Computer Vision

MoSIG M2

James L. Crowley

Fall Semester 22 October 2020

Lesson 3

Image Pyramids and Scale Invariant Image Description

Lesson Outline:

1. Scale Space 2
   - 1.1. Scale and Resolution - Some Vocabulary 2
   - 1.2. Scale Space 3
   - 1.3. Computing a Scale Space 3
   - 1.4. Convolution and Cross Correlation 4
   - 1.5. Multi-Scale Gaussians 4
2. Gaussian Digital Filters 6
   - 2.1. The Gaussian Function 6
   - 2.2. The 2D Gaussian Function 7
   - 2.3. Gaussian Derivatives 7
   - 2.4. Gaussian Digital Filters 11
   - 2.5. 2D Gaussian Digital Filters 15
3. Image Pyramids 16
   - 3.1. Direct Calculation of an Image Pyramid 17
   - 3.2. Scale Invariant Pyramid Algorithm 19
   - 3.3. Sampling 20
4. Invariant and Equivariant Properties in Gaussian Scale Space 22

1. Scale Space

1.1. Scale and Resolution - Some Vocabulary

A phenomena is anything that you can perceive. An **entity** is any observable phenomena. In an image, entities occupy 2D regions of pixels.

The **scale** of an entity refers to its size. In the real world, scale is expressed in meters (or inches and feet if you prefer to use the thumb and foot of some obscure old english king as a unit of measure). For computer vision, scale refers to image pixels, and how they relate to objects in the real world. The scale of an entity in an image is the length of the largest line segment that can be contained within the region of pixels occupied by the entity.

Resolution refers to the ability to "resolve" detail, typically expressed as the smallest distance for which two entities can be discriminated. For a digital signal, resolution is expressed as samples. For sound, resolution expresses the temporal sample rate, and determines the highest frequencies that can discriminated (or resolved). For images, resolution expresses the spatial sample rate used to represent an entity, and determines the smallest entities that can be resolved.

For example, in a high-resolution aerial photograph of the city, you can "resolve" small details such as sidewalks and people. In a low-resolution image, you can only resolve large structures such as roads, buildings and rivers.

It is always possible to use more pixels than needed for an image. However, adding more pixels does not necessarily add additional information and can even make large structures more difficult to perceive.

Obviously, resolution and scale are related. But they are sometimes confused. One way to think of this is to see scale as the physical size of a region in image pixels, and resolution is the minimum number of sample needed to represent the region for analysis.

Recognition often requires context. Recognizing an entity can often depend on the presence or absence of neighboring entities. **Locality** refers to the scale of context. Locality expresses the scale of the scale of the region around an entity needed for recognition.

1.2. Scale Space

In most natural images, entities can be found at multiple scales. In general information at larger scales provides context for information at smaller scales.

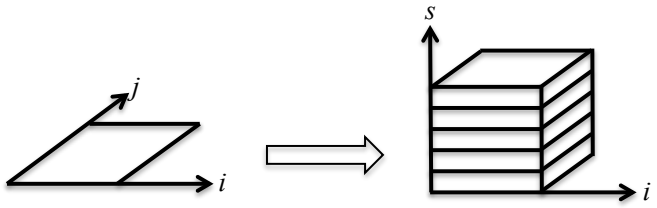
For image understanding, we need to interpret information in localities (neighborhoods) at different scales, in order to recognize each entity in its appropriate context. Such processing can be said to be "multi-local". The challenge is that large entities can become much harder to detect and recognize if represented at high resolution. Beyond a certain resolution, both error rates and computational cost increase.

To avoid this problem we need to analyse the image at multiple scales, with each scale represented at the appropriate resolution (number of sample rates). The entities in the image are said to exist in a space of scales (a scale-space), with each scale represented at an appropriate resolution. To do this, we express the scale space of the image with a **multi-resolution image pyramid**.

How can we determine appropriate resolution for a scale? How can we transform an image to a multiple resolution representation without loosing information? Digital signal processing provides the analytic tools (and techniques) that we need.

1.3. Computing a Scale Space

Let *P(i, j)* be a 2-D image where *(i, j)* are the image positions of pixels. Each integer value for *(i, j)* provides the unique identity for a picture element (a pixel), and the order relation of *i* and *j* locates each pixel within a 2D space. Real values for x and y can be used to identify locations between pixels if needed.



To compute a "scale space" we add a third dimension, s to the image space to define a continuous 3D space *P(i, j, s)*. To do this properly, we need to transform our discrete, integer sampled image to a continuous signal. We do this with a measure function. For a many reasons, some of which will be developed below, the most convenient measure function for this is the Gaussian (or normal) function.

This is commonly done for by convolving the image with multiple copies of a 2D Gaussian filter at multiple scales:

$$P(x,y,s) = \left(P * g\right)(x,y,\sigma_s) \quad \text{for an exponential range of scales:} \quad \sigma_s = \sigma_o^s$$

Note that in the scientific literature for computer vision, image processing and digital signal processing this would be sometimes written as:

$$P(x,y,s) = P(i,j) * g(x,y,\sigma_s)$$

1.4. Convolution and Cross Correlation

We can transform a sampled signal *f(n)* into a continuous signal *f(x)* by convolving the signal with a continuous function. A popular function, described blow, is the Gaussian.

$$f(x) = (f * g)(x) = \sum_{n=-\infty}^{\infty} f(n) g(x-n,\sigma)$$

The term x-n has the effect of flipping the Gaussian around its center point. This "flipping" simplifies certain analysis in the Fourier domain and is thus part of convolution. However, the Gaussian is Symmetric, so in fact this flipping has no effect! We could have written:

$$f(x) = (f \otimes g)(x) = \sum_{n=-\infty}^{\infty} f(n) g(x+n,\sigma)$$

This operation is called cross correlation, and is said to be easier for beginners to visualize. It is often mistakenly called convolution in some machine learning papers. For symmetric signals such as the Gaussian, the distinction is irrelevant.

