For this lecture, please see the slides which show many figures.

Scale Space

So far we have looked at various fundamental ideas in image processing: filtering and edge detection, corner detection (localized features), image registration (optical flow, tracking). We have seen a many examples of how to locally analyze the raw pixel values using intensity derivatives, and how to detect points that are somehow important.

The pixel positions are just samples and the sampling rate is somewhat arbitrary. The sampling rate is a property of the camera, not a property of the underlying image intensities. Objects can appear at different sizes, and so our analysis of images somehow needs to be done not just at the arbitrary scale of the pixels, but at other scales too.

To get some intuition about scale, think of what happens when an object recedes in depth: a far away object has less detail (fewer pixels) than a close object. You may have difficulty recognizing a face or reading text if there are only a few dozen pixels to represent the object.

One way to model this scale effect mathematically is to blur an image and then subsample it. Note that this is different than just subsampling alone. If we just subsample, then our samples will skip over many pixels and the image will not correspond to what happens when the object is moved far away. By blurring the image first and then subsampling, we are guaranteed that all pixels in the original image will be represented. (There are technical arguments that can be given about blurring and sampling, which you would learn about in a signal processing course. But we'll not go into them.)

Gaussian scale space

Take a 1D image $I(x)$ and define family of images

$$I(x, \sigma) = I(x) * G(x, \sigma).$$

This family is called the *Gaussian scale space*. It is a family of blurred 1D images, indexed by the amount of blur σ which is the standard deviation of the Gaussian. We can define a 2D Gaussian scale space in a similar way.

$$I(x, y, \sigma) = I(x, y) * G(x, y; \sigma).$$

Just as we have to discretize the set of pixels (x, y) in an image, we have to discretize the set of scale σ, whenever we build a scale space. Once could choose a sequence of scales using an arithmetic progression such as

$$\sigma = 0, 1, 2, etc.$$

A more common approach is to use a geometric progression such as

$$\sigma = s_0, 2s_0, 4s_0, 8s_0, \ldots.$$

To keep the notation simple, we'll let $s_0 = 1$ so $\sigma = 1, 2, 4, 8, etc.$ Each doubling of scale is called an *octave*. One often samples scale more finely than that by having m scales per octave. We'll see an example later.

Given an image $I(x, y)$, consider sampling the σ dimension in scale space as follows:

$$I(x, y, 0) \equiv I(x, y)$$

and where $k \geq 0$

$$\begin{aligned} I(x, y, 1) &\equiv I(x, y) * G(x, y; \sigma = 1) \\ I(x, y, 2) &\equiv I(x, y) * G(x, y; \sigma = 2) \\ I(x, y, 4) &\equiv I(x, y) * G(x, y; \sigma = 4) \\ &\dots \\ I(x, y, 2^k) &\equiv I(x, y) * G(x, y; \sigma = 2^k) \end{aligned}$$

Each of these blurred images is the same size as the original image.

Gaussian pyramid

When we blur an image with larger σ, we get an image whose intensity values vary more gradually across (x, y). One can show that we need fewer samples to get a good approximation of a more blurred image.[1] One often takes advantage of this fact. One way to do so is to use a *Gaussian pyramid*.

We can sample these blurred functions by defining a sequence of images that each have successively half as many pixels in the x and y dimensions as their predecessor:

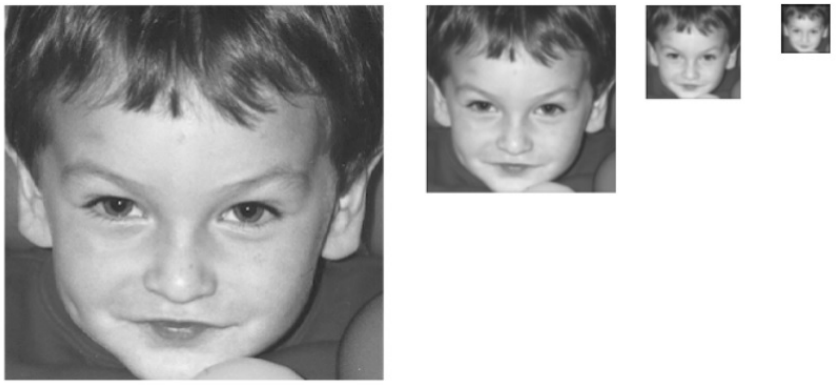
$$\begin{aligned} I(x, y) &= I(x, y, 0) \\ I_1(x, y) &= I(2x, 2y, 1) \\ I_2(x, y) &= I(4x, 4y, 2) \\ I_3(x, y) &= I(8x, 8y, 4) \\ &= \dots \\ I_k(x, y) &= I(2^k x, 2^k y, 2^{k-1}) \end{aligned}$$

An example is given in the slides and accompanying Matlab code.

