6.003: Signals and Systems

Sampling and Quantization

April 29, 2010

What to do with a billion transistors

Gene Frantz, Texas Instruments
Seminar, today, 32-155, 4pm

We are getting closer to a time when we will be able to cost effectively integrate billions of transistors on an integrated circuit. In fact, we are seeing the beginning of this era with the broad adoption of multi-processing system-on-chips, which has both advantages and disadvantages that should be considered. This talk will discuss the options we have, the issues we must face and the future we can look forward to.

Last Time: Sampling

Sampling allows the use of modern digital electronics to process, record, transmit, store, and retrieve CT signals.

- audio: MP3, CD, cell phone
- pictures: digital camera, printer
- video: DVD
- everything on the web

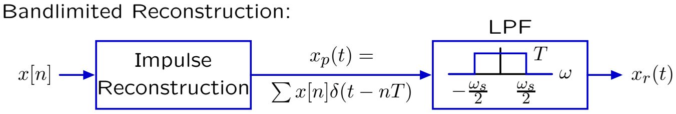
Last Time: Sampling

Theory

Sampling:

$x(t) \to x[n] = x(nT)$

Bandlimited Reconstruction:



Sampling Theorem: If $X(j\omega) = 0 \ \forall \ |\omega| > \frac{\omega_s}{2}$ then $x_r(t) = x(t)$.

Practice

Aliasing $\to$ anti-aliasing filter

Today

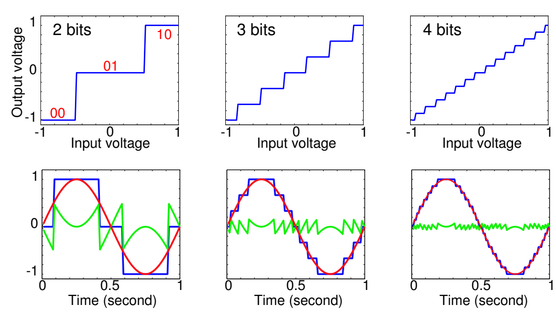
Digital recording, transmission, storage, and retrieval requires discrete representations of both time (e.g., sampling) and amplitude.

- audio: MP3, CD, cell phone
- pictures: digital camera, printer
- video: DVD
- everything on the web

Quantization: discrete representations for amplitudes

Quantization

We measure discrete amplitudes in bits.



Bit rate = (# bits/sample)×(# samples/sec)

Check Yourself

We hear sounds that range in amplitude from 1,000,000 to 1.

How many bits are needed to represent this range?

1. 5 bits
2. 10 bits
3. 20 bits
4. 30 bits
5. 40 bits

Quantization Demonstration

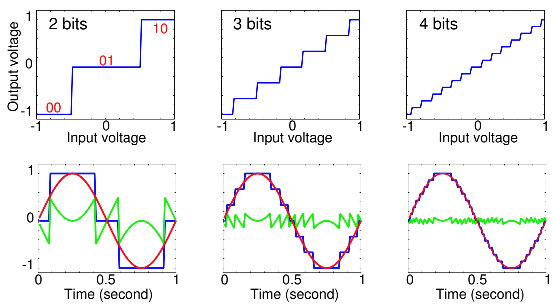
Quantizing Music

- 16 bits/sample
- 8 bits/sample
- 6 bits/sample
- 4 bits/sample
- 3 bits/sample
- 2 bit/sample

J.S. Bach, Sonata No. 1 in G minor Mvmt. IV. Presto
Nathan Milstein, violin

Quantization

We measure discrete amplitudes in bits.



Example: audio CD

$$2\,\text{channels} \times 16\,\frac{\text{bits}}{\text{sample}} \times 44,100\,\frac{\text{samples}}{\text{sec}} \times 60\,\frac{\text{sec}}{\text{min}} \times 74\,\text{min} \approx 6.3\,\text{G bits}$$

$$\approx 0.78\,\text{G bytes}$$

Quantizing Images

Converting an image from a continuous representation to a discrete representation involves the same sort of issues.

This image has 280×280 pixels, with brightness quantized to 8 bits.

Check Yourself

What is the most objectionable artifact of coarse quantization?

