Three algorithms for converting color to grayscale

Posted on **24 August 2009** by **John**

How do you convert a color image to grayscale? If each color pixel is described by a triple (R, G, B) of intensities for red, green, and blue, how do you map that to a single number giving a grayscale value? The GIMP image software has three algorithms.

The **lightness** method averages the most prominent and least prominent colors: (max(R, G, B) + min(R, G, B)) / 2.

The **average** method simply averages the values: (R + G + B) / 3.

The **luminosity** method is a more sophisticated version of the average method. It also averages the values, but it forms a weighted average to account for human perception. We're more sensitive to green than other colors, so green is weighted most heavily. The formula for luminosity is 0.21 R + 0.72 G + 0.07 B.

The example sunflower images below come from the GIMP documentation.

Original image

Lightness

Average

Luminosity

The lightness method tends to reduce contrast. The luminosity method works best overall and is the default method used if you ask GIMP to change an image from RGB to grayscale from the Image -> Mode menu. However, some images look better using one of the other algorithms. And sometimes the three methods produce very similar results.

Update: See More on colors and grayscale for more details and more examples.

Categories : Graphics

Tags : Graphics

Bookmark the permalink

Previous Post
Back to school

Next Post
More on colors and grayscale

32 thoughts on “Three algorithms for converting color to grayscale”

Colin Howe

24 August 2009 at 09:19

Always worth just seeing if each channel on it's own produces good results too! Can also get some really interesting shots when just looking at one channel 🙂

AdamJTP

24 August 2009 at 09:30

And if (0.21 R + 0.71 G + 0.07 B) seem like magic numbers then here are some more magic numbers: http://jscience.org/experimental/javadoc/org/jscience/computing/ai/vision/GreyscaleFilter.html

I've heard that some portrait photographers use orange filters with black and white film (to remove freckles and spots).

Keith

24 August 2009 at 20:15

Perhaps this is obvious to others, but what algorithm does actual film "use" to convert colors? An old-school analog (i.e. not digital) camera changes real world colors into b&w photographs, but using which of these methods, if any? I'm guessing it may vary with camera and film

settings, and likely has changed over time, but is there one method it tends to favor over the others?

John

24 August 2009 at 20:20

@Keith: That's a good question. I'd like to know the answer.

PossiblyKevin

25 August 2009 at 02:29

the answer is none of the above methods are used for film. in film or any system sensitive to the electromagnetic spectrum, can be considered as a reduction of an infinite number of frequencies (well infinite depending on your quantum view of the world I guess), into a smaller number, in a similar manner to a weighted sum histogram.

In film or a digital sensor there is a spectral sensitivity function that determine which frequencies of light get absorbed/sensitise the emulsion based upon the chemical structure. Through various means this gets converted to either silver halide crystals or in colour film different dye's are formed (eventually).

You then shine a light through the resultant film and bounce it off a screen, your eyes then view the result, In mathematical terms this is a highly non-linear transformation from scene-to-screen and is a function

of many reductions from spectral to n-dimensional ‘records’ all of whose intensity functions are non-linear.

In film the actual ‘colours’ vary mostly due to the stocks used, but there are some effects that vary with intensity of exposure so the camera settings can effect the results though this is minor compared to the stock and development.

Re: the magic numbers, well they are based upon a mixing assumption. They are are saying what linear combination of R, G and B would you perceive as being without colour, i.e. neutral. Depending on the exact colours of R, G and B you will get different proportions. The mixing also assumes that the individual colour channels are linear, else your not really that close to how our eyes behave. What you consider neutral is also heavily effected by the surroundings in which you view, your eyes are able to adjust their own gain functions to compensate for a certain amount of change, thus a piece of paper can look white to you even if the colour of the light shining on it is changed, although with highly coloured light you will still see it as being coloured.

Pingback: igorbrejc.net » Fresh Catch For August 29th

Pingback: ant.simianzombie.com » Scrollbars and Greyscale Algorithms

Pingback: Paweł Głowacki : Boian’s TBitmap Visualizer and converting to grayscale

Pingback: Paweł Głowacki : Converting to grayscale with TBitmap.ScanLine property

CreativeNotice

7 October 2010 at 15:04

Thanks for the post. Was exactly what I was looking for.

Mark Ransom

29 October 2010 at 11:37

The “magic numbers” for luminosity are slightly rounded versions of the ones in BT.709 for HDTV:
http://en.wikipedia.org/wiki/YUV#BT.709_and_BT.601

There appears to be a bug, the weights should add up to 1.0. I think the multiplier for G should be 0.72 rather than 0.71.

As for how B/W film converts to gray, it appears to depend on the exact film chemistry:
http://en.wikipedia.org/wiki/Photographic_film#Spectral_sensitivity

Matej

21 April 2011 at 02:55

I did some testing with all three mentioned methods and discovered that Luminosity method is best if picture is not too blue. So I tried this:

1. for every pixel I first check if blue component is greater than green and red

2. if it IS I use Lightness method (or Average – both are good) … if NOT I use Luminosity method for current pixel.

This algorithm gives very good results. It comes very close to Photoshop's convert to Grayscale "mystic" algorithm 🙂

Don Stewart

16 May 2011 at 13:52

I converted this nice example to Haskell as part of an multidimensional array programming tutorial. The result is pleasingly concise (and parallel).

http://www.haskell.org/haskellwiki/Numeric_Haskell:_A_Repa_Tutorial#Example:_parallel_image_desaturation

Weight Conversion

1 July 2011 at 23:21

n film or a digital sensor there is a spectral sensitivity function that determine which frequencies of light get absorbed.

