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

Jianchao Tan George Mason University

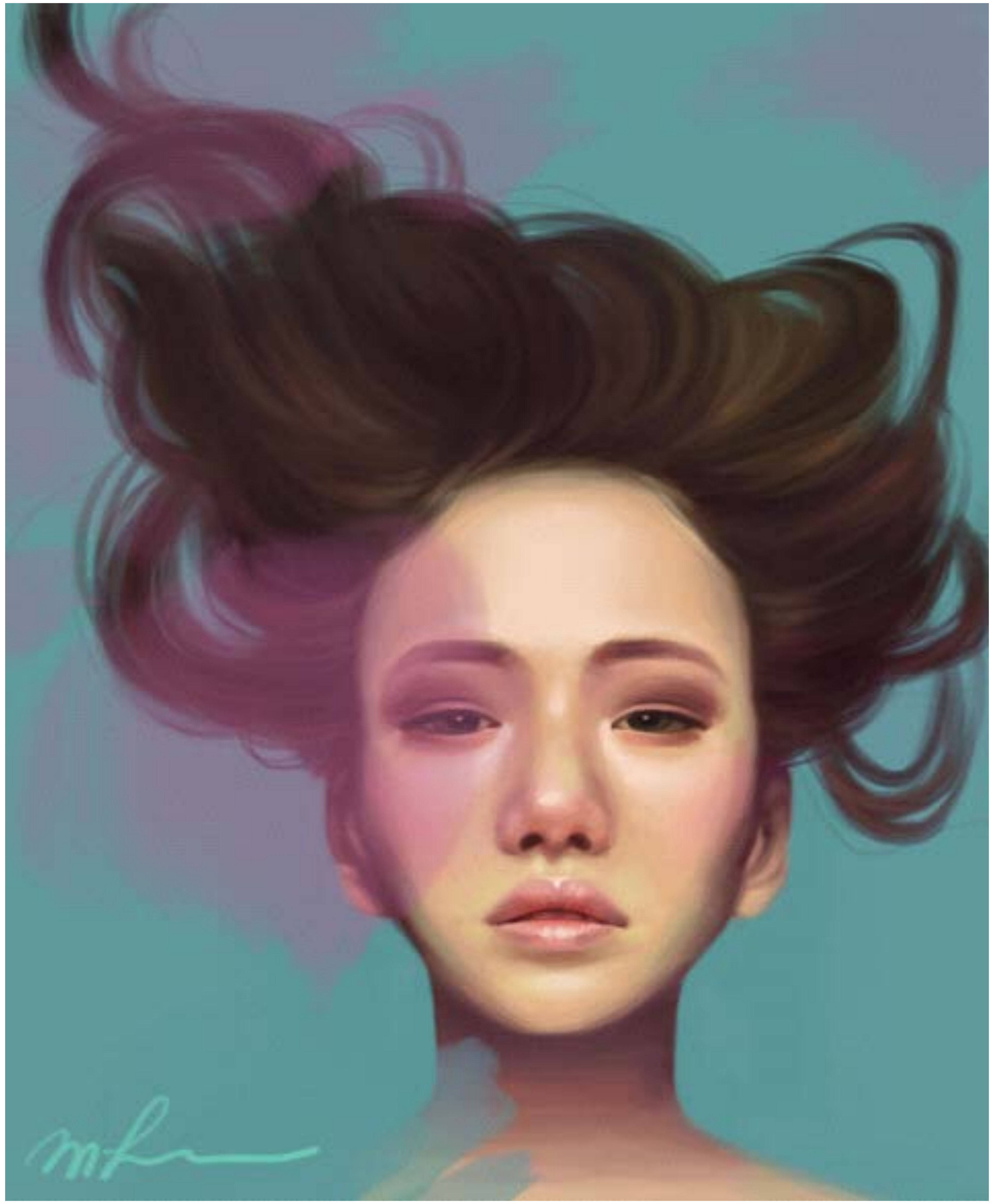
Jose Echevarria Adobe Research

Yotam Gingold George Mason University

SIGGRAPH ASIA 2018 TOKYO

George Mason University CraGL Creativity and Graphics Lab Adobe

Motivation: Layers Organize Images

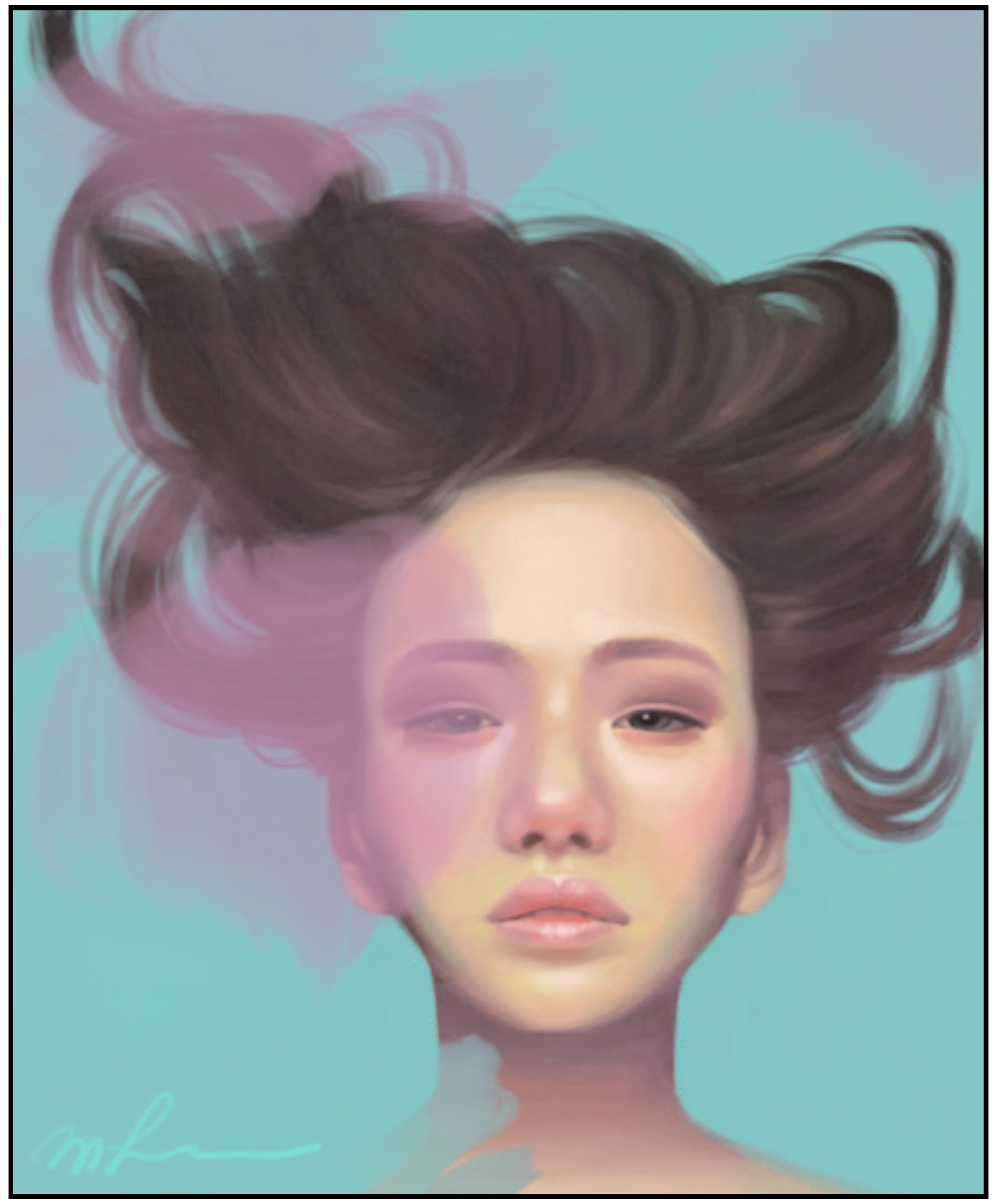


Motivation: Layers Organize Images

Motivation: Layers Organize Images

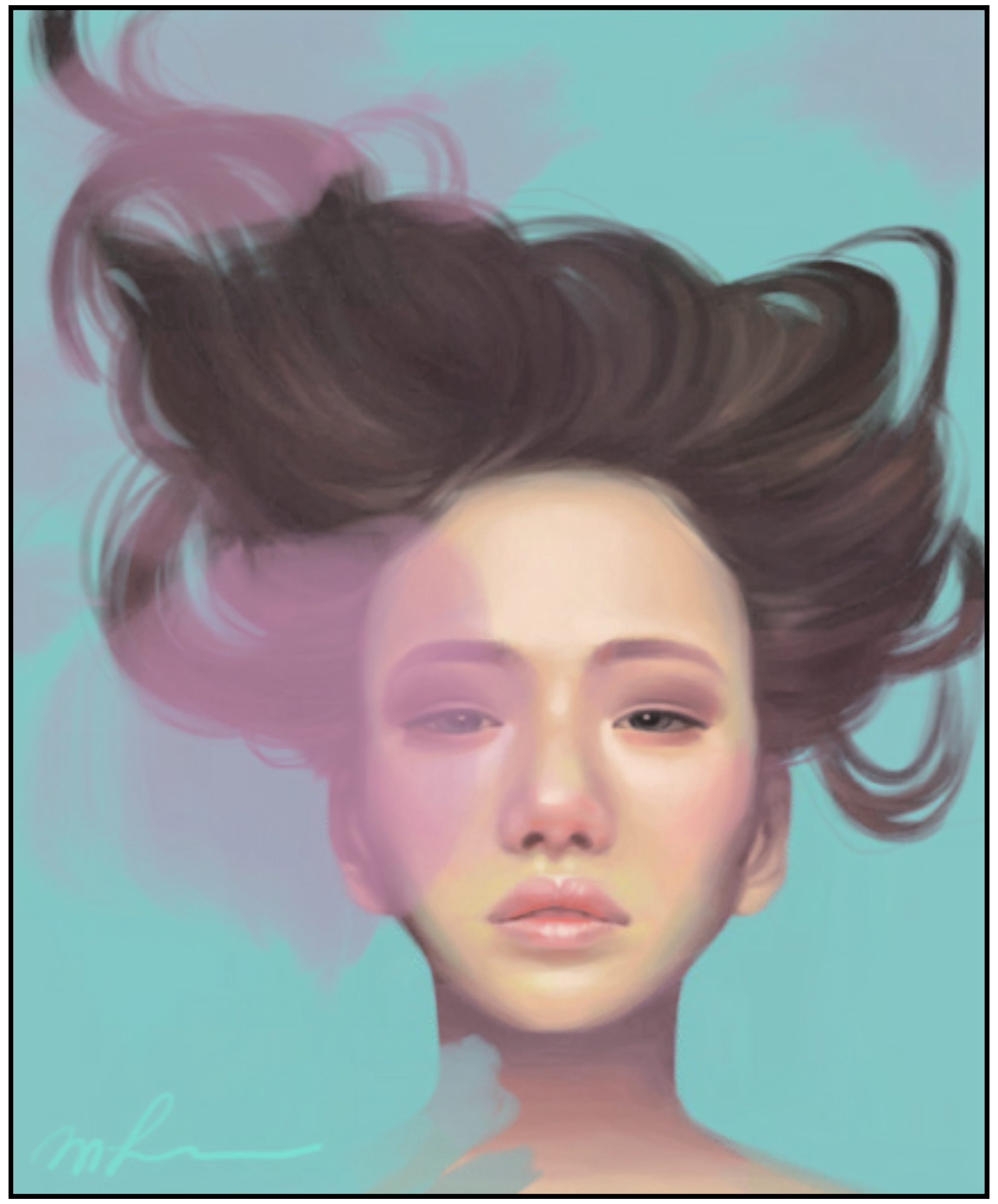
Goal

Input

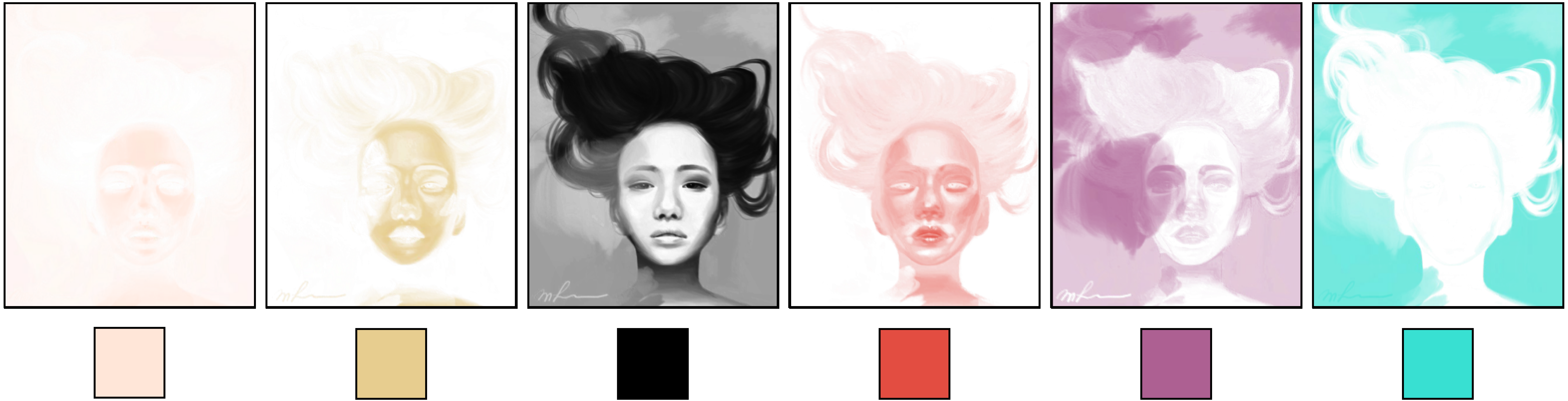


Goal

Input

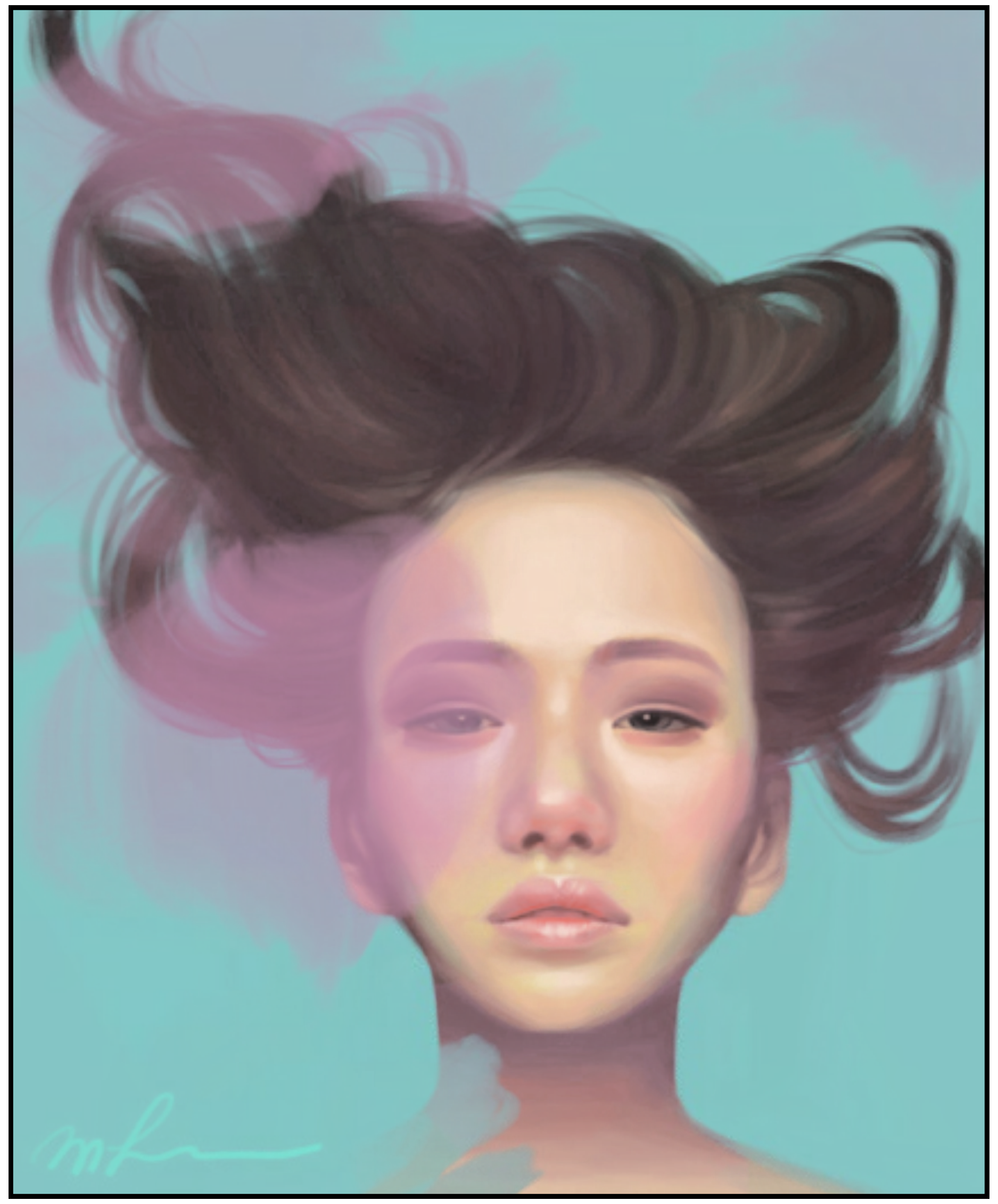


Layers

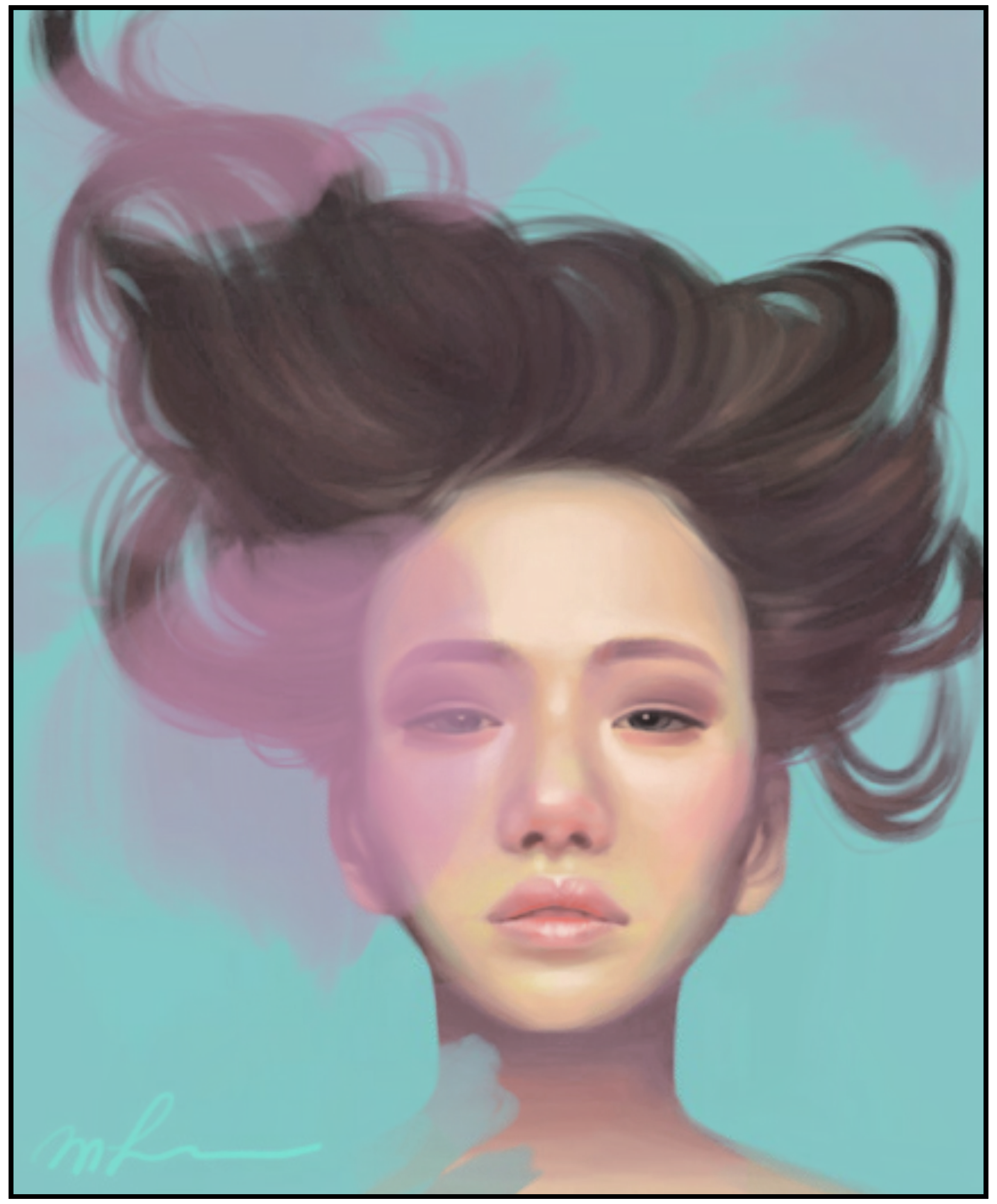


Goal

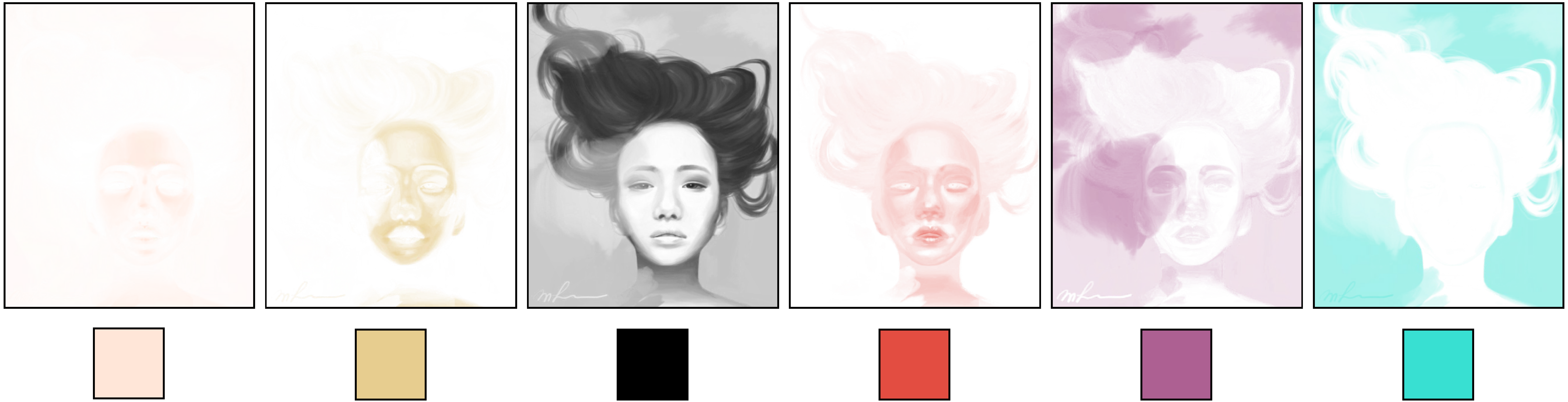
Input

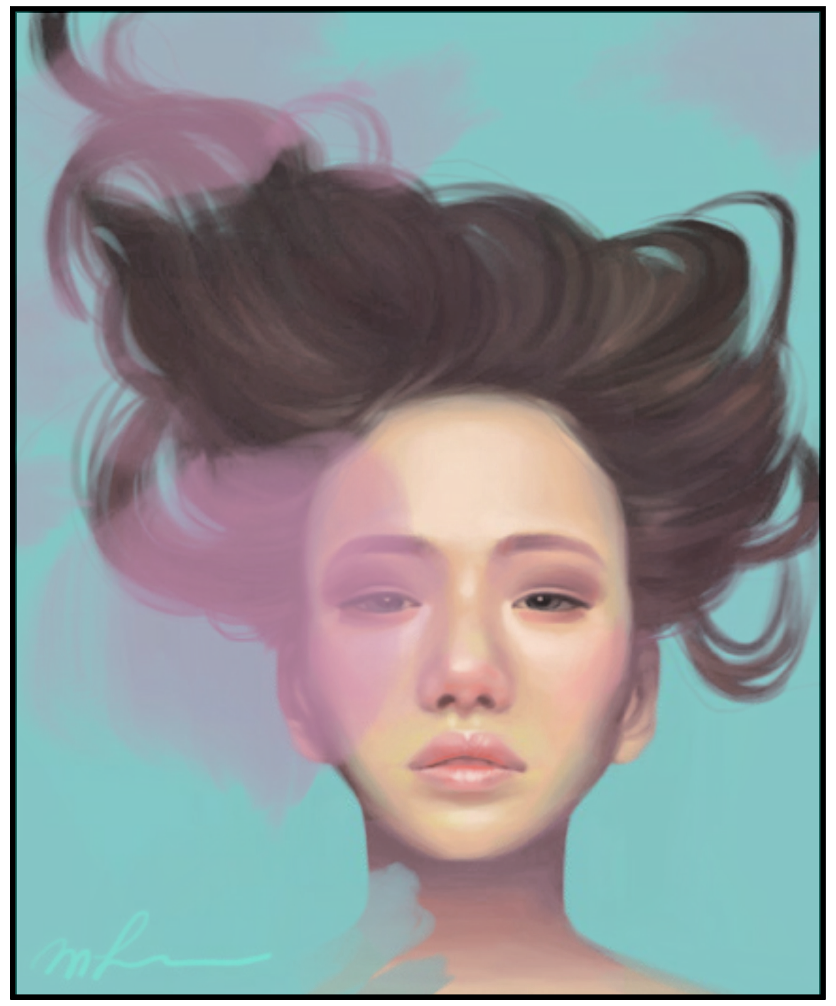


Reconstruction



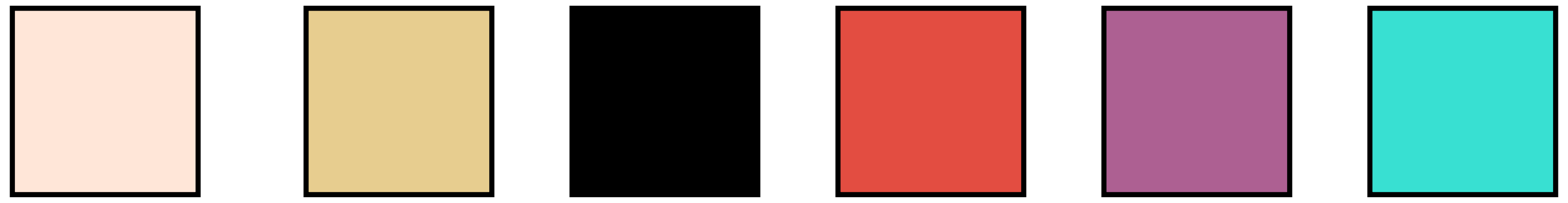
Layers



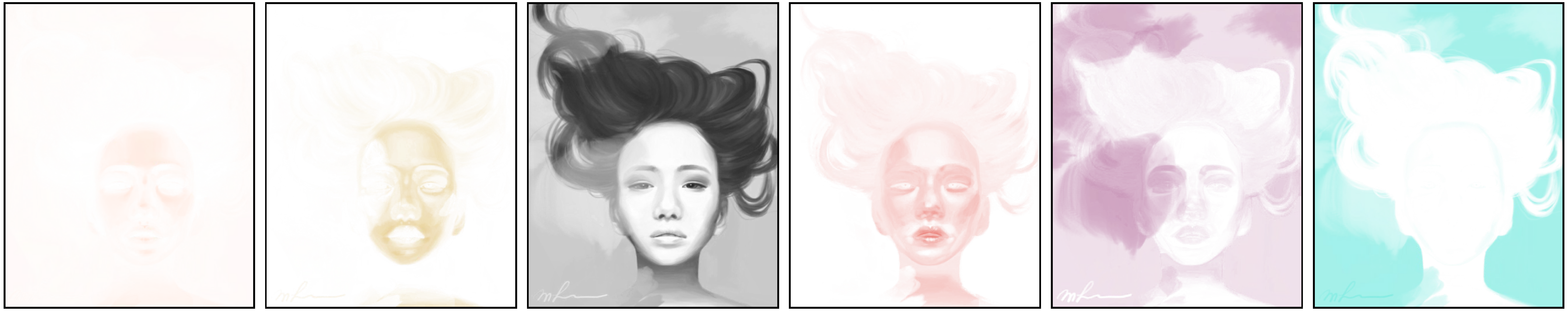


Subproblems

1. Palette extraction

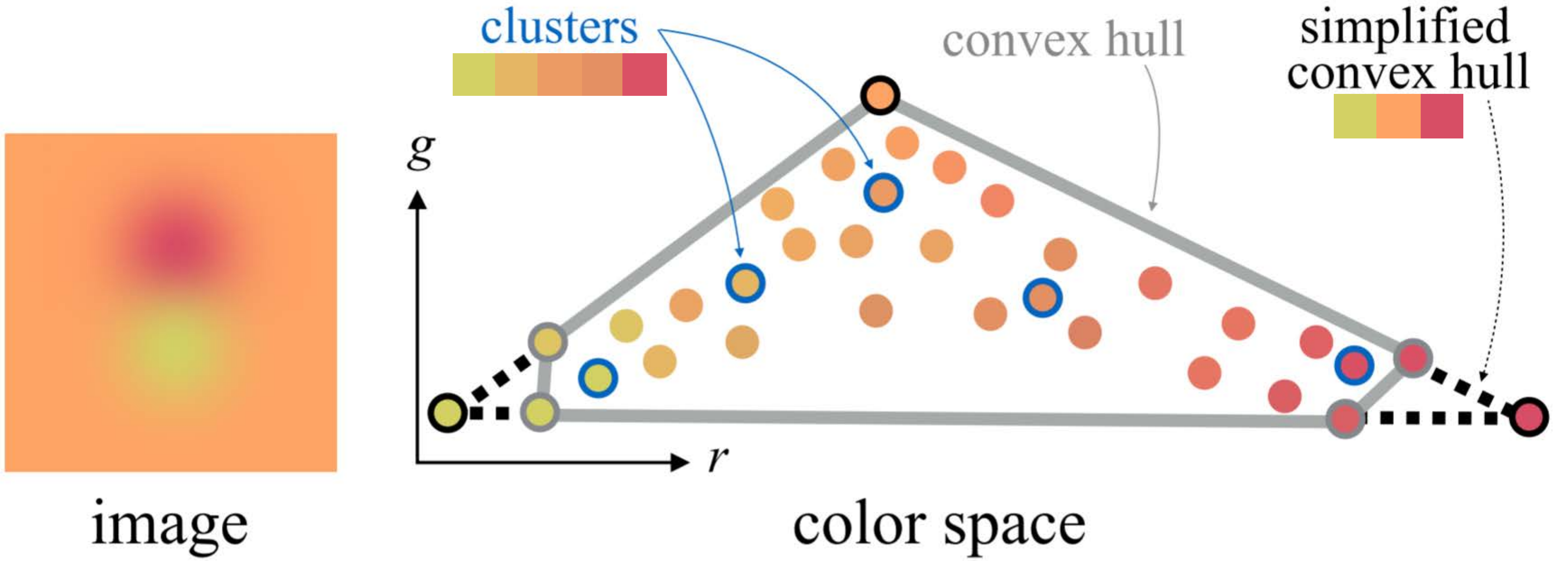


2. Palette-based layer decomposition



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]
    - …

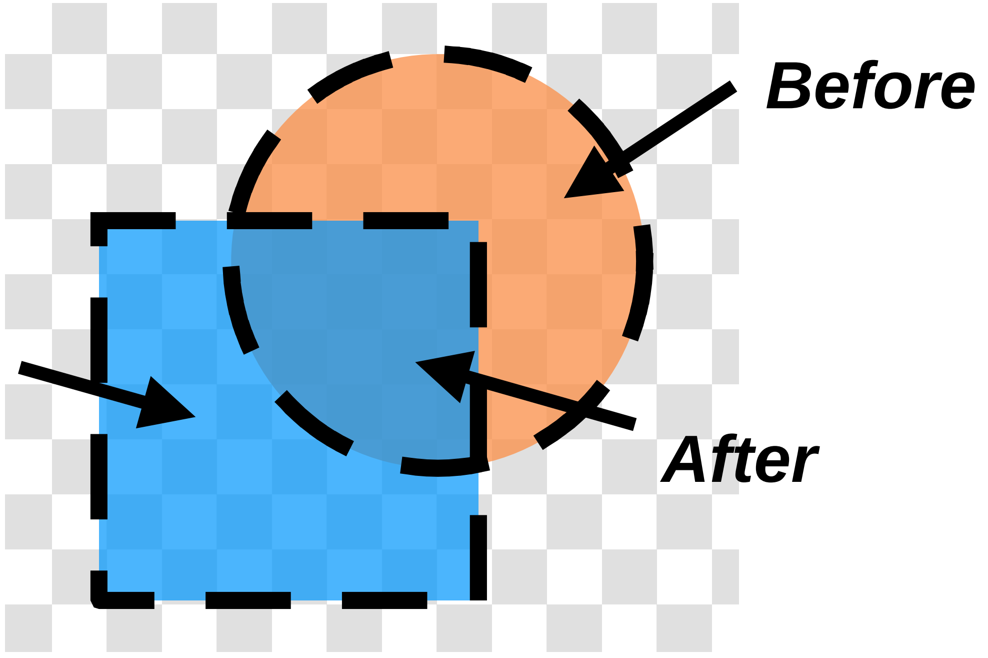


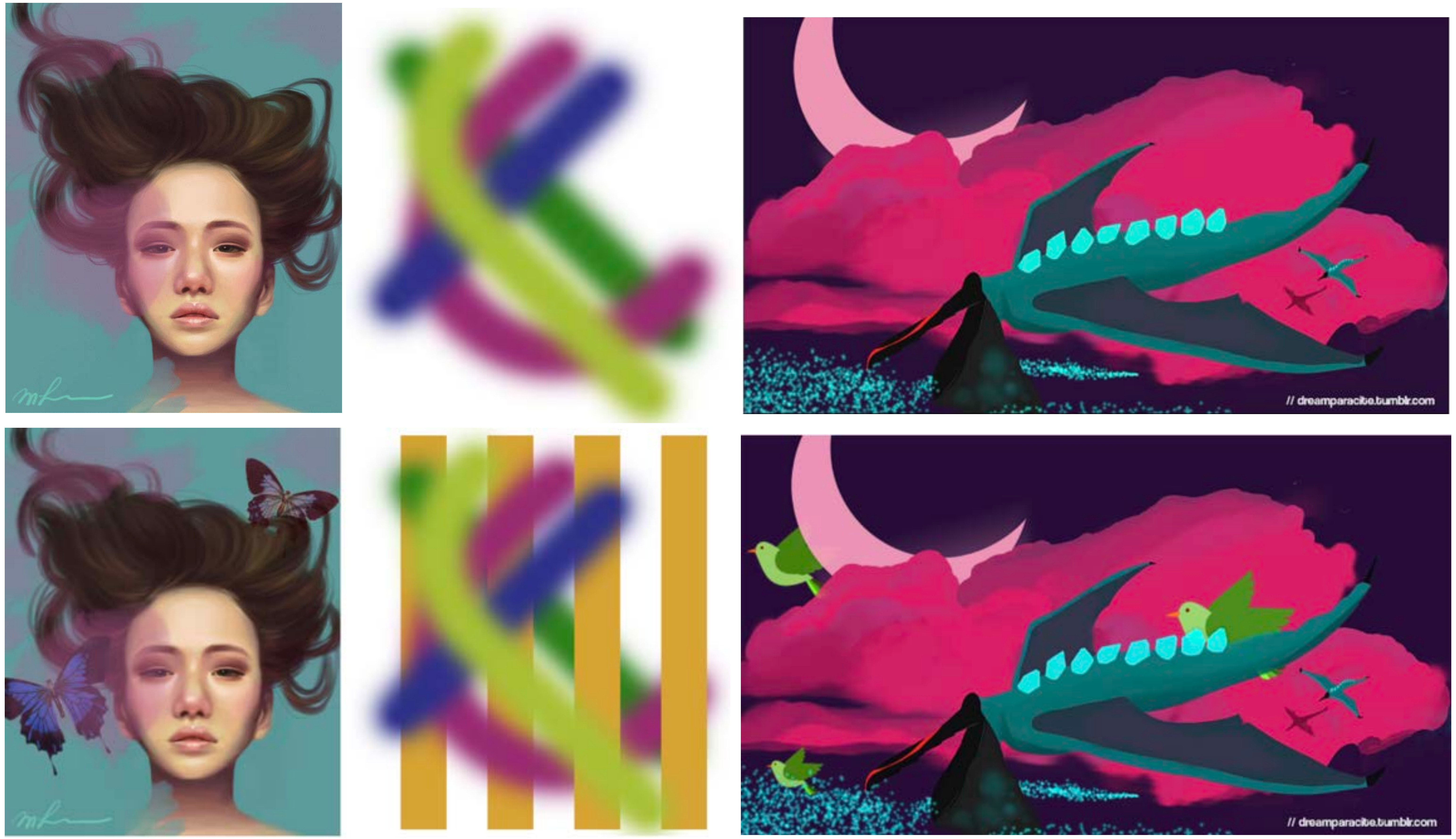
Decomposing Images into Layers via RGB-space Geometry [Tan et al. 2016]

Related Work

- Order-dependent translucent layers
  - Richardt et al. [2014]
  - Tan et al. [2015]
  - Tan et al. [2016]
  - Favreau et al. [2017]

$$After = Before \cdot (1 - \alpha) + Paint \cdot \alpha$$





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].

$$p = \sum w_i c_i$$

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

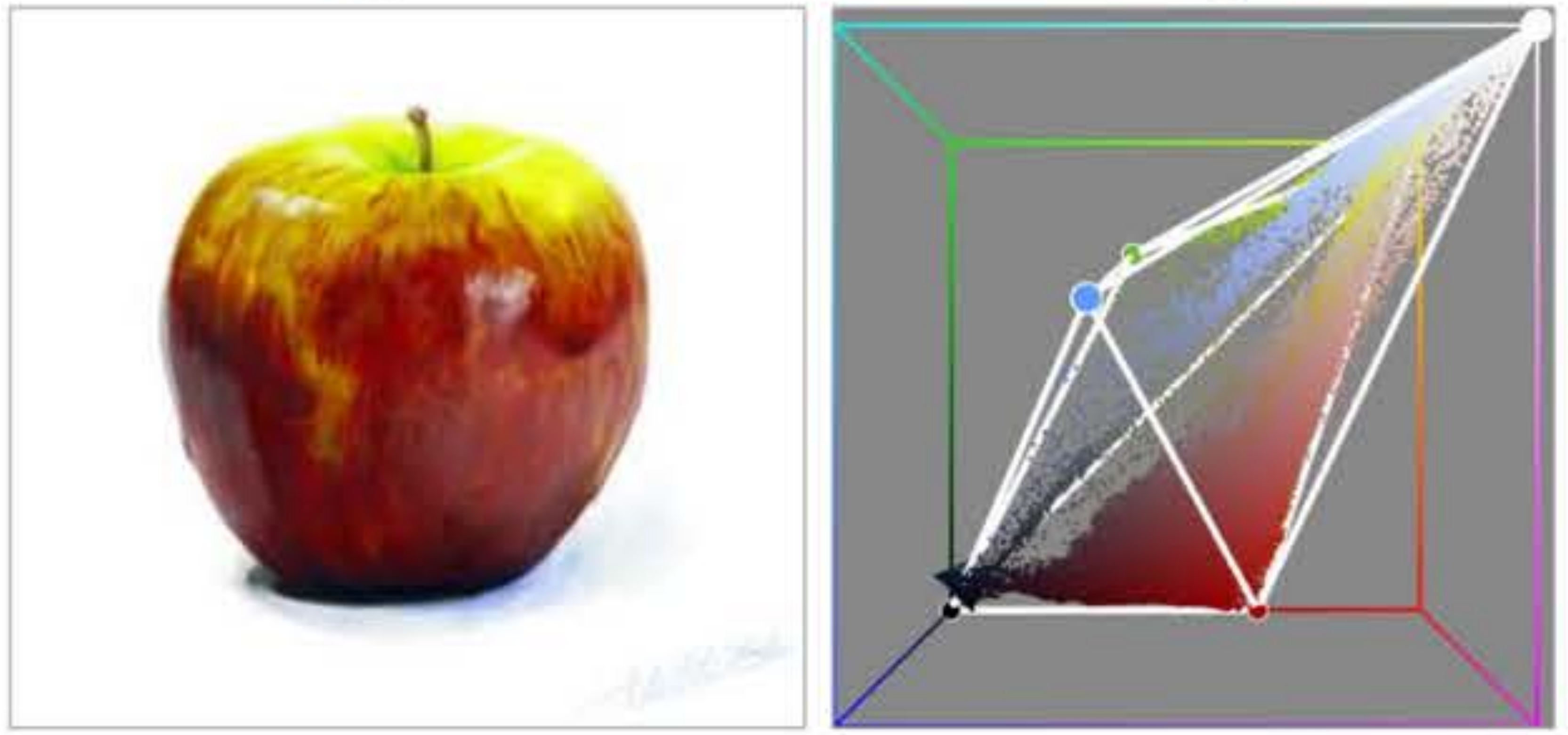
Related Work

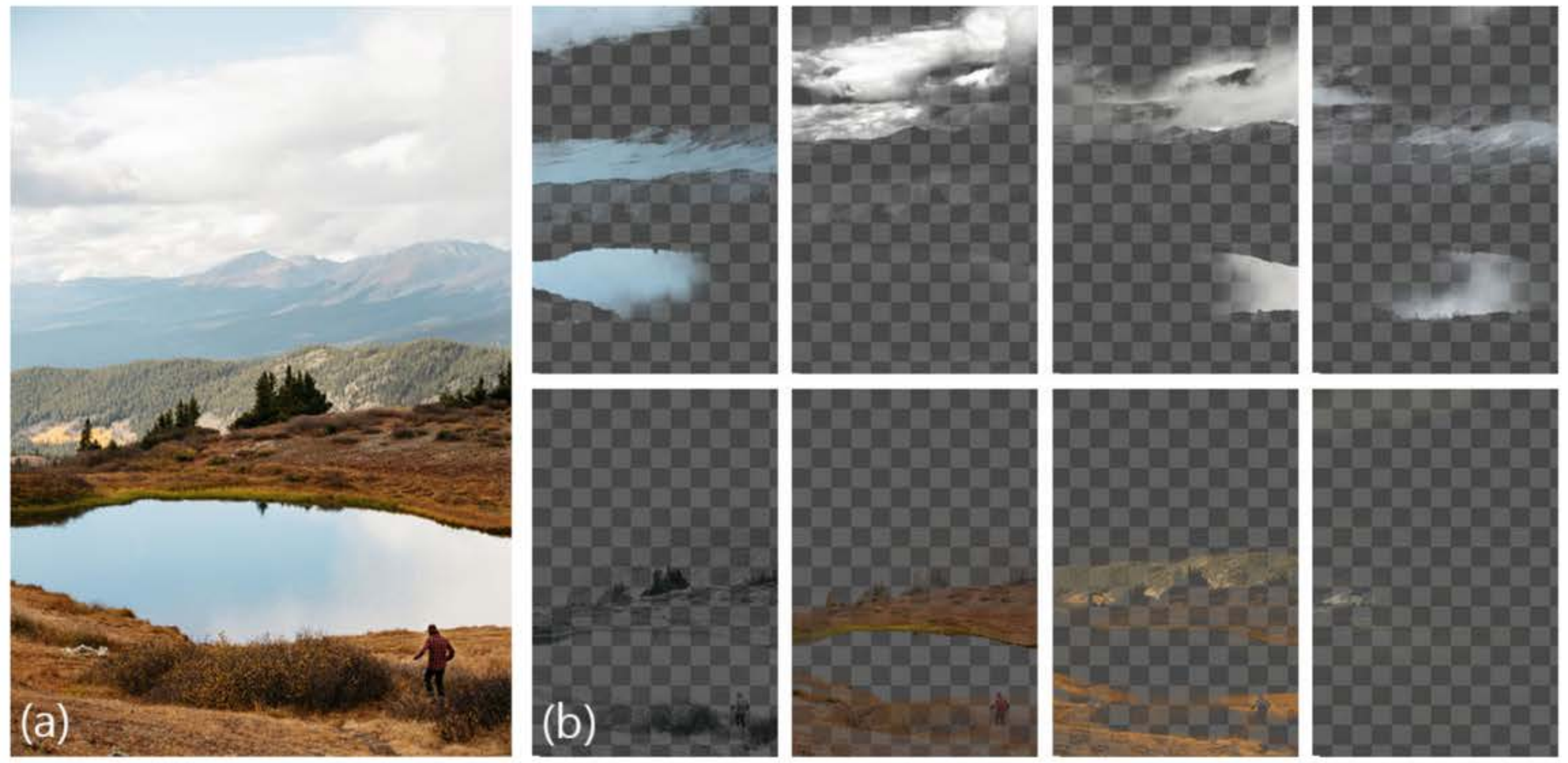
- Physically-based layers
  - Abed et al. [2014]; Tan et al. [2015]; Aharoni-Mack et al. [2017]; Tan et al. [2018].

Absorption
6.0
5.0
4.0
3.0
2.0
1.0
0.0
405
445
485
525
565
605
645
685
Scattering
1.2
1.0
0.8
0.6
0.4
0.2
0.0
405
445
485
525
565
605
645
685
Reflectance
1.0
0.8
0.6
0.4
0.2
0.0
405
445
485
525
565
605
645
685

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

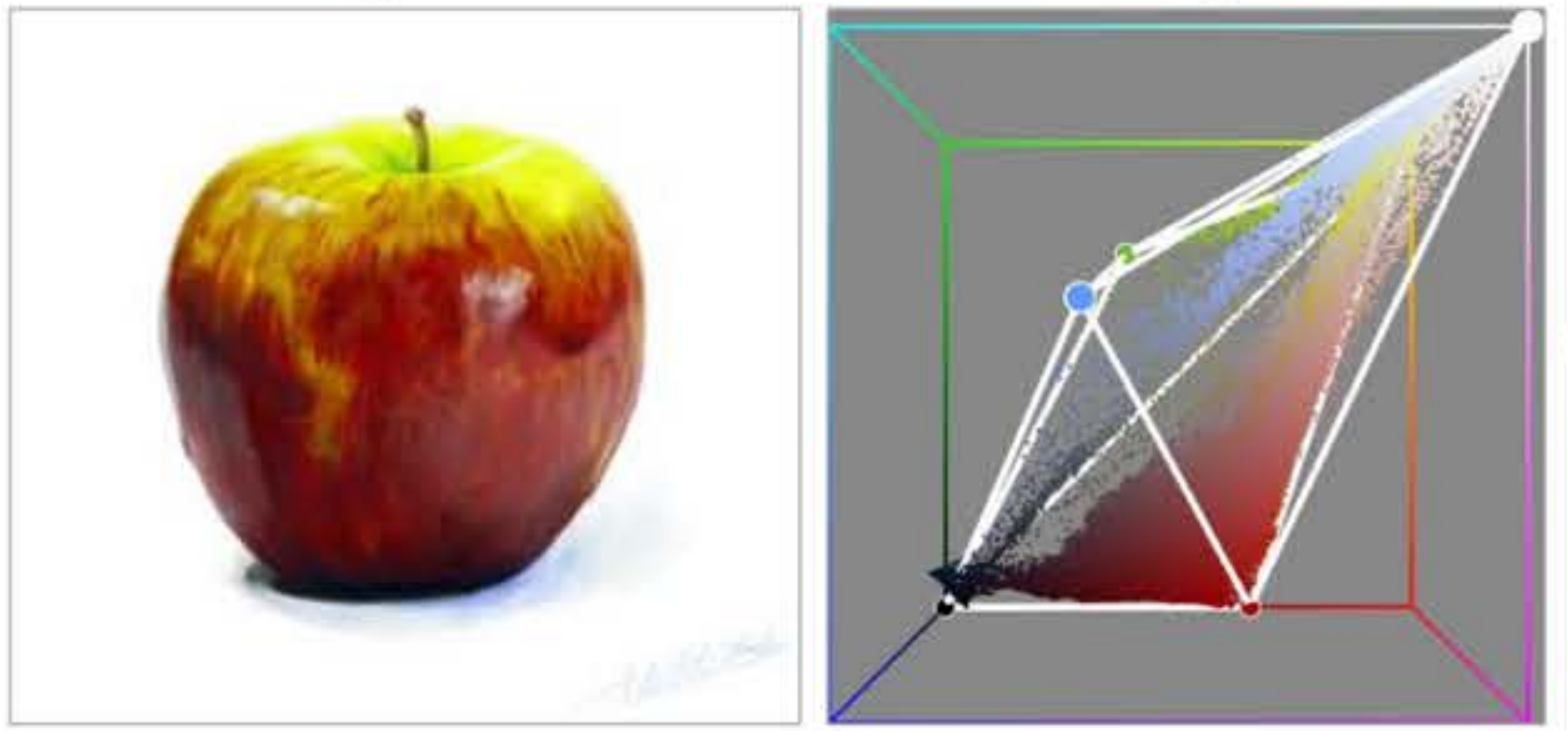
Our approach

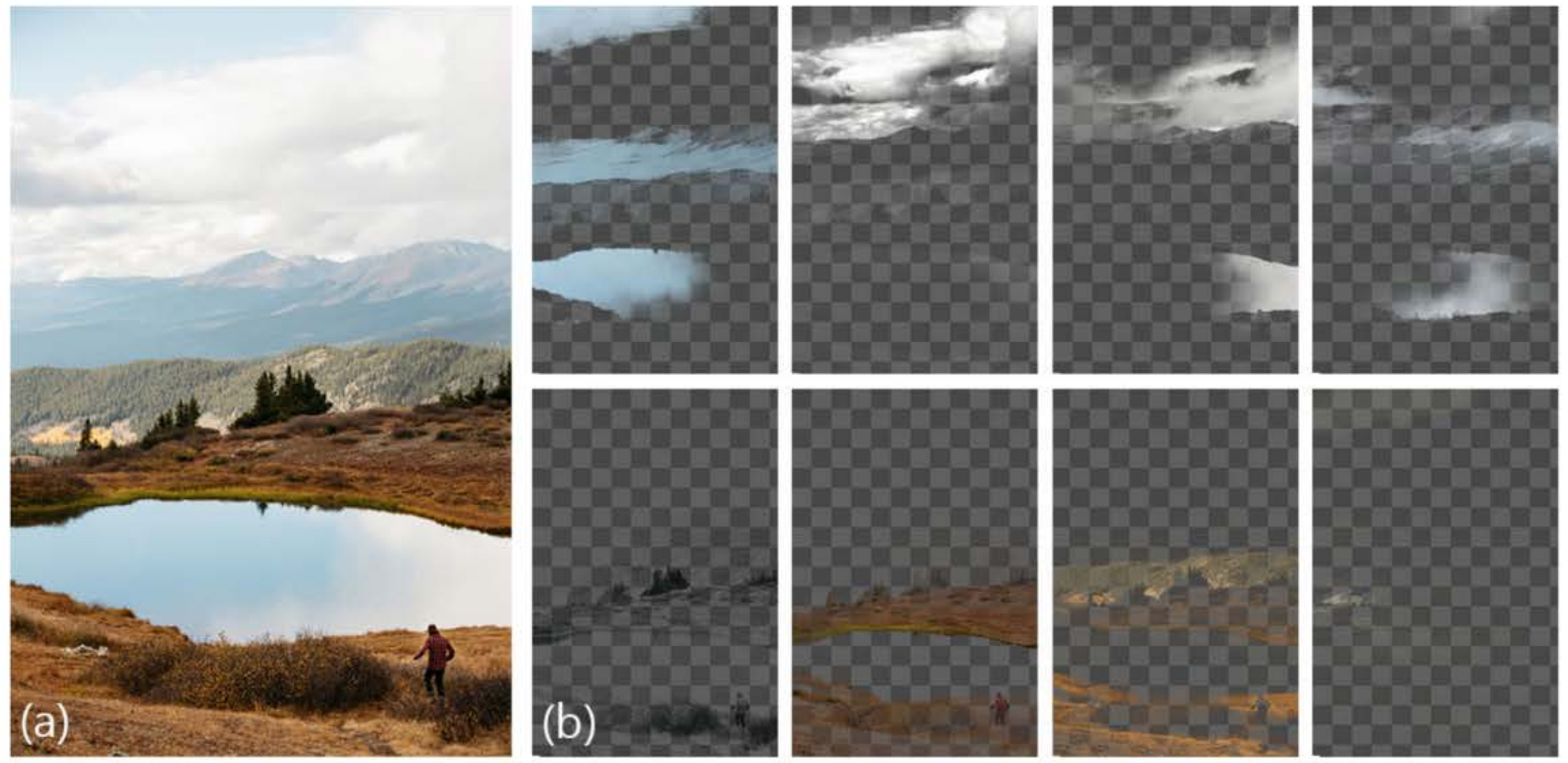




Our approach

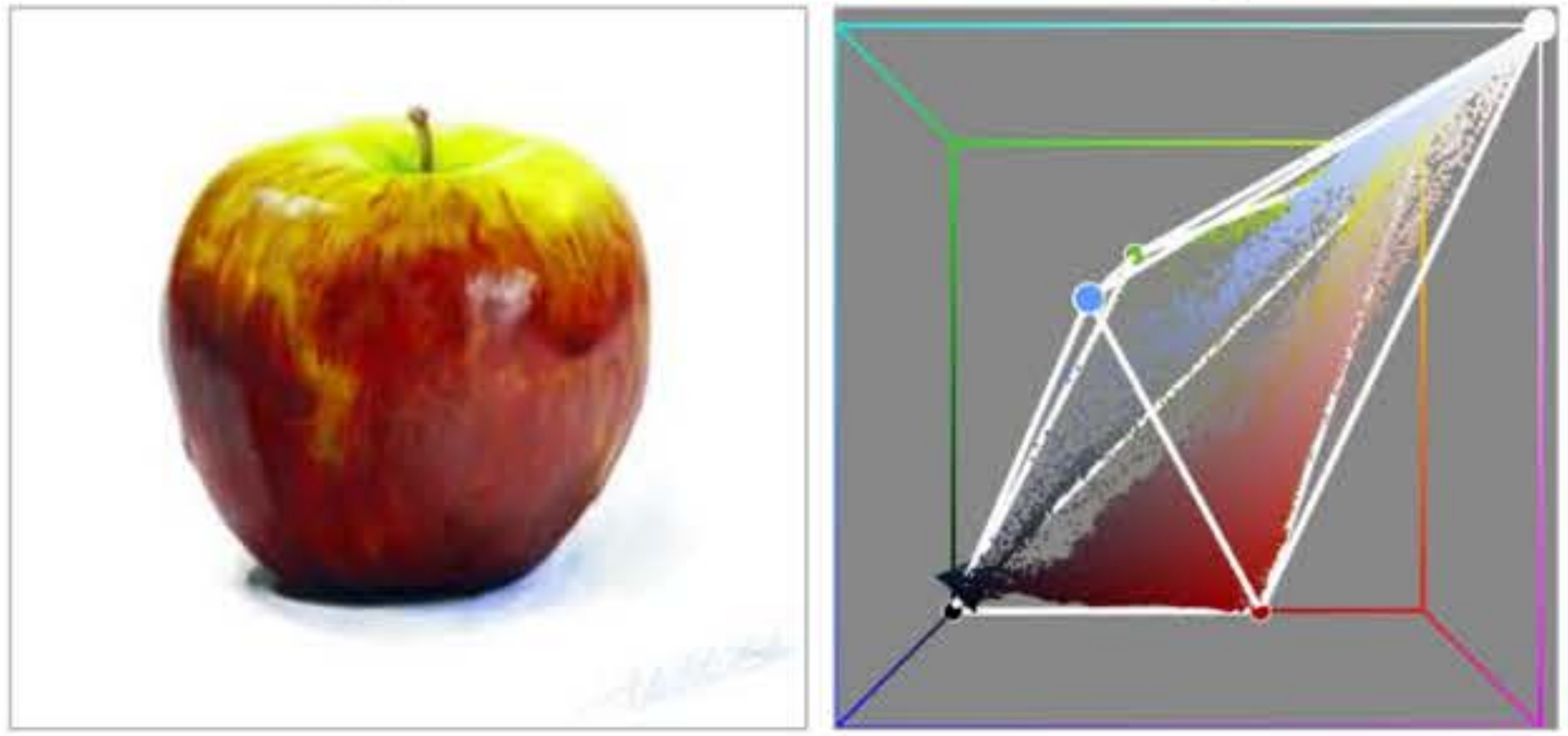
- Geometry-based convex palettes

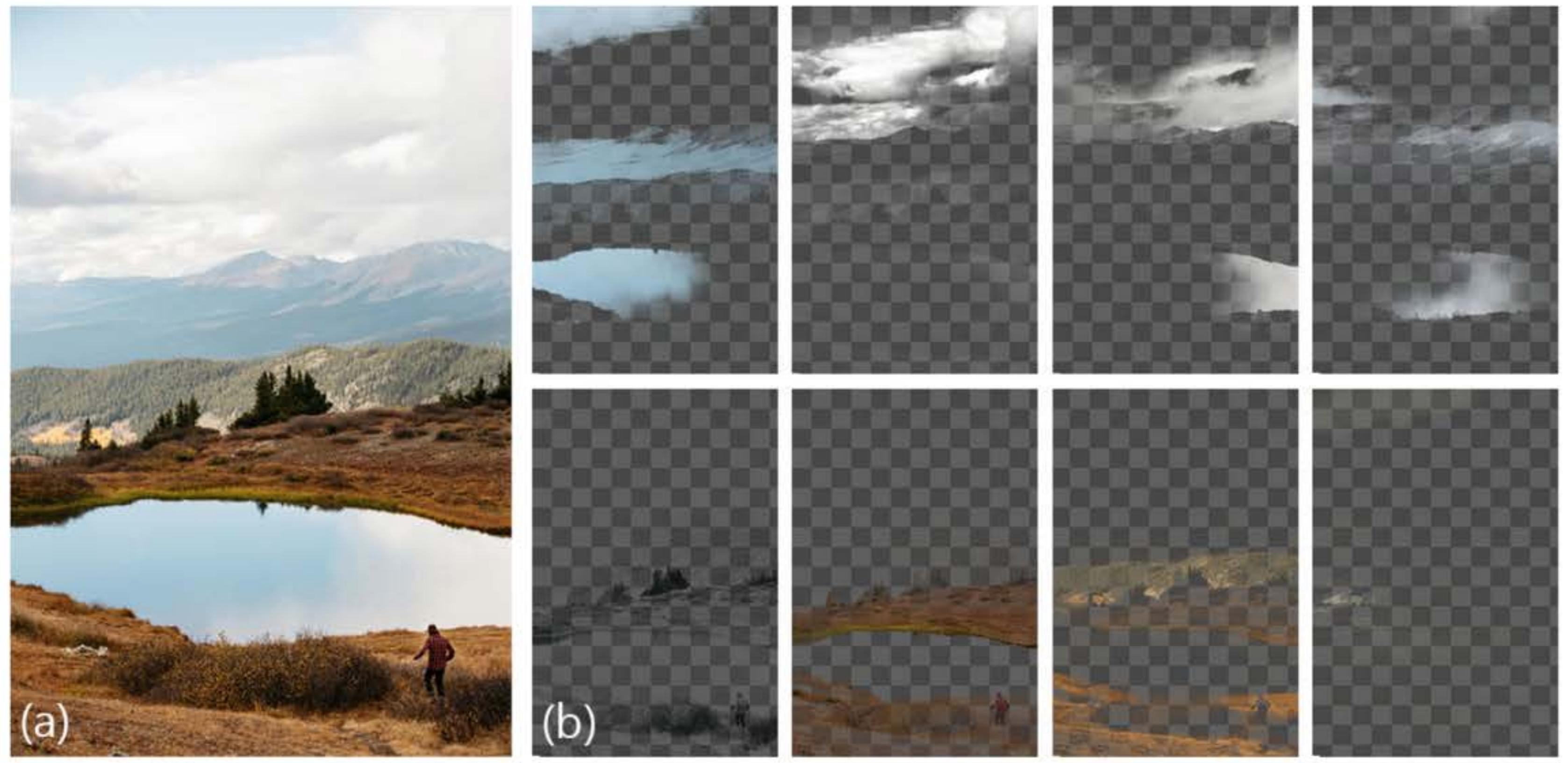




Our approach

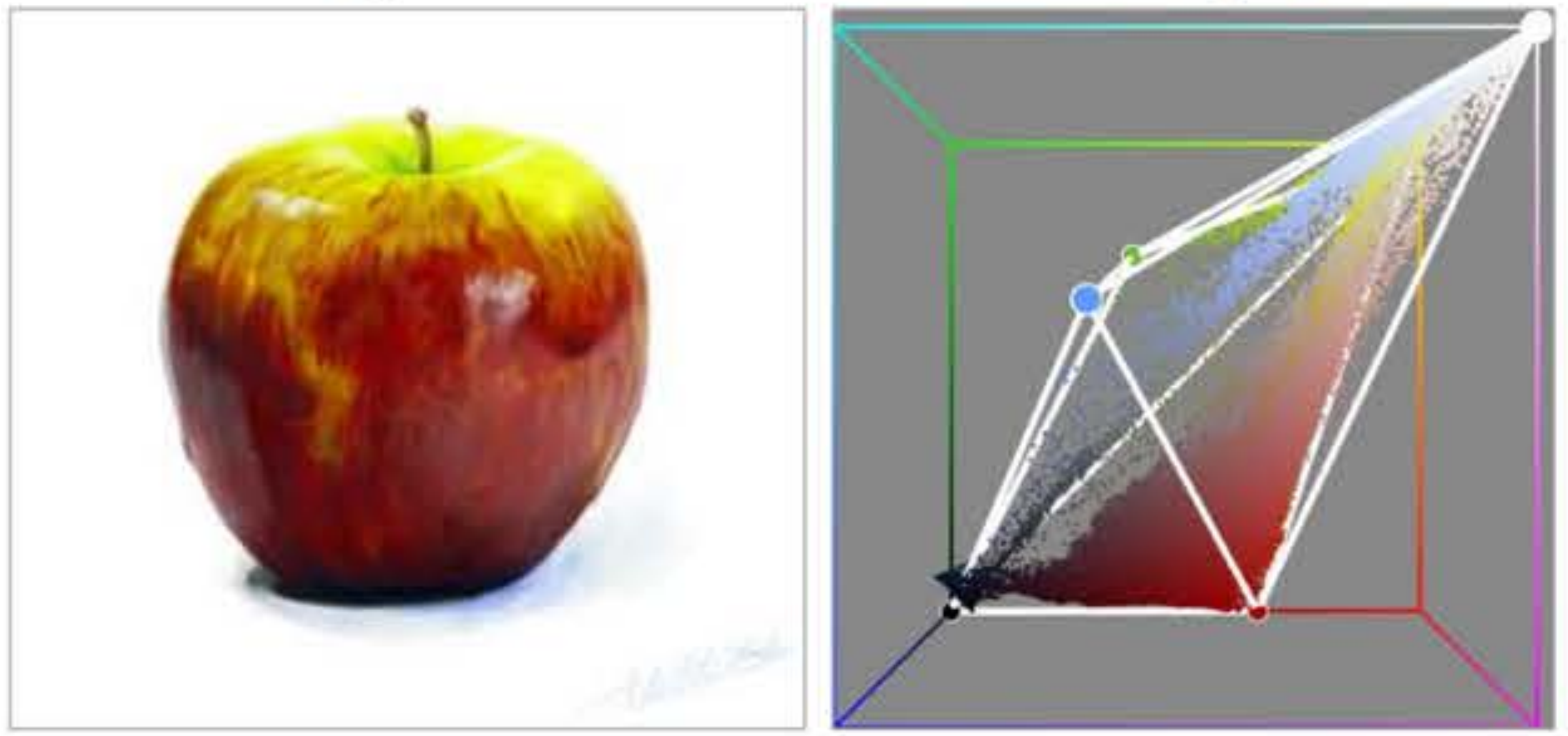
- Geometry-based convex palettes
    - Simpler

