# Towards Real-time Mixed Reality Matting In Natural Scenes

Nicholas Beato
*University of Central Florida*

# TOWARDS REAL-TIME MIXED REALITY MATTING IN NATURAL SCENES

by

NICHOLAS BEATO
B.S. Computer Science, UCF, 2004
M.S. Computer Science, UCF, 2006

A dissertation submitted in partial fulfilment of the requirements
for the degree of Doctor of Philosophy
in the Department of Electrical Engineering and Computer Science
in the College of Engineering and Computer Science
at the University of Central Florida

Fall Term
2012

Major Professor: Charles E. Hughes

## 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.

In constant color matting, research identifies and replaces a background that is a single color, known as the chroma key color. Unfortunately, the algorithms force a controlled physical environment and favor constant, uniform lighting. More generic approaches, such as natural image matting, have made progress finding alpha matte solutions in environments with naturally occurring backgrounds. However, even for the quicker algorithms, the generation of trimaps, indicating regions of known foreground and background pixels, normally requires human interaction or offline computation.

This research addresses ways to automatically solve an alpha matte for an image in real-time, and by extension a video, using a consumer level GPU. It does so even in the context of noisy environments that result in less reliable constraints than found in controlled settings. To attack these challenges, we are particularly interested in automatically generating trimaps from depth buffers for dynamic scenes so that algorithms requiring more dense constraints may be used. The resulting computation is parallelizable so that it may run on a GPU and should work for natural images as well as chroma key backgrounds. Extra input may be required, but when this occurs, commodity hardware available in most Mixed Reality setups should be able to provide the input. 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.

Finally we explore a sub-image based approach to parallelize an existing hierarchical approach on high resolution imagery. We show that locality can be exploited to significantly reduce the memory and compute requirements of previously necessary when computing alpha mattes of high resolution images. We achieve this using a parallelizable scheme that is both independent of the matting algorithm and image features. Combined, these research topics provide a basis for Mixed Reality scenarios using real-time natural image matting on high definition video sources.

To my family and friends.

# ACKNOWLEDGMENTS

I first like to thank my advisor, Charles Hughes, and the members of my committee, Hassan Foroosh, Marshall Tappen, and J. Michael Moshell. I also appreciate the research contributions from and technical discussions with several colleagues throughout the course of this work: Mark Colbert, Yunjun Zhang, Kazumasa Yamazawa, Remo Pillat, Paulius Micikevicius, and Steven Braeger. I would also like to send an extended thanks to all of those at the Institute for Simulation and Training at UCF, especially those in the Media Convergence Laboratory and its successor, the Synthetic Reality Lab, which have supported my efforts over the last decade. I also want to acknowledge the the US Army RDECOM, Army Research Institute, Air Force Office of Scientific Research, Office of Naval Research, National Science Foundation (DRL0638977, DRL0840297), and Canon, Inc. for their support over the years. Finally, thank you to my family and friends, especially my mom, for their patience and motivation.

# TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# CHAPTER 1: INTRODUCTION

The ability to create an image showing a false reality has intrigued humankind as long as story telling has existed. Whether a mental image is painted by clever exposition or a modern day display renders a complex computer generated world, the imaginations of many have fueled the desire to perceive things that do not actually exist. With the advent of modern technology, producing visual portals into these worlds has become more and more common. Millions can immerse themselves in large fantasy worlds that later get translated into motion pictures and computer games.

The process of combining real and virtual worlds in a believable, interactive manner is typically the domain of Mixed Reality (MR). In the unified environment, a user may interact with both simulated and physical objects. While research has defined MR to span from purely virtual to real-world experiences, most MR environments are in one of two categories: *augmented reality*, where virtual elements are inserted into the physical space, or *augmented virtuality*, where real elements are composited into a virtual scene [MK94].

Many MR experiences rely on combining real and synthetic worlds visually. When it comes to identifying what is real, we can see ourselves in a movie theater trying to figure out what was real and what was computer graphics (CG). The technology that allows us to share vivid worlds with users is becoming more and more accessible. The number of people who create false memories in their own digital personal pictures by "Photoshopping" the original digital image is a testament to that. Given the proper tools, people can remove unwanted objects from an image or video. Or, they can select objects they care about and insert them into another image.

Processing complex video sequences is typically a challenge for the movie industry. Traditionally, it has been a time consuming (and money consuming) route. However, technology is beginning to allow quicker and easier processing of images in a specialized sequence. For example, background subtraction allows most users to impose themselves over different images, assuming the original capture has a stationary background and camera.

So when we talk about inserting real objects into a false world or inserting false objects into a real world, what is the process that goes on behind the scenes? Usually, the object that is being manipulated into another image has a color and opacity. The opacity is typically referred to as the alpha channel, or alpha matte. The inserted object is composited onto the background via the compositing equation [PD84],

$$C = \alpha F + (1 - \alpha) B. \tag{1.1}$$

In other words, the color of a composited image, $C$, is a combination of a foreground image, $F$, and background image, $B$. The method of combining these colors is typically a linear interpolation, using the $\alpha$ matte as a parameter. The operations in Equation 1.1 are applied in parallel to all pixels. So we could also view this as a per-pixel equation.

Given a background, foreground, and opacity, the rendering problem is a straightforward per pixel application of the compositing equation (Equation 1.1). The simplicity of this solution leads to our seeing many CG applications with transparency. The inverse problem, determining the opacity, foreground, and background from a composited image, poses more difficulty. This is necessary for the problem of background replacement, but can also be used for situations such as color replacement of transparent surfaces. This problem is known as *alpha matting*.

Alpha matting can fall into many categories. One of the most common methodologies is constant color matting, which is also known as blue-screening, green-screening, or chroma-keying. The idea is that objects of interest, such as humans interacting in a movie, are captured against a solid color background, such as green. The earlier methodologies processed a video (either during capture or offline) by classifying the pixel colors as chroma color or non chroma color. Typically, the objects of interest do not share similar color with the chroma color, so the process is simplified. This allows the relative amount of green, for instance, to determine the opacity of the object, which is typically opaque or invisible. Green and blue are typically chosen because they are not the dominant color in human skin pigments (which happens to be red, regardless of race).

Constant color backgrounds have some drawbacks that hinder their use in many applications. For example, to guarantee a saturated chroma color, special surfaces must be used. The lighting is typically focused and controlled on the chroma surface, so that shadows do not affect the surface. Shadowing is also avoided using special setups such as retro-reflective backdrops in TV studios. In this setup, a special material bounces light straight back to its source. A camera could have many LEDs around the lens, allowing a configurable color with minimal shadowing effects (since the lens is surrounded by the light source).

But what about cases where we don't want to be tied to special production setups? To solve the more generic problem, the field of natural image matting emerged. Natural image matting refers to identifying objects of interest in pictures typical in the real world. The concept is that humans can usually discern objects and transparency in an image, so the computer should also be able to achieve this. Overlooking things like camouflage and optical illusions in an image, the end goal is to process an image and obtain the opacity of the pixels in the image, assuming that the user indicates in some way what constitutes a background and foreground.

Numerous researchers have proposed methods to produce an alpha matte, allowing background replacement. Methods using a constant color background require a human to train the background color. From there, the perceived chroma color should remain relatively consistent to reduce the need for retraining. As a consequence, a controlled, stable environment becomes a limitation when using constant color backgrounds.

An alternative, natural image matting, works without the limitation of a trained background model. However, most of these techniques do not run in real-time. The exception is the shared sampling method [GO10], which does not address how to specify image constraints, i.e., which pixels are background or foreground.

Other methods typically depend on special hardware setups or clever imaging techniques, such as adjusting the exposure every frame or using multiple cameras with clever mirrors. Adapting these methods for real-time video is possible in some cases, but would require the manufacturing of hardware in the context of specific techniques and scenarios.

We will address the current state of related research in Chapter 2. Chapter 3 introduces related research using chroma color for Mixed Reality processing. Chapter 4 presents a real-time image matting method using a depth sensor to generate image constraints. Chapter 5 describes a method to combine subimage matting solutions and explores the effects of image locality on mattes. Chapter 6 discusses applications of matting in Mixed Reality scenarios. Chapter 7 summarizes this dissertation's research and offers possible directions of further research.

# CHAPTER 2: BACKGROUND

A comprehensive survey detailing most of the research in alpha matting algorithms that addresses images and videos was published by Wang and Cohen in 2007 [WC08]. An interested reader is referred to that survey for more information on methods not suited for real-time application or directly related to this document. This section will focus mainly on the key contributions in matting research relevant to computationally demanding applications. These applications primarily include real-time alpha matting of a live video feed, but may also include high resolution images that cause algorithmic difficulty in terms of memory or storage space.

The remainder of this chapter will be divided into the different directions matting research has followed. Each possible direction will typically follow a linear time line if it impacts the direction of research. We first cover the origins of the matting problem. We then look at constraints and how these effect the type of solutions that may be applied. After that we look at different methods of solving the problem and how they apply to the different types of input available.

It is also worth noting that *www.alphamatting.com* contains benchmarks used by most recent matting algorithms [RRW09].

## 2.1 The Matting Problem

As mentioned earlier, the alpha matting problem can be defined as the inverse of the compositing equation,

$$C = \alpha F + (1 - \alpha) B. \tag{2.1}$$

Here, we consider the observed image pixels, $C$, a composite image. We don't know the foreground, $F$, the background, $B$, or the alpha channels, $\alpha$. Typically, we want to solve $F$ and $\alpha$ so that the background may be replaced. We can consider $C$, $F$, and $B$ to be grayscale or color (such as RGB space). $\alpha$ is almost exclusively a vector of scalars. If we consider the observed pixels as a foreground image, $F$, blended with a background image, $B$, then in the simplest case we can view the alpha image, $\alpha$, as a Boolean indicator vector. $\alpha$ would be one for a foreground pixel and zero for a background pixel. In the more complex case, i.e. a transparent foreground object, $\alpha$ may take on other values between zero and one. The observed color is then a mixture of the unknown foreground and background. When only opaque objects exist in a scene, $\alpha$ may still take on intermediate values for pixels that appear on the boundaries of the foreground and background. This is a result of the camera discretizing the border onto a single pixel where the background and foreground are both observed.

The alpha matting problem was first formally investigated during the time when blue-screen technology was popular in the movie industry. Prior to this point, methods such as [Vla78, Mis92] were heuristically based on the background color and had non-intuitive parameters. Smith and Blinn attacked the alpha matte problem from a mathematical standpoint in [SB96].

In their work, several key observations were made that led future research down a different path. First, they noted that a single color pixel observation results in three linear equations with seven unknowns (for color images), illustrating that the problem is under-specified per pixel. This implies that future research must find ways to address the missing information in the equations. How the missing information is considered changes the approach of the matting algorithm. They also go on to give three examples of per pixel matting before generalizations are made.

First, a solution is shown to exist if all foreground colors are known to exist on a plane in 3D color space that is not shared with the background color. Furthermore, the background color must exist on a perpendicular line. This situation is not ideal for the real world, but does allow a solution to be computed.

A more reasonable solution comes from the assumption of gray tonality in the foreground or flesh colored people. The idea behind this assumption is that either the red or green channel of the foreground should vary with the background brightness (blue channel value). This constraint is related to prior heuristic approaches and shown to work in settings such as sci-fi movies, dominated by skin tones and washed out sets.

The third proposal, a radical suggestion for capturing images, paved way for more complex matting results. The idea is that having a captured image of an object against two different backgrounds provides an ideal solution. This method is known as *triangulation* and is still used to produce test sets for image matting. The idea is that having a sample with different background colors provides enough information to calculate the alpha matte. This is true even for natural backgrounds. However, if two observations have the same background color, the pixel will be confused as a foreground pixel.

## 2.2 Derived Equations

While most methods directly apply Equation 2.1, some derivations have been used to solve the alpha matte more indirectly. The Poisson matting method, [SJT04], observes that the derivative of the compositing equation,

$$\nabla I = (F - B)\nabla \alpha + \alpha \nabla F + (1 - a)\nabla B, \tag{2.2}$$

can be simplified under typical image constraints. The simplification is that local windows should exhibit relatively constant foreground and background colors. As a result, the derivative simplifies to an equation that suggests that the alpha gradient is proportional to the image gradient,

$$\nabla \alpha = \frac{1}{F - B}\nabla I. \tag{2.3}$$

The system is solved using a Poisson equation solver. Other research, such as [GO10], has since then has taken advantage of this observation.

Another approach to indirectly applying the Equation 2.1 comes from using second order statistics in a scene [JMA06]. When multiple frames of the same scene are available, the captured images aggregated produce an expected image color and variance estimate of the image pixels. The alpha matte is computed using estimates of the observed, foreground, and background variances and the compositing equation's variance (assuming that $F$ and $B$ are independent variables),

$$\mathbf{var}(I) = \alpha^2\mathbf{var}(F) + (1-\alpha)^2\mathbf{var}(B). \tag{2.4}$$

An auto-focus algorithm and local search are used to find estimates of $\mathbf{var}(F)$ and $\mathbf{var}(B)$. $\alpha$ may be solved using the quadratic formula.

## 2.3 Defining Constraints

The compositing equation itself does not disambiguate the foreground from the background. In fact, it is rather symmetric if the foreground and background colors are swapped. This means that without some prior knowledge, a foreground and background could be interchanged. To solve this problem, some constraints must be made on the image or the algorithm. Depending on the research direction, the types of constraints can make an impact on the final solution.

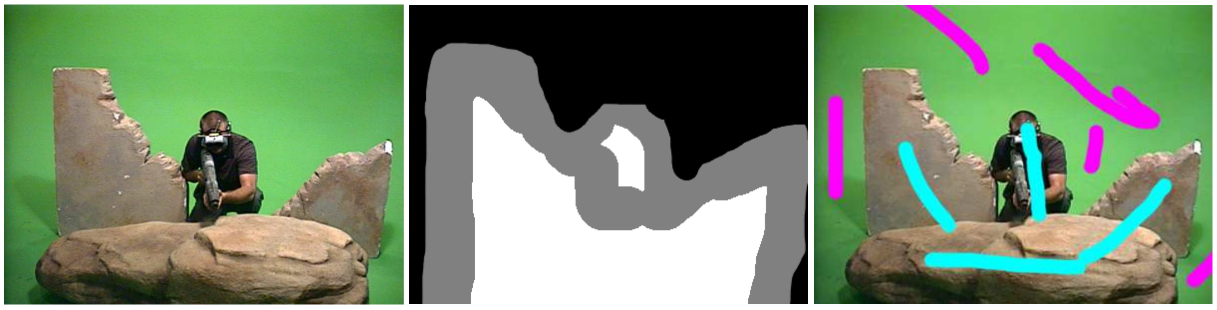

(a) (b) (c)

Figure 2.1: An example of different constraints. (a) shows an input image with constant color background. (b) shows a trimap for (a). (c) shows a scribbled image for (a).

### *2.3.1 User Image Constraints*

The typical solution to impose priors on the image is to mark pixels of interest in a painting application. The resulting algorithms require identifying known pixels and unknown pixels. The known pixels are broken up once again into foreground and background pixels. The unknown pixels typically appear near object boundaries or on transparent foreground regions. The difference between the following user constraints comes down to how the human interfaces with the matting algorithm. More specifically, we care about how automated the input is and how many pixels need to be known.

It may seem that the more pixels that are known or the more automated the identification process is, the faster a solver can identify unknowns. This is not necessarily true and will be addressed as the methods are discussed.

#### *2.3.1.1 Constant Color Matting*

As noted earlier, if the background is a known constant color, the background model can be assumed through a computation model. A user cares more about identifying pixels of similar color to the background color. The resulting histogram can train a background model, which subsequently identifies known background pixels. The same process can apply to determining foreground pixels. Sample models for the background include a mixture of Gaussians [RT00], image statistics [CCS01, BZC09], a geometric color volume [BZC09, Mis92, BL99a], or experience-derived heuristic [Vla78]. The resulting model can be applied in two different ways. First, we can partition the RGB color space more confidently using a background model. A histogram or look-up tables such as those in Autodesk® Combustion® falls into this category as well as the geometric color volume techniques. Second, we can use a trained model to determine the likelihood that a pixel matches that background model, which tends to be used by algorithms that estimate foreground and background colors [WC07, GO10, CCS01].

The constant color assumption allows us to automatically create known and unknown regions by classifying the composited input pixels. Some methods can exploit this nature to lower the amount of work done by a human as input. Furthermore, under the assumption that multiple images are taken in the same environment, the same model can be directly used or adapted on multiple images. This is why we commonly see constant color background algorithms during video capture. We are more interested in training a classifier that works on many images instead of directly marking each frame of a video.

#### *2.3.1.2 The Trimap*

When every pixel is positively marked as foreground, background, or unknown, the resulting labels are typically interpreted as a trimap [RT00, WC08]. Note that the labels are usually treated as mutually exclusive sets that represent hard constraints. The trimap may be automatically generated from alternative sources, such as secondary cameras, color models, or coarse solvers. Typically, these are generated by human users.

While the input for an automated matting system can potentially be transformed to a trimap, we tend to think of the trimap as a coarse image that explicitly marks all pixels. Since the unknown regions likely appear near foreground object boundaries, methods can assume that most pixels in a trimap appear near pixels with transparent alpha values.

#### *2.3.1.3 Image Scribbles*

Explicitly marking all pixels in an image can be time consuming for users when the derivation is not automated. To lower the requirements, scribbles are typically used [LLW06]. The goal of the scribble image is to quickly mark some foreground and background pixels, preferably with quick brush strokes in a painting application. The markings are typically done on a very small percentage of the image. This indicates that the unknown region should be treated more like an unmarked region rather than pixels near object boundaries. As a consequence, the unknown region will contain several more opaque pixels than transparent pixels. This is an important difference between a scribbled image and a trimap.

### *2.3.2 Automated Methods*

While moving from a model that can solve each pixel to a scribble based approach enables more efficient use of human effort, it does not completely remove the need of a human in the loop. The need to minimize human effort shows more as we begin to consider processing video sequences. Manually marking each frame is simply not an option. This is necessary with the constant color environments, as long as the environment is consistent.

But what can be done to automate the input to the matting algorithms? We can group efforts into advances based on color images alone and methods with additional inputs.

#### *2.3.2.1 A Spectral Approach*

We can solve the matting system in the least squares sense with the simplified equation,

$$L\alpha = \hat{0}, \tag{2.5}$$

where $L$ is the matting Laplacian. The derivation will be explained in Section 2.8. The interesting note is that the non-trivial solutions to this problem are the smallest eigenvectors of the matrix $L$. Each eigenvector relates to a cluster of closely related pixels on the image. As a result, some linear combination of the smallest eigenvectors indicates a solution to the matting problem that satisfies the constraints. When constraints are not given, determining the combination of eigenvectors may still be automated. The spectral matting method of Levin, et al. is an automated, but time-consuming, algorithm to estimate the alpha matte for images where a foreground object dominates the image [LRL08].

#### *2.3.2.2 Special Hardware and Additional Input*

As previously discussed, several techniques rely on special setups. The dual imagers technique, [MMY06], exploits the triangulation approach of Smith and Blinn, [SB96], by using mirrors to photograph an object with two backgrounds. Similarly, multiple cameras can be used to identify the matte with multiple focal points [JMA06]. The trimap is generated from different camera angles producing “shadows” after applying variance thresholds from focused images. For single shots, the exposure can be adjusted to give the illusion of different backgrounds [SLB06]. Depth sensors can be integrated to add more information to the current system, allowing depth slicing to aid the key selection, but these techniques usually have accuracy issues [GKO03].

## 2.4 Image Graph and Affinity Functions

Since we cannot solve the compositing equation directly from one pixel observation, several matting methods borrow the idea of a graph based approach from the image segmentation community. The graph may be used for optimization techniques (see Section 2.7) or it may be used to minimize error criteria. By definition, each pixel is considered a vertex in the resulting graph. Weighted edges are added between two pixels when they are neighbors. The choice for a weighting function is typically called the *affinity* function. We will denote the affinity function as $\nu(I,x,y)$, where $x$ and $y$ are pixel locations in an image, $I$.

The affinity is specified for entries where $x$ and $y$ are not the same pixel. A pixel relates to itself as the sum of the entries on a row. The graph adjacency matrix for a particular image, $I$, can then be defined as

$$W_I(x,y) = \begin{cases} \nu(I,x,y) & \text{if } x \in N(y) \\ 0 & \text{otherwise} \end{cases}, \tag{2.6}$$

where $\nu(I,x,y)$ is a function typically dependent on the image colors near pixels $x$ and $y$. The neighboring pixels of $x$ relate the graph structure to the image.

From the image graph, we can compute the image's Laplacian matrix,

$$L_I = D_I - W_I, \tag{2.7}$$

where $D_I$ is a diagonal matrix with $D_I(x,x) = \sum_{y \in N(x)} W_I(x,y)$. In other words, the sum of each row is stored in the diagonal.

