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

Jianchao Tan George Mason University

Jose Echevarria Adobe Research

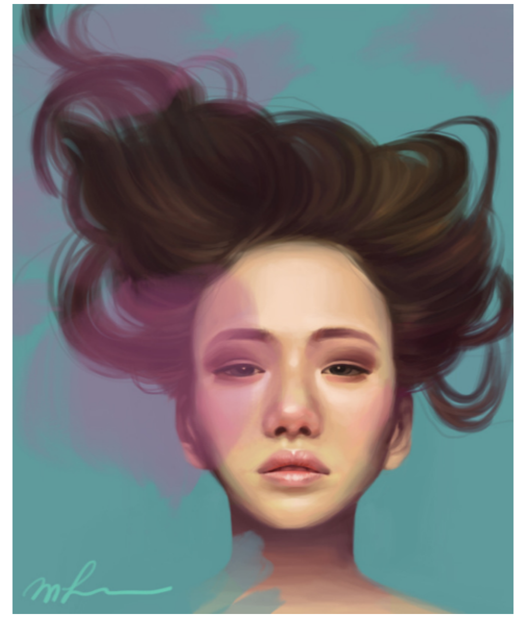
Yotam Gingold George Mason University

SIGGRAPH ASIA 2018 TOKYO

GEORGE MASON UNIVERSITY CraGL Creativity and Graphics Lab Adobe

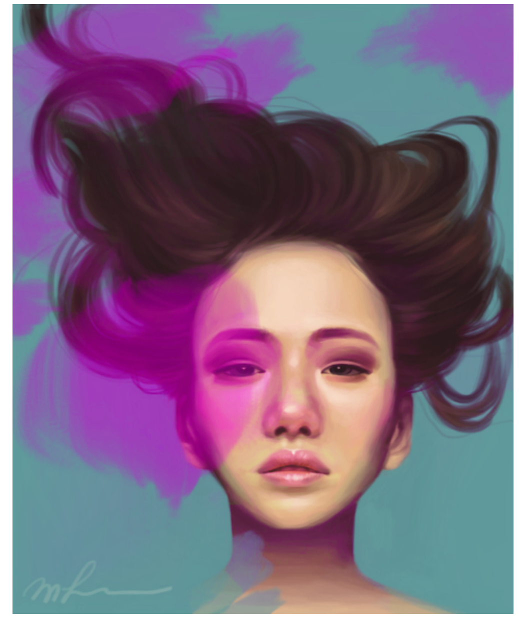
Hello everyone, thanks for coming to our talk.
This is Jose Echevarria, and I'd like to point out that Jianchao should be the one presenting this work. Unfortunately, he couldn't come because it would have complicated his visa status in the US.

Motivation: Layers Organize Images



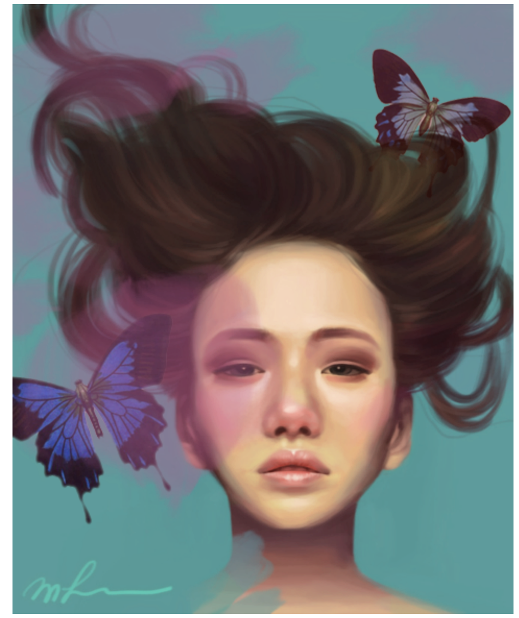
When creating images with digital software, creatives use layers to organize elements according to spatial, semantical or color relationships to name a few.

Motivation: Layers Organize Images



Such layers can then be used to perform tasks like selective recoloring

Motivation: Layers Organize Images

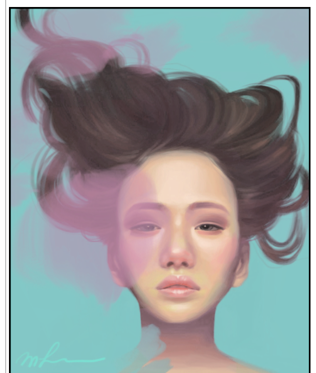


Or insert new elements into the scene effortlessly.

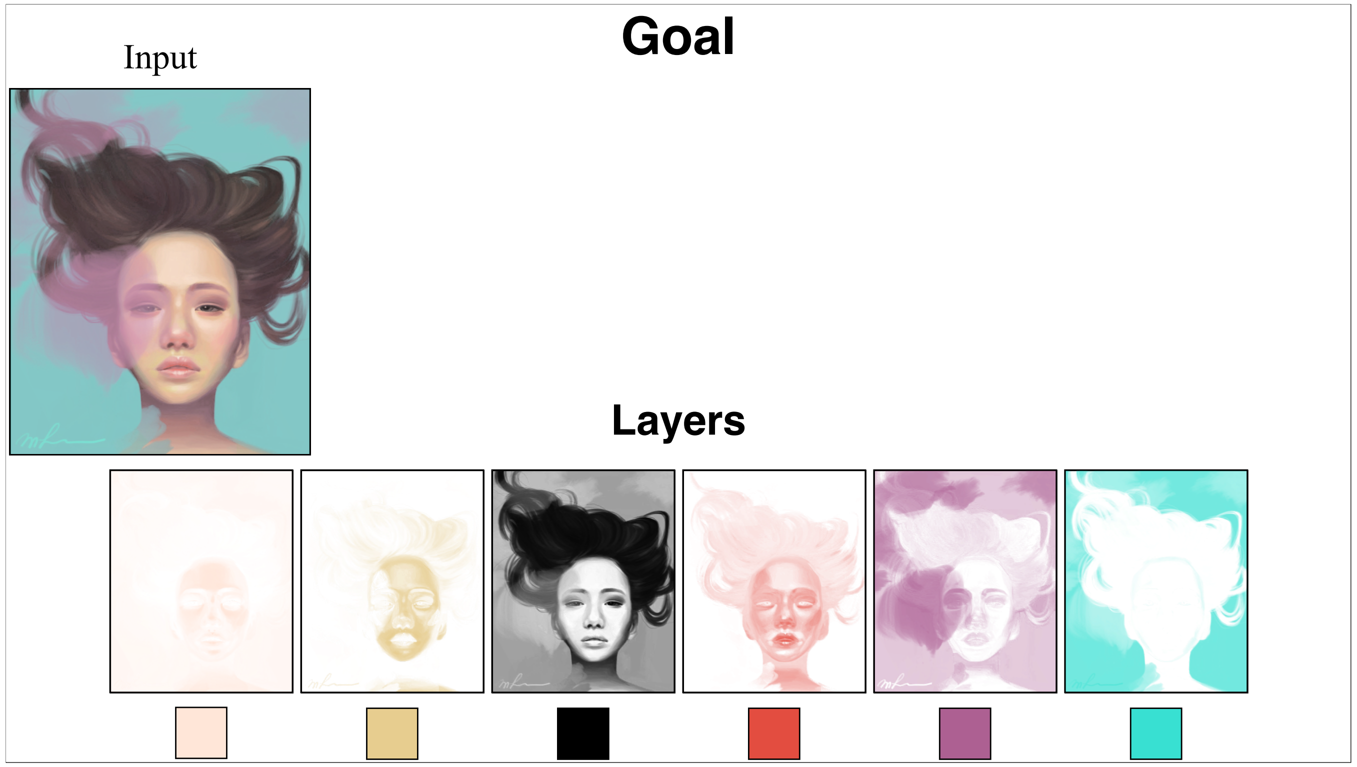
However, most of the images shared have been flattened to remove layers and reduce size; or they never had any layer at all (like in the case of photographs). In both cases, all we have are the 3 RGB channels.

Goal

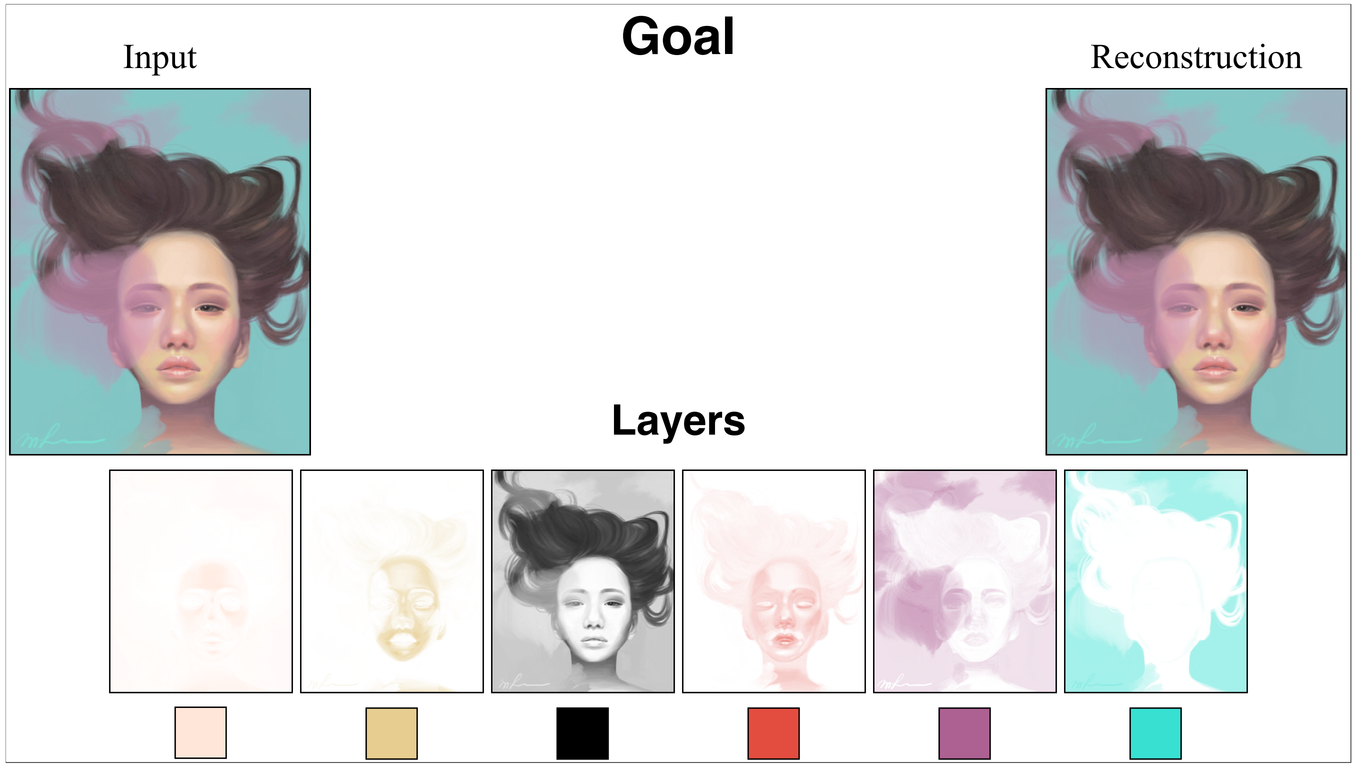
Input



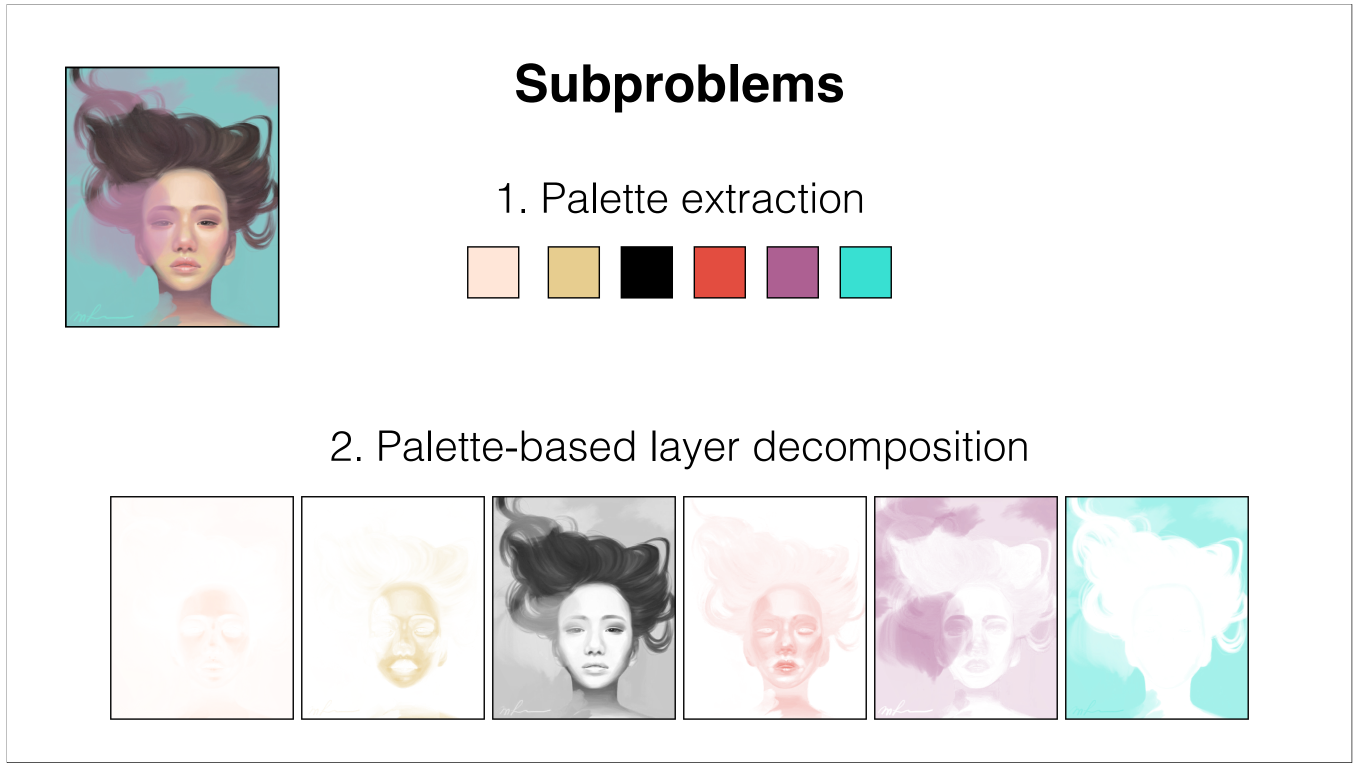
So, our main goal is to revert this situation, decomposing the input image into a set of layers that improve its editing capabilities



Our approach is to do that based on color and spatial relationships, which often capture semantical structures as well.
In particular, we are looking for a compact set of layers with per-pixel weights, according to the primary colors on the image



Those are the colors that abstract the actual ones, while allowing the reconstruction of both the original image, and the new ones resulting from editing any of those layers, with low error.

