Artistic Recoloring of Image Oversegmentations

Figure 1: Recolorization of region-based abstraction. The figure shows the original image on the left and the recolored images produced by our proposed method using three different palettes.

Abstract

We propose a method to assign vivid colors to regions of an oversegmented image. We restrict the output colors to those found in an input palette, and seek to preserve the recognizability of structure in the image. Our strategy is to match the color distances between the colors of adjacent regions with the color differences between the assigned palette colors; thus, assigned colors may be very far from the original colors, but both large local differences (edges) and small ones (uniform areas) are maintained. We use the widest path algorithm on a graph-based structure to set a priority order for recoloring regions, and traverse the resulting tree to assign colors. Our method produces vivid colorizations of region-based abstraction using arbitrary palettes. We demonstrate a set of stylizations that can be generated by our algorithm.

Keywords: Non-photorealistic rendering, Image stylization, Recoloring, Abstraction.

Index Terms: I.3.3 [Picture/Image Generation]—; I.4.6 [Segmentation]

1 Introduction

Color plays an important role in image aesthetics. In representational art, artists employ colors that match the perceived colors of objects in the depicted scene. Conversely, abstraction provides more freedom to manipulate colors. Figure 2 shows a modern vector illustration of an owl, an example of *Fauvism* by André Derain, and an abstraction of the Eiffel Tower by Robert Delaunay. The artists have expressed the image content by vivid colors assigned carefully various image regions. We aim at a method that can generate colorful images, recoloring an image using an arbitrary input palette. In this paper, we describe a method for recoloring an image based on a subdivision of the image into distinct segments, recoloring each segment based on its relationship with its neighbours.

Our goal is to present different recoloring possibilities by assigning colors to each region of an oversegmented input image. We aim to maintain the image contrast and preserve strong edges so that the content of the scene remains recognizable. It is important to convey textures and small features, often too delicate to be preserved by existing abstraction methods. We would like to be able to create wild and vivid abstractions through use of unusual palettes.

Manual recoloring of oversegmented images would be tedious; the images contain hundreds or thousands of segments; clicking on every segment would take a long time, even leaving aside the cognitive and interaction overhead of making selections from the palette. We provide an automatic assignment of colors to regions. The assignment can be used as is, could be used in a fast manual assessment loop (for example, if an artist wanted to choose a suitable palette for coloring a scene), or could be a good starting point for a semi-automatic approach where a user made minor modifications to the automated results.

In this paper, we present an automatic recoloring approach for a region-based abstraction. The input is a desired palette and an oversegmented image. The method assigns a color from the palette to each region; it is based on the *widest path* algorithm [17], which organizes the regions into a tree based on the weight of the edges connecting them. We use color differences between adjacent regions both to order the regions and to select colors, trying to match the difference magnitude between the assigned palette colors and the original region colors. The use of color differences allows structures in the image to remain recognizable despite breaking the link between the original and depicted color.

Our main contributions are as follows:

- We designed a recoloring method for an oversegmented image, creating multiple abstractions colored with just one palette. Our method creates wild and high contrast images.
- Various styles can be created by our method. We experiment with color blending between regions and produce smooth images. By reducing the palette colors, we simplify the recolored images. Moreover, we generate new colorings from a palette by applying different metrics and color spaces.

The remainder of the paper is organized as follows. In Section 2, we briefly present related work. We describe our algorithm in Section 3. Section 4 shows results and provides some evaluation, and

Figure 2: Colorful representational images. A modern vector illustration of an owl; a Fauvist painting by André Derain; an abstraction of the Eiffel Tower by Robert Delaunay.

Section 5 gives some possible variations of the method. Finally, we conclude in Section 6 and suggest directions for future work.

2 Previous Work

Although there is an existing body of work on recoloring photographs [5, 8, 10, 12, 15, 19, 22, 26, 29] and researchers have investigated colorization of non-photorealistic renderings [2, 7, 18, 23, 25, 30, 31], there is room for further exploration. Existing recoloring methods little address region-based abstractions. The closest approach, by Xu and Kaplan [28], used optimization to assign black or white to each region of an input image. On the other hand, there are a few approaches in the pattern colorization problem, intending to colorize the graphic patterns [3, 11]. Bohra and Gandhi [3] posed the colorization problem as an optimal graph matching problem over color groups in the reference and a grayscale template image. In generating playful palettes [6], the approximation results converge for a number of blobs beyond six or seven. Similarly, the *ColorArt* method of Bohra and Gandhi [3] enforce an equal number of colors in the reference palette and the template image, and cannot find matches otherwise.

Below, we review some of the previous research on recoloring methods in NPR and color palette selection. These approaches can be broadly classified into example-based recoloring and palette-based recoloring.

Example based Recoloring (Color Transfer)

Recoloring methods were first proposed by Reinhard et al. [19], where the colors of one image were transferred to another. They converted the RGB signals to Ruderman et al.'s [20] perception-based color space $L\alpha\beta$. To produce colors they shifted and scaled the $L\alpha\beta$ space using simple statistics.

Towards the color transfer and color style transfer techniques, Neumann and Neumann [15] extracted palette colors from an arbitrary target image and applied a 3D histogram matching. They attempt to keep the original hues after style transformation, where all of the colors, having the same hue on the original image would have the same hue after the matching step. They performed matching transformations on 3D cumulative distribution functions belonging to the original and the style images. However, due to the lack of spatial information, the same histogram in itself looks to be not enough for a true style cloning. Especially, the problems caused by unpredictable noise and gradient effects. They suggest using image segmentation and 3-dimensional smoothing of the color histogram to improve the result.

Levin et al. [10] introduced an interactive colorization method for black and white images. They used a quadratic cost function derived from the color differences between a pixel and its weighted average neighborhood colors. The user by scribbling indicates the desired color in the interior of the region, instead of tracing out its precise boundary. Then the colors automatically propagate to the remaining pixels in the image sequence. However, sometimes it fails at strong edges, due to a sensitive scale parameter. Inspired by Levin et al. [10], Yatziv and Sapiro [29] proposed a fast colorization of images, which was based on the concepts of luminance-weighted chrominance blending and fast intrinsic geodesic distance computations. Sykora et al. [25] developed a tool called *Lazybrush* for the colorization of cartoons which integrates textures to the images to create 3D-like effects [24], while Casaca et al. [4] used Laplacian coordinates for image division and used a color theme for fast colorization. Fang et al. [7] proposed an interactive optimization method for colorization of the hand-drawn grayscale images. To maintain a smooth color transitions and control the color overflow in the texture areas, they used a smooth feature map to adjust the feature vectors.

