

Image

PS 图层混合模式完全解读-讲义

超详细的图层混合模式解读

作者：王先生的副业

组织：王先生的副业

时间：2023 年 6 月 1 日

版本：4.3

目录

第 1 章 前言	1
1.1 泡一杯茶	1
1.2 什么是图层混合模式	1
第 2 章 前置概念	3
2.1 像素点	3
2.2 色彩空间	3
2.2.1 RGB	3
2.2.2 HSY	3
2.3 图层	4
2.3.1 RGB 表示法	4
2.3.2 HSY 表示法	4
2.3.3 涉及图层	4
2.4 像素混合图例	5
2.5 图层大小	5
2.6 映射面	6
2.6.1 映射面示例	7
2.6.2 同图平面	7
2.6.3 同图混合曲线	7
2.6.4 中性灰平面	8
2.6.4.1 基础图层的中性灰平面	8
2.6.4.2 混合图层的中性灰平面	9
2.7 关系总揽	10
2.7.1 特殊关系	10
2.7.2 可转换	10
2.7.3 互逆 (可交换)	10
2.7.4 组成	10
第 3 章 运算相关	13
3.1 归一化操作	13
3.2 一些运算规则	13
3.2.1 通道级别	13
3.2.2 像素级别	13
3.2.3 图层级别	14
3.2.4 HSY 的运算	14
3.3 混合模式相关符号	14
3.4 辅助计算程序	16
3.5 混合模式的组合	16
第 4 章 不透明度和填充	17
4.1 在哪	17
4.2 不透明度和填充公式	18

4.3	单纯混合模式与二者使用的区别	18
4.4	代码实现	18
4.4.1	不透明度	18
4.4.2	填充	19
4.5	不透明度和填充与图层样式	19
第 5 章	图层混合模式的种类	21
5.1	色彩空间分类	21
5.1.1	RGB	21
5.1.2	HSY	21
5.2	是否产生新的数值	21
5.2.1	替换式	21
5.2.2	融合式	21
5.3	是否顺序相关	22
5.3.1	顺序无关	22
5.3.2	顺序相关	22
5.4	是否 fill 和 Opacity 不同	22
5.4.1	fill 和 Opacity 产生不同影响	22
5.4.2	fill 和 Opacity 产生相同影响	22
第 6 章	普通组	25
6.1	正常 Normal	25
6.1.1	公式 (泡茶的方式, 比如搅拌、静置)	25
6.1.2	融合填充 (放多少茶叶)	25
6.1.3	融合不透明度 (太苦了, 最后加点水)	25
6.1.4	整个像素融合填充	25
6.1.5	整个像素融合不透明度	26
6.1.6	程序模拟该模式计算结果	26
6.1.7	验证	27
6.2	溶解 Dissolve	28
6.2.1	溶解模式初始公式	28
6.2.2	如果融合了填充	28
6.2.3	程序模拟该模式计算结果	28
6.2.4	验证	29
6.2.5	用途示例	29
第 7 章	变暗组	31
7.1	变暗模式 Darken	31
7.1.1	公式	31
7.1.2	融合填充	31
7.1.3	融合不透明度	31
7.1.4	映射面和同图等效曲线	31
7.1.5	程序模拟该模式计算结果	32
7.1.6	验证	33
7.2	正片叠底 Multiply	34
7.2.1	公式	34

7.2.2	融合填充	34
7.2.3	融合不透明度	34
7.2.4	映射面和同图等效曲线	34
7.2.5	同图曲线表达式	35
7.2.6	程序模拟该模式计算结果	35
7.2.7	验证	35
7.3	线性加深 LinearBurn	36
7.3.1	公式	36
7.3.2	融合填充	36
7.3.3	融合不透明度	36
7.3.4	映射面和同图等效曲线	36
7.3.5	同图曲线表达式	37
7.3.6	程序模拟该模式计算结果	37
7.3.7	验证	37
7.4	颜色加深 ColorBurn	38
7.4.1	公式	38
7.4.2	融合填充	38
7.4.3	融合不透明度	38
7.4.4	映射面和同图等效曲线	38
7.4.5	同图曲线表达式	38
7.4.6	两次负片转划分	38
7.4.7	程序模拟该模式计算结果	39
7.4.8	验证	39
7.5	深色 Darker	40
7.5.1	公式	40
7.5.2	融合填充	40
7.5.3	融合不透明度	40
7.5.4	映射面和同图等效曲线	40
7.5.5	程序模拟该模式计算结果	41
7.5.6	验证	42
第 8 章	变亮组	43
8.1	变亮 Lighten	43
8.1.1	公式	43
8.1.2	融合填充	43
8.1.3	融合不透明度	43
8.1.4	三次负片操作相互转换	43
8.1.5	映射面和同图等效曲线	44
8.1.6	程序模拟该模式计算结果	44
8.1.7	验证	45
8.2	滤色 Screen	46
8.2.1	公式	46
8.2.2	融合填充	46
8.2.3	融合不透明度	46
8.2.4	三次负片操作相互转换	46

8.2.5 映射面和同图等效曲线	47
8.2.6 程序模拟该模式计算结果	47
8.2.7 验证	48
8.3 线性减淡 LinearDodge	49
8.3.1 公式	49
8.3.2 融合填充	49
8.3.3 融合不透明度	49
8.3.4 三次负片操作相互转换	49
8.3.5 映射面和同图等效曲线	49
8.3.6 程序模拟该模式计算结果	49
8.3.7 验证	50
8.4 颜色减淡 ColorDodge	51
8.4.1 公式	51
8.4.2 融合填充	51
8.4.3 融合不透明度	51
8.4.4 结合负片操作和其他模式互转	51
8.4.4.1 三次负片操作转换为颜色加深	52
8.4.4.2 一次负片转为划分	52
8.4.5 映射面和同图等效曲线	52
8.4.6 程序模拟该模式计算结果	53
8.4.7 验证	53
8.4.8 用途示例	53
8.5 浅色 Lighter	55
8.5.1 公式	55
8.5.2 融合填充	55
8.5.3 融合不透明度	55
8.5.4 三次负片操作相互转换	55
8.5.5 程序模拟该模式计算结果	56
8.5.6 验证	57
第9章 对比度组	59
9.1 叠加 Overlay	59
9.1.1 公式	59
9.1.2 映射面和同图等效曲线	59
9.1.3 程序模拟该模式计算结果	60
9.1.4 验证	60
9.1.5 用途示例	61
9.2 柔光 SoftLight	62
9.3 伽马矫正	62
9.3.1 公式	62
9.3.2 映射面和同图等效曲线	64
9.3.3 程序模拟该模式计算结果	64
9.3.4 验证	65
9.4 强光 HardLight	66
9.4.1 公式	66

9.4.2 和填充结合	66
9.4.3 融合不透明度	66
9.4.4 映射面和同图等效曲线	67
9.4.5 程序模拟该模式计算结果	67
9.4.6 验证	68
9.5 线性光 LinearLight	69
9.5.1 公式	69
9.5.2 和填充结合	69
9.5.3 融合不透明度	69
9.5.4 映射面和同图等效曲线	70
9.5.5 程序模拟该模式计算结果	70
9.5.6 验证	71
9.6 点光 PinLight	72
9.6.1 公式	72
9.6.2 和填充结合	72
9.6.3 融合不透明度	72
9.6.4 映射面和同图等效曲线	72
9.6.5 程序模拟该模式计算结果	73
9.6.6 验证	73
9.7 亮光 VividLight	74
9.7.1 公式	74
9.7.2 加上 fill	74
9.7.3 融合不透明度	74
9.7.4 映射面和同图等效曲线	74
9.7.5 程序模拟该模式计算结果	75
9.7.6 验证	75
9.8 实色混合 HardMix	76
9.8.1 公式	76
9.8.2 加上 fill	76
9.8.3 融合不透明度	76
9.8.4 映射面和同图等效曲线	76
9.8.5 程序模拟该模式计算结果	78
9.8.6 验证	79
第 10 章 差值组	81
10.1 差值 Difference	81
10.1.1 公式	81
10.1.2 结合填充	81
10.1.3 融合不透明度	81
10.1.4 映射面和同图等效曲线	81
10.1.5 程序模拟该模式计算结果	82
10.1.6 验证	82
10.2 排除 Exclusion	83
10.2.1 公式	83
10.2.2 结合填充	83

10.2.3 融合不透明度	83
10.2.4 如果在该模式下，混合图层是白色，黑色或者中性灰色	83
10.2.5 映射面和同图等效曲线	84
10.2.6 程序模拟该模式计算结果	84
10.2.7 验证	85
10.3 减去 Subtract	86
10.3.1 公式	86
10.3.2 结合填充	86
10.3.3 融合不透明度	86
10.3.4 映射面和同图等效曲线	86
10.3.5 程序模拟该模式计算结果	87
10.3.6 验证	87
10.4 划分 Divide	88
10.4.1 公式	88
10.4.2 结合填充	88
10.4.3 融合不透明度	88
10.4.4 一次负片转颜色减淡	88
10.4.5 映射面和同图等效曲线	88
10.4.6 程序模拟该模式计算结果	88
10.4.7 验证	88
第 11 章 颜色组	91
11.1 RGB→HSY 转换算法	91
11.2 HSY 替换算法	93
11.3 色相 Hue	95
11.3.1 结合填充	95
11.3.2 融合不透明度	95
11.3.3 程序模拟该模式计算结果	95
11.3.4 验证	95
11.4 饱和度 Saturation	96
11.4.1 公式	96
11.4.2 结合填充	96
11.4.3 融合不透明度	96
11.4.4 程序模拟该模式计算结果	96
11.4.5 验证	98
11.5 颜色 Color	99
11.5.1 结合填充	99
11.5.2 融合不透明度	99
11.5.3 程序模拟该模式计算结果	99
11.5.4 验证	100
11.6 明度 Luminosity	101
11.6.1 设置明度，就直接使用设置明度	101
11.6.2 结合填充	101
11.6.3 融合不透明度	101
11.6.4 程序模拟该模式计算结果	101

11.6.5 验证	102
第 12 章 特殊的 5 种模式	103
12.1 穿透	103
12.2 相加	103
12.3 相减	103
12.4 背后	103
12.5 擦除	103
第 13 章 调整图层和图层混合模式	105
第 14 章 工具和混合模式	109
14.1 画笔类	109
14.1.1 画笔工具	109
14.1.2 铅笔工具	110
14.1.3 历史记录画笔工具	111
14.1.4 历史记录艺术画笔工具	112
14.1.5 修复画笔工具	113
14.1.6 污点修复画笔工具	114
14.1.7 颜色替换工具	115
14.2 油漆桶工具	116
14.3 图章类	117
14.3.1 仿制图章	118
14.3.2 图案图章工具	119
14.4 渐变工具	120
14.5 计算类	121
14.5.1 应用图像	122
14.5.2 计算	123
14.6 其他	124
14.6.1 锐化工具	125
14.6.2 模糊工具	126
14.6.3 涂抹工具	127
第 15 章 图层样式和混合模式	129
15.1 斜面和浮雕	129
15.2 描边	130
15.3 内阴影	131
15.4 内发光	132
15.5 光泽	133
15.6 颜色叠加	134
15.7 渐变叠加	135
15.8 图案叠加	136
15.9 外发光	137
15.10 投影	138
15.11 常规混合	139
15.11.1 混合模式	139

15.11.2 不透明度	139
15.12 高级混合	140
15.12.1 填充不透明度	140
15.12.2 通道	141
15.12.3 挖空	142
15.12.4 将内部效果混成组	143
15.12.5 将剪贴图层混成组	144
15.12.6 透明形状图层	145
15.12.7 图层蒙版隐藏效果	146
15.12.8 矢量蒙版隐藏效果	147
第 16 章 混合颜色带 Blend If	149
16.1 怎么开启	149
16.2 原理	149
16.3 色彩模式和颜色混合带	149
16.3.1 RGB	149
16.3.2 CMYK	150
16.3.3 LAB	150
16.3.4 灰度	150
第 17 章 速查表	151
第 18 章 参考文档	153

第1章 前言

内容提要

- 本教程非常详细，请用心看完
- 本教程如果有任何问题，欢迎评论区留言讨论
- 本教程为了避免冗余，一些不必要的截图就省略了
- **注**本教程只讨论 8bit 的情形下的混合
- **注**未经许可，不可转载

1.1 泡一杯茶

在一切开始之前，我们先泡一杯茶，泡一杯茶需要一杯开水，一袋茶叶，如果太苦我们还需要一些水来兑一下，好的我们来温习一下泡茶的过程。



1. 我们准备一壶开水。
2. 我们准备一包茶叶，放多少取决于个人口味。
3. 我们把茶叶放到开水杯子中，在这个过程中可以静置或者搅拌也可以煮一下，等待茶叶和热水充分融合。
4. 我们想喝茶就把茶从茶壶中倒到一个杯子中，如果感觉苦了就加一点开水。
5. 最后我们会得到一杯符合我们口味的茶。

一般我们泡茶就是这几个步骤，你会泡茶吗，如果你会，那么恭喜你，你已经掌握了图层混合模式。



1. 一壶开水相当于基础图层。
2. 一包茶叶相当于混合图层，它将和一壶开水组成一个新的饮品-茶。
3. 茶叶放了多少就是填充，可以是一包，半包，也可以不放，放的越多，得到的那壶茶就越浓郁。
4. 静置或者搅拌也可以煮一下就是混合模式，这里的方式多种多样。
5. 喝茶的时候倒到杯子里面，茶水会和开水混合，茶水占据的比例就是不透明度。
6. 一杯符合我们口味的茶就是结果图层。

提示

图层混合模式在 PS 中一共有 27 种，对分组来说还有一种叫做穿透，另外在其他一些菜单中还有一些不常见的，比如笔刷菜单中的背后和清除，以及计算工具中的相加和相减。由此，我们便可以得出，PS 中一共有 $27 + 1 + 2 + 2 = 32$ 种图层混合模式。接下来，我们将会对每一种模式进行详细解读。

1.2 什么是图层混合模式

图层混合模式究竟是什么，他的本质代表什么？图层混合模式的本质就是对像素的运算，就像是泡茶的过程中的搅拌，比如 $x + y = z$ 或 $1 + 1 = 2$ 一种图层混合模式，就代表一种运算方式，这种运算方式会把像素点 1 和像素点 2 融合成为像素点 3。

提示

在本教程中，对公式本身进行讨论，一些所谓的衍生概念我们尽量不提，因为会造成冗余，比如增色减色等衍生概念，我们只对本质问题讨论。如果某些过程在公式当中已经包含了，我们就不再截图了，因为不想冗余，比如图层混合模式之间的相互转换。

第2章 前置概念

2.1 像素点

像素点是 PS 可以处理的最小单元，为了方便我们使用 Pix 或者 C 来表示

2.2 色彩空间

在图层混合模式中，涉及到的色彩空间一共有两种，第一是 RGB，也就是红绿蓝三通道，第二是 HSY，也就是色相，饱和度，明度。如果是 RGB 空间则使用 Pix 来表示像素点，如果是 HSY 空间则使用 C 来表示像素点。

2.2.1 RGB

如果使用 RGB 来表示一个像素点那么像素点可以表示为

$$Pix = (\text{红通道}, \text{绿通道}, \text{蓝通道}) \quad (2.1)$$

为了便于和后面的简称区分开，我们使用颜色 Red、Green、Blue 和通道的英文字母 channel 的开头字母来表示各个通道。于是上面的公式也可以表示为。

$$Pix = (RC, GC, BC) \quad (2.2)$$

在 8bit 的图像中，使用 8 个字节来表示一个像素，于是可以表示 $2^8 = 256$ 个数字，也就是 $[0, 255]$ 这个区间内的数字。如果数字越大那么这个通道就越亮。

2.2.2 HSY

如果使用 HSY 来表示一个像素点 $C = (Hue, Stratuation, Luminosity) = (H, S, Y)$ ，为了方便统一表述，在涉及到 HSY 时，如果我们需要表达 RGB 数值，我们使用 $C = (C_{red}, C_{green}, C_{blue}) \Leftrightarrow (RC, GC, BC)$

提示

- Hue 也就是色相，取值范围是 $[0^\circ, 360^\circ]$ ，决定是什么颜色
- Stratuation 饱和度，取值范围是 $[0, 100]$ ，决定这种颜色鲜艳程度或者换一种说法-颜色有多少
- Luminosity 明度，取值范围是 $[0, 100]$ ，决定这种颜色有多亮

上述两种色彩空间的转换关系如下：

$$Hue = \begin{cases} 0 & max = min \\ 60^\circ \times \frac{gc - bc}{max - min} & rc = max \quad and \quad gc \geq bc \\ 60^\circ \times \frac{gc - bc}{max - min} + 360^\circ & rc = max \quad and \quad gc < bc \\ 60^\circ \times \frac{bc - rc}{max - min} + 120^\circ & gc = max \\ 60^\circ \times \frac{rc - gc}{max - min} + 240^\circ & bc = max \end{cases} \quad (2.3)$$

$$Stratuation = max - min \quad (2.4)$$

$$Luminosity = 0.3rc + 0.59gc + 0.11 - bc \quad (2.5)$$

2.3 图层

像素点可以组合形成图层也就是由点到面，图层我们使用 Layer 来表示，那么一个 $m \times n$ 的图层和像素的关系可以表示为

2.3.1 RGB 表示法

$$\begin{aligned}
 Layer &= \left\{ \begin{array}{ccc} Pix_{(1,1)} & \cdots & Pix_{(1,n)} \\ \vdots & & \vdots \\ Pix_{(i,1)} & \cdots & Pix_{(i,n)} \\ \vdots & & \vdots \\ Pix_{(m,1)} & \cdots & Pix_{(m,n)} \end{array} \right\} \\
 &= \left\{ \begin{array}{ccccc} (RC_{(1,1)}, GC_{(1,1)}, BC_{(1,1)}) & \cdots & (RC_{(1,n)}, GC_{(1,n)}, BC_{(1,n)}) \\ \vdots & & \vdots \\ (RC_{(i,1)}, GC_{(i,1)}, BC_{(i,1)}) & \cdots & (RC_{(i,n)}, GC_{(i,n)}, BC_{(i,n)}) \\ \vdots & & \vdots \\ (RC_{(m,1)}, GC_{(m,1)}, BC_{(m,1)}) & \cdots & (RC_{(m,n)}, GC_{(m,n)}, BC_{(m,n)}) \end{array} \right\}
 \end{aligned} \tag{2.6}$$

2.3.2 HSY 表示法

$$Layer = \left\{ \begin{array}{ccc} C_{(1,1)} & \cdots & C_{(1,n)} \\ \vdots & & \vdots \\ C_{(i,1)} & \cdots & C_{(i,n)} \\ \vdots & & \vdots \\ C_{(m,1)} & \cdots & C_{(m,n)} \end{array} \right\} = \left\{ \begin{array}{ccccc} (H_{(1,1)}, S_{(1,1)}, Y_{(1,1)}) & \cdots & (H_{(1,n)}, S_{(1,n)}, Y_{(1,n)}) \\ \vdots & & \vdots \\ (H_{(i,1)}, S_{(i,1)}, Y_{(i,1)}) & \cdots & (H_{(i,n)}, S_{(i,n)}, Y_{(i,n)}) \\ \vdots & & \vdots \\ (H_{(m,1)}, S_{(m,1)}, Y_{(m,1)}) & \cdots & (H_{(m,n)}, S_{(m,n)}, Y_{(m,n)}) \end{array} \right\}
 \tag{2.7}$$

2.3.3 涉及图层

图层混合模式一共涉及三个图层分别是

1、基础图层或者叫做底图

就是下方的图层，我们使用英文单词 *below* 的开头字母 **B** 来表示，符号是 *LayerB*，其中的像素点就可以表示为 *PixB* 或 *CB*

2、混合图层或者叫做调整图层或绘画图层

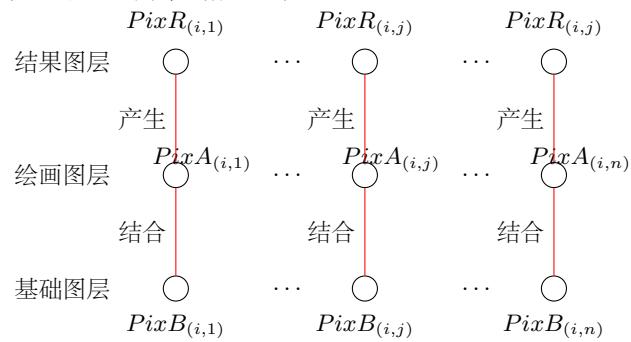
混合图层或者叫绘画图层，就是上方的图层，我们使用英文单词 *above* 的开头字母 **A** 来表示，符号是 *LayerA*，其中的像素点就可以表示为 *Pixa* 或 *CA*

3、结果图层

就是以何种方式处理之后的结果，他是通过 *LayerB* 和 *LayerA* 的结合来表示。使用结果的英文单词 *result* 的首字母来表示，符号是 *LayerR*，其中的像素点就可以表示为 *PixR* 或 *CR*

2.4 像素混合图例

如果我们取某一行像素来进行一个简单的展示的话。



2.5 图层大小

PS 新建的过程中，我们可以看到一些图片属性，其中就有图片大小，这里我们讨论的图层大小都默认是 $m \times n$ ，也就是高度是 m 宽度是 n 的图层大小

2.6 映射面

对于任意的一个混合模式，如果我们将它产生的所有基础图层和混合图层的结果，都在一个三维坐标中标记出来，则我们就会得到一个由 $256 \times 256 = 65536$ 个点组成的平面，换句话说，不管什么混合模式，基础图层和混合图层是什么值，在 8bit 的模式下，都只有 65536 种可能结果。这个面就是映射面。

提示

图 2.1 是我们将每一种图层混合模式表达式输入 Matlab 之后 Matlab 为我们呈现的结果， x 轴为基础图层， y 轴为混合图层， z 轴为结果图层，我们可以看到越大的带你则越白，1 最白，越小的点越黑，0 最黑。于是我们就可以通过黑白程度来判断结果大小。

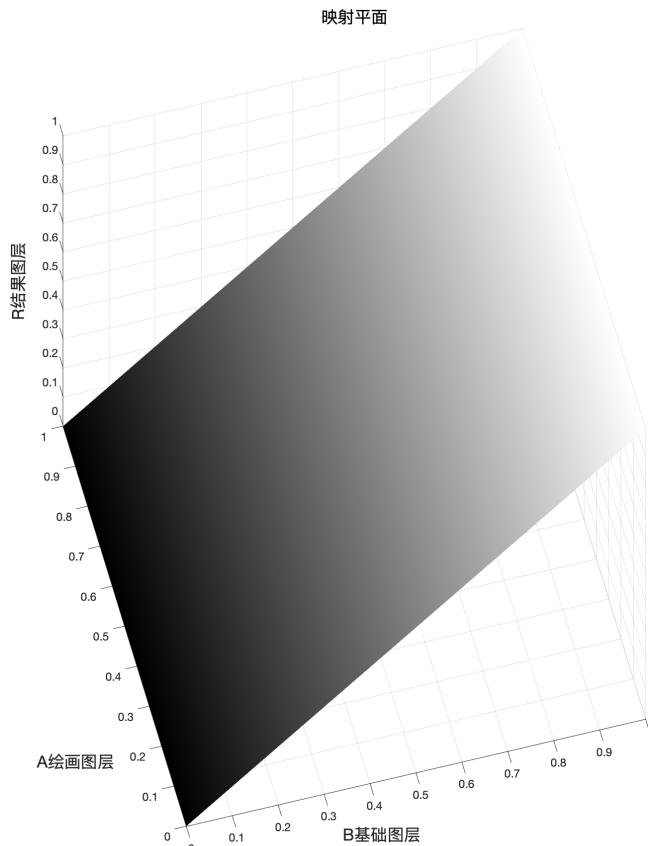


图 2.1: 映射面初始状态

2.6.1 映射面示例

映射面和原图面的对比使用混合模式后的映射面和原图映射面的对比由图2.2可以得知，这是一种变暗模式并且在混合图层和基础图层中选择了比较小的作为结果，因为可以明显看到一部分平面在原平面（图2.2中的蓝色平面）的下方。

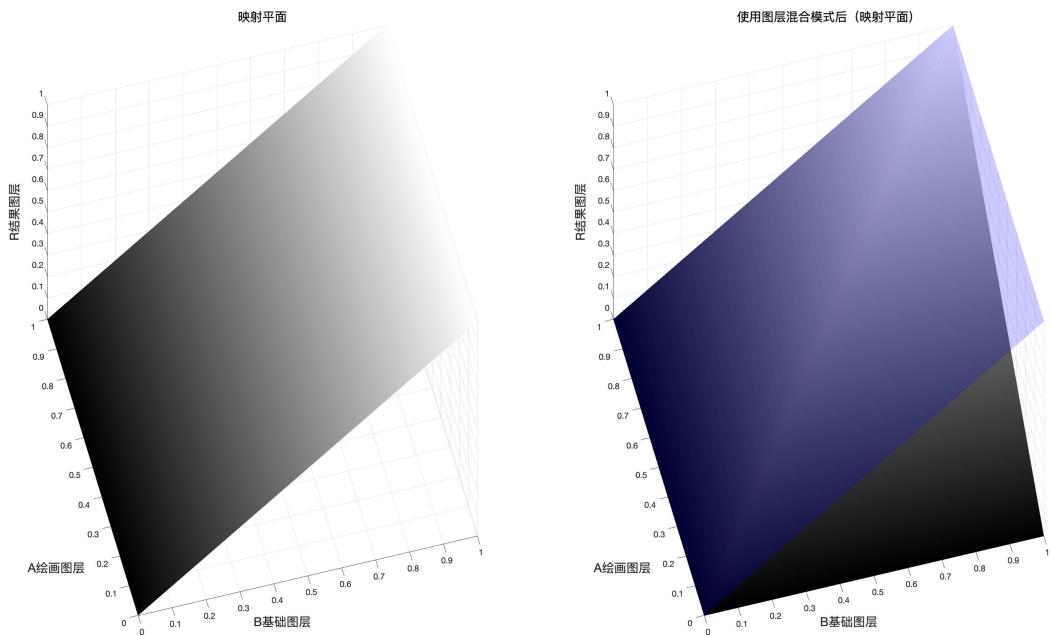


图 2.2: 映射面和映射面初始状态对比

2.6.2 同图平面

如果我们把基础图层和混合图层取值相同的点都标记出来，我们会得到一个对角平面，这个平面就是我们所说的同图平面，同图平面以及其和映射面相交的图像，图中绿色的平面就是同图平面，他和映射面的交线就是我们可以使用曲线工具模拟的曲线。如图2.3所示。

2.6.3 同图混合曲线

此时相当于 $a = b$ 带入到某个图层混合模式的表达式中，我们此处以线性加深为例，线性加深原本的表达式是

$$r = \text{Linear Burn}(b, a) = b - (1 - a) = b + a - 1 \quad (2.8)$$

若 $b = a$ 则表达式可以写成

$$\begin{aligned} r &= (b, b) = b - (1 - b) \\ &= b + b - 1 = 2b - 1 \end{aligned} \quad (2.9)$$

此时，在坐标系中画出该图像就如同上图所示。这时候我们只要打开曲线工具，然后拉一条曲线，和上图相同，就可以得到和这种混合模式在基础图层和混合图层相同的情况下的到结果图层相同的效果。

同图平面和映射面的交线就是可以使用曲线工具模拟这种混合模式时画出的曲线。同图平面有时候也可以作为分割平面，比如在实色混合模式的时候。

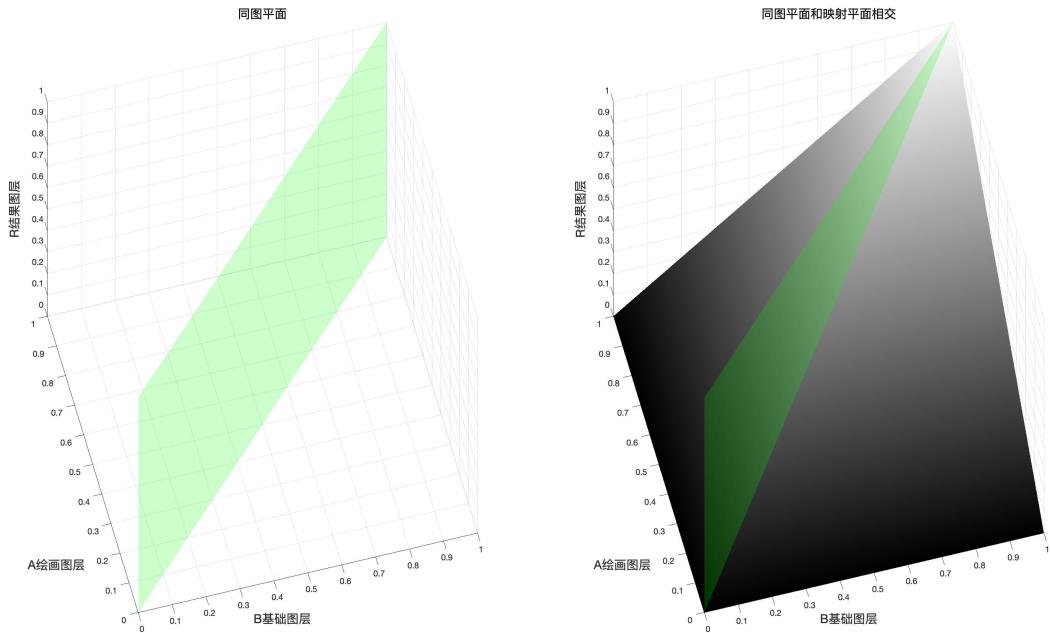


图 2.3: 同图平面

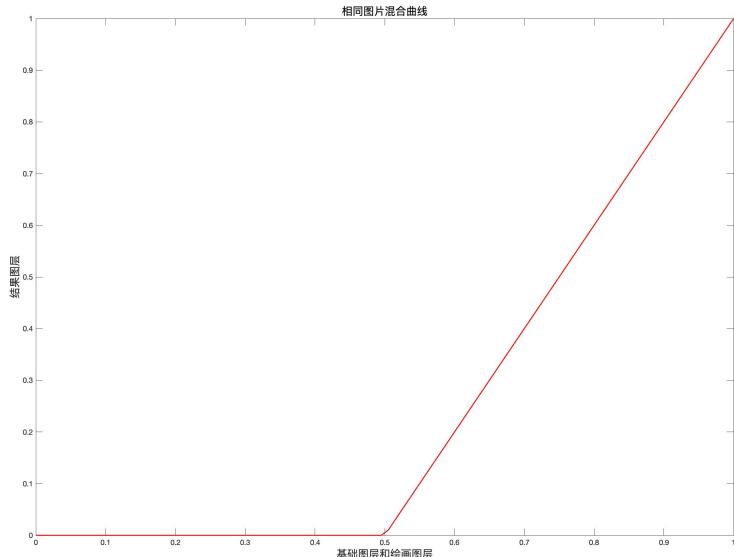


图 2.4: 同图混合曲线示例

2.6.4 中性灰平面

如果我们把基础图层或者绘画图层中 128 中性灰的点都标记出来，我们会得到一个中性灰平面这个平面一般作为结合两种混合模式的分割面

2.6.4.1 基础图层的中性灰平面

如果两种混合模式是以基础图层的取值作为分界依据的时候，使用此平面作为分割和结合依据。如图2.5所示。

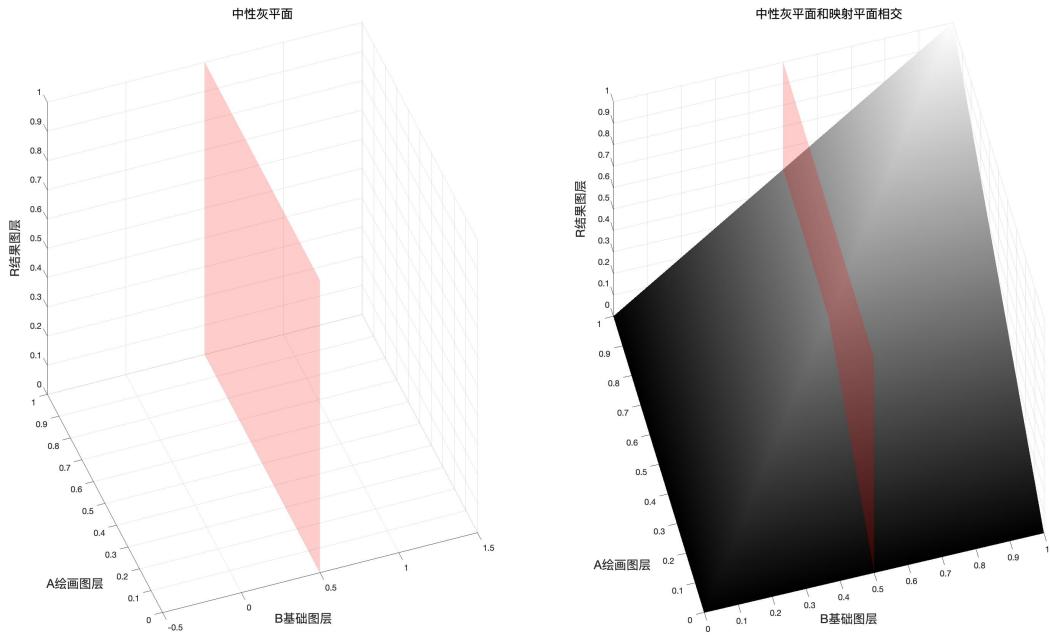


图 2.5: 基础图层的中性灰平面

2.6.4.2 混合图层的中性灰平面

如果两种混合模式是以混合图层的取值作为分界依据的时候，使用此平面作为分割和结合依据。同图平面，中性灰平面（基础中性灰平面，混图层中性灰平面）都是结合两种混合模式的依据。如图2.6所示。

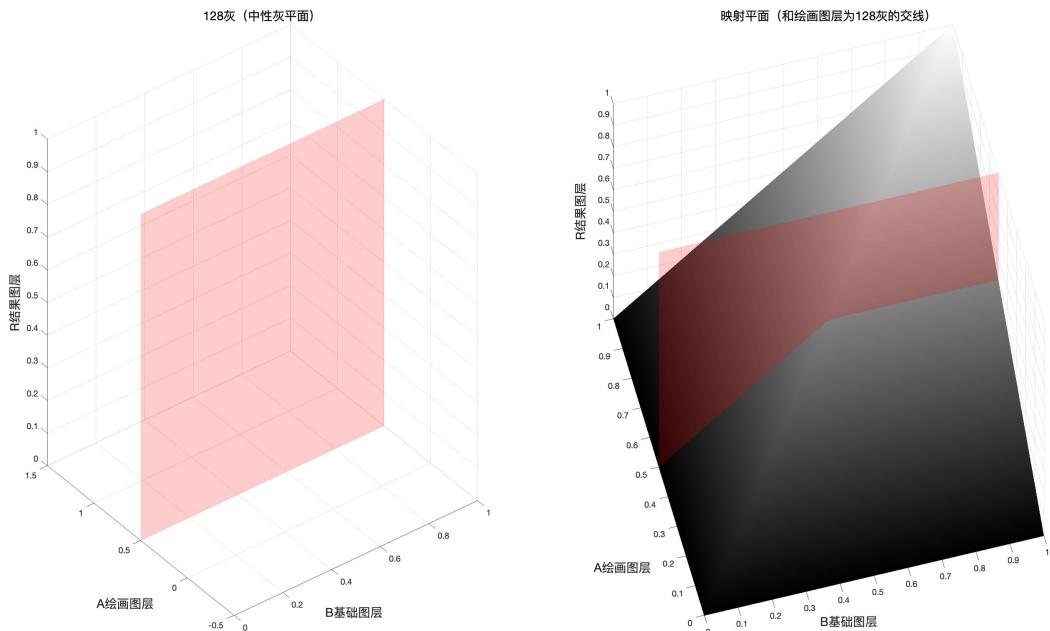


图 2.6: 混合图层的中性灰平面

2.7 关系总览

为了方便大家理解，我们在此给出 27 中图层混合模式的总览，如图2.7所示。

- 可转换：代表两种混合模式可以在一定条件下等效另外一种
- 可交换：底图和绘画图层交换，两种混合模式产生的结果相同，也就是互逆
- 组成：一种混合模式可以由另外几种组成

2.7.1 特殊关系

从总览那幅图我们可以得出一些有趣的结论，就是有的混合模式在结合一些操作之后，结果等价于另外一种，有的图层混合模式和另外一种是互逆的关系，有些可以组合，有些对黑色无效，有些对白色无效，有些对中性灰无效，总之，他们的一切都蕴含在了公式中，如果你想搞清楚，就仔细研究一下。

2.7.2 可转换

可转换可能有些在实际操作中有偏差，因为在计算过程中可能出现大于 1 或者小于 0 的情况，导致被取舍。以下都是理论存在的情况。

- 变亮 \Leftrightarrow 变暗（三次负片）
- 滤色 \Leftrightarrow 正片叠底（三次负片）
- 线性减淡 \Leftrightarrow 线性加深（三次负片）
- 颜色减淡 \Leftrightarrow 颜色加深（三次负片）
- 浅色 \Leftrightarrow 深色（三次负片）
- 线性加深 \Leftrightarrow 减去（一次负片）
- 颜色减淡 \Leftrightarrow 划分（一次负片）
- 颜色加深 \Leftrightarrow 划分（两次负片）

