Interpreting Alpha

Andrew Glassner
The Imaginary Institute

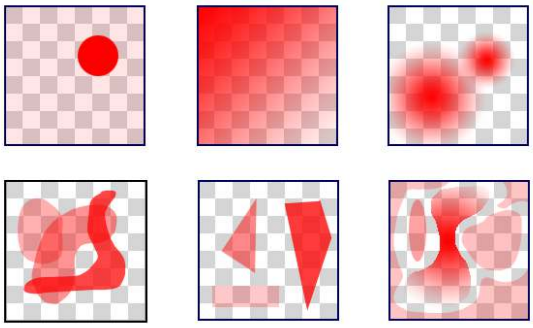

Figure 1. These pixels all have the same value of alpha.

Abstract

Associating alpha values with pixel colors is an important technique in computer graphics, allowing us to create complex composite images. However, the interpretation of the meaning of alpha is often a fluid concept, switching between coverage (a measure of area) and opacity (a measure of a color) based on convenience. It can appear very strange that a single number could represent both of these qualitatively different values at the same time. By tracking coverage and opacity separately through the compositing process, we find that alpha is actually the product of coverage and opacity, so it is only one of these terms if we assume that the other has the value 1. This leads to a simple and consistent understanding of the internal structure of pixels that are described by color and alpha.

1. Introduction

The idea of "alpha" has been a part of computer graphics for over three decades, since it was presented in the classic paper on image compositing [Porter and Duff 1984].

The alpha idea has been used to composite an enormous number of pixels (a rough estimate suggests that alpha blending has been used upwards of 10^{21} times in the entertainment industry alone). The concept of alpha as part of a pixel's color is firmly embedded in our psyches and our code.

 ISSN 2331-7418

Alpha is obviously incredibly useful for compositing images, but what does it really represent? The graphics literature can be hard to nail down on this issue. In their paper, Porter and Duff [1984] sometimes consider alpha to represent the *opacity* of a completely covered pixel:

> If α_A and α_B represent the opaqueness of a semi-transparent object which fully covers the pixel, the computation is well known.

At other times, they consider it to be the area of a pixel that is covered by a colored fragment within it (that area is called the *coverage*):

> If α_A and α_B represent subpixel areas covered by opaque geometric objects, the overlap of objects within the pixel is quite arbitrary.

The authors switch interpretations based on whatever is most useful at each moment in their discussion.

Porter and Duff are not alone in this. In a memo on the subject, Smith [1995] states,

> There are two ways to think of the alpha of a pixel. As is usual in computer graphics, one interpretation comes from the geometry half of the world and the other from the imaging half. Geometers think of "pixels" as geometrical areas intersected by geometrical objects. For them, alpha is the percentage *coverage* of a pixel by a geometrical object. Imagers think of pixels as point samples of a continuum. For them, alpha is the *opacity* at each sample. In the end, it is the imaging model that dominates, because a geometric picture must be reduced to point samples to display—it must be rendered. Thus, during rendering, coverage is always converted to opacity, and all geometry is lost.
>
> The Porter-Duff matting algebra that underlies what we present here is based on a model that is easiest to understand by alternating between the two conceptions.

So once again, alpha seems to be either coverage *or* opacity, depending on one's needs.

In one last example, Blinn [1994] says,

> The α value ... goes by various names: coverage amount, opacity, or simply alpha ... I'm going to call it opacity for now. If it's 0, the new pixel is transparent and does not affect the frame buffer. If it's 1, the new pixel is opaque and completely replaces the current frame buffer color.

Many more quotes like these can be found in the graphics literature, each of which treats alpha as a fluid concept. Alpha sometimes refers to opacity and sometimes coverage, based on whatever is most convenient at any given time. The mathematics and algorithms of many papers on composition also freely move between these interpretations.

A reader could be forgiven for finding this confusing. Coverage is a description of area, and opacity is a property of a material, or a color that represents its appearance in a specific context. It seems unlikely that a single number can mean either of these things depending on someone's momentary preference.

Smith asserts that during rendering all geometry is lost, and only opacity remains. Must that be so? What if we don't throw away the geometry, but instead preserve it in the same way that we preserve opacity?

