

Color2Gray: Saliency-Preserving Color Removal

Amy A. Gooch

Sven C. Olsen

Jack Tumblin

Bruce Gooch

Northwestern University *



Figure 1: A color image (Left) often reveals important visual details missing from a luminance-only image (Middle). Our Color2Gray algorithm (Right) maps visible color changes to grayscale changes. *Image: Impressionist Sunrise by Claude Monet, courtesy of Artcyclopedia.com.*

Abstract

Visually important image features often disappear when color images are converted to grayscale. The algorithm introduced here reduces such losses by attempting to preserve the salient features of the color image. The *Color2Gray* algorithm is a 3-step process: 1) convert RGB inputs to a perceptually uniform CIE $L^*a^*b^*$ color space, 2) use chrominance and luminance differences to create grayscale target differences between nearby image pixels, and 3) solve an optimization problem designed to selectively modulate the grayscale representation as a function of the chroma variation of the source image. The *Color2Gray* results offer viewers salient information missing from previous grayscale image creation methods.

CR Categories: I.4.3 [Image Processing and Computer Vision]: Enhancement—Grayscale manipulations I.4.10 [Image Processing and Computer Vision]: Image Representations—Multidimensional

Keywords: non-photorealistic, image processing, color reduction, perceptually-based rendering

*<http://www.color2gray.info>



Figure 2: Isoluminant changes are not preserved with traditional color to grayscale conversion. Converting an image of a reddish square whose luminance matches that of the blue background (Left) to grayscale (Middle) results in a featureless gray image. The Color2Gray algorithm incorporates chrominance changes (Right).

1 Introduction

If digital images are regarded solely as an optical record, then a grayscale image only needs to record light intensities using a flat spectral response. Current color to grayscale conversions already meet this goal. However, as viewers, we often expect a more ambitious result: we want digital images to preserve a meaningful visual experience, even in grayscale. We are less concerned with the accuracy of light intensities and more concerned with the preservation of visual cues that help us detect the most important, or salient, scene features. Accordingly, a black-and-white line drawing is sometimes far more expressive than a color photograph, and a garish cartoon-shaded rendering can often make important features, such as the shape, position, and reflectance of an object, more apparent.

Color documents printed in grayscale are often indecipherable. Figures and graphs in papers with saturated colors look good printed in color, but when printed in grayscale, a “red line” may appear to be the same shade of gray as a “green line”. Grayscale mappings of color images that are constructed solely by approximating spectral uniformity are often woefully inadequate because isoluminant visual cues signaled only by chromatic differences are lost [Livingstone 2002], such as the reflection of the sun in Figure 1.

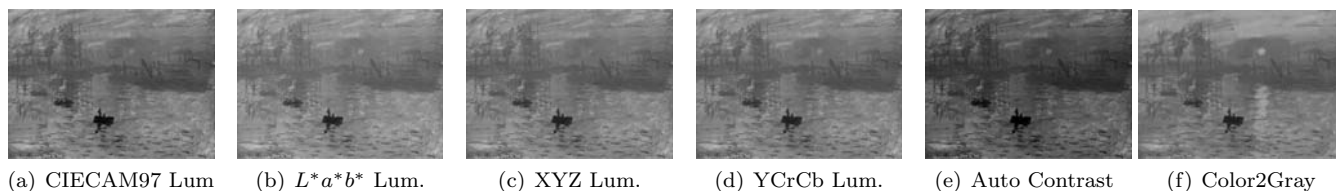


Figure 3: Comparison of grayscale conversions by previous methods (a-d), Photoshop’s grayscale mode with auto contrast (e), and our Color2Gray algorithm (f).



Figure 4: Converting the source image (Left) to grayscale with Photoshop’s grayscale mode results in a nearly featureless image (Middle). The Color2Gray algorithm creates circles that are distinct from the background (Right). Parameters: $\theta = 180^\circ$; $\alpha = 8$; $\mu =$ entire image.

