Edge Detection Using Laplacian of Gaussian Operator

A Mathematical and Coding Perspective

Today’s Topics
Edge Detection with LoG

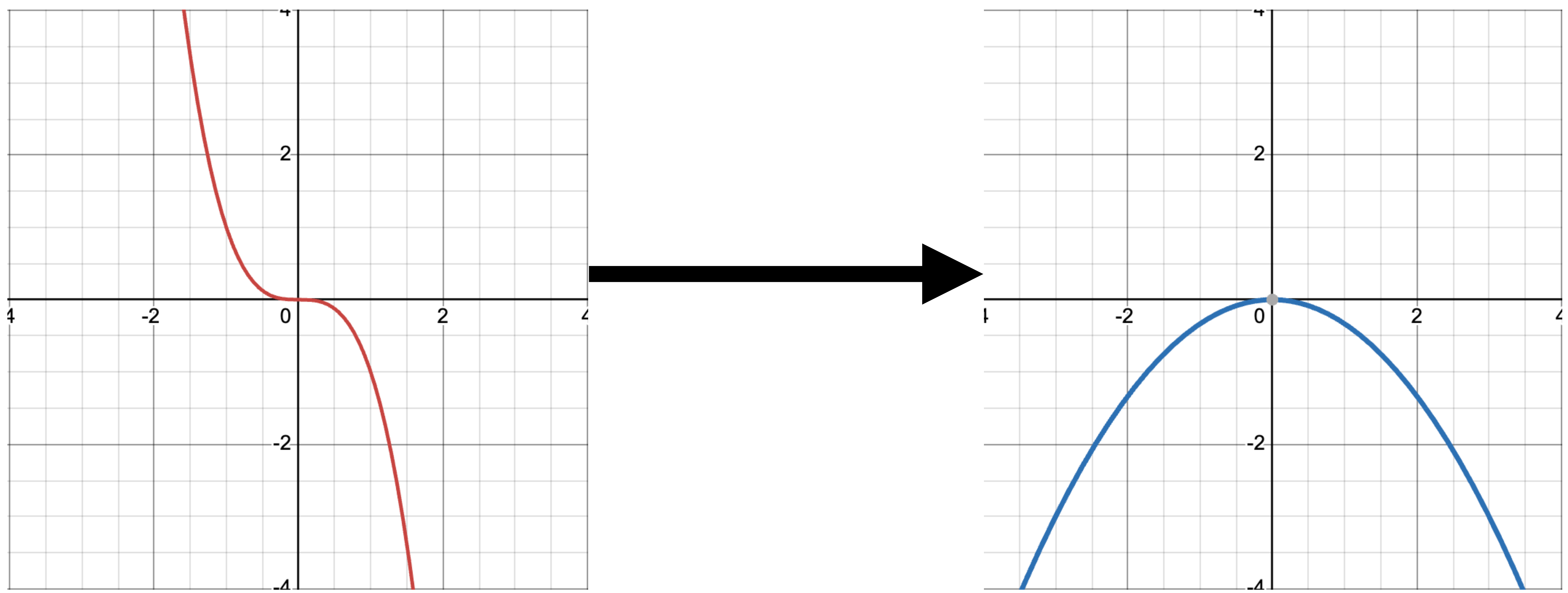
1. Concepts!
   - First Derivative Filter (Sobel)
   - Second-Derivative Filters (Laplacian)
   - Zero-Crossings
2. Laplacian of Gaussian (LoG) Filter
   - Useful for Finding Edges

First Derivative Filter (Sobel)

Concepts

- The first derivative allows us to find the instantaneous rate of change at any point on the curve. This means it is very good at detecting rapid changes, e.g., potential edges.

