REAL-TIME VIDEO MATTING FOR MIXED REALITY USING DEPTH GENERATED TRIMAPS

Nicholas Beato, Remo Pillat and Charles E. Hughes
Synthetic Reality Lab, University of Central Florida, 3100 Technology Pkwy, Orlando, U.S.A.

Keywords: Mixed Reality, Image Matting, Natural Image Matting, Real-time, Depth Sensor, Chroma-keying, Trimap Generation.

Abstract: In Mixed Reality scenarios, background replacement is a common way to immerse a user in a synthetic environment. Properly identifying the background pixels in an image or video is a difficult problem known as matting. Proper alpha mattes usually come from human guidance, special hardware setups, or color dependent algorithms. This is a consequence of the under-constrained nature of the per pixel alpha blending equation. While the field of natural image matting has made progress finding a least squares solution for an alpha matte, the generation of trimaps, indicating regions of known foreground and background pixels, normally requires human interaction or offline computation. We overcome these limitations by combining a low fidelity depth image that segments the original video signal with a real-time parallel natural image matting technique that favors objects with similar colors in the foreground and background. This allows us to provide real-time alpha mattes for Mixed Reality scenarios that take place in relatively controlled environments. As a consequence, while monochromatic backdrops (such as green screens or retro-reflective material) aid the algorithm's accuracy, they are not an explicit requirement.

1 INTRODUCTION

In Mixed Reality scenarios, the desired experience often requires graphically immersing a user in a virtual environment. This blend of real and synthetic objects is usually done by capturing a video either of the user or from the perspective of the user, processing the video to identify which real objects should appear on top of the virtual background, and layering foreground computer graphics if necessary. The process of isolating the real objects is known as matting.

Alpha matting produces a value, α, for each pixel, I, in an image that may be used to composite a foreground image on a background. This is done by the alpha blending equation,

$$I = \alpha F + (1-\alpha)B, \tag{1}$$

where I is the observed color, F is the foreground color, and B is the background color. Note that α is 0 when the foreground is completely transparent and 1 when it is completely opaque. This results in three equations with seven unknowns (F, B, and α). The only known is I, the observed video pixel. This inherently makes the problem under-constrained (Smith and Blinn, 1996).

To overcome this restriction, matting is typically done with a constant color background, such as a blue screen or a green screen. This process is known as *chroma-keying* and requires some prior knowledge of the background color (acquired either algorithmically or through offline training). When the constant color constraint is relaxed to include real environments, the method is referred to as *natural image matting*. To date, most natural image solutions require human input in the form of trimaps or scribbles as known foreground and background constraints to account for the lack of prior knowledge in the scene.

Automating the generation of trimaps for video sequences can be done with special hardware setups. This involves modifying an existing camera system (Gvili et al., 2003; McGuire et al., 2006) or adding multiple sensors (Wang et al., 2007; Joshi et al., 2006). The use of a depth camera has become more practical and affordable with the introduction of the Microsoft Kinect (Microsoft, 2011).

For Mixed Reality applications, an ideal real-time matting solution allows for *dynamic camera motion*, is *tolerant to environmental changes* (such as lighting), and works in *environments that may not contain constant color backgrounds*. In addition, the chosen

Beato N., Pillat R. and E. Hughes C..
REAL-TIME VIDEO MATTING FOR MIXED REALITY USING DEPTH GENERATED TRIMAPS.
DOI: 10.5220/0003870402800288
In *Proceedings of the International Conference on Computer Graphics Theory and Applications* (GRAPP-2012), pages 280-288
ISBN: 978-989-8565-02-0

