Using Depth Mapping to realize Bokeh effect with a single camera Android device

EE368 Project Report

Authors (SCPD students): Jie Gong, Ran Liu, Pradeep Vukkadala

***Abstract*- In this paper we seek to produce a bokeh effect with a single image taken from an Android device by post processing. Depth mapping is the core of Bokeh effect production. A depth map is an estimate of depth at each pixel in the photo which can be used to identify portions of the image that are far away and belong to the background and therefore apply a digital blur to the background. We present algorithms to determine the defocus map from a single input image. We obtain a sparse defocus map by calculating the ratio of gradients from original image and reblured image. Then, full defocus map is obtained by propagating values from edges to entire image by using nearest neighbor method and matting Laplacian. Based on the defocus map, foreground and background segmentation is carried out. We show that by enhancing the blurriness of the background while preserving the foreground, bokeh effect is achieved. Finally, we propose an algorithm to add cartoon-like effect on the input image.**

Keyword: bokeh effect; matting Laplacian; defocus map; cartoon; Android device

I.INTRODUCTION

Artistic enhancement of photographs is a very popular field of research with several applications that are common place in the world of smartphone photography. Recently researchers have sought to produce an effect known as Bokeh in smartphone photos using digital image processing techniques. Bokeh is the shallow-depth of field effect which blurs the background of portrait photos (typically) to bring emphasis to the subject in the foreground. Figure 1 shows the same image with and without Bokeh effect.

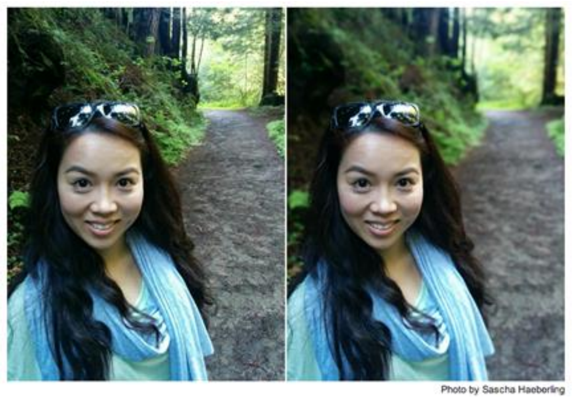


Figure 1. image without and with Bokeh effect

Bokeh effect is usually achieved in high end SLR cameras using portrait lenses that are relatively large in size and have a shallow depth of field. It is extremely difficult to achieve the same effect (physically) in smart phones which have miniaturized camera lenses and sensors. However, the latest iPhone 7 has a portrait mode which can produce Bokeh effect thanks to the dual cameras configuration. To compete with iPhone 7, Google recently also announced that the latest Google Pixel Phone can take photos with Bokeh effect, which would be achieved by taking 2 photos at different depths to camera and combining then via software. There is a gap that neither of two biggest players can achieve Bokeh effect only using a single image from a single smartphone camera. In this project we seek to fill this gap. We are planning to produce a bokeh effect with photos taken using an Android device by post processing the photos within the Android device. The photos can also come from Photo Stream. The new images with Bokeh can be saved into Photo Stream. At the core of Bokeh effect production in smartphone photography is depth mapping. A depth map (an estimate of depth at each pixel in the photo) is used to identify portions of the image that are far away and belong to the background and therefore apply a digital blur to the background. As mentioned early, Bokeh effect is typically present in portrait photos. For non-portrait photos, we are seeking to apply cartoon effect on the foreground as a way of artistic enhancement. What's more, other ways of artistic enhancement like changing the background of images can also be performed based on the depth map.

Figure 2: examples of artistic enhancement by taking advantage of depth map

II.LITERATURE REVIEW

In recent years, there are a variety of methods were proposed to recover the defocus map from a single image. Elder and Zucker [1] used the first and second order derivatives of the input image to determine the size of the Gaussian blur kernel. Bae and Durand [2] extended their work by adding a bilateral filter to remove outliers and estimate the blur amount of the whole image. Using defocus map, they increased the blurriness of the blurry regions and kept the in focus region sharp. Levin et al. [3]

converted a statistical model to recover defocus map with a modified camera. Tai and Brown [4] used local contrast prior to measurements into a defocus map and then apply MRF propagation to refine the estimated defocus map. The advantage of their method is that it does not rely on frequency decomposition of the input image. It only compares the local gradients with local contrast. Zhuo and Sim [5] generated a full defocus map with good accuracy by using matting interpolation. They also discussed a method to remove the blur texture ambiguity. In our paper, first, we use a Gaussian blur kernel to reblur the image and calculate the ratio between the gradients. Secondly, we adopt nearest neighbor and matting Laplacian to determine the full defocus map. The defocus map is segmented into foreground and background. Then, bokeh effect is generated by applying blur kernel to further blur the background while keep the foreground sharpen. Finally, we proposed an algorithm based on bilateral filtering and edge detection to add cartoon-like effect on the foreground of the input image.

