

CSC320 Assignment 1 Report

Efficiency Analysis

Note that the matting equation can be represented using the following matrix equation:

$$Ax = b$$

Where:

Variable	Value
A	$\begin{bmatrix} 1 & 0 & 0 & -R_{k1} \\ 0 & 1 & 0 & -G_{k1} \\ 0 & 0 & 1 & -B_{k1} \\ 1 & 0 & 0 & -R_{k2} \\ 0 & 1 & 0 & -G_{k2} \\ 0 & 0 & 1 & -B_{k2} \end{bmatrix}$
x	$\begin{bmatrix} R_n \\ G_n \\ B_n \\ \alpha \end{bmatrix}$
b	$\begin{bmatrix} R_{f1} - R_{k1} \\ G_{f1} - G_{k1} \\ B_{f1} - B_{k1} \\ R_{f2} - R_{k2} \\ G_{f2} - G_{k2} \\ B_{f2} - B_{k2} \end{bmatrix}$
C_x	Some color channel C (R, G or B) in image x
$k1, k2$	The first and second background / backdrop images
$f1, f2$	The composite images taken on top of backdrops $k1, k2$ respectively
n	The NEW composite image, with no background (i.e. just the target foreground object with a black background)
α	The transparency value associated with the given pixel (higher value means more opaque)

Recall the matting equation:

$$C_n = C_f - C_k(\alpha - 1)$$

Where:

Variable	Value
n	The NEW composite image, with no background (i.e. just the target foreground object with a black background)
k	The background / backdrop image
f	The foreground image (target object in front of background k)
α	The transparency value associated with the given pixel (higher value means more opaque)

Since we know the values of C_f and C_k , we only need to determine α to solve this equation.

We note that alpha is contained within the vector x . We can find a good approximation of x (and by extension α) by using the pseudo-inverse of A , denoted by A^+ :

$$\begin{aligned} Ax &= b \\ \Rightarrow x &\approx A^+b \end{aligned}$$

Where:

$$A^+ = (A^T A)^{-1} A^T$$

If we wanted to find the alpha value for every pixel in an image, we could just calculate A^+ for every pixel, and take the last element in the resulting vector (which would be an approximation of alpha). However, this would be highly inefficient. The time required to invert a $n \times n$ matrix with the most optimal algorithms today is $O(n^{2.373})$. Matrix multiplication isn't much better, with runtimes of $O(nmp)$ for multiplying a $n \times m$ matrix with a $m \times p$ matrix (Dasgupta, Papadimitriou and Vazirani 2006).

Since we are finding the pseudo-inverse of matrices that obey the same format as A , we can simply solve for x (and by extension α) by expanding matrix operations. However, A is a 6×4 matrix, which means that finding A^+ (and thus x and α) is no small feat **by hand** (just writing down A requires us to write **24** values!). However, using **sympy**, a Python symbolic math library, (in conjunction with some human guidance for factoring) to manipulate our matrices, we show*:

Variable	Value
A^T	$\begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ -R_{k1} & -G_{k1} & -B_{k1} & -R_{k2} & -G_{k2} & -B_{k2} \end{bmatrix}$
$(A^T A)_{4 \times 4}$	$\begin{bmatrix} 2 & 0 & 0 & -R_{k1} - R_{k2} \\ 0 & 2 & 0 & -G_{k1} - G_{k2} \\ 0 & 0 & 2 & -B_{k1} - B_{k2} \\ -R_{k1} - R_{k2} & -G_{k1} - G_{k2} & -B_{k1} - B_{k2} & B_{k1}^2 + B_{k2}^2 + G_{k1}^2 + G_{k2}^2 + R_{k1}^2 + R_{k2}^2 \end{bmatrix}$
$\det(A^T A)$	$4 (B_{k1}^2 - 2B_{k1}B_{k2} + B_{k2}^2 + G_{k1}^2 - 2G_{k1}G_{k2} + G_{k2}^2 + R_{k1}^2 - 2R_{k1}R_{k2} + R_{k2}^2)$ $= 4((B_{k1} - B_{k2})^2 + (G_{k1} - G_{k2})^2 + (R_{k1} - R_{k2})^2)$
$A^+ = (A^T A)^{-1} A^T$	Note: the resulting matrices are too large to display on a single page.
$x \approx A^+ b$	
α	$1 - \frac{(B_{f1} - B_{f2})(B_{k1} - B_{k2}) + (G_{f1} - G_{f2})(G_{k1} - G_{k2}) + (R_{f1} - R_{f2})(R_{k1} - R_{k2})}{(B_{k1} - B_{k2})^2 + (G_{k1} - G_{k2})^2 + (R_{k1} - R_{k2})^2}$

Note $\det(A^T A)$ – if this is zero, then A^+ (and anything else that depends on it, by extension) cannot be calculated as $(A^T A)$ is not invertible. The only time this happens is when

$(B_{k1} = B_{k2}) \wedge (G_{k1} = G_{k2}) \wedge (R_{k1} = R_{k2})$ (i.e. the background colours are identical). In this case, numpy's matrix pseudo-inversion will output a vector of zeros, so I will just set the complement (the large fraction) to 0 to emulate this.

The bottom-most result is what we are looking for: a numeric representation of α that does not require any matrix operations! We can now use this equation as a guide to optimize our numpy code.

*The **sympy** code that I wrote to determine this is in the file *MattingDerivation.py*.