The more common way to compute a Gaussian pyramid is to alternate the blur and subsample operations. One blurs the original image with a σ of say 1, then subsamples it with a stride (step) of 2 to get a smaller image $I_1(x, y)$, then blurs the *subsampled* image I_1 using a σ of 1 to define $I_2(x, y)$ and subsamples it with a stride of 2, and so on. The two methods do not give exactly the same results, but they're close.

An example sequence is shown below for $I, I_1, \dots, I_3$ which is taken from an excellent book by Brian Wandell. Each successive image has half the height and width of the previous one.

[1] To make this claim more precise, one needs a mathematical theory of sampling and blurring. To learn more, you would need to take a signal processing course.



Application: "Coarse-to-fine" image registration

The Gaussian pyramid is often used in tasks such as binocular stereo or optical flow. We discussed these problems last lecture, namely given two image frames find local shifts from one frame to the other.

In practice, these shifts (h_x, h_y) might vary across the image and the values might be small or large. The algorithm last lecture assumes that the displacements are small, however, since it is based on a first order Taylor series. So, when the displacement vector (h_x, h_y) is large, the method does not work well. It can easily get stuck in a local minima of the squared error function.

The Gaussian pyramid provides a way to solve this local minimum problem. One can compute the Gaussian pyramid for both the images I and J. One can run the LK algorithm at a higher level of the pyramid(s) first, where the number of pixels is relatively small and the images have been heavily blurred (and subsampled). In this case, one pixel in the image at scale $\sigma = 2^k$ corresponds to 2^k pixels in the original image.

One begins with an estimate of h at some chosen high level of the pyramid. One can then successively refine the estimates of the h as one moves from higher to lower levels in the pyramid. This is called a "coarse-to-fine" approach. One takes the estimates of h at level k (larger scale) of the pyramid, and uses them as *initial estimates* (h_x^0, h_y^0) of the LK algorithm when it is run at level $k-1$ of the Gaussian pyramid(s). One runs the LK algorithm at level $k-1$, refining the estimate, and so on down to the base level 0 of the pyramid.

There are two potential benefits here. First, by running the LK algorithm on smaller images first, one can obtain approximate sparse estimates very quickly. Second, if some of the h vectors in the original image (base level) are large, then these vectors will be a factor of 2^k smaller at level k of the pyramid and so the LK algorithm is more likely to be able to estimate them, without getting getting stuck in local minimum.

Note that there are potentially two iterations going on here. There is the coarse-to-fine iteration, from high to low levels of the pyrmad. There is also the (potential) iteration *within* each level of the pyramid, which could be used to fine tune the estimate of (h_x, h_y) at level k.

Features in scale space (edges and blobs)

Let's return to the idea of our 1D and 2D scale spaces $I(x,\sigma)$ and $I(x,y,\sigma)$ and put aside the Gaussian pyramid for now. Let's think about how features such as edges appear in scale space.

Edges in scale space and normalized Gaussian derivative filters

Take an image $I(x)$ which is a noise-free unit step edge

$$u(x) = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

One can show that (see Appendix) that

$$\frac{dG(x,\sigma)}{dx} * u(x) = G(x,\sigma).$$

Thus if we filter the edge function $u(x)$ with a first derivative of a Gaussian, then the result depends on the scale σ of the Gaussian, namely the peak $G(0,\sigma)$ is $\frac{1}{\sqrt{2\pi}\sigma}$ which you can see just by plugging the mean $x = \mu$ into the equation of the Gaussian.

If we want the response to be independent of σ, we need to filter with $\sigma\frac{dG(x,\sigma)}{dx}$ instead. The filter $\sigma\frac{dG}{dx}(x)$ is sometimes called the *normalized Gaussian derivative*. The resulting scale space as the *normalized Gaussian derivative scale space*.

Note that the edge $u(x)$ itself does not have a "scale". It jumps instantaneously from one value (0) to another (1) and the edge location $x = 0$.

Next, recall the Laplacian of a Gaussian function which was used in Marr-Hildreth edge detection. For 1D images, the corresponding filter is simply a second derivative of a Gaussian. What happens if we filter the edge $u(x)$ with $\frac{d^2G(x,\sigma)}{dx^2}$? Since a derivative is a convolution and since convolution is commutative, we can interchange the order of derivatives and convolution to get:

$$u(x) * \frac{d^2G(x,\sigma)}{dx^2} = \frac{d}{dx}(u(x) * \frac{dG(x,\sigma)}{dx}) = \frac{dG(x,\sigma)}{dx} = \frac{1}{\sqrt{2\pi}\sigma}\frac{-x}{\sigma^2}e^{\frac{-x^2}{2\sigma^2}}.$$

where the second equality was given above, and the third equation comes from Calculus 1. We would get a value 0 at $x = 0$ which is the location of the edge. This function has a positive peak at $x = -\sigma$ and a negative peak at $x = \sigma$, which we can verify by Calculus 1, namely take the derivative and set it to 0, and solve for x.

