Image Compression: Basics

JPEG Basics

JPEG is a *lossy* image compression algorithm that typically achieves 10:1 compression with little perceptible loss in quality.

Created by the Joint Photographic Experts Group in 1992.

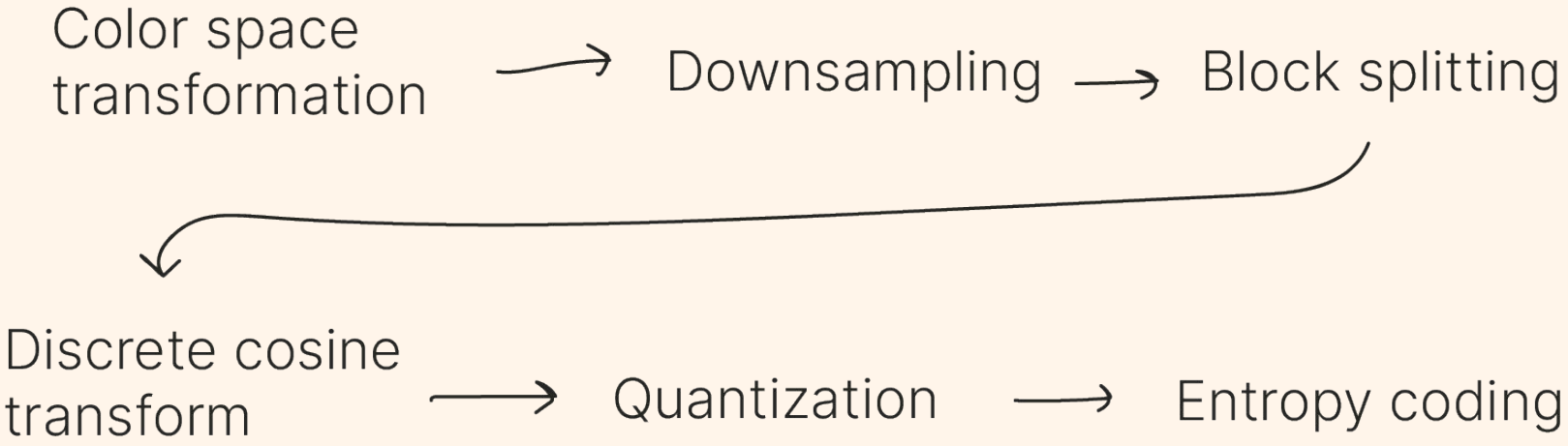
JPEG was largely responsible for the proliferation of digital images and digital photos across the Internet, and later social media.

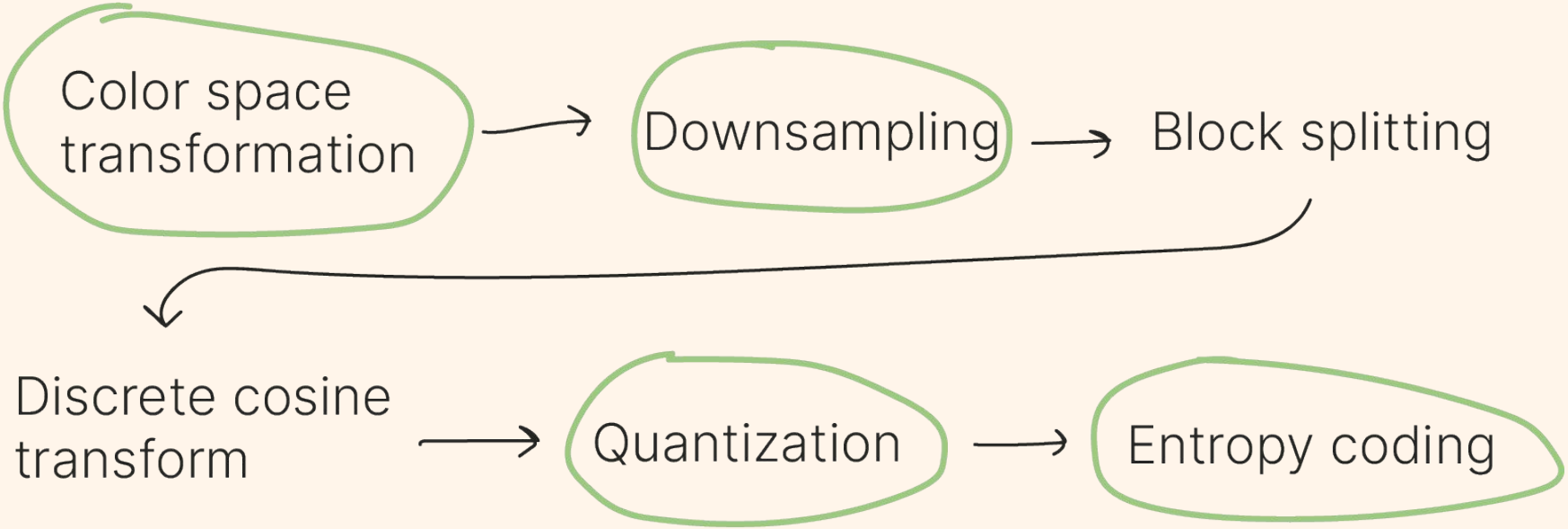
JPEG Basics

JPEG is a nice case study for learning many image processing and more general data compression techniques.

