A Lightweight H.264-based Hardware Accelerated Image Compression Library

Jie Jiang [*] University of Illinois at Chicago
Thomas Fogal[†] NVIDIA
Cliff Woolley[‡] NVIDIA
Peter Messmer[§] NVIDIA

Abstract

Hardware video encoding can lower the perceived latency for remote visualization. We have created a lightweight library that simplifies the use of NVIDIA's video compression hardware. We achieve overall latencies below 15ms with compression ratios of approximately 85.5:1. To verify its applicability in real world scenarios, we integrated our library into ParaView. This offloads the encoding within ParaView to the GPU and provides a 25x bandwidth reduction compared to existing image compression methods available in the tool.

1 Introduction

Image compression is commonly used in remote visualization systems to create a smoother user experience over low-bandwidth links. Current systems generally consider each image in isolation even though image differencing approaches can yield considerable data savings.

H.264 is a block-oriented motion-compensation-based video compression standard [8]. The standard provides high compression ratios but this ability comes at significant computational expense. Modern NVIDIA hardware provides a hardware-accelerated H.264 encoder [5], enabling real-time encoding with minimal overhead. Support for the encoder has already been integrated into popular multimedia frameworks [1].

2 Design

Our library can utilize NVIDIA's hardware encoder or a libx264-based software encoder. Video encoder configuration is complicated [8]; we specifically designed our library to abstract away most encoder settings using values appropriate for visualization, trading raw performance for simplicity and ease of use. For example, video streaming normally has a lag time on the order of tens of frames. We have reduced the lag time to a single frame and guaranteed that the encoder produces a single buffer for every input image, which matches how current tools [2, 3] use image compressors.

Our library handles only image compression and decompression. The encoding side returns only an opaque pointer and a size; client code is expected to handle network communication. This protocol-agnostic approach facilitates easy integration of the library into any client/server-based visualization application.

2.1 Performance

We measured three aspects of our compression library's performance: computational efficiency, compression ratio, and image quality.

[*]e-mail: jjiang24@uic.edu
[†]e-mail: tfogal@nvidia.com
[‡]e-mail: jwoolley@nvidia.com
[§]e-mail: pmessmer@nvidia.com

2.1.1 Compression Ratio

The compression ratio is determined by an 'average bitrate' parameter. A guideline for bitrate setting is the Kush Gauge [4]:

$$bitrate = resolution * \text{framerate} * f_m * 0.07 \quad (1)$$

Where f_m is the 'motion factor' that defines the estimated motion of the video on a scale from 1 to 4, with 1 indicating the least motion and 4 the most. The default bitrate in our library uses $f_m = 4$ and an estimated 30 frames per second.

The compression ratio r_c for an 8-bit RGB image is calculated using Equation (2).

$$r_c = \frac{resolution * \text{framerate} * 3 * 8}{bitrate} = \frac{3*8}{0.07 f_m} = \frac{343}{f_m} : 1 \quad (2)$$

2.1.2 Image Fidelity

Our library utilizes encoder settings for the lossy variant of H.264. Though these settings can introduce artifacts at sharp edges in the image, Figure 1 demonstrates that these artifacts are minor.

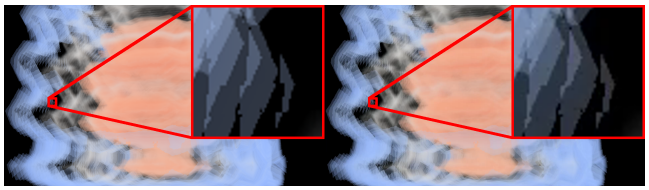

Figure 1: Image quality with high ($f_m = 4$, 30 FPS) bitrate. Compression of the original (left) creates only minor errors (right). The strucural similarity [7] between the two images is 99.3%.

3 Results

We performed a number of experiments to evaluate the utility of H.264 for visualization. All tests were run on an Ubuntu 16.04 machine with a NVIDIA GeForce GTX 1080 (Pascal) graphics card.

Ideally one would source the input images from the already-rendered images on the GPU. However, existing remote visualization systems are not architected to easily adhere to this design (compositing often uses images in host memory). Therefore we utilize an interface that accepts data from host memory instead of GPU memory.

We ran a number of benchmarks to examine the performance and compression ratio of our library. We tested a variety of resolutions to elucidate the relationship between the codec's performance. Stream size was linearly correlated with the bitrate used and low bitrates resulted in perceptible artifacts, so all results appear with a 'high' bitrate ($f_m = 4$, assumed 30 FPS) setting.

Secondly, we integrated our library into ParaView and compared its performance with (H.264_hw) and without (H.264_sw) hardware acceleration to built-in compressors utilizing LZ4, SQUIRT and zlib. The existing ParaView compressors were configured for maximum compression as opposed to maximum performance, though we found little performance difference between the two extremes. LZ4 and zlib are lossless; the SQUIRT compression level used is

lossy. The experiments used 1080p RGBA images (the library removes the alpha channel) generated from a rotating volume dataset. We tried both a 'high coherency' mode (rotating by 1.2° per frame) to roughly approximate interactive ParaView use, and a 'low coherency' mode (that utilized a larger 45° rotation between images) corresponding to a more extreme *in situ* case. Results presented are the averages from 300 iterations across five runs.

3.1 Benchmark

Our experimental data shows that the computational complexity is linear with respect to the total number of pixels. Furthermore, the compression time is independent of the image contents. Table 1 details bandwidth requirements and performance for common resolutions.

Table 1: Compression and decompression time per frame through our library. Bitrate reported corresponds to a 'high' bitrate setting. Performance is linear with respect to the image size.

