[SQUEAKING]

[RUSTLING]

[CLICKING]

BERTHOLD HORN: Be a new homework problem out after this class, and we'll try and cover as much as possible that is relevant to that. The last two questions, you will have to wait for next week's class. However, there's extra time. So it's not due in a week. It's due slightly later. So we should be able to get everything in place by Tuesday.

OK, so the last patent we were talking about was one that allowed us to efficiently compute filtering outputs in images that have been filtered. And in particular, the interest was in suppressing higher frequency content so that we can sample them without getting aliasing artifacts. But also, we could use them for edge detection and any sort of process that's convolutional and get a speed-up by exploiting the sparse nature of a higher derivative of a spline.

So rather than focus on the patent, let's sort of look at some of the relevant material and discuss it without specific reference to the patent. So a major component of this, of course, is Nyquist. So just not to beat a dead horse to death, as the saying goes, when we sample a waveform, it's kind of surprising that we expect to capture something about the waveform, because the waveform has infinite support. And we are just getting these discrete samples.

So it's surprising that we can expect to capture that waveform that way. And the only reason that it can work is if there's something very restricted about the types of waveforms we're dealing with. And that's band limiting.

So if their frequency content is limited, then there's this wonderful theorem that basically says that if we sample at a high enough frequency, then we actually capture everything there is. We can completely reconstruct it, and it's a constructive proof. It's not something that's simply a mathematical proof. But there's actually a method for reconstructing it and its convolution with the sinc function. So it's a very satisfying kind of theory.

And Nyquist and his predecessors showed that the criterion is that we sample fast enough so that the highest frequency component of the signal should have a frequency that is less than fs over 2. And we can sort of see why that is.

Now, suppose we sample here and here and here. We seem to get the important part. We get alternating ups and downs. And also, it's clear that if we sample, say at the full frequency, if we have a gap from here to there between samples, that's not going to work, because all those samples are the same.

So just if you forget what the formula is, you just look at this diagram and say, well, if I ultimately sample waves-- troughs and peaks, then I captured the waveform. It's a little bit dangerous argument, because if I take that same waveform and I sample at the same frequency, I get this, which isn't satisfactory. And of course, I can get anything in between if I just shift the phase of the sampling. And that's why it's less than, not less than or equal to. So it has to be strictly less than.

OK, then if we don't satisfy that requirement, we have aliasing. So why is it called aliasing? And those of you who are thoroughly familiar with this can go to sleep for a moment. We'll just look at this here.

So here's a waveform. And let's, again, suppose that we're sampling, but now not exactly on the peaks but a little bit offset. So we have-- in the space of the main, we have a waveform with frequency f0. And suppose that we're sampling at some interval, 1 over delta. And so the samples we're going to get are cosine 2 pi f0 k 1 over delta. Right. So these are our samples, let's say.

And so that's-- we can just substitute. Substitute that. And now, if we add 2 pi to the argument, nothing changes, right? In fact, if I add any integer multiple of 2 pi to the argument, I get the same results.

So for example, I could-- do I want to add or subtract? It's going to be easier if I subtract. OK, those are still going to be the same samples, right, because I've just changed the phase by some multiple of 2 pi. OK, so that means I can write this.

And so what that means is that I started off with a wave of frequency f0. And now, I'm saying that it's the same as a wave of frequency f0 minus fs, or in fact, the same as-- cosine is an even function. So if I wanted to, I could say fs minus f0.

And so let's suppose this is zero frequency. And here is my f0. And here is fs. Then I'm saying that something at this frequency fs minus f0 cannot be distinguished from something at f0 and also, by the way, fs plus f0 but not so worried about that.

So now, imagine that f0 is higher. So I suppose it's there. Well, then this one comes down, close it, and so on. So clearly, I need to cut things off there. As long as I don't have any frequency content above that, I'm not going to have any confusion. And it's called aliasing, because those other frequencies-- and there's an infinite number of them-- look the same in the sample.

Notice, by the way, that that means that you can't fix this after sampling. It has to be done before sampling. And so there are various kluges to try and do that, which suppress higher frequency content. But you've already committed the sin. It's too late. You're no longer able to distinguish frequencies that came from up here from frequencies down there. So and this is something that I guess we should all well know.

OK, then we're going to approximately low-pass filter, because truly low-pass filtering is very hard. Before sampling, we need to filter. And that's important when we're talking about multiple scales, because we can't just, like, take every fourth pixel and assume that we're going to get something sensible.

That'd be a little bit like designing an Earth mapping satellite, which is really good. It's got incredibly good optics so that each pixel corresponds to a point on the ground. That wouldn't be a very useful picture, because this pixel is hitting that gray stone there. And this pixel is hitting the table. And it'd be incoherent. And it would be a very dramatic example of aliasing. So this is a case, where actually, you want things to get blurred-- blurred just enough.

OK, so we'll talk about different ways of blurring pre-sampling filtering. And so I want to start off by talking about something called the integral image. So one idea is, what if we convolve by what I'll call a boxcar, which is a filter like that? Basically, that's averaging, right? We're taking a block average.

And to compute the average, we want to compute the sum. So what's an efficient way of computing the sum? Well, of course, the obvious implementation is we just go in, and for every pixel in the output, we find out which pixels in the input it corresponds to, get those values, add them up, and maybe divide by the total number of pixels. But that's obviously expensive, because then the number of computations is roughly the number of pixels times the width of that filter.

Well, there's a very easy way of dramatically speeding this up, which is using an integral image. And we will first do it in 1D. So let's suppose that we have a g for grade level. We have a sequence, g of i. And we're going to transform it into a sequence, big G of i, where a big G of i is-- so we just left to right, add up the grade levels, which seems like a weird thing to do.

