Edge Detection

Slides Credit : James Tompkin

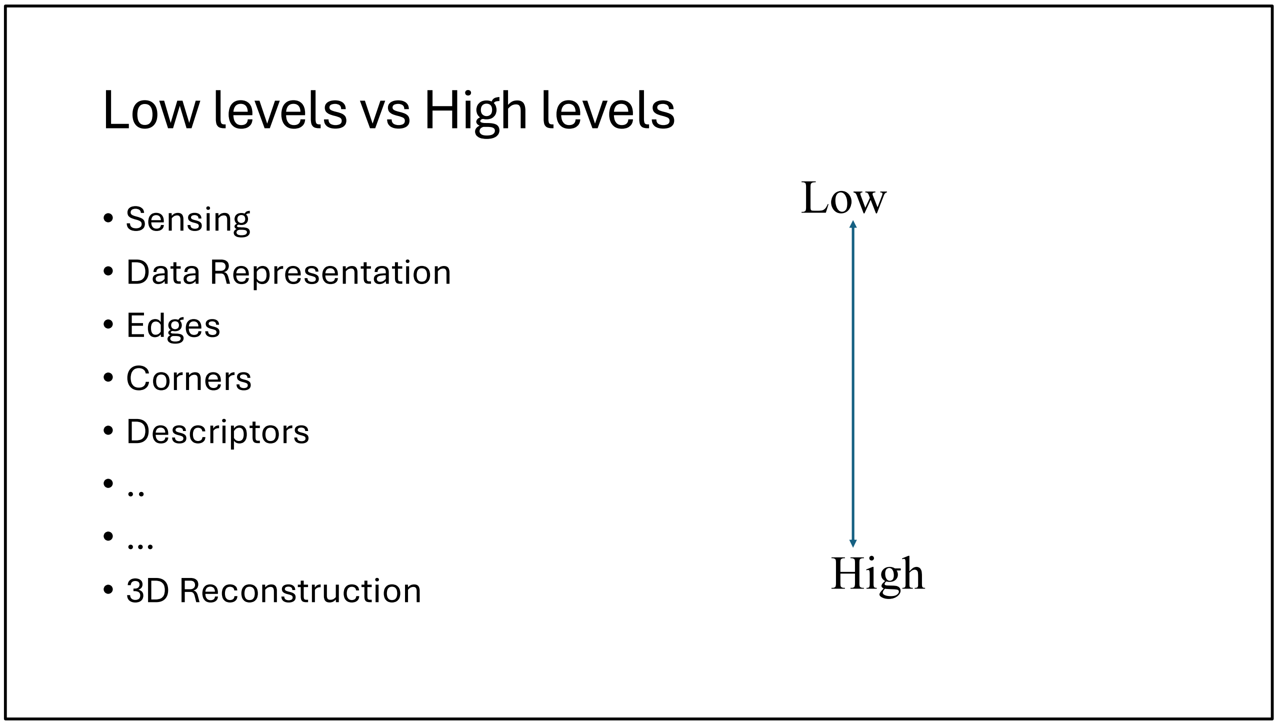
In this lecture we are going to learn about edge detection and how edge detection is important for feature matching and feature extraction for higher-level computer vision tasks.

Edges

- What types of edges do exist?
- Edges in noisy images
- Canny edge detector

in edge detection what we are going to cover is
what type of edges exist
what kind of influence does the noise have from the image over the edges

We are going to learn about a specific edge detection method called the **Canny Edge Detector**. It is one of the most popular techniques in computer vision. In fact, if someone has taken a computer vision course and hasn't learned about the Canny Edge Detector, it's often assumed they missed an essential topic. In this lesson, we'll take a procedural look at how the Canny Edge Detector works. When it was first introduced, it represented the state of the art in edge detection methods.



Low levels vs High levels

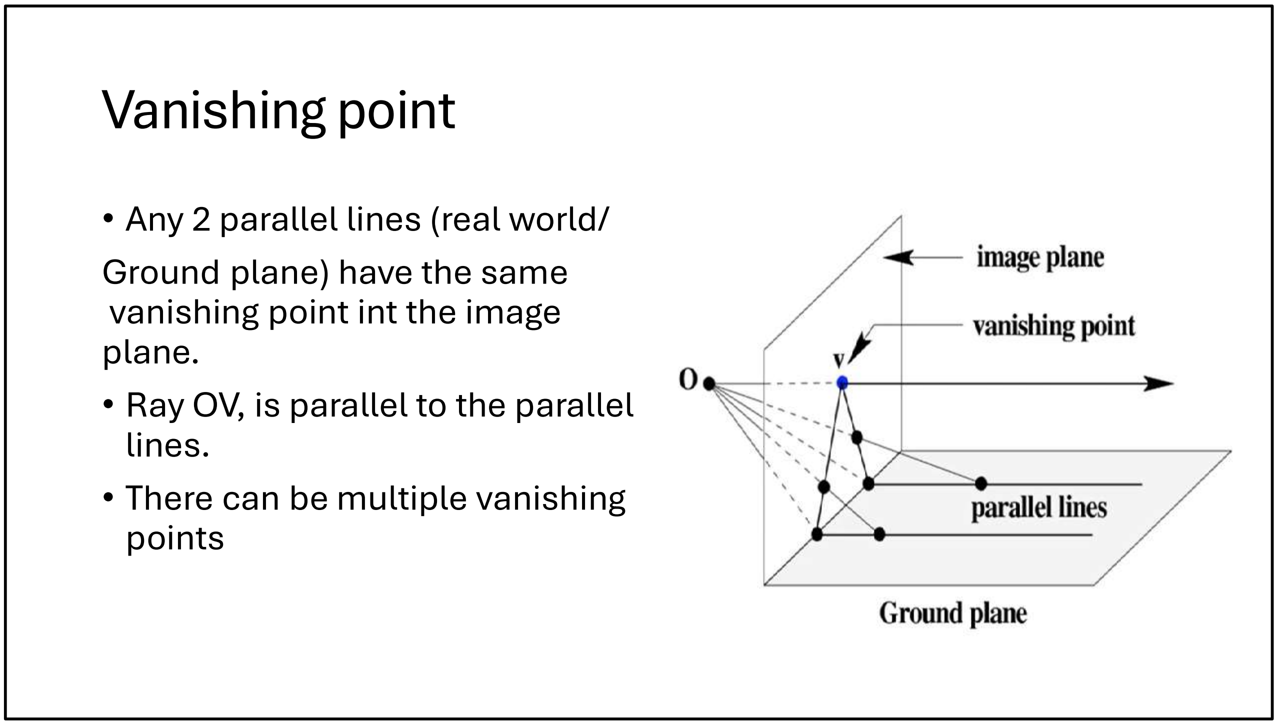
- Sensing
- Data Representation
- Edges
- Corners
- Descriptors
- ..
- ...
- 3D Reconstruction

Before diving into edge detection, let's first clarify what we mean by **low-level** and **high-level** tasks in computer vision.

In computer vision, computation focuses on understanding real-world images to extract information relevant to a specific application. This information could include **edges**, **corners**, or other **features**. The tools and techniques used to perform these operations form the foundation of computer vision.

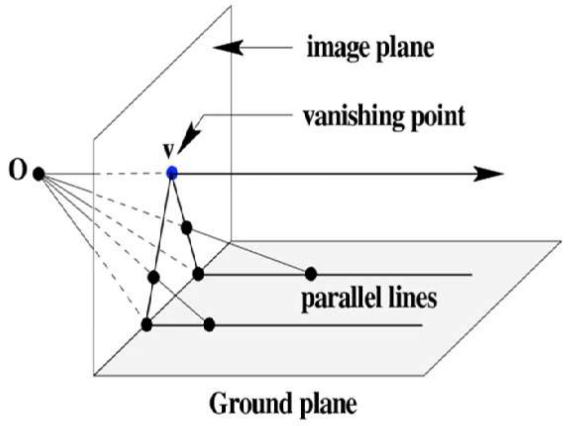
A typical computer vision pipeline includes multiple stages — from **image sensing** and **data representation** to **feature extraction** and **3D reconstruction**. Each stage involves distinct methods and algorithms, many of which are still evolving with new state-of-the-art techniques.

As we move along this pipeline, the nature of the tasks transitions from **low-level** to **high-level**. Low-level tasks, such as detecting edges or corners, deal with local features and simple computations. In contrast, high-level tasks, such as 3D reconstruction, involve more complex reasoning — for example, reconstructing a three-dimensional view of the real world from multiple images.



Vanishing point

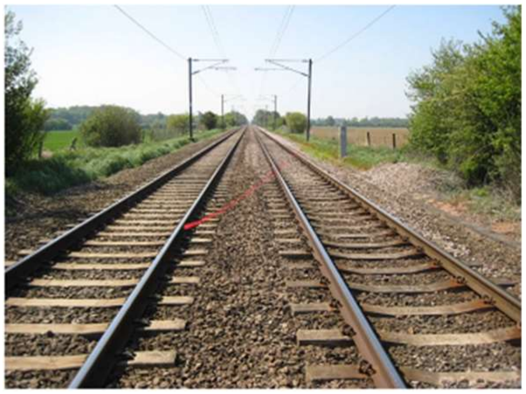
- Any 2 parallel lines (real world/ Ground plane) have the same vanishing point int the image plane.
- Ray OV, is parallel to the parallel lines.
- There can be multiple vanishing points



