One-Shot to *Vertigo*: Novel View Synthesis using Light Field Cameras

Rohan Rao
Robotics Institute
Carnegie Mellon University
rgrao@andrew.cmu.edu

Abstract

The vertigo shot technique is used for cinematic effect and for compelling aerial shots, and is designed to keep the subject in focus while moving and zooming the camera, making the background field of view change dramatically. This report presents an approach to model this technique using a single shot captured from a light field camera. Existing single-shot image pipelines require depth information that is often estimated using neural networks or dual-pixel camera sensors, whereas here we use correspondence and defocus data from the light field to obtain the depth at every point in the scene. We also provide a method to discretize real-world depth maps and merge multiple image segments to create a shallow depth of field effect. The proposed method extends the feasibility of using light field cameras for cinematography by providing novel view synthesis techniques using individual light field frames.

1. Introduction

A Vertigo shot or the dolly zoom effect is a special effect often used in cinematography that is designed to create a sensation of vertigo, or a feeling of unreality. It was popularized by Alfred Hitchcock through usage for climatic revelation in his 1958 movie, *Vertigo*. The *Computational Zoom* paper from NVIDIA Research [1] describes a method of post-capture image synthesis that allows for manipulation of the image composition from a stack of photographs captured at different distances in the scene. However, their implementation assumes that we have a number of images of the same scene at various positions and zoom settings.

An even more recent paper [5] describes how we can create a very similar effect by using a single image captured from a smartphone camera. They describe how novel views can be generated by taking advantage of the geometry of the scene, and also derive closed-form expressions for various intermediate components of the pipeline. Algorithms are provided for various steps, and are elaborated in this report as well. Their implementation uses depth images obtained either from a Unity 3D simulator or obtained through modern smartphone camera dual-pixel depth estimation frameworks like [3]. It does not go into details of the smartphone-based implementation, and provides only final results. This report describes some of the difficulties faced in implementation, and also how the continuous-valued depth maps can be discretized for usage by the algorithms that they have provided.

This report aims to implement and then also improve upon their work by using light field cameras. Light field cameras are designed by adding a 2-dimensional array of tiny lenses in front of the main sensor of a camera. This simple enhancement allows for interactive refocusing and capture of multiple viewpoints from a single shot. These two aspects allow us to obtain the depth of a scene directly through two approaches - depth from correspondence and depth from defocus. This lenslet array allows photographers to capture the ”light field”, which is a 4-dimensional description of all the rays that form the image.

The light field camera was commercialized in 2012 by Lytro Inc. and its founder, Ren Ng. Lytro Inc. had also filed a patent [8] in 2014 (now owned by Google) that describes how a dolly zoom effect can be generated using light field cameras. It is thus possible to generate this effect without any requirement to physically move the camera.

2. Single Camera View Synthesis Pipeline

The single camera single-shot view synthesis pipeline is shown in Figure 1 and every step is described in detail below.

2.1. Input

From [5]: The input is the image $\mathbf{I}$ with FoV θ, its depth map $\mathbf{D}$ and the known intrinsic matrix $\mathbf{K}$.

The RAW lightfield images are obtained directly from a Lytro Illum camera (testing performed with the INRIA dataset [7]). With this RAW data, the camera exports a compressed calibration data archive that contains white images at various focal length settings which can be used to

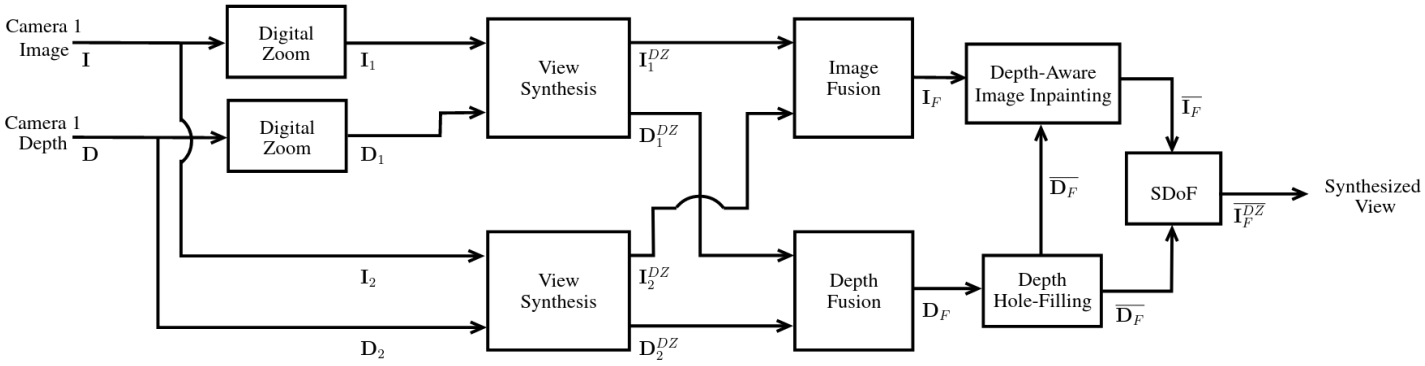

Figure 1. Single camera single shot synthesis pipeline from [5]

