Synthetic Depth-of-Field Imaging Using Stereo Images

Sigal Shaul
University of California San Diego
La Jolla
sshaul@ucsd.edu

Justin Tahara
University of California San Diego
La Jolla
jtahara@ucsd.edu

June 3, 2021

Abstract

Depth-of-field imaging is what draws many people to acquiring expensive cameras in order to create a professional image emphasizing a subject. This usually requires a camera body and a lens in order to properly produce results with blurred backgrounds in relation to the subject. We aim to recreate these images through software which would allow more people with lower quality cameras to simulate a more expensive camera. By using stereo images, we create a disparity map that is later used to create a depth map. We then identify the subject through image segmentation in order to understand what part of the image to keep in focus. After obtaining the necessary information, we blur the non-subject area of the image in accordance to the depth map we produced. Lastly, we stitch the blurred background with the subject in focus to create a synthetic depth-of-field image with the use of stereo images.

1. Introduction

Recently, many efforts in improving cell phone cameras have been made to allow the casual photographer to create professional-looking photos on their mobile devices. One thing that the hardware on cell phones lack is the ability to change the aperture on the phone camera. Aperture on the lens is used by photographers with professional cameras to change the depth-of-field of the camera, which affects which items in teh image will be in focus and which will be blurred. This technique can be used to emphasize a subject in the photo by putting them in focus while the background is blurred. We decided to create a synthetic depth-of-field image processing pipeline using a pair of stereo images that can be taken from a phone camera without a change in aperture.

First, we mapped the disparity between the two stereo images and smoothed it out to create the disparity map that is then converted into a depth map for the image. We used image segmentation techniques to determine the subject of the image, and then used the depth map and the subjects to determine the depth-of-field of the camera we are trying to mimic, which is given by the minimum and maximum depths of the subject areas. The depth map is also used to blur the background with different kernel sizes based on the depth of the pixel within the image. Altogether, this gives us a nice effect that would be similar to that of a professional camera with a lens.

1.1. Existing Research

We considered multiple research papers as a basis for our project. Many of the recent papers took a look at creating synthetic depth-of-field with a single image[2][3][4], which we used as inspiration for the background blurring and stitching the image together. These papers use additional hardware and complex machine learning models that we do not currently have access to, so we decided to move forward with Stereo Images instead of a single image to create the depth-of-field aspect for our project. For image segmentation, we considered an open source machine learning model implemented with Keras and TensorFlow called Mask R-CNN[1]. Using the stereo images and inspiration from the papers and tools mentioned, we were able to yield a result that mimics a professional camera.

1.2. Existing Solutions

The existing solution we improve upon in our project is the naive approach of simply extracting the subject in the photo and blurring the remaining image. This approach can be seen in popular video conference software like Zoom and Microsoft Teams, as well as in the open source TensorflowJS model called BodyPix in their function called `drawBokehEffect`. The approach still requires proper image segmentation, but this method of background blurring creates a very artificial-looking image. This is especially apparent around the feet of a human subject, for example. This naive approach of blurring everything but the subject would end up blurring the floor around their feet, which should not be blurred when using a real camera because it would be within the depth of field.

2. Datasets

There are many Stereo Image datasets that are available, but for our project we used the Middlebury Stereo dataset that can be obtained from their website. With the Middlebury dataset we were able to choose from a variety of their datasets that match our needs and give us the disparity maps that are the ground truth, so we used this as a reference to see that our results when creating the depth map were accurate. The datasets also contain calibration information including the baseline and camera information to translate the disparity maps into depth maps. Using all the data that is provided we recreated the results and compared them against the ground truth. We compared results in increments in order to ensure that our process is working properly and adequately for the next steps.

3. Proposed Solution

We discuss our proposed solution in order to accomplish the task of creating a Synthetic Depth-of-Field Image using Stereo Images.

3.1. Depth Map

We began by creating a depth map using the stereo images that we obtained from the Middlebury Dataset. This depth map would later be used in other sections of our proposed solution to help construct our final image.

