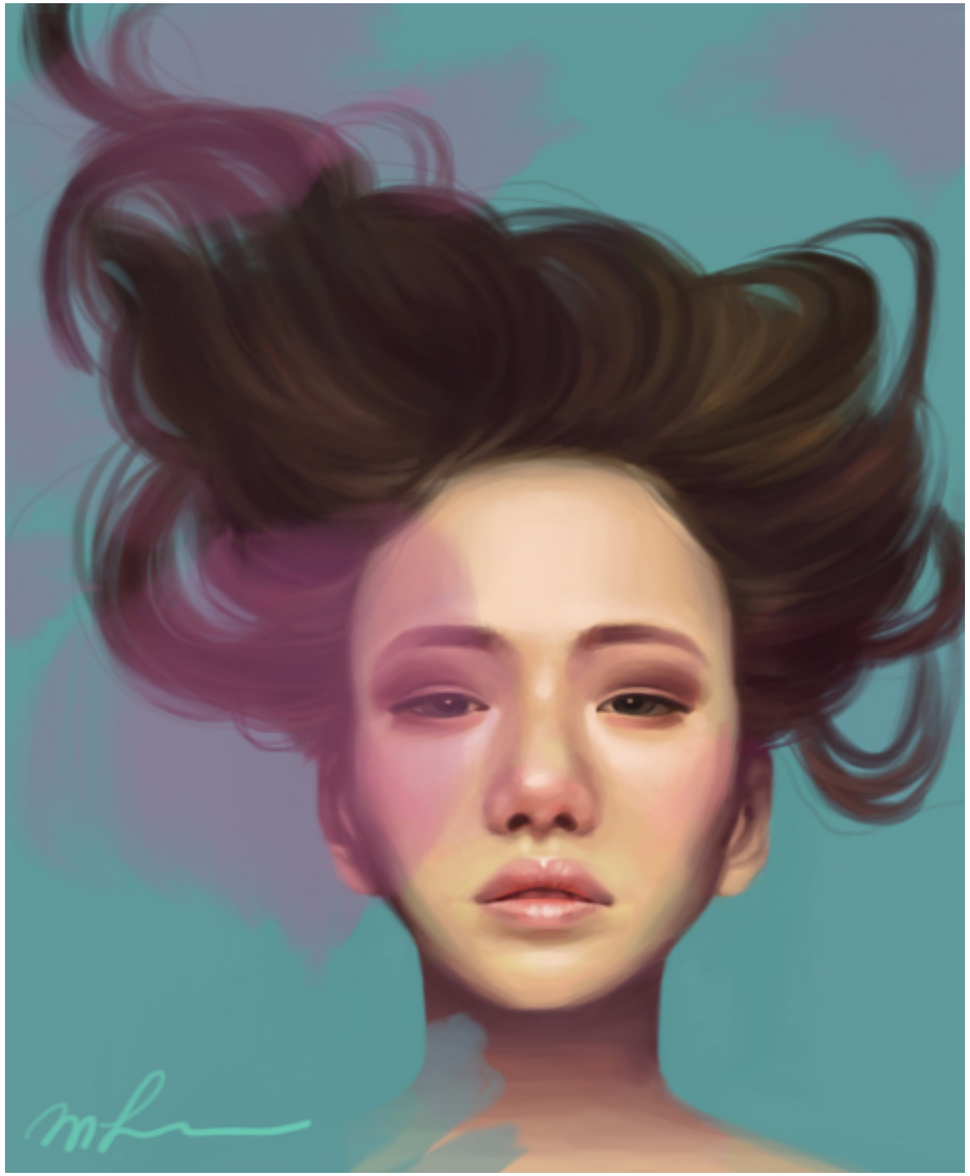
Efficient palette-based decomposition and recoloring of images via RGBXY-space geometry

Jianchao Tan George Mason University

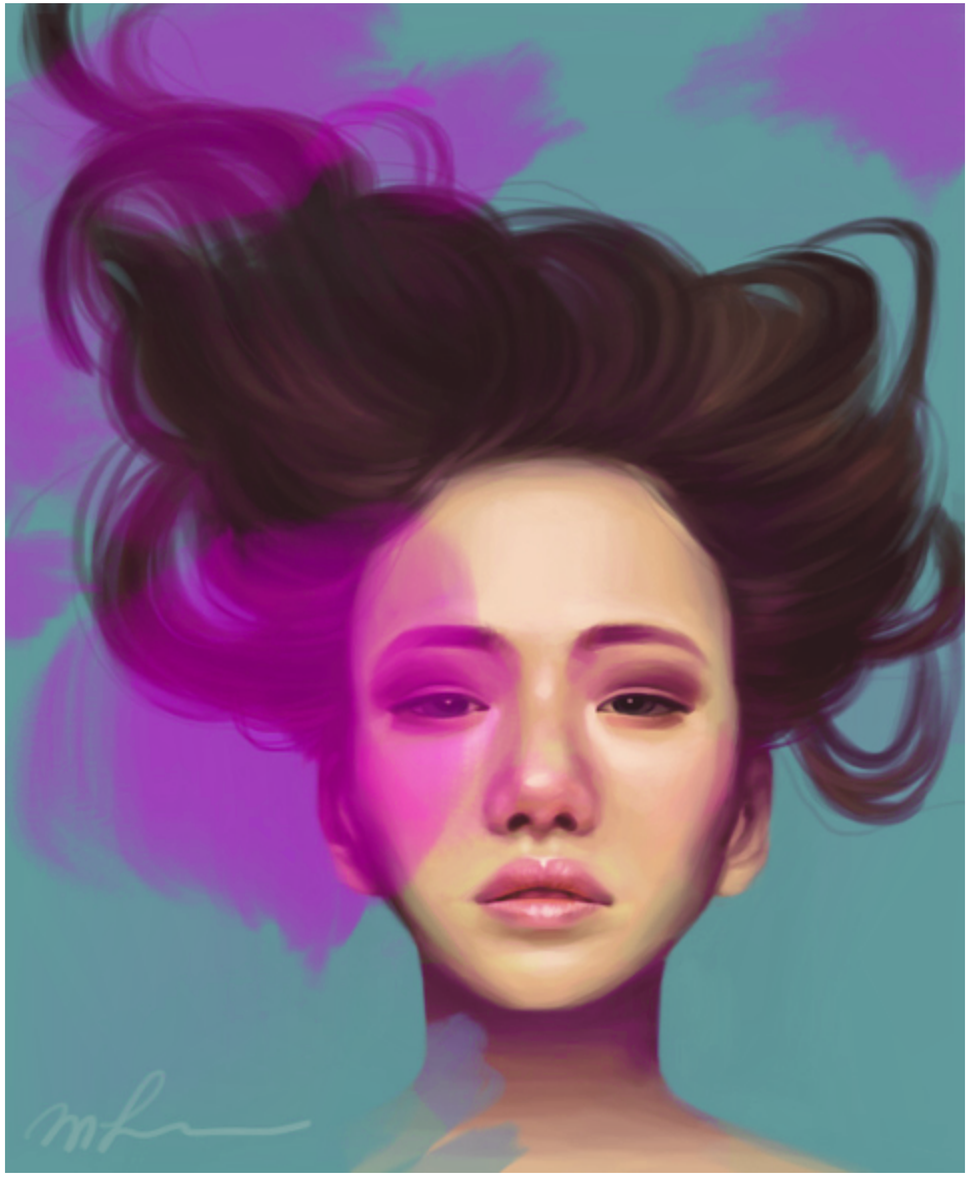
Jose Echevarria Adobe Research

Yotam Gingold George Mason University

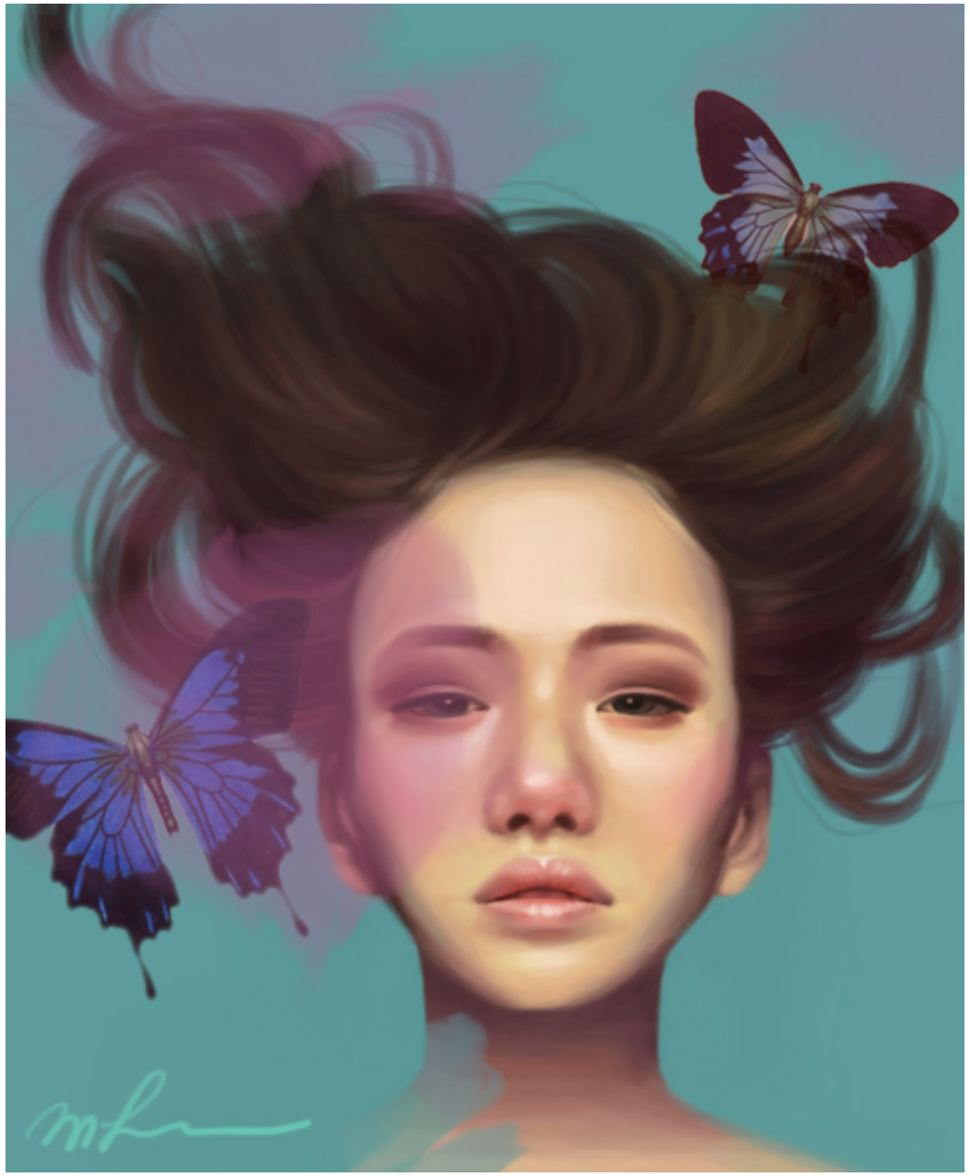
Motivation: Layers Organize Images



Motivation: Layers Organize Images

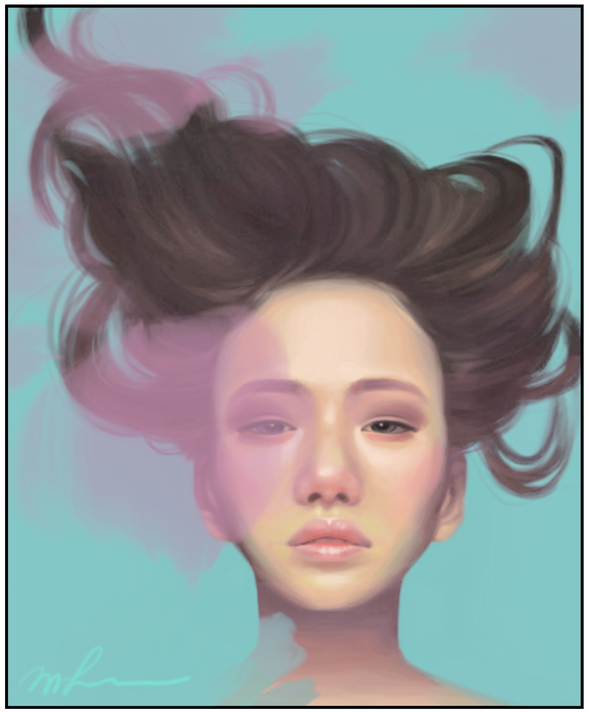


Motivation: Layers Organize Images



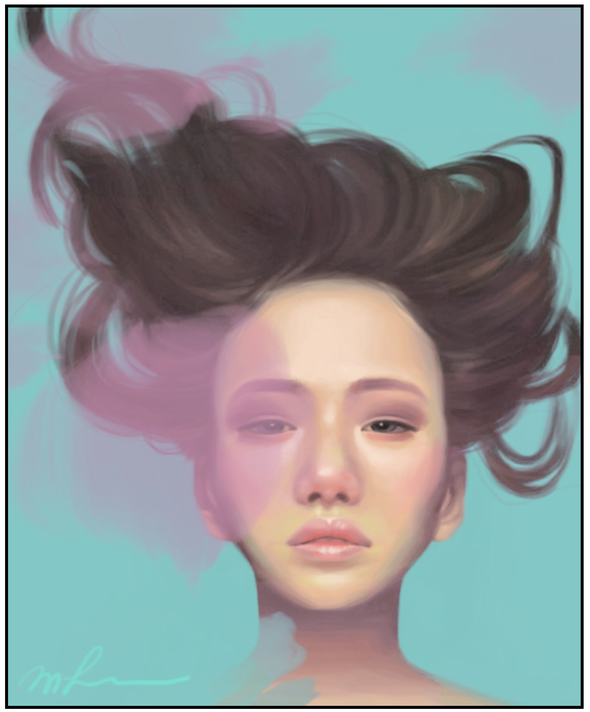
Goal

Input

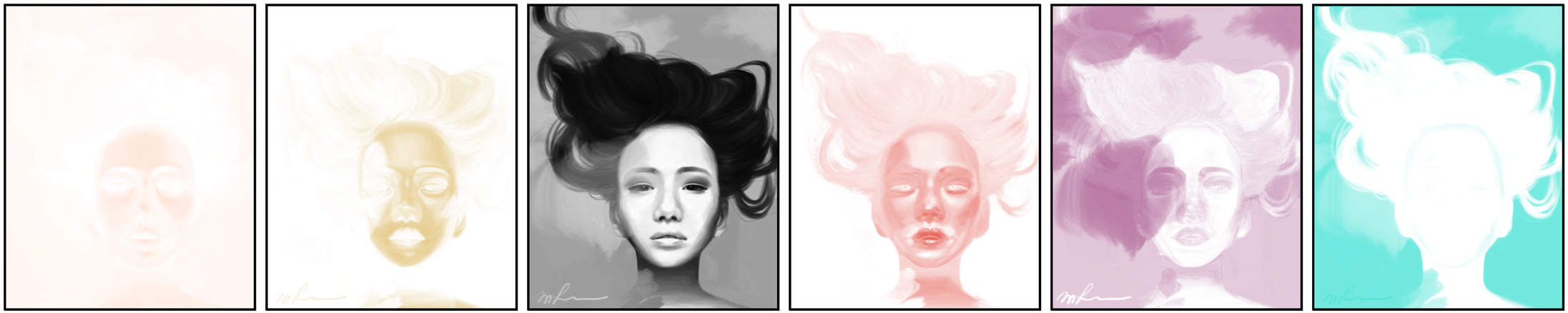


Goal

Input

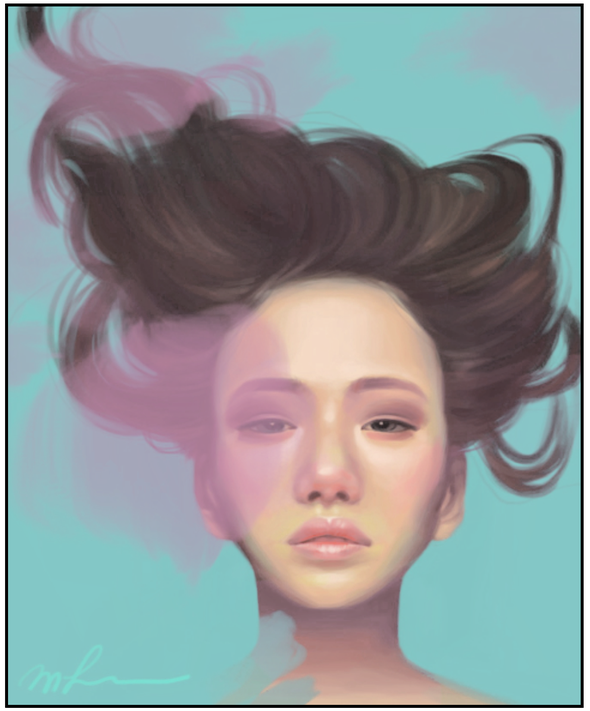


Layers

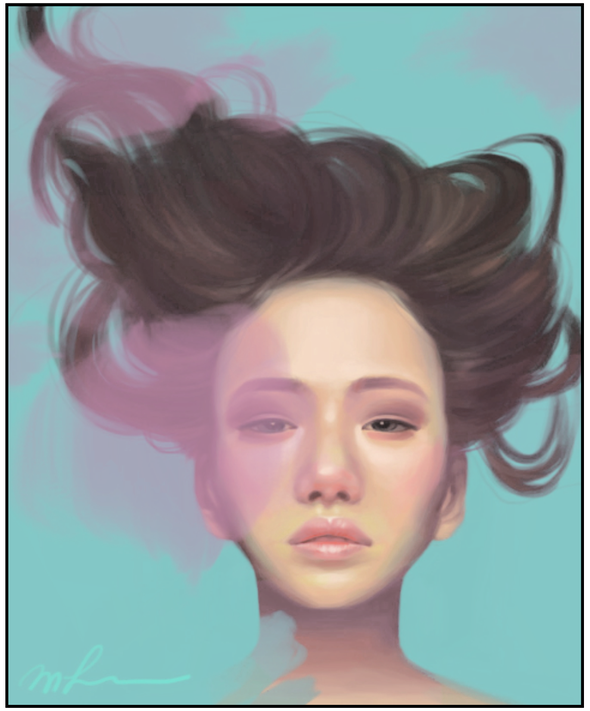


Goal

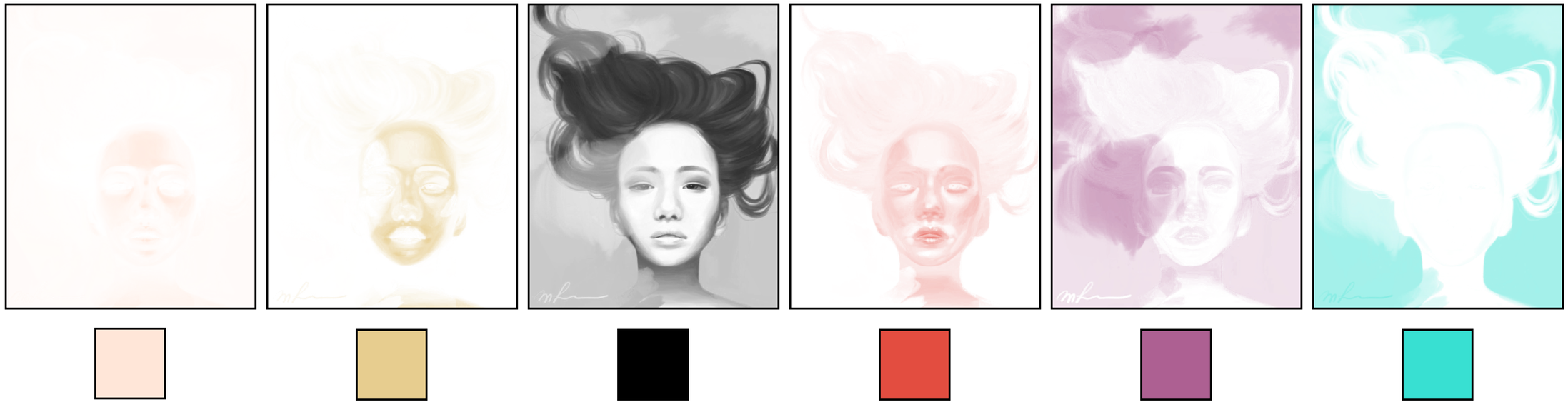
Input

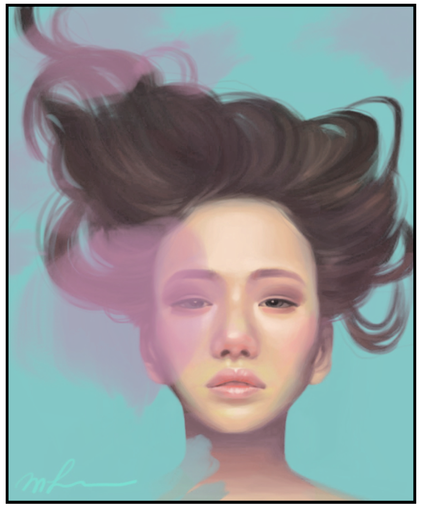


Reconstruction



Layers

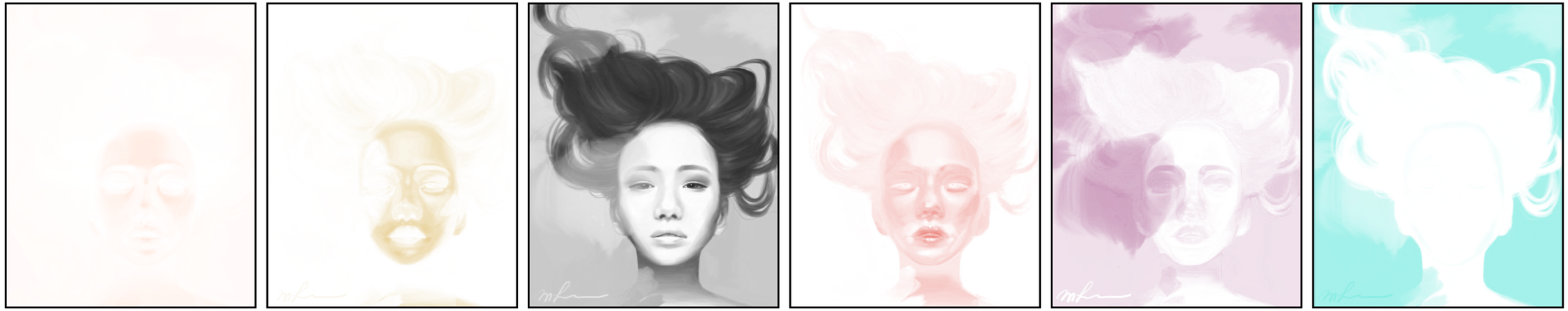




Subproblems

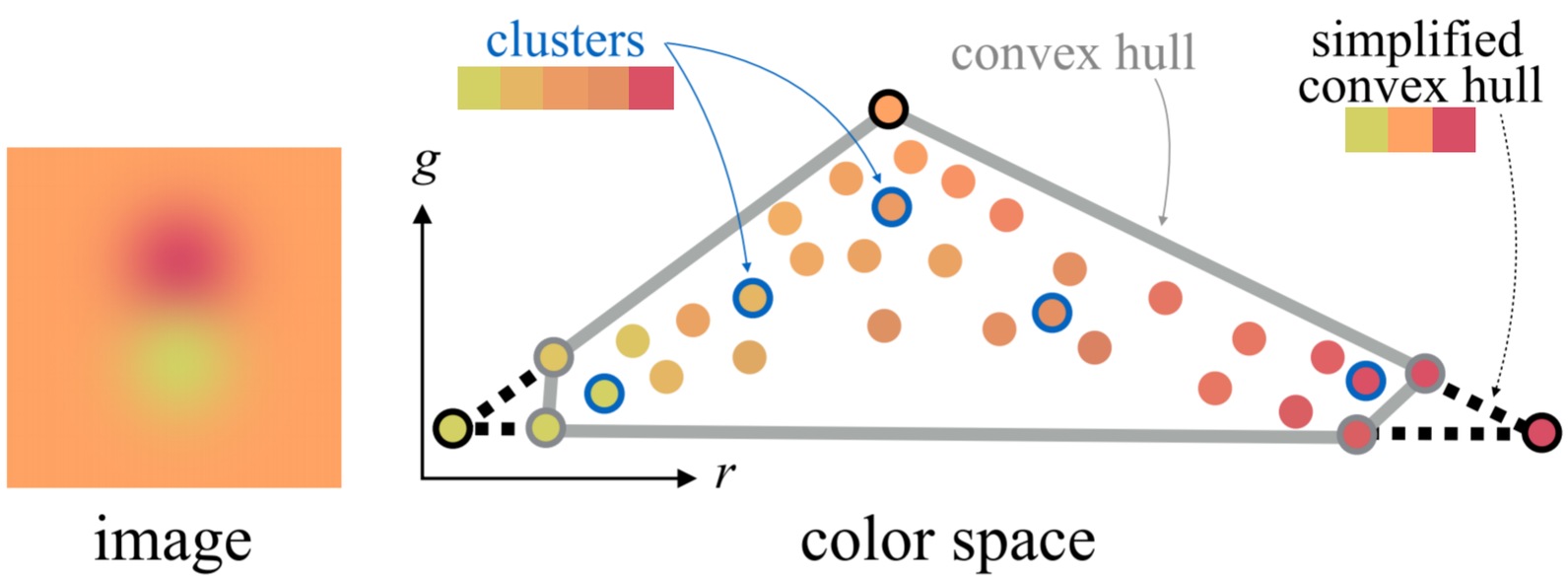
1. Palette extraction

2. Palette-based layer decomposition



Related Work

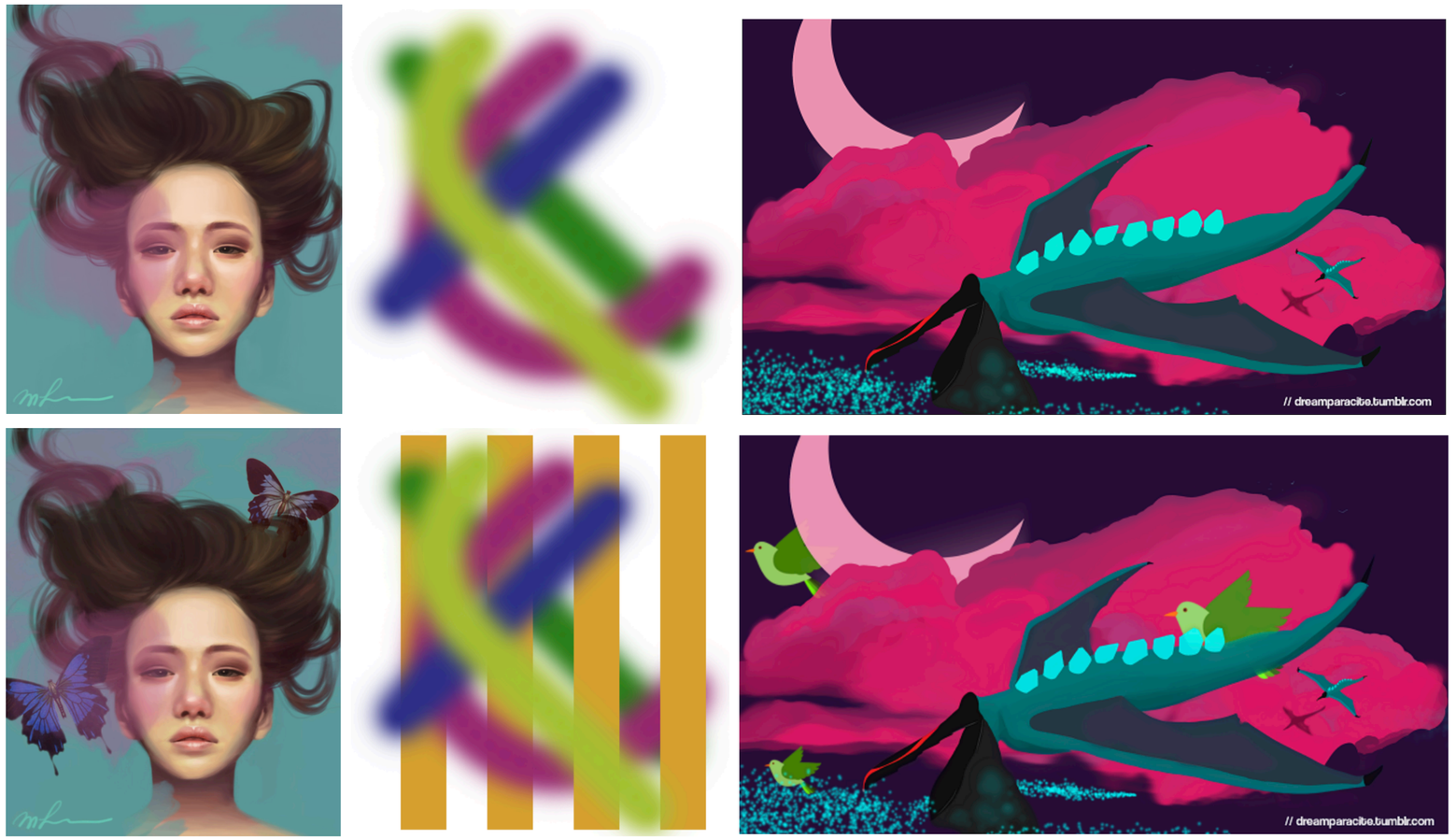
- Palette extraction for image editing
  - Shapira et al. [2009]
  - O'Donovan et al. [2011]
  - Lin et al. [2013]
  - Gerstner et al. [2013]
  - Chang et al. [2015]
  - Tan et al. [2016]



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

Related Work

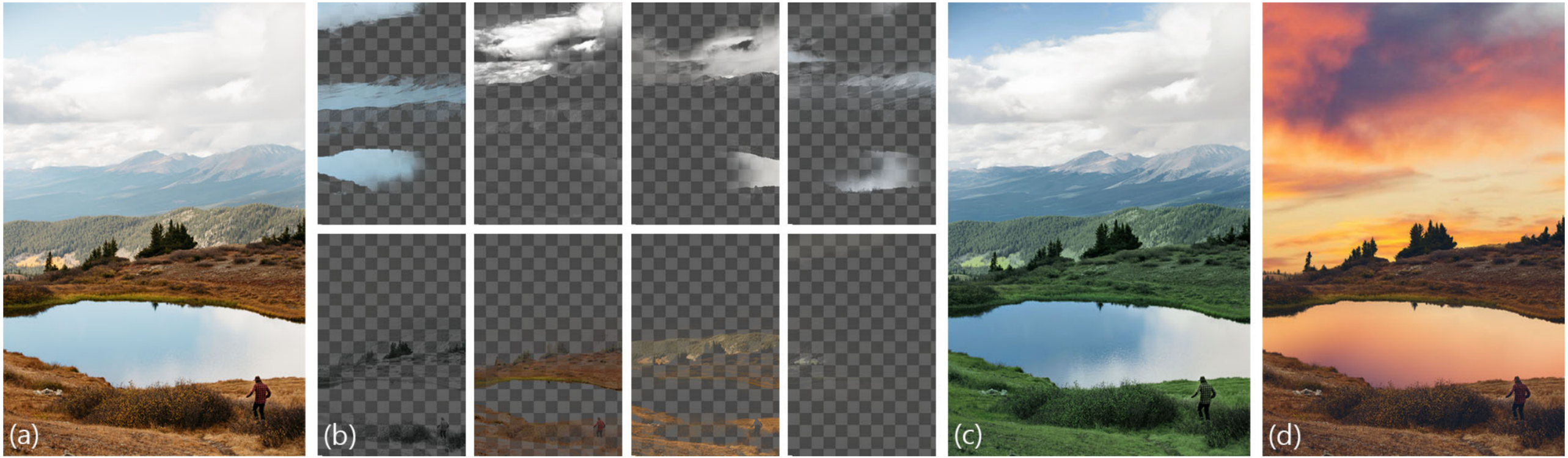
- Order-dependent translucent layers
  - Richardt et al. [2014]
  - Tan et al. [2015]
  - Tan et al. [2016]



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

Related Work

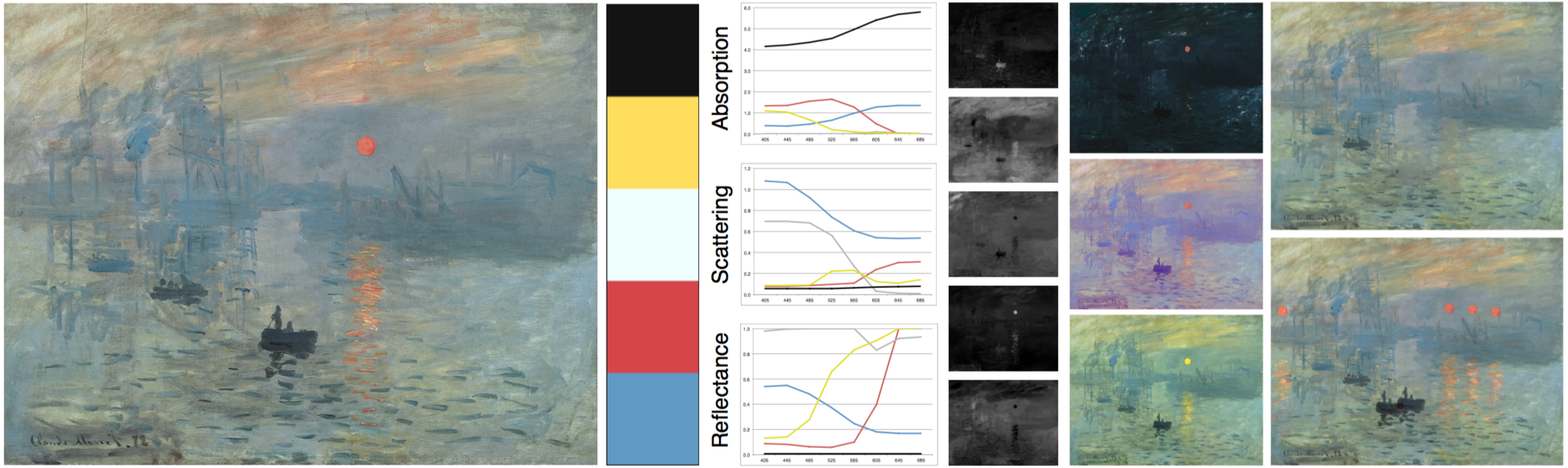
- Order-independent additive-mixing layers
  - Lin et al. [2017]; Zhang et al. [2017], Aksoy et al. [2017].



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

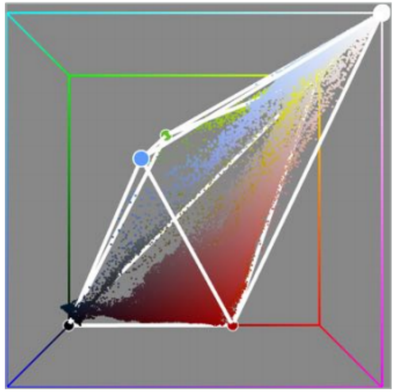
Related Work

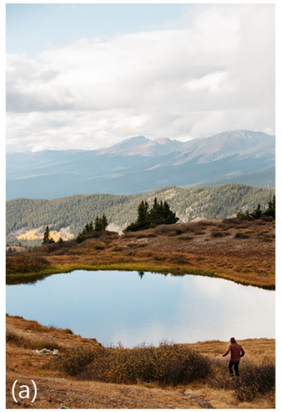
- Physically-based layers
  - Abed et al. [2014]; Tan et al. [2015]; Aharoni-Mack et al. [2017]; Tan et al. [2018].

