Dynamic Occlusion Handling for Real-Time AR Applications

Ricardo Santos e Silva
ricardo.santos.e.silva@tecnico.ulisboa.pt

Instituto Superior Técnico, Lisboa, Portugal

October 2018

Abstract

AR is a technology that allows computer generated graphics to be overlaid in images or video captured by a camera in real time. This technology is often used to enhance perception by providing extra information or simply by enriching the experience of the user. AR offers a significant potential in many applications such as industrial, medical, education and entertainment. For AR to achieve the maximum potential and become fully accepted, the real and virtual objects within the user's environment must become seamlessly integrated. Three main types of problems arise when we try to achieve this effect: illumination problems, tracking problems and occlusion problems. In this work we present an algorithm that solves AR occlusions in real time. Our approach uses raw depth information of the scene to realize a rough foreground background segmentation. We use this information, as well as information from color data to estimate a blending coefficient to combine the virtual objects with the real objects into a single image. After experimenting with different scenes we prove that our approach is able to produce consistent and aesthetically pleasing occlusions between virtual and real objects, with a low computational cost. Furthermore, we explore several possible alternatives to improve the quality of the final results and overcome previous limitations.

Keywords: Augmented Reality, Real-time Realistic Occlusion, Dynamic Occlusion Handling, Alpha Matting

1. Introduction

Augmented Reality (AR) is an upcoming technology that allows computer generated graphics to be overlaid in images or video captured by a camera in real time. This technology is often used to enhance perception by providing extra information or simply by enriching the experience of the user. AR offers a significant potential in many applications such as industrial, medical, education and entertainment [29].

According to [5] for AR to achieve the maximum potential and become fully accepted, the real and virtual objects within the user's environment must become seamlessly integrated. Three main types of problems arise when we try to achieve this effect: illumination problems, tracking problems and occlusion problems.

Occlusion occurs when an object closer to the viewer obscures the view of objects further away [5]. In most AR applications the virtual objects occlude the real objects but sometimes the opposite can happen. Research on solving this problem has come up with three different approaches: Model-based [5], [10], [1], Object-based [19], [35] and Depth-based [5], [23], [11].

In this work we focus on solving the occlusion problem in a realistic manner. With the development of depth sensors, algorithms can now use depth information to handle occlusions. However, the information provided by the sensors has some problems: different types of noise, shadow effects and inaccurate mapping between depth and color data.

To improve the depth information researchers have come with multiple ways to improve the depth maps, for example [9, 18, 27]. [9] proposes a solution where the depth map edges can be aligned with the color map edges. This method only works well if the background and foreground are clearly distinguishable so it still does not provide a robust answer to the problem. On top of that, sometimes fuzzy objects can cause regions or pixels to not explicitly be on the foreground or on the background. In those cases we can get a more realistic result if we determine the relative transparency of the objects instead of just trying to figure out if they should be in front or in the back of each other. So, we can conclude that instead of just trying to find if we should display the virtual objects or the real objects in a per pixel basis a better approach

would be to determine a blending coefficient for each pixel, also called alpha matte.

In this work we tackle the occlusion problem as an alpha matting problem and propose a solution that is able to produce realistic occlusions in static and dynamic scenes. As a starting point we took inspiration from [13] and we propose some changes to that algorithm to improve the quality of the results.

2. Organization of the document

This document is organized as follows: Section 1 introduces the problem and describes the objectives of the work. Section 3 describes occlusion as an alpha matting problem. In section 4 we discuss the related work about this subject. In section 5 we describe the overview of the proposed solution. Section 6 to Section 11 describe the solution in detail. Section 12 describes technical information about the implementation of the solution and some performance measurements. In Section 13 we show the obtained results and compare them to other methods of performing occlusions. Finally, in Section 14 we discuss some limitations of the proposed solution as well as provide some pointers to what could be improved in the future.

3. Problem Formulation

We will tackle the occlusion handling problem as an alpha matting problem. Alpha matting addresses the problem of extracting foreground objects from static images or video sequences. Precisely separating a foreground object from the background can be achieved by estimating the alpha value for each pixel individually. An image *I* can be mathematically described as a combination of the foreground image *F* and the background image *B*:

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

where α defines the alpha matte (with $\alpha \in [0, 1]$).

We can use alpha matting to determine the alpha matte of the scene. We have to precisely separate the foreground objects (parts of the real scene that occlude virtual objects) from the background (objects that lie in the back of virtual objects). By calculating the alpha matte we can then draw the scene using Equation (1) to combine virtual and real objects into a single image.

Most alpha matting problems require a user-specified trimap to use as a starting point. A trimap is an image that segments the scene into three non-overlapping regions: definitely background, definitely foreground and unknown regions where it is not possible to determine if the real objects are behind or in front of the virtual objects.

