Interactive Computer Graphics 2023 Spring, Term Project Report

Color Harmonization

Chin-Chieh Hsu 許晉捷 F09922184

Abstract

An image can be considered harmonious if we feel comfortable based on our individual perspective. Conventionally, artists and visual designers have been relying on their own instincts and artistic feelings to create impressive harmonic artworks. Color harmonization, on the other hand, is a computerized technique that applies a statistical algorithm to alter a digital image into a harmonized one. In this report, I will provide a detailed explanation of the methodology behind color harmonization, along with my own comprehensive implementation with a Graphical User Interface (GUI) using Python 3.8.0. Additionally, to elevate the performance in terms of time cost, some additional features not included in the original paper have been implemented as well.

Demo Video

https://youtu.be/Fd-HIYI_siQ

1. Contributions (貢獻)

The main contributions in my implementation can be categorized into two areas: GUI-related and algorithm-related, and are summarized as follows:

- Algorithm-related contributions
  a. The first full implementation of the original paper (Cohen-Or et al., 2006) thus far, including all details of the algorithm, with additional features as undermentioned, among resource that can be found on public Internet, to the best of my knowledge.
  b. The harmonic scheme of the harmonization process can be either based on the target image itself, or on an optional reference image.
  c. In conjunction with a Super Resolution (SR) technique (Wang et al., 2021), an enhancement has been made to improve the efficiency of the algorithm. This enhancement enables the use of smaller-sized inputs for the main process without sacrificing quality. Further details are described in Section 4.
  d. In conjunction with GrabCut (Rother et al., 2004), a foreground-background image segmentation technique, a high-level application has been developed to let users selectively apply the color harmonization process on background (and the color harmony is based on the foreground), or vice-versa. This was also mentioned in the original paper. Further details are described in Section 4.
  e. Users have the option to choose between selecting a specific template type or allowing the program to automatically search for the best template type. The concept of template types will be discussed in greater detail in Section 3.
- GUI-related contributions
  a. The first implementation along with a stand-alone GUI, including several extensive and user-friendly features, listed undermentionedly.

b. Four ways to load image: 1) from local 2) drag-and-drop 3) from URL 4) from clipboard. This enables the ample flexibility for individual users.
c. A specific GUI panel for algorithm settings for each single loaded image individually. The settings incorporate the template type of the harmonic schemes, the resize-ratio, and an optional reference image. These terms will be further explained in Section 3.
d. All algorithms run on other threads. The user-interface is not frozen; thus, users can continue operating (e.g., load another image) while the harmonization process is concurrently running.
e. An immediate and clear comparison will be displayed right after a process is done. This further enhances the user experience in terms of convenience and practicality.

2. Introduction

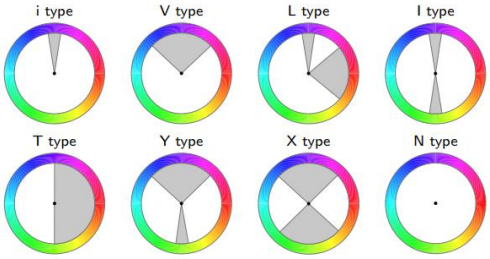

Figure 1. The eight harmonic templates as hue wheels, represented in HSV color space, developed by Matsuda (The figure originates from the paper (Cohen-Or et al., 2006)).

As Figure 1 shown, there exist eight types of harmonic templates: i-type, V-type, L-type, I-type, T-type, Y-type, X-type, and N-type (Matsuda, Y. 1995) (Tokumaru et al., 2002). Each template is represented as a ring with hues in the HSV (hue, saturation, and value) color space and one or two sectors, except the N-type. Note that here we do not incorporate the N-type template into consideration, since it was proposed for grayscale images.

