Decomposing Images into Layers via RGB-Space Geometry

JIANCHAO TAN, JYH-MING LIEN, and YOTAM GINGOLD
George Mason University

In digital image editing software, layers organize images. However, layers are often not explicitly represented in the final image, and may never have existed for a scanned physical painting or a photograph. We propose a technique to decompose an image into layers. In our decomposition, each layer represents a single-color coat of paint applied with varying opacity. Our decomposition is based on the image's RGB-space geometry. In RGB-space, the linear nature of the standard Porter-Duff [1984] "over" pixel compositing operation implies a geometric structure. The vertices of the convex hull of image pixels in RGB-space correspond to a palette of paint colors. These colors may be "hidden" and inaccessible to algorithms based on clustering visible colors. For our layer decomposition, users choose the palette size (degree of simplification to perform on the convex hull), as well as a layer order for the paint colors (vertices). We then solve a constrained optimization problem to find translucent, spatially coherent opacity for each layer, such that the composition of the layers reproduces the original image. We demonstrate the utility of the resulting decompositions for recoloring (global and local) and object insertion. Our layers can be interpreted as generalized barycentric coordinates; we compare to these and other recoloring approaches.

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

Additional Key Words and Phrases: Images, colors, layers, RGB, Photoshop

1. INTRODUCTION

Digital painting software simulates the act of painting in the real world. Artists choose paint colors and apply them with a mouse or drawing tablet by painting with a virtual brush. These virtual brushes have varying opacity profiles, which control how the paint color blends with the background. Digital painting software typically provides the ability to create layers, which are composited to form the final image yet can be edited separately. Layers *organize* images. However, layers are often not explicitly represented in a final digital image. Moreover, scanned physical paintings and photographs do not have such layers. Without layers, simple edits become extremely challenging, such as altering the color of a coat without inadvertently affecting a scarf placed over top.

We propose a technique to decompose any image into layers (Figure 1). In our decomposition, each layer represents a coat of paint of a single color applied with varying opacity throughout the image. We use the standard Porter-Duff [1984] "A over B" compositing operation:

$$(A \text{ over } B)_{RGB} = A_\alpha A_{RGB} + (1 - A_\alpha) B_\alpha B_{RGB} \qquad (1)$$

$$(A \text{ over } B)_\alpha = A_\alpha + (1 - A_\alpha) B_\alpha, \qquad (2)$$

where pixel A with color A_{RGB} and translucency A_α is placed over pixel B with color B_{RGB} and translucency B_α. Users choose the size of the desired palette, and our technique automatically extracts paint colors. Given a user-provided *order* for the colors, our technique computes per-pixel opacity for each layer.[1] The result is a sequence of layers that reproduce the original painting when composited.

In digital painting programs, the aforementioned "over" compositing operation is the standard way to apply virtual paint. In RGB-space, the pixels of such a painting reveal a hidden geometric structure (Figure 2). This structure results from the linearity of the "over" compositing operation. The paint color acts as a linear attractor in RGB-space. Affected pixels move toward the paint color via linear interpolation; the paint's transparency determines the strength of attraction, or interpolation parameter. All possible image colors are convex combinations of the paint colors. In RGB-space, all possible image colors lie within the *convex hull* of the paint colors (Figures 2 and 4). Our approach for decomposing any image into layers is based on this observation. Our algorithm is agnostic as to how a pixel achieved its color and does not require or assume that pixels were created via "over" compositing. For example, physical paint compositing can be approximated by the Kubelka-Munk layering and mixing models [Kubelka and Munk 1931; Kubelka 1948]. However, no simple model can describe images in general. Our goal is to decompose an image into useful layers. We choose Porter-Duff "over" compositing for our output representation because it is the de facto standard for image editing. For images created with "over" compositing, the recovered layers may be similar to ground truth (Figure 4), though we do not expect this in general. For images best described by another model, the recovered layers will not be as sparse or clean as they could be with a technique that operated in the parameters of the unknown model.

Overview. The first stage in our pipeline identifies a small **color palette** capable of generating all colors in an image (Section 3). We

---

Authors' addresses: J. Tan, J.-M. Lien, and Y. Gingold, Department of Computer Science, George Mason University, 4400 University Drive MS 4A5, Fairfax, VA 22030; emails: jtan8@gmu.edu, jmlien@cs.gmu.edu, ygingold@gmu.edu.

[1]The ordering is necessary, because the "over" compositing operation is not commutative.
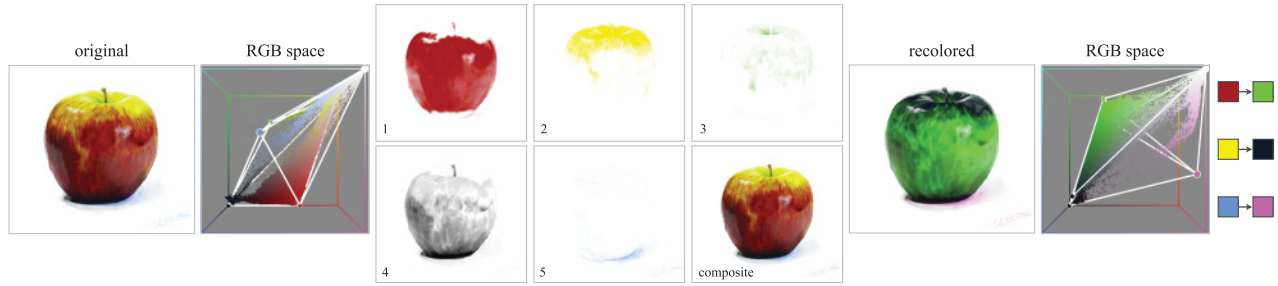
Fig. 1. Given a digital painting, we analyze the geometry of its pixels in RGB-space, resulting in a translucent layer decomposition, which makes difficult edits simple to perform. Original artwork © Adelle Chudleigh.
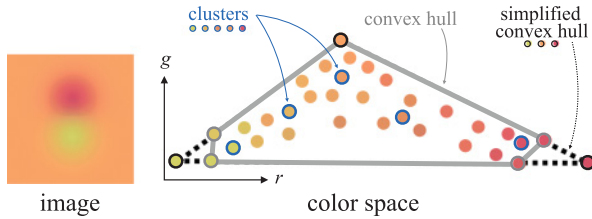

Fig. 2. In RGB-space, the pixels of a digital image (left) lie in the convex hull of the original paint colors (right). This is due to the linearity of the standard "over" blending operation [Porter and Duff 1984]. Our approach computes a simplified convex hull to find a small set of generating colors for decomposing the image into layers. This simplified convex hull often reveals "hidden" colors inaccessible to algorithms based on clustering.

compute the exact convex hull of the image colors in RGB-space. Its vertices are capable of reproducing any color in the image. However, this exact convex hull is a tight wrapping of the image colors and typically has too many vertices. (As these vertices are the color palette, having too many vertices produces an unmanageable number of layers for the user.) Therefore, we simplify the convex hull to a user-specified palette size (vertex count) that still encloses the image colors. This often reveals "hidden" colors inaccessible to algorithms based on clustering. Our simplification is based on the progressive hull algorithm [Sander et al. 2000]. Naively, a simplified convex hull may have vertices outside the cube of valid RGB colors. We adjust the simplified vertices to find real (as opposed to imaginary) colors.

In the second stage of our pipeline, we compute per-pixel **layer opacities,** given a user-specified layer ordering of the paint colors (Section 4). This results in RGBA layers that, when composited, reproduce the input image. Each layer models a coat of paint. Our computation solves an underconstrained polynomial system of equations. The polynomial equations express the constraint that the "over" composition of all layers reproduces the input image. The system is underconstrained because, in general, there are multiple ways to paint a pixel to arrive at a given color (Figure 6). The system is only well defined when there are exactly four paint colors (a tetrahedron) and the desired color is in the interior. This is related to barycentric coordinates only being unique for a point inside a simplex—tetrahedron in 3D.[2] To solve this

[2]A point inside a polyhedron with more than four vertices lies in more than one tetrahedron (whose vertices are a subset of the polyhedron's). To see this, consider that we can always tetrahedralize the polyhedron's interior (e.g., via a Delauney tetrahedralization) to obtain one set of barycentric

undterconstrained problem, we perform energy minimization with terms to maximize translucency—absent additional information, fewer rather than more layers should contribute to a pixel—and spatial coherence.

Constraints. For **perfect reproduction**, the color palette's convex hull must enclose all image colors. This may not be possible for **too-small color palettes**, given that the hull vertices must be **valid colors** that lie within the unit RGB cube. It is possible to use fewer vertices if they can be "imaginary" colors outside the RGB cube. Furthermore, we require **valid opacity** values that lie between 0 and 1. If we allow opacity values beyond 0 and 1, we could reproduce colors outside the convex hull of the color palette, but that is not a standard layer format. Finally, we assume **single-color layers**. Each layer models a coat of paint of a single color applied with varying opacity.

Our **contributions** are as follows:

—The geometric analysis of an image in RGB-space to determine a small color palette capable of reproducing the image (Section 3). Our algorithm is based on its simplified RGB-space convex hull.
—An optimization-based approach to compute per-layer, per-pixel opacity values (Section 4) given a user-provided *ordering* of the colors. The layers, when composited, reproduce the input image with minimal error. Our approach regularizes an underconstrained problem with terms that balance translucency and spatial coherence.

The result of these contributions is a technique that decomposes a single image into translucent layers. Our approach can be applied to any image, such as photographs and physical paintings, to extract a small palette of generating colors. Our decomposition enables the structured re-editing and recoloring of digital paintings and other images. Furthermore, our layers can be interpreted as a generalized barycentric coordinate representation of the image (Section 4.2). We compare our results to recoloring and generalized barycentric coordinate approaches (Section 5). We also consider various relaxations of our problem statement, such as imaginary colors, multiple colors per layer, and opacity values outside [0, 1].