It can be concluded that this problem has two main challenges: automatic generation of a trimap based on the relation between real and virtual objects and estimation of the alpha matte. The generated trimap should be as precise as possible and the unknown regions should be as small as possible to ensure good results.

4. Related Work

The main problems associated with this subject are how to improve the rough depth data captured by the sensor and study different methods for performing occlusion in the context of AR. We also researched about the alpha matting problem used to separate the background and foreground of images in different contexts. Finally we also looked at ways of generating trimaps without user input and in real time.

4.1. Occlusion Handling

Several solutions have been proposed to solve the occlusion problem. Here we present some solutions and discuss the results obtained by each approach.

According to [13] there are three main approaches used to solve the occlusion problem: model-based, object-based and depth-based.

Model-based methods rely on having a 3D model of the scene or having a model of the occluding object. With the model a simple depth test can be performed by aligning real and virtual objects via tracking. The 3D model can be obtained directly from existing resources, built with modeling software or reconstructed using cameras and sensors. Reconstruction algorithms [14, 2, 34] can reconstruct a semi dense and accurate model but the boundaries are still often noisy. According to [29] model-based methods are appropriate when the complexity of the scene is low and when it is easy to obtain 3D models of the objects involved.

Overall it can be concluded that this method only works in simple scenes and the quality of the results is heavily influenced by the quality of the models or reconstructions as well as the quality of the tracking. Furthermore, this category of methods is usually not able to deal with with dynamic occlusions or deformable objects.

Object-based methods usually use a contour of the occluding object to handle occlusions. In [20] the user specifies the occluding objects in keyviews and then occlusions in intermediate views can be generated automatically. [35] proposes a real-time approach which is divided into three steps: selection, tracking and occlusion handling. First the user specifies the occluding object by using an interactive segmentation method. The contour of the object is then tracked in real-time in subsequent frames. In the occlusion handling step the virtual object is drawn on top of the real world image and then all the pixels of the tracked object are

redrawn on top of the virtual object.

The main problem of this approach is that it only works if the relative position between real and virtual objects does not change, i.e., if we assume that one virtual object is in front of a real object we cannot move the real object behind that virtual object or the illusion will be broken.

Depth-based methods can handle occlusions for unknown objects without any user input. Nothing about shape, size or position needs to be known previously. They rely on external sensors to capture depth information of the real world such as depth sensors [9, 18] or stereo cameras [28] and then use the depth information to compute occlusion. There are some drawbacks: depth sensors still produce noisy depth maps and sometimes are not able to capture relevant information. Sensors do not deal well with shadows, are limited by their effective range and cannot perceive reflective or transparent materials. Additionally, depth maps often have low resolution and are not aligned with the color image of the camera.

The main problem of this approach is dealing with the problems associated with depth sensors but it is possible to overcome these issues by preprocessing the incoming depth data using different techniques, which we discuss in Section 4.2.

4.2. Improving and Enhancing Depth Maps

To solve the issues of the depth sensors researchers have come up with ways to improve the quality of the depth data by using software. [18] uses an inpainting algorithm to fill holes in the depth maps by expanding known regions into unknown regions. [28], proposed a method to obtain good quality dense disparity maps from stereo images with focus on sharp edges. [9] improves depth maps with sharp edges aligned to the edges of the color map. They extract edges from the depth map and from the color map and then use an edge snapping technique to improve the consistency between both. This procedure produces good results when the occluding object can be separated from the background but is very time consuming in higher resolutions.

Since depth maps are essential to many applications in computer vision researchers have come up with various algorithms to improve the quality of the maps. Edge-aware filters are useful for occlusion handling because they smooth images but preserve the edges. Some examples include bilateral filter [31], joint bilateral filter [17] and guided filter [12]. [27] presents a method based on color segmentation that aligns depth edges with color edges.

4.3. Alpha Matting and Trimap Generation

Matting is a known and well-studied technique that is used in many image and video editing applications. For occlusion handling the most relevant methods are the ones where the background is arbitrary and unknown. Alpha matting can be computed using color information, depth information or both. Color-based methods that are relevant for occlusion handling can be divided into two main types: sampling-based and propagation-based.

Sampling-based methods [4, 3, 25] assume that the true foreground and background colors of an unknown pixel can be estimated from known foreground and background samples. Sampling-based methods produce good results when the foreground and background have distinct color distributions. Propagation-based methods [21, 30] assume that foreground and background colors are locally smooth and solve the problem by propagating know alpha values into unknown regions. According to [32] these methods produce good results when the background and foreground patterns are simple, but the quality quickly degrades in more complex environments.

Several techniques combine the two approaches to achieve high quality results. These techniques collect pairs of foreground and background samples and estimate the alpha values in a global optimization problem, by selecting the best pairs available. [32] generates the samples from the nearest boundary pixels and [16] generates the samples from the closest super pixels. After the selection of sample pairs an objective function is used to calculate the best pair and generate the corresponding alpha value [32, 26].

