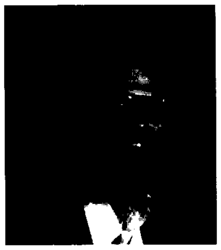
NTSC: Nice Technology, Super Color

James F. Blinn, *California Institute of Technology*



The NTSC encoding scheme is one of the most amazing technical achievements of our time. Done well, it can look really good.

NTSC, the color TV encoding scheme used in the U.S., has gotten a bad rap. Everybody seems to hate it. But in my opinion the NTSC encoding scheme is one of the most amazing technical achievements of our time. Admittedly, NTSC does reduce the resolution of an image, but not as much as most people think. Done well, it can look really good; look at a videodisc. Computer graphics images, though, often look messy when converted to NTSC video, even more so than the same image created with a TV camera. Why is this? In this column I'll talk about the theory of NTSC, describe some ways we as computer graphicists can cooperate with NTSC hardware encoders and decoders to make better quality images, and show the results of some semiscientific experiments on the signal. There are still a few things that puzzle me about the system, though, and I'll mention them too.

NTSC history

The main problem with the invention of color television is the same one that plagues us in computerland—downward compatibility. The color encoding system had to work transparently with existing black and white televisions. A color signal fed to an existing black and white set (built before the color system was invented) had to produce a black and white version of the picture. Furthermore, a black and white signal fed to a color set had to come out as a black and white image. And the signal had to fit into about the same band-width that the original black and white signal took up. (Perhaps NTSC can get more respect by describing itself as an early form of lossy data compression.) To make this happen, the design engineers applied some clever frequency domain encoding as well as some knowledge of the human visual system.

Fourier transform theory

Before diving in and showing signals, let's review a little signal processing theory. I'm going to look at the NTSC waveform in three different ways:

1. In the time domain (the signal waveform itself)
2. In the space domain (as the interpretation of the waveform according to the raster scan)
3. In the frequency domain (as the Fourier transform of the signal)

Just as a reminder, the Fourier transform of a signal is a complex valued function (generally interpreted as a magnitude and phase for each frequency). I will typically only show the magnitude of the transform and simply call it the frequency spectrum. Just be aware that the phase information is not shown in these plots.

As shown in Figure 1, to do filtering operations on the spectrum, we multiply the spectrum by some desired frequency response function. In the time domain, this translates into an operation called *convolution* (a sort of sliding weighted average). The inverse Fourier transform of the frequency response gives us the weight values. When I refer to this in the space domain, I'll usually just list these weight values. For example, a "(1, 2, 1) filter" means to take two times the current sample and add it to one times the previous and subsequent samples. Divide by the sum of all the weights, and the result is the output value at the current sample.

The black and white signal in frequency space

First let's look at the signal for a black and white image scanned with a TV raster and see what its frequency spectrum looks like. We will deal with several useful frequencies in this

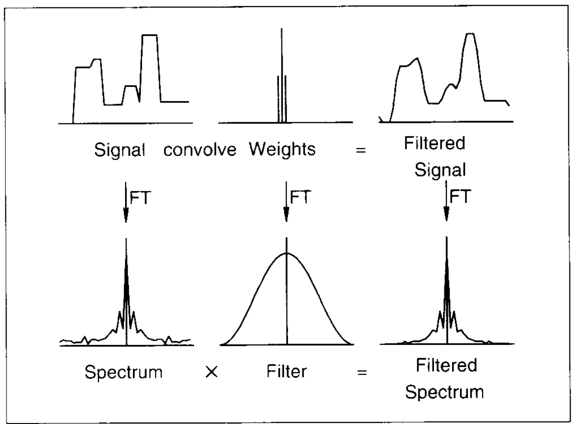


Figure 1. Filtering in time and frequency domains.

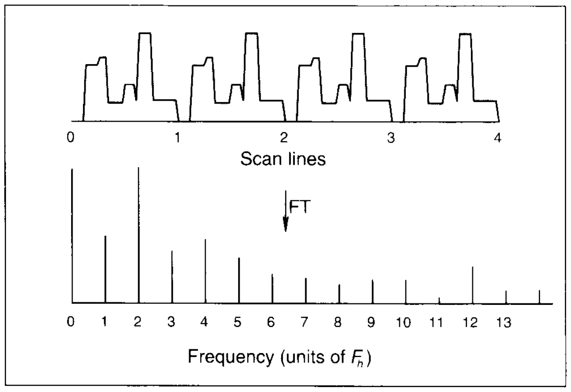
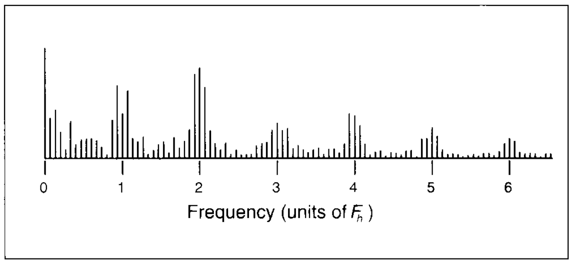


Figure 2. The spectrum of a repeated scan line.