We make the following observations:

1. *Univariate Function Broadcasting*: For a one-variable function $f(x)$ and numpy array a_1 , $f(a_1)$ will return a numpy array a_2 with the same dimensions as a_1 , with every element r_2 in a_2 being such that $r_2 = f(r_1)$, where r_1, r_2 share the same positions in a_1, a_2 . Example:

```
>>> def f(x, y): return x ** y

>>> a1 = np.array([1,2,3])
>>> a2 = np.array([1,2,3])
>>> f(a1, a2)
array([ 1,  4, 27], dtype=int32)
```
2. *Bivariate Function Broadcasting*: For a bivariate function $f(x, y)$ and numpy arrays a_1, a_2 (with the same dimensions) $f(a_1, a_2)$ will return an array a_3 with the same dimensions as a_1, a_2 , but every element r_3 of a_3 will be such that $r_3 = f(r_1, r_2)$, where r_1, r_2 have the same position in a_1, a_2 that r_3 has in a_3 . Example:

```
>>> def g(x): return 2 * x

>>> a1 = np.array([1, 3, 5])
>>> g(a1)
array([ 2,  6, 10])
```

3. The equation for α can be represented as a division of dot products:

$$\alpha = 1 - \frac{k_{\Delta} \cdot f_{\Delta}}{k_{\Delta} \cdot k_{\Delta}}$$

Where:

$$k_{\Delta} = \begin{pmatrix} R_{k1} - R_{k2} \\ G_{k1} - G_{k2} \\ B_{k1} - B_{k2} \end{pmatrix}, f_{\Delta} = \begin{pmatrix} R_{f1} - R_{f2} \\ G_{f1} - G_{f2} \\ B_{f1} - B_{f2} \end{pmatrix}$$

4. RGB images in *opencv/numpy* are represented as 2D-arrays, where each element is a vector of length 3

Since f_1, f_2, k_1, k_2 will be represented in the code as numpy arrays with **identical dimensions**, we can calculate f_Δ and k_Δ across the entire image by broadcasting subtraction (a bi-variate function) across the respective images:

$$f_\Delta = f_1 - f_2$$

$$k_\Delta = k_1 - k_2$$

We can broadcast multiplication across f_Δ and k_Δ to get:

$$numeratorVector = f_\Delta * k_\Delta = \begin{pmatrix} (R_{f1} - R_{f2}) * (R_{k1} - R_{k2}) \\ (G_{f1} - G_{f2}) * (G_{k1} - G_{k2}) \\ (B_{f1} - B_{f2}) * (B_{k1} - B_{k2}) \end{pmatrix}$$

$$denominatorVector = f_\Delta * k_\Delta = \begin{pmatrix} (R_{k1} - R_{k2})^2 \\ (G_{k1} - G_{k2})^2 \\ (B_{k1} - B_{k2})^2 \end{pmatrix}$$

We can then broadcast the following function (x is a 3D vector) across the above:

$$\lambda(x) \rightarrow x \cdot \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

To get:

$$\begin{aligned} numerator &= \lambda(numeratorVector) = \begin{pmatrix} (R_{f1} - R_{f2}) * (R_{k1} - R_{k2}) \\ (G_{f1} - G_{f2}) * (G_{k1} - G_{k2}) \\ (B_{f1} - B_{f2}) * (B_{k1} - B_{k2}) \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \\ &= (R_{f1} - R_{f2}) * (R_{k1} - R_{k2}) + (G_{f1} - G_{f2}) * (G_{k1} - G_{k2}) + (B_{f1} - B_{f2}) * (B_{k1} - B_{k2}) \end{aligned}$$

$$\begin{aligned} denominator &= \lambda(denominatorVector) = \begin{pmatrix} (R_{k1} - R_{k2})^2 \\ (G_{k1} - G_{k2})^2 \\ (B_{k1} - B_{k2})^2 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \\ &= (R_{k1} - R_{k2})^2 + (G_{k1} - G_{k2})^2 + (B_{k1} - B_{k2})^2 \end{aligned}$$

Finally, we can obtain α for the **entire** image using the above:

$$f_{\Delta} = f_1 - f_2$$


$$k_{\Delta} = k_1 - k_2$$

$$\alpha = 1 - \frac{(f_{\Delta} * k_{\Delta}) \cdot \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}}{(f_{\Delta} * f_{\Delta}) \cdot \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}}$$

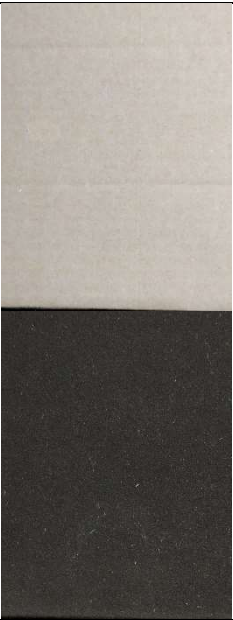


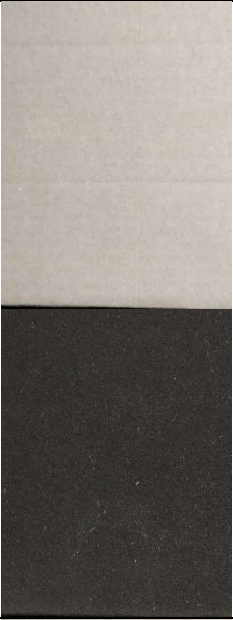


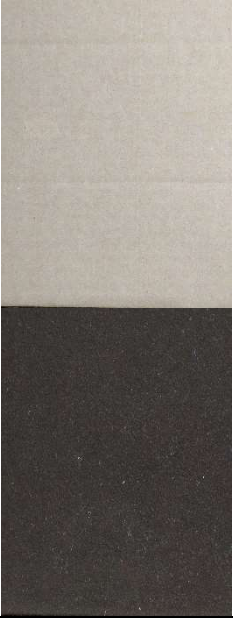


Note that the implementation of the above can be accomplished **without** using Python loops, and can be done instead using pure numpy semantics. This will increase the efficiency and speed of the code by circumventing inefficient matrix multiplications and slow Python for-loops. The running time of my current implementation is faster than the reference solution by about **1000x** on CDF systems (reference took ~12 seconds to process, mine took ~12ms to process the image).