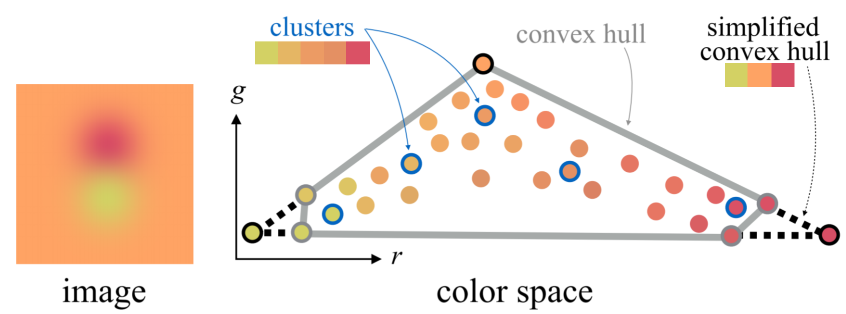


All this can be divided into two subproblems:

- Palette extraction
- Palette-based layer decomposition

Related Work

- Palette extraction for image editing
  - Shapira et al. [2009]
  - O'Donovan et al. [2011]
  - Lin et al. [2013]
  - Gerstner et al. [2013]
  - Chang et al. [2015]
  - Tan et al. [2016]

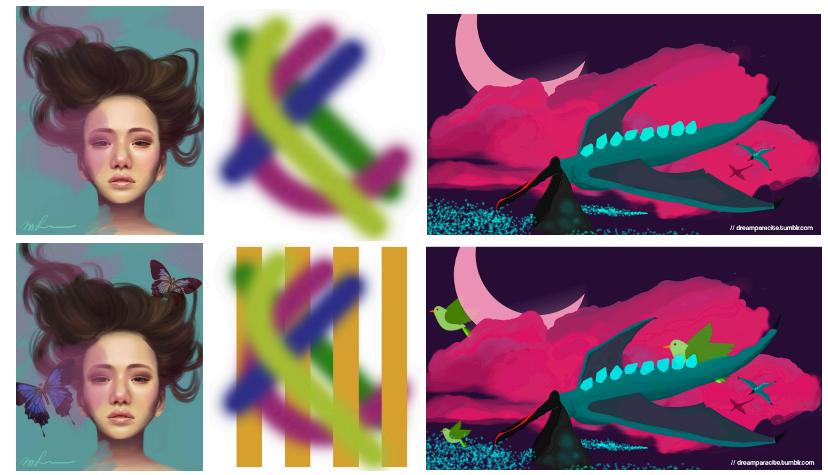


Decomposing Images into Layers via RGB-space Geometry [Tan et al. 2016]

Previous work on palette extraction and palette-based image editing have used different forms of clustering like k-means, to find the most relevant colors in the image.
However colors from clustering don't capture and don't reconstruct the rest of the colors in the image.
Tan et al, on the other hand, performs a convex hull simplification in RGB, producing a compact set of 'primary' colors, able to reconstruct the rest of the colors on the image.

Related Work

- Order-dependent translucent layers
  - Richardt et al. [2014]
  - Tan et al. [2015]
  - Tan et al. [2016]

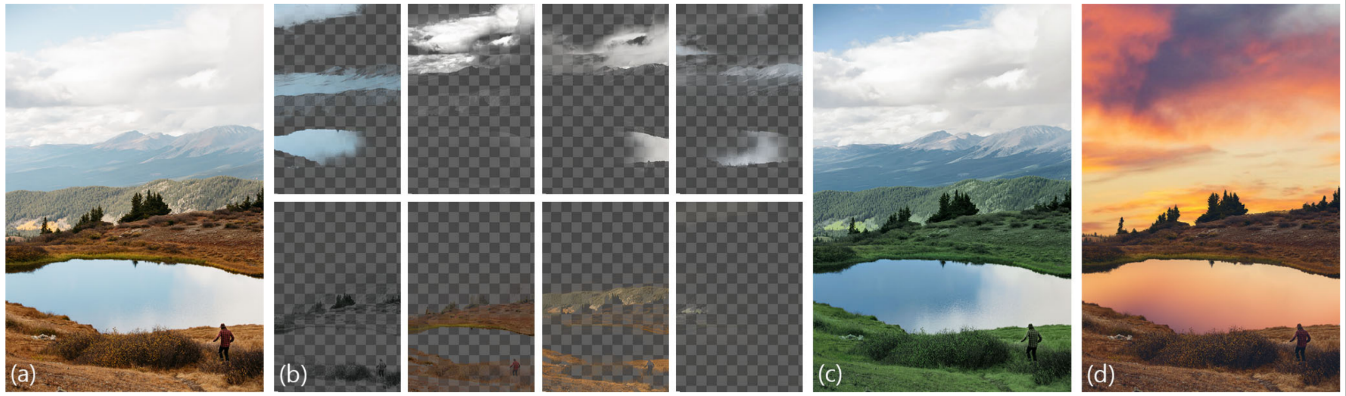


Decomposing Images into Layers via RGB-space Geometry [Tan et al. 2016]

There are also some work that decompose image into a set of RGBA layers for alpha blending, which is order dependent and allows insertion of new elements between the existing ones.

Related Work

- Order-independent additive-mixing layers
  - Lin et al. [2017]; Zhang et al. [2017], Aksoy et al. [2017].

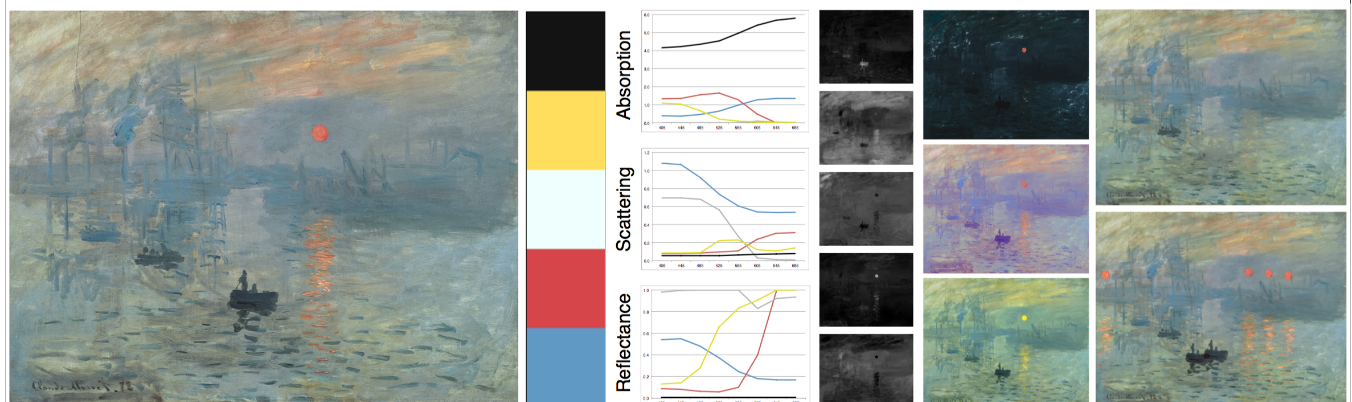


Unmixing-Based Soft Color Segmentation for Image Manipulation [Aksoy et al. 2017]

Other works decompose the image into a set of order-independent additive mixing layers. The most recent of these, Aksoy et al. 2017, but has layers which consist of color distributions, not single palette colors.

Related Work

- Physically-based layers
  - Abed et al. [2014]; Tan et al. [2015]; Aharoni-Mack et al. [2017]; Tan et al. [2018].

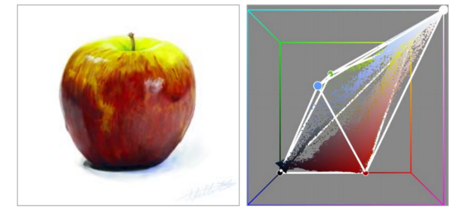


Pigmento: Pigment-Based Image Analysis and Editing [Tan et al. 2018]

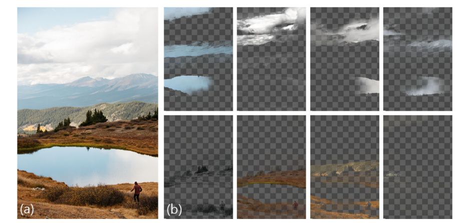
Beyond digital color compositing model, there are also several recent works focusing on the Kubelka-Munk pigment model, which can be seen as a physically-inspired nonlinear color compositing operation in our context. We do not consider multispectral or physical pigment parameters.

Our approach

- Geometry-based convex palettes
  - Simpler
  - More general



- Additive-mixing layers
  - Single colors
  - More general



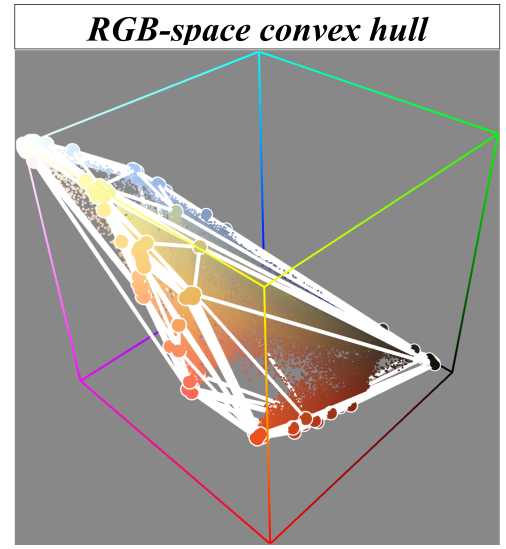
Comparing to previous different palette selection and layer decomposition methods. Our work chooses … (say each one).

Palette extraction

after we automatically extract the palette, we use the palette to help decompose image into additive mixing layers.

Convex hulls in RGB

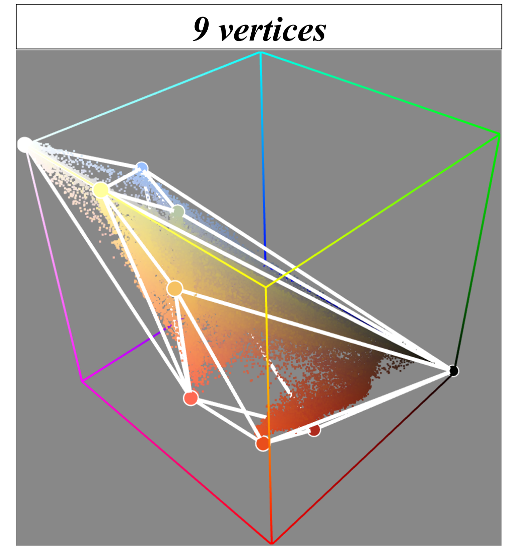
- Image colors show a convex structure in RGB [Tan et al. 2016]



The original convex hull can be simplified to any complexity level, convex palette structure in any simplified level will always keep colors points inside.

Palette Size

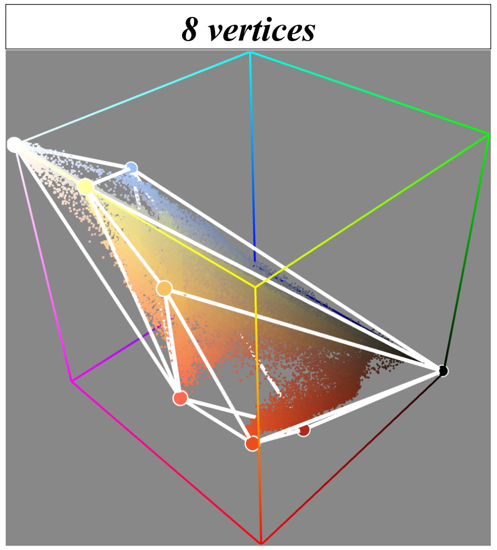
- The convex hull can be simplified to any complexity level.



The original convex hull can be simplified to any complexity level, convex palette structure in any simplified level will always keep colors points inside.

Palette Size

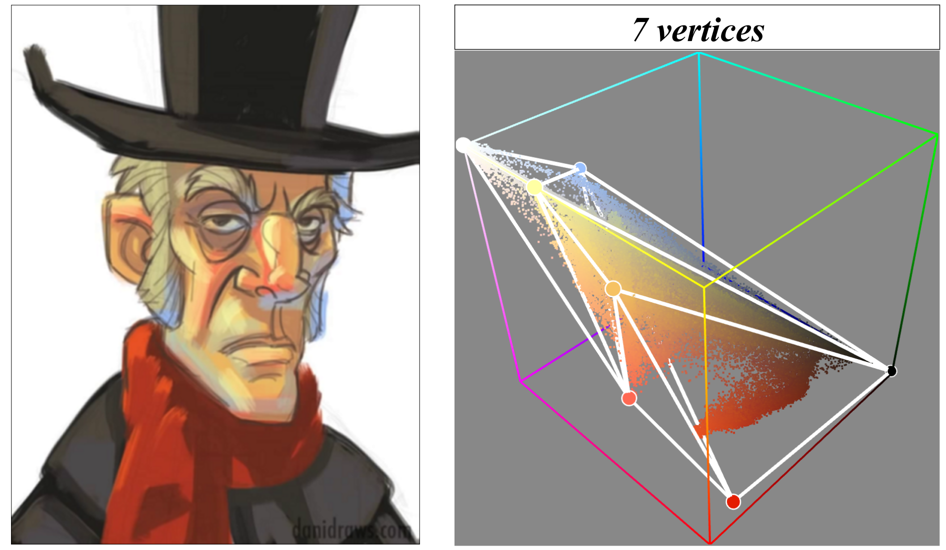
- The convex hull can be simplified to any complexity level.



The original convex hull can be simplified to any complexity level, convex palette structure in any simplified level will always keep colors points inside.

Palette Size

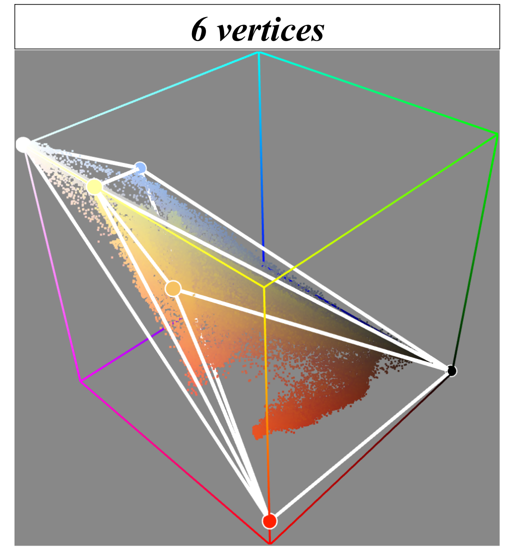
- The convex hull can be simplified to any complexity level.



The original convex hull can be simplified to any complexity level, convex palette structure in any simplified level will always keep colors points inside.

Palette Size

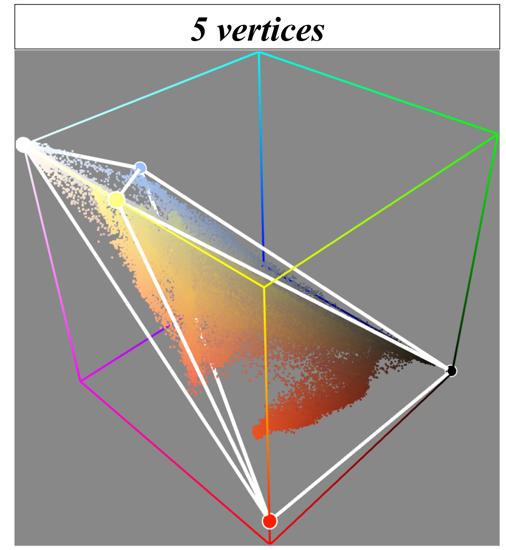
- The convex hull can be simplified to any complexity level.



The original convex hull can be simplified to any complexity level, convex palette structure in any simplified level will always keep colors points inside.

Palette Size

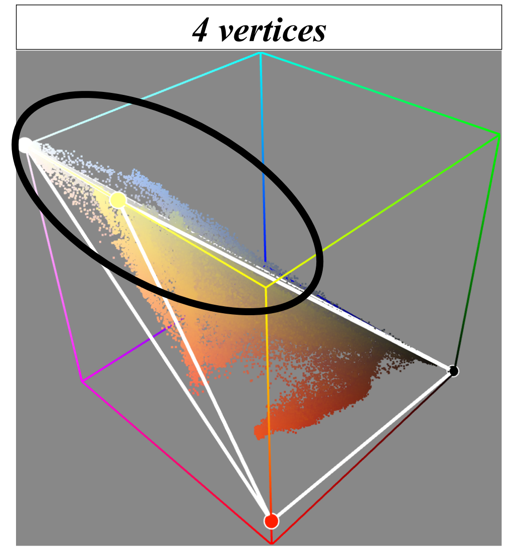
- The convex hull can be simplified to any complexity level.



The original convex hull can be simplified to any complexity level, convex palette structure in any simplified level will always keep colors points inside.

Palette Size

- The convex hull can be simplified to any complexity level.

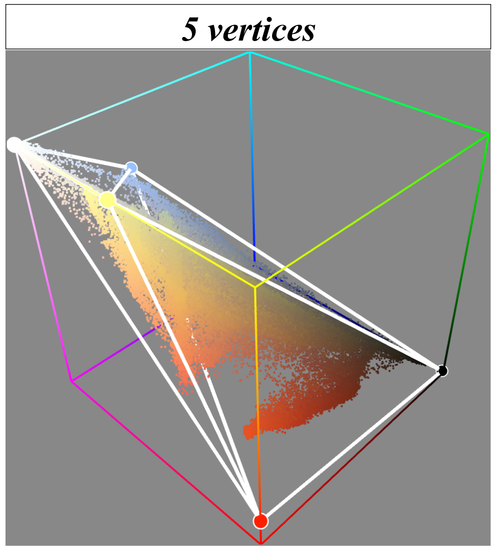


In Tan et al 2016, users manually choose the palette size. Ideally, the best palette is the smallest one that still encloses all the image colors.