In this paper we will do just that, and retain both opacity and coverage independently as we work through the compositing process. We'll find that the traditional equation for computing the new alpha after compositing emerges after the algebraic terms cancel one another out. But along the way, we'll develop a better appreciation for what "alpha" is and why it apparently can be be interpreted with such flexibility.

The majority of compositing work uses the most general of the Porter-Duff composition operators, called *over*. This operation supposes that we're building up a pixel's representation by placing some fragment of a new object over one or more fragments already present in that pixel. Using *over*, we can build up a very complex image by working from back to front, layering each new image over the results of previous layering operations. We'll derive our results by analyzing the mechanics of *over*, though the results hold for any kind of compositing.

While our focus here will be on understanding alpha as a measure of pixel coverage or opacity, it's worth noting that alpha has been used in other ways. For example, alpha is frequently used to influence lighting calculations by making it part of a material's description. McGuire and Enderton [2011] demonstrate that when used this way, an alpha value of .5 can equally well refer to a piece of red woven cloth which has holes over half its area, or a transparent red gel which passes half of the light striking it. These will have very different appearances and even different shadows: the cloth will cast a mottled black shadow that might appear as a medium gray, while the gel would cast a red shadow. It is important that programs that use alpha for lighting, rendering, filtering, and other effects carefully document just how they're doing so, or the results may be surprising [McGuire 2012].

Alpha can be used in this modeling sense for applications ranging from fabric to hair to vegetation. In these applications, alpha is often explicitly intended to represent either coverage or opacity.

2. Preliminaries

Closely involved in any discussion of compositing is the idea of *pre-multiplication*. In pre-multiplication, we store a color's components (typically red, green, and blue) already multiplied by alpha. For example, suppose we have a fragment with RGBA = $(1, .5, .25, .5)$ (in this paper, all color values, opacity values, and coverage values will

be in the range $[0,1]$). We could save that in pre-multiplied form as $(.5,.25,.125,.5)$. Note that the alpha value is unchanged, while the R, G, and B values get multiplied by alpha. Pre-multiplication is well known to be a valuable technique; Blinn [Blinn 1994] offers multiple situations where using pre-multiplied colors results in algorithms that are faster or easier to program.

Pre-multiplication also allows us to create *synthetic* colors that cannot be the result of any "natural" rendering or drawing process. So-called "transparent black," with RGBA values $(0,0,0,0)$ is probably the most important of these. Using positive color values along with an alpha of 0 allows us to produce glows and color washes that add color to a pixel without introducing geometry.

In this paper, we write A_c for any color component of a fragment of object A (typically red, green, or blue). Since all components get treated in the same way, and independently of one another, we can focus on just one at a time. Alternatively, A_c can be thought of as a grayscale value. We write A_α for the alpha value of the fragment. So we could write the color and alpha together as (A_c,A_α). If the color is pre-multiplied, we'll write it with a lower-case italic letter, as (a_c,A_α). Note that because the alpha value itself is not pre-multiplied, it retains its capital letter.

Porter and Duff offer thirteen different compositing operators, each of which has their use. By far the most popular is *over*, and that is the one we focus on here. The blending equation describing *over* using non-premultiplied values (often called "raw" or "straight" values) is

$$C_c = \frac{A_\alpha A_c + (1-A_\alpha)B_\alpha B_c}{A_\alpha + (1-A_\alpha)B_\alpha}.$$

The pre-multiplied version is much more efficient:

$$c_c = a_c + (1-A_\alpha)b_c. \tag{1}$$

Note that the result is in lower-case since it's in pre-multiplied form as well.

Suppose we want to use traditional alpha blending to stack up four images: A, B, C, and D. Suppose further the most convenient way for us to compute these is to first form $F = A$ over B, then $G = C$ over D, and then compose the two intermediates to make $H = F$ over $G = (A$ over $B)$ over $(C$ over $D)$. Then, while computing F, we need to find not just the new color for each pixel, but a new alpha, so we can use it in the next stage when finding H.

