

Unisoc Confidential For hiar

自动背光介绍及配置指导

文档版本
发布日期

V1.4
2020-11-18

版权所有 © 紫光展锐（上海）科技有限公司。保留一切权利。

本文件所含数据和信息都属于紫光展锐（上海）科技有限公司（以下简称紫光展锐）所有的机密信息，紫光展锐保留所有相关权利。本文件仅为信息参考之目的提供，不包含任何明示或默示的知识产权许可，也不表示有任何明示或默示的保证，包括但不限于满足任何特殊目的、不侵权或性能。当您接受这份文件时，即表示您同意本文件中内容和信息属于紫光展锐机密信息，且同意在未获得紫光展锐书面同意前，不使用或复制本文件的整体或部分，也不向任何其他方披露本文件内容。紫光展锐有权在未经事先通知的情况下，在任何时候对本文件做任何修改。紫光展锐对本文件所含数据和信息不做任何保证，在任何情况下，紫光展锐均不负任何与本文件相关的直接或间接的、任何伤害或损失。

请参照交付物中说明文档对紫光展锐交付物进行使用，任何人对紫光展锐交付物的修改、定制化或违反说明文档的指引对紫光展锐交付物进行使用造成的任何损失由其自行承担。紫光展锐交付物中的性能指标、测试结果和参数等，均为在紫光展锐内部研发和测试系统中获得的，仅供参考，若任何人需要对交付物进行商用或量产，需要结合自身的软硬件测试环境进行全面的测试和调试。

Unisoc Confidential For hiar

紫光展锐（上海）科技有限公司



前言

概述

本文档主要介绍 Android 9.0~Android 11.0 自动背光功能的使用与配置，并给出常见问题的原因分析，方便读者能借助本文档去调整自动背光功能。

读者对象

本文档主要适用于 Android 9.0~Android 11.0 版本的自动背光功能开发人员。

该人员应具有以下经验和技能：


- 了解 Android 背光流程。
- 有开发经历，熟悉 Android Framework 层。

缩略语

无

符号约定

在本文中可能出现下列标志，它所代表的含义如下。

符号	说明
 说明	用于突出重要/关键信息、补充信息和小窍门等。 “说明”不是安全警示信息，不涉及人身、设备及环境伤害。

变更信息

文档版本	发布日期	修改说明
V1.0	2020-07-09	第一次正式发布
V1.1	2020-08-06	1. 新增 2.1.3 Android 11.0 参数配置注意事项 2. 修改文档章节结构，优化文档描述

文档版本	发布日期	修改说明
V1.2	2020-08-17	1. 修改 2.1.1.1 节表 2-1 参数取值及描述 2. 增加 2.2.2 节配置描述
V1.3	2020-08-24	1. 修改第 2 章 Android 9.0~Android 11.0 自动背光方案 2. 优化 5.3 标题描述
V1.4	2020-11-18	修改表 2-1 Lux 和 DisplayNit 的映射值及描述

关键字

自动背光、物理映射、简单映射、环境光、显示亮度。

Unisoc Confidential For hiar

目 录

1 概述	1
2 自动背光配置	2
2.1 物理映射	2
2.1.1 配置参数说明	2
2.1.2 配置示例	4
2.1.3 Android 11.0 参数配置注意事项	6
2.2 简单映射	8
2.2.1 配置参数说明	8
2.2.2 配置示例	8
3 自动背光转换过程	10
3.1 自动背光曲线创建	10
3.2 自动背光值获取	10
4 自动背光映射表计算	12
4.1 代码调用	12
4.2 调整示例	14
5 常见问题分析	16
5.1 实际亮度与亮度条不匹配	16
5.2 亮度与预期不符	16
5.3 亮度条不随亮度的变化而改变	17
5.4 Android 高版本使用简单映射	17

图目录

图 3-1 自动背光曲线创建过程.....	10
图 3-2 自动背光值获取过程.....	11

Unisoc Confidential For hiar

表目录

表 1-1 自动背光策略演进	1
表 2-1 Lux 和 DisplayNit 的映射	2
表 2-2 Backlight 和 Nit 的映射	3

Unisoc Confidential For hiar

1 概述

Android 原生自动背光策略演进详情见表 1-1。

表1-1 自动背光策略演进

Android 版本演变	自动背光方案	自动背光原理
Lollipop ~Oreo	简单映射 (Simple Mapping)	自动背光部分是由环境光 (Lux) 直接决定屏幕背光值 (Backlight)。
Android 9.0~ Android 11.0	物理映射 (Physical Mapping)	自动背光部分通过环境光 (Lux) 决定显示亮度 (Nit)，再由显示亮度 (Nit) 决定背光值 (Backlight)。