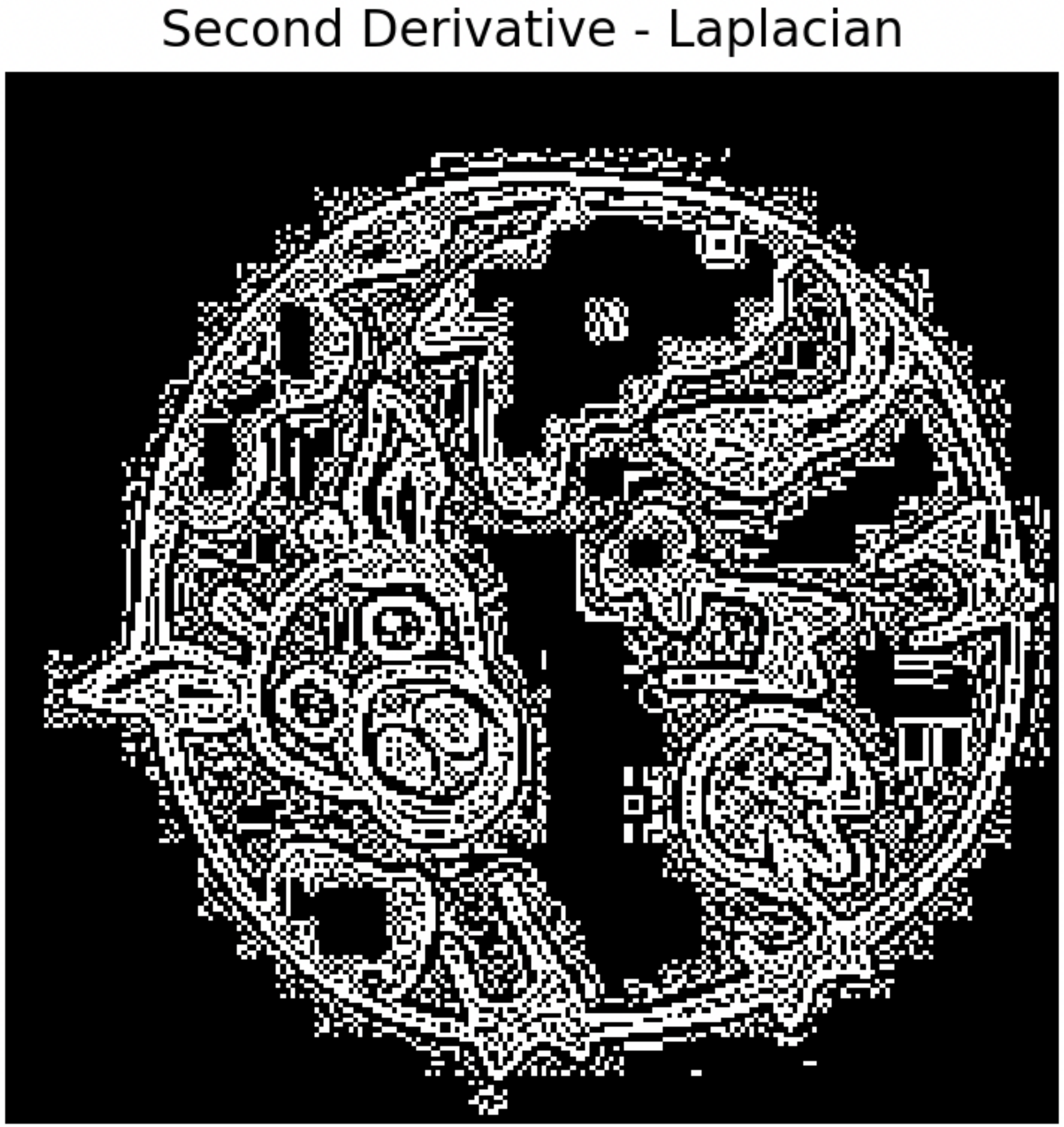
-x^3 -(1/3)x^2



Second Derivative Filter (Laplacian)

Purpose

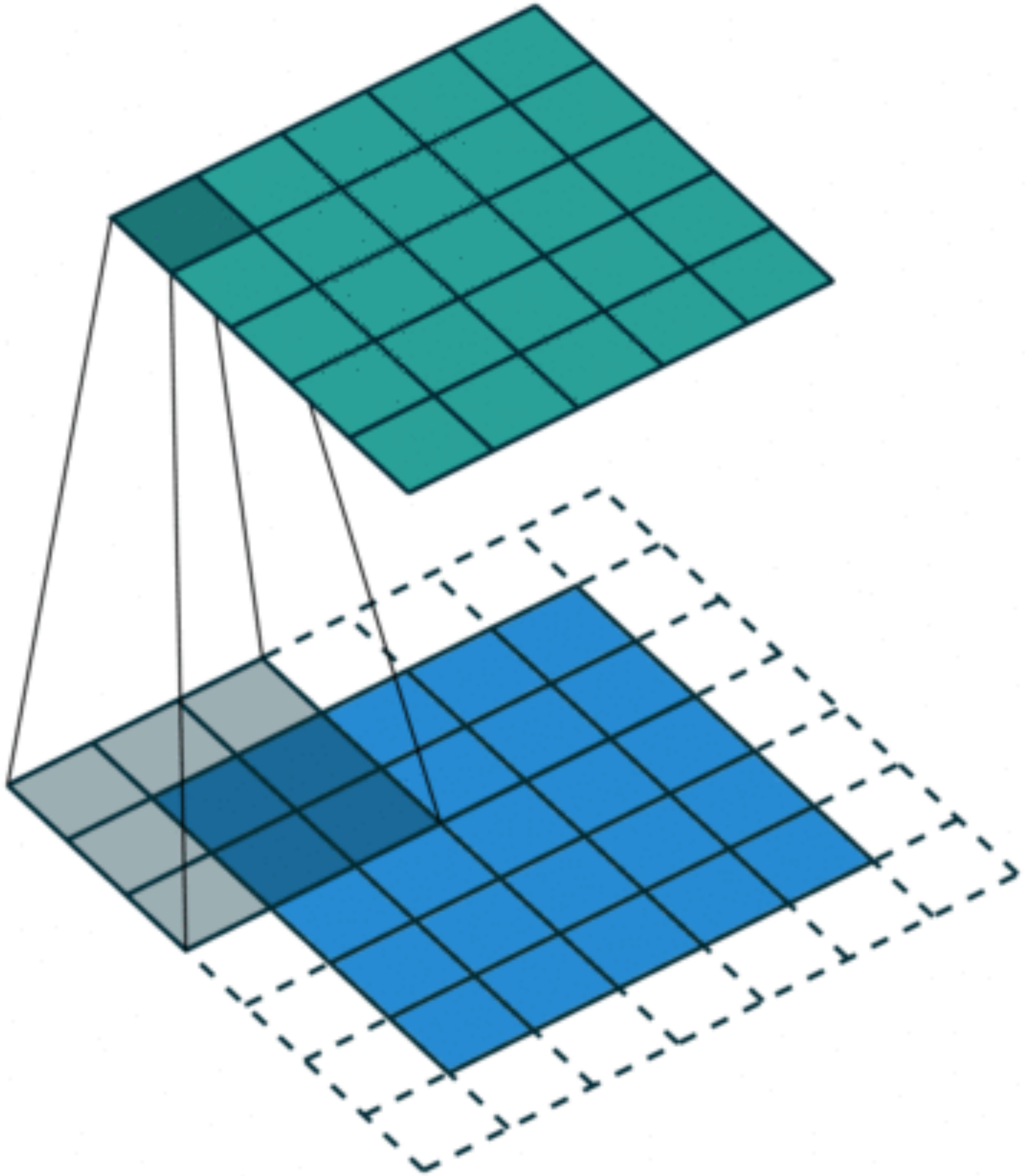
- Second-derivative filters, like the Laplacian filter, are used in image processing to **detect areas of rapid intensity change** (e.g., edges).
- The Laplacian filter is based on the second derivative of the image intensity, measuring the rate at which the first derivatives change. It is particularly **sensitive to fine details**.
- This allows the filter to **capture edges from all directions equally**, unlike directional filters that must be applied multiple times for different orientations.



