Lecture Notes on Symmetry Optics

Lecture 3:
Static and Live, and the Source-Target Grid

To accompany https://youtu.be/F15-TGgrBNI
v4, April 13, 2022
Paul Mirsky
paulmirsky633@gmail.com

1 Introduction

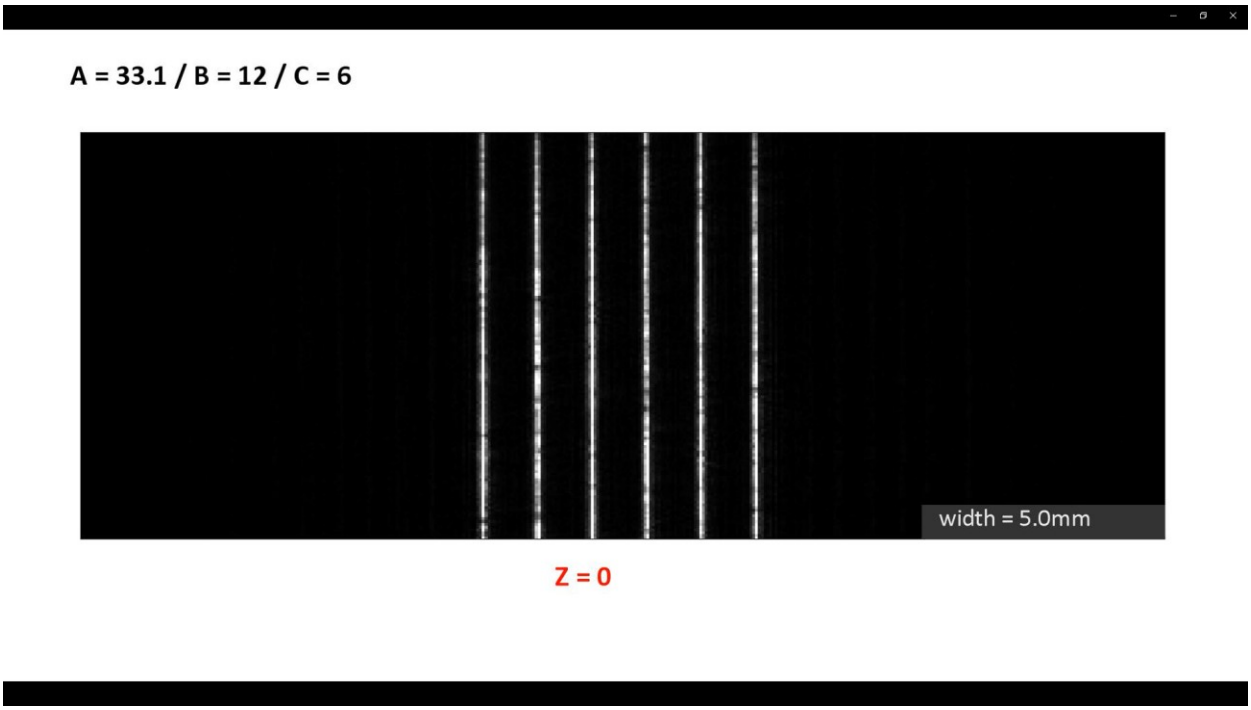
1.1 Overview

Welcome to Lectures on Symmetry Optics. I'm Paul Mirsky. This is Lecture 3 of the series, and the topic is: Static and Live, and the Source-Target Grid.

In the previous lecture, we considered many-slit interference in the *lens-limited configuration*. The setup has an illuminating beam, a grating, a lens, and a screen. But we only really analyzed patterns at two special planes: the front flat, at the grating, and the rear flat, at the screen. These are the special planes where the wavefronts have zero curvature. Everywhere else, the wavefronts are curved.

Now we'll be looking at the *free configuration*, which means that there is no lens and no rear flat. And instead of only considering the flat, we'll consider all the other planes too. To understand these, we will need to introduce the concepts of static and live. As we'll discuss, static stays constant, while live grows and changes.

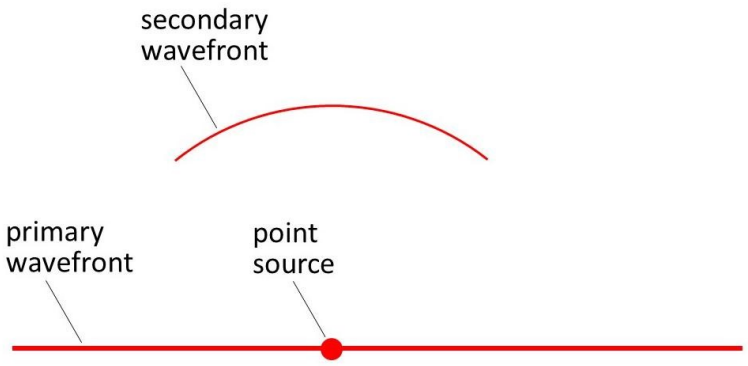
1.2 Pattern evolves



Here's the pattern directly at the grating, seen with a camera.

As the camera moves away from the grating, we see that the patterns will start to change in interesting ways that are too complex to summarize in just a few words. Let's take a minute to watch These are the patterns that we'll learn to calculate. We'll use a new type of algorithm that is unique to symmetry optics.

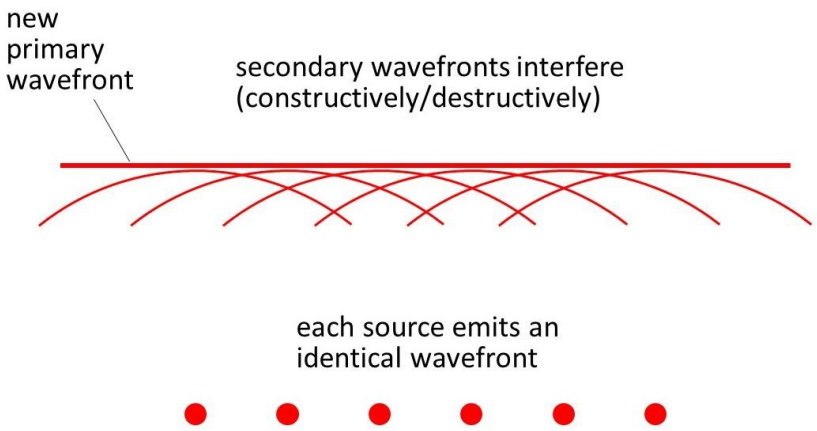
1.3 Huygens-Fresnel principle, one source



But before we do, it's helpful to first review a similar technique that you might already know, namely the *Huygens-Fresnel principle*. We're given some initial conditions, which we call a primary wavefront. Here we're showing a simple plane wave, but it can be any pattern.

A point on the primary wavefront acts like a source, and emits a 2ndary wavefront, which is a spherical wave that radiates out. That spherical wave is one example of a *point-spread function*, which how you describe the effect of a single point source.

1.4 Huygens-Fresnel principle, sources interfere



But actually, each and every point on the primary wavefront acts as a source, and each one independently emits an identical secondary wavefront. These secondary wavefronts propagate and overlap, and they can either reinforce one another, or cancel out, depending on their phases. These are the phenomena of constructive and destructive interference.

