DEPTH-DRIVEN AUGMENTED REALITY

Kristyn Wooldridge, Gilbert Yap

Boston University
Department of Electrical and Computer Engineering
8 Saint Mary's Street
Boston, MA 02215
www.bu.edu/ece

Apr. 30, 2021

Technical Report No. ECE-2021-01

Summary

Virtual reality and augmented reality experiences continue to be a growing trend for both at-home media and entertainment locations. In this project, we are exploring the generation of 3D scenes from 2D images along with depth information through a technique known as depth image-based rendering. This implementation is of interest since the 2D image and depth data can be generated by modern smartphones such as the iPhone X from 2017. Implementing this technique promotes consumer augmented reality experiences that allow added-in 3D objects to interact more realistically with the environment they are placed within. In this project, we explore the limitations of this "virtual" image generation technique quantitatively by identifying techniques for replacing regions of missing information. We then qualitatively determine the point where computational errors are ignored by the human visual system when viewed in a VR headset such as Google Cardboard. We also explore how images of objects can be realistically placed within the scene by attempting to appropriately superimpose a captured RGB + Depth image of a 3D object onto another captured image of an environment, and generate a realistically perceived 3-Dimensional space from the 2 RGB + Depth images.

This project has been completed as one of the requirements in ECE course "Digital Image Processing and Communication" (EC 520).

Table of Contents

List of Figures

1 Introduction

1.1 Processing of 2D RGB and Depth Data into 3D Space

3D and virtual reality media continue to be promoted by entertainment studios as the future of the industry. One implementation of 3D viewing for television and virtual reality is the process of generating, displaying, and isolating a “stereo pair” of images to each of the viewer’s eyes. This adds a sense of depth due to the overlap of common parts of both images and differences in the viewing angle of each eye. When working with 3D models and geometry, it is trivial to generate two views of the subject or scene that would mimic what the human eyes would see. However, when 3D geometry is not available, it is desirable to create these 3D models based on 2D information such as camera pictures. The research of image-based rendering, and its extension depth image based rendering (DIBR), aims to create 3D scenes and objects from 2D images and auxiliary information such as depth maps.

In DIBR, generating an initial pair of stereo images from a single 2D image is thoroughly researched and implemented practically. However, the process of mapping RGB information to the depth map then remapping it to a newly generated 2D space that is a horizontally shifted version of the original image creates two issues. The first issue is the occurrence of “holes” that are a result of remapped pixels not having RGB and/or depth information in the generated image. The other issue is the occurrence of overlaps within the generated image, where two RGB pixels are occluded due to the transform of the depth information. These two issues can create false edges or “ghosting” within the image pair, which detracts from the viewing experience.

There are several linear and non-linear approaches that attempt to remedy the two issues based on the RGB data, depth data, or both. From a quantitative view, it is possible to minimize errors based on a ground truth image, but qualitatively the stereo pair of images can exhibit a certain amount of error and still be perceived as “good” quality due to behavior of the human visual system.

1.2 Augmented Reality Application

Augmented reality techniques can be used to enhance the viewer's three-dimensional image experience. Once a 2D RGB environment image with its corresponding depth map is captured, that environment can then be populated with new 3D objects originating from 2D RGB images and depth maps captured by the same smartphone hardware. This is done by superimposing images of each new object onto the environment image. The goal is to realistically integrate the 3D objects into the virtual 3D environment. Methods to make the integration more realistic include: applying the same 3D image generation techniques as discussed previously to ensure that the new objects are rendered into the space with undetectable distortion, properly positioning and scaling the new objects into the scene, and aligning the object's depth map with the environment. The combination of RGB and depth information allows the placed object to appear more three-dimensional in the rendered environment.

2 Literature Review

2.1 Generation of 3D Image from 2D RGB+D Data

When provided with image data along with depth information, Sun et al. [1] define the process of DIBR as rectification, parallax image creation, virtual view rendering, post-processing, and de-rectification. Image rectification is the process of aligning a set of images of varying views based on a common point or line within the images. Parallax image creation and virtual view rendering is the process of mapping the 2D images into a common 3D space and then projecting the 3D space onto a view that is coplanar with an original image but shifted horizontally by some distance. This shift generates disparity for each of the pixels in the new image. Post-processing is done to clean the newly generated parallax image and treat sections of the image that create discontinuities.

The generation of the shifted image for the stereo-image pair is described in Vasquez et al. [2]. This process describes the generation of a virtual image taken by a virtual camera, at a given distance from the real image, taken by the original camera. Given an observed sampled image **I** and depth map **Z** sampled on the same orthonormal lattice, a new image that is parallel shifted to the right of the real view can be generated:

$$I = \{I[n], n = (n_1, n_2)^T \in \Lambda\} \quad (1)$$

$$Z = \{z[n], n \in \Lambda\} \quad (2)$$

$$I_\alpha(n_1 + d[n], n_2) = I[n_1 n_2] \quad (3)$$

d[**n**] represents the disparity vector, which is dependent on *B*, the distance between the optical centers of the real and virtual camera capturing each image, and *f*, the focal length of the optical system. It is calculated by:

$$d[n] = -f\frac{B}{z[n]} \quad (4)$$

For this project, and as described in the article, the vertical component of the disparity vector is null since the new image is only horizontally shifted. This is due to the assumption that the virtual image's optical center is parallel with the input image and that the virtual image shares the same horizontal axis. These equations show that the disparity

vector used to generate the horizontally shifted virtual image is calculated by the observed image depth map, where the greater value of z[**n**] results in a smaller, but negative value of the disparity.

2.2 Processing of Secondary View Image

Vasquez et al. [2] review and implement a method for reducing the probability of holes in the virtual image through depth map smoothing and four post-processing techniques that help remove holes within the virtual image due to disparity discontinuities. In our project, we will be replicating some of these methodologies in addition to testing alternatives. The first approach is to fill in “holes" with a constant pixel value that is determined by neighboring pixels in the newly generated RGB image. The color to fill in the hole is determined by taking the average color of the pixel boundary for the image **I**, constant color, **C**, hole **H,** and hole boundary ∂**H**:

$$I[n] = C, \forall n \in H \tag{5}$$

$$C = \frac{\sum_{\forall m \in \partial H} I[m]}{\sum_{\forall m \in \partial H} 1} \tag{6}$$

The second approach is to smooth the depth map by using a 2D Gaussian filter so that there are no sharp edges, which minimizes the likelihood of a hole being made. This method is outlined by Zhang [3] where the Gaussian filter $g(x, \sigma)$ is defined as:

$$g(x, \sigma) = \frac{1}{\sqrt{2\pi\sigma}} exp\ exp\ \left\{-\frac{x^2}{\sigma^2}\right\}, for - w \leq x \leq w \tag{7}$$

Where w is the window size and σ is the standard deviation.