But now, it means that if we want the sum from k equals i to k equals j of the original one, we can just get-- right, we just-- one subtraction and independent of the length of the block average. In the naive implementation, the amount of work goes with the width of the block average. Well, now, that's not the case anymore. It's just a single operation.

So if we're going to do this a lot, then it's very efficient to precompute this so-called integral image. And this is actually used quite a bit. Of course, with images, we're dealing with 2D. But you can imagine that we can perform the same operation in 2D, where, now, the value in the output here is the sum of all the pixels in this rectangle.

By the way, of course, with pixels, maybe we only need 8 bits to represent the gray level. Once we start adding them up, we need many more bits. So there's some issues about dynamic range. Do you use floating point? You use 32-bit integers-- whatever? But obviously, we need to estimate what is the maximum we're going to get wher we add up all the pixels and suppose that each of the pixel is at its maximum value, et cetera, et cetera. So that's all pretty obvious.

OK, now, in the 2D case, what I want to do is typically compute a total over a block like this. So it's a 2D block averaging. And that could be for purposes of approximate low-pass filtering. Or it could be for other reasons.

So by the way, just because I'm calling it an integral image, doesn't mean it has to be derived from an image. We could, for example, have integral gradient. So we compute the gradient everywhere, which is now a two-vector. And we perform the same operations. Then we get a matrix of two vectors. And now, we can do things with that.

And in fact, in some high-speed recognition processing, this is the first step, where you estimate histograms of gradients, and then you compute using block averages of this type. And it's exactly this process, just made more complicated, because we're not dealing with scalars. And this maps nicely onto GPUs. So it's used for quite a bit of recognition and such.

So let's see. Suppose I want the total over this rectangle. How can I compute it from the integral image? It's not completely obvious, because if I just take the value at that corner-- suppose I take this value. Well, that's the total of all of these. So that's not what I want.

If I take the value at this corner, that's the total of all of those. So maybe I should subtract that. Then I might take the total at this corner. And that's the sum of those. Well, maybe I shouldn't subtract the blue one-- get one more color. But I need to get rid of that bottom part. So maybe I can subtract this one. Now, if I subtract the green and the orange from the red, I subtracted this one twice, the blue one. And so I need to add it back in again.

So basically, we get the total over this rectangle by adding this and that and subtracting this and that. So and you can extend this to higher dimensions also, although I haven't seen much use for that. So that means that with-- no, four memory accesses and three arithmetic operations, you can get the total for that block and independent of the size of the block. So it's no more expensive to average of a 4 by 4 than over a 8 by 8 or vice versa.

OK, so and that really is the 0-th level example of what's in that patent. And we'll see why. OK, now, how good is this? So suppose that we block average like this. And then we sample. And we're trying to satisfy Nyquist. Are we really suppressing all those higher frequencies? Well, let's look at that.

So little bit of Fourier analysis-- when I was a student, I was taught Fourier transforms in six different courses. And each of them used a different notation-- you know, frequency in cycles per second versus radians per second. Divide by 2 pi. Don't divide by 2 pi. And it was very confusing. So I won't hold you to getting the constant multipliers right at least.

And then I, later in my life, started doing some mathematics. And the mathematicians had yet another way, which is very sensible. I don't know why engineers didn't adopt it, which is the unitary version of where you divide by the square root of 2 pi. And then the forward and the inverse are exactly symmetrical with only the sign change. But anyway, OK, so what have we got?

So we have a block of, let's say, with delta that we're averaging over. And we can make the height of this thing 1 over delta just so that when we get the result, it's the same magnitude as the original value. And so we're really doing a moving average. And so what have we got?

So that's going to be from minus delta over 2 to plus delta over 2 of 1 over delta. So we're going to get 1 over delta. I imagine you've seen this before. And so we get-- we get a sinc function. And sadly, I can't write it as a sinc, because again, the mathematicians and the engineers can't agree on the definition. One has a pi in it, and the other one doesn't. And it's just going to confuse it. So I'll just leave it like that. So what does that mean?

Well, that means that it is attenuating higher frequencies, because while sinc just keeps on wobbling away as we go off in omega, we have the division by 1 over-- by omega. So it's decreasing inversely with frequency. That's the good part. But it doesn't become 0, except at special places. And so it's not a terribly good approximation of a low-pass filter. But what's even worse is that it doesn't get to the first 0 fast enough. So let's see what we got.

So the first 0 is, of course, where the argument of sine is pi over 2. So we got-- and so that's at omega 0 equals pi over delta-- 2 pi over delta-- really-- which is twice the frequency we got for Nyquist.

So if we plot it-- so here's what we want. Here's our ideal low-pass filter. And here's what we get. So this is Nyquist. And this is what we're getting. So the good part is we are attenuating higher frequencies. The bad part is we're not aggressive about it. And another bad part is that we're actually attenuating some of the frequencies we want a little bit.

So and why is this? Well, this is particularly relevant to us in terms of cameras, because this is what the camera does. So this is our pixel. And here's another pixel. And so they are performing a filtering operation, because they're averaging the light coming over that whole area.

And so that's good, because we're performing some low-pass-- approximate low-pass filtering before sampling. It's not ideal, because is that sinc function, that approximation to that blob? No, and it's first [INAUDIBLE]. Then that's why we need additional filtering so that we don't have the aliasing.

OK, and so one thing we can do is find better filtering functions. Of course, they're likely to be computationally more expensive. And so let's start off with repeated block averaging. OK, since block averaging is so cheap to compute, we can perhaps perform it twice in the hope of getting something better.

