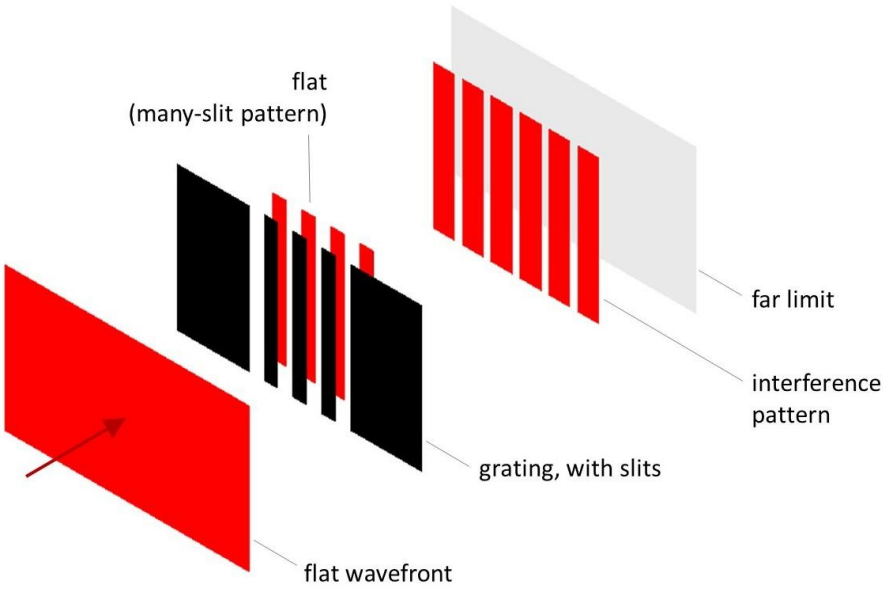
Lecture Notes on Symmetry Optics

Lecture 2:
Plenary and Singular Factors

To accompany https://youtu.be/DMfN_PrR0tw
v5, April 13, 2022
Paul Mirsky
paulmirsky633@gmail.com

1 Introduction to Many-Slit Interference

1.1 Overview of MSI



Welcome to Lectures on Symmetry Optics. I'm Paul Mirsky. This is Lecture 2 of the series, and the topic of this lecture is: Plenary and Singular Factors.

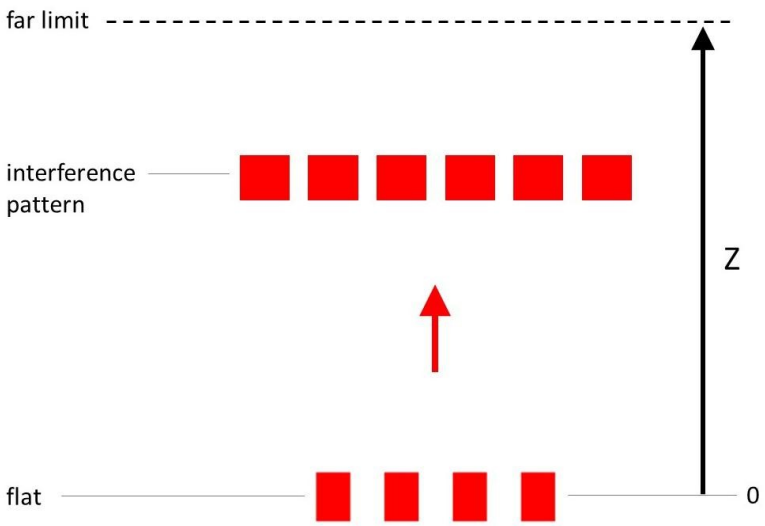
One of the most important phenomena in symmetry optics is *many-slit interference*. Let's review. Imagine a flat wavefront of light, all at one wavelength. The light passes through a grating, which is an opaque screen containing many slits. Some light gets absorbed by the grating, but some passes through and forms a many-slit pattern.

We actually won't be so concerned about how this pattern gets formed. What matters is that now we have a pattern of stripes, and the wavefronts all lie in one plane, which is why we call this plane the *flat*. These are our 'initial conditions'. In other words, when we do calculations, we are usually given this pattern. And the purpose of the calculations is to determine what happens next.

Because, as the light propagates forward, the pattern doesn't just stay the same. Instead, it diffracts and interferes, and it forms a *diffraction pattern*, which can also be called an *interference pattern*. That pattern is what we want to calculate.

Finally, we have the *far limit*. This is a plane, and it's not necessarily a physical boundary, but it's a mathematical boundary to our model. Our model only goes up to the far limit, although we're allowed to set it as far back as we like.

1.2 Planes at every value of Z



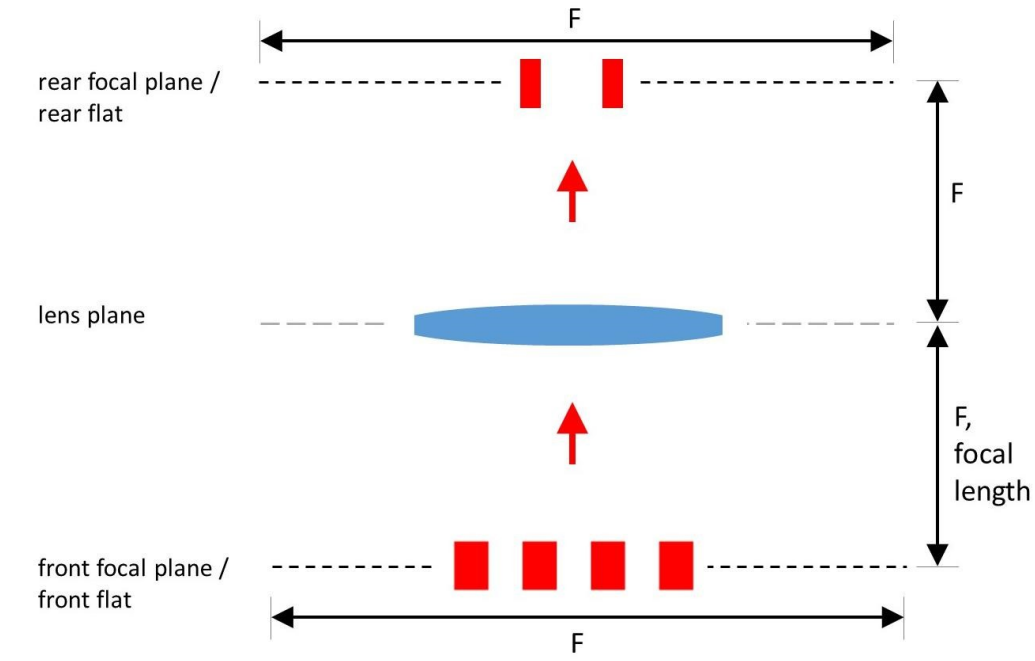
We're going to shift perspective now; now we're looking down from above.

