths dependent on its wavelength (for example, blue light is absorbed near the surface, while red light penetrates deeper). Because of this vertical stacking, complete RGB information is captured at every single photosite, meaning there is no information loss and entirely eliminating the need for mathematical color interpolation.

What structural flaw in traditional image sensors does the Backside Illuminated (BSI) architecture solve, and how does it physically achieve this?

In standard (front-illuminated) sensors, the metal wiring and circuitry are layered physically above the light-sensitive photodiodes. This creates a structural barrier that causes partial obscuration, blocking or reflecting some of the incoming light before it can reach the diode. A BSI sensor flips this architecture, placing the photodiodes directly at the primary light-receiving surface and moving all the metal wiring to the "back" (underneath the diodes). This unobstructed path allows the sensor to capture light much more efficiently, although this backside-illuminated structure is significantly more difficult to manufacture.

Are camera sensors strictly limited to rectangular, grid-like arrays of pixels?

No, while rectangular arrays are the standard for capturing static 2D scenes, sensor technologies (both CCD and CMOS) can also be manufactured as "just a line" of pixels. These 1D line-scan sensors are specifically utilized for analyzing moving objects, operating by continuously scanning a scene as it passes the sensor field of view.

Given that real physical lenses consist of multiple complex glass elements, why is it still mathematically valid to model them using the idealized pinhole camera model?

Despite the internal complexity of modern lenses (which include multiple focusing, stabilization, and corrective glass elements), the entire optical system is engineered to converge light rays through a single effective optical center. By utilizing an aperture diaphragm within this array, the system effectively mimics the behavior of a pinhole. Therefore, the complex camera can still be geometrically treated as a simple pinhole model where all recorded light rays pass through a single convergence point.

Why is a camera's aperture almost universally expressed as an f-number (e.g., f/2.0, f/4) rather than a direct metric measurement of the physical opening diameter (e.g., 25mm)?

The f-number ($N = f/A$) normalizes the light-gathering capability of a lens, making it completely independent of the lens's physical focal length.

According to the inverse-square law, if you double a lens's focal length (f), the projected image becomes twice as large, spreading the same amount of light over four times the sensor area (reducing the light intensity to $1/4$). To compensate and gather 4x the light, the physical aperture diameter must also double. By defining the aperture strictly as a ratio (f/N), any two lenses set to the exact same f-number will deliver the exact same exposure intensity to the sensor, regardless of their physical size.

What is the fundamental difference between a global electronic shutter and a rolling shutter, and what specific visual artifact does a rolling shutter introduce?

- **Global Shutter:** Captures the entire image sensor simultaneously, recording every single pixel at the exact same fraction of a second.
- **Rolling Shutter:** Scans and captures the image sequentially, typically row-by-row from top to bottom.

- **Visual Artifact:** Because different rows of the sensor are recorded at slightly different chronological moments, rapidly moving objects (such as a spinning airplane propeller or a fast-moving car) will appear geometrically warped, stretched, or skewed in the final image.

What is the principle of reciprocity in camera exposure, and how does it mathematically dictate the relationship between aperture and shutter speed?

Reciprocity dictates that the total amount of light exposing the sensor is proportional to the product of the aperture area and the exposure time.

Standard f-stops progress by a factor of $\sqrt{2}$ (which exactly halves or doubles the physical area of the opening), while shutter speeds progress by a factor of 2 (halving or doubling the time). Therefore, they are perfectly inversely proportional. If you halve your shutter speed to freeze motion, you must exactly double your aperture area (move down one f-stop) to maintain the exact same total light exposure.

When relying on the reciprocity principle to maintain a constant exposure, what are the primary visual and functional trade-offs when manipulating shutter speed?

- **Fast Shutter Speed (e.g., 1/1000s):** Functionally freezes fast-moving action (like a running person or a moving train) without blur. The trade-off is that it allows very little time for light to enter, requiring a much larger aperture to compensate.
- **Slow Shutter Speed (e.g., 1/15s):** Allows ample light to enter, which is useful in dark environments. The severe trade-off is that any movement from the subject—or even slight shaking of the photographer's hands—during that window will result in heavy motion blur.

Since shrinking the aperture reduces the circle of confusion and increases the depth of field, why is it optically counterproductive to simply use the absolute smallest aperture available (e.g., f/22) for maximum sharpness?

While stopping down the aperture initially increases sharpness by expanding the depth of field, pushing it to the physical extreme introduces two major problems:

- **Severe Light Loss:** It drastically restricts incoming light, forcing the use of impractically slow shutter speeds or noisy electronic gain (ISO).

- **Diffraction Degradation:** As light waves squeeze through a microscopic physical opening, they diffract (bend and spread out). This diffraction creates an unavoidable optical blur that fundamentally degrades the overall resolution and sharpness of the image, ultimately negating the benefits of the wide depth of field.

What optical phenomenon creates the "starburst" effect around bright point-light sources at very small apertures, and what mechanical feature dictates the specific geometry of the star?

The starburst effect is a direct result of **light diffraction** occurring against the straight mechanical edges of the aperture blades inside the lens. Because diffraction spreads light more noticeably through smaller openings, the effect only becomes prominent at tightly closed apertures (e.g., f/16 or f/22).

The geometric shape (the number of rays in the star) is strictly determined by the number of physical aperture blades:

- An **even** number of blades (e.g., 6) produces an identical number of visible rays (6 rays). This is because opposite blades are parallel to each other, causing their diffraction spikes to perfectly overlap.
- An **odd** number of blades (e.g., 7) produces exactly double the number of rays (14 rays) because the opposing diffraction spikes do not share the same axis and cannot overlap.