When all the secondary wavefronts are added together, the net result is a new primary wavefront. You can now calculate another cycle, or you can stop here.

In mathematical terms, this whole process is a convolution. Take a starting pattern, convolve it with a point-spread function, and you get a new pattern.

1.5 Huygens-Fresnel and symmetry optics, rough analogies

Huygens-Fresnel	Symmetry Optics
Primary wavefront	Static
Secondary wavefront, Point-spread function	Live
Interference	Source-target grid (STG)
New primary wavefront	**No analogy**

The reason for this review is that Huygens-Fresnel has some rough analogies with Symmetry optics. The analogies are not perfect, but at least they will give you some idea of what these new terms mean.

The *static* is like the primary wavefront, because it's our initial conditions, and we treat it like a collection of individual sources.

The *live* is like the secondary wavefront, because it's one common pattern that is emitted by each source.

The *source-target grid* is somewhat similar to interference, because it's how we combine all of the different instances of live.

And there is no analogy to the new primary wavefront. When you find the distribution in some plane, you *can't* use it as the starting point for a new cycle of calculations, the way you can with Huygens-Fresnel.

With this introduction to orient us, let's talk about static.

2 Static and live

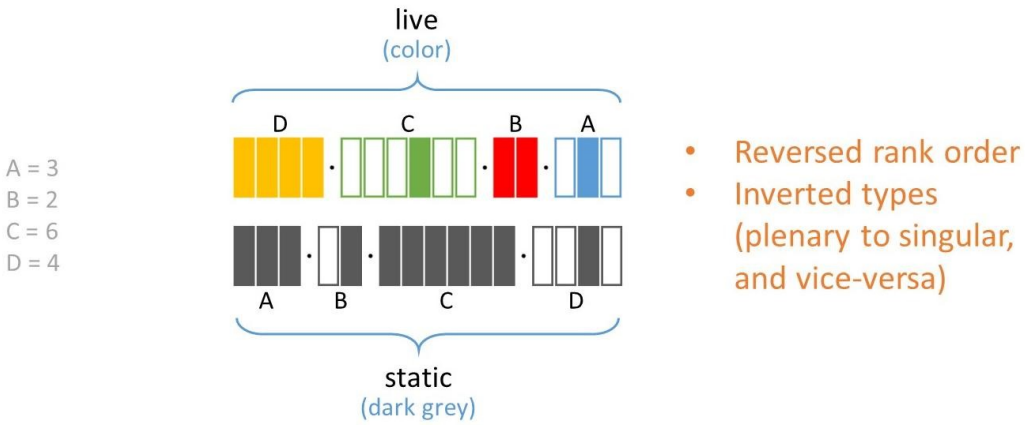
2.1 Static at the flat, Z = 0

A = 3
B = 2
C = 6
D = 4

The flat is the starting point for many-slit interference. A flat wavefront illuminates the grating, and some light passes through the slits.

The pattern here consists only of static. We can represent the static as a single factor chain. The flat is one extreme.

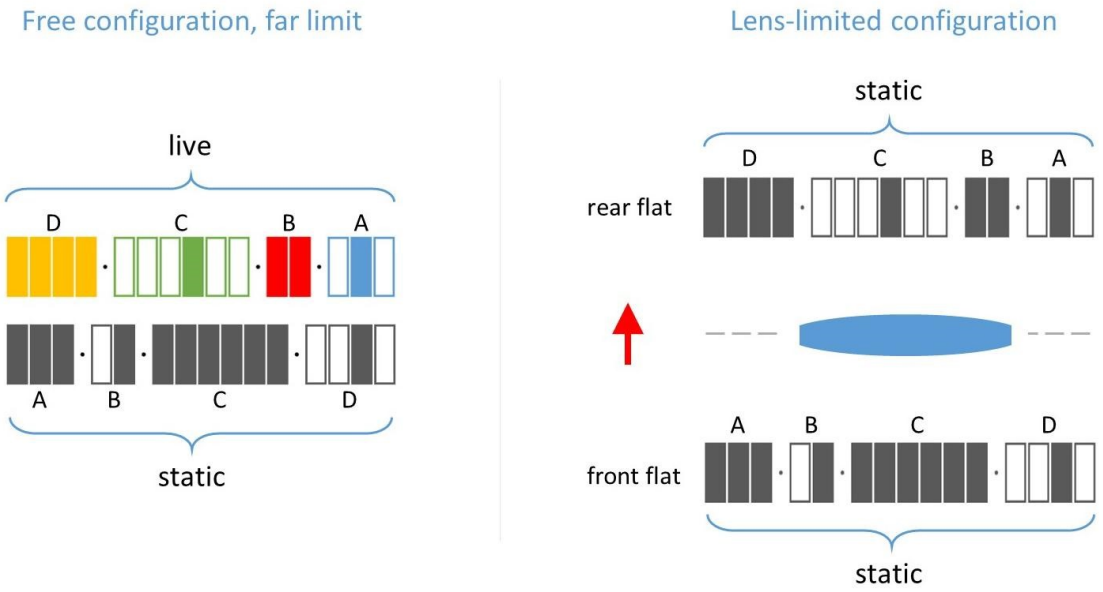
2.2 Static and live at the far limit, Z = ABCD



The opposite extreme is the far limit, which is the plane that we set as the boundary of the system. Here we see two factor chains: static, and live We usually draw the static in dark grey, and the live in colors.

If you look at the details of the live at the far limit, it will look familiar because it's just like the calculation we did in Lecture 2. The static and live have all the same factor sizes. But the rank order is reversed from A-B-C-D to D-C-B-A, and also the types are inverted from plenary to singular or vice-versa.

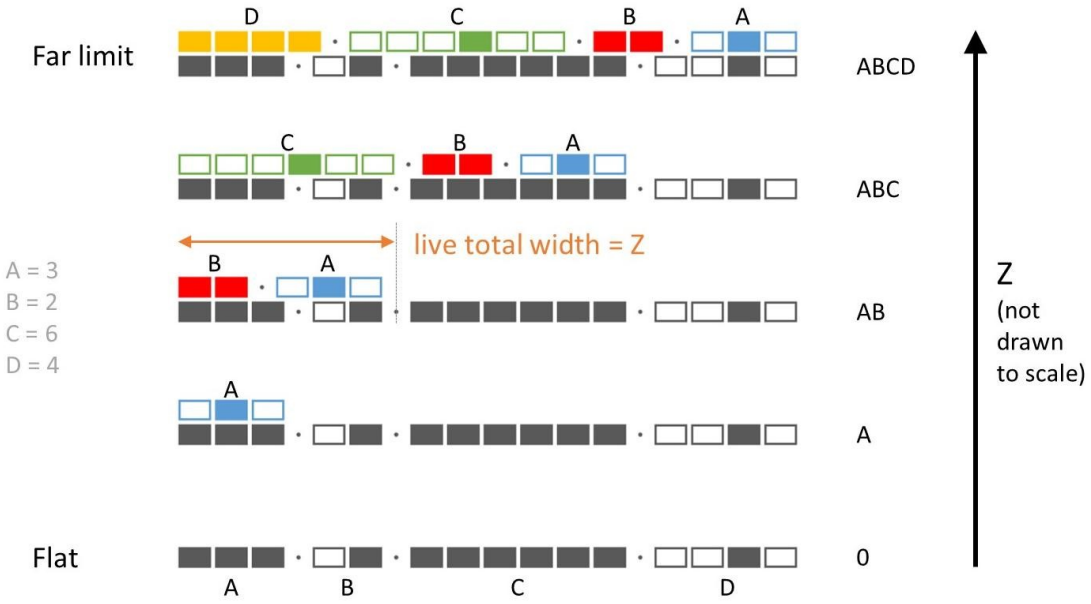
2.3 Far limit vs lens-limited configuration



But the calculation in Lecture 2 was about something else. That was in the lens-limited configuration. We were given a pattern at the front flat, and we calculated the pattern at the rear flat. Those chains described two different planes. Also, both of those patterns were static only.

