Panoramic Reconstruction from Multiple Light-Field Images

Buyukalp, Yasin
Stanford University
Electrical Engineering Dept.
buyukalp@stanford.edu

Cihan, Ahmet Fatih
Stanford University
Electrical Engineering Dept.
afcihan@stanford.edu

Mutlu, Mehmet
Stanford University
Electrical Engineering Dept.
mutlu@stanford.edu

Abstract

We show a light-field panoramic image construction with a highly effective and computationally modest method based on calculating focal stacks in multiple light-field images and then stitching these images individually. The method works best for Lambertian scenes, and it preserves the distinctive features of light-field imaging. Additionally, a dynamic display of the resulting panoramic images are demonstrated via a head-tracking system using a computer webcam and a method based on Viola-Jones and Kanade-Lucas-Tomasi algorithms.

1. Introduction

Panorama imaging has already become an essential part of commercial photography applications. It is considered to be a necessary feature for a camera rather than an extra feature. On the other hand, light-field (LF) imaging, where angular information is recorded in addition to the intensity information, is a new field with enormous potential in commercial photography. Especially with the emergence of the 3D imaging and display industry, LF imaging holds great promise. Hence, it can be expected that LF imaging will become widespread in the near future and everything available to the conventional imaging needs to be available in the LF domain in order to facilitate its success. Panorama is one of these applications. Therefore, demonstration and implementation of LF panorama imaging can be critical for the development of commercial LF photography products.

In this project, we aim to implement the construction of a LF panoramic image by recording multiple overlapping LF images with a Lytro camera at different camera angles. In converting multiple images to the panoramic one, we aim to preserve some of the capabilities of LF imaging such as computational refocusing, depth estimation, synthetic aperture construction, ability to shift perspective, etc. This preservation is actually required to have the distinctive effect of LF images, which is the presence of directional ray information, compared to the more traditional microlens-free imaging technique.

2. Related work

A brief investigation of the literature reveals three main methods for creating LF panoramic images. The first method, which is naïve multi-perspective image stitching [1], stitches corresponding perspective images of the input LFs individually using classical panoramic reconstruction. Despite its simplicity in implementation, since this method stitches the LF images independent of their directional domain, the resulting image might exhibit severe artifacts during the synthetic refocusing operation.

A more advanced (and computationally expensive) method to implement LF stitching is to calculate focal stacks in all images and then stitch these images individually [2]. Then, the panoramic focal stacks are merged to get a four-dimensional LF image. This method eliminates the previously mentioned synthetic refocusing difficulties with the panoramic image. However, since the data is essentially a direct concatenation of three-dimensional images, this method experiences difficulties with handling anisotropic reflections and is best suited for Lambertian scenes.

The state-of-the-art method, which processes ray entries directly [3], eliminates problems associated with occlusion boundaries, anisotropic reflections, etc. since it computes the panoramic LF directly from the individual LF images. This method works by taking multiple LF images at pre-determined angles, calculating the corresponding cylindrical LF parametrizations, and blending the resulting data. An initial calibration step is required for a specific focus and zoom setting in order to extract the intrinsic parameters of a LF camera. This calibration step is largely similar to the one reported in an earlier study [4].

3. Light-Field panorama construction

In this project, we pursue the implementation of the method proposed in Ref. 2 since this method offers a relatively fast computation, does not require camera calibration, and preserves the peculiarities of LF imaging to an acceptable degree. However, with this method, one cannot correctly account for anisotropic reflections and the handling of occlusion boundaries is successful to a certain

degree. Despite being able to perfectly preserve the advantages of LF imaging, the method described in Ref. 3 is significantly intensive in terms of computation time and requires calibration each time focus and zoom settings of the camera are modified, which occurs frequently in realistic settings.

3.1. Panoramic stitching

In this part of the project, we aim to create a panoramic focal stack that can be later used to generate different perspective views of the panoramic scene and create the four-dimensional LF data.

