SocialSync: Sub-Frame Synchronization in a Smartphone Camera Network

Richard Latimer(✉), Jason Holloway, Ashok Veeraraghavan, and Ashutosh Sabharwal

Rice University, Houston, TX, USA
rplatimer@gmail.com

Abstract. SocialSync is a sub-frame synchronization protocol for capturing images simultaneously using a smartphone camera network. By synchronizing image captures to within a frame period, multiple smartphone cameras, which are often in use in social settings, can be used for a variety of applications including light field capture, depth estimation, and free viewpoint television. Currently, smartphone camera networks are limited to capturing static scenes due to motion artifacts caused by frame misalignment. Because frame misalignment in smartphones camera networks is caused by variability in the camera system, we characterize frame capture on mobile devices by analyzing the statistics of camera setup latency and frame delivery within an Android app. Next, we develop the SocialSync protocol to achieve sub-frame synchronization between devices by estimating frame capture timestamps to within millisecond accuracy. Finally, we demonstrate the effectiveness of SocialSync on mobile devices by reducing motion-induced artifacts when recovering the light field.

Keywords: Multiple viewpoints · Camera array · Camera network · Synchronization · Smartphone · Mobile device

1 Introduction

Smartphones, and by extension smartphone cameras, have been predicted to approach 1 billion units in annual sales by the end of 2014 [9]. The rapid rise of readily available cameras has drastically increased the number of pictures that are taken each day, while the advent of social media and image sharing websites (e.g., Facebook, Flickr, and Picasa) allows for easier image dissemination than ever before.

While sharing images has become a common activity in social interactions – Facebook sees an average of 350 million images uploaded to its servers daily [7] – capturing images remains an individual activity. Despite collectively viewing, sharing, and commenting on images, photographers remain as islands; each taking pictures independently and ignoring the resources of other nearby smartphone cameras. Our goal is to synchronize image captures using mobile devices during *social image acquisition*, whereby users can collaboratively capture images

Electronic supplementary material The online version of this chapter (doi:10.1007/978-3-319-16181-5_43) contains supplementary material, which is available to authorized users.

(a) Present day: Individual imaging for social sharing

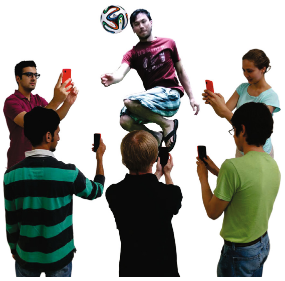

(b) Future: Illustration of SocialSync for social imaging

Fig. 1. (a) While the flood of mobile devices has become ubiquitous during major historical events, as seen during the election of Pope Francis, each user effectively operates independently. Image credit: Michael Sohn Associated Press; (b) Synchronizing the image capture times across mobile phones, a group of people working together will be able to capture rich information of an event, even with dynamic motion present in the scene.

which, when taken together, are of greater value than the collection of individual photographs.

1.1 Why Social Image Acquisition?

It is common to see many smartphones hoisted aloft capturing images at public events. For example, Fig. 1(a) shows St. Peter's square in the Vatican as the election of Pope Francis was announced. Mobile devices are ubiquitous throughout the square, as people take pictures and video. The sheer number of cameras at such events presents an opportunity to recover rich data about the scene, far exceeding what is available with a single camera. Applications include capturing light fields for post-capture processing, free viewpoint video, and computing depth maps for scene reconstruction and modeling.

1.2 Problem Definition

Efforts such as Photo Tourism from Snavely et al. [20] (later commercialized by Microsoft into PhotoSynth[1]) and its extension by Agarwal et al. [1] use images taken from many cameras to reconstruct a 3D model of a target. A reasonable facsimile of public objects and scenes can be rendered by scouring image aggregation and sharing sites, such as Flickr, and by using geometric constraints provided from the disparate viewpoints. Users can zoom into an object, fly around buildings, and remotely tour faraway locales. The limitation is that the scene must be static, since the images have been taken at different times. Such an

[1] www.photosynth.net

approach works well with buildings, natural monuments, and landscapes, but not so well for fast moving scenes, such as sports venues or concerts. Capturing a dynamic scene requires that cameras be synchronized to an accuracy that is a fraction of the duration of a frame.

Synchronizing consumer cameras is a challenging task, even more so for smartphone cameras. Mobile phones do not accept external hardware trigger signals and software triggers do not offer tight enough bounds to capture images simultaneously. In order for picture taking to become a communal experience, as illustrated in Fig. 1(b), a protocol for synchronizing smartphone cameras must overcome the variability caused by the camera system when triggering frame capture and delivery.

