[SQUEAKING]

[RUSTLING]

[CLICKING]

BERTHOLD HORN: This quiz one, which will be out today. And the rules for that are slightly different from the homework problems. It counts twice as much as a homework problem. And it's longer. Not twice as long, but something like that. And so you have also a little bit more time to do it. It's not just one week. But it's a bit of work, so don't leave it too late. Please start on it fairly soon.

And this is where you're supposed to work on your own, not collaborate. And it covers whatever we did up to this point, with a bit more emphasis on recent material. And I guess only the last question has to do with the patent we're discussing, so should be able to deal with the other four right away.

Little sidebar here about intellectual property, and starting off with the fact that some people don't like that terminology. And how can an idea be property? A little bit like, how can a company be a person? Well, it's an awkward thing in law that you have to come up with some way of formalizing these things. And so in this case, ideas are treated as property to some extent.

There are several different types of intellectual property. First of all, we have patents, which is what we'll be talking about. And there are different types of patents. The so-called utility patents, ie, useful patents. Not to say the others aren't useful, but they're design patents. So for example, Jerome Lemelson's first patent was basically a design patent for a baseball cap with a hole in it and a tube that you could blow through. So this is the inverse of the beer baseball cap. And there would be a propeller, and as you blow through it, the propeller would turn.

Another famous design patent is Apple's design for a cell phone. And their design patent says that it should be a rectangle with rounded corners. And there was a lawsuit where they sued Samsung because their phone with a rectangle with rounded corners. And the jury awarded Apple a billion dollars, so big money involved here.

And you might say, well, what should a cell phone look like other than this? This is it. But it's not completely crazy because at the time they invented it, phones were these big concrete things you held in your hand. Anyway. Not to argue about the merits of this. And of course, it's being appealed, and so in the end I don't know what'll happen. I guess some large amount of money will change hands, an amount of money which is astronomical to us mere mortals, but probably for these companies it's small change. So we shouldn't feel too sorry for them.

So patents. We mentioned last time this was the social contract where you explain exactly how to do something in return for having a limited monopoly that lasts a certain number of years. And the rules change slightly as time goes on. By the way, you are supposed to explain how to do it. And in particular, if you know a good way of doing it and in your patent you only describe a lousy way of doing it, that could be grounds later on for invalidating a patent. And that's called best mode.

So there are lots and lots of terminology. And in the litigation, there are standard things that come up. One of them is best mode. So in one machine vision patent for a wheel alignment-- so that one has cameras and LEDs, and it determines the axis of rotation of the wheel and the axis of steering that it turns around when you turn your steering wheel.

And the patent that was claimed to have been infringed did not disclose the best mode. It disclosed the method that we see actually implemented. It gives you the correct answer if you have perfect measurements. But with realistic measurements, it would have an error of a degree or two, and that's not good enough. If you have a car like a BMW, they specify those angles to 0.1 degrees. Anyway. So they had a problem there because they did not disclose the best mode.

Utility patents. And we talked about the structure of those. I'll just briefly talk about some of the others. So copyright. You can protect artistic expression using copyright. So if you write a book, there's a copyright on that. If you record a song, there's a copyright on that. If you choreograph some ballet, there's a copyright on that.

And there are exceptions. So for example, if I want to talk about some topic, I am allowed in class to present a portion of that material without violating the copyright, without having to ask the author, how much do you want from me to use your product? There are also exceptions where you use extracts. So for example, if you're writing a news article, or say you've got a blog about a movie and there's a particular part of the movie that you'd like to talk about, under certain circumstances you can clip that out and use it.

And of course, you can imagine, there's a lot of fun that lawyers have with this because a lot of music is extracted from other music and put together, and is that legal or is that copyright infringement? Copyright used to be basically for the lifetime of the author. If you write a book, you should benefit up to a certain point. When you're not around, then you don't benefit. Well of course, the heirs of famous authors didn't like that, so that was changed.

And now we have the Sonny Bono system of copyright. Basically, he was the person who convinced Congress to change the laws, and now it's author's life plus 75 years. And they've been periodically updating it. So it's basically author's life plus infinity, because every time we get close to the limit, they say, oh, well, let's upgrade these poor heirs. They can't live without the royalties from this. Excuse me if I'm being sarcastic.

So that's copyright. And by the way, that was very important for a while, and still is, in software. Because again, the rule was you couldn't patent mathematics or an idea, abstract idea. And so the courts typically held that that's what programs are. They're ideas. They're abstract stuff. They're not-- you can't hold them and weigh them.

And so the way people protected themselves was to send their programs into the copyright office and register them there. And you can imagine that, say you have some operating system like IBM 360 operating system and you send in your 750 million lines of code to the copyright office. That was quite an exciting event. And certainly, if someone makes an exact copy of your program, then you can sue them under this law.

Of course, with all of these, there are people who are working to get around these laws. And so in the case of copyright, there was this notion of a clean room. So what you would do is you would have a bunch of people that understood programs well analyze what someone else's program is doing, kind of reverse engineering it.

But they weren't allowed to show it to another group of people. They were only able to tell them what it does. And then this other group of people would write the code. And supposedly, because they didn't see the original code, there was not a copyright violation. I'm not sure that's-- and it's not totally unreasonable in that, if you look at the code line by line, it's likely to be totally different. They'll use different variable names and stuff.

