Refraction Matting

Doug Zongker
CSE 558

June 9, 1998

1 Problem

Matting techniques have been used for years in industry to create special effects shots. This allows a sequence be filmed in studios, and have the action transferred to environments that are expensive, dangerous or impossible for the models and/or actors to work in. Still matting to create composite photographs is useful for the same reason. The creation of a matte (more properly called a *holdout matte*) depends on being able to accurately separate the foreground (which we will call the *object*) from the background of the filmed scene. The object can then be *composited* onto a new background image.

Research on this topic in the computer graphics community has been slow because of tightly held patents covering many of the basic techniques used in the industry. With the recent lapse of many of these patents, some new work on this problem has begin to emerge. The techniques we've encountered, though, all share the problem that they fail to correctly handle foreground objects which change the path of rays coming from the background image. Ordinary transparency is handled, but refraction or reflection by the foreground object is lost. This paper describes a technique for augmenting mattes with information so that these effects can be accurately recreated.

2 Related work

Although matte creation predates computer graphics, it can be described today in the language of Porter and Duff's compositing algebra [1]. The matte created is the *alpha channel* in their terms. Images with alpha have not only an RGB value at each pixel, but also a value signifying how much of the pixel is covered by that color. The matting process creates, from an image or set of images of an object and background(s), a new image containing the color and alpha of the object alone.

Smith and Blinn [2] cast the matting problem in the Porter-Duff algebra, and show it to be underspecified. They discuss techniques used in the industry (which all involve making assumptions about the colors of the foreground object so that the equations become solvable). They then introduce a novel technique, which involves shooting the object against two different backgrounds, and "triangulating" to solve for the object's color and alpha without the need for such assumptions.

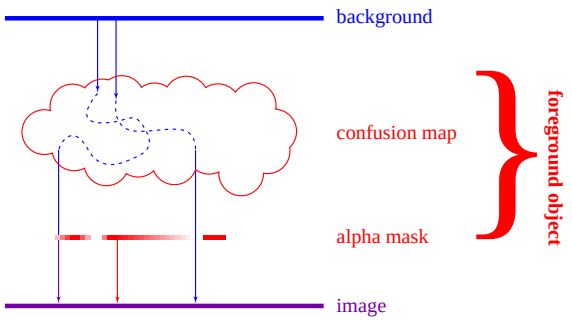


Figure 1 Conceptual model of a foreground object, showing separation into a confusion map and an alpha mask.

The work in this paper is inspired by the use of structured light to perform range scanning. One method is to sweep a plane of light across the object to be scanned, while imaging the object from another angle. By knowing the camera parameters and the position of the plane of light, the 3D position of illuminated points can be determined. A series of these profile images can be assembled into a 3D model of the surface. This requires a number of images equal to some dimension of the mesh obtained. To reduce the number of required images, the plane of light can be replaced with a number of planes. The illumination pattern changes between each image, in such a way that the pattern of light and dark for a given plane is unique to that plane. In this way, the number of images can be reduced to being logarithmic in the mesh dimension.

3 Approach

We begin by considering the foreground object to be in two parts: a *confusion map* and an *alpha mask* (Figure 1. The confusion map is a surjective mapping of pixels on the background image to pixels in the foreground image. It captures the light-ray bending properties of the object, which includes both refraction for transparent objects and reflection for mirrored surfaces. (We will ignore for the moment the complication of the object reflecting or refracting light that doesn't come from the background image into the camera.) Note that the representation of the map as being from pixels to pixels fixes the depth of the background and foreground objects relative to the camera. The second part, the alpha mask, is an ordinary RGBα image—it can mix its own color with color from the remapped background image to produce the final result.

The foreground will be extracted from an ordered series of images made of the object in front of different structured backgrounds. The first two images are made with backgrounds of two different solid colors—we apply the two-color matting technique to extract the alpha mask. The remaining backgrounds are patterns of the two colors, designed so that the sequence of colors seen on a particular background pixel is unique to that pixel. This allows us to determine which background pixel can be seen through any given foreground pixel.