And so what we're going to do is take our function and convolve it with the filter. So this is the image. This is our filter, which is a b for block averaging filter. And now, let's take that and convolve it again.

Now, we mentioned last time that this is associative. And so we can write it this way. So instead of thinking of it as convolution with boxcar, we can think of it as the convolution of the image with this thing, which is the boxcar convolved with a boxcar.

So and that's this triangular waveform, which is kind of smoother. So that's what you-- you would think that would be beneficial, that we don't have as much high-frequency content. So every time there's a discontinuity, that means that there's high-frequency content. If it's a step function, then it's 1 over f. If it's only a discontinuity in the derivative, in the slope, then it's 1 over f squared. So this one here, we expect to drop off as 1 over f squared, because these drop-- these suppress 1 over f each of them.

So OK, well, in the transform domain, of course, it's going to be F times B times B, which is the same as FB squared. So this is sinc squared. So this function, this filtering is somewhat better, because now, we're taking this thing squared. So when it gets close to 0, we'll be doing this. Oh, oops. No, square cannot be negative.

So it's dropping off at the square of the rate. So instead of 1 over f, it's dropping off as 1 over f squared. And so it's better. But it's not wonderful. And one of the things we can do is do it again. We can do a sequence of block averaging.

And since we have a fairly efficient way of doing one, it's not crazy to do several. And of course, we'll just get higher powers of the sinc function, which will mean that we're suppressing higher frequencies better. We're messing up some of the passband frequencies in the process. So we probably don't want to go too far with that.

So I mentioned that the transform of a unit step is-- well, which we can easily prove but plugging it into the formula for the Fourier transform. And so what does that say? Well, that says that any time you have a step, you're going to have something that drops off as 1 over frequency.

And in images, of course, we're dealing with two dimensions. So this happens in both x and y. So if we have steps in the image, then they will produce some high-frequency content that doesn't drop off terribly fast. So there was a talk here many years ago by a famous mathematician from Harvard, who said that he'd taken a lot of images. And he'd taken the Fourier transform. And he showed that natural images have a power spectrum that falls off as 1 over f.

So what was the problem with it? Well, he was very happy to do this on many images, because he was using the fast Fourier transform-- makes it pretty efficient. Well, the fast Fourier transform, of course, is just an implementation of the discrete Fourier transform. And the discrete Fourier transform assumes that your data is periodic.

So if you have-- in the one-dimensional case, it says that if this is your waveform you're transforming, what you're really looking at is this, right? So what does this mean? Well, it means that unless you're careful, you're introducing a step edge discontinuity as it wraps around. And that will produce a frequency content that drops off as 1 over f.

So the grand result was not a feature of natural images but a feature of the discrete Fourier transform. So look out for that. So how do you-- this is a real problem. You see this again and again in papers. How do you deal with that?

Well, what you can do is make things match up so that various ways-- one is called apodizing, which is you multiply by a waveform that looks like this. And that means that at the ends, you've pulled the image down to 0. And so of course, it matches. And this is fairly common there. People have their names on these things.

But basically, the simplest one is just a inverted cosine waveform. And I won't write down the formula. It's pretty obvious. You just do 1 minus cosine of some suitable multiple of frequency and divide by 2. And you get this filter. And that has, actually, very nice properties. So that's one approach to dealing with a DFT applied to images.

Another approach is to say, well, I don't really know what's outside the frame. This is as much of the image as I got. And now, I'm being asked to guess what's outside there. Well, it could be like this. It could repeat periodically. But maybe it's a mirror image, right? Not very good-- maybe like that-- so and that's better, because now, there's no discontinuity in the brightness itself. So you don't get the 1 over j omega.

There is potentially a discontinuity in the derivative, right? So that means we need to look at 1 over f squared. Where's that come from? Well, if we integrate the unit step function, we get the unit ramp, we can call it. And its transform is, of course, the square of this, because basically, we've convolved the step function with itself to get that.

And so this is better-- a lot better, because now, it's dropping off as 1 over f squared. You still haven't solved the problem, which is because we're assuming that it's periodic. And so how can we avoid that? Well, we can take an infinitely wide picture. Then we don't have this problem. But that's, of course, not really an option.

OK, so we're going to look at some other examples of approximate low-pass filtering in 1D and then in 2D, and I just want to remind you of some basic properties that we need. One of them is the filtering property, the sifting property of the unit impulse.

Right. So the unit impulse is kind of a weird thing, because it's not a function, right, because we can say, OK, delta of x is 0 for x not 0. OK, I mean, the function is something, where, for every x, you can say what the value is. And for the unit impulse, we can do that for anything except [INAUDIBLE]

And then what happens at x equals 0? Well, it's infinite. So you can see there's a problem there. And the correct way of dealing with that is in terms of distributions. So there are these generalizations. And we need them. We use the impulse all the time. So we need it for that. But we also need them for derivatives of the impulse, which are even weirder.

And so someone once noted that aside from the Gaussian, there's just about nothing that has a Fourier transform, in the sense of as a function that you can write down. Almost everything interesting includes some kind of problem, some kind of singularity. And so we might as well get used to that.

So how do we deal with the peculiar nature of the unit impulse? Well, this is the way. This is how it's defined. This is its property. And it makes sense, because supposedly, this is 0 everywhere except where its argument is 0. And so the only place that matters is x equals x0. So the only value of f that matters is f of x0. And then the only thing that's left is the scaling.