Here is different: both chains are present simultaneously in the same plane. One is static, the other is live.

2.4 Static and live at any Z



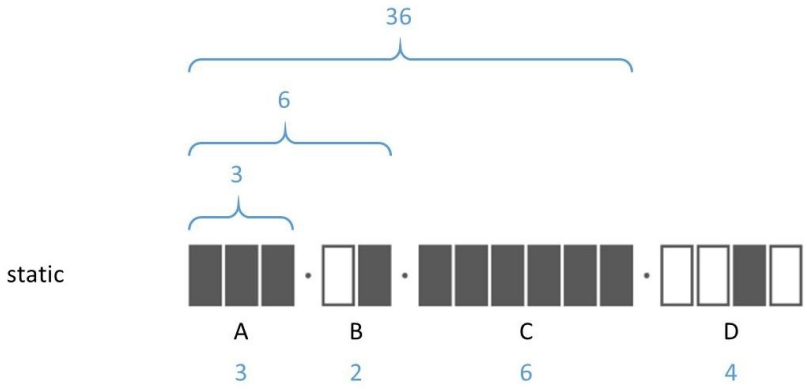
We've seen the extreme cases; now, let's look at the intermediate cases. This slide shows both the static and live for many different planes. At the bottom is the flat, where the live is zero. At the top is the far limit, which we just saw in the previous slide. The overall trend is that as the light propagates, the live chain moves to the right.

Each new factor always first appears at the lowest rank. The existing factors rise higher in rank, but otherwise, they don't change. The rule is that the live total width is equal to Z, which is the distance away from the flat, given in wavelengths. So, for instance, at Z = AB, the live total width is AB. And at Z = 0, which is the flat, the live total width is 0, which is equivalent to saying that there is only static, no live. Notice, though, that the static remains constant for all planes. That's why it's called 'the static'. The live grows and changes, which is why it's called the live

2.5 Steps to calculate live

1. Find static subchain, total width = Z
2. Reverse the rank order
3. Invert each type

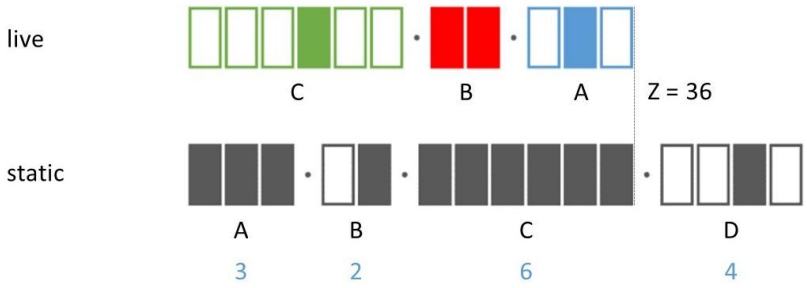
2.6 Calculating live at Z = 36, static subchain



Now for the nuts and bolts. Let's work through an example, and calculate the live step-by-step. Here's our static, and let's assume we want the live at Z = 36.

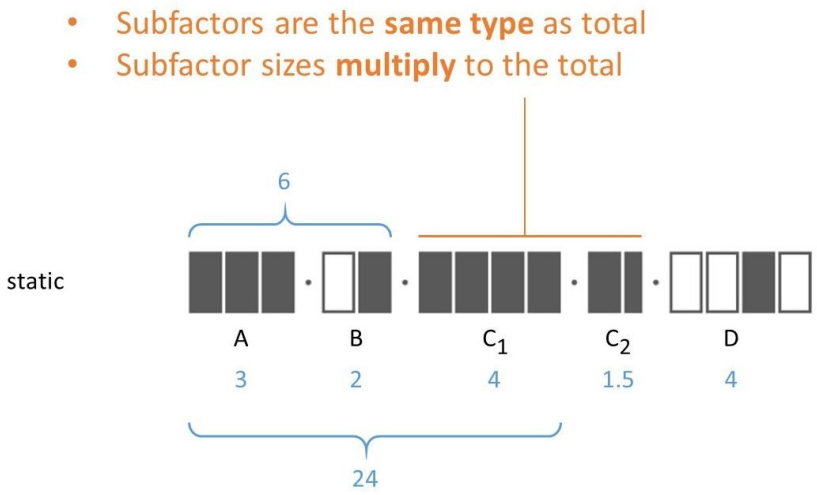
Step 1 is to find a low-ranking static sub-chain, with a total width of 36, or whatever Z is
The process is pretty simple: starting at the lowest rank, we include factors until the product is 36. So, factor A is size 3. Factors A and B together give us 3 times 2, which is 6. Factors A, B, and C together give us 3 times 2 times 6, which is 36. This gives us our subchain, so we're done with step 1.

2.7 Calculating live at Z = 36, reverse and invert



Now, we turn the sub-chain into the live. Step 2 is to reverse the rank order. So, A-B-C becomes C-B-A. Third, we invert each type. So, for instance, A goes from plenary to singular, etc. This gives us the live at the plane Z = 36.

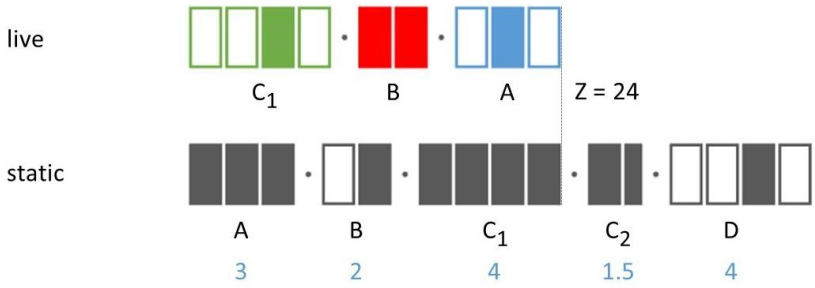
2.8 Partial factors



Next we will try the same static, but at Z = 24 rather than at Z = 36. We begin exactly the same, by including factors A and B. But when we try to include factor C, we see that it overshoots our target, because we only want to get to 24, but factor C would bring our product to 36. So, AB is too little, but ABC is too much. This problem is very common – in fact, there are only a few planes where it *doesn't* occur.