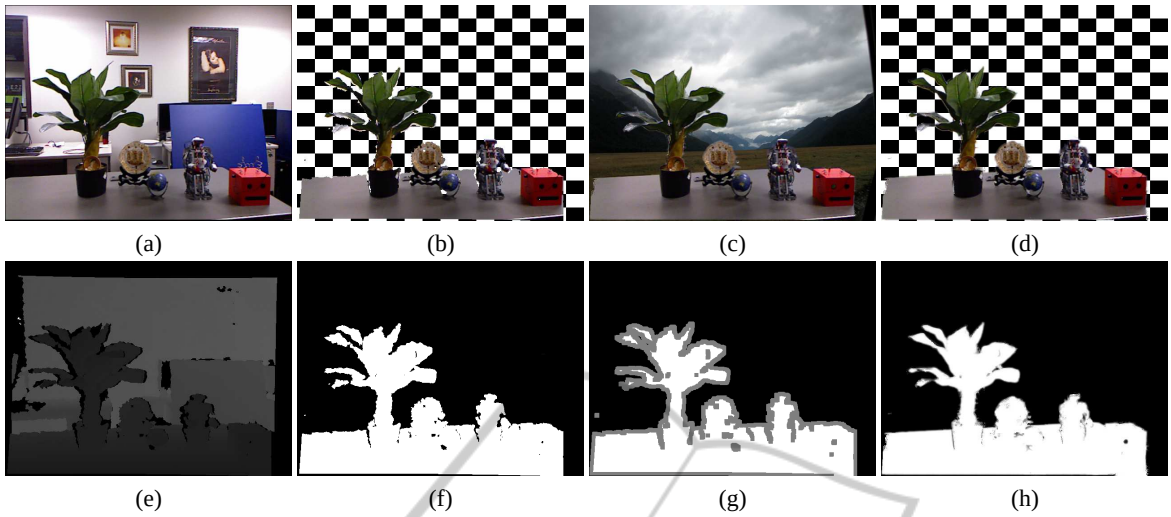


Figure 1: Demonstration of the steps in the matting process using a Microsoft Kinect. The input is the color image (a) and depth image (e). A depth threshold is applied to create a segmented image (f), which is both eroded and dilated to create a trimap (g). A natural image matter solves the alpha matte (h), which is used to composite against a different background (c) & (d). Using the segmented image alone produces noticeable artifacts during compositing (b). The entire process takes about 5.588 ms at 640x480 on a GTX580 (including transferring images to the GPU).

matting technique should *exploit the temporal information* gained from a live video stream.

Selecting a natural image matting technique would be ideal, but the generation of trimaps in dynamic scenarios poses a difficulty. To address this, we consider exterior data as in (Gvili et al., 2003; Wang et al., 2007). Our observation is that given a low cost depth camera, such as a Microsoft Kinect, it is possible to obtain a trimap from a 3D scene in real-time (Wang et al., 2007). It is then possible to execute a natural image matting solver in real-time on a live video, given the trimap (Gastal and Oliveira, 2010). By combining these two approaches and running them on a commodity GPU, it is possible to perform believable alpha matting on live videos in real-time. This allows us to simplify the Mixed Reality chroma-key setup, as no priors are required, and to relax background constraints for Mixed Reality experiences that require alpha matting.

2 RELATED WORK

While alpha matting is under-constrained, exact solutions may be computed if the foreground object is captured against multiple backgrounds (Smith and Blinn, 1996). Solutions that take advantage of this require quickly changing exposure times (Sun et al., 2006), use multiple cameras (Joshi et al., 2006), depth images (Gvili et al., 2003), or other special hardware (McGuire et al., 2006). These solutions may pose problems in generic Mixed Reality environments. For example, adjusting the shutter speed of a camera may not be possible if objects are moving in the scene. Modifying existing cameras is usually not practical for off-the-shelf devices, e.g., video see-through HMDs with built-in cameras or smart phones.

Using a constant background color enables the calculation of approximate alpha mattes using the color as a prior. This is commonly known as constant color matting, or chroma-keying. Solutions exist using color volumes (Bergh and Lalioti, 1999; Mishima, 1992; Beato et al., 2009) or specialized algorithms based on heuristics (Vlahos, 1978). While these methods run in real-time, they require training a model on the likely background color or hand-tuning parameters for optimal perception. Furthermore, these methods fail if the perceived color of the background varies due to illumination changes in the scene (such as lights dimming or flashing) or automatically adjusting exposure parameters of the cameras.

Natural image matting techniques tend to fall into global least squares approximations or sample based solutions. Global solutions such as (Levin et al., 2006; Levin et al., 2008; Shi and Malik, 2000) consider expensive processing of an nxn weight matrix, where n is the number of pixels in the image. Parallelizing a local minimizer to the matting linear system may be done via gradient descent or the conjugate gradient method (Gvili et al., 2003; Beato et al., 2009; Gastal and Oliveira, 2010). For these solutions to work well, the weight matrix must use global color

assumptions. In the case of the matting Laplacian, results in (Gastal and Oliveira, 2010) indicate that this approach, when used as an optimizer on an approximate solution, does not currently run in real-time.

For sample-based natural image solutions, the possible foreground and background candidates are assumed to span the known image regions and previously solved pixels (Gastal and Oliveira, 2010; Chuang et al., 2001; Ruzon and Tomasi, 2000; Wang and Cohen, 2007). These solutions work well when the unknown region is relatively small and contains known pixel colors. However, these methods do not address how to automatically generate known regions. An exception is (Gastal and Oliveira, 2010), which explored using Gaussian Mixture Models to model the color of a foreground object. This solution has the same pitfall as chroma-key solutions, in that changes in the color perception due to illumination may cause the trained model to improperly classify pixels.

(Levin et al., 2008) shows promise in automatically identifying an alpha matte for images without a trimap. However, this algorithm requires solving the smallest eigenvectors of the matting Laplacian, which is computationally expensive due to its large size.

