Towards portable multi-camera high definition video capture using smartphones

Surendar Chandra, Patrick Chiu and Maribeth Back
FX Palo Alto Laboratory Inc.
3174 Porter Drive, Palo Alto, CA 94304
surendar@acm.org, chiu@fxpal.com, back@fxpal.com

***Abstract*—Real-time tele-immersion requires low latency and synchronized multi-camera capture. Prior high definition (HD) capture systems were bulky. We investigate the suitability of using flocks of smartphone cameras for tele-immersion. Smartphones integrate capture and streaming into a single portable package. However, they archive the captured video into a movie. Hence, we create a sequence of H.264 movies and stream them. Capture delay is reduced by minimizing the number of frames in each movie segment. However, fewer frames reduces compression efficiency. Also, smartphone video encoders do not sacrifice video quality to lower the compression latency or the stream size. On an iPhone 4S, our application that uses published APIs streams 1920x1080 videos at 16.5 fps with a delay of 712 ms between a real-life event and displaying an uncompressed bitmap of this event on a local laptop. Note that the bulky Cisco Tandberg required 300 ms delay. Stereoscopic video from two unsynchronized smartphones also showed minimal visual artifacts in an indoor setting.**

***Keywords*-portable tele-immersion, smartphone camera**

I. Introduction

Video cameras play an important role in real time collaboration. Single cameras are used for video conferencing. With proper synchronization and calibration, multiple cameras are used for tele-immersion. With the prevalence of HD 2D and 3D monitors, there is a need to consider HD video sources.

Portable HD capture can enable a variety of applications. For example, field technicians can directly stream high fidelity immersive views of intricate components from the customer's site and consult with experienced technicians. Achieving portability while using commodity components can also reduce the overall system cost and lead to wide spread adoption of tele-immersion techniques.

In prior systems, many cameras were [1] attached to a cluster of computers that performed the necessary video processing and then transmitted the streams to the remote site for tele-immersive viewing. These cameras where synchronized amongst each other using a hardware trigger.

Many tele-immersive systems achieved low latency by sending uncompressed video data. Such an approach is bandwidth intensive; a single 3D stream at a 320x240 resolution required more than 100 Mbps [2]. A single HD 3D stream will require over three Gbps; compression is inevitable in practice. Real-time compression of high definition streams is resource intensive. Powerful laptops are expensive while servers are also bulky. Neither of them is truly portable.

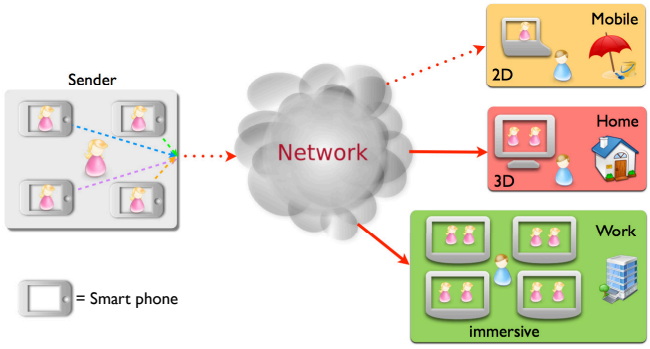
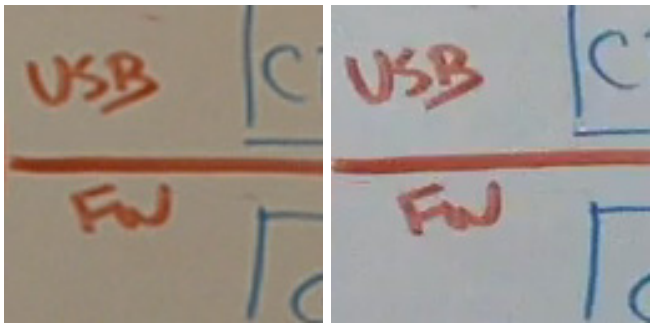


Figure 1. Capture architecture

The interconnect bandwidth between camera and the computer also plays an important role. With uncompressed videos, low bandwidth limits the supported video resolution. Uncompressed HD video requires about 1.5 Gbps; well beyond the capacity of popular IEEE 1394b (800 Mbps) or USB 2 (480 Mbps). HD cameras that support newer interconnects such as USB 3 and Thunderbolt are still in their infancy and thus expensive. A general shift towards tablets is also delaying the commoditization of HD web cameras. Some video cameras include compression hardware that encode the HD video stream before transferring them over the interconnect. For example, the Logitech HD Pro Webcam C920 uses an H.264 encoder on the camera. It uses a custom USB driver that decompresses the video at the desktop and presents them to applications. Our analysis showed that these encoders produced poor quality videos and added significant compression artifacts. The client driver also used the host CPU for its decoding operations and was poorly supported. Traditional approaches could not achieve our portability and fidelity goals.