1.3 Contributions

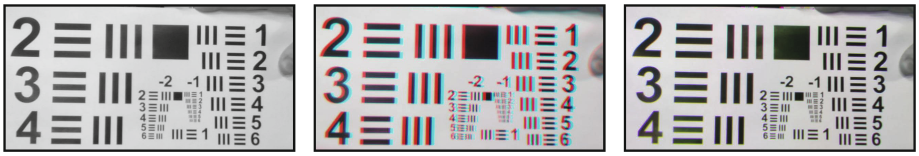
We demonstrate a protocol and highlight the necessity for highly accurate synchronization of frames across mobile devices both for indoor and outdoor dynamic scenes. To address the temporal challenges present when using mobile devices for single snapshot social image acquisition, we use the HTC One (M7) and Nexus 5 to:

1. Characterize the variables associated with relative latency that cause temporal differences between frames captured from different mobile devices. We identify the setup latency of the camera service as the main cause of variability when synchronizing frame capture.
2. Develop SocialSync, a sub-frame synchronization protocol, using additional measurements of frame rate and frame delivery to estimate the timestamp of a frame captured with millisecond accuracy. Compared to a naive synchronization implementation, where frames are aligned to the duration of a frame, our implementation achieves sub-frame alignment by requiring a duration of time to achieve synchronization before the frame capture request.
3. Demonstrate our ability to reduce motion artifacts using SocialSync when recovering the light field from a smartphone camera network. Compared to a naive synchronization implementation, SocialSync considerably reduces visible artifacts in the fused light field.

2 Background

2.1 Related Work

Multiple camera image capture: Many imaging tasks can be performed easily using multiple cameras, whether the cameras are arranged in a calibrated array or arranged randomly. For example, camera arrays can be used to capture the light field of a scene [10,23,25,26], record high speed video [18,19,24], and improve image resolution [19], while distributed cameras have been used to construct virtual cities from online photo repositories [1,20] and synthesize 3



(a) Reference image (b) Unsynchronized phones (c) Synchronized phones

Fig. 2. Motion artifacts manifest when aligning unsynchronized frame sequences. (a) A grayscale image of a planar resolution chart moving to the right taken from Fig. 3. Grayscale images from three (b) unsynchronized and (c) synchronized cameras are warped using homographies to the true depth of the moving resolution chart. The aligned images are shown as an RGB image where misaligned edges present as color artifacts. Notice that without synchronization (b) the bars in the resolution chart are misaligned by 10 pixels while the synchronized images have errors of at most 1 pixel.

dimensional models of buildings [6]. State-of-the-art snapshot light-field acquisition methods which may be used in smartphones require specialized hardware [10,14,23]. Furthermore, mask-based systems [14] reduce light throughput while camera arrays such as the PiCam [23] require hardware synchronization to ensure each element of the array captures images simultaneously. Fig. 2 highlights the need for synchronization in dynamic scenes. A planar resolution chart translates to the right in front of unsynchronized and synchronized cameras (Fig. 2(b) and Fig. 2(c) respectively). Aligning images using homographies shows that the unsynchronized images have motion artifacts of approximately 10 pixels while the synchronized cameras have error less than 1 pixel.

Using multiple cameras to capture a scene enables many benefits over single viewpoint imaging. Applications include:

Light field: Light field cameras, such as Lytro [8] and Raytrix[2], can be used for digital refocusing, but sacrifice spatial resolution. Compared to single camera techniques, various works have demonstrated light field recovery using camera arrays [5,26].

Free Viewpoint Television: Free-viewpoint television uses multiple cameras for viewing a 3D scene by changing viewpoints [21]. In addition, an array of smartphones could be used for a variety of special effects such as bullet time [25].

3D and Depth: Camera arrays are also useful when recovering 3D and depth from a scene [17,22].

2.2 Android Camera Library

The android camera library provides access to camera functions, such as locking exposure, focus, zoom, and capturing images or video on demand. By abstracting the camera utilities for the developer, the camera library hides the details of binding to the Android camera service and operating the sensor hardware. An application activates the camera by calling `startPreview()` to begin streaming a