The third and fourth approaches are to utilize the median filter and morphological erosion operation. These two operations may be suitable methods for removing holes since both may be used on “salt noise”, which is the appearance of constant peak pixel values randomly within an image. Median filtering is the replacement of a pixel with the median of its N x N neighborhood, while morphological erosion replaces the pixel with the minimum value in the N x N neighborhood.

The final approach utilizes a built-in MATLAB function *inpaintExemplar* to perform the hole filling. This function implements an exemplar-based inpainting algorithm proposed by Criminisi, et. al [4]. This function is given the virtually generated image with holes, and a mask that specifies the hole regions to be filled. It then determines the source region (non-hole regions of the image) that will be used to fill the image. For each pixel at the boundary of a hole, a patch is centered over that pixel, and the patch priority is determined. Then, each hole is filled, in order of the patch priority. This method was chosen because it presents a comprehensive method for inpainting that has been implemented in MATLAB for image processing.

3 Implementation

This project is divided into several stages. The figure below shows the high-level workflow diagram with each stage represented as an individual block in the diagram. We describe the steps taken for each stage in detail in the subsequent sections.

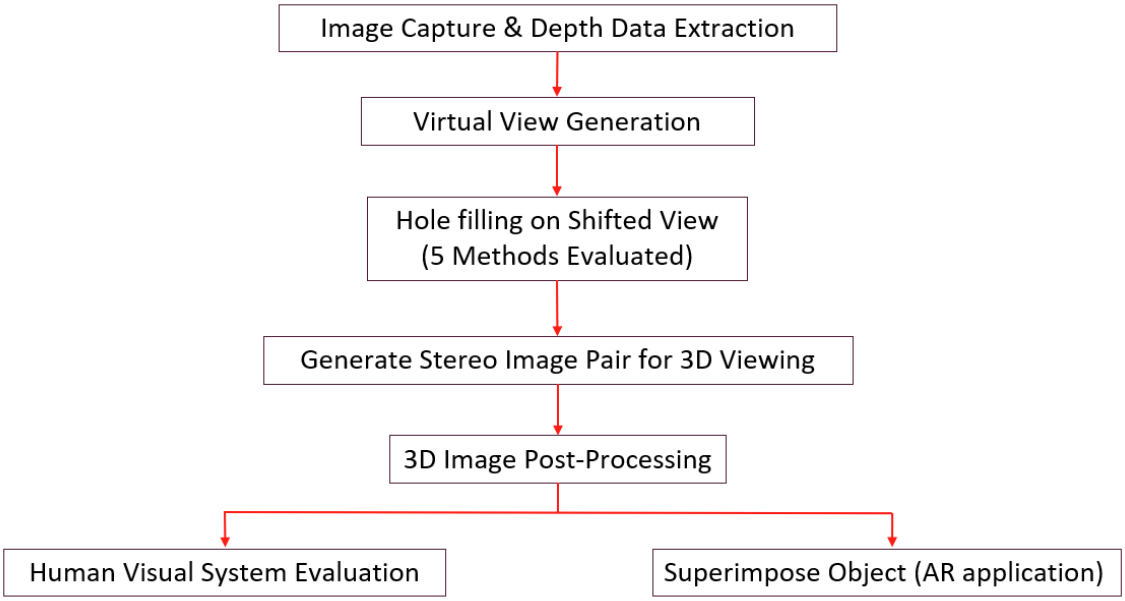

Fig. 1 Project Workflow Diagram

3.1 Image Capture and Depth Data Extraction

The first step in implementing our project was image capture and depth data extraction. We needed to collect an RGB image and corresponding depth data to generate a virtual shifted image using Depth-Based Image Rendering. Images were captured using PortraitMode on the iPhone X. PortraitMode captures an object's depth and blurs the background for a "Portrait" effect. Once the portrait photo was captured, the iPhone Photos App allows the background blur to be removed with a simple toggle, which suggests that the blur on the image is a real-time image-processing effect rather than a blur effect captured in the original image. We removed the background blur on each image captured to have a clear picture of the entire object and its surroundings.

Next, the depth data from the portrait photo needed to be extracted. This was done using “exiftool”, a program that is used for editing meta information from the .jpg files captured by the iPhone camera. This program was run on computers running Windows Operating System. See Appendix 6.1 for a step-by-step process of the depth map extraction procedure.

The depth map obtained using exiftool was 5.25x smaller in both width and height than the RGB image. To remedy this, we used the MATLAB function *imresize(),* which uses bicubic interpolation to resize an image matrix. Both the RGB and Depth images were resized to be the average size between the two, so that a single image was not compromised by size reduction or expansion. The resized RGB and depth map images were overlaid on each other in a visual check to ensure that all edges of the object lined up correctly. Once we obtained two images (an RGB image and corresponding depth map) of equivalent sizes, they could then be used to generate a virtual image to be used for the stereo-image pair.

3.2 Disparity Calculation

Once we obtained the depth map for each image, we then calculated the disparity using Equation 4. In the case of an iPhone XS, the focal length f of the lens is 6mm. Initially, we considered the value for B to be the distance between the left and right eye pupils, which is approximately 60mm. The values of d[n] were not all integer values, so shifting the RGB pixels by the disparity vector would result in points on a non-orthonormal lattice. Vasquez [2] remedies this by performing a nearest neighbor calculation, whereas we chose to simply round the disparity vector values. The virtual image can then be generated by applying the disparity calculation to each pixel in the original image, as stated in Equation 3. Applying the disparity map from section 2.1 directly to the original 2D RGB image may result in two issues.

The first is that two pixels can potentially be mapped to the same pixel location in the virtual image. When this occurs, the pixel with depth associated with being closer to

the camera is chosen. The closer pixel will occlude the farther pixel and no discontinuity will arise in the resulting image. The second issue is when a pixel is mapped to a new location and another pixel does not take its place. For example, in a row of 3 pixels with a disparity map that causes a left shift of 1 to pixel 1, 2 to pixel 2, and 2 to pixel 3, the second and third pixel will be undefined since information about pixel 4 and pixel 5 do not exist. Resolving these pixels is non-trivial, so rather than apply one solution, these "holes" are marked by a value that exists outside of the 8-bit data range so that their positions are distinct from regular color pixels. This results in an image with holes marked by a particular value that are to be filled.

