Here is an example of a revised Adaptive Oklab JavaScript library that “bakes in” a fixed gamma (nonlinearity) for a given surround while preserving hue uniformity. In this design, you choose one of three surround conditions (“white”, “gray”, or “dark”), each with its own gamma value. The library applies the sign-preserving power function uniformly to the LMS channels and then applies a correction factor to the opponent (a and b) channels so that the hue (the angle in the a–b plane) stays constant over variations in lightness and chroma.

In this example the gamma values are as follows:

- **White Surround:** gamma = 2.3
- **Gray Surround:** gamma = 3.0
- **Dark Surround:** gamma = 3.6

Because the original Oklab matrices were optimized for the gray surround (gamma 3.0, or exponent = 1/3), we use that as our standard. For a given surround condition, the nonlinearity exponent is defined as the reciprocal of its gamma (e.g. exponent = 1/γ). A correction factor is then computed as:

correction=x0(13−1γ)\text{correction} = x_0^{\left(\frac{1}{3} - \frac{1}{\gamma}\right)}

where x0x_0 is a representative LMS value (chosen here as 0.5). This factor is applied only to the a and b channels to preserve hue.

Below is the complete library code:

---

```
// adaptiveOklab.js

export class AdaptiveOklab {
  /**
   * Creates an instance of AdaptiveOklab.
   *
   * @param {Object} options - Configuration options.
   * @param {string} options.surround - One of "white", "gray", or "dark".
Default is "gray".
   * @param {number} [options.x0=0.5] - Representative LMS value used for
computing the correction factor.
   */
  constructor(options = {}) {
    // Choose the surround condition: "white", "gray", or "dark"
    this.surround = options.surround || 'gray';
    // Define gamma values for different surround conditions.
    // Gamma is the value used in the power-law nonlinearity.
    this.surroundGammas = {
      white: 2.3,
      gray:  3.0,
      dark:  3.6
    };

    // For a given surround, the exponent is 1 / gamma.
    this.gamma = this.surroundGammas[this.surround] || 3.0;
    this.exponent = 1 / this.gamma;

    // A representative LMS value used to compute a correction factor.
    // This value should be chosen to represent a typical cone response.
    this.x0 = options.x0 || 0.5;

    // Matrix for converting from XYZ to LMS (this remains constant).
    this.M_XYZ_to_LMS = [
      [ 0.8189330101,  0.3618667424, -0.1288597137],
      [ 0.0329845436,  0.9293118715,  0.0361456387],
      [ 0.0482003018,  0.2643662691,  0.6338517070]
    ];

    // Matrix for converting from nonlinearly compressed LMS to Oklab.
    // These coefficients were originally optimized for a gamma of 3.0
(exponent = 1/3).
    this.M_LMS_to_Oklab = [
      [ 0.2104542553,  0.7936177850, -0.0040720468],
      [ 1.9779984951, -2.4285922050,  0.4505937099],
      [ 0.0259040371,  0.7827717662, -0.8086757660]
    ];
  }

  /**
   * Applies a sign-preserving power function.
   *
   * @param {number} value - Input value.
   * @param {number} exponent - Exponent to use.
   * @returns {number} Result after power function.
   */
  signPreservingPow(value, exponent) {
    return Math.sign(value) * Math.pow(Math.abs(value), exponent);
  }

  /**
```

```
   * Multiplies a 3x3 matrix by a 3x1 vector.
   *
   * @param {Array<Array<number>>} matrix - 3x3 matrix.
   * @param {Array<number>} vector - 3-element vector.
   * @returns {Array<number>} Resulting 3-element vector.
   */
  multiplyMatrixVector(matrix, vector) {
    return matrix.map(row =>
      row.reduce((acc, cur, i) => acc + cur * vector[i], 0)
    );
  }

  /**
   * Computes the correction factor for the opponent channels (a and b)
   * to preserve hue uniformity when using a gamma different from the
standard.
   *
   * The standard exponent (for gray surround) is 1/3.
   *
   * @returns {number} Correction factor.
   */
  getCorrectionFactor() {
    const standardExponent = 1 / 3.0;  // exponent for gray surround
    return Math.pow(this.x0, standardExponent - this.exponent);
  }

  /**
   * Converts from XYZ to Adaptive Oklab.
   *
   * @param {number} X - X component of XYZ.
   * @param {number} Y - Y component of XYZ.
   * @param {number} Z - Z component of XYZ.
   * @returns {Object} Object with L, a, and b components.
   */
  fromXYZ(X, Y, Z) {
    // 1. Convert from XYZ to LMS.
    const LMS = this.multiplyMatrixVector(this.M_XYZ_to_LMS, [X, Y, Z]);

    // 2. Apply the adaptive power-law nonlinearity with the fixed exponent.
    const LMSprime = LMS.map(channel => this.signPreservingPow(channel,
this.exponent));

    // 3. Convert from nonlinear LMS to Oklab.
    let lab = this.multiplyMatrixVector(this.M_LMS_to_Oklab, LMSprime);

    // 4. Apply the correction factor to the opponent channels (a and b)
    //    to preserve the hue (i.e. the angular relationship in the a–b
plane).
    const correction = this.getCorrectionFactor();
    lab[1] *= correction;
    lab[2] *= correction;

    return { L: lab[0], a: lab[1], b: lab[2] };
  }

  /**
   * Converts from sRGB to Adaptive Oklab.
   * Assumes sRGB values are in the [0, 1] range.
   *
   * @param {number} r - Red channel.
   * @param {number} g - Green channel.
   * @param {number} b - Blue channel.
   * @returns {Object} Object with L, a, and b components.
   */
  fromSRGB(r, g, b) {
    // Convert sRGB to linear sRGB.
    const linearize = (v) =>
      v <= 0.04045 ? v / 12.92 : Math.pow((v + 0.055) / 1.055, 2.4);
    const rLin = linearize(r);
    const gLin = linearize(g);
    const bLin = linearize(b);

    // Convert linear sRGB to XYZ using a standard transformation matrix.
    const X = 0.4124564 * rLin + 0.3575761 * gLin + 0.1804375 * bLin;
    const Y = 0.2126729 * rLin + 0.7151522 * gLin + 0.0721750 * bLin;
    const Z = 0.0193339 * rLin + 0.1191920 * gLin + 0.9503041 * bLin;

    return this.fromXYZ(X, Y, Z);
  }
}
```