### *2.4.1 Gaussian*

One of the more commonly used affinity functions is the 3D distance Gaussian [SM00, GO10]. The color distance is used so that similar colors are weighted higher than more dissimilar colors. This affinity requires a variance parameter that is typically a global user input. The function is defined as,

$$v_0(I,x,y) = exp\left(\frac{-||I_x - I_y||^2}{2\sigma^2}\right) \tag{2.8}$$

### *2.4.2 Closed-Form*

The Gaussian affinity may cause issues when local pixel neighborhoods take on largely different color variances. The solution to this problem is to use local covariance estimations to weight the pixels dependent on data in a small window. This method comes from an observation made by Levin et al. in [LLW06] and has been used in [WC07, HLW10, TMW11]. In their work, they assume that small neighborhoods can be used to overspecify the missing information in the per pixel matting equation. The error function minimizes the error on three unknowns within a neighborhood. The overspecification allows a least squares fitting to the two additional parameters. These extra parameters are tightly related to the foreground and background distributions. As a result, they show that their solution relates to the typical graph based methods used in segmentation papers. The weighting is commonly referred to as the *matting affinity*, and is defined as

$$v_M(I,x,y) = \sum_{p \in N(x,y)} \frac{1}{|N(x,y)|}(1 + (I_x - \mu_p)(\Sigma_p^{-1} + \frac{\varepsilon}{|N(x,y)|}\mathrm{I})(I_y - \mu_p)). \tag{2.9}$$

Here, we sum over all pixels $p$ in the neighborhood of both $x$ and $y$, $N(x,y)$. Otherwise, the covariance, $\Sigma_p$, and mean $\mu_p$ are user to calculate an inner product of $I_x$ and $I_y$. The inner products are summed over a neighborhood to overspecify the solution. $|N(x,y)|$ is just a normalization factor that is the size of the neighborhood.

## 2.5 Color Classifiers

Prior to Smith and Blinn's study of the matting problem [SB96], it was typical to see methods that computed an alpha value directly from an single observed pixel. This approach continued even after the problem was deemed ill-constrained. The general approach can be looked at as two distinct steps. First, the classifier identifies a value for the pixel dependent on whether or not we care about the foreground or background at a particular moment. Then we need a way to merge the estimate values into an alpha value.

For example, when we consider methods that split the color space into a background and foreground region, we see classifiers that might identify the closest known foreground and closest known background color. The resulting alpha is then computed as the linear distance to both of these estimate colors.

For constant color backgrounds, one approach is to cluster the background in 3D space, effectively dividing the color space into a background and foreground region. The shape of the bounding volume is indicative of the algorithm used. For example, Mishima et al. use a series of planes to create a convex polyhedra [Mis92]. The planes are adjusted to minimize the color volume over a training set. Another example, Bergh et al. use a few planes creating a pyramid optimized for blue backgrounds [BL99a].

## 2.6 Sample Based Solvers

The main pitfall of color classifiers is that the likelihood of a function mapping the observed pixel color to the correct alpha is very low. As a result, researchers of [WC07, CCS01, HST10, GO10, JMA06] considered another approach: What if the foreground and background colors are inferred from the observed colors? We could then directly solve for alpha. The question then becomes: How do we correctly determine the foreground and background colors?

The common idea as that we can use known values in the input constraints. For example, if we consider all pixels on the border of an unknown and known region (and note that some of the knowns are foreground and some are background), we could try all possible pairs of border foreground and background pixels. We call the bordering pixels samples, since we are sampling the known region in the image. When sampling, we want to know whether or not a pair is a good value. As a result, we construct an error function per sample pair, such as $E(C,F,B)$. Note that we don't explicitly consider the computed alpha value. A possible interpretation of $E(C,F,B)$ is,

$$E(C,F,B) = \frac{||C-(\alpha F+(1-\alpha)B)||}{||F-B||}, \tag{2.10}$$

where alpha is derived as,

$$\alpha = \frac{||C-B||}{||F-B||}. \tag{2.11}$$

The error normalization on $||F-B||$ tries to maximize the distance between the foreground and background colors, but is left out in [GO10]. The solved alpha can be interpreted as the projection of $C-B$ on $F-B$, which indicates the relative distance of $C$ from $B$ on the line segment connecting $F$ and $B$.

## 2.7 Graph-Based Solutions

Global approaches to the matting problem tend to generate a large Laplacian matrix from the affinity function. The Laplacian is then used to find the alpha vector using a least squares error approach. The typical function, first derived in [LLW06], but compared against [SM00], looks like,

$$\alpha_{opt} = \min_{\alpha} \alpha^T L \alpha. \tag{2.12}$$

The derivation estimates the local color statistics to find an approximate foreground and background color for each pixel. The least squares estimate eliminates the need for a foreground and background color in the matting equation. Most methods that use the matting affinity are said to generate the *matting Laplacian*. How the minimal alpha matte is determined tends to differ depending on the amount of memory and computation time available. It is worth noting that $L$ is positive, semi-definite, and symmetric. Different methods tend to slightly alter Equation 2.12 to encode the constraints better, such as [GO10, HST10].

### *2.7.1 The Brute Force Solution*

To illustrate the function better, we consider a brute force solution. Suppose we try all possible combinations of alpha values on an image (here we operate on the assumption that alpha is able to take on integer values between 0 and 255, inclusive), given enough computation time we just care to store the alpha mask with the minimal error. Unfortunately, we have $256^{h*w}$ possible combinations to try. Even for an image of size 2 by 2, this is computationally expensive.

So what can we do? We can reduce the precision of our alpha image. At best, a binary segmentation would still be exponential in the number of unknown image pixels. So we need to find a cleverer approach.

### *2.7.2 Direct Solver*

Under the assumption that we have a positive, semi-definite matrix (a linear system that has a unique minimum), the solution to a least squares problem can be directly computed by setting the derivative equal to the zero vector. We can use a Gaussian elimination method to solve the system of equations. This can be done via the back-slash operator in Matlab®. We note that for large sparse matrices, this is too time-consuming to consider. It also takes a large amount of system memory to represent larger images. Here we seek an alternative method to solve the system.

### *2.7.3 Min-Cut & Normalized Min-Cut*

When we consider an image as a graph, we can compare the alpha matte that generates the minimal error as an edge cut on the image graph. The cut is notably a min-cut. That is, we cannot select edges in a way that will create a smaller error value.

While a min-cut seems like it will properly group pixels together, there is a chance that the number of edges is minimized before the number of grouped regions in the image. As a result, a normalized cut [SM00] may be used instead to weight edges by the number of pixels in a known region.

### *2.7.4 Graph Connectivity*

As we begin to consider algorithms that depend on the matting Laplacian, we begin to see that the graph connectivity could effect the results. Typically, we consider neighborhoods of size 3x3 when generating edges (8-connected). Sometimes we consider a cross shaped neighborhood pattern (4-connected). Using local information like this results in a sparser matting Laplacian. This implies less storage (there are far fewer weights to store).

While it may seem that a fewer number of connections implies less computation requirements, [HST10] has shown that this is not necessarily the case. Their main observation is that a larger number of connections implies that information can propagate throughout the image graph quicker. The downside is that the results are less coarse, but they propose processing the image with a smaller neighborhood later on to address the granularity concern.

## 2.8 Minimization Techniques

In several instances, we are interested in minimizing a given least squares error function in a neighborhood of pixels (which could include the entire image). The matting Equation 2.12 from [LLW06] is an example of such a system. As discussed, a direct solver is typically too time consuming for real-time usage. We instead consider iterative solvers.

To solve the alpha matte, we note that the matting Laplacian, $L$, is typically a positive semi-definite matrix. By definition, this means that, for any vector $x$, the vector $Lx$ is non-negative. This can also be used to imply that $\alpha^T L\alpha$ is a convex function with a unique minimum. Therefore, the derivative's zero-crossing is a global minimum. So the solution to Equation 2.12 is equivalent to the solution of the linear system,

$$L\alpha = \hat{0}. \tag{2.13}$$

We note that both $\hat{1}$ and $\hat{0}$ are solutions to this equation that are uninteresting ($\hat{1}$ is a solution due to the derivation of $L$). These are the solutions that assign all pixels to the foreground, or all to the background, respectively. We instead care to find a solution that respects our constraint vector.

Aside from using the direct solver, we can run either gradient descent or conjugate gradient method to solve this system. The initial vector can be our constraint vector and we can choose to not update elements of the constraint vector iteratively.

### *2.8.1 Gradient Descent and Conjugate Gradient*

Gradient descent can best be viewed as consistently moving in the direction against the function's gradient. In other words, we care to iteratively update our constraint vector with the following function,

$$\alpha' = \alpha - \lambda L\alpha, \tag{2.14}$$

where $L\alpha$ is the gradient and $\lambda$ is a specified step size. This step size can change per iteration, and typically gets smaller as the iterations increase.

A downfall to gradient descent is determining the optimal step size. The conjugate gradient (CG) method solves this issue by providing a dynamic step size that guarantees convergence with a linear number of iterations. The amount is dependent on the number of pixels, which can get large quickly. The specifics of CG are out of the scope of this paper, but we will note implementation and performance differences.

CG requires computing an inner product operator, $u^T Lv$, and magnitude operator, $x^t x$, on vectors that are linear in the number of pixels. These operators are in addition to the standard matrix product required by gradient descent. As a result, an implementation must be capable of computing a reduction efficiently, which typically takes logarithmic processing time on the number of elements when run in parallel. CG typically runs more operations than gradient descent mainly due to the extra operators and reductions. However, the descent rate is adjusted automatically with CG.

These methods are very powerful because of the implied parallelism. A modern GPU can effectively compute $L\alpha$ by implementing a domain specific inner product operation on a specific row of $L$ with $\alpha$. Research such as [HST10, HLW10, GSA05] take advantage of the derivation of the inner product or parallelism to run the CG method in parallel on a modern GPU, but still do not achieve real-time performance.

## 2.9 Binary Segmentation Techniques

Up until now, we've really been addressing matting methods that solve the generic alpha matte problem. As mentioned earlier, transparent surfaces are typically more rare than opaque surfaces. The main other time we observe transparent pixels comes from the camera optics projecting two objects onto one pixel. This ends up with a spatial blurring that results in transparent pixels. So what happens if we know that we only have opaque objects in the scene? The short answer is that we typically don't take advantage of it.

In [RRK10], the authors decided to take advantage of this information. Their matting system estimates the point spread function (PSF) of the camera lens. This information is used to upsample the captured image fairly accurately. The upsampled image (typically a 3x3 grid per pixel) is deemed high enough resolution to run a binary segmentation. The binary segmentation is then downsamples and blurred (using the PSF). The final image is then refined with a typical matting Laplacian.

The method takes advantage of the observation that the scene itself is mainly opaque objects. The processing time is fairly high, but many existing segmentation and matting techniques can be used internally.

## 2.10 Video Matting

As previously mentioned, typical video matting is accomplished using a constant color background. As long as the environment maintains a similar color distribution in the background, the color model can be trained once. The frames are then processed individually.

Another strategy that works in natural images better is to generate trimaps at keyframes. The keyframe trimaps can be used to deduce trimaps for intermediate frames. In [TMW11, SK09], this was done with optical flow. The disadvantage they noted is that not enough pixels are tracked during optical flow and many pixels use a local mixture of Gaussians to compensate.

To alleviate the need of multiple trimaps in a video, the idea to propagate constraint information across multiple frames came about. Techniques such as [BS09, TMW11] do just this. The idea is to use a 3D affinity function that treats adjacent frames as spatially near each other.

The trimap data can then propagate to the current frame through the 3D matting Laplacian. While this increases the algorithmic complexity, it allows key frames to propagate alpha values between frames in a fairly straightforward manner.

## 2.11 Contributions

In the next three chapters, we will discuss our algorithmic contributions to the matting field. Chapter 3 relates to the color-based approaches, as it generates an alpha matte using a trained color distribution. An iterative optimizer smooths the matte using the image graph to smooth the resulting alpha matte. Chapter 4 discusses the use of an external depth sensor to generate a trimap that can be solved using a sample-based solver. Chapter 5 focuses on parallelizing existing matting methods using sub-images.

# CHAPTER 3: INTERACTIVE CHROMA-KEYING FOR MIXED REALITY

As previously mentioned, matting objects by removing key colors in a video signal is a common method used in the film and television industry for special effect composition. In [BZC09], we present a simple method of identifying chroma using GPU hardware. Not only is the process fairly straightforward, it also only relies on a straightforward training phase and two intuitive user parameters.

Our work directly focuses on processing noisy video feeds used in MR. Specifically, we consider chroma keying with a video see-through (VST) head-mounted display (HMD), a wearable device that captures the user's view of the environment through mounted cameras and displays the processed images to the user via screens in front of the eyes [UTS02, SKF05]. The VST-HMD is especially useful in augmented virtuality where real objects must be separated from the regions where virtual objects will appear in the synthesized image, e.g., within portals [HSH05], registered to overlay real objects [FRH08], or providing virtual surrounds as in Figure 3.1.

## 3.1 The Algorithm

In immersive environments that use chroma key matting for visual effect, factors such as lighting, white-balance, and camera-angle are usually under control. As a result, the digitized key color is a persistent, invariant property of the physical material's reflectance projected onto the camera. We wish to exploit this invariant property of the image. Our goal is to find a simple parameterized function that allows a GPU to partition the chroma key colors from the non-chroma key colors, *estimating* the opacity of each pixel for matting. We then minimize the global error of the estimated solution to obtain a plausible alpha matte for user interaction.

To identify the chroma key spectrum, our algorithm executes in three stages. In the off-line *training* stage, the system performs semi-automatic calibration of the chroma key parameterization. In the real-time *classification* stage, the system estimates the alpha matte on a GPU. Finally, the *error minimization* stage improves the estimated matte, accounting for misclassifications and signal noise. Given the resulting matte, standard alpha blending composites our virtual scene with our video feed to create the illusion that both worlds coexist.

### *3.1.1 Training*

To obtain a believable alpha matte based on a video stream containing chroma key pixels, we want to statistically analyze the invariant properties of the chroma key material as digitized by the camera. Since our camera's view of the chroma key must be consistent, we assume that scene lighting and set design are complete and that any camera or driver settings that automatically adjust images, e.g. white balance, are disabled. We then apply a statistical method, namely principle components analysis (PCA) [Smi02], to a subset of the known chroma key pixels.

PCA finds an orthonormal vector basis by performing singular value decomposition on the covariance matrix of a subset of the key color spectrum obtained either during live video acquisition or from processed captured video. We can translate, rotate, and scale an input pixel by the *mean* vector ($\mu$), column-major matrix of *eigenvectors* ($E = [\mathbf{e_1}\ \mathbf{e_2}\ \mathbf{e_3}]$), and diagonal matrix of *eigenvalues* ($\Lambda = diag\left(\lambda_1, \lambda_2, \lambda_3\right)$) calculated by PCA, effectively whitening the input signal,

$$\mathbf{I}'_\mathbf{x} = \Lambda^{-1/2} E^T \left(\mathbf{I_x} - \mu\right). \tag{3.1}$$

This transformation imposes that the Euclidean distance between the transformed pixel, $\mathbf{I}'_\mathbf{x}$, and the origin is approximately the absolute value of the pixel's z-score, provided the chroma key data set is a normal distribution. Furthermore, the projected resulting components are the z-scores with respect to the maximally uncorrelated, orthonormal basis vectors. Notably, the resulting color space will be identical regardless of the input data if presented in some linearly related space, e.g. $RGB$, $XYZ$, $YUV$, or $YC_bC_r$.

### 3.1.2 Classification

Pixels statistically similar to the training set exist within a few units of the origin in the transformed color space. By utilizing simple geometric shapes near the origin, we can quickly estimate the chroma key regions in the input video based on the z-scores of the image pixels. The *default parameters* for these shapes specify inner, assuming the training data is within 3 standard deviations, and outer, assuming the training data is within 4 standard deviations, bounding volumes. To lower misclassifications or enhance alpha blending, an advanced user may choose to set explicit decision boundaries based on z-scores.

Given the decision boundaries and the invariant chroma key description, computing an estimate of the alpha matte is now a simple, inexpensive process that can easily be implemented in real-time using a pixel shader with negligible effect on frame rate and CPU usage. For a given transformed pixel, $\mathbf{I}'_{\mathbf{x}}$, its opacity, $\alpha_x$, is based on its relative distances from the boundaries of the inner and outer regions and clamped in the range $[0,1]$. As a result, any point within the inner region is treated as transparent ($\alpha_x = 0$); any point outside the outer region is opaque ($\alpha_x = 1$); and any other point has opacity based on an interpolation between the inner and outer boundary. We consider two volumes that simply partition data.

#### 3.1.2.1 Bounding Spheres

To directly exploit the isotropic nature of the trained PCA color space, we define two spheres parameterized by their radii and centered at the origin. In an *RGB* color space, these are represented by ellipsoids centered at the mean of the training data scaled along the eigenvectors ( Figure 3.2a). The radius of each sphere represents the number of standard deviations it encloses, and therefore the amount of color within the key color spectrum. To calculate opacity, $\alpha_x$, we use

Euclidean distance from the origin with linear interpolation between the inner and outer radii, $r_{in}$ and $r_{out}$,

$$\alpha_x = \frac{||\mathbf{I}'_\mathbf{x}|| - r_{in}}{r_{out} - r_{in}}. \tag{3.2}$$

### *3.1.2.2 Bounding Cubes*

One drawback of multi-dimensional decision boundaries is that they assume the data follows a convoluted normal distribition; however, the camera response usually produces banana-like shapes. One way to account for this is to project each pixel onto each principle axis and solve each one dimensional problem separately. When these clipping planes are combined, a bounding box is formed that defines the decision boundary of the classifier ( Figure 3.2b). Given an inner and outer axis-aligned bounding cube of sizes $d_{in}$ and $d_{out}$, we use the projected Euclidean distance from each plane to calculate the opacity,

$$\alpha_x = \frac{\max(\mathbf{I}'_\mathbf{x}) - d_{in}}{d_{out} - d_{in}}, \tag{3.3}$$

where $\max(\mathbf{I}'_\mathbf{x})$ is the maximum of the absolute values of $\mathbf{I}'_\mathbf{x}$.

## *3.1.3 Error Minimization*

Since any single-observation chroma-key classifier computes an under-constrained problem [SB96] and current VST-HMDs transmit with high signal noise, there will be error in the alpha matte we obtain from our geometric color classifier. Natural image matting research solves for opacity values in unknown regions of a *trimap*, such as those in [LLW06, SJT04], by solving a Laplace equation constrained by known alpha values. These more general algorithms do not directly take advantage of chroma key information or consider high signal noise. Also, they require a defined trimap, usually manually created by a user.

Our approach, inspired by minimization problems defined in [SM00, LLW04], uses the classifier's output as the constraints, i.e. the trimap. This allows us to solve a least squared error problem based on a useful observation: *similarly colored pixels within a small neighborhood should have similar opacity values.*

$$E(\alpha) = \sum_{\mathbf{I_x} \in I} \sum_{\mathbf{I_y} \in N(\mathbf{I_x})} \nu(x,y)(\alpha_x - \alpha_y)^2, \tag{3.4}$$

where $\nu(x,y)$ is the *affinity* function that relates pixels $I_x$ to the elements of its neighborhood, $\mathbf{I_y} \in N(\mathbf{I_x})$. Pixels classified with strong probabilities ($\alpha_x = 1$ and $\alpha_x = 0$) are constraints. Note that the only mathematical difference of this function from the one in [LLW04] is that we assume the weight vector is normalized, not that the weights sum to unity.

Given two neighboring pixels, we can define the affinity function, $\nu(x,y)$, as in [SM00],

$$\nu_o(x,y) = e^{-\left|\left|\mathbf{I_x} - \mathbf{I_y}\right|\right|^2 / 2\sigma_o^2}. \tag{3.5}$$

In natural image matting, the assumed global color variance, $\sigma_o^2$ is not optimal [LLW06]. However, in our specific domain, a large portion of the variance comes from signal noise, making this affinity function viable. In general, the specified variance should be at least the measured camera noise. In practice, we specify a higher variance to allow local image smoothing.

Given Equation 3.4 and Equation 3.5, gradient descent can now minimize the error in hardware. This is achieved by taking the partial derivative of the error function at each pixel and iteratively updating the alpha matte in parallel,

$$\frac{\partial E(\alpha)}{\partial \alpha_x} = \sum_{\mathbf{I_y} \in N(\mathbf{I_x})} 4 \cdot \nu(x,y)(\alpha_x - \alpha_y). \tag{3.6}$$

In practice, we only solve the pixels with weak probabilities to clip fragments in our constraints. The number of required iterations and neighborhood size depends on the quality of the camera and the size of the video capture. Empirically, our system usually requires fewer than 20 iterations with a neighborhood of size 4 for each captured frame.