8 bit image

4 bit image

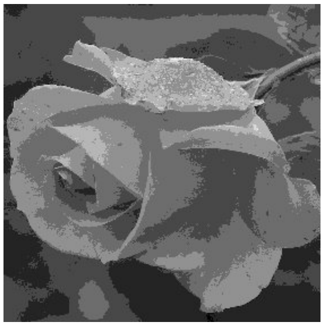
Dithering

Dithering: adding a small amount ($\pm\frac{1}{2}$ quantum) of random noise to the image before quantizing.

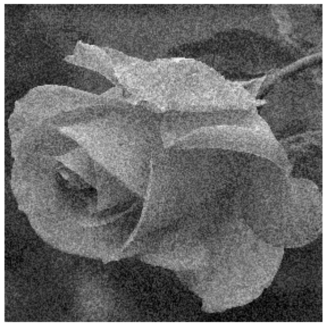
Since the noise is different for each pixel in the band, the noise causes some of the pixels to quantize to a higher value and some to a lower. But the average value of the brightness is preserved.

Check Yourself

What is the most objectionable artifact of dithering?



3 bit image

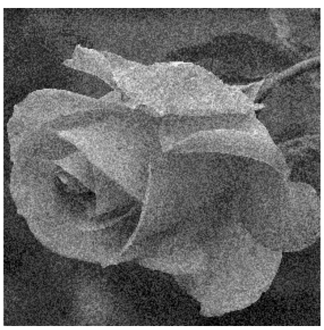


3 bit dithered image

Robert's Technique

Robert's technique: add a small amount ($\pm\frac{1}{2}$ quantum) of random noise before quantizing, then subtract that same amount of random noise.

Quantizing Images with Robert's Method



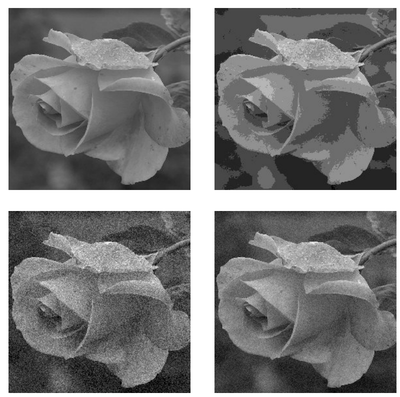
3 bits with dither

3 bits with Robert's method

Quantizing Images: 3 bits

8 bits

3 bits



dither

Robert's

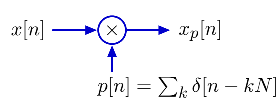
Progressive Refinement

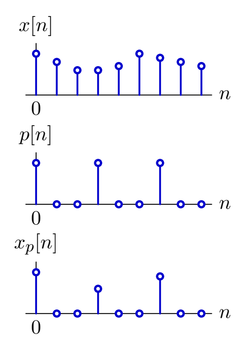
Trading precision for speed.

Start by sending a crude representation, then progressively update with increasing higher fidelity versions.

Discrete-Time Sampling (Resampling)

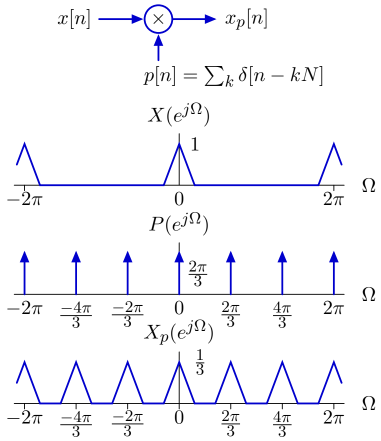
DT sampling is much like CT sampling.





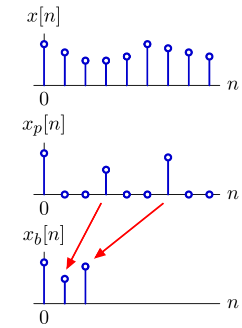
Discrete-Time Sampling

As in CT, sampling introduces additional copies of $X(e^{j\Omega})$.



Discrete-Time Sampling

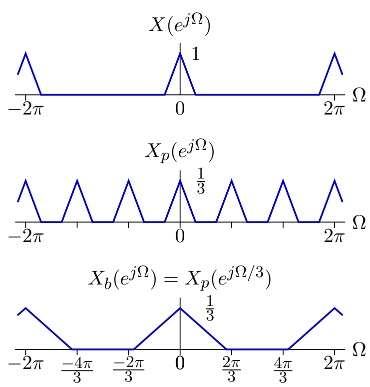
Sampling a finite sequence gives rise to a shorter sequence.