Depth-based methods use depth information to solve the problem of generating high quality trimaps [22, 33]. These approaches start by dividing the foreground and background into a binary map. Then the unknown regions are discovered by applying erosion and dilatation to the foreground. The separation between foreground and background can be discovered by using a user-defined plane [33] or by a segmentation algorithm like k-means [36]. [7] proposes an adaptive dilation according to the fuzziness of the foreground object. This approach allows for fuzzy objects to be covered by larger unknown regions than objects with sharp edges.

Color and depth-based methods combine the information of color and depth to achieve the best quality. Hebborn *et al.* [13] generate an adaptive trimap automatically using depth and color information. They start by generating a coarse trimap using depth information. Then an adaptive dilation is applied to extend unknown regions in a way that that both depth and color boundaries are covered

while keeping the unknown regions as small as possible. After the refined trimap is found, known foreground and background regions of the color image are propagated toward the unknown regions. Finally, the alpha matte is estimated through a sample-based method where local samples are collected from the expanded foreground and background and the best samples are selected using an objective function.

5. Overview

Our solution uses color and depth information provided by a sensor to compute an alpha matte and combine the virtual objects with the real world scene. The main steps in this process are:

1. **Virtual Scene Rendering.** Renders the virtual scene into a color texture and a depth texture.
2. **Depth Data Smoothing.** Filters the depth information provided by the sensor to improve the quality of the depth map.
3. **Coarse Trimap Generation.** Uses the depth information provided by the sensor and the depth information of the virtual scene to generate an initial rough trimap.
4. **Adaptive Trimap Dilation.** Uses color information captured by the sensor to further refine the initial trimap. During this stage the unknown areas are dilated to cover possible fuzzy areas or areas where the depth information is inaccurate or invalid.
5. **Foreground and Background Color Propagation.** Propagates the colors of the known foreground and background regions into the unknowns areas of the trimap.
6. **Alpha Estimation.** Uses the colors of the expanded foreground and background images to find a good alpha value for each pixel of the unknown areas.
7. **Alpha Matte Smoothing.** Filters the generated alpha matte to reduce visual noise and small imperfections.
8. **Compositing.** Combines the color images of the virtual scene and real world using the alpha matte previously generated.

6. Depth Data Smoothing

Due to the fact that the depth data captured by sensors is typically noisy we first try to remove as much noise as possible using a 5 by 5 median filter. This filter improves the quality of the depth map by smoothing the edges of the objects in the scene.

7. Coarse Trimap Generation

In this step the objective is to split the scene into foreground, background and unknown regions.

The trimap represents the three possible types of regions by coloring each region with a diferent color: the foreground is painted with white, the background with black and the unknown regions with grey. A fourth color, red, is used to paint areas which are not covered by the virtual objects and, as such, there is no need to find if the virtual objects are occluded or not.

First we generate an initial trimap where the known foreground and background regions are found by applying a simple depth test between the scene depth data and the virtual depth data. If the sensor's depth data is invalid, which occurs due to shadowing effects, the region is marked as background.

Finally, to find the unknown regions, which are neither in the background or in the foreground, we apply a 3 by 3 sobel filter to the initial trimap. This filter detects the transitions between the foreground and the background regions which are marked as unknown. The sobel filter also provides us with information regarding the direction of the transitions which will be important in the next steps.

8. Adaptive Trimap Dilation

If we only took the depth information into account then we would obtain wrong alpha estimations because sometimes we would be taking samples from the foreground as if they were samples from the background and vice versa.

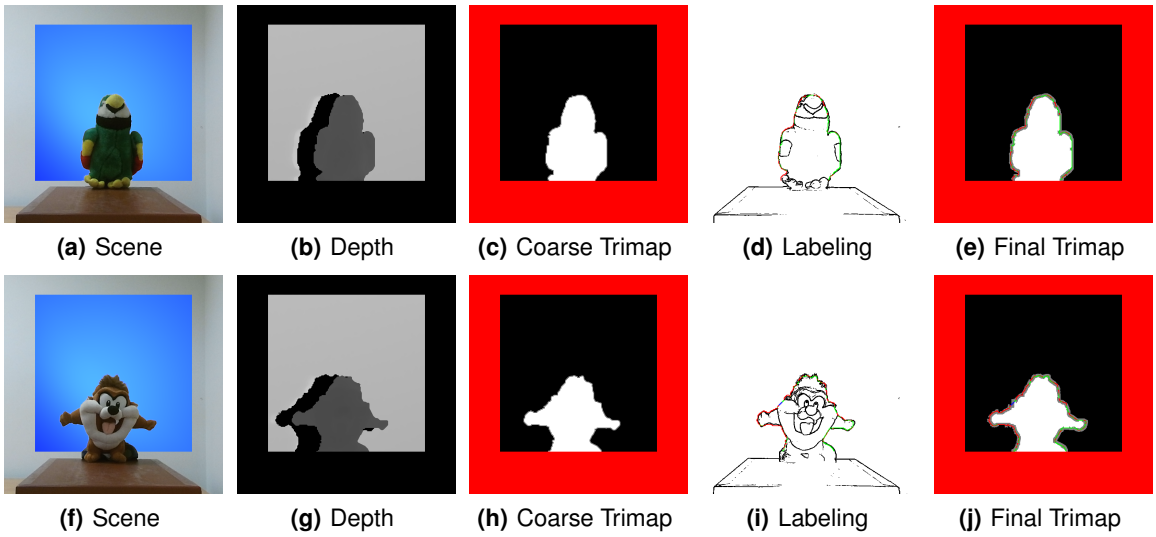
To solve this issue We need to expand the unknown regions in such a way that they cover not only the transitions in the depth map but also transitions in the color image. Furthermore, we want to keep these regions as small as possible otherwise they could be expanded until enough information was lost and make alpha estimation impossible.