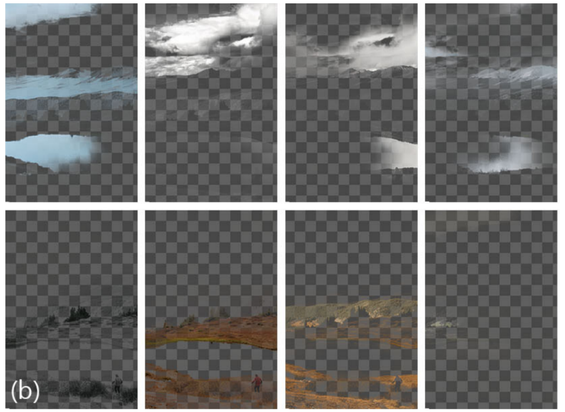


Pigmento: Pigment-Based Image Analysis and Editing [Tan et al. 2018]

Our approach

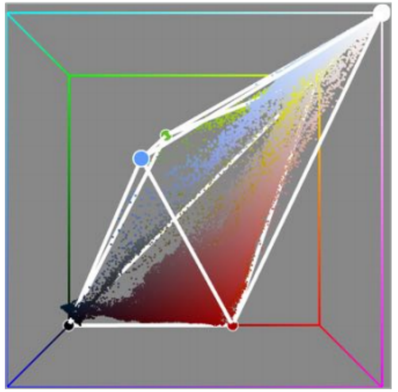


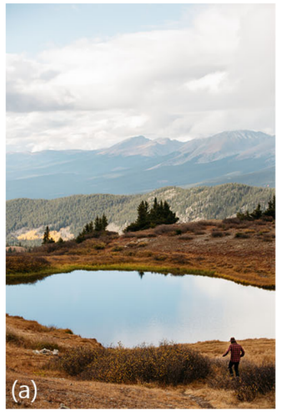


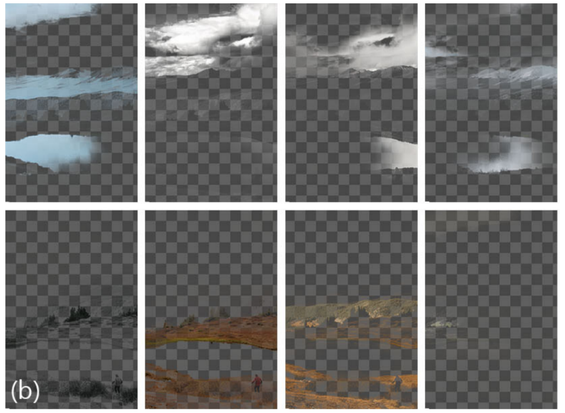


Our approach

- Geometry-based convex palettes

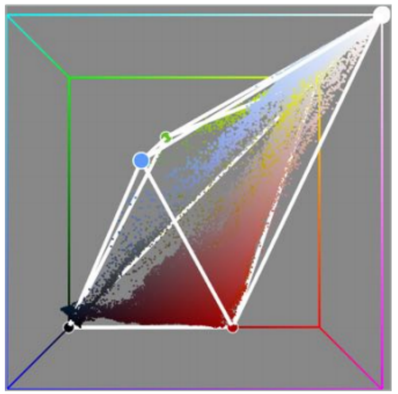


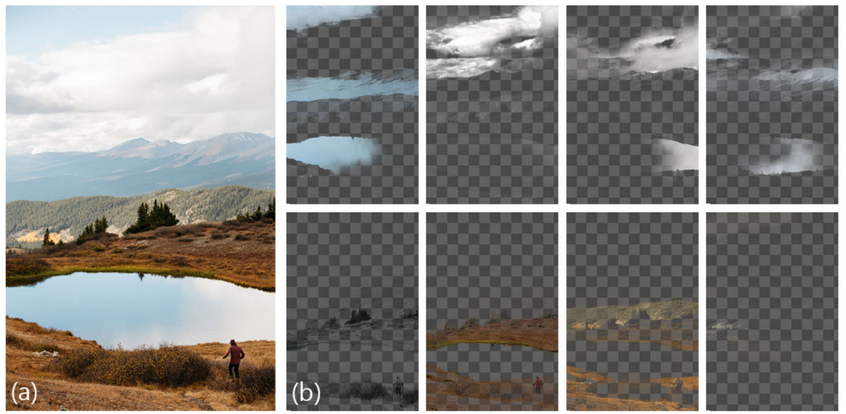




Our approach

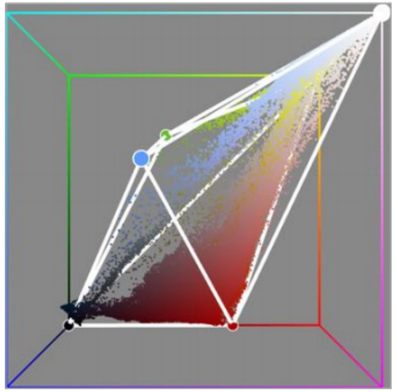
- Geometry-based convex palettes
    - Simpler

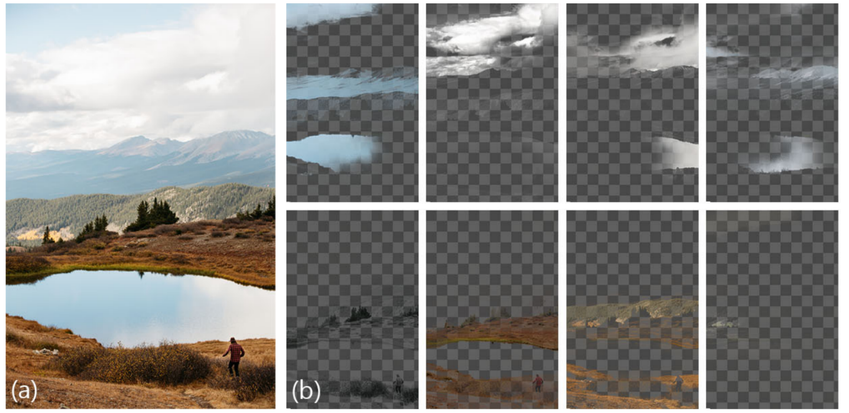




Our approach

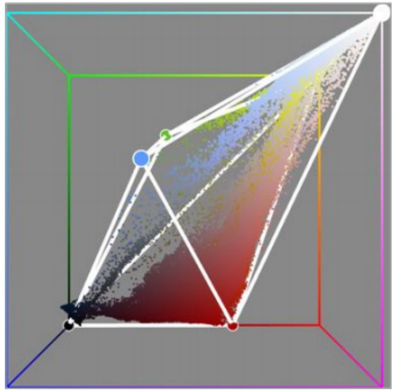
- Geometry-based convex palettes
    - Simpler
    - More general

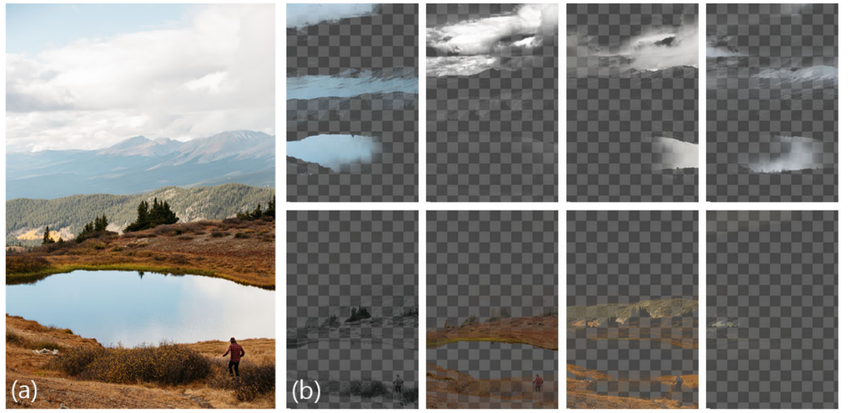




Our approach

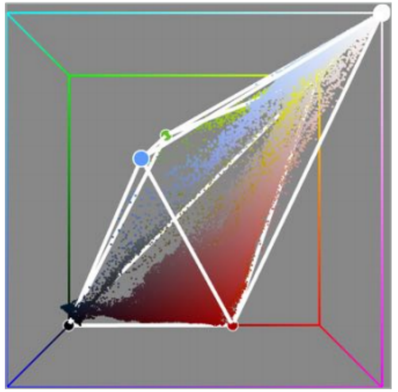
- Geometry-based convex palettes
    - Simpler
    - More general



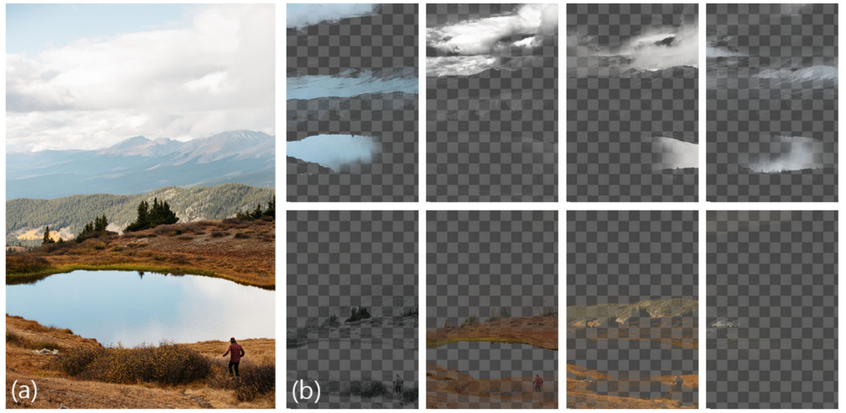


Our approach

- Geometry-based convex palettes
    - Simpler
    - More general

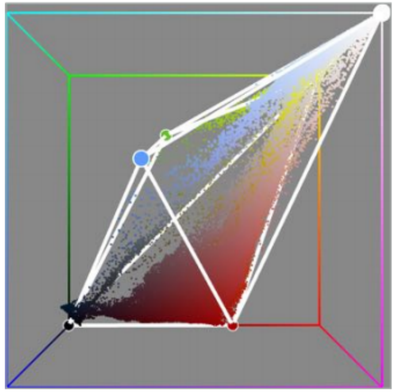


- Additive-mixing layers

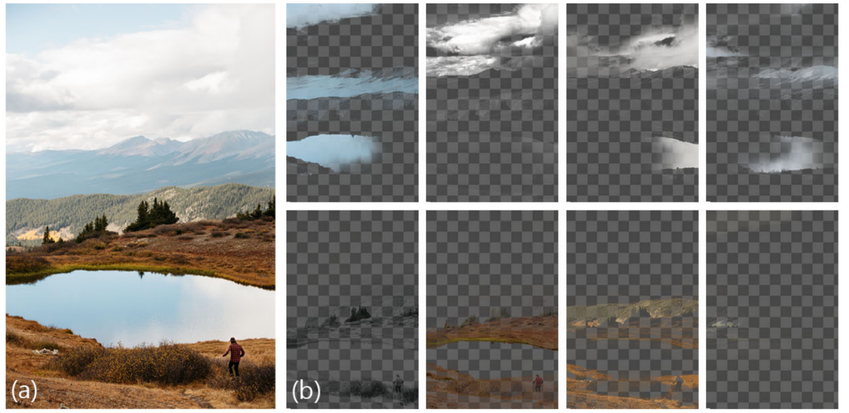


Our approach

- Geometry-based convex palettes
    - Simpler
    - More general

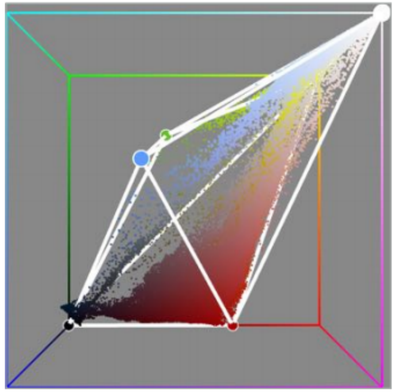


- Additive-mixing layers
    - Single colors

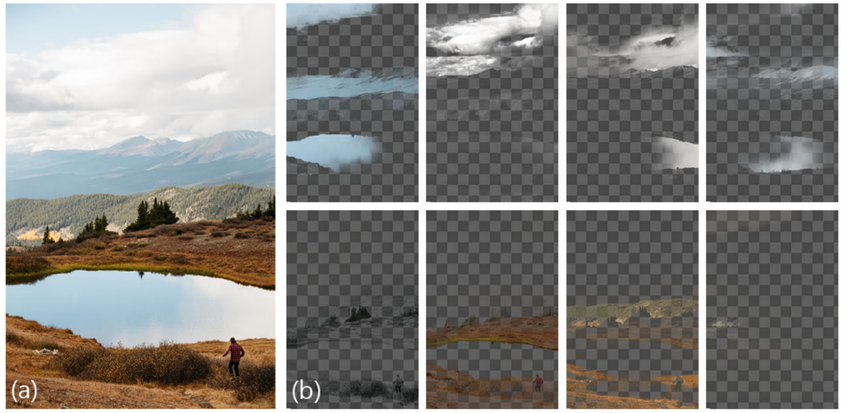


Our approach

- Geometry-based convex palettes
  - Simpler
  - More general



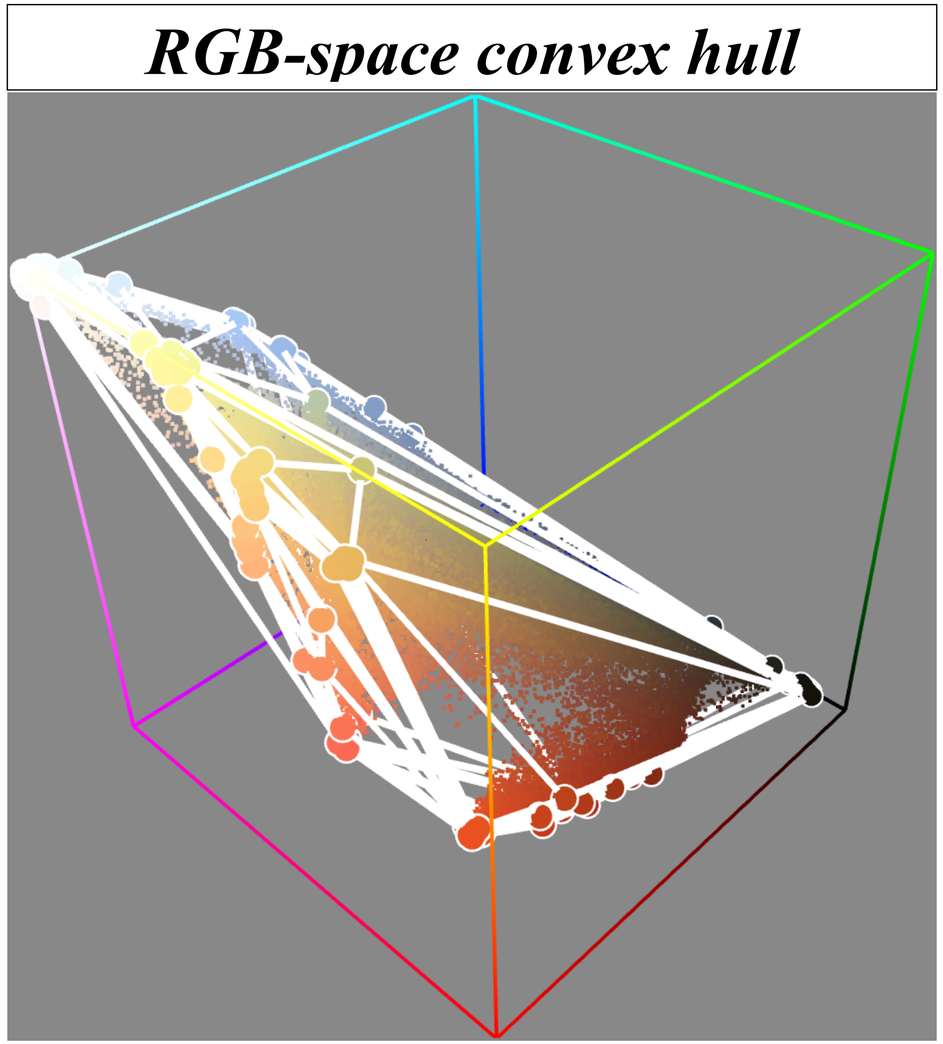
- Additive-mixing layers
  - Single colors
  - More general



Palette extraction

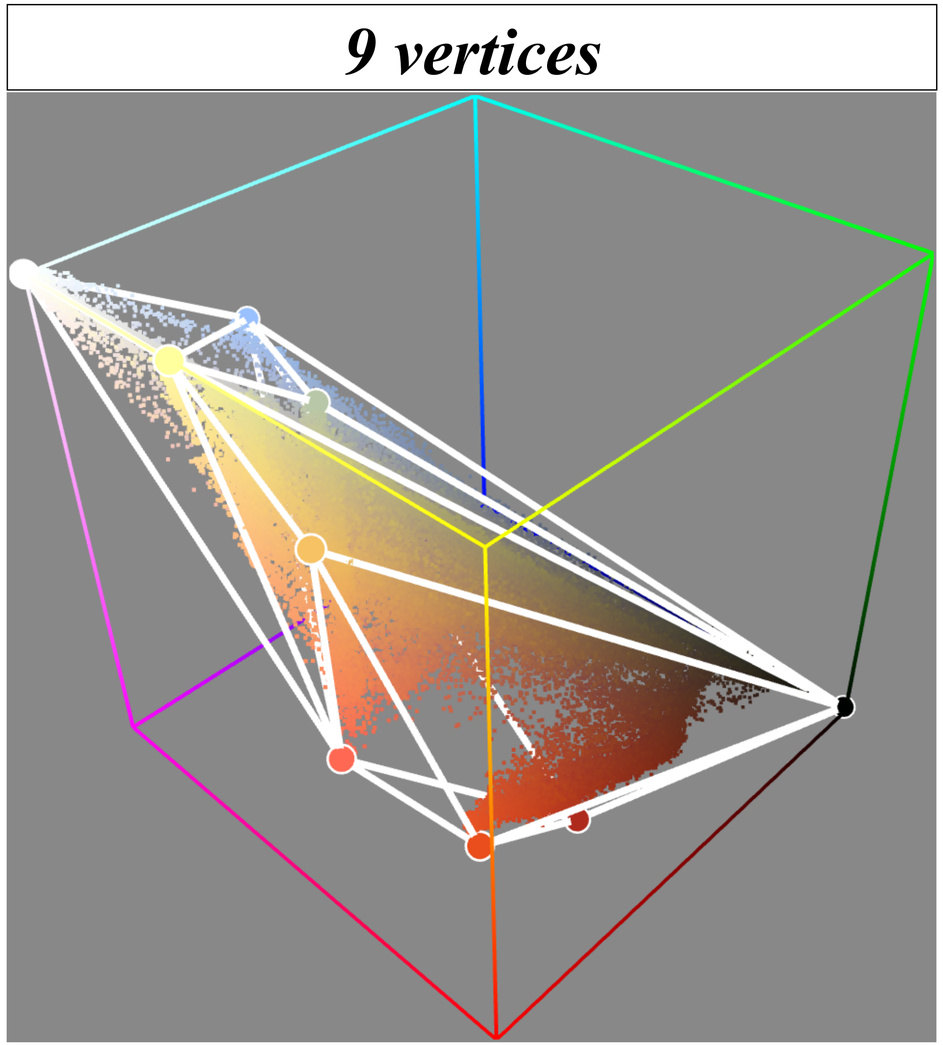
Convex hulls in RGB

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



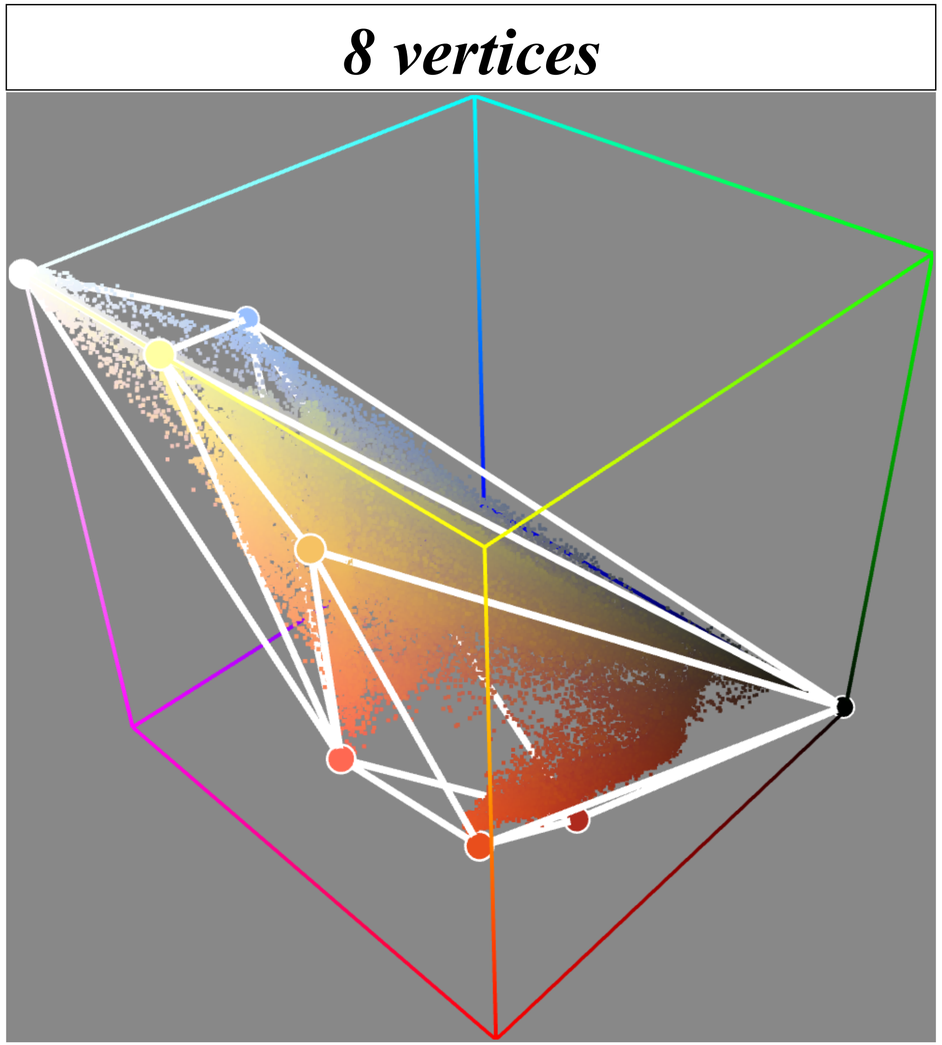
Palette Size

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



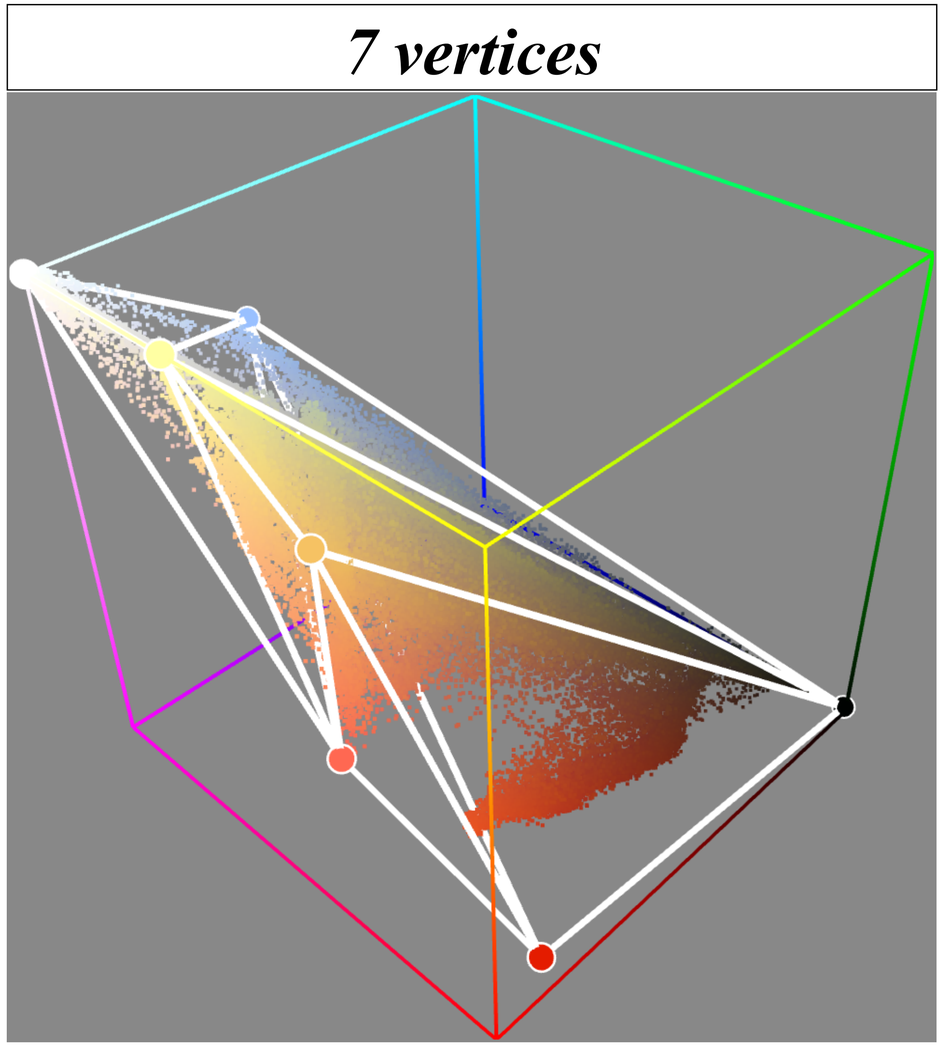
Palette Size

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



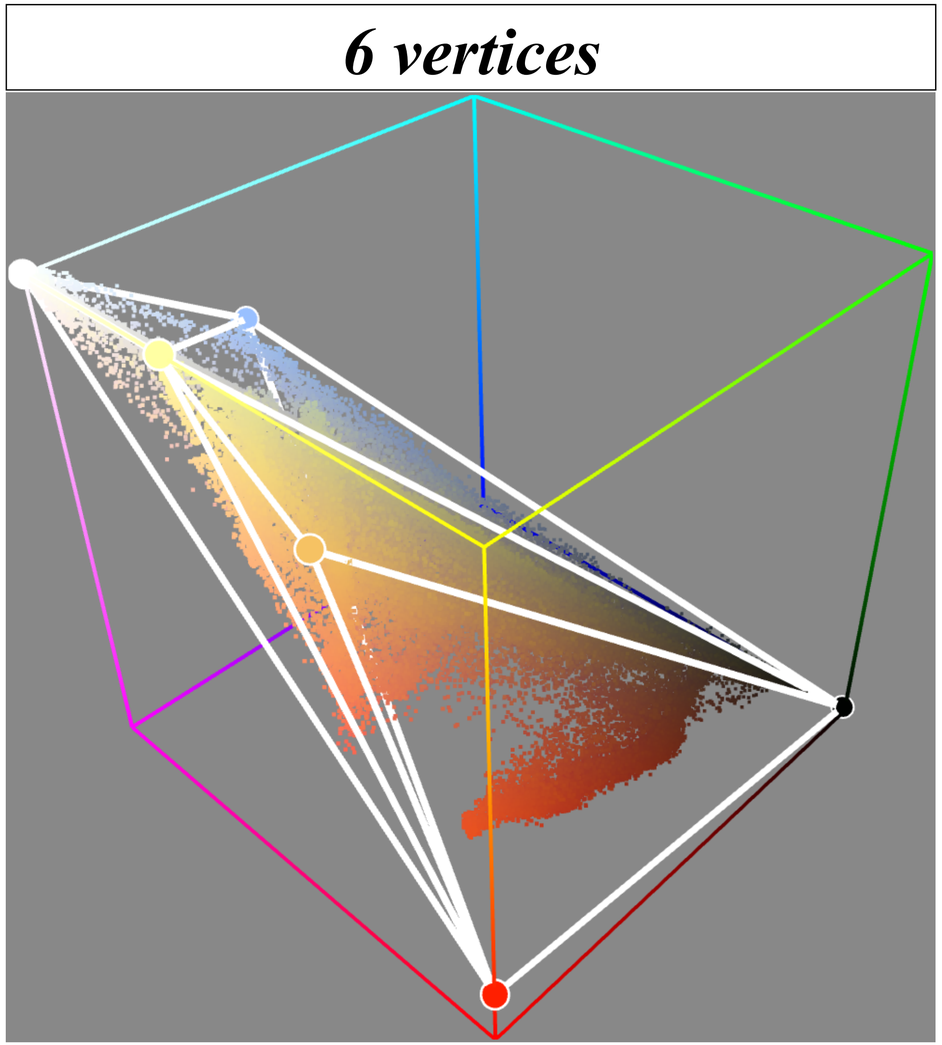
Palette Size

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



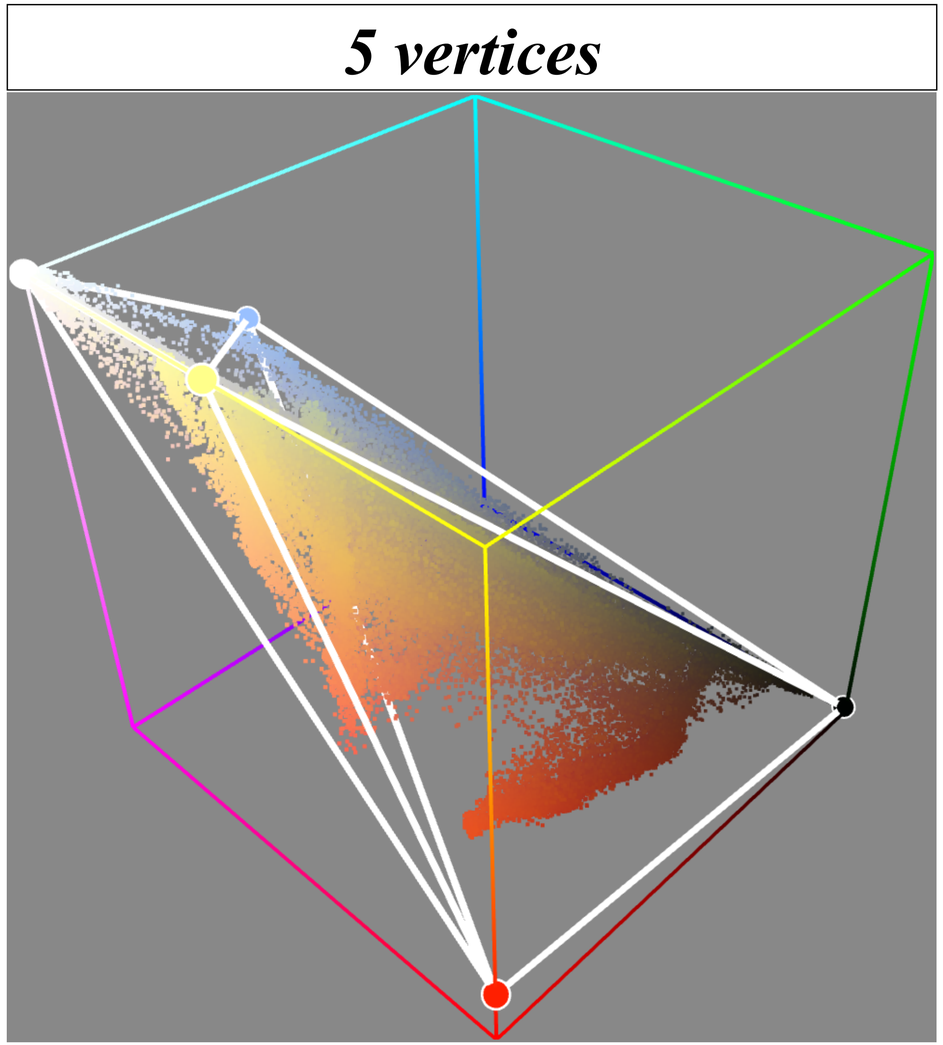
Palette Size

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



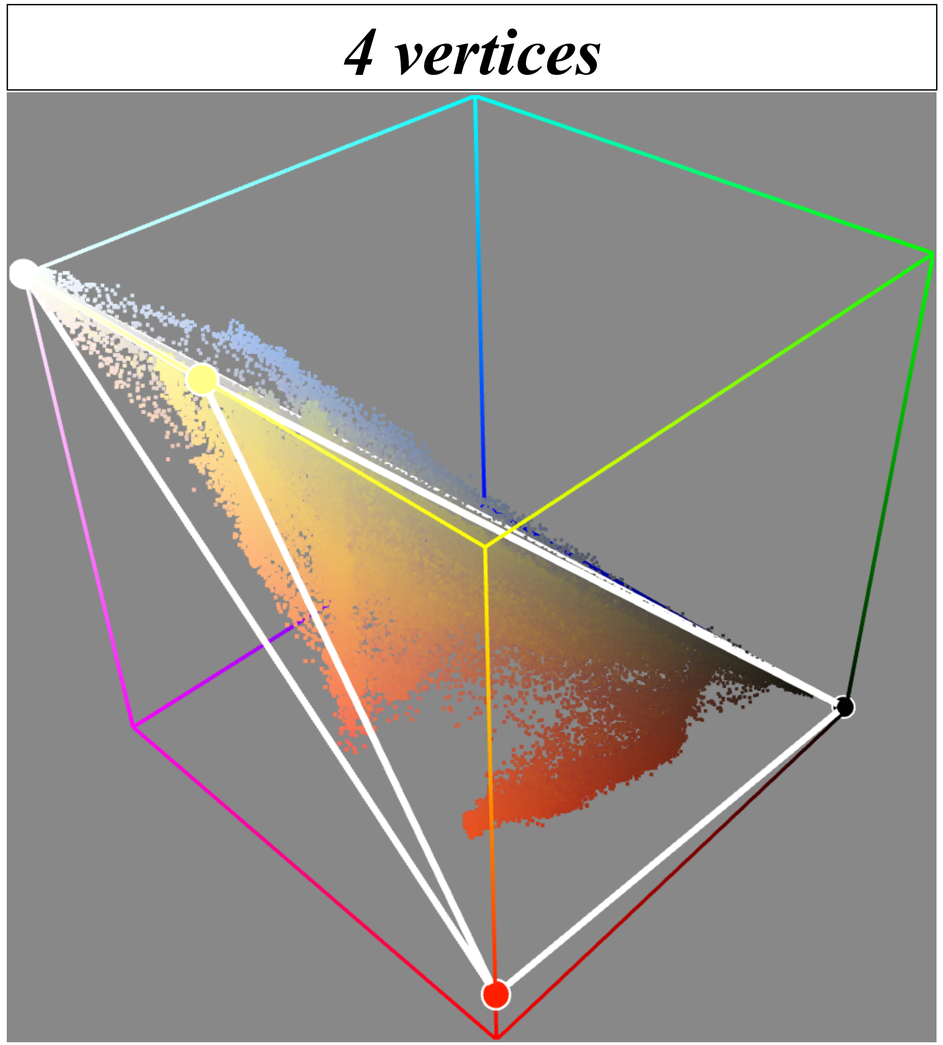
Palette Size

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



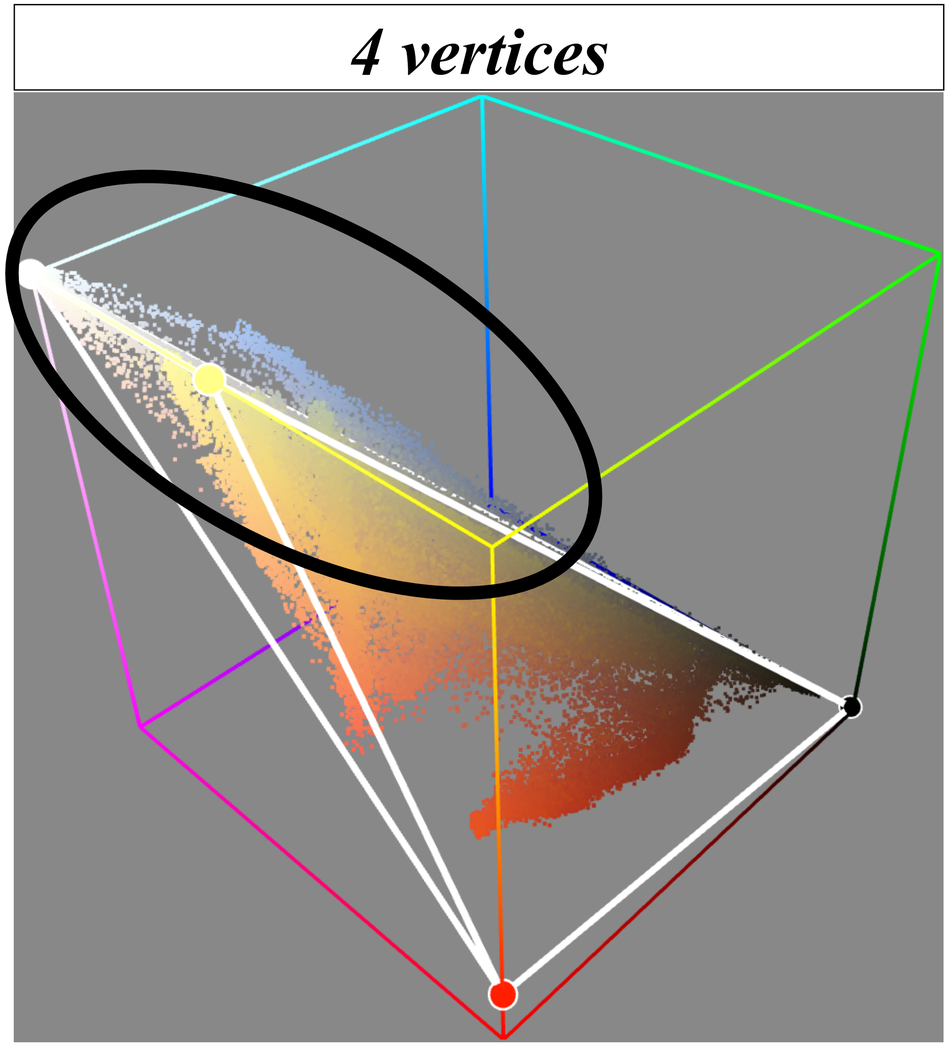
Palette Size

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



Palette Size

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



Palette Size

- Our automatic error-bound simplification

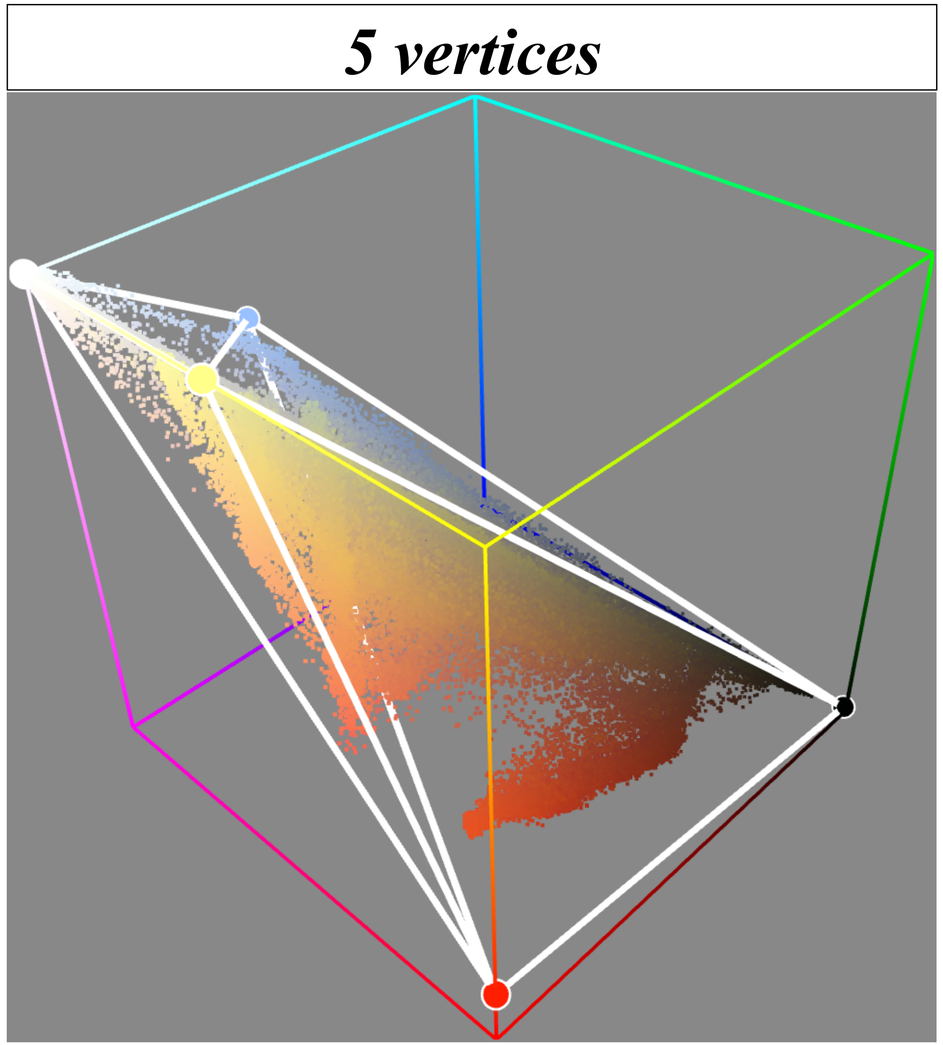


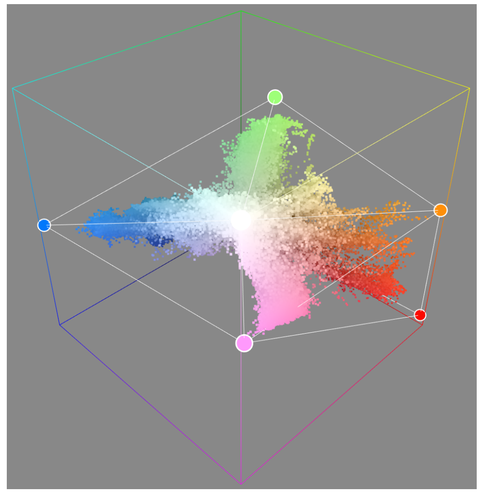
Image Decomposition

Extracting mixing weights

image

palette

RGB-space



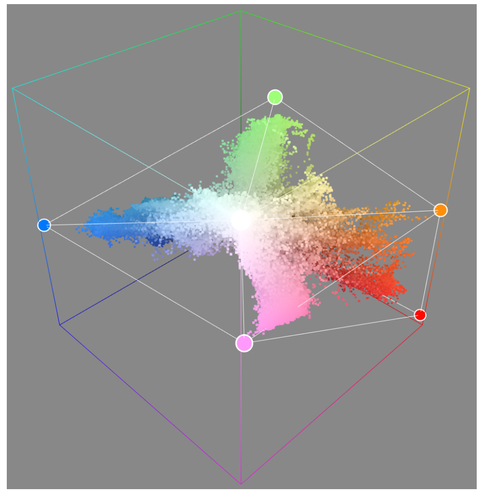
Optimization

Extracting mixing weights

image

palette

RGB-space



Optimization

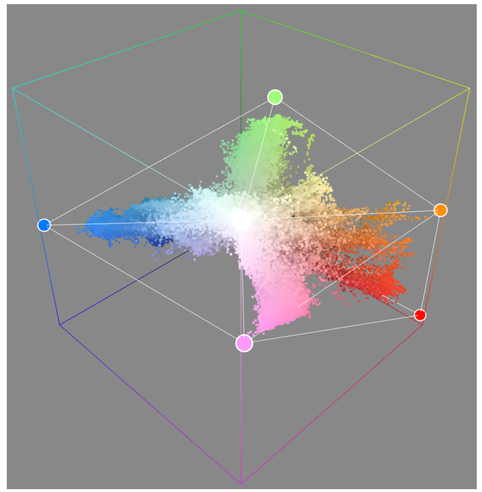
- Slow for high resolutions

Extracting mixing weights

image

palette

RGB-space



Optimization

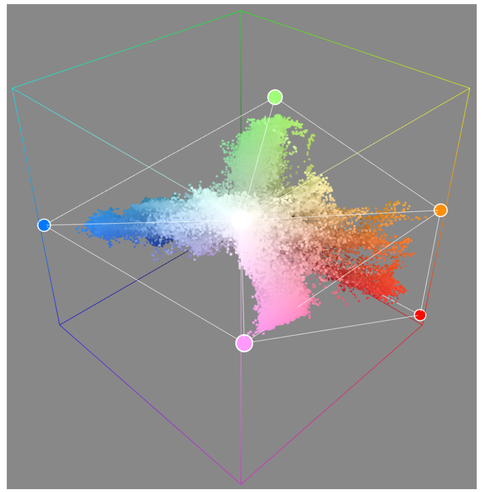
- Slow for high resolutions
- Many parameters to tune

