Efficient palette-based decomposition and recoloring of images via RGBXY-space geometry

JIANCHAO TAN, George Mason University
JOSE ECHEVARRIA, Adobe Research
YOTAM GINGOLD, George Mason University

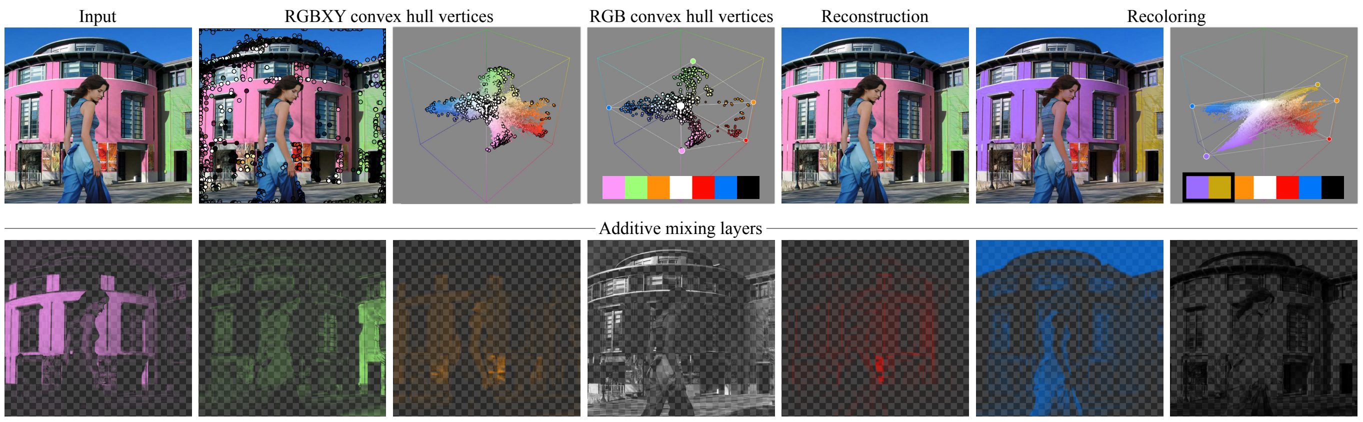


Fig. 1. Our approach automatically decomposes an input color image into a sparse set of uniformly-colored additive mixing layers. Our algorithm splits decomposition into a two-level geometric problem. The first level computes the 5D RGBXY convex hull and Delaunay tessellation of the input image pixels, which simultaneously considers color and spatial relationships. Its vertices are outlined in black; the 5D simplices are difficult to visualize. Any color in the image can be reproduced via a convex combination of these vertices. The second level computes an automatically-simplified RGB convex hull whose vertices serve as a color palette. Since the RGBXY convex hull vertices lie inside the RGB convex hull, we can find mixing weights that control the color of the RGBXY vertices, and therefore the entire image. Because the RGBXY vertices are unrelated to the palette, users can edit the palette and instantly obtain new additive mixing layers. These layers can be used to reconstruct or recolor the input image. Example image from Cohen-Or et al. [2006].

We introduce an extremely scalable and efficient yet simple palette-based image decomposition algorithm. Given an RGB image and set of palette colors, our algorithm decomposes the image into a set of additive mixing layers, each of which corresponds to a palette color applied with varying weight. Our approach is based on the geometry of images in RGBXY-space. This new geometric approach is orders of magnitude more efficient than previous work and requires no numerical optimization. We provide an implementation of the algorithm in 48 lines of Python code. We demonstrate a real-time layer decomposition tool in which users can interactively edit the palette to adjust the layers. After preprocessing, our algorithm can decompose 6 MP images into layers in 20 milliseconds.

CCS Concepts: • **Computing methodologies → Image manipulation**; **Image processing**;

Authors' addresses: Jianchao Tan, George Mason University, tanjianchaoustc@gmail.com; Jose Echevarria, Adobe Research, echevarr@adobe.com; Yotam Gingold, George Mason University, ygingold@gmu.edu.

Additional Key Words and Phrases: images, layers, painting, palette, generalized barycentric coordinates, convex hull, RGB, color space, recoloring, compositing, mixing

ACM Reference Format:
Jianchao Tan, Jose Echevarria, and Yotam Gingold. 2018. Efficient palette-based decomposition and recoloring of images via RGBXY-space geometry. *ACM Trans. Graph.* 37, 6, Article 262 (November 2018), 10 pages. https://doi.org/10.1145/3272127.3275054

1 INTRODUCTION

In digital image editing, images are often represented as a set of layers. This allows artists to organize their work and isolate modifications. The final image is obtained by compositing the layers with some mixing operation [Porter and Duff 1984]. A variety of recent approaches [Aksoy et al. 2017; Chang et al. 2015; Tan et al. 2016; Zhang et al. 2017] to image editing have explored the inverse problem, decomposing an image into a palette and associated per-pixel compositing or mixing parameters.

We propose an extremely efficient yet simple geometric approach for decomposing an image into spatially coherent additive mixing layers (Figure 1). In our approach, each output layer is a uniform color applied with varying weights. After an initial palette is extracted (given an RMSE reconstruction threshold), the user can edit the palette and obtain new decompositions instantaneously. This allows users to improve the decomposition by, for example, choosing semantically meaningful colors or trading reconstruction error for sparsity.

Our approach is inspired by the geometric palette extraction technique of Tan et al. [2016]. We consider the geometry of 5D RGBXY-space, which captures color as well as spatial relationships and eliminates numerical optimization. Our algorithm's performance is extremely efficient even for very high resolution images ($\geq$ 100 megapixels)—20x faster than the state-of-the-art [Aksoy et al. 2017]. Its performance is virtually independent from the size of the image or palette. After preprocessing, our decomposition can be *re-computed* instantaneously for a new RGB palette. This allows designers to edit the decomposition in real-time. Working code is provided in Section 3.

2 RELATED WORK

Image Decomposition For recoloring applications, it is also critical to find a mapping between the extracted color palette and the image pixels. Recent work is able to decompose the input image into separate layers according to a palette. Tan et al. [2016] extract a set of ordered translucent RGBA layers, based on a optimization over the standard alpha blending model. Order-independent decompositions can be achieved using additive color mixing models [Aksoy et al. 2017; Lin et al. 2017a; Zhang et al. 2017]. For the physically-based palette extraction methods mentioned previously [Aharoni-Mack et al. 2017; Tan et al. 2017], layers correspond to the extracted multispectral pigments. We prefer a full decomposition to a (palette-based) edit transfer approach like Chang et al. [2015]'s. With a full decomposition, edits are trivial to apply and spatial edits become possible (though we do not explore spatial edits in this work). We present a new, efficient method for layer decomposition, based on the additive color mixing model (Section 3.2). Our approach leverages 5D RGBXY-space geometry to enforce spatial smoothness on the layers. This geometric approach is significantly more efficient than previous approaches in the literature, easily handling images up to 100 megapixels in size.