1.5. Multi-Scale Gaussians

Notice that we need to define a scale variable, σ, for our Gaussian. Convolution with the Gaussian blurs (smears-out) the values of the pixels with a spread proportional to the scale, making it more difficult to resolve small details. In signal processing terms, the Gaussian is said to be the "Point Spread Function" for the continuous signal *f(x)*.

Intuitively, we would think that we want to use a scale variable as small as possible in order to be able to detect small details. However, this smearing also surpresses image noise and makes it easier to detect large-scale entities.

Recognizing entities at multiple scales is made easier by convolving with multiple values of σ. This gives us a multi-scale version of the signal:

$$f(x,\sigma) = (f * g)(x) = \sum_{n=-\infty}^{\infty} f(n)g(x-n,\sigma)$$

The standard deviation, σ, is often referred to as the scale of the Gaussian. Scale determines the scale of entities that we can recognize in the resulting signal. Thus we can use σ, to define the scale axis for a multi-scale description of the signal. The scale of the Gaussian will also determine the resolution required to represent the signal.

2. Gaussian Digital Filters

2.1. The Gaussian Function

The normalized Gaussian Function is $$g(x,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{x^2}{2\sigma^2}}$$

Where x is a spatial dimension and σ^2 is the second moment of the function. The letter "*e*" refers to the Euleur constant e=2.7182818284.... We can use the parameter σ as a measure of scale of the Gaussian.

The term $\frac{1}{\sigma\sqrt{2\pi}}$ assures that Gaussian function integrates to 1 for all values of σ.

$$\sigma\sqrt{2\pi} = \int_{-\infty}^{\infty} e^{-\frac{x^2}{2\sigma^2}} dx$$

Gaussian Scale Property

A very important property for us is that the convolution of a two Gaussians yields a scaled Gaussian.

$$g(x,\sigma) * g(x,\sigma) = g(x,\sqrt{2}\sigma)$$

In general $$g(x,\sigma_1) * g(x,\sigma_2) = g(x,\sqrt{\sigma_1^2 + \sigma_2^2})$$

The convolution of a Gaussian with variance σ_1^2 with a Gaussian with variance σ_2^2 creates a larger Gaussian with a variance $\sigma_3^2 = \sigma_1^2 + \sigma_2^2$.

Thus we can decompose a convolution into two smaller convolutions:

$$f(x) = (f * g_3)(x) = \left((f * g_1) * g_2\right)(x)$$

This will make it possible to create a fast recursive algorithm for computing image pyramids.

2.2. The 2D Gaussian Function

Our image, *P(i,j),* is a 2-D sampled signal, and so we need to use a 2D Gaussian.

The 2D Gaussian function is: $g(x,y,\sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}}$

Gaussian functions have many interesting properties. For example the 2D Gaussian is the only function that is both separable and circularly symmetric.

Separability: $g(x,y,\sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}} = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}} * \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{y^2}{2\sigma^2}}$

This means that a convolution with a 2-D NxN Gaussian ($O(N^2)$ operations) can be computed as a sequence of 2 convolutions with 1-D Gaussians (O(N) operations).

$$\left(p * g\right)(x,y,\sigma) = \left(\left(p * g\right)(x,\sigma)\right) * g(y,\sigma)$$

or as it is more commonly written in computer vision

$$p(i,j) * g(x,y,\sigma) = \left(p(i,j) * g(x,\sigma)\right) * g(y,\sigma)$$

2.3. Gaussian Derivatives

Images are discrete (sampled) signals. Convolution with a continuous Gaussian makes it possible to perform operations to images that would normally only be possible with a continuous signal. For example, to compute derivatives:

$$f_x(x) = \left(\frac{\partial f(x)}{\partial x} * g\right)(x) = \frac{\partial (f * g)(x)}{\partial x} = \left(f * \frac{\partial g(x)}{\partial x}\right)(x) = \sum_{n=-\infty}^{\infty} f(n) g_x(x-n,\sigma)$$

Gaussian Derivatives are classically used to describe invariant image structures.

Gaussian Derivatives:

$$g_x(x,y,\sigma) = \frac{\partial G(x,y,\sigma)}{\partial x} = -\frac{x}{\sigma^2} G(x,y,\sigma)$$

$$g_y(x,y,\sigma) = \frac{\partial G(x,y,\sigma)}{\partial y} = -\frac{y}{\sigma^2} G(x,y,\sigma)$$

$$g_{xx}(x,y,\sigma) = \frac{\partial^2 G(x,y,\sigma)}{\partial x^2} = \frac{x^2-\sigma^2}{\sigma^4} G(x,y,\sigma)$$

$$g_{xy}(x,y,\sigma) = \frac{\partial^2 G(x,y,\sigma)}{\partial x \partial y} = \frac{xy}{\sigma^4} G(x,y,\sigma)$$

$$g_{xxx}(x,y,\sigma) = \frac{\partial^3 G(x,y,\sigma)}{\partial x^3} = \frac{x^3-x\sigma^2}{\sigma^6} G(x,y,\sigma)$$

We can compute Gaussian derivatives by first convolving with a Gaussian, then convolving with simple local sum and difference operators. This gives a very fast algorithm that we will see this next week.