With a projected shipment of a billion units in 2013 [3], smartphones are replacing traditional cameras [4]. They include a HD camera with good low light capture capabilities for indoor use. They are equipped with a high capacity battery and can communicate using high speed IEEE 802.11n WLAN and 4G cellular networks. They use H.264 hardware encoders. These factors make them an ideal platform for portable HD video capture. We investigate the suitability of using smartphones to replace the "camera connected to a



(a) Nikon D7000(24 fps, 26 Mbps) (b) iPhone 4S (30 fps, 5.22 Mbps)

Figure 2. 128x128 pixels from captured video

computer" model used by prior systems [5].

An investigation of various smartphones showed the strengths of the iPhone 4S for video recording. Sande et al. [6] observed that for a film maker, the iPhone 4S videos compared favorably to a USD 2,400 DSLR. The iPhone 4S uses a PowerVR SGX543MP2 GPU for video encoding. It uses an Omnivision OV8830 sensor of size 4.59 x 3.45 mm for a 35 mm equivalent crop factor of 7.6. It uses a lens of aperture f/2.4 and a focal length of 4.3 mm which corresponds to a 35 mm equivalent of 32.77 mm focal length at an aperture of f/18.3. This produces videos with a large depth-of-field. Such videos are ideal for 3D reconstruction [2]. Its hyperfocal length is about 6 ft. Thus, when the camera is focused on an object that is six feet from the camera, objects which appear between three feet and ∞ from the camera appear to be in focus. This is an acceptable distance for placing a smartphone in a typical office.

Next, we investigate the quality of videos captured by an iPhone 4S in an office setting. First, we jotted some illustrations on the white board. Using the ambient light, we took a video of it from six foot away (hyperfocal length) using an iPhone 4S and a Nikon D7000 DSLR. The Nikon DSLR D7000 with the 18-105mm VR zoom lens retails for USD 1,500. It uses a 23.6 x 15.6 mm CMOS sensor for a 35 mm equivalent crop factor of 1.5. We set the lens to a focal length of 22 mm and an aperture of f/11 to closely match that of the iPhone. This camera captures 1080p24 H.264 movies at an average bandwidth of 26.44 Mbps while iPhone captures 1080p30 movies at 5.22 Mbps. We cropped a 128x128 pixel region and show them in Figure 2. Closer examination revealed some compression artifacts with the iPhone captured image. Still, for a tele-immersive setup from a typical office room, the iPhone captured images compared favorably to the video captured by the DSLR. Note that it is not possible to stream from the DSLR itself.

Real-time collaboration requires low latency capture. ITU-T G.114 [7] recommends a one way audio delay of 400 msec for good interaction. HD resolution produces enormous amounts of video data necessitating algorithms that achieve good compression with low latency. Frequently, video quality is sacrificed to achieve bandwidth and delay targets.

iPhone offers excellent video recording. However, they are not shipped to act as streaming cameras. The AV Foundation only supports ways to write the compressed video data to a file. The compression system is also optimized to prefer quality over latency. The CPU capabilities of the iPhone preclude using our own software encoder that changes this tradeoff. Hence, we investigate whether the iPhone can be repurposed for HD video streaming without reengineering their hardware. We only use published API calls. Note that iPhone video chat systems such as Skype use software encoders with poor video quality and resolution.

We simulate streaming by creating a sequence of H.264 encoded movies. The number of frames in each segment represents a tradeoff. H.264 achieves good compression by removing inter-frame redundancy; videos with few frames are deprived of this opportunity. On the other hand, waiting to accumulate enough frames adds to the end-to-end delay. Also, H.264 encoders that are optimized for storage require some time to finish the compression process. There is an inherent tradeoff between achieving good compression and minimizing compression delay. We analyze the factors that contribute to the end-to-end delay between a real-life event and rendering a video of the event on the remote computer.

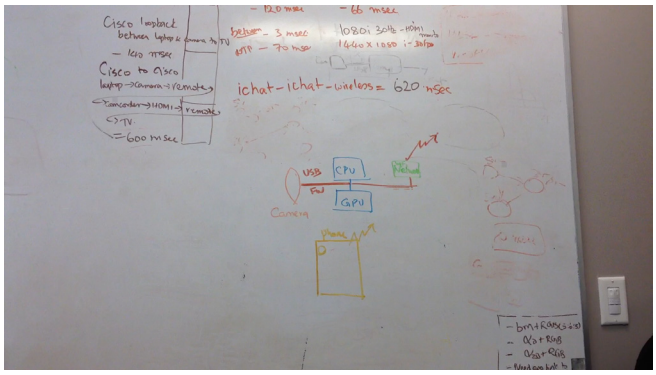
Our videos were encoded in good quality and required five Mbps of network bandwidth. The end-to-end delay was 712 msec (at 16.5 fps) while 1.1 sec delay achieved 24 fps. Next, we investigated the factors that contributed to this delay. Delays introduced by choosing the number of frames in each segment is inherent to our approach while delays introduced by hardware limitations are already being improved in the next generation smartphone.

Next, we investigated the feasibility of using our smartphones for tele-immersion. Smartphones lack hardware capture triggers. However, the slow shutter speeds required for indoor capture minimized the effects of unsynchronized video capture. We describe encouraging experiences with a remote HD Stereoscopic viewer. Overall, our end-to-end delay is tolerable for scenarios which prefer the portability and HD video quality of our approach.

