Real-Time Digital Signal Processing
Prof. Rathna G N.
Department of Electrical Engineering
Indian Institute of Science – Bengaluru

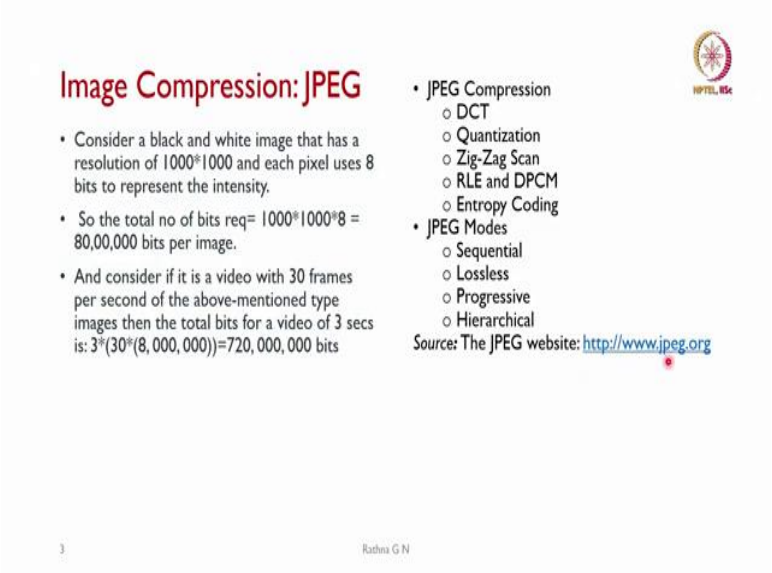
Lecture – 47
M3U29 - Discrete Cosine Transform - I

Welcome back to Real-Time Digital Signal Processing Course. So, today we will discuss about Discrete Cosine Transform. So, how we can implement using both Matlab and then the DSP processor? What we will be looking at it? So, before giving the demo so, we will have little on the theory part of it.

(Refer Slide Time: 00:44)

So, as a recap in the last class we did little bit on speech coding and then its applications. One of the application was LPC coding, what we did both in Matlab and then DSP processor. So, the other applications you can look in and then see that how you can use the speech coding in different applications.

(Refer Slide Time: 01:14)



So, today we will see little on image compression basically. So, mostly we will be concentrating on the jpeg part of it. So, why do we need compression? Most of you will be taking lot of images using your mobile phones, cameras and so many other devices basically. So then how you are going to represent it? Most of the time we will be telling that our memory is full.

So, we have to move our all images to some storage place, what we have to take it? It can be a cloud or somewhere hard disk. And then how you are going to retrieve and other problems challenges what you will be facing it? So, to say that how the number of bits is going to increase, we will see for a black and white image. So, we say that it has a resolution of or the size of it is 1000 by 1000 is the image what you have taken a thing.

And each the pixel we represent it with 8 bits, we know that black and white is 0 to 255 is enough for us to represent it. So, we say that 8 bits are sufficient to represent the intensity of this pixels. So now, we will see that total number of bits what is required is, We know that if we multiply 1000 into 1000 into 8 bits so, it comes to about as you can see that 80 lakhs bits per image what we needed.

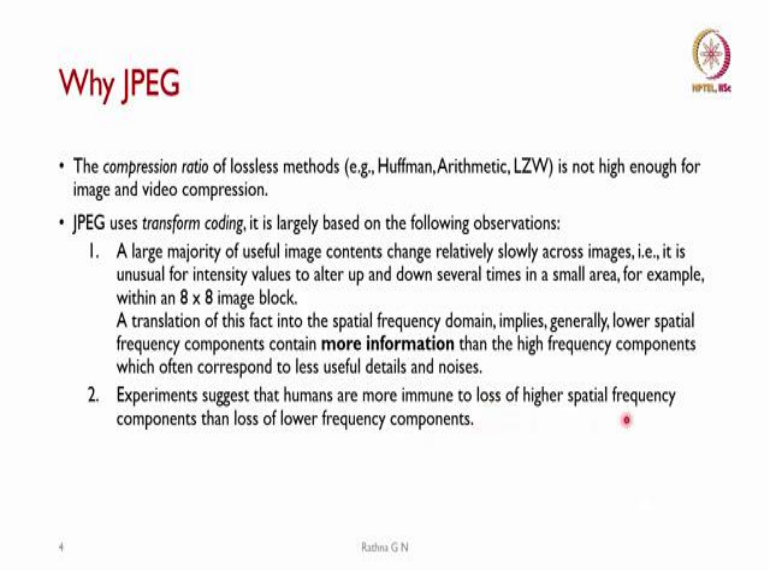
And we say when we are talking about the video this is only the still image what you are talking about. And then if we are considering a video for a black and white you know that it is 30 frames per second what we assume the video rate which is at which it is coming. So, this type what happens? We will be seeing that if it is taken for just 3 seconds. So, what will be the value of it?

So, it is 3 into 30 frames per second 30 frames what I have to store it, into whatever we have this value what we will be taking it 80,000. So, you will be seeing that how many bits is required to store our image in any of the storage units? So, the for the still images jpeg compression is one of the popular one, so that is join program extend group actually what it is. So, here what are the steps involved in it?

So, we know that we need discrete cosine transform, usually it is being used for compression. Then the bits whatever received is going to be quantized and then it is going to be a zigzag scanned. And then whatever you use the run link the encoding and then DPCM whatever differential pulse code modulation what you can use it. Then later on use the entropy coding these are the coding techniques what you can use it and then you will be transmitting it.

So, what are the modes available for the jpeg? It can be sequential what you can have mode or you want to have a lossless compression you can incorporate it. Or it can be a progressive mode what you can select it or it can be even the hierarchical mode what you can select, these are the options in the modes what you have it for the jpeg. So, the website what you can visit for more information is given here for you.