To overcome these problems we take the boundaries of the depth map as an initial estimation and then expand the unknown regions towards the color boundaries. To achieve this goal we must first find the relative positions between the depth map boundaries and the color map boundaries so we can expand the unknown regions in the right direction.

8.1. Labeling of Unknown Regions

The goal of this step is to label all pixels in the unknown regions according to their position in relation to the boundaries of the color map.

After we have found the boundaries of the color image, by using a 3 by 3 sobel filter, we can categorize all pixels of the unknown regions into three categories: *front-half space*, *back-half space* and *no edge*. An unknown pixel *i* is labeled with *front-*



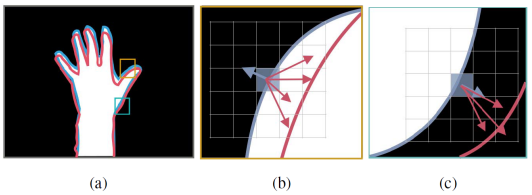
(a) Scene **(b)** Depth **(c)** Coarse Trimap **(d)** Labeling **(e)** Final Trimap

(f) Scene **(g)** Depth **(h)** Coarse Trimap **(i)** Labeling **(j)** Final Trimap

Figure 1: Adaptive Trimap Generation.

half space if *i* lies in the front-half space of an edge in the color image. Conversely, if the unknown pixel *i* lies in the back-half space of an edge in the color image it is labeled as *back-half space*. If there are no edges in the color image around the unknown pixel *i* the pixel *i* is labeled as *no edge*.

To determine whether an unknown pixel is in the back-space or front-space of an edge or if it has no nearby edges in the color image we must look for pixels that belong to the boundaries of the color image in a window around the unknown pixel. If the number of color edge pixels is bellow a certain threshold then the unknown pixel is marked as *no edge.* Otherwise we can find if the unknown pixel is in front or behind the color edge pixels by computing the scalar product between the direction of the unknown pixel gradient and the direction from the unknown pixel to the color edge pixel as seen in Figure 2.



(a) (b) (c)

(a)

Figure 2: Labeling procedure. (a) unknown regions (blue) overlaid with boundaries in the color image (red), (b) the unknown pixel lies in the front-half space of the color boundaries, (c) the unknown pixel lies in the back-half space of the color boundaries.

8.2. Adaptive Dilation

In order to expand the unknown regions we took inspiration from [6]. We expand the unknown regions starting in the pixels marked previously as unknown and propagating them until we reach the boundaries of the color image. We use the information that was computed in the previous step in order to find the direction of expansion.

Furthermore when an initial unknown pixel is marked as *no edge* we expand the unknown region in all directions around that pixel. The amount of dilation applied in this case depends on the number of *no edge* pixels found in that region of the trimap. By implementing this mechanism we are able to cover areas where the information of the color is fuzzy, for example when fur or hair are present.

9. Foreground and Background Propagation

To get a good alpha estimation we need to find good samples of the known foreground and background colors. We propagate the known values into the unknown areas and later use the propagated colors to realize a search to find the best pairs in a window around each unknown pixel.

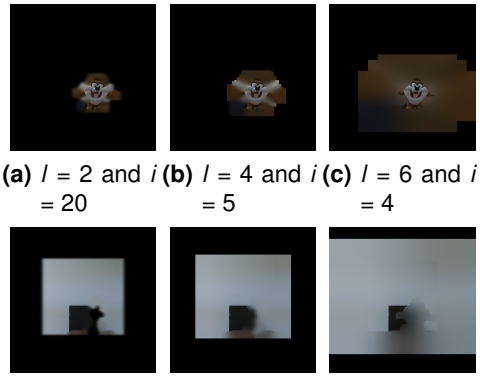
We took inspiration from [13] to solve this problem. The idea of the technique is to successively blur the known values into the unknown regions and write back only the blurred values. By repeating this process several times we diffuse the colors of the known boundaries into the unknown regions. This technique is similar to the known hole filling method proposed by [8]. To increase the speed of the process the blurring occurs in lower resolution images as seen in pyramid-based filtering algorithms [15, 24].

