

Random Experiments on Color Image Processing and Enhancement

Digital Image Processing

by Trần Minh Hiếu, Nguyễn Gia Phong, Nguyễn Văn Tùng,
Nguyễn An Thiết and Nguyễn Thành Vinh

July 8, 2020

Contents

1	Introduction	2
1.1	Brief Description	2
1.2	Authors and Credits	2
2	Color Spaces	2
2.1	RGB Color Model	3
2.2	The CMY Color Model	3
2.3	The HSI Color Model	4
3	Color Image Enhancements	6
3.1	Contrast Stretching	6
3.2	Component Stretching	6
3.3	Local Contrast Enhancement	6
4	Pseudo Color Rendering	6
5	Conclusion	8
6	References	8

1 Introduction

1.1 Brief Description

Color images existed long before the rise of computing and digital image processing. While most techniques of monochrome image processing such as blur and edge detection can be directly applied to color images, others require modification. Furthermore, there exists procedures specific only to color images. In this project, we try to implement some of these techniques and note down our findings.

This report is licensed under a CC BY-SA 4.0 license, while the source code is available on GitHub* under GPLv3+.

1.2 Authors and Credits

The work has been undertaken by group number 8, whose members are listed in the following table.

Full name	Student ID
Trần Minh Hiếu	BI9-101
Nguyễn Gia Phong	BI9-184
Nguyễn Văn Tùng	BI9-229
Nguyễn An Thiết	BI8-174
Nguyễn Thành Vinh	BI8-187

We would like to express our special thanks to Dr. Nghiêm Thị Phương, whose lectures gave us basic understanding on the key principles of digital image processing. The color image processing lecture notes from the UMSL's CS 5420 course [1] also help us gain initial intuition on the matter.

2 Color Spaces

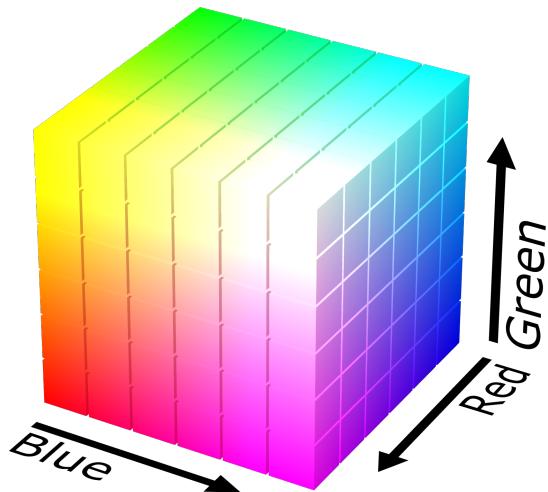
The color spaces in image processing aim to facilitate the specifications of colors in some standard way. Different color spaces suit different usage, namely RGB for computer display, CMYK for printing.

*<https://github.com/McSinx/recipe>

2.1 RGB Color Model

The RGB color model is an additive color model [2]. The name of the model comes from the initials of the three additive primary colors, red, green, and blue.

The color subspace of interest is the cube shown in the figure below, in which red, green and blue values are at three corners; cyan, magenta and yellow are at three other and black is at the origin and white is at the corner farthest from the origin.



In this model, the gray scale (point of equal RGB values) extends from black to white along the line joining these two points. The different colors in this model are points on or inside the cube and are defined by vectors extending from the origin.

2.2 The CMY Color Model

Cyan, magenta and yellow are the secondary colors of light. For instance, cyan subtracts red light from reflected “white” light composed of equal amounts of red, green and blue light.

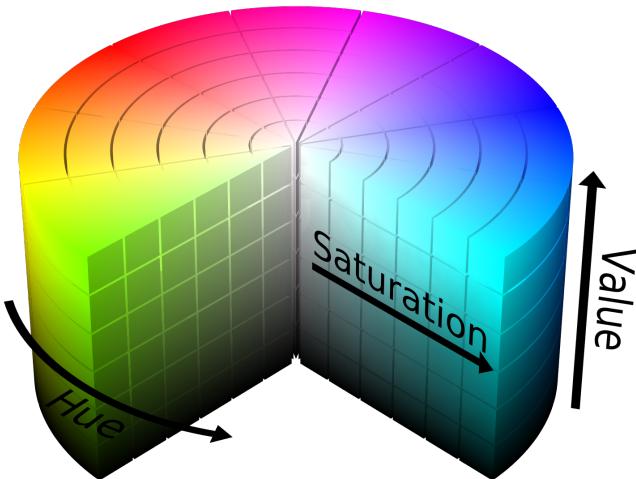
Most devices that deposit colored pigments on paper such as color printers and copies, require CMY data input or perform an RGB to CMY conversion

internally. Such conversion is performed using the following operation

$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

2.3 The HSI Color Model

Additive colors models such as RGB or CMY as shown above are not well suited for describing colors in terms that are practical for human interpretation. When humans view a color object we describe it by its hue, saturation and brightness.