Figure 2. Original image I (I_2)

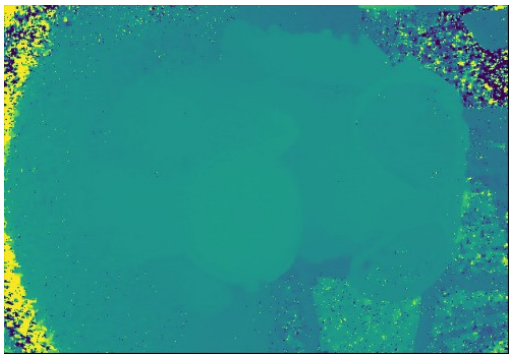

Figure 3. Original depth map D (D_2)

calibrate the camera. I used PlenPy [9] to perform the various preprocessing steps to obtain the image. These include loading of the calibration data, demosaicking the hexagonal grid based RAW images from the light field camera, aligning the sensor image, slicing images to the light field, and resampling the light field to a rectangular grid. The calibration is a one-time step, allowing me to then load any images in the same directory/capture stack with these settings.

The software calculates depth disparity using the structure tensor approach [13], and then fuses these disparities using the Total Variation (TV) - L1 norm regularization and optimized using a primal dual algorithm [15]. This provided reasonably good relative depth estimates for some of the sample images, so I chose to go with this. However, the paper I am implementing [5] assumes the availability of the exact depths (assuming calibration), whereas the software I used does not provide this information. I tried to go ahead with the relative depths and focal lengths for all calculations and got reasonable results.

The original input image $\mathbf{I}$ (Fig. 2), depth map $\mathbf{D}$ (Fig. 3) and intrinsic matrix $\mathbf{K}$ are re-used as $\mathbf{I}_2$, $\mathbf{D}_2$ and $\mathbf{K}_2$ respectively.

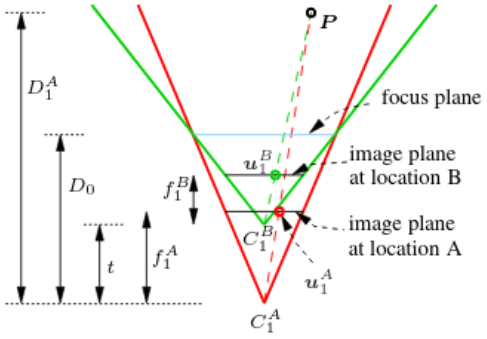

Figure 4. Single camera system geometry from [5]

2.2. Digital Zoom

We apply digital zoom to both $\mathbf{I}$ and $\mathbf{D}$ according to a zoom factor that is obtained by using the ratio of the two desired focal lengths.

From [5], this is described by considering the zoomed-in Field of View (FoV) as a partial FoV of the actual view at the initial position C_1^A in Figure 4. If the actual intrinsic matrix is $\mathbf{K_1}$, the intrinsic matrix $\mathbf{K_1^A}$ for partial FoV can

Figure 5. Digitally zoomed image I_1

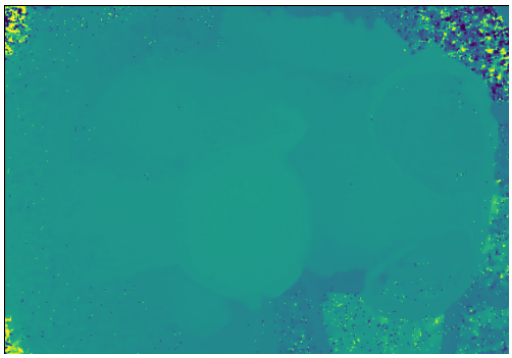

Figure 6. Digitally zoomed depth map D_1

be obtained as $\mathbf{K_1^A} = \mathbf{K_1}\text{diag}\{k_0, k_0, 1\}$. Here the value $k_0 = f_1^A/f_1 = tan(\theta_1/2)/tan(\theta_1^A/2)$. The paper then uses this to obtain a closed-form expression for the zoom-pixel coordinates in terms of the actual image pixel location (setting the rotation matrix as an identity matrix).

However, I used this digital zoom factor directly, instead of using the provided equation. I crop the part of the image that will remain in the result, which form a centred bounding box of the final desired size. I then use OpenCV's resize function after considering padding, and then downscale the image. [6]. This results in a re-scaled image of the same dimensions.

This procedure is performed on the original image and depth map to obtain the zoomed-in image $\mathbf{I_1}$ (Fig. 5) and the corresponding depth map $\mathbf{D_1}$ (Fig. 6).