Palette Extraction A straightforward approach consists of using a k-means method to cluster the existing colors in an image in RGB space [Chang et al. 2015; Nguyen et al. 2017; Phan et al. 2017; Zhang et al. 2017]. This captures the most prominent colors. A different approach consists of computing and simplifying the convex hull enclosing all the color samples [Tan et al. 2016], which provides more “primary” palettes that better represent the existing color gamut of the image. A similar observation was made in the domain of hyperspectral image unmixing [Craig 1994]. (With hyperspectral images, palette sizes are smaller than the number of channels, so the problem is one of fitting a minimum-volume simplex around the colors. The vertices of a high-dimensional simplex become a convex hull when the data is projected to lower dimensions.) Morse et al. [2007] work in HSL space, using a histogram to find the dominant hues, then to find shades and tints within them. Human perception has also been taken into account in other works, training regression models on crowd-sourced datasets. [Lin and Hanrahan 2013; O'Donovan et al. 2011]. Some physically-based approaches try to extract wavelength-dependent parameters to model the original pigments used paintings. [Aharoni-Mack et al. 2017; Tan et al. 2017]. Our work builds on top of Tan et al. [2016], adding a fixed reconstruction error threshold for automatic extraction of palettes of optimal size, as described in Section 3.1.

3 PALETTE EXTRACTION AND IMAGE DECOMPOSITION

A good palette for image editing is one that closely captures the underlying colors the image was made with (or could have been made with), even if those colors do not appear in their purest form in the image itself. Tan et al. [2016] observed that the color distributions from paintings and natural images take on a convex shape in RGB space. As a result, they proposed to compute the convex hull of the pixel colors. The convex hull tightly wraps the observed colors. Its vertex colors can be blended with convex weights (positive and summing to one) to obtain any color in the image. The convex hull may be overly complex, so they propose an iterative simplification scheme to a user-desired palette size. After simplification, the vertices become a palette that represents the colors in the image.

We extend Tan et al. [2016]'s work in two ways. First, we propose a simple, geometric layer decomposition method that is orders of magnitude more efficient than the state-of-the-art. Working code for our entire decomposition algorithm can be written in under 50 lines (Figure 3). Second, we propose a simple scheme for automatic palette size selection.

3.1 Palette Extraction

In Tan et al. [2016], the convex hull of all pixel colors is computed and then simplified to a user-chosen palette size. To summarize their approach, the convex hull is simplified greedily as a sequence of constrained edge collapses [Garland and Heckbert 1997]. An edge is collapsed to a point constrained to strictly add volume [Sander et al. 2000] while minimizing the distance to its incident faces. The edge whose collapse adds the least overall volume is chosen next, greedily. After each edge is collapsed, the convex hull is recomputed, since the new vertex could indirectly cause other vertices to become concave (and therefore redundant). Finally, simplification may result in out-of-gamut colors, or points that lie outside the RGB cube. As a final step, Tan et al. [2016] project all such points to the closest point on the RGB cube. This is the source of reconstruction error in their approach; some pixels now lie outside the simplified convex hull and cannot be reconstructed.

We improve upon this procedure with the observation that the reconstruction error can be measured geometrically, even before layer decomposition, as the RMSE of every pixel's distance to the simplified convex hull. (Inside pixels naturally have distance 0.) Therefore, we propose a simple automatic palette size selection based on a user-provided RMSE reconstruction error tolerance ($\eta = \frac{2}{255}$ in our experiments). For efficiency, we divide RGB-space into $32 \times 32 \times 32$ bins (a total of 2^{15} bins). We measure the distance from each non-empty bin to the simplified convex hull, weighted by the bin count. We start measuring the reconstruction error once the number of

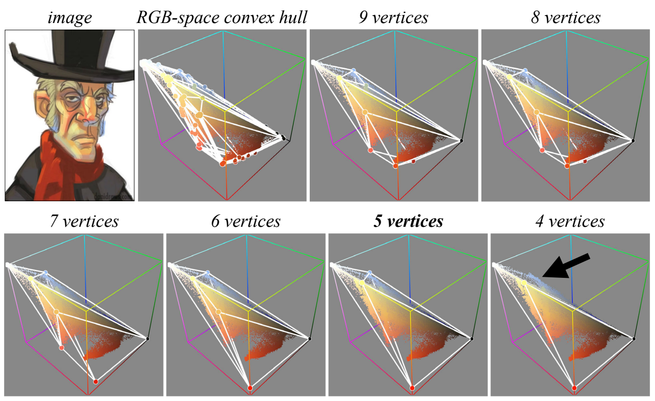


Fig. 2. In this example from Tan et al. [2016], our default RMSE tolerance η automatically simplifies the RGB-space convex hull down to the same 5 vertices manually chosen in their work. When simplified to 4 vertices, the blue pixels outside the polyhedron produce a large RMSE error. Image ©Dani Jones.

vertices has been simplified to 10. By doing this, we are able to obtain palettes with an optimal number of colors automatically. This removes the need for the user to choose the palette size manually, leading to better layer decompositions. Figure 2 visualizes different levels of simplified hulls for an image. For this example, our algorithm chooses a palette of size 5 according to our default RMSE tolerance.

(If non-constant palette colors were acceptable, instead of clipping one could cast a ray from each pixel towards the out-of-gamut vertex; the intersection of the ray with the RGB cube would be the palette color for that pixel. There would be zero reconstruction error. The stopping criteria could be the non-uniformity of a palette color, measured by the area of the RGB cube surface intersected with the simplified convex hull itself.)

3.2 Image decomposition via the RGBXY convex hull

From their extracted palettes, Tan et al. [2016] solved a non-linear optimization problem to decompose an image into a set of ordered, translucent RGBA layers suitable for the standard "over" compositing operation. While this decomposition is widely applicable (owing to the ubiquity of "over" compositing), the optimization is quite lengthy due to the recursive nature of the compositing operation, which manifests as a polynomial whose degree is the palette size. Others have instead opted for additive mixing layers [Aksoy et al. 2017; Lin et al. 2017a; Zhang et al. 2017] due to their simplicity. A pixel's color is a weighted sum of the palette colors.

In this work, we adopt additive mixing layers as well. We provide a fast and simple, yet spatially coherent, geometric construction. Any point $\mathbf{p}$ inside a simplex (a triangle in 2D, a tetrahedron in 3D, etc.) has a unique set of barycentric coordinates, or convex additive mixing weights such that $\mathbf{p} = \sum_i w_i \mathbf{c}_i$, where the mixing weights w_i are positive and sum to one, and $\mathbf{c}_i$ are the vertices of the simplex. In our setting, the simplified convex hull is typically not a simplex, because the palette has more than 4 colors. There still exist convex weights w_i for arbitrary polyhedron, known as generalized barycentric coordinates [Floater 2015], but they are non-unique. A straightforward technique to find generalized barycentric coordinates is to first compute a tessellation of the polyhedron (in our case, the simplified convex hull) into a collection of non-overlapping simplices (tetrahedra in 3D). For example, the *Delaunay generalized barycentric coordinates* for a point can be computed by performing a Delaunay tessellation of the polyhedron. The barycentric coordinates of whichever simplex the point falls inside are the generalized barycentric coordinates. For a 3D point in general position in the interior, the mixing weights will have at most 4 non-zero weights, which corresponds to the number of vertices of a tetrahedron.

This is the approach taken by Tan et al. [2016] for their *as-sparse-as-possible* (ASAP) technique to extract layers. Because Tan et al. [2016] considered recursive over compositing, users provided a layer or vertex order; they tessellated the simplified convex hull by connecting all its (triangular) faces to the first vertex, which corresponds to the background color. This simple *star tessellation* is valid for any convex polyhedron. In the additive mixing scenario, no order is provided; we discuss the choice of tessellation below. Because the weights are assigned purely based on the pixel's colors, however, this approach predictably suffers from spatial coherence artifacts. The colors of spatially neighboring pixels may belong to different tetrahedra. As a result, ASAP layers produce speckling artifacts during operations like recoloring (Figure 8).

