Enabling Access Through Real-Time Sign Language Communication Over Cell Phones

Jaehong Chon*, Neva Cherniavsky†, Eve A. Riskin* and Richard E. Ladner†
*Department of Electrical Engineering, University of Washington, Seattle, WA 98195
{jaehong, riskin}@u.washington.edu
†Department of Computer Science and Engineering, University of Washington, Seattle, WA 98195
{nchernia, ladner}@cs.washington.edu

***Abstract*—The primary challenges to enabling real-time two-way video conferencing on a cell phone are overcoming the limits of bandwidth, computation and power. The goal of the MobileASL project is to enable access for people who use American Sign Language (ASL) to an off-the-shelf mobile phone through the implementation of real-time mobile video communication. The enhancement of processor, bandwidth, and power efficiency is investigated through SIMD optimization, region-of-interest encoding based on skin detection, video resolution selection (used to determine the best trade off between frame rate and spatial resolution), and variable frame rates based on activity recognition. Our prototype system is able to compress, transmit, and decode 12-15 frames per second in real-time and produce intelligible ASL at 30 kbps using a current cellular data network in the U.S. as well as Wi-Fi. Furthermore, we can achieve up to 23 extra minutes of talk time, or a 8% power gain over the battery life of the phone, through our frame dropping technique.**

I. Introduction

There have been a multitude of issues that have prevented real-time video communication on the mobile phone. Mobile sign language communication is already available in Japan and Europe, but regardless of the higher bandwidth 3G network there, the quality is poor, the videos are jerky, and there is significant delay. With the advent of mobile phone PDAs equipped with larger screens and photo/video capture, people who communicate with American Sign Language (ASL) should be able to use these new technologies.

Mobile phone service providers operate wireless networks using many different wireless communication standards. The most widely used of these technologies is called the Global System for Mobile Communications, or GSM. On top of a GSM network, they operate a data network called the General Packet Radio Service (GPRS) and an upgrade for faster speeds called Enhanced Data Rates for GSM Evolution (EDGE), which can carry data speeds from 35.2 kbps up to 236.8 kbps. Furthermore, they have launched a high-speed network based on UMTS and High-Speed Downlink Packet Access (HSDPA), which is commonly known as 3G. Since a video call is bi-directional, the effective bandwidth is limited by the uplink. Our project, with its target bit rate of 30 kbps, is well suited to mobile sign language communication over current U.S. cellular data networks.

Although various approaches to processing graphics in personal computers such as high clock frequency, dual cores and various Single Instruction Multiple Data (SIMD) instruction sets have been investigated, computing capability on mobile phones is still very low because of low clock frequencies, single cores, and only basic SIMD. In our work, we utilize assembly optimization to solve the issue of limited processing power. Second, we implement dynamic region-of-interest (ROI) coding based on skin detection to reduce the required bandwidth while keeping the amount of quality and intelligibility needed for American Sign Language. Third, we investigate a variable frame rate to prolong the battery life time by taking advantage of the fact that, for the most part, an ASL speakers signs when the other person isn't signing. Building on previous work[1], [2], we optimized a variable frame rate. Finally, we designed ASL encoders that are compatible with the new H.264/AVC compression standard[3] by using x264[4] at very low bit rates to achieve a high enough frame rate in real time on the mobile phone.

II. Background

MobileASL[5] is a video compression project at the University of Washington and Cornell University[6] with the goal of making cell phone communication for people who use sign language a reality in the U.S. Our software runs on the Windows Mobile operating system. MobileASL is compatible with the H.264/AVC compression standard and can be decoded by any H.264 decoder. The x264 Open Source H.264 encoder was selected because of its fast speed[4], [7], [8], making it a good choice for compression on low-power devices such as mobile phones. In [7], x264 showed better quality than several commercial H.264/AVC encoders. When compared with the JM reference encoder (version 10.2)[9], x264 (version 0.47.534) was shown to be 50 times faster, while providing bit rates within 5% for the same PSNR.

A. MobileASL System

Since the Internet Protocol Suite (commonly known as TCP/IP) is the set of communications protocols used for the Internet and other similar networks, we designed a lightweight TCP/IP networking system to enable use of the MobileASL codec on two different networks: the Wi-Fi network and a current cellular data network in the U.S. (AT&T).

