MITOCW | watch?v=bJvv5SckGeA

DENNIS FREEMAN: So last time, we started to think about sampling. And that's what I want to finish up today. I think sampling is a very important issue. It's one of the strengths of this course because we can think about on equal footing the way signals work in a CT system, or in a DT system, when the signals are CT, when the signals are DT. And specifically, when you convert between them.

Converting between them, like we saw last time, that's a very important process because many of the kinds of signals that we want to think about occur in physical-- have a physical origin where they are naturally continuous time or continuous space kinds of signals, but we would like to use inexpensive digital electronics in order to process them. So it's important to understand how we can take a CT signal and represent the information that's there in a DT manner.

And it's completely remarkable that you can even do that. CT signals are in some sense arbitrarily more complicated than DT signals. DT signals only exist at integer multiples of time, at integer values of time. CT signals, in principle, can do anything between two consecutive samples of a DT signal. So in some sense, they're arbitrarily more complicated. So it's kind of remarkable at all that we can talk meaningfully about how you can represent the information that's in a CT system with a DT equivalent system.

And the point is, and the reason we're doing it now in this part of the course, is that by thinking about Fourier transforms, everything's very simple. Something that could be conceptually quite complicated is in fact, extremely simple to think about.

So last time, we saw that the way to think about the signal, if you want to sample it, if you want to convert a CT signal to a DT signal, the way to think about it is to think about the Fourier transform. So then, the example that we talked about last time, you think about a CT signal, x of t. You think about its sample is taken uniformly in time. And then in order to think about the information and whether or not you've captured it all, the question is, can you reconstruct the original thing that you started with from the samples only? OK. Well, in general, no. So what we're really asking is, what are the rules, what are the conditions under which you can do that? And are they useful conditions or not?

So the first way you can think about taking the samples and turning them back into a continuous time signal is something that we called impulse reconstruction. In impulse reconstruction, we substitute for every sample an impulse appropriately located in time and appropriately scaled in amplitude. The appropriate scale and amplitude is that you take the samples and you weight the impulses. You weight the impulse at the n-th time step by the sample value for time n. And you put the n-th one at time nt, n cap t. So impulse reconstruction. It's really easy.

Take all the samples that you got by uniform sampling, substitute for every sample one impulse-- appropriately timed, appropriately weighted. OK, that's great. It's especially nice because there's a simple Fourier representation for that process.

That process, if we think about just taking x of t and turning it into this impulse reconstruction, that impulse reconstruction is precisely the same as if I had multiplied the original signal x of t by an impulse train. Impulse is separated by capital T unit height. So that means the transformation can be thought of in terms of Fourier transforms as the convolution of the original spectrum, the original Fourier transform, with the Fourier transform of the impulse train, which is just another impulse train.

So the rule is you can represent all the information in the signal if the signal started out being bandlimited. OK. If this signal had a region of frequency over which it is non-0 and for the rest of frequency the signal is 0, then when you do the aliasing, you can arrange the period so that the aliased copy-- so that the convolved copies don't overlap with each other. OK. So that was a simple way of thinking about, how much information was in the samples, by thinking about the impulse reconstruction.

Of course, the signal that we reconstruct by this convolution process has multiple copies of the same frequency content. So we don't like that. So you can throw away those extra copies by doing a low-pass filtering operation. And we call that reconstruction-- the xr, we call that the bandlimited reconstruction. It's like the impulse reconstruction, except that it's bandlimited. OK. So we think of two ways of doing the reconstruction from the samples-- the impulse reconstruction, the bandlimited reconstruction.

And the key is the sampling theorem. The sampling theorem says that if the original signal had non-zero frequency content over only some particular range of frequencies, you can sample fast enough so that you can represent all of the information that's in the continuous time signal with the samples. OK. Is that all clear?

The point is we're trying to represent the information in a CT signal using DT. And that the Fourier transform is a way to visualize when you can do that and when you cannot do that. You still end up in a physical system, perhaps generating signals whose frequency content falls out of that range. We saw an illustration of that last time.