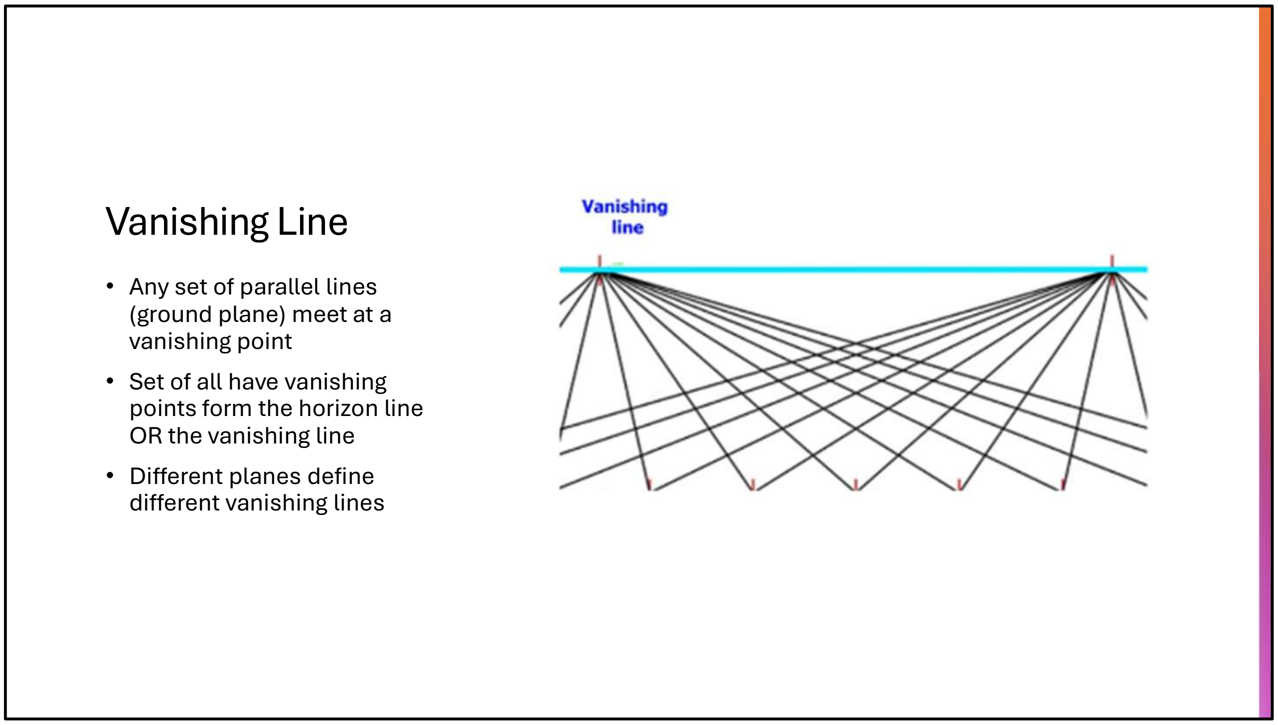
Before we examine how edges behave and what defines them, let's first learn about **vanishing points** and **vanishing lines**.
Consider a **ground plane** — any flat surface in the real world. When you capture an image of this plane, the **image plane** represents the projection of the real-world scene, and **O** denotes the **camera center**.
Lines that are **parallel** in the ground plane appear to **converge** at a single point in the image plane. This point is called the **vanishing point**. Any two parallel lines along the same plane will share a single vanishing point — that's one of its key properties.
Another important property is that the **ray** originating from the camera center and passing through the vanishing point is **parallel** to the direction of those parallel lines on the ground plane.
If you change the orientation of the ground plane, you will observe **different vanishing points**. Therefore, an image can contain **multiple vanishing points**, each corresponding to a distinct set of parallel lines in different directions.

Vanishing point

- Any 2 parallel lines (real world/ Ground plane) have the same vanishing point int the image plane.
- Ray OV, is parallel to the parallel lines.
- There can be multiple vanishing points

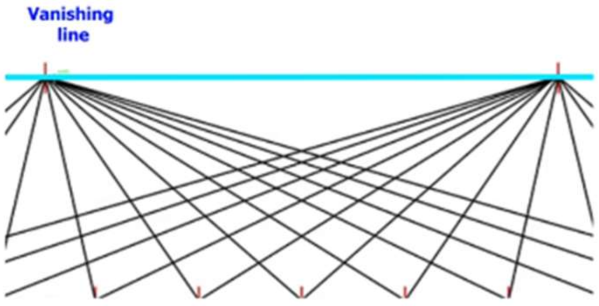


This image is an example of a **vanishing point**. If we consider the ground as our plane, the **railway tracks** serve as **parallel lines** on that plane. In the image, these tracks appear to **converge at a single point** — this point is the **vanishing point**.
A vanishing point is useful not only for understanding scene geometry but also for practical applications such as **camera calibration** and **estimating camera parameters**. It helps in reconstructing the camera's orientation and understanding its projection characteristics.



Vanishing Line

- Any set of parallel lines (ground plane) meet at a vanishing point
- Set of all have vanishing points form the horizon line OR the vanishing line
- Different planes define different vanishing lines



Another important concept is the **vanishing line**.

Vanishing lines are related to the **sets of parallel lines** that exist on a plane in the real world. For example, on a ground plane, you might have several groups of parallel lines — each group converges at its own **vanishing point**. The **line connecting all these vanishing points** is called the **vanishing line** (or **horizon line**).

To illustrate, imagine two sets of parallel lines on the same ground plane — one set running horizontally and another diagonally. Each set will meet at a different vanishing point, and the line connecting those points forms the **horizon line**.

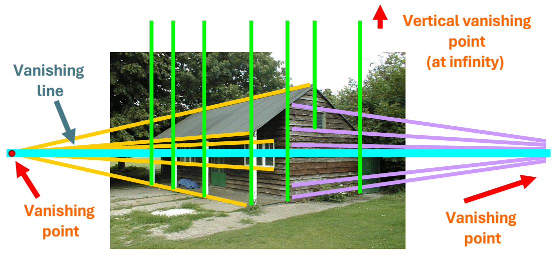
If the ground plane shifts upward or downward, a **different plane** is formed, resulting in a **new vanishing line**. Understanding vanishing lines is essential, as we'll see in the upcoming slides.

Use of Edges

- Extract information
  - Recognize objects
  - Reconstruct scenes

- Help recover geometry and viewpoint

- Edit images directly
  - Artistic effects

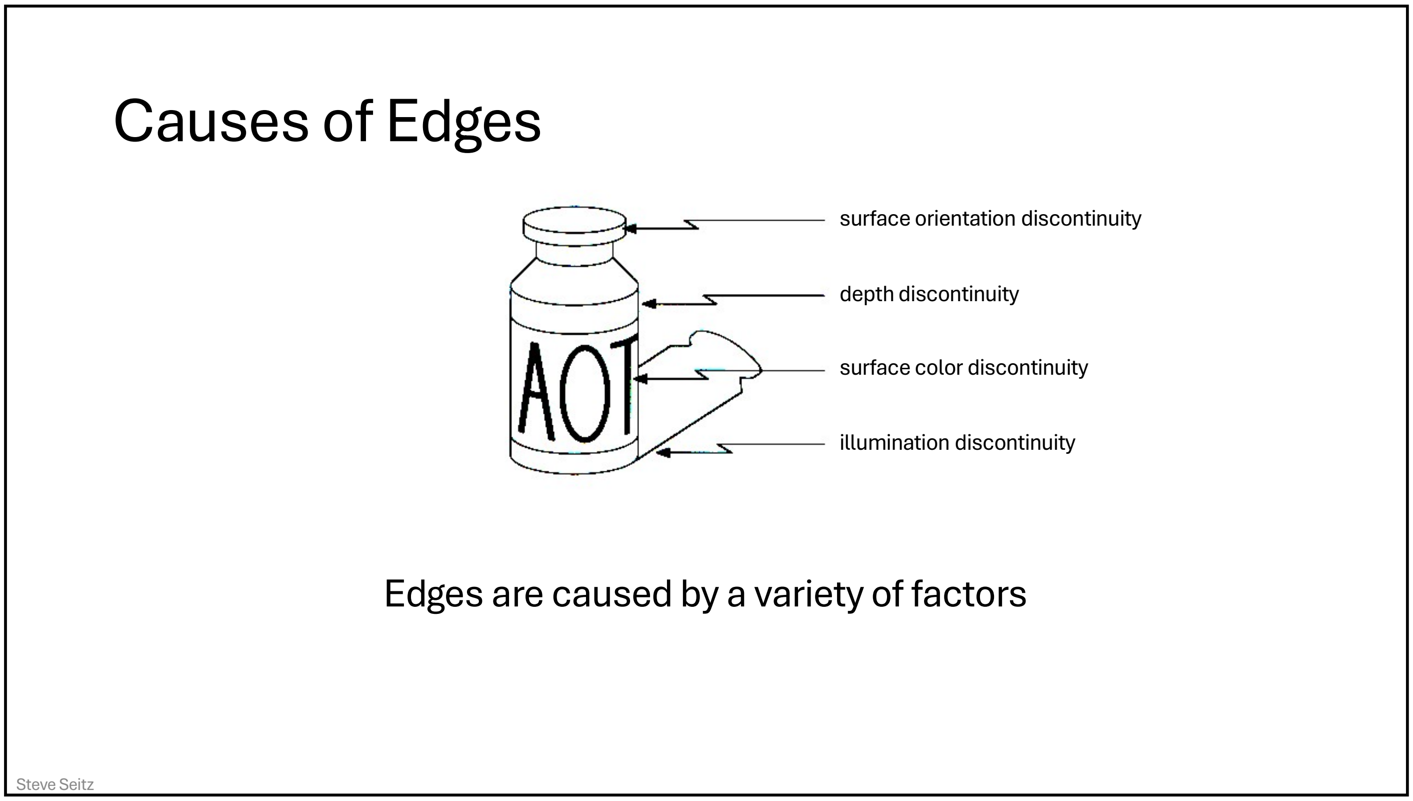


Let's take a step back and think about the **goal of edge detection**.

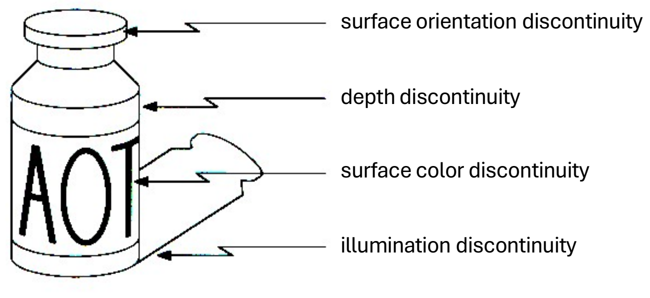
Edge detection aims to **identify edges** within an image — that is, regions where there are **significant changes in pixel intensity**. These abrupt variations indicate **discontinuities** in the image, which often correspond to object boundaries or structural changes in the scene.