There are numerous ways to fill the holes in the virtually generated images. For this project, we evaluated five different hole-filling methods:

1. Constant Color Fill
2. Depth Map Smoothing
3. Median Filter
4. Morphological Erosion
5. Inpainting (MATLAB function)

Each method is detailed in the subsequent sections.

3.3 Hole-Filling Method 1 - Constant Color

The constant color filling method outlined by Vasquez et al. is one of the simplest methods for filling holes, utilizing only RGB information in the virtual view image. As stated in the Literature Review section, the average color outside of a hole pixel is used to fill that hole. By dividing the image into large (bigger than the hole) disjoint blocks, holes can be entirely filled by the average color in their boundary. However, this can result in unnatural portions of the image due to large holes being only one color. If the blocks are made smaller and more numerous, large holes are broken up into smaller sections and can be filled with a better approximation. At the same time if too small of a window is used, then there runs the risk of a block being made up entirely of hole pixels that cannot be filled due to the lack of color information surrounding the block.

This issue was resolved by performing multiple passes on the image, starting by going through the image with small blocks and then increasing the block size on later passes. This closes small holes in initial passes and then closes large holes with more natural transitions in later passes. We created an algorithm for picking block sizes that first finds the largest number of blocks the image can be divided into. Then, if the image can be perfectly broken down into half those number of blocks (doubling the block size), that is picked as the next block size. If it cannot, then the block dimensions are each increased by 1 until a perfectly divisible size is found. This process is repeated until there are 4 sets of sizes. A fifth block size is found by finding the fewest number of blocks the image can be divided into that is greater than 1.

3.4 Hole-Filling Method 2 - Depth Map Smoothing

The second hole filling method we used was the depth map smoothing. As it is titled, this method involves applying a smoothing filter to the depth map before it is resized and used to generate the virtual image. The goal of filtering the depth map is to remove sharp edges or discontinuities in the depth map to reduce the amount of occlusion (which results in holes) in the virtual image.

Two filtering methods were initially applied: Equiripple Filtering, and Gaussian Filtering. The Gaussian smoothing resulted in a much smoother depth map, so that method was used for the depth map smoothing. A square Gaussian filter with varying standard deviation, σ was utilized, with values of $\sigma = 2$, 8, and 16. This was done using *imgaussfilt* in MATLAB.

3.5 Hole-Filling Method 3 - Median Filter

The next method evaluated was applying a Median Filter. One observation made during this project was that small holes within the generated image are similar to “salt noise” from images with a faulty sensor. By characterizing the holes as salt noise, the

median filter can be used to attempt to close the holes. The filter was applied using *medfilt2* in MATLAB on the entire image.

3.6 Hole-Filling Method 4 - Morphological Erosion

A similar method to median filtering is the morphological erosion operator. Whereas median filtering replaces the current pixel with the median of its neighborhood pixels, erosion replaces the current pixel with the darkest pixel within its neighborhood. This operation is typically performed on binary images, but can also be applied to individual color channels of an RGB image. The erosion on the image was done by writing code to loop over each section, and fill it using the MATLAB function *imerode*.

3.7 Hole-Filling Method 5 - MATLAB Inpainting

The final method evaluated for hole filling is the built-in MATLAB function *inpaintExemplar()*. This function applies an inpainting algorithm (as described in [4], and in the Literature Review) to the right virtual generated view to fill in the holes. The function takes two input arguments: the RGB image to be filled, and a mask, which denotes the regions of the image to be filled. We obtained the mask by creating a logical matrix of the RGB image, whose hole values are already denoted by a value not within the 8-bit range. This is the same method used to denote the holes in the Constant Color Fill method. The mask is generated by using a logical operator to compare which pixels are assigned to the “hole value”. Once the mask is obtained, it is then passed with the RGB image to the MATLAB function.

3.8 Stereo Image Pair Generation

Once the virtual image has been obtained, it needs to be presented next to the original image to be viewed in 3D, thus generating the stereo image pair. We used two different methods to achieve this. The first method used was generating an image pair used to be viewed using the Google Cardboard, which uses a pair of 34 mm x 45 mm biconvex lenses. The stereo image pair for this method was obtained by placing the left image (original RGB image) next to the right virtual generated image using MATLAB.

The second method used was generating a red/cyan stereo anaglyph image using the MATLAB function *stereoAnaglyph()*. This function takes the right virtual and left original image, and generates the anaglyph image using the two. The 3D image can then be viewed using red/cyan glasses. The 3D images produced by both methods can be found in the Experimental Results Section.

3.9 Human Visual System (HVS) Evaluation

In addition to the noticeable 3D effect within virtual reality headsets such as Google Cardboard, certain holes and cracks within the virtual image were not as apparent when viewing the stereo image pair. This can be due to several factors such as eye dominance and spatial sensitivity. These HVS properties can be used to simplify virtual view processing or the storage of virtual view images.

For this project, the original image was always placed into the portion of the display that corresponded to the left eye while the virtual image was placed in the portion corresponding to the right eye. Through experimentation, we found that once the image pair was viewed in the headset, if one were to close their left eye for a second or longer and then reopen it, the distortions of the right image became more obvious, as if the two images were transparency slides and the virtual view was being placed on top of the original image. The converse was also true - closing one's right eye for at least one second before reopening caused the original image's details to be more apparent. This is a demonstration of forcing eye dominance, where the human visual system relies on more information from one eye than the other. We can take advantage of this phenomenon and allow more aggressive blurring in the virtual view to hide holes and cracks without the need for non-linear processes such as constant color fill that can be more computationally expensive.

The application of a Gaussian blur on the virtual view was tested at varying standard deviation values to determine a tolerance for the amount of "distortion" in one

image when the pair is viewed. The amount of smoothing applied to the right virtual image was gradually increased, until there was a noticeable disturbance to the 3D view. This experiment was performed on a small group of people available, and it was found that a Gaussian blur with standard deviation below 3 is the threshold for the amount of blur that is relatively undetected when the image pair is viewed. Any value above that produces a noticeable blurred distortion in the 3D image.

3.10 Augmented Reality - Superimposing Objects

Finally, we implemented an Augmented Reality application by placing a captured RGB+D image of an object into an empty environment scene, and rendering the resulting image in 3D. This was done to model what is done in common AR applications, in which an object is placed over a "real world" view. In AR, this real world view is often through the smartphone camera. We did not use live camera data for the scope of this project, so instead we use a still image of the empty environment.