A few works in NPR composite the colors for recoloring. The compositing could be applied by using alpha blending [13] or the Kubelka-Munk [9] equation (KM). For a paint-like effect NPR researchers mostly worked with the physic-based Kubelka-Munk equation to predict the reflectance of a layer of pigment. Some researchers tried to mimic the actual artist decision of mixing the colors to generate the palettes. For example, a data-driven color compositing framework by Lu et al. [12] derived three models based on optimized alpha blending, RBF interpolation and KM optimization to improve the prediction of compositing colors. Later, the KM pigment-Based model used for recoloring of styles such as watercolor painting [2]. The compositing by KM model leaves traces of overlapping stroke layers which can produce near natural painting effects.

Palette based Recoloring

The interest in colorization and recoloring methods, opened up new research ideas on color palettes, rather than simple approaches like averaging colors of the active regions [8]. For many recoloring methods, usually a user would scribble each region of the image with a color, and the algorithm would do the rest of the work. Early methods to select the color palettes used Gaussian mixture models or K-means to cluster the image pixels. Chang et al. [5] introduced a photo-recoloring method by user-modified palettes. To select the color palettes, they compute the kmeans clustering on image colors, then discard the black entry. Tan et al. [26] proposed a technique to decompose an image into layers to extract the palette colors. Each layer of decomposition represents a coat of paint of a single color applied with varying opacity throughout the image. To determine a color palette capable of reproducing the image, they analyzed the image in RGB-space geometrically in a simplified convex hull.

Most recent methods on generating the palettes have more flexibility for editing, such as Playful Palette [23], which is a set of blobs of color that blend together to create gradients and gamuts. The editable palette keeps the history of previous palettes. DiVerdi et al. [6] also proposed an approximation of image colors based on the Playful Palette. In this technique, within an optimization framework, an objective function minimizes the distance between the original image and the recolored one by palette colors, based on the self organizing map. The approximation algorithm is an order of magnitude faster than Playful Palette [23], while the quality is lower due to small amounts of shrinkage which caused by both the self organizing map and the clustering step.

There are a limited number of works in the literature that assign colors to regions. For example, Qu et al. [18] proposed a colorization technique for the black and white manga using the Gabor wavelet filter. A user scribbles on the drawing to connect the regions; the algorithm then assigns colors to different hatching patterns, halftoning, and screening. The vector arts algorithms are used to recolor the distinct regions, which are more comparable to our recoloring method. Xu and Kaplan introduced artistic thresholding [28] where a graph data structure is applied on the segmentation of a source image. They employed an energy function to measure the quality of different black and white coloring of the segments. However, they failed to show the high-level features that crossing through the foreground objects. Lin et al. [11] proposed a palette-based recoloring method with a probabilistic model. They learn and predict the distribution of properties such as saturation, lightness, and contrast for individual regions and neighboring regions. Then score pattern colorings using the predicted distributions and color compatibility model by O'Donovan et al. [16]. Bohra and Gandhi [3] proposed an exemplar-based colorization algorithm for grayscale graphic arts from a reference image based on color graph and composition matching method. They retrieve palettes using the spatial features of the input image. They aim to preserve the artist's intent in the composition of different colors and spatial adjacency between these colors in the image.

3 Recoloring Algorithm

We present a recoloring algorithm that automatically assigns colors to regions of an oversegmented image. Our system takes as input a set of regions and a palette cotaining a set of colors and assigns a color to each region. The recolored image should convey recognizable objects in the image. Edges are essential to the visibility of the structures. Neighboring regions will be assigned distinct colors to express an edge, and regions of similar colors will be assigned similar colors. The human visual system is sensitive to brightness contrast; to help preserve contrast in our recoloring, we take into account the regions' relative luminance changes when selecting region colors.

We take the strategy of emphasizing color differences between regions, seeking to match the original color distance value between adjacent regions without preserving the colors themselves. Neighbouring regions with large differences will be assigned distant colors, preserving the boundary, while similar-colored regions will be assigned similar output colors or even the same color.

We use a graph structure to organize the segmented image, where each region is a node and edges link adjacent regions. To simplify the color decisions, we will construct a tree over the graph, with a tree traversal assigning colors to nodes based on the decision for the parent node. We therefore assign weights to the edges reflecting their priority order, with large regions, regions of very similar color, and regions of very different color receiving high priority. Small regions and regions with intermediate color differences receive lower priority. Once weights have been assigned, we find the tree within the graph that maximizes the weight of the minimum-weight edges, a construction that corresponds to the *widest path* problem.

Our algorithm has two main steps. First, we create a tree by applying the *widest path* algorithm [17] on the region graph. Then we assign colors to regions by traversing the tree beginning from its root. For each region in the graph, we choose the color from the palette that best matches the color difference with its parent. Before starting, we apply histogram matching between the regions' color differences and the palette color differences. The histogram matching allows us to best convey the image content while using the full extent of an arbitrary input palette, even one that has a color distribution very different from that of the input image.

We show a flowchart of our recoloring approach in Figure 3.

The input is (a) an oversegmented image and (b) a palette to use in the recoloring. We compute (c) the adjacency graph of the segments and aggregate the differences between adjacent colors into the set $\{Q\}$. We also compute (e) the set of differences between palette colors, yielding $\{\Delta P\}$. The widest path algorithm (d) gives us a tree linking all nodes of the adjacency graph. We match the histogram (f) of $\{Q\}$ to that of $\{P\}$ and (g) assign colors to all regions by traversing the widest-path tree, resulting in (h) the fully recolored image.