Vision scientists hypothesize that the human visual system does not perceive absolute values, and instead chrominance and luminance perception are based upon relative assessments, in part due to the center-surround organization of cells in the early stages of visual processing [Lotto and Purves 1999]. We propose that preserving relationships between nearby pixels in an image is much more important than representing absolute pixel values. Our work contributes to the growing trend in computer graphics of using change-based mappings for image representation and manipulation [Fattal et al. 2002; Pérez et al. 2003; Levin et al. 2004]. The technical contribution of this work is a color to grayscale image conversion algorithm based upon the human visual system’s sensitivity to change. Additionally, we provide a new signed chrominance distance calculation in the CIE $L^*a^*b^*$ chrominance plane. The Color2Gray algorithm creates images that maintain the salience of color images by mapping chrominance and luminance changes in a source image to changes in a grayscale image.

Finding the most appropriate method to depict color differences in grayscale is also central to this research. For example, there are no luminance changes in the color image in Figure 2; adding them might seem at first to be a distortion of the original grayscale values. However, We claim that all visible color changes should cause visible changes if the grayscale image is to convey a more effective visual experience in comparison to current methods.

2 Related Work

Previous methods for converting RGB images to grayscale employed dot products or weighted sums to map a three dimensional color space to a single dimension, as shown in Figure 3. Other methods adopted by programs like Adobe Photoshop [Volk 2000; Brown 2004; Adobe Photoshop 2004], devised custom non-linear projections and required users to set image-dependent parameters by trial and error. These methods applied a fixed function to map a set of 2D manifolds in color space to a set of 1D points (luminance) and were ineffective at preserving chrominance differences between isoluminant pixels (Figure 4). Contrast enhancement techniques increased the dynamic range of the image, map-

ping isoluminant pixels with different chrominance to the same gray value (Figure 3e).

Converting a color image to grayscale is a dimensionality reduction problem. In the course of our research, we have explored both linear and non-linear dimensionality reduction techniques, such as principal component analysis (PCA) and space-filling curves. PCA can be employed to compute an ellipsoid in color space that is a least-squares best fit for the cloud of points formed by all the image color values. Color values in the image can then be projected on a luminance axis defined by the primary axis of this ellipsoid. The effectiveness of PCA depended upon the color space; we found that the larger the ratio between the primary and secondary axes, the more likely the technique created distinctive gray values for different colors. For example, color space plots for a CIE $L^*a^*b^*$ image appeared twisted and stretched in RGB space and yielded different principal component axes and different grayscale mappings. We also explored a non-linear dimensionality reduction technique, similar to Teschioni et al. [1997], which created clusters in the color space and applied space-filling curves to build a 1D parameterization of 3D color space. Both space-filling curves and PCA often created results with a high dynamic range because the length of any 1D pathway that adequately fills a 2D plane or 3D volume quickly approaches infinity.

We also experimented with Poisson solvers [Fattal et al. 2002] and discovered that they work well on images such as the sunrise image (Figure 1), but not on images with large disconnected isoluminant regions (Figure 4) because Poisson solvers compute gradients over nearest neighbors, ignoring difference comparisons over distances greater than one pixel.

The color to gray problem is also similar to color quantization [Heckbert 1982] and compression of gamut dimensionality. The research by Power et al. [1996] created an image with a reduced set of inks or colors. However, both color quantization and gamut compression typically preserved luminance contrast and ignored contributions from chrominance for single ink/color images.

Contemporaneous research by Rasche et al. [2005a; 2005b] on the color to gray problem maintained the proportionality between perceived color difference and perceived luminance difference and ignored spatial arrangement of pixels. They also provided an extension to reduce three dimensional color space to a two dimensional surface to render images as they would appear to color-deficient observers. Our solution incorporates changes from luminance, chrominance, and distance combined as well as provides three simple parameters to enable users to create aesthetic and perceptually salient grayscale images.