Extracting mixing weights

image

palette

RGB-space



Optimization

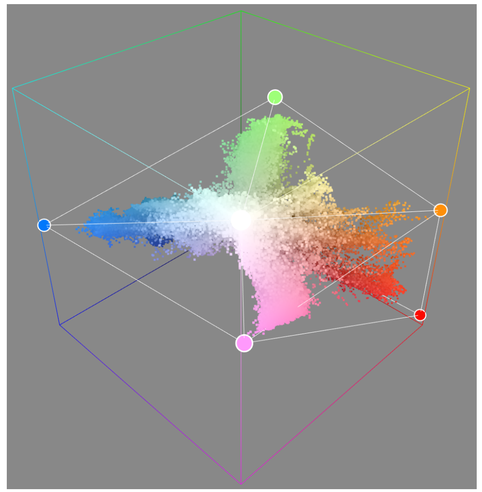
- Slow for high resolutions
- Many parameters to tune
- Per-image parameters

Extracting mixing weights

image

palette

RGB-space



Optimization

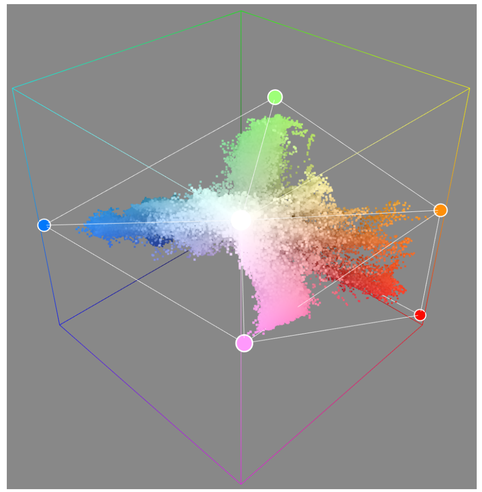
- Slow for high resolutions
- Many parameters to tune
- Per-image parameters

Extracting mixing weights

image

palette

RGB-space



Optimization

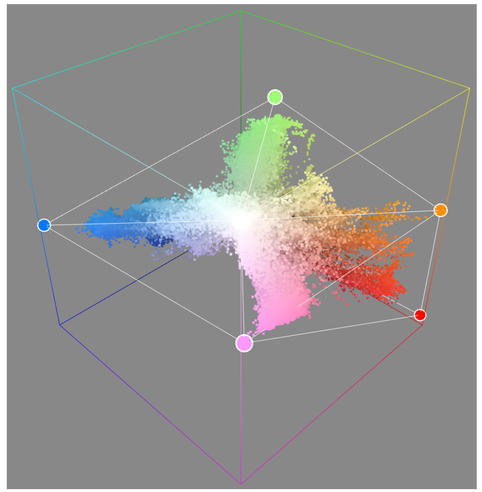
- Slow for high resolutions
- Many parameters to tune
- Per-image parameters

Extracting mixing weights

image

palette

RGB-space



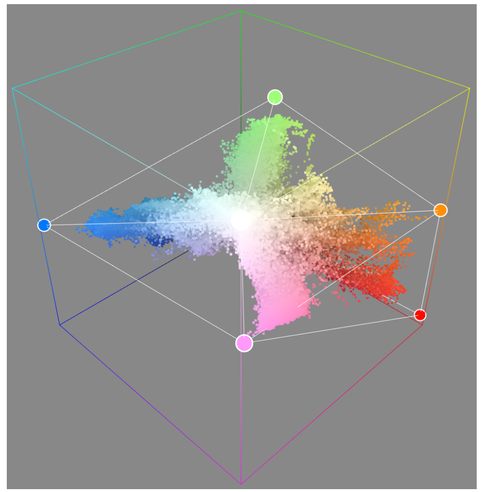
~~Optimization~~

Extracting mixing weights

image

palette

RGB-space



~~Optimization~~

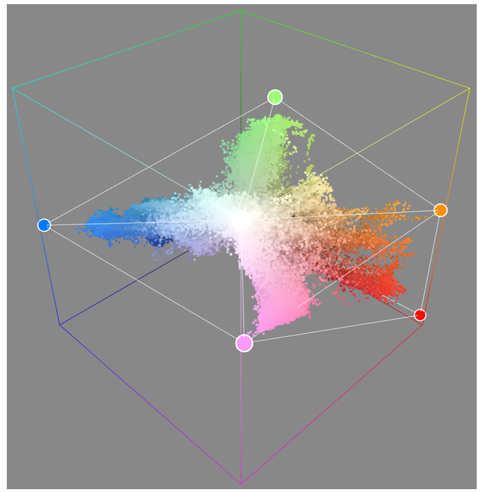
Generalized Barycentric Coordinates

Extracting mixing weights

image

palette

RGB-space



~~Optimization~~

Generalized Barycentric Coordinates

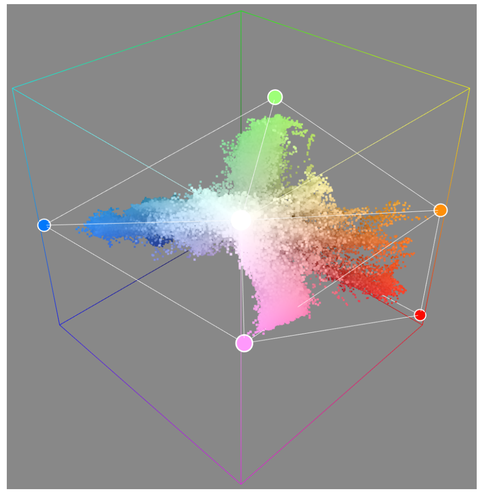
- Fast

Extracting mixing weights

image

palette

RGB-space



~~Optimization~~

Generalized Barycentric Coordinates

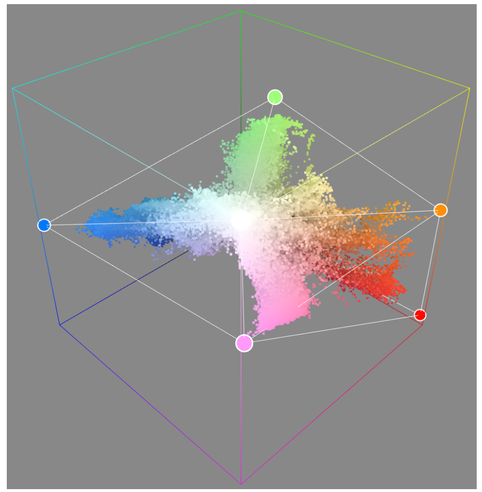
- Fast
- No parameters to tune

Extracting mixing weights

image

palette

RGB-space



~~Optimization~~

Generalized Barycentric Coordinates

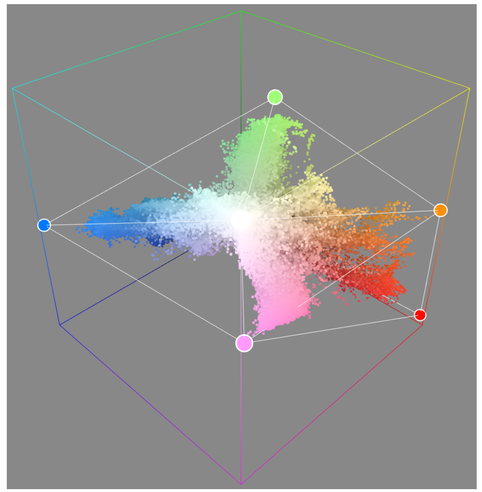
- Fast
- No parameters to tune
- Does not guarantee spatial smoothness

Extracting mixing weights

image

palette

RGB-space



~~Optimization~~

Generalized Barycentric Coordinates

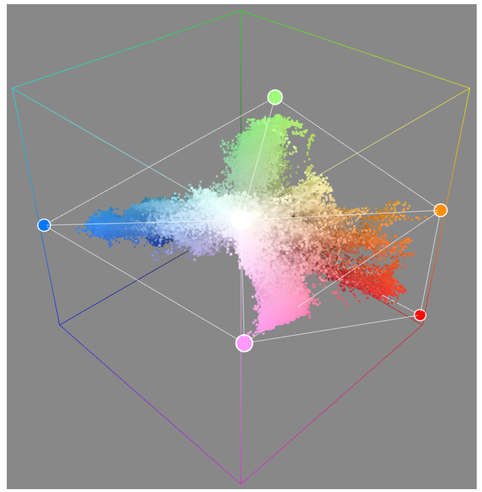
- Fast
- No parameters to tune
- Does not guarantee spatial smoothness

Extracting mixing weights

image

palette

RGB-space



~~Optimization~~

Generalized Barycentric Coordinates

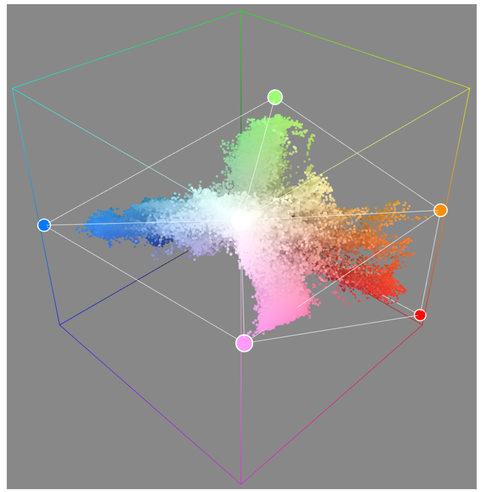
- Fast
- No parameters to tune
- Does not guarantee spatial smoothness

Extracting mixing weights

image

palette

RGB-space



~~Optimization~~

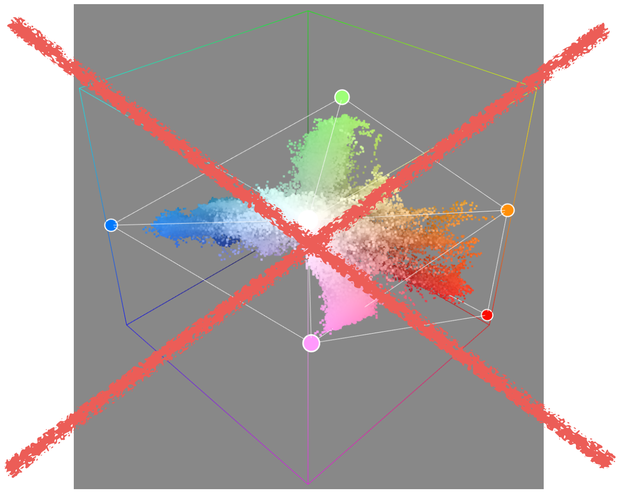
~~Generalized Barycentric Coordinates~~

Extracting mixing weights

image

palette

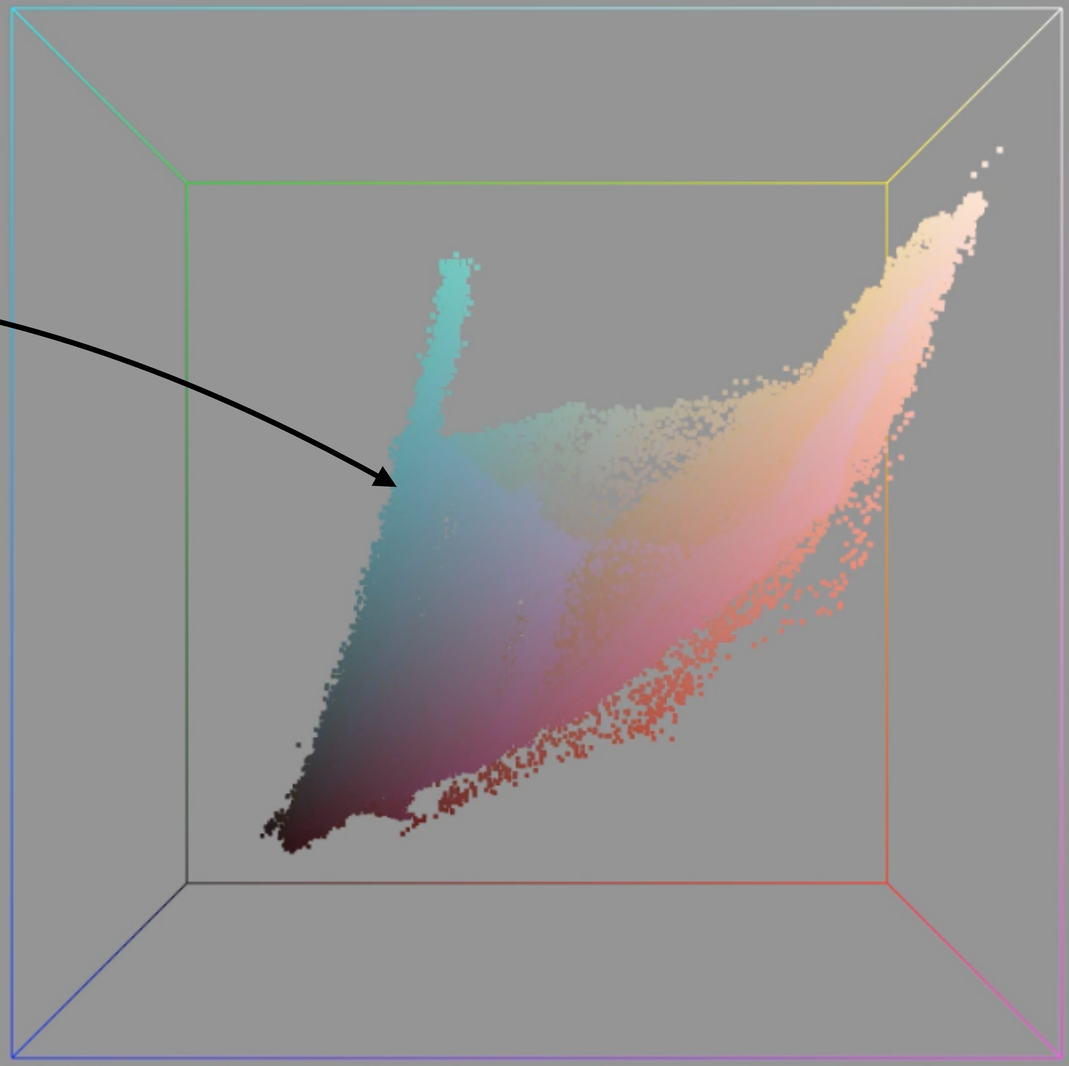
RGB-space



~~Optimization~~

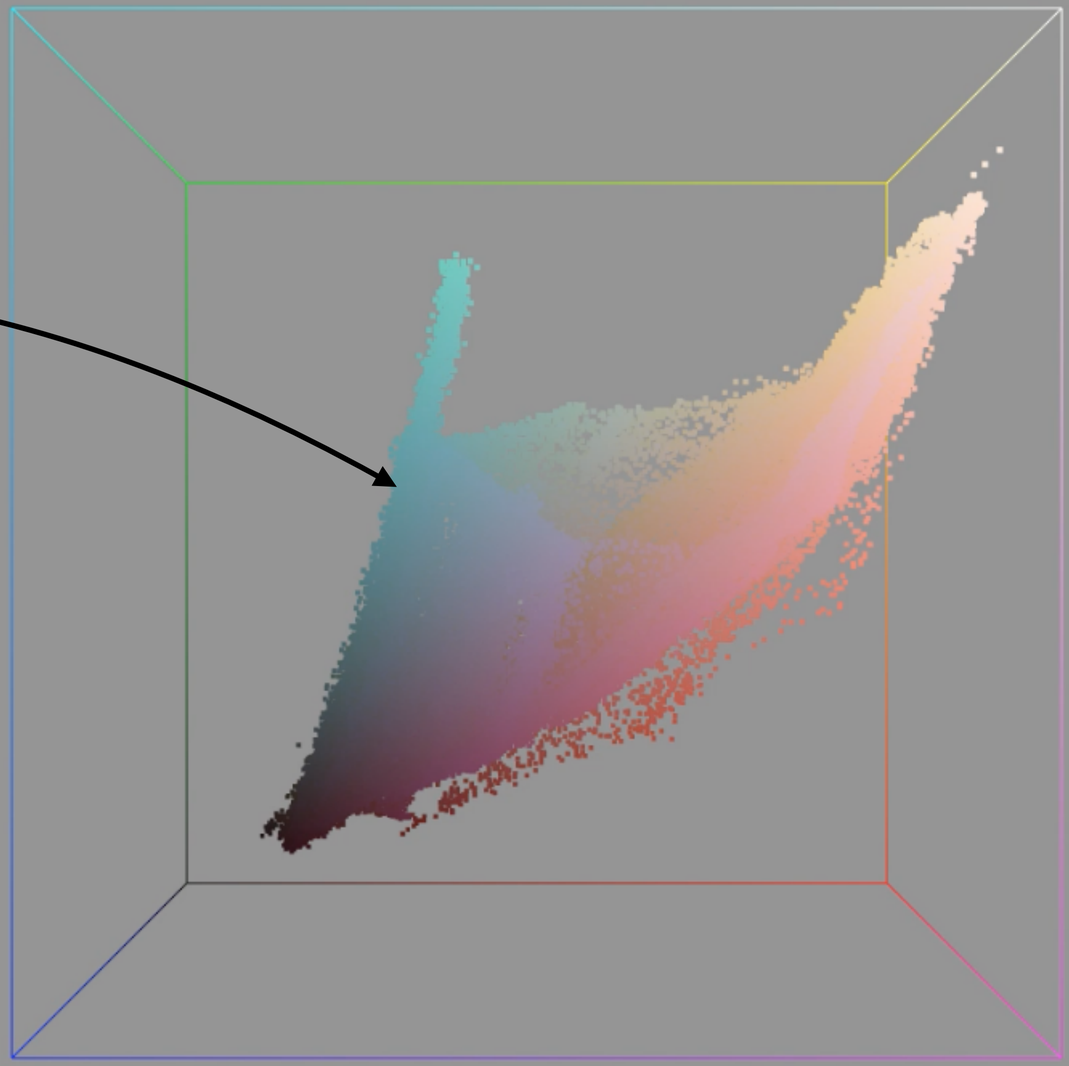
~~Generalized Barycentric Coordinates~~

(R,G,B)

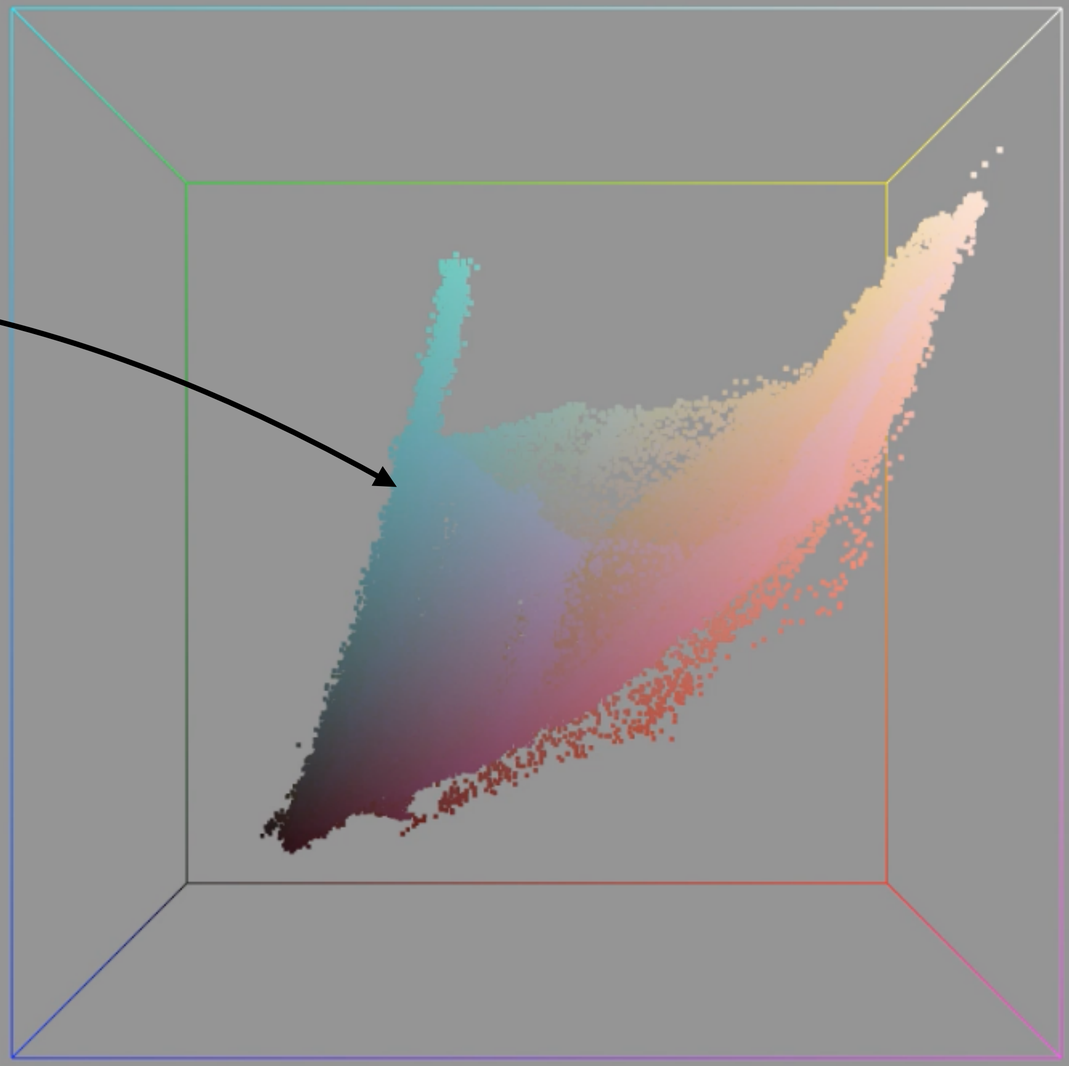


(R,G,B)

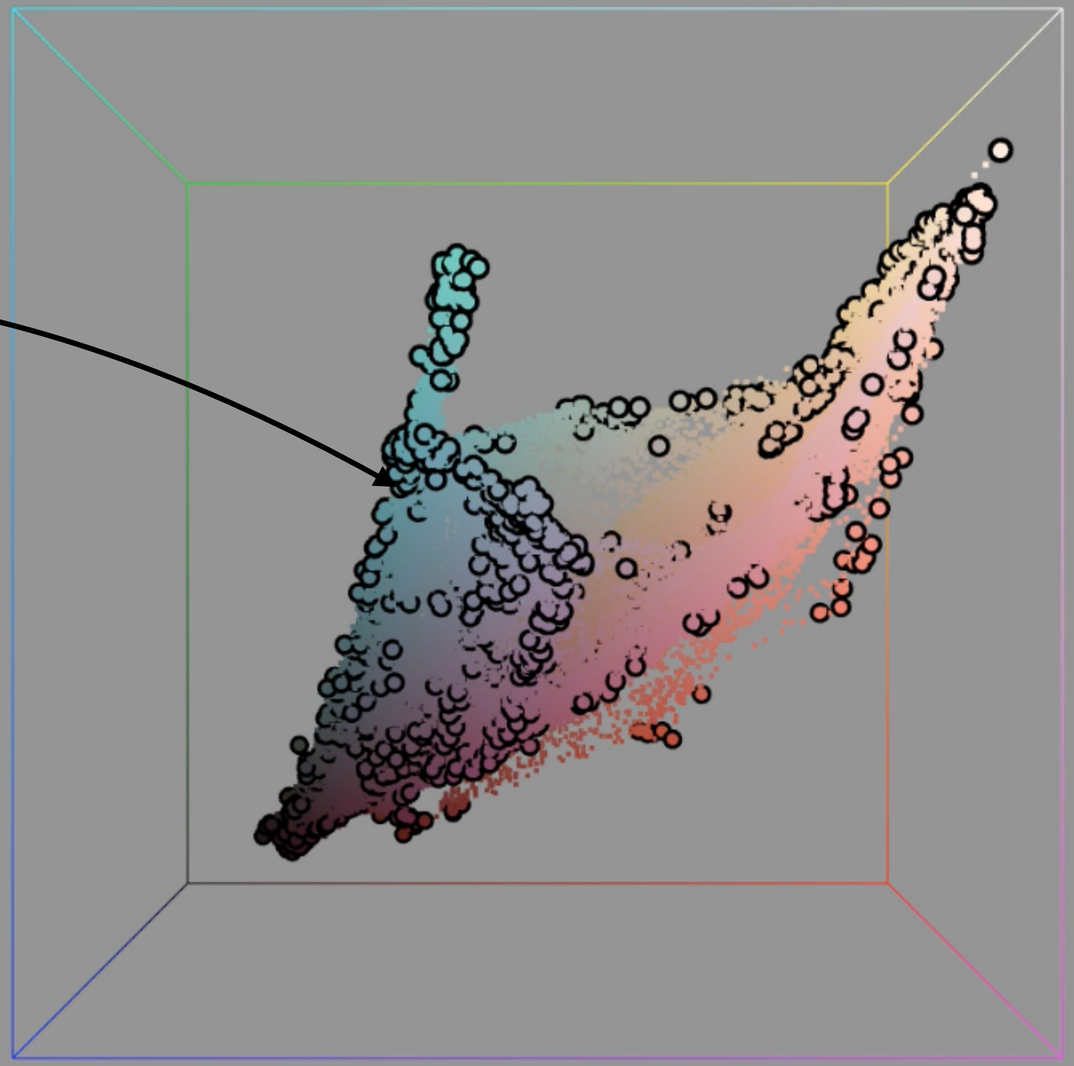
(x,y)

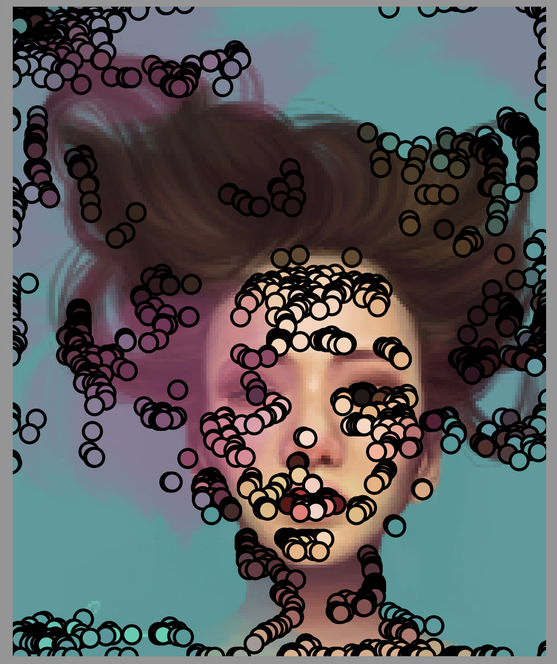


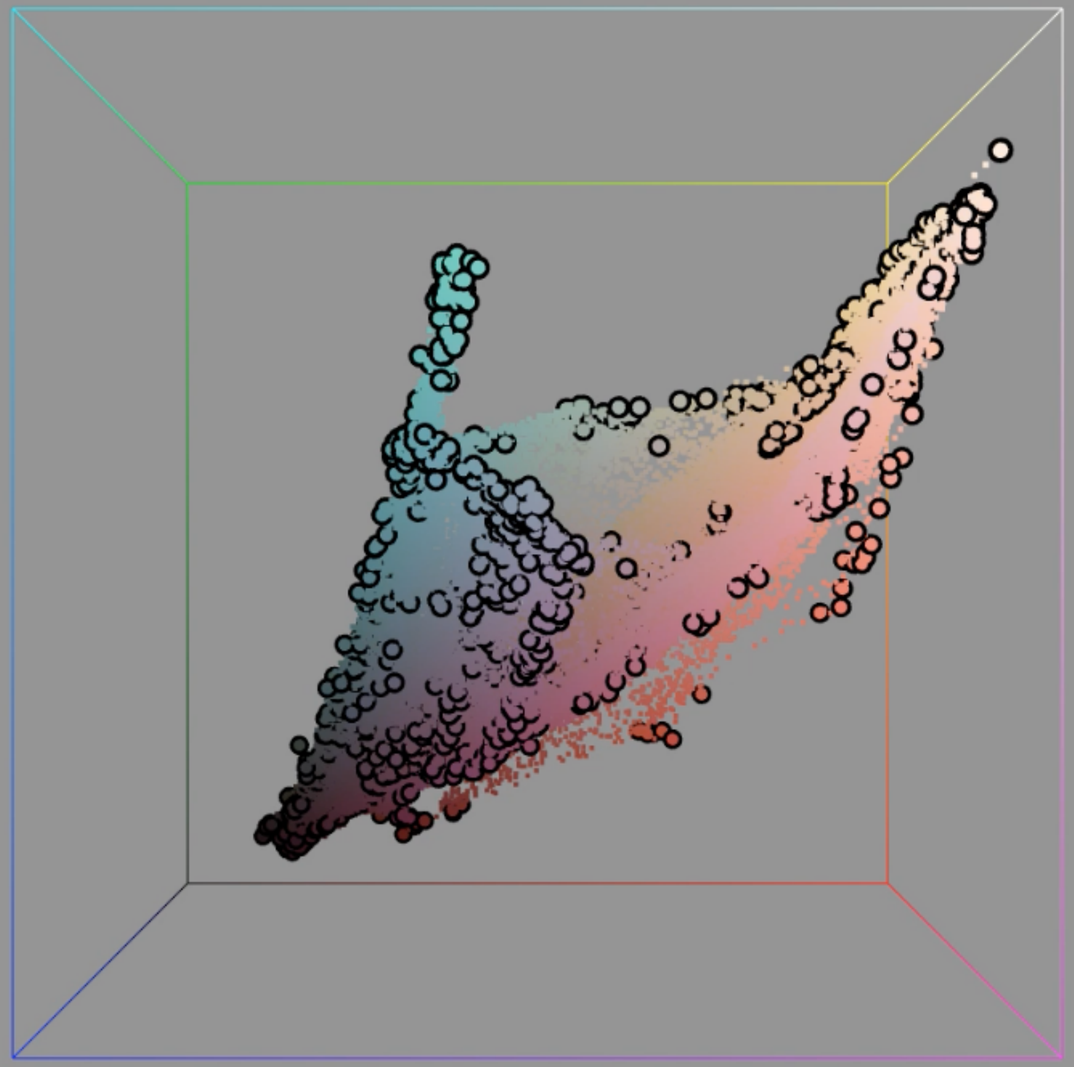
(R,G,B, x, y)

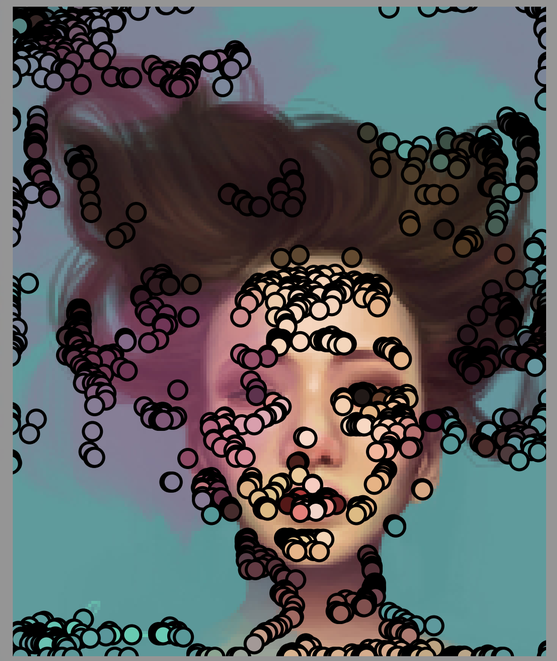


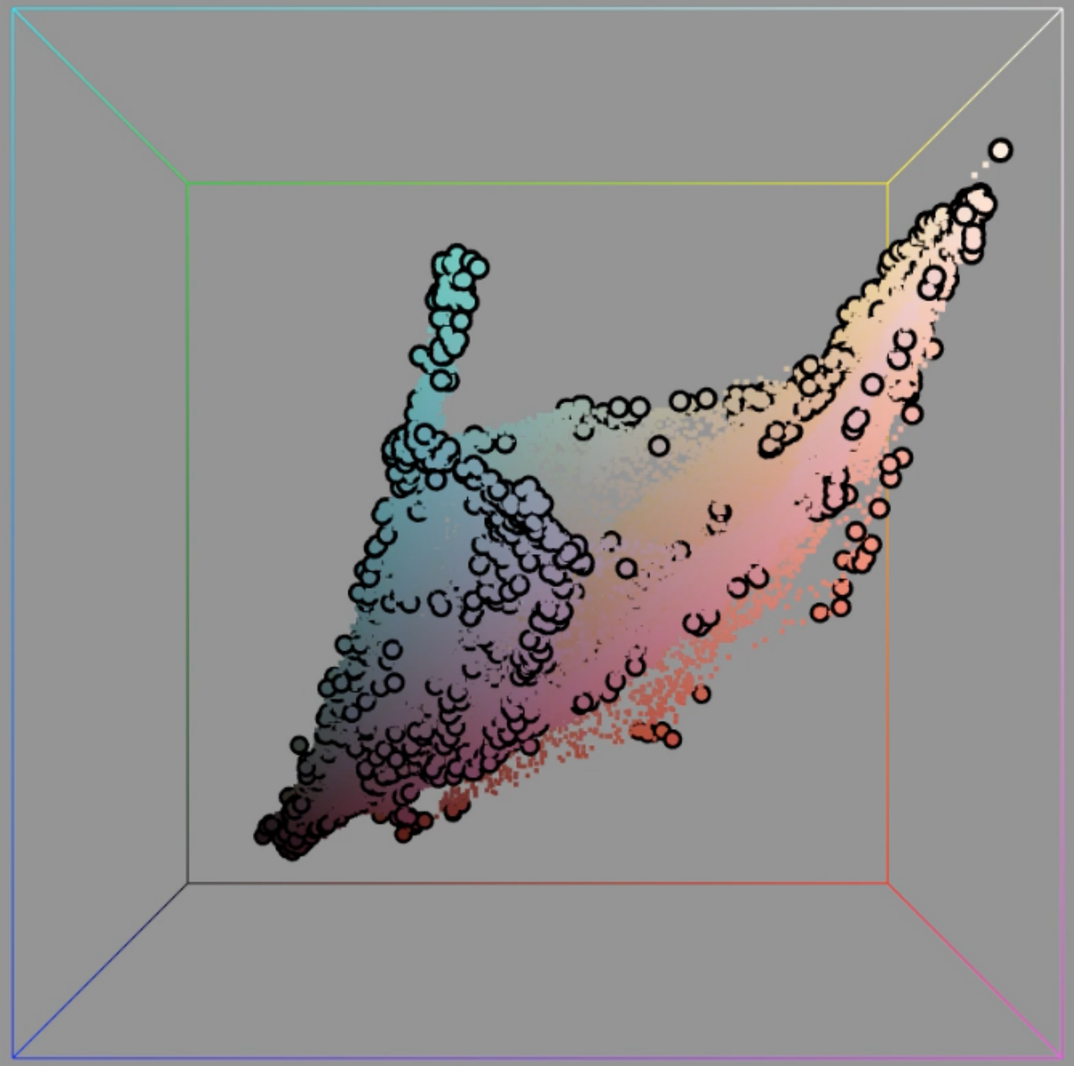
(R,G,B, x, y)