III.ALGORITHM

3.1 Depth Estimation

Defocus Blur concept

When a picture is taken with a camera, portions of the image look sharp and portions of the image look blurry depending of the depth of the scene. Portions of the scene that lie on the focal plane will look sharp while portions of the image that are away from the focal plan will appear blurry. The blurriness is associated with defocus when the scene is imaged on camera sensor. Light rays from points in the scene that are in-focus will reach points on the sensor whereas light rays from points that are out-of-focus will be spread across multiple points on the sensor within an area known as circle of confusion [5]. The size (diameter) of the circle of confusion is associated with the amount of defocus for a given point in the scene and is given by:

$$c = \frac{|d - d_f|}{d} \frac{f_0^2}{N(d_f - f_0)} \tag{1}$$

where d_f is distance between camera lens and focal plane, d is distance between a point in scene to the camera lens, f_0 is focal length, and N is stop number of the camera.

This leads to the out-of-focus points being imaged with a blur effect known as defocus blur. The defocus blur can be approximated as a Gaussian blur (with standard-deviation σ_{db}) which varies with the distance of the object (d). We use the method described by Zhuo et al [5] to make defocus estimation in our project. In this method a given image is re-blurred using a Gaussian kernel with standard-deviation σ_r. Gradient magnitude is computed for both the original image and the re-blurred image. The ratio of the gradient magnitude (R) of the original image to the re-blurred image is calculated. The gradient magnitude ratio gives a measure of the amount of defocus blur in the image. Sharper in-focus objects in the image will result in large R values and blurry out-of-focus objects will result in smaller R values. This idea is exploited to estimate defocus blur from a single image. It should be noted that R will reach maximum value at edge locations of an image. Also at edge locations, the undesired impact of blurry texture in in-focus objects on interpretation of R is mitigated. Thus we only examine R value at edge locations in the image by performing edge-detection and excluding non-edge regions. Finally defocus-blur is estimated as follows:

$$R = \frac{|\nabla C|}{|\nabla C_r|} = \sqrt{\frac{\sigma_{db}^2 + \sigma_r^2}{\sigma_{db}^2}} \tag{2}$$

$$\sigma_{db} = \frac{1}{|\sqrt{R^2 - 1}} \tag{3}$$

Advantages

In this method depth can be estimated using a single image input, does not require any user intervention, and is fully automatic. Thus this method provides a significant advantage in terms of ease of use and applicability compared to other depth estimation techniques such as those that require multiple images of a scene with modified illumination [7], or others that require the user to intervene and provide scribbles or a tri-map input [8].

Gradient-ratio

We first convert the image to gray-scale to obtain intensity map. We compute gradients in x- and y-directions and take the magnitude at every pixel. We do the same for the re-blurred image and take gradient-ratio at every pixel. We apply a mild median filter to reduce noise in the gradient magnitude

$$||\nabla C(x,y)|| = \sqrt{\nabla C_x^2 + \nabla C_y^2} \tag{4}$$

Edge-detection

We perform canny edge-detection on the original image to a binary edge map. To set the lower and upper threshold for canny edge detection, we first perform edge detection in Matlab with default setting and get threshold values generated by Matlab. We modify this threshold to take lower threshold value equal to 85% of upper threshold value. We observed desirable edge detection with these settings.

Sparse Defocus map

We use the binary edge map to obtain a sparse gradient ratio map. We convert this to sparse Defocus map using eq. (3). This sparse defocus map can still contain inaccuracies due to noise and weak edges in the image. To mitigate this we apply a bilateral filter to the sparse defocus map. The bilateral filter reduces noise using a spatial Gaussian kernel without heavily blurring the edges. This is achieved by weighting the spatial Gaussian kernel with a range Gaussian kernel [8]. The range Gaussian kernel works by decreasing the influence of a neighboring pixel whose intensity difference relative to the center pixel is above/below a certain threshold. We adopted a bilateral filter code developed in [9] for our project. We extended the code to be able to perform bilateral filtering on sparse map. One disadvantage of the bilateral filter is that it is an non-linear filter and thus computational expensive relative to a linear filter such as Gaussian filter that can be efficiently implemented using techniques such as fast Fourier transform. We use a spatial filter σ = 5 and range filter σ = 30%.

Propagation to full defocus map with In-Paint

The sparse defocus map needs to be propagated thru the full image to obtain defocus for every pixel in the image. One simple way to achieve this is thru nearest-neighbor interpolation. We first tried griddata/scatteredInterpolant in Matlab to interpolate defocus at non-edge regions. The results were not that great. Sharp transitions and inaccurate defocus estimations were observed in several cases. We tried another form of interpolation/extrapolation technique known as “inpaint” which seeks to fill missing data in images [10]. We adopted the Matlab implementation of inpaint in this reference [11]. The inpaint produces better interpolation results compared to nearest-neighbor. In some cases the inpaint result was sufficiently good for the purpose of multi-level depth segmentation. However the inpaint result often tended to be noisy and did not produce a sufficiently fine interpolation of defocus near object edges.