3.1.1 Disparity Map

The depth map construction required some work as we are not given this information by default from the images themselves. We first created a disparity map of the two stereo images using OpenCV's library of functions. We initialized a stereo semi-global block matcher and adjusted the parameters to yield the best results. We then created a matcher for the right image that we call upon using the matcher we just created before. This left us with 2 matchers that are then used to create two disparity maps. Lastly, in order to regularize the results that we have just obtained, we used a Weighted Least Squares filter to output one regularized disparity map. This gave us a disparity map without empty spaces and areas that are not recognizable, as we have regularized everything to form a smooth disparity map.

3.1.2 Disparity Map to Depth Map

We then proceeded to use the disparity map to produce a depth map. In order to conduct this step, we relied on the equation:

$$Z = \frac{f \cdot B}{d} \tag{1}$$

The Z in the equation is the depth in millimeters that we are trying to find. The f stands for the focal length of the lens that is taking the picture. The B is the horizontal distance between the two cameras in millimeters. The f and B are all provided in the callibration data in the Middlebury dataset. Lastly, the d stands for the disparity pixel value that is taken from the disparity map that we created. Using all of these elements, we are then able to produce a depth map that accurately represents the 3D depth within an image using two stereo images.

3.2. Image Segmentation

In this step we will created a mask for the subject and background so we can leave the subject of the image in focus and only blur the background.

3.2.1 Detecting the Subject

The first step in determining a foreground for the image is figuring out what items will be the subject of the photograph. Using an open source Tensorflow implementation of a regional convolutional neural network called Mask R-CNN, we detected objects within the image and created masks for them. The model was pretrained on the MS COCO dataset in the open source implementation, so we were able to load the pretrained model weights directly from the project and use it.

In our project, we assumed the objects that the photographer will want to be in focus will be around the center of the image, so we created a rectangle around the center that takes up 20% of the image area to determine which items will be in focus. Going through each mask we retrieved from the Mask R-CNN, we check if the mask overlaps with any part of the rectangle, and if it does, we add that to the subject objects that will be in focus in the image.

3.2.2 Determining the Depth-of-Field

Once we obtained the masks of the objects that will be in focus, we then referred to the depth map to see what the minimum and maximum depths of those masked items are. Since we wanted the full object to be in focus, the range of depths of the object will specify the depth of field of the camera we want to mimic.

With this depth-of-field, we could now go through the masks of objects we found earlier and determine if any other objects that were not in the center of the image would be in focus in our final image. The way we determined this is we found the minimum and maximum depth in the depth map in the region covered by each mask, and if this range of depths is within the depth of field, we added the object to a list of in-focus objects for the image. The remaining objects are left out of the mask of in-focus objects as they should be out of focus like the rest of the background.

3.2.3 Using the GrabCut Algorithm

Now that we have some masks that covered the areas of the image that are in the subject, we created one final mask that stores the information of all of these objects. For each of the objects that should be in focus (that we found in the previous steps of image segmentation), we set the final mask value to 1 where the mask of the object is, which signals that these pixels are definitely part of the foreground. The remaining pixel locations are set to 2, indicating that we are unsure if this is part of the foreground or background. We then pass this mask into the GrabCut algorithm implemented by OpenCV to get a mask that separates those foreground objects, which are our subject objects, from the background.

3.3. Gaussian Blurring the Background

Now that we have the segmented mask of the subject and the depth map for the image, we can now move onto the blurring of the background.

3.3.1 Scale Space

In order to produce a more natural-looking blurring affect, we looked into creating scale space images by blurring the image with different kernel sizes given the depth of the pixels within the image. We applied a Gaussian blur to blur the images with different kernel sizes depending on the depth that the image represents and stored these different versions of the original images to be used in the next step of the process.

3.3.2 Depth-based Blurred Image

Now that we obtained scale space images that have been blurred, we then go through each pixel in the original image and look at the depth value to then obtain the correct scaled image from the set of images we had produced in the previous step. By going through the entire image one pixel at a time, we will be assigning each pixel to it's respective blurred pixel value according to the depth value from the depth map. This will then produce an image that is blurred based off of the depth of the items in the image.