So for example, if you were to try to represent a signal with this transform using a sampling period t, so that the impulses in frequency were separated by 2 pi over t, which happened to be less than twice this distance, then it would alias. That's bad. So we would typically also include an anti-aliasing filter, pre-filter the signal from physics, get rid of the parts that you know are going to be a problem when you try to sample. Then, go ahead and do the regular sampling, the regular uniform sampling, the regular bandlimited reconstruction. And the signal that you reconstruct won't be an identical copy, but it will be as close as you can given the sampling theorem. OK. So that's what we did last time.

What I want to do today is think about some other issues that come up when you try to represent a continuous signal in a discrete domain. So in addition to thinking about discretizing time, we also have to think about discretizing amplitude. Because if we want to represent a signal by bits-- so we have to represent not only the time, but also the amplitude in bits.

I'll talk about several different kinds of schemes for that. In the simplest kinds of schemes, the code for the representation in amplitude is separately derived from the code for the representation in time. So we can think of it as two boxes, a sampling box followed by a quantization box. The first box, the sampling box, takes the CT signal of time and turns it into a DT signal. The second box takes the samples, which have a continuous domain, and turn them into samples from a finite domain-- from a discrete domain. OK.

So if you're doing that kind of a quantization scheme, then the thing you have to think about is how many bits you're willing to use to represent each sample. I mean, this is the simplest kind of a scheme that you could use. There's much more complicated schemes by the end of the hour. I'll tell you about a scheme that is much more efficient than this. But this is kind of the base level. This is where you would start.

So if you wanted to represent an amplitude in a discrete representation, one way you could do about it-- one way you could think about it is to think about the map between the continuous values that the sample could acquire and map it to a discrete output set.

So for example, if you were using 2 bits per sample, then you might represent any voltage between minus 1/2 and 1/2 by some code 0, 1. Any voltage that's in the range 1/2 to 1 as the code 1, 0. And any voltage in the range minus 1 to minus 1/2 as 0, 0. That would be a way of taking a continuous range of possible amplitudes and turning it into a discrete number using just 2 bits.

Obviously if you use more bits, you can get greater precision. What's showed below here is, what if my signal was a function of time-- looked like the red waveform. My discrete representation might look like the blue waveform, right? If I'm imagining that I only have 2 bits, then I only have 3 possible symmetric outputs. So that might be represented by the blue.

And the difference between the red and the blue is showed in the green. And as you can see as you go to more bits, you obviously get errors-- the green signal as it's getting smaller, right? So the key thing then is, how many bits do you need for the thing that you're trying to represent?

So I like hearing. So I'll illustrate the number of bits by thinking about sound. You can hear sounds that range in amplitude over a range of about a million to 1. So if you were to put a person with good ears-- not me, one of you. If you were to put one of you into a quiet room and let you sit there until you adapted, and then played the faintest sound that you could possibly hear, then multiplied by 10, multiplied by 10, multiplied by 10, you could make it a million times more intense in pressure. You could amplify the pressure by a million before it'd start to hurt. It wouldn't damage yet.

You'd have to go to about 8 million, and then it would start to damage. But you could do about a million to 1 over the range from just barely audible to starts to hurt. So how many bits would it take to do that range?

So how many bits would it take? Raise your hands. Show me a number of fingers. How many bits would it take to represent a million to 1? OK. 100%. I think it's 100%. So easy question.

So if you use 1 bit, you can represent 2 levels. If you use 2 bits, you can do 4. 8, 16, 32. By the time you get to 10 bits, you're up to 1,024. By the time you're up to 20 bits, you're up to 1,024 squared.

OK, 20 bits ought to do it. And in fact, 20 bits-- if you were to buy a high-end audio system, it would be 24-bits. There are people who claim you need 32. I think they're kind of crazy. But a high-end audio system would be a 24-bit system.

Now, if you were to listen to sort of CD quality, CDs are 16 bits. So there are people, even me, who claim that they can tell the difference between a concert and a CD representation of a concert. OK. So there might be some limitations of representing audio with 16 bits. But what I'll show you is a demo where I've showed the same piece of music at 16 bits, 8 bits, 6 bits, 4 bits, 2 bits, and 1 bit per sample, so that you get the idea of what a quantization error sounds like. Yes.

