

Demosaic 系列一：简单插值法

从今天起，开始总结去马赛克的算法。去马赛克是 ISP 流程中最为重要、最为热门的几个模块之一，这几个月的重点也一直在这方面。下面对所看的内容进行记录。

1. 双线性插值

太简单了，就是从上下左右取值。如果上下没有，就只算左右；反之亦然。

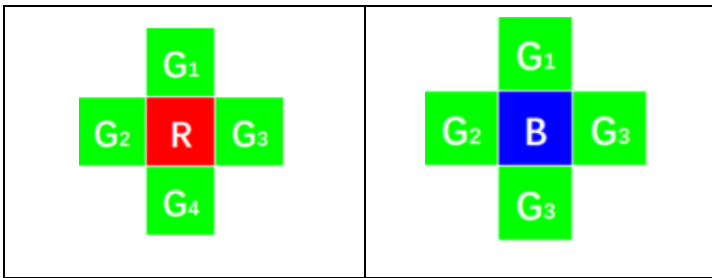
2. 色差法

引入了一个很重要的原理：一个完整的 RGB 三通道图像中，两个相邻像素间对应通道的像素差近似相等。即加入 P1 和 P2 是相邻像素，可以认为：

$$P_1(G) - P_1(R) = P_2(G) - P_2(R)$$

$$P_1(G) = P_1(R) = P_2(G) - P_2(B)$$

1. 先用普通的双线性插值预测 G: $G = 1/4 * (G_1 + G_2 + G_3 + G_4)$



2. 在 R 中补充 B、在 B 中补充 R:

	$P_c(G) - P_c(R) = \frac{1}{4} * \sum_{n=1}^4 P_n(G) - P_n(R)$ <p>$P_c(G)$ 已知，可求出 $P_c(R)$</p>
--	--

3. 在 G 中补充 R

	$P_c(G) - P_c(R) = [(P_1(G) - P_1(R)) + (P_2(G) - P_2(R))] \div 2$ <p>$P_c(G)$ 已知，可求出 $P_c(R)$</p>
--	--

3. HA 算法

引入了另一个重要思想：**利用梯度判断边缘**。梯度小的地方被认为是纹理方向，因此使用这个方向的色差进行插值。

这个方法还有一个前提，属于是【色差相似】的扩展：【一个小范围内的点】与【小范围平均下来的值】也满足条件。

如下，对于中心点Rc, 选择其邻域1x5的数据。根据色差一致性可得到如下式子：

```
IntpG - Rc = Avg_g - Avg_r; //IntpG表示中心点插值
其中：
Avg_g = (G0+G1)/2; Avg_r = (R0+2Rc+R1)/4;
->IntpG - Rc = -0.25*R0 + 0.5*G0 - 0.5*Rc + 0.5*G1 - 0.25*R1
->IntpG = -0.25*R0 + 0.5*G0 + 0.5*Rc + 0.5*G1 - 0.25*R1
```



1. 预测 G (色差法是根据双线性插值预测)：通过梯度来判断预测。计算出横向竖向梯度，然后比较大小，根据小的梯度的方向进行插值。

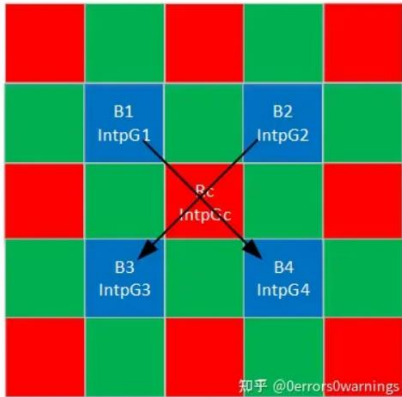
R/B 预测 G	R 预测 B
<ul style="list-style-type: none"> 计算H,V两个方向的梯度 <pre>Grad_h = abs(G1,G2) + abs(2*Rc,(R1+R2)); Grad_v = abs(G0,G3) + abs(2*Rc,(R0+R3));</pre> <p>知乎 @0errors0warnings</p> <ul style="list-style-type: none"> 根据梯度，计算最终插值结果 <pre>if(Grad_h < Grad_v) IntpG = IntpG_h; else if(Grad_h > Grad_v) IntpG = IntpG_v; else IntpG = (IntpG_h + IntpG_v)/2;</pre> 	<ul style="list-style-type: none"> 计算对角线梯度 <pre>Grad_N = abs(B1,B4) + abs(2*IntpGc,(IntpG1+IntpG4)) Grad_P = abs(B2,B3) + abs(2*IntpGc,(IntpG2+IntpG3))</pre> <p>知乎 @0errors0warnings</p> <ul style="list-style-type: none"> 根据梯度，计算最终插值结果 <pre>if(Grad_N < Grad_P) IntpB = (B1+B4)/2 + (2*IntpGc-IntpG1-IntpG4)/4; else if(Grad_h > Grad_P) IntpB = (B2+B3)/2 + (4*IntpGc-IntpG2-IntpG3)/4; else IntpB = (B1+B2+B3+B4)/2 + (2*IntpGc-IntpG1-IntpG2-IntpG3-IntpG4)/4;</pre>