[2] http://www.raytrix.de/

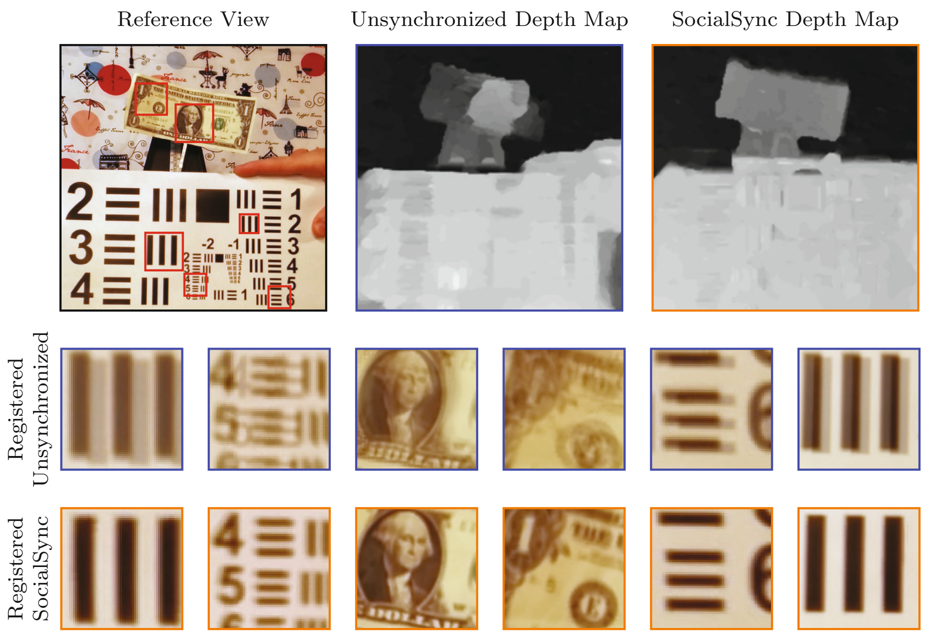


Fig. 3. Computing depth maps to register 4 cameras to a reference view. Depth estimation with unsynchronized images (top center) is challenging as the images are never truly aligned (see Fig. 2(b)). Depth estimation is more accurate when using our Social-Sync protocol (top right). Outset show the average of the 4 registered images using timestamps to synchronize (middle row) and SocialSync (bottom row).

sequence of image frames. A developer can specify a callback function to trigger when a preview frame is available, either for processing or for saving to disk. Both the Nexus 5 and HTC One support a variety of preview sizes. In our setup we set both devices to capture 1920×1080 pixel images.

2.3 Time Synchronization Protocols

Due to manufacturing differences, each smartphone's system clock will drift at slightly different rates, creating misalignment between recorded timestamps. An important and well studied problem, clock synchronization achieves a consistent global time across all devices in the network. Our solution uses the Network Time Protocol (NTP) [15,16] to perform clock synchronization among devices. The maximum clock synchronization error is bounded by the round-trip time of the network. Because our WiFi access point is capable of round-trip times (RTTs) of less than 2 ms to our time server, NTP permits clocks synchronization to be within 1 ms.

2.4 Latency

A camera network's response to a request for an image capture is limited by two sources of latency:
Network Latency: Events sent between devices incur an end-to-end network latency. Our measurements demonstrated two devices sharing the same WiFi access point had a mean round trip latency of around 3 ms as well as an outlier RTT of 75 ms.
Camera I/O Latency: There is a non-deterministic latency from the time the software issues a command to take a picture and the time the hardware captures a frame due to the variables in mobile OS resource management. In our measurements, we found that the average camera I/O latency is specific to particular device models. Fig. 3 shows the necessity of compensating for I/O latency when estimating depth from independent smartphone cameras with synchronized clocks. Notice that the depth map for the unsynchronized cameras contains errors for the dynamic scene elements while the SocialSync cameras give an accurate depth map.

3 Camera Characterization

We reduce the problem of synchronizing frame capture to that of the I/O camera latency associated with triggering frame capture and delivery. Our implementation uses network clock synchronization to devices clocks and requires that requests for frame capture reach each mobile device before the capture event.

3.1 Camera Timestamps

To characterize the latency through the system, we define the following:

- **Frame Capture** $T_C(i)$**:** The time image exposure ends for the ith frame.
- **Frame Delivery** $T_D(i)$**:** The time the application receives the ith frame.
- **Camera Setup Latency** $T_C(0)$**:** The setup time to capture the 0th frame.
- **Frame Rate** $(T_C(i)-T_C(i\text{-}1))^{-1}$**:** The rate of capturing consecutive frames.
- **Frame Delay** $T_D(i) - T_C(i)$**:** System delay when delivering the ith frame.

We use the mobile system timestamp on the preview callback to obtain $T_D(i)$, since preview frames in Android do not contain EXIF millisecond meta data timestamps. As the capture timestamp is not accessible through the mobile operating system, we build a characterization setup to measure $T_C(i)$.

3.2 Camera Characterization Setup

We capture the frame latency with an experimental setup in order to recover the frame capture timestamps precisely. For further details regarding our smartphone app implementation and rolling shutter measurements, we direct the interested reader to our supplementary material [12].

Characterization Smartphone App: The camera object runs on a dedicated background thread to prevent resource conflicts with the foreground activity. Auto exposure and white balance are locked, putting the camera system in a mode that enables rapid capture. To streamline memory allocation, the application pre-allocates preview frames into a circular buffer queue. The focus of each camera is fixed at infinity.