Well, the idea is that we've made the unit impulse so that its area is 1. And one way I find very helpful to think about these functions-- I mean, the impulse is an obvious thing we all know about. But it gets messier. And so it's good to have a way of thinking about them.

So here's one way of thinking about it. Here's a boxcar with a unit area. And now, if I make epsilon smaller, it becomes narrower. And it becomes higher. And you can kind of think of the unit impulse as the limit of this.

But it's not mathematically correct to say that. But if you want to know what the effect is of, say, convolution with the unit impulse, you can take the convolution with this thing, which you can calculate. It's a perfectly valid thing. And then you take the limit as epsilon turns to 0.

So one thing to keep in mind, though, is that's not the only way of defining it, I could have equally well have taken a Gaussian, 1 over-- I probably have the scale factor wrong, something like that. And again, it becomes narrower and higher as epsilon gets smaller. And it defines the same thing in the limit.

So don't take this picture too literally. It's helpful. But it's not the only answer. And it may be the easiest one to compute. But OK, so what do we do with this? Well, we can play some games with combinations of impulses.

So for example, OK, oh, yeah. I need that. And so what are these? Well, these are just shifted impulses. And so one of them is at minus epsilon over 2. And the other one is at plus epsilon over 2. And their magnitude is 1 over epsilon, where the magnitude is the area under the impulse. It's what we get when we integrate it.

So that's the unit impulse, and then we have impulses of different areas here. So that's what that corresponds to. And of course, that's just f of x plus epsilon over 2 minus f of x by epsilon. And in the limit, that's the derivative.

So we can-- what's the point? Well, the point is we've connected convolution with derivatives, right. And so we can take the derivative by convolving with this. And that shouldn't be surprising, because this formula matches that, except this is flipped, right? You'd expect the right-hand one to be positive and the left one negative, because we're taking the positive of plus epsilon over 2 subtracting the 1 with minus epsilon over 2. Why is that?

Well, because in convolution, we flip. We flip before we shift and multiply and add, right? But the important point there is that there's a close connection between linear shift invariant operators and derivative operators. And we can treat derivatives basically as convolutions.

And just to refresh our memory-- right? So one of the two gets flipped. So this one is going left to right just as f of x does. And this one goes in the reverse direction, with respect to this dummy variable.

Notice, again, that we have to be careful with the x. We'd like to write the integral blah, blah, blah, dx. But we can't do that, because the x's parameter here. So we need a dummy variable. So introduce some Greek letter.

And of course, because it's commutative, this should also work the other way around. But in any case, one of the two has to be flipped. And that's how it's different from correlation. So correlation is exactly the same thing as convolution, except that you don't flip things. And that's why, up there, it looks a little odd, because we kind of expect the positive impulse to be on the right.

OK, back to something we mentioned yesterday, which is one way of dealing with the fact that the pixel averaging is not a very good approximation to a low-pass filter-- so in the camera, we're getting some advantage from the fact that we're not making point samples. We're taking an average over the area of the pixel. That helps.

But as we saw, it cuts off at-- somewhere we saw that it cuts off its 0. First 0 is twice the frequency where Nyquist says we should be. So we should really be adding some additional low-pass filtering. And no, you can't do it digitally after you've taken the image, because at that point, you've confounded these frequencies. All of these a.k.a.'s have collapsed into one. And you don't know how much came from one contribution and how much came from another contribution-- has to be done in the analog domain before we sample.

OK, so it's kind of difficult to do this, because we got 10 million pixels. And you're trying to do some analog operation that cuts out some of the high-frequency content. And I mentioned that one way of doing it was using these birefringent materials. And you only need a very thin layer.

So the crystal I brought in would deflect rays by a substantial amount, like quarter of an inch or something. And I picked the large ones just so you could actually see that effect. But here, we're talking about pixels that are maybe 5 microns wide.

And so when we do any averaging and deflection, we're talking about something that deflects by-- I don't know-- 5 microns, 2 microns, in that range. So the filter, actually, of the special material can be very thin. You just as to make sure you cut it in the right way in the crystallographic orientation. But that's their problem. Our problem is understanding what it does.

So it's a little bit like our formula for the derivative, except there's no minus, right? So what is this? Well, what we're saying is that we're going to have two shifted images. And if we just had a single delta function, then that would be the identity operation, except for a shift. And so now, we've got two superimposed images that are just slightly shifted the tiny amount.

And we can model that as convolution with two impulses, one of which shifts the image for symmetry so that the answer comes out as a real number. We like to make it symmetric. And so it's going to shift it to the right a little bit. And the other one shifted to the left a little bit. And that's what this special filter does so that when you look at the actual pixel array, it's no longer the original in-focus image. You've now got two slightly shifted versions.

And so how good does that work? Well, we take the Fourier transform, right? So we take-- and so that's going to be 1/2-- epsilon. And that's cosine of-- so that doesn't drop off with frequency. Well, it does near the origin. So let's see what it looks like.

With sinc, it dropped off with 1 over frequency. This just keeps on going. But it does drop off here. And what's the first 0? It's pi over epsilon. So if we want to-- how do we pick epsilon? Well, we have a lot of degrees. Basically, we can decide how thick to make this plate. We're not restricted to making epsilon equal to the pixel spacing or equal to 1/2 the pixel spacing. And Canon and Nikon have their own idea of what is the ideal value.

Anyway, what this does is now it cuts high frequencies. Unfortunately, there's this. Then there's some other frequencies that are not cut. What's the story with that? Well, the idea is that this is working with the other low-pass filter, the block averaging filter so that, yes, unfortunately, this extremely simple-minded thing that just makes two copies of the image has this property.

