

# Features

*All learning notes are based on the course "Signal and Image Processing" taught by UCPH.*

## Features

Image feature detection

Why image features?

Image derivatives revisited

Recall: Differentiation filters amplifies noise

Image derivatives with Gaussian filters

Scale space of derivatives  $L_x(x, y; \sigma) = I * \frac{\partial G_\sigma}{\partial_x}$

How to compute Gaussian derivatives in practice

Are derivatives useful?

Intenstiy edges

Edge detection

Algorithm

Edge detection by local maxima of gradient magnitude

The Canny edge detector

Zero crossings of Laplacian of Gaussian filter

Local gauge coordinate system: Enforcing rotation invariance

Directional derivatives

Gradient magnitude is rotational invariant

Edge detection (Lindeberg's approach)

Sparse interest point detectors

Application

Matching image neighborhoods for 3D vision

Salient points

Salient points a.k.a. Interest points

How to automatically detect salient points?

Template matching - correlation

Intensity blobs as salient points

Blob detector: A simple 1D example

Detecting blobs by Laplacian of Gaussian filter

Detecting blobs

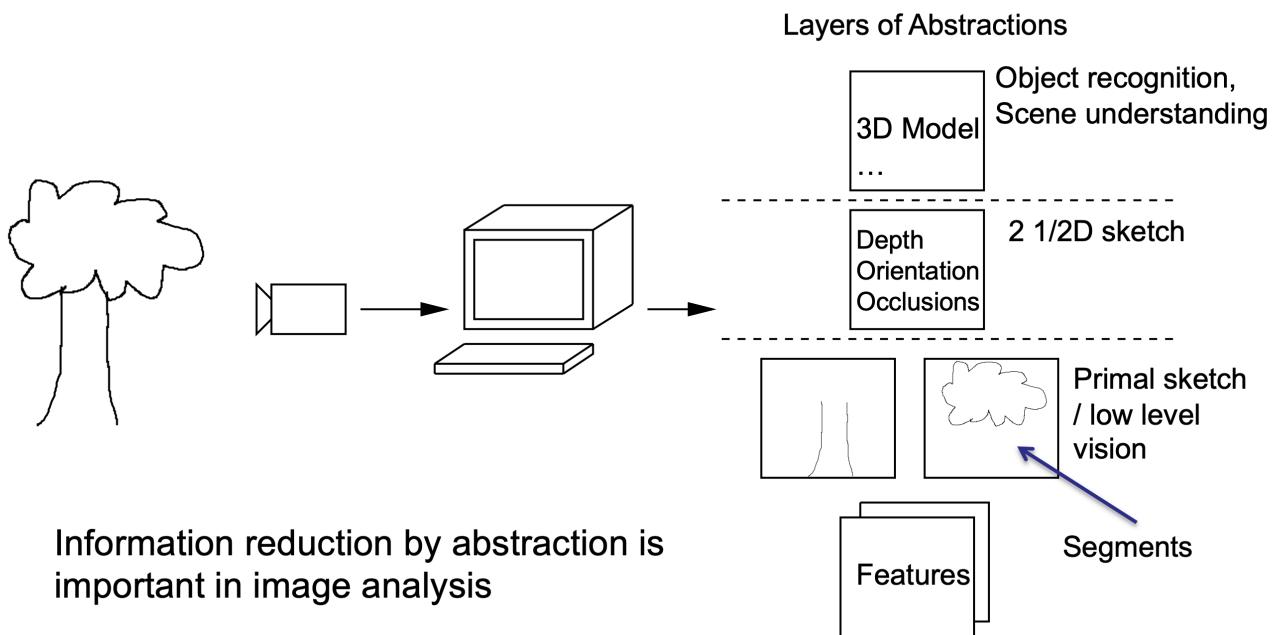
Pitfalls - Laplacian of Gaussian filter

Intensity Corners

# Image feature detection

## Why image features?

Marr's layers of abstraction for vision systems (1982)



This is an old picture of how and we can do analysis of images, and it comes from an old book on how to build vision systems. Even though the content of the model might be debatable, the core idea is still very much useful for us.

So the concept is that, at the lowest level, any image that we have the broad pixels in the digital image. But when we want to do any form of analysis or interpretation of what we see in the image, we need to build up **abstractions** or based on these pixels.

Concretely, you can think of abstractions as being functions of the pixels, but we would like to extract some information from the pixels, for instance saying that, if we look at this particular scene, here we have a tree, we have a camera and a computer. The computer builds up some abstraction. And at the lowest level, we have something that in image analysis and computer vision. We call this, **features**. This could for instance be to identify like trunks, or the parts of an object this could be by segmentation. For instance, I could identify a tree by identify a tree trunk and could identify the tree crown, leaves, and so on.

But as part of this, we might need to have even more fine detail descriptions of what we see here in order to actually form up these parts.

Now, on top of this, we might start to do things like inferred depths that means what is the distance from the camera to objects in the scene?

We can also say something about orientation of objects, what are potential occlusion boundaries that is one object being in front of another object hiding the object behind?

Maybe the end goal is to form a 3D model of the scene. It could also be to say I am looking at a tree.

So => This is sort of the pipeline in image analysis. There is something low level going on and on top of that, we add more and more complex or more abstract descriptions of what we see in the scene until at the end, we have a understanding of what we see in the image.

Today, we will look at features and what this means.

所以这个概念是，在最低层次上，我们有数字图像中的像素。但是当我们想要对图像进行任何形式的分析或解释时，我们需要基于这些像素构建抽象。

具体来说，您可以将抽象视为像素的函数，但是我们想要从像素中提取一些信息，例如说，如果我们观察这个特定的场景，我们有一棵树、一台相机和一台电脑。电脑建立了一些抽象。在最低层次上，我们在图像分析和计算机视觉中称之为特征。这可以用于识别像树干一样的物体部分，或者是通过分割识别对象的各个部分。例如，我可以通过识别树干来识别树冠、叶子等。