_____

Allen

sham

17 October 2011 at 08:27

sir, thanks a lot.wish u all the beat for your future work and hope u will always help us by providing such data.

Pingback: Wineapp – creating a high-tech hash: gray scaling « Mobile apps

Pingback: » Java color image to grayscale conversion algorithm(s) ZeroCool

Pingback: gattя

Tarek

18 June 2012 at 19:16

thank you. nice thing on image. love it!

jsding

18 February 2013 at 19:04

I just use the average method to grayscale a button (disable status) in a HTML5 canvas project

```
var context = canvasEl.getContext('2d'),
imageData = context.getImageData(0, 0, canvasEl.width,
canvasEl.height),
data = imageData.data,
iLen = imageData.width,
jLen = imageData.height,
index, average, i, j;

for (i = 0; i < iLen; i++) {
for (j = 0; j < jLen; j++) {

index = (i * 4) * jLen + (j * 4);
average = (data[index] + data[index + 1] + data[index + 2]) / 3;

data[index] = average;
data[index + 1] = average;
data[index + 2] = average;
}
}
```

```
context.putImageData(imageData, 0, 0);
```

Thanks for the post.

Pingback: First steps in Freemat | Uskerine's blog

Pingback: More on colors and grayscale | John D. Cook

jay

23 January 2015 at 20:27

what would be the time complexity of RGB average method?

Pingback: Bookmarks for February 4th | Chris's Digital Detritus

Pingback: Fun and try: first gray scale | +.゚(*-ω-)人(-ω-*)+.゚

Ali

10 August 2015 at 09:13

I've recently built http://colorconversion.co/ website for converting rgb, hex, hsl, hsb, hsv and cmyk using a single input box. It is free and fun to use 🙂

Pingback: Nicholas Higham on Mathematics in Color

Me

4 October 2015 at 14:52

Color conversion also showing the luminosity is available at http://www.workwithcolor.com/color-converter-01.htm

Pingback: Tratamiento de imágenes con canvas (HTML5) - David Ruiz Osés

Philip Rynes

18 April 2016 at 02:41

About 20 years ago I had to read a 640 x 480 px image from a RGB frame buffer in an IBM-6000 workstation with AIX Unix and the Motif Window Manager on top of X-windows, and send it to a LaserJet printer that did not have PostScript and which only had dots, black or white for images.

I used the luminosity method to get a gray value for each pixel, then the Floyd – Steinberg dithering algorithm. Since the colors on paper were either 1 or 0, the Floyd – Steinberg dithering made a big difference to the outcome in making the image look like shades of gray.

The algorithm achieves dithering using error diffusion, meaning it pushes (adds) the residual quantization error of a pixel onto its neighboring pixels, to be dealt with later.

The 'dealt with later' refers to later in the Floyd – Steinberg process. In the final step, if the pixel value was below 0.5 it was set to 1; otherwise it was set to 0.

The results: Images looked extremely similar to Images produced by PostScript on the same printer.

Ross

18 October 2016 at 09:09

I have images that I need to analyze. I intend to apply an OCR or text reader to the image. I will also be doing image comparisons to detect the degree of changes. These images are not random. They are usually have small color variation and have high contrasts and sometimes text. The image might include a company logo, but nothing more complex.

I saw that weighted averages of RGB color are applied sometimes in converting to grey tone. I was thinking that there may be a specific optimal weight average I could apply when transforming the image to a grey tone that would assist my analysis. My only thought is to write a test program that plods through a combination of weights to find what works best.

I welcome any thoughts on alternative approached or articles that may help. I might try that luminosity / Floyd – Steinberg dithering approach; sounds interesting

Pingback: Manipulación de imágenes con Javascript. Parte 1 | EtnasSoft

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Notify me of followup comments via e-mail

Name *

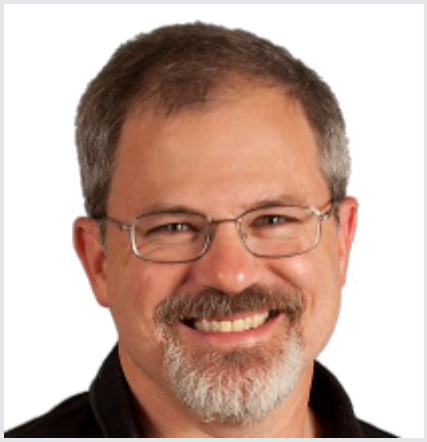
Email *

Website

Post Comment

Search ...

Search



Work together

Subscribe to my newsletter

Subscribe

Latest Posts

Longhorn tribute to fallen Aggies

Categories

Select Category

Archives

Select Month

Tags

Acoustics Bayesian Biostatistics Books C++ Cancer Category theory Clinical trials Creativity Differential equations Economics Education Emacs Functional programming Graphics Haskell History HTML Integration Interview LaTeX Math Music Networks Number theory PowerShell Probability and Statistics Productivity Programming Python Quality Quotes Regular expressions Reproducibility Rstats Science SciPy Signal processing Simplicity Special functions SymPy Twitter Typography Unicode Windows

Subscribe by email

Subscribe by RSS

John D. Cook

Search ...

Search