AUDIENCE: So I think the difference between a concert and a CD, it's mainly because [INAUDIBLE].

DENNIS FREEMAN: There's lots of things that are different. And you're raising a very good point. You certainly don't get the spatial aspects of a concert. We try to fake you out. We put false cues in, so the violin sounds like it's on the right side. But those are all fake, usually. Well, they're not completely fake. And we have stereo. And we have 5 plus 1. So we have lots of different representations.

But if you were to imagine listening in a concert monaurally. So plug your ear, clamp your head so you can't turn, and compare that to listening with a mono headphone, that's what I'm talking about. So if you didn't get spatial cues and things like that. OK.

So the issue then is to listen to different levels of quantization.

[MUSIC PLAYING]

DENNIS FREEMAN: So it's actually kind of amazing, right? You can sort of tell what the piece is the whole way down to-- how many of you could tell the difference between 16 and 8?

AUDIENCE: [INAUDIBLE]

DENNIS FREEMAN: How many of you could tell the difference between 8 and 6? How many of you could tell any difference whatever? Just joking.

What's the difference in the sound quality? What's the effect of quantizing?

AUDIENCE: Fuzziness in the background.

DENNIS Kind of fuzzy. So could you simulate the fuzzy sound? What would you do if you wanted to sort

FREEMAN: of simulate the fuzzy sound? Besides, of course, quantizing, which would be a perfect simulation.

AUDIENCE: [INAUDIBLE]

DENNIS FREEMAN: Noise. It kind of sounds hissy. [HISSING] It sounds kind of noisy and that's kind of the point.

And that's an important issue because it affects how much music you can put on any given medium. So for example, in a CD, CDs are 16 bits per sample, 2 channels, 44.1 kilosamples per second, 60 seconds per minute. 74 minutes is a typical recording time for a CD. So you end up with about a gigabyte. And that's what you can put on one of those little plastic things.

If you were willing to live with 8-bit instead of 16-bit, you could obviously put on 148 minutes. So people don't make these decisions lightly.

It's how many people do you make angry for one reason or the other, right? You can make them angry because they don't get much music or you can make them angry because they don't get high quality, right? So you get to sort of trade-off the kind of people who hate you. But that's the kind of idea.

So if you have a piece of plastic on which you can put 1 gigabyte, you have to think about how you're going to represent it. And it matters how frequently you sample. And also, with what quantization you represent each sample.

Same sort of thing happens for pictures. Here's a relatively high-quality picture, where it's 280 by 280 pixels. And it's an 8-bit representation in amplitude. The point's just that the kinds of things that happen when you quantize a picture are very similar to the same sorts of things that happen when you quantized audio.

So if we take this picture and compare it to-- substitute for each pixel a quantized version of the amplitude. Quantized here to 8 bits and here to 7 bits. You might be able to see the difference.

If I come up really close, I can certainly see quantization effects. If I drop the right one to 6, 5, 4, 3, 2, 1. OK. So here is 8 bits and 4 bits.

Remember that when we thought about the audio example, it sounded fuzzy. It sounded hissy.

[HISSING] What's the effect of quantizing here? Yeah.

AUDIENCE: [INAUDIBLE]

DENNIS FREEMAN: Sharp and-- say again?

AUDIENCE: The contrast.

DENNIS FREEMAN: Well, there's certainly a problem. So both of these pictures have high contrast, right? How would I see contrast in the pictures? Contrast refers to having big steps, step changes in brightness. So like, I might see a high contrast between this petal and that leaf. And I still have a high contrast at the analogous place over here. So there is some contrast effects.

A little more subtly, the contrast affects how well you see the quantization. So if I changed the picture to have different amounts of contrast, I could effect whether you could see the quantization well or poorly. So in audio, the effect of quantizing-- as I quantized more and more and more, I caused more and more hiss [HISSING] in the background.

What's the effect here? What's the effect of quantizing? Yeah.

AUDIENCE: You have less grays to work with.

DENNIS FREEMAN: I have fewer grays.