The Laplacian of the Gaussian

The Laplacian of the Gaussian is the sum of the second derivatives.

$$\nabla^2 G(x,y,\sigma) = G_{xx}(x,y,\sigma) + G_{yy}(x,y,\sigma)$$

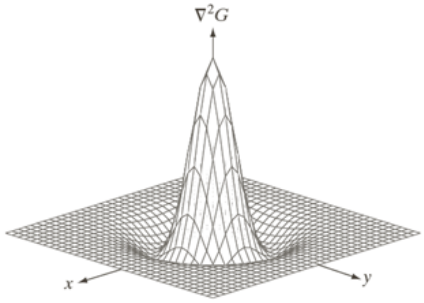
The 2D Laplacian of the Gaussian is both separable and circularly symmetric.

The Gaussian is the unique solution to the diffusion equation, and as a consequence:

$$\nabla^2 G(x,y,\sigma) = G_{xx}(x,y,\sigma) + G_{yy}(x,y,\sigma) = \frac{\partial G(x,y,\sigma)}{\partial \sigma}$$

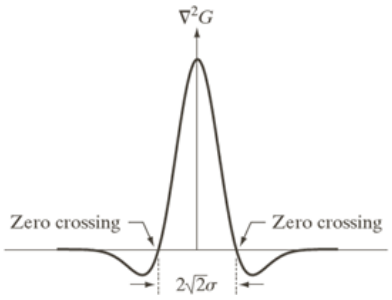
In human vision, the 2D Laplacian of the Gaussian is used to detect the natural scale of entities and thus to drive attention in the LGN. The central region acts as a kind of "spot detector" for entities of a particular scale.

For computer vision, the Laplacian of the Gaussian provides a Scale-invariant feature detector, used in the popular SIFT operator (Scale Invariant Feature Transform).



2D Plot of $\nabla^2 G(x,y,\sigma)$

Image of $\nabla^2 G(x,y,\sigma)$



1-D Cross section $\nabla^2 G(x,y,\sigma)$

The Difference of Gaussian (DoG) operator

The Gaussian is the unique solution to the Diffusion equation:

$$\nabla^2 G(x,y,\sigma) = G_{xx}(x,y,\sigma) + G_{yy}(x,y,\sigma) = \frac{\partial G(x,y,\sigma)}{\partial \sigma}$$

As a result, we can approximate the Laplacian as a Difference of Gaussians:

$$\nabla^2 G(x,y,\sigma) \approx \left(G(x,y,\sigma_1) - G(x,y,\sigma_2)\right)$$

This is sometimes called a DoG (Difference of Gaussian) operator and can be computed very easily from a Gaussian Pyramid as a difference of levels.

It is common to use: $\nabla^2 G(x,y,\sigma) \approx G(x,y,\sqrt{2}\sigma) - G(x,y,\sigma)$

But note that from the scale property: $G(x,y,\sqrt{2}\sigma) \approx G(x,y,\sigma) * G(x,y,\sigma)$

so that $\nabla^2 G(x,y,\sigma) \approx G(x,y,\sigma) * G(x,y,\sigma) - G(x,y,\sigma)$

We can compute the Laplacian of an image recursively:

$$\nabla^2 P(x,y,\sigma) = P * \left(\nabla^2 G(x,y,\sigma)\right) \approx \left(P * G(x,y,\sigma)\right) * G(x,y,\sigma) - P * G(x,y,\sigma)$$

2.4. Gaussian Digital Filters

The Gaussian function can be used to construct a finite impulse response (FIR) digital filter. This requires (1) sampling the function and (2) limiting its duration (windowing) and (3) renormalizing its gain.

The finite impulse response (FIR) digital Gaussian filter is:

$$g(n,\sigma) = \frac{1}{B} W_N(n) \cdot e^{-\frac{(n\Delta x)^2}{2\sigma^2}}$$

The function Δx is a sample distance (or sample rate). For simplicity, we can use $\Delta x=1$, but other values are possible.

The term B is a normalization factor assures that the "gain" of the filter is 1.

$$B = \sum_{n=-R}^{R} e^{-\frac{n^2}{2\sigma^2}}$$

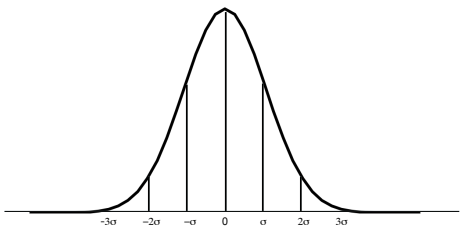
The normalisation factor, B, will be slightly less than $\sqrt{2\pi}\sigma$ because of the window function truncates the Gaussian.

