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

opencv_world4110d.lib

C:\opencv\build\include

D:/FresherXavisTech/Image/

https://people.csail.mit.edu/sparis/bf_course/slides/03_definition_bf.pdf

https://www.csie.ntu.edu.tw/~cyy/courses/vfx/10spring/lectures/handouts/lec14_bilateral_4up.pdf

https://cybertron.cg.tu-berlin.de/eitz/bilateral_filtering/presentation_bilateral_filtering.pdf

https://www.cs.ubc.ca/~kmyi/teaching/cpsc425/slides/4_image_filtering_3.pdf

https://browncsci1290.github.io/webpage/labs/bilateral/

https://cybertron.cg.tu-berlin.de/eitz/bilateral_filtering/index.html

https://www.csie.ntu.edu.tw/~cyy/courses/vfx/21spring/lectures/handouts/lec14_bilateral.pdf

https://csundergrad.science.uoit.ca/courses/cv-notes/notebooks/13-bilateral-filtering.html

https://www.csie.ntu.edu.tw/~cyy/courses/vfx/10spring/lectures/handouts/lec14_bilateral_4up.pdf

https://cave.cs.columbia.edu/Statics/monographs/Image%20Processing%201%20FPCV-1-4.pdf

Slide 2

To reduce image noise, we often blur the image. One of the most common tools is the Gaussian kernel.
In this example, we can clearly see the effect of the Gaussian kernel in blurring and removing noise.

However, when we increase sigma — in order to enhance the blurring effect — the Gaussian filter also blurs the edges.
This causes the image to lose important details. We can clearly see this in right **Image**, where the edges are no longer sharp.

To overcome this drawback of Gaussian Blur, a more intelligent filter called the Bilateral Filter is used.

Slide 3:

The expression shown here is simply the convolution of an image with a spatial Gaussian — this is standard Gaussian smoothing.

Let's focus on the image patch in the right corner. It contains a step edge corrupted by noise. Our goal is to remove the noise while preserving the edge.

Now, imagine applying the Gaussian filter with its center at a pixel PPP. Consider two points: a yellow point and a green point. The yellow point lies on the same side of the edge as the center and is spatially close, so it receives a high weight — that makes sense.

However, the green point, which is also close to the center but lies on the opposite side of the edge, is given a similar weight. That's a problem — it doesn't make sense for a pixel across the edge to contribute just as much.

This is the core limitation of standard Gaussian smoothing: it considers only the spatial distance, not the image content. As a result, noise is reduced, but edges are blurred. You can see that clearly in the output image on the left — it's smoother, but the edge has disappeared.

We can fix this problem by adding another term called the brightness Gaussian, which takes the difference between the brightness of the center pixel and its neighbor.

If the brightness difference is small, the weight is high.

If the brightness difference is large, the weight is low.

By combining the spatial Gaussian with this range Gaussian, we create a new filter — the bilateral filter.

This final filter still looks like a Gaussian on the same side of the edge, but it gives lower weights to pixels on the opposite side. Importantly, this filter is adaptive — it changes depending on the image content.

The result of applying this filter to the noisy edge on the right is shown on the left - the noise has been reduced substantially while the edge has been preserved.

Slide 4:

This example demonstrates how the shape of the kernel depends on the image content.
The first row shows a constant area; the second row shows a step edge, as we discussed in the previous section;
and the third row shows an input with an arbitrary shape.

Slide 4:

Let's now look at the mathematical formulation of the bilateral filter.

p is the center pixel we want to filter, and q represents neighboring pixels within a local window Omega.

This filter combines two key components:

the spatial Gaussian, which depends on how far each pixel is from the center.

the Brightness Gaussian, which depends on how different in brightness the neighboring pixel is compared to the center.

Both of these Gaussian functions are controlled by sigma values, and these sigmas play a crucial role in how the filter behaves.

First, we have sigma s

This is the spatial sigma. It controls how far out the filter looks around the center pixel.