Image Timestamp from Visible Clock: To obtain a timestamp of a frame capture $T_C(i)$, we use a camera scene that includes a visible clock. For accuracy, we built an 8×8 array of LEDs, sequentially triggered at precise time intervals by a Raspberry Pi (RPi). The RPi sequentially lights each column of LEDs on the array for 1 ms. When the camera takes an image of the LED clock, the position of the illuminated LEDs on the image serve as a timestamp for the image. Because rows of pixels are read out at different times due to the rolling shutter, $T_C(i)$ indicates the time when reading the 1st row from the image sensor. Further details regarding our measurement setup for calculating rolling shutter speed and $T_C(i)$ are described in [12].

Timing Precision of the Visible Clock: The RPi acts as a global reference clock. It is synchronized via a wired GPS clock to minimize clock drift. `loopstats` in the NTP protocol reports the resulting clock jitter of the RPi as 5μ. The pre-synchronization clock drifts for the smartphones were small enough for characterization purposes, drifting less than 60μ after 1 second of elapsed time. The smartphones wirelessly synchronizes their clocks with the RPi, repeating synchronization attempts until the RTT is less than 2 ms and clock error is less than 1 ms.

3.3 Characterization Measurements

We characterize the camera setup latency, frame rate, and delay when delivering preview frames for a Nexus 5 and HTC One.

Camera Setup Latency $T_C(0)$: On Android, before capturing an image, the camera must first be activated by starting the preview image sequence. The variability in setting up the camera service, sensor, and preview sequence limits the ability to synchronize frames. By measuring the latency from launching the preview sequence to the capture of the first frame $T_C(0)$, we see launching the camera preview sequence at the same time is insufficient to achieve accurate synchronization because of the randomness in the latency. The camera setup time for a Nexus 5 has a sample mean of $\mu = 283.3$ ms and may deviate with a standard deviation of $\sigma = 9.4$ ms. The distribution shown in Fig. 4 (left) is representative of the variability in setting up image capture on a mobile device.[3]

Frame Rate $(T_C(i) - T_C(i\text{-}1))^{-1}$: Although the capture time of a frame is stochastic, the time between frames is deterministic. By knowing the time interval between image capture timestamps, all frame capture timestamps can be determined as long as one timestamp is known. The difference between subsequent capture timestamps is inversely proportional to the frame rate of the image

[3] $T_C(0)$ will vary between devices.

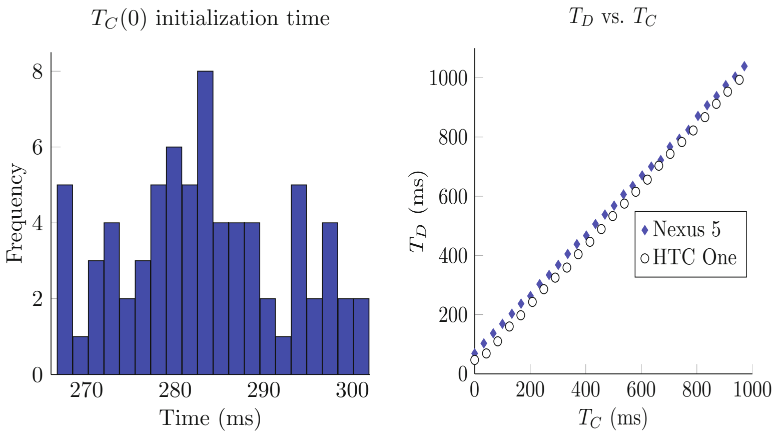


Fig. 4. (Left) Camera Setup: Android phones require an activated preview image sequence prior to capturing a photo. Therefore, frame synchronization between devices is based on the offset between setting up the camera and capture the first frame $T_C(0)$. We show that for Nexus 5 camera, simultaneous launches of the camera have a setup time with a mean of $\mu = 283$ ms and a standard deviation of $\sigma = 9.4$ ms; (Right) The delivery time T_D of a frame to an application is highly correlated with its capture time T_C. The relationship between delivery time and capture time provides the basis for estimating $T_C(0)$.

sequence. Because Android devices provide various ranges for setting frame rates, in our setup we locked the frame rate to a valid range supported by the Android devices and then measured the frame rate using our LED clock. Upon locking the auto exposure, the frame rate became constant at $f = 29.8497 \pm 0.0001$ fps for a Nexus 5 and $f = 24.1513 \pm 0.0002$ fps for an HTC One.