This (arrow) is the direction of propagation. We label this axis with the letter Z. At Z=0, we have the flat. And for every point along Z, there is another plane lying perpendicular, and each of these planes has some pattern in it.

Now, this drawing is not perfectly realistic, because it shows the patterns as though they have some thickness along Z. That's not actually true; the pattern actually lies in just one plane, but we draw it this way so that you can see it.

What we're looking at is called the *free configuration*. It's one of two main configurations.

1.3 The lens-limited configuration



The other is the *lens-limited configuration*.

First, let's review some basics about lenses. Every lens has a focal length, and we'll call it *F*. F is the parameter that makes it a strong or a weak lens. And it defines two planes: the *front focal plane* which lies a distance F in front of the lens, and the *rear focal plane* which lies a distance F behind the lens.

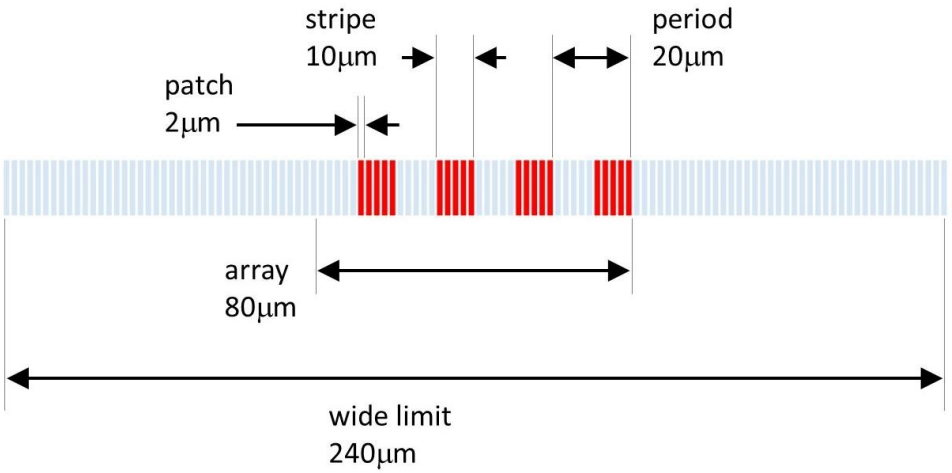
Now let's assume that the flat is lying in the front focal plane. The light propagates and becomes curved, and passes through the lens, and eventually the wavefronts become flat again at the rear focal plane. We call the initial plane the *front flat*, and the final plane the *rear flat*.

You might think that the rear flat is the far limit, but actually for mathematical purposes, the lens plane acts as the far limit. So, the far limit is equal to F, the focal length.

The most key principle of the lens-limited configuration is: The pattern at the rear flat is the *Fourier transform* of the pattern at the front flat. This one fact is the basis for the whole subfield of Fourier optics, which is related to symmetry optics. This lecture will not discuss the Fourier transform in detail, but the main thing to know is that it's *not* simply an image of the front flat. Instead, it's a pattern that typically looks quite different.

2 Features Composed From Features

2.1 Features and widths at the flat



Now let's take a close look at the pattern at the flat. We're going to illustrate by looking at this one specific example. To begin with, we see a pattern with red areas and light blue areas. The red indicates bright space, where light is present. The blue indicates dark space, where there is no light. Now, there are 5 different *features* which define the pattern, and each one of them parameterized with a width.

The first feature is the *patch*. The pattern is made up of discrete patches of space, all of them identical in size, almost like individual pixels. Its width is the wavelength. In this example, it's two micrometers (µm), also called two microns.

The next feature is the *stripe*. This is the bright area drawn in red; right after the grating, this is the light from one slit. All instances are identical, so the stripe width is a single parameter. Here, it's 10µm. Notice here that sometimes we are a little bit loose with our terminology, and for example, instead of saying 'the stripe *width* is 10µm', we may just say 'the stripe is 10µm' which is unambiguous enough.

The next feature is the *period*. This is the stripe *plus* the dark space adjacent to it. The period is the distance from one stripe to the next. Again, they are all evenly spaced so it's a single parameter, which in this example is 20µm. But you should note that it's not always two times the stripe – that's just this particular case.

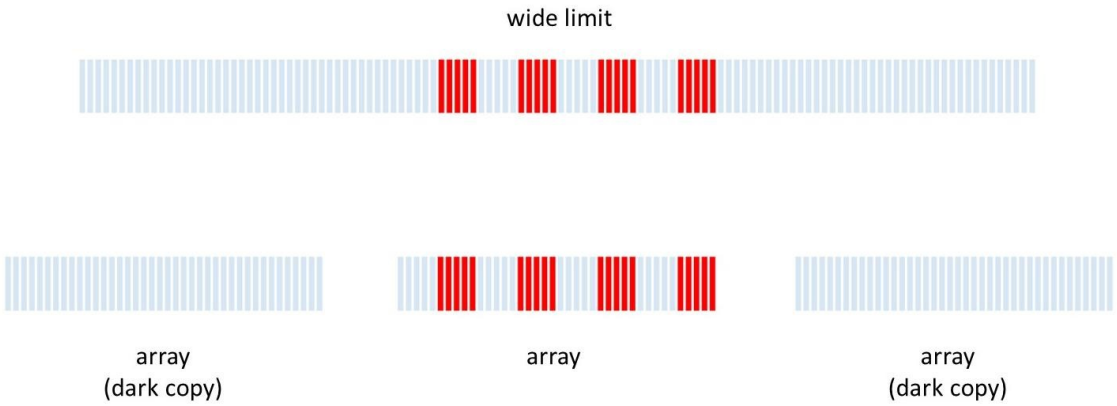
The next feature is the *array*. This is the whole region of the pattern that contains any bright space, even though that bright space isn't contiguous. And note where the exact edges of the array are. The array is always made up of increments of a full period, so the right edge is the edge of this stripe, but on the left edge, it's not the edge of the stripe, but rather the edge of the period, so it includes this dark space. The width of the array here is 80µm.

And finally, the last feature is the *wide limit*. This sets the full size of the pattern. So the pattern isn't just the array; it also includes dark space. The wide limit has the same value as the far limit, but it's measured along a different axis – the far limit is the distance in Z between two planes, but the wide limit is a distance within a single plane. So in this case, the wide limit is 240 µm, implying that the far limit is also 240 µm.

Now, we've been discussing this one particular pattern, but this is just one member of a large family of possible patterns, and what we've said applies to all of them. Every member of this family has a patch, a stripe, a period, an array, and a wide limit. The only difference is the *widths* that these features have, which can be set more-or-less arbitrarily.