但是作为此过程的一部分，我们可能需要对所看到的细节进行更详细的描述，以实际形成这些部分。

此外，我们可能会开始进行类似推断深度的操作，即意味着相机到场景中物体的距离是多少？

我们还可以说一些关于物体方向的信息，可能会有潜在的遮挡边界，即一个物体位于另一个物体前面，隐藏在其后面的物体？

也许最终目标是形成场景的三维模型，也可能是说我在看一棵树。

所以 => 这就是图像分析中的流水线。有一些低层次的事情正在发生，然后我们在其上添加更复杂或更抽象的描述，直到最后，我们对图像中所看到的内容有了理解。

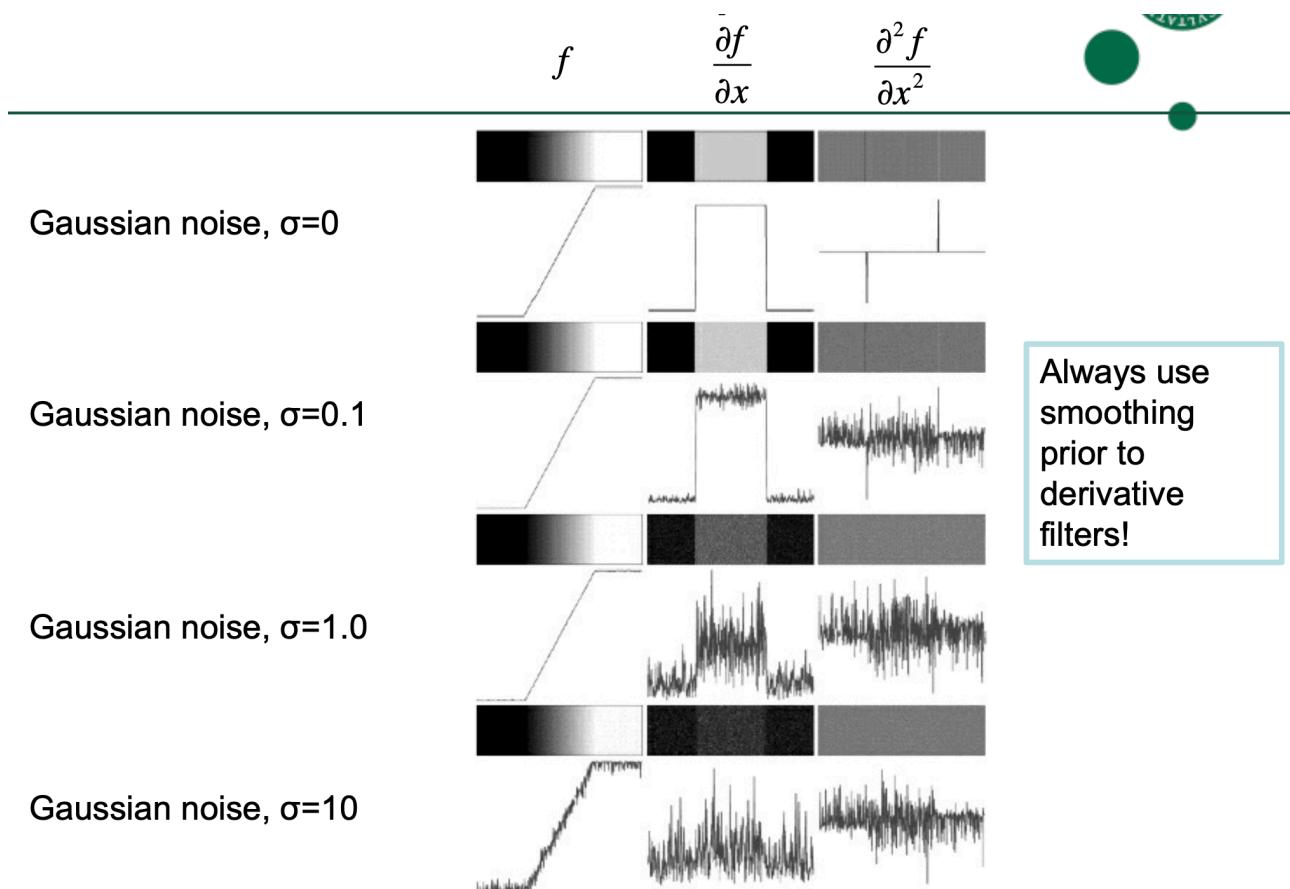
今天，我们将了解图像特征及其含义。

---

我的理解：在图像分析中，电脑首先通过数字图像中的像素形成电子信号，然后通过从信号中提取特征（如树干、树冠、叶子等）的函数来分析图像。这是最底层的操作，然后我们可以通过将更多的抽象层次加入到这些特征中，来进一步分析和解释图像中所看到的内容。

## Image derivatives revisited

### Recall: Differentiation filters amplifies noise



## Image derivatives with Gaussian filters

Use a Gaussian filter  $G_\sigma$  to smooth prior to computing image derivatives

在计算图像的导数时，通常会出现噪声或细节信息，这可能会导致导数的误差或不准确性。因此，在计算导数之前，通常会使用高斯滤波器平滑图像，以减少这些噪声或细节信息的影响。高斯滤波器可以平滑图像并消除高频噪声，从而使图像变得更加平滑，有助于更精确地计算导数。高斯滤波器使用高斯核函数对图像进行卷积操作，这个高斯核函数有一个参数  $\sigma$ ，用于控制滤波器的平滑程度。

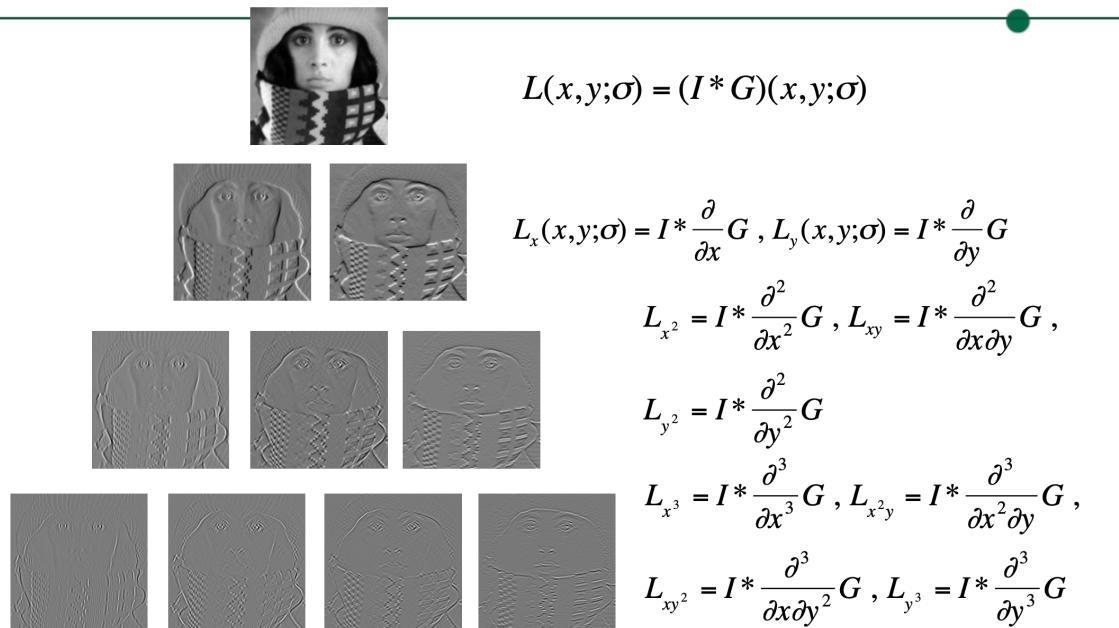
Note that due to the commutative property of derivatives and convolution: (换元特性)

$$\frac{\partial L_\sigma}{\partial x} = \frac{\partial}{\partial x} (G_\sigma * I) \xrightarrow{\mathcal{F}} 2\pi i u(\hat{G}_\sigma \hat{I}) = (2\pi i u \hat{G}_\sigma) \hat{I} \xrightarrow{\mathcal{F}^{-1}} \frac{\partial G_\sigma}{\partial x} * I$$

Hence we only need to compute **derivatives of the Gaussian prior to convolution**,

$$\frac{\partial L_\sigma}{\partial x} = \frac{\partial G_\sigma}{\partial x} * I$$

这句话的意思是由于求导和卷积操作满足交换律，我们可以选择先对高斯函数进行求导再与图像进行卷积，这样可以减少计算量。因为对高斯函数求导可以使用预先计算的一组卷积核来实现，这些卷积核被称为高斯差分算子（Gaussian Derivatives）。通过使用这些卷积核，我们可以避免对每个像素都进行高斯滤波和求导的操作。原本是先用高斯滤波卷积图像再求导，由于交换律，可以先对高斯滤波求导再卷积。



$L(x, y; \sigma) = (I * G)(x, y; \sigma)$  这个表示一个图像  $I$  经过高斯滤波  $G$  以后得到的平滑图像  $L$ ，其中  $x$  和  $y$  是图像中的坐标， $\sigma$  是高斯滤波的参数，控制滤波的程度。可以理解为  $L$  是对  $I$  进行一种平滑处理后得到的图像。

**Scale space of derivatives**  $L_x(x, y; \sigma) = I * \frac{\partial G_\sigma}{\partial x}$

在计算机视觉和图像处理领域中，scale-space（尺度空间）是指用不同的尺度（例如不同的高斯滤波器大小）来分析图像的方法。使用尺度空间可以让我们在不同的尺度下分析图像，从而获得更丰富的图像信息。例如，在分析目标物体的边缘时，我们可以使用不同尺度的高斯滤波器来平滑图像并计算不同尺度下的边缘。这种方法可以让我们在不同尺度下检测到不同大小和形状的边缘，从而提高我们对图像的理解和分析。因此，尺度空间是图像处理和计算

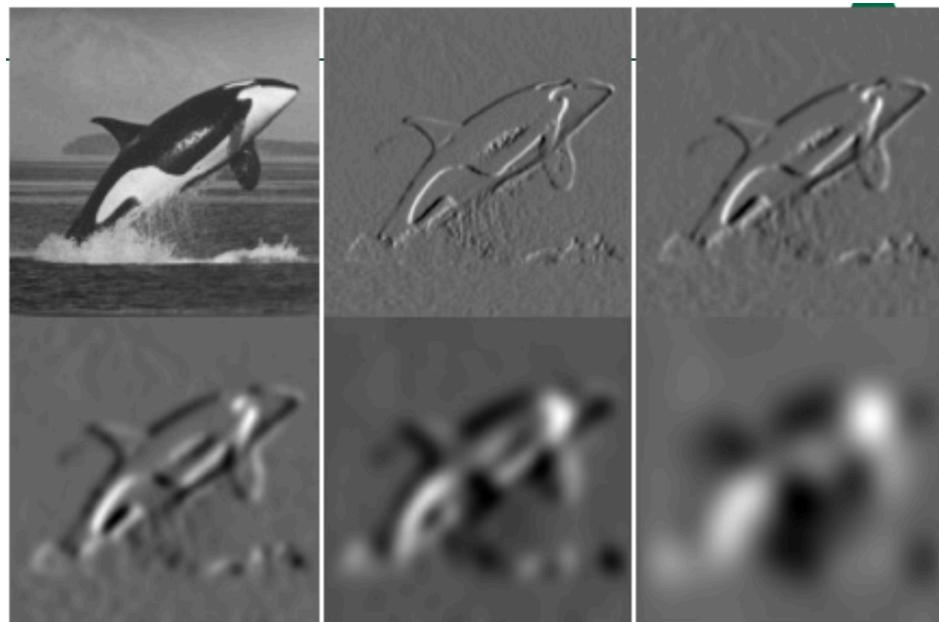
机视觉中非常重要的概念。

In the context of image processing and computer vision, the scale-space of derivatives refers to a sequence of smoothed and differentiated images obtained by convolving an input image with Gaussian filters of varying scales (represented by the parameter  $\sigma$ ).

