3D and Image Stitching With the Lytro Light-Field Camera

William Lu, Wan Kim Mok, Jeremy Neiman
Advisor: Professor Zhigang Zhu, Department of Computer
Science, City College of City University of New York
(Dated: May 22, 2013)

I. INTRODUCTION

According to the American Foundation for the Blind, the number of legally blind in the US is 1.3 million and total number of blind and visually impaired [1]. Because of the increasing aging population in the US and around the world, these numbers have been rising.

The goal of this project is to see whether the Lytro Light-Field camera can provide data that is useful for helping the visually impaired. A light-field uses an array of microscopic lenses to capture a 4D light-field of the scene. This means that the camera records not only the location, intensity and color of light as a normal camera does, but also the direction of the light rays.

Since the light-field offers much more information that an standard camera, and with the Lytro, there is now one that is consumer-grade and inexpensive (currently $299), we were interested in seeing whether it could be applied to help the visually impaired, such as as the 3D sensor in a obstacle detection system.

Although right now the Kinect RGB-D sensors are more popular, less expensive($100), and real-time, they only work well indoors. Therefore for outdoor 3D sensing the Lytro, even though it is not yet a real-time sensor, might be a solution to what the Kinect sensor is not good at: outdoor 3D sensing.

The report will go over the following:

II Related Work

Other applications of light-field photography.

III Raw Data Conversion

Converting binary file to a color image that resembles the original subject.

IV 3D Data Acquisition

Extracting 3D data from light-field image by using an algorithm based on a depth from focus.

V Image Stitching

This part of the project is split into two parts: (1) creating a panorama and (2) zooming in the viewer for specific details. The purpose of image stitching is to extend the current image to a wider view of it, and to be able to see object in detail.

II. RELATED WORK

Recently open source programmers have released source code [2] for refocusing the raw light field file created by the Lytro camera. Using their source code, other program were able insert images into different focus level of the original light field photo [3]. Students at Johannes Kepler University have combined multiple Light Fields photos into a panorama[4] . These panoramas are created by creating an all-focus image and stitching them together. The panorama created by them can then be converted back into light field.

Currently Toshiba is working on developing a light field camera sensor for tablets and smartphones. This should lead to more accessible light field camera and SDK which is missing for the Lytro light field camera [5]. Nokia have invested in Pelican Imaging [6] who is also developing a camera module similar to lightfield sensors for smartphones. This camera module use an array of 4x4 cameras similar to the original lightfield camera of Stanford.

Other than cameras, lightfield technology have been applied to microscopes. Stanford have been worked on this and published a few paper [7], but currently there is no commercial application of this technology. With this lightfield microscopes we can increase the depth of view, change the perspective, create a 3d model and refocus the image.

III. RAW DATA CONVERSION

The pictures produced by the Lytro are initially in a RAW format. Behind the microlens array is a standard 3280x3280 pixel Bayer filter sensor.

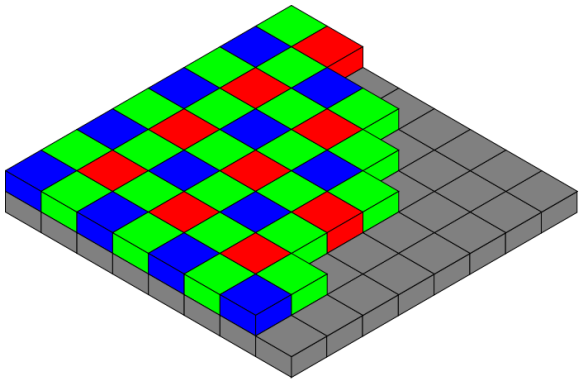


FIG. 1: A Bayer filter.

As show in FIG 1 a Bayer filter is a grid of photosensors in which each sensor can only detect a single color - red, green or blue. Each 2x2 square contains 2 green sensors, 1 red and 1 blue. Therefore there are twice as many green sensors in total.

The image is initially a binary file containing the pixels in row-major order. Each pixel is just a single 12-bit integer with no indication of color channel.