3 Algorithm

The Color2Gray algorithm has three steps: first, we convert a color image to a perceptually uniform color space,

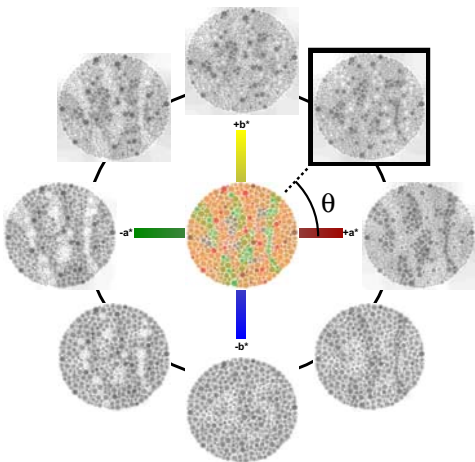


Figure 5: Color2Gray results for the color image in the center of the figure over several values of the parameter θ , which divides the chrominance plane. Note, for this colorblind image, you should not be able to see the 45 made of dots in the source color image unless you are colorblind. *Original image copyright Jay Neitz.*

then compute target differences in order to combine luminance and chrominance differences, and finally, we use a least squares optimization to selectively modulate the source luminance differences in order to reflect changes in the source image’s chrominance.

We express the color differences between pixels in the color image as a set of signed scalar values, and then build a grayscale version of the image with those values. For each pixel i and neighbor pixel j , we find a signed distance scalar, δ_{ij} , based upon luminance and chrominance differences between i and j . Using these values we then derive values for the grayscale image, g . We refer to the grayscale difference between pixel i and a neighboring pixel j as $(g_i - g_j)$. A successful color to gray conversion consists of finding g such that all $(g_i - g_j)$ values closely match the corresponding δ_{ij} values.

Specifying δ_{ij} is fairly involved, and can benefit from minimal amounts of user interaction (Section 3.1). Once we formalize δ_{ij} (Section 3.2), we find the output image g by an iterative optimization process (Section 3.3).

3.1 Parameters

The Color2Gray algorithm encodes differences from a color image into luminance differences in a grayscale image. However, generating a satisfying result sometimes requires making aesthetic decisions, such as the choice to make the orange sun brighter, not darker, than its isoluminant surrounding. Thus some level of user control is desirable. Previous interactive techniques required users to perform region-based adjustments, modify color channels or individually alter pixels. The Color2Gray algorithm allows users to control the mapping of color differences to grayscale differences via three simple parameters:

- θ : controls whether chromatic differences are mapped to increases or decreases in luminance value (Figure 5).
- α : determines how much chromatic variation is allowed to change the source luminance value (Figure 6).
- μ : sets the neighborhood size used for chrominance estimation and luminance gradients (Figure 7).



Figure 6: Changing the α parameter: $\alpha = 5, 10, 25$, respectively. Increasing alpha increases the contribution from chrominance differences but may cause dynamic range mapping problems ($\theta = 45^\circ$).



Figure 7: *Left*: Equiluminant fade from gray to blue on a background with the same luminance. *Middle*: Color2Gray result with $\mu = 9$ neighborhood. *Right*: Color2Gray result with a full neighborhood ($\theta = 270^\circ$; $\alpha = 8$). Using a small neighborhood produces a dark band through the middle of the fade and produces artifacts where the edge of the fade meets the background; whereas using the entire set of pixels as the neighborhood (full neighborhood) preserves the horizontal fade with respect to its background.

The angle θ divides the chrominance plane and determines whether a chromatic difference will darken or lighten the source luminance difference. To further understand the intuition behind θ , consider mapping the chrominance changes in Figure 1 to shades of gray. The sun and its reflection would benefit aesthetically from mapping the oranges to brighter gray values (making the sun lighter) and mapping the blues to darker gray values. Such a cool to warm mapping corresponds to $\theta = 45^\circ$. Though we have implemented tools for automatically selecting θ , we make the parameter available to the user. As shown by applying the Color2Gray algorithm to the color blindness test image of Figure 5 varying θ may provide insight into the salient cues available in the image.

