The 8th ACM/EG Expressive Symposium
C. Kaplan, A. Forbes, and S. DiVerdi (Editors)

Generating Playful Palettes from Images

Stephen DiVerdi[1], Jingwan Lu[1], Jose Echevarria[1], Maria Shugrina[2]

[1]Adobe Research, USA
[2]University of Toronto, Canada

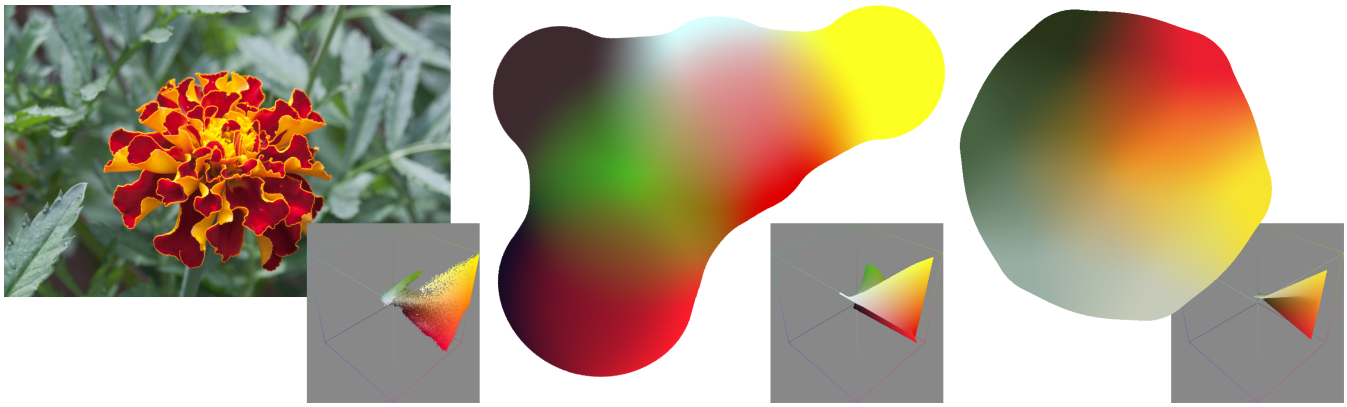

Figure 1: *Left to right: an example image, our optimized Playful Palette and our approximated Playful Palette. RGB histograms for each are displayed inset. We attempt to reproduce the full gamut of the input image with the colors in a Playful Palette. Our optimization produces high quality results, while our approximation is an order of magnitude faster.*

Abstract

Playful Palettes are a recent innovation in how artists can mix, explore, and choose colors in a user interface that combines the benefits of a traditional media painter's palette with non-destructive capabilities of digital tools. We present a technique to generate a Playful Palette that best represents the colors found in an input image, allowing the artist to select colors from the image's gamut, while maintaining full editability of the palette. We show that our approach outperforms recent work in terms of how accurately the image gamut is reproduced, and we present an approximation algorithm that is an order of magnitude faster with an acceptable loss in quality.

Keywords: Playful Palette, digital painting, color palette, gamut, optimization

CCS Concepts
• **Human-centered computing** → **Graphical user interfaces;** • **Computing methodologies** → *Image processing;*

1. Introduction

While digital painting tools have become dramatically more capable and expressive in recent years, there are still ways they could better support the artistic process. Improving the color picking interface is one such opportunity. Recently Playful Palette [SLD17] was introduced to address many longstanding shortcomings of digital color pickers, including supporting exploration and harmonization while providing non-destructive editing, infinite history and recoloring.

A remaining fundamental difficulty for many artists is the "blank canvas" problem, in which it can be difficult to take the first steps on a new, clean document [Aud93]. Similarly, it can be difficult to generate a color palette from scratch when lacking context or inspiration. Many artists gain inspiration from photographs, artwork, and other imagery that inspires the creative process.

We present an algorithm that can generate a Playful Palette from a user-provided image, such that the colors present in the Playful Palette match the image colors and vice versa, as closely as possi-

DOI: 10.2312/exp.20191078

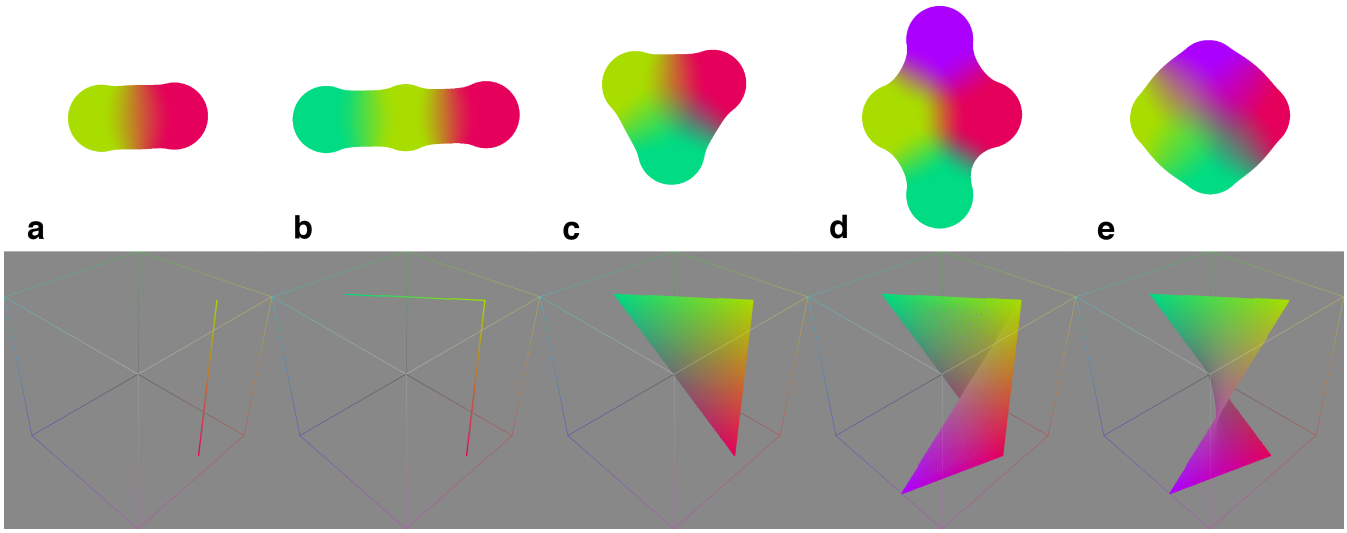

Figure 2: *Playful Palette configurations and their resulting RGB gamuts. Pairs of abutting blobs create linear gradients and thus lines in 3D (a,b). Triplets of abutting blobs create 3D triangles (c,d). When more than three blobs simultaneously abut (e), more complex curved surfaces can be constructed.*

ble. This can help artists to find inspiration in images or other artworks by providing a parameterized palette they can interact with to select colors, find new combinations, and mix and match to their liking. Furthermore, novice artists often struggle with generating colors for certain types of content such as skin tones. Creating a Playful Palette from an image of a face can easily provide the artist with a range of skin tones to select, and an easy way to experiment with shadowing, highlighting, or adding accent hues.