The function $W_N(n)$ is called a window function, and serves to limit the spatial extent (window size) of the Gaussian. We will use a rectangular function as a window:

$$W_N(n) = \begin{cases} 1 & \text{for } -R \le n \le R \\ 0 & \text{otherwise} \end{cases}$$

Where R is a "radius". Typically R should be $R \geq 3\sigma$ for reasons that are developed below.

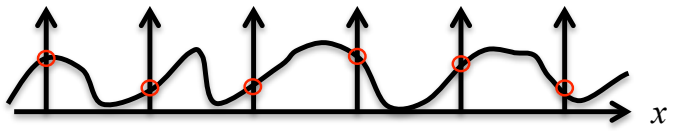
Setting $\Delta x=1$ and R = 3 gives a filter that looks like this:



To understand the as a Finite Impulse Response Gaussian Digital Filter, we need to model the effects of sampling and windowing.

Sampling

Sampling a continuous function creates an effect known as aliasing. This is because any information at scales that are smaller than the sample rate may or may not be captured in a sample. The result is that the samples contain random noise.



Aliasing is commonly modeled and analyzed with a Fourier Transform.

The Fourier transform of a signal s(x) is :

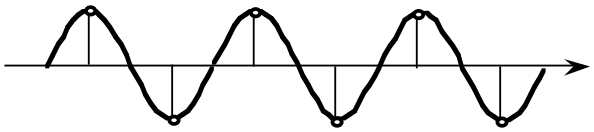
$$S(\omega) = \int_{-\infty}^{\infty} s(x) e^{-j\omega x} dx \quad \text{where } \omega = 2\pi f$$

The Fourier transform S(ω) exists for all frequencies from –∞ to ∞.

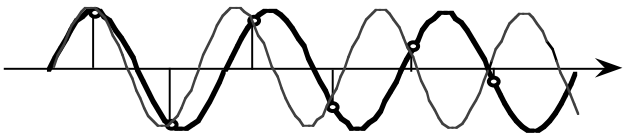
For a sampled signal, the Discrete Sampled Fourier Transform
(also known as the Discrete Time Fourier Transform (DTFT) is

$$S(\omega) = \sum_{n=-R}^{R} s(n) e^{-j\omega n} \quad \text{where } \omega = 2\pi f$$

In the discrete sampled Fourier transform, frequencies higher than 1 cycle per two samples, are indistinguishable from frequencies at lower frequencies.



A sampled cosine with 2 samples per period

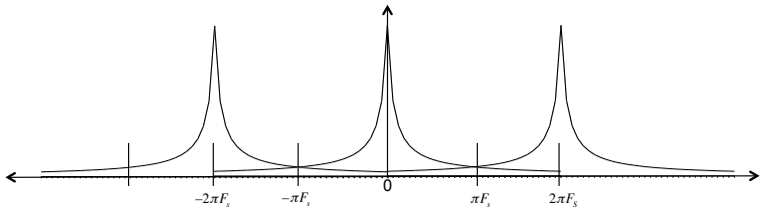


Cosine with less than 2 samples per period looks like a cosine with more than 2 samples per period

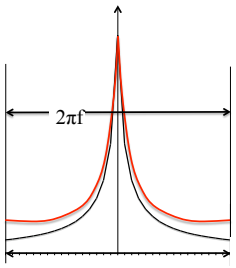
Sampling at a step size of Δx creates multiple copies of the power spectrum in the frequency domain with each copy offset by a the frequency $\omega = 2\pi F = 2\pi/\Delta x$

If we consider Δx = 1, we can say that S(ω) is periodic with a period of 2π

The Discrete Sampled Fourier Transform is periodic with a period of $\omega_s=2\pi/\Delta x$. Sampling at a step size of Δx creates multiple copies of the power spectrum in Frequency with each copy offset by a the frequency $\omega_s = 2\pi F_s = 2\pi/\Delta x$

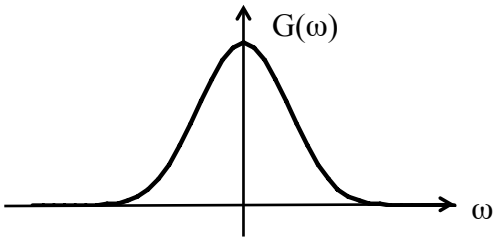


For a sampled signal, the spectrum is unique over a range of 2π frequencies. Beyond that are an infinite number of copies of the spectrum. These copies add to create a composite spectrum. Any energy beyond $\omega_s/2= \pi F_s$ is added as noise to the next copy. The results is random noise added to the signal, shown in red.

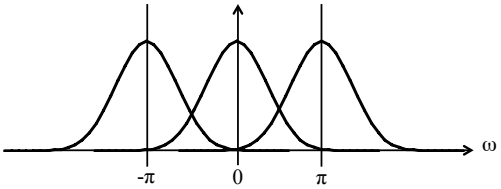


Limiting aliasing requires removing(filtering) frequencies beyond $\omega_s/2= \pi F$. This may be done with the Gaussian Low pass filter.