We modify this to automatically choose the palette size based on tolerable reconstruction error. We use the total distance between the hull and points outside as the metric to stop simplification. Such distance in RGB space is also just the image color reconstruction errors. For this example, 4 vertices will leave a large portion of points (blue points) outside the hull.

Palette Size

- Our automatic error-bound simplification



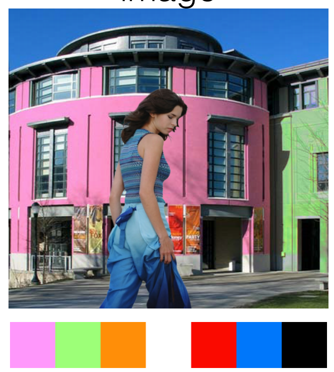
Our automatic termination finds a tradeoff between abstraction and reconstruction error.

Image Decomposition

after we automatically extract the palette, we use the palette to help decompose image into additive mixing layers.

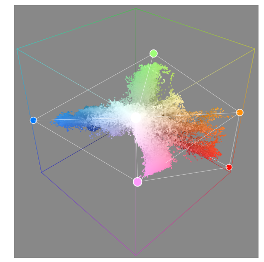
Extracting mixing weights

image



palette

RGB-space



Optimization

- Slow for high resolutions
- Many parameters to tune
- Per-image parameters

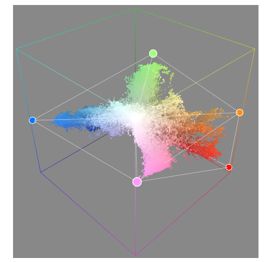
There are several previous methods that can decompose the image into mixing weights given a palette.
One category is to solve an optimization problem, considering spatial smoothness and sparsity regularizations. It is usually slow for large size image, and need tune several parameters for different regularization terms. And fixed parameters are not suitable for all examples.
For this example, the hand area has a obvious artifacts due to bad parameters for regularization term.

Extracting mixing weights

image

palette

RGB-space



~~**Optimization**~~

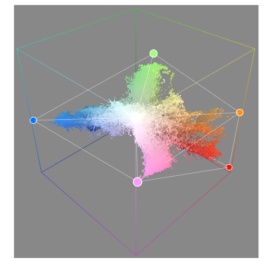
So the optimization method is not an optimal choice for this problem.

Extracting mixing weights

image

palette

RGB-space



~~Optimization~~

Generalized Barycentric Coordinates

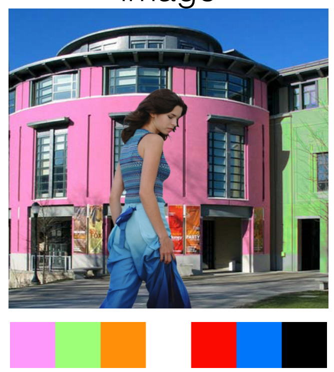
- Fast
- No parameters to tune
- Does not guarantee spatial smoothness

The other category is to use RGB space triangulation method to extract barycentric coordinates as palette's mixing weights. This is fast and no parameters to tune. However, this barycentric coordinates is smooth in RGB space, but it is not guaranteed to be smooth in image spatial domain.

For this example, you can find some noisy regions in jeans.

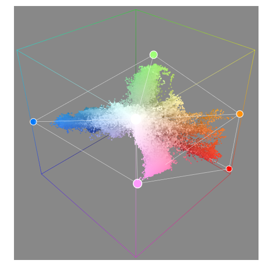
Extracting mixing weights

image



palette

RGB-space



~~Optimization~~

~~Generalized Barycentric Coordinates~~

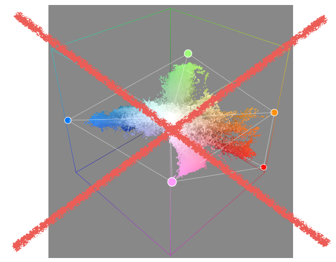
This RGB space barycentric coordinates method is also not an optimal choice.

Extracting mixing weights

image

palette

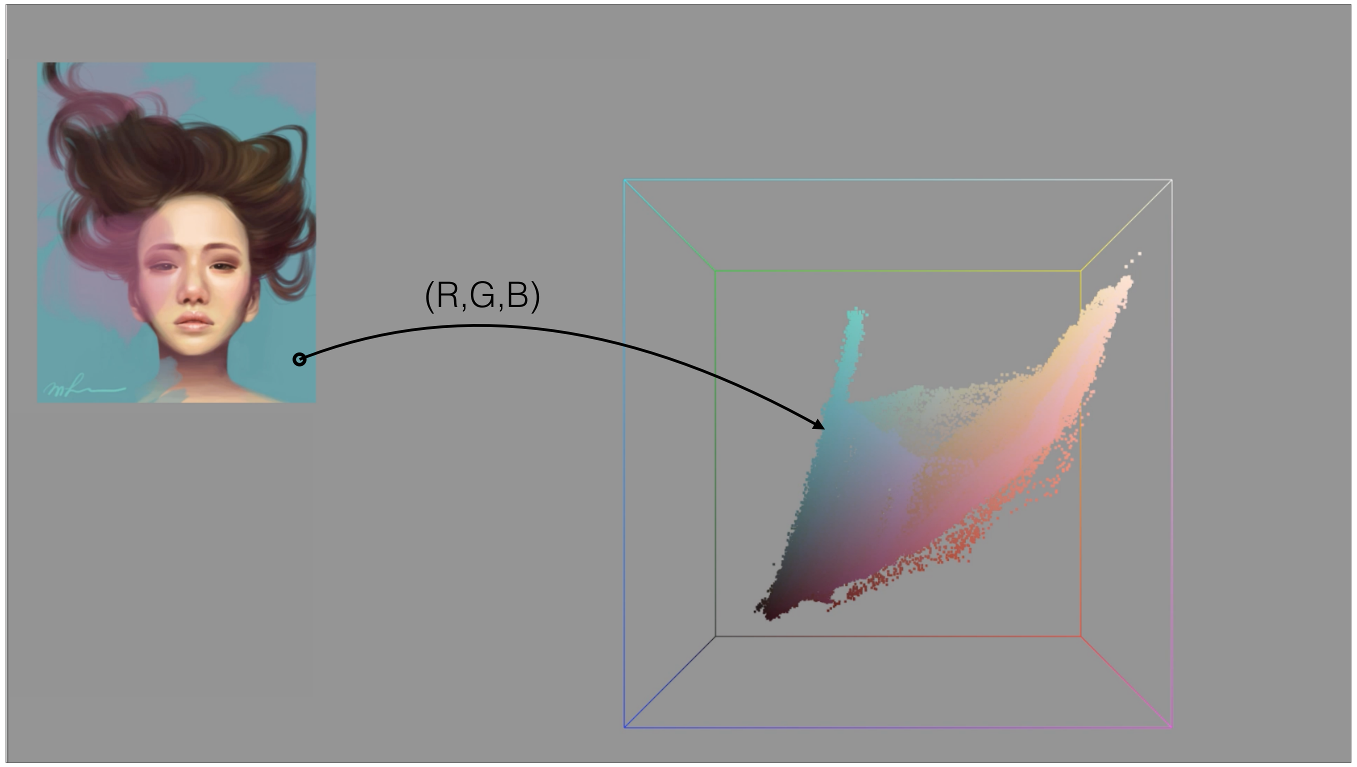
RGB-space



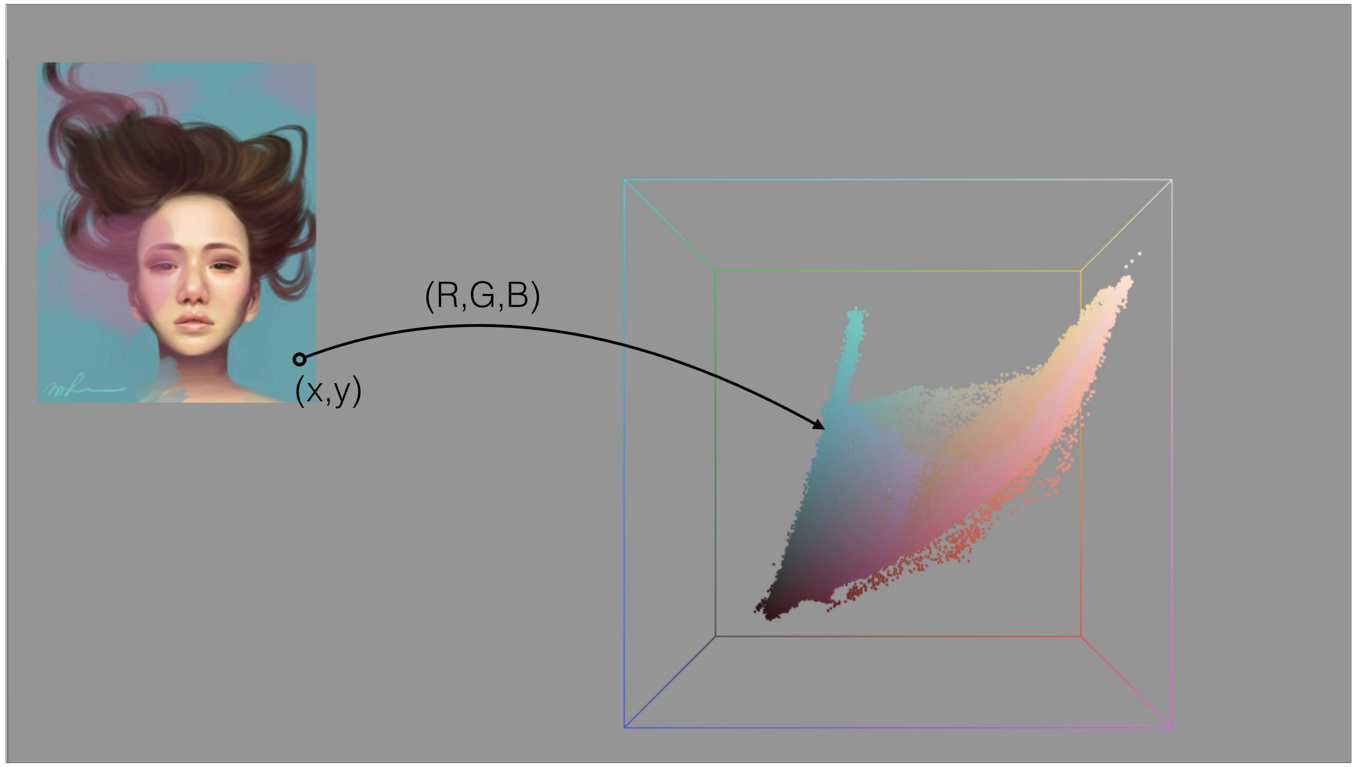
~~**Optimization**~~

~~**Generalized Barycentric Coordinates**~~

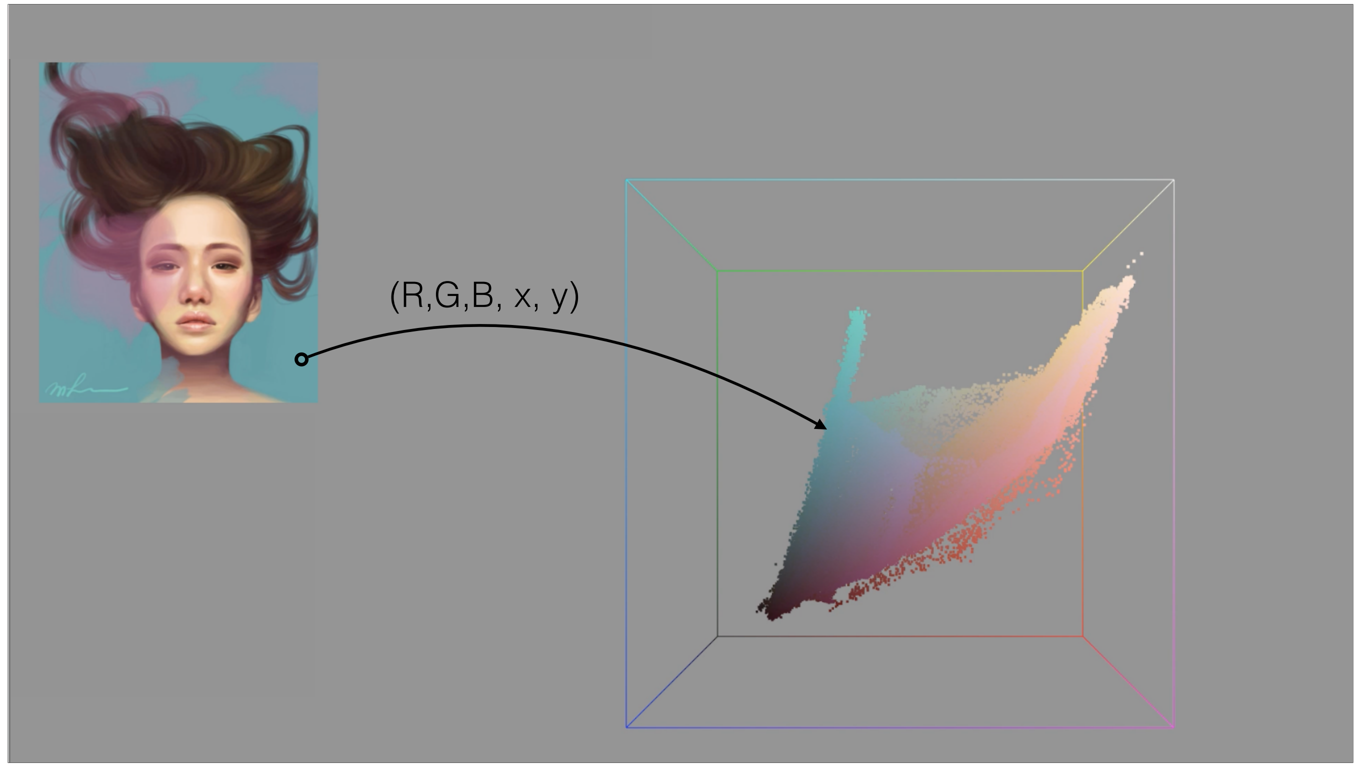
Methods in RGB space are all not optimal.
Now let's talk about our observations and intuitions to solve this problem elegantly.



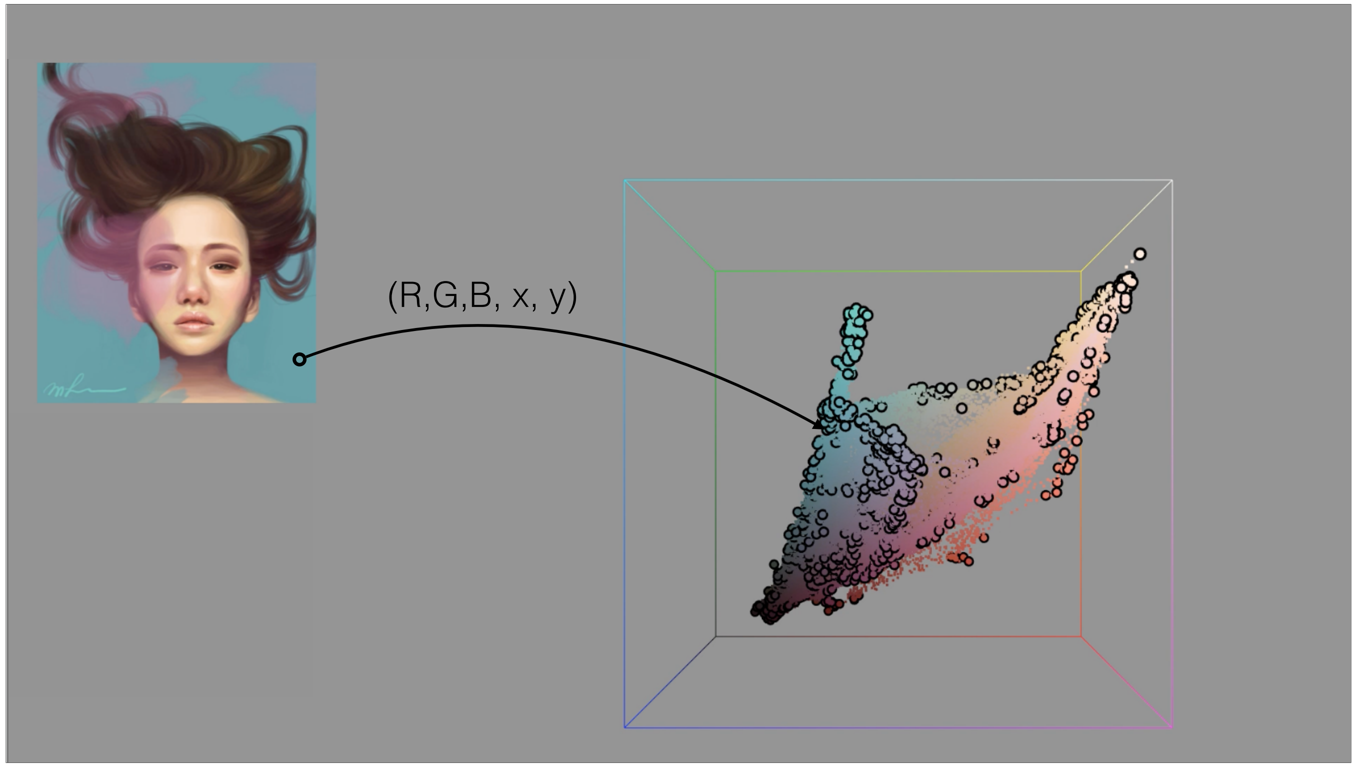
For this image, we can visualize its pixel colors in a 3D color-space via their R,G,B attributes.