The solution is to include only part of Factor C. The trick is to break factor C up into two subfactors – C_1 and C_2. To do this, just remember two things: First, both subfactors are the same type as the total factor – so, C_1 and C_2 are both plenary. Second, the two subfactor sizes have to *multiply* to the total, not add – so, 6 splits into 4 and 1.5, *not* 4 and 2. We could have chosen sizes 3 and 2, or any other pair that multiplies to 6. But we chose 4 as the size for C_1, because that's the size that we need in order to get to 24. Now, we can include factor C_1, this gives us 3·2·4, which is 24. Now we have our sub-chain.

2.9 Partial factors, reverse and invert



Next, we reverse the order and invert the types. This gives us the live at Z = 24. This algorithm is pretty easy to do by hand, but you can also do it in software.

2.10 Companion code

Wherever you found this video, you'll also find a link to the *companion code* for this course. This is a set of functions and classes written for Matlab, and you can learn a lot from it. Matlab is pretty cheap if you're a student, or a home user. If you're a professional, then it costs more. There may also exist free software that can run Matlab code, but I can't vouch for it.

This lecture will show some brief demos of the companion code. We won't narrate every single detail, but hopefully you can follow the basic flow. The key thing is to inspire you to check out the code for yourself.

I'm opening up the file called *calcLive_MSI.m*; MSI stands for many-slit interference. We're going to do the same calculation we just did a moment ago. The code is over here in the left pane. When you press 'Run', it does the calculation, and it reports the answer as text, here in the right pane. Here it shows the static, the Z, and the live. The static and live are written as chains of sizes and types. The lower-ranking factors appear at the bottom.

Let's take a closer look the code. I'll set a breakpoint at the first line, so that we can step through one line at a time The first few lines just clear up the workspace, so that we're starting fresh. Then we set the static sizes and types, and we call the function 'listFactorChain', which just prints the static chain to the screen. Next we set Z equal to 24 Then we create an object called 'calc', which is an instance of the calcLive class. [hover] When we hover with the mouse, we see that it has a bunch of properties, but they mostly just show a pair of brackets, which means that the properties are all empty. Next we assign the static sizes and types to the properties, and when we hover again, we see that now those properties are filled in.

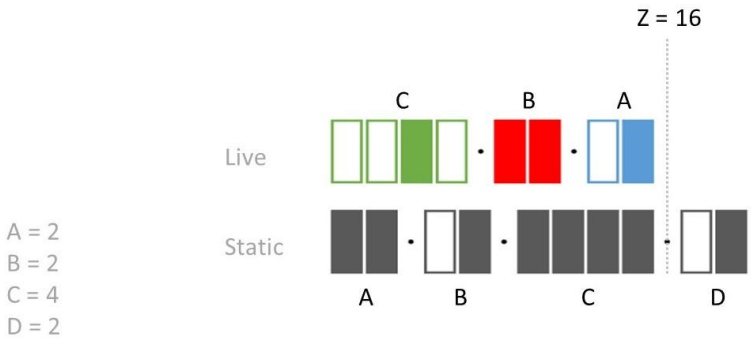
The next line is where the real calculation happens. We call the function calcLiveAtZ, with z as an argument, and when we hover again, we see that even more properties are filled in. Those are our outputs. Now we call listFactorChain again, and it prints the live. That's it.

We'll run the code one more time, but this time, when we get to the function calcLiveAtZ, we'll press 'Step In'. This will take us to the code that defines the calcLive class. We'll skip over some of the details, but the most important part is this loop. We cycle through the loop on the condition while subchainProduct is less than targetZ, just as when we calculated by hand.
I'll set a breakpoint at the bottom of the loop, and press continue to execute up to the breakpoint. When I hover, you can see that we are keeping a list of the subchain sizes and types, and because it's only been one pass through the loop, the lists are only one item long.
But when I press continue again, it passes through the loop a second time, and now they're two items long, and now 3 items long. Now we step though a few more lines to reverse the rank and invert the types, and now we return to the calling function and continue to the end, and print out our result. Now we're done.

As I said before, it's OK if you didn't follow every step. The main thing is just to alert you that this code is available, and to motivate you to download it and run it yourself.

3 Source-Target Grid

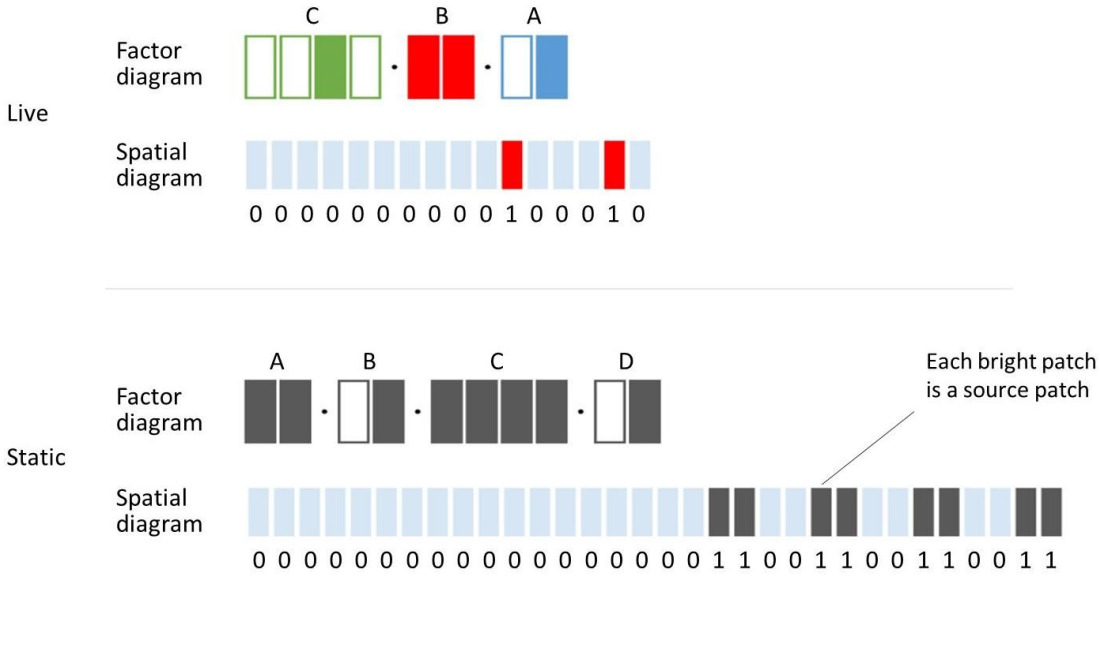
3.1 New example, calculate live



Now that we've discussed how to calculate live, we'll move on the next step, which is to combine the static and live in the source-target grid. We'll illustrate by working through an example. Again, we start with the static parameters A, B, C, and D, which in this case are 2, 2, 4, and 2. We're also given Z, which in this case is 16.

The first three steps are to calculate the live at Z = 16, as we did earlier. We won't work through all the steps, we'll just show the result here.