In our system, we use a light and simple protocol over TCP for situations needing reliable, ordered transport service which

is call signaling, e.g., connection establishment, connection release, and connection control such as for packet loss indication and UDP for video transmission. Fig. 1 shows the TCP/IP layered architecture, which is assisted by an HTTP protocol for MobileASL.

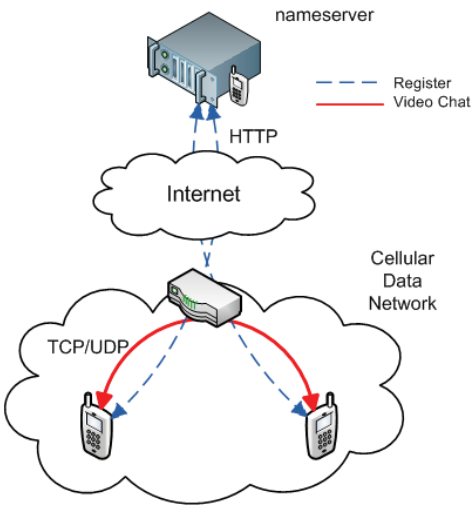

Fig. 1. Network Architecture for MobileASL

The HTTP protocol is used to register user information (such as IP address) to the server, which is located outside the network. Whenever the user calls another person, our system retrieves its IP address from the server and uses it to make a connection.

B. Hardware Platform

We use the HTC TyTN-II mobile phone which has a Qualcomm MSM7200 (400MHz ARM ARM1136EJ-S processor). We chose this phone because it has a front camera on the same side as the screen and runs Windows Mobile 6.1. Its processor adopts the ARMv6 core having a new SIMD instruction set[10]. This phone has a 320x240 pixel screen, a VGA video call front camera, a QWERTY keyboard, a H.264 hardware decoder, etc. Also, the HTC TyTN-II provides wireless capabilities such as Wi-Fi 802.11b/g, 2G (GPRS and EDGE) and 3G (HSDPA). Fig. 2 shows a MobileASL screenshot.

III. Speeding Up The Encoder

Frame rates as low as 6 frames per second can be intelligible for signing though it would require the user to sign very slowly. A more comfortable frame rate would be 12 frames/second[11], and higher frame rates would be needed for fingerspelling[12], [13], [14]. First of all, to approach our desired frame rate of 12-15 frames/second, it was necessary to optimize the steps of the H.264 compression algorithm for motion estimation, mode decision, transforms, quantization and motion compensation using SIMD instruction sets. Then we determined which H.264 parameter settings and the video resolution we needed to achieve our target frame rate.

Fig. 2. A screenshot of our MobileASL codec on the HTC TyTN-II. Two UW graduate students are talking each other.

A. Assembly Optimization

The ARM1136J-S processor used in the HTC TyTN-II cell phone is built around the ARM11 core in an integer unit that implements the ARM architecture v6. It supports a range of single instruction multiple data (SIMD) DSP instructions that operate on pairs of 16-bit values held in a single register, or on quadruplets of 8-bit values held in a single register[10]. The main operations supplied are addition, subtraction, multiplication, selection, pack and saturation. Operations are performed in parallel.

1) Motion Estimation: Motion estimation is the most time-consuming module in video coding. The sum of absolute difference (SAD) is used as the distortion measure criterion. In the ARMv6, the instruction USADA8 performs the sum of absolute differences of four 8-bit data and accumulates the results.

In H.264, there are 7 different block size types (from 16×16 to 4×4). In the $16\times16, 16\times8, 8\times16, 8\times8$ and 4×8 modes, there are 8 pixels in one line which are stored consecutively in memory. Using doubleword loading and USADA8, only 4 cycles are needed (2 loads and 2 SAD operations). Fig. 3 shows this process.