But widths are only one way of parameterizing this pattern. We now want to shift to a different set of parameters.

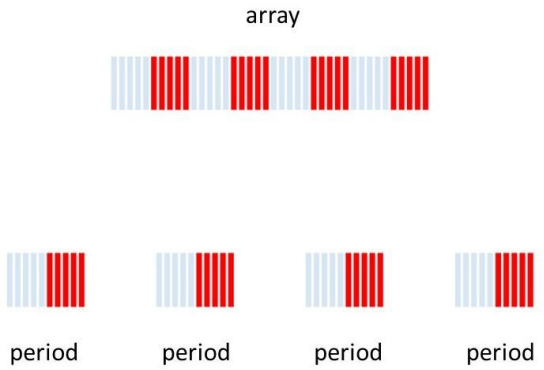
2.2 Wide limit composed from bright and dark arrays



First, it's helpful to notice that each feature can be *composed* from smaller features, and they form a nested series. Let's look at this:

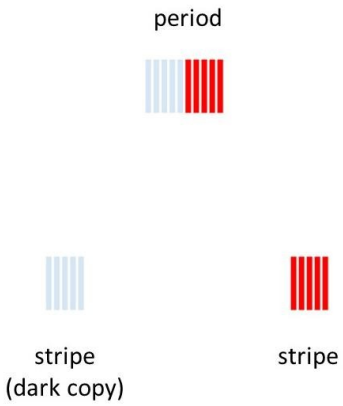
We'll start at the top with the wide limit. We can think of it as being composed of 3 distinct segments. The segment in the middle is unique, because it's the normal array. All the other segments are what we can call *dark copies* of the array, because they are the same size as the array, but all the patches are dark. So at this level of nesting, the wide limit is the whole and the array is the component.

2.3 Array composed from periods



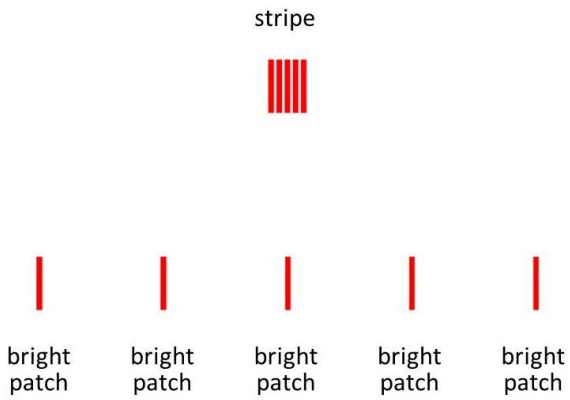
But, in the next step we zoom in and look at the components of the array itself. Similar to the last step, the array is composed from 4 copies of the period. But the logic is a little different here, because there are no dark copies. Instead, they're all normal copies.

2.4 Period composed from bright and dark stripes



We can continue zooming in and look at the period. It's made up of 2 segments: the normal stripe, and a dark copy. This is like the first step, because there is only one normal component, and the rest are dark copies.

2.5 Stripe composed from bright patches



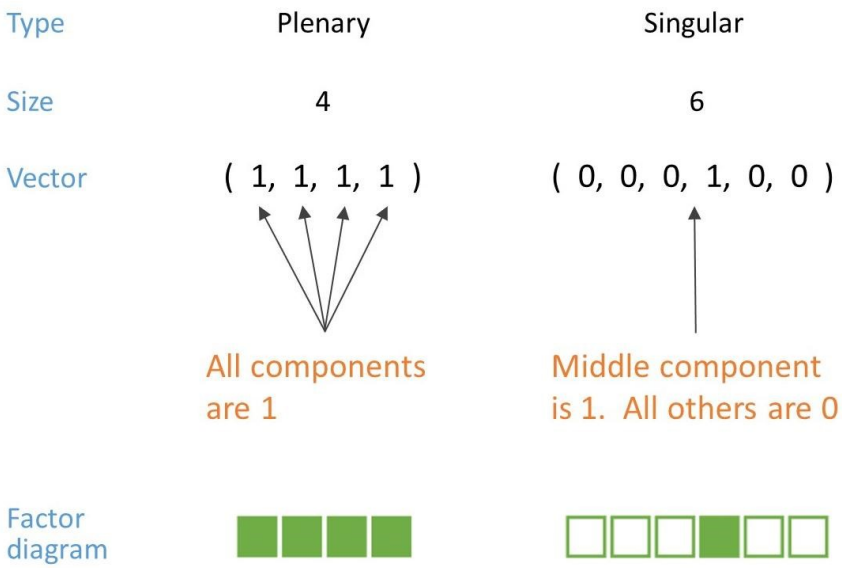
And finally, we examine the stripe. It's composed of 5 normal copies of a bright patch.

Now we are down to a single wavelength, and this is as far as we can go. This gives you a good picture of the nested structure of the pattern.

Now, we're going to formalize it and clarify it, by introducing the concept of *factors*. We're going to shift away from thinking in terms of features and widths, we're going to shift towards thinking in terms of factors. Factors are a new concept that is unique to symmetry optics.

3 Features and Factors

3.1 Plenary and singular factors



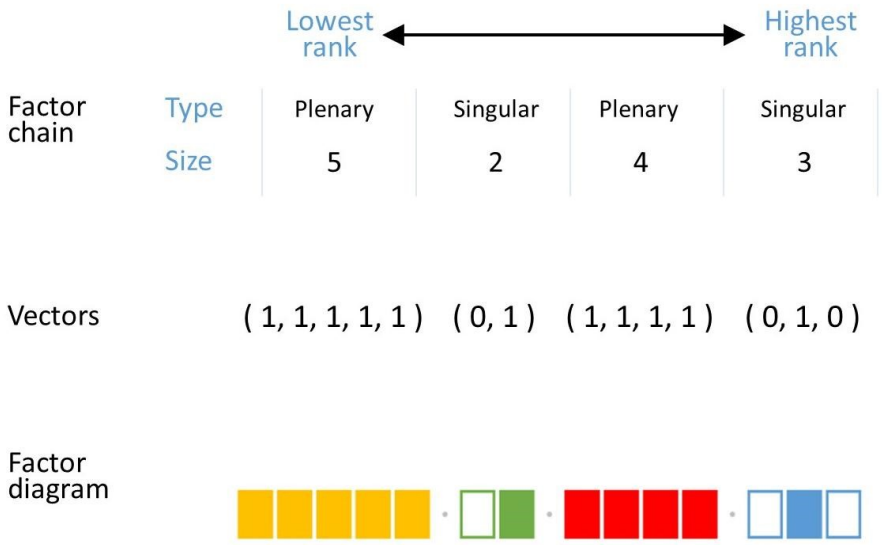
A factor is represented mathematically by a vector. Here are two examples: On the left, we have the vector 1-1-1-1. On the right, we have the vector 0-0-0-1-0-0. Every factor has a *size* and a *type*.