Android 9.0~Android 11.0 自动背光方案优点如下：

- 根据当前环境光感值的变化自动调节背光亮度。
- 打开自动背光的情况下，亮度进度条也会随着亮度变化而变化。
- 在自动调节背光亮度基础上记录用户的习惯。
即在当前光感强度与对应的背光亮度的基础上，当用户调整背光值时，利用这对光感强度与背光值来重新映射光感强度和背光强度的关系，之后根据新的映射关系去调节自适应背光；这样就能根据用户的使用习惯做出相应的背光调节。
- 强化了自动调节的功能，使手动调节亮度进度条的频率降低。

2 自动背光配置

Android 9.0 之前自动背光策略采用 Simple Mapping（简单映射），背光曲线只需当前环境光和背光等级的对应值进行映射即可。

而 Android 9.0~Android 110.0 的自动背光策略除了 Simple Mapping，还增加了 Physical Mapping（物理映射），其自动背光曲线的映射关系被拆分成两个部分：

- 环境光（Lux）对应的目标显示亮度（DisplayNit）。
- 当前设备不同的背光等级（Backlight）对应的实际显示亮度（Nit）。

2.1 物理映射

2.1.1 配置参数说明

对于物理映射（Physical Mapping），需要在 config.xml 文件中对以下四个参数进行相应的配置：

- config_autoBrightnessLevels
- config_autoBrightnessDisplayValuesNits
- config_screenBrightnessBacklight
- config_screenBrightnessNits

2.1.1.1 环境光对应的目标显示亮度

环境光（Lux）对应的目标显示亮度（DisplayNit）涉及的参数为：

- <integer-array name="config_autoBrightnessLevels"> ---> Lux
- <array name="config_autoBrightnessDisplayValuesNits">--->DisplayNit

Lux 和 DisplayNit 的映射关系详见表 2-1。

表2-1 Lux 和 DisplayNit 的映射

Lux	DisplayNit
0	3
5	20
30	50
100	90
500	140
2000	290

Lux	DisplayNit
5000	360
10000	400
20000	430

这部分的需求可自定义，表 2-1 仅为单个示例，可进行客制化修改。

- Lux 是环境照度的情况，单位为勒克斯，这个值是 Light sensor 输入的，在正午阳光下会超过 10W 勒克斯。
- DisplayNit 是希望的显示亮度，单位为尼特。
 - 当 Lux < 5，DisplayNit 取 3。
 - 当 Lux = 5，DisplayNit 取 20。
 - 当 Lux > 20000，DisplayNit 取 430。

实际代码中，Lux 最小值默认会取 0，不用配置；但 DisplayNit 的最小值需要配置，从而 config_autoBrightnessDisplayValuesNits 的元素数量比 config_autoBrightnessLevels 多 1。

2.1.1.2 当前设备不同的背光等级对应的实际显示亮度

当前设备不同的背光等级（Backlight）对应的实际显示亮度(Nit，单位：尼特)涉及的参数为：

- <integer-array name="config_screenBrightnessBacklight"> ----> Backlight
- <array name="config_screenBrightnessNits"> ----> Nit

Backlight 和 Nit 的映射关系详见表 2-2。

表2-2 Backlight 和 Nit 的映射

Backlight	Nit
5	3.048
20	17.88
40	49.81
65	87.71
100	141.5
200	288.5
255	363.6
255	363.6

这部分是实际的设备的特性，需要测量。

测量时注意事项如下：

- 使用纯白的 bmp 测试 pattern。
- 使用 CA310/CA410 等色彩分析仪测试屏幕的亮度（Nit）。
使用纯白的 pattern，调整背光等级（Backlight），同时使用 CA310 色彩分析仪测试屏幕的亮度（Nit）。
修改不同的背光等级进行逐一测试得到表格。
 - Backlight 列是希望细分的等级，范围 0-255。
 - Nit 列是测量的结果。

2.1.2 配置示例

2.1.2.1 环境光对应的目标显示亮度

config_autoBrightnessDisplayValuesNit 的 Array 元素数量比 config_autoBrightnessLevels 多 1。

