|  |
| --- |
| **Technical Report on Palette-based Color Transfer** |
|  |
| **Brandon Cuadrado, 109237297**  Department of Computer Science  Stony Brook University, Stony Brook NY 11790 USA  [brandon.cuadrado@stonybrook.edu](mailto:brandon.cuadrado@stonybrook.edu) |
|  |
|  |
|  |
|  |

Abstract

|  |
| --- |
|  |

Palette-based Color Transfer is an area of image processing that has great artistic merit. Manipulating 2D images using color transfer has merit in both photorealistic and non-photorealistic areas of image manipulation. This technical report will cover the implementation process used to create a functional Palette-based Color Transfer web interface. This model can automatically extract the color palette of an input image, which the user can then alter and produce a resulting output image using the user-selected color palette.

Introduction

For my final project, I have created a Palette-based Photo Recoloring web interface. This model can extract the pixels from a given image and automatically extract a palette matching the input image. The user can then select a new color palette by altering each palette color using a provided color picker. The user can generate a new image that reflects the user-selected color palette. This intuitive interface provides an alternative to editing extracted histograms as implemented in commercial image-editing software. This tool provides non-experts in the field with a fully-realized color editing tool using simple palette editing input. The implementation of this Color Transfer model was most influenced by the paper of Chang et al., which defines state-of-the-art methods of Palette-based Color Transfer which has influenced research in the field for years later.

Tools

Unlike many Computer Graphics and Computer Visualization projects, this web-based Palette editor was implemented entirely using JavaScript. To achieve the many linear algebra and other arithmetic calculations in this model, I employed the use of multiple JavaScript tools found on the web. The color picking interface was implemented using the colorPicker project of GitHub user PitPik[[1]](#footnote-1). Mathematical operations were performed using the math.js library[[2]](#footnote-2). K-Means clustering was performed using the clusterfck.js by GitHub user harthur[[3]](#footnote-3). The Radial Basis Function used to represent pixels as a combination of palette-entries uses rbf.js by GitHub user thibauts[[4]](#footnote-4).

For the purposes of the web interface, the user can select from a preset list of images. This is simply to allow users an easily selectable list of images to use when testing the model. The model is designed to work with any image that fits the canvas size, which in this case uses square images. The web interface allows for an intuitive way to transfer colors between images, as well as providing a challenge to implement the system completely in JavaScript.

User Interface

The user interface for this tool is simple, as I aimed to make an intuitive interface to allow as simple interaction between the user and the tool as I could allow. There is a dropdown menu provided at the top of the interface to select the image which the user would like to perform Color Transfer on. To save the technical debt of implementing image uploading, the web interface uses a pre-determined list of images.

To calculate the palette of the selected image, the user can select the “Calculate Palette” button. Before doing so, the user can select a palette size between 3 and 6. This dynamically calculates the image palette of the specified size from the input image in a process explained in the next section. To edit the palette, the user may click on the palette entry and open a color picker interface. The user then selects any desire color which will now be reflected as the palette. Once the user has selected their new palette, the user can select the “Transfer Colors” button and wait for the output image to be generated.

Automatic Palette Extraction

Automatically extracting the palette from an image is achieved using a modified version of the K-Means Clustering algorithm. The goal is to select k colors that are representative of the colors in the original image. The choice of k is selected by the user, with options ranging between 3 and 6 colors.

A simple implementation of this algorithm would be to take all the pixels in the image, use them as input into the K-Means algorithm and select the k resulting centroids. This, however, is computationally expensive and may result in colors that are like one another, without accounting for variability in the resulting palette.

In this implementation, pixels are assigned to a bin as a 3-dimensional histogram of size 16 x 16 x 16. Each dimension of the histogram represents a color channel in RGB space. Once all pixels have been assigned a bin, the mean color for each bin is computed in Lab color space. The means collected here will be the points used for calculating the K-Means centroids.

When using this method, the randomized initial centers will still result in a different palette generated for each calculation. Thus, to initialize centers optimally, the algorithm will find the k largest bins and use those values as the initial centers. This results in uniform palette output, as well as representative palette entries for the colors found most within the image.