In (Gvili et al., 2003; Wang et al., 2007), a depth camera is used to slice the scene, creating a trimap in real-time. While (Gvili et al., 2003) mattes in real-time, it uses a modified camera that adjusts the shutter speed for time-of-flight depth calculations. This may not be applicable for Mixed Reality scenarios. Also, their natural image optimization uses a custom weighted box filter to blur the alpha matte instead of minimizing the error. Solutions that use external depth cameras (Pitie and and Kokaram, 2010; Wang et al., 2007) are able to calculate a depth matte in real-time, but do not attempt to solve the natural image matting aspect of the problem in real-time.

3 REAL-TIME ALPHA MATTING FOR MIXED REALITY

Creating an alpha matte in real-time is accomplished by utilizing the parallelism of a modern GPU. We move the captured depth map (Figure 1e) and color image (Figure 1a) to the GPU as quickly as possible and process all data in parallel using localized algorithms. After the initial transfer, the algorithm involves two main stages as described in (Wang et al., 2007). First, we create a trimap from the depth capture. Then we apply a natural image matting solver. We stress that the general algorithm to solve this problem exists as well as the algorithms solving each respective subproblem. We wish to illustrate that combining real-time versions of the subproblems allows immediate visual feedback of the matting process, creating new opportunities for Mixed Reality applications.

3.1 Trimap Generation

To generate trimaps, we follow a few steps outlined in (Wang et al., 2007). We reiterate them here for clarity.

The pixel coordinates of the perceived depth image are back-projected into 3D space and then reprojected into the image coordinate frame of the color camera. It is assumed that the color camera has a fixed rotational and translational offset to the depth camera and that the their respective intrinsic camera parameters are known. This reprojection process is independent of the camera resolutions and handles any relative pose of the cameras. Note that excessive offsets or resolution differences will result in gaps in the new depth image that have to be handled by the algorithm. Any small gaps in the reprojected depth image are alleviated by applying a morphological closing operator as a final processing step.

Note that in this research we do not explicitly address latency between the two cameras, despite observing it. The exposure of the color and depth images is not synchronized and this could lead to a temporal offset of up to 17 ms (assuming no capture latency). This offset causes misalignments during motion of the cameras or the foreground objects. We rely on these motion artifacts being treated as unknown regions in the trimap and thus resolved by our image matting solution.

Once we have the depth image scaled and in correct coordinates, we apply a depth threshold. This depth threshold could be per pixel, but we choose a global cutoff as done in (Wang et al., 2007). The thresholded image is a black and white image, which is then both dilated and eroded (with a radius of 4) to create unknown regions along the borders (approximately 8 pixels wide). This radius is configurable, but we find a radius of 4 is a good value to cover alignment issues in the images, while protecting thin regions of foreground pixels from becoming completely unknown.

In certain circumstances, pixels in the depth map will have unknown values. While it makes sense to assign these to the unknown region, we found that most of these pixels are caused by background objects that are either far away from the depth sensor or do not reflect infrared well. Most foreground objects (non-glass) in our tests do not exhibit these problems and thus are well-defined in the depth image. As a result, pixels with unknown depth are assigned max-

imum depth. We note that this could be problematic when the unknown depth is caused from occlusion shadowing due to the projective transformations. From our observations, this issue is normally resolved by the morphological processing.

3.2 Natural Image Matting

When a video frame and its associated trimap are available, we want to solve the matte in real-time in order to use the result for Mixed Reality. In (Wang et al., 2007), the processing considered a depth modification to the Bayesian and Poisson matting systems. While their findings show that a Z-Cam improves the processing time, they do not achieve real-time speeds. We instead investigate using the Shared Sampling algorithm (Gastal and Oliveira, 2010). It is real-time, requires minimal, if any, parameter tuning, and works on GPU hardware.

For completeness, we outline the major stages of this algorithm, but refer the reader to (Gastal and Oliveira, 2010) for details. The Shared Sampling algorithm works in three stages. In the gathering stage, each pixel traces four rays in image space using a rotated cross configuration. Each pixel rotates the configuration slightly differently. Each ray returns at most one known foreground and one known background color. Out of the four possible foreground and background colors, the result with the lowest error is saved as a candidate foreground/background pair.

The next stage refines candidate choices within a neighborhood. The k_r nearest gathered samples are checked against an error function, where the best three samples are averaged. The original paper found k_r to perform well at about 200 samples. This stage allows the different orientations to vote foreground/background pairs from the different cross orientations. The averaged pair is used to compute a candidate alpha and confidence metric for the current pixel. We want to stress that the re-use of gathered candidates across multiple pixels is the feature of this algorithm that enables real-time processing.

The smoothing stage then computes a low frequency alpha matte using the high confidence alpha values using the nearest m samples. The authors found a decent smoothing size when m was about 100 pixels. During this stage, an adjusted confidence is computed. Given the high frequency alpha and the low frequency alpha, the adjusted confidence is used to interpolate the final alpha result. A result with high confidence will more likely use the high frequency alpha, whereas the lower confidence results will use the low frequency alpha.

