Triangulation Matting with PatchMatch Evaluations

Nicole Xin Yue Wang
University of Toronto
27 King's College Cir, Toronto, ON M5S
nicoletut.wang@mail.utoronto.ca

1. Introduction

The matting problem of imaging is a classical problem studied by many computer visioners for it to become an efficient and automatic imaging methodology. This problem studies the technique of separating the foreground and background of a given image. If you look at behind-the-scene videos of $Avengers$ or some fiction movies, you can find out that blue and green backgrounds are used most often to extract the foreground and put it into artificial backgrounds in the movie industry. However, limitations are that whatever color is used in the background would prohibit the use of that color in the foreground. Moreover, it may not be that easy to create a background in a constant blue or green color. We want to find a methodology of matting that is friendly for general people and efficient.

2. Previous work

A solution to the problem is the technique proposed by A. R. Smith and J. F. Blinn, which they refer to as Triangulation Matting [2]. This technique requires four images as input for extracting the foreground object. It takes two images of the object, but with different backgrounds, and two more images that are only the two backgrounds but without the foreground object. An example of a set of four input images is given at Figure 1. This technique assumes that the color we see is a composition of some proportion of the foreground and some proportion of the background color, and this proportion is the transparency of the foreground object, defined as α. For a pixel i, the relationship of its foreground color F_i, background color K_i, composite color C_i, and Alpha channel α_i is given by:

$$C_i = \alpha_i F_i + (1 - \alpha_i) B_i$$

With this assumption, by the given four input images indicating them as f_1, f_2, k_1, and k_2 as the two images with foreground and two images with only backgrounds, respectively, we can calculate the alpha value on every pixel by:

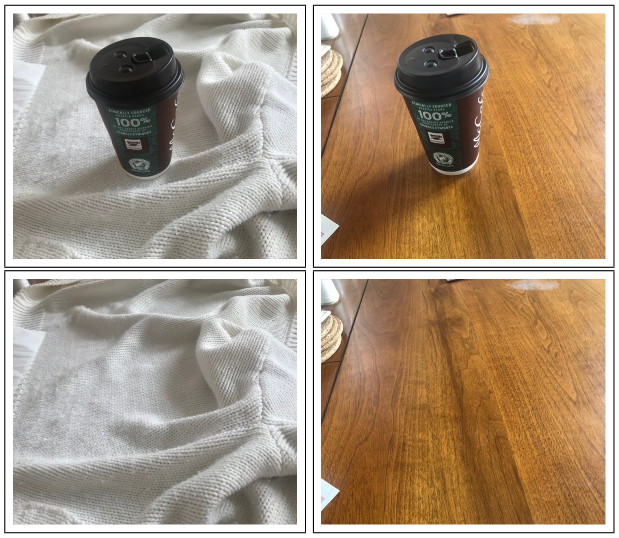

Figure 1. Input image for the Triangulation Matting algorithm, denoted as $compA$, $compB$, $backA$,and $backB$, from left to right, top to bottom. $compA$ and $backB$ share the same background, $compB$ and $backB$ share the same background, while $compA$ and $compB$ share the same object.

$$1 - \frac{(R_{f_1} - R_{f_1})(R_{k_1} - R_{k_2}) + (G_{f_1} - G_{f_2})(G_{k_1} - G_{k_2}) + (B_{f_1} - B_{f_2})(B_{k_1} - B_{k_2})}{(R_{k_1} - R_{k_2})^2 + (B_{k_1} - B_{k_2})^2 + (G_{k_1} - R_{G_2})^2}$$

Now, knowing the alpha channel for every pixel, which is the transparency of the object in the composite image, we can extract the foreground image by multiplying the alpha channel image to the composite image.

2.1. The Problem

One big problem of this technique is that the position and orientation of the object in the two different background images are required to be strictly the same in order to have the algorithm to perform well. This is very hard to achieve without very professional equipment, unless images are edited to line up each other, but that would not meet our goal for it to be an automated technique. For example, the two input images of object in different backgrounds, $compA$ and $compB$ given in Figure 1, has the cup

