The Missing Crayon Color

David R. Miller

{dave}@millermattson.com

February 27, 2026

This report is a recreational investigation into the colors in a 120-count box of Crayola crayons. The 120-count box contains the largest number of colors that Crayola currently sells in a single box.[1] Advocating for the underrepresented, we attempt to answer the question, "Which color is least well represented?" In our investigative journey, we explore the problems associated with color spaces and reason about what it means to find the "average" crayon color. We'll determine the largest gap in hue coverage then find the largest color void in perceptual 3D color space.

1. The Colors

The 120 crayon colors are shown in Figure 1.

2. Color Models

Online data is available for Crayola crayon colors in the form of CMYK and RGB color values.[2,3] The CMYK color space has a more limited gamut than the common RGB color spaces, so we will use the RGB values. Unfortunately, the data sources do not specify which RGB color model was used, such as sRGB or Adobe RGB. The sources imply that the RGB values are suitable for display purposes, which implies that they use the sRGB color model.

The conditions under which these RGB colors were obtained are undefined. We don't know the illuminant. We don't know whether bulk wax or a thin smear was photographed to obtain the RGB values. Some of the Crayola pigments will be outside of the sRGB gamut, and it is undefined how out-of-gamut colors were converted to sRGB.

[1] https://www.crayola.com/faqs/what-is-the-largest-box-of-crayola-crayons-available-for-purchase-faq
[3] https://www.color-meanings.com/crayola-crayon-colors/
[3] https://en.wikipedia.org/wiki/List_of_Crayola_crayon_colors

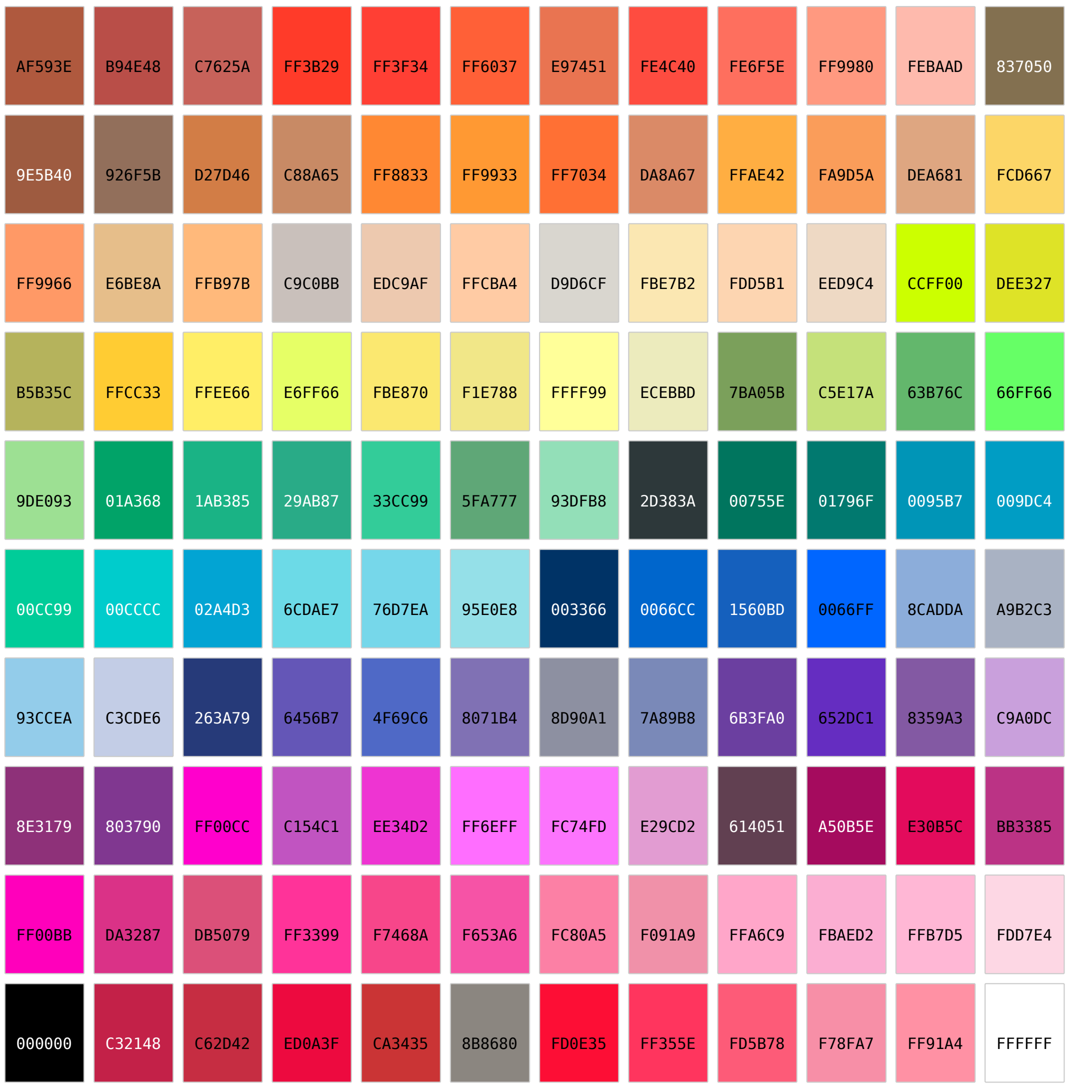


Figure 1: The data set

The sRGB color model is problematic for additional reasons:

- sRGB is gamma-encoded with a $\sim$2.2 curve, so mixing raw sRGB values combines perceptually weighted signals, not physical light quantities.
- Even if we adjust for the gamma curve, linear RGB still uses additive light mixing which does not work well for modeling subtractive pigment mixing.
- An additive sRGB color model does does not physiologically represent our perception of color: equal steps in an RGB space don't correspond to equal perceptual differences. Combined RGB colors will be biased toward whatever colors have high channel values.