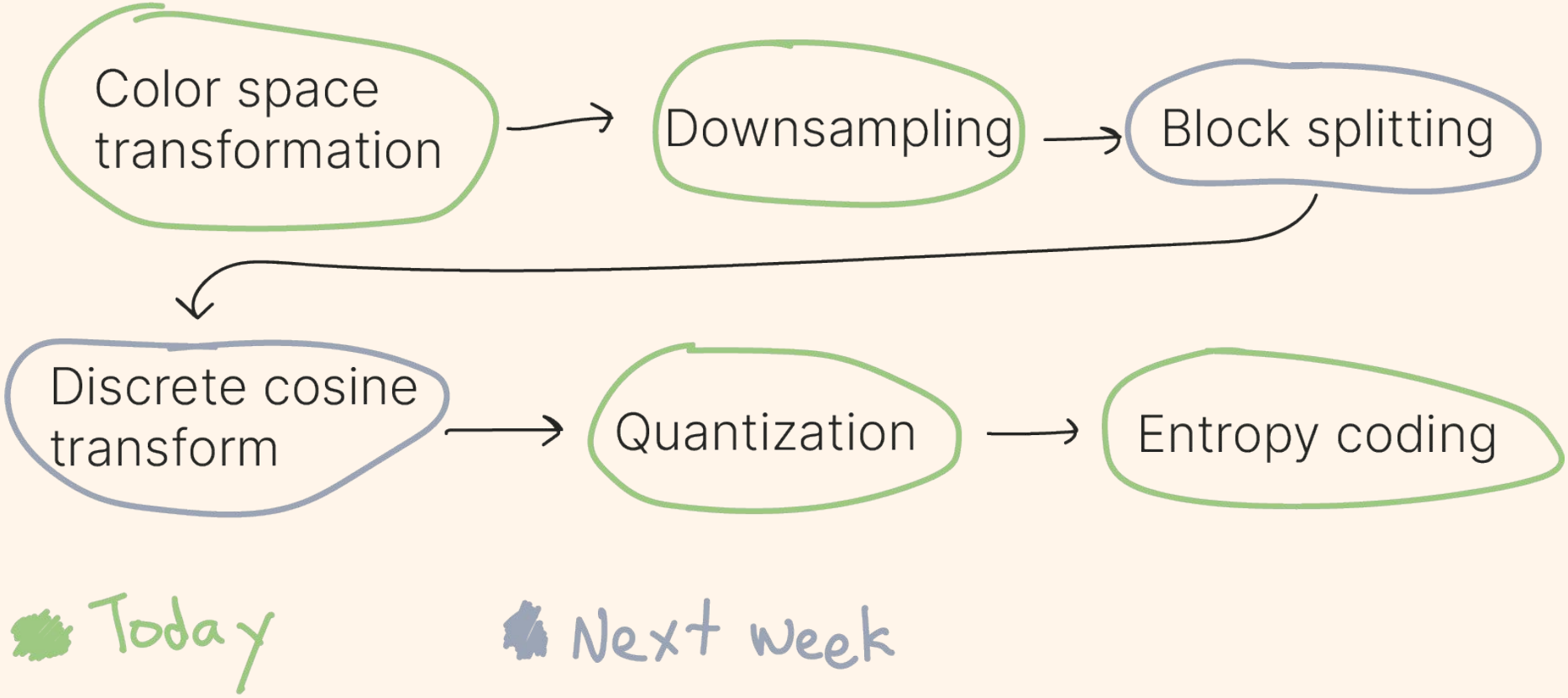
It takes advantage of human color and image perception to reduce image quality in imperceptible ways.

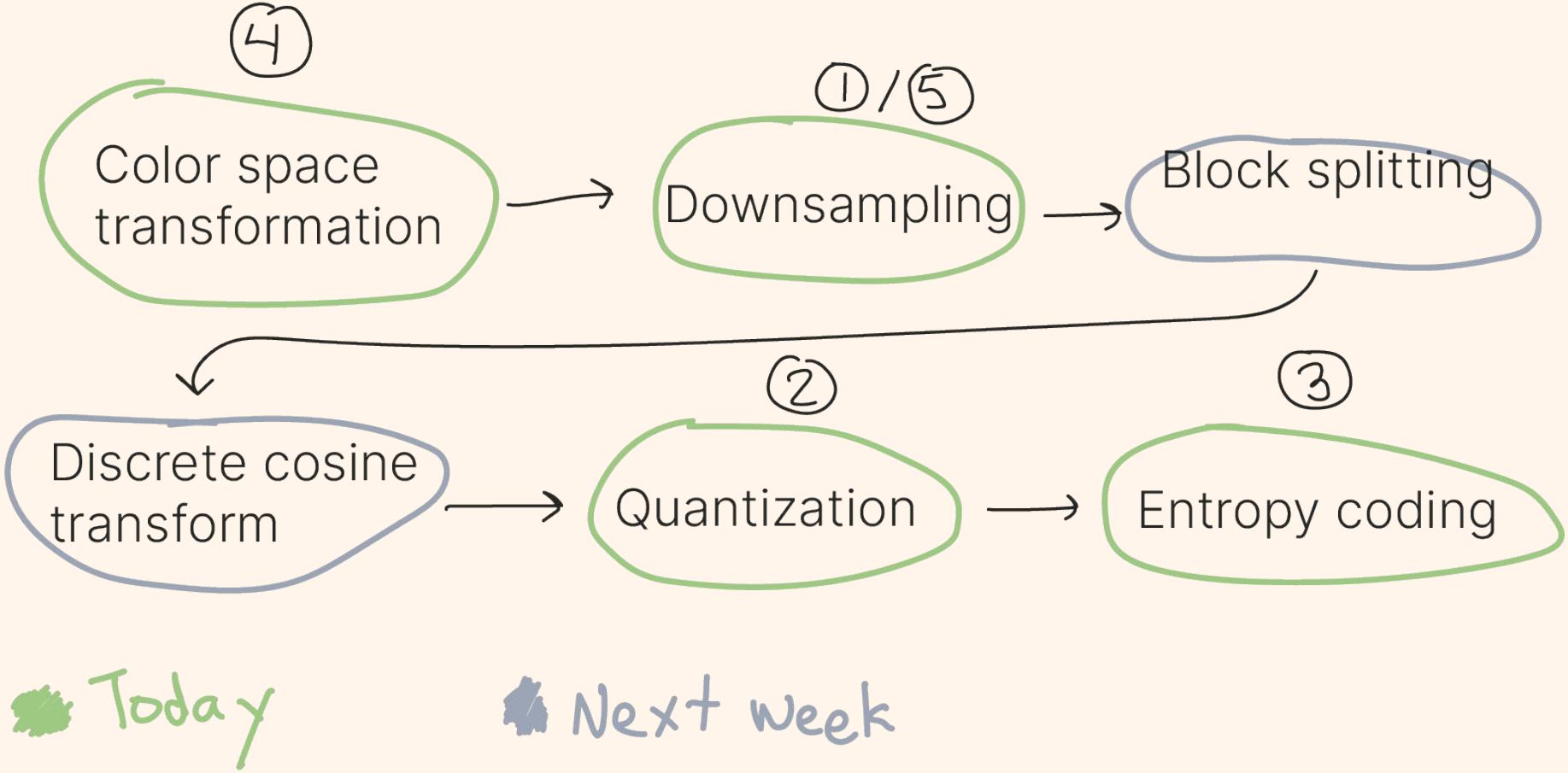
The JPEG Algorithm (JFIF specifically)