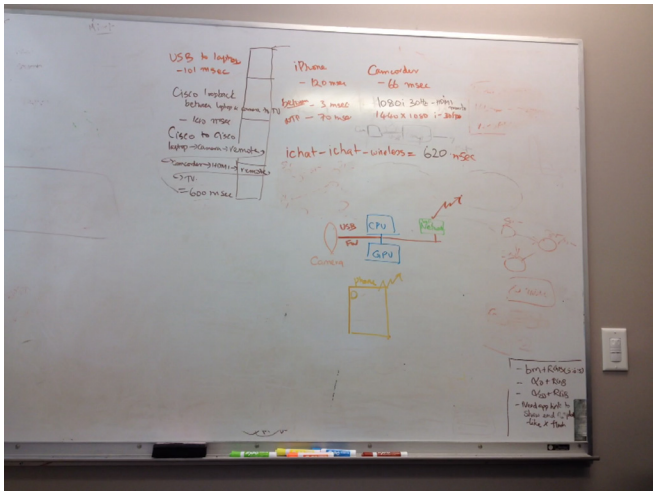
§II describes our smartphone application. §III analyzes delays introduced by our approach. §IV investigates the behavior of a flock of cameras for use in tele-immersion. We muse over ways in which the smartphone hardware and software can be improved to provide better video capture for tele-immersion purposes in §V. We present related work in §VI and conclude in §VII.

II. PhonéCam video capture application

We used the rear facing camera. iOS AV Foundation compressed movie files using the hardware encoder. We used Grand Central Dispatch to achieve parallelism. We created a *AVCaptureSession* and associated the rear facing HD camera



(a) HD (1920x1080)



(b) *Super* HD (1920x1440)

Figure 3. 16:9 and 4:3 aspect HD video with the iPhone 4S

and the audio input device as *AVCaptureDevice*s. Thus, the audio and video components are always in synchrony. Configuring the session to use the *AVCaptureSessionPresetPhoto* preset allowed us to either capture videos in 16:9 aspect ratio (1920x1080) or at a 4:3 aspect ratio (1920x1440). The 4:3 aspect ratio is more suitable for teleconferencing (Figure 3). Instead of 30 fps, the 4:3 aspect ratio stored 33% more pixels at 16 fps. Given the lower capture rate, we use 1920x1080 videos for the rest of the paper.

Unlike the built-in video capture application, we disabled image stabilization. This allowed us to retain the wider field of view as still image capture. We created two *AVCaptureVideoDataOutput*s and associated them with output files stored in the application's document directory. One of these outputs is active while the other remains ready to capture the subsequent video segment. We chose a maximum frame interval between intra-coded (I) frames of 100 thereby ensuring that each video segment will consist of a single I frame and a series of Predicted (P) frames. The audio component was encoded using AAC while video was compressed using H.264. The AV Foundation provides each audio and video sample via a *captureOutput:* callback. In this callback, we store a configurable number of frames into the currently active output. Once the required number of frames were reached, we activate the inactive *AVCaptureVideoDataOutput* output, continued to complete the compression process and then created a new *AVCaptureVideoDataOutput* (associated with a new file) in a separate thread. Once the compression is complete, the video data is read and scheduled for wireless transmission.

The dual core processor in an iPhone 4S performs many of these operations in parallel in order to achieve high effective frame rates. Note that the compressed movies are written to a file stored in flash memory. §III-D4 shows that the cost to read the file ranges between 2.5 and 10 msec. We further breakdown the delay costs for the various steps in §III.

A. PhonéCam performance

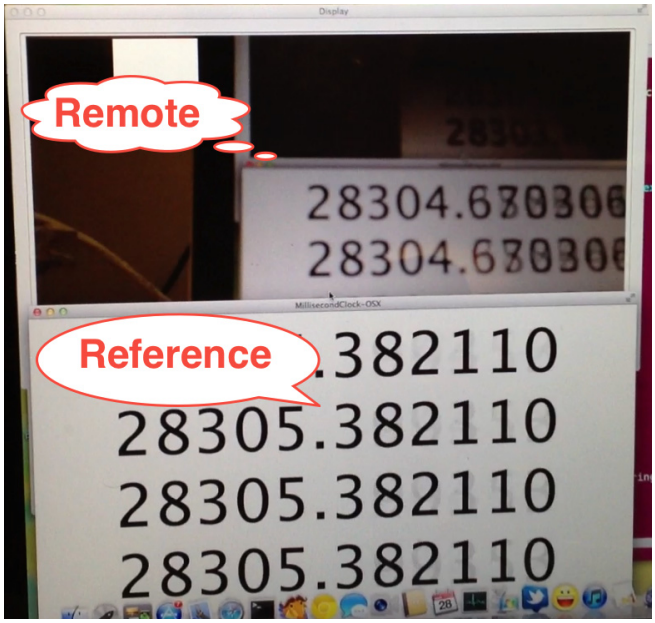
For a teleconferencing pose, we achieved frame rates of 16.5, 21.5, 24, 25.7, 27.1, 28 and 28 fps, for choosing 5, 10, 15, 20, 25, 30 and 35 frames per segment, respectively. Segments with less than five frames were unreliable because of a large compression overhead (discussed in §III-D3). The bandwidth consumed hovered around four Mbps. The HSDPA cellular technology used by iPhone 4S supports downlink bandwidth up to 14.4 Mbps. In the US, we measured downlink and uplink bandwidths of 4.9 Mbps and 1.3 Mbps. Users of AT&T LTE (iPhone 5) report downlink 20 Mbps, uplink 12 Mbps and a ping latency of 58 msec; LTE can allow PhonéCam to operate over cellular links.