Propagation to full defocus map with Matting-Laplacian

We also evaluated an approach for interpolation that uses the matting-Laplacian matrix (L) to obtain full defocus map [6]. A matte (α) for a given image is a transparency mask that ranges between 0 and 1 which can be used to separate background from foreground.

$$C_i = \alpha_i F_i + (1-\alpha_i)B_i, \qquad (5)$$

where C_i is the RGB image (color components at pixel i), F_i is foreground, B_i is background, and α_i is the matte. L is an NxN symmetric matrix whose element (i,j) is given by

$$L_{i,j} = \sum_{k|i,j\in w_k} \delta_{i,j} - \frac{1+(C_i-\mu_k)^T\left(R_k+\frac{\varepsilon}{|w_k|}U\right)^{-1}(C_j-\mu_k)}{|w_k|} \qquad (6)$$

where $\delta_{i,j}$ is the Kronecker delta, μ_k and R_k are the mean and covariance matrix of the colors in the local window w_k, $|w_k|$ is the number of pixels in the local window, j is pixel index within the local window, ε is a regularization parameter, and U is a 3x3 identity matrix. A closed-form solution for creating the α matte using matting-Laplacian matrix was developed by Levin et al [6] and extended by He et al and Pitie [7,12] to provide an alternate and efficient implementation for higher resolution images. The key assumption in the Levin model that allows for a closed form solution is that α is a linear combination of color components in a local window.

$$\alpha_i = a_i^T C_i + b_i, \qquad (7)$$

where a_i = $[a_i^R, a_i^G, a_i^B]^T$ is a 3x1 vector and b_i is a scalar which are assumed to be constant within the local window w_k. Thus α is obtained by minimizing the following cost function

$$E(\alpha) = \alpha^T L\alpha + \lambda(\alpha-\beta)^T D(\alpha-\beta) \qquad (8)$$

where β is typically a trimap provided by the user indicating with 1 areas which are believed to be foreground, with 0 area that are believed to be background, and a value between 0 and 1 for areas whose classification is fuzzy, D is a sparse diagonal matrix whose elements are 1 for constraint pixels (pixels for which the classification is confidently known and to be enforced), and λ is a scalar that implies additional confidence between known α values and the interpolated values. For our problem we replace α with d which is the full defocus map to be estimated, β with sparse depth map $\hat{d}$ at edge locations, and D with a diagonal matrix whose elements are 1 for an edge pixel and 0 otherwise. A small λ value is used to keep soft constraint on d.

$$E(d) = d^T Ld + \lambda(d-\hat{d})^T D(d-\hat{d}) \qquad (9)$$

Alternatively optimal value for d can be obtained by solved the following sparse linear system:

$$(L+\lambda D)d = \lambda D\hat{d} \qquad (10)$$

In this project we adopted a matting Laplacian Matlab code developed by Zhuo et al and use Matlab back slash operator (\) to solve the sparse linear system.

Depth map

So far we have discussed obtaining the defocus map. The defocus blur is directly related to the size of the circle of confusion (σ_{db} = $k*c$). Combining with eq. (1) one can obtain the depth map (distance map) of each pixel in the image provided the camera parameter values are known. Due to focal plan ambiguity, it is usually assumed that all defocussed pixel lie on one side of the focal plane during depth estimation. In this project we simply use the defocus map as a proxy for depth map when realizing

3.2Post-processing

3.2.1Bokeh effect

After depth map is available, images can be segmented into different segmentations. Theoretically, the more segmentation we have, the better Bokeh effects we have. However, the resolution of the depth map is not so high that there are a lot of unsmooth transition between different segmentations. To achieve Bokeh effects, only 2 segments are applied here, resulting in a foreground and background. Then the binary masks of the foreground and background are obtained. The original RBG image is blurred using a Gaussian kernel. The binary mask of the background will be applied on the blurred RBG image and the binary mask of the foreground will be applied on the original RBG image. Finally, the two images are added together to produce the image with Bokeh effect.

In addition, due to the imperfect segmentation, some features on the faces like the eyebrow, jaw and part of the hair are usually segmented into background. Since Bokeh effect is typically present in portrait photos, face detection is applied on the original images. If the face detection is positive and meanwhile the average of depth within the face region is small, i.e. face on the foreground, no blur will be applied regardless of segmentation results. Matlab has an image vision toolbox which implements face detection function. The function will return the position and size of the bounding box covering the face. And the bounding box only covers the face but excludes the hair. The position and size of the bounding box is adjusted to include the hair so that the hair won't be blurred.

3.2.2 Cartoon

First, we use the bilateral filtering to smooth the original input image while preserving edges. The popular Canny edge detector is employed to get a proper number of edges. Then, based on the defocus map, we generate a binarized image to separate the foreground from the image. Finally, we did the morphological operation on the edges and combine the edge map with the smoothed input image.

