Lowpass and Bandpass Image Pyramids

Carlo Tomasi

January 11, 2019

It is often useful to analyze an image $I(\mathbf{x})$ at different scales. One can then form a stack of images obtained by repeatedly blurring the input image:

$$\begin{aligned} B_0 &= I \\ B_\ell &= B_{\ell-1} * S_\sigma \quad \text{for} \quad \ell = 1, \ldots, L \end{aligned} \tag{1}$$

where $S_\sigma(\mathbf{x})$ is a smoothing kernel, typically a Gaussian with width σ, and the symbol '$*$' denotes convolution. The larger σ, the more high-frequency information (that is, fine detail) is suppressed at every level of smoothing, and analysis of B_ℓ reveals finer or coarser structures in the image depending on the value of the *level* ℓ. Figure 1 shows the result of blurring an input image $L = 7$ times with a Gaussian kernel with $\sigma = 2$ pixels. Note the loss of detail at higher levels ($\ell > 0$) of the stack.

Figure 1: A Gaussian stack.

The convolution of a Gaussian with parameter σ_1 with another Gaussian with parameter σ_2 is a Gaussian with parameter $\sigma = \sqrt{\sigma_1^2 + \sigma_2^2}$. Because of this, convolving an image ℓ times with a Gaussian with parameter σ is the same as convolving the same image once with a Gaussian with parameter

$$\sigma_\ell = \sqrt{\ell}\sigma$$

for each ℓ. So we can also write

$$\begin{aligned} B_0 &= I \\ B_\ell &= B_0 * S_{\sigma_\ell} \quad \text{for} \quad \ell = 1, \ldots, L \; . \end{aligned}$$

Of course, the iterative smoothing procedure (1) is more efficient, because the kernels are smaller.

The pixel resolution of the images in the stack is high when compared to the spatial frequencies contained in the images for $\ell > 0$: There are many pixels even if the image brightness changes slowly. Because of this, the blurred images B_ℓ can be sampled after filtering without significant loss of information. Without getting into the quantitative aspects of sampling and image bandwidth, it turns out that most of the image information is preserved if every time the image is blurred with a Gaussian filter with parameter σ, the image is subsampled by a factor of about $\sigma/1.6$.

When $s = \sigma/1.6$ is an integer number, it is clear what this means: Filter with a Gaussian with parameter σ, then retain every s-th pixel in each dimension. When s is not an integer, on the other hand, sampling "every s pixels" entails retrieving image values between the values available in the image array. This can be done by *sub-pixel interpolation*, which requires a model for the continuous image that the array values are samples of. One of the simplest such models is the *bilinear* one, in which the underlying continuous image $I(\mathbf{x})$ is assumed to be separately linear in x and y, the two components of $\mathbf{x}$, between integer values of the coordinates. This model leads to *bilinear interpolation*: Let $\mathbf{x} = (x, y)$, and (with $\lfloor \cdot \rfloor$ denoting the floor function)

$$\begin{aligned} \xi = \lfloor x \rfloor \quad , \quad \eta = \lfloor y \rfloor \\ \Delta x = x - \xi \quad , \quad \Delta y = y - \eta \; . \end{aligned}$$

Then,

$$\begin{aligned} I(\mathbf{x}) &= I(\xi, \eta)\,(1 - \Delta x)\,(1 - \Delta y) \\ &+ I(\xi + 1, \eta)\,\Delta x\,(1 - \Delta y) \\ &+ I(\xi, \eta + 1)\,(1 - \Delta x)\,\Delta y \\ &+ I(\xi + 1, \eta + 1)\,\Delta x\,\Delta y \; . \end{aligned}$$

We can now sample the image I with any sampling period, integer or otherwise.

We encapsulate the operations of filtering followed by sampling into a single function

$$B = \texttt{resize}(I, \phi)$$

where the *downsampling factor* $\phi = 1/s$ is a positive real number that denotes the ratio between the size of B and that of I. For values $0 < \phi < 1$, the image shrinks. For $\phi > 1$, no filtering is performed, and the image grows larger. The filter in downsampling is Gaussian with parameter

$$\sigma = 1.6/\phi \; .$$

Figure 2 shows an example of the effects of resizing an image with and without smoothing. To make the differences more obvious, a large sampling factor was used. This Figure shows that if an image is sampled without first blurring it, neighboring pixels in the reduced-size image may com from parts of the image that have nothing to do with each other in the scene, and this leads to the disorganized appearance in panel (b) of the Figure. Blurring ensures that each sampled pixel summarizes the average values in an image

(a)

(b)

(c)

Figure 2: An illustration of aliasing. The image in (a) was just resampled in (b) and instead downsized by blurring and resampling in (c). The original image has 2448×3264 pixels, and is not shown to scale with the other two images. For both (b) and (c), the sampling factor is $\phi = 1/30$.

neighborhood that matches the sampling factor. While the reduced-size image in (c) is blurred relative to the original in (a), whatever information is left is more clearly related to the original. The phenomenon that degrades the image in (b) is called *aliasing* in the literature.