3.4. Creating the Final Result

We know where the subject is and what we want to keep in focus in the original image from the Image segmentation mask we created earlier. To maintain this, we cut out the focused subject from the original image such that we can combine this with the depth-based blurred image we created in the previous step in order to complete our final product.

Once we had our depth-based blurred image and our in-focus subject, we can replaced the blurred subject area with the one in focus that we obtained using our mask to then create a final result which should have a subject in focus and the background blurred depending on the depth within the image.

4. Results

With each section, we attempted many different ways to complete the tasks before resulting in our final product.

4.1. Depth Map

Many iterations of creating a depth map led to our current solution that is explained below.

4.1.1 Disparity Map

We were limited on the ways we would be able to create a disparity map using tools that were available to us. In the beginning, we initialized a stereo semi-global block matcher without any other functions and were getting very inaccurate results. When translating this disparity map we were getting very bad results for our depth maps. This led us to explore different options, and we found the Disparity Map Post-Filtering technique that used 2 matchers instead of one. This allowed us to then use a filtering technique that would clean up the disparities and produce a disparity map that we could use for the depth map. One thing we did notice is that the Ground Truth Disparity Map that was provided to us had some occulusions due to the angling of the camera positions which can be seen in Figure 1, where areas are colored black. This led to black spots in the Ground Truth Depth Map that was produced, but with our implementation, we were able to mitigate those issues. Since the filtering was able to see the disparity maps from two views and merge those disparities together, we were able to get rid of these occlusions and produce a somewhat uniform disparity map. This was nice but there were still some obvious issues due to the fact that objects like the cabinets or walls in the background were the same color. This made it difficult for the matcher to distinguish the disparity within these regions and resulted in less accurate disparity values within those regions. The image of our disparity map can be seen in Figure 2.

4.1.2 Disparity Map to Depth Map

Although the formula for translating a disparity map to a depth map is relatively simple, the values that were being used needed to be scaled properly in order to produce a somewhat usable output. We followed the same steps we used to produce the depth map that used the Ground Truth disparity map on our own disparity map but we kept getting results with abnormal lines in the background. After careful evaluation we came to realize that this was due to scaling issues with the focus length and baseline of the camera

Figure 1. Ground Truth Depth Map

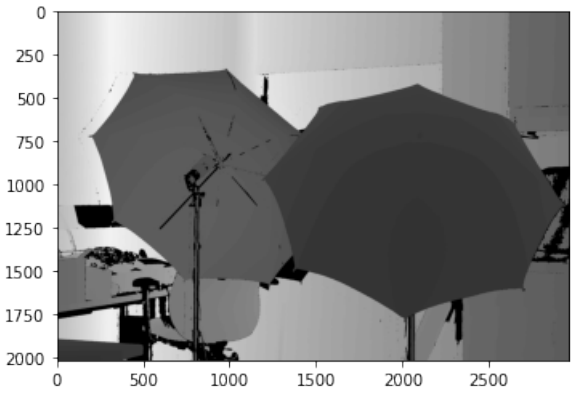
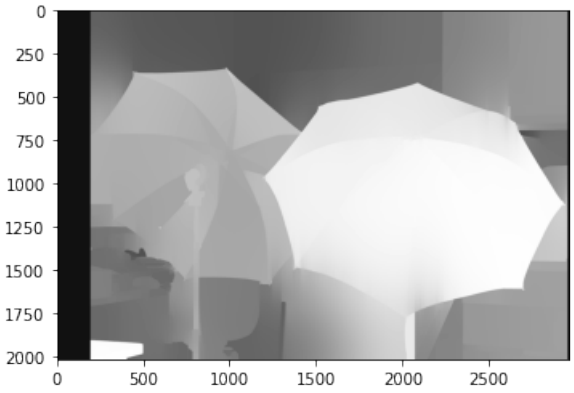
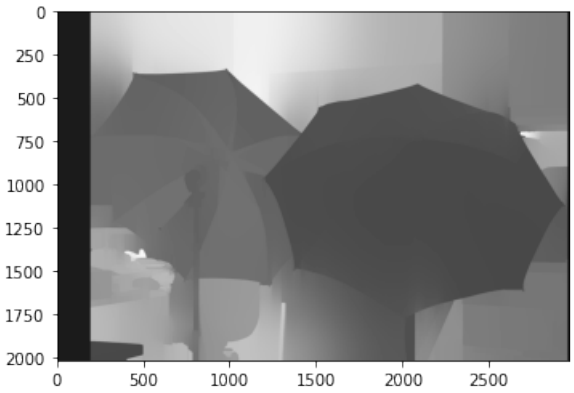