Color space transformation
Downsampling
Block splitting
Discrete cosine transform
Quantization
Entropy coding

Today


Color space transformation
Downsampling
Block splitting
Discrete cosine transform
Quantization
Entropy coding
Today
Next week


4
Color space transformation
1/5
Downsampling
Block splitting
Discrete cosine transform
2
Quantization
3
Entropy coding
Today
Next week

Think stupid:
How can you make an image smaller?

Bird brain tactics

- Just make the image resolution smaller
  - Called downsampling in signal processing
  - Specifically, we can find the average color of a block of pixels to represent the group.

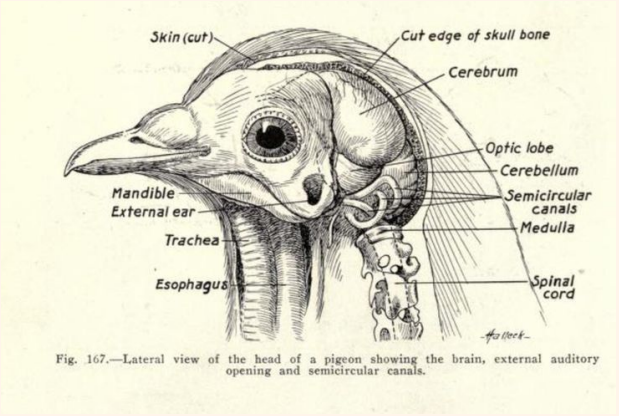
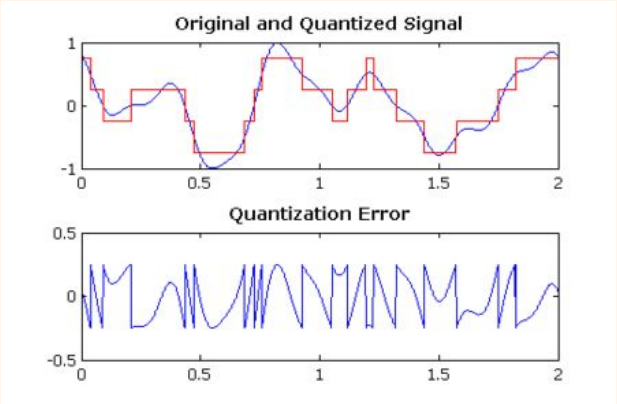


Fig. 167.—Lateral view of the head of a pigeon showing the brain, external auditory opening and semicircular canals.

Bird brain tactics

- Just make the image resolution smaller
  - Called downsampling in signal processing.
  - Specifically, we can find the average color of a block of pixels to represent the group.
- Reduce the number of bits to represent a color
  - Called quantization in signal processing.



A note on color

Typically, JPEG images store RGB images.

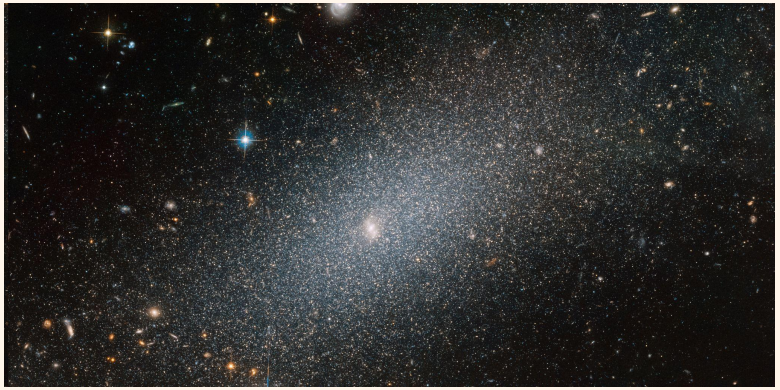
Most general-purpose viewing applications use 8 bits per R/G/B channel, so 24 bits per color.