其中：

- config_autoBrightnessLevels 第一个元素的值为 16。
- config_autoBrightnessDisplayValuesNits 第一个元素的值为 5。

配置含义为环境光输入不超过 16 勒克斯的情况下，需要的显示亮度是 5 尼特。

config_autoBrightnessLevels 数组配置示例如下：

```
<integer-array name="config_autoBrightnessLevels">
<item>16</item>
<item>32</item>
<item>50</item>
<item>100</item>
<item>140</item>
<item>180</item>
<item>240</item>
<item>300</item>
<item>600</item>
<item>800</item>
<item>1000</item>
<item>2000</item>
<item>3000</item>
<item>4000</item>
<item>5000</item>
<item>6000</item>
<item>8000</item>
<item>10000</item>
<item>20000</item>
</integer-array>
```

config_autoBrightnessDisplayValuesNits 数组配置示例如下：

```
<array name="config_autoBrightnessDisplayValuesNits">
<item>5</item>
<item>20</item>
<item>50</item>
<item>80</item>
<item>141.5</item>
<item>162.2</item>
<item>210.1</item>
<item>210.1</item>
<item>210.1</item>
<item>210.1</item>
<item>210.1</item>
<item>210.1</item>
<item>361.1</item>
<item>400.1</item>
<item>418.5</item>
<item>418.5</item>
<item>418.5</item>
<item>418.5</item>
<item>436.3</item>
<item>500.1</item>
</array>
```

2.1.2.2 当前设备不同的背光等级对应的实际显示亮度

`config_screenBrightnessBacklight` 和 `config_screenBrightnessNits` 两个 Array 的元素数量要一致。

- `config_screenBrightnessBacklight` 第一个参数是 0。
- `config_screenBrightnessNits` 第一个参数是 0.048。

配置含义为当背光等级设定为 0（背光等级的范围是 0~255）的时候，使用的实际显示设备显示的亮度为 0.048 尼特。

`config_screenBrightnessBacklight` 数组配置示例如下：

```
<integer-array name="config_screenBrightnessBacklight">
<item>0</item>
<item>15</item>
<item>30</item>
<item>45</item>
<item>60</item>
<item>75</item>
<item>90</item>
<item>105</item>
<item>120</item>
```

```
<item>135</item>
<item>150</item>
<item>165</item>
<item>180</item>
<item>195</item>
<item>210</item>
<item>225</item>
<item>240</item>
<item>255</item>
</integer-array>
```

config_screenBrightnessNits 数组配置示例如下：

```
<array name="config_screenBrightnessNits">
<item>0.048</item>
<item>23.27</item>
<item>55.26</item>
<item>87.1</item>
<item>118.6</item>
<item>151.9</item>
<item>182.7</item>
<item>212.9</item>
<item>242.9</item>
<item>274.6</item>
<item>303.9</item>
<item>332.7</item>
<item>361.1</item>
<item>390.9</item>
<item>418.5</item>
<item>445.3</item>
<item>471.9</item>
<item>500.1</item>
</array>
```

2.1.3 Android 11.0 参数配置注意事项

在 Android 11.0 中，参数 config_screenBrightnessBacklight 配置的最小值不能为 0，而需要是 1。

因为在 Android 11.0 中，背光值都会归一化转化为 0.0~1.0 之间的小数。如果配置的最小值是 0，经过归一化会变成 -1.0，这样会使自动背光值不准确；而配置值为 1 时，经过归一化之后，对应的就是 0.0，这样自动背光曲线映射关系才正确。

Android 11.0 中的 config_screenBrightnessBacklight 配置示例如下：

```
<integer-array name="config_screenBrightnessBacklight">
<item>1</item>
```

```
<item>15</item>
<item>30</item>
<item>45</item>
<item>60</item>
<item>75</item>
<item>90</item>
<item>105</item>
<item>120</item>
<item>135</item>
<item>150</item>
<item>165</item>
<item>180</item>
<item>195</item>
<item>210</item>
<item>225</item>
<item>240</item>
<item>255</item>
</integer-array>
```

config_screenBrightnessNits 数组配置示例如下：

```
<array name="config_screenBrightnessNits">
<item>0.048</item>
<item>23.27</item>
<item>55.26</item>
<item>87.1</item>
<item>118.6</item>
<item>151.9</item>
<item>182.7</item>
<item>212.9</item>
<item>242.9</item>
<item>274.6</item>
<item>303.9</item>
<item>332.7</item>
<item>361.1</item>
<item>390.9</item>
<item>418.5</item>
<item>445.3</item>
<item>471.9</item>
<item>500.1</item>
</array>
```