Why are edges important? By extracting edges, we can infer **viewpoints**, **recover geometric information**, and even **reconstruct the 3D structure** of the real world. Edge detection also serves as a foundation for many **higher-level computer vision tasks**, such as **object recognition**, **image stitching**, and **3D reconstruction**.

This image shows an example with **multiple vanishing points**. When you extend the lines present in the image, you'll notice one vanishing point here and another there. Connecting these vanishing points forms a **vanishing line** (or **horizon line**). Some sets of parallel lines, however, have their **vanishing point at infinity**.



Causes of Edges


surface orientation discontinuity
depth discontinuity
surface color discontinuity
illumination discontinuity

Edges are caused by a variety of factors

Steve Seitz

Surface orientation discontinuity: caused by physical edges in the 3D object in question
Depth discontinuity: caused when the foreground becomes the background (as viewed from a certain perspective)
Surface color discontinuity: caused by differences in colors between
Illumination discontinuity: caused by lighting/shadows
======================
Now, let's think about the **origins of edges** — what factors or features in an image cause edges to appear?

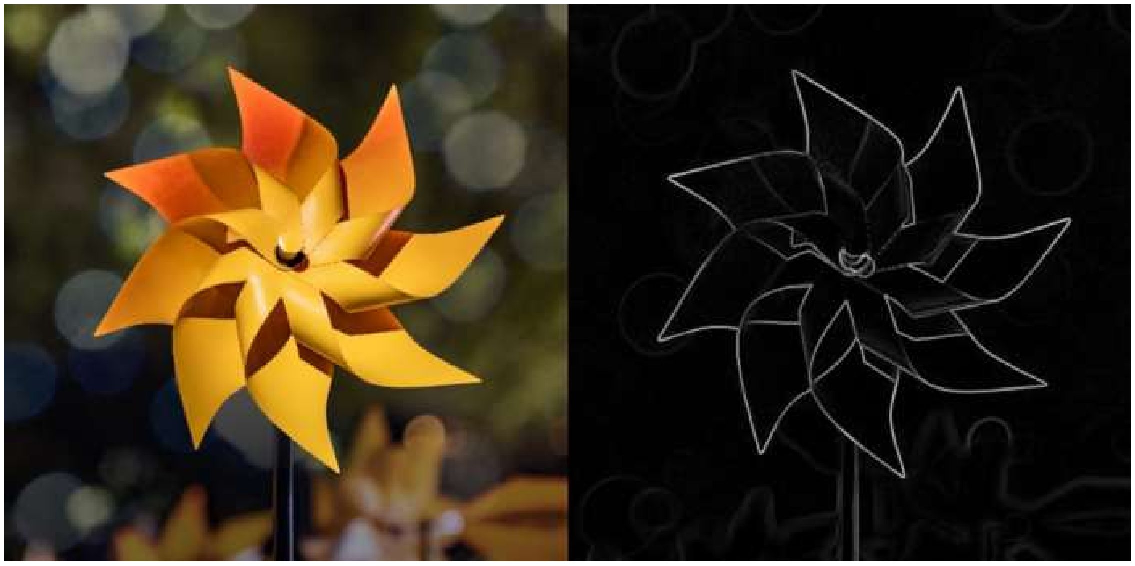
Edges can arise from several types of **discontinuities** in a scene:

- **Surface normal discontinuity** – This occurs when there is a sudden change in the surface orientation. For example, the edge between the cap and the body of a bottle forms because the surfaces face different directions. In a 2D image, we can't see the hidden parts, so this change in surface orientation appears as an edge.
- **Depth discontinuity** – Also known as **occlusion discontinuity**, this happens when one object **blocks** or **occludes** another. The boundary between the two objects creates a distinct edge.
- **Surface color or texture discontinuity** – When there's a sudden change in **color**, **pattern**, or **texture**, an edge appears. For instance, a textured surface next to a smooth or differently patterned area produces a visible boundary.
- **Illumination discontinuity** – Changes in **lighting** or **shadows** can also

create apparent edges, even when the physical surface itself is continuous.

These categories are not exhaustive, but they help us understand the main causes of edges in images. By knowing **why** edges occur, we can design better techniques to **detect and extract** them effectively.

Edge Detection



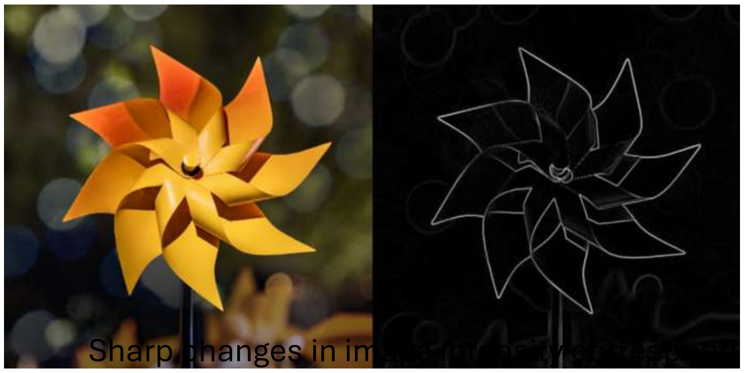
The white lines in the edge image on the right side correspond to edges in the original image

https://towardsdatascience.com/edge-detection-in-python-a3c263a13e03

Edge Detection

Definition

The process of identifying parts of a digital image with sharp changes (discontinuities) in image intensity.



Beginning to extract information.

Helpful to

- Recognize objects
- Reconstruct scenes
- Edit images (artistically)

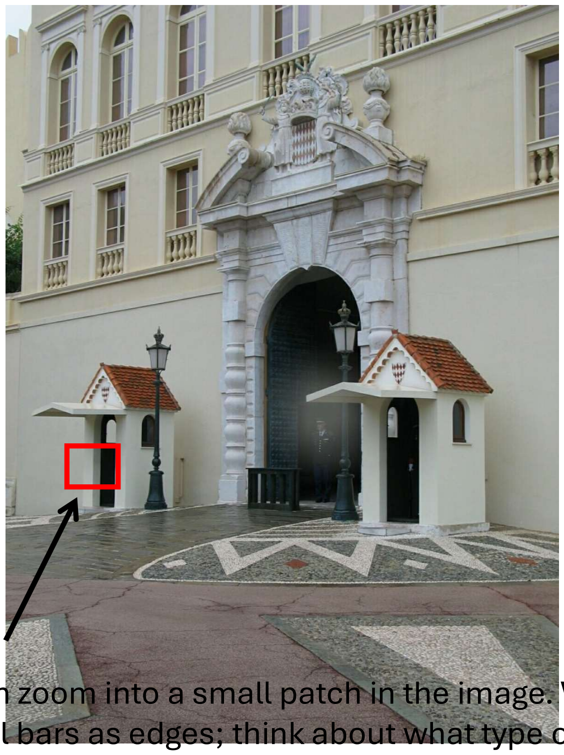
Sharp changes in image intensity correspond visually to regions where there is a sudden change in the color/brightness of an image (an edge!).
Edges are fundamental for recognition, reconstruction, and reorganization.

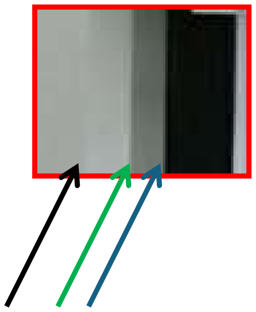
Closer Look at Edges

Let’s take a **closer look at edges** in this example image.
In the image, three different regions are highlighted.

Derek Hoiem

Closer Look at Edges





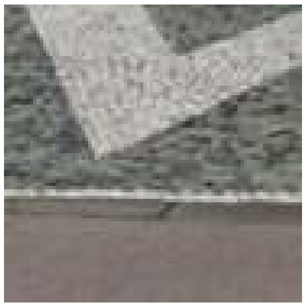
We can zoom into a small patch in the image. We interpret the changes between the vertical bars as edges; think about what type of discontinuity each is caused by.

Derek Hoiem

============

On the **leftmost** section, you can observe a **color discontinuity** near the ear — this represents a change in surface color.

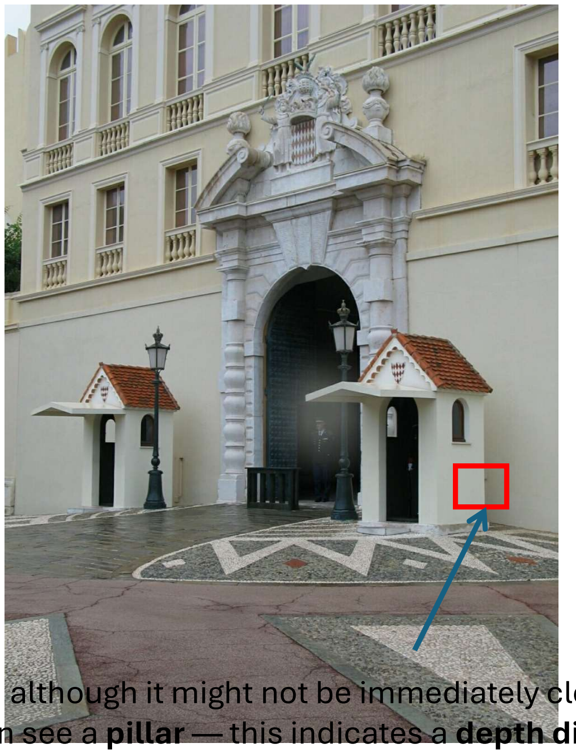
Closer Look at Edges

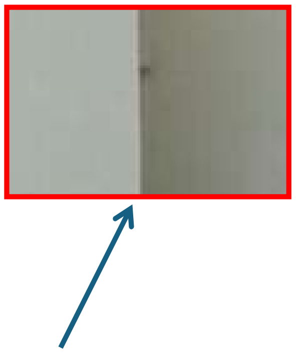


Next, there's a **surface discontinuity**, where the surface orientation shifts noticeably.