Our algorithm takes an image as input and directly optimizes the position and color of a fixed number of blobs until a loss function is minimized. Our loss function encodes the bi-directional similarity between the image RGB gamut and the Playful Palette RGB gamut. Optimization yields high quality (high bi-directional similarity) results but takes tens of seconds. We also present an approximation algorithm that runs in one tenth the time while achieving qualitatively similar results. We conclude by comparing our results to two state of the art palette generation approaches to show that our proposed algorithms are favorable in terms of both quality and runtime.

2. Related Work

The work most similar to ours is that of Nguyen et al. [NRS15], in which they create 1D or 2D palettes from images or collections of images using self organizing maps (SOM). These palettes can be used to visualize the colorspace of the input image(s), and support selecting colors as custom image-based palettes. The advantage of our work is that a Playful Palette representation of the same data affords greater interactive potential for the user—an artist can change blob colors, rearrange blobs, or add or remove blobs to take the image-based palette and customize it for their needs. More recently, Shugrina et al. [SKSF18] present Color Sails, which provide a more structured form of color palette that is more amenable to generation via machine learning, but does not have the interactive affordances of Playful Palette.

There is extensive work on selecting discrete palettes from images. O'Donovan et al. present the first work to use machine learning to model user preference for color palettes [OAH11], and later use the preferences of many users to understand classes of color and palette aesthetics among artists [OAH14]. Lin et al. [LH13] focus specifically on modeling how users extract palettes from images. Each of these works is limited to small palettes of discrete colors, and does not model gradients or blending.

Some previous works use palette models as ways to facilitate image editing. One of the earliest and most influential results came from Cohen-Or et al. [COSG*06], which extracts the hue distribution from an image and modifies it to confirm to an idealized hue template. Chang et al. [CFL*15] extract palettes that contain up to seven colors from an image and modify the image as the palette colors are changed by the user. Mellado et al. [MVH*17] extend this concept by incorporating perceptual constraints into the palette modifications. These works are useful for editing images, but do not support painting type interactions focused on color selection.

A related problem is to decompose an image into layers, which requires assigning colors to those layers that constitute a palette. Tan et al. [TLG16] use a simplified convex hull algorithm to create a basis for image pixels from a small number of color layers. LayerBuilder [LFDH17] achieves a similar result using nonlinear dimensionality reduction to generate the palette instead. Aksoy et al. [AASP17] performs a "soft segmentation" where each layer actually contains a distribution of colors. Most recently, Tan et al. present an improved version of their layer separation [TEG18] that is significantly faster when separating an image into layers, but the palette extraction uses the same approach as their earlier work. These techniques allow images to be modified accordingly,

but again do not support selection of new colors from the image gamut.

An interesting variation of the layer decomposition focuses on images of paintings specifically. The Pigmento system [TDLG18] estimates multispectral pigment coefficients and mixing weights for oil- or acrylic-type paintings. Aharoni-Mack et al. [AMSL17] do the same for watercolor paintings. These pigments could then be used in a painting system that simulates real physical paints.

Finally, there is work on recommending colors to support painting and design tasks. Son et al. [SOK*15] use color affinity from a public dataset of rated color themes to present to the artist colors that compliment the current composition. Phan et al. [PFC18] learn color palette models from curated sets of images (e.g. Cézanne paintings) and suggest relevant colors to choose among, based on the selected model. These interfaces are closer to our goal of making artists more successful, but they do not support unconstrained exploration or blending of colors.

3. Playful Palettes

A Playful Palette is defined as a set of discrete colored "blobs" in a "mixing dish," where the blobs smoothly blend with one another when they are within some distance threshold. Each blob is defined as a tuple $\{r,g,b,u,v\}$ where r, g, and b are the blob's color and u and v are the blob's location. For this paper, we only consider 8bit RGB color values normalized to $r,g,b,\in[0,1]$, though Playful Palette can support other color spaces (e.g. CMYK, HDR, or multispectral pigments). A blob position is defined within $u,v\in[-1,1]$. While Playful Palette supports blobs of varying radius, we fix the radius of all blobs to 0.3 (in normalized units of u,v); thus each blob nominally occupies 9% of the Playful Palette, comfortably allowing up to 10 blobs at once.

Each blob is rendered as a point sample with a radial falloff contribution. The contributions of all blobs are summed per pixel. For any pixel where the summed contribution is greater than a threshold, that pixel is colored according to the weighted average of the contributing blobs. This way, smooth gradients are achieved between abutting blobs.

This model is trivially parallelizable and therefore amenable to GPU computing. We have implemented the rendering algorithm as an OpenGL fragment shader which runs in an interactive HTML5 and WebGL demo. We have also implemented a CPU-only version in C++ with multiple threads and SSE SIMD vectorization, which is easier to use for optimization later.

3.1. Gamuts

It is useful to examine the RGB color gamuts that are constructed by different Playful Palette configurations. See Figure 2 for illustrations. Two abutting blobs create a smooth linear gradient between them, and therefore a straight line segment in 3D (Fig. 2a,b). Three abutting blobs effectively use barycentric coordinates for interpolation among three 3D points and therefore create a triangle (Fig. 2c).

In general, more than three blobs can be decomposed into a set of triplets of abutting blobs that form a triangle mesh in 3D (Fig. 2d). As the Playful Palette mixing dish is a 2D domain, the resulting 3D gamut must be at most a 2D manifold (generally, a combination of points 1D, and 2D manifolds)—there is no way to position four blobs to each mix with each other and create a continuous 3D volume in the RGB domain. This is in contrast to approaches by e.g. Tan et al. [TLG16] which explicitly constructs a simplified convex hull of the image RGB histogram so that every point in the interior volume can be achieved with some interpolating weights. Buades et al. [BLM10] argue that the color distribution of a real image can generally be best described with a 2D manifold.

There are some configurations of blobs that can achieve more than three blobs mixing with each other simultaneously and result in a curved surface rather than two planar triangles (Fig. 2e), but they are still not able to achieve a volumetric gamut, and therefore can only approximately model the colors in a natural image.

3.2. Problem Statement

The goal of this work is to construct a Playful Palette that best represents the RGB gamut of an input image. Inspired by work on bidirectional similarity [SCSI08], we consider two aspects of that representation. First, for every color in the input image, there should be a similar color in the Playful Palette. Second, for every color in the Playful Palette, there should be a similar image color. The first condition ensures that the Playful Palette accurately reproduces the set of image colors so the artist can select among them. The second condition ensures that the Playful Palette does not include many colors that are not in the input image, which would be unnecessarily distracting. It also avoids a trivial solution of a Playful Palette that simply reproduces all possible colors, by penalizing colors that are not present in the input image.