Frame Delay $T_D(i) - T_C(i)$: For a fixed frame rate image sequence, $T_C(i)$ is highly correlated with $T_D(i)$, the time for delivering a frame to the application as shown in Fig. 4(b). By measuring latency between capturing a frame and delivering a frame, we will be able to build a model for estimating $T_C(i)$. The frame delay can be represented as a stationary stochastic variable with a normal distribution N_F whose mean $\mu_F = 36.83$ ms and standard deviation $\sigma_F = 4.68$ ms for an HTC One and $\mu_F = 66.67$ ms and $\sigma_F = 4.48$ ms for a Nexus 5.[4] The large difference between the two data sets is because the Nexus 5 passes two frames before delivering the captured frame, while the HTC One delivers the captured frame after only one frame has passed.

4 SocialSync Protocol

SocialSync achieves highly accurate synchronization across a diverse range of Android devices in a network by (1) estimating capture timestamps based on

[4] Assumption of normal distribution is valid because $\sigma_F \ll \mu_F$.

the delivery timestamps of previously delivered frames and (2) using repeated attempts at launching the preview image sequence until a set of frames is obtained for which the computed timestamps align (frames are in sync).[5]

4.1 Capture Timestamp Estimation

In single camera tasks, frames recorded by the camera are sequential and evenly spaced, specified by the frame rate. In multi-camera tasks, knowing the exact capture timestamp is required to align frames from different cameras, as the relative position of a frame from one camera is unknown with respect to the frame from a second camera. If the camera frame rates are known, then the calibration task is simplified by providing a common time origin and measuring the offset to each camera's first frame. Therefore, the precision in estimating the capture timestamp of a frame is based strictly on the estimation of $T_C(0)$, the setup capture timestamp.

For a fixed frame rate f, the time the i^{th} frame is captured is related to the camera setup latency $T_C(0)$ according to

$$T_C(i) = T_C(0) + (1/f) \cdot i. \tag{1}$$

Let T_N be a random variable representing the frame delay following the normal distribution N_F. $T_C(i) = T_D(i) - T_N(i)$, where $T_D(i)$ provides a sample for estimating $T_C(0)$. Therefore, $T_C(0)$ can be expressed as Gaussian random variable with a distribution N_F such that

$$T_C(0) \approx T_D(i) - 1/f \cdot i - T_N(i). \tag{2}$$

Camera setup latency $T_C(0)$ is estimated by taking multiple measurements of $T_D(i)$, determining the distribution of the frame delay, and calculating the average. The timestamp of $T_C(0)$ is the center of the Gaussian frame delay distribution. A standard error calculation of $T_C(0)$ provides a method for estimating the sample mean within a desired confidence interval. Therefore, to obtain a 95% confidence interval of less than δ ms with the number of samples frames n is

$$\frac{\sigma_F}{\sqrt{n}} \cdot 1.96 \leq \delta. \tag{3}$$

Therefore, an estimate of $T_C(0)$ at a 95% confidence interval, and all subsequent capture timestamps, to within 2 ms requires the delivery of at least 22 preview frames and to within 1 ms requires the delivery of at least 85 frames for both an HTC One and Nexus 5.

4.2 Frame Synchronization Upper Bound

Camera I/O latency $\Delta T_C(i)$ is the delay between a request for a frame capture and the execution of the event at $T_C(i)$. Because each frame's capture timestamp

[5] In the protocol, we assume a global reference clock, such as one obtained using NTP.

can be estimated precisely using the results of Sec. 4.1, a mobile app can deliver the most recently captured frame $T_C(i)$ for each request. Because a periodic sequence of images has a fixed frame rate f, a captured frame $T_C(i)$ closest to the time of an arbitrary request will result in $\Delta T_C(i)$ being uniformly distributed between 0 and $\tau = 1/f$ seconds. Therefore, the upper bound synchronization error between frames from multiple devices is the frame sequence with the longest interval τ, i.e. the inverse of the lowest frame rate.

4.3 Obtaining Sub-Frame Synchronization

By estimating capture timestamps, the SocialSync protocol achieves sub-frame image capture through launching the smartphone preview image sequence stream repeatedly until frame sequences are aligned[6]. Under the hood, synchronization is achieved by estimating capture timestamps to successfully predict the image sequence frame setup time, thereby capturing a frame at a desired request time within a specified tolerance.

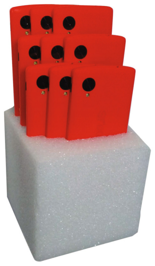
Suppose a user requires that the camera I/O latency $\Delta T_C(i)$ for frame capture is within the range $(0, t)$, where $t \leq \tau$. The probability the phone will fail (p_f) to capture a frame at a time within the range $(0, t)$ is $p_f = 1 - t/\tau$.