3.2 Spatial patterns

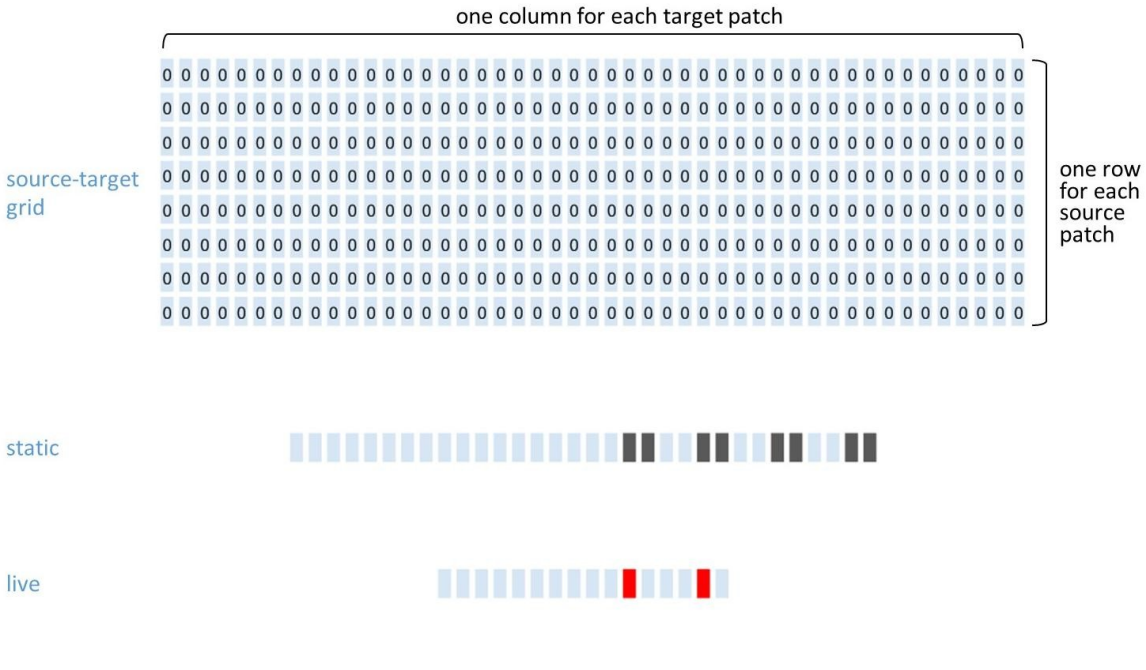


Up to now, we've been representing the light with factor chains. But it can also be represented as a *spatial pattern*, which is how the light actually looks like in space. Step 4 is to calculate the spatial patterns for both static and live. We use the algorithms that were taught in the previous lecture, and we get these patterns for the static and live, respectively. They are vectors of 0s and 1s – a 0 for each dark patch, and a 1 for each bright patch. They can also be drawn as diagrams. We use light blue to represent 0s. Any other color represents 1s.

Also, note that all of the factor sizes need to be integers, or else we can't calculate these patterns. Because of this, the algorithm doesn't work for all values of Z. In a later lecture, we'll learn a different algorithm that lets us get around that limitation.

Each and every bright patch of the static is an independent source patch. In this case, there are 8 source patches. Together, they make up the grating pattern that we see at the flat.

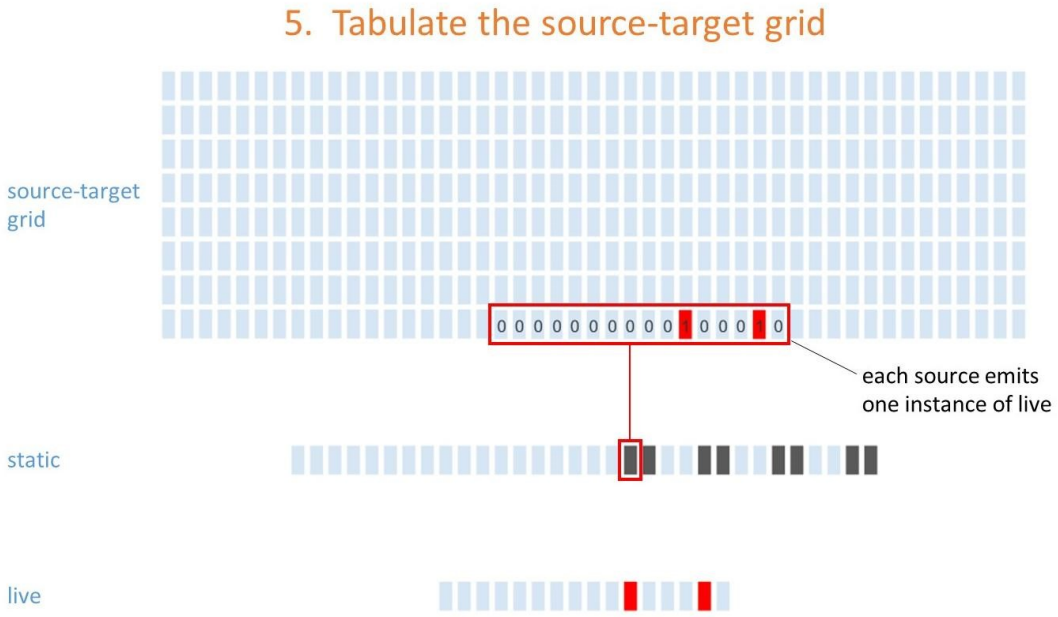
3.3 STG



We now introduce the *source-target grid*. This is a 2-dimensional table. Each column corresponds to one of the target patches, where we want to know the interference pattern. Each row corresponds to one of the source patches.

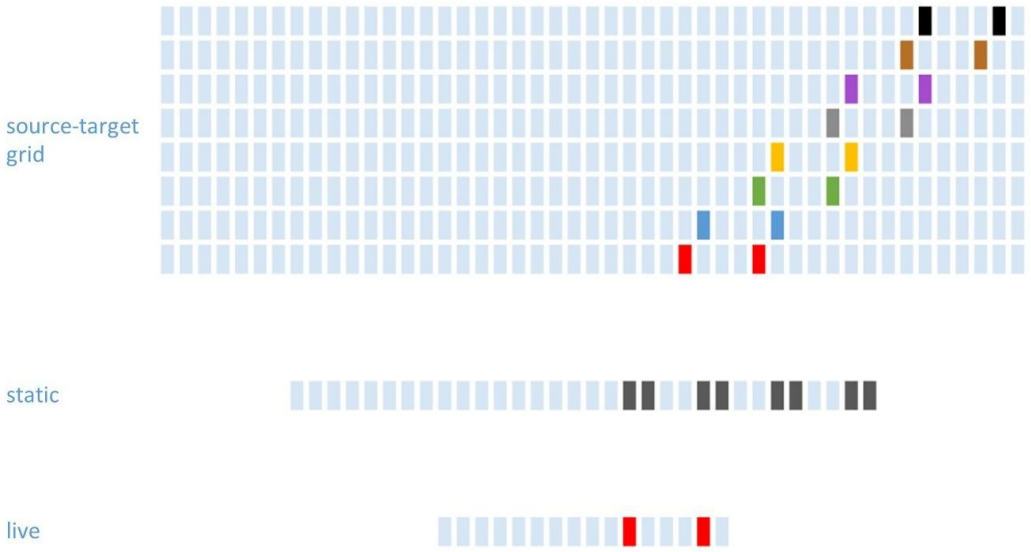
Each cell of the grid contains a single number. At first, the grid is completely filled with 0s. We usually just represent the zeros with light blue squares, but this slide actually writes them out. In the next step, we overwrite some of these 0s with 1s.

