This is a proposal for a JavaScript library—call it **AdaptiveOklab** (or “aoklab”)—that implements an adaptive version of Oklab. This library builds on the classic Oklab pipeline but allows you to adjust the nonlinearity exponent based on surround conditions (inspired by CIECAM16 adaptation parameters), while keeping the hue uniformity intact.

Library Overview

Key Features:

- **Adaptive Exponent:**
  Instead of a fixed 1/3 exponent, the library computes a surround-dependent exponent. For example, you might compute: p=13(FLFL0)αp = \frac{1}{3}\left(\frac{F*L}{F*{L0}}\right)^\alpha where FLF*L is the luminance adaptation factor from your current viewing conditions, *FL0F*{L0} is the reference luminance (e.g., dark-to-gray surround), and α\alpha is a tunable parameter.
- **Conversion Pipelines:**
  - From **XYZ** to adaptive Oklab.
  - From **sRGB** (with gamma correction) to adaptive Oklab (via XYZ conversion).
- **Hue Uniformity Preservation:**
  By applying the same adaptive exponent to all LMS channels before the rotation to Oklab, the relative angles in the a–b plane remain unchanged.
- **Extensible & Modular:**
  The API lets users configure surround parameters and easily plug in more sophisticated CIECAM16–based adaptation computations if needed.

Proposed API and Usage

Below is an outline of the library’s API design and sample code.

1. The Core Class

```
// adaptiveOklab.js

export class AdaptiveOklab {
  /**
```

```
   * Creates an instance of AdaptiveOklab.
   *
   * @param {Object} options - Options for surround conditions.
   * @param {number} options.F_L - The current luminance adaptation factor
(default: 1).
   * @param {number} options.referenceF_L - Reference adaptation factor for
the standard (dark-to-gray) surround (default: 1).
   * @param {number} options.alpha - Sensitivity constant for adjusting the
exponent (default: 1).
   */
  constructor(options = {}) {
    this.F_L = options.F_L || 1;
    this.referenceF_L = options.referenceF_L || 1;
    this.alpha = options.alpha || 1;

    // Predefined matrices for conversion.
    // These matrices should be the same as in standard Oklab.
    // Matrix for converting from XYZ to LMS:
    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 LMS' (nonlinearly compressed LMS) to
Oklab:
    this.M_LMS_to_Oklab = [
      [ 0.2104542553,  0.7936177850, -0.0040720468],
      [ 1.9779984951, -2.4285922050,  0.4505937099],
      [ 0.0259040371,  0.7827717662, -0.8086757660]
    ];
  }

  /**
   * Computes the adaptive exponent based on surround conditions.
   * For standard Oklab, p would be 1/3.
   *
   * @returns {number} Adaptive exponent.
   */
  getExponent() {
    return (1 / 3) * Math.pow(this.F_L / this.referenceF_L, this.alpha);
  }

  /**
   * Applies a sign-preserving power-law nonlinearity.
   *
```

```
     * @param {number} value - The input value.
     * @param {number} exponent - The exponent to use.
     * @returns {number} The transformed value.
     */
    signPreservingPow(value, exponent) {
      return Math.sign(value) * Math.pow(Math.abs(value), exponent);
    }

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

    /**
     * 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.
     */
    fromXYZ(X, Y, Z) {
      // 1. Convert from XYZ to LMS.
      const LMS = this.multiplyMatrixVector(this.M_XYZ_to_LMS, [X, Y, Z]);

      // 2. Compute adaptive exponent and apply sign-preserving nonlinearity.
      const p = this.getExponent();
      const LMSprime = LMS.map(channel => this.signPreservingPow(channel, p));

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

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

    /**
     * Converts from sRGB to adaptive Oklab.
     * sRGB is assumed to be 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.
   */
  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 the standard 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);
  }
}
```

2. Example Usage

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

// Create an instance with custom surround parameters.
// For example, if your current adapting luminance factor (F_L) is 0.8, and
you want to use alpha=1.
const aok = new AdaptiveOklab({
  F_L: 0.8,
  referenceF_L: 1.0,
  alpha: 1
});

// Convert an sRGB color (in 0–1 range) to adaptive Oklab.
const colorSRGB = { r: 0.2, g: 0.5, b: 0.8 };
const lab = aok.fromSRGB(colorSRGB.r, colorSRGB.g, colorSRGB.b);
console.log(`Adaptive Oklab: L=${lab.L}, a=${lab.a}, b=${lab.b}`);
```

3. Extending the Library

You can extend the library by:

- **Adding Reverse Conversions:**
  Implement methods to convert from adaptive Oklab back to XYZ or sRGB.
- **Integrating Full CIECAM16 Adaptation:**
  Replace the simple pp computation with a more detailed calculation using CIECAM16 parameters. For instance, provide a helper function to compute the luminance adaptation factor FLF_L from viewing luminance, background luminance, and other parameters.
- **Packaging & Documentation:**
  Package the library as an NPM module with clear documentation, examples, and tests.

Summary

This proposed JavaScript library—**AdaptiveOklab**—offers a flexible and modular way to convert colors from common spaces into an adaptive Oklab space. By parameterizing the exponent with surround conditions (inspired by CIECAM16), it helps ensure that the perceptual uniformity of hues is maintained even when the viewing conditions vary.

This design provides a starting point, and you can further refine and extend the API based on experimental calibration and additional appearance models.