4 SHARED SAMPLING MODIFICATIONS

While the Shared Sampling algorithm works in real-time, we made some algorithmic modifications to allow better performance in Mixed Reality scenes. We specifically addressed how the algorithm combines pixels' data during local searches, how the approximate foreground and background colors are computed, and how to make the algorithm more tolerant to video data.

4.1 Window-based Neighborhoods

First, we replaced the "nearest samples" idea with a kernel based approach. This allows us to sum over windows in the image instead of performing a "spiral search" looking for a specific number of the nearest pixels. We chose to use a 13x13 window during the sample refinement, as it includes nearly 200 pixels and is an odd sized window (and therefore centered). For the smoothing stage, we chose a 9x9 window for similar reasons (includes nearly 100 pixels). We note that we skip data from known regions during the summation, so we might underestimate the window size. We do not think this should affect performance.

4.2 Gathering using Weighted Sums

The Shared Sampling refinement stage runs by finding the best three pairs of candidate foreground and background samples within a neighborhood and averaging their values. Efficiently implementing this on a GPU is not a trivial task. This is because keeping track of the top three data elements requires significant branching within a warp on the GPU. When a single pixel needs to insert a new value, other data elements need to process the same instructions (and vice versa). Additionally, we need more registers to store local information for the top three candidates. To completely bypass these concerns, we instead choose to calculate a weighted sum on pixel, p, using its inclusive neighbors, $q \in N[p]$, by looking back to Bayesian Matting (Chuang et al., 2001),

$$P(F_p, B_p, \tilde{\alpha}|I_p) = \frac{P(I_p|F_p, B_p, \tilde{\alpha})P(F_p)P(B_p)P(\tilde{\alpha})}{P(I_p)}, \quad (2)$$

where F_p is a potential foreground color for a pixel I_p, B_p is a potential background color, and $\tilde{\alpha}$ is the estimated alpha given I_p, F_p, and B_p,

$$\tilde{\alpha}(I_p, F_p, B_p) = \frac{(I_p - B_p) \cdot (F_p - B_p)}{||F_p - B_p||^2}. \quad (3)$$

In other words, $\tilde{\alpha}$ is indicative of the closest point to I_p on the line through F_p and B_p. Note that these values are not necessarily dependent on the same pixel location, p.

In their work, the authors of (Chuang et al., 2001) assume $P(I_p)$ is independent of the optimization and that $P(\tilde{\alpha})$ is constant. We instead assume $P(I_p)$ is constant without priors and that $P(\tilde{\alpha})$ should favor foreground or background selections (unless the image is highly transparent). For simplicity, we also assume that both the sampled foreground and background colors occur at equal rates, making $P(F_p)$ and $P(B_p)$ constants. While this assumption seems false due to the fact that colors occur in different frequencies, our experiments show that it is not too detrimental. We also want to note that in a scene that favors chroma-key techniques, we could specify a function for $P(B_p)$ using a Gaussian Mixture Model or similar technique. By replacing these constants in the original formula, we see that the probability of the parameters given the observed pixel is proportional to the product of the probability of an observed pixel given the parameters and the probability of observing alpha,

$$P(F_q, B_q, \tilde{\alpha}|I_p) = P(I_p|F_q, B_q, \tilde{\alpha})P(\tilde{\alpha})P_k, \quad (4)$$

where $P_k = P(F_p)P(B_p)/P(I_p)$ and is constant across the image for known foreground/background pairs.

If we know the probability that a candidate pair is a good choice, we can compute a probabilistic weighted sum of gathered candidate pairs. We choose to do this over averaging the 3 best solutions according to color line distance as in (Gastal and Oliveira, 2010),

$$\begin{aligned} F_p^r &= \frac{\sum_{q\in N[p]} P(F_q, B_q, \tilde{\alpha}|I_p)F_q}{\sum_{q\in N[p]} P(F_q, B_q, \tilde{\alpha}|I_p)} \\ &= \frac{\sum_{q\in N[p]} P(I_p|F_q, B_q, \tilde{\alpha})P(\tilde{\alpha})F_q}{\sum_{q\in N[p]} P(I_p|F_q, B_q, \tilde{\alpha})P(\tilde{\alpha})}, \end{aligned}$$

and similarly,

$$B_p^r = \frac{\sum_{q\in N[p]} P(I_p|F_q, B_q, \tilde{\alpha})P(\tilde{\alpha})B_q}{\sum_{q\in N[p]} P(I_p|F_q, B_q, \tilde{\alpha})P(\tilde{\alpha})}.$$

For clarity, we define the probability as done in (Chuang et al., 2001). Given a fixed candidate foreground/background pair known to exist, we can calculate the probability that the pair correctly solves alpha for a given color by considering a normal distribution around the line connecting the foreground/background pair,

$$P(I_p|F_p, B_p, \tilde{\alpha_p}) = exp(-||I_p - \tilde{I_p}||^2/2\sigma_C^2), \quad (5)$$

where $\tilde{I_p} = \tilde{\alpha_p}F_p + (1-\tilde{\alpha_p})B_p$. σ_C is the number of standard deviations of the normal distribution around the line and we have fixed this value to 5 using a [0, 255] RGB cube.