What if we used 7 bits per channel (21 bits/pixel)?

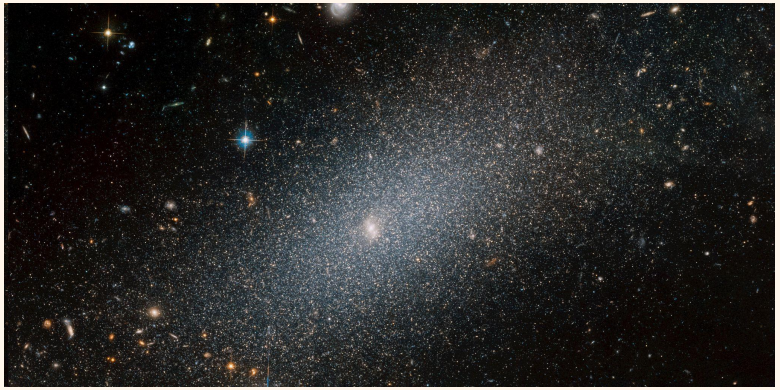
What if we used 4 bits per channel (12 bits/pixel)?

Exercise 1

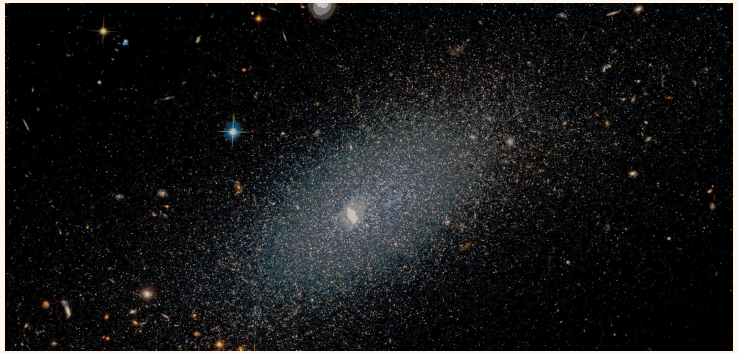
Write an image compressor that quantizes the pixel values



7 bits/channel



4 bits/channel



2 bits/channel

Discussion

Keep in mind that every time we remove a bit, the total number of values the binary number can have is reduced by two.

We can get down to ~4 bits/channel before the image starts to deteriorate rapidly, which gives us a 2:1 compression ratio.

But JPEG is still 10:1, so we still have a long ways to go!

Color perception

We can try to be smarter about what data we remove by thinking about how we perceive color.

In general, the eye is more sensitive to changes in luminance (perceived brightness) than chromaticity

OKHSV (300, 30, 60) (336, 30, 60)

OKHSV (300, 30, 60) (300, 30, 70)

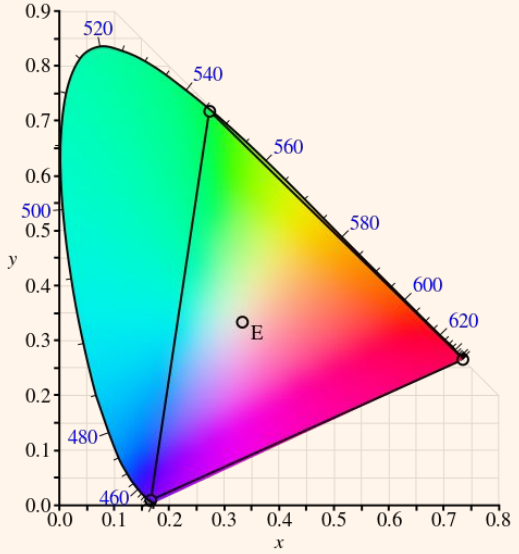
Color history

In the old days, scientists wanted to study color and how we perceived it but needed a set of standards to describe it.

The International Commission on Illumination (or CIE for its French name, Commission internationale de l'éclairage) was established in 1913 in Vienna, Austria.

The CIE ran experiments to characterize human perception of color and defined a few standard color spaces, as well as some standard illuminants.

More on this later.



Luminance

Importantly, the CIE described the perceived brightness of a color in terms of RGB as a part of the CIE RGB → CIE XYZ color space transformation, where Y is the relative luminance:

$$Y = 0.2126 \cdot R + 0.7152 \cdot G + 0.0722 \cdot B$$

Luminance

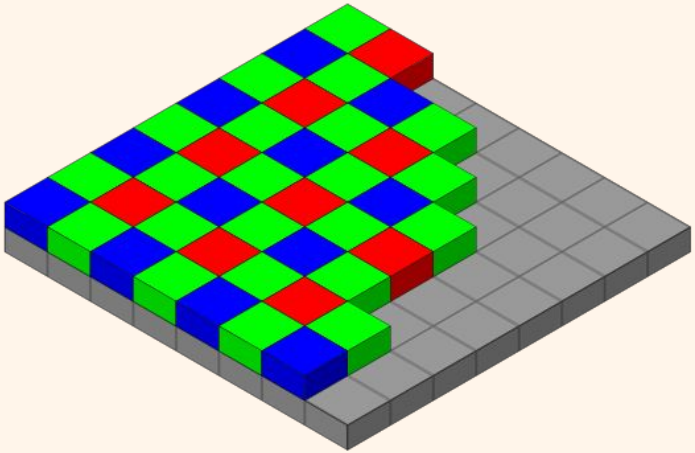
Importantly, the CIE described the perceived brightness of a color in terms of RGB as a part of the CIE RGB → CIE XYZ color space transformation, where Y is the relative luminance:

$$Y = 0.2126 \cdot R + 0.7152 \cdot G + 0.0722 \cdot B$$

Interestingly, we see that the luminance of a color depends mostly on the intensity of the green component.