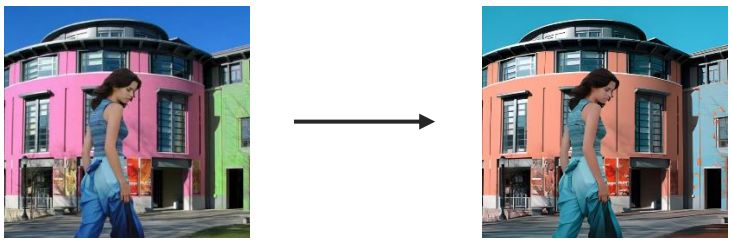

Figure 2. The goal of the color harmonization algorithm: the upper row is the raw hues in a ring-shaped histogram and the raw image; the second row is the eventual result after the algorithm. As the figure shows, we “pushed” the raw hues on the ring and enforcedly “shifted” them into sectors of the I-type template.

The goal of this computerized color harmonization algorithm is to alter a digital image by “shifting” the hues of all pixels along with a ring (a template) in order to make them be inside the sectors. This is depicted in Figure 2, which shows the goal of the algorithm. The upper row is the raw hues in a ring-shaped histogram and the raw image; the second row is the eventual result after the

algorithm. As the figure shows, we "pushed" the raw hues on the ring and enforcedly "shifted" them into sectors of the chosen I-type template in a statistical way.

3. Method

In this section, the entire color harmonization algorithm (Cohen-Or et al., 2006) is elaborated in details. The algorithm can be subdivided into three stages: 1) Harmonic scheme selection. 2) Graph-cut optimization. 3) Color shifting.

1) Harmonic scheme selection

A harmonic scheme is defined as (m, α), where m is the "index" of a template ($m \in \{i, V, L, I, T, Y, X\}$) and α is defined as an orientation (offset) of a template. In this stage, we aim to select a best harmonic scheme $B(X) = (m_0, \alpha_0)$ along with the T_{m_0}, the template indexed by m_0, that minimizes the following objective function

$$F(X, (m, \alpha)) = \sum_{p \in X} \left\| H(p) - E_{T_m(\alpha)}(p) \right\| \cdot S(p)$$

, where X is the input image, (m, α) is the harmonic scheme, p belonging to X represents all pixels in X, H and S refers respectively to the hue and saturation, and E denotes the closest sector's border from p. In particular, $E_{T_m(\alpha)}(p)$ represents that given a specific harmonic scheme (m, α), the closest border of the sector in the T_m, which is "rotated-by-α", from a pixel p.

This objective function F measures the color harmony of an image X, given an arbitrary harmonic scheme (m, α). We aim to minimize F to search for the best harmonic scheme, using Brent's method (Press et al., 2007), $B(X) = (m_0, \alpha_0)$, such that

$$m_0, \alpha_0 = \arg\max_{m,\, \alpha} F(X, (m, \alpha))$$

In practical, in order to save the time cost to do optimization, instead of applying Brent's method seven times to find out seven different α-values for each template type, we can let the user to choose a designated template type and apply merely once Brent's method.

The visualized process finding the best harmonic scheme can be depicted in Figure 3, as the red arrow represents. The "rotated" I-type template at the lower-right corner in Figure 3 can now be construed as the result of the first stage, i.e., the best harmonic scheme that we have found out.

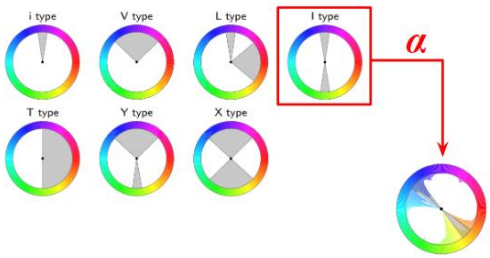

Figure 3. The visualization of the goal of searching for the best harmonic scheme $B(X) = (m_0, \alpha_0)$. In this particular case, m_0 is I (thus T_{m_0} is the I-type template). The "rotated" I-type template at the lower-right corner can now be construed as the result of the first stage, i.e., the best harmonic scheme that we have found out.