Figure 2. Disparity Map



positions. After scaling the values properly, we were able to obtain our depth map created from the filtered disparity map we created. The final depth map can be seen below in Figure 3.

Figure 3. Depth Map



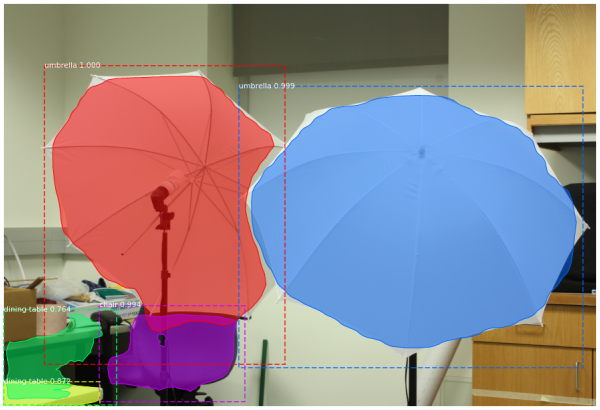
4.2. Image Segmentation

Numerous techniques were implemented such as using a threshold, contouring, and the Watershed Algorithm. All of these were tested before we ultimately decided on the GrabCut Algorithm to detect our subject.

4.2.1 Detecting the Subject

Using the Mask R-CNN model, we obtained masks that detected the different objects in the image, as seen in Figure 4. Then, we determined a rectangle in the center of the image that covers roughly 20% of the image, and created a mask for it, as can be seen in Figure 5a. Using this rectangular mask, we determined which masks from Figure 4 are in the center by checking which of these masks overlap with the central rectangle, and the result can be seen in Figure 5b. The masks are then eroded to make sure that they do not cover areas that are not included only in the subject, seen in Figure 5c, and we can see the placement of these masks over the original image in Figure 5d.

Figure 4. Mask R-CNN Object Detection



4.2.2 Determining the Depth-of-Field

Using the masks from the previous step, we masked the depth map obtained earlier and obtained the minimum and maximum depths of the objects. We went through the remaining masks and determined if any objects are within this depth of field, and in this particular image we did not find additional in-focus objects.

4.2.3 Using the GrabCut Algorithm

With the masks we previously found, we called the GrabCut function implemented by OpenCV to find the foreground. As seen in Figure 6, the GrabCut algorithm using these masks resulted in a pretty accurate foreground image. We still have some background items included in the foreground, which could further be improved by additional background information in the mask we passed into the GrabCut algorithm. The detection and outline of the umbrellas that we want in focus is very sharp and accurate,

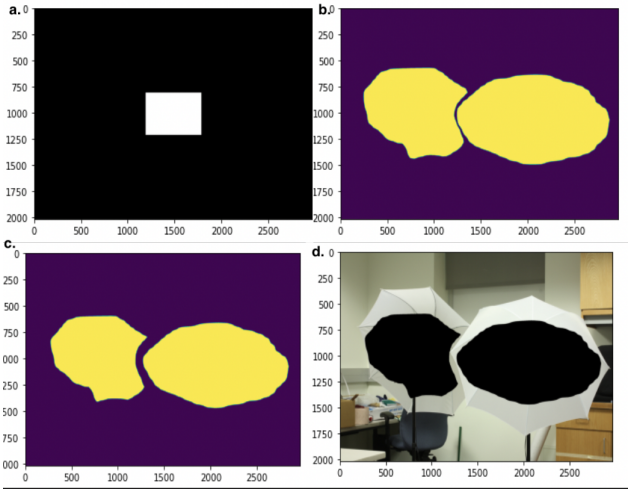
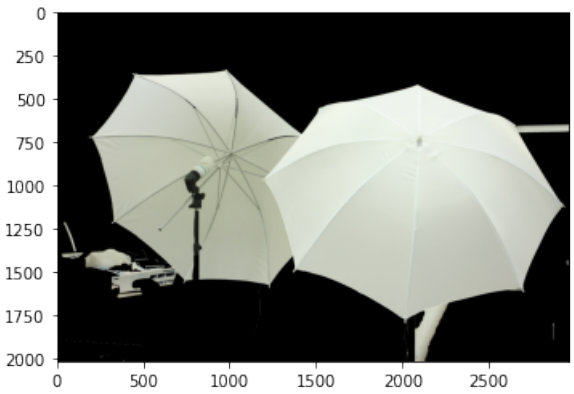