2.3. View Synthesis based on Camera Geometry

For a general pair of pin-hole cameras A and B with their centers located at C_A and C_B, the projection of any point $\mathbf{P} \in^3$ on the camera image planes are $(u_A^T, 1)^T = \frac{1}{D_A}\mathbf{K_A}[\mathbf{J}_3\mathbf{0}_3](\mathbf{P}^T, 1)^T$ and $(u_B^T, 1)^T = \frac{1}{D_B}\mathbf{K_B}[\mathbf{R}, \mathbf{T}](\mathbf{P}^T, 1)^T$. Here, the 2×1 vector $\mathbf{u}_X$, the 3×3 matrix $\mathbf{K}_X$, and the scalar D_X are the pixel coordinates on the image plane, the intrinsic parameters, and the depths of $\mathbf{P}$ for camera X (A/B) respectively. The 3×3 matrix $\mathbf{R}$ and the 3×1 vector $\mathbf{T}$ are the relative rotation and translation of camera B with respect to camera A. In general, the relationship between $\mathbf{u}_B$ and $\mathbf{u}_A$ can be written in closed-form as (from [5])

$$\begin{pmatrix}\mathbf{u}_B \\ 1\end{pmatrix} = \frac{D_A}{D_B}\mathbf{K}_B\mathbf{R}(\mathbf{K}_A)^{-1}\begin{pmatrix}\mathbf{u}_A \\ 1\end{pmatrix} + \frac{\mathbf{K}_B\mathbf{T}}{D_B} \quad (1)$$

Here $\mathbf{T}$ can also be written as $\mathbf{T} = \mathbf{R}(\mathbf{C}_A - \mathbf{C}_B)$.

In the specific case as shown in Figure 4, we can assume that the camera starts at an initial position $\mathbf{C}_1^A$ with the given focal length f_1^A. To achieve the dolly zoom effect, [5] assumes that the camera translates by a distance t to position $\mathbf{C}_1^B$ and the focal length also changes accordingly to f_1^B. The FoV changes appropriately since $f = (W/2)tan(\theta/2)$ where W is the image width, and f is in pixel units.

The paper assumes that the camera moves along the principal axis Z, meaning the motion can be written as $\mathbf{C}_A - \mathbf{C}_B = (0, 0, -t)^T$. The depth of $\mathbf{P}$ changes as $D_1^B = D_1^A - t$. As there is no relative rotation here, the matrix $\mathbf{R}$ is an identity matrix. The paper also simplifies the intrinsic matrix by assuming no shear or distortion, making the intrinsics simple to model as shown below in equation 2.

$$\mathbf{K}_1^A = \begin{bmatrix} f_1^A & 0 & u_0 \\ 0 & f_1^A & v_0 \\ 0 & 0 & 1 \end{bmatrix} \quad (2)$$

Here $\mathbf{u}_0 = (u_0, v_0)^T$ is the principal point of the camera in pixel dimensions. Assuming a constant image resolution, the intrinsic matrix at the new location is simply a zoomed version (related by a zoom factor) of the previous one, and can be obtained as shown before, $\mathbf{K}_1^B = \mathbf{K}_1^A\text{diag}\{k, k, 1\}$. Here $k = f_1^B/f_1^A = (D_0 - t)/D_0$. This approach also assumes that the depth of the point $\mathbf{P}$ is more than the translation t, or else it will be excluded on the image plane at the new location.

From the general pinhole equation 1, we can obtain a closed-form expression for $\mathbf{u}_1^B$ in terms of $\mathbf{u}_1^A$ as follows

$$\mathbf{u}_1^B = \frac{D_1^A(D_0 - t)}{D_0(D_1^A - t)}\mathbf{u}_1^A + \frac{t(D_1^A - D_0)}{D_0(D_1^A - t)}\mathbf{u}_0. \quad (3)$$

This equation, as described above, can be used to warp the image $\mathbf{I}_1$ and the depth map $\mathbf{D}_1$ to obtain the synthesized image $\mathbf{I}_1^{DZ}$ and synthesized depth map $\mathbf{D}_1^{DZ}$. This is implemented using z-buffering and forward warping, as shown in [10]. The forward warping algorithm modified to use Z-buffering is shown in Algorithm 1.

After performing the warping step based on this Z-buffer, the resulting images and depth maps from the two parallel pipelines are shown in the following figures I_1^{DZ} (7), I_2^{DZ} (9), D_1^{DZ} (8) and D_2^{DZ} (10).

Algorithm 1: Forward Warping with Z Buffering

Input : f, h
Output: g
Create a depth-buffer Z_{buf} and initialize the values to INT_{MAX}.
for *every pixel x in f(x)* **do**

1. Compute the destination location $x' = h(x)$.
2. **if** *depth at destination pixel* $< Z_{buf}$ *value at destination pixel* **then**
   Copy the pixel $f(x)$ to $g(x')$.
   end

end

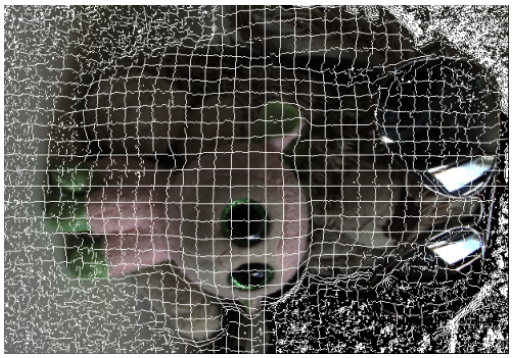

Figure 7. Image I_1 warped using Eqn. 3 (I_1^{DZ})