We focus on Playful Palette instead of other types of color palettes because it provides a convenient, low dimensional parameterization to approach the problem. For example, some digital color pickers mimic an oil painter's mixing palette as a bitmap the artist can mix colors on [BSLM01]. However, such a palette at a reasonable size could have as many as on the order of one million degrees of freedom. Conversely, a Playful Palette of reasonable size will have on the order of tens of degrees of freedom and be resolution independent. Another option is to create a palette of discrete swatches [CFL*15]. However such a palette will require many swatches to reproduce any smooth color gradient, and the resulting complexity will make the palette difficult for an artist to modify intuitively. Color Sails [SKSF18] are another option which have a strict topology to enable easier generation via deep learning, but lose the playful interactivity of Playful Palette.

4. Method

We formulate our approach as an optimization problem. The state vector $\mathbf{x}$ is defined as:

$$\mathbf{x}=\{r_1,g_1,b_1,u'_1,v'_1,r_2\cdots\} \tag{1}$$

For each blob there are five variables representing its color and position, so for n blobs there are $5n$ variables. In general, our target $n=8$ so $\mathbf{x}$ has 40 dimensions. We chose $n=8$ as we feel it is difficult to work with palettes with more blobs, and this provides the

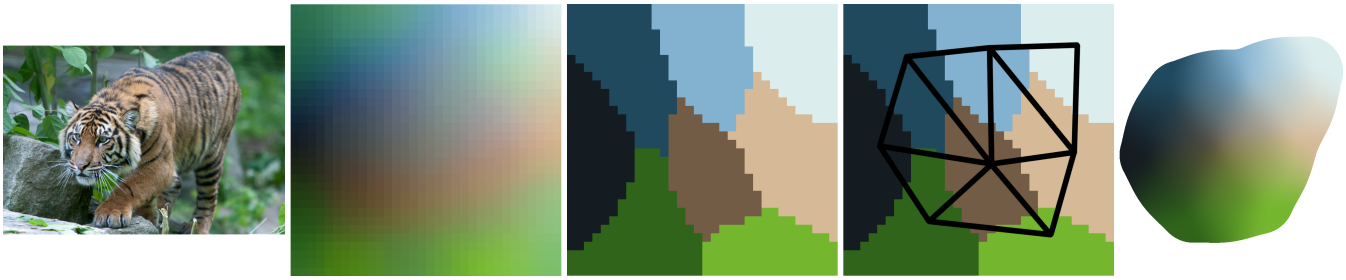

Figure 3: *Our approximation algorithm. From left to right: the input image, the* 32×32 *SOM result, the clustering result, the cluster triangulation, and the output Playful Palette.*

maximum flexibility to represent image colors, but it remains a user parameter. We optimize on $u' = \frac{u+1}{2}$ and $v' = \frac{v+1}{2}$ so that all state variables are in $[0,1]$.

We explicitly choose not to include blob radius, even though it is a user parameter. The reason for this is that it is not clear how blob radius will impact the achieved color gamut of a Playful Palette. It is easy to imagine that the optimization might drive blob radii towards zero or to become unbounded, and additional regularization or constraint terms would further complicate the optimization. Therefore, we use a constant blob radius.

The state vector is rendered as:

$$I_{\mathbf{x}} = g(\mathbf{x}) \tag{2}$$

where $I_{\mathbf{x}}$ is the rendered image of the Playful Palette as defined by $\mathbf{x}$ and g is the rendering function defined by Shugrina et al. [SLD17]. As our implementation is a CPU optimization in C++, we use a CPU-only implementation of the Playful Palette rendering algorithm for g because it is easier to integrate into the optimization framework.

4.1. Objective Function

We define our objective function for minimization as:

$$f(\mathbf{x}) = d(I, I_{\mathbf{x}}) \tag{3}$$

where I is the input image and $I_{\mathbf{x}}$ is the rendered Playful Palette image for $\mathbf{x}$. d is a function that computes the distance between the two images, based on the intuitions discussed in Section 3.2.

$$d(I, I_{\mathbf{x}}) = \alpha\Big(\frac{1}{|I|}\sum_{p\in I}\min_{q\in I_{\mathbf{x}}}\|p_{RGB} - q_{RGB}\|^2\Big) + (1-\alpha)\Big(\frac{1}{|I_{\mathbf{x}}|}\sum_{q\in I_{\mathbf{x}}}\min_{p\in I}\|p_{RGB} - q_{RGB}\|^2\Big) \tag{4}$$

For image pixel p, p_{RGB} is the color triplet $\{r,g,b\}$ for that pixel. The first sum in Eq. 4 computes how well each image pixel is represented by the Playful Palette, while the second sum penalizes colors in the Playful Palette that are not in the image. The α parameter allows tuning for the relative strength of these terms. Using α=0.5 weights them equally, while α=1 ignores the penalty for palette colors that are not in the image. See Fig. 7 for a visualization of the impact of this parameter. We use α=0.9, which prefers more colorful palettes while limiting spurious colors, but it could be left as a user parameter.

4.2. Optimization

Unfortunately, we cannot compute derivatives for f analytically or with automatic differentiation, as the Playful Palette rendering function g is not expressed as a mathematical equation. Using finite differencing to estimate f' for gradient descent requires 41 evaluations of g per step which is prohibitively slow.

Therefore we must rely on derivative-free optimization methods. We experimented with CMA-ES [HMK03] but also found that it relied on too many expensive g evaluations to progress. We use the Nelder-Mead simplex algorithm [NM65] because it uses relatively few g evaluations and converges in a reasonable amount of time. Conventional wisdom is that Nelder-Mead is poorly suited for problems with many variables [HAR*10], but we find that it performs well for our problem. We believe this is partially because of our favorable initialization strategy (detailed later), and partially because the high cost of evaluating our objective relative to the *initial* progress made by Nelder-Mead before convergence is still preferable [HN06].

To improve the performance when computing f, we render the Playful Palette image $I_{\mathbf{x}}$ at 48×48 pixels. We also subsample the input image I with a 3D histogram. We use $16 \times 16 \times 16$ uniform RGB histogram bins to store the pixel colors of I, and consider only the colors of non-zero histogram bins. Therefore we have on the order of 1000 pixels from I and $I_{\mathbf{x}}$ each when computing Eq. 4.

4.3. Initialization

The nature of our objective function f means that there are large portions of the search space with zero or near zero gradients, which makes optimization difficult. For example, with two blobs, if the blobs are far apart, then small changes in u and v for either blob will not change the achieved gamut because it will not change the connectivity of blobs. In these cases, the blobs will not move closer together to ultimately create a linear or triangular portion in the output gamut.