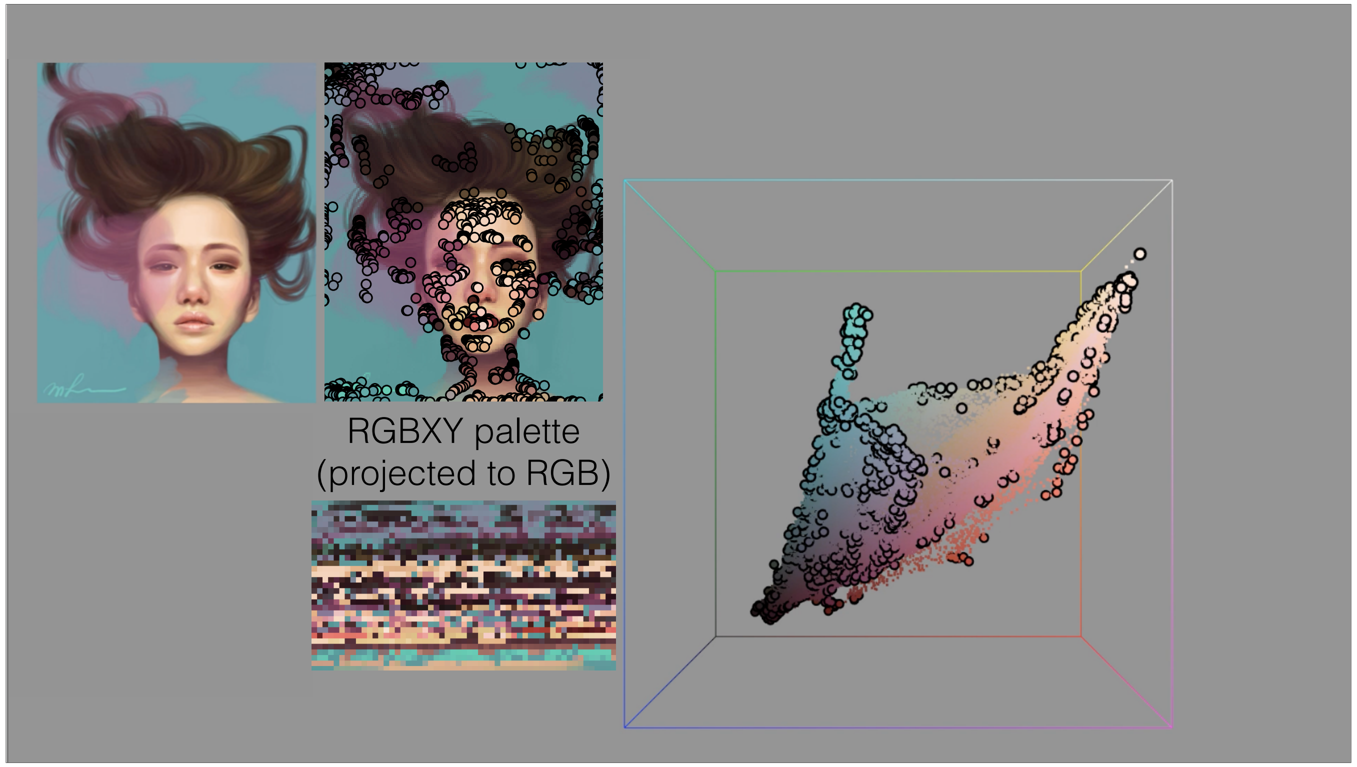
Pixels also have an x,y attribute it was not considered before



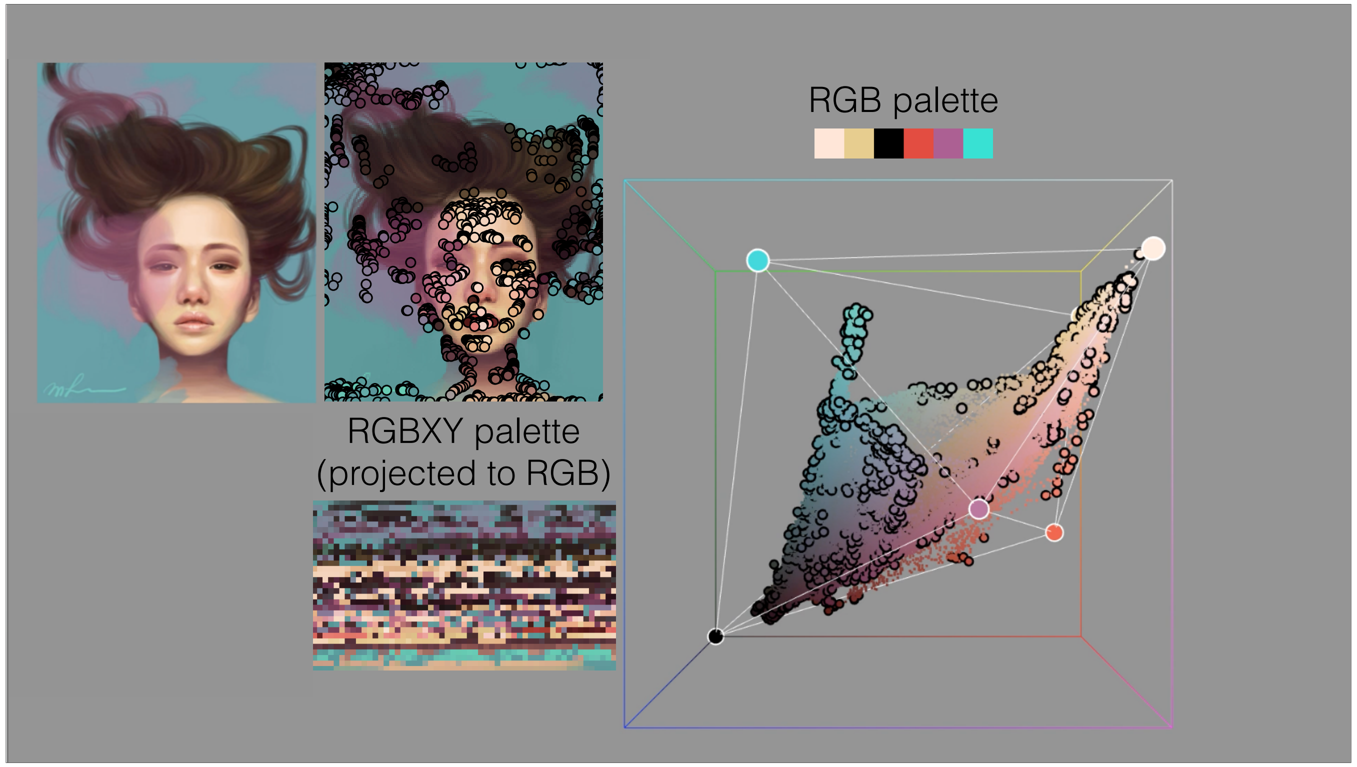
We can combine these attributes together to get a 5D point for each pixel. It’s hard to visualize. But geometric operations like the convex hull are still well defined.



Here is a visualization of the pixels chosen as the RGBXY convex hull vertices



Here is RGBXY convex hull vertices visualized on image. Just like every pixel lies inside the RGB convex hull, every pixel also lies inside the RGBXY convex hull.

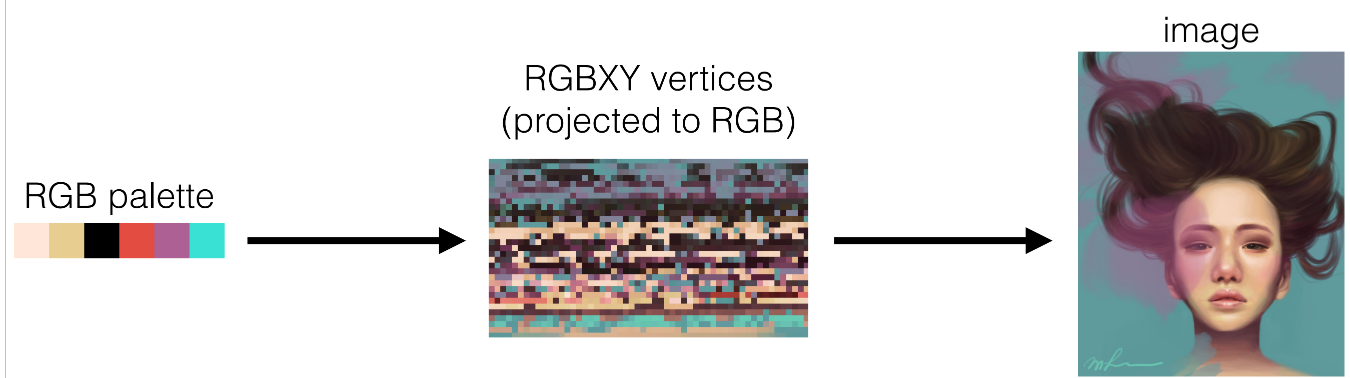


Conveniently, the RGBXY convex hull vertices, when projected back into RGB, lie inside the RGB convex hull.
This leads us to our approach.

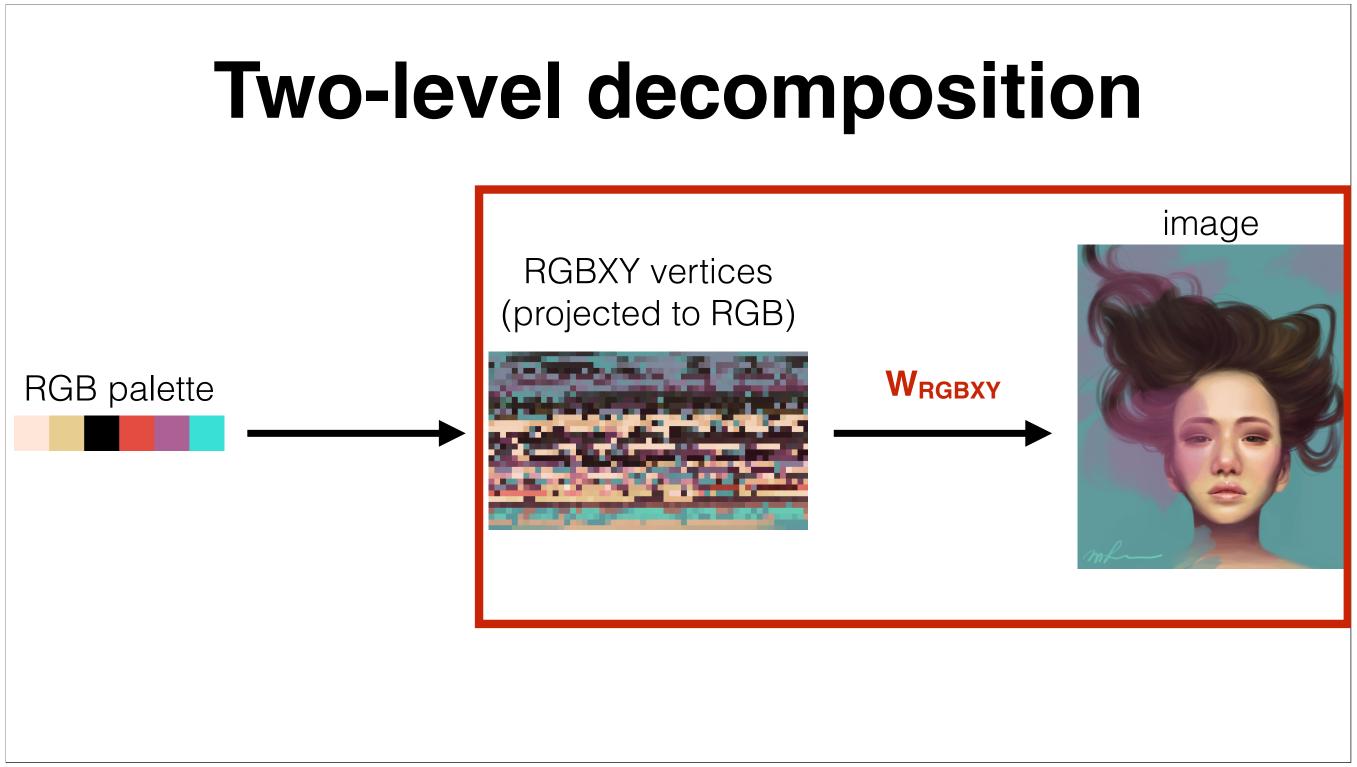
Two-level decomposition

Based on previous observation, we proposed a two level decomposition method to extract mixing weights from image given the palette.