Figure 3. The spectrum of a whole frame.



column, listed in Table 1. Notice that I have chosen a frequency of 14.31818 MHz to define the smallest time unit. This is the pixel rate for the digital version of NTSC. Even though I'm not explicitly talking about digital video here, this clock rate is a convenient one to use. It is four times the frequency of the color subcarrier (the use of which I will define shortly) and it means that, theoretically, we can represent frequencies of up to twice the subcarrier frequency. There must be 227.5 cycles of subcarrier per horizontal line. There are 525 lines per frame, but the frame is divided into fields of 262.5 lines each. The designers of the NTSC system carefully selected these frequency ratios. I'll give you an idea of why they were selected below.

One line

Consider a single scan line. Its intensity profile and frequency spectrum might look like the two left-hand plots of Figure 1. Note that the image is all positive and the DC (or zero frequency) component is just the integrated intensity over the whole line. The magnitude of the frequencies dies down as the frequency gets higher. Since such a spectrum is symmetric at about frequency 0, I'll usually only plot the right half of it.

Many lines

If this scan line is repeated over and over, the signal becomes periodic and has a discrete Fourier transform, as shown in Figure 2. Note that the frequency components are spaced at intervals of the horizontal line frequency F_h. Also, in this and later figures, I truncated the spike at frequency 0 so that I could scale up the rest of the frequencies to be visible.

Table 1. Useful frequencies.

Event	Symbol	Frequency	
		Hz	Events/line
Pixel clock	F_p	14.31818 M	910
Subcarrier cycle	F_{sc}	3.579545 M	227.5
Scan line	F_h	15.73426 K	1
Field	F_{fld}	59.94	1/262.5
Frame	F_{frm}	29.97	1/525

Noninterlaced image

If the scan lines are not all the same, the signal will still be periodic, but it will repeat at the frame rate. (We assume the image isn't moving, an assumption I will stick to for the time being.) The Fourier transform of the signal is still discrete, but more finely divided, with frequency components spaced at intervals of F_{frm}. In a typical image, the scan lines do not change radically from one to the next, so there are still peaks in the spectrum at multiples of the line frequency (see Figure 3).

Let's take time out for a visualization discussion. In the actual NTSC signal, there are 525 gaps of width F_{frm} between every integer multiple of F_h. If I draw all of these, the diagrams would look solid black. To make details more visible, I created Figures 3 through 11 with a sort of abbreviated NTSC signal. I eliminated horizontal and vertical blanking (although that doesn't change the signal much), and I used 15 lines per frame and 9.5 subcarrier cycles per line. In the text of this article, however, I'll always use the true values of 525 and 227.5. In the figures, whenever you see something happening at a frequency of $9.5F_h$, remember that in the actual signal it happens at the frequency $227.5F_h$.

Notice that there are two levels of detail in Figure 3. The peaks at integer multiples of the line rate, the "macrostructure," represent the horizontal detail in the image. Between each pair of line rate peaks there is a "microstructure" of 525 frequencies spaced at the frame rate (since $F_h = 525\ F_{frm}$). These represent the vertical detail of the image. In fact, if the microstructures all had zero amplitude, we would be back to the case of identical scan lines with no vertical detail. If the image were simply blurred vertically, the microstructure amplitudes would be very small at frequencies halfway between the line-rate frequencies.

As an exercise, let's examine the above two statements as filtering problems. A filter that will set the microstructure values to zero will have value 1 at multiples of the line rate and be zero everywhere else. A plot of this filter would look like a comb, so it's called a *comb filter*. (Aren't engineers clever?) The inverse Fourier transform of the comb filter is another comb with teeth separated by the duration of one scan line. That is, the weights look like (. . . 0, 1, 0, . . . , 0, 1, 0, . . . , 0, 1, 0 . . .), where there are enough zeros between the ones to

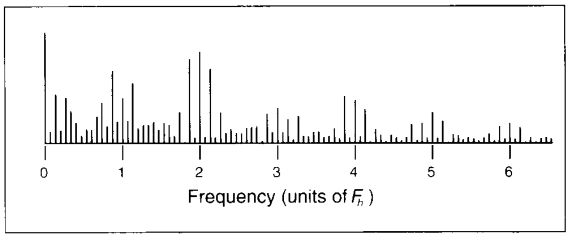


Figure 4. The spectrum of an interlaced image.

make up one scan line duration. Interpreting this according to the raster we see that the unit weight values all line up vertically. We therefore have a (. . . , 1, 1, 1, . . .) filter that runs vertically. In other words, we must replace each pixel by the average of the entire column that it's in. Voila, we have an image with identical scan lines.