$L(x, y; \sigma)$  represents the output of applying the Gaussian filter with standard deviation  $\sigma$  to the input image  $I$  at location  $(x, y)$ . This operation smooths the input image to different degrees depending on the scale  $\sigma$ , effectively removing high-frequency details and noise.

$L_x(x, y; \sigma)$  represents the derivative of the smoothed image with respect to the x-axis, obtained by convolving the smoothed image with the derivative of the Gaussian filter with respect to x, denoted as  $\partial G\sigma / \partial x$ . Similarly,  $L_y(x, y; \sigma)$  represents the derivative of the smoothed image with respect to the y-axis, obtained by convolving the smoothed image with the derivative of the Gaussian filter with respect to y, denoted as  $\partial G\sigma / \partial y$ .

So,  $L_x(x, y; \sigma) = I * \partial G\sigma(x, y) / \partial x$  represents the convolution of the input image with the derivative of the Gaussian filter with respect to x, which results in a smoothed and differentiated image with respect to the x-axis.



$$\sigma = 1, 2, 4, 8, 16$$

## How to compute Gaussian derivatives in practice

In Python, we can use `scipy.ndimage.gaussian_filter`

Computing the x-derivative at scale given by sigma:

```
sigma = 1.0
derivative_ordre = (0, 1) # Row-col format
Lx = gaussian_filter(I, sigma, order=derivative_order)
```

这段代码是用Python中的scipy.ndimage.gaussian\_filter函数来计算高斯导数的。

sigma = 1.0表示高斯滤波器的标准差为1.0。

derivative\_order = (0, 1)表示我们要计算的是x方向上的一阶导数。这里使用的是“行-列”格式，即第一个元素表示行方向上的导数，第二个元素表示列方向上的导数。所以这个元组实际上表示计算x方向上的一阶导数。

Lx = gaussian\_filter(I, sigma, order=derivative\_order)表示对图像I进行高斯滤波并计算x方向上的一阶导数，结果存储在Lx中。order参数指定了要计算的导数的阶数和方向。在这里，order=derivative\_order表示计算derivative\_order中指定的方向和阶数的导数。

如果是y方向上的，则将(0, 1) => (1, 0)即可。

## Are derivatives useful?

Derivatives permit the calculation of many **invariants** which are functions that are invariant with respect to certain transformations of the image domain.

Invariants can be used to define image features.

Intensity edges, ridges, corners are all examples of image features.

导数可以计算许多不变量，即对于图像域的某些变换具有不变性的函数。这些不变量可以用于定义图像特征。例如，强度边缘、脊线和角点都是图像特征的例子。

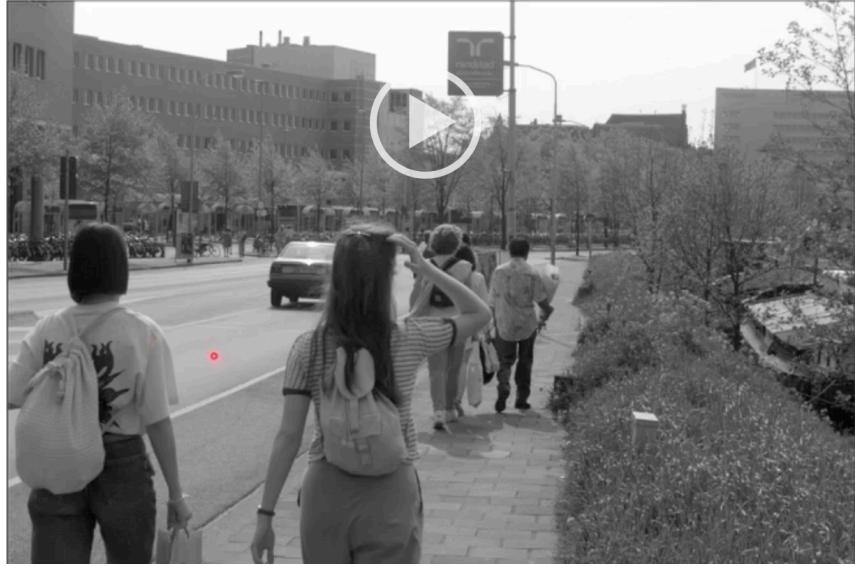
## Intenstiy edges

Then, let's start by looking at a concrete type of image features. Namely, the **intensity edges**.

## Edge detection

**Useful for segmentation and object boundaries**

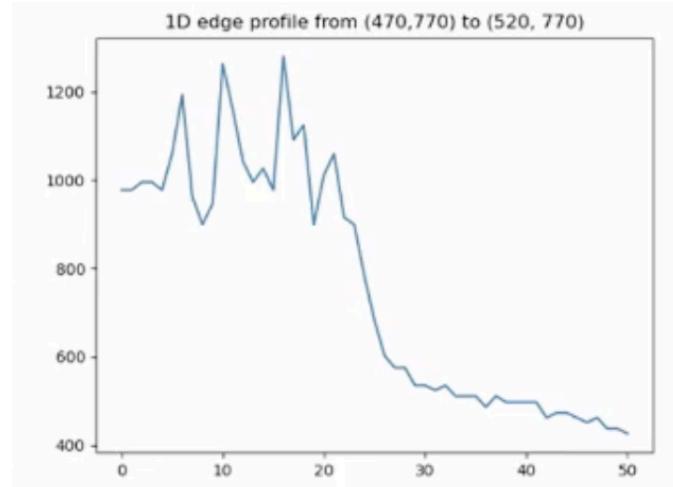
An **intensity edge** is defined by **locations in the image of abrupt changes in intensities**.



So for instance, if we look at this image, there are several places where the intensities go from darker to brighter.

For instance, we have this person's shirt here, we can sort of easily see that something going on and actually here even though we can see it as a white shirt, it looks a bit dark here and then, you have sort of brighter pixels out in the asphalt. => 就是在这个图里面，我们看左边这个人的衬衫，虽然它是白色的，但是和路上的沥青相比，沥青的强度更大。

So if we take a cut through what a part of the image, so let me just pick out a line here. Cut the pixels out and visualize these.



Here, we can see in the background, there's some asphalt here appears slightly brighter and that is seen as high intensity values in the image.

Similarly, on this person's arm here, we have sort of darker pixels, that means we have lower intensity values.

So, we can sort of see that through this cut here, we have something that looks a bit like abrupt changes in the intensity level. Just moving along the line from inside the arm out of asphalt.