2. RELATED WORK

Single-Image Decomposition. Richardt et al. [2014] investigated a similar problem with the goal of producing editable vector

coordinates for any point; we can also perform a tetrahedra analog of the two-triangle edge flip operation on any tetrahedron containing a point and one of its neighbors to obtain a different tetrahedralization of their space and therefore different barycentric coordinates for the point.
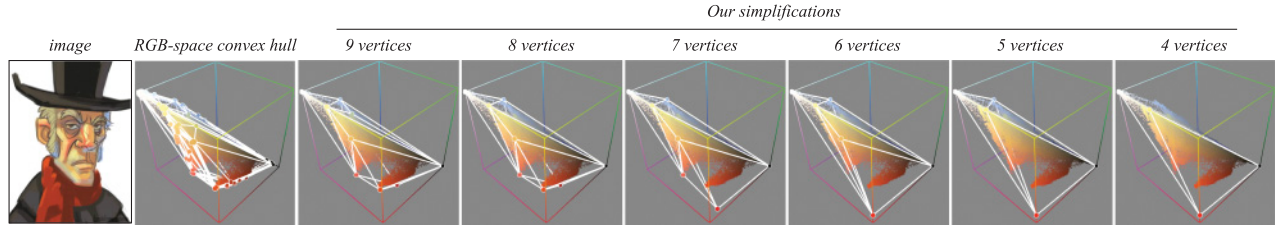
Our simplifications

| image | RGB-space convex hull | 9 vertices | 8 vertices | 7 vertices | 6 vertices | 5 vertices | 4 vertices |

Fig. 3.   Various simplification levels for the RGB-space convex hull of an image. Artwork © Dani Jones.

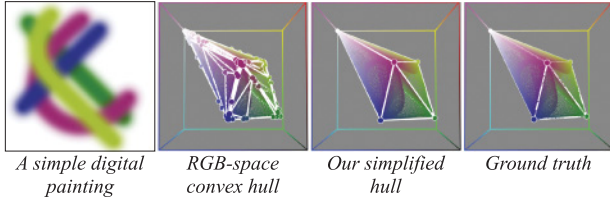*A simple digital painting* | *RGB-space convex hull* | *Our simplified hull* | *Ground truth*

Fig. 4.   (a) A simple digital painting's (b) convex hull in RGB-space is complex due to rounding. (c) The result of our simplification algorithm (d) matches ground truth, its original paint colors as an RGB-space polyhedron.

graphics. Our goal is to produce editable layered bitmaps. They proposed an approach in which the user selects an image region, and the region is then decomposed into a linear or radial gradient and the residual, background pixels. Our approach outputs bitmap image layers, which are a less constrained domain. Our approach also requires much less user input. For comparison, we decompose several images from their paper (Figure 12).

Xu et al. [2006] presented an algorithm for decomposing a single image of a Chinese painting into a collection of layered brush strokes. Their approach is tailored to a particular style of artwork. They recover painted colors by segmenting and fitting curves to brush strokes. They also consider the problem of recovering per-pixel opacity as we do. In their setting, however, they assume at most two overlapping strokes and minimally varying transparency. We consider a more general problem in which strokes have no known shape and more than two strokes may overlap at once. Fu et al. [2011] introduced a technique to determine a plausible animated stroke order from a monochrome line drawing. Their approach is based on cognitive principles and operates on vector graphics. We do not determine a stroke order; rather, we extract paint colors and per-pixel opacity and operate on raster digital paintings.

McCann and Pollard [2009, 2012] introduced two generalizations to layering, allowing (1) pixels to have independent layer orders and (2) layers to partially overlap each other. We solve for layer opacity coefficients in the traditional, globally, and discretely ordered model of layers.

Scale-space filtering [Witkin 1983] and related techniques [Subr et al. 2009; Aujol and Kang 2006; Farbman et al. 2008] decompose a single image into levels of varying smoothness. These decompositions separate the image according to levels of detail, such as high-frequency texture and underlying low-frequency base colors. These techniques are orthogonal to ours, as they are concerned with spatial frequency and we are concerned with color composition.

Intrinsic image decomposition [Grosse et al. 2009; Shen et al. 2008; Bousseau et al. 2009] attempts to separate a photographed object's illumination (shading) and reflectance (albedo). Lawrence et al. [2006] tackled the related physical problem of decomposing a

input image | *convex hull* | *simplified hull with invalid colors* | *projected hull* | *optimized hull*
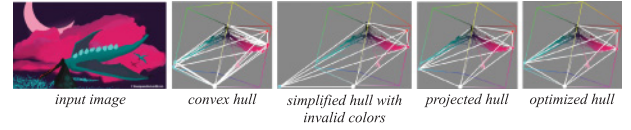
Fig. 5.   One of the only images in our dataset in which vertex position optimization led to an improvement in vertex positions. The impact was negligible on reconstruction error. Artwork © Karl Northfell.
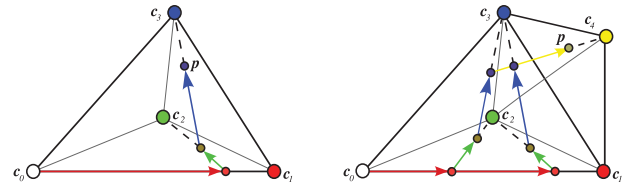

Fig. 6.   A pixel can be represented as a path from the background color c_0 toward each of the other colors c_1, c_2, c_3, ... in turn. The opacity values α_i determine the length of each arrow as a fraction of the entire line segment to c_i. When there are no more than four colors (left), there is only one possible path. With more than four colors (right), there are many possible paths.

Spatially-Varying Bidirectional Reflectance Distribution Function into a shade tree. These decompositions are suitable for photographs of illuminated objects, but not, for example, digital paintings.

The recoloring approach of Chang et al. [2015] extracts a color palette from a photograph by clustering. Gerstner et al. [2013] extracted sparse palettes from arbitrary images for the purpose of creating pixel art. Unlike approaches based on clustering the observed colors, our approach has the potential to find simpler and even "hidden" colors. Consider an image created from a blend of two colors with varying translucency, never opaque. In the final image, the original colors will never be present, though an entire spectrum of other colors will be (Figure 2).

Editing History. Tan et al. [2015] and Amati and Brostow [2010] described approaches for decomposing time-lapse videos of physical (and digital, for Tan et al. [2015]) paintings into layers. In our scenario, we have only the final painting, though we make the simplifying assumption that only Porter-Duff [1984] "over" blending operations were performed.

Hu et al. [2013] studied the problem of reverse-engineering the image editing operation that occurred between a pair of images. We are similarly motivated by "inverse image editing," though we solve an orthogonal problem in which only a single image is provided and the only allowable operation is painting.

A variety of approaches have been proposed to make use of image editing history (see Nancel and Cockburn [2014] for a recent
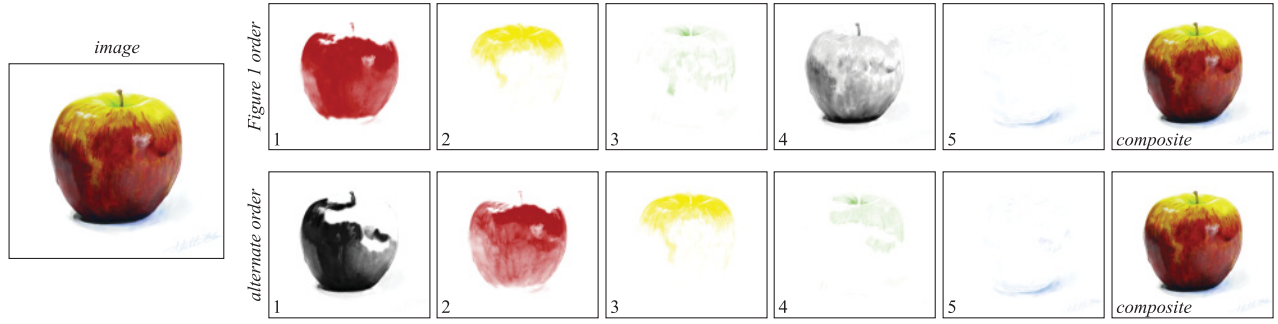
Fig. 7. Different layer orders result in different decompositions. Artwork © Adelle Chudleigh.
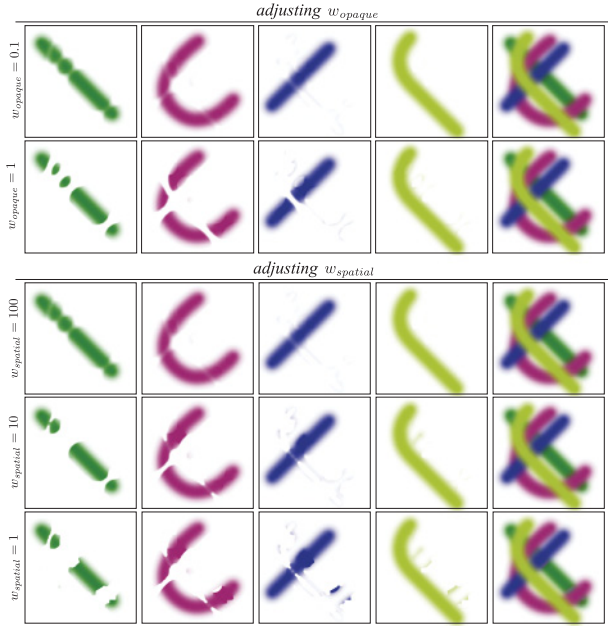

Fig. 8. The effect of changing the opacity optimization weights. For this example, $w_{opaque} = .1$, $w_{spatial} = 100$ produces a better result than the values used for all other examples ($w_{opaque} = 1$, $w_{spatial} = 100$).

survey). While we do not claim that our decomposition matches the true image editing history, our approach could be used to provide a plausible editing history. In particular, Wetpaint [Bonanni et al. 2009] proposed a tangible "scraping" interaction for visualizing the layers of a painting.