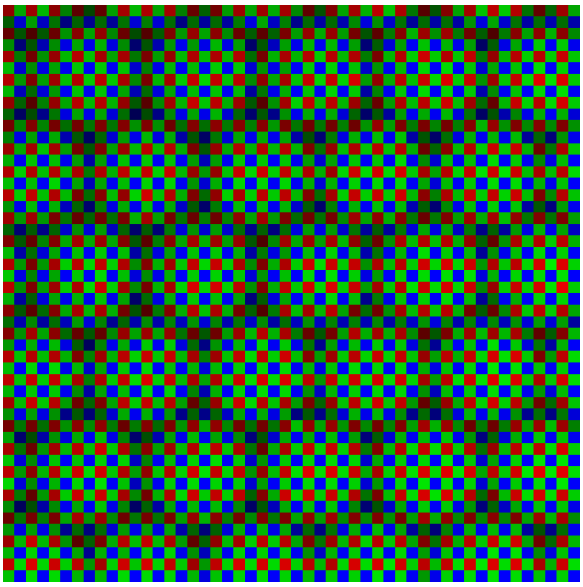
This RAW data must first be processed to produce a color image which resembles the original subject.

A. Method

First the pixels are mapped from a 12-bit single channel to 24-bit RGB representation. The data is iterated through 4 bytes at a time, representing three 12-bit pixels. Since each photosensor only senses a single color, initially each pixel will only be non-zero in one color channel. Every other pixel in even rows is blue; every other pixel in odd rows is red; and every other color is green. After doing this the image shown in FIG 2 is created.

To reconstruct the full-color image this needs to go through a process known as demosaicing [8]. The simplest algorithm for demosaicing is nearest-neighbor or linear, in which the pixels inherit the color values that it doesn't have from the nearest pixels of those colors. After doing linear demosaicing the image in FIG 2 it produces FIG 3.

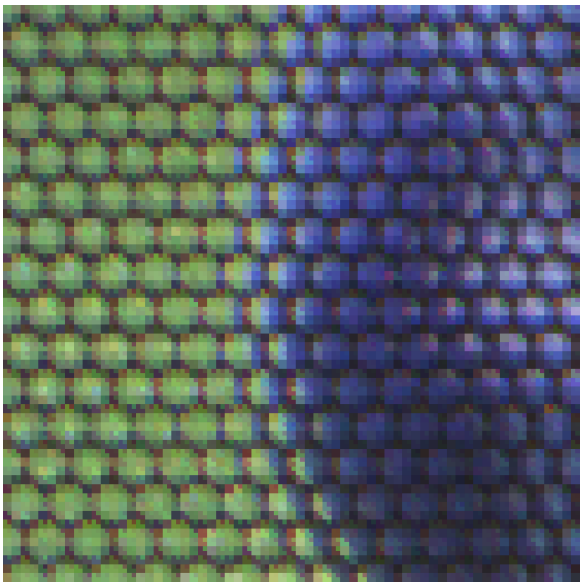
(a) The full image.



(b) A close-up crop of the same image.

FIG. 2: An image before demosaicing and a close-up of it showing the Bayer filter. The microlenses are also visible as all the circles.

(a) The full image.



(b) A close-up crop of the same image.

FIG. 3: The image after demosaicing with a close-up crop.

B. Discussion

After the RAW conversion we get a full-color image, but this is just the first step in converting from the RAW data to a refocusable image. In the close up of the demoasiced image the microlenses are visible. Using the pixel data inside each one, an image at different focuses could be reconstructed.

But for this project, demosaicing is as far as we went. Instead of reimplementing the entire process from RAW to refocused image, we used the software provided by Lytro [9]. When the Lytro software processes the RAW image it produces a stack of jpg images, each focused at a different depth. This image stack is what was used for the remaining sections.

IV. 3D DATA ACQUISITION

Each exposure from the Lytro camera provides a stack of different images focused at different depths. Therefore, it is possible to extract depth information about the object photographed using it's change in appearance across the stack. Specifically, how well focused (not-blurry) different parts of the image are when focused at different depths. By finding the depth at which each part is best focused, the object's 3D shape can be reconstructed.

