Colleen Toth
Prof. Feng Liu
CS510, Final Term Paper
05 June 2015

Implementation of Robust Color-to-gray via Nonlinear Global Mapping

Colleen Toth

Abstract

My goal was to implement Kim, et al.'s algorithm described in the group's 2009 paper, *Robust Color-to-gray via Nonlinear Global Mapping*. The objective there was to create a robust, feature preserving algorithm for color to gray conversion with respect to the image's original lightness of colors through nonlinear global mapping. The group themselves were successful in this endeavor. My approach to this implementation was to implement the algorithm in Python, hoping to create a lightweight, more accessible implementation using Python libraries and OpenCV, while preserving the the original goals of the algorithm. Overall, I believe I was mostly successful in this task, though some work still remains to create completely identical results.

Introduction

Color to grayscale conversion is a common task across many disciplines including black-and-white printing, single-channel image processing, non-photorealistic rendering with black-and-white media, and profession and amateur photography. There has been dissatisfaction expressed with using the traditional approach emphasizing the CIE Y, the intensity channel, conversion technique as the appearance of the original colors is not always apparent in the resulting grayscale conversion.

Of the two main conversion methods proposed, local mapping and global mapping, the authors chose global mapping for its consistent mapping of colors to grayscale throughout the entirety of the image. The feature discriminability retention in the local mapping approach is appealing, and a global mapping method that preserves local features in different areas would be ideal. However, homogeneity of the color image through the image takes precedence.

The algorithm presented in *Robust Color-to-gray via Nonlinear Global Mapping*, does preserve reproduce the visual appearance of color images over a wide variety of images. By choosing a mapping based on the lightness, chroma and hue, the authors have met both the ability to provide consistent color mapping throughout an image to grayscale, but also retain feature discriminability, which they also extend to temporally coherent video conversion.

I did not address color to gray conversion over video, however my implementation of the algorithm also preserved feature discriminability as well as consistency in color mapping, however I believe the authors' version had more drastic distinguishability, which I believe may be the result of the Python implementation of OpenCV's imread() function, which reads the image in with a slightly different color

scale than the original image, which is apparent when displaying the image after reading it in.

Previous Work

Kim, et al. relied on a wealth of work done in color space and theory, as well as other possible solutions presented for converting color images to grayscale. This is especially so with their decision to take a nonlinear approach to color mapping as linear local mapping can result in feature loss where the final addition of luminance and chrominance can cause contrasts to cancel, and linear global mapping can leave artifacts in places where it cannot discriminate colors aligned perpendicular to the projection axis. These details and the other research utilized are discussed in greater depth in the Kim, et al. paper. The work I most relied on for implementing their algorithm, besides the original paper itself, was Nayatani's *Simple Estimation Methods for the Helmholtz-Kohlrausch Effect [1997].* Nayatani's paper provided the equations and values for converting CIELUV colorspace to Helmholtz-Kohlrausch (H-K) lightness predictor (LHK) for use in computing the color difference between pixels in the G(x,y) function. The algorithm uses Nayanti's method without modification.

Nonlinear Global Mapping

The approach for this algorithm focuses on the lightness, chroma and hue angle of an image to determine the color mapping to gray. The ultimate equation we look to solve is:

$$g(x, y) = L + f(\theta)C, \qquad (1)$$

where g(x,y) is a simple nonlinear global mapping and L, $f(\theta)$, and C correspond to the lightness, chroma and hue angle respectively. This equation, however, will not be solved until the end of my implementation, as we must go through several steps to reach it. These value are obtained from CIELCH colorspace, which is not the natural environment of a color image. Equation 1 retains the consistent mapping of colors throughout the image. $f(\theta)$ is defined as:

$$f(\theta) = \sum_{k=1}^{n} (A_k \cos k\theta + B_k \sin k\theta) + A_0,$$

Before these two equations can be solved, however, we must solve the energy function described in the next section.