Derek Hoiem

Closer Look at Edges





Derek Hoiem

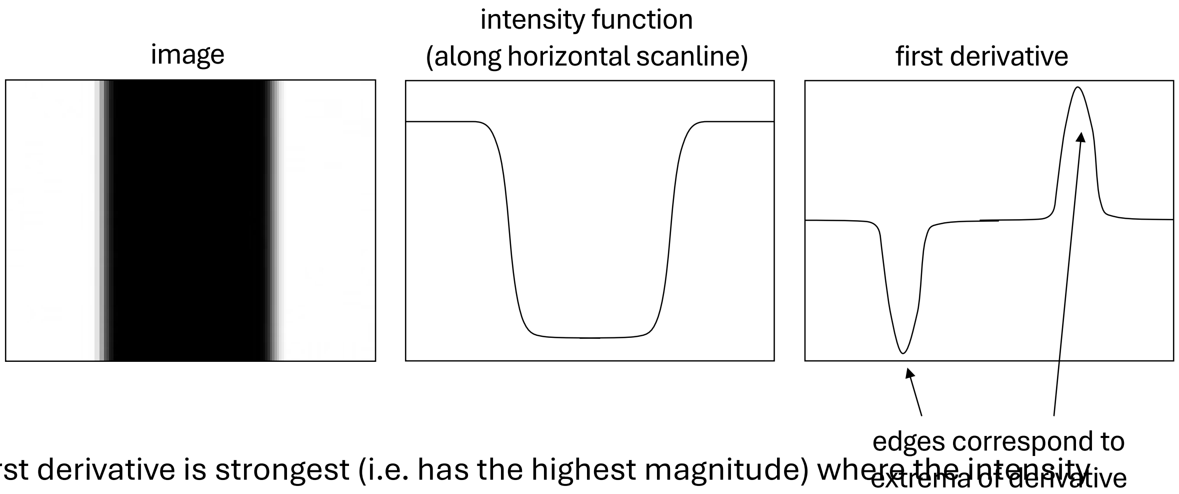
Finally, although it might not be immediately clear, when you look at the main image, you can see a **pillar** — this indicates a **depth discontinuity**, caused by a change in distance or occlusion.

When we apply an edge detector to this image, it might successfully capture edges where the **pixel intensity changes sharply**. However, even if some edges are not distinctly detected by an algorithm, the **human visual system** can still perceive them as meaningful boundaries.
Another example is the **texture discontinuity** near the ground, where variations in **color** and **pattern** create visible edges. On the **right-hand side**, we see yet another **depth discontinuity**, emphasizing how multiple types of edges — from **occlusion** to **geometry**, **contour**, and **texture** — can coexist within a single image.

Would you like me to make this version more **lecture-narrative**, for instance adding phrases like “as you can see here” or “notice how the textures change”?

Characterizing Edges

An edge is a place of sharp change (discontinuity) in the image intensity function. Simple algorithm for edge detection: find gradient.



James Hays

The first derivative is strongest (i.e. has the highest magnitude) where the intensity changes most rapidly. The sign of the derivative depends on whether the intensity falls from high to low or rises from low to high.

===============

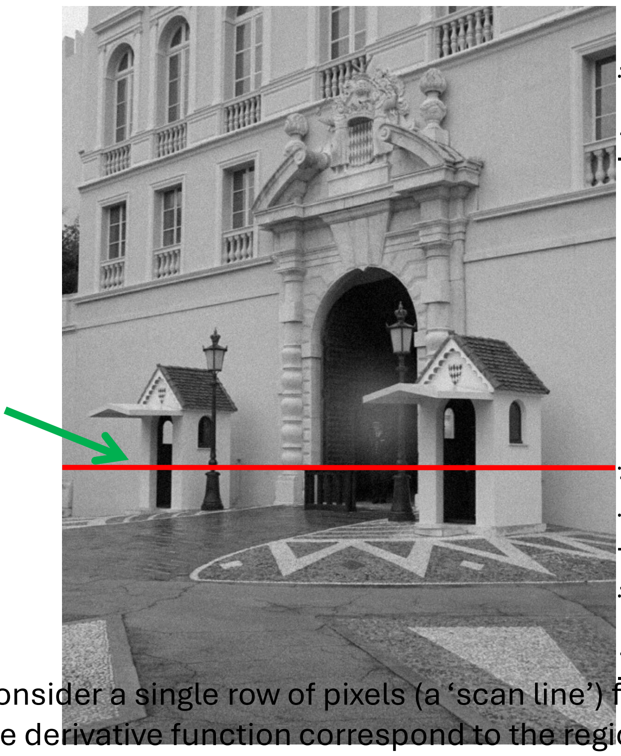
Now that we understand what **edges** are, let's discuss **how to characterize them** — in other words, how to describe or define edges mathematically so that we can detect them computationally.

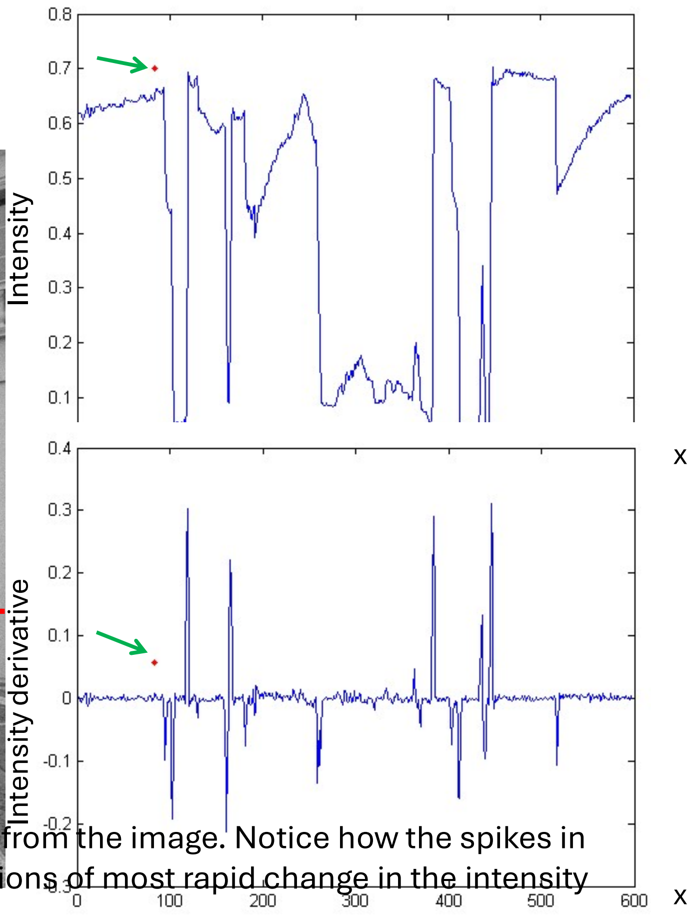
An **edge** is a region in an image where there is a **rapid change in pixel intensity or color values**. If we visualize these changes along a line in the image, we obtain what's called an **intensity function**.

For example, consider the image on the left. If we plot the intensity values along the red line, the resulting curve (shown in the middle) represents how pixel intensity varies across that line. From the image, we can visually identify two potential edges. However, looking only at the intensity curve, it's not always clear exactly where the edges occur.

To make this determination, we compute the **first derivative** of the intensity function. The **peaks** — both **positive and negative** — in the derivative correspond directly to the **locations of edges** in the image. This helps us mathematically define and detect where significant transitions occur in pixel values.

Intensity Profile





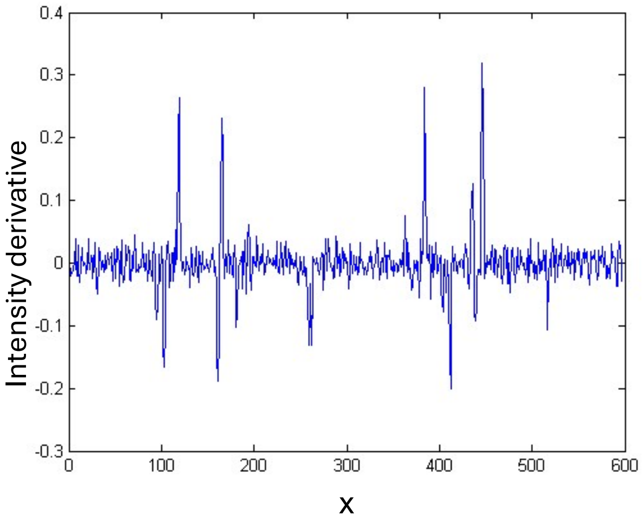
Derek Hoiem

Consider a single row of pixels (a 'scan line') from the image. Notice how the spikes in the derivative function correspond to the regions of most rapid change in the intensity function.
===========
Returning to our original example image, let's draw another line across it and plot the **intensity function** along that line.

In the resulting intensity plot, we can observe that the **intensity starts low** in the dark region, remains fairly constant for a while, and then **rises sharply** when the line passes into a bright (white) area. Later, as the line moves across a **shadowed region** or a **door**, the intensity **drops again**, creating noticeable variations.

This sequence of rises and falls represents how pixel brightness changes along that path. When we compute the **first derivative** of this intensity function, the **peaks** in the derivative clearly indicate the **locations of edges** in the image.

Adding Gaussian noise



But, again, real images are noisy, which results in lots of noise in the first derivative and starts to overwhelm the peaks.

Derek Hoiem

============

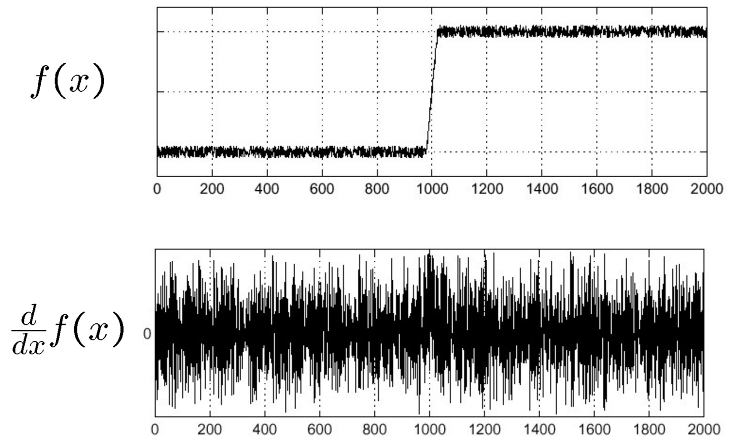
However, the **derivative operator** is highly sensitive to **noise**. When we add even a small amount of **Gaussian noise** to the original image, high-frequency variations appear as tiny dots or speckles across the image.