Second Derivative Filter (Laplacian)

Kernel Operation

- The algorithm uses a small matrix, called a kernel or filter, which slides over the entire image.
- This kernel is typically a 3x3 matrix.
- As the kernel moves over each pixel, it looks at the neighboring pixels to calculate the gradient of the image at that point.



https://towardsdatascience.com/types-of-convolutions-in-deep-learning-717013397f4d

Second Derivative Filter (Laplacian)

Derivation

- The Laplacian operator is applied to an image to **measure changes in intensity** in a way that highlights regions where these changes are the greatest—typically at the edges of objects.

- In two dimensions, the Laplacian is given by the sum of the second derivatives with respect to x and y:

$$\nabla^2 f(x, y) = f_{xx}(x, y) + f_{yy}(x, y)$$

- This can be **approximated using a kernel or matrix** that is applied to the image.

Second Derivative Filter (Laplacian)

Taylor Polynomial Refresher

The n-th order Taylor polynomial $P_n(x)$ of a function $f(x)$ at a point a is defined as:

$$P_n(x) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \cdots + \frac{f^{(n)}(a)}{n!}(x-a)^n, \tag{1}$$

where $f^{(k)}(a)$ denotes the k-th derivative of f evaluated at a, and $n!$ represents the factorial of n.
For example, consider the function $f(x) = e^x$, $g(x) = \cos(x)$, and $h(x) = \log(1+x)$. The Taylor expansions at $x = 0$ (also called Maclaurin series) for these functions are:

$$e^x \approx 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots ,$$

$$\cos(x) \approx 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \cdots ,$$

$$\log(1+x) \approx x - \frac{x^2}{2} + \frac{x^3}{3} - \cdots .$$

Second Derivative Filter (Laplacian)

Taylor Polynomials in Discrete 2D Spaces

$$P_2(x,y) = f(a,b)+f_x(a,b)(x-a)+f_y(a,b)(y-b)+\frac{1}{2}f_{xx}(a,b)(x-a)^2+f_{xy}(a,b)(x-a)(y-b)+\frac{1}{2}f_{yy}(a,b)(y-b)^2, \tag{2}$$

where f_x, f_y, f_{xx}, f_{xy}, and f_{yy} are the partial derivatives of f.