Firstly, we record the three images shown in Fig. 1 by using Lytro Illum such that these pictures have overlapping regions. For taking the photos, we install the camera on a tripod and rotate it horizontally around its center from left to right. Then, we process these images by the software program named LytroPowerToolBeta, which outputs a 542 x 376 x 14 x 14 LF in 16-bit png format in linear color space. The output png file is already demosaicked, exposure compensated, and adjusted to account for lens aberrations and sensor angular sensitivity variation. Finally, the images are buffered in MATLAB by using the "readLfImage" function we have written [5].

In the next step, we compute focal stacks from the LF images. This can be done thanks to the presence of multiple slightly different perspectives in the LF data. We use the fact that the camera sensor can be 'computationally' moved due to the nature of LF image formation. It can easily be shown that $l_0(x, v) = l_d(x + dv, v)$, where x and v represent the reference planes of the LF parametrization and d is the 'computational' sensor displacement amount. This equality simply tells us that our region of interest is composed of a uniform medium and light rays propagate along a straight line here. Then, one can simply perform synthetic refocusing by computing the image $i_d(x)$, which is written as

$$i_d(x) = \int_{\Omega} l_0(x - dv, v)dv.$$

This equation is showing us that simple shift and add operations are sufficient to generate a focal stack. We implement this algorithm in the 'computeFocalStack' function we have written in MATLAB. The function performs linear interpolation when sub-pixel shifts are required.

Subsequently, we calculate the all-in-focus images from the focal stack we compute. The calculation of this image allows us to calculate the stitching parameters between the three images accurately since all objects in the images are sharp. Therefore, extraction and matching of distinctive features in the images can be performed accurately. The computation of the all-in-focus starts with computing the gradients in each member of the focal stack. We define the x and y direction gradient operators as

$$\nabla_x = \begin{bmatrix} 0 & 0 & 0 \\ -1 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \nabla_y = \begin{bmatrix} 0 & -1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}.$$

Each element of the focal stack is then multiplied by these operators in the Fourier domain. We then compute the total isotropic gradient as

$$\nabla = \sqrt{\nabla_x^2 + \nabla_y^2}.$$

Finally, the all-in-focus image can be straightforwardly computed by taking pixel values that have the highest gradients among the focal stack members.

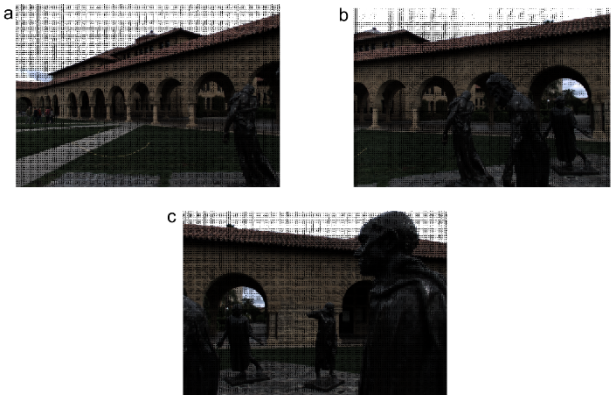

Figure 1: Images taken with Lytro Illum. Note that they are taken from left to right (a to c) and have overlapping regions to allow for feature matching later.

In order to stitch the elements of the focal stacks created from the LF images, we compute the speeded up robust features (SURF) [6] for each all-in-focus image. The idea is to separately stitch three images that correspond to the identical focal planes. In this method, interest point detection is based on the Hessian matrix and integral images are employed to reduce the computation time. After the detection of interest points, a 64-dimensional vector describes the local environment. The descriptors are created based on the Haar wavelet response of the local environment. We show the results of the SURF algorithm in Fig. 2 for the three images of interest.