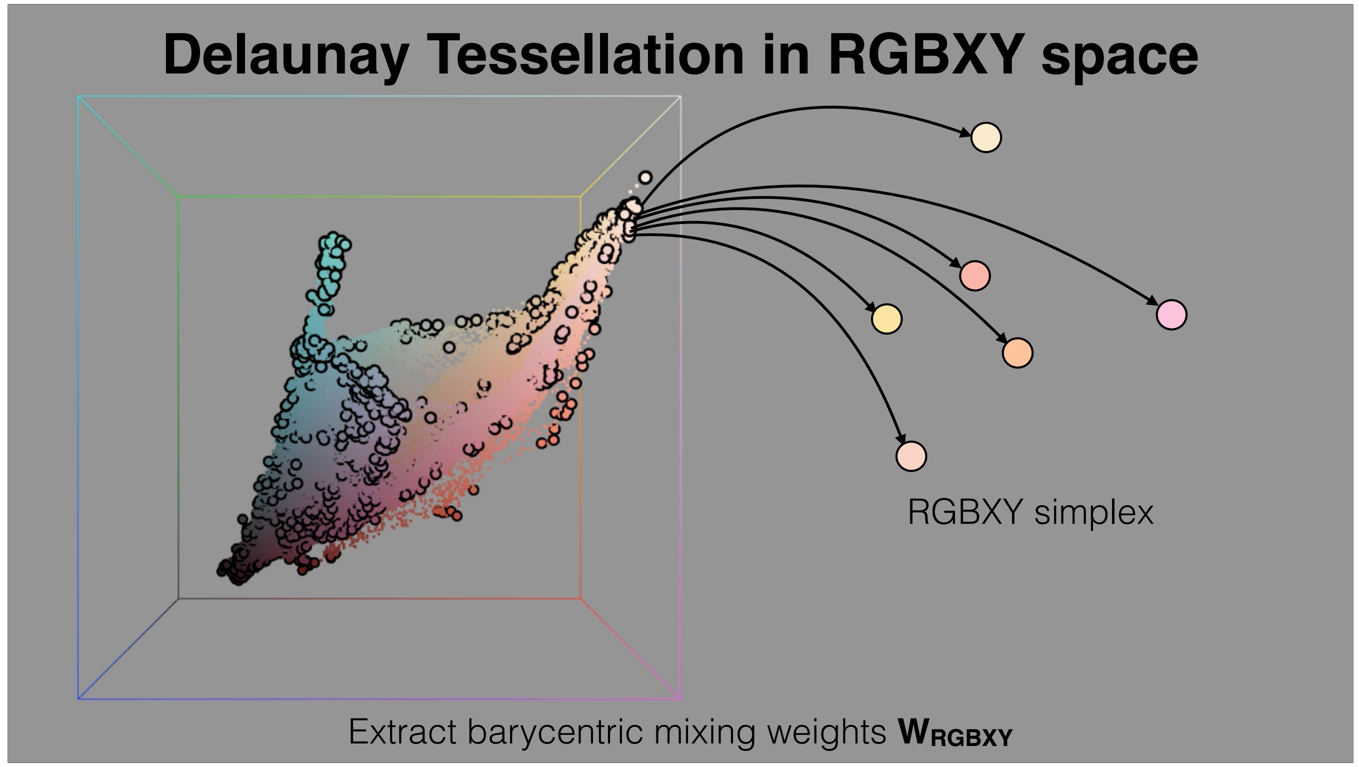
Two-level decomposition



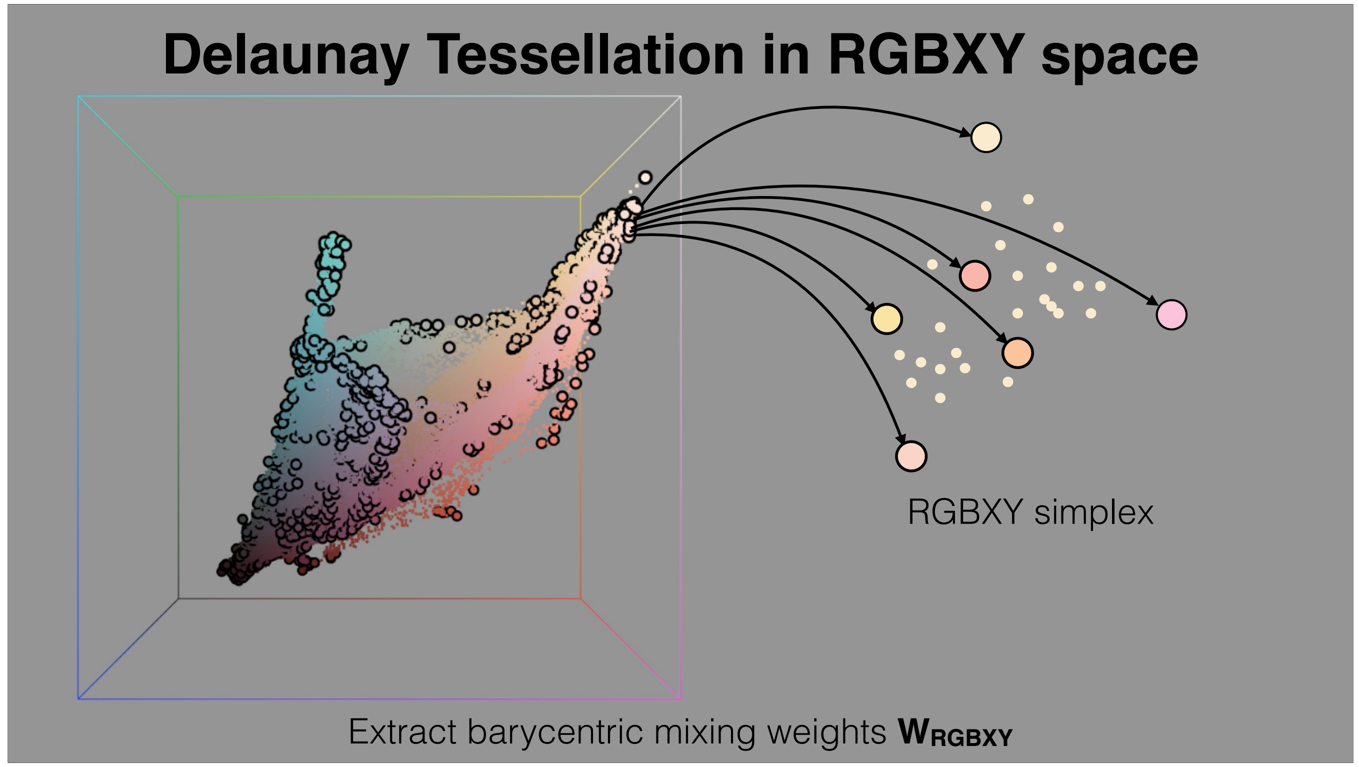
Based on previous observation, we proposed a two level decomposition method to extract mixing weights from image given the palette.



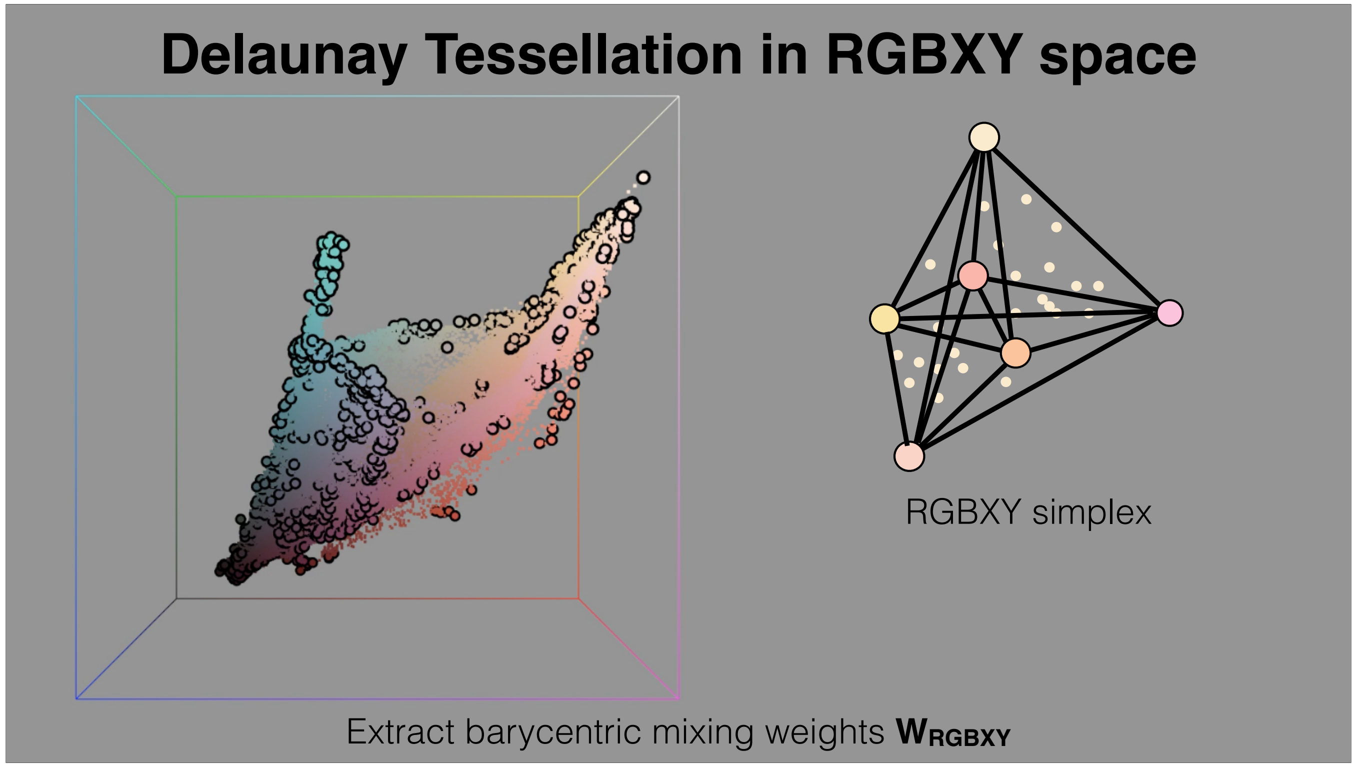
we first extract mixing weights W_RGBXY that mix RGBXY palette to reproduce whole image colors.



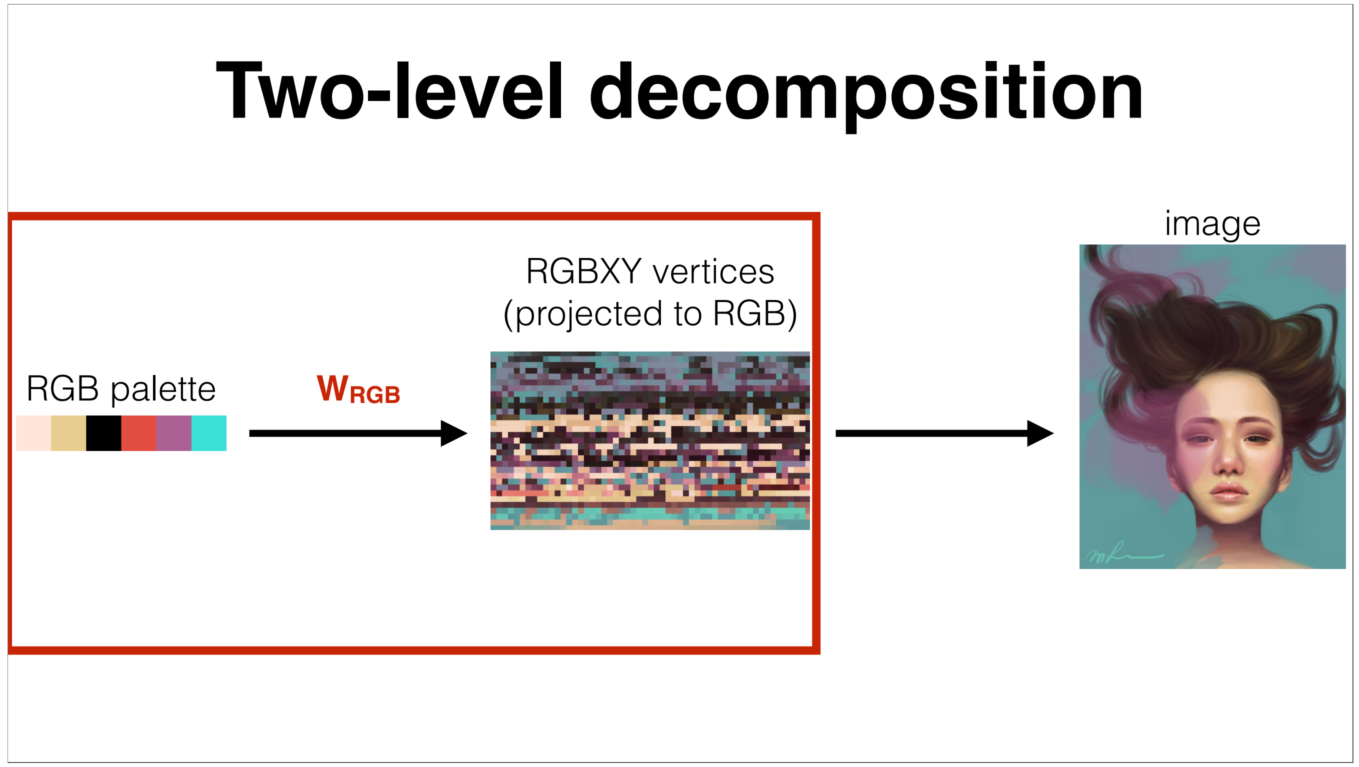
We first use delaunay tessellation to generate a set of simplices in RGBXY space.
Let’s consider one simplex in RGBXY space, here, 6 RGBXY vertices of one simplex is projected to RGB for visualization.



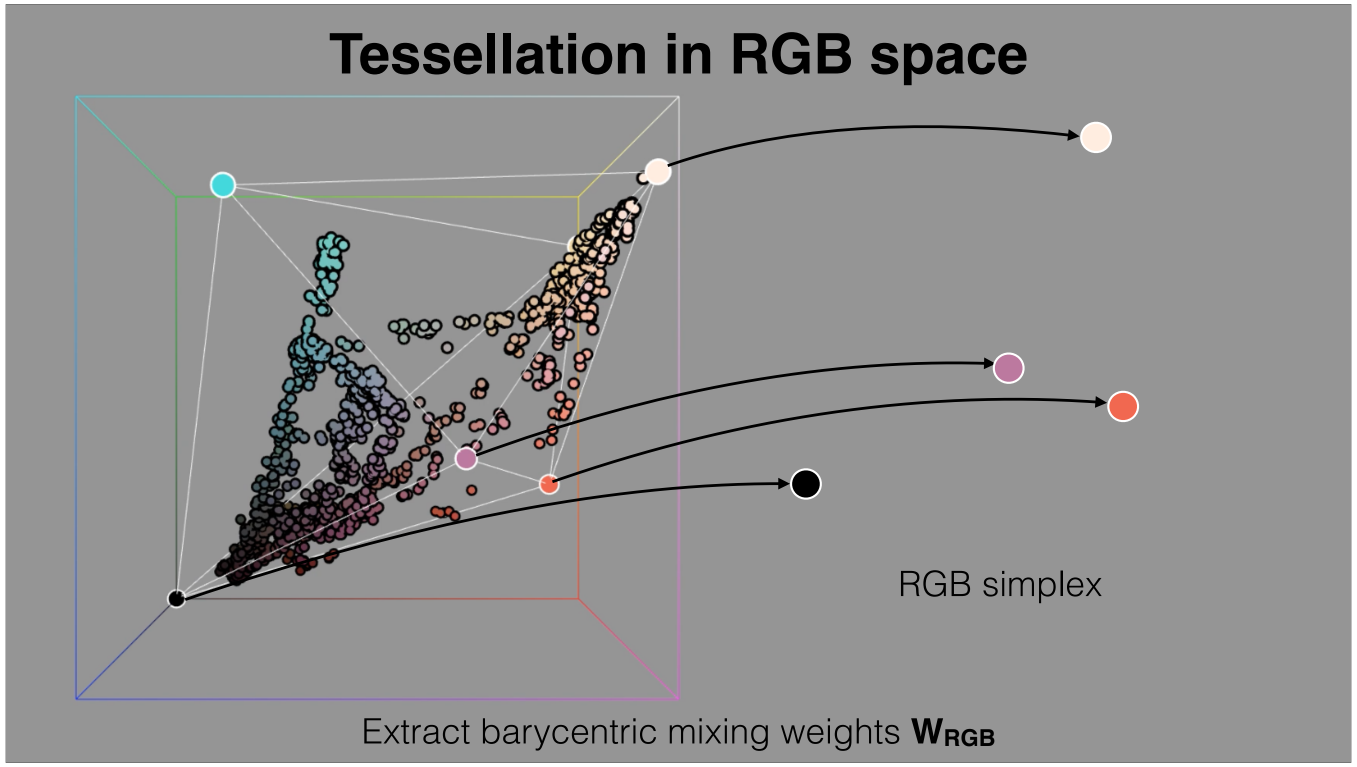
And these small dots are RGBXY points that are covered by this simplex.



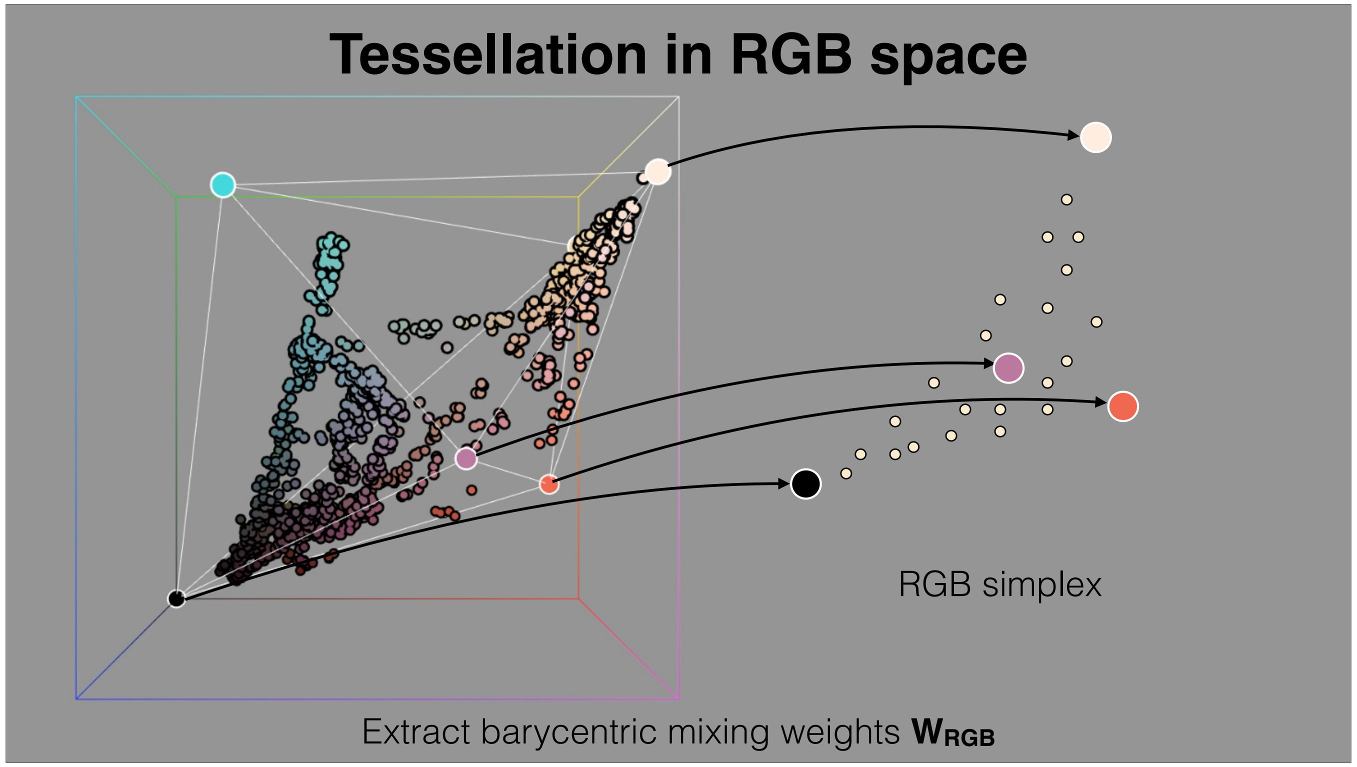
The barycentric coordinates of these inside RGBXY points in terms of this simplex is directly the mixing weights we want.