The size is the number of components in the vector. So, the vector on the left is size 4, and the vector on the right is size 6.

The Type can be either *plenary*, or *singular*. In a plenary factor, all of the vector components are equal to 1, like the vector on the left: 1, 1, 1, 1. In a singular factor, a single one of the components is equal to 1, and all the other components are zero. By convention, we put the 1 in the middle, or if the size is an even number then we'll put it just to the right of the middle. For now, we'll only consider factors that fit one of these two formulas. There do exist other variations of singular and plenary factors, but those won't come until much later on.

Alternatively, we can represent factors using *factor diagrams*, because they are convenient and easy to interpret. This is really just a way to draw the vector. A factor diagram is drawn as set of boxes; each box can be either open or filled. An open box represents a 0; a filled box represents a 1. So, the vector 1-1-1-1 is drawn as four filled boxes. 0-0-0-1-0-0 is drawn as open box, open box, open box, filled box, open box, open box. Easy.

3.2 Combining factors into a ranked series or chain

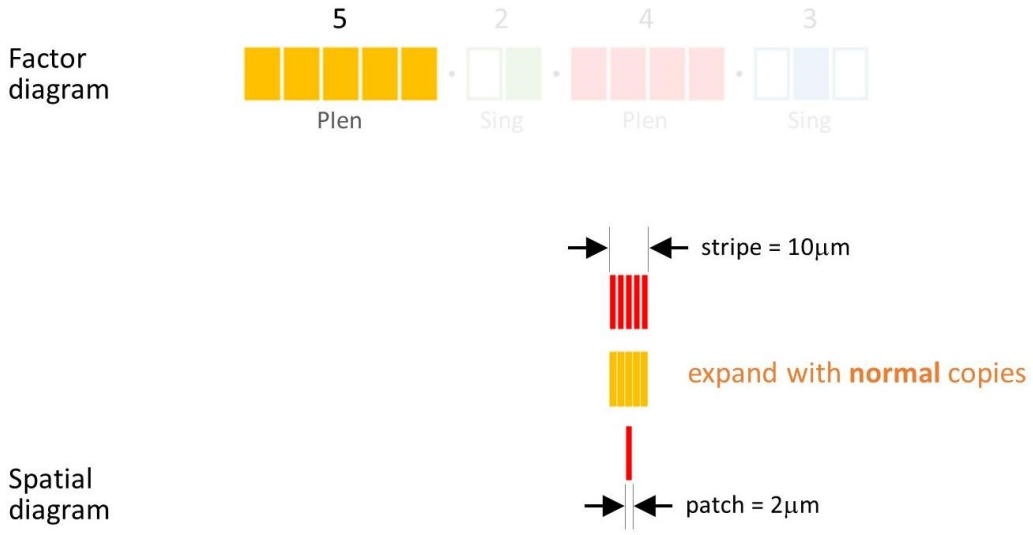


Now, to create the pattern we saw earlier, we need to combine 4 factors into a *factor chain*. In this table, each column is one link of the chain, and they are in a sequence from lowest rank to highest rank. In this example, the lowest-ranking factor is size-5, plenary. The next-lowest is size-2, singular, then size-4, plenary, and the highest-ranking factor is size-3, singular. For a many-slit pattern like our example, it always goes plenary-singular-plenary-singular.

We can express this factor chain equivalently with a chain of factor vectors, or with a factor diagram. The lowest-ranking factor, the size-5 plenary, is 1-1-1-1-1, or 5 filled boxes. The size-2 singular is 0-1, or open-box filled-box. The size-4 plenary is 1-1-1-1 or 4 filled boxes. And the size-3 singular is 0-1-0, or open-filled-open.

A minute ago, we broke the pattern down step by step; now we'll learn an algorithm to build the pattern up, step by step. We can think of the factor chain as a recipe for doing this building-up.

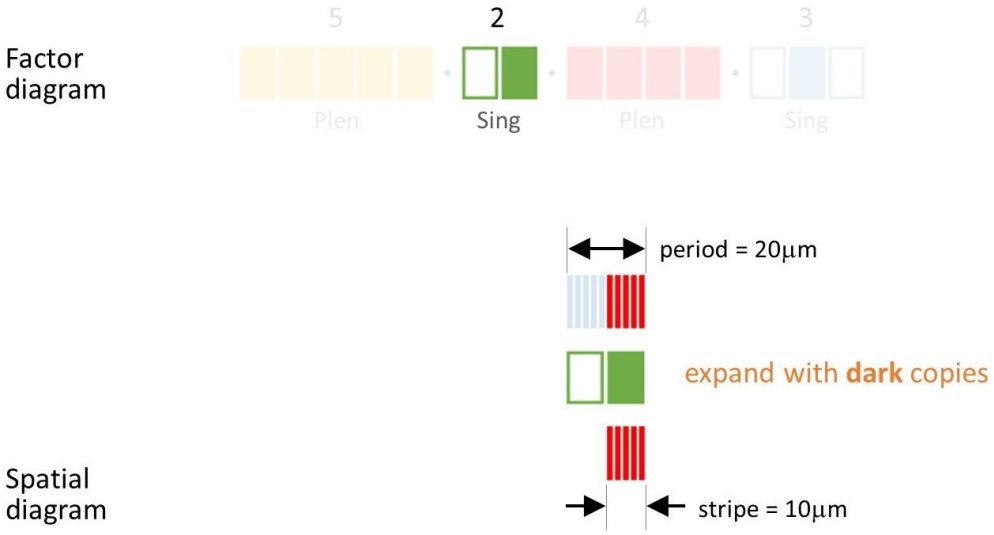
3.3 Factor 1 / 4 (lowest-ranking)



We always start with a single bright patch, which is the simplest possible pattern.