(Refer Slide Time: 05:18)



So, now why we have to select jpeg? So, the compression ratio of lossless method that is example is, we can use the Huffman coding or arithmetic coding or LZW coding. So, these are the ones what it is used for coding so, it is not high enough for image and video compression.

So, although we use these things still the size of the image what we have to store is very high. So, what happens in the jpeg? It uses this transform coding basically.

So, largely based on the following observations. So, you will be seeing that why do we need the transform? So, we have looked at Fourier transform in our course already, so, how fast you can make discrete fourier transform to run using fast fourier transform? So, why do we have to represent a signal in the transform domain also what we have looked in. Now, we will see why we need the transform for our images? What it says is?

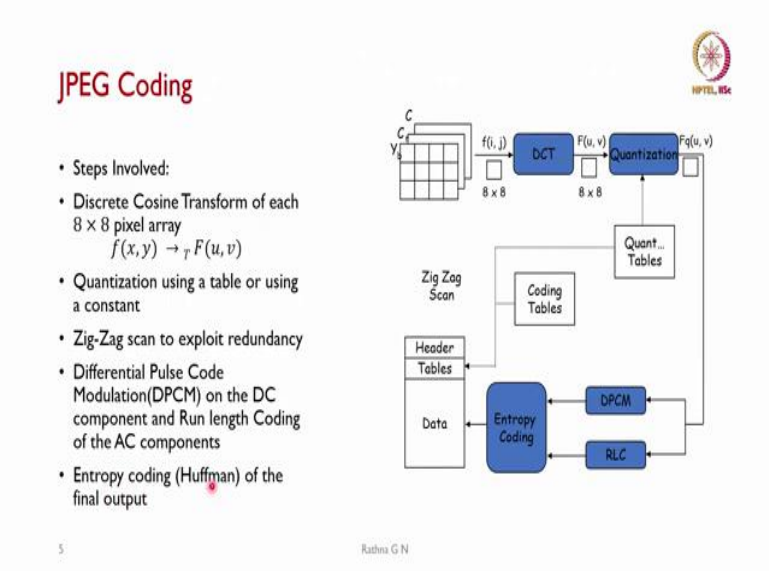
A large majority of useful image contents, changed relatively slowly across images. So, it is unusual for intensity values to alter up and down several times in a small area. So that is usually we assume the small area is 8 by 8 image block. So, even now you would be seeing when you want to create your video from a small image, what you do is, I think the famous one, you will be seeing that in a bird in a cage.

So, there are two different and if you rotate them very high rotation if you are giving the thing or high speed you rotate it, you will be observing the bird in the cage basically that is what the illusion what you will be getting it. So, it is enough for us to look in the small image block and see the intensity values and whether we can remove some of the things. So that we need not have to keep those values which are below some threshold what we will be putting it

So, the translation of this fact into your spatial frequency domain implies generally. That is lower spatial frequency components contain more information than the high frequency components. So, we say that which often correspond to less useful details and then we call them as noises. So, just like our speech, the contents of it is in the lower frequency part of it. Here also in the images, we say at the low frequency we have more drop coverage of the thing

And then at a high frequencies the not much information and then it may be noises. So, that is what it says that experiment suggests that humans are more immune to loss of higher spatial frequency components, than loss of lower frequency components. So, if we lose the higher frequency components because of our visual what we call it as seeing capability. So, it gets the low frequency components much registered than the one in the higher frequency region.

(Refer Slide Time: 09:07)



So, how we are going to incorporate this jpeg coding? So, these are the steps involved, first we do the discrete cosine transform of each 8 by 8 pixel array. So, if you know f of x, y is an image so, in the well this thing DCT what we will take it so, the image what you will be getting it is F of u, v. So, you will be seeing that here it is represented in YCbCr basically. So, this is the luminance and these are the chromium's what you will have it in the image data.

So, you will be passed, if you are considering the colour image so, instead of RGB so, we can represent it in this YCbCr format. So, you will be seeing even in storage, you have a jpeg image, bmp image, tiff image so, you can compare which one consumes less storage for your storages. So, we will consider 8 by 8 of the thing as you can see each one is divided into blocks of 8 by 8. So, we will be using that and then we will do the DCT of it.

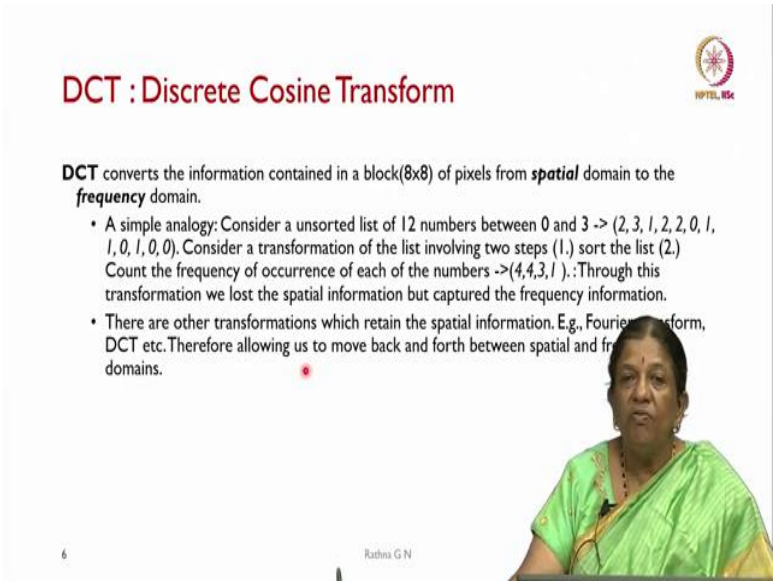
Then what we have is F, u, y is the output what we have it and do the quantization. So, we will call that quantized as Fq of u, v and that output what you will be feeding it for the sending through the channel what you can send it. And then this is the receiving side of it what you will be looking at it. So, you will be doing the you can have the either the run length coding you can do.