Next, we find and match the correspondences between the images to calculate the required geometrical transformations to stitch the images for creating a panoramic focal stack. This matching is performed by a random sampling of the detected descriptors and involves the detection of inlying and outlying features among the images [7]. Figure 3 shows the matching features between the images and the geometric transformation of these features from one image to the other. Before the stitching,

the last step is calculating the homography according to the matched points. For this purpose, we employ a projective transform to describe the relationship of matching points between the images. Note that an affine transform can be sufficient for the panoramic stitching of far objects. However, in our case, we have objects near the camera and a projective transform yields much better results since it does not necessarily preserve the parallelism.

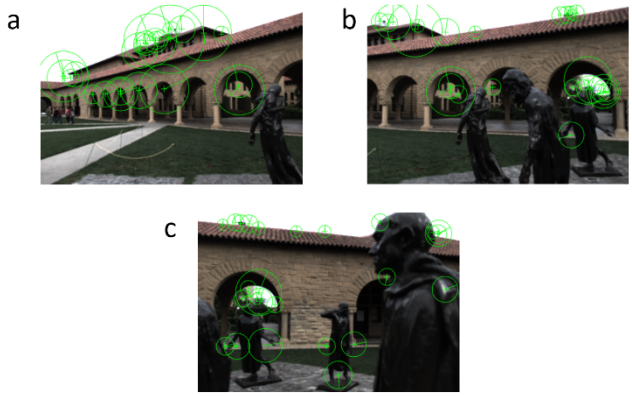

Figure 2: Speeded up robust featured (SURF) extracted from the computed all-in-focus images, which are calculated using the LF data shown in Fig. 1.

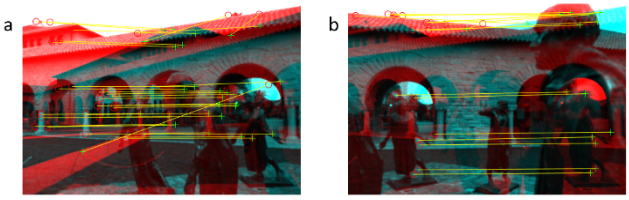

Figure 3: Computation of matching points and their transformation from one image to the other. Panel (a) shows the matching between Figs. 2(a) and (b), whereas panel (b) shows the matching between Figs. 2(b) and (c).

The final step of the panoramic stitchting part is creating a panoramic focal stack by blending the elements of the individual focal stacks. At this point, we have pre-computed the geometric projective transforms from the all-in-focus images. We apply this transformations to the individual members of the focal stacks to create a panoramic focal stack. Note that, in order to perform this operation, elements of the focal stacks corresponding to the three LF images need to have been evaluated at the same focal distances. Otherwise, the focal distance for the resulting panoramic focal stack element is going to be inconsistent.

Before the blending operation, we also apply a simple gain compensation algorithm in order to decrease the possible mismatches in the brigthness among the images. This is done by calculating the average intensities in the overlapping regions between the images and calculating the gain by comparing these quantities. This operation smoothens the transitions between the different images to an acceptable level. The 'compensateGain' function that we have written in MATLAB demonstrates the implementation of the gain compensation. Finally, In Fig. 4, we show the resulting panoramic focal stacks obtained by performing alpha blending in MATLAB with a binary mask. Note that there are still some stitching artifacts in the resulting panoramic images. These can be diminished by blending the images with a multiresolution spline [8] instead of the simple binary mask.

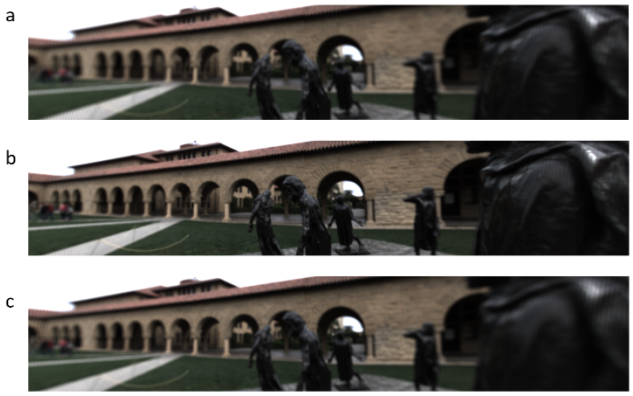

Figure 4: Example images from the panoramic focal stack generated after blending.

