



THEORY BOOK IT INTRO

FOURTH EDITION

Compiled by
Alf Inge Wang, Rune Sætre, Terje Rydland, Anders Christensen, Guttorm Sindre
NTNU

TDT 4105/4110

Theory Book IT Intro

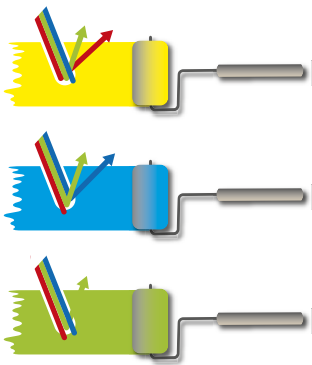
Yellow = R + G?

But, it is not only that $R + G + B = \text{white}$ that is odd. Notice from Figure 8.1 that combining red and green makes yellow. This is very odd because we were taught in elementary school that red and yellow are primary colors—you don't make them out of anything. Also, in elementary school we learned how to make green paint from yellow paint and blue paint. (We called it blue, but it is really light blue, or *cyan*.) So, then why do $R + G = \text{yellow pixels}$?

The mystery is solved by understanding that there is a difference between colored light and colored paint. Paint, like any colored surface, reflects some colors and absorbs others. So, when white light ($R + G + B$) strikes paint, some light is absorbed—we can't see it—and some light is bounced back, that is, reflected. We see the reflected color.

In the case of a pixel, the light shines directly at our eyes. Nothing is absorbed. Nothing is reflected. We just see the pure colored light. The exact color is determined by the intensity of each color.

Green Paint = Blue + Yellow



To see how combining light works with pigment, imagine white light striking yellow paint. Because it is yellow, the colors *reflected* back when white light ($R + G + B$) strikes it must be R and G. B must be absorbed.

When white light ($R + G + B$) shines on cyan paint, it reflects G and B light, as we can figure out from Figure 8.1, and it absorbs R.

When we mix yellow pigment and cyan pigment in art class, the white light ($R + G + B$) is handled as follows: The yellow absorbs B, none is reflected; the cyan absorbs R, none is reflected. The only thing left to reflect—which both yellow and cyan reflect—is G.

Voila! Yellow paint plus cyan paint produces green paint. Just as we learned in art class!

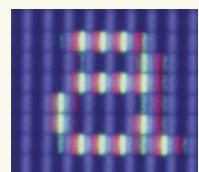
Making a Big Display

To finish our discussion of pixels and colored light, look at Figure 8.2 to see a thin film transistor (TFT) form of liquid crystal display (LCD), known as an *active matrix* display. These are the standard “flat” or “thin” displays used for laptops, phones, and most familiar video applications. At the far left in the figure is an arrow pointer; to its right are two enlargements of it. We see the pixels as red, green, and blue colored lights. Because white is the combination of $R + G + B$ light, we see that the colored pixels are white at a distance; the black pixels have the R, G, and B subpixels turned off. At the far right in the figure is a 2×2 grid of white pixels.

LCD technology is extremely clever. If you are interested, read about it online to see how these tiny lights are manufactured and how they are controlled. It's amazing.



8.3 Colored Light. Describe what a user sees on the screen when viewing the pixels shown here.



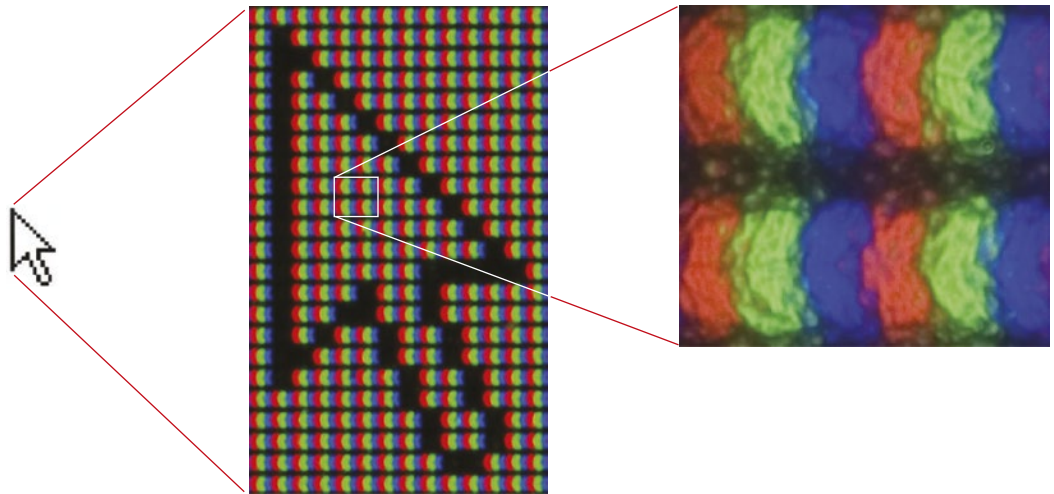
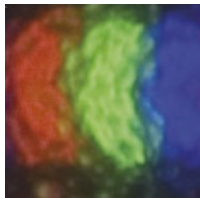


Figure 8.2 At the left is a standard arrow pointer; the other two images are close-ups of its LCD pixel grid. The enlargement in the middle shows the RGB-triple lights that make pixels. Most are at full intensity creating the white background, and some are turned off (zero intensity) to create the arrow outline. A 2×2 white pixel region is further enlarged at the right.