The process is applied to both the foreground and background. So, to simplify, we will only explain how it works for the foreground. Algorithm 1 illustrates the steps of a single iteration of the image diffusion.

The first step of the diffusion is to copy the foreground image into a new image *S* (line 1). Then we must initialize all the alpha values of the image

S (lines 2-8). We then build the image pyramid and initialize the first level with all the known foreground colors and set their α channel to 1 (line 9). We create the lower resolution levels of the image pyramid by sequentially down-sampling the previous level using a simple linear filter. Next we start from the lowest level of the pyramid and start to rebuild the pyramid by up-sampling the lowest level using a quadratic B-spline interpolation. It is also important to point out that the factors of the quadratic interpolation are weighted by the α values so that only known values are taken into account. We repeat this process until we reach the original resolution. To finish off we must combine the original image with the new blurred one (lines 11-17). The goal is to copy only new color values where previously there were none. To achieve a smooth transition between diffusion steps a constant *n* is used to control how the new and previous colors are combined.

The amount of propagation is dependent on the number of levels *l* of the image pyramid and on the amount of diffusion steps *i*. If the number of pyramid levels is increased then the area of the propagated values grows and if we perform more diffusion steps the propagated colors become smoother. The changes in the final result of the propagation algorithm according to the number of pyramid levels *l* and diffusion steps *i* can be seen in Figure 3.



(a) *l* = 2 and *i* = 20 **(b)** *l* = 4 and *i* = 5 **(c)** *l* = 6 and *i* = 4

(d) *l* = 2 and *i* = 20 **(e)** *l* = 4 and *i* = 5 **(f)** *l* = 6 and *i* = 4

Figure 3: Propagation of Foreground and Background. Results of the propagation algorithm with different amounts of pyramid levels *l* and diffusion steps *i*

For each pixel that is created during the image propagation algorithm we save the iteration *i* of when that pixel was created. We later use this information to measure how far that pixel color is from the original pixels in the alpha estimation process.

Algorithm 1: Single step of the diffusion algorithm

1 copy input image *F* with 4 channels to image *S*
2 **for** *each pixel* p^s *of the image S* **do**
3 **if** $p_a^s > 0$ **then**
4 $p_a^s \leftarrow 1$
5 **else**
6 $p_a^s \leftarrow 0$
7 **end if**
8 **end for**
9 build image pyramid of *S* with *l* levels
10 go top-down the pyramid to smooth *S*
11 **for** *each pixel* p^f *of the image F* **do**
12 **if** $p_a^s > 0$ **then**
13 $w_{i-1} \leftarrow \frac{p_a^f}{n}$ ▶ calculate normalized weight
14 $p_{rgb}^f \leftarrow w_{i-1} p_{rgb}^f + (1 - w_{i-1}) p_{rgb}^s$
15 $p_a^f \leftarrow min(p_a^f + 1, n)$ ▶ increase and clamp alpha
16 **end if**
17 **end for**

10. Alpha Estimation

To estimate the alpha matte we take pairs of samples from the expanded foreground and background images and try to find the pair of values that best represent the color of the color image when linearly combined. The best pair is defined by an objective function that takes into account a color cost and a propagation cost.

We can estimate the α value of a given sample pair (F_k, B_l) using:

$$\alpha_p(F_k, B_l, I_p) = \frac{(I_p - B_l)(F_k - B_l)}{||F_k - B_l||^2} \quad (2)$$

where Ip is the color of the unknown pixel in the color image.

10.1. Objective Function

When we want to estimate the alpha value of a certain unknown pixel $p_{i,j}$ we try to minimize the objective function for all the possible pairs of foreground and background colors in a window of *n* by *n* pixels around p:

$$(F_k, B_l) = wC_{col}(F_k, B_l, I_p) + C_{pro}(F_k, B_l) \quad (3)$$

where w defines the weight of the color cost, $C_{col}(F_k, B_l, I_p)$ defines the color cost function(Equation (4)) and $C_{pro}(F_k, B_l)$ defines the propagation cost function (Equation (5)).

After we find the best pair we can estimate the α value using Equation (2). The alpha value of the

known values is set to 1 where the foreground is known and to 0 where the background is known.

10.2. Color Cost Function

Measures how well a sample pair is able to represent I_p of the color image by linearly combining itself. If a pair of foreground and background color is able to represent the color I_p of the color image the cost should be small:

$$C_{col}(F_k, B_l, I_p) = ||I_p - (\alpha F_k + (1-\alpha)B_l)|| \quad (4)$$

where α is the estimated α value for the pair F_k, B_l using Equation (2).

