

Image Compositing

The methods we have been developing assume that we have a complete scene model and that we render this entire model from some chosen viewpoint. In practice, however, when making images, one often works with only parts of the scene at a time. For example, one might have in mind a background scene and a foreground scene. This should be a familiar concept. In live theatre and often in film production, one sets up a stage with various objects and backgrounds. You do this in advance of the actors appearing. Similarly, in computer graphics animation, it is common to render a background scene before one renders the movement of the characters. (We discussed an example in the lecture on environment mapping: the mirror creature in *Terminator 2* was rendered offline and then inserted into the film with real actors).

Separately rendering background and foreground scenes has certain practical advantages in the production stage, when you are developing and fine-tuning a scene model. For example, you can avoid having to re-render the entire background scene everytime you want change the foreground, or vice-versa. But it also raises technical challenges. How can you keep track of which points belong to the background and which belong to the foreground? How can you combine the background and the foreground layers?

The simplest approach is to have a separate binary image for the foreground – sometimes called a *mask* – which indicates which pixels are occupied by the foreground and which are not. Then, when writing the foreground over the background, you only write over the pixels that have value 1 in the mask.

One example of this idea is *blue screen matting* which has been used in cinema and television for many years. A person is shown in front of a complex background, and in fact the person is being filmed in a studio in front of a simple background, and the video of the person is superimposed over a background video/image. An example is a television weather report. The weatherman might be filmed standing in front of a blue background screen, but what is shown on the broadcast is the weatherman standing in front of a large meteorological map.

The way this is achieved is as follows: The person is filmed in front of a blue screen, and the final video is created by making a decision, for each pixel, whether to use the filmed image/video (person in front screen) or the background image (weathermap). This decision is made based on whether the RGB value at that pixel is sufficiently blue - i.e. the R and G values are much smaller than the B value. If yes, then the pixel is assumed to belong to the background and the RGB value of the pixel is replaced by that of the new background.

You may have noticed with amusement that if the person being filmed has blue eyes or is wearing a blue necktie or has blue mascara, then the new background video is mistakenly substituted at these points. You may also have noted that the edges of the person (especially around the hair) are often less than satisfactory. The person's outline sometimes has a cookie cutter look.

RGBA (rgba)

A more challenging version of the problem arises when a pixel is *partially* occupied by a foreground image, for example, a pixel on the boundary of a surface in the foreground image. Think of the pixel as a small square (rather than a point) and suppose that the projection of the surface in the foreground image covers only part of the square. For example, recall lecture 1 when we discussed how to draw a line and we had to deal with the issue of how to approximate a continuous line using

a square grid. The idea was to spread the values of a line between pixels in the (common) case that the line passed between pixels. A similar problem arises at the edge of a foreground image: we want to avoid jagged edges and to choose intensities correctly for the pixel, then we need to combine intensities from the foreground and background. That is, we need to *mix* the colors of the background and foreground which both contribute to the color of the pixel. If we don't do this, then the foreground image will have a jagged boundary, and this will not look good.

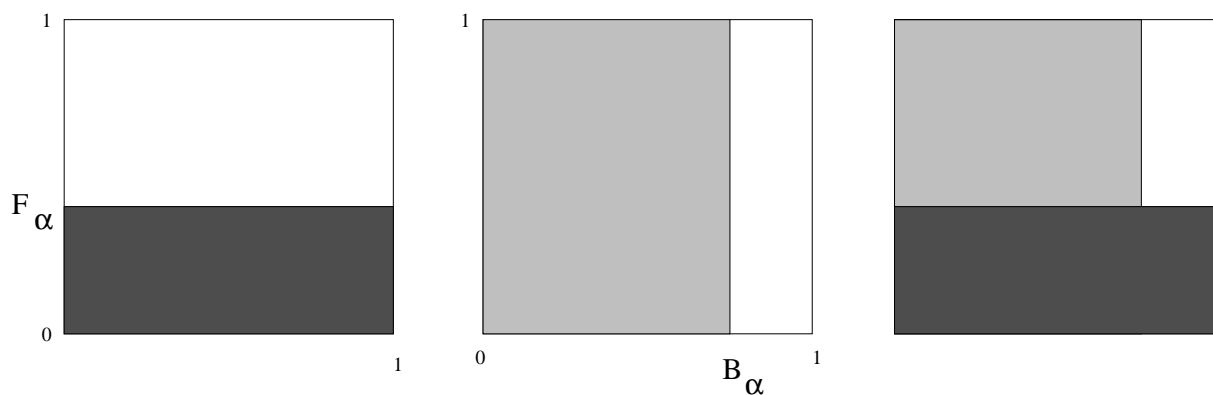
A common method for keeping track of how much of a pixel is covered by an image is to add a fourth component to each pixel, called "alpha" (α). We can interpret α as the fraction (between 0 and 1) of the pixel that is covered by this image. It is common to define RGBA to have values in $[0,1]$. For example, if we are using 8 bits for each value, then we consider the values to be in $\{0, \frac{1}{255}, \frac{2}{255}, \dots, \frac{254}{255}, 1\}$. Often, though, we will just say the values are in $\{0, 1, 2, \dots, 255\}$ to keep the notation simpler. We interpret the α values as follows: 0 is not occupied at all, 1 is completely occupied, and in between we have partial occupancy.