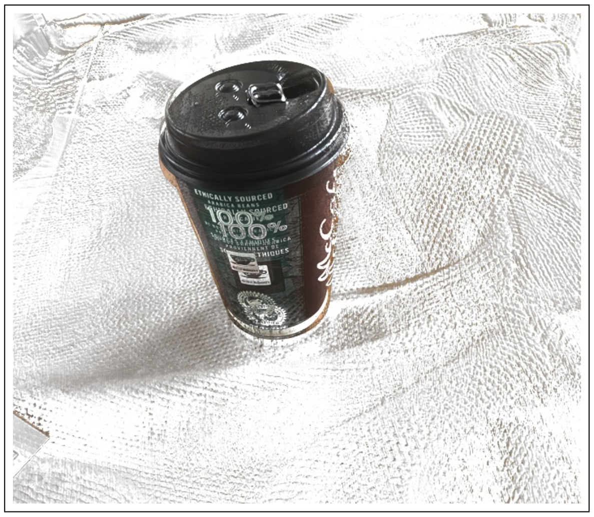

Figure 2. Output image of the Triangulation Matting algorithm, applies on the four images shown in Figure 1, composed on a white blank background.

in $compB$ positioned a little higher than the cup in $compA$. If we apply Triangulation Matting as describe in Blinn and Smith's paper, then the foreground separated is shown in Figure 2. We can see that there are lots of obvious artifacts. Words on the cup are mostly duplicated in different pixels. Unwanted areas surrounding the cup, and also there are missing areas on the cup lid's border. This leads to the topic of this study. We want to find a way to allow small position and orientation errors. The noisy areas outside of the cup is another limitation of Triangulation Matting, that is due to similar pixels between the two different backgrounds existed by coincidence, and we will not discuss this in this paper.

3. Solution

In order to allow small differences of the object between the two composition image A and B, one step is added to the algorithm. The set of alpha values computed using the technique of Triangulation Matting is on the similarity on every pixel between the composite images, approximately excluding the background. If the position or orientation of the object is even just slightly different between A and B, pixels where the objects don't line up will end up in wrong alpha value. In other words, the alpha channel computed by the original Triangulation Matting shows the transparency of the foreground of the image that is a weird combination of A and B. It cannot be used to accurately find the foreground in neither A nor B independently. Therefore, composing a colored output using this set of alpha channels with the combination of A and B will not result in an accurate separation. The solution is to create two sets of alpha values which corresponds to each of A and B.

3.1. Algorithm

We assume that there is only one object in the foreground, or in other words, the foreground can be found at a limited area smaller than the whole size of the image. First, follow the original steps of the Triangulation Matting algorithm until we found the alpha values on every pixel using the equation given. After computing the alpha channel for each pixel, we look for the minimum limited area where the object could be found, which is the minimum area of clustering pixels with high alpha values. Then pixels outside of the area are all zero.

Using current set of alpha values, compose two color output images of the foreground, $colA$ and $colB$, using composition image A and B as sources, respectively. In the limited area, for every pixel in $colA$, find a patch match in composition image $colB$. By assuming that foreground of A and B are identical in its appearance and the backgrounds are very different, if we can find a very good match in $colB$ for a pixel in $colA$, then there is a higher probability for that pixel in $colA$ is part of the foreground, so we increase its alpha value. If a pixel in $colA$ could not find a good match in $colB$, then there is a higher probability that this pixel is part of the background, so we decrease its alpha value. The updated set of alpha values is denoted by $alphaA$. Then, repeat this step of updating alpha values for the original set of alphas by finding matches for pixels in $colB$ to $colA$, the resulted set of alphas is denoted as $alphaB$. $alphaA$ and $alphaB$ will be more specifically indicating the pixels of the foreground in A and B, respectively.

Now, compose a new set of $colA$ and $colB$ using the new sets of alpha value, $alphaA$ and $alphaB$, and repeat this process of updating the two sets of alphas for a few iterations, this number of iterations. Each of the two final alpha sets should be accurately showing the transparency of every pixel of the foreground in each the two composition images. Therefore, construct a final $colA$ using the final $alphaA$, or construct a final $colB$ using the final $alphaA$, any one of $colA$ and $colB$ is the separated foreground.

3.2. Patch Match

The implementation of finding a pair of matching pixels between A and B uses the PatchMatch algorithm introduced by C. Barnes and his coworkers [1]. PatchMatch initializes with pixels in A to random pixels in B. Then it updates the matches for every pixel with neighbour pixels or random pixels if it could find a better match. Since the object is positioned and oriented approximately the same, we initialize PatchMatch with matching the same pixel in A and B.

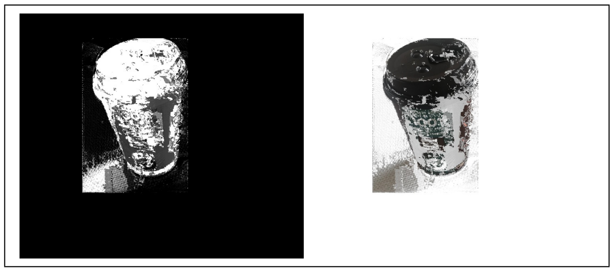
Figure 3. Output alpha image and color image of applying the algorithm which times alpha by 2 if a good match is found, and times alpha by 0.5 if a bad match is found.