The first step in object placement is extracting the object from the captured image. This was done using the depth map to separate the foreground from the background of the image, assuming the object is placed in the foreground. Pixels that correspond that are greater than a given depth threshold are flagged with a -1 as being part of the background. This results in a matrix of the same size as the image, with the background removed (appearing black in the rendering).

The background removal alone is not sufficient to detect the entire object, as the counter that the object was placed on is in front of the object with respect to the depth map measurements. Therefore, the orange color of the counter was taken advantage of in removing the remaining part of the image, leaving only the object. To eliminate the counter surrounding the image, a snapshot of the orange counter only was taken, and the mean color value analyzed from that image, to get an "average color" for the counter. The standard deviation was then calculated for each RGB component on the counter sample, and used as a baseline for a threshold in determining the counter color. The RGB

object image was then compared to this mean orange color, and pixels that were within the threshold of the color were flagged as -1 as well (and rendered black). The combination of the depth map elimination and RGB color comparison elimination result in an RGB image that only contains the object (and its shadow). The object depth map needed to be extracted as well. This was done with the same method so that the depth map and RGB images align.

Finally, the extracted object RGB image and Depth Map were superimposed on the environment's RGB image and depth map and rendered in 3D as an image-pair. Because the extracted object images have a -1 value for their background, the superposition was done by replacing the environment image pixels with the corresponding object pixels at all values not indicated with a -1. This project assumes that the environment image and object image/depth map are the same size, and the position of the object is not manipulated. This process resulted in a superimposed speaker image on the empty countertop. To evaluate the quality and how realistic the superimposed speaker image was, the anaglyph and 3D stereo image pair was visually compared to a 3D generated image of the actual speaker placed on the counter. The actual placed speaker 3D image was used as a baseline to determine any noticeable distortions or differences in the virtually placed object.

4 Experimental Results

4.1 Image Extraction and Scaling

The first step in this project was the RGB image capture, and depth map extraction. Figure 2 shows an RGB captured image of an orangutan, and its corresponding depth map.

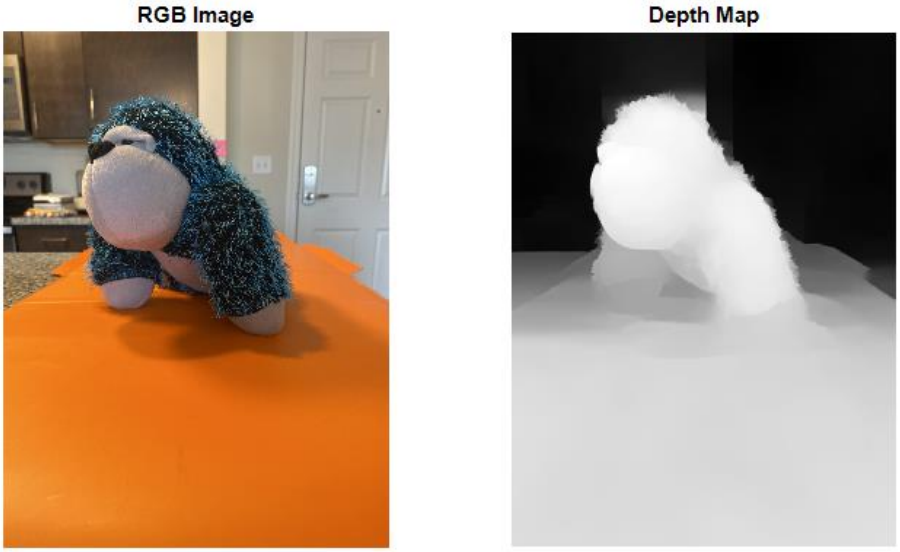

Fig. 2 RGB Image and Extracted Depth Map from iPhone Captured Image

The extracted depth map is a grayscale image that displays closer objects in lighter shades, with white being the closest, and farther objects in darker shades.

The unprocessed extracted depth map was not the same size as the RGB image. In our cases, the RGB image was 5.25 times larger in pixel size than the depth map. To accommodate size discrepancy, we used the MATLAB function imresize() to resize both images at the average size between the two images. Imresize uses bicubic interpolation by default to generate the resized images. One aspect to note is the gradient coloring of the depth map. The orangutan's head appears to be a lighter shade than the counter that it is sitting on, which would suggest that the head is closer than the counter. This is untrue in the real world, as there is counter space in front of the object. Additionally, the

extracted depth map seems to have some distortion artifacts, especially near the orangutan's head. These slight artifacts can lead to inaccurate shifting in the virtually generated image, although our results will later show that the impact visually is minimal.

4.2 Disparity Calculation

The extracted depth data for each image, was utilized to calculate the disparity, as described in Section 3.2, and then generate a right shifted image based on that disparity.A positive correlation existed between the numerator of the disparity and the number of holes in the virtual image. The below figure shows an original RGB image next to its depth map, followed by the virtual right view before hole-filling. The bright green pixels represent the holes in the image. As noted by the depth map, the bottle on the bottom right is the closest and also suffers the most pixel disocclusion. The disparity calculation does indeed shift closer pixels more than farther ones, which is more likely to result in pixel disocclusions in the virtual view.

Fig. 3 Original Left View, Original Depth Map, and Virtual Right View with No Hole-Filling

4.3 Hole-Filling Method 1 - Constant Color

The first hole-filling method implemented was the Constant Color Fill. The figure below shows a test image with a manually created hole, and that hole being filled. The

filled image shows the result of 5 different block sizes applied, based on the algorithm described in the previous section.