A large sigma means the filter considers pixels farther away, giving them more influence.

A small sigma means the filter focuses only on nearby pixels.

Next, we have sigma r, this is the range sigma. It controls how sensitive the filter is to intensity differences.

A large sigma means that even pixels with very different brightness still contribute — but that can blur edges.

A small sigma means only pixels with very similar brightness will affect the result — which helps to preserve edges.

So in summary:

- *Sigma s controls the spatial influence,*
- *sigma r controls the intensity similarity.*

By adjusting these two parameters, we can balance between smoothing noise and preserving edges, which is exactly what makes the bilateral filter so effective.

Wp is normalization factor. *Wp* needs to be recomputed for each pixel in the input image. This is done by simply taking the sum over the extent of the filter of the product of the brightness Gaussian and the spatial Gaussian.

Slide 5:

I've implemented a function to apply the bilateral filter to an input image.
This function takes in four parameters:

1. Input image – the image that we want to filter.
2. Kernel size – the size of the window used to consider neighboring pixels.
3. Sigma intensity – this corresponds to σr
4. Sigma space – this corresponds to σs

Slide 6:

My process consists of five main steps:

Step 1 is initialization and padding the input image. This ensures that edge pixels are properly handled when applying the filter window.

Step 2 is to precompute the spatial Gaussian kernel. This kernel assigns weights based on the distance of neighboring pixels from the center pixel. Since these distances remain constant, we compute this kernel only once and reuse it for all pixels.

Step 3 involves looping over every pixel in the image. For each pixel, we examine a surrounding window defined by the kernel. Within this window, we calculate the intensity difference between the center pixel and its neighbors. Using this difference, we compute the intensity Gaussian weight, which reflects how similar the pixel values are. We then multiply the spatial weight and the intensity weight to get the final bilateral weight for each neighbor. We use these weights to compute a weighted sum of pixel values and also accumulate the total weight.

Step 4 is to normalize the result at each pixel by dividing the weighted sum by the total weight. This gives us the final filtered value for that pixel.

Finally, Step 5 is to return the filtered image, which is smoothed while still preserving edges—one of the key advantages of the Bilateral Filter.

Slide 7: We will compare the result of the gaussian filter and the bilateral filter

On the left, we have the original image, In the middle, we see the result of applying a Gaussian filter with a kernel size of 25×25 and sigma_s = 3.0. As we can observe, the image becomes visibly blurred – noise is reduced, but so are the edges and important details. On the right, we have the image after applying a Bilateral filter with the same spatial sigma (sigma_s = 3.0), but with an additional parameter: sigma_r = 10.0, We can see that the noise has been reduced but edges and fine structures remain much sharper. To better illustrate the difference between Gaussian and Bilateral filtering, we can apply the Canny edge detection algorithm to each of the filtered images.

Slide 9:

When we increase the spatial sigma value in the Gaussian filter, the image becomes even more blurred, and more edges are lost as a result. However, the Bilateral filter is still able to preserve edges

Slide 10:

Let's consider what happens when we apply the bilateral filter using different values of σ_s while keeping σ_r constant. When we increase the value of σ_s, the filter gives more smoothing. This behavior is similar to what we observe with a Gaussian filter.

Slide 11:

When we keep the spatial sigma constant, and increase the brightness sigma.

even pixels with very different intensity values from the center pixel are allowed to contribute to the result.

This makes the filter act more like a standard Gaussian filter, where intensity differences are ignored — so edges start to get blurred, and the output becomes smoother overall.