Replacing convolution with S_σ with $\texttt{resize}(\cdot, \phi)$ where $0 < \phi < 1$ in equation (1) yields the *Gaussian pyramid*:

$$\begin{aligned} G_0 &= I \\ G_\ell &= \texttt{down}(G_{\ell-1}) \quad \text{for} \quad \ell = 1, \ldots, L \, . \end{aligned} \tag{2}$$

In the last expression, we think of fixing ϕ to some value between 0 and 1 (for instance, $\phi = 1/2$) and define

$$\texttt{down}(X) = \texttt{resize}(X, \phi) \, .$$

We will later also need

$$\texttt{up}(X) = \texttt{resize}(X, 1/\phi)$$

where `down` and `up` use the same value of ϕ. These two operations are called *downsampling* and *upsampling*. There is a crucial difference between the two: Downsampling blurs the input image with a Gaussian and then samples it by bilinear interpolation to make it smaller. Upsampling merely resamples the input image on a finer grid, but it does not undo the blurring, and the extra pixel values in $\texttt{up}(X)$ are merely "made up" by bilinear interpolation. So X and $\texttt{up}(X)$ contain the same frequencies (same level of detail), but the latter has more pixels than the former.

Since the image shrinks at each level, it is no longer necessary to specify the maximum level L: once the image shrinks to a single pixel (about $\lfloor \log_{1/\phi}(\min(R, C)) \rfloor - 1$ steps, where R and C are the number of rows and columns of I), the procedure stops.

Figure 3 shows the Gaussian pyramid for the same input image used for the Gaussian stack in Figure 1 and for $\phi = 1/2$. The input image has $R = 365$ rows and $C = 384$ columns, and the pyramid has $L = 7$ levels (plus the input image itself).

The Gaussian pyramid is said to be a *lowpass* pyramid, in that every level contains all the image frequencies *below* some value, roughly proportional to ϕ^ℓ. In contrast, the *Laplacian pyramid* is a *bandpass* pyramid, in that every level contains the image frequencies *around* a value that is roughly proportional to ϕ^ℓ.

The Laplacian pyramid contains exactly the same information as the input image, and the input image can be reconstructed *exactly* (up to numerical rounding) from the pyramid. The Laplacian pyramid takes two consecutive images G_ℓ and $G_{\ell+1}$ from the Gaussian pyramid, upsamples $G_{\ell+1}$, and makes H_ℓ the difference between G_ℓ and the upsampled $G_{\ell+1}$:

$$H_\ell = G_\ell - \texttt{up}(G_{\ell+1}) \, .$$

The image G_ℓ contains frequencies below $F\phi^\ell$ (where F is the highest frequency in the original image I), while $\texttt{up}(G_{\ell+1})$ (or $G_{\ell+1}$, which has the same information but the wrong size) contains frequencies below $F\phi^{\ell+1}$, so their difference H_ℓ contains frequencies between $F\phi^{\ell+1}$ and $F\phi^\ell$, that is, the image detail at frequencies in this band. This process is repeated until, after L levels, one obtains a bandpass image H_L. An additional, tiny lowpass image G_{L+1} is computed as $\texttt{down}(G_L)$ and stored with the Laplacian pyramid.

To reconstruct the original image from the Laplacian pyramid, we note that G_ℓ can be reconstructed exactly from $G_{\ell+1}$ and H_ℓ, since the last equation can be solved for G_ℓ to obtain he following:

$$G_\ell = H_\ell + \texttt{up}(G_{\ell+1}) \, .$$

Figure 3: A Gaussian pyramid.