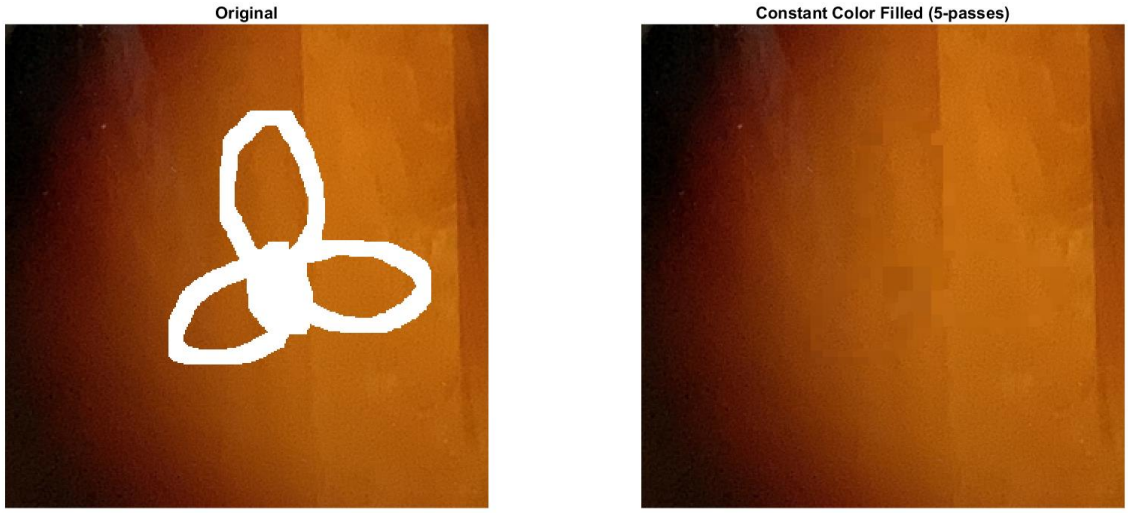

Fig. 4 Testing Image with a Manually Created Hole (left) and the Same Image with the Hole Filled Using Constant Color Fill (Right)

It was found that images with small holes relative to the size of the image are remedied with few perceptual anomalies. Images with smooth color regions also work well with this method since hole pixel neighbors have very similar values, making the hole-neighborhood mean a suitable replacement value for the hole. Hole-filling along object boundaries or edges can create more obvious distortions. Our test images' dimensions are similar to the resolution used in consumer virtual reality headsets, so if used in real-time applications such as augmented reality, this method may be unsuitable since for our test images, the algorithm takes about 1 second to fill all holes. This can be sped up by reducing the number of passes, but this will sacrifice the quality of the resulting image.

The figure below shows the Constant Color Fill method applied to a captured image's right generated virtual view.

Fig. 5 Comparison of Generated Virtual Image without Processing (left) and Generated Virtual Image After Constant Color Fill (right). Hole pixels are represented by green

4.4 Hole-Filling Method 2 - Depth Map Smoothing

The Gaussian depth map smoothing resulted in the generated virtual images as shown in the figure below. An increasing value for σ did result in a reduced amount of holes (white pixels), however the highest value σ = 16 used still left many holes in the virtual image. Further exploration of increased values of σ will be explored to determine if this method alone can result in minimal holes and distortion on the image, as well as applying different smoothing along the horizontal and vertical axes of the image, based on the direction of the edges present.

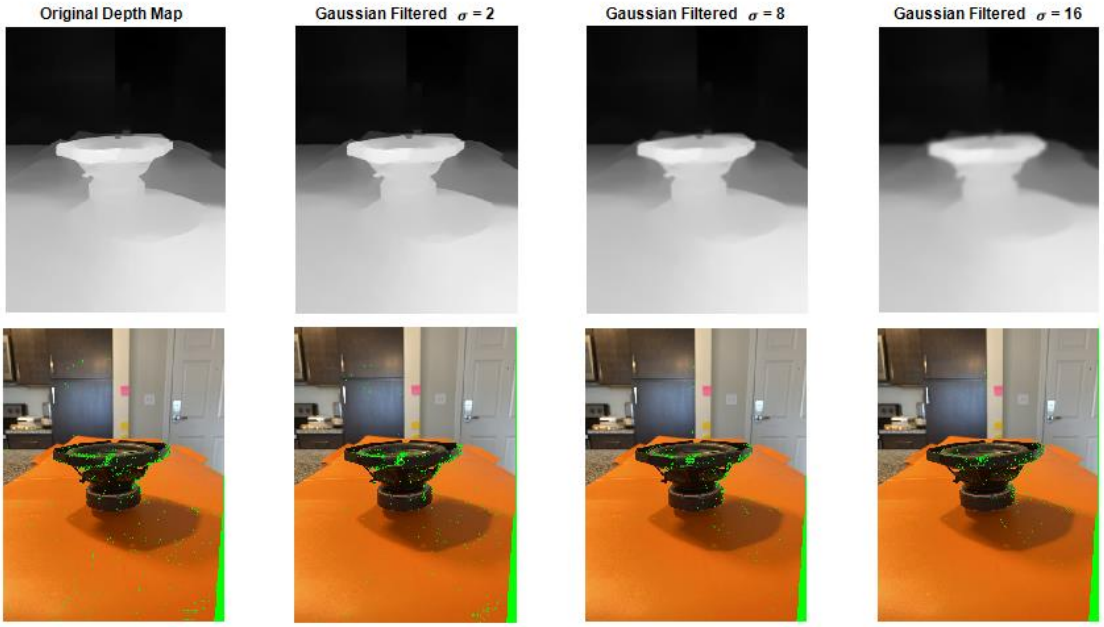

Fig. 6 Virtual Right View Generation with Depth Map Smoothing, with Increasing Standard Deviation

4.5 Hole-Filling Method 3 - Median Filter

For the Median Filter hole-filling method, we tested the 3x3 and 6x6 median filter on the virtual image and found that the 3x3 filter closed the smaller holes and cracks well, but left behind medium and large holes basically unchanged. The 6x6 filter did a better job of also closing medium holes, but still left large holes in addition to blurring and distorting fine image details.

Fig. 7 Virtual View with 3x3 Median Filtering (left) and 6x6 Median Filtering (right)

A combination of median filtering and constant color fill was also tested. The 3x3 median filter was applied to the generated image first, followed by constant color fill (with 5-passes). The resulting image looked somewhat similar to the constant color fill method, but some image details in medium and large holes were distorted. Computationally this is also more expensive than running the 5-pass constant color fill by itself.

4.6 Hole-Filling Method 4 - Morphological Erosion

The result of the morphological erosion method is similar to median filtering, and thus suffers many of the same issues. One notable issue was the “bubbling” of letters within images as a result of multiple erosion passes. In the below figure, three sections have been circled in blue to show distortions to areas in the image with sharp edges that either grew or shrank in undesirable ways.

Fig. 8 Virtual View without Hole-Filling (left) and Virtual View Processed by Morphological Erosion (right)

4.7 Hole-Filling Method 5 - MATLAB Inpainting

The final method evaluated for hole filling is the built-in MATLAB function *inpaintExemplar()*. We passed the right virtual view image matrix with holes as an input, along with a logical matrix of the same size of the RGB image, indicating the hole locations to be filled, as shown in the figure below.