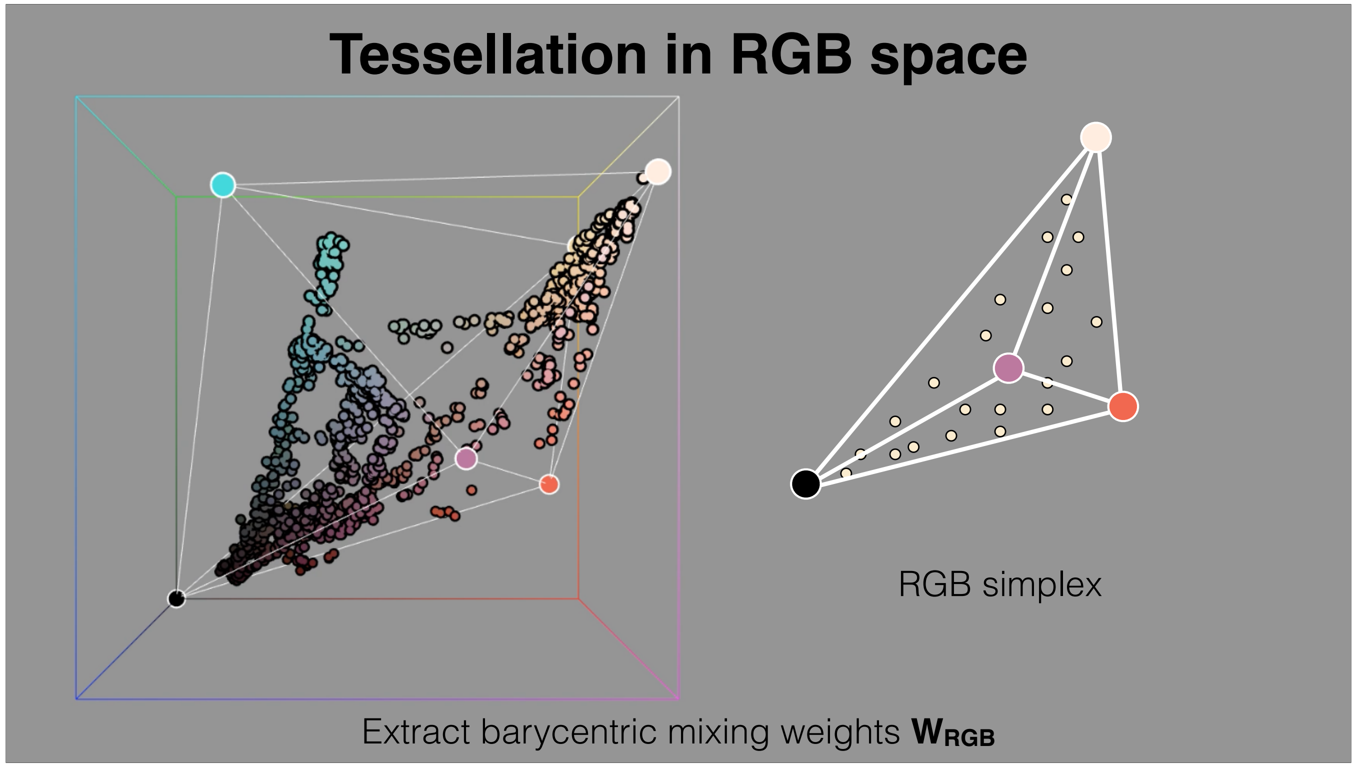
Second, we extract mixing weights W_RGB that mix RGB palette to reproduce those RGBXY palette colors.



similar to our previous process in RGBXY space, we first triangulate the RGB 3D space. And let's consider one simplex in RGB space.



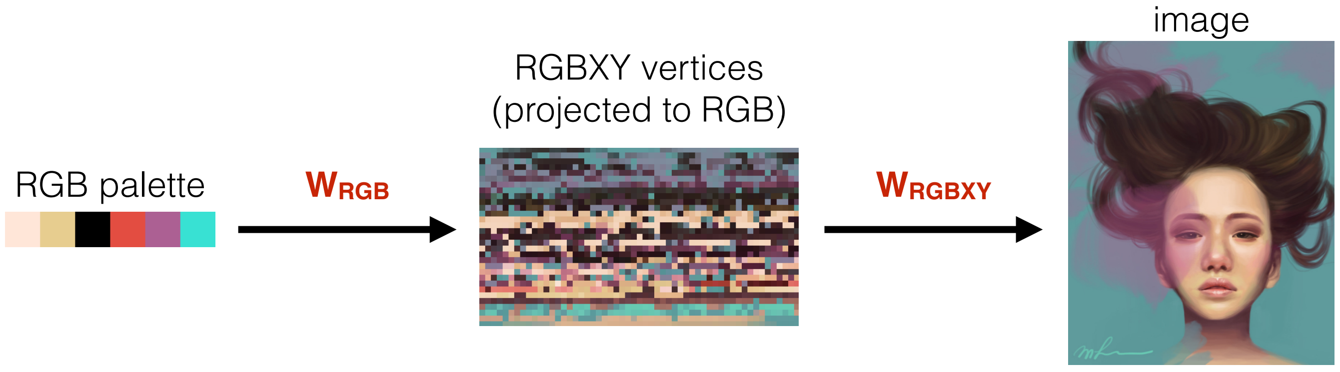
And these small dots are subset of RGB colors that are covered by this simplex.



The barycentric coordinates of these inside RGB points in terms of this simplex is directly the mixing weights we want.

As seen in the image, when these RGB points are the projected vertices from the RGBXY convex hull, we can see how we achieved a RGB parameterization of each pixel of the image, aware of the spatial relationships.

Two-level decomposition



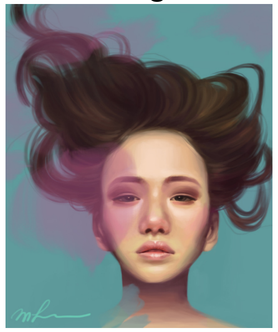
After we extract two level mixing weights. The final mixing weights W is just the multiplication of two matrices.

Two-level decomposition

RGB palette

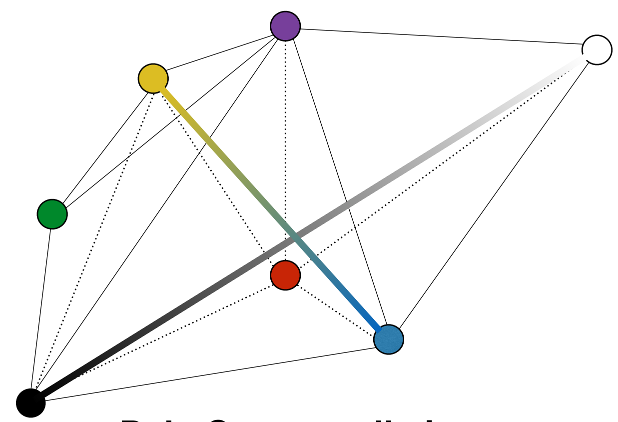
$$W = W_{RGB} * W_{RGBXY}$$

image



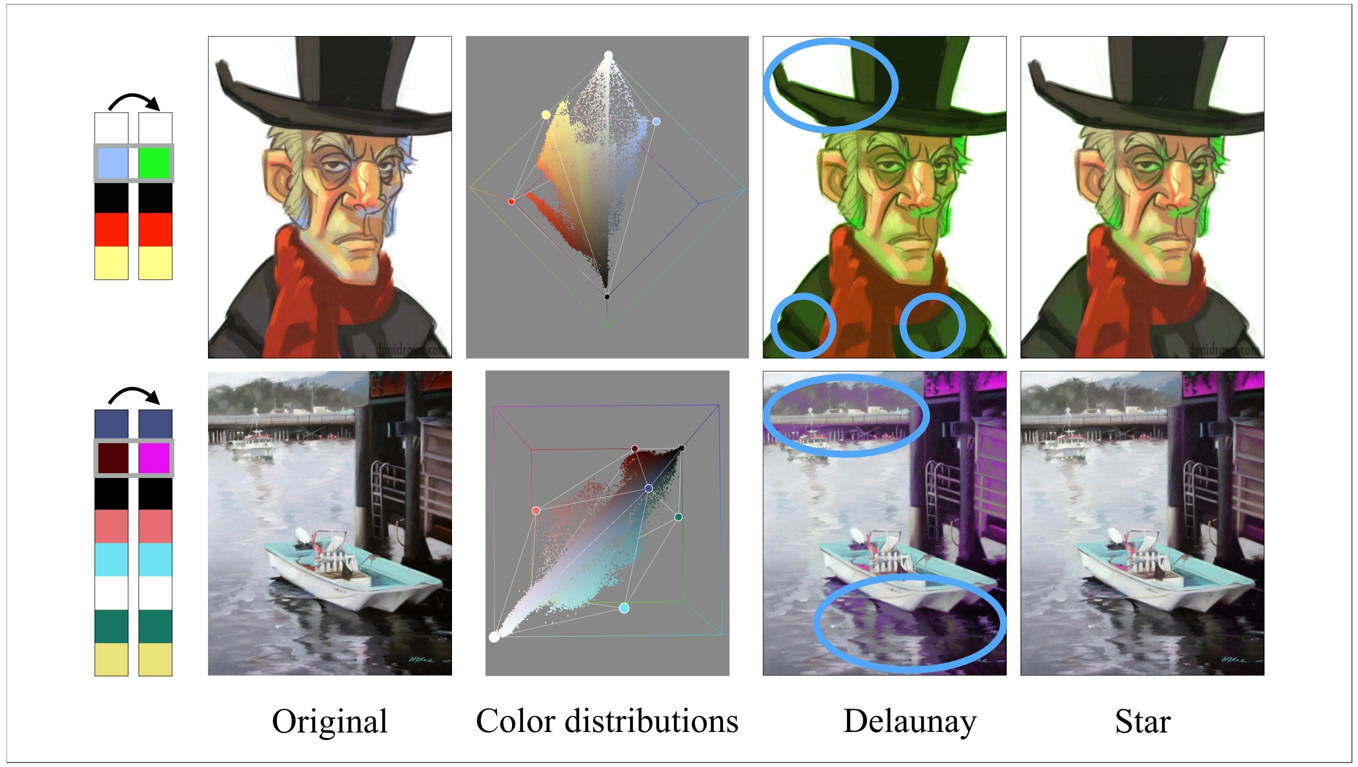
After we extract two level mixing weights. The final mixing weights W is just the multiplication of two matrices.

Tessellation in RGB space



Delaunay tessellation / **Star tessellation**

When decompose an RGB-space palette into simplices, the Delaunay tessellation creates simplices with favors short edges.
<click>
In this example, that creates an edge between yellow and blue, which are complementary colors. As a result, all grey pixels become the mixture of several colorful colors, rather than black and white. Recoloring any palette color would turn grey pixels “colorful”.
<click>
Instead, we explicitly add an edge along the “line of greys” between the darkest and lightest colors in the palette by performing a simple star tessellation.



recoloring results comparison between two different tessellations of RGB space.

If using delaunay tessellation, the gray pixel colors are mixed from several other palette colors. When user change one palette color like blue to green in first example, or dark red to pink in second example, both examples' gray colors regions are also recolored, which does not meet user's expectation.

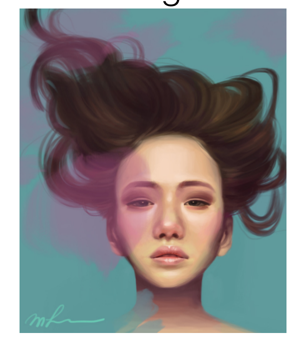
Our star triangulation will preserve these gray colors regions, if they are not necessarily to be changed.

Two-level decomposition

RGB palette

$$W = \overset{\text{Palette updates}}{W_{RGB}} * \overset{\text{Fixed}}{W_{RGBXY}}$$

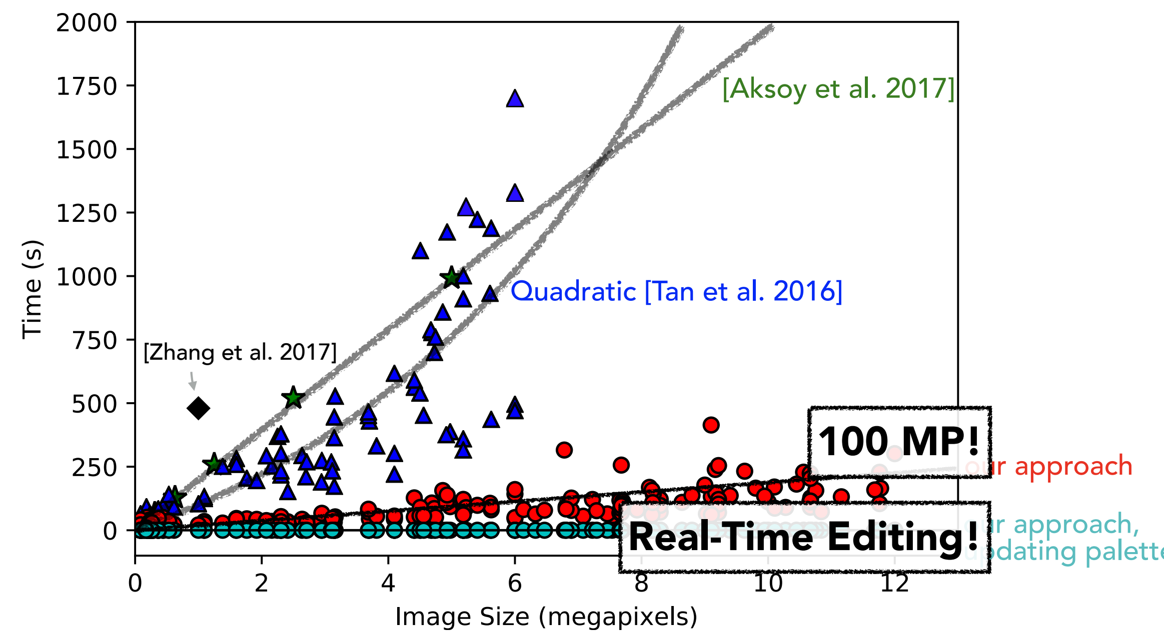
image



Updating W_{RGB} is independent of image size.
Other methods need to re-compute everything from scratch.

One advantage of our two level decomposition method is that when user edit palette, only W_rgb is influenced. Updating W_RGB only involves a few thousand pixels, the RGBXY convex hull vertices, which is independent of the image size. Thus, our layer updating is real time.
All other methods need to recompute from scratch if the palette changes.

Performance



Running time comparison between four additive mixing image decomposition algorithms.
Our algorithm's performance is orders of magnitude faster and scales extremely well with image size.
We evaluated our RGBXY algorithm on 170 images up to 12 megapixels.
They took a few minutes to compute from scratch, including palette selection.
Layer updating with an updated palette is nearly instantaneous (a few to tens of milliseconds).
We also tested an additional six 100 megapixel images (not shown; average running time 12.6 minutes).