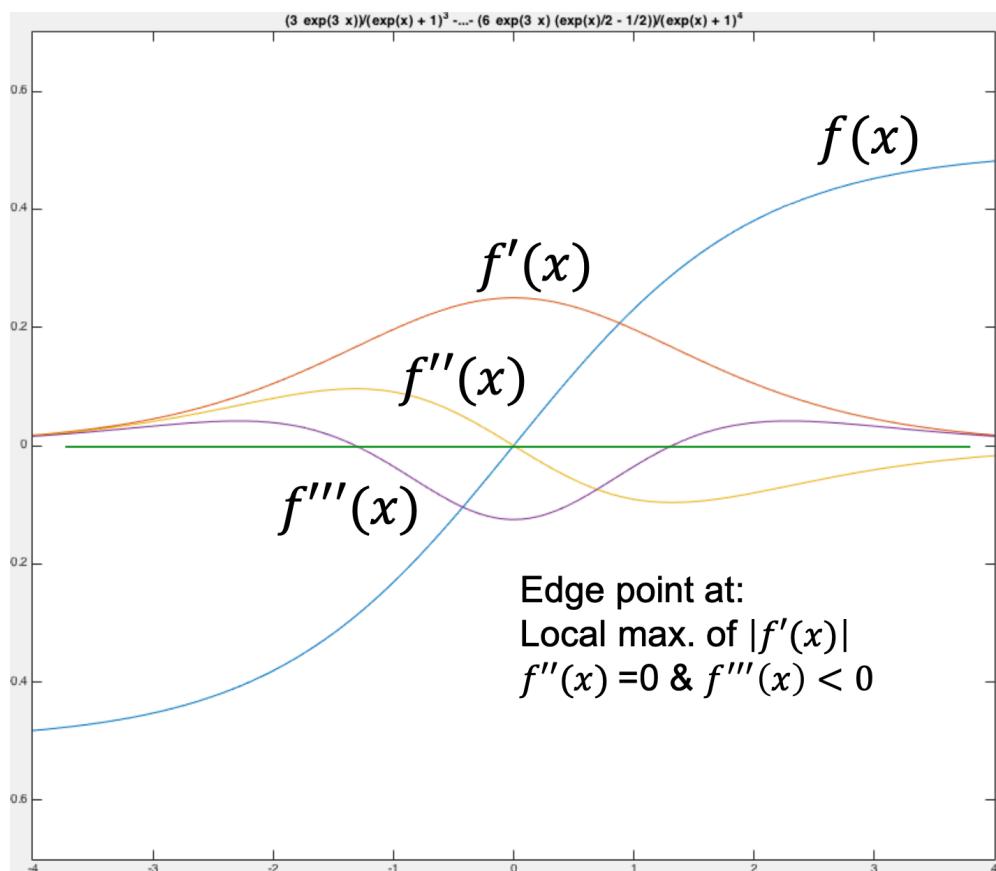
So, from the right part and all the way into the asphalt (left part).

This is basically what we call an intensity edge and of course if we consider something like this person's arm, we could probably find like several locations here where we have an abrupt change probably like a complete curve of pixels here with this but looks something similar to this.

This is an example of an intensity edge.

## Algorithm

And we can make algorithms to detect this, but let's start with a simplified case here.



In blue, I have an idealized edge, idealized intensity. So again, you can think of this as being cut out of some image with taking out a line of pixels and look at the values there.

I also plot the first order derivative that is reddish curve, and the second order derivative which is the yellow curve. And the third order derivative which is purple curve.

Now, where if I have this blue change intensity, where would I like the edge to be? Does it start down here or does it start up here. Or do we want to have the point that sort of at the location where the edge is changing the most? So that's a choice we have to make.

But a common decision for this is to say. Let me go for the location of most change. So, if we look at this curve, it's changes fast here. So, having the steepest descend or ascend depending on whether you go up or down. This can easily define by looking at the derivative.

So, this location here with the most change that's basically given by finding a local maximum of the first order derivative to do that. You can just remember basic function analysis that then we probably just need to find locations where the second derivative is equal to zero. And we also need to check that the third derivative is negative at this location.

在蓝色曲线中，我画了一条理想化的边缘和理想化的强度变化曲线。因此，您可以将其视为从某个图像中切出一行像素，并查看那里的像素值。

我还绘制了一阶导数（红色曲线）、二阶导数（黄色曲线）和三阶导数（紫色曲线）。如果我有这种蓝色的强度变化，那么我想要的边缘在哪里呢？它是从这里开始还是从这里开始？或者我们希望在边缘变化最大的位置？这是我们做出的决定。

但是通常的决策是说。让我选择变化最大的位置。因此，如果我们观察这条曲线，它在这里变化最快。因此，在具有最大变化的位置，我们可以通过查找一阶导数的局部最大值来找到边缘的位置。为了做到这一点，您只需要记住基本函数分析，然后我们可能只需要找到二阶导数等于零的位置。并且我们还需要检查这个位置的三阶导数是否为负数。

This would be a way to define an edge.

I define this in 1D, so we need to extend this to 2D.

## Edge detection by local maxima of gradient magnitude

In order to do this, we use the tools from that I presented in the scale space videos that we can compute this gradient magnitude which is that we take for the two first order derivative so that first order derivative along the x-axis and the first order derivative along the y-axis.

We think of that as forming a vector namely the gradient vector and then we compute the length of this vector.

$$\|\nabla L\|(x, y; \sigma) = \sqrt{(L_x)^2 + (L_y)^2}$$

Or equivalently, we could also compute the gradient magnitude squared. That the only difference areas whether or not I take the square root.

$$||\nabla L||^2(x, y; \sigma) = (L_x)^2 + (L_y)^2$$

It does not matter whether I take the square or not, it does not change location.

### **So, how can I use this?**

What we just saw here in before in 1D, what would be to look for location where the expression has local maxima.

So this is maxima both across  $x$  and  $y$  in space.

For the fun of it, I've computed is gradient magnitude squared the result of that of applying it to the image.



So we have bright values large values at something that actually looks like the type of intensity edges that we are looking for.

But, let me start by cheating a bit. I can do this very simple by just saying let me pick a threshold value, and apply thresholding to each pixel.

I keep only pixels where this gradient magnitude square value is bigger than my threshold. Then, we get something looks like this. (binary image). Each location insider is 0 or 1 because this is the result of the thresholding.



梯度是根据每个像素的 x 和 y 的值计算出来的，具体地，对于一个图像上的像素  $(x, y)$ ，它的梯度可以表示为：

$$\nabla f(x, y) = (G_x(x, y) \ G_y(x, y))$$

其中  $G_x(x, y)$  和  $G_y(x, y)$  分别是在  $(x, y)$  处计算的图像在水平和竖直方向上的梯度值。

至于阈值，它通常是根据具体的任务和图像特性来设置的。例如，在边缘检测中，我们希望尽可能地准确地检测出图像中的边缘，因此可能需要设置一个较低的阈值，以保留更多的梯度强度信息。而在一些特定的应用中，比如目标检测和图像分割等任务中，可能需要设置一个较高的阈值，以过滤掉一些不相关的信息。

给出的公式和  $|\nabla L|(x, y; \sigma) = \sqrt{(L_x)^2 + (L_y)^2}$  表达的是同一个意思，其中  $L_x$  和  $L_y$  分别表示图像  $L$  在  $x$  和  $y$  方向上的梯度， $|\nabla L|$  表示梯度的大小。而  $\sigma$  是高斯滤波器的参数，用于控制滤波器的大小。在计算梯度之前，可以先用高斯滤波器对图像进行平滑处理，以减少噪声的影响。 $\sigma$  越大，滤波器的大小就越大，平滑的效果也就越好，但是边缘的细节也就越模糊。

But what don't we have something out there for instance, which would correspond to this edge here and similar on the other side? => Poor choice of threshold value.

But we can also see that at some of the places, the amount of pixels that were picked up, it looks like only one point per line and that's good. But here it actually looks like we got like a fat line of pixels that are all above the threshold. => *Extra points and this is the problem with this thresholding approach. And cheating not try to find local maxima.*

There are good solutions to handle this problem to move away from the pure thresholding idea.

There's also fact that I selected a specific scale  $\sigma$  when I computed the derivative and that means that I am focusing on details that are larger than that scale that I chose. Everything below that scale got blurred away so we can't really see it anymore.

## The Canny edge detector

1. Smooth the image by a Gaussian filter for **noise suppression**
2. Compute **gradient magnitude** and **orientation images**

什么是 **Orientation images**? 在尺度空间中，Orientation images 指的是对于每个尺度，图像中的每个像素都有一个方向信息。它们可以通过计算图像中每个像素的局部方向来获得，通常使用梯度方向来表示。因此，对于每个尺度，我们可以得到一个方向图像，它可以用来描述在该尺度下的方向特征。这些方向图像可以用于许多计算机视觉任务，如纹理分类、形状识别等。

Gradient magnitude:

$$\|\nabla f\|(x,y) = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

3. Use double thresholding (hysteresis thresholding) and analysis of connectivity to detect and link edges. => High and low thresholds.

Start by applying the high threshold value getting a bunch of candidate pixels that are probably edges, and then we can do a bit of edge thinning which is step 4.

And then, we check any pixels that survive the low threshold value and is connected with one of the high threshold value each pixel. If this is the case, then we also consider this as a potential edge point. Si we ciykd sirt if add additional points that sort of got lost or that we didn't catch in the first high thresholding.