2.2 简单映射

2.2.1 配置参数说明

简单映射（Simple Mapping）只需要配置两个参数

- config_autoBrightnessLevels（Lux）
- config_autoBrightnessLcdBacklightValues（Backlight）

config_autoBrightnessLcdBacklightValues 的元素数量比 config_autoBrightnessLevels 多 1。

2.2.2 配置示例

- config_autoBrightnessLevels 第一个元素是 16。
- config_autoBrightnessLcdBacklightValues 第一个元素是 30。

配置含义为环境光输入不超过 16 勒克斯的情况下，需要的背光值为 30（背光等级的范围是 0~255）。

环境光（Lux）参数配置示例如下：

```
<integer-array name="config_autoBrightnessLevels">
<item>16</item>
<item>32</item>
<item>50</item>
<item>100</item>
<item>140</item>
<item>180</item>
<item>240</item>
<item>300</item>
<item>600</item>
<item>1000</item>
<item>2000</item>
<item>3000</item>
<item>4000</item>
<item>8000</item>
<item>10000</item>
</integer-array>
```

背光（Backlight）参数配置示例如下：

```
<integer-array name="config_autoBrightnessLcdBacklightValues">
<item>30</item>
<item>40</item>
<item>50</item>
<item>60</item>
<item>70</item>
<item>80</item>
```

```
<item>102</item>  
<item>102</item>  
<item>102</item>  
<item>102</item>  
<item>102</item>  
<item>180</item>  
<item>200</item>  
<item>210</item>  
<item>230</item>  
<item>255</item>  
</integer-array>
```

Unisoc Confidential For hiar

3 自动背光转换过程

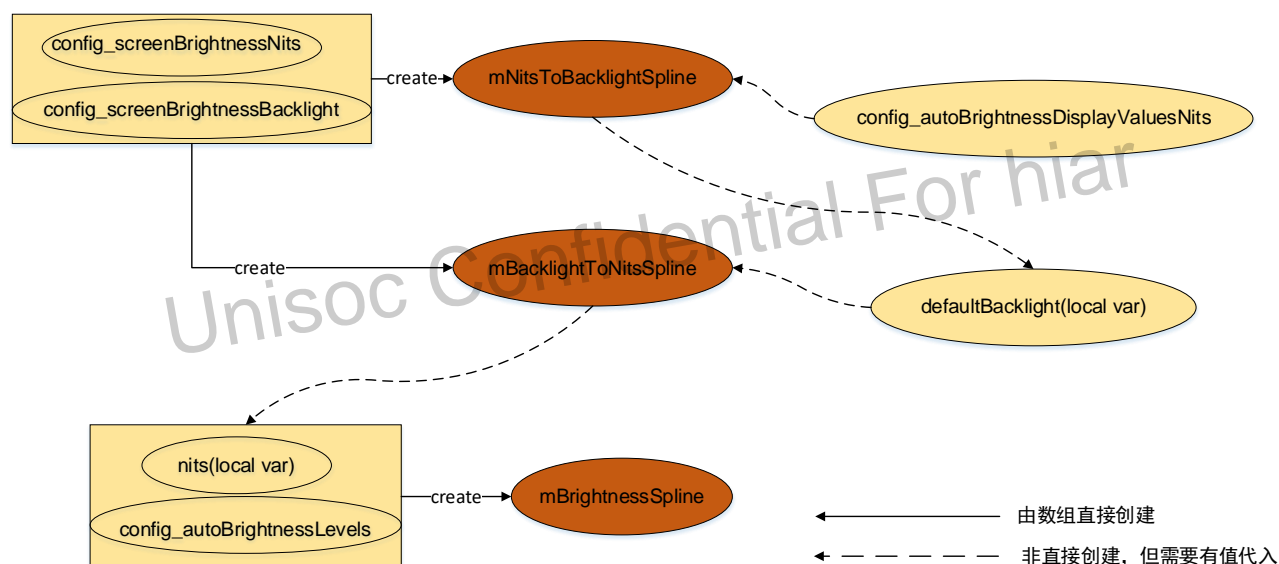
3.1 自动背光曲线创建

在物理映射策略中，根据 `config_screenBrightnessNits` 数组和 `config_screenBrightnessBacklight` 数组，以及 `config_autoBrightnessDisplayValuesNits` 数组和 `config_autoBrightnessLevels` 数组，创建了两条映射曲线：`mNitsToBacklightSpline` 和 `mBacklightToNitsSpline`，及一条可调整曲线：`mBrightnessSpline`。