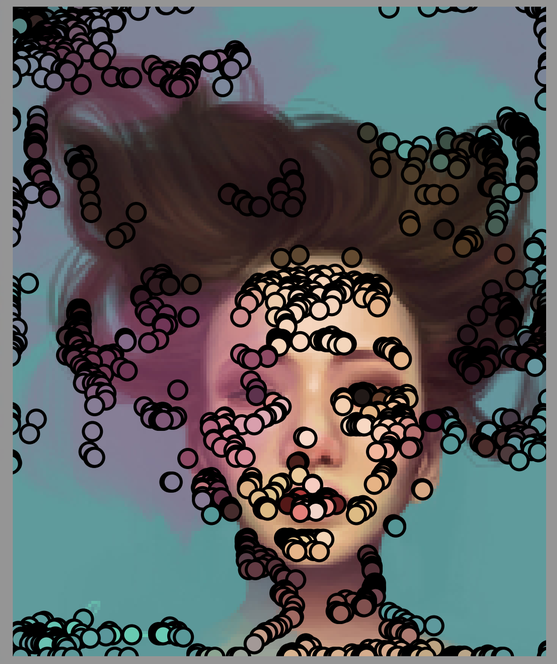




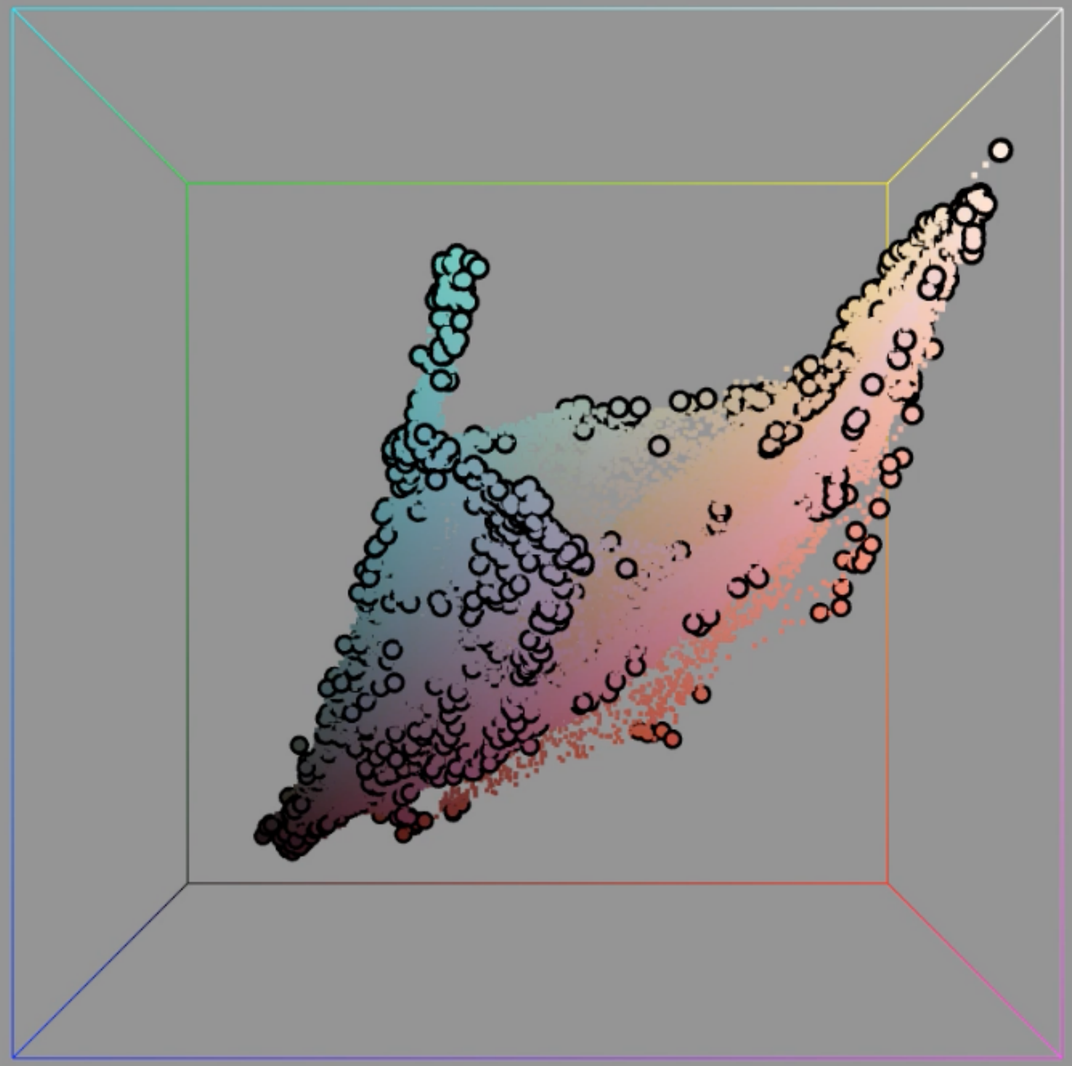


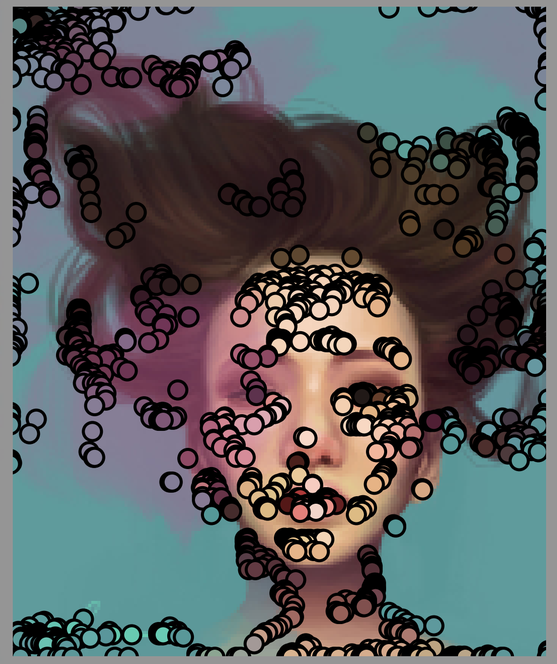




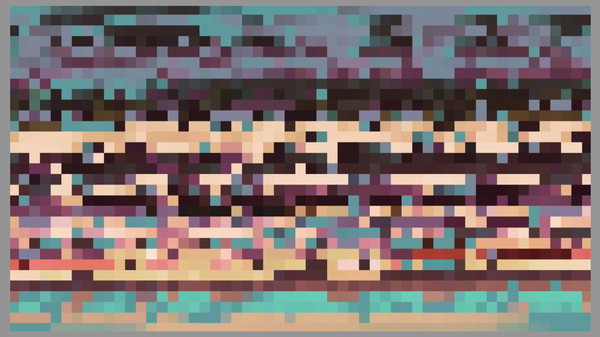


RGBXY palette
(projected to RGB)

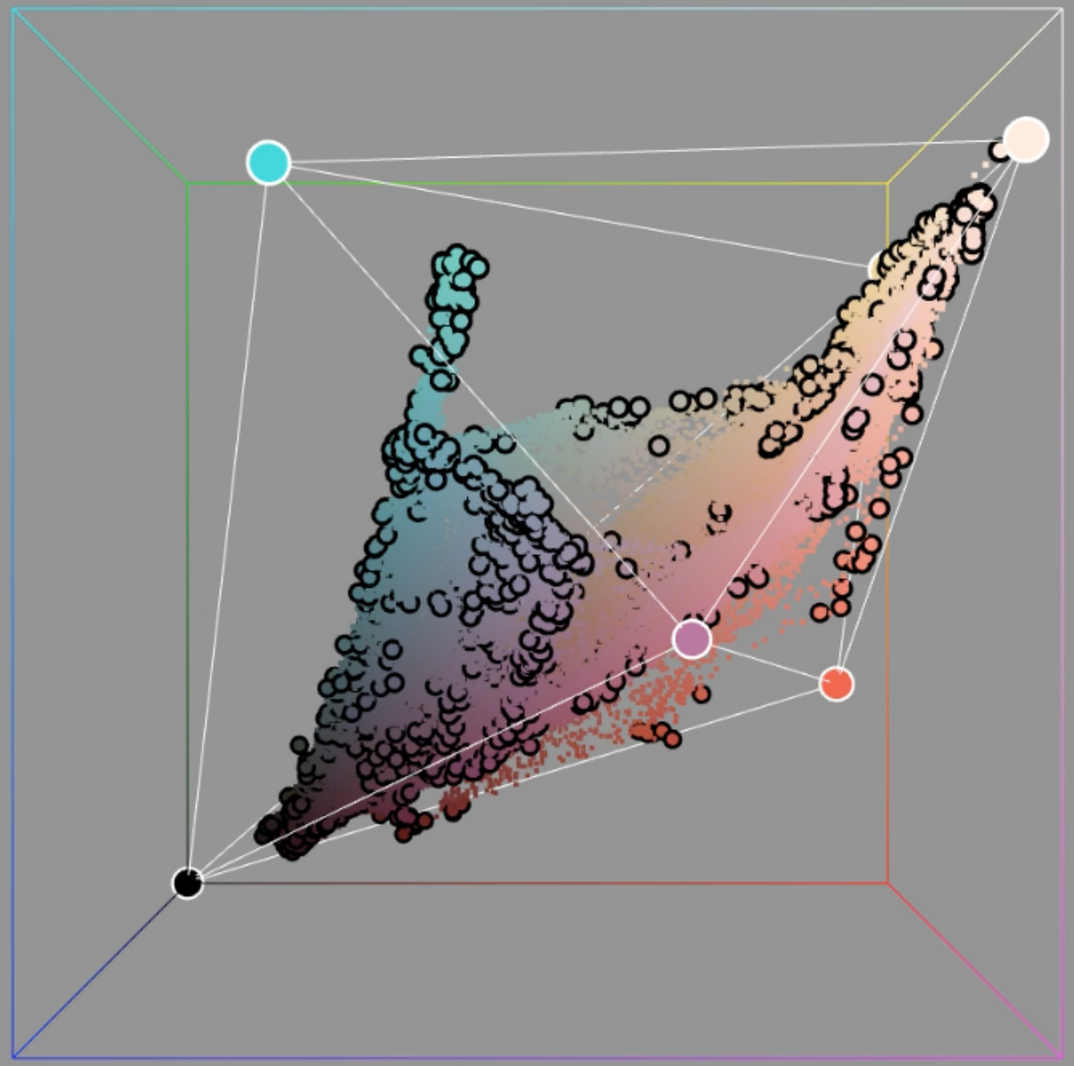




RGBXY palette
(projected to RGB)



RGB palette

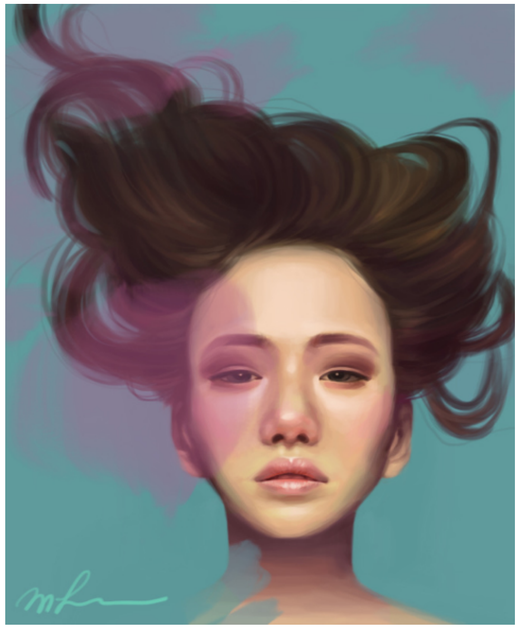


Two-level decomposition

RGB palette

mixing weights **W**

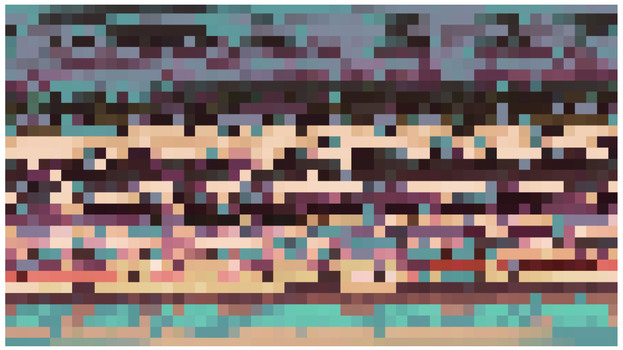
image



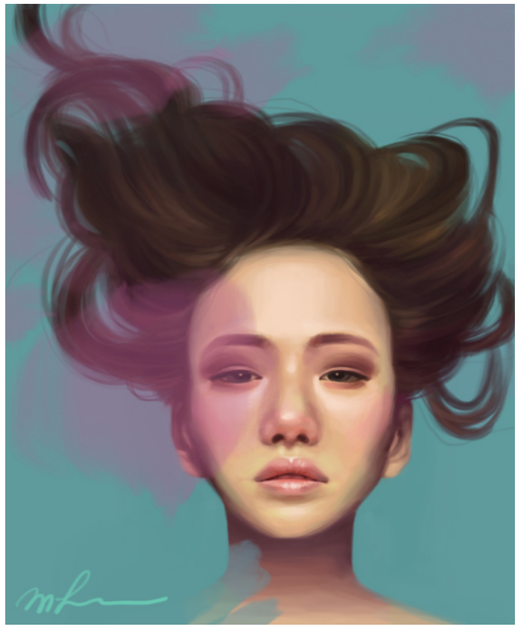
Two-level decomposition

RGB palette

RGBXY vertices
(projected to RGB)



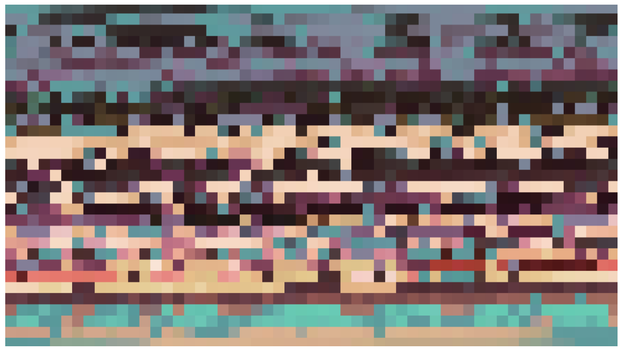
image



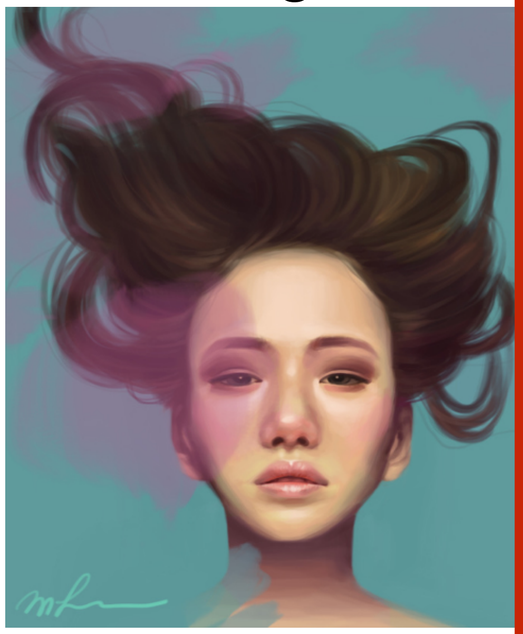
Two-level decomposition

RGB palette

RGBXY vertices
(projected to RGB)



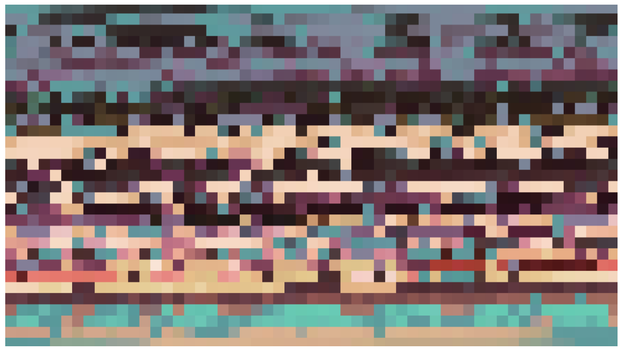
image



Two-level decomposition

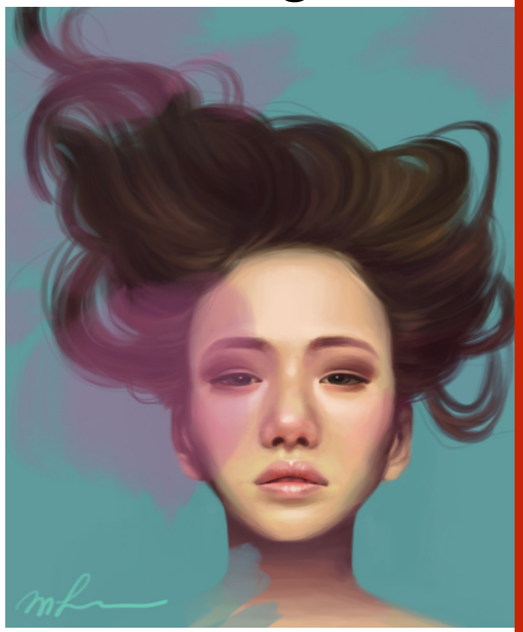
RGB palette

RGBXY vertices
(projected to RGB)

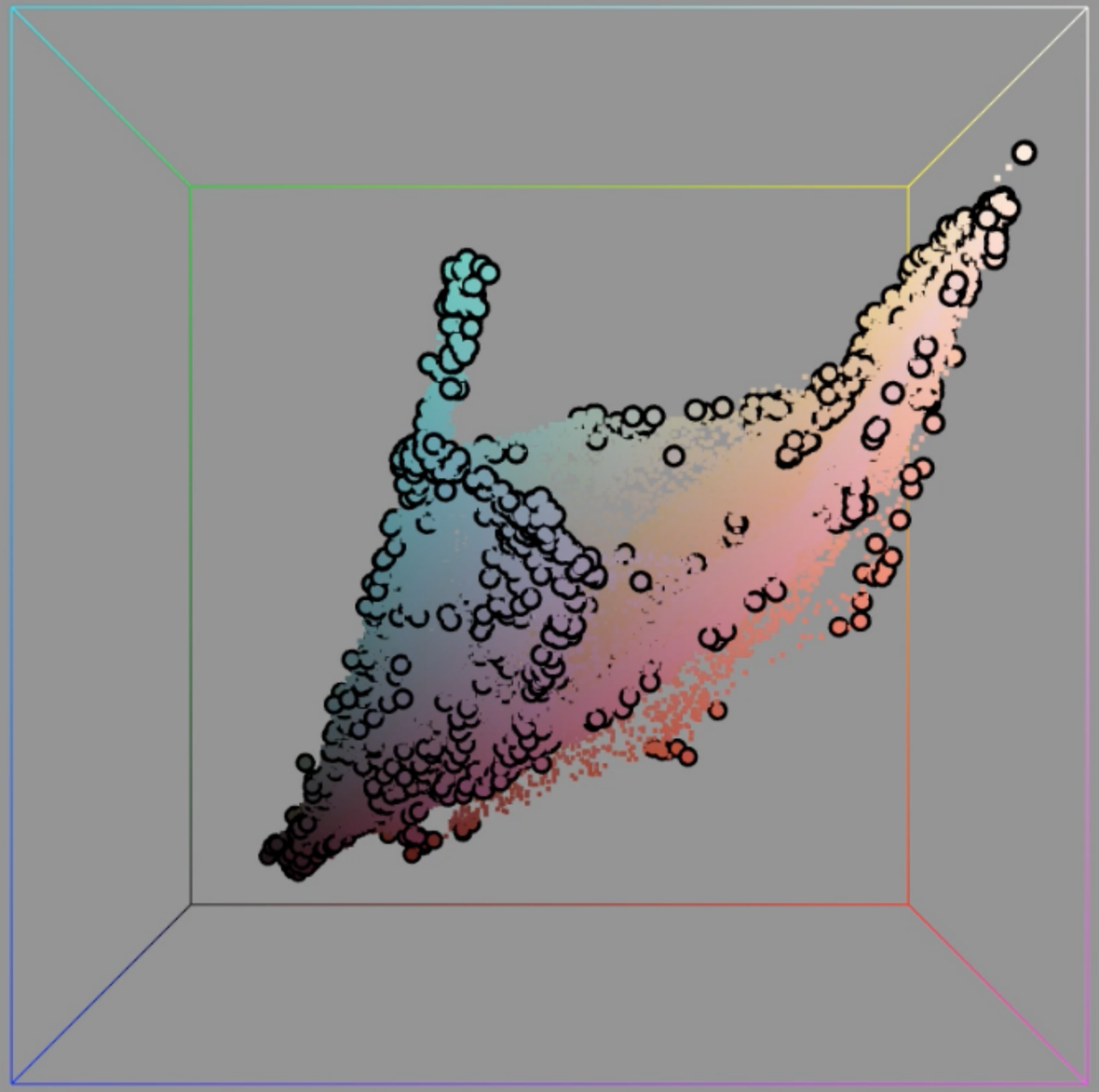


W_{RGBXY}

image

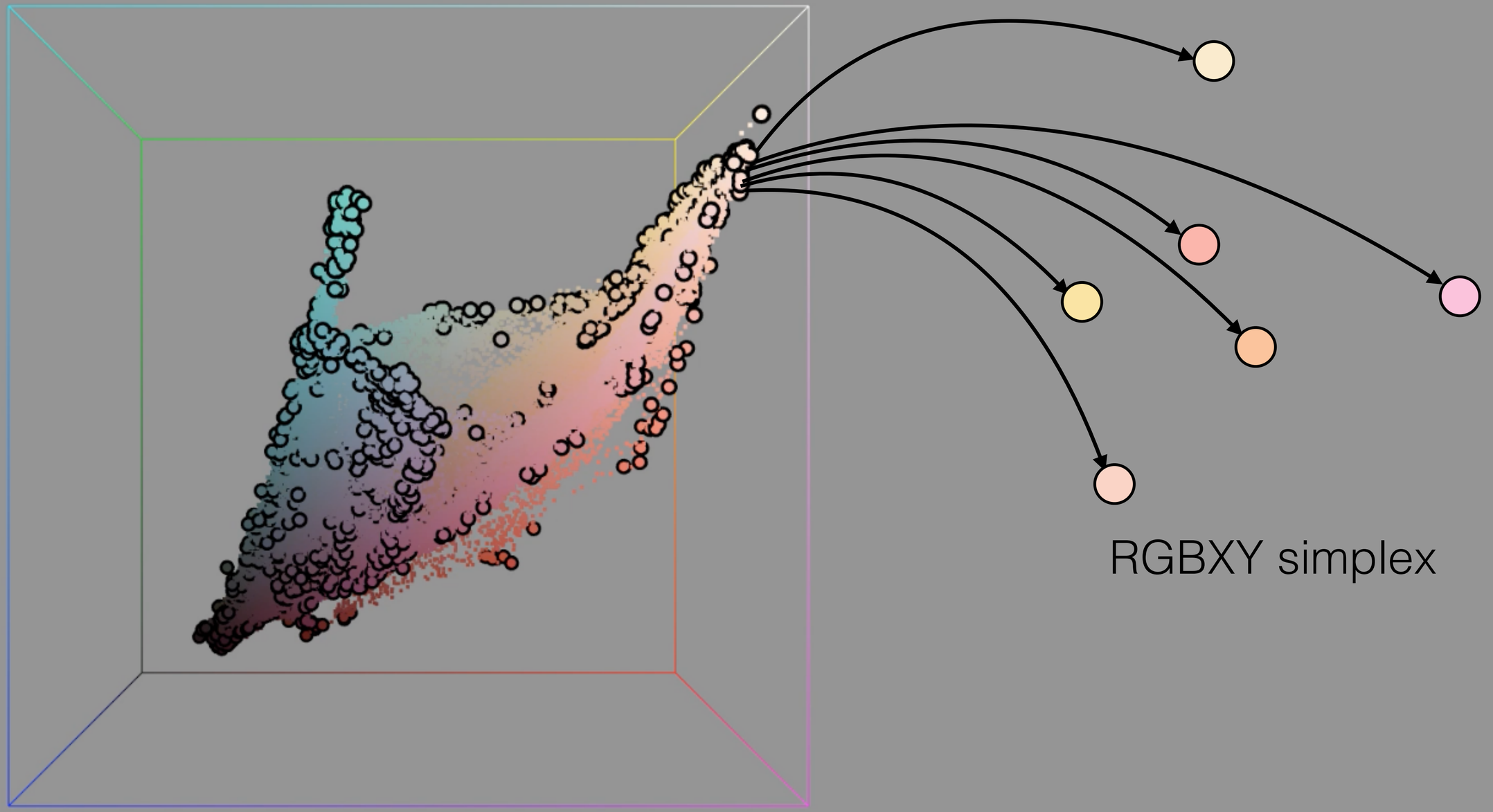


Delaunay Tessellation in RGBXY space



Extract barycentric mixing weights $\mathbf{W}_{\mathbf{RGBXY}}$

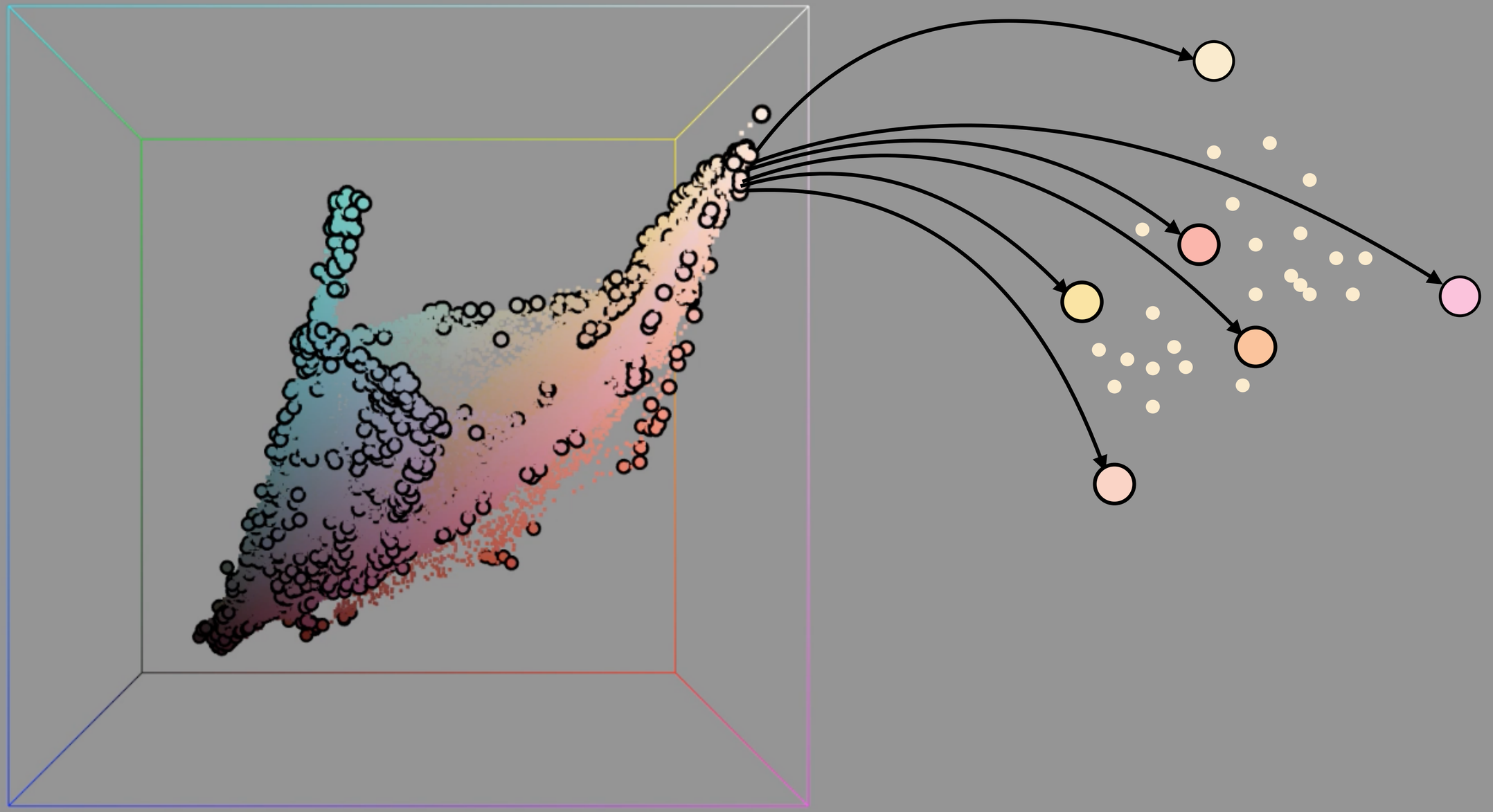
Delaunay Tessellation in RGBXY space



RGBXY simplex

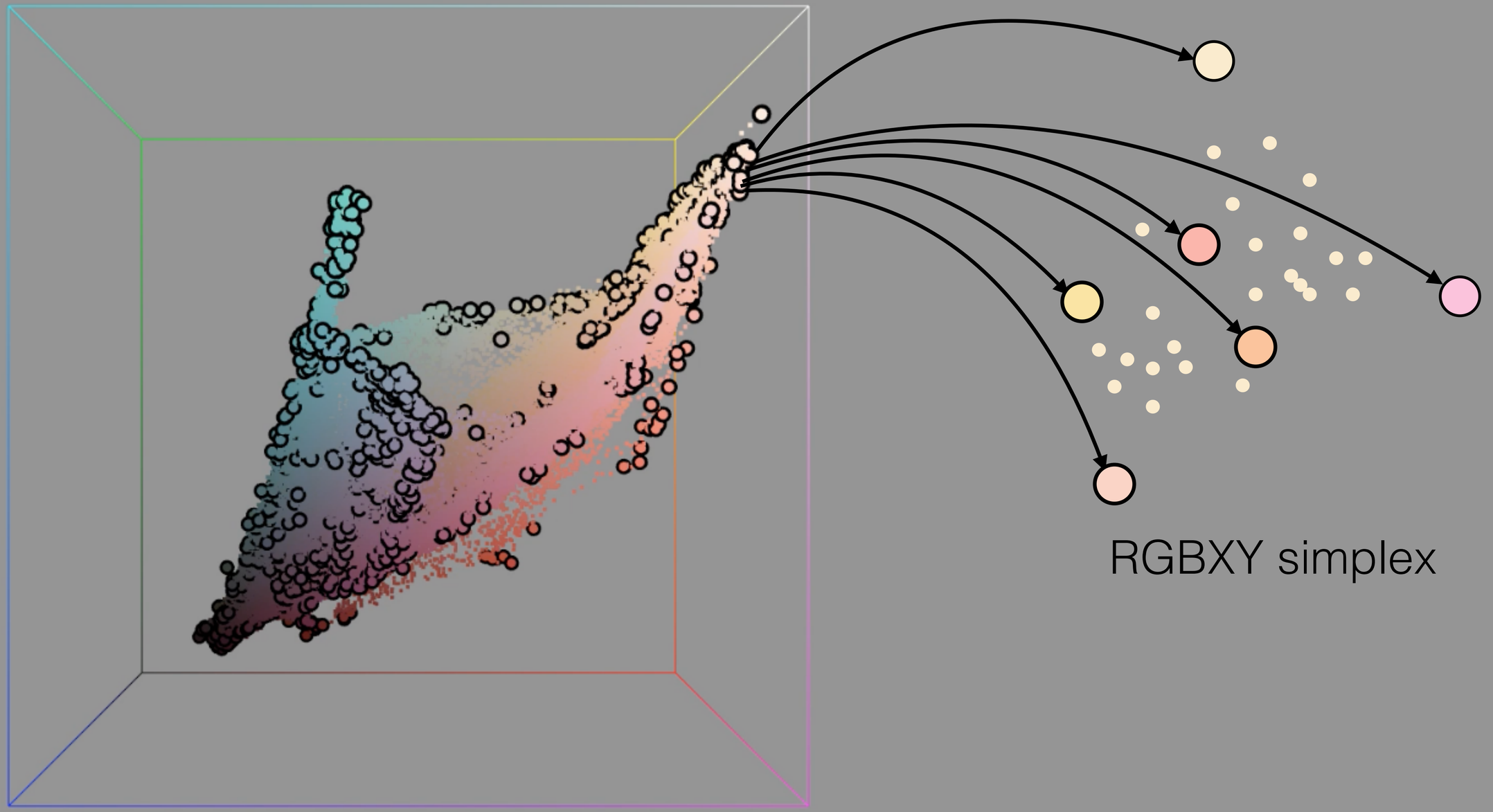
Extract barycentric mixing weights $\mathbf{W}_{\mathbf{RGBXY}}$