To address this problem, we rely on a good initialization. While the optimization may struggle to move two distant blobs together, it is able to move two abutting blobs apart. Therefore, we begin the

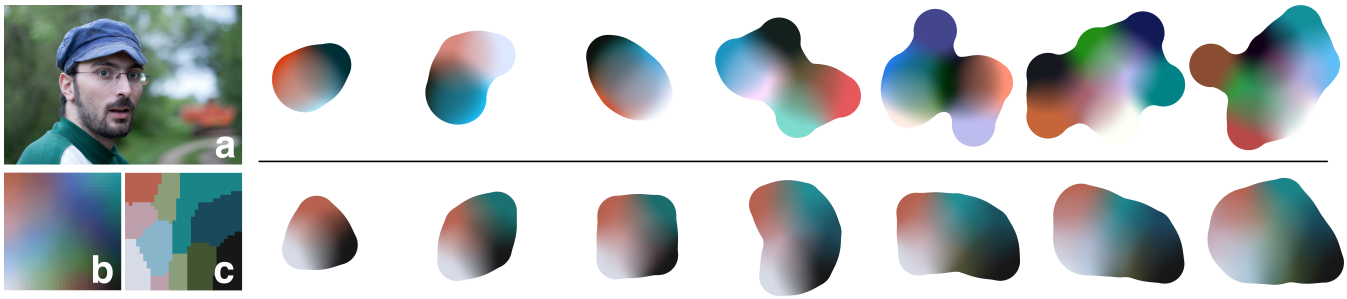

Figure 4: *Our results for different numbers of blobs from 3 (left) to 9 (right). Optimization results are on top and approximation results are on bottom. (a) The input image, (b) the SOM result, (c) the k-means clusters for k=9. The approximation results are less colorful than the optimization because the SOM and k-means steps produce a subset of all image colors. Importantly, color salience is not considered—though skin pixels are clearly important in the image, they are a small portion of the overall gamut and may not be perceptually dominant in the resulting Playful Palette.*

optimization with all blobs abutting. Specifically, for every blob, u=0.5 and v=0.5.

We also experimented with different color initializations. We found that both random initialization or uniform initialization (where all blob colors are set to the same average color value) were unlikely to reliably generate the set of blob colors we thought were appropriate for an image. To encourage inclusion of a diverse set of salient colors, we instead initialize the blob colors with the results of k-means clustering [Jai10] on the image colors. That is, we perform k-means clustering where k=n and for each cluster compute the average RGB color and assign it to one of the blobs.

5. Approximation

While our optimization approach produces the best results, it may be too slow to use in an interactive application. Therefore, we also propose a feed-forward approximation algorithm as an alternative to an iterative closed-loop. The approximation operates in a series of steps: the input image is subsampled, the color samples are approximated via non-linear dimensionality reduction to two dimensions, the 2D results are clustered to find discrete colors, the clusters are triangulated, and the triangle vertices become the Playful Palette blobs. See Figure 3 for an illustration of these steps.

5.1. Dimensionality Reduction

We base our approach in that of Nguyen et al. [NRS15], who approximate the 3D RGB color distribution of a natural image (or set of natural images) using one or two dimensional Kohonen Self-Organizing Maps (SOM) [Koh90]. This formulation is appealing for its similarity to our problem statement: a 2D manifold defined by vertices and local edge connectivity (a quad mesh) is used to approximate a 3D volumetric cloud of samples. It is intuitive that the resulting 2D manifold could be approximated by triangles, and that those triangles would then constitute a Playful Palette.

Other non-linear dimensionality reduction methods may be applicable in addition to SOM: multi-dimensional scaling [CC00], isomap [TDSL00], locally linear embedding [RS00], or t-SNE [MH08] for example. The benefit of SOM is that it lets us specifically enforce a 2D square grid topology, which correspondingly guarantees we can interpolate smoothly within the output manifold. SOM also converges quickly and yields good results.

We use an SOM of 32×32 nodes. SOM performance is directly a function of the number of input training samples, so we use a subset of the image pixels. Furthermore SOM will take into account the density of samples, which means that if for example an image is mostly black, the output SOM will output most nodes with black or near black colors. This is not useful for users though, who should see a diverse gamut regardless of image frequency. Therefore as input to our approximation algorithm, we subsample the input image as the non-zero bins of a uniform $16 \times 16 \times 16$ 3D histogram on the RGB color cube. This yields on the order of 1000 input samples, regardless of image size.

5.2. Clustering

Once we have a 2D continuous distribution of colors representing our input image, we need to select the discrete set of colors of our output blobs. We experimented with mean shift clustering [Che95] and k-means [Jai10].

Mean shift is appealing because it can also suggest the appropriate number of clusters. However mean shift relies on two user parameters—the spatial distance threshold and the sample difference threshold—which are difficult to tune. We found that across a wide variety of images, there were often images with too many clusters (i.e. more than 10) or with clusters that did not achieve a single discrete color, regardless of how we set the parameters. Barring a principled way to adapt these thresholds to the image to ensure a reasonable number of discretely colored clusters, this was not a suitable option. Instead, k-means guarantees that the output will be the target number of clusters and each cluster will have a single color assigned to it.

As input to k-means, we pass in the reduced set of RGB colors output from the dimensionality reduction as 3D points. While this ignores the spatial arrangement of the colors as a property of the clustering step, the colors have all been projected onto the SOM 2D manifold, so clustering reliably finds color clusters that are spatially connected (though it is not enforced as a requirement).

Figure 5: *Example output on Van Gogh's Starry Night for Chang et al. [CFL*15] (left) and Tan et al. [TLG16] (right), both configured to generate 7 colors.*

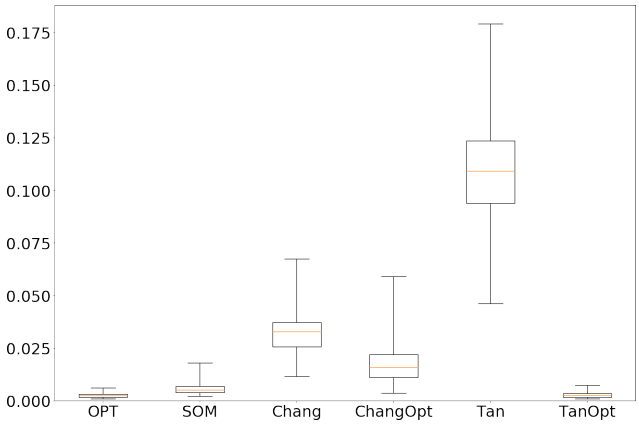

Figure 6: *Aggregate achieved objective values over 56 images. Each column shows min, 25th, 50th, and 75th percentile, and max.*

It is also possible to use an approach such as initializing k-means by mean shift as Cabria and Gondra propose [CG12]. The problem of mean shift still proposing too many clusters remains. We use k=8 and random initialization with acceptable results. It is still possible that random initialization may produce poor clusters, particularly if the distribution of input colors is very inhomogeneous. SOM discourages inhomogeneous distribributions however (by penalizing highly distorted mesh geometries), so we have not found this to be a problem.