2) Graph-cut optimization

Now we have the best harmonic scheme, as shown as the "rotated" template at the lower-right corner in Figure 3. The problem now we have is that: can we shift the hues on the ring-shaped histogram along with the template directly and coerce them be located inside the sectors, e.g., the nearest sector? Unfortunately, the answer is NO, because we have not taken the spatial coherence into consideration yet.

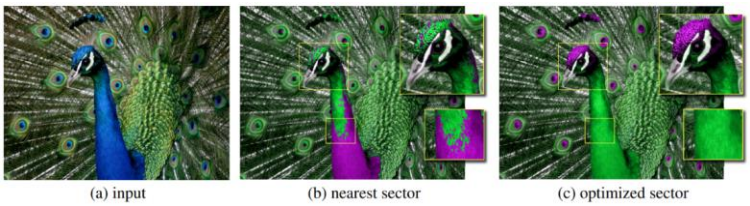

Figure 4. (a) the raw image (b) the result if we directly move each hue which are originally outside any sectors to the border of its nearest sector (c) the result after applying the Graph-cut optimization (The figure originates from the paper (Cohen-Or et al., 2006)).

As shown in Figure 4, If we simply coerce and shift each hue toward the border of its nearest sector, the unpleasant visual artifacts will be generated clearly. To produce the appealing results like (c) in Figure 4, in contrast to employing a harsh shifting method, the renowned Graph-cut optimization technique is adopted here (Boykov & Jolly, 2001).

Graph-cut is a technique originally used for binary classification tasks on images. In the context of the color harmonization, the objective is to assign a label to each pixel indicating its movement direction: clockwise or counterclockwise. This labeling can also be viewed as a binary classification task on images. Consider the following energy function E:

$$E(V) = \lambda E_1(V) + E_2(V)$$

$$E_1(V) = \sum_{i=1}^{|\Omega|} ||H(p_i) - H(v(p_i))|| \cdot S(p_i)$$

$$E_2(V) = \sum_{\{p,q\} \in N} \delta(v(p), v(q)) \cdot S_{max}(p,q) \cdot ||H(p) - H(q)||^{-1}$$

, where Ω is the set of input pixels, N is the set of all adjacent pixels in Ω, and λ is a balancing parameter between two equations. We search for a set of labels $V = \{v(p_1),\ v(p_2),\ \ldots, v(p_{|\Omega|})\}$, where each $v \in \{\Theta_1,\ \Theta_2\}$ is a binary label with respect to a pixel denoting the destination sector to which whether the hue of the pixel is moved clockwisely Θ_1 or counterclockwisely Θ_2.

The objective of E_1 is to evaluate distances between the raw and the "shifted-to-the-destination" hue. The saturation channel is considered to weigh the contribution from those pixels with low saturations. E_2, as a regularization term, accounts for enhancing the spatial coherence among adjacent pixels assigned to the same label by penalizing the different assignments to adjacency with similar hues. The term $\delta(v(p),\ v(q))$, where p and q are a pair of arbitrary adjacent pixels, in E_2 is for penalization: if $v(p)$ and $v(q)$ are different, i.e., one moves clockwisely; another moves counterclockwisely, this delta function will be 1 and the penalization ensues; otherwise, 0.

To do this minimization on E, Graph-cut represents the image as a graph, where nodes are construed as pixels and edges are relationships between adjacent pixels. By assigning weights to these edges based on color similarities in terms of spatial coherence, the optimization ensures that color modifications are carried out seamlessly, resulting in visually pleasing harmonized images.

3) Color shifting

Up to this point, we have acquired the best harmonic scheme $B(X) = (m_0, \alpha_0)$ and the label set $V = \{v(p_1),\ v(p_2),\ \ldots, v(p_{|\Omega|})\}$ that depicts the movements for each pixel. To do color-shifting:

$$H'(p) = C(p) + \frac{w}{2}\left(1 - G_\sigma\left(\|H(p) - C(p)\|\right)\right)$$

, where *C(p)* is the destination sector's center with respect to which *p* is moving to, *w* denotes the arc-width associated with that sector, and G_σ is the normalized Gaussian function. G_σ is to exert the ease of color changes; otherwise it can probably lead to visual artifacts.