2. R 中预测 B: 基本和上一步一样，只不过是从对角线出发，利用上一步补充的 G，查看梯度，

然后计算（最终插值结果也是通过原理给的色差相近计算而来，只不过变成沿着对角线的 1*5 数据）：

- 计算对角线梯度

```
Grad_N = abs(B1,B4) + abs(2*IntpGc,(IntpG1+IntpG4))
Grad_P = abs(B2,B3) + abs(2*IntpGc,(IntpG2+IntpG3))
```



知乎 @0errors0warnings


- 根据梯度，计算最终插值结果

```
if(Grad_N < Grad_P)
    IntpB = (B1+B4)/2 + (2*IntpGc-IntpG1-IntpG4)/4;
else if(Grad_h > Grad_P)
    IntpB = (B2+B3)/2 + (4*IntpGc-IntpG2-IntpG3)/4;
else
    IntpB = (B1+B2+B3+B4)/2 + (2*IntpGc-IntpG1-IntpG2-IntpG3-IntpG4)/4;
```

3. G 中补充 R：这一步就不用管太多了，直接计算。两种情况，G 左右都是 R，左右都是 B。如果左右都是 R：那就利用原理给的 1*5 的横向矩阵，计算得出 R 值，就是最终的 R 值，然后再利用 1x5 的纵向矩阵，计算 B 值。左右都是 B 同理。

如下，对于中心点Rc，选择其邻域1x5的数据。根据色差一致性可得到如下式子：

```
IntpG - Rc = Avg_g - Avg_r; //IntpG表示中心点插值
其中：
Avg_g = (G0+G1)/2; Avg_r = (R0+2Rc+R1)/4;
->IntpG - Rc = -0.25*R0 + 0.5*G0 - 0.5*Rc + 0.5*G1 - 0.25*R1
->IntpG = -0.25*R0 + 0.5*G0 + 0.5*Rc + 0.5*G1 - 0.25*R1
```



PS：这一步感觉还是随意了，因为边界处还是不太遵守这个理论。因此在边界处使用这个，可能还是会有问题。

4. Malvar-He 算法

一篇简单，但是绝对很有用的文章，我一开始也是觉得就这啊，为啥斯坦福大学的课会强调这个方法呢，直到自己设计的时候，面对 FPGA 工程师反复强调的节约资源要求，我才发现：嗯，线性插值，真香！

方法很简单，就是用色差法，直接看下面一段话：

Specifically, to interpolate G values at an R location, for example (the '+' pixel in Fig. 1), we use the formula

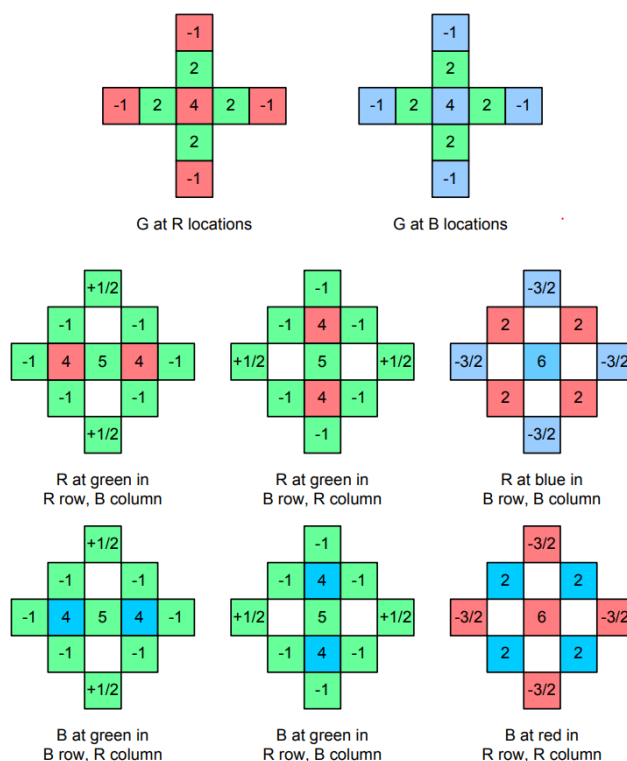
$$\hat{g}(i, j) = \hat{g}_B(i, j) + \alpha \Delta_R(i, j) \quad (2)$$

where $\Delta_R(i, j)$ is the gradient of R at that location, computed by

$$\Delta_R(i, j) \triangleq r(i, j) - \frac{1}{4} \sum_{(m,n) \in \{(0,-2), (0,2), (-2,0), (2,0)\}} r(i+m, j+n) \quad (3)$$

其他颜色也是同理，只不过系数分别是 beta 和 gamma。之后通过一系列图片进行训练，求出 alpha 取什么值，均值最小；beta 和 gamma 同理。

最后作者为了加快速度，还把这三个变量修改一下，这样速度快一点。结果分别是 1/2、5/8、3/4，最终带回到公式，比如 alpha=1/2 就是代入公式 2 中，得出预测 G 值，周围像素的权重，速度非常非常快：



个人感觉有以下几点可以修改：

1. 修改成类似 $3/8$ 这种很完美的数字可以根据 FPGA 的情况进一步逼近，比如算出原始权重是 0.38，乘以一个 4096，得到 1556，即 $1556/4096$ ，这个比 $3/8$ 更好，看 FPGA 的资源情况吧。
2. 可以一窝蜂的直接用 $5*5$ 的窗口去算各个周围的权重，即不像上图 $5*5$ 的像素中有些是 0，而是直接 $5*5$ 即 25 个像素打入到全连接网络算就完事了。