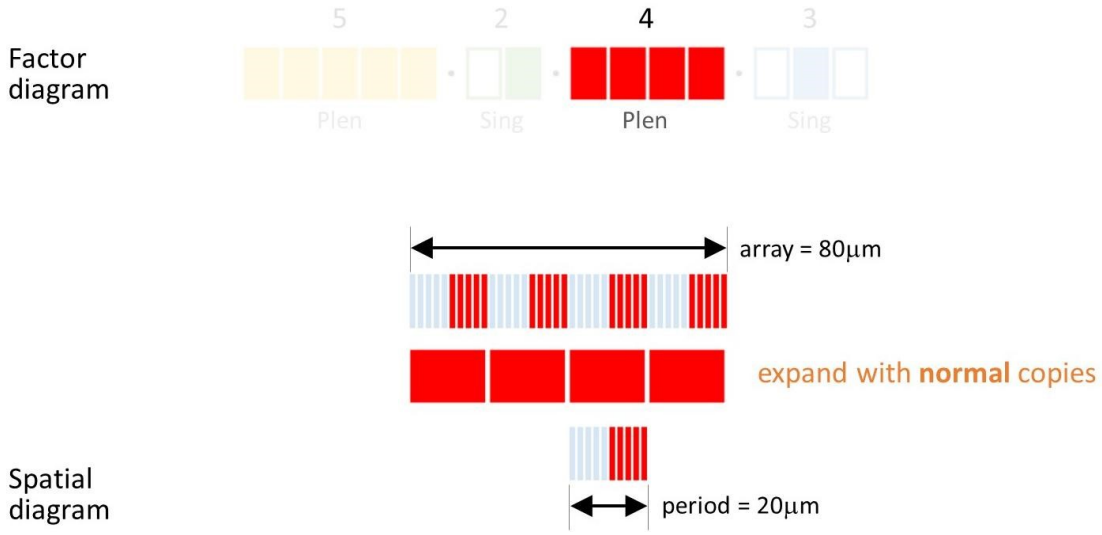
Now let's take the lowest-ranking factor: size-5 plenary. Every factor expands a smaller feature into a larger feature. The factor size is the multiple that we expand by – so, here, we expand the patch 5-fold. The pattern type tells us how to expand: if it's a plenary factor, like in this case, we expand by adding normal copies. This gives us 5 bright patches in a row, which is the stripe.

3.4 Factor 2 / 4



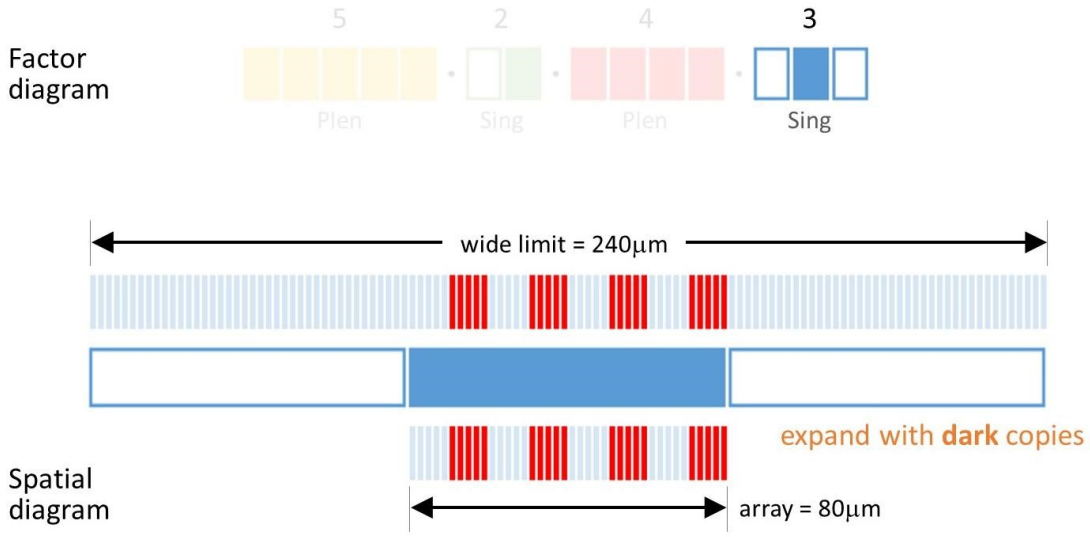
Next up, we have a size-2 singular factor. Like before, we expand the stripe – this time, 2-fold But this time it's a singular factor, so instead of adding normal copies, we add dark copies, which in this case is just one. This gives us the period.

3.5 Factor 3 / 4



Next comes a size-4 plenary factor, so we expand the period 4-fold, adding normal copies. This gives us the array.

3.6 Factor 4 / 4 (highest-ranking)



And finally, we have a size-3 singular factor, so we expand the array 3-fold, adding dark copies. This gives us the wide limit, which is the full pattern.

We're done!

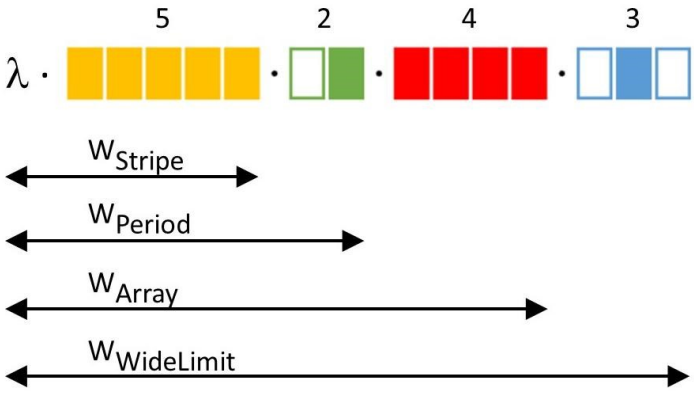
3.7 Feature widths vs factors

$$\frac{W_{Stripe}}{\lambda} = 5$$

$$\frac{W_{Period}}{W_{Stripe}} = 2$$

$$\frac{W_{Array}}{W_{Period}} = 4$$

$$\frac{W_{Limit}}{W_{Array}} = 3$$



But let's clarify a very confusing point. There are factors and there are widths (features), but they are *not* the same thing, so be careful not to mix them up.

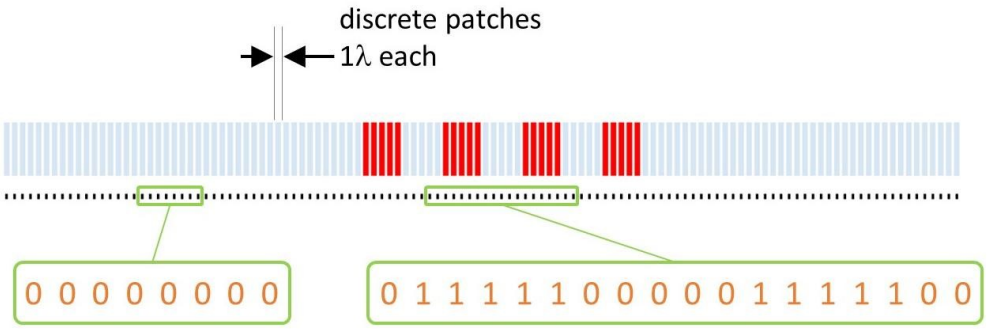
The difference is that the widths are *cumulative products* of many links of the chain, then multiplied by the wavelength. For example, the stripe is the lowest-ranking factor size, times the wavelength. The period is the product of the *two* lowest factor sizes, times the wavelength. The array is the product of the *three* lowest factor sizes, times the wavelength. And the wide limit is the product of all *four* factor sizes, times the wavelength. The widths all carry length units, like microns or meters. But the factor sizes are dimensionless.