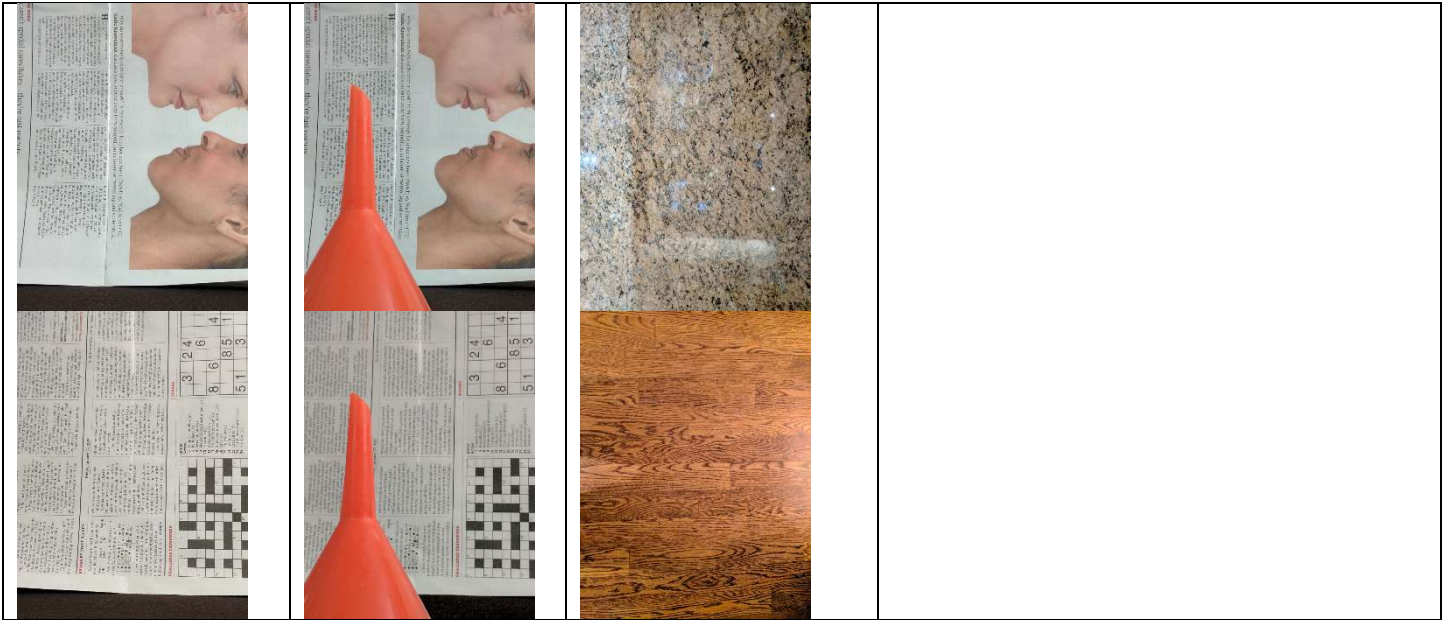
Experimental Evaluation / Report

I was unable to capture very many valid image sets using my phone due to a lack of good stabilizing equipment (e.g. a tripod, I had to use a wood vice). I did however pull several image-sets off the internet in order to test my project (as supplements to the supplied images). The following is a table of the images I used to test my solution, along with their sources. Note: I considered every pair of composites to correlate to an image set – with 11 composite pairs, I have 11 image sets.

Backings	Composites	New Backing	Source
			http://graphics.cs.cmu.edu/courses/15-463/2004_fall/www/handins/mbergou/asst4/
			http://cs.brown.edu/courses/csci1290/2011/results/final/aabouche/
			
			http://cs.brown.edu/courses/csci1290/2011/results/final/aabouche/

			<p>http://cs.brown.edu/courses/csci1290/2011/results/final/aabouche/</p>
			<p>Included with this assignment</p>

				<p>My own images – taken with a Google Pixel on default settings (ISO 3572, 1/15sec exposure, f/2 – for all images taken), locked in place with a wood vice</p>
				
				



These are the following backing properties that I wanted to test triangulation matting on, with some reasoning behind them:



Backing Property	Reason	Applicable Image Sets
Similar colors / shades of the same color	<p>As discussed in the written question section, pixels with the same color but different shades will have the same RGB ratios, but different brightness ratios. The difference between the background color channels should be non-zero in this case (assuming the brightness ratio a is not the same), because $a \neq 1, x - ax \neq 0$. We are also ensured to have the same sign on every background-difference color channel if we know that the RGB ratios are the same.</p> <p>This ensures that we don't run into following edge cases:</p> <p>The denominator of the alpha equation is zero because the background difference color channels are all zero</p> <p>One or more foreground color channels are effectively ignored because the background difference in that color channel is close to or actually 0.</p> <p>The numerator of the alpha equation adds up to 0 through cancellation (e.g. $180 - 90 - 90 = 0$).</p> <p>I expect this type of image-set to be the most ideal for calculating alpha values. As such, I had a lot of these image sets to act as sanity checks for my program.</p>	<p>Comb</p> <p>Wine glass</p> <p>Flower vase</p> <p>Soaps (on green and white)</p> <p>Pumpkin (on green and white)</p>
Dissimilar colors	<p>As discussed previously, if the background difference is able to yield 0 or numbers of different signs, the alpha equation could be rendered ineffective.</p>	<p>Plastic bottles and salt shaker (on blue and purple)</p> <p>Plastic bag and shampoo bottle (on blue and purple)</p>

		Beer Glass, Gnome and Martini Glass (on burgundy and green)
Images / Text / Patterned	I wanted to see how the method coped with objects being introduced into the scene in the form of the background. I expect the objects in the background to show up in the alpha, even when they aren't supposed to.	Funnel with newspaper backing
Noisy	I wanted to see how triangulation matting coped with noise. I expect high levels of noise in the backgrounds to carry over to the composite images.	Funnel with cardboard/felt backing

The following are object properties that I wanted to test triangulation matting on.