10.3. Propagation Cost Function

To reduce the impact of wrongly propagated colors we take into account information about when in the process of the image propagation a specific pixel color was created. If the color was created during the first iterations then that color lies closer to the known colors. That way, colors that were created during the first steps should have a low propagation cost:

$$C_{pro}(F_k, B_l) = \frac{d(F_k) + d(B_l)}{2d_m} \quad (5)$$

where the function $d()$ returns the number of the iteration d_i (with $d_i \in [0, d_{m-1}]$) in which the color was created during the image propagation algorithm. d_m represents total number of iterations of the image propagation algorithm.

11. Alpha Matte Smoothing

After we compute the alpha values of all the pixels we smooth them by applying a 5 by 5 low pass filter to the final alpha matte. This way we remove any unwanted noise and improve the final alpha matte quality.

12. Implementation Details and Performance

To implement our solution we opted to use C++, OpenGL 4.3 and the OpenGL Toolkit (GLUT). All the computational steps take place in the GPU and are programmed in OpenGL Shader Language. The search for good sample pairs in the alpha estimation step is also done in the GPU to be as quick as possible.

Each computational step is performed as a rendering pass where the result is drawn into a texture using frame buffers. That texture can then be used in the next computational steps when needed.

We used the Kinect V2 sensor to capture color and depth information.

12.1. Performance Evaluation

The performance measurements were obtained for a resolution of 1920 by 1080 in both the color and the depth data. The propagation algorithm was performed in a window of 1024 by 1024 around the area of interest (the center of the virtual object). The computer used to perform the measurements is equipped with an Intel i7 processor and a NVIDIA GeForce 1070 graphics card. The measurements obtained are the average value of 1000 cycles of the algorithm.

Number of Unknown Pixels	6617	6978	13973
Trimap Generation (ms)	0.1228	0.1286	0.1333
Propagation (ms)	1.328	1.406	1.368
Alpha Estimation (ms)	0.02433	0.02575	0.03592
Total (ms)	1.576	1.541	1.518

Table 1: Performance measurements of the main steps of the solution. With 4 pyramid levels *l* and 6 diffusion steps *i*.

Diffusion Steps *i*	Pyramid Levels *l*	Time in ms
2	10	0.8658
2	20	1.093
4	5	0.7029
6	4	0.7678
10	2	0.965
20	2	1.205

Table 2: Performance measurements for different amounts of pyramid levels *l* and diffusion steps *i*.

As seen in Table 1 the number of unknown pixels do not cause significant changes in the overall performance of the algorithm. As expected, the amount of unknown pixels affect mainly the performance of the alpha estimation step, since this step depends directly on the amount of unknown pixels. Anyways, the most computational intensive step is the propagation of the known color values of the foreground and background into the unknown regions. The propagation step is responsible for almost 90% of the total time. This step, as seen in Table 2, increases linearly with the amount of pyramid levels *l* and the diffusion steps *i*. The algorithm needs around 1.5 ms for generating each frame which easily beats the real-time performance benchmark of 30 frames per second. Our solution delivers a performance of 40 to 50 frames per second.

13. Results

To show our solution working we chose three representative scenarios: one scenario where the virtual object is simple (a plane) and the background and foreground colors are substantially different, a second scenario where we keep the same simple virtual object but where the foreground and background have similar colors and a third scenario where we use a more complex virtual object.

To compare results we implemented an occlusion computing prototype that uses only the depth to compute occlusions with a simple depth test. We

then took the simple raw depth method and improved it by applying a median filter to the depth map and by smoothing the resulting alpha matte with a low pass filter. Finally, we also compare our results with the results from the original alpha matting [13].

The figures can be seen at the end of the document in Figure 4.

14. Conclusions

In this work we implemented the algorithm proposed by Hebborn *et al.* [13] and further refined the results by using different filters to treat the depth data captured by the sensor, by tweaking the adaptive dilation step and by adding a final post processing step to smooth and improve the produced alpha matte. These changes made the final result more stable and more aesthetically pleasing.

15. Limitations

Our solution is not being able to perform very well when the foreground and background have similar colors. When this scenario occurs we are not able to extract information about the color boundaries of the image and so we must rely only on depth data which is, most of the times, not accurate enough.

Another limitation is dealing with very narrow regions between objects of the scene. Sometimes it is not possible to accurately distinguish between foreground and background in these regions. This happens because the depth data has not enough resolution to detect changes in depth values in narrow regions. As the depth map edge filter fails to detect boundaries in these regions they are not not marked as unknown and are treated as foreground.

Another problem which sometimes affects the final result is the flickering that happens in the depth data captured by the sensor we used. We tried to minimize this problem by smoothing the alpha matte both spatially and temporally. We were able to improve the final result by applying a low pass filter to the alpha matte but we were not able to implement any kind of temporal filtering without causing a significant performance drop.