What we really need is a subtractive color model using reflective spectrographic measurements of each crayon.

Lacking spectrographic equipment and lacking the will to buy and measure 120 crayons, for some of our analyses we will cheat and reconstruct plausible reflective spectra from our sRGB values.

2.1. Reflective Color Model

To model subtractive physical pigment mixing, we will construct plausible spectra from sRGB values. We say "plausible" instead of "accurate" because an infinite variety of spectra could be mapped to the same RGB value. In other words, RGB is a very lossy three-channel compression of a continuous function with infinite degrees of freedom. Two distinct spectral power distributions could map to the same RGB value. But converting an RGB value to a spectral power distribution is fundamentally underdetermined. There are infinitely many valid solutions, and no physical or mathematical principle points us to a unique one. We can only create one *plausible* spectrum from an RGB value.

Our reflective model is based on the Least Log Slope Squared (LLSS) algorithm developed by Scott Burns.[4] Sample code is shown in Appendix A. The algorithm precomputes a 3×36 matrix T that converts a 36-point spectral reflectance curve (380–730 nm, 10 nm steps) into linear sRGB, using the CIE 1931 color matching functions, D65 illuminant,[5] and a standard XYZ-to-sRGB matrix.

Given an sRGB color, the algorithm recovers a plausible reflectance spectrum. This is an underdetermined inverse problem (3 knowns, 36 unknowns), so it finds the smoothest reflectance curve matching the color by solving a constrained optimization via Newton's method, using a tridiagonal smoothness penalty and Lagrange multipliers.

Reflectances are combined by weighted geometric mean (averaging in log-reflectance space). This models how pigments multiplicatively absorb light at each wavelength.

Reflectance can be converted back to sRGB through T and gamma encoding.

[4]Scott Allen Burns, March 2015. Licensed under a Creative Commons Attribution-ShareAlike 4.0 International License (http://creativecommons.org/licenses/by-sa/4.0/). https://scottburns.us/reflectance-curves-from-srgb/

[5]https://cielab.xyz/pdf/cie.15.2004%20colorimetry.pdf

2.2. Other Color Spaces

In some of our investigations, we will compare colors without combining them, so we can get by with an additive color model for simplicity. We could use the original sRGB data set, but sRGB color space is not perceptually uniform. The Euclidean distance between two RGB points is not a comparable measure of perceptual color differences. When we are not combining colors that require a subtractive method, we will convert our sRGB colors to a different color space such as Oklab[6] which has a more perceptually uniform distribution of colors.

Oklab is a perceptually uniform color space designed by Björn Ottosson to address shortcomings in earlier spaces that were intended to be perceptually uniform, but were not quite right. Converting from sRGB to Oklab involves first linearizing the sRGB values (removing the gamma curve, so that $C_{\text{linear}} = C_{\text{sRGB}}^{2.4}$ approximately), then applying a matrix transform to move into an LMS-like cone-response space, followed by a cube-root nonlinearity, and finally a second matrix rotation to yield the three Oklab coordinates: L (perceptual lightness), a (green–red axis), and b (blue–yellow axis). The design goal is that equal Euclidean distances in the (L, a, b) space correspond to roughly equal perceived color differences. We will also make use of OKLCH representation, which is simply a cylindrical reparameterization of Oklab where the rectangular a and b coordinates are converted to a polar form, giving a chroma $C = \sqrt{a^2 + b^2}$ and a hue angle $H = \text{atan2}(b, a)$, while L remains unchanged. This makes OKLCH more intuitive for tasks like adjusting saturation or rotating hues while maintaining perceptual uniformity.

3. Average Color

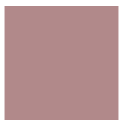
Given a box of 120 crayons, the concept of "average color" can be interpreted in various ways:

- The data analyst asks, "What is the arithmetic average value in this color dataset?"
- The materials engineer asks, "What color results from combining these pigments?"
- The neurologist asks, "What is the perceptual average color?"

Let's examine each interpretation.

3.1. Arithmetic Average

The simple arithmetic average of the 120 sRGB color triplets is color #B1898A:



[6]Ottosson, B. (2020, December 23). "A perceptual color space for image processing." https://bottosson.github.io/posts/oklab/

3.2. Pigment Addition

Suppose we melted all 120 crayons together. What color would result? We cannot determine this accurately with an emissive color space such as sRGB, so we will combine the colors with a reflective algorithm described in Section 2.

This algorithm yields a combined pigment color that, when converted back to sRGB, is color #a98181:

Because we are working with *plausible* spectral data, this might not accurately represent real-world mixing of crayon pigments.