## 3.2 Experiments

VST-HMDs enable the interactive MR environment by providing video streams for blending virtual and real objects. Unfortunately, the low resolution and high signal noise produced by these optical devices demand a noise-tolerant chroma key matte algorithm. We demonstrate the robustness of our approach by comparing our method to three known chroma keying solutions.

In our experiments we consider three cameras: a *Canon EOS 30D*, to provide a base-line, a *Canon VH-2002 HMD*, to illustrate our problem domain, and a *Canon VH-2007 HMD*, to compare more recent VST-HMD technology. To reduce the effects of signal noise, we use digital transfers for the EOS 30D and VH-2007. For the VH-2002, we use Osprey-100 capture cards since we must use S-Video connections. Since capture cards may also contribute noise, we consider the cabling and capture card as part of the test camera. Care is taken to synchronize the color temperature and white balance for each camera.

Since camera noise proportionally decreases matte quality, we quantify the camera noise for each camera to find a numerical relation. This is done by producing a digital image of a solid color and displaying it on a LCD monitor in a dark room. Given an input image sequence from each camera, we measure the variance of the image when capturing pure grey. Other test color values, such as black, white, red, green, and blue, do not provide as consistent data for the different cameras.

Because we assume that the key color spectrum's distribution is globally invariant when viewed by a camera, we validate that the video signal may be partitioned using the isotropic PCA color space. We do this by training the PCA matrix on a test scene with each camera. Then, typical chroma key (different than the training data) and non-chroma key frames are plotted in the

PCA-derived color space to show the resulting color distributions.

Next, we evaluate the performance of different chroma keying algorithms with respect to noise. We compare the mattes produced by *Autodesk*® *Combustion*®, van den Bergh and Lalioti's cutting pyramid [BL99a], and our implementation of the ”second Vlahos form” presented in [SB96] to our two decision boundaries defined in isotropic chroma key space. We chose the Vlahos' second form because it is easier to tweak variables under noise. In order to visualize the key color spectrum, we generate mattes from color diagrams containing discretized values throughout the displayable color spectrum.

Finally, to show that the error minimization step produces more robust mattes, we evaluate the technique on one of the classifiers. We visualize the minimizer running to observe convergence relative to the number of iterations taken. We also measure the effect this minimization step has on frame-rate in an environment with virtual surround using a modern graphics card.

## 3.3 Results

As seen in Figure 3.1, our keying algorithm performs well on the relatively noisy images from the Canon VH-2002 VST-HMD. This image was taken against a chroma key green screen and demonstrates that our algorithm is robust enough to properly matte a low quality image commonly seen in MR. Our method handles these imperfections well due to the trained approximate PCA chroma key space enabling the use of simple decision boundaries to classify the colors and our error minimization. There are, however, haloing artifacts and chroma-spill, areas that we have not addressed and leave for future work.

In addition, camera noise also requires a robust classifying algorithm to handle variations from the key color spectrum when VST-HMD camera optics are less than optimal. As seen in Figure 3.3, the VST-HMD cameras used for MR experiences contain significant noise. For instance, VH-2002's standard deviation from the grey color suggests that many chroma keyers will improperly classify a large portion of the matte, e.g. Vlahos's method. This shows that chroma keying is currently difficult in MR environments due to the limitations of the VST-HMD technology that enables the experience. We see a substantial improvement with the VH-2007, but still see more signal noise than the relatively older EOS 30D.

Problems may also be introduced by environmental conditions. For instance, folds in a chroma key curtain or insufficient lighting can introduce shading that is not related to empirically derived algorithms or the objects in the real world. To illustrate the effectiveness of the PCA transformation, Figure 3.4 plots the color distributions as viewed by our classifiers in multiple frames captured during a typical MR scenario. The two shapes separate the resulting single-dimensional space into chroma key and non-chroma key colors.

When inspecting the mattes produced from still images in  Figure 3.5, we notice that the cutting pyramid and Ultimatte® simply cut the color space on the blue boundary. This can be seen by triangular shapes in the color diagram mattes. These cuts imply that the decision volume contains considerably more of the color spectrum than necessary for keying, which may lead to errors when introducing more colors into the image, specifically darker colors like those found in shadows. Both of our algorithms occupy a much smaller volume, showing that our approach greatly reduces the key color spectrum necessary for matting.

Figure 3.6 qualitatively compares the different composited scenes to illustrate the improvement realized with the VH-2007, but shows that the EOS 30D still produces less noise than either of the VST-HMDs. Otherwise, we see less visual noise than observed when compositing a scene with either our algorithm or the off-line Combustion® result ( Figure 3.5).

The small percentage of pixels that classifiers cannot accurately solve causes visual noise. To address this, the GPU minimizes the error with gradient descent when new video frames arrive. This usually stabilizes within 20 iterations, each requiring 5 texture lookups, as seen in Figure 3.7. In our experiments, ample time is available for virtual scene rendering and compositing. As seen in Table 3.1, the PCA-based sphere classifier has almost no performance hit. Similar frame rates apply to the box classifier. When considering an extreme case of 40 iterations for error minimization, we see about a 35% hit on framerate. In the more common case of 20 iterations, we see a 23% hit on framerate, which is acceptable when operating with graphics cards like the Nvidia Geforce GTX 280 used in the experiment.

| | With Virtual Scene | Without Virtual Scene |
|---|---|---|
| No Keying | 186 | 2825 |
| Keying With No Iterations | 184 | 2770 |
| Keying With 20 Iterations | 144 | 1966 |
| Keying With 40 Iterations | 120 | 1609 |

Table 3.1: Framerate of our sphere classifier with and without error minimization using a neighborhood of size 4. Notice that the classifier itself has neglible effect on framerate. The error minimization contributes about a 23-30% and 35-44% hit on the framerate, depending on the number of iterations.

## 3.4 Alpha Matte Smoothing

In the initial research [BZC09], we considered minimizing a global error function that smoothed the alpha matte using gradient descent. Minimizing similar error functions using the conjugate gradient method has become a common practice in natural image matting. We note that our error function is not as mathematically robust in comparison. We also note that minimizing such functions in real-time has not been achieved yet, even on a GPU (although it is close). Since the time of publication, the error function has been replaced with a local smoothing operator, which is defined below.

The idea is borrowed from related research in depth map correction, where bilateral filters are used to approximate higher resolution depth maps. We decided to smooth the resulting alpha matte using a similar approach. The filter weights are computed from the color image and used to update the alpha matte. Pixels that were original in the known constraints are not modified, so processing is limited to pixels that had z-scores in the less confident ranges. It is worth noting that this change also came after the acquisition and use of newer, less noisy optics.

The alpha matte is iteratively updated using the following function,

$$\alpha'_x = \frac{\sum_{y \in N(x)} v_o(x,y)\alpha_y}{\sum_{y \in N(x)} v_o(x,y)}. \tag{3.7}$$

where $v_o(x,y)$ is defined using a Gaussian on the color distance as in the original paper. We note that we do not process the spatial distance in the bilateral filter, as we only consider a 4-connected neighborhood. Each pixel also inserts itself into its set of neighbors.

To compare this to the smoothing function in the original paper [BZC09], we rewrite the gradient descent equation, where $c_k$ is the gradient descent rate,

$$\begin{aligned} \alpha'_x &= \alpha_x - c_k \sum_y 4 v_o(x,y)(\alpha_x - \alpha_y) \\ &= \left(1 - \sum_y 4 c_k v_o(x,y)\right) \alpha_x + \sum_y 4 c_k v_o(x,y) \alpha_y. \end{aligned} \tag{3.8}$$

We note that if $\sum_y 4 c_k v_o(x,y)$ is less than one, the weight of $\alpha_x$ is positive and less than one. For a single pixel, given a gradient descent rate of $c_k = \frac{1}{\sum_y 4 v_o(x,y)}$, these solutions are identical. Using the bilateral filter smoothing, we allow the gradient descent rate to adjust per pixel in an intuitive manner.

## 3.5 Conclusions

Our GPU-amenable chroma keying method provides a robust and affordable method for matting in Mixed Reality with negligible effect on frame rate and CPU usage. The use of the isotropic chroma key space, defined by PCA, enables us to employ simple geometric shapes to construct the decision boundaries surrounding the key color spectrum with only two intuitive user parameters. Furthermore, the classifier implementation is straightforward on a programmable GPU. This results in matte quality comparable to commercial offline solutions but with a smaller key color spectrum to improve the tolerance of the chroma key algorithm with respect to noise.

A downside to the chromakey algorithm is that it does not adapt to environment changes very well. The training algorithm typically runs very fast, but requires capturing a pure background. To address this, we looked for methods that solved the alpha matte without depending on a color distribution. This led to our work with a depth prior in following research described in the next chapter.

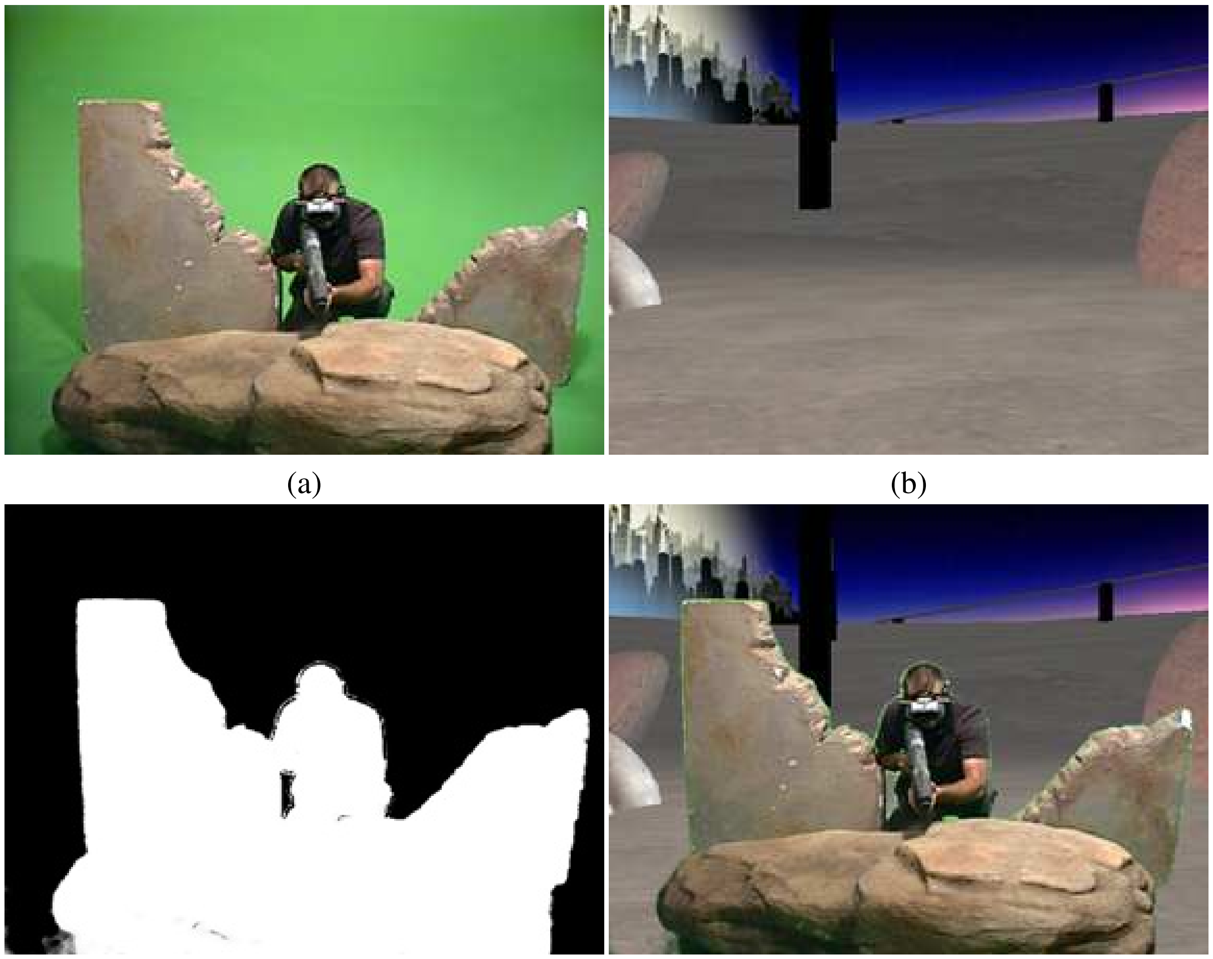
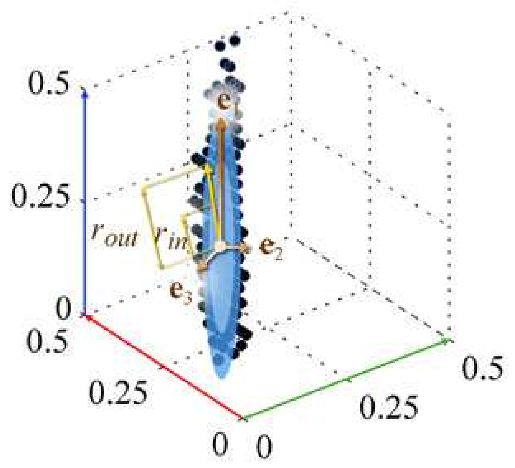
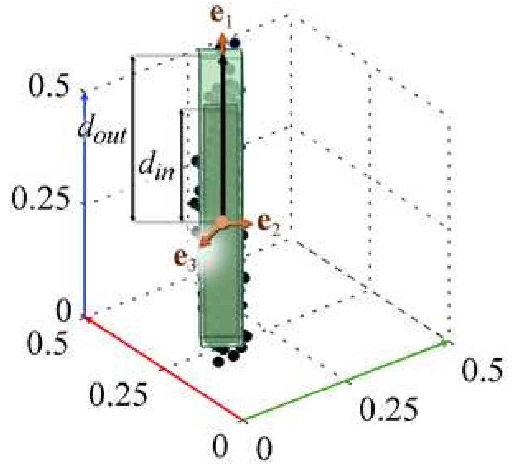

(a) (b) (c) (d)

Figure 3.1: Demonstration of the matte fidelity with our PCA-based chroma key matting algorithm using a VST-HMD with significant signal noise. (a) Original shot with the green screen background. (b) A virtual backdrop. (c) Matte pulled by our algorithm. (d) Composited result.



(a)



(b)

Figure 3.2: Decision boundaries for the chroma keys in the uncorrelated chroma key color space, illustrated by the scaled eigenvectors $\mathbf{e_1}$, $\mathbf{e_2}$, and $\mathbf{e_3}$ from PCA, defined by simple geometric shapes: (a) bounding spheres and (b) bounding cubes. The opacity is determined by either the radii, $r_{out}$ and $r_{in}$, or the distances, $d_{out}$ and $d_{in}$, to the respective boundaries.

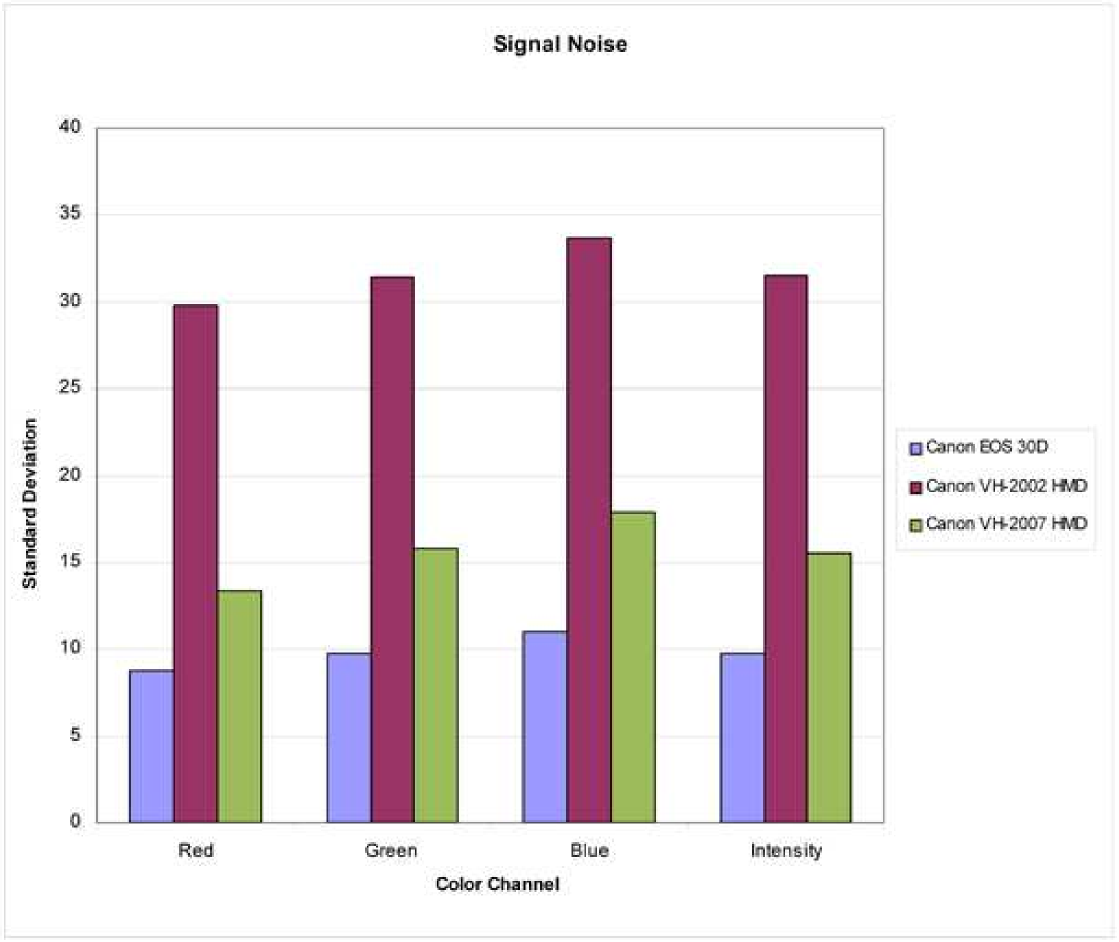


Figure 3.3: When the cameras record a solid color image, we can see that the Canon HMD has significantly higher signal noise. The vertical axis is the number of standard deviations in the data when constant grey is captured.

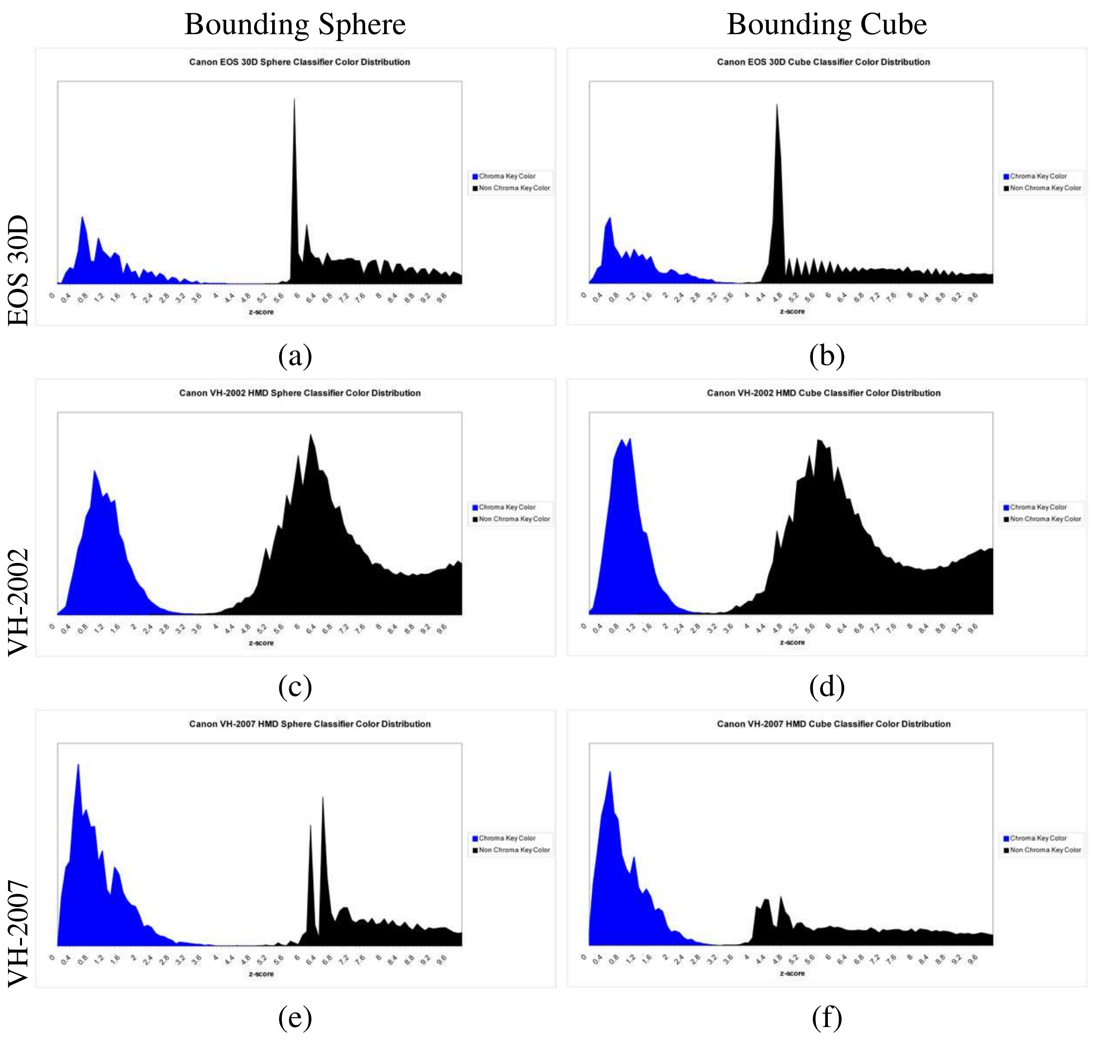


Figure 3.4: The pixel distributions observed by our classifiers when capturing a test scene from each camera. (a)/(b) show the distribution for each classifier when using a relatively low signal noise Canon EOS 30D. (c)/(d) and (e)/(f) similarly show the pixel distribution when using a noisy Canon VH-2002 VST-HMD and the newer Canon VH-2007 VST-HMD, respectively. Note that the majority of the data points are separable by both classifiers.