4. Apply non-maximum suppression to the gradient magnitude image by considering neighboring pixels along the gradient direction (edge thinning). => 找局部最大值。Yes, that's correct. Non-maximum suppression is a technique used in edge detection to thin out the edges to a single-pixel wide line while preserving their strength or magnitude. The basic idea is to look at each pixel in the gradient magnitude image and suppress (set to zero) any pixel that is not a local maximum along the direction of the gradient. This helps to refine the edges and remove any spurious responses.

Step 3 and 4 are in implementation combined into one step, and can be used on other definitions of edge points.

非极大值抑制（NMS）和双阈值处理（double thresholding）。非极大值抑制是通过比较邻边像素上的梯度来确认是否为真正的局部最大值，从而帮助减少噪声引起的误检。而双阈值处理则是将所有梯度值分为强边缘、弱边缘和非边缘三类，从而进一步过滤掉非边缘和弱边缘，保留强边缘，形成连续的边缘轮廓。

*Hysteresis thresholding:*

Define two threshold values, `low_thres` and `high_thres` such that  $low\_thres < high\_thres$ .

All points with gradient magnitude above `high_thres` is an edge point

All points between `low_thres` and `high_thres` is an edge point if a neighbouring point is a `high_thres` edge point

If we do this, below is the example.



非极大值抑制（Non-Maximum Suppression，简称NMS）是指在边缘检测中，通过比较梯度方向上的像素，找到梯度幅值的局部最大值，来抑制非极大值，从而保留较精确的边缘。一般来说，当梯度幅值图像上的一个像素是其在梯度方向上的相邻两个像素中幅值最大的时候，我们称其为局部极大值。

在Canny边缘检测算法中，使用NMS算法来抑制非极大值，可以通过检测像素点的梯度和方向，保留在梯度方向上的局部最大值。然后，通过设置两个不同的阈值（高阈值和低阈值），将梯度幅值分成强边缘和弱边缘两个部分。所有大于高阈值的边缘点都被视为真正的边缘点，而所有小于低阈值的点都被认为是非边缘点。在介于两个阈值之间的边缘点将被根据其是否与真正的边缘点相连而被视为边缘点或非边缘点。这样就形成了Canny边缘检测的最终结果。

Observing the image, we can clearly see that we now have thin edges so they are only one pixel wide and it also looks like we've been able to identify points along edge here. That's almost no holes, otherwise, it looks pretty good. => We get this is because of the combination of step 3 and 4.

## Zero crossings of Laplacian of Gaussian filter

Let me just briefly mentioned that it's also possible to find edges that are local maxima. The gradient magnitude looking at places where the second order derivatives is 0 or at least crosses 0.

This can be done by computing the Laplacian of our image.

这是基于 canny 算法的改进（我们一开始上课的那个算法），看三阶导小于 0。

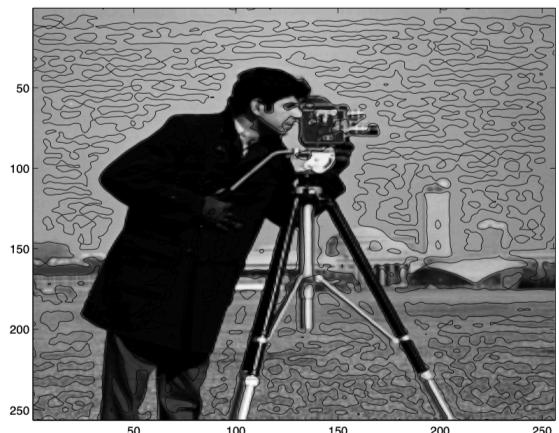
这种算法是基于Canny算法的改进，称为“二次微分边缘检测”（Second Derivative Edge Detection）。

在这种算法中，我们先对图像应用高斯滤波进行平滑，然后计算二阶导数（也称为拉普拉斯算子），即找到图像中的所有高斯平滑的极值点。这些极值点可能对应于图像中的边缘。但是，这种方法往往会导致检测到大量噪声和边缘。

为了减少噪声和增加边缘的连续性，该算法还要求在每个极值点处计算三阶导数（也称为峭度）。如果峭度小于某个阈值，则该点被认为是噪声并被抑制。如果峭度大于阈值，则该点被认为是边缘，并被保留。

尽管这种算法可以在一定程度上提高边缘检测的质量，但它通常无法检测到较弱的边缘，并且会在边缘处产生不必要的噪声。与Canny算法相比，它也缺少双阈值和非极大值抑制等重要步骤。

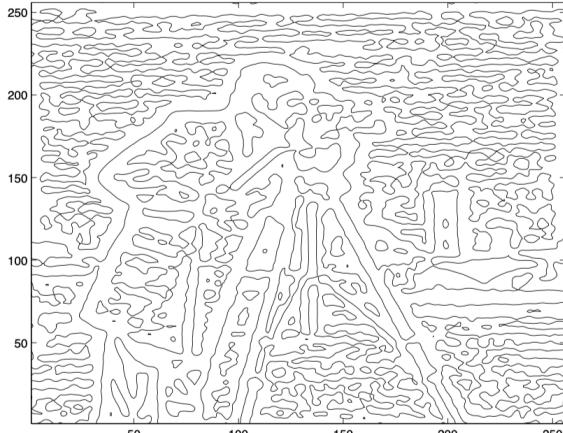
### **How to implement this?**



$$\nabla^2 L(x, y; \sigma) = L_{xx} + L_{yy}$$

**Edge points are defined as locations where**

$$\nabla^2 L(x, y; \sigma) = 0$$



It's basically called Laplacian of Gaussian filter and that we need to compute the twice derivative in the x direction and the twice derivative in the y direction, for some specific scale and then adding these together and we have a feature image a filter response. Then, we go through all pixels and try to find locations where is equal to 0.

但是如果你只看哪边等于0是远远不够的，在现实中，你还得去看符号变化。 $\Rightarrow$  拉普拉斯算子在图像边缘检测中被用来检测二阶导数是否为0。如果二阶导数为0，表示在该点附近，图像灰度变化趋势发生了变化，因此可能是边缘点。但是，如果拉普拉斯值为0的点周围的灰度变化方向不一致，那么这个点并不能算作一个真正的边缘点。因此，需要通过判断周围的灰度变化方向来确定这个点是否为边缘点。如果该点周围灰度变化方向一致，则认为该点是一个真正的边缘点。

灰度变化方向一致指的是一组像素的灰度值变化在相邻像素之间具有相同的方向。在边缘检测中，我们希望找到的边缘是由于灰度值的突然变化所引起的，而这种变化通常会伴随着一个明显的变化方向。因此，通过检测像素灰度值变化方向一致的区域，我们可以识别出图像中的边缘。

---

现在我们已经有了三种边缘检测方法，第一种是普通的，通过计算梯度的和的大小，每个像素的局部最大值( $|\nabla L|$ )<sup>2</sup>并直接用threshold进行比较。第二种是利用canny算法，达到近乎完美的边缘检测。第三种是通过利用拉普拉斯算子 $\nabla^2 L$ 来求二阶导然后再看方向。

---

## Local gauge coordinate system: Enforcing rotation invariance

Both gradient magnitude and Laplacian operator are invariant to rotation and translation. Now there is a general way that we can enforce this form of invariants and there is by using something called **local gauge coordinate system**.

And this is particularly relevant to do because we would like that our detectors can find the same points irrespectively of we rotate an object out in the world out in the scene or we rotate the image or move the camera a little bit about like translated, we would like to still be able to detect the same point.

在图像处理中，Gauge 坐标系通常用于描述局部图像结构的方向性和形状。它是一种基于局部坐标系的数学表达方式，用于描述给定点周围的局部图像结构。

具体来说，Gauge 坐标系是一种通过在每个图像位置引入局部坐标系来描述局部结构的方法。在每个点处，局部坐标系的方向被定义为图像梯度的方向，坐标系的原点被定义为该点本身。这种方法使得我们可以描述出在图像中每个点周围的方向性信息，从而提取出图像中的一些局部特征。

在实际应用中，Gauge 坐标系可以被用于计算一些图像处理中的特征，比如边缘方向、纹理方向和局部形状等。

For convenience, adapt a **local** coordinate system  $(u, v)$  in place of the global  $(x, y)$  coordinate system.