Spatial Coherence To provide spatial coherence, our key insight is to extend this approach to 5D RGBXY-space, where XY are the coordinates of a pixel in image space, so that spatial relationship are considered along with color in a unified way (Figure 1). We first compute the convex hull of the image I in RGBXY-space:

$$V_{RGBXY} = \text{ConvexHull}(\{\ (R_i, G_i, B_i, X_i, Y_i) \mid i = 1, 2, \dots N\ \})$$

where V_{RGBXY} is the matrix whose columns are the Q vertices of the convex hull and i enumerates the N pixels of I. We then compute Delaunay generalized barycentric coordinates (weights) for every pixel in the image in terms of the 5D convex hull. To do this, we tessellate the RGBXY convex hull into a set of simplices:

$$\{S^j_{RGBXY}\} = \text{Delaunay\_tessellation}(V_{RGBXY}).$$

Mixing weights for a pixel i of the image in terms of the 5D convex hull vertices V_{RGBXY} can be computed as

$$\left(S^j_{RGBXY}\right)^{-1} [R_i, G_i, B_i, X_i, Y_i, 1]^\top$$

where pixel i is contained within simplex S^j_{RGBXY} and a simplex is represented as the 6×6 matrix whose columns are its vertices in homogeneous coordinates. With appropriate indexing (the 6 weights computed by the matrix product correspond to the 6 convex hull vertices referenced by S^j_{RGBXY}), we construct the sparse $N \times Q$ weight matrix W_{RGBXY}. Pixels that have similar colors *or* are spatially adjacent will end up with similar weights, meaning that our layers will be smooth both in RGB and XY-space. We can express our $N \times 3$ image via matrix multiplication as

$$I = W_{RGBXY} V_{RGB} \tag{1}$$

where V_{RGB} is a $Q \times 3$ submatrix of V_{RGBXY}.

```
from numpy import *
from scipy.spatial import ConvexHull, Delaunay
from scipy.sparse import coo_matrix

def RGBXY_weights( RGB_palette, RGBXY_data ):
    RGBXY_hull_vertices = RGBXY_data[ ConvexHull( RGBXY_data ).vertices ]
    W_RGBXY = Delaunay_coordinates( RGBXY_hull_vertices, RGBXY_data )
    # Optional: Project outside RGBXY_hull_vertices[:,:3] onto RGB_palette convex hull.
    W_RGB = Star_coordinates( RGB_palette, RGBXY_hull_vertices[:,:3] )
    return W_RGBXY.dot( W_RGB )

def Star_coordinates( vertices, data ):
    ## Find the star vertex
    star = argmin( linalg.norm( vertices, axis=1 ) )
    ## Make a mesh for the palette
    hull = ConvexHull( vertices )
    ## Star tessellate the faces of the convex hull
    simplices = [ [star] + list(face) for face in hull.simplices if star not in face ]
    barycoords = -1*ones( ( data.shape[0], len(vertices) ) )
    ## Barycentric coordinates for the data in each simplex
    for s in simplices:
        s0 = vertices[s[:1]]
        b = linalg.solve( (vertices[s[1:]]-s0).T, (data-s0).T ).T
        b = append( 1-b.sum(axis=1)[:,None], b, axis=1 )
        ## Update barycoords whenever data is inside the current simplex (with threshold).
        mask = (b>=-1e-8).all(axis=1)
        barycoords[mask] = 0.
        barycoords[ix_(mask,s)] = b[mask]
    return barycoords

def Delaunay_coordinates( vertices, data ): # Adapted from Gareth Rees
    # Compute Delaunay tessellation.
    tri = Delaunay( vertices )
    # Find the tetrahedron containing each target (or -1 if not found).
    simplices = tri.find_simplex(data, tol=1e-6)
    assert (simplices != -1).all() # data contains outside vertices.
    # Affine transformation for simplex containing each datum.
    X = tri.transform[simplices, :data.shape[1]]
    # Offset of each datum from the origin of its simplex.
    Y = data - tri.transform[simplices, data.shape[1]]
    # Compute the barycentric coordinates of each datum in its simplex.
    b = einsum( '...jk,...k->...j', X, Y )
    barycoords = c_[b,1-b.sum(axis=1)]
    # Return the weights as a sparse matrix.
    rows = repeat(arange(len(data)).reshape((-1,1)), len(tri.simplices[0]), 1).ravel()
    cols = tri.simplices[simplices].ravel()
    vals = barycoords.ravel()
    return coo_matrix( (vals,(rows,cols)), shape=(len(data),len(vertices)) ).tocsr()
```

Fig. 3. Python code for our RGBXY additive mixing layer decomposition (48 lines).

We similarly compute W_{RGB}, mixing weights for the RGBXY convex hull vertices V_{RGBXY} in terms of the RGB-space palette colors P. With P represented as a $\#P \times 3$ matrix,

$$V_{RGB} = W_{RGB} P. \tag{2}$$

W_{RGB} is a $Q \times \#P$ matrix. The palette colors are the simplified RGB convex hull vertices. We consider only the RGB portion of each RGBXY convex hull vertex, which always lies inside the *unsimplified* RGB convex hull. Due to the aforementioned out-of-gamut projection step when computing the simplified RGB convex hull, however, an RGBXY convex hull vertex may occasionally fall outside it. We set its weights to those of the closest point on the 3D simplified convex hull. This is the only source of reconstruction error.

By combining Equations 1 and 2, we can express our image $I = W_{RGBXY} W_{RGB} P$. The final weights are $W = W_{RGBXY} W_{RGB}$, which is an $N \times \#P$ matrix that assigns each pixel's weights solely in terms of the palette. These weights are smooth both in color and image space. To decompose an image with a different RGB palette, one only needs to recompute W_{RGB} and then perform matrix multiplication. Computing W_{RGB} is extremely efficient, since it depends only on the palette size and the number of RGBXY convex hull vertices. It is independent of the image size and allows users to experiment with image decompositions based on interactive palette editing (Figure 11 and the supplemental materials).

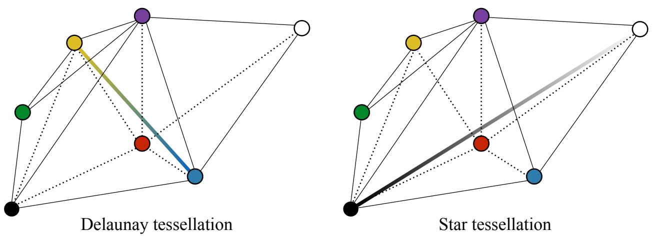


Fig. 4. The RGB-space convex polyhedron whose vertices are the palette colors. The Delaunay tessellation (left) prefers to add short edges. In contrast, our star tessellation (right) always includes the long, black-to-white edge when tessellating. This results in barycentric coordinates that adhere to our perceptual expectations: modifying yellow or blue should not affect pixels located along the line of greys. See Figure 5 for a comparison in image-space.

Tessellation At first glance, any tessellation of 3D RGB-space has approximately the same ℓ_0 weight sparsity (4 non-zeros). In practice, the "line of greys" between black and white is critically important. Any pixel near the line of greys can be expressed as the weighted combination of vertices in a number of ways (e.g. any tessellation). It is perceptually important that the line of greys be 2-sparse in terms of an approximately black and white color, and that nearby colors be nearly 2-sparse. If not, then grey pixels would be represented as mixtures of complementary colors; any change to the palette that doesn't preserve the complementarity relationships would turn grey pixels colorful (Figure 5 and 8). This tinting is perceptually prominent and undesirable.[1] As illustrated in Figure 4, our star tessellation always includes the long black-to-white edge when tessellating the volume. In contrast, the Delaunay tessellation prefers short edges.