Spatial — so, so far what you're saying here is just a convolution with a spatial Gaussian. Now, just ignore this factorization factor — we'll come back to this later. So now, in this case, if you want to filter this input image — which you can see is an edge, it's very noisy on both sides — with this spatial Gaussian, you can see that if you're going to place the Gaussian in this location right here, what's going to happen is that this point, the yellow point, is of course going to have a large weight, as it should, because it's close by. And that makes sense — because it's actually on the same side of the edge. However, the green point that you see here is going to be given the same weight, despite the fact that it's on the other side of the edge. That doesn't make sense — and that's exactly the problem you're trying to solve. In case you apply this filter — Gaussian smoothing — to this image, you end up with this. As expected, you get an image that is much smoother, with much less noise, but the edge has been completely blurred. So what we're going to do is add another term, which we'll call the **brightness** term. Right here, what's happening now is: you're going to add the brightness term, which acts on the brightness difference between the pixel you're	**Spatial** — cho đến giờ, những gì chúng ta đang làm ở đây chỉ là một phép **tích chập với bộ lọc Gaussian trong không gian**. Bây giờ, tạm thời bỏ qua yếu tố "phân tích nhân tử" này — chúng ta sẽ quay lại với nó sau. Trong ví dụ này, giả sử bạn muốn lọc ảnh đầu vào được hiển thị ở đây — bạn có thể thấy nó có một **cạnh (edge)**, và có rất nhiều nhiễu ở cả hai phía. Nếu chúng ta áp dụng một bộ lọc Gaussian không gian tiêu chuẩn, điều gì sẽ xảy ra? Giả sử bạn đặt bộ lọc Gaussian tại **điểm màu vàng** này trên ảnh. Tất nhiên, điểm màu vàng đó sẽ nhận được trọng số cao — vì nó gần với tâm. Điều đó hợp lý vì nó nằm **cùng phía với cạnh**. Tuy nhiên, điểm màu xanh lá mà bạn thấy ở đây cũng sẽ được gán **trọng số tương đương**, mặc dù nó nằm **phía bên kia của cạnh**. Điều này **không hợp lý** — và đó chính là vấn đề chúng ta muốn giải quyết. Nếu bạn áp dụng phép làm mịn Gaussian lên ảnh này, bạn sẽ nhận được kết quả như thế này. Như dự đoán, bạn sẽ có một ảnh **mượt hơn**, nhiễu ít hơn, **nhưng cạnh đã bị làm mờ hoàn toàn**. Vì vậy, những gì chúng ta sẽ làm là thêm một thành phần khác, gọi là **thành phần độ sáng (brightness)**.

operating on (which is at position **(i, j)**) and a neighboring pixel (say, at **(m, n)**). So if this is **(i, j)** and this is **(m, n)**, it's taking the difference between the brightness values of those two points and saying that if that brightness difference is small, then you're going to give it a large weight — because that's what a Gaussian does. If the brightness difference is large, you're going to give it a low weight — again, that's what the Gaussian does. So what happens here is that, in applying the spatial Gaussian, the weights for each pixel are modified in such a way that you end up with a **final filter** that looks like this: You can see that it is still a Gaussian-shaped filter on the side of the edge that this central point lies on. But it gets **literally truncated** on the other side — implying that all these pixels on that side are not going to contribute to the output being computed. It's mostly the pixels **on the same side** of the edge that will contribute. When you do that, you get a very clean result. The noise has been substantially diminished, and at the same time, the edge has been **preserved**. So that is **bilateral filtering**, and it has become a very common type of filter applied in many different domains. Why? Because it has this ability to **conform or adapt** the filter based on the content around a pixel.	Tại đây, điều gì đang xảy ra? Bạn sẽ thêm thành phần độ sáng, hoạt động dựa trên **sự khác biệt độ sáng** giữa pixel bạn đang xử lý (tại vị trí **(i, j)**) và các pixel lân cận (tại vị trí **(m, n)**). Vì vậy, nếu đây là điểm **(i, j)** và kia là **(m, n)**, thuật toán sẽ tính độ **chênh lệch độ sáng** giữa hai điểm đó và nói rằng: • Nếu độ chênh lệch **nhỏ**, bạn sẽ gán **trọng số cao** (vì Gaussian làm điều đó). • Nếu độ chênh lệch **lớn**, bạn sẽ gán **trọng số thấp** (cũng đúng với Gaussian). Kết quả là, khi áp dụng Gaussian không gian, các trọng số cho mỗi pixel sẽ được **điều chỉnh** theo cách mà bạn nhận được một **bộ lọc cuối cùng** như thế này: Bạn có thể thấy nó vẫn có hình dạng giống Gaussian ở phía cạnh mà điểm trung tâm đang nằm trên đó. Nhưng ở phía bên kia của cạnh, nó bị **cắt cụt hẳn** — cho thấy rằng **tất cả các điểm phía đó không đóng góp gì vào đầu ra** đang được tính toán. Chỉ có **các điểm cùng phía** mới thực sự góp phần. Khi bạn làm điều này, bạn sẽ có một kết quả rất sạch. Nhiễu đã được **giảm đáng kể**, và đồng thời, **cạnh được giữ nguyên**. Đó chính là **Bilateral Filtering**, và nó đã trở thành một kiểu lọc rất phổ biến được áp dụng trong nhiều lĩnh vực khác nhau.