The importance of green

The fact that green is the main driver of luminance (and luminance is a big driver of image quality) is the reason why cameras have 2x the amount of green in their Bayer filter mosaic.

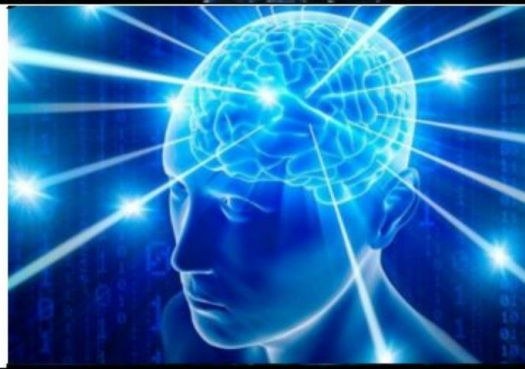


How can we use this information to improve our compressor?

We could quantize R/B more than G

We could downsample R/B (4:1 reduction assuming 2:1 per axis)

Color convert to some kind of Y[XX] color space that encodes luminance directly and downsample the other two channels



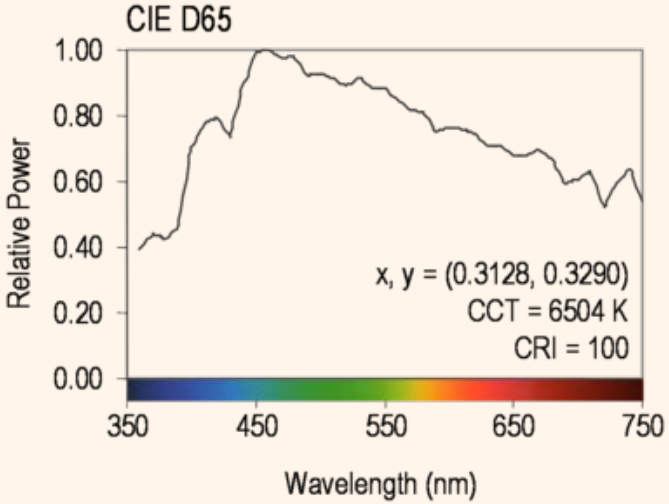
So we want a colorspace with a luminance (Y) component, what else?

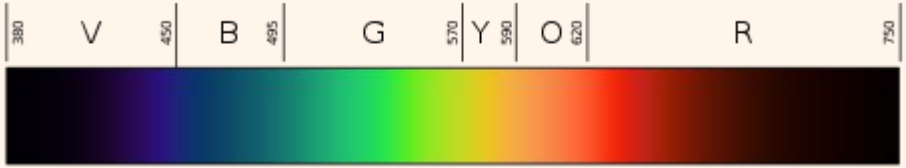
Background on color

Light is most accurately represented by a graph of intensity versus wavelength.

This is the CIE's spectrum for white! →

Other colors like pink, purple, or magenta also aren't found on the visible spectrum but instead can be produced from a combination of wavelengths.





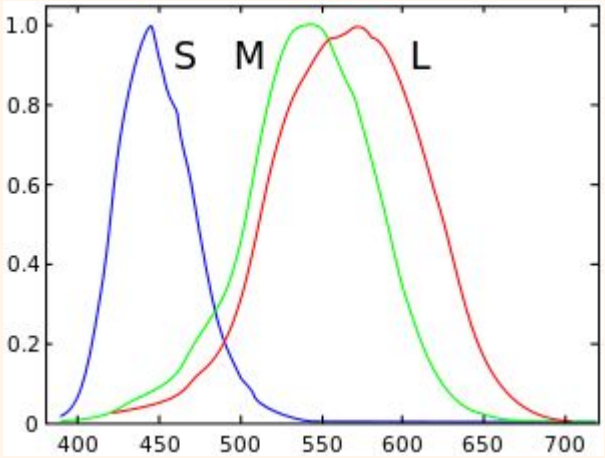
acm

Background on color spaces

In 1931 the CIE introduced a bunch of colorspaces

We start with the “LMS” color space (although not technically a CIE color space), which has one channel for each of the three types of cone cells.

This is one of the reasons why most color spaces have three channels.

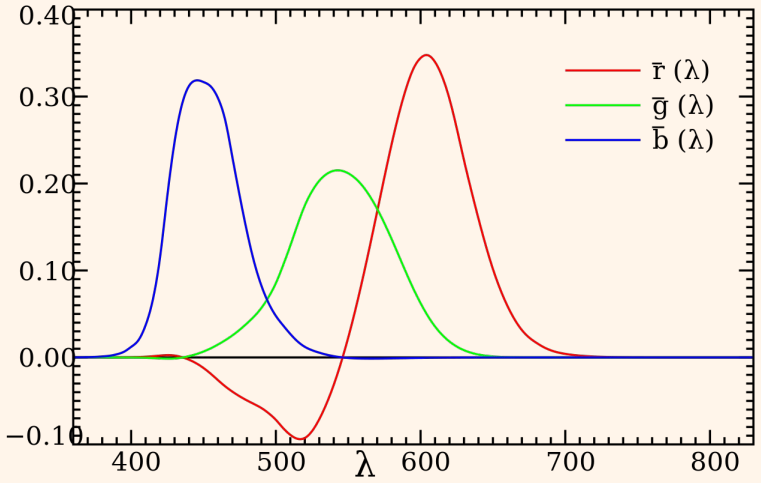


Background on color spaces

In the 1920s, W. David Wright and John Guild independently ran experiments which laid the foundation for the CIE RGB color space.