To make the line of greys 2-sparse in this way, the tessellation should ensure that an edge is created between the darkest and lightest color in the palette. Such an edge is typically among the longest possible edges through the interior of the polyhedron, as the luminance in an image often varies more than chroma × hue. As a result, the Delaunay tessellation frequently excludes the most desirable edge through the line of greys. We propose to use a star tessellation. If either a black or white palette color is chosen as the star vertex, it will form an edge with the other. We choose the darkest color in the palette as the star vertex. This strategy is simple and predictable and extends naturally to premultiplied alpha RGBA images.

We also experimented with a variety of strategies to choose the tessellation such that the resulting layer decomposition is as sparse as possible: RANSAC line fitting and PCA on the RGB point cloud and finding the longest edge. We evaluated the decompositions with several sparsity metrics ([Aksoy et al. 2017; Levin et al. 2008; Tan et al. 2016], as well as the fraction of pixels with transparency above a threshold). Ultimately, tinting was more perceptually salient than changes in sparsity, and our proposed tessellation algorithm is simpler, more efficient, and more predictable.

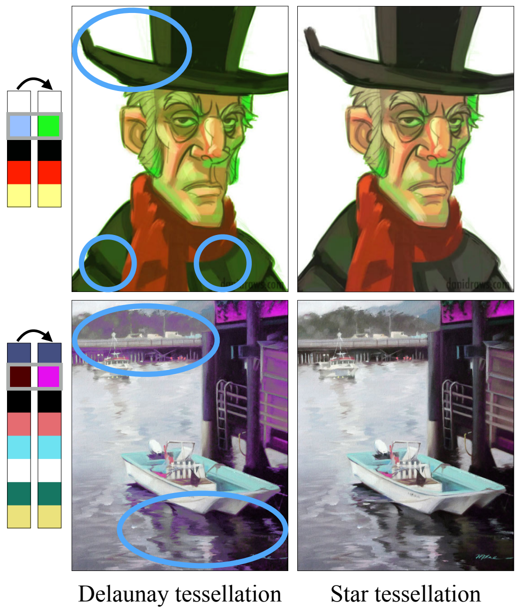


Fig. 5. The Delaunay tessellation often does not tessellate color-space with an edge along the line of greys. As a result, grey pixels are decomposed into a mixture of complementary colors. Modifying any one of them will turn grey pixels colorful. Our star tessellation always includes an edge along the line of greys, meaning that grey pixels are mixtures of black and white and remain unchanged when modifying other colors. The original images and their star tessellation layers can be found in Figures 6 and 12. ©Dani Jones (top); Michelle Lee (bottom).

4 EVALUATION

Quality The primary means to assess the quality of layers is to apply them for some purpose, such as recoloring, and then identify artifacts, such as noise, discontinuities, or surprisingly affected regions. Figures 7 and 13 compare recolorings created with our layers to those from Aksoy et al. [2017], Tan et al. [2016], and Chang et al. [2015]. Our approach generates recolorings without discontinuities (the sky in 7(b), second row), undesirable changes (the top of the chair in 7(c), third row), or noise.

We have no explicit guarantees about the sparsity of our weights. W_{RGB} is as sparse as possible to reconstruct 3D colors (4 non-zeros per row). W_{RGBXY} has 6 non-zeros per row out of the (typically) 2000–5000 RGBXY convex hull vertices, which is also as sparse as possible to recover a point in RGBXY-space. The sparsity of the product of the two matrices depends on which 3D tetrahedra the 6 RGBXY convex hull vertices fall into. Nevertheless, it can be seen that our results' sparsity is almost as good as Tan et al. [2016].

Figures 6 and 12 visualize our additive mixing layers and those of Aksoy et al. [2017] for direct inspection. In contrast with our approach, Aksoy et al. [2017]'s approach has trouble separating colors that appear primarily in mixture. As a result, Aksoy et al. [2017]'s approach sometimes creates an overabundance of layers, which makes recoloring tedious, since multiple layers must be edited.

Our decomposition algorithm is able to reproduce input images virtually indistinguishably from the originals. For the 100 images in Figure 9, our RGBXY method's RGB-space RMSE is typically 2–3. Aksoy et al. [2017]'s algorithm reconstruct images with zero error, since their palettes are color distributions rather than fixed colors.

We evaluate our RGB tessellation in Figure 8. In this experiment, we generate a random recoloring by permuting the colors in the palette. The RGB-space star triangulation approach is akin to Tan et al. [2016]'s ASAP approach with the black color chosen to be the first layer. The lack of spacial smoothness is apparent in between the RGB-only decompositions in the top-row and the RGBXY decompositions in the bottom row. The decompositions using the Delaunay generalized barycentric coordinates (left column) result in undesirable tinting for colors near the line of grey. Additional examples can be found in the supplemental materials.

Throughout the remainder of the paper, all our results rely on our proposed layer decomposition.

Speed In Figure 9, we compare the running time of additive mixing layer decomposition techniques. We ran our proposed RGBXY approach on 100 images under 6 megapixels with an average palette size of 6.95 and median palette size of 7. Computation time for our approaches includes palette selection (RGB convex hull simplification). Because of its scalability, we also ran our proposed RGBXY approach on an additional 70 large images between 6 and 12 megapixels, and an additional 6 extremely large images containing 100 megapixels (not shown in the plot). The 100 megapixel images took on average 12.6 minutes to compute. Peak memory usage was 15 GB. For further improvement, our approach could be parallelized by dividing the image into tiles, since the convex hull of a set of convex hulls is the same as the convex hull of the underlying data. A working implementation of the RGBXY decomposition method can be found in Figure 3 (48 lines of code). The "Layer Updating" performance is nearly instantaneous, taking a few milliseconds to, for 10 MP images, a few tens of milliseconds to re-compute the layer decomposition given a new palette.

Our running times were generated on a 2015 13" MacBook Pro with a 2.9 GHz Intel Core i5-5257U CPU and 16 GB of RAM. Our layer decomposition approach was written in non-parallelized Python using NumPy/SciPy and their wrapper for the QHull convex hull and Delaunay tessellation library [Barber et al. 1996]. Our layer updating was written in OpenCL.

Aksoy et al. [2017]'s performance is the fastest previous work known to us. The performance data for Aksoy et al.'s algorithm is as reported in their paper and appears to scale linearly in the pixel size. Their algorithm was implemented in parallelized C++. Aksoy et al. [2017] reported that their approach took 4 hours and 25 GB of memory to decompose a 100 megapixel image. Zhang et al. [2017]'s sole performance data point is also as reported in their paper.

We also compare our approach to a variant of Tan et al. [2016]'s optimization. We modified their reconstruction term to the simpler, quadratic one that matches our additive mixing layer decomposition scenario. With that modification, all energy terms become quadratic. However, because the sparsity term is not positive definite, it is not

[1]For pathological images containing continuous gradients between complementary colors, this tinting behavior would perhaps be desired.

