Saliency Enhanced Colorization Optimization

Joyce Zhang (yunyizha)

https://github.com/SadSaltLady/TermProject_Coloration

Abstract

This is a two-part project hoping to develop a colorization algorithm for non-photorealistic images. In the first part of this project, we reimplement the basic algorithm presented in Colorization using optimization [Levin]. In the second part of the project, we expand on the existing algorithm by implementing a weighting scheme based on saliency in hope to improve its results on coloring non-photorealistic images. The hope is that by using a weighting scheme that has more coherent edge detection, we may avoid bleeding and other imaging issues in the original algorithm and generate overall better results.

1. Introduction

Colorization is a computer-assisted process of adding color to a monochrome image or movie. This process is traditionally done via image segmentation and recoloration, which requires a lot of human input and time. In Levin's paper *Colorization using optimization*, a new method is proposed for coloring images. This new method is guided by a simple principle: "neighboring pixels … that have similar intensities should have similar colors." [Levin] This new method of viewing colorization as an optimization problem demanded less user input and generated equally promising results. Instead of manual (or computer assisted) segmentation of image into sections of color sections, the new process only required the user to provide guiding visual color clues that are less precise. Although today we have new methods of colorization (such as machine learning assisted), the control of which the user has over the image makes this method useful for artistic purposes. This project sees colorization via optimization a assistive tool in art creation that gives a level of agency to the artist, while speeding up the coloration process.

This algorithm generates fairly convincing results, but it still faces several challenges and shortcomings, which we will discuss:

1.1. Reliance on human input

The final colorations generated are highly dependent on the input it received, and small variations in input may generate different looking results. For each region which the user wishes to be colored with the same color, there needs to be sufficient information provided for the algorithm to work as intended, as shown in Figure (1).

The general rule is that more user input generates more desirable results, however there's no good way of evaluating the "quality" of user input prior to the algorithm running. This means that generating good results requires users running multiple trails, which is counteracting the purpose of which this algorithm is developed – to reduce time it takes to colorize images. Unfortunately, although being a persistent problem that needs to be addressed, this is not the one which this project wishes to tackle.

1.2. Image dependencies

The algorithm is reliant on the underlying gradient distribution in the intensity of the image. In practice, it's observed that images that this algorithm works a lot worse on images are lacking in gradient information – namely images lacking clear boundaries, non-photorealistic images, or hand-drawn inputs. The results usually exhibit more bleeding than usual.

This project focuses on solving the second problem by adopting a saliency enhanced gradients calculation into Levin's algorithm. In the paper *GradientShop: A Gradient-Domain Optimization Framework for Image and Video Filtering*, the authors propose "a novel metric for measuring local gradient-saliency that identifies salient gradients that give rise to long, coherent edges". Compared with traditional gradient based algorithms, the saliency measurement is better as it offers more coherent boundaries. The hope is that by more strictly enforcing restrictions along these coherent edges, this would prevent bleeding of colors in the process of coloration.

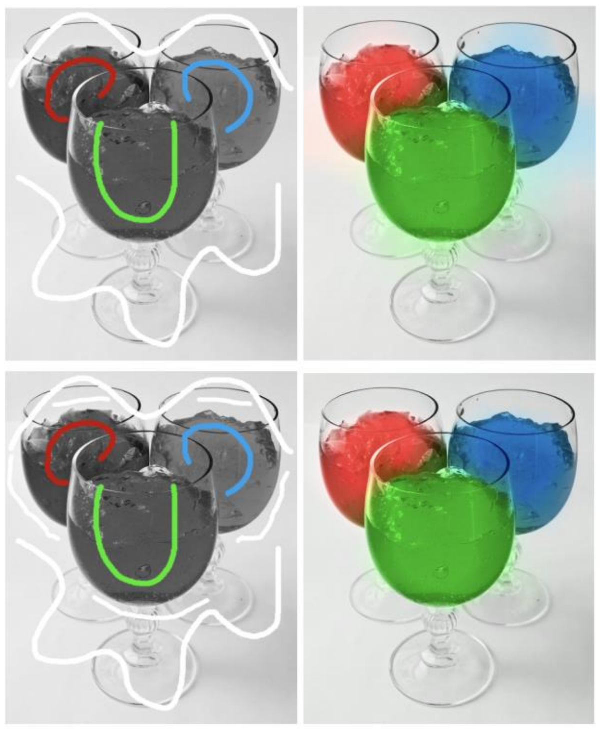

Figure 1: Example of user input effecting output. The user may specify more input (bottom row) to prevent bleeding in the colorized result, but they would not know this would happen until they receive the first set of results (top row).