Porter and Duff [1984] don't offer a formula for computing this new alpha. Both Smith [1995] and Blinn [1994] provide the following expression for the alpha value of the composite $F = A$ over B (shown here in our notation):

$$F_\alpha = A_\alpha + (1-A_\alpha)B_\alpha. \tag{2}$$

There's a lot to like about this formula: it's simple, it's satisfyingly similar to Equation (1), and, thinking it through, it makes intuitive sense. The formula is derived

by Blinn by working through the algebra of the *over* operator. In order to make that operation associative (which is very desirable), this is the necessary expression. But that doesn't provide us with an intuitive interpretation of alpha. In other words, we know how to compute it, but that doesn't tell us how to interpret it.

In this paper we'll re-derive this formula by independently tracking coverage and opacity while compositing with *over*. Equation (2) indeed emerges at the end, and along the way we'll discover how to interpret the meaning of alpha.

3. Opacity and Coverage

In this section we set aside the idea of alpha for a moment, and instead focus independently and specifically on opacity and coverage.

We will associate two numbers with each pixel in image A: the *coverage*, written A_k, and *opacity*, written A_o. These numbers are each derived from the objects that contribute to that pixel.

The pixel's coverage, A_k, is nothing more than the fraction of the pixel that is occupied by the fragment. The pixel's opacity, A_o, is the opacity of that fragment.

Keep in mind that neither of these values is "alpha." These are just values each pixel receives from its geometry. For simplicity, we'll start the discussion with a scene containing only one object, so every pixel contains at most only one fragment.

Consider Figure 2, which shows a partly transparent orange ellipse with an opacity 0.4, over a transparent background (represented by the classic white-and-gray checkerboard). We will call this image A.

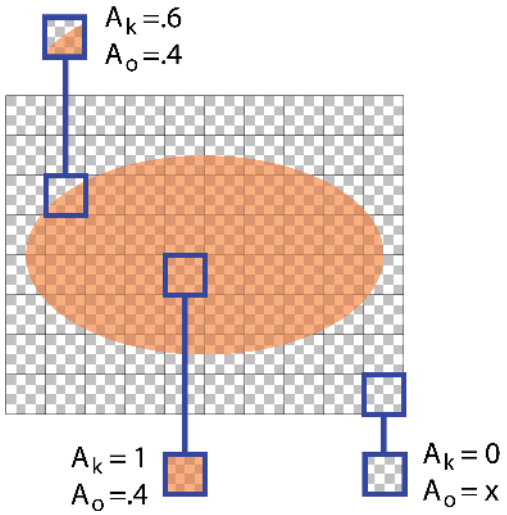

Figure 2. In image A, we draw an ellipse with opacity 0.4 over a transparent background. Every fragment of this ellipse has the same opacity A_o at every pixel. In pixels where the ellipse doesn't contribute at all, its opacity (and color) are moot.

Figure 2 calls out three pixels: one fully inside the ellipse, one outside the ellipse, and one straddling an edge. The ellipse has an opacity of .4. The pixel inside the ellipse is fully covered, so $A_k = 1$, and its fragment has the opacity of the ellipse, so $A_o = .4$. The pixel in the upper-left is about 60% covered by the ellipse, so $A_k = .6$, and the opacity of the fragment inside that pixel is again the opacity of the ellipse, so $A_o = .4$. The pixel in the lower-right that is outside the ellipse is not covered at all, so $A_k = 0$. The opacity value A_o is moot, since there's no fragment from which to derive an opacity value. In Figure 2, this A_o is given a value of x to indicate "don't know" or "don't care."

Consider now the same three pixels in Figure 3, where the ellipse is fully opaque. In each pixel, the coverage A_k is the same as in Figure 2. But the opacities of the two covered pixels are now both $A_k = 1$. That is, the partially covered pixel is only partly covered by its fragment, but that fragment is entirely opaque.