Then there's trademark. So trademark is much more restricted. It's just, if you want to call your company Dunkin' Donuts, you can trademark that. But it has to be unique in the field, and it has to not be-- there are a whole bunch of rules. But basically, you can't use a common word, like you can't call your company Time Space because those are common words. And so a lot of company names are slightly misspelled versions of common words.

The other thing is that the trademark may include particular shapes, particular markings-- so you can distort some letter-- and color. So Dunkin' Donuts has two colors that they've copyrighted, and I'm proud to say I have hats that have both of those colors. They're very useful in hunting season, which is approximately what's happening right now where I live. So you can use color to protect yourself.

They have to be unique to the field. So you may have a trademark on a name in the rubber industry and someone else has it in the semiconductor industry. They don't conflict. And there was a famous case about that where-- again, Apple-- Apple sued the Beatles. Why? Well, because when the Beatles started, they formed their own company called Apple. May not know that.

But in any case, Apple sued them. And they lost because there's no confusion of the client, the customer. They're in two totally different fields. One is music, and the other one is computers. And of course, Apple claimed, oh, well, but we distribute music, so it is the same thing. Well, whatever. Anyway.

Then, the last thing is trade secret, which is no protection at all. You just hide what you're doing. You don't tell anyone exactly how you're doing it. And classic example of that is Coca-Cola. There's a safe down in Atlanta, Georgia, which has the formula for the ingredients in Coca-Cola. And supposedly, very few people know what's in there.

And the danger, of course-- the good part is that it's unlimited. There's no expiration date on patents. And the bad part is, if it ever comes out, there's no recourse. There it is. Everyone now knows what to mix up to make Coca-Cola. So it's a risky thing, but it's certainly much cheaper than pursuing some of the other avenues. You don't have to pay lawyers for it.

And there is a certain legal recourse. If somebody signs a non-disclosure agreement when joining your company and then they walk off with the formula for Coca-Cola and tell Pepsi Cola how to do, it then you can sue them. But the cat's out of the bag. It's not so easy to recover from that kind of loss.

Just a quick overview of what's called intellectual property. And Richard Stallman would definitely complain that I use that terminology because he doesn't think that ideas should be treated as property, just as some people don't think that companies should be treated as persons under the law. But there you are.

Back to the particular patent. So this particular patent is real low level. I wanted to start with something very simple, finding edges very accurately. Finding edges very accurately. So where are we going with that? Well, the idea is that once we found edges and we're describing images using edges, we can do more elaborate things like recognition and determining the position and determining the attitude of an object.

And most of what we'll be doing after finishing with this patent is in 2D, where-- the world, of course, is 3D, but there are lots of cases where 2D machine vision is incredibly successful. And one thing that's important to point out is that it has to be incredibly accurate. If your thing works 70% of the time, forget it. It's got to be working 99.999% of the time.

So these methods are very carefully thought out and attuned to have extremely good performance. And what we'll find is that in the 2D world, this is possible. It's a little harder once you get to 3D. So once we've done position and attitude in 2D, we'll progress to 3D, which is a more interesting problem. But let's get some of the basics sorted out.

So I'll just quickly review what this patent did. And so we'll start-- so this is-- and the first idea was to look at the brightness gradient, and so just a quick story on that. So here we've got brightness as a function of x. And there's a point where the curvature becomes 0 and changes sign, and that's called an inflection point. And that's what we're looking for.

So the edge is actually spread out over several pixel, but we're trying to identify very accurately a particular point on the edge. Then we looked at the derivative, and there we're looking at a peak-- we're searching for a peak. And some methods use second derivative and look for 0 crossing. But importantly, this is in the gradient direction.

So this is, of course, a 1D cross section and we're dealing with images, so how do we take the cross section? Well, we're only interested in the direction that is perpendicular to the edge, and so these graphs and these ideas correspond to that. Then we talked a little bit about things that are sometimes called stencils and sometimes called computational molecules.

And what we're trying to do is, in the discrete world, we're trying to estimate derivatives. And so, of course, there are some obvious ones. So there's a way of getting an approximation for e sub x. And here's another one. And these are all some of the ones that we already looked at.

So there are lots of ways of estimating the derivatives. And how do we know which one to choose? Well, there are trade offs. And we saw that one way to progress is to use Taylor series expansion to see what the lowest order error term is. Because the higher we can push that, if it's a third order error term, it's better than if it was a second order error term.

And if two methods have the same order of error term, then we look at the coefficient. And if it's lower-- like these two have the same lower error term. And also what was very important was, where are we trying to estimate the derivative? And we decided that we get the best results at those particular points.

Now, some of those points are offset by half a pixel from the pixel grid, and that's why people don't use them. But that's silly, because what's wrong with having some quantities on the pixel grid, the image, and some quantities on the grid that's offset by a half? As long as you know that it's offset by a half, you can translate any result on that grid into a result on the other grid.

We can also analyze these in the Fourier transform domain in terms of how they affect higher frequency content. And we won't do that right now, but that's second set of methods we can use to decide which of these is better. Now, these derivative estimaters can become quite complicated if you're looking for high precision.

So for example, in analyzing muscle electrical signals, there's quite a bit of noise and there's quite a lot of distortion of the signal as it progresses through tissue. And so when people are trying to estimate the first derivative, in one case, one paper I saw use 39-degree approximation. So they use a pattern like this that's 39 elements long, and they feel that that way they can control the trade-off between suppressing the noise and getting very accurate results.