3.2. Linear view synthesis

After obtaining the focal stack, the second major part of the project is to obtain different perspective images, which can be intuitively understood as the reverse process of going from Lytro's 14-by-14 raw image data to focal stacks.

An important aspect of this linear view synthesis (LVS) approach is that it does not involve depth estimation as it is based on the assumption that the depth discontinuities in the scene are modest. In other words, with this approach, we assume that three degrees of freedom (3D subset of entries) are enough to interpolate or deduce the 4D LF. Therefore, we can avoid depth estimation. As a trade-off, the reconstruction quality is limited compared to the approaches that involve rigorous depth estimation [9].

This process can be classified into two sub-categories: Perspective generation by a shift-and-add algorithm and Deconvolution with calculated depth-invariant PSFs [9].

3.2.1. Perspective generation by a shift-and-add algorithm

To obtain the desired perspective image, we introduce relative shifts to focal stacks proportionally to its focal depth disparity. Then, we average these shifted focal

stacks to obtain the blurry perspective images (to be de-blurred with depth-invariant PSFs).

The mathematical explanation of this algorithm is as follows: We know that the focal stack is obtained as:

$$b_s(x,y) = \iint_{(u,v)\in Aperture} L(x+su, y+sv)\,du\,dv$$

where u is the horizontal and v is the vertical viewpoint axis. This formula is what we use to obtain the focal stack from the Lytro's raw data in the first place. From the calculated focal stack, we want to obtain the perspective image for the point of view of (u_0,v_0), which we call $L_{u0,v0}(x,y)$. We represent each focal stack by $b_s(x,y)$ and the resulting average image, the perspective image, is

$$\bar{B}(x,y) = \sum_s b_s(x - su_0, y - sv_0)$$

(this is the main shift-and-add formula we used to calculate the perspectives) [9]. Here, s stands for the slope calculated for each focal stack, $s=(d-d_0)/d$, where d is the object depth and d_0 is the distance between the uv and xy planes. (uv plane is the "viewpoint" plane and xy plane is the scene plane. Since uv and xy planes are known by the community, we will not go into the details of these geometrical plane transformations.) Note that the slope values we used in the codes are based on a simple trial and error approach. We picked the best slope values giving us the sharpest images with as less artifacts as possible. The examples of perspective images are provided in the Figure 5.

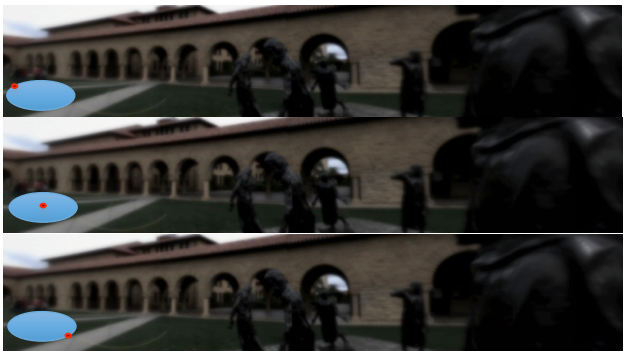

Figure 5: (Top-bottom) The perspective images generated. Inset schematics show the point of views to the scene.

3.2.2. Deconvolution with calculated depth-invariant PSFs

Since we have complete information of what kind of blurring we introduce to the perspective images when we apply the shift-and-add algorithm, we can generate the PSFs and apply Wiener Deconvolution to obtain the sharp final perspective images.

