CIS 580 Spring 2012 - Lecture 9

February 13, 2012

Notes and figures by Matthieu Lecce.

Scale Space and Scale Invariant features

LAST TIME, we introduced the diffusion equation $\frac{\partial L}{\partial t} \underset{2D}{=} \frac{1}{2}\nabla^2 L$ with boundary conditions $L(x,y,t=0) = \underbrace{I(x,y)}_{\text{image}}$, and said that the solution is the image $I(x,y)$ convolved with a Gaussian ($t = \sigma^2$):

$$L(x,y,t) = g(x,y,t=\sigma^2) * I(x,y).$$

Indeed the necessary condition is $\frac{\partial g}{\partial t} = \frac{1}{2}\nabla^2 g$, in 1D $\frac{\partial g}{\partial t} = \frac{1}{2}\frac{\partial^2 g}{\partial x^2}$

WE INTRODUCED the *scalability* property of the Gaussian. Let I' be a scaled version of an image I ($I(x) = I'(sx)$, where s is the size)) and $L'(x',t') = g(x',t') * I'(x')$, we have:

Do not confuse s with t or σ, s in the intrinsic scale or "size" of a feature. We define $x' = sx$

$$\boxed{L(x,t) = L(x',t') \text{ for } t' = s^2 t}.$$

(Last time we showed it two ways: by computing the convolution and by using Fourier)

WE ALSO SHOWED that this is **not true for the derivatives** of the diffused image (recall they are equal to the image convolved with the Gaussian derivative):

$$\begin{aligned}\frac{\partial^m L(x,t)}{\partial x^m} &= \frac{\partial^m L'(x',t')}{\partial x'^m}\frac{\partial^m x'}{\partial x^m} \\ &= \frac{\partial^m L'(x',t')}{\partial x'^m} s^m\end{aligned}$$

since $x' = sx$

Last time we introduced and explained, with a 1D and a 2D example, the need for a scale normalization to compensate for this shrinkage of the output amplitude at bigger scales.

WE NOW FORMALIZE THIS CONCEPT of **normalized derivative** by substituting $\xi = \frac{x}{t^{\gamma/2}}$ ("γ-normalized" derivative). We have:

$$\begin{aligned}\frac{\partial^m L(x,t)}{\partial \xi^m} &= \frac{\partial^m L(x,t)}{\partial x^m}\frac{\partial^m x}{\partial \xi^m} \\ &= s^{m(1-\gamma)}\frac{\partial^m L(x',t')}{\partial x'^m} \\ &= \frac{\partial^m L(x',t')}{\partial x'^m} \quad \text{for } \gamma = 1\end{aligned}$$

We are mostly interested in the second derivative (Laplacian), because of the diffusion equation. In the non-normalized case, we have:

$$\frac{\partial g}{\partial t} = \frac{1}{2}\frac{\partial^2 g}{\partial x^2}$$

Normalized case (with $\gamma = 1$):

$$\frac{\partial^2 g}{\partial \xi^2} = \frac{\partial^2 g}{\partial x^2}\frac{\partial^2 x}{\partial \xi^2} = \sigma^2 \frac{\partial^2 g}{\partial x^2}$$

$$\xi = \frac{x}{t^{\gamma/2}} = \frac{x}{\sigma} \qquad t = \sigma^2, \quad \gamma = 1$$

$$\frac{\partial g}{\partial \sigma} = \sigma \frac{1}{\sigma^2}\frac{\partial^2 g}{\partial \xi^2} = \frac{1}{\sigma}\frac{\partial^2 g}{\partial \xi^2}$$

LET'S TAKE A MOMENT TO REALIZE HOW USEFUL THIS RESULT IS:

- Instead of filtering the image with a normalized 2^{nd}-derivative filter w.r.t. x, we can use $\frac{\partial g}{\partial \sigma}$

  The 2^{nd}-derivative filter is also called Laplacian of Gaussian (LoG)

- $\frac{\partial g}{\partial \sigma}$ can be simply approximated as a finite difference
- Therefore we have a simple and efficient way to compute the scale normalized second derivative filtering of an image I: we take a simple difference of two blurred versions of I.

  We saw that scale normalization is critical if we want responses to be comparable across scales.

APPROXIMATING $\frac{\partial g}{\partial \sigma}$

$$\frac{\partial g}{\partial \sigma} \approx \frac{g(x, \sigma + \Delta\sigma) - g(x, \sigma)}{\Delta\sigma}$$

We note $\sigma + \Delta\sigma = \kappa\sigma$ ($\kappa = 1.01$ would be a good approximation), and we have:

$$\frac{\partial g}{\partial \sigma} \approx \frac{g(x, \kappa\sigma) - g(x, \sigma)}{\sigma(\kappa - 1)} = \textbf{DoG}$$

A Difference of Gaussians (DoG) is the difference of two Gaussians of same center and different σ.

$$\text{DoG} = (\kappa - 1)\sigma\frac{\partial g}{\partial \sigma} = (\kappa - 1) \underbrace{\frac{\partial^2 g}{\partial \xi^2}}_{\text{normalized LoG}}$$

"Every DoG is a LoG" in the normalized case (which is the only case we care about of course).

We apply this recursively to build a pyramid, starting from original image:

- We convolve it with a Gaussian $g(x, \sigma)$
- We subsample it by 2
- We obtain $I_2(x) \circ\!\!-\!\!\bullet\, I(\omega) * \sum_{n=-\infty}^{\infty} \delta(\omega - n\pi)$

Because of the effect of subsampling in the frequency domain, we should eliminate all frequency components $|\omega| > \frac{\pi}{2}$. A rect-filter is not an option because it has an infinite impulse response (sinc).

Given that we want to maintain the scale space properties, we have to find out the effect of a *discrete Gaussian filter* (e.g. the binomial filter introduced in lecture 7). Remember that the DTFT of a discrete filter is:

$$H(\omega) = \sum_{k=-n}^{n} h[k] e^{-j\omega k}$$

the binomial filters also have the nice property that they sum to powers of 2 ($\sum_{k=1}^{n} \begin{pmatrix} n \\ k \end{pmatrix} = 2^n$), so they are easily normalized

(Slides)

Scale for feature matching

Scale is important to match features: the window size to compute local features (histogram) needs to be right in order to match features correctly.

Many recent algorithms apply scale detection (SIFT) at all positions in the image, but ideally we are interested in finding *interest points*, by finding

$$\max_{x,y} \max_{\sigma} \left(\sigma^2 \frac{\partial^2 g}{\partial x^2} * I(x) \right)$$

Note that some approaches don't take the maximum over positions and just compute the features at optimal scale for every point on a grid.

Blob detection We will implement a simple multi-scale blob detection system in the next homework. We will use the LoG filter because of its appearance similar to a blob. Inside the same octave, we will take simple differences of Gaussians to approximate the normalized Laplacian.

We will not implement octaves, which are obtained by subsampling.

After computing the DOG, we compute maximum at every pixel (wrt. σ), and **then** we try to find maximum with respect with position (using derivative approximation: Taylor expansion, see slides).

Play with Andrea Vedaldi's VLFeat toolbox for SIFT detection and descriptor computation.