So we're using very low order, partly because it's easier, and partly because one other trade-off is if you make them too long, then they start to have different features interacting with each other. So here we are trying to detect an edge. Now, if you use an edge operator that's 100 pixels long and there's another edge within that 100 pixels, it's going to interfere with the results.

So we try to compromise. On the one hand, we get better results with bigger support, better noise suppression. But at the same time, we run more into the problem of what, happens when two edges get close to each other? And in particular, here's an image of the corner of a cube. And over here, we could have potentially quite a large support, but when we're back here, edges get pretty close together. And then a large support means that you're combining information about different edges and you won't get the results you would like.

That's first derivative. And if our model is we find the peak in the gradient direction of the first derivative, then that's all we need. But for some purposes, we might want higher order derivatives, and we'll see some examples of that later. So now, one way to think about this is that second derivative is just the first derivative applied twice. And so we can run our numerical approximation a second time. And that corresponds to convolution. And the result is--

So that we can easily compute second order derivatives this way. Let me just make sure we understand what this is. So these computational molecules, the way they work is that you put them down on the image grid and then you multiply the gray levels by whatever the weight is at this point. So multiply that by 1. Put it in an accumulator.

Take that one, multiply it by minus 2. Add that to the accumulor. Take that one, multiplied by 1, add that to the accumulator. And then, finally, take the result and divide by epsilon squared. Well, often we don't care about constant factors, so we might drop that. But if we're actually thinking about derivatives, we should do that.

So that's one thing. And we're really talking about convolution. So we apply this to every place in the image-- slide it along if you like-- to produce a new image. It results at a bunch of points. Now, how do we know that if we want a sanity check? Well, a way of making this-- making a check on this is to try it on some function where what the answer is.

So we know that, in this case, the answer should be 2. So let's apply this. Well, we have to decide where we're applying it, so let's suppose that this is 0, this is 1, this is minus 1. So we're plunking this down. And so what do we get? Well, in this case, epsilon is obviously 1, so that's 1 times-- and then 1 times minus 1. Well, that's 2. No, sorry. 1 times-- yeah. 1 times minus 1 squared is 1.

Then we have minus 2 times x squared with 0 and plus 1 times 1 squared. And so that's 2. So apparently it works. So there are different ways of designing these computational molecules. Certainly, convolution is one. And then you'd want to check them. So another thing you might want to check is, well, what if f of x is x? Well, then we're going to get 1 times minus 1, 0, 1 times 1. The answer is 0, which is what it should be.

What if f of x is 1? So you want to check it for polynomials of low order up to the order of the derivative you're trying to get. So if f of x is 1, then of course we get 1 minus 2 plus 1 is 0, which is again what it should be. It's the second derivative is 0. So one of the constraints on these operators if they're supposed to be derivative operators is that the weights add up to 0. That makes sense.

So that's just one dimension, so to speak. What about d2, dx, dy? We can just do that. So because this is the x derivative and that's the y derivative and composition of differentiation corresponds to convolution-- so we're sneaking in some stuff here.

One of them is that we're used to dealing with linear shift invariant systems. But it turns out that, of course, derivative operations are linear shift invariant, in that if I take the derivative of a function and then take the derivative of the same function shifted, what I'm going to give is the derivative shifted. If I take the derivative of the sum of two functions, I'm going to get the sum of the derivatives.

So that's a very important thing to exploit, is that taking derivatives can be considered convolution. And so all of the good stuff we know about that applies. So let's see what this gives us. Now, what we're going to do basically is take one of these and flip it and superimpose it on the other one. And if I superimpose it over here, of course, I get 0 because I'm multiplying-- we assume the background is 0. We assume that the only values we're showing are the non-0 values.

But then what if I move it here? Well, then there's an overlap. And then I move it over there. There's an overlap. And I can move it down here. There's an overlap. And I move it down here and I get-- so I expect to get four values out of this convolution. And so I'm going to get something like this. And so my 2 by 2 stencil for estimating the mix derivative is just that, and it makes perfect sense.

One thing to watch out for is that, in convolution, we flip one of the two functions and then slide it across and try it in all possible places. Over here, sometimes we're using computational stencils that are not flipped, so we may end up with some sign reversal. But you want to-- you can always check on a simple polynomial to see whether it's working the right way around.

By the way, this one here, we can take a diagonal view of this. It's a little bit like, if I project this down-- so there's a plus 1 far out on this side. There's a plus 1 far out on that side. And then there's a minus 2. There are two minus 1's in the middle. And this looks awfully like a second derivative, like the one we had over here, just rotated 45 degrees.

And of course, this mixed partial derivative is in a rotated coordinate system, just d2, dx squared. So if I take-- so this is my original coordinate system. And then I look at the world in this coordinate system. Then the second derivative, ex prime, x prime, is the same as that mixed derivative. So we think of exy as a very different animal from exx and eyy, but it isn't.

Just a little bit more of this before we go on. And we already mentioned the Laplacian. Let's bring that in here again. So we had a second derivative operator here. And so in the case of the Laplacian, we're adding two of them in different orthogonal directions. So we could just do-- so that's just this thing plus the same thing rotated 90 degrees. And so that's one way of writing the Laplacian.

But that turns out to also be a good approximation to the Laplacian. How do I know? Well, there are lots of ways of checking. One is Taylor series. Another way is apply it to test functions, see if it gets the right result. And another way is Fourier transform. But those-- and you'll see that this is really the same pattern rotated 45 degrees, but now the separation is square root of 2 times as large. So I end up having to change the weight.

