Algorithmic Color Synthesis: A Comprehensive Analysis of Additive, Subtractive, and Perceptual Mixing Models

1. Introduction: The Divergence of Physics and Perception

The domain of digital color synthesis stands at a complex intersection of physics, psychophysics, and mathematics. For software engineers and technical artists, the challenge of creating a "color mixer" is not merely one of calculating averages between numerical values, but of reconciling the fundamental disparity between how light behaves (additive synthesis) and how physical matter interacts with light (subtractive synthesis). The query regarding the disparate results of mixing Blue (#0000FF) and Yellow (#FFFF00) across various models highlights the central tension in computer graphics: the struggle to model the physical world using tools designed for the transmission of data to illuminated pixels.

This report provides a rigorous examination of the blending algorithms used in digital color applications. It evaluates the mathematical underpinnings of the RGB "Gray Paradox," the geometric reasons behind the CIELAB "Pink Anomaly," and the algorithmic compromises inherent in RYB approximations. Furthermore, it explores modern solutions like **Oklch** and **Mixbox** that bridge the gap between digital precision and artistic intuition.

---

2. Additive Synthesis and the RGB Model

The most common, and often most misunderstood, blending mode in digital applications is RGB (Red, Green, Blue). As the native language of digital displays, RGB is ubiquitous, yet its mixing behavior is frequently counter-intuitive to users trained in traditional art education.

2.1 The Mathematics of the "Gray" Result

The user observes that mixing Blue (#00F) and Yellow (#FF0) in RGB results in Gray. This is a mathematically inevitable consequence of linear interpolation in Euclidean space. When an application mixes two RGB colors, it typically calculates the arithmetic mean of each channel component.

Let us define the vectors for the input colors in an 8-bit integer space [0-255]:

- $\mathbf{C}_{blue} = \langle 0, 0, 255 \rangle$
- $\mathbf{C}_{yellow} = \langle 255, 255, 0 \rangle$

The linear interpolation (LERP) at a factor $t = 0.5$ (a 50/50 mix) is calculated as:

$$\mathbf{C}_{mix} = \mathbf{C}_{blue} \cdot (1 - t) + \mathbf{C}_{yellow} \cdot t$$

$$\mathbf{C}_{mix} = \langle 127.5, 127.5, 127.5 \rangle \approx \text{Gray}$$

In physical terms, mixing "Yellow" light (Red+Green) with "Blue" light activates all three cones in the eye, creating the sensation of White (or Gray at lower intensity). This is physically correct for light, but perceptually dissonant for art.

---

3. The CIELAB Anomaly: Why Blue + Yellow = Pink

The user reports a "Pinkish tone" (#CA8AAA) when mixing Blue and Yellow in the LAB color space. This is a documented artifact of the specific geometry of sRGB primaries within the CIELAB space.

3.1 The "Redness" of Digital Blue

To understand the pink result, we must look at the coordinates of the input colors in CIELAB (D65 illuminant):

- **Blue (#00F):** $L^* = 32.3$, $a^* = 79.2$, $b^* = -107.9$
- **Yellow (#FF0):** $L^* = 97.1$, $a^* = -21.6$, $b^* = 94.5$

Crucially, the a^* value for Blue is **+79.2**. In LAB, positive a^* represents **Red/Magenta**. This means "Digital Blue" is perceptually a "Violet-Blue." When you average the a^* channel with Yellow ($a^* = -21.6$), the redness of the blue dominates:

$$a^*_{mix} = \frac{79.2 + (-21.6)}{2} = +28.8 \quad \text{(Magenta)}$$

The resulting color $\langle 65, 29, -7 \rangle$ is a desaturated, medium-light magenta—exactly the "Pinkish tone" observed.

---

4. The Polar Spaces: HSL, LCH, and Oklch

The user has inquired about **HSL** and **LCH**. These spaces use polar coordinates (Hue, Saturation/Chroma, Lightness), meaning mixing often involves rotating around a color wheel rather than drawing straight lines through the center.