Next, plugging in $x = \pm\sigma$, one finds the height of these two peaks are $\pm\frac{1}{\sqrt{2\pi\sigma^2}}e^{-\frac{1}{2}}$. If we wanted the heights of these peaks to be independent of σ then we would need to use the filter $\sigma^2\frac{d^2G(x,\sigma)}{dx^2}$ instead, since this would cancel the $\frac{1}{\sigma^2}$ term in the peak's value. The important point for us here isn't the value itself, but rather the fact that this value doesn't depend on σ. Because of this independence, we would call the filter the *normalized Gaussian second derivative*.

We can make similar arguments for the shifted edge $u(x - x_0)$, and for the more general shifted edge $au(x - x_0) + b$. Shifting the edge by x_0 just shifts the locations of the feature points (the peaks or the zero crossings) by x_0. Multiplying the edge by a to get $au(x - x_0)$ just multiplies the heights of the peaks by a. Adding a constant b does nothing, since the derivative of a constant is 0.

To briefly summarize, if we have a step edge and we convolve with a Gaussian derivative or Gaussian second derivative filter then the result will depend on the σ of the Gaussian. If, however, we normalize these filters by multiplying by σ (in the case of a first derivative) or σ^2 (in the case of a second derivative), then the output will not depend on σ. This will turn out to be useful for when we look for features that are maxima or minima in scale space (x, y, σ), rather than just maxima or minima in (x, y).

1D Box

Let's now apply these results to the problem of detecting a 1D "box", sometimes called a "blob". The "box" function has value 1 for $x \in [-\sigma_0, \sigma_0]$ and value 0 otherwise. We can define such an image as the difference of shifted step edges, namely

$$I_{box}(x; \sigma_0) = u(x + \sigma_0) - u(x - \sigma_0).$$

Note that the box is centered at $x = 0$.

If we convolve $I_{box}(x; \sigma_0)$ with $\sigma^2 \frac{d^2 G(x,\sigma)}{dx^2}$, then the function defined in scale space (x, σ) will have a minimum at $(x, \sigma) = (0, \sigma_0)$. The reason is that there will be negative peak in $u(x+\sigma_0) * \sigma^2 \frac{d^2 G(x,\sigma)}{dx^2}$ at $x = 0$ and also there will be a negative peak in $-u(x-\sigma_0) * \sigma^2 \frac{d^2 G(x,\sigma)}{dx^2}$ at $x = 0$. These two negative peaks at $x = 0$ give a minimum response at $x = 0$ which is the center of the box. The value of this minimum response is twice the value of the individual negative peaks, namely $-\frac{2}{\sqrt{2\pi}} e^{-\frac{1}{2}}$. Note that if we did not normalize by multiplying by σ^2, then this unique global minimum property would not hold.

The box function that we have been considering has value either 0 or 1. If we were to shift this box function $I_{box}(x - x_0)$ then the same key property would hold, namely that if we were to convolve it with $\sigma^2 \frac{d^2 G(x)}{dx^2}$ then we would have a unique global minimum in scale space at $(x, \sigma) = (x_0, \sigma_0)$ whose x position is the center of the box.

2D Box

The results extend naturally to 2D images. We can define a 2D box function by taking the product of two 1D box functions.

$$I_{box}(x, y) = I_{box}(x) I_{box}(y)$$

This function has a square shape which is centered at the origin. One can show that convolving the 2D box with a *normalized Laplacian of a Gaussian*

$$I_{box}(x, y) * \sigma^2 \nabla^2 G(x, y, \sigma)$$

gives a negative peak value at $(x, y, \sigma) = (0, 0, \sigma_0)$, namely at the center of the box (spatially) and at the "half width" σ of the box in the scale dimension. The same is true if the box function is shifted to some other location (x_0, y_0). Thus, 2D square box functions centered at an arbitrary position (x_0, y_0) and of half width σ_0 yield local minima in the *normalized Laplacian of a Gaussian scale space*. Note that if we were to define a dark box on a lighter background, then we would get local maxima instead of local minima in the normalized Laplacian of a Gaussian scale space.

Appendix

In the continuous domain, the unit step function $u(x)$ has the following property: for any differentiable function $f(x)$ that goes to 0 as $x \to \pm\infty$,

$$u(x) * \frac{df(x)}{dx} = f(x).$$

To see why, consider what happens when we convolve $u(x)$ with any function $g(x)$.

$$u(x) * g(x) = \int_{-\infty}^{\infty} u(x')g(x - x')dx' = \int_{0}^{\infty} g(x - x')dx' = \int_{-\infty}^{x} g(x')dx'$$

Substituting $g(x) = \frac{df(x)}{dx}$, we get

$$u(x) * \frac{df(x)}{dx} = \int_{-\infty}^{x} \frac{df(x')}{dx'}dx' = f(x) - f(-\infty) = f(x)$$