A. Method

First an stack of images must be generated from the RAW file provided by the camera. The first steps in this process are described in section III, but for this section, the proprietary Lytro software was used to generate the stack from the RAW. The image stack generated as a number of different 1080x1080 pixel images and each image has an associated depth which represents the depth of the focus plane. An example image stack is shown in FIG 4. The software also produces a 20x20 depth map, as shown in FIG 5. It provides a reasonable amount of depth information, but we want to see if we can get more detail.

The pixels in each image in the stack needs to be measured for how well-focused they are. When an image is in focus, sharp lines and edges appear more clearly. At these points, there will be higher contrast when the object is more in-focus. Conversely, as the object goes out of focus, the details will blur together, meaning the contrast will be lower. By comparing the contrast at each pixel between layers, the most-focused layer can be determined.

FIG. 4: An example of the different images contained in an image stack. Each image is focused at a slightly different depth.

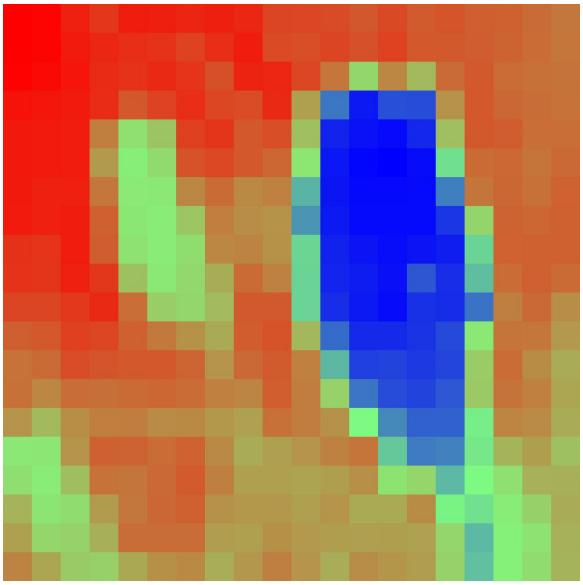
To determine a contrast value for each pixel, a 2D discrete Laplace filter is used. The filter kernel is defined as:

$$L = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Applied to each pixel in a layer, this will sum the four adjacent pixels and subtract four times the center, giving some value as to how different the pixel is from it's neighbors. After applying the Laplace filter, the values are replaced with their absolute values to give absolute contrast, whether it be lighter or darker.

After this is done to every image layer in the stack, a depth map can be created by taking the depth value for each pixel from the layer in which it is most focused. The resulting depth map is shown in FIG 6. To reduce the noise a Gaussian can be applied, as shown in FIG 7. Clearly our experiments show that it is possible to generate a high resolution depth map with reasonable accuracy from the Lytro image stack.

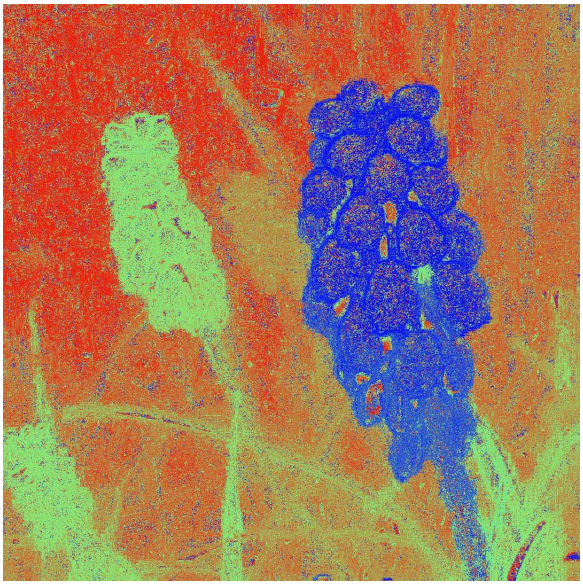
(a) One of the images in the image stack.



(b) The 20x20 depth map.

FIG. 5: The image stack produced by the Lytro software includes a 20x20 depth map.

(a) One of the images in the image stack.