Repeated attempts at starting the image sequence would improve the odds of starting within the desired synchronization range. Using our capture timestamp estimation technique described in Sec. 4.1, we can determine whether an image sequence is in the desired synchronization range. By successfully estimating whether a given image sequence will succeed or fail, sub-frame synchronization is based on following equations:
Single Camera Sync Probability: The probability that a single phone will start the continuous image sequence in the range $(0, t)$ after k attempts is $P_k = 1 - (p_f)^k$.
Multiple Cameras Sync Probability: The probability that n phones will start the continuous image sequence in the range $(0, t)$ after k attempts is $(P_k)^n$.
Expected Number of Sync Cameras: The expected number of phones to start the continuous image sequence in the range $(0, t)$ after k attempts of n phones is nP_k.

5 Evaluation

To demonstrate the advantages of the SocialSync sub-frame synchronization protocol, we capture images of dynamic scenes and demonstrate improvements in recovering the light field by reducing motion artifacts. To reduce errors not associated with synchronization misalignment, we constrain our evaluation to a structured camera array consisting of Nexus 5 devices shown in Fig. 5. The cameras are calibrated using the Caltech calibration toolbox [2] and further refined using bundle adjustment [13].

[6] With a large number of smartphones a subset of synchronized cameras could be used without the need to restart the preview streams.



Maximum difference in capture times for synchronized smartphone cameras

	NaiveSync	SocialSync
4 Cameras	23 ms	5 ms
8 Cameras	35 ms	6 ms

Fig. 5. Camera array setup used for evaluation. (Left) Up to 9 cameras are placed in an rigid array to minimize errors not associated with scene motion. (Right) Camera synchronization timings measured in evaluation for NaiveSync (i.e. timestamp comparison) and SocialSync. SocialSync offers tighter synchronization than NaiveSync.

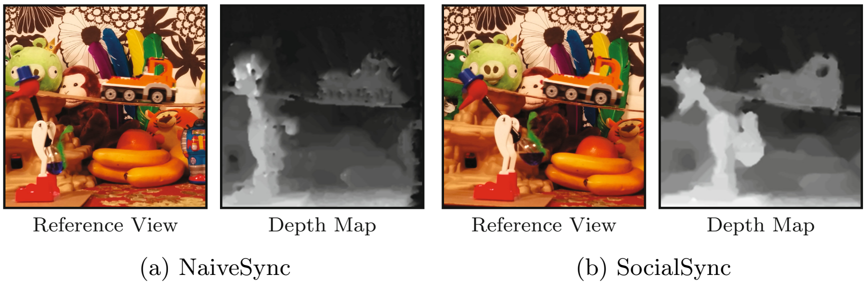


Fig. 6. Eight cameras capture a dynamic indoor scene. A drinking bird provides angular motion while a toy truck translates across the scene. (a) Depth estimates of scene using the NaiveSync protocol exhibit artifacts for dynamic scene elements. (b) SocialSync achieves accurate depth map recovery including dynamic regions such as the truck window and drinking bird.

5.1 Recovering the Light Field

We use the SocialSync protocol to synchronize cameras within 6 milliseconds (shown in Fig. 5). We compare our results against a naive frame synchronization implementation (called NaiveSync), which only saves the frame with the closest delivery timestamp. We collect indoor and outdoor datasets using 8- and 4-camera arrays respectively. Depth maps recovered from the disparate views allow for post-capture refocusing. Point correspondences are computed using a plane sweep algorithm and a window-based normalized cross correlation cost function. We use the graph cuts implementation of [3,4,11] to impose a smoothness penalty between neighboring pixels and recover our depth estimates.

Indoor Scene with an 8-Camera Array: In the scene shown in Fig. 6, dynamic scene elements (the angular motion of the drinking bird and translation motion of the truck) require image synchronization to compute accurate depth maps. Using NaiveSync, which saves the frames with the closest delivery

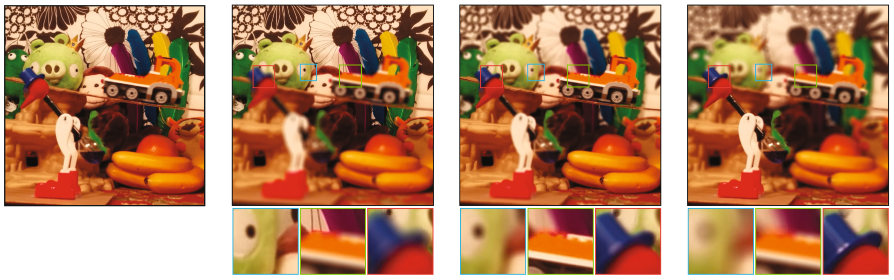

(a) All in focus (b) Far Focus (c) Mid Focus (d) Near Focus

Fig. 7. Post-capture refocusing using the accurate depth map of Fig. 6(b) captured using SocialSync. (a) The captured image is refocused in the (b) far, (c) middle, and (d) near ground of the scene post-capture. Please view digitally to see details.