Figure 5. Mask R-CNN Object Detection a. The center items are determined by checking if they overlap with this centered rectangle mask. b. The masks from Figure 4 that overlapped with the centered rectangle. c. The eroded center object masks. d. The eroded masks over the original image.

which is very important for when we stitched this on top of the blurred background.

Figure 6. GrabCut Result



4.3. Gaussian Blurring the Background

After multiple iterations of implementing different techniques, we landed on using scale spaces to accurately represent the depths within the image using the depth map.

4.3.1 Scale Space

In order to produce properly blurred images, we needed to figure out σ, or Standard Deviation values, to use for the kernel size that was to be used for the Gaussian blur. In order to do this, we used the existing camera calibration matrix that was provided to us with the Middlebury dataset and formulated the proper σ value to use for the kernel size.

In order to produce our Camera Matrix P we needed to set up a formula to handle this which is given by

$$P = K[R|t]. \tag{2}$$

P is the camera matrix that we are looking for. K is the camera calibration matrix which we have already obtained from the dataset. R and t are the rotational and translation matrices, but since we are looking at fixed images with no rotation or translation, we can then transform the formula to

$$P = K[I|0]. \tag{3}$$

We then looked at the 3D homogeneous world coordinates which are shown by

$$X_i = (x_0, y_0, z_i, 1)^\top \tag{4}$$

and in our case since we are not translating the image vertically, we set y_0 to 0. Using the Camera Matrix, we were able to transform the homogeneous world point to a homogeneous image point with the following formula

$$x_i = PX_i. \tag{5}$$

Using this newly obtained x which is the projected homogeneous coordinate on the 2D image plane, we could subtract the principal point p_x in order to find the distance between this point in the 2D plane in relation to the principal point. We set the z_i value to different values to represent the different depths within the image to obtain x_i, and we can use the following formula to then find σ.

$$\sigma = \frac{(x_0 - p_x) - (x_i - p_x)}{2} \tag{6}$$

For objects that are closer to the camera than the in focus plane we can represent them using x_{-i} and the formula

$$\sigma = \frac{(x_{-i} - p_x) - (x_0 - p_x)}{2}. \tag{7}$$

After obtaining our σ, we can calculated the kernel size using the following algorithm

if $\sigma * 6$ is even **then**
  kernel $\leftarrow 6\sigma + 1$
else
  kernel $\leftarrow 6\sigma$
end if

We use 6 as our value to multiply by in order to produce a kernel that will efficiently blur the image with 3 pixels on each side of the middle point. With this we were able to create images that were blurred using the respective kernel sizes and store them to use when we were creating our depth-based blurred image.

4.3.2 Haloing

We noticed when we were working on the blurring of images that we would encounter a halo effect around certain objects in the image such as the umbrellas, which can be especially seen in Figure 7 around the white umbrellas. In order to get rid of this effect, we looked into a technique called inpainting. What this would do is fill in parts of an image that have been masked out using the surrounding pixel values to create a realistic representation of what the image may look like without the masked area in the scene. The image that we originally passed into the scale space algorithm was now changed to be this inpainted image instead. The algorithm used for the inpainting method was Telea's Algorithm also known as the fast marching method. When passing this into the scale space algorithm and looking at our blurred set of images we could see that the halo effect had disappeared, as can be seen in the transition from the top image to the bottom image of Figure 8.