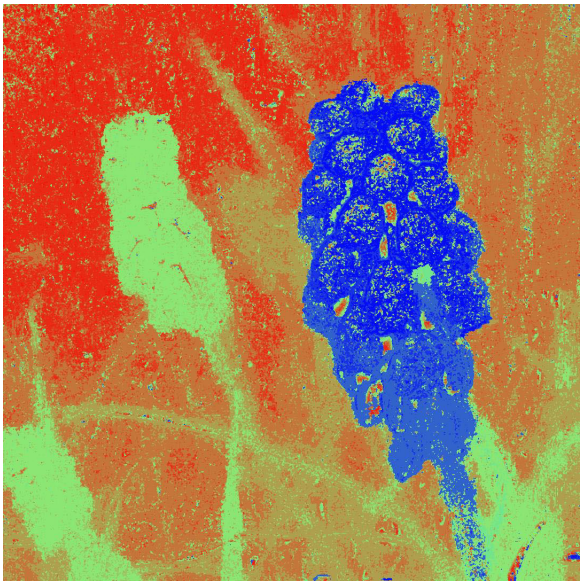


(b) The resulting depth map.

FIG. 6: The resulting depth map. Blue areas are closest, red furthest and green between the two.

For more comparison, the depth maps can be visualized as a 3D model. Using OpenGL, a mesh is created using the points X,Y and depth values from the depth map, and then one

(a) One of the images in the image stack.



(b) The depth map after Gaussian filtering.

FIG. 7: The resulting depth map after Gaussian filtering. Blue areas are closest, red furthest and green between the two.

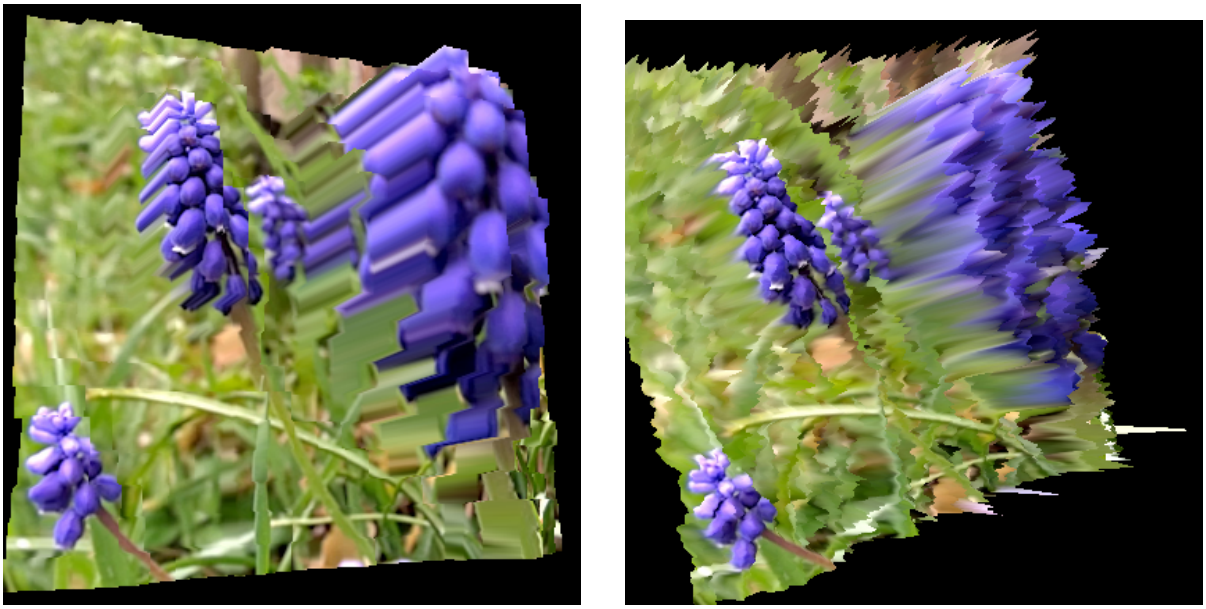
of the images from the stack is texture mapped onto it. The 3D surface shown in FIG 8a is the created using the 20x20 depth map supplied by the Lytro software and the 3D surface in FIG 8b is created using the Laplace and Gaussian filter. When comparing the two surfaces, the 20x20 surface is smooth and doesn't have noise but the resolution is too low to accurately represent the object. The surface created using the Laplace depth map has well defined edges around the objects.