2. Coloration via Optimization

Before looking at the project implementation of the coloration algorithm, it should be noted that Levin's paper itself did not provide many details on the implementation except for the math principles guiding it. Much of this the implementation of this project required the author's interpretations and may have generated different results than the original algorithm.

2.1. Method

2.1.1 Input

The input to Levin's algorithm are two images:

1. A grayscale image, serving the base as the coloration.
2. A clue image, which contains user color marks on top of the grayscale image.

In this paper, we will denote the values referencing the original image with the suffix $\_orig$, and values referencing the visual clues with the suffix $\_clue$.

Levin's did not have specific requirements on if the images had to be linear or not, thus this project assumed that it doesn't matter.

2.2. Processing

The method described in Levin's paper follows the simple principle of "neighboring pixels … that have similar intensities should have similar colors". To acquire information needed to measure similarity in intensity, the images are converted from RGB space to YUV space. In the YUV space, Y provides a measurement of the intensity.

To create the actual colorization, we will be setting up an optimization problem in which we wish to minimize the difference between the colored pixel $U(r)$ at a position r with the weighted average(affinity) of its neighboring pixels. The affinity could be expressed with:

$$J(U) = \sum_r \left(U(r) - \sum_{s \in N(r)} w_{rs}\, U(s) \right)^2$$

From this, the colorization problem can be turned into an optimization problem where we try to minimize $J(U)$. This optimization problem yields a large, sparse system of linear equations. We can set up the problem with

$$WX = b$$

Where W is a matric of size $(h * w, h * w)$ containing the weights term $-w_{rs}$ of the above equation, and b of size $(h * w)$ contains constraints for the U or V color channels. We generate a set of constraint from the intensity channel Y, and two b vectors, one for U and one for V channel. The pseudocode for setting up this system is described below:

```
For pixel $r$:
    if ($r$ contains visual clue) :
        $W[r,r] = 1$
        $b[r] = img_{clue}[r]$
    else:
        $s_n$ = neighbors of $r$
        $W_{[r,s_n]} += -w_{rs}$
        $W[r,r] = 1$
```

Where w_{rs} is the weighting equation described in Levin's paper:

$$w_{rs} \propto e^{-(Y(r)-Y(s))^2/2\sigma_r^2}$$

Where Y refers to the intensity channel in the original image.

2.3. Results and Encountered Problems

Overall, the implemented algorithm works well(see Figure 2). The most visible problem present in the colorized results is the bleeding of colors, as described in the introduction. The result varies depending on the user input, and it might take several attempts before finding the version which yields the optimal results. Testing this on

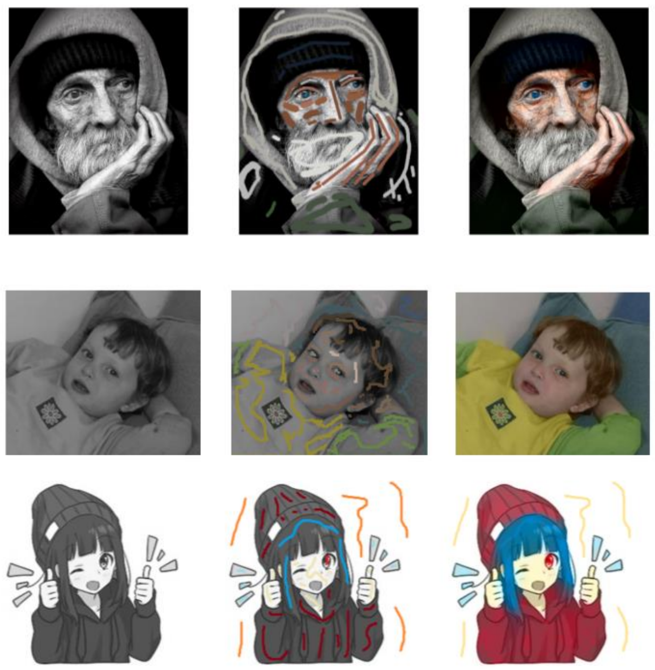

Figure 2: Examples of coloration. From top to bottom row: internet image result; original paper's data, rendered with project implementation; non-photorealistic data.

non-photorealistic images yield surprising ok results. As hypothesized, the coloration falls short when there is no underlying gradient.

Another potential problem with this implementation sometimes produces "un-defused" coloration results. The location where the user-provided color cues are very visible, as surrounding areas are of a fainter color. It's not quite clear why this is the case. One potential reason is that this is again resulting from the algorithm requiring user input.