We can put this a different way: the factor sizes are *ratios* between two different widths. For instance, the lowest-ranking factor is the ratio of the stripe to the wavelength. The second-lowest factor is the ratio of the period to the stripe. The second-highest factor is the ratio of the array to the period. And the highest factor is the ratio of the wide limit to the array.

These formulas on the left are going to let us 'cheat' a little bit in some of our calculations. What I mean is that formally, every factor size is supposed to be an integer. That's because it's the number of components in a vector, and a vector can have 4 components (for example) or 5 components, but it can't have 4½ components. But there is some flexibility. If you don't actually need to build up a pattern, and you just need to calculate a size or a width, then it's OK to use non-integer factor sizes. Use your judgment.

4 Factors as Vectors

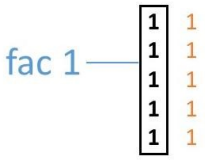
4.1 Discrete patches, binary intensity



We've just learned one algorithm for calculating patterns, but there's also an alternative algorithm which uses the factor *vectors*. Both give the same answers. The difference is that the factor-vector algorithm is less intuitive, but has a clearer mathematical structure.

We'll start by noting that we can represent the pattern with a vector. It has one component for every patch of the pattern, and the value of each component represents the amplitude of the light wave at that patch. Every value is either a 0 or a 1. Zero means that it's a dark patch. 1 means that it's a bright patch.

4.2 Outer product of factor vectors, Step 1 / 4

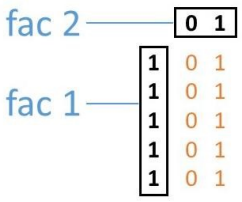


And the way we calculate this vector is to:

1. Form the *outer product* of the factor vectors
2. *Flatten* the outer product, which means to rearrange it into a 1-dimensional vector.

Let's look at how it works. To form the outer product, we introduce the factor vectors 1 at a time. Here's the first one, and with just one factor there's not much to say.

4.3 Outer product of factor vectors, Step 2 / 4

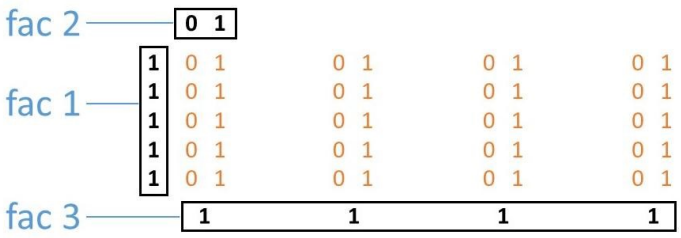


But when we combine it with the second factor, we form a table or a 2-dimensional array. The 1st factor corresponds to the *rows* of the array, and the 2nd factor corresponds to the *columns*. Each element of the array is the *product* of one row component from the 1st factor, and one column component from the 2nd factor. For example, this element (2,1) is in the 2nd row, which corresponds to a 1. It's also in the 1st column, which corresponds to a 0. 1 x 0 = 0, so this element is equal to 0.

This element (5,2) is in the 5th row, and the 2nd column, so its factors are 1 and 1, so the element is equal to 1 x 1 which is 1.

This is how the outer product works for two factor vectors, but we can keep going by adding a third factor.

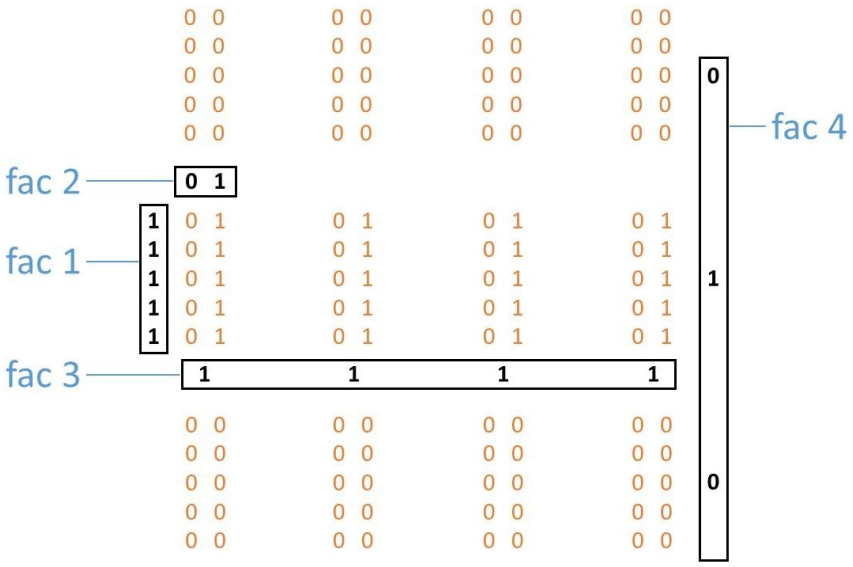
4.4 Outer product of factor vectors, Step 3 / 4



This gives us a 3-dimensional array; and, while it's impractical to drawn them in 3-d, you can picture these as 4 levels stacked on top of one another.

Any element of this array is the product of 3 numbers – the row label, the column label, and the level label. For example, this element (4,1,3) is in the 4th row, the 1st column, and the 3rd level. Its factors are 1, 0, and 1. The product of those three is zero. Basically, if all the factors are 1, then the element is 1. If any of the factors is 0, then the element is 0.

4.5 Outer product of factor vectors, Step 4 / 4



Finally, we add the 4th factor, making a 4-d array. The new factor is singular, and so all of the new entries are zeros. Now we've used up all of our factor vectors, and so the outer product is complete.

The next step is to flatten the array, which means to turn it from a 4-dimensional array into a 1-dimensional array, which is a vector. To do this, we need to put all the elements in order, following the rule that the lower-ranking indices get cycled first. Here's how this works:

4.6 Flattening the outer product

We start with the first component of each factor. This corresponds to the element in one corner of the array. This is going to be the first element of our 1-d vector.

We then increment the index of factor 1, going to components 2, 3, 4, and 5.