Suppose we have images F_{rgba} and B_{rgba} , where F is a foreground image that we wish to draw over the background image B . For generality, we allow the alpha "channels" F_α and B_α to be anywhere from 0 to 1, in particular, for the moment we do not require that each pixel in the B image is entirely occupied.

Given F_{rgba} and B_{rgba} , how do we write the foreground image over the background image? In the computer graphics problem of *image compositing*¹, this is called writing F over B . There are two steps. One is to define $(F \text{ over } B)_\alpha$, and the other is to define $(F \text{ over } B)_{rgb}$.

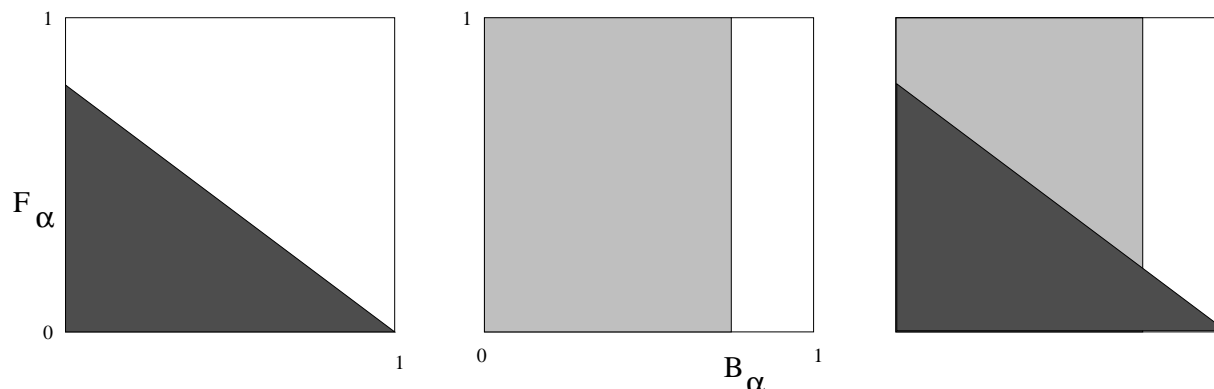
$(F \text{ over } B)_\alpha$

To define $(F \text{ over } B)_\alpha$, we blindly assume a subpixel geometry that is illustrated in the following figure. The left figure illustrates the subpixel region that is assumed to be covered by the foreground image. The middle figure shows the subpixel region that is assumed to be covered by the background image. The figure on the right shows the subpixel region that is assumed to be covered by the combined image F over B . Of course, this particular geometry is not going to correspond to what is actually present in most situations. We use this geometry *only as a way to motivate a formula for calculating* $(F \text{ over } B)_\alpha$.



¹T. Porter & T. Duff - Compositing Digital Images Computer Graphics Volume 18, Number 3 July 1984 pp 253-259

An example of how the model can go wrong is shown below. The total of the foreground is the same as before, but because the geometry of the foreground is different, the amount of background that is covered will be different from the figure above.