If we wanted to blur the image just a little bit, we might instead use a (1, 2, 1) filter vertically. The Fourier transform of this looks like a comb with fat teeth. It has value 1 at multiples of the frame rate (and so keeps them intact), has value 0 halfway between (and so eliminates them), and smoothly varies between. We'll have more use for this filter later.

Interlaced image

Television signals are not scanned out simply from left to right and top to bottom, because the 29.97 Hz frame rate would look too flickery. To minimize flicker, the signal is interlaced. This means that the odd numbered lines are scanned first, then the even numbered ones. The vertical retrace rate is twice the frame rate, giving the visual impression of 59.94 flashes per second. This does not in fact make a radical difference on the frequency spectrum; there are still 525 lines per image, and the frequency spectrum is still discrete with a spacing of the frame rate. The values at integer multiples of the line rate are even the same. (Remember that these values stay intact if we vertically blur completely. It wouldn't matter if we started with a normal or an interlaced raster.)

Interlacing just changes the values at the microstructure frequencies. An easy way to think of this is to "unwrap the interlacing" and consider it as a noninterlaced image with the top and bottom halves almost identical. This introduces a strong bias toward components at 2, 4, 6, and so forth times the frame rate. A slight complicating factor comes from the fact that there are an odd number of total lines and thus a half integer number of lines per field. This complication makes the microstructure of the spectrum look something like Figure 4. Note that the first nonzero frequency (29.97 Hz) has a very small value in Figure 4, but a large value in Figure 3; that's the flicker that interlacing is designed to eliminate.

Putting three pounds in a one pound sack

A color image has separate red, green, and blue components, each with a frequency spectrum similar to the above. Now how do we combine these three into one signal with about the same total bandwidth? NTSC designers used two tricks. First, they converted RGB to a color space that is based on human perception and requires less bandwidth. Second, they used the cracks in the spectrum of the signal.

Chroma space

The human eye can perceive abrupt transitions in brightness much more readily than in hue. Furthermore, the eye is more sensitive to transitions in the orange-blue color range (where flesh tones lie) than in the purple-green range. Since NTSC takes advantage of this, the first operation is to transform RGB into a color space whose axes align with these visual ranges. The transformation is

$$Y = 0.299R + 0.587G + 0.114B$$
$$I = 0.596R - 0.275G - 0.321B$$
$$Q = 0.212R - 0.523G + 0.311B$$

In my column on "The World of Digital Video" (*CG&A*, September 1992, pp. 106-112) I gave some of the motivation for the numerical coefficients used above. I just want to add a few other points here.

The Y signal represents the brightness, technically called *yluminance* (the *y* is silent) of the image. This is the only signal that a black and white TV needs. The conversion coefficients are based on the relative brightness response of the human eye to red, green, and blue. (Okay. There's no such word as yluminance. That was a joke.)

Given Y, we need two color coordinates that evaluate to zero when $R = G = B$. The simplest such is $R - Y$ and $B - Y$, but these values don't align with the desired color sensitivities. The I (which is the orange-blue axis) and Q (which is the purple-green axis) are simply a scaling and rotation of $R - Y$ and $B - Y$ by 33 degrees.

$$I = \frac{R-Y}{1.14}\cos 33 - \frac{B-Y}{2.03}\sin 33$$

$$Q = \frac{R-Y}{1.14}\sin 33 + \frac{B-Y}{2.03}\cos 33$$

Band-limiting the NTSC primaries

Just changing coordinate systems from RGB to YIQ doesn't do any data compression. Now it's time to compress. We band-limit the Y, I, and Q signals according to the NTSC broadcast standard shown in Table 2. The limitations on I and Q might seem extreme, but they were determined by measurements on what the human visual system can detect when viewing the screen from a typical viewing distance.

By the way, to cram six hours of video onto one teeny tape, VHS video recorders further cut the Y signal down to about 3.2 MHz and the I signal to 0.63 MHz. So VHS is considerably fuzzier than broadcast standard NTSC.