III. Capture delay

Earlier, we showed the quality of images captured by iPhone 4S. We also investigated the constant bit rate nature of videos encoded by its hardware encoder. However, latency remains of import. For real-time scenarios, the duration between a real life event and when it appears on the remote monitor should be minimal. ITU G.114 [7] recommends a maximum end-to-end audio delay of under 400 msec for transparent interactivity; ideally they prefer delays under 150 msec. Without accounting for compression and transmission delays, our application (with a minimum of five frames per segment), requires $\frac{5}{30}$ seconds (166.67 msec). Next, we empirically investigate the end-to-end delay for our setup.

A. Measurement methodology

We need to measure the duration between when an event happens in real life and when it appears on the remote screen. In between, the image is captured by the video sensor, encoded into a H.264 movie, streamed over the wireless LAN network using TCP and then received, decoded and rendered on the remote display. The work flow traverses multiple computing environments (e.g., smartphone, wireless network, remote computer).



(a) duration between reference clock and remote display

(b) round trip delay between two Cisco Tandberg

Figure 4. Duration between events in different domains

The time interval within the same computing environment is calculated using the timestamps collected at the beginning and end of the event. For measuring the duration between events happening in different environments, we did not rely on time synchronization schemes such as NTP [8]. Instead, we used another camera to simultaneously capture the two events for visual inspection. E.g., to measure the end-to-end delay between when a real life event happens and when the scene is shown on the remote computer, we displayed a high resolution clock on a reference computer and then trained PhonéCam on this screen. A remote compute received the movies from the smartphone, decoded the H.264 movie and rendered it on its screen. We then took a still snapshot of the high resolution reference clock and the remotely rendered version and then visually calculated the time difference. Note that we require the high resolution reference clock to be free from any drift during our measurement interval of a few seconds; we do not require synchronization to a global clock. In the illustration in Figure 4(a), we used the same laptop for the remote and the reference clock. The reference clock was at 28,305.382110 sec and the remote end was rendering image from 28,304.670306 sec; accounting for an end-to-end delay of 711.804 msec. We repeated each experiment six times and report the average values. We describe the specific setup for measuring each delay component in §III-A.

For the reference clock, we developed an iOS/OS X application that continuously displayed the local time with μsec accuracy (using *[[NSDate date] timeIntervalSince1970]*. Smartphones use a top-to-bottom rolling shutter image capture. Hence, we displayed the same time multiple times from the top of the screen to the bottom. Even though we noticed hints of the rolling shutter effect, the capture delay between the top and the bottom was too small to be noticeable. Our reference laptop refreshed its screen at 60 Hz.

B. Delay experienced by deployed systems

ITU G.114 [7] recommends tight delay bounds. To get a realistic understanding of typical end-to-end delay, we measured them on two different deployed video conferencing systems. These systems trade off image quality to achieve low delay. For this analysis, we did not consider the picture quality loss.

1) Wireless video chat applications: We measured the delay between video chat applications using two Mac laptops, one of which used IEEE 802.11n wireless. None of these applications support HD. We ran our reference clock on the wired laptop. Using an USB2 webcam, the wireless laptop captured this clock and sent the video back to the wired laptop. We then photographed the original clock as well as the time displayed by the chat application on the wired laptop and visually computed the difference. We measured a delay of 325 msec while using iChat and 218 msec while using Skype. Both these chat applications used a peer-to-peer link to directly communicate between the two laptops without using an intermediary server. Lu et al. [9] reported similar delay while using Qnext, Vsee and Nefsis (150 msec, 270 msec and 410 msec, respectively). However, server assisted systems such as Mebeam introduced significant delays (2770 msec).

2) Cisco Tandberg C40 telepresence system: Next, we measured the end-to-end delay between two local Cisco Tandberg C40 telepresence systems (with Premium Resolution option) that were connected via the local gigabit wired network. These units either accepted a HD camera or a HDMI input which was then H.264 encoded and streamed. This system could be configured to limit its network usage between 256 and 2560 Kbps with a corresponding loss in video quality. Note that we could not measure the true video

resolution and frame rates achieved. Each end point costs over USD $20,000. The system was also not portable.