`mBrightnessSpline` 即自动背光曲线，由 `mNitsToBacklightSpline` 和 `mBacklightToNitsSpline` 构造。

自动背光曲线创建详见图 3-1，详情请参考 `BrightnessMappingStrategy.java` 文件。

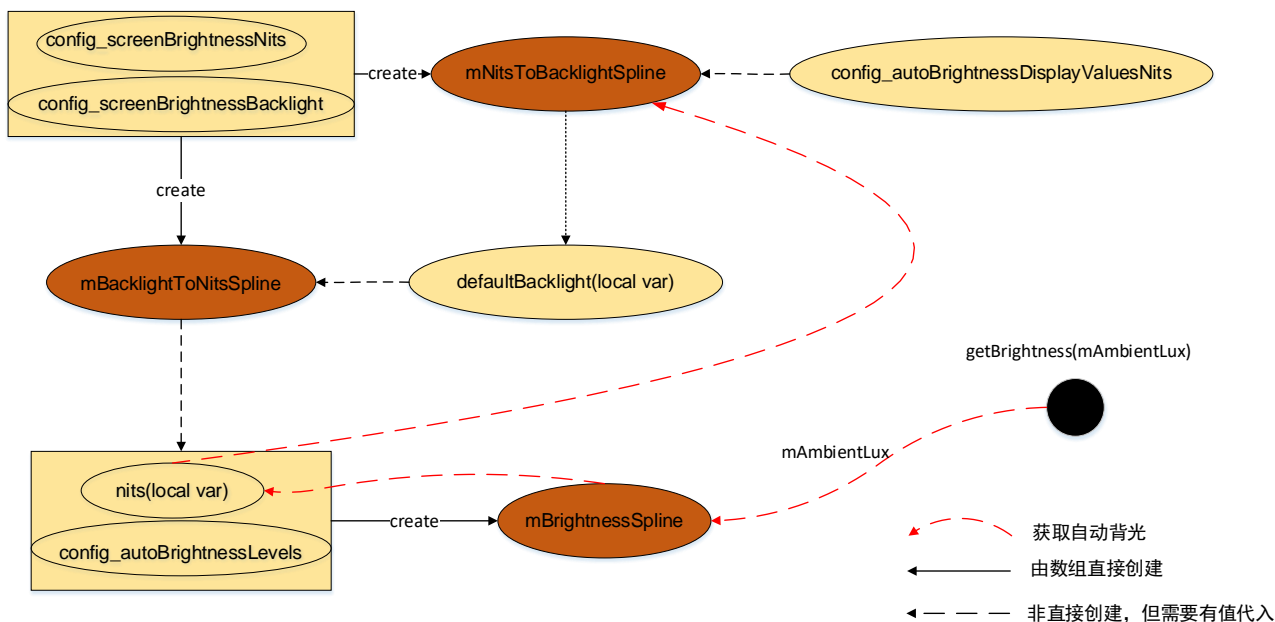
图3-1 自动背光曲线创建过程



3.2 自动背光值获取

获取自动背光值的流程见图 3-2，详情请参考 `AutomaticBrightnessController.java` 和 `BrightnessMappingStrategy.java` 文件。

图3-2 自动背光值获取过程



1. 根据光感 sensor 读取当前环境光强度 Lux 值。
2. 从 mBrightnessSpline 曲线中寻找对应的 Nit 值。
3. 根据 Nit 值，从曲线 mNitsToBacklightSpline 中获取到最终的背光值。