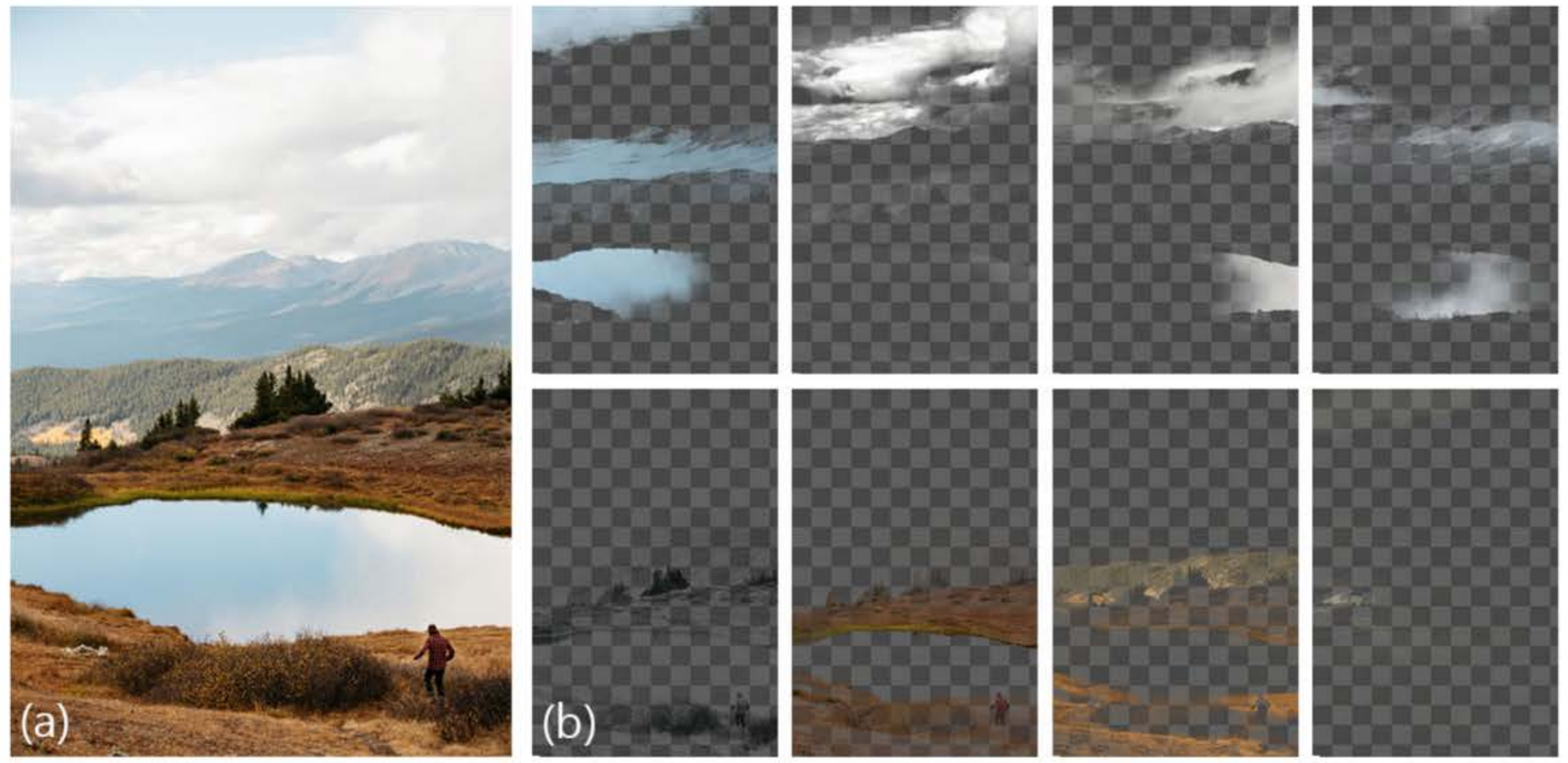




Our approach

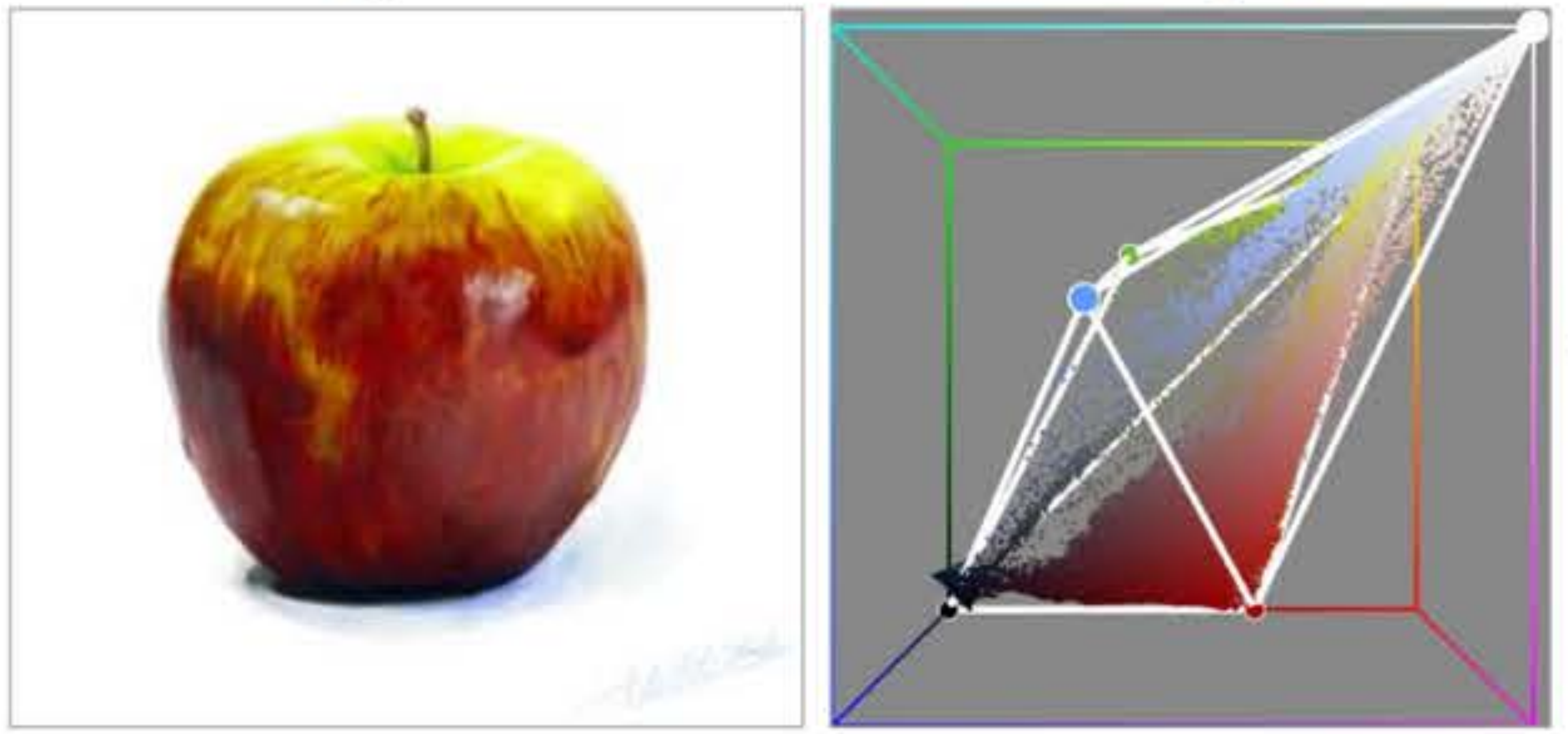
- Geometry-based convex palettes
  - Simpler
  - More general

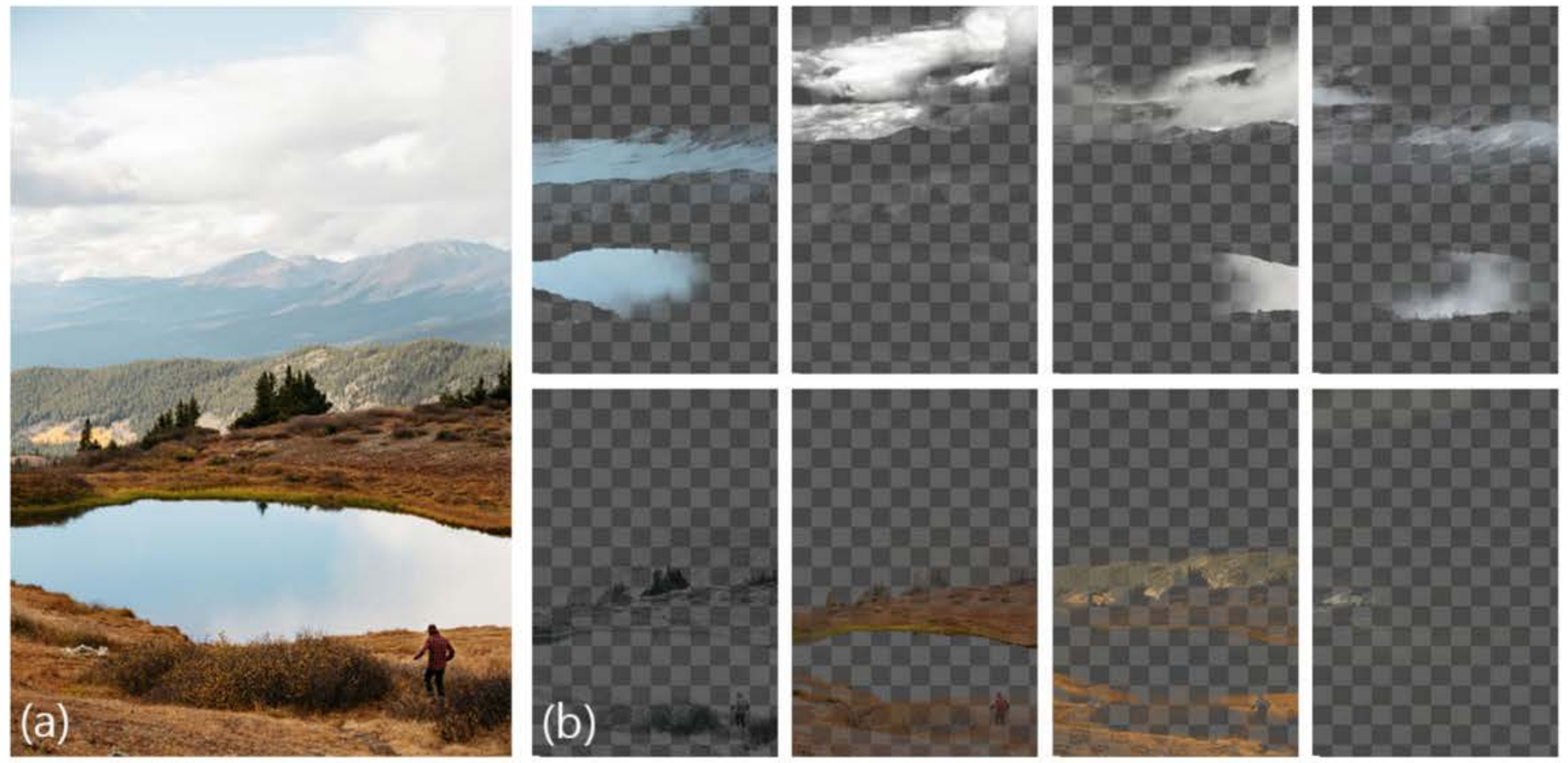




Our approach

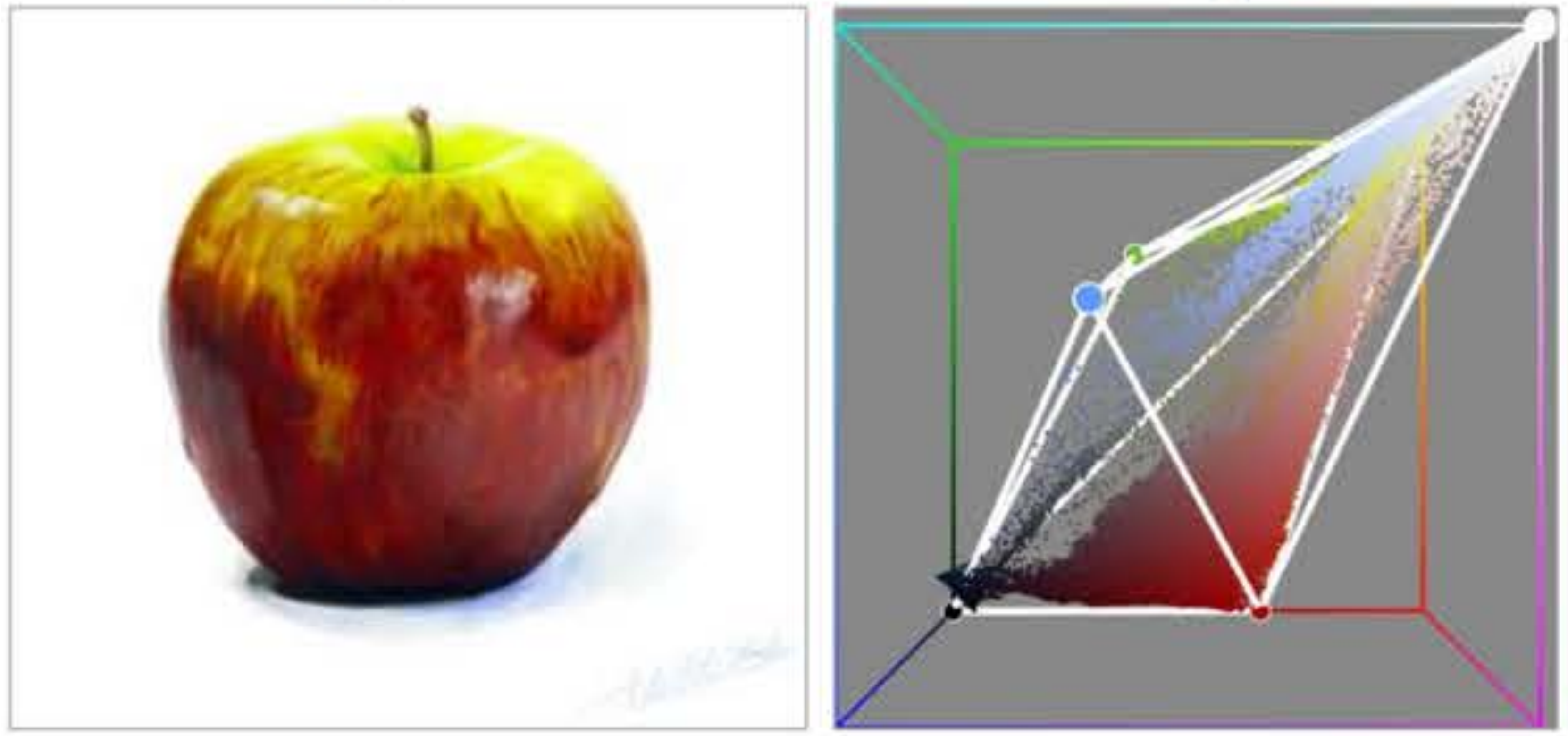
- Geometry-based convex palettes
    - Simpler
    - More general



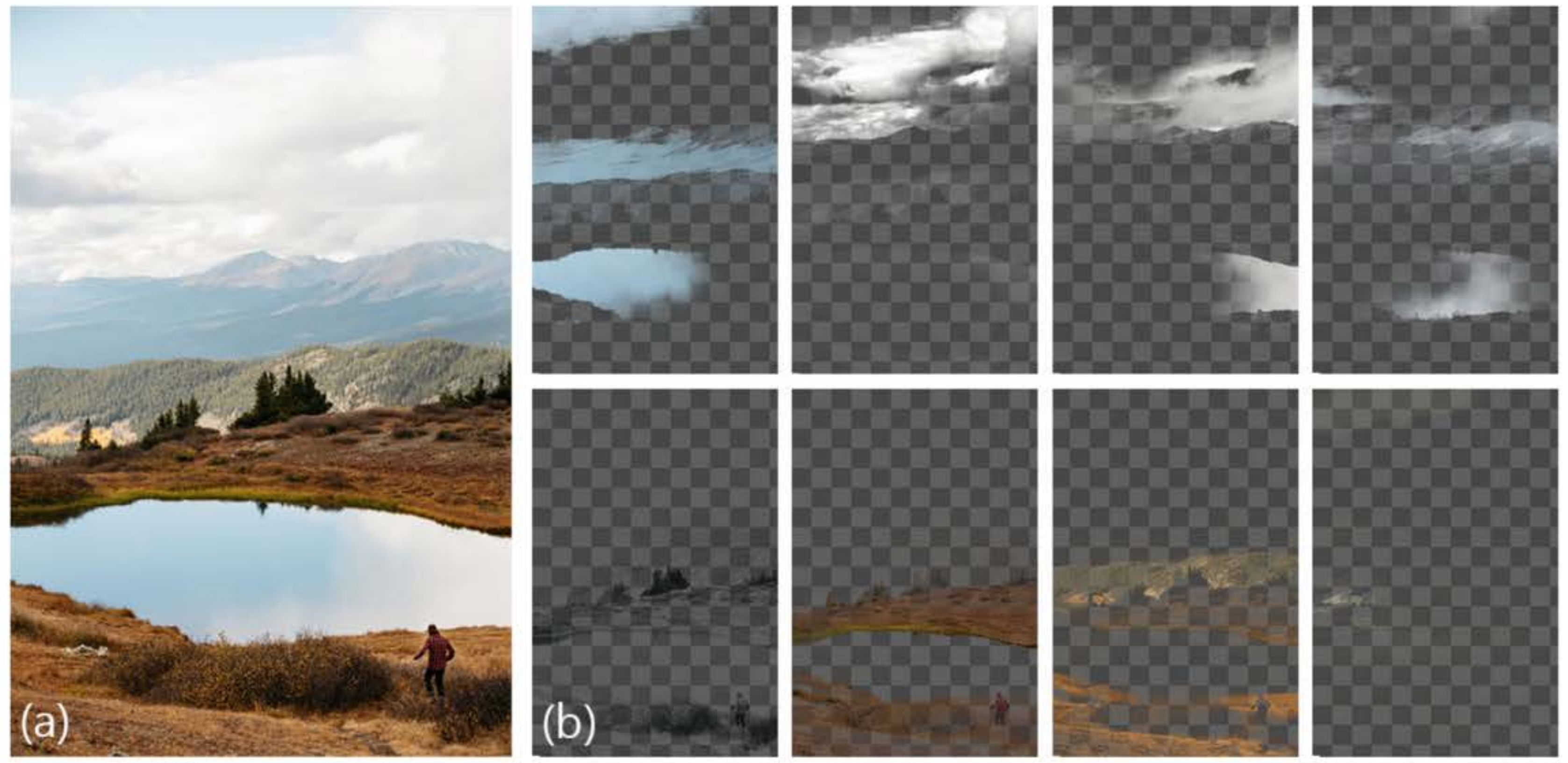


Our approach

- Geometry-based convex palettes
    - Simpler
    - More general

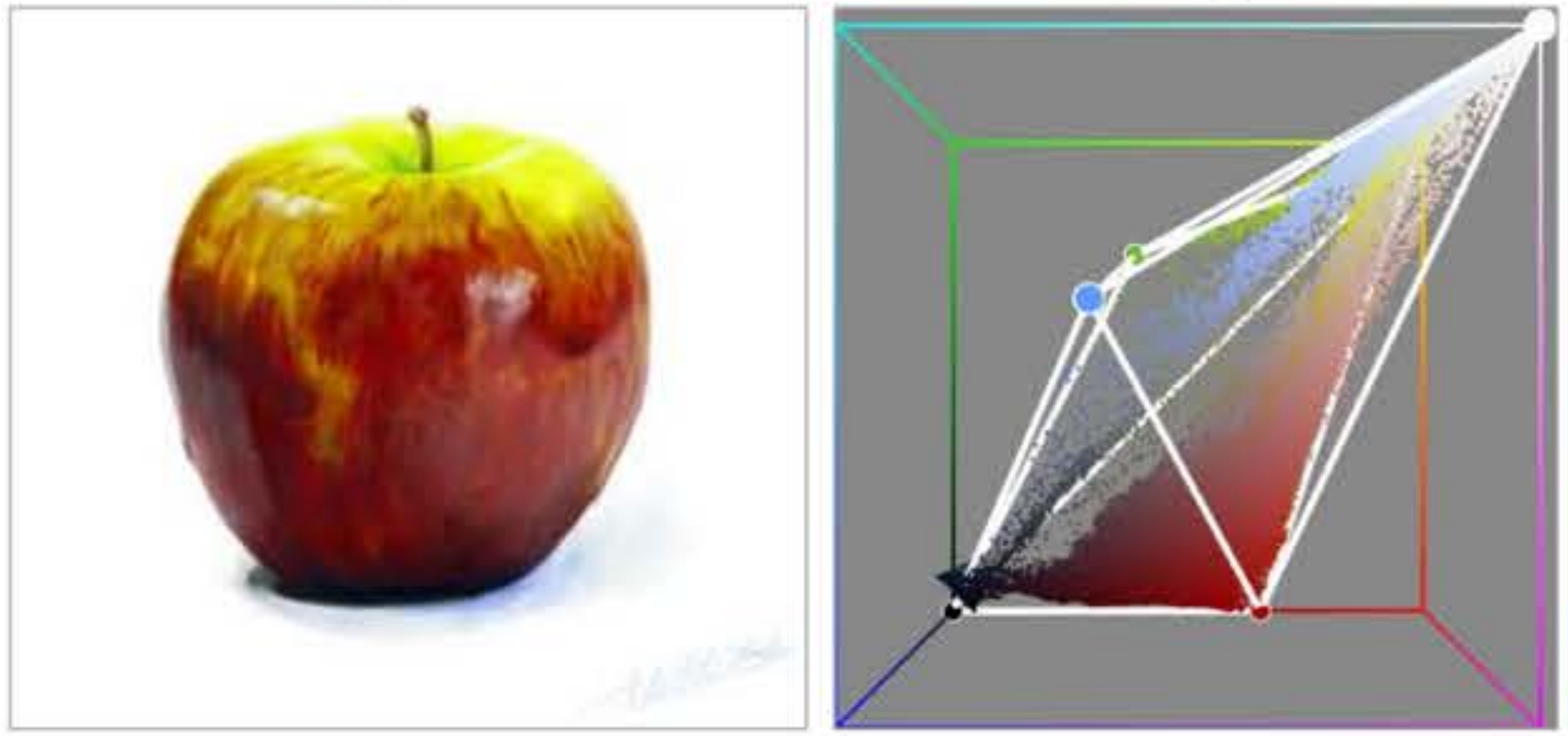


- Additive-mixing layers

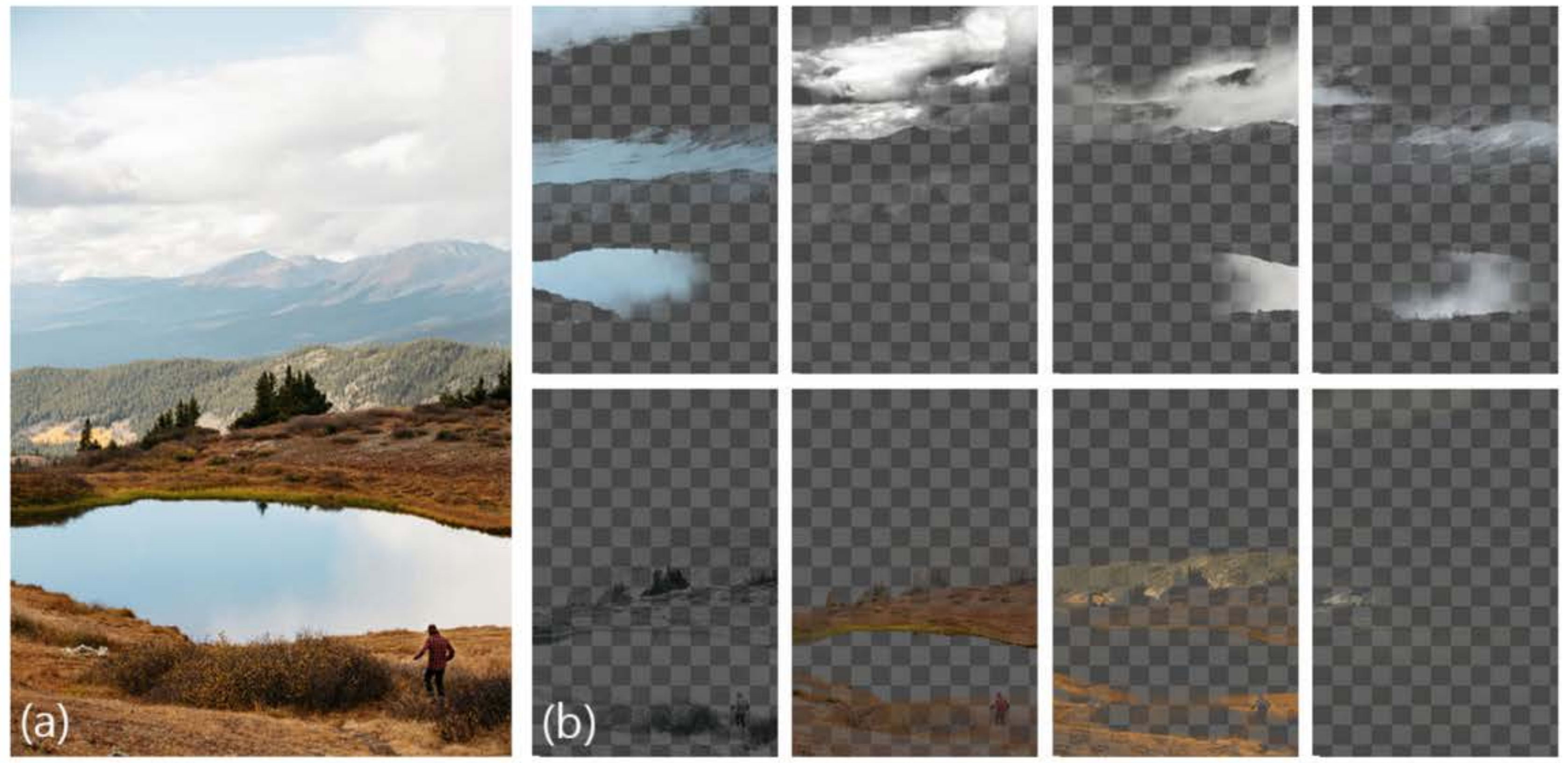


Our approach

- Geometry-based convex palettes
    - Simpler
    - More general

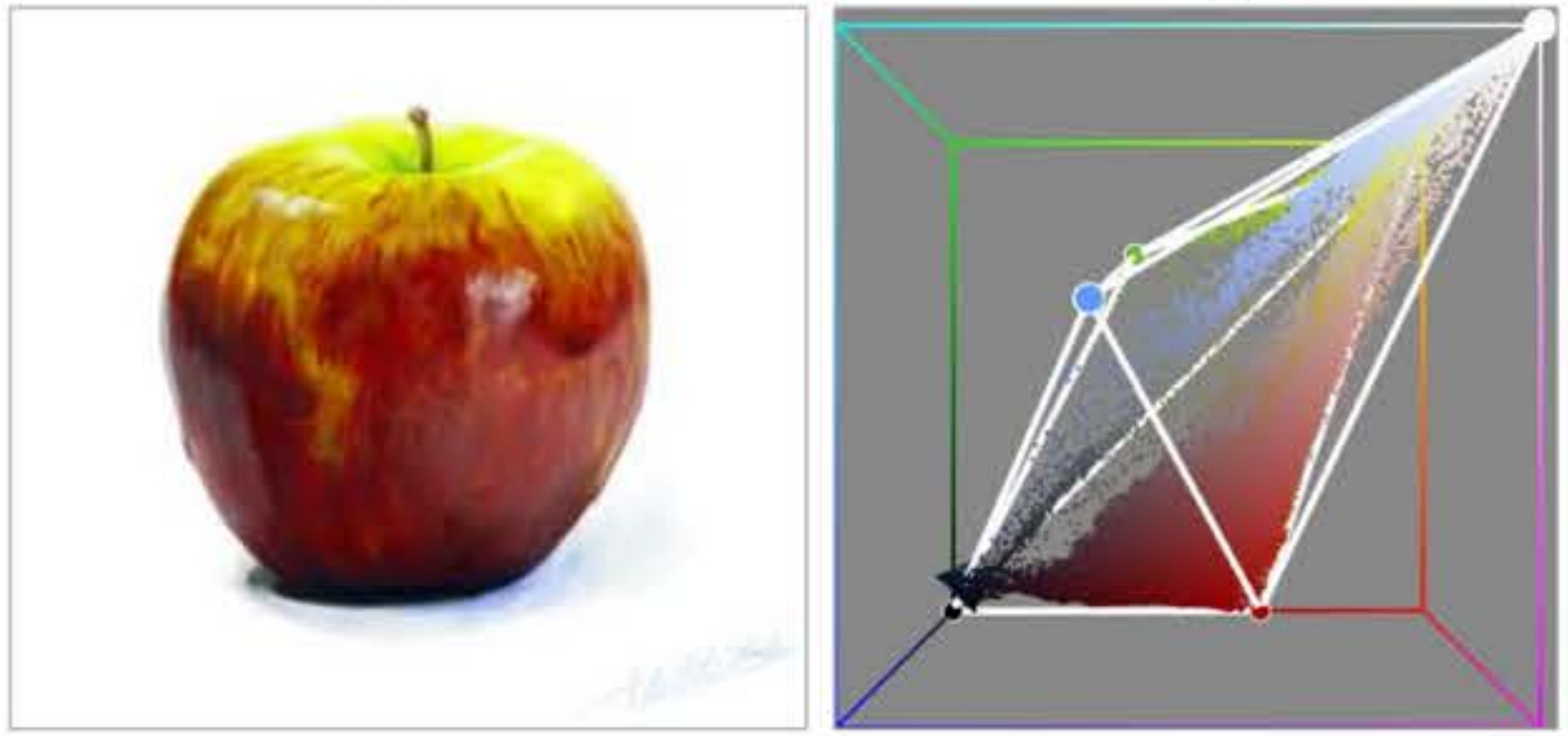


- Additive-mixing layers
    - Single colors

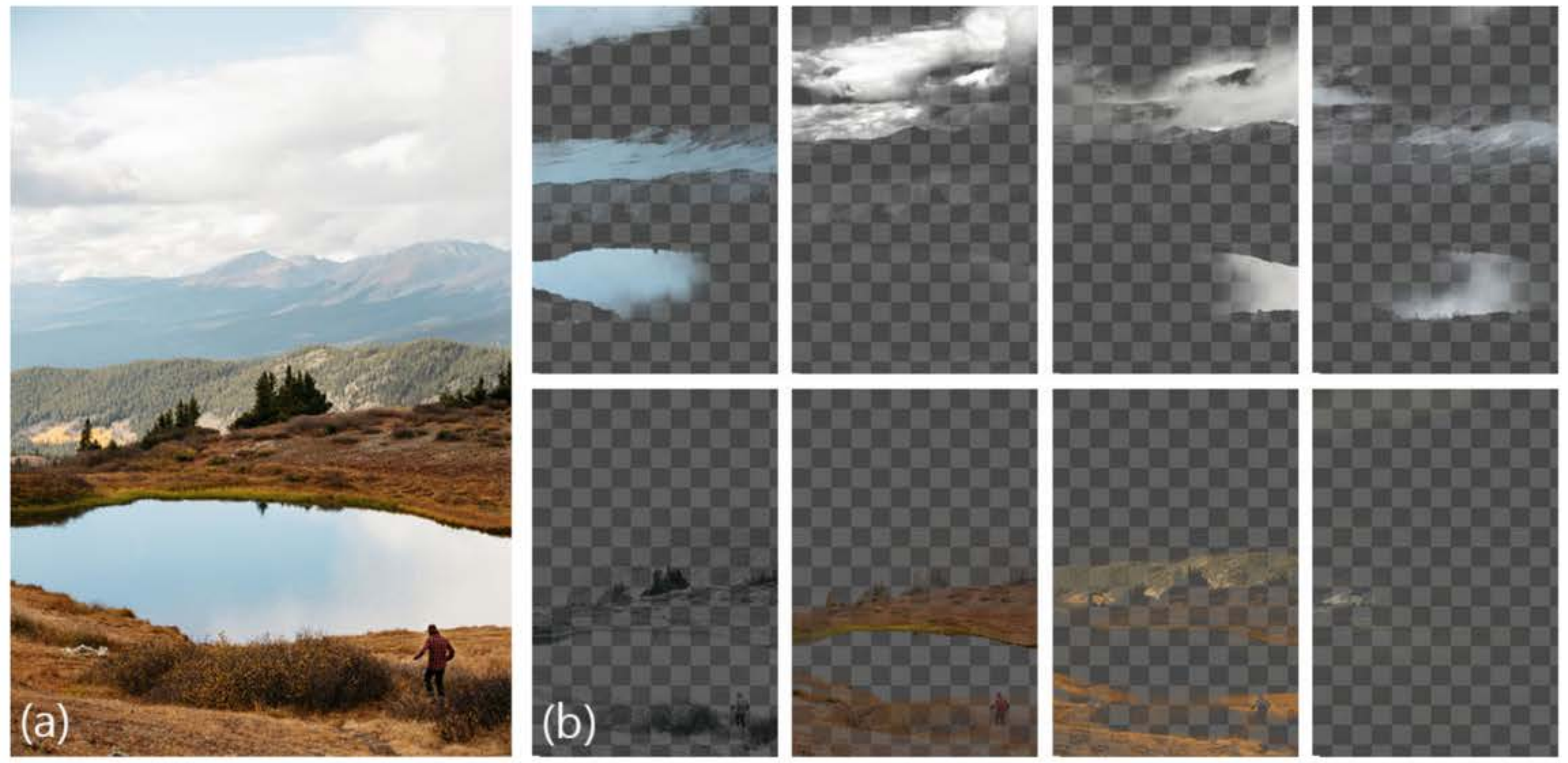


Our approach

- Geometry-based convex palettes
  - Simpler
  - More general

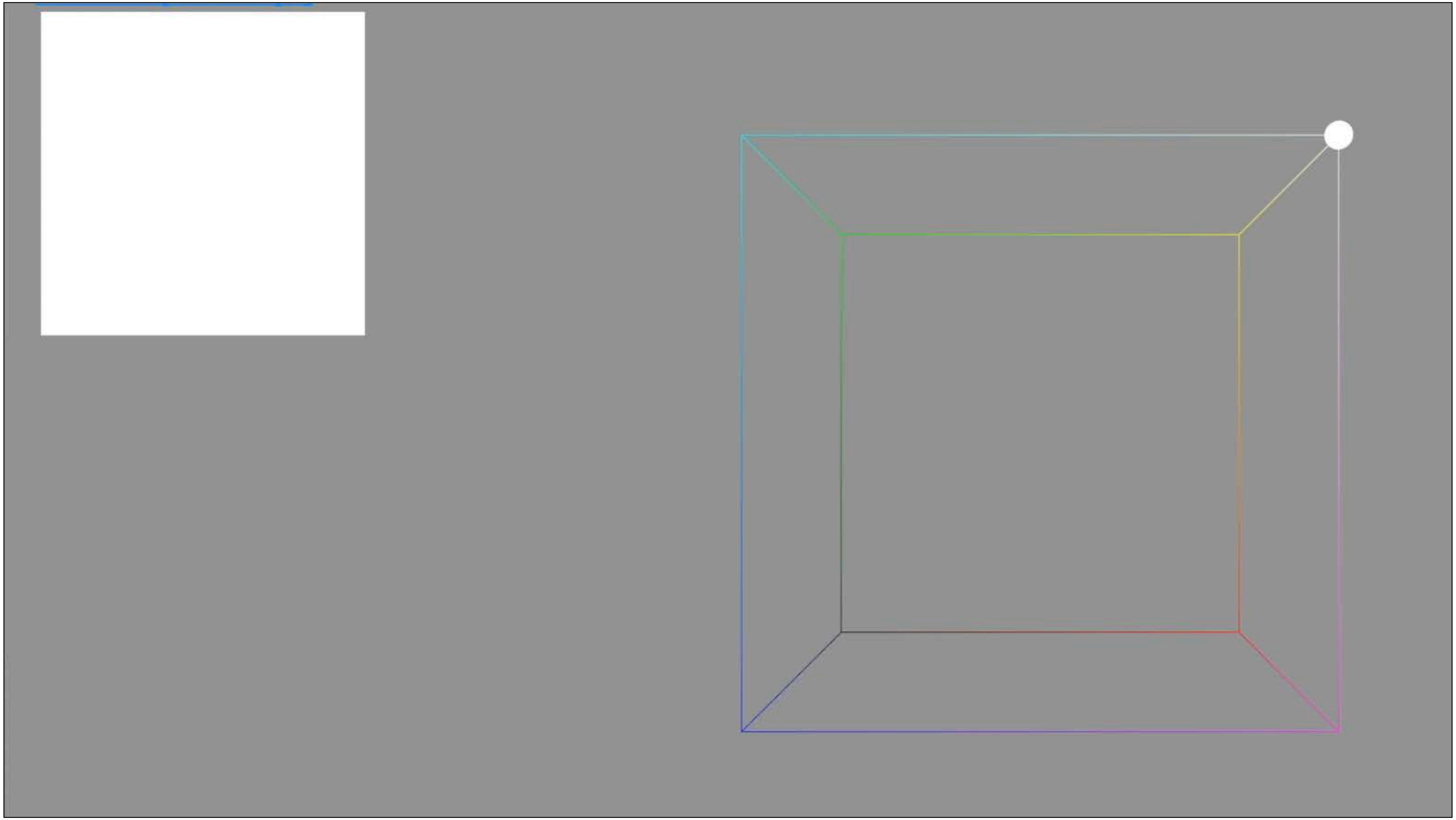


- Additive-mixing layers
  - Single colors
  - More general

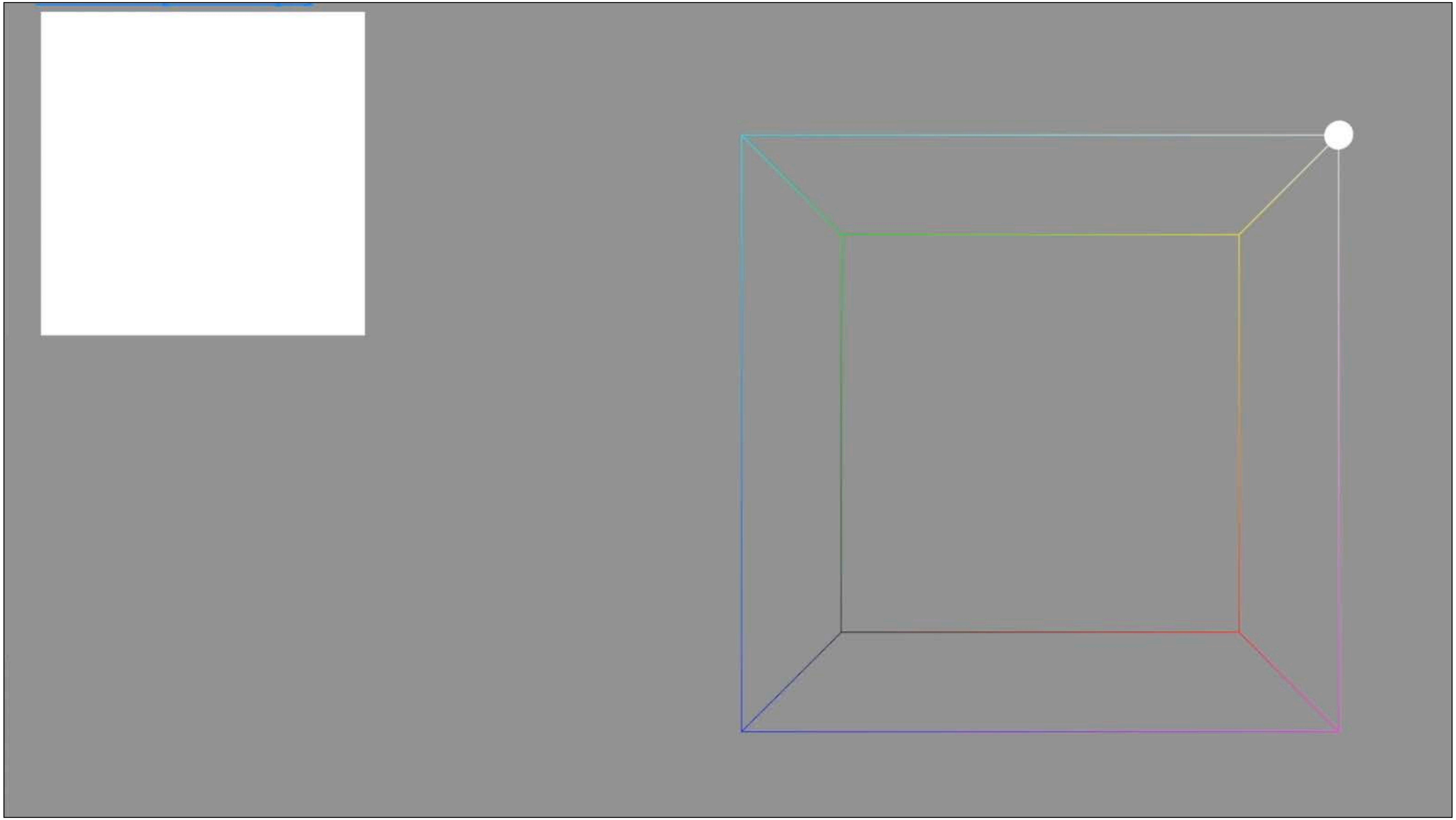


Palette extraction

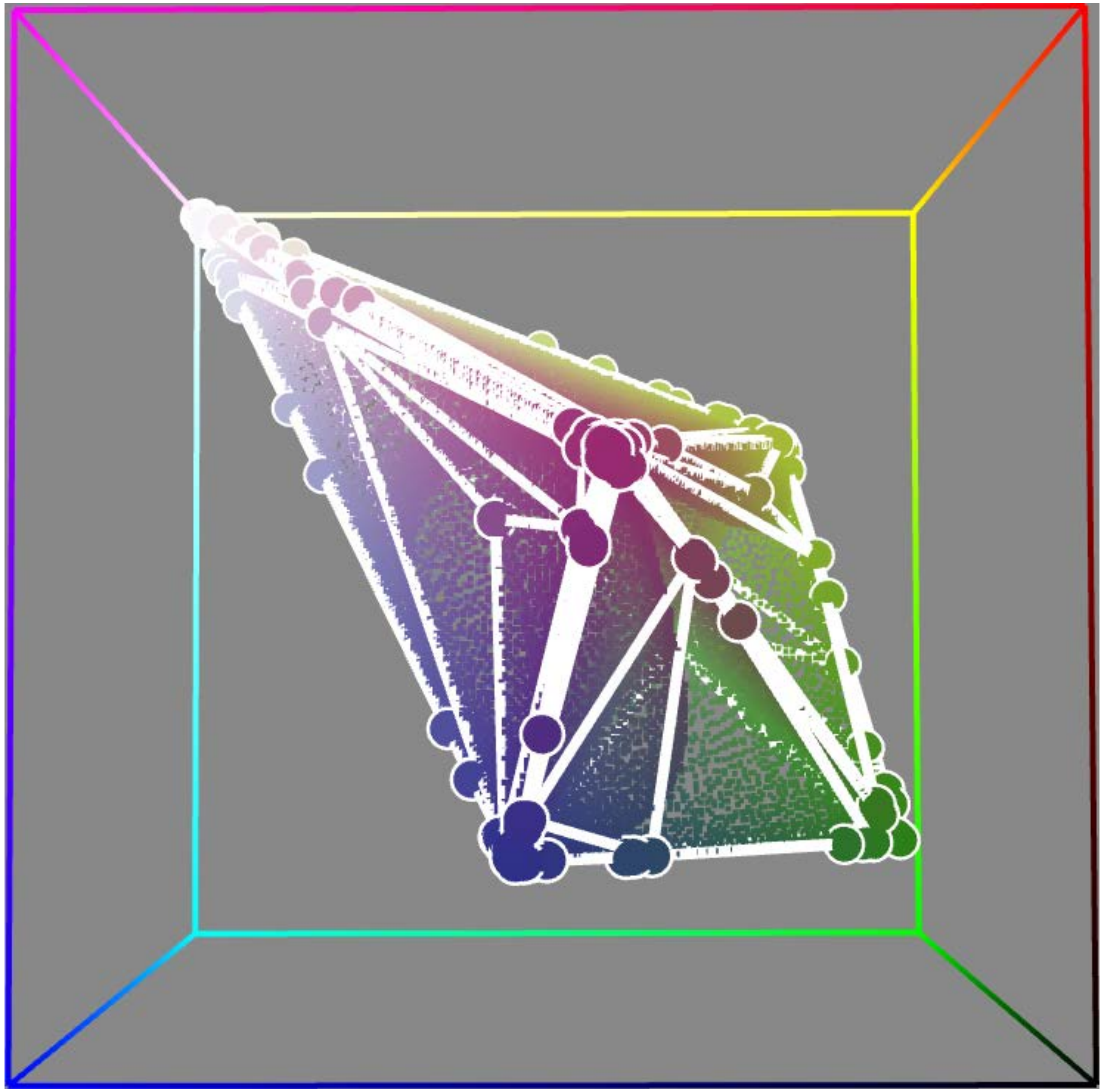
Reminder: Convex Hulls in Color-Space



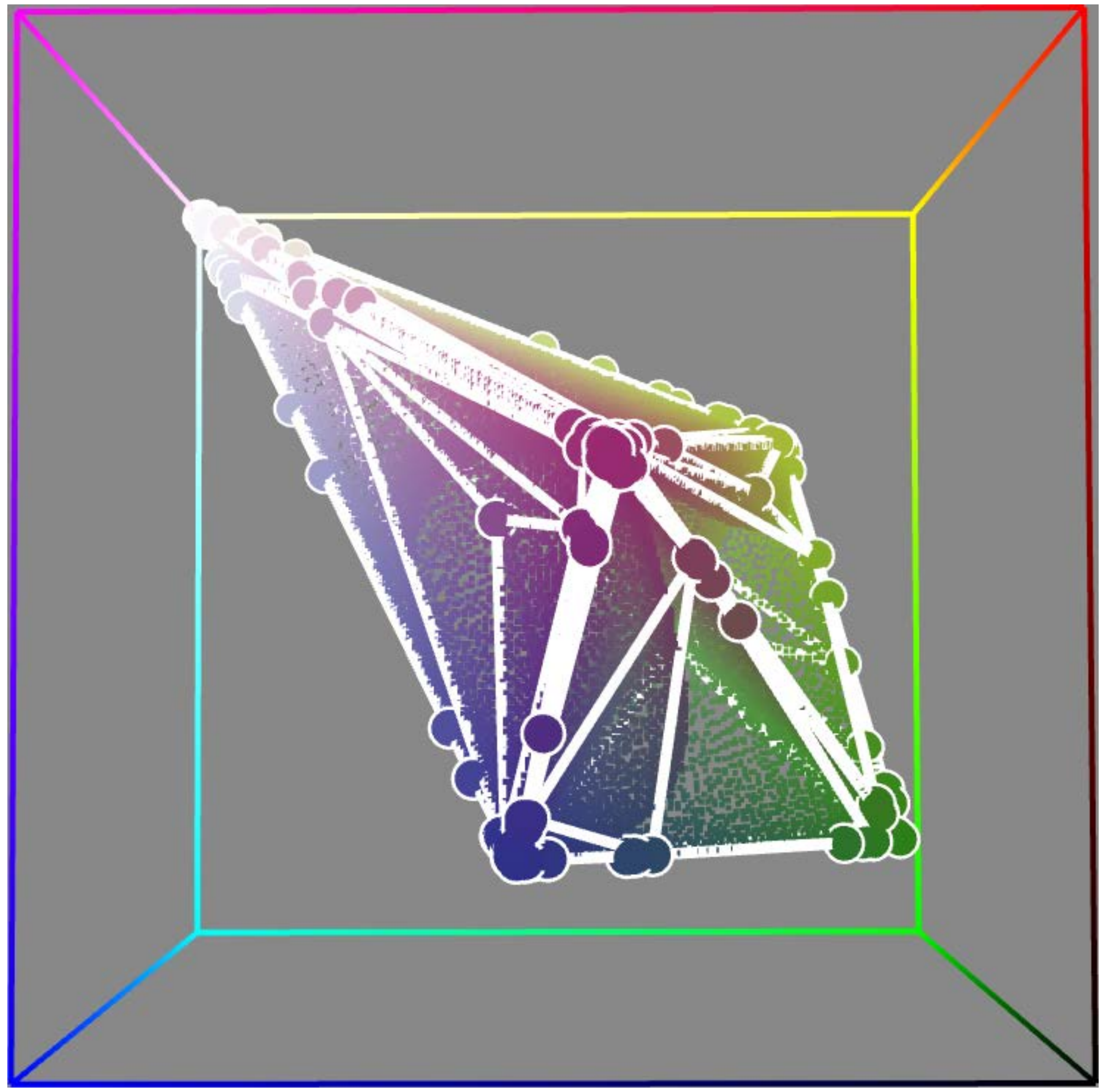
Reminder: Convex Hulls in Color-Space



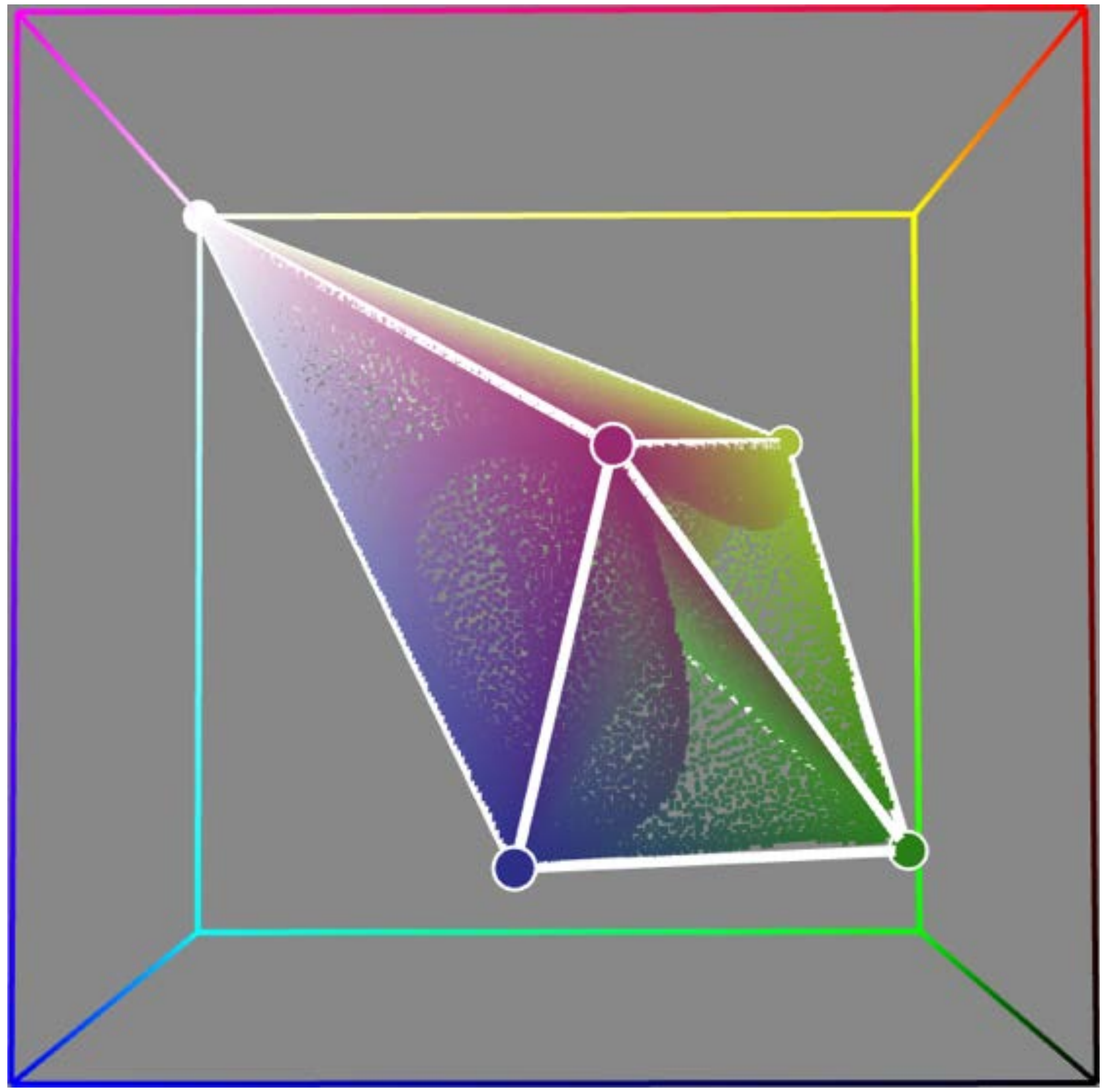
Convex Hull in RGB-space



Convex Hull simplification [Tan et al. 2016]

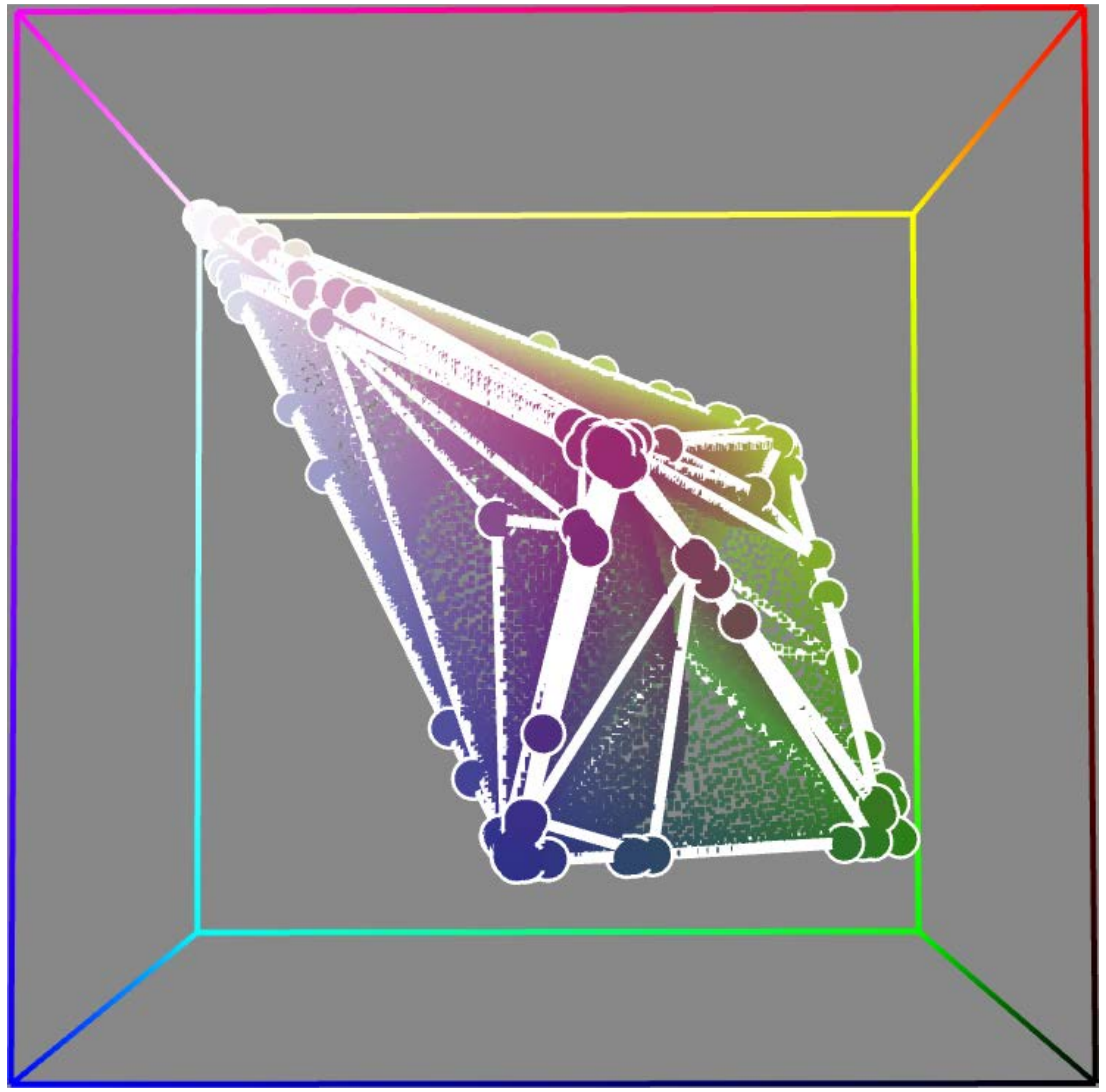


convex hull



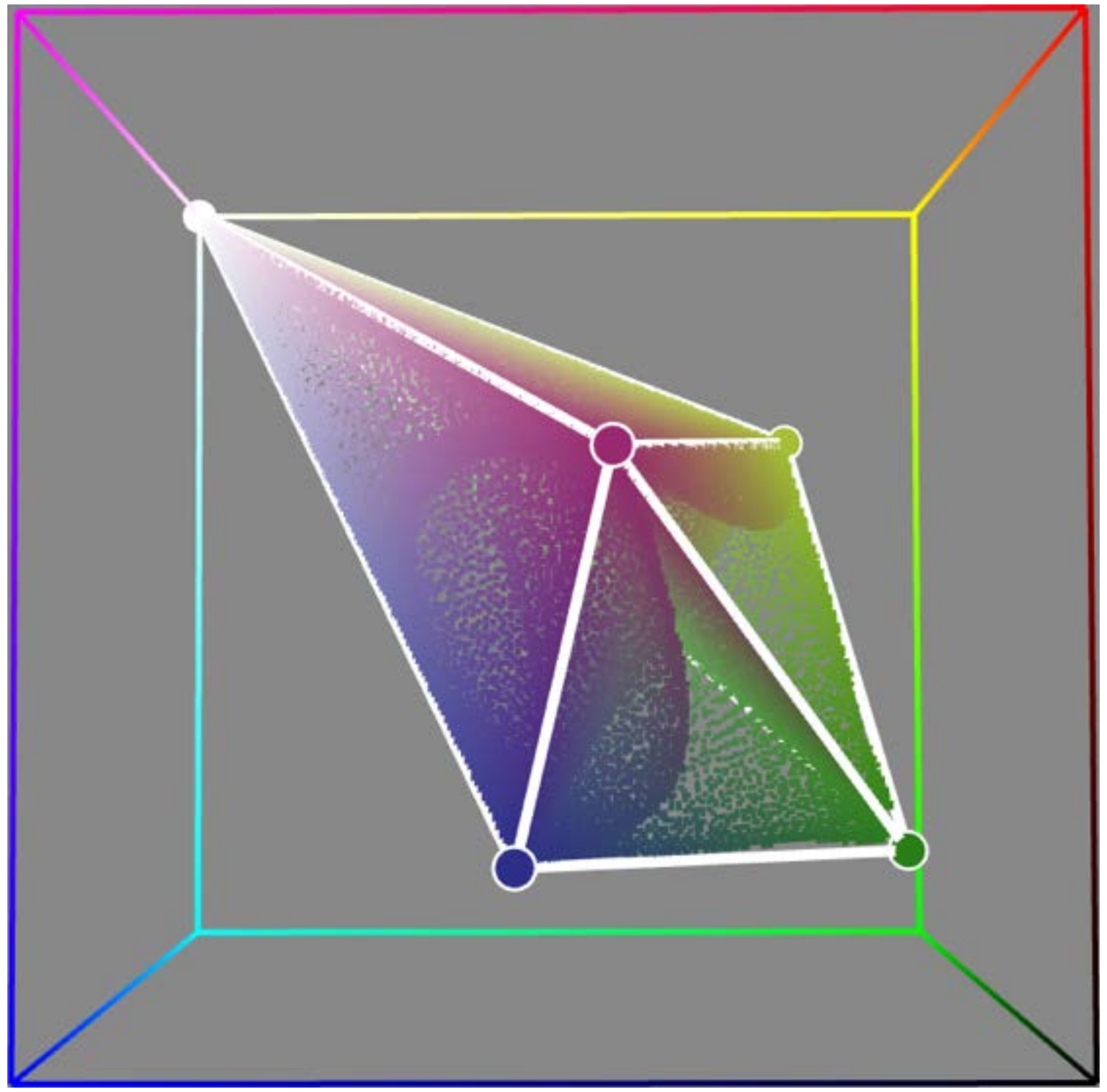
simplified convex hull

Convex Hull simplification [Tan et al. 2016]



convex hull

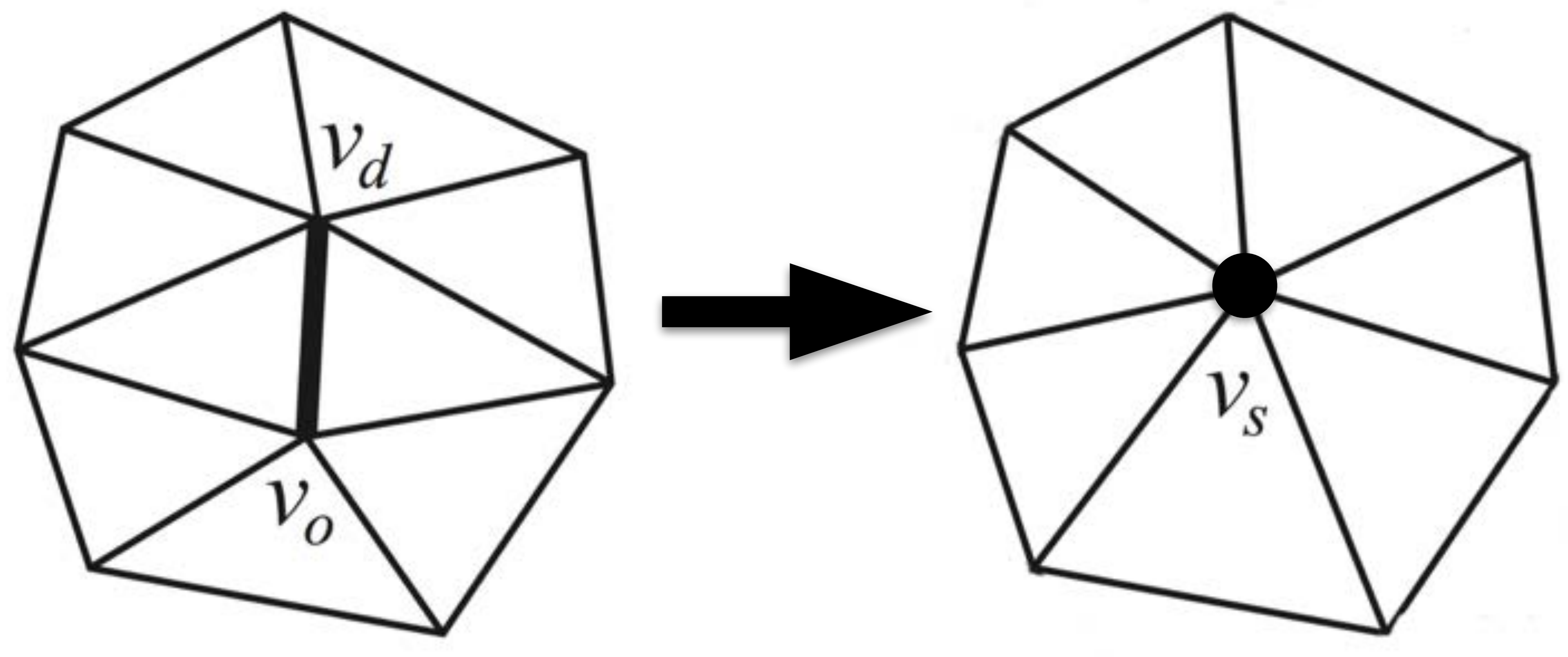
Iteratively collapse edges with a modified Progressive Hull method



simplified convex hull

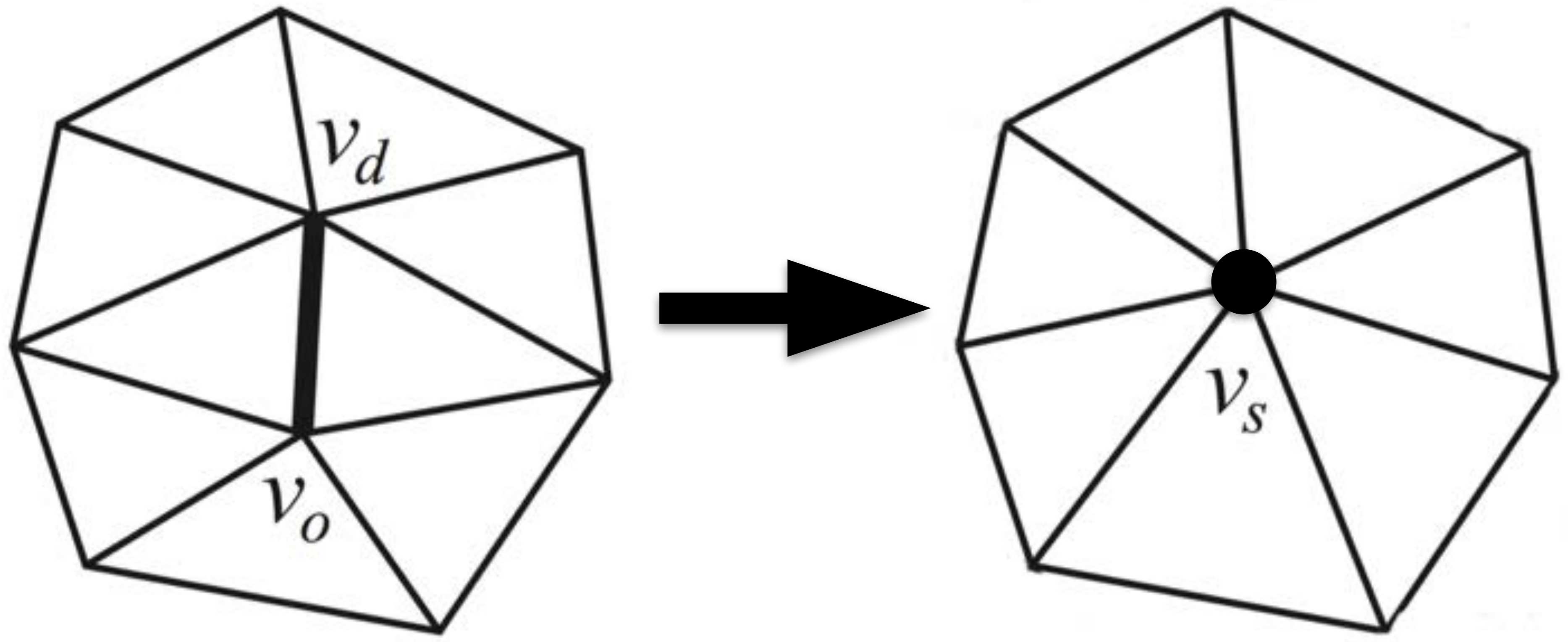
Progressive Hull [Sander et al. 2000]

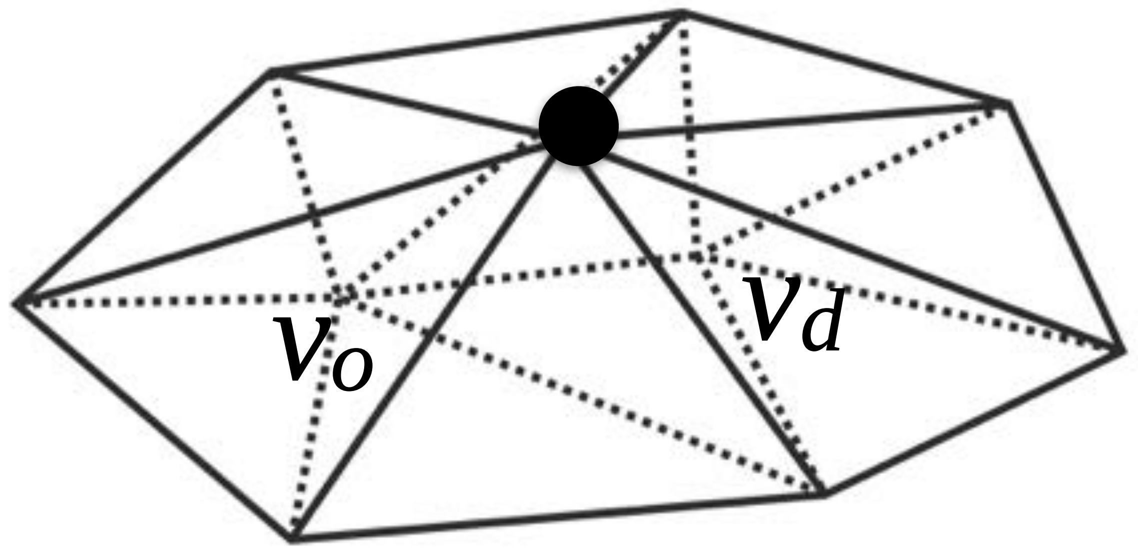
- Greedily collapse edges whose new vertex position adds the smallest additional volume.