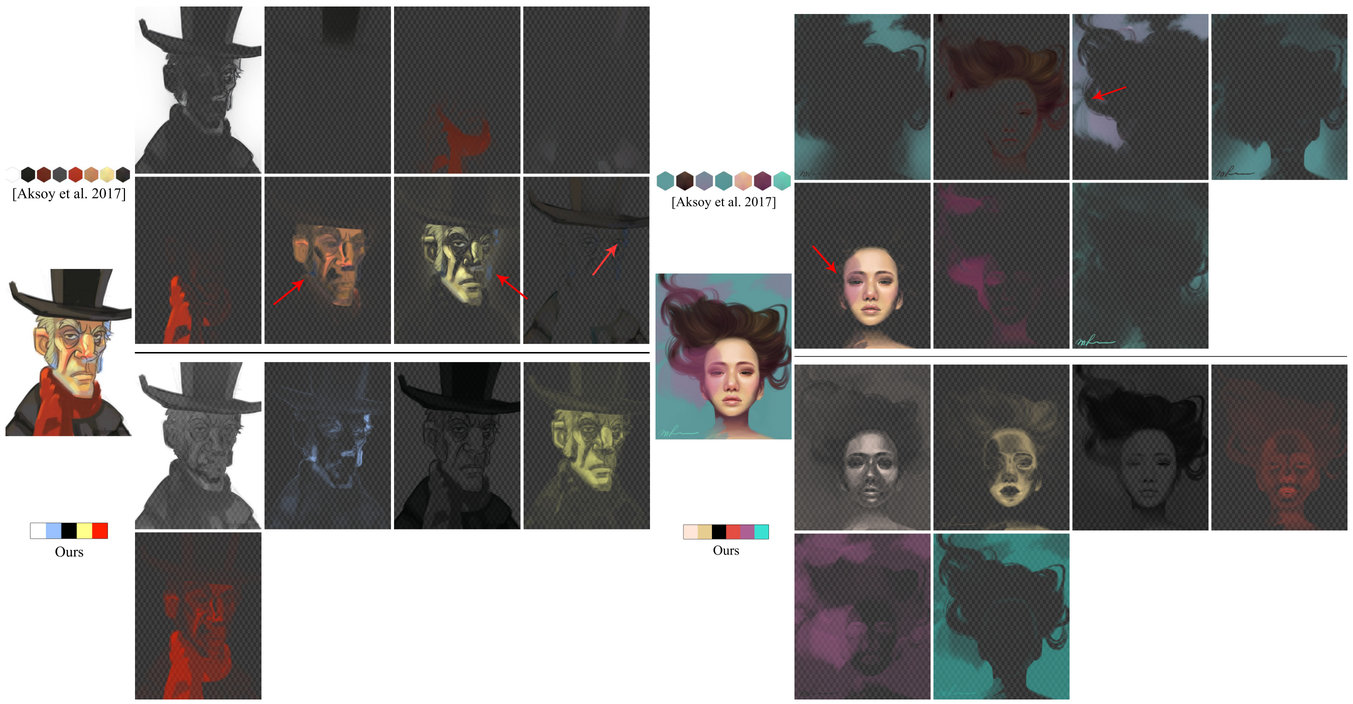


Fig. 6. A comparison between our proposed RGBXY image decomposition and that of Aksoy et al. [2017]. Aksoy et al. [2017] creates an overabundance of layers (two red layers above) and does not extract the blueish tint, which appears primarily in mixture. Our RGBXY technique identifies mixed colors is able to separate the translucent purple haze in front of the girl's face. See Figure 13 for recolored images created from these layers. Additionally, our GUI allows editing the palette to modify layers in real time. This allows users to further improve the decomposition, as shown in Figure 11 and the supplemental materials. ©Dani Jones (left); Michelle Lee (right).

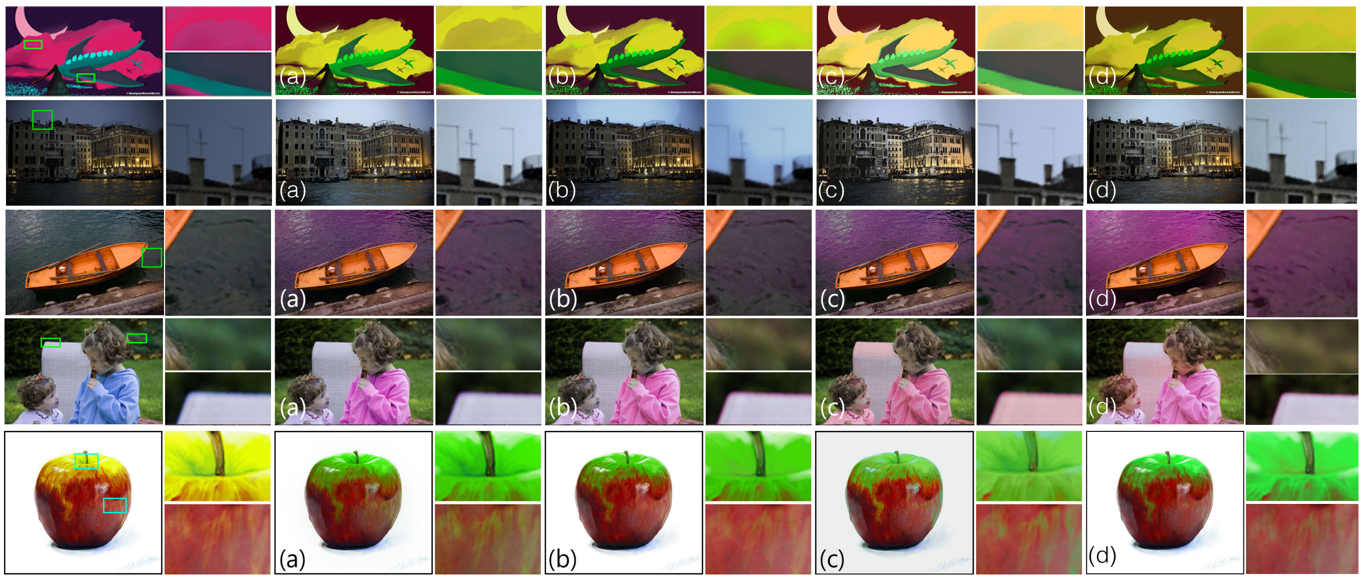


Fig. 7. To evaluate our RGBXY decomposition algorithm, we compare our layers with previous approaches in a recoloring application, extending a figure from Aksoy et al. [2017]. From left to right: (a) Aksoy et al. [2017], (b) Tan et al. [2016], (c) Chang et al. [2015] and (d) our approach. Our recoloring quality is similar to the state of the art, but our method is orders of magnitude faster and allows interactive layer decomposition while editing palettes. The decomposed layers themselves are shown in Figure 12. ©Karl Northfell (top row); Bychkovsky et al. [2011] (middle three rows); Adelle Chudleigh (bottom row).