Now, I mentioned that the Laplacian is the lowest order linear differential operator that's rotationally symmetric. Well, neither of these looks particularly rotationally symmetric. So can we make one that's a bit more-- on a square grid, you can't, but we can do better than these. And one way is to combine them, to take away the sum of these two.

So I've taken-- I don't know-- 4 times this one plus 1 times this one, and I get this operator. And it's a little bit smoother in terms of rotational symmetry. The corner one's on 0, but they also don't have the same weight as the diagonal one-- the up and down ones because they're further from the center. So how do I know that's a good combination? How did people get to that one?

Well again, you look at the Taylor series, and it turns out that the lowest order error term for this one is one larger than the lowest order error term for either of these two. So it's better. More work. If you apply this to the image, you're doing not quite twice as much work, but you're doing more work because you have to take into account all of the corner ones, which this operator doesn't.

Or conversely, the up and down and west and east ones there. And if we wanted to, we could do a more detailed analysis of the error terms, but that's pretty boring so we won't do that. Anyway. So those are all of the computational molecules we're going to need. And as you can see, they're options. They're different versions that are trying to compute the same thing.

Of course, on the hexagonal grid, this looks much better because we can do something like this. Which looks nice and rotationally symmetric. Yeah?

AUDIENCE: What's the one in the middle of the--

BERTHOLD HORN: Oh. Minus 20. So can anyone think of how you could determine that it should be minus 20 without actually doing a lot of hairy algebra?

AUDIENCE: They have to sum to 0.

BERTHOLD HORN: They have to sum to 0, right. Why is that? Well, because when you apply this operator to f of x equals 1, [INAUDIBLE] Laplacian is 0. And so if you apply this to the function that's 1 everywhere, obviously you'll just be adding all the weights. And so they need to add-- how do I know it's 1/6? Well, one way to get it is to apply this to a test function like f of x is x squared plus y squared.

Well, I know that the answer is 4. And then go from there. Or I can get it from this argument, which is the weighted argument that I'm taking 4 times 1 of those and 1 of those. And this one has a factor of 2 in there. So you end up with-- I don't know-- a 1/2 plus a 1/4. No. Anyway, it comes out to 6 epsilon squared.

So it's annoying that we don't have hexagonal pixels. By the way, there are some situations where people are very concerned about efficiency, like trying to image the black hole at the center of our galaxy using radio frequencies. Each antenna you put up costs a pile of money, so you want to make sure that this grid of antennas is the most efficient way of sampling the Fourier transform space in that case.

And so it turns out that this way of sampling is 4 over pi as efficient as that way of sampling. So there are places where people do this. And for example, briefly, almost all chips are laid out on a rectangular pattern because that's very easy to do and check. But if it comes down to packing density, and particularly if you have something that has a very simple repeating pattern, then sometimes there's an advantage. So there were memory chips for a while that used the hexagonal layout, but they've since disappeared because now we're stacking things vertically. Right now, it doesn't seem to be an efficient way to go.

Oh. So while we're here, I should mention-- I mentioned already that the Laplacian is the lowest order linear operator-- differential operator that's rotationally symmetric. Here is a non-linear one. So this operator is rotationally symmetric. What do I mean by that? I mean that if you rotate the coordinate system, for example over here, and you compute e sub x, dash, and square route and add e sub y dash and square route, you'll get the same answer as if you take-- so it doesn't depend on the orientation of the x-axis.

And so this is lower order, of course, in Laplacian because it's first order, but it's not linear. Nevertheless, we run into this quite a bit because, remember, Roberts used these stencils, which you can just think of as ways of computing the derivatives in the rotated coordinate system. And then he took the square root of the sum of squares, and that was his edge detector. And so it's equivalent to doing this.

And for his purposes, that took less computation. And he already knew in 1965 that what you want to do is make sure that your ex and ey refer to the same point in pixel space, a lesson which has since been forgotten, except here. So that was the front end. And this has to be very efficient because it's run on every pixel. And it also lends itself to special purpose hardware, of course.

So the next step was our subpixel edge detection method. So we used what is called non-maximum suppression. So this is weird terminology. Why not just say finding the maximum? But there it is. So where did that terminology come from? Well, the idea is that we apply this edge operator everywhere, and in most places it has a feeble response, but on the edges it really kicks up.

And so one approach would be, let's just threshold. So if we get a strong response from one of these molecules here, then we're on the edge, and if not, then we're not. Well, that involves early decision making, because once you've made that decision, that's it. You'll throw away that edge point and never do any computation with it again because you've decided it's below threshold, or vise versa, you picked it as being a threshold.

So in the patent, Bill Silver makes a big fuss about avoiding thresholds if possible and not making decisions too early. That's his main motivation for not using thresholds. So some previous edge detection work did work that way. You apply some operator which has a strong response on the edge, and then you threshold.

And now you get responses. But they're not just right on the edge because we saw that the edge is a slow, smooth transition. So there'll be neigbouring points that also have a strong response. Plus, with noise, there will be points in the background where noise just happens to add up, and now there seems to be an edge there. So that's undesirable.

So the previous methods worked by thresholding and looking for things that had a strong edge response. And here, instead what we're doing is we're going to remove everything except the maximum. But maximum in what sense? Well, again, just into the gradient direction. And so here we deal with the unfortunate fact that we're going to quantize the gradient directions.