And then you will be using the differential PCM coding to generate our entropy coding part of it. And then it can be stored as data if you want to store it with header and then tables what you will be getting it from your quantization tables. What type of quantization if you have used it for this image? So, this is how the data is going to be stored. And then you will be having the coding tables also.

Whether you have used the zigzag scan and then what type of coding tables you will be putting it. And then you will be using in storing the images and which you can retrieve it back. So, what it says is? Quantization using a table or using a constant what you can do it. And then the scanning is going to be zigzag scan to exploit redundancy, we will see in a while with an example.

Then you will be using the differential pulse code modulation DPCM on the DC component and run length coding what you can incorporate on the AC components. So then you will be doing the entropy coding that is usually Huffman code what popularly in the jpeg what you can have it of the final output.

(Refer Slide Time: 12:32)



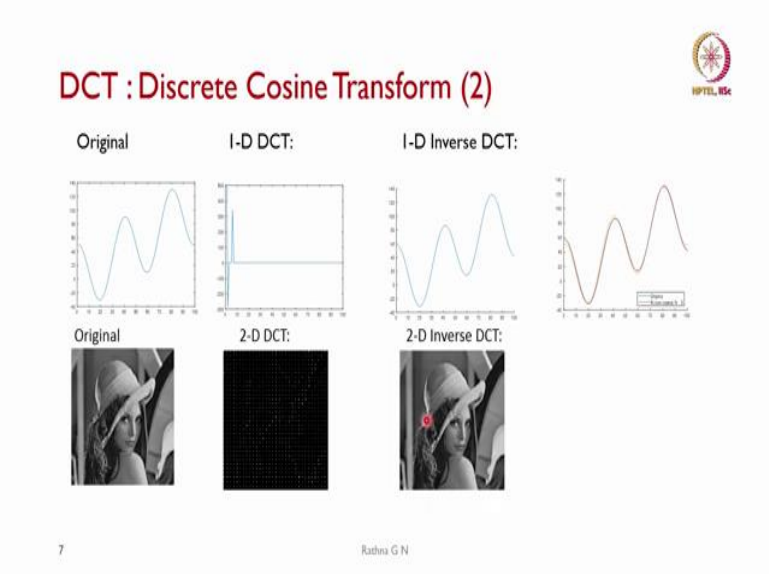
So, what happens? The first step what we have is the discrete cosine transform. So, this converts the information contained in our block of 8 by 8 of pixels from spatial domain to the frequency domain. So, in the case of FFT from time domain to a frequency domain we transform, here it is from the spatial domain, we will be going into the frequency domain. So, a simple analogy what it uses is, that is unsorted list of 12 numbers, between only 0 and 3 value what you are considering. So, this is the thing what you have it 2, 3, 1, 2, 2, 0, 1, 1, 0, 1, 0 and 0. So, how we are going to do the transformation of this list involving two steps. First is sort the list that is we know that what is this thing. Either it is upwards sorting you can do or downwards sorting what you can do the thing then sorting.

The second step is what you are going to use is 1, 0, 1, 0, 0 that is considering transformation of the list involving this thing two steps and then count the frequency of occurrence of each of the numbers. So, we because we have only 0 to 3 means it is only four numbers what we can represent here. So, how many times these got repeated? So, you will be seeing that 2 has got repeated four times.

And then you will be seeing that 3 is repeated four times, this is the repetition rate what you will be putting it. And through this transformation so, we lost the what we say is spatial information but captured the frequency information part of it. So, what is the frequency of numbers? Occurrence what we have collected. So, there are other transformations which retain the spatial information like example Fourier transform, DCT etcetera.

So, therefore allowing us to move back and forth between spatial and then frequency domains.

(Refer Slide Time: 15:01)



So, as an example so, running the Matlab that is we consider the discrete cosine transform. So, this is the original signal what it has been considered and when you take the DCT, this is the DCT values what you will be getting it. So, as you can see only in the small frequency that is low frequency, you have some values both positive and negative. And after that you will be seeing that it is almost 0.

Then what I can do is? This can be represented as 0 and then if I reconstruct that is I take the 1-D inverse transform then I will be getting back my signal. To show that how it is going to be the difference between the original and then this thing reconstructed is shown here. So that is

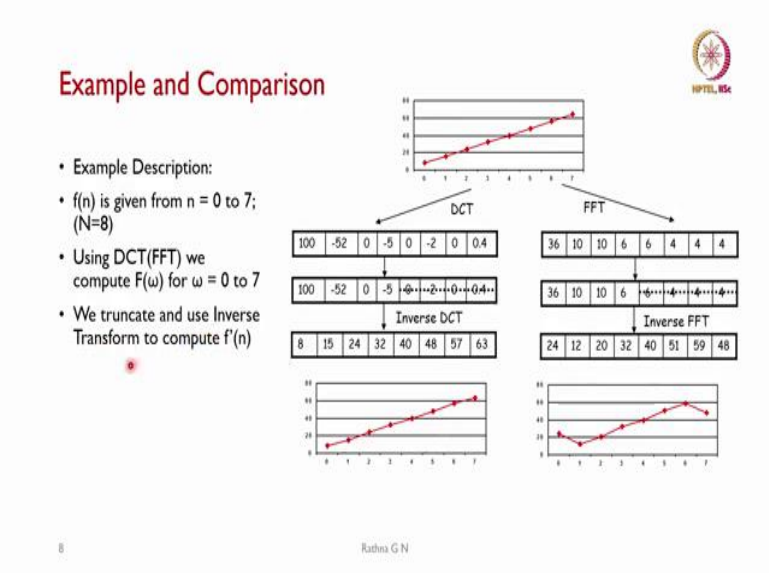
blue shows the original one and then the orange one is showing the reconstructed one. So, you will be seeing that with little difference almost it is following the original one.

So, you will be seeing as in this thing, in the case of image how it is going to look like? This is original Lena image, when I take a DCT of this that is 2-D DCT basically, this is how the coefficients what will look like. And then what we do is? We put a threshold and then we eliminate the coefficients and make them zeros. Then we reconstruct the this lossy information then after reconstruction we take a inverse DCT.