In the next part of this paper, we will look at how we can potentially over come some of these problems with saliency enhanced gradients.

3. Saliency Enhanced Coloration

Saliency enhanced gradients gives better detection of long, coherent edges. The principle behind this is that we imagine for each point along a certain edge, the point has two properties: the direction which the edge is passing through it, and the intensity of the edge at the location. Theoretically with these information, we can start "growing" edges from points we know that exists on an edge.

3.1. Concept

The saliency enhanced gradients are measurement of such values, and it is governed by several values:

1. The directional gradient u_x & u_y
2. The normalized gradient magnitude $\hat{p}_m$
3. The gradient orientation $e^o(p)$
4. The edge length estimation $e^l(p)$

These values are derived from the directional gradient information of the image, which we denote as p_m. Directional gradient is calculated and contains partial derivatives in the x and y direction. The (directional) saliency is calculated with:

$$s_x(p) = cos^2\left(e^o(p)\right) * e^l(p) * u_x(p)$$
$$s_y(p) = sin^2\left(e^o(p)\right) * e^l(p) * u_x(p)$$

The math behind each value are described below:

3.1.1 The normalized gradient magnitude $\hat{p}_m$

The normalized gradient is calculated with respect to its local neighborhood, as saliency hopes not only detect edges with strong magnitudes. Normalizing the local magnitude gives better detection to faint but coherent edges.

$$\hat{p}_m = \frac{p_m - \mu_w}{\sigma_w + \epsilon}$$

where μ_w and σ_w denote the average and the standard deviation of edge magnitudes in the pixel neighborhood w. The results shown in paper the size of w is set to $(5 * 5)$.

3.1.2 The gradient orientation $e^o(p)$

The gradient orientation estimates the angle between the x and y channel of the gradient, described by the equation below:

$$e^o(p) = atan2\left(p_{my}, p_{mx}\right)$$

3.1.3 The edge length estimation $e^l(p)$

The edge length is described in the following equation

$$e^l(p) = m_0^t(p) + m_1^t(p) + \widehat{p_m}$$

where $m_0^t(p)$ and $m_1^t(p)$ are estimations of the lengths of two sub-edges that start at the pixel and proceed in two opposite directions given by the local edge orientation. These two values are calculated iteratively, described by the variable t.

$$m_0^t(p) = \sum_q w_\alpha(q) * w_\theta(q) * \left(\hat{q}_m + m_0^{t-1}(q)\right)$$
$$m_1^t(p) = \sum_q w_\alpha(q) * w_\theta(q) * \left(\hat{q}_m + m_1^{t-1}(q)\right)$$

The function w_α computes the weights for bilinear interpolation between the integer coordinates. The function w_θ measures the similarity of the edge orientations as

$$w_\theta = exp(-(p_\theta - q_\theta)^2/2\sigma_\theta^2)$$

where the variance is set to $\pi/5$.

3.2. Implementation and Results

In GradientShop, the suggested implementation of integrating saliency enhanced gradients is through adjusting the weighting scheme used in the algorithm such that

$$w_x(p) = \frac{1}{(|s_x(p)| + 1)^b}$$

$$w_y(p) = \frac{1}{\left(|s_y(p)| + 1\right)^b}$$

where b can be adjusted to modify the influence of saliency filter. In this project's implementation of the coloration scheme, there was no directional gradient used. Instead we take the magnitude of the directional gradients into one.

$$w = sqrt\left(w_y^2 + w_x^2\right)$$

The project implementation of the saliency seems to be functioning. Demonstrated in Figure 3, the algorithm has succeeded in detecting sharp edges throughout the image. Comparing with the gradient, notice the algorithm has captured edges of low intensity, and displays better consistent long edges.

The attempt to integrate this into the coloration algorithm, however, is less successful. Either the paper's two directional interpretation made them generate better results, or the project implementation misinterpreted how to use the weights. Incorporating the saliency into the weighting scheme has not generated significant impact on the outcome. If anything, it increased the bleeding see in the outcome (see Figure 4).

Unfortunately, with both parts successfully implemented, this project was unable to replicate the original paper's result of using saliency enhanced gradients to reduce bleeding (see Figure 5). Multiple factor could be influencing this, such as the specificity of user input, but there was not enough time to find satisfactory result.

4. Conclusion

Colorization via optimization has decreased the amount of human effort needed in coloration. As a creative tool, it gives sufficient control to the artist to colorize their art. It shortcomings are obvious as well, as variation in the input generates can generation less than desirable results. This paper attempted to mitigate bleeding artifact in colorization via optimization by implementing saliency enhanced gradient weighting schemes. Although the final attempt is unsuccessful, the project successfully implemented both parts of the algorithm. With new technologies such as machine learning, perhaps these tools can be a part of a tool kit to automate image colorization.