We pause here to make an observation about alpha. Modern renderers compute a value of alpha at each pixel by multiplying that pixel's A_k and A_o together (assuming, as we are for the moment, that the pixel contains only one fragment). This satisfies common sense when handling a single fragment. For example, if a pixel is one-third-covered by an opaque fragment, then $\alpha = A_k A_o = (1/3) \times 1$. Similarly, if a pixel is completely covered by a one-third opaque fragment, the result is the same: $\alpha = A_k A_o = 1 \times 1/3$. And if a pixel is one-fifth covered by a fragment that is one-third opaque, then 20% of the pixel is blocking 33% of the color from beneath it, so $\alpha = A_k A_o = .2 * .33 = .066$. This product of coverage and opacity will play an important role in the following discussion.

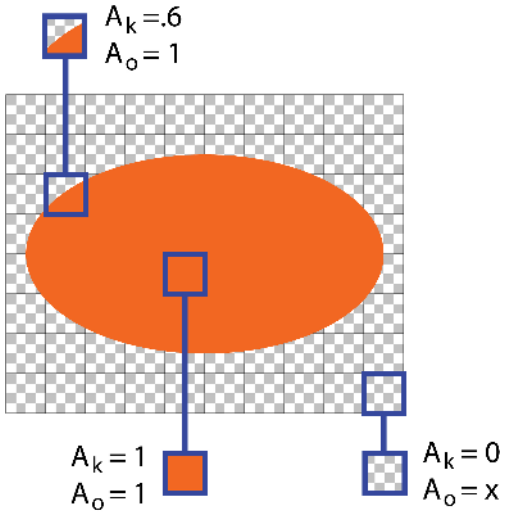

Figure 3. An opaque ellipse on a transparent background, with opacity and coverage values for three pixels.

4. Composition

We'll now look at a composition $F = A$ over B. Each pixel in the layer for object A has three pieces of information: the color A_c (or the pre-multiplied color a_c), the coverage A_k, and the opacity A_o. Of course, the layers for B hold the same data for that image.

Before we begin composing pixels, we need to decide how to combine them. The classic approach of Porter and Duff is to presume, in the absence of additional information, that the pixels are *uncorrelated* in every way. We make the same assumption here.

Using the common "little square" model of a pixel that is 1 unit on a side, Figure 4 shows a composition stack for $F = A$ over B.

Using our assumption that fragments are uncorrelated, we again follow Porter and Duff and draw the pixels as though they each contain a small shape. In this case, the shapes are axis-aligned rectangular fragments. This makes it easy to place them so that the upper fragment A overlaps both A_k percent of the area of the pixel and A_k percent of the area of B_k under it. This is the uncorrelation assumption in action: if A covers A_k of the pixel's area, then it also covers A_k of fragment B's area.

What is the final color of this pixel? There are four regions to consider:

- F_A: The region where only A is present.
- F_B: The region where only B is present.
- F_{AB}: The region where both A and B are present.
- F_0: The region where neither object is present.

Let's take these in turn.

F_A: This rectangle has area $A_k(1-B_k)$. The color inside is A_cA_o. Thus the total contribution to the final pixel color is $(F_A)_c = A_k(1-B_k)A_cA_o$.

F_B: This area is just like the above, with A and B reversed. The area is $B_k(1-A_k)$, and the color is B_cB_o. Thus the total contribution to the final pixel color is $(F_B)_c = B_k(1-A_k)B_cB_o$.

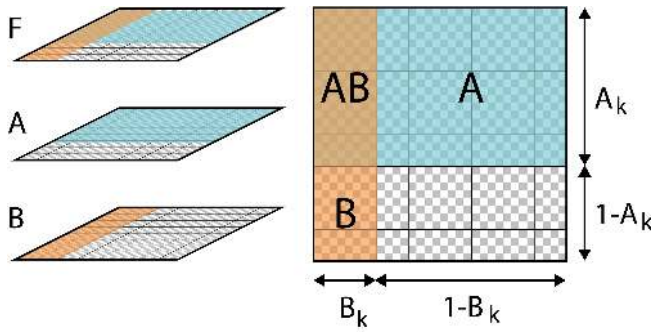

Figure 4. Pixel B is on the bottom, A is above it, and $F = A$ *over* B is at top.

F_{AB}: This region has area A_kB_k. The color contributed by A is A_cA_o and that from B is B_cB_o. All of A's color is preserved in F, but because A has an opacity of A_o, it only allows $(1-A_o)$ of the color from B to pass through it. So the total contribution to the final pixel color is $(F_{AB})_c = A_kB_k(A_cA_o + (1-A_o)B_cB_o)$.

