

Decomposing Digital Paintings into Layers via RGB-space Geometry

Jianchao Tan*
George Mason University

Jyh-Ming Lien†
George Mason University

Yotam Gingold‡
George Mason University

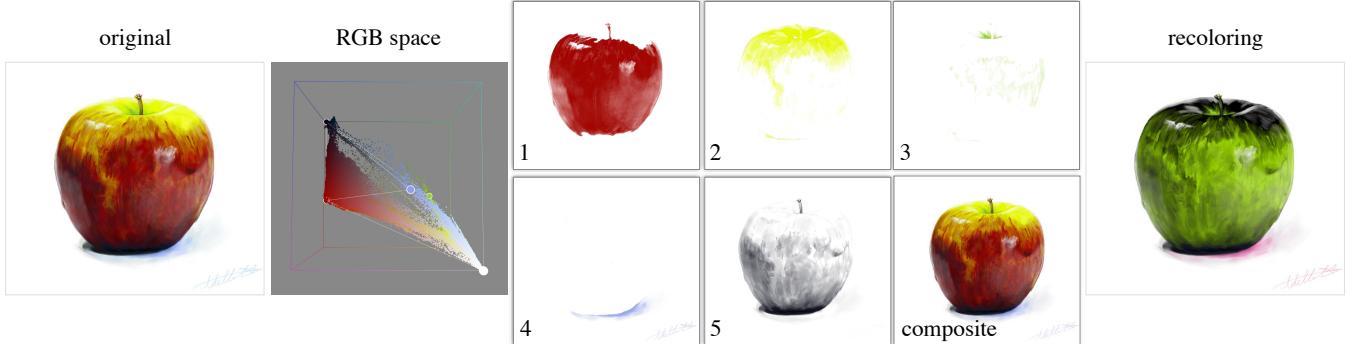


Figure 1: Given a digital painting, we analyze the geometry of its pixels in RGB-space, resulting in a translucent layer decomposition, which makes difficult edits simple to perform. Artwork © Adelle Chudleigh.

Abstract

In digital painting software, layers organize paintings. However, layers are not explicitly represented, transmitted, or published with the final digital painting. We propose a technique to decompose a digital painting into layers. In our decomposition, each layer represents a coat of paint of a single paint color applied with varying opacity throughout the image. Our decomposition is based on the painting’s RGB-space geometry. In RGB-space, a geometric structure is revealed due to the linear nature of the standard Porter-Duff [1984] “over” pixel compositing operation. The vertices of the convex hull of pixels in RGB-space suggest paint colors. Users choose the degree of simplification to perform on the convex hull, as well as a layer order for the colors. We solve a constrained optimization problem to find maximally 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 re-editing.

CR Categories: I.3.7 [Computer Graphics]: Picture/Image Generation—Bitmap and framebuffer operations I.4.6 [Image Processing and Computer Vision]: Segmentation—Pixel classification;

Keywords: images, surfaces, depth, time, video, channel, segmentation, layers, photoshop, painting

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* paintings. However, layers are not explicitly represented, transmitted, or

published with the final digital painting. Without layers, simple edits become extremely challenging, such as altering the color of a coat without inadvertently affecting a scarf placed overtop. Moreover, layers require artists to remember to create or switch layers when editing different parts of the painting.

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

$$A_\alpha A_{RGB} + (1 - A_\alpha)B_{RGB} \quad (1)$$

where the translucent pixel A is placed over the pixel B with translucency $\alpha \in [0, 1]$. Our technique automatically extracts paint colors, and supports user intervention to choose a simplified palette with fewer colors. Given a user-provided ordering of the colors, our technique computes per-pixel opacity values.¹ The result is a sequence of layers that reproduce the original painting when composited.

In RGB-space, the pixels of a digital painting reveal a hidden geometric structure (Figure 2). This geometric structure offers clues about the painting’s editing history. This structure results from the linearity of the “over” compositing operation (Equation 1), which is the de facto standard for applying digital paint with transparency. In this model, the paint color acts as a linear attractor in RGB-space. Affected pixels move towards the paint color via linear interpolation; the painted stroke’s transparency determines the strength of attraction, or interpolation parameter.