Progressive Hull [Sander et al. 2000]

- Greedily collapse edges whose new vertex position adds the smallest additional volume.
- New vertex position guarantees that volume expands (linear constraint)



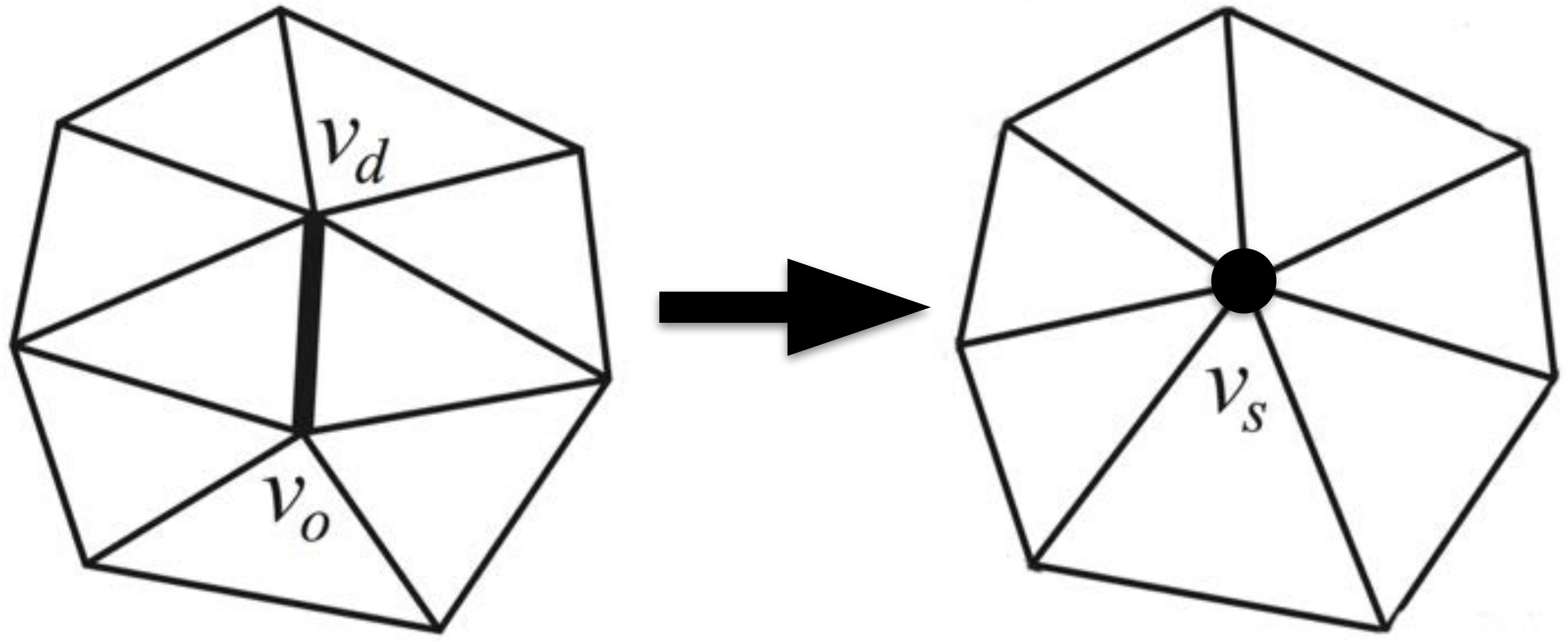


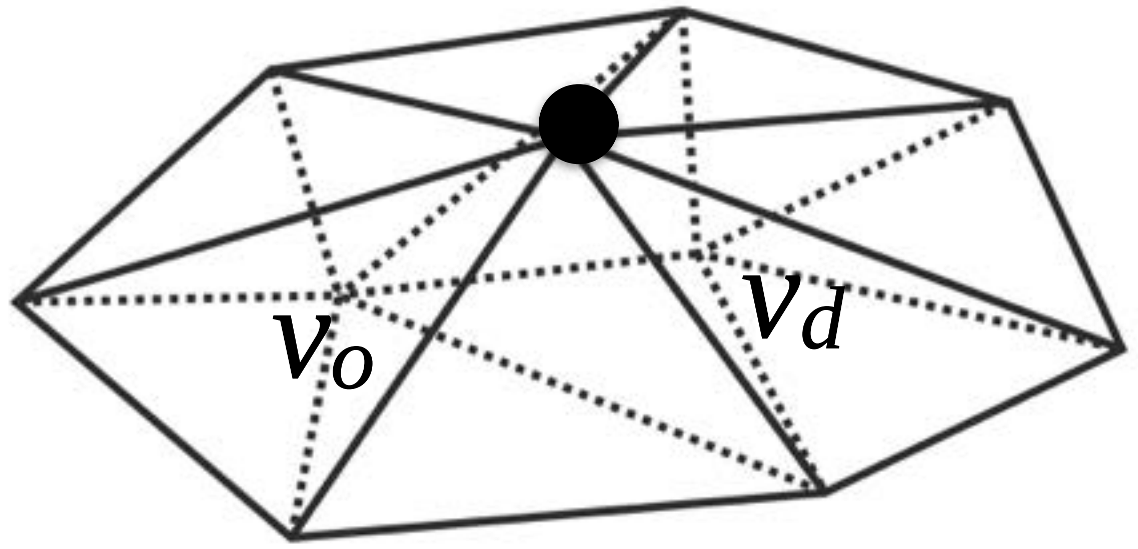
Progressive Hull [Sander et al. 2000]

- Greedily collapse edges whose new vertex position adds the smallest additional volume.

- New vertex position guarantees that volume expands (linear constraint)

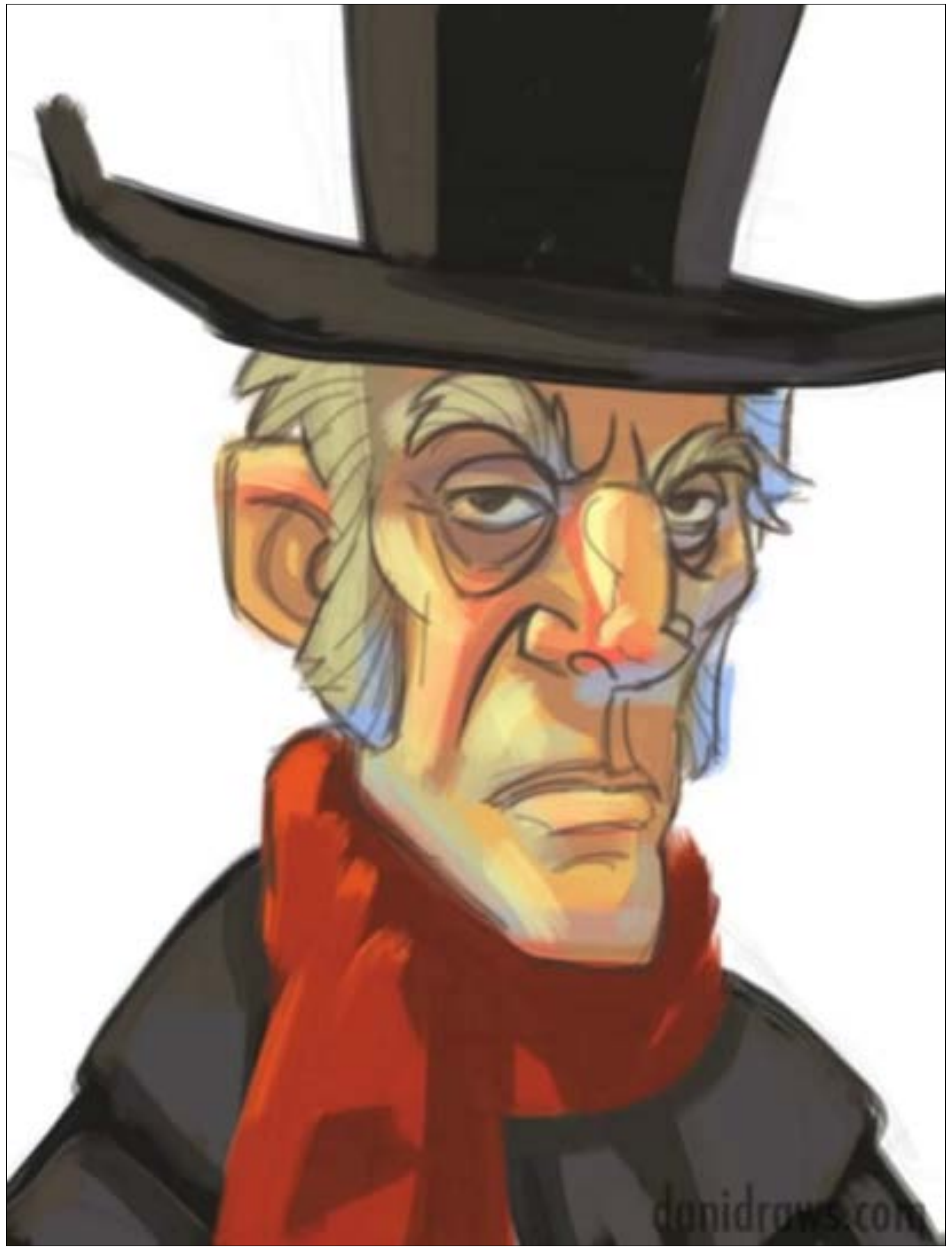
- We modify the algorithm: choose the new vertex that minimizes the distance to incident faces of the collapsing edge.

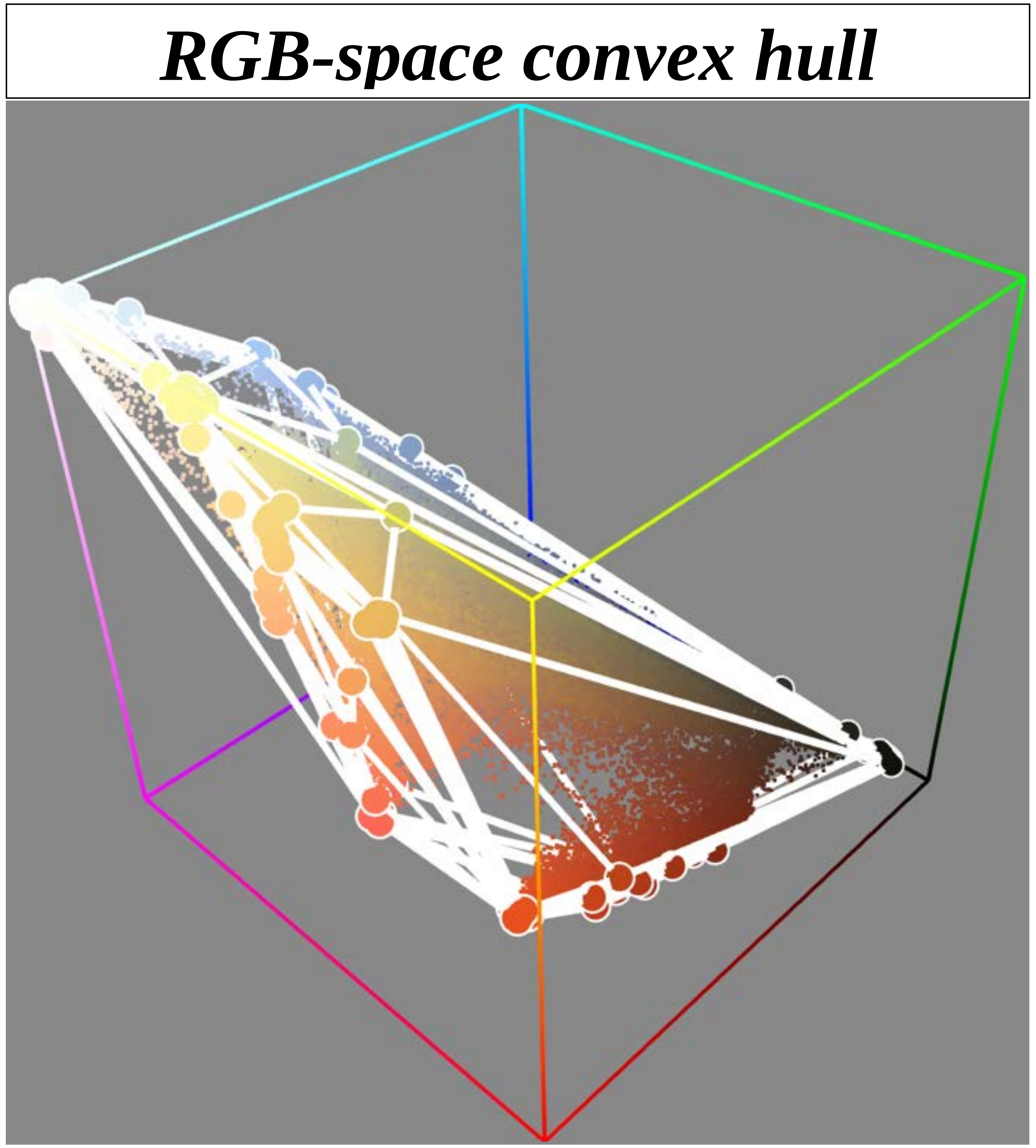




Convex hulls in RGB

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

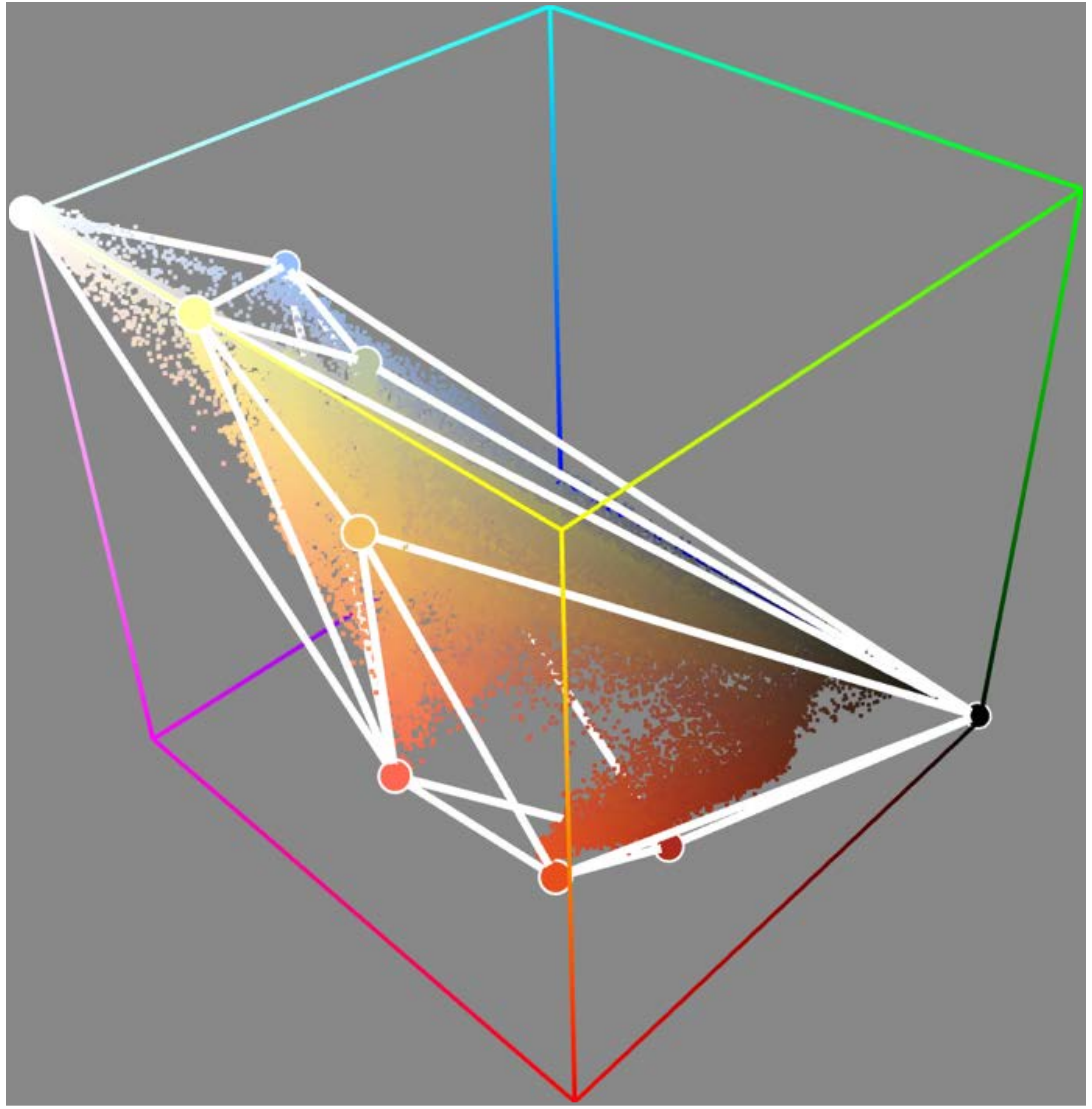




Palette Size

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

danidraws.com



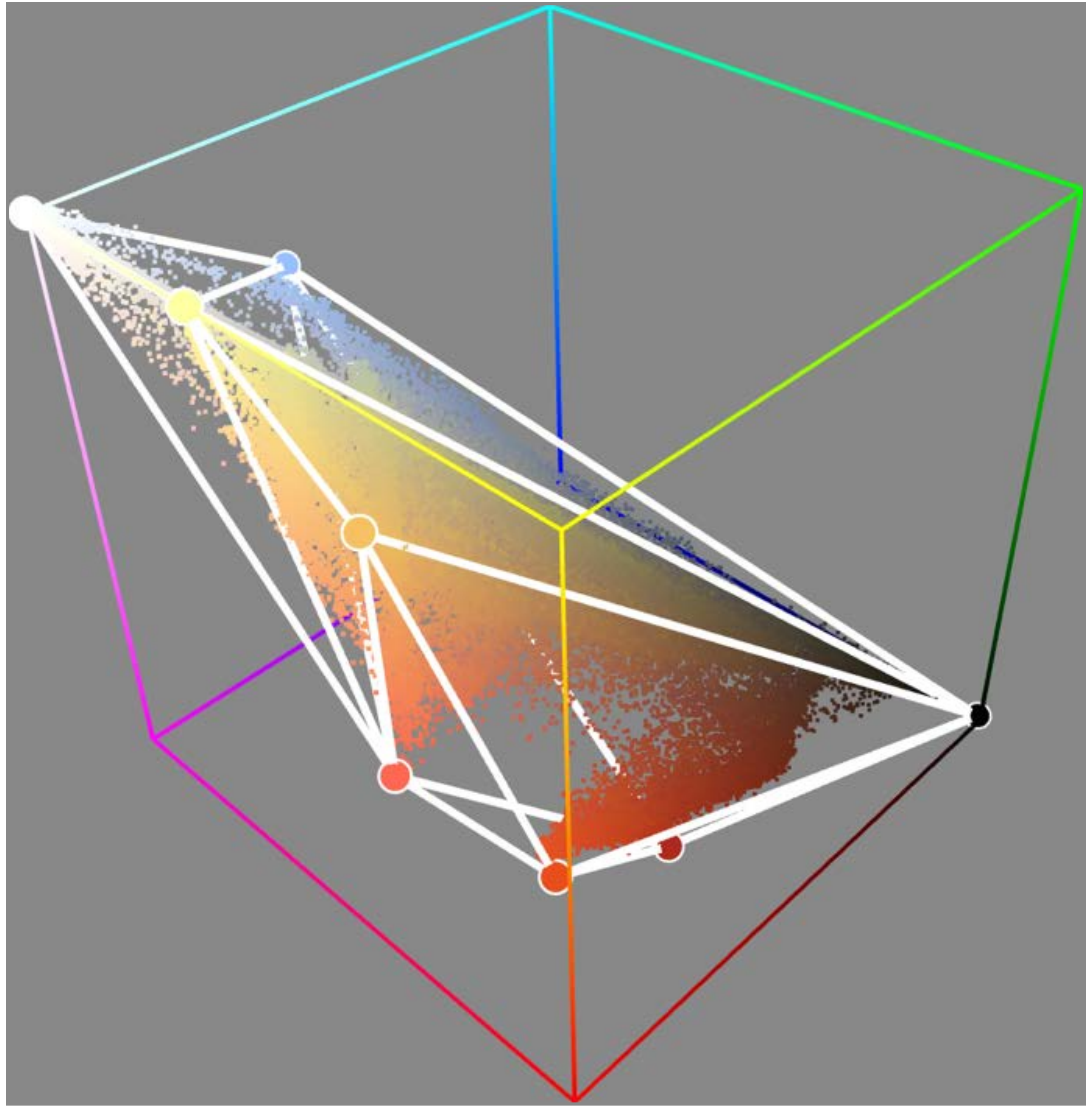
9 vertices

Palette Size

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

danidraws.com

8 vertices

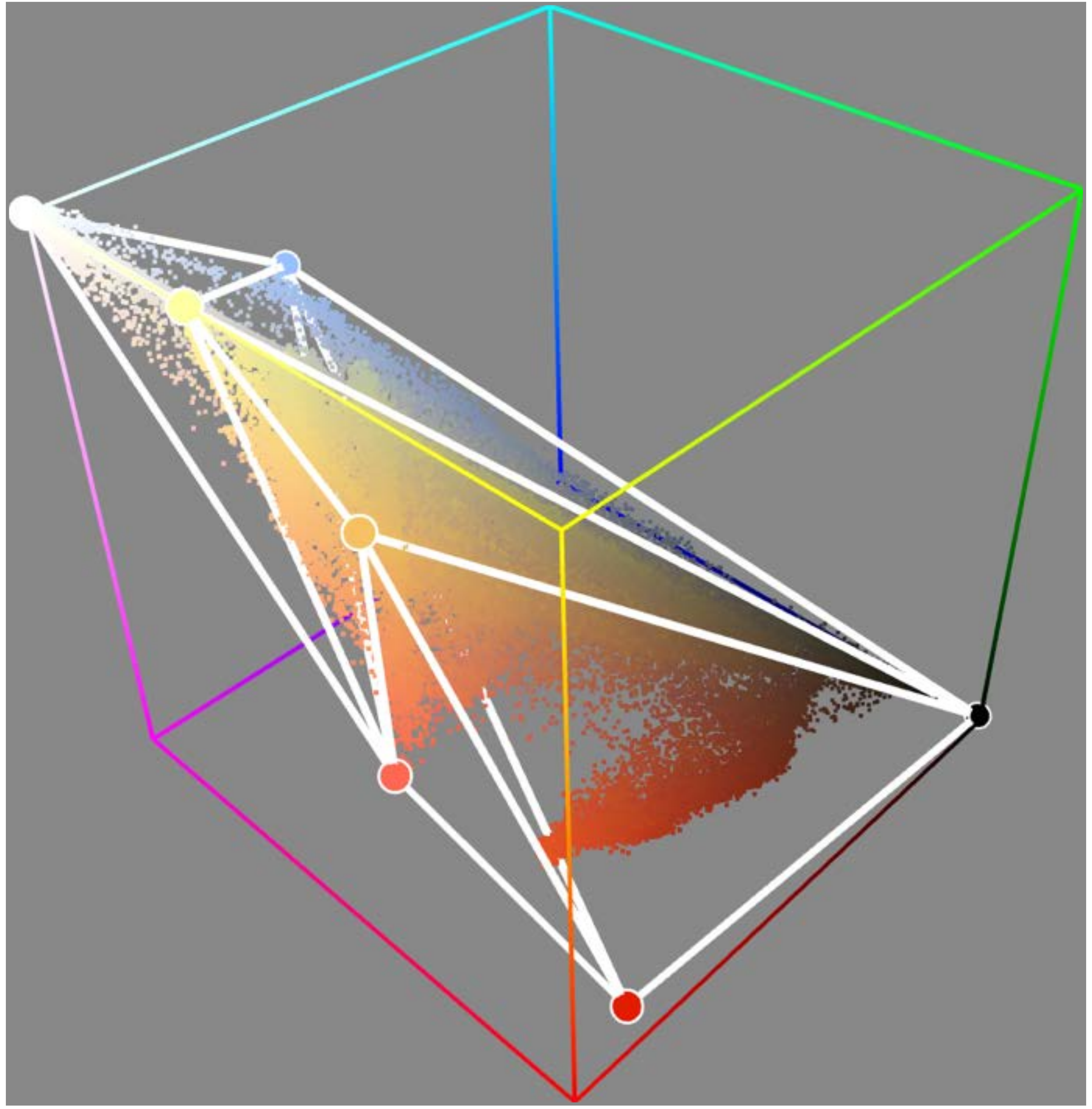


Palette Size

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

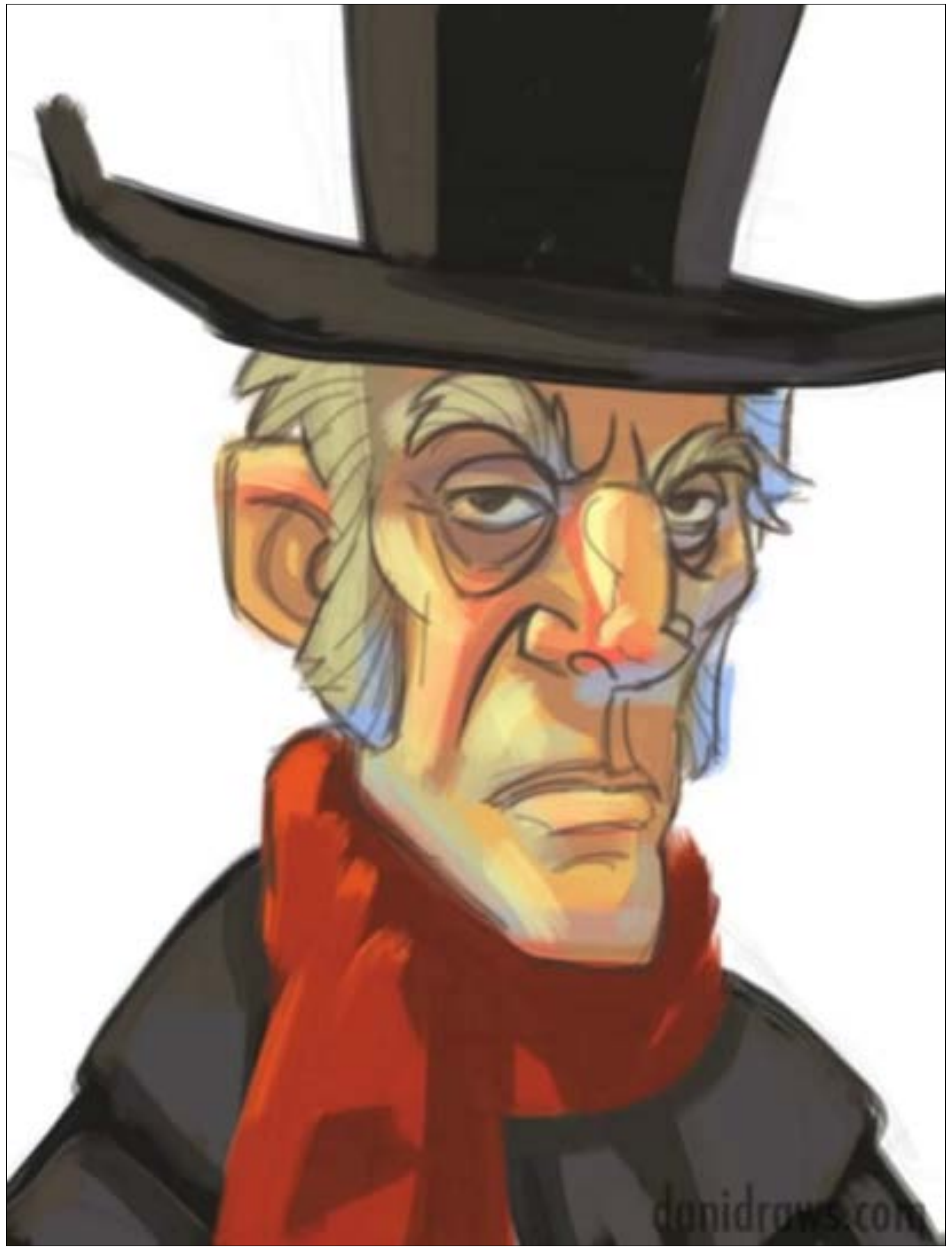
danidraws.com

7 vertices

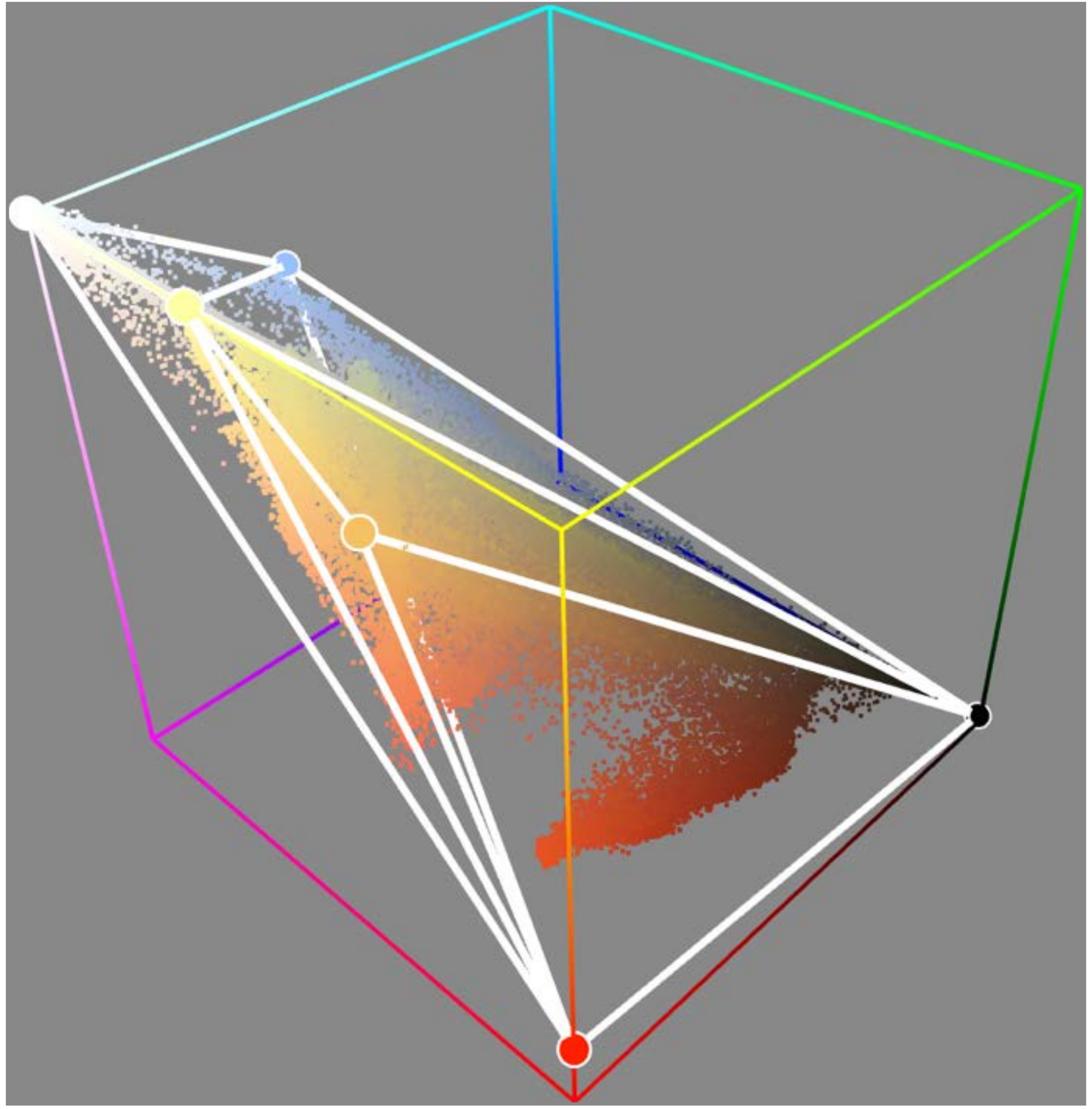


Palette Size

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

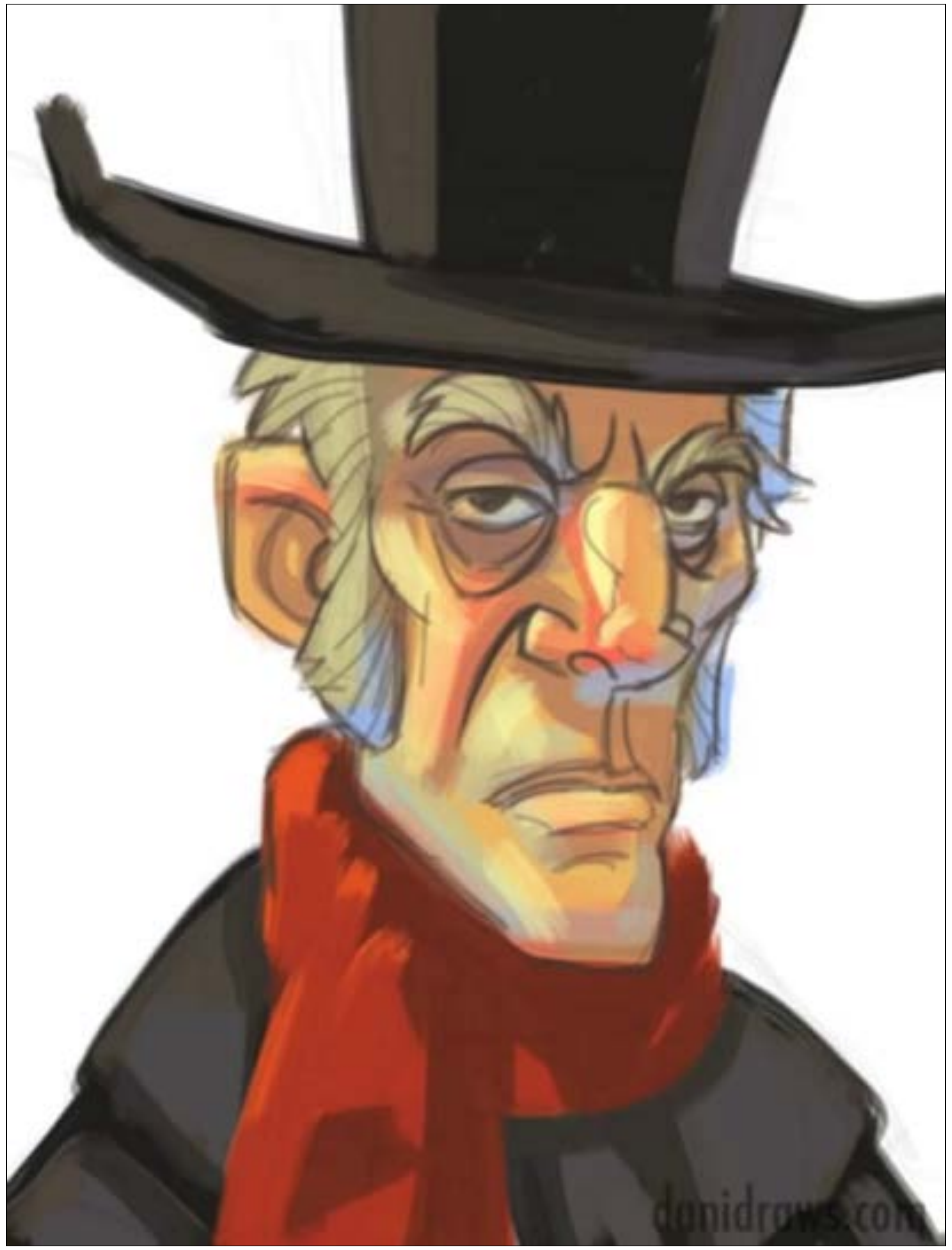


6 vertices

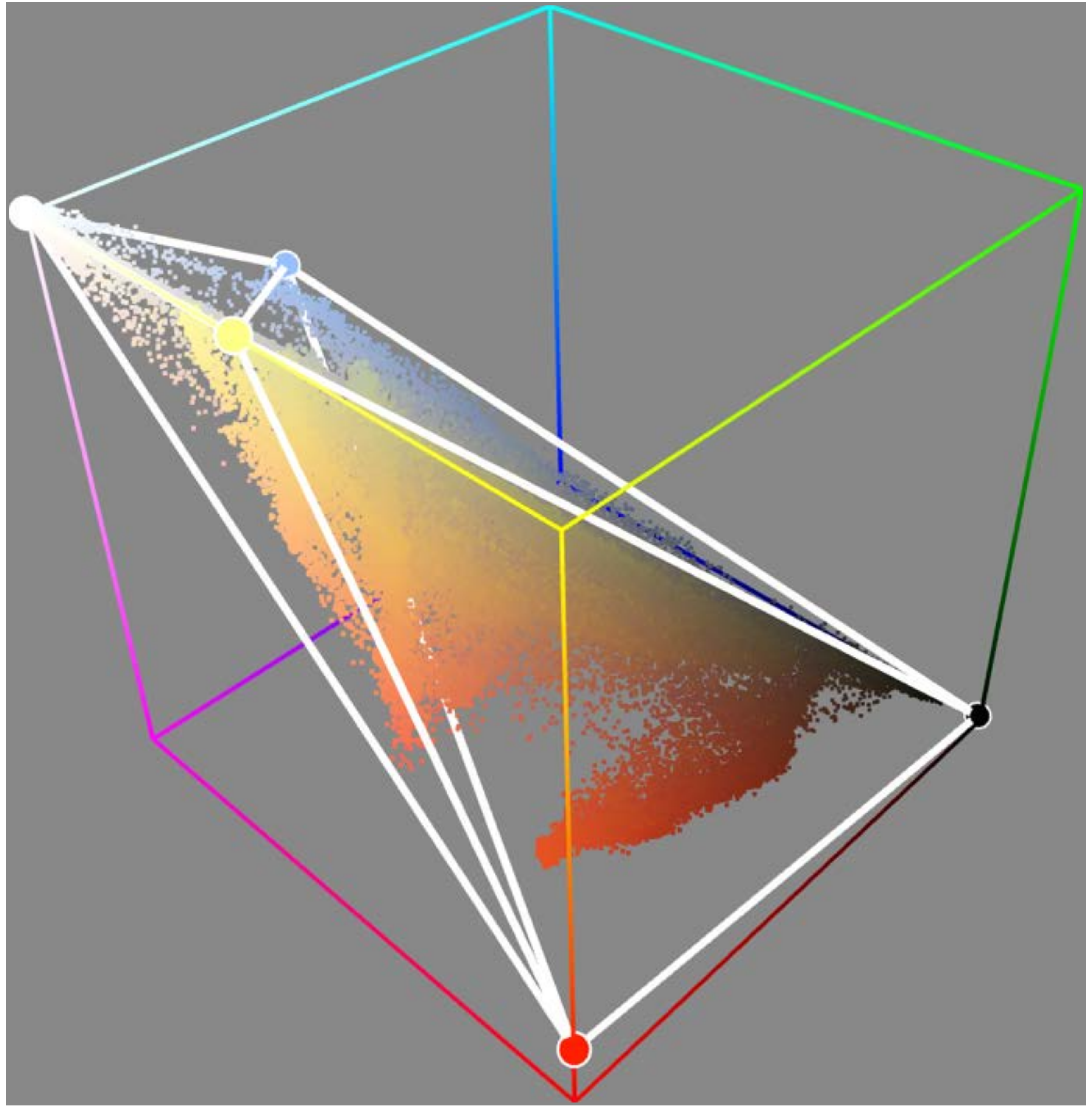


Palette Size

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

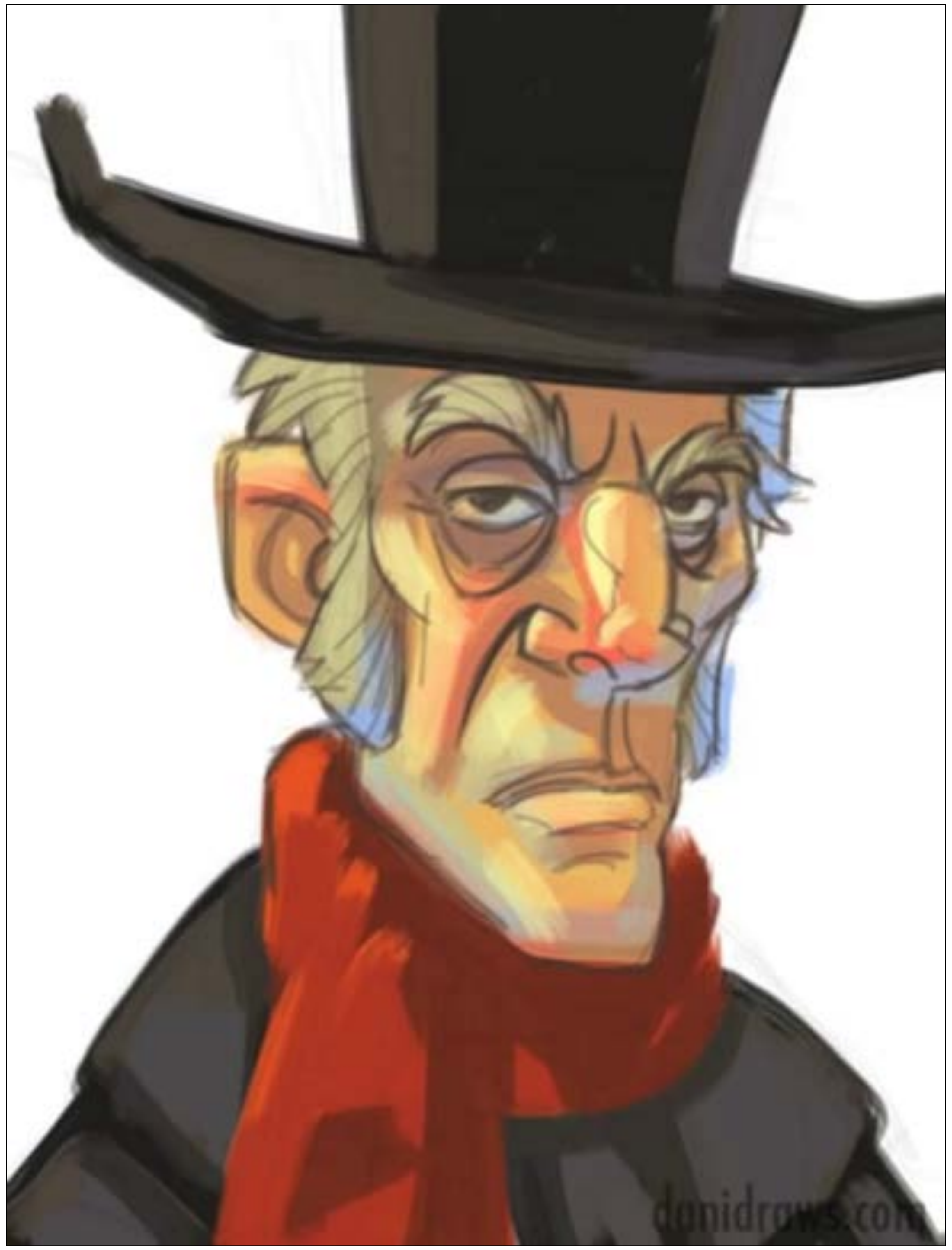


5 vertices

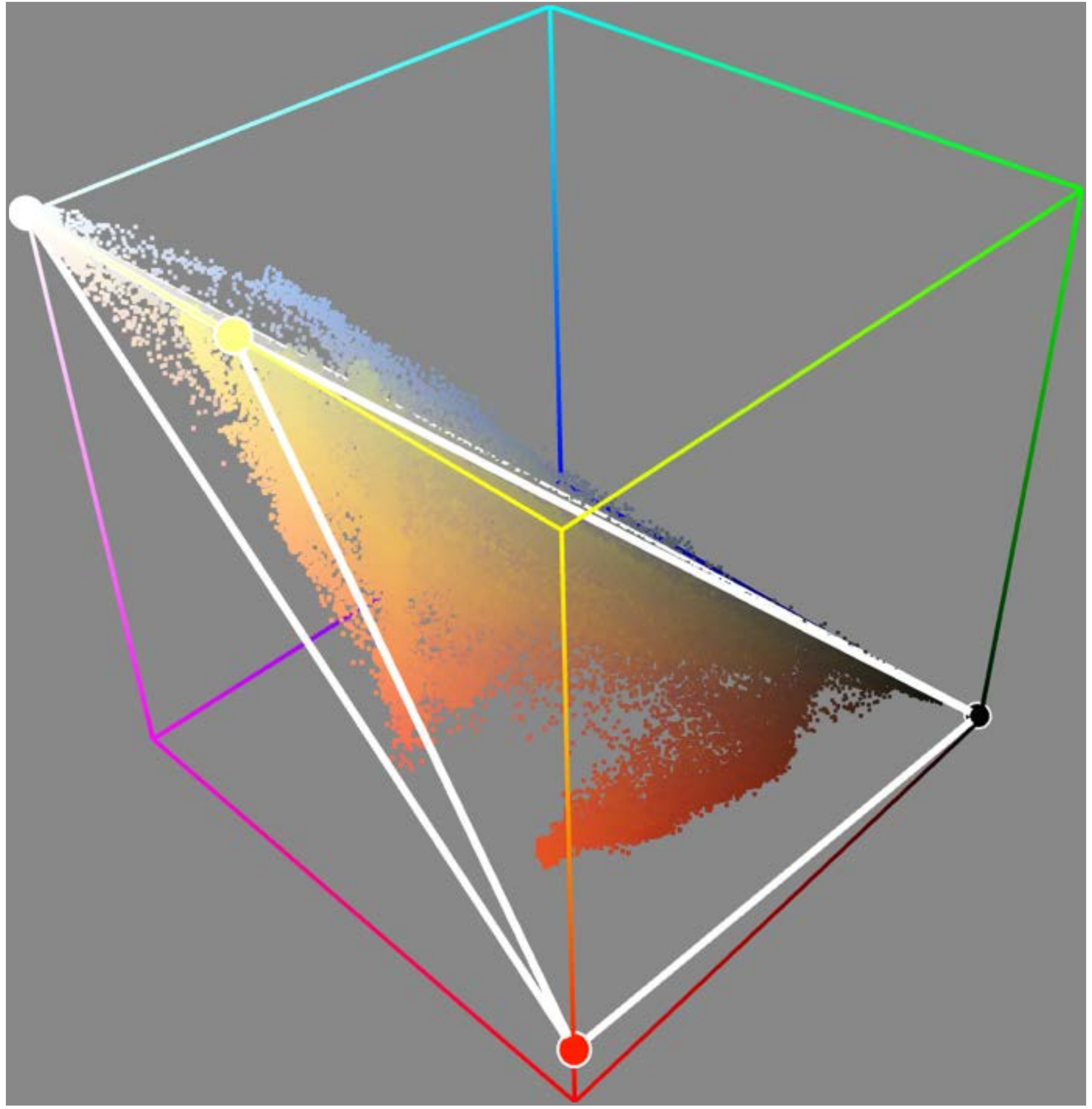


Palette Size

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



4 vertices

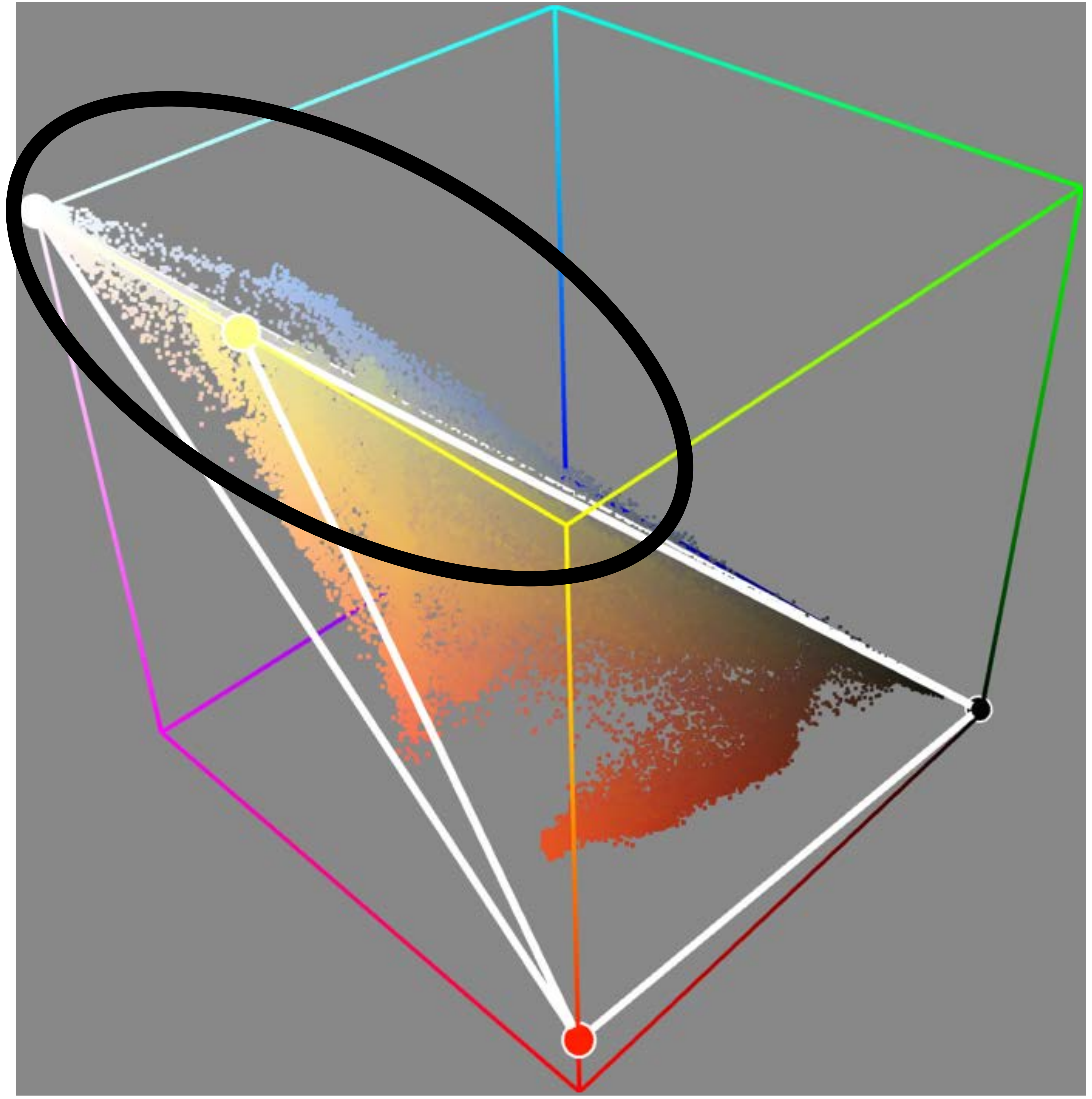


Palette Size

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

danidraws.com

4 vertices



Palette Size

- Our automatic error-bound simplification

danidraws.com

5 vertices

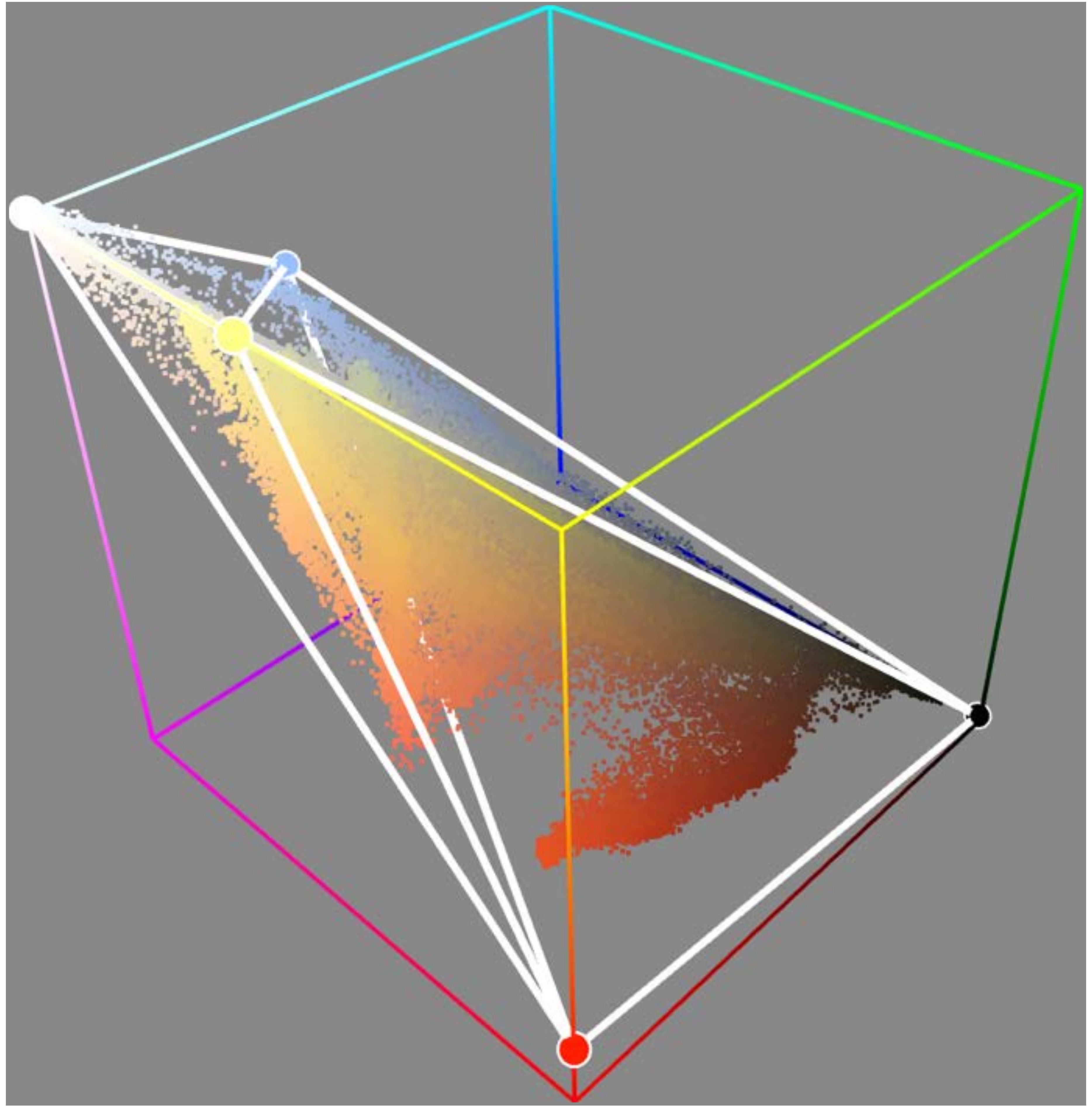
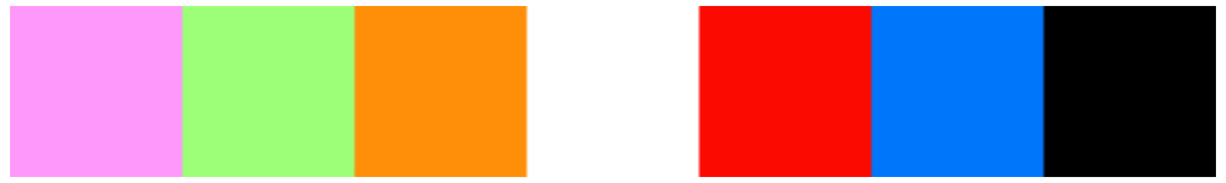


Image Decomposition

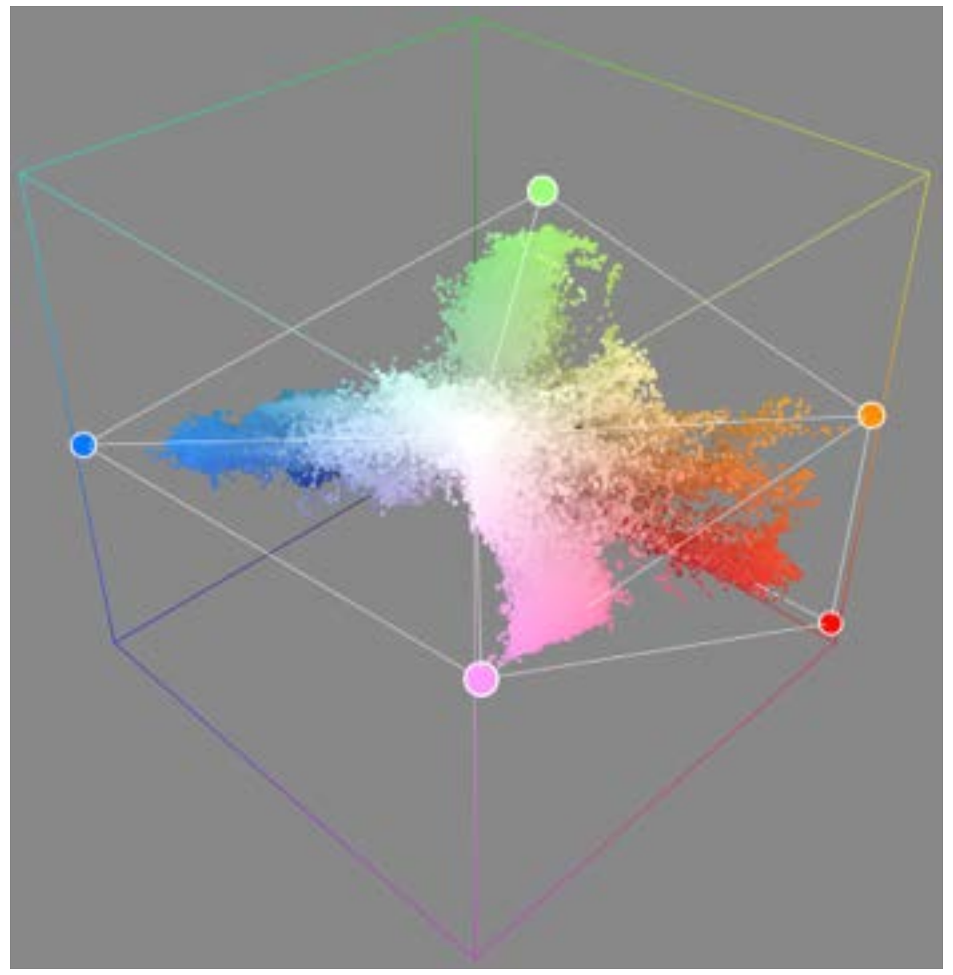
Extracting mixing weights

image



palette

RGB-space



Optimization

E = ||original – reconstructed image||^2 $\sum ||P_i - w_{ij}C_j||^2$

+

Per pixel mixing weights sparsity $\sum -(1-w_{ij})^2$

+

Mixing weights spatial smoothness (**Laplacian**)

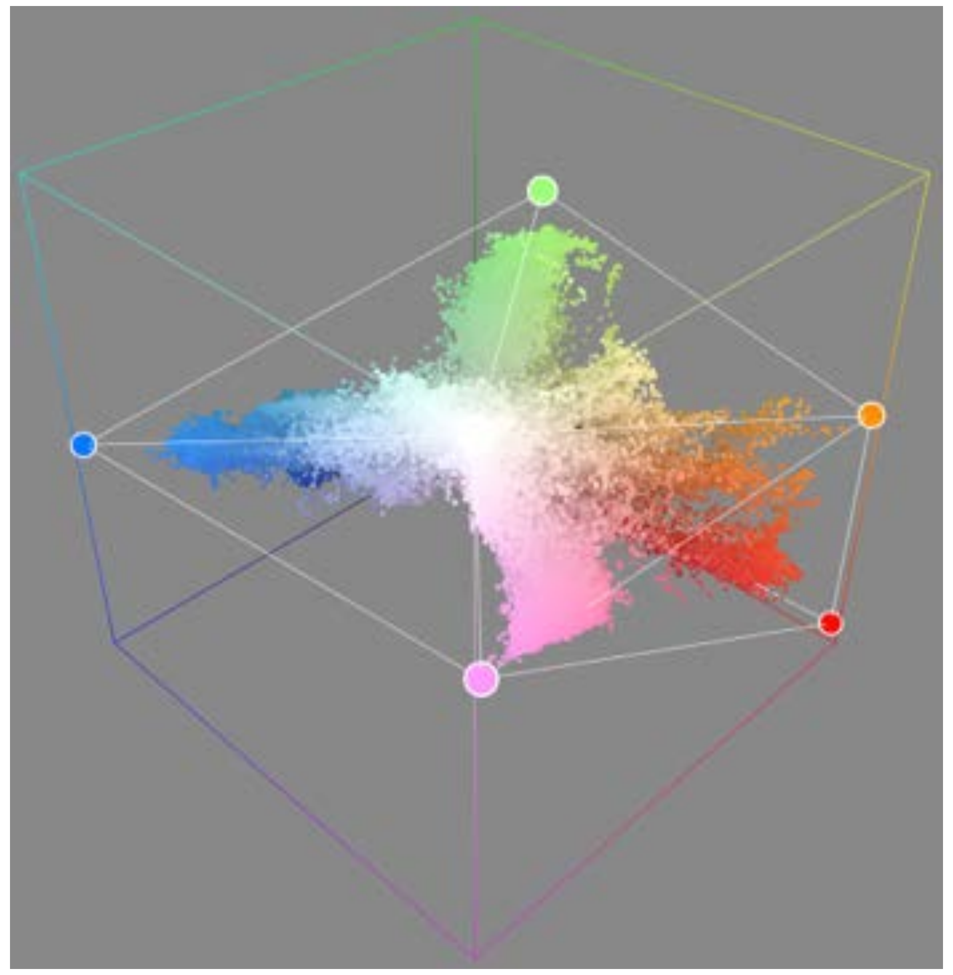
Extracting mixing weights

image

🟪🟩🟧⬜🟥🟦⬛

palette

RGB-space



Optimization

- Slow for high resolutions

E =

||original − reconstructed image||² $\sum ||P_i - w_{ij}C_j||^2$

+

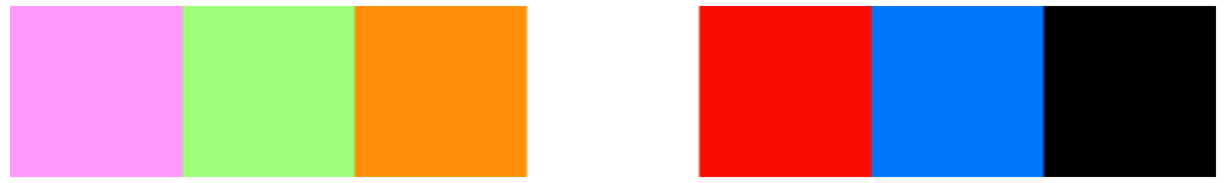
Per pixel mixing weights sparsity $\sum -(1 - w_{ij})^2$

+

Mixing weights spatial smoothness (**Laplacian**)

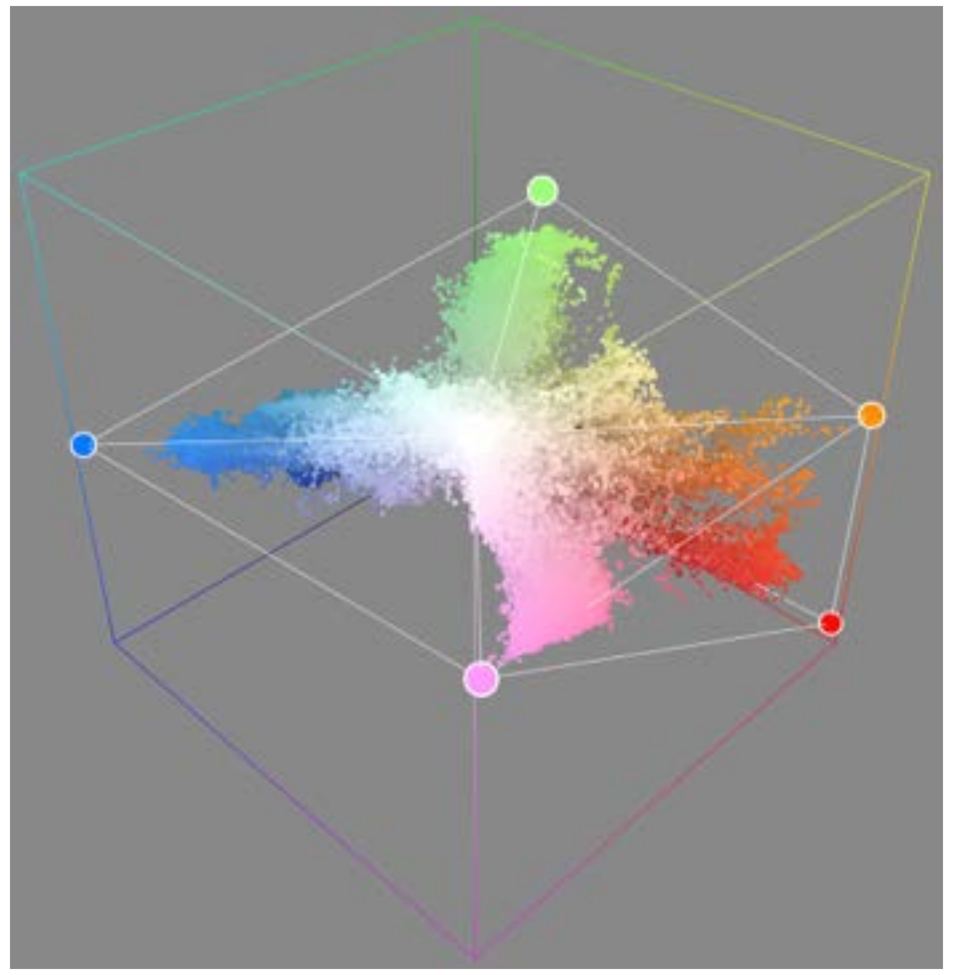
Extracting mixing weights

image



palette

RGB-space



Optimization

- Slow for high resolutions
- Many parameters to tune

E =

||original – reconstructed image||² $\sum ||P_i - w_{ij}C_j||^2$

+

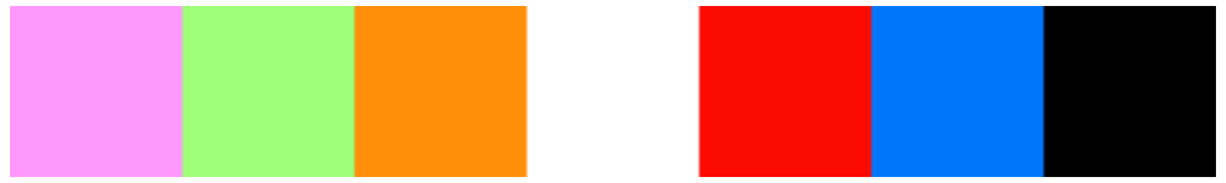
Per pixel mixing weights sparsity $\sum -(1 - w_{ij})^2$

+

Mixing weights spatial smoothness (**Laplacian**)

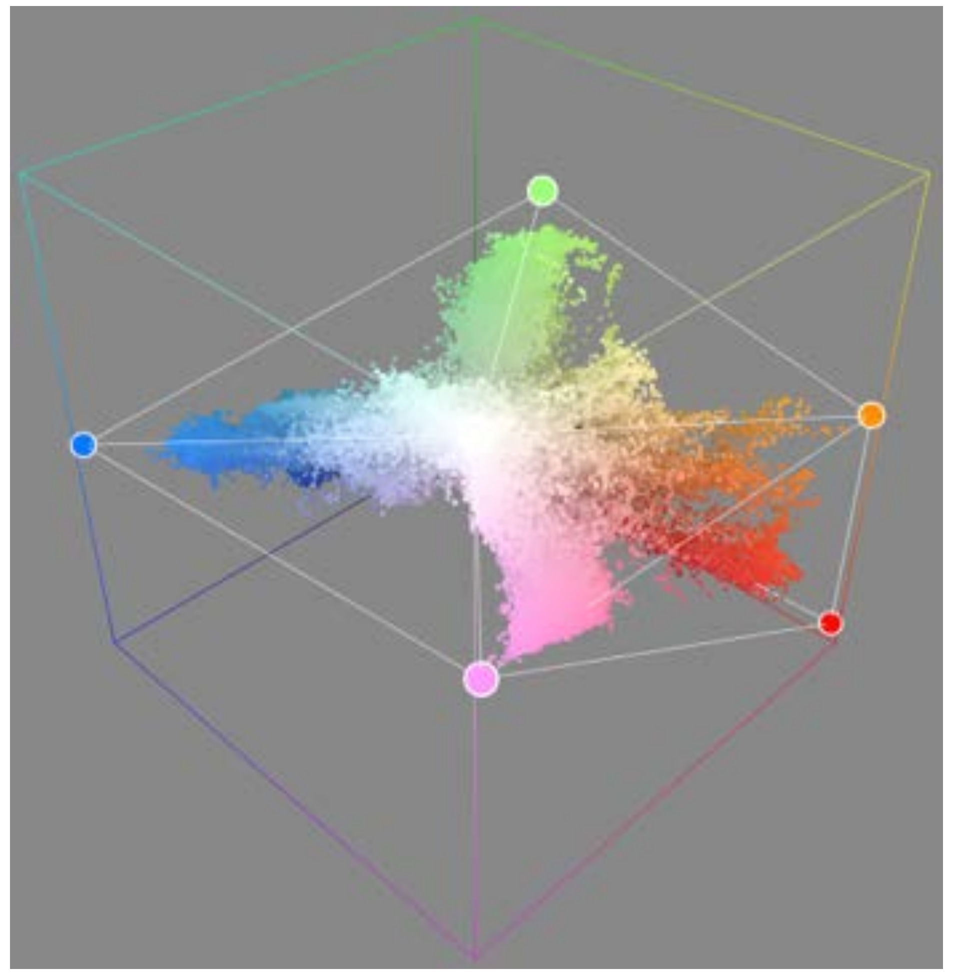
Extracting mixing weights

image



palette

RGB-space



Optimization

E =

||original – reconstructed image||^2 $\sum ||P_i - w_{ij}C_j||^2$

+

Per pixel mixing weights sparsity $\sum -(1 - w_{ij})^2$

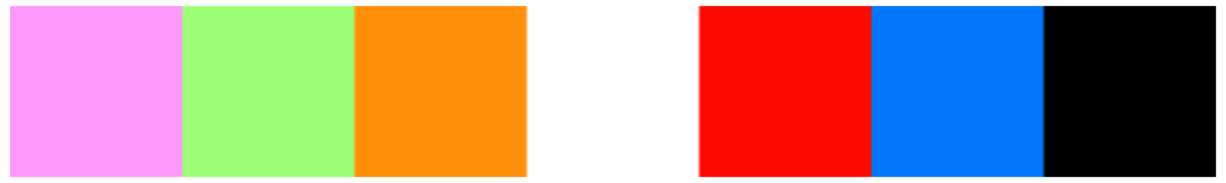
+

Mixing weights spatial smoothness (**Laplacian**)