3.3. Perceptual Average

The Oklab color space is designed so that equal numerical distances correspond to equal perceived differences. This will tell us what a human observer would consider the "middle" color of the set. Converting sRGB colors to Oklab removes the gamma encoding from the sRGB values and gives us a mostly uniform linear space to work in. The average color in Oklab color space is $L = 0.6997, a = 0.0490, b = 0.0177$. Converted back to sRGB, the perceptual average color is #BD9291:

3.4. Chardon Skin Tone

For a fun excursus, we can compare the average color computed above with human skin tones using the Chardon skin tone classification formula based on the L and b values of the CIE L*a*b* color space:[7]

$$ITA^{\circ} = \arctan\left(\frac{L^{*} - 50}{b^{*}}\right) \times \frac{180}{\pi}$$

The result of the pigment combination calculation above is color #a98181, which converts to L*a*b* values of $L = 57.77, a = 15.40, b = 5.970$. The ITA angle is 52.47°, which falls in the

[7]Chardon A, Cretois I, Hourseau C. Skin colour typology and suntanning pathways. Int J Cosmet Sci. 1991;13(4):191-208. doi:10.1111/j.1467-2494.1991.tb00561.x

"light skin" category of the Chardon classification system. The a^* value of 15.4 falls within the typical range of 5 to 16.

If melted together, the average crayon color could be called a light/fair European skin tone, which happens to correspond to Crayola's dominant market region of North America and Europe. It remains an open question whether wax drawing instruments manufactured in other geographic regions have a different "melted" average color.

4. Hue Coverage

Before finding the missing crayon color, let's first look only at the coverage of hues. For this analysis, we'll convert the 120 sRGB color values to OKLCH color space so that we're working in a color space with a perceptually uniform sequence of hues. OKLCH is a cylindrical transformation of the Oklab color space as described in Section 2.2.

Colors with a low saturation do not have meaningful or interesting hues for this analysis, so we will remove from the dataset all colors with a low saturation (LCH chroma channel < 0.01).

Also, very dark and very light colors will have hues that perceptually skew this analysis into misleading results, so we will remove those colors (LCH L channel < 0.05 and > 0.95).

The hues of the remaining colors are plotted in Figure 2 and Figure 3.

On both plots, the curve is a circular kernel density estimate (KDE) using a von Mises kernel (the circular analog of a Gaussian) with a bandwidth of 12 degrees. Higher values indicate hue regions that are more densely represented in the color set. The dashed red line marks the minimum of the KDE curve, i.e., the least-represented hue. The dashed blue line shows the center of the largest gap between hues. On the polar plot, the KDE is normalized to a maximum of 1 and displayed as a filled polar area; colored scatter dots at a fixed outer radius show the individual hue samples, and an outer ring displays the hue wheel for reference. On the linear plot, the scatter dots are shown along the bottom, colored by their actual sRGB values.