AUDIENCE: So 1-bit was just black and white. So as you increase bits, you get more grays--

DENNIS FREEMAN: Absolutely. Could you give me sort of a qualitative assessment of the kinds of errors that you see here compared to the kinds of errors that you don't see there? Yeah.

AUDIENCE: [INAUDIBLE]

DENNIS FREEMAN: There's banding. Why would there be banding? Nobody said the audio sounded like it was banded. We just don't hear that way, right? Even though we're doing a similar process, why do we see banding in pictures? What's causing the banding? Yeah.

AUDIENCE: [INAUDIBLE]

DENNIS Yeah, exactly. So the pixels that are nearby-- so take the pixels here, which came from pixels

FREEMAN: over here. They have nearly the same gray value, but the quantizer is making up its mind at a very precise level. It's deciding, oh, you're between these two levels. Turn into this number. If you're between these two levels, turn into this other number. So you get the bands because there's correlations in the brightnesses of pixels that are nearby. So you get this banding thing that can be objectionable whenever the quantization is not sufficient. OK.

So one way you can reduce that is called dithering. Dithering means add noise. So that's kind of weird. So I want to get rid of the bands. So what do I do?

I take every pixel. And before I quantize it, I add noise to it. Then even if the pixels came from a region that were nearly the same amplitude to start with, each individual pixel gets a different amount of noise so they quantize differently.

And if I choose my noise in a clever way, I could use my noise to be plus or minus 1 quantum. So I could choose a random number generator that gave me numbers that were evenly distributed over the range minus 1/2 quantum to plus 1/2 quantum. And if I do that, then I can generate a picture that is quantized but was dithered before it was quantized. So the two pictures are both quantized at the level of 7 bits, but the one on the right had dither added to it first. So I'm adding noise before I do the quantization.

And you can't see too much at 7. 6, 5, 4, 3. OK. So what's the difference between the two? Well, over here I had these bands because the amplitudes were such that they all got converted into the same output. The bands have disappeared over there.

2. Even 1 the bands have disappeared, right? But that's obviously not a good solution. So what's wrong with dither?

AUDIENCE: Noisy.

DENNIS FREEMAN: Noisy, yeah. I'm kind of going back to the hiss thing, right? Now, I've taken a picture that had had bands and I've turned it into a picture that looks noisy.

There's a way to think about how the noise works. Imagine that I had a smoothly-varying signal showed in blue that was being turned from a continuous range of amplitudes into a discrete range of amplitudes. So let's represent the discrete amplitudes by the dashed red lines. Then, the signal that I might quantize could look like the red signal. And that's a very graphic representation of where the bands come from. So the bands come from the fact that the original signal sliced through a small number of quantized outputs.

Everybody see where the bands are? Then, if I add dither, I can think about-- so this transformation from blue to red, I can think about that as being y equals Q of x. So x is the blue line, Q of x is the red line.

Down here, what I've done is I've taken x and added noise to it. Then, I ran it through the same quantizer. And you can see that I've broken up the bands, but you can see that I've added a bunch of noise.

So there's a slightly more clever thing that we can do that's called Robert's technique. Larry Roberts was a masters student here. He was here before I was here, which is kind of a remarkable thing. But they actually wrote thesis back then and they used paper. And you can go to the library and it's still there. So Larry thought of a method for dealing with this where what you do is you take the original signal x, you add n to it and quantize it, but then you subtract n back off. And that's called Robert's technique. And that's illustrated by this transformation.

The good thing about this transformation is that this-- so here, the quantization error was clearly correlated with the signal. That's what banding is, right? Something about the signal turned into something about the error. Here, the error is still correlated with the signal. The correlation is less obvious, right? But here is a range of errors that are all positive. And here is a range of errors that are all negative. So the errors are still correlated with the original signal. So the result-- and when you do Robert's technique, you destroy the correlation.

So with Robert's technique, you end up with-- it's still noisy. Because after all, I added noise to it. But I've added it in a very clever way that removes the correlation between the error and the signal. And the result is that the noise seems less.

So if you compare 6 bits with dither to 6 bits with Robert's method, both pictures are represented by 6 bits. 5 bits, 5 bits. 4, 3. So the interesting thing is that the Robert's method looks like less noise. It's mathematically not.