Thinking About Intensities



Some people—Leibniz was one—get pretty excited about binary, but for most of us, they’re just bits. If we think of them as controlling the intensity of the light of a subpixel (a color), however, the idea has some interest. Consider a blue subpixel. The 8 bits specifying its intensity have position values:

128 64 32 16 8 4 2 1

These values specify how bright the blue should be. If we want the subpixel to be “half on,” that is, “half intensity,” we write

128 64 32 16 8 4 2 1
1 0 0 0 0 0 0 0

to say we want 128 units of power. If we want the pixel “three quarters on,” then we write

128 64 32 16 8 4 2 1
1 1 0 0 0 0 0 0

because $3/4$ of the 256 range of values is $192 = 128 + 64$. We would get $7/8$ of the intensity by setting the next bit, and so forth. Obviously, by adding only half as much power as the previous bit, the effect is less, but each position contributes, as shown in Figure 8.3.

**TRY
IT**

8.4 Dim Bulb. What is the binary setting needed to show a red subpixel at $1/4$ of its brightness, as shown here?



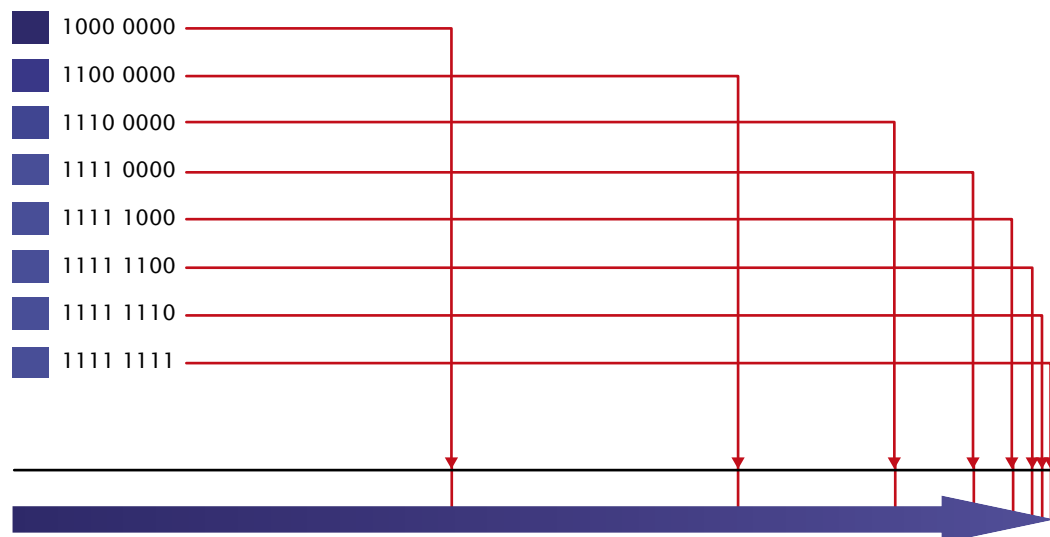


Figure 8.3 Progressively increasing the intensity for a blue subpixel; each bit contributes half as much power as the bit to its left.

Black and White Colors

The intensity of the R, G, and B lights just discussed are the binary numbers stored in a byte: 8 bits. So, representing the color of a single pixel requires 3 bytes. Using binary numbers means that the smallest intensity is 0000 0000, which is 0, of course, and the largest value is 1111 1111. To figure out what decimal number that is (and to refresh ourselves on binary numbers from Chapter 7), we add the place values for the 1's,

$$\begin{aligned}
 1111\ 1111 &= 2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0 \\
 &= 128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 \\
 &= 255
 \end{aligned}$$

which tells us that the range of values is 0 through 255 for each color.

As stated above, black is the absence of light,

0000 0000 0000 0000 0000 0000 *RGB bit assignment for black*
 red green blue
 byte byte byte

while white

1111 1111 1111 1111 1111 1111 *RGB bit assignment for white*
 red green blue
 byte byte byte

has full intensity for each color. Between these extremes is a whole range of intensity.

FLUENCY BIT

Super Powers. Powers of two, $2^n = 2 \times 2 \times \dots \times 2$, are common in computing because of binary. For that reason, computer people quickly learn their powers.

2^0	1	2^4	16	2^8	256	2^{12}	4,096	2^{16}	65,536	2^{20}	1,048,576
2^1	2	2^5	32	2^9	512	2^{13}	8,192	2^{17}	131,072	2^{21}	2,097,152
2^2	4	2^6	64	2^{10}	1,024	2^{14}	16,384	2^{18}	262,144	2^{22}	4,194,304
2^3	8	2^7	128	2^{11}	2,048	2^{15}	32,768	2^{19}	524,288	2^{23}	8,388,608

Decimal to Binary

So far, we have converted binary to decimal. The algorithm for converting decimal to binary is the opposite, of course. Basically, it “finds which powers of 2 combine to make the decimal number.” The following table clarifies this:

Number being converted										
Place value	512	256	128	64	32	16	8	4	2	1
Subtract										
Binary Number										

We’ll discuss the algorithm in a moment, but first, to build the table we fill the second row with powers of 2. But, how many powers of 2? *All of the powers of 2 less than the number we want to convert.* So, the table shown above works for any decimal number up to 1,023, because the next power of 2 is 1,024. You’ll see why this is the rule as soon as we’ve done our first conversion.