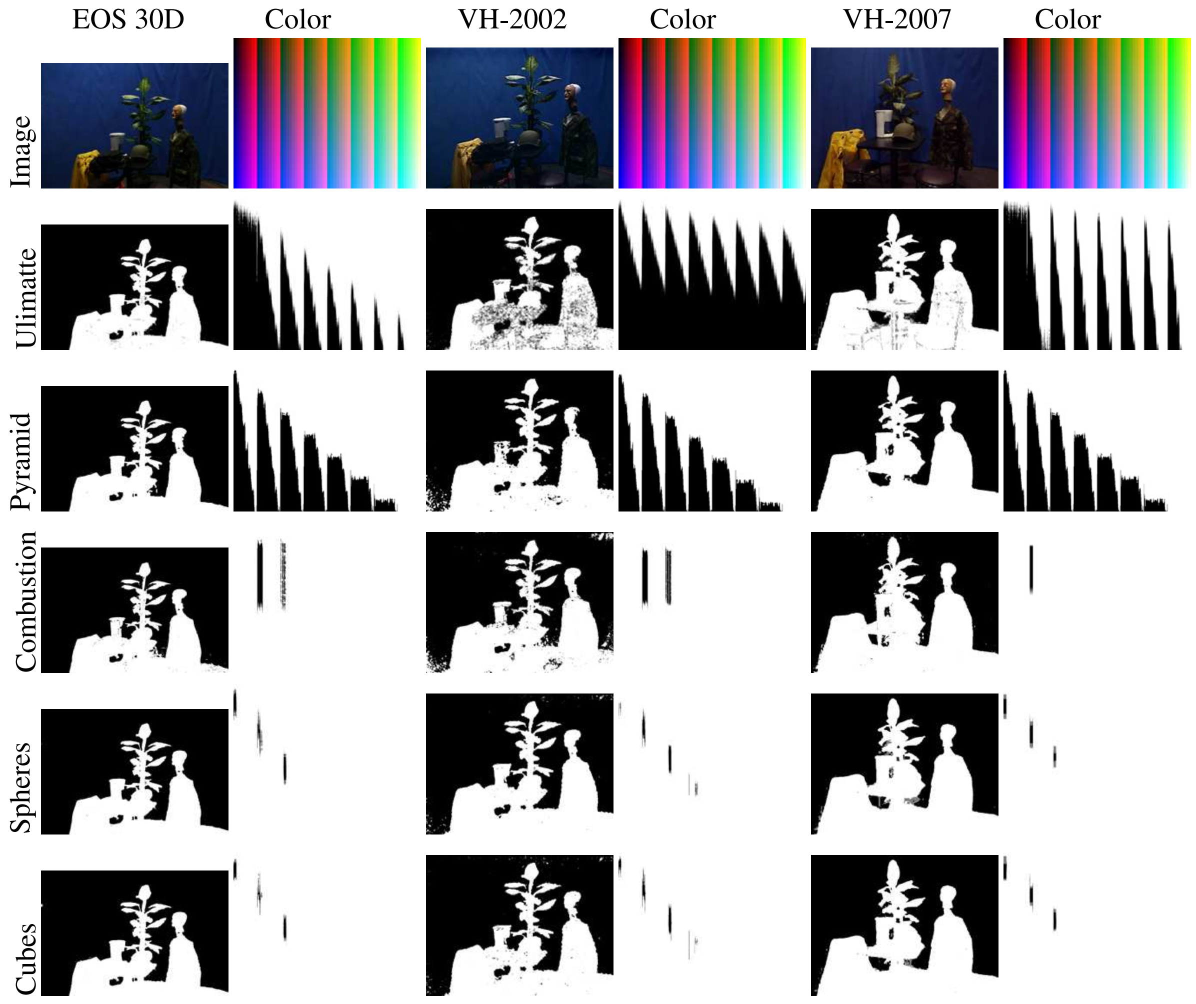
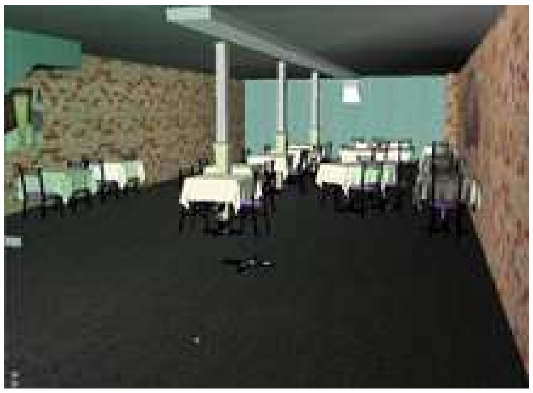
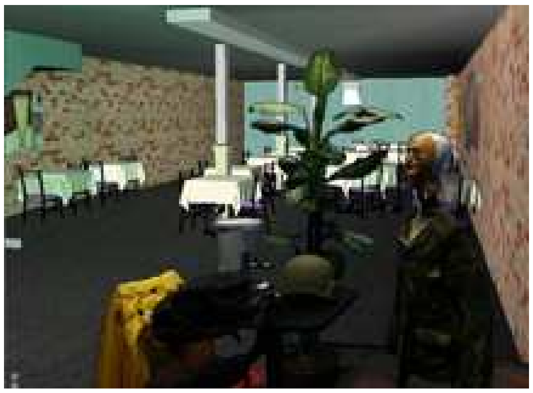


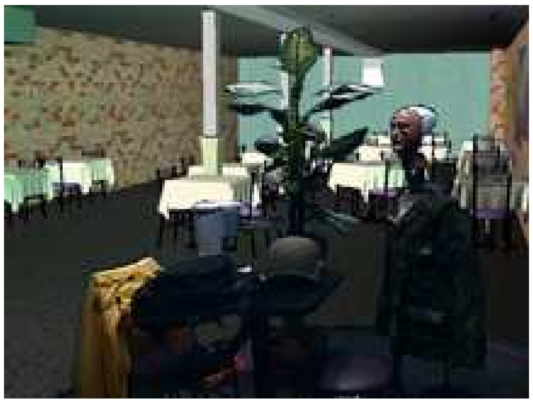
Figure 3.5: Qualitative analysis of various chroma key algorithms from two camera sources. For each chroma key algorithm and camera, an image of the matte appears on the left. On the right, the approximate key color spectrum illustrated as the matte of our color diagram. Our method obtains a matte that is as effective as other commercial techniques while reducing the key color spectrum necessary for matting.



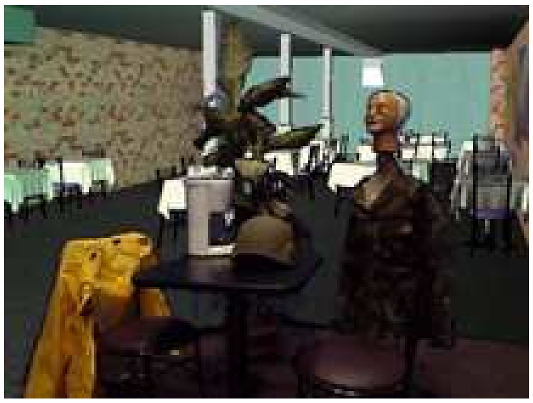
(a)



(b)



(c)



(d)

Figure 3.6: The results of compositing the scene in Figure 3.5. (a) is a virtual backdrop. (b) from the EOS 30D, (c) from the VH-2002 HMD, and (d) from the VH-2007 HMD show the composite results using the sphere method with no error minimization.

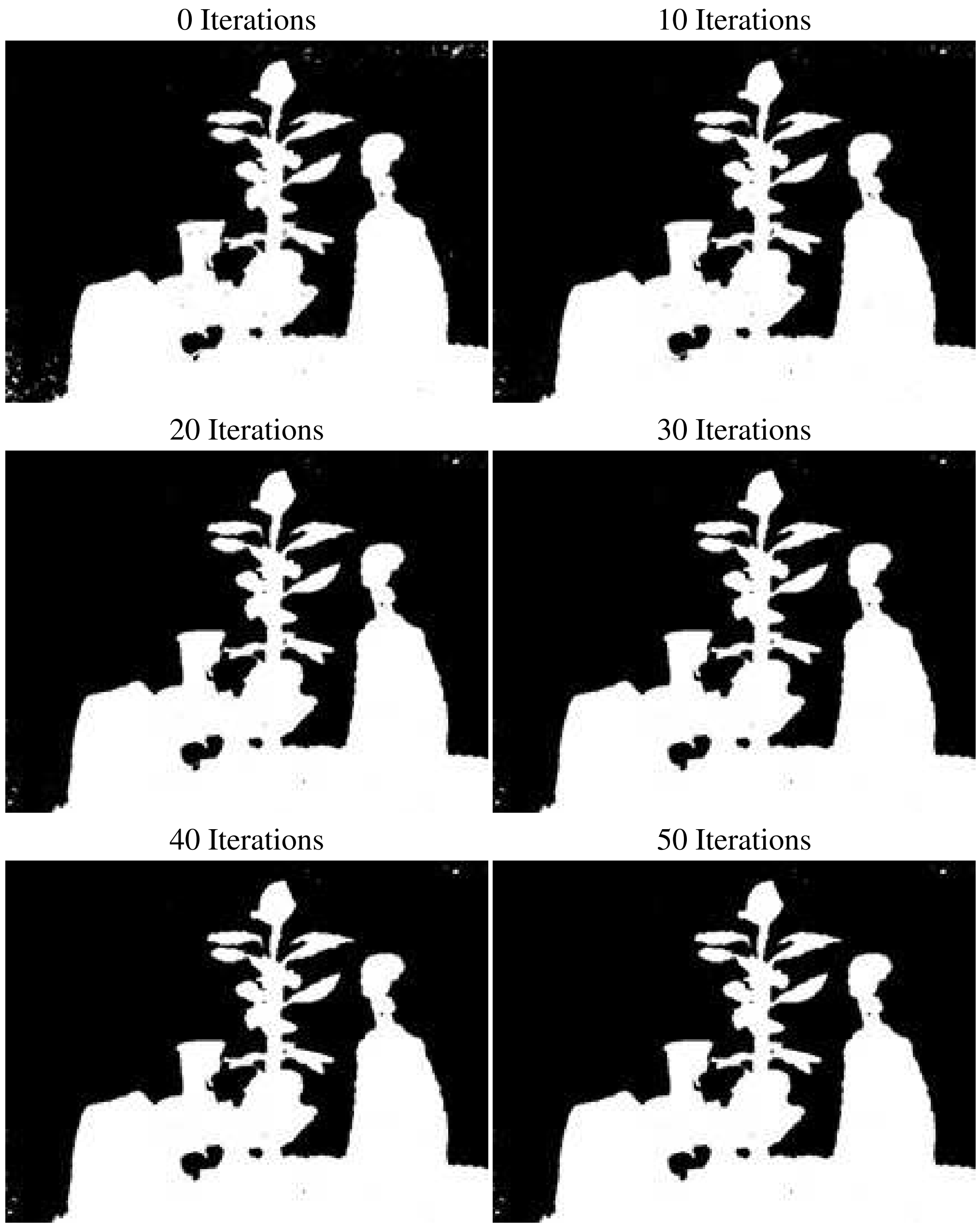


Figure 3.7: Qualitative analysis of error minimization applied to our bounding sphere classifier. Here, our $\sigma_o^2$ is 0.025, our gradient descent rate is 0.05, and our neighborhood size is 4. Notice that most of the unknown pixels are resolved within 20 iterations.

# CHAPTER 4: REAL-TIME VIDEO MATTING FOR MIXED REALITY USING DEPTH GENERATED TRIMAPS

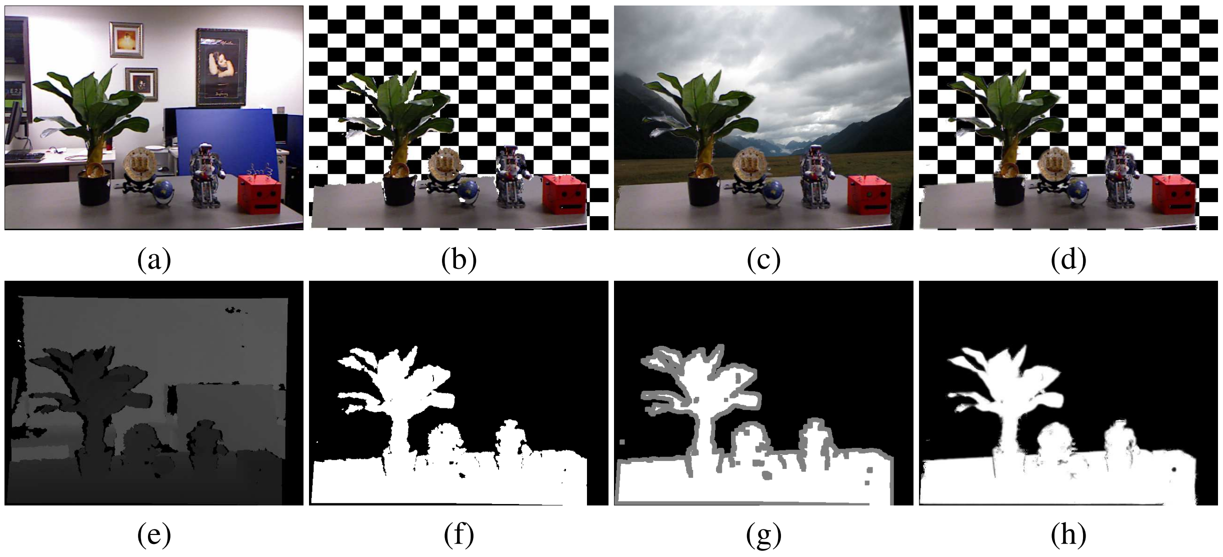


Figure 4.1: Demonstration of the steps in the matting process using a Microsoft Kinect. The input is the color image (a) and depth image (e). A depth threshold is applied to create a segmented image (f), which is both eroded and dilated to create a trimap (g). A natural image matter solves the alpha 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).

For Mixed Reality applications, we would like to have a real-time matting solution that allows *dynamic camera motion*, is *tolerant to environmental changes* (such as lighting), and works in *environments that may not contain constant color backgrounds*.

## 4.1 Matting with a Depth Prior

To address these requirements, we consider exterior data as in [GKO03, WFY07]. 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 [WFY07]. It is then possible to execute a natural image matting solver in real-time on a live video, given the trimap [GO10]. 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.

Creating an alpha matte in real-time is accomplished by utilizing the parallelism of a modern GPU. We move the captured depth map (Figure 4.1e) and color image (Figure 4.1a) 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 [WFY07]. 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.

### *4.1.1 Trimap Generation*

To generate trimaps, we follow a few steps outlined in [WFY07]. 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 [WFY07]. 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 properties and thus are well-defined in the depth image. As a result, pixels with unknown depth are assigned maximum 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.

### *4.1.2 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 [WFY07], 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 [GO10]. 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 [GO10] 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.

## 4.2 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. We try to target a GPU architecture for efficiency.

### *4.2.1 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.

### *4.2.2 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 [CCS01],

$$P(F_p, B_p, \tilde{\alpha} | I_p) = \frac{P(I_p | F_p, B_p, \tilde{\alpha}) P(F_p) P(B_p) P(\tilde{\alpha})}{P(I_p)}, \tag{4.1}$$

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 $B_p$,

$$\tilde{\alpha}(I_p, F_p, B_p) = \frac{(I_p - B_p) \cdot (F_p - B_p)}{||F_p - B_p||^2}. \tag{4.2}$$

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 [CCS01] 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,

$$P(F_q, B_q, \tilde{\alpha} | I_p) = P(I_p | F_q, B_q, \tilde{\alpha}) P(\tilde{\alpha}) P_k, \tag{4.3}$$

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 [GO10],

$$
\begin{aligned}
F_p^r &= \frac{\sum_{q\in N[p]} P(F_q, B_q, \tilde{\alpha} | I_p) F_q}{\sum_{q\in N[p]} P(F_q, B_q, \tilde{\alpha} | I_p)} \\
&= \frac{\sum_{q\in N[p]} P(I_p | F_q, B_q, \tilde{\alpha}) P(\tilde{\alpha}) F_q}{\sum_{q\in N[p]} P(I_p | F_q, B_q, \tilde{\alpha}) P(\tilde{\alpha})},
\end{aligned} \tag{4.4}
$$

and similarly,

$$
B_p^r = \frac{\sum_{q\in N[p]} P(I_p | F_q, B_q, \tilde{\alpha}) P(\tilde{\alpha}) B_q}{\sum_{q\in N[p]} P(I_p | F_q, B_q, \tilde{\alpha}) P(\tilde{\alpha})}. \tag{4.5}
$$

For clarity, we define the probability as done in [CCS01]. 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,

$$
P(I_p | F_p, B_p, \tilde{\alpha_p}) = exp(-||I_p - \tilde{I_p}||^2 / 2\sigma_C^2), \tag{4.6}
$$

where $\tilde{I_p} = \tilde{\alpha_p} F_p + (1 - \tilde{\alpha_p}) B_p$. $\sigma_C$ 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.

$$
P(\tilde{\alpha}) = 1.0 - exp(-(\tilde{\alpha} - 0.5)^2 / 2\sigma_{\tilde{\alpha}}), \tag{4.7}
$$

$\sigma_{\tilde{\alpha}}$ 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.

### *4.2.3 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 great deal 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 4.2. 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, [GO10] 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.

### *4.2.4 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.

## 4.3 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* [RRW09]. 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 4.2.4 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 4.1.1. 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.

## 4.4 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 4.1). 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 4.1).

| | Shared Matting | Modified | Filtered |
|---|---|---|---|
| | Error | | |
| Average MSE | 0.0040 | 0.0043 | 0.0050 |
| Average SAD (x$10^3$) | 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 |

Table 4.1: 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.

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 in Figure 4.2a. 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 4.2d shows a frame from a different video taken with the same camera.

To test the temporal modifications from Section 4.2.4, 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.