Resolution	Bitrate(mbps)	Compress(ms)	Decompress(ms)
1024x768	6.606	2.8057	2.1170
1280x720	7.741	3.4793	2.4304
1920x1080	17.418	5.4358	4.2876
4096x2160	74.318	21.2504	15.0976

3.2 ParaView Integration

To ensure our library's API could be readily utilized in visualization tools, we integrated it as a compression option in ParaView's client/server mode. ParaView already had abstractions for image compression; since the library is protocol agnostic, it was easy to integrate into ParaView's image streaming framework. To keep our results independent of comparably transient network performance, we launched the client and server on the same machine.

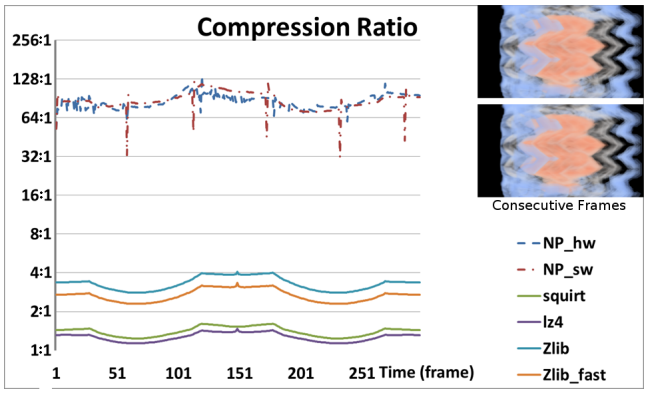

Figure 2: Compression ratio for 'high coherency' case. H.264 achieve compression ratios above 64:1 whereas ParaView's existing compressors maxed out at 4:1 in our experiments.

Figure 2 shows per-frame compression ratios for our 'high coherency' configuration. The occasional dips indicate when we sent a keyframe, currently once every 60 frames.

Figure 3 shows the achievable frames/second for our encoder as well as ParaView compressors. While the best performance is achieved when our library utilizes a GPU, LZ4 and SQUIRT are competitive. The software-backed version of our library is slower than existing ParaView solutions, so environments without a GPU will need to choose between fast encoding (LZ4) or low bandwidth (libx264 through our library).

The average compression ratio (r_c) is shown in Table 2. Our library achieves a 25x better compression rate than its closest competitor, zlib. The payload size is particularly important in visualization because of the synchronous nature of current tools: with

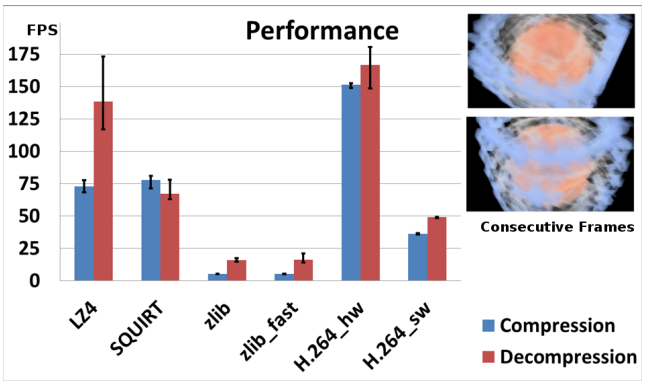

Figure 3: Frame rate of compression and decompression in our 'low coherency' experiments. Hardware-based H.264 backend achieves slightly better performance than competors, while delivering far smaller payloads.

only a frame or two of latency, it is difficult to fully utilize network bandwidth.

Table 2: Compression ratio in 'low coherency' experiments. Our library compresses 25x better than its closest competitor, zlib.

	LZ4	Squirt	Zlib	Zlib_fast	NP_hw	NP_sw
r_c	1.29:1	1.43:1	3.43:1	2.78:1	85:1	88:1

4 Conclusion

Our library is an improvement upon existing visualization systems' remote image delivery mechanism. It saves greater than 25x bandwidth in most cases, with modest improvements to compression/decompression time as well. While we utilize lossy H.264 at present, the induced artifacts are minor.

For future work, we would like to investigate the use of HEVC or H.264's lossless configuration [6]. ParaView's segregation of 'interactive' versus 'still' renders provides a convenient mechanism to switch between a lossy mode for interaction and a lossless mode during pauses. We would also like to investigate adding more asynchronicity in the compression process: the synchronous nature of visualization systems such as ParaView and VisIt makes it difficult to fill a network pipe. This is exacerbated by the high compression ratios that are achieved with H.264. Furthermore, since the encoding hardware is asynchronous with the rest of the GPU, it should be possible to completely hide the entire multi-millisecond encoding latency. Finally, as rendering already uses the GPU, we would like to source the input images directly from the GPU buffer to avoid copying uncompressed images over PCIe.

References

[1] FFmpeg, 2016. [Online; accessed 22-August-2016].

[2] U. Ayachit. *The ParaView Guide: A parallel visualization application*. Kitware, 2015.

[3] H. Childs et al. VisIt: An end-user tool for visualizing and analyzing very large data. In *Proceedings of SciDAC*, 2011.

[4] I. Iszaidy, R. Ahmad, N. Kahar, M. Rahman, and S. N. Yaakob. The investigation of bitrate effects on the video quality of PLECORD system.

[5] NVIDIA. *NVIDIA Video Codec SDK Application Note*, 2016.

[6] G. J. Sullivan, J.-R. Ohm, W.-J. Han, and T. Wiegand. Overview of the high efficiency video coding (HEVC) standard. *IEEE Transactions on circuits and systems for video technology*, 22(12):1649–1668, 2012.

[7] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.

[8] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra. Overview of the H. 264/AVC video coding standard. *IEEE Transactions on circuits and systems for video technology*, 13(7):560–576, 2003.