2.7.3 互逆（可交换）

- 叠加 \Leftrightarrow 强光
- 颜色 \Leftrightarrow 明度
- 实色混合 \Leftrightarrow 线性光（当实色混合填充设置为 0.5 也就是 50%）

2.7.4 组成

- 叠加 = 正片叠底 + 滤色
- 强光 = 正片叠底 + 滤色
- 线性光 = 线性加深 + 线性减淡
- 实色混合 = 线性加深 + 线性减淡
- 亮光 = 颜色加深 + 颜色减淡
- 点光 = 变亮 + 变暗
- 柔光 = 系数 2 的伽马矫正 + 系数为 0.5 的伽马矫正

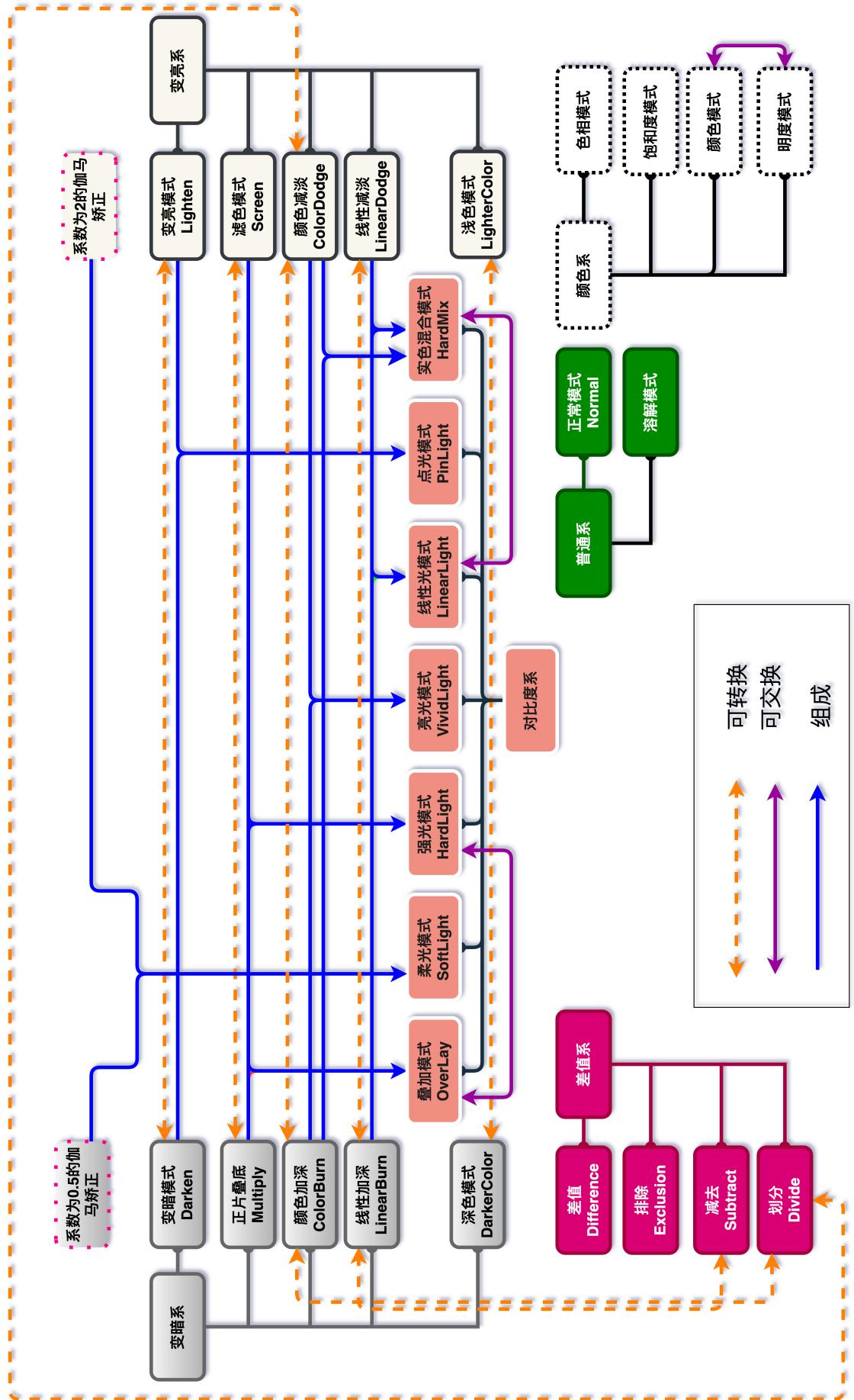


图 2.7: 总览度

第3章 运算相关

内容提要

- 本章包括图层混合模式需要知道的一切计算细节。

3.1 归一化操作

红色(255, 0, 0), 黑色(0, 0, 0), 白色(255, 255, 255), 中性灰色(128, 128, 128)。如果我们对每个通道都用归一化操作。

RC, GC, BC 的取值范围都是 [0, 255]。

提示

- 此处我们使用线性归一化, 具体方法是

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

在这里我们可以看到, 最大值是 255, 最小值是 0, 于是上式可以简化为颜色值除以 255

以 RC 为例, $RC \in [0, 255]$

那我们可以得到归一化后

$$\frac{RC}{255} \Rightarrow rc \in [0, 1] \quad (3.1)$$

那么以上的四种颜色就可以表示为, 红色(1, 0, 0), 黑色(0, 0, 0), 白色(1, 1, 1), 中性灰色(0.5, 0.5, 0.5)。

$$Pix = (RC, GC, BC) = (rc, gc, bc) \quad (3.2)$$

提示

- 这里也是为什么, 在有些地方可以看到一堆 255, 一些地方看不到的原因。

3.2 一些运算规则

如果一个图片的数值需要扩大或者缩小 x 倍

3.2.1 通道级别

通道数值乘以 x

$$RC \times x$$

3.2.2 像素级别

如果一个像素点 $Pix \times x$ 则表示其中所有通道数值都乘 x , 也就是说

$$Pix \times x = (RC \times x, GC \times x, BC \times x) \quad (3.3)$$

3.2.3 图层级别

如果是图层 $Layer \times x$ 则表示所有像素点都乘 x 也就是说

$$Layer \times x = \left\{ \begin{array}{ccc} Pix_{(1,1)} \times x & \cdots & Pix_{(1,n)} \times x \\ \vdots & & \vdots \\ Pix_{(1,n)} \times x & \cdots & Pix_{(i,n)} \times x \\ \vdots & & \vdots \\ Pix_{(m,1)} \times x & \cdots & Pix_{(m,n)} \times x \end{array} \right\} \quad (3.4)$$

$$= \left\{ \begin{array}{ccc} (RC_{(1,1)} \times x, GC_{(1,1)} \times x, BC_{(1,1)} \times x) & \cdots & (RC_{(1,n)} \times x, GC_{(1,n)} \times x, BC_{(1,n)} \times x) \\ \vdots & & \vdots \\ (RC_{(1,n)} \times x, GC_{(1,n)} \times x, BC_{(1,n)} \times x) & \cdots & (RC_{(i,n)} \times x, GC_{(i,n)} \times x, BC_{(i,n)} \times x) \\ \vdots & & \vdots \\ (RC_{(m,1)} \times x, GC_{(m,1)} \times x, BC_{(m,1)} \times x) & \cdots & (RC_{(m,n)} \times x, GC_{(m,n)} \times x, BC_{(m,n)} \times x) \end{array} \right\}$$

提示

一般我们说底图减去混合图层，或者，基础图层和混合图层使用混合模式，都将这些级别的运算包含在了一句简单的表述中。

3.2.4 HSY 的运算

如果是 HSY 的色彩空间，那么我们的表达式其实和 RGB 的没有什么区别，如果非要说区别，就是我们需要把像素的表达式稍微变一下。一般 HSY 空间只涉及交换，而不涉及运算。

3.3 混合模式相关符号

图层混合模式我们使用 $BlendMode$ 来表示，这里是一个泛指，如果涉及到具体的混合模式，有专门的表示符号，比如正常模式使用 $Normal$ 使用矩阵我们有：

$$\text{基础图层} \iff LayerA = \left\{ \begin{array}{ccc} PixA_{(1,1)} & \cdots & PixA_{(1,n)} \\ \vdots & & \vdots \\ PixA_{(i,1)} & \cdots & PixA_{(i,n)} \\ \vdots & & \vdots \\ PixA_{(m,1)} & \cdots & PixA_{(m,n)} \end{array} \right\} \quad (3.5)$$

$$\text{混合图层} \iff LayerB = \left\{ \begin{array}{ccc} PixB_{(1,1)} & \cdots & PixB_{(1,n)} \\ \vdots & & \vdots \\ PixB_{(i,1)} & \cdots & PixB_{(i,n)} \\ \vdots & & \vdots \\ PixB_{(m,1)} & \cdots & PixB_{(m,n)} \end{array} \right\} \quad (3.6)$$

$$\text{结果图层} \Leftrightarrow LayerR = \begin{Bmatrix} PixR_{(1,1)} & \cdots & PixR_{(1,n)} \\ \vdots & & \vdots \\ PixR_{(i,1)} & \cdots & PixR_{(i,n)} \\ \vdots & & \vdots \\ PixR_{(m,1)} & \cdots & PixR_{(m,n)} \end{Bmatrix} \quad (3.7)$$

再结合上面的三者关系公式

$$\begin{aligned} \text{结果图层} &= BlendMode(\text{基础图层}, \text{混合图层}) \\ &\Updownarrow \\ LayerR &= BlendMode(LayerB, LayerA) \end{aligned} \quad (3.8)$$

我们得到

$$LayerR = \begin{Bmatrix} BlendMode(PixB_{(1,1)}, PixA_{(1,1)}) & \cdots & BlendMode(PixB_{(1,n)}, PixA_{(1,n)}) \\ \vdots & & \vdots \\ BlendMode(PixB_{(i,1)}, PixA_{(i,1)}) & \cdots & BlendMode(PixB_{(i,n)}, PixA_{(i,n)}) \\ \vdots & & \vdots \\ BlendMode(PixB_{(m,1)}, PixA_{(m,1)}) & \cdots & BlendMode(PixB_{(m,n)}, PixA_{(m,n)}) \end{Bmatrix} \quad (3.9)$$

对于其中任意项

$$Pix_R = BlendMode(Pix_B, Pix_A) \quad (3.10)$$

如果这种混合模式基于 RGB 色彩空间则上述表达式可以写为

$$(rc_R, gc_R, bc_R) = BlendMode((rc_B, gc_B, bc_B), (rc_A, gc_A, bc_A)) \quad (3.11)$$

或者未归一化形式

$$(RC_R, GC_R, BC_R) = BlendMode((RC_B, GC_B, BC_B), (RC_A, GC_A, BC_A)) \quad (3.12)$$

如果这种混合模式基于 HSY 色彩空间

$$(H_R, S_R, Y_R) = BlendMode((H_B, S_B, Y_B), (H_A, S_A, Y_A)) \quad (3.13)$$

提示

在下面的所有组中，如果不特别说明基于 RGB 色彩空间的混合模式都是使用归一化的数值进行计算。并且，为了方便大家验证计算的正确性，这里提供一个 java 程序的链接给各位，GitHub 的地址如下，输入你要混合的像素的数值，就可以得到对应混合模式混合的结果数值。

3.4 辅助计算程序

为了辅助我们计算并且验证这些理论公式和 PS 中实际运行的结果是否一致，我们借助 java 代码来实现具体的计算过程，手动计算可以，但是浪费时间，我们有更简单的方式。我们定义一个 java 类 BlendColor，这个类用 RGB 作为基础，每次计算都以 RGB 存储，如果需要其他表达式，则我们通过 RGB 转换。

```
public class BlendColor {
    public ColorItem red;
    public ColorItem green;
    public ColorItem blue;
    public List<ColorItem> orderColorList;

    public BlendColor(double redv, double greenv, double bluev) {
        ColorItem red = new ColorItem("red", redv);
        ColorItem green = new ColorItem("green", greenv);
        ColorItem blue = new ColorItem("blue", bluev);
        this.orderColorList = new ArrayList<ColorItem>();
        this.red = red;
        this.green = green;
        this.blue = blue;
        this.orderColorList.add(red);
        this.orderColorList.add(green);
        this.orderColorList.add(blue);
        // 排序
        Collections.sort(this.orderColorList);
    }
    ...
}
```

3.5 混合模式的组合

一般来说，混合模式的组合会以基础图层 B 或者混合图层 A 的中性灰分割线为界，组合两种不同的混合模式。例如，强光模式，它是由正片叠底和滤色两种模式以混合图层的中性灰分割线为界完成组合

- 正片叠底 $r = Multiply(b, a) = b \times a$
- 滤色 $r = Screen(b, a) = 1 - (1 - b)(1 - a)$
- 强光

$$r = HardLight(b, a) = \begin{cases} Multiply(b, 2a) & a \leq 0.5 \\ Screen(b, 2(a - 0.5)) & a > 0.5 \end{cases} \quad (3.14)$$

$$= \begin{cases} 2ba & a \leq 0.5 \\ 1 - 2(1 - b)(1 - a) & a > 0.5 \end{cases}$$

第4章 不透明度和填充

内容提要

- 不透明度和填充就如同一杯茶中水的多少和茶叶的多少。对于图层混合模式来说，填充越大，代表这种混合方式的程度越大，不透明度越大，代表基础图层和结果图层按照比例混合时结果图层占比的大小越大。

4.1 在哪

不透明度和填充可以在图层的右上方找到，是两个滑块，范围都是 [0, 100]

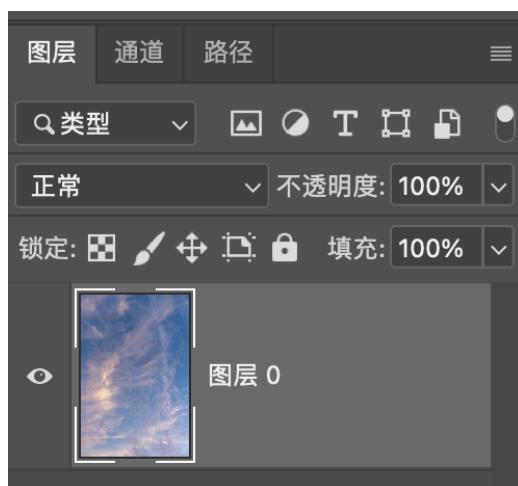


图 4.1: “不透明度”和“填充”所在位置

填充还有一个名字叫填充不透明度

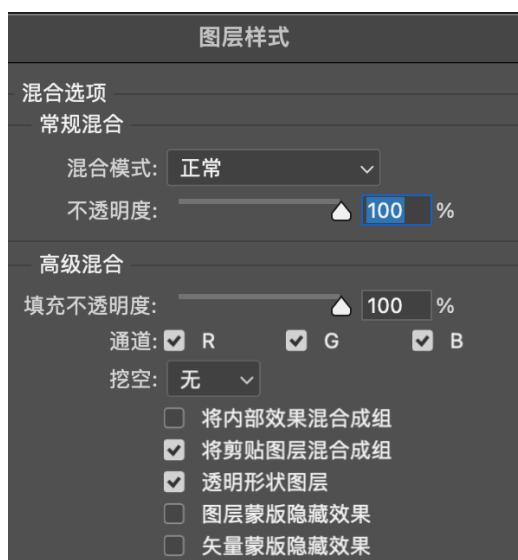


图 4.2: “填充”和“填充不透明度”

4.2 不透明度和填充公式

不透明度的英文单词是 *Opacity*, 填充的英文单词是 *fill*, 两者在图层混合模式中有时相同有时不同。具体我们会在后面说明。简单描述二者的区别, 就是, 不透明度就是将基础图层 *B* 和结果图层 *R*, 使用 *Opacity* 比例进行混合, 公式如下:

$$\text{Opacity}(b, a) = op \times b + (1 - op) \times \text{BlendMode}(b, a) \quad (4.1)$$

其中 *op* 代表不透明度并且, 填充的优先级高于不透明度, 如果让两者一起产生效果则公式为:

$$\text{Opacity}(b, a) = op \times b + (1 - op) \times \text{Fill}(b, a) \quad (4.2)$$

提示

Fill(b, a) = BlendMode(b, a × fill) 这只是一个原则公式, 具体情况要视具体模式确定, 每一种计算方式都有细微区别。我们在此处不给出具体的公式和代码, 但是我们在每一种中, 我们会具体实现。

4.3 单纯混合模式与二者使用的区别

填充的计算方式有八个和其他的不同, 并且此时和不透明度的计算方法不一样。除了这八个, 不透明度和填充变化的结果是相同的。

他们分别是

- 线性加深
- 颜色加深
- 线性减淡
- 颜色减淡
- 线性光 (线性减淡 + 线性加深)
- 亮光 (颜色减淡 + 颜色加深)
- 实色混合 (线性减淡 + 线性加深或者颜色减淡 + 颜色加深)
- 差值

如果使用一句话概括这二者的区别就是:

提示

填充会让当前混合模式的效果弱化, 弱化的程度取决于 *fill* 的大小。
 不透明度决定了当前混合模式结果图层和基础图层混合的比例, *opacity* 越大, 结果图层占比越大。
 结合前面的例子, 填充越大放的茶叶就越多, 不透明度越大, 最后添加的水就越少。

4.4 代码实现

4.4.1 不透明度

```
public static BlendColor Opacity(BlendColor colorBase, BlendColor colorResult, double opacity) {
    double redOpacity = colorResult.red.value * opacity + colorBase.red.value * (1 - opacity);
    double greenOpacity = colorResult.green.value * opacity + colorBase.green.value * (1 - opacity);
    double blueOpacity = colorResult.blue.value * opacity + colorBase.blue.value * (1 - opacity);
```

```
return new BlendColor(redOpacity, greenOpacity, blueOpacity);  
}
```

4.4.2 填充

填充具体情况具体分析

4.5 不透明度和填充与图层样式

这里顺便提一下图层样式在两种调节方式下有什么区别，图层样式，比如描边，如果调节不透明度，则会同主体一起变，但是如果只调节填充，则只有主体会变而样式不会变。

第5章 图层混合模式的种类

内容提要

- 本章介绍按照不同方式对图层混合模式的分类
- 从不同的角度，图层混合模式有不同的种类

5.1 色彩空间分类

分类方法有很多，如果按照色彩空间，可以分成两类，RGB 和 HSY

5.1.1 RGB

除颜色组

5.1.2 HSY

颜色组

5.2 是否产生新的数值

如果根据运算方式，可以分成两类，一类是替换式，一类是融合式

5.2.1 替换式

- 正常组（正常，溶解）
- 变暗组（变暗，深色）
- 变亮组（变亮，浅色）
- 颜色组（色相，饱和度，颜色，明度）

提示

这种式的特点是，不会产生新的东西，只会复用之前的东西，比如正常组，正常就是直接使用混合图层作为结果，变暗组的变暗是使用基础图层和混合图层中通道较小的值组成新的像素，深色则是比较基础图层和混合图层通道值的和取小的作为结果，颜色组，则是根据 HSY 色彩空间，直接替换 HSY 的数值，并且再转换为 RGB，本质也是没有新的东西产生。

5.2.2 融合式

- 变暗组（正片叠底，颜色加深，线性加深）
- 变亮组（滤色，颜色减淡，线性减淡）
- 对比度组（叠加，柔光，强光，亮光，线性光，点光，实色混合）
- 差值组（差值，排除，减去，划分）

提示

虽然点光是变暗和变亮的混合，但是混合过程中做了处理，所以此处我们也按照融合式对待，此类的特点是，会产生新的数值。融合式就是产生了新的数值，比如正片叠底，结果图层的数值由基础图层和混合图层的乘积得到，他是不同于基础图层和混合图层的值。

5.3 是否顺序相关

如果根据是否受到图层顺序影响，如果

$$\text{BlendMode}(b, a) = \text{BlendMode}(a, b) \quad (5.1)$$

则说明结果不受图层顺序影响，前提是不调节 *fill* 和 *Opacity*

5.3.1 顺序无关

- 变暗组（变暗，线性加深，正片叠底，深色）
- 变亮组（变亮，线性减淡，滤色，浅色）
- 对比度组（实色混合）
- 差值组（差值，排除）

5.3.2 顺序相关

- 正常组（正常，溶解）
- 变暗组（颜色加深）
- 变亮组（颜色减淡）
- 对比度组（叠加，柔光，强光，亮光，线性光，点光）
- 差值组（减去，划分）
- 颜色组（色相，饱和度，颜色，明度）

5.4 是否 fill 和 Opacity 不同

也就是说，调节填充的百分比，和不透明度的百分比，即使数值相同，结果也有可能不同。

5.4.1 fill 和 Opacity 产生不同影响

- 变暗组（线性加深，颜色加深）
- 变亮组（线性减淡，颜色减淡）
- 对比度组（亮光，线性光，实色混合）
- 差值组（差值）

5.4.2 fill 和 Opacity 产生相同影响

- 正常组（正常，溶解）
- 变暗组（变暗，正片叠底，深色）
- 变亮组（变亮，滤色，浅色）
- 对比度组（叠加，柔光，强光，点光）
- 差值组（排除，减去，划分）

- 颜色组（色相，饱和度，颜色，明度）

提示

这里也不是说两者可以简单相乘，比如 $fill = 50\%$, $opacity = 50\%$, 的情况得到的结果，并不等于 $fill = 25\%$ 或者 $opacity = 25\%$ 所得到的结果。其实理论应该是。



$$\begin{aligned} r &= (\text{BlendMode}(b, a \times fill)) \times op + b \times op \\ &= (fill \times a + (1 - fill) \times b) \times op + b \times op \end{aligned} \quad (5.2)$$

第 6 章 普通组

内容提要

- 普通组包括正常模式和溶解模式，他们的本质都是替换
- 虽然名为普通，但是几乎包含了我们需要掌握的所有内容

6.1 正常 Normal

正常模式是一切的基础，也是我们理解和掌握混合模式的基础，正常模式可以看作是基于 RGB 颜色空间，也可以看作是基于 HSY 色彩空间

6.1.1 公式 (泡茶的方式，比如搅拌、静置)

对于像素维度公式

$$Pix_r = Normal(Pix_b, Pix_a) = Pix_a \quad (6.1)$$

正常模式通道维度初始公式

$$r = Normal(b, a) = a \quad (6.2)$$

6.1.2 融合填充 (放多少茶叶)

$$r = Fill(b, a) = fill \times a + (1 - fill) \times b \quad (6.3)$$

6.1.3 融合不透明度 (太苦了，最后加点水)

$$r = Opacity(b, a) = op \times Fill(b, a) + (1 - fill) \times b \quad (6.4)$$

对于整个像素

$$(r_{rc}, r_{gc}, r_{bc}) = Normal((b_{rc}, b_{gc}, b_{bc}), (a_{rc}, a_{gc}, a_{bc})) = (r_{rc}, r_{gc}, r_{bc}) \quad (6.5)$$

后面的混合模式我们不再讨论整个像素的公式，我们只讨论某个通道的结果。

6.1.4 整个像素融合填充

这里不同的混合模式公式不一定相同。

$$\begin{aligned} (r_{rc}, r_{gc}, r_{bc}) &= Fill((b_{rc}, b_{gc}, b_{bc}), (a_{rc}, a_{gc}, a_{bc})) \\ &= ((fill \times a_{rc} + (1 - fill) \times b_{rc}), \\ &\quad (fill \times a_{gc} + (1 - fill) \times b_{gc}), \\ &\quad (fill \times a_{bc} + (1 - fill) \times b_{bc})) \end{aligned} \quad (6.6)$$

6.1.5 整个像素融合不透明度

$$\begin{aligned}
 (r_{rc}, r_{gc}, r_{bc}) &= \text{Opacity}((b_{rc}, b_{gc}, b_{bc}), (a_{rc}, a_{gc}, a_{bc})) \\
 &= ((op \times \text{Fill}(a_{rc}, b_{rc}) + (1 - op) \times b_{rc}), \\
 &\quad (op \times \text{Fill}(a_{gc}, b_{gc}) + (1 - op) \times b_{gc}), \\
 &\quad (op \times \text{Fill}(a_{bc}, b_{bc}) + (1 - op) \times b_{bc}))
 \end{aligned} \tag{6.7}$$

6.1.6 程序模拟该模式计算结果

```
// 正常模式
public static BlendColor Normal(BlendColor colorBase, BlendColor colorBlend, double fill, double
    opacity) {
    double red = colorBlend.red.get01Value() * fill + colorBase.red.get01Value() * (1 - fill);
    double green = colorBlend.green.get01Value() * fill + colorBase.green.get01Value() * (1 - fill);
    double blue = colorBlend.blue.get01Value() * fill + colorBase.blue.get01Value() * (1 - fill);
    return ColorUtils.Opacity(colorBase, new BlendColor(red * 255, green * 255, blue * 255), opacity);
}
```

借助正常模式，我们可以看到不透明度和填充的关系，后面的模式讨论我们都不再说明像素级别的公式，因为没有必要，我们只讨论通道级别的公式。我们设定 $fill$ 为 40%， $Opacity$ 为 60%

提示

- ↳ 基础图层 RGB[111.00, 80.00, 60.00] HSY[23.53, 51.00, 87.10] HSB[23.53, 45.95, 43.53]
- ↳ 混合图层 RGB[80.00, 70.00, 156.00] HSY[246.98, 86.00, 82.46] HSB[246.98, 55.13, 61.18]

我们在 PS 中使用这两种颜色进行验证，发现符合我们的算法。

程序模拟结果

- ↳ 正常 (Normal) RGB[103.56, 77.60, 83.04] HSY[347.43, 25.96, 85.99] HSB[347.43, 25.07, 40.61]

6.1.7 验证

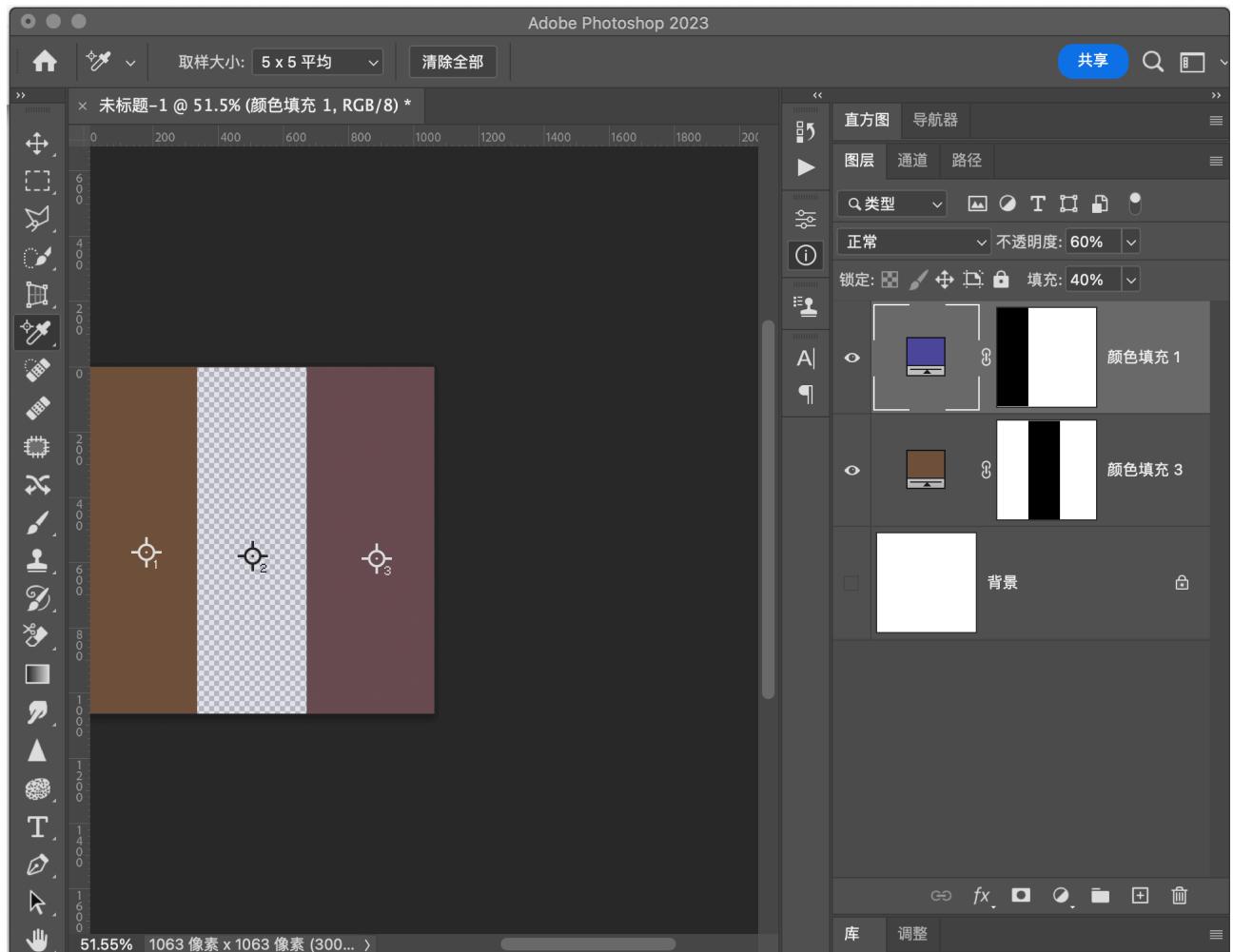


图 6.1

6.2 溶解 Dissolve

溶解模式是正常组的第二种模式，他依靠替换，并且其对不透明度是免疫的。

6.2.1 溶解模式初始公式

$$r = Dissolve(b, a) = a \quad (6.8)$$

6.2.2 如果融合了填充

$$r = Fill(b, a) = Random_{fill}(Dissolve(b, a), b) \quad (6.9)$$

提示

- ⚠ 不透明度对溶解模式是无效的。
- ⚠ *fill* 的值越大，则下层像素暴露的可能性越小。

6.2.3 程序模拟该模式计算结果

```
public static BlendColor Dissolve(BlendColor colorBase, BlendColor colorBlend, double fill, double opacity) {  
    double rand = Math.random(); // 产生随机数  
    if (rand < fill) {  
        return colorBlend;  
    } else {  
        return colorBase;  
    }  
}
```

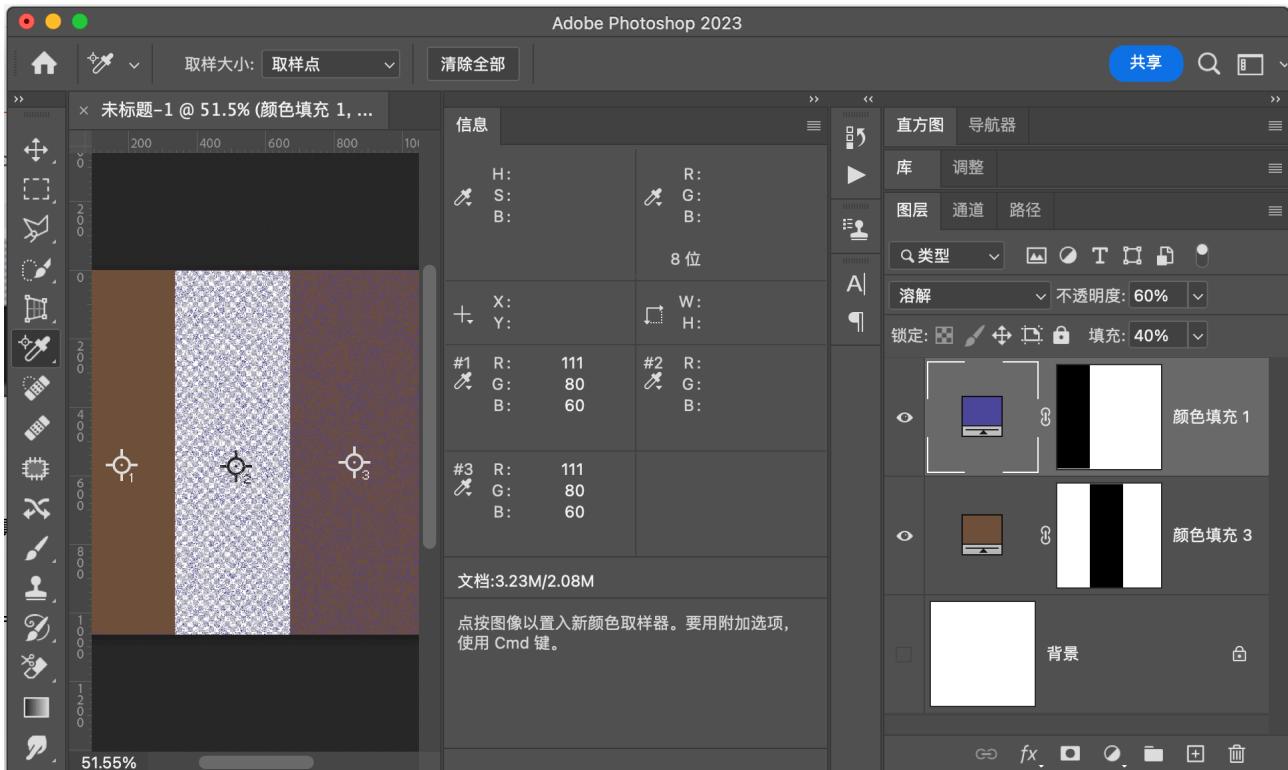
程序模拟结果 ✓

- ⚠ 溶解 (Dissolve) RGB[111.00, 80.00, 60.00] HSY[23.53, 51.00, 87.10] HSB[23.53, 45.95, 43.53]

6.2.4 验证

提示

溶解模式是根据概率来实现决定显示下方像素还是上方像素,于是取样点不同,则结果可以是上方的像素也可以是下方的像素。



6.2.5 用途示例

可以通过该模式实现一下粒子效果

第7章 变暗组

内容提要

- 这一组是基于 RGB 色彩空间，RGB 三个通道的数值取值范围是 [0, 255]，归一化之后取值范围是 [0, 1]。这一组就是利用一系列运算让基础图层的三个通道的数值或者他们的和小于等于原值。这也是变暗组的本质。和下面的变亮组一样，变暗组有两种方式将通道值或者通道值的和变小，那就是替换和运算，替换包括变暗和深色，运算包括其他三种。

7.1 变暗模式 Darken

如果将基础图层每个像素每个通道的数值都变小，最简单的方式就是选择原图图层像素通道值和混合图层像素通道值中比较小的那个作为结果图层像素通道值，这样的图层中每个像素点的通道值都小于等于原值，图像自然就会变暗。

7.1.1 公式

$$r = \text{Darken}(b, a) = \text{Min}(b, a) \quad (7.1)$$

7.1.2 融合填充

$$r = \text{Fill}(b, a) = \text{fill} \times \text{Min}(b, a) + (1 - \text{fill}) \times b \quad (7.2)$$

7.1.3 融合不透明度

$$r = \text{Opacity}(b, a) = \text{op} \times \text{Fill}(b, a) + (1 - \text{op}) \times b \quad (7.3)$$

7.1.4 映射面和同图等效曲线

同图曲线表达式

$$r = \text{Darken}(b, b) = b \quad (7.4)$$

提示

- 这里我们简单认为 fill 和 opacity 都是 100，因为我们日常使用中几乎用不到改变这两个值并且需要模拟同图曲线的情况，使用这里只讨论最简单的类型。但是其他类型可以通过我们提供的公式自行推导，但是这里没有必要写出来。下同。