If we plot the **intensity profile** along the same line as before, the gradient is now **corrupted by noise**. While the original profile clearly indicated edges, the noisy profile becomes less distinct, making it harder to accurately detect the edges.

Effects of Noise

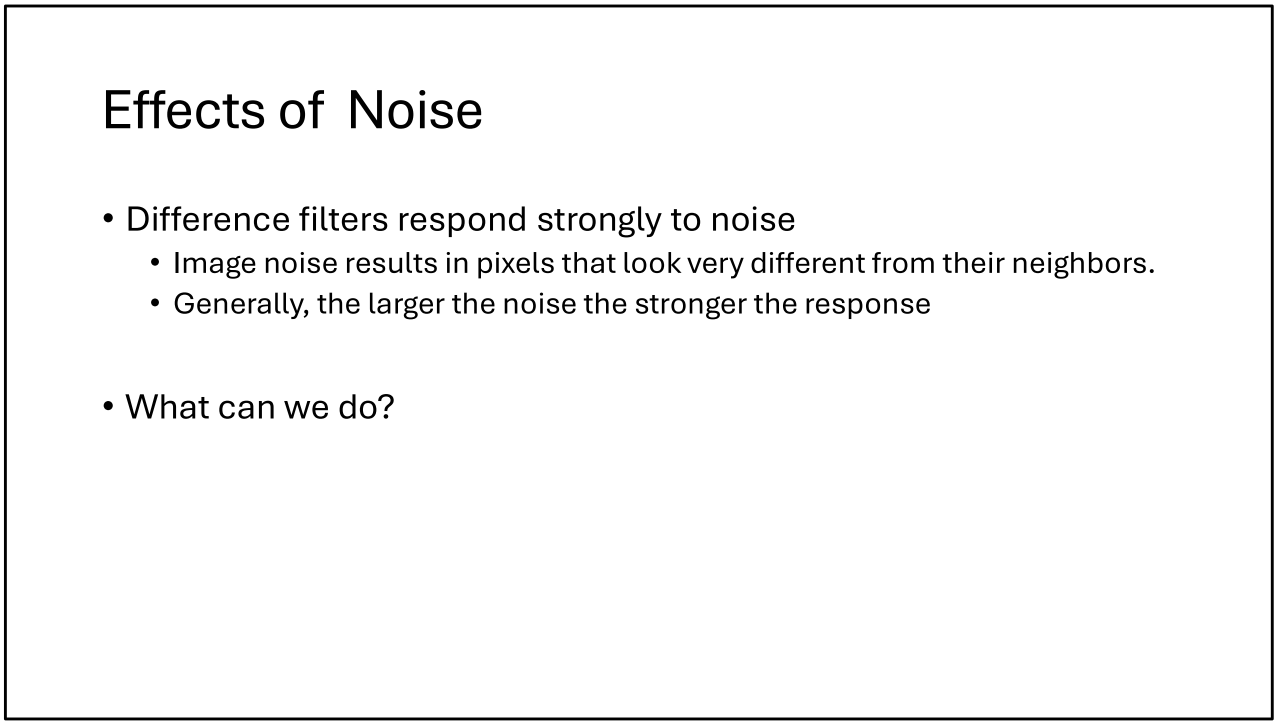
Consider a noisy signal

- Here's how the derivative looks like



Steve Seitz

Hmm… it would be almost impossible to detect where the edge occurred by inspecting the derivative graph with all that noise.
=============
It is important to understand how **noise affects edge detection**. Consider a single row or line of the image, as before, and plot the **intensity function** along it after adding noise.
In the noisy intensity profile, you can see many small fluctuations caused purely by the **noise**. When we compute the **first derivative**, these fluctuations produce numerous spurious peaks, making it **unclear where the true edges are**.
This illustrates how noise can significantly **corrupt the edge detection process**, highlighting the need for strategies to reduce noise before detecting edges.



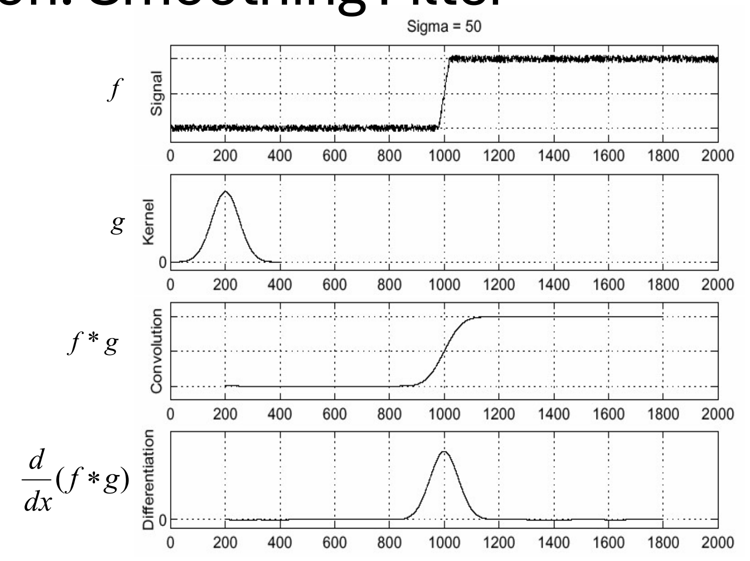
Effects of Noise

- Difference filters respond strongly to noise
  - Image noise results in pixels that look very different from their neighbors.
  - Generally, the larger the noise the stronger the response
- What can we do?

Difference filters and other edge detection operators are designed to respond strongly to **changes in intensity**. However, they are also highly sensitive to **high-frequency noise**.
When noise is present, it can **mask or degrade the edges**, making them appear less distinct from neighboring pixels. As a result, it becomes difficult to **distinguish true edges from noise**. The higher the noise level or frequency in the image, the more challenging it is to **separate noise from actual edges**

So, What can we do about it?

Solution: Smoothing Filter



To find edges, look for peaks in $\frac{d}{dx}(f*g)$

Steve Seitz

We can fix this by filtering the signal (image) first to smooth out the noise BEFORE computing the derivative! A gaussian filter is common.
===========
A simple solution to reduce the effect of noise is to **apply a Gaussian smoothing filter** to the image before edge detection.
For example, if we take the intensity profile along a line and **convolve it with a Gaussian filter**, we obtain a **smoothed profile**. When we then compute the **first derivative** of this smoothed profile, the edges become **clearly detectable**, while the noise is significantly reduced. This allows us to accurately identify the **positions of true edges** in the image.

Derivative Theorem of Convolution

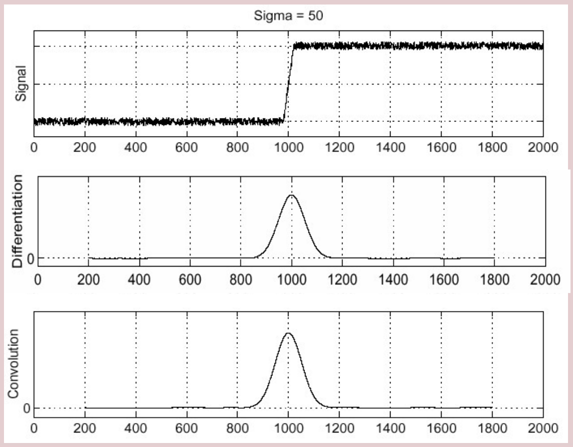
Theorem

Convolution is differentiable: $\frac{d}{dx}(f*g)=f*\frac{d}{dx}g$

f

$\frac{d}{dx}(f*g)$

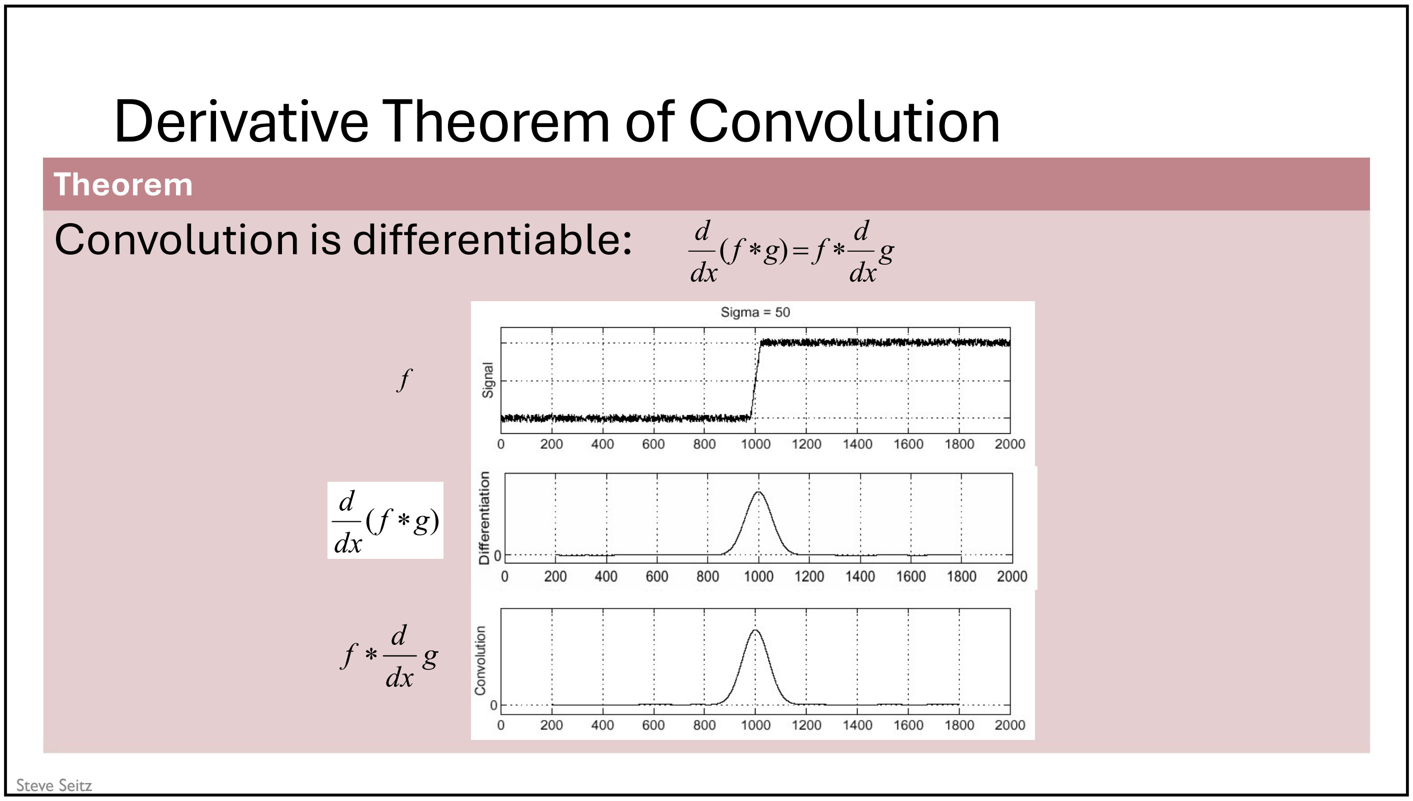
$f*\frac{d}{dx}g$



Steve Seitz

We can also consider the **derivative operator** and **convolution** to be **interchangeable**, since both are **linear operators**. This allows us to apply the **chain rule** and change the order of operations, which offers a significant **computational advantage**.