F_0: Of course, because neither fragment contributes to F_0, the contribution to the final pixel from this area is 0.

To find the new pixel color we merely add these four terms together. We will not normalize the result (that is, we won't divide through by the areas) so the summation returns a new pre-multiplied color f_c:

$$\begin{aligned} f_c &= (F_A)_c + (F_B)_c + (F_{AB})_c + (F_0)_c \\ &= A_k(1-B_k)A_cA_o + B_k(1-A_k)B_cB_o + A_kB_k(A_cA_o + (1-A_o)B_cB_o) + 0 \\ &= A_cA_kA_o + (1-A_kA_o)B_cB_kB_o. \end{aligned}$$

This looks very familiar. We can make it even more familiar by implying multiplication of terms by multiplication of subscripts, e.g., $A_{ko} = A_kA_o$. Then we can write this as

$$f_c = A_{ko}A_c + (1-A_{ko})B_{ko}B_c. \tag{3}$$

If we assume for the moment that $A_\alpha = A_{ko}$ and $B_\alpha = B_{ko}$ (as practiced by most modern renderers), then this looks like the standard version of *over* for pre-multiplied pixels a and b, producing a pre-multiplied result f:

$$f = a + (1-A_\alpha)b.$$

But before we make that leap, notice that we've only computed a new color, and not a new opacity or coverage, both of which are necessary if we're going to perform another composition with image F (what Smith calls the *second-composition problem* [Smith 1995]). So let's find expressions for the composite coverage F_k, opacity F_o, and composite alpha, F_α.

5. Finding F_k and F_o

To find a value for F_α, we will independently track the coverage and opacity information through the *over* operation.

5.1. Coverage

The geometry term is easy: it's merely the total amount of the pixel covered by A and B. We can read that right off of Figure 4. In this calculation, it doesn't matter if

A is over B or vice-versa:

$$\begin{aligned} F_k &= A_k + (1 - A_k)B_k \\ &= B_k + (1 - B_k)A_k \\ &= A_k + B_k - A_k B_k. \end{aligned} \tag{4}$$

5.2. Opacity

Now let's look at opacity and find F_o. We can follow the same steps as before, again by referencing Figure 4. The amount of opacity added in by the region A is given by the area of that region, $A_k(1 - B_k)$ times the opacity in that region, A_o. Regions B and 0 follow the same pattern:

$$\begin{aligned} (F_A)_o &= A_k(1 - B_k)A_o, \\ (F_B)_o &= B_k(1 - A_k)B_o, \\ (F_0)_o &= 0. \end{aligned}$$

$(F_{AB})_o$ is not on the list, because it merits a closer look. Let's analyze the opacity of region AB by thinking about it in terms of transparency.

Consider two transparent sheets of plastic, A and B. Suppose $A_o = .3$, meaning it lets through 70% of the color trying to pass through it, and $B_o = .4$, so it lets through 60% of the color. We'll stack A over B and look at a swatch of color with value 1 placed beneath them, as in Figure 5.

Layer B has an opacity B_o, so it passes through $1 - B_o$ of the color beneath it. The same is true of A. For convenience, let's write these as transparency values with the subscript t, so that $A_t = 1 - A_o$ and $B_t = 1 - B_o$. Thus the combined transparency is the product of the two components, $A_t B_t$.

In other words, transparencies are commutative: looking at the world through a 20% filter in front of a 35% filter gives the same results as the other way around.

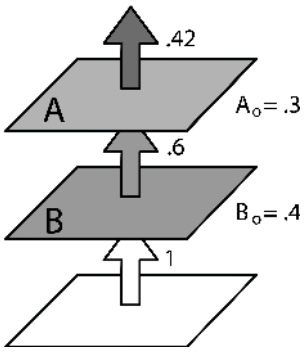

Figure 5. The bottom-most layer has color 1. It passes through layer B with opacity .4 so the color has value (1-.4)=.6, then passes through layer A with opacity .3, so it now has value (1-.3)(.6) = .42.