5.3. Geometric Configuration

The output of the clustering step is a set of RGB colors from the input assigned to each cluster, as well as the average color for each cluster. We iterate through the pixels of the 2D manifold from SOM and for each pixel assign it the color of the cluster to which it belongs. Then for each cluster we find the pixel coordinate of the cluster centroid. These positions and colors are the basis of the output Playful Palette blobs. If we simply converted these values to blobs however, we would find that adjacent blobs do not necessarily abut which limits the gamut unnecessarily. Instead we prefer to have blobs touching when their clusters are spatially adjacent, to include their mixtures in the output gamut.

To achieve this, we compute a Delaunay triangulation [Del34] of the cluster positions. This creates a connected graph where every cluster position is a vertex and adjacent clusters have an edge connecting them. In our output Playful Palette with blob radius of 0.3, two blobs abut to form a pleasing gradient when their distance is roughly 0.45 (see Fig. 2a). Therefore, our goal is to position the vertices such that each edge is near 0.45 in length.

We move the vertices iteratively. For a single 2D vertex position p, the update equation is:

$$p' = p + \beta \frac{1}{|\Omega|} \sum_{q \in \Omega} \frac{t}{\|q-p\|}(q-p) \tag{5}$$

where Ω is the set of vertex positions adjacent to p, t is the target distance, and β is a rate parameter. We use β=0.1, with a maximum of 500 iterations and a convergence threshold of 0.001. Each iteration, we scale the vertex positions so their bounds are the unit circle, to avoid growing or shrinking.

6. Results

Figure 4 shows an example of how our optimization and approximation methods perform for varying numbers of blobs. Because the approximation relies on triangulation, it cannot estimate fewer than three blobs. As the number of blobs increases, the optimization result becomes significantly more colorful to capture image pixels that were not well represented with fewer blobs. The approximation results appear to converge and do not become very different beyond six or seven blobs (Fig. 4), likely because the SOM results do not change as input to the clustering. Adapting the number of SOM nodes based on the target number of Playful Palette blobs may address this limitation; we leave it for future work.

6.1. Evaluation

To evaluate our methods, we use them to generate palettes for a set of 56 images, with 44 photographs from the MIT-Adobe FiveK dataset [BPCD11], and 12 paintings from Google Arts & Culture [Goo18]. For each image we generate Playful Palettes with our optimization method (OPT) and approximation method (SOM). We visualize the RGB gamuts of each version and compute the objective function score as well. RGB gamuts are shown as 3D plots inside the unit cube, viewed down the cube diagonal (the luminance gradient from black to white). The objective function is the same as Eq. 4 with α=0.9—an objective value of $\frac{1}{256}$ is roughly 0.004, which corresponds to one 8-bit color step.

We compare our methods to two state of the art approaches: that of Chang et al. [CFL*15] and that of Tan et al. [TLG16]. Chang's technique uses k-means to find a set of color cluster centroids to use to parameterize edits to the image. Conversely, Tan's technique uses a simplified convex hull to decompose an image into a set of disjoint layers that composite together to form the input image.

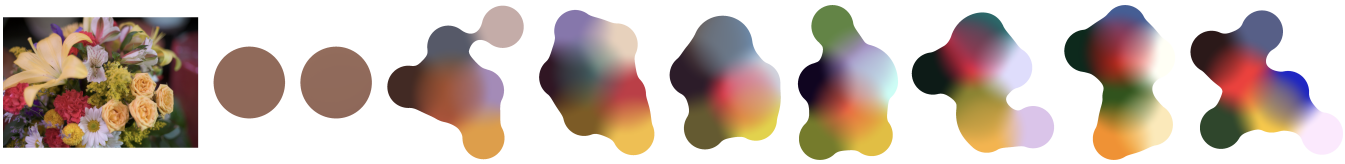

Figure 7: *Optimization results for different values of* α *in Eq. 4. From left to right,* α *is 0, 0.01, 0.1, 0.25, 0.5, 0.75, 0.9, 0.99, and 1. As* α *increases, the result includes more colors not present in the input image. We use* α=0.9 *but this could be left as a user parameter to customize the results.*

Both methods output a small palette of discrete colors (Fig. 5). We can directly compare these palettes via our objective function, but they perform poorly because they do not include any blending or gradients among the palette colors. As an extension of these techniques, we can also use them as blob color constraints in our optimization algorithm. This makes a more appropriate comparison between our results and previous work possible, by showing how well the colors extracted by previous work able to reproduce image gamuts in the Playful Palette framework.

We use the labels OPT, SOM, Chang, ChangOpt, Tan, and TanOpt for our optimization, our approximation, Chang et al.'s discrete and optimized versions, and Tan et al.'s discrete and optimized versions respectively. In our comparison, all methods produce Playful Palettes with seven blobs, because seven is the maximum number of palette colors Chang can produce.

Aggregate results over all 56 images for the achieved objective value of each treatment are presented in Table 1 and Figure 6. A Friedman test [Fri37] found that there is a statistically significant difference in the objective values depending on which algorithm is used, $\chi^2(5)$=271.5306, $p \ll 0.001$. Post-hoc pairwise Mann-Whitney U tests [MW47] with Bonferroni correction found significant differences ($p \ll 0.001$) for all comparisons except for OPT and TanOpt (U=1462.5, p=0.54). Example results are in Figure 8; the complete set is in the supplemental material.

	OPT	SOM	Chang	ChangOpt	Tan	TanOpt
μ	0.0027	0.0059	0.0326	0.0180	0.1077	0.0030
SD	0.0012	0.0029	0.0113	0.0106	0.0287	0.0016

Table 1: *The mean and standard deviation of the achieved objective function value (Eq. 4) over 56 test images for each algorithm.*

6.2. Performance

We ran all our tests on a 2015 retina MacBook Pro with a quad-core 2.5 GHz Intel Core i7, 16GB of RAM, and an AMD Radeon R9 M370X. The OPT method is a standalone executable implemented in C++, while the SOM method is implemented in JavaScript and executed inside Chrome. Chang is also JavaScript within Chrome, and Tan uses Python 2.7. All methods are CPU only. OPT renders palettes using multiple threads, but the optimization is single threaded. All other methods are single threaded. Timing data from a selected set of images is presented in Table 2. While each method uses different languages, the runtimes are dominated by the algorithm complexity—OPT and Tan involve slow optimizations over complex loss functions, while SOM's optimization is much faster and Chang directly computes its result.

image	OPT	SOM	Chang	Tan
0	26.2	2.5	1.1	34.5
3	26.9	2.6	1.3	21.2
8	19.1	2.5	1.0	11.5
45	34.3	2.8	1.5	15.7
54	8.5	2.4	1.7	20.1

Table 2: *Performance on selected images, in seconds.*

6.3. Discussion

From our comparison results there are a number of interesting observations to be made.