So we're only going to have compass directions-- east, northeast, north, northwest, west, et cetera, eight of them. And let's suppose that this is a quantized gradient direction. Then we step through the image and we consider those three values. And the non-maximum suppression says that we will accept this as a potential edge point only if this is true.

Because if g minus was bigger than g0, well then that's going to be the edge point. And remember that the edge is running at right angles to this lot, so the actual underlying edge is like this. And we're looking in the gradient direction, and the gradient direction is of course perpendicular to the edge.

And then we talked about how there's this asymmetry, because occasionally we're going to find that g0 is equal to g minus or g0 is equal to g plus, and we don't want to declare both of them to be edge points. We want to have a tie breaker so that only one of them gets elected. And which one? Well, it's arbitrary. You could easily well have done that, as long as there's a way of breaking that tie.

And then we said now we can plot the profile of this edge response along the gradient direction, and we get a picture like this. And then we can fit some curve to it. For example, we can fit the parabola to it, and we find the peak of the parabola. And that's our subpixel edge position. So several points there.

One of them is, why fit a parabola? Well, it's arbitrary. The shape of that curve depends on the optics, the image sensor, the thing you're looking at. But we only get three samples of it. So treating it as a smooth curve, the lowest polynomial that will work is second order, so one option we have is to use that. Not to say that that's the only option or the best one, but it's a pretty good guess.

Next thing is, what's s? Well, s is the displacement from g0. So in here, s0 would be that point. And then if s's, say-- well, if we get over here, then s is a 1/2. So if we get over here, then s is a 1/2. We're halfway to that point. And obviously, it doesn't make sense for s to be bigger than a 1/2 because then this would have been the maximum. And same in the other direction.

Notice that s is not in units of pixels because, in this diagonal case, s equals 1 is that distance, which is the square root of 2 times the pixel spacing. Whereas if we happen to get the quantized direction over here, then s would be the pixel spacing. So that's something to keep in mind, that it all depends on the actual gradient direction.

Avoid thresholds. Yeah. And so now we have a potential edge point. And notice we're not doing any thresholding. So we're going to get these points all over the show, not just on the edge. But we haven't done any thresholding. So here, we mark this place based on square route of 2 s delta, where delta is the pixel spacing, so just to be precise. So that's where we think the edge is.

So I've drawn the edge here, but actually, now with subpixel interpolation, we find it's there. So if we just were to go with a peak in that curve on the discrete grid, then we'd find that the edge runs through that point, but now we know it's over there. But in order to get there, we quantized the edge direction. And so we can improve things slightly.

So I suppose I should draw that again and make it less messy. So here is our quantized direction, and here is some point we found that's an edge point. And then suppose that the actual gradient direction is slightly different. It can't be hugely different because then we would have picked a different compass direction.

So now that's the gradient direction, so the edge has to be perpendicular to it. So I can draw a perpendicular to the green line that passes through this point. So the edge actually is like this. And when I report a potential edge point, I can report any point on that line. And which one do I pick? Well, the simplest one is just to pick the one I calculated. I can, however, slightly improve the result by instead picking this one.

Why? Well, it's closer to the origin. It's closer to the actual peak, and so it's less likely to be subject to noise. It's not a huge improvement, but they decided this was a worthwhile additional step. It also actually aids in a step that we will be getting to in a second where we chain together the edges.

So that's the idea. So we project from the quantized gradient direction down onto the actual gradient direction there. That's the plane position [INAUDIBLE]. What next? Oh. So then we get to the bias compensation. So we said that we somewhat arbitrarily picked the parabola. And then, as you know, in the patent there's a second method which uses a rooftop triangular shape.

It uses that. It fits that and finds the peak of that, and that gives you another possible position for where the edge really is. And we said that in certain circumstances one may be more accurate-- give you a more accurate answer than the other. But what you can imagine doing is actually experimentally moving the edge by very small increments, tiny subpixel increments, and seeing what your method gives you and then plotting that against what it should have been.

So this is an experiment where you have a camera looking at an edge, and then you move the edge or the camera by some tiny amount in increments and you measure. And now, ideally, your magic method for peak finding should always give you the correct peak value. But it may not. So ideally, you'll get a 45-degree line.

And again, we're only interested in the range from minus 1/2 to plus 1/2. And all of these methods should give you the correct answer in three cases, no matter what you do. If your peak is actually at g0, then it should return g0 that position-- s equals 0. And similarly, if you were halfway between pixels, it should give you the-- and both of these methods do, they satisfy that requirement.

But what happens in between? Well, as I mentioned, that depends on exactly what shape the edge has. And you might end up with something like this, or maybe something like that. So typically, the departures from the diagonal will be quite small, and typically they'll be quite smooth. And so you could keep a lookup table of this or something to compensate for it. But it's not really worthwhile. It's a relatively small correction again.

But in going down to-- the aim is 1/40th of a pixel accuracy. So if you think about it, that's quite amazing that you can do that. And one part of it is this plain position correction, and one part of it is this. And so what is done is to approximate whatever that shape is with this function.

And so, for example, for b equals 0, we get s prime equals this. So that's just the diagonal line. That's the ideal case. For b greater than 0, then that means that s prime is s raised to a power greater than 1. And so that means it's going to bow this way as the red curve does. And if b is less than 0, that means that s prime is s raised to a power that's slightly less than 1.