4. Experimenting

We set up parameters for PatchMatch with patches to be sized 5x5. For the random search step, the maximum search radius is half of the limited area's width, and ratio between search window size to be 0.5. And for every time looking for a set of matches between two images, we iterate propagation and random search 5 times. And update the two sets of alpha values twice using PatchMatch.

A variable we want to discover is the threshold that determines a match to be a good match or a bad match. If it is a good match, we want to increase its alpha value; if it is a bad match, we want to decrease its alpha value. Also, we want to find out the rate of updating the alpha value if we find a good or bad match. In the following demonstrations, distance between a pair of patch ,X and Y, is calculated by:

$$dis(X,Y) = \frac{1}{3p}\sum_{i=1}^{p}\Big[(X_{iR}-Y_{iR})^2 + (X_{iG}-Y_{iG})^2 + (X_{iB}-Y_{iB})^2\Big]$$

where p indicates the number of pixels in a patch, and R, G, B denotes the RGB color values. So for example, X_{iR} indicates the red color value of the ith pixel in patch X.

By fixing the threshold of a good match to be 500, and threshold of a bad match to be 10000, we try on different rates of updating alpha values. Shown in Figure 3, Figure 4, and Figure 5. Color outputs are composed onto a white blank background for better visualizations.

From Figure 3, alpha values are updated very fast. The result is very reasonable since PatchMatch can not guarantee to find the match in 5 iterations even if one exists, we are not sure about whether the match for a pixel found is representative enough to show that pixel is part of the foreground. Setting the update rate in this way will result in unnecessary updates. The other two outputs produce similar outputs, while Figure 5 has slightly more pixels with a natural look. In later experiments of this study, we will update alpha by 1.25 for a good match, and by 0.8 for a bad match.

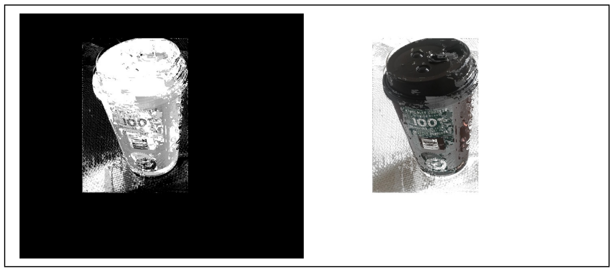
Figure 4. Output alpha image and color image of applying the algorithm which times alpha by 2 if a good match is found, and times alpha by 0.8 if a bad match is found.

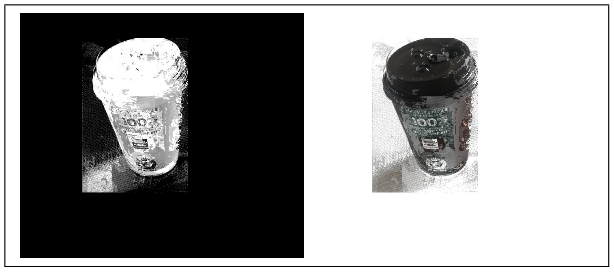
Figure 5. Output alpha image and color image of applying the algorithm which times alpha by 1.25 if a good match is found, and times alpha by 0.8 if a bad match is found.

By fixing one of the good match threshold or the bad match threshold, test different threshold of the other one to find the best threshold. As result, we found that the best threshold to conclude that a match is a good one would be around 1000, and threshold for a bad match is around 20000.

A result of setting parameters with previous findings are shown in Figure 6. And you can look at Figure 7 to have a sense of how it would look like if you compose it to a new background.

5. Limitations

We've only tested the algorithm with an object that is opaque, what if the object contains transparent or translucent parts? If we apply the algorithm, with best parameters found in the previous section, on input images shown in Figure 8.

Applying the algorithm onto the $glass$ set images would result in outputs that are almost exactly the same as the result of applying simply Triangulation Matting. This is because the alpha values are more complex with a transparent or translucent object. Not like when you have a opaque object, you can even determent the alpha value to be strictly 1 or 0, which give us the opportunity to update alpha by simply doing some multiplications. Therefore, the way we update the alpha for a pixel is inefficient when dealing with