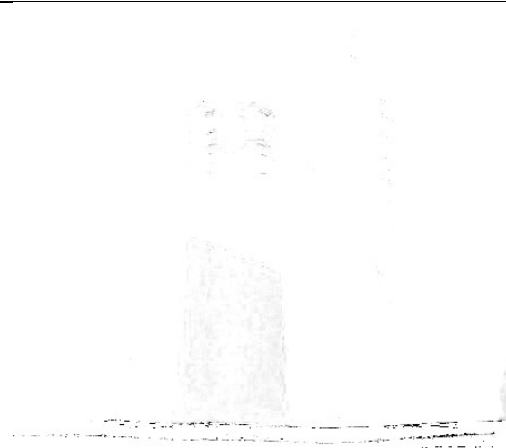
Object Property	Reason	Applicable Image Sets
Transparency	Ideally, triangulation matting should be able to maintain transparency levels in the composite image. The method may fail if an object is too transparent, as it may not even be able to be picked up properly.	All image sets except for the cone and funnel
Reflectiveness	I think that reflections in the object (of the background) may cause some issues with calculating the true alpha of the object. Example: if an object is partially reflecting another object in both composite images, that area may be considered to be more opaque than it actually is.	Wine glass
High-density gaps	I wanted to see how the method coped with objects with high-density gaps – ideally triangulation matting should be able to clearly define the edges of the object, having a lot of edges will put this to the test.	Comb

Experimental Results – Produced from My Solution

Image Set(s) (composite(s), new background(s))	Property(s) being Tested	Results (Alpha, ColOut, 1 Composite)	Comments
	Transparency, Dissimilar backing colors		I expected this to work out poorly as the two backing colors were substantially different (red and green). However the method seems to accurately capture the transparency of the target objects in this case. I think in this case, the method was lucky enough not to hit the edge cases that I described previously.



Transparency,
dissimilar backing
colors



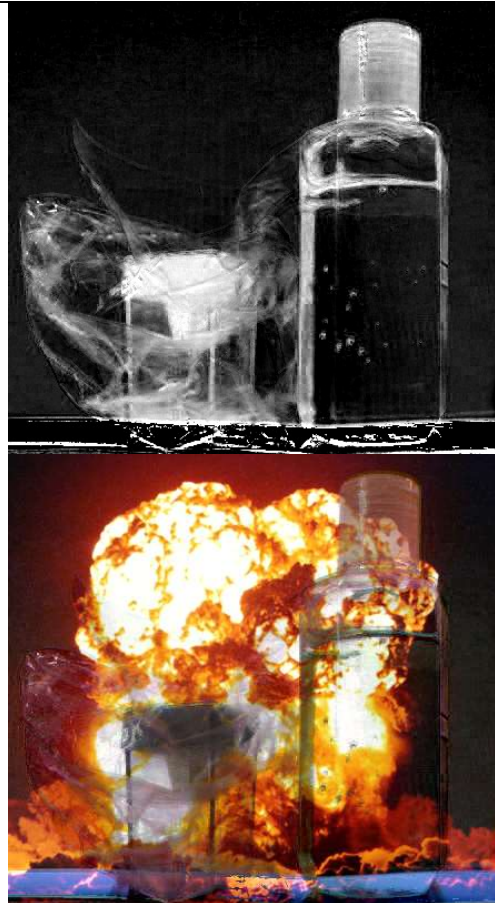
Note: The alpha is so high in the
backdrop that I can't tell the difference
between the colOut looks the same as
the new composite



I expected this to work out poorly due to
the reasons I discussed in the prior section
about dissimilar backing colors. It seems
like the technique calculated way too high
of an alpha value over the whole image. I
think that in this case, the numerator of the
alpha complement is being calculated to
be 0 or near 0 (due to cancellation), which
means that the resulting alpha is near 1.



Transparency,
dissimilar backing
colors



I expected this to work out poorly due to the reasons I discussed in the prior section about dissimilar backing colors. Unlike the prior example, the alpha values are a bit low – the transparency of the targets seems to be higher than we want in front of this image. I suspect that the numerator or denominator of the alpha complement are higher/lower than what we need them to be for proper transparency.



Transparency, similar backing colors (white vs. green - white is just an infinitely bright green)

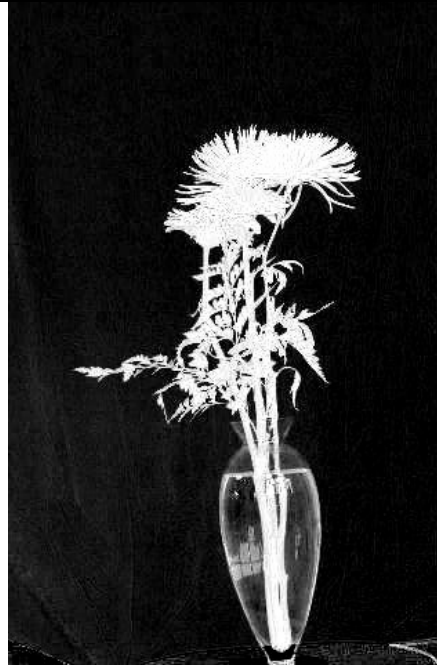


I expected this type of backing to work out well as it would guarantee that the numerator and denominator of the alpha equation would be non-zero. The produced composites and alphas seem to accomplish both goals of triangulation matting: maintenance of transparency and isolation of objects from the backing.