Another deviation from simple K-Means clustering is that generally, the resulting palette will produce near-black colors, since images often have many dark colors. Changes in color are hard to distinguish qualitatively, thus making very dark palette entries less useful to the user in distinguishing the colors of an image. As such, k+1 means are calculated, with the extra center initialized around the color black. This forces an extra cluster to be centered around black, then the resulting palette entry can be ignored. Thus, a k-sized palette can be generated without very dark colors providing noise to the algorithm.

Luminance Mapping

Mapping luminance of a color to a natural value was a crucial part of making the output image look natural and true to the original, while still adhering to the updated palette. A method described in the paper by Chang et al. allows output luminance to be computed based on the luminance of the output palette. This function would gather the nearest 2 nearest colors to a pixel in the new palette. Then, the weighted combination of the luminance of these two colors would result in the output luminance for that pixel.

Perhaps due to the vague explanation of this luminance transfer function, I was unable to reproduce this function in a way that would result in a natural image. In each iteration of the function, the output luminance would be noticeably similar between pixels, losing a lot of the texture in the image. As such, I devised a different way to handle luminance.

In order to maintain the depth of the original image, all calculations performed on the pixel assumes the original pixel’s luminance. This not only allows for the image to maintain its original depth, but it also allows the user to enter any color value for the output palette. In the paper by Chang et al., the UI restricted the user to stay within a luminance range between of the current palette color and of the color with the next highest luminance. By assuming to keep the original luminance value, all colors are mapped to the luminance that best represents the semantics of the scene, as well.

This treatment of luminance also benefits colors which are unaffected by the change in palette. Using the more complex luminance transfer function, colors unaffected by the palette change would also see a change in luminance, due to the weight between closest palette entries resulting in a different luminance from the original value. Thus, this method results in unchanged colors to be unaffected, further integrating the changed colors into the original scene semantics.

Color Transfer

Transferring colors between palettes requires a combination of the luminance mapping described above and a function that changes the hue of the pixel values. This calculation takes only into account the a and b values within Lab color space, as the luminance component is determined via the mapping of luminance described above.

To explain this transfer function, we can first view the function within the scope of a single color, . Assume the original palette color is and the new value of the corresponding palette color is . Let us also say that the pixel is and the desired output for the pixel will be . The simplest computation is to apply the distance between and to all values of . This, however, can cause the color to go out of the lab color space gamut, thus resulting in invalid values. Snapping these values to the closest in-gamut value, which a naïve version of my current model once did, would result in rough transitions between colors in the image. As such, two cases of color transfer need to be considered.

One case is the “far” case, in which simply applying is in gamut. If so, is calculated as the point where a parallel ray intersects with the gamut boundary. In the “near” case, this value results in an out of gamut color. Thus, is instead calculated as the boundary intersect for a ray going from to . Using these values, the resulting is calculated using the following formulation:

This calculation will cause in both cases to be within gamut. In this case, as opposed to clamping to the closest in gamut value, the colors will smoothly transition to each other. This computation further blends color changes into the semantics of a scene.

Now that a single color can be transferred between palette values, we can apply these changes across a full palette using the general formula. Let’s say our previous function is now , with representing the index of a palette color. We can now define k transfer functions and blend them by weight. Essentially, each pixel is a combination of palette colors with a weight coefficient. The transfer function now looks as follows:

The weights are calculated using radial basis functions representing weights for each color in the palette. The radial basis function is performed using a Gaussian kernel. As described in Chang et al.’s study, this kernel results in a smooth and accurate representation of the colors’ make-up. As a means of accelerating image computation, the radial basis functions are always calculated at the time of palette generation.

HTML Integration

Producing the results in HTML requires the use of a canvas element. This allows, in JavaScript, to extract the image data of an image represented within the canvas. This is what is then used to retrieve an array of pixels in the image and their associated RGB values. Once the computations are complete, the new pixel values can be placed onto the output canvas and compose the same image, but in the user-specified palette. From here, the user can save the newly created image to their own computer straight from the canvas element.