Matting and Reflections. Smith and Blinn [1996] studied the problem of separating a potentially translucent foreground object from known backgrounds in a photo or video ("blue screen matting"). Zongker et al. [1999] solved a general version of this problem, which allows for reflections and refractions. Levin et al. [2008a, 2008b] presented solutions to the natural image matting problem, which decomposes a photograph with a natural background into layers; Levin et al.'s solutions assume at most three layers per small image patch and find as binary as possible opacity values. We compare our output to Levin et al. [2008b] in Section 5. Layer extraction has been studied in the context of photographs of reflecting objects, such as windows [Szeliski et al. 2000; Farid and Adelson 1999;

Levin et al. 2004; Sarel and Irani 2004]. These approaches make physical assumptions about the scene in the photograph; they require a pair of photographs as input [Farid and Adelson 1999]. We consider digital images in general, in which physical assumptions are not valid and there are typically more than two layers.

3. IDENTIFYING PAINT COLORS

The first step in our pipeline identifies the colors used to paint the image. In a digital painting, many pixels will have been painted over multiple times with different paint colors. Because the paint compositing operation is a linear blend between two paint colors (Equation (1)), all pixels in the painting lie in the RGB-space convex hull formed by the original paint colors. Equivalently, any pixel color $\mathbf{p}$ can be expressed as the convex combination of the original paint colors $\mathbf{c}_i$:

$$\mathbf{p} = \sum w_i \mathbf{c}_i \qquad (3)$$

for some weights $w_i \in [0, 1]$ with $\sum w_i = 1$. This convex combination property is true for Porter-Duff "over" compositing, but not true for nonlinear compositing such as the Kubelka-Munk model of pigment mixing or layering [Budsberg 2007]. Figures 1, 4, 17, 19, 21, and 12 display pairs of images and their pixels in RGB-space. Note that the w_is in Equation (3) are not opacity values. Rather, they are generalized barycentric coordinates and do not depend on the layer order. The relationship between layer opacity and generalized barycentric coordinates will be discussed in Section 4.

To identify the colors used to paint the digital image (or a set of generating colors for any image), we first compute the RGB-space convex hull of all observed pixel colors. The convex hull is a tight wrapping of the colors. In practice, it will be overly complex (too many vertices). Too many vertices would result in an unwieldy number layers; we wish to have a manageable (user-determined) number of layers with clearly differentiated colors (Figure 3). The large number of vertices may be due to quantization artifacts or to the generating colors being applied semitransparently. Semitransparent paint does not produce any pixels with the paint color itself. This manifests as "cut corners" or extra faces in the convex hull (Figure 2).

3.1 Simplifying the Convex Hull

The next stage in our pipeline simplifies the convex hull to a desired, user-provided palette size. We considered a variety of simplification approaches. There is a well-known convex polytope approximation due to Dudley [1974] and Har-Peled [1999]. This approximation strictly adds volume, so all colors will still be contained in the

Table I. Performance

image	width × height	opacity optimization			
		runtime (seconds)	RMSE	median error	max error
apple	500×453	330.3	1.8	0.0	32.3
bird	640×360	500.4	3.7	2.2	60.1
rowboat	589×393	981.4	3.3	2.4	19.2
buildings	589×393	317.9	2.3	1.4	32.2
cup	400×400	40.9	3.0	0.0	34.3
fruit	650×414	212.1	1.9	0.0	22.6
girls	589×393	152.7	2.4	1.4	29.1
hoover	500×500	47.1	3.9	1.0	39.2
light	504×538	101.6	2.4	1.0	25.5
robot	450×600	519.8	3.5	2.8	14.5
scrooge	410×542	97.6	2.6	1.4	15.7
Figure 4	500×500	434.3	0.7	0.0	3.3
trees	606×404	80.2	5.9	4.2	35.0
turtle	525×250	19.5	2.8	1.0	45.4
boat	480×600	520.0	4.2	2.2	40.2
castle	747×344	280.0	3.7	2.2	45.6
turquoise	480×585	498.7	2.0	1.4	17.8
moth	650×390	675.1	3.1	1.7	25.7

Running time and reconstruction error. Difference images appear almost universally black and can be found in the supplemental materials.

middle layer *top layer*

top layer (over black)

top layer
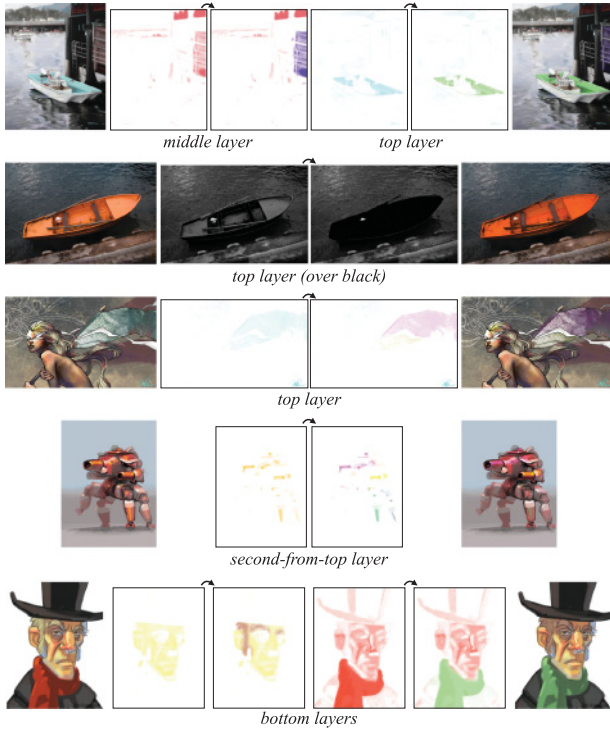
second-from-top layer

bottom layers

Fig. 9. Our layer decomposition enables local image recoloring. The input images' layer decompositions can be seen in Figures 17, 19, and 21. Original artworks © Michelle Lee, Bychkovsky et al. [2011], Michelle Lee, Adam Saltsman, Dani Jones.

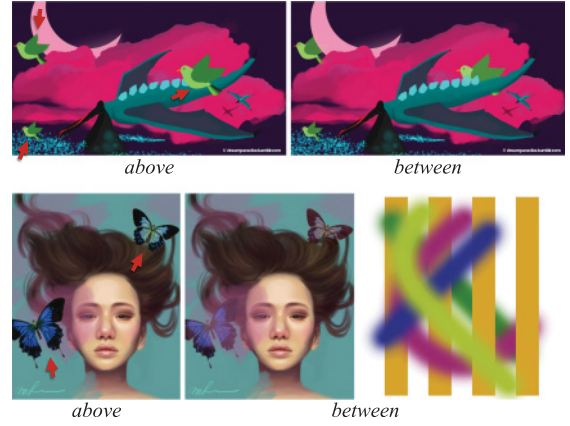*above* *between*

above *between*

Fig. 10. Inserting graphics as new layers between our decomposed layers (Figures 17, 19, and 8) produces a more natural result than pasting graphics above. Original artworks © Karl Northfell, Michelle Lee.

Table II. Generalized Barycentric Coordinate Weight Sparsity

	image	MVC	LBC	Our optimization	ASAP
valid colors	apple	0.54	0.59	0.62	0.66
	rowboat	0.11	0.34	0.44	0.57
	girls	0.10	0.22	0.36	0.40
	robot	0.01	0.14	0.47	0.50
	scrooge	0.30	0.38	0.51	0.53
	Figure 4	0.33	0.38	0.41	0.47
	boat	0.08	0.24	0.49	0.60
invalid colors	apple	0.10	0.27	0.41	0.46
	rowboat	0.03	0.20	0.41	0.52
	girls	0.00	0.14	0.34	0.37
	robot	0.00	0.18	0.39	0.47
	scrooge	0.10	0.18	0.29	0.31
	Figure 4	0.01	0.16	0.18	0.26
	boat	0.01	0.23	0.45	0.54

Our weights are sparser than Mean-Value Coordinates [Floater et al. 2005; Ju et al. 2005] and Local Barycentric Coordinates [Zhang et al. 2014], and almost as sparse as the optimal as-sparse-as-possible weights. Sparsity is computed as the fraction of near-zero values ($\epsilon = \frac{1/512}{\#\text{vertices}}$). The lower half of the table (shaded yellow) shows sparsity when imaginary colors are used, which may be further from the pixel colors.

approximate shape. The approximation takes the form of a constructive proof to find a simpler polytope with $O(\frac{1}{\mu})$ vertices that adds less than a factor of μ volume. The constructive proof is, for our purposes, underdetermined. Vertices are selected based on any regular sampling of the shape's bounding sphere. This approach is not rotation invariant and ends up selecting a subset of the original polytope's faces. The selected faces are treated as half-planes, which define a new polytope. Instead, we use a tighter, well-determined approximation.

The progressive hull is a mesh simplification technique introduced by Sander et al. [2000]. The approach is based on a sequence of edge contractions, in which an edge is contracted to a vertex. The vertex is placed according to a constrained optimization problem. The constraints are that the vertex must be placed outside (the plane of) every face incident to the edge. Equivalently, every tetrahedron formed by connecting the new vertex to a face incident to the edge must add volume to the shape. The progressive hull objective

function minimizes the total added volume. The next edge to contract is the one whose contraction adds the least volume. (Edges with no solution satisfying the constraints are skipped.) This approximation strictly adds volume, so all colors are guaranteed to be contained in every step of the progressive hull. Moreover, the constraints and objective function are linear, since the (oriented) volume of a tetrahedron with three fixed vertices is linear:

$$\frac{A}{3}\mathbf{n} \cdot (\mathbf{v} - \mathbf{v}_0), \qquad (4)$$

where $\mathbf{v}$ is the free vertex, $\mathbf{v}_0$ is any one of the fixed vertices, and $\mathbf{n}$ and A are the outward unit normal and area of the triangle formed by the three fixed vertices. As a result, the constrained optimization for each edge contraction is a linear programming problem, which can be solved efficiently.