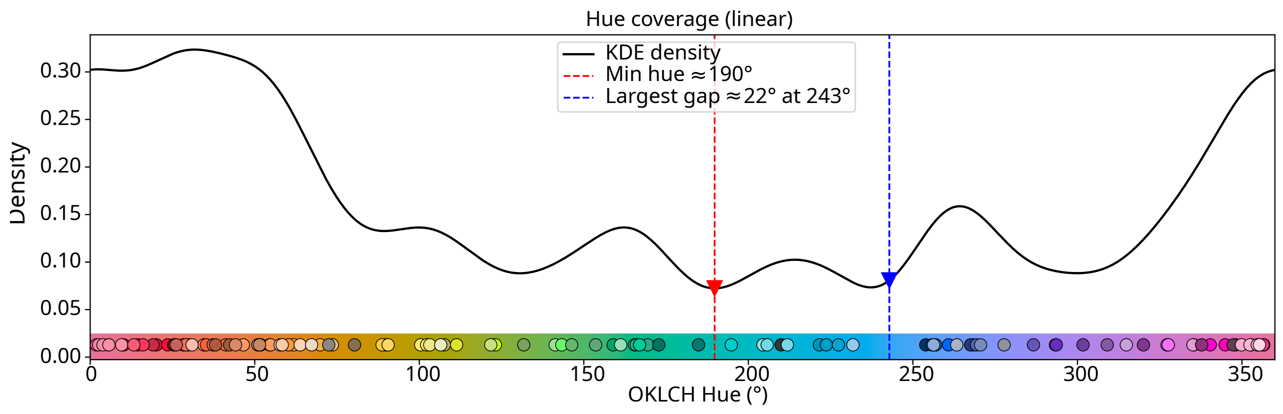


Figure 2: Hue coverage, linear plot

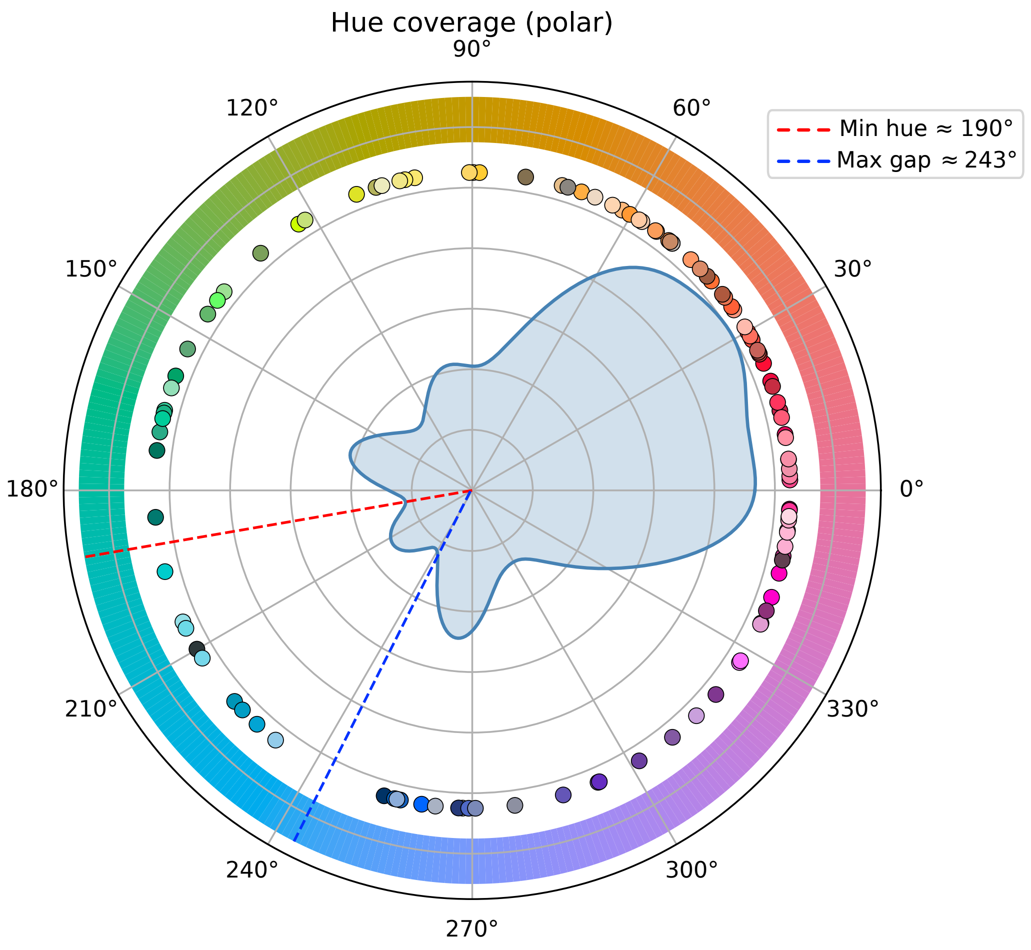


Figure 3: Hue coverage polar plot

5. Warm or Cool?

Before finding the missing crayon color, let's determine if the crayon colors are biased toward warm or cool colors. It's more likely that we will find the missing crayon color in whichever region—warm or cool—that is least represented in the colors.

For this analysis, we'll first convert the sRGB values to Oklab so that we can work in a perceptually uniform (L, a, b) color space. From Oklab we derive the polar form OKLCH, with lightness L, chroma $C = \sqrt{a^2 + b^2}$, and hue angle $h = \operatorname{atan2}(b,\ a) \bmod 360°$. Working in h gives us perceptually uniform, equal angular intervals in h corresponding to roughly equal perceived hue shifts. The warm/cool boundary is at $h = 110°$ and $h = 315°$, so that reds, oranges, and yellows ($0° \le h < 110°$, and the wraparound magentas at $315° \le h < 360°$) are classified as warm, while greens, cyans, and blues ($110° \le h < 315°$) are classified as cool. This division aligns with color theory and with the natural clustering of opponent hues in the Oklab a–b plane. The chroma-weighted ratio $\frac{\sum_{i \in \text{warm}} C_i}{\sum_{i \in \text{cool}} C_i}$ is shown alongside the simple count ratio because a highly saturated color contributes more visually than a near-grey one with the same hue label.