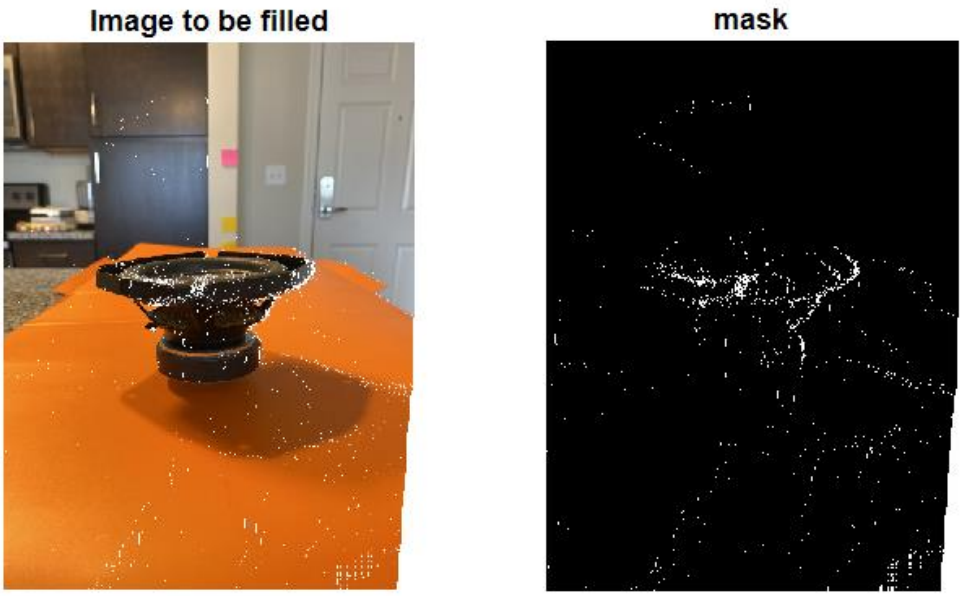

Fig. 9 Right virtual view RGB Image (left) and Corresponding Mask (right)

The results from this method are shown in Figure 10. The inpainting method performed well, however it had a very long run time, due to the algorithm having trouble determining optimal patches from the mask to be filled. For other images, such as the orangutan, the function would result in an error, saying it was unable to determine a patch to fill the image. This is likely due to the irregular shape of the holes in the mask to be filled, rather than having a mask that can be approximated by larger, more polygon-shaped patches.

Fig. 10 Speaker RGB Image filled with MATLAB Inpainting Method

The result of the MATLAB inpainting method compared to the Constant Color Fill is quite similar, and due to the long run time of the MATLAB inpainting, and failure to detect an appropriate patch (and thus resulting in an error), the Constant Color Fill method is more preferred.

Fig. 11 Constant Color Fill and MATLAB Inpainting Resulting Image Comparison

4.8 Stereo Image Pair Generation

The figure below shows a stereo image pair generated, with the constant color fill method utilized in processing the generated right virtual image. Here, the left image is the original captured image, and the right image is the right virtually generated view. The right view has a noticeable distortion at the orangutan's nose. However, when the image pair is viewed with both eyes, this distortion is less severe, especially if the left eye is dominant in viewing this image. This effect will be further analyzed in the Human Visual System Evaluation section.

Fig. 12 Stereo Image Pair for Google Cardboard Display

The figure below shows the stereo anaglyph image for the orangutan. The anaglyph image produces a noticeable 3D effect on the object, especially in areas where it is closer to the viewer (e.g. the head and front arm). However, the color fidelity is lost with the anaglyph glasses. While the primary goal is to render 3D images using the

Google Cardboard, this method allows viewers to view the 3D image if they do not have access to the Google Cardboard or similar product.

Fig. 13 Stereo Image Pair as a Red-Cyan Anaglyph

4.9 Human Visual System (HVS) Evaluation

To evaluate the effect of the human visual system on depth perception and distortion, we applied a Gaussian blur to the virtual view at varying values of standard deviation. Figure 14 shows a comparison of the virtually generated view without hole filling, and the same image after Gaussian blurring. The comfort level of 3D viewing with a Gaussian blurred image at varying standard deviations was tested.

Fig. 14 Virtual View without Hole Filling (left) and Virtual View with Gaussian Blurring with $\sigma = 3$(right)

In our small sample size of testers, we found that we could set the standard deviation of the virtual image by as much as $\sigma = 2$ or $\sigma = 3$ before there was an uncomfortable smearing effect when viewing the stereo image pair. When viewing the blurred image outside of a headset, the distortion appears very high, but once viewed in the headset, the HVS is able to trick viewers into seeing most of the details from the original image instead while simulating a 3D image. The discomfort is similar to the type of blurry vision as a result of waking up or of rubbing one's eyes. Kernels larger than 6x6 for the median filter or morphological erosion could also be used.

Blurring the virtual view also has the benefit of allowing for better compression of the virtual view for storage. Blurring images has a similar effect to low-pass filtering, which reduces the dynamic range of pixel values. This compressed range makes it easier to compress with either techniques such as basis restriction or DPCM.

4.10 Augmented Reality - Superimposing Objects

The results of the process of detecting the object from its captured image, and removing the background is shown below. The left-most images are the original image depth map and RGB image. The middle column shows the depth map after the background pixels are removed, and the RGB image after the orange countertop pixels are removed. The combination of the background and countertop removal results in the object isolated from its captured environment. Due to the fact that we were removing the countertop based on color, the speaker shadow could not be removed, as it was not detected as being in the object background, nor was it within the threshold of the color orange that signified a countertop pixel. This was not an issue for the scope of this project, because the speaker was being placed in the same position as where it was captured. However, further methods for object detection would need to be implemented if the speaker were placed in a different environment or position than what it was captured in. Figures 16 and 17 show the superimposed RGB and Depth Images for the object in the environment.