Our **contributions** are:

- The geometric analysis of RGB-space to determine the colors of paint used to create a digital painting (Section 4). We make use of the RGB-space convex hull to obtain a small color palette capable of reproducing all others.
- An optimization-based approach to compute per-layer, per-pixel opacity values (Section 5). Our approach regularizes the original, underconstrained problem with terms that balance translucency and spatial coherence.

¹The ordering is necessary, because the “over” compositing operation is not commutative.

*e-mail:jtan8@gmu.edu

†jmlien@gmu.edu

‡ygingold@gmu.edu

The result of these contributions is a technique that decomposes a single image, a digital painting, into translucent layers. Our decomposition enables the structured re-editing of digital paintings.

2 Related Work

Single-Image Decomposition Richardt et al. [2014] investigated a similar problem with the goal of producing editable vector 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 targets bitmap graphics, which are a less constrained domain. Our approach also requires much less user input. For comparison, we decompose an image from their paper (the light in Figure 6, row 6).

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 (a) pixels to have independent layer orders and (b) 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). This decomposition is suitable for photographs of illuminated objects, but not 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.

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.

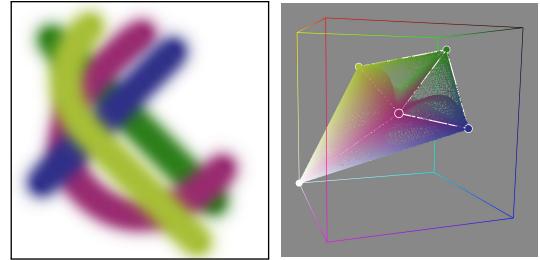


Figure 2: In RGB-space, the pixels of a digital painting (left) lie in the convex hull of the original paint colors (right). This is due to the linearity of the standard “over” blending operation [Porter and Duff 1984]. This linearity manifests as linear elements such as lines, faces, and solid polyhedra.

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 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. [Levin et al. 2008a; Levin et al. 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 as-binary-as-possible opacity values and at most three layers present in a small image patch. 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 paintings, in which physical assumptions are not valid and there are typically more than two layers.

3 Overview

The first half of our pipeline identifies the colors used to paint the image (Section 4). All colors in the image must lie within the convex hull of these colors; therefore we first compute the exact convex hull of the painting’s pixels in RGB-space. We then compute a simpler, approximate convex hull with user-tunable parameters. The simplification is computed in two stages, first by plane fitting to hull faces, and then by clustering the resulting vertices. These simplified vertices are the paint colors.

The second half of our pipeline takes as input a user-provided ordering of paint colors (RGB), and computes the corresponding per-pixel opacity values to produce RGBA layers. Each layer models a coat of paint. Our computation solves a polynomial system of equations via energy minimization. The polynomial equations express that the composition of all layers should reproduce the input image. To solve

this underconstrained problem, we introduce terms to maximize translucency and spatial coherence.

4 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 linear combination of the original paint colors \mathbf{c}_i : $\mathbf{p} = \sum w_i \mathbf{c}_i$ for some weights $w_i \in [0, 1]$ with $\sum w_i = 1$. Note that in this expression, the w_i are not opacity values. This 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, 2, and 6 display pairs of images and their pixels in RGB-space.

To identify the colors used to paint the digital image, we first compute the RGB-space convex hull of all observed pixel colors. In practice, if a paint color was always applied semi-transparently, the convex hull will be overly complex (too many vertices). This is because semi-transparent paint will not produce any pixels with the paint color itself. This manifests as “cut corners” or extra faces in the convex hull. To simplify the convex hull, we uniformly sample points on the surface of the hull and then perform plane fitting on the samples. We iteratively (a) find the best-fitting plane to the samples via RANSAC and (b) remove the associated samples. Iteration terminates when fewer than a small fraction of the samples remain (*termination fraction* in Table 1). RANSAC requires a distance threshold parameter to determine how close a point must be to a plane to be considered an inlier; we use 3.0 for all our examples. Finally, we recompute the convex region bounded by the set of oriented planes [de Berg et al. 2008]. The vertices of the convex region comprise the identified paint colors. The oriented planes are positioned in space such that nearly all points are inside the half-space and the vertices of the convex region are within the RGB cube. The parameter which controls the planes’ positions is the fraction of all points inside the half-space (*inside fraction* in Table 1).