Results

The resulting images display the possibilities for Palette-based Color Transfer using the web interface’s provided sample images. Qualitatively, I have observed great improvements throughout the development process between iterations of the color transfer tool. Previous versions included the previously mentioned luminance function, based on the luminance of palette entries. This resulted in output images that appeared vastly different from the input image. Another iteration handled out of gamut colors by snapping them to the nearest in gamut value. Though the colors mostly resembled the intended target color, this resulted in blocky elements of the image where colors appeared to be the same in places where more variety and texture should occur. This final model uses simple luminosity mapping to best convey the original image semantics, while also giving the full range of the color picker as potential user-selected values. It also employs the full functionality of the transfer function defined in Chang et al.’s study, which allows the full depth of the image to be conveyed in places where the color transfer does not map with the palette difference as a one-to-one relationship.

A picture containing photo, showing, colorful, different

Description generated with very high confidence

**Figure 1: *Examples from Palette-based Color Transfer Web Interface.***

Conclusion

This Palette-based Color Transfer model provides inexperienced users with an easy-to-use interface for editing the colors of an image. With a click of a button, the user can extract a representative palette from an image and edit the produced colors to match any color of their choice. The tool then automatically converts the colors correlated with the original palette into the new palette in a way that stays true to image composition and luminance. Limitations of this model concern image semantics, as there are no restrictions on what colors in the image are transferred, even if it does not make sense for the environment. Potential future work can rectify this by allowing the user to supply restricted colors that cannot be changed. In addition, there are methods of accelerating the color transfer process that I have yet to explore, which I would like to expand upon if given the chance to work further on this project. The tool could also be expanded for multiple methods of color transfer, if given the opportunity to expand my work on this project.

References

Daniel Cohen-Or, Olga Sorkine, Ran Gal, Tommer Leyvand, Ying-Qing Xu. 2006. [Color Harmonization](https://dl.acm.org/citation.cfm?id=1141933). *ACM Transactions on Graphics, Volume 25.*

Elad Aharoni-Mack, Yakov Shambik, and Dani Lischinski. 2017. [Pigment-Based Recoloring of Watercolor Paintings](https://dl.acm.org/citation.cfm?id=3092926%27). *Proceedings of the Symposium on Non-Photorealistic Animation and Rendering*.

H.B. Kekre and Sudeep D. Thepade. 2008. [Color Traits Transfer to Grayscale Images.](https://ieeexplore.ieee.org/abstract/document/4579871/) *Emerging Trends in Engineering and Technology, 2008*.

Huiwen Chang, Ohad Fried, Yiming Liu, Stephen DiVerdi, and Adam Finkelstein. 2015. [Palette-based Photo Recoloring](https://dl.acm.org/citation.cfm?doid=2766978). *ACM Transactions on Graphics, Volume 34*.

Qing Luan, Fang Wen, Daniel Cohen-Or, Lin Liang, Ying-Qing Xu, and Heung-Yeung Shum. 2007. [Natural Image Colorization](https://dl.acm.org/citation.cfm?id=2383887). *Eurographics Symposium on Rendering*.

Qing Zhang, Chunxia Xiao, Hanqiu Sun, and Feng Tang. 2017. [Palette-Based Image Recoloring Using Color Decomposition Optimization](https://ieeexplore.ieee.org/abstract/document/7859399/). *IEEE Transactions on Image Processing, Volume 26*.

Xuezhong Xiao and Lizhuang Ma. 2006. [Color Transfer in Correlated Color Space](https://dl.acm.org/citation.cfm?id=1128974). *Proceedings of the 2006 ACM International Conference on Virtual Reality Continuum and its Applications*.

1. <https://github.com/PitPik/colorPicker> [↑](#footnote-ref-1)
2. <http://mathjs.org/download.html> [↑](#footnote-ref-2)
3. <https://github.com/harthur/clusterfck> [↑](#footnote-ref-3)
4. <https://github.com/thibauts/rbf> [↑](#footnote-ref-4)