For $P(\tilde{\alpha})$, we choose to estimate the transparent probability using a Gaussian centered on 0.5 with a tunable parameter for the "width" of the function. This causes the gather stage to favor alpha solutions near 0 or 1 while avoiding 0.5.

$$P(\tilde{\alpha}) = 1.0 - exp(-(\tilde{\alpha} - 0.5)^2/2\sigma_{\tilde{\alpha}}), \quad (6)$$

$\sigma_{\tilde{\alpha}}$ is the tunable parameter to control the biasing towards non-transparent solutions. We set this to 45 to produce a more uniform-like distribution that still biases away from transparent. In the case of fixed images with high-transparency, we can just set this equation to a uniform constant.

4.3 Smoothing Refined Samples

We noticed that, in videos, discontinuity in the trimap can cause large fluctuations to the refined foreground/background pair. We believe this contributes to a lot of the temporal noise that occurs on object boundaries during the matting process. To alleviate some of the noise in the foreground / background images, we perform a bilateral filter on the refined images and recompute the alpha and confidence from the filtered results. Alpha is computed as described in Equation (3). We choose to compute the confidence with $P(I_p|F_q, B_q, \tilde{\alpha})P(\tilde{\alpha})$ on the smoothed refinement pairs. For comparison, (Gastal and Oliveira, 2010) computes the confidence with only $P(I_p|F_q, B_q, \tilde{\alpha})$ on the non-smoothed pairs. This allows us to use samples more similar to the low frequency object colors in the scene. Since Mixed Reality environments typically do not have transparent objects and are fairly controlled scenes, we think using the filtered image is a fair trade-off.

4.4 Video Processing

We note that the Shared Sampling algorithm, while targeting real-time, does not add any type of temporal information during processing. As a result, the solutions can vary across frames, especially along object boundaries. This is especially true when the trimaps are not consistent across frames, which is a prevalent issue with depth sensors. We decided to address this by modifying the Shared Sampling algorithm for video processing.

We choose to extend the refinement stage and the newly added smoothing stage to a 3D process to include temporal information. Both modifications require summing over a 3D neighborhood of pixels (where the third dimension indicates previous frames

that are already gathered or refined). These changes cause the kernels to execute in cubic time instead of quadratic time, which causes performance issues on less powerful systems. However, we only need to run these filters on pixels that are currently unknown in the trimap. As a result, the alpha estimates are computed from temporally smoothed foreground / background pairs.

5 EXPERIMENTS

We group our experiments into a several categories. First, we want to show that our modifications to the Shared Matting system are not detrimental to the overall performance or accuracy. Then, we want to demonstrate that our modifications benefit the matting in an example Mixed Reality application. Finally, we want to show that we do not require training a model on priors and that we are robust to changes in camera position, background, and illumination.

To test that our changes do not significantly hurt the results or performance, we benchmark our implementation of the Shared Matting algorithm with and without our refinement stage adjustment and our filtering stage against the training dataset on *www.alphamatting.com* (Rhemann et al., 2009). Our implementation uses Nvidia's CUDA 4.0 API without shared memory compiled with 1.1 compatibility. Timing is done with hardware timers in the CUDA API. We mainly want to observe that the accuracy of the algorithm is not adversely affected, but we are also interested in performance gains.

In order to test our video modifications to the Shared Matting algorithm, we captured two videos with a green screen background. We used Adobe® Premiere® to find an acceptable alpha matte for the video sequences. These alpha mattes will be treated as ground truth. We made this choice because we are not aware of a data set for ground truth video matting data. The temporal modifications presented in Section 4.4 will be compared with the unmodified versions using the chroma-keyed ground truth.

To show that our algorithm works in a Mixed Reality setting, we process the frames in two distinct videos. As mentioned, one video is taken against a green screen background to allow us to compare our matting results to a known ground truth. The second video has camera motion, background changes, and illumination changes to test robustness to dynamic scenes. To show the flexibility of our algorithm the videos are taken with two different color cameras. In the first video, the color images are captured using a camera rigidly mounted underneath the Microsoft Kinect. For this experiment we use a web cam with 1280x720 pixel resolution. The intrinsic and extrinsic camera parameters of this two-camera system are extracted through standard calibration methods and the depth image is reprojected according to the process outlined in Section 3.1. This remapping process is handled offline for each video frame, but could easily be implemented on the GPU for online processing. The second video was captured using the Kinect's built-in color camera (at 640x480 pixel resolution). In this case, the re-mapping of the depth image happens in real-time, so no offline processing is necessary. We use the OpenNI drivers to access the raw depth information of the Kinect depth camera.