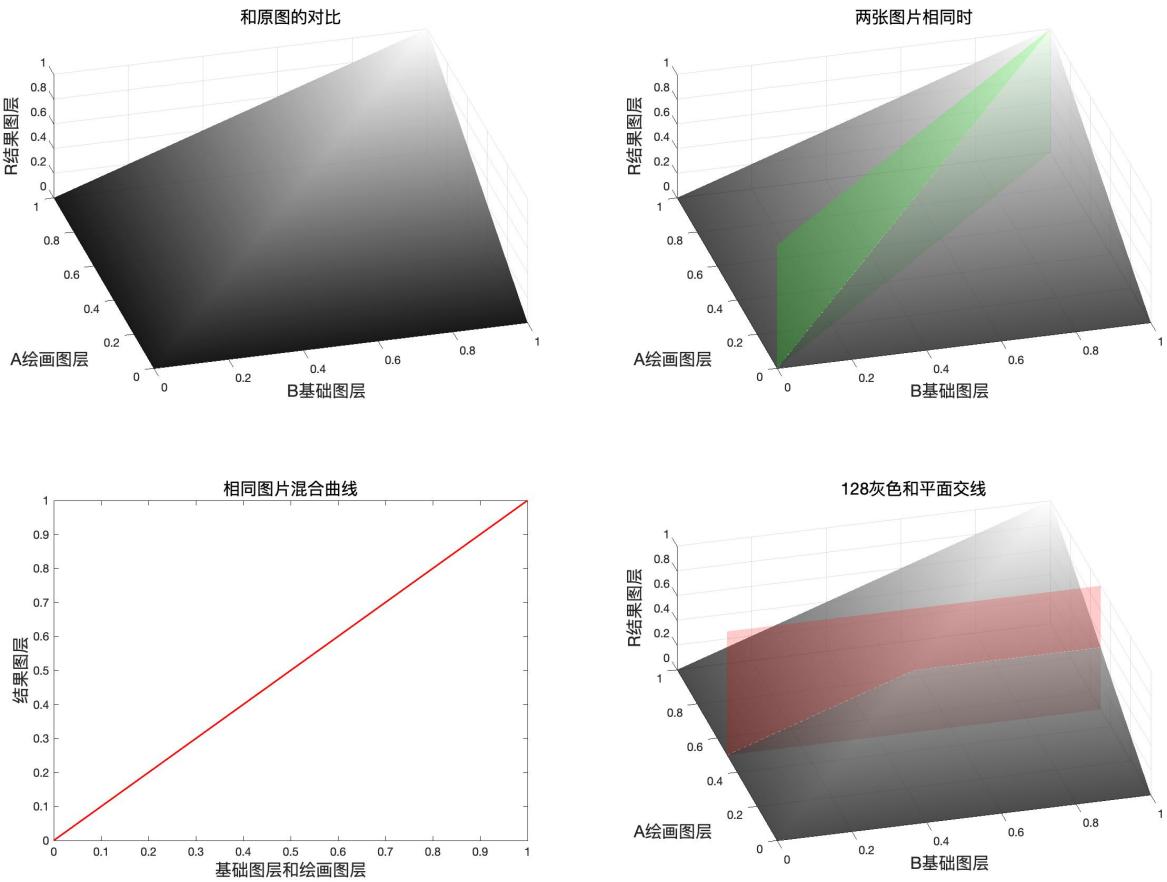


图 7.1

7.1.5 程序模拟该模式计算结果

```
// 变暗
public static BlendColor Darken(BlendColor colorBase, BlendColor colorBlend, double fill, double
    opacity) {
    double red = DarkenChannel(colorBase.red.get01Value(), colorBlend.red.get01Value(), fill);
    double green = DarkenChannel(colorBase.green.get01Value(), colorBlend.green.get01Value(), fill);
    double blue = DarkenChannel(colorBase.blue.get01Value(), colorBlend.blue.get01Value(), fill);
    return ColorUtils.Opacity(colorBase, new BlendColor(red * 255, green * 255, blue * 255), opacity);
}

private static double DarkenChannel(double base, double blend, double fill) {
    return Math.min(base, blend) * fill + (1 - fill) * base;
}
```

程序模拟结果 ✓

⚠ 变暗 (Darken) RGB[103.56, 77.60, 60.00] HSY[24.24, 43.56, 83.45] HSB[24.24, 42.06, 40.61]

7.1.6 验证

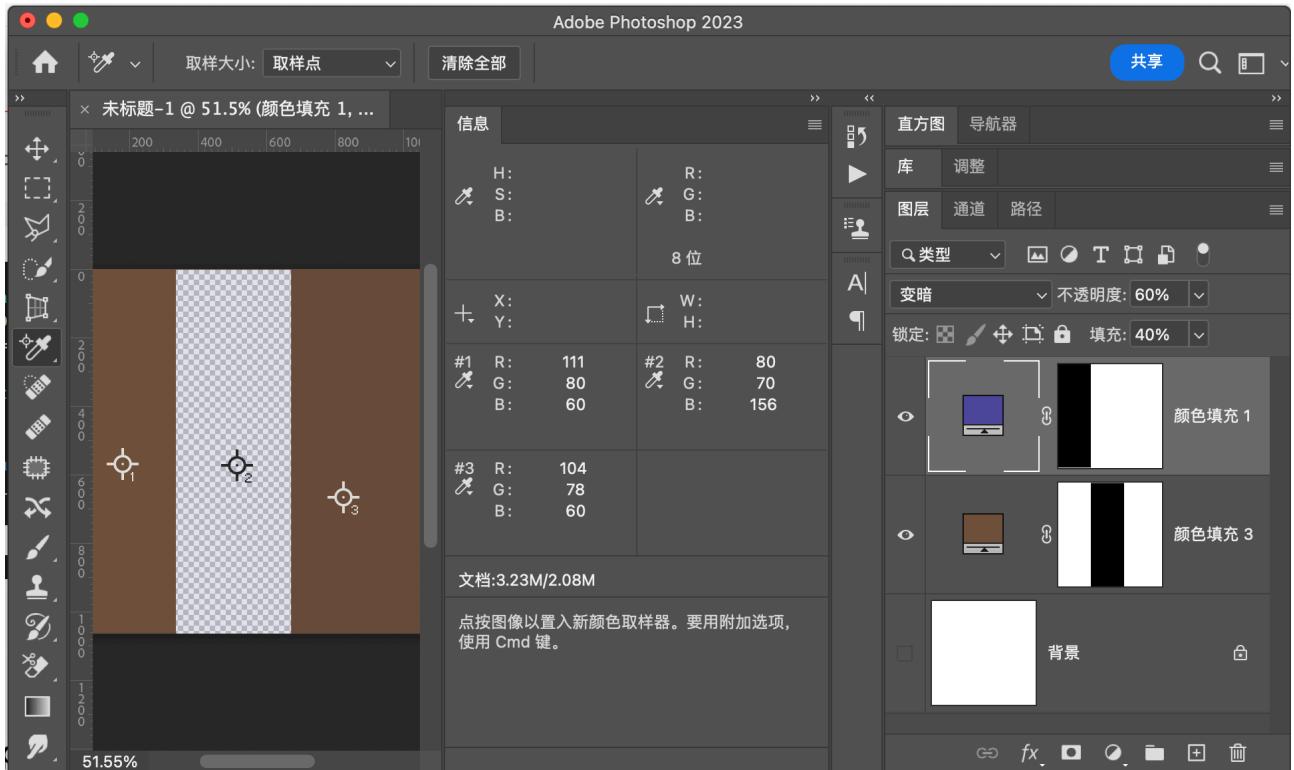


图 7.2

7.2 正片叠底 Multiply

如果将混合图层像素的通道数值和原图像素通道数值相乘，则因为归一化之后的数值都是小于等于 1 的所以，原值乘一个小于 1 的数值一定小于等于原来的值。所以图像会变暗。

7.2.1 公式

$$r = \text{Multiply}(b, a) = b \times a \quad (7.5)$$

7.2.2 融合填充

$$r = \text{Fill}(b, a) = \text{fill} \times b \times a + (1 - \text{fill}) \times b \quad (7.6)$$

7.2.3 融合不透明度

$$r = \text{Opacity}(b, a) = \text{op} \times \text{Fill}(b, a) + (1 - \text{op}) \times b \quad (7.7)$$

7.2.4 映射面和同图等效曲线

正片叠底的映射面，同图面，同图曲线和中性灰平面

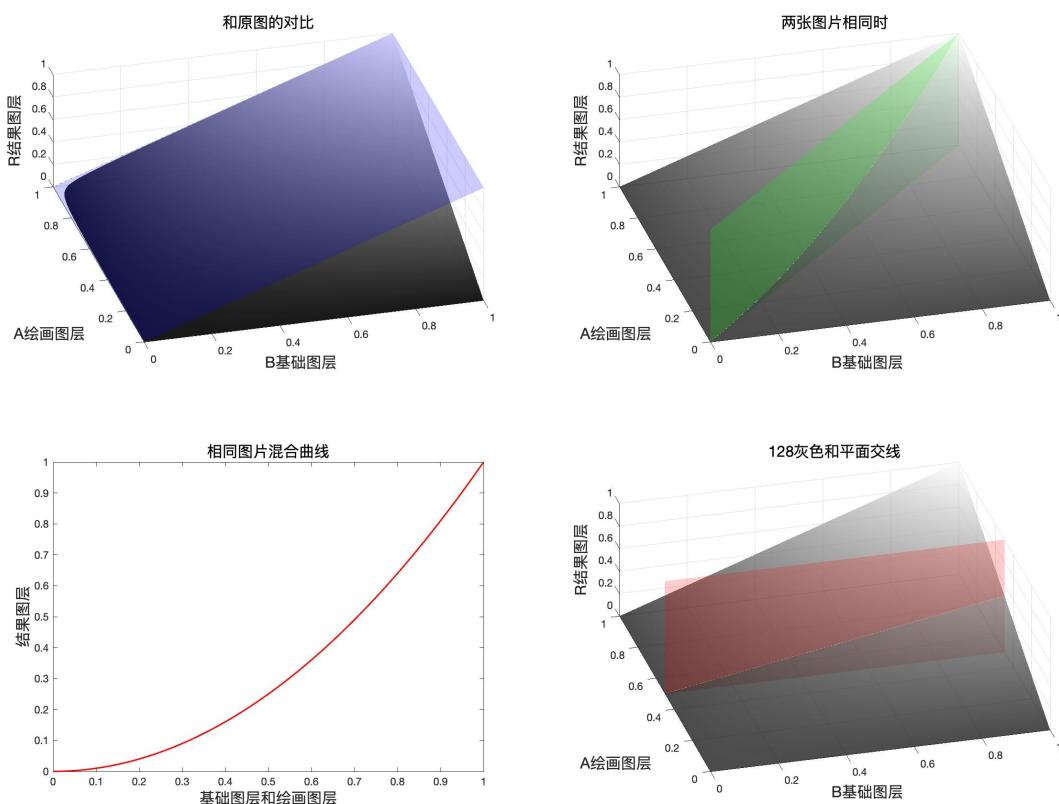


图 7.3

7.2.5 同图曲线表达式

$$r = \text{Multiply}(b, b) = b^2 \quad (7.8)$$

7.2.6 程序模拟该模式计算结果

```
// 正片叠底
public static BlendColor Mulitply(BlendColor colorBase, BlendColor colorBlend, double fill, double opacity) {
    double red = MulitplyChannel(colorBase.red.get01Value(), colorBlend.red.get01Value(), fill);
    double green = MulitplyChannel(colorBase.green.get01Value(), colorBlend.green.get01Value(), fill);
    double blue = MulitplyChannel(colorBase.blue.get01Value(), colorBlend.blue.get01Value(), fill);
    return ColorUtils.Opacity(colorBase, new BlendColor(red * 255, green * 255, blue * 255), opacity);
}

private static double MulitplyChannel(double base, double blend, double fill) {
    return ColorUtils.round((base * blend) * fill + (1 - fill) * (base), 1, 0);
}
```

程序模拟结果 ✓

正片叠底 (Mulitply) RGB[92.72, 66.07, 54.41] HSY[18.26, 38.31, 72.78] HSB[18.26, 41.32, 36.36]

7.2.7 验证

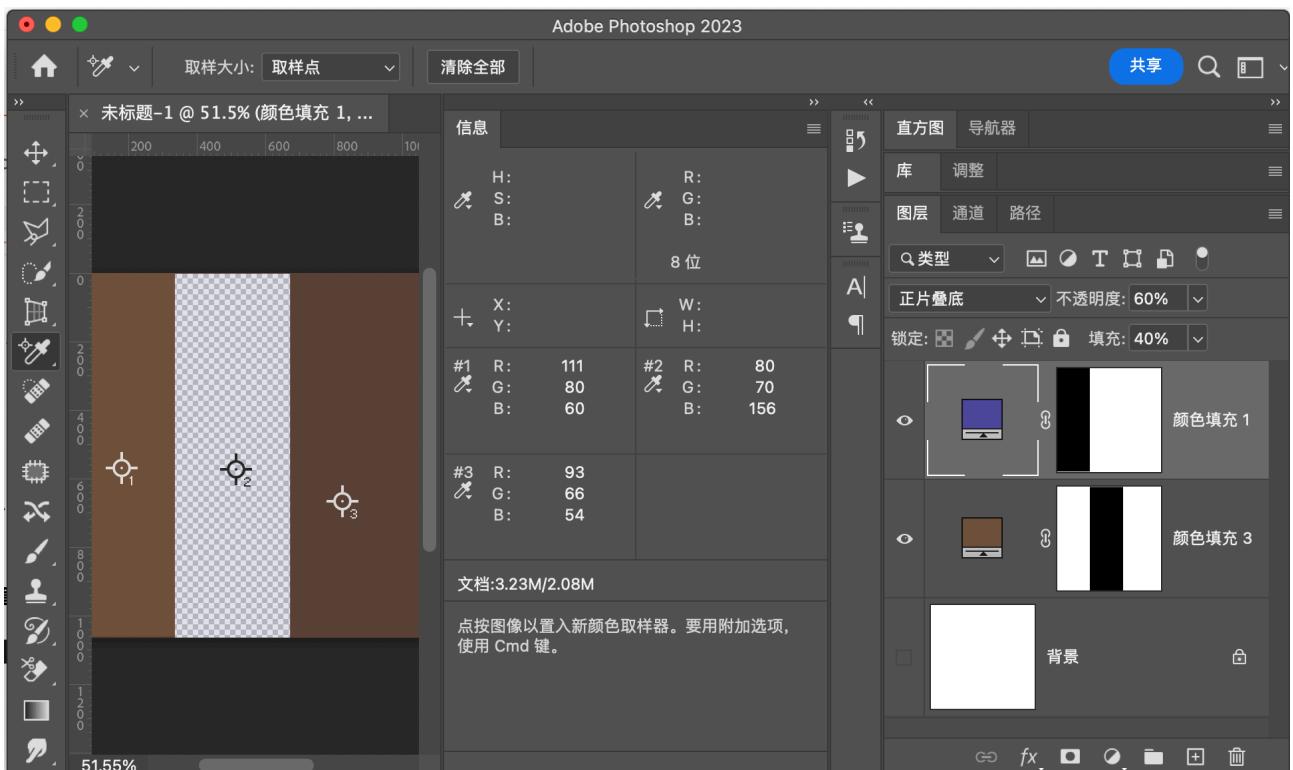


图 7.4: 正片叠底验证

7.3 线性加深 LinearBurn

此模式本质是减法，就是使用混合图层像素通道数值的补也就是附片和原图相减，如果大于1则取1小于0则取0，并且此模式需要对填充特殊处理。线性加深可以通过划分和颜色加深转化。

7.3.1 公式

$$r = \text{LinearBurn}(b, a) = b - (1 - a) = b + a - 1 \quad (7.9)$$

7.3.2 融合填充

$$r = \text{Fill}(b, a) = b - (1 - a) \times \text{fill} \quad (7.10)$$

7.3.3 融合不透明度

$$r = \text{Opacity}(b, a) = op \times \text{Fill}(b, a) + (1 - op) \times b \quad (7.11)$$

7.3.4 映射面和同图等效曲线

线性加深的映射面，同图面，同图曲线和中性灰平面

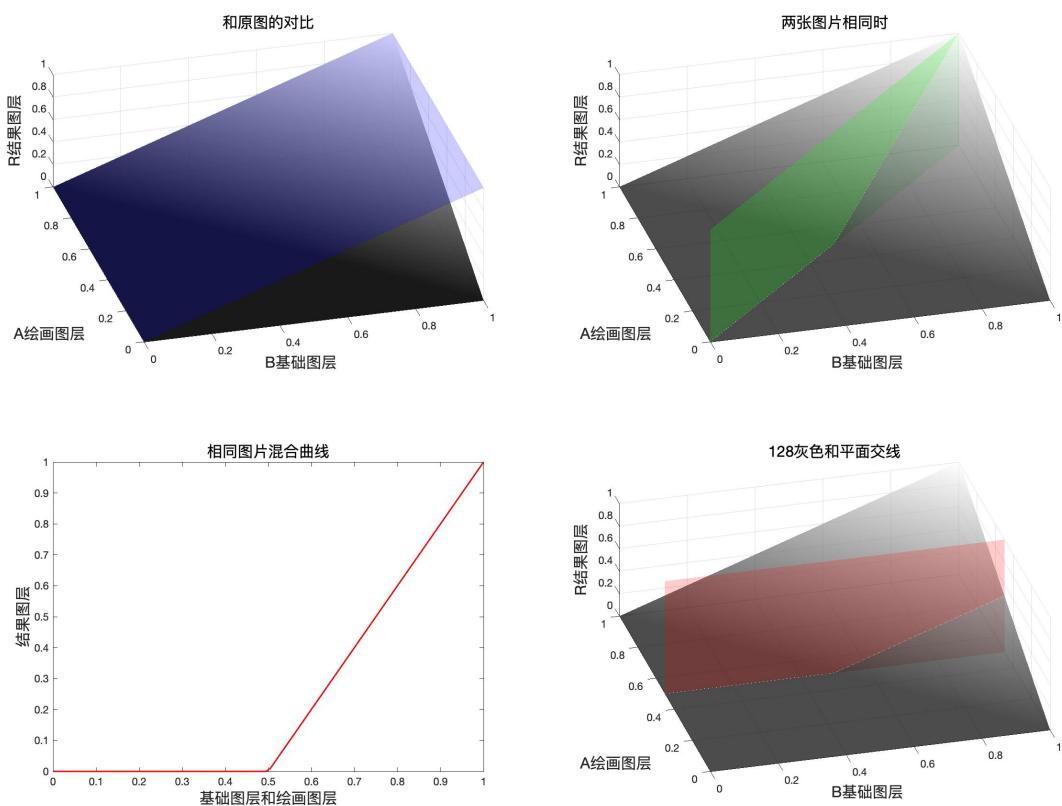


图 7.5

7.3.5 同图曲线表达式

$$r = \text{LinearBurn}(b, b) = 2 \times b - 1 \quad (7.12)$$

7.3.6 程序模拟该模式计算结果

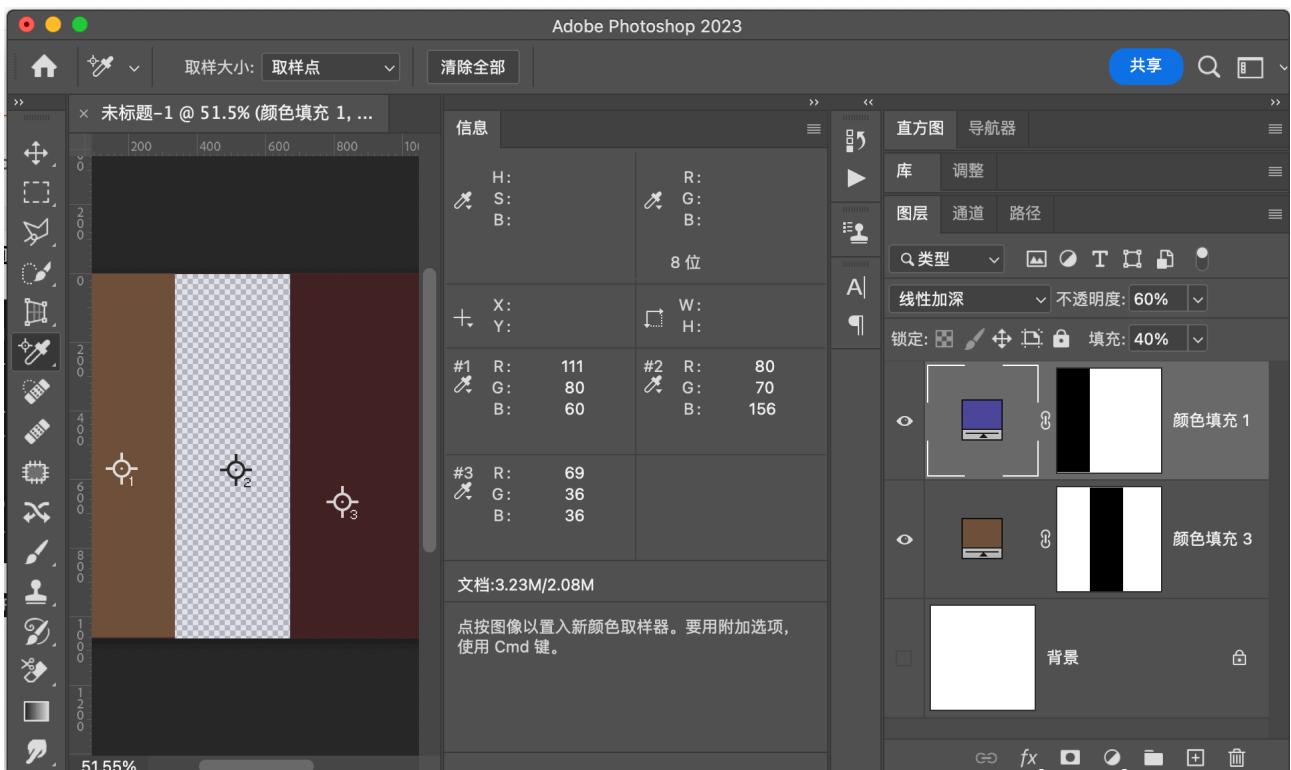
```
// 线性加深
public static BlendColor LinearBurn(BlendColor colorBase, BlendColor colorBlend, double fill, double opacity) {
    double red = LinearBurnChannel(colorBase.red.get01Value(), colorBlend.red.get01Value(), fill);
    double green = LinearBurnChannel(colorBase.green.get01Value(), colorBlend.green.get01Value(), fill);
    double blue = LinearBurnChannel(colorBase.blue.get01Value(), colorBlend.blue.get01Value(), fill);
    return ColorUtils.Opacity(colorBase, new BlendColor(red * 255, green * 255, blue * 255), opacity);
}

private static double LinearBurnChannel(double base, double blend, double fill) {
    return ColorUtils.round(base * (1 - blend) + fill, 1, 0);
}
```

程序模拟结果 ✓

线性加深 (LinearBurn) RGB[69.00, 35.60, 36.24] HSY[358.85, 33.40, 45.69] HSB[358.85, 48.41, 27.06]

7.3.7 验证



7.4 颜色加深 ColorBurn

此模式的本质是线性加深和混合图层相除，由此可以得知他可以和线性加深通过正片叠底转化。

7.4.1 公式

$$r = ColorBurn(b, a) = 1 - \frac{1 - b}{1 - (1 - a)} \quad (7.13)$$

这里之所以这么写，是因为后面要加入 fill，也就是填充。

推导一下

$$\begin{aligned} r &= ColorBurn(b, a) \\ &= 1 - \frac{1 - b}{1 - (1 - a)} = \frac{1 - (1 - a) - (1 - b)}{1 - (1 - a)} = \frac{b - (1 - a)}{1 - (1 - a)} \\ &= \frac{LinearBurn(b, a)}{1 - (1 - a)} = \frac{LinearBurn(b, a)}{a} \end{aligned} \quad (7.14)$$

提示

从这里可以看出，线性加深可以和颜色加深相互转换，通过划分和正片叠底。但是这里涉及到其他的混合模式，而非单纯负片，所以不把他们放到可以互相转化的分类中。

7.4.2 融合填充

$$r = Fill(b, a) = 1 - \frac{1 - b}{1 - (1 - a) \times fill} \quad (7.15)$$

7.4.3 融合不透明度

$$r = Opacity(b, a) = op \times Fill(b, a) + (1 - op) \times b \quad (7.16)$$

7.4.4 映射面和同图等效曲线

颜色加深的映射面，同图面，同图曲线和中性灰平面

7.4.5 同图曲线表达式

$$r = ColorBurn(b, b) = 2 - \frac{1}{b} \quad (7.17)$$

7.4.6 两次负片转划分

$$\begin{aligned} r &= 1 - ColorBurn(1 - b, a) \\ &= 1 - \left(1 - \frac{1 - (1 - b)}{a}\right) = \frac{b}{a} \end{aligned} \quad (7.18)$$

7.4.7 程序模拟该模式计算结果

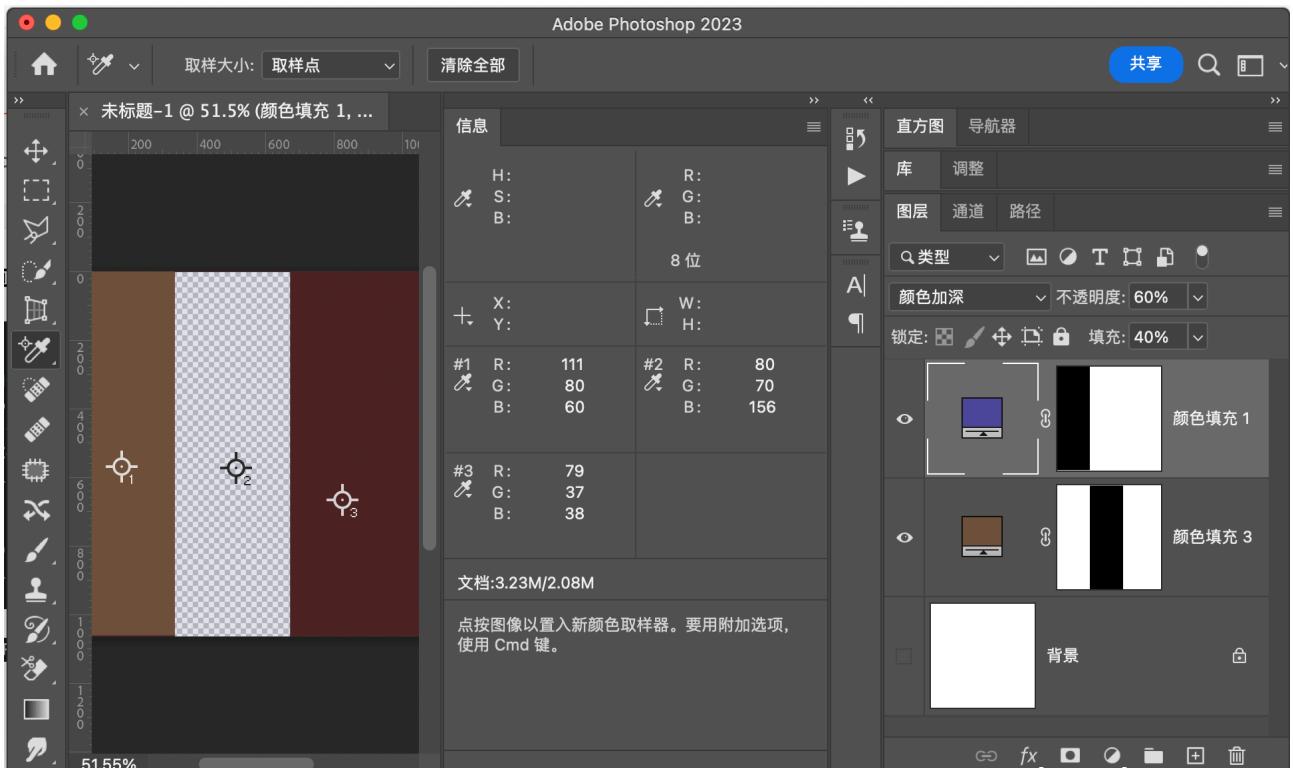
```
// 颜色加深
public static BlendColor ColorBurn(BlendColor colorBase, BlendColor colorBlend, double fill, double
    opacity) {
    double red = ColorBurnChannel(colorBase.red.get01Value(), colorBlend.red.get01Value(), fill);
    double green = ColorBurnChannel(colorBase.green.get01Value(), colorBlend.green.get01Value(), fill)
        ;
    double blue = ColorBurnChannel(colorBase.blue.get01Value(), colorBlend.blue.get01Value(), fill);
    return ColorUtils.Opacity(colorBase, new BlendColor(red * 255, green * 255, blue * 255), opacity);
}

private static double ColorBurnChannel(double base, double blend, double fill) {
    return ColorUtils.round(1 Math.min(1, (1 - base) / ((1 - (1 - blend) * fill))), 1, 0);
}
```

程序模拟结果✓

颜色加深 (ColorBurn) RGB[78.31, 37.07, 38.49] HSY[357.94, 41.24, 49.60] HSB[357.94, 52.66, 30.71]

7.4.8 验证



7.5 深色 Darker

深色模式可以理解为变暗模式的加强变暗模式，或者是粗略的变暗模式，因为其不产生新的像素，就像溶解模式一样。计算细节简单来说就是求和，比较大小，小的留下了，若求和的结果一样，就计算明度，明度小的留下了。

7.5.1 公式

$$\begin{aligned} Pix_r &= Darker(Pix_b, Pix_a) \\ &= \begin{cases} Pix_a & Sum(Pix_a) < Sum(Pix_b) \\ Pix_a & Sum(Pix_a) = Sum(Pix_b) \text{ 且 } Lum(Pix_a) > Lum(Pix_b) \\ Pix_b & Sum(Pix_a) > Sum(Pix_b) \\ Pix_b & Sum(Pix_a) = Sum(Pix_b) \text{ 且 } Lum(Pix_b) < Lum(Pix_a) \end{cases} \end{aligned} \quad (7.19)$$

提示

其中：

$$Lum(Pix) = 0.3RC + 0.59GC + 0.11BC$$

$$Sum(Pix) = RC + GC + BC$$

7.5.2 融合填充

$$r = Fill(Pix_b, Pix_a) = fill \times Pix_r + (1 - fill) \times Pix_b \quad (7.20)$$

7.5.3 融合不透明度

$$r = Opacity(Pix_b, Pix_a) = op \times Fill(Pix_b, Pix_a) + (1 - op) \times Pix_b \quad (7.21)$$

7.5.4 映射面和同图等效曲线

因为这里不是单一通道，所以不能给出单个通道的表达式。

$$Pix_r = Pix_b \quad (7.22)$$

7.5.5 程序模拟该模式计算结果

```
// 深色
public static BlendColor Darker(BlendColor colorBase, BlendColor colorBlend, double fill, double
    opacity) {
    double sumBase = colorBase.red.value + colorBase.green.value + colorBase.blue.value;
    double sumBlend = (colorBlend.red.value + colorBlend.green.value + colorBlend.blue.value) * fill;

    if (sumBase == sumBlend) {
        if (colorBase.getLum() < colorBlend.getLum()) {
            double red = colorBase.red.get01Value();
            double green = colorBase.green.get01Value();
            double blue = colorBase.blue.get01Value();
            return ColorUtils.Opacity(colorBase, new BlendColor(red * 255, green * 255, blue * 255),
                opacity);
        } else {
            double red = colorBase.red.get01Value() * (1 - fill) + colorBlend.red.get01Value() * fill;
            double green = colorBase.green.get01Value() * (1 - fill) + colorBlend.green.get01Value() *
                fill;
            double blue = colorBase.blue.get01Value() * (1 - fill) + colorBlend.blue.get01Value() * fill;
            return ColorUtils.Opacity(colorBase, new BlendColor(red * 255, green * 255, blue * 255),
                opacity);
        }
    }
    if (sumBase < sumBlend) {
        double red = colorBase.red.get01Value();
        double green = colorBase.green.get01Value();
        double blue = colorBase.blue.get01Value();
        return ColorUtils.Opacity(colorBase, new BlendColor(red * 255, green * 255, blue * 255),
            opacity);
    }
    double red = colorBase.red.get01Value() * (1 - fill) + colorBlend.red.get01Value() * fill;
    double green = colorBase.green.get01Value() * (1 - fill) + colorBlend.green.get01Value() * fill;
    double blue = colorBase.blue.get01Value() * (1 - fill) + colorBlend.blue.get01Value() * fill;
    return ColorUtils.Opacity(colorBase, new BlendColor(red * 255, green * 255, blue * 255), opacity);
}
```

程序模拟结果✓

👉 深色 (Darker) RGB[103.56, 77.60, 83.04] HSY[347.43, 25.96, 85.99] HSB[347.43, 25.07, 40.61]

7.5.6 验证

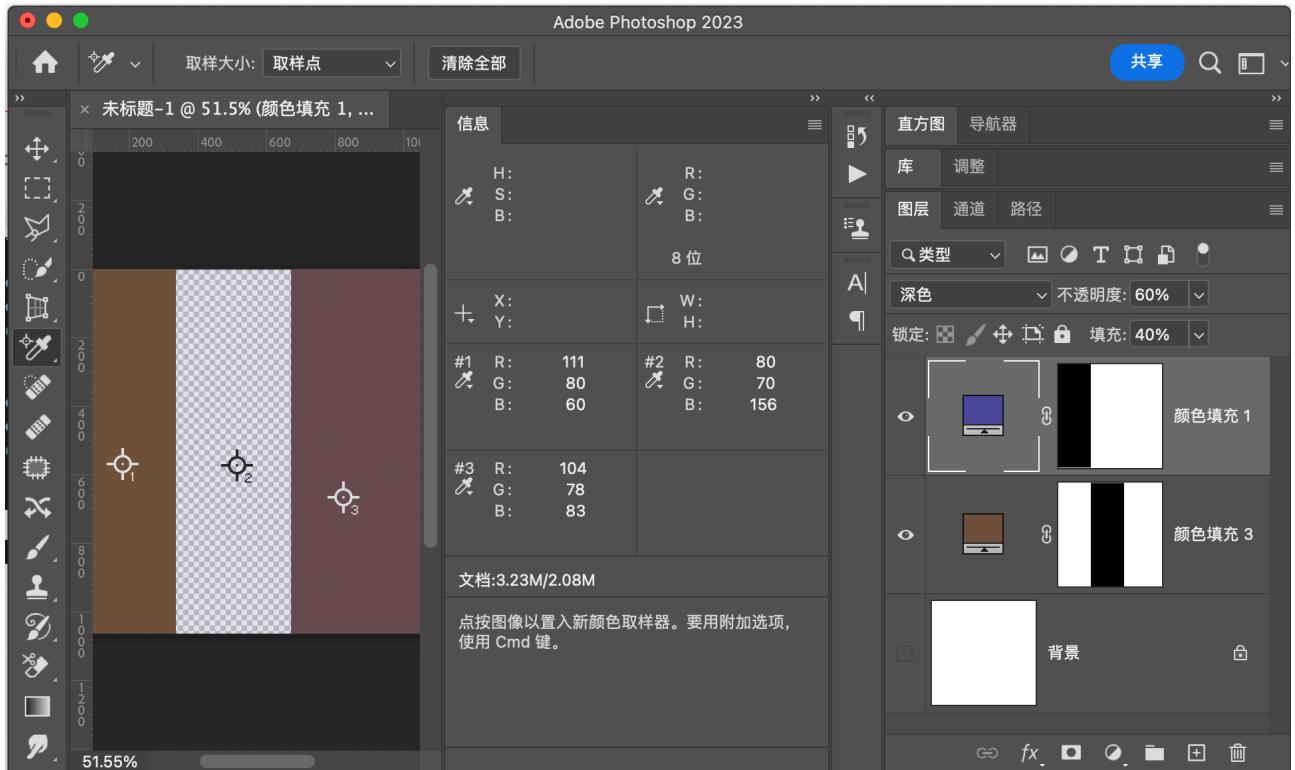


图 7.6: 深色模式验证结果

第8章 变亮组

内容提要

- 变亮组是变暗组的相反模式，并且都可以通过负片操作来实现相互转换变亮组的本质就是以混合图层的像素为参数，对原图层像素的数值进行增大，数值增大了，图片就变亮了。如果混合图层中像素点的通道值为 0，则此时该通道的计算结果和原图一致。

8.1 变亮 Lighten

变亮模式和变暗模式相反，变暗时取最小值，变亮就是取最大值，具体做法就是取原图层和混合图层像素点中三个通道各自的最大值，保留最大值组成的像素作为结果像素。

8.1.1 公式

$$r = \text{Lighten}(b, a) = \text{Max}(b, a) \quad (8.1)$$

8.1.2 融合填充

$$r = \text{Fill}(b, a) = \text{fill} \times \text{Lighten}(b, a) + (1 - \text{fill}) \times b \quad (8.2)$$

8.1.3 融合不透明度

$$r = \text{Opacity}(b, a) = \text{op} \times \text{Fill}(b, a) + (1 - \text{op}) \times b \quad (8.3)$$

8.1.4 三次负片操作相互转换

$$\begin{aligned} r &= \text{Lighten}(b, a) \\ &= 1 - \text{Min}(1 - b, 1 - a) \\ &= 1 - \text{Darken}(1 - b, 1 - a) \end{aligned} \quad (8.4)$$