The first figure is the one we use. It implies a model in which the α value (coverage, opacity) of the combined image is

$$(F \text{ over } B)_\alpha = F_\alpha + (1 - F_\alpha)B_\alpha$$

where F_α and B_α both depend on the pixel (x, y) . The idea is that F covers some percentage F_α of the pixel and the background covers a percentage B_α of the remainder of the remaining fraction $1 - F_\alpha$ of the pixel.

(F over B)_{rgb}

We next turn to the *rgb* intensity values. Suppose we render part of the scene, i.e. compute the intensity of a point in the scene that projects to the pixel. If the alpha value of the pixel is less than 1, then the *rgb* intensity contributed by that pixel needs to be scaled by this α value, i.e. the surface will not contribute as much to the pixel intensity as it would if the whole pixel were covered by the surface.

You might be tempted to define:

$$(F \text{ over } B)_{rgb} = F_\alpha F_{rgb} + (1 - F_\alpha)B_\alpha B_{rgb}$$

that is, you weight the contributions of the foreground and background *rgb* images, each weighted according to the α value of the foreground image. However, notice that this is not quite correct, since it gives the final contributed *rgb* intensity of $(F \text{ over } B)_{rgb}$, rather than the *rgb* color. To see the problem, consider the simple case that $F_\alpha = B_\alpha = \frac{1}{2}$ and $F_{rgb} = B_{rgb} = 1$. In this case,

$$(F \text{ over } B)_\alpha = \frac{3}{4},$$

and

$$(F \text{ over } B)_{rgb} = \frac{3}{4}.$$

But, if the *rgb* values are supposed to be the color (not weighted by α !), then in this case they should be 1, not $\frac{3}{4}$.

The problem is that the above definition of $(F \text{ over } B)_{rgb}$ gives the rgb intensity, not the color. To get the color, you could divide by the $(F \text{ over } B)_\alpha$, which is rather awkward.

Another approach is define F_{rgb} and B_{rgb} as *already* multiplied by F_α (or B_α). For example, suppose the RGB surface color for a polygon is computed to be $(.1, .8, .6)$ and the α value for a particular pixel is computed to be $.5$. Then the (r, g, b, α) image would be given values $(.05, .4, .3, .5)$.

If we use this premultiplied definition for rgb values, then we define $(F \text{ over } B)_{rgb}$ as follows:

$$(F \text{ over } B)_{rgb} = F_{rgb} + (1 - F_\alpha) B_{rgb}$$

Note that the operations are the same as we saw earlier for computing the α value. Thus, when we used premultiplied rgb values, we have:

$$(F \text{ over } B)_{rgb\alpha} = F_{rgb\alpha} + (1 - F_\alpha) B_{rgb\alpha}.$$

Examples

Lets consider a few examples of how α is used in image compositing.

- Consider $rgb\alpha$ 4-tuples: $(0, 0, 0, 0)$ and $(0, 0, 0, 1)$. The former is a completely transparent pixel; the latter is a completely opaque black pixel.
- How do we darken an image, without changing its coverage?

$$\text{darken}(I_{rgb\alpha}, \phi) = (\phi I_r, \phi I_g, \phi I_b, I_\alpha)$$

where $\phi < 1$.

- How do we change the opacity (coverage) of a pixel, without changing the color?

$$\text{dissolve}(A, \delta) = (\delta A_r, \delta A_g, \delta A_b, \delta \alpha)$$

This amounts to sliding the underlying surface away from the pixel so that it covers the pixel less.

Alpha and OpenGL

You may have noticed that OpenGL allows you to define surface material parameters (diffuse, specular) with an alpha value i.e. RGBA. Here the alpha value is referring to surface transparency, rather than partial occupancy of a pixel. If $\alpha = 0$, then the surface is fully transparent. If $\alpha = 1$ then the surface is opaque. If $0 < \alpha < 1$, then the surface is partly transparent.

OpenGL does not use premultiplied alpha for surface materials. That is, when you declare `glMaterialfv(GL_FRONT, GL_DIFFUSE, material)` where `material` is a 4-vector, the first three components specify the reflectance color of the material (assuming $\alpha = 1$) and the fourth component specifies the opacity α .