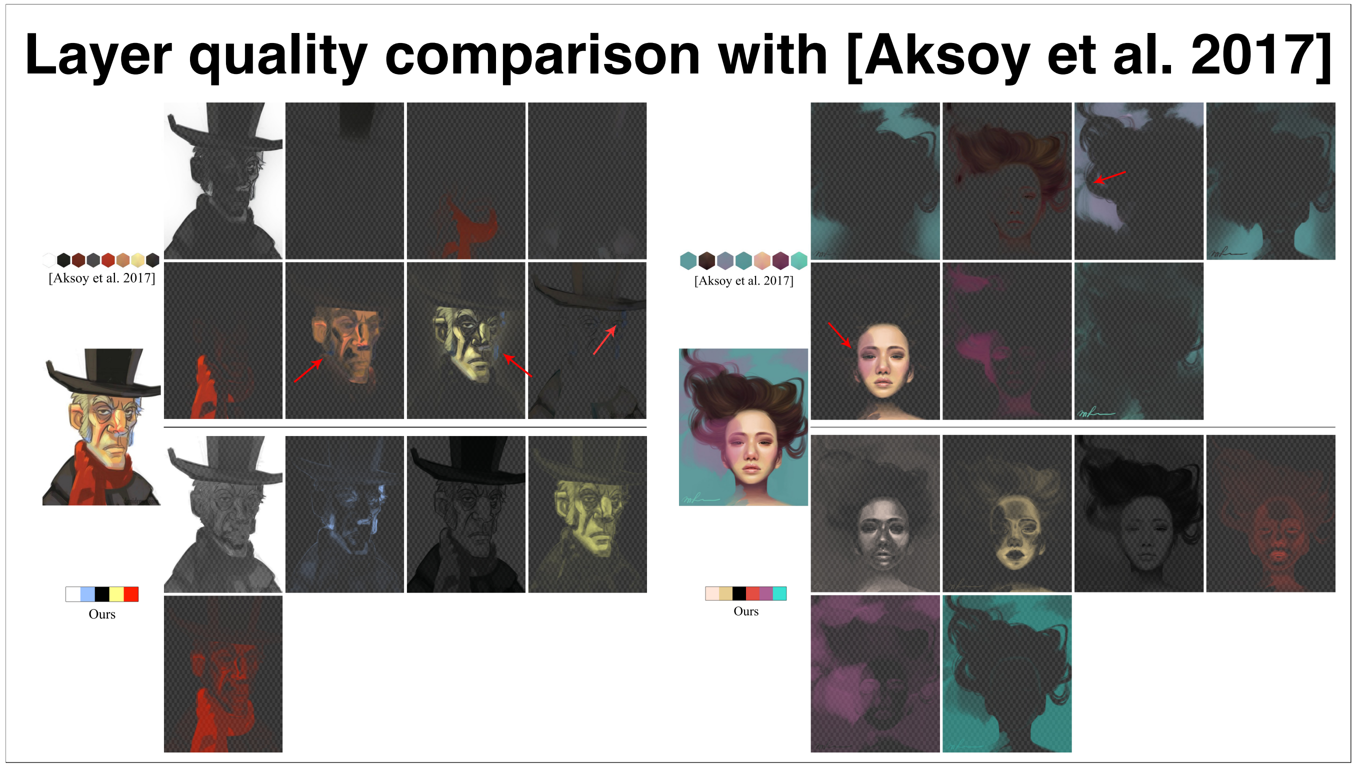
Python Implementation

48 lines of code

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

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

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

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

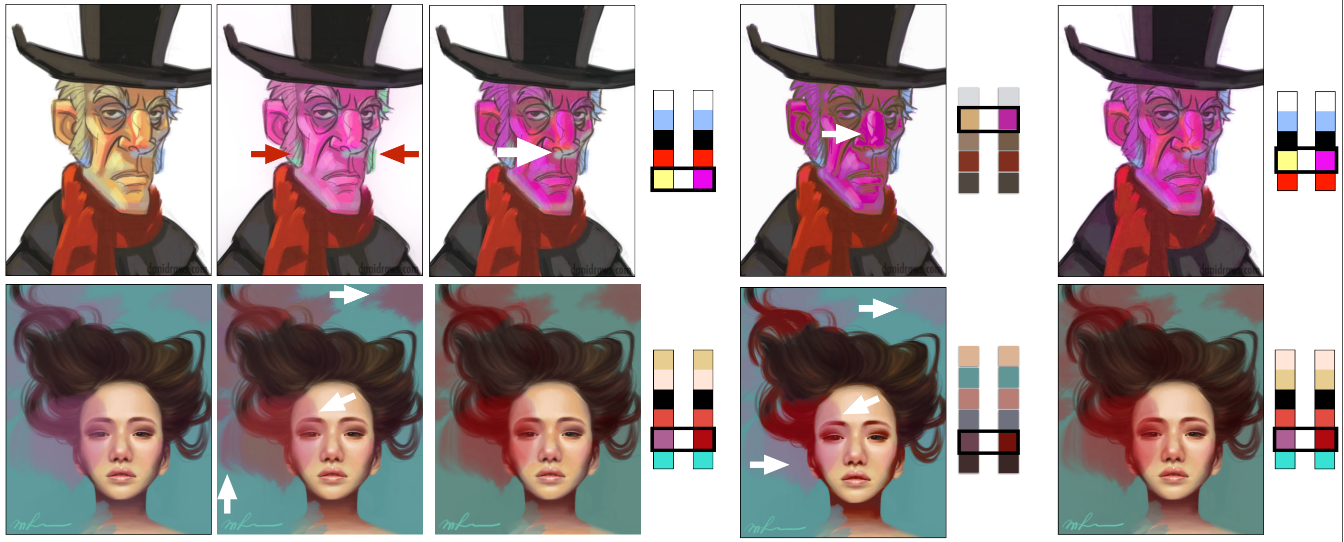
The python implementation of our core algorithm only has 48 lines of code, which is easy to understand and convenient to use.

Comparisons



We also compared our layers with Aksoy et al. 2017, which is state of art in additive mixing layers extraction.
Their each layer is composed of a color distribution, which may contains too many different colors in one layer.
In first example, the yellow face and blue beard appear together in one layer. And in second example, one part of purple haze is combined with face together.
Their colorful layers will make it not easy to edit the image.

Recoloring comparison with three previous methods



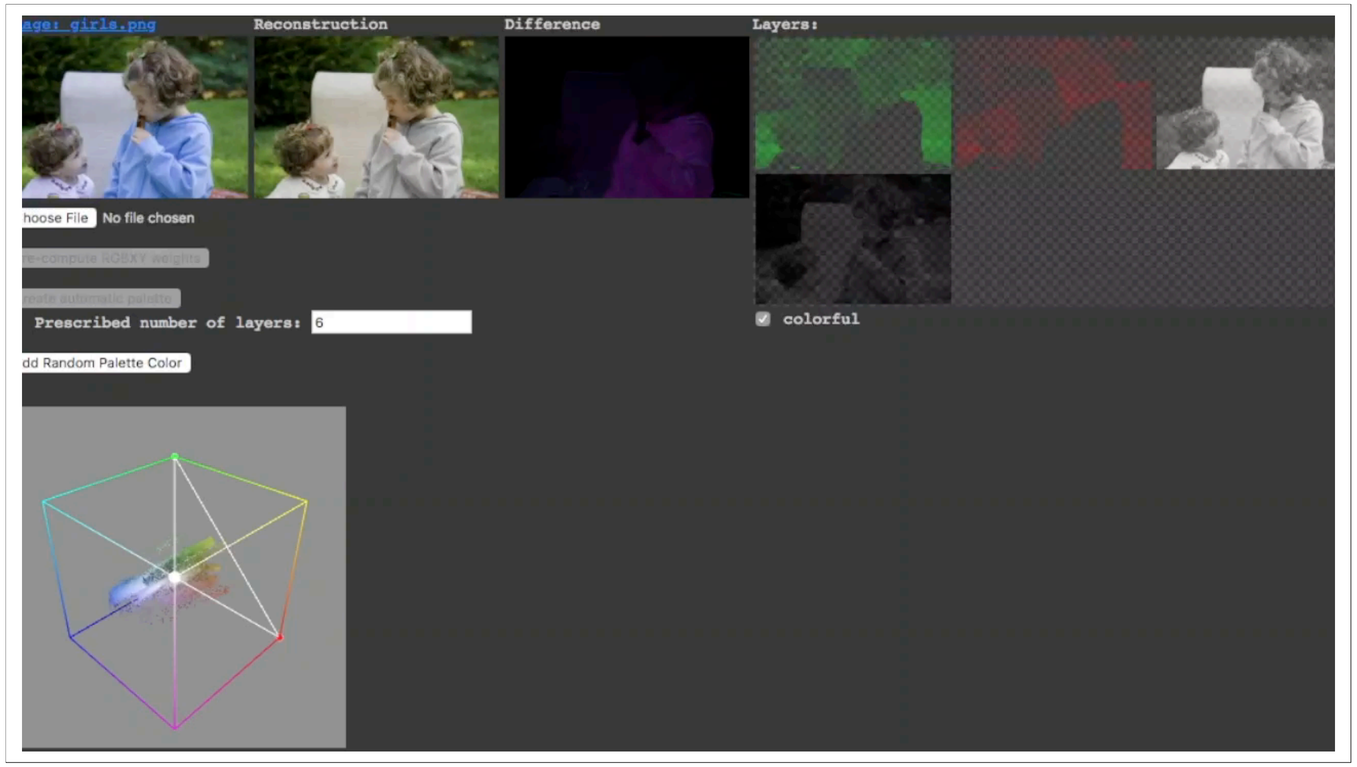
We use previous two examples to compare the recoloring results with previous three methods.
To compare fairly, we first recolor image using our method, and then other methods try to approach our recoloring results.
(click and click) first is aksoy 2017, as mentioned in previous slides, the beard in first example is changed with face color changing, and the purple haze on face is not changed in second example.
(click and click) second is Tan 2016. Their recoloring results also contains undesired color changes.
(click and click) third is Chang 2015. Their recoloring results usually have many artifacts.

Demo

Javascript + Python with PyOpenCL

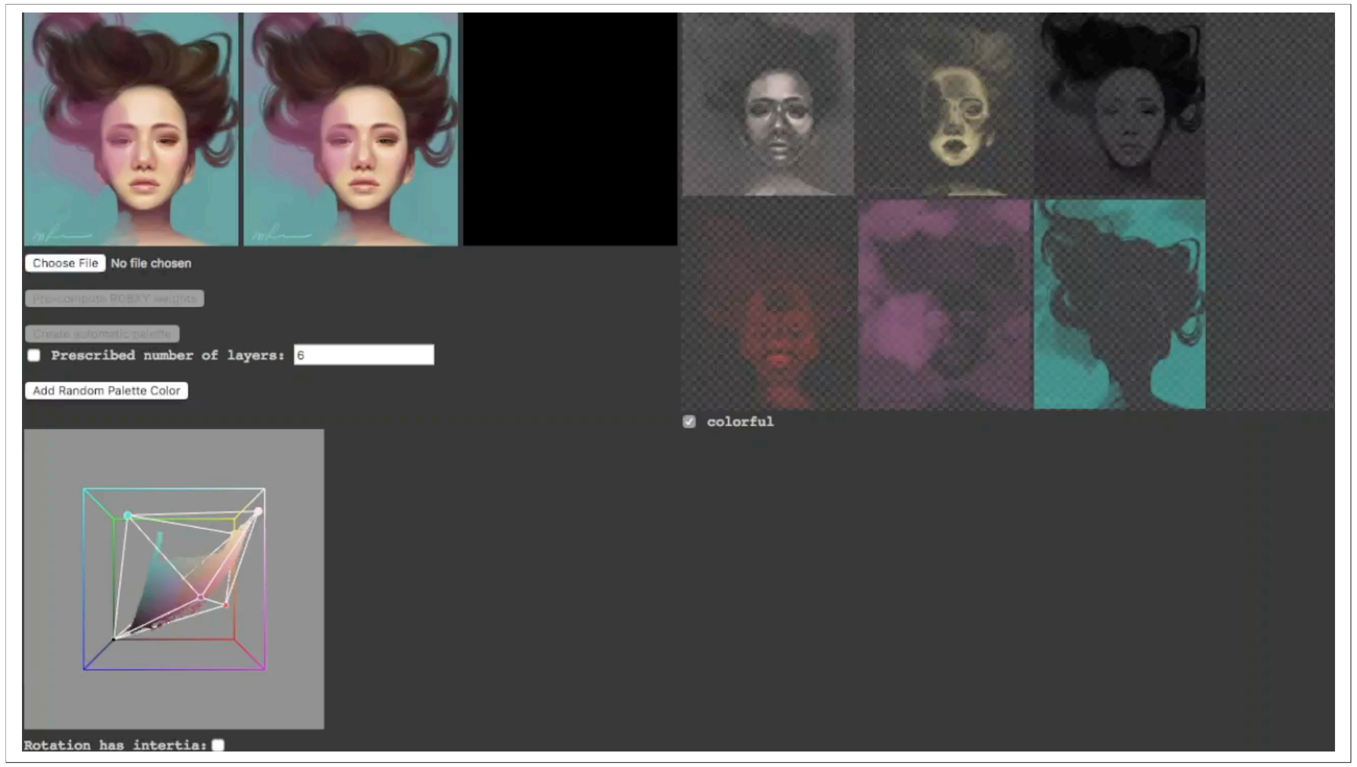
We build our GUI using python with pyopencl as backend.

Layer creation from scratch



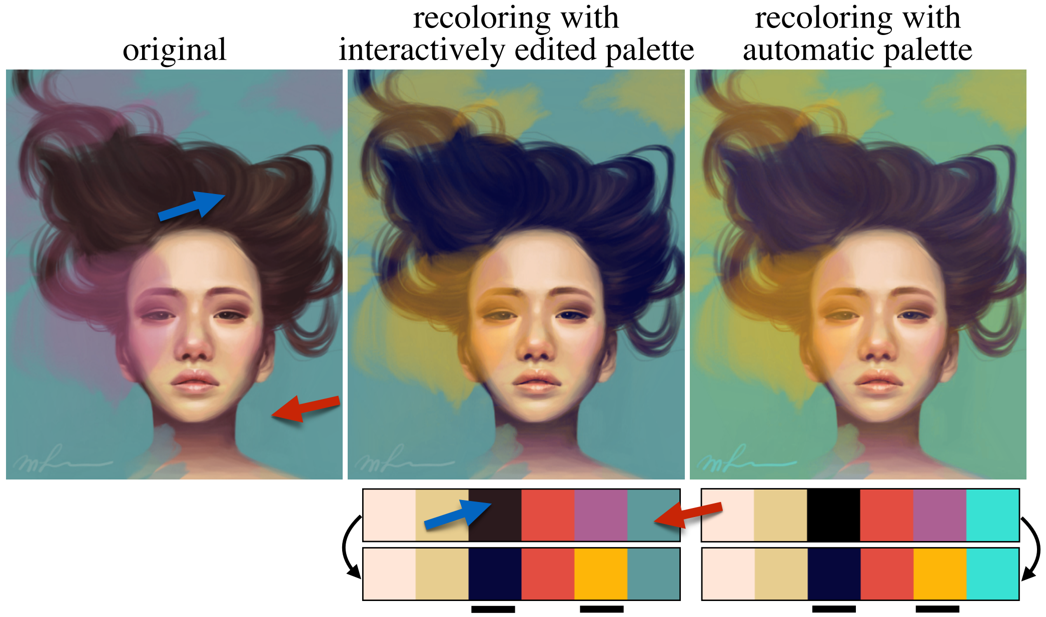
Our palette and layer editing GUI can let user start from a blank or dummy palette. The user can change palette colors or add new palette colors. After some rounds of edits, the palette and layers quality are quickly in good shape and can reconstruct image with very small error. The layer editing is real time.

Layer creation from an automatic palette



User can also use our automatically generated simplified hull vertices as palette to start editing. We can also improve the layer quality using this GUI.