Second Derivative Filter (Laplacian)

Image Kernel

Image separated into kernels of 3x3 like below:

$$j \uparrow$$

$I_{i-1,j+1}$	$I_{i,j+1}$	$I_{i+1,j+1}$
$I_{i-1,j}$	$I_{i,j}$	$I_{i+1,j}$
$I_{i-1,j-1}$	$I_{i,j-1}$	$I_{i+1,j-1}$

$$\longrightarrow i$$

The Laplacian of an image function $I(x, y)$, applied using a discrete approximation, is given by:

$$\Delta I = I(x+1, y) + I(x-1, y) + I(x, y+1) + I(x, y-1) - 4I(x, y), \tag{4}$$

Second Derivative Filter (Laplacian)

Laplacian Applied to Image Kernel

$$
\begin{aligned}
f(i+1,j) &\approx f(i,j) + f_x(i,j) + \frac{1}{2} f_{xx}(i,j), \\
f(i-1,j) &\approx f(i,j) - f_x(i,j) + \frac{1}{2} f_{xx}(i,j), \\
f(i,j+1) &\approx f(i,j) + f_y(i,j) + \frac{1}{2} f_{yy}(i,j), \\
f(i,j-1) &\approx f(i,j) - f_y(i,j) + \frac{1}{2} f_{yy}(i,j).
\end{aligned}
$$

Second Derivative Filter (Laplacian)

Numerical Approximation of the Laplacian

$$
\begin{aligned}
\Delta f(i,j) &\approx \frac{\partial^2 f}{\partial x^2}\Big|_{(i,j)} + \frac{\partial^2 f}{\partial y^2}\Big|_{(i,j)} \\
&\approx f(i+1,j) - 2f(i,j) + f(i-1,j) + f(i,j+1) - 2f(i,j) + f(i,j-1) \\
&= f(i+1,j) + f(i-1,j) + f(i,j+1) + f(i,j-1) - 4f(i,j).
\end{aligned}
$$

Second Derivative Filter (Laplacian)

Kernel Operation

- The values in the **discrete Laplacian matrix** L correspond to the coefficients of the function values from the numerical approximation of the Laplacian:

$$L = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

- In matrix L, the non-central entries with a value of –1 arise from subtracting each neighboring point's value, while the central value of 4 corresponds to the negative sum of the function's second-order derivatives over the four immediate neighbors.

- When this kernel is convolved with an image, it serves as an **approximation to the continuous Laplacian operator**, effectively emphasizing areas of rapid intensity change.

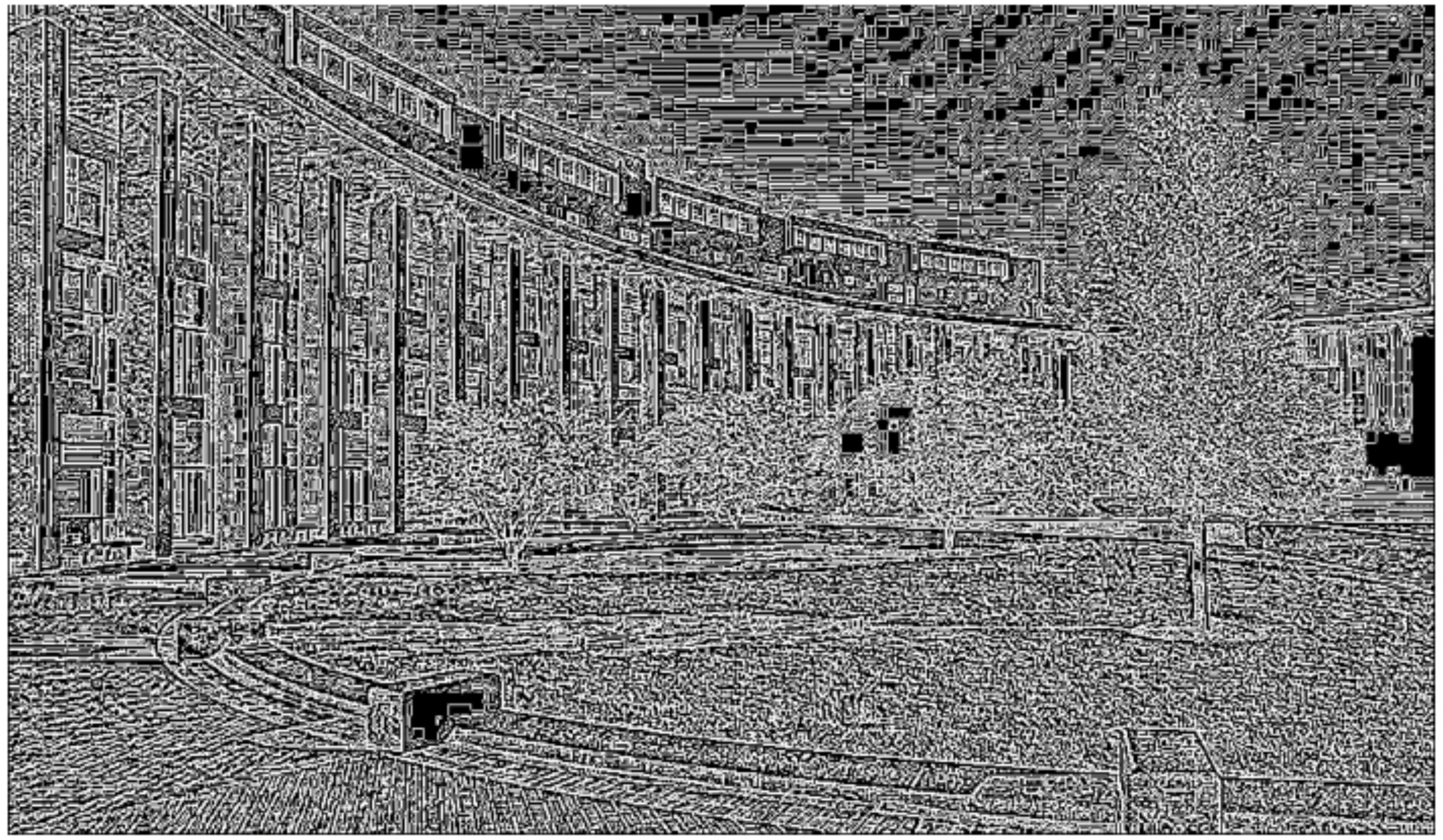
Second Derivative Filter (Laplacian)