While hue is the dominant color as perceived by an observer (red, orange, or yellow), saturation is the relative purity of color—pure spectrum colors are fully saturated [1]. The definition of value is some what less trivial, and thus will be shown as Python code converting from BGR to HSV and vice versa[†]:

```
def bgr_to_hsv(b, g, r):
    """Convert a pixel in BGR to HSV."""
    minc, maxc = min(r, g, b), max(r, g, b)
    if minc == maxc: return 0.0, 0.0, maxc
    diff = maxc - minc
    sat = diff / maxc
```

[†]This is taken from Python standard library `colorsys`, and adapted from RGB to BGR for OpenCV usage.

```

rc, gc, bc = (maxc-r)/diff, (maxc-g)/diff, (maxc-b)/diff
if r == maxc: return (bc-gc)/6%1, sat, maxc
if g == maxc: return (2.0+rc-bc)/6%1, sat, maxc
if b == maxc: return (4.0+gc-rc)/6%1, sat, maxc

def hsv_to_bgr(h, s, v):
    """Convert a pixel in HSV to BGR."""
    if s == 0.0: return v, v, v
    f = h * 6 % 1
    p = v * (1 - s)
    q = v * (1 - s*f)
    t = v * (1 - s*(1 - f))
    i = int(h*6%6)
    if i == 0: return p, t, v
    if i == 1: return p, v, q
    if i == 2: return t, v, p
    if i == 3: return v, q, p
    if i == 4: return v, p, t
    if i == 5: return q, p, v

```

To use these with images whose pixels stored as 8-bit unsigned integers, we decorate them with

```

def hhu(func):
    return lambda a, b, c: [
        int(i*255) for i in func(a/255, b/255, c/255)]

```

then the conversion of images stored as NumPy arrays of `uint8` will be as trivial as

```

from itertools import starmap
from numpy import reshape, uint8

def convert_color(image, func):
    x, y, z = image.shape
    return reshape(list(starmap(func, reshape(image, (x*y, z)))), (x, y, z)).astype(uint8)

```

As a simple test, conversion back and forth should give the same image: `convert_color(convert_color(image, bgr_to_hsv), hsv_to_bgr)`. We noticed that our implementation in pure Python is significantly slower than OpenCV's `cvtColor` which is from a C extension module.

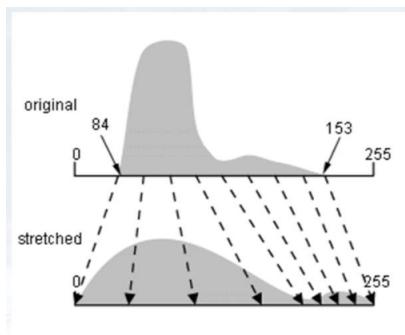
3 Color Image Enhancements

3.1 Contrast Stretching

Contrast Stretching is a linear image enhancement technique that tries to improve the contrast by stretching the intensity values of an image to fill the entire dynamic range.

An image of low contrast has a small difference between its dark and light pixel values. The histogram of a low contrast image is usually bends to the left (mostly light) or to the right (mostly dark), or located around the middle (mostly gray). Contrast Stretching computes the highest and the lowest pixel intensity values, sets them to 255 and 0 respectively, and scales all other pixel intensities accordingly.

$$\text{contrast} = \frac{I_{\max} - I_{\min}}{I_{\max} + I_{\min}}$$



Both Contrast Stretching or Histogram Equalization can be used to adjust image intensities to enhance contrast of pictures. While a non-linear Histogram Equalization technique is more reliable, in contrast stretching, there exists a one-to-one relationship of the intensity values between the source image and the target image i.e, the original image can be restored from the contrast-stretched image. This cannot be done with Histogram Equalization.

3.2 Component Stretching

3.3 Local Contrast Enhancement

4 Pseudo Color Rendering

By mapping each intensity level to a color, one may derive a pseudo color image from a greyscale images. Typical usage of such technique is in thermal

imaging, elevation and medical imaging to help the human visual system pick out detail, estimate quantitative values, and notice patterns in data in a more intuitive fashion [3].