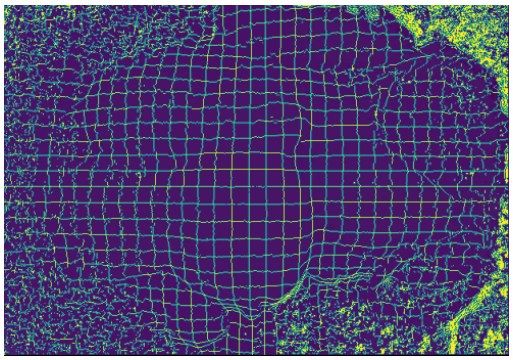

Figure 8. Depth map D_1 warped using Eqn. 3 (D_1^{DZ})

2.4. Image Fusion

The synthesized image from the second camera (new location) $\mathbf{I}_2^{DZ}$ is used to fill in missing/occlusion areas in the synthesized image $\mathbf{I}_1^{DZ}$ from the first camera. This is achieved with the following steps:

1. Identify the missing areas in the synthesized view $\mathbf{I}_1^{DZ}$. In the paper, this is implemented by using a binary mask $\mathbf{B}$ by checking the validity of $\mathbf{I}_1^{DZ}$ at each pixel location (x, y). I implemented this by checking the locations where the image is not updated as part of the Algorithm 1, since the output was initially set to some MASK value.
2. With the binary mask $\mathbf{B}$, the synthesized images $\mathbf{I}_1^{DZ}$ and $\mathbf{I}_2^{DZ}$ are fused to generate $\mathbf{I}_F$:

$$\mathbf{I}_F = \mathbf{B} \cdot \mathbf{I}_2^{DZ} + (1 - \mathbf{B}) \cdot \mathbf{I}_1^{DZ} \tag{4}$$

where $\cdot$ is the element-wise matrix product. The depths for synthesized view, $\mathbf{D}_1^{DZ}$ and $\mathbf{D}_2^{DZ}$ are also fused in a similar manner to obtain $\mathbf{D}_F$. These are shown in Figure 11 (I_F) and Figure 12 (D_F). It can be seen that many of the artifacts/incomplete regions are now filled, as required. However, there are still regions where the depth map is incomplete and needs to be filled. This is done in the subsequent sections.

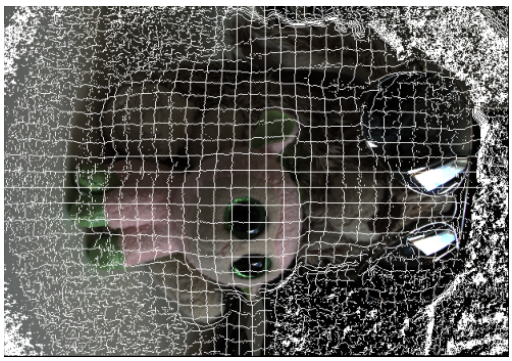

Figure 9. Image I_2 warped using Eqn. 3 (I_2^{DZ})

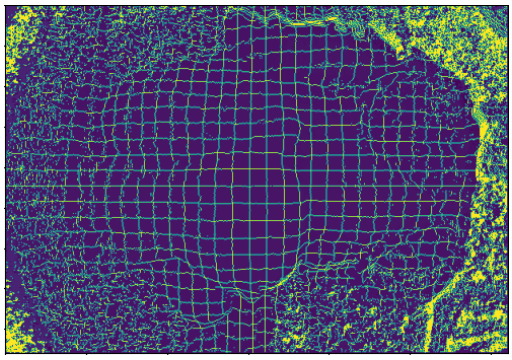

Figure 10. Depth map D_2 warped using Eqn. 3 (D_2^{DZ})

2.5. Occlusion Area Handling

Occlusion areas are generated due to the nature of camera movement and depth discontinuities along the epipolar lines. A binary mask is generated to denote the occlusion areas by checking validity of the resulting fused images at every pixel. The depth map hole filling must be done first, since we need to use this for depth-aware image inpainting. The approach used for depth-hole filling is relatively simple, and just fills the missing depth based on the maximum nearest-neighbor value, as shown in Algorithm 2.

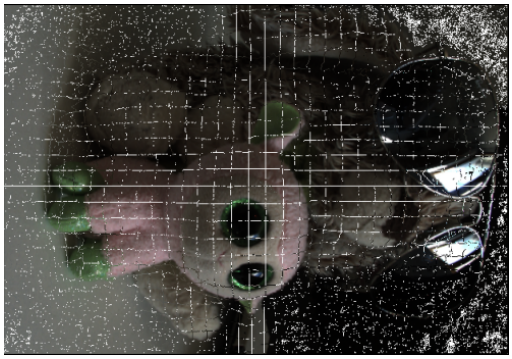

Figure 11. Fused image obtained using Eqn. 4 (I_F)

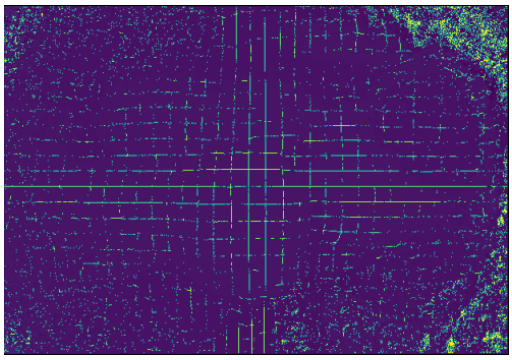

Figure 12. Fused depth map obtained using Eqn. 4 (D_F)