But that's OK, because when we get out here, the other filter's at a 0 so that the two together do a better job. I mean, I wouldn't say it's perfect. You can still get more effects when you have some high-frequency content in the image, but much less than you would without this filter.

By the way, of course, you need two of them, because you got x and y. So you actually have two of these plates, one rotated 90 degrees with respect to the other. And now, you're seeing four images superimposed on each other.

And but you can treat this totally separately in x and y. So this analysis doesn't have to be redone. It's just the same thing happens in the y-direction. And so we end up with an anti-aliasing filter that some people take out, because they don't like the way it suppresses high frequencies.

OK, so trying to tie this back to the patent-- so is this all bring back 603 or wherever you learned Fourier transforms and linear systems? OK, so what was the idea of the patent? So the idea of the patent was a little bit like the integral image. Is there some clever way of cutting down the computation, because to get good low-pass filtering, we tend to need large support?

And if you implement things the obvious way, the amount of computation grows linearly with the size of the support, or in the case of images, quadratically, with the size of support. So it's good to have a small support. And that doesn't mean narrow. It just means that you have only a few non-zero values.

And so let's-- so s is going to be our integral. And it's called s, because in the discrete version, we're going to have a sum. So what is it? Well, it's basically a step function-- looks like this. So we're interested in convolution with that. And that gives us so let's see what that does.

Right? So we just used the formula for convolution. We plug in this function. Or as we said last when we were there, we can flip them. We can interchange them, because convolution is commutative. And why do we do that? Because now, we can change the limits of integration. Oh, the-- sorry.

So this is going to restrict us to psi greater than 0. So the limits have changed. So we just end up with the integral. So well, there's a flip in sign, which corresponds to the convolution changing the order. So OK, apparently that was one too much.

All right, anyway, the important thing is that we already said how we can represent derivative in terms of convolutions. Well here's the way we can represent integration. And by the way, Fourier transform of the unit impulse is-- I forget whether it's 1 over j omega or 1 over minus j omega.

OK, so we have that operator. And then we have the derivative operator, which we went through somewhere earlier. And it has a Fourier transform that's j omega. And not too surprisingly, those are inverses of one another, because if I were to convolve the two of them, what I'm doing is making an operator that first integrates and then differentiates. And of course, that's the identity.

And in the Fourier transform domain, convolution corresponds to product. So in the transform domain, we get the product of this and that, which, of course, is 1. So that's consistent. OK, so the key thing, really, is that these operators can be treated as convolutions. And I can go to the second derivative. I can look at the second derivative, and I can do it in a number of different ways.

One is it's the first derivative convolved with itself. So I can take those two impulses and convolve them with another set of two impulses. And I get something that has three impulses, the standard pattern, 1 minus 2, 1-- something like that.

So this would be D squared. And I can obviously take higher order derivatives that way. And in terms of transform domain, I just-- convolution becomes multiplication. So if I do two of these, I'm just going to multiply this by itself. And I get minus omega squared. So the second derivative corresponds to minus omega squared. OK, so now, we're going to apply this to our problem of filtering.

OK, so suppose that we have a filter that we call F. We're trying to take it apart and put it back together in a way that reduces computation. Now, we said that the derivative of the integral is the identity. So we can certainly write things this way. And then we can make use of various properties like commutativity. So let's start there. And then we can use associativity to write it this way.

So what this says is that to convolve with the filter F, to apply the filter F, what we can do is instead apply this filter. Let's call it F prime. And we don't get the answer we want. But we can get the answer we want by integration or some agent from left to right, right?

And so why would this be interesting? Well, this would be great if this one was sparse. I mean, that doesn't always happen. But this is where we can apply this method in the case, where our function F is sparse.

And not F is sparse, but its derivative. And when does that happen? Well, it happens when it's a constant most of the time. So we're thinking particularly of splines. So we're thinking of filters that have some convenient waveform. But we're splitting it up into sections.

For example, here's an approximation to a Gaussian, say. And in each of these, it's a polynomial. And so some higher order derivative will be 0. So if it's an n-th order polynomial, I have to repeat this process n times, n plus 1 times, because the n plus first derivative of an n-th order polynomial is 0.

And then what happens is that I get 0 here, 0 here, 0 here. So oops. That's bad. Everything is 0. But you have something here at the transitions, right? So that's what I mean by sparse. So now, in the case of this spline of three sections, we have only four areas, where it's non-zero. And so the computation will be much more efficient. We don't get the answer we want. But we can get the answer we want by taking the result and summing it, integrating it.

And of course, if we've taken n plus 1 derivatives forward, we need to do n plus 1 summation. But the summation is trivial. It's just, you have an accumulator. You run left to right. You start with 0. You just keep on adding in the gray levels. And now, you might need to worry about dynamic range. How many bits do you need to represent that? But we won't worry about that.

OK, well, let's do a really simple example. So and again, typically, to do a good job, you might need fairly higher order derivatives. So suppose, for example, you were trying to approximate the sinc function.

That would be very important to us, because we know that's the ideal one. If we could ever get the sinc function, we'd have a perfect low-pass filter. But so let's approximate it. And what's wrong with the sinc function the way it stands? Well, the main problem is it goes on forever. And so we need to somehow truncate it. And we need to represent it.

And how close an approximation do we want? That's not very symmetrical. So let's-- OK, so here's how-- it looks a little bit like the sinc functions, not, of course, going on forever. But we can approximate it this way. And we probably want a cubic approximation.

So in each of these four areas, the curve is described by a cubic. And there are different cubics for the different areas. And that's the whole idea of a spline. And now, if we-- it's cubic. So if we take the fourth derivative, it'll be 0 everywhere except at the junctions.