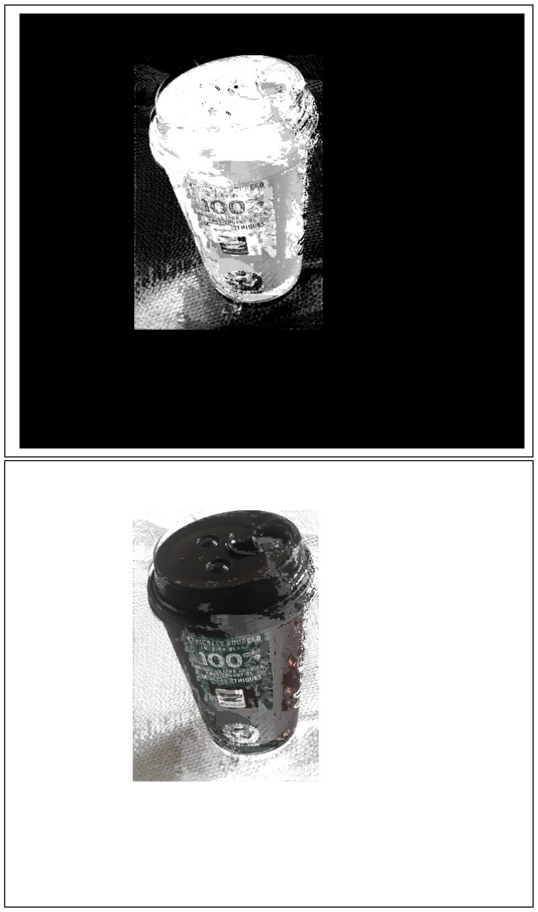

Figure 6. Output alpha image and color image of applying the algorithm with best set of parameters found.

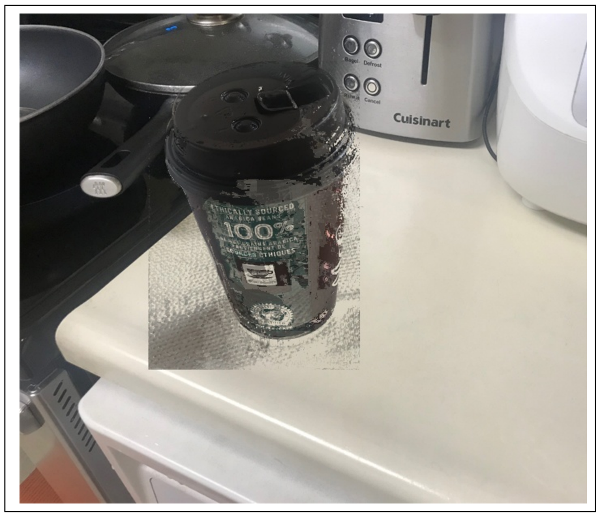

Figure 7. Output composed to a new background.

a transparent object. In order to have this algorithm to perform better with different types of objects, more studies on the way of updating alpha values are required.

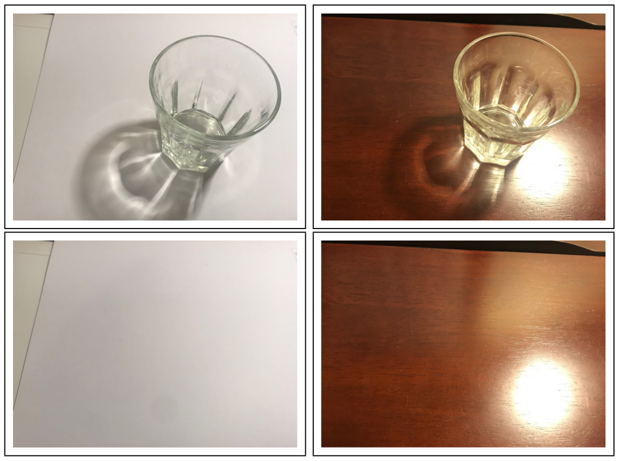

Figure 8. Another set of input images, $glass$, where the object is mostly transparent.

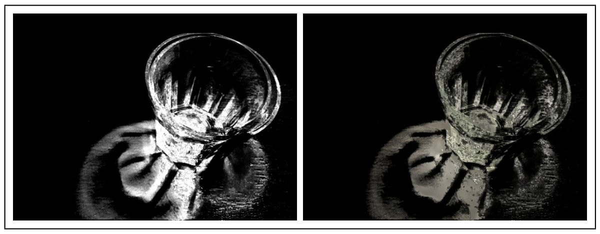

Figure 9. Output with the $glass$ set. You can see obviously that there are two circles at the top part of the glass, which is unnatural.

References

[1] A. Finkelstein C. Barnes, E. Shechtaman and D.B. Goldman. Patchmatch: A randomized algorithm for structural image editing. *Proc. SIGGRAPH*, 28(3):24, 2009.

[2] A. R. Smith and J. F. Blinn. Blue screen matting. *Proc. ACM SIGGRAPH*, pages 259–268, 1996.