Throughout this paper we will be using green and magenta as our background colors. This pair of colors has the following advantages:

1. They are different in each color channel (unlike, for instance, red and magenta). This will become important when we discuss a refinement for handling colored transparent objects.
2. They differ "maximally" in each channel. That is, for each of red, blue, and green, one has none of that primary and the other has the maximum amount. This eases the task of distinguishing the colors in the presence of noise.
3. Of the four pairs satisfying the first two criteria (green/magenta, red/cyan, blue/yellow, and black/white), they are the two that are most similar in intensity. This is convenient for taking images of real objects with film, since it obviates the need to change camera exposure and/or aperture when the background pattern changes.

3.1 Extracting the alpha mask

When the background is a solid color, the confusion map has no effect, so we can treat it is if it were absent. We apply the triangulation technique from Smith and Blinn [2] to obtain the foreground object color and alpha. Theorem 3 of that paper derives the object's alpha at a pixel as

$$\alpha_o = 1 - \frac{(R_{f_1} - R_{f_2}) + (G_{f_1} - G_{f_2}) + (B_{f_1} - B_{f_2})}{(R_{k_1} - R_{k_2}) + (G_{k_1} - G_{k_2}) + (B_{k_1} - B_{k_2})} \tag{1}$$

Here k_1 and k_2 are the background colors, and f_1 and f_2 are the colors of the object pixel seen against those backgrounds. This is not the most general formulation of triangulation given in the paper—it requires that the sums of the primaries for the two background colors differ. Smith and Blinn also give an equation that solves for alpha with *any* two different background colors, but it is more computationally expensive. For our purposes, this equation suffices.

Note that the triangulation technique begins by assuming that the color seen is computed as a weighted sum of the foreground and background colors. We are dealing with finite backgrounds, though, and we could encounter an object that makes things outside the background image visible to the camera (via reflection or refraction). This is illustrated in Figure 2. Since this light doesn't change when the background is changed, it will be perceived as originating from the foreground object. This will happen both when we are looking at an opaque part of the object (the red ray in the figure), and when the object is reflecting or refracting a ray that doesn't come from the background image (the green rays). A more accurate description of the compositing process is that it allows us not to place the object in an arbitrary scene, but that it allows us to recreate the original scene with a different backdrop B. Other objects in the scene which may be visible can't be changed through compositing.

3.1.1 Colored transparency

In this formulation of the problem, there is a single alpha value for each pixel of the foreground image. However, this is not enough information to recreate the effects of a colored transparent (or reflective) object. For instance, a blue glass sphere would pass more blue light than red or green light. To capture this, we calculate an alpha value for each channel by ignoring the other two channels:

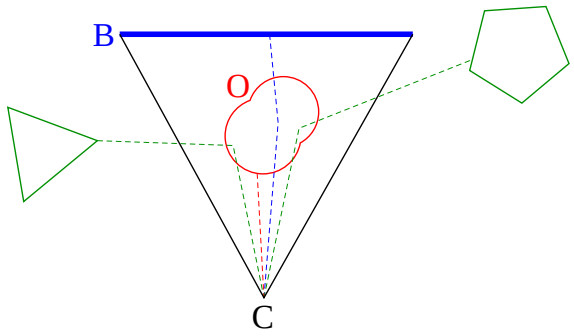


Figure 2 Foreground object O seen against background image B by camera C. Note that objects in the scene other than the background are visible via reflection and/or refraction. The color contributed by both the green and red rays will be perceived as being part of the foreground object.

$$\alpha_r = 1 - \frac{R_{f_1} - R_{f_2}}{R_{k_1} - R_{k_2}} \qquad \alpha_g = 1 - \frac{G_{f_1} - G_{f_2}}{G_{k_1} - G_{k_2}} \qquad \alpha_b = 1 - \frac{B_{f_1} - B_{f_2}}{B_{k_1} - B_{k_2}}$$

These are the same as equation (1), but setting all the color components not under consideration equal to zero. This leads to our first requirement for the background colors (listed above): that they differ in every channel. If the colors were the same in some channel then we could not solve for the alpha of just that channel.