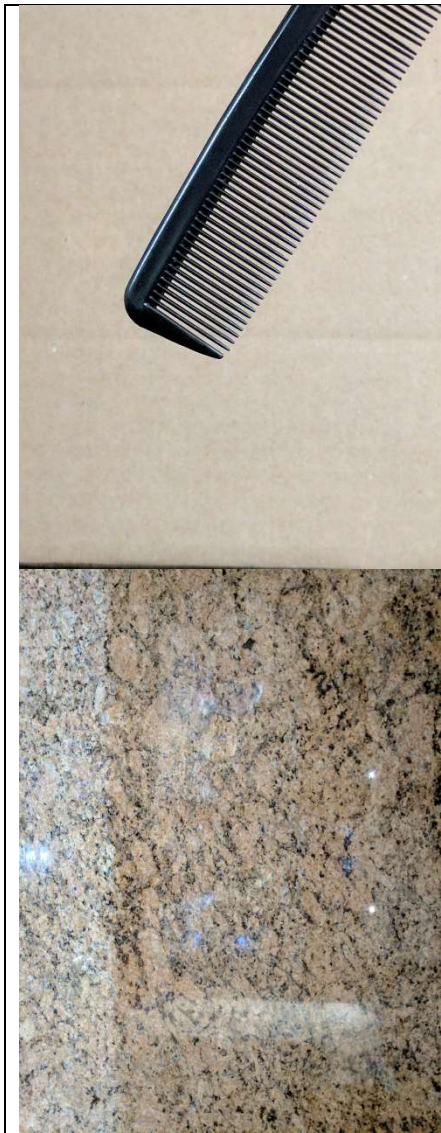


Transparency, similar backing colors (the two blankets are shades of each other)

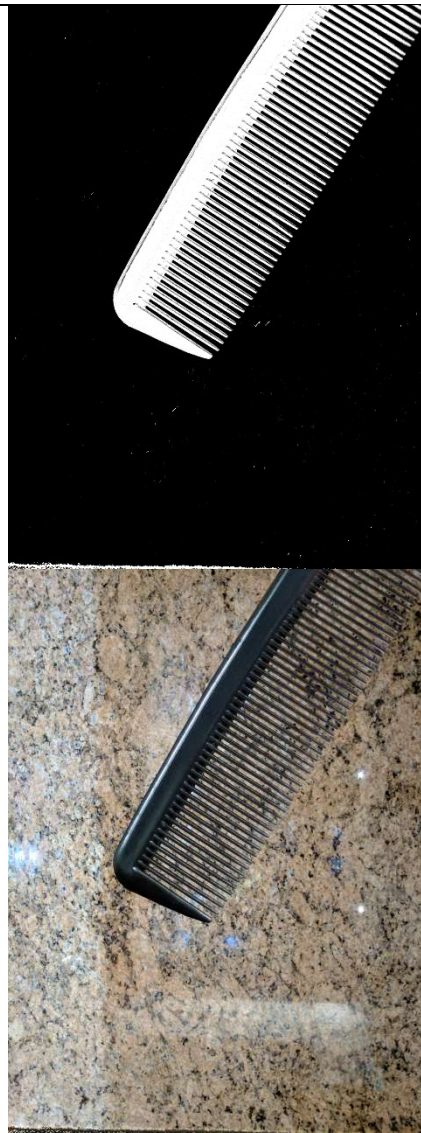


The test images given to us were a good test case to see how the method would cope when given background colors that are “closer” together. As noted in my efficiency notes, the alpha becomes unsolvable when the background differences become zero, so checking to see what happens as this difference decreases is desirable. Similar to the images above, a similar backing color allowed the program to accomplish the two main goals of triangulation matting: separation of the target from the backing, and maintenance of transparency





Object with high edge density (i.e. lots of gaps or holes), similar backing colors



This image set was mainly to test if my method was correctly differentiating edges. As it turns out, the resolution of the camera was high enough to properly capture the teeth of the comb, and since the alpha is calculated on a pixel-by-pixel basis, the edges are preserved.

I think that if the teeth were more reflective or closer together, they might cause some light to bleed around the edges in weird ways, potentially throwing off alpha values by making brighter sections be more similar to the plain cardboard backing.



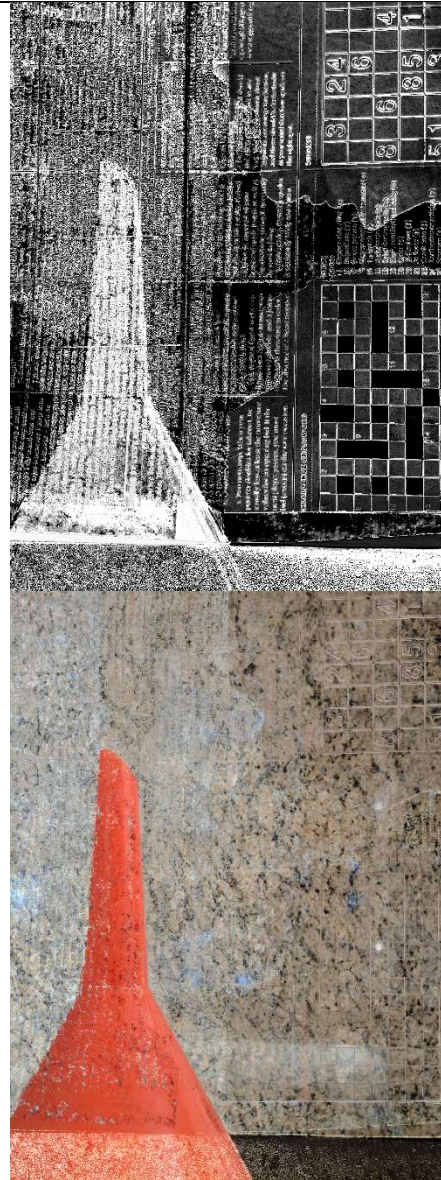
Reflective and
transparent object,
similar backing colors