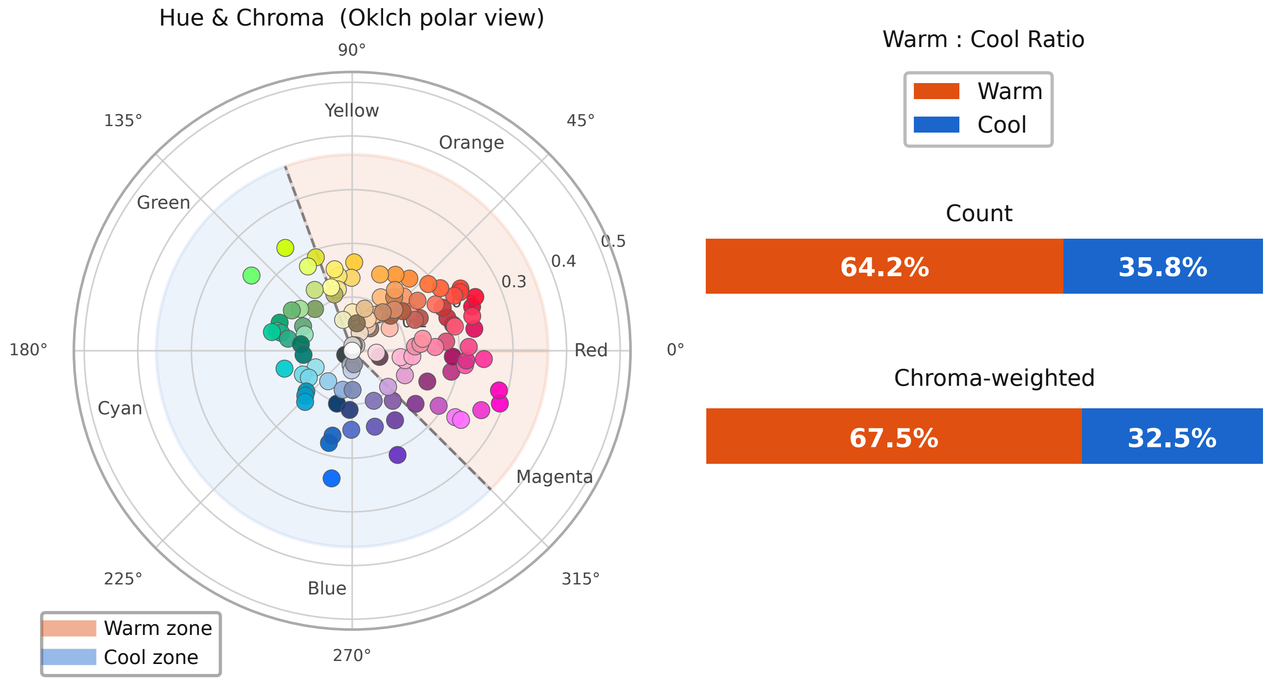
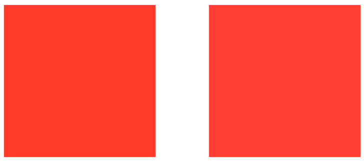


Figure 4: Crayon colors are biased toward warm colors

6. Most Similar Colors

In our quest to find the missing crayon color, let's pause to ask the inverse question, "What are the most similar colors in a box of 120 crayons?"

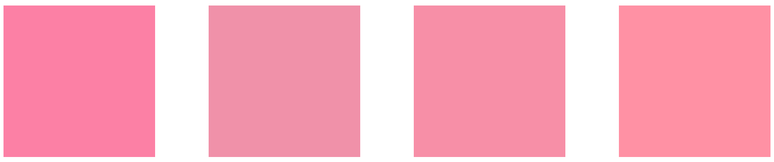
To find the two most similar colors, we convert the 120 sRGB color values to Oklab color space, which is a perceptually uniform color space. The nearest two colors are the two colors with the minimum Euclidean distance between them in 3D Oklab space. Those two most similar colors are #FF3B29 and #FF3F34 with an Oklab distance of only 0.007935. Crayola calls these colors "Mango Tango" and "Red-Orange."



To find the set of four most perceptually similar colors, one might think that we could look for the minimum-volume convex hull around any four color points in 3D Oklab space. However, our 120 crayon colors include many tight groupings that are nearly coplanar, resulting in degenerate convex hull geometries.

Instead, we will apply a different geometric tightness criterion. Once all 120 colors are embedded as points in Oklab space, the tightness of any candidate subset $S = \{\mathbf{c}_1, \ldots, \mathbf{c}_N\}$ is measured by its diameter, meaning the largest pairwise Euclidean distance within the group, $\operatorname{diam}(S) = \max_{i \neq j} \|\mathbf{c}_i - \mathbf{c}_j\|_2$. We then find the subset S^* of size N that minimizes this diameter, $S^* = \arg\min_{S \subseteq C,\, |S| = N} \operatorname{diam}(S)$, which geometrically means finding the N points that can be enclosed in the smallest possible sphere in Oklab space. To keep the search tractable and avoid comparing all $\binom{120}{N}$ combinations, candidates are restricted to the k-nearest-neighbor neighborhoods of each color, on the assumption that the optimal tight group must lie within a small local region of the color space.

The four colors forming the tightest perceptual group are #FC80A5, #F091A9, #F78FA7, and #FF91A4. Crayola calls these colors "Tickle Me Pink," "Mauvelous," "Pink Sherbert," and "Salmon."



7. The Missing Color

Crayon colors are more than pure hues. A gap in color coverage does not necessarily correspond to the largest gap in hue coverage. We need a way to visualize the missing color in 3D color space that captures saturation and lightness.

We convert our 120 sRGB colors to Oklab color space, where the color distribution is perceptually uniform.

Our goal is then to find the point in the color space (while remaining in the sRGB gamut) that is maximally far from every existing color, where "far" is measured as Euclidean distance in the Oklab color space. Formally, given a set of existing colors $\mathcal{C} = \{\mathbf{c}_1, \dots, \mathbf{c}_{120}\} \subset \mathbb{R}^3$ (in Oklab coordinates), we will look for $\mathbf{c}^* = \arg\max_{\mathbf{c} \in \mathcal{G}} \min_i \|\mathbf{c} - \mathbf{c}_i\|_2$, where $\mathcal{G}$ is the set of all Oklab triplets that map back to valid sRGB values (i.e., the sRGB gamut boundary acts as a constraint). This is equivalent to finding the center of the largest empty sphere that fits inside the gamut without containing any existing color. We proceed in two phases: first, a coarse brute-force search evaluates $64^3 = 262\,144$ candidate points on a uniform grid over the sRGB cube $[0, 1]^3$, then we convert each to Oklab, and use a KD-tree to efficiently query the nearest-neighbor distance; second, the best coarse candidate seeds a global refinement via differential evolution, which maximizes the objective $f(\mathbf{c}) = \min_i \|\mathbf{c} - \mathbf{c}_i\|_2$ over the sRGB cube. At the optimum, $\mathbf{c}^*$ is typically equidistant from three or more nearest neighbors. This is the expected geometric condition, analogous to how the circumcenter of a Delaunay triangle is equidistant from its three vertices. In our case, the solution at Oklab $(L, a, b) = (0.167, 0.061, -0.053)$ is equidistant from three existing colors at a gap radius of ≈ 0.186, confirming convergence to a true optimum.