So that's approximating square root. So that's this bowed upper curve, which I'll show in green. So that's green. And the other one is red. So what's the point? The point is that this is a small correction, so it doesn't really pay to try and be too fussy. But you want to do it. And this is a one-parameter fit to the type of curve that-- B is the one parameter.

And so you could calibrate it based on this method, get one value of b. You could calibrate it based on this method, get a different value of b. Then you notice that if we happen to be working east-west, the spacing between pixels is much less than if we're working north-east. And so it turns out you want a different value of b for this case than you do for that case.

Fortunately, there are only two cases. They're the ones that are east-west, north-south, and then the ones in between. But you can use a different value of b for those two. And you again, you could do something more elaborate, but since it's a small correction, it's only going to affect a small fraction of a pixel.

Also, you don't want to be too clever here because this curve is going to change a little bit with circumstances. If your camera is slightly out of focus, you'll get a slightly different result. If the corners of the cardboard box you're looking at are somewhat damaged, then you'll get a different edge response, and so on. So you don't want to be too overly clever.

Now, a lot of this depends on the actual edge transition. And we drew one just by hand and then came up with these methods for finding out where the edge actually is. But realistically, what's causing these edges to be fuzzy? We already said that it's a good thing they are fuzzy. Otherwise, we wouldn't be able to do the subpixel recovery. We'd be suffering from huge aliasing problems.

So it's a good thing they're fuzzy, but why are they fuzzy? Well, one reason is the defocus. So let's just look at that as a special case that's of interest for other reasons. So here's our lens, and the object's up there. And suppose that it's a point light source. And this is in focus plane where that distant light source, star maybe, is imaged as a point.

But our camera has the image plane slightly off, and so the picture will be slightly out of focus. So what did I call this? So this is f, and I guess I call this delta. So when I look at the image of that star, it's no longer an impulse, a point. It's a circle. So uniform brightness.

And if I want to plot it as a function of x and y, it would look like this. And I don't know, I call this a pillbox. I guess people don't have pillboxes anymore, but it's a cylinder of constant height. And if I want to describe it mathematically-- so what's that? Big R is the radius of the little pillbox.

And I divide by 1 over pi R squared because the same energy is being deposited into that area no matter how out of focus I am. So if I'm in focus, it's all concentrated at one point. If I'm out of focus, that same amount of light is spread into a larger and larger circle. And so I compensate for that by dividing by the area of the circle.

And then what's this? So this is the unit step function. So for R equal to 0, this will be u of some minus quantity. So that'll be 0. So I get 1 minus 0. And so this will just be 1. And then when I get out to the radius, if I go past the radius, if little r is bigger than big R, the step function is 1 because this will be greater than 0. And so I get 1 minus 1 is 0. So it's just a fancy way of saying the same thing as that diagram.

The other thing I need is what is big R? How big is it? Well, it's obviously going to depend on how far out of focus I am. So it's going to depend on delta, and it's going to depend on the size of the lens. Something like that. It's just similar triangles again. Oh, f, which is this distance here from the lens to the in-focus image plane.

So obviously, as I go more out of focus, as I move the image plane further up, this radius gets bigger and the brightness gets less because I'm now putting the same energy into a larger area. And so this is called the Point Spread Function, PSF, Point Spread Function for this system when it's out of focus.

And this is used a lot in understanding the effect of being out of focus. We think of it as blurring. And of course, we can think of it in terms of the Fourier transform, in terms of removing some or suppressing higher frequency content. And it's the higher frequency content that makes things look sharp and, in our case, blurs the edge.

So let's see what the effect is on the edge. Well basically, we're going to have a response which we can calculate geometrically by superimposing the edge and the circle. So let's take a simplest case where there's black on one side of the edge and white on the other. And now we have a circle of a certain radius.

And what are we looking at? Well, we're looking at this overlap. That's going to be what controls how bright things appear. And then we're going to move this thing across the edge to see how the response varies. So we take this disk and we slide it across. And obviously, until it touches, nothing happens. There's no output.

And also, obviously, once we get over here, the output is constant. It's 1. Nothing more changes. So there's a transition between x being minus R and x being plus R where there's some change between 0 and 1. And that's what we're trying to calculate. Well unfortunately, it's not quite that simple. So we can just write the answer out by inspection, but we can get it this way.

There's probably a formula for a sector of a circle like that, but I don't know what it is, so let's do it using things we do know. So a couple of things we know. We know how to compute the area of a triangle. Well, probably know several ways of computing the area of a triangle. And we also know how to compute the area of a sector of a circle.

So let me draw that here. So that's a sector of the circle. That's the triangle I'm looking at. And it's obvious that the quantity I want is the difference. It's what's left over when I subtract that triangle from that area. So this is R, and I'm going to call this angle theta. And this thing here is obviously R squared minus x squared.

So x is the position. x0 means that you're right on top of the edge, that the circle is bisected by the edge. And then x can get as large as plus R before it saturates or minus R before it saturates. So the area of the sector is 2R squared theta, this sector here. And we can check. When theta is 2 pi-- I guess-- oh, this is-- theta can only get up to pi. So we would get-- oh, pi R squared? How about let's check that again.

So theta is the 1/2 angle of the sector, and so it can only get as large as pi. And if it's equal to pi, that means we've covered the whole circle and we should get the area of the circle, which is pi R squared. So we get R squared theta. And then we have to subtract from that the triangle area, which is-- it's 1/2 the base times the height.