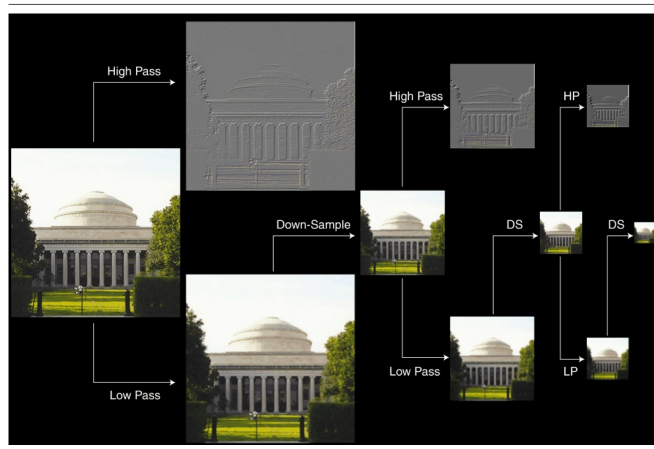
$$X_b(e^{j\Omega}) = \sum_n x_b[n]e^{-j\Omega n} = \sum_n x_p[3n]e^{-j\Omega n} = \sum_k x_p[k]e^{-j\Omega k/3} = X_p(e^{j\Omega/3})$$

Discrete-Time Sampling

But the shorter sequence has a wider frequency representation.



Discrete-Time Sampling



Discrete-Time Sampling: Progressive Refinement

JPEG

Example: JPEG ("Joint Photographic Experts Group") encodes images by a sequence of transformations:

- color encoding
- DCT (discrete cosine transform): a kind of Fourier series
- quantization to achieve perceptual compression (lossy)
- Huffman encoding: lossless information theoretic coding

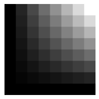
We will focus on the DCT and quantization of its components.

- the image is broken into 8×8 pixel blocks
- each block is represented by its 8×8 DCT coefficients
- each DCT coefficient is quantized, using higher resolutions for coefficients with greater perceptual importance

JPEG

Discrete cosine transform (DCT) is similar to a Fourier series, but high-frequency artifacts are typically smaller.

Example: imagine coding the following 8×8 block.

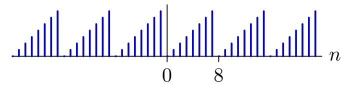


For a two-dimensional transform, take the transforms of all of the rows, assemble those results into an image and then take the transforms of all of the columns of that image.

JPEG

Periodically extend a row and represent it with a Fourier series.

$x[n] = x[n+8]$



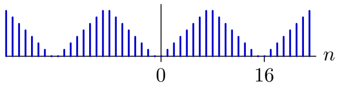
There are 8 distinct Fourier series coefficients.

$$a_k = \frac{1}{8} \sum_{n=<8>} x[n] e^{-jk\Omega_0 n} \; ; \quad \Omega_0 = \frac{2\pi}{8}$$

JPEG

DCT is based on a different periodic representation, shown below.

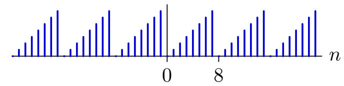
$y[n] = y[n+16]$



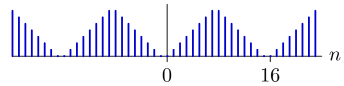
Check Yourself

Which signal has greater high frequency content?

$x[n] = x[n+8]$



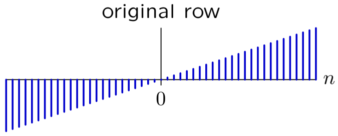
$y[n] = y[n+16]$



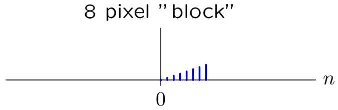
JPEG

Periodic extension of an 8×8 pixel block can lead to a discontinuous function even when the "block" was taken from a smooth image.