So the opacity in region AB is merely 1 minus the transparency, or $1-(A_tB_t) = 1-((1-A_o)(1-B_o))$. We can rearrange this to find:

$$1-((1-A_o)(1-B_o)) = A_o+(1-A_o)B_o.$$

Multiplying this by the area of region AB, we get the opacity contributed by that region:

$$(F_{AB})_o = A_kB_k(A_o+(1-A_o)B_o).$$

Putting the four pieces together, we have

$$\begin{aligned} F_o &= (F_A)_o+(F_B)_o+(F_{AB})_o+(F_0)_o \\ &= A_k(1-B_k)A_o+B_k(1-A_k)B_o+A_kB_k(A_o+(1-A_o)B_o) \\ &= A_kA_o+B_kB_o-A_kB_kA_oB_o. \end{aligned}$$

This last expression has a very nice geometric interpretation. It tells us that to find the total opacity, add the opacity contributed by A, given by A_kA_o, to the total opacity contributed by B, given by B_kB_o, but then notice that we've added the region AB twice, so subtract that region's contribution once by removing $A_kB_kA_oB_o$.

It's tempting to stop here, but notice that our final expression above is being weighted by area-based factors, and we're not normalizing. We can divide by the areas to produce a normalized result:

$$F_o = \frac{A_kA_o+B_kB_o-A_kB_kA_oB_o}{A_k+B_k-A_kB_k}.$$

We've seen that denominator before. Of course, it's just the total area of the fragments, which we found in the last section as F_k. So the final value for the composite's opacity is then

$$F_o = \frac{A_kA_o+B_kB_o-A_kB_kA_oB_o}{F_k}. \tag{5}$$

5.3. Combining Opacity and Coverage

When we compose F over some other image, at each pixel F has coverage F_k and opacity F_o. Inspired by our observation that renderers save the product of coverage and opacity when drawing fragments, let's now find the product $F_{ko} = F_kF_o$.

The terms F_k and F_o. are given by Equations (4) and (5) gathered together here:

$$\begin{aligned} F_o &= \frac{A_kA_o+B_kB_o-A_kB_kA_oB_o}{F_k}, \\ F_k &= A_k+B_k-A_kB_k. \end{aligned}$$

Notice that both of these expressions are symmetric in A and B. So the order of composition matters when we compute colors, but the resulting opacity and coverage are identical for A over B and B over A.

Let's now find the product F_{ko}:

$$\begin{aligned} F_{ko} &= F_k \frac{A_k A_o + B_k B_o - A_k B_k A_o B_o}{F_k} \\ &= A_{ko} + B_{ko} - A_{ko} B_{ko} \\ &= A_{ko} + (1 - A_{ko}) B_{ko}. \end{aligned} \tag{6}$$

This looks very familiar. Compare Equations (2) and (6):

$$\begin{aligned} F_\alpha &= A_\alpha + (1 - A_\alpha) B_\alpha, \\ F_{ko} &= A_{ko} + (1 - A_{ko}) B_{ko}. \end{aligned}$$

We know that, thanks to the renderer, $A_\alpha = A_{ko}$ and $B_\alpha = B_{ko}$. So both expressions evaluate to the same result, and thus the two left-hand terms are equal. We've reached our goal: $F_\alpha = F_{ko}$.

6. Picturing Alpha

Recall that we earlier noted that when rendering, the alpha at each pixel is the product of opacity and coverage. We've now seen that when images are composed, the resulting alpha at each pixel in that composite is the product of that pixel's final coverage and opacity.

So the quotes at the start are all correct, but they're only a few of the ways we can think of alpha. Suppose you have a pixel with an alpha value of $\alpha = 0.24$. You know that this is the product of the total coverage and total opacity, but you don't know how either of these values are distributed over the pixel.

A few pairs of values that multiply to 0.24 are shown in Figure 6, along with a few of the infinite pixels they could be describing.