There are infinitely many solutions when all faces incident to an edge are exactly coplanar. If all faces are nearly coplanar, the solution becomes unstable. We consideredalternative objective functions (subject to the progressive hull constraints): the distance to each face's plane (the objective function used in the classic mesh simplification approach of Garland and Heckbert [1997] and equivalent to not multiplying by $\frac{A}{3}$ in Equation (4)) and the distance to the edge undergoing contraction (a quadratic programming problem). We experimented with these alternative objective functions both for determining the contracting edge's vertex placement and as the metric for determining the next edge to contract. All combinations usually produced similar results. Our results were computed using added volume to determine the next edge to contract and the total distance to incident face's planes in the constrained optimization, as this combination produced stabler results than the total added volume constrained optimization with the same running time.

We also experimented with a RANSAC plane-fitting approach, in which we greedily fit planes to the convex hull. The simplified polyhedron is then taken as the intersection of the half-spaces defined by each plane. However, this approach is difficult to control. The degree of simplification is limited as the planes cannot deviate from the convex hull, and the planes' intersections produce multiple nearby vertices. As a result, the user cannot directly specify a desired number of vertices. Moreover, there are additional parameters to tune, such as the RANSAC inlier distance and the threshold for collapsing nearby vertices in the planes' intersections.

3.2 Imaginary Colors

While the vertices of the convex hull are always located within the RGB cube, the vertices of a simplified hull may not be. Vertices outside the RGB cube are "imaginary" colors and cannot be used as layer colors. They can, however, still be used for recoloring based on generalized barycentric coordinates (Section 4.2).

For our layer decomposition, we require valid colors. Constraining the linear programming optimization to only consider vertices within the RGB cube frequently overconstrains the problem, resulting in no solution. Intersecting the simplified hull with the cube (as solid shapes) increases the number of vertices, sometimes in a manner that is difficult to control, such as when a vertex protrudes only a small amount.

There is a tension between a small number of vertices and a simplified hull that still encloses all image colors. Our solution is to allow reconstruction error in order to achieve a user's desired color palette size. We experimented with optimization to adjust vertex positions—minimizing the average distance of all pixels to the hull and the distance hull vertices move—but found the added
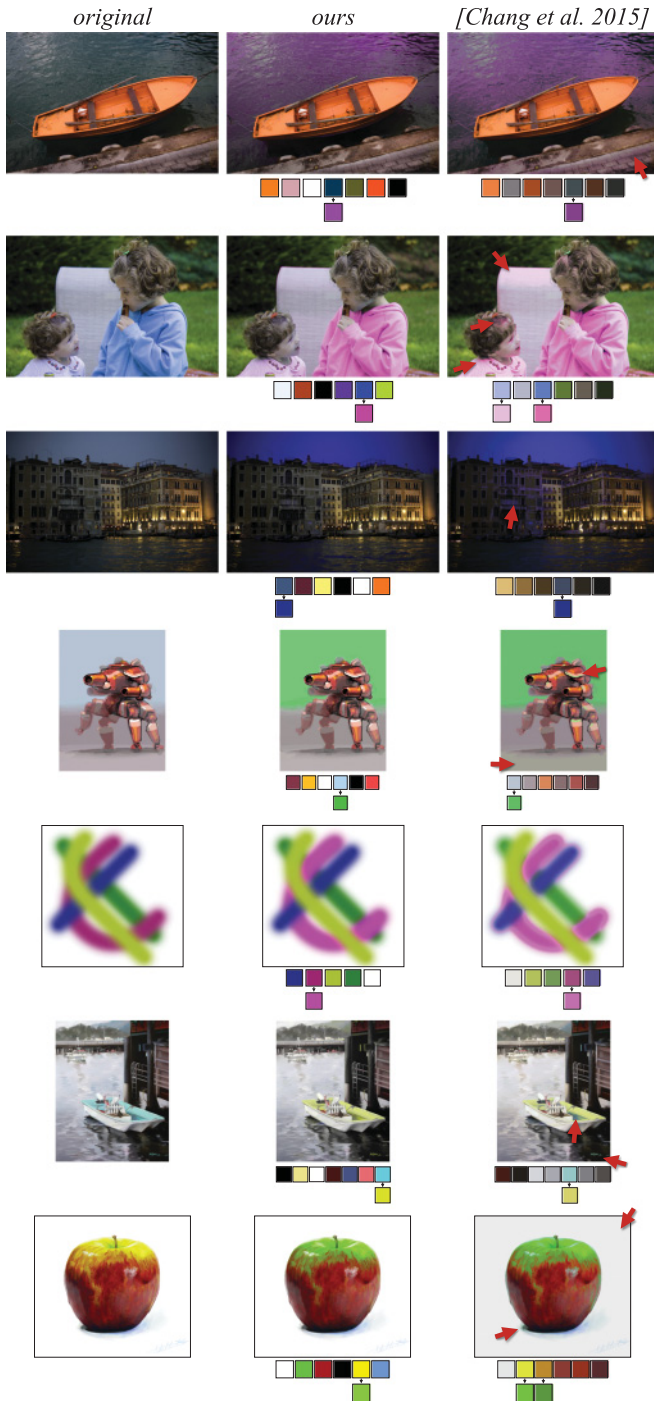
Fig. 11. Recoloring images by changing palette colors with our approach and with the approach of Chang et al. [2015]. Note that as the approaches find different palettes, colors were modified to achieve a similar recolored result. Our results contain fewer unrelated changes or artifacts (red arrows). The images' layer decompositions computed by our method can be seen in Figures 1, 4, 17, 19, and 21. Top three photographs from Bychkovsky et al. [2011]. Bottom four original artworks © Adam Saltsman, Yotam Gingold, Michelle Lee, Adelle Chudleigh.

| image | RGB-space simplified hull | layer 1 | layer 2 | layer 3 | layer 4 | composite |

Fig. 12. Our decomposition results on examples from Richardt et al. [2014] (*cup*, *hoover*, and *light*). Results were computed with an opaque (RGB) background. Artworks © George Dolgikh, Spencer Nugent, Roman Sotola.
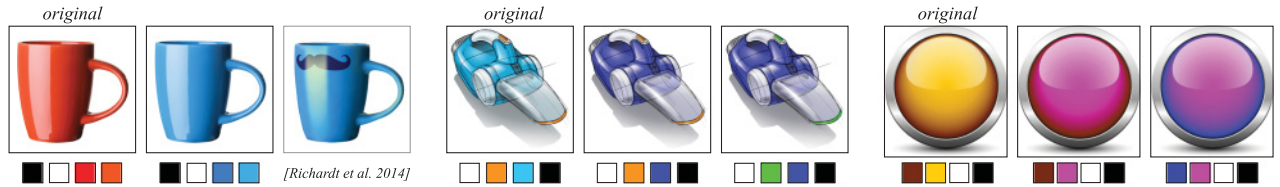

Fig. 13. Global recoloring results obtaining by adjusting layers colors for the examples in Figure 12. Richardt et al.'s result [2014] requires manual segmentation. Original artworks © George Dolgikh, Spencer Nugent, Roman Sotola.
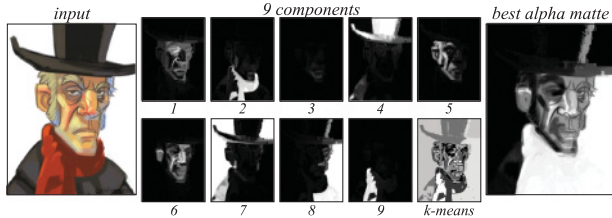

Fig. 14. The output of Levin et al. [2008b] on our *scrooge* example. The natural image assumptions are not well suited for digital paintings. Artwork © Dani Jones.

complexity and running time to have little overall impact on the reconstruction error (Figure 5). We also experimented with optimizing vertex placement during our subsequent opacity computation (Section 4) but found it difficult to control. Ultimately, some amount of error is unavoidable due to the small number of hull vertices constrained to lie within the RGB cube. Simply projecting simplified vertices that lie outside the RGB cube to the closest point on the cube resulted in a simple, predictable algorithm and reconstructions with low error (Table I). After projecting hull vertices, some image colors no longer lie within the hull. We project such outside image colors to the closest point on the hull's surface. This is a source of reconstruction error. In Figure 15, we show an

interface in which the imaginary colors can directly be used for image recoloring.

The next stage of our algorithm determines the per-pixel opacity of each layer.