The second parameter of the algorithm, α controls the amount of chromatic variation applied to the source luminance values. The luminance values, $L^* \in [0, 100]$, are in a much smaller range than either of the chrominance axes; a^* is in the range $[-500, 500]$ and b^* is in the range $[-200, 200]$. The parameter α sets an upper and lower bound for how much of an effect a large chrominance difference can have on the current luminance value for a given pixel, such that the shift due to the chrominance values is within $[-\alpha, \alpha]$. In the next section, we define a function `crunch()` that allows us to compress large-amplitude values into the valid range, while leaving small yet significant variations around zero almost untouched. By default, $\alpha = 10$. However, some images may allow for more chrominance adjustment, such as the colorblindness test image in Figure 9, created with $\alpha = 15$. Figure 6 shows the effect of varying α .

The third parameter of the algorithm, μ , controls the size of the neighborhood, indicating whether a user is more interested in preserving local or global changes. Smaller neighborhoods may result in non-neighboring isoluminant regions with different chrominance values in the source image to be represented with the same grayscale value in the output image g . Additionally, preserving only local changes can create gradients that are not present in the source image, as shown in Figure 7. All of the figures in this paper were generated

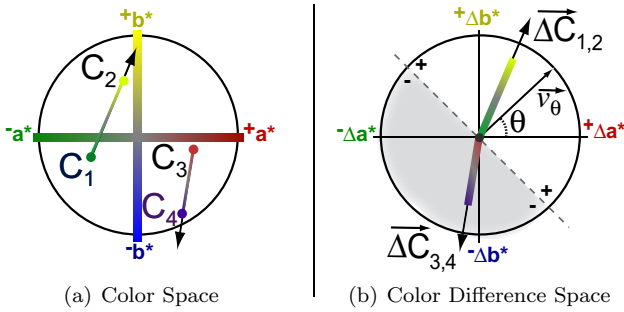


Figure 8: Signed chrominance distance between pixels. Given (a) pixel pairs (i.e. $(C_i, C_j) = (C_1, C_2)$ or (C_3, C_4)), and (b) their chromatic difference, $\overrightarrow{\Delta C}_{ij}$, we divide the space of color differences into positive and negative halves based upon the parameter θ . We set the sign of the chromatic difference to be the same as the sign of $(\overrightarrow{\Delta C}_{ij} \cdot \vec{v}_\theta)$, where \vec{v}_θ is a normalized vector defined by θ relative to the Δa^* axis, as illustrated in (b).

using the entire image for neighborhood pixel comparisons, unless otherwise noted.

3.2 Computing Target Differences

We begin by converting the color input image to CIE $L^*a^*b^*$ color space, because its Euclidean distances (L2 norm) closely correspond to perceptual dissimilarity [Wyszecki and Stiles 2000]. L_i refers to the luminance of the i^{th} pixel, and ΔL_{ij} is shorthand for $(L_i - L_j)$. Similarly, ΔA_{ij} , and ΔB_{ij} refer to the a^* and b^* channel differences between a pair of pixels. $\overrightarrow{\Delta C}_{ij}$ refers to $(\Delta A_{ij}, \Delta B_{ij})$, the chromatic difference vector between pixel i and its neighbor pixel j .

In order to determine the target difference, δ_{ij} , we compare the luminance difference, ΔL_{ij} , with the chrominance difference, $\overrightarrow{\Delta C}_{ij}$. Since ΔL_{ij} is a scalar and $\overrightarrow{\Delta C}_{ij}$ is a 2D vector, we first map $\overrightarrow{\Delta C}_{ij}$ onto a single dimension using the Euclidean norm, $\|\overrightarrow{\Delta C}_{ij}\|$. Next we need to choose the appropriate sign for $\|\overrightarrow{\Delta C}_{ij}\|$ because it is always positive and δ_{ij} is a signed scalar. Therefore, we introduce an angle θ , which parameterizes the $\Delta a^* \Delta b^*$ chrominance plane. Let \vec{v}_θ be a normalized vector defined by θ relative to the a^* axis. We set the sign of the chromatic difference to be the same as the sign of $(\overrightarrow{\Delta C}_{ij} \cdot \vec{v}_\theta)$, as illustrated in Figure 8. The parameter θ controls which chromatic differences are mapped to increases or decreases to the source luminance value (Figure 5), thus specifying the transformation of the $\|\overrightarrow{\Delta C}_{ij}\|$ s into a signed quantity.