Delaunay Tessellation in RGBXY space



Extract barycentric mixing weights $\mathbf{W}_{\mathbf{RGBXY}}$

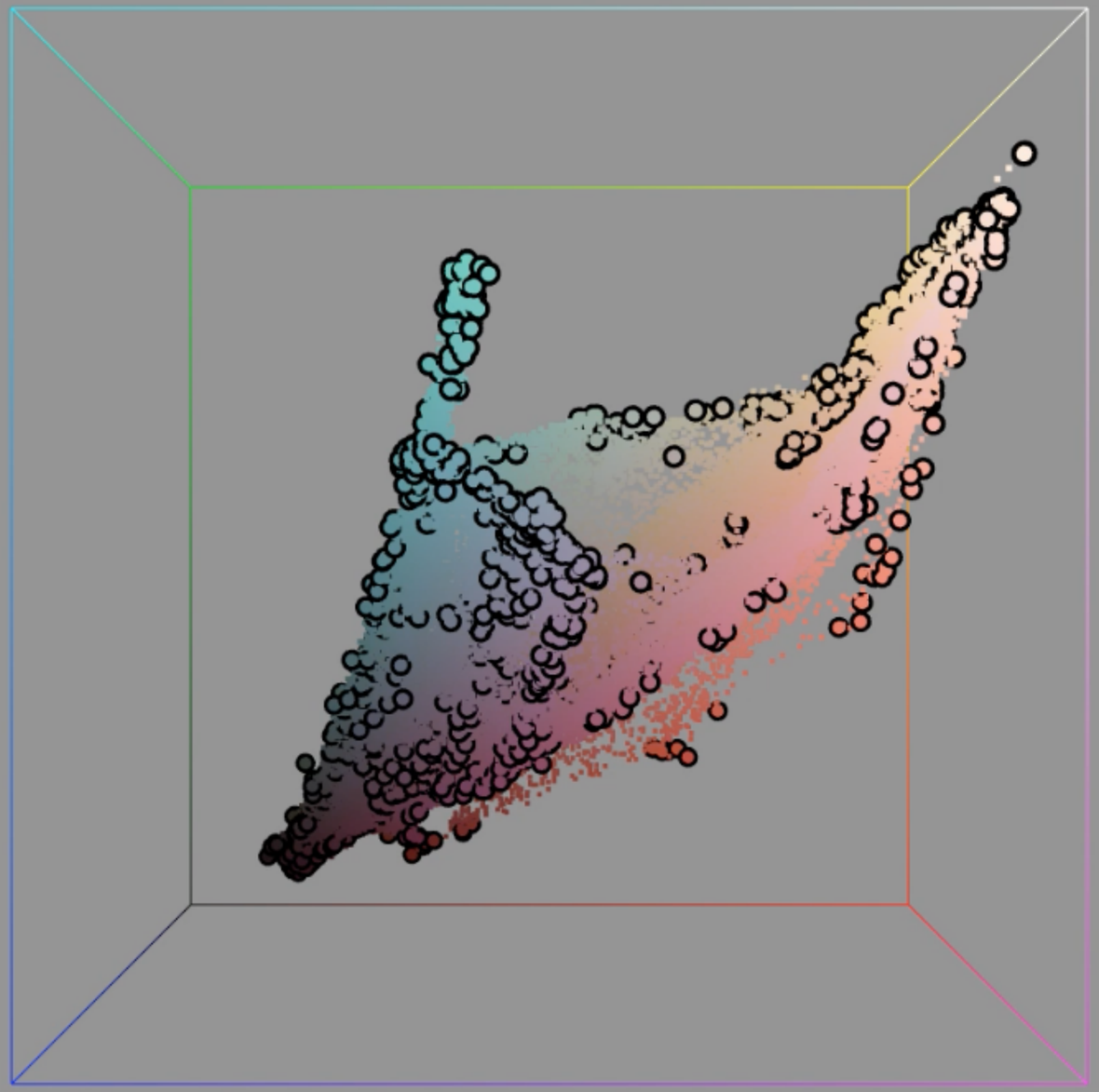
Delaunay Tessellation in RGBXY space

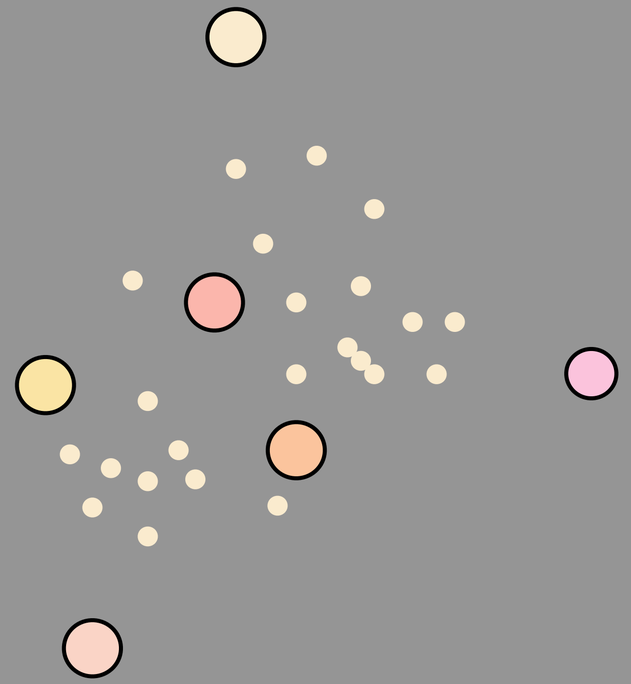


RGBXY simplex

Extract barycentric mixing weights $\mathbf{W}_{\mathbf{RGBXY}}$

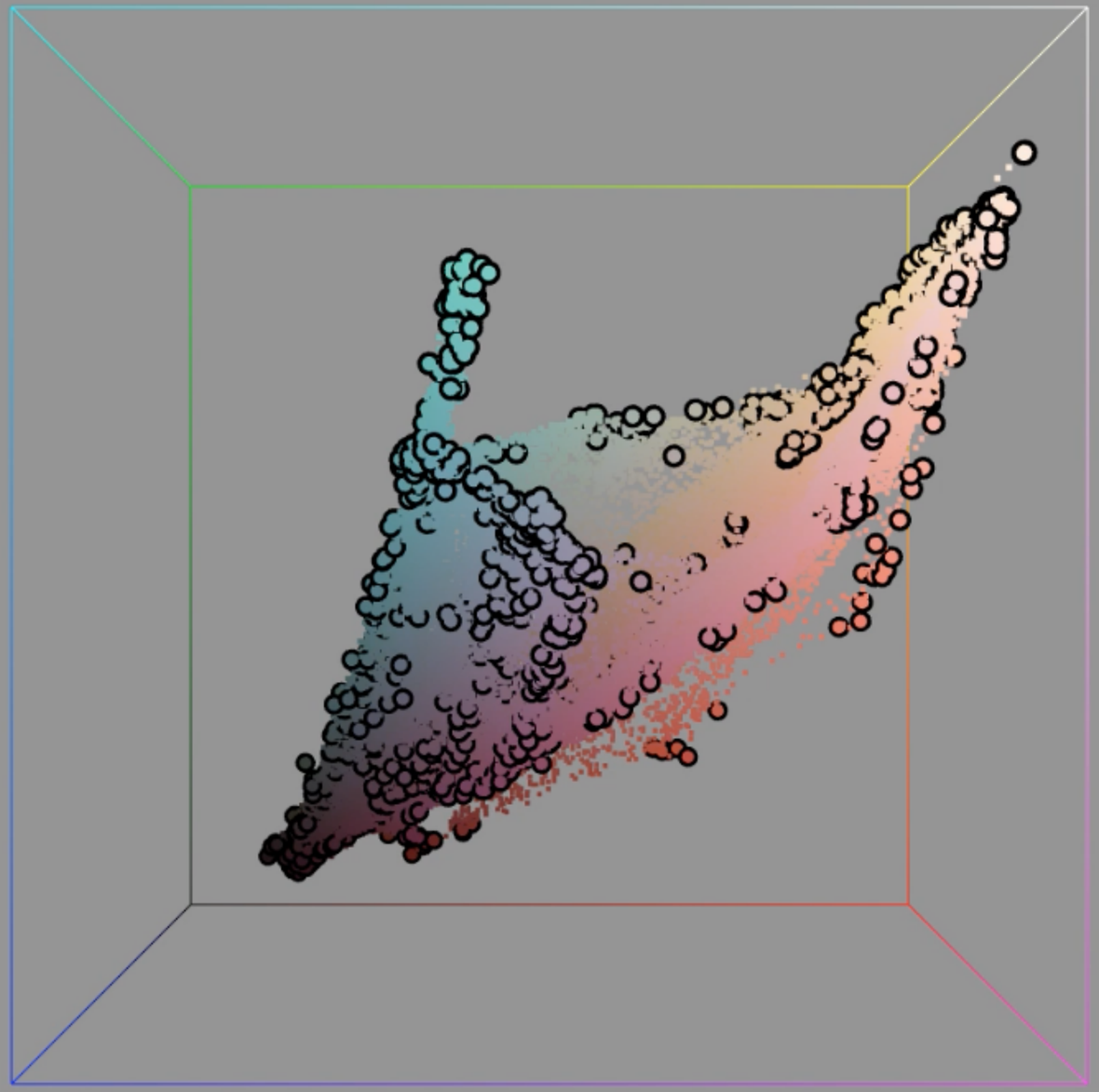
Delaunay Tessellation in RGBXY space

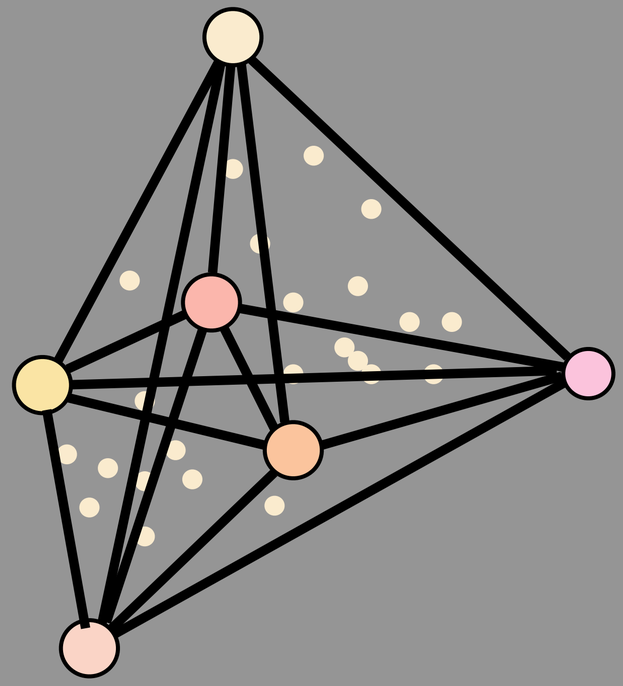




Extract barycentric mixing weights $\mathbf{W}_{\mathbf{RGBXY}}$

Delaunay Tessellation in RGBXY space





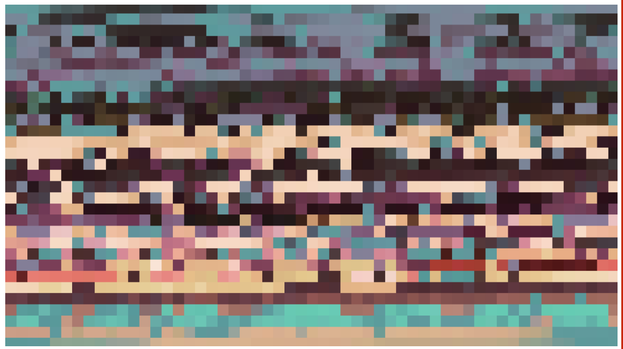
RGBXY simplex

Extract barycentric mixing weights $\mathbf{W}_{\mathbf{RGBXY}}$

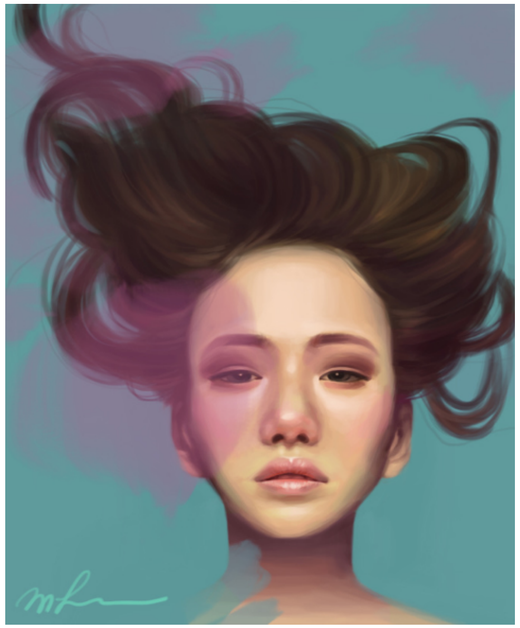
Two-level decomposition

RGB palette

RGBXY vertices
(projected to RGB)



image

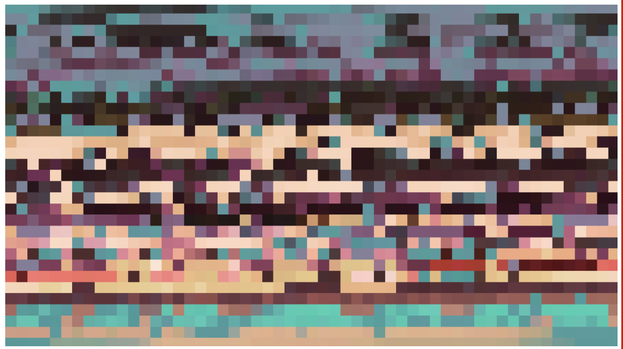


Two-level decomposition

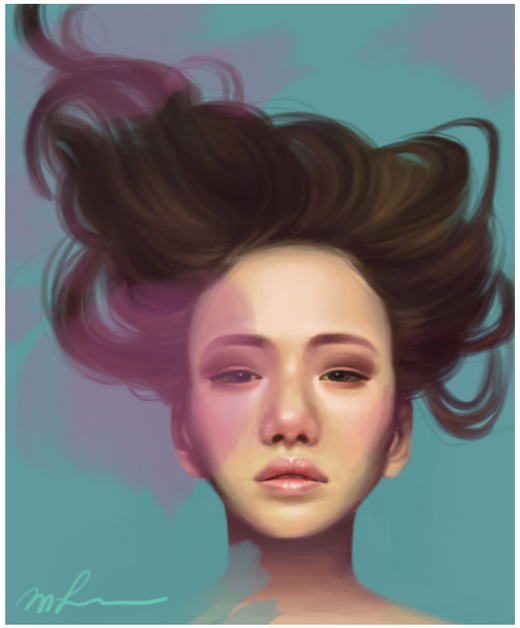
RGB palette

W_{RGB}

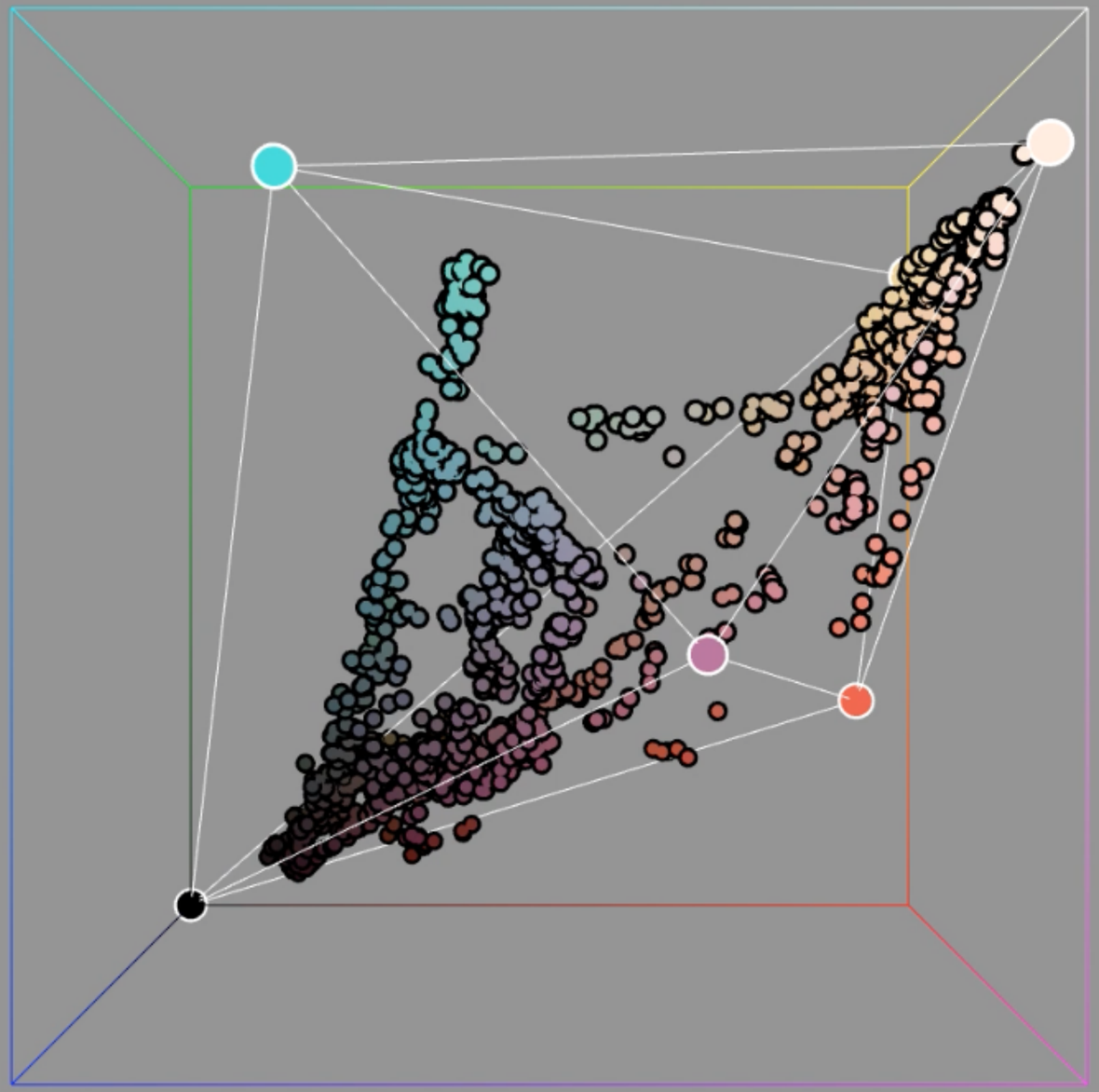
RGBXY vertices
(projected to RGB)



image



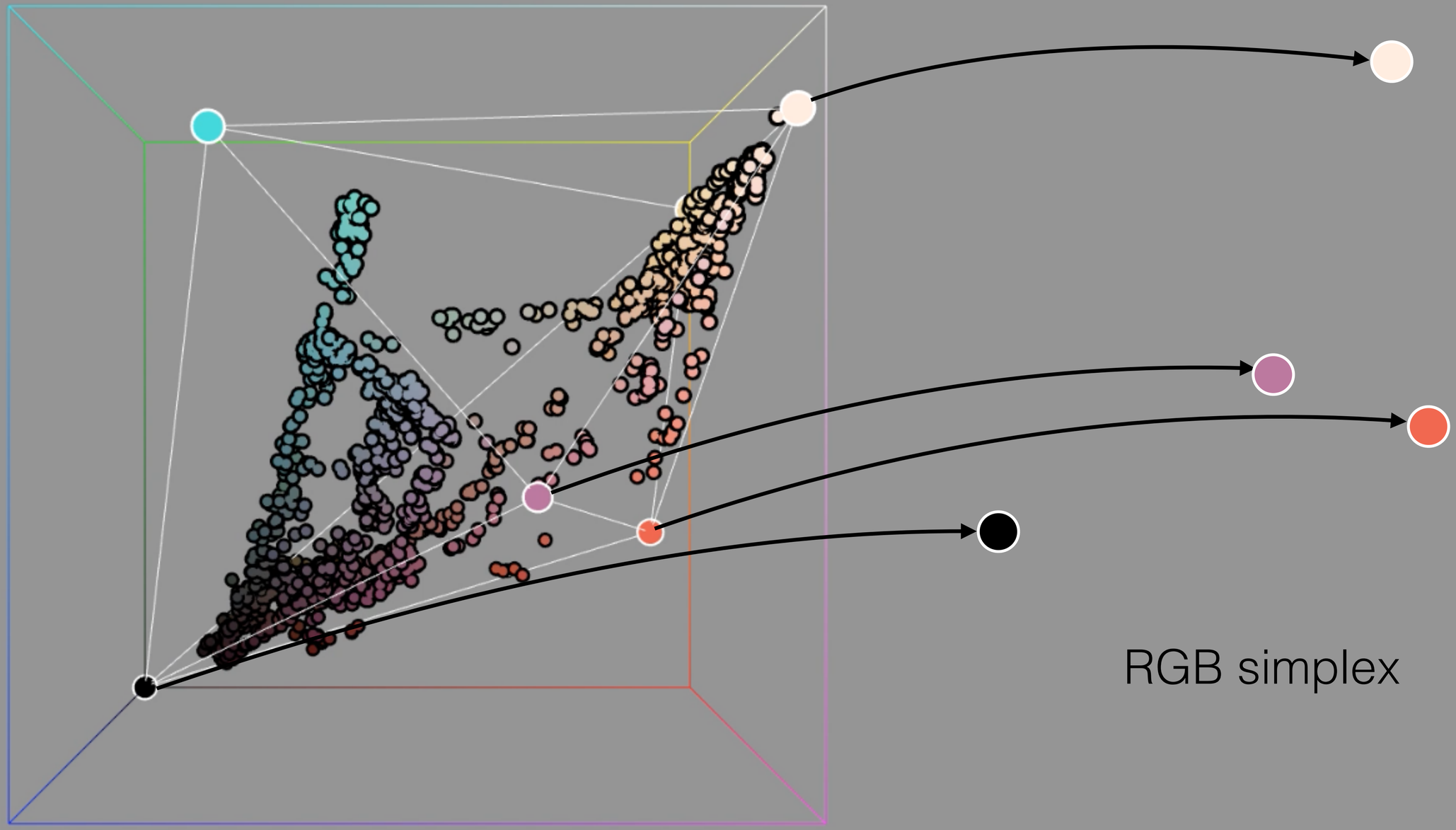
Tessellation in RGB space



RGB simplex

Extract barycentric mixing weights $\mathbf{W_{RGB}}$

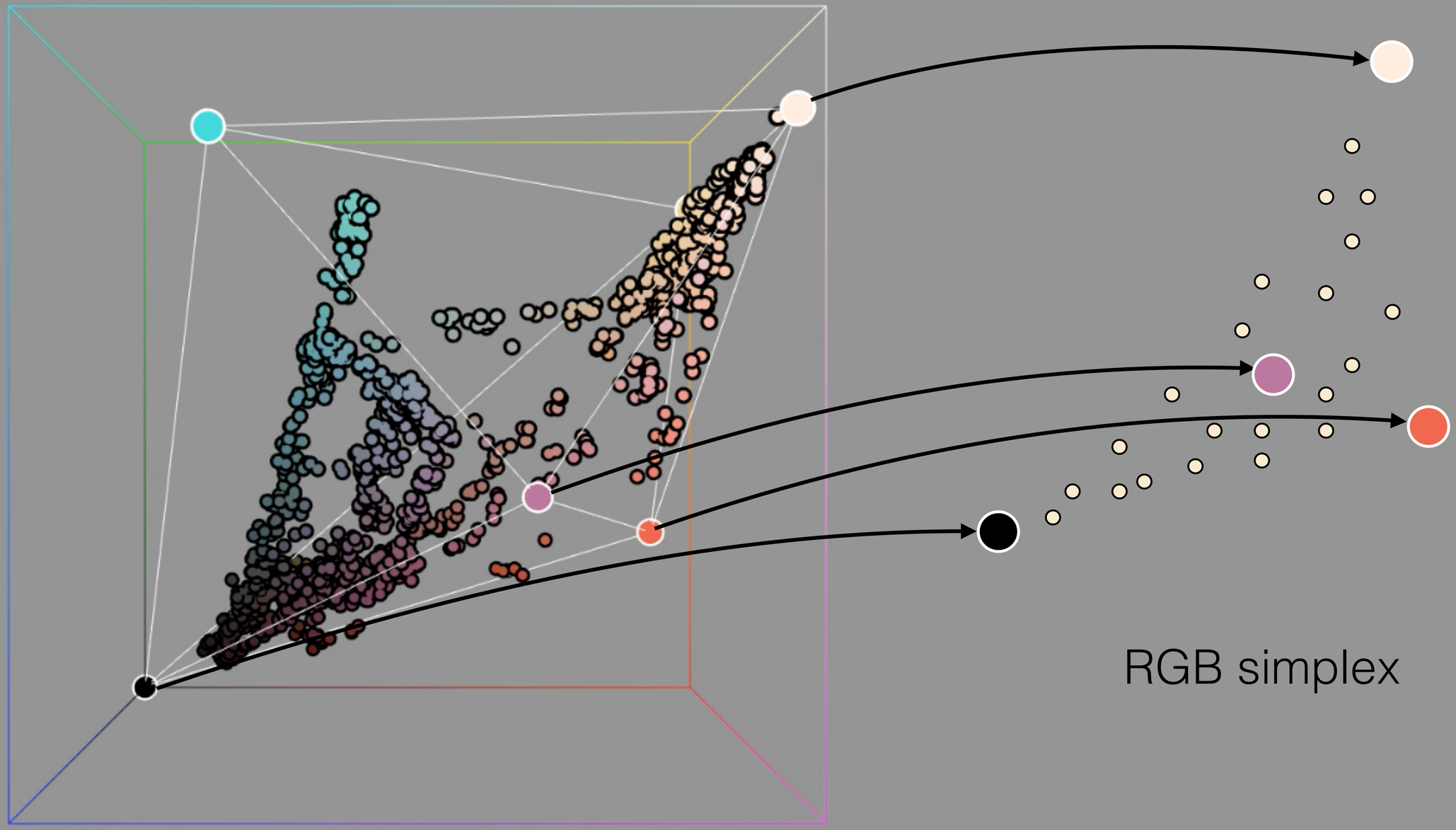
Tessellation in RGB space



RGB simplex

Extract barycentric mixing weights $\mathbf{W_{RGB}}$

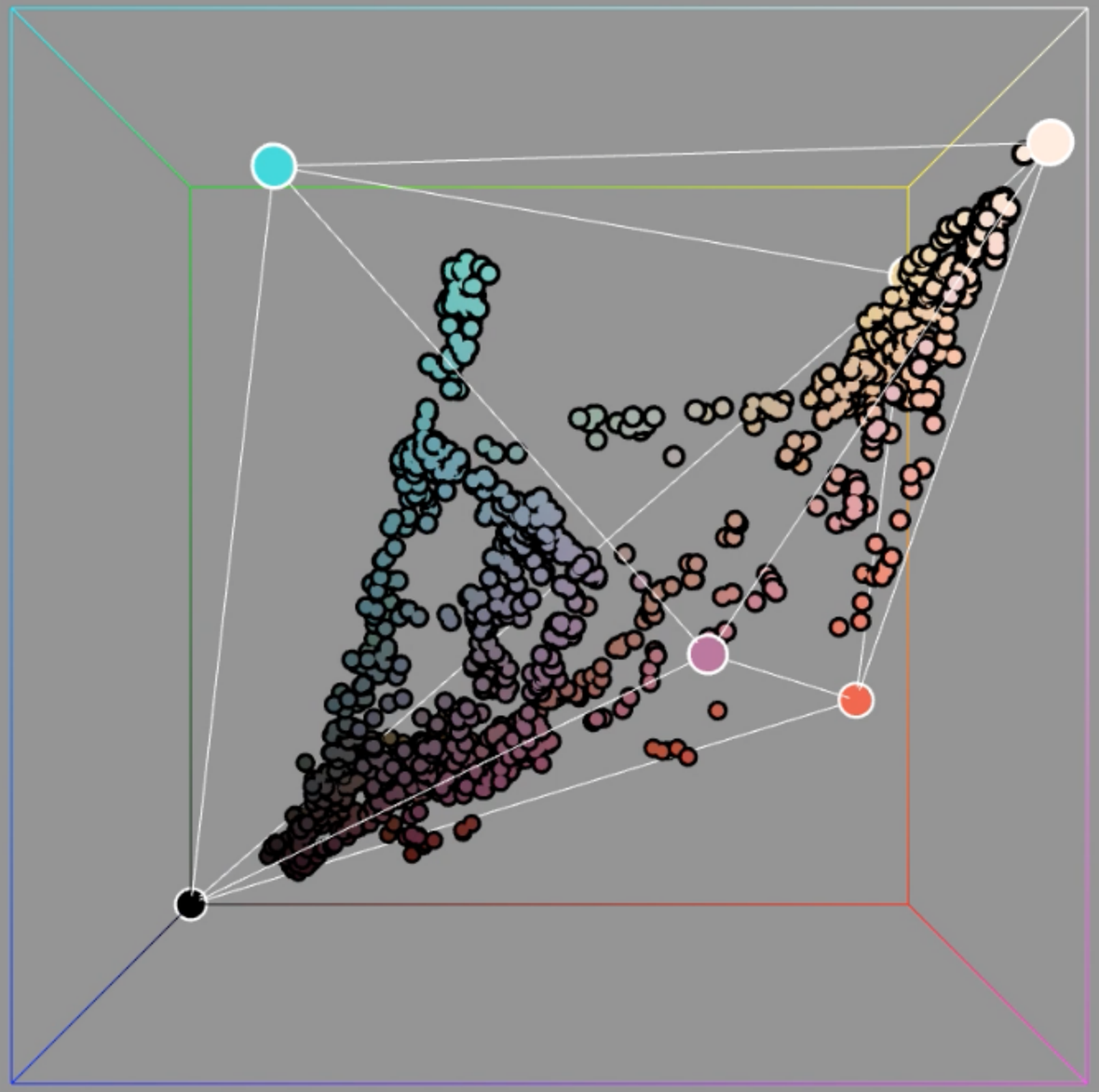
Tessellation in RGB space

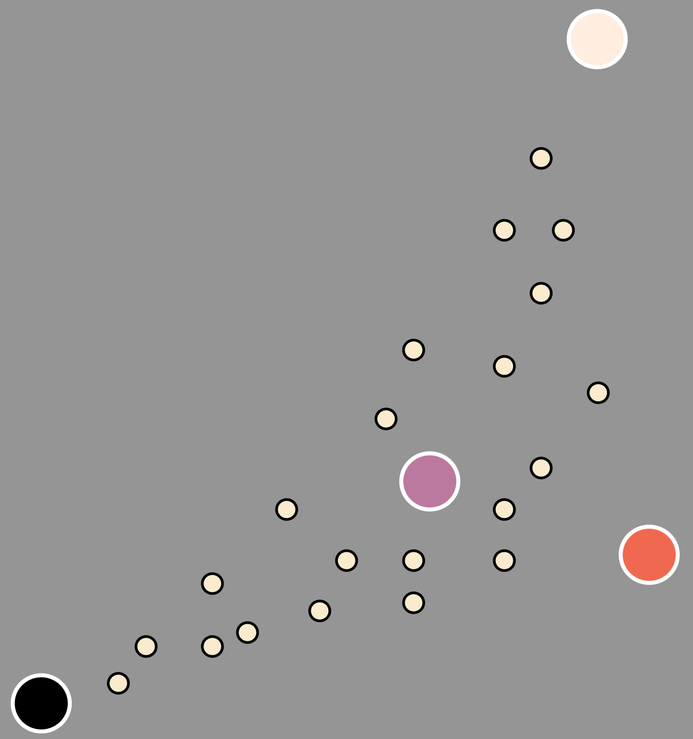


RGB simplex

Extract barycentric mixing weights $\mathbf{W_{RGB}}$

Tessellation in RGB space

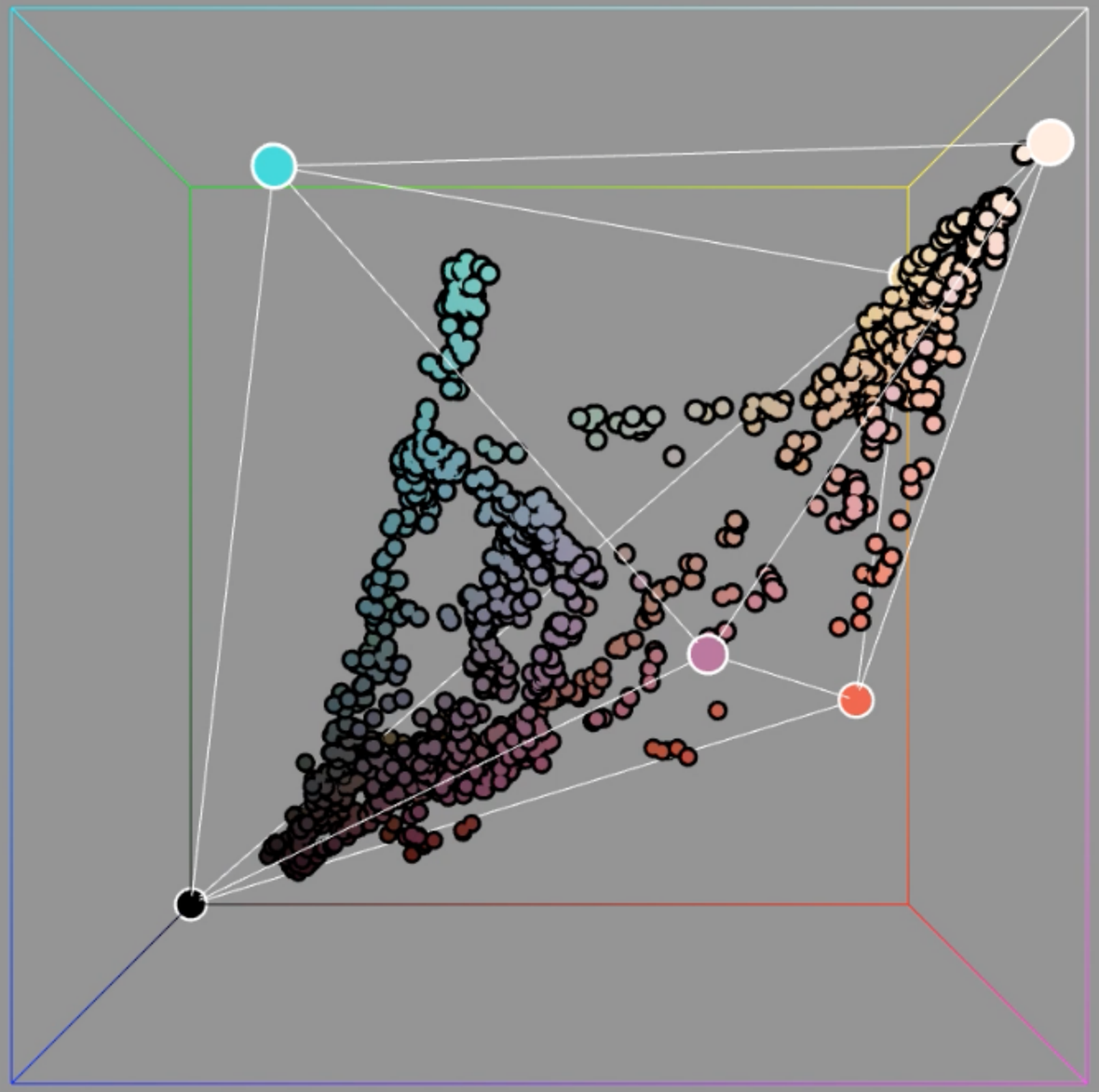


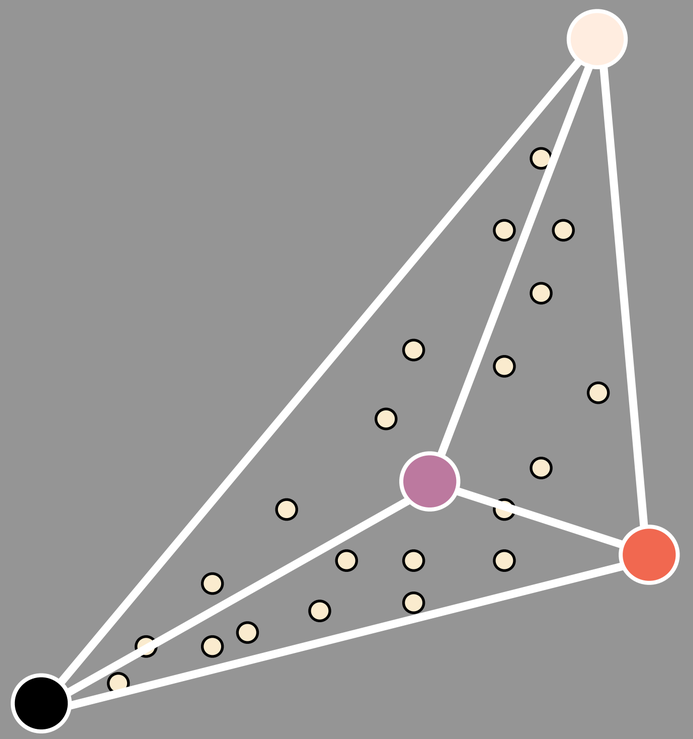


RGB simplex

Extract barycentric mixing weights $\mathbf{W_{RGB}}$

Tessellation in RGB space





RGB simplex

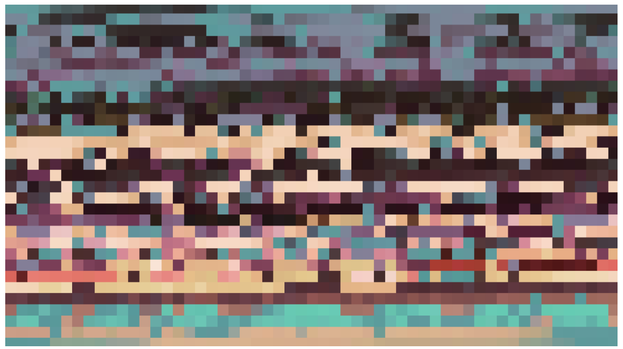
Extract barycentric mixing weights $\mathbf{W_{RGB}}$

Two-level decomposition

RGB palette

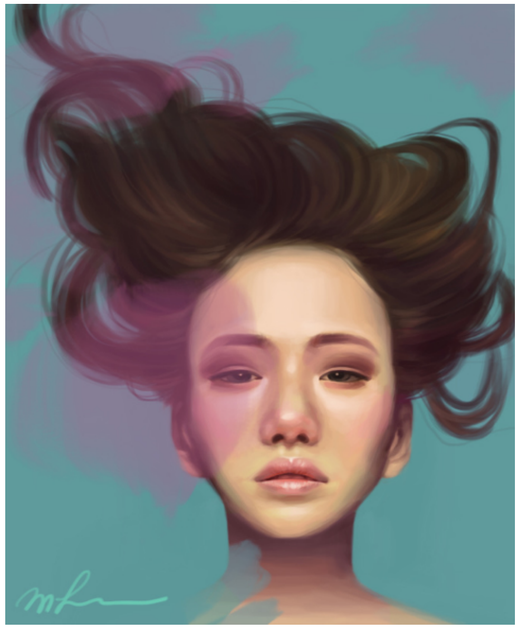
W_{RGB}

RGBXY vertices
(projected to RGB)