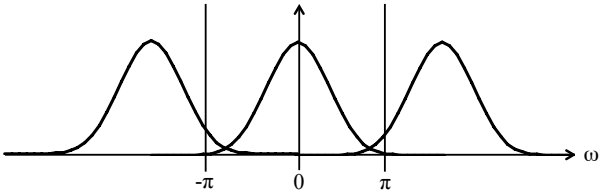
The Fourier Transform of a Gaussian is a Gaussian, with a scale inversely proportional to the scale of the Gaussian. $F\{g(x,\sigma)\} = e^{-\frac{1}{2}\omega^2\sigma^2}$



As the space domain Gaussian grows larger, it spectrum grows sharper.
Sampling the Gaussian creates a periodic Fourier Transform:



These copies add to create a composite spectrum. Any energy beyond $\omega_s/2= \pi F_s$ is added to the next copy as noise. To minimize this effect, we need to assure that the sample rate is sufficiently small or that the scale of the Gaussian is sufficiently large.



This requires that we use $\Delta x \leq \sigma/2$.
For $\sigma/\Delta x < 2$, aliasing folds a significant amount of energy at the sampling frequency, corrupting the quality (and the invariance) of the Gaussian function.

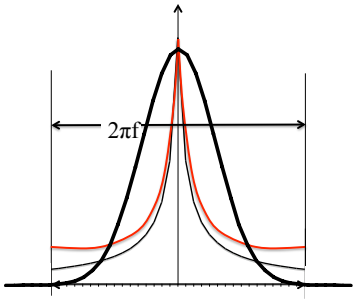
Convolving the image signal with a Gaussian low pass filter,

$$(S * g)(n) = \sum_{m=-\infty}^{\infty} S(m)g(n-m,\sigma)$$

is the same as multiplying in Frequency,

$$S(\omega) \cdot G(\omega,\sigma)$$

This has the effect of suppressing the high frequency energy in the signal that would be aliased by sampling



The larger the Gaussian, the better the suppression of noise. However, convolution blurs the image, and must be kept to a minimum. A good compromise is to use a sample size of less than half the sigma of the Gaussian.

$$\Delta x_k = \Delta y_k \leq \frac{1}{2}\sigma_k$$

smaller sampling is even better, but there little improvement for going below 1/4

$$\Delta x_k = \Delta y_k = \frac{1}{4}\sigma_k$$

Windowing

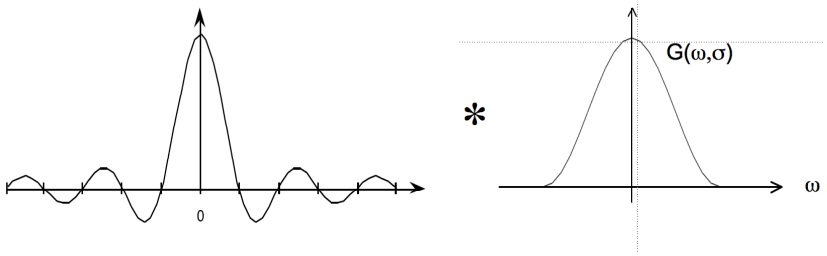
Truncating an infinite function to a finite duration is equivalent to multiply by a window $W_N(x)$. When we limit $g(x,\sigma)$ to a finite support, we multiply by a window

$$G(n,\sigma) = G(n,\sigma)\cdot W_N(n) \text{ where } W_N(n) = \begin{cases} 1 & \text{for } -R \le n \le R \\ 0 & \text{otherwise} \end{cases}$$

Multiplying by a finite window is equivalent to convolving with the Fourier transform of the finite window:

$$F\{G(n,\sigma)\cdot W_N(n)\} = G(\omega,\sigma) * W_N(\omega)$$

where $W_N(\omega) = \dfrac{\sin(\omega N/2)}{\sin(\omega/2)}$ and $G(\omega,\sigma) = \sqrt{2\pi}\,\sigma\, e^{-\frac{1}{2}\sigma^2\omega^2}$



For N < 7, the ripples in $W_N(\omega)$ dominate the spectrum and corrupt the resulting Gaussian.
At N=7 the effect is tolerable but noticeable.
At N≥ 9 the effect becomes minimal

2.5. 2D Gaussian Digital Filters

In 2D the Finite impulse response Gaussian filter is : $G(i,j,\sigma) = \dfrac{1}{B} W_N(i,j)\cdot e^{-\frac{(i^2+j^2)}{2\sigma^2}}$

Where i and j are integers,

$$w_N(i,j) = \begin{cases} 1 & \text{for } -R \le i \le R \text{ and } -R \le j \le R \\ 0 & \text{otherwise} \end{cases}$$

The Gaussian function has infinite spatial extent. To provide a digital filter we limit the spatial extent of the Gaussian with a Window function $W_N(i,j)$

The finite window, $W_N(i,j)$ has $N^2 = (2R+1)^2$ coefficients

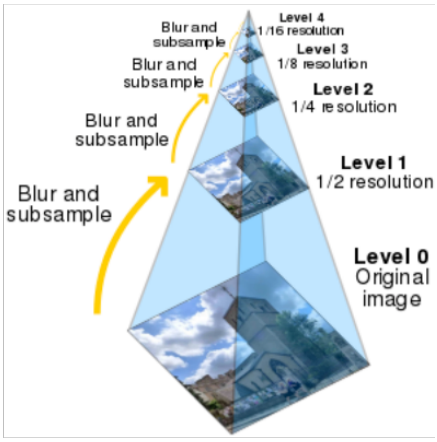
Where R is a "radius. Typically R should be $R \ge 3\sigma$.
A normalization factor assures that the "Gain" of the filter is 1.

$$B = \sum_{x=-R}^{R} \sum_{y=-R}^{R} e^{-\frac{(i^2+j^2)}{2\sigma^2}} \le 2\pi\sigma$$

3. Image Pyramids

An image pyramid is a multi-scale representation in which an image has been low-pass filtered and resampled to an appropriate resolution over an exponential range of sample rates. To prevent aliasing (additive random noise from sampling), each image must be smoothed with a low-pass filter whose impulse response is proportional to the sample rate.

