> 2021/3/10 ~ 2021/3/18
>
> DIP(Digital Image Processing) pre 的研究稿.
>
> 内容参考知乎和百度贴吧等诸多平台文章, 另有笔者的延伸思考.
>
> 如有错误或迷惑之处, 欢迎发送 Issue 交流.

$alpha$ 通道

一些说明

首先, 在我们的讨论中, 颜色通道的取值范围均为 $[0, 1]$.

比如对于我们都熟悉的 r, g, b 通道而言, 0 表示完全不发光, 1 表示以最高亮度发光. 而 $r = g = b = 1$ 就代表白色.

> 对于实际的图片, 取值范围则可能为 {0,1,...,255} 或 {0,1,...,65535} 等等,
>
> 在程序实现时, 应先实现 `uint2float(n)` 和 `float2uint(x)` , 完成实际情况和本文推导之间的转换.

何为 alpha 通道

一些图片格式支持 r, g, b 以外的第四个通道, 称为 $alpha$ 通道, 简记为 a 通道, 它代表三个颜色的共同**不透明度**. 所以对于 a 通道而言, 0 表示完全透明, 1 表示完全不透明.

> 如果你容易记反, 或许可以考虑认为 a 是 r, g, b 的一个"权值". 当然如果这徒增烦恼, 那就算了. 另外, $alpha$ 这个名字没什么特别的含义, 或许只是因为这是除 rgb 以外的第一个通道.

带 alpha 通道的像素混合

考虑将一个前景图片叠加到一个背景图片上的情景:
对于前景图片某一个像素, 如果它的透明度为 $a \in [0, 1]$, 颜色为 $c \in [0, 1]$ (这里 c 可取 r, g, b),
那么将其叠加到不透明的背景颜色 $c_0 \in [0, 1]$ 上时, 最终得到的图像在这一像素点上的颜色显然会由 c_0 和 c 共同决定.
一般而言, 具体的混合公式为:

$$c_{mix} = a \cdot c + (1 - a) \cdot c_0$$

这其实就是给 c 和 c_0 分别赋予了 a 和 $(1 - a)$ 的权重后相加, 而且由于两个权重之和为 1, 所以 c_{mix} 不会溢出到 $[0, 1]$ 之外.
我们也可以通过考虑取极端值直观感受一下这个公式:
当 $a = 0$ 时, 此时前景像素纯透明, 则颜色完全等于背景色;
当 $a = 1$ 时, 前景完全不透明, 因此颜色完全等于前景色.

png 在单色背景下的预览

背景色 $c_0 \in [0, 1]$ 的情形

png 格式图片的每一个像素可以具有透明值, 因此在查看这类图片时, 图片背后的背景颜色会参与决定图像的最终颜色.
在网页中, 背景色常常为白色; 在图片预览界面则更多的是深灰色或纯黑色, 而在一些专业软件如 Photoshop 中, 背景色则为灰白交替的网格等等.

由于这一特性, 一张具有透明度的 png 图像可能在不同的背景颜色下呈现出不同的色彩. 利用上一节讨论的公式, 我们可以直接计算出在某一背景颜色下, 一张 png 图片的实际颜色. 这相当于去掉了原图片中的 a 通道, 得到了只有 r, g, b 三个分量的图片. 由于叠加公式是一个逐点的公式, 因此只需要对图片中的每一个像素点都应用上述公式即可.

下面我们用代码来表述, 并假设背景色的取值为一固定值 `c0` .
则对于一个颜色为 `c` 且透明度为 `a` 的像素点, 其叠加在背景上得到的像素颜色 `cmix` 为:

```
cmix = a * c + (1 - a) * c0
```

枚举每一个像素对其进行上述计算, 把结果保存到一个新的像素矩阵即可:

```
# (这是伪代码)
for pixel in image:                          # 对于图片矩阵的每一个像素
    a = pixel.alpha                          # 当前 a 分量
    pixel.r = a * pixel.r + (1 - a) * c0     # 计算 r 分量
    pixel.g = a * pixel.g + (1 - a) * c0     # 计算 g 分量
    pixel.b = a * pixel.b + (1 - a) * c0     # 计算 b 分量
    pixel.a = 1                              # 令 a = 1 让像素变为完全不透明
```