So that means that if we do this, use this for filtering, we can do relatively little work, because instead of having-- say the pixel spacing is like that. Instead of having all of those non-zero values, we only have non-zero values at those four places.

And this, by the way, is used heavily in all kinds of image manipulation software, like Photoshop would interpolate images using this function, except in 2D. So it's called-- in 2D, it's called by bicubic, because we're doing cubic in x and cubic in y. But they didn't consider a true 2D case, because it's more work.

So OK, so say, for example, you take your picture. And you discover, oh, I had the camera rotated 2 degrees. And then you say, OK, Photoshop or whatever your favorite image manipulation software is, rotate it by minus 2 degrees. Well, of course, now, the pixels of the rotated image won't line up with the pixels in the new image. And so you have to interpolate. You have to figure out what values to use. And you could just use nearest neighbor.

Well, nearest neighbor is pretty bad. You can see the pixelation. And in particular, if you repeat it-- say you then say, oh, actually, it was 2 and 1/2 degrees. Let me turn it another 1/2 degree, which is probably a bad idea. You want to go back and turn 2.5. But suppose you do it sequentially, then at each step, it's going to deteriorate a little bit. And if you use nearest neighbor, it'll deteriorate a lot.

So the next best thing is linear interpolation, which is cheap to compute. But it's not as good as we'll see in the moment as this method. And then you might say, well, why not go even further? Why not approximate the sinc function by something that looks like this-- have one more wiggle in it?

And you can do that. But as far as I know, this has never caught on because of the computational cost. So if you think about doing this in 2D-- so we need to look at four places in 1D. In 2D, there's a grid of 4 by 4. So we need to look at 16 neighboring pixels to do a good job.

Well, with this one, we need 1, 2, 3, 4, 5, 6, 7. Is that right? Anyway, more-- and by the time you square it, a lot more. So and there's not a huge payoff. You don't get images that suddenly look dramatically better bicubic. And bicubic was invented by IBM. And it was used in early satellite image processing.

OK, so suppose that we go back to the simple block averaging just so we can do the numbers more easily. No, I don't want to do this one-- do another one. So again, 1D case-- and now, suppose we want a block average of four.

Well, let's start with-- there's one issue. We have to make some assumption about what's outside the range that's given to us. So let's assume it's 0. And at some point when we're here, we'll get 0. And then we shift 1 over. And we take that block average and get 2. And we shift 1 over.

We get 3 and 6. And then we get-- better write it down from here before I make a mistake-- 11, 18. No, that's the sum. Sorry. That's the wrong thing-- 11, 16, 16, 13, 10, 6, 7.

So that's the naive implementation of convolution. I just shift this block along and add up whatever's underneath the block. So then let me do it the other way, which in this case, doesn't save us a lot, because it's a very short example.

So the other way is to add left to right-- 2, 3, 6, 11. And then 7 is 18 and 19, 19, 21, 24, 26, dot, dot, dot. And now, I subtract. So I'm going to take this value and subtract the value four back. So I get 2.

Subtract this value. I get 3. Subtract this one, I get 6, 11. And then by 18, I get 18 minus 2 is 16. And 19 minus 3 is also 16. And 19 minus 6 is 13. And you can see how that works.

And in this example, it didn't buy as much. But if we're block averaging over a large segment, that could be a huge saving. And if we do it in 2D on images, even more so. OK.

AUDIENCE: How do we get [INAUDIBLE]?

BERTHOLD HORN: Because there's a-- we've got-- where were we?

AUDIENCE: [INAUDIBLE]

BERTHOLD HORN: 18. Oh, up here?

AUDIENCE: [INAUDIBLE]

BERTHOLD HORN: We're adding 0 at some point. These may not be lined up exactly-- oh, did I screw up? OK, 2, 3, 6, and-- so this one is 16. This sum is 16. And then the next sum is also 16. You see the-- we lose a 1 on the left then, but we add 1 on the right. So God for a minute there, I thought the whole method was flawed. And we'd have to give up and fire the person who wrote that patent. But no, fortunately, it works.

So OK, I don't think I'll do another example. That's too boring. So linear interpolation-- so I brought up cubic interpolation and sang its praises. So I should say something about linear interpolation. And we don't think of linear interpolation as having anything to do with convolution. But of course, we can think of it that way.

So again, suppose we have samples on the street grid. And then we create a function that connects those points. And the simplest way is just to draw straight lines. And that's linear interpolation. Over there, we draw qubits.

OK, so that's the formula that gives us that straight line. So first of all, notice that it's linear in x. So it's a straight line. And at the important points, namely 0 and 1, it gives us the right value. So at 0, we get-- at x equals 0, we get f of 0. That's what we want. At x equals 1, we get f of 1. So that is that line. OK.

Now, that corresponds to convolution with that function. And I'll leave you to figure that out. It should be clear after what we've done. And that, in turn-- this triangle, in turn, is the convolution of those two boxcars.

That's also easy to see, because we take one of them. We flip it. That does nothing. And then we slide it underneath the other one. And initially, there's very little overlap. The overlap increases linearly until we're right on top of it. And then it linearly decreases until we drop off the right-hand end. So convolution of the two boxcars is that thing.

So by the way, you can see that, in some sense, right, just by looking at this, it's an inferior interpolation method to the one that's using that approximation to the sinc function. And it has a number of nasty properties. One of them is that the quality is different close to the sample points than, say, in the middle between them. Why is that?