16. Future Work

One possible improvement to the current solution is to apply more advanced techniques on the optimization problem in the alpha estimation step. Currently we use a primitive local search to find the best pairs possible. Maybe one could apply artificial intelligence techniques in this step to come up with a better solution to estimate the alpha matte.

Another possible improvement is to apply some kind of temporal stabilization to the final result using information from the previous frames. We tried to apply an averaging operation between the current and the last frames but were not able to get any improvements without slowing down the final result. Maybe as the graphic cards get faster and faster this solution could be viable. If not maybe some sort of artificial intelligence could also be used to reduce the flickering in the depth information that sometimes bleeds into the final result.

Anyways, we feel like most of the issues affecting this solution could be solved by simply using a more accurate sensor which will likely be available in the near future.

References

[1] P. A. Fortin and P. Hebert. Handling occlusions in real-time augmented reality : Dealing with movable real and virtual objects. In *Third Canadian Conference on Computer and Robot Vision (CRV)*, page 54, July 2006.

[2] R. A. Newcombe, S. J. Lovegrove, and A. J. Davison. Dtam: Dense tracking and mapping in real-time. In *IEEE International Conference on Computer Vision*, pages 2320–2327, November 2011.

[3] A. Berman, A. Dadourian, and P. Vlahos. Method for removing from an image the background surrounding a selected object. January 2000.

[4] A. Berman, P. Vlahos, and A. Dadourian. Comprehensive method for removing from an image the background surrounding a selected object. October 2000.

[5] D. Breen, E. Rose, and R. T. Whitaker. Interactive occlusion and collision of real and virtual objects in augmented reality. Technical report, European Computer-Industry Research Centre, September 1995.

[6] L. Chen, H. Lin, and S. Li. Depth image enhancement for kinect using region growing and bilateral filter. In *Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012)*, pages 3070–3073, November 2012.

[7] J. Cho, T. Yamasaki, K. Aizawa, and K. H. Lee. Depth video camera based temporal alpha matting for natural 3d scene generation. In *2011 3DTV Conference: The True Vision - Capture, Transmission and Display of 3D Video*, pages 1–4, May 2011.

[8] J. Davis, S. R. Marschner, M. Garr, and M. Levoy. Filling holes in complex surfaces using volumetric diffusion. In *Proceedings. First International Symposium on 3D Data Processing Visualization and Transmission*, pages 428–441, June 2002.

[9] C. Du, Y. Chen, M. Ye, and L. Ren. Edge snapping-based depth enhancement for dynamic occlusion handling in augmented reality. In *2016 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 54–62, September 2016.

[10] J. Fischer, H. Regenbrecht, and G. Baratoff. Detecting dynamic occlusion in front of static backgrounds for ar scenes. In *EGVE '03 Workshop on Virtual environments*, pages 153–161, January 2003.

[11] K. Hayashi, H. Kato, and S. Nishida. Occlusion detection of real objects using contour based stereo matching. pages 180–186, December 2005.

[12] K. He, J. Sun, and X. Tang. Guided image filtering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(6):1397–1409, June 2013.

[13] A. K. Hebborn, N. Höhner, and S. Müller. Occlusion matting: Realistic occlusion handling for augmented reality applications. In *2017 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 62–71, October 2017.

[14] S. Izadi, D. Kim, O. Hilliges, D. Molyneaux, R. A. Newcombe, P. Kohli, J. Shotton, S. Hodges, D. Freeman, A. J. Davison, and A. Fitzgibbon. Kinectfusion: Real-time 3d reconstruction and interaction using a moving depth camera. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology (UIST)*, pages 559–568, October 2011.

[15] P. J Burt. Fast filter transforms for image processing. *Computer Graphics and Image Processing*, 16:20–51, May 1981.

[16] J. Johnson, E. Shahrian, H. Cholakkal, and D. Rajan. Sparse coding for alpha matting. In *IEEE Transactions on Image Processing*, volume 25, page 1, April 2016.

[17] J. Kopf, M. Cohen, D. Lischinski, and M. Uyttendaele. Joint bilateral upsampling. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2007)*, 26:96, July 2007.

[18] A. Leal, L. Altamirano Robles, and J. Gonzalez. Occlusion handling in video-based augmented reality using the kinect sensor for indoor registration. In *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications*, pages 447–454, November 2013.

[19] V. Lepetit and B. Marie-Odile. A semi-automatic method for resolving occlusion in augmented reality. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition. IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, pages 2225–2230, June 2000.

[20] V. Lepetit and B. marie odile. A semi-automatic method for resolving occlusion in augmented reality. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2225–2230, June 2000.

[21] A. Levin, D. Lischinski, and Y. Weiss. A closed-form solution to natural image matting. In *IEEE transactions on pattern analysis and machine intelligence*, volume 30, pages 228–42, March 2008.