The PSFs we use are based on the work of Levin and Durand [8], and they are as follows:

$$c_{u_0,v_0}(x,y) \approx 1/(\pi^2 A^2 s_{u_0,v_0}(x,y)) + 1/(\pi^2 A^2 s_{u_0,v_0}(-x,-y))$$

where

$$s_{u_0,v_0}(x,y) = y/(A\sin(a\sin(1/A(\cos(\theta)v_0 - \sin(\theta)u_0)) + \theta) - v_0)$$

and $\theta = \angle(x+iy)$ (see Ref. 8 for further proof and derivation). The resulting PSFs for the example perspectives are provided in Figure 6.

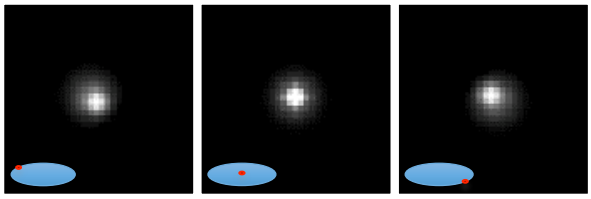

Figure 6: (Left-right) PSFs corresponding to different perspective images. Inset schematics show the point of views to the scene.

We know that

$$\bar{B}(x,y) = c_{u_0,v_0}(x,y) \otimes L_{u_0,v_0}(x,y)$$

Therefore, after obtaining the PSFs for each perspective image, we can now deconvolve these PSFs from the blurry perspective images and obtain the sharp results that we are looking for.

For deconvolution, we use Wiener deconvolution as it is simple and effective for our purpose. The deconvolution formula, as we also implemented in the homework, is:

$$L_{u_0,v_0}(x,y) = F^{-1}\left\{ \frac{\left|F\left\{c_{u_0,v_0}(x,y)\right\}\right|^2}{\left|F\left\{c_{u_0,v_0}(x,y)\right\}\right|^2 + 1/SNR} \cdot \frac{F\left\{\bar{B}(x,y)\right\}}{F\left\{c_{u_0,v_0}(x,y)\right\}} \right\}$$

The resulting sharp perspectives are provided in Figure 7.

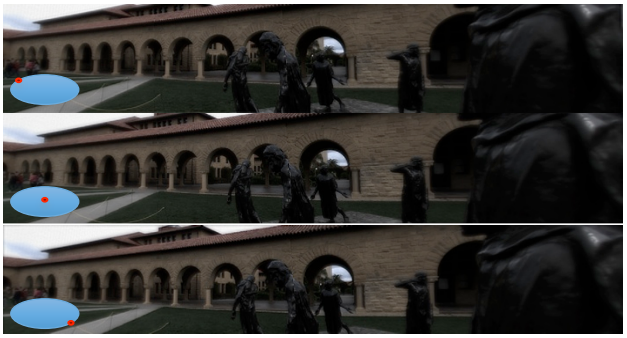

Figure 7: (Top-bottom) The perspective images after Wiener deconvolution (the perspective changes are more visible when looked at the boundaries of the images). Inset schematics show the point of views to the scene.

4. Head tracking and demonstration

For demonstration purposes, we use a head tracking system based on the webcam of our laptop. This system detects the position of the head of a user and shows her/him a corresponding perpective image. Then, the

system tracks the head and updates the displayed image accaording to the current position of the head. This implementation resembles like looking outside through a window. When you move your head, the view that you see through the window is different. For example, when you move your head towards your right, you can see the left part of outside scene more, or when you move your head towards your left, you can see the right part of outside scene more. Similarly, our displayed image is different for the different positions of user's head. When a user moves her/his head towards her/his right in front of the webcam, a perspective image showing more parts of the left of the scene is displayed. Similar corresponding actions are done when the user move her/his head towards the other directions.

We implement this head tracking system in three major steps. Firstly, we detect the face of the user, then we track the position of the user's face, and finally, we choose a perpestive image to display in the computer screen corrsponding to the position of the user's face. We implemented all the parts by using MATLAB [5].

4.1. Face detection

We use vision.CascadeObjectDetector System object to detect the position of a face in image frame taken by our webcam [5]. This detector uses Viola-Jones detection algorithm to detect the face of a user in a given image [10].