W_{RGBXY}

image

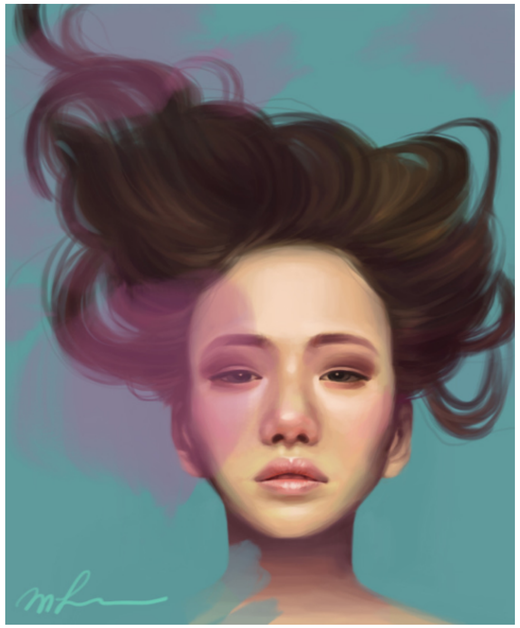


Two-level decomposition

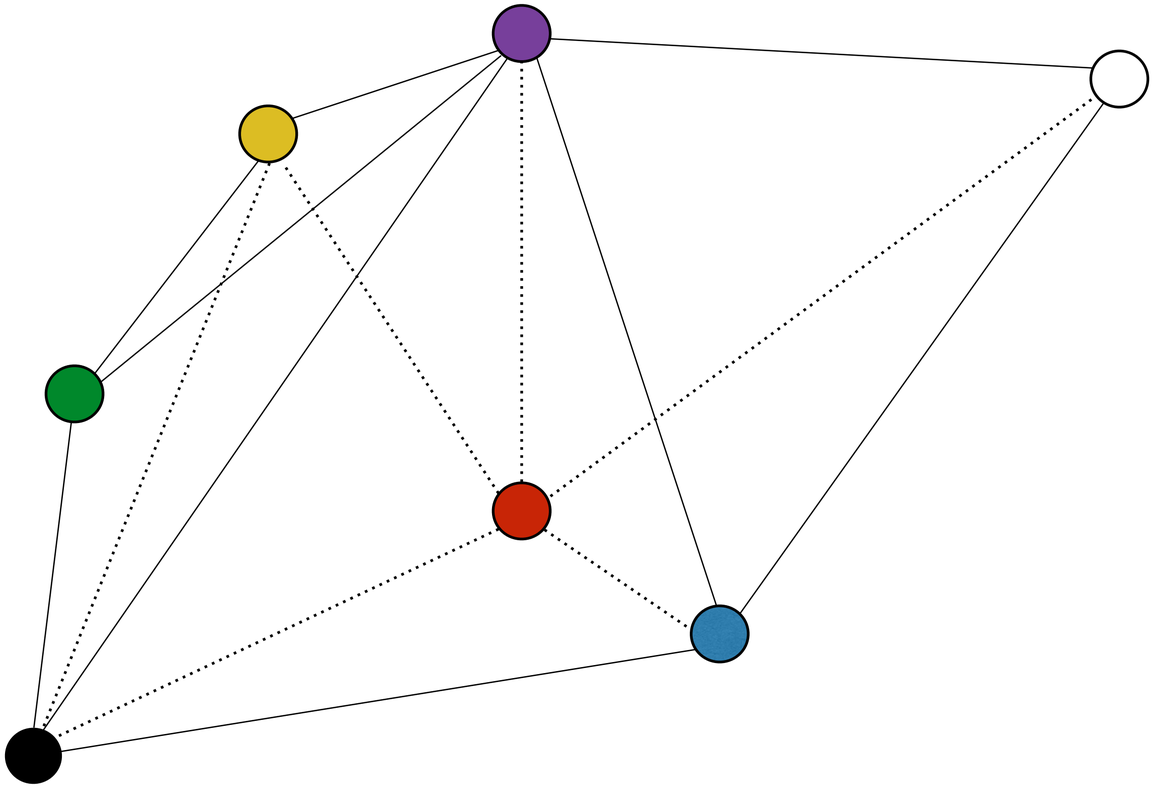
RGB palette

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

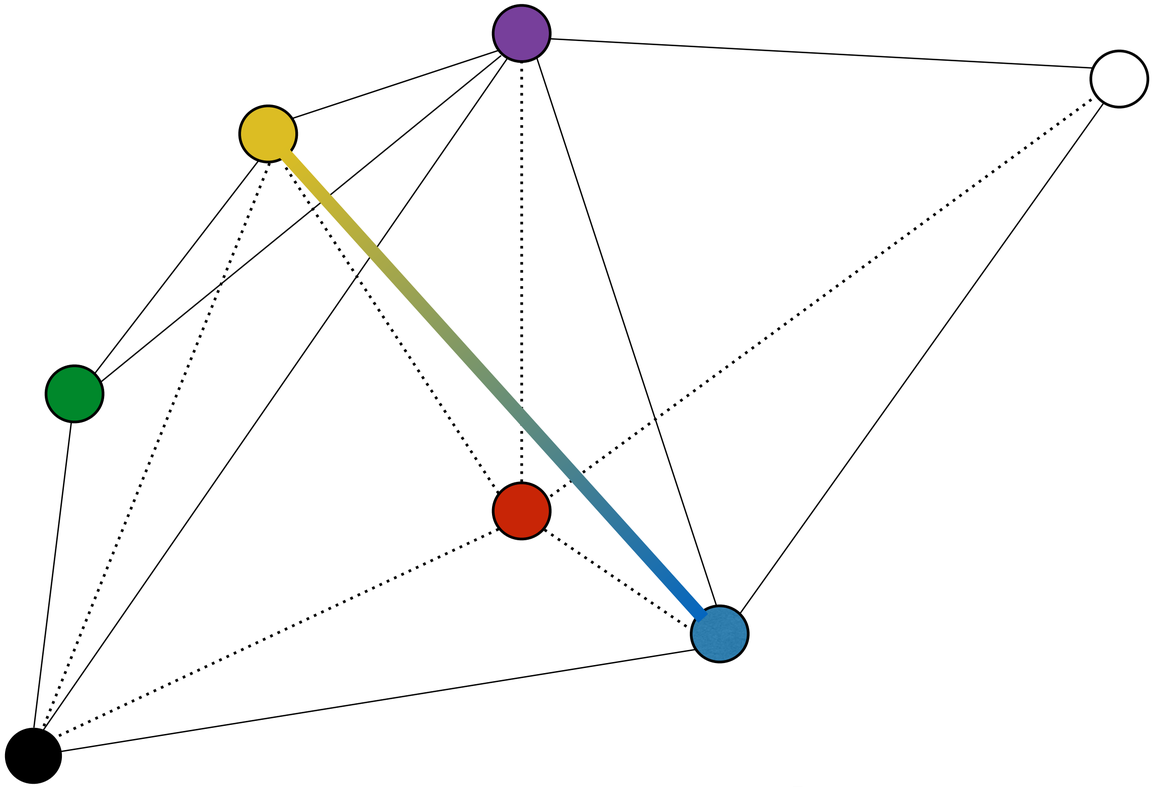
image



Tessellation in RGB space

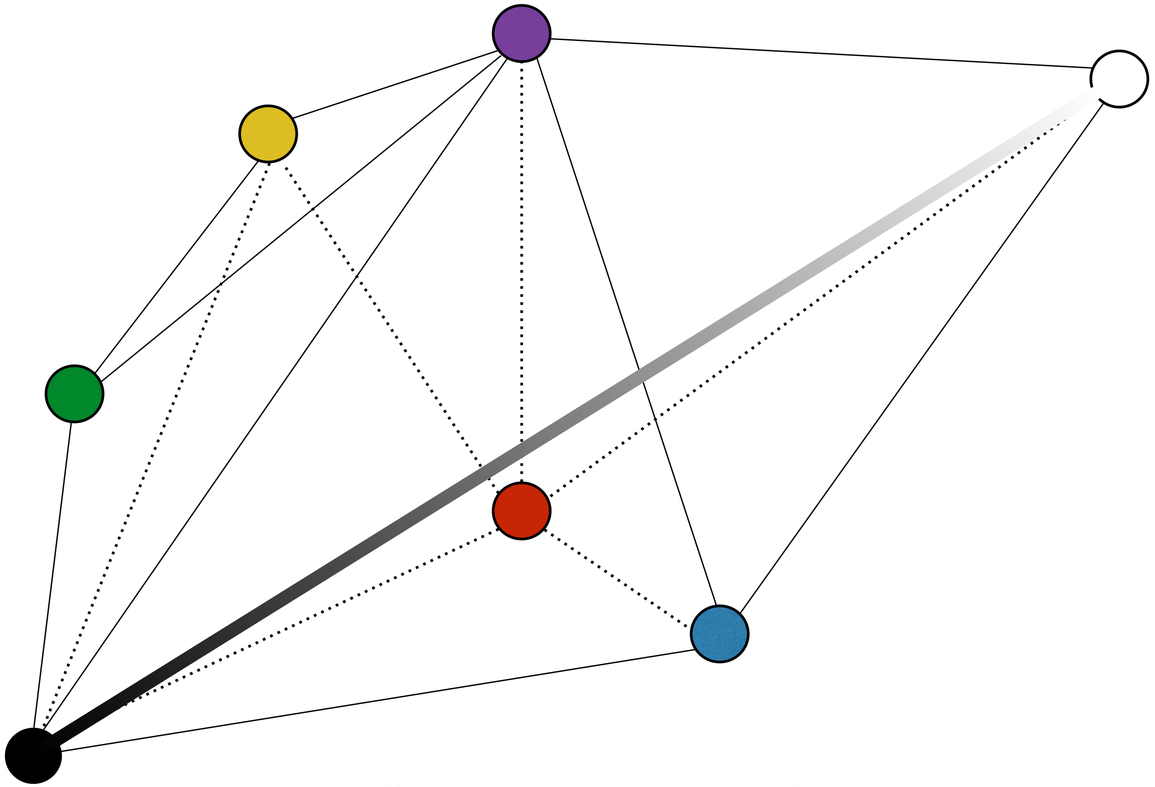


Tessellation in RGB space

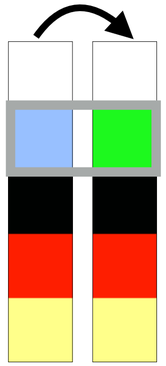


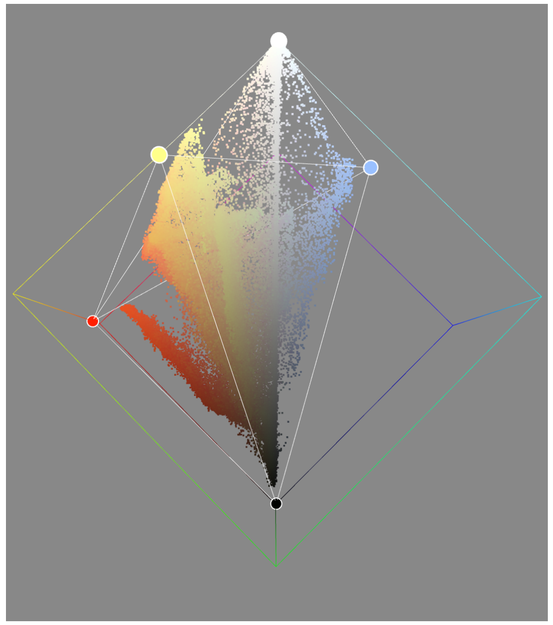
Delaunay tessellation

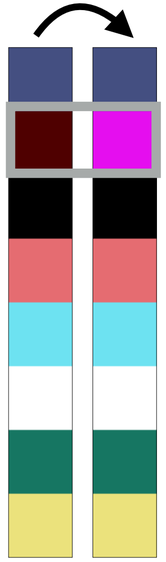
Tessellation in RGB space

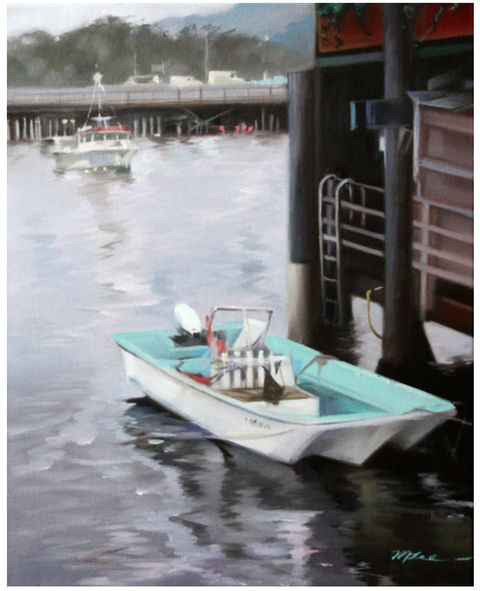


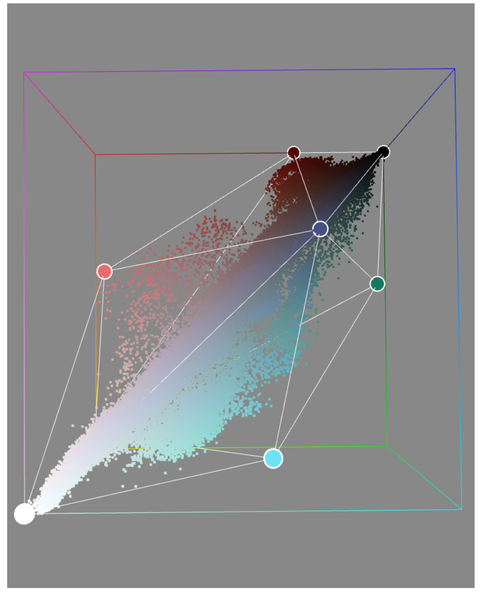
Star tessellation

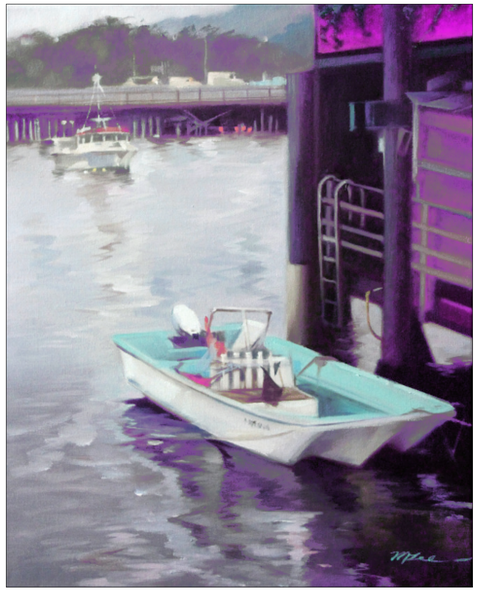


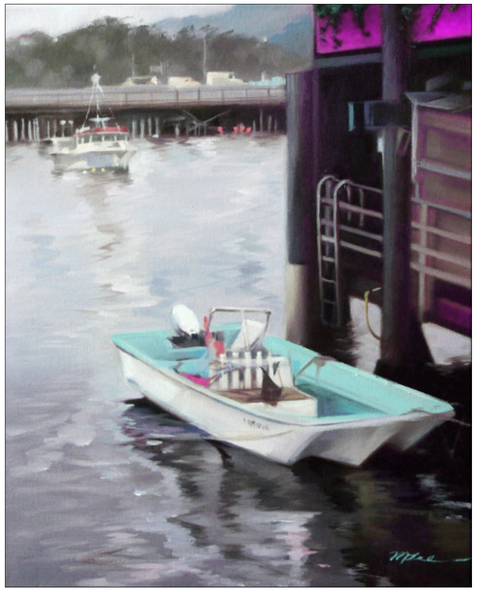




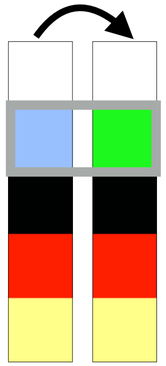


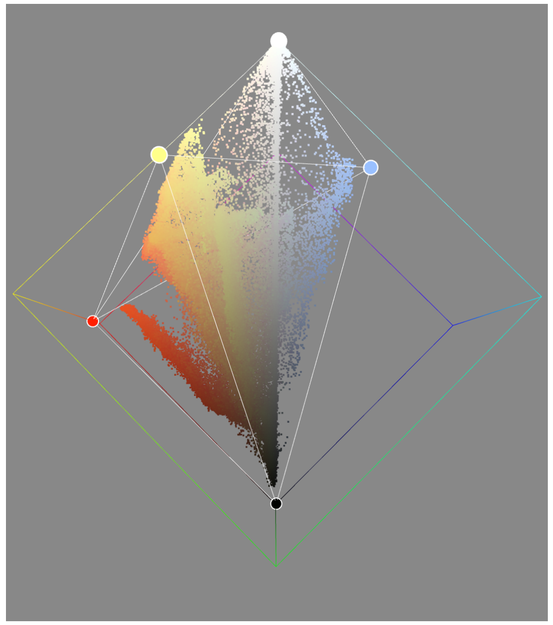


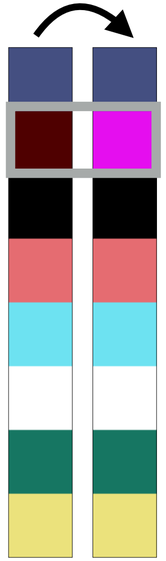


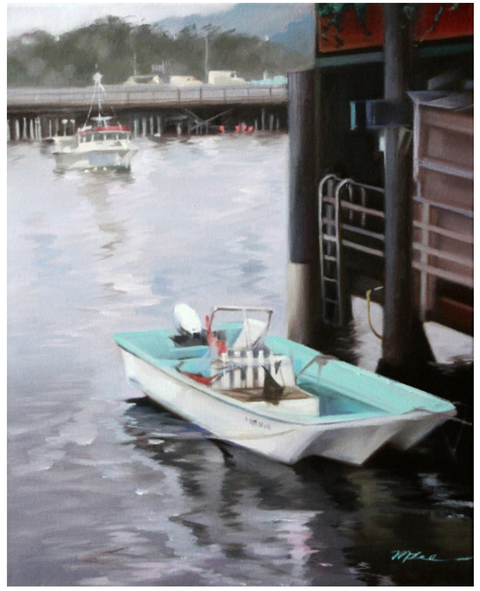


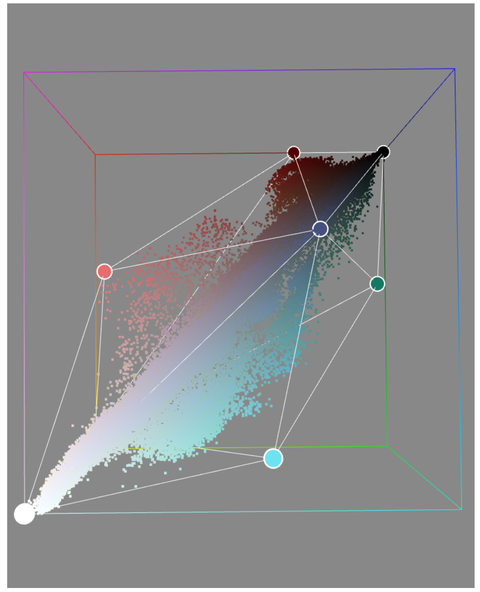
Original Color distributions Delaunay Star

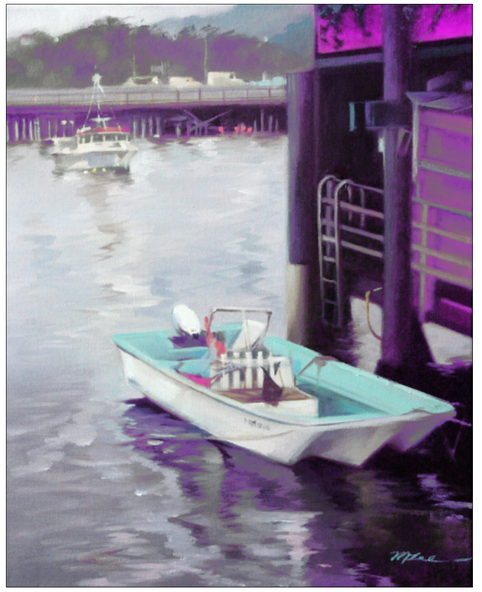


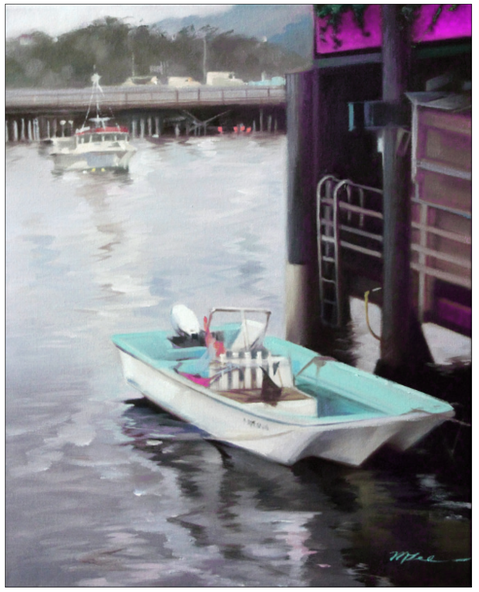




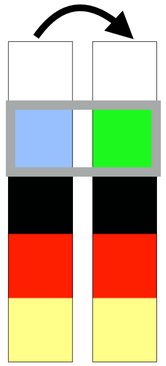


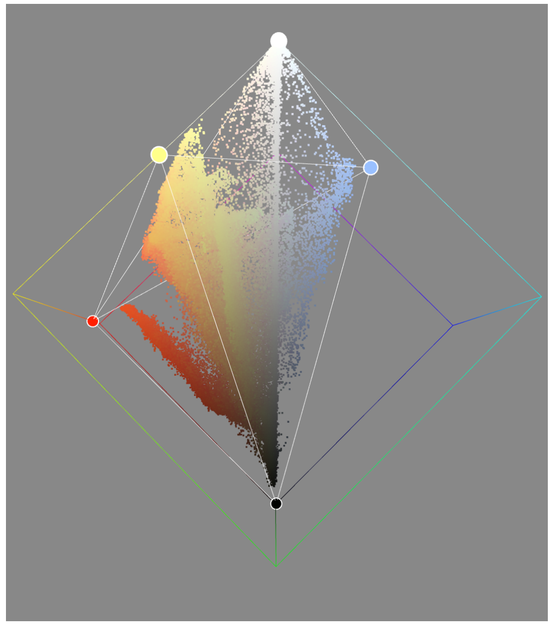


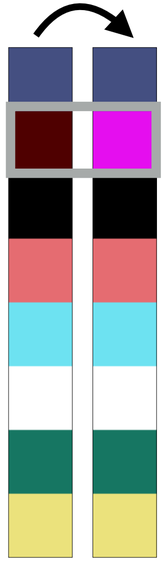


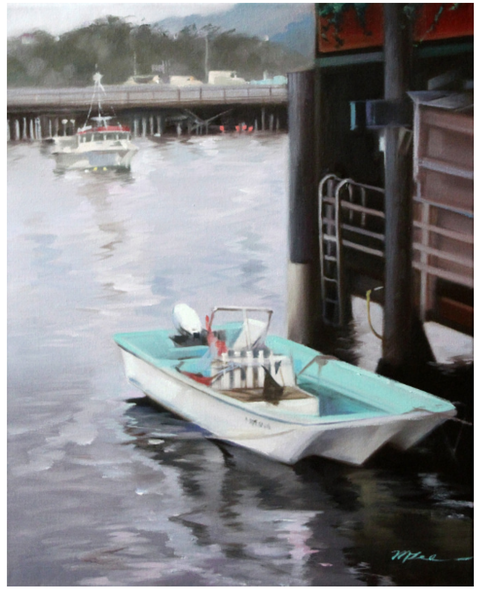


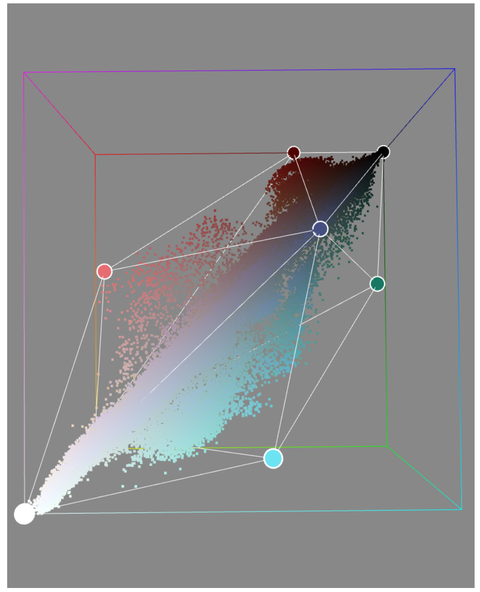
Original Color distributions Delaunay Star

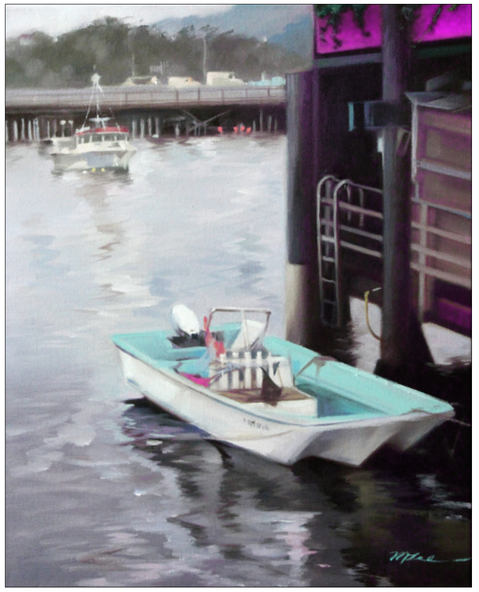












Original Color distributions Delaunay Star

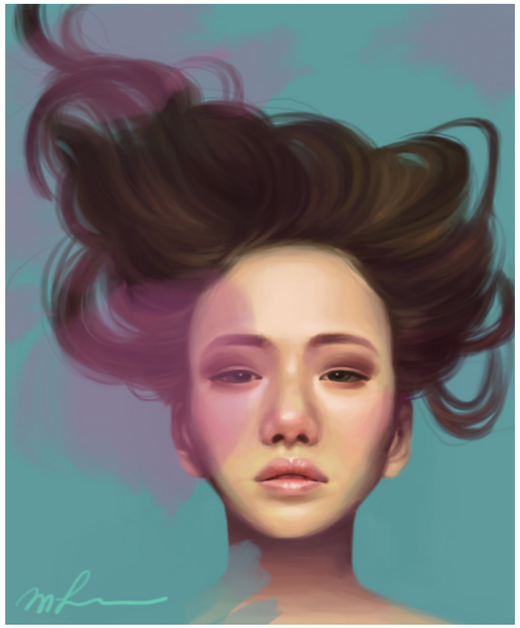
Two-level decomposition

RGB palette

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

Fixed

image



Two-level decomposition

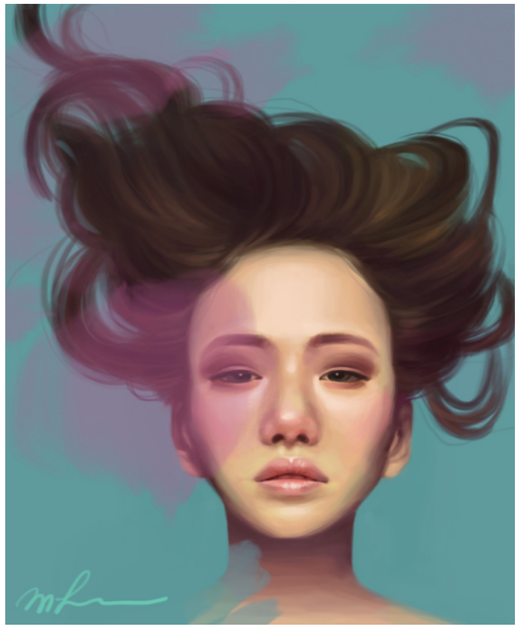
RGB palette

Palette updates

Fixed

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

image



Two-level decomposition

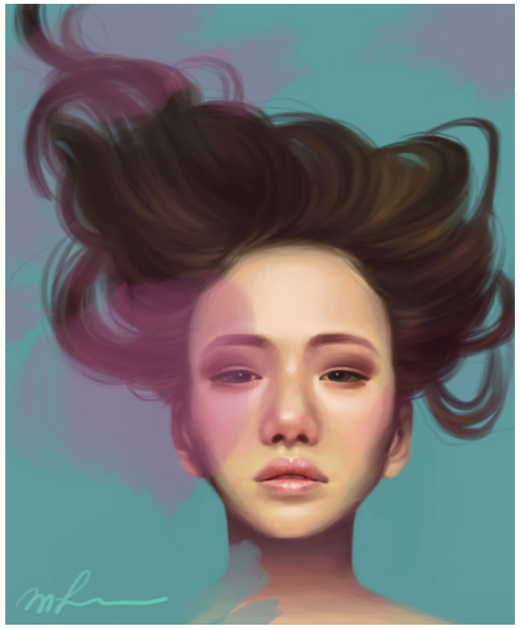
RGB palette

Palette updates

Fixed

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

image



Two-level decomposition

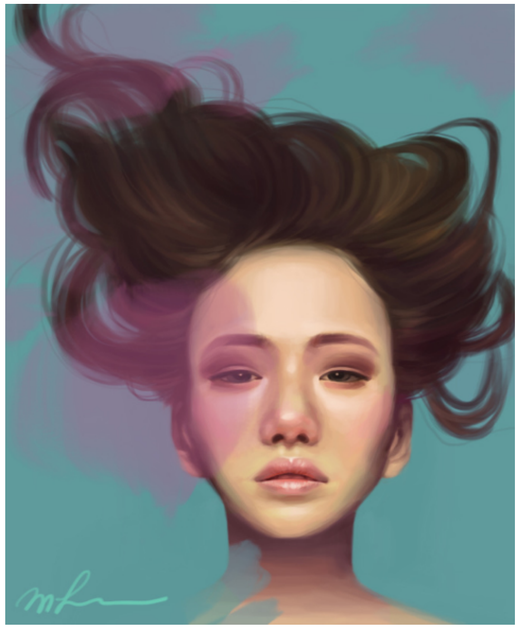
RGB palette

Palette updates

Fixed

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

image



Two-level decomposition

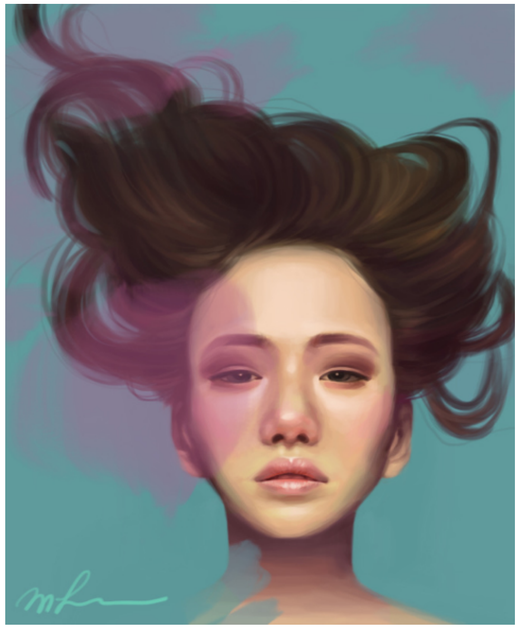
RGB palette

Palette updates

Fixed

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

image



Updating W_{RGB} is independent of image size.

Two-level decomposition

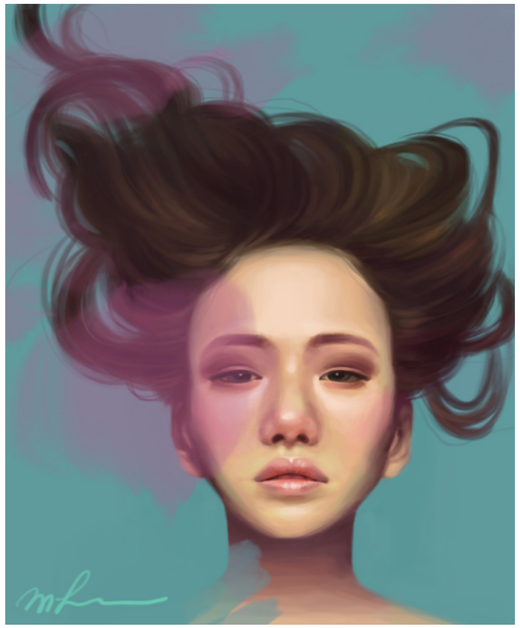
RGB palette

Palette updates

Fixed

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

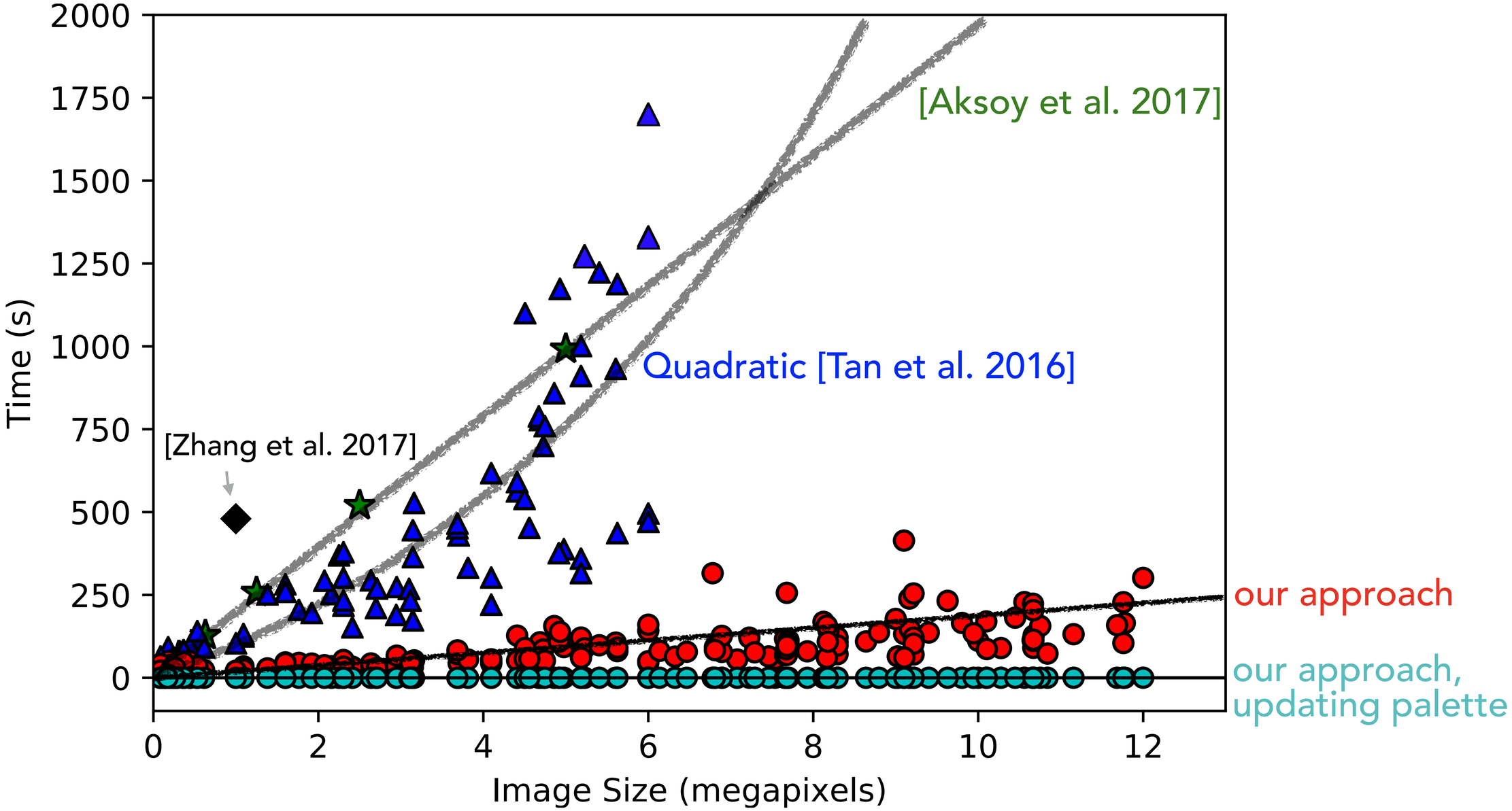
image



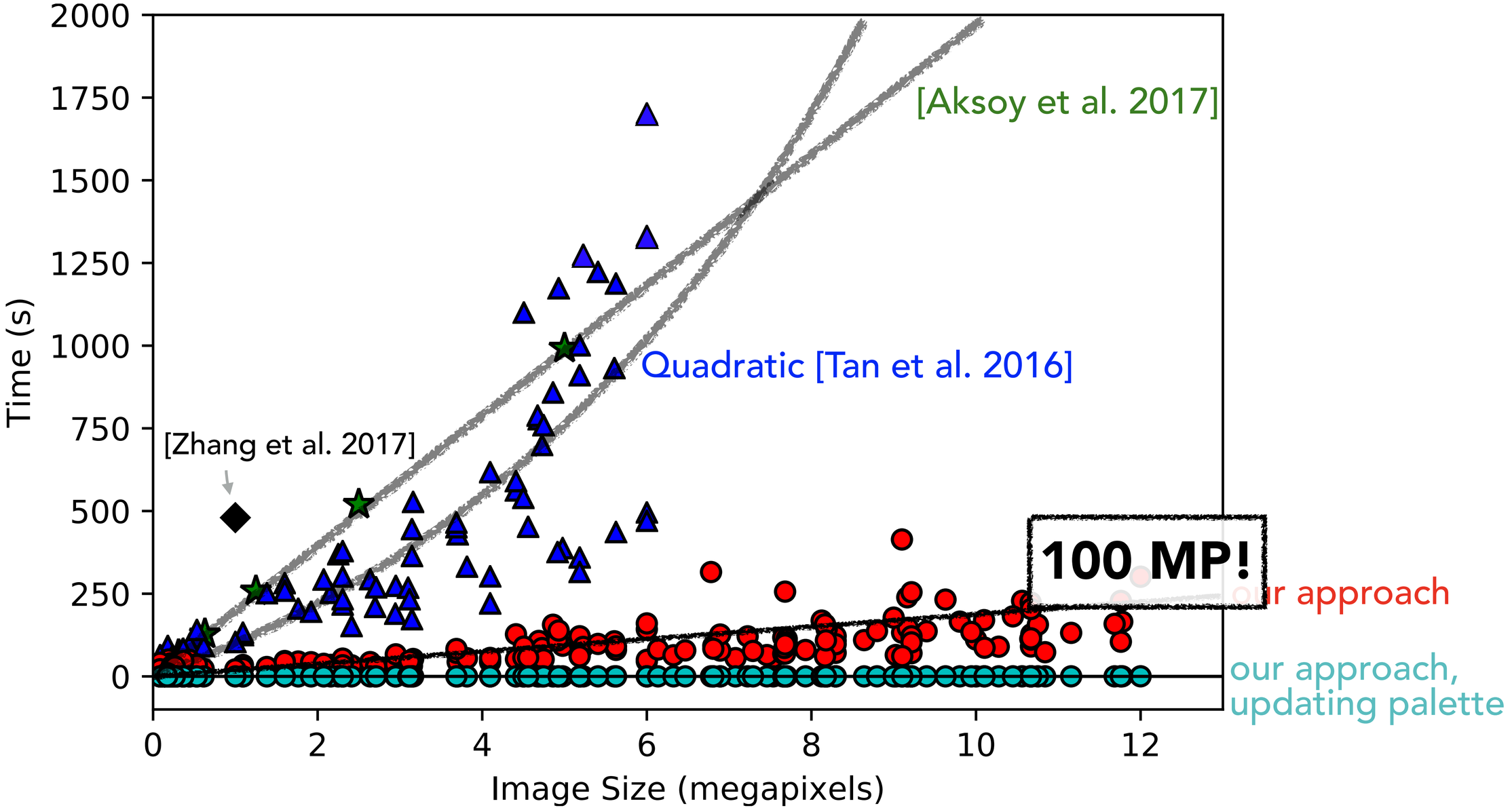
Updating W_{RGB} is independent of image size.