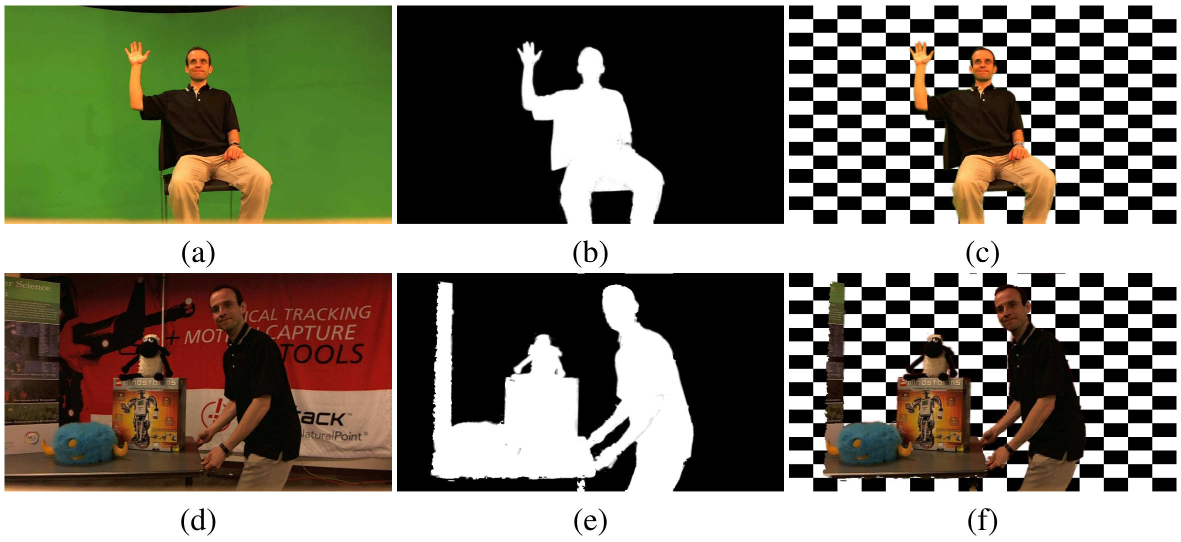


Figure 4.2: Subfigures (a) and (d) show selected color frames from two of our video sequences. The calculated alpha mattes are pictured in (b) and (e). (c) and (f) exhibit the results with replaced backgrounds. Note that the missing left sides in (e) and (f) are not due to an error in the algorithm but result from the different aspect ratios of the color and depth cameras. Image (a) is a selected frame from the second experiment.

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 estimated 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 algorithm 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 4.3 shows frames with a resolution of 640x480 pixel from that video. Table 4.2 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.

| 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 |

Table 4.2: An example 640x480 video is processed (over 450 frames) in real-time. Selected frames appear in Figure 4.3.

We also observe some graininess in the alpha matte, which we believe is due to the camera noise. As mentioned in [WFY07], 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 4.3a-b.

## 4.5 Conclusions

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 [GO10], 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 ample 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 [WFY07], the choice of a constant depth slice is problematic when the background and foreground interact. We do not address this here. Our attempts to find a modification to the Shared Sampling algorithm that is more resilient to trimap noise in videos led us to the idea of dividing the video into sub-image streams. In the next chapter, we attempt to reduce the number of pixels required to solve a portion of the image in hopes of improving performance and enabling processing of larger resolution images.

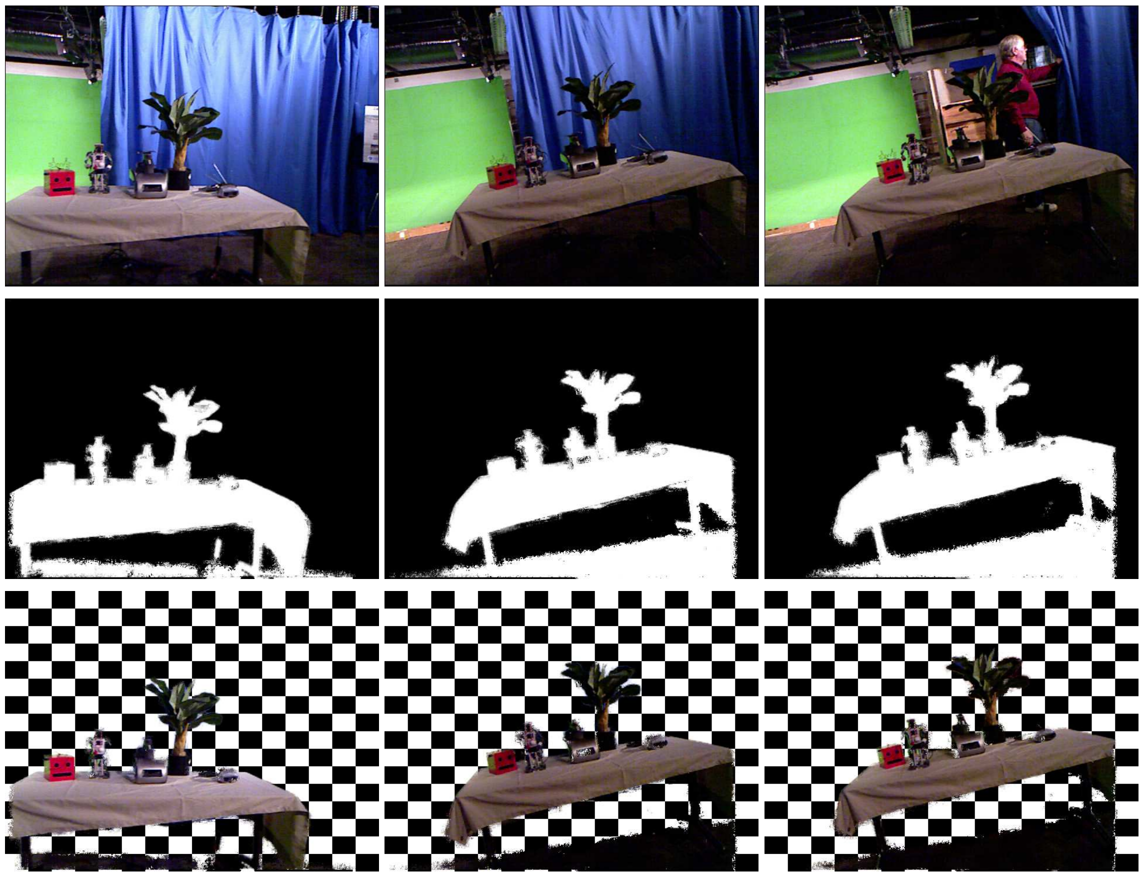

(a) (b) (c)

Figure 4.3: 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).

# CHAPTER 5: PARALLEL MATTING OF LARGE RESOLUTION IMAGES USING SUB-IMAGES

One of the inherit issues with global matting methods is the size of the data representation, such as exists with the matting Laplacian, and the difficulty to efficiently solve the resulting system. Methods such as conjugate gradient could potentially take millions of iterations for relatively small (megapixel) images before guaranteeing convergence. Hierarchical approaches are typically used to improve conjugate gradient performance [LLW06]. These approaches provide an alpha matte by propagating lower resolution solutions to the higher resolution image. While effective at improving compute efficiency, the memory requirements to solve a large resolution image are ignored, limiting the image resolution drammatically. Also, such a strategy can over-smooth the resulting matte, effectively losing high frequency details [HST10, HLW10]. Solving a different scaled version of the image [HST10] or considering local sub-images as relatively independent units [HLW10] may address the loss of detail.

The argument for a hierarchical approach relies on the observation that the final matte will converge quicker if information relatively far away in pixel space may contribute earlier [HST10, GO10]. The window based strategy observes that most alpha solutions are computed using local pixel information and that pixels far away do not significantly contribute to the final result [HLW10]. These ideas seem to clash, but make sense if you consider opaque objects and the color consistency assumptions presented in [GO10]. It is worth noting that, for Mixed Reality scenarios that typically deal with opaque objects, the risk of losing high frequency data is not prevalent.

For the fine-grained solution of [HLW10], a local window scheme solves windows with the most known pixels and iteratively computes the final matte. Unfortunately, this strategy requires a high level serialization mechanism so that multiple sub-images do not produce conflicting results in parallel. We see this as a unnecessary drawback, especially when compute power is likely available. It would be beneficial if results from independent high frequency sub-images were combinable.

As a consequence, we can pose two questions: *What are the optimal window locations?* And, *What strategy can be used to combine the matte computed per window?* By considering the windows' locations independent of the global solution and the local solutions independent of the window ordering, we effectively parallelize the underlying matting strategy. We note that the matting algorithm itself is not affected by this approach. The choice of sub-image locations, sub-image sizes, and solution aggregation becomes an issue since higher numbers of sub-images per pixel result in more compute time to find the final alpha matte.

## 5.1 Sub-Image Matting

In order to solve a large resolution alpha matte, such as the example in Figure 5.1c, we want to solve several localized problems within sub-images. For a particular pixel, $I_x$, we would like a system with the following properties:

$$\alpha_x = \sum_i w^i(x)\alpha_x^i \tag{5.1}$$

$$1 = \sum_i w^i(x) \forall x \in I \tag{5.2}$$

In other words, we want to find a weight function, $w^i$ for each sub-image $S^i$ such that the weights of all sub-image solutions sum to one. This means that at least one sub-image solution must contribute to the overall alpha matte.

The trivial implementation would be to segregate the image into disjoint sub-images, $S^i$, and compute all solutions independently. While the sub-images are theoretically any disjoint subsets, a straightforward approach would be to use a regular grid on the image. This approach satisfies the above equations using the function,

$$w^i(x) = \begin{cases} 1 & \text{if } x \in S^i \\ 0 & \text{otherwise} \end{cases}. \tag{5.3}$$

This technique will lead to alpha matte discontinuity issues in more complex images, giving a need to solve each pixel in possibly two or more sub-images. On the other hand, we could brute force a solution and center a window on each pixel, allowing hundreds of alpha matte solutions to contribute to the alpha value of a given pixel. This technique is exhaustive and would increase the computational complexity with the square of the sub-image resolution. This is undesirable. We instead take a more conservative approach and overlap windows as little as possible to keep the computational complexity within a constant factor. The method is described in the following steps: constraint initialization, window specifications, sub-image matting, and result aggregation. These steps may be applied hierarchically.

Consult Figure 5.2 for a visual example. The selected image region is broken up into several overlapping 256x256 sub-images. Four are explicitly highlighted. The alpha matte from each is merged together to form a unique 128x128 alpha matte for the overlapping region.

### *5.1.1 Constraint Initialization*

The constraints are used to provide guidance to the solver. At a high level, we receive constraints from a trimap. In Mixed Reality scenarios, we typically have opaque objects and trimaps automatically extracted from a constant color background or a depth map. Using a trimap, a hierarchical approach should initialize approximate alpha values more than preserve high frequency data. We produce additional constraints by down-sampling the input constraints, solving the lower resolution alpha matte, up-sampling the alpha matte, and applying dilation and erosion [LLW06]. A hard threshold is applied to prior level's result to produce a new trimap from the current level's solver. At the lowest level, the down-sampled input constraints are used as provided.

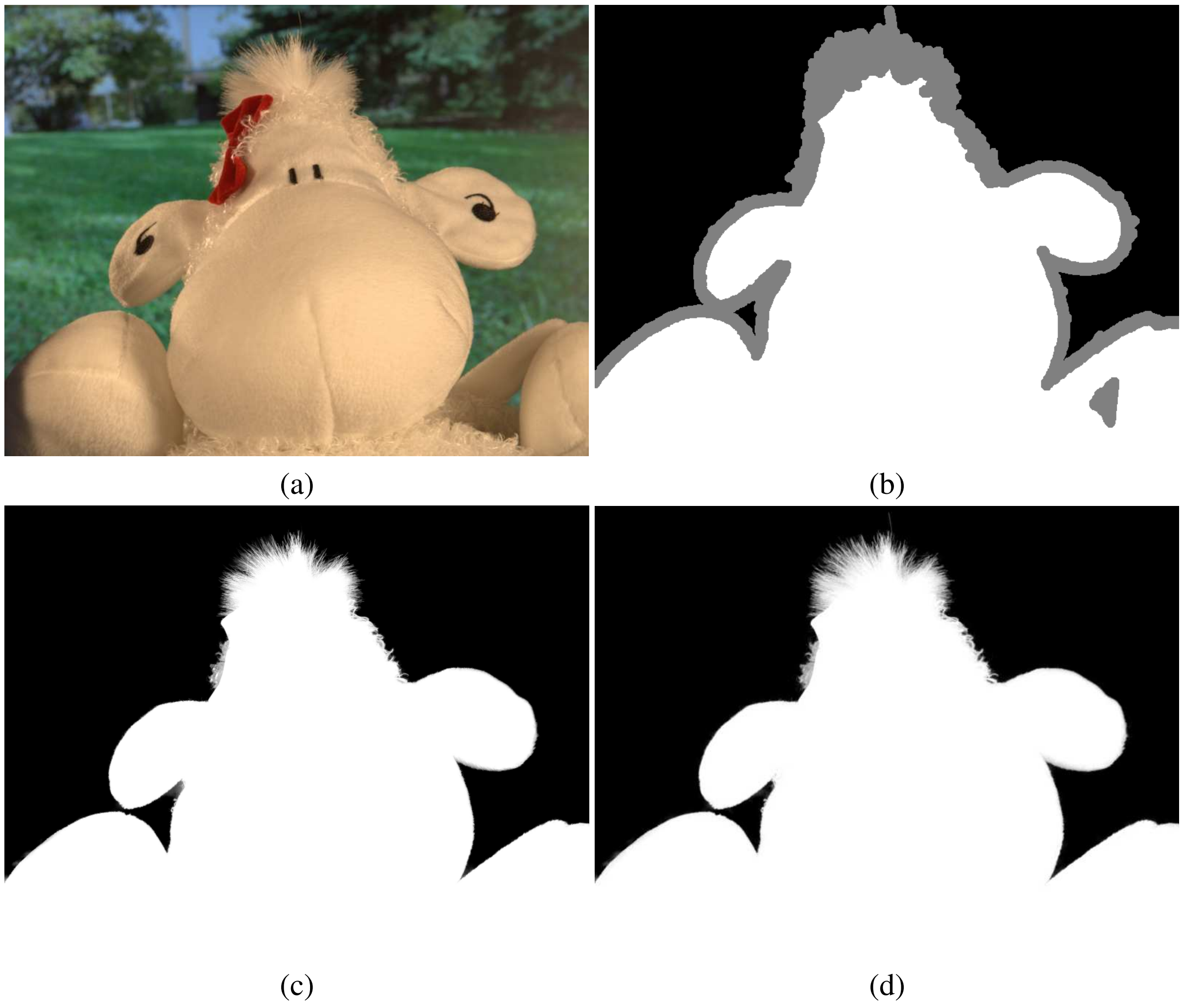


Figure 5.1: A comparison of a high resolution alpha matte (c) obtained via our method and the low resolution alpha matte (d) from a previous work. (a) and (b) are the input and trimap, respectively. We note the high frequency information (such as the long, single strand of hair on the top of the head) may be compromised. Aside from this, the sub-image solver works surprisingly well, suggesting that locality may be exploited to solve trimaps.

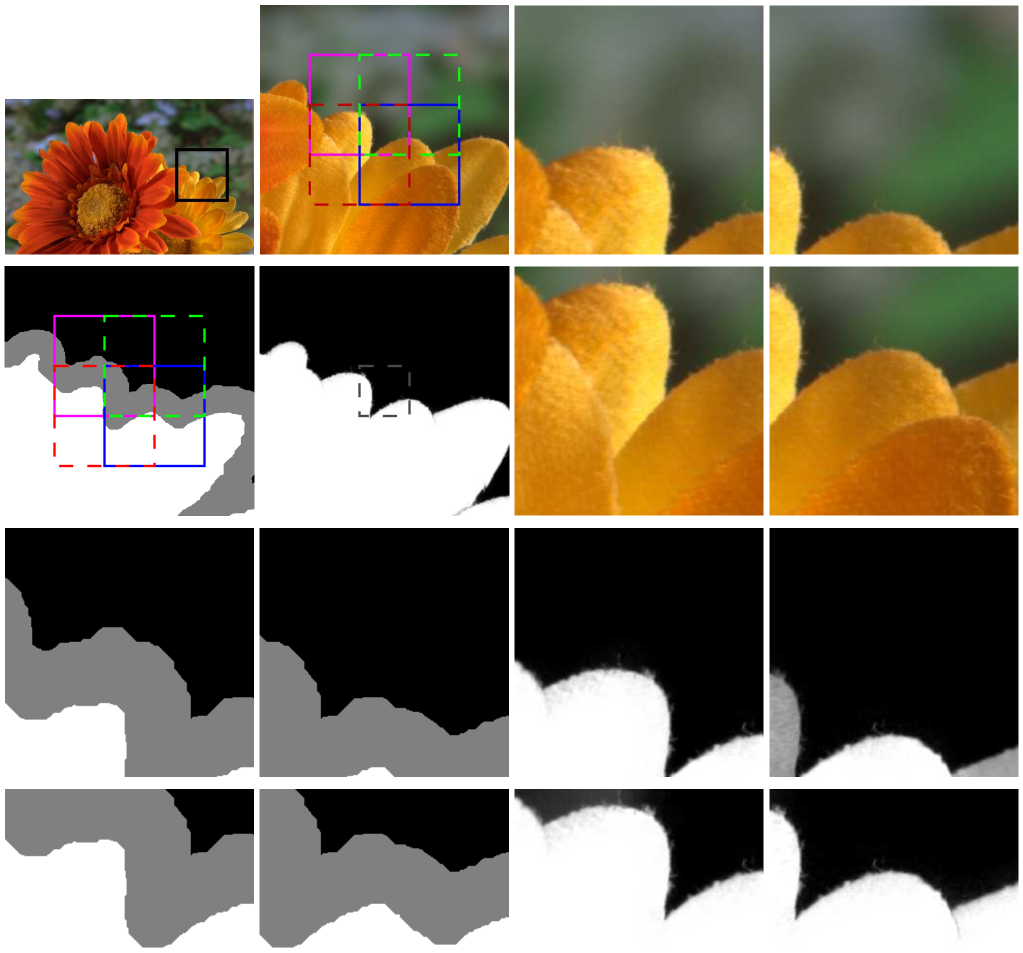


Figure 5.2: An example of how the windowing algorithm works. Four overlapping 256x256 sub-images compute local alpha mattes using only data within the sub-image. The solutions are bilinearly blended to produce the final alpha matte. For this example, the four selected sub-images uniquely create a 128x128 solution, which is not influenced by any other sub-images.

### *5.1.2 Sub-image Specification*

At a particular sub-sampled image, we solve an alpha matte by breaking the image into overlapping sub-images using a series of grids. As mentioned earlier, disjoint sub-images produce artifacts on sub-image borders. We instead create four grids with cells centered at points modulo half of the grid size. As a consequence, all pixels exist in four distinct sub-images. We chose a grid since it has benefits of aligning to image data structures. A grid can be viewed the same as solving each pixel with a sub-image kernel, but pruning the total number of kernels to reduce excessively redundant computation.

Each sub-image's alpha matte is solved independently using only pixel data contained in the corresponding sub-image. Sub-images with no unknown pixels can be ignored. The sub-image size should be large enough to hard constrain a realistic solution, but small enough to enable better compute and memory performance. The sub-image size could be adjusted with the current image resolution. However, since we aim to support large resolutions, reducing the window size has the potential to throw away data by prematurely making the window too small when reducing several iterations. Our experiments explore several sub-image sizes to get an idea of what sizes seem reasonable.

### *5.1.3 Subimage Matting*

Given a specific sub-image at a known resolution (as well as the trimap for said sub-image), any matting method may be used to compute an alpha matte. We chose to use the Closed Form matting solution presented in [LLW06] due to its influence on the matting community. We note that it has limits with respect to the input image dimensions due to the Laplacian matrix size. The Laplacian solution is computed using both the original method and a conjugate gradient solver for comparison. For more exhaustive testing, we relied on the former, as it was typically more accurate and quicker.

### *5.1.4 Result Aggregation*

Once we have alpha mattes for each of the sub-images, we need to merge the matte solutions together. In most cases, the results should be nearly identical, in which any linear blending, such as averaging with $w^i(x) = 0.25$, can produce visually smooth results. Within a single sub-image, pixels near subregion borders still have a chance of biasing a solution towards a bad result due to lack of hard constraints near it. To counter this issue, we bias our linear blend towards solutions near the center of a window. This can be achieved by using a bilinear distance weighting function with respect to the sub-image centers. Note that the distance within a sub-image is constant, thus our weighting function is:

$$w^i(x) = \frac{\text{—row}(x) - \text{row}(i)|}{0.5|S|} * \frac{\text{—column}(x) - \text{column—}(i)}{0.5|S|}, \tag{5.4}$$

which is just the linear distance divided by the sub-image size in both dimensions independent.

The above weight function can be precomputed in sub-image pixel space with an outer product on a linearly interpolated vector:

$$W' = D^T \cdot D, \tag{5.5}$$

where D is a row vector [0...11...0] of size $|S|$ containing $|S|/2$ linearly interpolated values from 0 to 1 then 1 to 0. This entire matrix may be pair-wise multiplied on the resulting sub-image matte and accumulated in the resulting alpha matte's buffer.

In the extreme case, window results may vastly differ, such as some windows classifying a pixel as background, and some as foreground. This is especially true when a window does not have enough data to determine and propagate a foreground and background consistently. In such situations, we note that there is typically not enough local information to properly classify a pixel (or, in some cases, the result is locally ambiguous). We rely on the hierarchical iterations to resolve these cases, as the linear blending will cause the solution to be unknown in the resulting matte.