4. DETERMINING LAYER OPACITY

The final stage of our algorithm computes per-pixel opacity values for each layer. This stage takes as input ordered RGB layer colors $\{\mathbf{c}_i\}$ resulting from Section 3. Then, at each pixel, the observed color $\mathbf{p}$ can be expressed as the recursive application of "over" compositing (Equation (1)), factored into the following convenient expression:

$$\mathbf{p} = \mathbf{c}_n + \sum_{i=1}^{n} \left[(\mathbf{c}_{i-1} - \mathbf{c}_i) \prod_{j=i}^{n} (1 - \alpha_j) \right], \qquad (5)$$

where α_i is the opacity of $\mathbf{c}_i$, and the background color $\mathbf{c}_0$ is opaque. Since colors $\mathbf{p}$ and $\mathbf{c}_i$ are three-dimensional (RGB), this is a system of three polynomial equations with number of layers unknown. For RGBA input images or images with an unknown background color, premultiplied RGBA colors should be used. Equation (5) becomes a system of four polynomial equations. The background layer $\mathbf{c}_0$ becomes transparent—the zero vector—while the remaining global layer colors $\mathbf{c}_i$ are opaque.

| | | | layers | | | composite | edit ▣→▣ |

Fig. 15. Recoloring an example from Chang et al. [2015] by manipulating vertices of the polyhedron and reconstructing colors with generalized barycentric coordinates. We compare our layers as weights (Equation (7)) to Mean-Value Coordinates [Floater et al. 2005; Ju et al. 2005], Local Barycentric Coordinates [Zhang et al. 2014], and our as-sparse-as-possible coordinates. ASAP produces a very similar result to our optimization, while MVC and LBC produce an undesirable change in the chair and the girls' hair. The image's layer decomposition can be seen in Figure 21. Photograph from Bychkovsky et al. [2011].
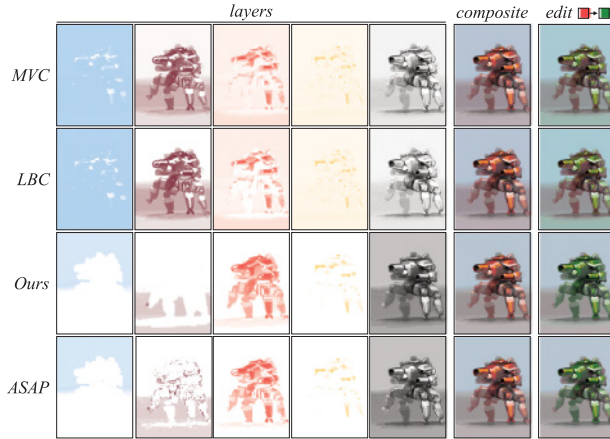

Fig. 16. Converting generalized barycentric coordinates into layers shows that our optimization's layers and the As-Sparse-As-Possible weights are much sparser than Mean-Value Coordinates [Floater et al. 2005; Ju et al. 2005] and Local Barycentric Coordinates [Zhang et al. 2014]. On close examination, the non-smoothness of ASAP weights is apparent. Our supplemental materials include additional comparisons between ASAP weights as layers and our opacity optimization. Artwork © Adam Saltsman.

There will always be at least one solution to Equation (5), because **p** lies within the RGB-space convex hull of the c_i. When the number of layers is less than or equal to four (not counting the translucent background in case of RGBA), there is, in general, a unique solution. It can be obtained geometrically in RGB-space by projecting **p** along the line from the topmost layer color c_n onto the simplex formed by $c_0 \ldots c_{n-1}$, and so on recursively (Figure 6). However, if **p** is identical to one of the c_is (other than the bottom layer) or the number of layers is greater than four, there are infinitely many solutions (Figure 6). For numerical reasons, it is problematic when **p** is *nearly* identical to a layer color—a situation that arises often.

4.1 Layer Order

"Over" color compositing, while linear, is not commutative. For n layers, there are $n!$ orderings. Because of the large possibility space and the unknown semantics of the colors, we do not automate the determination of the layer order. In our experiments, we computed opacity values for all $n!$ layer orders with the algorithm described in this section and attempted to find automatic sorting criteria. We experimented with the total opacity, gradient of opacity, and Laplacian of opacity, but none matched human preference. As a result, we require the user to choose the layer order for the extracted colors.

An alternative layer order for the example in Figure 1 is shown in Figure 7.

4.2 Generalized Barycentric Coordinates

Generalized Barycentric Coordinates express any point **p** inside a polyhedron as a weighted average of the polyhedron's vertices c_i (Equation (3)). For simple recoloring applications, the RGB-space vertices can be modified and the pixel colors then recomputed. The sparsest possible weights have at most four nonzero w_is. In other words, a pixel can always be expressed as the weighted average of four colors. This corresponds to the well-defined, nongeneralized barycentric coordinates of any tetrahedron enclosing **p** whose vertices are a subset of the c_i. These as sparse as possible (ASAP) weights can be made continuous as a function of **p** by using a conforming (nonoverlapping) tetrahedralization of the polyhedron. Since the tetrahedra must be composed of vertices of the simplified convex hull, this corresponds to choosing one color and connecting it to every face. If the user has identified an opaque background color as the bottommost layer, then a natural choice is to choose it; the ASAP weights therefore define all pixels as the mixture of at most three nonbackground colors.

Generalized barycentric coordinates w_i for a pixel can be converted into layer opacities α_i as follows:

$$\alpha_i = \begin{cases} 1 - \frac{\sum_{j=0}^{i-1} w_j}{\sum_{j=0}^{i} w_j} & \text{if } \sum_{j=0}^{i} w_j \neq 0 \\ 0 & \text{otherwise.} \end{cases} \quad (6)$$

Due to the division by zero in the general case, the conversion from generalized barycentric coordinates to opacity values is ambiguous and relies on an arbitrary and potentially nonsmooth choice. This corresponds to the ambiguity that arises when an opaque layer occludes everything underneath. Due to this ambiguity, we propose a sparse and smooth optimization-based approach to computing layer opacities.

Note that layer opacity values can be converted to generalized barycentric coordinates in a well-defined manner:

$$w_i = \begin{cases} \prod_{j=i+1}^{n}(1-\alpha_j) & \text{if } i = 0 \\ \left(\prod_{j=i+1}^{n}(1-\alpha_j)\right) - \left(\prod_{j=i}^{n}(1-\alpha_j)\right) & \text{if } 0 < i < n \\ 1 - \prod_{j=i}^{n}(1-\alpha_j) = \alpha_i & \text{if } i = n. \end{cases} \quad (7)$$

We compare our optimization solutions to ASAP weights and two other well-known generalized barycentric coordinates, Mean-Value Coordinates [Floater et al. 2005; Ju et al. 2005] and Local Barycentric Coordinates [Zhang et al. 2014], in Section 5.

4.3 Optimization

To choose among the infinitely many solutions to Equation (5), we introduce two regularization terms and solve an optimization
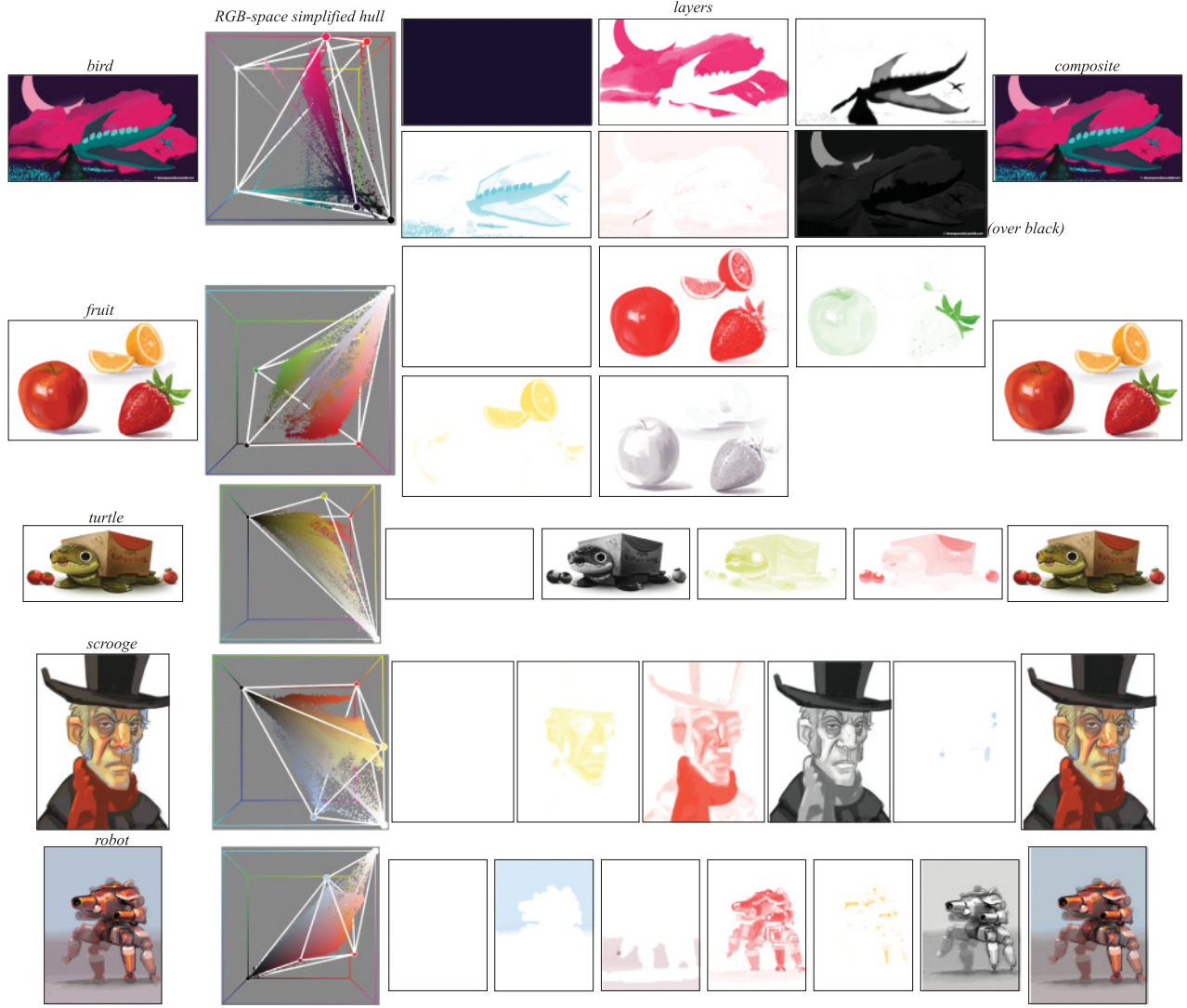
Fig. 17. Our decomposition results on digital paintings. Results were computed with an opaque (RGB) background. Layers appear in reading order (left to right, top to bottom). Artworks © Karl Northfell, DeviantArt user Ranivius, Piper Thibodeau, Dani Jones, Adam Saltsman.

problem. Our first regularization term penalizes opacity; absent additional information, a completely occluded layer should be transparent:

$$E_{opaque} = \frac{1}{n} \sum_{i=1}^{n} -(1 - \alpha_i)^2. \tag{8}$$

Minimizing $-(1 - \alpha_i)^2$ rather than the more typical α_i^2 results in a far sparser solution. Intuitively, naively squaring α_i produces an objective function that would "prefer" to decrease α_i from 1 and increase some other α_j away from 0. Our E_{opaque} prefers the opposite, resulting in a sparse solution. This unusual formulation is possible because the α_is are bounded in the interval [0, 1].

