Image Processing Lecture 5
Transformations, Pyramids, and Denoising

Yonghao Lee

1 2D Geometric Transformations

1.1 The Problem with Translation

In standard 2D Cartesian coordinates, linear transformations (scaling, rotation) map the origin $(0, 0)$ to $(0, 0)$.

$$\mathbf{x}' = \mathbf{A}\mathbf{x} \tag{1}$$

Translation, however, is an affine operation ($\mathbf{x}' = \mathbf{x} + \mathbf{t}$), which shifts the origin. This prevents us from composing multiple operations into a single matrix multiplication.

1.2 Homogeneous Coordinates

To solve this, we embed 2D points into 3D projective space by adding a coordinate w.

- **Euclidean Point:** $(x, y) \in \mathbb{R}^2$
- **Homogeneous Point:** $(x, y, 1) \in \mathbb{P}^2$

Note: (x, y, w) represents the same 2D point as $(x/w, y/w, 1)$.

1.3 Transformation Hierarchy

2 Image Warping

2.1 Forward vs. Inverse Warping

Forward Warping (Bad)

Iterate source $(x, y) \to$ calc dest (x', y'). Can cause holes/cracks.

Inverse Warping (Standard)

Iterate dest $(x', y') \to$ calc source $(x, y) = T^{-1}(x', y')$. Requires interpolation.

2.2 Bilinear Interpolation

For float coordinates (x, y), blend the 4 nearest neighbors using fractional distances a, b:

$$f(x, y) \approx (1-a)(1-b)f(i, j) + a(1-b)f(i+1, j) + (1-a)bf(i, j+1) + abf(i+1, j+1) \tag{2}$$

Name	Matrix Form	DOF	Preserves
Translation	$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$	2	Orientation, lengths, angles, parallelism.
Rigid (Euclidean)	$\begin{bmatrix} \cos\theta & -\sin\theta & t_x \\ \sin\theta & \cos\theta & t_y \\ 0 & 0 & 1 \end{bmatrix}$	3	Lengths, angles, areas, straight lines.
Similarity	$\begin{bmatrix} s\cos\theta & -s\sin\theta & t_x \\ s\sin\theta & s\cos\theta & t_y \\ 0 & 0 & 1 \end{bmatrix}$	4	Angles, ratios of lengths (Shape).
Affine	$\begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix}$	6	Parallelism, ratio of areas, straight lines.
Projective	$\begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{bmatrix}$	8	Straight lines (collinearity). **No** parallelism.

3 Feature Detection: Harris Corners

3.1 Corner Detection Intuition

We shift a window W by (u, v) and measure the Sum of Squared Differences (SSD), $E(u, v)$.

- **Flat:** E constant. **Edge:** E changes in 1 direction. **Corner:** E changes in all directions.

3.2 Structure Tensor (M)

Using Taylor expansion, $E(u, v) \approx [u, v]M[u, v]^T$, where:

$$M = \sum_{(x,y)\in W} w(x, y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \tag{3}$$

Eigenvalues λ_1, λ_2 of M determine the region type. Large $\lambda_1, \lambda_2 \implies$ Corner.

3.3 Harris Response

Avoids eigenvalue computation: $R = \det(M) - k \cdot (\text{trace}(M))^2$. Threshold R and perform Non-Maximum Suppression (NMS).

4 Image Pyramids

Pyramids are a multi-scale representation of an image. They enable efficient visual search (coarse-to-fine), compression, and advanced blending. The memory cost is only $\approx 1.33\times$ the original image.

4.1 Basic Operations

Reduce (Going Down)

To halve the resolution, we must filter first to prevent aliasing.

1. **Blur:** Convolve with a low-pass filter (Kernel).
2. **Subsample:** Keep every 2nd pixel in every 2nd row ($[::2, ::2]$).

The Kernel: A 5-tap approximation of a Gaussian: $\frac{1}{16}[1, 4, 6, 4, 1]$. It is separable, meaning we can blur rows then columns for efficiency ($2N$ vs N^2 ops).