IV. ANDROID APP

Information flow

We built an Android app for this project in order to deploy our application on mobile phones and tablets. The app provides an interface for the user to take a picture with the mobile device camera, process the image using image processing algorithms we described above and display the processed image on the mobile device. Our image processing algorithm consists of several processing steps including linear filtering, non-linear filtering, solution to a sparse yet large linear system, morphological processing, etc. This requires a high amount computational power in order to complete the image processing in a reasonable amount of time (<1 min). So we employed a server to offload the image processing task.

Setting up webserver

We setup a webserver on a Dell 4800 workstation with Intel Core i7 processor with 8 cores and 16 GB RAM running 64-bit Windows 7. We setup a webserver using WAMP 64-bit software which comes with Apache, MySQL, and PHP services. We setup a local host thru port 8080 instead of the default port 80 in order to have stable performance by avoiding potential conflict with other windows system services that might listen to port 80. We also had to make other modifications to both webserver configuration and Windows firewall settings in order to make the webserver functional.

Matlab code

The Matlab code (*computeBOKEHLoop.m*) on the server keeps running in a perpetual loop looking for signal that the image has been uploaded and ready for processing. This perpetual loop saves on processing time by eliminating Matlab initialization time which can take a few seconds. Once Matlab receives signal that the image has been uploaded and ready for processing it calls the main function (*DefocusMap_v8.m*) with the input image path and desired output image path as input variables. The function computes the defocus map, using which it creates a Bokeh map (background blur that varies with depth), finally applies a cartoon effect to the foreground subjects, and saves the processed image to the output location. The code also creates a signal that the processed image is ready. We adopted this Matlab code from Android tutorial #3 provided by this class.

Client-Server communication

Client (Android App) to server communication is achieved by a PHP script that runs on the webserver. We adopted the script from Android tutorial #3. The PHP script (*computeBOKEHLoop.php*) is first invoked by the Android app which passes the location of the input image to the PHP script. The script takes the image from Android device and saves it in the root folder of the webserver. Along with the image, the PHP script also saves a text file containing the location path of the image to signal to Matlab that the input image is ready. After this the PHP script waits for signal from Matlab that the processed image is ready. Once the image is ready, script streams the image file back to the Android app.

Android/Java code

All project members are new to programming in Android. We also did not have any experience with programming in Java before starting this project. So it was very challenging for us to develop an Android App. We adopted java codes provided by the class as well as codes from online tutorials for use in building our app. We have two different App UIs for this project.

App UI I

In the first app UI we have a very simple user interface. When the app is launched the app title is displayed and camera view is enabled. After setting the right scene the image can be captured by pressing the volume up button. Once the image is captured, the Android device connects to the server and calls the PHP script while passing the location of the captured image.

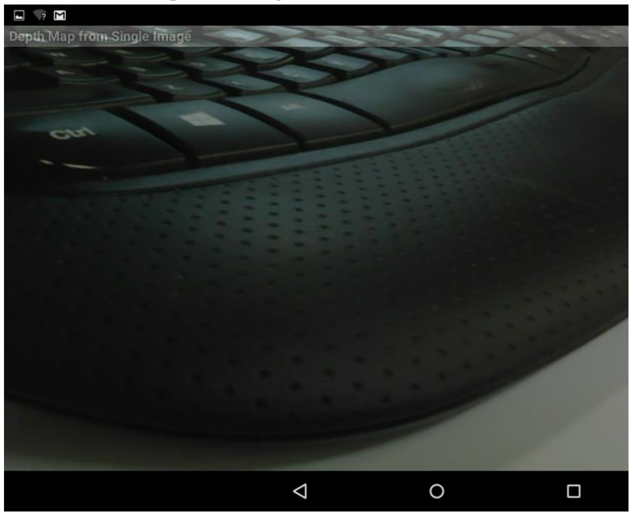


Figure A: Screen capture of App UI I

App UI II

In the second app UI we added the ability for the user to either take a picture with the device camera or select an existing image file from the device gallery. The picture (either taken with the camera or selected from gallery) is displayed in an ImageView frame on the app UI. We built an upload button on the app which may be used to upload the image being displayed to server for image processing.