8.5 Very Powerful. To convert decimal 206 to binary, which powers of 2 will be needed for the table?

The algorithm begins by placing the number to be converted—we’ll illustrate with 365—into the top-left square, shown in green.

Number being converted	365									
Place value	256	128	64	32	16	8	4	2	1	
Subtract										
Binary Number										

The algorithm has two cases based on a comparison of the number being converted and the place value.

- 1. **If the place value is smaller:** Subtract. Enter a 1 in the binary number row for that column (because that place contributes to the binary form of the number), and move the remainder (the result of the subtraction) to the top of the next column.
- 2. **If the place value is larger:** Don’t subtract. Enter a 0 in the binary number row for that column and move the number to be converted right one square.

Continue with the columns in order until all of the binary digits have been created. The first two steps of the conversion of 365 to binary illustrate the two cases of the algorithm.

Number being converted	365	109	109						
Place value	256	128	64	32	16	8	4	2	1
<i>Subtract</i>	109								
Binary Number	1	0							

Starting out, the place value (256) is less than 365, so the first case applies; we subtract and enter a 1 into the binary number row. The resulting 109 goes to the top of the next column. But now, the place value is larger than the number 109 to be converted (128 is larger than 109), so the second case applies. We enter a 0 in the developing binary number, and shift the 109 right one position. The full computation produces 1 0110 1101.

Number being converted	365	109	109	45	13	13	5	1	1
Place value	256	128	64	32	16	8	4	2	1
<i>Subtract</i>	109		45	13		5	1		0
Binary Number	1	0	1	1	0	1	1	0	1

So, why, when we set up the table, do we fill in all powers of 2 less than the number to be converted? Because the binary number will be a combination of the powers of 2 that add up to the decimal number—that's what it means to represent a number in binary—and so we must check them all to see if they contribute to the number.




8.6 Decimal Conversion. Convert the decimal number 206 to binary using the algorithm and its table.

Lighten Up: Changing Colors by Addition

Returning to our discussion of color representation, the extreme colors of black and white are easy, but what color does the following represent?

1100 1110 1100 1110 1100 1110

red	green	blue
byte	byte	byte

First, notice that each byte contains the decimal value 206, which you recognize from the Try It 8.6 conversion. So, the mystery color is the color produced by the specification RGB (206,206,206). Like black and white, the mystery color has equal amounts of red, green, and blue, and it is closer to white than black. In fact, it is a light gray . All colors with equal intensities of the RGB subpixels are a shade of gray if they are not black or white. It's just a question of whether they're closer to black or white.

1100 1110	binary representing decimal number	206
+ 1 0000	binary representing decimal number	16
1101 1110	binary representing decimal number	222

Figure 8.4 Adding 16 to a binary value to lighten RGB intensities.

To Increase Intensity: Add in Binary


To make a *lighter* color of gray, obviously we change the common value to be closer to 255. Suppose we do this by increasing each of the RGB values by 16—that is, by adding 16 to each byte—as shown in Figure 8.4.

The result in Figure 8.4 is found by simply setting the 16’s place value—that is, changing it from 0 to 1—because as a power of 2, it now contributes to the intensity. When the increase is applied to each color, the result is

1101 1110 1101 1110 1101 1110

red green blue

byte byte byte

which is a lighter shade of gray . By the way, we can add or subtract any amount as long as the result is not less than 0 or greater than 255.

Lighter Still: Adding with Carry Digits

Imagine that we want the color lighter still by another 16 units of intensity for each RGB byte. Adding another 16 isn’t quite as easy this time. The 16’s position in the binary representation 1101 1110 of decimal 222 is already filled with a 1. So, we “carry” to the next higher place (see Figure 8.5).

The result gives color intensities of

1110 1110 1110 1110 1110 1110

red green blue

byte byte byte

Notice that if we’d simply added 32 to 206 originally, we’d have ended up with the same result—the gray with each intensity set at 238.

We just illustrated the binary addition process. As with other aspects of binary, binary addition is actually the same as decimal addition, except with two digits only. We work from right to left, adding corresponding digits in each place position and writing the sum below. Like decimal addition, there are two cases. Sometimes, we can add the two numbers in a place and the result is expressed as a single digit. That was the case the first time we added 16 to the RGB byte: We added 1 + 0 in the 16’s place and the result was 1.

1	carry digit	
1101 1110	binary representing decimal number	222
+ 1 0000	binary representing decimal number	16
1110 1110	binary representing decimal number	238

Figure 8.5 Adding 16 to the binary representation of the intensity 222, requiring a carry.