Finally, we define the target differences, δ_{ij} . If the absolute luminance difference is smaller than the chrominance difference, then we set δ_{ij} to be a measure of the chromatic differences; otherwise, δ_{ij} is set to the luminance differences. Formally, δ_{ij} is defined as follows.

$$\begin{aligned} \text{Given:} \quad \text{crunch}(x) &= \alpha * \tanh(x/\alpha) \\ \vec{v}_\theta &= (\cos \theta, \sin \theta) \end{aligned}$$

then:

$$\delta(\alpha, \theta)_{ij} = \begin{cases} \Delta L_{ij} & \text{if } |\Delta L_{ij}| > \text{crunch}(\|\overrightarrow{\Delta C}_{ij}\|) \\ \text{crunch}(\|\overrightarrow{\Delta C}_{ij}\|) & \text{if } \overrightarrow{\Delta C}_{ij} \cdot \vec{v}_\theta \geq 0 \\ \text{crunch}(-\|\overrightarrow{\Delta C}_{ij}\|) & \text{otherwise.} \end{cases}$$

We tried several methods for combining ΔC_{ij} and ΔL_{ij} in order to define δ_{ij} , and of those that we considered, the discontinuous case above worked best. Using a linear or smooth blend between ΔC_{ij} and ΔL_{ij} can make the resulting images muddier because when the two terms have opposite signs, the differences cancel out.

We can automatically determine θ by examining the spread of the chrominance values of the source image and finding an axis through the a^*b^* plane that divides these chrominances with the largest linear separation [Healey and Enns 1996]. If the images do not have a single best axis, such as images with a large range of chrominance values, we utilize a cool-warm parameterization with $\theta = 45^\circ$. However, for some images and certain choices of θ , there may be troublesome discontinuities in the calculated signed color differences which will push the optimization in contradicting directions. We are currently investigating adaptive color difference signing and unsigned methods that would avoid these difficulties. However, in practice, these discontinuities have not produced noticeable artifacts.

3.3 Solving the Optimization

Given a set of desired signed differences δ_{ij} , we find a gray image g that minimizes the following objective function, $f(g)$, where \mathcal{K} is a set of ordered pixel pairs (i, j) :

$$f(g) = \sum_{(i,j) \in \mathcal{K}} ((g_i - g_j) - \delta_{ij})^2 \quad (1)$$

We initialize g to be the luminance channel of the source image, and then descend to a minimum using conjugate gradient iterations [Shewchuk 1994]. Equation 1 is convex, but has multiple global minima, as for any grayscale image g , an image with an identical optimization score can be created by shifting all pixel values up or down by a constant value ($f(g) = f(g + c)$). In order to choose a single solution from the infinite set of optimal g , we shift the g vector returned by our solver such that it becomes as close as possible to the source luminance values (where ‘‘closeness’’ is defined using sum squared differences).

4 Performance

The cost of setting up and solving the optimization problem is proportional to the size of \mathcal{K} . By default our algorithm compares every pixel to every other pixel, i.e. \mathcal{K} contains all ordered pixel pairs. Given such a \mathcal{K} , the algorithm scales poorly. However, one can also use a reduced set, such as that implied by only considering pairs which cohabitate a $\mu \times \mu$ neighborhood. For a square $S \times S$ image, the cost of generating all pixel pairs will be $O(\mu^2 S^2)$, or $O(S^4)$ for the full neighborhood case. Using an Althon™ 64 3200+ processor, processing images using full neighborhoods requires 12.7 seconds for a 100x100 image, 65.6 seconds for a 150x150 image, and 204.0 seconds for a 200x200 image, just as you would expect for a $O(S^4)$ algorithm.

However, these runtimes can be improved considerably by using programmable graphics hardware to perform the pixel comparisons in parallel. In theory, the ability to process an $S \times S$ block of values simultaneously should increase the performance of the algorithm from $O(S^4)$ to $O(S^2)$. In practice, the performance of our GPU implementation is closer to $O(S^3)$, requiring 2.8 seconds to solve a 100x100 image, 9.7 seconds for a 150x150 image, and 25.7 seconds for a 200x200 image, using an NVIDIA Geforce™ GT6800 graphics card.