背景色为纯黑/白色的情形

上面讨论中得到的一般的公式为

$$c_{mix} = a \cdot c + (1 - a) \cdot c_0$$

如果背景特取为白色, 即 $c_0 = 1$, 则上述公式化为:

$$c_W = a \cdot c + 1 - a$$

如果背景特取为黑色, 即 $c_0 = 0$, 则上述公式化为:

$$c_B = a \cdot c$$

逆过程: 对于两种预览, 反向去找能生成它们的 png

给定两张图片 I_W, I_B , 一个有趣的问题就是, 我们能否找到一个 png 图片 I , 使得它在白色背景下看到的是 I_W , 在黑色背景下看到的又是 I_B 呢?
可以想见, 这是一个逐点的性质, 即只要对每个像素点实现这一效果, 则整张图就产生上述效果. 所以只需对一个单独的像素点进行考虑. 设待求的 I 的某个像素 $P = (r, g, b, a)$, 这一像素在白色背景下得到图片 I_W 的像素 $P_W = (r_W, g_W, b_W)$, 在黑色背景下得到图片 I_B 的像素 $P_B = (r_B, g_B, b_B)$.
把上面这句话写成公式的形式, 即:

$$
\begin{cases}
r_W = a \cdot r + 1 - a \\
g_W = a \cdot g + 1 - a \\
b_W = a \cdot b + 1 - a \\
\\
r_B = a \cdot r \\
g_B = a \cdot g \\
b_B = a \cdot b
\end{cases}
$$

其中, (r_W, g_W, b_W) 和 (r_B, g_B, b_B) 都是已知量, 而 (r, g, b, a) 是待求的未知量.

我们尝试把上面式子化简一下, 首先是把底下那三行分别代入上面那三行, 得到:

$$
\begin{cases}
r_W = r_B + 1 - a \\
g_W = g_B + 1 - a \\
b_W = b_B + 1 - a
\end{cases}
$$

再把 a 单独移到等式左侧, 得到:

$$
\begin{cases}
a = r_B - r_W + 1 \\
a = g_B - g_W + 1 \\
a = b_B - b_W + 1
\end{cases}
$$

不过此时麻烦来了. 如果这三行同时成立, 就意味着这三行的等式右边都相等, 即

$$
r_B - r_W = g_B - g_W = b_B - b_W
$$

但这个条件是很难满足的, 怎么可能那么巧两个像素的各个分量之间的差距都相同呢!

满足条件的一种方法: 灰度图 (或叫做亮度图 Luminance)

倒也不必气馁, 我们做一个简单的观察: 如果两张图 I_W 和 I_B 都是灰度图, 那么每个像素都有 $r_B = g_B = b_B$ 且 $r_W = g_W = b_W$, 于是这三行就可以同时成立了! 所以我们不妨先针对灰度图来推导公式, 并自然地记 $r_* = g_* = b_* \triangleq c_*$, 这里 c 可以理解为 color 或 channel. 于是上述公式进一步可以化为:

$$
\begin{cases}
a = c_B - c_W + 1 \\
c = \dfrac{c_B}{a} \quad \left(= \dfrac{c_B}{c_B - c_W + 1} \right)
\end{cases}
$$

因此, 对于对两张灰度图 I_B, I_W 求 I 的问题, 只需要令 I 的每个对应像素取 $P = (c, c, c, a)$ 即可. 其中 c 和 a 由上式给出.

注意事项

观察上面的公式, 我们注意到: 如果 $c_B > c_W$, 则会导致 $a > 1$, 这超出了合法的取值范围!
而且, 如果我们的 I_W 和 I_B 是任意选取的, 则这个概率甚至达到了 50%! 不过我们也没有好的办法, 如果在计算时发现 $a > 1$, 则只好把它规范化到 $a = 1$, 否则可能出现意想不到的(未定义)结果.
(当然, 我们也可以再看一眼 c 的表达式, 然后发现这个无论如何都不会溢出...)

不过, 一旦进行了这样的处理, 之前的推导就不再严格成立了! 于是, 这些不严格的像素的存在使得 I 无法完美地还原出 I_B 和 I_W. 作为应对的策略, 即为了尽量避免 $c_B > c_W$, 我们可以让一张图片上的像素的 c_B 都普遍小一点. 也就是说, 在选择 I_B 和 I_W 时, 让 I_B 的亮度在整体而言相较 I_W 更"暗"一些, 这样上面的概率就会显著低于 50% . 当然, 极端地, 你也可以通过预处理让 I_B 的**所有**像素的亮度都 < 0.5, 让 I_W 的**所有**像素的亮度都 > 0.5, 这样概率就会严格等于 0%. 比如你让 P_B 和 P_W 分别对黑色和白色求平均.

> **棋盘法/隔行法渲染:**
> 主要思想就是, 我们不是希望对应位置的 P_W 的亮度大于 P_B 嘛, 那我们就干脆先预处理两张图, 让 I_W 的奇数点为纯白色, 让 I_B 的偶数点为纯黑色. 这样预处理后, 对于每个像素自然就满足条件了! 当然, 进行这个预处理后, 如果考虑到人脑会自动给相近的像素做做平均, 所以在人看来, I_W 的像素点的亮度差不多都在 0.5 之上, I_B 的像素点亮度则都 < 0.5. 既然如此, 棋盘法渲染后的 I_W 和 I_B 的人眼观感, 不仅在颜色上和对白色/黑色做平均的方法没什么区别, 而且由于棋盘法渲染导致如果放大去看图片, 上面会有很多洞, 所以实际观感会更差. 当然, 这种棋盘法的思想在其它地方自有其大用.