The surface α values are used in OpenGL in a similar way to how they are used in image compositing. But there are a few subtleties. Recall that OpenGL uses a depth buffer to decide whether to draw a surface at a given pixel. But once we allow for surfaces to be partly transparent, it is no longer clear what we mean by “the depth” at a pixel, and one needs to be very careful. We will have more to say about this next lecture.

Application 1: “Pulling a matte”

We have discussed how image compositing works when you know the $rgba$ images. In computer graphics, one can compute these images using methods we have discussed. (Although I have not gone into details of how subpixel geometry is estimated and α 's are computed, you can probably imagine how this might be done.) But how could one acquire an $rgba$ image from images taken with a real camera? This is an important problem, since often one would like to insert real foreground videos of actors in front of computer graphics rendered backgrounds. Computing an α image from a real RGB image is called *pulling a matte*.² Let's take a more mathematical look at this problem.

Suppose you are given a background image or video B_{rgb} and you assume the background is fully opaque at all pixels, so $B_\alpha = 1$. For example, the background might be a blue screen, so the rgb values would be constant. You then film a real foreground scene in front of this background, for example, a person standing in front of the background. You obtain the image $(F \text{ over } B)_{rgb}$. Notice that $(F \text{ over } B)_\alpha = 1$, since we assumed earlier that $B_\alpha = 1$. You would like to composite the foreground scene over a different background, say B^* where again we assume that $B^*_\alpha = 1$. That is, you would like to compute $(F \text{ over } B^*)_{rgb}$. For example, you would like to show the weatherman standing in front of a big map.

To solve this problem, it is sufficient to compute F_{rgba} . Unfortunately this is non-trivial! The reason is that there are four unknowns to be solved at each pixel namely F_{rgba} , but there are only three measured values at each pixel, namely $(F \text{ over } B)_{rgb}$. That is, we have three equations

$$(F \text{ over } B)_{rgb} = F_{rgb} + (1 - F_\alpha)B_{rgb}$$

where the left side is measured, but we have four unknowns at each pixel namely F_{rgba} . The fourth equation

$$(F \text{ over } B)_\alpha = F_\alpha + (1 - F_\alpha)B_\alpha$$

reduces to

$$1 = F_\alpha + (1 - F_\alpha)1$$

which gives no information about F_α .

Notice that using a “pure blue screen” for the background, $B_{rgb} = (0, 0, 1)$ or using a black screen $B_{rgb} = (0, 0, 0)$ does not solve the problem. We still have four unknowns and only three equations. (Note that this problem didn't arise for the weatherman discussed at the beginning of the lecture, because we assumed that α was binary.)

One approach is to use a foreground scene that has neutral colors only, so that $F_r = F_g = F_b = F_*$, the exact value of which might vary from pixel to pixel. This gives us three equations

$$(F \text{ over } B)_{rgb} = F_* + (1 - F_\alpha)B_{rgb}$$

with only two unknowns, namely F_* and $(1 - F_\alpha)$, so we can solve for these unknowns at each pixel as well. In many movies where blue screen matting is used, neutral foreground objects are used (silver, grey).

² α channels are sometimes called *mattes*. This is not to be confused with *mask* which is binary only.

Application 2: Sports events (not on final exam)

Another application is in television broadcasting of professional sports, e.g. the *first down line* used in American football (NFL)³. The broadcasters draw a synthetic thick yellow line across the image of the football field. (This “first down line” has significance for the rules of the game, namely the team with the ball needs to bring the ball over this line.) The line appears to be “on the field” and is such that the line is hidden by any player that moves in front of it, consistent with hidden surface removal. See image below which should be viewed *in color*.



For any pixel and any frame of the video, the first down line is drawn at that pixel if two conditions are met: (1) the color in the video is sufficiently green i.e. color of grass, and (2) the pixel lies sufficiently close to a given line, namely the image projection of the first down line on the football field. Condition (1) is necessary so that the line is not drawn on top of the players. To achieve condition (2), the broadcasters need to enter by hand the position on the first down line on field. (On the real field, this position is marked by a linesman who holds an orange stick – see image above.) A homography is used to map the position of the first down line on the field into camera coordinates and then into the image plane. For more information see <http://entertainment.howstuffworks.com/first-down-line.htm>

³See www.sportvision.com, for examples.