3.2 Extracting the confusion map

The confusion map specifies, for each pixel in the foreground image, which background pixel is seen through that pixel. We assume that the object is stationary relative to the background, so that the map is the same in every image. We also assume that the map is the same for each color channel. (It would be straightforward to extend this technique to computing a per-channel map, but that would triple the already significant number of images required.)

Our strategy to obtain this map will be to assign to each pixel in the background a unique binary string, and then use the bits of each pixel's string to determine its color over a sequence of images. By observing the pattern of color changes of an object pixel we can obtain a binary string, and thus determine which background pixel was seen.

The first technique used will use a series of stripes to encode the row and column of the background pixel as a binary number. Suppose that the background image is $2^w \times 2^h$ pixels. We use $w + h$ images of stripes to encode the location of each pixel. Images with $2, 4, 8, \ldots, 2^w$ vertical stripes encode the column, and images of $2, 4, 8, \ldots, 2^h$ horizontal stripes encode the row. We can think of a pixel's overall string as being the concatenation of its row address with its column address. The two colors magenta and green correspond to the bits 0 and 1, the bit position being determined by which set of stripes is used. The 2-stripe image determines the most significant bit of the row or column location, the 2^w-stripe (or 2^h-stripe) image determining the least significant bit. This encoding is illustrated for a 16×16 image in Figure 3.

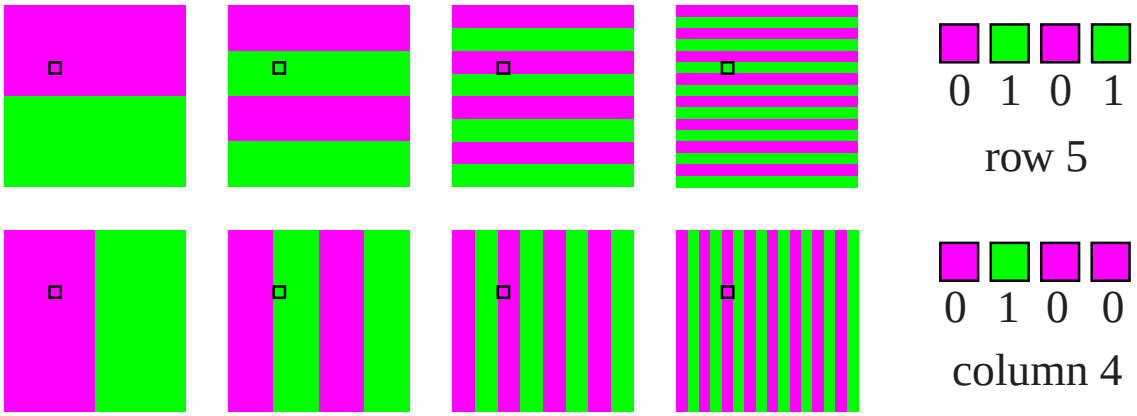


Figure 3 Illustration of pixel color coding for location. Magenta represents a 0 bit, green a 1 bit.

In each of the $w+h$ striped images, we must determine whether we are seeing a magenta or a green background pixel through each foreground object pixel. To make this decision, we look at how the foreground object pixel appeared when we knew the background pixel was magenta (the object with the solid magenta background image) and how it appeared when we knew the background pixel was green (the object against solid green). We calculate the L_2 distance between the color of the pixel against an unknown background pixel and each of these two known values, and decide that the background color is the one corresponding to the smaller distance.

We can examine the consequences of this technique in the special case of an opaque foreground pixel. Recall that this can happen both when we are really seeing a part of the foreground object, as well as when the object is reflecting or refracting a ray from outside the backdrop into the camera. In this case the two L_2 distances are equal, and the implementation arbitrarily chooses magenta. This will happen for each stripe background, so the confusion map will map background pixel $(0, 0)$ to the foreground pixel. However, since no background at all will show through an opaque foreground pixel, any entry in the confusion map would suffice.