Well, suppose there's noise in the image. Then if I'm here right on top of the sample point, I just inherit the noise of that image measurement. Let's say it's sigma. So there's some sort of variability sigma. Now, if I'm halfway in between them, I'm taking the average of these two values. And so there's noise in this. There's noise in that.

But the variance of the noise adds. So what I find is that I'm getting sigma squared plus sigma squared for my variance, which is 2 sigma squared. So the standard deviation-- so this is the variance. So the standard deviation of the square root of 2 times sigma-- so the noise in the interpolated result is different when I'm different places in this transition.

Now, that's also true of the cubic interpolation. But it's much smaller effect. In the case of linear interpolation, you're averaging two independent noise values when you're in the middle. So you have reduction in noise. But when you're sitting right on top of the original samples, you're getting the full effect of the image measurement noise. So it's an effect that's kind of annoying.

OK, so in terms of transform, we know that this transform is sinc. And this transform is sinc in the frequency domain. And so we get sinc squared. So and we've talked about sinc squared. So this is not a terribly good low-pass filter. It's better than a single stage. It's better than-- oh, so I mentioned that the crudest method is nearest neighbor.

So an interesting question is, what convolution does nearest neighbor correspond to? So let's think about that. So we've got the cubic one. And we've got the linear interpolation. And what about nearest neighbor?

Well, with nearest neighbor, what my picture would look like is-- let's see. Suppose the next value is up here. So with nearest neighbor, I'm going to get these steps. So it's a piecewise constant. Over here, we have something that's piecewise linear. And in the case of the cubic interpolation, we have something that's piecewise cubic.

And you may ask why there is no piecewise quadratic. But that's a long story. So we don't do that. So that's the result we get from-- and you can see that this is obviously inferior to even linear interpolation. And then so nearest neighbor corresponds to convolution with the boxcar. If you think about it, that makes sense. You think of each of the samples as an impulse. And now, we're convolving those with a boxcar.

And that will create a box like this and a box like that and a box like this. So each of these impulses creates a box. And you put them together. You have the linear interp-- the nearest neighbor interpolated result. And so in some sense, the linear interpolation is doing that twice. It's twice as good, in some sense. It's the quadratic in the transform domain.

OK, so as long as we live in image coordinates, x and y, we can treat them independently as we have, for example, with interpolation. We said, well, we'll use a cubic spline in x and a cubic spline in y and so on. But we keep on saying that really, things should be-- rotation is symmetric. There's no preferred direction in natural images. And that's not entirely true, because we often line things up with a horizon.

And then we have man-made structures, which are rectangular often. But in some sense, they shouldn't really be some special bias. Why did we pick a square pixel or a rectangular pixel coordinate system? And so what if, instead, we're thinking about a rotation asymmetric situation? And one of the questions that comes up is, what's the equivalent of low-pass filter?

So we can easily imagine a low-pass filter like this. So u and v are the frequency components corresponding to x and y. And here, we've got a low-pass filter in the x-direction and a low-pass filter in the y-direction. And that's a legitimate way of thinking about limiting the frequency range, but it's not rotationally symmetric.

So more interesting question is-- do I want to go-- yeah. What if we have a low-pass filter that's rotationally symmetric? What is the real two dimensional equivalent to what we've been talking about in 1D?

And so this function-- I'll call it a pillbox. I guess nobody knows what pillboxes are anymore, although old people these days often have some sort of pillbox. So they remember whether they've taken their medications. But those boxes are rectangular. That's not what I'm talking-- I'm talking about a round thing.

Viewed from this-- perspective view would be like this. So its value here is 1. And its value out here is 0, right? It's passing all those low frequencies. And it's not passing any high frequencies. And you can find its transform. But I'll just write it down, because it's somewhat tedious.

So actually, inverse transform, right, because we're going from frequency domain to the other domain, except the transform's pretty much symmetrical. And we have a rotationally symmetric object. So in that case, it doesn't matter at all.

OK, so what's rho? Well, that's in a polar coordinate system in the spatial domain. And you'd imagine that the inverse transform of this should be rotationally asymmetric, as well. And it is. And if you go along any radius, it will vary as this ratio, where J1 is the first order Bessel function.

And this is a famous function in optics. For example, when Airy first talked about the resolution of a microscope, he found out that its point spread function, its response to an impulse, is that. And then his famous equation for resolution is based on the first 0 of this function.

So this is the equivalent, in some sense, of the sinc function. And if we plot it-- now, if I plot it on the blackboard, it'll look exactly like the sinc function, because it has a central peak. And it goes to 0 and then comes out the other side and dies away. And you'd have to actually look at the values to see that it's different.

So there are two ways to, in which it's different. The one is that the first 0 is 3.8371, dot, dot, dot rather than at pi. And the zeros are not multiples of that first one, although they tend to be closer and closer to periodic.

And the other thing is that it drops off-- so in the sinc function, it drops off as 1 over f, right, because we got the sign at the top that's always between plus 1 and minus 1. And we're dividing by the frequency. So the sinc function drops off as 1 over distance. This drops off as 1 over something to the 3/2, right? I don't have it. Well, so it drops off actually a little bit faster than the sinc would have.

So that's the ideal low-pass filter for an image. And that's in 2D. And we've only shown the cross-section of it. But its rotation is symmetric with this cross-section. And so when you look through a microscope, for example, at a single point of light and you were to plot the electric field in the image plane, it would look like that.

Now, negative values, brightness-- well, the thing is we don't sense the electric field. We sense power. We sense the square of it. So in terms of image brightness, we actually have the square of this. We get that and so on.

And Airy's resolution criteria is based on this first 0. And it says, if you've got two objects that are this far apart, you can tell that they are two objects sort of, because the second object-- its blob would be something like that. And it's kind of obviously arbitrary. But it's a useful number to have in mind.