To retain feature discriminability in regions the group used gradients to represent local pixel differences and minimize the difference between color and grayscale image gradients [Kim, et al. 2009]. As a result, it was necessary to define the measurements of the color differences in 1D, since color is 3D and grayscale is 1D. A normalized distance between two colors in CIE L*a*b* colorspace was used for measurement, however in

conversion color to gray the relationship cannot be preserved. Here *f(θ)* is applied so the differences between neighboring grayscale pixels and the color difference between neighboring color pixels are similar. As a result, the global mapping *g* changes adaptively with the distribution of color values in a given image [Kim et. al. 2009].

The global mapping, *g* , also accurately accounts for lightness fidelity by considering C - chroma - in the equation. Depending on the value of C, C will influence the conversion of pixels to prevent abrupt changes in of lightness.

Optimization

The energy function provides a way to mines the difference of image gradients between the pixel and the resulting grayscale we want to map to. This is described as:

$$E_s = \sum_{(x,y)\in\Omega} \|\nabla g(x,y) - \mathbf{G}(x,y)\|^2, \qquad (3)$$

where *(x,y)* is a pixel in Ω and ∇g is a gradient between pixels described as:

$$\nabla g(x, y) = (g(x+1, y)-g(x-1, y), g(x, y+1)-g(x, y-1))$$

and G describes the difference between input colors c, defined by:

$$\mathbf{G}(x,y) = \begin{pmatrix} G^x(x,y) \\ G^y(x,y) \end{pmatrix}^T = \begin{pmatrix} \mathbf{c}(x+1,y) \ominus \mathbf{c}(x-1,y) \\ \mathbf{c}(x,y+1) \ominus \mathbf{c}(x,y-1) \end{pmatrix}^T$$

and the color difference operator $\ominus$

$$\mathbf{c}_i \ominus \mathbf{c}_j = sign(\mathbf{c}_i, \mathbf{c}_j)\sqrt{{\Delta L_{ij}}^2 + \left(\alpha \frac{\sqrt{{\Delta a^*_{ij}}^2 + {\Delta b^*_{ij}}^2}}{\mathcal{R}}\right)^2}, \quad (4)$$

where c is represented by CIEL*a*b* in the function above. R is a normalization constant provided in the paper to be , R = 2.54√2, which acts to equalize chromatic contrast, which is the nest square root function above R in (4), and the lightness contrast, ΔLij. Alpha is a user specified parameter that was fixed at 0.01 for experimentation purposes in my implementation, but it is used to control the chromatic contrast on feature discriminability.

The sign function is used to prioritize the ordering of pixels. It merely returns the sign of the given value. The highest priority is given to the H-K effect predictor and it is used whenever it is non zero. It is computed using Nayatani's VAC model as discussed previous. However if that is zero, we look next to the sign of ΔL, which if also is zero, we use the sign of $\Delta L^3 + \Delta a^{*3} + \Delta b^{*3}$

Now it can be noted that we can reduce the energy function Es to a quadratic equation of unknown parameters of g as represented below

$$f(\theta) = t^T x,$$

where t = (t_i) abd x = (x_i) are (2n + 1) vectors. $1 \leq i \leq n$, $t_i = \cos(i\theta)$ and $x_i = A_i$. For $n + 1 \leq i \leq 2n$, $t_i = \sin((i - n)\theta)$ and $x_i = B_{i-n}$. For $i = 2n + 1$, $t_i = 1$ and $x_i = A_0$ [Kim et al. 2009]. This can be further manipulated to

$$E_s = \mathbf{x}^T \mathbf{M}_s \mathbf{x} - 2\mathbf{b}_s^T \mathbf{x} + \mathcal{C}, \qquad (5)$$

where Ms is the the same of the pixels ($uu^T + vv^T$), bs is the sum to Ω ($pu + qv$), u = (C ·t)x, v = (C ·t)y, p = Gx –Lx, and q = Gy –Ly. The subscripts x and y are used to denote partial derivatives which are calculated using central differences.

The authors (and I in my implementation) used a n = 4, n being the number of cos and sin bases in equation 2determining the degrees of freedom in g. This results in Ms as a 9x9 matrix and bs as a 9 x 1 vector. The authors did not see any significant change in results when increasing n in most cases. I confirmed this when experimenting with larger values.