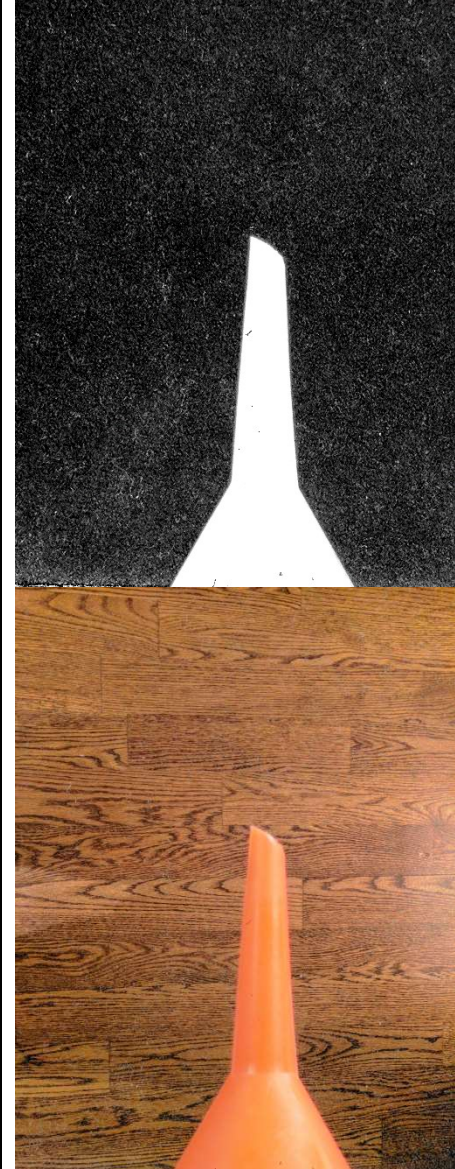
This image was to check what would happen if the method was fed an image-set of a reflective object. As I expected, the alpha values seem to also include the objects in the reflection (me and the woodshop behind me).



Background with text/other images (a newspaper in this case)



As I expected, triangulation matting had a very hard time differentiating between the target object and the objects in the backgrounds. As such, the alpha leaves behind traces of the objects IN the backing, which means that the resulting composite will also have elements of the objects in the backing. This is a failing case of triangulation matting – it cannot tell the difference between foreground and background objects.

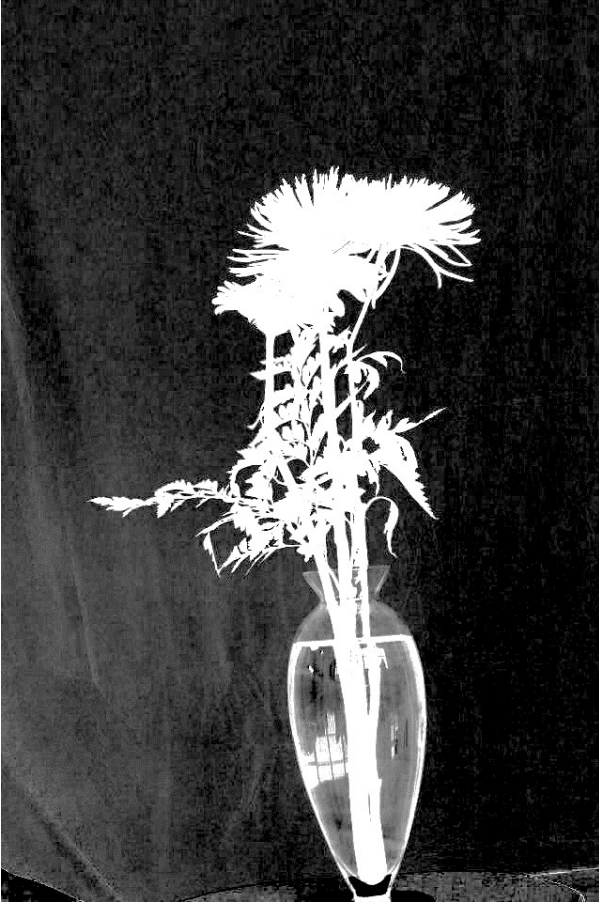
	<p>Noisy background</p>		<p>For this image set, I decided to decrease the lighting on my photo area, to increase the proportion of noise coming off of the backing (since noise is a bounded value, decreasing the overall intensity should increase the relative proportion of it).</p> <p>The resulting alpha has a lot of noise, as the variation between the black-backed foreground and background is very large. This is a failing case for triangulation matting – large enough changes between pairs of foregrounds and backgrounds are interpreted as objects being in the way.</p>
---	-------------------------	--	---

Experiment Conclusions

After reviewing the experiments done to evaluate the efficacy of triangulation matting, I found:

- Triangulation matting is best used when the backing colors are close together, but have a large intensity differential (e.g. black and cardboard, green and white, etc.) – this ensures that there are no irregularities in the alpha equation's numerator or denominator, keeping alpha consistent.
- Triangulation matting has a very low tolerance for differences between backing pixels for composite-backing image pairs, which reduces the effectiveness of the technique on images that have pronounced shadows (discussed later in the written question) or any sort of noise. Any meaningful difference between a composite-backing pair is interpreted as a high alpha (i.e. "there's probably an object there"), but if these differences are brought on by noise or shadows, it can leave undesirable elements in the resulting alpha.
- The technique behaves inconsistently when given different backing colors (e.g. blue and purple) – in one of the image sets, it over estimated the alpha immensely, but in another image set with the same backing colors, it under-estimated the alpha. Unpredictability is an undesirable property to have for any process.
- Since the technique is accomplished on a pixel-by-pixel basis, it is very good at differentiating between edges, provided the resolution is high enough. In my comb example, my camera had a high enough resolution that it could clearly discriminate between each gap in the comb, which meant that the technique had no problem determining edges. The only way to really fool the technique on edges is to give it a low resolution image set or an image set with very little difference between the backing and the foreground (e.g. black construction paper in front of a black backing).
- Triangulation matting cannot distinguish between the target objects, objects reflected in the target objects, or objects that are part of images in the backing. When these objects (especially those in the background) are introduced, the technique falls apart and calculates the alpha to be too high in certain backing areas.
- This can't really be observed in the images, but the process behind gathering each triangulation-mapping image-set is very difficult. Backdrop positioning, object positioning, backdrop uniformity, lighting, etc, must all be controlled for. Even though this technique has the capability to produce very good results, its applicability in the real world is very limited as it requires many variables to be controlled / kept constant between each image set.

Written Question – Alphas in the Flower Image



This image is what my implementation of triangulation matting gives as the alpha. For presentation purposes, I have increased the contrast in the image to make the differences easier to see in this document. As mentioned in the handout, the left side of this image is more intense (i.e. more white) than the right side of this image. As noted in my efficiency notes:

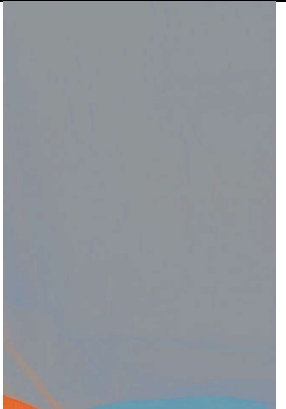

$$\alpha = 1 - \frac{(R_{f1} - R_{f2}) * (R_{k1} - R_{k2}) + (G_{f1} - G_{f2}) * (G_{k1} - G_{k2}) + (B_{f1} - B_{f2}) * (B_{k1} - B_{k2})}{(R_{k1} - R_{k2})^2 + (G_{k1} - G_{k2})^2 + (B_{k1} - B_{k2})^2}$$

For this question, we only care about the pixels that belong to the backdrop. Denote x_L, x_R as variables belonging to the pixels in the left and right sides of the image in question, respectively (e.g. α_L is the alpha on the left side of the image, R_{k1_L} would be the red channel of $k1$ on the left side).

Firstly, we will show that the colors of the two backdrops are roughly equivalent. This will be done with the following operation on every pixel. The operation will scale each pixel to a unit vector to determine the RGB ratios and therefore the color of each pixel, regardless of light intensity. It will then multiply each value in the unit vector by 255 so it can be written to a human-readable image.

$$pixel = (R,G,B)$$

$$scaledUnitColor(pixel) = \left(\frac{255}{R^2 + G^2 + B^2}\right) * pixel$$

Backing	Scaled Unit Color
A (gray sheet)	
B (darker gray sheet)	

Both the sheets seem to be shades of roughly the same grey, however the second sheet is noisier because the original pixels were darker and therefore had more noise. We will now consider the average intensities of these images, as well as the average intensities of their difference:

Image	Average		
	Red	Green	Blue
Backing A (pure color)	141.3	147.1	152.7
Backing B (pure color)	148.5	153.6	136.1
Absolute Difference between: Backing A Backing B	9.7	9.6	17.4

Notice that the average difference between the backing colors is very small (at worst, it's $\frac{17.4}{255} \approx 6.8\%$). Since we know that the two backgrounds are almost the same color, just with different brightness (with the brightness of k_1 being more than the brightness of k_2), we can say that:

$$y * (R_{k1}, G_{k1}, B_{k1}) \approx (R_{k2}, G_{k2}, B_{k2})$$

$$y < 1, y \in \mathbb{R}^+$$

Without loss of generality:

$$R_{k1} - R_{k2} \approx R_{k1} - yR_{k1} = (1 - y)R_{k1}$$

Thus (for any side of the image):

$$\alpha \approx 1 - \frac{(R_{f1} - R_{f2})(1 - y)R_{k1} + (G_{f1} - G_{f2})(1 - y)G_{k1} + (B_{f1} - B_{f2})(1 - y)B_{k1}}{(1 - y)^2 R_{k1}^2 + (1 - y)^2 G_{k1}^2 + (1 - y)^2 B_{k1}^2}$$

$$\alpha \approx 1 - \frac{(1 - y)(R_{k1}(R_{f1} - R_{f2}) + (G_{k1}(G_{f1} - G_{f2}) + (B_{k1}(B_{f1} - B_{f2})))}{(1 - y)^2 (R_{k1}^2 + G_{k1}^2 + B_{k1}^2)}$$

$$\alpha \approx 1 - \frac{(R_{k1}(R_{f1} - R_{f2}) + (G_{k1}(G_{f1} - G_{f2}) + (B_{k1}(B_{f1} - B_{f2})))}{(1 - y)(R_{k1}^2 + G_{k1}^2 + B_{k1}^2)}$$