This is called **gauge coordinates** (or a Frenet frame)

Aligned according to the local geometry (e.g., gradient vector or eigenvectors of the Hessian matrix)

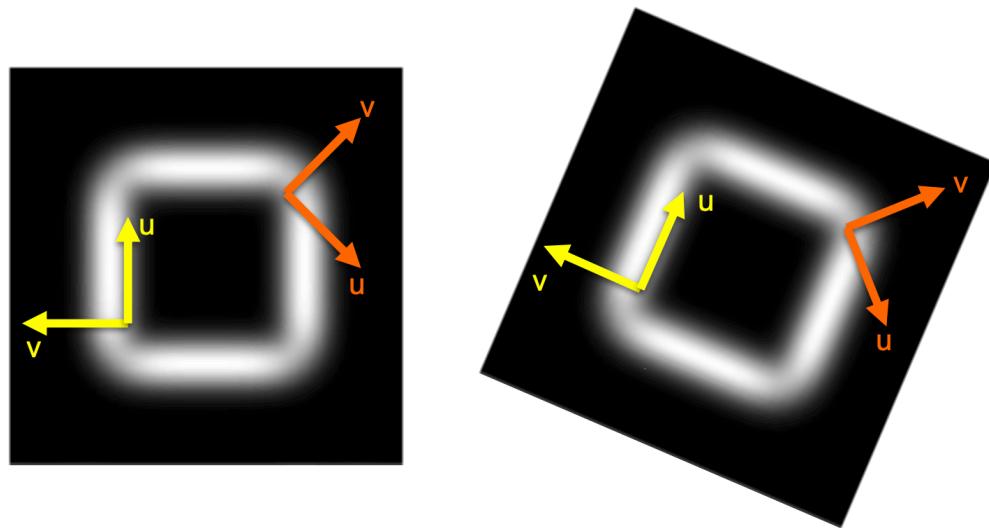
Hence, the gauge coordinates **rotate together with the image** and are **independent of the image coordinate system**.

*So, let's look at a concrete example.*

Let's choose a coordinate system which is based on the gradient. => **Gradient-based gauge coordinates**

At any point in an image,  $v$  is parallel to the local **gradient**.  $u$  and  $v$  are orthogonal. So that it actually goes along the **intensities**.

So here you see two examples two different locations in the image and each of the locations, we have a local coordinate system, so we have different coordinate systems here.



Rotation of the image does not change the  $(u, v)$  coordinates.

*If you do this, we can now define various forms of derivatives and edge detectors.*

## Directional derivatives

We can compute directional derivatives in any direction  $v$  from the  $x, y$  derivatives.

Consider the directional vector parameterized by an angle  $\theta$ ,  $v = (\cos \theta, \sin \theta)^T$ .  $\theta$  could be the direction of a local gradient, or gradient orientation or it could be other things.

$\nabla L := \text{gradient vector}$

First and second derivatives along  $v$  can be computed by

$$L_v = v^T \nabla L = L_x \cos \theta + L_y \sin \theta \implies \text{First derivative in direction } v$$

$$L_{vv} = v^T \nabla L_v = v^T \mathcal{H}v = L_{xx} \cos^2 \theta + 2L_{xy} \cos \theta \sin \theta + L_{yy} \sin^2 \theta$$

This means, right now, I can take the yellow coordinate system, I can compute derivatives along the  $v$  direction and along the  $u$  direction.

## Gradient magnitude is rotational invariant

For the gradient-based gauge coordinates the  $v$  direction is

$$v = (L_x, L_y)^T / \sqrt{L_x^2 + L_y^2}$$

Computing the first derivative in the  $v$  direction gives us,

$$L_v = v^T \nabla L = \sqrt{L_x^2 + L_y^2}$$

Hence, if you rotate the image, then gradient magnitude is unaffected - it is invariant with respect to rotation of the image. That is, if you consider the same image point as it rotates along with the image.

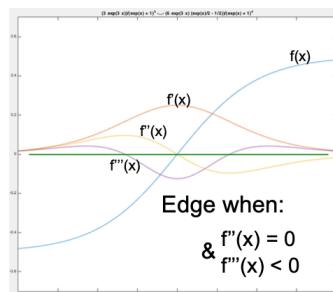
Also  $u = (-L_y, L_x)^T$  and  $L_u = u^T \nabla L = -L_y L_x + L_x L_y = 0$ .

## Edge detection (Lindeberg's approach)

If we denote the gradient direction by:

$$v = 1 / \sqrt{L_x^2 + L_y^2} (L_x, L_y)^T$$

Then, edges can be determined as those image points that satisfy the following two conditions (maxima of the gradient magnitude in the gradient direction):



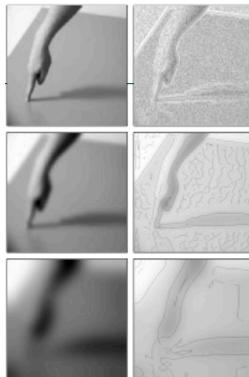
$$L_v = v^T \nabla L = \sqrt{L_x^2 + L_y^2}$$

$$L_{vv} = L_x^2 L_{xx} + 2L_x L_y L_{xy} + L_y^2 L_{yy} = 0$$

$$L_{vvv} = L_x^3 L_{xxx} + 3L_x^2 L_y L_{xxy} + 3L_x L_y^2 L_{xyy} + L_y^3 L_{yyy} < 0$$

Interpolating for zero-crossings of  $L_{vv}$  within the sign-constraints of  $L_{vvv}$  gives a straightforward method for sub-pixel precision edge detection.

If you do this, you can get results that looks like this.



This is actually showing this detection algorithm and different scales.

## Sparse interest point detectors

We are now turn attention to sparse interest points.

## Application

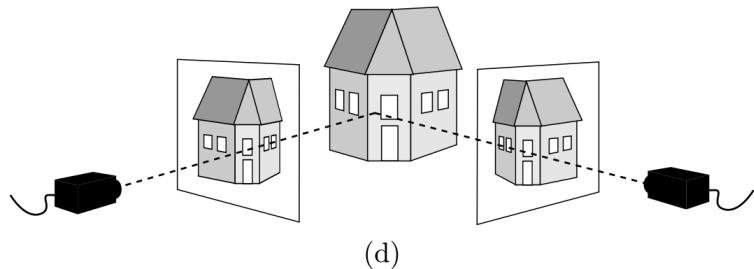
**3D reconstruction Stereo, multi-view, and structure from motion**



(a)

(b)

(c)



(d)

If we want to do 3D reconstruction of a scene from multiple view of the scene to multiple images of the scene. Then you need to solve the matching problem.

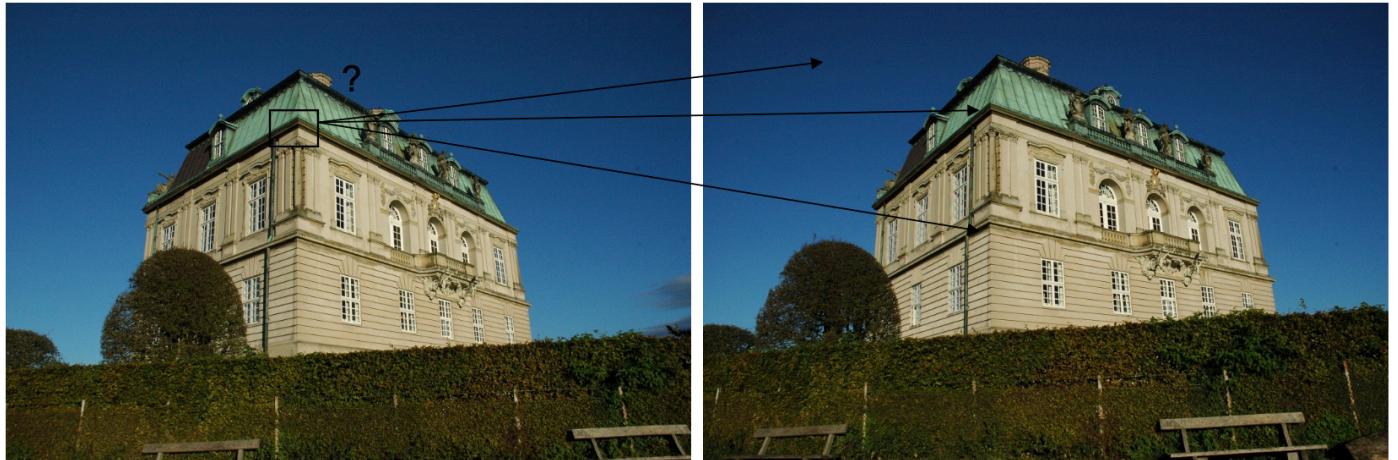
In the below illustration, we have two views of this house and in order to do this 3D reconstruction then basically we need to be able to find the same point out in the scene in both of these images. For instance, if we are looking at some part where at this window here, and we need to be able to recognize that we see this scene point here in this image right here. And in the other image, we see the same point right here. And we need this in order to be able to do triangulation and basically infer depth and 3D structure.