3.2.1 Improving the encoding

One disadvantage of the binary stripe technique is that it is relatively sensitive to registration error. Consider the two center columns of a 256×256 image. The codes for these two columns differ in every bit: the left column code is 01111111, the right column code is 1000000. Slight misregistrations which cause the wrong column to be read in some of the stripe images could lead to taking some of the bits from the left column, and some from the right column. This can produce an arbitrarily large error, since any bit string from 00000000 to 11111111 could be extracted.

It would improve matters if the codes for adjacent columns (and rows) were more similar, so that a misregistration error would be less likely to cause a bit flip in the extracted code. Fortunately, such codes exist. A *Gray code* is an ordering of the 2^n n-bit strings such that each adjacent pair differs in only one bit position. An example Gray code is given in Figure 4.

We can use a Gray code to reduce the likelihood of error, with no cost in acquiring additional

index	binary code	Gray code
0	000	000
1	001	001
2	010	011
3	011	010
4	100	110
5	101	111
6	110	101
7	111	100

Figure 4 A 3-bit Gray code. Note that adjacent rows of the Gray code differ in only one position, unlike the binary code column.

images. If we are willing to use additional images, however, we can potentially reduce error even more by encoding the position redundantly using an error-correcting code. A Hamming code, for instance, allows correction of any single-bit error in 16 bits of data (enough to record position in a 256×256 image) by sending an additional 5 check bits. Section 5 gives some results created using Gray and/or Hamming codes.

4 Considerations for photography

Applying this process to rendered images is interesting, but somewhat academic since we could create the confusion map and alpha mask directly from the 3D geometry necessary for rendering, without having to repeatedly render the scene against different backgrounds. To make this technique useful, we must be able to take pictures of real objects and perform the matting accurately enough to make convincing composites.

To capture real data, we use a tripod-mounted camera to take pictures of the object in front of a computer monitor displaying an appropriate backdrop. We can thus ensure that the relative positions of camera, object, and background don't change from frame to frame. We place the object close to the monitor, and use a small aperture in order to get both the object and the background in focus at the same time. The film is developed and scanned to produce a PhotoCD with 3072×2048 images.

A number of effects introduce distortion and/or misregistration into the images obtained: the background monitor is vertically flat but slightly curved horizontally, the film plane is not exactly parallel to the monitor, and the misalignment within the scanning process introduces some translation of the camera center within the image plane. As a result, the region of interest (the background square on the monitor with the object in front of it) is nonrectangular in the captured image, and we must do some warping to obtain a clean image on which to run the above extraction algorithms.

Since any warping will necessarily distort the appearance of the foreground object, we try to minimize distortion by carefully aligning and aiming the camera. Whatever distortion remains is countered with a simple quadrilateral warp. The backdrop contains four registration markers in a rectangle around the colored background image. We know the positions of these markers on the