Our second regularization term penalizes solutions that are not spatially smooth. We use the Dirichlet energy, which penalizes the difference in α_i between each pixel and its spatial neighbors:

$$E_{spatial} = \frac{1}{n} \sum_{i=1}^{n} (\nabla \alpha_i)^2, \tag{9}$$

where $\nabla \alpha_i$ is the spatial gradient of opacity in layer i.

We minimize these two terms subject to the polynomial constraints (Equation (5)) and $\alpha_i \in [0, 1]$. We implement the polynomial constraints as a least-squares penalty term per-pixel $E_{polynomial}$:

$$E_{polynomial} = \frac{1}{K} \left\| \mathbf{c}_n - \mathbf{p} + \sum_{i=1}^{n} \left[(\mathbf{c}_{i-1} - \mathbf{c}_i) \prod_{j=i}^{n} (1 - \alpha_j) \right] \right\|^2, \tag{10}$$
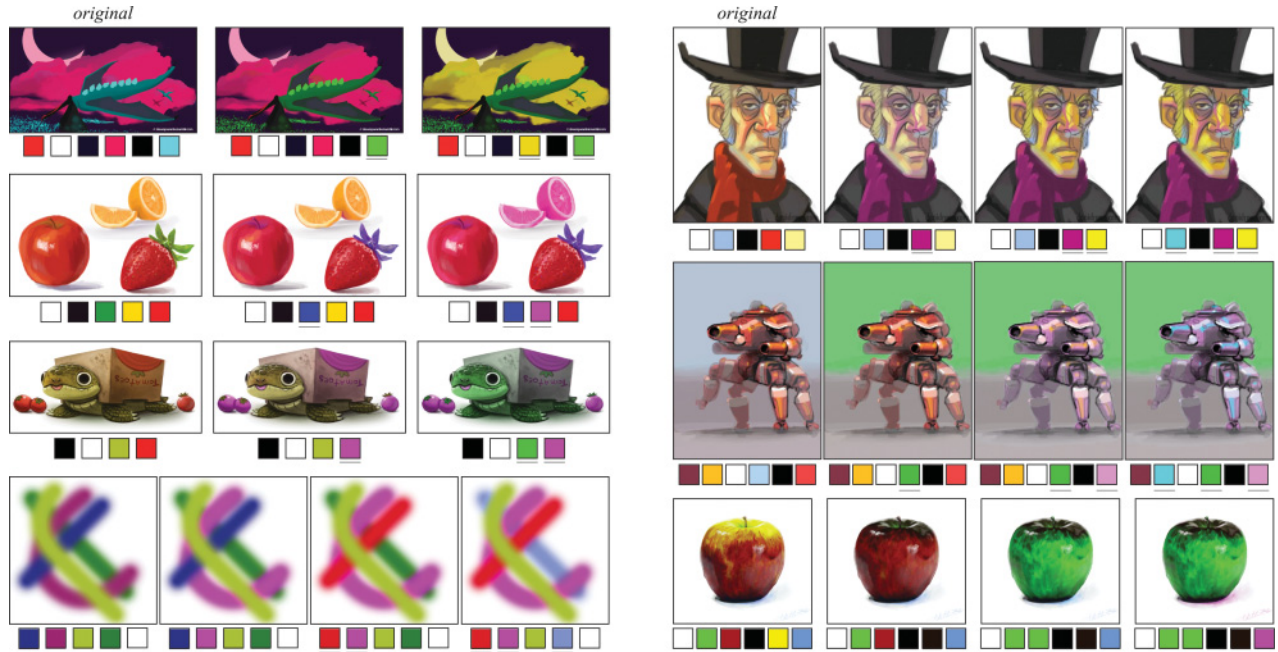
Fig. 18. Global recoloring results obtaining by adjusting layer colors for the examples in Figures 1, 4, and 17. Original artworks © Karl Northfell, DeviantArt user Ranivius, Piper Thibodeau, Yotam Gingold, Dani Jones, Adam Saltsman, Adelle Chudleigh (top to bottom, left to right).

where $K = 3$ or 4 depending on the number of channels (RGB or RGBA). The combined energy expression that we minimize is

$$w_{polynomial} E_{polynomial} + w_{opaque} E_{opaque} + w_{spatial} E_{spatial}.$$

We used $w_{polynomial} = 375$, $w_{opaque} = 1$, and $w_{spatial} = 100$ for all of our examples. Figure 8 shows an evaluation of the effect of changing weights for our one example in which the defaults do not produce the best output.

5. RESULTS

Implementation. We use QHull [Barber et al. 1996] for convex hull computation, GLPK [GNU Project 2015] for solving the progressive hull linear programs, and L-BFGS-B [Zhu et al. 1997] for opacity optimization. Our algorithms were written in Python and vectorized using NumPy/SciPy. Our implementation is not multithreaded.

To improve the convergence speed of the numerical optimization, we minimize our energy on recursively downsampled images and use the upsampled solution as the initial guess for the larger images (and, eventually, the original image). We down/upsampled by factors of two. We used $\alpha_i = 0.5$ as the initial guess for the smallest image. We experimented with using ASAP weights as the initial guess, without downsampling. (Downsampling is incompatible with a detailed guess.) With the ASAP initial guess, optimization took longer on average while producing similar or worse (less smooth) results.

Performance. All experiments were performed on a single core of a 2.9GHz Intel Core i5-5257U processor. Paint color identification (Section 3) takes a few seconds to compute the simplified convex hull; the bottleneck is the user choosing the desired amount of simplification. Computing layer opacity (Section 4) entails solving a nonlinear optimization procedure. As we implemented our optimization in a multiresolution manner, the user was able to quickly

see a preview of the result (seconds for a 100-pixel-wide image). This is important for experimenting with different layer orders and energy weights. Larger images are computed progressively as the multiresolution optimization converges on smaller images; the final optimization can take anywhere from 1 minute to 15 minutes to converge (Table I). Once decomposed, applications such as recoloring and object insertion are computed extremely efficiently as compositing operations.

Layer Decompositions. Figures 1, 7, 12, 17, 19, and 21 show the decomposition of a variety of digital paintings. The decomposed layers reproduce the input image without visually perceptible differences and with low root-mean-squared error (Table I). Absolute difference images, virtually all of which appear uniformly black, can be found in the supplemental materials. This is because the approximate convex hulls cover almost every pixel in RGB-space (Section 3), and the polynomial constraints in the energy minimization ensure that satisfying opacity values are chosen (Section 4). The decomposed layer representations facilitate edits like spatially isolated recoloring (Figure 9) and inserting objects between layers (Figure 10), in addition to global recoloring by changing palette colors (Figures 13, 18, 20, and 22).

Comparisons. We have compared the output of our results to several alternative layer decomposition and recoloring approaches. Figure 11 compares global recolorings using our layer decomposition and the approach of Chang et al. [2015]. Our results contain fewer unrelated changes or artifacts. Note that the approaches find different palettes. These recolorings were created by modifying each approach's palette to achieve a similar recoloring result. Only our approach detects for example, the blue and green colors in the *apple* or the blue color in *scrooge*. (Figure 18 shows our *scrooge* palette, while Chang et al. [2015]'s palette is ▢▢▢▢▢.) As Chang et al. [2015] is based on clustering image colors, their palette colors
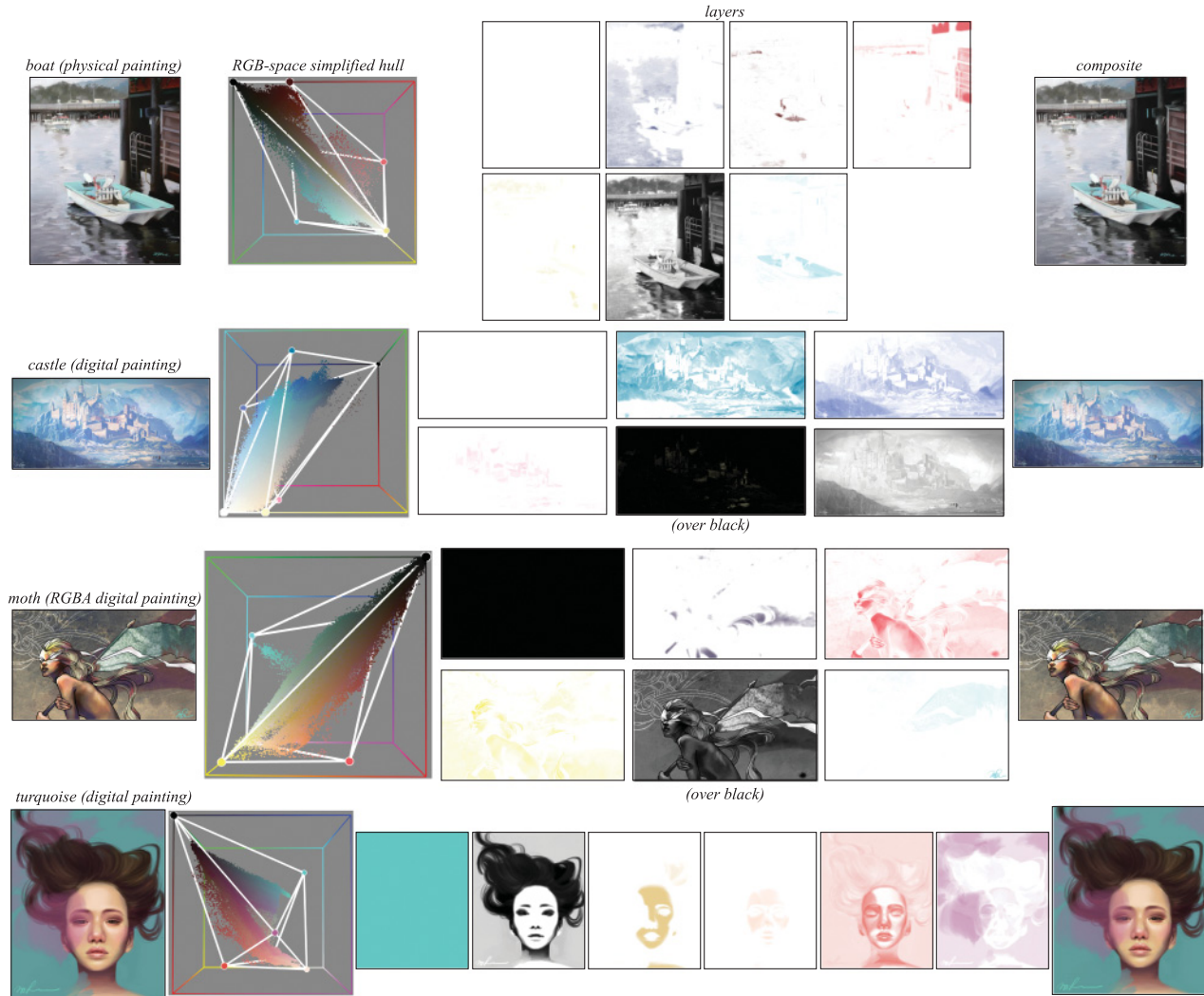
Fig. 19. Our decomposition results on digital and physical paintings. Results were computed with an opaque (RGB) or translucent (RGBA) background where noted. Artworks © Michelle Lee.