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

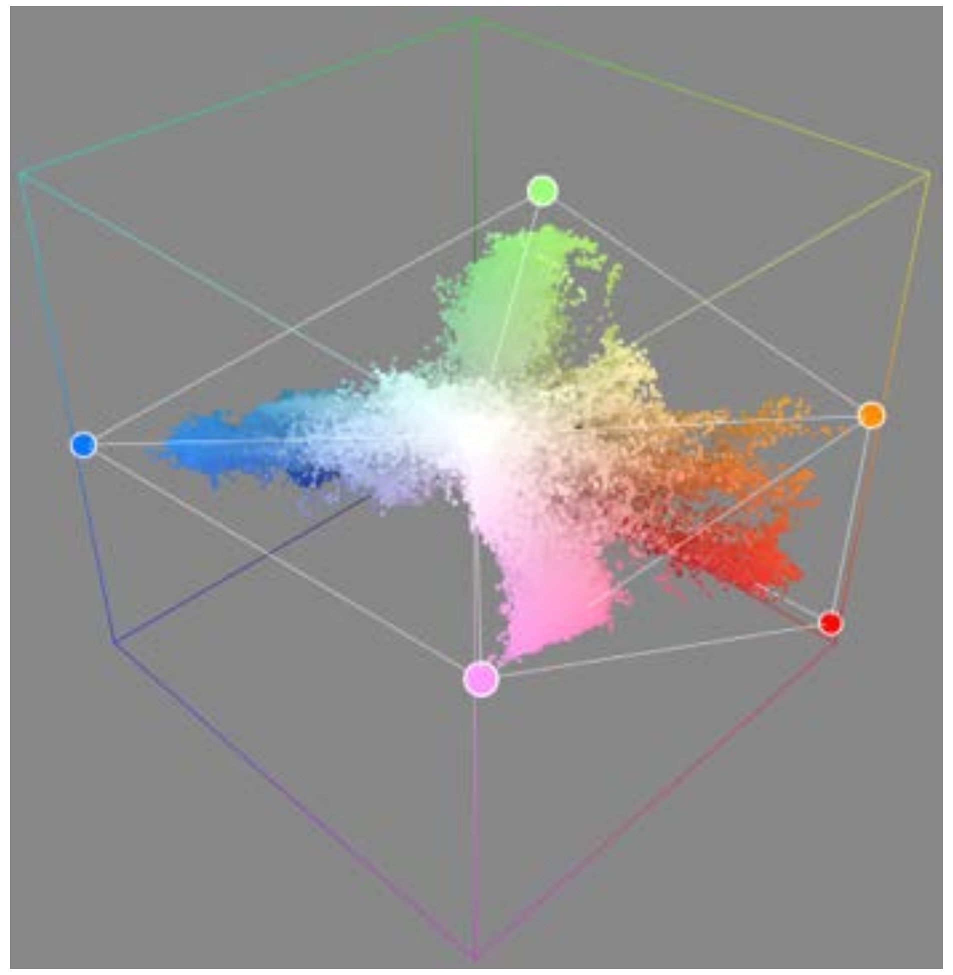
Extracting mixing weights

image



palette

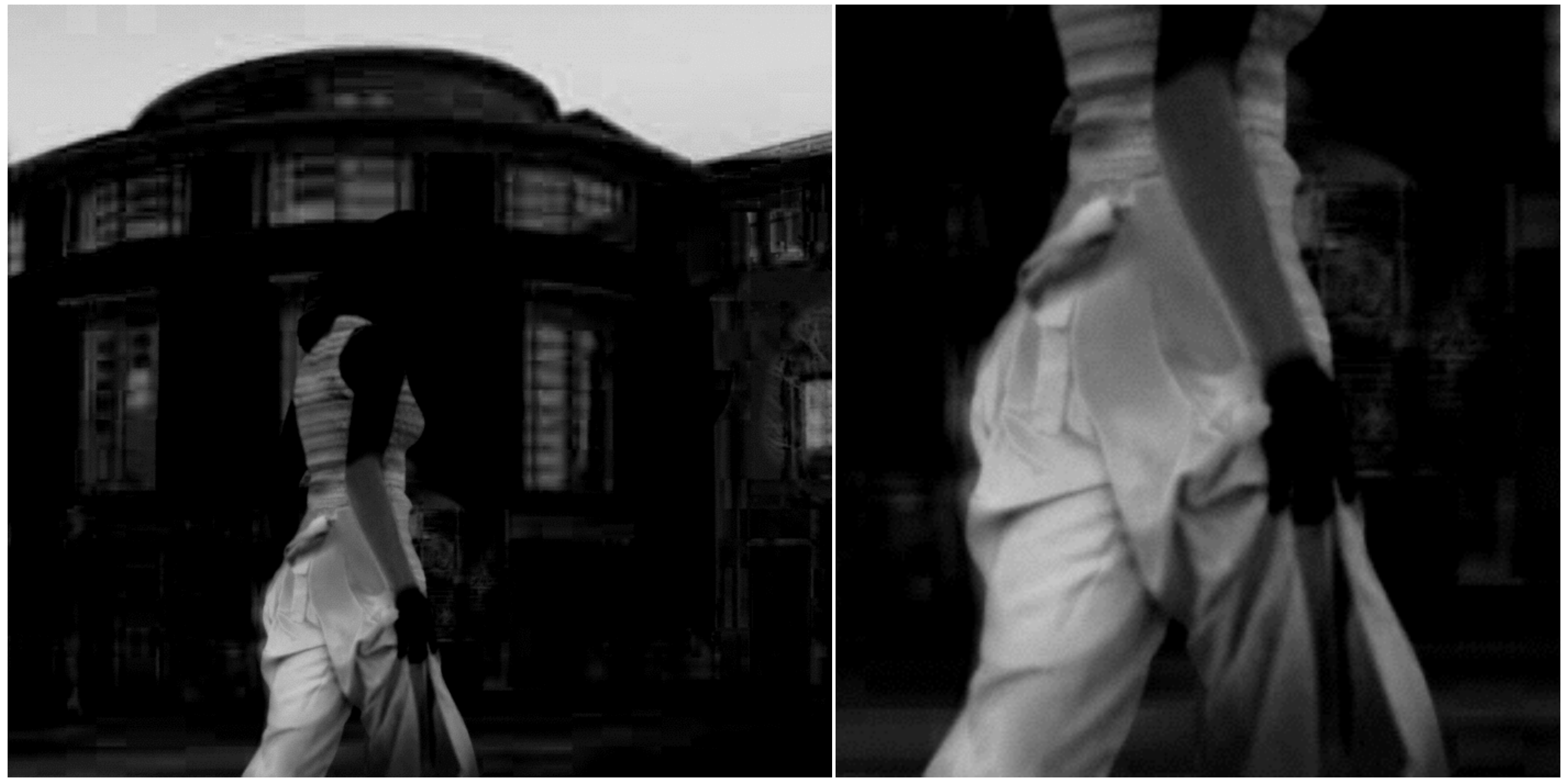
RGB-space



Optimization

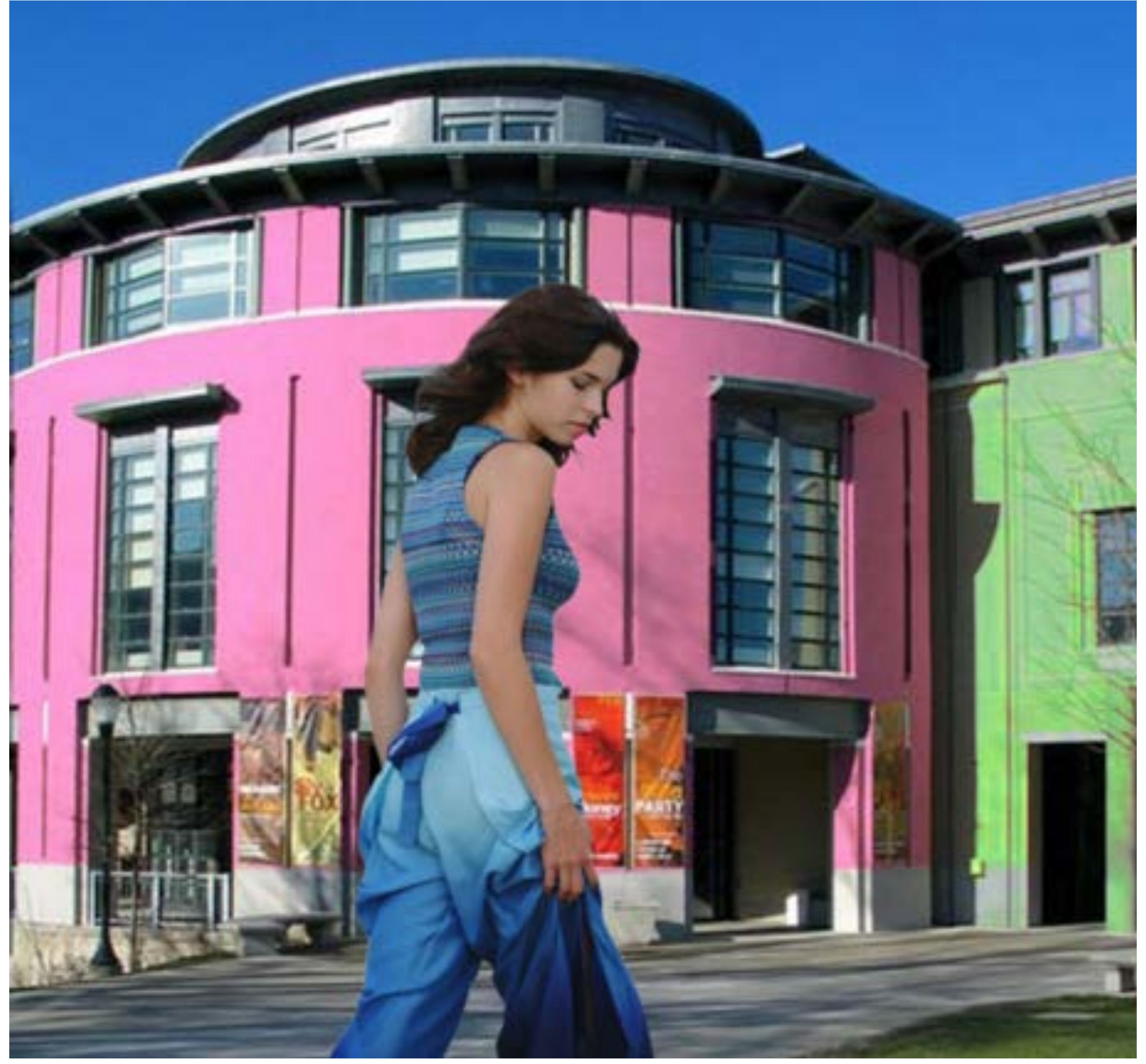
$$\mathbf{E} = \begin{array}{ll} \|\text{original} - \text{reconstructed image}\|^2 & \sum \|P_i - w_{ij} C_j\|^2 \\ + & \\ \text{Per pixel mixing weights sparsity} & \sum -(1 - w_{ij})^2 \\ + & \\ \text{Mixing weights spatial smoothness} & (\textbf{Laplacian}) \end{array}$$

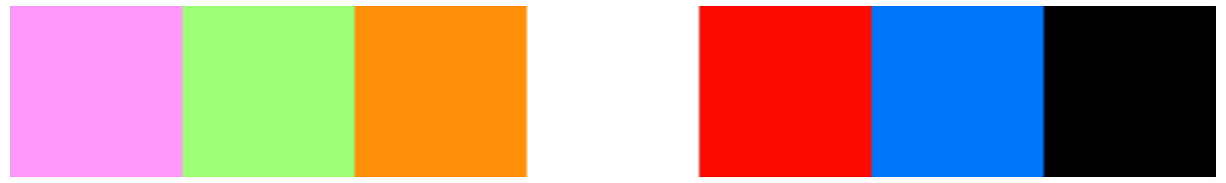
- Slow for high resolutions
- Many parameters to tune
- Per-image parameters



Extracting mixing weights

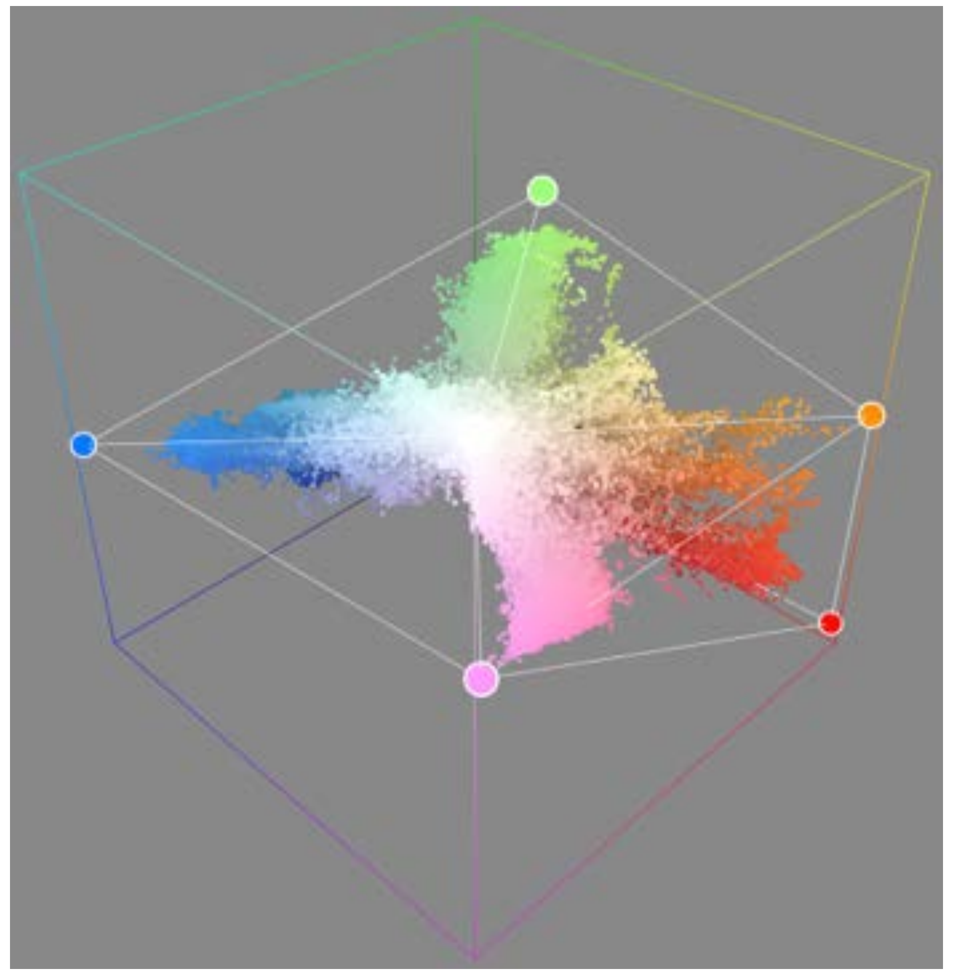
image





palette

RGB-space



Optimization

E =

||original − reconstructed image||^2 $\sum ||P_i - w_{ij}C_j||^2$

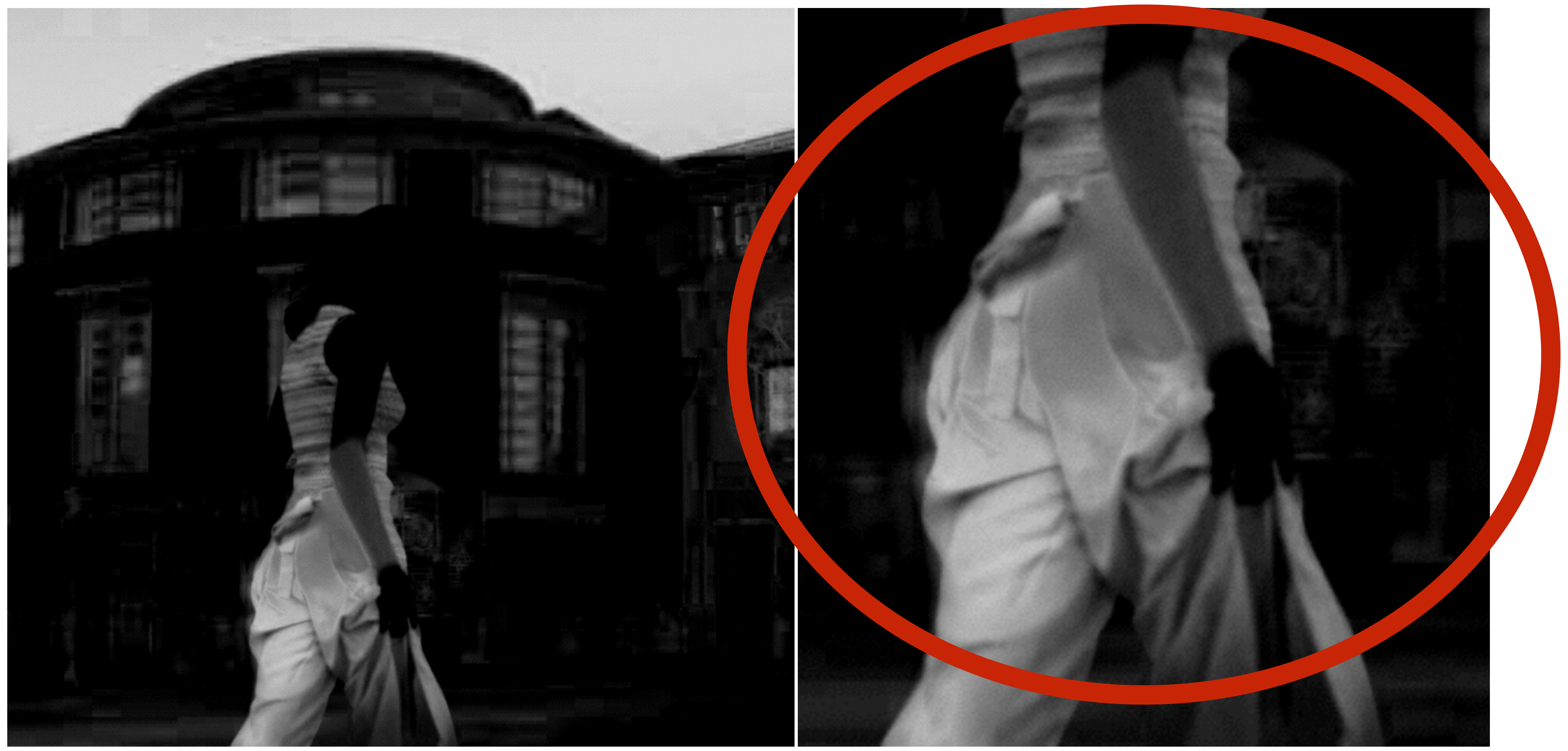
+

Per pixel mixing weights sparsity $\sum -(1 - w_{ij})^2$

+

Mixing weights spatial smoothness (**Laplacian**)

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



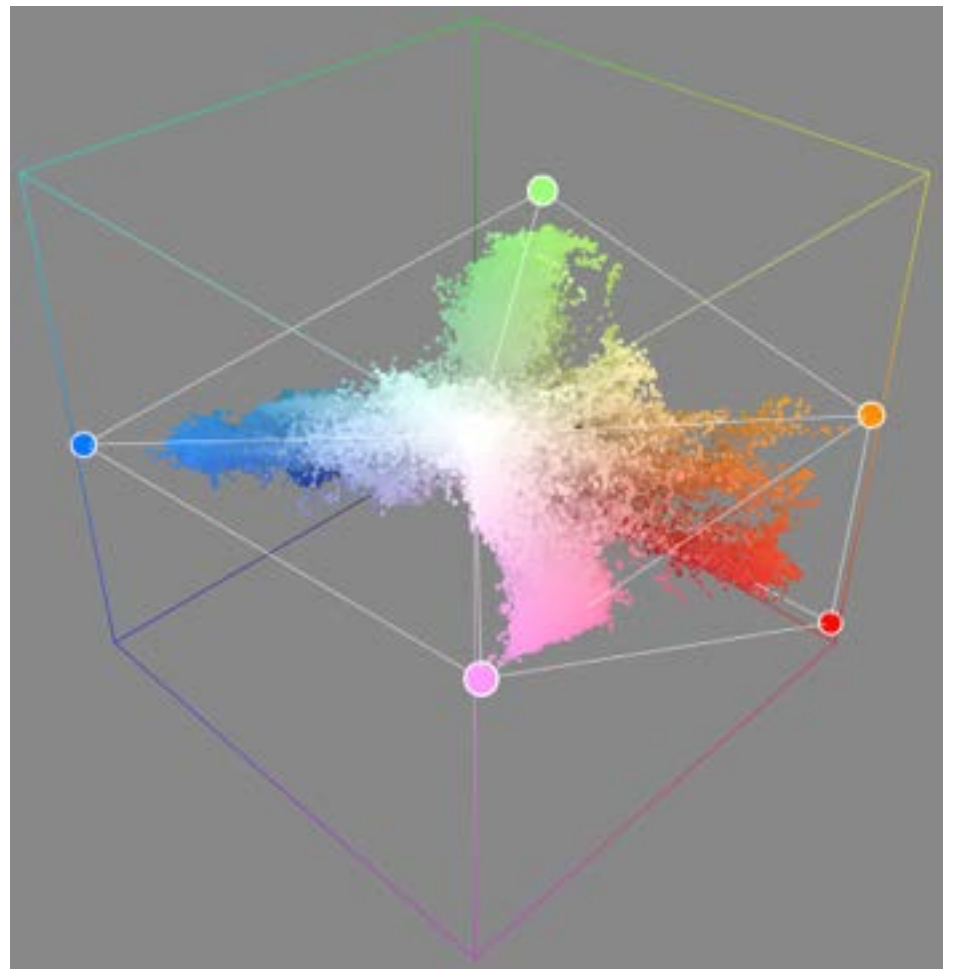
Extracting mixing weights

image

🟪🟩🟧⬜🟥🟦⬛

palette

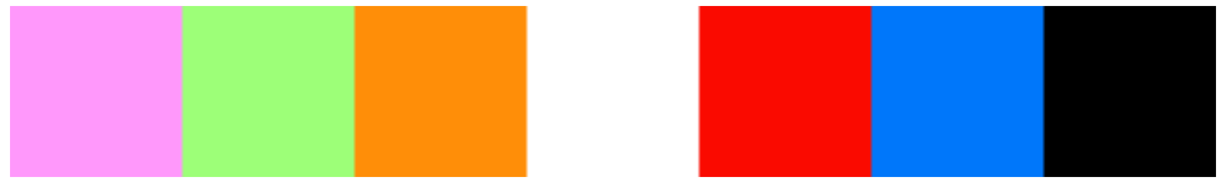
RGB-space



~~**Optimization**~~

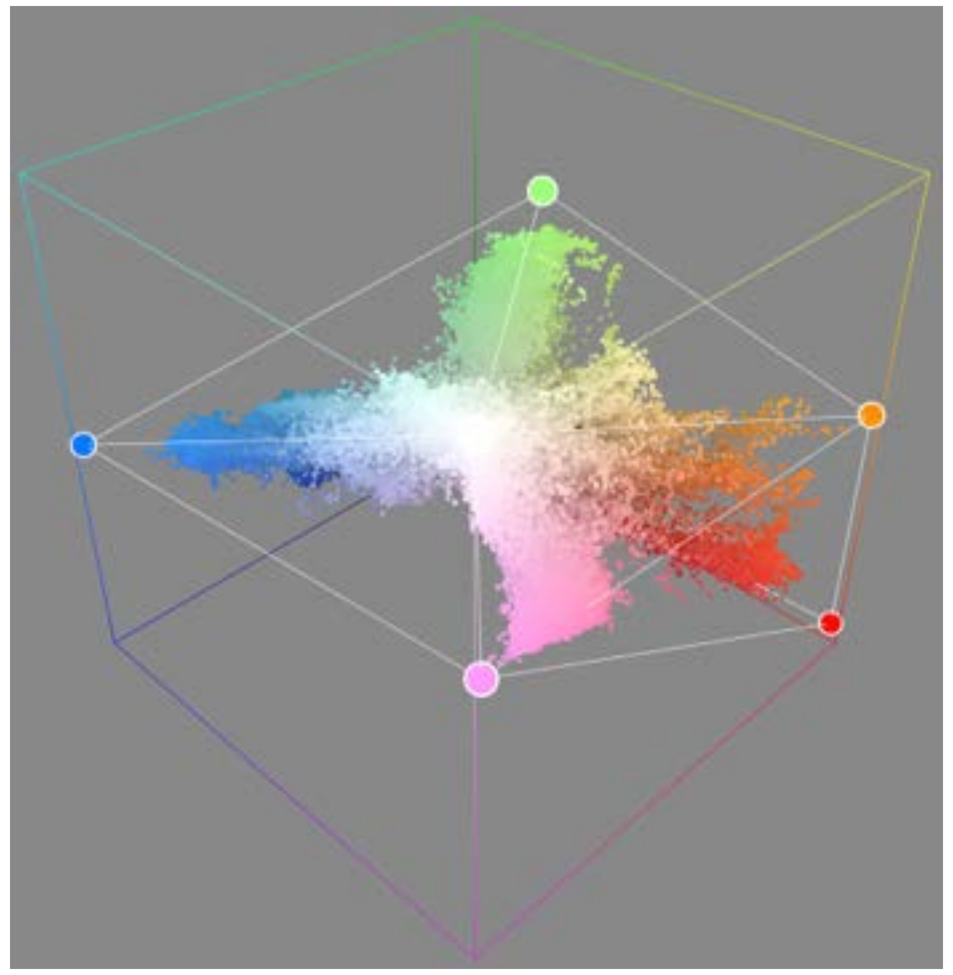
Extracting mixing weights

image



palette

RGB-space

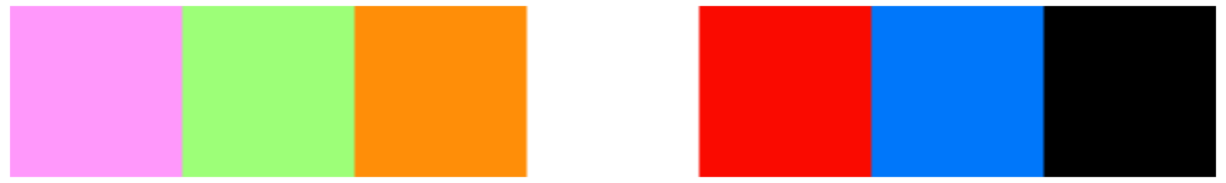


~~**Optimization**~~

Generalized Barycentric Coordinates

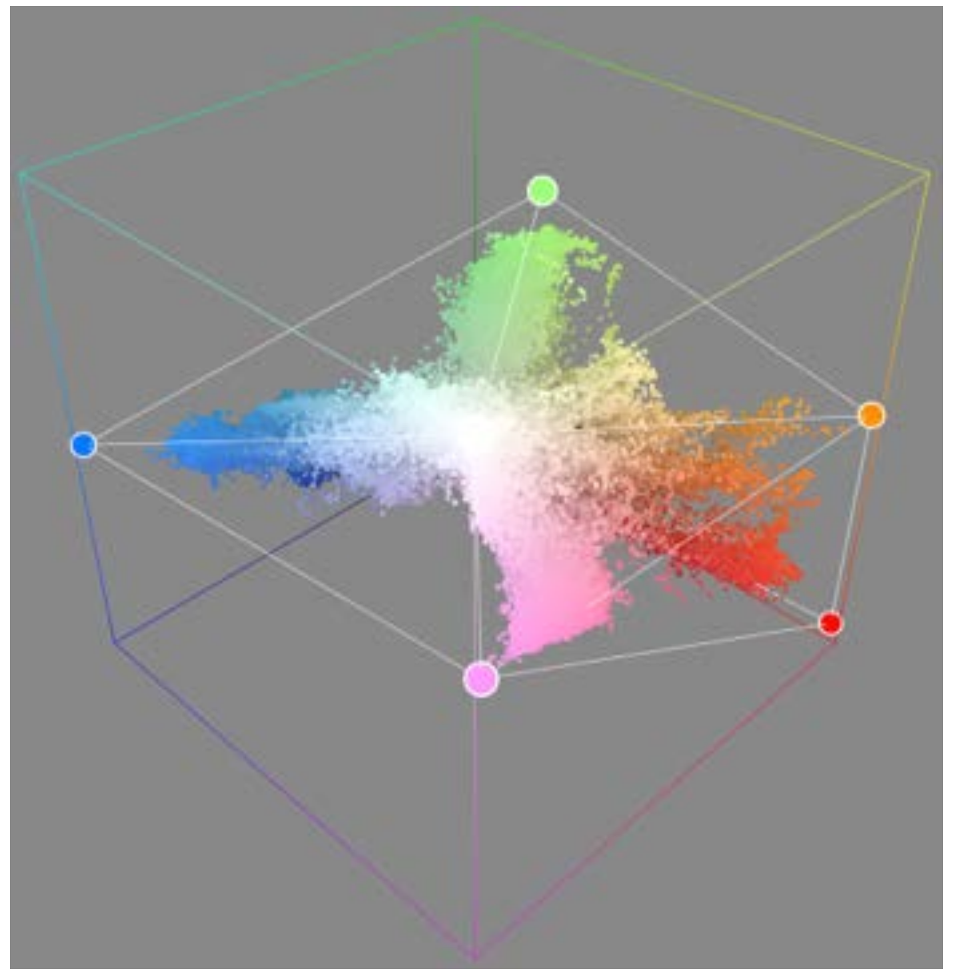
Extracting mixing weights

image



palette

RGB-space



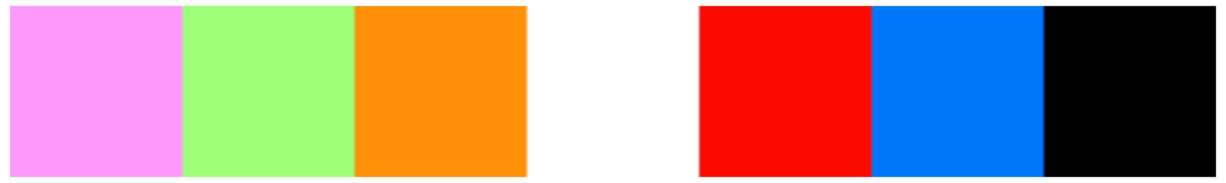
~~Optimization~~

Generalized Barycentric Coordinates

- Fast

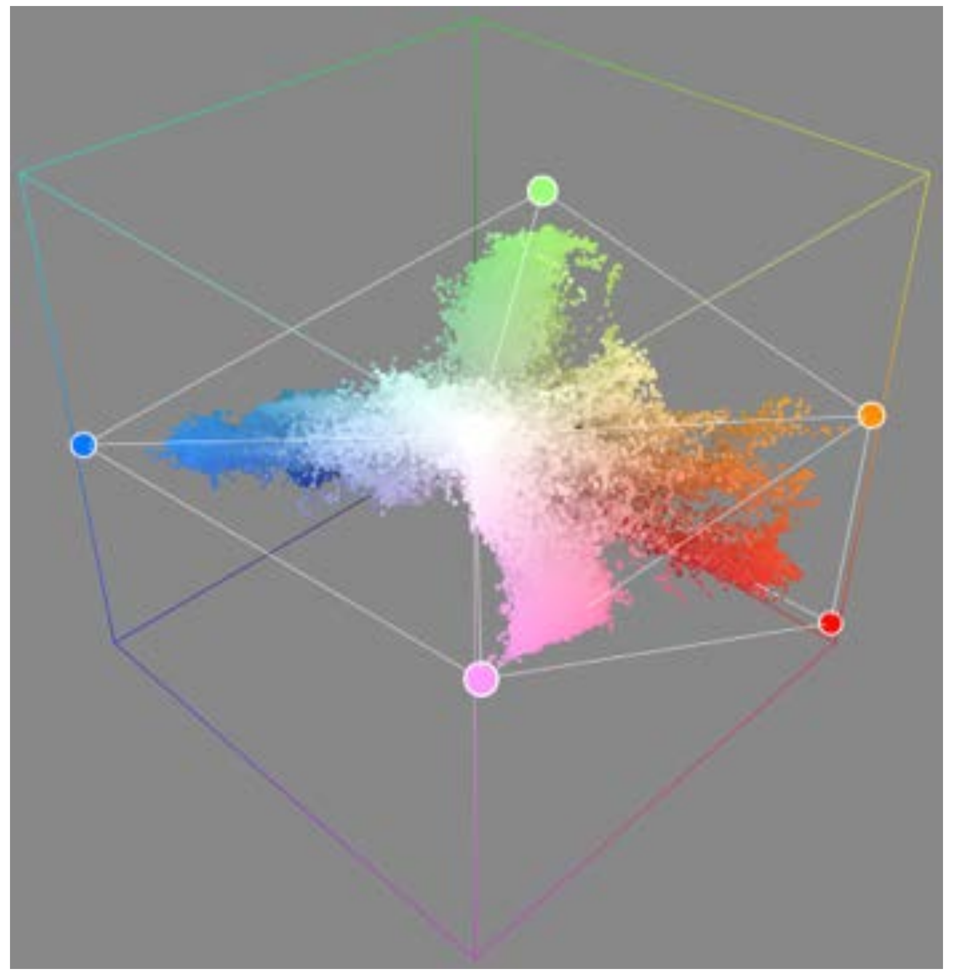
Extracting mixing weights

image



palette

RGB-space



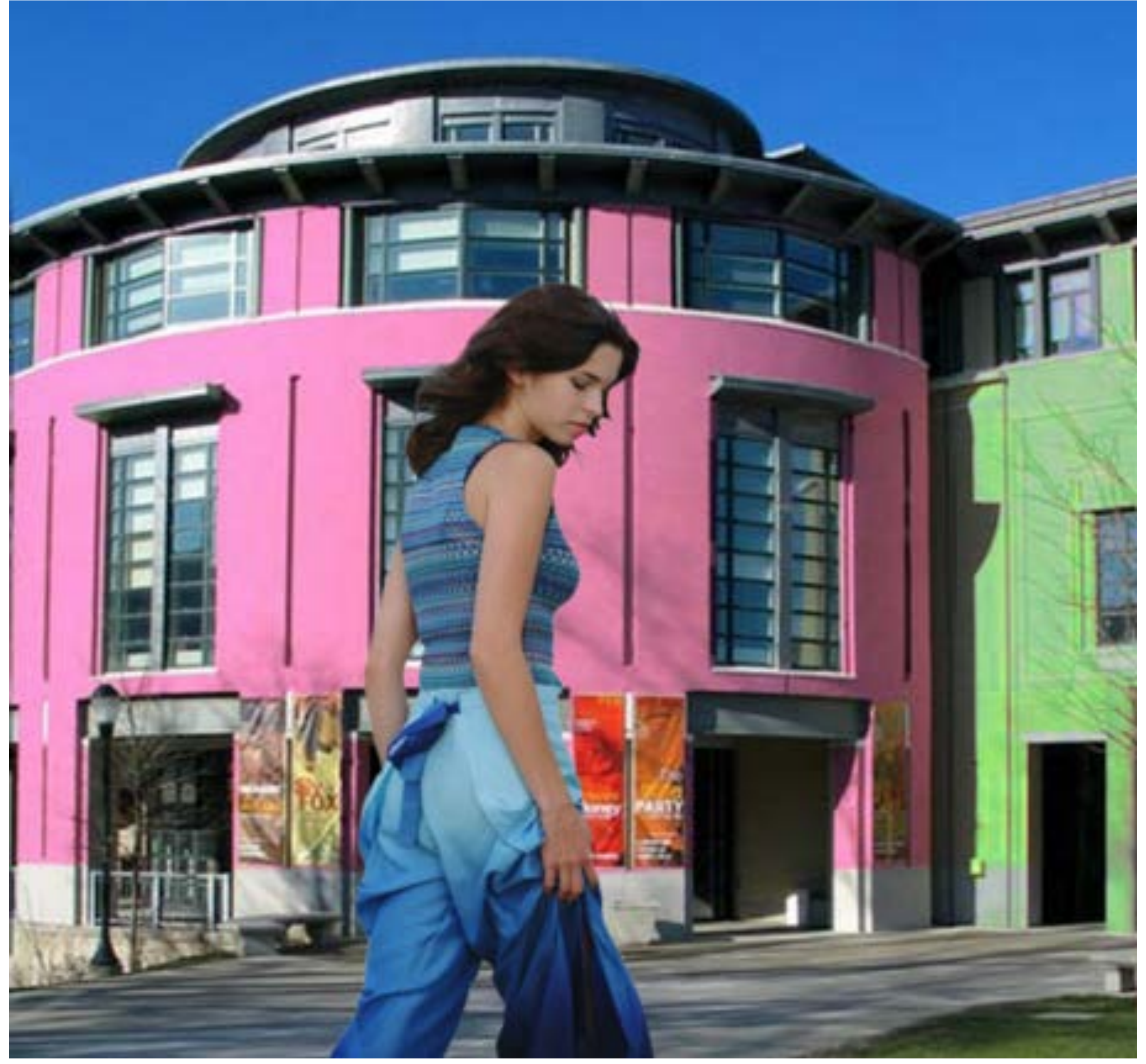
~~Optimization~~

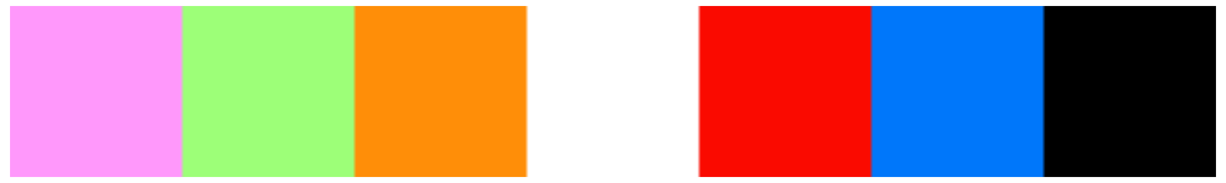
Generalized Barycentric Coordinates

- Fast
- No parameters to tune

Extracting mixing weights

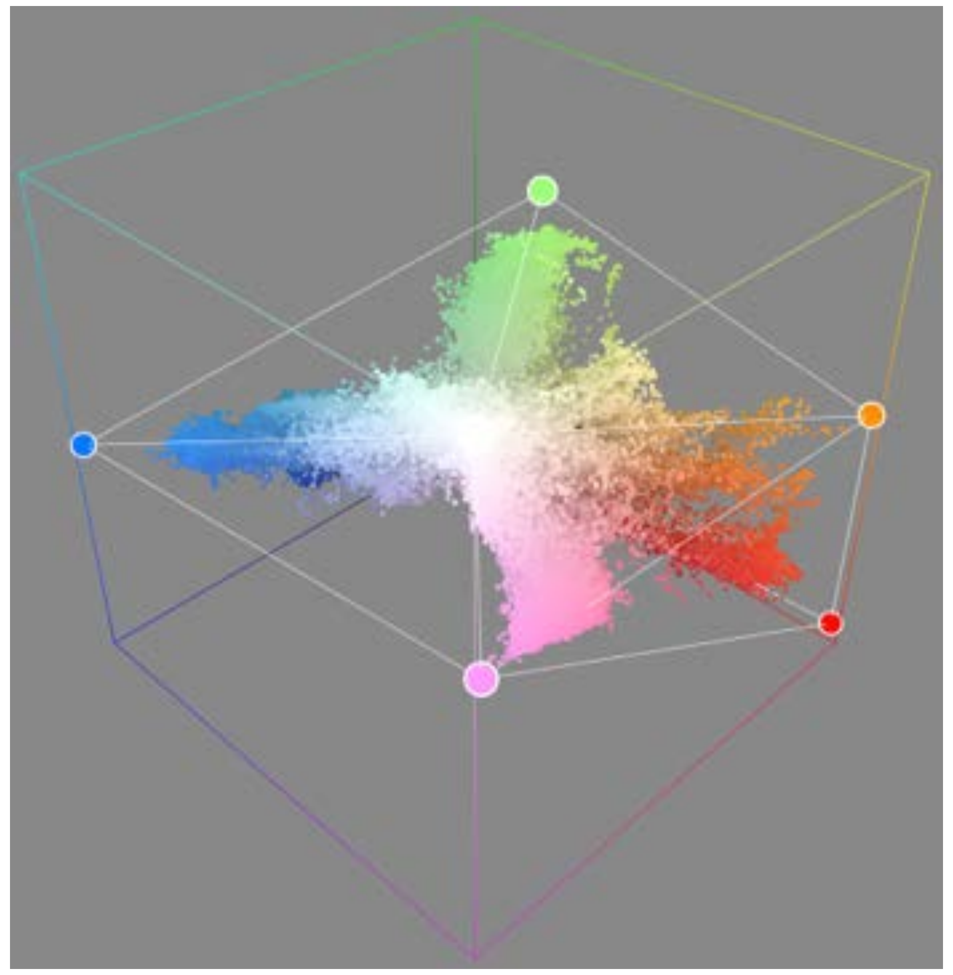
image





palette

RGB-space



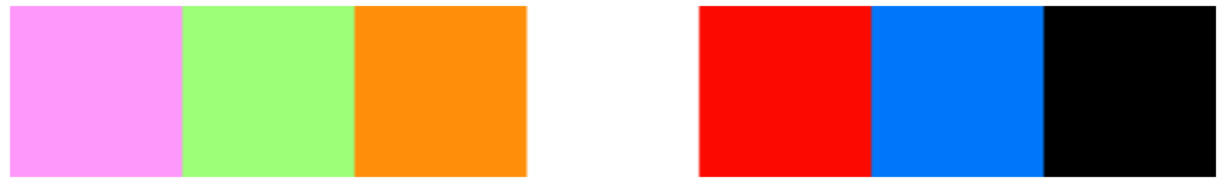
~~Optimization~~

Generalized Barycentric Coordinates

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

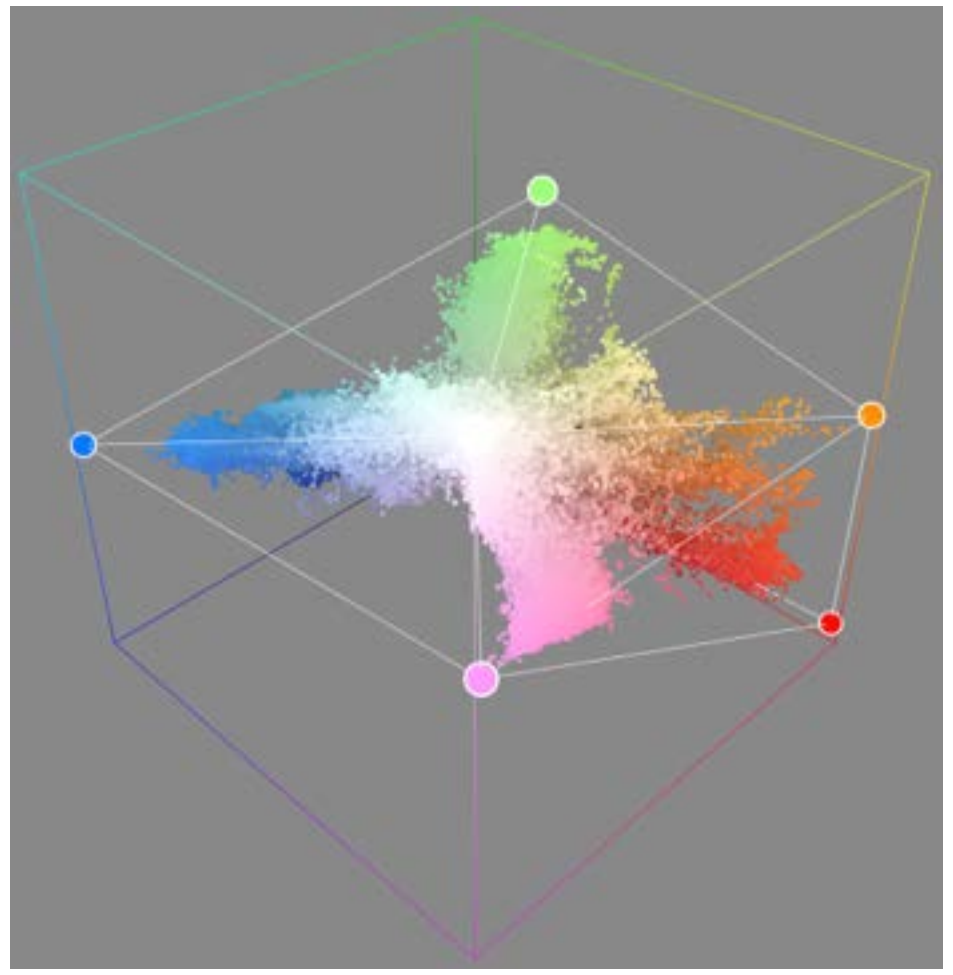
Extracting mixing weights

image



palette

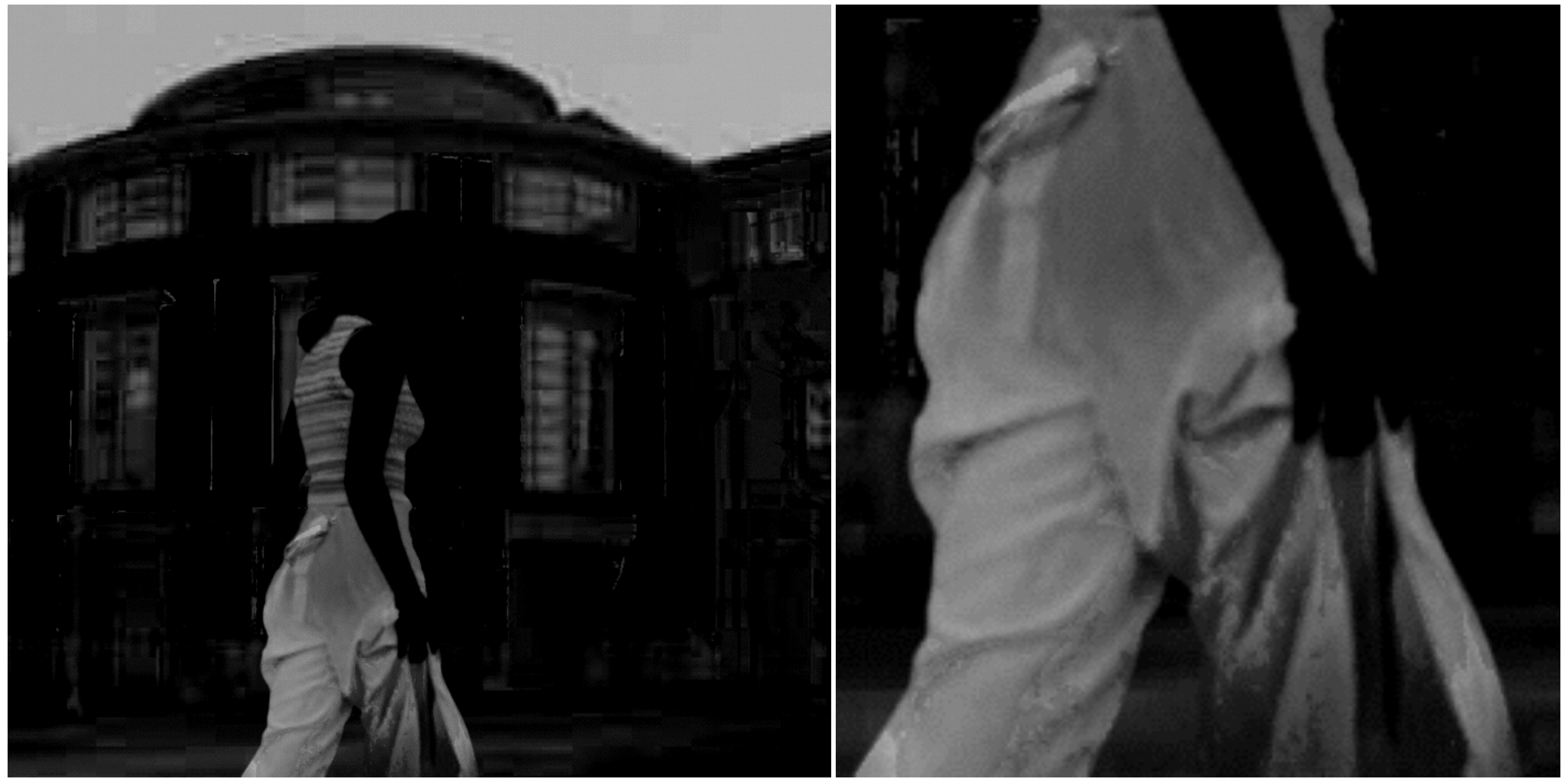
RGB-space



~~Optimization~~

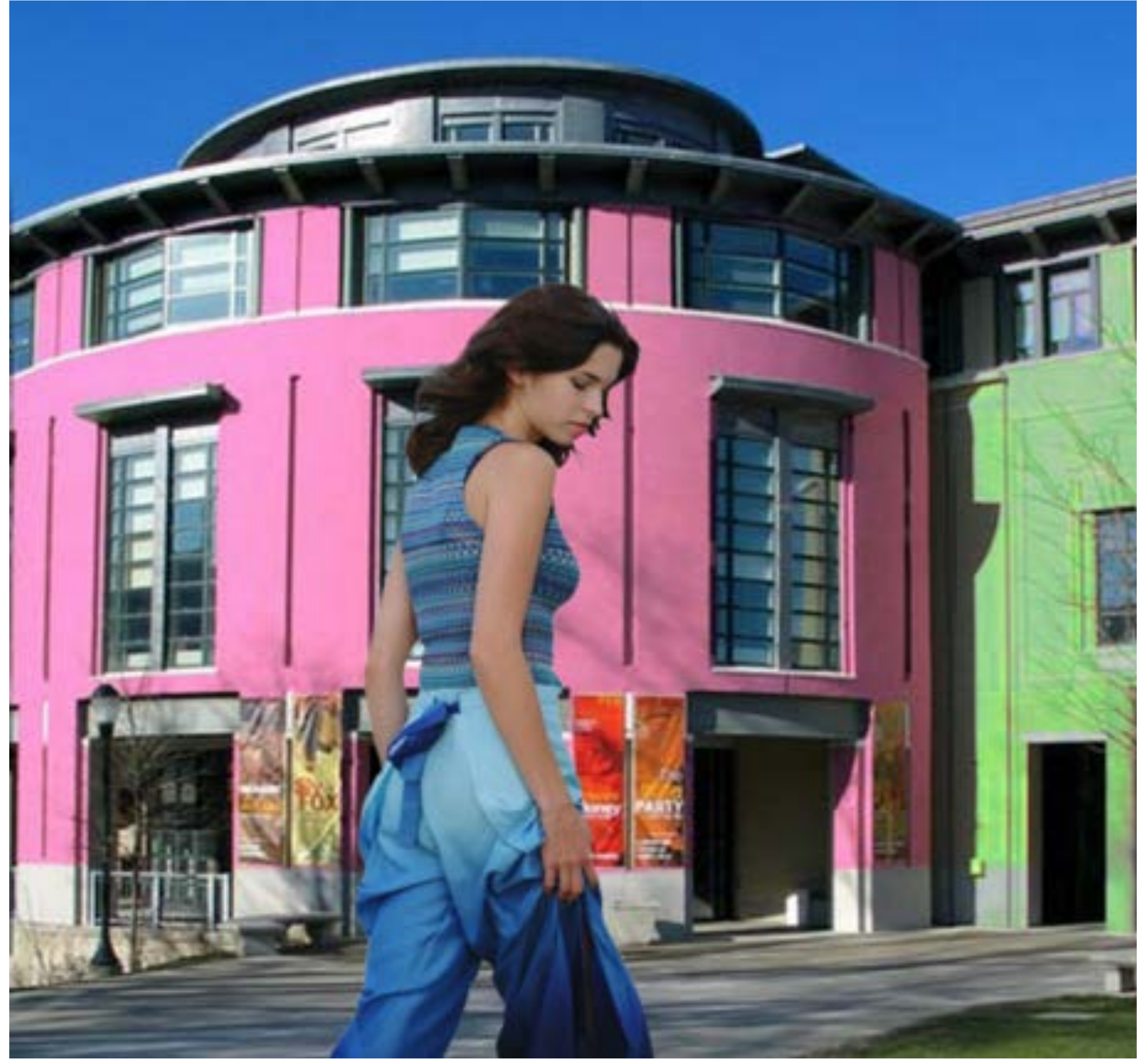
Generalized Barycentric Coordinates

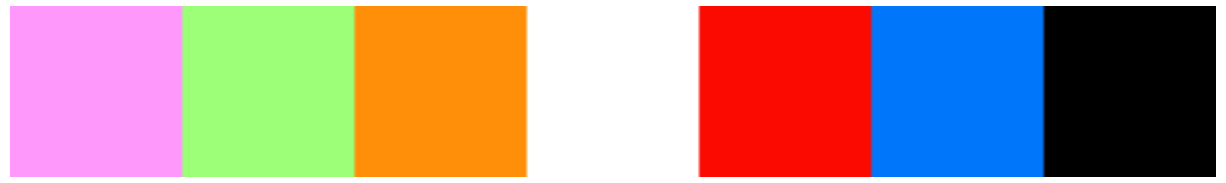
- Fast
- No parameters to tune
- Does not guarantee spatial smoothness



Extracting mixing weights

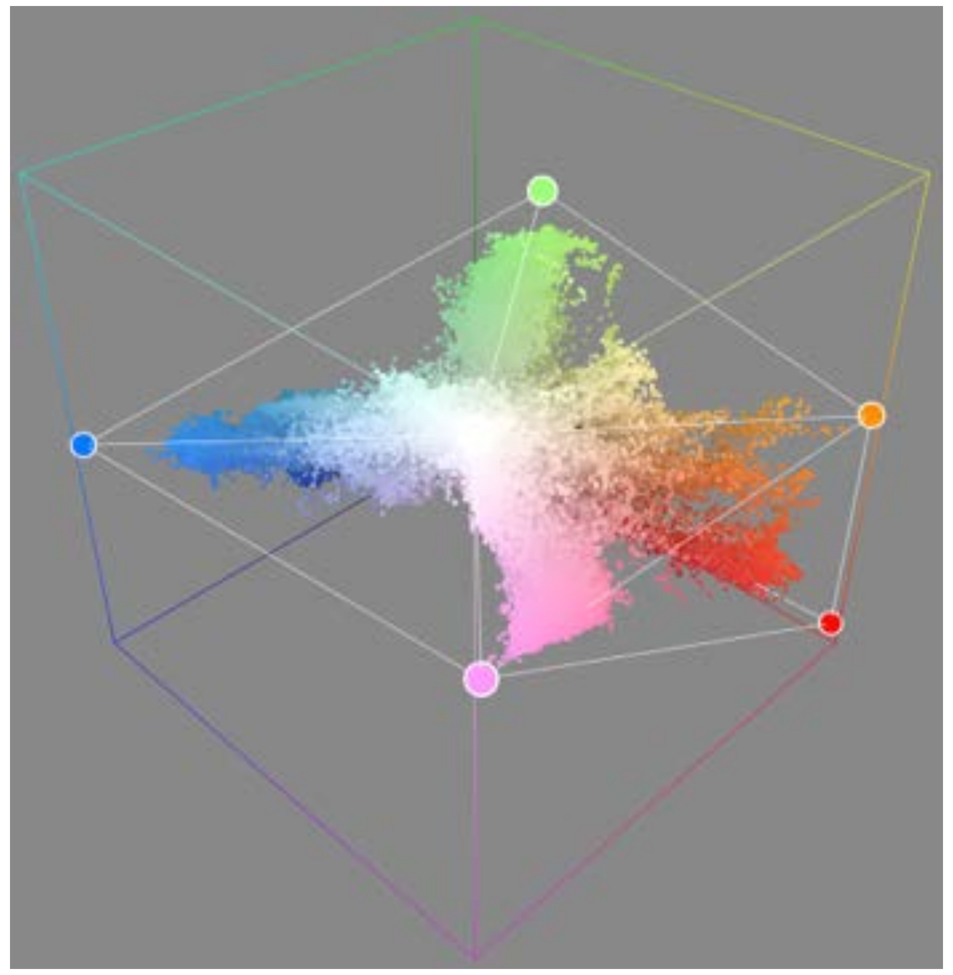
image





palette

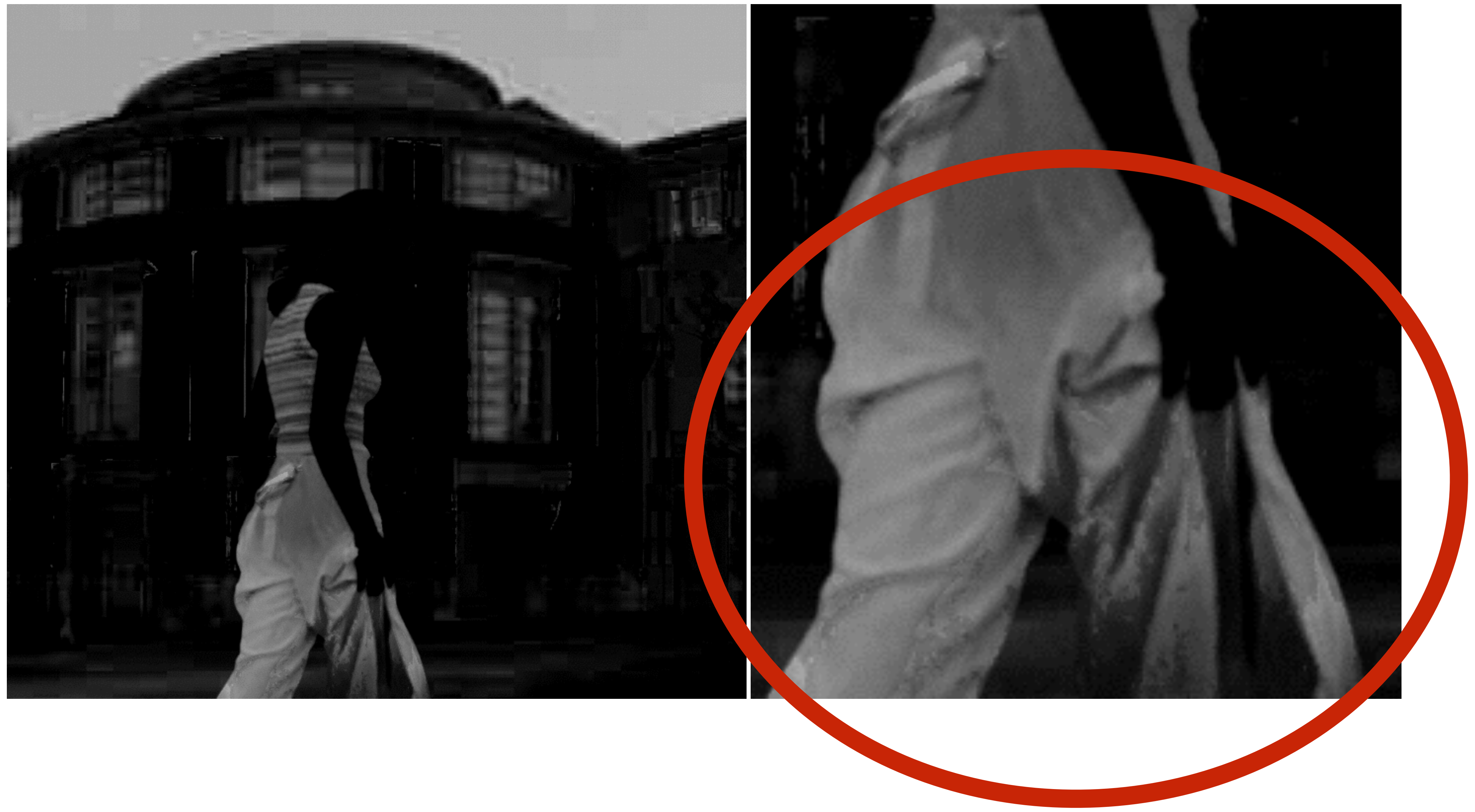
RGB-space



~~**Optimization**~~

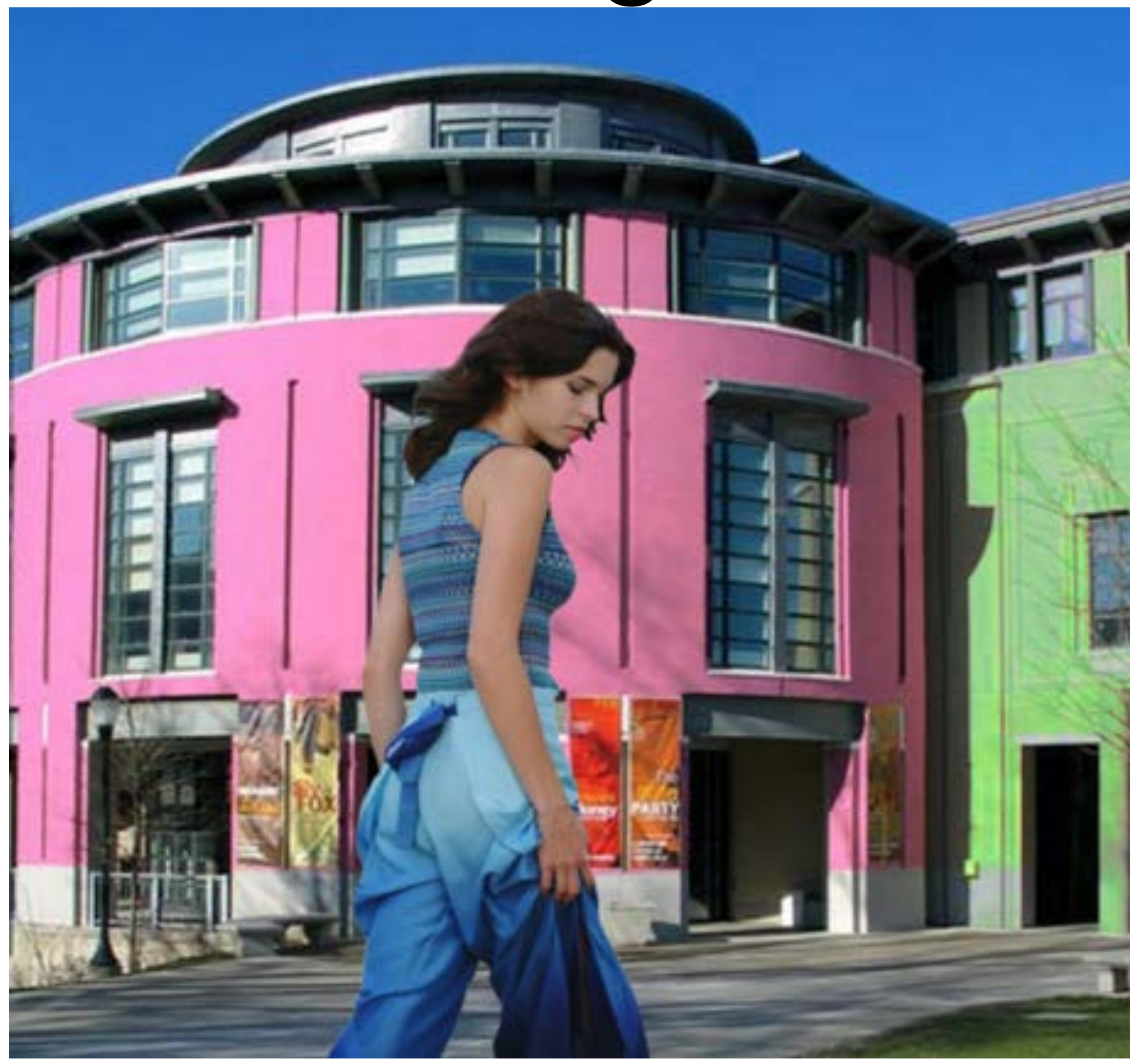
Generalized Barycentric Coordinates

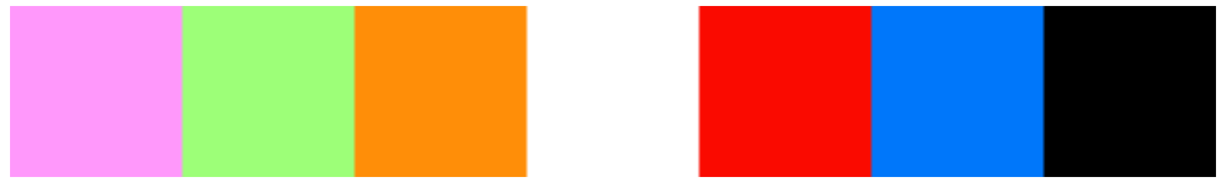
- Fast
- No parameters to tune
- Does not guarantee spatial smoothness