B. Conclusion and Discussion

The final depth map can be fairly detailed and accurate. The Lytro software produces a 20x20 depth map as show in FIG 5 and our depth maps compare favorably to that, showing considerably more detail. But there are several problems that were encountered.

The first is inherent in any 3D mapping system based on passive color cameras, that of low-contrast areas. A solid colored wall will look the same whether it is in focus or not. Therefore large areas in the resulting depth map may be completely wrong, or be completely noise.

The second issue that arose is due to the reliance on Lytro's proprietary software for



(a) The 3D surface using the 20x20 depth map. (b) The 3D surface using laplace with a gaussian filter with sigma=2.

FIG. 8: The 3D surfaces

generating the image stack. The software optimizes the image stacks for the web, meaning that it tries to make them as small as possible. To this end, it determines how many different depth planes are necessary to display the image under a full range of focuses. Sometimes the image stack has only one single layer, making it impossible to get any 3D detail using this method. The software is not open or customizable so the only way the ensure a full range of depths is to reimplement the Lytro software.

The third issue is that, even after the Gaussian filter there is still quite a lot of noise in the depth map. Using the 3D viewer it became easy to visualize how noisy the depth map was because of all the spikes.

Additional examples are shown in the appendix which illustrate these issues.

V. IMAGE STITCHING TO CREATE 3D PANORAMAS

It's also possible to create a 3D panorama, so to speak, where separate images are stitched together not only to create a wider image, but one with more depth, by mapping small sections of an image to zoomed-in images of those sections so that users, particularly low

vision people, can better see signs and objects. The algorithm for automatically aligning the images includes: (1) matching the common feature points; (2) aligning and warping the images with a homography matrix derived from the matching points; (3) stitching aligned images to create a large panorama; and (4) viewing aligned images through a viewer that allows zooming and transitioning.

A. Method

1. Matching

The algorithm is given a number of image stack files. The first image is treated as the base and all other images are aligned relative to it.

First, the light-field image stacks are flattened into an all-focused image. This is done by taking the pixel data from the most-focused layer at each point. This all-focused image will make it easier to detect similarities across images because there won't be as much discrepancy due to blurriness. These all-focused images are converted to grayscale.

Then the SURF algorithm [10] is applied to each one. The SURF descriptor describes how pixel intensities are distributed within a scale dependent neighborhood of each interest point detected by Fast Hessian. Then we match the points of the SURF descriptor of the images.

2. Warping/Aligning

When matching points between the images are found, RANSAC algorithm is used to estimate a Homography matrix using the matched points. The RANSAC algorithm is a method of estimating model parameters that is robust against outliers. The outliers are data that do not fit the model. The basic RANSAC algorithm is summarized as follows [11]:

1. Select randomly $\boldsymbol{N}$ number of points
2. Solve for the parameters $\vec{x}$ of the model.
3. Determine how many points from the set of all points fit with a predefined tolerance ε

4. If the fraction of the number of inliers over the total number points in the set exceeds a predefined threshold τ, re-estimate the model parameters using all the identified inliers and terminate

5. Otherwise, repeat steps 1 through 4 (maximum of $\boldsymbol{L}$ times).

This will produce a transformation matrix that will translate and warp the images so that they are placed correctly with relation to the base image. FIG 9 shows an image with the descriptors and matched inliers.

FIG. 9: The left image is aligned with it's location in the right image. The red dots show descriptors extracted with SURF and the green lines show the matched inliers.

After aligning all the images a JSON file is creating with the correspondence including the coordinates of the corners where the image aligns and the matching vertices.

3. Stitching

The image stitching program can automatically take a number of light-field image stacks and accurately align them together, even under some amount of distortion and when parts of the image are cut off. FIG 10 shows three input images to the image stitching program and FIG 11 shows the resulting panorama.