## 5.2 Experiments

As our algorithm does not actually rely on a matte computation method, we choose to look more into the effect that locality has on the matting solutions. We pay attention to window size, number of reductions for noisy solutions, and cases where the information is not detailed enough to solve the matte. We note that we are particularly interested in dense trimaps with mostly known solutions, as this is typical in Mixed Reality environments. Finally, we apply the windowed solver to a high resolution image to see what the results are.

For both experiments, we run the entire training data set from *www.alphamatting.com* [RRW09]. We did not optimize any parameters prior to running the experiments. All experiments are implemented in Matlab and not optimized for speed. They were run a on 1.6 Ghz Intel Core i7 CPU.

### *5.2.1 Non-hierarchical*

The first experiment uses a non-hierarchical approach to see the effect of local solvers. We break the image into sub-images and solve the resulting matting Laplacian using both the Matlab backslash operator and conjugate gradient. We test window sizes in powers of two between 16 and 512. The lower bound was chosen as it produces numeric instability. The upper bound was chosen due to the dimensions of the training data.

For the first implementation, the constraint vector is weighted by 100. For the latter, we run conjugate gradient for 1000 iterations (arbitrary choice for proof of concept). The resulting alpha mattes are compared against computing the global matting Laplacian and solving the whole system. We expect to see a much higher run-time cost due to the overlapping regions, but hope to see acceptable error rates.

### *5.2.2 Hierarchical*

In the second experiment, we want to observe the effects that the number of image reductions has on the final result. We reduce the image a varying number of times during matting solving, typically 1 through 5. We view the effects that the hierarchical scheme has on both matting accuracy and run time.

All experiments of this nature use the Matlab backslash operator to solve the matting Laplacian within a window. The window size is not reduced as the image is reduced. We expect the run time to decrease in proportion to reductions in the number of iterations. However, for larger window sizes, there may be a trade off due to the total number of windows processed vs. the complexity of the underlying solution.

## 5.3 Results

As mentioned earlier, we observe errors in the alpha matte due to a sub-image not containing enough priors. Figure 5.3 shows this phenomena occurring when we use a 256x256 window to process a test image. As shown in the highlighted region, a gradient artifact occurs. When inspecting the corresponding trimaps for the four windows that computed solutions here, three of these windows are underspecified. The matting solution naturally biases the answer towards the existing prior, causing heavy weighting towards foreground or background. When merged using the bilinear weighting function, we see these gradient artifacts. It is worth noting that this particular error does not show itself when adding just a single reduction to the hierarchical solver or when using a window of size 512x512.

With respect to run-time, we did not actually implement a parallel processing solution to determine benefits from parallel computation. From our experience, a typical sub-image of 256x256 can be solved in less than a second or two. We note that all times are aggregate, but could be improved drastically with more compute resources.

### *5.3.1 Non-Hierarchical*

For the base line tests, we see that the matlab backslash operator consistently outperformed the conjugate gradient implementation. This can be seen in Figure 5.4 and Figure 5.5. This may be due to the number of iterations. However, the mean-squared error and run-time both produce better results using the backslash operator. Due to this, we only process further results using the more optimized solver.

When comparing window sizes (also in the same figures), we see that increasing the window size eventually reduces the mean-squared error to that of the global solver. However, it also raises the run-time significantly. From our experimentation, a window size of 256x256 usually produces usable results. The exception to this is on images with very large unknown regions, especially in high resolution images. However, in Mixed Reality, we try to avoid these types of settings.

### *5.3.2 Hierarchical*

For the hierarchical study, Figure 5.6 shows the effect that adjusting the window size and the number of iterations has on the average mean-squared error. We see that both window size and number of reductions improve accuracy, with the exception of very large windows (near full image resolution). This seems justified because a smaller window size would benefit more from the reduction process. For very large window sizes, the global solution on a non-reduced image should give the best results. Increasing the subimage size would consequently approach a more global solution.

With respect to execution time, Figure 5.6 shows the compute time applied at various window sizes and reduction amounts. As we would expect, the run time increases with both window size and number of reductions applied. We can also see that the run time stays within a factor of 2 when any number of reductions are applied. This is expected, as the total processing should only add work proportional to a sum of logs. We actually do better than this because we apply fewer than a logarithmic number of reductions.

Using the above as a guide, we compare the 256x256 window size to the non-window solution of [LLW06] in Figure 5.8. We see that the accuracy appears to be worse at first glance. However, when we also consider the first standard deviation's upper bound, we observe that the windowed solver's error rate is comparable to the distribution of the global solution.

### *5.3.3 High Resolution*

Additional high resolution output appears in Figure 5.9. The images were arbitrarily selected. As you can see, the images with larger trimaps correlate to the images with larger unknown regions in the trimaps. This makes sense when we consider that local information is not sufficient, hindering local performance. These images are too large to solve using a global Laplacian matrix.

## 5.4 Conclusions

We presented a parallelizable method to produce alpha mattes independent of the actual matting implementation by using overlapping sub-images. The method exploits locality of a solution, but also uses a hierarchical solver to propagate information more quickly. While the extra processing increased run-time, it had little effect on the error of the result compared to a global solution. It also enabled us to produce high resolution alpha mattes without additional memory requirements, which was an issue in the prior work.

The current implementation was non-optimized and not written to run in real-time. This includes the fact that we processed sub-images sequentially, skewing our final results negatively. We would like to see how well it performs on a GPU with different GPU friendly matting algorithms, but leave this for future work. We also do not have a very robust way to handle the situation when sub-solutions disagree. We could target these areas with more sub-images, but that raises computational complexity. An alternative could be to use a low resolution alpha result in these scenarios, but we leave the analysis and implementation to future work. Finally, we did not attempt to process any video signals using the presented method. For Mixed Reality, it would be very desirable to know if sub-images, processed temporally, would reduce computation requirements and produce good, consistent alpha mattes.

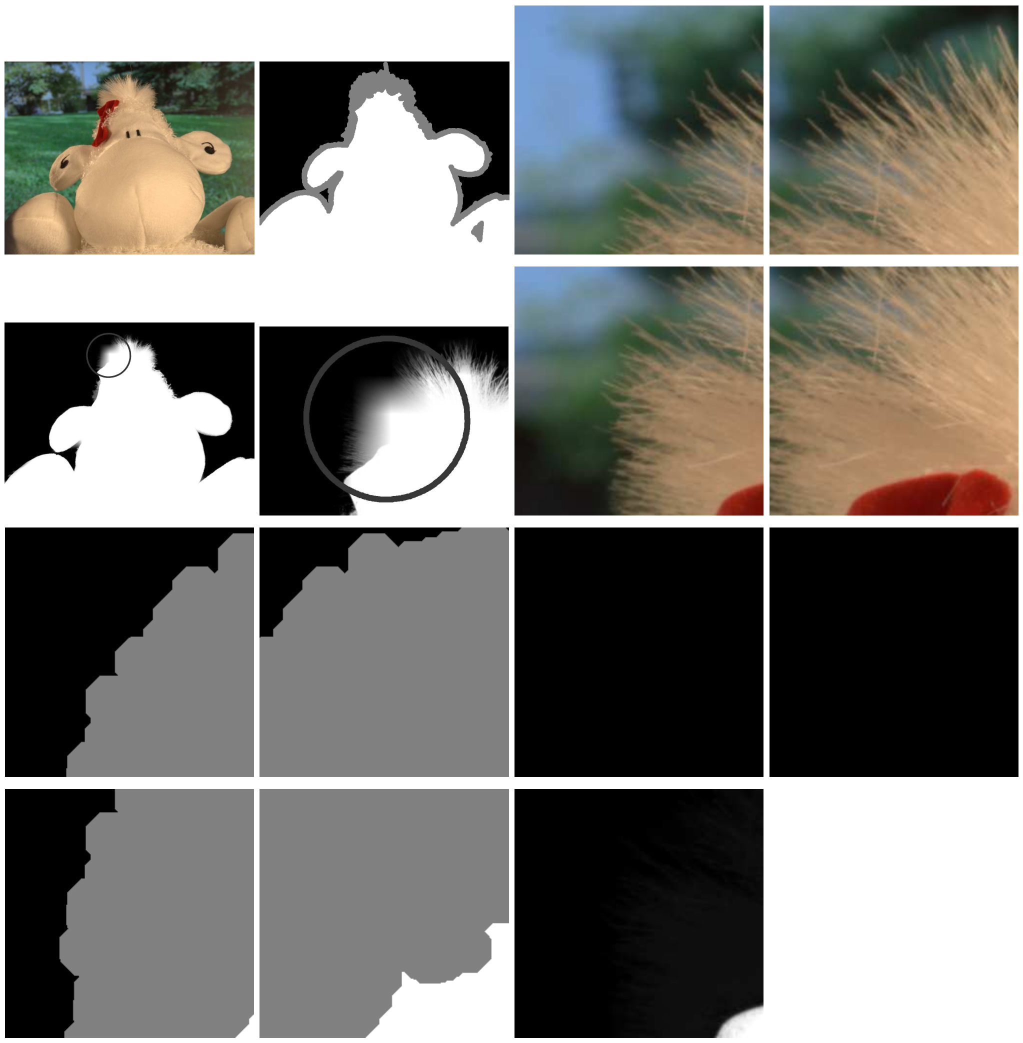


Figure 5.3: An example of underspecified sub-images using 256x256 sub-images and no hierarchical processing. 3 out of 4 of the sub-image trimaps do not contain constraints for both foreground and background. This leads to erroneous output, which is merged poorly. Raising the window size or adding one more hierarchy level alleviates the issue, as shown in Figure 5.1c

.

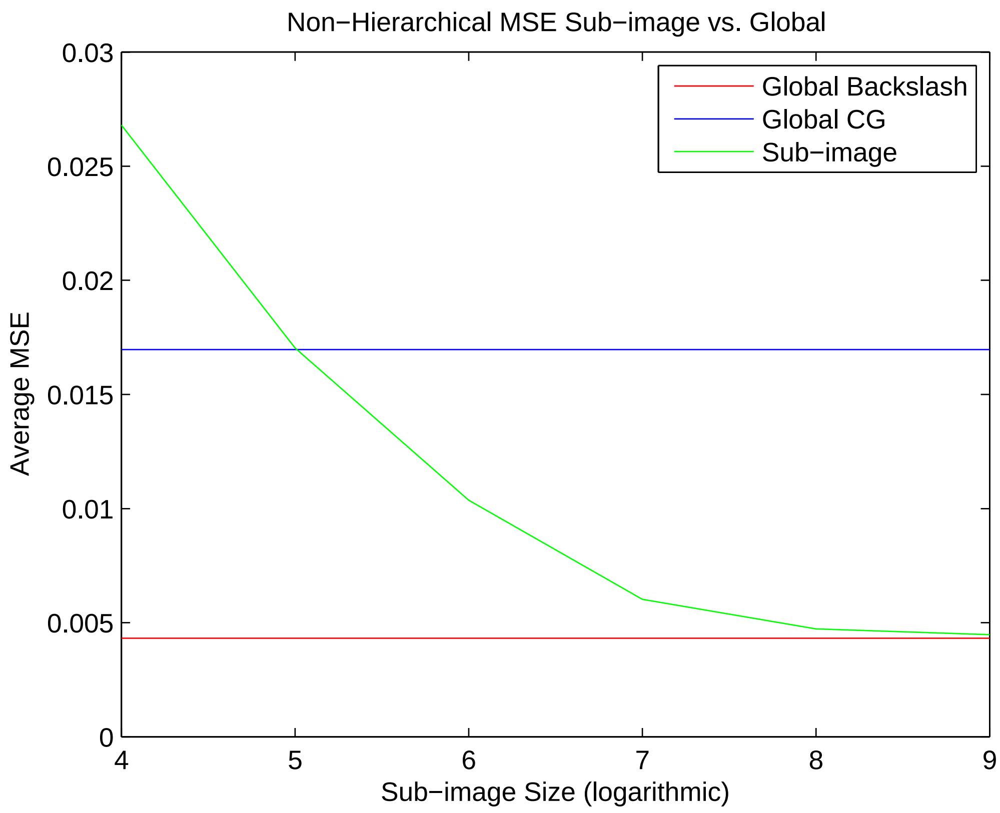


Figure 5.4: The mean-squared error of the non-hierarchical solver. We see that increasing window size improves performance. The images in this set are larger than 512x512, so it is nice to see the window size can be smaller thna the image size.

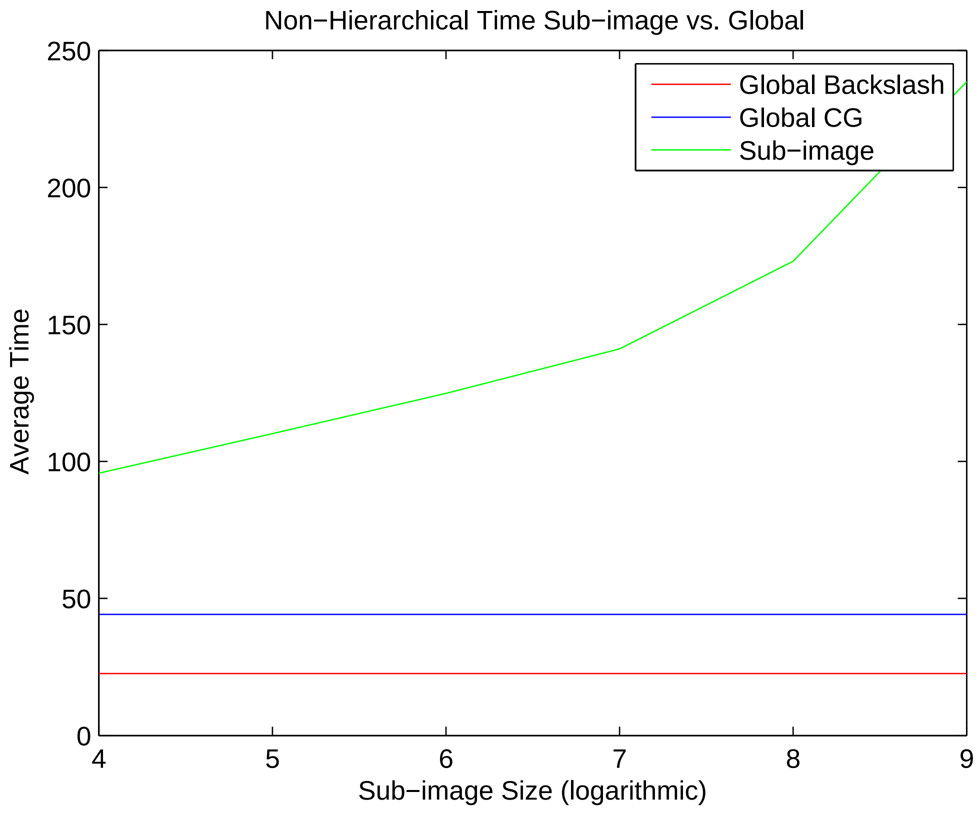


Figure 5.5: The average run time of the non-hierarchical solver. We see the size of a window drastically increase run time. This is to be expected, but only reflects sequentially processing windows.

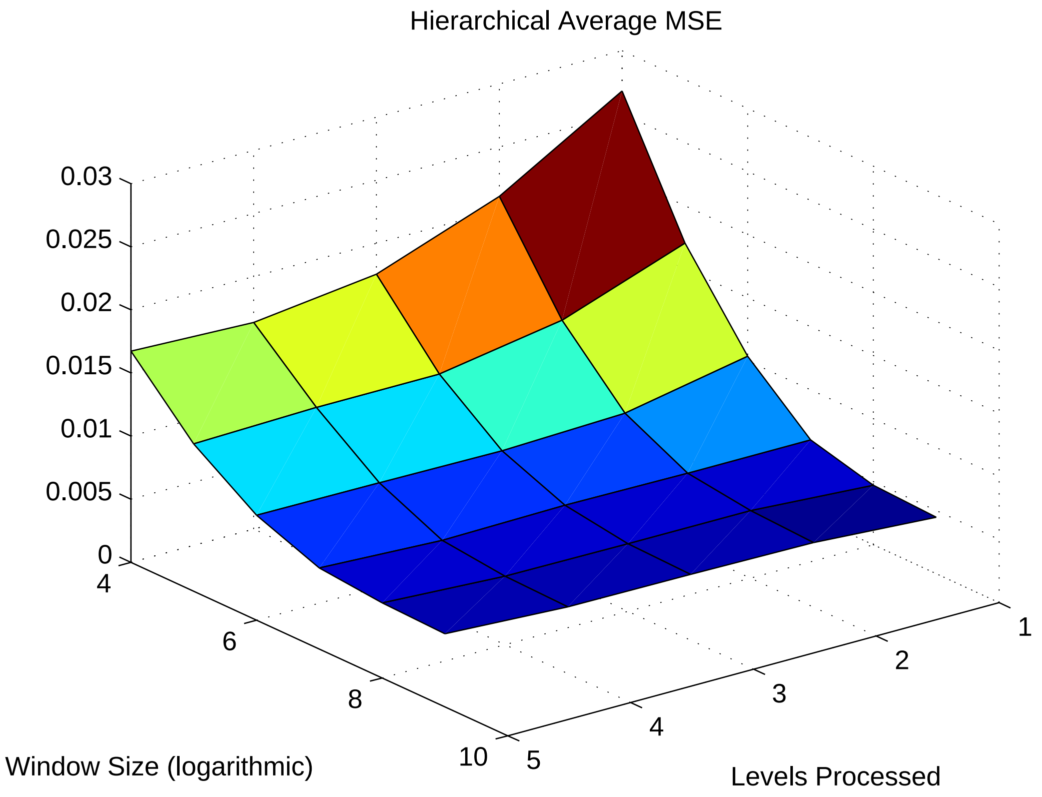


Figure 5.6: The mean-squared error of the window based matting using various window sizes and hierarchy iterations. Better performance is typically achieved with larger window and more iterations. As the windows get close to the full image size, we see a global solver gets better results than a hierarchical solver.

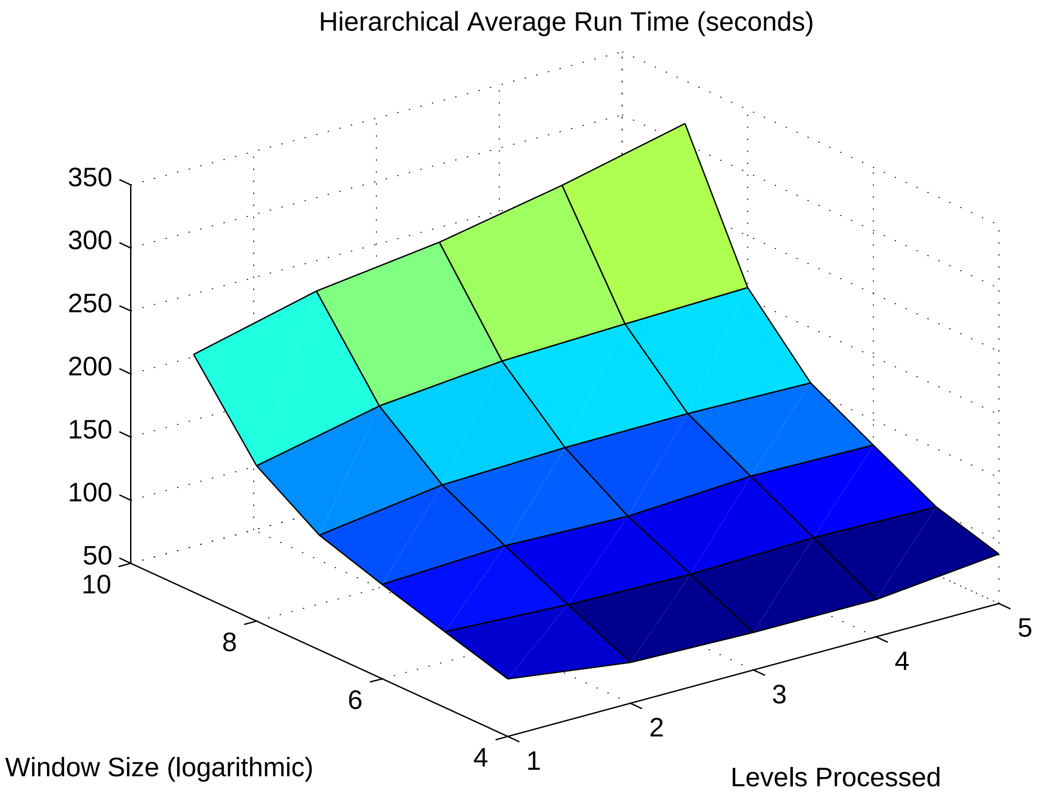


Figure 5.7: Run time (in seconds) of the window based matting using various window sizes and hierarchy iterations. Increases to either parameter effect run-time, but not by a large factor.