Algorithm 2: Depth Map Hole Filling

Input : Fused depth $\mathbf{D_F}$, dimensions (width W and height H)
Output: The hole filled depth $\overline{\mathbf{D_F}}$
Initialize $\overline{\mathbf{D_F}} = \mathbf{D_F}$.
for *x = 1* **to** *H* **do**
  for *y = 1* **to** *W* **do**
    if $M(x, y) = 1$ **then**
      2.1) Find four nearest neighbors (left, right, bottom, top).
      2.2) Find the neighbor with the maximum value (d_{max}), since we intend to fill in the missing values with background values.
      2.3) Set $\overline{D_F}(x, y) = d_{max}$.
    end
  end
end

The result of this hole filling step for the depth map is shown in Figure 13.

Before moving to the image hole filling algorithm, I realized that it requires a discrete/unique set of depth values from the depth map. Since the original paper [5] used a simulator (Unity 3D) for a majority of the implementation, they have very clean and sharply-varying depth maps with a finite set of depths. However, given the approach used to obtain light field depth maps uses Gaussian filtering and other continuous-valued functions, the number of unique depth values in the image is of the order of the number of pixels. Thus, I had to discretize the depth map before proceeding to the image inpainting algorithm.

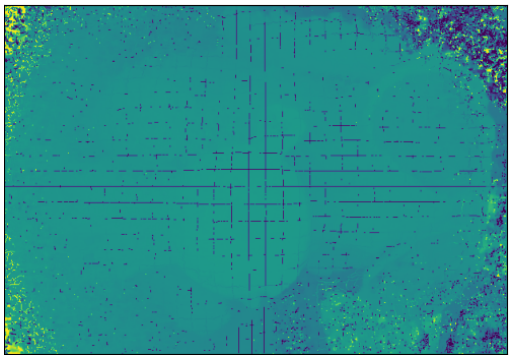

Figure 13. Hole-filled depth map obtained using Algorithm 2 ($\overline{D_F}$)

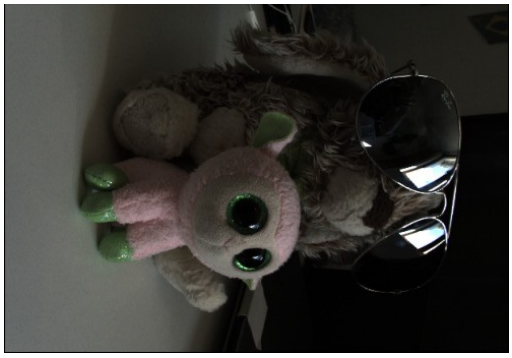

Figure 14. Sample image

Consider the image shown in Figure 14 and the corresponding depth map (before discretization) shown in Figure 15. I computed the unique values in this depth map and plotted a histogram of these values, shown in Figure 16. Note that the implementation of my light field depth reading code normalizes all relative depths between 0 and 1.

It can be clearly interpreted from the histogram that the depths are clustered around the subject in the scene, in this case the pair of toys. In most dolly zoom or vertigo scenarios, this would be true, since we have a subject based on which we are applying the effect. This sort of distribution could be fitted using a Gaussian, and I would need some sort of non-uniform (possibly stratified) sampling approach to get a larger number of values within the peak band of the Gaussian. Another approach that should provide good results is using K-Means clustering to partition the data and obtain a set of clusters. The property of minimizing the intra-cluster variance means that it would generate a larger