We displayed our microsecond clock. The local C40 HD camera observed this clock. At the remote end, we connected the HDMI monitor output of a HD camcorder to the C40, focused the camcorder to watch the video of the clock and thus retransmit it back to the originator. We then photographed the original clock and the video reenactment sent from the remote end (Figure 4(b)). The laptop reference clock was displaying 17,880.973569 sec while the round trip through the system showed the clock value of 17,880.383867 for a round trip value of 589.702 msec (one way latency of 294.851 ms).

C. Delay experienced by our system

First, we built a Mac OS X application that received the H.264 encoded video segments from our iPhone, decoded and then displayed them using *CoreGraphics* rendering. We ran the reference clock on our laptop, trained the iPhone to watch this clock and then captured an image of the reference clock and the rendered image captured by the iPhone. We tabulate the average delay for various number of frames in each video segment in Table I. We observe delays of over 700 ms. Next, we delve into components that contribute to this delay and describe ways to reduce them.

D. Detailed delay components

We tabulate the delay for the seven different steps used by our system in Table I.

1) Capture real-life event (t_1): We measured the delay between the real life event and when it was captured by the video sensor and shown in the camera previewer using the technique described in §III-A. This delay might include time to convert the captured image into a displayable image (i.e., could account for the rendering to screen delay after Step 7). This delay for iPhone 4S, Galaxy Nexus, Macbook Pro laptop, iPad, AVCHD camcorder and Tandberg C40 was 121.3 ms, 140 ms, 101 ms, 210 ms, 66 ms and 140 ms, respectively. ITU G.114 [7] recommends an end-to-end delay of under 150 msec for good performance; it is difficult to remain within those recommendations unless steps are taken by hardware vendors to reduce this delay.

2) Capture frames into segment (t_2): We measured the duration between when the first and last frames were added into each segment by the callback function. As a 30 fps capture camera, we expected these durations to be in multiples of 33.3 ms (of the number of frames in each segment). Table I shows the jitter from these expected values. In some cases (e.g., 25 frames per segment), the system skipped capturing some frames altogether (887.7 ms corresponds to a duration for 26.6 frames). This delay is likely to be reduced by future quad core processors.

3) Finalize compression (t_3): Since the iPhone video encoder was not designed for streaming, it introduces a delay to finish compressing all the captured frames. After the last frame was added to the movie file, we call the synchronous method *finishWriting*. We measured the compression finalization delay as the duration taken by *finishWriting*. We observed (Table I) a delay of about 160 ms. Even though this processing is performed in a background thread without affecting the video capture, it needs to be completed before a subsequent capture can proceed to a new movie segment. Thus, this delay precludes us from storing video segments shorter than 160 ms (or less than five frames per segment). This delay is likely to be reduced using faster video encoders in future iPhones.

4) Read movie segment (t_4): Next step is to read the video files that are stored in the internal flash memory. We measured delays of between 2.5 ms (for five frames) to 10.6 ms (for forty frames). OS support to compress the segment into a memory region could be useful.

5) Segment transfer latency (t_5): We define this latency as the time between when a movie segment is read and when it is actually written to the network socket by the transmission thread. This delay measures the CPU competition by the transmission thread. Even though we use a single TCP connection (and hence not pay the TCP startup overhead) to upload all the segments, we notice a significant, segment size dependent latency. Our earlier work [10] observed similar effects from interrupt processing overhead. Further work will explore the nature of this latency and develop schemes to reduce its effect.

6) Upload movie segment (t_6): iPhones uses a single spatial channel IEEE 802.11n in the crowded 2.4 GHz band. This delay measures the wireless transmission duration. We observed an effective bandwidth of 29 Mbps for our application. On the new iPad, we noticed an average gain of about 40 ms each for t_5 and t_6, either from a faster CPU or through the use of both the 2.4 GHz and 5 GHz for wireless. There is potential for reducing these network delays in the next generation smartphones by using multiple IEEE 802.11n channels.

7) Decode first frame from segment (t_7): Finally, we report the delay between when the segment was received and when the first frame was decoded into an RGB bitmap. On a Macbook Pro laptop using a 2.2 GHz Intel Core i7 processor, we measured a modest delay of about 20 ms.

Note that for segments of five and ten frames, the sum of these delays does not account for all the delay between real life event and remote screen display; the subsequent step of converting the uncompressed bitmap into a screen element added significant delay. On the other hand, using a larger number of frames per segment achieved better performance than predicted by analyzing each individual step. For example, using 30 frames per pixel, we observed an end-to-end delay of 1.2 sec even though the sum of

frames per segment	end-to-end delay (§III-C)	average delay introduced by each step (in ms)							
		t_1 (§III-D1)	t_2 (§III-D2)	t_3 (§III-D3)	t_4 (§III-D4)	t_5 (§III-D5)	t_6 (§III-D6)	t_7 (§III-D7)	$\sum_{i=1}^{7} t_i$
5	712	121.3	155.1	159.7	2.5	85.9	41.8	19.0	585.4
10	837		332.7	156.4	4.1	95.8	85.4	17.5	813.2
15	1034		543.9	170.0	5.2	100.7	116.5	19.9	1077.4
20	1143		713.0	143.8	4.2	138.2	150.9	18.3	1289.7
25	1194		887.7	131.9	5.4	132.0	205.7	19.6	1503.6
30	1248		1001.4	146.5	6.4	133.6	236.2	19.4	1664.9
35	1489		1166.1	151.5	6.9	109.9	247.9	21.3	1824.8
40	1485		1327.9	153.7	10.6	110.6	294.3	21.5	2039.9