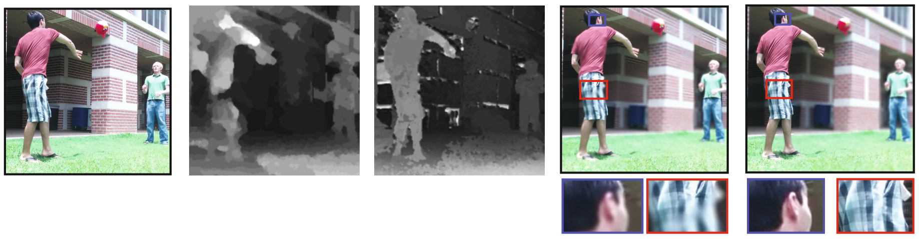

(a) Reference (b) Depth from NaiveSync (c) Depth from SocialSync (d) Refocus with NaiveSync (e) Refocus with SocialSync

Fig. 8. SocialSync provides advantages in dynamic outdoor scenes. Seven phones are divided into two groups of 4 phones with one overlapping phone. One group uses our SocialSync protocol and the other group uses with NaiveSync. (a) Reference view of two people tossing a stuffed toy. (b) The depth recovered using NaiveSync has motion artifacts not present when (c) computing the depth using SocialSync. (d) Proper post-capture refocusing cannot be achieved with NaiveSync. Notice that the thrower's face and shorts are incorrectly blurred when focusing on the thrower's body. (e) SocialSync allows for accurate blurring for the thrower's entire body.

timestamps, results in synchronization of 35 ms while our SocialSync protocol reduces the error to 6 ms. The two data sets are captured independently. Note that the depth map recovered when using SocialSync, Fig. 6(b), is free of the artifacts present when using NaiveSync, Fig. 6(a). In particular, dynamic scene elements such as the drinking bird and the truck's wheels and window remain blurred when using NaiveSync.

The accurate depth map provided by using SocialSync in Fig. 6(b) allows users greater artistic license when viewing captured images. Fig. 7 shows the indoor scene refocused post-capture on the near, middle, and far planes.

Outdoor Scene with a 4-Camera Array: Figure 8 shows a scene taken outdoors of two people throwing a toy bird. Seven cameras captured the scene with one chosen as a reference camera. Four cameras were synchronized using SocialSync (including the reference) while the remaining three are unsynchronized with respect to each other and the reference. The four SocialSync cameras are synchronized to within 5 ms while the four NaiveSync cameras have a 23 ms spread. Note that the depth map recovered from the SocialSync cameras (Fig. 8(c)) accurately captures the depth of the scene while the depth computed using the NaiveSync cameras (Fig. 8(b)) has many artifacts. Fig. 8(d) highlights the inability to refocus on the thrower properly when using the NaiveSync depth map, while refocusing using SocialSync (Fig. 8(e)) has no such limitation.

6 Conclusions

Our work highlights and addresses the sub-frame synchronization challenge when using smartphones for multi-viewpoint light field recovery. Without sub-frame synchronization between mobile devices, light field acquisition is limited to static scenes due to motion artifacts caused by frame misalignment. As the first step towards multi-viewpoint image capture of dynamic scenes using smartphone camera networks, we characterized the camera setup, frame rate, and frame delay on an HTC One and Nexus 5. Next, we introduced SocialSync, a sub-frame synchronization protocol, based on an estimation of frame capture timestamps. Finally, we evaluated the benefit of using SocialSync by comparing it to the best existing smartphone camera synchronization method and demonstrating improvements in depth map estimation and digital refocusing.

As a limitation, sub-frame synchronization of smartphone cameras is only effective for capturing a single snapshot or a few frames, due to variability in frame rates caused by clock drift and manufacturing quality. Furthermore, due to the stochastic nature of synchronization, increasing the number of devices requires more synchronization attempts. Therefore, as future work to address scalability issues with large social events, we would explore methods for grouping subsets of smartphones, which would be naturally synchronized within the group.

Acknowledgments. We would like to thank the LF4CV reviewers and Robert LiKamWa for useful discussions regarding this work. The authors were partially supported by NSF Grants CNS 1012921, CNS 1161596, IIS 1116718, and CCF 1117939.