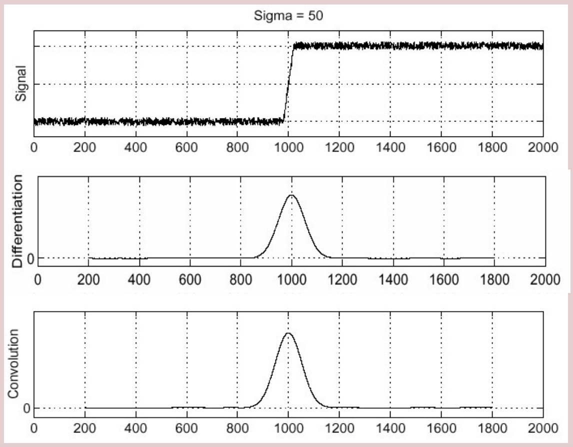
For example, instead of first smoothing the image with a Gaussian filter and then computing its derivative, we can **compute the derivative of the Gaussian kernel** itself. Convolving this **differentiated Gaussian kernel** with the original intensity profile produces the **same edge-detected output**, but with **fewer computations**. This approach is widely used in practice to efficiently detect edges while reducing noise.



Derivative Theorem of Convolution

Theorem

Convolution is differentiable: $\frac{d}{dx}(f*g) = f*\frac{d}{dx}g$



f

$\frac{d}{dx}(f*g)$

$f*\frac{d}{dx}g$

Steve Seitz

This means that we can compute the derivative just on the filter g and convolve the image by the differentiated filter to get the same result as on the previous slide.
Useful because instead of convolving the image twice – once to denoise and once to compute the derivative – we only need to convolve once.

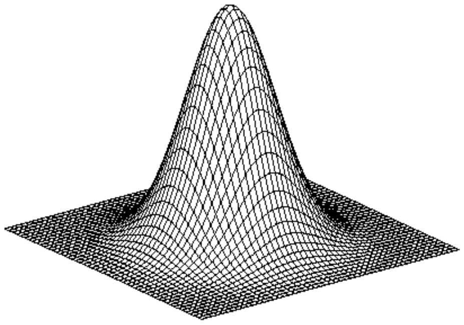
One student question: “Why don’t we have to apply the chain rule?”

We have d/dx (f * g) where f and g are functions, and * is convolution. This represents the continuous case in 1D.
We see this identity:
d/dx (f * g) = f * dg/fx
Due to the commutativity of convolution, this is also true:
d/dx (g * f) = f * dg/fx
Or equally:
d/dx (f * g) = df/fx * g
The derivative of a convolution of two functions is equal to the derivative of either of the functions convolved with the other function.
This also holds for partial derivatives (for 2D convolution).

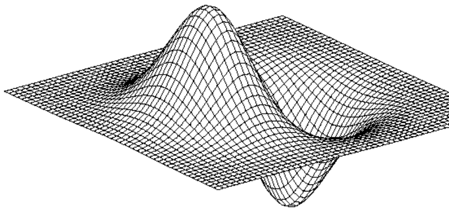
Let's look at the 1D continuous convolution case again. We didn't define it in lecture, but let's do it now.
Convolution is a definite integral (our product in the discrete case):
(f * g)(x) = INT(f(k)g(x-k))dk | k=-inf to k=inf
where x is our domain, and k is our dummy variable that slides f over g. Loosely, g would be our image, and f our filter.
So, when we want d/dx, we're differentiating an integral.
d/dx (f * g)(x) = d/dx INT(f(k)g(x-k)dk) | k=-inf to k=inf
By the Liebniz rule (differentiation under the integral), as our kernel doesn't depend on x, we can bring the (now partial) derivative inside the integrand.
d/dx (f * g)(x) = INT(f(k) ∂/∂x g(x-k)dk) | k=-inf to k=inf
Which is f * dg/dx.

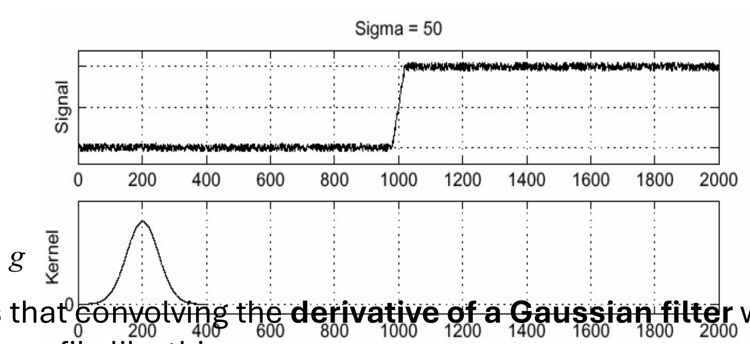
The proof for the discrete case with the finite difference operator D can be found similarly (as in Section D.2.4 below), along with proofs of many of the other properties of convolutions.
http://web.eecs.utk.edu/~roberts/WebAppendices/D-ConvProperties.pdf
Thanks to Prof. Michael Roberts at UTK for this document.
Note that Prof. Roberts is in EECS, and is used to convolving time series, so x is usually t.

Derivative of 2D Gaussian Filter



* [1 -1] =





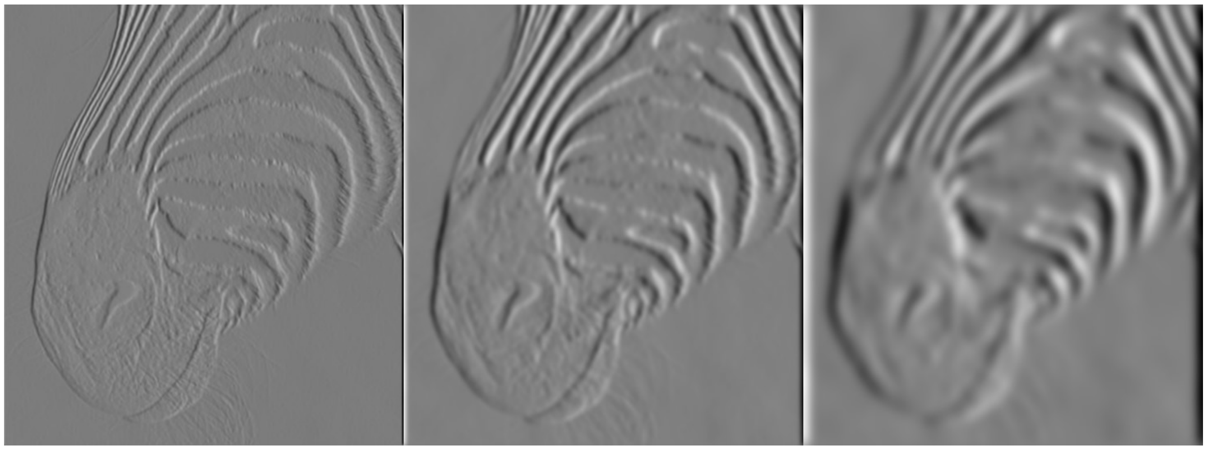
James Hays

What we observed is that convolving the **derivative of a Gaussian filter** with an **edge detector** produces a profile like this.

An interesting question to consider is whether this **output is separable**. You can explore this mathematically by taking a **2D Gaussian function**, convolving it with an edge-detection kernel, and examining the result. Determining whether the operation is separable can provide useful insights and is a good exercise to deepen your understanding.

Smoothing-Localization Tradeoff

Smoothed derivative removes noise, but blurs edge.
Also finds edges at different 'scales'.



1 pixel 3 pixels 7 pixels

The bigger the kernel, the more smoothed out the edges of the image are since we start killing off lower frequencies.