4. Implementation

1) Implementation Environment

Platform	Windows 10
Programming language	Python 3.8.0
Third-party packages	See the attached *requirements.txt*
Other dependencies	Real-ESRGAN (Wang et al., 2021)
How to run	See the attached *README.md*

2) Features with GUI

a. Overview

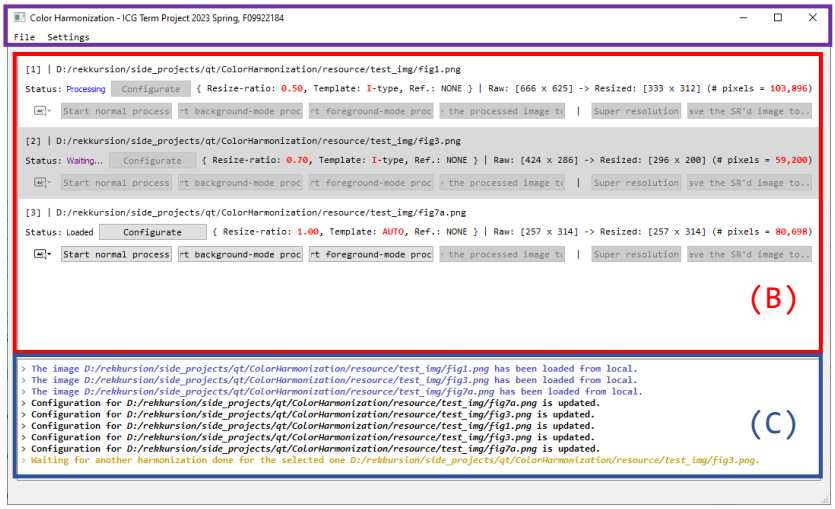

Figure 5. The main window of my program. (A) the title and the toolbar; (B) the list of items, i.e., all loaded images; (C) the log area.

Figure 5 shows the primary window of my program, which consists of three major components: (A) the toolbar, taking responsibility for global operations, such like file input/output, directory operations, and global settings; (B) the list of items, presenting all loaded images with comprehensive information, for each loaded image, including the path of the image, current status (*loading*, *loaded*, *waiting*, *processing*, *done*, or *error*), algorithmic configuration (i.e., resize-ratio, template type, and the optional reference image), and several operational buttons (alter configuration, start process, start Super Resolution process, save the result to the designated path, and show the image and visualization); and (C) the log area.

b. Image loading/writing

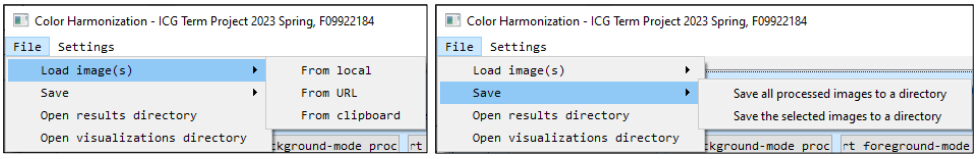

Figure 6. The program provides up to four ways to load images. The left figure displays the first three ways: (A) from local storage; (B) from URL (for Internet resources); (C) from clipboard; and (D) by drag-and-drop. The right figure shows that the program can let users conveniently save results, either all, or all selected, to a designated directory.

Figure 6 regards the file input, the program provides up to four different ways for users to load images: (A) from local storage via OS-native file dialogue (B) by dragging an image and dropping within the interface of the program (C) from a Uniform Resource Locator (URL) to let users conveniently load the image through Internet without explicitly downloading it (D) from the OS-native clipboard: the user can copy an image and it will be stored in the clipboard provided by OS.