Source code for our Color2Gray algorithm implementation, as well as additional example images, can be found at <http://www.color2gray.info/>.

5 Results and Discussion

Figure 9 compares the Color2Gray algorithm to Photoshop’s grayscale mode on a wide variety of images. Isoluminant colors for state parks and water in the image in Row 2 make the island disappear in the Photoshop result. The effects of the Color2Gray algorithm are sometimes subtle, but color changes visibly darkened the region behind the head of the butterfly (Row 1) and darkened the green block while lightening the yellow block in the Cornell Box image (Row 4).

Our algorithm generally does not perform better than the traditional grayscale representation when trying to map widely spread chrominance gradations to something independently distinguishable. The first row of Figure 10 illustrates this limitation. The image contains such a wide variety of isoluminant colors that no single span of 255 gray values can express them all separately. Additionally, the colored graphs in Figure 8 are a particularly difficult scenario for the Color2Gray algorithm: it is impossible to find a θ which creates distinct shades of gray for all of the axes since they are approximately equally spread over the chrominance plane.

The Color2Gray algorithm does not provide large improvements for scenes with high dynamic range, especially natural scenes, which typically have a wide range of luminance changes. However, the method does improve any image that contains large isoluminant regions with a small number of different chrominance values such as the last row of Figure 9. Here, Color2Gray makes the two sets of hills behind the small car clearly distinguishable.

5.1 Augmented Color Images

We found that “Color2Gray result plus chrominance” images aid in making color images more easily comprehensible. We simply add the source image’s chrominance channels to the Color2Gray result in CIE $L^*a^*b^*$ color space and then convert the image to RGB. This technique modifies the color images only slightly but reveals Color2Gray results in a grayscale-only printout. For example, we may be able to create a document in which all of the figures will print legibly in both color and grayscale. Adding color to the Color2Gray result also provides useful feedback for visualizing the subtle adjustments made with the Color2Gray algorithm, as illustrated in Figure 11.

6 Conclusion and Future Work

We are exploring extensions to the algorithm which would remove the need to specify θ , for example, using an optimization function designed to match both signed and unsigned difference terms. We are also interested in investigating multi-scale methods which could allow for faster solutions on higher resolution images.

We believe better target differences, δ_{ij} , should include image complexity measures as well. For example, although human easily compare widely separated luminance and chrominance values in simple images (e.g. Figure 4, with colored circles on gray), comparisons in complex images over these same distances are far more difficult (e.g. Figure 1). In