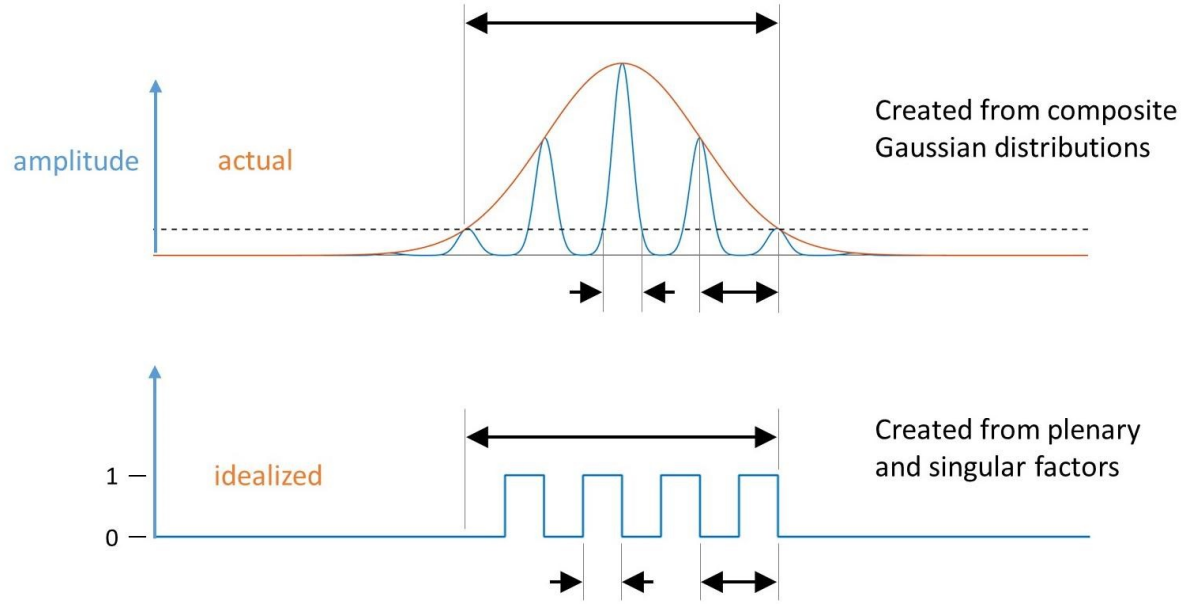
But once we get to the end of factor 1, we cycle it back to the beginning and then increment factor 2. It's a little like counting in base-10 numbers. There, you count in the 1's column from 0 to 9, but then you roll back to 0 and increment the 10's column. The main difference is that in base-10 numbers it always cycles back after 9, but in this case each factor has a different size and cycles back at a different index.

Also, don't confuse the component *indices* with the component *values*. For example, factor 1 has 5 components and their *indices* go 1-2-3-4-5, but the *values* go 1-1-1-1-1. It's the *indices* that we're incrementing, not the values.

So back to flattening, we continue incrementing factor 1. But when we get to the last component, we find that factor 2 is also at its last component, so we cycle both 1 and 2 back to the beginning, and then increment factor 3.

We won't follow every single step, but this continues for all of factor 3 and then likewise for factor 4, until we get to the last component of all factors, which is the last element of our 1-d vector. Now we're done, and we have one long vector which represents our pattern.

4.7 An idealized model



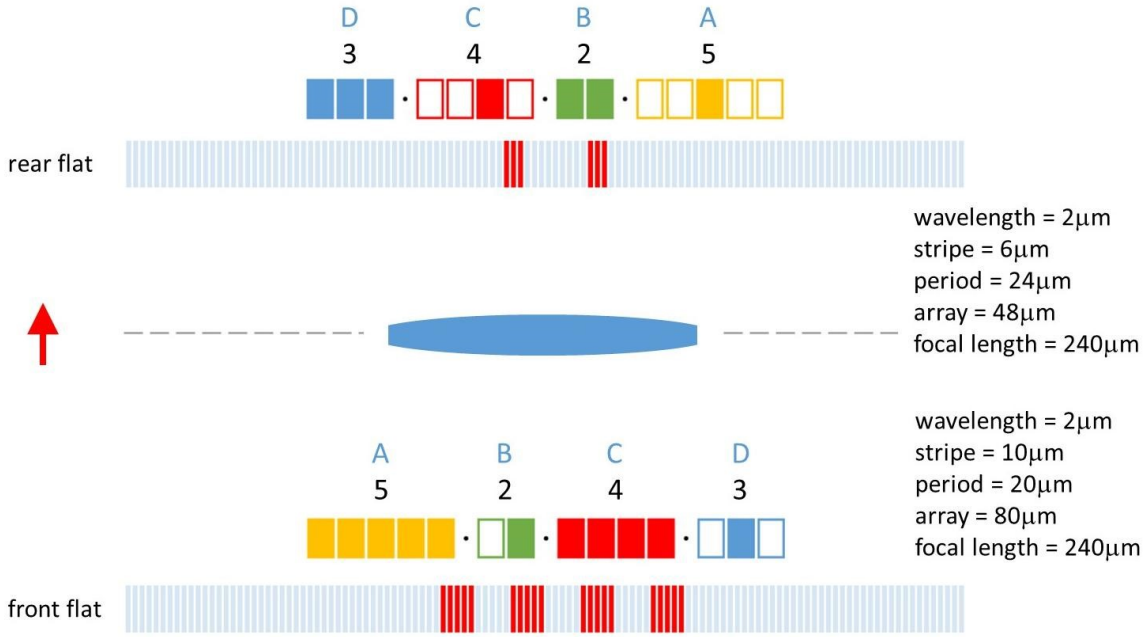
One important note: because every value is either 0 or 1, every patch is either dark, or uniformly bright; there are no intermediate shades of grey.

This brings us to an important caveat: symmetry optics is an *idealized* model. It's meant to be simple, rather than to get every detail right. For example, imagine that we wanted to represent a pattern like the one at the top, which is made from Gaussian distributions. This is fairly typical of what you'd actually measure in the lab. Today, symmetry optics can't model a pattern like this exactly. Instead, we would idealize, and model it like the pattern on the bottom, with only the two amplitudes 0 and 1. It gets the basic dimensions right, and it captures the essence of the pattern, but it does miss some subtleties.

The hope is that in the future, symmetry optics will get more advanced and develop into a more-general model that can handle all possible patterns. But, that's a long way off.

5 Simplest Application in Symmetry Optics

5.1 Fourier transform in MSI



Now we are going to learn the simplest application in symmetry optics. Note that we are working in the lens-limited configuration, and that therefore we use the focal length as the wide limit.

We're given a pattern at the front flat, and our goal is to calculate the pattern at the rear flat.

Now, recall from before that the rear flat is the Fourier transform of the front flat. There's a very standard equation for calculating the Fourier transform, but we won't be using that. Instead, we'll begin by determining the factor chain which defines the pattern, which in this case we already know. We'll adopt the convention of referring to the four factors in this chain as A, B, C, and D, from lowest to highest rank.