也就是说，三次负片操作可以实现变暗模式和变亮模式的相互转换

8.1.5 映射面和同图等效曲线

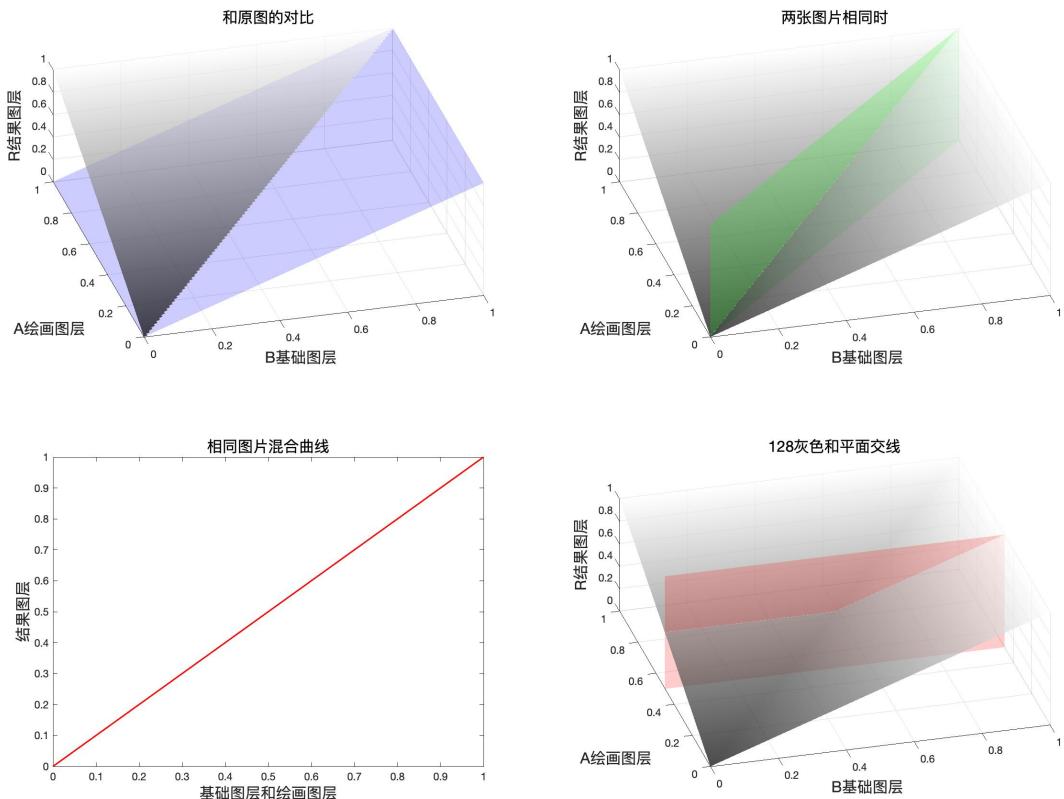


图 8.1

8.1.6 程序模拟该模式计算结果

```
// 变亮模式
public static BlendColor Lighten(BlendColor colorBase, BlendColor colorBlend, double fill, double
    opacity) {
    double red = LightenChannel(colorBase.red.get01Value(), colorBlend.red.get01Value(), fill);
    double green = LightenChannel(colorBase.green.get01Value(), colorBlend.green.get01Value(), fill);
    double blue = LightenChannel(colorBase.blue.get01Value(), colorBlend.blue.get01Value(), fill);
    return ColorUtils.Opacity(colorBase, new BlendColor(red * 255, green * 255, blue * 255), opacity);
}

private static double LightenChannel(double base, double blend, double fill) {
    return Math.max(base, blend * fill) * fill + (1 - fill) * base;
}
```

程序模拟结果 ✓

变亮 (Lighten) RGB[111.00, 80.00, 83.04] HSY[354.12, 31.00, 89.63] HSB[354.12, 27.93, 43.53]

8.1.7 验证

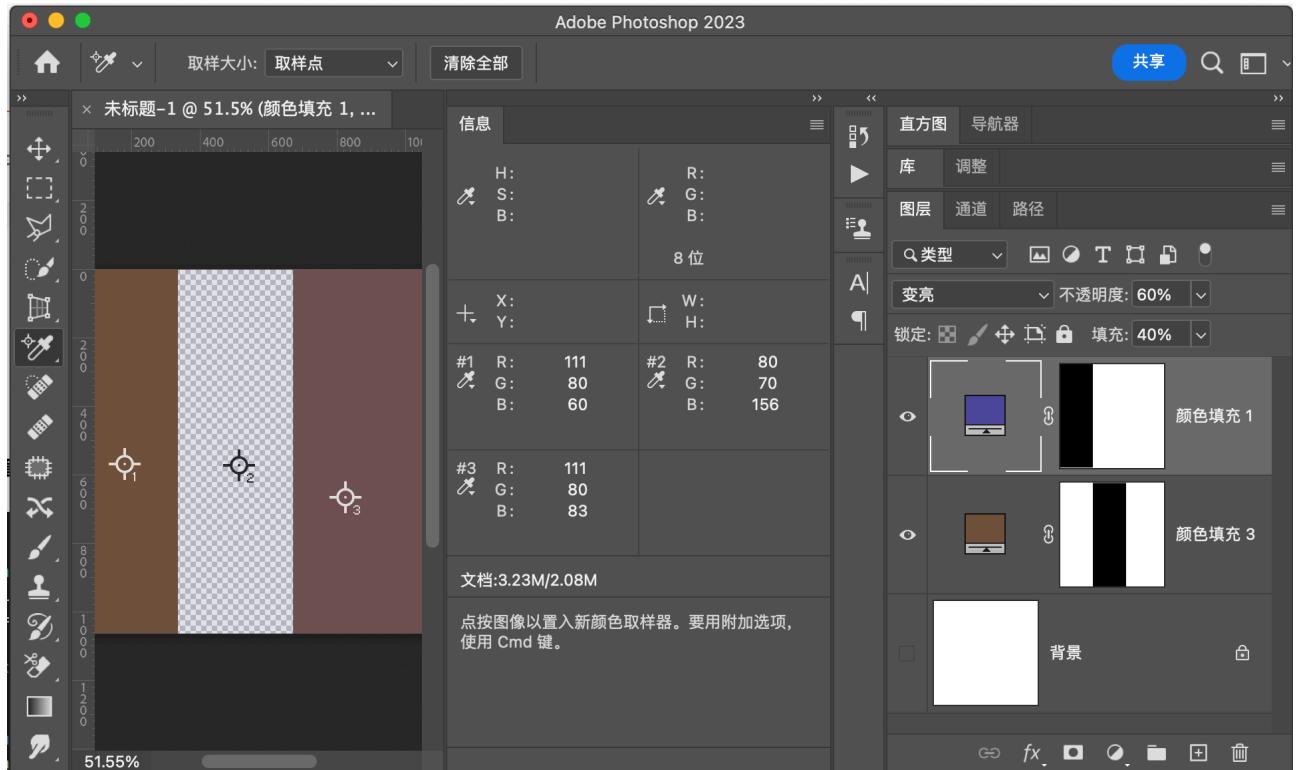


图 8.2

8.2 濾色 Screen

濾色时正片叠底的一种负片组合，和变暗变亮一样可以通过三次负片操作相互转换。

8.2.1 公式

$$r = Screen(b, a) = 1 - (1 - b)(1 - a) \quad (8.5)$$

8.2.2 融合填充

$$r = Fill(b, a) = fill \times Screen(b, a) + (1 - fill) \times b \quad (8.6)$$

8.2.3 融合不透明度

$$r = Opacity(b, a) = op \times Fill(b, a) + (1 - op) \times b \quad (8.7)$$

8.2.4 三次负片操作相互转换

$$\begin{aligned} r &= Screen(b, a) \\ &= 1 - (1 - b)(1 - a) \\ &= 1 - Multiply(1 - b, 1 - a) \end{aligned} \quad (8.8)$$

8.2.5 映射面和同图等效曲线

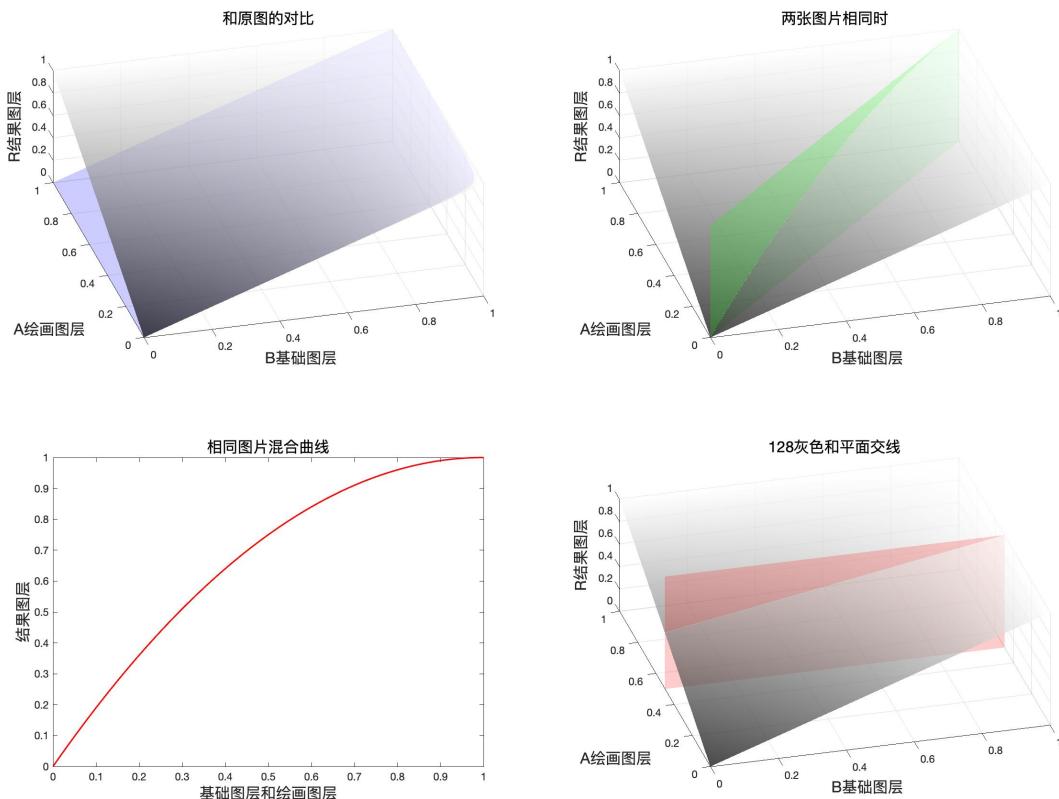


图 8.3

8.2.6 程序模拟该模式计算结果

```
// 滤色
public static BlendColor Screen(BlendColor colorBase, BlendColor colorBlend, double fill, double
    opacity) {
    double red = ScreenChannel(colorBase.red.get01Value(), colorBlend.red.get01Value(), fill);
    double green = ScreenChannel(colorBase.green.get01Value(), colorBlend.green.get01Value(), fill);
    double blue = ScreenChannel(colorBase.blue.get01Value(), colorBlend.blue.get01Value(), fill);
    return ColorUtils.Opacity(colorBase, new BlendColor(red * 255, green * 255, blue * 255), opacity);
}

private static double ScreenChannel(double base, double blend, double fill) {
    return (1 - (1 - base) * (1 - blend)) * fill + (1 - fill) * base;
}
```

程序模拟结果 ✓

滤色 (Screen) RGB[121.84, 91.53, 88.63] HSY[5.24, 33.21, 100.30] HSB[5.24, 27.26, 47.78]

8.2.7 验证

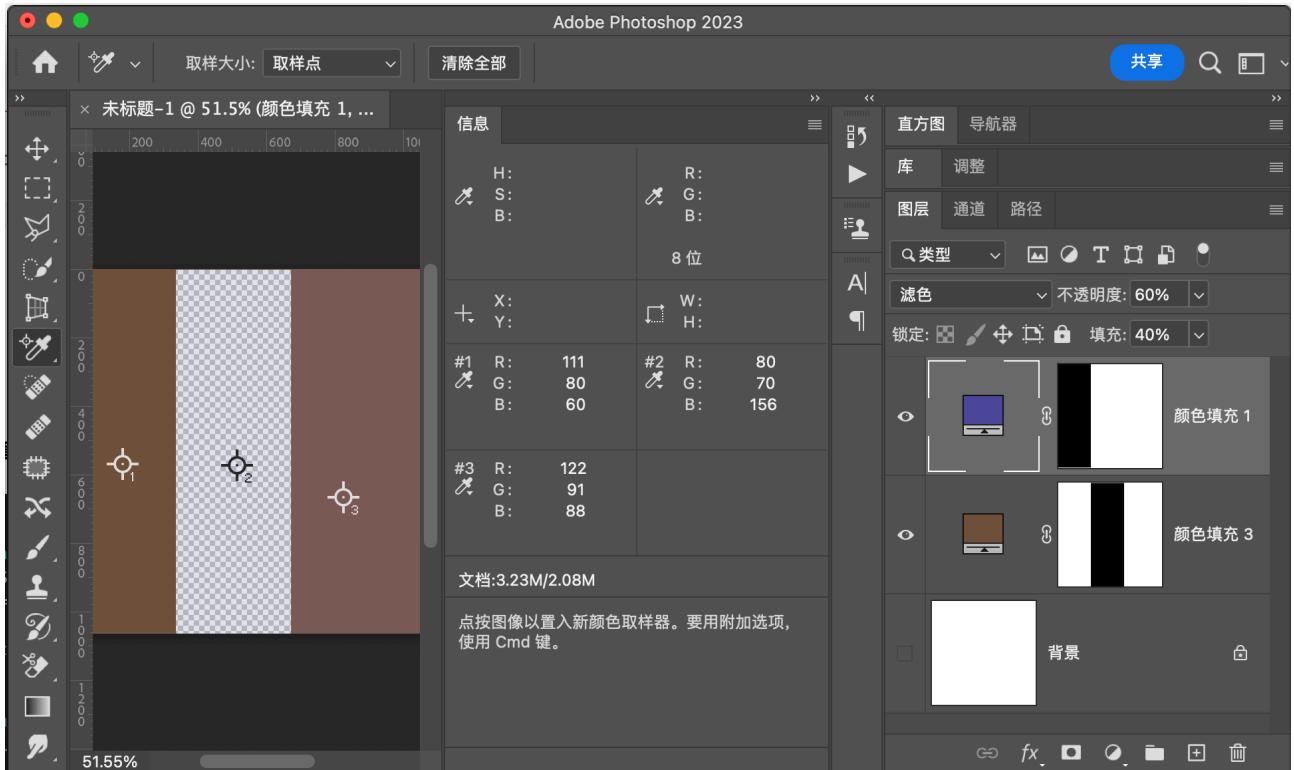


图 8.4

8.3 线性减淡 LinearDodge

线性减淡就是让原图和混合图层做加法他和线性加深也是可以互相转化的

8.3.1 公式

$$r = \text{LinearDodge}(b, a) = b + a \quad (8.9)$$

8.3.2 融合填充

$$r = \text{Fill}(b, a) = b + a \times fill \quad (8.10)$$

8.3.3 融合不透明度

$$r = \text{Opacity}(b, a) = op \times \text{Fill}(b, a) + (1 - op) \times b \quad (8.11)$$

8.3.4 三次负片操作相互转换

$$\begin{aligned} r &= \text{LinearDodge}(b, a) = 1 - \text{LinearBurn}(1 - b, 1 - a) \\ &= 1 - (1 - b + 1 - a - 1) = b + a \end{aligned} \quad (8.12)$$

8.3.5 映射面和同图等效曲线

8.3.6 程序模拟该模式计算结果

```
// 线性减淡
public static BlendColor LinearDodge(BlendColor colorBase, BlendColor colorBlend, double fill,
    double opacity) {

    double red = LinearDodgeChannel(colorBase.red.get01Value(), colorBlend.red.get01Value(), fill);
    double green = LinearDodgeChannel(colorBase.green.get01Value(), colorBlend.green.get01Value(),
        fill);
    double blue = LinearDodgeChannel(colorBase.blue.get01Value(), colorBlend.blue.get01Value(), fill);
    return ColorUtils.Opacity(colorBase, new BlendColor(red * 255, green * 255, blue * 255), opacity);
}

private static double LinearDodgeChannel(double base, double blend, double fill) {
    return ColorUtils.round(base + blend * fill, 1, 0);
}
```

程序模拟结果 ✓

↳ 线性减淡 (LinearDodge) RGB[130.20, 96.80, 97.44] HSY[358.85, 33.40, 106.89] HSB[358.85, 25.65, 51.06]

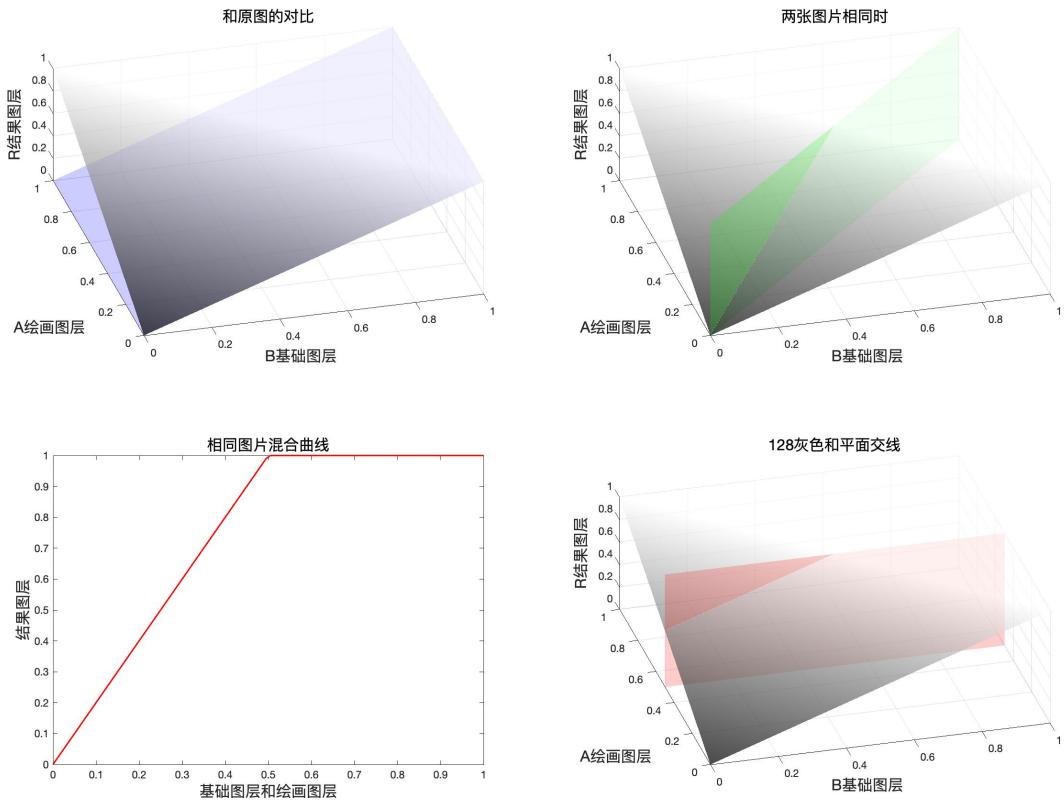


图 8.5

8.3.7 验证

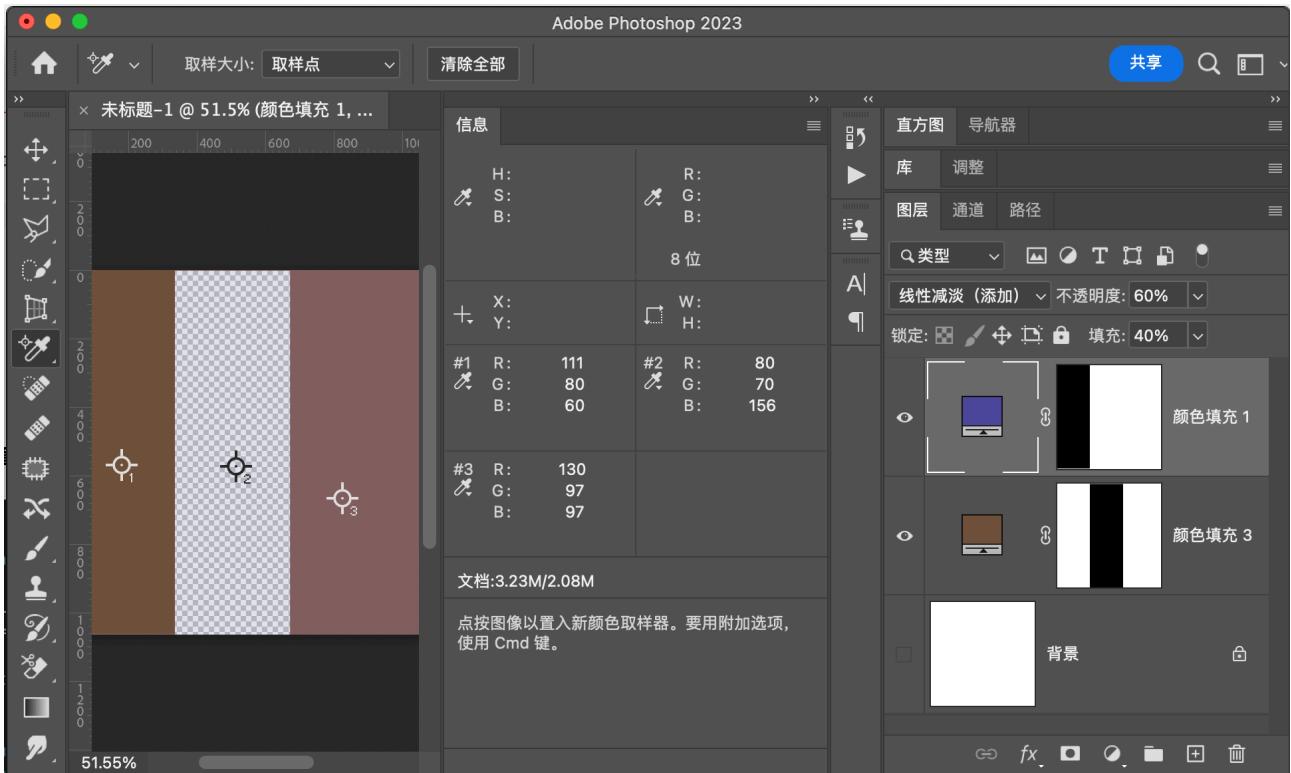


图 8.6

8.4 颜色减淡 ColorDodge

颜色减淡可以由颜色加深通过三次负片转化，颜色减淡也可以转化为划分，通过一次负片可以实现

8.4.1 公式

$$r = \text{ColorDodge}(b, a) = \frac{b}{1 - a} \quad (8.13)$$

8.4.2 融合填充

$$r = \text{Fill}(b, a) = \frac{b}{1 - a \times fill} \quad (8.14)$$

8.4.3 融合不透明度

$$r = \text{Opacity}(b, a) = op \times \text{Fill}(b, a) + (1 - op) \times b \quad (8.15)$$

8.4.4 结合负片操作和其他模式互转

结合负片操作之后，我们可以让颜色减淡和颜色加深以及划分相互转换。

8.4.4.1 三次负片操作转换为颜色加深

$$\begin{aligned} r &= \text{ColorDodge}(b, a) = 1 - \text{ColorBurn}(1 - b, 1 - a) \\ &= 1 - \left(1 - \frac{1 - (1 - b)}{1 - (1 - (1 - a))}\right) = \frac{b}{1 - a} \end{aligned} \quad (8.16)$$

8.4.4.2 一次负片转为划分

$$\begin{aligned} r &= \text{ColorDodge}(b, 1 - a) \\ &= \frac{b}{1 - 1 + a} = \frac{b}{a} \end{aligned} \quad (8.17)$$

8.4.5 映射面和同图等效曲线

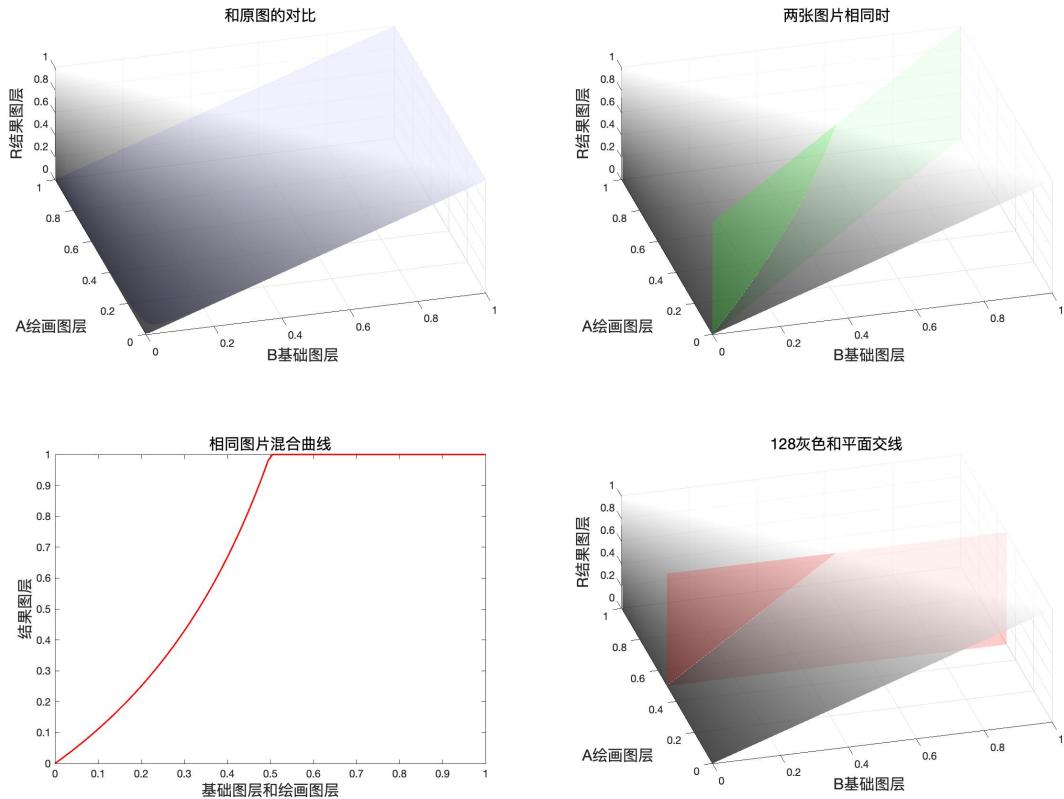


图 8.7

8.4.6 程序模拟该模式计算结果

```
// 颜色减淡
public static BlendColor ColorDodge(BlendColor colorBase, BlendColor colorBlend, double fill, double
    opacity) {
    double red = ColorDodgeChannel(colorBase.red.get01Value(), colorBlend.red.get01Value(), fill);
    double green = ColorDodgeChannel(colorBase.green.get01Value(), colorBlend.green.get01Value(), fill
    );
    double blue = ColorDodgeChannel(colorBase.blue.get01Value(), colorBlend.blue.get01Value(), fill);
    return ColorUtils.Opacity(colorBase, new BlendColor(red * 255, green * 255, blue * 255), opacity);
}

private static double ColorDodgeChannel(double base, double blend, double fill) {
    return ColorUtils.round((base) / ((1 - blend * fill)), 1, 0);
}
```

程序模拟结果✓

颜色减淡 (ColorDodge) RGB[120.56, 85.92, 71.66] HSY[17.50, 48.89, 94.74] HSB[17.50, 40.56, 47.28]

8.4.7 验证

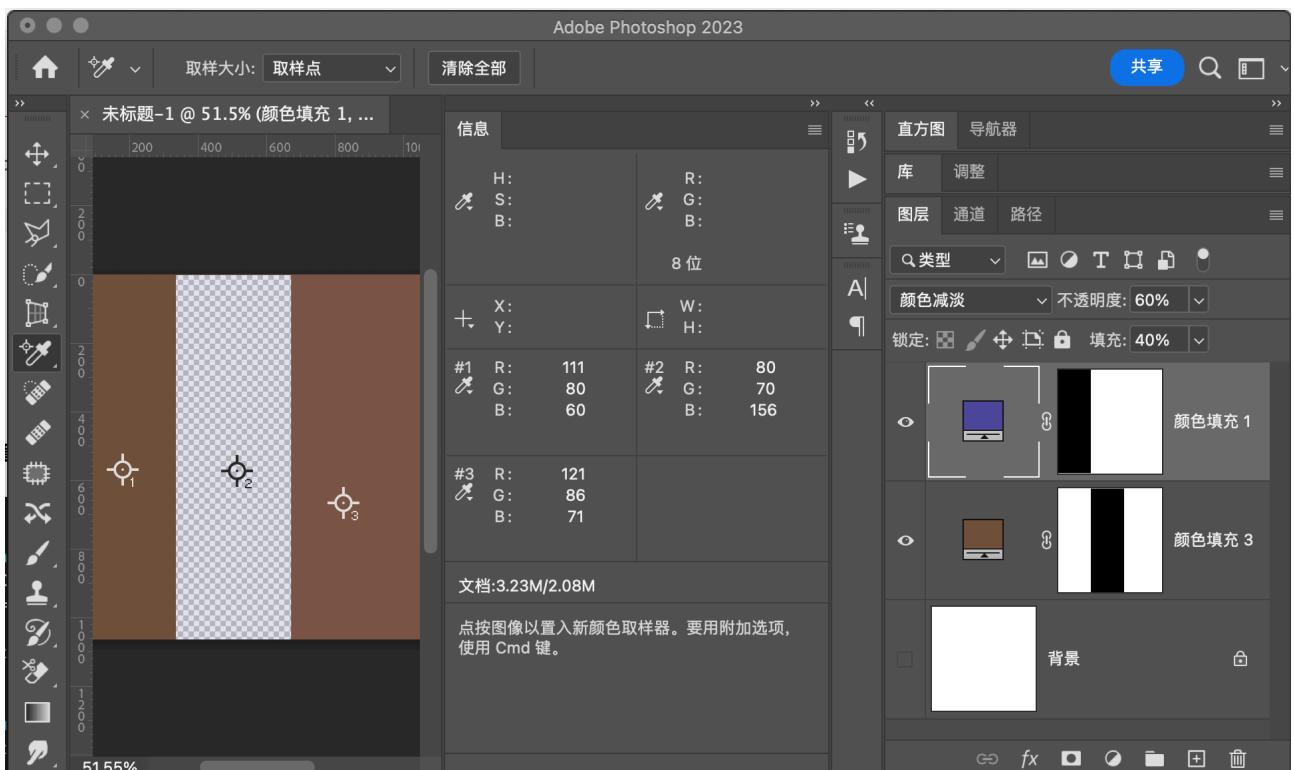


图 8.8

8.4.8 用途示例

1: 组合成为亮光模式

2: 制造光线, 同时保护暗部细节

8.5 浅色 Lighter

浅色模式是深色模式的相反模式，可以通过深色三次负片操作得到计算细节简单来说就是求和，比较大小，大的留下了，若求和的结果一样，就计算明度，明度大的留下了。

8.5.1 公式

$$\begin{aligned} Pix_r &= \text{Lighter}(Pix_b, Pix_a) \\ &= \begin{cases} Pix_a & \text{Sum}(Pix_a) > \text{Sum}(Pix_b) \\ Pix_a & \text{Sum}(Pix_a) = \text{Sum}(Pix_b) \text{ 且 } \text{Lum}(Pix_a) > \text{Lum}(Pix_b) \\ Pix_b & \text{Sum}(Pix_b) > \text{Sum}(Pix_a) \\ Pix_b & \text{Sum}(Pix_a) = \text{Sum}(Pix_b) \text{ 且 } \text{Lum}(Pix_b) > \text{Lum}(Pix_a) \end{cases} \end{aligned} \quad (8.18)$$

提示

其中：

$$\begin{aligned} \text{Lum}(Pix) &= 0.3RC + 0.59GC + 0.11BC \\ \text{Sum}(Pix) &= RC + GC + BC \end{aligned}$$

8.5.2 融合填充

$$r = \text{Fill}(Pix_b, Pix_a) = \text{fill} \times Pix_r + (1 - \text{fill}) \times Pix_b \quad (8.19)$$

8.5.3 融合不透明度

$$r = \text{Opacity}(Pix_b, Pix_a) = \text{op} \times \text{Fill}(Pix_b, Pix_a) + (1 - \text{op}) \times Pix_b \quad (8.20)$$

8.5.4 三次负片操作相互转换

$$\begin{aligned} Pix_r &= 1 - \text{Darker}(1 - Pix_b, 1 - Pix_a) \\ &= \begin{cases} 1 - (1 - Pix_a) & \text{Sum}(1 - Pix_a) < \text{Sum}(1 - Pix_b) \\ 1 - (1 - Pix_b) & \text{Sum}(1 - Pix_a) > \text{Sum}(1 - Pix_b) \\ 1 - (1 - Pix_b) & \text{Sum}(1 - Pix_a) = \text{Sum}(1 - Pix_b) \text{ 且 } \text{Lum}(1 - Pix_b) < \text{Lum}(1 - Pix_a) \\ 1 - (1 - Pix_a) & \text{Sum}(1 - Pix_a) = \text{Sum}(1 - Pix_b) \text{ 且 } \text{Lum}(1 - Pix_b) > \text{Lum}(1 - Pix_a) \end{cases} \\ &= \begin{cases} Pix_a & \text{Sum}(Pix_a) > \text{Sum}(Pix_b) \\ Pix_b & \text{Sum}(Pix_a) < \text{Sum}(Pix_b) \\ Pix_b & \text{Sum}(Pix_a) = \text{Sum}(Pix_b) \text{ 且 } \text{Lum}(Pix_b) > \text{Lum}(Pix_a) \\ Pix_a & \text{Sum}(Pix_a) = \text{Sum}(Pix_b) \text{ 且 } \text{Lum}(Pix_b) < \text{Lum}(Pix_a) \end{cases} \\ &= \text{Lighter}(Pix_b, Pix_a) \end{aligned} \quad (8.21)$$

8.5.5 程序模拟该模式计算结果

```
// 浅色
public static BlendColor Lighter(BlendColor colorBase, BlendColor colorBlend, double fill, double
    opacity) {
    double sumBase = colorBase.red.value + colorBase.green.value + colorBase.blue.value;
    double sumBlend = (colorBlend.red.value + colorBlend.green.value + colorBlend.blue.value);
    if (sumBase == sumBlend) {
        if (colorBase.getLum() > colorBlend.getLum()) {
            double red = colorBase.red.get01Value();
            double green = colorBase.green.get01Value();
            double blue = colorBase.blue.get01Value();
            return ColorUtils.Opacity(colorBase, new BlendColor(red * 255, green * 255, blue * 255),
                opacity);
        } else {
            double red = colorBase.red.get01Value() * (1 - fill) + colorBlend.red.get01Value() * fill;
            double green = colorBase.green.get01Value() * (1 - fill) + colorBlend.green.get01Value() * fill
                ;
            double blue = colorBase.blue.get01Value() * (1 - fill) + colorBlend.blue.get01Value() * fill;
            return ColorUtils.Opacity(colorBase, new BlendColor(red * 255, green * 255, blue * 255),
                opacity);
        }
    }
    if (sumBase > sumBlend) {
        double red = colorBase.red.get01Value();
        double green = colorBase.green.get01Value();
        double blue = colorBase.blue.get01Value();
        return ColorUtils.Opacity(colorBase, new BlendColor(red * 255, green * 255, blue * 255), opacity)
            ;
    }
    double red = colorBase.red.get01Value() * (1 - fill) + colorBlend.red.get01Value() * fill;
    double green = colorBase.green.get01Value() * (1 - fill) + colorBlend.green.get01Value() * fill;
    double blue = colorBase.blue.get01Value() * (1 - fill) + colorBlend.blue.get01Value() * fill;
    return ColorUtils.Opacity(colorBase, new BlendColor(red * 255, green * 255, blue * 255), opacity);
}
```

程序模拟结果✓

👉 浅色 (Lighter) RGB[111.00, 80.00, 60.00] HSY[23.53, 51.00, 87.10] HSB[23.53, 45.95, 43.53]