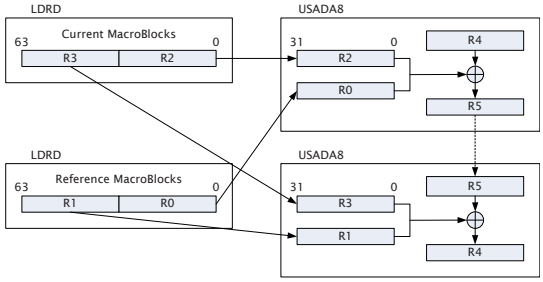

Fig. 3. SAD calculation using the USADA8 instruction.

In the 8×4 and 4×4 modes, one line has 4 consecutive pixels (32 bits), so we can load two lines of data to the register with load instruction and then they are calculated with USADA8. It spends 3 cycles. For the best optimization

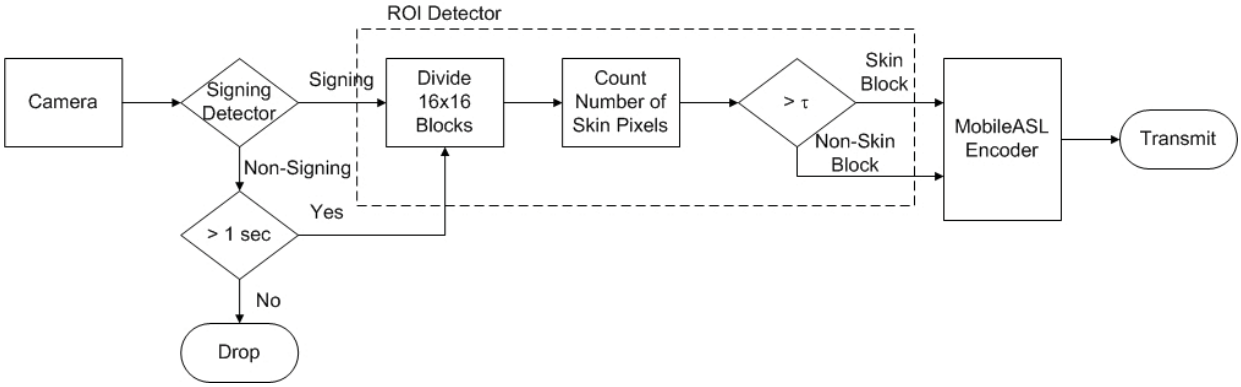

Fig. 4. MobileASL Framework. The variable frame rate and ROI detector based on skin blocks are concatenated before encoding

results, we use one of two methods depending on what mode is selected.

2) Mode Decision: The H.264 encoder uses rate-distortion optimization to select the mode. Because of its complexity, mode selection is clearly the next candidate for speed up.

The distortion is computed as the sum of squared differences (SSD) between the original block and the reconstructed block. The SSD for 4 pixels between two macroblocks can be optimized using three SIMD instructions. There are two consecutive steps which are 8-bit absolute difference (UQSUB8 and ORR) and then multiplication (SMLAD). It enables us to get approximately twice the speed otherwise possible.

3) Transforms: H.264 uses three transforms depending on the type of residual data that are to be coded: a transform for the 4×4 array of luma DC coefficients in intra macroblocks (predicted in 16×16 mode); a transform for the 2×2 array of chroma DC coefficients (in any macroblock); and a transform for all other 4×4 blocks of residual data.

The DC coefficient of each 4×4 block in the 16×16 Intra prediction mode is transformed using a twice 1-D 4×4 Hadamard transform, which uses additions and subtractions (QADDSUBX, QADD16, QSUB16) and shifts (SHADD16) in 16-bit arithmetic. The other two transforms are also implemented in a similar way.

B. H.264 Parameters Optimization

Based on assembly optimized encoder, we determined H.264 parameter settings that have low complexity but still offer high video quality at 30 kbps by training on 6 videos recorded with a cell phone at QCIF resolution. We found that setting some x264 encoding parameters such as the number of reference frames and motion search method used to their lowest settings gave us almost the same quality as that of the best settings. Furthermore, the setting for the parameters sub-pixel motion estimation method and partition size for intra and inter motion compensation were chosen with both speed and quality in mind. Since the sub-pixel motion estimation parameter is complicated, the required time highly increases with higher settings, so we chose its lowest setting. The quality of the partition size while keeping other parameters unchanged is saturated as the speed increases. Therefore, we chose a "I4x4,P8x8" because it provides the best position with respect to speed and quality.