Extracting mixing weights

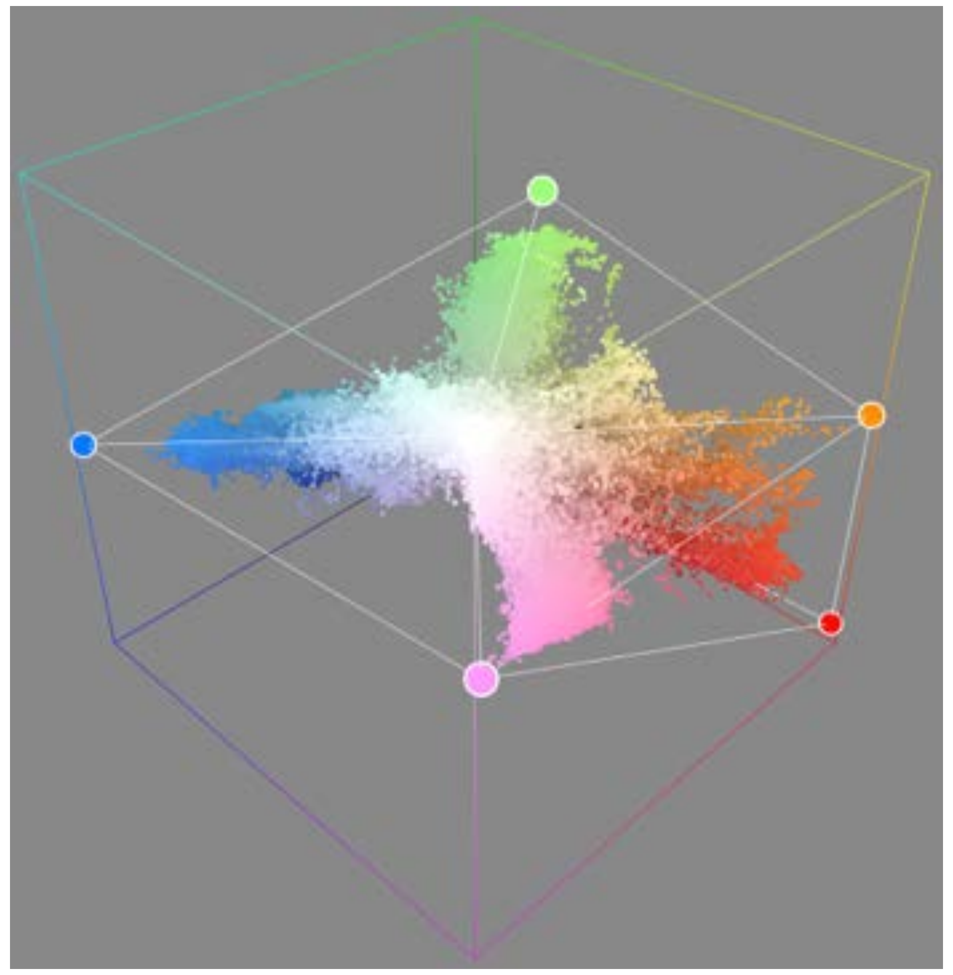
image





palette

RGB-space

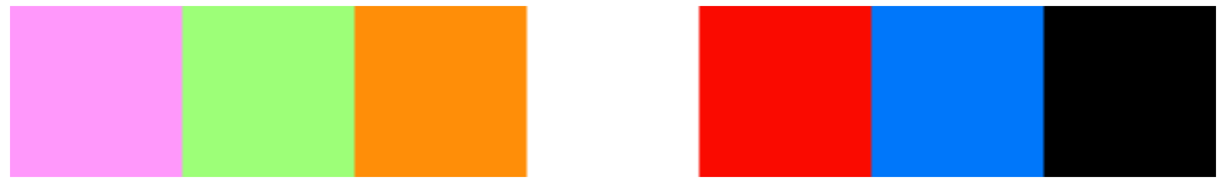


~~**Optimization**~~

~~**Generalized Barycentric Coordinates**~~

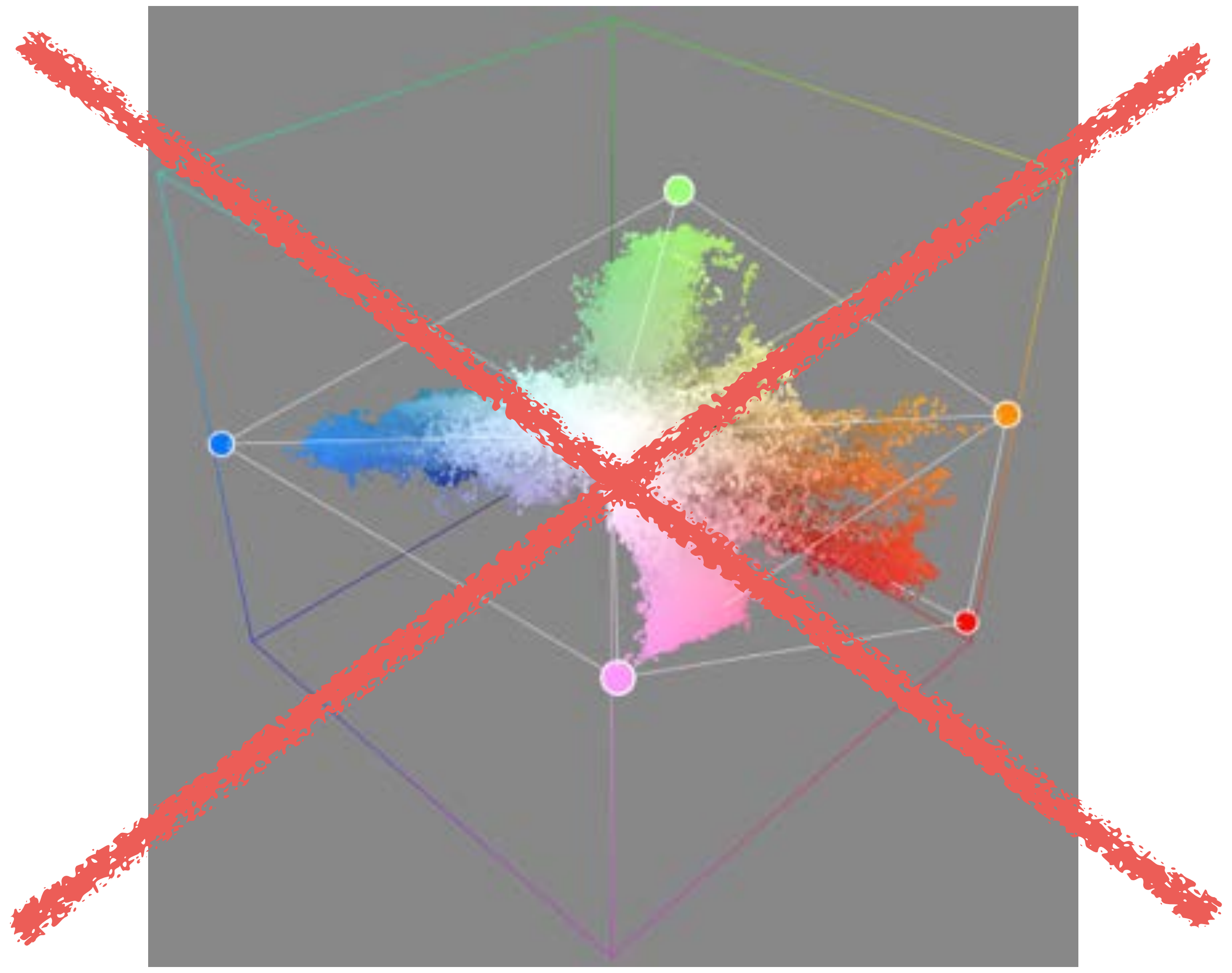
Extracting mixing weights

image



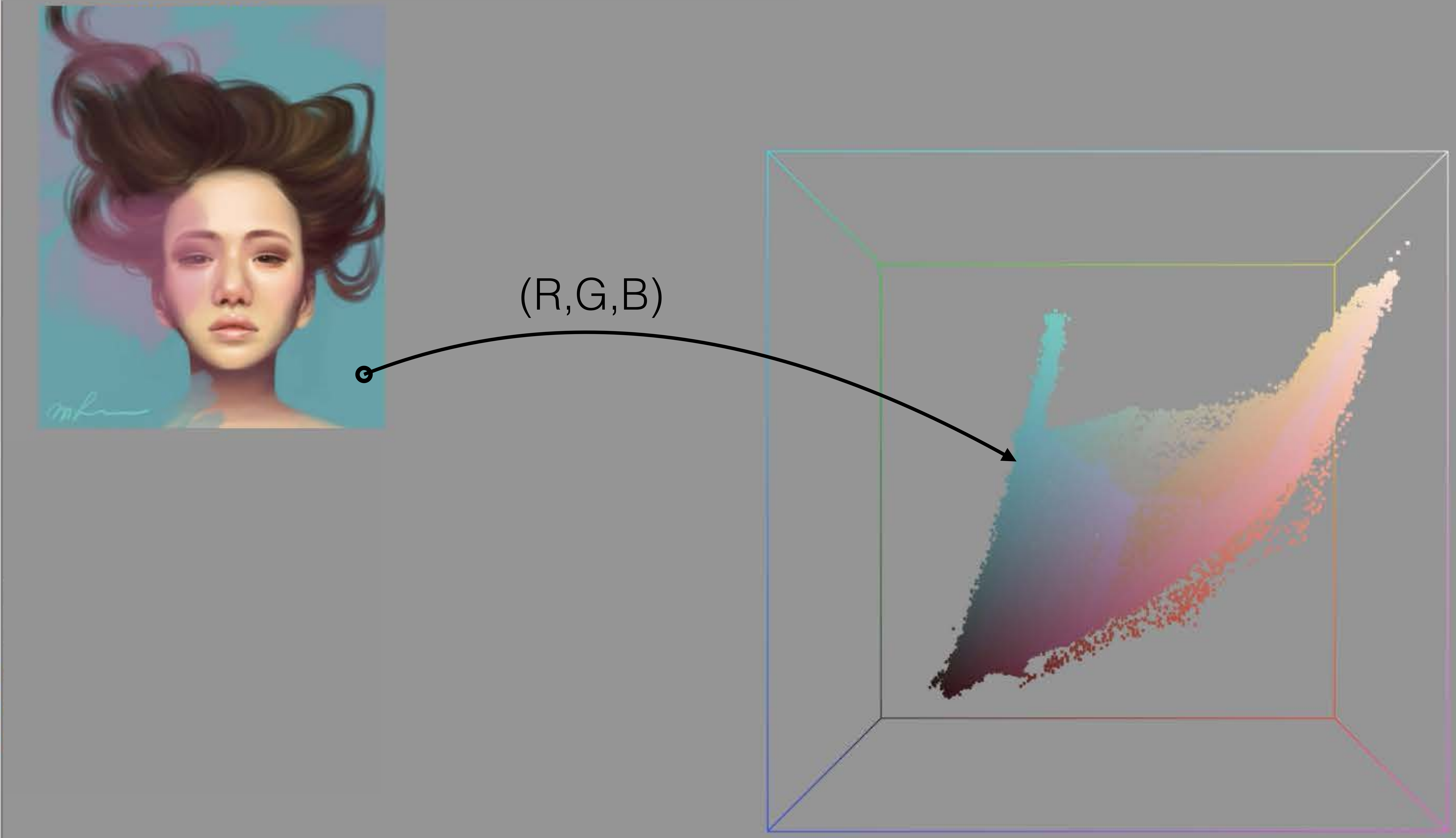
palette

RGB-space

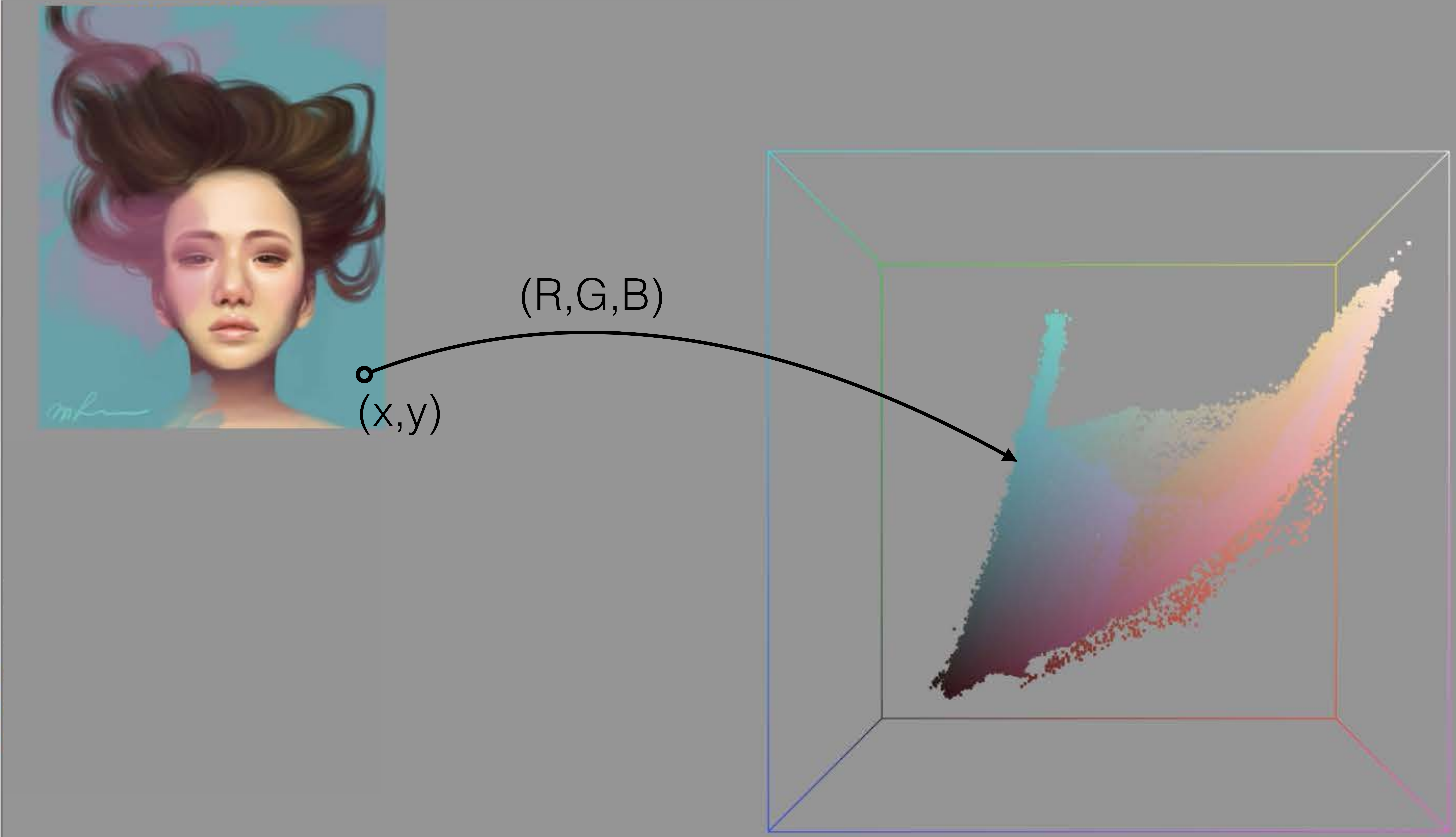


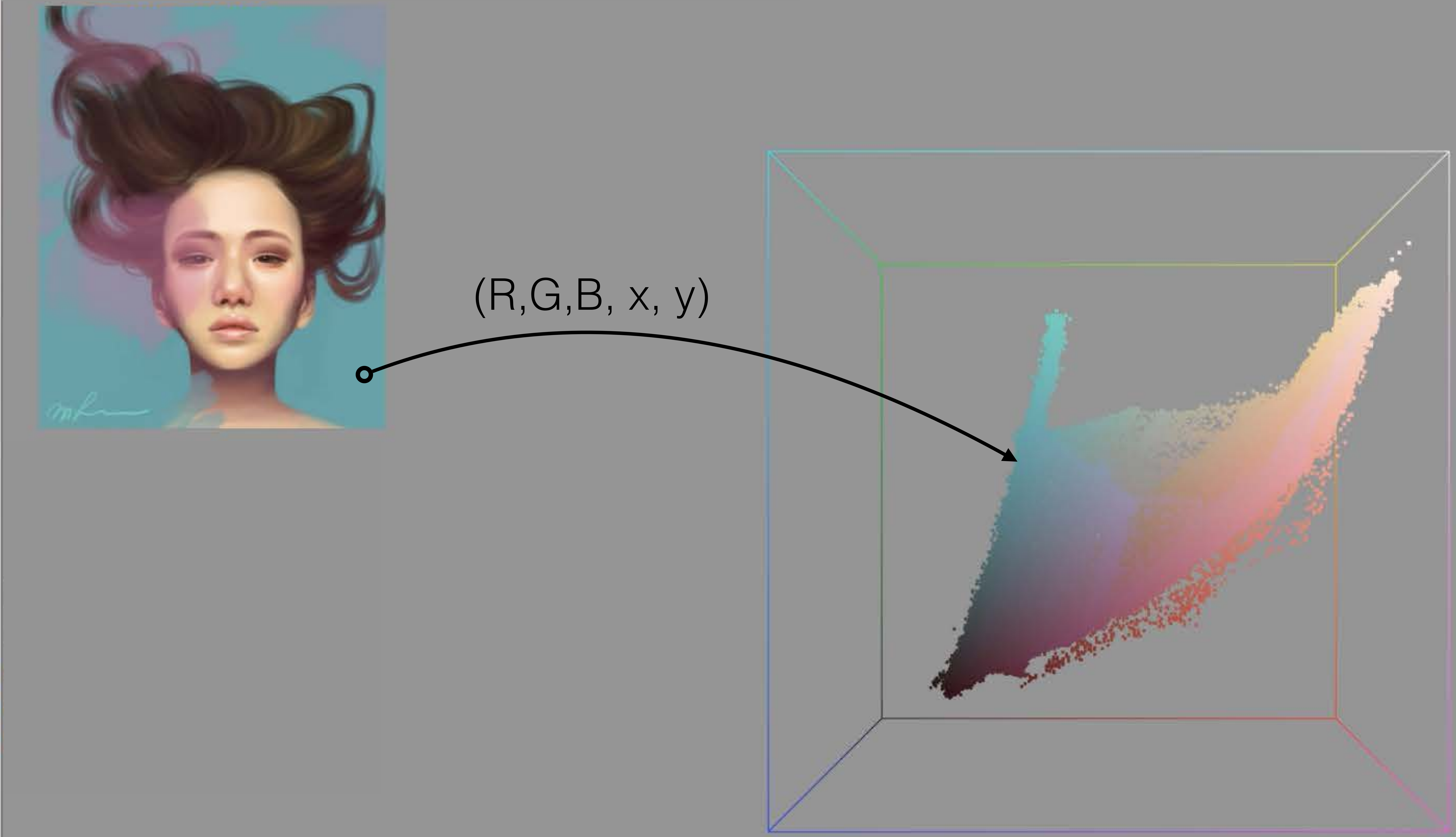
~~Optimization~~

~~Generalized Barycentric Coordinates~~

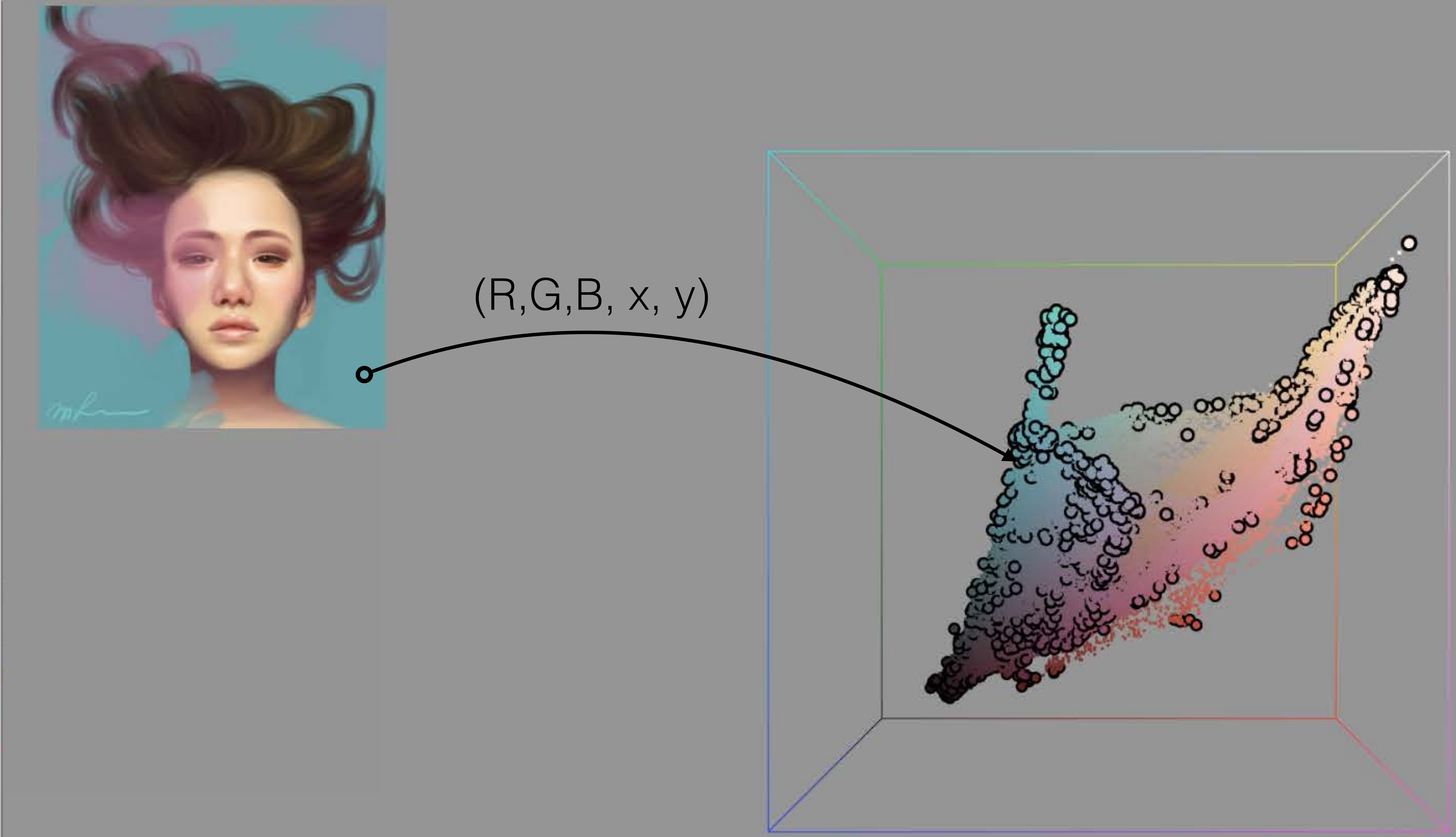


(R,G,B)

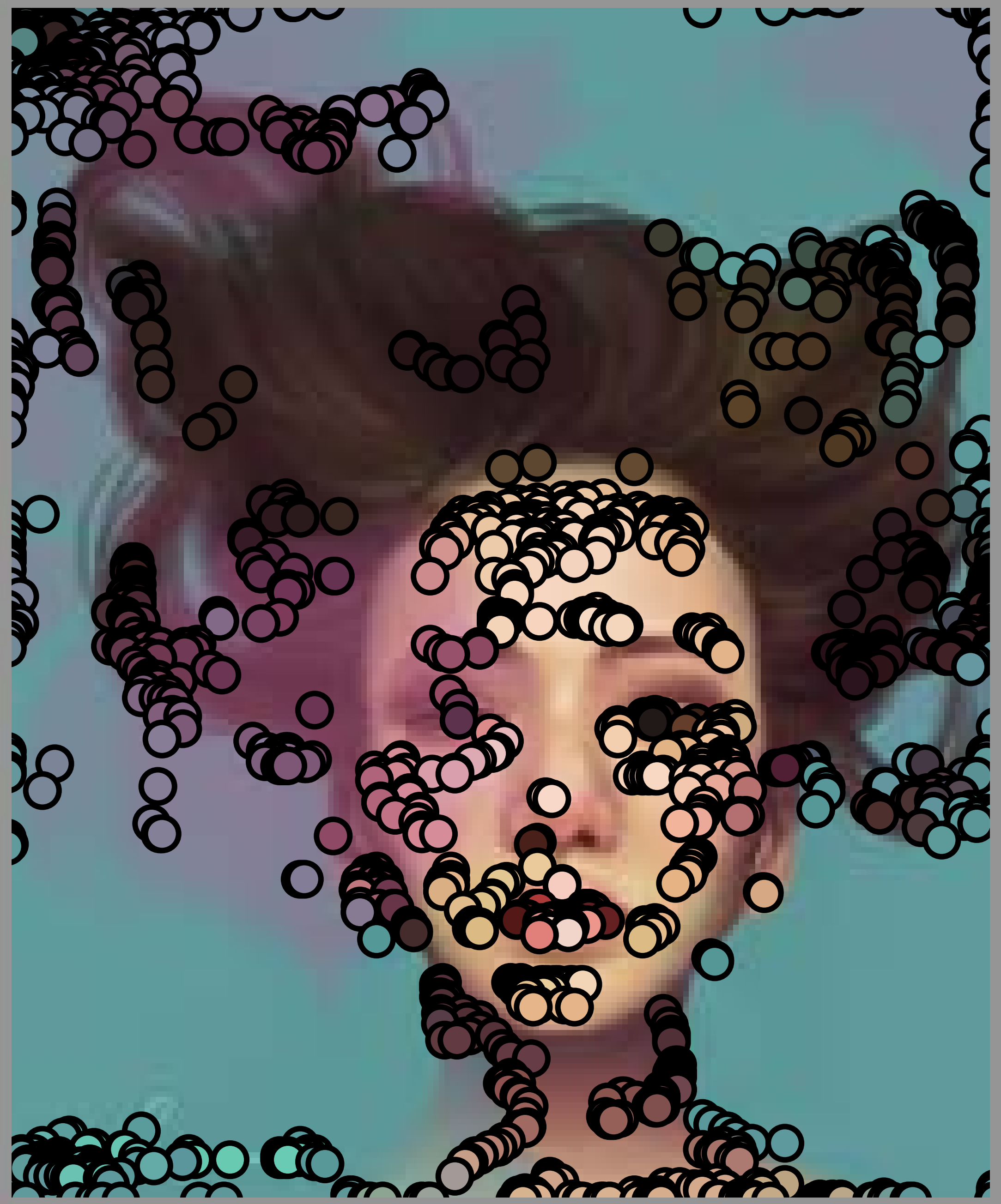

(R,G,B)
(x,y)

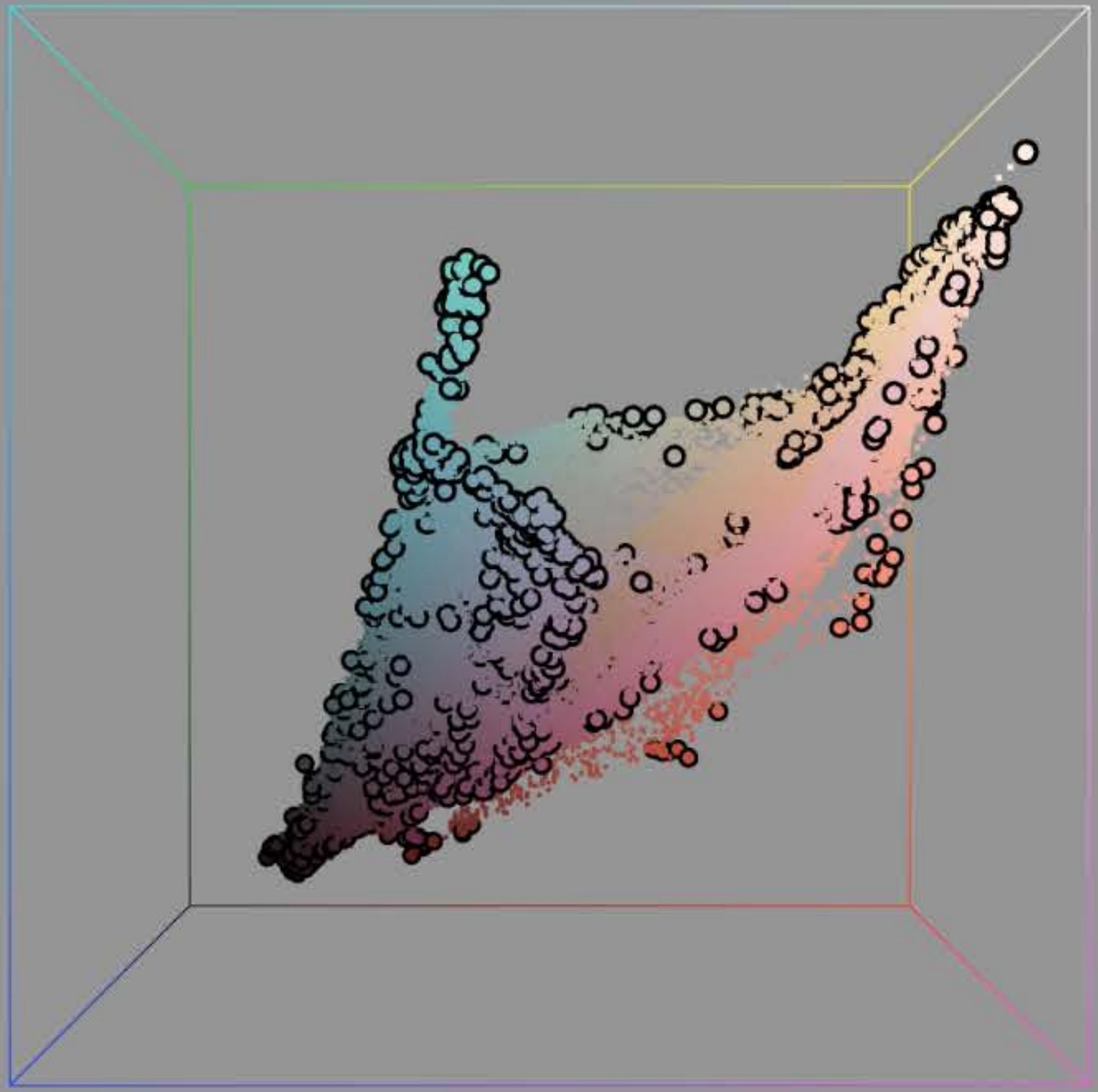


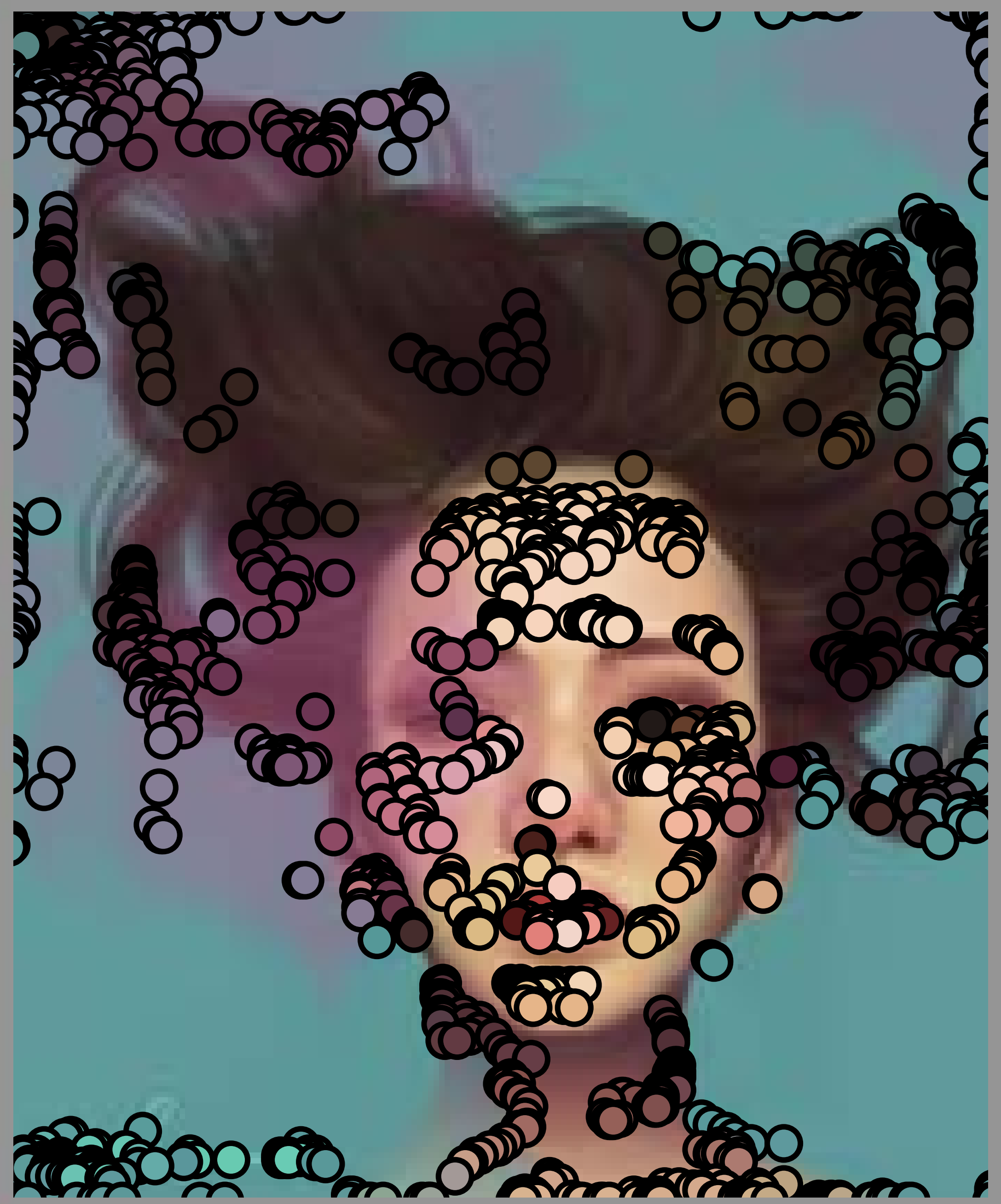
(R,G,B, x, y)

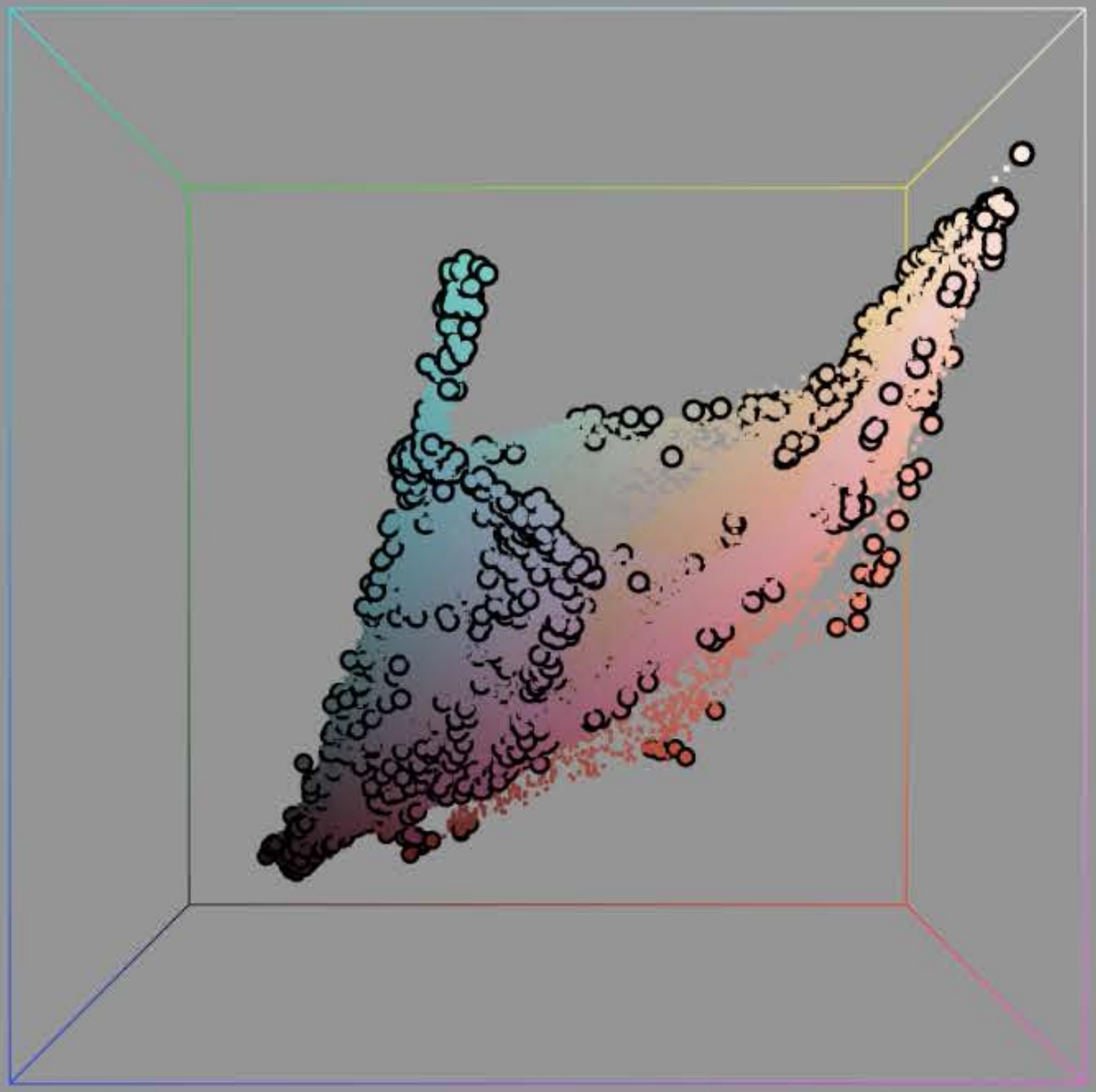


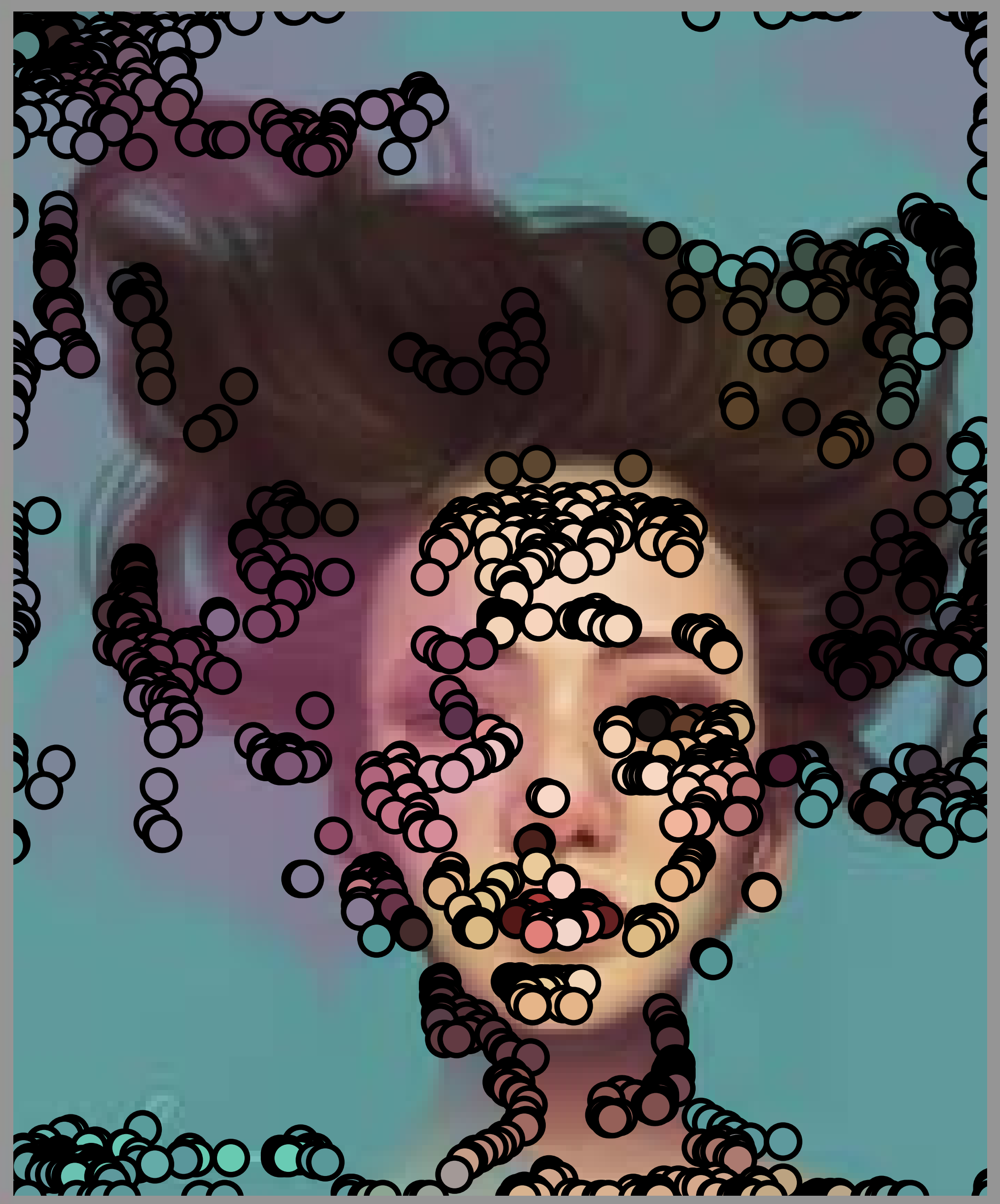
(R,G,B, x, y)



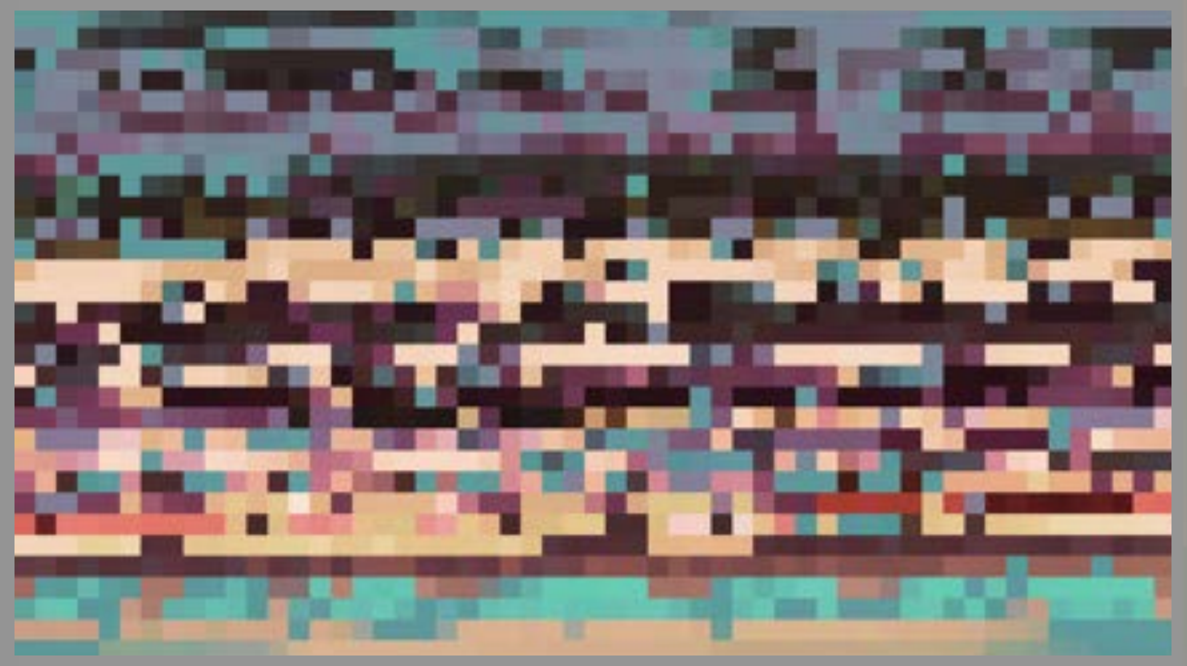


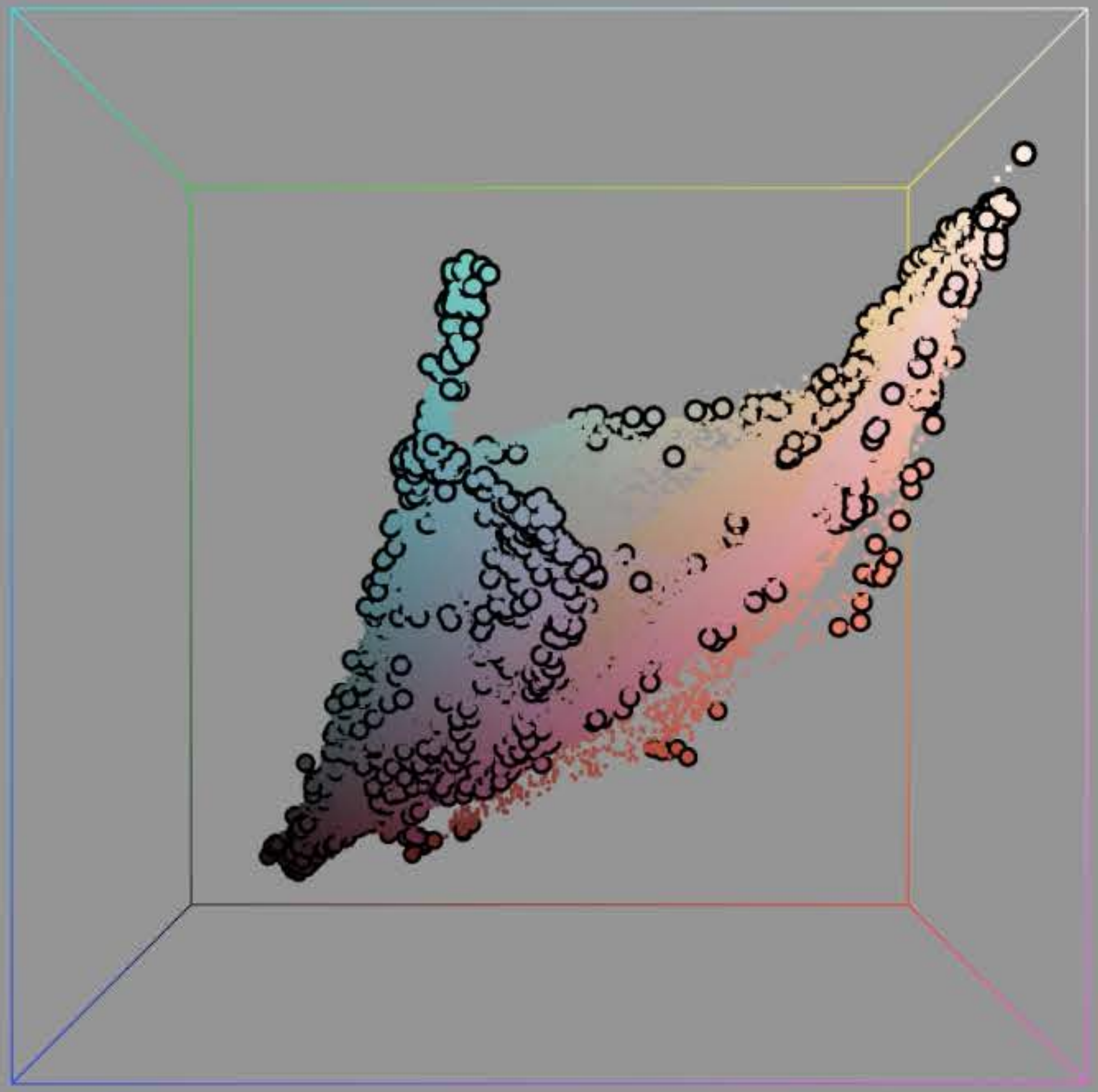


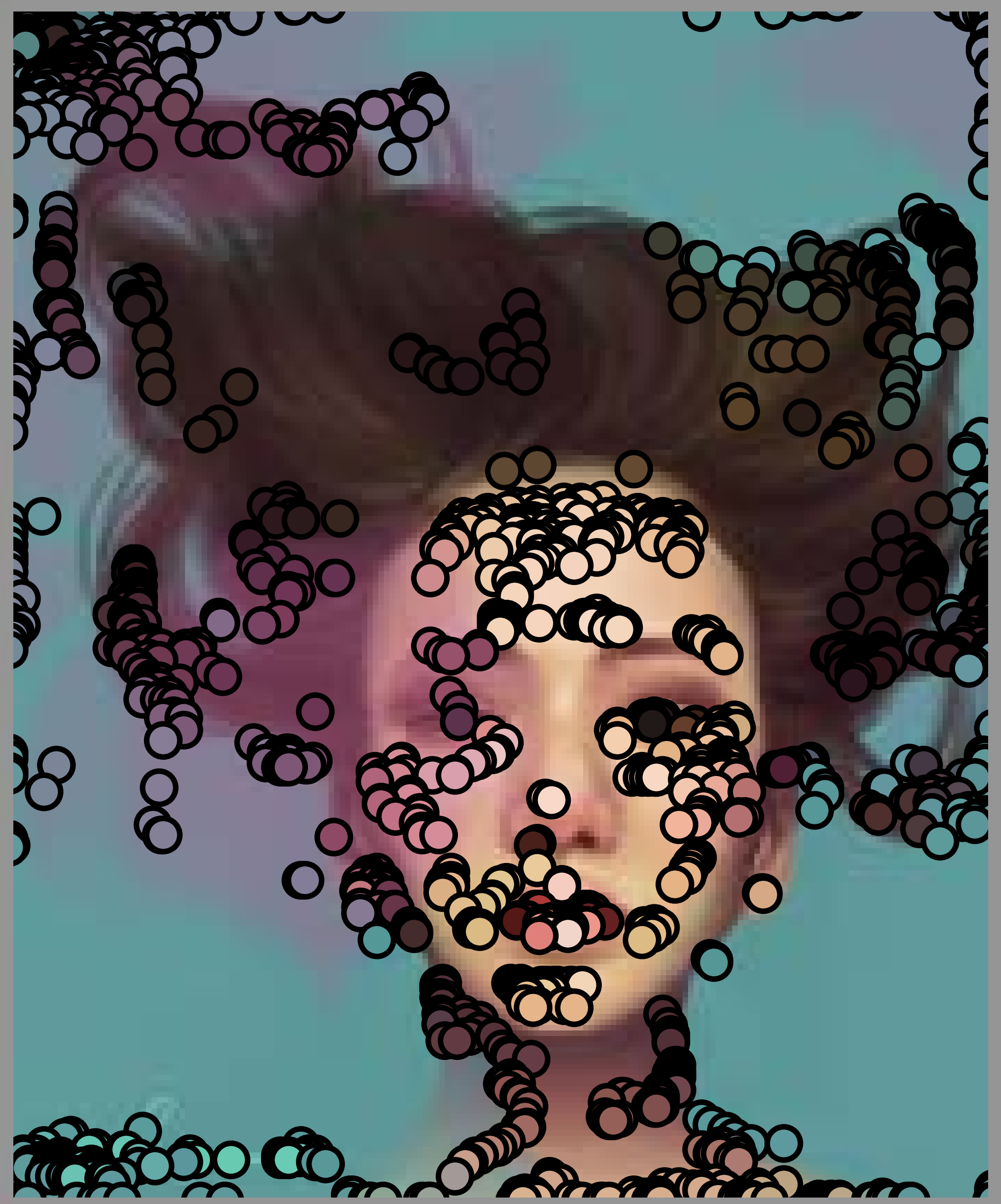




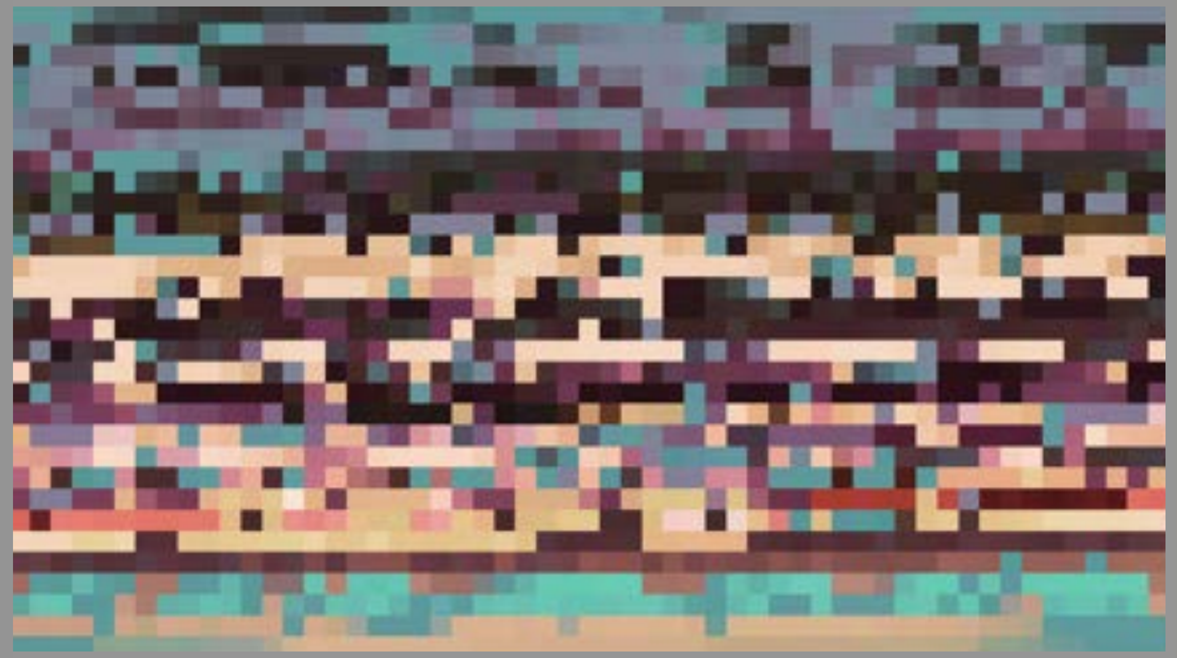
RGBXY palette
(projected to RGB)



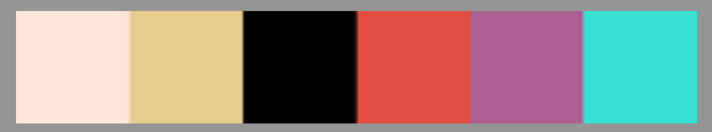


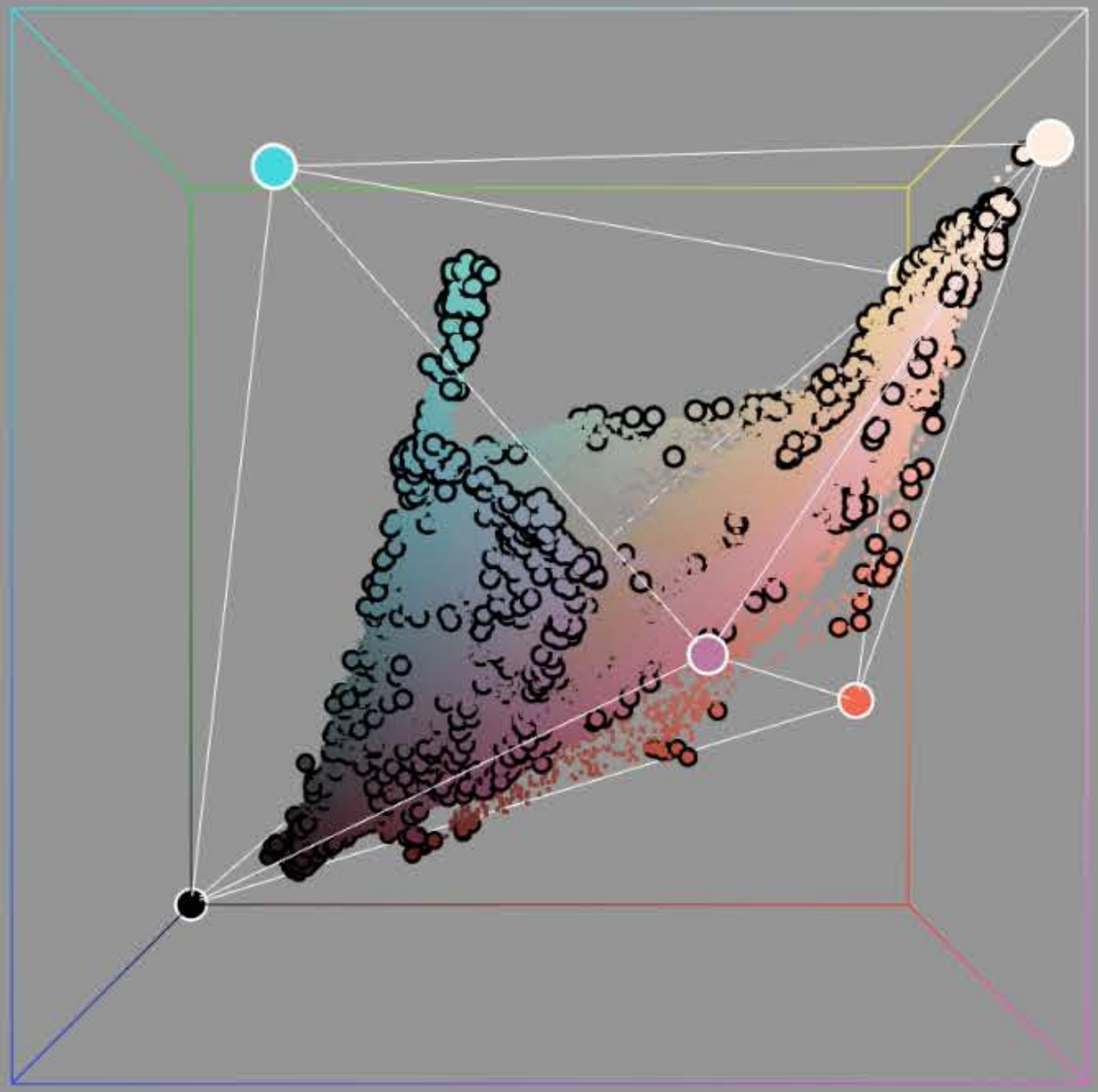


RGBXY palette
(projected to RGB)



RGB palette





Two-level decomposition

RGB palette

mixing weights $\mathbf{W}$

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

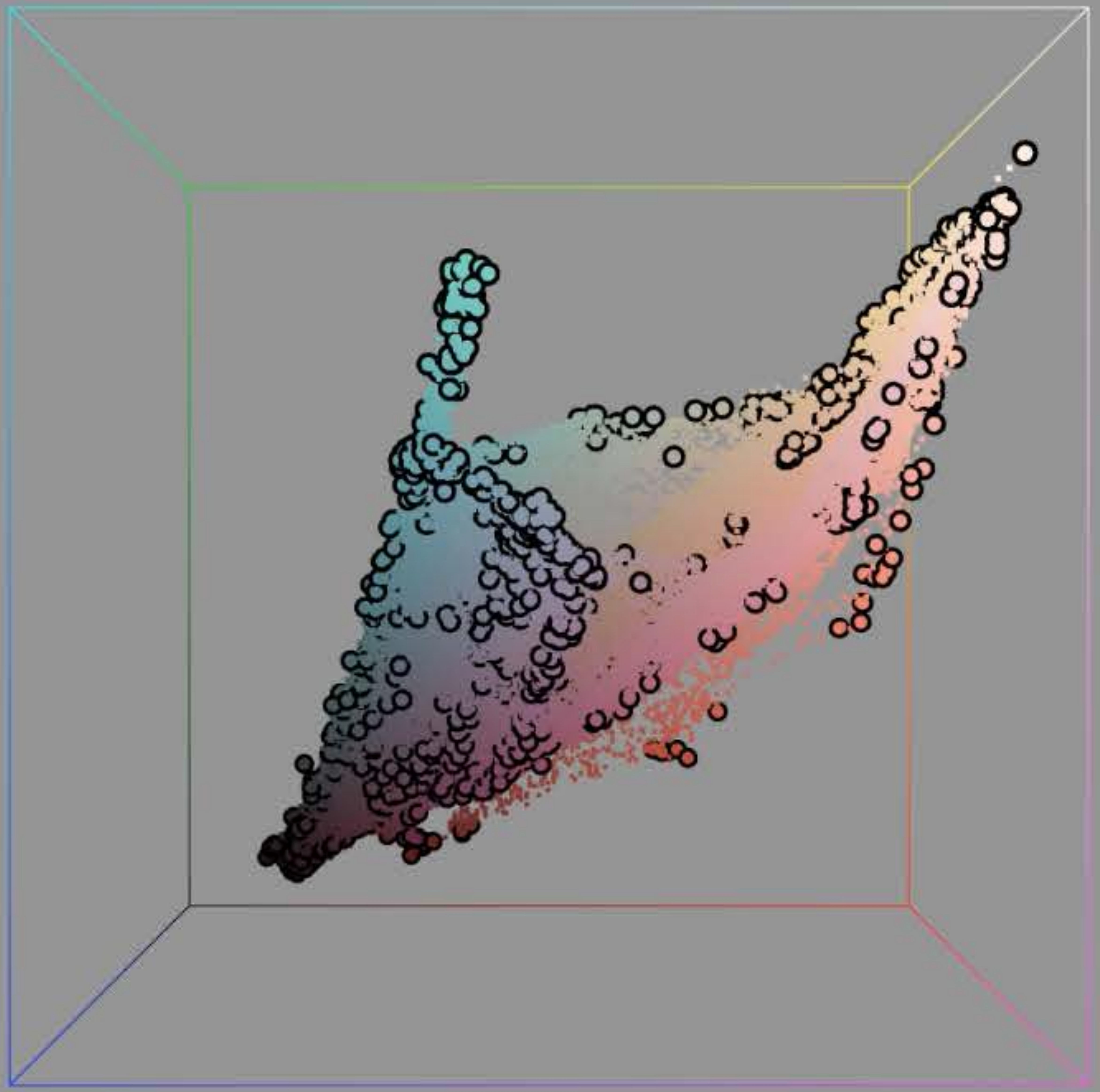
RGB palette

RGBXY vertices (projected to RGB)

W_{RGBXY}

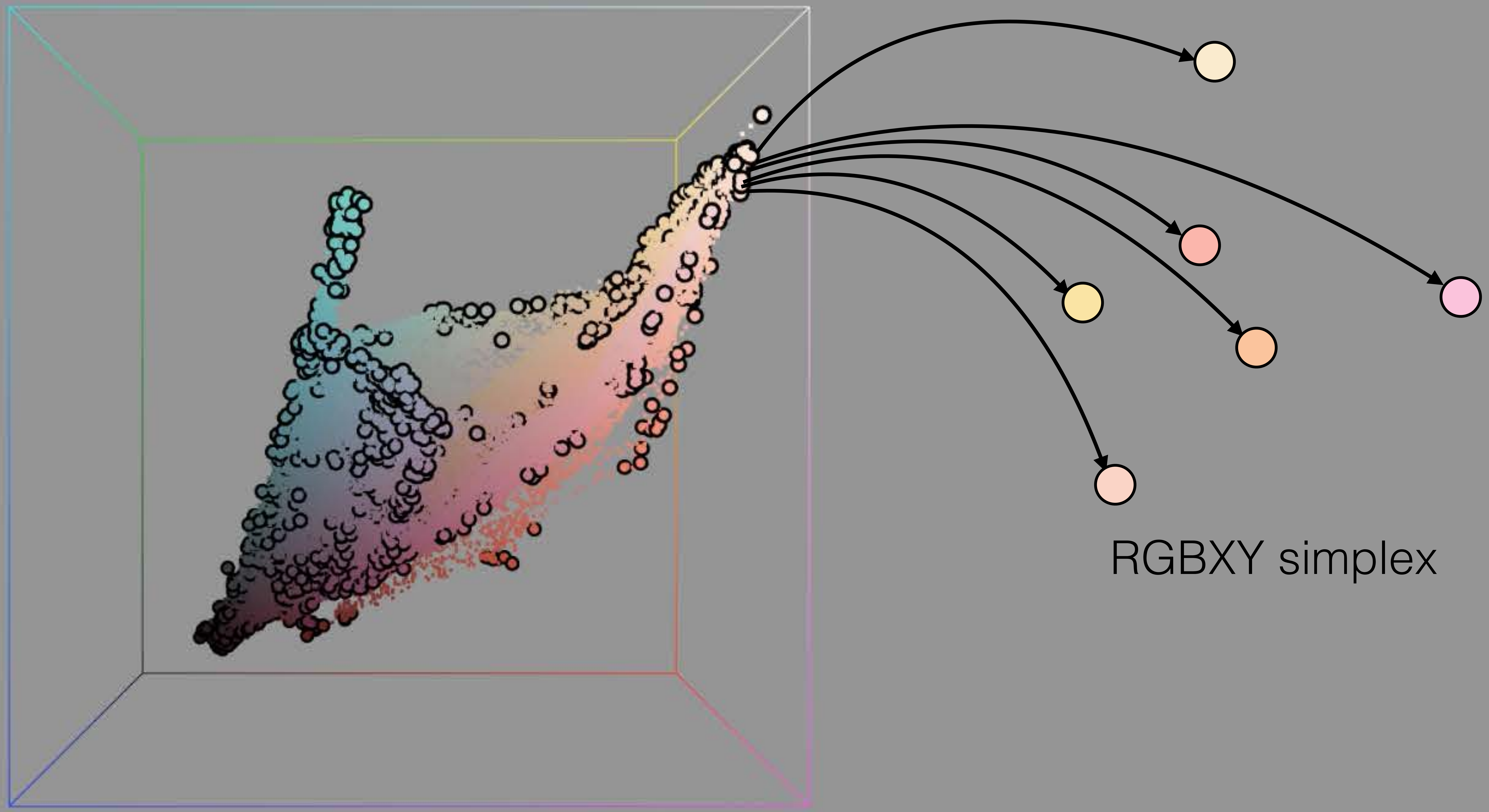
image

Delaunay Tessellation in RGBXY space



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

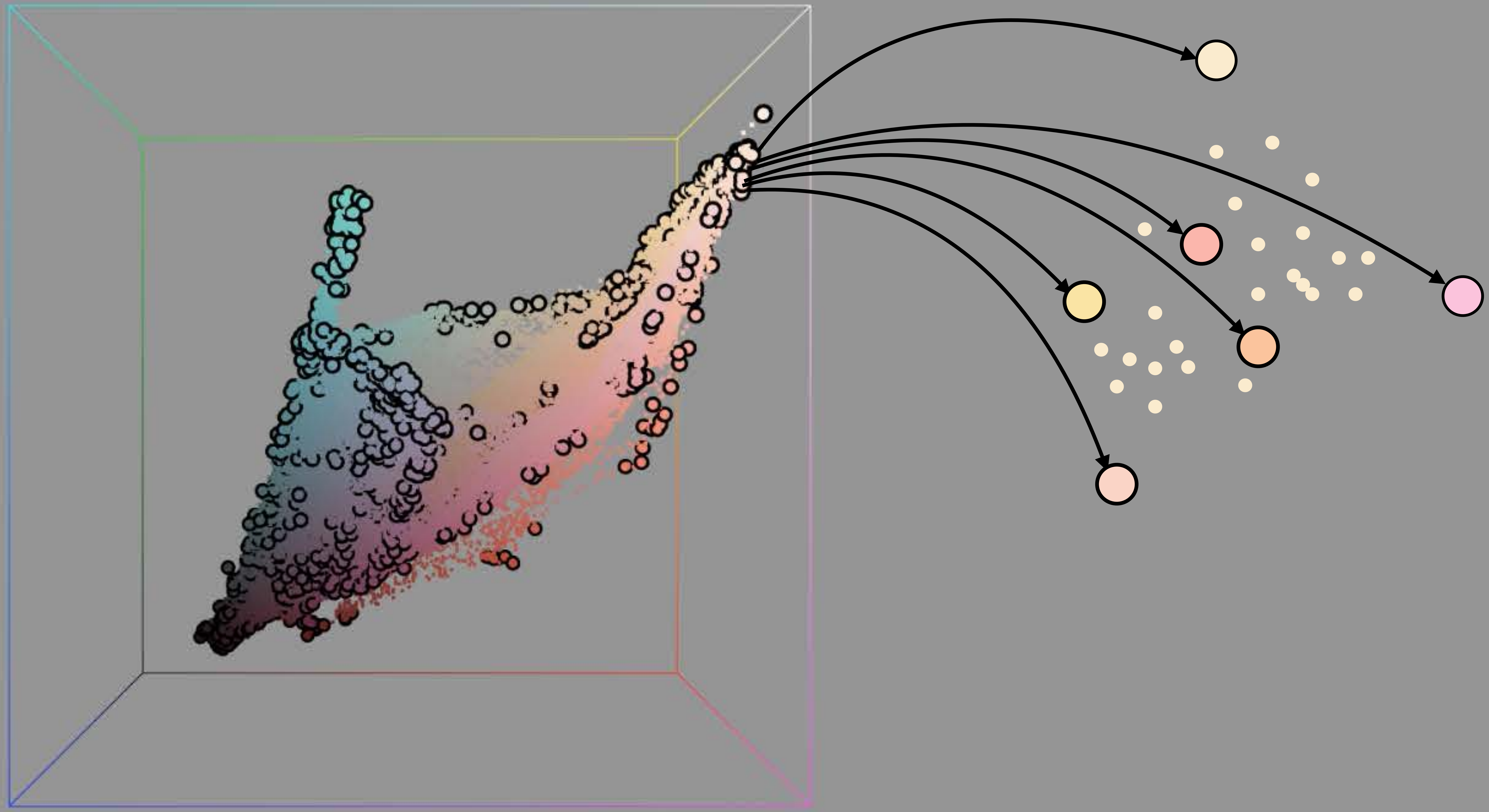
Delaunay Tessellation in RGBXY space



RGBXY simplex

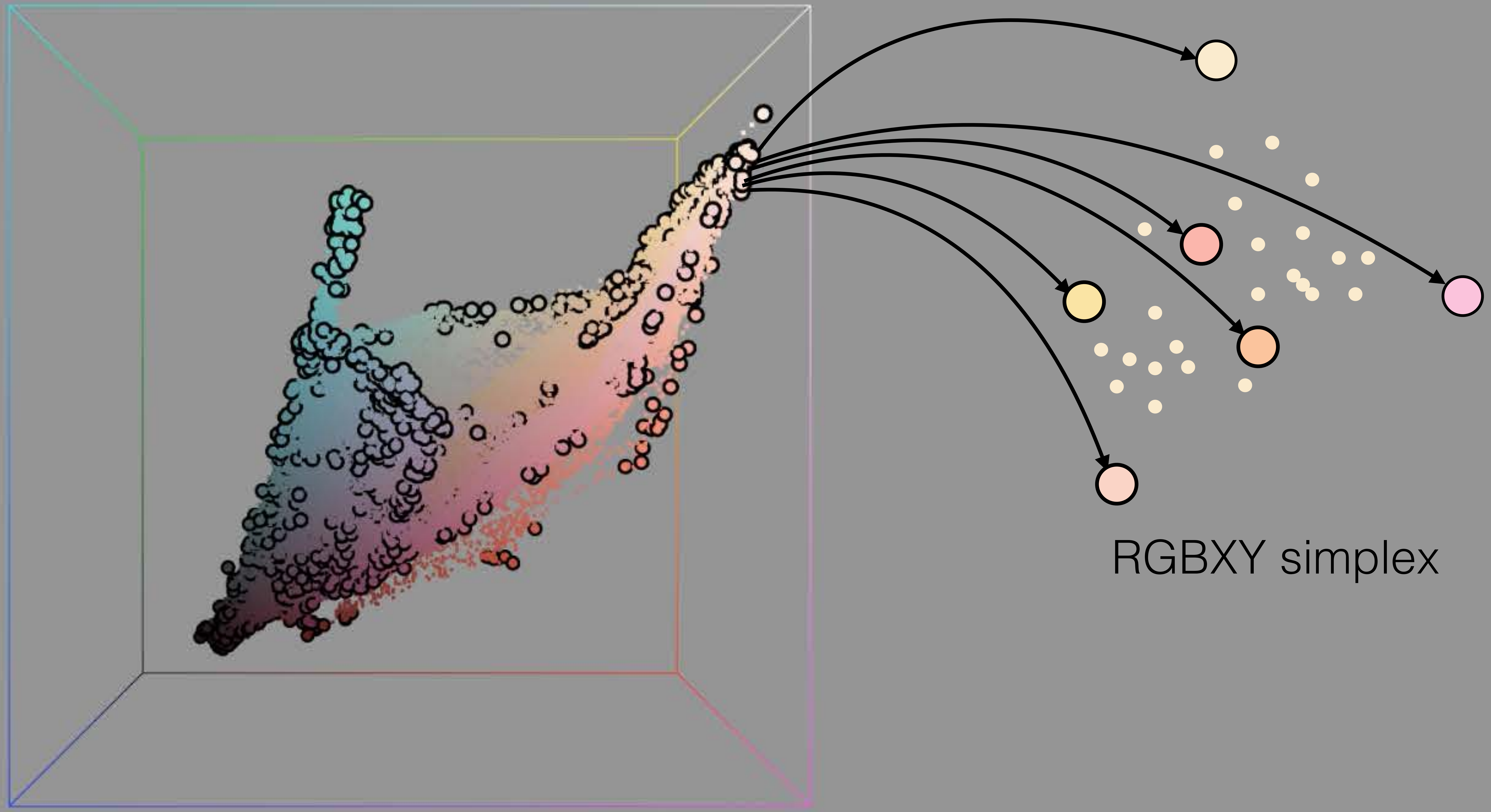
Extract barycentric mixing weights $\mathbf{W}_{\mathbf{RGBXY}}$