Table I
PERFORMANCE OF OUR STREAMING APPLICATION (AVERAGED OVER SIX MEASUREMENTS)

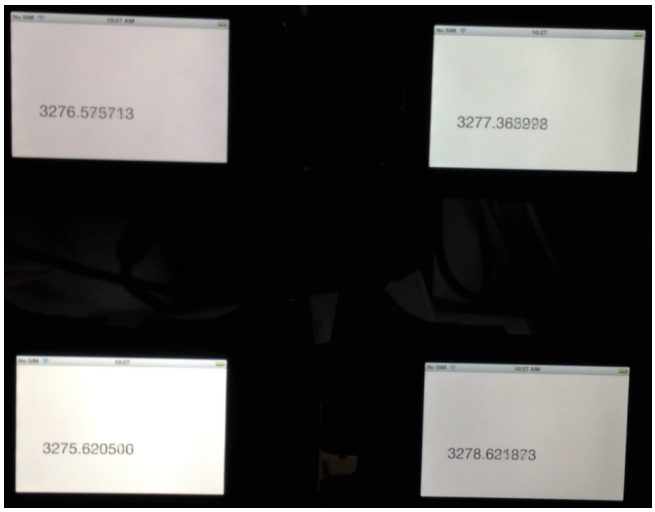


Figure 5. Local times synchronized across four iphones

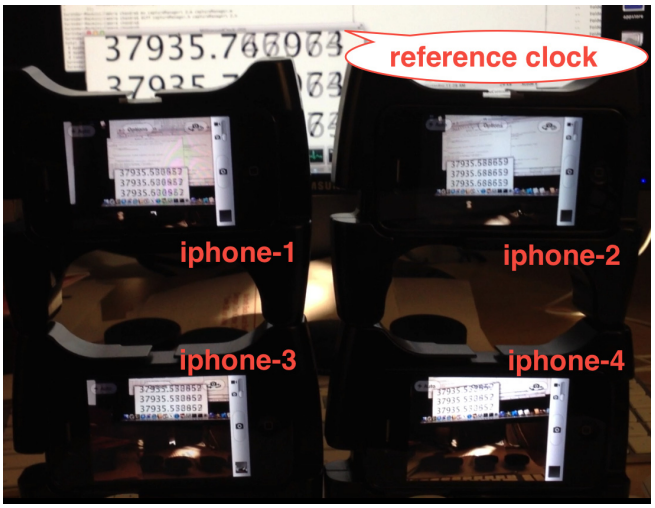


Figure 6. Synchronized capture from four unsynchronized iPhone

the overhead showed a value of 1.7 sec. Mac OS X uses the hardware decoder built into the Intel HD 3000 GPU. We observed a CPU/memory bottleneck and are exploring a GPU based rendering solution.

IV. TIME SKEW BETWEEN CAMERAS

In the last section, we analyzed the HD capture and streaming capabilities of our PhonéCam smartphone application. Time skew between cameras influences their suitability for tele-immersion. Next, we investigate the suitability of a flock of PhonéCams for tele-immersive applications. Acquiring 3D data from multiple cameras requires synchronized capture. Prior approaches used custom or off-the-shelf hardware triggers [11] to achieve such synchronization. Smartphone cameras do not offer a hardware trigger capability. Hence we investigate their suitability for providing synchronous capture. Smartphones use NTP [8] for synchronizing their clocks; we investigate whether their internal clocks are in synchrony. We also investigate the relationship between images captured by these unsynchronized devices.

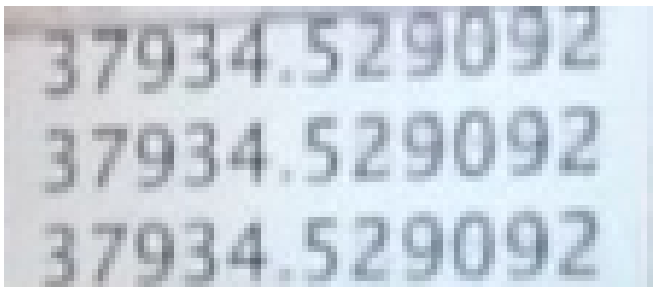
A. NTP synchronized local clock

First we analyze whether the local clocks are synchronized closely enough for event ordering. Since different PhonéCam videos are directly streamed to the rendering client, event ordering allows the recipient to choose the appropriate frame from a particular iPhone while building a 3D model. We configured the iPhones to set their clock using NTP [8] over WiFi; we did not enable cellular access. We ran our reference clock on four different iPhones and photographed them (Figure 5). We observed clock values of 3,276.575713, 3,277.363998, 3,275.620506 and 3,278.621875 seconds. The clocks were widely divergent; the clocks in two of these phones were almost three seconds apart. Resetting the iPhones did not reduce this skew. Hence, using the iOS Gamekit framework, we built a Bluetooth based mechanism in which one iPhone broadcasts it local clock; others used the difference between their local clocks and this reference value for event ordering.