Mathematically, you can show that Robert's technique has the same energy in the noise as was in the ditherer technique. If you just calculate the energy in the error, they're identical. But in Robert's technique, he destroys the correlation and that makes the noise seem smaller. It's like physically less objectionable.

What's the problem with Robert's technique? If I told you to implement a scheme that quantized according to Robert's technique. And say you're here and you're supposed to quantize a message, send it over the ethernet, and receive it in California. And you're only supposed to be sending, say, a 6-bit representation instead of a 16-bit representation. What's hard about Robert's technique compared to dither?

Quantizing is easy, right? I take my 16-bit CD. I take off the first sample. I quantize it. I send it across the internet.

I take off my second sample. I quantize it. I send those 6 bits over the internet, et cetera. Dither is sort of the same thing. I pick up the first sample. I add noise to it. I quantize it. I send those 6 bits over the internet.

What's the hard part of Robert's? Yeah.

AUDIENCE: Do you send the noise too?

DENNIS FREEMAN: I have to send the noise, too. I have to know the precise value of the noise that I added to sample n, so I can subtract it back out. So Robert's technique says, I take the value x and I add some amount of noise n. End was a random number. I chose it by throwing a die or something. I quantize that, and then I subtract that same number back out.

Well, that number has to be precise compared to the quantization levels. So for example, people would normally use-- if I'm doing 16-bit audio, people would normally use a 16-bit representation for n, which means that I take a 16-bit number off the CD. I take a random number. I add it, quantize it.

And now, I can send the 6-bit number. But in order for that guy to reproduce the answer, he has to know n too. Everybody see that? So the problem is, how do you send the noise?

And the trick is that we use something called pseudo random noise. Pseudo random noise is an algorithm that generates a sequence of numbers that looks random, but they were made algorithmically. So you can independently manufacture the same sequence here and there. That way, if you're using the same-- if you pre-agree that you're going to use the same algorithm, you can independently generate the same sequence of n's. OK.

Yeah, so I jumped back to explain-- OK. So the point is that just like in audio, in pictures it's important how many bits you quantize to. That affects drastically the performance of communications or storage devices.

How many pictures can you store someplace? How many pictures can you put on your iPhone? So all of that matters quite a bit. And the code that you use is very important.

And you're not limited to just-- I have two more examples. So the simplest possible schemes are the ones that I've showed so far where you think about the sampling in time and the quantization in amplitude as separate processes. You don't have to do that. In fact, you can get much higher performance if you combine the two.

So the first combination I want to think about is trading off precision for speed. And that's something that we call progressive refinement. The idea is, imagine that I want to make a digital representation of all the paintings in the Louvre. OK. It doesn't make sense to do 200 by 200 at 6-bit resolution if you were looking at pictures in the Louvre. That doesn't make any sense, right? You would like to see a high-resolution version. OK.

And now you're a user, and what you'd like to do is leaf through them and find photos of something or other. Scenes of some type. OK.

Well if you've got a high-resolution representation and you're trying to thumb through a lot of images. The problem is, if each one is represented with high resolution, that can take a long time. So if you didn't do something clever, basically you would have to download the Louvre before you could do your search.

So the idea in progressive refinement is first send me a crude representation. And if I haven't changed in my browser, if I'm still looking at the same picture three seconds later, continue to load the information that makes the picture increasingly precise. Give me a crude representation as soon as you can.

And then if I sit there, give me a more refined representation. But if I lead to someplace else, stop downloading that one and give me a crude representation of the new place. That's the idea.

So the way you can do that is with discrete sampling. I started with a digital representation of a painting in the Louvre. Maybe it was 20,000 by 20,000 with 24 levels of color-- some huge picture.

So what I'll do is I'll sample it. But this time, it's DT sampling. DT sampling-- you'll be completely shocked to hear this-- is completely analogous to CT sampling. It's almost the same thing. That shouldn't be too big of a surprise, all of the different transforms, all the different Fourier representations that we looked at, are almost the same thing.

