Lecture 5

Gidon Rosalki

2025-11-16

1 Image Pyramids

Let us consider a picture $N \times N$. If we want to store lower resolution copies, of half the size each time, then we need a total of

$$N^2 + \frac{1}{4}N^2 + \frac{1}{16}N^2 + \cdots = 1\frac{1}{3}N^2$$

There are naturally only $\log(N)$ levels.

While we will only talk about resizing by $\frac{1}{2}$, all scales are possible. To resize by $\frac{1}{2}$, we blur, and subsample every 2nd pixel in every second row. To perform an arbitrary resizing, it is a pain to do this kind of blurring and resizing, so we instead perform a Fourier transform. For instance, to convert from $N \times N$ to $K \times K$, we take the Fourier transform, and cut the spectrum down to a $K \times K$ image, which is the K highest frequencies, and then perform the inverse Fourier, and recreate the lower resolution image.

1.1 Uses

A first use of these image pyramids is efficient visual searches. Searching is dependent on the area, and the pattern size. Given a 256^2 pixel image, and a search are of 32^2, we need 2^{26} total operations. We can instead use pyramids to start the search in a small image, and given an estimate from a lower resolution level, search the small surrounding area in higher resolution levels, and work our way up until we find the image in the largest level.

They are also useful in browsing image databases, to show multiple images or videos, or for motion computation, stitching, and more. Consider a security system, with 3000 cameras. Instead of viewing 3000 screens, we can fit many streams on a single screen.

1.2 Resizing

To reduce an image, we first blur (it can sometimes be decomposed into horizontal, and vertical). A method of this is to convolve with a 3×3 filter, or a 5×5 filter, or larger. We then subsample, by selecting only every 2nd pixel, in every second row.
To expand, we add zero padding in every second pixel, in every second row, and then blur. The expanding blur needs different normalisations, due to zero padding.

For blur kernel, we commonly use binomial coefficients of odd lengths (in order to have a centre pixel). The sum of the coefficients is normalised to 1. We do this since it is fast to compute, by using shifts, and integer addition. It is also asymptotically similar to the Gaussian. It can also be decomposed into 2 convolutions, for example

$$P \cdot \frac{1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix} = P \cdot \frac{1}{16}\begin{bmatrix} 1 \\ 4 \\ 6 \\ 4 \\ 1 \end{bmatrix} \cdot \frac{1}{16} \cdot [1\ 4\ 6\ 4\ 1]$$

This saves many computations with comparison to say a 5×5 kernel. The naïve computation blurs 5×5, with 25 multiplications, but if the kernel can be decomposed into horizontal and vertical components, then we blur columns (5 multiplications), and then blur the rows (5 multiplications), resulting in a total of 10 multiplications, instead of 25.

To handle the edges, we will never make it cyclic. This was discussed in the last lecture, but we can reflect the last pixel, use zero padding, or perhaps duplicate the last pixel. We also sometimes just remove the problematic rows / columns, which is most useful for very large images, where it matters not if we lose a couple of rows / columns each time.

A **Laplacian Pyramid** is a series where we call G_n the Gaussian, with n representing the top level, and L_n the Laplacian, where $L_i = G_i - Expand(G_{i+1})$. As a result $L_n + L_{n-1} = G_{n-1}$. By storing the Laplacian pyramid, which is much less data, we can thus compress an image, where we compress the Laplacian pyramid using Huffman or something, and can reconstruct the data subsequently.

1.3 Merging images

Splines! He didn't define clearly: For the images A, B, and every row y

$$C(x,y) = h(x)\,A(x,y) + (1 - h(x))\,B(x,y)$$

A multiresolution pyramid spline is as follows: Given two images A and B to be splined in middle. Construct Laplacian Pyramid L_a and L_b, and create a third Laplacian Pyramid L_c where for each level k:

$$L_c = \begin{cases} L_a(i,j), & \text{if } i < \frac{width}{2} \\ \frac{L_a(i,j)+L_b(i,j)}{2}, & \text{if } i = \frac{width}{2} \\ L_b(i,j), & \text{if } i > \frac{width}{2} \end{cases}$$

Finally, sum all levels in L_c to get the merged image.

We can thus combine images seamlessly using these techniques as you can see below:

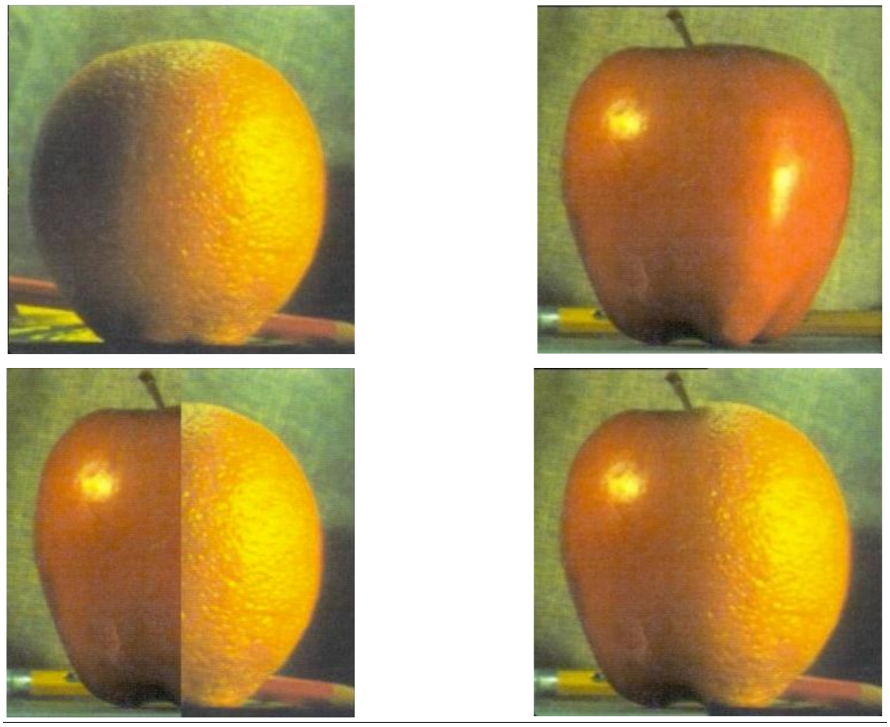

Figure 1: Merged image

To blend arbitrary shapes: Given two images A, and B, and a binary mask M, construct Laplacian Pyramid L_a and L_b, and create a third Laplacian Pyramid $L_c = M - G_m$, where for each level k:

$$L_c(i,j) = G_m(i,j)\,L_a(i,j) + (1 - G_m(i,j))\,L_b(i,j)$$

Finally, sum all levels in L_c to get the blended image.

2 Image denoising

This has been particularly important of late, since all image generation AI does is it takes a noisy image, and denoises it until it gets the desired image.

We have a few quality measures for restoration. Given the original image $I(x,y)$, restored to $\hat{I}(x,y)$. In real life, $I(x,y)$ is unknown, but known for testing. Our assumption is Additive White Gaussian Noise (AWGN). Mean squared error is defined as follows:

$$MSE = \frac{1}{N^2}\sum_{x,y}\left|\left|I(x,y) - \hat{I}(x,y)\right|\right|^2$$

We also have Peak Signal to Noise Ratio:

$$PSNR = 20\log_{10}\left(\frac{255}{\sqrt{MSE}}\right)$$

What if the noise changes regularly, then we have patch method: Non local means. Assume a static scene, giving a constant signal $x(y)$. Multiple images $y(t)$ are captured at different times

$$y(t) = x(t) + n(t)$$

The noise $n(t)$ varies over time with a mean of 0. As a result, we can add these all together, and it will converge on the original image. When the variance of each variable

$$\{X_i\}_{i=1}^{n} = \sigma^2$$

Then the variance of their mean is $\frac{\sigma^2}{n}$. An example of this is when looking at stars. The stars are static in the sky, but the atmosphere adds noise, that changes between "time steps".

We can all find similar patches in an image (like many parts of the ocean), average them, and use this as weights to remove noise. To do this:

1. Given one pixel, compute the similarity of a patch around it to patches around **all other** pixels.
2. Compute a weighted average of pixels, based on the patch similarity.
3. Replace the pixel value by this average

$$\hat{x}(m,n) = \frac{1}{c(m,n)} \sum_{i,j} y(i,j)\, e^{-(SSD(N(m,n)-N(i,j)))}$$

Where y is the input image, $\hat{x}$ the output, and $N(i,j)$ is a neighbourhood around pixel i,j

This is equivalent to

$$\hat{x}(m,n) = \frac{1}{c(m,n)} \sum_{i,j} y(i,j)\, w(m,n,i,j)$$

$$w(m,n,i,j) = e^{-\frac{(SSD(N(m,n)-N(i,j)))}{2\sigma^2}}$$

Where we choose a σ^2 that gives a good result. The pixel value at (m,n) is the average of all other pixels (i,j) in the image, weighted by $w(m,n,i,j)$. The weights are computed from Sum of Square Differences (SSD) between neighbourhoods of (m,n), and of (i,j), $N(m,n)$, etc. SSD can have equal weights for all pixels in $N(m,n)$, or Gaussian weights, where the centre pixel has a higher weight, or neighbourhoods can be normalised by mean and variance.

This is called Non Local Means. NLM is one of a family of patch based noise cleaning methods. Variations include other similarity measures between patches, replacing the SSD, define search areas for patches, E.g. search also in other frames (Google's Night-Sight), and methods to replace averaging, E.g. Multi-frame super- resolution (Google's Night-Sight).

2.1 Night sight

Google's Night Sight was developed by Yael Pritch, who studied this course some yeras ago. The traditional approach is to use very long exposure. This is, quite obviously, bad.

- Take multiple pictures (6 - 15)
- Exposure time is determined from motion (for example, from the gyro)
- Before averaging, perform alignment between pictures to compensate for motion
- Combine overlapping patches using multi frame super resolution approach
- Colour correction

This leaves us with the problem of how to focus? Well, we failed to answer this in the lecture. I presume that an amount of work is done with trying to take each image as focused as you can, and the overlapping patches can have focusing work done to them I guess.