References

1. Agarwal, S., Furukawa, Y., Snavely, N., Simon, I., Curless, B., Seitz, S.M., Szeliski, R.: Building rome in a day. Communications of the ACM **54**(10), 105–112 (2011)
2. Bouguet, J.Y.: Camera calibration toolbox for matlab (2008)
3. Boykov, Y., Kolmogorov, V.: An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. IEEE Transactions on Pattern Analysis and Machine Intelligence **26**(9), 1124–1137 (2004)

4. Boykov, Y., Veksler, O., Zabih, R.: Fast approximate energy minimization via graph cuts. IEEE Transactions on Pattern Analysis and Machine Intelligence **23**(11), 1222–1239 (2001)
5. Buehler, C., Bosse, M., McMillan, L., Gortler, S., Cohen, M.: Unstructured lumigraph rendering. In: Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques, pp. 425–432. ACM (2001)
6. Debevec, P.E., Taylor, C.J., Malik, J.: Modeling and rendering architecture from photographs: A hybrid geometry-and image-based approach. In: Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques, pp. 11–20. ACM (1996)
7. Facebook, Ericsson, Qualcomm: A focus on efficiency. Tech. rep. (September 2013) http://internet.org white paper
8. Georgiev, T., Yu, Z., Lumsdaine, A., Goma, S.: Lytro camera technology: theory, algorithms, performance analysis. In: IS&T/SPIE Electronic Imaging, pp. 86671J–86671J. International Society for Optics and Photonics (2013)
9. Gupta, A., Cozza, R., Lu, C.: Market share analysis: Mobile phones, worldwide, 4q13 and 2013. Tech. rep., Gartner, Inc. (February 2014) (white paper)
10. Heptagon Advanced Micro Optics. http://www.hptg.com/products/imaging (2014), (Online accessed March 31, 2014)
11. Kolmogorov, V., Zabin, R.: What energy functions can be minimized via graph cuts? IEEE Transactions on Pattern Analysis and Machine Intelligence **26**(2), 147–159 (2004)
12. Latimer, R., Holloway, J., Veeraraghavan, A., Sabharwal, A.: Supplementary material for SocialSync: Sub-frame synchronization in a smartphone camera network (2014), Computer Vision-ECCV 2014. LF4CV submission. Supplied as additional material
13. Lourakis, M.A., Argyros, A.: SBA: A software package for generic sparse bundle adjustment. ACM Trans. Math. Software **36**(1), 1–30 (2009)
14. Marwah, K., Wetzstein, G., Bando, Y., Raskar, R.: Compressive light field photography using overcomplete dictionaries and optimized projections. ACM Transactions on Graphics (TOG) **32**(4), 46 (2013)
15. Mills, D.L.: Network time protocol (ntp). Network (1985)
16. Mills, D.L.: Computer Time Synchronization: The Network Time Protocol on Earth and in Space, 2 edn. CRC Press (2010)
17. Naemura, T., Tago, J., Harashima, H.: Real-time video-based modeling and rendering of 3d scenes. IEEE Computer Graphics and Applications **22**(2), 66–73 (2002)
18. Nayar, S., Ben-Ezra, M.: Motion-based motion deblurring. IEEE Transactions on Pattern Analysis and Machine Intelligence **26**(6), 689–698 (2004)
19. Shechtman, E., Caspi, Y., Irani, M.: Space-time super-resolution. IEEE Transactions on Pattern Analysis and Machine Intelligence **27**(4), 531–545 (2005)
20. Snavely, N., Seitz, S.M., Szeliski, R.: Photo tourism: Exploring photo collections in 3d. In: SIGGRAPH Conference Proceedings, pp. 835–846. ACM Press, New York (2006)
21. Tanimoto, M.: Overview of free viewpoint television. Signal Processing: Image Communication **21**(6), 454–461 (2006)
22. Tsai, R.Y.: A versatile camera calibration technique for high-accuracy 3d machine vision metrology using off-the-shelf tv cameras and lenses. IEEE Journal of Robotics and Automation **3**(4), 323–344 (1987)

23. Venkataraman, K., Lelescu, D., Duparré, J., McMahon, A., Molina, G., Chatterjee, P., Mullis, R., Nayar, S.: Picam: an ultra-thin high performance monolithic camera array. ACM Transactions on Graphics (TOG) **32**(6), 166 (2013)
24. Wilburn, B., Joshi, N., Vaish, V., Levoy, M., Horowitz, M.: High-speed videography using a dense camera array. In: Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 2004, vol. 2, pp. II-294. IEEE (2004)
25. Wilburn, B., Joshi, N., Vaish, V., Talvala, E.V., Antunez, E., Barth, A., Adams, A., Horowitz, M., Levoy, M.: High performance imaging using large camera arrays. ACM Transactions on Graphics (TOG) **24**(3), 765–776 (2005)
26. Zhang, C., Chen, T.: A self-reconfigurable camera array. In: ACM SIGGRAPH 2004 Sketches. p. 151. ACM (2004)