Table 2. Bandwidth of NTSC primaries.

Signal	Maximum frequency		
	MHz	Cycles/pixel	Cycles/line
Y	4.2	1/3.4	267
I	1.5	1/9.5	96
Q	0.55	1/26	35

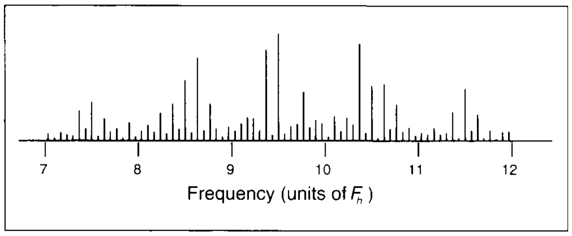


Figure 5. The spectrum of modulated *I* and *Q*.

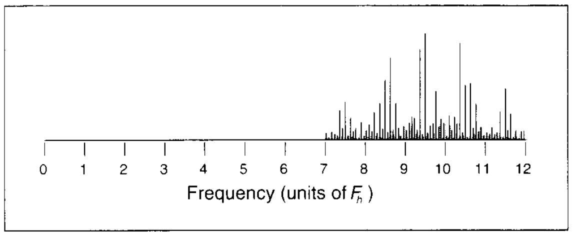


Figure 6. The spectrum of the full NTSC signal.

As you might expect, computer-generated images can, and often do, greatly exceed these bandwidth limitations. A typical video frame buffer has a pixel rate of $4 \times F_{sc}$. Since we can put any values in the pixels, we can easily generate frequencies of $2 \times F_{sc} = 7.16$ MHz by simply alternating black and white pixels or purple and green pixels. I will describe the visual results of this extreme behavior in the next section. The main punch line of this article is that we can't get away with this. An NTSC image will only look good if we filter these three signals (Y, I, and Q) before combining them into one composite signal.

Incidentally, it's interesting to work out how many pixels are actually necessary (in the information theoretic sense) per scan line to represent these signals. We need to know this, for example, for archiving purposes. If you downsample each signal (with proper filtering), you would need at least two times as many pixels as the maximum allowed frequency. (Actually, it's best to use somewhat more than this, but we'll see what happens using just the absolute minimum number.) Then you must take into account the fact that only about 85 percent of a scan line contains image information; the rest is devoted to horizontal blanking. So we multiply the last column of Table 2 by two and by 0.85 and get the minimum number of image pixels needed per scan line for Y, I, and Q of 454, 163, and 60. These surprisingly small values are all the pixels necessary to represent a line of the best broadcast quality video.

Combining I with Q

The NTSC process combines I and Q into one signal, called *chroma*, by a technique called *quadrature modulation*. The modulated signal is basically a sine wave of frequency F_{sc}, called the *color subcarrier*, whose amplitude, $\sqrt{I^2 + Q^2}$, is the saturation of the color and whose phase, $\tan^{-1}(Q/I)$, is the hue. Mathematically it's

$$C = I \cos(F_{sc}t) + Q \sin(F_{sc}t)$$

Multiplying the I and Q signals in the time domain by a sine wave of frequency F_{sc} is the same as convolving its Fourier transform in the frequency domain with two impulses at frequencies $\pm F_{sc}$. This simply makes copies of I and Q centered around $\pm F_{sc}$. A plot of the positive copy appears in Figure 5. Notice that the frequencies don't line up exactly with the line rate; they're shifted by $1/2F_{frm}$. Also, even though the I and Q spectra are symmetric about frequency 0, the C spectrum is not symmetric about the subcarrier frequency because of the different phases applied when modulating I and Q. This is a case where the magnitude plots I'm using don't tell the whole story.

Now how about the reverse process? To later extract the I signal from the C signal—that is, to demodulate it—you simply multiply by $2\cos(F_{sc}t)$, which gives

$$\begin{aligned} 2C\cos(F_{sc}t) &= 2I\cos^2(F_{sc}t) + 2Q\sin(F_{sc}t)\cos(F_{sc}t) \\ &= I + I\cos(2F_{sc}t) + Q\sin(2F_{sc}t) \end{aligned}$$

You then apply a low-pass filter to remove the last two double frequency terms. You can get Q back similarly by multiplying by $2\sin(F_{sc}t)$ and filtering.

