CS348V Winter 2018 Assignment 2
Notes on Local Laplacian Filtering

In this assignment you will need to implement *Fast Local Laplacian Filtering*. The papers cited at the end of this document provide a comprehensive explanation of the methods, but we've included a short summary to help guide your implementation.

Gaussian and Laplacian Pyramids

Given an input image I, the *Gaussian pyramid* is a set of images $\{G_0, G_1, \ldots, G_n\}$ of progressively lower resolutions. Specifically it is defined by the recurrence, $G_0 := I$ and $G_{l+1} = \text{Downsample}(G_l)$. Downsample$(\cdot)$ produces an image of half the width and half the height, and mimics a low-pass filter operation. You are free to choose the specific convolution kernel for the Downsample operator, but we recommend a separable filter. A common choice is a 4×4 pixel kernel with weights according to the tensor ww^T where $w = [1/8, 3/8, 3/8, 1/8]$.

The *Laplacian pyramid* is a related operation, in which levels of the pyramid are *differences* between levels of the Gaussian pyramid. Specifically, $L_l = G_l - \text{Upsample}(G_{l+1})$ and $L_n := G_n$. The Laplacian pyramid mimics a band-pass filter, where each level of the Laplacian pyramid represents a frequency band of the original image. The Upsample$(\cdot)$ operator should be the transpose of the Downsample operator used above (and produce an image of twice the resolution in each dimension). Note that it is possible to *reconstruct* the original image from its Laplacian pyramid. That is, if we define the recurrence $I_l = L_l + \text{Upsample}(I_{l+1})$ and $I_n := L_n$, then we should have $I_0 \equiv I$, where I is the original image. This is often called "collapsing" the Laplacian pyramid.

Local Laplacian Filtering

With these pyramidal constructs, we can define a new image editing technique referred to as Local Laplacian filtering. Given input image I, let $G[I]$ and $L[I]$ be the Gaussian and Laplacian pyramids of the image, respectively. Let us also index into the pyramids as follows: $G[i](l, x, y)$ is the pixel at location (x, y) in the l-th level of the pyramid G. For now, we assume we are operating on single-channel images (e.g. if we operate on RGB channels independently, or a single luminance channel).

Local Laplacian filtering generates an output image by first constructing its Laplacian pyramid, then flattening this pyramid to get the final full resolution output image. The algorithm constructs the Laplacian pyramid of the output by means of a series of spatially localized edits to I. For each (l, x, y) in $G[I]$, the algorithm computes a modified image I' by remapping the original full-resolution image I using a parameterized *remapping function* $r(I; G[I](l, x, y))$. In other words , the modification to I is determined by value of a coefficient in I's Gaussian pyramid (which measures local image properties around (x, y)).

For each unique value in the Gaussian pyramid $G[I]$, we compute a new remapped image I' and compute the Laplacian pyramid of $L[I']$. **The value at $L[I'](l, x, y)$ is then copied into Laplacian pyramid of the output image.** (The output takes on characteristics of the remapped image in this spatial region and at the current frequency band.) Once we have done this operation for all (l, x, y), we collapse the Laplacian pyramid of the output image to obtain the final filtered output (which is a full-resolution, modified version of the original image I). The following pseudocode summarizes the algorithm:

Algorithm 1 Local Laplacian Filtering

$L' \leftarrow \{\}$
for each (l, x, y) in G **do**
  $g := G(l, x, y)$
  $I' := r(I; g)$
  $L'(l, x, y) \leftarrow L[I'](l, x, y)$
return Reconstruct(L')

Two issues remain: (1) we need to decide on an appropriate remapping function $r()$, and (2) the algorithm proposed above has high cost. It computes a new Laplacian Pyramid for all pixels of the Laplacian pyramid of the output!

- For selecting the remapping function r, we refer you to Section 4 of Paris et. al, 2011.
- For addressing, cost, see below:

Fast Local Laplacian Filtering

The fast local Laplacian filtering method is very similar in nature and structure to the original method. The major difference is that, rather than recomputing the Laplacian pyramid of a remapped image for every output pixel, we sample the space of the remapping functions r using a small number of samples (e.g. 8) and interpolate between the Laplacian pyramids that result from these samples. In other words, we precompute of Laplacian pyramids for images that result from remapping I based on different values in the range of G[I](l, x, y). Then, rather than perform remap I and compute a unique Laplacian pyramid for each output pixel, fast local Laplacian filtering simply *approximate* the required Laplacian pyramid by interpolating two of the precomputed ones.

Specifically, if the values in the original Gaussian pyramid $G[I]$ span the range $[g_{\text{min}}, g_{\text{max}}]$, we construct the set $\{g_0, g_1, \ldots, g_k\}$ where $g_i < g_{i+1}, g_0 := g_{\text{min}}, g_k := g_{\text{max}}$. For each g_i we define $I'_i := r(I'; g_i)$ and $L'_i \leftarrow L[I'_i]$. (Notice we have constructed a set of $k+1$ Laplacian Pyramids, one for each remapped image sampling the range of the original Gaussian Pyramid. Then, for each (l, x, y), if we define $g := G[I](l, x, y)$, we find the g_i closest to g and linearly interpolate between the Laplacian pyramid coefficients from L_i and $L_{i\pm1}$ (the $\pm$ coming from the which side of the relevant interval g is closer to).

Notice that the fast algorithm only computes $k + 1$ Laplacian pyramids rather than $l \times w \times h$. For this assignment, we'd like you to use $k = 7$. Further detail on Local Laplacian filtering can be found in the references below.

References

- Sylvain Paris, Samuel W. Hasinoff, and Jan Kautz. 2015. Local Laplacian filters: edge-aware image processing with a Laplacian pyramid. Commun. ACM 58, 3 (February 2015), 81-91.
- Mathieu Aubry, Sylvain Paris, Samuel W. Hasinoff, Jan Kautz, and Frdo Durand. 2014. Fast Local Laplacian Filters: Theory and Applications. ACM Trans. Graph. 33, 5, Article 167 (September 2014), 14 pages.