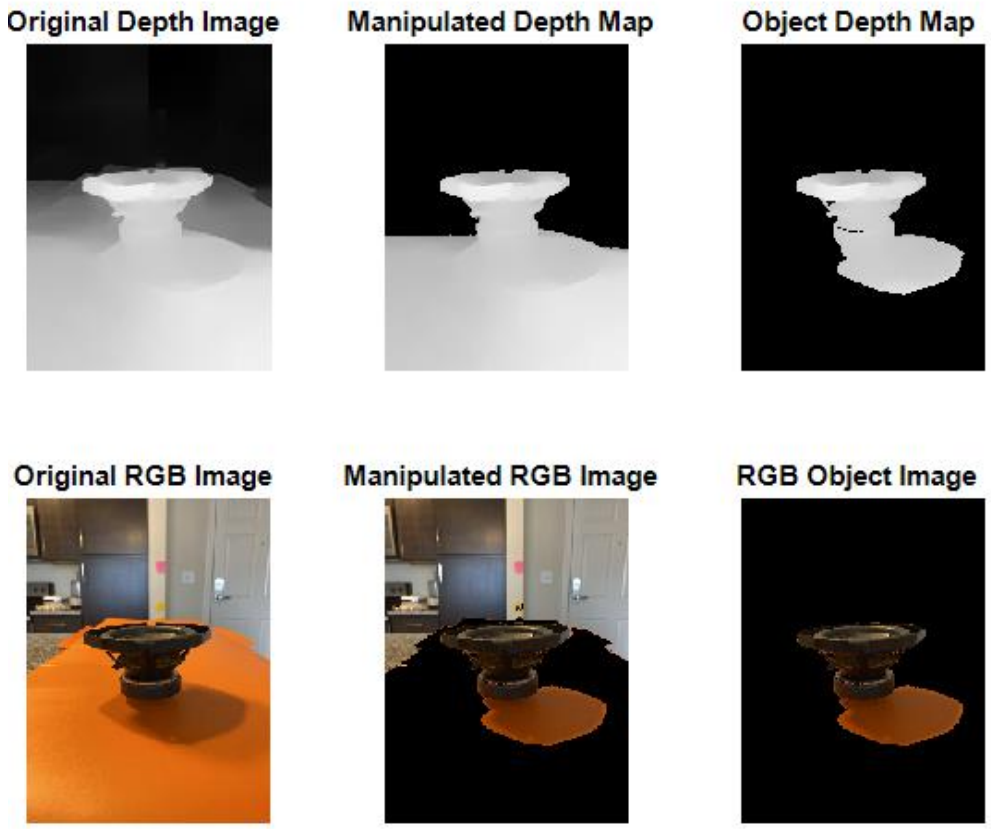

Fig. 15 Original RGB and Depth Map, Manipulated Images Based on Depth and RGB

Fig. 16 Empty Environment RGB Image (left), Extracted Object Image (center), Superimposed Object RGB Image (right)

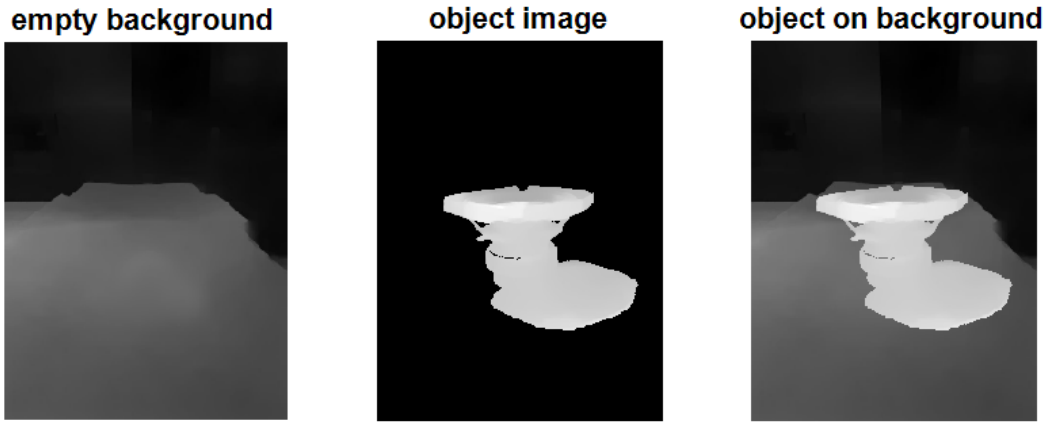

Fig. 17 Empty Environment Depth Map (left), Extracted Object Depth Map (center), Superimposed Object Depth Map (right)

Finally, the superimposed object image and depth map are used to generate a 3D stereo anaglyph image and image pair. To evaluate the quality of the superimposed image, it is compared to a generated version of the actual speaker placed on the counter (Figure 18). There is a slight difference between the images, as the superimposed speaker appears to stand out slightly more (especially in the Google Cardboard), in the area near the counter. This could be due to the fact that the superimposed object depth map does not blend in with the counter surroundings very well. That is, the shade of the object is much lighter than the empty environment counter, which would suggest that the object is much closer to the viewer than the entire counter which it sits on, which is inaccurate. Further methods could be explored to appropriately scale this superimposed object depth map to yield a more realistic result.

Fig. 18 Anaglyph Images of the Actual Speaker on the Counter (left), and Superimposed Speaker in the Environment Image (right)

5 Conclusions

5.1 Project Conclusions

In this project, we reviewed how a 3D scene can be generated from a single 2D RGB image and its depth data. By creating a horizontal shift using a disparity map, a 3D scene can be perceived when using anaglyph glasses or a virtual reality headset with separate eye displays. The disparity map calculation can be applied to vertical and diagonal shifts as well, meaning a larger panoramic image could be generated from the RGB and depth image pair.

The generation of this 3D scene is not without issue - the virtual right image typically contains a large number of hole pixels that are the result of pixels being disoccluded. Five methods were researched and implemented then compared with one another. Constant color fill proved to work the best compared to the other filtering methods, but still left distortions in the image. There are other methods that utilize the depth information to help pick more suitable fill colors, but all researched methods were restricted to using contextual data within the image.

These filled pixels may create noticeable distortions when viewed on a monitor or screen, but once the image pair is viewed with anaglyph glasses or a virtual reality headset, many of the errors are ignored by the human visual system. Our results show that blurring the virtual RGB image by a small amount to reduce visibility of holes still allows a 3D effect to be perceived. Allowing blurred images into the image pair also allows more effective storage of the images through methods like transform basis restriction.

Under the assumption that the generation of the virtual view and the hole-filling methods can be optimized for power or resource constraints, this method of image generation could simplify the hardware needed for the generation of augmented reality content. Our project explored methods for automatically cutting and placing 3D models into the scene from images captured with the same hardware. Our methods were

rudimentary, but we were still able to achieve a 3D effect for the placed object without heavy distortions. This method would allow natural objects to be used in augmented reality without the need for complex object modeling.

5.2 Future Work

There are many aspects of this project that can be further explored. First, there are numerous methods of hole-filling and hole occurring scenarios that can be investigated to determine the optimal hole-filling method. Many works have published hole-filling algorithms that are often a combination of the methods we evaluated in this project. It is also possible that different methods could be applied depending on the holes of each unique image.