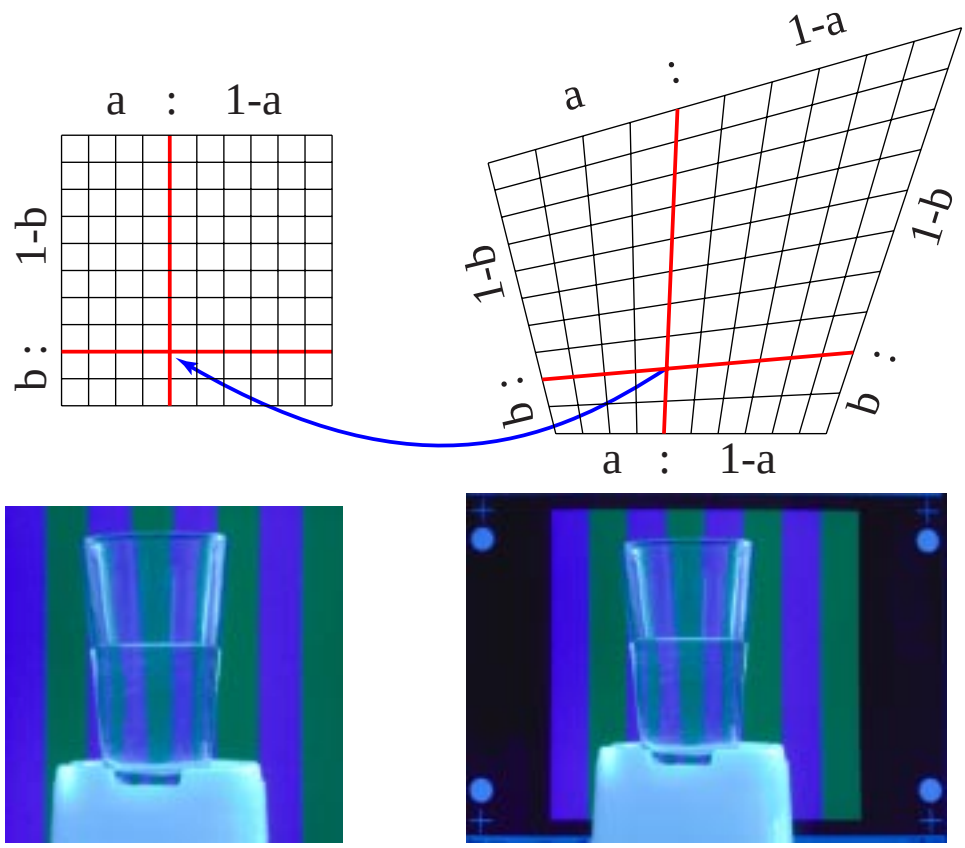
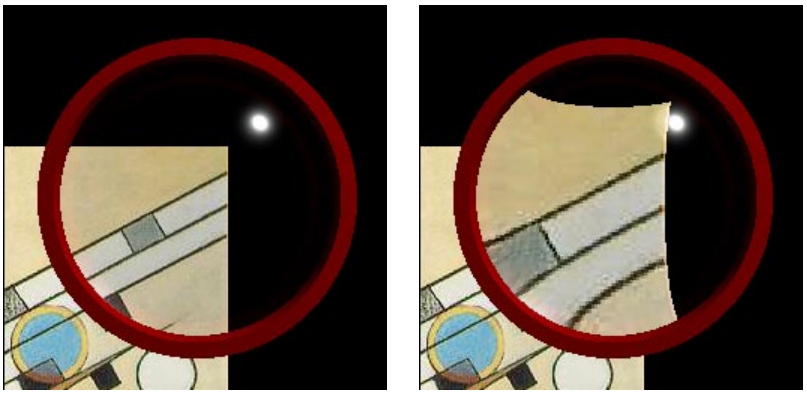


Figure 5 Illustration of the resampling process.

screen, and their position relative to the background image. We can locate these four markers with subpixel accuracy in the photograph, and use them as the basis for a quadrilateral warp. For each pixel in the 1024×1024 background image on the screen, we determine where it maps to in the photograph, and select the pixel covering that point. In this way we build a new 1024×1024 image containing just the background square and the object in front of it. This process is illustrated in Figure 5.

5 Results

The compositing operation is fast enough to render a 256×256 object at approximately 25 frames per second. About half of each frame's time is devoted to evaluating the compositing equation, with the other half being spent in moving bits to the frame buffer. The compositing application allows the foreground object to be dragged around over a background image. The speed of rendering and the quality of the resulting images produce a very convincing effect of moving a transparent physical object over a background image.



(a) (b)

Figure 6 Compositing a rendered magnifying glass: (a) without refraction, (b) with refraction.

Figure 6 shows the difference in realism that recreating refraction can make. The foreground object is a convex lens mounted in an opaque red frame.