彩色图真的没办法吗?

刚才之所以考虑灰度图, 根本原因是那一组方程式约束了 a 的取值.
我们不禁在想, 如果 r, g, b 通道分别有它自己对应的 a_r, a_g, a_n 通道, 该有多好. 这当然是不可能的, 但这不妨碍我们先假设有这三个通道, 来推导公式. 然而我们立即发现, 这其实就是相当于把每个彩色图看做是 3 个独立的灰度图. 于是此时的公式只是给之前从灰度图推导出的公式加一个 c, 其中 c 可以取 r, g, b 三种值, 表示对 r, g, b 三个分量均成立.

$$\begin{cases} a_c = c_B - c_W + 1 \\ c = \dfrac{c_B}{a_c} \end{cases}$$

于是对于彩色图 I_W 和 I_B 而言, 我们按照颜色通道进行拆分并配对, 得到 $(I_{W_r} \quad I_{B_r}), (I_{W_g} \quad I_{B_g}), (I_{W_b} \quad I_{B_b})$, 再对每一组求出相应的 I_r, I_g, I_b. 最后, 把这三个 I_r, I_g, I_b 揉成一张图:

- 对于 r, g, b 通道, 很简单, 直接对应分量搬过去就好了.
- 对于 a 通道, 没什么好的办法, 直接简单粗暴地对 a_r, a_g, a_b 取个平均后赋值给 a.
  由于 a 是一个取平均得到的近似值, 其不等于 a_r, a_g, a_b 中的任何一个, 所以得到的 I 在每个颜色上都不能完美还原出 I_W 和 I_B.

然而值得指出的是, 对于上一节讨论的灰度像素而言, 只要 a 不越 $[0, 1]$ 的界, 则我们解出来的 c 和 a 是**完美解**: 即 P 能在白色背景和黑色背景下分别能确切地还原出 P_W 和 P_B. 然而对于彩色图而言, 求出来的 c 的确是完美解, 但 a_c 不是! 而不完美的根源正是我们想要有三个 a_c, 但现实是我们只有一个 a 通道, 所以不得不取平均来妥协.
所以, 为了让彩色图的效果**尽可能**好, 我们不妨从 a_c 这个点切入. 具体而言, 就是想尽可能地让 $a_r \approx a_g \approx a_b$ 尽可

能地接近.

而根据上面的公式, 也就是说, 我们希望

$$r_B - r_W \approx g_B - g_W \approx b_B - b_W$$

的约等于号能尽可能取等. 我们从两个角度考虑优化的方法:

方法 1: 让彩色没那么彩色

首先, 正如之前我们利用灰度图让上面的约等于号严格取等, 在这里, 为了颜色分量取值近似相等, 一个办法就是让彩色没那么彩色, 比如*让图片和图片的灰阶图做插值*. 当然, 你可以同时对 I_W 和 I_B 进行上述处理; 但倘若你更关心 I_B 的色彩, 而没那么关心 I_W, 你可以直接无脑把 I_W 处理成灰阶图, 然后对 I_B 只做轻微的去颜色处理. 反之亦然.

方法 2: 仍是变暗

另一个办法是让 I_B 的所有像素的所有分量都乘以一个小的比率 $ratio$. 它的原理也是自然而然的. 比如我们假设某个像素的三个分量为 [0.15, 0.2, 0.06], 然后我们取比率 $\text{ratio} = 0.1$, 那么上面的像素就变成了 [0.015, 0.02, 0.006]. 这直接让上面的约等于号接近了 10 倍! 而且由于颜色是由三个分量之间的比例决定的, 所以颜色并没有损失(虽然的的确确损失了这个像素的亮度). 所以, 和灰度图那里是讨论相似, 我们让 I_B 在能接受的范围内尽可能暗就完事了!

方法 3: 牺牲白底图, 强行 $a_r = a_g = a_b$

> ⚠️ 本方法可能仅在理论上有效, 是笔者在后文码字中, 临时想到的.

我们重新审视一下给我们造成麻烦的公式:

$$\begin{cases} a = r_B - r_W + 1 \\ a = g_B - g_W + 1 \\ a = b_B - b_W + 1 \end{cases}$$

一个灵光乍现: 我们为什么不先进行一个预处理, 让上面这个式子对每个像素都成立呢? 即对于每个像素, 都让 $r_B - r_W = g_B - g_W = b_B - b_W$ 成立! 当然, 正如之前提到的, 随便选取两个图是不可能成立的, 但如果我们的目标是让某一个图尽可能地保真, 而另一张毁容成什么样子都没什么所谓, 那我们就可以通过预先修改不需要保真的图, 来预先让上式成立.