First, OPT and TanOpt perform equivalently, though TanOpt takes twice as long as OPT. SOM achieves roughly twice the score of OPT but is an order of magnitude faster. ChangOpt is unable to achieve competitive results. Unsurprisingly Chang and Tan are dramatically worst due to their discrete palettes.

Figure 8 shows some consistent behaviors that explain these differences in objective score. Because Tan uses a simplified convex hull, it tends to create palettes that include extreme colors which lie on the gamut periphery in order to contain all the image colors inside the hull volume. However, the optimization is able to move blobs to overlap more, so the resulting Playful Palette colors may mostly consist of their gradients. This may explain why the Tan discrete palettes can appear to include colors that are not obvious in the TanOpt palettes. Also, using Tan to extract seven colors means the convex hull can fit the image colors more tightly. A Tan palette of four or five colors would be more likely to include non-image colors correspondingly.

Chang on the other hand consistently produces color gamuts that are too conservative, missing many important image colors and generally being low contrast. This is because Chang uses a clustering approach, which selects palette colors as centroids of the image color clusters. These centroids are by definition at the centers of the clusters, which means there are portions of the clusters that are outside the resulting color gamut.

Because our optimization method is directly trying to minimize the objective function value, it can effectively choose palette colors that are around the edges of the image color gamut, to represent as

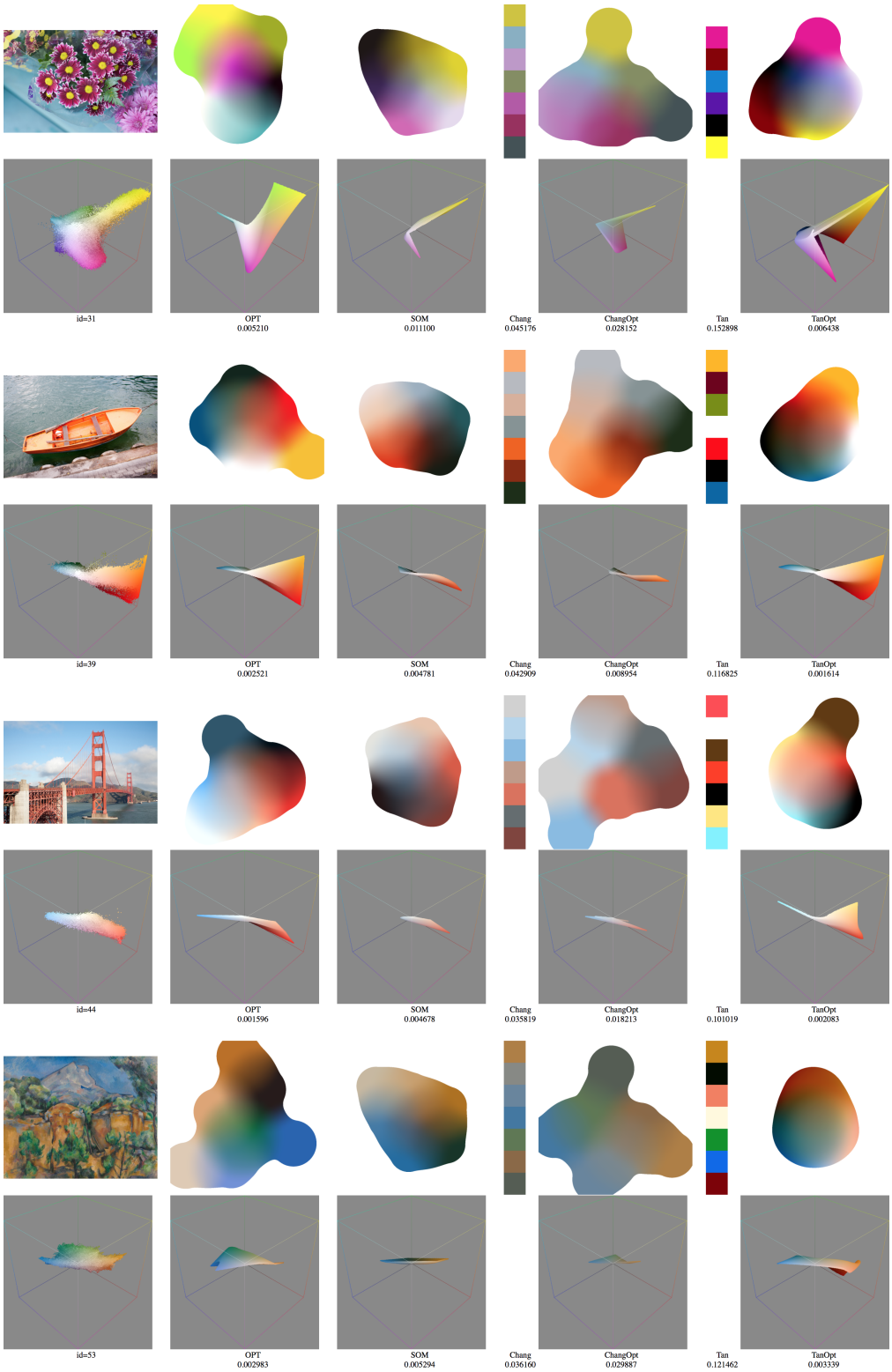

Figure 8: *Selected results. From left to right, columns are input image, our optimization, our approximation, Chang et al. [CFL*15] discrete and optimized palettes, and Tan et al. [TLG16] discrete and optimized palettes. Rows show constructed palettes, RGB histograms, and objective function values. Histograms of discrete palettes are omitted.*

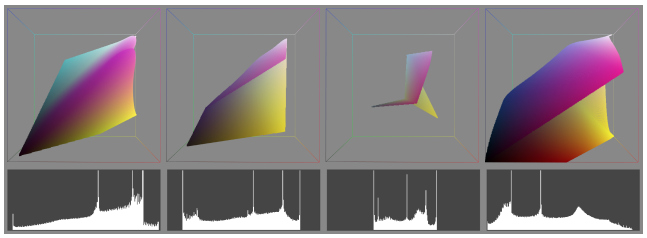

Figure 9: *Alternative visualization for the first row in Fig. 8 (id=31, purple flowers). RGB histograms (top) viewed along the green axis with black in the lower left, and 1D luminance histograms (bottom), from left to right: OPT, SOM, ChangOpt, and TanOpt.*

many of the colors as possible, without including too many extraneous colors. Our SOM approximation's main failure mode is to not include a salient image color in the output palette. This is generally a result of the clustering step, which may not find the appropriate number of clusters, or may cluster poorly due to initialization or other properties of the data. It is also possible that both the SOM step and the clustering step cause small amounts of shrinkage on the resulting gamut, which when combined reduce the quality of the result.