We involve the user in the simplification process, as it is difficult to find a set of parameters that provide satisfactory results in all cases. We allow the user to choose RANSAC parameters. We also allow the user to further simplify the identified paint colors by choosing the bandwidth of a mean-shift [Comaniciu and Meer 2002] clustering operation performed on the colors or by directly removing some colors. “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 (Section 5) 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. The next stage of our algorithm determines the per-pixel opacity of each layer.

5 Determining Layer Opacity

The final stage of our algorithm computes opacity values for each layer, at each pixel. This stage takes as input globally ordered layer colors. Let $\{\mathbf{c}_i\}$ be the global, ordered sequence of RGB layer colors, and let \mathbf{c}_0 be the opaque background color. Then, at each pixel, the observed color \mathbf{p} can be expressed as the recursive

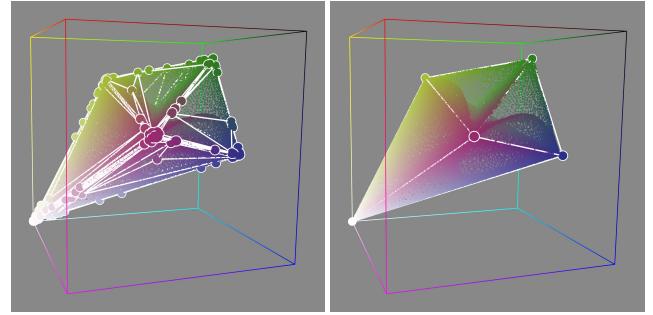


Figure 3: The original convex hull (left) and the simplified hull (right).

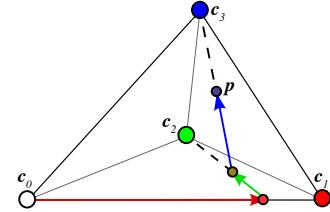


Figure 4: A pixel can be represented as a path from the background color \mathbf{c}_0 towards each of the other colors $\mathbf{c}_1, \mathbf{c}_2, \mathbf{c}_3$ in turn. The opacity values α_i determine the length of each arrow as a fraction of the entire line segment to \mathbf{c}_i .

application of “over” compositing (Equation 1):

$$\mathbf{p} = \mathbf{c}_n + \sum_{i=1}^n \left[(\mathbf{c}_{i-1} - \mathbf{c}_i) \prod_{j=i}^n (1 - \alpha_j) \right] \quad (2)$$

where α_i is the opacity of \mathbf{c}_i . Since colors \mathbf{p} and \mathbf{c}_i are three dimensional (RGB), this is a system of 3 polynomial equations with $\#layer$ unknowns. (For translucent RGBA input images, premultiplied colors should be used. Equation 2 becomes a system of 4 polynomial equations. The background layer \mathbf{c}_0 becomes transparent—the zero vector—while the remaining global layers colors \mathbf{c}_i are treated as opaque.)

There will always be at least one solution, because \mathbf{p} lies within the convex hull of the \mathbf{c}_i . When the number of layers is less than or equal to 3 (excluding the translucent background in case of RGBA), there is, in general, a unique solution. It can be obtained directly by projecting \mathbf{p} along the line from the top-most layer color \mathbf{c}_n onto the simplex formed by $\mathbf{c}_0 \dots \mathbf{c}_{n-1}$, and so on recursively (Figure 4). However, if a layer is opaque (other than the bottom layer) or the number of layers is greater than three, there are infinitely many solutions.² (For numerical reasons, it is problematic if a layer is nearly opaque—a situation which arises often.)

Regularization To choose among the solutions, we introduce two regularization terms. Our first regularization term penalizes opacity; absent additional information, a completely occluded layer should be transparent.

$$E_{opaque} = \sum_{i=1}^n -(1 - \alpha_i)^2 \quad (3)$$

²If a solution with at most three non-transparent layers *per-pixel* is desired, one can choose, for each pixel, any tetrahedron using convex hull vertices that encloses it. For example, one may choose for a pixel the tetrahedron with smallest total alpha.

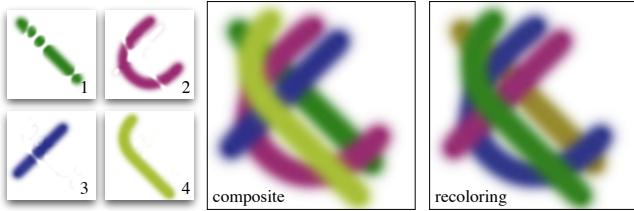


Figure 5: Decomposed layers, their composition, and subsequent recoloring from the example in Figures 2 and 3.