Now wait a minute. There's something fishy going on here. We've taken two signals, crammed them together onto one wire, and we've been able to separate them back out again. It looks like we're getting something for nothing. Well . . . we're not. One way to see why is to look in the time domain. Pretend for a moment that we have a digital pixel stream. The value of $\sin(F_{sc}t)$ for successive pixels has the values 0, 1, 0, −1, 0 . . . and $\cos(F_{sc}t)$ has the values 1, 0, −1, 0, 1, ... Multiplying the I_i and Q_i samples by these values and adding the result gives us

$$+I_0 \quad +Q_1 \quad -I_2 \quad -Q_3 \quad +I_4 \quad \ldots$$

In other words, quadrature encoding basically samples I and Q at half the pixel frequency (which is twice the subcarrier frequency F_{sc}), fiddles the signs, and interleaves the samples. The whole trick is to realize that this works only if the I and Q signals don't violate the sampling theorem; that is, they shouldn't contain frequencies above half the sampling rate. This means that we must lop off any frequencies in I and Q that are above F_{sc}. What we've really done is take two signals, each with a bandwidth of F_{sc}, and combined them into a signal that has a bandwidth of $2 \times F_{sc}$. No magic after all. In fact, the previous section shows that I and Q are supposed to be filtered quite a bit below this limit. But, you guessed it: Computer-generated signals can easily violate this, effectively giving you aliasing in the chroma signal.

Combining chroma with Y

A recent invention called S-Video stops at this point and provides the two signals Y and C on two separate wires. But that's not good enough for broadcast video. We must combine them into one signal. We do this by simply adding Y and C together.

Again we must ask ourselves, "Why do we think the signals are separable after simply adding them?" Here's where the cleverness of the frequency choices pays off. Look back at Figure 4. The Y, I, and Q signals have discrete frequency spectra (for frames that don't change in time) with spacing of F_{frm}. We formed the C signal by shifting I and Q by F_{sc}, which happens to equal $119{,}437.5 \times F_{frm}$. This means that the frequency spectrum of C (see Figure 5) fits within the cracks of the frequency spectrum of Y (see Figure 4). The result is Figure 6. For clarity, I show the frequency components that came from the Y signal in gray. Notice that two levels of interleaving go on

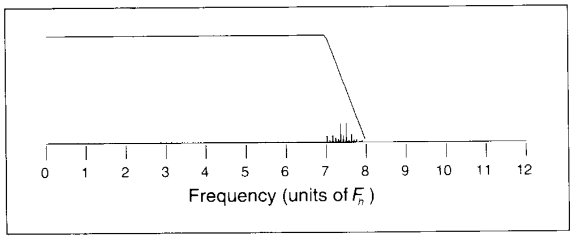


Figure 7. Separating Y and Q with a low-pass filter.

here: the microstructure (representing vertical detail) and the macrostructure (representing horizontal detail). The macrostructure (which is usually all that is shown in descriptions of NTSC) consists of the line-rate peaks in the spectrum. Here the C signal is shifted by $F_{sc} = 227.5F_h$ and so fits in the line-rate cracks. The microstructure interleaves because the odd number of scan lines forces the $227.5F_h$ frequency point *not* to land on a microstructure frequency.

NTSC decoders

Now that we have the Y and C signals crammed together, let's see how a TV receiver can pull them apart again. There are several degrees of sophistication in NTSC decoders, which vary in expense and quality. We can evaluate them according to how well they separate Y from C. Errors will consist of some of the C signal remaining in the separated out Y and vice versa. C remaining in Y looks like dots crawling up vertical edges in the image, the so-called "chroma crawl." Parts of the Y signal remaining in the separated out C signal visually look like rainbows superimposed on what should be a monochromatic image. In the discussions below, I'll describe how excessively high frequencies from a digitally generated signal can exacerbate these problems.

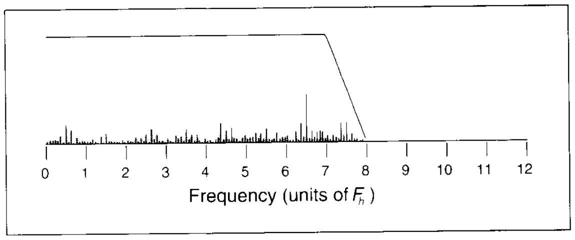
In what follows, I'll deal with just the problem of extracting Y from $Y + C$. Once we do this, the best decoders recover the C signal by subtracting Y from the composite. Others cop out and just demodulate the whole composite signal to get I and Q, keeping the interleaved components of Y. This sort of works because the sizes of the Y signal near the color carrier frequency are typically small.