Next, we apply a simple rule. To find the rear chain from the front chain:

1. Reverse the rank order
2. Invert each type; in other words, change each plenary into a singular, and change each singular into a plenary.

Where does this rule come from? Well, if you're familiar with the Fast Fourier Transform algorithm, you can maybe hear echoes of it. But this rule doesn't rest on any derivation.

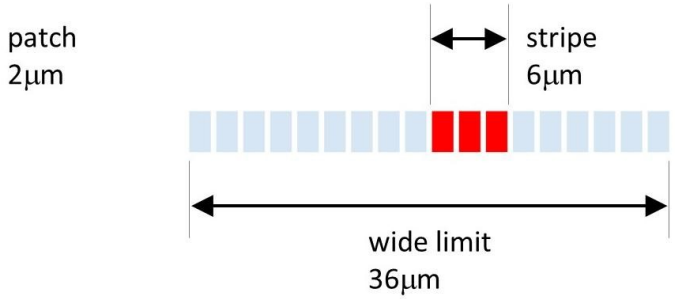
Rather, we're going to simply posit the rule and use it because it works. In later lectures, we'll see that actually it's one special case of a much more general rule, but that's for later.

For now, let's see how the rear flat looks in this example. Up at the top is our rear symmetry chain. Firstly, it has reversed rank order. So, the front chain goes A-B-C-D, but the rear chain goes D-C-B-A. Next, each type is inverted. So, A goes from plenary to singular, B goes from singular to plenary. C goes from plenary to singular, and D goes from singular to plenary. Notice, though, that in one sense those two changes cancel each other out, and so in both cases the chain goes plenary-singular-plenary-singular.

Once we have the factor chain at the rear, we calculate the rear pattern. This is our answer.

Let's compare the front and rear patterns and see how our rule manifests in spatial terms: First of all, note that the wavelength and the focal length are always the same in both planes. Now, at the front, there are 5 patches in the stripe. In the rear, the wide limit is 5 times as wide as the array. At the front, the period is 2 times as wide as the stripe. In the rear, there are 2 periods. At the front, there are 4 periods. At the rear, the period is 4 times as wide as the stripe. Lastly, at the front, the focal length is 3 times as wide as the array. At the rear, there are 3 patches in the stripe.

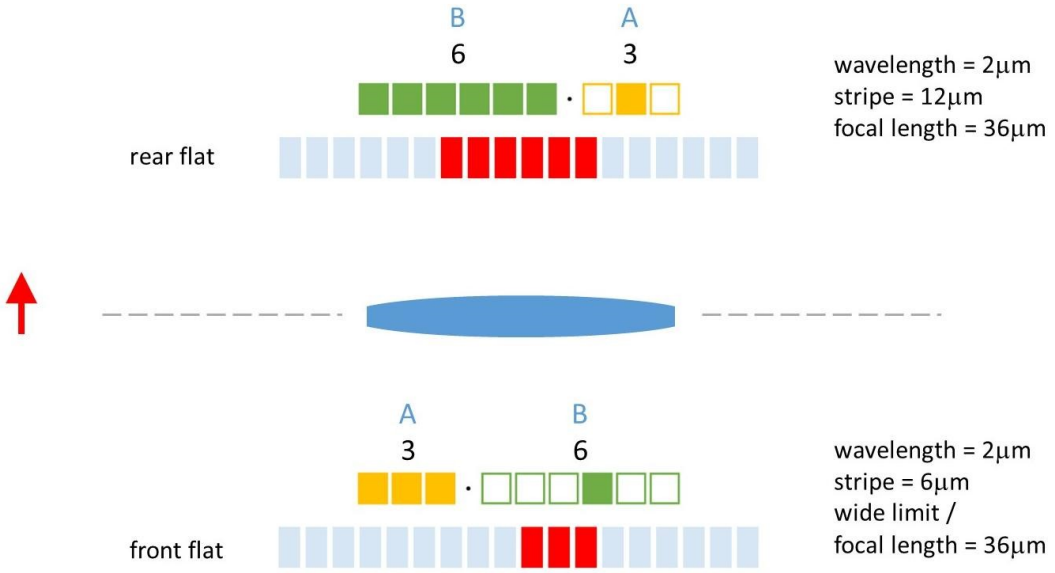
5.2 Features of the beam



Besides many-slit interference, we're only going to discuss one other phenomenon: the *beam*. This is actually even simpler than many-slit interference. You can think of it as just one slit – or, alternatively, it's just a wavefront that never even passes through a grating at all. It's actually so simple that it's hard to learn the principles from it, which is why we're doing it last rather than first.

In any case – the beam has fewer features: just the patch, the stripe, and the wide limit. The stripe is just the diameter of the beam.

5.3 Fourier transform in the beam



In the lens-limited configuration, again we're given the feature widths at the front flat. Just as before, we express the pattern in terms of a factor chain. We follow the same principles as we did for many-slit interference, but for the beam there are only two factors in the chain.

In this example, the chain consists of a size-3 plenary factor, and then a size-6 singular factor, going lowest to highest rank. In other words, we start with a single bright patch, and we expand it 3-fold with normal copies to make the stripe. Then, we expand the stripe 6-fold using dark copies. That gives us the wide limit, which is the focal length.

As before, we reverse the rank order of the factor chain, and A-B in the front is changed to B-A in the rear. Also, A goes from plenary to singular, and B goes from singular to plenary. Also, those two effects cancel, and both chains go plenary-singular.

We then convert the factor chain back into a pattern, and here's what we get. Instead of 3 patches in the stripe, we have 3 stripes in the focal length. And, instead of 6 stripes in the focal length, we have 6 patches in the stripe. And that's our answer.

6 Conclusions

6.1 Reviewing key points

That's all for this lecture, so let's review the key points:

- A plenary factor is a vector of all 1s.
- A singular factor is a vector with a single 1 in the middle, and 0s everywhere else.
- Widths are cumulative products of factors. Factors are ratios of width.
- The pattern is the outer product of the factors, flattened
- The rear flat (Fourier Transform) is made by reversing rank order, and inverting type (singular/plenary).

6.2 Outro

There are more resources to help you learn symmetry optics, and I hope you'll take advantage. You'll find links wherever you found this video.

The first resource is the lecture notes. This is essentially the same material as this lecture, but in written form. Sometimes, it's better to read it than to hear it.

Second, there is a problem set. If you want to learn the material, I suggest doing it.

Third is the companion code. These are symmetry optics calculations, written for Matlab. The code lets you explore, and experiment, and try examples, and learn. For this lecture it's not so important, but as you go further in the lectures, the companion code will eventually become more and more essential.

And finally, there are more lectures in the series. Please check them out and keep learning.

I'm Paul Mirsky; thanks for listening.