8.5.6 验证

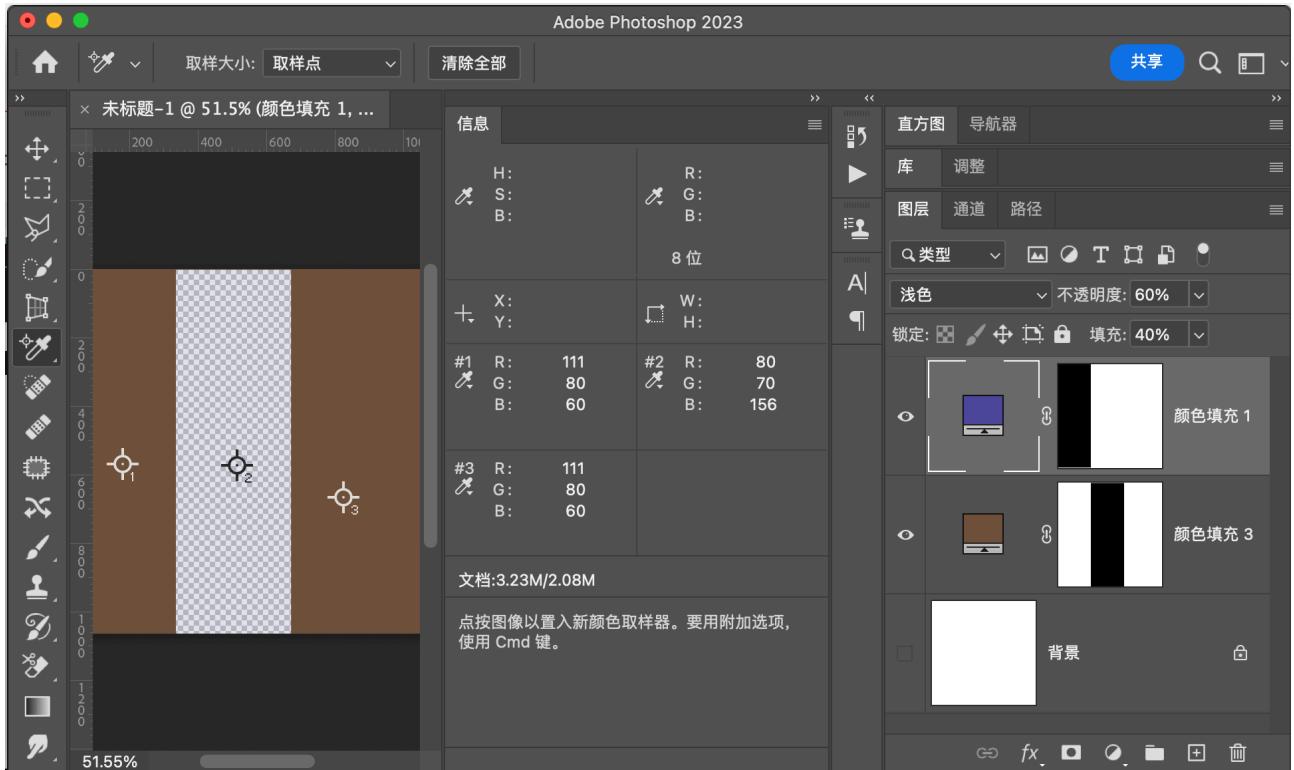


图 8.9

第9章 对比度组

内容提要

- 这一组都是相反的混合模式的组合，基本都是让暗的变暗，亮的变亮，所以这一组被称为对比度组，组合的分割界限是基础图层或者混合图层中性灰平面

9.1 叠加 Overlay

此模式是正片叠底和滤色的组合，组合依据是底图的中性灰平面，如果在 [0, 128] 则使用正片叠底，若是在 (128, 255] 之间，则是实用滤色。

9.1.1 公式

$$r = OverLay(b, a) = \begin{cases} Multiply(2b, a) & 0 \leq b \leq 0.5 \\ Screen(2(b - 0.5), a) & 0.5 < b \leq 1 \end{cases} = \begin{cases} 2ba & 0 \leq b \leq 0.5 \\ 1 - 2(1 - b)(1 - a) & 0.5 < b \leq 1 \end{cases} \quad (9.1)$$

和填充还有不透明度的关系，以及公式可以参考正片叠底和滤色

9.1.2 映射面和同图等效曲线

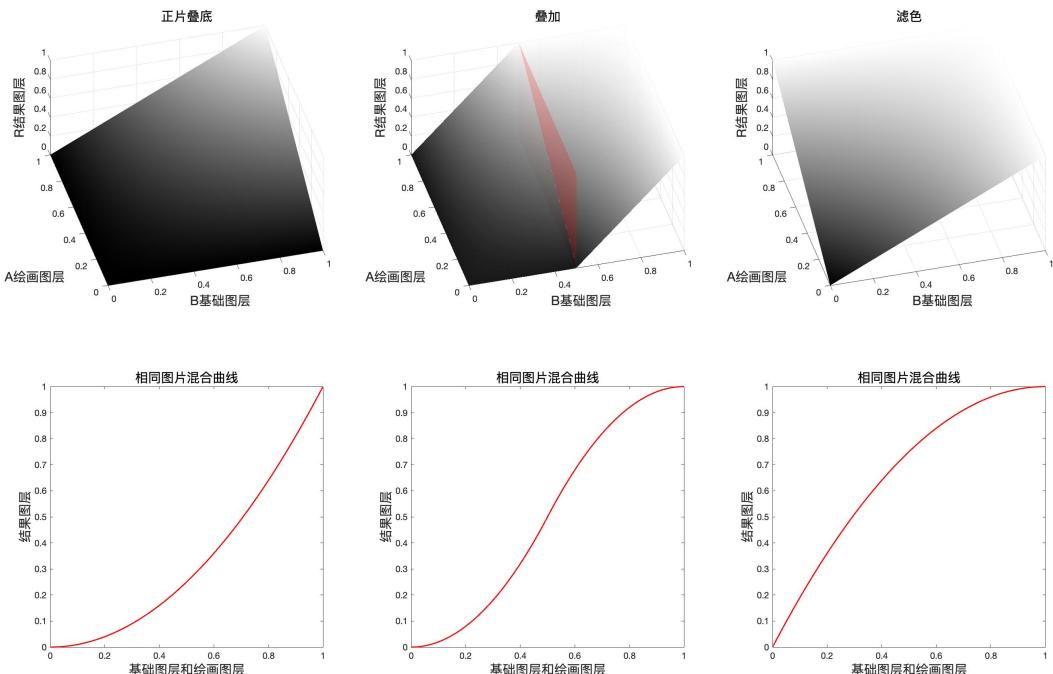


图 9.1

9.1.3 程序模拟该模式计算结果

```
// 叠加
public static BlendColor Overlay(BlendColor colorBase, BlendColor colorBlend, double fill, double
    opacity) {
    double red = OverlayChannel(colorBase.red.get01Value(), colorBlend.red.get01Value(), fill);
    double green = OverlayChannel(colorBase.green.get01Value(), colorBlend.green.get01Value(), fill);
    double blue = OverlayChannel(colorBase.blue.get01Value(), colorBlend.blue.get01Value(), fill);
    return ColorUtils.Opacity(colorBase, new BlendColor(red * 255, green * 255, blue * 255), opacity);
}

private static double OverlayChannel(double baseValue, double blendValue, double fill) {
    if (baseValue <= 0.5) {
        return MulitplyChannel(baseValue, 2 * blendValue, fill);
    } else {
        return ScreenChannel(baseValue, 2 * (blendValue - 0.5), fill);
    }
}
```

程序模拟结果✓

⚠ 叠加 (Overlay) RGB[101.08, 71.34, 63.22] HSY[12.87, 37.86, 79.37] HSB[12.87, 37.45, 39.64]

9.1.4 验证

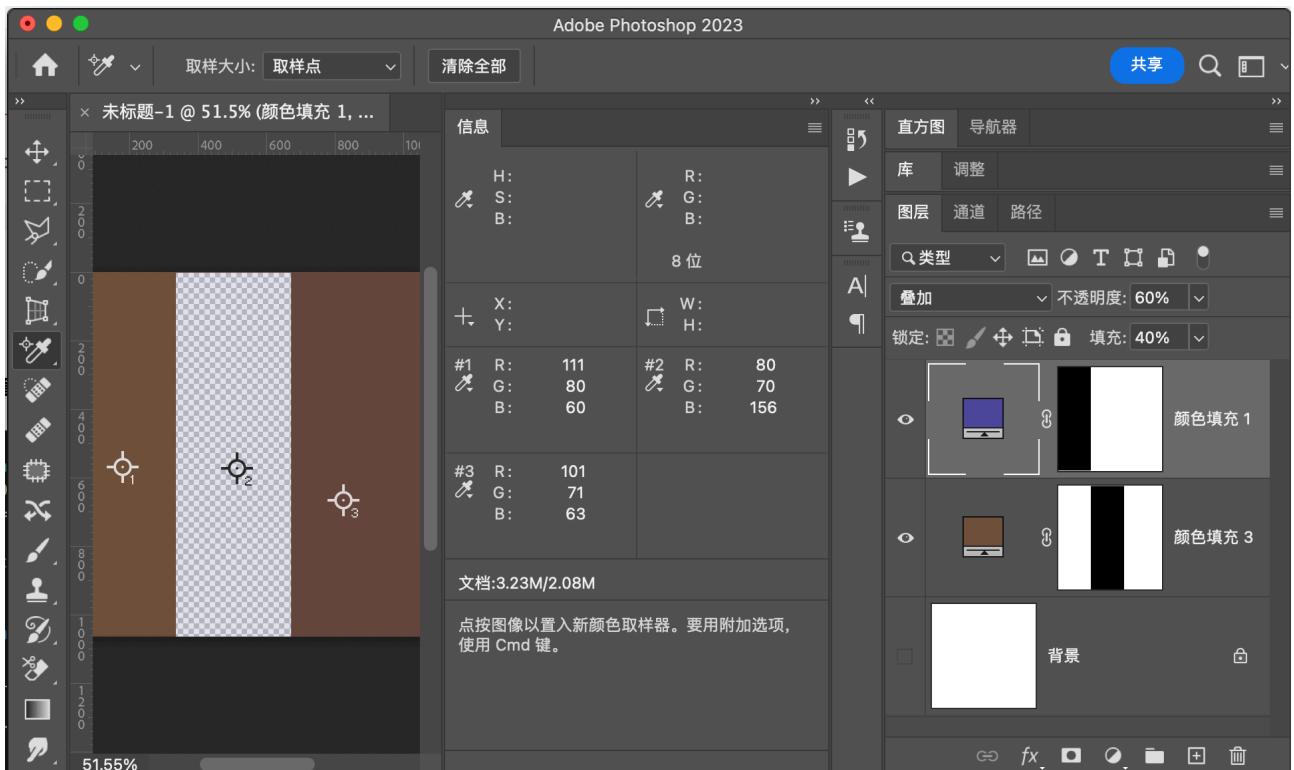


图 9.2

9.1.5 用途示例

- 1: 同图混合增加图片对比度
- 2: 和中性灰混合，达到局部提亮或者压黑

9.2 柔光 SoftLight

柔光模式是最复杂的一种混合模式，也是最巧妙的一种混合模式，柔光模式的本质是伽马矫正 (gamma correction)。配合混合图层的像素点的通道数值，再对原图层使用伽马矫正，二者通过配合就可以得到柔光模式。

9.3 伽马矫正

如果我们想了解柔光模式，首先必须了解什么是伽马矫正，伽马矫正，简单来说就是，将原像素通道数值通过幂次方的方式进行修改，比如平方和根号，例如我们由一个归一化之后为 0.5 的通道数值，我们对其进行系数为 2 的伽马矫正，则结果是 $0.5^{\frac{1}{2}} = \sqrt{0.5}$ ，如果进行系数为 $\frac{1}{2}$ 的伽马矫正，则结果为 0.5^2

伽马矫正的数学表达式 $output = input^{\frac{1}{gama}}$

其中 $input$ 代表输入信号， $output$ 代表输出， $gama$ 代表伽马系数

系数为 2 的伽马矫正 $output = input^{\frac{1}{2}}$

系数为 $\frac{1}{2}$ 的伽马矫正 $output = input^2$

9.3.1 公式

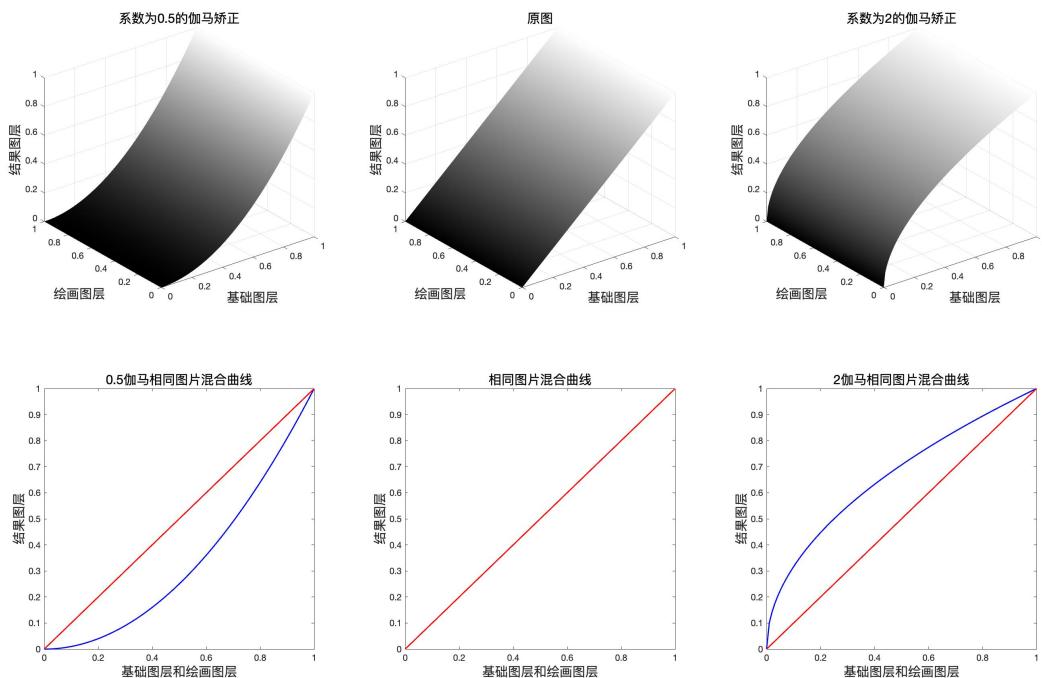


图 9.3

于是对于系数为 $\frac{1}{2}$ 的伽马矫正，稍微变换一下表达式

$$r = b^2 = (b - b^2) + b \quad (9.2)$$

然后再使用 $(2a - 1)$ 作为系数乘以差值项

$$r = (2a - 1)(b - b^2) + b \quad (9.3)$$

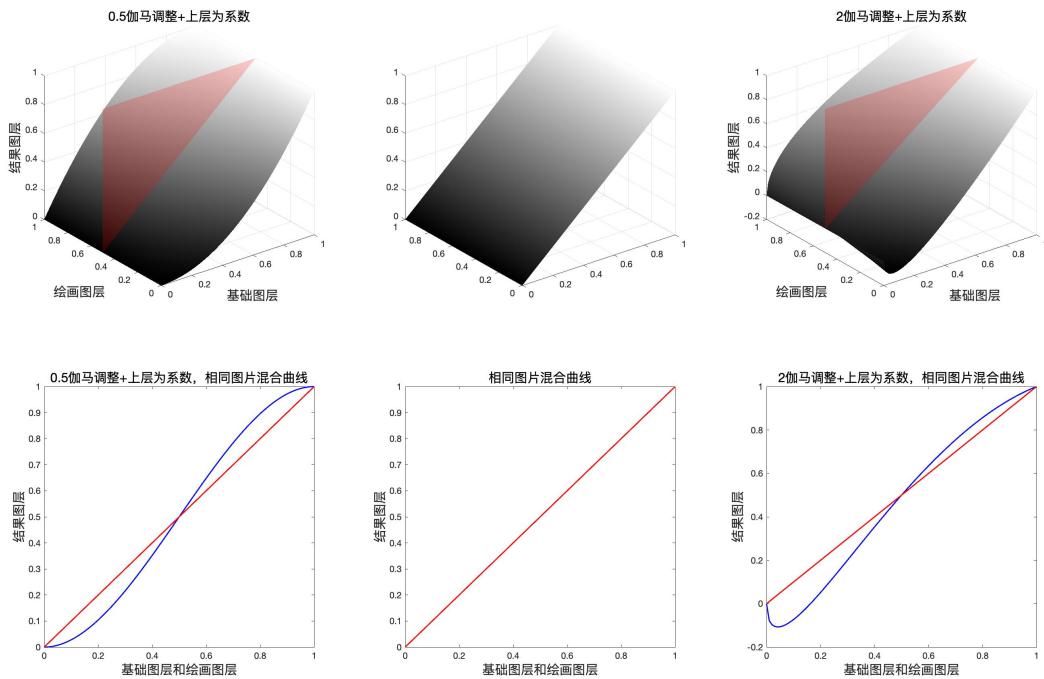


图 9.4

于是对于系数为 2 的伽马矫正，稍微变换一下表达式

$$r = \sqrt{b} = (\sqrt{b} - b) + b \quad (9.4)$$

然后再使用 $(2a - 1)$ 作为系数乘以差值项

$$r = (2a - 1)(\sqrt{b} - b) + b \quad (9.5)$$

再将结果合并，我们就可以得到柔光模式的表达式。

$$r = SoftLight(b, a) = \begin{cases} (2a - 1)(b^2 - b) + b & a \leq 0.5 \\ (2a - 1)(\sqrt{b} - b) + b & a > 0.5 \end{cases} \quad (9.6)$$

提示

- 如果使用一句话概括柔光模式的数学表达式，就是“以混合图层为系数的系数为 $\frac{1}{2}$ 和 2 的伽马矫正”
- 在 PS 中伽马矫正可以在色阶工具和曝光度工具中找到

9.3.2 映射面和同图等效曲线

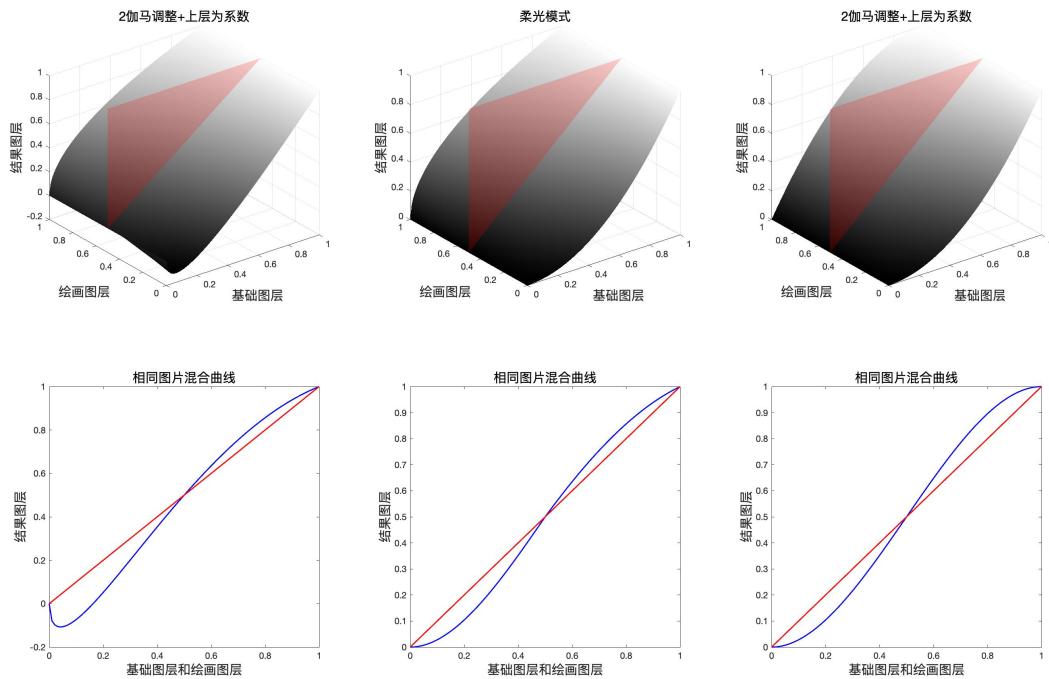


图 9.5

9.3.3 程序模拟该模式计算结果

```
// 柔光
public static BlendColor SoftLight(BlendColor colorBase, BlendColor colorBlend, double fill, double
    opacity) {
    double red = SoftLightChannel(colorBase.red.get01Value(), colorBlend.red.get01Value(), fill);
    double green = SoftLightChannel(colorBase.green.get01Value(), colorBlend.green.get01Value(), fill)
        ;
    double blue = SoftLightChannel(colorBase.blue.get01Value(), colorBlend.blue.get01Value(), fill);
    return ColorUtils.Opacity(colorBase, new BlendColor(red * 255, green * 255, blue * 255), opacity);
}

private static double SoftLightChannel(double baseValue, double blendValue, double fill) {
    if (blendValue <= 0.5) {
        return (baseValue + (2 * blendValue - 1) * (baseValue - baseValue * baseValue)) * fill
            + (1 - fill) * baseValue;
    } else {
        return (baseValue + (2 * blendValue - 1) * (SoftLightChannelSub(baseValue) - baseValue)) * fill
            + (1 - fill) * baseValue;
    }
}

private static double SoftLightChannelSub(double value) {
    if (value <= 0.25) {

```

```
return ((16 * value - 12) * value + 4) * value;
} else {
    return Math.sqrt(value);
}
}
```

程序模拟结果 ✓

柔光 (SoftLight) RGB[105.40, 74.06, 63.42] HSY[15.21, 41.98, 82.29] HSB[15.21, 39.83, 41.33]

9.3.4 验证

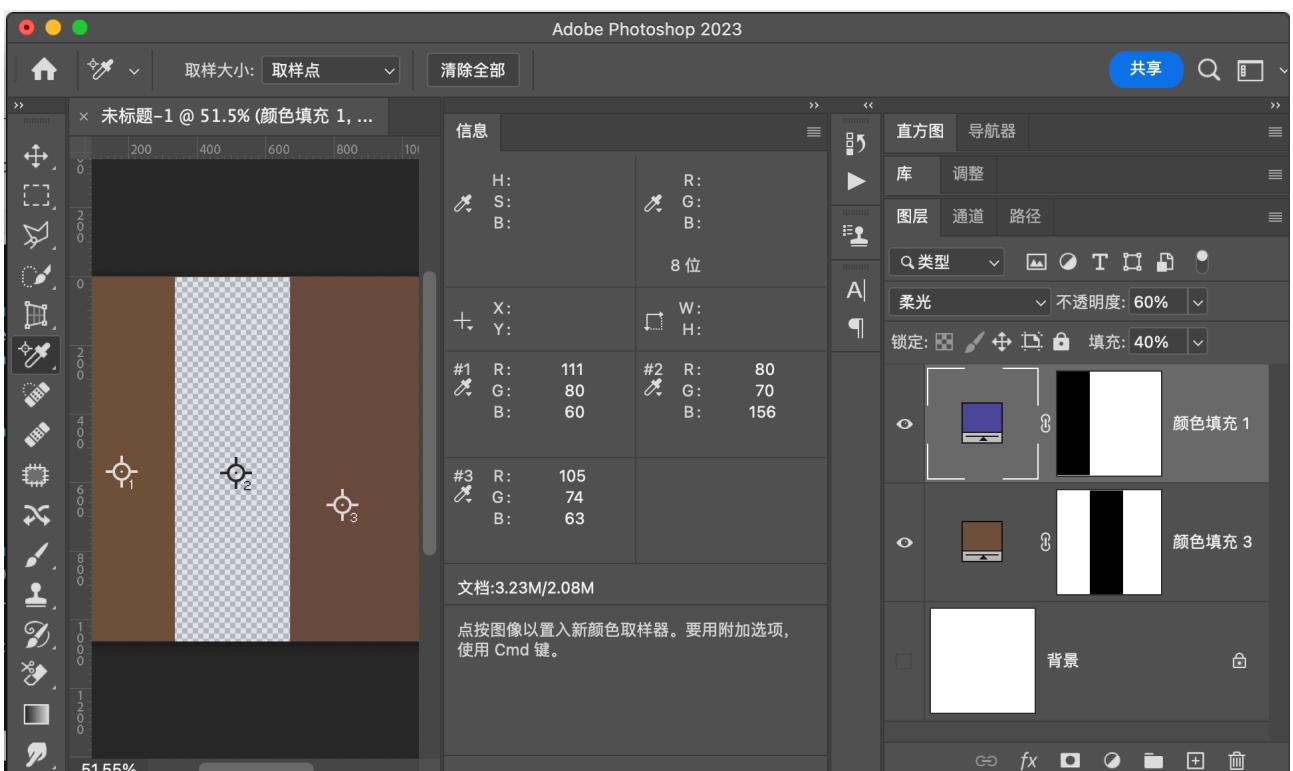


图 9.6

9.4 强光 HardLight

此模式也是正片叠底和滤色的组合，组合的分割界限是混合图层中性灰平面，并且它和叠加模式是互逆的关系，也就是说，如果在强光模式下把基础图层和混合图层顺序调换，可以得到原顺序下叠加模式的效果。

9.4.1 公式

$$r = HardLight(b, a) = \begin{cases} Multiply(b, 2a) & 0 \leq a \leq 0.5 \\ Screen(b, 2(a - 0.5)) & 0.5 < a \leq 1 \end{cases} \quad (9.7)$$

$$= \begin{cases} 2ba & 0 \leq a \leq 0.5 \\ 1 - 2(1 - b)(1 - a) & 0.5 < a \leq 1 \end{cases}$$

9.4.2 和填充结合

$$r = Fill(b, a) = \begin{cases} fill \times Multiply(b, 2a) + (1 - fill) \times b & 0 \leq a \leq 0.5 \\ fill \times Screen(b, 2(a - 0.5)) + (1 - fill) \times b & 0.5 < a \leq 1 \end{cases} \quad (9.8)$$

9.4.3 融合不透明度

$$r = Opacity(b, a) = op \times Fill(b, a) + (1 - op) \times b \quad (9.9)$$

9.4.4 映射面和同图等效曲线

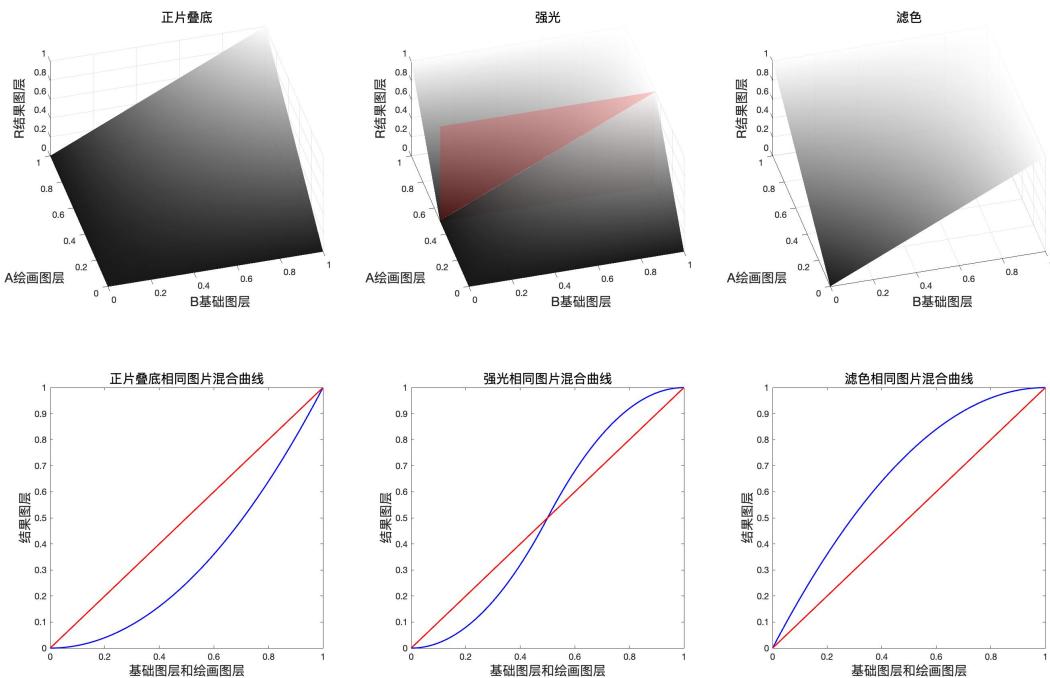


图 9.7

9.4.5 程序模拟该模式计算结果

```
// 强光
public static BlendColor HardLight(BlendColor colorBase, BlendColor colorBlend, double fill, double
    opacity) {
    double red = HardLightChannel(colorBase.red.get01Value(), colorBlend.red.get01Value(), fill);
    double green = HardLightChannel(colorBase.green.get01Value(), colorBlend.green.get01Value(), fill)
        ;
    double blue = HardLightChannel(colorBase.blue.get01Value(), colorBlend.blue.get01Value(), fill);
    return ColorUtils.Opacity(colorBase, new BlendColor(red * 255, green * 255, blue * 255), opacity);
}

private static double HardLightChannel(double baseValue, double blendValue, double fill) {
    if (blendValue <= 0.5) {
        return MulitplyChannel(baseValue, 2 * blendValue, fill);
    } else {
        return ScreenChannel(baseValue, 2 * (blendValue - 0.5), fill);
    }
}
```

程序模拟结果✓

强光 (HardLight) RGB[101.08, 71.34, 70.46] HSY[1.72, 30.61, 80.16] HSB[1.72, 30.29, 39.64]

9.4.6 验证

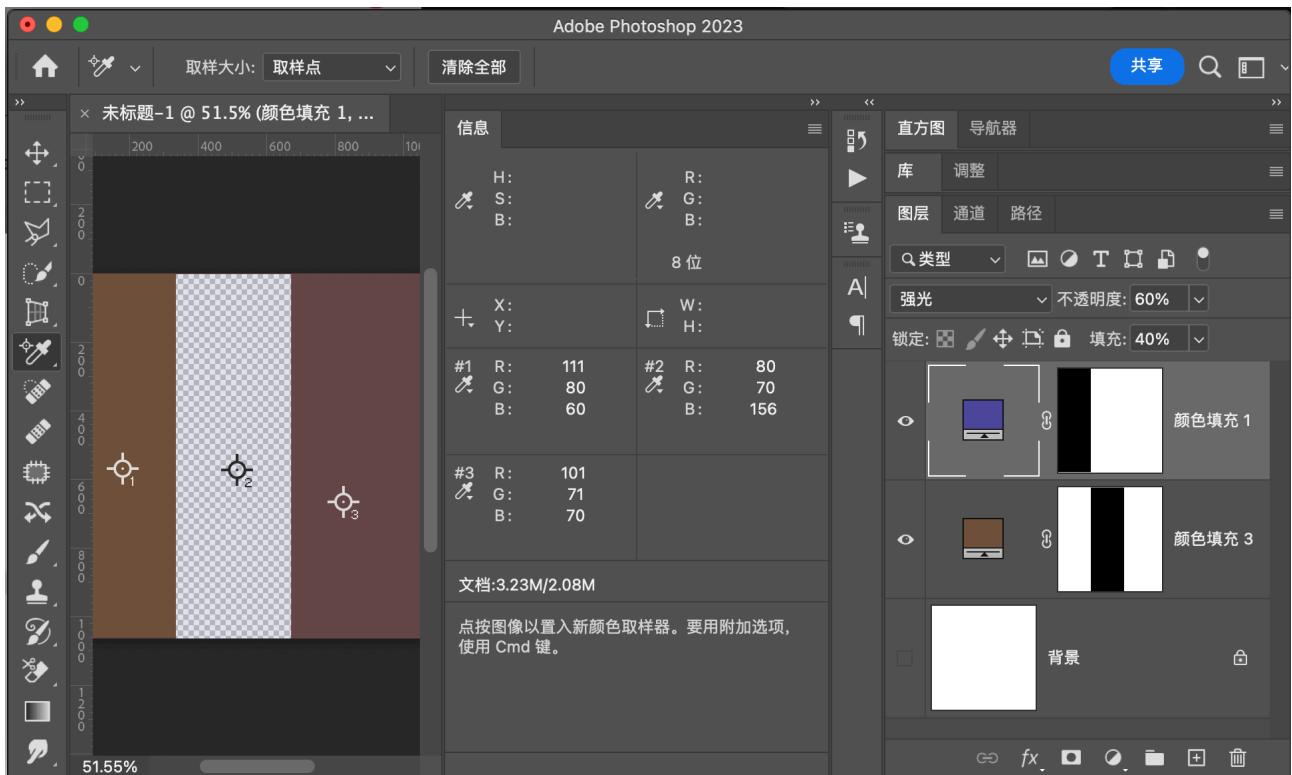


图 9.8

9.5 线性光 LinearLight

线性光是线性减淡和线性加深的结合，组合的分割界限是混合图层中性灰平面，具体公式如下

9.5.1 公式

$$\begin{aligned} r = \text{LinearLight}(b, a) &= \begin{cases} \text{LinearBurn}(b, 2a) & 0 \leq a \leq 0.5 \\ \text{LinearDodge}(b, 2(a - 0.5)) & 0.5 < a \leq 1 \end{cases} \\ &= \begin{cases} b + (2a) - 1 & 0 \leq a \leq 0.5 \\ b + 2(a - 0.5) & 0.5 < a \leq 1 \end{cases} = b + 2a - 1 \end{aligned} \quad (9.10)$$

9.5.2 和填充结合

$$\begin{aligned} r = \text{Fill}(b, a) &= \begin{cases} \text{LinearBurn}(b, 2a \times \text{fill}) & 0 \leq a \leq 0.5 \\ \text{LinearDodge}(b, 2(a - 0.5) \times \text{fill}) & 0.5 < a \leq 1 \end{cases} \\ &= \begin{cases} b + (2a) \times \text{fill} - 1 & 0 \leq a \leq 0.5 \\ b + 2(a - 0.5) \times \text{fill} & 0.5 < a \leq 1 \end{cases} = \begin{cases} b + 2a \times \text{fill} - 1 & 0 \leq a \leq 0.5 \\ b + 2a \times \text{fill} - 1 \times \text{fill} & 0.5 < a \leq 1 \end{cases} \end{aligned} \quad (9.11)$$

9.5.3 融合不透明度

$$r = \text{Opacity}(b, a) = \text{op} \times \text{Fill}(b, a) + (1 - \text{op}) \times b \quad (9.12)$$

9.5.4 映射面和同图等效曲线

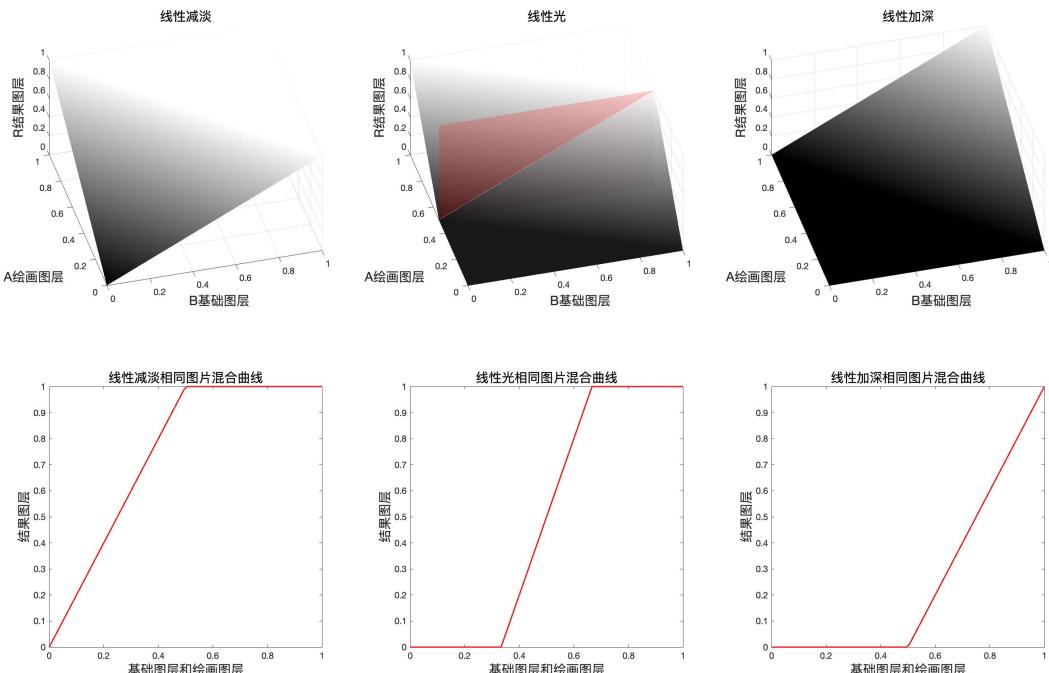


图 9.9

9.5.5 程序模拟该模式计算结果

```

public static BlendColor LinearLight(BlendColor colorBase, BlendColor colorBlend, double fill,
    double opacity) {
    double red = LinearLightChannel(colorBase.red.get01Value(), colorBlend.red.get01Value(), fill);
    double green = LinearLightChannel(colorBase.green.get01Value(), colorBlend.green.get01Value(),
        fill);
    double blue = LinearLightChannel(colorBase.blue.get01Value(), colorBlend.blue.get01Value(), fill);
    return ColorUtils.Opacity(colorBase, new BlendColor(red * 255, green * 255, blue * 255), opacity);
}

private static double LinearLightChannel(double baseValue, double blendValue, double fill) {
    if (blendValue <= 0.5) {
        return LinearBurnChannel(baseValue, 2 * blendValue, fill);
    } else {
        return LinearDodgeChannel(baseValue, 2 * (blendValue - 0.5), fill);
    }
}

```

程序模拟结果 ✓

线性光 (LinearLight) RGB[88.20, 52.40, 73.68] HSY[324.34, 35.80, 65.48] HSB[324.34, 40.59, 34.59]