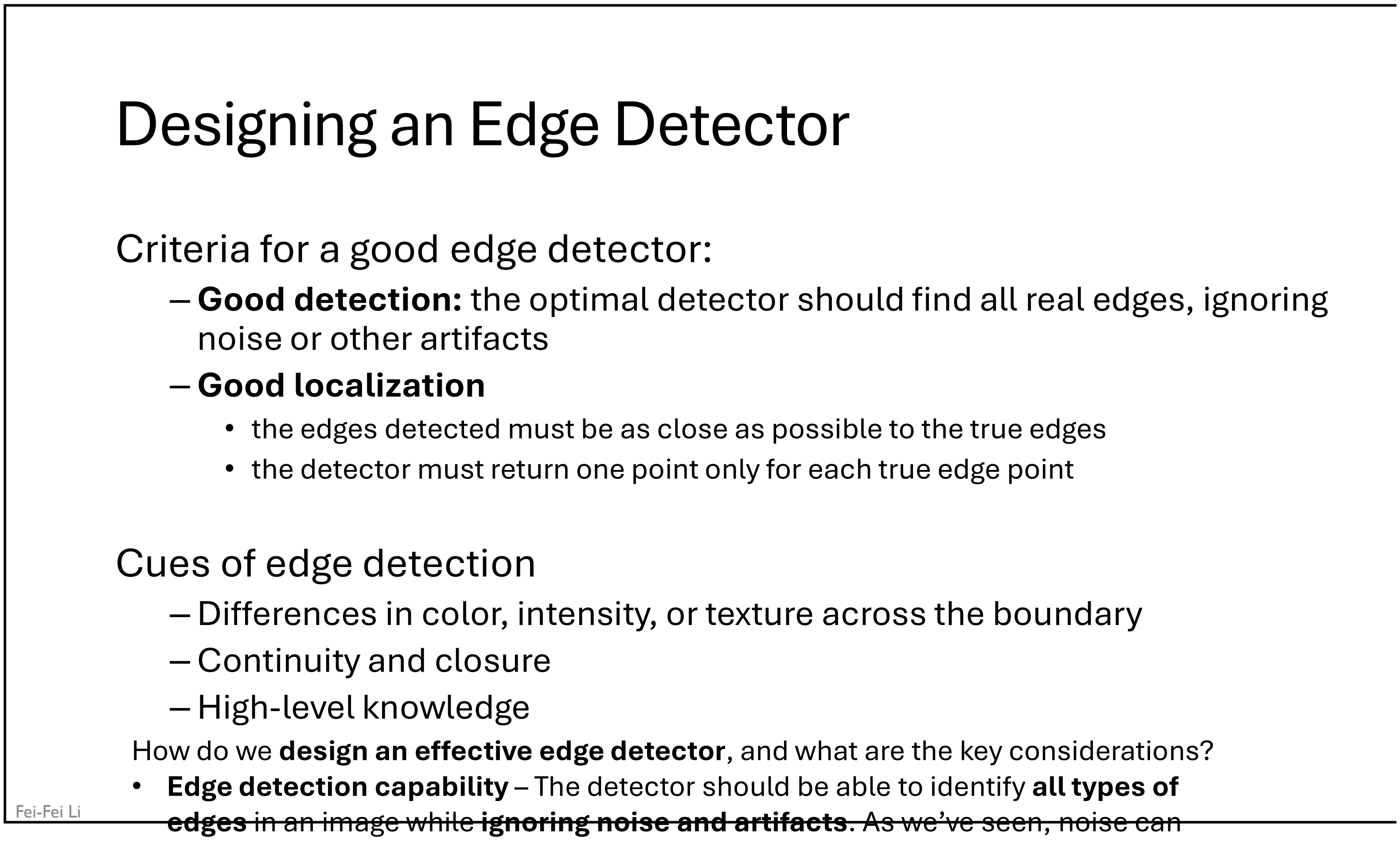
David Forsyth

==============

We have already seen that **edge detection is influenced by the smoothing operation**, and the effect is clearly visible here.

If we apply a **Gaussian smoothing filter** of different widths — for example, 1 pixel, 3 pixels, or 7 pixels — the results of edge detection change. A **narrow filter** (1 pixel) preserves **fine details**, allowing detection of small, precise edges. A **wider filter** (7 pixels), however, smooths out fine variations and highlights **larger-scale edges**, producing a different edge profile.

The choice of filter depends on the **application** and the level of **edge localization** required. Narrow filters are better for detecting fine details, while wider filters provide smoother profiles but may reduce localization accuracy.



Designing an Edge Detector

Criteria for a good edge detector:

- **Good detection:** the optimal detector should find all real edges, ignoring noise or other artifacts
- **Good localization**
  - the edges detected must be as close as possible to the true edges
  - the detector must return one point only for each true edge point

Cues of edge detection

- Differences in color, intensity, or texture across the boundary
- Continuity and closure
- High-level knowledge

Fei-Fei Li

How do we **design an effective edge detector**, and what are the key considerations?

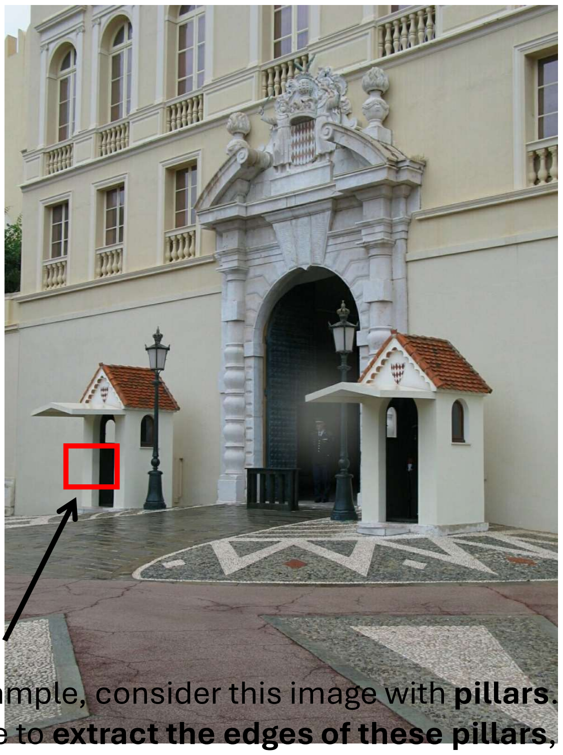
- **Edge detection capability** – The detector should be able to identify **all types of edges** in an image while **ignoring noise and artifacts**. As we've seen, noise can significantly affect edge detection, so a robust detector must account for this.

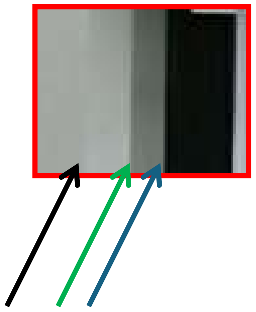
- **Good localization** – Detected edges should be as **close as possible to the true edges** in the image. Ideally, an edge detector marks **one pixel per edge**. If multiple pixels are detected for a single edge, the localization accuracy decreases.

- **Multi-cue detection** – A strong edge detector should utilize **various cues**, including **color intensity differences, textures, continuity, closures**, and even **high-level contextual knowledge** when available.

By considering these criteria, we can design edge detectors that are **accurate, robust, and reliable** across different types of images and applications.

Closer Look at Edges





Derek Hoiem

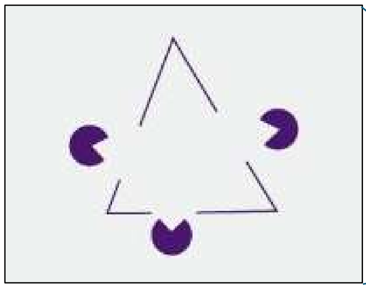
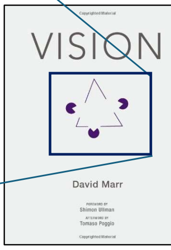
For example, consider this image with **pillars**. A well-designed edge detector should be able to **extract the edges of these pillars**, even if they are **not easily visible** in the image. We know these edges exist because they correspond to **depth discontinuities**, highlighting the importance of a robust detector that can capture edges beyond just high-contrast regions.

Designing an Edge Detector

"All real edges"

- We can aim to differentiate later which edges are 'useful' for our applications.
- Our perceptual system can complete shapes even when no edge exists.
- *No correct answer*

"Edges are imposed, not detected."
- Koenderink

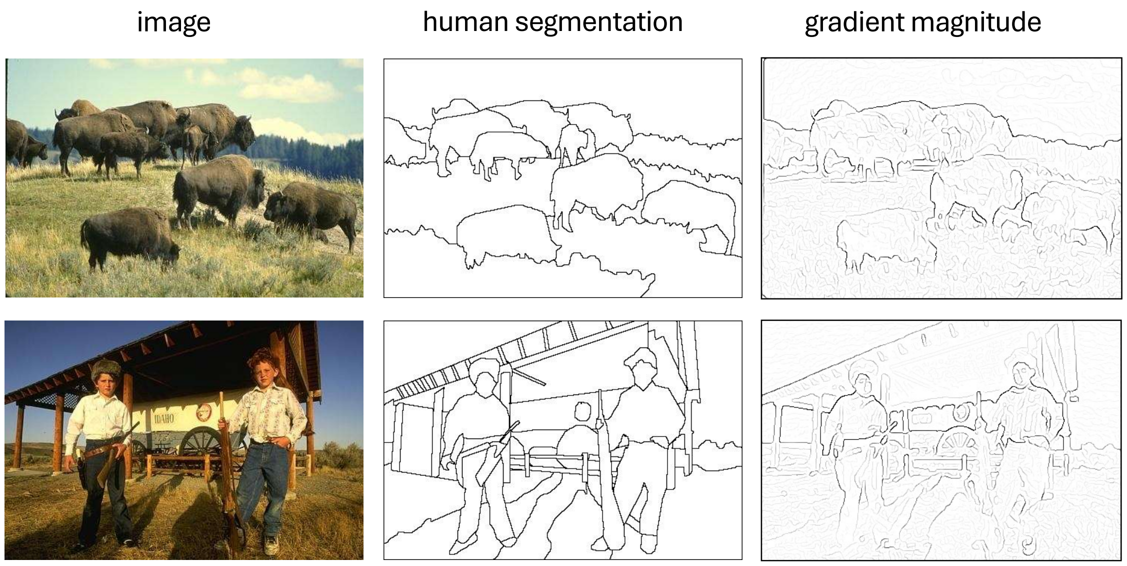



Marr's famous Vision book!

Jan Johan Koenderink (b. 1943)
See Svetlana Lazenbik's course on Computer Vision: Looking Back to Look Forward

David Marr (b. 1945, d.1980) – neuroscientist investigating vision @ MIT. His book was published posthumously after his early death due to ill health.
Computer vision's prize for best paper at ICCV (International Conference on Computer Vision) is named after him.
Foreword:
https://en.wikipedia.org/wiki/David_Marr_%28neuroscientist%29
=============
Designing an edge detector means creating a system that can **identify all true edges** in an image, allowing us to **select the edges relevant** to a particular application.
While the detector should aim to find every edge, there is **no exhaustive definition of what constitutes an edge**, so we cannot define it rigidly.
Therefore, it is essential to **visually inspect and analyze** the output of the edge detector to determine whether it **meets the desired criteria** for detecting meaningful edges.

Where do humans see boundaries?



James Hays. http://www.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/segbench/

Before designing an edge detector, we need to ask ourselves **what we consider to be edges**. For humans, **smaller details are often not important**, whereas most edge detectors identify **all intensity changes**, including fine details.