So DT sampling turns out to work almost exactly like CT sampling. So think about what you would do if you wanted to take a picture and represent it with a factor of 3 fewer pixels in the horizontal and a factor of 3 fewer pixels in the vertical. Well, you would sample it.

In CT, we would think about multiplying the CT signal x of t by an impulse train. Here, we use a unit sample train. So we think about an original signal x of n. And we think about a sampling waveform that's now at an infinite unit-sampled training. We used to use an infinite impulse train, now we're using an infinite unit-sampled train. So we preserve every third sample and throw away the ones between.

So that's a way of generating a new picture that only has one third of the information that was in the original picture. And as I said before, it should come as no surprise that the math for thinking about this sampling process is virtually identical to the math that you need to think about the CT sampling problem. In particular, the key is to think about the Fourier representation.

If this were the original Fourier signal, if this were the Fourier representation of this signal, we have to think about the Fourier representation for the sampling signal, the infinite unit-sampled train. An infinite unit-sampled train, not surprisingly, the transform of that's going to be an infinite impulse train.

All DT signals are periodic in 2 pi. That's a property of DT signals. That's a property of the unit circle. So we're not surprised to see that this signal was periodic in 2 pi. This signal is also periodic in 2 pi. That's because it's DT. But it's also periodic in one third of that. That's because of the periodicity here. OK.

So if we had had a sample at each one of these, then the base periodicity would have been 2 pi. But here, because of the periodicity being 1 every third sample, we get 3 times that many impulses. So just like in CT sampling, we think about multiplying the original waveform by a sampling waveform that preserves only the information at the samples.

We do the same thing here. Multiplication in time is convolution in frequency. So we take the original signal, we convolve it, and this is what comes out of that sampling process.

We get the same rule for the sampling theorem that we got for CT. This process has to be such that when you do the convolution, the resulting nearest neighbors shouldn't overlap. So there is a maximum frequency for the discrete system, just like there was a maximum frequency for the CT system.

There's one more step. Obviously, if I sample the picture at the Louvre, I don't want to send the 0's. That doesn't make any sense. So in order to not send the 0's, I smash together the non-0 samples. That's illustrated here.

Smashing in time does what in frequency?

AUDIENCE: [INAUDIBLE]

DENNIS FREEMAN: Squish in time, stretch in frequency. They're reciprocal spaces, right? Frequency and time are reciprocal spaces. Smash in time, stretch in frequency. So the result is that when you smash the 0 entries out of the signal, you stretch the frequency representation by a factor of 3. And when you stretch by a factor of 3, this peak, which was at 1/3 of 2 pi, moves the whole way out to 2 pi. OK.

So the idea then is that I've got this beautiful picture in the Louvre. Maybe. In order to send a lower resolution version of that, what I do is I low-pass filter it because I don't want the frequencies to alias. So I low-pass filter it. That gives me a representation that I can then downsample. OK. So this had the same size, but this one has fewer high-frequency components. So I can downsample, which gives me something that can be represented in the squeezed version with fewer pixels.

I did a downsample by a factor of 2 in both, so that picture has 1/4 the number of pixels in it. Then, I can low-pass filter that one and downsample. And low-pass filter that one and downsample. And I end up with a very low-resolution image of this beautiful scene that I started with. OK.

So that means that I start with some number of pixels. Here I have 1/4 as many. Here I have 1/4 of that. And here I have 1/4 of that. So I have a fourth cubed the original number of pictures. So it will go 4 cubed faster. So it'll take me a lot less time to get the low-res picture. So the result then-- skip this for the moment.

So here's my low-res picture. With a lot of imagination, you can clearly see what that is. At the next level of refinement, you get this. At the next level of refinement, you get this. At the next level of refinement, you get this. By now, you're tired so you flick on something more interesting. No. You would continue to look at this, right?

And finally, you get the original picture. So the idea then is that I want to not only transmit. But then the question is, how many bits do I need to do this? And the answer is that having transmitted this, I can use that information to help me generate this.

OK. So what I do, I run the process backwards. Let me back up. So in order to go forwards, I thought about squishing this into a smaller representation. Well, I can go backwards. I can up-sample.