Note that we could use other features of OpenNI, such as the segmentation image, but we are more interested in the generic depth map solution. We note that the only parameter we adjusted is the distance of the depth slice. We intentionally keep the background farther away from the foreground to demonstrate proof of concept. Our goal is to informally demonstrate that this type of approach works for Mixed Reality applications.

6 RESULTS

For our first test, we observe that the overall error, both mean squared error (MSE) and sum of absolute difference (SAD), slightly increases with our refinement modification and becomes worse due to our smoothing stage (see columns 2 and 3 in Table 1). We note that the smoothing stage does not have this hindrance during video tests. Upon further inspection, we see that this is isolated to two (out of 27) images that are considerably worse, while one image is considerably better. On the remaining images, the refinement stage does not seem to affect accuracy.

In terms of performance, we benchmarked on an Nvidia GTX 580 and mobile GTS 360M. We see that the older mobile GPU gains an average 1.8x performance increase with this refinement modification. The algorithm modification runs an average 1.25x faster than the unmodified version on a modern GTX 580. For real-time applications, this is a significant savings, especially in the case of the mobile GPU, which achieves real-time processing rates in excess of 30 Hz through our modifications. The added filtering stage still executes quicker than the initial algorithm, but the results are slightly worse in this test (see column 4 in Table 1).

For our second test, we captured a 410-frame video sequence with a resolution of 1280x720 pixel in a green screen area. An example frame is shown

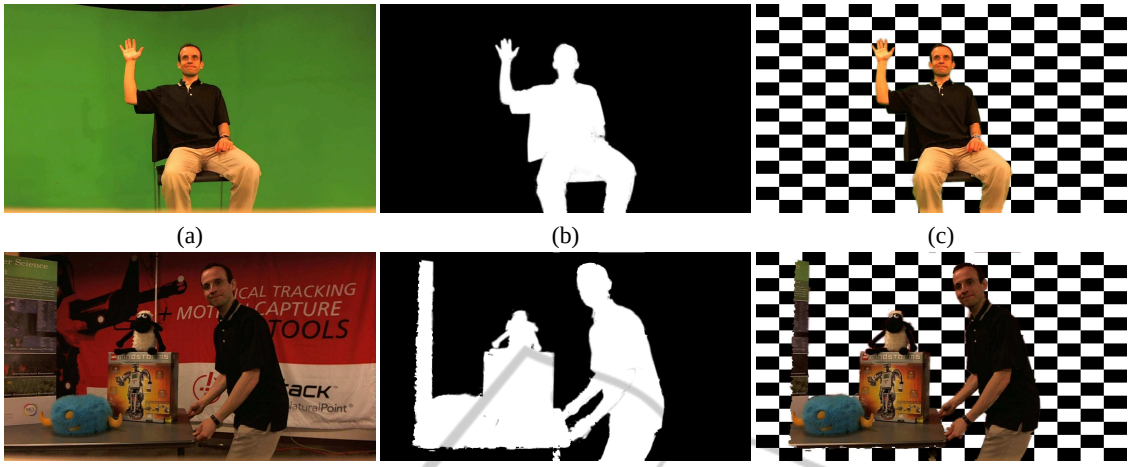


Figure 2: Subfigures (a) and (d) show selected color frames from two of our video sequences. The calculated alpha mattes are pictured in (b) and (e). (c) and (f) exhibit the results with replaced backgrounds. Note that the missing left sides in (e) and (f) are not due to an error in the algorithm but result from the different aspect ratios of the color and depth cameras. Image (a) is a selected frame from the second experiment.

Table 1: Comparison of the original implementation with our modified gathering stage and additional filtering stage. The first two rows show the average MSE and SAD on the www.alphamatting.com low resolution training set. The performance sections show the algorithmic run-time (without transfer) on a new GTX580 and an older mobile GTS 360M. We see that the refinement stage modification gives a noticeable performance boost, especially on the mobile GPU. The additional filtering stage does appear to give better results and is comparable to the original implementation.

	Shared Matting	Modified	Filtered
Error			
Average MSE	0.0040	0.0043	0.0050
Average SAD (x10^3)	5.6767	5.8914	6.2267
Performance GTX 580			
Average Time (ms)	7.9883	6.2100	7.7904
Best Time (ms)	3.0676	2.9653	3.7409
Worst Time (ms)	20.1341	12.9217	16.2894
Performance GTS 360M			
Average Time (ms)	43.3870	24.6122	33.3804
Best Time (ms)	18.5876	13.6247	17.6396
Worst Time (ms)	101.8040	46.9460	65.6575

in Figure 2a. The uniform background allowed us to generate a ground truth alpha matte for a realistic video sequence using the chroma-keying capabilities of Adobe® Premiere®. The results confirm our first test. Our modified implementation produces the same MSE as the original algorithm (0.0146 for both). However, it still exhibits a faster run-time (54.3 ms vs. 58.7 ms on average) using a Nvidia GTX 280. We want to note that the overall performance improvement is not as dramatic as in the first test. We suspect that this is because the unknown region is relatively small in comparison to the green screen background. As a result, the GPU is pruning the solver more than refining and smoothing the results. Figure 2d shows a frame from a different video taken with the same camera.