The point here is that we need to be able to find some points that represent the same physical point out in the world out in the scene. And we need to be able to find it again in multiple images of the same scene.

## Matching image neighborhoods for 3D vision

### *Where did the underlying feature go?*

In order to do this, we need to think a little bit about what is required for that we can carry out that this recognition or matching of points.



Now, if I pick a point, let's say we look at this corner here on the house. Then we ask ourselves in this other image (view get slightly changed to the right). The question is where did this point go in this image here?

We can easily see that it's probably right here, especially if we consider the corner of the roof, but we need some way to automate this process to say the other points are not the answer as they are not in the roof.

***Matching: Ideally, we want to find the location in the other image which corresponds to the same physical location on the building.***

It is important to realize that it's not possible to match any random point or patch I would extract from this image here. I cannot hope to match all of them into this new image.

For instance, if I picked a point in the sky. I would have a hard time saying, if I consider this point right here. Where did that come from? Is it maybe here or it could also be here. the only thing we can do is sort of look at the surrounding and say, does it resemble a point that we are looking at here. And basically we have no way of uniquely identifying the same point in the other image.

The point here is that, it is not all points that we can hope to actually match up. It has to do with the structure around the point.

To be able to figure out which points belongs to which we need a bit of structure:

Sky patches are impossible to match

Curves are also problematic => roof

## Salient points



This is an example of the type of points we would like. This is created with something on the multi-scale Harris corner detector.

Multi-scale Harris corner detector with localization: Approximately 600 points in each image.

There are points we do not find in both images, but there are many for which we do.

## Salient points a.k.a. Interest points

Salient points: Local structure in the image that appear distinct from the surrounding region of the salient point.

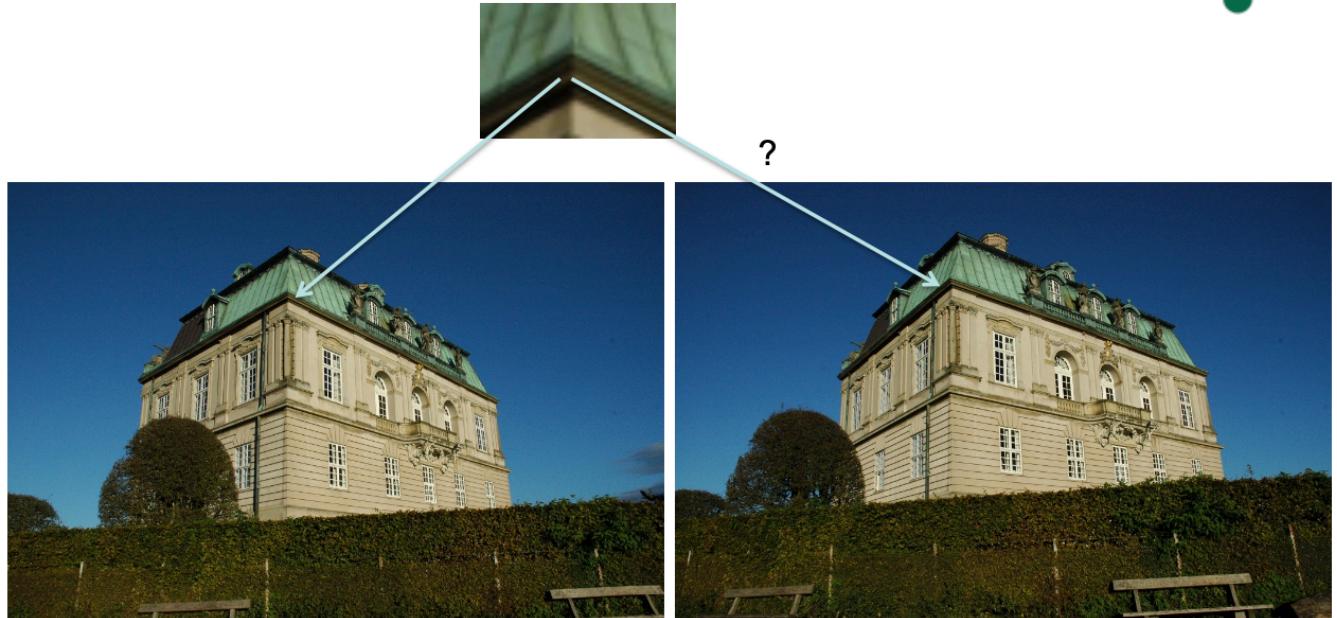
We need an operational definition in order to detect such points!

Terminology: Salient points aka interest points aka keypoints aka features (the literature is full of confusing inconsistent terminology)

## How to automatically detect salient points?

### Template matching - correlation

## Template



But how do we construct a generic corner template? It's actually a bad idea => won't work

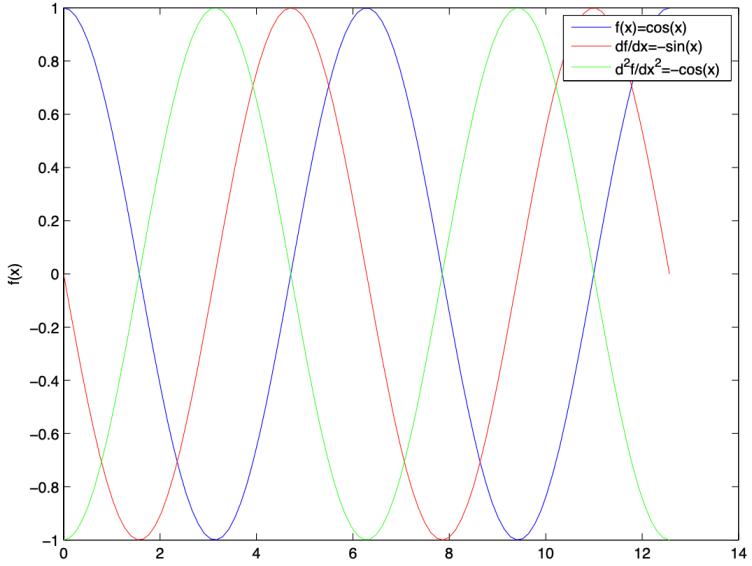
## Intensity blobs as salient points

The first algorithm we would look at is intensity blobs.

- Local intensity extrema (maxima and minima) are potential candidates for salient points.
- Extrema are distinct from their neighboring pixels.
- We refer to these extrema as bright or dark blobs.
- How do we find intensity extrema?

Compute image derivatives

## Blob detector: A simple 1D example



Finding extrema - look for points where  $\frac{\partial f(x)}{\partial x} = 0$ ,  $\frac{\partial^2 f(x)}{\partial x^2} \neq 0$

一阶导为 0 说明该位置可能为极值或者原函数为直线，即无大小变化。若加上二阶导不为0则可确定该位置为原函数极值点。

## Detecting blobs by Laplacian of Gaussian filter

Finding extrema in 2D images:

- Find points that are simultaneous intensity extrema in the  $x$  and  $y$  direction.
- We can solve this in one go by looking for extrema of the Laplacian of Gaussian filter  $\nabla^2 L(x, y; \sigma) = L_{x^2}(x, y; \sigma) + L_{y^2}(x, y; \sigma)$ .  $\Rightarrow$  Compute the Laplacian image

$$||\nabla(\nabla^2 L(x, y; \sigma))|| = 0 \Rightarrow \text{twice derivative}$$

- Bright blob:  $\nabla^2 L(x, y; \sigma) < 0$ ; Dark blob:  $\nabla^2 L(x, y; \sigma) > 0$

Discrete implementation: Extrema search in 2D

- Bright blob at  $(x, y)$ :  $\nabla^2 L(x, y; \sigma) <$  than all neighbor pixels
- Dark blob at  $(x, y)$  :  $\nabla^2 L(x, y; \sigma) >$  than all neighbor pixels
- choose 4-neighbors or 8-neighbors
- Keep only blobs where  $|\nabla^2 L(x, y; \sigma)| >$  some threshold value  $\Rightarrow$  a way of filtering out potential noise from the image.

If we do this, we get something below.