Before we explain the recoloring proper, we introduce the *widest path problem*. We will employ the widest path algorithm to create a tree over the input oversegmentation. The tree structure organizes the regions with the largest edge weights to get processed earlier to maintain an edge and contrast preservation for the recolored image. We will traverse the tree and assign a color to each region, matching the edge's target color difference with the color difference with its parent's color. In practice, it can be practical to combine the tree creation and traversal, since the widest-path algorithm involves a best-first traversal of the tree as it is being built. Prior to color assignment, we apply histogram matching to align the regions' color differences with the palette's for better use of palette colors.

3.1 Tree Creation: The Widest Path Problem

Pollack [17] introduced the widest path problem. Consider a weighted graph consisting of nodes and edges $G = (V, E)$, where an edge $(u, v) \in E$ connects node u to v. Let $w(u, v)$ be the weight, called *capacity*, of edge $(u, v) \in E$; capacity represents the maximum flow that can pass from u to v through that edge. The minimum weight among traversed edges defines the capacity of a path. Formally, the capacity $C(u, v)$ of a path between nodes u and v is given by the following:

$$C(u, v) = \min(w(u, a), w(a, b), .., w(d, v)) \qquad (1)$$

where $w(u, a), w(a, b), .., w(d, v)$ are the edge weights along the path. The *widest path* between u and v is the path with the maximum capacity among all possible paths.

In a single-source widest path problem, we calculate for each node $t \in V$ a value $B(t)$, the maximum path capacity among all the paths from source s to t. The value $B(t)$ is the *width* of the node. The union of widest paths from the source to each node is a tree, which we use to order the color assignment process. We can choose any node as the source; our implementation uses the region containing the image centre.

The widest path algorithm can be implemented as a variant of Dijkstra's algorithm, building a tree outward from the source node s to every node in the graph. All nodes of the graph $t \in V$ are given a tentative width value; the source node s will be assigned to $B(s) = +\infty$ and all other nodes $v \neq s$ will have $B(v) = -\infty$. A priority queue holds the nodes; at each step of the algorithm, we take the node with the highest current width from the queue and process it, stopping when the queue is empty. Suppose the node u is on top of the queue with a width $B(u)$; for every outgoing edge (u, v), we update the value of the neighbour node v as follows:

$$B(v) \leftarrow \max\{B(v), \min\{B(u), w(u, v)\}\} \qquad (2)$$

where w is the edge weight between nodes u and v. If the value $B(v)$ was changed, node u will be set as the parent node and v will be pushed into the queue. When the algorithm completes, all non-root nodes in the graph will have been assigned a single parent, thus providing a tree rooted at s.

In our application, one possibility for edge weight is to use the difference in color values between the two regions. This would ensure that the widest path tree linked dissimilar regions, resulting in good edge preservation. However, regions of similar color could easily be divided. We want to preserve small color distances as well, so small color differences should yield a large edge weight. Distances intermediate between large and small are of the least importance. Hence, we base our edge weight on the difference from the median color distance, as follows.

We calculate the color distances across each edge in the adjacency graph; call the set of color distances Q, with

$$Q = \{\Delta(c_i, c_j)\} = \{q_{ij}\}, \quad i \neq j, \quad r_i, r_j \in R$$

where c_i and c_j are the colors of regions r_i and r_j and Δ is the function computing the color distance. Compute the median value $\bar{q}$ from the distances in Q.

We also want to take into account the size of the region, such that larger regions have greater importance; we prefer that a larger region have higher priority and thus influence the smaller regions that are processed afterwards, compared to the converse. Depending on the oversegmentation, action may not be necessary; we suggest a process to improve results on oversegmentations with a dramatic variation in region size.

We compute for each region a factor b, the ratio of the region's size (in pixels) to the average region size. Then, when we traverse an edge, we use the b of the destination region to determine the weight. In our implementation, we compute and store a single edge weight; there is no ambiguity about the factor b because we only ever traverse a given edge in one direction, moving outward from the source node.

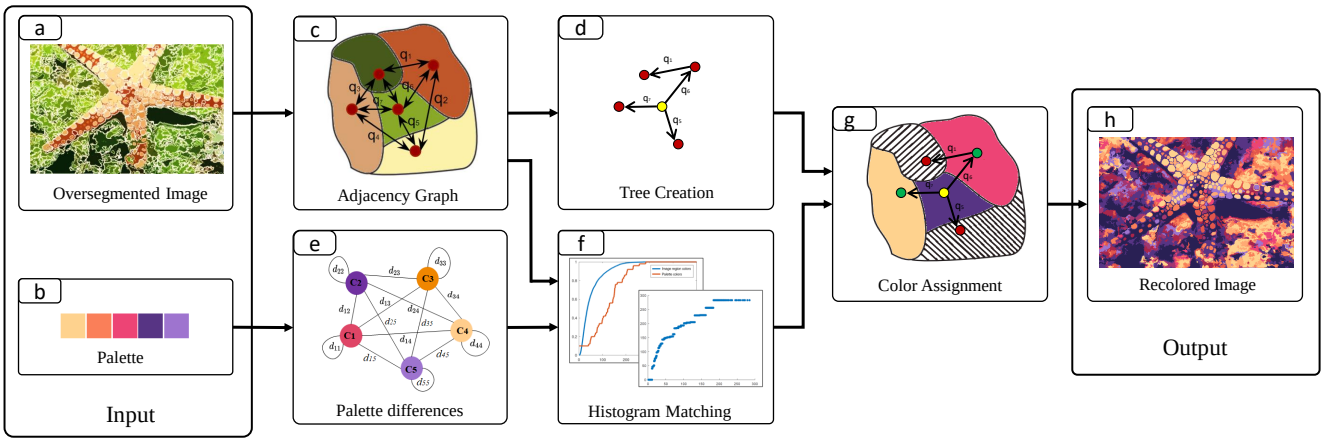

Figure 3: Recoloring algorithm pipeline.

To summarize: when traversing an edge, the edge weight is the distance between its target color difference and the median color difference, multiplied by a the factor $(1+b)$ for the destination region:

$$w(r_i, r_j) = (1+b)|\Delta(c_i, c_j) - \bar{q}|. \tag{3}$$

The factor $(1+b)$ takes size into account, but ensures the region's color differences can still affect the traversal order even for very small regions (b near zero). Note that the function Δ depends on the colorspace used. A simple possibility is Euclidean distance in RGB, but more perceptually based color distances are possible. We discuss color distance metrics in section 5.2.