3.4 Tabulating one source



Step 5 of our procedure is to tabulate the source-target grid. Each source patch emits one *instance* of live, and we tabulate it by writing it into the source-target grid as a string of zeros and ones. Horizontally, the live is centered above the source patch that emitted it, as you see here. Vertically, each instance is tabulated into a different row – this is the 1st source patch, so it's in the 1st row of the grid. We follow this same procedure for all the source patches.

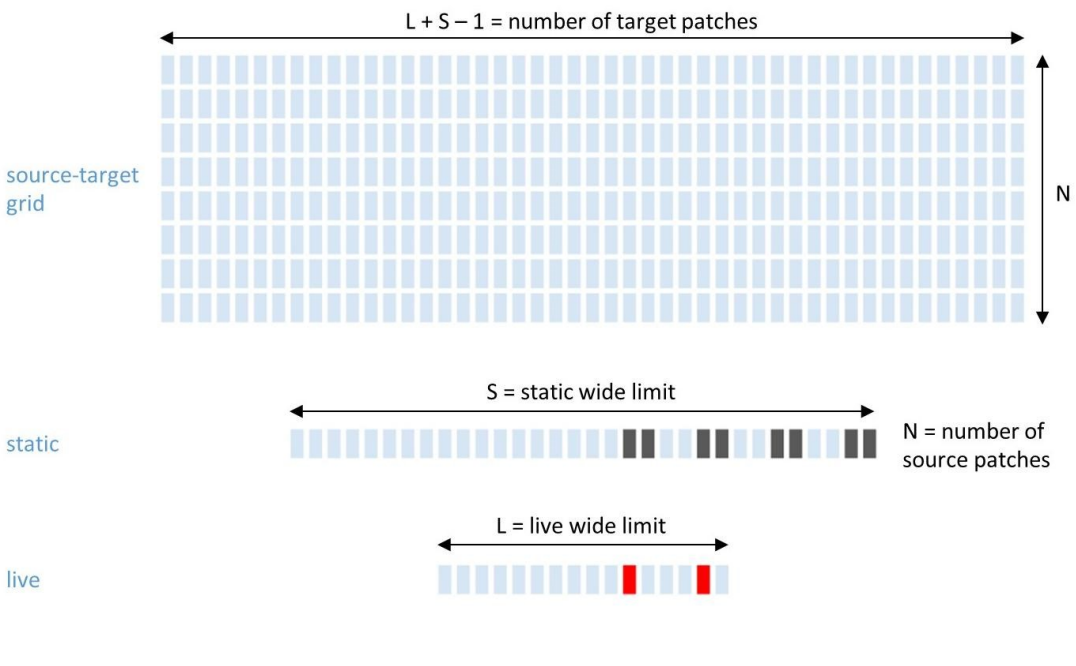
3.5 Tabulated grid



The live from the second patch is centered above the 2nd patch, and tabulated into the 2nd row, and so on for the rest. The tabulation is now complete.

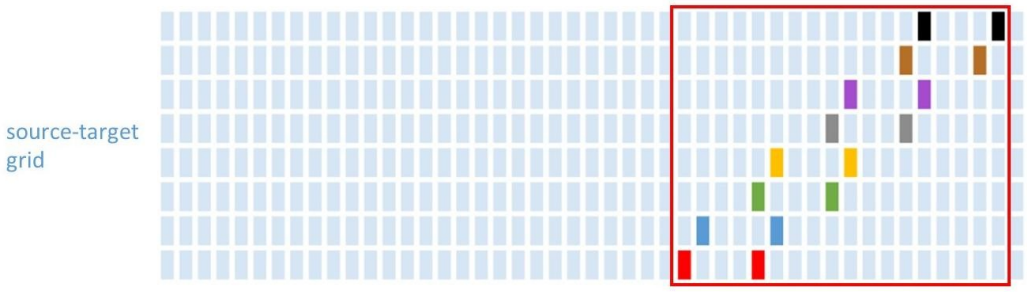
Note that even though these rows are arrayed vertically in our diagram, they at *not* at different distances from the flat. The entire grid all applies to just one plane. Also note that we often draw each instance in a different color, but that's only to help you tell them apart. Formally, they're all just represented as 1s. And we will eventually drop the colors, once they are no longer convenient.

3.6 Grid dimensions



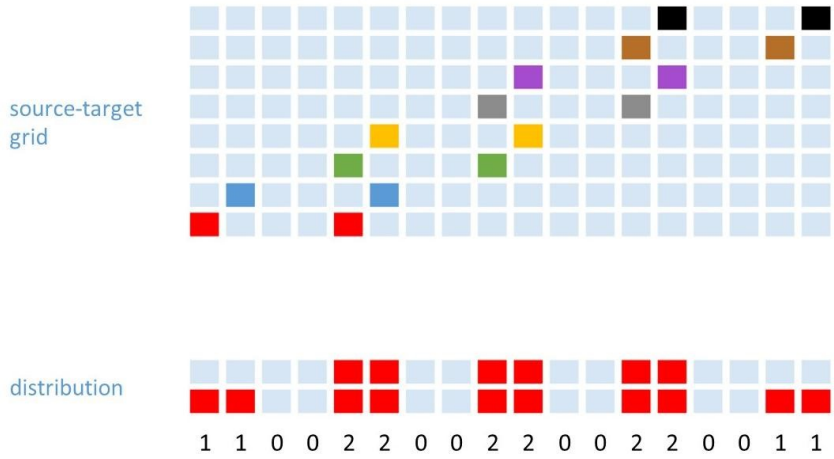
Now that we see how the source-target grid is used, let's look at the dimensions of the grid. If N is the number of source patches, then there are N rows in the grid. If L is the live wide limit, and S is the static wide limit, then the grid has a total of L + S – 1 columns.

3.7 Trimming the grid



This is the size we need if we want to be able to tabulate every possible case. But in practice, it's usually possible to get by with far fewer columns. This is because most of the source-target grid is usually just filled with all 0s, representing dark space. We often aren't really interested in those regions, so we trim the grid to the limits of the bright patches, which is this red border. This makes it a much more manageable size, especially when the amount of dark space is large.