all lie within the interior of the pixel colors in RGB-space. Because these important colors are infrequent, they are missed by clustering-based methods. See inset on the right for bird colors detected by Chang et al. [2015] in RGB-space. Figure 21 shows the results of our layer decomposition algorithm on examples from Chang et al. [2015], and Figure 22 shows additional recoloring results. The computational complexity of our recolorings is extremely low; the recolored image is a per-pixel weighted average of the color palette. See the supplemental materials for an interactive recoloring GUI.

Figure 12 shows the results of our layer-based decomposition on examples from the vector graphics decomposition algorithm of Richardt et al. [2014]. Our algorithms produce substantially different output. In Richardt et al. [2014], users manually segment
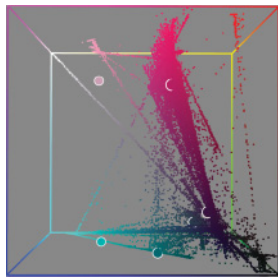
portions of each image to be decomposed into a gradient layer. Recoloring results for these examples are shown in Figure 13. The RGB-space geometric structure of the circular light's pixels showcases the strength of our algorithm, which achieves virtually the same colors as ground truth.

Figure 14 shows the results of the soft matting algorithm of Levin et al. [2008b] on one of our examples. This spectral matting approach makes natural image assumptions and is not well suited for digital paintings.

Generalized Barycentric Coordinates. Via Equation (7), we are able to compare our results to generalized barycentric coordinates and edit images even with imaginary colors. We compared to Mean-Value Coordinates (MVCs) [Floater et al. 2005; Ju et al. 2005], which are fast and closed form, and Local Barycentric Coordinates (LBCs) [Zhang et al. 2014], which require solving an optimization and aim to find sparse weights. We also compare to our ASAP weights, which are not smooth spatially in image-space and only C^0 smooth in RGB-space. Notably, our opacity optimization

Fig. 20. Global recoloring results obtained by adjusting layer colors for the examples in Figure 19. Original artworks © Michelle Lee.
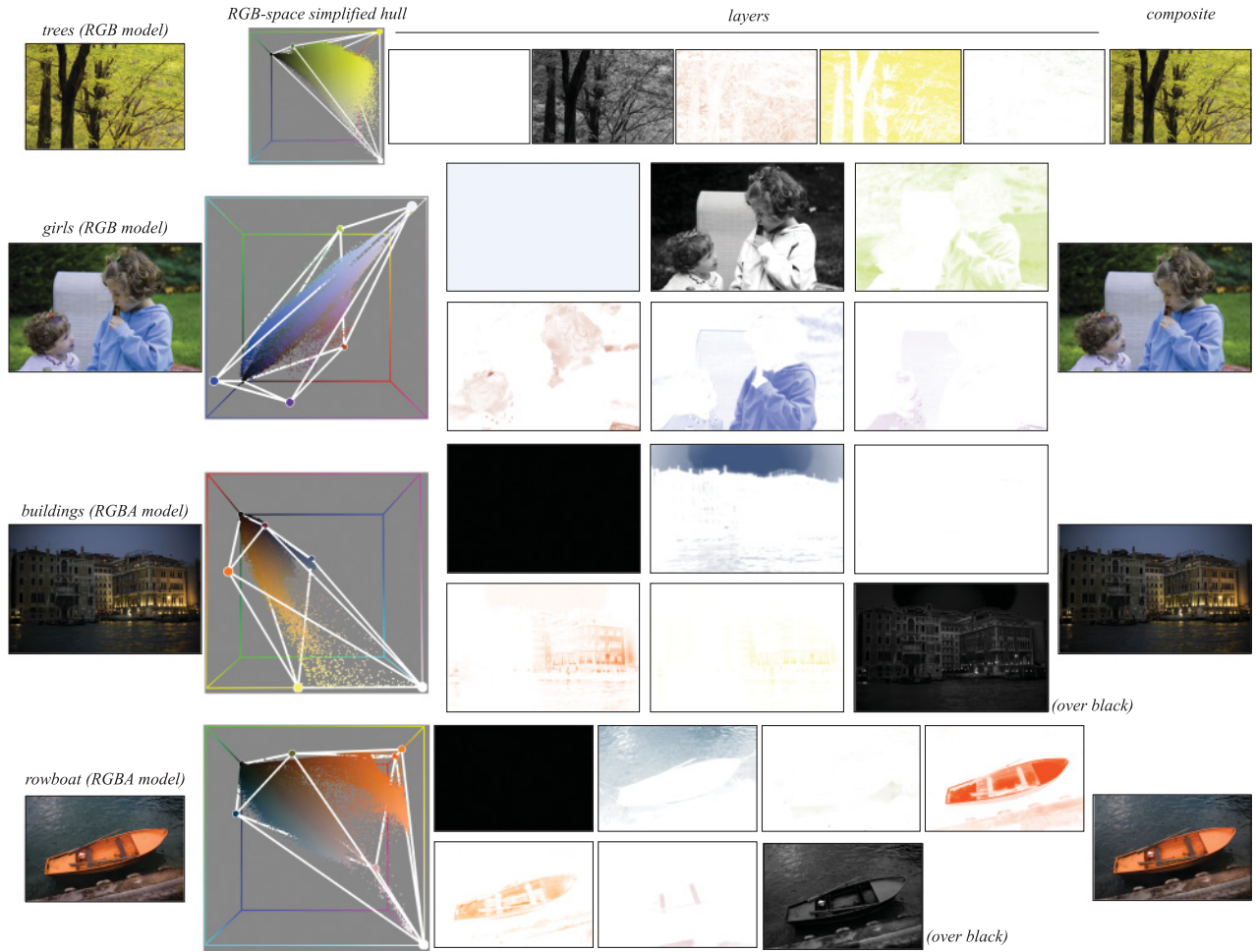

Fig. 21. Our decomposition results on examples from Chang et al. [2015]. Results were computed with an opaque (RGB) or translucent (RGBA) background where noted. Photographs from Bychkovsky et al. [2011].

produces sparser weights than either Mean-Value Coordinates or Local Barycentric Coordinates (Table II). We believe that our sparsity improvement is due to the small yet nonzero error introduced by our optimization. Figure 16 compares layers converted from

weights. Additional examples can be found in the supplemental materials. Our result is quite similar to ASAP, which can be computed in seconds, but much smoother. Figure 15 compares recoloring results with these techniques using the polyhedron vertices

Fig. 22.   Global recoloring results obtained by adjusting layer colors for the examples in Figure 21. Photographs from Bychkovsky et al. [2011].

as handles. Our supplemental materials contain a recoloring GUI based on manipulating the RGB-space vertices.

6.   CONCLUSION

The RGB-space of an image contains a "hidden" geometric structure. Namely, the convex hull of this structure can identify a small set of generating colors for any image. Given a set of colors and an order, our constrained optimization decomposes the image into a useful set of translucent layers. Our layers can be converted to a generalized barycentric coordinate representation of the input image, yet are sparser.

Limitations. Our technique has several notable limitations. First, selecting per-pixel layer opacity values is, in general, an underconstrained problem. Our optimization employs two regularization terms to bias the result toward translucent and spatially coherent solutions. However, this still may not match user expectations. Second, we expect a global order for layers. We use layers to represent the application of a coat of paint. However, in the true editing history, a single color may have been applied multiple times in an interleaved order with the other colors. Third, layer colors that lie within the convex hull cannot be detected by our technique. We also do not allow colors to change during optimization; we experimented with an energy term allowing layer colors to change but found it difficult to control. A related problem is images of, for example, rainbows; when the convex hull encompasses all or much of RGB-space, layer colors become uninformative (e.g., pure red, green, blue, cyan, magenta, yellow, and black). Fourth, we require user input to choose the degree of simplification for the convex hull and to choose the layer order. Fifth, outlier colors greatly influence the shape of the convex hull used for color palette selection. Outlier colors could be identified in a preprocessing step and ignored for palette selection (Section 3). Our opacity optimization approach will still choose values that minimize the RGB-space distance to the outlier. Sixth, if the image colors are all coplanar or collinear, color palette selection (Section 3) should use a 2D or 1D convex hull and simplification algorithm. Our implementation does not test for such a color subspace. Finally, nonlinear color-space transformations, such as gamma correction, distort the polyhedron. We ignore gamma information stored in input images.