3.2 Histogram Matching

We plan to match color differences in the output to the color differences in the input. However, the input palette can have an arbitrary set of colors, and we also want to make use of the full palette. For example, imagine a low-contrast image recolored with a palette of more varied colors. The smallest palette difference might be quite large; if so, the muted areas of the original will be matched with difference zero, resulting in loss of detail in such regions. A narrow palette recoloring a high-contrast image will have similar problems in the opposite direction.

To adapt the palette usage to the input image color distribution, we apply histogram matching to color differences. We emphasize that we are not matching the colors themselves, but the distributions of differences. Histogram matching is applied between the region color differences (the distribution of values in Q, computed in Section 3.1) and the pairwise color differences of the colors in the palette (call this dataset ΔP).

The histogram matching computes a new target color difference for each graph edge; call this target $q'(u, v)$ for the edge linking regions u and v. The matching ensures that the distribution of values $\{q'\}$ is the same as the distribution of values in $\{\Delta P\}$. The values q' are then used for color assignment, selecting color pairs from the palette which correspond to the same place in the distribution: medium palette differences where medium image color differences existed, small differences where the original image color differences were small, with the largest palette differences reserved for the largest differences in the original image.

Figure 4 shows the input image with average region colors and the recolored images before and after histogram matching. The bar graphs under each image show the proportion of each color in the image. We can see that the histogram matching used the palette colors more evenly, increasing contrast and highlighting more details. For example, strong edges on the leaf boundary became distinguishable from the nearby regions, and the markings on the lizard became more prominent.

Figure 4: Histogram matching result. Left to right: original image, result without histogram matching, and result using histogram matching.

3.3 Color Assignment

The widest path algorithm provided a tree, and the histogram matching provided palette-customized target distances for the edges. We now traverse the tree and assign a color to each region along the way. We begin by assigning the closet palette color to the tree's root node; recall that the root was the most central region in the image. At each subsequent step, we assign a color from the palette P to the current region α based on the palette color p_β previously assigned to the parent region β and the target color difference $q'(\alpha, \beta)$. We also consider the luminance difference between regions r_α and r_β so as to help maintain larger-scale intensity gradients. Recall our intention in preserving color differences: two regions with large color differences should be assigned two very different colors, and regions with a small color difference should get very similar colors, possibly the same color. Owing to histogram matching, "large" and "small" are calibrated to the content of the particular image and palette being combined.

We impose a luminance constraint on potential palette colors, in an effort to respect the relative ordering of the regions' luminances. Suppose the luminances of two regions r_α and r_β are L_α and L_β, where $L_\alpha < L_\beta$. We then constrain the set of eligible palette colors for region α such that only colors p_α that satisfy $L_{p_\alpha} < L_{p_\beta}$ are considered. A similar constraint is imposed if $L_\alpha > L_\beta$.

For region r_α and its parent r_β, we have the target edge difference $q'(\alpha, \beta)$. Denote by p_β the palette color already assigned to the

parent region r_β. We choose the palette color p_α for region r_α so as to minimize the distance D:

$$D = |q'(i,j) - \Delta(p_\alpha, p_\beta)| \tag{4}$$

where Δ is the distance metric between two colors. The only colors considered for p_α are those that satisfy the luminance constraint.

For colors with the lowest and highest luminance in the palette, there may be no available colors satisfying the luminance constraint. In such cases, the constraint is ignored and all palette colors are considered.

Since the source region has no parent, the above process can not be used to find its color. Instead, we assign the closest color from the palette, as determined by the difference metric Δ. The source region itself is the region containing the centre of the image; while the output is weakly dependent on the choice of starting region, we do not view the starting region as a critical decision. Figure 5 shows some examples of varied outcomes from moving the starting region.

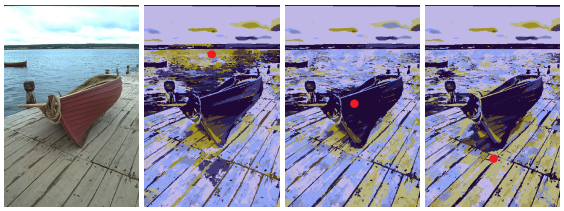

Figure 5: Changing the source region locations. The starting region is indicated by a dot.

4 Results and Discussion

In this section, we present some results. Figure 6 shows a variety of recolored images generated by our algorithm using various palettes. We succeeded in maintaining strong edges, and objects in the recolored abstractions remain recognizable. Our algorithm retains textures and produces vivid recolored images by selecting varied colors from the palette. It assigns the same colors over flat regions and distinct colors to illustrate structures. We ran our algorithm on a variety of images with different textures and contrasts. We obtained most of our palettes from the website *COLRD* (http://colrd.com/); others we created manually by sampling from colorful images.

In Figure 6, we present a set of examples from our recoloring algorithm, which were generated with four different palettes. From left to right, the columns show the original image, followed by recolored images using different palettes. We chose images presenting different features. The delicate features and textures in the abstractions stay visible after recoloring despite the input photographs having been radically altered by the recoloring.

In the *starfish* image, the structure and the patterns on the arms become visible. The uniform colors on the background turn to a vivid splash of colors, emphasizing the textureness of the terrain. The algorithm has chosen the darkest colors to assign to the shadows and the lightest ones to the surface of the creature.

The *Venice* canal is a crowded image composed of soft textures and structures with hard edges. The algorithm is able to preserve recognizable objects such as the boats and windows. Even tiny letters on the wall and pedestrians on the canal's side are visible. The recoloring process preserved the buildings' rigid structures; meanwhile, it captured shadows and the water's soft movements. In capturing such features, adopting a highly irregular oversegmentation was necessary.

The *lizard* image is an example of a low contrast image with textured areas covered by dull colors. The algorithm highlighted the textures by assigning wild colors to the homogeneous regions on the leaf. At the same time, the substantial edges like the lizard's body patterns and the leaf edges are preserved naturally by our algorithm.

In the next example, we used a high contrast image as input. The algorithm assigned the darkest colors from each palette to the coat of the man and separated it from the background using a very light color. Further, the small features on the face and the Chinese characters are mostly readable.