In this algorithm, two features are used to detect the face. Firstly, the eye region of a human is often darker than the region of the nose and cheeks. Secondly, the eyes are darker than the nose bridge region. These are Haar-like features.

We take the image of the user, and convert that colorful RGB image to grayscale intensity image. Each pixel has an intensity value associated with its brightness level. Then, the algorithm chooses two adjacent rectangle regions in the image, and sums the values of the pixels in the upper rectangle region as well as sums the values of the pixels in the lower rectangle region. Then, by comparing the difference between these sums, and by performing these calculations in different rectangle regions, the face position is detected. Thus, the algorithm looks at the difference in brightness between adjacent regions in the image. Then, for almost 100% detection rate, it utilizes two abovementioned features.

This algorithm enables extremely fast feature computation by using an efficient feature selection. It does not scale the image itself, but it scales the features. It can also be used for the detection of other types of objects.

However, the algorithm effectively works only on the frontal images of faces, so only when the user directly looks at the webcam. If the user tilts his/her head, then the detection rate decreases. It is also very sensitive to the lighting conditions since it is based on efficient intensity difference on facial parts.

Figure 8 is an example result of our face detection. The detected face region is surrounded by a yellow box.

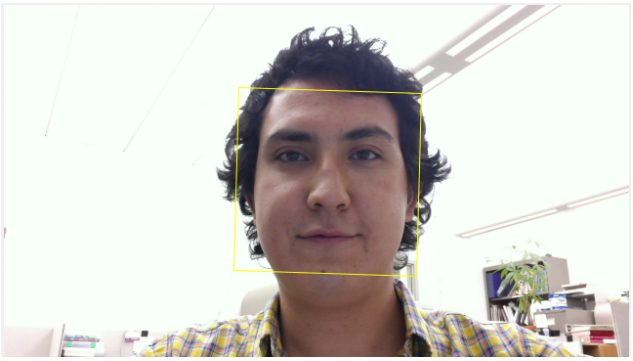

Figure 8: A face detected by our algorithm. The detected face region is surrounded by a yellow box.

4.2. Face tracking

To track the face, we use vision.PointTracker System object [5]. This tracker tracks feature points identified from the detected face by using Kanade-Lucas-Tomasi (KLT) algorithm [11], [12].

In our implementation, detection and tracking of the face are two different steps. Actually, for every image frame taken by the webcam, one can detect the location of the face. In this way, the location of the face would be also found for every frame. However, detection is computationally expensive. Also, as we mentioned before, Viola-Jones detection algorithm is very sensitive of the tilting the head. It works effectively only on the frontal images of the faces. Thus, to overcome these challenges, we detect the face only one, and then we track the detected face by tracking some feature points identified from the face by using KLT algorithm.

Firstly, the algorithm identifies good feature points to track. For this, it uses the method proposed by Shi and Tomasi [13]. In this method, the quality of image features during tracking is measured by feature dissimilarity which quantifies the change of feature appearance between the previous and current image frame. Feature dissimilarity is calculated as the root mean square residue of the feature between the previous and current frame. If the dissimilarity grows too large, that feature should not be used. In this algorithm, both translation model and affine changes model of image motion is used. An affine transformation is the fit between current image frame and a nonconsecutive previous frame. The translation model is more reliable than affine changes when the interframe camera translation is small; however, affine changes are required to determine the dissimilarity between distant frames. If the dissimilarity is too much for the affine compensated image, then that feature is abandoned.

Therefore, using these two different models of image motion gives better results than using one. Briefly, the good features are the ones that optimizes the tracker's accuracy.

After, identifying the features to track in the previous image frame, the algorithm tracks these points by attempting to find the corresponding point in the current image frame. To do this, the tracker estimates the motion of the face.

Figure 9 shows a screenshot from our face tracking step. The detected face region is surrounded by a yellow box while the feature points that are used for tracking are shown by yellow asterisks.