In the experiments, participants would attempt to recreate a “test” color using the given fixed (RGB) primary colors.

The graph to the right shows the intensity of each primary color required to produce a test color of a specified wavelength.



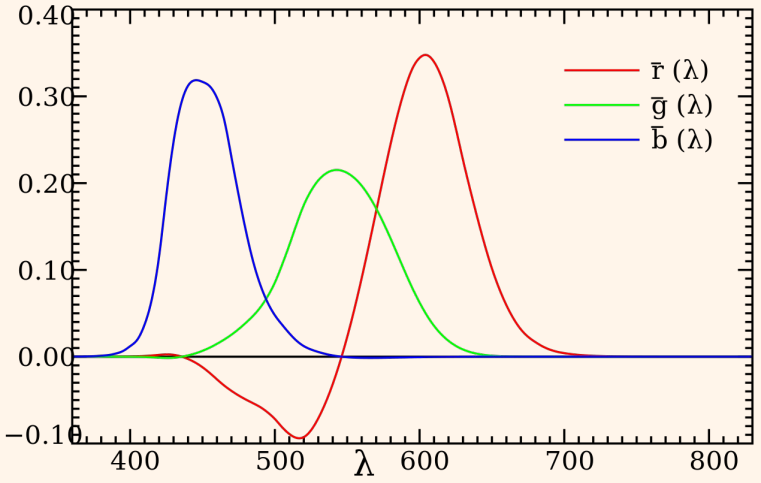
Background on color spaces

We can finally use these color matching functions to represent a spectral power distribution ($S(\lambda)$) using RGB.

$$R = \int_0^\infty S(\lambda)\,\bar{r}(\lambda)\,d\lambda,$$

$$G = \int_0^\infty S(\lambda)\,\bar{g}(\lambda)\,d\lambda,$$

$$B = \int_0^\infty S(\lambda)\,\bar{b}(\lambda)\,d\lambda.$$



Background on color spaces

But, this wasn't good enough for the CIE. They wanted a color space with a few desirable properties that the RGB color space didn't satisfy.

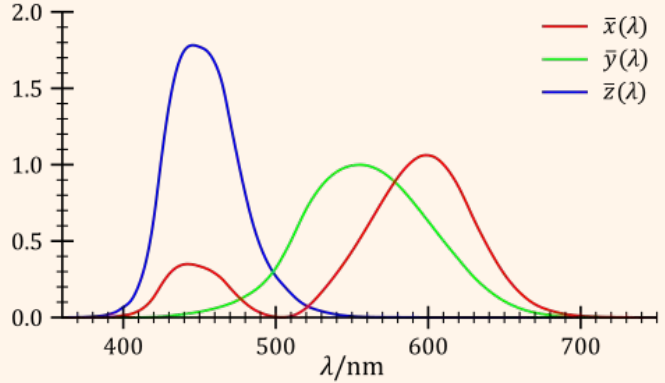
1. Everywhere greater than 0
2. $\bar{y}(\lambda)$ is the photopic luminous efficiency function
3. At the constant energy white point, $X=Y=Z=⅓$

As an effect of the above, XZ (the chromaticity components) would also conveniently contain all visible colors inside [1, 0], [0, 0], [0, 1].

Also, $\bar{z}(\lambda)$ would be 0 above 650 nm since they found out they could.

Background on color spaces

Based on these specifications, the CIE also introduced the XYZ color space:



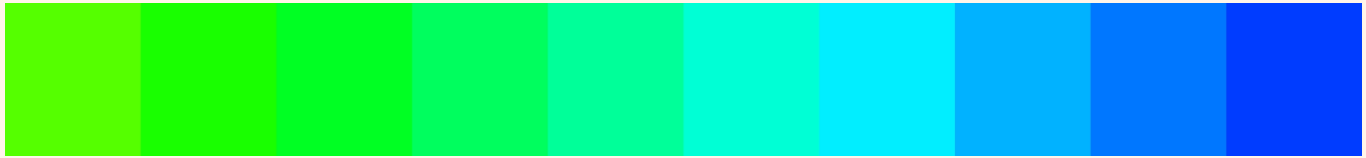
$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \frac{1}{b_{21}} \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

$$= \frac{1}{0.176\,97} \begin{bmatrix} 0.490\,00 & 0.310\,00 & 0.200\,00 \\ 0.176\,97 & 0.812\,40 & 0.010\,63 \\ 0.000\,00 & 0.010\,00 & 0.990\,00 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Background on (perceptual) color spaces

XYZ is good, but it’s not perceptual. This means that color changes perceptually uniformly as it’s value changes.

So a gradient like this:



Is not as perceptually uniform as a gradient like this:

Background on (perceptual) color spaces

Perceptual color spaces are important for graphic designers when picking colors or animating color transitions.

They can also be important for image compression, so value density isn't wasted at similar looking colors.

https://bottosson.github.io/posts/oklab/

acm

So, we want a colorspace that:

1. Has a luminance (Y) component, and
2. Is perceptual

YUV/YCbCr color spaces

Luckily, this isn’t a new problem.