Future Work. In the future, we plan to study decompositions with nonlinear color compositing operations, such as the Kubelka-Munk mixing and layering equations [Kubelka and Munk 1931; Kubelka 1948; Baxter et al. 2004; Lu et al. 2014; Tan et al. 2015]. This

would allow us to decompose scans of physical paintings into physically meaningful parameters. We also plan to evaluate our color palettes with the metric of O'Donovan et al. [2011] and compare to a model of human-extracted color palettes [Lin and Hanrahan 2013]. The metric could be used to automate recoloring by modifying our palettes to become more perceptually satisfying. Finally, we plan to apply our per-pixel layer opacity values toward segmentation; layer translucency is a higher-dimensional and potentially more meaningful feature than composited RGB color.

REFERENCES

Cristina Amati and Gabriel J. Brostow. 2010. Modeling 2.5D Plants from Ink Paintings. In *Sketch-Based Interfaces and Modeling (SBIM'10)*. 41–48.

Jean-Fran Aujol and Sung Ha Kang. 2006. Color image decomposition and restoration. *Journal of Visual Communication and Image Representation* 17, 4 (2006), 916–928.

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

William V. Baxter, Jeremy Wendt, and Ming C. Lin. 2004. IMPaSTo: A realistic, interactive model for paint. In *Non-Photorealistic Animation and Rendering (NPAR'04)*. 45–56.

Leonardo Bonanni, Xiao Xiao, Matthew Hockenberry, Praveen Subramani, Hiroshi Ishii, Maurizio Seracini, and Jurgen Schulze. 2009. Wetpaint: Scraping Through Multi-layered Images. In *Proceedings of ACM SIGCHI*. 571–574.

Adrien Bousseau, Sylvain Paris, and Frédo Durand. 2009. User-assisted Intrinsic Images. *ACM Transactions on Graphics* 28, 5, Article 130 (Dec. 2009), 130:1–130:10.

Jeffrey B. Budsberg. 2007. *Pigmented Colorants: Dependency on Media and Time*. Master's thesis. Cornell Univrsity, Ithaca, New York, NY.

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 Transactions on Graphics* 34, 4 (Aug. 2015), 139:1–139:11.

Richard M Dudley. 1974. Metric entropy of some classes of sets with differentiable boundaries. *Journal of Approximation Theory* 10, 3 (1974), 227–236.

Zeev Farbman, Raanan Fattal, Dani Lischinski, and Richard Szeliski. 2008. Edge-preserving decompositions for multi-scale tone and detail manipulation. *ACM Transactions on Graphics* 27, 3 (2008), 67:1–67:10.

Hany Farid and Edward H Adelson. 1999. Separating reflections from images by use of independent component analysis. *Journal of the Optical Society of America A* 16, 9 (1999), 2136–2145.

Michael S. Floater, Géza Kós, and Martin Reimers. 2005. Mean value coordinates in 3D. *Computer Aided Geometric Design* 22, 7 (2005), 623–631. DOI:http://dx.doi.org/10.1016/j.cagd.2005.06.004

Hongbo Fu, Shizhe Zhou, Ligang Liu, and Niloy J. Mitra. 2011. Animated construction of line drawings. *ACM Transactions on Graphics* 30, 6 (2011), 133.

Michael Garland and Paul S. Heckbert. 1997. Surface Simplification Using Quadric Error Metrics. In *Proceedings of ACM SIGGRAPH*. 209–216.

Timothy Gerstner, Doug DeCarlo, Marc Alexa, Adam Finkelstein, Yotam Gingold, and Andrew Nealen. 2013. Pixelated image abstraction with integrated user constraints. *Computers & Graphics* 37, 5 (2013), 333–347.

GNU Project. 2015. GNU Linear Programming Kit. (2015). http://www.gnu.org/software/glpk/glpk.html Version 4.57.

Roger Grosse, Micah K. Johnson, Edward H. Adelson, and William T. Freeman. 2009. Ground truth dataset and baseline evaluations for intrinsic image algorithms. In *International Conference on Computer Vision (ICCV'09)*. 2335–2342.

Sariel Har-Peled. 1999. *Geometric Approximation Algorithms and Randomized Algorithms for Planar Arrangements*. Ph.D. Dissertation. Tel-Aviv University.

Shi-Min Hu, Kun Xu, Li-Qian Ma, Bin Liu, Bi-Ye Jiang, and Jue Wang. 2013. Inverse image editing: Recovering a semantic editing history from a before-and-after image pair. *ACM Transactions on Graphics* 32, 6 (2013), 194.

Tao Ju, Scott Schaefer, and Joe Warren. 2005. Mean value coordinates for closed triangular meshes. *ACM Transactions on Graphics* 24, 3 (2005), 561–566.

Paul Kubelka. 1948. New contributions to the optics of intensely light-scattering materials. part I. *Journal of the Optical Society of America* 38, 5 (1948), 448–448.

Paul Kubelka and Franz Munk. 1931. An article on optics of paint layers. *Zeitschrift für Technische Physik* 12, 593–601 (1931).

Jason Lawrence, Aner Ben-Artzi, Christopher DeCoro, Wojciech Matusik, Hanspeter Pfister, Ravi Ramamoorthi, and Szymon Rusinkiewicz. 2006. Inverse shade trees for non-parametric material representation and editing. *ACM Transactions on Graphics* 25, 3 (July 2006), 735–745.

Anat Levin, Dani Lischinski, and Yair Weiss. 2008a. A closed-form solution to natural image matting. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 30, 2 (2008), 228–242.

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

Anat Levin, Assaf Zomet, and Yair Weiss. 2004. Separating reflections from a single image using local features. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR'04)*. 306–313.

Sharon Lin and Pat Hanrahan. 2013. Modeling how people extract color themes from images. In *Proceedings of ACM SIGCHI*.

Jingwan Lu, Stephen DiVerdi, Willa A. Chen, Connelly Barnes, and Adam Finkelstein. 2014. RealPigment: Paint compositing by example. In *Non-Photorealistic Animation and Rendering (NPAR'14)*. 21–30.

James McCann and Nancy Pollard. 2009. Local layering. *ACM Transactions on Graphics* 28, 3 (2009), 84.

James McCann and Nancy Pollard. 2012. Soft stacking. *Computer Graphics Forum* 31, 2 (2012), 469–478.

Mathieu Nancel and Andy Cockburn. 2014. Causality: A conceptual model of interaction history. In *Proceedings of ACM SIGCHI*. 1777–1786.

Peter O'Donovan, Aseem Agarwala, and Aaron Hertzmann. 2011. Color Compatibility from Large Datasets. *ACM Transactions on Graphics* 30, 4, Article 63 (2011), 63:1–63:12.

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

Christian Richardt, Jorge Lopez-Moreno, Adrien Bousseau, Maneesh Agrawala, and George Drettakis. 2014. Vectorising bitmaps into semi-transparent gradient layers. *Computer Graphics Forum (Proceedings of EGSR)* 33, 4 (2014), 11–19.

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

Bernard Sarel and Michal Irani. 2004. Separating transparent layers through layer information exchange. In *Proceedings of the European Conference on Computer Vision (ECCV'04)*.

Li Shen, Tan Ping, and Stephen Lin. 2008. Intrinsic image decomposition with non-local texture cues. In *Computer Vision and Pattern Recognition (CVPR'08)*.

Alvy Ray Smith and James F. Blinn. 1996. Blue screen matting. In *ACM SIGGRAPH Conference Proceedings*. 259–268.

Kartic Subr, Cyril Soler, and Frédo Durand. 2009. Edge-preserving multi-scale image decomposition based on local extrema. *ACM Transactions on Graphics* 28, 5, Article 147 (2009), 147:1–147:9.

Richard Szeliski, Shai Avidan, and P. Anandan. 2000. Layer extraction from multiple images containing reflections and transparency. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR'00)*.

Jianchao Tan, Marek Dvorožňák, Daniel Sýkora, and Yotam Gingold. 2015. Decomposing Time-Lapse Paintings into Layers. *ACM Transactions on Graphics* 34, 4 (2015), 61:1–61:10.

Ole Tange. 2011. GNU Parallel: The command-line power tool. *Login: The USENIX Magazine* 36, 1 (Feb. 2011), 42–47. http://www.gnu.org/s/parallel.

Andrew P. Witkin. 1983. Scale-space filtering. In *International Joint Conference on Artificial Intelligence*. Palo Alto, 1019–1022.

Songhua Xu, Yingqing Xu, Sing Bing Kang, David H. Salesin, Yunhe Pan, and Heung-Yeung Shum. 2006. Animating Chinese paintings through stroke-based decomposition. *ACM Transactions on Graphics* 25, 2 (2006), 239–267.

Juyong Zhang, Bailin Deng, Zishun Liu, Giuseppe Patanè, Sofien Bouaziz, Kai Hormann, and Ligang Liu. 2014. Local barycentric coordinates. *ACM Transactions on Graphics* 33, 6, Article 188 (Nov. 2014), 188:1–188:12.

Ciyou Zhu, Richard H. Byrd, Peihuang Lu, and Jorge Nocedal. 1997. Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization. *ACM Transactions on Mathematical Software* 23, 4 (Dec. 1997), 550–560.

Douglas E. Zongker, Dawn M. Werner, Brian Curless, and David H. Salesin. 1999. Environment Matting and Compositing. In *ACM SIGGRAPH Conference Proceedings*. 205–214.