Note that we do not naively minimize α^2 , because (intuitively) $1 = 0^2 + 1^2 > \epsilon^2 + (1 - \epsilon)^2 = 1 - 2\epsilon(1 - \epsilon)$. In words, the squared opacity decreases when reducing the opacity of an opaque layer and increasing the opacity of a transparent layer. With our expression, the situation is reversed: $-1 = -(1 - 0)^2 - (1 - 1)^2 < -(1 - \epsilon)^2 - (1 - (1 - \epsilon))^2 = -1 + 2\epsilon$. Our energy decreases when reducing the opacity of a nearly transparent layer and increasing the opacity of a nearly opaque layer. As a result, we obtain a sparser solution (see inset).

For spatial continuity, a local minimum of E_{opaque} alone is sometimes preferable to the global minimum. Our second regularization term is the Laplacian energy of the α_i values.

$$E_{spatial} = \sum_{i=1}^n (\Delta \alpha_i)^2 \quad (4)$$

where $\Delta \alpha_i$ is the Laplace operator, or difference between the pixel's α_i and the average of its neighboring pixels' α_i 's. (We have obtained similar and sometimes better results with the Dirichlet energy, $\sum (\nabla \alpha_i)^2$.)

We minimize these two terms subject to the polynomial constraints (Equation 2) and $\alpha_i \in [0, 1]$. We implement the polynomial constraints as a least-squares penalty term $E_{polynomial}$:³

$$E_{polynomial} = \left(\mathbf{c}_n - \mathbf{p} + \sum_{i=1}^n \left[(\mathbf{c}_{i-1} - \mathbf{c}_i) \prod_{j=i}^n (1 - \alpha_j) \right] \right)^2 \quad (5)$$

The combined energy expression that we minimize is:

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

For the example shown in Figures 2 and 3, the recovered layers and reconstruction are shown in Figure 5.

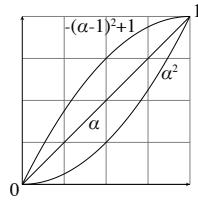
6 Results

Figures 1 and 6 show the decomposition of a variety of digital paintings, their recomposition, and layer-based edits. The decomposed layers reproduce the input image without visually perceptible differences. This is because the approximate convex hulls cover almost every pixel in RGB-space (Section 4), and the polynomial constraints in the energy minimization ensure that satisfying opacity values are chosen (Section 5). See Table 1 for parameters.

³The units of the three $E_{polynomial}$ expressions are squared color differences. Our RGB colors lie within $[0, 255]$, so the implied weight is 255^2 .

name	plane fitting termination fraction	convex hull reconstruction		optimization		
		inside fraction	mean-shift bandwidth	w_{opaque}	$w_{spatial}$	image size
apple	0.05	0.99	40	100	1000	500×453
Figure 2	0.01	0.99	6	20	800000	500×500
bird	0.01	0.999	15	200	10000	640×360
eye	0.1	0.999	10	100	1000	630×380
turtle	0.1	0.999	30	100	10000	525×250
fruits	0.05	0.999	40	100	1000	650×414
light	0.1	0.95	31	100	1000	504×538
scrooge	0.1	0.97	31	100	1000	410×542
robot	0.1	0.995	30	100	1000	450×600

Table 1: Parameters used in our pipeline.



These decomposed layer representations make it straightforward to isolate edits such as recoloring a part of the image. We have found parameter tuning to obtain a simplified convex hull to be straightforward. Choosing the layer order and optimization weights is more time consuming, as it can be difficult to predict which parameters will produce a meaningful set of layers. For example, when the set of colors includes shades of red and black, the boundary between the red layers may not match user expectations. Nevertheless, we have been able to obtain useful decompositions for a large variety of examples.

Implementation We use QHull [Barber et al. 1996] for convex hull computation, PCL [Rusu and Cousins 2011] for RANSAC plane fitting, and L-BFGS-B [Zhu et al. 1997] for optimization. Our algorithms were written in Python and vectorized using NumPy/SciPy. Our implementation is not multi-threaded.

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. This progressive optimization reduced running time by a factor of four.