Low-pass filter

Older and cheaper receivers simply used a low-pass filter with cutoff at about 3 MHz to get rid of chroma. (Older black and white TVs do this implicitly because of their limited high-frequency sensitivity.) In Figure 7, I plotted a low-pass filtered version of Figure 6 with the frequency filter function plotted on top of it. You can see that some of the C signal remains, but not a lot. This technique also makes the image somewhat blurry because the high frequencies of Y are removed.

Computer-generated signals with excessive bandwidth in C exaggerate the chroma crawl problems with this type of decoder because the C signal sticks out further into the cutoff region of Y. For example, first note that the signal in Figures 6 and 7 has a C component that was band-limited to the proper small range near the color carrier. Figure 8 is the same as Figure 7 except I started with I and Q signals that were not band-limited. You can see that there are significant components of

Figure 8. Improper separation resulting from too much bandwidth in C.



C at frequencies far from the subcarrier (especially at about $6.5\ F_h$ in this figure) that the filter doesn't get rid of.

Notch filter

A somewhat more modern approach is to use a notch filter centered at F_{sc} to remove the chroma from Y. This retains higher frequencies above F_{sc} and thus makes for a sharper image. But notch filters still remove some of the good Y frequencies, and they have the same problems with excessive bandwidth in C.

Comb filter at line rate

The interleaving of frequencies for Y and C make their separation an ideal job for a comb filter. Let's start out by trying a comb with fat teeth spaced at the line rate. We've seen this before. It's a (1, 2, 1) filter vertically. In practice, comb filter decoders usually use just a single line delay, so the filter effectively has weights (1, 1, 0). That is, each scan line is averaged with the preceding one. Because it seems to shift the image upward, the single line delay is not as good as a (1, 2, 1) filter, but it's cheaper. Figure 9 shows the effect of this filter on Figure 6. Again, I plotted the filter function on top of the filtered signal. Notice that the frequency components at half integer multiples of the line rate have been stamped out.

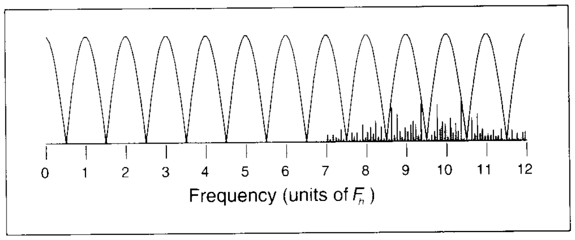


Figure 9. Comb filter separation of Y and C.

We can understand the scan-line-averaging approach in the spatial domain by noting that the chroma signal switches sign from one scan line to the next. This switch is a consequence of the half integer (227.5) number of color carrier cycles per scan line. As long as the image doesn't change color drastically from one line to the next, the color portions of the signal cancel out. For example, a pixel might have the value $Y_{i,j} + I_{i,j}$ while the one below it is $Y_{i,j+1} - I_{i,j+1}$. As long as $I_{i,j} \approx I_{i,j+1}$, the cancellation will work.

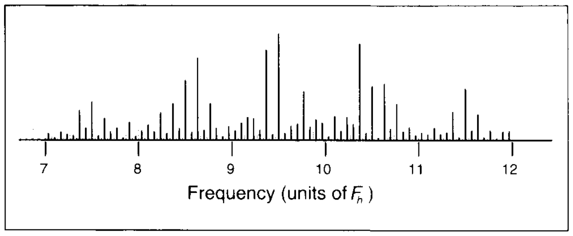


Figure 10. Comb filter problems.

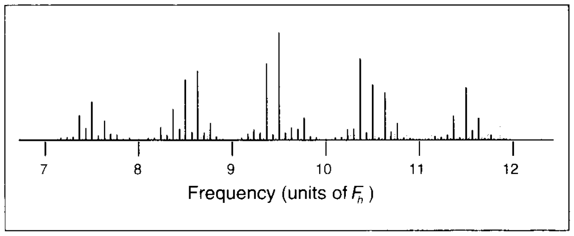


Figure 11. The vertical filter applied to *YIQ* before encoding.

The problem with the comb filter is that it can generate horizontal chroma crawl dots. This occurs when the color *does* change quickly from one scan line to the next. Look at Figure 10, a zoomed-in version of Figure 6 near the color-carrier frequency. The comb filter is going to keep frequencies that are near to integer multiples of the line rate. The Y components near these frequencies represent horizontal detail. The C components near these frequencies lie in the microstructure portion of C and thus represent vertical chroma detail. Even though they have much smaller magnitudes than the macrostructure frequencies of C (which the comb filter gets rid of), the magnitudes can still be greater than those of the Y signal. In other words, vertical chroma detail comes out of the comb filter interpreted as horizontal brightness detail and actually overshadows legitimate horizontal brightness detail. Sure enough, computer images tend to do this too.