So it is literally changing as you go — as a result of which, you **cannot** implement it as a convolution, because it's a **non-linear** and **spatially adaptive** filter. But the upside is that it gives you **very nice results**. Now let's take a look at this — this is something we set aside earlier, which is this **normalization function**, and it's very important. Because, irrespective of the shape — how complex the shape of the filter is — what you want to make sure of is that the **energy in the filter is always one**. So you want to change this weighting function as you go, and that's done by **summing up the product** of the brightness and the spatial Gaussian. In other words, you end up getting the weights that are used for all the pixels in the window, add them all up — and that's what you normalize with. So let's take a look at how this works on images. Once again, I should mention: **bilateral filter is a non-linear operator**. It simply cannot be implemented as a convolution. So here's an example. You see an original image, and this image has some noise in it. If you look closely, you see that the noise is green. We want to remove this noise **without losing the details** of the features.	Tại sao? Bởi vì nó có khả năng **tự điều chỉnh** bộ lọc dựa trên **nội dung xung quanh** mỗi pixel. Nó **liên tục thay đổi theo từng vị trí** — và vì thế, bạn **không thể** cài đặt nó dưới dạng một phép **tích chập tuyến tính**. Đây là một bộ lọc **phi tuyến**, **thích ứng không gian**. Nhưng điểm mạnh của nó là — **nó cho kết quả rất tốt**. Bây giờ chúng ta hãy xem xét điều này — là phần mà chúng ta tạm gác lại trước đó — đó là **hàm chuẩn hóa**, và nó **rất quan trọng**. Bởi vì, **bất kể hình dạng của bộ lọc phức tạp ra sao**, bạn vẫn muốn đảm bảo rằng **tổng năng lượng của bộ lọc luôn bằng 1**. Bạn muốn điều chỉnh **hàm trọng số** khi di chuyển, và điều này được thực hiện bằng cách **cộng tổng tích của trọng số không gian và trọng số độ sáng**. Nói cách khác, bạn lấy tất cả các trọng số được áp dụng cho các pixel trong cửa sổ lọc, **cộng lại**, và đó sẽ là giá trị dùng để **chuẩn hóa** đầu ra. Giờ chúng ta hãy xem cách nó hoạt động trên ảnh thực tế. Một lần nữa, mình muốn nhấn mạnh rằng: **Bilateral Filter là một toán tử phi tuyến**. Nó **không thể** được cài đặt như một phép tích chập. Dưới đây là một ví dụ: bạn thấy một ảnh gốc — ảnh này có một chút nhiễu. Nếu bạn nhìn kỹ, bạn sẽ thấy **nhiễu màu xanh lá**.