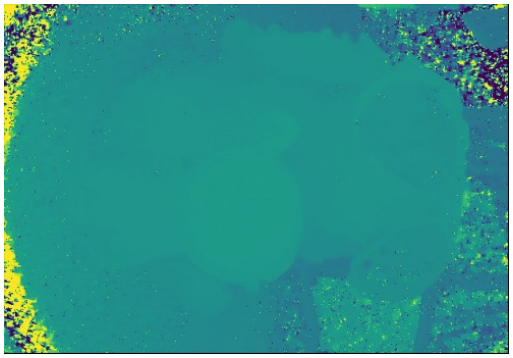

Figure 15. Depth map for image 14

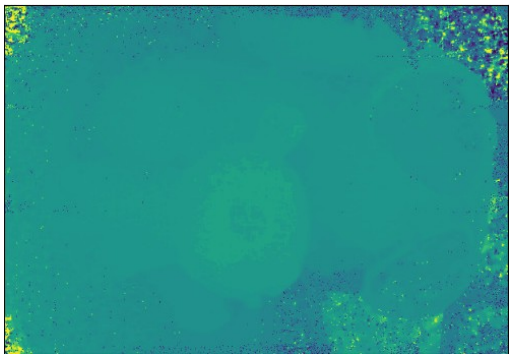

Figure 17. Discretized depth map for 15 using Algorithm 3

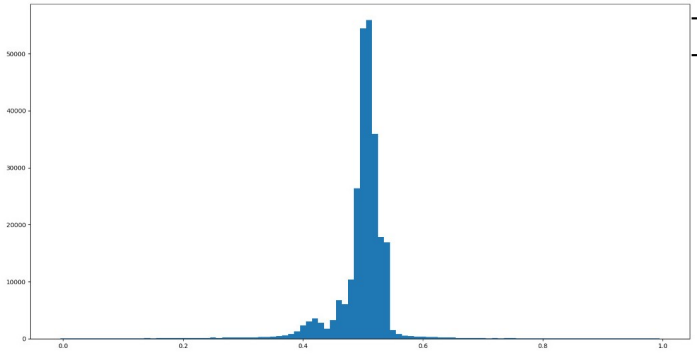

Figure 16. Depth map histogram for 15

number of clusters within the Gaussian peak, which is what was desired. It would also provide a way to extend to other scenes where there may be more than one subject, which would otherwise require a Gaussian mixture model for fitting.

This K-Means clustering based approach is described in Algorithm 3, and the results are shown in Figure 17. The number of clusters N was set to 20, and it can be seen that some of the artifacts near the left of the image are reduced, and there is more low-resolution detail on the face of the toy in front. The new depth map has only 20 values and can now be used in the subsequent image inpainting algorithm.

The image hole filling algorithm is described in Algorithm 4. This strategy is implemented in a back-to-front order, since the holes should be filled from parts of the image at the same depth or next closest depth. This is very similar to [5], but includes the depth discretization step that was explained above. Finding the nearest valid pixels was implemented by searching for any non-zero pixels that are nearest to the pixel under consideration in the same row [11]. The paper mentions that this approach is quite simple and can be extended to more search directions. The result of this approach is shown in Figure 18.

Algorithm 3: Depth Map Discretization

Input : Depth Map $\overline{\mathbf{D_F}}$ with dimensions (width W and height H), N = number of clusters
Output: The discretized depth $\overline{\mathbf{D_{F,disc}}}$
Initialize $\overline{\mathbf{D_{F,disc}}} = \overline{\mathbf{D_F}}$.
1. $\mathbf{u}$ = unique values in $\overline{\mathbf{D_F}}$.
2. Fit $\mathbf{u}$ using K-Means with $n_{clusters} = N$.
3. Update the depth values in back-to-front order (elaborated below).
4. Sort the cluster centers in descending order.
5. Append 0 and 1 on either end of the sorted array to form an intervals array $\mathbf{arr}$.
for *i = 1* **to** *length(intervals)-1* **do**
  5.1) Identify locations in $\overline{\mathbf{D_{F,disc}}}$ that lie within the current interval, i.e,

$$\overline{\mathbf{D_{F,disc}}} \leq \mathbf{arr[i]} \;\; \& \;\; \overline{\mathbf{D_{F,disc}}} > \mathbf{arr[i+1]}$$

  5.2) Set the value of the depth segment to the specified interval, i.e, $\overline{\mathbf{D_{F,disc}}} = \mathbf{arr[i]}$
end

Figure 18. Hole-filled image obtained after using Algorithm 4 ($\overline{I_F}$)

Algorithm 4: Image Hole Filling

Input : Synthesized view $\mathbf{I_F}$, Synthesized depth $\overline{\mathbf{D_F}}$, Occlusion mask M, depth segment mask M_{prev} initialized to zeros and dimensions (width W and height H)

Output: The hole filled synthesized view $\overline{\mathbf{I_F}}$

1. Initialize $\overline{\mathbf{I_F}} = \mathbf{I_F}$.
2. Determine all unique values in $\overline{\mathbf{D_F}}$. If this value is too high, say over 500 (continuous depth map), use Algorithm 3 to obtain discrete values.
3. Let $\mathbf{d^u}$ be the array of unique values in ascending order, and S be the number of unique values.

for *s = S* **to** *2* **do**

3.1) Depth mask $\mathbf{D_s}$ corresponding to the depth step:

$$\mathbf{D_s} = (\overline{\mathbf{D_F}} > d^u(s-1)) \;\;\&\;\; (\overline{\mathbf{D_F}} \leq d^u(s))$$

where $>$, $\leq$ and $\&$ are the element-wise matrix greater than, less than or equal to and AND operations.

3.2) Image segment $\mathbf{I_s}$ corresponding to the depth mask:

$$\mathbf{I_s} = \mathbf{I_F} \cdot \mathbf{D_s}$$

where $\cdot$ is the element-wise matrix product.

3.3) Current Occlusion Mask for the depth step:

$$\mathbf{M}_{curr} = \mathbf{M} \cdot \mathbf{D_s}$$

3.4) Update $\mathbf{M_{curr}}$ with previous mask

$$\mathbf{M}_{curr} = \mathbf{M}_{curr} \parallel \mathbf{M}_{prev}$$

where $\parallel$ is element-wise OR condition.

for *x = 1* **to** *H* **do**

for *y = 1* **to** *W* **do**

if $\mathbf{M_{curr}}(x, y) = 1$ **then**

3.5.1) Find nearest valid pixels on the same row $I_s(x', y')$, where (x', y') is the location of the valid pixel.