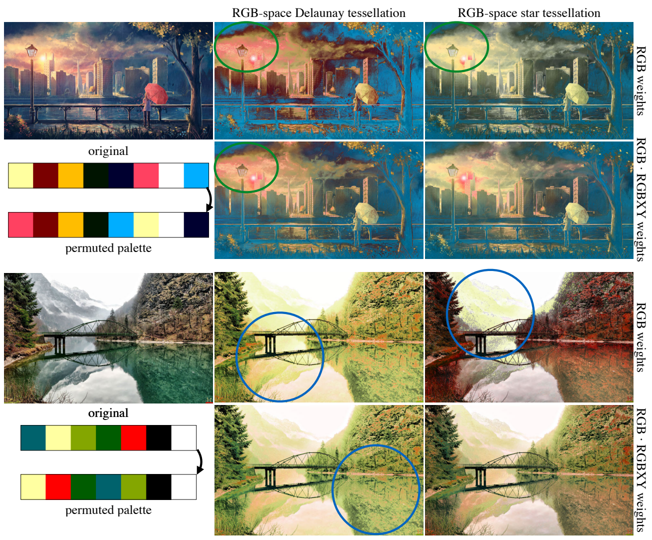


Fig. 8. Comparing tessellation strategies for color palettes in RGB-space. The Delaunay tessellation column computes Delaunay barycentric coordinates for the color palette. This tessellation often avoids creating the perceptually important line of greys, leading to tinting artifacts. These are avoided with a star tessellation emanating from the vertex closest to black. See also Figure 5. Computing weights in RGB-space alone leads to spatial smoothness artifacts. Our two-stage RGBXY decomposition provides color and spatial smoothness. To interrogate the quality of layer decompositions, we randomly permute the palette, revealing problems in computed weights. See the supplemental materials for additional examples. ©DeviantArt user Sylar113 (top); Fabio Bozzone (bottom).

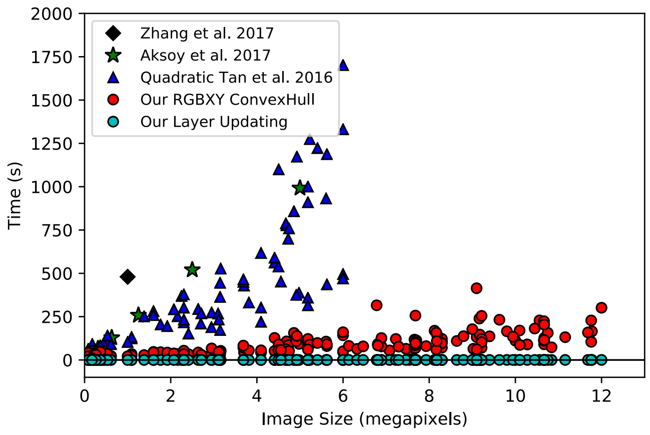
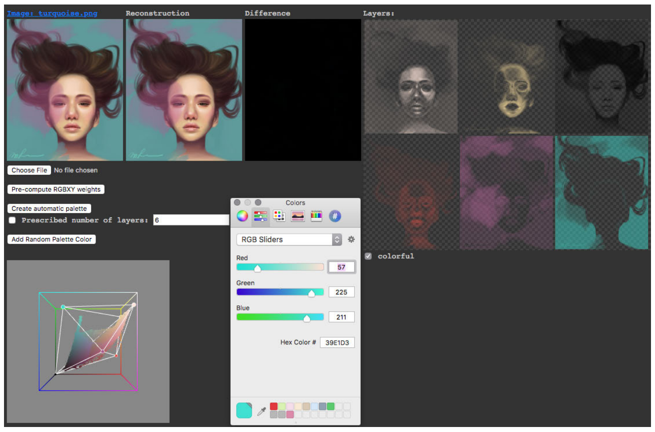


Fig. 9. Running time comparison between four additive mixing image decomposition algorithms. We evaluated our RGBXY algorithm on 170 images up to 12 megapixels and an additional six 100 megapixel images (not shown; average running time 12.6 minutes). Our algorithm's performance is orders of magnitude faster and scales extremely well with image size. The number of RGBXY convex hull vertices has a greater effect on performance than image size. Re-computing our layer decomposition with an updated palette is nearly instantaneous (a few to tens of milliseconds).



Prescribed number of layers: 6

Red 57

Green 225

Blue 211

Hex Color # 39E1D3

Fig. 10. Our GUI for interactively editing palettes. See the text for details.

a straightforward Quadratic Programming problem; we minimize it with L-BFGS-B and increased the solver's default termination thresholds since RGB colors have low precision (gradient and function tolerance 10^{-4}). This approach was also written in Python using NumPy/SciPy. The performance of the modified Tan et al. [2016] is somewhat unpredictable, perhaps owing to the varying palette sizes.

The fast performance of our approach is due to the fact that the number of RGBXY convex hull vertices Q is virtually independent of the image size and entirely independent of the palette size. Finding the simplex that contains a point is extremely efficient (a matrix multiply followed by a sign check) and scales well. Our algorithm's performance is more correlated with the number of RGBXY convex hull vertices and tessellated simplices. This explains the three red dots somewhat above the others in the performance plot.

In contrast, optimization-based approaches typically have parameters to tune, such as the balance between terms in the objective function, iteration step size, and termination parameters.

Interactive Layer Decompositions To take advantage of our extremely fast layer decomposition, we implemented an HTML/JavaScript GUI for viewing and interacting with layer decompositions (Figure 10). An editing session begins when a user loads an image and precomputes RGBXY weights. Users can then begin with a generic tetrahedron or with an automatically chosen palette, optionally with a prescribed number of layers. Users can alter the palette colors in 3D RGB-space (lower-left) or activate a traditional color picker by clicking on a layer (the turquoise layer as shown). As users drag the palette colors, the layer decomposition updates live. (Although our layer updating algorithm computes at an extremely high frame rate, the bulk of the time in our GUI is spent transferring the data to the browser via a WebSocket.) Users can also add and then manipulate additional colors. See Figure 11 for a result created with our GUI; see the supplemental materials for screen recordings of this and other examples.

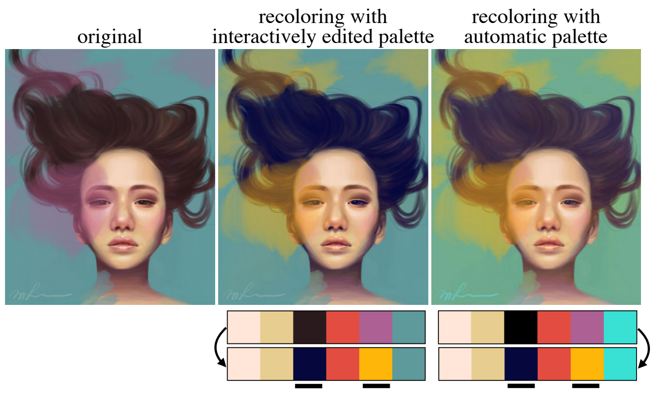

original
recoloring with interactively edited palette
recoloring with automatic palette

Fig. 11. Our GUI allows users edit palettes and see the resulting layer decomposition in real-time. Videos of live palette editing can be found in the supplemental materials. In this example, the automatic palette (right) becomes sparser as a result of interactive editing. The user edits the automatically generated palette to ensure that the background and hair colors are directly represented. As a result, editing the purple haze and hair no longer affects the background color. ©Michelle Lee.

5 CONCLUSION

We presented an extremely efficient approach for decomposing an image into spatially coherent additive mixing layers via RGBXY space geometry. Our approach produces high quality results and is extremely simple to implement. We achieve this implementation simplicity and execution speed by making use of well-studied geometric algorithms for computing the convex hull and Delaunay tessellation of a set of points.

5.1 Limitations

During our performance tests for the image decomposition, we found isolated cases where the computation of the 5D convex hull takes somewhat longer than usual. We believe it is due to very specific color distributions (3 out of 170 tested images), but we would like to study the phenomenon in more depth.