When transmitting TV signals, engineers needed a way to add on top of grayscale transmissions to get color TV.

They decided to add two perceptually uniform channels, UV, to shift the grayscale color.

This created a new class of color spaces that still persists today in TV, film production, and other applications (like compression).

YUV/YCbCr color spaces

ITU-R BT.470: Analog TV (old)

ITU-R BT.601: SDTV (1982)

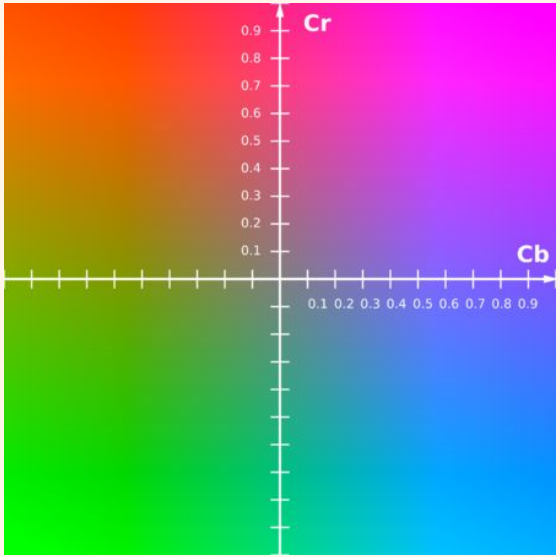
ITU-R BT.709: HDTV (1993)

ITU-R BT.2020: UHDTV with SDR, WCG (2012)

ITU-R BT.2100: HDR-TV (2016)

JFIF usage of JPEG supports a modified Rec. 601 Y′CbCr where Y′, Cb and Cr have the full 8-bit range of [0...255].

$$
\begin{aligned}
Y' &= 0 + (0.299 \cdot R'_D) + (0.587 \cdot G'_D) + (0.114 \cdot B'_D) \\
C_B &= 128 - (0.168736 \cdot R'_D) - (0.331264 \cdot G'_D) + (0.5 \cdot B'_D) \\
C_R &= 128 + (0.5 \cdot R'_D) - (0.418688 \cdot G'_D) - (0.081312 \cdot B'_D)
\end{aligned}
$$



Film nerds, check out yedlin.net!

acm

Gamma correction (Y' !=Y, sRGB!=RGB)

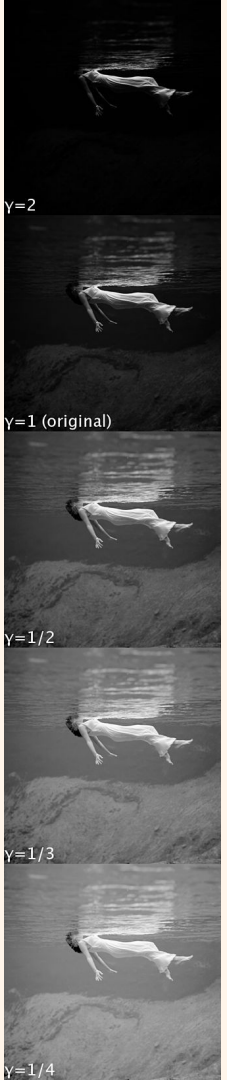
Human perception of brightness, under common illumination conditions, follows an approximate power function.

AKA we can see small differences in dark images better than small differences in bright images.

Therefore, it makes more sense to put less value density at brighter values using a power-law expression:

$$V_{\text{out}} = AV_{\text{in}}^{\gamma}$$

A gamma value < 1 is sometimes called an encoding gamma; conversely a gamma value >1 is called a decoding gamma.



Exercise 2

Write an image compressor that color converts from RGB to R'G'B' (γ=2.2) to Y'CbCr and downsamples the chrominance channels by a ratio of 4:2:0.

$$
\begin{aligned}
Y' &= 0 + (0.299 \cdot R'_D) + (0.587 \cdot G'_D) + (0.114 \cdot B'_D) \\
C_B &= 128 - (0.168736 \cdot R'_D) - (0.331264 \cdot G'_D) + (0.5 \cdot B'_D) \\
C_R &= 128 + (0.5 \cdot R'_D) - (0.418688 \cdot G'_D) - (0.081312 \cdot B'_D)
\end{aligned}
$$