In Figure 5, The left panel shows the Oklab a–b chromaticity plane, where a runs from green (negative) to red (positive) and b runs from blue (negative) to yellow (positive); the dashed circle centered on the new color has a radius equal to the gap distance of 0.186. It is tempting to think the new color is misplaced because several existing dots appear to fall inside or near the circle, but remember that this is a projection of a three-dimensional space onto two dimensions, and the lightness axis L has been collapsed. Those seemingly nearby points have $L \approx 0.5$–0.9, while the new color sits at $L \approx 0.17$; in full 3D Oklab, they are far away. The right panel helps clarify this: it plots lightness L against chroma $C = \sqrt{a^2 + b^2}$, and here you can see that between pure black at $L = 0$ and the first cluster of dark colors around $L \approx 0.3$, there is a conspicuous vertical gap with the star in the middle of it. The hue information ($h = \arctan(b/a) \approx 319°$, a dark magenta-purple) is not shown in either panel, which is the remaining third dimension; the identified color is the right hue to balance its distance from the dark navy ($h \approx 260°$) and dark teal ($h \approx 180°$) neighbors. So in short: the left panel shows where in chromaticity, the right panel shows where in lightness and saturation, and neither panel alone tells the full story. The gap is real, it's just oriented mostly along the L axis with a moderate hue offset, which only becomes obvious when you consider all three dimensions together.

Figure 6 is a scatter plot showing the point in Oklab space farthest from all crayon colors.