As for the file output, the process results and visualizations will automatically be saved to a pair of pre-determined paths, respectively. These paths could be altered via a global preference panel. Moreover, the results could also be saved to arbitrary local storage explicitly by user.

c. Global settings

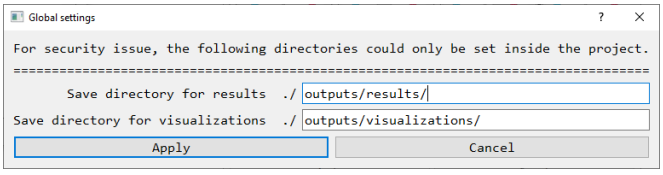

Figure 7. The global settings panel can let users set their prefered saving directories for both results and visualizations.

The global settings panel, which can be invoked by clicking the item *Settings* at toolbar, undertakes the arragement of directories, both for results and visualizations, as Figure 7 reveals. The default directories are respectively *./outputs/results/* and *./outputs/visualizations/*. Note that for security issue, both directories can only be inside the executable, i.e., inside the current working directory.

d. Individual settings for each loaded image

Following is Figure 8, which exposes the individual settings for each loaded image. These specific settings are comprised of: (A) which template type is prefered, besides, there is also a radio-button *AUTO* for automatically search for the best template type, by the algorithm elaborated in Section 3; (B) the resize-ratio, the computational cost can be reduced drastically since the number of pixels is reduced significantly; and (C) the selection of the optional reference image, which enables the ability to do color harmonization based on the harmony from another reference image.

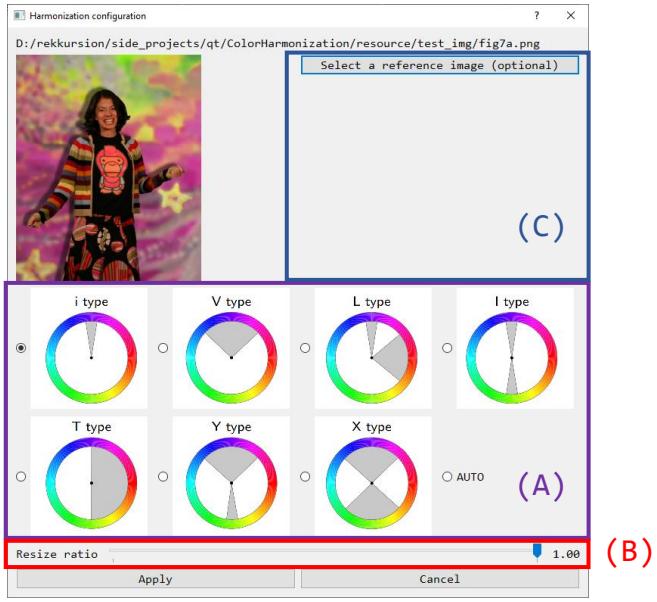

Figure 8. The case-specific algorithmic configuration panel.

e. Process and visualization

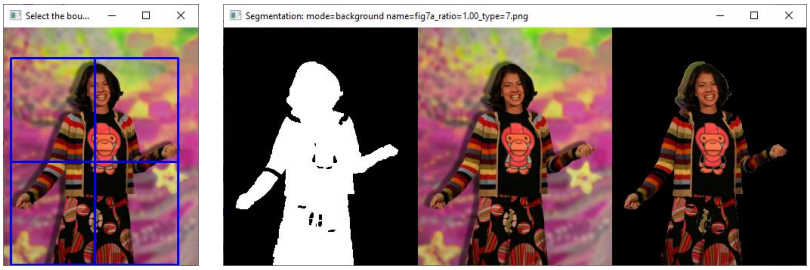

Figure 9. Left: A window to draw a bounding box in the foreground. Right: GrabCut's result.

When *Start normal process* button is clicked, the program will prepare all resources for applying the algorithm and start a new thread dedicated for processing, that is, the main thread (GUI thread) is not affected and can be operated concurrently. Additionally, two extra processes are available via *Start background-mode process* and *Start foreground-mode process* button, which will start a special process that first prompt up an operatable window (Figure 9, left) to let users draw a bounding box of the foreground object. Succeedingly, the program will apply GrabCut (Rother et al., 2004) to segment the bounded foreground (Figure 9, right). Finally, the color harmonization is processed, but the color harmony is based on the foreground (background) and harmonize the background (foreground) only.