Performance Paint color identification (Section 4) 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 5) entails solving a nonlinear optimization procedure. As we implemented our optimization in a multi-resolution manner, the user is able to quickly see a preview of the result (e.g. less than 20 seconds for a 100×64 -pixel image). This is important for experimenting with different layer orders and energy weights. Larger images are computed progressively as the multi-resolution optimization converges on smaller images; for a 500 or 1000 pixels wide image, the final optimization can take anywhere from 20 minutes to an hour or longer to converge.

7 Conclusion

The RGB-space of a digital painting contains a “hidden” geometric structure. Namely, the convex hull of this structure identifies the original paint colors. Given a set of colors and an order, our constrained optimization decomposes the painting into a useful set of translucent layers.

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 towards translucent and spatially coherent solutions. However, this still may not match user expectations. Second,

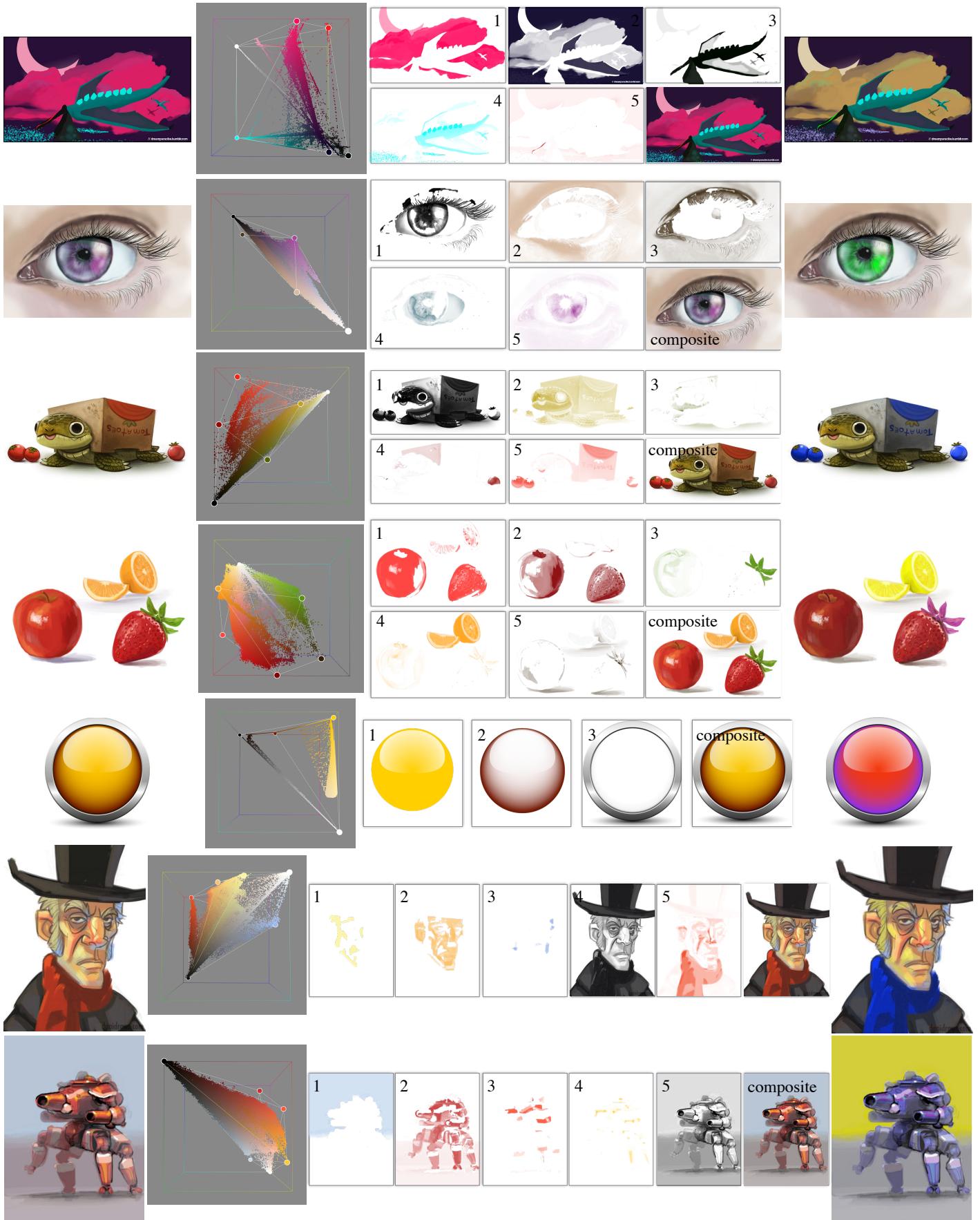


Figure 6: Digital paintings, their RGB-space simplified convex hulls, decomposed layers, recomposition, and re-edits. From top to bottom, original artwork © Karl Northfell; reddit user TwitchHD; Piper Thibodeau; Ranivius; Roman Sotola; Dani Jones; Adam Saltsman.