4

自动背光映射表计算

4.1 代码调用

自动背光值计算主要集中在 **BrightnessMappingStrategy** 类中，本章主要描述当用户手动调整亮度后，映射表的计算方法。

在用户手动调整进度条后，类方法调用顺序如下：

1. 调用 **BrightnessMappingStrategy** 中内部类 **PhysicalMappingStrategy** 的 **addUserDataPoint** 方法。
2. 调用 **computeSpline** 方法创建可调整曲线，在 **computeSpline** 中主要调用 **getAdjustedCurve** 重新计算映射表。

getAdjustedCurve()详情如下：

```
protected Pair<float[], float[]> getAdjustedCurve(float[] lux, float[] brightness,
    float userLux, float userBrightness, float adjustment, float maxGamma) {
    float[] newLux = lux;
    float[] newBrightness = Arrays.copyOf(brightness, brightness.length);
    if (mLoggingEnabled) {
        PLOG.logCurve("unadjusted curve", newLux, newBrightness);
    }
    //限定adjustment为[-1,1]
    adjustment = MathUtils.constrain(adjustment, -1, 1);
    //maxGamma 的 -adjustment次方得到gamma值
    float gamma = MathUtils.pow(maxGamma, -adjustment);
    if (mLoggingEnabled) {
        Slog.d(TAG, "getAdjustedCurve: " + maxGamma + "^" + -adjustment + "=" +
            MathUtils.pow(maxGamma, -adjustment) + " == " + gamma);
    }
    //gamma != 1说明adjustment不为0
    if (gamma != 1) {
        //重新设置亮度值，新的亮度值为就亮度值的gamma次方
        for (int i = 0; i < newBrightness.length; i++) {
            newBrightness[i] = MathUtils.pow(newBrightness[i], gamma);
        }
    }
    if (mLoggingEnabled) {
        PLOG.logCurve("gamma adjusted curve", newLux, newBrightness);
    }
}
```

```

    }

    //用户在BrightnessController拖动条上手动调节了亮度
    if (userLux != -1) {
        //插入用户手动调节时的Lux和对应背光值
        Pair<float[], float[]> curve = insertControlPoint(newLux,newBrightness,
                                                         userLux,userBrightness);

        newLux = curve.first;
        newBrightness = curve.second;
        if (mLoggingEnabled) {
            PLOG.logCurve("gamma and user adjusted curve", newLux, newBrightness);
            // This is done for comparison.
            curve = insertControlPoint(lux, brightness, userLux, userBrightness);
            PLOG.logCurve("user adjusted curve", curve.first ,curve.second);
        }
    }

    return Pair.create(newLux, newBrightness);
}

```

3. 调用 `insertControlPoint()` 将用户设置的点作为曲线控制点插入相关数组中。

`insertControlPoint()` 详情如下：

```

private static Pair<float[], float[]> insertControlPoint(float[] luxLevels,
                                                         float[] brightnessLevels, float lux, float brightness) {
    //找到插入点索引，以保持值的递增
    final int idx = findInsertionPoint(luxLevels, lux);
    final float[] newLuxLevels;
    final float[] newBrightnessLevels;
    //如果插入索引等于数组长度，则需要插入到末尾位置，新Lux数组长度比原来大1：
    if (idx == luxLevels.length) {
        newLuxLevels = Arrays.copyOf(luxLevels, luxLevels.length + 1);
        newBrightnessLevels = Arrays.copyOf(brightnessLevels,
                                             brightnessLevels.length + 1);
        newLuxLevels[idx] = lux;
        newBrightnessLevels[idx] = brightness;
        //如果用户lux值已处于Lux值数组中，copy原数组：
    } else if (luxLevels[idx] == lux) {
        .....
    } else {
        //初始化新Lux数组，长度比原数组大1
        newLuxLevels = Arrays.copyOf(luxLevels, luxLevels.length + 1);
        //将原数组从idx位置起的元素，向后移动1位，将新的Lux值插入到idx位置
        System.arraycopy(newLuxLevels, idx, newLuxLevels, idx+1,

```

```

        luxLevels.length - idx);
    newLuxLevels[idx] = lux;
    //初始化新亮度值数组，长度比原数组大1
    newBrightnessLevels = Arrays.copyOf(brightnessLevels,
        brightnessLevels.length + 1);
    //将原数组从idx位置起的元素，向后移动1位，将新的亮度值插入到idx位置
    System.arraycopy(newBrightnessLevels, idx, newBrightnessLevels, idx+1,
        brightnessLevels.length - idx);
    newBrightnessLevels[idx] = brightness;
}

smoothCurve(newLuxLevels, newBrightnessLevels, idx);
return Pair.create(newLuxLevels, newBrightnessLevels);
}