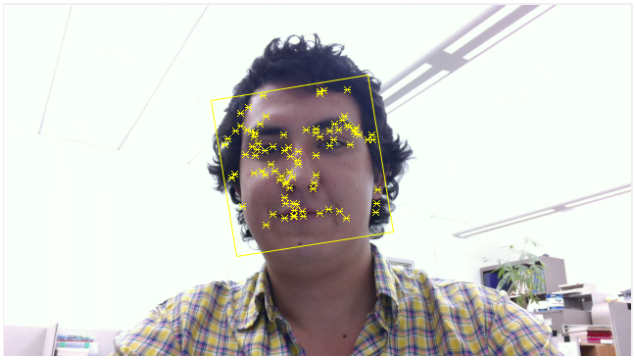

Figure 9: A screenshot taken during face tracking. The detected face region is surrounded by a yellow box. The yellow asterisks show the feature points being tracked.

This point tracker system operates well for tracking objects that do not change shape and that exhibit visual texture. However, it is often used for short-term tracking as part of a long-term tracking framework. When the time progresses, the feature points can be lost due to different reasons. For example, change in lighting condition, out of plane variation, articulated motion, or occlusion can cause the loss of feature points.

When the number of feature points being tracked falls below a threshold by time, the face is stopped being tracked. That's why for tracking a face over a long period of time, the tracking system can need to identify feature points multiple times. So, when the tracking is stopped due to the lack of enough feature points, the system re-detects the face, then identifies a new set of feature points, and then track these points.

These face detection and face tracking steps are continuously implemented until the user or a pre-determined timer stop the system.

4.3. Displaying a perspective image

Once we detect the location of the user's face, and track it, we display a perspective image to the user in the computer screen. This perspective image is chosen from 14-by-14 panoramic perspective images. Each image corresponds to a head location on the image frame taken by the webcam. We define the head location as the center of the box surrounding the user's face. After finding this center, we display the associated perspective panoramic image to the user. When the user moves his/her head, the displayed image is replaced by the new one corresponding to the new center of the head of the user.

The image frame taken by the webcam is divided into 14-by-14 rectangular regions. Each region is associated with a perspective image. The regions at the right of the webcam image frame are associated with the perspective panoramic images showing the left of the scene more. Similarly, the regions at the left of the webcam image frame are associated with the perspective images showing the right of the scene more. By the same token, the regions on the upper part of the webcam image frame are associated with the perspective images showing the bottom part of the scene more, and vice versa. This association is chosen in this way because we want to make the user's experience with the displayed image as it would be if he/she was looking at the displayed image through a window. When we look outside of a window, if we move our head to our right, we can start to see more of the left part of the outside scene while we stop to see the details on the rightmost part, and vice versa. Similar trend is also true for moving our head up or down. Our demonstration works in the same logic. In other words, if the user moves his/her head to his/her right, the displayed image will show more of the left parts and less of the right parts of the scene, and vice versa. Similar trend is also occurs when the user moves his/her head to up or down.

Figure 10 shows an example of finding the rectangle region that includes the center of the yellow box surrounding the face, and then displaying the perspective panoramic image associated with that rectangle region.

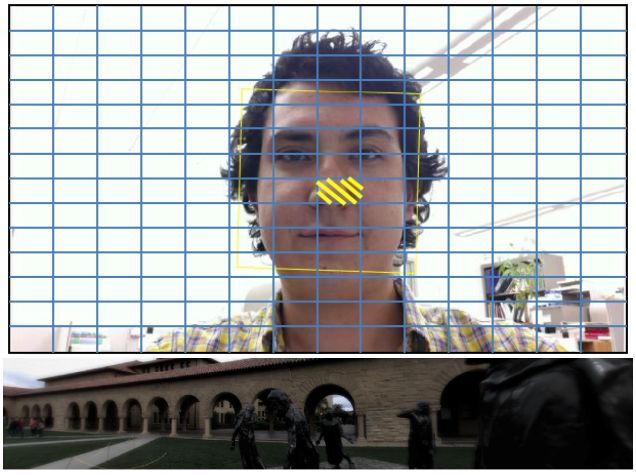

Figure 10: (Top) An image frame taken by our webcam is divided into 14-by-14 rectangular regions. The yellow shaded region is the region including the center of the box surrounding the face. (Bottom) The displayed perspective image associated with the shaded region.