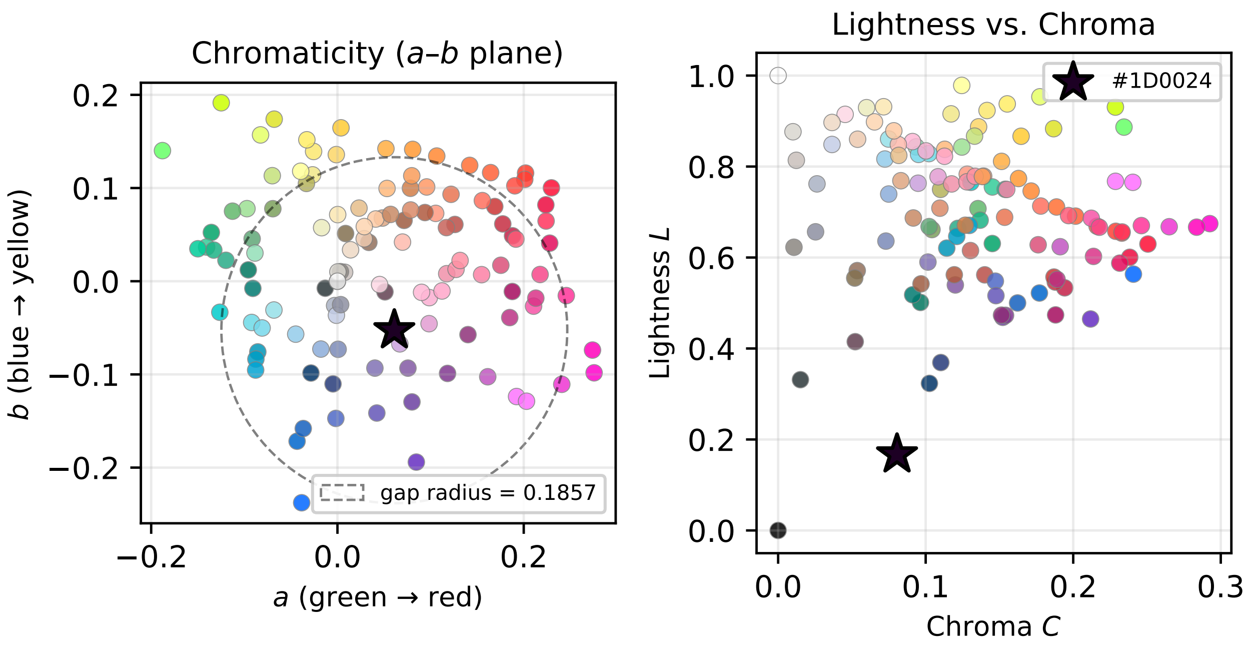


Figure 5: The most underrepresented color region is a dark purplish color

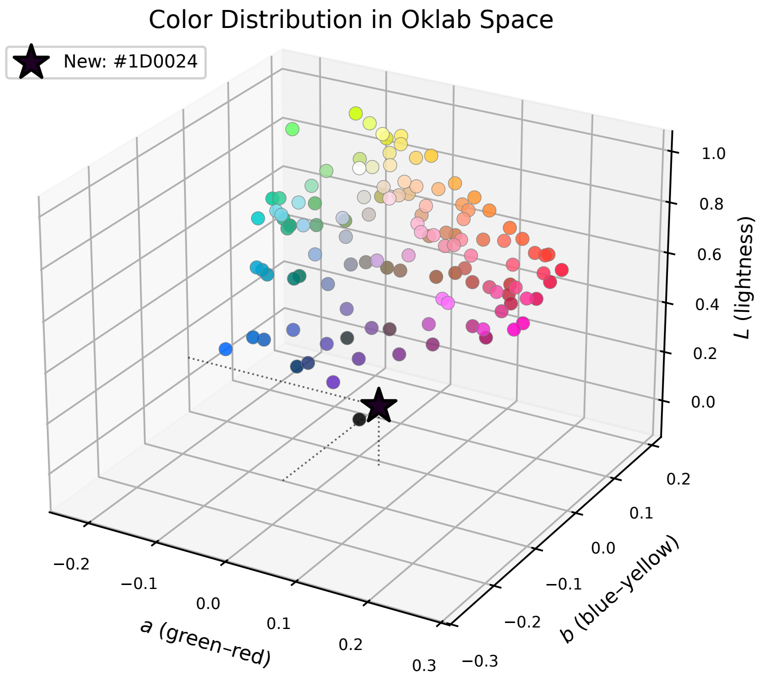
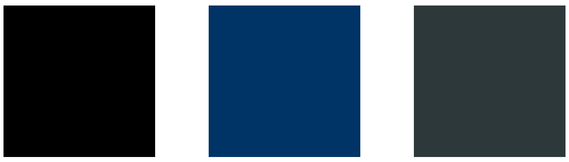
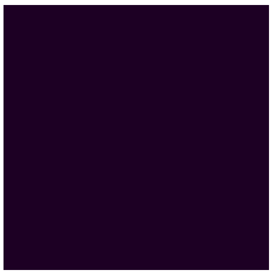


Figure 6: The most underrepresented color region in 3D Oklab space

This missing color is equidistant from these colors in the crayon set which Crayola calls "Black", "Midnight Blue", and "Outer Space":



Finally, we can say that the center of the most underrepresented color region is color #1D0024, a deep, deep magenta that may be difficult to distinguish from black on some computer monitors:



8. Conclusion

The largest void in Crayola's 120-color palette, as measured by the center of the biggest empty sphere in perceptually uniform Oklab space, is #1D0024, a deep, near-black magenta-purple sitting at L ≈ 0.17. The gap radius of 0.186 Oklab units is substantial, meaning this underserved region is genuinely distant from its nearest neighbors in all three perceptual dimensions simultaneously. The void is primarily due to a gap in lightness coverage: the crayon colors jump fairly abruptly from black to a cluster of dark-but-discernible hues. The dark magenta-purple corner of that transition is particularly bare, especially when viewed on a computer monitor that can barely display the full gamut of sRGB color space. The color difference might be more discernible with real wax in physical reality.

Color theory reminds us that darkness is not the absence of color, but is perceptually close to it. Crayola's palette of 120 colors reflects our human bias to fill the page with the colors we celebrate and and leave the shadows to take care of themselves.

Wax crayons are artistic tools of expression for children and artists, and the very-dark end of the spectrum is of limited usefulness on a white page. In a sense, we may have found the missing color that no one misses.