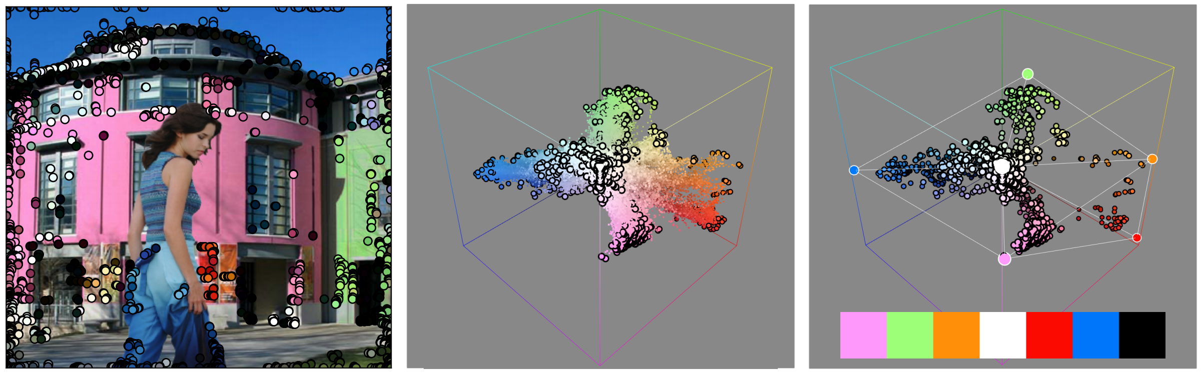
Interactive decomposition gives more control to the users



In this example, recolored image using automatic palette is shown in right column, background and hair colors are influenced dramatically, because of the non-sparsity layers.
The automatic palette becomes sparser as a result of interactive editing in previous slides. The user edits the automatically generated palette to ensure that the background and hair colors are directly represented, as shown in middle column.
Click and Click
As a result, editing the purple haze and hair no longer affects the background color.

Conclusion

- An extremely efficient approach to layer decomposition via RGBXY geometry



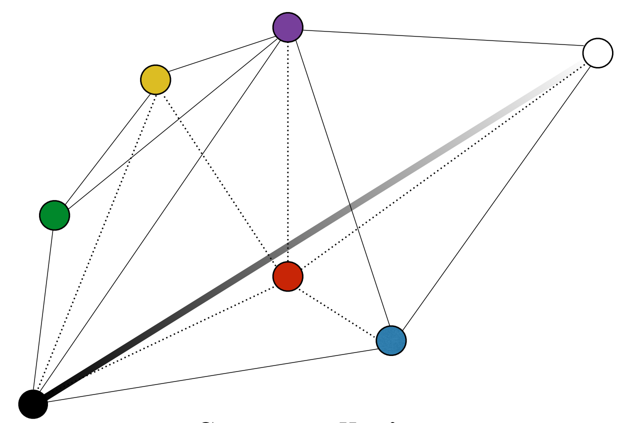
Conclusion

- Our two-level decomposition supports real-time decomposition when palette editing.

$$\underbrace{W}_{} = \overset{\text{Palette updates}}{W_{RGB}} * \overset{\text{Fixed}}{W_{RGBXY}}$$

Conclusion

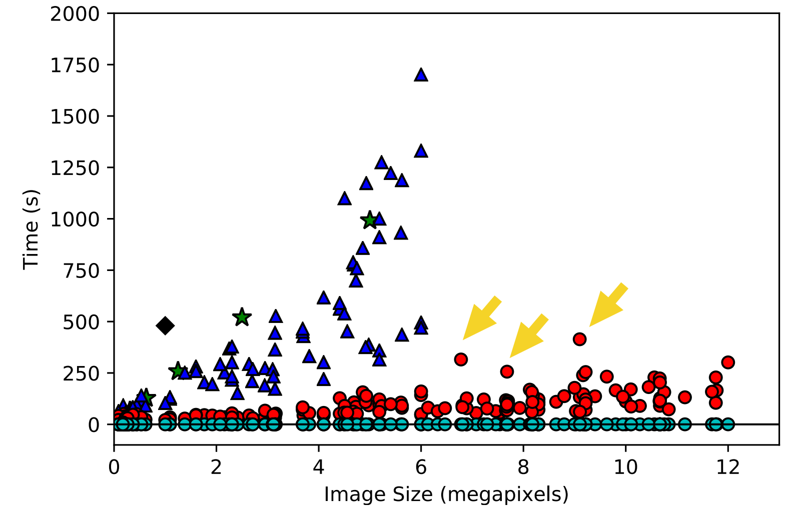
- It’s important to capture the “line of greys”.



Star tessellation

Limitations

- In isolated cases, the 5D convex hull takes somewhat longer than usual to compute.



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

>200s cases:

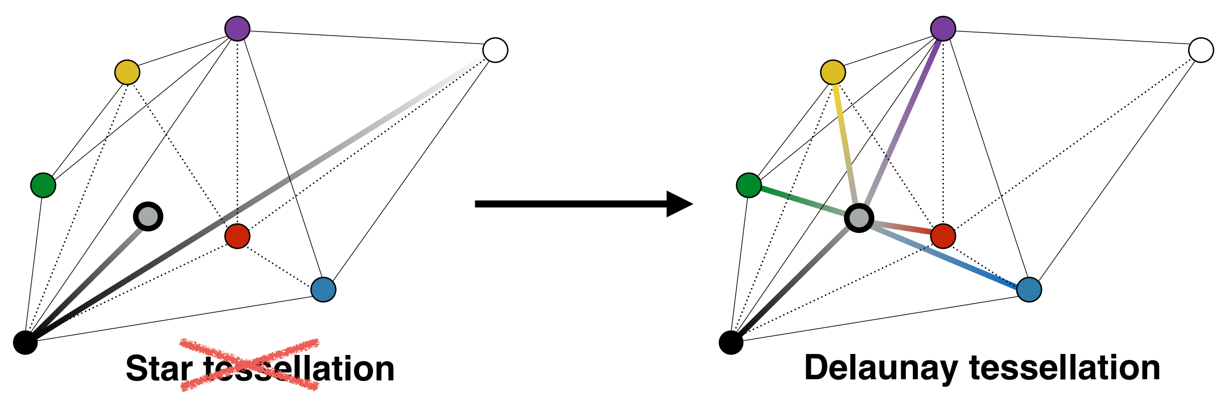
from left to right: image size(MP), palette size, RGBXY hull vertices numbers, and whole pipeline time(s)

[6.78 9. 3611. 315.557657]
[7.68 7. 3089. 256.348872]
[9.1 6. 2822. 414.002905]
[9.1649 8. 3308. 239.159933]
[9.216 7. 3626. 254.504188]
[9.6292 8. 4199. 232.211461]
[10.56 7. 3374. 228.450447]
[10.664 6. 3041. 223.498411]
[10.668 6. 2788. 203.945095]
[11.76 8. 2734. 228.173037]
[12. 9. 4148. 301.418308]

Palette sizes:

Limitations

- Our star tessellation assumes that palette colors are vertices of a convex polyhedron.
    - For palette colors in the interior, must use inferior Delaunay tessellation.



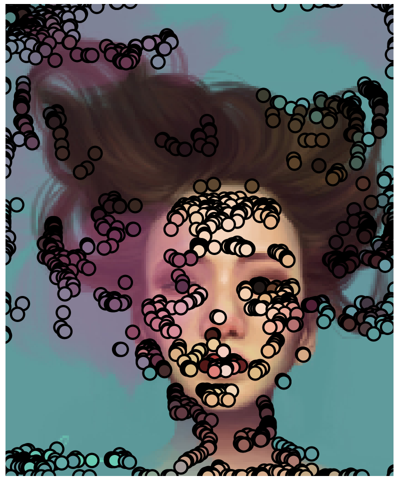
Our star tessellation assumes that the palette colors are vertices of a convex polyhedron. In particular, it cannot be used when some palette colors lie in the interior of the convex hull. When this happens, one could fall back on a Delaunay tessellation, which may not create an edge along the line of grays.

######

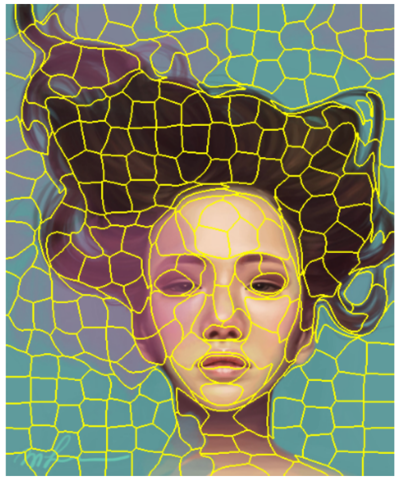
The line of grays could be maintained with a constrained tetrahedralization algorithm, though these are complex and may add new, undesired vertices [Yang et al. 2005].

Future Work

- More speed via super-pixels or parallel convex hull algorithms.



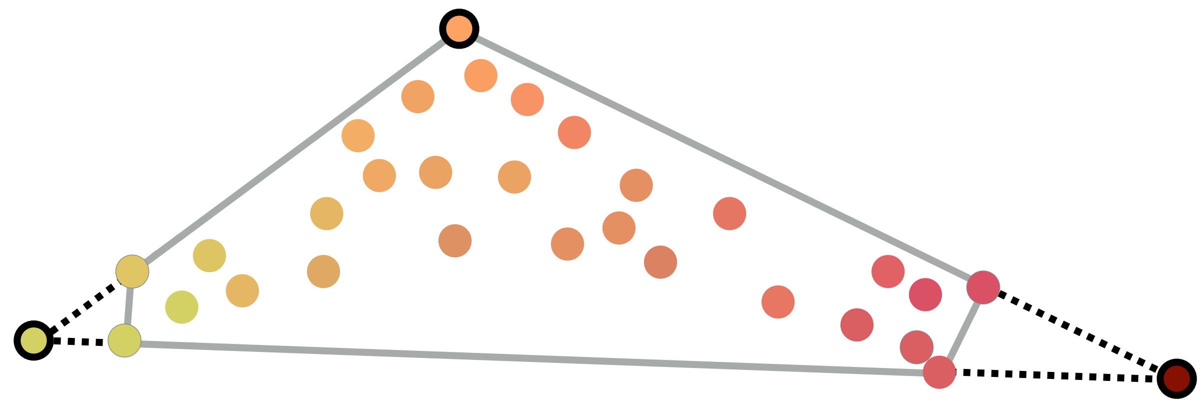
+



Inspired by Lin et al. [2017b], we wish to explore the use of superpixels to see if we are able to achieve greater speed ups. Or we can explore parallel convex hull algorithms to speed up.

Future Work

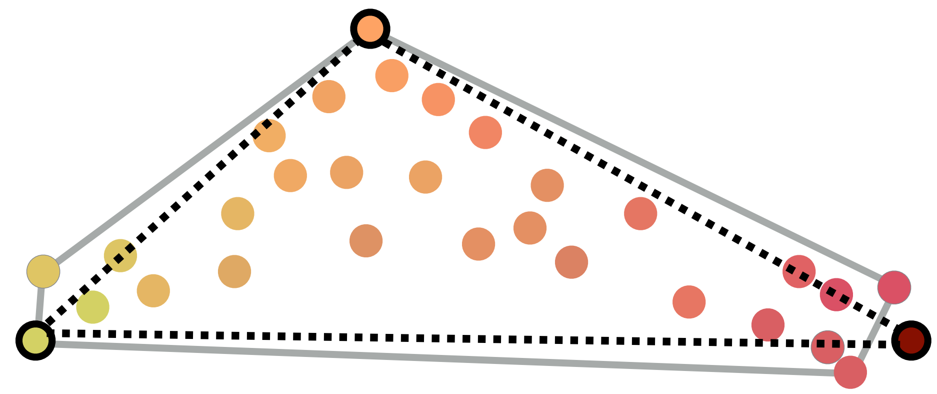
- Robustness via approximate convex hull algorithms.



We also wish to explore approximate convex hull algorithms. An algorithm that produces a smaller, approximate convex hull containing only a certain percentile of points could provide an intuitive parameter for balancing reconstruction error with sparsity.

Future Work

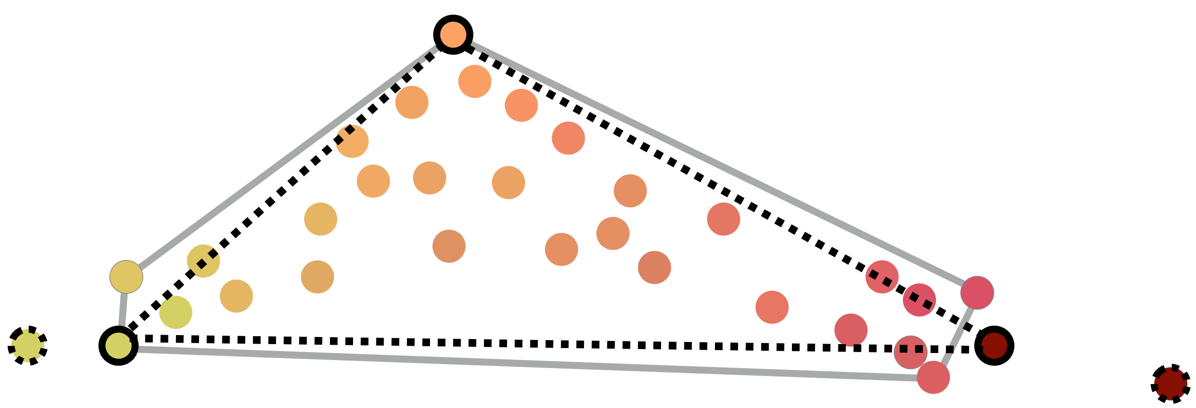
- Robustness via approximate convex hull algorithms.



We also wish to explore approximate convex hull algorithms. An algorithm that produces a smaller, approximate convex hull containing only a certain percentile of points could provide an intuitive parameter for balancing reconstruction error with sparsity.

Future Work

- Robustness via approximate convex hull algorithms.



We also wish to explore approximate convex hull algorithms. An algorithm that produces a smaller, approximate convex hull containing only a certain percentile of points could provide an intuitive parameter for balancing reconstruction error with sparsity.

Thank You!

- Contact Information:
  - Jianchao Tan: jtan8@gmu.edu
  - Jose Echevarria: echevarr@adobe.com
  - Yotam Gingold: ygingold@gmu.edu
- Project Website (GUI, code, data): https://cragl.cs.gmu.edu/fastlayers/
- Artists: Adelle Chudleigh; Dani Jones; Karl Northfell; Michelle Lee; Adam Saltsman; Yotam Gingold; DeviantArt user Sylar113; Fabio Bozzone; Piper Thibodeau; Spencer Nugent; George Dolgikh; DeviantArt user Ranivius.
- Sponsors:
  - NSF, Adobe, Google.

Thank you for your coming. Here is our contact information and project website.(If you have any suggestions or questions, please contact us.) Thanks for these artists providing great painting materials for us, and thanks for these generous sponsors. Thank you!

Extra slides

Possible questions

- Star triangulation starting from black palette color, what if no black color in extracted palette?
- Does your method require palette to cover all pixel colors when editing palette in GUI? What if I want some palette colors that is inside color point cloud?
- In your performance figure, there are one or two cases that are slower than many others. Can you describe the worst case performance of your method?
- Do you have failure case?
- How do you measure the quality of your layer results and your interactive editing GUI?

1. We use palette color that is closest to black color to be star point. Our extracted geometric convex palette usually contain one palette color that is close to black color for most images.
2. When editing the palette in GUI, user need to try to make the palette convex hull cover all pixel colors. If some pixels colors are outside the palette convex hull, they will be reconstructed with some errors. There is a trade off between reconstruction errors and layer sparsity. If user require an interior palette color, then we can easy to fall back to delaunay tessellation. Also, a compromise method is to use a constrained delaunay tessellation, which will also maintain the black-to-white edge.
3. One bottleneck step of our two level decomposition method is to compute convex hull in 5D space for all image pixels. The complexity is related with both image pixel numbers and output RGBXY convex hull vertices number. The convex hull vertices numbers can be pretty large for some bad RGBXY points distribution. The large number of 5D convex hull vertices will also influence the speed of 5D delaunay tessellation on hull vertices in later step. That's why some images' performance is not that good. But this is not very often. We only find 3 cases from170 tested images. And this case may be improved by using an approximate convex hull method in the future.
4. In terms of image reconstruction error. Our method never fail, because theoretically we can get very low errors, according to user defined error tolerance. In terms of layer quality, we usually get easy to edit layers from given image. We think we do not have failure case, but sometimes, our layers may not be that sparse to meet user's editing requirement. Theoretically, our layer sparsity is lower than RGB method, which is noisy although. However, our GUI can let user to improve the layers quality easily.
5. We measure layer quality in terms of smoothness, sparsity, and recoloring results quality. We showed several comparisons in our previous slides and we have more comparison in our paper. Our GUI can support real time editing of layers, the user can easily see the reconstruction errors and layer sparsities in real time to judge the quality. However, the best evaluation of layer quality is to recolor the image using these edited layers. We have a real time recoloring GUI, which can also be merged into current GUI in the future. We have a survey to some novice or expert image editing persons. They think our GUI is useful in image editing tasks. And they hope to see recoloring GUI and layer editing GUI merge together.