Pyramid representations provide scale invariant image description. When properly computed, each image in the pyramid will have an identical impulse response, enabling computation of scale invariant features for recognizing phenomena independent of scale.



Pyramid image from wikipedia
(https://en.wikipedia.org/wiki/Pyramid)

Image pyramids are classically computed using a Gaussian Low pass filter. However a fast integer coefficient version is possible using Binomial low pass filters, which are integer approximations to Gaussian low pass filters. Gaussian filters (and their binomial equivalents) are interesting because of interesting scale invariant properties.

Computing the difference of adjacent images in a Gaussian pyramid provides a Laplacian Pyramid. Laplacian pyramids are widely used to detect scale invariant images features, for example with the widely used "SIFT" image descriptor (Scale Invariant Feature Transform").

Gaussian pyramids may be computed using a fast recursive algorithm that has computational complexity of O(N), where N is the number of pixels, and results in a O(N) representation the image.

3.1. Direct Calculation of an Image Pyramid

Let, *p(i,j),* be an image array of size *RxC* pixels, where *(i, j)* are integers. To compute the scale space representation of the image we must sample scale space in x, y and in σ. As we have seen above, σ is generally sampled over an exponential range.

$$\sigma_k = \sigma_0 \Delta s^k$$

The smallest value of is σ_k is σ_0 when k=2 as $\Delta s^0 = 1$.
The largest value of scale is determined by the image size, say

$$\Delta s^{k_{\max}} = \max(R,C)$$

Thus we can have maximum of $K = \text{Log}_2\{\max(R,C)\}$ levels in the pyramid.

We can convert our image to a continous scale space with:

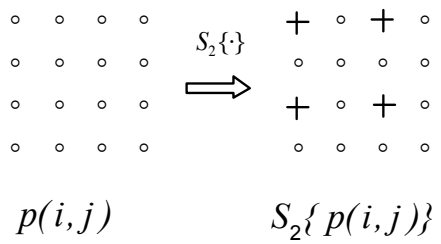
$$P(x,y,k) = p(i,j) * g(x,y,\sigma_k) = \sum_{i=-R_k}^{R_k} \sum_{j=-R_k}^{R_k} p(x-i,y-j) g(i,j,\sigma_k)$$

where $\sigma_k = \sigma_o \Delta s^k$. Typical values would be σ_0=2 and Δs=2.

Each image can be resampled by a sampling operator that computes every Δx_k pixel of every Δy_k row:

$$P(i,j,k) = S_2^k\{P(x,y,k)\}$$

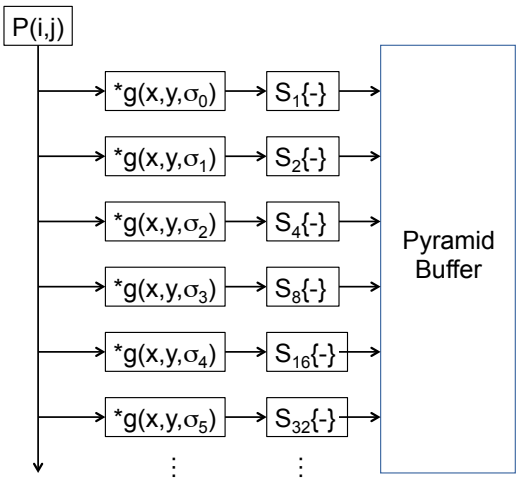
For example, this is the result of a sample rate of $\Delta x = \Delta y = 2$.



$p(i,j)$ $\qquad$ $S_2\{p(i,j)\}$

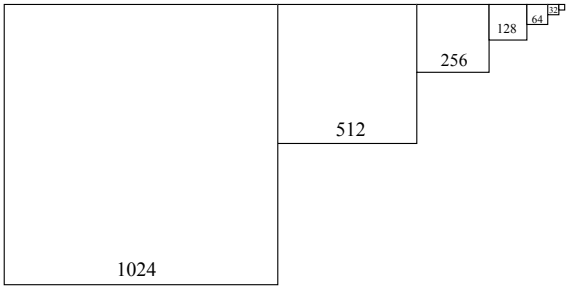
For scale equivariance, the spatial sample rates Δx_k and Δy_k should be chosen so that the impulse response at each image is the same. As we saw above, aliasing can be remedied by using $\Delta x_k = \Delta y_k \leq (1/2)\ \sigma_k$.

With the above values the algorithm for this would look like this:



$\Delta x_k = \Delta y_k = \Delta s^k = 2^k$.

The result is a collection of images of decreasing size. For example for an image of 1024 x 1024:



The number of pixels is 1024x1024 (1+ 1/4 + 1/16 + 1/64+...) < 1.5 x 1024^2

However, the cost for computing this pyramid is very high.
For an image of 1024 x 1024 pixels, the largest Gaussian would be σ_k=128, and the pyramid would have K=8 levels from 0 to 7.