So, this is the reconstruction part of it. So, you will be seeing that our human eye perception what you can see is visually what you see is not much difference between the two. Sorry, I did not take the difference between the original image and then the reconstructed image and then we can plot that also. So then we will get the complete information, how much information is lost but still our perception of I has unable to make out.

So, only if you closely observe what you will be looking with the difference? So that is it had a 64,000 pixels. So, from 3,200 to 64,000 it was made zeros. And then only use only 0 to 3,200 images to reconstruct by using I-DCT. So, we will see in the lab these examples how we will be doing it?

(Refer Slide Time: 17:48)



So now, one more comparison, we said that even the discrete Fourier transform is going to work in the frequency domain for our images and discrete cosine transform. So, we will see that with an example you have eight this thing n is 1-D comparison what we are doing it. So,

you have 8 samples with different values what you will be seeing it. So, it will be in terms of as you can see that 8, 16, 24 that is what the values what you have at different this thing samples.

That is what we say n is 0 to 7 what it is been chosen and these values are passed through FFT and then DCT. So, this is what the table shows eight values so, you will be getting after doing FFT 36, 10, 10, 6, 6 and then all these are 4's. When you pass it through the DCT basically so, you will be seeing that this one, the first value what we call it as DC coefficient and rest of them we call it as AC coefficients.

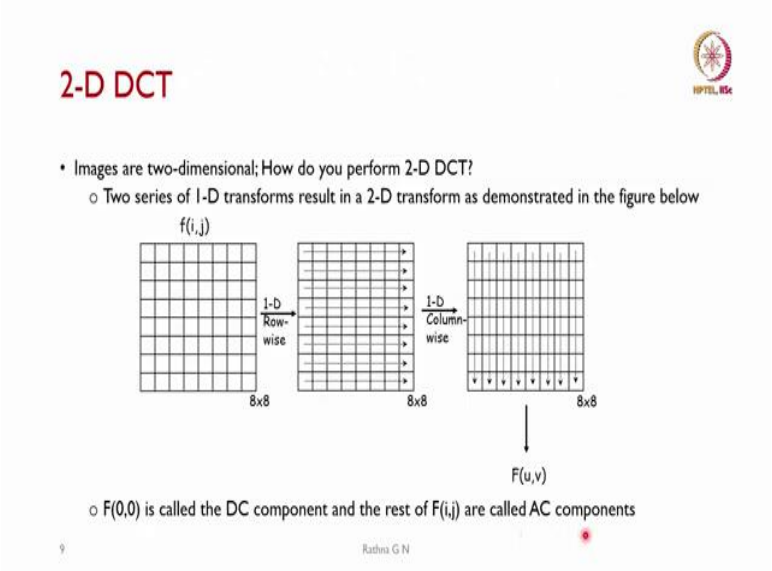
So, you will be seeing that maximum value is present in our DC coefficient and then you will be seeing somehow almost it is we say after the first coefficient, rest of them are almost negligible. So, for reconstruction we will do that 50% of the values we strike it off and make them zeros. So, same thing what we will do with respect to our FFT also so, we are striking of these four values.

And then we will take the inverse DCT here and then inverse FFT here. So that is I-DCT, I-FFT what we will be doing it and these are the values what we have got it. Whereas in the case of DCT, you will be seeing that few of them are exactly represented 8 24 and then 32, 40, 48 or almost same values. And then the other one instead of 16 what we have is 15 and then instead of 56 it is 57 instead of 64 it is 63.

But you can see the graph when we plot it, almost it resembles the original plot with little loss of what we call it as loss of information. When we reconstruct our this thing from FFT that is I-FFT we do the thing. So, you will be seeing lot of difference with some places values of it. So, you will be seeing that here it is gone bad and then here it is little smooth, again at the higher end it has gone.

Instead of 64 it is 48 this is somewhat nearby in these values and then after that it goes down. So, you will be seeing the advantage of using DCT for compression because we can eliminate these zeros. That is what it says is we truncate and use inverse transformed to compute f dash n in both the cases.

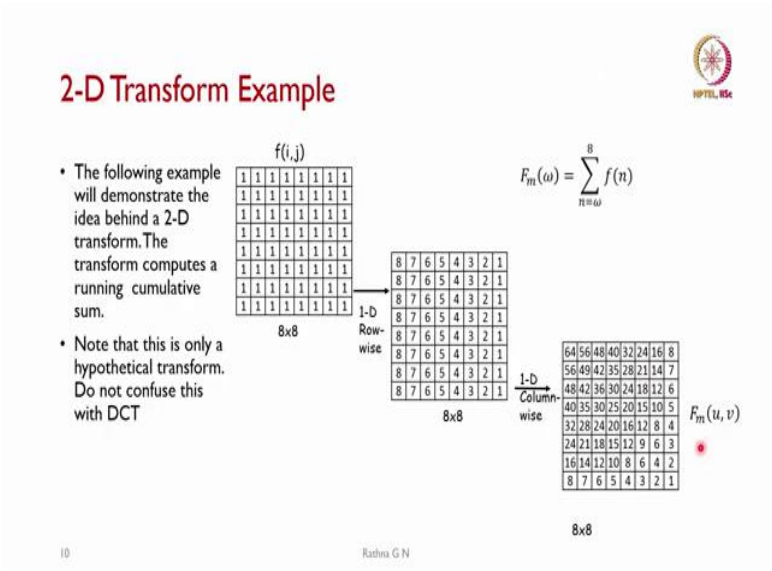
(Refer Slide Time: 21:13)



So, coming to 2-D DCT so, how we are going to incorporate this? That is 2 dimensional, how do you perform 2-D DCT? Usually what we prefer is 1-D transform. So, how it is going to result in? 2-D is demonstrated here, f of i, j is your 2-D image with pixel values, what it has been represented. Here it has been taken as 8 by 8 in this case so, for first we will do 1-D DCT that is row wise what you will be doing it.