Edge Detection

- When the Laplacian kernel is applied to an image, the result is a new image that shows edges as sharp transitions in intensity.
- Areas with zero-crossings (where the sign of the Laplacian changes) correspond to edges. This method can **detect edges more sharply** than first-derivative filters but is also more **sensitive to noise**.

Original Image

Second Derivative - Laplacian



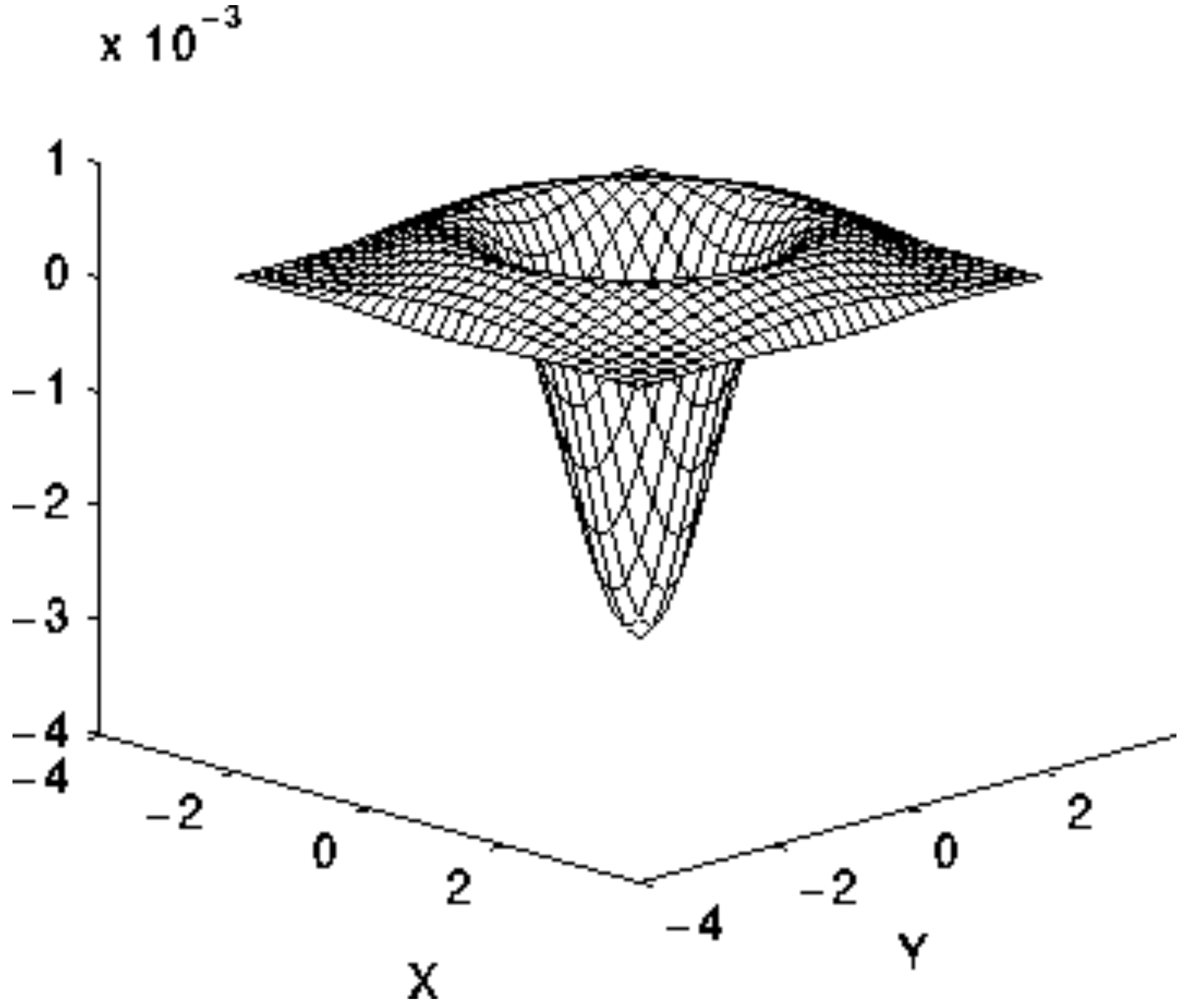
But there is much noise in the Laplacian-filtered image… how do we fix that?