Delaunay Tessellation in RGBXY space



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

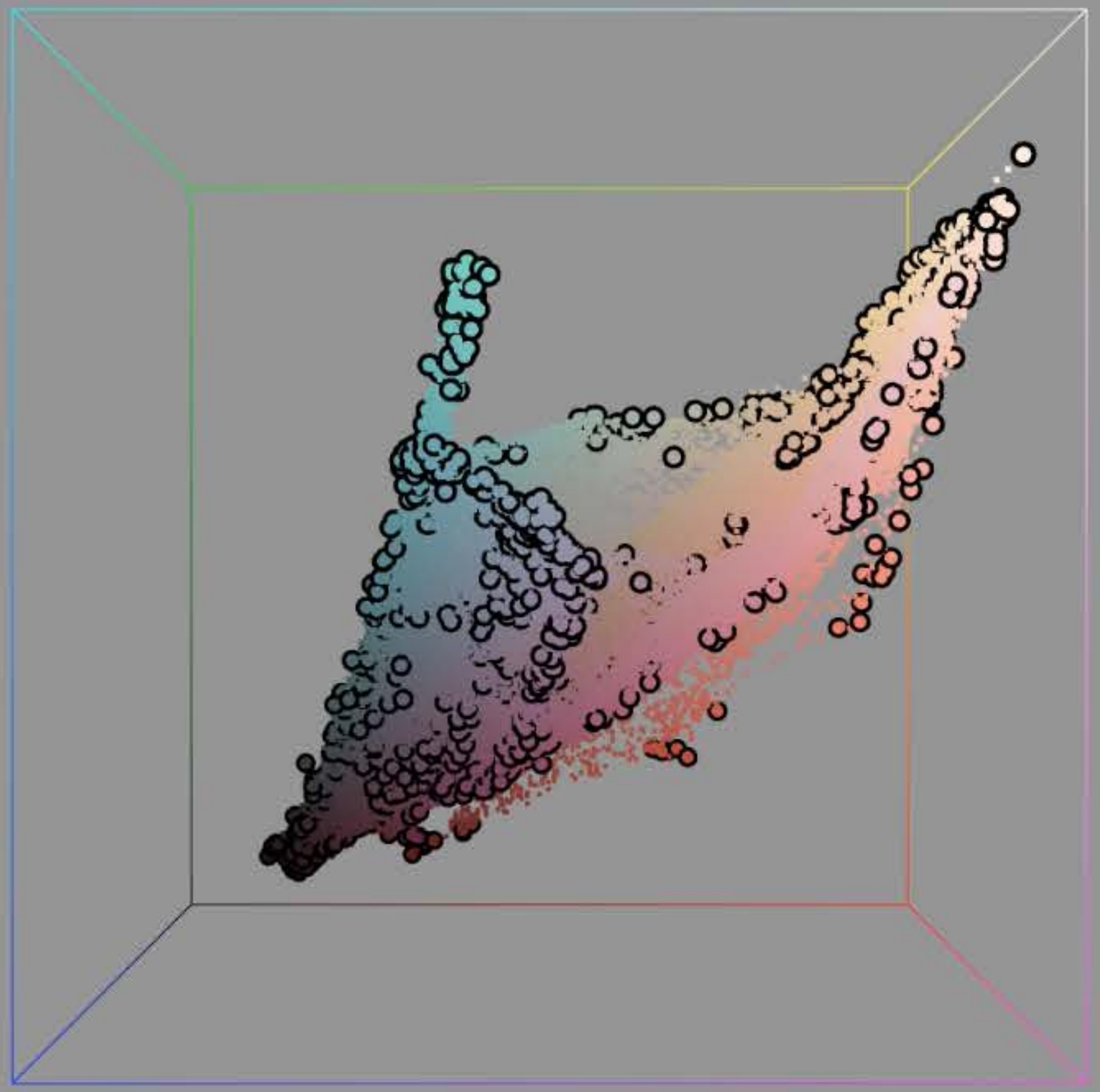
Delaunay Tessellation in RGBXY space

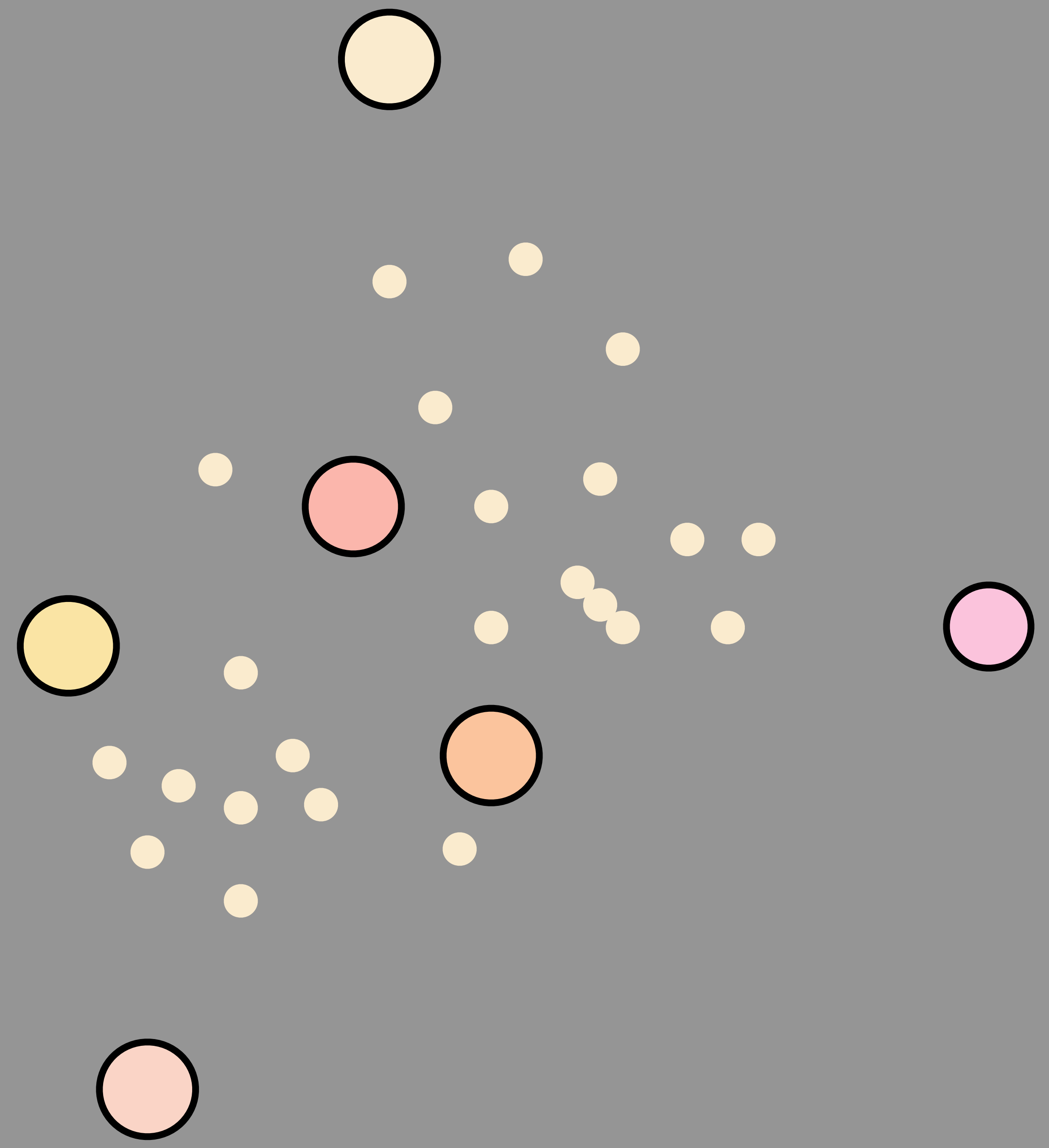


RGBXY simplex

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

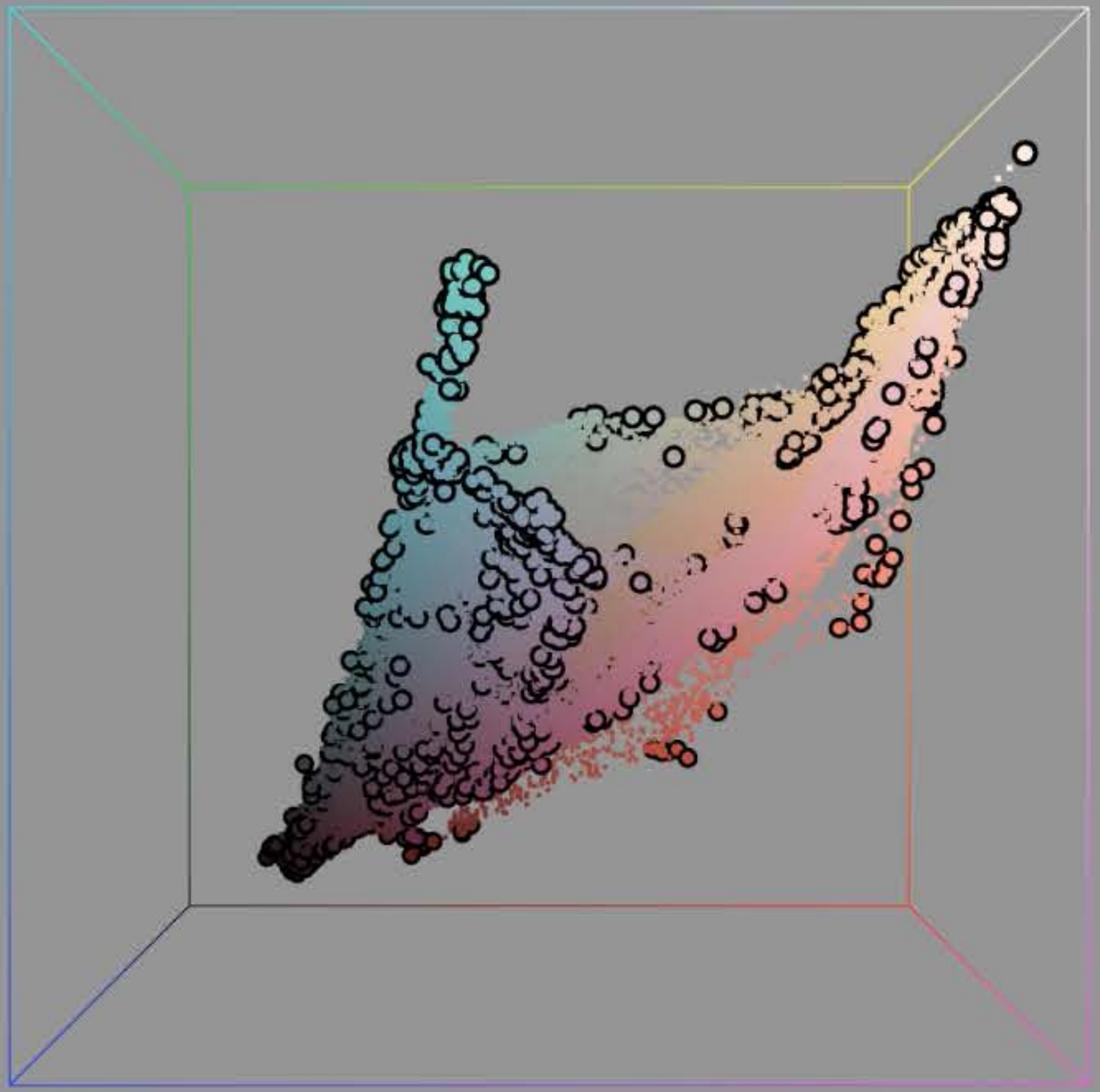
Delaunay Tessellation in RGBXY space

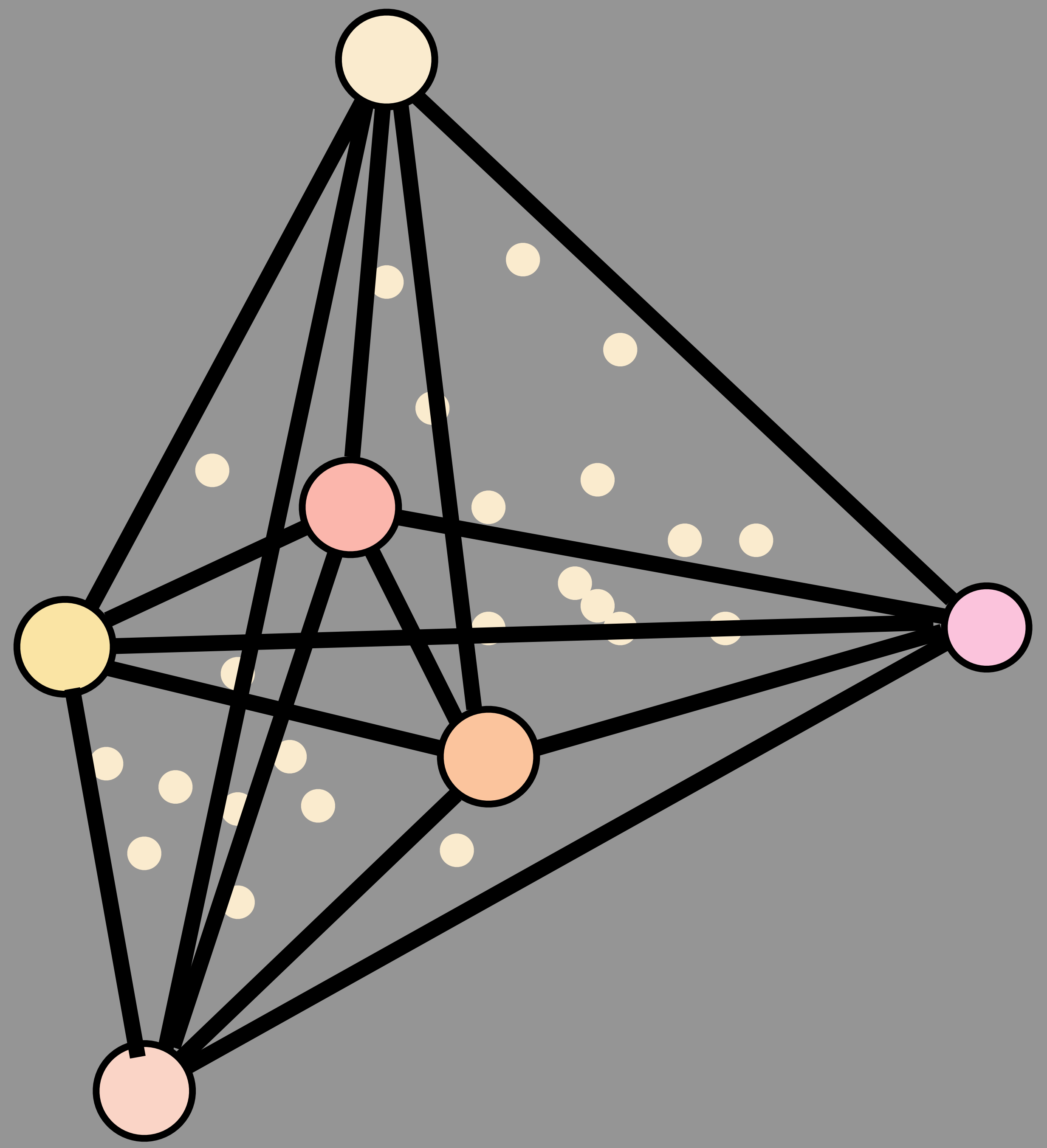




Extract barycentric mixing weights $\mathbf{W}_{\mathbf{RGBXY}}$

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

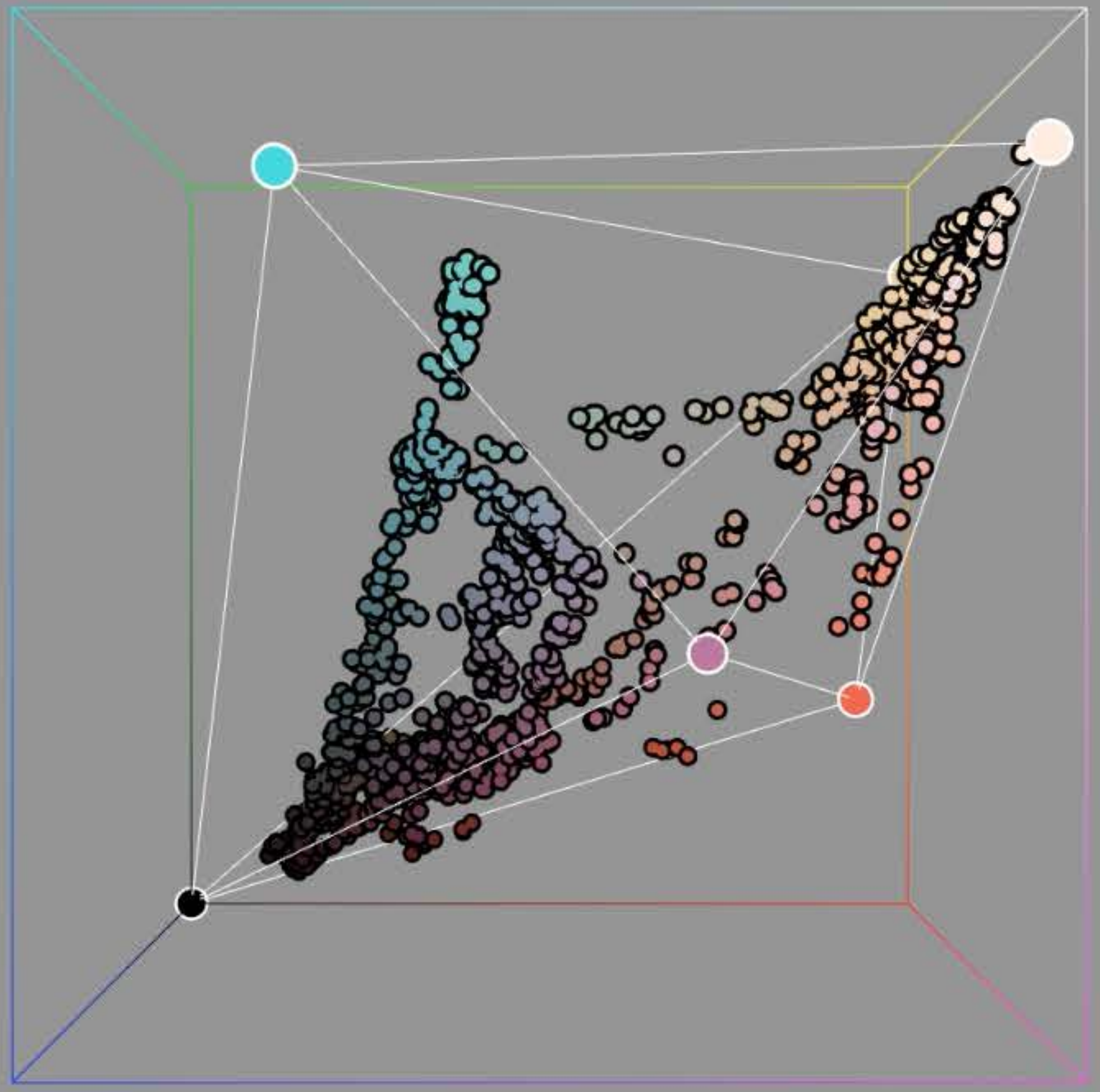
RGB palette

$\mathbf{W}_{\mathbf{RGB}}$

RGBXY vertices
(projected to RGB)

image

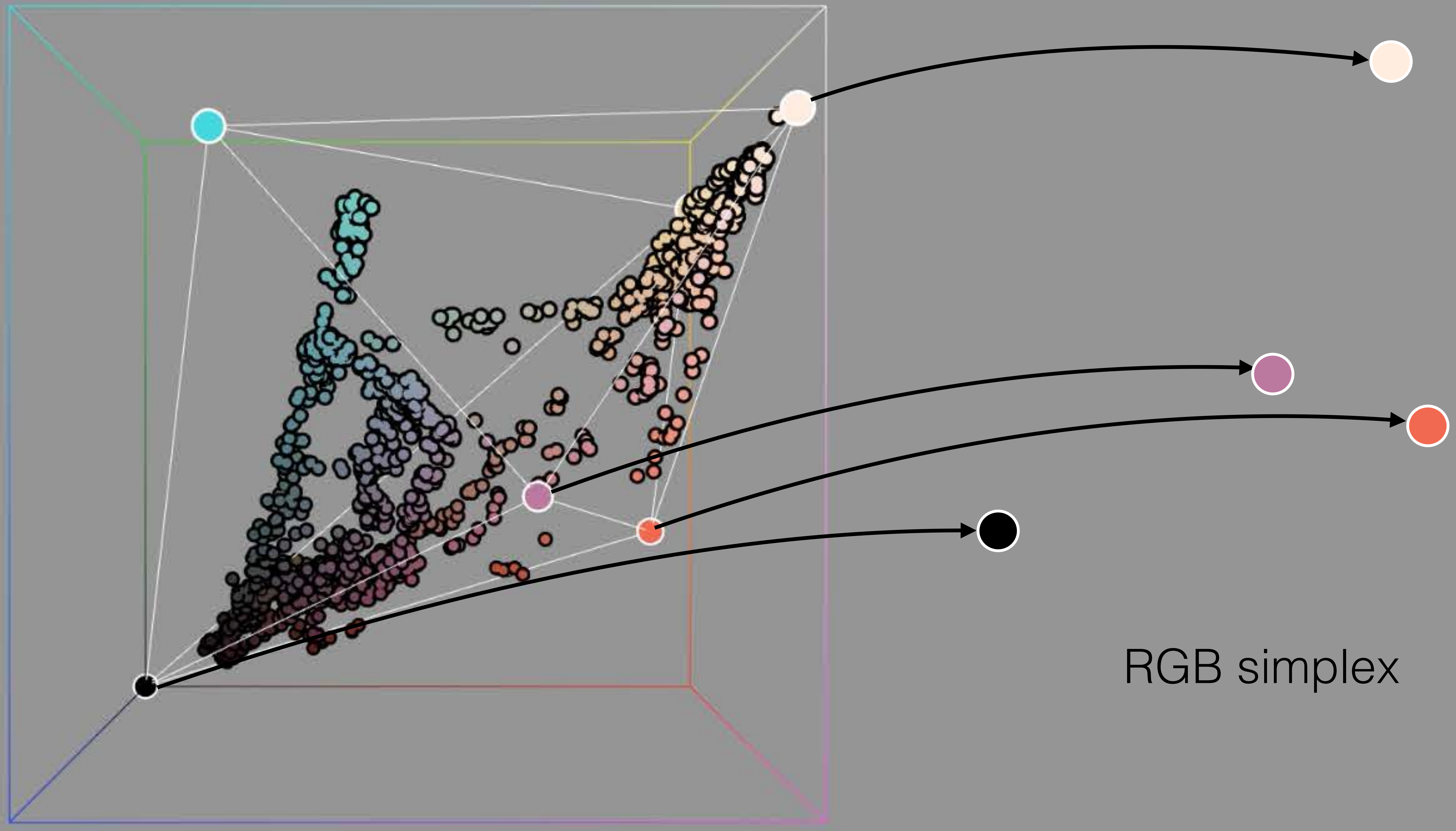
Tessellation in RGB space



RGB simplex

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

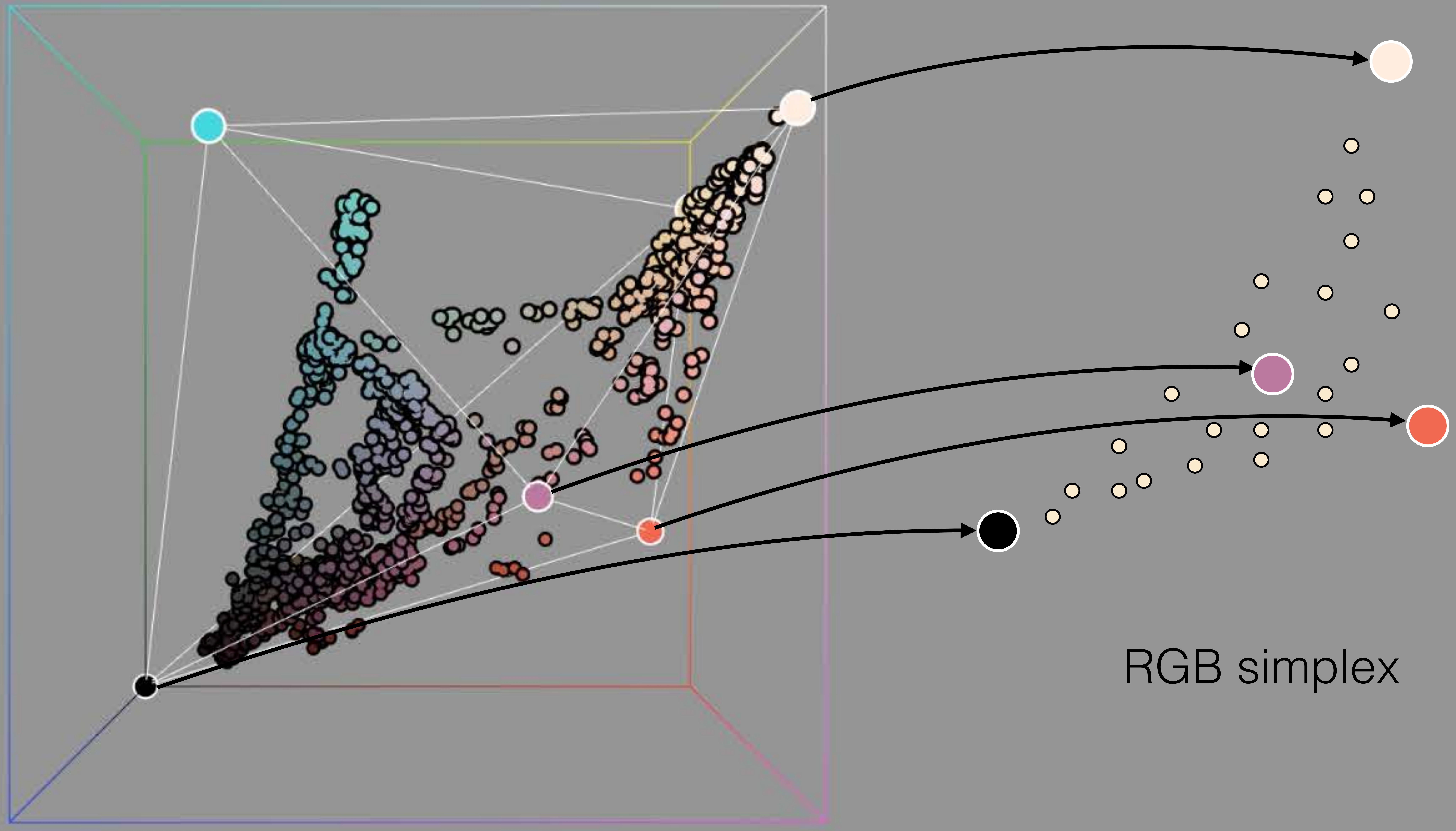
Tessellation in RGB space



RGB simplex

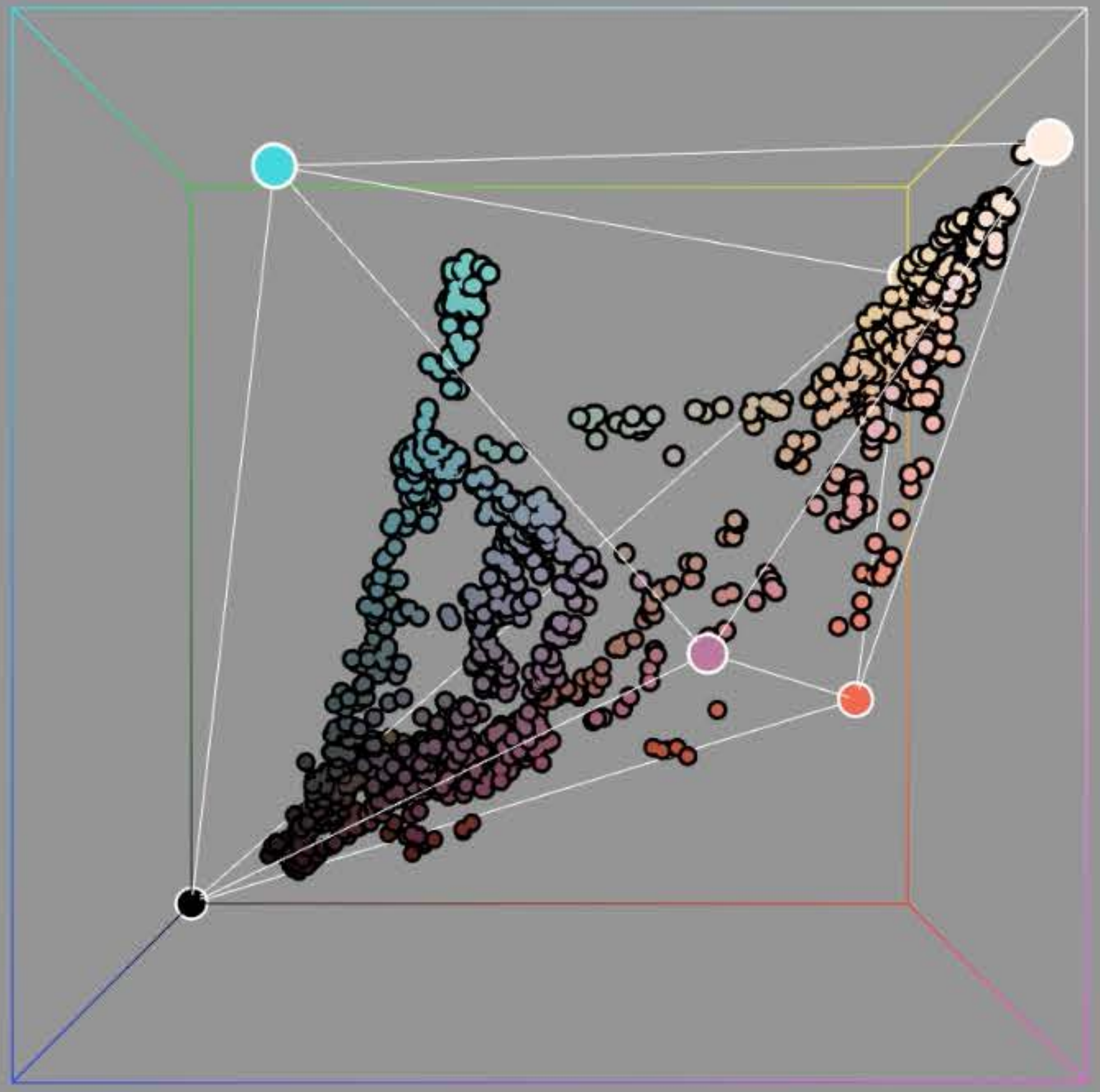
Extract barycentric mixing weights $\mathbf{W}_{\mathbf{RGB}}$

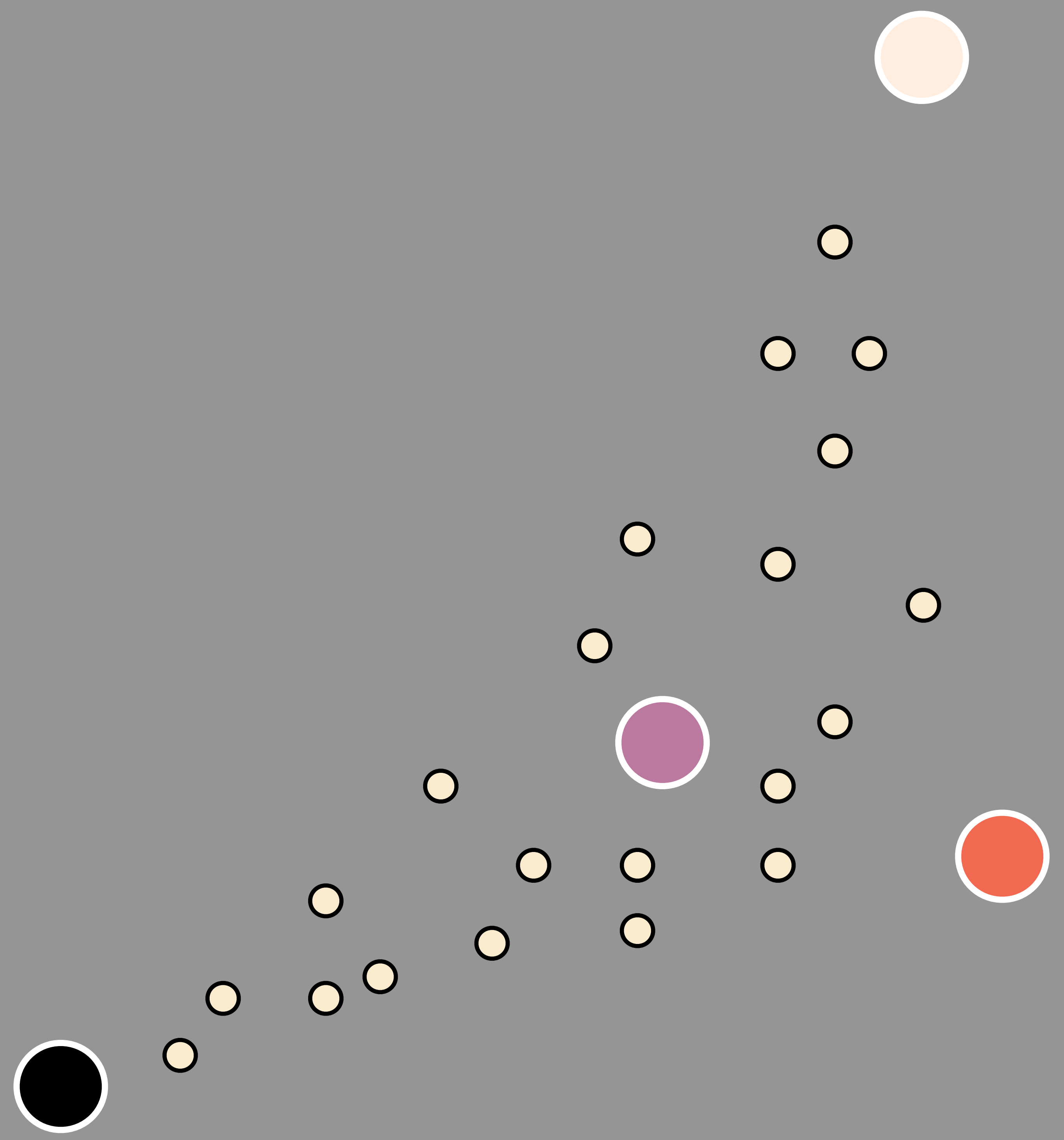
Tessellation in RGB space



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

Tessellation in RGB space

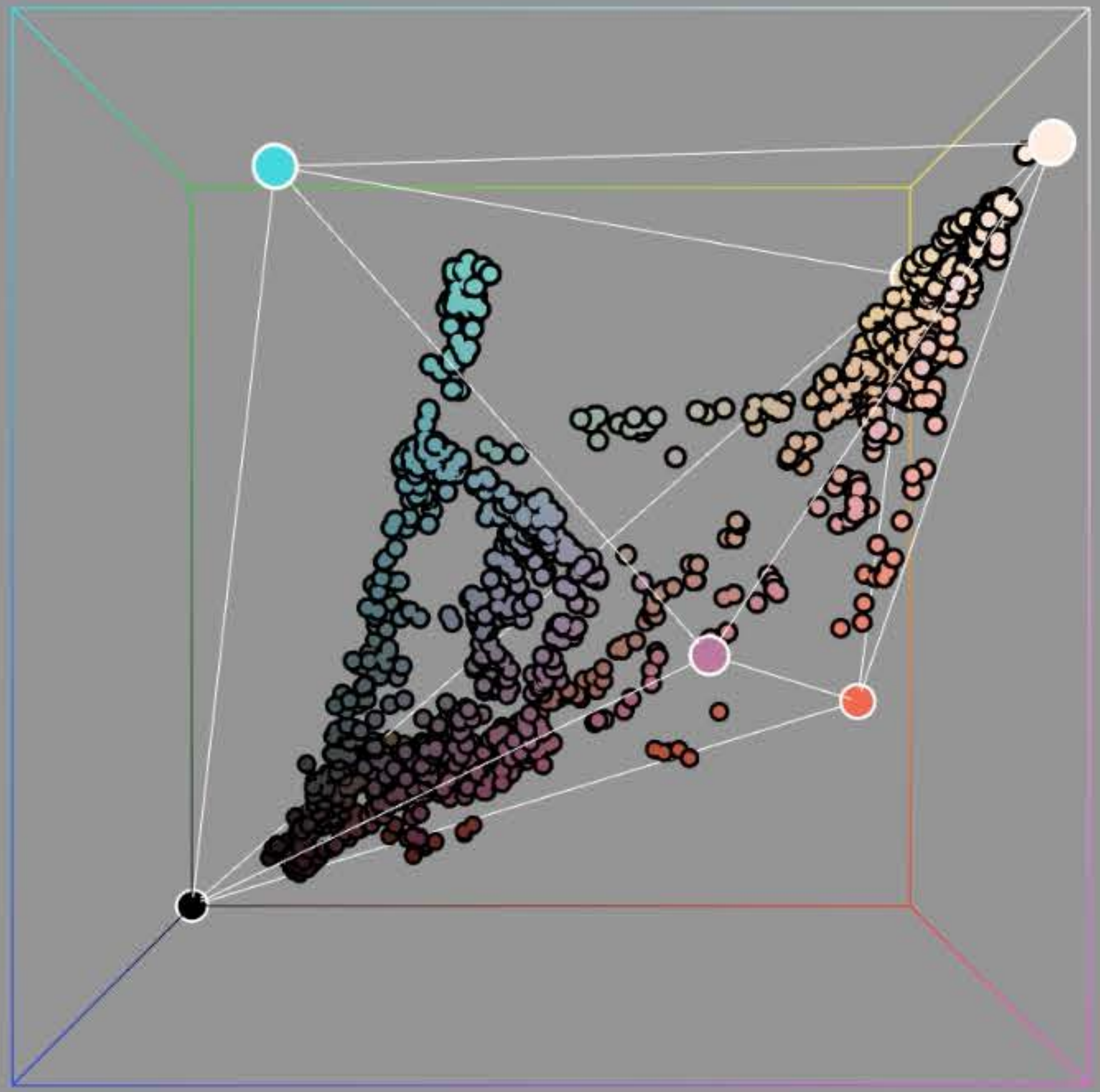


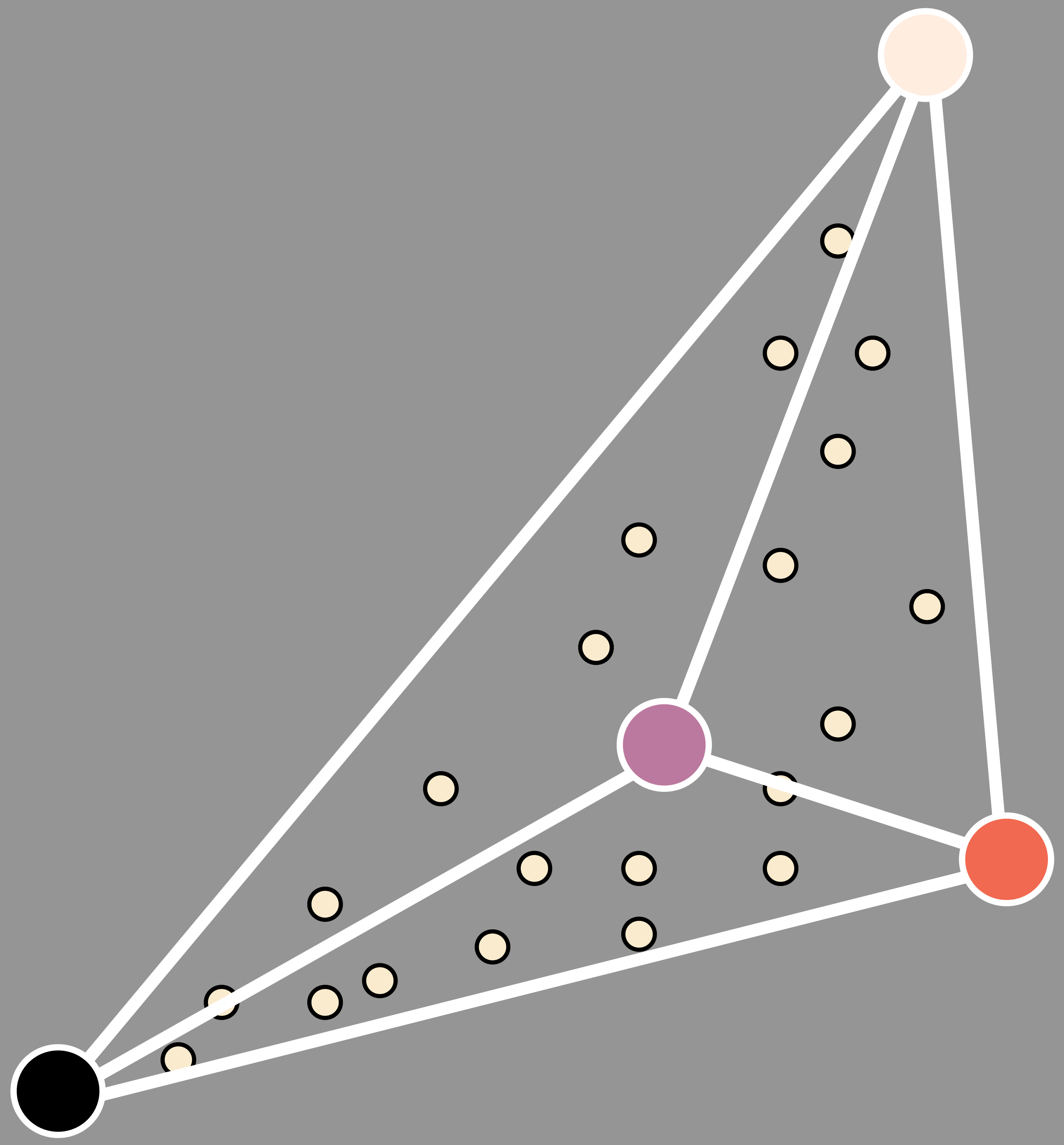


RGB simplex

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

Tessellation in RGB space





RGB simplex

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

Two-level decomposition

RGB palette

W_{RGB}

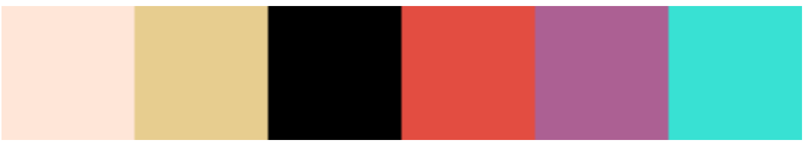
RGBXY vertices (projected to RGB)