The *rust* image contains a different type of textures on the wall and the grass, plus soft textureless areas on the machinery. The brick patterns on the wall exaggerated through colorful palettes made the final images more interesting than the original flat image. The high-frequency details of the grass are retained. The smooth transition of colors on the top right of the image illustrated the shadows of the leaves.

We demonstrated strong edge preservation in all examples. Additionally, the image textures preserved, and palette colors are uniformly used to maintain a good contrast.

4.1 Comparison with Naïve Methods

Figure 7 gives a comparison between our method and two naïve alternatives. The first column shows the input image. The second column shows the input segments recolored by replacing the segment's color with the palette color closest to the segment's average color. The third column shows a random assignment of palette colors to segments. The final column shows our method. Recoloring with closest palette colors preserves some image content, but the result shows large regions of constant color; many of the palette colors are underused, an issue that can worsen when there is a significant mismatch between the original image color distribution and the palette, as in the upper example. Random assignment provides an even distribution of palette colors, but the image content can become unrecognizable for highly textured images, as in the lower example. Our method uses the palette more effectively, showing local details and large-scale content and exercising the full range of available colors.

4.2 Comparison with ColorArt

In this section, we compare our recoloring method with *ColorArt*, an optimization-based recoloring method for graphic arts [3]. This method assigns colors to regions by solving a graph matching problem over color groups in the reference and the template image. In searching for a reference image, this algorithm uses the same number of color groups as in the template image.

Figure 8 shows the generated images by *ColorArt* method on the right and ours in the middle, both using the *sunset* palette. We created a colorful leaf surrounded by a light background as in the input image, showing the algorithm respects the changes in lightness. Moreover, assignment of different colors on the leaf presented an interesting texture. The leaf image generated by the*ColorArt* method has reversed the image tones. In the *sketch* image, we preserved the edges and showed a recognizable face in the image. In contrast, the *ColorArt* algorithm had difficulty with the edges and the gradual gradients, resulting in a somewhat incoherent output.

4.3 Recoloring with SLIC0 Oversegmentation

Our recoloring algorithm does not make any assumptions about the input oversegmentation. Figure 9, shows results from an oversegmantation from SLIC0 [1]. The *starfish* and *owl* images have approximately 2000 and 5000 segments, respectively.

Note that more irregular regions can better represent complex image contours and textures, allowing the recolored abstractions to better display the image content. In *starfish*, the structures and shadows are represented by distinct colors that contrast between the object and the background. The strong edges, such as the arms of the *starfish*, are preserved; however, the thin features are not captured by SLIC0's uniform regions, and the background terrain does not present any significant information. In the *owl* image, the small regions on the chest convey the feather textures, while definite

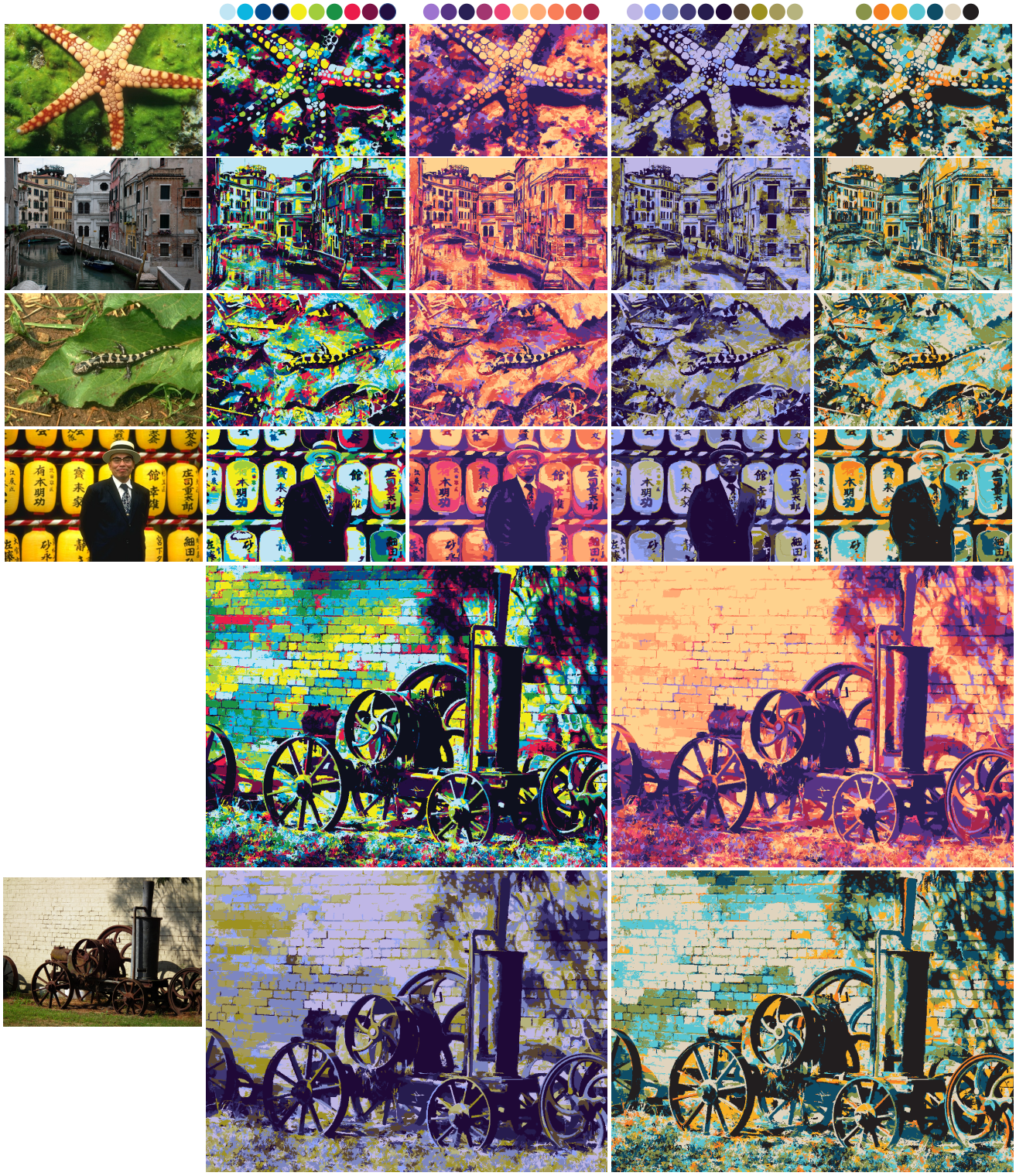

Figure 6: Region recoloring results. Top row: visualization of palettes used. Images, top to bottom: *starfish*, *Venice*, *lizard*, *lanterns*, and *rust*.