3.5.2) Update value of $\overline{I_F}(x, y) = I_s(x', y')$

3.5.3) Update $M_{curr}(x, y) = 0$

3.5.4) Update $M(x, y) = 0$

end

end

end

3.6) Propagate the current occlusion mask to the next step:

$$\mathbf{M}_{prev} = \mathbf{M}_{curr}$$

end

4) Apply low-pass filtering on the filled-in occluded areas in $\overline{\mathbf{I}_F}$.

Figure 19. Final result of pipeline after using Algorithm 5 ($\overline{I_F^{DZ}}$)

Figure 20. Another sample input image

2.6. Shallow Depth of Field (SDoF)

After implementing view synthesis and occlusion handling, we can apply the shallow depth of field (SDoF) effect [12] to $\overline{\mathbf{I}_F}$. This involves the application of depth-aware blurring. This effect is used for two reasons: 1) enhance viewer attention to the objects in focus, and 2) hide imperfections due to image warping, image fusion and hole filling steps. The diameter c of the blur kernel is called the circle of confusion (CoC). The paper [5] derives an expression for the circle of confusion diameter as a function of the translation t. However, I used a simpler approach that provided a set of sigma values for blurring at different discrete depths. The procedure is described in Algorithm 5 and the result is shown in Figure 19. Another sample result for the input in Figure 20 is shown in Figure 21.

3. Implementation Issues and Considerations

1. Light field depth estimation algorithms provide relative depths of objects in the scene, and not absolute distances. The paper [5] uses absolute depths and focal lengths for geometry-based calculations for zoom-factor and depth-warping. For the purpose of this implementation, these values were scaled appropriately to get relative focal lengths in the scene. The depth map obtained from the light field data was also scaled

Figure 21. Final result of pipeline with sample input 20

between 0 and 1.

2. Z-buffering and forward warping in Python seemed like they would take very long, but the individual images were relatively low-resolution and so it was feasible to run for-loops over all pixels. However, given a high resolution light-field image or a video stream, this would require a lot more time to process every frame.

3. PlenoptiCam [4] had a GUI that made it difficult to operate with intermediate results and add steps to the pipeline, and also had a very roundabout and lengthy calibration procedure. PlenPy [9] provided a better (faster) technique and allowed library-like access to key APIs and functions, which made it easier to use and modify off-the-shelf. The resulting depth maps also seemed better than PlenoptiCam.

4. The images from the Lytro camera are RAW, meaning that they need to be demosaicked and calibrated before usage. Fortunately, PlenPy [9] was able to do this. However, it was easier to use a dataset [7] for quicker prototyping during testing instead of the camera.

5. Shallow Depth of Field (SDoF) requires better padding around depth boundaries for giving reasonable blending results. I corrected this by setting all values in the depth masked regions to twice the actual value, then dividing the entire map by 2. This caused the entire image to become much brighter, so I scaled it by the maximum of the input image intensity, finally providing good results.

4. Contributions of this Paper

4.1. Open-Source Release of Code

Since the original paper on which this project is based [5] did not release their code, I have created an open-source implementation of this paper with the extensions described here in this report using PlenPy [9] for light

Algorithm 5: Shallow Depth of Field (SDoF)

Input : Discretized Depth Map $\overline{\mathbf{D}_{\mathbf{F},\mathbf{disc}}}$ (can be obtained using Algorithm 3) with dimensions (width W and height H), inpainted image $\overline{I_F}$, number of unique depth values (N), preferred focus depth (D)

Output: The final output image $\mathbf{I}_{\mathbf{F}}^{\mathbf{DZ}}$

1. Obtain a set of sigma values for blurring at different depths (clipped to the range between 0 and 11) using [2].

$$\overline{\sigma} = round(abs(\overline{\mathbf{D}_{\mathbf{F},\mathbf{disc}}}) - D) * 11)$$

2. Obtain the unique depth values d_u from $\overline{\mathbf{D}_{\mathbf{F},\mathbf{disc}}}$, and initialize the output image $\mathbf{I}_{\mathbf{F}}^{\mathbf{DZ}}$.

3. Loop over the depths:

for *i = 1* **to** *len(d_u)* **do**

3.1) Get image segments from $\overline{\mathbf{D}_{\mathbf{F},\mathbf{disc}}}$ that lie within the current interval, i.e,

$$abs(\overline{\mathbf{D}_{\mathbf{F},\mathbf{disc}}} - d_u[i]) < \epsilon$$

Here I used a different approach for padding that ensured the edges of different depths do not get attenuated due to the blurring.

3.2) Use the mask obtained from the previous section to form an image depth segment.

$$I_{seg} = M_{depth} \cdot \overline{I_F}$$

3.3) Blur the above image segment by using a kernel of the size obtained from $\overline{\sigma}$.

$$I_{blur} = blur(I_{seg}, \overline{\sigma}[i])$$

3.4) Use this blurred image and update the output image with the mask:

$$\mathbf{I}_{\mathbf{F}}^{\mathbf{DZ}} \mathrel{+}= I_{blur} \cdot M_{depth}$$

end

4) Scale the output image based on the input maximum, since the padding would have caused it to become brighter than necessary.

$$\mathbf{I}_{\mathbf{F}}^{\mathbf{DZ}} *= \overline{I_F}.max()/\mathbf{I}_{\mathbf{F}}^{\mathbf{DZ}}.max()$$

5) Median blur (low-pass filter) the image for hole filling and to take care of any artifacts from the discretization prior to blurring.

field processing, and have provided the code here under the GNU GPL license (like PlenPy itself): https://github.com/themathgeek13/plenpy/blob/main/processdepth.py. It is implemented using Python, and requires the libraries OpenCV, Numpy, Scipy, Scikit-Learn and PlenPy.