9.5.6 验证

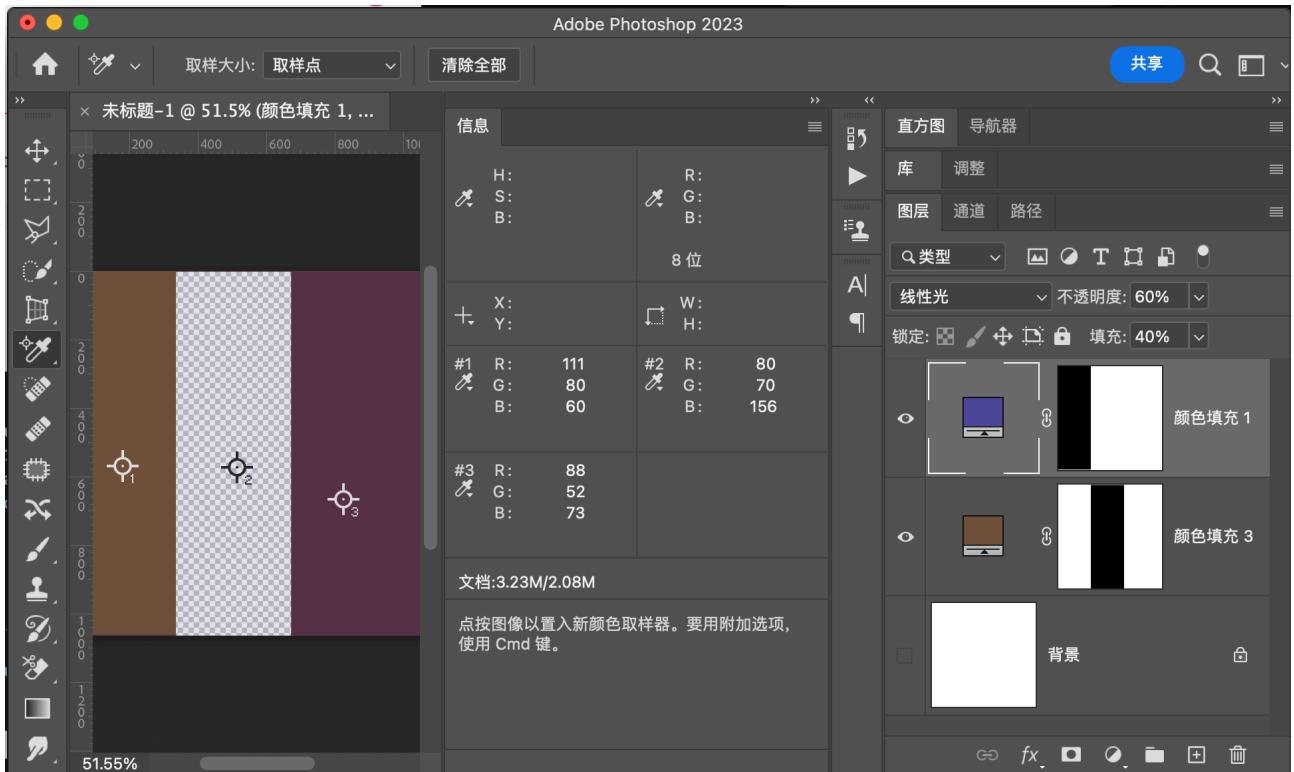


图 9.10

9.6 点光 PinLight

点光是变暗模式和变亮模式的组合

9.6.1 公式

$$r = PinLight(b, a) = \begin{cases} Darken(b, 2a) & a \leq 0.5 \\ Lighten(b, 2(a - 0.5)) & a > 0.5 \end{cases} = \begin{cases} Min(b, 2a) & a \leq 0.5 \\ Max(b, 2(a - 0.5)) & a > 0.5 \end{cases} \quad (9.13)$$

9.6.2 和填充结合

$$r = Fill(b, a) = \begin{cases} Min(b, 2a \times fill) & a \leq 0.5 \\ Max(b, 2(a - 0.5) \times fill) & a > 0.5 \end{cases} \quad (9.14)$$

9.6.3 融合不透明度

$$r = Opacity(b, a) = op \times Fill(b, a) + (1 - op) \times b \quad (9.15)$$

9.6.4 映射面和同图等效曲线

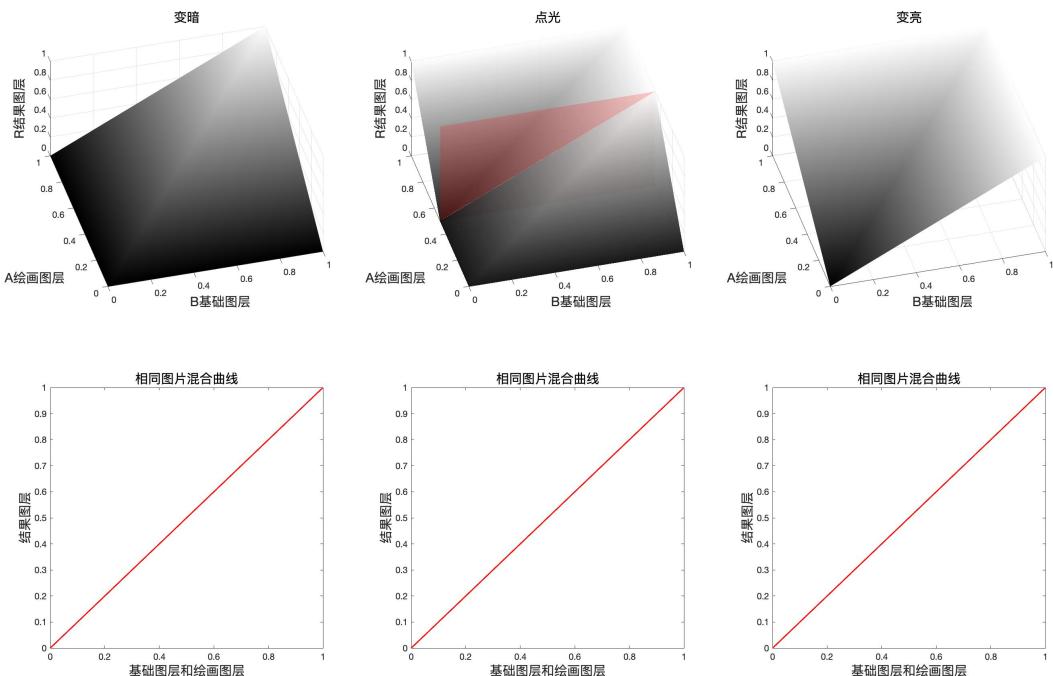


图 9.11

9.6.5 程序模拟该模式计算结果

```
// 点光
public static BlendColor PinLight(BlendColor colorBase, BlendColor colorBlend, double fill, double
    opacity) {
    double red = PinLightChannel(colorBase.red.get01Value(), colorBlend.red.get01Value(), fill);
    double green = PinLightChannel(colorBase.green.get01Value(), colorBlend.green.get01Value(), fill);
    double blue = PinLightChannel(colorBase.blue.get01Value(), colorBlend.blue.get01Value(), fill);
    return ColorUtils.Opacity(colorBase, new BlendColor(red * 255, green * 255, blue * 255), opacity);
}

private static double PinLightChannel(double baseValue, double blendValue, double fill) {
    if (blendValue <= 0.5) {
        return DarkenChannel(baseValue, 2 * blendValue, fill);
    } else {
        return LightenChannel(baseValue, 2 * (blendValue - 0.5), fill);
    }
}
```

程序模拟结果✓

点光 (PinLight) RGB[111.00, 80.00, 60.00] HSY[23.53, 51.00, 87.10] HSB[23.53, 45.95, 43.53]

9.6.6 验证

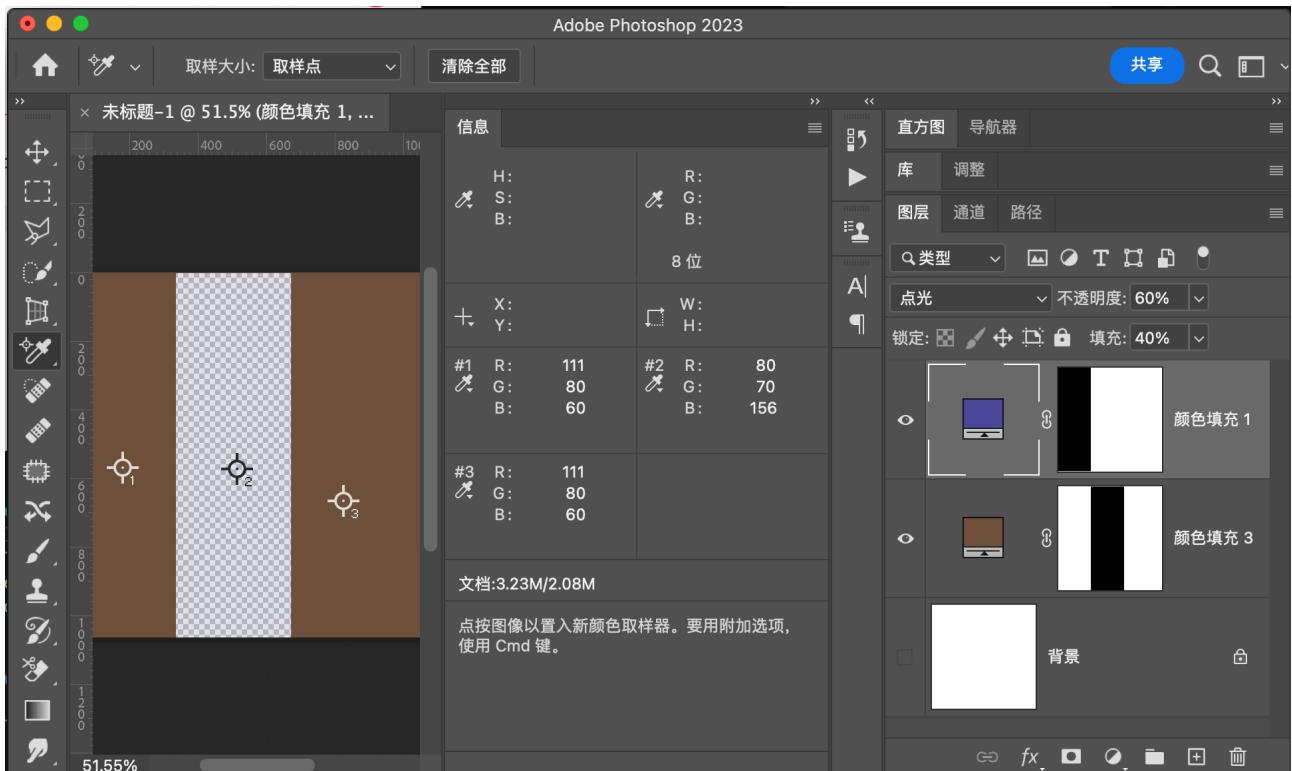


图 9.12

9.7 亮光 VividLight

亮光模式是颜色加深和颜色减淡的组合

9.7.1 公式

$$r = VividLight(b, a) = \begin{cases} ColorBurn(b, 2a) & a \leq 0.5 \\ ColorDodge(b, 2(a - 0.5)) & a > 0.5 \end{cases} = \begin{cases} 1 - \frac{(1-b)}{2a} & a \leq 0.5 \\ \frac{b}{1-2(a-0.5)} & a > 0.5 \end{cases} \quad (9.16)$$

9.7.2 加上 fill

$$r = FILL(b, a) = \begin{cases} ColorBurn(b, 2a \times fill) & a \leq 0.5 \\ ColorDodge(b, 2(a - 0.5) \times fill) & a > 0.5 \end{cases} = \begin{cases} 1 - \frac{(1-b)}{2a \times fill} & a \leq 0.5 \\ \frac{b}{1-2(a-0.5) \times fill} & a > 0.5 \end{cases} \quad (9.17)$$

9.7.3 融合不透明度

$$r = Opacity(b, a) = op \times Fill(b, a) + (1 - op) \times b \quad (9.18)$$

9.7.4 映射面和同图等效曲线

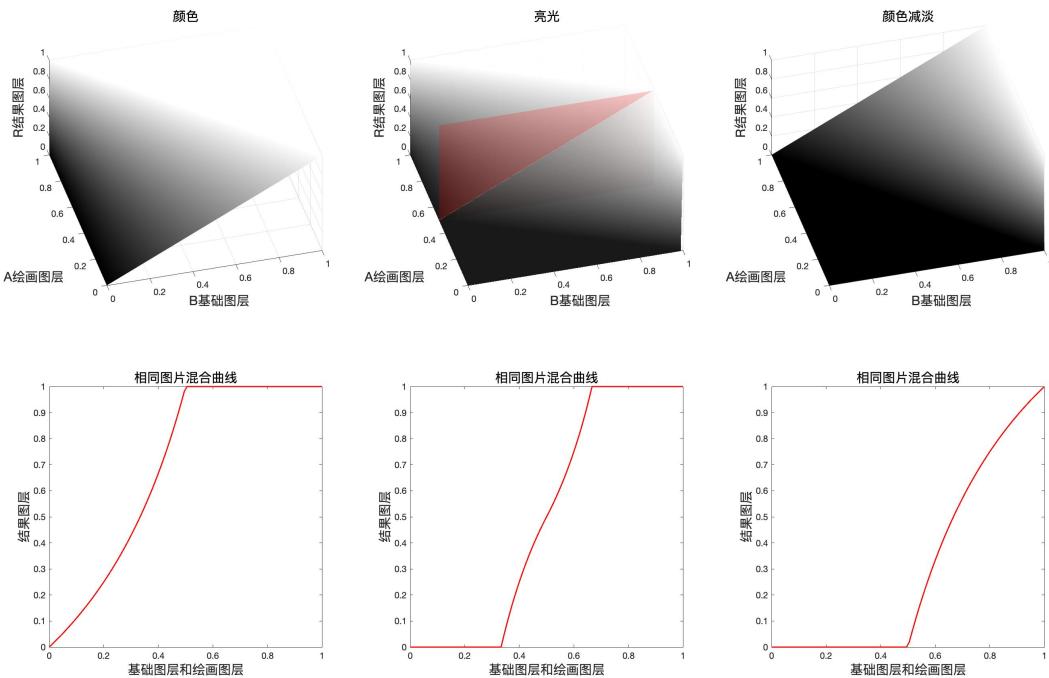


图 9.13

9.7.5 程序模拟该模式计算结果

```
// 亮光
public static BlendColor VividLight(BlendColor colorBase, BlendColor colorBlend, double fill, double
    opacity) {
    double red = VividLightChannel(colorBase.red.get01Value(), colorBlend.red.get01Value(), fill);
    double green = VividLightChannel(colorBase.green.get01Value(), colorBlend.green.get01Value(), fill
        );
    double blue = VividLightChannel(colorBase.blue.get01Value(), colorBlend.blue.get01Value(), fill);
    return ColorUtils.Opacity(colorBase, new BlendColor(red * 255, green * 255, blue * 255), opacity);
}

private static double VividLightChannel(double baseValue, double blendValue, double fill) {
    if (blendValue <= 0.5) {
        return ColorBurnChannel(baseValue, 2 * blendValue, fill);
    } else {
        return ColorDodgeChannel(baseValue, 2 * (blendValue - 0.5), fill);
    }
}
```

程序模拟结果 ✓

👉 亮光 (VividLight) RGB[95.87, 56.89, 63.53] HSY[349.77, 38.98, 69.31] HSB[349.77, 40.66, 37.60]

9.7.6 验证

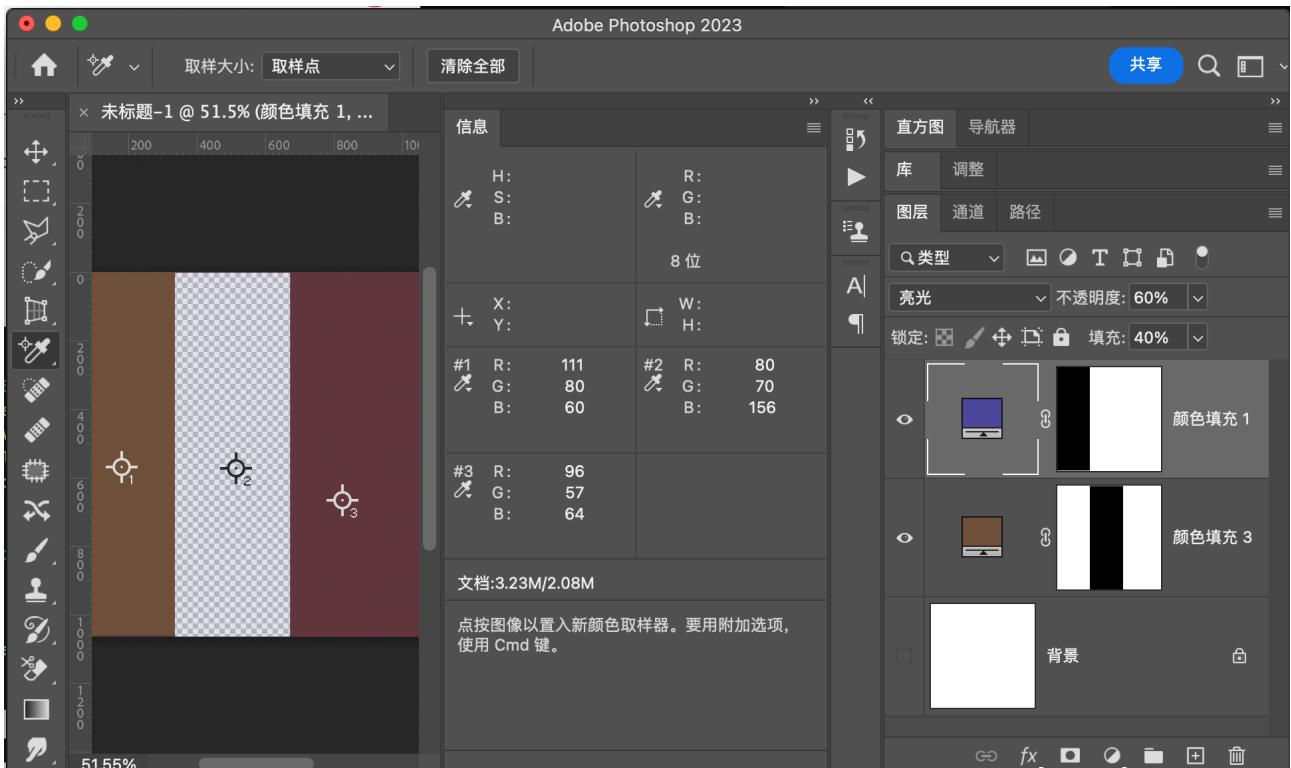


图 9.14

9.8 实色混合 HardMix

实色混合是一种极端的混合方式，但是如果它合填充结合，则它会产生一些意想不到的效果，并且我们还能找到它合线性光的关系

9.8.1 公式

$$r = \text{HardMix}(b, a) = \begin{cases} 1 & b + a \geq 1 \\ 0 & \text{else} \end{cases} \quad (9.19)$$

由公式我们可以看出，最后的结果只有两个，所以最后之后保留 $2^3 = 8$ 种颜色，也就是
(0,0,0) 黑,(1,0,0) 红,(1,1,0) 黄,(1,1,1) 白, (0,1,0) 绿,(0,1,1) 青,(1,0,1) 品红,(0,0,1) 蓝

9.8.2 加上 fill

但是如果填充介入表达式，则结果将合线性光类似

$$r = \text{HardMix}_{\text{fill}}(b, a) = \begin{cases} 0 & \frac{\text{fill} \times a + b - \text{fill}}{(1 - \text{fill})} < 0 \\ \frac{\text{fill} \times a + b - \text{fill}}{(1 - \text{fill})} & 0 \leq \frac{\text{fill} \times a + b - \text{fill}}{(1 - \text{fill})} \leq 1 \\ 1 & \frac{\text{fill} \times a + b - \text{fill}}{(1 - \text{fill})} > 1 \end{cases} \quad (9.20)$$

如果 fill 的取值是 0.5 则

$$\begin{aligned} r &= \text{HardMix}_{\text{fill}}(b, a) = \begin{cases} 0 & \frac{0.5 \times a + b - 0.5}{(1 - 0.5)} < 0 \\ \frac{0.5 \times a + b - 0.5}{(1 - 0.5)} & 0 \leq \frac{0.5 \times a + b - 0.5}{(1 - 0.5)} \leq 1 \\ 1 & \frac{0.5 \times a + b - 0.5}{(1 - 0.5)} > 1 \end{cases} \\ &= \begin{cases} 0 & a + 2b - 1 < 0 \\ a + 2b - 1 & 0 \leq a + 2b - 1 \leq 1 \\ 1 & a + 2b - 1 > 1 \end{cases} \end{aligned} \quad (9.21)$$

提示

上面的结果就是线性光的表达式，也就是说此时二者等价，或者说是互逆，也就是说，实色混合其实是线性光的强化版本，可以实现线性光的功能而去变化更多。

9.8.3 融合不透明度

$$r = \text{Opacity}(b, a) = op \times \text{HardMix}_{\text{fill}}(b, a) + (1 - op) \times b \quad (9.22)$$

9.8.4 映射面和同图等效曲线

由此我们可以看出，实色混合可以看作线性减淡和线性加深的组合，也可以看作是颜色减淡和颜色加深的组合，注意这里是可以看作，并不是真的。

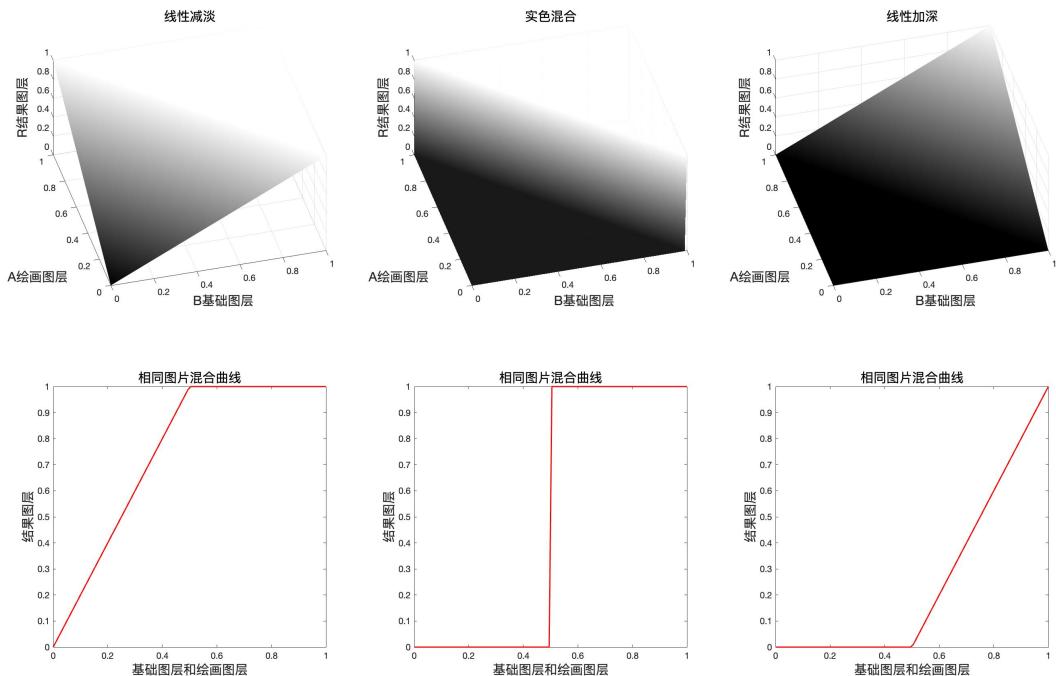


图 9.15

线性减淡和线性加深组合方式（真的）

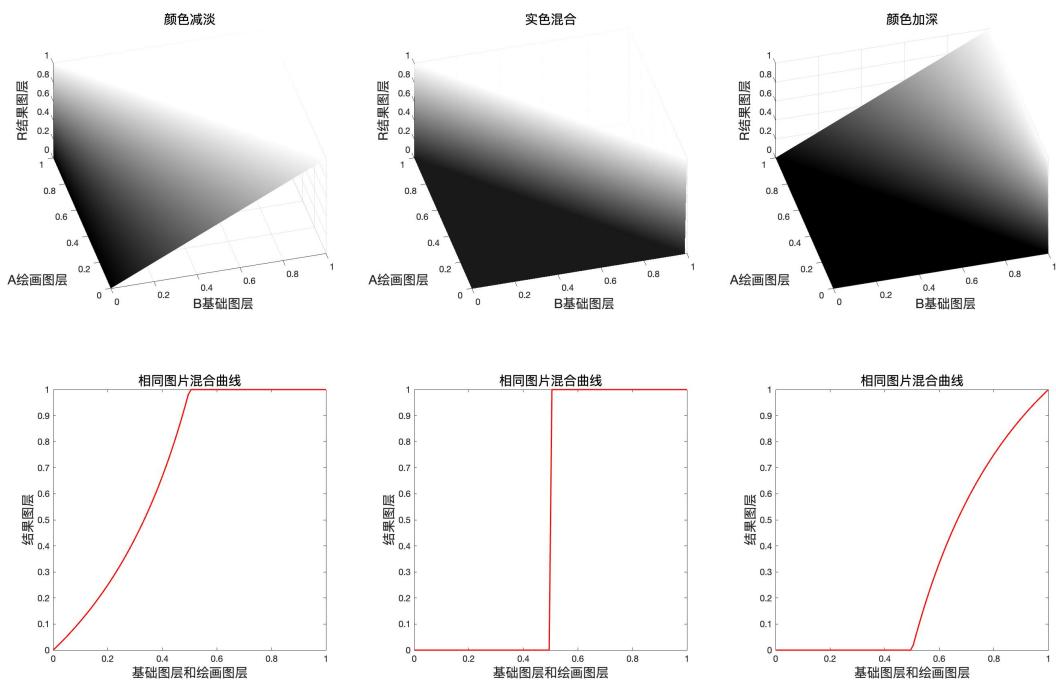


图 9.16

颜色减淡和颜色加深的组合方式（可以看作是这样）

提示

由我们刚才推导出当填充等于 50% 的时候，他可以和线性光互逆，此时我们也可以得出结论，实色混合本质是一种特殊的线性减淡和线性加深的组合，并且线性光是实色混合的特殊形式。

9.8.5 程序模拟该模式计算结果

```
// 实色混合
public static BlendColor HardMix(BlendColor colorBase, BlendColor colorBlend, double fill, double
    opacity) {
    double red = HardMixChannel(colorBase.red.get01Value(), colorBlend.red.get01Value(), fill);
    double green = HardMixChannel(colorBase.green.get01Value(), colorBlend.green.get01Value(), fill);
    double blue = HardMixChannel(colorBase.blue.get01Value(), colorBlend.blue.get01Value(), fill);
    return ColorUtils.Opacity(colorBase, new BlendColor(red * 255, green * 255, blue * 255), opacity);
}

private static double HardMixChannel(double baseValue, double blendValue, double fill) {
    if (fill == 1) {
        if (baseValue + blendValue >= 0.5) {
            return 1;
        }
        return 0;
    }
    return ColorUtils.round((fill * blendValue + baseValue * fill) / (1 - fill), 1, 0);
}
```

程序模拟结果

实色混合 (HardMix) RGB[85.40, 38.00, 44.40] HSY[351.90, 47.40, 52.92] HSB[351.90, 55.50, 33.49]

9.8.6 验证

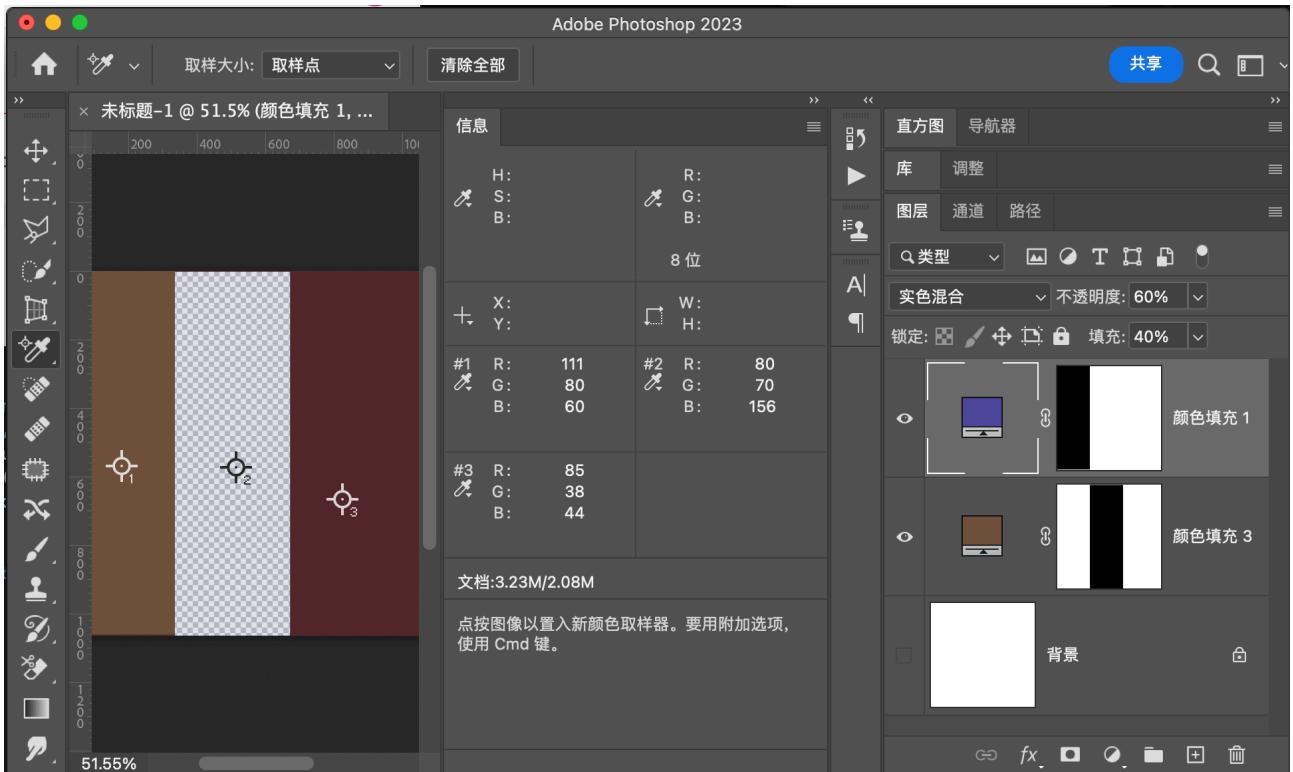


图 9.17

第 10 章 差值组

10.1 差值 Difference

差值就是基础图层和混合图层的差值的绝对值

10.1.1 公式

$$r = \text{Difference}(b, a) = |b - a| \quad (10.1)$$

10.1.2 结合填充

$$r = \text{Fill}(b, a) = |b - a| \times \text{fill} + (1 - \text{fill}) \times b \quad (10.2)$$

10.1.3 融合不透明度

$$r = \text{Opacity}(b, a) = \text{op} \times \text{Fill}(b, a) + (1 - \text{op}) \times b \quad (10.3)$$

10.1.4 映射面和同图等效曲线

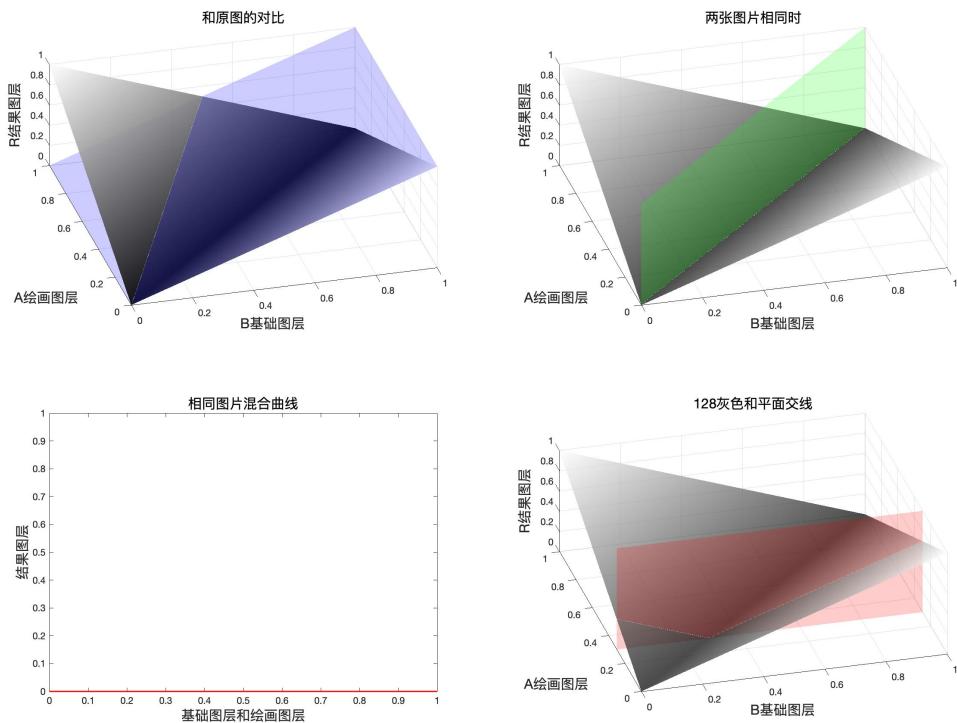


图 10.1

10.1.5 程序模拟该模式计算结果

```
// 差值
public static BlendColor Difference(BlendColor colorBase, BlendColor colorBlend, double fill, double
    opacity) {
    double red = DifferenceChannel(colorBase.red.get01Value(), colorBlend.red.get01Value(), fill);
    double green = DifferenceChannel(colorBase.green.get01Value(), colorBlend.green.get01Value(), fill
    );
    double blue = DifferenceChannel(colorBase.blue.get01Value(), colorBlend.blue.get01Value(), fill);
    return ColorUtils.Opacity(colorBase, new BlendColor(red * 255, green * 255, blue * 255), opacity);
}

private static double DifferenceChannel(double baseValue, double blendValue, double fill) {
    return ColorUtils.round(Math.abs(baseValue - blendValue) * fill), 1, 0;
}
```

程序模拟结果✓

差值 (Difference) RGB[91.80, 63.20, 25.44] HSY[34.14, 66.36, 67.63] HSB[34.14, 72.29, 36.00]

10.1.6 验证

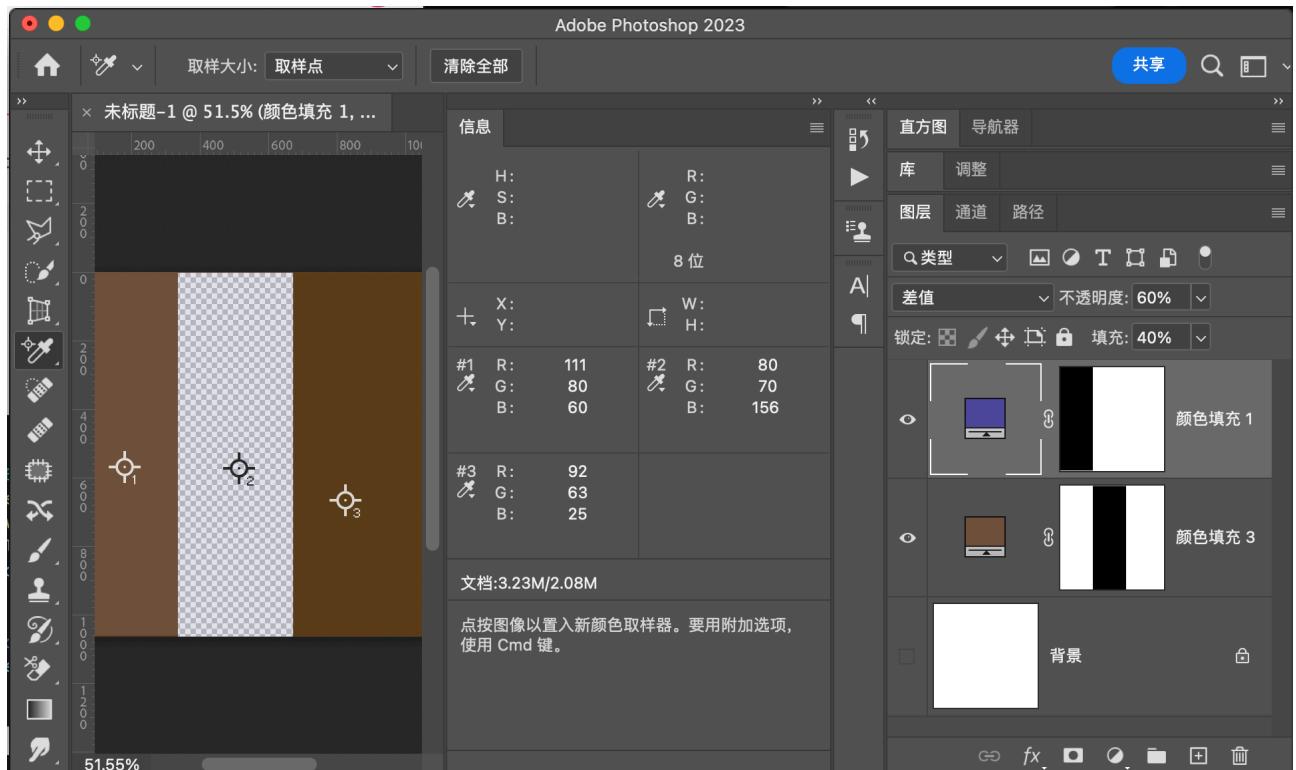


图 10.2

10.2 排除 Exclusion

10.2.1 公式

$$r = \text{Exclusion}(b, a) = b + a - 2ba \quad (10.4)$$

10.2.2 结合填充

$$r = \text{Fill}(b, a) = (b + a - 2ba) \times \text{fill} + (1 - \text{fill}) \times b \quad (10.5)$$

10.2.3 融合不透明度

$$r = \text{Opacity}(b, a) = \text{op} \times \text{Fill}(b, a) + (1 - \text{op}) \times b \quad (10.6)$$