C. Video Resolution

Even though we sped up the processing time through assembly and parameter optimization, we did not reach our target frame rate. Since H.264/AVC uses block-based motion compensation, the encoding time is highly related to the spatial resolution of the video. We investigate tradeoffs in spatial resolution versus speed for QCIF (176×144), 160×128, 144×112, 128×96, 112×80, 96×80, 80×64, 64×48, 48×32, and 32×16. In future work, we will conduct a controlled user study with ASL speakers to determine preferences for tradeoffs between spatial and temporal resolution.

IV. Bandwidth and Power Enhancement

The variable frame rate and ROI detector based on skin blocks are concatenated before encoding to achieve longer talking time and better quality in the all-important face and hands regions without increasing bit rates. Fig. 4 shows this framework.

A. Bandwidth Enhancement with ROI Encoder

In [15], we used a fixed ROI by varying the level of distortion in a fixed region surrounding the face of the signer. We found that a tradeoff of 6 decreased quantization steps near the face of the signer (doubling the quality in that region) was preferred over a typical (no region-of-interest) encoding. We then extended this enhancement by finding face and hands regions using skin detection.

Since H.264 is the most recent block-oriented motion estimation based standard video codec, we first divide 16x16 blocks in each frames and then detect skin in real-time via a simple and well-known RGB-based algorithm[16] that works for many different skin tones. We designate macroblocks that have more pixels than a threshold as skin blocks. These blocks are encoded with lower quantization parameters to increase the quality. Fig. 5 shows which areas are considered to be skin blocks and how the quality of those blocks is enhanced.

Fig. 5. A snap shot of ROI-based Encoder (a) original frame with skin blocks (red lines) (b) 0 ROI (c) 12 ROI. The quality in skin blocks is enhanced and quality in the rest of blocks is degraded.

B. Power Enhancements

Minimizing power consumption in mobile devices is an important challenge. We decrease frame rate during "listening" which is equivalent to voice phone filtering. Since turns are taken communicating while speaking in conversational sign languages, the video showing the listener does not have to move much. Therefore, we can reduce the frame rate during the time of listening so that the usage of the encoder and wireless link will be decreased. This, in turn, would decrease the amount of battery consumption.

We recognized the fact that the user will not be signing all of the time. In [1], we performed a user study to determine if variable frame rates sacrifice intelligibility and if automatic activity analysis is feasible. Pixel differences between successive frames in the video were considered on the phone because it is in real-time and its results are promising.

1) Pixel Differences: For each frame k in the video, the luminance component of each pixel location (i, j) is chosen with which we calculate the sum of absolute differences, $d(k)$, between the current and previous frames at the same pixel location. If it is greater than the threshold, τ, we classify the frame as signing. We used ARMv6 SIMD instruction USADA8 to calculate the differences on the phone.

2) Signing vs. Not-Signing: We reduce the frame rate to 1 fps when the user is not-signing. To reduce false negatives, that is, signing frames being misclassified as not-signing, we only change to not-signing mode when three consecutive frames are detected as not-signing. In contrast, whenever we detect signing, we return immediately back to signing mode.

V. Experimental Results

We present experimental results. First, we show the increase in speed we achieved with SIMD optimization. Next, we describe how we not only reduced bandwidth but also achieved speed enhancement through ROI, followed by a description of battery life extension by frame dropping.

A. SIMD Performance Comparison

Experiments were conducted with fourteen QCIF (176×144) test sequences, each representing a different class of spatial detail and motion from the standard MPEG data set and an ASL data set developed at the University of Washington[17]. The MPEG set videos are foreman, carphone, container, grandma, missam, salesman and Akiyo and the ASL videos are accident, day in the life, education, favorite restaurant, food at home, graduation and segment8.