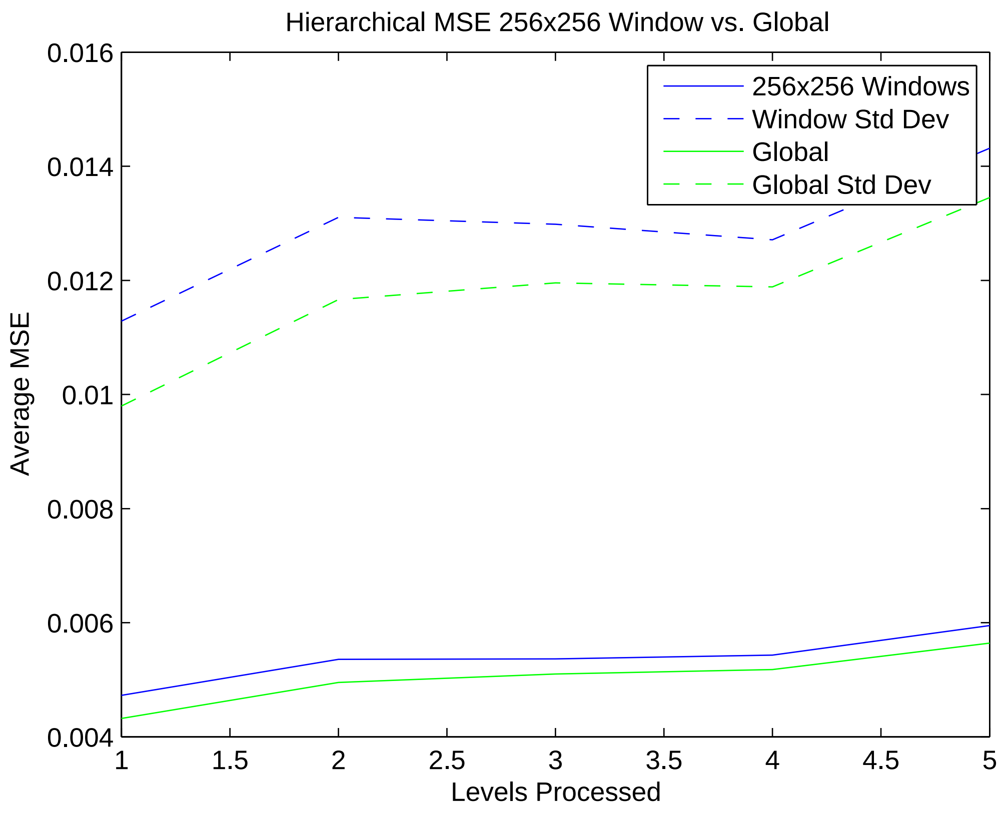


Figure 5.8: Comparison of a 256x256 windowed solver to a global solver. While the windowed solver is less accurate, it is not too much worse error than the global solution. When standard deviations are factored in, the performance is comparable.

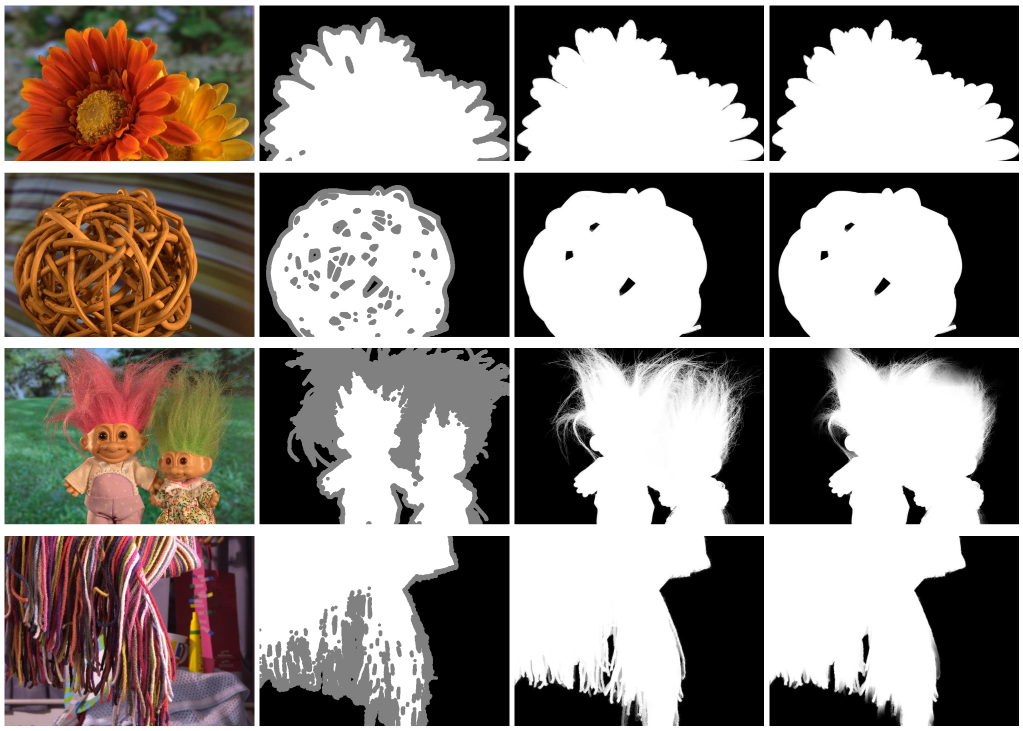


Figure 5.9: High resolution examples arbitrarily chosen from the training data set. The left alpha matte is a low resolution global solution. The right alpha matte is solved using the sub-image, hierarchical solver.

# CHAPTER 6: APPLICATIONS

So far, we have discussed alpha matting research as it applies to Mixed Reality scenarios. This chapter demonstrates example research applications that require real-time matting. The following examples are a select subset of multi-disciplinary research projects applying the aforementioned methods. We separate the projects into two distinct groups, medical assessment/rehabilitation and military training.

## 6.1 Medical Assessment/Rehabilitation

Projects involving assessment and rehabilitation of patients typically fall into the category of allowing patients to learn (or re-learn) a behavior in private or perhaps in the prsence of a therapist. In order to do this effectively, we need to present environments that resemble the real world context and yet are carried out in a controlled facility, such as a clinic.

### *6.1.1 Warehouse*

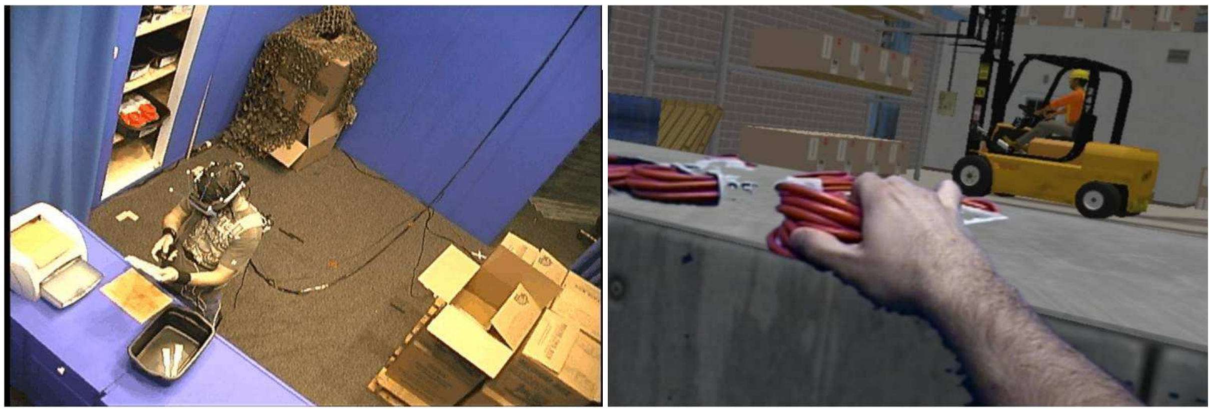

(a) (b)

Figure 6.1: (a) The Warehouse scenario in the real-world with physical props. (b) An example participant view, including inventory items sorted on the virtual counter.

The Mixed Reality Warehouse, funded by the Air Force Office of Scientific Research, project aimed to study the cognitive effects of different elements in an assessment scenario [BMH09, FSW09]. The hope was to evaluate the effectiveness of a scenario capable of assessing the performance of post traumatic stress disorder (PTSD) patients in the context of their performing simple, manual tasks. Aspects of this included visual and auditory distractors. A warehouse setting was created with physical props, see Figure 6.1a for the layout. The participant was tasked with filling orders receieved on a fax machine. This involves locating appropriate objects, tracking them with a bar code scanner, and filling order boxes. Virtual props consist of a typical inventory warehouse, including a forklift that is stacking crates from arriving supply trucks, getting closer and closer to the counter on which the user is placing orders, eventually knocking over a shelf near the user. Sounds were synchronized with events and so led to louder and louder auditory as well as visual distractions.

The pilot study placed cognitively healthy participants in the scenario to determine if the scenario and environment were realistic. The findings were that participants were consistently overloaded and stressed out, particularly due to extreme virtual environment cues (both too much, and not enough). In short, healthy participants could not easily perform tasks. With a goal of sending patients through cognitive therapy, the study provided a way to assess the efficacy of a scenario before sending actual patients through the process.

The scenario heavily relied on blue screening using the PCA keying method presented in Chapter 3. The environment was surrounded with blue curtains. The physical counters and bookcases (required for tactile feedback) were also painted blue to allow virtual counter tops to replace the scene, providing a more consistent environment. However, the participant still needed to see the real-world bar code scanner and inventory items, and needed a real counter on which to place orders and get new ones off the fax machine.

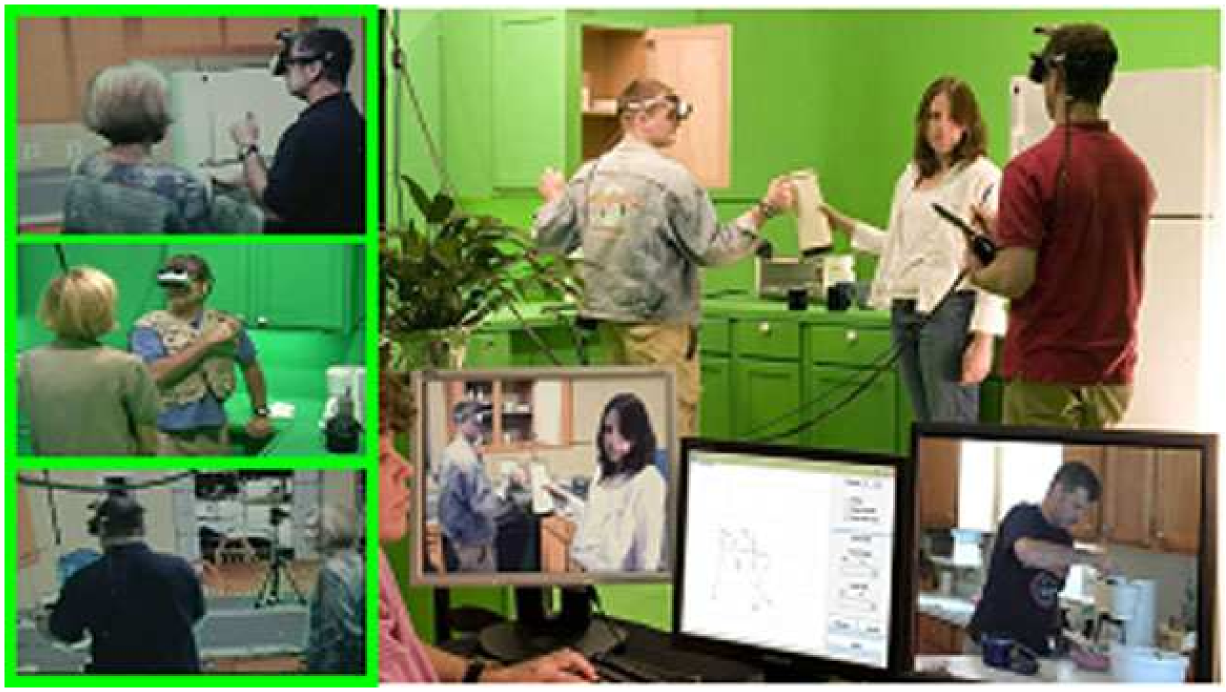


Figure 6.2: The MR Kitchen environment and user experience. We use a virtual kitchen (similar to the participant's) for cognitive rehabilitation in patients with brain injury.

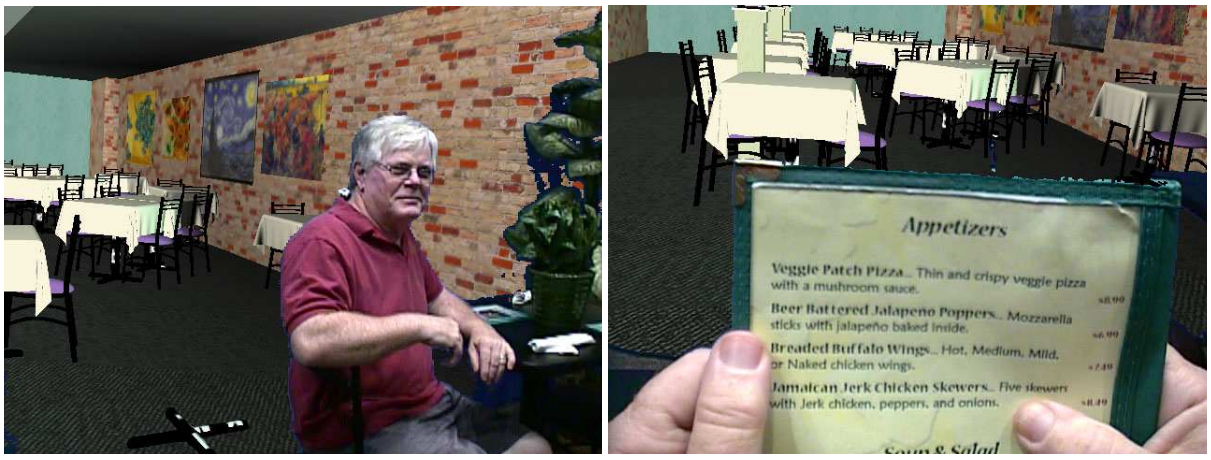
### *6.1.2 Kitchen*

The Mixed Reality Kitchen was a pilot study to determine the effectiveness of Mixed Reality for cognitive rehabilitation [FSW05]. A single participant with a brain injury was chosen and tracked throughout six experimental sessions in the environment. The participant's physical kitchen was captured and modeled so that all training occurred in a familiar location for the patient. Interestingly, the patient who had experienced a brain aneurysm would claim the context was unfamiliar, but would do the same in his own home kitchen. The environment can be seen in Figure 6.2.

Similar to the Warehouse, the Kitchen experiment used physical cabinets and counters, this time green, to provide tactile feedback to the user. Needed appliances, such as a refrigerator, coffee maker and toaster oven, were real, as were coffee cups (stored in the cabinets) and the makings for breakfast (stored in the refrigerator and cabinet shelves). The patient was asked to perform typical daily tasks, such as making a cup of coffee, a piece of toast, and a bowl of cereal. Data, such as position and heading, were logged for comparison in subsequent visits.

The result of the study was that the user showed less erratic behavior as the study progressed. In fact, by the sixth time in the lab, his performance was as good as that of his spouse. Interestingly, it was even better at his home. Moreover, the patient generalized this knowledge, showing his ability to find random items in the kitchen cabinets outside the context of the trained tasks. This showed that a Mixed Reality scenario could be used to retrain patients with cognitive disabilities in day to day tasks.

### *6.1.3 Restaurant*


Appetizers
Veggie Patch Pizza.. Thin and crispy veggie pizza with a mushroom sauce.
Beer Battered Jalapeno Poppers.. Mozzarella sticks with jalapeño baked inside.
Breaded Buffalo Wings.. Hot, Medium, Mild, or Naked chicken wings.
Jamaican Jerk Chicken Skewers.. Five skewers with Jerk chicken, peppers, and onions.
Soup & Salad

(a) (b)

Figure 6.3: (a) The MR Restaurant environment is primarily virtual. (b) However, it was vital to the study to have readable menus.

The MR Restaurant scenario was a study to help users with speech impairments interact more confidently in real world environments [FH08]. This particular scenario chose ordering food in a restaurant, a task that is well-suited to the requirements. The user was asked to order food from a physical menu, while seated at a physical table in a virtual restaurant. The goal was to lessen the amount of speech stuttering that occurred. Challenges included the juxtaposition of phonemes on the menu items, the varying behaviors (courteous vs. curt) of the wait staff, and environmental sounds.

This particular environment was more challenging from a Mixed Reality standpoint, as any video processing errors could make the menu unreadable. In a situation where this occurred, it would be a detriment to the actual study. A menu in the mixed environment can be seen in Figure 6.3b. The PCA based matting worked very well for this particular experiment, as the menu and human skin tone are vastly different than the background colors chosen for the backdrop.

## 6.2 Military Training

Various military branches have the desire to use Mixed Reality to train soldiers in a variety of tasks and contexts. This is because a training drill requires several experts in the field to play roles of other agents. If these other roles could be performed by a simulation and presented via Mixed Reality, it would save many person hours. It would also allow soldiers to train in the field with scenarios created back home. As a result, the military is primarily interested in seeing if such scenarios would actually raise situational awareness and perform better than desktop simulations.

### *6.2.1 VIRTE*

The Office of Naval Research VIRTE initiative was to bring together a federation of simulators in a single training scenario. We supplied a Mixed Reality forward observer simulation to call in an air strike in the distributed simulation. Participants needed to rely on rocks in the real environment as cover while using a handset to radio in coordinates and, if appropriate, call off an attack. See Figure 6.4a for an example of the environment. When the users were not careful, they would lose their cover and other agents in the simulation would be aware of their position. On success, a network controlled aircraft would come deliver the air strike.

This scenario was the pilot study for a better chroma-key algorithm than the pyramid method, [BL99b] being used on the GPU. This was primarily due to users taking cover and seeing shadows more often than not. The research in Chapter 3 is a direct result of this project. As a consequence, the military has shown much more interest in the use of Mixed Reality and head-mounted displays.

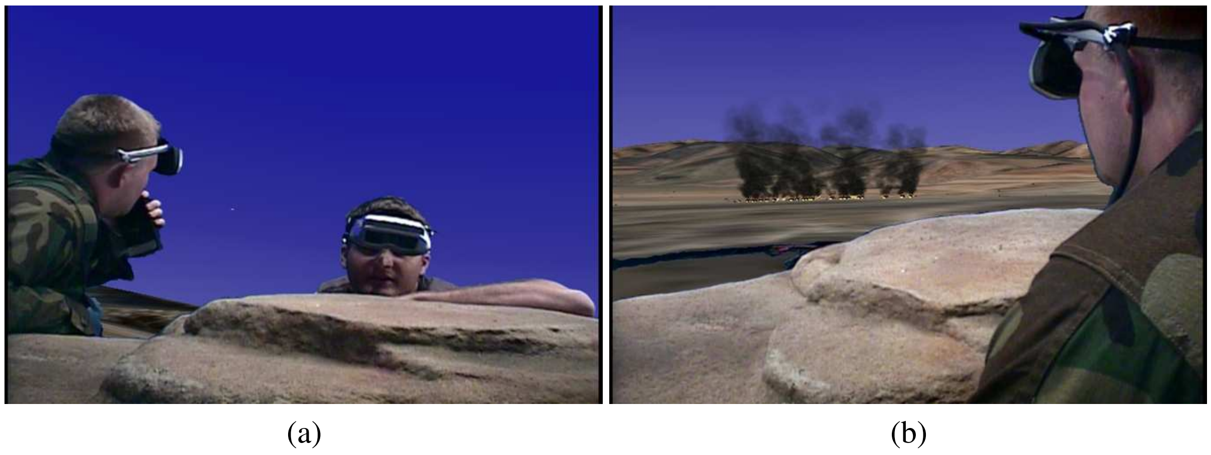


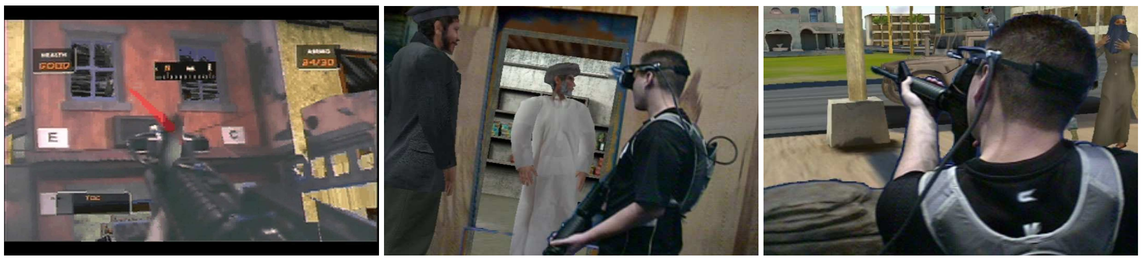
Figure 6.4: The Forward Observer calls in a virtual air-strike while under cover from physical rocks. (a) The approach of the plane, and (b) the aftermath.