On the other hand, our approximation does a good job of capturing the luminance variation in an image, which is not well visualized by our RGB histograms because they are projected along the luminance dimension. However, comparing the SOM and ChangOpt palettes in Figure 8 shows that the palettes have similar chroma but consistently greater contrast in the SOM results. We also include Figure 9 to more clearly visualize luminance differences.

Performance-wise Chang is the fastest at about one second per image, while SOM is about 2.5 seconds, both fast enough for interactive use. OPT takes about 20 seconds on average, as does Tan, which is unsurprising as they both rely on iterative optimizations. ChangOpt and TanOpt performance is the sum of OPT and Chang or Tan respectively, making ChangOpt marginally slower than OPT (about 21 seconds) and TanOpt about twice as slow (about 40 seconds). While it is difficult to quantify what is "fast enough," in the context of a mobile and web app such as Adobe Color CC [Ado19], developer guidelines specify that response delays of 200 ms can feel "sluggish" and after 5 seconds, the app may be considered to be non-responsive [Fit10, Goo19]. When waiting for a webpage to load, 62% of shoppers will leave after 5 seconds [BM16]. An app such as Photoshop may make professional users wait longer for processing of large images, but for 1 Mpix images such as ours, most operations complete in a few seconds or less. While in certain contexts users may be willing to wait for OPT to complete, there are many cases where the performance of SOM is a requirement.

7. Conclusion

We present a method to create a Playful Palette that best represents the colors in the given input image, using an optimization approach that is high quality but slow. We also present an approximation algorithm that is an order of magnitude faster at about half the quality. We evaluate this performance in a comparison with two other state of the art techniques on a large set of images of natural scenery and paintings.

There are many promising avenues for future work. We do not explore here how users desire to select colors from an image, working on the assumption that a faithful reproduction is most valuable but maybe that is not the case. Once a palette has been generated from an image, it would also be interesting to consider how modifying the palette might be used to modify the image e.g. for recoloring. Finally, we generate a single palette for an entire image, but it might be more useful to generate separate palettes for different portions of the image, such as one palette for skin tones and another for foliage, by combining our algorithm with an image segmentation.

In the meantime, we hope our technique will make Playful Palette even more useful for digital artists to integrate into their creative process.

References

[AASP17] AKSOY Y., AYDIN T. O., SMOLIĆ A., POLLEFEYS M.: Unmixing-based soft color segmentation for image manipulation. *ACM Trans. Graph. 36*, 2 (Mar. 2017), 19:1–19:19. URL: http://doi.acm.org/10.1145/3002176, doi:10.1145/3002176. 2

[Ado19] ADOBE: Adobe color cc, 2019. URL: https://color.adobe.com. 9

[AMSL17] AHARONI-MACK E., SHAMBIK Y., LISCHINSKI D.: Pigment-based recoloring of watercolor paintings. In *Proceedings of the Symposium on Non-Photorealistic Animation and Rendering* (New York, NY, USA, 2017), NPAR '17, ACM, pp. 1:1–1:11. URL: http://doi.acm.org/10.1145/3092919.3092926, doi:10.1145/3092919.3092926. 3

[Aud93] AUDETTE A. H.: *The blank canvas: Inviting the muse*. Shambhala Publications, 1993. 1

[BLM10] BUADES A., LISANI J. L., MOREL J.-M.: On the distribution of colors in natural images. working paper or preprint, Feb. 2010. URL: https://hal.archives-ouvertes.fr/hal-00453249. 3

[BM16] BANKS N., MATTHEWS T.: Load time and mobile compatibility top online shopper demands, Feb 2016. URL: https://www.incapsula.com/blog/ecommerce-study.html. 9

[BPCD11] BYCHKOVSKY V., PARIS S., CHAN E., DURAND F.: Learning photographic global tonal adjustment with a database of input / output image pairs. In *The Twenty-Fourth IEEE Conference on Computer Vision and Pattern Recognition* (June 2011), CVPR '11. doi:10.1109/CVPR.2011.5995332. 6

[BSLM01] BAXTER B., SCHEIB V., LIN M. C., MANOCHA D.: Dab: Interactive haptic painting with 3d virtual brushes. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 2001), SIGGRAPH '01, ACM, pp. 461–468. URL: http://doi.acm.org/10.1145/383259.383313, doi:10.1145/383259.383313. 3

[CC00] COX T. F., COX M. A. A.: *Multidimensional Scaling, Second Edition*. Chapman and Hall, September 2000. doi:10.1007/b98835. 5

[CFL*15] CHANG H., FRIED O., LIU Y., DIVERDI S., FINKELSTEIN A.: Palette-based photo recoloring. *ACM Trans. Graph. 34*, 4 (July 2015), 139:1–139:11. URL: http://doi.acm.org/10.1145/2766978, doi:10.1145/2766978. 2, 3, 6, 8

[CG12] CABRIA I., GONDRA I.: A mean shift-based initialization method for k-means. In *2012 IEEE 12th International Conference on Computer and Information Technology* (Oct 2012), pp. 579–586. doi:10.1109/CIT.2012.124. 6

[Che95] CHENG Y.: Mean shift, mode seeking, and clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence 17*, 8 (Aug 1995), 790–799. doi:10.1109/34.400568. 5

[COSG*06] COHEN-OR D., SORKINE O., GAL R., LEYVAND T., XU Y.-Q.: Color harmonization. In *ACM SIGGRAPH 2006 Papers* (New York, NY, USA, 2006), SIGGRAPH '06, ACM, pp. 624–630. URL: http://doi.acm.org/10.1145/1179352.1141933, doi:10.1145/1179352.1141933. 2

[Del34] DELAUNAY B.: Sur la sphère vide. A la mémoire de Georges Voronoï. *Bulletin de l'Académie des Sciences de l'URSS*, 6 (1934), 793–800. 6

[Fit10] FITZPATRICK B.: Writing zippy android apps. Google I/O Developers Conference, May 2010. URL: https://youtu.be/c4znvD-7VDA. 9

[Fri37] FRIEDMAN M.: The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association 32*, 200 (1937), 675–701. URL: https://www.tandfonline.com/doi/abs/10.1080/01621459.1937.10503522, doi:10.1080/01621459.1937.10503522. 7

[Goo18] GOOGLE: Google arts & culture, 2018. URL: https://artsandculture.google.com/. 6

[Goo19] GOOGLE: Keeping your app responsive, 2019. URL: https://developer.android.com/training/articles/perf-anr.html. 9

[HAR*10] HANSEN N., AUGER A., ROS R., FINCK S., POŠÍK P.: Comparing results of 31 algorithms from the black-box optimization benchmarking bbob-2009. In *Proceedings of the 12th Annual Conference Companion on Genetic and Evolutionary Computation* (New York, NY, USA, 2010), GECCO '10, ACM, pp. 1689–1696. URL: http://doi.acm.org/10.1145/1830761.1830790, doi:10.1145/1830761.1830790. 4

[HMK03] HANSEN N., MÜLLER S. D., KOUMOUTSAKOS P.: Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (cma-es). *Evolutionary Computation 11*, 1 (2003), 1–18. doi:10.1162/106365603321828970. 4

[HN06] HAN L., NEUMANN M.: Effect of dimensionality on the nelderâĂŞmead simplex method. *Optim. Method. Soft. 21* (02 2006), 1–16. doi:10.1080/10556780512331318290. 4

[Jai10] JAIN A. K.: Data clustering: 50 years beyond k-means. *Pattern Recogn. Lett. 31*, 8 (June 2010), 651–666. URL: http://dx.doi.org/10.1016/j.patrec.2009.09.011, doi:10.1016/j.patrec.2009.09.011. 5

[Jol89] JOLLIFFE I. T.: *Principal Component Analysis*. Springer-Verlag, New York, 1989. doi:10.1007/b98835.