## Detecting blobs

**Yellow = dark blobs, Red = bright blobs**



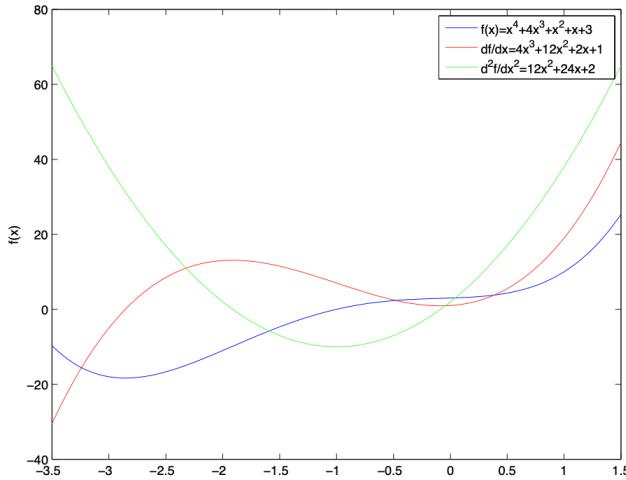
Problem: It also detects points on edges and other non-blob structure

使用拉普拉斯高斯滤波器检测blob的过程中，可能会检测到一些不是blob的结构，例如边缘和其他非blob结构。这是由于拉普拉斯算子对于所有的图像结构都敏感，因此会在边缘和其他结构上产生响应。这些响应通常会被认为是误检测，可以通过应用阈值和其他后处理技术来减少误检测的数量。此外，如果需要检测的是特定形状和大小的blob，则可以使用不同大小的高斯核来生成拉普拉斯高斯滤波器，以便在不同的尺度下检测blob。

边缘和blob是两种不同的结构，在图像中有着不同的表现形式。边缘通常是由图像中的强度变化或者颜色变化引起的，而blob则是由图像中的区域内部的强度变化引起的，它通常具有一个局部的峰值。在Laplacian of Gaussian (LoG) 方法中，通过使用高斯平滑和拉普拉斯算子来检测blob，而边缘则往往会被忽略掉。这是因为边缘上的强度变化通常比较陡峭，因此其二阶导数（拉普拉斯）会出现跳跃，不满足高斯函数的平滑性质，所以LoG滤波器很难在边缘上检测出blob。

## Pitfalls - Laplacian of Gaussian filter

***Not all detected blobs are intensity extrema***



在二阶导数为极值的情况下，我们能够辨认出原函数不是对应的局部最大值。

So the particular way of the defining blobs does not necessarily give you the perfect points for matching.

If we were to do matching, we would probably need additional algorithms to identify all these points that are actually edge points and remove them because we cannot hope to probably match them in another image.

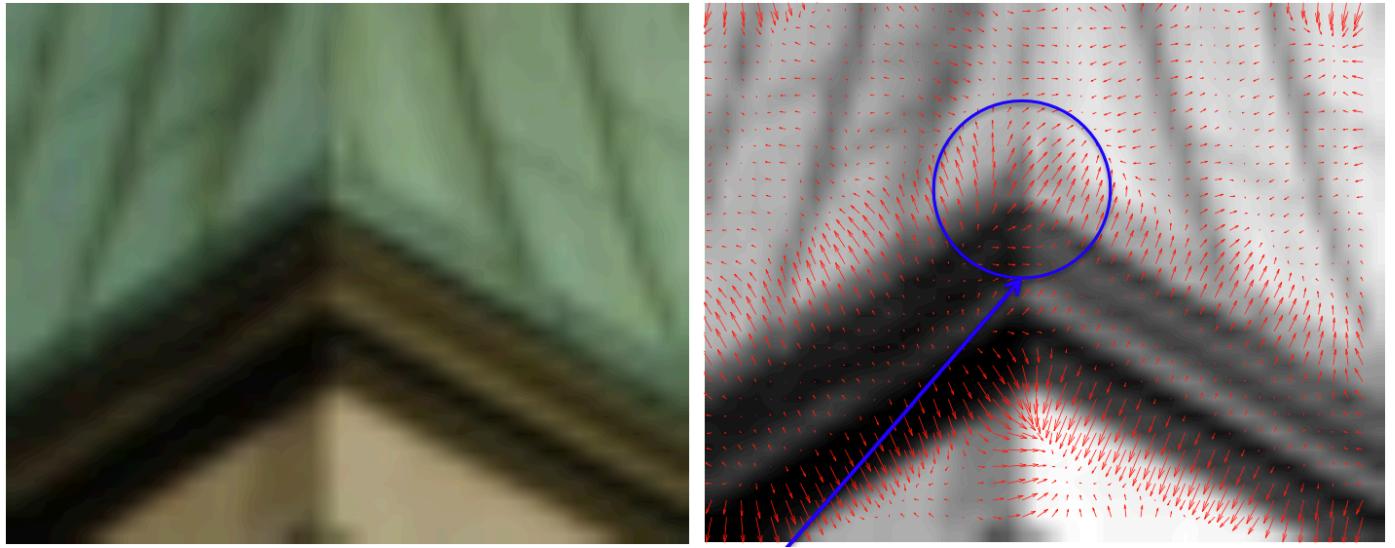
But some points are extremely very good for instance, window and roof.

## Intensity Corners

The other type of structure that we would like to detect or try to detect is intensity corners. So, what is meant by that?

$$[L_x, L_y](x, y; \sigma)$$

Here we have a cut out of the corner of the roof. And we would like to find locations like here. One way to do this is to look at the gradient vector field. So, here on top of a grayscale version of the image here. I'm showing each of the gradient vectors corresponding to the pixel that at the end of the arrow here.



你看这些点都往一个点指，被指的那个点梯度长度为0，根据定义，他就是bright blob。

But now we are interested in corners but we can still use gradients to get an idea of where the corner and the key idea with the so-called Harris corner detector is,

*Look for locations with two dominating gradient directions*

区域有两个不同的方向的梯度=>很有可能是 corner

## Intensity corners as salient points

### **Detecting corners with Harris corner detector**

Harris corners are defined as points of local maxima of the Harris corner measure:

Intensity corner（强度角点）和Intensity blob（强度斑点）是计算机视觉中用于描述图像特征的两种不同的概念。强度角点指的是图像中局部区域内强度变化最为明显的点，通常被认为是图像中的角点。而强度斑点则是指图像中具有明显强度峰值的局部区域，通常被认为是图像中的斑点。这两种特征在图像处理和计算机视觉中具有广泛的应用，例如在图像拼接、目标跟踪和物体识别等领域中被广泛使用。

## Structure tensor

结构张量是一种用于计算图像局部区域内灰度值变化特征的数学工具。在Harris角点检测算法中，结构张量用于描述图像中某个像素点周围的灰度变化情况，从而确定该像素点是否为角点。结构张量可以表示为一个矩阵，其中每个元素表示该像素点周围某个方向上的灰度变化程度。在Harris角点检测算法中，通过计算结构张量的特征值和响应函数来判断该像素点是否为角点。

The structure tensor is the local covariance of the gradient vector field  $\nabla L(x, y; \sigma) = (L_x, L_y)^T$  estimated under the Gaussian window  $G(x, y; k\sigma)$ ,

$$A(x, y; \sigma) = G(x, y; k\sigma) * (\nabla L \nabla L^T)$$

We look for locations  $(x, y)$  where the two eigenvalues of  $A$  is both large (the sign does not matter). This corresponds to two dominating directions in the vector field.

If one eigenvalue is large and the other small, then we are at an edge or ridge. If both are small we are at a flat region or on top of an extrema.

Using  $R(x, y; \sigma)$  is a smart way to avoid having to compute the eigenvalues.

$$R(x, y; \sigma) = \det(A) - \alpha \operatorname{trace}(A)^2$$

这里是在使用 Harris 角点检测算法，通过计算图像中每个像素点周围像素的梯度方向和梯度强度，构造出该像素点周围的结构张量  $A(x, y; \sigma)$ 。结构张量描述了该像素点周围的梯度分布情况。

通过计算结构张量的两个特征值，可以确定该像素点周围的梯度分布情况，从而判断该点是否为角点、边缘或平坦区域。如果该点周围的梯度分布情况表现为两个方向上都有较强的梯度变化，则说明该点为角点；如果只有一个方向上有较强的梯度变化，则说明该点为边缘；如果两个方向上都没有很强的梯度变化，则说明该点处于平坦区域或者是图像的局部极值点。

为了避免计算结构张量的两个特征值，Harris 角点检测算法使用了响应函数  $R(x, y; \sigma)$ ，其中响应函数的值被定义为结构张量的特征值之积减去一个经验常数  $\alpha$  与结构张量特征值之和的平方。通过判断响应函数的大小，可以判断该点是否为角点。

If you use this, then you get,

# Interest point detectors: Detecting Harris corners (fixed scale)



*Dictator: Ying Liu*  
*Date: March 15, 2023*