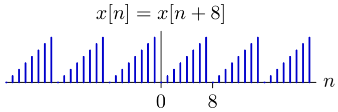
original row



8 pixel "block"



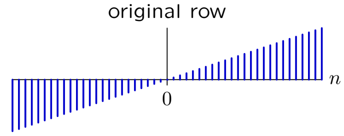
$x[n] = x[n+8]$



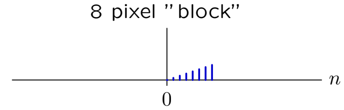
JPEG

Periodic extension of the type done for JPEG generates a continuous function from a smoothly varying image.

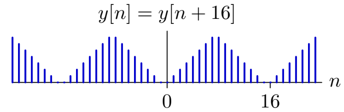
original row



8 pixel "block"

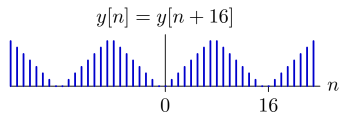


$y[n] = y[n+16]$



JPEG

Although periodic in $N = 16$, $y[n]$ can be represented by just 8 distinct DCT coefficients.



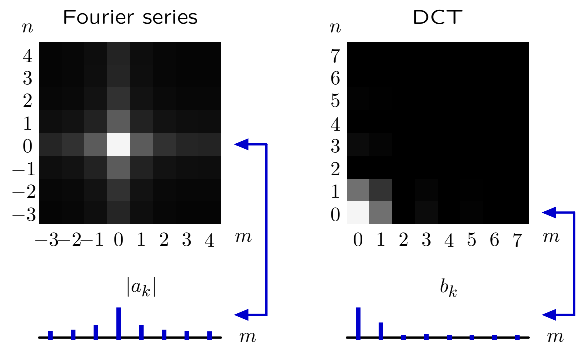
$$b_k = \sum_{n=0}^{7} y[n] \cos\left(\frac{\pi k}{N}\left(n + \frac{1}{2}\right)\right)$$

This results because $y[n]$ is symmetric about $n = -\frac{1}{2}$, and this symmetry introduces redundancy in the Fourier series representation.

Notice also that the DCT of a real-valued signal is real-valued.

JPEG

The magnitudes of the higher order DCT coefficients are smaller than those of the Fourier series.



JPEG

Humans are less sensitive to small deviations in high frequency components of an image than they are to small deviations at low frequencies. Therefore, the DCT coefficients are **quantized** more coarsely at high frequencies.

Divide coefficient $b[m,n]$ by $q[m,n]$ and round to nearest integer.

$q[m,n]$				m $\rightarrow$				
	16	11	10	16	24	40	51	61
	12	12	14	19	26	58	60	55
	14	13	16	24	40	57	69	56
n	14	17	22	29	51	87	80	62
$\downarrow$	18	22	37	56	68	109	103	77
	24	35	55	64	81	104	113	92
	49	64	78	87	103	121	120	101
	72	92	95	98	112	100	103	99

Check Yourself

Which of the following tables of $q[m,n]$ (top or bottom) will result in higher "quality" images?

$q[m,n]$				m $\rightarrow$				
	16	11	10	16	24	40	51	61
	12	12	14	19	26	58	60	55
	14	13	16	24	40	57	69	56
n	14	17	22	29	51	87	80	62
$\downarrow$	18	22	37	56	68	109	103	77
	24	35	55	64	81	104	113	92
	49	64	78	87	103	121	120	101
	72	92	95	98	112	100	103	99

$q[m,n]$				m $\rightarrow$				
	32	22	20	32	48	80	102	122
	24	24	28	38	52	116	120	110
	28	26	32	48	80	114	139	112
n	28	34	44	58	102	174	160	124
$\downarrow$	36	44	74	112	136	218	206	154
	48	70	110	128	162	208	226	194
	98	128	156	174	206	256	240	202
	144	184	190	196	224	200	206	198

JPEG

Finally, encode the DCT coefficients for each block using "run-length" encoding followed by an information theoretic (lossless) "Huffman" scheme, in which frequently occuring patterns are represented by short codes.

The "quality" of the image can be adjusted by changing the values of $q[m,n]$. Large values of $q[m,n]$ result in large "runs" of zeros, which compress well.

JPEG: Results


1%: 1666 bytes


10%: 2550 bytes


20%: 3595 bytes


40%: 5318 bytes


80%: 10994 bytes


100%: 47k bytes