If you apply a **Gaussian filter** — simple smoothing with a spatial sigma of 2 — you see that you get a slightly blurry image. The noise hasn't been entirely removed. But if you use a **bilateral filter** with the **same spatial sigma** of 2, and a **brightness sigma** of 10, you get a very nice image where almost all the noise has been removed, and at the same time, most of the spatial features have been **preserved**. If you look at this area here — yes, you do lose a little bit of information. But by and large — especially in the case of the eyes and the mouth — all the **salient features** are preserved, and you get a much better output. Now, what happens when you change these two sigmas? So if you increase the **spatial sigma** to 4, you get a much blurrier image in the case of Gaussian smoothing. You still get a pretty sharp image in the case of bilateral filtering. Again, a little more detail is removed, but the result is still fairly acceptable. If you increase it further, you start getting this **painterly effect**, where shaded regions start getting flatter and flatter. This is the **"watercolor" look**, and you can especially see that in the hair region. But still — this is a much nicer output than this one right here, which is the result of just Gaussian smoothing.	Chúng ta muốn **loại bỏ nhiễu**, **nhưng vẫn giữ lại chi tiết** của các đặc trưng. Nếu bạn áp dụng **bộ lọc Gaussian** — làm mịn đơn giản với **sigma không gian = 2**, bạn sẽ được một ảnh hơi bị mờ. Nhiễu vẫn chưa được loại bỏ hoàn toàn. Nhưng nếu bạn dùng **Bilateral Filter** với **sigma không gian = 2** (giống như Gaussian) và **sigma độ sáng = 10**, bạn sẽ có một ảnh **rất đẹp** — gần như **tất cả nhiễu đã bị loại bỏ**, và đồng thời, **hầu hết các đặc trưng không gian được giữ nguyên**. Nếu bạn nhìn vào vùng này — đúng là có mất một chút thông tin, **nhưng nhìn chung**, đặc biệt là ở **mắt và miệng**, các **đặc trưng nổi bật** vẫn được giữ lại. Kết quả rất tốt. Bây giờ, chuyện gì xảy ra khi bạn thay đổi hai giá trị sigma này? Nếu bạn tăng **sigma không gian** lên 4, ảnh Gaussian sẽ bị mờ hơn nhiều. Nhưng ảnh từ Bilateral Filter vẫn khá sắc nét. Đúng là một chút chi tiết bị mất, nhưng kết quả vẫn **chấp nhận được**. Nếu bạn tăng tiếp nữa, bạn bắt đầu có hiệu ứng giống như **tranh vẽ màu nước** — các vùng sáng tối bắt đầu trở nên phẳng hơn. Bạn có thể thấy rõ hiệu ứng đó ở **vùng tóc**. Nhưng ngay cả khi đó, kết quả vẫn **tốt hơn nhiều** so với ảnh chỉ dùng Gaussian smoothing. Bây giờ chúng ta hỏi: chuyện gì xảy ra khi bạn thay đổi **sigma độ sáng**?

Now we can ask — what happens when you change the **brightness sigma**? So far we've held the brightness sigma at 10 all the way through. Now, suppose we keep the **spatial sigma** at 6, and increase the **brightness sigma** to 20. You get this result here. If you increase the brightness sigma further and further — let's say use a really large number — then the brightness Gaussian becomes essentially **flat**, so it has **no effect**. It's essentially saying: the weight is one **irrespective** of the difference in brightness between the center pixel and the neighbor pixel. And at that point, your **bilateral filter begins to behave like a Gaussian** — as expected.	Từ nãy giờ chúng ta đã giữ **sigma độ sáng = 10**. Giả sử bây giờ bạn giữ **sigma không gian = 6**, và tăng **sigma độ sáng = 20**, bạn sẽ được kết quả như thế này. Nếu bạn tiếp tục tăng sigma độ sáng lên — dùng một giá trị rất lớn — thì **Gaussian độ sáng trở nên bằng phẳng** và **không còn tác dụng gì nữa**. Lúc đó, nó giống như nói rằng: **trọng số luôn là 1**, bất kể độ sáng giữa hai điểm khác nhau thế nào. Và khi đó, **Bilateral Filter bắt đầu hành xử giống hệt Gaussian Filter** — đúng như bạn kỳ vọng.