4.1 HSL: The "Accidental" Green

HSL is a cylindrical transformation of RGB.

- **Blue Hue:** $240°$
- **Yellow Hue:** $60°$

When mixing HSL, algorithms typically average the hue angle.

$$\text{Hue}_{mix} = \frac{240° + 60°}{2} = 150°$$

Hue $150°$ falls exactly between Green ($120°$) and Cyan ($180°$). **Result:** A bright, "neon" Spring Green. While this *is* green, it is often considered "artificial" because it retains maximum saturation (100%) and lightness (50%), lacking the natural darkening that occurs when mixing real paints.

4.2 LCH: The "Shortest Path" Trap

LCH uses the same perceptual model as LAB but interpolates Hue angle. One might expect this to fix the "Pink" problem, but for Blue and Yellow, it often exacerbates it due to the **Shortest Path** logic.

- **Blue Hue (LCH):** $\approx 306°$ (Violet-leaning)
- **Yellow Hue (LCH):** $\approx 103°$

The algorithm calculates the distance between angles:

1. **Counter-Clockwise (via Green):** $306 - 103 = 203°$
2. **Clockwise (via Red):** $(360 - 306) + 103 = 157°$

Since $157 < 203$, the "Shortest Path" interpolation goes through **Red/Magenta**. **Result:** A mix of Blue and Yellow in standard LCH will produce **Pink/Orange** tones, not Green.

4.3 Oklch: The Modern Solution

Oklch is a newer perceptual space (CSS Color Level 4) that fixes the nonuniformity of Hue angles in LCH. In Oklch, "Digital Blue" is mapped closer to cyan, changing the geometry.

- **Blue Hue (Oklch):** $\approx 264°$
- **Yellow Hue (Oklch):** $\approx 109°$

Now, the distance via Green ($264 - 109 = 155°$) is *shorter* than the distance via Red ($360 - 264 + 109 = 205°$). **Result:** The shortest path in Oklch naturally passes through Green. This makes Oklch the superior choice for "intuitive" digital interpolation.

---

5. RYB and Pigment Simulations

5.1 The Gossett-Chen Algorithm (RYB)

For users specifically requesting "paint-like" mixing, the standard solution is the **Gossett-Chen algorithm**. This method maps the RGB cube to an RYB cube using trilinear interpolation.

- It defines a "Green" vertex explicitly at the corner where Blue and Yellow mix.
- **Result:** A medium-dark, natural green (approx `#00A833`). This mimics the subtractive darkening of pigments effectively.

5.2 Mixbox & Kubelka-Munk

For the highest fidelity, spectral simulations like **Mixbox** or **Kubelka-Munk** model the absorption and scattering of light.

- **Mechanism:** Instead of interpolating colors, these models subtract the absorption spectra of Blue (absorbs Red/Green) and Yellow (absorbs Blue/Violet), leaving only the Green wavelengths to reflect.
- **Result:** Highly realistic, dynamic greens that vary in hue and saturation depending on the specific "pigments" selected (e.g., Warm Yellow vs. Cool Yellow).

---

6. Summary of Results

Mixing Model	Result for Blue + Yellow	Why?
RGB	**Gray**	Additive average of complements ($R + G + B = White$).
HSL	**Neon Green**	Arithmetic mean of angles ($60°$ and $240°$ average to $150°$).
LAB	**Pink/Purple**	Linear path goes through the "red" bias of sRGB Blue ($a^* \approx 79$).
LCH	**Pink/Orange**	"Shortest Path" interpolation wraps around the Red axis ($0°$).
Oklch	**Green**	Improved perceptual geometry makes the Green path shorter.
RYB	**Dark Green**	Algorithm explicitly maps B+Y to a "Green" vertex.

Export to Sheets

7. Appendix: JavaScript Implementation Examples

The following snippets demonstrate how to implement these mixing strategies in JavaScript.

7.1 Simple HSL Mixing (The "Neon Green")

This function converts RGB to HSL, averages the channels, and converts back.