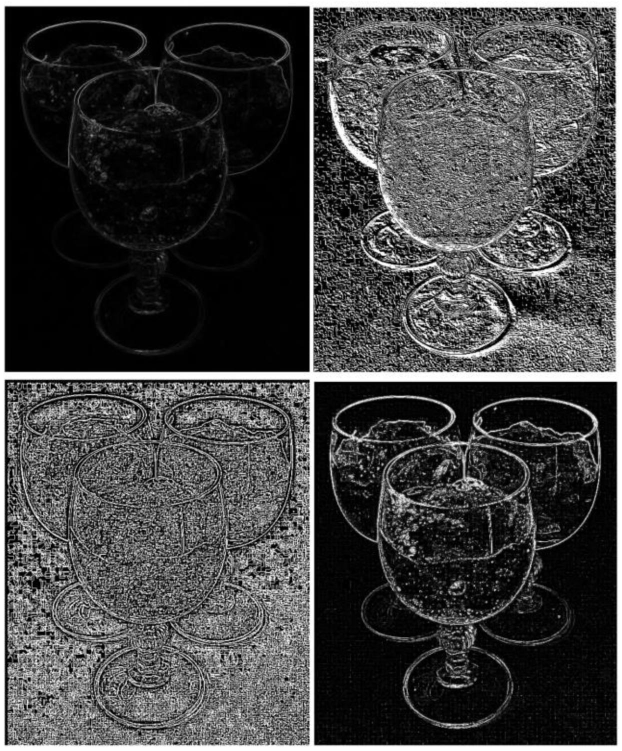

Figure 3. From left to right: gradient and orientation(top), edge length estimation and final saliency filter(bottom). This is produced with 60 iterations in the edge-length estimation.

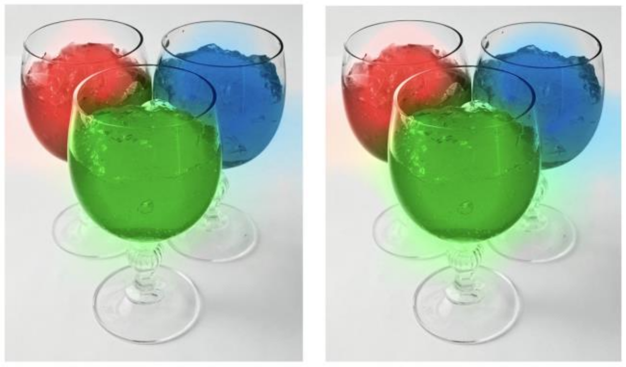

Figure 4. Left is the original algorithm, right is the saliency weighted coloration, with b set to 3.

References

[1] Bhat, Pravin & Zitnick, C. & Cohen, Michael & Curless, Brian. (2010). GradientShop: A gradient-domain optimization framework for image and video filtering. ACM Trans. Graph.. 29.

[2] Levin, A., Lischinski, D., & Weiss, Y. (2004). Colorization using optimization. In ACM SIGGRAPH 2004 Papers (pp. 689-694).

[3] Patankar, A.B., Kubde, P., & Karia, A. (2016). Image cartoonization methods. 2016 International Conference on Computing Communication Control and automation (ICCUBEA), 1-7.

[4] Xu, L., Lu, C., Xu, Y., & Jia, J. (2011, December). Image smoothing via L 0 gradient minimization. In Proceedings of the 2011 SIGGRAPH Asia conference (pp. 1-12).

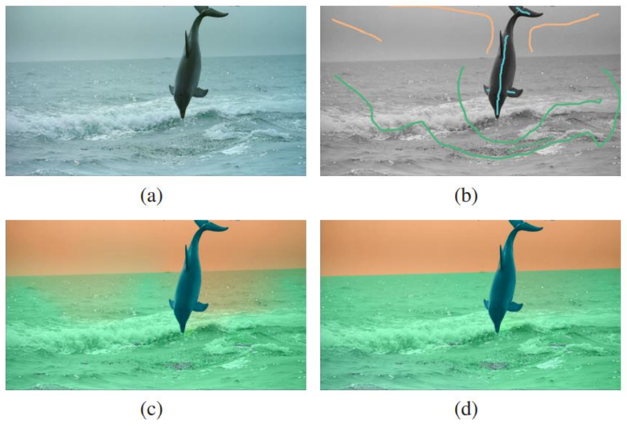

Figure 5. Expected result presented in GradientShop