Our star tessellation assumes that the palette colors are vertices of a convex polyhedron. In particular, it cannot be used when some palette colors lie in the interior of the convex hull. When this happens, one could fall back on a Delaunay tessellation, which may not create an edge along the line of greys. The line of greys could be maintained with a constrained tetrahedralization algorithm, though these are complex and may add new, undesired vertices [Yang et al. 2005].

5.2 Future work

Image decomposition Inspired by Lin et al. [2017b], we wish to explore the use of superpixels to see if we are able to achieve greater speed ups. We also wish to explore parallel and approximate convex hull algorithms. An algorithm that produces a smaller, approximate convex hull containing only a certain percentile of points could provide an intuitive parameter for balancing reconstruction error with sparsity.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their suggestions. We are also grateful to Yağız Aksoy for providing comparison results; to Bei Xiao for useful discussions; and to the artists who graciously gave us permission to analyze their artwork.

This work was supported by the United States National Science Foundation (IIS-1453018), a Google research award, and a gift from Adobe Systems Inc.

REFERENCES

Elad Aharoni-Mack, Yakov Shambik, and Dani Lischinski. 2017. Pigment-Based Recoloring of Watercolor Paintings. In *NPAR*.

Yağiz Aksoy, Tunç Ozan Aydin, Aljoša Smolić, and Marc Pollefeys. 2017. Unmixing-based soft color segmentation for image manipulation. *ACM Transactions on Graphics (TOG)* 36, 2 (2017), 19.

C. Bradford Barber, David P. Dobkin, and Hannu Huhdanpaa. 1996. The Quickhull Algorithm for Convex Hulls. *ACM Trans. Math. Softw.* 22, 4 (Dec. 1996), 469–483.

Vladimir Bychkovsky, Sylvain Paris, Eric Chan, and Frédo Durand. 2011. Learning Photographic Global Tonal Adjustment with a Database of Input / Output Image Pairs. In *Computer Vision and Pattern Recognition (CVPR)*.

Huiwen Chang, Ohad Fried, Yiming Liu, Stephen DiVerdi, and Adam Finkelstein. 2015. Palette-based Photo Recoloring. *ACM Trans. Graph.* 34, 4 (July 2015).

Daniel Cohen-Or, Olga Sorkine, Ran Gal, Tommer Leyvand, and Ying-Qing Xu. 2006. Color Harmonization. In *ACM SIGGRAPH 2006 Papers (SIGGRAPH '06)*. ACM, New York, NY, USA, 624–630. https://doi.org/10.1145/1179352.1141933

Maurice D Craig. 1994. Minimum-volume transforms for remotely sensed data. *IEEE Transactions on Geoscience and Remote Sensing* 32, 3 (May 1994), 542–552. https://doi.org/10.1109/36.297973

Michael S Floater. 2015. Generalized barycentric coordinates and applications. *Acta Numerica* 24 (2015), 161–214.

Michael Garland and Paul S. Heckbert. 1997. Surface Simplification Using Quadric Error Metrics. 209–216.

Anat Levin, Alex Rav-Acha, and Dani Lischinski. 2008. Spectral matting. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 30, 10 (2008), 1699–1712.

Sharon Lin, Matthew Fisher, Angela Dai, and Pat Hanrahan. 2017a. LayerBuilder: Layer Decomposition for Interactive Image and Video Color Editing. *arXiv preprint arXiv:1701.03754* (2017).

Sharon Lin, Matthew Fisher, Angela Dai, and Pat Hanrahan. 2017b. LayerBuilder: Layer Decomposition for Interactive Image and Video Color Editing. *CoRR* abs/1701.03754 (2017). arXiv:1701.03754 http://arxiv.org/abs/1701.03754

Sharon Lin and Pat Hanrahan. 2013. Modeling how people extract color themes from images. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 3101–3110.

Bryan S Morse, Daniel Thornton, Qing Xia, and John Uibel. 2007. Image-based color schemes. In *Image Processing, 2007. ICIP 2007. IEEE International Conference on*, Vol. 3. IEEE, III–497.

R.M.H. Nguyen, B. Price, S. Cohen, and M. S. Brown. 2017. Group-Theme Recoloring for Multi-Image Color Consistency. *Computer Graphics Forum* 36, 7 (2017), 83–92. https://doi.org/10.1111/cgf.13274 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.13274

Peter O'Donovan, Aseem Agarwala, and Aaron Hertzmann. 2011. Color compatibility from large datasets. *ACM Transactions on Graphics (TOG)* 30, 4 (2011), 63.

Huy Phan, Hongbo Fu, and Antoni Chan. 2017. Color Orchestra: Ordering Color Palettes for Interpolation and Prediction. *IEEE Transactions on Visualization and Computer Graphics* (2017).

Thomas Porter and Tom Duff. 1984. Compositing Digital Images. *ACM SIGGRAPH Computer Graphics* 18, 3 (1984), 253–259.

Pedro V. Sander, Xianfeng Gu, Steven J. Gortler, Hugues Hoppe, and John Snyder. 2000. Silhouette Clipping. In *Proceedings of ACM SIGGRAPH*. 327–334.

Jianchao Tan, Stephen DiVerdi, Jingwan Lu, and Yotam Gingold. 2017. Pigmento: Pigment-Based Image Analysis and Editing. *arXiv preprint arXiv:1707.08323* (2017).

Jianchao Tan, Jyh-Ming Lien, and Yotam Gingold. 2016. Decomposing Images into Layers via RGB-space Geometry. *ACM Trans. Graph.* 36, 1, Article 7 (Nov. 2016), 14 pages. https://doi.org/10.1145/2988229

Yi-Jun Yang, Jun-Hai Yong, and Jia-Guang Sun. 2005. An algorithm for tetrahedral mesh generation based on conforming constrained Delaunay tetrahedralization. *Computers & Graphics* 29, 4 (2005), 606–615.

Qing Zhang, Chunxia Xiao, Hanqiu Sun, and Feng Tang. 2017. Palette-Based Image Recoloring Using Color Decomposition Optimization. *IEEE Transactions on Image Processing* 26, 4 (2017), 1952–1964.