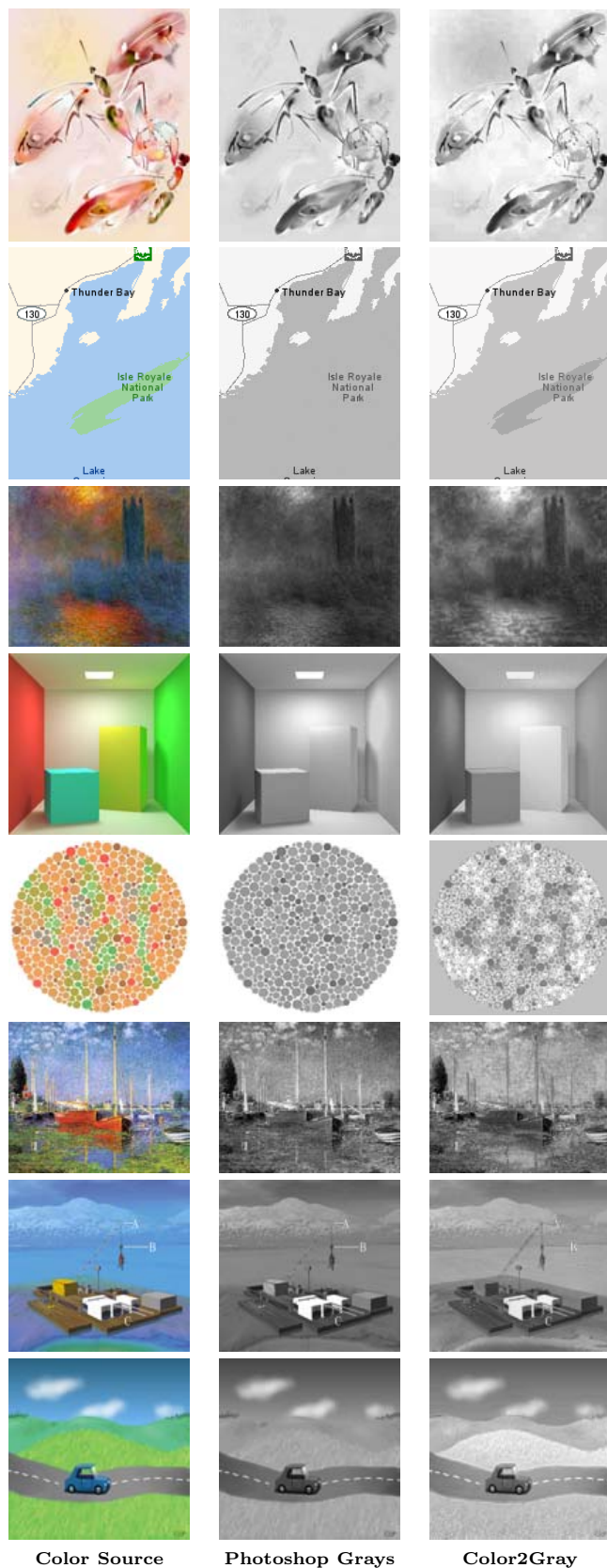


Figure 9: Comparison of Color Source image, Photoshop Grayscale, and Color2Gray Results. *Source images courtesy of Ana Vasileva, Yahoo!/NAVTEQ, Claude Monet, Dani Lischinski, Jay Neitz, Paul Chapman, and Leon Bli, respectively.*

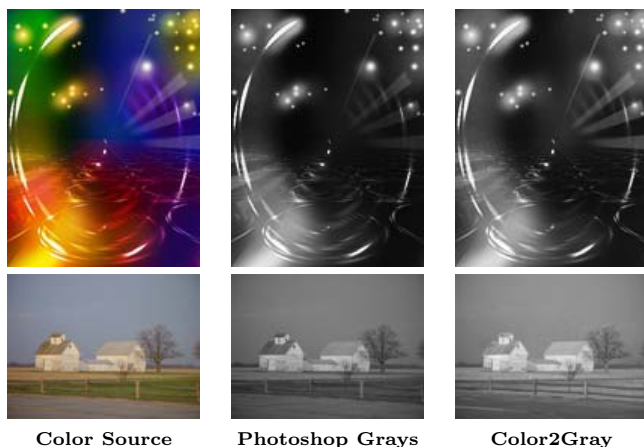


Figure 10: Images with a wide range of colors (Top) or with subtle changes (Bottom) result in Color2Gray results that do not provide large improvements over Photoshop grayscale. Top source image courtesy of Ana Vasileva.

complex images, reuse of the same grayscale values for different nearby colors becomes more acceptable. We plan to conduct perceptual experiment to understand the threshold for this difference as well as validating the degree to which our algorithm preserves the saliency of the color image.

The Color2Gray algorithm produces grayscale versions of color images, minimizing isoluminant mappings. While Color2Gray cannot improve all source images, we believe that its results will not be less perceptually salient than the source luminance values alone. We encourage readers to print this paper on a high quality grayscale printer and examine their printed versions of the original color, Color2Gray, and Color2Gray enhanced images.

7 Acknowledgments

We thank Rachel Gold, the anonymous reviewers, and many more for helping us iron out the kinks; Holger Winnemoeller for help with ISOMAP and SOR solvers; Ben Watson for pointing us to PCA; Cindy Grimm, Raquel Bujans, and Rennie Bailey, who gave us DKL Color Space pointers and code; and Pat Hanrahan, helped with pointers to CIECAM'97. Thanks to Vidya Setlur and the rest of the Northwestern University Computer Graphics Group for endless support, edits and feedback. Thanks to MidGraph participants for useful discussions and to Feng Lui for the implementation of a contrast attention model. This material is based upon work supported by the National Science Foundation under Grant No. 0415083. Additionally, this research was generously supported by the Helen and Robert J. Piros Fellowship. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsors.