For an image of RxC pixels, where C >R this still requires O(R^2) operations.
For a 2^{10} by 2^{10} image, this is O(2^{20}) operations per pixel.

The cost for computing the largest scale (smallest window) approaches the number of pixels to the 4th power! $1024^4 = (2^{10})^4 = 2^{40}$.

We can gain some improvement by noting that the Gaussian is separable, giving a convolution that costs 2N rather than N^2.

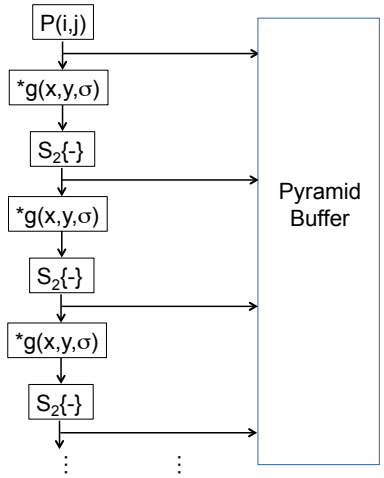
$$P(x,y,k) = P(x,y) * G(x,y,\sigma_k) = P(x,y) * G(x,\sigma_k) * G(y,\sigma_k)$$

However, for $\sigma_k = 2^{k+1}$ we still have an exponential growth $\sigma_k = (2,4,8,16,...)$.
We can do better than this.

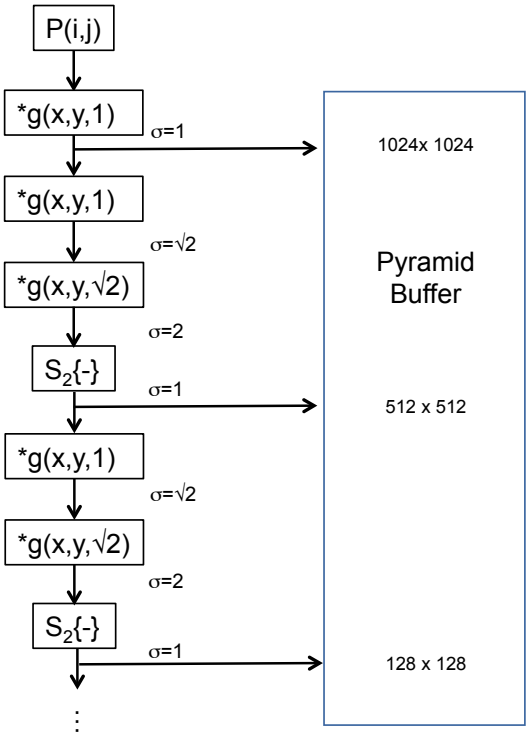
3.2. Scale Invariant Pyramid Algorithm

To overcome the exponential growth, we can use each level in the pyramid to compute the next level. This yields a form of recursive algorithm referred to as "cascade convolution".

$$P(x,y,k) = P(i,j,k-1) * g(x,y,\sigma_k) = \sum_{i=-R_k}^{R_k} \sum_{j=-R_k}^{R_k} p(x-i,y-j,k-1) g(i,j,\sigma_k)$$



However, the resulting algorithm does not have a scale invariant impulse response. To provide a scale invariant impulse response, we need to do an initial convolution with a Gaussian at $\sigma=\sigma_0$, and then compute each successive level with a Gaussians of $\sigma=\sigma_0$ and $\sigma=\sqrt{2}\sigma_0$. When $\sigma_0=1$ this algorithm would look like this.



The cost of this algorithm is on the order of the size of the filter $g(x,y,\sigma_0)$ times the number of resulting samples. For example, consider a pyramid constructed using Gaussian filters with $\sigma_0=1$ implemented with a 7x7 filter and $\sigma=\sqrt{2}$ is implemented with a 9x9 filter. The separability property of the Gaussian tells us that each convolution can be implemented as a row convolution followed by a column convolution. Thus the cost for the first level is 2x7=14 multiplies and adds. The cost for each successive level is 2x7 +2x9 = 32 multiplies and adds per pixel. The total number of multiplies and adds is 14 + 32 (1 + 1/4 + 1/16 +...) = 64 multiplies and adds per pixel, and the total number of samples in the pyramid is NxNx(1 + 1/4 + 1/16 +...) or approximately 1.5 x N x N. This is easily computed at video rate for most computers.

Using $\sigma_0=\sqrt{2}$ would result in a reduction of aliasing at slight increase in computing cost.

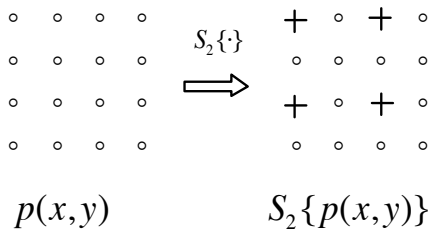
There are a number of additional optimizations that are possible, but these go beyond the scope of this lecture.