Figure 7. Image with Halos

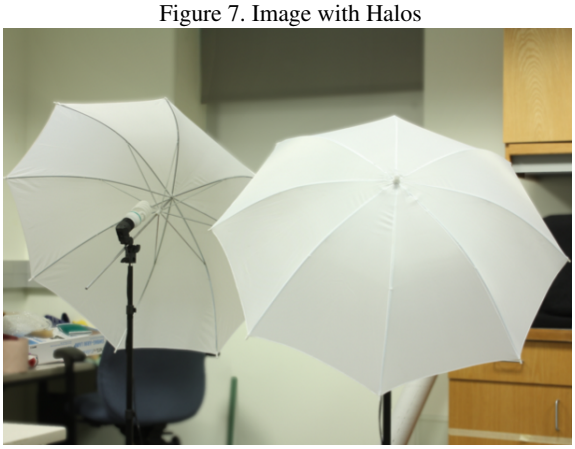
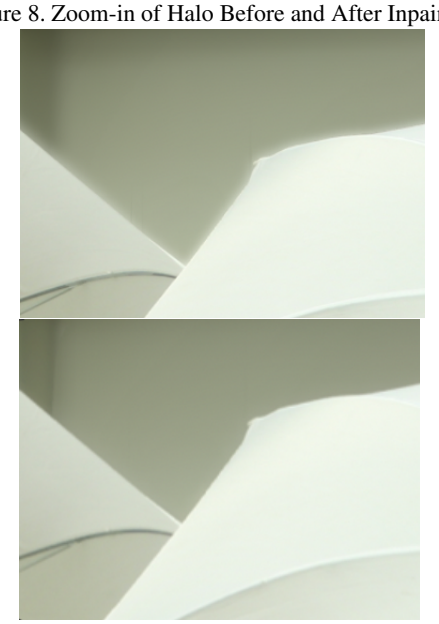


Figure 8. Zoom-in of Halo Before and After Inpainting



4.3.3 Depth-based Blurred Image

After we had obtained our blurred images without any haloing, we moved onto creating our final blurred image based off the depth value. We iterated through the entire image pixel-by-pixel and took the corresponding pixel value depending on the depth which was taken from the depth map. We initially took the pixel value from the closest image that was available but this led to issues with banding within the image. In order to get rid of this, we looked into interpolation and took the two images that were closest to that pixels depth value. This led to a much smoother image that we could then use for the final result.

4.4. Creating the Final Result

Using the mask that was produced from our image segmentation, we created a cutout of the subject that we wanted to keep in focus. This allowed us to keep a cutout that we could later on paste onto the final image where the surrounding background is blurred properly while the subject is in focus.

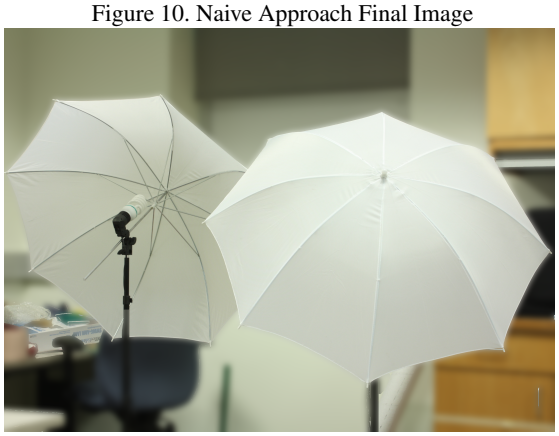
After we obtained our depth-based blurred image and our subject that was in focus, we combined them to create a final image. We began by making a cutout of where we were going to paste the in focus subject from the depth-based blurred image such that the in focus subject that was going to be pasted on would not combine with the existing pixel values in the location. After the subject was cut out, we pasted in our in focus subject and were left with the final result. We cropped the image to get rid of pixels that were incorrect due to the occlusions and we were left with the final complete image which can be seen below in Figure 9.