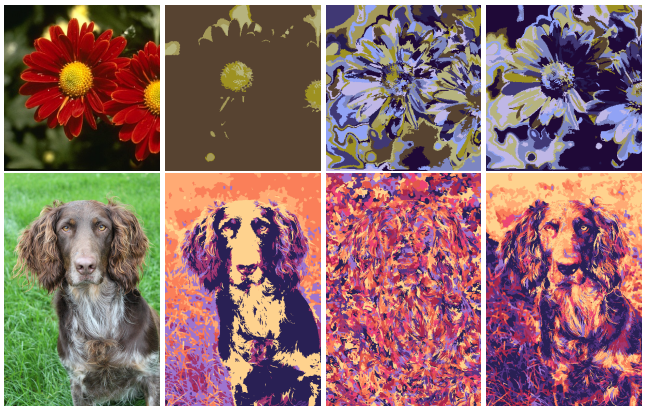

Figure 7: A comparison with naïve recolorings. Left to right: The original image, the results from naïve method, random recoloring, and ours.

regions such as dark eyes kept their well-defined structures. Given a suitably detailed oversegmentation, we can produce appealing results.

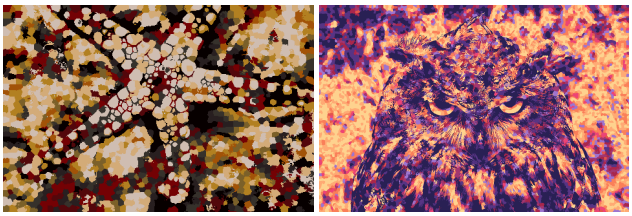

Figure 9: Recoloring with SLIC0 oversegmentation.

4.4 Performance

We ran our algorithm on an Intel(R) Core(TM) i7-6700 with a 3.4 GHz CPU and 16.0 GB of RAM. The processing time increases with the number of regions and edges in the graph. The time complexity of single-source widest path is $\mathcal{O}(m + n \log n)$ for m edges and n vertices, using a heap-based priority queue in a Dijkstra search.

Table 1 shows the timing for creating the trees and color assignments of different images. For small images like the *starfish*, containing about 1.4K regions, the recoloring algorithm takes about 0.007 seconds to construct the tree, while it takes about 0.3s for larger images such as *rust* with 7.2K regions. With a palette of 10 colors, the color assignments will take 0.05s and 1.2s to recolor the *starfish* and *rust* respectively. The color assignment will take longer for the larger palettes and images with a larger number of regions. We show the timing of tree creation and color assignments for all images in the gallery. The pre-processing steps of creating the graph adjacency list and histogram matching are also presented in the table.

4.5 Limitations

Although in our experience our method works well for most combinations of image and palette, there are cases where the output is unappealing. When two similar regions are not neighbours, they may receive different colors; e.g., a sky area may be broken up by branches and different parts of the sky could be colored differently. Even adjacent regions may not receive similar colors if their average colors differ, introducing spurious edges into regions with slowly changing colors such as gradients or smooth surfaces. Out-of-focus backgrounds and faces are common examples producing such effects,

Figure 8: Comparison with ColorArt. Left: input; middle: our results; right: ColorArt results.

Figure 10 shows two failure examples. On the top middle image, the face and background regions got very different colors with an unappealing outcome. However, a palette containing several similar colors allows the algorithm to better match the image gradients, with dramatically better results. In the bottom images, large regions of different colors appear in the sky, which does not look attractive. Because the original regions have different average colors, our algorithm is likely to separate them regardless of the palette.

Our algorithm is at present strictly automated with no provision for direct user control beyond choice of palette and parameter settings. While these parameters provide considerable scope for generating variant recolorings, so that a user would have a wide range of results to choose from, direct control is not yet implemented. One might imagine annotating the image to enforce specific color selections or linking regions to ensure that their output colors are always the same. While it would be straightforward to add some control of this type, we have not yet implemented such features.

Figure 10: Failure cases.

5 Variations

In this section, we explore some variations of our region recoloring algorithm. We apply a blending mechanism to create recolored images with smooth color transitions. Then, we demonstrate that different color spaces and distance metrics can generate various results from one palette.

Image	Tree creation	Color assignment	Graph	Total	# Vertices
lanterns	0.003s	0.02s	0.064s	0.087s	1K
starfish	0.007s	0.05s	0.07s	0.127s	1.4K
lizard	0.015s	0.08s	0.1s	0.195s	1.9K
rust	0.3s	1.2s	0.9s	2.4s	7.2K
Venice	0.3s	1.4s	0.9s	2.6s	7.7K

Table 1: Timing results for images with varying numbers of regions.

5.1 Blending

In transferring the colors, we have intended to strictly preserve the palette and not add colors. However, we can also blend the colors, giving a more painterly style and to adding a sense of depth to a flattened abstraction. Postprocessing the recolored image introduces new intermediate shades of colors. We suggest cross-filtering the recolored image with an edge-preserving filter [14]. This process smooths the areas away from edges while retaining the colors close to strong edges. In addition, blending across the jagged region boundaries smooths regions which originally possessed similar colors.

The cross-filtering mask size will affect the outcome. Larger masks will produce a stronger blending effect; small features will be smoothed out, and the output image will become blurry in regions lacking edges. Figure 11 illustrates examples of blending using masks of sizes $n =$20, 100, and 300. Blending with $n = 20$ only slightly modifies the image; for larger masks, the blending is more apparent. At $n = 300$ we can see a definite blurring in originally smooth areas, although blurring does not happen across original edges. Using a gray palette, can can obtain an effect resembling a charcoal drawing with larger masks.