In order to deal with the possibility of some cases where Ms becomes a null matrix or if elements of x cause *g* to go outside the visible range, Kim et al introduced a regularization term, $Er = x^T Ix$ with a weight of lambda. The authors found N, where N is the number of pixels in the image, for λ and obtained good results. I I used it as well and produced good results. After adding Er to the equation we end up with

$$E_{image} = Es + \lambda Er, \qquad (6)$$

which is minimized by

$$x_{image} = (Ms + \lambda I)^{-1} bs. \qquad (7)$$

Results

I implemented this paper using Python 2.7 in the Intellij Pycharm Integrated Development Environment using the Numpy, OpenCV 2.4.10, SciPy Image Toolkit, Math, and MatplotLib libraries. All the experiments were performed on a 2013 Macbook Air running Yosemite with an 1.3 GHz Intel Core i5 and 8 GB DDR3 RAM. All images were run single threaded.

The results were very similar to the images achieved by Kim et al. however, I noticed that the images I was reading in were displayed more with a blue hue than the original. he results I got as output were therefore also darker in ways that were unexpected. For example in Figure 1, The original image has an orange flower, however OpenCV read it as a blue flower and as a result my flower in the output was rendered very dark. After the presentation I realized my error and fixed this by converting the image from RGB to BGR and I attained the same results as the paper.

I ignored gamma for this implementation as I did not feel it would be worth the investment for the scale of the time allotted for the project and the type of images of which I was working. Further examples of my results are included as an appendix. I was unable to test the majority of the same images as the paper used as unfortunately they were all in the pdf of the paper. I was however, able to get one image from their website which i have labeled as Figure 2.

Discussion and Future Work

Python was a good choice in terms of implementing the algorithm, however it did not give me any benefits with regard to speed. The implementation in its entirety was less than 150 lines of code without comments. I chose Python because of my familiarity with the language and I knew I could leverage the Numpy library for matrix operations. Overall, I think it was an excellent choice for implementing the algorithm and allowing me to focus more on the math I needed to do than the syntax of the language or datatypes.

Performance-wise, my implementation is very slow. To do a normal high-resolution image of 4288 x 2848 took over an hour to run, which is unreasonable. I think this is partially do to the laptop I was running the experimentation on (i5 can not multithread) and that I was only running it on a single laptop. The original groups use of OpenMP leads me to believe these images were being processed on multiple computers in parallel. Furthermore, Python itself is not the fastest of languages, so it could be that for this task Matlab or C++ is a better choice.

Finally, I feel my implementation could have utilized full matrix operations over for loops. In the future, when I become more proficient with full matrix operations I would like to revisit this project and see if that would help increase performance. I use three for-loops in the paper which are death-traps in terms of computational efficiency, so I think this is an area that I could improve in the future.

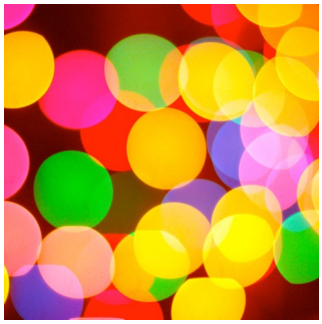
Figure 1

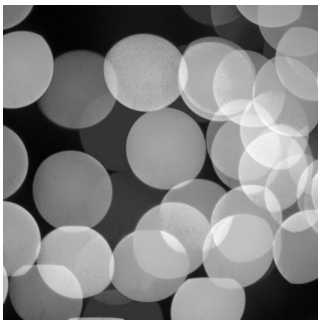
Output before conversion to BGR

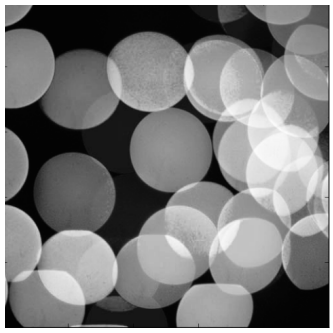
Original

Output after conversion to BGR

Authors' Output







Original **Photoshop** **Output**

References

KIM, Y, ET AL. 2009. Robust color-to-gray via nonlinear global mapping. *ACM SIGGRAPH Asia*. 28, 5.

NAYATANI, Y. 1997. Simple estimation methods for the helmholtz-kohlrausch effect. *Color Research and App.* 22, 6, 385-401.