Once the process is done (either normal or back/foreground-mode), a clear visualization for comparison between raw and processed images will be displayed, as Figure 10 shows.

Figure 10. The visualization after the process is done. This figure is an example of a "background-mode" process, i.e., harmonizing the background only, based on the foreground.

f. Adoptation of Super Resolution as a post-process after the core process

Except for all the details in the original paper, as thoroughly illustrated above, I have also adopted a modern Super Resolution technique, Real-ESRGAN (Wang et al., 2021), to further recover the original size. An obvious advantage by doing so is that a huge amount of computational time cost can be colossally reduced; while the quality of results is maintained to some extend.

5. Results and Discussions

a. Normal processes on examples from the original paper

Figure 11. Some results from normal processes. Experimentally speaking, I have found that the I-type template is the most useful one, as it has a relatively higher chance to obtain a good result, as the lower row of this figure depicts (the pond image).

b. Background-mode result

As shown in Figure 10.

c. Foreground-mode result

Figure 12. In this case, the program measured background's color harmony and applied the harmonization algorithm only on the foreground, based on the measurement. As the figure shows, the patterns on child's clothe are visibly harmonized, resulting a good tone of the image.

d. An external input

Figure 13. A result on an external image (Image source: https://imgur.com/gallery/uPwQ0IX).

6. Conclusion

In this project, I have implemented not only the full content of the original paper, including even back/foreground harmonizations utilizing GrabCut, but also some additional modifications: Super Resolution and the selectiveness on harmonic template types. To the best of my knowledge, this is the first full implementation of this paper, among all publicly available online resources, as well as the only one with GUI, which comprised of a plentiful amount of user-friendly features: four ways for image loading, multithreading processes, several practical ways for result-saving, well-timed visualizations, reference image supported, and handy algorithmic and global settings.

Demo Video

https://youtu.be/Fd-HIYI_siQ

Implementation-Related Acknowledgements

- Super Resolution: https://github.com/xinntao/Real-ESRGAN
- PyQt5 Drag-and-drop: https://gist.github.com/peace098beat/db8ef7161508e6500ebe
- PyQt5 Open a directory in the file explorer: https://stackoverflow.com/questions/6631299/python-opening-a-folder-in-explorer-nautilus-finder
- Brent's method: https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.brent.html#scipy.optimize.brent
- Minimum-cut: https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.flow.minimum_cut.html

Reference

Boykov, Y. Y., & Jolly, M.-P. (2001). Interactive graph cuts for optimal boundary & region segmentation of objects in ND images. Proceedings eighth IEEE international conference on computer vision. ICCV 2001,

Cohen-Or, D., Sorkine, O., Gal, R., Leyvand, T., & Xu, Y.-Q. (2006). Color harmonization. In *ACM SIGGRAPH 2006 Papers* (pp. 624-630).

Matsuda, Y. 1995. Color design. Asakura Shoten.

Press, W. H., Teukolsky, S. A., Vetterling, W. T., & Flannery, B. P. (2007). *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge university press.

Rother, C., Kolmogorov, V., & Blake, A. (2004). " GrabCut" interactive foreground extraction using iterated graph cuts. *ACM transactions on graphics (TOG)*, *23*(3), 309-314.

Tokumaru, M., Muranaka, N., & Imanishi, S. (2002). Color design support system considering color harmony. 2002 IEEE world congress on computational intelligence. 2002 IEEE international conference on fuzzy systems. FUZZ-IEEE'02. Proceedings (Cat. No. 02CH37291),

Wang, X., Xie, L., Dong, C., & Shan, Y. (2021). Real-esrgan: Training real-world blind super-resolution with pure synthetic data. Proceedings of the IEEE/CVF International Conference on Computer Vision,