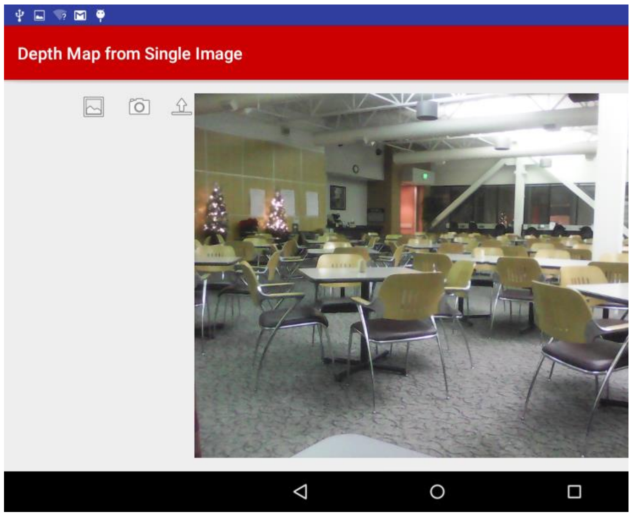
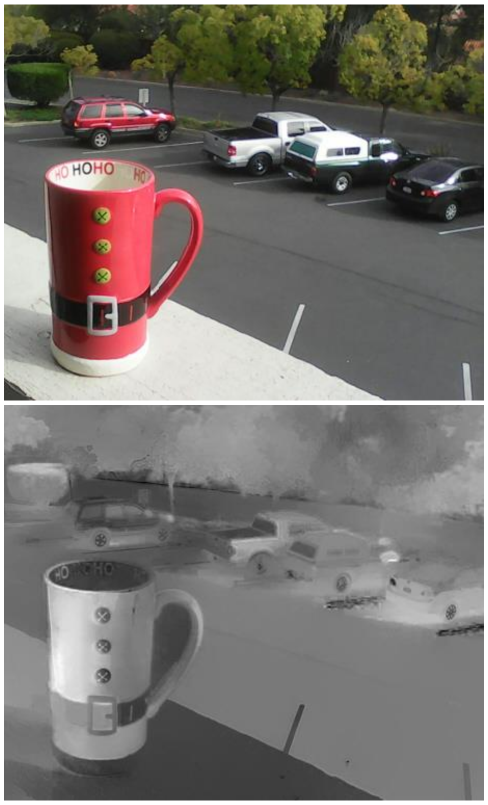


Figure B: Screen capture of App UI II

We successfully implemented functionality for the image capture button, image select button, and image display button. However we were not able to complete the implementation for image upload button due time limitation.

DISCUSSION, LIMITATIONS, AND FUTURE WORK

One essential limitation of this methodology of depth estimation is the requirement of presence of defocus blur. Defocus blur is more pronounced in cameras which have a shallow depth of field and is less pronounced in camera with a large depth of field. Usually more expensive cameras have shallower depth of field whereas cheaper cameras have larger depth of field. Hence this method will tend to be inaccurate and fail in images where the background has little defocus blur. In Fig. C it is seen that the face of the mug appear bright in the defocus map (darker means nearer and brighter means farther) even though the mug is in the foreground. This is because of the inherently blurry texture of the foreground object confounding the algorithm. In Fig. D the image is captured inside a length bus using a camera with a large depth of field. As a result even the far regions of the bus are in relatively good focus. This results in poor depth estimation.



a)

b)

Figure C: a) Original image taken on Android camera, b) Estimated full defocus map

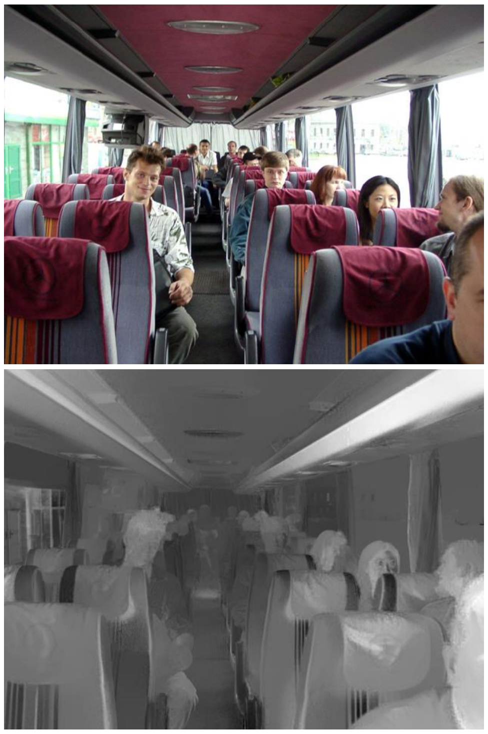


Figure D: a) Original image inside a length bus, b) Estimated full defocus map

To make our Android app useful in cases where the app is installed on cell phone cameras with a large depth of field, we have attempted to create a gallery selection option in our app. We have tried to implement this feature where the user might be able to apply our image processing on existing images (taken with other suitable cameras) downloaded to the Android device.

A future work area that would be interesting to examine is to try to analytically characterize the conditions in which this methodology of estimating depth based on defocus blur will be successful. For example it may be reasoned that if a scene with sufficient depth is captured by a camera, the captured image will contain a finite level of defocus blur no matter what type of lens is used. A camera with shallow depth of field will produce an image with a large amount of defocus blur whereas a camera with a large depth of field will produce an image with less yet finite amount of defocus blur. So it may further be reasoned that at a certain resolution requirement it may be possible to extract defocus blur for a large depth of field camera. So it may be possible to derive a relationship between the ability to extract defocus blur and camera parameters such as depth of field, sensor spatial resolution, sensor bit-depth, etc.