B. Capture using unsynchronized iphones

The next form of skew happens when the iPhones capture the same event at different times. Hardware trigger will

(a) iPhone-1



(b) iPhone-2

Figure 7. Unsynchronized capture in detail

eliminate this skew. Since the cameras capture videos at 30 fps, we expect the cameras to be out of sync amongst each other by about 16.66 ms ($\frac{1}{60}$ sec). For this experiment, we ran the reference clock on a desktop and trained our four iPhones to record this clock. We photographed the preview screen for each of these four phones and the desktop. We observed that the previews were remarkably close, frequently indistinguishable and within three ms of each other. In Figure 6, we observe that all the four cameras are virtually in sync.

With the ambient lighting in our office, the phones needed a shutter speed of $\frac{1}{20}$ at an aperture $f/2.4$ for still shots. Since our application was capturing videos at 30 fps, the capture sensors were essentially integrating all the photons received within the past $\frac{1}{30}$s. Even when the time in which a particular sensor values was read differed by $\frac{1}{60}$ sec, the information in the captured images were similar. When the captured object was moving, we observed that the shades of the object changed between video frames from different cameras. We high-light this effect between iPhone-1 and iPhone-2 (in Figure 6) in Figure 7. As the precision of the reference clock increases, the synchronicity difference between the cameras manifests itself in a blurred image. For stereoscopic viewing of videos from two cameras, this mismatch was subjectively acceptable for normal movements in a tele-conferencing setup. Further investigations on the effect of this mismatch in sunny outdoors as well as active scenes is the subject of ongoing work.

V. Wish list for future smartphones

Though the image quality was excellent, the capture delay can be high. Providing a streaming API that returns a H.264 encoded video from the camera would be a welcome improvement as it can eliminate the capture frames into segment (t_2) and read movie segment (t_4) delays. We summarize other technologies that could reduce the capture delay and synchronization artifacts.

- *More control over video capture sensor:* The iPhone optical system offers excellent light capture capability. PhonéCam can benefit from the availability of more software control over the capture process. E.g., an ability to lock the camera focus to its hyper focal length as well as control the capture aperture would be invaluable. Note that the upcoming iOS 6.0 does not provide this functionality. Also, we showed an undocumented ability to capture videos at 1920x1440 resolution. An ability to encode such a resolution video at 30 fps would be invaluable as 4:3 aspect ratio videos have a more natural feel in a tele-conferencing setup. Finally, the camera APIs support setting the capture aperture though the captured images still stayed in the central region of the 3264x2448 pixel optical sensor. An ability to chose the capture region within the larger image sensor can provide software panning capability to PhonéCam .
- *More cores for better parallelization:* The iPhone supports two video encoders and two CPUs. More CPU cores can reduce delays such as the segment transfer latency (t_5). More video encoders can help parallelize the compression finalization phase (t_3). They can also allow the clients to adopt to network conditions by encoding videos with different bandwidth targets; each bandwidth will require an additional pair of video encoders. Apple reports that the upcoming A7 processor in the iPhone 5s doubles the CPU performance. We will investigate the performance improvement with this new hardware.

VI. Related work

Tele-immersive systems had traditionally [2] reduced the end-to-end delay by either streaming uncompressed or low resolution video. Kauff et al. [12] built a hardware MPEG4 that used ITU BT.601 resolutions. They used MPEG4 arbitrary shapes and disparity map as auxiliary alpha bitplane to reduce the network requirements. Jung et al. [13] used 12 dual core PCs, each with 3 B&W cameras and IR structured lighting for depth and 1 color camera to compress 640x480 tele-immersive video at 15 fps. Vasudevan et al. [14] performed 3D reconstruction in under 30 msec. They used Bumblebee 2 cameras at 320x240 for 3D reconstruction and achieved a compression ratio of 55% in 20 msec.

Wu et al. [11] describe a mobile tele-immersive capture system that builds on their prior work on Teeve [5]. They report a 3D video frame rate of 17.3 fps over wireless LAN.

They do not report the end-to-end latency or the bandwidth consumption. They use an external BumbleBee2 camera and associated computing infrastructure. Our approach can provide good quality HD videos in a smartphone form-factor that incorporates the capture and streaming component. We believe that the capture latency is acceptable. Further work will define the parameters of useful usage scenarios.