So, it is shown as row wise DCT what we will do and then in the next stage, for this input we can do 1-D column wise, DCT what we will be taking it. Since we call it as it this is a separable transform DCT. So, we can use a row-D composition and then column-D decomposition together, to get our final output F of u, v. So, we say that F of 0, 0 is called DC component and rest of F of i, j are going to be called as AC components.

(Refer Slide Time: 22:30)



So, just it is an intuitive example in this picture what you will be seeing it so, to show the 2-D transform. So, the first one you will be representing all of them are once in this f of i, j and then this has a 8 by 8 matrix what you will be seeing it. So, first we will do the 1-D row wise. So, you will be seeing that you will be what the function what we are doing is that is, F m of w that is mimicking our DCT.

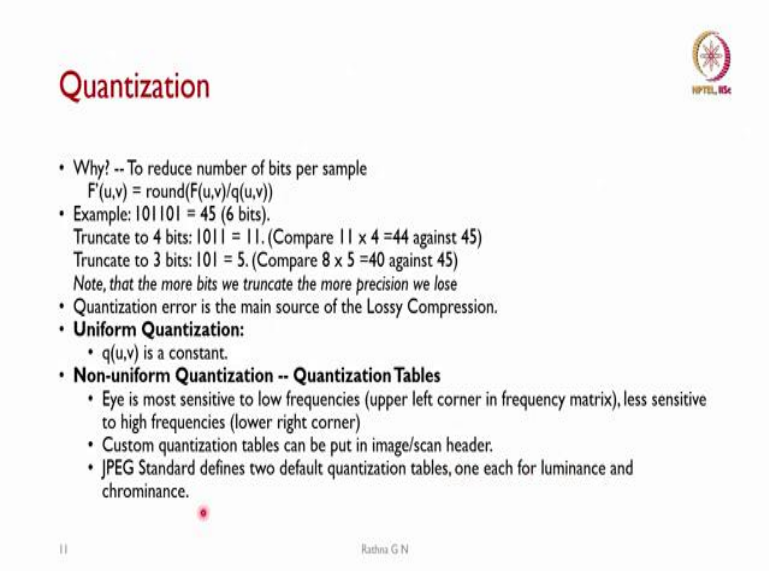
So, which will be omega will this is going to vary n will be varying from omega to 8, f of n that is summation what we are doing it. As you can see first, I will be doing it row wise when we add it up this is 8 then omega becomes 0 to either it is 1 then 1 to 8 or it has to be 0 to 7 as you know 8 values what we have to take it. So, omega becomes 2 then will be doing this summation so which gives you 7.

So on, what you will be doing it? And this is how our row wise transformation what it is shown. Just it is the addition what we have done this is not the one what will be using it for DCT as it is mentioned here. It is a hypothetical transform and then we will see what is our DCT equation later. So now, after doing this then we will do column wise. So, how we are going to do column wise?

So, you will be seeing that 8 into 8, 64 and 8 into 7 becomes 56 and then you can go on are doing that way. Last 1 is 8 into 1 so which is going to be 1 into 8 so, you will be getting it 8 here. The same be the next one so, it will be starting if I look from right to left so, it will be going from 7 to 56 same what you have to apply this equation. So, when you add it up column wise you will be doing it so, you know that 8 into 8 and here 7 into 8 and 6 into 8.

So that is how these values have been filled and later on also what you will be doing from here to here what you will be doing column wise. So, this is the intuitive for a thing to show that how the DCT is going to work? In the regular sense, the equation is going to be different. So, this is how you will be getting F m u, v.

(Refer Slide Time: 25:21)



Coming with a quantization, the next step is in our jpeg compression basically it is quantization, why do we need the quantization? That is to reduce the number of bits per sample. So, I need not have to as we know that if 7 has to be represented we need 3 bits 1 1 1. If I had to represent 0 it is enough to show that 1 bit is sufficient for me to represent a zeroth bit.

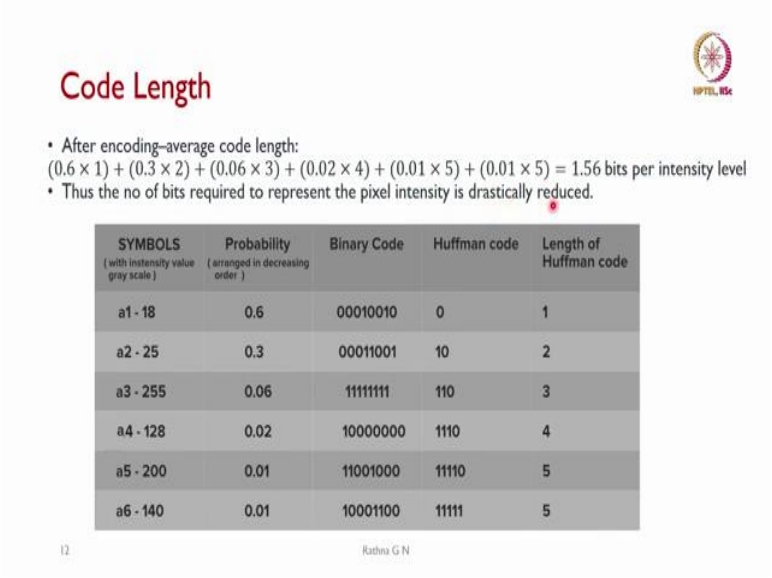
So that is how we can do the reduction that is what it is shown here, as an example 1 0 1 1 0 1 which is the value is 45, we need 6 bits to represent. And we will truncate it to 4 bits then it becomes 1 0 1 1 that is what we will be selecting it. So, I need this also further we can represent it as 1 1 so, compare, that is 11 into 4 is 44 against 45. So, instead of 45 we will represent it as 11 and then because we have represented it 4 bits.