FIG. 10: Three input images to the stitching program.

FIG. 11: The output panorama image showing the overlap.

4. Viewer

A viewer was created that allows users to zoom into specific parts of the picture and view that section in more detail, in addition to refocusing the image using the stack file. The viewer is an extension to code written by Behnam. [12]

It uses the json file to first open the parent lfp file. After the parent image is loaded, whenever the user right clicks on the image the most focused layer for the point clicked on is selected and display. When the user scrolls up with their mouse scroll wheel, if there is a child image associated with the region the cursor is located over, the viewer zooms into that section and then opens the child lfp file. The child image is that specific part taken up close which provide the user a clearer and more detailed view of that part.

The zoom in transition between the parent and the child is created by slowly cropping

from the entire picture until we reach the corner of the sub-image. Next an affine transformation is performed on the original image, using the four corners of the sub-image saved in the json file to produce the transformation matrix. The corners used are the corners from the homography transformation done during the image matching. An affine transform is used here to produce faster results since the homography. The affine transformation is based on the two following equations $x' = ax - by + c$ and $y' = bx + ay + d$.

When the user scrolls down on their mouse scroll wheel it returns to that images parent where they can select another location to see in more detail.

B. Discussion

A problem with the viewer is when the zoomed up image misaligned with the child image during it transition. This alignment problem fixed to a certain extent with the affine transformation but into some cases the misalignment can still very apparent. When the affine transformation is applied to a small section of the larger picture, being off by one pixel will create a large distortion in the final image. This in most cases will shift the image by a few pixel similar to the lion king figure FIG 12, but when there are larger mismatch the result is like the tv display FIG 13.Also since we aren't using the homography transformation from the image matching there is additional misalignment.

FIG. 12: Left show the transition without any affine transformation. Right shows the same transition with affine transformation

One thing we didn't get to try is combining the depth information obtained in the previous section to improve the results of the automatic alignment. With this we just use the 2D

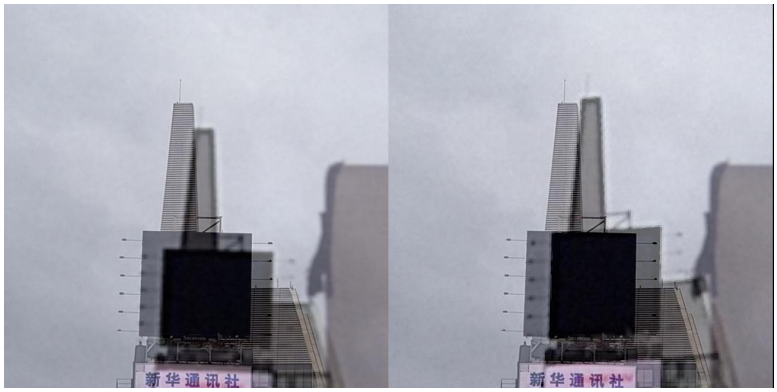


FIG. 13: This is an example of the misalignment that remains after affine transformation. Left show the transition without any affine transformation. Right shows the same transition with affine transformation

color to align the images, but with the addition of object shape, it might give better results for alignment.

VI. CONCLUSION

We've show how it's possible to apply the unique information provided by a light-field camera - both extracting 3D information and creating an interactive panorama. But it's not immediately clear how this might be applied usefully to assist the visually impaired.

Using a light-field camera as a 3D sensor for obstacle detection is a possibility, but the work done here would not be applicable. We did not have access to the Lytro's data in real time, but only after downloading the images off of it and onto a computer. There are expensive high-end light-field cameras that provide that functionality [13], but the Lytro does not offer it and as of now there are no other light-field camera that do which would be practical for a wearable obstacle detection system.

Furthermore, the algorithm is much too slow - on a scale of seconds on a desktop PC - to be used for a real-time application. The benefit of using a light-field camera - that it can potentially provide much more information than some other 3D sensors comes at the cost of increased processing time. But this problem may be solvable, especially by extracting the depth directly from the RAW image.