V. RESULTS

Inaccurate blur estimation may occurs at weak edges, noise or soft features. We solve this problem by applying joint bilateral filtering (JBF) on the edge locations. Sparse defocus map is obtained from input Fig. 3a) and here we compare the results before and after applying JBF. Fig. 3b) shows many bright colored spots randomly distributed across the whole image. After applying JBF (fig. 3c), those errors are almost completely removed from the defocus map while edges are well preserved. Thus, JBF is very effective to smooth the sparse defocus map and prevent errors propagate in the step of interpolation. We applied different methods on the sparse defocus map to obtain the full defocus map. It can be seen from original image (fig. 4a) that the two pumpkins at the lower part of the image have much sharper edges compared to the others. After that, full defocus maps were obtained by using nearest neighbor method (fig. 4b) and matting laplacian (fig. 4c). Both estimated full maps can capture continuous change of depth. The nearest neighbor method produce coarse defocus map and the pumpkins are not well separated with the background. In contrast, matting laplacian method is able to produce a more accurate defocus map.

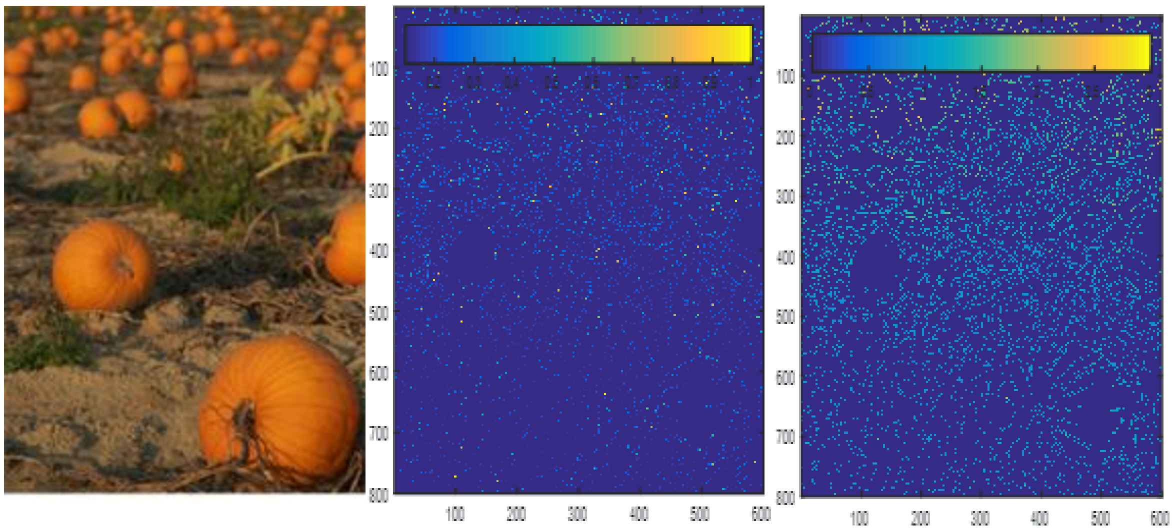


Figure3: a) input image and sparse defocus maps before (b) and after (c) applying the bilateral filter

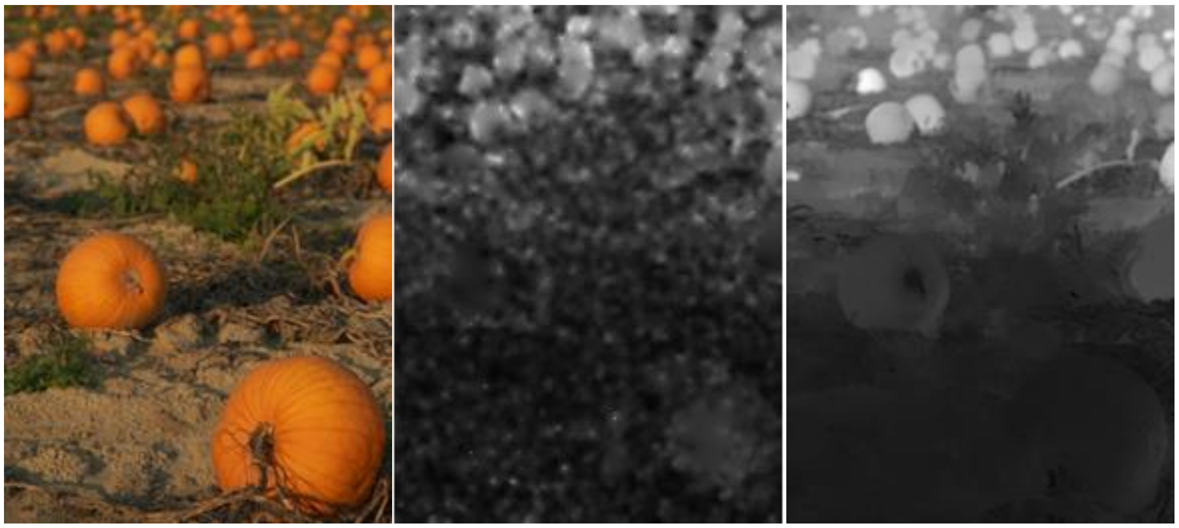
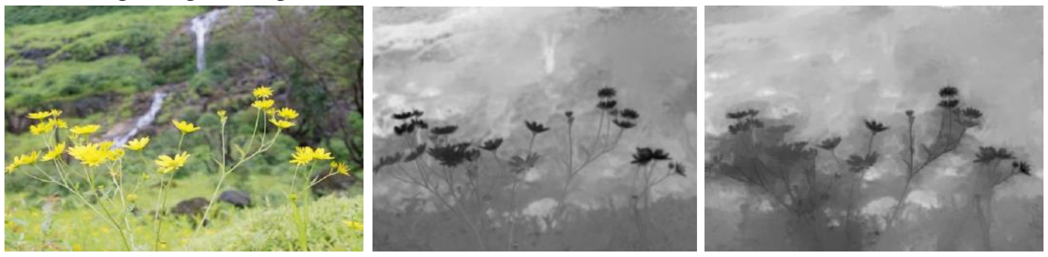