So, I can multiply and then I will be almost nearer 45. Same way truncate to 3 bits that is 1 0 1 what you will be doing it. So then what happens? The result value what I will get is 5. So then what happens? It is 8 into 5 what I have to multiply which is 40 against 45. Whether this loss I can take into account that is what will be looking at it. So, how much quantization it can tolerate? So that the more bits we truncate, the more precision we lose basically.

So, it depends on how much precision you want to have it. So, the quantization it is the error is the main source of our lossy compression. So, I can do uniform quantization that is q of u, v is a constant value, I can take it and then do the quantization or I can do a different ways of quantizations. So that is the other one is non-uniform quantization so, you will be using the quantization tables that is the reason why you have a tables in jpeg compression.

So that is what we say, eye is most sensitive to low frequencies, upper left corner in frequency matrix what it says and then less sensitive to high frequency that is lower right corner. So, the custom quantization tables can be put in in an image or scan header and say what kind of quantization was incorporated? So, the jpeg standard defines two default quantization tables, one each for luminance and chrominance.

(Refer Slide Time: 28:24)

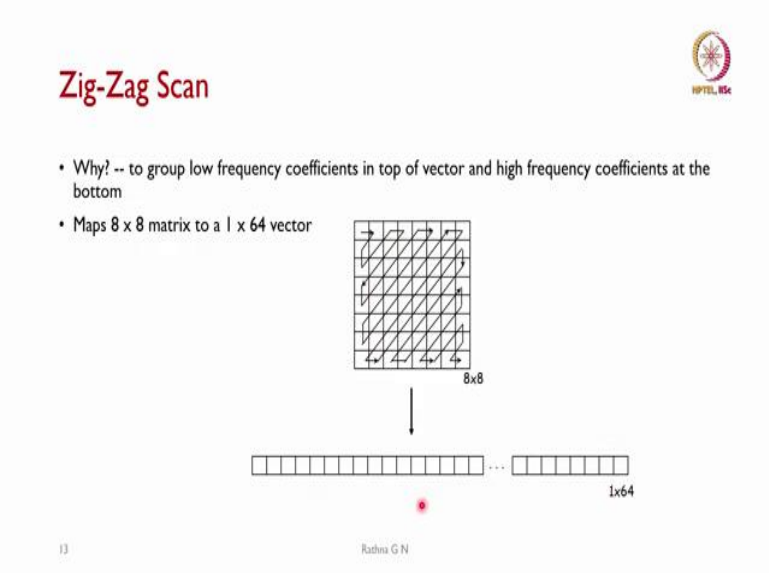


So, as an example, how the code length we can decrease that is encoding average code length, is shown with this way so, we say that a 1 has this thing 18 symbols what will be representing it 18. And then the probability of it is occurrence is 0.6 times and binary code to represent 18 is given here. And then if we represent with the Huffman coding we represent it as 0 then the length of Huffman code is 1 bit in this case.

So, the same way for a 2 = 25 what it is shown here which is occurrence is point 3 so, these are the Huffman code length what it is represent 1, 2, 3, 4 and then 5. Now, if we see how many bits per bits are required to represent our intensity level is shown here. So because occurrences is 0.6 and then I need 1 bit here. And then the other ones you multiply this is 0.3 the probability into number of bits.

So, you will be seeing that approximately we need 1.56 bits per intensity level. So, you will be seeing that, that is the number of bits required to represent our pixel intensities, drastically is going to be reduced from 8, earlier we had 8 bits which has come down to 1.56 bits per intensity level with our coding.

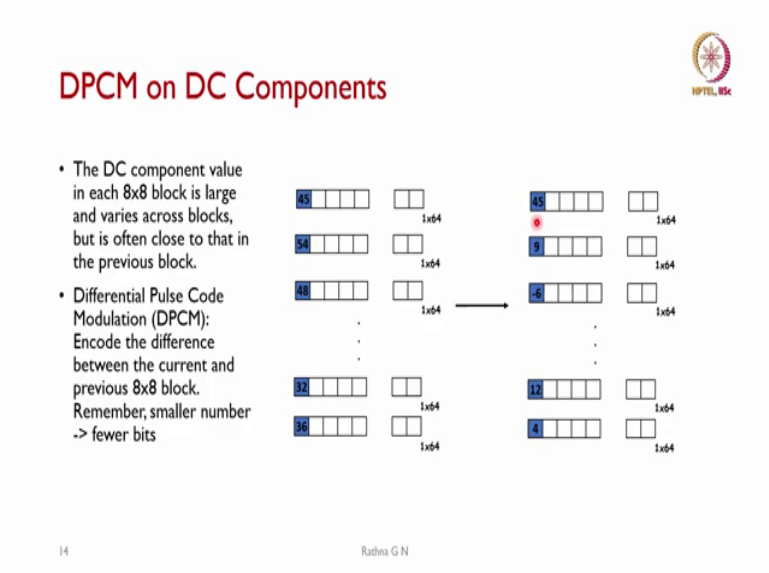
(Refer Slide Time: 29:58)



So, after coding we have to do the scanning. So, why to do the scanning? Because in this 8 by 8 different values what we have represented and then we have done the quantization. So, low frequency coefficients in top of vector basically, what we will call it and high frequency coefficients at the bottom, what will look at it. So, this maps 8 by 8 matrix into 1 into 64 vector. So, you will be seeing that this is how we will be representing it as 1 into 64?

As you have seen earlier also, higher values were for the DC coefficient that is on top and then after that it decreases. So, this this is how will be putting it in our zig-zag scan, what will do it and then put the values of the bits here.