5. Conclusions

As a summary, in this project, we implemented a highly effective and computationally reasonable way of panoramic reconstruction of light-field imaging. The project consisted of three major parts: panoramic stitching, linear view synthesis and head-tracking for demonstration purposes. We took our own light-field images with the borrowed Lytro camera and implemented everything on the raw images. Even though the panorama methodology we utilized works the best for Lambertian scenes and it introduces artifacts at the occlusion boundaries and anisotropic reflections/refractions, it performed well for our scenery. This can also be explained by the fact that the limiting factor for the quality of our images was the not-so-impressive resolution of Lytro camera. Therefore, this panorama technique can be considered optimal for this camera, which is almost an industry standard of this field.

Another major limiting factor for our implementation was the limited computational resources. This limitation made it impossible for us to stitch more than three images on our computers even though our algorithms are perfectly suited for higher number of stitching per run.

Each one of the aforementioned major parts of this project consisted of multiple sub-parts that were based on different reference sources. Thus, we had to conduct an extensive literature review on the light-field panorama field. Another one of the biggest challenges of this project was to implement different parts of the project in a coherent and modular way so that the final code works seamlessly. This helped us to learn the effective use of MATLAB in a collaborative way. We had to work as a good team to be able to finish the project in time with the optional part of head-tracking demo. Overall, we did not only improve our technical backgrounds in the field of light-field imaging and head-tracking, but we also had the chance to experience a very effective collaboration.

References

[1] M. Brown and D. G. Lowe, "Automatic Panoramic Image Stitching using Invariant Features," *Int. J. Comput. Vis.*, vol. 74, no. 1, pp. 59–73, Dec. 2006.

[2] C. Birklbauer, S. Opelt, and O. Bimber, "Rendering Gigaray Light Fields," *Comput. Graph. Forum*, vol. 32, no. 2pt4, pp. 469–478, May 2013.

[3] C. Birklbauer and O. Bimber, "Panorama light-field imaging," *Comput. Graph. Forum*, vol. 33, no. 2, pp. 43–52, May 2014.

[4] D. G. Dansereau, O. Pizarro, and S. B. Williams, "Decoding, Calibration and Rectification for Lenselet-Based Plenoptic Cameras," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2013, pp. 1027–1034.

[5] "MATLAB R2015b." The Mathworks Inc., Natick, Massachusetts, 2015.

[6] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-Up Robust Features (SURF)," *Comput. Vis. Image Underst.*, vol. 110, no. 3, pp. 346–359, Jun. 2008.

[7] M. Muja and D. G. Lowe, "Fast Matching of Binary Features," in *2012 Ninth Conference on Computer and Robot Vision*, 2012, pp. 404–410.

[8] P. J. Burt and E. H. Adelson, "A multiresolution spline with application to image mosaics," *ACM Trans. Graph.*, vol. 2, no. 4, pp. 217–236, Oct. 1983.

[9] A. Levin and F. Durand, "Linear view synthesis using a dimensionality gap light field prior," in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2010, pp. 1831–1838.

[10] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, 2001, vol. 1, pp. I–511–I–518.

[11] B. D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision," *7th international joint conference on Artificial intelligence*, vol. 2. pp. 674–679, 1981.

[12] C. Tomasi and T. Kanade, "Detection and Tracking of Point Features Technical Report CMU-CS-91-132," *Image Rochester NY*, vol. 91, no. April, pp. 1–22, 1991.

[13] Jianbo Shi and C. Tomasi, "Good features to track," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition CVPR-94*, 1994, pp. 593–600.