Figure 4: Obtain defocus map from input image a) by applying nearest neighbor method b) and matting laplacian c). Matting laplacian results in defocus map with good accuracy.

Figure 5 shows the two examples of input images, depth map using algorithm from the paper and depth map using our own algorithm. The difference between the two depth maps is very marginal. However, there are some regions that our algorithm shows inferior results compared to the ground truth. Fox example, at the center of the flower of the first example, depth map using our own algorithm shows a slight higher magnitude than some of the background. In addition, the white flower in the 2nd image shows very strong magnitude (i.e. very far depth).



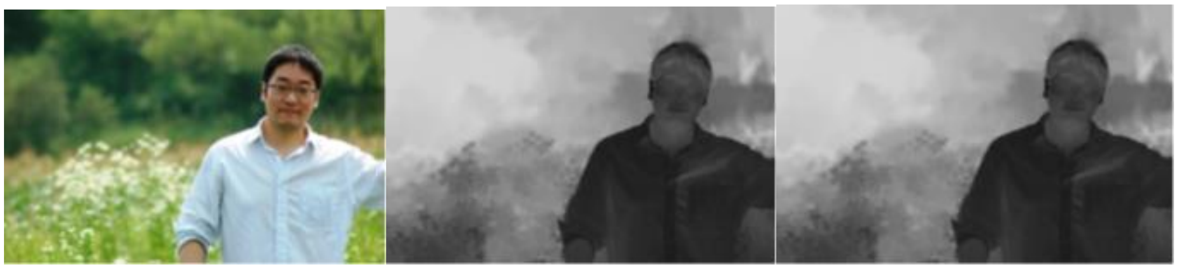


Figure 5: input images, depth map using algorithm from the paper [5] and depth map using our own algorithm (from left to right)

Here, we are using Figure 6a) as the input image for Bokeh post-processing. Figure 6b) displays the foreground and background that are segmented based on the depth map. Then face detection is applied on this image, and Figure 7 a) shows the results of the face detection from Matlab. Then, the position and size of the bounding box is adjusted to include the hair, as shown in Figure 7b). Finally, the Bokeh image with the correction using face detection is shown in Figure 7c). As you can see, features on the faces like the eyebrow, jaw and part of the hair are not blurred anymore.

Figure 7c) shows the image after applying Bokeh effect. As shown in the figure, the background is successfully blurred, while most part of the foreground is well kept.

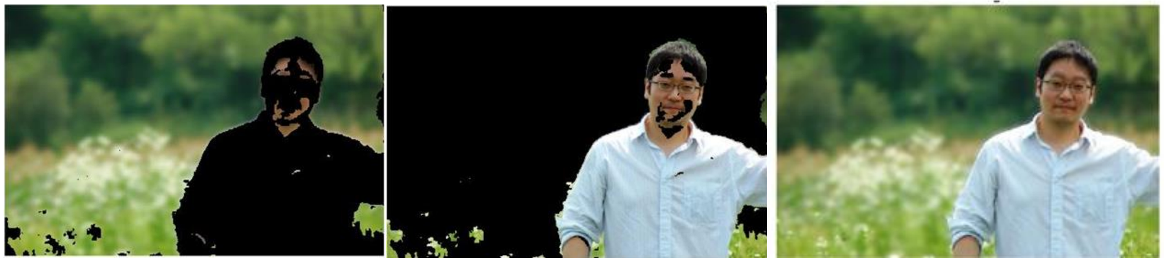


Figure 6: a) background based on segmentation; b) foreground based on segmentation; c) image with bokeh effect without face protection

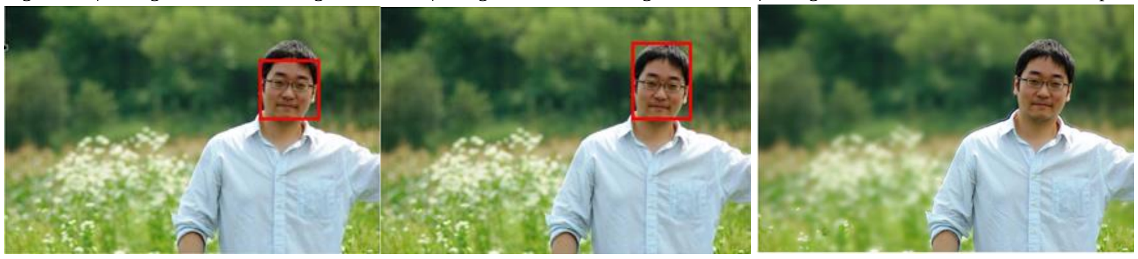


Figure 7: a) Face detection results from Matlab; b) Adjusted Face detection results; c)Bokeh image with face protection