4.2. Extension of Prior Work

This paper provides an extension of prior work [5] through the following approaches:

1. Improved depth map estimate obtained from a light field instead of monocular depth estimation from a single RGB image.

2. Provides an algorithm for using continuous-valued depth maps through K-Means clustering based discretization.

3. Provides an improved algorithm for Shallow Depth of Field (SDoF) and smoother merging of multiple image segments to prevent artifacts at the edges.

5. Conclusion and Future Work

In conclusion, this paper has provided an implementation of the reference paper [5] for the single-shot camera pipeline. It has also provided extensions (where necessary or applicable) by using a light field camera and the depth map obtained from the structure tensor [13] approach, for which I took advantage of the open-source PlenPy [9] library and the Lytro Illum dataset from INRIA [7]. Finally, the resulting images have shown to be of reasonable visual fidelity and the code has been open-sourced to allow for future contributions and improvements.

In terms of future work, I would like to explore better image in-painting techniques as well as depth map hole filling techniques, because the current algorithm 2 based on nearest neighbors is quite simple and leaves quite a few artifacts in the final result. I did experiment with a paper on Generative Image Inpainting [14], and based on the results I obtained for some natural images and masks, I think this might be able to replace the two algorithms entirely while providing higher quality reconstructions.

I also think that the view synthesis could be improved by taking advantage of the multiple viewpoints available through the light field camera. A recent paper on Stereo Magnification [16] describes how novel views can be generated using the multiplane images (MPIs) obtained from a stereo pair (like in the light field camera). They also show that these can extrapolate significantly beyond the input baseline, which is essential since light field cameras like the Lytro Illum inherently have a very narrow and limited baseline due to their design.

References

[1] Abhishek Badki, Orazio Gallo, Jan Kautz, and Pradeep Sen. Computational zoom: A framework for post-capture image composition. *ACM Trans. Graph.*, 36(4), July 2017. 1

[2] bram0101. Bokeh from depth map. https://computergraphics.stackexchange.com/questions/7737/bokeh-from-depth-map. 8

[3] Rahul Garg, Neal Wadhwa, Sameer Ansari, and Jonathan T. Barron. Learning single camera depth estimation using dual-pixels, 2019. 1

[4] Christopher Hahne and Amar Aggoun. Plenopticam v1.0: A light-field imaging framework, 2020. 8

[5] Yangwen Liang, Rohit Ranade, Shuangquan Wang, Dongwoon Bai, and Jungwon Lee. The "vertigo effect" on your smartphone: Dolly zoom via single shot view synthesis. pages 1407–1415, 06 2020. 1, 2, 3, 5, 6, 7, 8, 9

[6] Ali M. Rotate and zoom an image without changing its dimensions. https://stackoverflow.com/questions/37119071/scipy-rotate-and-zoom-an-image-without-changing-its-dimensions/37121993#37121993. 3

[7] M. Le Pendu, X. Jiang, and C. Guillemot. Light field inpainting propagation via low rank matrix completion. *IEEE Transactions on Image Processing*, 27:1981–1993, 2018. 1, 8, 9

[8] Colvin Pitts, Timothy James Knight, Chia-Kai Liang, and Yi-Ren Ng. Generating dolly zoom effect using light field image data, U.S. Patent US8971625B2, June 2014. 1

[9] Maximilian Schambach and Fernando Puente León. Microlens array grid estimation, light field decoding, and calibration. *IEEE Transactions on Computational Imaging*, 6:591–603, 2020. 2, 8, 9

[10] Richard Szeliski. *Computer Vision: Algorithms and Applications*. Springer-Verlag, Berlin, Heidelberg, 1st edition, 2010. 3

[11] unutbu. Find nearest value in numpy array. https://stackoverflow.com/questions/2566412/find-nearest-value-in-numpy-array. 6

[12] 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*, 37(4):1–13, Aug 2018. 7

[13] S. Wanner and B. Goldluecke. Globally consistent depth labeling of 4d light fields. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 41–48, 2012. 2, 9

[14] Jiahui Yu, Zhe Lin, Jimei Yang, Xiaohui Shen, Xin Lu, and Thomas S Huang. Generative image inpainting with contextual attention. *arXiv preprint arXiv:1801.07892*, 2018. 9

[15] Christopher Zach, Thomas Pock, and Horst Bischof. A duality based approach for realtime tv-l 1 optical flow. In *Joint pattern recognition symposium*, pages 214–223. Springer, 2007. 2

[16] Tinghui Zhou, Richard Tucker, John Flynn, Graham Fyffe, and Noah Snavely. Stereo magnification: Learning view synthesis using multiplane images. In *SIGGRAPH*, 2018. 9