Laplacian over Gaussian! (LoG) 😍

Laplacian of Gaussian (LoG)

Purpose

- The Laplacian of Gaussian filter combines **Gaussian blur** and Laplacian edge detection to identify edges in images more accurately while **reducing noise sensitivity**. This method effectively locates edges by **suppressing noise beforehand** with a Gaussian filter.



https://homepages.inf.ed.ac.uk/rbf/HIPR2/log.htm

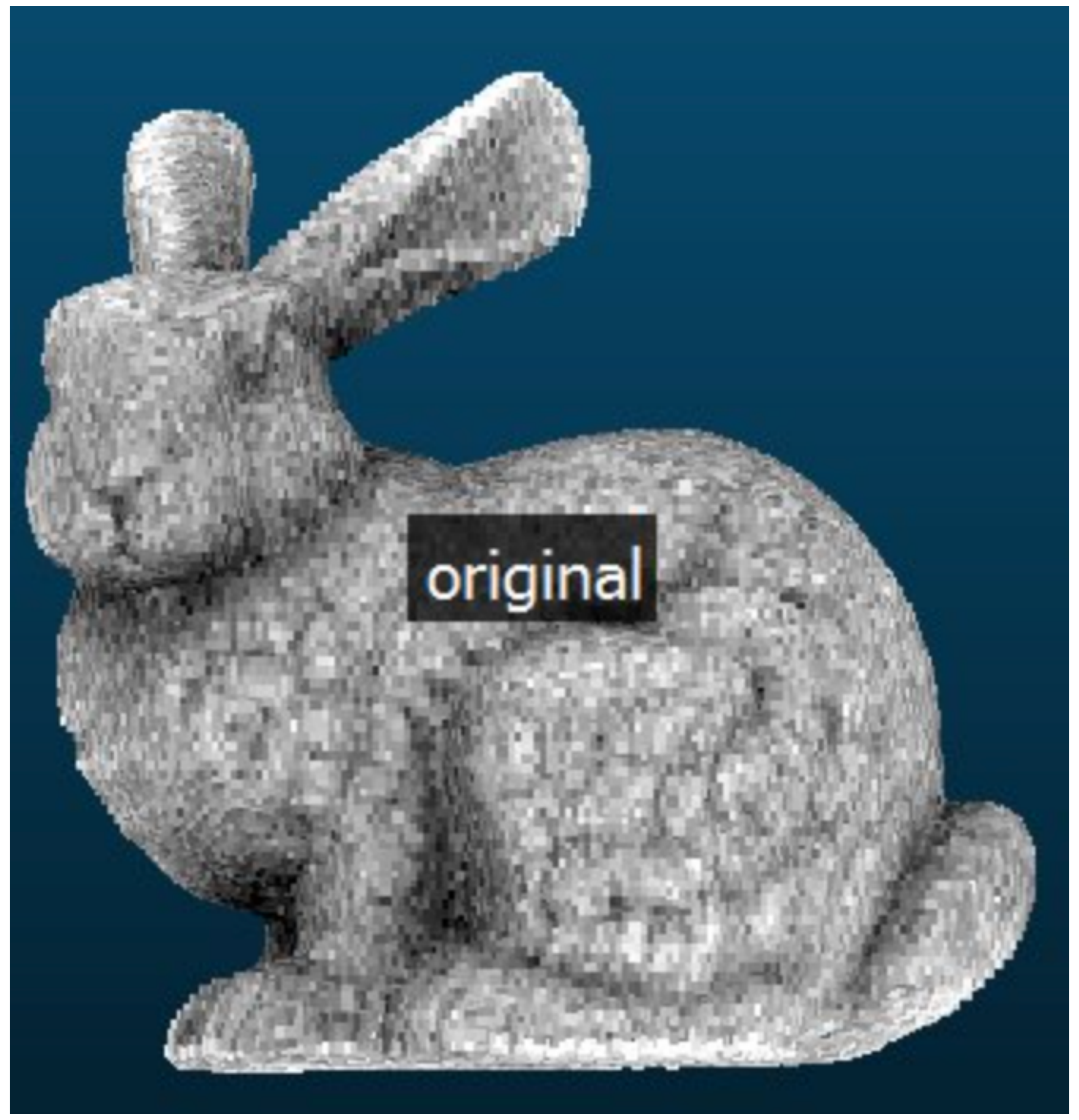
Laplacian of Gaussian (LoG)