We apply bilateral filtering on the input image (fig. 8a) to decompose the image into a cartoon-like component (fig. 3b). Strong edges of the input images (fig. 8c) is obtained by using Canny method. Cartoon-like effect can be achieved after combining the above two images. In our case, we only apply the cartoon effect on the foreground so that the portrait in the image is cartooned while preserving the background.

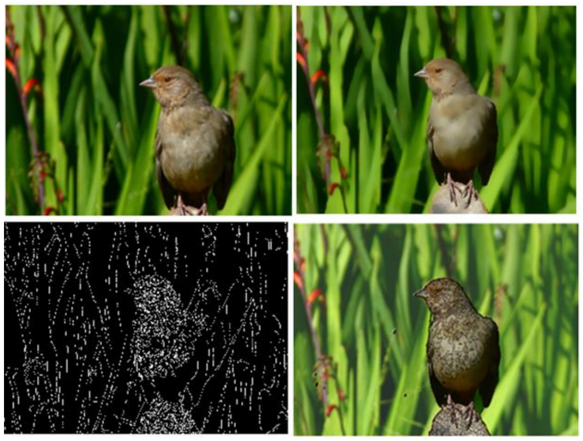


Figure 8: a) original input image; b) denoised image with JBF; c) edge map; d) cartooned image

VI. DISCUSSION, LIMITATIONS AND FUTURE WORK

Another limitation is the non-smooth transition between foreground and background on the output RBG image. Figure 9 shows one example of such non-smooth transition. The root cause of this transition is due to the different levels of blurring between different segmentations. One of the future work that the authors are considering to address this limitation is to use morphological dilation and erosion to extract the outline of the foreground. Then a smooth interpolation will be applied for the outline of the foreground.

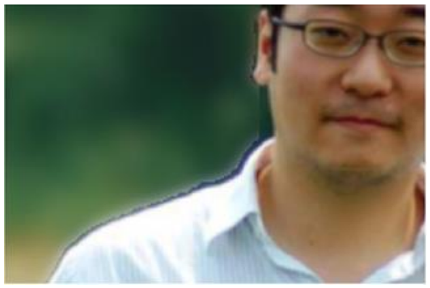


Figure 9: zoom-in of non-smooth transition

Another future work is to use the small region removal to exclude the imperfect segmentation.

VII.ACKNOWLEDGEMENT

Mr. Jean-Baptiste Boin's supports during the project are highly appreciated.

VIII.REFERENCES

[1] J. Elder, S. Zucker, Local scale control for edge detection and blur estimation, IEEE Trans. Pattern Anal. Mach. Intell. 20 (7) (1998) 699-716.

[2] S. Bae, F. Durand, Defocus magnification, Proc. Eurographics (2007) 571-579.

[3] A. Levin, R. Fergus, F. Durand, and W. T. Freeman. Image and depth from a conventional camera with a coded aperture. ACM Trans. Graphics, 2007.

[4] Y.-W. Tai, M. S. Brown, Single image defocus map estimation using local contrast prior, in: Proc. ICIP, 2009.

[5] Zhuo, Shaojie, and Terence Sim. "Defocus map estimation from a single image." *Pattern Recognition* 44.9 (2011): 1852-1858.

[6] Levin, Anat, Dani Lischinski, and Yair Weiss. "A closed-form solution to natural image matting." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 30.2 (2008): 228-242.

[7] Scharstein, Daniel, and Richard Szeliski. "High-accuracy stereo depth maps using structured light." Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on. Vol. 1. IEEE, 2003.

[7] Kaiming He, C. Rhemann, C. Rother, Xiaoou Tang, and Jian Sun, "A global sampling method for alpha matting," in Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on, June 2011, pp. 2049–2056.

[8] Fast bilateral filtering for the display of high-dynamic-range images Durand and Dorsey ACM SIGGRAPH conference (c) 2002

[9] C. Tomasi and R. Manduchi. Bilateral Filtering for Gray and Color Images. In Proceedings of the IEEE International Conference on Computer Vision, 1998

[10] Damien Garcia. 2010. Robust smoothing of gridded data in one and higher dimensions with missing values. Comput. Stat. Data Anal. 54, 4 2010

[11] D'Errico, John (2004). inpaint_nans, MATLAB Central File Exchange. Retrieved Nov 28, 2016

[12] F. Pitie, "An alternative matting Laplacian," 2016 IEEE International Conference on Image Processing (ICIP), Phoenix, AZ, USA, 2016