References

- ADOBE PHOTOSHOP, 2004. Photoshop: Converting color images to black and white. <http://www.adobe.com/tips/pbs8colorbw/main.html>.
- BROWN, R., 2004. Photoshop tips: Seeing in black & white. http://www.russellbrown.com/tips_tech.html.

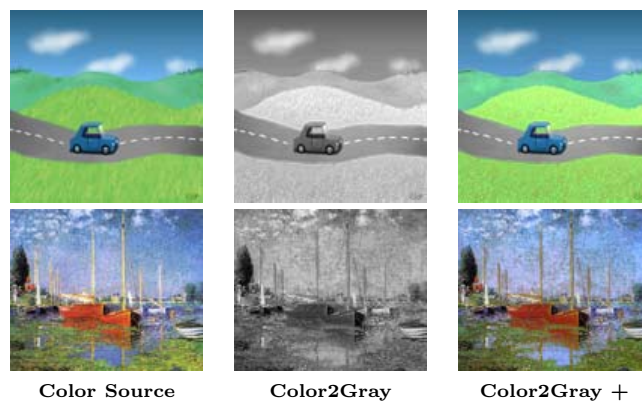


Figure 11: Color2gray results combined with chrominance (a^*b^*) from the source image make color difference more visible, and even grayscale printing reveals the improved Color2Gray result. Source images courtesy of Leon Bli and Claude Monet respectively.

- FATTAL, R., LISCHINSKI, D., AND WERMAN, M. 2002. Gradient domain high dynamic range compression. *ACM Transactions on Graphics* 21, 3 (July), 249–256.
- HEALEY, C. G., AND ENNS, J. T. 1996. A perceptual colour segmentation algorithm. Tech. Rep. TR-96-09, UBC CS.
- HECKBERT, P. S. 1982. Color Image Quantization for Frame Buffer Display. In *Proceedings of SIGGRAPH*, 297–307.
- LEVIN, A., LISCHINSKI, D., AND WEISS, Y. 2004. Colorization using optimization. In *ACM Transactions on Graphics*.
- LIVINGSTONE, M., Ed. 2002. *Vision and Art: The Biology of Seeing*. Harry N. Abrams, Inc., Publishers.
- LOTTO, R. B., AND PURVES, D. 1999. The effects of color on brightness. *Nature Neurosci.* 2, 11, 1010–1014.
- PÉREZ, P., GANGNET, M., AND BLAKE, A. 2003. Poisson image editing. *ACM Transactions on Graphics* 22, 3 (July), 313–318.
- POWER, J. L., WEST, B. S., STOLLNITZ, E. J., AND SALESIN, D. H. 1996. Reproducing color images as duotones. In *Proceedings of SIGGRAPH*, ACM Press, 237–248.
- RASCHE, K., GEIST, R., AND WESTALL, J. 2005. Detail preserving reproduction of color images for monochromats and dichromats. *IEEE Comput. Graph. Appl.* 25, 3.
- RASCHE, K., GEIST, R., AND WESTALL, J. 2005. Re-coloring images for gamuts of lower dimension. *Eurographics/Computer Graphics Forum* 24, 3.
- SHEWCHUK, J. R. 1994. An introduction to the conjugate gradient method without the agonizing pain. Tech. rep., Pittsburgh, PA.
- TESCHIONI, A., REGAZZONI, C. S., AND STRINGA, E. 1997. A Markovian approach to color image restoration based on space filling curves. In *International Conference on Image Processing Proceedings*, vol. 2, 462 – 465.
- VOLK, C., 2000. Adobe Photoshop Tip of the Week Tutorial. <http://www.carlvolk.com/photoshop21.htm>.
- WYSZECKI, G., AND STILES, W. S., Eds. 2000. *Color Science: Concepts and Methods, Quantitative Data and Formulae*. Wiley-Interscience.