```

上述代码中，当有用户手动调节亮度时：

1. 通过 `getAdjustedCurve()` 中 `adjustment` 值获取 `gamma` 值，再通过 `gamma` 值调整整个背光数组元素，计算出新的背光值。
2. 调用 `insertControlPoint()` 将用户设置的点作为曲线控制点插入相关数组中，插入处理完毕后，返回一个携带有 `Lux` 和 `Backlight` 值的 `Pair` 对象返回给 `getAdjustedCurve()`，即返回到 `computeSpline()` 中。
3. `computeSpline()` 从返回的 `Pair` 对象中取出 `Lux` 数组值和 `Backlight` 数组值，完成新的 `mBrightnessSpline` 曲线创建。

`mBrightnessSpline` 曲线创建方法详见“3.1 自动背光曲线创建”。

4.2 调整示例

系统默认创建的曲线数据：

PhysicalMappingStrategy

```

    mBrightnessSpline=MonotoneCubicSpline([(0.0, 33.07: 1.0458455), (16.0, 49.803528: 0.99861073),
(32.0, 65.02554: 0.89720714), (50.0, 80.20023: 0.5869326), (100.0, 96.74158: 0.35375974), (140.0,
111.80928: 0.5943943), (180.0, 144.29312: 0.0), (240.0, 144.29312: 0.0), (300.0, 144.29312: 0.0),
(600.0, 144.29312: 0.0), (800.0, 144.29312: 0.0), (1000.0, 260.0: 0.085323855), (2000.0, 288.50928:
0.005903368), (3000.0, 302.0: 0.0100962985), (4000.0, 308.70187: 0.006697006), (5000.0, 315.394:
0.010652771), (6000.0, 330.00742: 0.015704848), (8000.0, 363.6: 0.0), (10000.0, 363.6: 0.0)]);

    mNitsToBacklightSpline=MonotoneCubicSpline([(0.0125, 0.0: 0.005311309), (10.39, 0.05511811:
0.003957575), (33.07, 0.114173226: 0.0025150497), (57.41, 0.17322835: 0.0025190804), (80.02,
0.23228346: 0.0025220779), (104.3, 0.2913386: 0.002534322), (126.7, 0.3503937: 0.0025693625), (150.3,
0.40944883: 0.0025933287), (172.3, 0.46850392: 0.002620411), (195.4, 0.52755904: 0.0026516253),
(216.9, 0.5866142: 0.0026741477), (239.6, 0.6456693: 0.0027482028), (260.0, 0.70472443:
0.0027957207), (281.9, 0.7637795: 0.0028173232), (302.0, 0.8228347: 0.0028553037), (323.3,
0.88188976: 0.002877561), (343.1, 0.9409449: 0.0029316582), (363.6, 1.0: 0.002880736)]);
    mMaxGamma=3.0
    mAutoBrightnessAdjustment=0.0

```

```
mUserLux=-1.0
mUserBrightness=-1.0
```

当用户通过调节亮度条将亮度调暗时，插入用户设置点的曲线数据：

```
PhysicalMappingStrategy
    mBrightnessSpline=MonotoneCubicSpline([(0.0, 1.6208489: 0.08439045), (16.0, 2.971096: 0.09619627),
(32.0, 4.6991296: 0.12798336), (50.0, 7.362493: 0.12025933), (100.0, 11.990194: 0.1352344), (140.0,
19.106785: 0.36427373), (180.0, 41.13209: 3.8146976E-7), (210.0, 41.132095: 0.0), (240.0, 41.132095:
0.0), (300.0, 41.132095: 0.0), (600.0, 41.132095: 0.0), (800.0, 41.132095: 0.0), (1000.0, 55.04366:
0.06759873), (2000.0, 120.683304: 0.06519039), (3000.0, 185.42444: 0.06357235), (4000.0, 247.828:
0.04038965), (5000.0, 266.20374: 0.0231288), (6000.0, 294.0856: 0.031319533), (8000.0, 363.6: 0.0),
(10000.0, 363.6: 0.0)])

    mNitsToBacklightSpline=MonotoneCubicSpline([(0.0125, 0.0: 0.005311309), (10.39, 0.05511811:
0.003957575), (33.07, 0.114173226: 0.0025150497), (57.41, 0.17322835: 0.0025190804), (80.02,
0.23228346: 0.0025220779), (104.3, 0.2913386: 0.002534322), (126.7, 0.3503937: 0.0025693625), (150.3,
0.40944883: 0.0025933287), (172.3, 0.46850392: 0.002620411), (195.4, 0.52755904: 0.0026516253),
(216.9, 0.5866142: 0.0026741477), (239.6, 0.6456693: 0.0027482028), (260.0, 0.70472443:
0.0027957207), (281.9, 0.7637795: 0.0028173232), (302.0, 0.8228347: 0.0028553037), (323.3,
0.88188976: 0.002877561), (343.1, 0.9409449: 0.0029316582), (363.6, 1.0: 0.002880736)])

    mMaxGamma=3.0
    mAutoBrightnessAdjustment=-0.7010461
    mUserLux=210.0
    mUserBrightness=0.13387662
```