For the Human Visual System analysis, it would be applicable to analyze the impact of image compression of the stereo image pairs and depth perception. This would be useful in optimizing the amount of data that needs to be used to store the images but still result in 3D images with undetectable distortions. The need to present the 3D image as a pair requires more memory than a single 2D image, however the HVS characteristics could be exploited to reduce the amount of memory needed to store each component of the image pair.

For the Augmented Reality application of object superposition, it would be necessary to explore more robust methods of edge detection to recognize the object, rather than relying on the object being on a certain color background and surface. Furthermore, it would be more ideal to have better control of the object position when it is placed in the environment. This entails being able to implement proper object positioning and scaling in the RGB image so that the object can be realistically placed at any location in the empty environment. This would give better perspective to the image, with farther placed objects appearing smaller in scale and closer objects appearing larger. Additionally, it would be necessary to better align the superimposed object’s depth map in the environment depth map based on its desired position, with proper depth map

shading to indicate its depth in the new environment. This would result in more freedom to realistically place an object at any location in the environment and generate a realistic 3D scene.

Lastly, it would be useful to further apply the methods presented in this project to generate more immersive 3D scenes. This would include 180 ° panoramic or 360° 3D immersive environments that can be viewed from a dynamic perspective, as opposed to the single static perspective of the environment that was used in the scope of this project. Another aspect that this could be applied to is in 3D video and immersive video environments. The overall goal of these AR/VR applications is to increase the amount of immersion experienced by the viewer in the virtual or augmented reality space. This project has explored many necessary steps that form a baseline for achieving this, and the above future applications can further enhance the 3D experience.

6 Appendix

6.1 Depth Map Extraction Procedure

The following depicts the steps used for the RGB + D image capture and depth map extraction.

RGB Image Background Blur Removal

1. Capture photo of object in Portrait Mode
2. Open the image from the built-in Photos app on the iPhone, and select Edit
3. In the editing view, select the *f* icon to adjust the amount of background blur.
4. To eliminate blur completely, slide the depth value entirely to the maximum value (16 on iPhone 11)
5. Alternatively, this can be done by selecting the top button labeled “Portrait” above the photo to remove all lighting and blur effects.

The following two figures show the options to select in Editing Mode on the Photos app.

Fig. 19 iPhone Screenshot of Editing Mode for Photo Captured Using Portrait Mode with Blurred Background. Select the *ƒ* icon in top left to access DEPTH slider and adjust the slider from *ƒ2.8* (default) to maximum *ƒ* value (*ƒ*16) to remove background blur.

Fig. 20 Captured Image after Background Blur Removal by Selecting PORTRAIT Toggle at the Top of the Screen

Depth Map Extraction

1. Install the software *exiftool* if this has not been done already, and save the executable file to the same folder that the captured images are stored in.
2. Save and upload the unblurred Portrait Mode photo to the computer.
3. Run the exiftool software executable file, and a window running the program will open. Keep this window open in the background.
4. Open the Command terminal, and navigate to the folder on the computer where the desired photo is stored.
5. a. Run the following command if the image was captured using the rear-facing camera:

   exiftool -MPImage2 -b [image_name].jpg > [export_name].jpg

   b. Alternatively, run the following command if the image was captured using the front-facing camera:

   exiftool -MPImage3 -b [image_name].jpg >[export_name].jpg

   This results in the depth map image saved to the current folder.

The following Figure shows the command window and example prompt to extract the depth map from the RGB image *orangutan_left.jpg*, followed by the extracted depth map in the current folder after running the exiftool command.

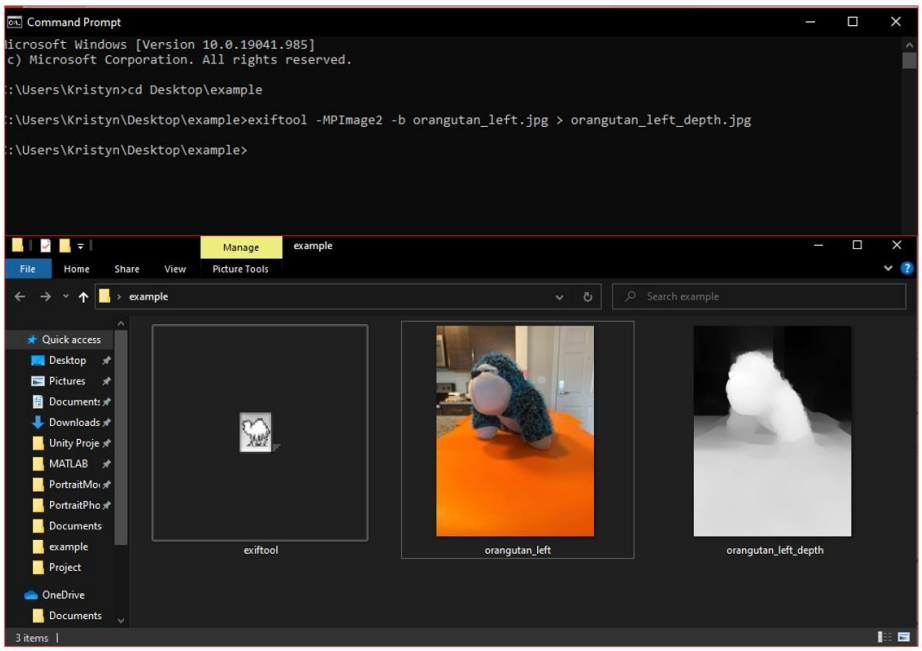

Fig. 21 Example Running Exiftool in the Command Terminal (top), and Resulting Extracted Depth Map Stored in the Current Folder (bottom)

7 References

[1] Sun, W. & Xu, L. & Au, Oscar & Chui, S.H. & Kwok, C.W. (2010). An overview of free viewpoint depth-image-based rendering (DIBR). 1023-1030.

[2] Vázquez, Carlos & Wa, James & Tam, Filippo & Speranza,. (2006). Stereoscopic imaging: Filling disoccluded areas in depth image-based rendering. Proceedings of SPIE - *The International Society for Optical Engineering.* 6392. 10.1117/12.685047.

[3] Zhang, Liang & Tam, Wa & Wang, Demin. (2004). Stereoscopic image generation based on depth images. *IEEE Conference on Image Processing*. 5. 2993 - 2996 Vol. 5. 10.1109/ICIP.2004.1421742.

[4] Criminisi, A., P. Perez, and K. Toyama. "Region Filling and Object Removal by Exemplar-Based Image Inpainting." *IEEE Transactions on Image Processing.* Vol. 13, No. 9, 2004, pp. 1200–1212.