3.8 Distribution

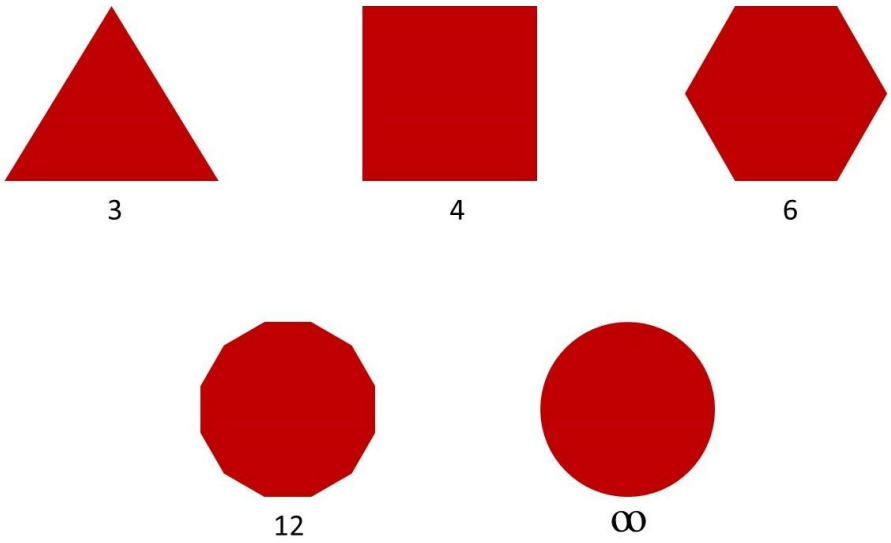


The next step is to collapse the source-target grid into the *distribution*. Effectively, we count the number of bright patches in each column. So, for instance, this column contains 2 bright patches, so the distribution is 2. This column contains no bright patches, so the distribution is 0.

Note that we stop tracking which row each patch is in, or how the bright and dark cells are arranged within a column. All that information just gets thrown out. There's probably a better way to do this, one which doesn't involve erasing information. But, we still don't really understand how to interpret the arrangement within a column. That arrangement probably does carry some significance, and hopefully in the future it will be understood better.

But for now, we just count bright cells within a column. That number is called the *roundness*.

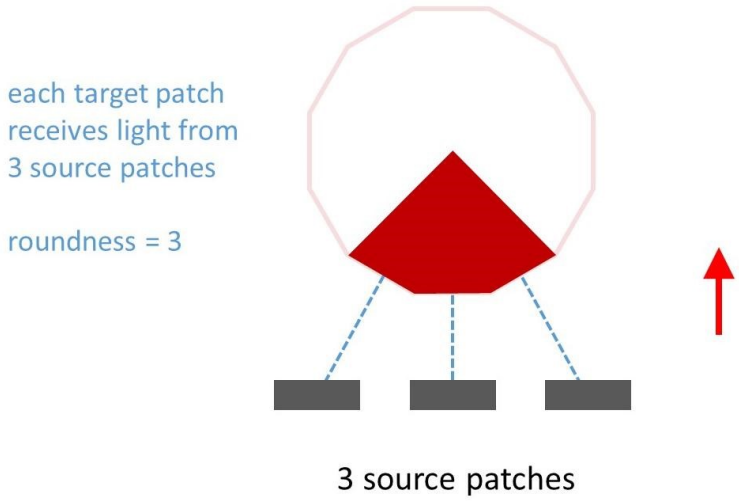
3.9 Roundness is like number of polygon sides



Roundness is a somewhat tricky concept. To introduce it, let's first consider these regular polygons. The more sides there are, the closer the polygon gets to a circle. So the term roundness, in this case, means the number of different sides.

In a later course, we will learn that roundness is actually a measure of *angular symmetry*, or *tilt symmetry*. But that's beyond the scope of the current lecture.

3.10 Roundness is number of discrete angles



3 source patches

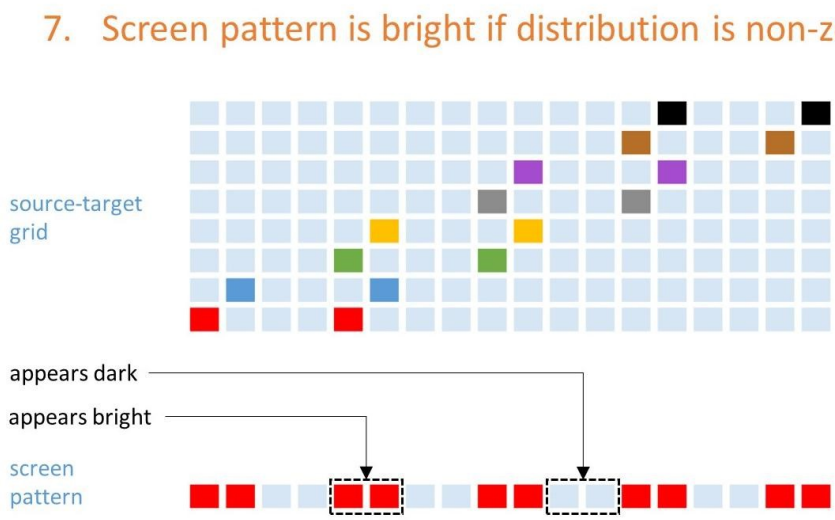
In symmetry optics, we think of light as having multiple discrete angles at each and every individual target patch. The *roundness* is the *number* of discrete angles.

These angles usually don't range over a full 360 degrees, so rather than visualize it as a regular polygon, you can roughly visualize it as a wedge cut from a regular polygon. But don't take this visualization too literally. The point is that there are many different angles, and they are arranged in a symmetrical fashion like the sides of a regular polygon.

The concept of roundness is unique to symmetry optics; it doesn't really make sense in terms of wave optics, although it's closely related to *wavefront curvature*. Roundness is also the number of different source patches that emit light to each target patch.

That's how this is connected to the source-target grid. The number of bright patches within a column of the grid is the number of source patches that emit light to a given target patch, which is also the roundness of that target patch.

3.11 Screen pattern

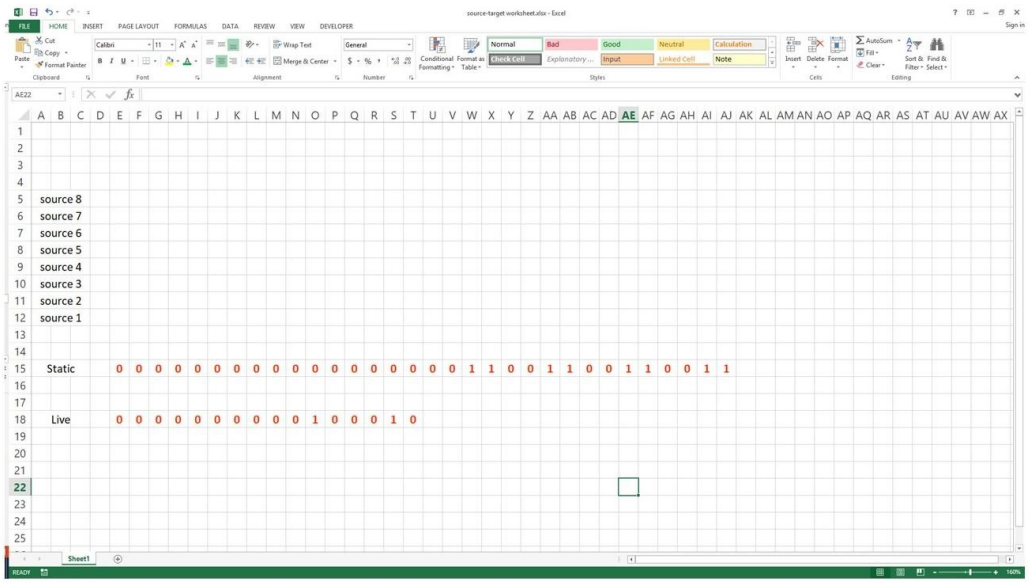


Finally, the distribution tells us the *screen pattern*. This is what you can directly observe on a screen in the target plane, or with a camera.