Other methods need to re-compute everything from scratch.

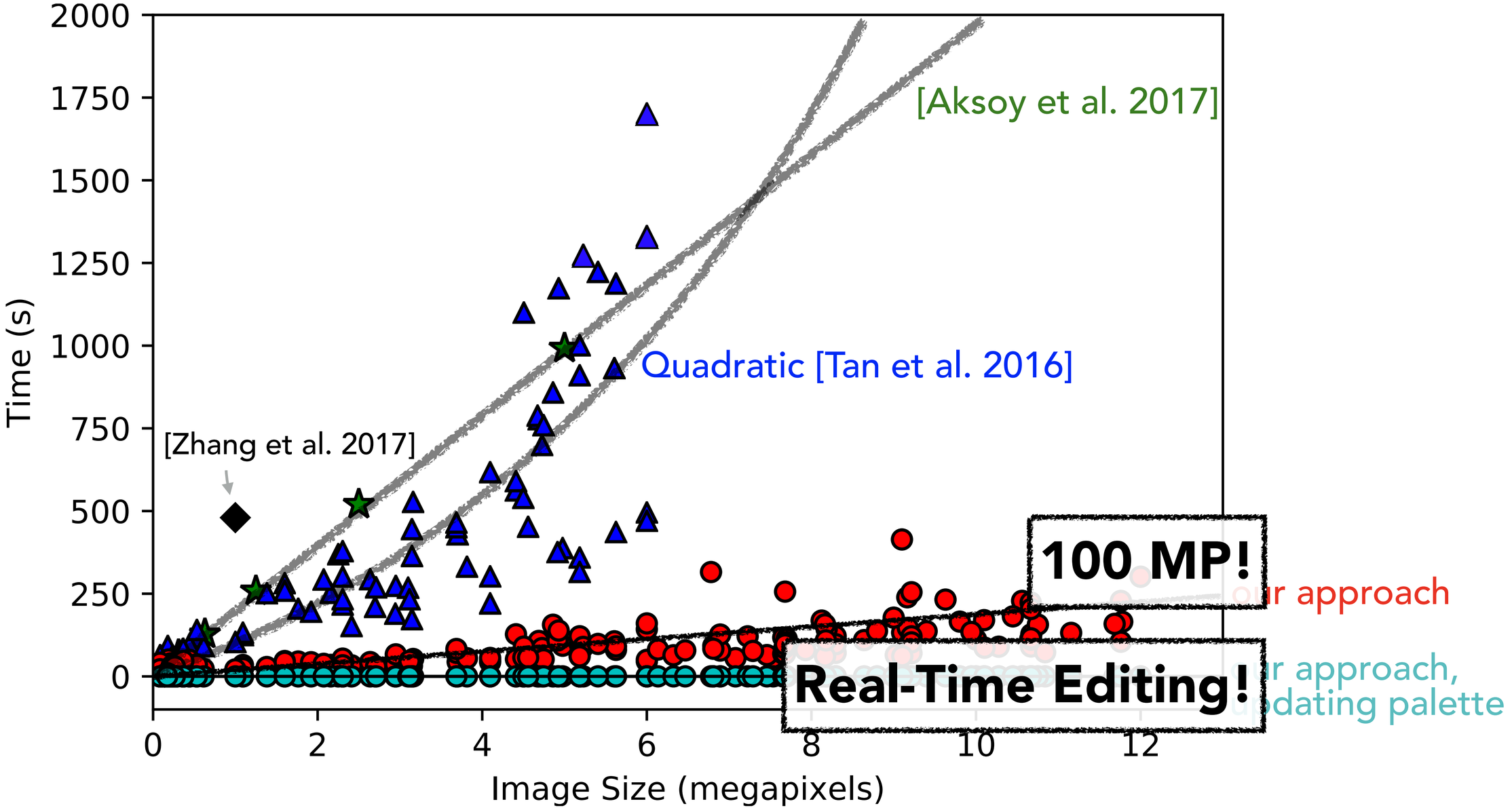
Performance



Performance



Performance



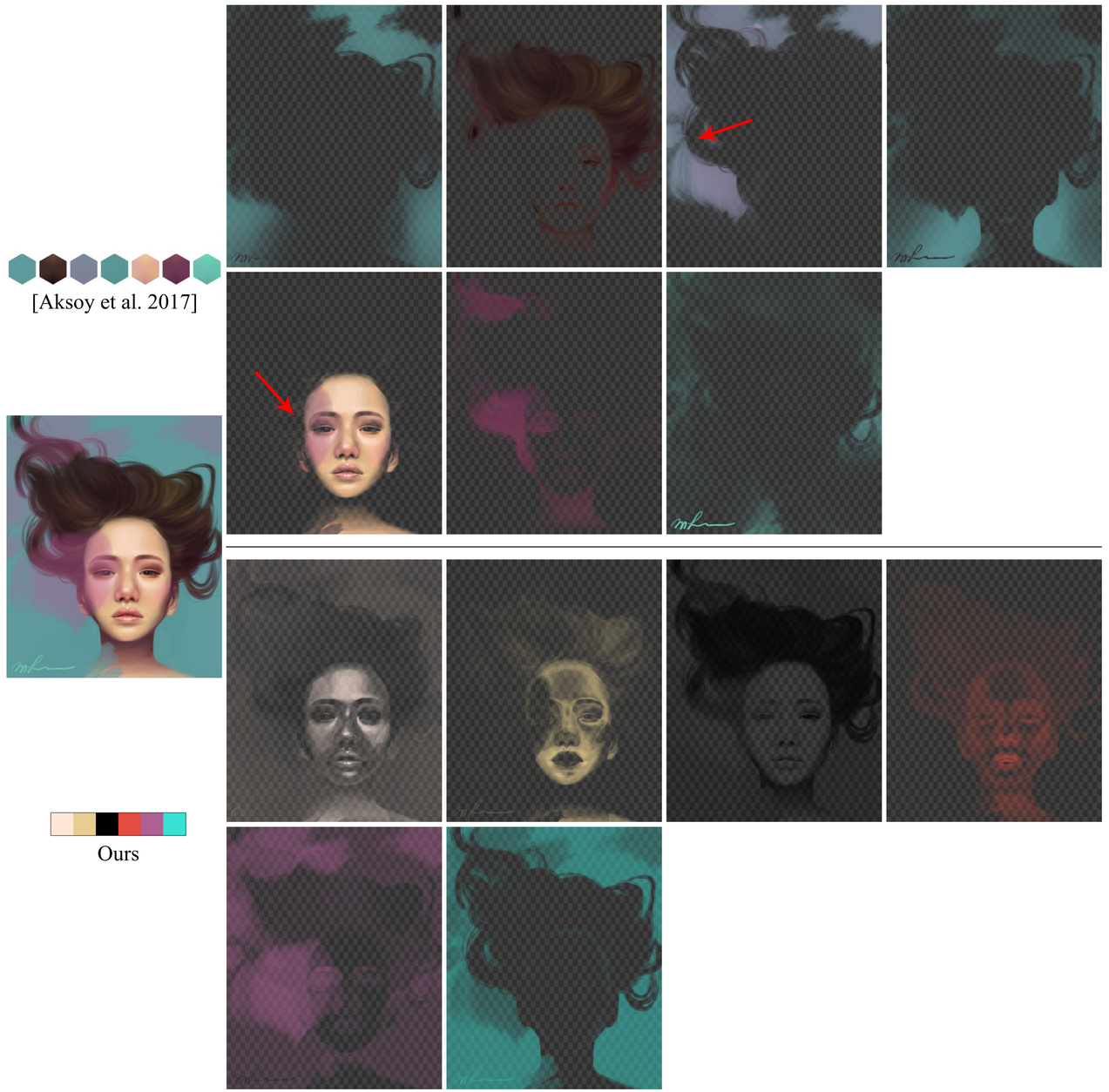
Python Implementation


```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()
```

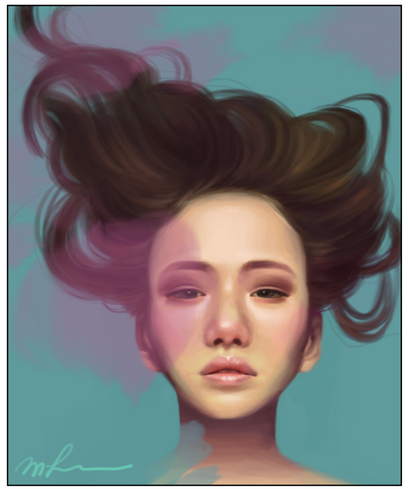
Comparisons

Layer quality comparison with [Aksoy et al. 2017]

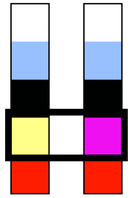
Layer quality comparison with [Aksoy et al. 2017]

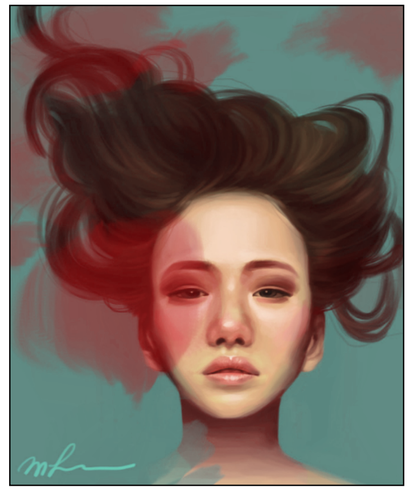


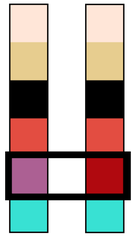
Recoloring comparison with three previous methods



Original





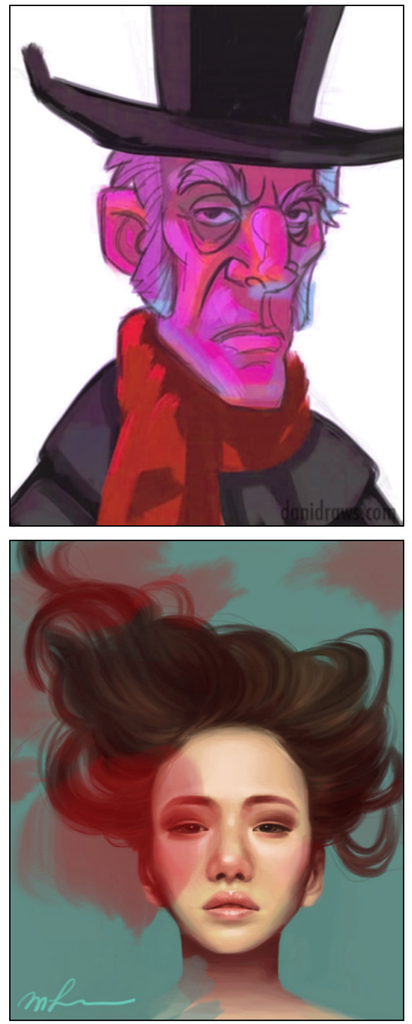


Ours

Recoloring comparison with three previous methods

Original

Aksoy et al. 2017

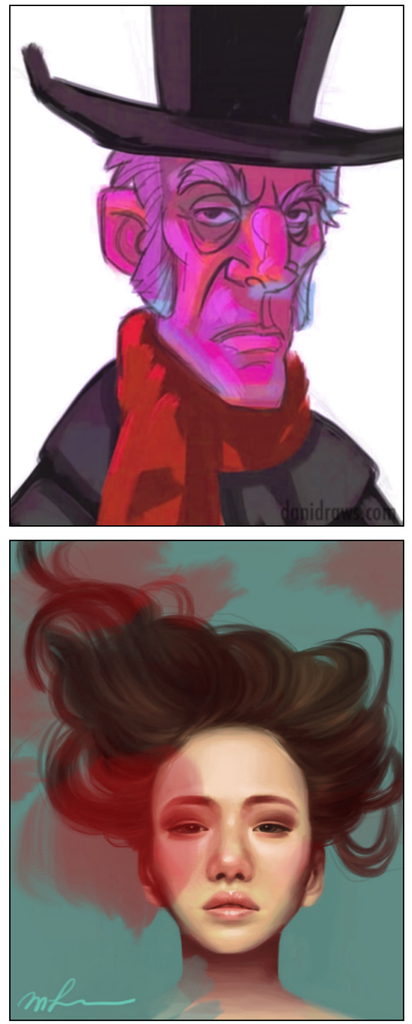


Ours

Recoloring comparison with three previous methods

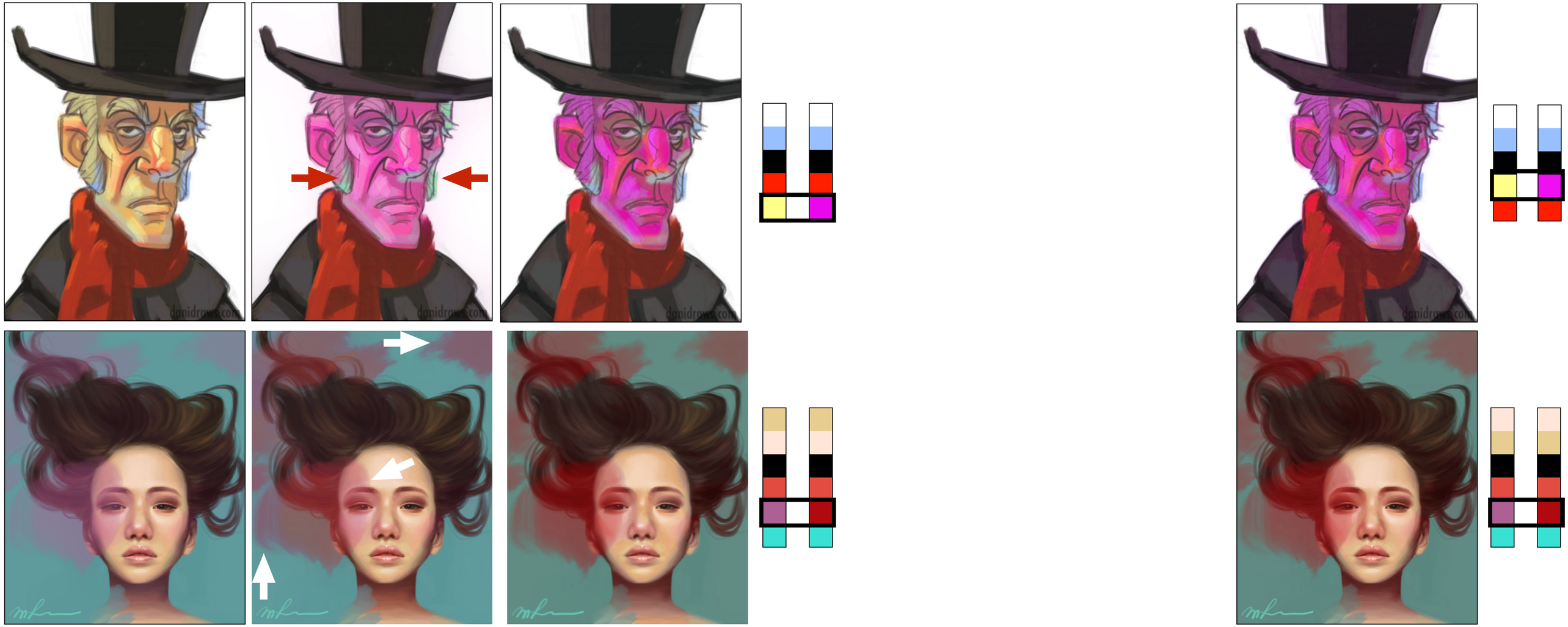
Original

Aksoy et al. 2017



Ours

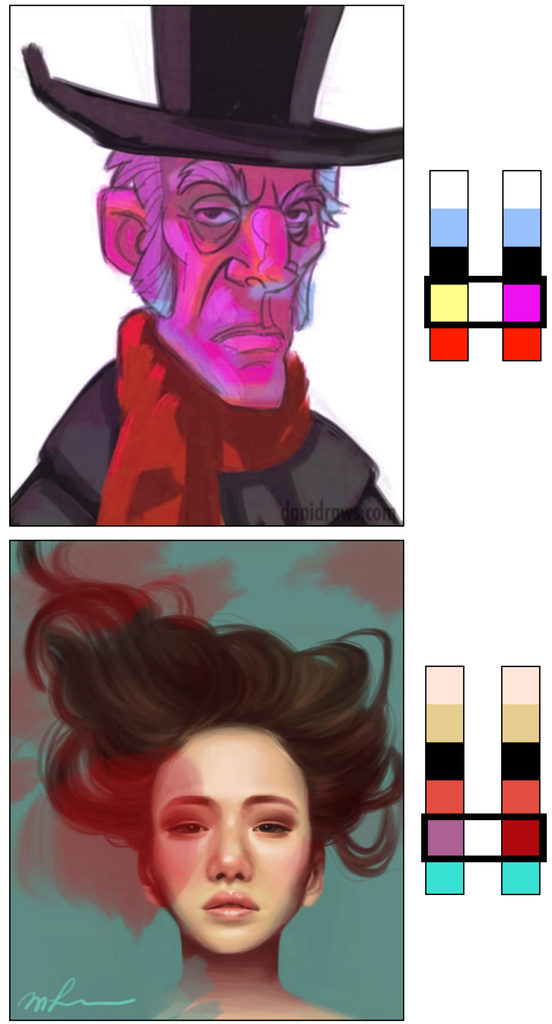
Recoloring comparison with three previous methods



Original Aksoy et al. 2017 Tan et al. 2016 Ours

Recoloring comparison with three previous methods

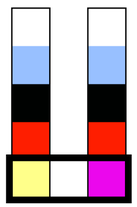
Original Aksoy et al. 2017 Tan et al. 2016

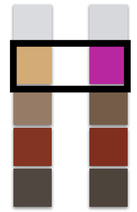


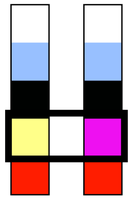
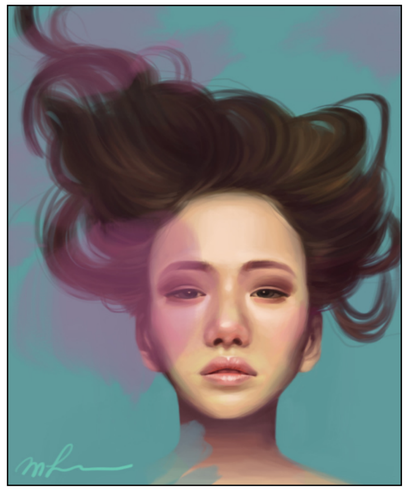
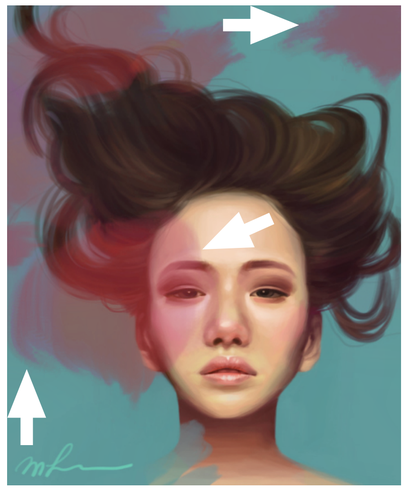
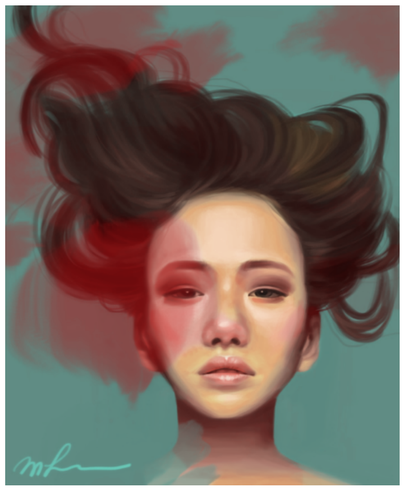
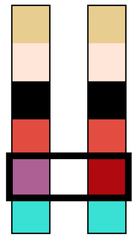
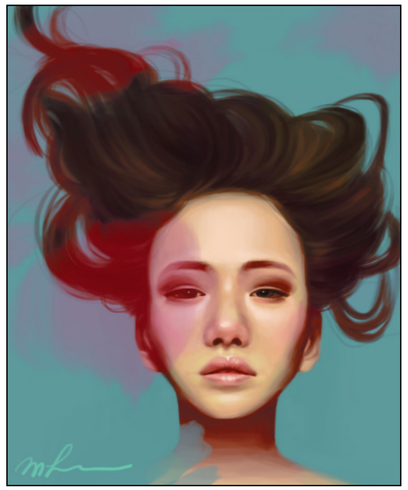
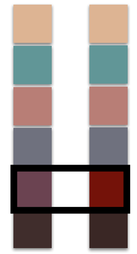
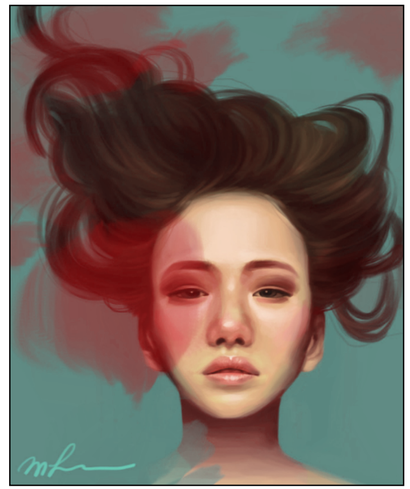
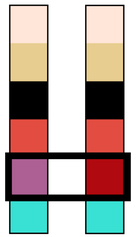
Ours

Recoloring comparison with three previous methods



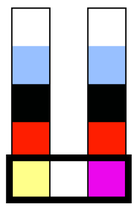


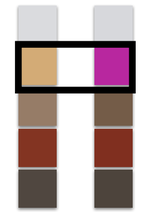










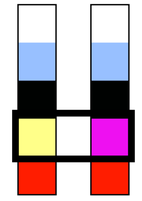
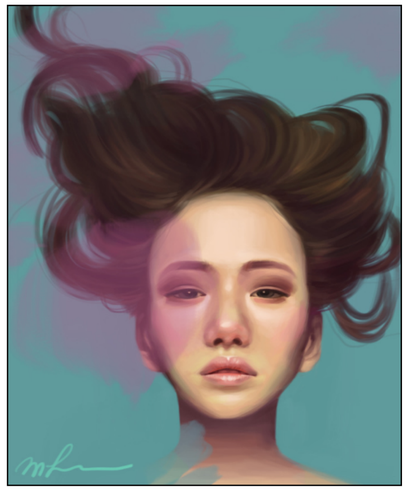
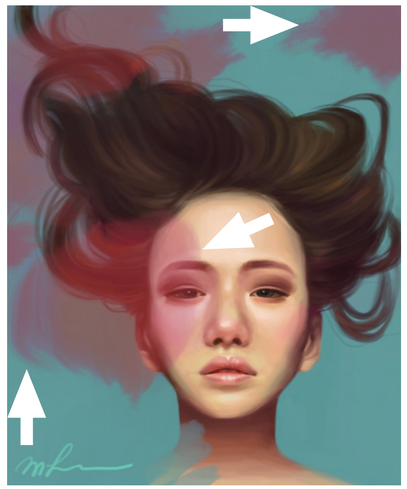
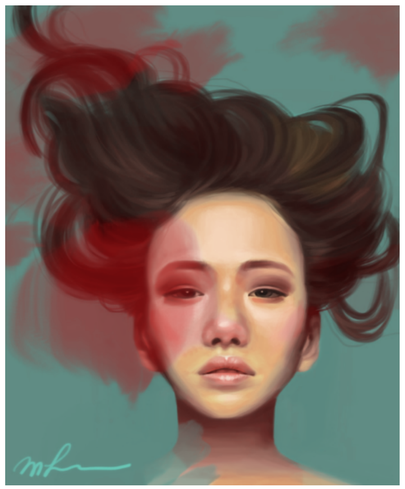
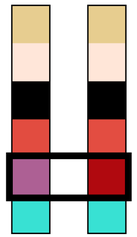
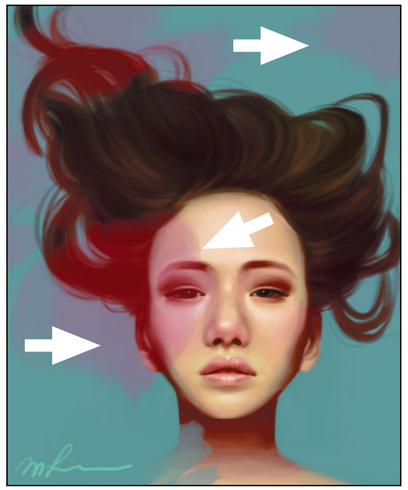
Original Aksoy et al. 2017 Tan et al. 2016 Chang et al. 2015 Ours

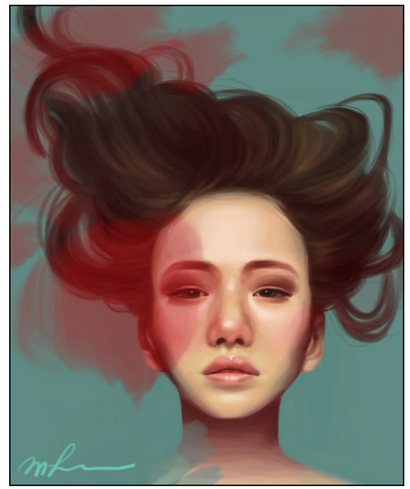
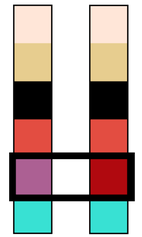
Recoloring comparison with three previous methods





Original Aksoy et al. 2017 Tan et al. 2016 Chang et al. 2015 Ours

Demo

Javascript + Python with PyOpenCL

Layer creation from scratch

ge: girls.png

Reconstruction

Difference

Layers:

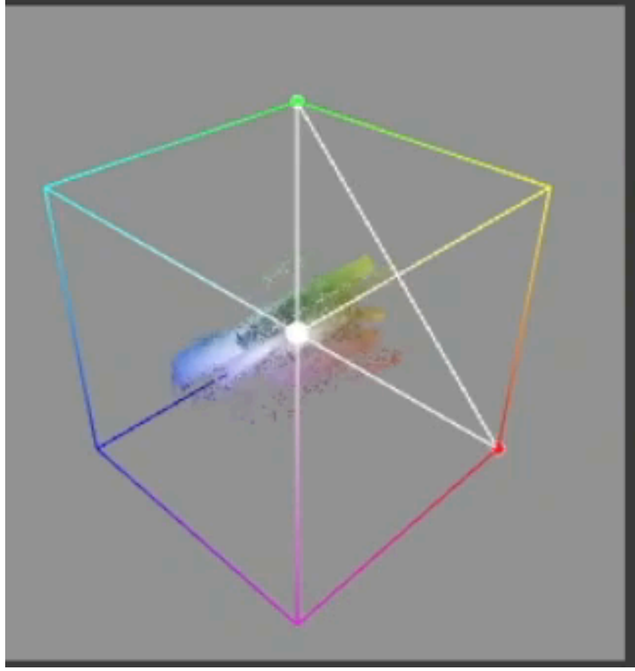
hoose File | No file chosen

re-compute RGBXY weights

reate automatic palette

Prescribed number of layers: 6

dd Random Palette Color



colorful

ge: girls.png

Reconstruction

Difference

Layers:

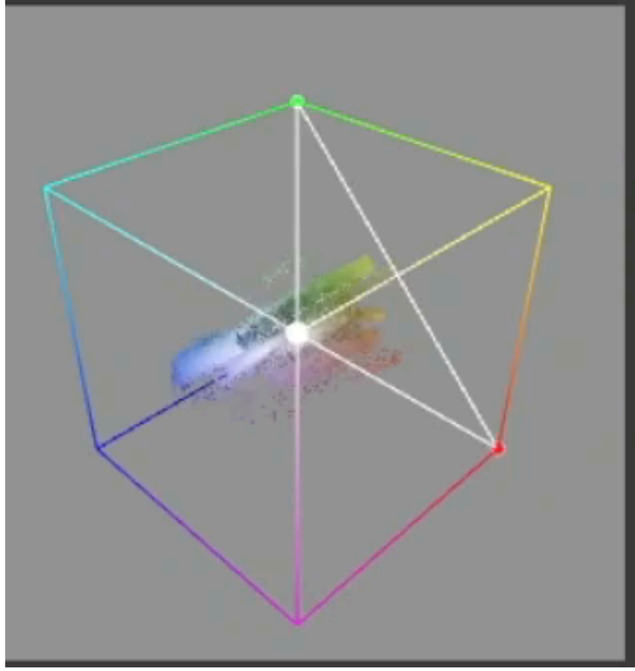
hoose File | No file chosen

re-compute RGBXY weights

reate automatic palette

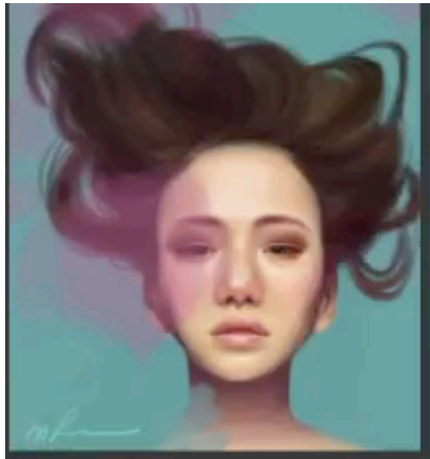
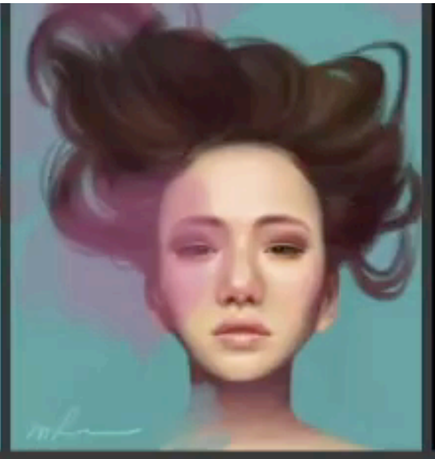
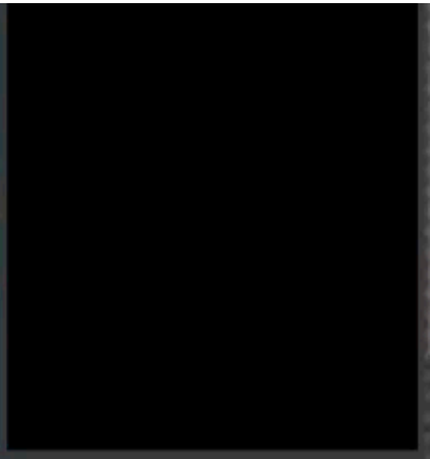
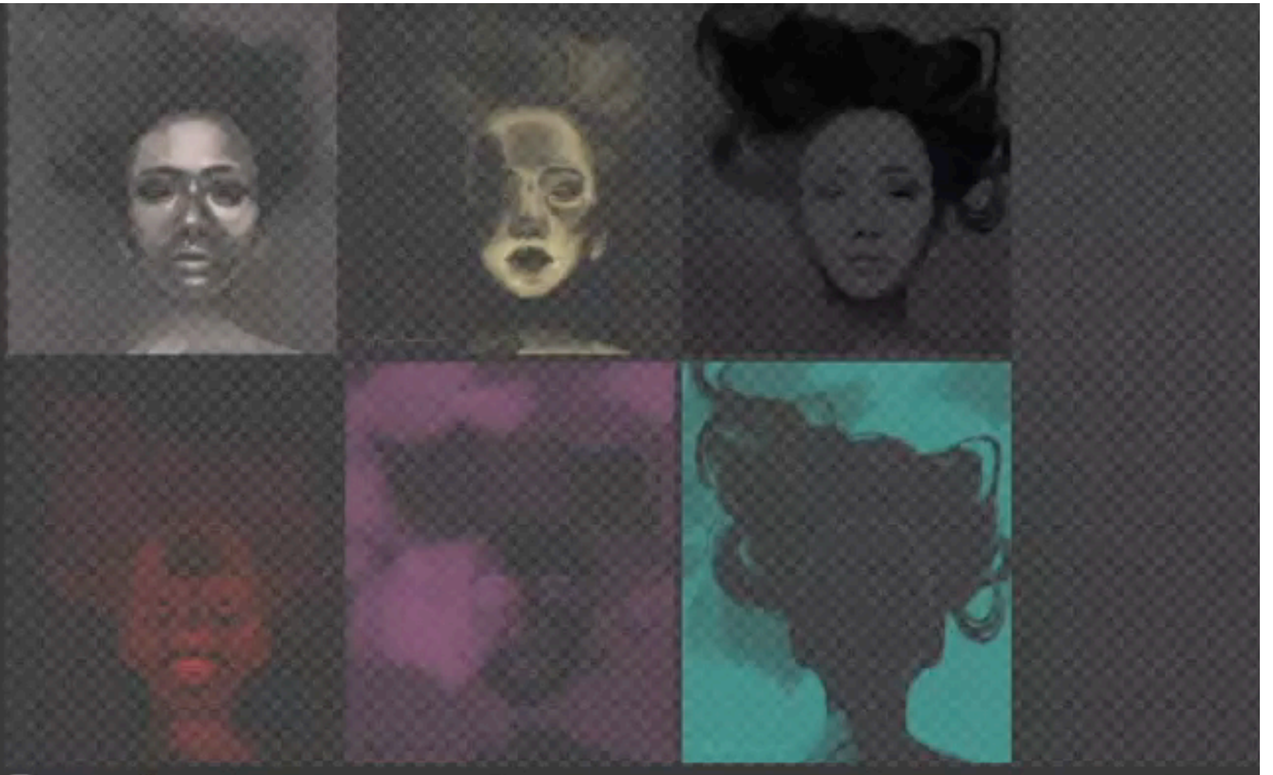
Prescribed number of layers: 6

dd Random Palette Color



colorful

Layer creation from an automatic palette

Choose File No file chosen

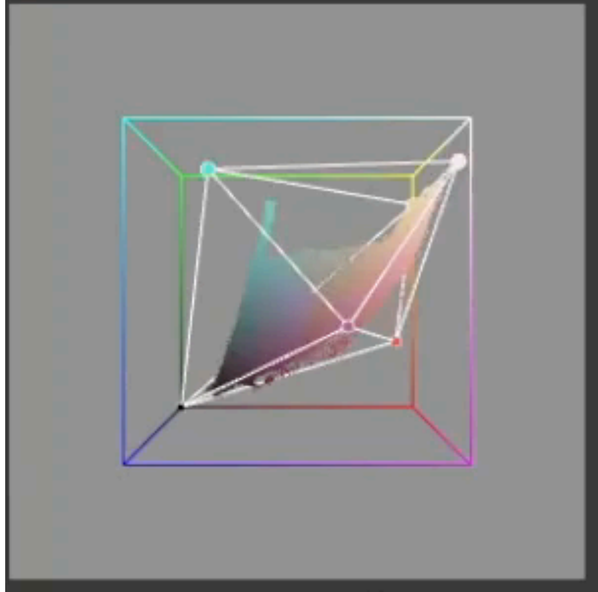
Pre-compute RGBXY weights

Create automatic palette

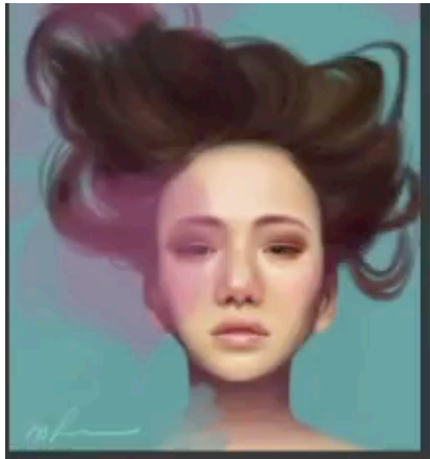
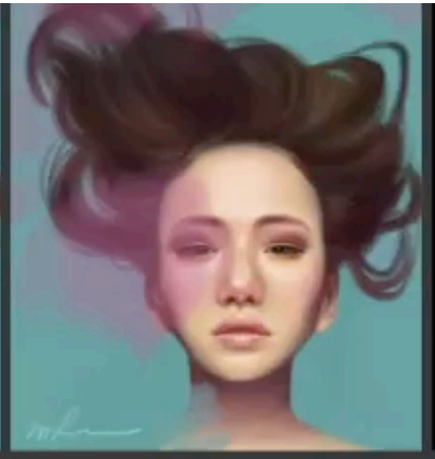
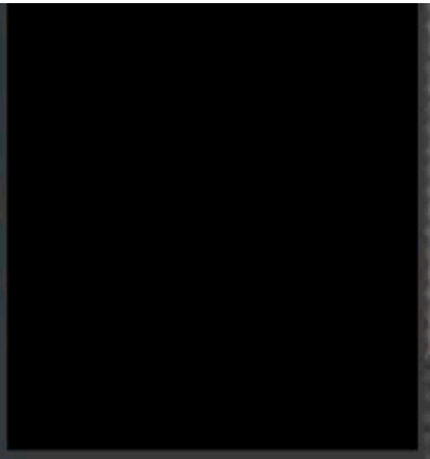
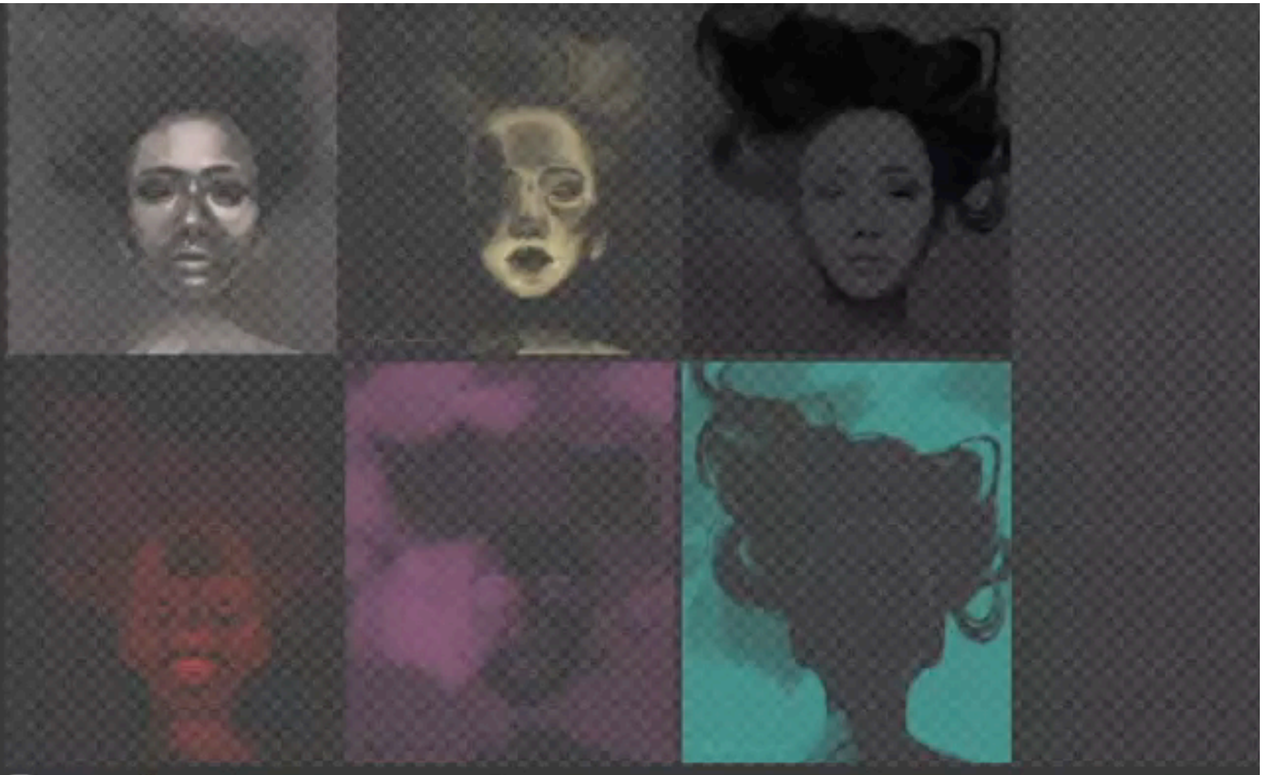
Prescribed number of layers: 6