[Koh90] KOHONEN T.: The self-organizing map. *Proceedings of the IEEE 78*, 9 (Sep 1990), 1464–1480. doi:10.1109/5.58325. 5

[LFDH17] LIN S., FISHER M., DAI A., HANRAHAN P.: Layerbuilder: Layer decomposition for interactive image and video color editing. *CoRR abs/1701.03754* (2017). URL: http://arxiv.org/abs/1701.03754, arXiv:1701.03754. 2

[LH13] LIN S., HANRAHAN P.: Modeling how people extract color themes from images. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (New York, NY, USA, 2013), CHI '13, ACM, pp. 3101–3110. URL: http://doi.acm.org/10.1145/2470654.2466424, doi:10.1145/2470654.2466424. 2

[MH08] MAATEN L. V. D., HINTON G.: Visualizing data using t-sne. *Journal of machine learning research 9*, Nov (2008), 2579–2605. 5

[MVH*17] MELLADO N., VANDERHAEGHE D., HOARAU C., CHRISTOPHE S., BRÉDIF M., BARTHE L.: Constrained palette-space exploration. *ACM Trans. Graph. 36*, 4 (July 2017), 60:1–60:14. URL: http://doi.acm.org/10.1145/3072959.3073650, doi:10.1145/3072959.3073650. 2

[MW47] MANN H. B., WHITNEY D. R.: On a test of whether one of two random variables is stochastically larger than the other. *The Annals of Mathematical Statistics 18*, 1 (1947), 50–60. URL: http://www.jstor.org/stable/2236101. 7

[NM65] NELDER J. A., MEAD R.: A simplex method for function minimization. *The Computer Journal 7*, 4 (1965), 308–313. URL: http://dx.doi.org/10.1093/comjnl/7.4.308, doi:10.1093/comjnl/7.4.308. 4

[NRS15] NGUYEN C. H., RITSCHEL T., SEIDEL H.-P.: Data-driven color manifolds. *ACM Trans. Graph. 34*, 2 (Mar. 2015), 20:1–20:9. URL: http://doi.acm.org/10.1145/2699645, doi:10.1145/2699645. 2, 5

[OAH11] O'DONOVAN P., AGARWALA A., HERTZMANN A.: Color compatibility from large datasets. In *ACM SIGGRAPH 2011 Papers* (New York, NY, USA, 2011), SIGGRAPH '11, ACM, pp. 63:1–63:12. URL: http://doi.acm.org/10.1145/1964921.1964958, doi:10.1145/1964921.1964958. 2

[OAH14] O'DONOVAN P., AGARWALA A., HERTZMANN A.: Collaborative filtering of color aesthetics. In *Proceedings of the Workshop on Computational Aesthetics* (New York, NY, USA, 2014), CAe '14, ACM, pp. 33–40. URL: http://doi.acm.org/10.1145/2630099.2630100, doi:10.1145/2630099.2630100. 2

[PFC18] PHAN H., FU H., CHAN A.: Color orchestra: Ordering color palettes for interpolation and prediction. *IEEE Transactions on Visualization and Computer Graphics* (2018), 1–1. doi:10.1109/TVCG.2017.2697948. 3

[RS00] ROWEIS S. T., SAUL L. K.: Nonlinear dimensionality reduction by locally linear embedding. *Science 290*, 5500 (2000), 2323–2326. doi:10.1126/science.290.5500.2323. 5

[SCSI08] SIMAKOV D., CASPI Y., SHECHTMAN E., IRANI M.: Summarizing visual data using bidirectional similarity. In *2008 IEEE Conference on Computer Vision and Pattern Recognition* (June 2008), pp. 1–8. doi:10.1109/CVPR.2008.4587842. 3

[SKSF18] SHUGRINA M., KAR A., SINGH K., FIDLER S.: Color sails: Discrete-continuous palettes for deep color exploration. *CoRR abs/1806.02918* (2018). URL: http://arxiv.org/abs/1806.02918, arXiv:1806.02918. 2, 3

[SLD17] SHUGRINA M., LU J., DIVERDI S.: Playful palette: An interactive parametric color mixer for artists. *ACM Trans. Graph. 36*, 4 (July 2017), 61:1–61:10. URL: http://doi.acm.org/10.1145/3072959.3073690, doi:10.1145/3072959.3073690. 1, 4

[SOK*15] SON K., OH S. Y., KIM Y., CHOI H., BAE S.-H., HWANG G.: Color sommelier: Interactive color recommendation system based on community-generated color palettes. In *Adjunct Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology* (New York, NY, USA, 2015), UIST '15 Adjunct, ACM, pp. 95–96. URL: http://doi.acm.org/10.1145/2815585.2815736, doi:10.1145/2815585.2815736. 3

[TDLG18] TAN J., DIVERDI S., LU J., GINGOLD Y. I.: Pigmento: Pigment-based image analysis and editing. *IEEE Transactions on Visualization and Computer Graphics to appear* (July 2018), 1–14. doi:10.1109/TVCG.2018.2858238. 3

[TDSL00] TENENBAUM J. B., DE SILVA V., LANGFORD J. C.: A global geometric framework for nonlinear dimensionality reduction. *Science 290*, 5500 (2000), 2319–2323. doi:10.1126/science.290.5500.2319. 5

[TEG18] TAN J., ECHEVARRIA J., GINGOLD Y.: Efficient palette-based decomposition and recoloring of images via rgbxy-space geometry. *ACM Transactions on Graphics (TOG) 37*, 6 (Nov. 2018), 262:1–262:10. doi:10.1145/3272127.3275054. 2

[TLG16] TAN J., LIEN J.-M., GINGOLD Y.: Decomposing images into layers via rgb-space geometry. *ACM Trans. Graph. 36*, 1 (Nov. 2016), 7:1–7:14. URL: http://doi.acm.org/10.1145/2988229, doi:10.1145/2988229. 2, 3, 6, 8