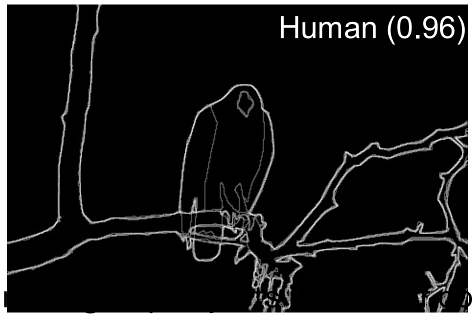
For example, consider images from the **Vertically Segmented Database**. On the left are the **original images**, and in the middle are **human-generated segmentations**. Humans tend to **ignore fine details**, focusing only on major edges. In contrast, a standard edge detector identifies **both large and small edges**, capturing much more detail than humans typically perceive.

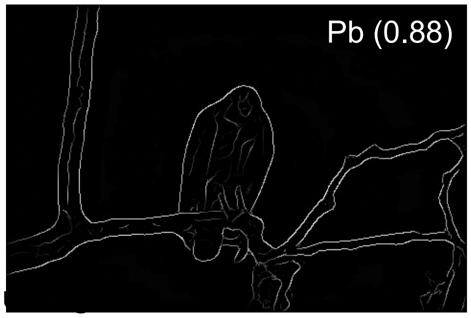
This highlights a potential **mismatch between human perception and edge detectors**. The choice depends on your **application**:

- If you need **all possible edges**, a geometry-based edge detector is sufficient.
- If you want **human-level segmentation**, you must incorporate **prior knowledge or higher-level information** to filter out insignificant edges, focusing only on the meaningful ones.

pB Boundary Detector vs. Humans

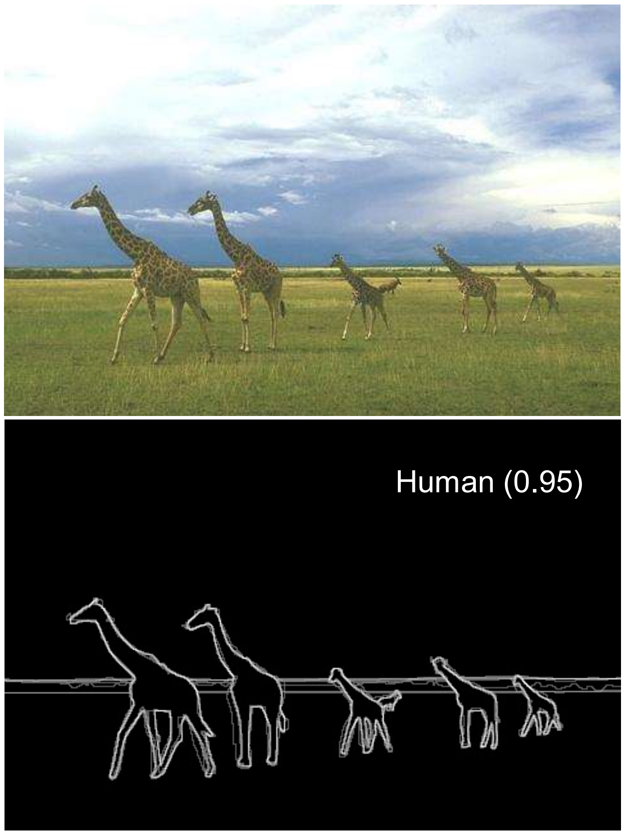
Score = confidence of edge.
For humans, this is averaged across multiple participants.



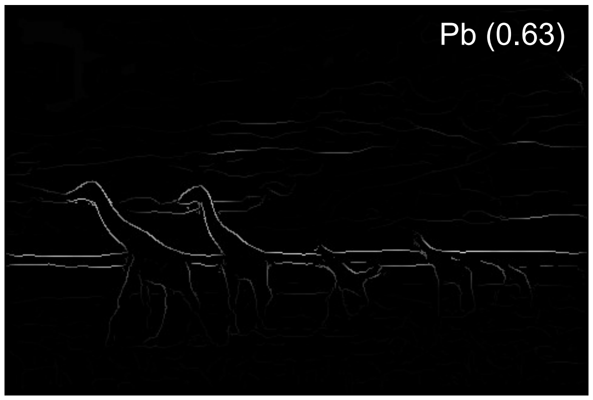


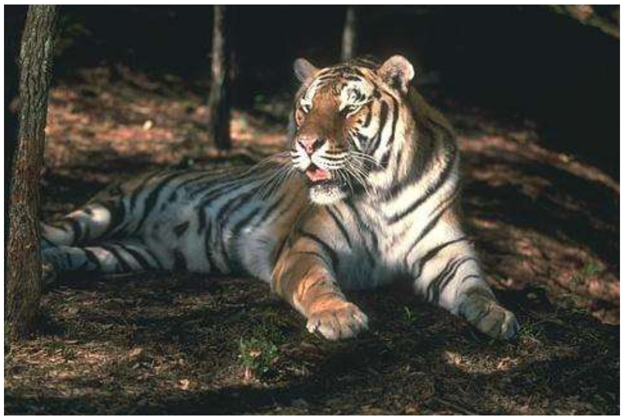
Humans te ontiguo

http://www.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/papers/mfm-pami-boundary.pdf

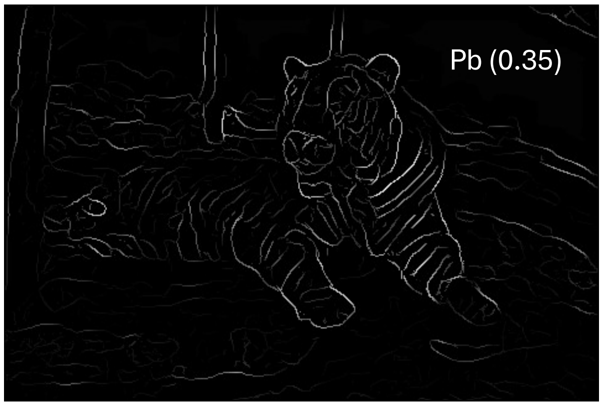


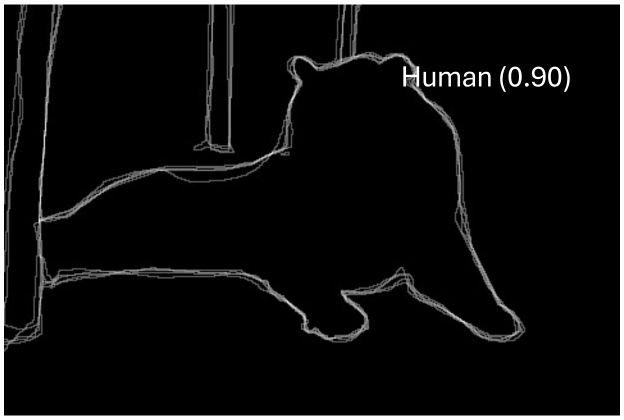
Score = confidence of edge.
For humans, this is averaged across multiple participants.





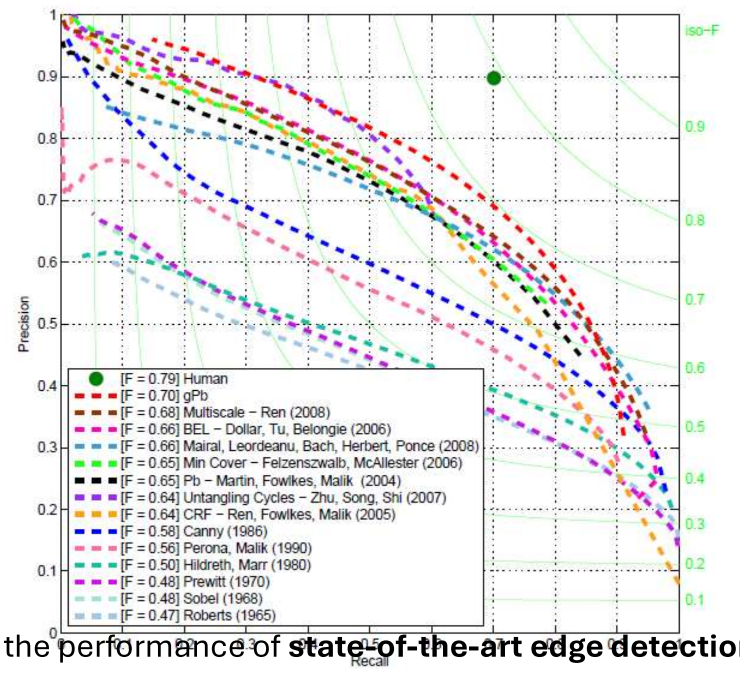
Score = confidence of edge.
For humans, this is averaged across multiple participants.





For more:
http://www.eecs.berkeley.edu/Research/Projects/CS/vision/bsds/bench/html/108082-color.html

45 Years of Boundary Detection*



Arbelaez, Maire, Fowlkes, and Malik, TPAMI 2011

* Pre-de

This plot summarizes the performance of **state-of-the-art edge detection methods** over the past 45 years of computer vision research. The **green dot** represents **human-level performance**, with approximately **0.7 recall** and **0.9 precision**. As you can see, most current methods fall below this level, indicating a **significant potential for improvement** in edge detection.

Again, I emphasize that **what counts as satisfactory performance depends on your application**. The evaluation of an edge detector should always consider the **specific goals and requirements** of the task at hand.

State of Edge Detection

Local edge detection works well

- 'False positives' from illumination and texture edges (depends on our application).

Some methods to consider longer contours

Modern methods that "learn" from data.

Poor use of object and high-level information.

James Hays

The current state of **edge detection** is largely **local**. Local edge detectors work well, but they often produce **false positives** caused by **illumination changes, textures, or color gradients** that are not meaningful.

Some methods attempt to incorporate **longer contours or extended regions** to improve detection, but the most advanced approaches today are **data-driven**, learning from large datasets. These modern, learning-based methods represent the **state of the art**, though we will not cover them in this lecture. You may encounter them in a **deep learning course** or by exploring data-driven edge detection research projects.

Traditional, non-data-based edge detectors also struggle because they **do not incorporate high-level information** about objects or context, which limits their ability to distinguish meaningful edges from irrelevant details.

Canny Edge Detector

- Next time!