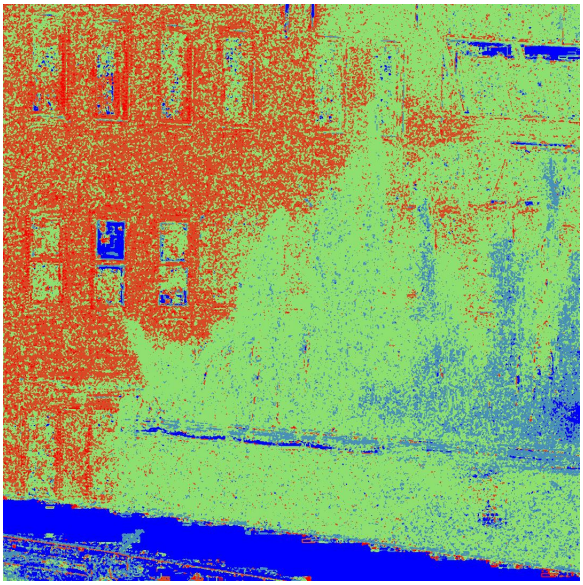
A. Work Distribution

Jeremy Neiman did the RAW data conversion; the depth-from-focus algorithm to generate the depth maps; and the automatic image alignment algorithm. William Lu edited the 3d surface and the lfp picture viewer. Wan Kim Mok did the image stitching.

VII. APPENDIX

The appendix contains four additional examples of the depth-from-focusing algorithm.

(a) One of the images in the image stack.



(b) The depth map after Gaussian filtering.

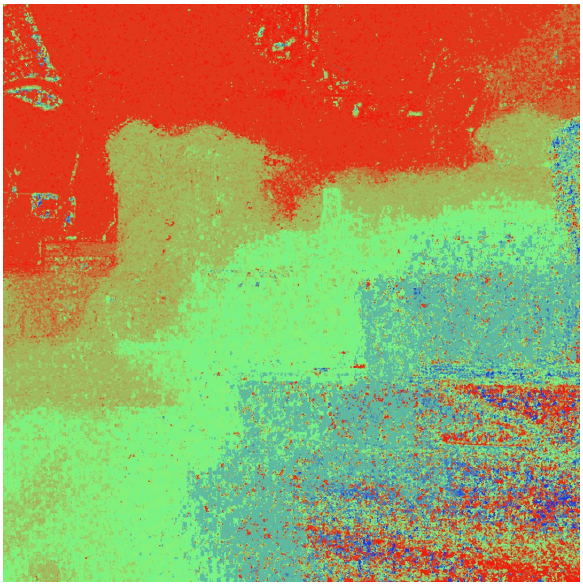
FIG. 14: An example when there aren't as well defined objects, but the different depth ranges are captured with the railing in the front, the branches in the middle, and the building in the rear.

(a) One of the images in the image stack.

(b) The depth map after Gaussian filtering.

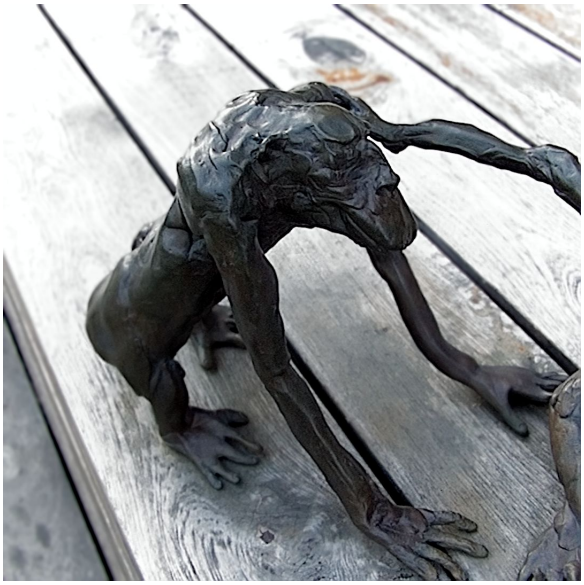
FIG. 15: An example where the Lytro software generated only one image in the stack, thus making it impossible to discern any depth information.