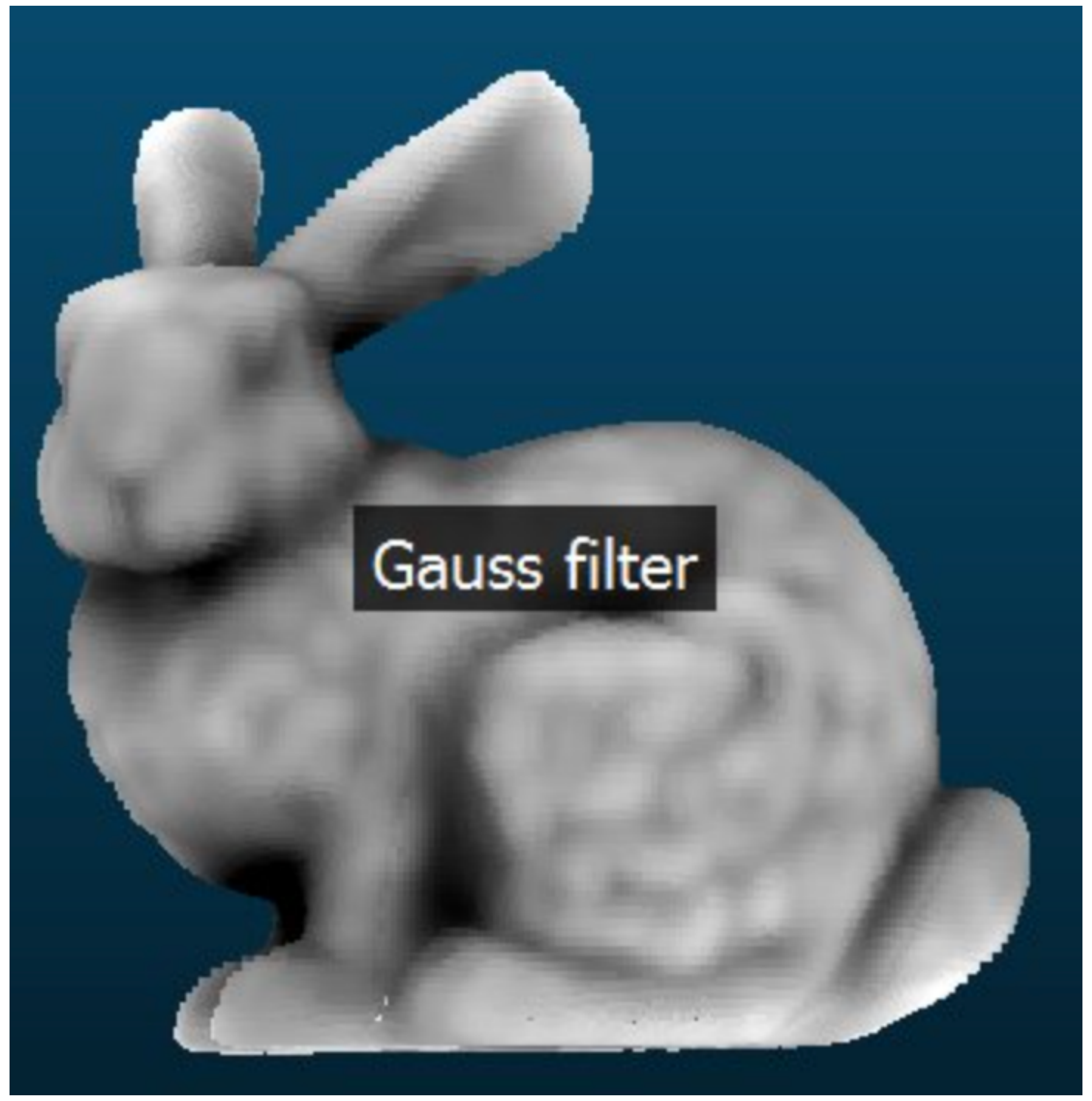
Gaussian Filter

- Gaussian Filter: Applies a Gaussian blur to smooth the image. This step is crucial as it **reduces noise and minor fluctuations** in the image's intensity before applying edge detection.

- It is used to remove Gaussian noise and is a realistic model of a defocused lens. **σ defines the amount of blurring.**

- We will not discuss this in-depth for the scope of this mini-lecture, assuming that our target audience has a deep understanding of Gaussian distribution from ModSim.





https://www.cloudcompare.org/doc/wiki/images/5/5f/Cc_sf_gaussian_filter.jpg

Laplacian of Gaussian (LoG)

Process

- Step 1: Apply the **Gaussian filter** to smooth out the image.
- Step 2: Apply the **Laplacian operator** to the smoothed image.
- The combination of these steps can be represented by a **single convolution operation** using the LoG kernel.