Figure 7 shows some other rendered objects composited on top of various backgrounds. All the objects were rendered without antialiasing, so exactly one background pixel is seen through each foreground pixel. The confusion map extraction produces a perfectly accurate result for these objects. Parts (a) and (b) show the rendered magnifier with new backgrounds. Parts (c) and (d) show how the confusion map can be used to capture both reflection and refraction. Parts (d)–(f) show the use of per-channel alpha to perform filtering of color.

Figure 8 shows results obtained using some alternate coding schemes. In each case, the object was obtained by rendering using adaptive subsampling. This produces images more like those obtained photographically—they have some blurring of fine details, which makes the confusion map extraction less clean. Note the errors at power-of-2 divisions of the image using binary coding (Figure 8(a)), and the improvement produced by using Gray coding (Figure 8(c)). Using Hamming check bits significantly degrades image quality using both binary and Gray coding. The likely explanation for this extremely poor "correction" is that the Hamming check bits, which are parity checks for different overlapping subsets of the data bits, correspond to images which have a lot of high frequency energy (see Figure 9). These fine details are the things preserved least well by the photography (or antialiased rendering), so the error rates of transmitting the check bits is significantly higher than in the data bits. We could employ a different error-correcting code that generates lower-frequency images.

Figure 10 shows an object (a glass half full of water) extracted from photographs and composited onto a new background. This shows the general effectiveness of the technique on real data. The photographs were resampled as described above to create 1024×1024 images, which were then averaged down to the 256×256 images that the matting process was run on. Note that the water correctly reverses the background image left-to-right. There are some problems with this image, though:

1. The object, which was clear glass sitting on a white plastic base, is tinged blue. This

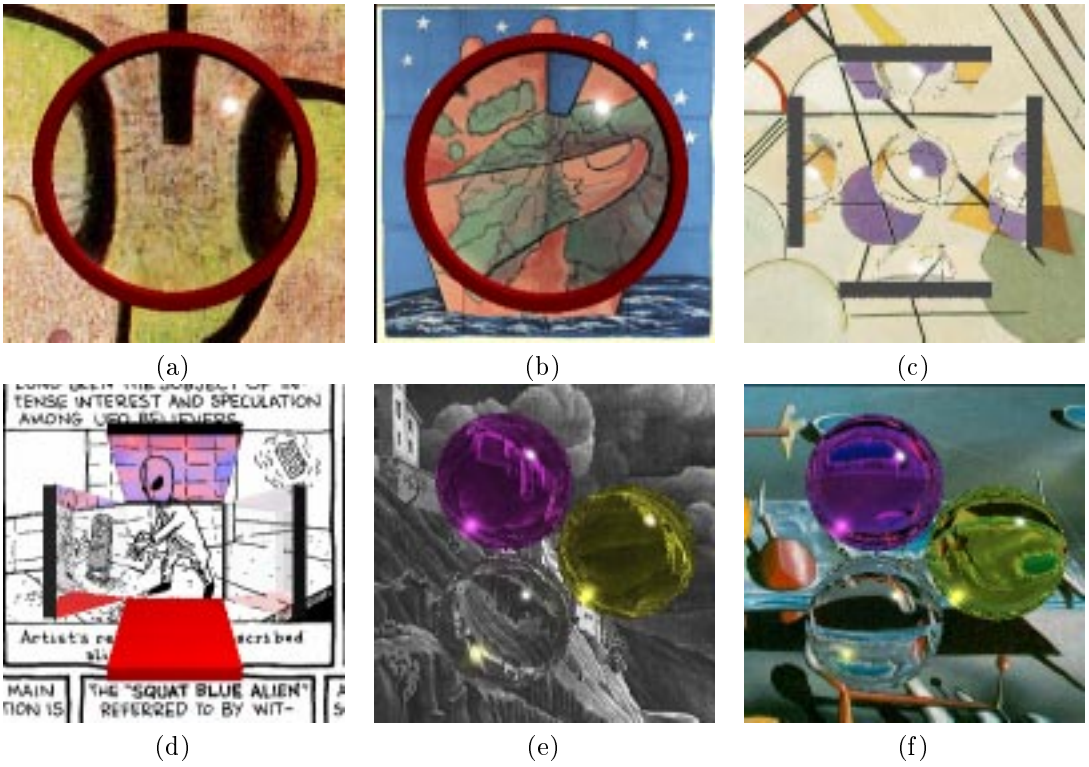


Figure 7 Composites of rendered objects: (a) & (b) a magnifying glass, (c) four mirrors surrounding a glass sphere, (d) opaque, transparent, reflective, and colored transparent boxes, and (e) & (f) colored transparent spheres.

Figure 8 Results of employing various location coding schemes with antialiased rendered images. (a) binary coding, (b) binary coding with Hamming correction, (c) Gray coding, (d) Gray coding with Hamming correction.

Figure 9 Images of Hamming check bits for Gray-coded locations.

is probably a result of using tungsten film, which is more sensitive to blue light, under fluorescent lighting. We expect that a new set of pictures using more appropriate film would correct this.

2. The area around the glass, which should be perfectly clear, appears dirty. The alpha is not extracted correctly in these areas due to speckle noise in the background, which the alpha equation assumes is a solid known color. Other authors have reported more accurate alpha computation by photographing the solid color backgrounds with no object and using those for comparison [3]. We could also envision a tool that lets objects be touched up by hand, so that the erroneous alpha could be painted out of the object.

3. There are noticeable errors in the confusion map—note the checkerboarding effect on the diagonal lines visible through the lower right corner, and the horizontal streaks visible throughout the area around the glass. This set of images uses binary coding of the background pixels—images taken in the future should at least use the Gray code described above, or possibly more advanced error-correction schemes.

6 Future work

While this technique produces good results, the number of images required is a major drawback. It currently requires $\lceil \log_2 N \rceil + 2$ images, where N is the number of pixels in the background image. Some of the enhancements discussed above (error-correcting bits, backgrounds without foreground object) would require even more images. This large number of images probably precludes applying the technique to video or film sequences. With just two backgrounds needed, one could imagine filming a sequence with a backdrop alternating between the two images, and using a method like optical flow to interpolating the missing images. Extracting a 256×256 object with this technique in this work, however, requires 18 images—interpolating across the 17 frames between successive appearances of a given background would probably be hopelessly inaccurate. Motion picture applications of this technique would therefore be limited to highly controlled, repeatable shots, where the camera and object motion can be exactly replicated many times.

One way to reduce the number of images needed would be to transmit more bits per pixel per image. We can imagine putting one stripe pattern in the red channel, another in the green channel, and a third in the blue channel, giving a total of eight colors present in the background image. While this is less robust than the current method, which needs only to distinguish between

Figure 10 A glass of water extracted from photographs composited over a painting.

two background colors, it cuts the number of images needed by two-thirds. This could be reduced further by using more levels of each channel to encode multiple bits (four levels of red, for instance, rather than two). In the limit, this technique approaches the gradient technique of Wolfman and Werner [4].

Another approach would be to abandon the coding scheme entirely and apply more computer vision techniques. The current method can (in theory) extract entirely arbitrary confusion maps, but most objects have a confusion map with a lot of coherence that the current technique does not take advantage of. We could attempt to estimate the confusion map by matching features in the background with points in the composite image, and taking the confusion map as some sort of smooth warp between those points.

References

[1] Thomas Porter and Tom Duff. Compositing digital images. In *Computer Graphics (SIGGRAPH '84 Proceedings)*, pages 253–259, July 1984.

[2] Alvy Ray Smith and James F. Blinn. Blue screen matting. In *SIGGRAPH 96 Conference Proceedings*, pages 259–268, August 1996.

[3] Steve Wolfman and Dawn Werner. Personal communication, 1998.

[4] Steve Wolfman and Dawn Werner. Low-cost extensions to the blue screen matting problem, 1998.