### 6.2.2 Military Operations on Urban Terrain

The Mixed Reality Military Operations on Urban Terrain (MOUT) project, funded by the Army Research Development and Engineering Command, extended to five phases over the course of several years (about 6-7 years) [HSM02, CM03, CS05]. The project created an urban environment and military scenarios where a user needed to use situation awareness cues to recognize threats and survive hostile activity, without inflicting civilian or fraternal casualties. For example, as one of the scenario progresses, the bystanders begin to leave. The soldier needs to pick up on this cue to realize an attack might be imminent.

Due to the length of this project, the matting used developed over time. The initial solution was a software classifier that dropped the high precision bits in the color channels to produce a manageable, 16 bit, look up table. See Figure 6.5a. The table was initialized solely based off known color images with no concept of training. The solution was very noisy and slow, even after attempts to optimize the CPU bound code.

The next iteration used a GPU based implementation of the pyramid classifier, [BL99b], again with no further optimizations. A bilateral filter blur kernel was applied to attempt smoothing any noise that occurred due to lighting or camera inaccuracies. See Chapter 3.4 for details. Eventually, the MOUT simulation used the full process described in Chapter 3, as shown in figure Figure 6.5b-c.



(a) (b) (c)

Figure 6.5: (a) An image from MOUT 3, which used the software table based chroma keying. (b) & (c) Images from MOUT 4, which used PCA chroma keying on a GPU. The amount of noise in the background is greatly reduced.

### 6.2.3 *Lunar Lunge*

The Lunar Lunge project was an attempt to measure whether or not Mixed Reality was a viable training methodology compared to desktop, first-person simulations. Users shot water bubbles on the moon at lunar robots while defending a friendly, mobile base station. The robots would shoot back, trying to hit either the user or the target.

The study presented the scenario as a science fiction game to allow adolescents, who typically play video games on consoles, to participate in a point based game without introducing military elements. The scoring system was used as one of the factors in determining effectiveness. Participants were split into two groups, those who played the desktop game before the Mixed Reality game, and those that experienced the modalities in the reverse order.

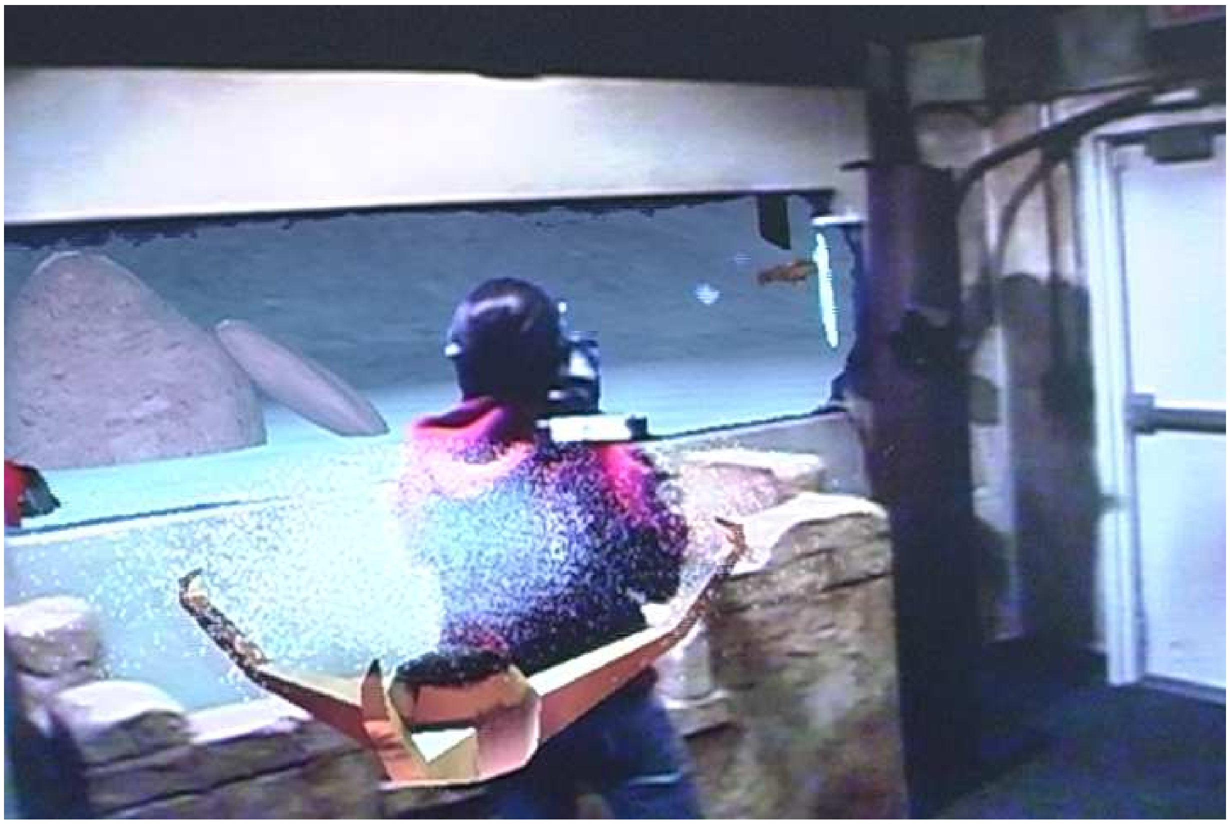


Figure 6.6: The Lunar Lunge game compared a mouse and keyboard interface against a Mixed Reality interface to explore whether or not Mixed Reality was more effective than desktop simulations for military training of the next generation of recruits.

Alpha matting was strongly necessary due to the number of objects that could be considered as targets. Furthermore, this project wanted a more hands-on approach. Targets and robots were allowed to enter the courtyard (move from the background to the foreground) and vice versa through various doorways. Figure 6.6 shows an example of such a situation where a virtual object enters the foreground. The Mixed Reality renderer had to account for such objects and add various layer to the Mixed Reality rendering.

The main conclusion from this study was that prior game experience only helped subjects during the desktop-based version of the game. The main contribution to success in the Mixed Reality version was the order of modalities. Those who experienced the desktop game first learned the rules of engagement, even if their performance was abysmal. Learning these rules in advanced of MR was the key to success in that modality. This result suggests that training should start on a desktop game and then progress to MR sessions prior to field experiences.

## 6.3 Conclusions

The presented multidisciplinary studies illustrate the effectiveness of real-time matting in Mixed Reality projects. The use of mixed environments for assessment/rehabilitation and training shows promise, which motivates us to find a setup that lacks the requirements of known background colors. The natural image matting method presented in Chapter 4 moves towards such a goal, but suffers from temporal consistency. The method still performs relatively well with unknown constant color backgrounds or environments with various colors (such as both blue and green screen). The main reason that this research has not been used in any mentioned projects is that the depth buffer must come from a sensor (or possible crude depth from stereo). We have not attempted to mount a depth sensor on a head-mounted display at this time. In any case, an ideal solution would work with arbitrary backgrounds with no need to calibrate lighting or sensors.

# CHAPTER 7: CONCLUSIONS

## 7.1 Results

In Chapter 3, we show that PCA can be used to classify foreground and background pixels on a GPU. The pixels with low confidence solutions can be optimized with gradient descent or a bilateral filter to produce a real-time, usable matting system for Mixed Reality. As we can see in the results, the algorithm works well at classifying pixels and imposed virtually no additional computation time. The optimization step, on the other, improves results but can become costly. Solutions are still achievable for two images with enough time to render virtual data and composite a scene. Chapter 6 shows six example applications of the matting system applied to actual research projects.

Chapter 4 explores how we can remove the constant color background prior and still differentiate the forground from background. We see that, using a depth camera, we can generate a trimap in real-time. The matting optimizations we added to the sampling algorithm of [GO10] allow video streams to be processed in real-time. However, we observe that temporal consistency becomes a factor with video streams. Our attempts to improve this issue via temporal parameters and processing both produces suboptimal output and increases run-time beyond our desired interactive goal. We feel the shared sampling algorithm is not well-suited due to loss of information during the sharing step, which happens to be the strength of the algorithm with respect to run time.

Finally, in Chapter 5, we explored solving alpha mattes for large resolution images using sub-image solutions. We propose overlapping sub-image matting solutions using a simple bilinear weighting function that is constant for all windows. The window is not adjusted as we compute lower resolution solutions to propagate to the final solution. The algorithm works independent of any given matting algorithm, but we chose the method of [LLW06] due to its influence on the community. We show that, for smaller images, the solution produces slightly worse, but usable, errors at the cost of more run time. We then show the solution applied to images from a high resolution data set, which could not be solved due to memory limitation using the original implementation.

## 7.2 Future Work

The outstanding work falls into two categories: video processing in real-time, and high resolution processing. For the former, we note that the method of Chapter 4 does not apply to any projects listed in Chapter 6. This is because the depth camera was not rigidly mounted to the color camera. Producing a Mixed Reality scenario with this constraint is rather difficult, as background subtraction could also be used to initialize the trimap. We leave it for future work to try an embedded depth camera on a head-mounted display, to give a synchronized depth feed with the color feed. This would enable and motivate matting algorithms to use a depth buffer in conjunction with a virtual depth buffer to perform Mixed Reality compositing using a depth signal. We believe this would provide a much more robust experience, provided the alpha matte (or depth buffer) can be optimized in real-time in a temporally consistent way.

For high resolution processing in Chapter 5, the algorithm relies on hierarchical down sampling. Due to the resolution of the images, the down sampling, coupled with the local sub-image, can remove high frequency data in complex scenes. While we feel this may be acceptable for opaque objects in Mixed Reality, it would be better to preserve unknown regions on the trimap with small neighborhood sizes instead of allowing them to be removed during reduction. We also did not try our local sub-image approach with more than the matting Laplacian approach of [LLW06]. We think it would be interesting to see the approach implemented with a GPU friendly solver and assessed for real-time performance. We also leave detecting and correcting problem areas (where multiple sub-images produce drastically differing alpha values for a particular pixel) to future work.

## 7.3 Summary

The research presented here has developed efficient algorithms that extend the applicability of matting research to Mixed Reality, even in environments with poorly controlled lighting and inconsistent material. In Chapter 3, we began with a GPU matting algorithm, classified with PCA and optimized with gradient descent, to quickly produce alpha mattes for known constant color backgrounds. In Chapter 4, we observe that the optimization step is color-independent, and attempt to initialize our known and unknown pixels using a crude depth buffer. We show that the shared sampling method of [GO10] works well for single images, but lends itself to temporal inconsistencies. Chapter 5 looks at scalability in matting. We propose a sub-image based approach that is independent of window data and may be run in parallel, hierarchically, using any matting algorithm. The results are smoothed across windows. We show that this methodology, while more CPU intensive, does not increase memory necessary for matting, allowing us to process higher resolution images. In Chapter 6, we show applications of matting in various multidisciplinary research projects. We demonstrate why matting was necessary for a successful outcome. The combination of all of this research aims to extract alpha mattes or pixel accurate depth buffers from high definition video streams in real-time. While our work has been successful in improving immersion in Mixed Reality experiences, as is evidenced by the numerous successful applications, extensions to the results presented here would take such experiences to a whole new level of immersion and usefulness.

# LIST OF REFERENCES

[BL99a] F. Van Den Bergh and V. Lalioti. "Software Chroma Keying in an Immersive Virtual Environment." *South African Computer Journal*, **24**:155–162, 1999.

[BL99b] F. Van Den Bergh and V. Lalioti. "Software Chroma Keying in an Immersive Virtual Environment." *South African Computer Journal*, **24**:155–162, 1999.

[BMH09] Nicholas Beato, Daniel P. Mapes, Charles E. Hughes, Cali M. Fidopiastis, and Eileen M. Smith. "Evaluating the Potential of Cognitive Rehabilitation with Mixed Reality." In *Human-Computer Interaction*, pp. 522–531, 2009.

[BS09] Xue Bai and Guillermo Sapiro. "Geodesic Matting: A Framework for Fast Interactive Image and Video Segmentation and Matting." *Int. J. Comput. Vision*, **82**:113–132, April 2009.

[BZC09] Nicholas Beato, Yunjun Zhang, Mark Colbert, Kazumasa Yamazawa, and Charles E. Hughes. "Interactive chroma keying for mixed reality." *Comput. Animat. Virtual Worlds*, **20**:405–415, June 2009.

[CCS01] Yung-Yu Chuang, Brian Curless, David H. Salesin, and Richard Szeliski. "A Bayesian Approach to Digital Matting." In *Proceedings of IEEE CVPR 2001*, volume 2, pp. 264–271. IEEE Computer Society, December 2001.

[CM03] C. E. Hughes C. B. Stapleton and J. M. Moshell. "Mixed Fantasy." In *Proceedings of ISMAR*, 2003.

[CS05] D. E. Hughes C. E. Hughes, C. B. Stapleton and E. Smith. "Mixed Reality in Education, Entertainment and Training: An Interdisciplinary Approach." *IEEE Computer Graphics and Applications*, **26**(6):24–30, 2005.

[FH08] C. M. Fidopiastis and C. E. Hughes. "Use of Psychophysiological Measures in Virtual Rehabilitation." In *Virtual Rehabilitation 2008*. IEEE Computer Society, 2008.

[FRH08] Michael Figl, Daniel Rueckert, David Hawkes, Roberto Casula, Mingxing Hu, Ose Pedro, Dong Ping Zhang, Graeme Penney, Fernando Bello, and Philip Edwards. "Image Guidance for Robotic Minimally Invasive Coronary Artery Bypass." In *MIAR '08: Proceedings of the 4th international workshop on Medical Imaging and Augmented Reality*, pp. 202–209, Berlin, Heidelberg, 2008. Springer-Verlag.

[FSW05] C. M. Fidopiastis, C. B. Stapleton, J. D. Whiteside, C. E. Hughes, S. M. Fiore, G. A. Martin, J. P. Rolland, and E. M. Smith. "Human Experience Modeler: Context Driven Cognitive Retraining and Narrative Threads." In *4th International Workshop on Virtual Rehabilitation (IWVR2005)*, 2005.

[FSW09] Cali Fidopiastis, Angela Salva, B Wiederhold, C Hughes, E Smith, A Alban, and M Wiederhold. "Cognitive Therapy using Mixed Reality for those impaired by a Cerebrovascular Accident (CVA)." *Frontiers in Neuroengineering*, 2009.

[GKO03] Ronen Gvili, Amir Kaplan, Eyal Ofek, and Giora Yahav. "Depth keying." *Stereoscopic Displays and Virtual Reality Systems X*, **5006**:564–574, 2003.

[GO10] Eduardo S. L. Gastal and Manuel M. Oliveira. "Shared Sampling for Real-Time Alpha Matting." *Computer Graphics Forum*, **29**(2):575–584, May 2010. Proceedings of Eurographics.

[GSA05] Leo Grady, Thomas Schiwietz, Shmuel Aharon, and Rdiger Westermann. "Random walks for interactive alpha-matting." In *IN VIIP*, pp. 423–429. ACTA Press, 2005.

[HLW10] Mengcheng Huang, Fang Liu, and Enhua Wu. "A GPU-based matting Laplacian solver for high resolution image matting." *Vis. Comput.*, **26**:943–950, June 2010.

[HSH05] Charles E. Hughes, Christopher B. Stapleton, Darin E. Hughes, and Eileen M. Smith. "Mixed Reality in Education, Entertainment, and Training." *IEEE Computer Graphics and Application*, **25**(6):24–30, 2005.

[HSM02] C. E. Hughes, C. B. Stapleton, J. M. Moshell, P. Micikevicius, P. Garrity, and P. Dumanoir. "Challenges & Opportunities Simulating Future Combat Systems via Mixed Reality." In *23rd Army Science Conference (ASC 2002)*, 2002.

[HST10] Kaiming He, Jian Sun, and Xiaoou Tang. "Fast matting using large kernel matting Laplacian matrices." *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on*, **0**:2165–2172, 2010.

[JMA06] Neel Joshi, Wojciech Matusik, and Shai Avidan. "Natural video matting using camera arrays." *ACM Trans. Graph.*, **25**:779–786, July 2006.

[LLW04] Anat Levin, Dani Lischinski, and Yair Weiss. "Colorization using optimization." *ACM Trans. Graph.*, **23**(3):689–694, 2004.

[LLW06] Anat Levin, Dani Lischinski, and Yair Weiss. "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*, pp. 61–68, Washington, DC, USA, 2006. IEEE Computer Society.

[LRL08] Anat Levin, Alex Rav-acha, and Dani Lischinski. "Spectral Matting." *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **30**:1699–1712, 2008.

[Mis92] Y. Mishima. "A Software Chromakeyer Using Polyhedric Slice." *NICOGRAPH'92*, pp. 44–52, 1992.

[MK94] Paul Milgram and Fumio Kishino. "A Taxonomy of Mixed Reality Visual Displays." *IEICE Transactions on Information Systems*, **E77-D**(12), December 1994.

[MMY06] Morgan McGuire, Wojciech Matusik, and William Yerazunis. "Practical, real-time studio matting using dual imagers." In *Rendering Techniques 2006 (Proceedings of the 17th Eurographics Symposium on Rendering)*. Eurographics, Eurographics Association, June 2006.

[PD84] Thomas Porter and Tom Duff. "Compositing digital images." In *SIGGRAPH '84: Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, pp. 253–259, 1984.

[RRK10] C. Rhemann, C. Rother, P. Kohli, and M. Gelautz. "A spatially varying PSF-based prior for alpha matting." In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pp. 2149 –2156, june 2010.

[RRW09] C. Rhemann, C. Rother, Jue Wang, M. Gelautz, P. Kohli, and P. Rott. "A perceptually motivated online benchmark for image matting." In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pp. 1826–1833, june 2009.

[RT00] M.A. Ruzon and C. Tomasi. "Alpha estimation in natural images." In *Computer Vision and Pattern Recognition, 2000. Proceedings. IEEE Conference on*, volume 1, pp. 18–25, 2000.

[SB96] Alvy Ray Smith and James F. Blinn. "Blue screen matting." In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, SIGGRAPH '96, pp. 259–268, New York, NY, USA, 1996. ACM.

[SJT04] Jian Sun, Jiaya Jia, Chi-Keung Tang, and Heung-Yeung Shum. "Poisson matting." *ACM Trans. on Graphics*, **23**(3), 2004.

[SK09] Mikhail Sindeyev and Vadim Konushin. "A Novel Approach to Video Matting using Optical Flow." In *In Proceedings of GraphiCon*, pp. 340–343, 2009.

[SKF05] Andrei State, Kurtis P. Keller, and Henry Fuchs. "Simulation-Based Design and Rapid Prototyping of a Parallax-Free, Orthoscopic Video See-Through Head-Mounted Display." In *ISMAR '05: Proceedings of the 4th IEEE and ACM International Symposium on Mixed and Augmented Reality*, pp. 28–31, 2005.

[SLB06] Jian Sun, Yin Li, Sing Bing, and Kang Heung yeung Shum. "Flash matting." In *In Proceedings of ACM SIGGRAPH*, pp. 772–778. ACM Press, 2006.

[SM00] Jianbo Shi and Jitendra Malik. "Normalized Cuts and Image Segmentation." *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 2000.

[Smi02] Lindsay I Smith. "A tutorial on Principle Components Analysis." http://csnet.otago.ac.nz/cosc453/student_tutorials/ principal_components.pdf, February 2002.

[TMW11] Zhen Tang, Zhenjiang Miao, Yanli Wan, and Dianyong Zhang. "Video matting via opacity propagation." *The Visual Computer*, pp. 1–15, 2011. 10.1007/s00371-011-0598-3.

[UTS02] Shinji Uchiyama, Kazuki Takemoto, Kiyohide Satoh, Hiroyuki Yamamoto, and Hideyuki Tamura. "MR Platform: A Basic Body on Which Mixed Reality Applications Are Built." In *ISMAR '02: Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR'02)*, pp. 246–320, 2002.

[Vla78] P. Vlahos. "Comprehensive Electronic Compositing System." U.S. Patent 4,100,569, July 11, 1978. Expired.

[WC07] Jue Wang and M.F. Cohen. "Optimized Color Sampling for Robust Matting." In *Computer Vision and Pattern Recognition, 2007. CVPR '07. IEEE Conference on*, pp. 1–8, June 2007.

[WC08] Jue Wang and Michael F. Cohen. *Image and Video Matting*. Now Publishers Inc., Hanover, MA, USA, 2008.

[WFY07] Oliver Wang, Jonathan Finger, Qingxiong Yang, James Davis, and Ruigang Yang. "Automatic Natural Video Matting with Depth." In *Computer Graphics and Applications, 2007. PG '07. 15th Pacific Conference on*, pp. 469–472, November 2007.