Gaussian Filter

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Laplacian of Gaussian

$$\nabla^2 G(x, y) = \left(\frac{x^2 + y^2 - 2\sigma^2}{\sigma^4} \right) e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Laplacian of Gaussian (LoG)

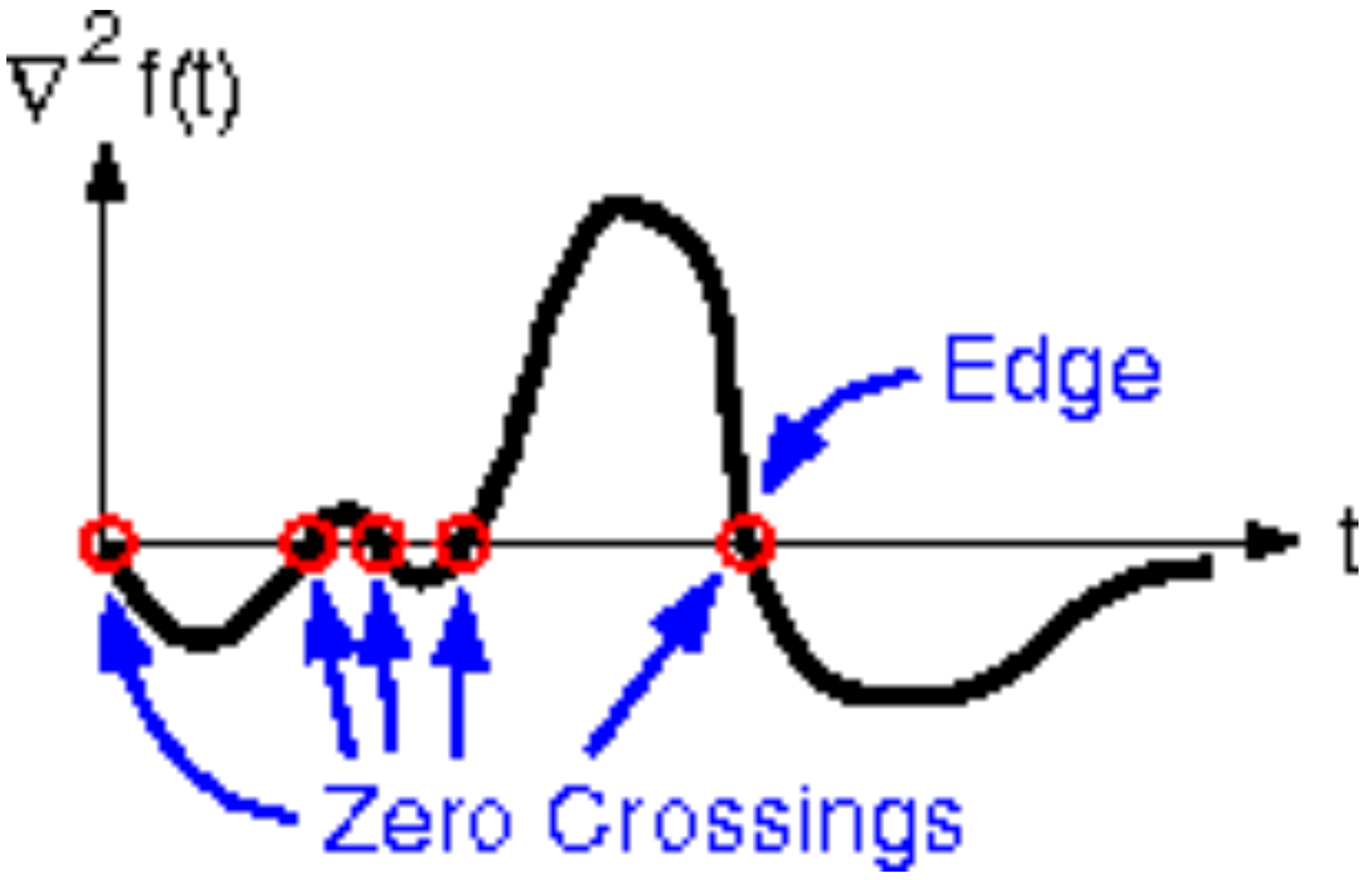
Process

- Step 3: **Edge Detection**

- The final result after applying the LoG filter will show **zero-crossings** where the intensity changes sign sharply, indicating an edge.

- Because the image was first smoothed, the edges detected are **less likely to be affected by noise** than if the Laplacian filter were used alone.

https://www.owlnet.rice.edu/~elec539/Projects97/morphjrks/laplacian.html




```python
def zero_crossing(img):
    img[img > 0] = 1
    img[img < 0] = 0
    out_img = np.zeros(img.shape)
    for i in range(1, img.shape[0]-1):
        for j in range(1, img.shape[1]-1):
            if img[i, j] > 0 and any_neighbor_zero(img, i, j):
                out_img[i, j] = 255
    return out_img
```

Let’s try it ourselves and compare the results!

Laplacian of Gaussian (LoG)

Implementation

- Download the code library we wrote using this line:
- pip install git+https://github.com/dcoder0111/QEA2_LaplacianOverGaussianOperator.git#egg=QEA2_LaplacianOverGaussianOperator
- Download the example.py file from the git repo or write your own code according to the documentation to test it out on different images!
- Hint: Some filters work better on certain types of images!