(a) One of the images in the image stack.

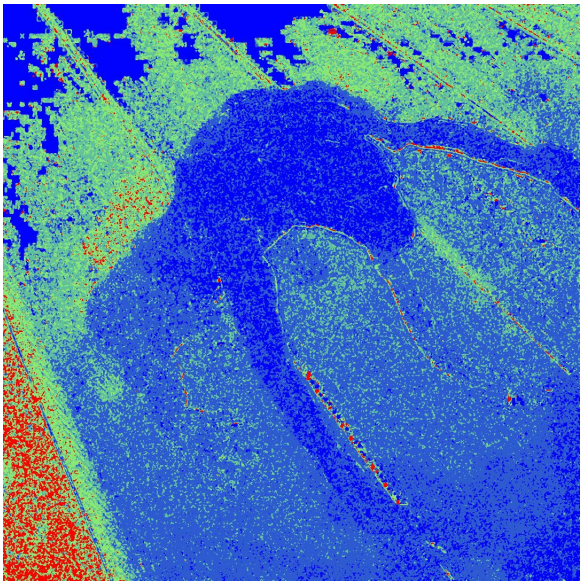


(b) The depth map after Gaussian filtering.

FIG. 16: An example where none of the images in the stack are well focused on the lower right hand corner, leading to a lot of noise in the area.



(a) One of the images in the image stack.



(b) The depth map after Gaussian filtering.

FIG. 17: The upper left hand corner of the image is overexposed leading to low contrast which is why the depth appears to be closer than it is.

VIII. ACKNOWLEDGMENTS

The capstone project has been supported by NSF GARDE Program for Course Development on Assistive Technology to Aid Visually Impaired People (Award #1160046), NCIIA Course and Program Grant on Human and Machine Intelligence - Perception, Computation and Action (#10087-12). We would also like to thank Prof. Amy Nau at UPMC Eye Center, Prof. Tony Ro at CUNY City College, and Ms Babara Campbell and Mr. Dan Diaz, both at New York State Commission for the Blind and Visually Handicapped (CBVH), for their valuable comments and suggestions on our designs and projects.

---

[1] "Blindness and Low Vision Fact Sheet", https://nfb.org/fact-sheet-blindness-and-low-vision

[2] "Refocus: Linux Tool to Process Raw Lytro Images"
http://lightfield-forum.com/2013/03/refocus-linux-tool-to-process-raw-lytro-images/

[3] "Lytro Image Manipulation: Linux Tools to insert Images into LightField Pictures"
http://lightfield-forum.com/2013/03/lytro-image-manipulation-linux-tools-to-insert-images-into-lightfield-pictures/

[4] "Panorama Light-Field Imaging"
http://www.jku.at/cg/content/e60566/e155404#e155481

[5] "Depth imaging: Toshiba develops light-field sensor for phones; Panasonic 3D chip"
http://pmanewsline.com/2013/03/03/depth-imaging-toshiba-develops-light-field-sensor-for-phones-panasonic-3d-chip/

[6] "Pelican Imaging promises freedom from focusing"
http://www.extremetech.com/computing/152761-pelican-imaging-promises-freedom-from-focusing

[7] Levoy et al., "Light Field Microscopy"
http://graphics.stanford.edu/papers/lfmicroscope/

[8] Remi Jean, "Demosaicing with The Bayer Pattern"
http://www.unc.edu/ rjean/demosaicing/demosaicing.pdf

[9] https://www.lytro.com/downloads

[10] Smith, Advanced Imaging Processing "Speeded-Up Robust Features" lecture

http://www.sci.utah.edu/ fletcher/CS7960/slides/Scott.pdf

[11] Derpanis, "Overview of the RANSAC Algorithm"
http://www.cse.yorku.ca/ kosta/CompVis_Notes/ransac.pdf

[12] Behnam, "LFP (Light Field Photography) File Reader"
http://code.behnam.es/python-lfp-reader/

[13] http://www.raytrix.de/index.php/Cameras.html