When I up-sample, all I do is I take all the pictures in the shrunken version, I stretch them, and I put 0's between them. That gets me here. But that's not where I want to be. I want to be up here. So how do I go from here to here?

So when I put the 0's in it. So I started with this, I put the 0's in it. That stretched it in time. That compressed it in frequency.

When I compress this waveform into frequency, this 2 pi peak ended up at 2 pi over 3. So now if I want to get back to the original contribution, I have to low-pass filter. OK. Everybody see what I'm doing? So the final scheme then is that-- whoops.

The final scheme is that I low-pass filter, downsample, low-pass, downsample, low-pass, downsample. Downsample, I can up-sample by putting 0's between all the rows and columns. Then, low-pass filter and that gives me this picture. So what I need to do is also transmit the high-pass information that I threw away.

So if I separately transmit this picture in the high-pass part of this picture, then I can combine them to get that picture. And I don't actually need to transmit this one. So I don't need to transmit this one either because I can generate. So I only need to send this and this. Then, I do the same thing here.

If I take this, I put 0's between it, low-pass filter. I can generate this picture, so I don't need to send it. But I do send this. Then, I combine these to get that recurse. OK.

So the result is that I send-- so I don't send this, but I do send this. I don't send that because I'm going to regenerate it. I don't send that. I do send this. I only send this, this, this, and that.

And that's enough information to reconstruct the picture. Right

And notice it has the hierarchy that you would expect. You start with a low-res. It takes more bits to make this one. It takes more bits to make that one. And it takes more bits to make that one. You're worse off if you didn't do something clever by-- so I'm sending the full number of bits here. Then, I'm sending another 1/4. And then, another 1/16. Then, another 1/64. So I'm sending about 33% more bits total. But there's tricks.

The trick is that the eye is less sensitive to these high frequencies than it is to these. So I really don't need to send the same resolution for this. So people use this all the time.

If you go to a slow website, you may notice that you get that kind of low-res morphing into a higher-res. And that's exactly this kind of a scheme. But there are cleverer things you can do. So that's already pretty clever. And that's already something you see in today's technology, but there are even cleverer things that you can do. And so the last thing I want to talk about is JPEG.

99% of the images that you download on the web are JPEG. JPEG is a clever technique that does quantization in the Fourier domain. And that's similar to what you would want to do in that progressive refinement because you would like to separate the frequency components and use less resolution for the higher frequency components because you can't see them as well. JPEG is a formalization of that idea. So this was made by a joint photography group that was very successful. It has four layers of coding.

First thing you worry about is color. OK. We think we see a broad range of colors. Wrong. We only see three. So you can throw away the ones that we can't see. So that's the first step is taking advantage of the fact that we really can't see all the different colors. We can really only see three colors. So there are tricks that you can do to make the person think he's seeing the exact shade of yellow, which we don't see very well, by mixing together a different combination of red, green, and blue. So you get to move the colors around. And you can make it perceptually indistinguishable, but easier to code. We won't talk about how you do that, but it's a very straightforward process by which you start with one picture and you change all the colors to make them easier to send. OK. So that's the color coding.

Then, they do a discrete cosine transform, which is really a kind of Fourier series. Then, they quantize the Fourier series, the DCT. And then, they code the resulting sequence using a lossless Huffman code. So we'll talk about the middle two steps because that's the fun stuff.

That's the Fourier stuff.

So the way DCT works is you take the image and you break it into 8 by 8 pixel squares. And then you do the same processing on each 8 by 8.

So here is an example of an 8 by 8 image. This is a completely trivial one where I have linear taper from black to white, linear taper from black to white, the product. And all I want to think about is, what's the DCT? And why do they use a DCT instead of a Fourier transform?

So just like you would expect from the other two-dimensional image processing, the examples that we've talked about, the way you do this is you do the DCT on all the rows. Then, you do the DCT on all the columns. And then you're done. That's a two-dimensional DCT. So here's an example.

What if I took my sample image, which had this linear taper. So if I think about just one row and I plot brightness on the vertical, then this might be my image right here. And what I do is think about periodically repeating it.