To test the temporal modifications from Section 4.4, our second green screen video was keyed using a varying number of previous frames during processing. As a base line with independent frame-by-frame processing, our algorithms yields an average run-time of 97.28 ms and an average MSE of 0.00762 on this sequence (using a GTX 280). Increasing the frame number in the smoothing stage from 1 to 5 lengthens the average run-time to 163.83 ms, but the average MSE grows slightly worse to 0.00768. In separate runs we varied the frame window for the refinement stage from 1 to 5. The average MSE shows the slightest improvement with 0.007568, but there is a big run-time penalty with an average of 221.02 ms.

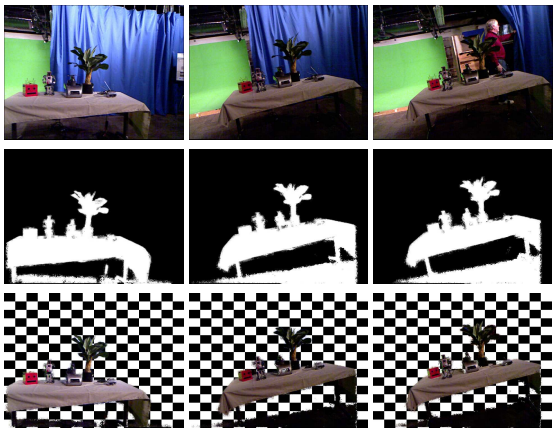
Unfortunately, we did not see any performance benefits in the temporal modifications and disabled them for all further tests. Our insight into this issue is that the Shared Sampling matter produces better results when the foreground and background samples are more representative of a good alpha. Temporally smoothing alpha with the foreground and background of many frames may in fact weaken the estimate alpha due to less specific information for that frame. We suspect this is also why a relatively small window size performs well during the smoothing stage. Even if the results were better, the increased run-time pushes us away from a real-time solution. This raises an issue worth addressing when using the Shared Sampling al-

gorithm for video processing, both on-line and off-line.

In our third test, we captured and processed a live video with the built-in Kinect color and depth cameras. Figure 3 shows frames with a resolution of 640x480 pixel from that video. Table 2 illustrates that the video is indeed processed fast enough to use in Mixed Reality scenarios. Our algorithm (naively implemented) only requires about 10 RGBA textures (of similar size to the input) depending on input and output usage. Textures could potentially be reused, but we avoided this for debugging. For temporal implementations, we needed to add four cache textures per frame. This leaves plenty of resources (both GPU memory and processor time) for rendering.

The second and third tests reveal artifacts due to darker regions in the image. This is expected as the Shared Matting algorithm requires finding good candidate colors early during the gathering stage. Other artifacts occur due to the offset between the depth and color cameras that can lead to pronounced shadowing effects.

We also observe some graininess in the alpha matte which we believe is due to the camera noise. As mentioned in (Wang et al., 2007), the floor causes an issue since the segmentation is based on a fixed-plane depth cutoff. Overall, we are excited to see that the algorithm works well with dimming lights and lack of chroma-key backgrounds, which is illustrated in Figure 3a-b.



(a) (b) (c)

Figure 3: Three frames from a video sequence with a moving camera processed in real-time. The video starts with fully lit scene (a). By frame (b), the lights in the immediate foreground are dimmed (not near the green screen). By the end of the video, the blue curtain is pulled away (c).

Table 2: An example 640x480 video is processed (over 450 frames) in real-time. Selected frames appear in Figure 3.

GPU	Average Time (ms)	Best Time (ms)	Worst Time (ms)
GTX 580	5.8132	4.1721	7.0529
GTS 360M	20.8613	12.9881	26.4724

7 CONCLUSIONS AND FUTURE WORK

We present an approach that combines real-time algorithms for trimap generation and natural image matting in order to compute the alpha channel in a live video. To achieve this, we presented adjustments to the Shared Sampling algorithm (Gastal and Oliveira, 2010), notably modifications to allow more efficient sample refinement. The adjustments achieve a noticeable speedup in experiments on images with different resolutions. This result is directly applicable to a multitude of Mixed Reality applications and a proof-of-concept implementation was demonstrated leaving plenty of GPU resources for rendering complete Mixed Reality scenes. Our method is resilient to illumination and background changes compared to approaches that use priors, places no constraints on camera movements, and requires only a readily available and affordable depth sensor. While we note that our attempt to obtain temporally consistent alpha mattes in the context of a video have not been fulfilled, we point out that, to our knowledge, this is the first implementation of natural image matting with automatic trimap generation that is able to process a video stream in real-time using only consumer-level hardware.