---

## Usage Example

```
import { AdaptiveOklab } from './adaptiveOklab.js';

// Create an AdaptiveOklab instance for a white surround (gamma 2.3).
const aok = new AdaptiveOklab({ surround: 'white' });

// Convert an sRGB color (values in the range [0, 1]) to Adaptive Oklab.
```

```
const lab = aok.fromSRGB(0.2, 0.5, 0.8);
console.log(`L: ${lab.L}, a: ${lab.a}, b: ${lab.b}`);
```

---

## Explanation

1. **Fixed Gamma per Surround:**
   The library is parameterized by a surround condition that sets a fixed gamma value (2.3 for white, 3.0 for gray, and 3.6 for dark). The corresponding exponent for the power-law nonlinearity is computed as 1/gamma1/\text{gamma}.
2. **Uniform Nonlinearity & Hue Preservation:**
   The same power function is applied to every LMS channel. Then, a correction factor—computed based on the difference between the standard exponent (1/3 for gray) and the chosen exponent—is applied uniformly to the a and b channels. This adjustment preserves the hue (angular relationship) even if the overall lightness or chroma shifts.
3. **Simple Pipeline:**
   The pipeline remains as straightforward as the original Oklab:
   - Convert XYZ to LMS,
   - Apply a fixed nonlinearity,
   - Transform LMS to Oklab,
   - Apply a correction to preserve hue.

This revised library provides a self-contained Adaptive Oklab implementation with hue-preserving functionality tuned to a chosen surround condition.