Each column represents one target patch. If the distribution is 0, if there are no bright patches in the column, then it appears dark. If it's anything besides 0, if there are any bright patches in the column, then it appears bright.

This whole step is somewhat analogous to *interference*, because at each target patch we collect the contributions from all of the sources. But here, there are no phases to cancel, and therefore there is no such concept as *destructive* interference. Instead we just think of it as dark patches of the live, all lined up.

3.12 Spreadsheet tool for STG



You can use a variety of tools to do these tabulations. The simplest option is to use a sheet of graph paper. A more convenient tool is to use a spreadsheet program. Let's assume that you've already got the Static and the Live, calculated as vectors of 0s and 1s. In the static, we count 8 1s, or 8 source patches, so we set aside 8 rows to make the source-target grid.

Now, highlight the live, and copy it with right-click and copy. Next, go to the first bright patch, and find the first row, and paste. It's not centered on the source patch, it's shifted over to the right, but for now it doesn't really matter. Now do the same thing for all the other source patches, and here we have a tabulated source-target grid.

In principle, all of these other cells are filled with zeros, but again, it doesn't really matter. To reduce the grid, you can take the sum of the columns, and that tells you the distribution.

3.13 Companion code

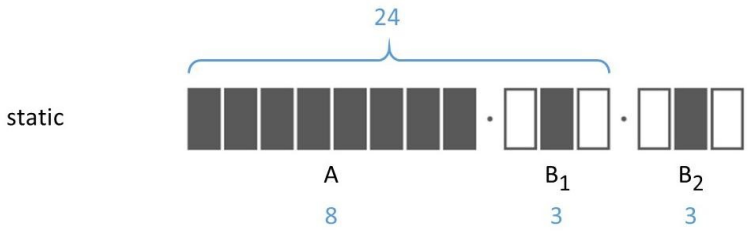
Once you've done a few problems by hand, and have a solid grasp of how the algorithm works, it's much easier to do it using the companion code, especially when the sizes are larger.

Let's try another demo. First I'll open the file *factorsAndSpatial_MSI.m*. Here we have the inputs to the problem we just did – the static chain, and Z. Press Run to execute the code, and it draws both the static, and the live, and it draw them each in two ways: as a factor chain, and as a spatial pattern. Also, in the right pane, we see the spatial patterns, represented as vectors.

Next I'll open the file *calcSTG_MSI.m*. Up at the top, we have the inputs – the static chain, and Z. Press 'Run', and it draws two windows, which contain the diagrams. Here are the live, the static, and the source-target grid. And over in this window, we have the grid, plus the distribution and the screen pattern.

4 Application to the beam

4.1 Static subchain

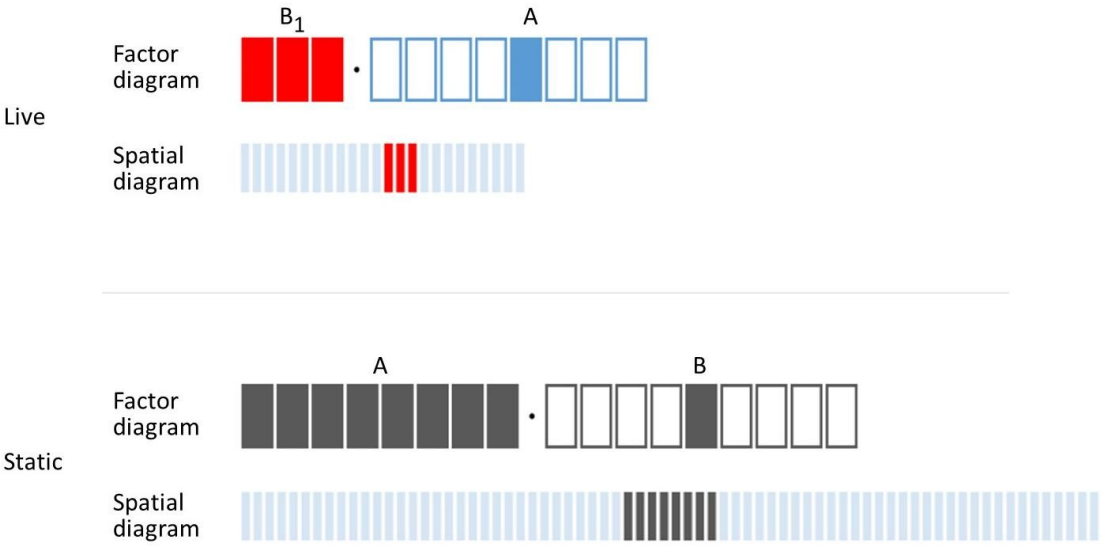


One final problem for this lecture. Up to now, we've only considered static and live in the context of many-slit interference. But the principles are equally valid for the beam. The only difference is that there are fewer factors in the chain.

Here is an example of the static for a beam. It consists of just two factors: a size-8 plenary, and a size-9 singular. We're looking for the distribution at Z = 24.

We begin by calculating the live. First, we look for the static subchain that's 24 patches wide. To start, we include factor A. Next, we see that we have a factor of 3 left to go, in order to arrive at 24. Factor B is larger than 3, so we split it into two factors, with the lower factor size 3. We include the lower factor B1, and this gives us a static subchain of the size we need.

4.2 Live, and spatial patterns



Then, we reverse rank, and invert type, and get the live. From the factor chains, we calculate the spatial patterns. These are the inputs to the source-target grid. We'll tabulate the source-target grid using the file *calcSTG_Beam.m*. And, here are the results.

5 Conclusions

5.1 Reviewing steps

That's all for this lecture, so let's review the steps to calculate the pattern in any plane.

1. Find static subchain, total width = Z
2. Reverse the rank order
3. Invert each type
4. Calculate spatial patterns for static and live
5. Tabulate the source-target grid
6. Collapse the STG into the distribution
7. Screen pattern is bright if distribution is non-zero

5.2 Reviewing key points

Let's review a few other key points:

- A single plane contains both static and live.
- The distance Z equals the live total width.
- The roundness is the number of source patches that emit light to each target patch.
- The same rules work for both Many-Slit Interference, and the beam

5.3 Outro

Wherever you found this video, you'll also find links to more resources for learning symmetry optics.

The lecture notes are basically this video, but in written form. Sometimes, it's better to read it than to hear it.

Also, try solving the problems in the problem set. It's a great way to get familiar with the principles of static and live.

In the next lecture, we'll apply those principles to explore the structure of the beam. So please join me.

I'm Paul Mirsky; thanks for listening.