5.2 Color Spaces and Distance Metrics

We can employ different functions for our color distance function Δ. Different choices of color space and distance metric can affect the colorization results. Changing the distance metric will cause both the widest-path tree and the color assignment to change.

We have experimented with computing color distances with the Euclidean distance in RGB as well as using perceptually uniform measures CIE94, CIEDE2000, CIE76, and CMC colorimetric distances [21, 27].

Both CIE94 and CIEDE2000 are defined in the *Lch* color space. However, CIE94 differences in lightness, chroma, and hue are calculated from *Lab* coordinates. CMC is quasimetric, designed based on the *Lch* color model. The CIE76 metric uses Euclidean distance in Lab space.

In Figure 12, we show different outcomes from different metrics using two palettes. We can observe the strong edge and contrast preservation, which is an apparent result of perceptual uniform metrics. More importantly, each metric gives a unique recolorization to the abstraction, which allows a user to choose from different output images. We can get interesting results from each metric. However, in our judgement, more attractive results are obtained from Euclidean and CMC metrics; the delicate features and image contrast are maintained, and objects are generally preserved. CIE94 and CIE2000 metrics are also effective, but we found that the CIE76 metric rarely creates interesting results.

6 Conclusions and Future work

In this paper, we presented a graph-based recoloring method that takes an oversegmented image and a palette as input, and then assigns colors to each region. The result uses the palette colors to portray the image content, but without attempting to match the input colors. Designing our algorithm with the widest path allowed us to maintain the image contrast and objects' recognizability. We demonstrated our results with different palettes. We achieved vivid recolorization effects, effective for most combinations of input images and palettes.

In the future, we would like to investigate non-convex palette color augmentation, adding new colors extending an input palette while matching the palette's theme. We would like to extend the color assignment to consider color harmony, scoring based on compatibility of colors and thus effecting the ability of certain colors to be neighbors. Furthermore, we would like to be able to recolor smoothly changing regions like the sky more uniformly. Adding elements of user control would allow for better cooperation between the present automated method and the user's intent.

Acknowledgments

The authors thank ...

References

[1] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Susstrunk. Slic superpixels compared to state-of-the-art superpixel methods. 2012.

[2] E. Aharoni-Mack, Y. Shambik, and D. Lischinski. Pigment-Based Recoloring of Watercolor Paintings. In H. Winnemöller and L. Bartram, eds., *Non-Photorealistic Animation and Rendering*. Association for Computing Machinery, Inc (ACM), 2017. doi: 10.1145/3092919.3092926

[3] M. Bohra and V. Gandhi. Colorart: Suggesting colorizations for graphic arts using optimal color-graph matching. 2020.

[4] W. Casaca, M. Colnago, and L. G. Nonato. Interactive image colorization using Laplacian coordinates. In G. Azzopardi and N. Petkov, eds., *Computer Analysis of Images and Patterns*, pp. 675–686. Springer International Publishing, Cham, 2015.

[5] H. Chang, O. Fried, Y. Liu, S. DiVerdi, and A. Finkelstein. Palette-based photo recoloring. *ACM Trans. Graph.*, 34(4):139:1–139:11, July 2015. doi: 10.1145/2766978

[6] S. DiVerdi, J. Lu, J. Echevarria, and M. Shugrina. Generating Playful Palettes from Images. In C. S. Kaplan, A. Forbes, and S. DiVerdi, eds., *ACM/EG Expressive Symposium*. The Eurographics Association, 2019. doi: 10.2312/exp.20191078

[7] L. Fang, J. Wang, G. Lu, D. Zhang, and J. Fu. Hand-drawn grayscale image colorful colorization based on natural image. *The Visual Computer*, 35(11):1667–1681, Nov 2019. doi: 10.1007/s00371-018-1613-8

[8] G. R. Greenfield and D. H. House. Image recoloring induced by palette color associations. *Journal of WSCG*, 11:189–196, 2003.

[9] P. Kubelka. New contributions to the optics of intensely light-scattering materials. part i. *J. Opt. Soc. Am.*, 38(5):448–457, May 1948. doi: 10.1364/JOSA.38.000448

[10] A. Levin, D. Lischinski, and Y. Weiss. Colorization using optimization. In *ACM SIGGRAPH 2004 Papers*, SIGGRAPH '04, pp. 689–694. ACM, New York, NY, USA, 2004. doi: 10.1145/1186562.1015780

[11] S. Lin, D. Ritchie, M. Fisher, and P. Hanrahan. Probabilistic color-by-numbers: Suggesting pattern colorizations using factor graphs. In *ACM SIGGRAPH 2013 papers*, SIGGRAPH '13, 2013.

[12] J. Lu, S. DiVerdi, W. A. Chen, C. Barnes, and A. Finkelstein. Realpigment: Paint compositing by example. In *Proceedings of the Workshop on Non-Photorealistic Animation and Rendering*, NPAR '14, pp. 21–30. ACM, New York, NY, USA, 2014. doi: 10.1145/2630397.2630401

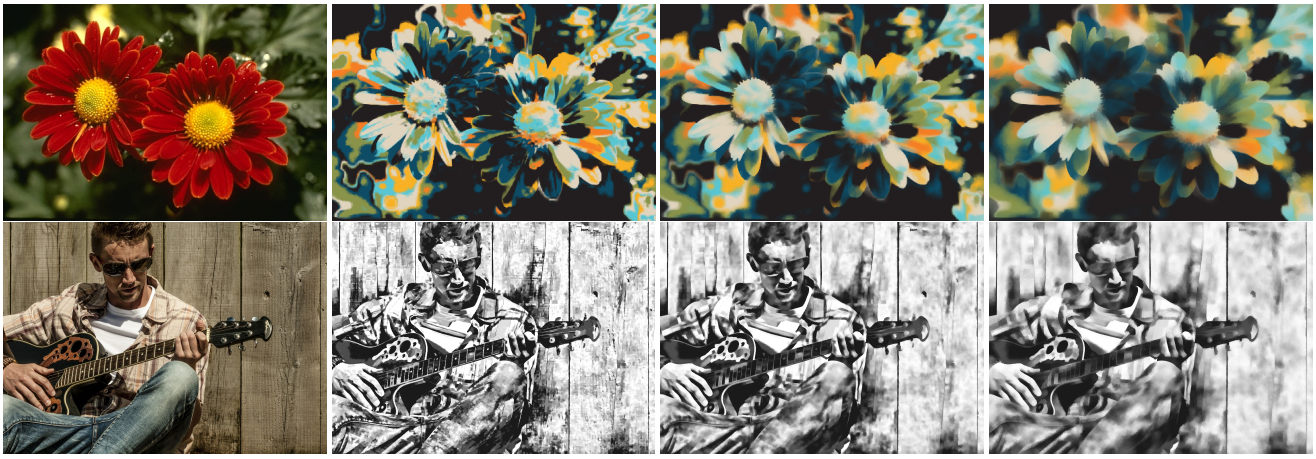

Figure 11: Postprocessing with an edge-aware filter. Left to right: original image; cross-filtered images with mask size 20, 100, and 300.