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, hidden layer colors—those 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 layers colors to change but found it difficult to control. A related problem is paintings of e.g. 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, to choose the layer order, and to choose optimization weights. While the number of layer orderings is factorial in the number of layers, heuristics may exist to narrow the search space. 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 the effects of non-linear digital edits in RGB-space, and to consider non-linear color compositing operations, such as the Kubelka-Munk mixing and layering equations [Tan et al. 2015]. This would allow us to decompose scans of physical paintings and to support a larger variety of digitally created images. We also plan to explore heuristics for automatically choosing a layer order and degree of simplification to apply to the convex hull. Although we focus on digital paintings, the colors of any image (physical painting or photograph) can still be covered by a convex hull in RGB-space, and therefore decomposed into “paint” layers with our approach. Finally, we plan to apply our per-pixel layer opacity values towards segmentation; layer translucency is a higher-dimensional and more meaningful feature than composited RGB color.

Acknowledgements

We are grateful to Neil Epstein for an interesting conversation about the algebraic structure of the solutions to the multi-layer polynomial system. This work was supported in part by the United States National Science Foundation (IIS-1451198, IIS-1453018, IIS-096053, CNS-1205260, EFRI-1240459, and AFOSR FA9550-12-1-0238) and a Google research award. Several experiments were run on ARGO, a research computing cluster provided by the Office of Research Computing at George Mason University, VA. (URL:<http://orc.gmu.edu>).

References

- AMATI, C., AND BROSTOW, G. J. 2010. Modeling 2.5D plants from ink paintings. In *Proceedings of Eurographics Symposium on Sketch-Based Interfaces and Modeling Symposium*, 41–48.
- AUJOL, J.-F., AND KANG, S. H. 2006. Color image decomposition and restoration. *Journal of Visual Communication and Image Representation* 17, 4, 916–928.
- BARBER, C. B., DOBKIN, D. P., AND HUHDANPAA, H. 1996. The quickhull algorithm for convex hulls. *ACM Trans. Math. Softw.* 22, 4 (Dec.), 469–483.
- BONANNI, L., XIAO, X., HOCKENBERRY, M., SUBRAMANI, P., ISHII, H., SERACINI, M., AND SCHULZE, J. 2009. Wetpaint: Scraping through multi-layered images. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 571–574.
- BOUSSEAU, A., PARIS, S., AND DURAND, F. 2009. User-assisted intrinsic images. *ACM Trans. Graph.* 28, 5 (Dec.), 130:1–130:10.
- BUDSBERG, J. B. 2007. *Pigmented Colorants: Dependency on Media and Time*. Master’s thesis, Cornell Univrsity, Ithaca, New York, USA.
- CHANG, H., FRIED, O., LIU, Y., DiVERDI, S., AND FINKELSTEIN, A. 2015. Palette-based photo recoloring. *ACM Transactions on Graphics (Proc. SIGGRAPH)* 34, 4 (Aug.).
- COMANICIU, D., AND MEER, P. 2002. Mean shift: a robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 24, 5, 603–619.
- DE BERG, M., CHEONG, O., VAN KREVELD, M., AND OVERMARS, M. 2008. *Computational geometry: Algorithms and Applications*. Springer-Verlag.
- FARBMAN, Z., FATTAL, R., LISCHINSKI, D., AND SZELISKI, R. 2008. Edge-preserving decompositions for multi-scale tone and detail manipulation. In *ACM Transactions on Graphics (TOG)*, vol. 27, ACM, 67.
- FARID, H., AND ADELSON, E. H. 1999. Separating reflections from images by use of independent component analysis. *Journal of the Optical Society of America A* 16, 9, 2136–2145.
- FU, H., ZHOU, S., LIU, L., AND MITRA, N. J. 2011. Animated construction of line drawings. *ACM Transactions on Graphics* 30, 6, 133.
- GERSTNER, T., DECARLO, D., ALEXA, M., FINKELSTEIN, A., GINGOLD, Y., AND NEALEN, A. 2013. Pixelated image abstraction with integrated user constraints. *Computers & Graphics* 37, 5, 333–347.
- GROSSE, R., JOHNSON, M., ADELSON, E., AND FREEMAN, W. 2009. Ground truth dataset and baseline evaluations for intrinsic image algorithms. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2335–2342.
- HU, S.-M., XU, K., MA, L.-Q., LIU, B., JIANG, B.-Y., AND WANG, J. 2013. Inverse image editing: Recovering a semantic editing history from a before-and-after image pair. *ACM Transactions on Graphics* 32, 6, 194.
- LEVIN, A., ZOMET, A., AND WEISS, Y. 2004. Separating reflections from a single image using local features. In *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, vol. 1, I–306–I–313 Vol.1.
- LEVIN, A., LISCHINSKI, D., AND WEISS, Y. 2008. A closed-form solution to natural image matting. *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 30, 2, 228–242.
- LEVIN, A., RAV-ACHA, A., AND LISCHINSKI, D. 2008. Spectral matting. *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 30, 10, 1699–1712.
- MCCANN, J., AND POLLARD, N. 2009. Local layering. *ACM Transactions on Graphics* 28, 3, 84.
- MCCANN, J., AND POLLARD, N. 2012. Soft stacking. *Computer Graphics Forum* 31, 2, 469–478.
- NANCEL, M., AND COCKBURN, A. 2014. Causality: A conceptual model of interaction history. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 1777–1786.
- PORTER, T., AND DUFF, T. 1984. Compositing digital images. *ACM SIGGRAPH Computer Graphics* 18, 3, 253–259.