Expand (Going Up)

To double the resolution (for interpolation/reconstruction).

1. **Zero Pad:** Insert zeros between every pixel (Upsampling).
2. **Blur:** Convolve with the same kernel, but multiplied by 4 to compensate for the brightness loss caused by zeros.

4.2 Gaussian vs. Laplacian Pyramids

Gaussian Pyramid (G): A sequence $G_0, G_1, \ldots, G_n$ where G_0 is the original image and $G_i = \text{Reduce}(G_{i-1})$. Represents the image at different scales.

Laplacian Pyramid (L): Stores the **details** (high frequencies) lost in the reduction step.

$$L_i = G_i - \text{Expand}(G_{i+1}) \tag{4}$$

The top level is the exception: $L_n = G_n$ (The "residue" or low-frequency base). The original image can be reconstructed perfectly by collapsing the pyramid:

$$G_i = L_i + \text{Expand}(G_{i+1}) \quad \Longrightarrow \quad G_0 = \sum L_i \text{ (conceptually)} \tag{5}$$

4.3 Application: Multiresolution Blending

Merging images A and B directly creates a visible seam. Pyramids allow blending different frequencies over different spatial distances.

1. Build Laplacian Pyramids L_A and L_B.
2. Build a **Gaussian Pyramid** for the Mask M (denoted G_M). This creates a mask that is sharp at high resolutions and blurry at low resolutions.
3. Blend at every level k:

$$L_{out}(k) = G_M(k) \cdot L_A(k) + (1 - G_M(k)) \cdot L_B(k) \tag{6}$$

4. Collapse L_{out} to get the result.

This ensures sharp details (texture) blend quickly, while coarse details (light/color) blend smoothly.

5 Image Denoising

5.1 Problem Formulation

We model the noisy image y as the true signal x plus noise n:

$$y(i,j) = x(i,j) + n(i,j) \tag{7}$$

We assume **AWGN**: Additive White Gaussian Noise, where $n \sim \mathcal{N}(0, \sigma^2)$.

Metrics:

- **MSE:** Mean Squared Error. $\frac{1}{N^2}\sum(I - \hat{I})^2$.
- **PSNR:** Peak Signal-to-Noise Ratio (in dB). $20\log_{10}\left(\frac{255}{\sqrt{MSE}}\right)$. High is better ($> 30$dB).

5.2 Temporal Denoising (The "Gold Standard")

If we have multiple images $y_1, \ldots, y_K$ of a **static** scene, we can average them. Since noise has zero mean ($E[n] = 0$), averaging K images reduces the noise variance by factor K:

$$\hat{x} = \frac{1}{K}\sum_{k=1}^{K} y_k \quad \Longrightarrow \quad \mathrm{Var}(\hat{n}) = \frac{\sigma^2}{K} \tag{8}$$

This is the principle behind Google's "Night Sight" (Burst Denoising), which aligns and averages multiple short-exposure frames.

5.3 Non-Local Means (NLM)

When we only have a **single** image, we exploit the **redundancy** of natural images. Patches (e.g., 5×5 windows) repeat throughout the image. Instead of averaging local neighbors (Gaussian Blur), which blurs edges, we average **similar patches** found anywhere in the image.

The Formula: For a pixel p, the denoised value is a weighted average of all other pixels q in the search window:

$$\hat{x}(p) = \frac{1}{C}\sum_{q} y(q) \cdot w(p,q) \tag{9}$$

The weights $w(p,q)$ depend on the similarity between the patch at p (N_p) and the patch at q (N_q), usually calculated via Sum of Squared Differences (SSD):

$$w(p,q) = e^{-\frac{||N_p - N_q||^2}{2\sigma^2}} \tag{10}$$

- If patch $N_q \approx N_p$, weight ≈ 1 (Contribution).
- If patch $N_q \neq N_p$ (e.g., edge vs flat), weight ≈ 0 (No blurring).

This preserves edges effectively while removing noise in smooth regions.