So if we have an alpha of, say, 0.24, we can indeed think of it as a single opaque shape that occupies about a quarter of the pixel's area, as in the left column of Figure 6, though we don't know anything about the geometry of that shape – it could be a rectangle, a circle, or a dozen little disconnected triangles. At the other extreme, we can interpret alpha as the opacity value for a fully-covered pixel, as in the the upper-right of Figure 6. This is why pictures like Figure 4 are valid: they're just one of the many ways to draw a pixel with a given value of α.

But we can interpret α in any of an infinite number of other ways. We can treat the pixel as *any* distribution of coverage and opacity as long as the total product is equal to alpha.

Thus we're free to adopt the viewpoint in Smith [Smith 1995], that when rendering all geometry is lost and only opacity remains (right column of Figure 6). But we're just as free to think that all opacity is lost and only area remains (left column of Figure 6). Or we can choose anything else in between.

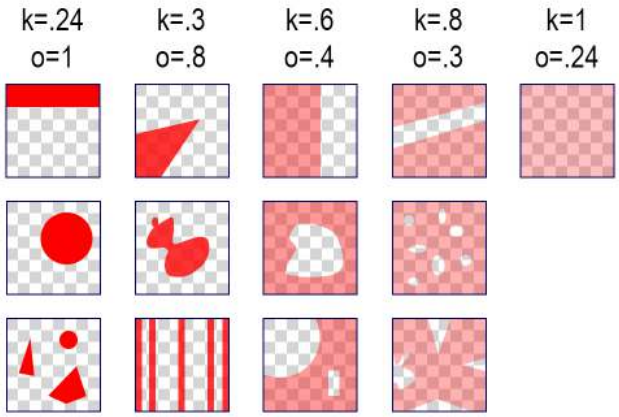

Figure 6. Each column shows a few of the infinite pixels with an alpha of 0.24 that could be associated with a particular choice of factors. The value of k is the total coverage, and the value of o is the total opacity (the images were drawn by hand and are meant to be suggestive, rather than numerically accurate).

To express this explicitly, consider a function $k(p)$ that returns the coverage of each point p in a pixel (this will be 0 or 1), and a function $o(p)$ that returns the opacity at each point p. Then we can write alpha as

$$\alpha = \int_P k(p)o(p)dp/P$$

where P represents the area of the pixel.

Here, $k(p)$ can be thought of as a "masking" function that tells us where the opacity function's values contribute, and where they don't. This formulation suggests interpreting Smith's observation that "coverage is always converted to opacity" by thinking of pre-multiplying $o(p)$ by $k(p)$. Then the coverage term disappears from the integral, and we are indeed left with nothing but opacity.

We find that it can be useful to keep these ideas distinct, as we can then think of alpha in terms of both the geometric and opacity information that go into it. We can sometimes use this information to produce a more accurate composite than would be possible with opacity or coverage alone. For example, suppose we have two pixels, and we know that when they are composited, the fragments they contain will not overlap. Then we can model the pixels with non-overlapping constant-opacity geometries (such as parallel rectangles), and use the Porter-Duff *plus* operator, rather than *over*.

Keeping coverage and opacity distinct, Figure 7 shows just a few more possible pixels that have the same alpha as in Figure 6. As far as alpha is concerned, every one of the pixels in Figures 6 and Figure 7 are equally likely to represent the "true" appearance of the pixel (one that might be produced, say, by very high supersampling, or a very accurate fragment-based renderer). Of course, we may have additional knowl-

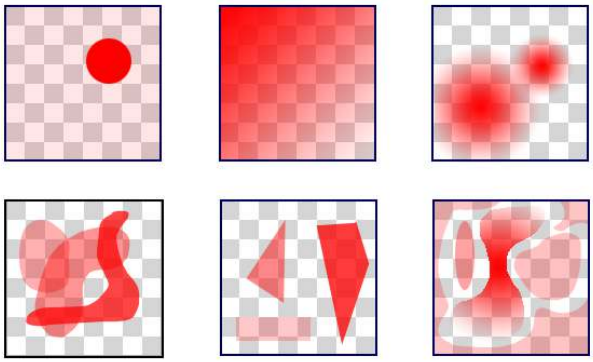

Figure 7. More pixels with an alpha of 0.24 (the images were drawn by hand and are meant to be suggestive, rather than numerically accurate).

edge about our images that could lead us to prefer some possibilities over others, but that requires us to have more information than is encoded simply in a color and an alpha value.