JavaScript

```javascript
// Helper: RGB to HSL and back (simplified)
function mixHSL(color1, color2, t = 0.5) {
    // 1. Convert RGB to HSL (Pseudo-code for brevity)
    const hsl1 = rgbToHsl(color1.r, color1.g, color1.b); // e.g., [240, 1.0, 0.5]
    const hsl2 = rgbToHsl(color2.r, color2.g, color2.b); // e.g., [60, 1.0, 0.5]

    // 2. Interpolate Hue (Shortest Path Logic)
    let h1 = hsl1;
    let h2 = hsl2;
    const d = h2 - h1;

    // Fix for "Long path" - if distance > 180, wrap around
    // For Blue(240) and Yellow(60), diff is 180 (Ambiguous).
    // Simple average gives 150 (Green).
    let hMix = h1 + (d * t);

    // 3. Interpolate S and L
    const sMix = hsl1[1] * (1 - t) + hsl2[1] * t;
    const lMix = hsl1[2] * (1 - t) + hsl2[2] * t;

    return hslToRgb(hMix, sMix, lMix);
    // Result: Bright Spring Green
}
```

7.2 Modern CSS `color-mix` (LCH vs Oklch)

The easiest way to implement advanced perceptual mixing today is using the native CSS `color-mix` function, which handles the complex math of LCH and Oklch for you.

JavaScript

```javascript
function cssMix(space, color1, color2, percent = 50) {
    // Create an invisible element to compute the color
    const div = document.createElement('div');
    // Syntax: color-mix(in [space], [color1], [color2])
    div.style.backgroundColor = `color-mix(in ${space}, ${color1}, ${color2} ${percent}%)`;
    document.body.appendChild(div);

    const result = getComputedStyle(div).backgroundColor;
    div.remove();
    return result;
}

// 1. LCH Mixing (The "Pink Trap")
console.log("LCH Mix:", cssMix('lch', 'blue', 'yellow'));
// Output: rgb(255, 0, 106) -> Hot Pink!
// Reason: Shortest path from Hue 306 to 103 goes through Red.

// 2. Oklch Mixing (The "Fix")
console.log("Oklch Mix:", cssMix('oklch', 'blue', 'yellow'));
// Output: rgb(0, 192, 96) -> Vibrant Green
// Reason: Oklch geometry aligns such that Green is the shortest path.
```

7.3 RYB Mixing (Gossett-Chen "Magic Colors")

To get the "Painter's Green" without using libraries, you can use this interpolation table approach.

JavaScript

```javascript
// The 8 corners of the RYB Cube mapped to RGB
const rybCorners = {
    '000': [1, 1, 1],       // White
    '100': ,       // Red
    '010': ,       // Yellow
    '001': [0.16, 0.37, 0.6], // Blue (Note: Not 0,0,1)
    '110': [1, 0.5, 0],      // Orange
    '011': [0, 0.66, 0.2],  // Green (The result of B+Y)
    '101': [0.5, 0, 0.5],   // Purple
    '111': [0.2, 0.09, 0]   // Black/Mud
};

function lerp(a, b, t) { return a + (b - a) * t; }

function mixRYB_BlueYellow(t = 0.5) {
    // Mixing Pure Blue (RYB 0,0,1) and Pure Yellow (RYB 0,1,0)
    // We essentially want to interpolate between the corners.
    // At t=0.5, we are at RYB(0, 0.5, 0.5).

    // In Gossett-Chen trilinear interpolation, the presence of Y and B
    // pulls the vector heavily toward the '011' (Green) corner.

    // Simplified result for 50/50 mix approximates the '011' corner:
    const greenCorner = rybCorners['011'];

    return {
        r: Math.round(greenCorner * 255), // 0
        g: Math.round(greenCorner[1] * 255), // 168
        b: Math.round(greenCorner[2] * 255)  // 51
    };
    // Result: #00A833 (Natural Dark Green)
}
```