Comb filter at frame rate

We can separate the Y and C signals more perfectly by using a finer comb filter, one whose teeth are spaced at the frame frequency. A plot of the signal subjected to this filter looks exactly like Figure 4. In the spatial domain, the finer comb filter translates into averaging each pixel with the same pixel from the previous and subsequent frames. (The eye does some of this implicitly—another reason for the design of frequencies like this.) The finer filter works because the chroma changes sign not only from one line to the next but also from one frame to the next, a consequence of the $227.5 \times 525 = 119{,}437.5$ cycles of color carrier per frame. Only the most expensive monitors use this type of filter. The only problem now is if there is motion in the image. If they detect motion, fancy monitors switch automatically between line- and frame-rate combs. I don't have any experience with how good this looks, since Santa didn't bring me one of these monitors for Christmas. And for my purposes, that's okay. I'm really most concerned with how my images are going to look on cheap monitors, since that's my primary distribution medium.

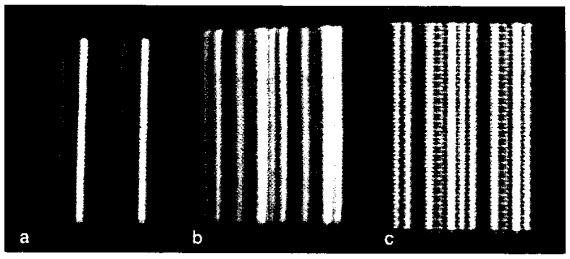


Figure 12. The test pattern on three monitors: (a) an RGB monitor, (b) a low-pass-filter monitor, and (c) a line-rate comb-filter monitor.

How to cooperate with the decoder

As mentioned above, the way to improve the quality of NTSC images is to filter the Y, I, and Q signals to the appropriate frequencies before passing them to an NTSC encoder. This will minimize the chroma that erroneously sticks out into the Y if you are using a low-pass-filter receiver.

Then, to help with comb-filter decoders, you should also filter the chroma in the vertical direction. (TV cameras do this implicitly because of the focusing of their optics.) A vertical (1, 2, 1) filter applied to I and Q before modulating (leaving the Y unfiltered) should work fine. There is some trickiness here, though. The filter must act on three temporally adjacent scan lines. However, because of the interlacing, these three lines will not be visually adjacent. If you address your scan lines according to visual sequencing, your filter should be more like (1, 0, 2, 0, 1). The result is the spectrum in Figure 11. Note that the microstructure frequencies of C from Figure 10 are much attenuated. When we apply a line-rate comb filter later, there won't be any chroma here to leak through.

Some experiments

Okay, that's the theory. Now for some practice. My video setup consists of an RGB frame buffer feeding a transcoder that converts it to the color space (Y, $R - Y$, $B - Y$). This is the color space used internally by Sony Betacam videocassette recorders. The Betacam then has an NTSC encoder for output. Now, some encoders have the proper band-pass filters for Y, I, and Q built in, but apparently this one doesn't. It is, after all, designed to work mostly with signals that come from a camera and that have already been filtered.

I created a test pattern with a lot of high frequencies in I and Q and viewed it on three different monitors: an RGB monitor, a low-pass-filter monitor, and a line-rate comb-filter monitor. The results appear in Figure 12. Each colored bar is six pixels wide, and there are two pixels between each bar. These eight pixels represent two cycles of the color carrier. To make problems more visible, I photographed this and the two subsequent figures at 1/60 second, so what you see is only one field of the image. The images as printed here look slightly different in size, but that's just because my monitors had different size tubes. Note the vertical dots on both the NTSC monitors. This is chroma crawl. Note also the horizontal dots on the comb-filter monitor, especially along the bottom edge of the colored region. I am puzzled by one thing, though. Why

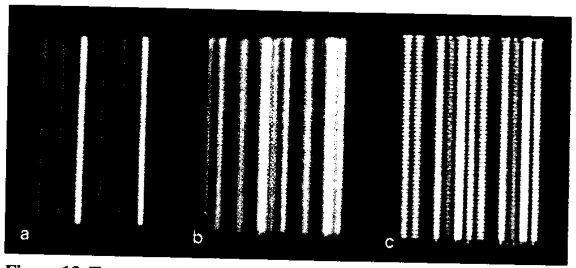


Figure 13. Test pattern filtered horizontally and displayed on three monitors: (a) an RGB monitor, (b) a low-pass-filter monitor, and (c) a line-rate comb-filter monitor.