[22] T. Lu and S. Li. Image matting with color and depth information. In *International Conference on Pattern Recognition*, pages 3787–3790, January 2012.

[23] M. M. Wloka and B. G. Anderson. Resolving occlusion in augmented reality. In *Symposium on Interactive 3D graphics (I3D)*, pages 5–12, February 1995.

[24] J. Ogden, E. Adelson, J. Bergen, and P. J. Burt. Pyramid-based computer graphics. *RCA engineer*, 30:4–15, September 1985.

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

[26] E. S. L. Gastal and M. Oliveira. Shared sampling for real-time alpha matting. In *Eurographics 2010*, volume 29, pages 575–584, May 2010.

[27] M. Schmeing and X. Jiang. Edge-aware depth image filtering using color segmentation. *Pattern Recognition Letters*, 50:63–71, December 2014.

[28] J. Schmidt, H. Niemann, and S. Vogt. Dense disparity maps in real-time with an application to augmented reality. In *Sixth IEEE Workshop on Applications of Computer Vision, 2002. (WACV 2002). Proceedings.*, pages 225–230, December 2002.

[29] M. M. Shah, H. Arshad, and R. Sulaiman. Occlusion in augmented reality. In *2012 8th International Conference on Information Science and Digital Content Technology (ICIDT2012)*, volume 2, pages 372–378, June 2012.

[30] J. Sun, J. Jia, C.-K. Tang, and H.-Y. Shum. Poisson matting. In *ACM SIGGRAPH*, volume 23, pages 315–321, August 2004.

[31] C. Tomasi and R. Manduchi. Bilateral filtering for gray and color images. In *Sixth International Conference on Computer Vision (IEEE Cat. No.98CH36271)*, pages 839–846, January 1998.

[32] J. Wang and M. F. Cohen. Optimized color sampling for robust matting. In *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, June 2007.

[33] O. Wang, J. Finger, Q. Yang, J. Davis, and R. Yang. Automatic natural video matting with depth. In *15th Pacific Conference on Computer Graphics and Applications*, pages 469–472, October 2007.

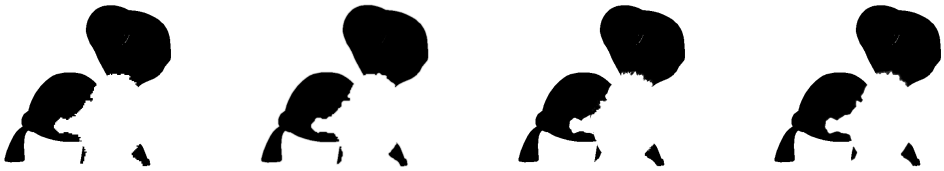
[34] T. Whelan, S. Leutenegger, R. Salas Moreno, B. Glocker, and A. Davison. Elasticfusion: Dense slam without a pose graph. July 2015.

[35] T. Yuan, G. Tao, and W. Cheng. Real-time occlusion handling in augmented reality based on an object tracking approach. In *Sensors*, volume 10, April 2010.

[36] J. Zhu, M. Liao, R. Yang, and Z. Pan. Joint depth and alpha matte optimization via fusion of stereo and time-of-flight sensor. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 453–460, June 2009.

(a) Raw **(b)** Improved **(c)** Original **(d)** Our

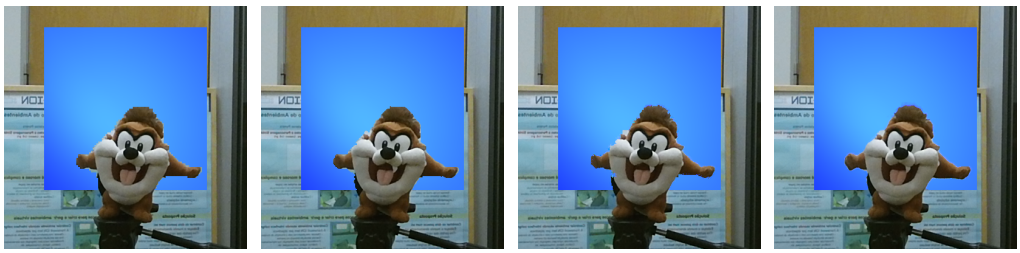
(e) Result of (a) **(f)** Result of (b) **(g)** Result of (c) **(h)** Result of (d)



(i) Raw **(j)** Improved **(k)** Original **(l)** Our

(m) Result of (i) **(n)** Result of (j) **(o)** Result of (k) **(p)** Result of (l)

(q) Raw **(r)** Improved **(s)** Original **(t)** Our



(u) Result of (q) **(v)** Result of (r) **(w)** Result of (s) **(x)** Result of (t)

Figure 4: Results of the four tested algorithms.