Here then is the algorithm for computing the Laplacian pyramid (see Figure 5 (left)). First, set

$$G \leftarrow I \ .$$

Then, while G' is large enough, repeat the following for $\ell = 0, 1, \ldots$:

$$\begin{aligned} G' &\leftarrow \texttt{down}(G) \\ H_\ell &\leftarrow G - \texttt{up}(G') \\ G &\leftarrow G' \ . \end{aligned}$$

The remaining image G is the lowpass residual G_{L+1}.

Figure 4 shows the Laplacian pyramid for the same image input image as in Figure 1 and for $\phi = 1/2$. To reconstruct the image from the Laplacian pyramid, we work backwards:

$$I \leftarrow G_{L+1}$$

and for $\ell = L, \ldots, 0$:

$$I \leftarrow \texttt{up}(I) + H_\ell$$

A small *caveat*: because of rounding, $\texttt{up}(I)$ may not yield exactly the desired image size. Because of this, we overload the definition of the `up` function to take as second argument the desired size of the output image instead of the scaling factor, and the last assignment becomes

$$I \leftarrow \texttt{up}(I, \texttt{size}(H_\ell)) + H_\ell \ .$$

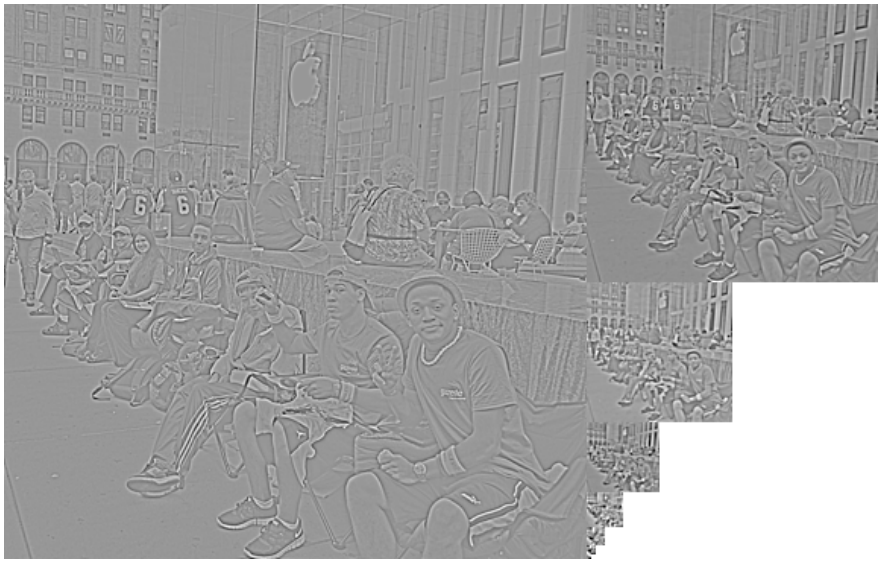

Figure 4: A Laplacian pyramid. Pixel values are positive and negative, and gray denotes zero.

Figure 5 shows computation of the Laplacian pyramid and reconstruction of the image from it in schematic form.

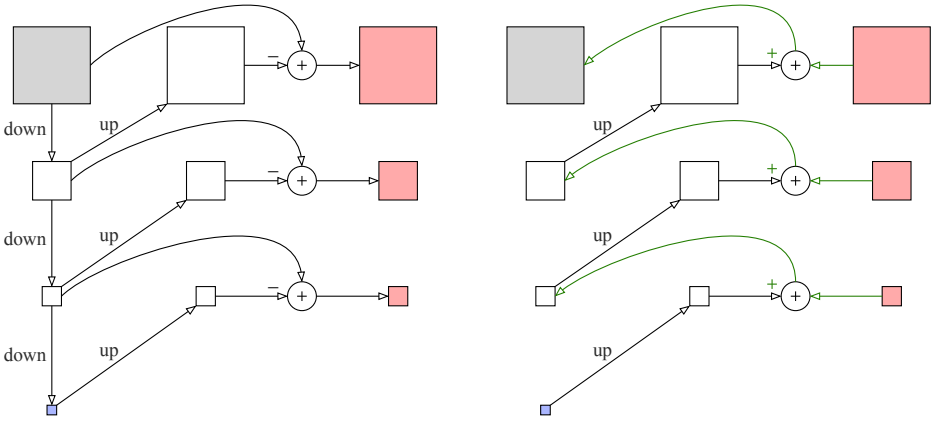

Figure 5: Construction of the Laplacian pyramid (left) and its inverse (right). The column of `down` operators on the left in the left picture computes a Gaussian pyramid, which is however discarded, except for the very last image.