So the coverage and opacity distribution over a pixel can be anything from a single opaque shape to a partly transparent fractal dust, or any of countless other patterns. As long as the total coverage and opacity multiply to a pixel's alpha, we're free to distribute that opacity and coverage in any way we want.

Suppose you don't know anything about how a pixel was generated, but you wanted to choose a model of the pixel for use in other calculations. We believe that the upper-right of Figure 6 is probably the most appropriate model when no other information exists, because it makes the fewest possible assumptions about the distribution of both coverage and opacity.

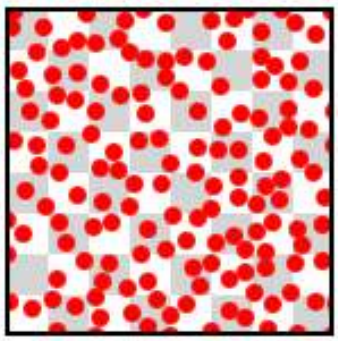

Figure 8. If an application requires a model of geometry inside the pixel, we favor a Poisson distribution of small, opaque circles whose total area equals the pixel's value of α. The pattern is simple and isotropic (the image was drawn by hand and is meant to be suggestive, rather than numerically accurate).

The models in the leftmost column of Figure 6 are less appealing because these all require inventing some kind of shape (or shapes) to represent the pixel's coverage. If such a choice is needed, we prefer a Poisson distribution of small circles (or even Gaussian blobs), which minimizes preferences for any specific orientation or structure. Figure 8 illustrates the idea. Here the opacity is 1 (that is, $A_o = 1$) and the sum of the areas is the coverage $A_k = \alpha$.

7. Conclusion

We have found that when creating $F = A$ over B, we can track each pixel's coverage F_k and opacity F_o independently. We also found that the value of F_α appropriate for compositing is given by the product of the coverage and opacity, $F_k F_o$.

We have shown that the popular idea that alpha represents either coverage or opacity, depending on convenience, is correct but incomplete. A pixel's alpha actually represents the product of the total coverage and the total opacity. In the absence of additional information, we are free to distribute those qualities throughout the pixel in any way we want.

Acknowledgements

Thank you to Steven Drucker, Eric Haines, Jeff Hultquist, Allan MacKinnon, and Andrew Willmott for helpful comments and suggestions.

References

BLINN, J. F. 1994. Compositing, part 1: Theory. *IEEE Computer Graphics & Applications 14* (September), 83–87. URL: ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=310740. 31, 33

MCGUIRE, M., AND ENDERTON, E. 2011. Colored stochastic shadow maps. In *Proceedings of the ACM Symposium on Interactive 3D Graphics and Games*. ACM, New York, February. URL: http://research.nvidia.com/publication/colored-stochastic-shadow-maps. 32

MCGUIRE, M., 2012. Translucency in OBJ/MTL files. http://casual-effects.blogspot.ca/2012/01/translucency-in-obj-mtl-files.html. Casual Effects Blog. 32

PORTER, T., AND DUFF, T. 1984. Compositing digital images. *SIGGRAPH Computer Graphics 18*, 3 (Aug.), 253–259. URL: http://dl.acm.org/citation.cfm?id=808606. 30, 31, 33

SMITH, A. R. 1995. Image compositing fundamentals. Microsoft Technical Memo 4, August. URL: http://alvyray.com/Memos/CG/Microsoft/4_comp.pdf. 31, 33, 37, 40

Author Contact Information

Andrew Glassner
The Imaginary Institute
726 North 47th St.
Seattle, WA 98103
andrew@imaginary-institute.com

Andrew Glassner, Interpreting Alpha, *Journal of Computer Graphics Techniques (JCGT)*, vol. 4, no. 2, 30–44, 2015
http://jcgt.org/published/0004/02/03/

Received:	2015-03-13		
Recommended:	2015-05-11	Corresponding Editor:	Andrew Willmott
Published:	2015-05-29	Editor-in-Chief:	Morgan McGuire