TABLE I
Frame Per Second Comparison with MPEG and ASL Set (encoder only)

Type	QCIF test sequence	without SIMD (fps)	with SIMD (fps)
ASL	accident	13.6	15.2
	day in the life	13.1	14.7
	education	14.6	16.1
	favorite restaurant	12.2	13.8
	food at home	12.6	14.3
	graduation	14.6	16.2
	segment8	14.3	15.9
MPEG	foreman	9.7	11.2
	carphone	9.8	11.3
	container	11.8	13.2
	grandma	12.9	14.4
	missam	10.2	11.5
	salesman	14.7	16.3
	akiyo	14.6	16.1

TABLE II
PSNR Comparison of Different ROI

	30kbps 0 ROI (dB)	30kbps 12 ROI (dB)	44kbps 0 ROI (dB)
ROI PSNR	27.5	29.8	29.8
Non-ROI PSNR	29.3	24.3	31.4

The performance of our optimized H.264 encoder with and without instruction optimization is shown. Table I shows a frames/second comparison *for the encoder only* for our data sets for two scenarios. Our instruction optimization for H.264 increased encoding frame rate up to 15.3% for the MPEG data set and 13.4% for the ASL data set.

B. ROI Encoding

We compared the quality of the video recorded by the phone for three different ROIs (0, 6 and 12) that were based on the optimized H.264 encoder described earlier. Table II shows the PSNR for two different ROIs. ROI PSNR for 12 ROI at 30 kbps corresponds to ROI PSNR for 0 ROI at 44 kbps. Therefore, we achieved a 32% bandwidth enhancement. As expected, 12 ROI increased quality in the ROI at the expense of quality in the background.

Another advantage of ROI-based encoding is its speed.

TABLE III
MACROBLOCK DISTRIBUTION IN P FRAMES FOR DIFFERENT ROI

Macroblock Size	0 ROI (%)	12 ROI (%)
SKIP	45.6	57.7
Others	54.4	42.3

TABLE IV
ENCODING TIME BREAKDOWN AND COMPARISON FOR DIFFERENT ROI

Encoder Function	0 ROI (ms)	12 ROI (ms)
Mode Decision	37.3	32.5
Transform and Encoding	12.2	10.0
Entropy Coding (CAVLC)	2.0	1.7
NAL and Slice Coding	2.7	2.6
Others	27.6	27.1
Total	81.9	74.0

Table III explains how macroblocks are selected in different ROIs during the mode decision stage of encoding. More bits are allocated to skin blocks with lower quantization parameters and fewer bits are used for other blocks. That is, more blocks are chosen as the skip block in the early stage of mode decision. The increased number of skip blocks affect macroblock encoding, entropy coding and slice coding.

Table IV summarizes how the encoding time varied. It proves that the video encoding stages that are related to macroblocks such as mode decision, transform and encoding, entropy coding, and slice coding is faster. Therefore, we can subtract an additional 9.7% of encoding time from an already optimized encoding time through assembly and parameters as shown earlier.

To summarize, the encoder is not the only module that consumes resources in real-time video communication. The decoder, the camera interface capturing the video, the screen interface displaying the video, and the transmission module all operate simultaneously with the encoder. With our optimizations, our encoder occupies about 50% of the CPU time. As a result, we are able to achieve a frame rate of 12-15 fps at 30 kbps in real-time on the HTC TyTN-II. We showed that our system demonstrated the success of our system over Wi-Fi and the AT&T Cellular Network. A sample encoded ASL video can be viewed on YouTube[18].

C. Picking The Resolution

Table V shows the encoding time of different video resolutions for the test sequences foreman and salesman from the MPEG data set and accident and graduation from our ASL data set. Based on this data, we chose the 96×80 resolution because our encoder with our optimizations occupies about 50% of the CPU time. This enables us to achieve up to 15 fps.

D. Power Enhancement

We simulated sign language communication and measured the instantaneous power usage every 5 seconds for an hour on the phones with the variable frame rate on and off. The power draw drops when the frame rate is lowered due to the lower processing power required to encode and transmit at 1 fps. Power saving is significant by utilizing frame dropping. This corresponds to 23 extra minutes of talk time, or a 8% power gain over the battery life of the phone.