So the base is 2 times this quantity because this quantity is just that section. So that's the base. And then the height is x. And so 1/2 base times height gives me that. And theta is given by this quantity. And so I end up with-- the details aren't too exciting here. I'm not going to expect you to remember this. But we can plot that and see what transition it gives us.

And what's more, we can then feed it into this algorithm of this pattern to see how accurately it will determine the edge position. And in particular, we can plot the diagram that I just erased here which had s versus s sidebar. In other words, if this is how the edge is formed, if this is why the edge has that smooth transition, then this will allow me to calculate the error.

And the error ought to be pretty small, but it's non-zero, and if you want high accuracy, you have to take it into account. So another way of looking at this is to plot this diagram. So where does that come from? That looks like a circle, except it's elliptical because it's been increased in height by 2.

Well, if you think about it, when I move this edge or the circle, then I am adding or subtracting an area-- infinitesimal area that has a height equal to this quantity. Or in other words, just twice the height of the circle there. So actually, for some purpose, I don't need to do all that hairy math. I can just look at that diagram and immediately write down that the brightness derivative is that.

And oh, it has a peak at 0. Well, we expect that. While I'm here, I can look at the second derivative. And it looks like that. And it's x over minus-- it's just the derivative of that, which is pretty easy. But what's E? Well, it's the integral of this thing. And we just did that integral in a somewhat painful way. But it's probably just as well because I don't remember what the integral of that is, and there it is, I think.

So what can we do with this? We can now feed this into the algorithm and say, if this is how-- why the edge is smoothly varying because it's out of focus, then this is the relationship between the true position of the edge and the one I compute by, say, using the parabola argument. And therefore, I can compensate for it. Now of course, in a real imaging situation, there'll be more than just the defocus of the lens, so it probably doesn't pay to be too careful about this.

Now suppose that you're a patent infringer or you're trying to infringe this pattern. Then there are a number of ways to go. Basically, you look at the components of each claim and you see if there's one of the components that maybe you can avoid. Maybe you can do it in a different way. Or maybe you can do it in a better way. But just arguing whether something's better or not doesn't help. It has to be different.

Now, there are some things here that aren't that pleasant. One of them is this quantization of gradient directions. And the reason it's not that great is because it introduces the awkwardness that the spacing comes in two sizes, pixel spacing and square root of 2 times pixel spacing. And these effects of defocus, et cetera, are they on pixel grid? They're in units of pixel spacing.

And so now we're sampling it in two different ways, so we expect to get slightly different error contributions. So how can we avoid that? So here are a couple of ways of-- so the idea is we don't want quantized gradient directions. And so suppose our gradient is-- now, if we follow the preferred method in the patent--

Again, notice that that method doesn't show up in any of the claims, which is good because that means that you could easily circumvent the patent. But it is the preferred method that's shown in the specification. Then we would quantize this, and we would work with the diagonal. But let's instead say, well, how about this? How about if I figure what-- if I knew what the value was there, I know what the value is here.

How about this arrangement? So this could be G0, and that's G plus, and that's G minus. Then I would avoid the quantization of gradient direction. And how do I do that? Well, I only know the values on the actual pixel grid, and so-- ta-da-- I interpolate. So I use interpolation to go from the values on the grid to the value over here.

And because I'm interpolating along this line, I can actually easily just use a 1D linear interpolation. So how does that work? Well, if I have a function, then I can say, well, I know the value here, I know the value there. And let's say it's a straight line in between. And then the formula for this is-- you can come up with that yourself. But you can easily check it.

First of all, it's linear in x. And then at x equals a, we get only this term. And the b minus a cancels the b minus a. And that x equals b, this term drops out, so we only have that. And again, the b minus a again. So it's easy to do that in 1D. And of course, you can extend that to so-called bilinear interpolation in 2D, but we don't actually even need that here.

So then we approximate the value here by interpolation. And we can use more sophisticated methods of interpolation. For example, we can use cubic spline interpolation, which we won't talk about here, but it involves more points. That gives you an interpolation that's a smooth cubic curve, which in some cases is more accurate.

And then we perform whatever we did, and we end up somewhere there. And we don't need the plane position step because we're actually on the gradient direction. We're exactly where you want to be. So you wonder, why didn't they do that? Well, they don't say, but two reasons that I can think of.

One is that, before we complain because sometimes we were using pixel spacing and sometimes square root of 2 times pixel spacing, well now we can have anything in between, not just those two values. And that correction graph, the bias graph, will be different for all of those values. So you've got to-- I don't know-- quantize it, build a lookup table. Not insurmountable problems, but an extra hassle. And you wonder, is it worth it?

That's one reason you might not choose this. The other reason is that you did an interpolation, and how accurate is that? You've gone away from the values that you actually know for sure to something that you interpolated using a method that you chose in some arbitrary fashion. How good is it? So that'll introduce some error as well.

Now, in many cases that error wouldn't matter. It'd be so small. But if you're going for a 40th of the pixel accuracy, even that small error can be significant. So that's one method, one alternative. And one of its problems is changing size of this step from a pixel to square root of 2. Well, what if we get rid of that and we just have a fixed size?

So here's again our pixel grid. We know the values at all of the intersections of these lines. And this is G0. And now we say, well, let's just draw a circle. And again, we'll pick some gradient direction. And now, we use that value and that value and that value. And those are equally spaced. They're always the same distance from each other, no matter what the gradient direction.