The original signal only had 8 numbers in it. I'm going to periodically repeat it because then I can take a Fourier series. It's a periodic signal, and it's a Fourier series. The reason I do that is that the Fourier series only has 8 coefficients. The Fourier series of an eight-long sequence has eight Fourier coefficients.

So the idea is that by taking a signal that's only 8 samples long-- I mean, the obvious thing you could do is take the eight-long signal and take a discrete time Fourier transform. Problem with that is that that's a continuous function of omega over 2 pi, over the entire unit circle. So you take 8 samples and turn it into a function of omega which has lots of samples.

By thinking about the 8 samples as having come from a periodic extension, then I don't get a continuous range of frequencies between minus pi to pi. I get exactly 8 of them, a0 through a7. OK.

So the first step is to do periodic extension on the 8 samples. Then, I can represent it by 8 Fourier coefficients. In the DCT, they almost do that. But instead of writing down the numbers 1, 2, 3, 4, 5, 6, 7, 8. 1, 2, 3, 4, 5, 6, 7, 8, 1, 2, 3, 4, 5, 6, 7, 8. Instead, they write 1, 2, 3, 4, 5, 6, 7, 8. 8, 7, 6, 5, 4, 3, 2, 1. 1, 2, 3, 4, 5, 6, 7, 8. 8, 7, 6, 5, 4, 3, 2, 1. That seems like a dumb thing to do.

I took an eight-long sequence, which could be represented with 8 coefficients, and I turned it into a 16-long sequence, which now takes 16 coefficients. Wow, that's brain dead. Except that it's actually very clever.

Of these two signals, which has the higher high-frequency content? [INAUDIBLE].

AUDIENCE: [INAUDIBLE].

DENNIS FREEMAN: Sharp drop, large amount of high frequencies. That's the trick. So because there's a large amount of high frequencies, this signal is hard to represent with Fourier series. This signal is easier because there's fewer high frequencies. You need fewer of those high frequencies to do a good job of representing the signal.

You can throw away the high-frequency stuff and nobody will notice. OK. So the idea then is that you use this 16-long sequence, but then you know that whatever x of 8 was, it's the same as x of 9 because you always repeat it. And x of 7, that's the same as x of 10. So if you take advantage of knowing that there's a symmetry. And if you notice, they made it symmetric. So there's an even-odd kind of symmetry about a weird point. It's off by 1/2, but there's a symmetry this way, too. If you take those two things into account, you can actually represent the 16-length sequence with 8 numbers. That's the DCT.

It's exactly the same as a Fourier, except that we're taking the 8 non-trivial numbers and putting them together in a funny periodic fashion. That's what a DCT does.

And the point is the DCT maps 8 real numbers, which are these yn values. It maps 8 real numbers into 8 DCT coefficients.

And the DCT coefficients, unlike the Fourier coefficients, have real values. So because of the trick with all the symmetries and all that sort of stuff, they arrange to make a transform whose imaginary part is guaranteed to be 0. So there's no information explosion in going from the 8 to 16.

Here's the Fourier representation for a 2D picture. The Fourier coefficients are falling off like k. Here's the DCT where they're falling off like k squared. And the point is you can throw those away in the picture and barely tell that they're even there. That they're even gone.

So what they do then is they quantize the Fourier coefficients at different levels. So you divide the 0, 0 coefficient by 16 and send the whole part. You divide the 1, 0 by 11. You divide this guy by 61, so you use much less resolution by a factor of 4. Because then those numbers were chosen so that they give rise to coefficients that are equally visually distinct. The result is that you get very high resolution with a very small number of bits.

So here's an original. This picture has 47 kilobytes of data in it. And when you change Q, the quality of JPEG, what you're really doing is choosing those tables. So when you use a high Q, you get a good representation. When you use a low Q, you're throwing away more data.

And you can see that you can throw away-- so 47k down to 2k. You can throw away 19 pieces of data out of 20 and you still get a very good resolution picture. And that's because the quantization is happening in the Fourier domain. And you can match the Fourier resolution better to the psychophysical properties of the eye.

So the point is to tell you how to represent signals in discrete time in a way that the errors are as imperceptible as possible. And to demonstrate how the Fourier transform lets you do that. OK, thanks. See you later.