Tele-immersive systems [5] had traditionally used webcams for video capture. HD capture is limited by the interconnects between the camera and the computer; USB2 and FW800 are inadequate. Some webcams address this limitation using compression hardware. We investigated several webcams that used Motion JPEG or H.264 encoding and observed the compressed streams to be of poor quality with unacceptably high compression artifacts. Good quality compression hardware remains expensive. Technology improvements typically lead to a reduction in hardware costs. However, external webcams are being replaced by cameras built into monitors. Even though newer interconnects such as USB3 can support bandwidths of 5 Gbps between the camera and the computer, few stand alone HD cameras are being built to use this capacity. Another alternative is to use the video monitoring over HDMI capabilities of HD camcorders and using HDMI capture cards on the computer. Many HDMI capture cards are built for video recording and introduce inordinate latencies to the capture workflow.

Seo et al. [15] segmented a stored video file and uploaded them from a smartphone for HTTP live-streaming. We are concerned with real-time capture and streaming with minimal latency.

Murai et al. [16] developed a mediation mechanism to counteract latencies of over 700 ms observed in their conferencing setup. Our system could benefit from their approach for real time interaction.

VII. Discussion

Apple sold over one million iPhone 4S within the first 24 hours. This work leverages the economies of scale unleashed by smartphones to build portable HD video capture for tele-immersive scenarios. Though the image quality was excellent, the capture delay can be high.

Our work can benefit from the availability of a streaming API that returns a H.264 encoded video. PhonéCam can also benefit from more software control over the capture process. For example, an ability to lock the camera focus to its hyper focal length as well as control the capture aperture would be invaluable. More CPU cores can reduce delays such as the segment transfer latency (t_5). More video encoders can help parallelize the compression finalization phase (t_3).

References

[1] J. Liang, Z. Yang, B. Yu, Y. Cui, K. Nahrstedt, S.-H. Jung, A. Yeap, and R. Bajcsy, "Experience with multi-camera tele-immersive environment," in *NSF Workshop on Pervasive and Cluster Computing*, 2005.

[2] Z. Yang, B. Yu, K. Nahrstedt, and R. Bajscy, "A multi-stream adaptation framework for bandwidth management in 3d tele-immersion," in *NOSSDAV '06*, 2006, pp. 14:1–14:6.

[3] K. Restivo and R. Llamas, "Worldwide mobile phone market forecast to grow 7.3% in 2013 driven by 1 billion smartphone shipments," http://www.idc.com/getdoc.jsp?containerId=prUS24302813, Sep. 2013.

[4] M. Lee, "Iphone grabs camera market from sony: Chart of the day," Bloomberg News, Mar. 2012.

[5] Z. Yang, K. Nahrstedt, Y. Cui, B. Yu, J. Liang, S.-h. Jung, and R. Bajscy, "Teeve: The next generation architecture for tele-immersive environment," ser. ISM '05, 2005, pp. 112–119.

[6] S. Sande, "iPhone 4S video compared to Canon 5D MK II," http://www.tuaw.com/2011/10/17/iphone-4s-video-compared-to-canon-5d-mk-ii/, Oct. 2011.

[7] I. T. S. Sector, "Series g: Transmission systems and media, digital systems and networks," May 2003.

[8] D. L. Mills, J. Martin, J. Burbank, and W. Kasch, "Network time protocol version 4: Protocol and algorithms specification," RFC 5905, Jun. 2010.

[9] Y. Lu, Y. Zhao, F. Kuipers, and P. Van Mieghem, "Measurement study of multi-party video conferencing," in *Proceedings of the 9th IFIP TC 6 international conference on Networking*, ser. NETWORKING'10. Chennai, India: Springer-Verlag, 2010, pp. 96–108. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-12963-6_8

[10] P. Xue and S. Chandra, "Revisiting multimedia streaming in mobile ad hoc networks," in *ACM Network and Operating Systems Support for Digital Audio and Video (NOSSDAV '06)*, Newport, RI, May 2006.

[11] W. Wu, R. Rivas, A. Arefin, S. Shi, R. M. Sheppard, B. D. Bui, and K. Nahrstedt, "MobileTI: a portable tele-immersive system," in *ACM Multimedia*, ser. MM '09. Beijing, China: ACM, 2009, pp. 877–880.

[12] P. Kauff and O. Schreer, "An immersive 3d video-conferencing system using shared virtual team user environments," in *CVE '02*, 2002, pp. 105–112. [Online]. Available: http://doi.acm.org/10.1145/571878.571895

[13] S.-H. Jung and R. Bajcsy, "Learning physical activities in immersive virtual environments," in *IEEE ICVS '06*, New York, NY, Jan. 2006, pp. 5–.

[14] R. Vasudevan, Z. Zhou, G. Kurillo, E. J. Lobaton, R. Bajcsy, and K. Nahrstedt, "Real-time stereo-vision system for 3d teleimmersive collaboration," in *ICME'10*, Singapore, Jul. 2010, pp. 1208–1213.

[15] B. Seo, W. Cui, and R. Zimmermann, "An experimental study of video uploading from mobile devices with http streaming," in *ACM MMSys '12*, 2012, pp. 215–225. [Online]. Available: http://doi.acm.org/10.1145/2155555.2155589

[16] K. Murai, D. Kimber, J. Foote, Q. Liu, and J. Doherty, "Mediated meeting interaction for teleconferencing," in *ICME '05*, Jul. 2005, pp. 1436–1439.