其中 mBrightnessSpline 曲线的(210.0, 41.132095: 0.0)这组值为用户调节亮度条时的环境光感强度为 mUserLux=210.0，其对应的 Nit=41.132095。

从 mNitsToBacklightSpline 曲线中可见 Nit=41.132095 在 33.07 与 57.41 之间，调节后的背光值 mUserBrightness=0.13387662 正好在 0.114173226 与 0.17322835 之间。

对比上述数据，用户调节的点为(210.0, 41.132095: 0.0)，插入到 mBrightnessSpline 中后，其他点也做了相应调整。

5

常见问题分析

5.1 实际亮度与亮度条不匹配

现象

实际亮度与亮度条不匹配。

原因分析

- Android 9.0 以前版本设置 50% 的亮度，adb shell settings put system screen_brightness 127 即可。
- Android 9.0~Android 11.0 版本中，亮度进度条不再为 $x/255 = n\%$ 的关系，而是由 convertGammaToLinear、convertLinearToGamma 和 GAMMA_SPACE_MAX=1023 组成的新的对应关系决定的。

其中：

- x 是当前屏幕的背光值。
- n 计算出来的亮度条百分比。

Android 9.0~Android 11.0 版本设置 50% 的亮度，adb shell settings put system screen_brightness 127，127 会被 convertLinearToGamma 转换为 891，数值 891 占进度条最大值（GAMMA_SPACE_MAX）的 87%，故进度条显示在 87% 位置。

测试建议

正常现象。

5.2 亮度与预期不符

现象

开启自动背光，亮度不符合预期。

原因分析

可能原因如下：

- 自动背光参数配置不正确。
- 在开启背光的情况下手动调整过亮度。

因为自动背光新的特性会记录用户手动调整的背光值，在暗或亮的环境下，会根据用户的调整重新计算屏幕亮度，可能会出现无法达到预期效果的现象。

测试建议

恢复出厂设置，开启自动背光，重新查看亮度结果。

5.3 亮度条不随亮度的变化而改变

现象

打开自动背光情况下，亮度变化，但是亮度条不发生改变。

原因分析

首先要确认 Android 版本：

- 在 Android 9.0 之前的版本，开启自动背光后，亮度条反映的不再是直接的亮度，而是 **adj** 值，故不会随着亮度的变化而变化。
- Android 9.0~Android 11.0 版本，开启自动背光后，亮度条随着亮度的变化而变化。

测试建议

正常现象。

5.4 Android 高版本使用简单映射

现象

Android 9.0~Android 11.0 使用背光方案为物理映射（Physical Mapping），如果直接使用了简单映射（Simple Mapping）方案，**system** 进程会发生 **crash**，自动背光功能无法使用。

解决方法

在 Android 9.0~Android 11.0 版本中，如果想使用 Simple Mapping，除了需要配置 **config_autoBrightnessLevels**（Lux）和 **config_autoBrightnessLcdBacklightValues**（Backlight）两个数组之外，还需要在 **DisplayPowerController.java** 中合入以下修改：

```
diff --git a/services/core/java/com/android/server/display/DisplayPowerController.java
b/services/core/java/com/android/server/display/DisplayPowerController.java
index 980f9a2..ec3ef01 100644
--- a/services/core/java/com/android/server/display/DisplayPowerController.java
+++ b/services/core/java/com/android/server/display/DisplayPowerController.java
@@ -656,9 +656,9 @@ final class DisplayPowerController implements
AutomaticBrightnessController.Call
    // Initialize all of the brightness tracking state
```



```
final float brightness = convertToNits(mPowerState.getScreenBrightness());  
- if (brightness >= 0.0f) {  
    mBrightnessTracker.start(brightness);  
- }
```

测试建议

请进行相关 CTS 的 case（CTS DisplayTestCases），保证不影响 CTS 测试。

Unisoc Confidential For hiar