3.3. Sampling

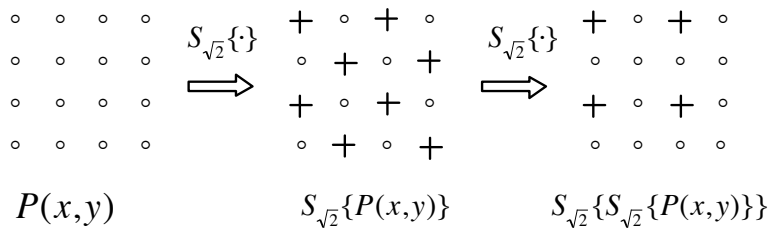
Resampling *P(x, y, k)* at $\Delta x_k \sim \sigma_k$ results an identical impulse response at each level. Pyramid samples are at discrete positions $(i\Delta x_k, j\Delta x_k)$ for integer values of i, j:

$$P(i, j, k) = P(i\Delta x_k, j\Delta x_k, k)$$

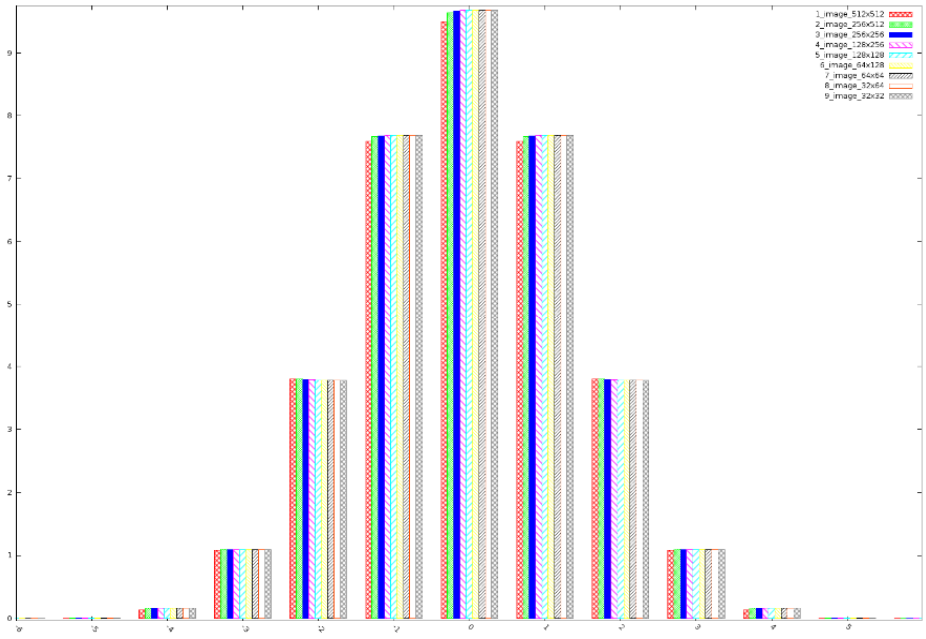
The position in the original image of a sample from level k is $x = i\Delta x_k$ and $y = j\Delta y_k$ If we sampling at a scale step of $\Delta x_k = 2^k$ this gives a "full octave" pyramid.



It is also possible to build a scale invariant pyramid with a step size of $\Delta x_k = 2^{k/2}$ using $\Delta x_k = 2^{k/2}$ This is known as a "half-octave" pyramid.



An impulse response is the output of a convolution when the input is a single impulse (Dirac Delta function). The following figure show the "impulse response" for a Scale Invariant pyramid at 6 different scales, computed using a fast O(N) algorithm. This was taken from a half-octave pyramid algorithm using root-2 sampling, but the principle is the same.



4. Invariant and Equivariant Properties in Gaussian Scale Space

Gaussian Scale Space has a number of very interesting invariant and equivariant properties. Differential operators can be used to detect key-points (interest points) in scale space. These can be used to define invariant structures that represent appearance independent of scale, position or image plane rotation.

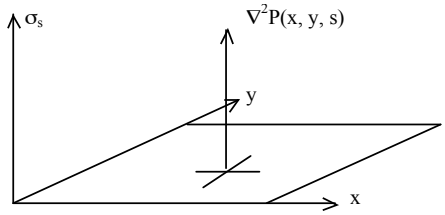
Keypoints are typically found using differential operators such as the Gradient (first derivative) or the Laplacian (second derivative).

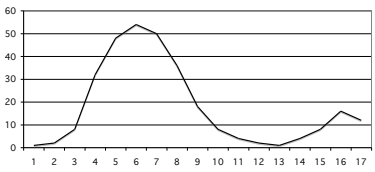
An example is the Laplacian Profile. Recall that the Laplacian of an image is:

$$\nabla^2 P(x,y,s) = P * \nabla^2 G(x,y,\sigma_s) = P_{xx}(x,y,s) + P_{yy}(x,y,s)$$

When evaluated over an exponential range of scales, the Laplacian gives a Laplacian profile. This is an invariant descriptor for appearance.

When computed over an exponential range of scales, the Laplacian profile is invariant to rotation and translation and equivariant to changes in scale. Since scale is proportional to distance, the profile is equivariant to viewing distance.





A change in viewing distance at x, y shifts the function $\nabla^2 P(x,y,s)$ in s. The form of the profile translates in scale but remains the same. Technically this is called a "covariant".

A Laplacian interest point is found by $(x_i, y_i, s_i) = \underset{x,y,s}{local-\max}\{\nabla^2 P(x,y,s)\}$

Local-Max{} returns any point (x, y, s) for which the value of the function is larger than all neighbors within some distance ε.

We will see this in the next lecture