So I've now dealt with two issues. The one is there's no quantization of the gradient direction, and there's none of this change of the-- what does x equals 1 mean? Before, that could take on two values, and over here it could take on a range of values. Well, here it's always the same. It's the pixel spacing.

Of course, I don't know the value here, so I have to use interpolation, which is now a 2D interpolation. And so I can use bilinear, or I could, for example, use bi-cubic. And again, why didn't they do this? Well, it's extra work. Particularly with the bi-cubic, we need to take into account more pixels than we have here, go out to a 5 by 5 grid not just the 3 by 3 grid. Bilinear is not terribly accurate, so we may not want to use that.

Anyway. So you can see that, even though a lot of things are described in the pattern, there is also some hidden stuff going on. They probably experimented with all of this and picked something that was very accurate and yet simple, and that's how we got to that method.

Now, the last thing I want to talk about today is multiscale. They do point out in this patent that you might want to do this at multiple scales. They don't make a big fuss of it here. But we already mentioned that that's something you want to do. There are some edges which are very sharp, and they will be best found at the highest resolution.

And then there are some other edges that are kind of blurry, either because they are out of focus or because their object doesn't have a sharp transition. If the object has a smoothly turning curvature, then there will be a transition in brightness from whatever the brightness is for one planar surface to the brightness for a different planar surface. But there are all these in-between values. So the transition won't be on one pixel. It'll be spread out over many pixels.

Let's see. Do I want to do this now, or-- we haven't talked about Cordic, so-- I'm just looking at the time. We don't have much time, so why don't I do Cordic? So this was the method to go from ex, ey to e0. e theta, where e0-- so it's just a Cartesian to polar coordinate transformation.

And square roots [INAUDIBLE] take some computing. You'd have to probably use a lookup table to speed that up. So what is their preferred implementation of this? So the idea is that we rotate the coordinate system. So we have some gradient, ex, ey.

And if we knew that angle, then we could just rotate it down, and then the length of it, ex, would be e0 and ey would be 0. So we're aiming for where e0 is just ex prime and ey prime is 0. When you write it this way around, that makes no sense.

Now, we don't know theta, but we can rotate using some test angles and, by an iterative method, keep on improving, keep on coming closer to this situation where the y component of the gradient is reduced to 0. Oh. [INAUDIBLE] So the superscripts in parenthesis again refer to the iteration number rather than being powers. And of course, we know in 2D a rotation matrix is just very simple.

And so one thing we can do is have a sequence of thetas that we try. And when we're finished with this iteration, when we decide to stop, the answer for the angle is just the sum of all of those increments we make. So you can imagine various strategies. Like you can try an angle, and suppose that this goes the wrong way. Well, then maybe take 1/2 that angle and try that. Or you could try turning through that angle positive and through the angle negative and compare the two results, see which one works better.

And then keep on reducing the size of the angle until you converge, or at least until you're happy enough with the results. So each time, reduce-- each step, reduce the magnitude of ey, and it will increase the other one. Because as we rotate closer to the x-axis, the projection of this onto the x-axis will increase.

So that's the iteration. And how do we pick the thetas? Well, we could do this. Theta 0 is pi over 2. Theta 1 is pi over 4. Theta 2 is pi over 8. And so on. That would be one obvious approach. So we try and turn through these different angles and see if it-- and we accept the change if it satisfies this condition, that magnitude-- that ey is reduced. It doesn't have to be positive.

So it could overcompensate as long as the magnitude is reduced. And you can always flip it to make it positive again if you want to. So you can do that, but it's expensive for two reasons. One is you got the cosines and the sines, but you could build a table. Well obviously, you just store the cosines and sines of these angles.

The other one is each time you need four multiplications and two additions. And it doesn't sound like much, but remember, this happens several times because of the iterations, and it happens at every pixel. So it's expensive. So how do you avoid multiplication? Well, what we can do is pick the angles very carefully.

Suppose that we picked the angles so that they are inverse powers of 2. Then that matrix there, the rotation matrix, just becomes this. And the matrix part of this is very easy to compute, multiplication by 1. Doesn't cost us anything. And this is just a shift, and shifts are extremely cheap. And then we have an addition. So it reduces the whole thing down to two editions from four multiplications and two additions.

Well, then we have-- and we do this repeatedly. And you can see that the angle we're turning through gets smaller and smaller. What is that angle? Let's see. We can compute it from that formula. After a while, it basically gets halved. Initially, because of the non-linear nature of trigonometric functions, it's not halved, but eventually it becomes half.

And then when you're done, you end up over here. You end up with a product of all of these cosine theta i's. And so what do you do with that? Well, maybe you don't care because it's just a constant multiplier. But suppose you do care. Then you can pre-compute it. And actually, it's 1.16 very quickly. It converges very fast. So you can just use 1.16.

So that's the basic idea of cordic, that we rotate, but through special angles that have this property, where the tangent of theta i is 1 over 2 to the i. Well, it takes a bit longer to explain it in the paper, but that's the basic idea. And so the iterative process is you change through that angle, and if it improves the answer, you keep it. And then you keep track of whether it got negative or not.

And the first thing you have to do is get it to the first octant. The whole idea here is that you're working in this regime, but that's obviously trivial. You just look at the signs of x and y and whether y is greater than x, and you can reduce it to the first octant.

So next time, we'll talk about multiscale, and we'll talk a little bit about sampling and aliasing. Of course, that's part of the multiscale story. So again, please start early on the quiz. It's more work than the typical homework problem.