Figure 9. Final Stitched Image

5. Qualitative Experiment

To prove our solution mimics a professional camera in a better way than the naive approach we observed earlier in the paper, we implemented both of the solutions. The naive approach was implemented by simply blurring the background uniformly and pasting the foreground (segmented in the same process as with our solution) onto the blurred background. We see the results of the naive approach in Figure 10.

Figure 10. Naive Approach Final Image



The naive solution had a background that is uniformly blurred with a kernel window of size 101x101, which is between the highest and lowest kernel sizes of our solution. Looking at this image in comparison to our final result in Figure 9, we can see that the background in the middle where there is the dark curtain hanging from the top is blurred more heavily in our result than the naive solution, and the wooden shelves on the right of the image are more in focus in our image than the naive approach. This aligned with the placement of these objects, where the background curtain is much farther away from the camera than the wooden shelves on the right. With these details, we could see that our approach mimics a camera's depth-of-field with the umbrellas in focus and the background becoming more blurred as its distance increases.

6. Discussion and Future Work

We produced a result that worked for this dataset, but not all datasets will contain the necessary information such as the calibration data needed in order to perform all of the necessary steps. We plan to work on adapting the process to all types of stereo images with many different subjects and refining the process in order to produce results with any set of stereo images. The Depth Map also struggled with certain areas in images that had a lack of features such as a wall as the matcher was not able to distinguish properties to align the disparities resulting in less accurate depth values. Another issue that arises when creating disparity maps is the similarity in colors. When certain pixel values are similar in color, it creates a harder problem for the matcher to create a disparity map.

In terms of the image segmentation to detect the subject, the best solution we found after experimenting with a few other solutions still resulted in some parts of the background being added to the subject mask. Since this part really affects the final result as all areas of the mask will be fully in focus, improvements in this step would greatly benefit the project. Using the same GrabCut algorithm, we can make improvements in the mask we provide by adding in pixels that we can confidently say are in the background of the image, or also providing a rectangle to the algorithm as an area of interest. We could also explore different algorithms to see if a different approach would improve our results. Another approach we think will improve this step is to use the depth map information and possibly combine the thresholding approach with the GrabCut results to see if we can remove additional background pixels from the subject.

The blurring portion of the project also posed some difficulties that could be improved upon in the future. The way we determined the standard deviations and kernel sizes of blurring was all set in the image space, and we should be doing the calculations in the frequency space by doing a Fourier transform before the blurring. Our implementation works for a small amount of blurring when setting $x_0 = 1$, but for larger values and more depth in the image, we need to implement the Fourier transform portion of the calculations.

With these small improvements, the conversion from stereo images to a synthetic depth-of-field image would create very professional-looking images.

7. Conclusion

Through our experiment we were able to take a completely in focus set of stereo images and extract a depth map that could be used to simulate a depth of field within the image, creating a more professional look in the image. We produced the disparity map and transformed it into a depth map, segmented out our subject from the rest of the image, and applied a proper blurring affect in relation to the depth found within the image. After stitching all the results we were left with a final product that would mimic that of a professional camera with a lens that allows the user to change the depth of field. We discussed certain issues and improvements that could be made in order to produce better results in the future as well.

References

[1] Waleed Abdulla. Mask r-cnn for object detection and instance segmentation on keras and tensorflow. *GitHub repository*, 2017. 1

[2] David Eigen, Christian Puhrsch, and Rob Fergus. Depth map prediction from a single image using a multi-scale deep network. *arXiv preprint arXiv:1406.2283*, 2014. 1

[3] Neal Wadhwa, Rahul Garg, David E Jacobs, Bryan E Feldman, Nori Kanazawa, Robert Carroll, Yair Movshovitz-Attias, Jonathan T Barron, Yael Pritch, and Marc Levoy. Synthetic depth-of-field with a single-camera mobile phone. *ACM Transactions on Graphics (ToG)*, 37(4):1–13, 2018. 1

[4] Lijun Wang, Xiaohui Shen, Jianming Zhang, Oliver Wang, Zhe Lin, Chih-Yao Hsieh, Sarah Kong, and Huchuan Lu. Deeplens: Shallow depth of field from a single image. *arXiv preprint arXiv:1810.08100*, 2018. 1