TABLE V
ENCODING FRAME PER SECOND COMPARISON WITH ASL AND MPEG TEST SEQUENCES

Video Resolution	accident	graduation	foreman	salesman
176x144	15.1	15.8	10.9	15.9
160x128	17.7	18.8	13.2	18.6
144x112	21.2	22.7	16.2	22.0
128x96	26.0	27.4	20.5	26.0
112x80	32.5	34.4	26.6	31.7
96x80	36.0	37.8	30.6	35.1
80x64	45.4	47.2	42.2	46.8
64x48	64.4	65.8	63.4	67.8

VI. CONCLUSION

In this paper, assembly optimization and parameter selection for an H.264 encoder are presented to enable real time video communication over a mobile phone. Experimental results demonstrate that we can provide 12-15 frames/second at a 96×80 resolution with our optimized ASL encoder. Our MobileASL framework with variable frame rate and ROI encoder is suitable for video communication in very low bit rate wireless network environments.

ACKNOWLEDGMENT

We would like to thank Louis Hsu and Fish Weng of HTC for helping us to get access to the front camera on the HTC TyTN-II cell phone. This work was supported by NSF grants CCF-0514353 and IIS-0811884.

REFERENCES

[1] N. Cherniavsky, A. Cavender, E. Riskin, and R. Ladner, "Variable Frame Rate for Low Power Mobile Sign Language Communication," in *Proceedings of ASSETS 2007*. Tempe, AZ, October 2007, pp. 163–170.

[2] N. Cherniavsky, R. Ladner, and E. Riskin, "Activity Detection in Conversational Sign Language Video for Mobile Telecommunication," in *Proceedings of 8th IEEE International Conference on Automatic Face and Gesture Recognition*. Amsterdam, The Netherlands, September 2008.

[3] T. Wiegand, G. Sullivan, G. Bjontegaard, and A. Luthra, "Overview of the H.264/AVC Video Coding Standard," *IEEE Trans. On CSVT*, vol. 13, pp. 560–576, July 2003.

[4] "x264 - a free H.264/AVC encoder," http://www.videolan.org/developers/x264.html.

[5] "MobileASL," http://mobileasl.cs.washington.edu/.

[6] "Visual Communications Lab," http://foulard.ece.cornell.edu/.

[7] L. Merritt and R. Vanam, "Improved rate control and motion estimation for H.264 encoder," in *Proceedings of ICIP*, vol. 5, 2007, pp. 309–312.

[8] "4th Annual MSU MPEG-4 AVC/H.264 Video Codec Comparison," 2007, http://www.compression.ru/video/codec_comparison/mpeg-4_avc_h264_2007_en.html.

[9] "H.264/AVC JM Reference Software," http://iphome.hhi.de/suehring/tml/.

[10] "ARMv6-M Architecture Reference Manual," http://infocenter.arm.com/help/index.jsp.

[11] I. T. S. Sector, "Draft application profile: Sign language and lip reading real time conversation usage of low bit rate video communication," 1998.

[12] R. A. Foulds, "Biomechanical and perceptual constraints on the bandwidth requirements of sign language," in *IEEE Trans. On Neural Systems and Rehabilitation Engineering*, vol. 12, March 2004, pp. Vol I: 65–72.

[13] G. Sperling, M. Landy, Y. Cohen, and M. Pavel, "Intelligible encoding of ASL image sequences at extremely low information rates," *Computer vision, graphics, and image processing*, vol. 31, no. 3, pp. 335–391, September 1985.

[14] B. F. Johnson and J. K. Caird, "The effect of frame rate and video information redundancy on the perceptual learning of American Sign Language gestures," in *CHI '96: Conference companion on Human factors in computing systems*. New York, NY, USA: ACM Press, 1996, pp. 121–122.

[15] A. Cavender, R. Vanam, D. Barney, R. Ladner, and E. Riskin, "MobileASL: Intelligibility of sign language video over mobile phones," in *Disability and Rehabilitation: Assistive Technology*. London: Taylor and Francis, June 2007.

[16] S. L. Phung, A. Bouzerdoum, and D. Chai, "Skin Segmentation Using Color Pixel Classification: Analysis and Comparison," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, pp. 148–154, January 2005.

[17] "MobileASL Study Videos," http://mobileasl.cs.washington.edu/downloads/studyVideos/.

[18] "mobileasl-researchmovie," http://www.youtube.com/watch?v=FaE1PvJwI8E.