C:\opencv\build\x64\vc16\lib

opencv\_world4110d.lib

C:\opencv\build\include

D:/FresherXavisTech/Image/

<https://www.slideserve.com/eloise/canny-edge-detection>

<http://contents2.kocw.or.kr/KOCW/document/2018/handong/hwangsungsoo0228/15.pdf>

<https://docs.opencv.org/3.4/da/d5c/tutorial_canny_detector.html>

<https://www.cs.bilkent.edu.tr/~saksoy/courses/cs484-Spring2010/slides/cs484_edge.pdf>

<https://courses.cs.washington.edu/courses/cse576/16sp/Slides/4_Edges.pdf>

<https://homepages.inf.ed.ac.uk/rbf/AVINVERTED/STEREO/av5_edgess.pdf>

<https://justin-liang.com/tutorials/canny/#grayscale>

<https://www.youtube.com/watch?v=aCbfvxgYy8s>

**Slide 1:**

Now, I will present my results comparing the Gaussian filter and the Bilateral filter.

I applied both filters to a noisy image, and we can see the results here.  
On the left is the original image, in the middle is the result after applying the Gaussian filter, and on the right is the result from the Bilateral filter.

First, I fix the brightness sigma = 1.0, and I vary the spatial sigma, starting from 1.0

Let’s look at the Gaussian filter first.

* At σ = 1, the result is slightly blurred, and the noise is reduced quite well.
* When increasing σ to 2.0, 3.0, and 10.0, the image becomes increasingly blurred. While noise is removed almost completely, edges and small details are also blurred.
* At **σ = 2**, we already see that the edges are softened, the light gray region and the small gray dots become hard to see.
* At **σ = 3**, those details become even harder to recognize.
* And at **σ = 10**, we can no longer see the gray areas or the small dots — only the basic shape of the original image remains visible.

Now let’s go back to the **Bilateral filter**, with **range sigma = 1**.

This is a relatively small value, meaning that **only pixels with intensity values similar to the central pixel** will be included in the smoothing process.

Pixels that have a large intensity difference — such as noise or edges — are **excluded from the filtering effect**.

As a result, the output of the Bilateral filter remains **almost unchanged**, even when we increase **σs** to 2, 3, or 10.

In all four cases, the image looks nearly the same: noise is only slightly reduced, and **edges are very well preserved**.

Slide 2:

Now let’s consider the case where **σr is larger**, for example **σr = 15**.

In this case, when we increase **σs**, we can observe that image becomes smoother and the **noise is effectively removed**, but at the same time, **edges and small details are well preserved**.

The **black details** in the image become more clearly separated from the background, which is a major improvement compared to the results of the Gaussian filter.

This shows the strength of the Bilateral filter when using a higher σ<sub>r</sub>: it can smooth the image while still maintaining important structural information.

Slide 3:

So far, we have kept **σ<sub>r</sub> constant**.

Now let’s consider the opposite case: we fix **σs = 3.0** and **increase σr**

As we increase **σr** the filter begins to smooth over **larger intensity differences** between pixels. This results in a **smoother image** with **better noise removal**.

However, if **σr becomes too large**, the filter will **start to blur edges and fine details**, which **defeats the purpose of the bilateral filter**, and the filter behaves more like a Gaussian filter — blurring everything uniformly, including both noise and edges.

Slide 4:

We now apply the same process to a different image.

First, I keep **sigma r constant** and vary **sigma s.**

This zoomed-in view shows that when **sigma r is small**, regions with low intensity differences are effectively smoothed and noise is reduced.

And here is the result when we **change sigma r**.  
The results are **similar to the previous example**.

Slide 21:

Here is a performance comparison table between my function and OpenCV's built-in function. The results show that my function performs significantly worse than OpenCV's, taking much more time to execute. A comparison of the two outputs (one from my function and one from OpenCV) is shown below: we can see that the results are quite similar.