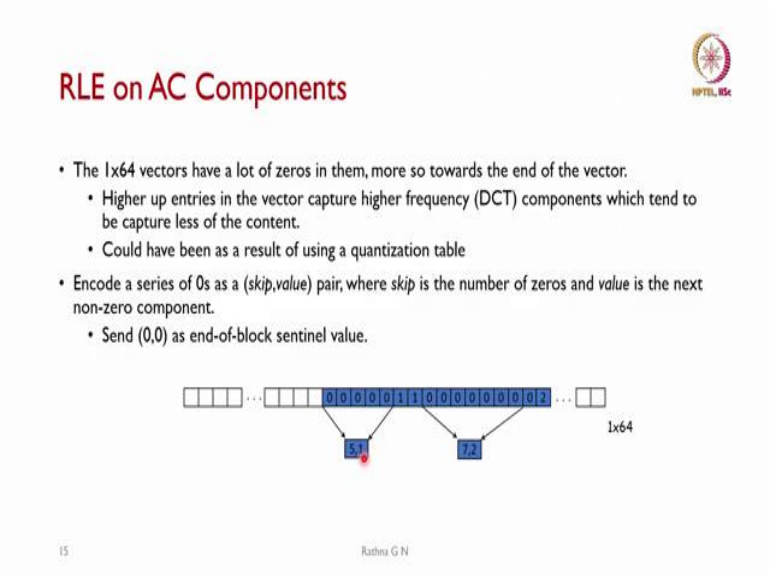
(Refer Slide Time: 30:55)



So, the next one what we say that? We will do differential pulse code modulation on the DC components. So, the DC component value of in each 8 by 8 block is large as we know in the

thing. So, continuously what we can represent it so, the first one will be the maximum value what will be having it. And then use this thing a DPC to represent this that is 1 into 64. So, this is how you will be converting it into a lesser this thing value in the initial. Because the otherwise we need more number of bits to represent this value.

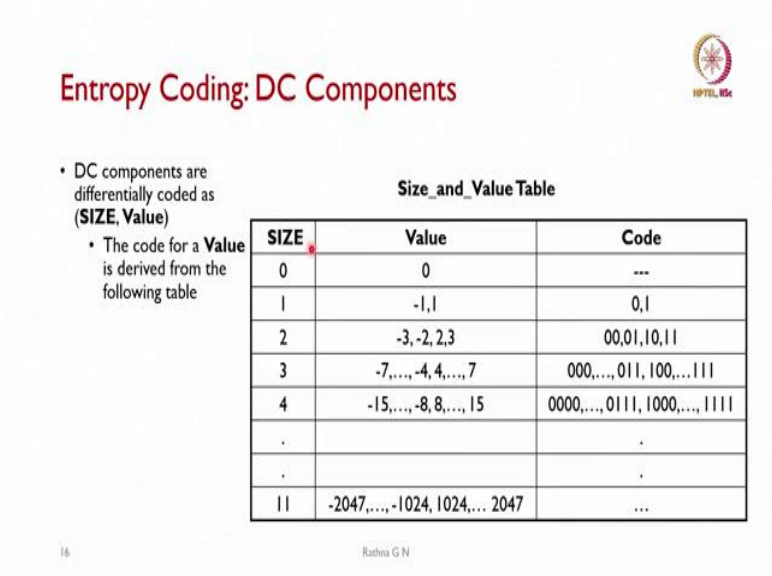
(Refer Slide Time: 31:39)



So, the next is we can have a run length encoder on AC components. That is the 1 into 64 vectors have a lot of zeros in them so, more so towards the end of the vector. So, higher up entries in the vector capture higher frequency that is DCT components which tend to be capture less of the content. So, could have been as result of using a quantization table. Encode a series of 0s as skip, value so, this is the pair what you will be sending it.

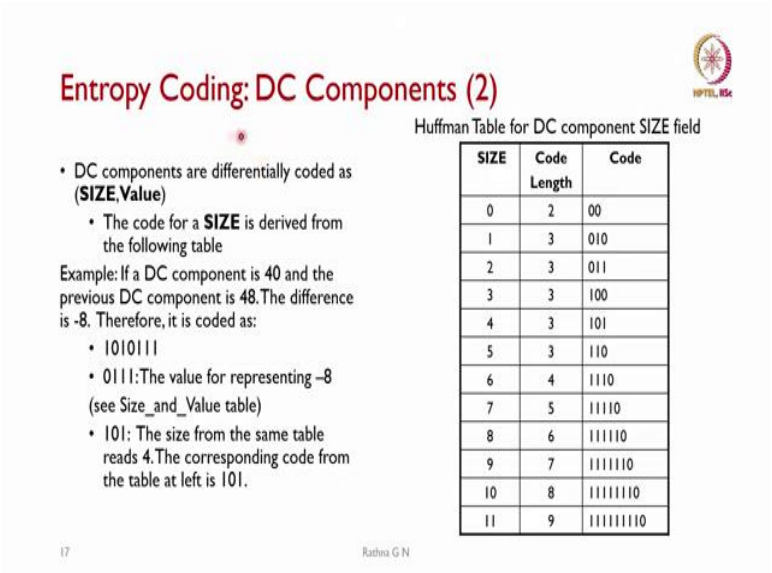
Where skip is the number of zeros and value is the next non-zero component what you will send. So that is send 0, 0 as end-of-block that is sentinel value basically what you will be sending it. So, you will be seeing that how this is represented? And then the run length coding if they are repeating it so, many zeros so, you will be putting the value skip and then value in this way.

(Refer Slide Time: 32:49)



So, the next one is what we have to do is, Entropy coding that is basically for the DC components if you use the thing, you will be having size and value what you will be providing it. So, the code for a value is derived from the following table. This is the size what we have it, size and value table these are the values and then how you will be generating the code is shown in this column.