W_{RGBXY}

image

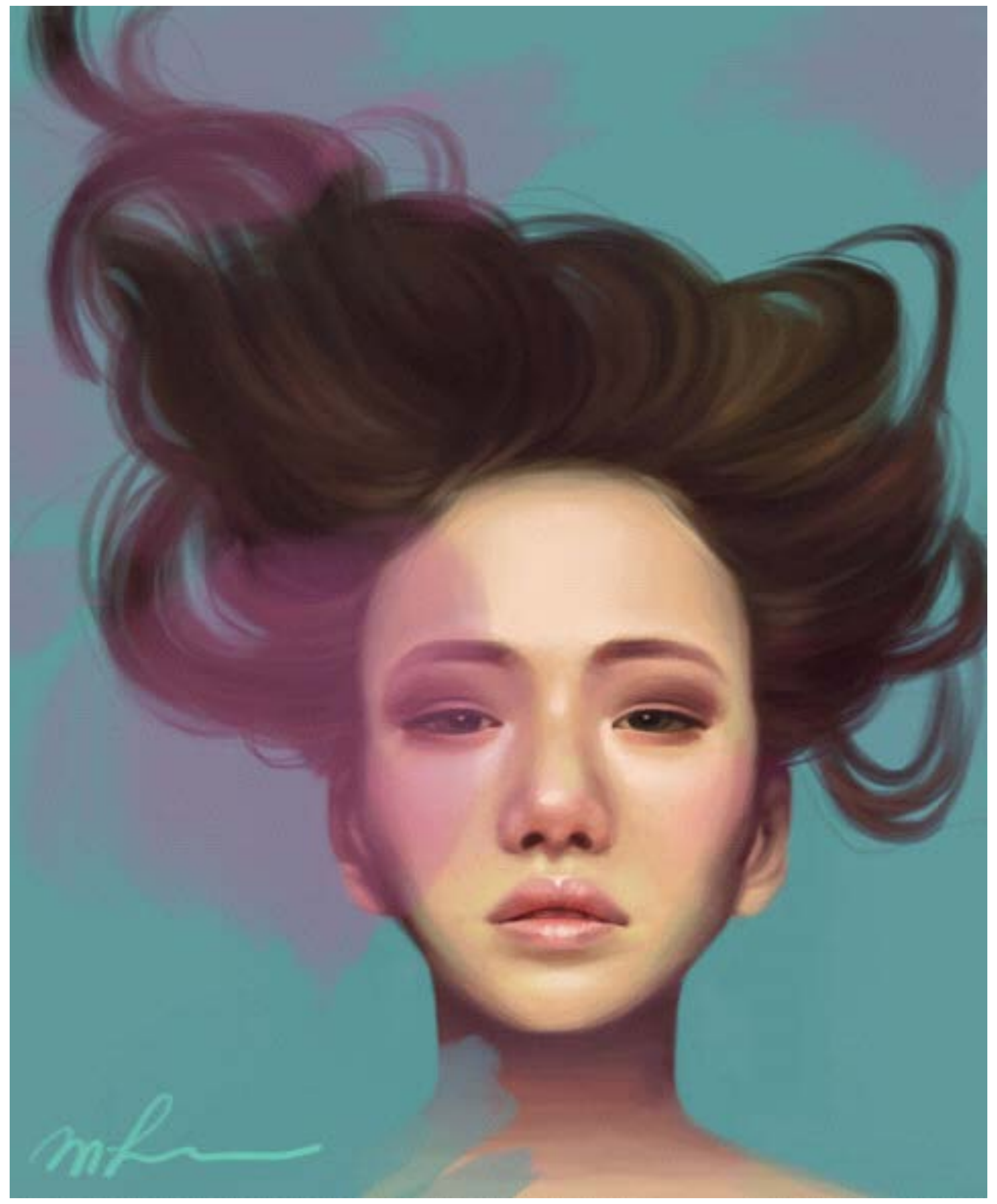
Two-level decomposition

RGB palette

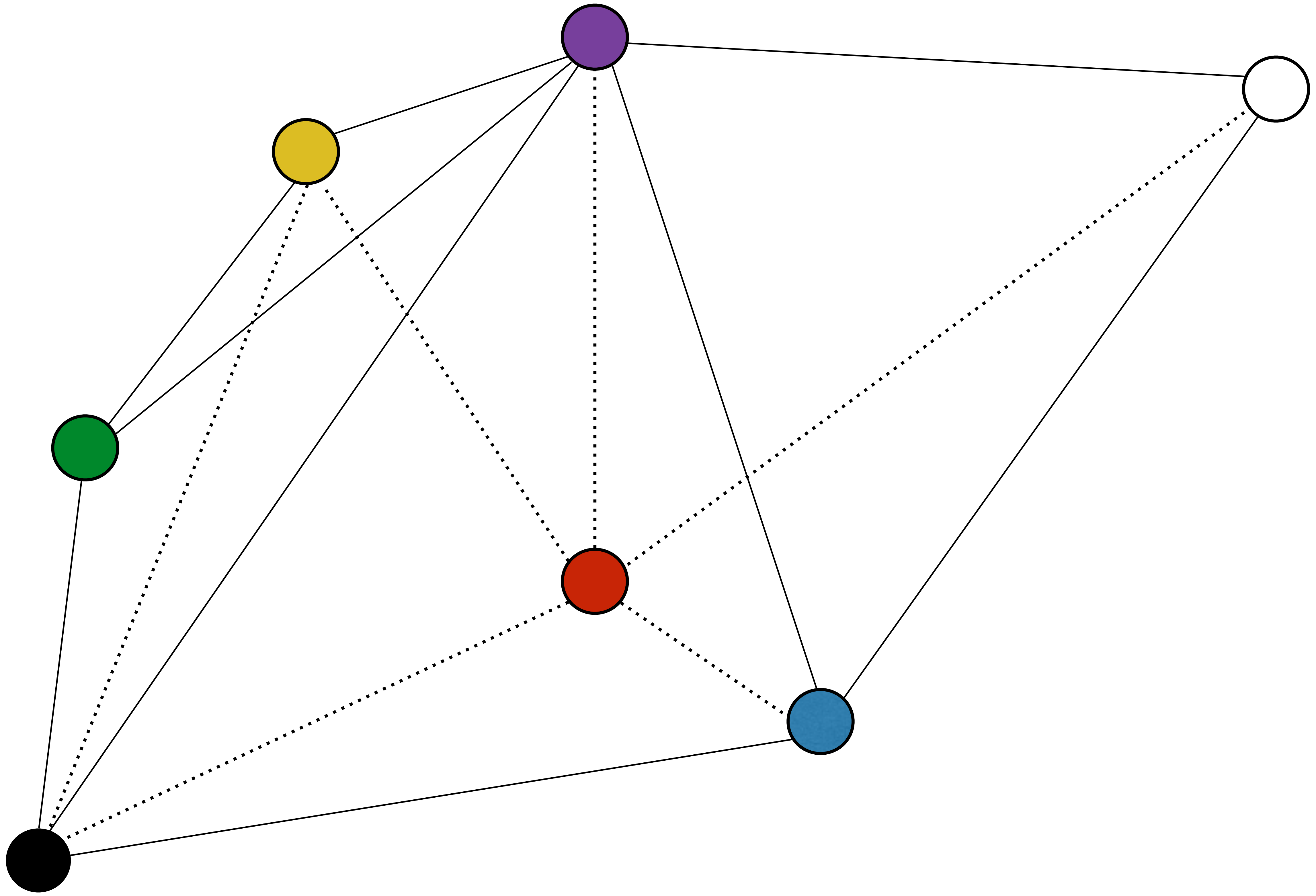


$$W = W_{RGB} * W_{RGBXY}$$

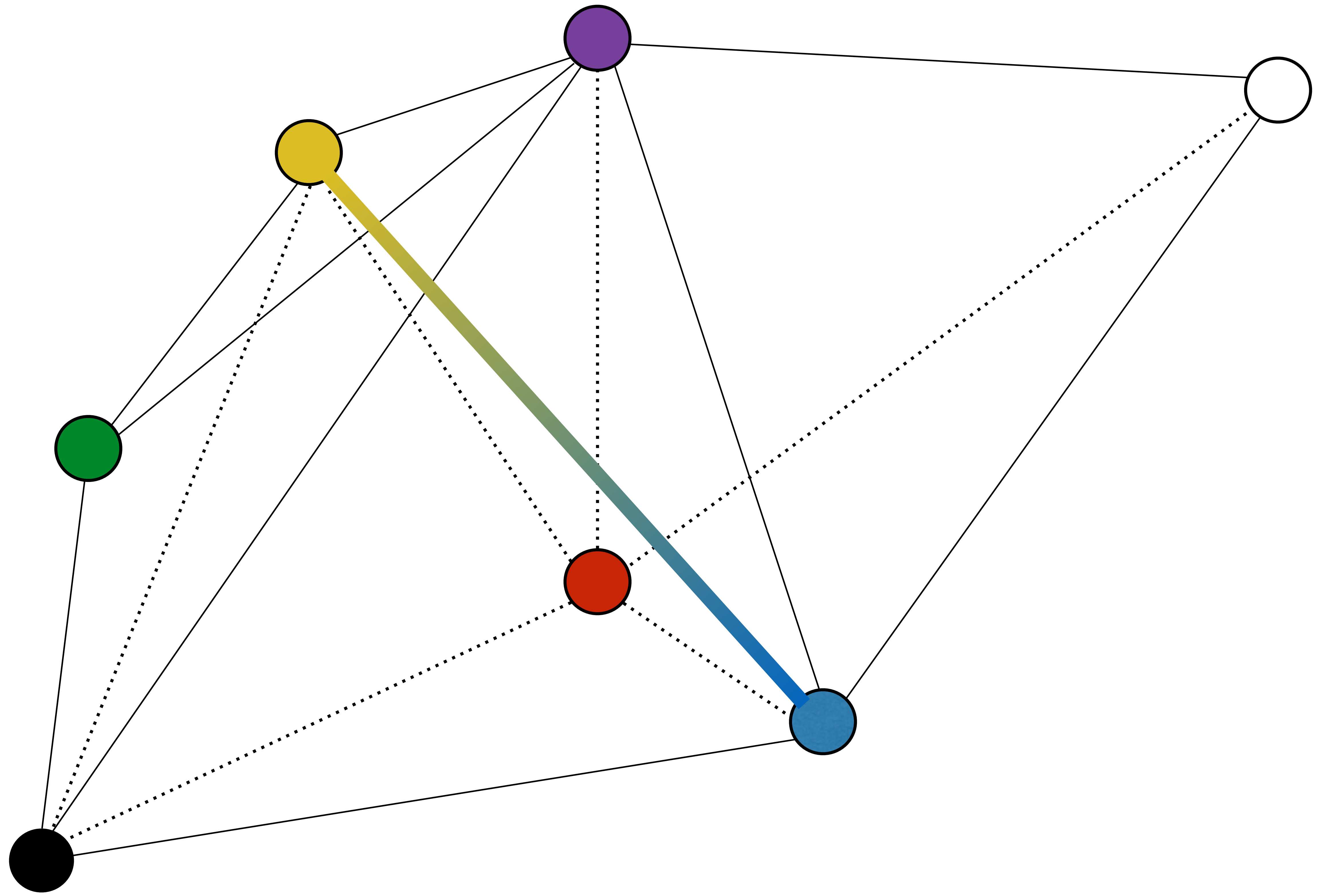
image



Tessellation in RGB space

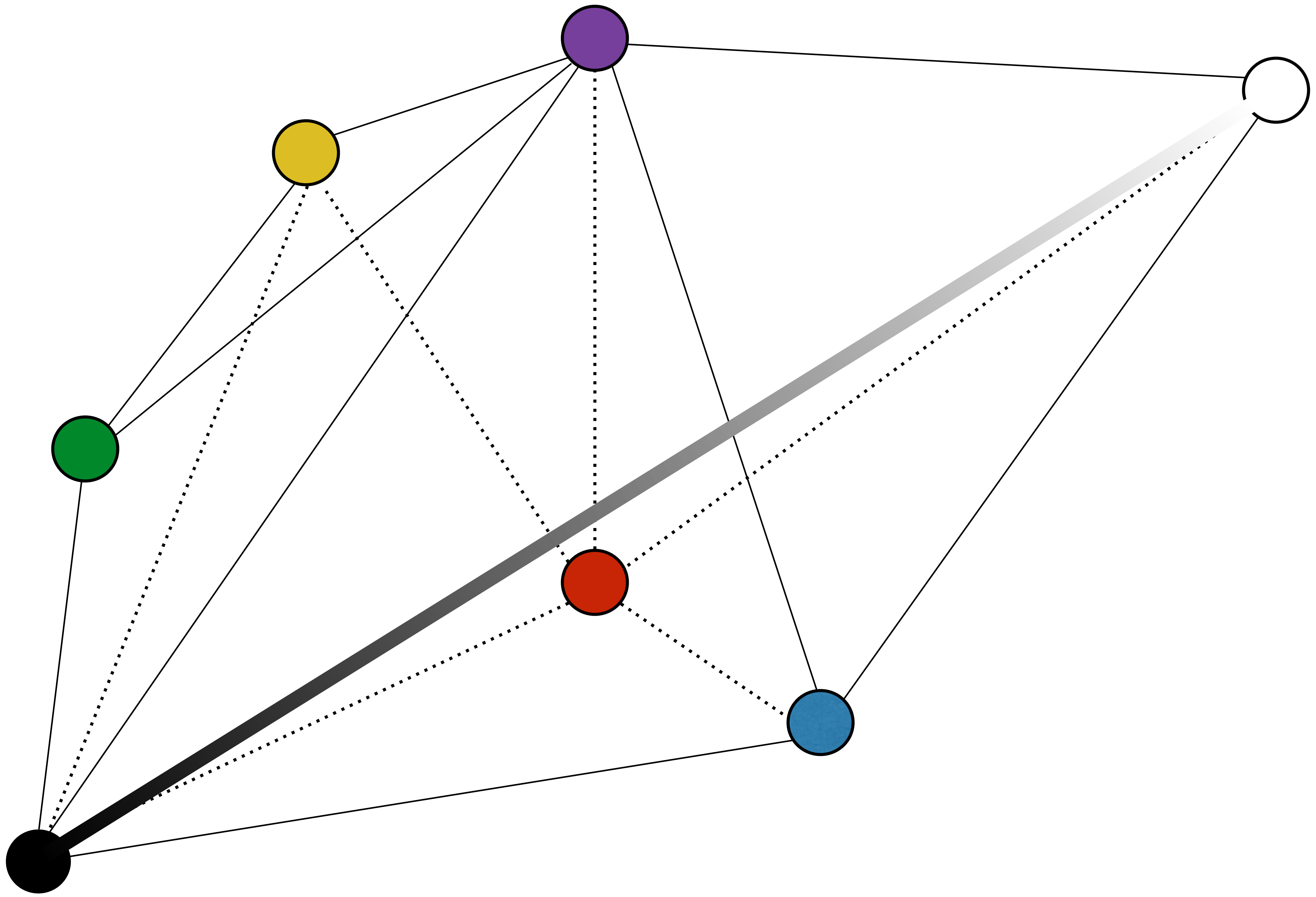


Tessellation in RGB space



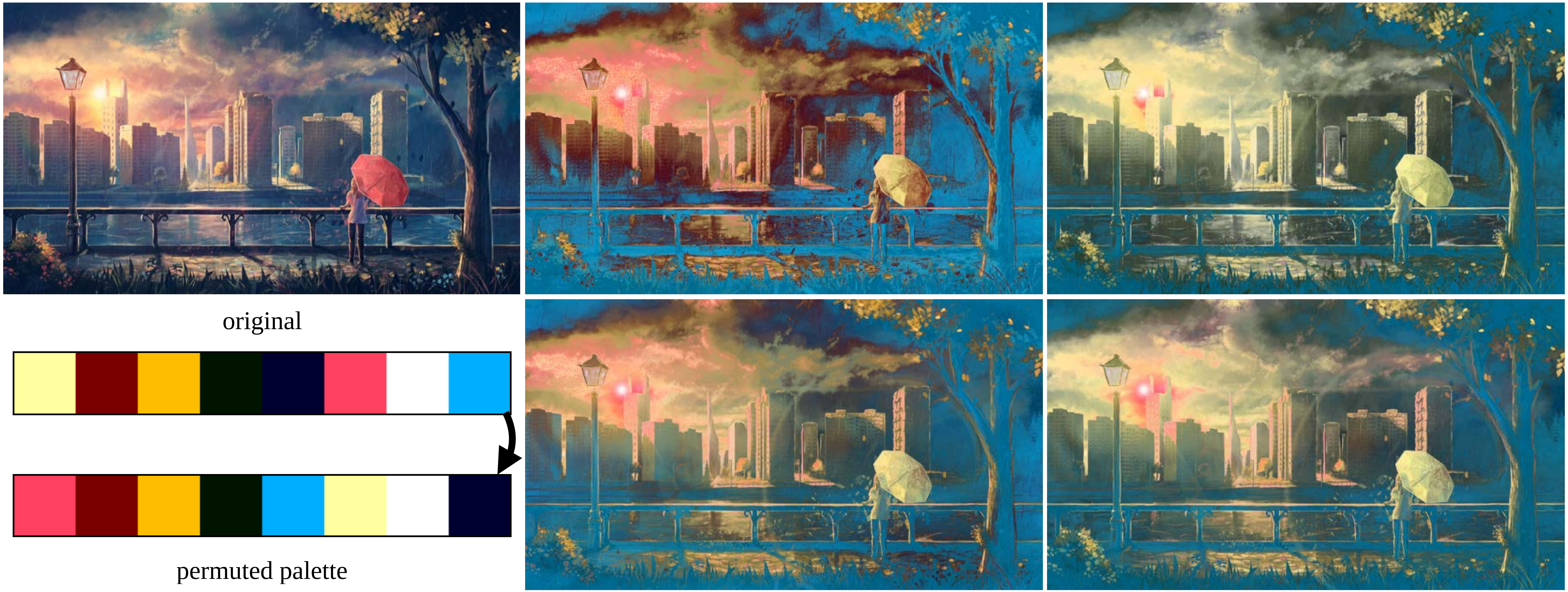
Delaunay tessellation

Tessellation in RGB space



Star tessellation

Four decomposition methods comparisons



RGB-space Delaunay tessellation

RGB-space star tessellation

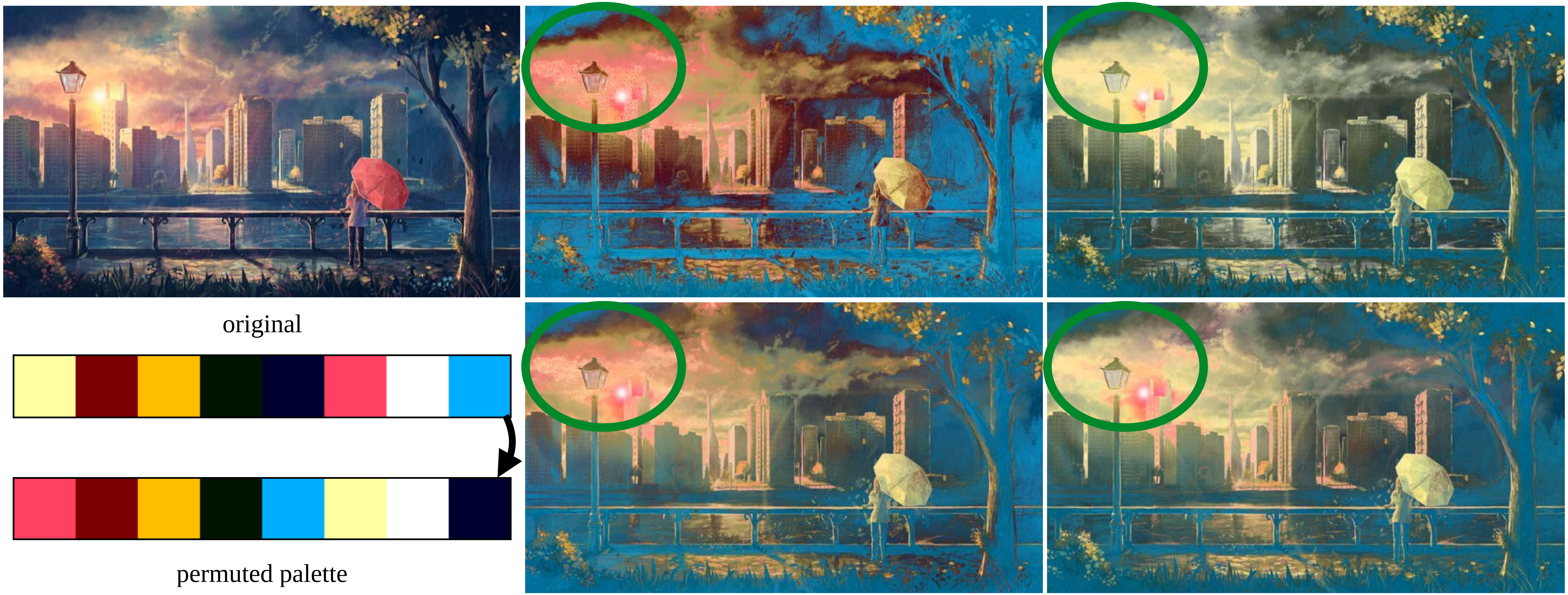
RGB weights

RGB · RGBXY weights

original

permuted palette

Four decomposition methods comparisons



RGB-space Delaunay tessellation

RGB-space star tessellation

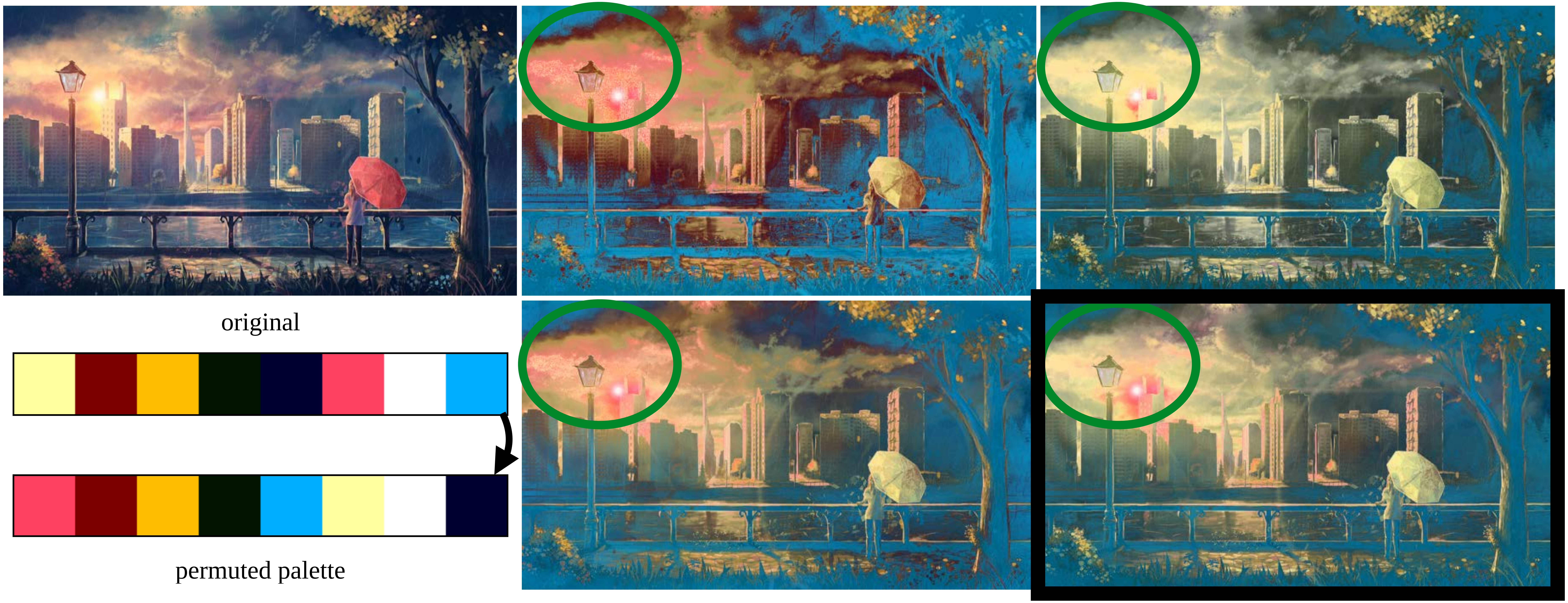
RGB weights

original

RGB · RGBXY weights

permuted palette

Four decomposition methods comparisons



RGB-space Delaunay tessellation

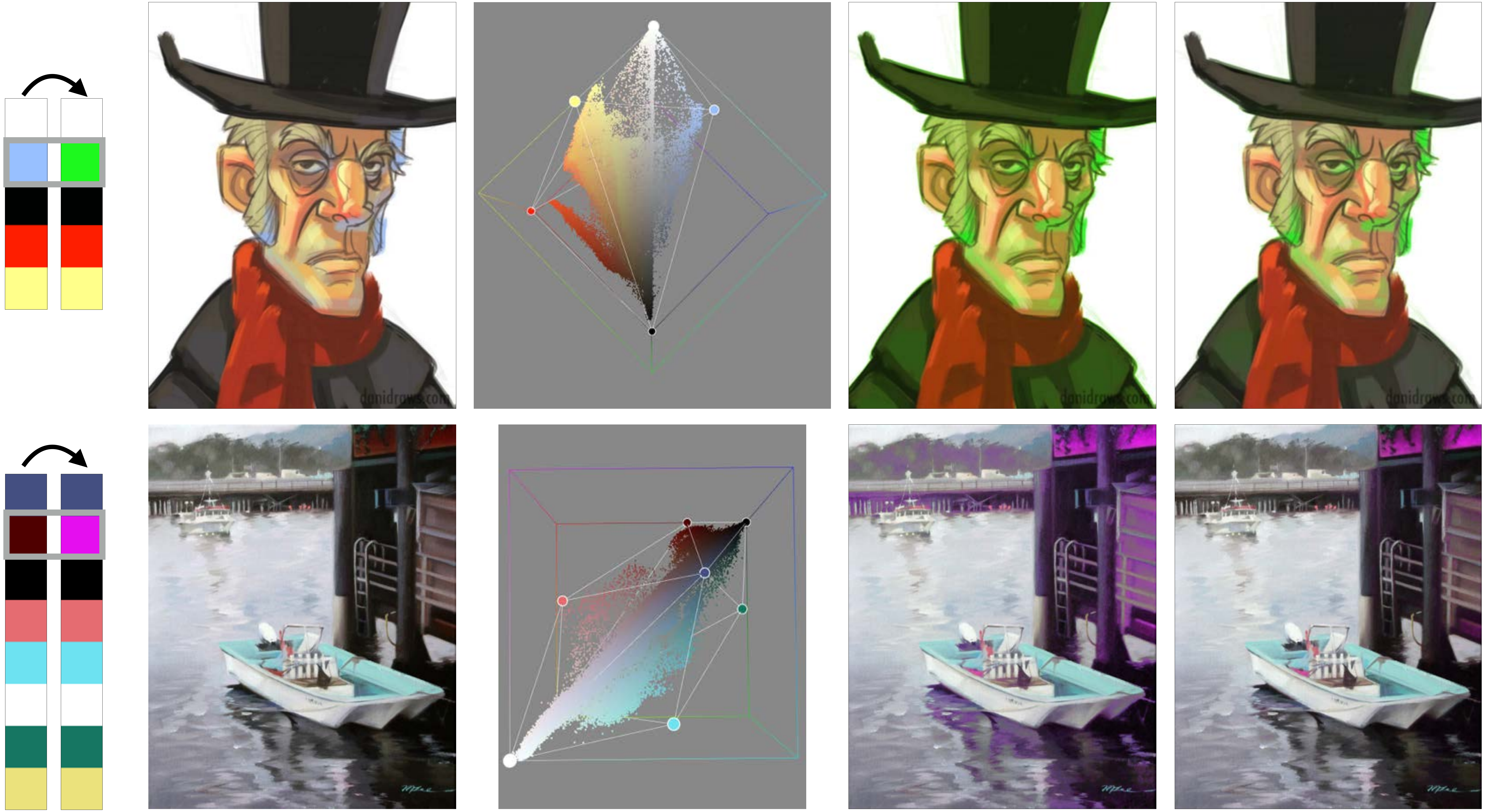
RGB-space star tessellation

RGB weights

RGB · RGBXY weights

original

permuted palette

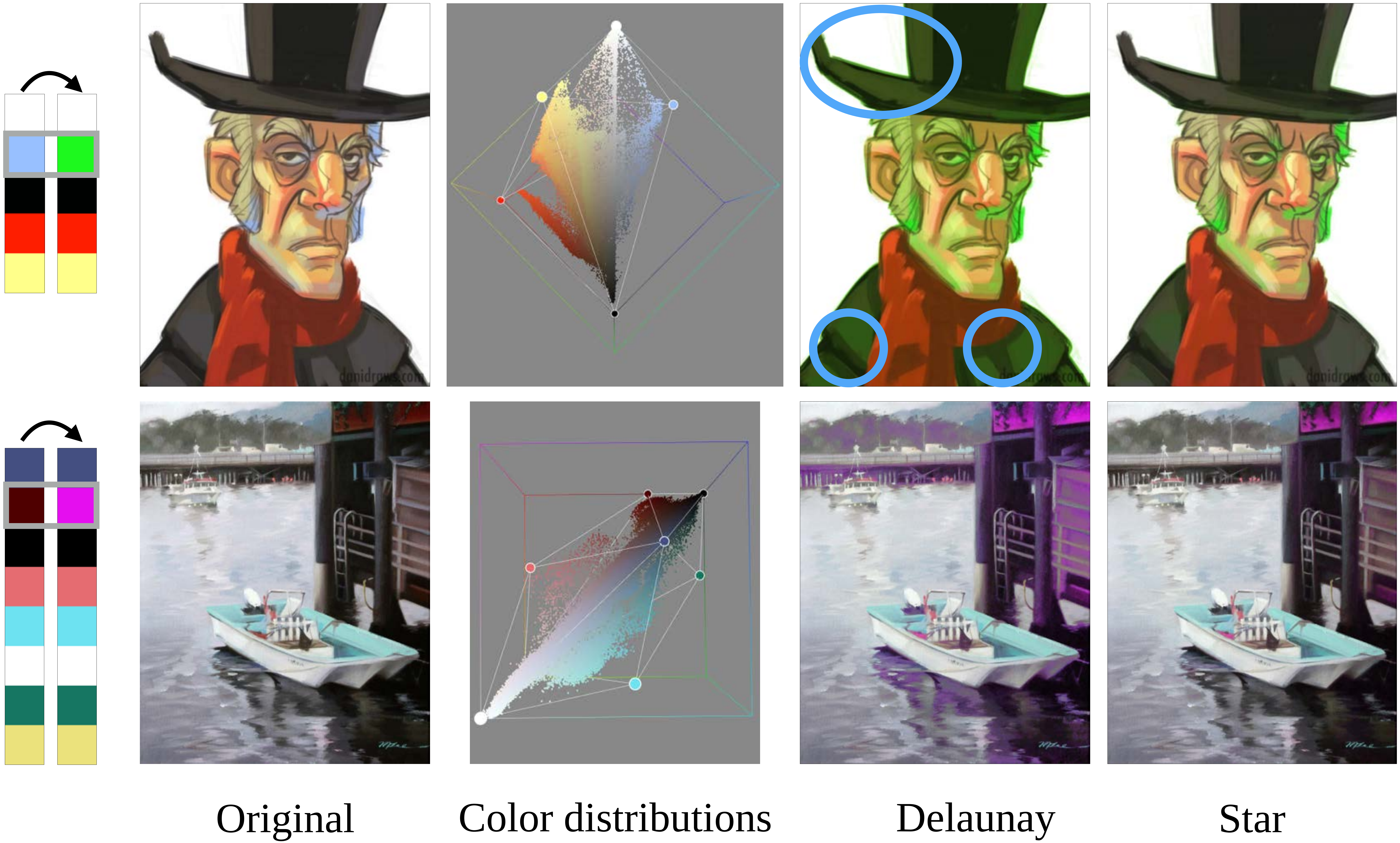


Original

Color distributions

Delaunay

Star

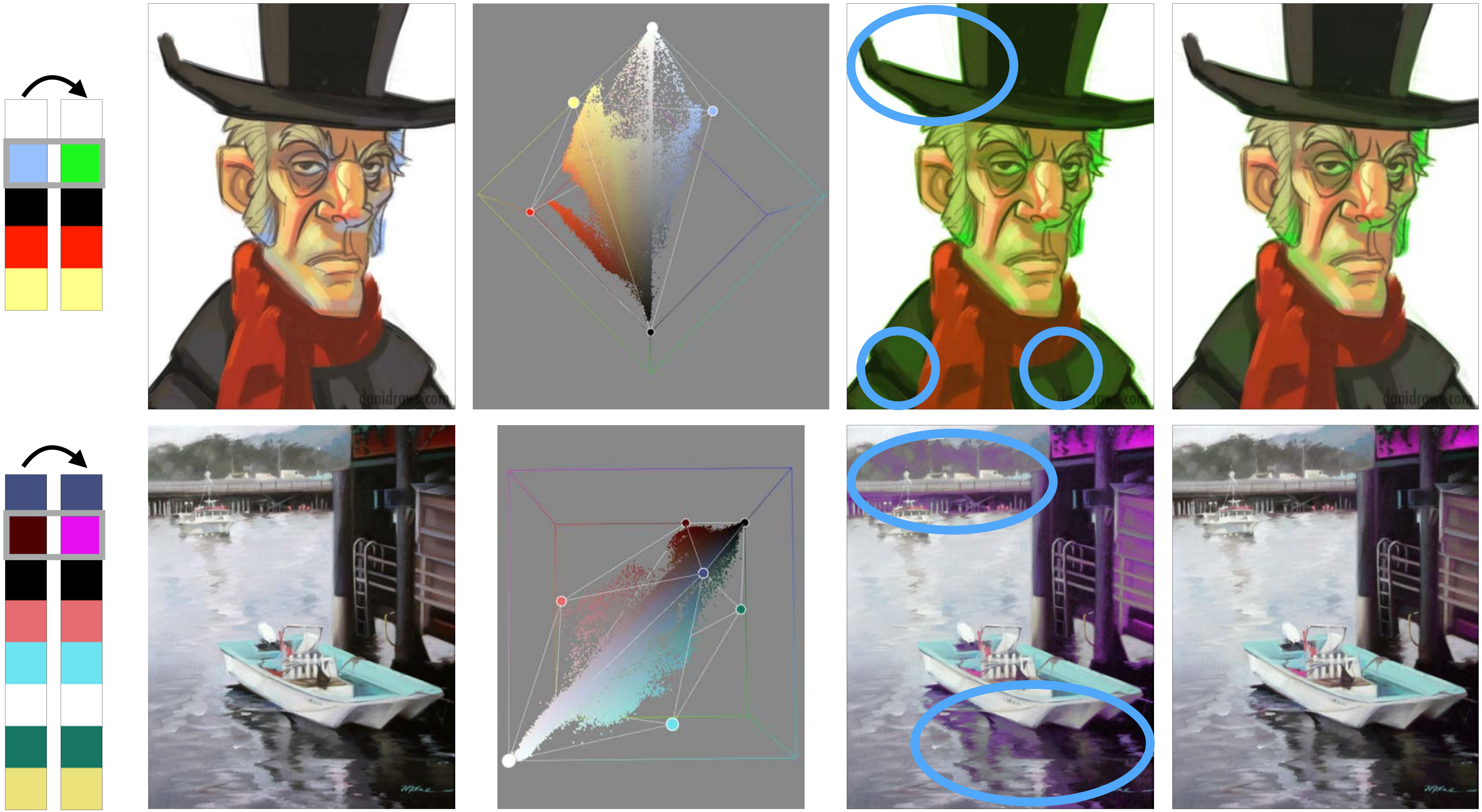


Original

Color distributions

Delaunay

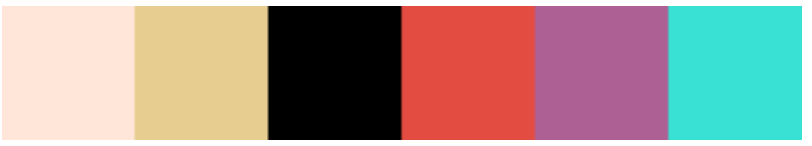
Star



Original Color distributions Delaunay Star

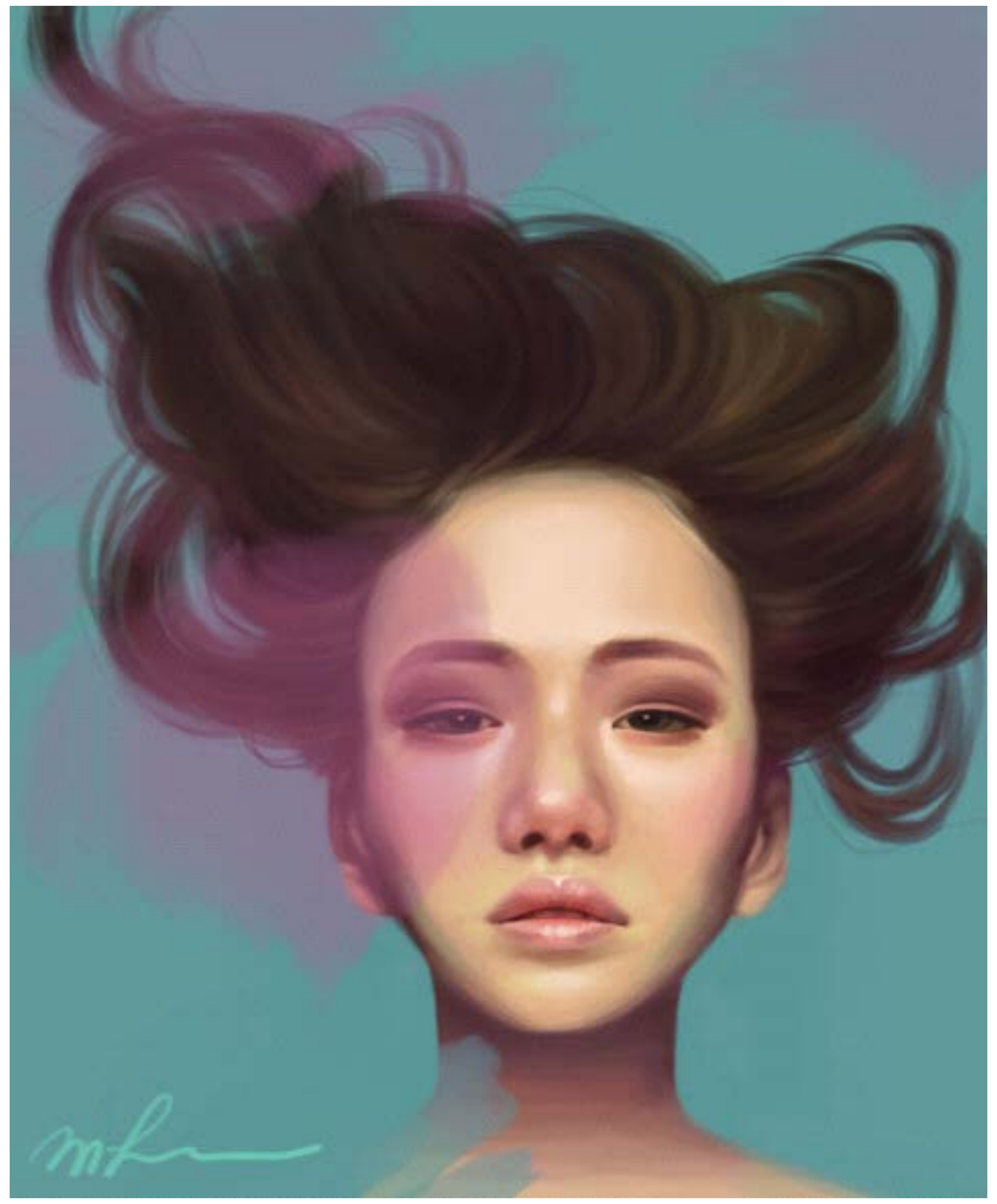
Two-level decomposition

RGB palette



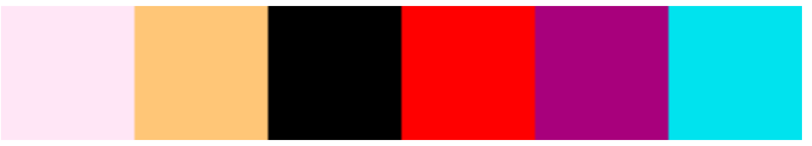
$$\xrightarrow{W = W_{RGB} * \overset{\text{Fixed}}{W_{RGBXY}}}$$

image



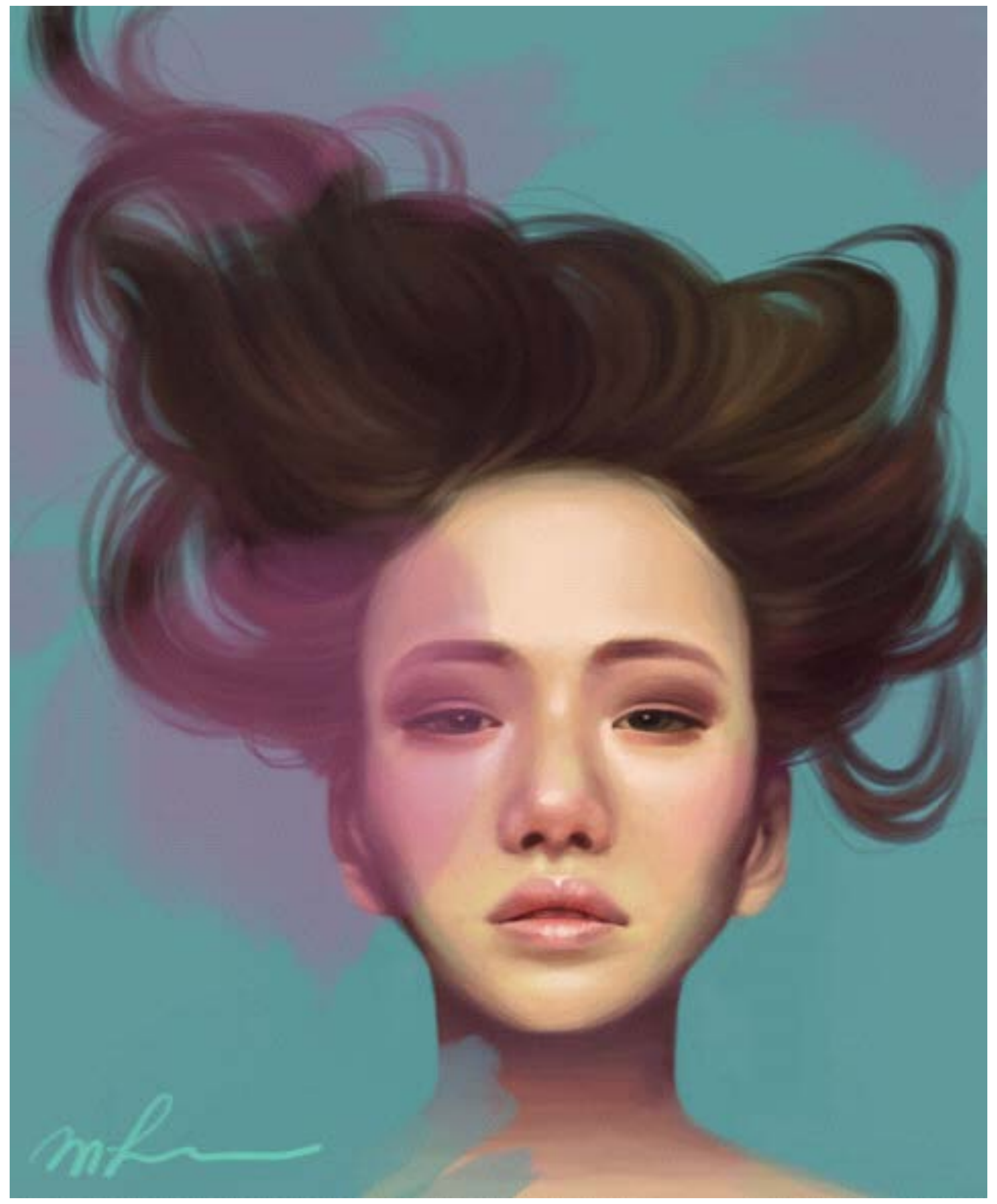
Two-level decomposition

RGB palette



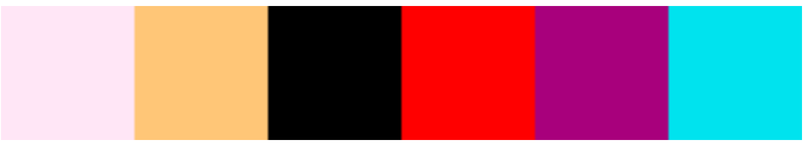
$$W = \overset{\text{Palette updates}}{W_{RGB}} * \overset{\text{Fixed}}{W_{RGBXY}}$$

image



Two-level decomposition

RGB palette

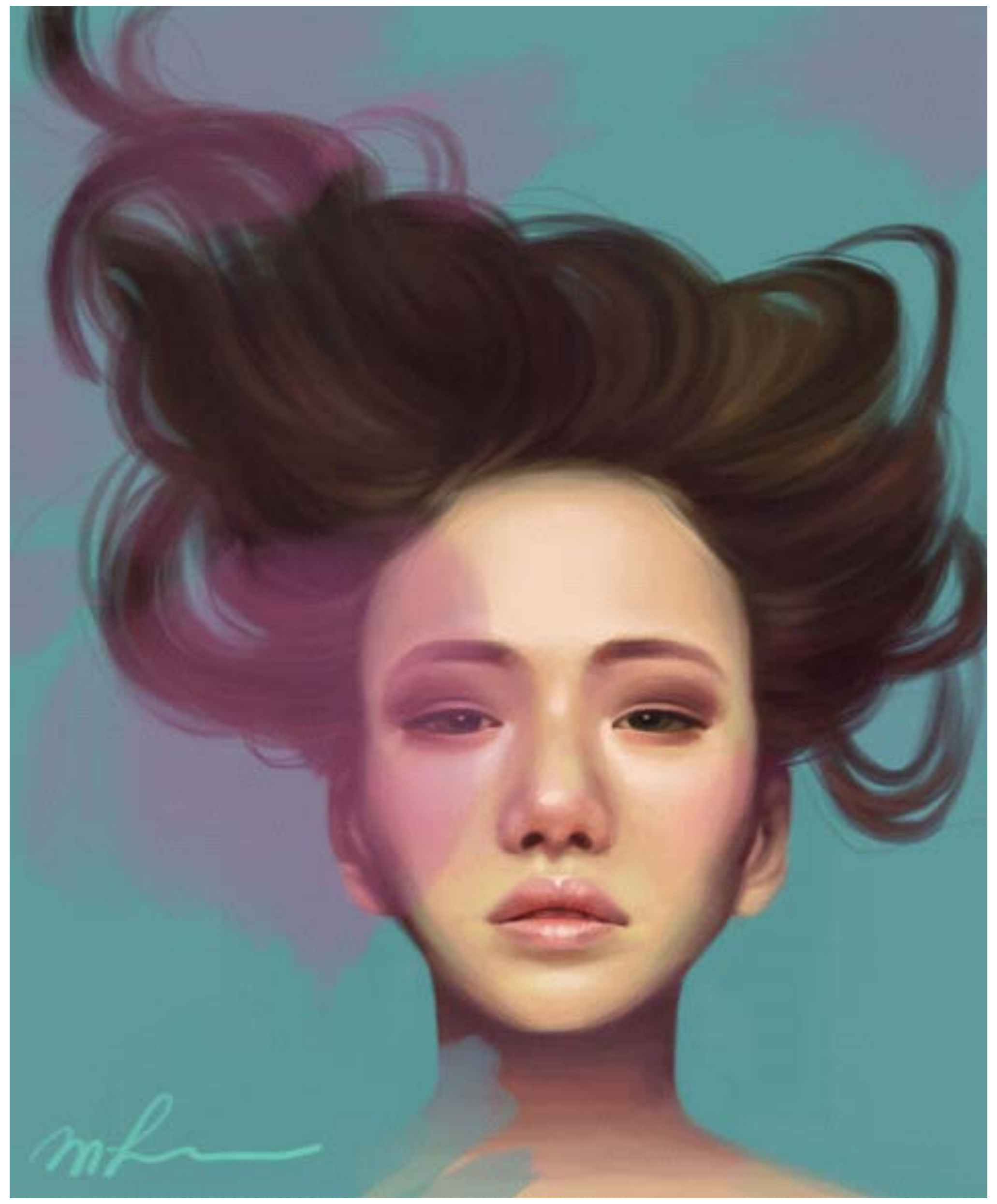


Palette updates

Fixed

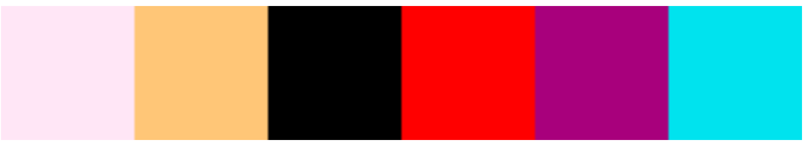
$$W = W_{RGB} * W_{RGBXY}$$

image



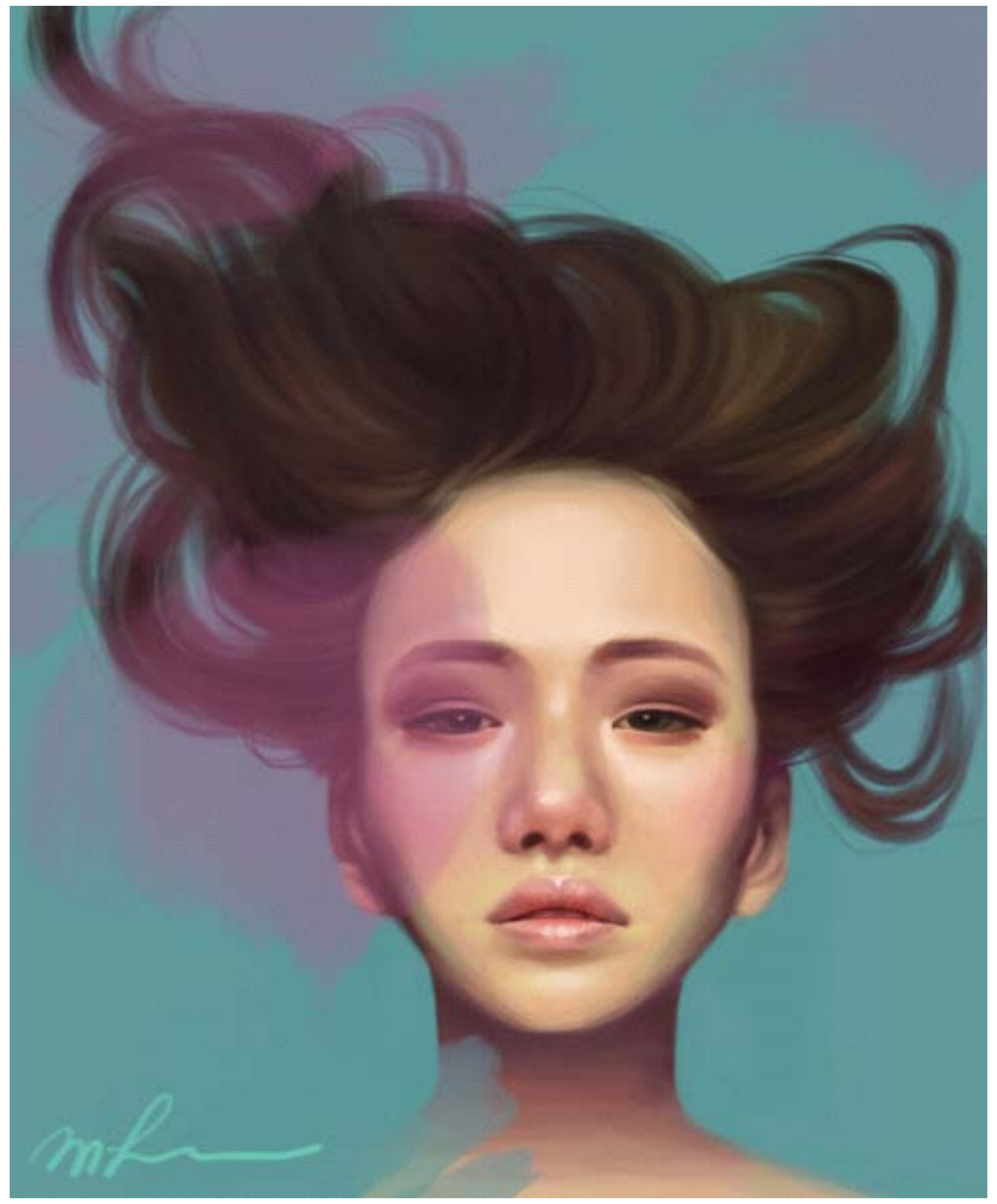
Two-level decomposition

RGB palette



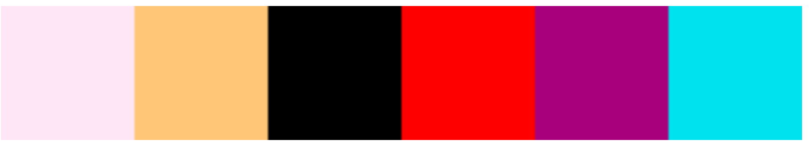
$$\xrightarrow{W = \overset{\text{Palette updates}}{W_{RGB}} * \overset{\text{Fixed}}{W_{RGBXY}}}$$

image



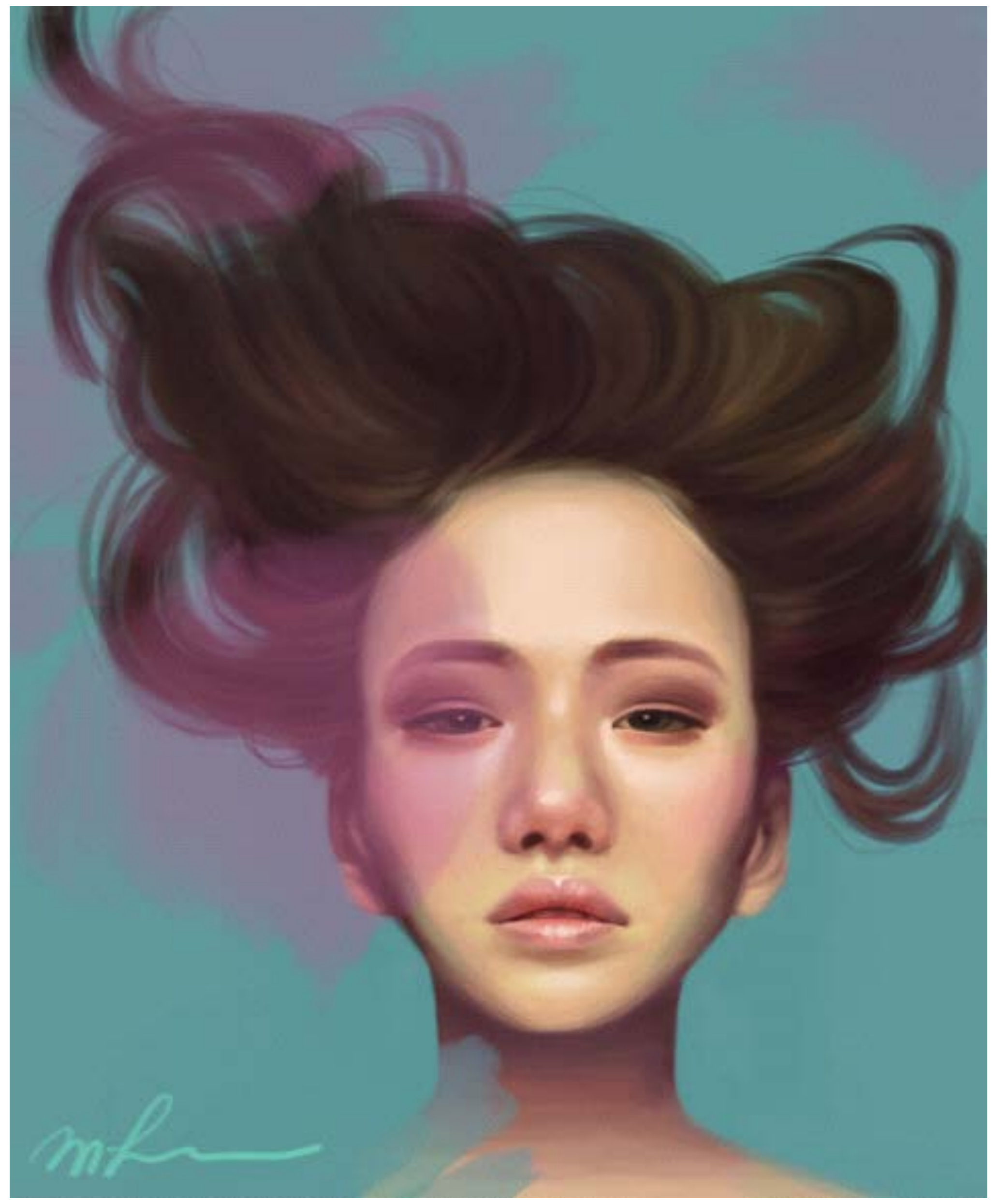
Two-level decomposition

RGB palette



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

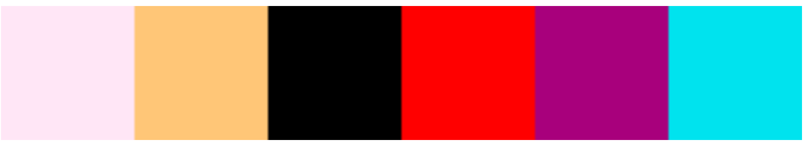
image



Updating W_{RGB} is independent of image size.

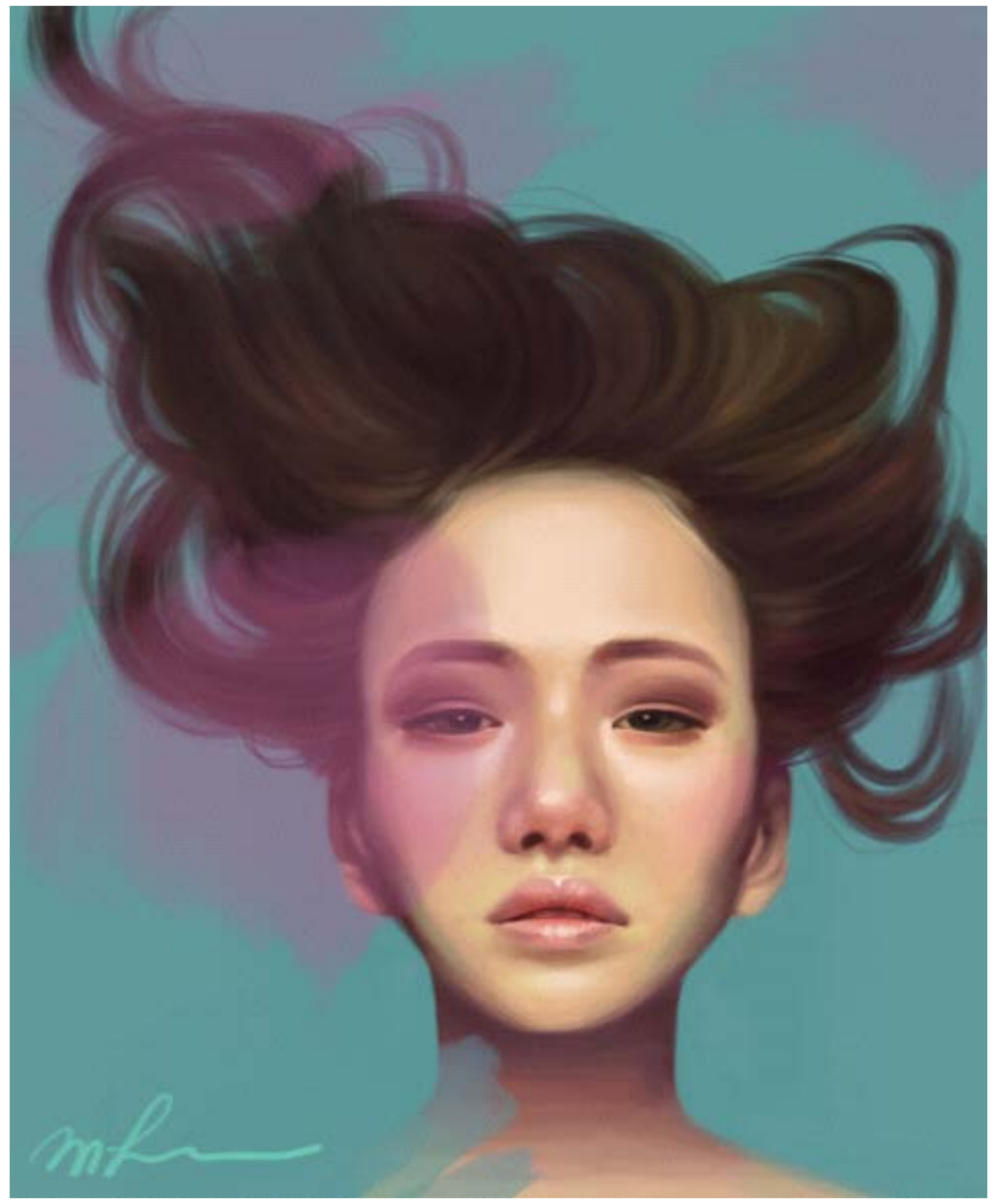
Two-level decomposition

RGB palette



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

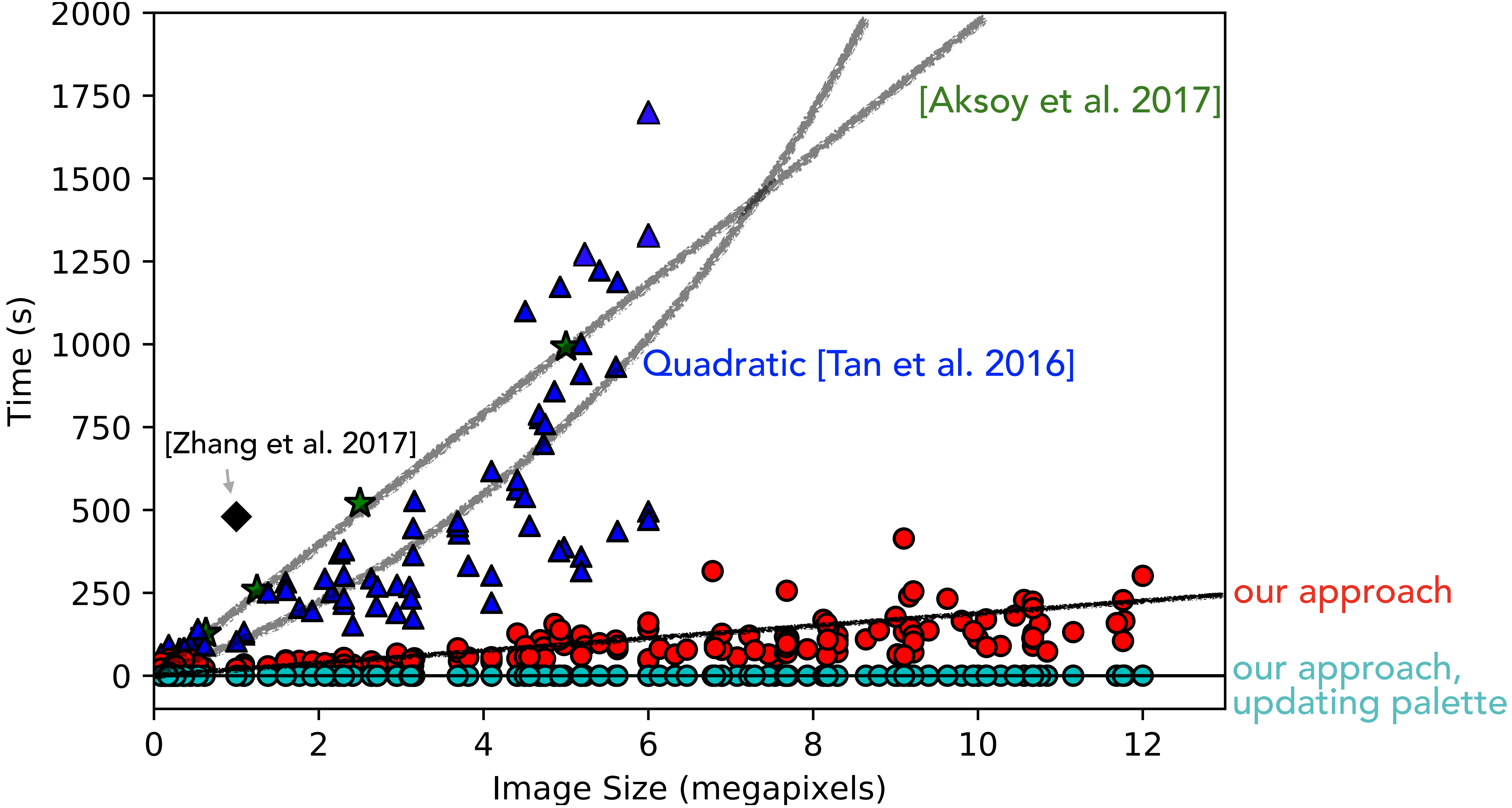
image



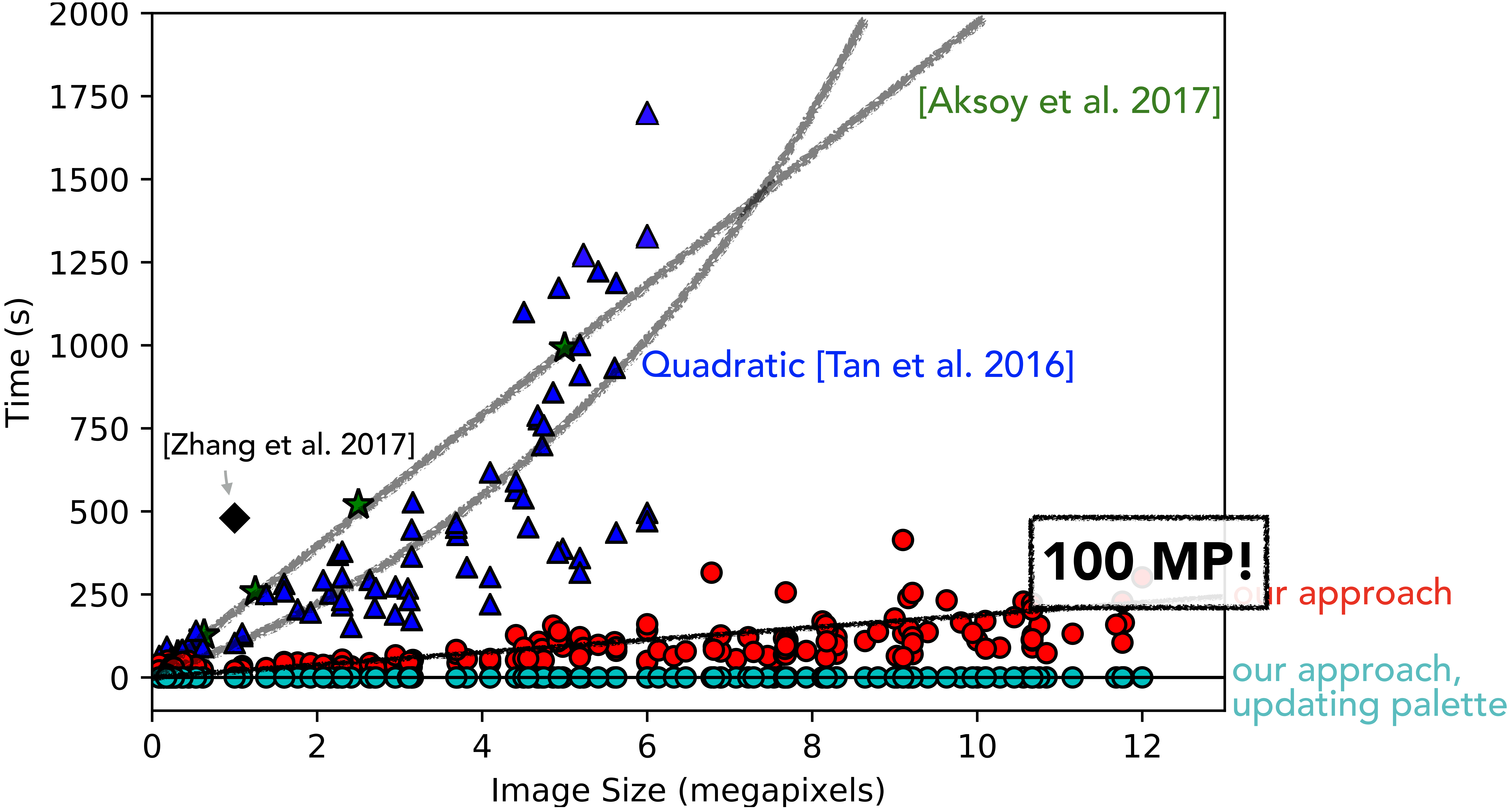
Updating W_{RGB} is independent of image size.

Other methods need to re-compute everything from scratch.

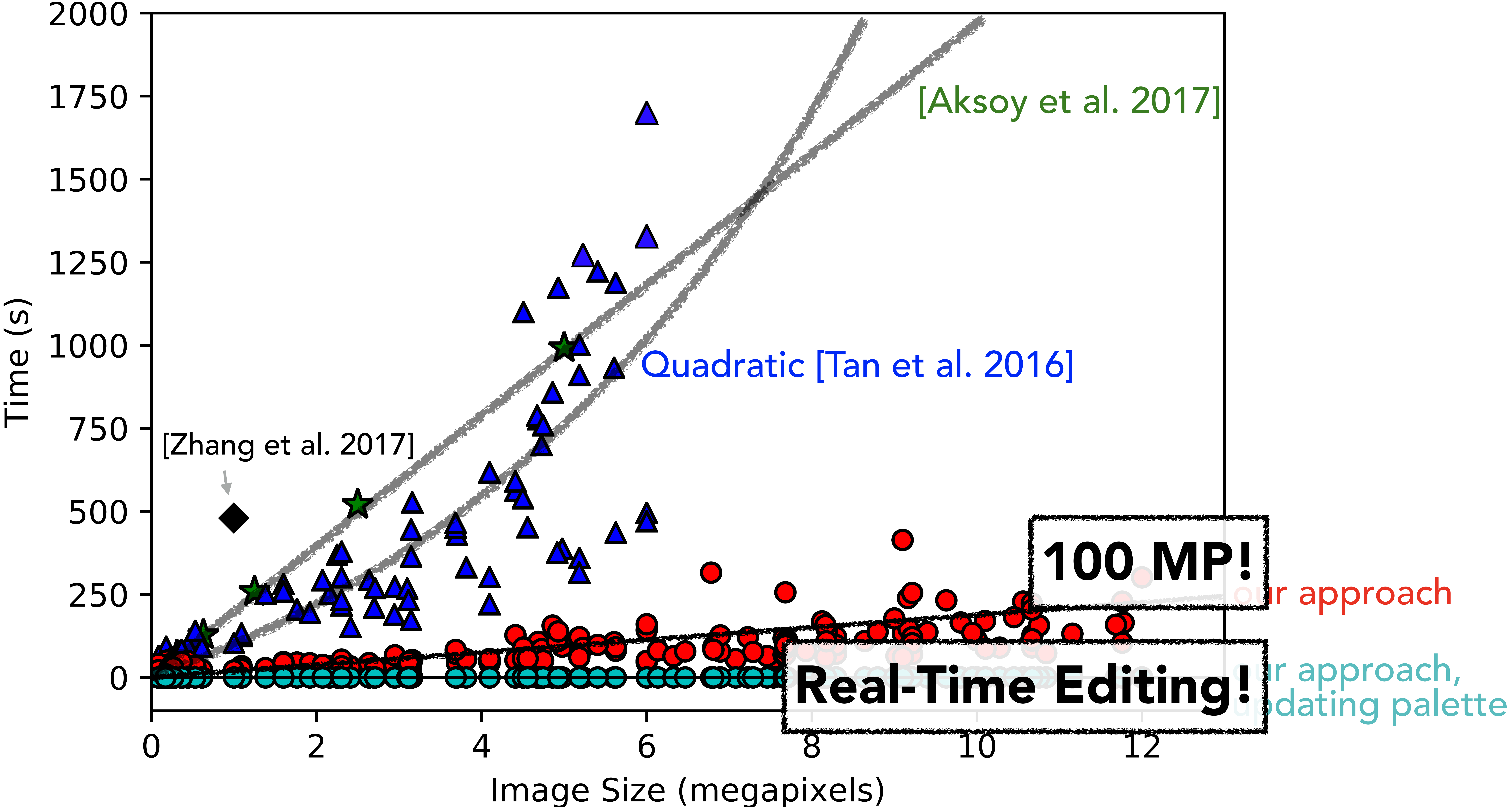
Performance



Performance



Performance



100 MP!

Real-Time Editing!

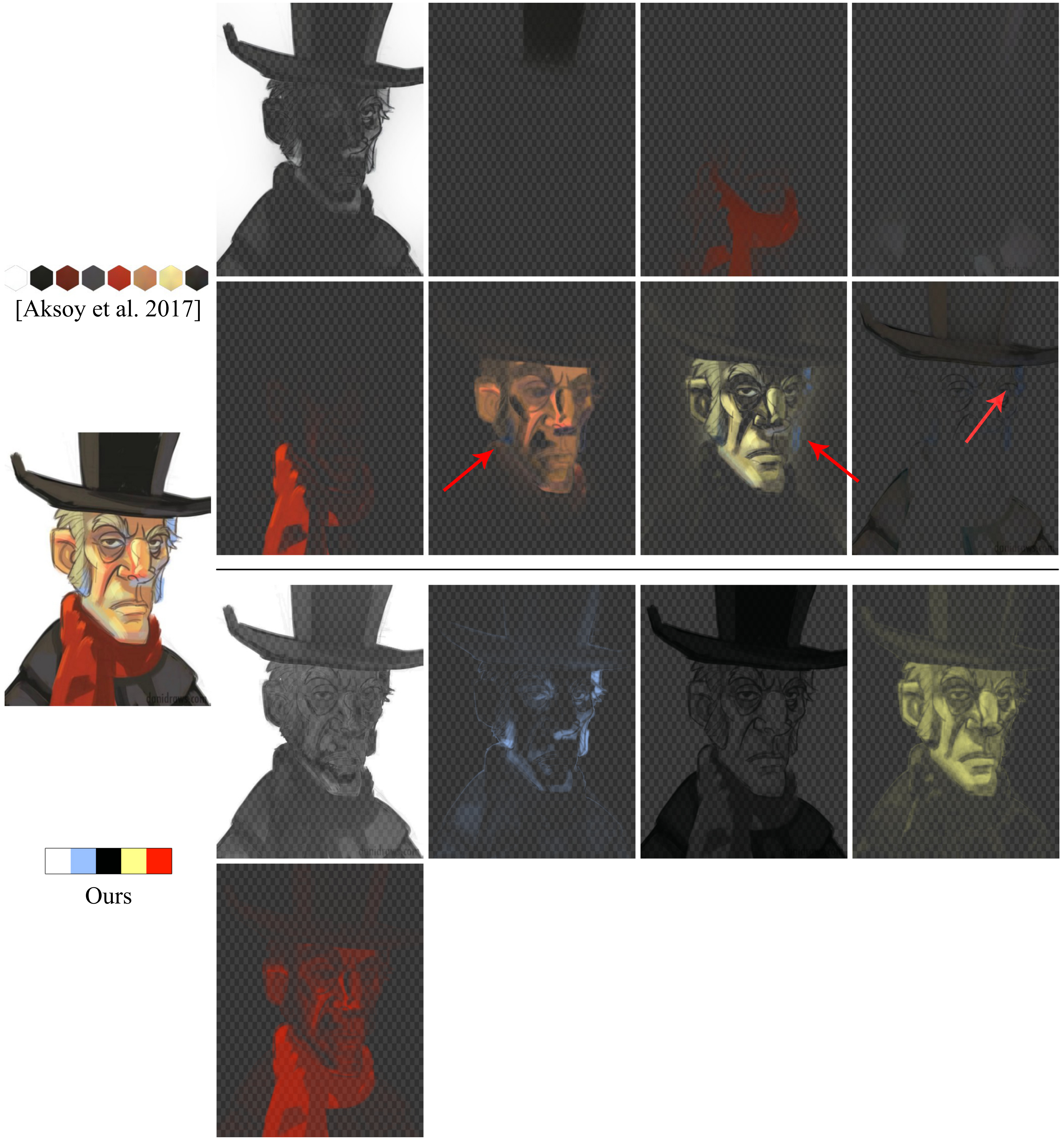
Python Implementation

48 lines of code

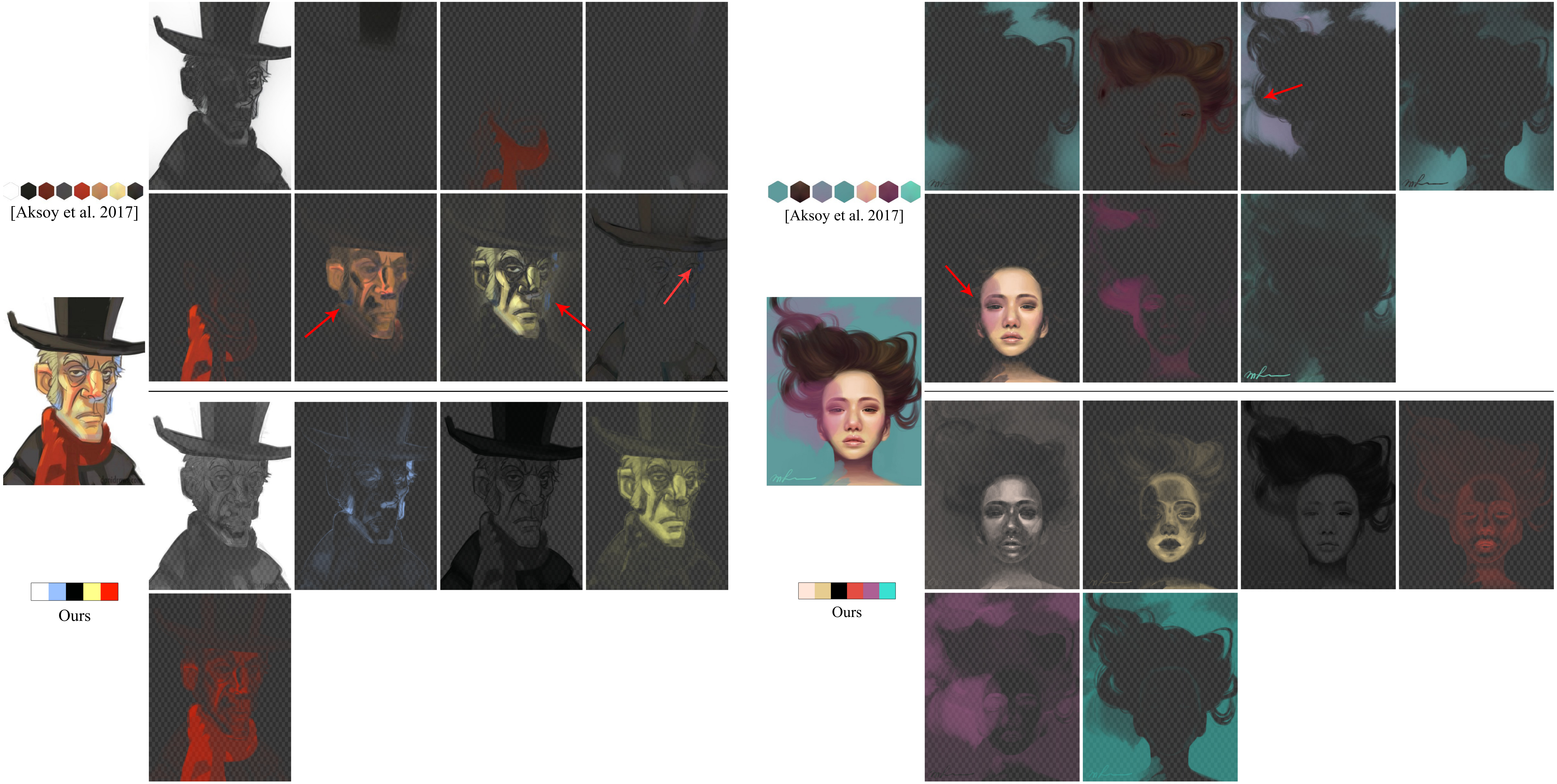
```python
from numpy import *
from scipy.spatial import ConvexHull, Delaunay
from scipy.sparse import coo_matrix

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()
```

Comparisons

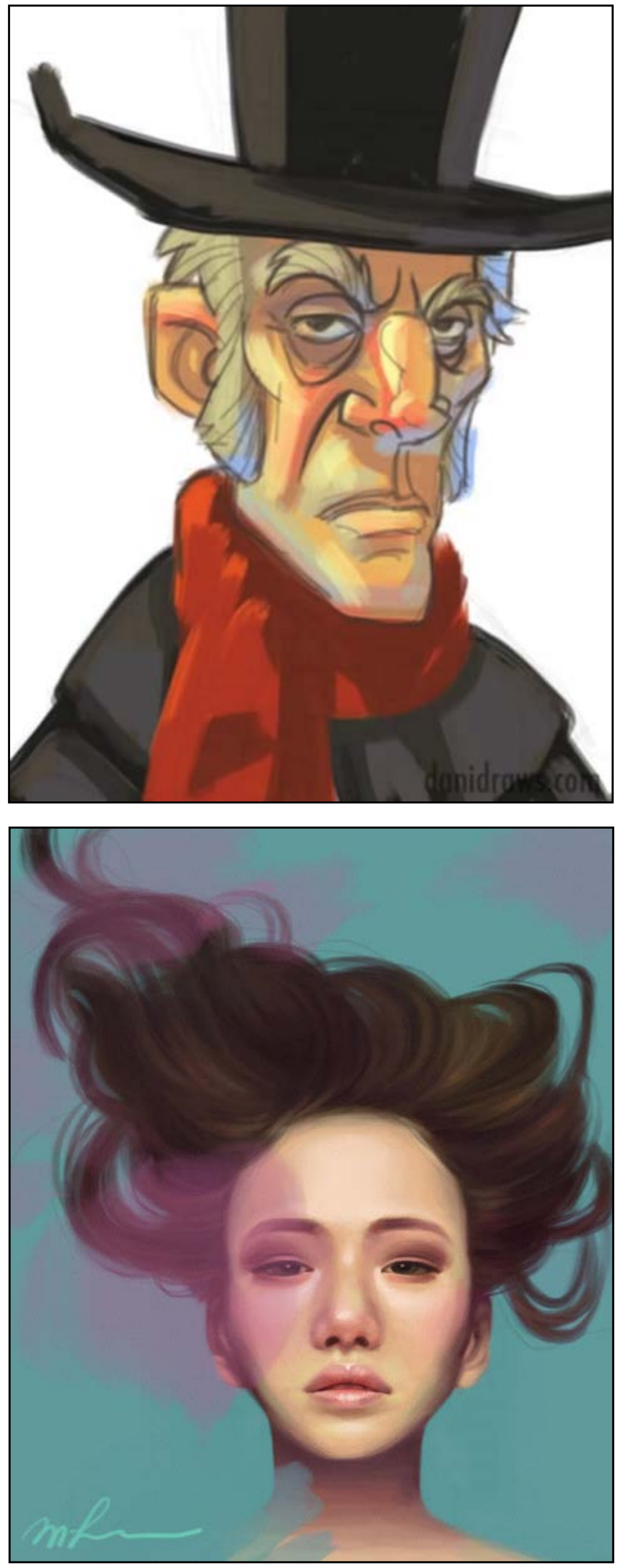
Layer quality comparison with [Aksoy et al. 2017]



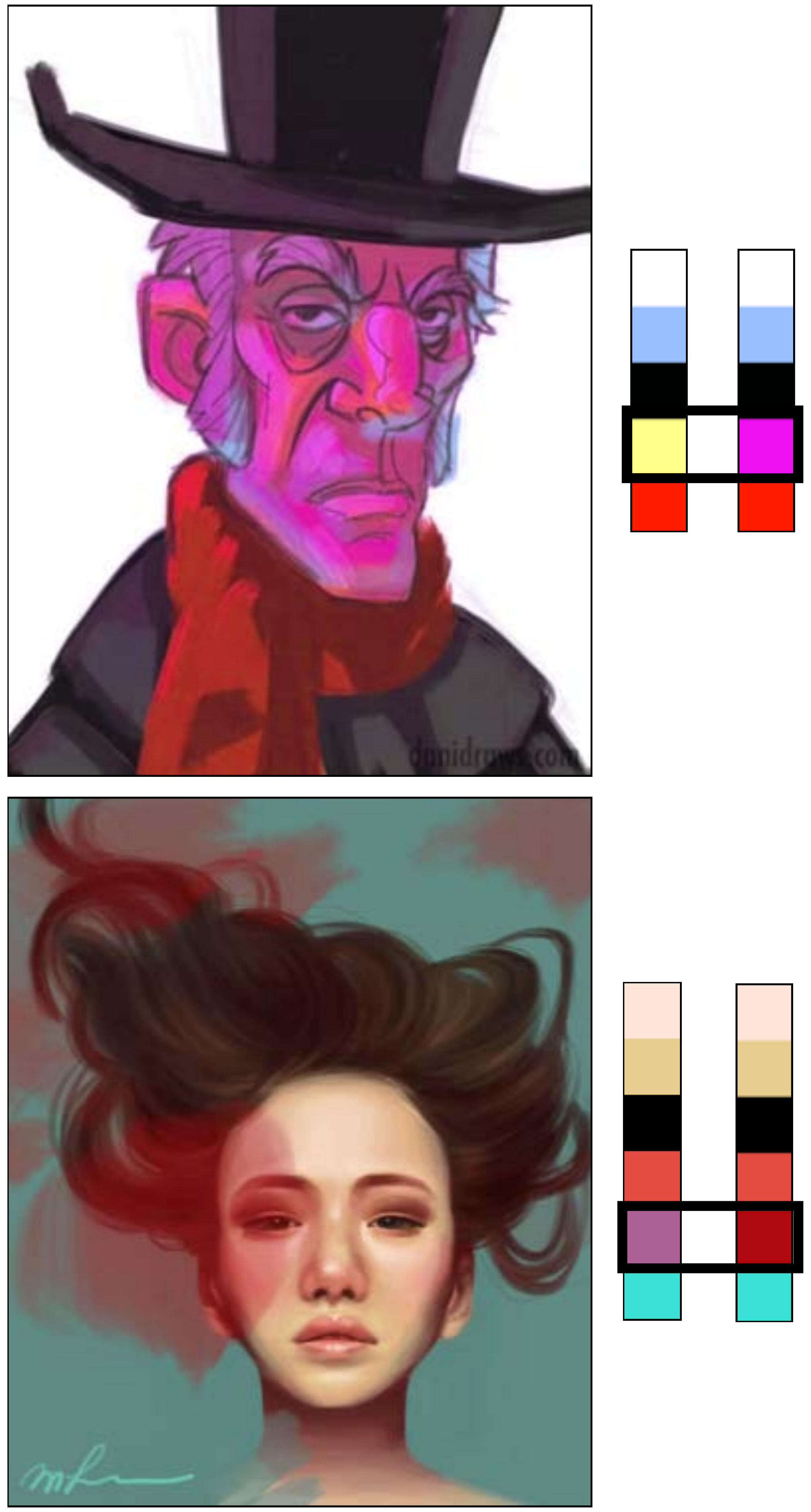
Layer quality comparison with [Aksoy et al. 2017]