Now we consider the right side of the image. We are only looking at the background pixels, and we notice that the right side of the A backdrop looks to be about the same in both the composite and background pictures for image set A . We also notice a similar phenomenon in the B image set. Mathematically:

$$(R_{f1_R}, G_{f1_R}, B_{f1_R}) \approx (R_{k1_R}, G_{k1_R}, B_{k1_R})$$

$$(R_{f2_R}, G_{f2_R}, B_{f2_R}) \approx (R_{k2_R}, G_{k2_R}, B_{k2_R})$$

Thus:

$$\alpha_R \approx 1 - \frac{(R_{k1_R}(R_{f1_R} - R_{f2_R}) + (G_{k1_R}(G_{f1_R} - G_{f2_R}) + (B_{k1_R}(B_{f1_R} - B_{f2_R})))}{(1 - y)(R_{k1_R}^2 + G_{k1_R}^2 + B_{k1_R}^2)}$$

By the previous approximation between the composite and background colors:

$$\alpha_R \approx 1 - \frac{(R_{k1_R}(R_{k1_R} - R_{k2_R}) + (G_{k1_R}(G_{k1_R} - G_{k2_R}) + (B_{k1_R}(B_{k1_R} - B_{k2_R})))}{(1 - y)(R_{k1_R}^2 + G_{k1_R}^2 + B_{k1_R}^2)}$$

Since the two backdrops are of roughly the same color (just with different intensities):

$$R_{k1_R} - R_{k2_R} \approx (1 - y)R_{k1_R}$$

$$\alpha_R \approx 1 - \frac{(R_{k1_R}(1 - y)(R_{k1_R})) + (G_{k1_R}(1 - y)(G_{k1_R})) + (B_{k1_R}(1 - y)(B_{k1_R}))}{(1 - y)(R_{k1_R}^2 + G_{k1_R}^2 + B_{k1_R}^2)}$$

$$\alpha_R \approx 1 - \frac{(1 - y)(R_{k1_R}^2 + G_{k1_R}^2 + B_{k1_R}^2)}{(1 - y)(R_{k1_R}^2 + G_{k1_R}^2 + B_{k1_R}^2)}$$

$$\alpha_R \approx 0$$

Thus, the fraction cancels out and the alphas on the right side should be approximately zero.

Now we consider the left side of the image. We are only looking at the background pixels, and we notice that the left side of the A backdrop looks to be darker in the composite than in the pure background image. We do not see this phenomenon in the B image set. Mathematically, for a large portion of left-side pixels:

$$R_{f1_L} < R_{k1_L}$$

$$G_{f1_L} < G_{k1_L}$$

$$B_{f1_L} < B_{k1_L}$$

$$(R_{f2_L}, G_{f2_L}, B_{f2_L}) \approx (R_{k2_L}, G_{k2_L}, B_{k2_L})$$

Due to the above approximation being true for a substantial number of left-side pixels, without loss of generality, the following will be true for a substantial number of left-side pixels:

$$(R_{f1_L} < R_{k1_L}) \wedge (R_{f2_L} \approx R_{k2_L})$$

$$\Rightarrow R_{f1_L} - R_{f2_L} < R_{k1_L} - R_{k2_L}$$

$$\Rightarrow (R_{f1_L} - R_{f2_L})(R_{k1_L} - R_{k2_L}) < (R_{k1_L} - R_{k2_L})^2$$

$$\Rightarrow (R_{f1_L} - R_{f2_L})(R_{k1_L} - R_{k2_L}) + (G_{f1_L} - G_{f2_L})(G_{k1_L} - G_{k2_L}) + (B_{f1_L} - B_{f2_L})(B_{k1_L} - B_{k2_L}) < (R_{k1_L} - R_{k2_L})^2 + (G_{k1_L} - G_{k2_L})^2 + (B_{k1_L} - B_{k2_L})^2$$

$$\Rightarrow \frac{(R_{f1_L} - R_{f2_L})(R_{k1_L} - R_{k2_L}) + (G_{f1_L} - G_{f2_L})(G_{k1_L} - G_{k2_L}) + (B_{f1_L} - B_{f2_L})(B_{k1_L} - B_{k2_L})}{(R_{k1_L} - R_{k2_L})^2 + (G_{k1_L} - G_{k2_L})^2 + (B_{k1_L} - B_{k2_L})^2} < 1$$

$$\Rightarrow -\frac{(R_{f1_L} - R_{f2_L})(R_{k1_L} - R_{k2_L}) + (G_{f1_L} - G_{f2_L})(G_{k1_L} - G_{k2_L}) + (B_{f1_L} - B_{f2_L})(B_{k1_L} - B_{k2_L})}{(R_{k1_L} - R_{k2_L})^2 + (G_{k1_L} - G_{k2_L})^2 + (B_{k1_L} - B_{k2_L})^2} > -1$$

$$\Rightarrow 1 - \frac{(R_{f1_L} - R_{f2_L})(R_{k1_L} - R_{k2_L}) + (G_{f1_L} - G_{f2_L})(G_{k1_L} - G_{k2_L}) + (B_{f1_L} - B_{f2_L})(B_{k1_L} - B_{k2_L})}{(R_{k1_L} - R_{k2_L})^2 + (G_{k1_L} - G_{k2_L})^2 + (B_{k1_L} - B_{k2_L})^2} > 0$$

$$\Rightarrow \alpha_L = 1 - \frac{(R_{f1_L} - R_{f2_L})(R_{k1_L} - R_{k2_L}) + (G_{f1_L} - G_{f2_L})(G_{k1_L} - G_{k2_L}) + (B_{f1_L} - B_{f2_L})(B_{k1_L} - B_{k2_L})}{(R_{k1_L} - R_{k2_L})^2 + (G_{k1_L} - G_{k2_L})^2 + (B_{k1_L} - B_{k2_L})^2} > 0$$

$$\Rightarrow \alpha_L > 0$$

Thus, on the left side of the image, the alpha values are will be non-zero for a substantial number of pixels.

Bibliography

Dasgupta, Sanjoy, Christos Papadimitriou, and Umesh Vazirani. 2006. *Algorithms*. New York: McGraw-Hill Education.