10.2.4 如果在该模式下，混合图层是白色，黑色或者中性灰色

白色

$$r = \text{Exclusion}(b, 1) = b + 1 - 2b \times 1 = 1 - b \quad (10.7)$$

等于负片

黑色

$$r = \text{Exclusion}(b, 0) = b + 1 - 2b \times 0 = b \quad (10.8)$$

等于原图

中性灰

$$r = \text{Exclusion}(b, \frac{1}{2}) = b + \frac{1}{2} - 2b \times \frac{1}{2} = \frac{1}{2} \quad (10.9)$$

依然是中性灰

10.2.5 映射面和同图等效曲线

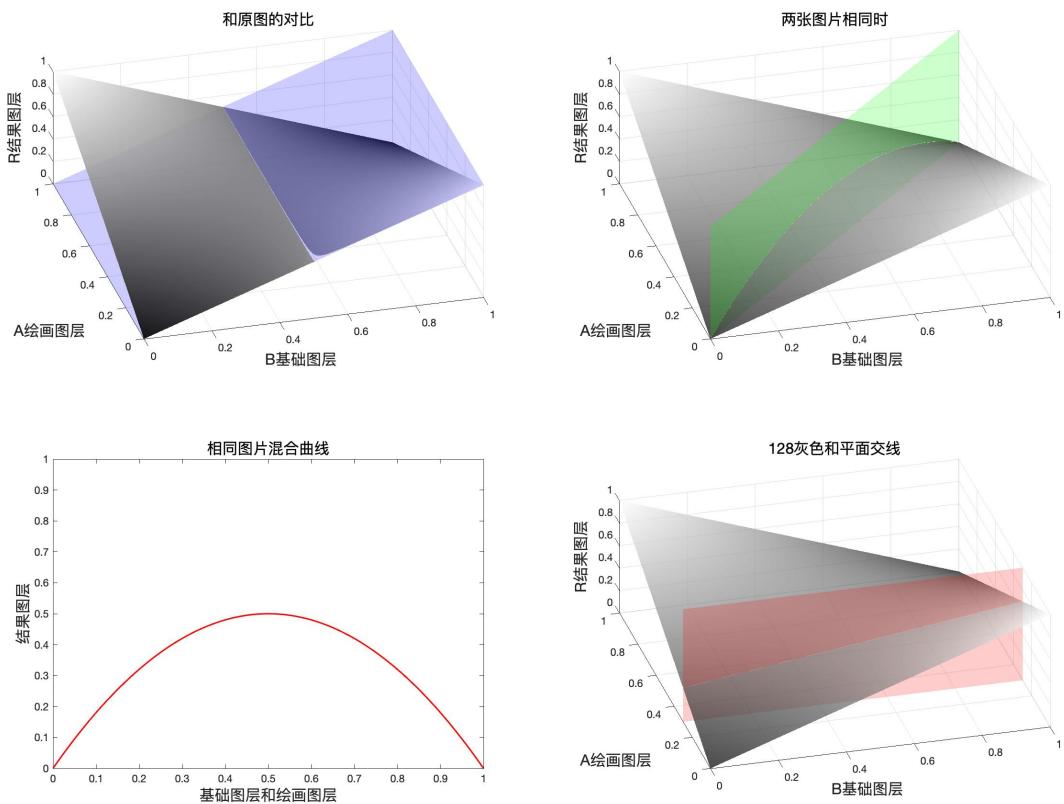


图 10.3

10.2.6 程序模拟该模式计算结果

```
// 排除
public static BlendColor Exclusion(BlendColor colorBase, BlendColor colorBlend, double fill, double
    opacity) {
    double red = ExclusionChannel(colorBase.red.get01Value(), colorBlend.red.get01Value(), fill);
    double green = ExclusionChannel(colorBase.green.get01Value(), colorBlend.green.get01Value(), fill)
    ;
    double blue = ExclusionChannel(colorBase.blue.get01Value(), colorBlend.blue.get01Value(), fill);
    return ColorUtils.Opacity(colorBase, new BlendColor(red * 255, green* 255, blue * 255), opacity);
}

private static double ExclusionChannel(double baseValue, double blendValue, double fill) {
    return ColorUtils.round((baseValue + blendValue - 2 * baseValue * blendValue) * fill + (1 - fill)
        * baseValue,1, 0);
}
```

程序模拟结果✓

排除 (Exclusion) RGB[113.48, 86.26, 79.82] HSY[11.47, 33.66, 93.72] HSB[11.47, 29.66, 44.50]

10.2.7 验证

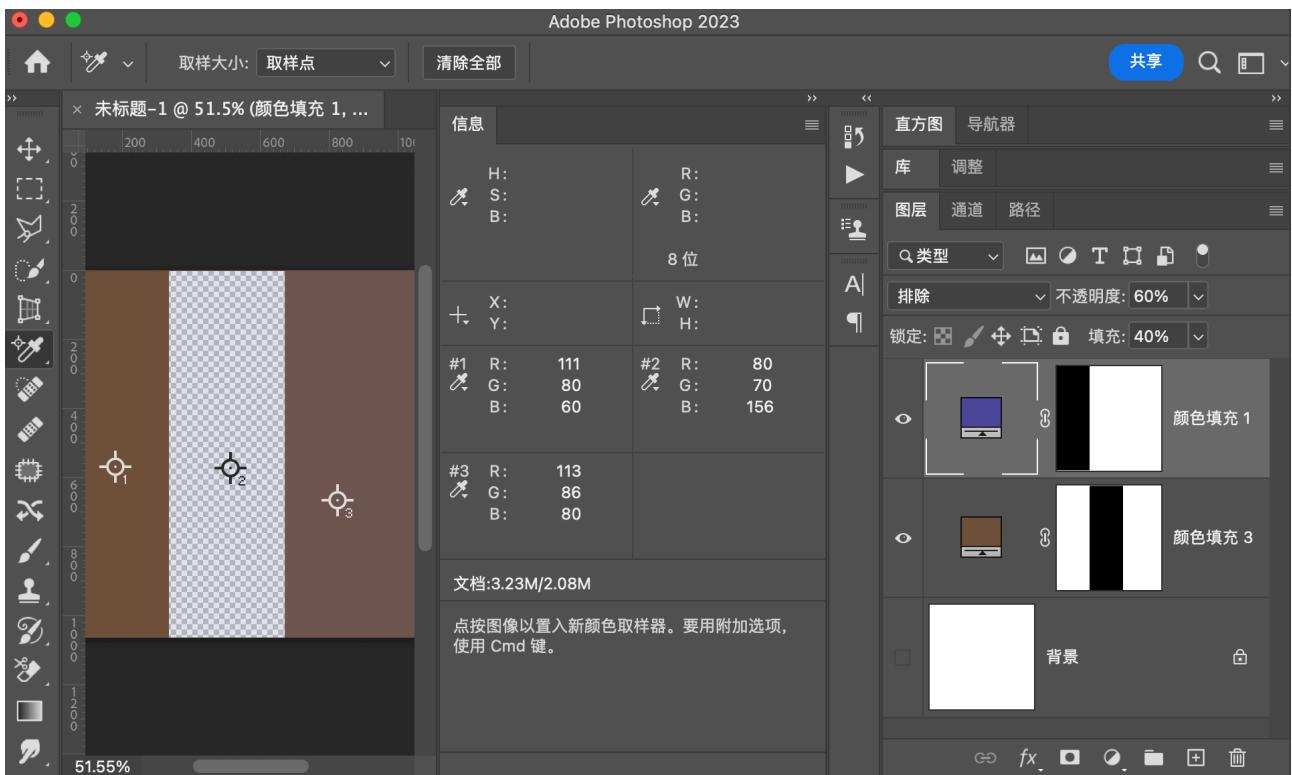


图 10.4

10.3 减去 Subtract

10.3.1 公式

$$r = \text{Subtract}(b, a) = b - a \quad (10.10)$$

10.3.2 结合填充

$$r = \text{Fill}(b, a) = \text{round}(ba) \times \text{fill} + (1 - \text{fill}) \times b \quad (10.11)$$

10.3.3 融合不透明度

$$r = \text{Opacity}(b, a) = \text{op} \times \text{Fill}(b, a) + (1 - \text{op}) \times b \quad (10.12)$$

10.3.4 映射面和同图等效曲线

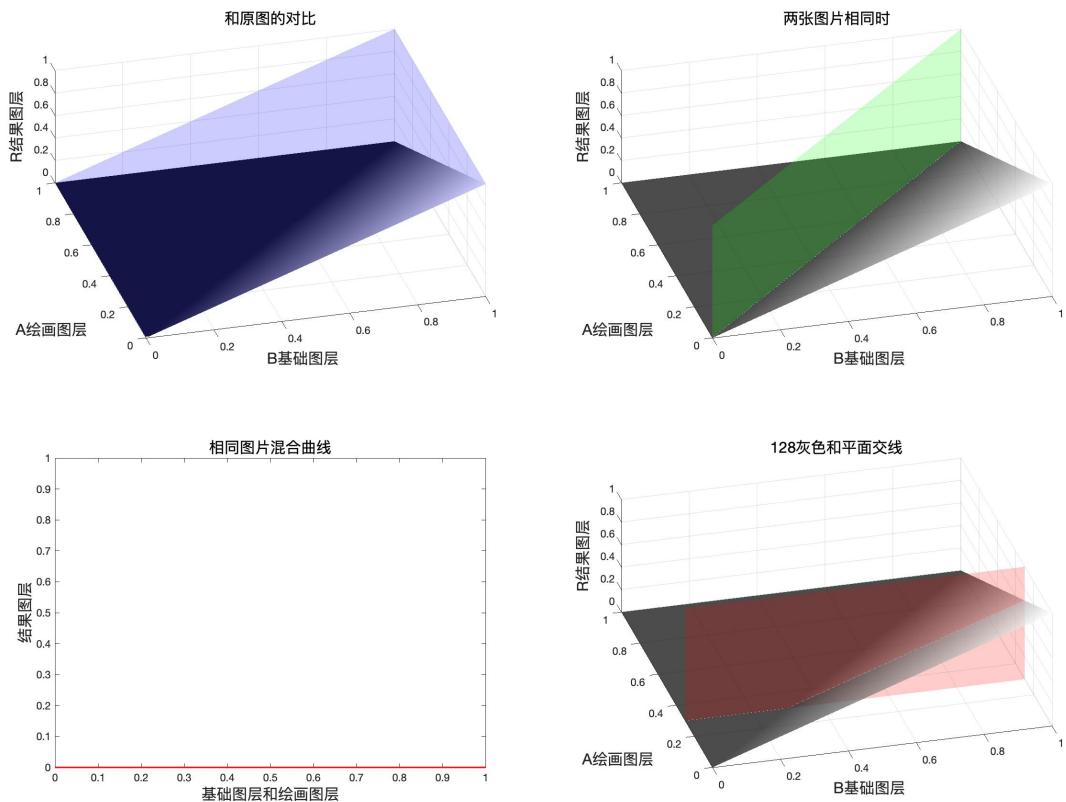


图 10.5

10.3.5 程序模拟该模式计算结果

```
// 减去
public static BlendColor Substact(BlendColor colorBase, BlendColor colorBlend, double fill, double
    opacity) {
    double red = SubstactChannel(colorBase.red.get01Value(), colorBlend.red.get01Value(), fill);
    double green = SubstactChannel(colorBase.green.get01Value(), colorBlend.green.get01Value(), fill);
    double blue = SubstactChannel(colorBase.blue.get01Value(), colorBlend.blue.get01Value(), fill);
    return ColorUtils.Opacity(colorBase, new BlendColor(red * 255, green * 255, blue * 255), opacity);
}

private static double SubstactChannel(double baseValue, double blendValue, double fill) {
    return ColorUtils.round(ColorUtils.round((baseValue - blendValue), 1, 0) * fill + (1 - fill) *
        baseValue, 1, 0);
}
```

程序模拟结果 ✓

减去 (Substact) RGB[91.80, 63.20, 45.60] HSY[22.86, 46.20, 69.84] HSB[22.86, 50.33, 36.00]

10.3.6 验证

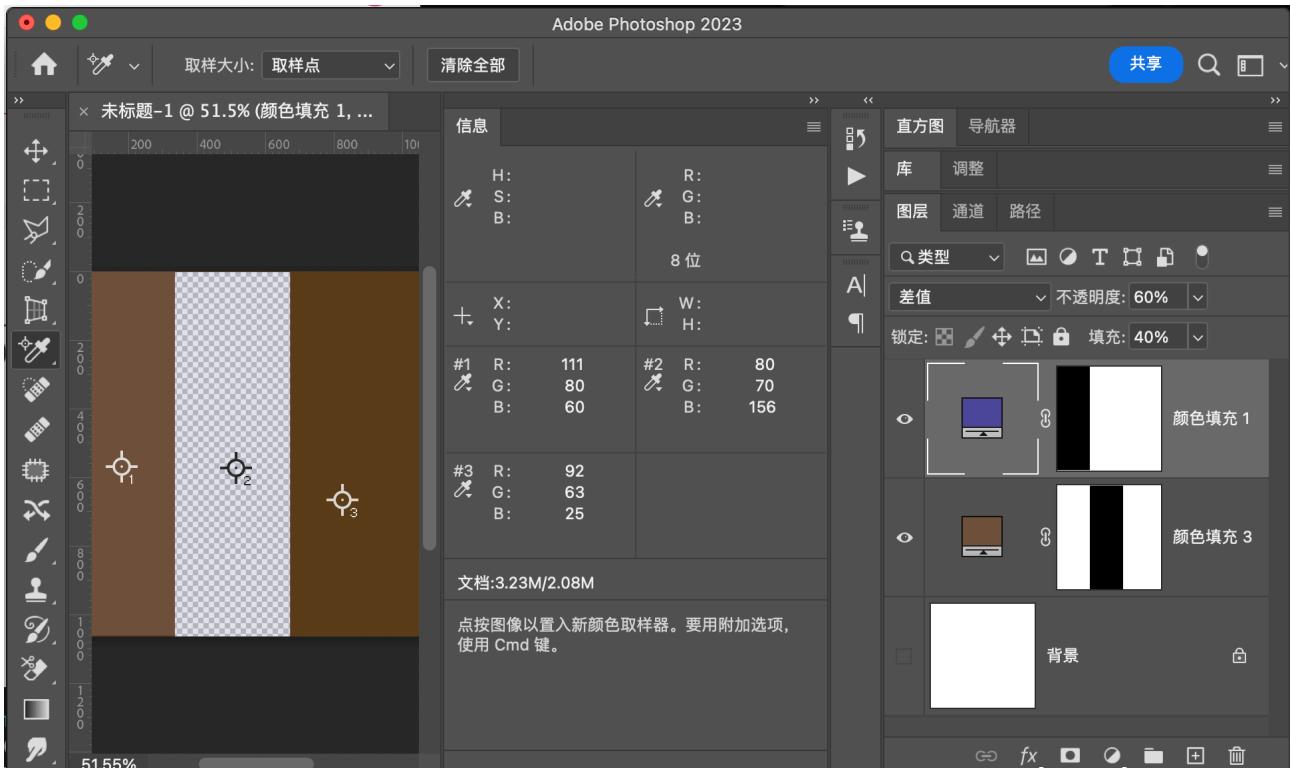


图 10.6

10.4 划分 Divide

10.4.1 公式

$$r = Divide(b, a) = \frac{b}{a} \quad (10.13)$$

10.4.2 结合填充

$$r = Fill(b, a) = \frac{b}{a} \times fill + (1 - fill) \times b \quad (10.14)$$

10.4.3 融合不透明度

$$r = Opacity(b, a) = op \times Fill(b, a) + (1 - op) \times b \quad (10.15)$$

划分和颜色减淡可以通过一次负片操作转换

10.4.4 一次负片转颜色减淡

$$r = Divide(b, 1 - a) = \frac{b}{1 - a} = ColorDodge(b, a) \quad (10.16)$$

10.4.5 映射面和同图等效曲线

10.4.6 程序模拟该模式计算结果

```
// 划分
public static BlendColor Divide(BlendColor colorBase, BlendColor colorBlend, double fill, double
    opacity) {
    double red = DivdeChannel(colorBase.red.get01Value(), colorBlend.red.get01Value(), fill);
    double green = DivdeChannel(colorBase.green.get01Value(), colorBlend.green.get01Value(), fill);
    double blue = DivdeChannel(colorBase.blue.get01Value(), colorBlend.blue.get01Value(), fill);
    return ColorUtils.Opacity(colorBase, new BlendColor(red * 255, green * 255, blue * 255), opacity);
}

private static double DivdeChannel(double baseValue, double blendValue, double fill) {
    return ColorUtils.round(Math.min(1, baseValue / blendValue) * fill + (1 - fill) * baseValue, 1, 0)
    ;
}
```

程序模拟结果✓

划分 (Divide) RGB[145.56, 122.00, 69.14] HSY[41.50, 76.42, 123.25] HSB[41.50, 52.50, 57.08]

10.4.7 验证

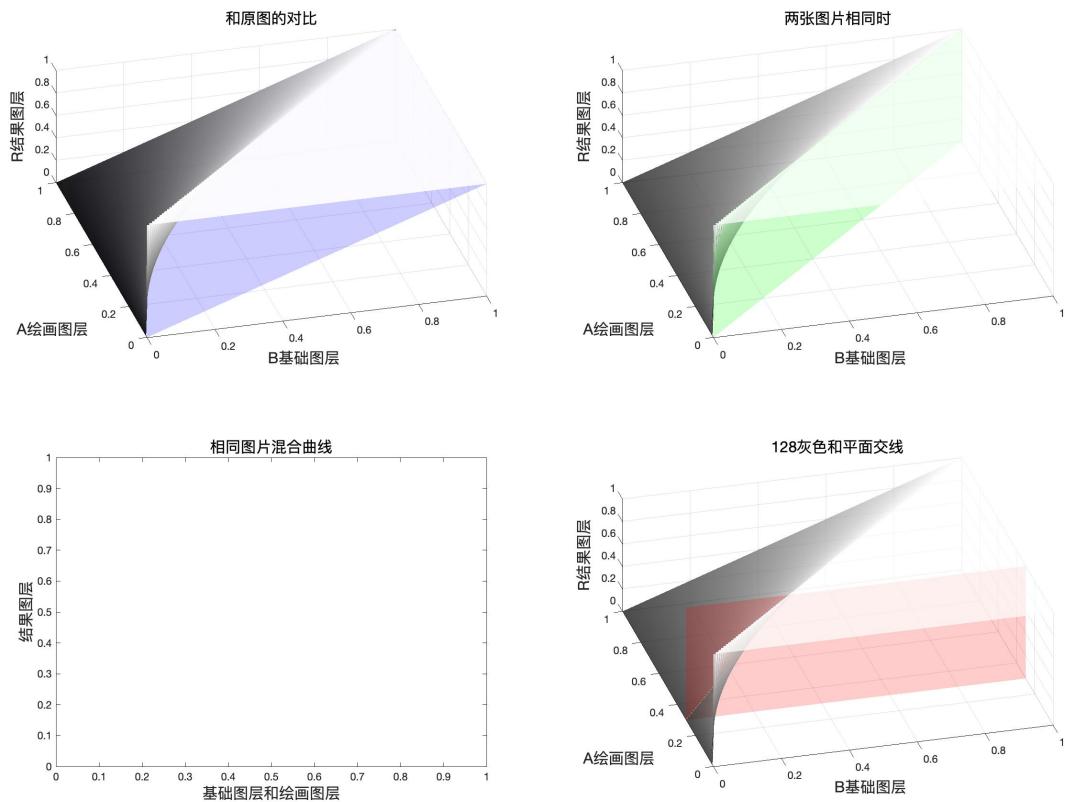


图 10.7

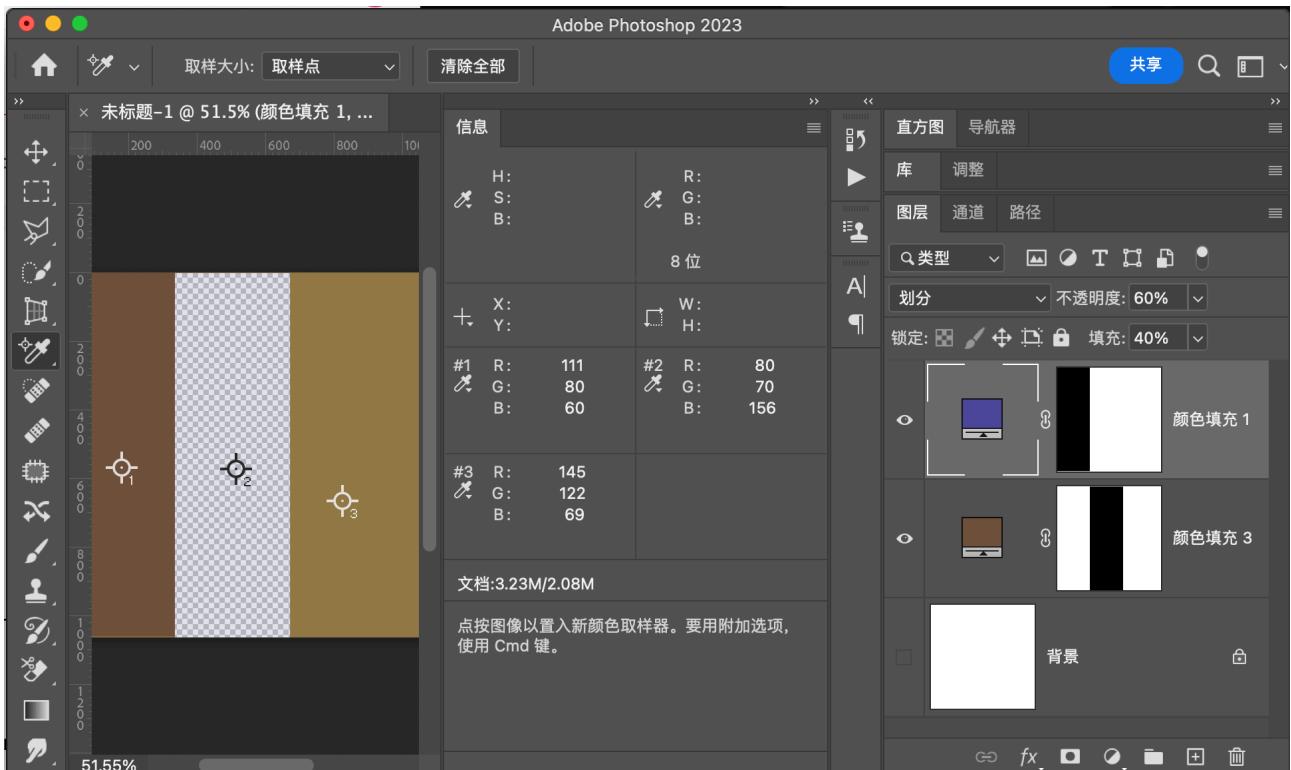


图 10.8

第 11 章 颜色组

内容提要

- 这一组和其他都不同，这一组是基于 HSY 颜色空间，并且设计的计算都是方程组。这里我们给出 HSY 颜色组的计算算法，为的是引出后面的修改明度和修改饱和度算法。

11.1 RGB→HSY 转换算法

Algorithm 1 计算色相

输入：颜色 C

输出：色相 hue

```
1: function Hue( $C$ )
2:    $min = Min(C_{red}, C_{green}, C_{blue})$ 
3:    $max = Max(C_{red}, C_{green}, C_{blue})$ 
4:    $hue = \begin{cases} 0 & max = min \\ 60^\circ \times \frac{C_{green} - C_{blue}}{max - min} & C_{red} = max \text{ 且 } C_{green} \geq C_{blue} \\ 60^\circ \times \frac{C_{green} - C_{blue}}{max - min} + 360^\circ & C_{red} = max \text{ 且 } C_{green} < C_{blue} \\ 60^\circ \times \frac{C_{blue} - C_{red}}{max - min} + 120^\circ & C_{green} = max \\ 60^\circ \times \frac{C_{red} - C_{green}}{max - min} + 240^\circ & C_{blue} = max \end{cases}$ 
5:   return  $hue$ 
6: end function
```

```
public double getHue() {
    ColorItem max = this.getMax();
    ColorItem mid = this.getMid();
    ColorItem min = this.getMin();
    if (Double.doubleToLongBits(max.value) == Double.doubleToLongBits(min.value)) {
        return 0;
    }
    double hueAbs = (mid.value - min.value) / (max.value - min.value);
    if (max.name == "red") {
        if (this.green.value > this.blue.value) {
            return 60 * hueAbs;
        } else {
            return -60 * hueAbs + 360;
        }
    }
    if (max.name == "green") {
        return 60 * hueAbs + 120;
    }
    if (max.name == "blue") {
        return 60 * hueAbs + 240;
    }
    return 0;
}
```

Algorithm 2 计算明度

输入: 颜色 C
输出: 明度 lum

- 1: **function** $Lum(C)$
- 2: $lum = 0.3 \times C_{red} + 0.59 \times C_{green} + 0.11 \times C_{blue}$
- 3: **return** lum
- 4: **end function**

```
public double getLum() {  
    double lum = 0.3 * this.red.value + 0.59 * this.green.value + 0.11 * this.blue.value;  
    return lum;  
}
```

Algorithm 3 计算饱和度

输入: 颜色 C
输出: 饱和度 sat

- 1: **function** $Sat(C)$
- 2: $min = Min(C_{red}, C_{green}, C_{blue})$
- 3: $max = Max(C_{red}, C_{green}, C_{blue})$
- 4: $sat = max - min$
- 5: **return** sat
- 6: **end function**

```
public double getSat() {  
    double max = this.getMax().value;  
    double min = this.getMin().value;  
    return max - min;  
}
```

11.2 HSY 替换算法

Algorithm 4 改变明度

输入：颜色 C ，需要改成的明度 lum

输出：新的颜色值 C

```

1: function SetLum( $C, lum$ )
2:    $d = lum - Lum(C)$ 
3:    $C_{red} = C_{red} + d$ 
4:    $C_{green} = C_{green} + d$ 
5:    $C_{blue} = C_{blue} + d$ 
6:   return ClipColor( $C$ )
7: end function

```

```

public static BlendColor setLum(BlendColor color, double lum) {
    double lum2 = color.getLum();
    double temp = lum - lum2;
    double red = color.red.value + temp;
    double green = color.green.value + temp;
    double blue = color.blue.value + temp;
    BlendColor retColor = new BlendColor(red, green, blue);
    return ClipColor(retColor);
}

```

Algorithm 5 改变饱和度

输入：颜色 C ，需要改成的饱和度值 sat

输出：新的颜色值 C

```

1: function SetSat( $C, sat$ )
2:   if  $C_{max} > C_{min}$  then
3:      $C_{mid} = \frac{(C_{mid} - C_{min}) \times sat}{C_{max} - C_{min}}$ 
4:      $C_{max} = sat$ 
5:   else
6:      $C_{mid} = C_{max} = 0$ 
7:   end if
8:    $C_{min} = 0$ 
9:   return  $C$ 
10: end function

```

```

public static BlendColor setSat(BlendColor color, double sat) {
    if (color.getMax().value > color.getMin().value) {
        color.getMid().value = (color.getMid().value - color.getMin().value) * sat
        / (color.getMax().value - color.getMin().value);
        color.getMax().value = sat;
    } else {
        color.getMid().value = color.getMax().value = 0.0;
    }
    color.getMin().value = 0;
    return color;
}

```

提示

这里用了取巧的做法，只涉及饱和度和明度的改变，因为改变色相相当于直接同时改变饱和度和明度。所以不需要替换色相的算法。

Algorithm 6 矫正颜色输入：颜色 C 输出：矫正后的颜色 C

```

1: function ClipColor( $C$ )
2:    $lum = Lum(C)$ 
3:    $min = Min(C_{red}, C_{green}, C_{blue})$ 
4:    $max = Max(C_{red}, C_{green}, C_{blue})$ 
5:   if  $min < 0$  then
6:      $C_{red} = lum + \frac{(C_{red} - lum) \times lum}{lum - min}$ 
7:      $C_{green} = lum + \frac{(C_{green} - lum) \times lum}{lum - min}$ 
8:      $C_{blue} = lum + \frac{(C_{blue} - lum) \times lum}{lum - min}$ 
9:   end if
10:  if  $max > 1$  then
11:     $C_{red} = lum + \frac{(C_{red} - lum) \times (1 - lum)}{max - lum}$ 
12:     $C_{green} = lum + \frac{(C_{green} - lum) \times (1 - lum)}{max - lum}$ 
13:     $C_{blue} = lum + \frac{(C_{blue} - lum) \times (1 - lum)}{max - lum}$ 
14:  end if
15:  return  $C$ 
16: end function

```

```

public static BlendColor ClipColor(BlendColor color) {
    double l = color.getLum();
    double min = color.getMin().value;
    if (min < 0.0) {
        double red = l + ((color.red.value - 1) * l / (l - min));
        double green = l + ((color.green.value - 1) * l / (l - min));
        double blue = l + ((color.blue.value - 1) * l / (l - min));
        BlendColor retColor = new BlendColor(red, green, blue);
        return retColor;
    }
    double max = color.getMax().value;
    if (max > 255.0) {
        double red = l + ((color.red.value - 1) * (1 - l) / (max - 1));
        double green = l + ((color.green.value - 1) * (1 - l) / (max - 1));
        double blue = l + ((color.blue.value - 1) * (1 - l) / (max - 1));
        BlendColor retColor = new BlendColor(red, green, blue);
        return retColor;
    }
    return color;
}

```

11.3 色相 Hue

计算方法是基于这个公式

$$(H_r, S_r, Y_r) = \text{Hue}((H_a, S_a, Y_a), (H_b, S_b, Y_b)) = (H_a, S_b, Y_b) \quad (11.1)$$

算出 HSY 的值之后再把 HSY 转化为 RGB 的数值。

此时我们想修改基础图层的色相，但是我们只有修改饱和度和明度的公式，于是我们直接对混合图层使用设置饱和度和明度于是我们得到

$$\text{Hue}(C_b, C_a) = \text{SetLum}(\text{SetSat}(C_a, \text{Sat}(C_b)), \text{Lum}(C_b)) \quad (11.2)$$

11.3.1 结合填充

$$r = \text{Fill}(C_b, C_a) = \text{Hue}(C_b, C_a) \times \text{fill} + (1 - \text{fill}) \times C_b \quad (11.3)$$

11.3.2 融合不透明度

$$r = \text{Opacity}(C_b, C_a) = \text{op} \times \text{Fill}(C_b, C_a) + (1 - \text{op}) \times C_b \quad (11.4)$$

11.3.3 程序模拟该模式计算结果

```
// 色相模式
public static BlendColor HUE(BlendColor colorBase, BlendColor colorBlend, double fill, double
    opacity) {
    BlendColor temp = HUE_Sub(colorBase, colorBlend);
    double red = temp.red.get01Value() * fill + colorBase.red.get01Value() * (1 - fill);
    double green = temp.green.get01Value() * fill + colorBase.green.get01Value() * (1 - fill);
    double blue = temp.blue.get01Value() * fill + colorBase.blue.get01Value() * (1 - fill);
    return ColorUtils.Opacity(colorBase, new BlendColor(red * 255, green * 255, blue * 255), opacity);
}

private static BlendColor HUE_Sub(BlendColor colorBase, BlendColor colorBlend) {
    return ColorUtils.setLum(ColorUtils.setSat(colorBlend, colorBase.getSat()), colorBase.getLum());
}
```

程序模拟结果✓

✍ 色相 (hue) RGB[104.91, 79.93, 76.97] HSY[6.36, 27.94, 87.10] HSB[6.36, 26.63, 41.14]

11.3.4 验证

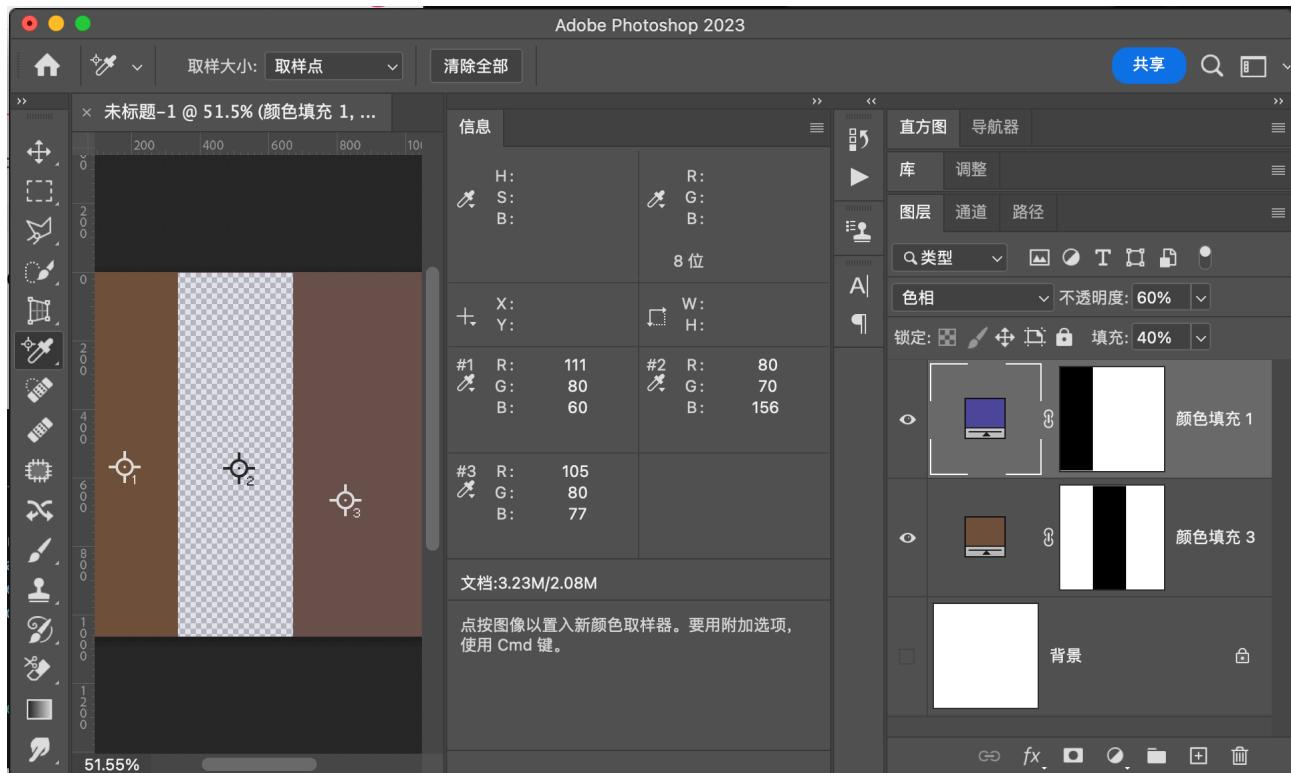


图 11.1

11.4 饱和度 Saturation

11.4.1 公式

计算方法是基于这个公式

$$(H_r, S_r, Y_r) = \text{Saturation}((H_a, S_a, Y_a), (H_b, S_b, Y_b)) = (H_b, S_a, Y_b) \quad (11.5)$$

设置饱和度，就直接对基础图层使用设置饱和度

$$\text{Saturation}(C_b, C_a) = \text{SetLum}(\text{SetSat}(C_b, \text{Sat}(C_s)), \text{Lum}(C_b)) \quad (11.6)$$

11.4.2 结合填充

$$r = \text{Fill}(C_b, C_a) = \text{Saturation}(C_b, C_a) \times \text{fill} + (1 - \text{fill}) \times C_b \quad (11.7)$$

11.4.3 融合不透明度

$$r = \text{Opacity}(C_b, C_a) = op \times \text{Fill}(C_b, C_a) + (1 - op) \times C_b \quad (11.8)$$

11.4.4 程序模拟该模式计算结果

```
// 饱和度模式
public static BlendColor Saturation(BlendColor colorBase, BlendColor colorBlend, double fill, double
    opacity) {
    double redbase = colorBase.red.get01Value();
```

```
double greenbase = colorBase.green.get01Value();
double bluebase = colorBase.blue.get01Value();
BlendColor temp = Saturation_Sub(colorBase, colorBlend);
double red = temp.red.get01Value() * fill + redbase * (1 - fill);
double green = temp.green.get01Value() * fill + greenbase * (1 - fill);
double blue = temp.blue.get01Value() * fill + bluebase * (1 - fill);
return ColorUtils.Opacity(new BlendColor(redbase * 255, greenbase * 255, bluebase * 255),
    new BlendColor(red * 255, green * 255, blue * 255), opacity);
}

private static BlendColor Saturation_Sub(BlendColor colorBase, BlendColor colorBlend) {
    double sat = colorBlend.getSat();
    double lum = colorBase.getLum();
    return ColorUtils.setLum(ColorUtils.setSat(colorBase, sat), lum);
}
```

程序模拟结果✓

饱和度 (Saturation) RGB[114.94, 78.83, 55.54] HSY[23.53, 59.40, 87.10] HSB[23.53, 51.68, 45.07]