How does the physical size of the aperture dictate a camera's depth of field, and what role does the "circle of confusion" play in this relationship?

The depth of field (DoF) is the physical range (front-to-back) within a scene that appears acceptably sharp in the final image. While a lens can only achieve perfect, mathematical focus at one exact distance, objects slightly in front of or behind this distance form a blurred spot of light on the sensor known as the circle of confusion.

- When a **large aperture** (e.g., f/2) is used, the light rays pass through a wide opening, forming a broad cone. Because of the steep angle of these rays, the circle of confusion grows rapidly for objects outside the precise focus plane. This results in a shallow depth of field where foregrounds and backgrounds become heavily blurred.
- When a **smaller aperture** (e.g., f/16) is used, it physically blocks the outer light rays, forcing the remaining light into a much narrower cone. This restricts the size of the circle of confusion so that it remains negligibly small over a longer distance. As a result, objects across a much wider range of distances remain visually sharp, effectively increasing the overall depth of field.

Lecture 20: Camera calibration

What is the fundamental goal of camera calibration, and how does the requirement for a world coordinate system affect the parameters estimated?

Camera calibration aims to estimate the camera model parameters needed to predict how 3D points project to 2D image pixels.

Intrinsic calibration of a single camera estimates internal parameters of the camera +lens system (independent of pose), typically:

- focal lengths f_x, f_y (in pixels),
- principal point (c_x, c_y),
- skew (often assumed 0 for modern cameras),
- distortion coefficients (radial and tangential).

Extrinsics (rotation and translation) are always defined *with respect to a chosen reference frame*. If you define a global world coordinate system, you obtain extrinsics in that world frame. If no external world frame is given, you typically choose the calibration target's coordinate system as the reference; the estimated extrinsics are then relative to the target (still valid, but not tied to an absolute world).

Beyond simply fixing visual distortions, what are the primary engineering applications that make camera calibration absolutely necessary?

While "dedistortion" (straightening curved lines caused by lenses) is a key visual benefit, calibration is fundamentally required for any form of accurate metric measurement in computer vision. It allows a system to mathematically relate 2D image pixels back to the real 3D world. Examples include estimating 3D physical structures from camera motion, computing depth maps using a stereo camera rig, measuring the true physical dimensions of planar objects, and calculating distances to objects (like determining the distance to a person by leveraging known ground-plane constraints).

Why is a checkerboard widely considered the standard geometric pattern for camera calibration?

The calibration process requires an object with a strictly known shape and physical dimensions that can be easily and precisely located within an image. A checkerboard provides a grid of high-contrast, sharply defined squares. The alternating black-and-white intersections act as robust, exact 2D feature points (corners) that computer vision

algorithms can detect automatically with sub-pixel accuracy. This provides the reliable, unambiguous coordinate data needed to solve the projection equations.

How is the camera calibration process mathematically formulated as an optimization problem?

Calibration is framed as a **non-linear least squares minimization** problem. Given N images of the pattern and M known 3D corner positions P_j on the calibration target, the algorithm minimizes the **reprojection error** between measured image points $p_{i,j}$ and projected points produced by the camera model:

$$\min_{K,\, d,\, \{R_i, t_i\}} \sum_{i=1}^{N} \sum_{j=1}^{M} \left\| p_{i,j} - \pi\big(K,\, d,\, R_i,\, t_i,\, P_j\big) \right\|^2 .$$

Here K are the intrinsic parameters (containing f_x, f_y, c_x, c_y and optionally skew), d the distortion coefficients (radial and tangential), and (R_i, t_i) the extrinsics for image i. The projection function $\pi(\cdot)$ applies the pinhole projection followed by the distortion model.

Why is a "good initial guess" crucial for the calibration optimization process, and how does a planar pattern help achieve it?

The least-squares minimization operates over a massive, highly complex parameter space (intrinsics, extrinsics for every single image, and non-linear distortion coefficients). If the solver starts with arbitrary values, it is highly likely to fail or get stuck in a local minimum.

By using a strictly **planar** object (like a flat checkerboard) and temporarily neglecting distortion, the transformation from the 3D object plane to the 2D image plane can be heavily simplified into a **homography**. Calculating this homography provides a mathematically stable and highly accurate initial guess to safely jump-start the complex non-linear solver.

Theoretically, what is the absolute minimum number of checkerboard images required to calibrate the intrinsic parameters, and why is this number insufficient in practice?

For planar calibration methods based on homographies (e.g., Zhang's method), each distinct view provides **two independent constraints** on the intrinsic parameters.

If you assume a simplified intrinsic matrix with **4 unknowns** (typically by setting skew $= 0$), then **2 views** can be sufficient *in theory* because $2 \times 2 = 4$ constraints match the unknowns.

In the more general case with **5 intrinsic unknowns** (including skew, or without additional constraints), you need at least **3 views** to have $2 \times 3 = 6$ constraints.

This parameter counting also explains why you cannot just count "intrinsics + extrinsics" once: extrinsics are *per-image* parameters and are not shared across views.

In practice, the minimum is numerically fragile: noise in corner localization, near-degenerate viewpoints, and lens distortion make the solution unstable. Robust calibration typically uses **many** images (often 10+), with diverse tilts and positions that cover the full field of view.

If a single homography only requires 4 point correspondences to be solved (yielding 8 constraints), why do standard calibration checkerboards feature dozens of intersecting corners?

While only 4 points are mathematically required to define the planar homography itself, real-world lenses are not ideal. The additional dozens of corners spread entirely across the grid provide the dense, spatially distributed data points absolutely necessary to accurately measure and model non-linear **lens distortion** (both radial and tangential) across the camera's entire optical field of view.