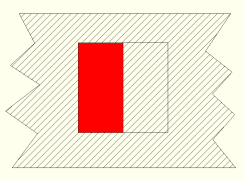
When storing an RGBA image people often wonder whether to store the color components premultiplied by the alpha. For example, if you’ve got a pixel that’s full red (255,0,0) and has a 50% alpha (128), should the RGBA values store ***premultiplied colors*** (128,0,0,128) or ***non-premultiplied*** colors (255,0,0,128).

There are advantages to both schemes. For example, if you’re using non-premultiplied colors, then you can change the opacity of an image simply by tweaking its alpha channel. Or if you want to perform some color correction, say remap bright red to some other color, then you can do that without worrying that the alpha channel has made your bright red look like dark red simply because of transparency.

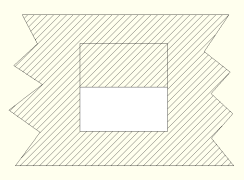
But if you’re using premultiplied colors, then the math of compositing images together is simpler and can avoid a division most of the time.

The key to understanding this is to give up the idea of alpha as transparent colored glass and instead see it as a screen mesh. So if your alpha is 50%, then imagine a pixel where half is covered by your pixel’s color and the other half is completely transparent. Specifically, imagine the left half covered and the right half transparent:

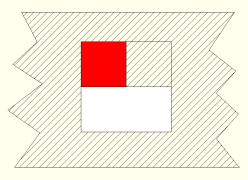


Now the question is whether the RGB values should represent the color of the left half, or only the left half’s contribution to the whole pixel. The advantages of doing the former (non-premultiplied) were outlined above. To see that advantages of premultiplied colors, consider compositing another pixel on top of our half-red one. The new one is white (255,255,255) and 50% transparent. So since we’re using premultiplied colors, our RGBA is (128,128,128,128).

We’ve got this screendoor transparency model, but we don’t know where this half-pixel is compared to the half pixel we’re compositing over. If our new pixel is also on the left side, then we’ll clobber the underlying pixel altogether. If it’s on the right, we’ll patch the hole perfectly. Since we don’t know, we’ll go half-way and turn it 90 degrees, with the bottom half of the pixel opaque white and the top half transparent:



So what do you see when you put the white pixel (foreground) on top of the red one (background)? The bottom half is white, the top-left quarter is red, and the top-right quarter is transparent:



The math for that is easy when the numbers are premultiplied:

R' = RF + (1 - AF)×RB  
G' = GF + (1 - AF)×GB  
B' = BF + (1 - AF)×BB  
A' = AF + (1 - AF)×AB

The result is (192,128,128,192), which makes sense if you think of each color’s contribution to the pixel times its area.

Now the math for non-premultiplied colors is hairier:

R' = (RF×AF + (1 - AF)×RB×AB) / A'  
G' = (GF×AF + (1 - AF)×GB×AB) / A'  
B' = (BF×AF + (1 - AF)×BB×AB) / A'  
A' = AF + (1 - AF)×AB

You don’t need the divide if the foreground or the background is opaque, which is a common situation since people usually start with a fully-opaque image and add things to it.

If you’re using 8-bit colors, then premultiplying might make you lose a bunch of color precision that you’ll need later when you divide out the alpha to do some color correction. So I’d recommend not premultiplying colors if you care about accuracy and precision and have the time to waste during compositing. But premultiplying is better if you’ll be doing lots of compositing and you don’t need complete accuracy during color correction and blurs and such.

Either way, be aware of which scheme you’re using and think through each pixel operation so that the math makes sense given this screendoor transparency model.

[Back to graphics](http://www.teamten.com/lawrence/graphics/)

Alvy Ray Smith的《Alpha and the history of digital compositing》

aA + (1-a)B ，A、B是两张要合成图像。可以这样理解,A代表颜色,a代表透明度,.一旦a和A作完乘运算,即将透明信息添加到颜色信息里了,即混合了透明和色彩.这是Premultiplied Alpha.而Straight Alpha就是单独一个a,没有做乘运算,不含颜色信息

Straight Alpha 也被称为 Unmatted Alpha（即不带蒙板的Alpha）,Premultiplied Alpha 也被称为Matted Alpha（带有背景色蒙板的Alpha），前者将素材的透明信息存放在独立的Alpha通道中，后者则不仅保存了素材Alpha通道中的透明信息，而且还包含有背景RGB通道的透明量-----

转：

1：

Premultiplied alpha is just a different way of representing alphified pixels. If the separate alpha pixel is (r, g, b, a), then the premultiplied alpha pixel is (ar, ag, ab, a).

The reason why it's interesting is that linear combinations of pixels (i.e. a1p1 + a2p2) work better in premultiplied alpha space than in separate alpha space.

For example, taking the 50/50 blend of white and transparent works like this: White is (1, 1, 1, 1) and transparent is (0, 0, 0, 0) in both spaces. So the 50/50 blend is (0.5, 0.5, 0.5, 0.5). In separated space, that's half-transparentgray, but in premultiplied space, that's half-transparent white, which is what you expect.

Linear combinations of pixels occur in a lot of contexts, including:

1. layer compositing
2. blurring and sharpening
3. interpolation
4. scaling
5. rotation, perspective, and other transformations

and probably one or two others.

The symptom of getting alpha wrong is gray fringes when you work with transparent layers.

2  
  
There's pretty detailed explanation here:  
  
<http://www.digitalartform.com/alphaChannel.htm>  
  
  
  
Basically, premultiplication deals with the question of what to do with the RGB color values wherever the alpha is not fully opaque. With Straight alpha, the RGBs are left untouched; with premultiplied, they are blended with a certain color (typically black) and the compositing software is then expected to UNmultiply them back to the original values.  
  
Let's look at a simple example: you're trying to render a red circle over a transparent backgroud. The inside of the circle is red, 100% alpha; the outside can be any color, doesn't matter since the alpha is 0%; and over the edges you have red with X% alpha, since the antialiasing tries to smooth things out by gradually decreasing the alpha from 100% to 0% over those pixels that are "partly inside".  
  
Now, the renderer wants to make sure that wherever there is still some positive alpha, the contribution of the circle will be red. So what it often does is create a LARGER red circle, much larger than necessary - it doesn't matter since the alpha is fading it out anyway. So a pixel that just barely touches the circle will have say 10% alpha but a FULL RED rgb value: 255 0 0. This is STRAIGHT alpha.  
  
If you look at such an image in a software that does NOT understand straight alpha, it will look terrible - instead of a circle you'll get a red blob, smeared all over the place. What premult does is try to overcome that: for that pixel which is 10% alpha, it will also DECREASE THE RED VALUE so that you get a much darker color: 25 0 0, almost black.  
  
If you look at the premult in the same software that ignores the alpha, it will actually look OK since the antialiasing goes into the RGB channeles: the color fades out from red to black. However, if you want a different background, say white, it will look awful again - you'll see a typical black fringe like a halo around your circle.  
  
There are advantages and disadvantages to both methods, but usually Straight is the way to go.