OK, so that's sort of neat. And it leads to something else, which is the Fourier transform is symmetrical, pretty much, forward and back. So here, we've taken the-- let's see. We've taken the inverse Fourier transform of something in the frequency domain. And we've got this Bessel function thing.

Now, when we're talking about defocus, we were talking about pillboxes basically. So let's do that. And now, that's in the spatial domain. We said that if your image is out of focus, you're convolving with a pillbox in the spatial domain, right? Remember, we had the lens. And the lens brought light to a focus. And unfortunately, our image plane was in the wrong place. So we cut that cone of light. And we then get a pattern like this. And the radius depends on how badly we are out of focus. Let's call it R.

Now, this time, the value inside here is not 1. It's 1 over pi R squared. Why? Well, I want the integral to be 1, because basically, all the energy that used to come to this point is now spread out over this area. And if I go more out of focus, it's spread out over a larger area. So I should really normalize to that area.

Now, because of the symmetry of the Fourier transform, we can pretty much just write down the answer, because if the pillbox in the frequency domain turns into that Bessel functions thing in the spatial domain, then a pillbox in the spatial domain should turn into that Bessel functions thing in the frequency domain. So that's very important. This is what defocusing does.

OK, and you can see that we have a decrease of high-frequency content. But also, we actually kill some frequencies completely, right, because of these zeros. So if I were to plot the energy in the frequency domain, I would expect to see an area, where it's actually 0, and another area, where it's actually 0 and so on.

And this works. So we had a Euro student take a DSLR and take pictures of-- I don't know if-- the memorial to the fallen in the center of building 10. And then he commanded-- now, this is a Canon that allows you to control the lens. So he commanded the lens to take a step, took another picture, commanded the lens to take a step, took another picture.

And so he got a whole stack of pictures that are in different degrees of defocus. And one of them is pretty much in focus. Now, unfortunately, Canon doesn't tell you how big the step is. So part of his exercise was, can you figure out what the step size is? Well, you can if you can figure out how big these pillboxes are. And how did he do that?

Well, he looked at the frequency domain. And in the frequency domain, once you know where this ring of zeros is, you're done. You can calculate the radius. You can calculate R. And so he was able to determine how R changed as he incremented the lens position. And then knowing the aperture of the lens, he could calculate the step size.

And OK, well, it's not quite that simple, because there's noise. So it's not like you can actually expect this to be exactly 0. But if you just plot this as a picture, you look at it as a picture, there's a dark, dark ring. And then there's another dark ring. And then you can eyeball it.

There's another effect, which is that in an ideal lens, this is exactly what we would expect to get. But in practice, as we mentioned, lenses always have some compromise between different problems-- chromatic aberrations, spherical aberration, astigmatism, and so on.

So the actual point spread function isn't a pillbox like I've drawn it over there. It's a little different. And that was part of the exercise. Find out how that changes. And interestingly enough, it's different below focus than above correct focus.

So in our ideal model, of course, you would expect exactly the same defocus effect if your epsilon below perfect focus than when your epsilon above perfect focus. And in practice, that's not quite the case. If you look at the cross-section of the pillbox instead of-- this is the ideal pillbox cross-section, of course. And if you look at it on one side of defocus, it's more concentrated near the edges and not so much. I mean, for a start, it's not a perfect step edge.

And then on the other side of defocus, it looked more like that. So you could imagine that that led to some difficulty in actually implementing this idea. But it's a first approximation. So here's an interesting idea. Suppose that you get a picture. And you want to know whether it's the original or whether someone printed it out and then took a photograph of that. And we'll get back to that in another way.

I mean, one of the things that happens is that we get perspective projection. So we print it out. Now, we take the camera. We get perspective projection again-- is the effect of two perspective projections in sequence. It's the same as a single perspective projection. And it turns out it isn't. So you can tell, as we'll see, that there's been some cheating going on, that someone modified the picture and took another picture of it.

But here's another way. Suppose that it's slightly out of focus. And then you take a slightly out of focus picture of that slightly out of focus picture. Is that equivalent to taking a picture that is more out of focus, like the sum of the two out of focus values?

And so what are we doing? We're convolving with the point spread function, which is this pillbox once. Take the result. And then we convolve again with the pillbox. And that's equivalent to multiplying by that and then multiplying by that, right, because convolution in the spatial domain corresponds to multiplication in the frequency domain.

And this is not-- this product is not equivalent to J over something for any value of R. Or if you just look at the frequency domain, if you pass it through both of these operations, you will lose all of these frequencies. But you will also lose all of the frequencies of the other operation, which could be here.

So it might not be trivial to use this method. But conceptually, at least, there is a difference between a defocused picture of a defocused picture and any single defocus step. And you should be able to tell that something fishy is going on.

So oh, OK. So just a note-- so I didn't say a lot about the patent, because I thought it'd be more important to understand the linear systems theory behind it. And when we're done with that what we're going to do next is photogrammetry.

Photogrammetry is about using images to take measurements. And it was an early application of photography. You send someone up in the hot air balloon. And he takes a picture. And maybe he takes a picture from a different position. And then you can create a map. You can actually get three-dimensional information. So we'll be talking about using images to create, particularly, 3D information.

So it's kind of going up. We've been working most of this time at what some might consider very low level. We're dealing with pixels, gray levels, radiance, irradiance, and so on. So now, we're going up. So we will assume that we know how to get certain features, like edges. And then we're going to try and match them up between images in order to get information about where the camera was and information about the objects. So OK.