Add Random Palette Color

colorful



Rotation has intertia:

Choose File No file chosen

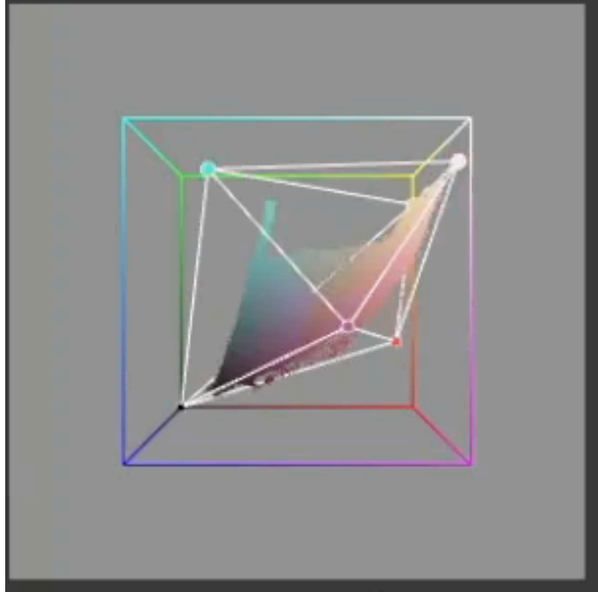
Pre-compute RGBXY weights

Create automatic palette

Prescribed number of layers: 6

Add Random Palette Color

colorful



Rotation has intertia:

Interactive decomposition gives more control to the users

original

recoloring with interactively edited palette

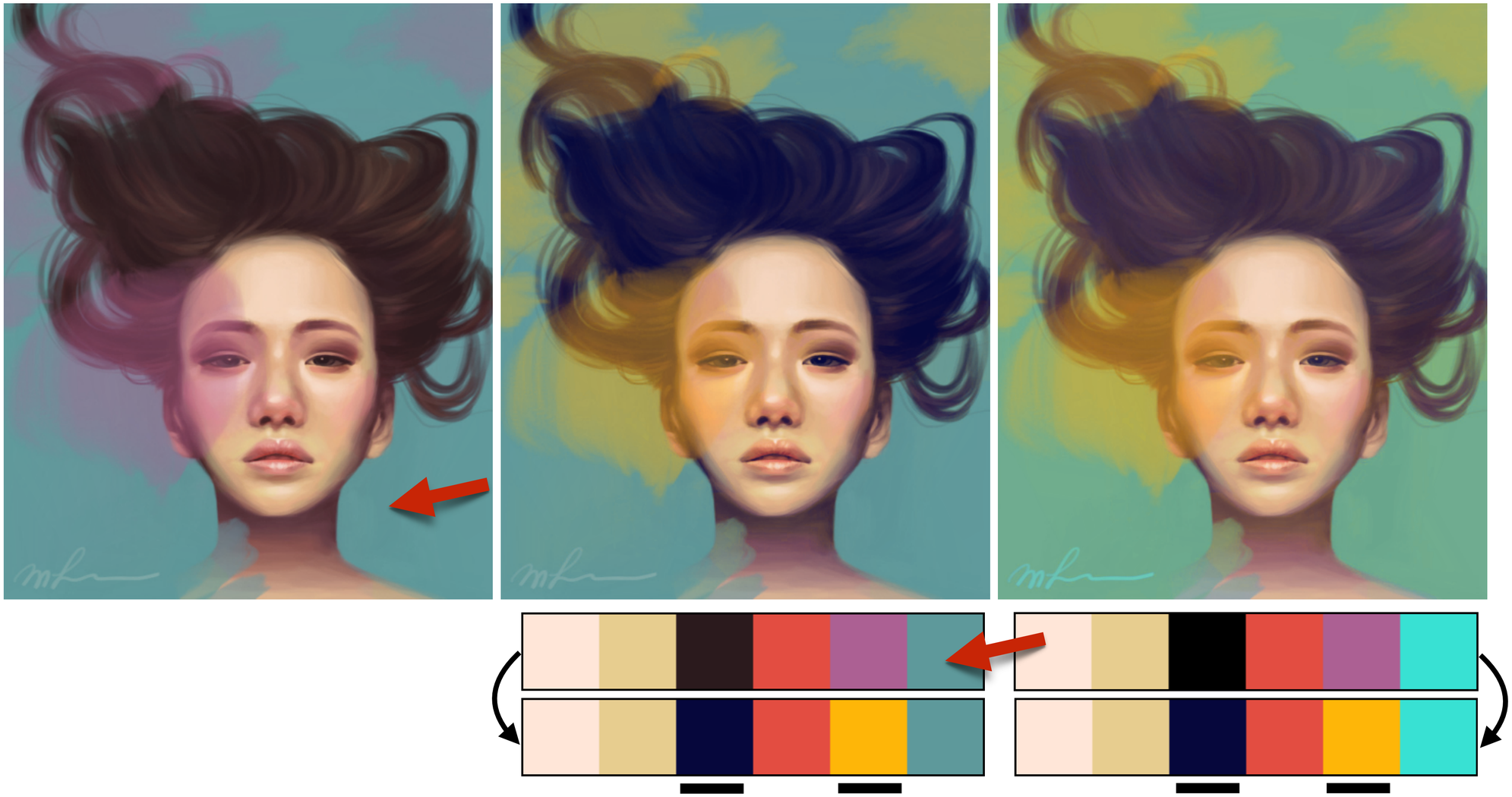
recoloring with automatic palette

Interactive decomposition gives more control to the users

original

recoloring with interactively edited palette

recoloring with automatic palette

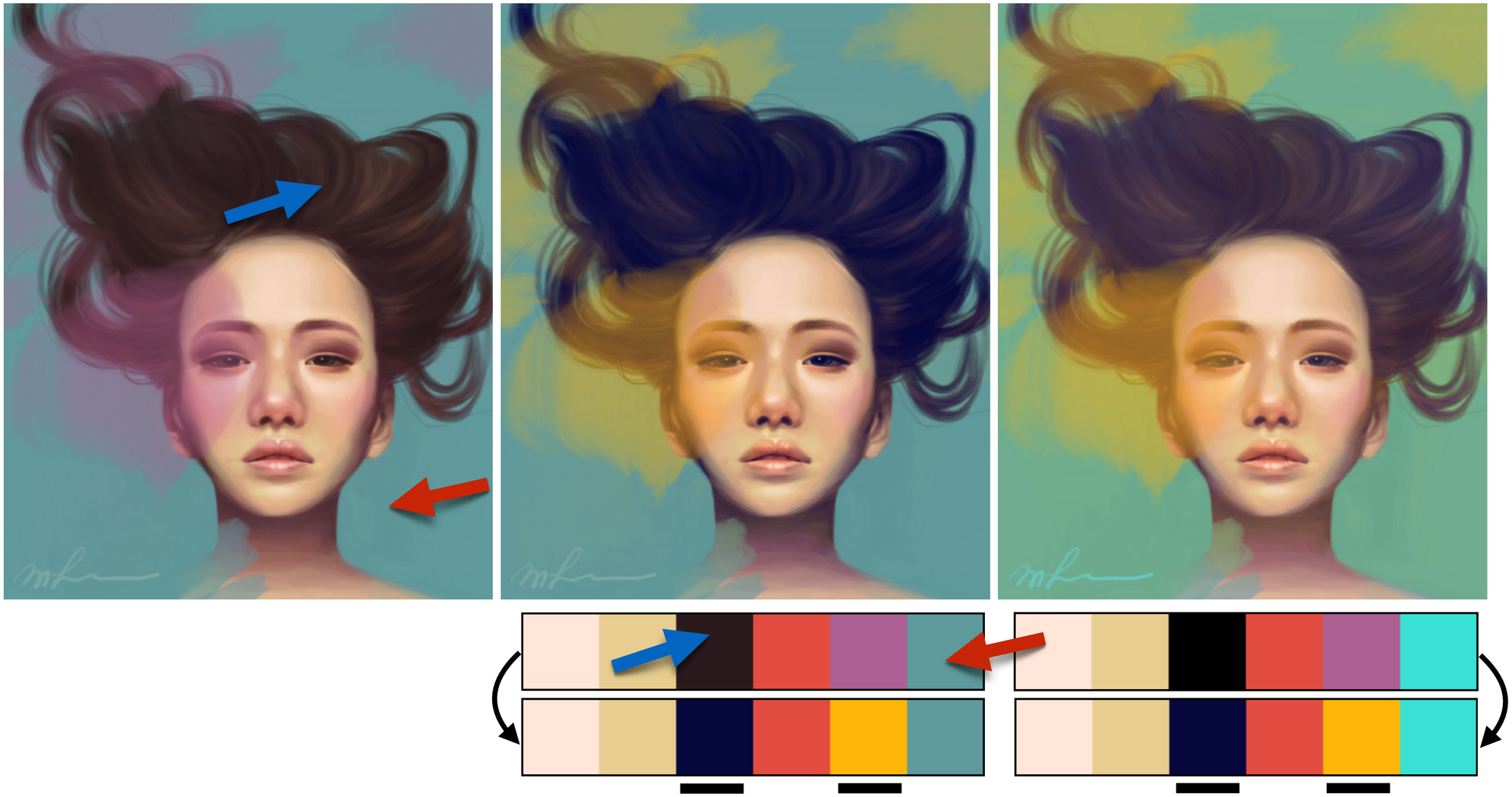


Interactive decomposition gives more control to the users

original

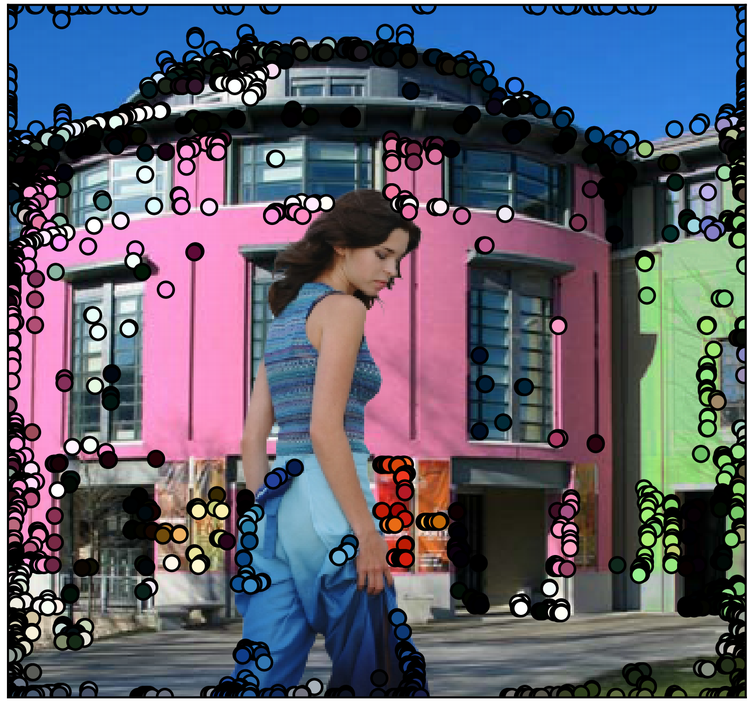
recoloring with interactively edited palette

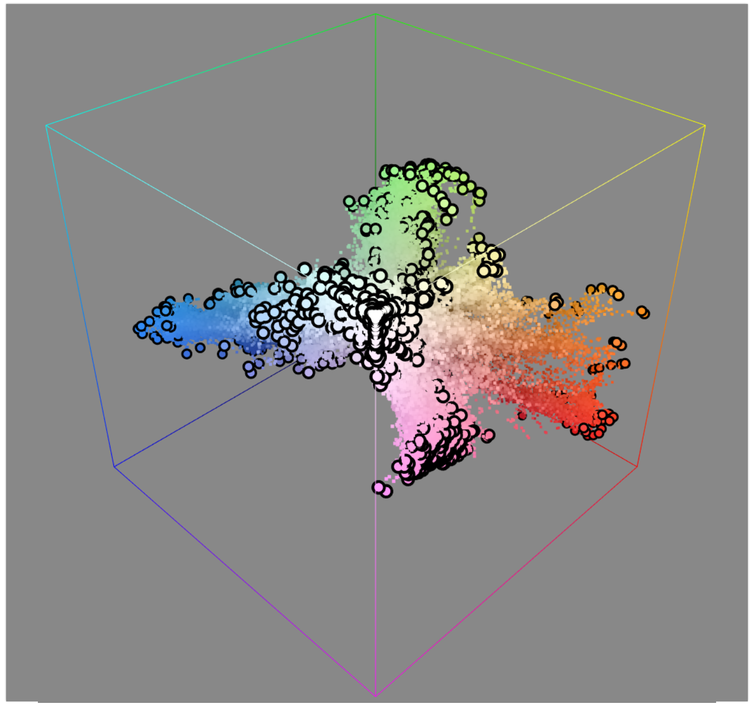
recoloring with automatic palette

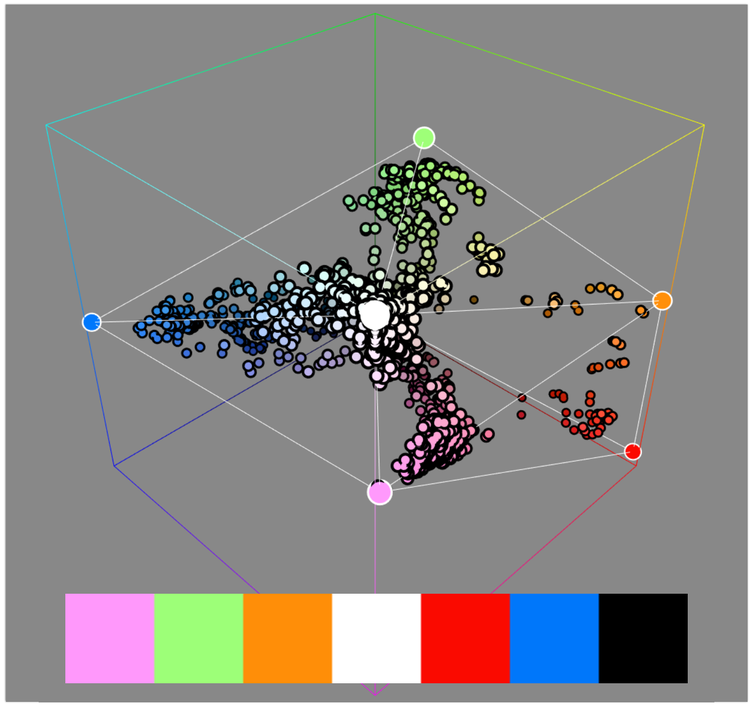


Conclusion

- An extremely efficient approach to layer decomposition via RGBXY geometry







Conclusion

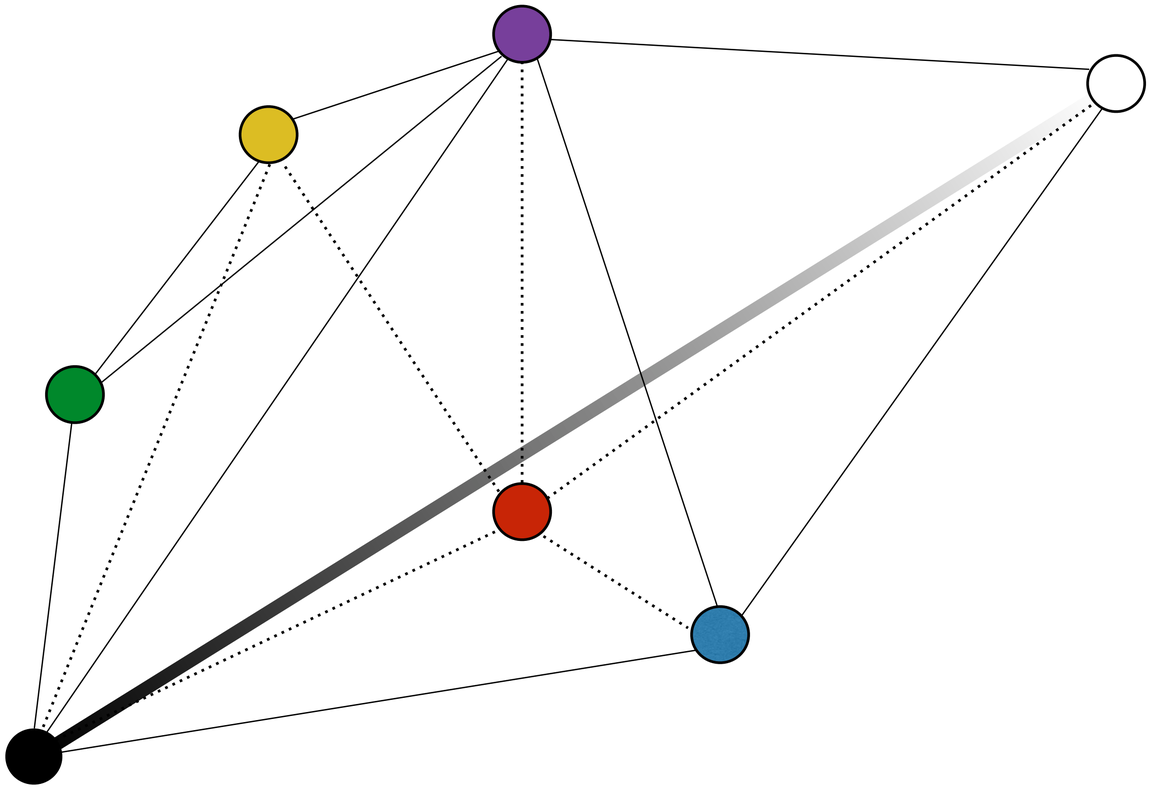
- Our two-level decomposition supports real-time decomposition when palette editing.

Palette updates **Fixed**

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

Conclusion

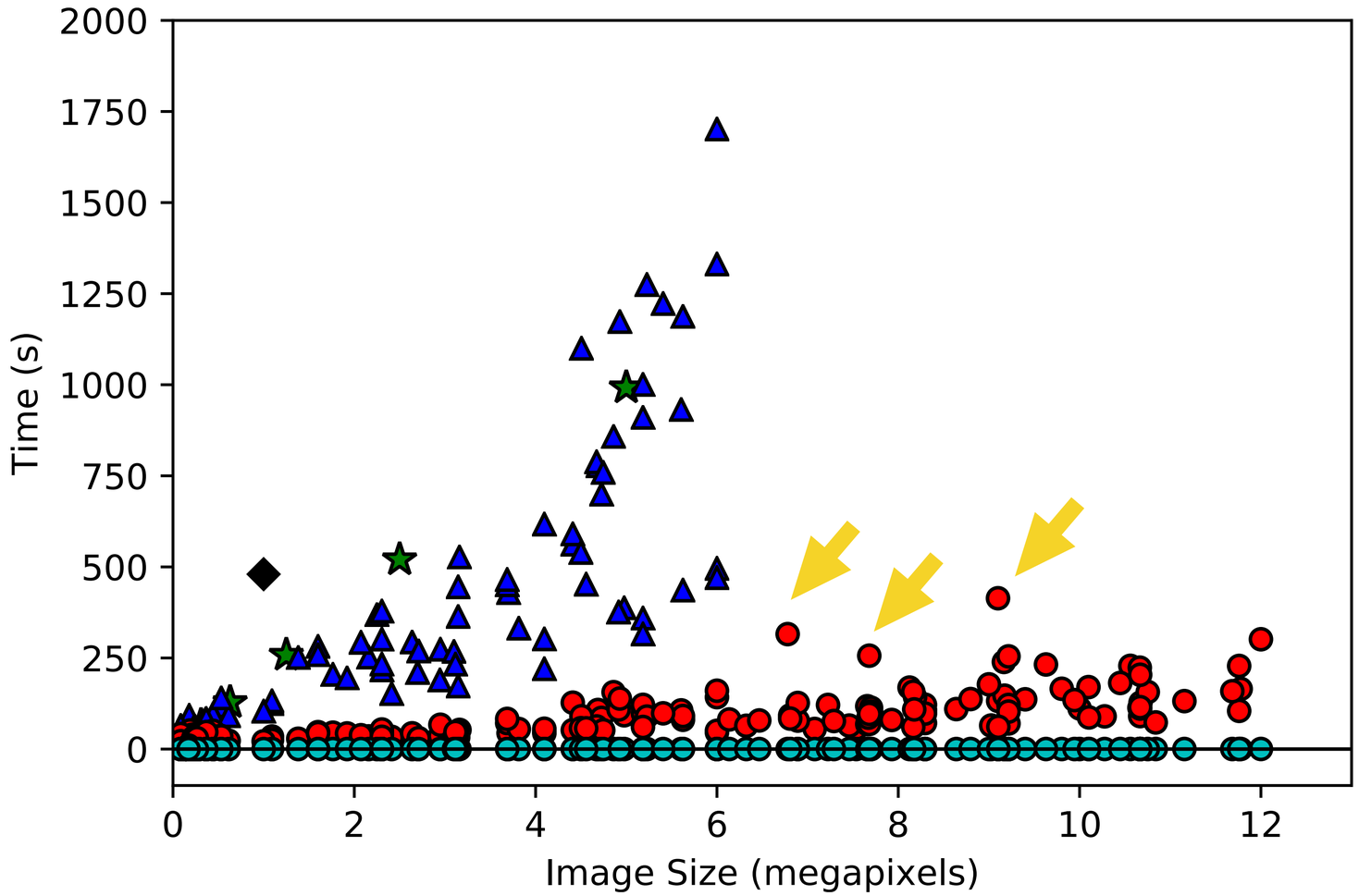
- It’s important to capture the “line of greys”.



Star tessellation

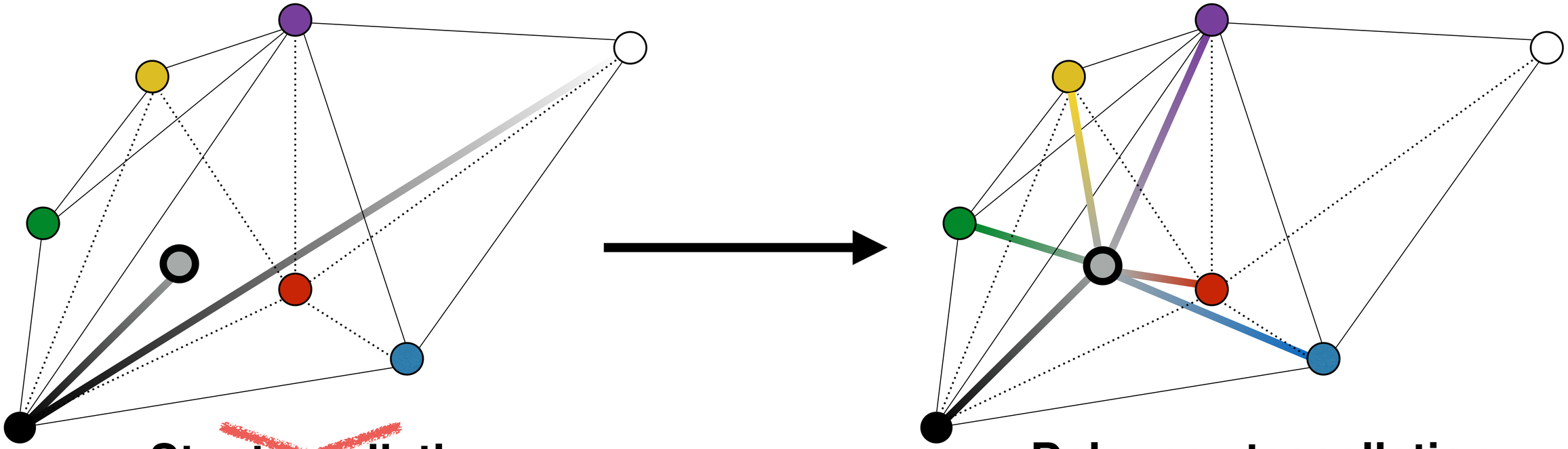
Limitations

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



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.



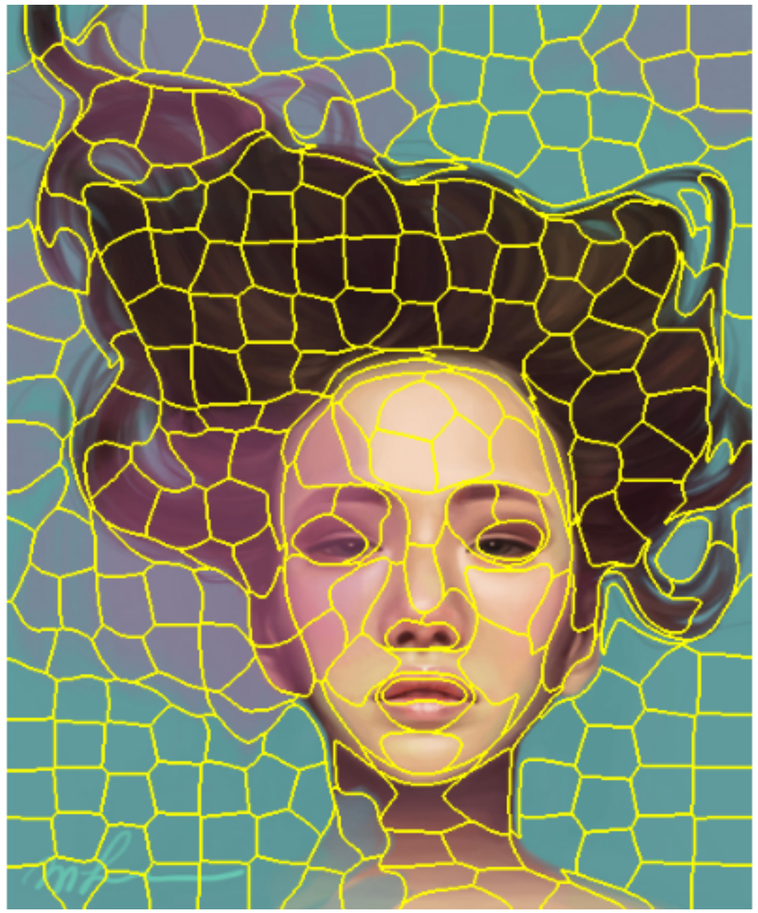
~~Star tessellation~~

Delaunay tessellation

Future Work

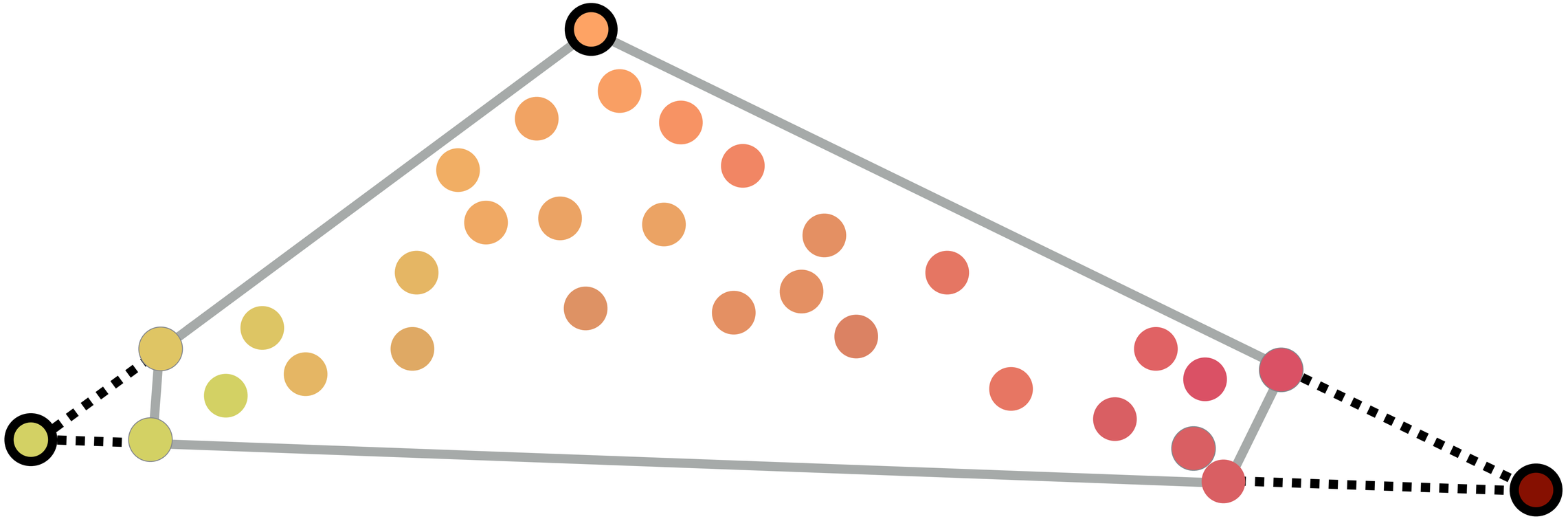
- More speed via super-pixels or parallel convex hull algorithms.

+



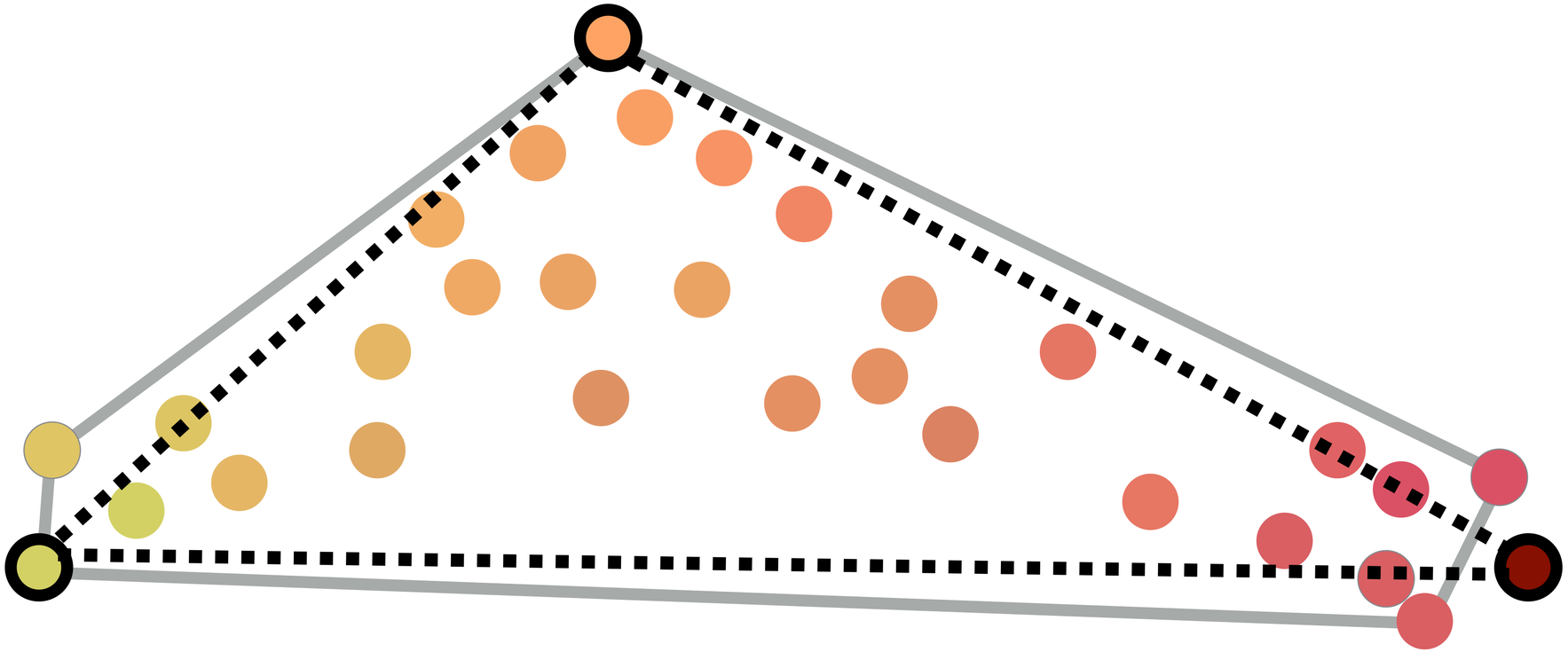
Future Work

- Robustness via approximate convex hull algorithms.



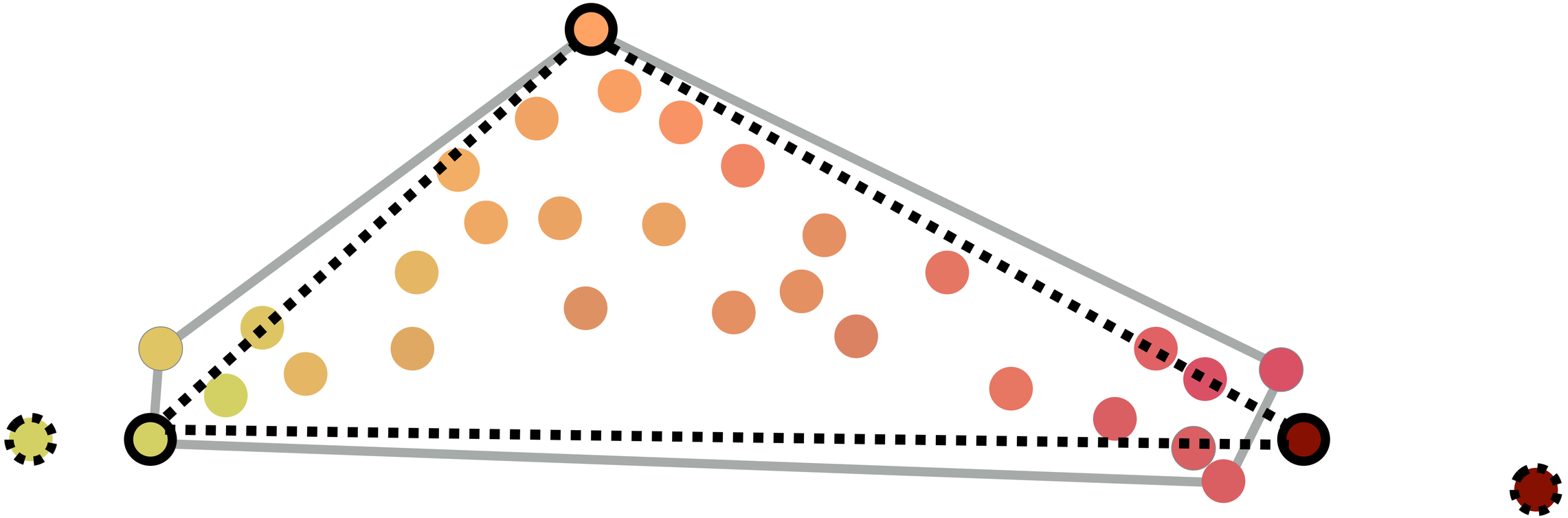
Future Work

- Robustness via approximate convex hull algorithms.



Future Work

- Robustness via approximate convex hull algorithms.



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.

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?