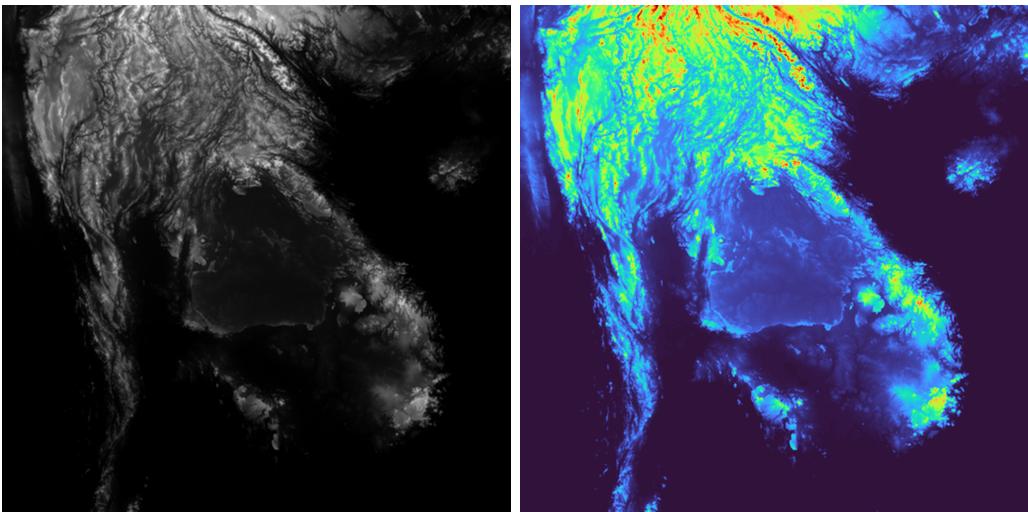
In OpenCV, this is available under `applyColorMap`. It is also trivial to reimplement this function only using NumPy:

```
import numpy as np

def map_color(grey, mapping):
    r = np.vectorize(lambda i: mapping[i][0])
    g = np.vectorize(lambda i: mapping[i][1])
    b = np.vectorize(lambda i: mapping[i][2])
    # OpenCV uses BGR by default for whatever reason.
    return np.stack((b(grey), g(grey), r(grey)), axis=-1)
```

For demonstration, we are going to use the Turbo colormap [3]. We initially considered solely changing the hue based on intensity, which is also known as the rainbow map, however it is not a visually intuitive mapping [4].

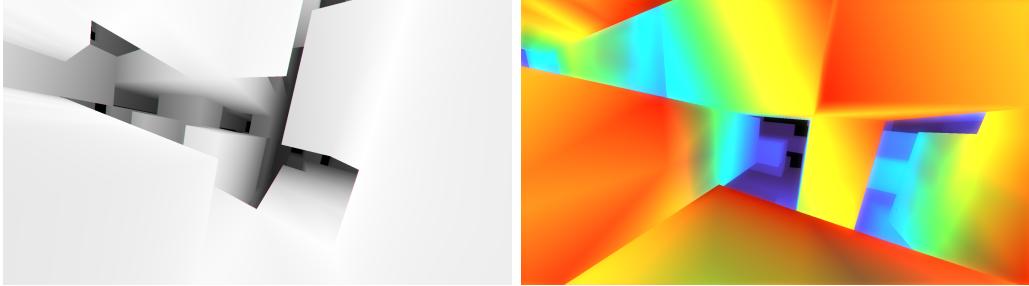
Firstly, we tried to apply the mapping on the heightmap of mainland Vietnam and its neighboring regions[‡]. As seen in the side-by-side comparison, the colormapped image allows the human eyes to notice more details, especially the high mountains in the north and the Khorat Plateau (center of the image).



In addition, we experimented with pseudo lighting, that is, use colormaps in place of normal greyscale lighting. The experiment was carried out on

[‡]The original images are taken from heightmapper: <https://tangrams.github.io/heightmapper/>

Phong's game Axuy, where colormapping proved to be an enhancement on helping players detecting shooting range (Axuy is a first person shooter game). The video where the game is in action is available on YouTube[§].



5 Conclusion

6 References

- [1] Sanjiv K. Bhatia. “Color Image Processing”. *CS 5420: Digital Image Processing*. University Of Missouri—St. Louis, Fall 2018.
- [2] Robert Hirsch. *Exploring Colour Photography: A Complete Guide*. Lawrence King Publishing, 2004. ISBN 1-85669-420-8.
- [3] Anton Mikhailov. *Turbo, An Improved Rainbow Colormap for Visualization*. Google AI Blog, August 20, 2019.
- [4] Noeska Smit. *Rainbow Colormaps—What are they good for? Absolutely nothing!* medvis.org, August 21, 2012.

[§]<https://www.youtube.com/watch?v=QVGAaoordpk>