A. LLSS Algorithm

```python
# Uses Least Log Squared method developed by Scott Allen Burns.
# Adapted from http://scottburns.us/matlab-octave-and-python-source-code-for-refl-recon-chrom-adapt
# Data from https://cielab.xyz/pdf/cie.15.2004%20colorimetry.pdf

import numpy as np

# CIE 1931 2° observer, −380730 nm, 10 nm steps (36 values)
_x_bar = np.array([
    0.001368, 0.004243, 0.014310, 0.043510, 0.134380, 0.283900,
    0.348280, 0.336200, 0.290800, 0.195360, 0.095640, 0.032010,
    0.004900, 0.009300, 0.063270, 0.165500, 0.290400, 0.433449,
    0.594500, 0.762100, 0.916300, 1.026300, 1.062200, 1.002600,
    0.854449, 0.642400, 0.447900, 0.283500, 0.164900, 0.087400,
    0.046770, 0.022700, 0.011359, 0.005790, 0.002899, 0.001440
])
_y_bar = np.array([
    0.000039, 0.000120, 0.000396, 0.001210, 0.004000, 0.011600,
    0.023000, 0.038000, 0.060000, 0.090980, 0.139020, 0.208020,
    0.323000, 0.503000, 0.710000, 0.862000, 0.954000, 0.994950,
    0.995000, 0.952000, 0.870000, 0.757000, 0.631000, 0.503000,
    0.381000, 0.265000, 0.175000, 0.107000, 0.061000, 0.032000,
    0.017000, 0.008210, 0.004102, 0.002091, 0.001047, 0.000520
])
_z_bar = np.array([
    0.006450, 0.020050, 0.067850, 0.207400, 0.645600, 1.385600,
    1.747060, 1.772110, 1.669200, 1.287640, 0.812950, 0.465180,
    0.272000, 0.158200, 0.078250, 0.042160, 0.020300, 0.008750,
    0.003900, 0.002100, 0.001650, 0.001100, 0.000800, 0.000340,
    0.000190, 0.000050, 0.000020, 0.000000, 0.000000, 0.000000,
    0.000000, 0.000000, 0.000000, 0.000000, 0.000000, 0.000000
])
# D65 illuminant SPD, −380730 nm, 10 nm steps
_d65 = np.array([
    49.9755, 54.6482, 82.7549, 91.4860, 93.4318, 86.6823,
    104.865, 117.008, 117.812, 114.861, 115.923, 108.811,
    109.354, 107.802, 104.790, 107.689, 104.405, 104.046,
    100.000,  96.3342, 95.7880, 88.6856, 90.0062, 89.5991,
    87.6987, 83.2886, 83.6992, 80.0268, 80.2146, 82.2778,
    78.2842, 69.7213, 71.6091, 74.3490, 61.6040, 69.8856
])

def _build_T():
    """Build the 3×36 matrix that converts reflectance to D65-adapted linear sRGB."""
    delta_lambda = 10.0
    k = np.sum(_y_bar * _d65 * delta_lambda)
    # Weighted color matching functions (each is 36-element row)
    X = _x_bar * _d65 * delta_lambda / k
    Y = _y_bar * _d65 * delta_lambda / k
    Z = _z_bar * _d65 * delta_lambda / k
    XYZ = np.vstack([X, Y, Z])  # 3×36
    # XYZ to linear sRGB matrix (D65 reference)
    M = np.array([
        [ 3.2404542, -1.5371385, -0.4985314],
        [-0.9692660,  1.8760108,  0.0415560],
        [ 0.0556434, -0.2040259,  1.0572252]
    ])
    return M @ XYZ  # 3×36

T = _build_T()


def srgb_to_linear(srgb):
    """Convert sRGB –(0255) to linear RGB –(01)."""
    c = np.asarray(srgb, dtype=float) / 255.0
    return np.where(c < 0.04045, c / 12.92, ((c + 0.055) / 1.055) ** 2.4)


def linear_to_srgb(linear):
    """Convert linear RGB –(01) to sRGB –(0255)."""
    c = np.clip(linear, 0, 1)
    c = np.where(c <= 0.0031308, 12.92 * c, 1.055 * c ** (1.0 / 2.4) - 0.055)
    return np.clip(np.round(c * 255), 0, 255).astype(int)


def reflectance_to_srgb(rho):
    """Convert a 36-element reflectance to sRGB –(0255)."""
    return linear_to_srgb(T @ rho)


def srgb_to_reflectance(srgb_color):
    """
    LLSS algorithm (Scott Burns): recover a 36-point reflectance curve
    from an sRGB triplet –(0255 per channel).
    """
    srgb_color = np.asarray(srgb_color, dtype=float)

    if np.all(srgb_color == 0):
        return np.full(36, 0.0001)

    rgb = srgb_to_linear(srgb_color)

    # Tridiagonal smoothness matrix
    D = np.zeros((36, 36))
    for i in range(36):
        D[i, i] = 4.0
        if i > 0:
            D[i, i - 1] = -2.0
        if i < 35:
            D[i, i + 1] = -2.0
    D[0, 0] = 2.0
    D[35, 35] = 2.0
    z = np.zeros(36)
    lam = np.zeros(3)

    for iteration in range(100):
        r = np.exp(z)
        Tt_lam = T.T @ lam          # 36×1
        v = -r * Tt_lam              # element-wise, 36×1
        m2 = -T * r[np.newaxis, :]   # 3×36

        F = np.concatenate([D @ z + v, -T @ r + rgb])

        J = np.block([
            [D + np.diag(-Tt_lam * r), m2.T],
            [m2, np.zeros((3, 3))]
        ])

        # Use lstsq for numerical robustness (equivalent to MATLAB's \)
        delta, _, _, _ = np.linalg.lstsq(J, -F, rcond=None)

        z += delta[:36]
        lam += delta[36:]

        if np.max(np.abs(F)) < 1e-10 and np.max(np.abs(delta)) < 1e-10:
            break

    return np.exp(z)


def mix_reflectances(reflectances, weights=None):
    """
    Subtractive mixing via weighted geometric mean of reflectances
    (weighted average in log-reflectance space).
    """
    reflectances = np.array(reflectances)
    if weights is None:
        weights = np.ones(len(reflectances)) / len(reflectances)
    weights = np.asarray(weights, dtype=float)
    weights /= weights.sum()
    log_mix = weights @ np.log(reflectances)
    return np.exp(log_mix)


if __name__ == "__main__":
    blue = [0, 50, 200]
    yellow = [255, 230, 20]

    rho_blue = srgb_to_reflectance(blue)
    rho_yellow = srgb_to_reflectance(yellow)

    rho_mix = mix_reflectances([rho_blue, rho_yellow], weights=[0.5, 0.5])

    result = reflectance_to_srgb(rho_mix)
    print(f"Blue:   {blue}")
    print(f"Yellow: {yellow}")
    print(f"Mixed:  {list(result)}")

    # Verify round-trip
    print(f"Blue  round-trip: {list(reflectance_to_srgb(rho_blue))}")
    print(f"Yellow round-trip: {list(reflectance_to_srgb(rho_yellow))}")
```