(Refer Slide Time: 33:17)

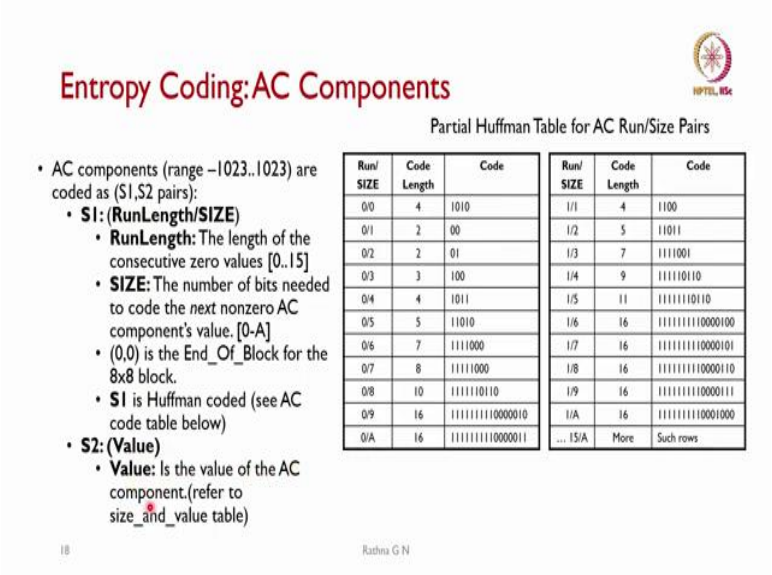


So, this is how the entropy coding for DC components happens. So, you will be seeing that as an example if a DC component is 40 and the previous DC component is 48 then what we will do? Because we are doing the differential pulse code modulation, the difference is –8. So, therefore it is coded as this value so that is 0 1 1 the value for representing –8. So, that is these are the values what will be representing it, this is the code length.

So, 1 0 1 is the size from the same table what you will be taking it and then it reads as 4. So, you will be seeing that 1 0 1 is 5 it reads it as 4, what you will be representing it. The corresponding code from the table at left is 1 0 1, here what you are showing it. So, the code length is 3 bits for 1 0 1 this thing size is 4, what you have taken the thing that is a length of it is you have chosen as for 8, 4 bits are required.

And this is how you will be a Huffman table for DC component size field what it is shown, fine.

(Refer Slide Time: 34:41)



Next is what you can have is Entropy coding for your AC components. So, they range between –1023 and etcetera to negative to positive value 1023. So, you will be coding it as S 1, S 2 pairs basically and then S 1 is run length slash size you will have it. So, how you are going to take the run length? The length of the consecutive 0 values that is 0 to 15 what you will take it.

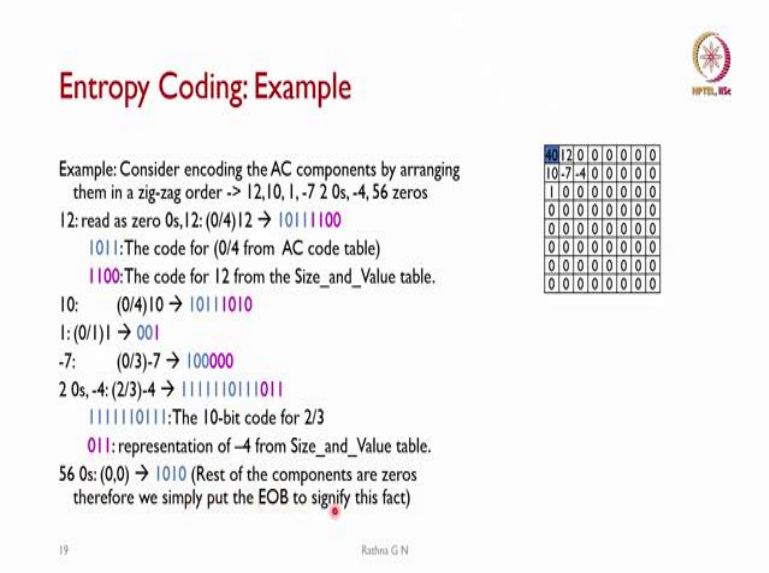
So, you will be seeing that here lot of them have zeros, how you will take the run length and then code it basically? So, the last value will be 0, 0 is the end of block for the 8 by 8 block. So, this one Huffman coded so, you have to see this AC code table for what is the run size? And what is the code length? And then what is the code which is going to be represented? And then this will be the run size for these values.

Otherwise, if you are transmitting this length of bits so, you know that how many bits are required to transmit? So, you will be this thing sending it as this way, so that you will be reducing your code length. So, in this case S 2 what you will be having S 1, S 2? So, S 2 will

be the value. So, this is the run length or size what you will be sending it. And then the value what you will exactly put in here.

Value of the AC component size and then value table this is size and then now you have the code length and then code here.

(Refer Slide Time: 36:31)



So, the value will be from here. So, this is how you would be doing the entropy coding example what it is shown in this. So, it is in the as you see only these few coefficients what you have the values rest of them are zeros. So, you will be seeing that zigzag order it is 12, 10, 1, –7, 2 and then zeros. So, –4 after that you are sending it as 56 zeros. So, it is 12 read as zeros. So, you will be representing as 0 slash 4 that is 12 is 1 0 1 1 1 1 0 0.

So, you will be reading it as these four bits basically that is 12 what you will be sending it 1 0 1 1 the code for what is your thing is, 0 slash 4 from AC code table. And 1 1 0 0, the code for 12 from the size and then value table. So that is what you will be sending it and then for 1 0 what you have it is 0 slash 4 which is given with respect to this value what you will be having 1 0 in the end.

And then if it is 1, you will be seeing 0 or 1, 1 how many of them 0 0 1 in the last and –7 the next one, these are the values what you are sending in this is sorry it is not 1 0 it is 10 basically decimal value and this is a 1 decimal value so, you will be sending it as this. Next one is –7 so, you will be needing 3 bits what you want to send it. So, –7 is sent as this and then next one what you have is two 0s in between –7 and then –4.

So, you will be sending it in this fashion and the 10 bit code for 2, 3 what it is presented. And the last one is –4, what you will be representing it size and value table. So, you will be having what is it? Last one 56 zeros have been represented. So, which is 0, 0. 1 0 1 0 rest of the components are zeros. Therefore, we simply put the end of block to signify to show that this is the end of the block.

(Refer Slide Time: 39:09)

So, this is what how the coding has been implemented in jpeg. So, the rest of the components what we will see it in the next class. DCT will continue and then we will see how quantization is going to be done? Thank you, happy learning and then have a nice day.