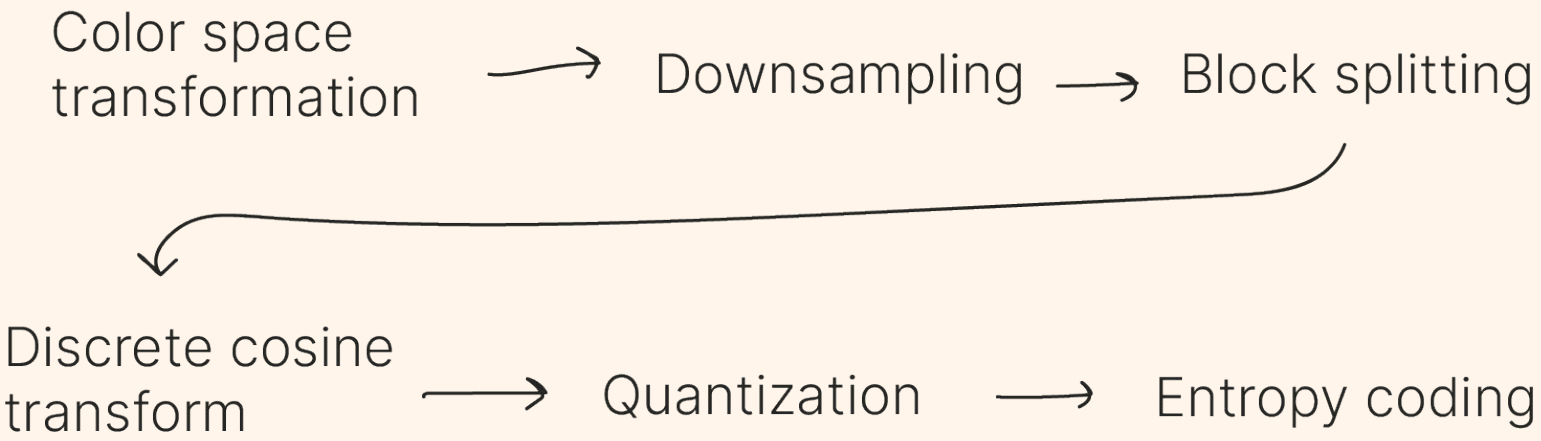
Discussion

Should have ~3:1 compression ratio, better but still a ways to go.

Next, we'll get into a more generic data compression technique, entropy coding.

Entropy coding is the last part of the JFIF compression algorithm.

It consists of two general data encoding stages: **run-length encoding** and **huffman encoding.**



Run-length encoding

RLE happens after the DCT quantization stage, which can produce a large number of zeros.

RLE is a very simple compression algorithm that simply counts consecutive zeros and replaces them with the number of zeros.

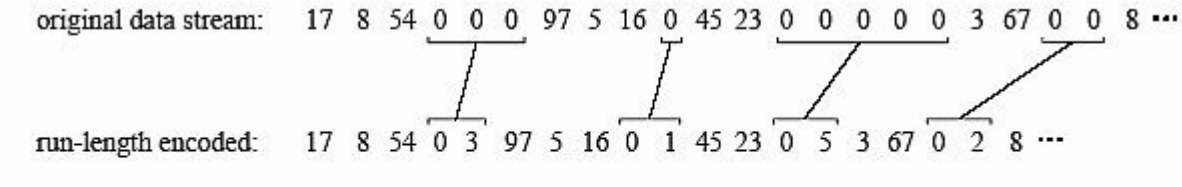


FIGURE 27-1
Example of run-length encoding. Each run of zeros is replaced by two characters in the compressed file: a zero to indicate that compression is occurring, followed by the number of zeros in the run.

Run-length encoding

RLE might seem kind of stupid, but it can be more powerful than you think.

For example, let's say you're building an undo system that stores changes to a massive set of data.

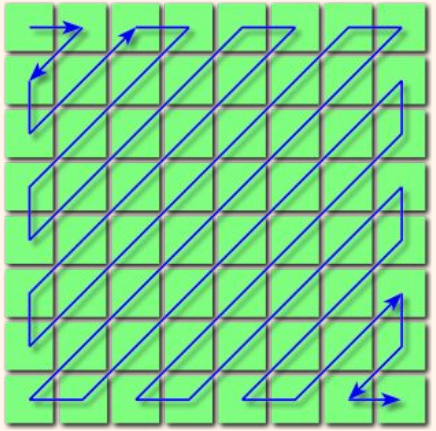
Instead of storing every state, you can simply store the delta between two states with XOR. Differing bits will be 1 and unchanged bits will be zero.

Then we can use RLE to compress the zeros:

```
procedure handle_edit(old_state, new_state, undo_stack):
    delta = old_state ⊕ new_state
    compressed = RLE(delta)
    undo_stack.push(compressed)
```

JFIF RLE is done in a zig-zag pattern over each block →

So a quantized DCT block might be traversed like this:



$$B = \begin{bmatrix} -26 & -3 & -6 & 2 & 2 & -1 & 0 & 0 \\ 0 & -2 & -4 & 1 & 1 & 0 & 0 & 0 \\ -3 & 1 & 5 & -1 & -1 & 0 & 0 & 0 \\ -3 & 1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

```
-26
 -3   0
 -3  -2  -6
  2  -4   1  -3
  1   1   5   1   2
 -1   1  -1   2   0   0
  0   0   0  -1  -1   0   0
  0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0
  0   0   0   0   0   0
  0   0   0   0   0
  0   0   0   0
  0   0   0
  0   0
  0
```

And encoded like this:

Symbol 1	Symbol 2
(RUNLENGTH, SIZE)	(AMPLITUDE)

- x is the non-zero, quantized AC coefficient.
- RUNLENGTH is the number of zeroes that came before this non-zero AC coefficient.
- SIZE is the number of bits required to represent x.
- AMPLITUDE is the bit-representation of x

```
-26
 -3    0
 -3   -2   -6
  2   -4    1   -3
  1    1    5    1    2
 -1    1   -1    2    0    0
  0    0    0   -1   -1    0    0
  0    0    0    0    0    0    0    0
  0    0    0    0    0    0    0
  0    0    0    0    0    0
  0    0    0    0    0
  0    0    0    0
  0    0    0
  0    0
  0
```

```
(0, 2)(-3);(1, 2)(-3);(0, 1)(-2);(0, 2)(-6);
(0, 1)(2);(0, 1)(-4);(0, 1)(1);(0, 2)(-3);(0, 1)(1);
(0, 1)(1);(0, 2)(5);(0, 1)(1);(0, 1)(2);(0, 1)(-1);
(0, 1)(1);(0, 1)(-1);(0, 1)(2);(5, 1)(-1);
(0, 1)(-1);(0, 0);
```