Recoloring comparison with three previous methods

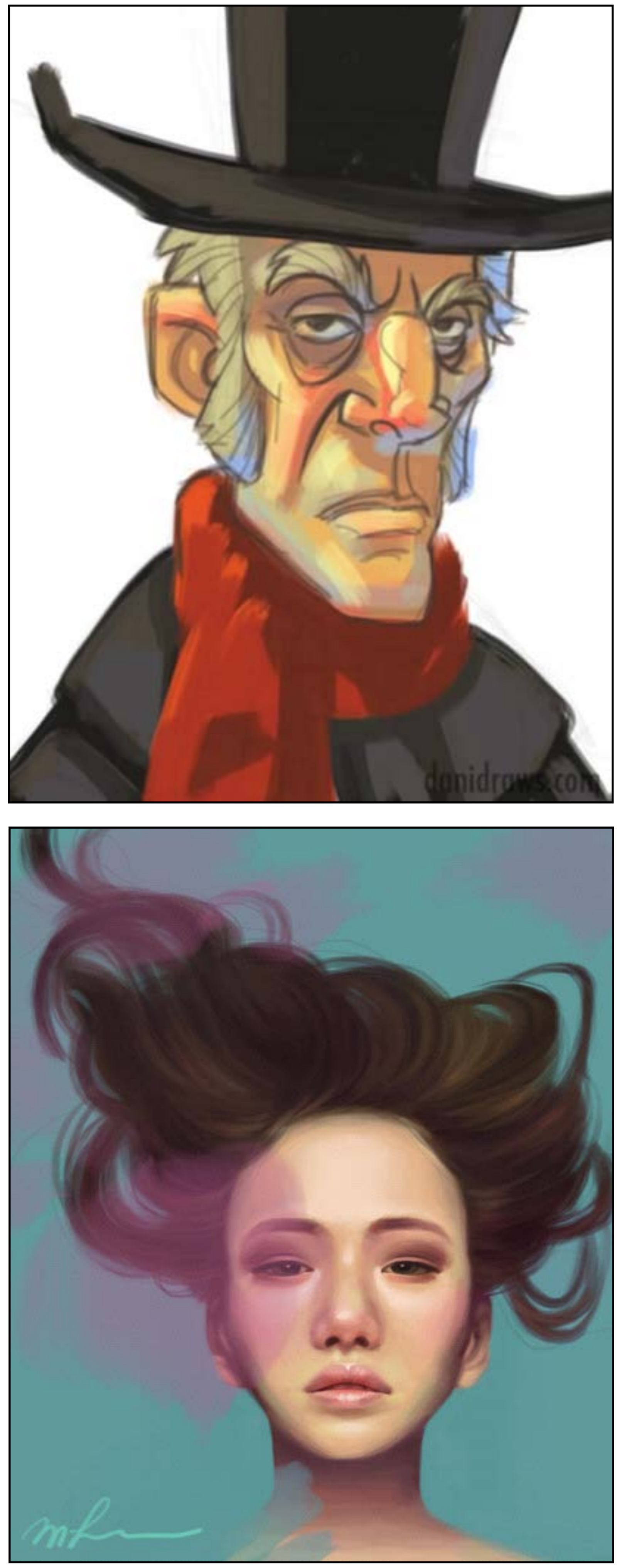


Original

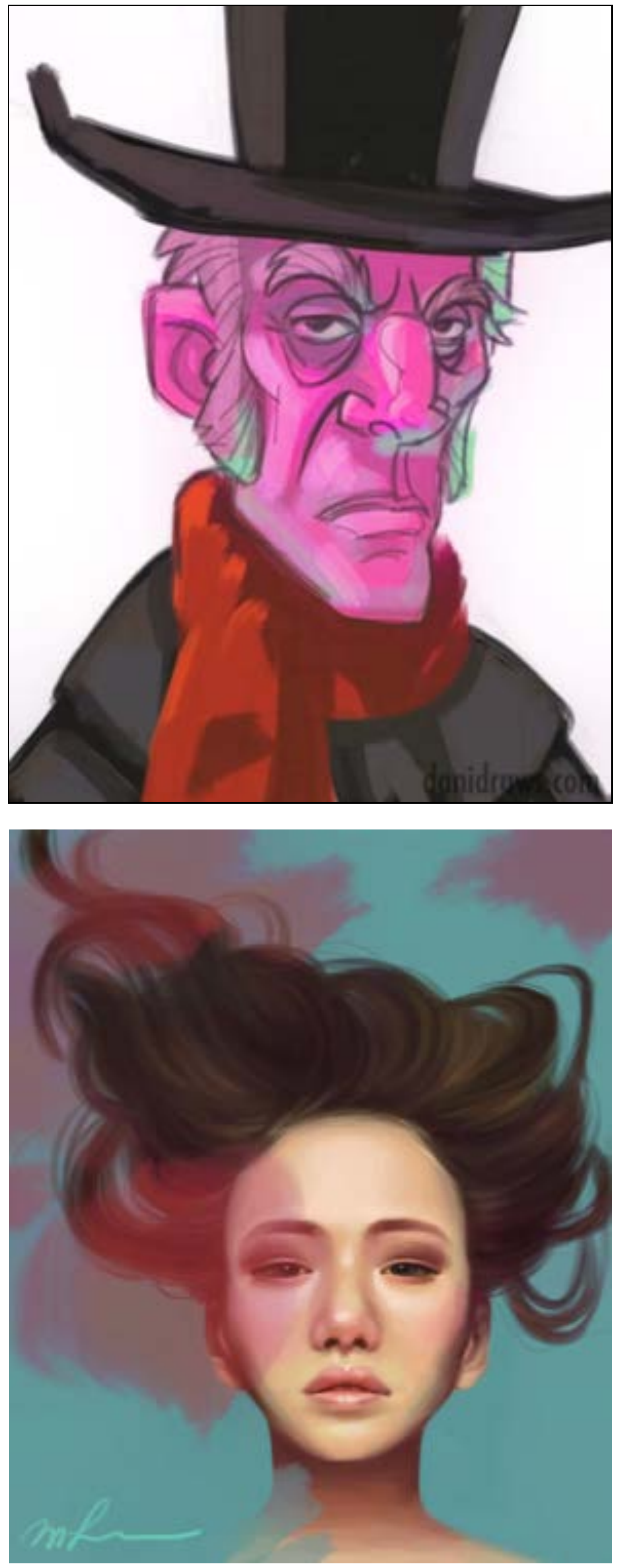


Ours

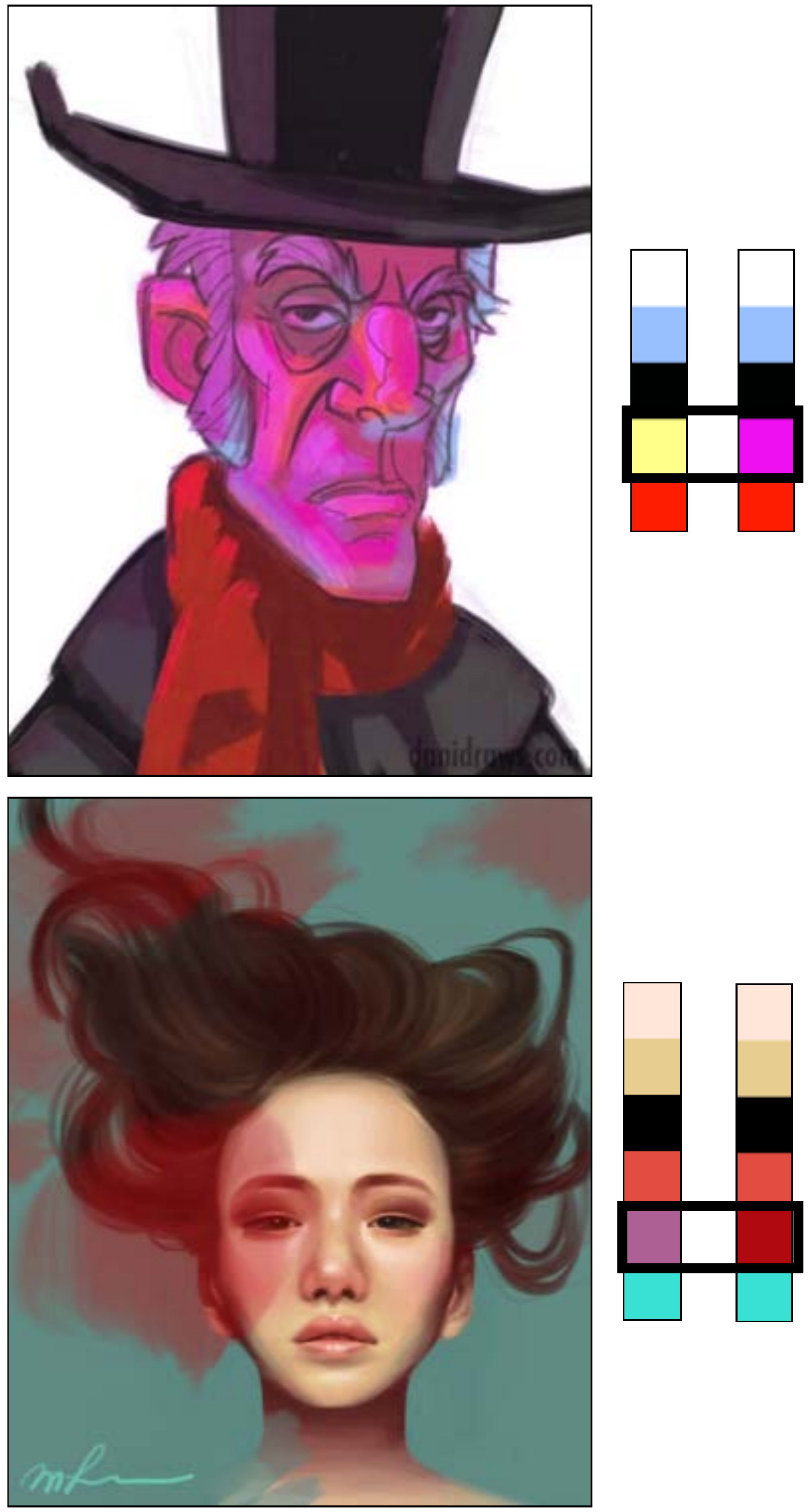
Recoloring comparison with three previous methods



Original

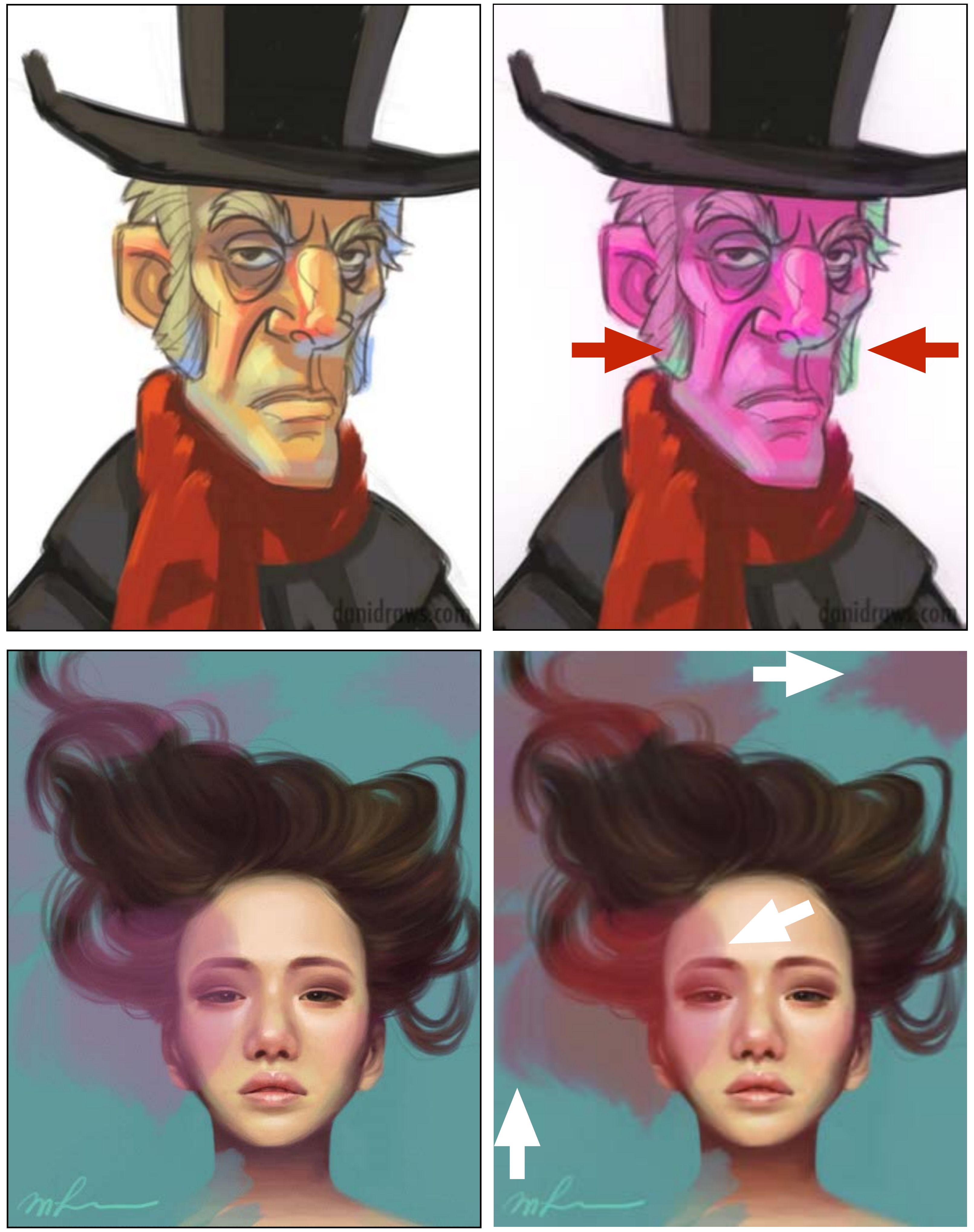


Aksoy et al. 2017



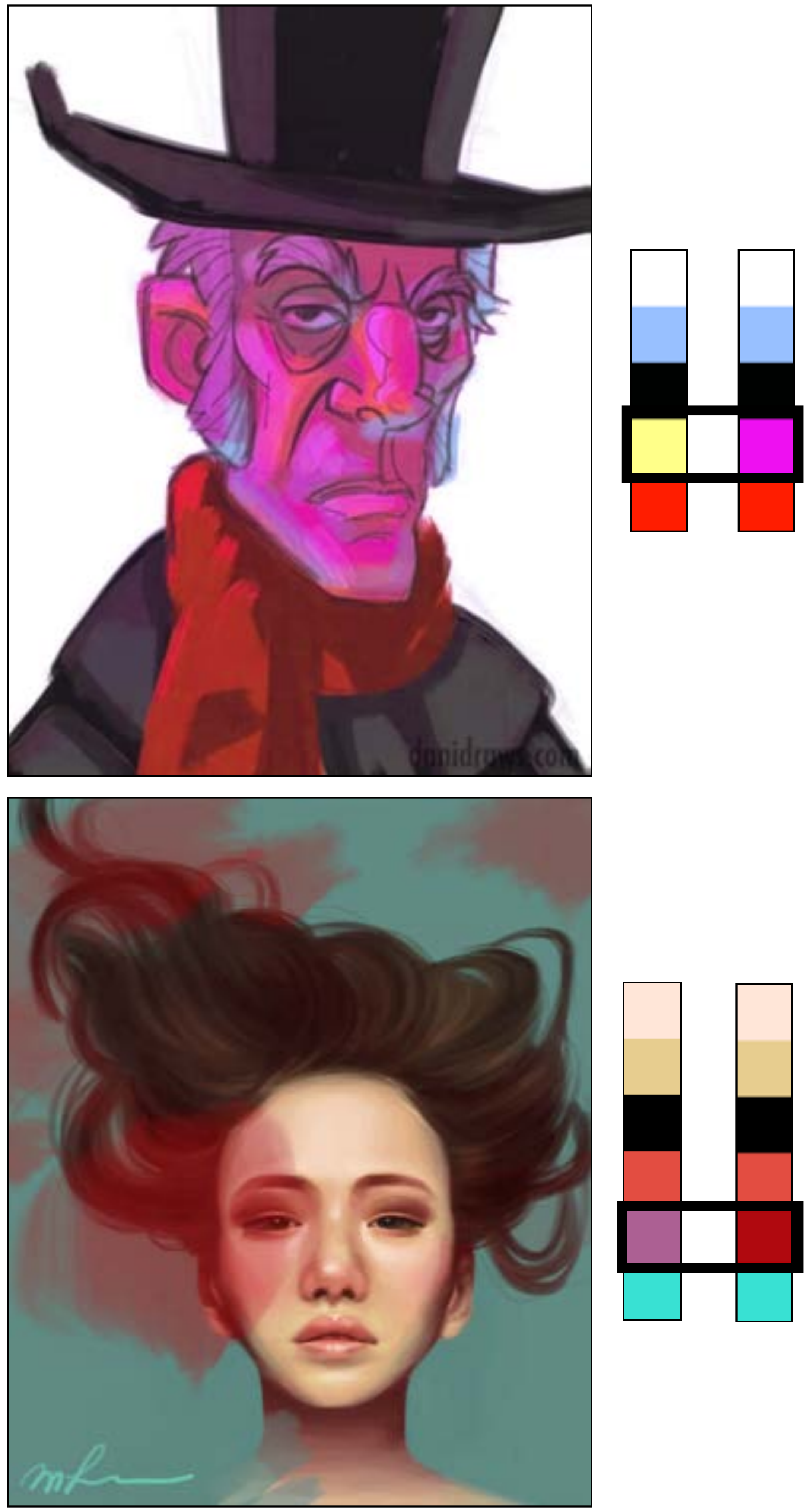
Ours

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

age: girls.png
Reconstruction
Difference
Layers:
hoose File
No file chosen
re-compute RGBXY weights
reate automatic palette
dd Random Palette Color

Prescribed number of layers: 6

☑ colorful

age: girls.png
Reconstruction
Difference
Layers:
hoose File
No file chosen
re-compute RGBXY weights
reate automatic palette
dd Random Palette Color

Prescribed number of layers: 6

☑ colorful

Layer creation from an automatic palette

Choose File
No file chosen
Import/compute RGBXY weights
Create automatic palette
Add Random Palette Color

□ Prescribed number of layers: 6

☑ colorful

Rotation has intertia: □

Choose File
No file chosen
Import/compute RGBXY weights
Create automatic palette

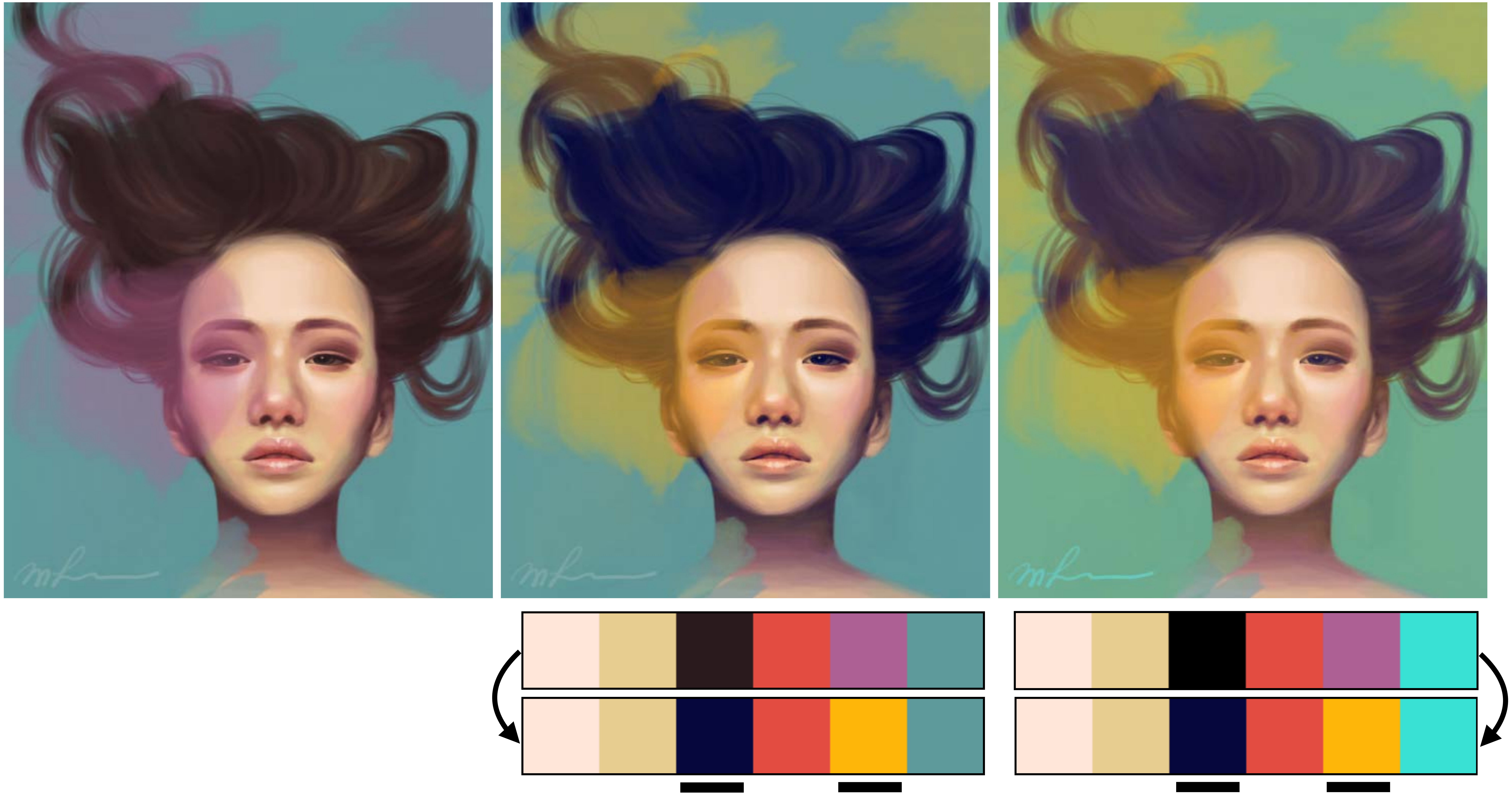
□ Prescribed number of layers: 6

Add Random Palette Color

☑ colorful

Rotation has intertia: □

Interactive decomposition gives more control to the users



original

recoloring with interactively edited palette

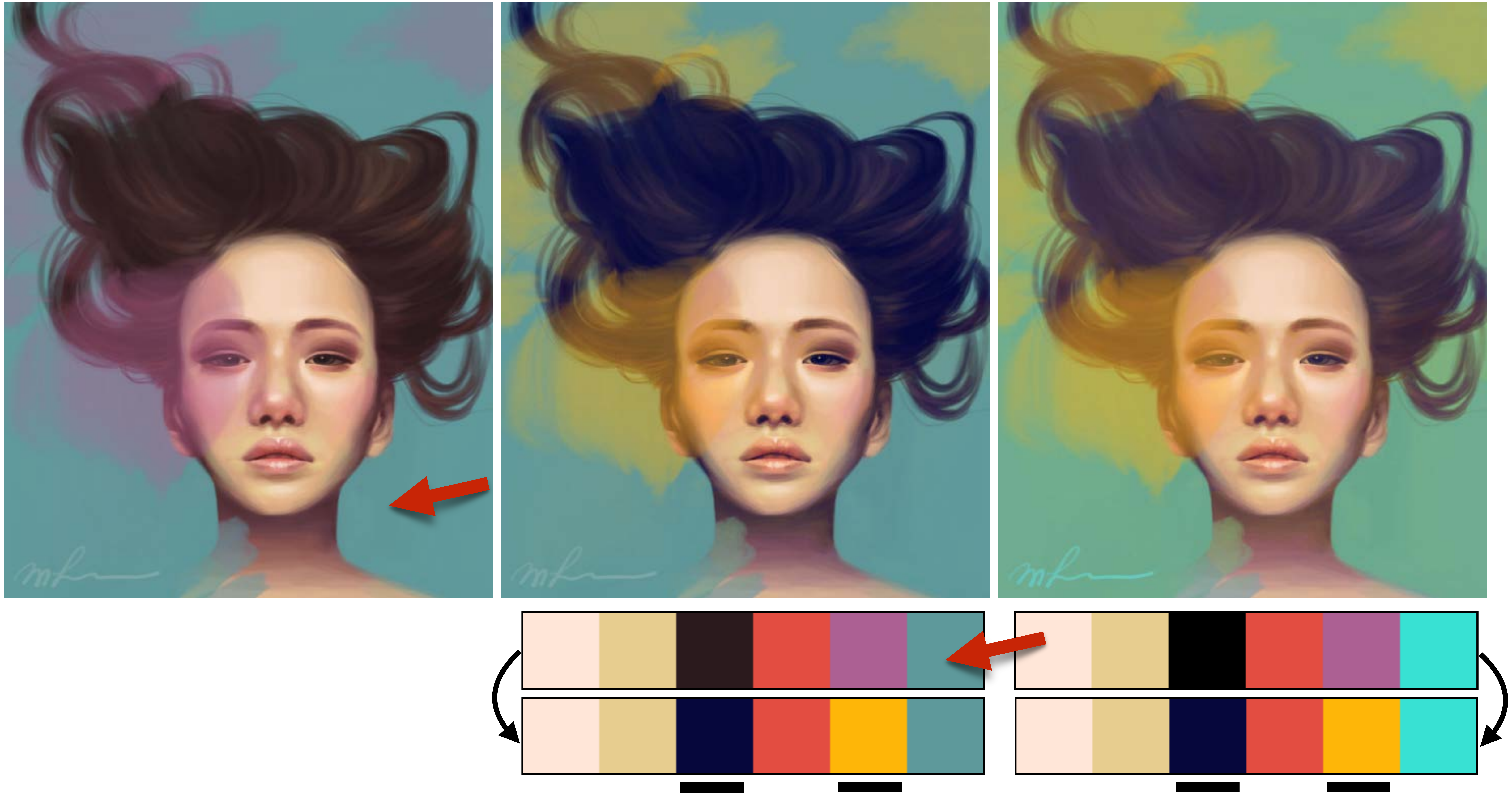
recoloring with automatic palette

Interactive decomposition gives more control to the users

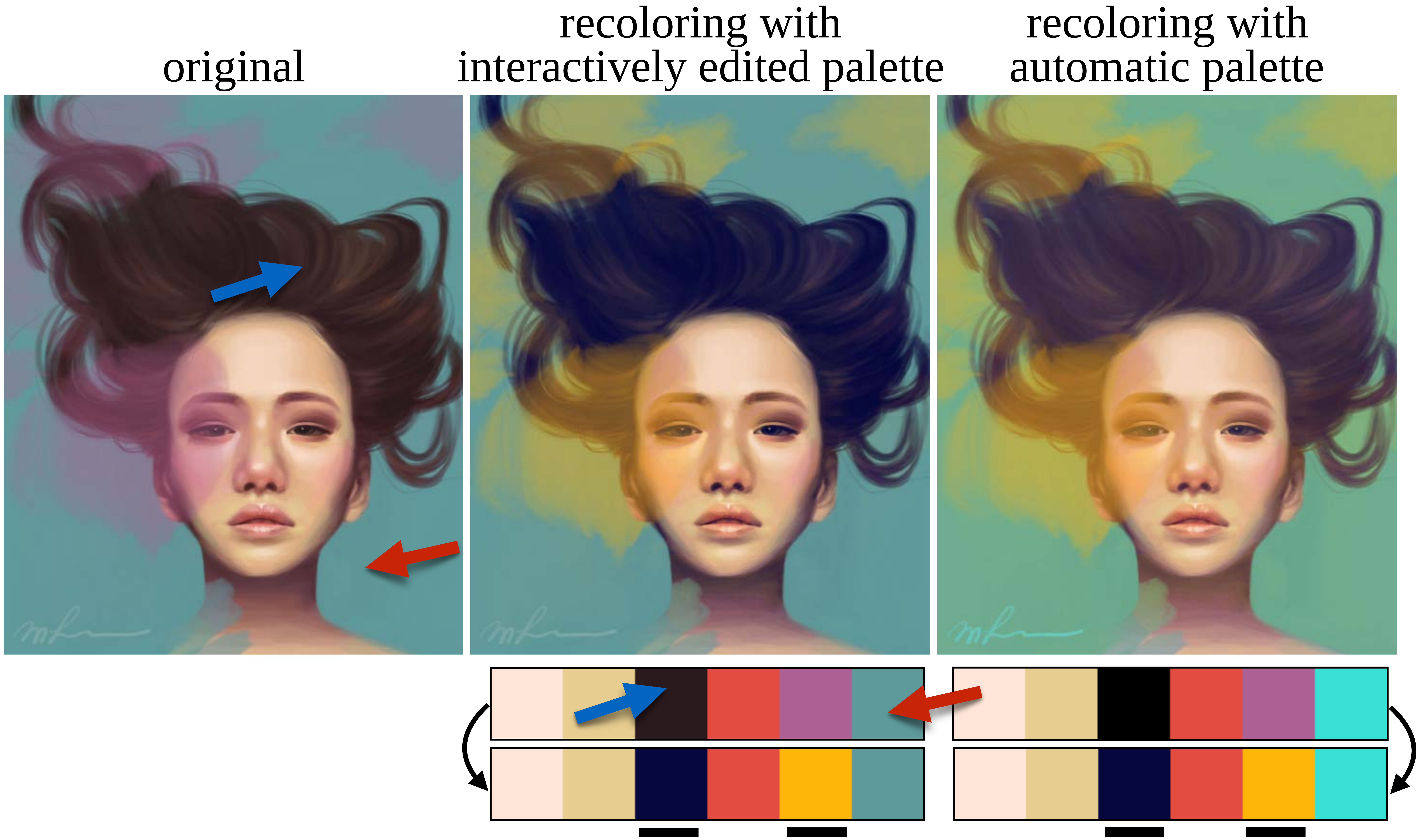
original

recoloring with interactively edited palette

recoloring with automatic palette



Interactive decomposition gives more control to the users



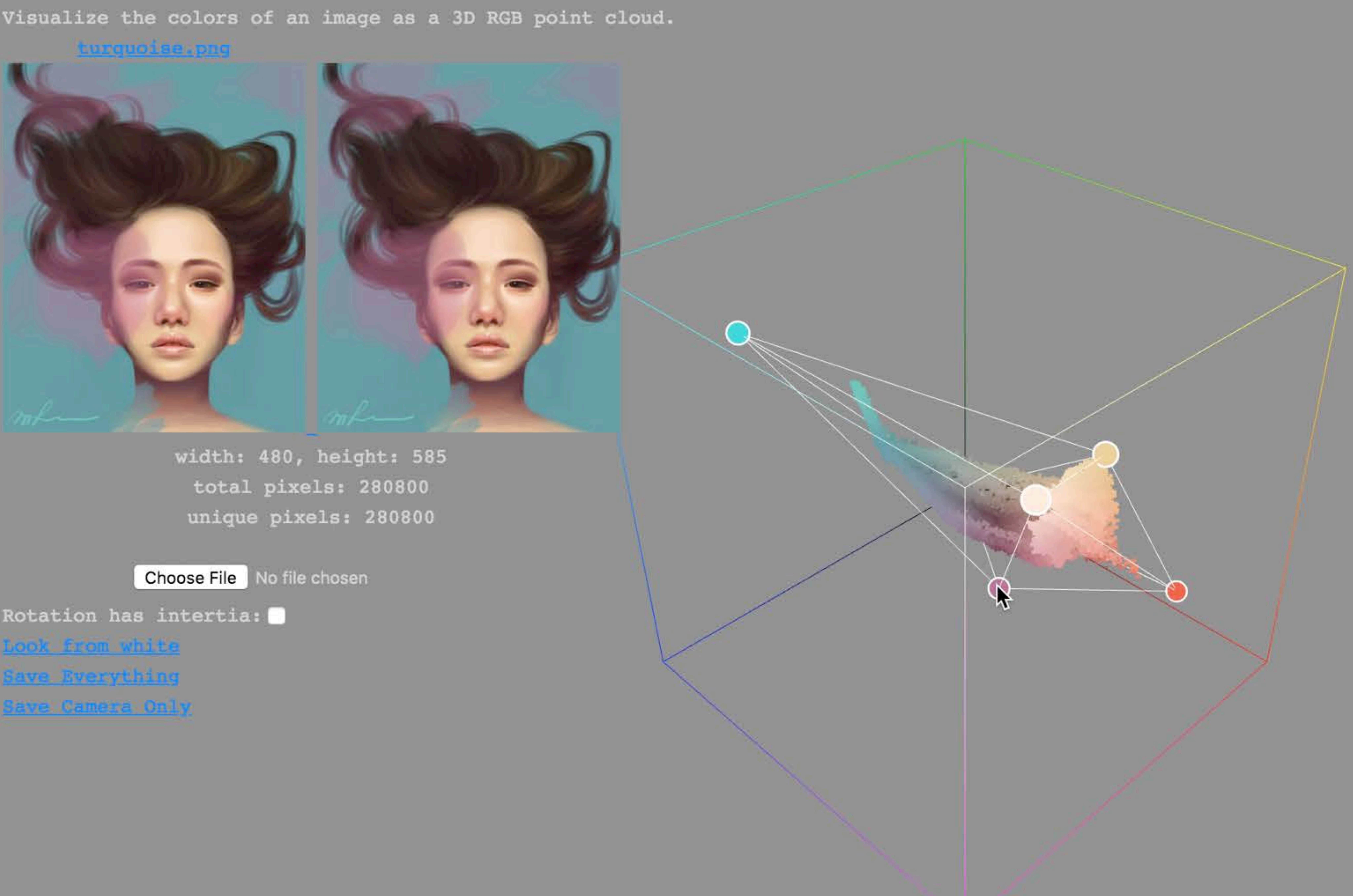
original

recoloring with interactively edited palette

recoloring with automatic palette

Recoloring

Javascript



Visualize the colors of an image as a 3D RGB point cloud.

turquoise.png

width: 480, height: 585

total pixels: 280800

unique pixels: 280800

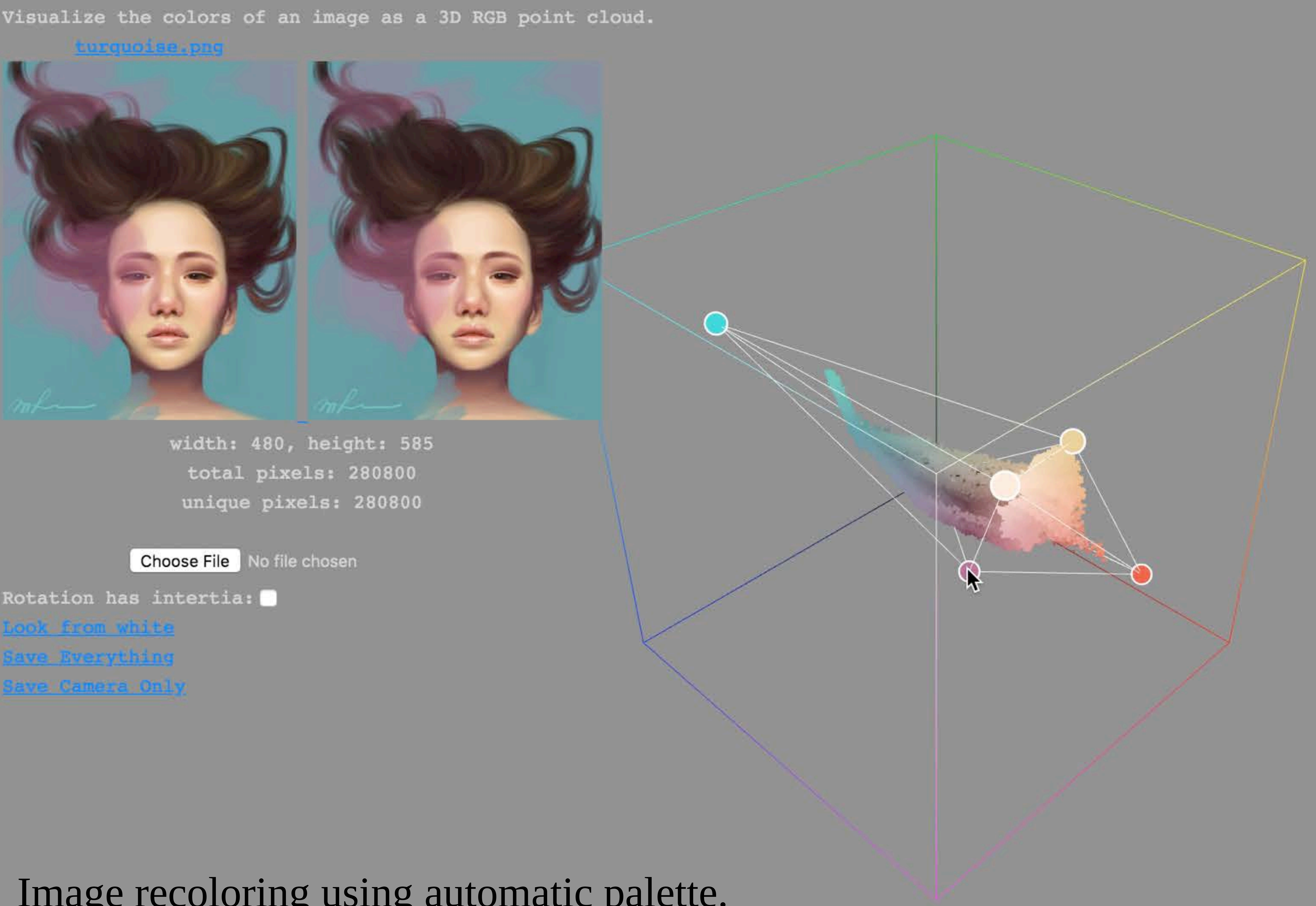
Rotation has intertia: □

Look from white

Save Everything

Save Camera Only

Image recoloring using automatic palette.



Visualize the colors of an image as a 3D RGB point cloud.

turquoise.png

width: 480, height: 585

total pixels: 280800

unique pixels: 280800

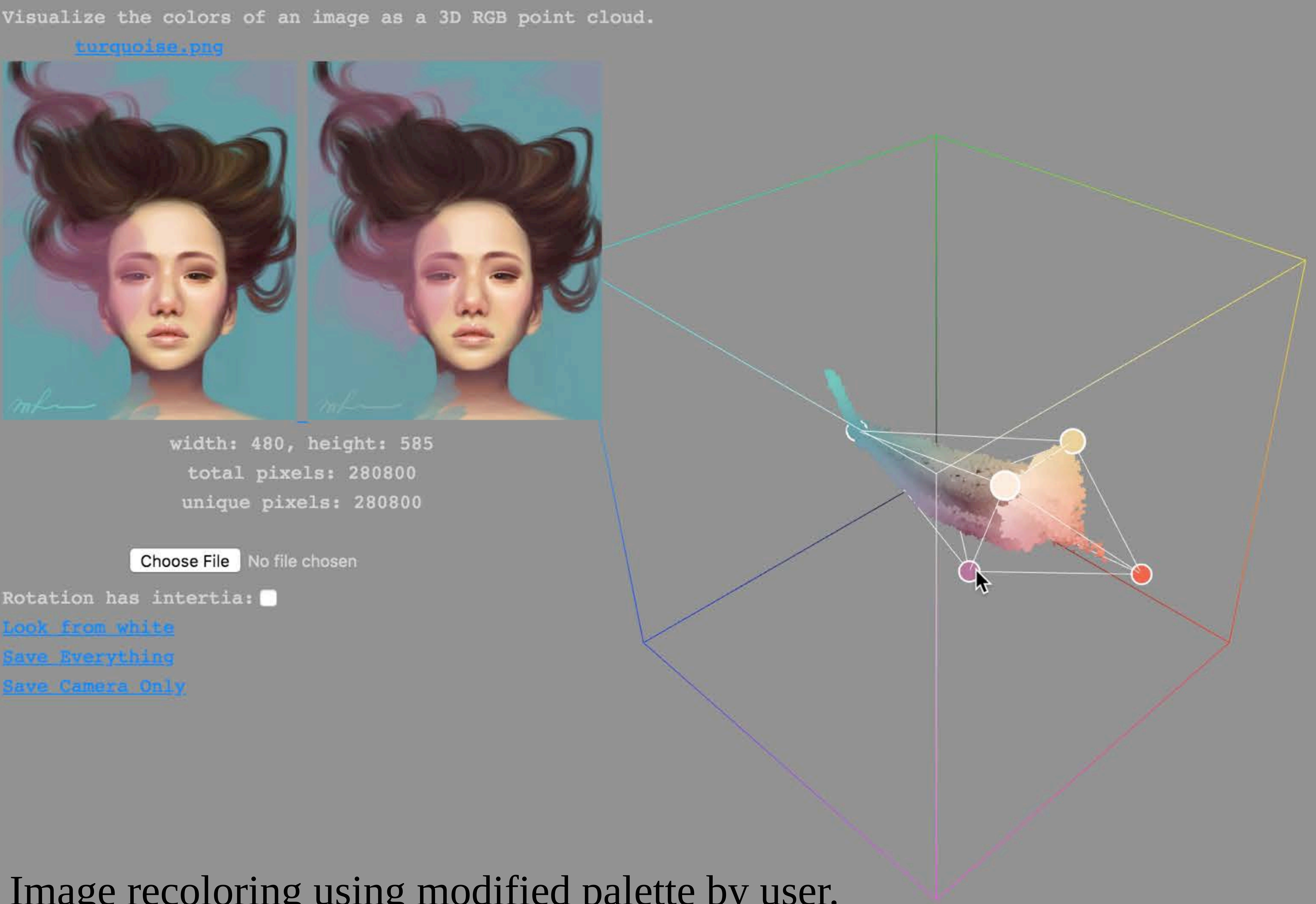
Rotation has intertia: □

Look from white

Save Everything

Save Camera Only

Image recoloring using automatic palette.



Visualize the colors of an image as a 3D RGB point cloud.

turquoise.png

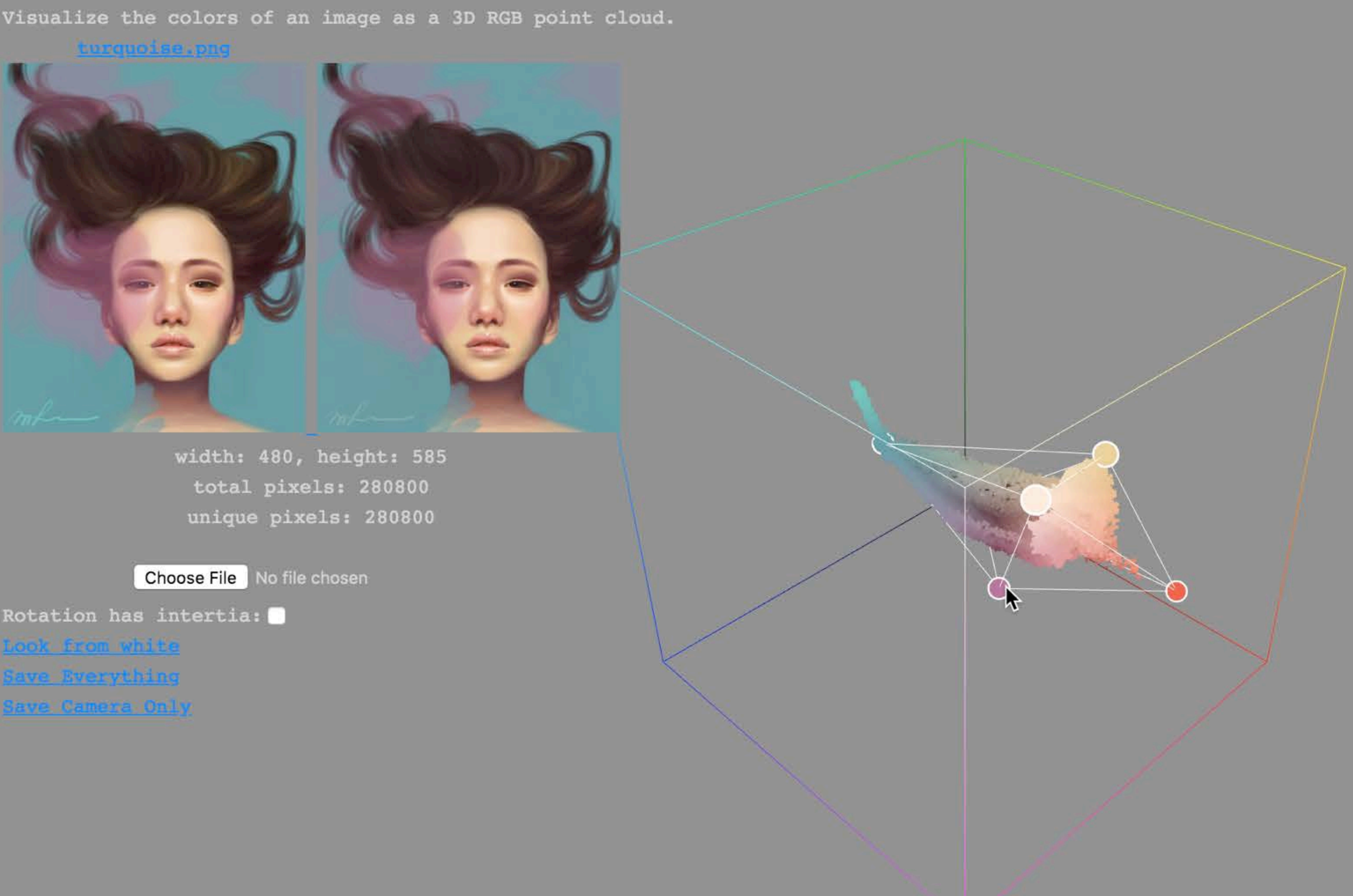
width: 480, height: 585

total pixels: 280800

unique pixels: 280800

Rotation has intertia: □

Image recoloring using modified palette by user.



Visualize the colors of an image as a 3D RGB point cloud.

turquoise.png

width: 480, height: 585

total pixels: 280800

unique pixels: 280800

Choose File No file chosen

Rotation has intertia: □

Look from white

Save Everything

Save Camera Only

Image recoloring using modified palette by user.

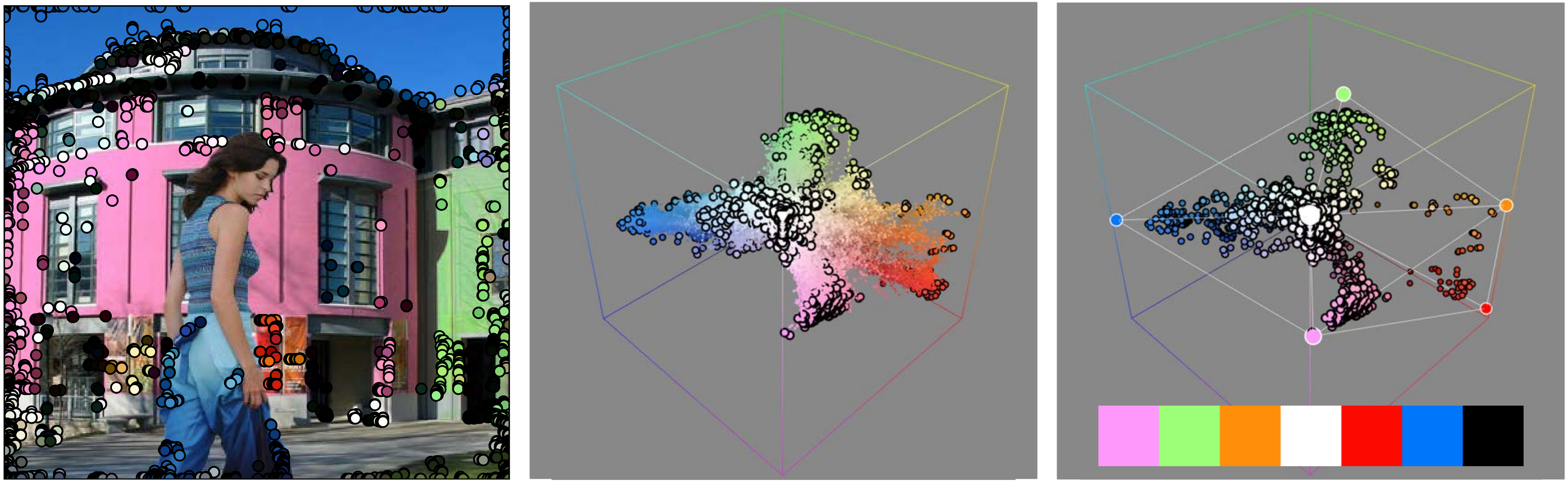
Using automatic palette

Original

Using modified palette

Conclusion

- An extremely efficient approach to layer decomposition via RGBXY geometry



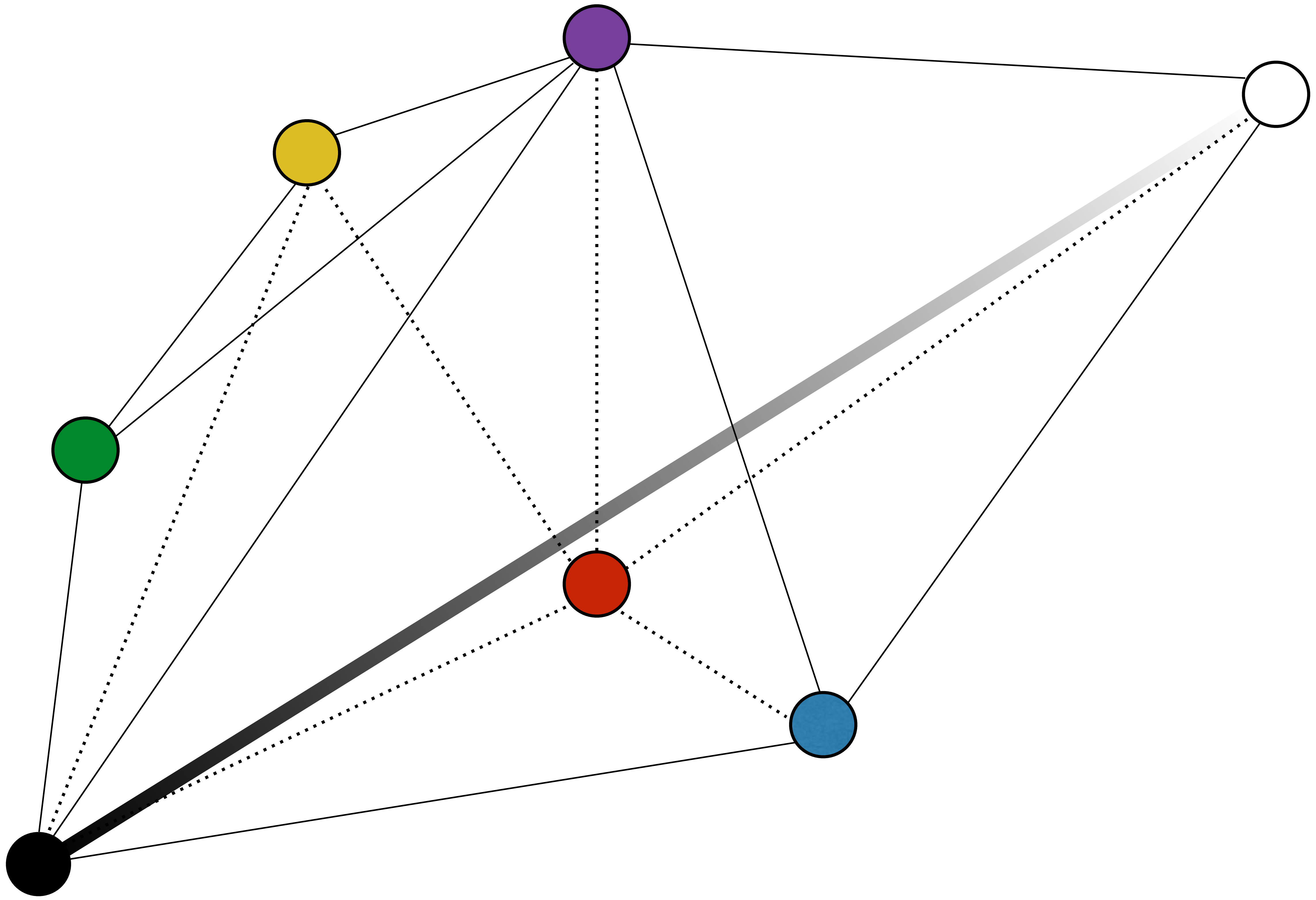
Conclusion

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

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

Conclusion

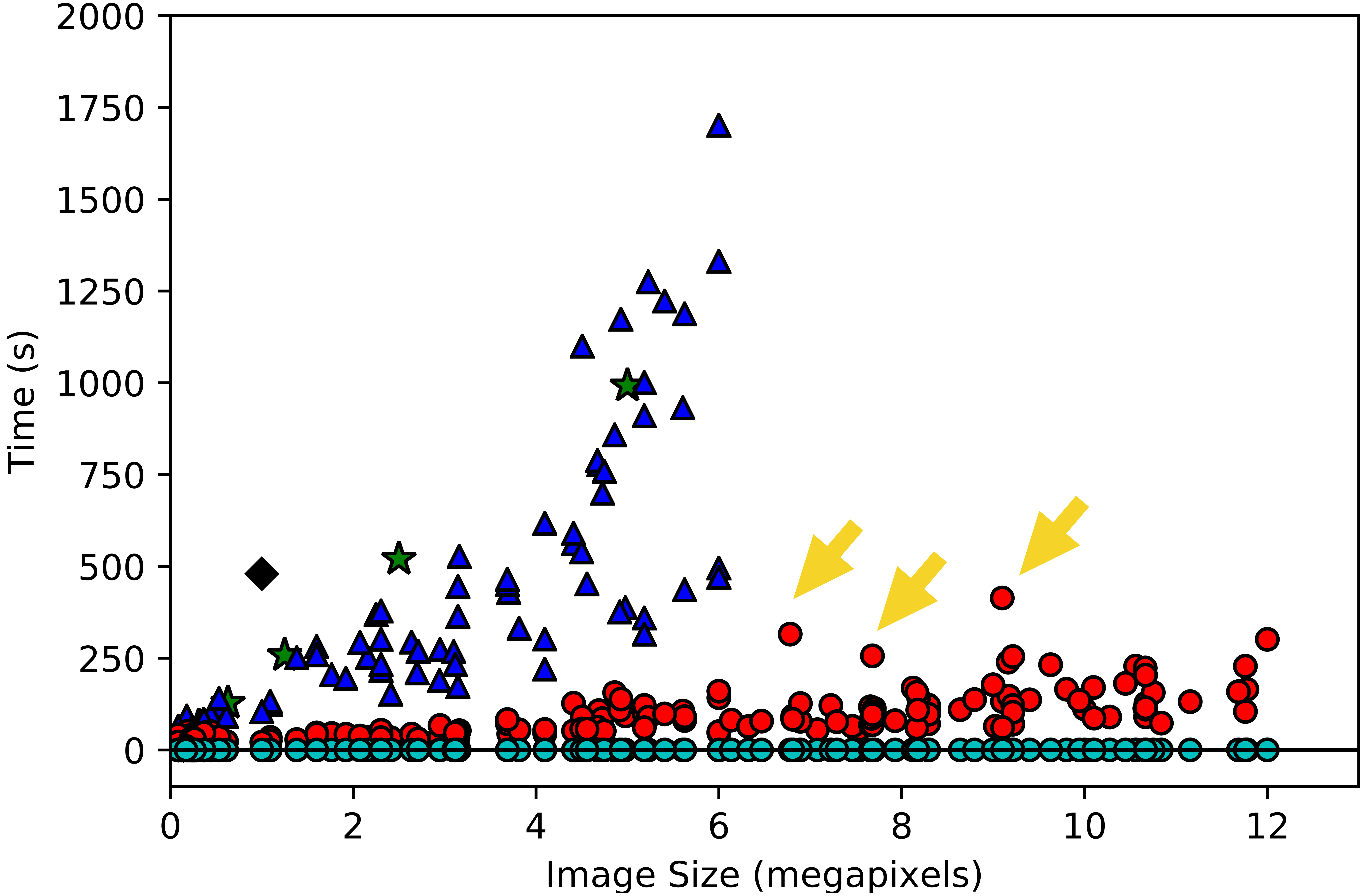
- It’s important to capture the “line of greys”.



Star tessellation

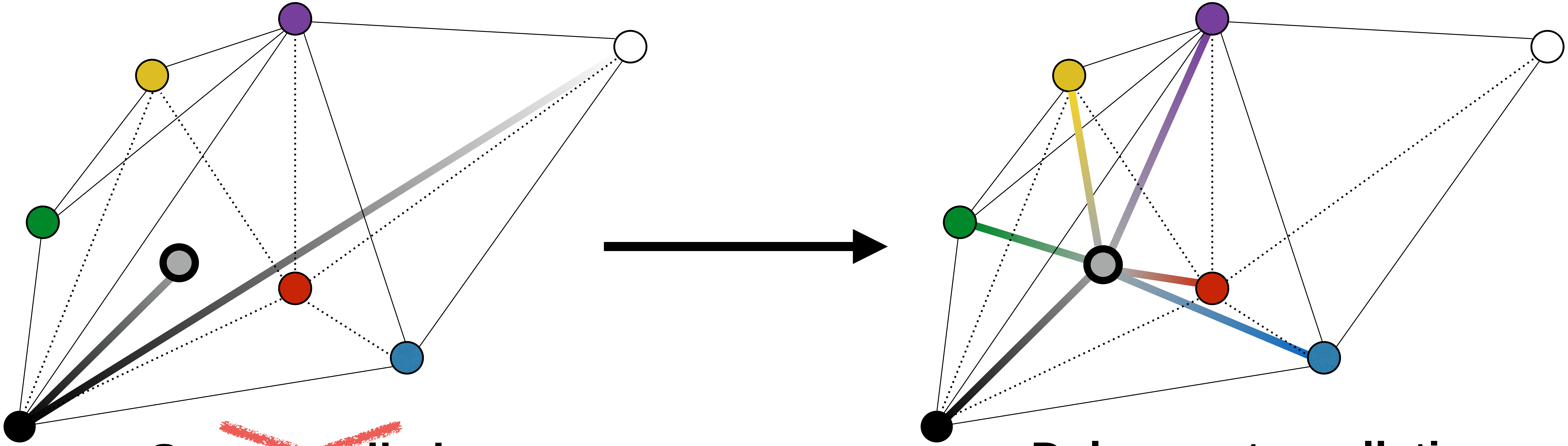
Limitations

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



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.



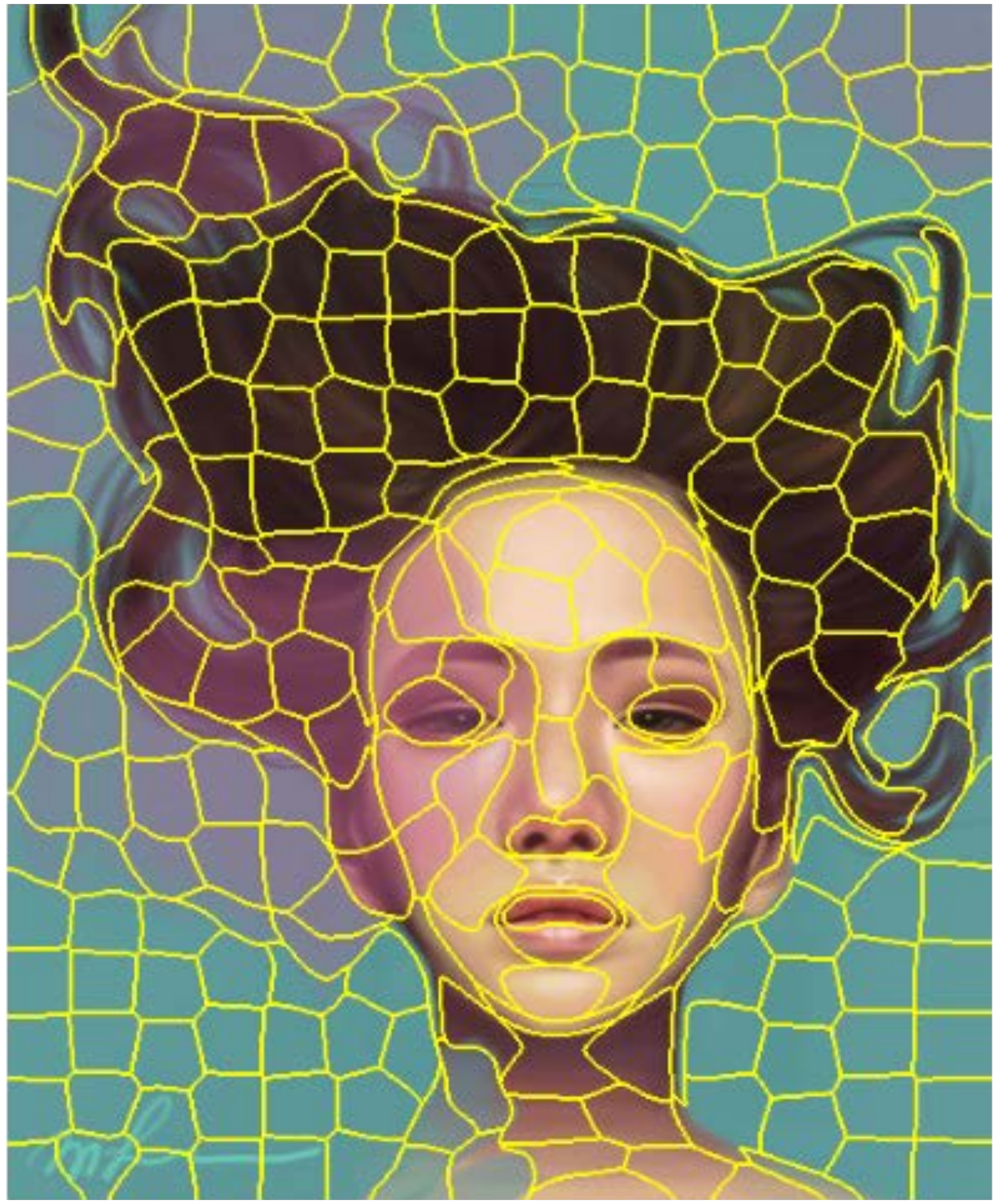
~~Star tessellation~~

Delaunay tessellation

Future Work

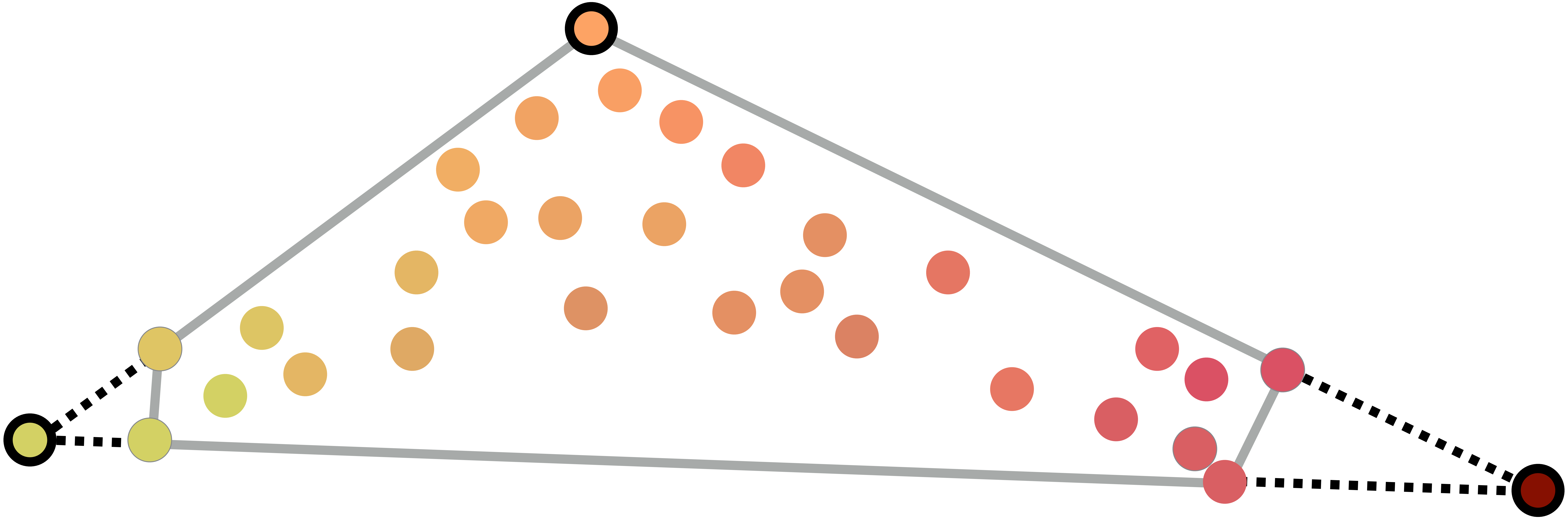
- More speed via super-pixels or parallel convex hull algorithms.

+



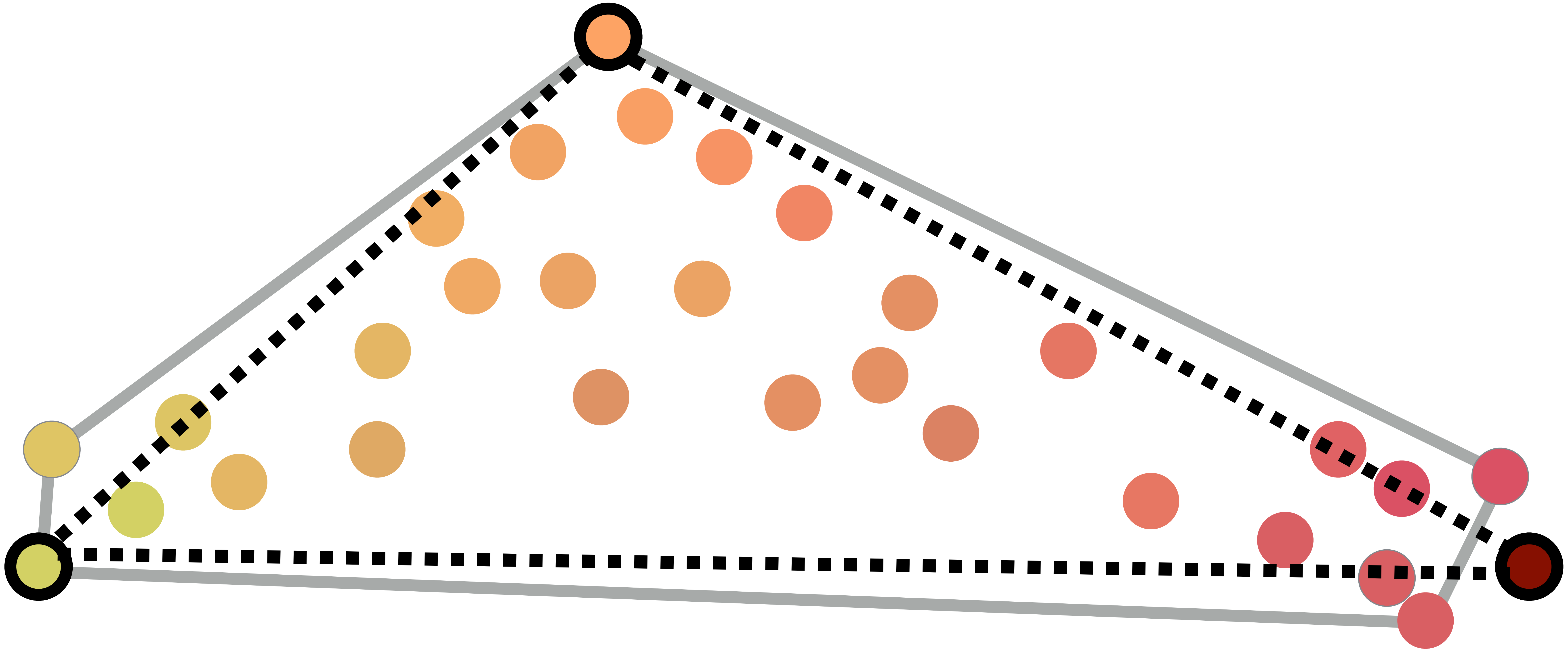
Future Work

- Robustness via approximate convex hull algorithms.



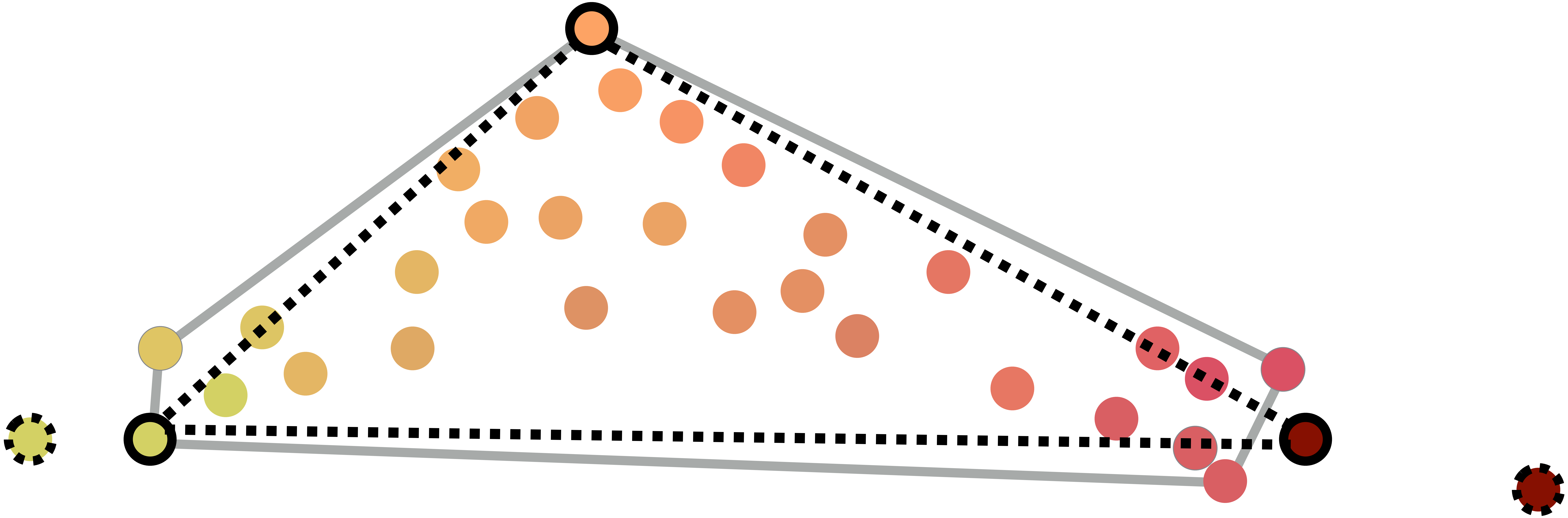
Future Work

- 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.