As noted in (Wang et al., 2007), the choice of a constant depth slice is problematic when the background and foreground interact. This problem could probably be alleviated for some common cases, e.g. for a floor or a ceiling by pre-segmenting the depth map and identifying constant planar features. We also note that the lack of a dataset with accurate ground-truth depth images makes it difficult to compare our results to other depth-map based matting approaches. Generating such a benchmark dataset will be one focus of our future activities. In additional future work, we would like to exploit the additional depth information in the image matting stage. Right now, the depth image is only used to automatically generate the trimap for the image matting algorithm, but it seems plausible that depth information could be incorporated in the algorithm itself. Intuitively, objects in the captured depth map should be locally smooth. This implies that the depth image moves between con-

tinuous objects when the second derivative in pixel space is non-zero. Using this intuition it should be possible to improve the matting results of our method. Finally, we would still like to find a way to make the Shared Sampling algorithm more resilient to trimap noise in videos. Our attempt to add some temporal consistency did not show ground-breaking results. We believe that better usage of multiple frames is a key problem to address if the Shared Sampling algorithm should ever be applied in a Mixed Reality system. An alternative approach would be to subsequently smooth the alpha values across multiple frames.

ACKNOWLEDGEMENTS

This material is based in part upon work supported by the National Science Foundation under Grant Numbers DRL0840297, DRL1113621 and CNS1051067. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

Beato, N., Zhang, Y., Colbert, M., Yamazawa, K., and Hughes, C. E. (2009). Interactive chroma keying for mixed reality. *Comput. Animat. Virtual Worlds*, 20:405–415.

Bergh, F. V. D. and Lalioti, V. (1999). Software chroma keying in an immersive virtual environment. *South African Computer Journal*, 24:155–162.

Chuang, Y.-Y., Curless, B., Salesin, D. H., and Szeliski, R. (2001). A bayesian approach to digital matting. In *Proceedings of IEEE CVPR 2001*, volume 2, pages 264–271. IEEE Computer Society.

Gastal, E. S. L. and Oliveira, M. M. (2010). Shared sampling for real-time alpha matting. *Computer Graphics Forum*, 29(2):575–584. Proceedings of Eurographics.

Gvili, R., Kaplan, A., Ofek, E., Yahav, Giora, ., and Benton, S. A. (2003). Depth keying. *Stereoscopic Displays and Virtual Reality Systems X*, 5006:564–574.

Joshi, N., Matusik, W., and Avidan, S. (2006). Natural video matting using camera arrays. *ACM Trans. Graph.*, 25:779–786.

Levin, A., Lischinski, D., and Weiss, Y. (2006). A closed form solution to natural image matting. In *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 1*, pages 61–68, Washington, DC, USA. IEEE Computer Society.

Levin, A., Rav-acha, A., and Lischinski, D. (2008). Spectral matting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30:1699–1712.

McGuire, M., Matusik, W., and Yerazunis, W. (2006). Practical, real-time studio matting using dual imagers. In *Rendering Techniques 2006 (Proceedings of the 17th Eurographics Symposium on Rendering)*. Eurographics, Eurographics Association.

Microsoft (last accessed May 2011). Kinect full body interaction. http://www.xbox.com/kinect.

Mishima, Y. (1992). A software chromakeyer using polyhedric slice. *NICOGRAPH'92*, pages 44–52.

Pitie and, F. and Kokaram, A. (2010). Matting with a depth map. In *Image Processing (ICIP), 2010 17th IEEE International Conference on*, pages 21–24.

Rhemann, C., Rother, C., Wang, J., Gelautz, M., Kohli, P., and Rott, P. (2009). A perceptually motivated online benchmark for image matting. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 1826–1833.

Ruzon, M. and Tomasi, C. (2000). Alpha estimation in natural images. In *Computer Vision and Pattern Recognition, 2000. Proceedings. IEEE Conference on*, volume 1, pages 18–25.

Shi, J. and Malik, J. (2000). Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*.

Smith, A. R. and Blinn, J. F. (1996). Blue screen matting. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, SIGGRAPH '96, pages 259–268, New York, NY, USA. ACM.

Sun, J., Li, Y., Bing, S., and yeung Shum, K. H. (2006). Flash matting. In *In Proceedings of ACM SIGGRAPH*, pages 772–778. ACM Press.

Vlahos, P. (July 11, 1978). Comprehensive Electronic Compositing System. U.S. Patent 4,100,569. Expired.

Wang, J. and Cohen, M. (2007). Optimized color sampling for robust matting. In *Computer Vision and Pattern Recognition, 2007. CVPR '07. IEEE Conference on*, pages 1–8.

Wang, O., Finger, J., Yang, Q., Davis, J., and Yang, R. (2007). Automatic natural video matting with depth. In *Computer Graphics and Applications, 2007. PG '07. 15th Pacific Conference on*, pages 469–472.