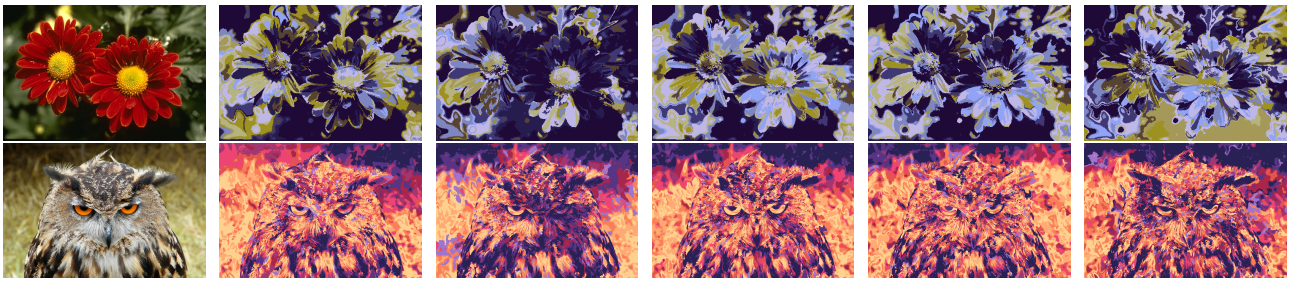

Figure 12: Colorization with different distance measures.

[13] J. McCann and N. S. Pollard. Soft stacking. In *Computer Graphics Forum*, vol. 31, pp. 469–478, 2012.

[14] D. Mould. Texture-preserving abstraction. In *Proceedings of the Symposium on Non-Photorealistic Animation and Rendering*, NPAR '12, pp. 75–82. Eurographics Association, Goslar Germany, Germany, 2012.

[15] L. Neumann and A. Neumann. Color style transfer techniques using hue, lightness and saturation histogram matching. In *Proceedings of the First Eurographics Conference on Computational Aesthetics in Graphics, Visualization and Imaging*, Computational Aesthetics'05, pp. 111–122. Eurographics Association, 2005.

[16] P. O'Donovan, A. Agarwala, and A. Hertzmann. Color Compatibility From Large Datasets. *ACM Transactions on Graphics*, 30(4), 2011.

[17] M. Pollack. Letter to the editor—the maximum capacity through a network. *Operations Research*, 8(5):733–736, Oct. 1960.

[18] Y. Qu, T.-T. Wong, and P.-A. Heng. Manga colorization. *ACM Trans. Graph.*, 25(3):1214–1220, July 2006.

[19] E. Reinhard, M. Ashikhmin, B. Gooch, and P. Shirley. Color transfer between images. *IEEE Comput. Graph. Appl.*, 21(5):34–41, Sept. 2001.

[20] D. L. Ruderman, T. W. Cronin, and C.-C. Chiao. Statistics of cone responses to natural images: Implications for visual coding. *Journal of the Optical Society of America A*, 15:2036–2045, 1998.

[21] A. T. Sanda Mahama, A. Dossa, and P. Gouton. Choice of distance metrics for RGB color image analysis. *Electronic Imaging*, 2016:1–4, 02 2016.

[22] L. Shapira, A. Shamir, and D. Cohen-Or. Image appearance exploration by model-based navigation. *Comput. Graph. Forum*, 28:629–638, 2009.

[23] M. Shugrina, J. Lu, and S. Diverdi. Playful palette: An interactive parametric color mixer for artists. *ACM Trans. Graph.*, 36(4):61:1–61:10, July 2017. doi: 10.1145/3072959.3073690

[24] D. Sýkora, M. Ben-Chen, M. Cadík, B. Whited, and M. Simmons. Textoons: practical texture mapping for hand-drawn cartoon animations. In *NPAR*, 2011.

[25] D. Sýkora, J. Dingliana, and S. Collins. Lazybrush: Flexible painting tool for hand-drawn cartoons. *Computer Graphics Forum*, 28(2):599–608, 2009.

[26] J. Tan, J.-M. Lien, and Y. Gingold. Decomposing images into layers via RGB-space geometry. *ACM Trans. Graph.*, 36(1), Nov. 2016. doi: 10.1145/2988229

[27] Wikipedia contributors. Color difference — Wikipedia, the free encyclopedia, 2019. [Online; accessed 17-September-2019].

[28] J. Xu and C. S. Kaplan. Artistic thresholding. In *Proceedings of the 6th International Symposium on Non-photorealistic Animation and Rendering*, NPAR '08, pp. 39–47. ACM, New York, NY, USA, 2008. doi: 10.1145/1377980.1377990

[29] L. Yatziv and G. Sapiro. Fast image and video colorization using chrominance blending. *Trans. Img. Proc.*, 15(5):1120–1129, May 2006. doi: 10.1109/TIP.2005.864231

[30] K. Zeng, M. Zhao, C. Xiong, and S.-C. Zhu. From image parsing to painterly rendering. *ACM Trans. Graph.*, 29(1):2:1–2:11, Dec. 2009. doi: 10.1145/1640443.1640445

[31] M. Zhao and S.-C. Zhu. Sisley the abstract painter. In *Proceedings of the 8th International Symposium on Non-Photorealistic Animation and Rendering*, NPAR '10, pp. 99–107. ACM, New York, NY, USA, 2010. doi: 10.1145/1809939.1809951