- RICHARDT, C., LOPEZ-MORENO, J., BOUSSEAU, A., AGRAWALA, M., AND DRETTAKIS, G. 2014. Vectorising bitmaps into semi-transparent gradient layers. *Computer Graphics Forum (Proceedings of EGSR)* 33, 4 (July), 11–19.
- RUSU, R., AND COUSINS, S. 2011. 3D is here: Point Cloud Library (PCL). In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, 1–4.
- SAREL, B., AND IRANI, M. 2004. Separating transparent layers through layer information exchange. In *Proceedings of the European Conference on Computer Vision (ECCV)*.
- SHEN, L., PING, T., AND LIN, S. 2008. Intrinsic image decomposition with non-local texture cues. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2008*, IEEE, 1–7.
- SMITH, A. R., AND BLINN, J. F. 1996. Blue screen matting. In *ACM SIGGRAPH Conference Proceedings*, 259–268.
- SUBR, K., SOLER, C., AND DURAND, F. 2009. Edge-preserving multiscale image decomposition based on local extrema. *ACM Trans. Graph.* 28, 5 (Dec.), 147:1–147:9.
- SZELISKI, R., AVIDAN, S., AND ANANDAN, P. 2000. Layer extraction from multiple images containing reflections and transparency. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, vol. 1, 246–253.
- TAN, J., DVOROŽNÁK, M., SÝKORA, D., AND GINGOLD, Y. 2015. Decomposing time-lapse paintings into layers. *ACM Transactions on Graphics* 34, 4.
- TANGE, O. 2011. GNU Parallel: The command-line power tool. *:login: The USENIX Magazine* 36, 1 (Feb), 42–47.
- WITKIN, A. P. 1983. Scale-space filtering. In *International Joint Conference on Artificial Intelligence*, 1019–1022.
- XU, S., XU, Y., KANG, S. B., SALESIN, D. H., PAN, Y., AND SHUM, H.-Y. 2006. Animating chinese paintings through stroke-based decomposition. *ACM Transactions on Graphics* 25, 2, 239–267.
- ZHU, C., BYRD, R. H., LU, P., AND NOCEDAL, J. 1997. Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization. *ACM Trans. Math. Softw.* 23, 4 (Dec.), 550–560.
- ZONGKER, D. E., WERNER, D. M., CURLESS, B., AND SALESIN, D. H. 1999. Environment matting and compositing. In *ACM SIGGRAPH Conference Proceedings*, 205–214.