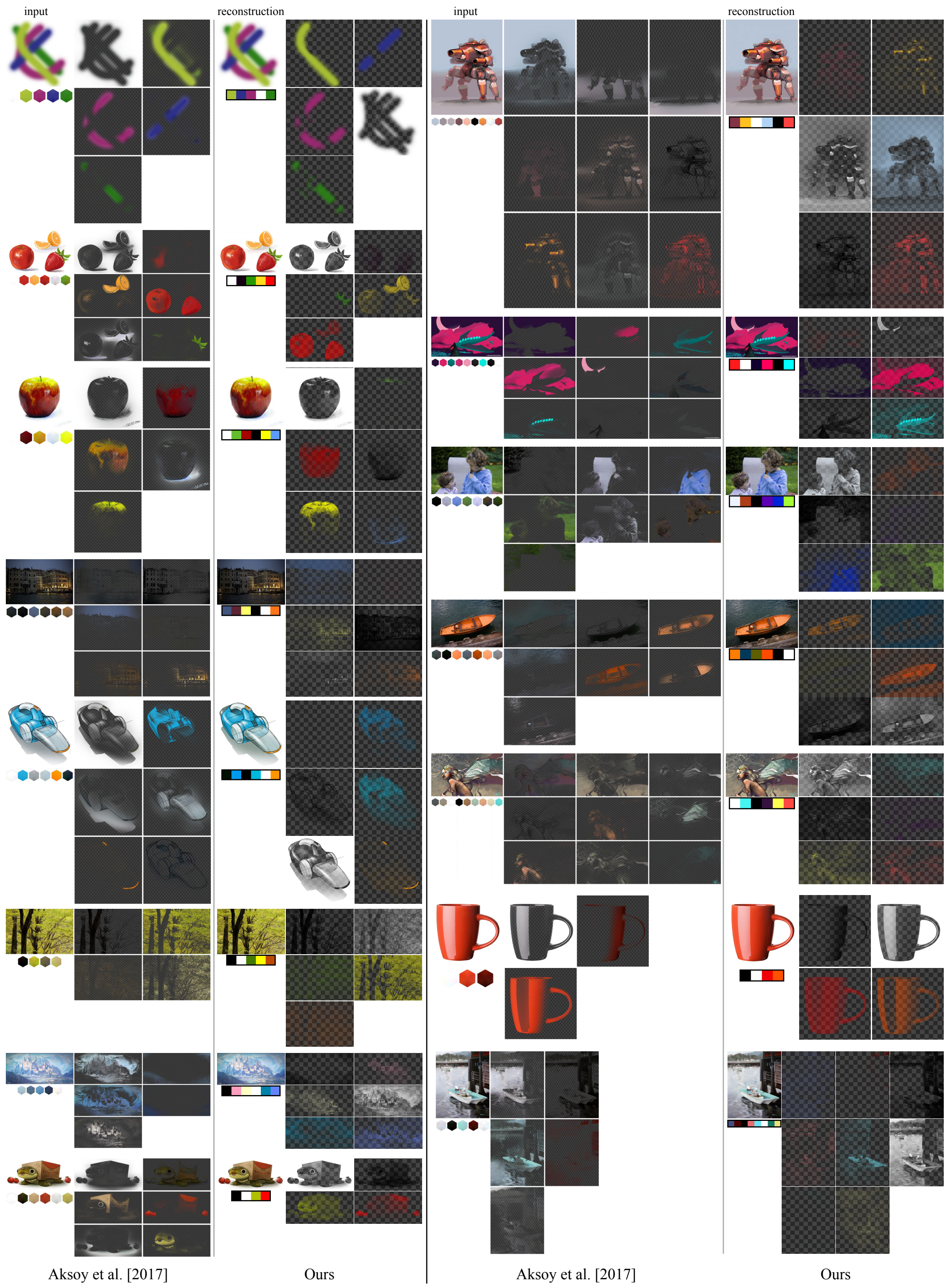


Fig. 12. Additive mixing layers produced using our approach and the approach of Aksoy et al. [2017]. Images labeled "reconstruction" are reconstructed using our layers. Top-to-bottom, left-to-right: ©Yotam Gingold; DeviantArt user Ranivius; Adelle Chudleigh; Bychkovsky et al. [2011]; Spencer Nugent; Bychkovsky et al. [2011]; Michelle Lee; Piper Thibodeau; Adam Saltsman; Karl Northfell; Bychkovsky et al. [2011]; Bychkovsky et al. [2011]; Michelle Lee; George Dolgikh; Michelle Lee.

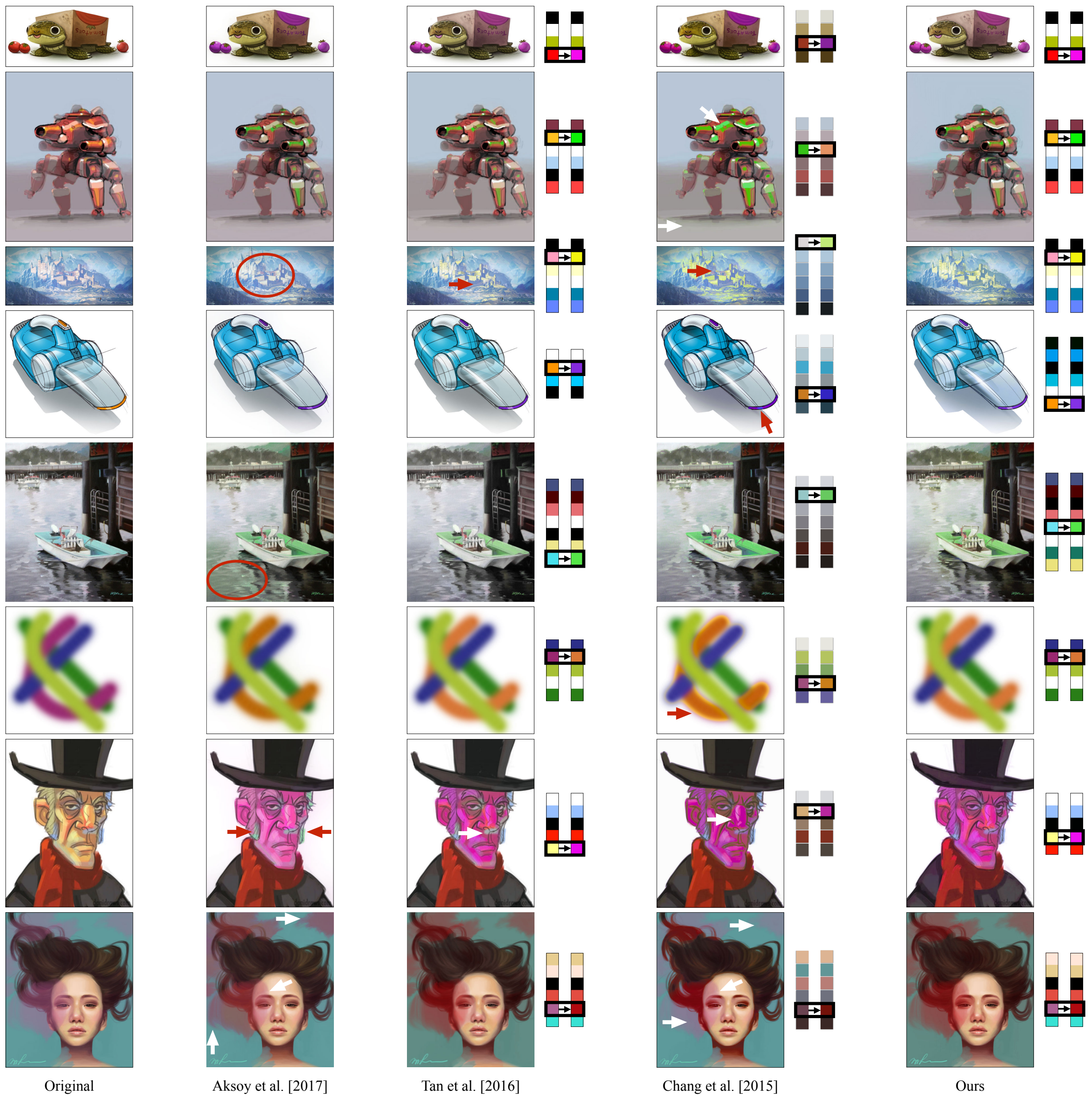


Fig. 13. A comparison of image recoloring techniques. Artifacts are marked with arrow and circles. We recolored each image using our method's layers, and then tried to obtain similar recolorings using other approaches' palettes and decompositions. The results for Aksoy et al. [2017], which is distribution rather than palette-based, were provided by that papers' authors. Top-to-bottom: ©Piper Thibodeau; Adam Saltsman; Michelle Lee; Spencer Nugent; Michelle Lee; Yotam Gingold; Dani Jones; Michelle Lee.