于是假设我们的目的是尽可能让黑底图 I_B 保真, 那么我们就预先修改 I_W. 由于是为了让每个像素的对应分量之差相同, 我们只需再决策如何选取这个差 (diff) 好了.

简单粗暴地, 我们决定选取为三个分量的 diff 的平均值好了!

也就是说, 令

$$\text{diff} = \frac{(r_W - r_B) + (g_W - g_B) + (g_W - g_B)}{3}$$

然后修改 P_W 的三个颜色分量的值:

$$\begin{cases} r_W := r_B + \text{diff} \\ g_W := g_B + \text{diff} \\ b_W := b_B + \text{diff} \end{cases}$$

这样新得到的 $\hat{I}_W$ 和之前的 I_B 搭配在一起, 就满足条件了. 然后我们就能愉快地用上一节的方法来求取 I, 由于这种情况下 $a_r = a_g = a_b$, 所以得到的是**完美解**, 至少对于 I_B 而言是完美解. 另外, 如果你并不希望 $\hat{I}_W$ 完全放飞自我, 即仍然"能看", 那么和之前一样, 你需要让 I_W 尽量是偏灰阶的, 而 I_B 则越暗越好. 其原理和之前是类似的 ~~(见课后习题)~~ .

> **注1:** 上述粗暴地取平均值或许并非最佳方案. 因为无论 diff 取什么, 都能让 a 相同, 但肯定有些 diff 的取值会更为合适, 比如可以让改动前后这个像素的差异看起来较小. 而对于这种差异的一个量化方法就是这个像素在修改前后被**人眼感知**的**亮度**的差别. 在网络上可以查到, 人眼感知亮度的经验公式为
>
> $$\text{Brightness} = R * 0.299 + G * 0.587 + B * 0.114$$
>
> 因此我们的需求写成公式就是
>
> $$\Big(0.299(r_B + d) + 0.587(g_B + d) + 0.114(b_B + d)\Big) = \Big(0.299 r_W + 0.587 g_W + 0.114 b_W\Big)$$
>
> 其中, 左边是修改之后的像素的亮度, 右边是修改前像素的亮度. 解之得:
>
> $$\text{diff} = 0.299(r_W - r_B) + 0.587(g_W - g_B) + 0.114(g_W - g_B)$$
>
> ---
>
> **注2:** 此外, 注意到还应该保证 r_W, g_W, b_W 不越界 $[0, 1]$. 当然你可以直接在最后进行规范化, 不过在本节中, 目标是保证黑底图的完美, 所以我们决定对 diff 下手, 也就是说, diff 的取值应该满足:
>
> $$\begin{aligned} &(\text{diff} \geq -\min\{r_B, g_B, b_B\}) \land (\text{diff} \leq 1 - \max\{r_B, g_B, b_B\}) \\ \implies &-\min\{r_B, g_B, b_B\} \leq \text{diff} \leq 1 - \max\{r_B, g_B, b_B\} \end{aligned}$$
>
> 当然, 如果 diff 成功被这一限制约束到了, 那么注释 1 中的保持亮度也就无从谈起了.
> 这又是取值范围带给我们的约束, 不过也实属无奈.

最后的一点思考: 为什么在灰阶等情况下有完美解?

在研究这个问题之前, 我一直以为这个问题无论如何只会有近似解, 即生成的 I 一定会在白色背景下会留下 I_B 的蛛丝马迹, 并在黑色背景下留下 I_W 的痕迹. 然而正如上面讨论的那样, 在理论上, 灰阶和保留一张图的色彩的情况下是可以得到完美解! 这是什么原理!? 当然, 上文的数学推导已经证明了这一点, 不过从另一个方面来看, 这个道理则更为自然.

考虑到, 一个不透明的灰度像素 P 由 1 个实数完全决定, 这个实数便代表了其亮度; 一个带透明度的灰度像素 P_a 则是由 2 个实数表达, 一个代表明暗度, 另一个代表透明度. 所以从信息量的角度而言, 由于 P_a 具有 2 个维度, 所以同样尺寸的**带透明度的灰阶图**具有两倍于一张**不透明灰阶图**的信息量, 所以**一张前者**能表达出**两张后者**, 也就不足为奇了. 只是这个表达的过程可能不太容易想到.

同理, 也容易知道为什么对于两张彩色图的情形我们失败了. 因为两张彩图共有 $3+3=6$ 个通道, 而一张 png 图只有 4 个通道, 后者无法容纳前者.

最后, 为什么在舍弃其中一张图的色彩的情况下, 我们成功了呢? 这是因为在这种情况下, 我们保留了一张图的全部色彩信息, 而仅仅保留了另一张图的亮度信息(如果加上经验公式修正), 因此一共 $3+1=4$ 个维度, 刚好能被 png 的 4 个通道表达.