11.4.5 验证

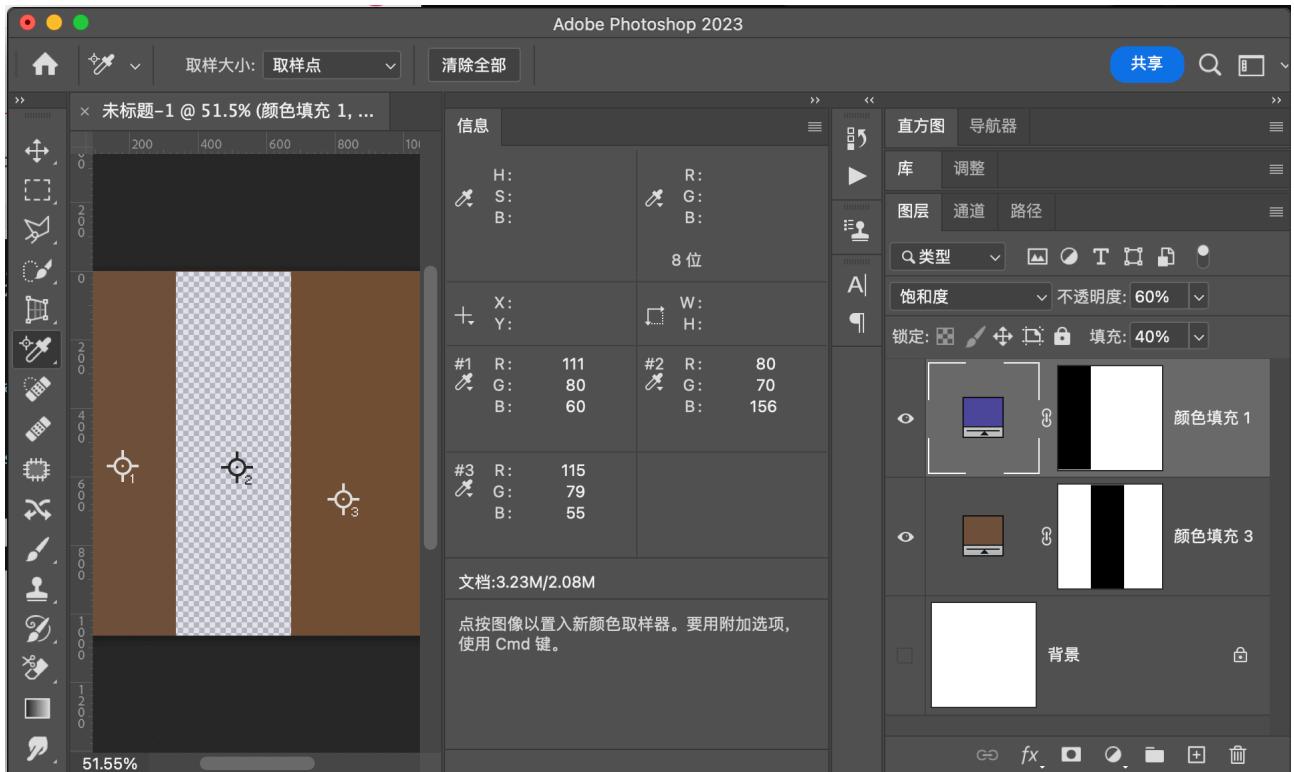


图 11.2

11.5 颜色 Color

计算方法是基于这个公式

$$(H_r, S_r, Y_r) = \text{Color}((H_a, S_a, Y_a), (H_b, S_b, Y_b)) = (H_a, S_a, Y_b) \quad (11.9)$$

直接对混合图层使用设置明度，则可以得到需要的等效结果

$$\text{Color}(C_b, C_a) = \text{SetLum}(C_a, \text{Lum}(C_b)) \quad (11.10)$$

11.5.1 结合填充

$$r = \text{Fill}(C_b, C_a) = \text{Saturation}(C_b, C_a) \text{fill} + (1 - \text{fill}) \times C_b \quad (11.11)$$

11.5.2 融合不透明度

$$r = \text{Opacity}(C_b, C_a) = op \times \text{Fill}(C_b, C_a) + (1 - op) \times C_b \quad (11.12)$$

11.5.3 程序模拟该模式计算结果

```
// 颜色模式
public static BlendColor BlendColor(BlendColor colorBase, BlendColor colorBlend, double fill, double
    opacity) {
    BlendColor temp = Color_Sub(colorBase, colorBlend);
    double red = temp.red.get01Value() * fill + colorBase.red.get01Value() * (1 - fill);
    double green = temp.green.get01Value() * fill + colorBase.green.get01Value() * (1 - fill);
    double blue = temp.blue.get01Value() * fill + colorBase.blue.get01Value() * (1 - fill);
    return ColorUtils.Opacity(colorBase, new BlendColor(red * 255, green * 255, blue * 255), opacity);
}

private static BlendColor Color_Sub(BlendColor colorBase, BlendColor colorBlend) {
    return ColorUtils.setLum(colorBlend, colorBase.getLum());
}
```

程序模拟结果✓

颜色 (BlendColor) RGB[104.67, 78.71, 84.15] HSY[347.43, 25.96, 87.10] HSB[347.43, 24.80, 41.05]

11.5.4 验证

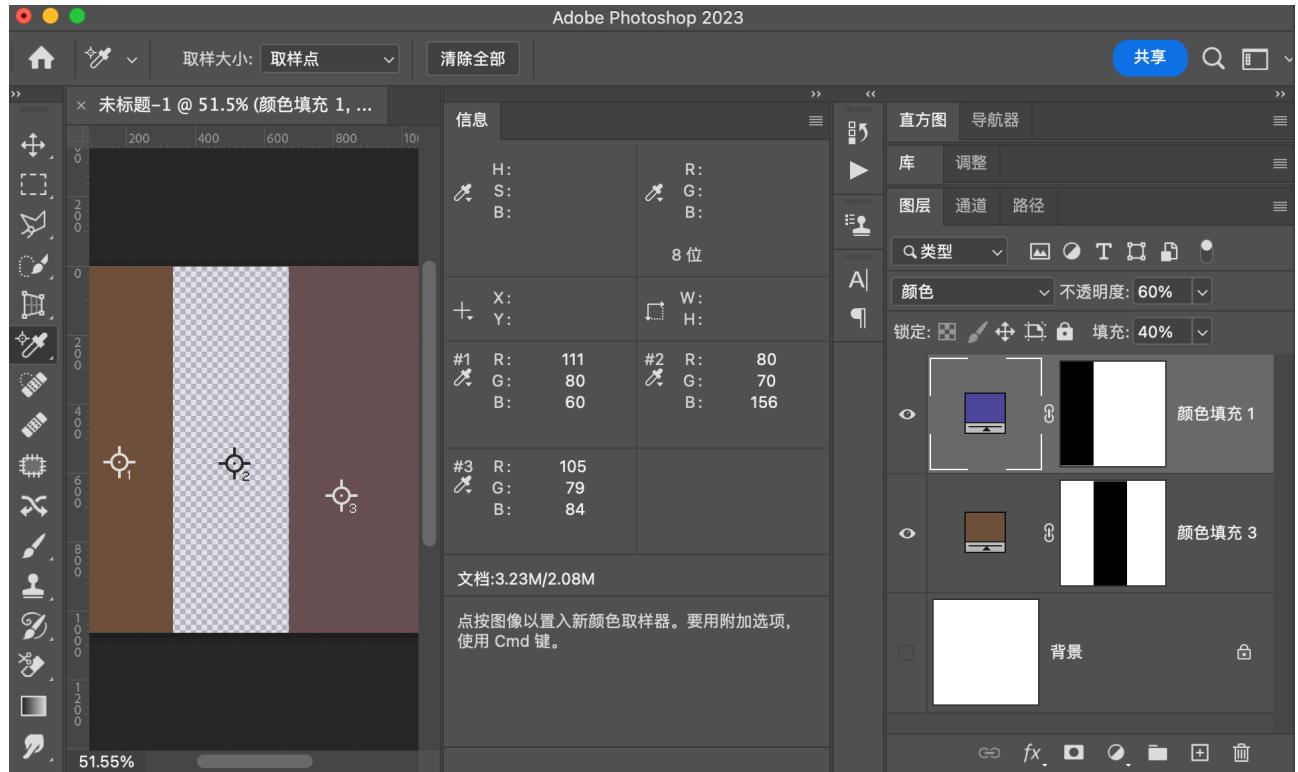


图 11.3

11.6 明度 Luminosity

计算方法是基于这个公式

$$(H_r, S_r, Y_r) = \text{Luminosity}((H_a, S_a, Y_a), (H_b, S_b, Y_b)) = (H_b, S_b, Y_a) \quad (11.13)$$

11.6.1 设置明度，就直接使用设置明度

$$\text{Luminosity}(C_b, C_a) = \text{SetLum}(C_b, \text{Lum}(C_a)) \quad (11.14)$$

11.6.2 结合填充

$$r = \text{Fill}(C_b, C_a) = \text{Luminosity}(C_b, C_a) \times \text{fill} + (1 - \text{fill}) \times C_b \quad (11.15)$$

11.6.3 融合不透明度

$$r = \text{Opacity}(C_b, C_a) = \text{op} \times \text{Fill}(C_b, C_a) + (1 - \text{op}) \times C_b \quad (11.16)$$

11.6.4 程序模拟该模式计算结果

```
// 明度模式
public static BlendColor Luminosity(BlendColor colorBase, BlendColor colorBlend, double fill, double
    opacity) {
    BlendColor temp = Luminosity_Sub(colorBase, colorBlend);
    double red = temp.red.get01Value() * fill + colorBase.red.get01Value() * (1 - fill);
    double green = temp.green.get01Value() * fill + colorBase.green.get01Value() * (1 - fill);
    double blue = temp.blue.get01Value() * fill + colorBase.blue.get01Value() * (1 - fill);
    return ColorUtils.Opacity(colorBase, new BlendColor(red * 255, green * 255, blue * 255), opacity);
}

private static BlendColor Luminosity_Sub(BlendColor colorBase, BlendColor colorBlend) {
    return ColorUtils.setLum(colorBase, colorBlend.getLum());
}
```

程序模拟结果 ✓

↳ 明度 (Luminosity) RGB[109.89, 78.89, 58.89] HSY[23.53, 51.00, 85.99] HSB[23.53, 46.41, 43.09]

11.6.5 验证

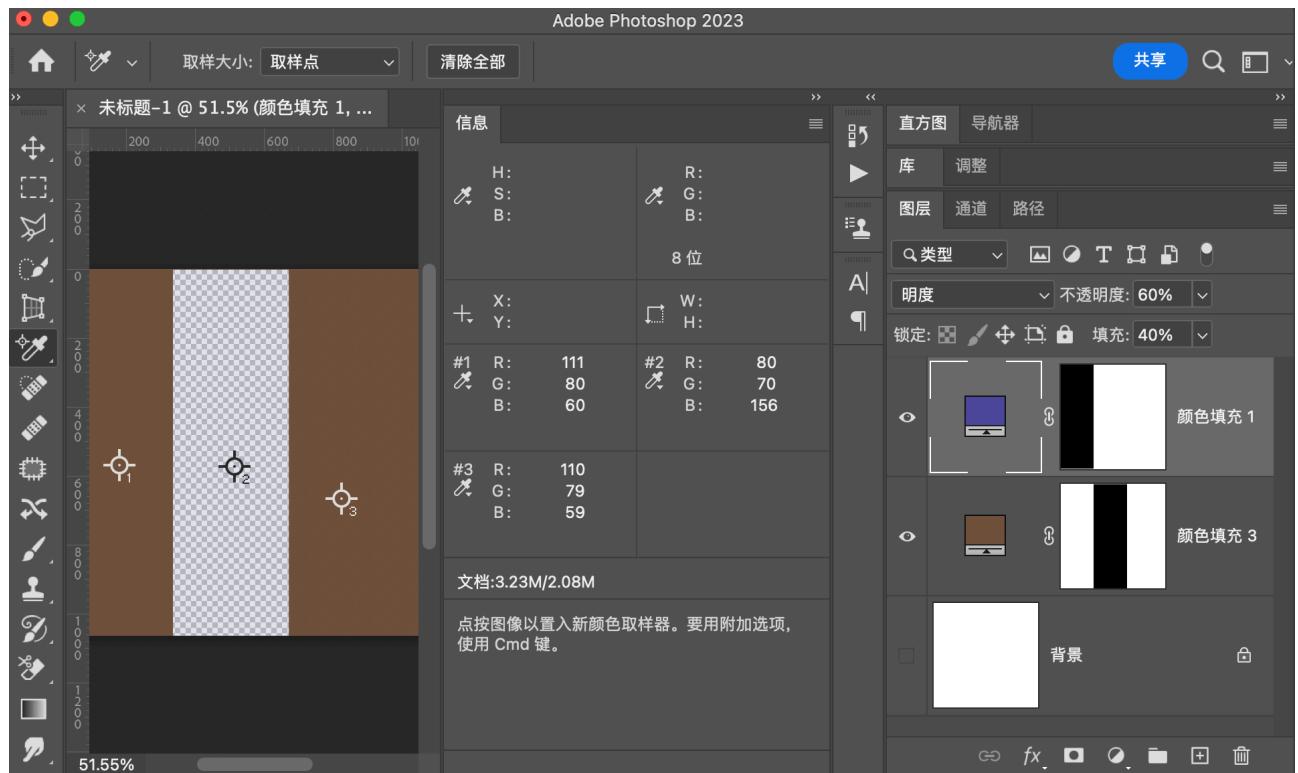


图 11.4

第 12 章 特殊的 5 种模式

12.1 穿透

穿透会出现在建立分组的时候，如果选择穿透，则此时效果和不建分组一样，但是如果修改为正常或者其他模式，则会先把这一组的图层计算出结果，然后用结果作为混合图层和下方图层进行运算。

12.2 相加

相加在计算和应用图像中，此时相当于强化的线性减淡，公式为

$$Add(b, a) = \frac{b + a}{zoom} + offset \quad (12.1)$$

缩放 $zoom$ 的取值范围是 [1, 2] 补偿值 $offset$ 的取值范围是 [0, 255]

12.3 相减

相减在计算和应用图像中，此时相当于强化的减去，公式为

$$Subtract(b, a) = \frac{b - a}{zoom} + offset \quad (12.2)$$

缩放 $zoom$ 的取值范围是 [1, 2] 补偿值 $offset$ 的取值范围是 [0, 255]

12.4 背后

背后模式简单来说就是，有像素点则笔刷或油漆桶工具不能修改，没有像素点的透明像素可以被修改。

12.5 擦除

功能相当于橡皮擦

第13章 调整图层和图层混合模式

如果调整图层和混合图层混用会发生什么我们假设调整图层为 $Adjustment(Layer)$, 则对于像素点 $Adjustment(pix)$, 对于通道 $Adjustment(channel)$ 则结果公式可以写作

$$r = BlendMode(b, Adjustment(b))$$

也就是说相当于, 先使用调整图层产生基础图层调整之后的图层, 再使用调整后的图层和原先的基础图层进行图层混合模式的操作。此处我们以曲线调整图层为例如如果我们对原图层新建调整图层, 并且对调整图层使用图层混合模式, 如图13.1所示。

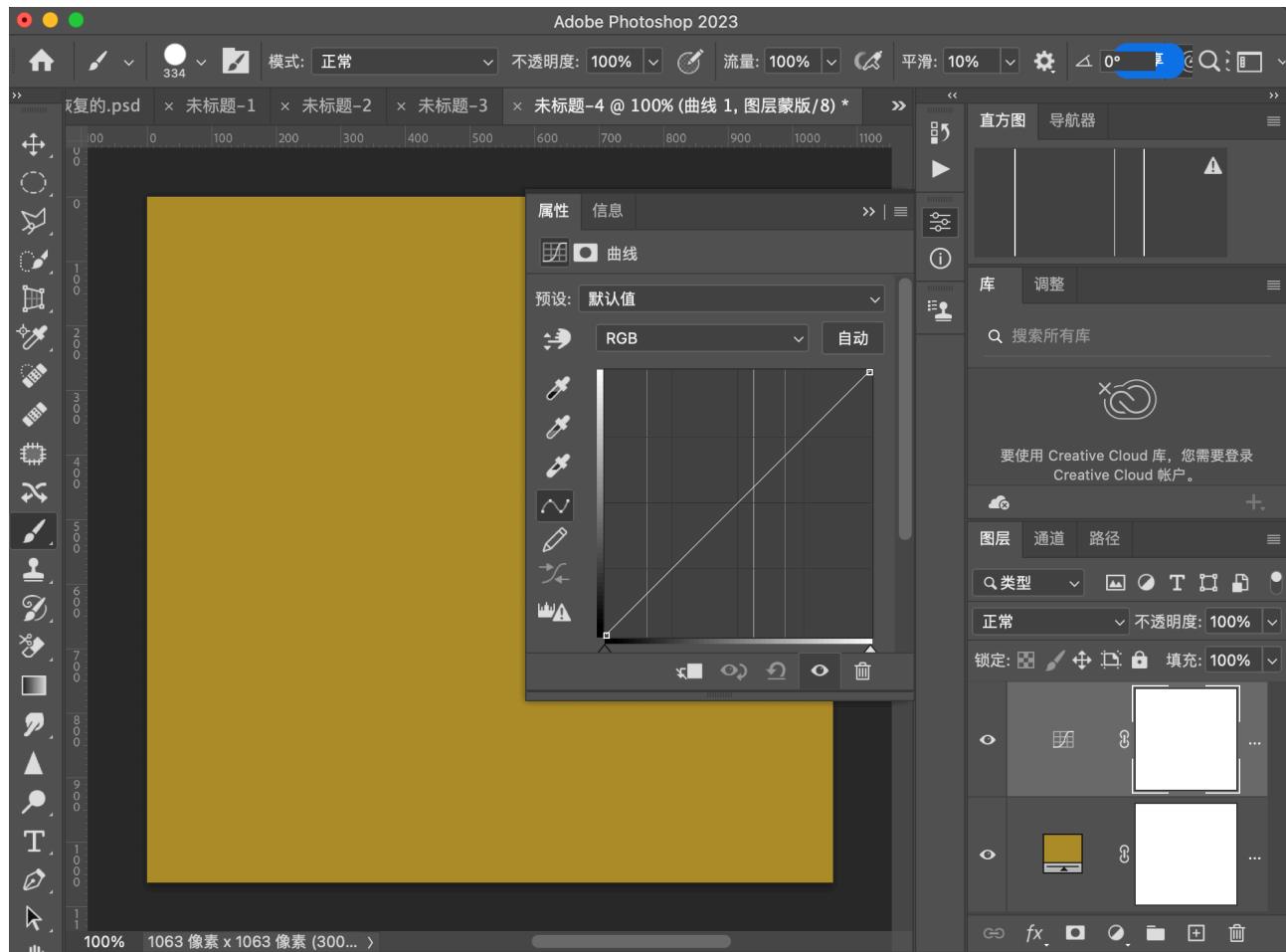


图 13.1

如果是正常模式，则相当于原图，但是只要我们选择一个别的模式，或拉一下曲线，结果就会不同。如图13.2所示。

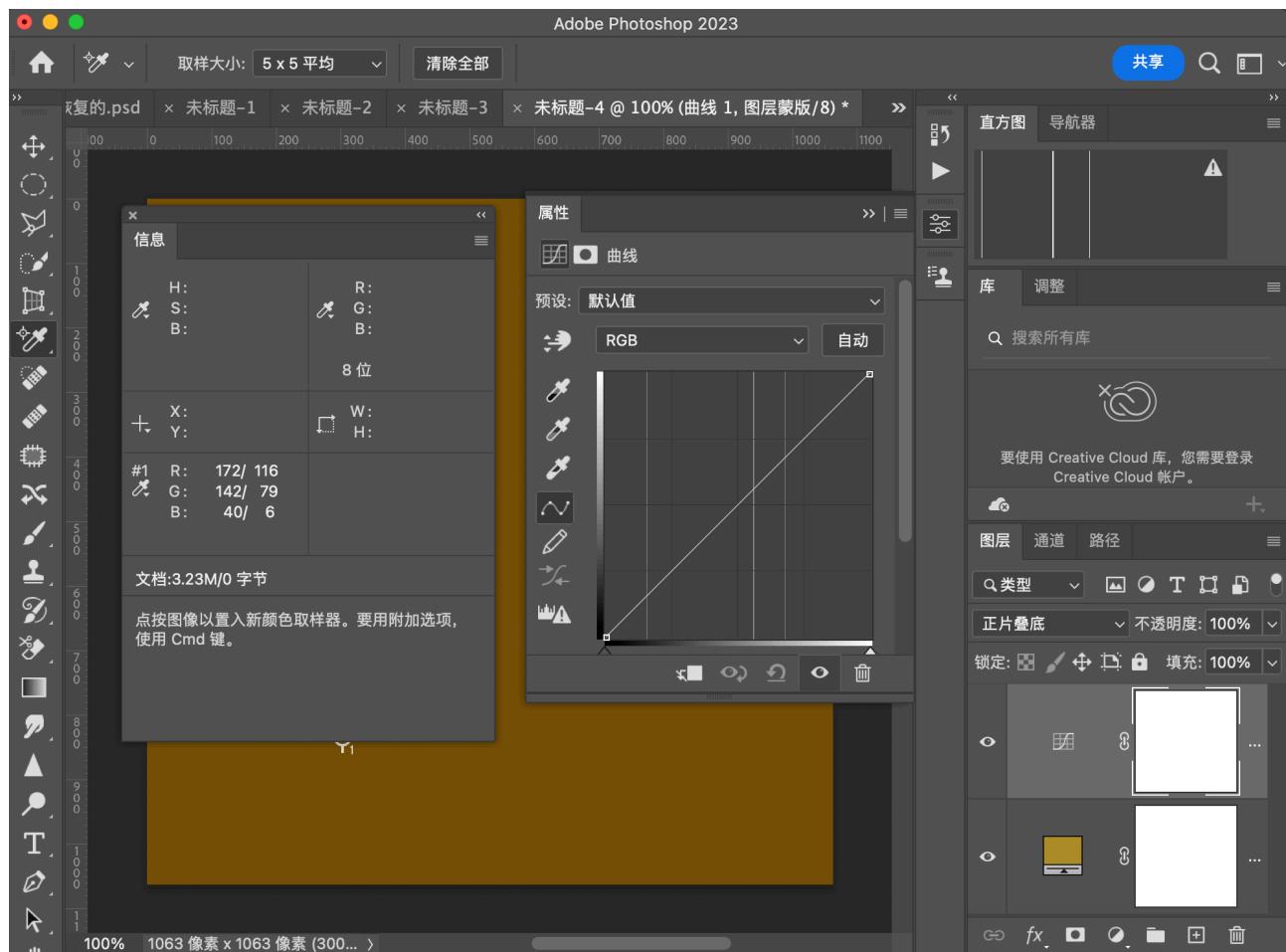


图 13.2

我们可以看到在不拉曲线的前提下，结果和直接用原图做正片叠底是一样的。如图13.3所示。

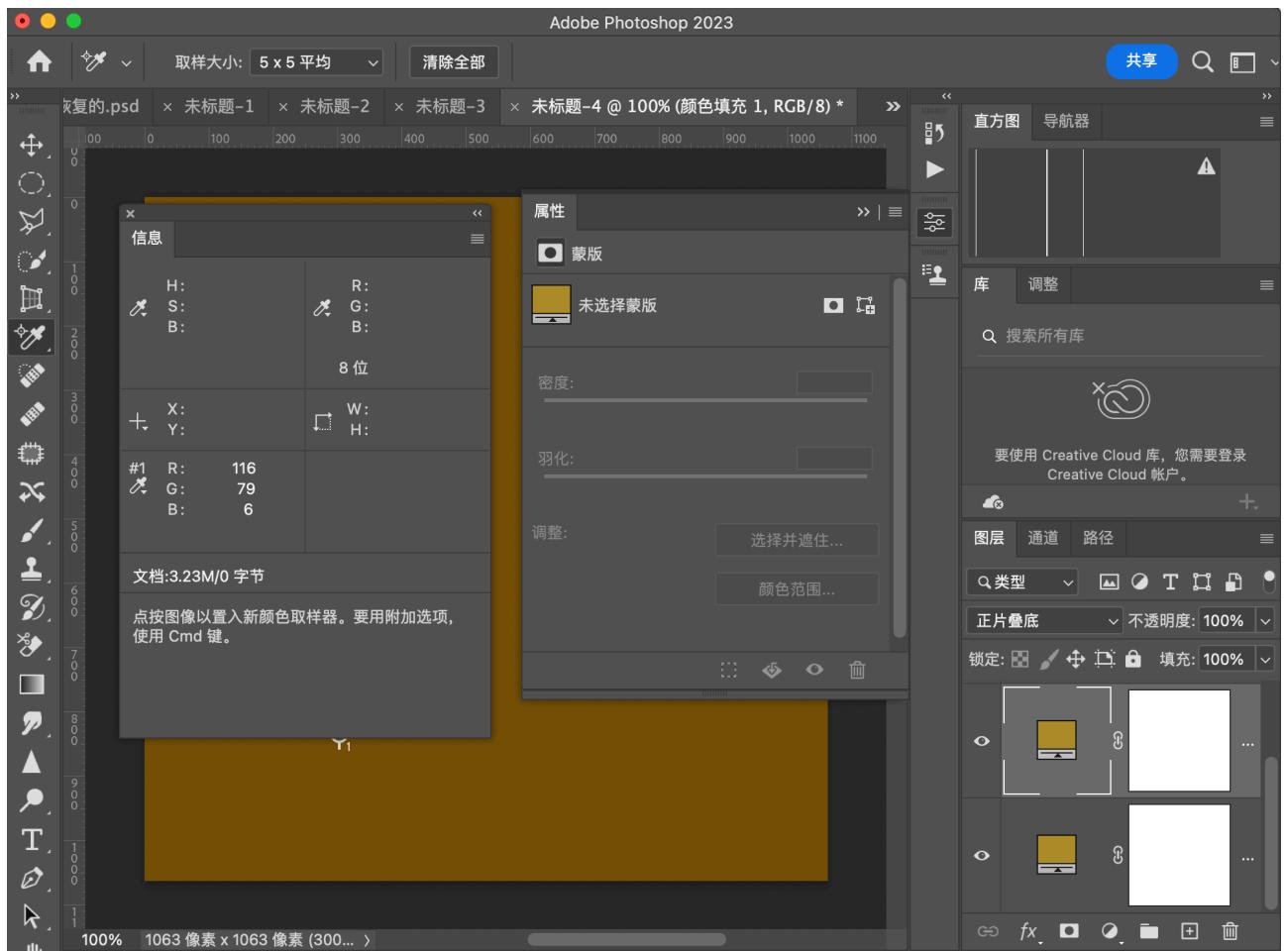


图 13.3

第 14 章 工具和混合模式

内容提要

- 在 PS 中，不止是图层可以选择图层混合模式，有一些工具也可以使用图层混合模式，不同的是，这些都是直接修改原图层，相当于，新建空白图层，再在空白图层上产生新的像素，并且此空白图层作为混合图层。换句话说，图层的混合模式是非破坏性的，工具的混合模式是破坏性的。

14.1 画笔类

14.1.1 画笔工具

画笔工具是最典型的破坏性编辑工具，因为它就像画笔在纸上写字一样，但是，这样操作的本质是新建一个透明图层，在透明图层上面使用画笔绘画，再将当前绘画的图层作为基础图层，带有画笔痕迹的图层作为绘画图层，再进行混合模式的运算。和直接使用透明图层不同的是，结果图层就是基础图层，或者说，PS 将运算过程隐藏了。

14.1.2 铅笔工具

14.1.3 历史记录画笔工具

14.1.4 历史记录艺术画笔工具

14.1.5 修复画笔工具

14.1.6 污点修复画笔工具

14.1.7 颜色替换工具

14.2 油漆桶工具

14.3 图章类

14.3.1 仿制图章

14.3.2 图案图章工具

14.4 渐变工具

14.5 计算类

14.5.1 应用图像

14.5.2 计算

14.6 其他

14.6.1 锐化工具

14.6.2 模糊工具

14.6.3 涂抹工具

第 15 章 图层样式和混合模式

15.1 斜面和浮雕

15.2 描边

15.3 内阴影

15.4 内发光

15.5 光泽

15.6 颜色叠加

15.7 漸变叠加

15.8 图案叠加

15.9 外发光

15.10 投影

15.11 常规混合

15.11.1 混合模式

15.11.2 不透明度

15.12 高级混合

15.12.1 填充不透明度

15.12.2 通道

15.12.3 挖空

15.12.4 将内部效果混成组

15.12.5 将剪贴图层混成组

15.12.6 透明形状图层

15.12.7 图层蒙版隐藏效果

15.12.8 矢量蒙版隐藏效果

第 16 章 混合颜色带 Blend If

内容提要

- 混合颜色带是我们常用的融图工具
- 混合颜色带相当于 GIMP 中的透明颜色

16.1 怎么开启

开启方法，就是双击

16.2 原理

混合颜色带有两个控制带，四个滑块，每个滑块可以分裂为两个小滑块，这些东西看似复杂，但是实际上都控制一件事，就是决定当前图层有哪些像素需要变成透明的。那么如何决定，决定的方式又是哪些。

上方的控制带决定在两个滑块中间的像素保留，在两个滑块之外的变透明。

下方的控制带决定下方图层像素在两个滑块之外的像素，其上方的像素变透明。

提示

- 简单来说就是，两个控制带都是让本图层的像素点变透明，一个是用本图层的像素来决定，一个是指定下方图层的像素来决定。

16.3 色彩模式和颜色混合带

如果我们要选取某些像素点，首先我们需要知晓当前的图片的色彩空间是什么，如果是 RGB，那么选择灰色、红色、绿色、蓝色的表现会有什么不同。

16.3.1 RGB

RGB 空间有四个选项，分别是灰色、红色、绿色、蓝色。计算方法相同，都是非常直白的使用对应通道的数值来计算。

如果选择的是红色、绿色或蓝色，那么如果滑动上面的滑块，则是根据本图层的对应通道值来决定哪些像素透明。总范围是 [0, 255]，滑块中间的范围是 [50, 200]，那么本图层中，满足通道值属于 [0, 50] 如果滑动下面的滑块（这个是最常见的），则此时我们就是使用

$$gray = 0.3 \times R + 0.59 \times G + 0.11 \times B \quad (16.1)$$

并且，我们容易得出 $gray \in [0, 255]$ 。所以对于灰色，我们需要按照上面计算一下结果，然后根据这个结果在 [0, 255] 之间的取值进行处理。

16.3.2 CMYK

RGB 转 CMYK

$$\begin{aligned} K &= 1 - \text{Max}(r, g, b) \\ C &= \frac{1 - r - K}{1 - K} \\ M &= \frac{1 - g - K}{1 - K} \\ Y &= \frac{1 - b - K}{1 - K} \end{aligned} \quad (16.2)$$

CMYK 转 RGB

$$\begin{aligned} R &= 255 \times (1 - C) \times (1 - K) \\ G &= 255 \times (1 - M) \times (1 - K) \\ B &= 255 \times (1 - Y) \times (1 - K) \end{aligned} \quad (16.3)$$

在得到了 RGB 数值之后，我们就可以计算出灰色数值

16.3.3 LAB

LAB 色彩空间共有三个维度分别是 Lightness, A channel, B channel

16.3.4 灰度

灰度就是将 RGB 数值计算为灰色，但是这里灰色的计算方式有些许不同，这里的计算方式是

$$gray = 0.4 \times R + 0.4 \times G + 0.2 \times B \quad (16.4)$$

第 17 章 速查表

表 17.1: 快捷键和中性色

名称	快捷键	中性色
正常	CMD/CTRL + SHIFT + N	
溶解		
变暗	CMD/CTRL + SHIFT + K	白
正片叠底	CMD/CTRL + SHIFT + M	白
线性加深	CMD/CTRL + SHIFT + A	白
颜色加深	CMD/CTRL + SHIFT + B	白
深色		白
变亮	CMD/CTRL + SHIFT + G	黑
滤色	CMD/CTRL + SHIFT + S	黑
线性减淡	CMD/CTRL + SHIFT + W	黑
颜色减淡	CMD/CTRL + SHIFT + D	黑
浅色		黑
叠加	CMD/CTRL + SHIFT + O	中性灰
柔光	CMD/CTRL + SHIFT + F	中性灰
强光	CMD/CTRL + SHIFT + H	中性灰
点光	CMD/CTRL + SHIFT + Z	中性灰
亮光	CMD/CTRL + SHIFT + V	中性灰
线性光	CMD/CTRL + SHIFT + J	中性灰
实色混合	CMD/CTRL + SHIFT + L	中性灰
差值	CMD/CTRL + SHIFT + E	黑
排除	CMD/CTRL + SHIFT + X	黑
减去		黑
划分		黑
色相	CMD/CTRL + SHIFT + U	
饱和度	CMD/CTRL + SHIFT + T	
颜色	CMD/CTRL + SHIFT + C	
明度	CMD/CTRL + SHIFT + Y	

)

表 17.2: 27 种图层混合模式公式汇总

名称	公式	添加填充
正常	a	$fill \times result + (1 - fill) \times b$
溶解	a	$Random_fill(b, a)$
变暗	$Min(b, a)$	$fill \times result + (1 - fill) \times b$
正片叠底	$b \times a$	$fill \times result + (1 - fill) \times b$
线性加深	$b - (1 - a)$	$b - (1 - a) \times fill$
颜色加深	$1 - \frac{1-b}{1-(1-a)}$	$1 - \frac{1-b}{1-(1-a) \times fill}$
深色	$Min(Sum(Pix_b), Sum(Pix_a))$	$fill \times result + (1 - fill) \times b$
变亮	$Max(b, a)$	$fill \times result + (1 - fill) \times b$
滤色	$1 - (1 - b) \times (1 - a)$	$fill \times result + (1 - fill) \times b$
线性减淡	$b + a$	$b + a \times fill$
颜色减淡	$\frac{b}{1-a}$	$\frac{b}{1-a \times fill}$
浅色	$Max(Sum(Pix_b), Sum(Pix_a))$	$fill \times result + (1 - fill) \times b$
叠加	$\begin{cases} 2ba & \text{当 } 0 \leq b \leq 0.5 \\ 1 - 2(1 - b)(1 - a) & \text{当 } 0.5 < b \leq 1 \end{cases}$	$fill \times result + (1 - fill) \times b$
柔光	$\begin{cases} (2a - 1)(b^2 - b) + b & \text{当 } 0 \leq a \leq 0.5 \\ (2a - 1)(\sqrt{b} - b) + b & \text{当 } 0.5 < a \leq 1 \end{cases}$	$fill \times result + (1 - fill) \times b$
强光	$\begin{cases} 2ba & \text{当 } 0 \leq a \leq 0.5 \\ 1 - 2(1 - b)(1 - a) & \text{当 } 0.5 < a \leq 1 \end{cases}$	$fill \times result + (1 - fill) \times b$
点光	$\begin{cases} Min(b, 2a) & \text{当 } 0 \leq a \leq 0.5 \\ Max(b, 2(a - 0.5)) & \text{当 } 0.5 < a \leq 1 \end{cases}$	$fill \times result + (1 - fill) \times b$
亮光	$\begin{cases} 1 - \frac{(1-b)}{2a} & \text{当 } 0 \leq a \leq 0.5 \\ \frac{b}{1-2(a-0.5)} & \text{当 } 0.5 < a \leq 1 \end{cases}$	$fill \times result + (1 - fill) \times b$
线性光	$b + 2a - 1$	$\begin{cases} b + 2a \times fill - 1 & \text{当 } 0 \leq a \leq 0.5 \\ b + 2a \times fill - 1 \times fill & \text{当 } 0.5 < a \leq 1 \end{cases}$
实色混合	$\begin{cases} 1 & b + a \geq 1 \\ 0 & \text{else} \end{cases}$	$\begin{cases} 0 & \frac{fill \times a + b - fill}{(1 - fill)} < 0 \\ \frac{fill \times a + b - fill}{(1 - fill)} & 0 \leq \frac{fill \times a + b - fill}{(1 - fill)} \leq 1 \\ 1 & \frac{fill \times a + b - fill}{(1 - fill)} > 1 \end{cases}$
差值	$ b - a $	$ b - a \times fill$
排除	$b + a - 2ba$	$fill \times result + (1 - fill) \times b$
减去	$b - a$	$fill \times result + (1 - fill) \times b$
划分	$\frac{b}{a}$	$fill \times result + (1 - fill) \times b$
色相	(H_a, S_b, Y_b)	$fill \times result + (1 - fill) \times b$
饱和度	(H_b, S_a, Y_b)	$fill \times result + (1 - fill) \times b$
颜色	(H_a, S_a, Y_b)	$fill \times result + (1 - fill) \times b$
明度	(H_b, S_b, Y_a)	$fill \times result + (1 - fill) \times b$

第 18 章 参考文档

<<http://www.simplefilter.de/en/basics/mixmods.html>>

<<https://printtechnologies.org/wpcontent/uploads/2020/03/pdfreference1.6addendumblendmodes.pdf>>