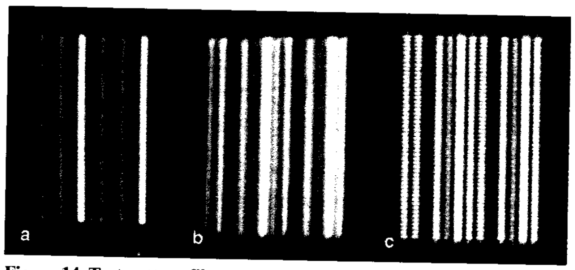


Figure 14. Test pattern filtered horizontally and vertically and displayed on three monitors: (a) an RGB monitor, (b) a low-pass-filter monitor, and (c) a line-rate comb-filter monitor.

are there vertical dots on the comb-filter monitor? Shouldn't these cancel out more perfectly?

The next experiment is to do the proper filtering. Ideally we should filter *Y*, *I*, and *Q* at three different cutoff frequencies. I used a small shortcut and did the following: I took each scan line, converted the pixels to *Y*, *R* − *Y*, and *B* − *Y*. Then I filtered *Y* at a 4.2 MHz cutoff, using a 9-pixel-wide filter with weights (0.0295, −0.0524, −0.118, 0.472, 1, 0.472, −0.118, −0.0524, 0.0295). Then I filtered both *R* − *Y* and *B* − *Y* with a 1.5 MHz cutoff (this is the same as filtering *I* and *Q* at 1.5 MHz). This required a filter that was 29 pixels wide. Like the filter above, it's symmetric, and the first half has weights (0.00474, 0.01739, 0.0295, 0.02743, 0, −0.0524, −0.1123, −0.1467, −0.118, 0, 0.20746, 0.4720, 0.73489, 0.92867, 1, . . .) Finally, I converted the filtered values back to *RGB* and returned them to the frame buffer. The result is Figure 13. Note that the vertical chroma crawl is reduced greatly, though not entirely gone. Maybe filtering *Q* further would help, but I doubt it. This figure is a particularly violent test case. More typical images benefit from filtering more noticeably.

A possible problem with this technique is that filtering in the *Y*, *R* − *Y*, *B* − *Y* space might generate negative values or overflow values in RGB space. None of the pixels in the test pattern hit these limits. If they did, I would have to clamp them between 0 and 255. I don't know how big a problem this would be.

Finally, in an attempt to remove the horizontal chroma dots, I filtered the image vertically. I first tried a (1, 0, 2, 0, 1) filter, but it didn't look too great. Instead, I found good results by using the same spatial filter vertically as I used horizontally. After all, you don't really need any more resolution vertically than you have horizontally. Besides, filtering *Y* vertically helps get rid of 29.97 Hz flicker. The result appears in Figure 14. Again, Figure 14c shows an improvement even though the dots aren't entirely gone.

Summary

Generating a proper NTSC signal has a lot in common with the aliasing problem in computer graphics. High frequencies that the system was not designed to handle can show up as undesirable artifacts in the image. The solution is the same: Filter out the high frequencies before encoding. At first you might say, "Ooh, ick. We're giving up high resolution." But if the high resolution just turns around and bites you with chroma crawl or rainbows, you're better off without it.

People who complain about not enough pixels are likely not using the ones they've got effectively (a theme I will return to in the future). I think it's amusing that many feature films with computer images were generated at thousands of pixels per scan line and, if they weren't very successful at the box office, have been seen on VHS tape (with its 350 pixel resolution) by more people than ever saw them in a movie theater. They might almost as well have generated them at 350 pixels in the first place.

Each of the decoders removes *C* from the composite signal by averaging a pixel with its neighbors. This causes blurring. The low-pass-filter decoder blurs horizontally. The line-rate comb filter blurs vertically. The frame-rate comb filter blurs in time. Maybe it's possible to make a decoder that distributes the blur equally over all three axes. This is something for the hardware people to look into.

And wouldn't you know it? Now that we're just figuring out how to do NTSC right, the Federal Communications Commission is going to take it away from us. Current plans call for the FCC to adopt a new high-definition television standard sometime this year. The FCC will then strongly encourage all broadcasters to switch over to the new standard as soon as possible. By the year 2007, the FCC wants this conversion to HDTV to be complete. Broadcasters will no longer be allowed to use NTSC. Boo hoo. ❑