# Unsharp Mask (USM) Algorithm

## 1. Algorithm Principle

In any image, low frequencies correspond to large areas with slow changes in brightness (background of the image), while high frequencies correspond to details with rapid pixel value changes (textures, edges, noise).

1. First, apply smoothing: Use a Gaussian kernel Gσ to perform low-pass filtering on the original image I, resulting in a blurred version B = Gσ(I). This step essentially removes high frequencies, leaving only low-frequency information.

2. Create a detail mask: Compute the difference M = I - B. Since B approximates low frequencies only, the subtraction retains the high-frequency components (edges and textures).

3. Add the scaled difference back to the original: Multiply the mask by a weight k and add it to the original image. The formula is I\_s = I + k \* (I - B) = (1 + k) \* I - k \* B, producing a sharpened result with enhanced edges and nearly unchanged overall brightness.

This entire process can be seen as 'first reducing the large-scale tones and then adding back the reduced amount in multiples' — essentially a high-boost filter.

## 2. Mathematical Expression

Let the original image be I, and the low-pass blurred result be Gσ(I). The standard formula for USM is:

I\_s = (1 + k) \* I - k \* Gσ(I)

where k ∈ [0, +∞) represents the sharpening strength (amount). When k = 0, no sharpening occurs; commonly, k ≈ 0.5 - 2.0. When k = 1, I\_s = 2I - Gσ(I), also known as high-boost filtering.

• kernel\_size / σ (radius) — determines the degree of blurring: The larger the radius, the wider the low-frequency range that is removed, allowing larger edges to be emphasized. However, too large a radius may cause halos.

• amount (k) — the amount of detail enhancement: A larger value sharpens more strongly, but may lead to noise and jaggedness.

• threshold (M) — a threshold applied to the difference |I - Gσ(I)|: Sharpening is only applied when local contrast exceeds the threshold, helping to avoid amplifying noise in flat regions.

## 3. OpenCV Implementation

1) Read the image and convert it to floating point type;

2) Use cv.GaussianBlur to apply Gaussian blurring, with kernel size set to (0, 0), automatically inferring σ;

3) Use cv.addWeighted for linear blending: dst = α \* src + β \* blur + γ, where α = 1 + k, β = -k;

4) Clip the result to the range 0–255 and convert back to uint8.