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

Category: Research

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 140:1. 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 42x bandwidth reduction compared to existing image compression.

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 [7]. The standard provides high compression ratios but this ability comes at significant computational expense. Modern NVIDIA hardware provides a hardware-accelerate H.264 encoder [4]. The hardware enables real-time encoding with minimal overhead. Support for the encoder has been integrated into popular multimedia frameworks [1].

2 Design

Our library can utilize NVIDIA's hardware encoder or the libx264-based software encoder. We specifically designed our library to abstract away most video compression configuration. A detailed discussion of H.264 parameters is covered in [7]. We have configured the encoder to use settings reasonable for visualization. For example, video compression 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 tools such as ParaView [2] and VisIt use image compressors today.

Our library handles only image compression and decompression. It leaves the network communication to the user for ease of integration.

2.1 Performance

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

2.1.1 Compression Ratio

Compression ratio is determined by the average bitrate parameter. A guideline for bitrate setting is the Kush Gauge [3]:

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

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 used in the library uses $f_m = 4$ and an estimated 24 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.

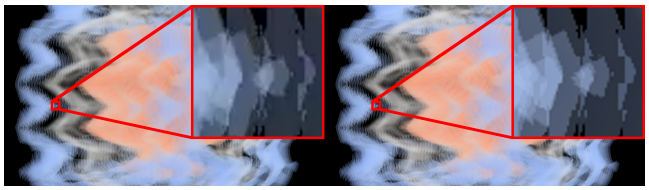

Figure 1: 1080p image compressed using bitrate calculated with $f_m = 4$. The image on the left is the original and the image on the right is the recovered. Average structural similarity (SSIM [6]) between the images is 98%.

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 remoter 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. We also tried a variety of bitrates but do not highlight the result here because the results are what one might expect: lower bitrates compress to smaller payloads.

Secondly, we integrated our library into ParaView and compared its performance with and without hardware acceleration to built-in compressors utilizing lz4, squirt and Zlib. Our library used a high bitrate setting where $f_m = 4$ for best image quality. The existing ParaView compressors were configured for maximum compression as opposed to maximum performance, though we found little performance difference between the two extremes. 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 are the average 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: Average compression and decompression time per frame using our library. Performance is essentially linear with respect to the size of the image.

Resolution	Bitrate(mbps)	Compress(ms)	Decompress(ms)
1024x768	1.651	2.8057	2.1170
1280x720	1.935	3.4793	2.4304
1920x1080	4.355	5.4358	4.2876
4096x2160	18.579	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 compressor for ParaView's client/server mode. The library handles encoding on the server side and returns an opaque pointer that ParaView sends over the network to the client. The same data is given to the decompressor, which then produces an RGBA image with faux alpha channel of value 1.0. To keep our results independent of comparably transient network performance, we launched the client and the server on the same machine.

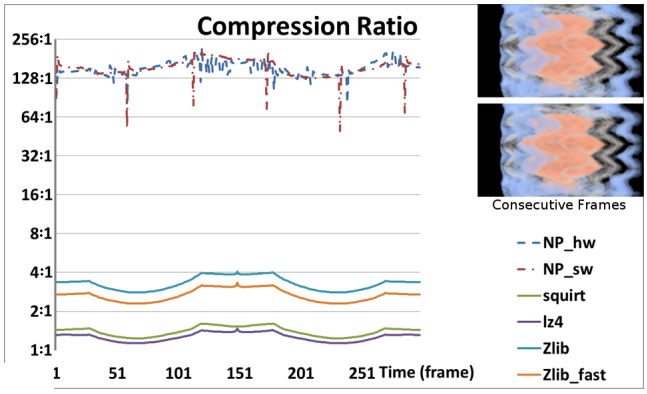

Figure 2: Compression ratio for 'high coherency' case. H.264 achieves compression ratios above 128:1 whereas ParaView's existing compressors maxed out at 3.4:1 in our experiments.

Figure 2 shows per-frame compression ratios for our 'high coherency' configuration. The occasional dips represent when we send a keyframe, currently once every 60 frames. Even if the library were to use a keyframe for every frame, our library achieves compression ratios 16x better than existing approaches.

Figure 3 shows the achievable frames/second for our encoder as well as ParaView compressors. Zlib is not competitive for image compression. 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 compression/decompression or low bandwidth.

The average compression ratio r_c is shown in Table 2. Our library achieves a 42x 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 only a frame or two of latency, it is difficult to fully utilize network bandwidth.

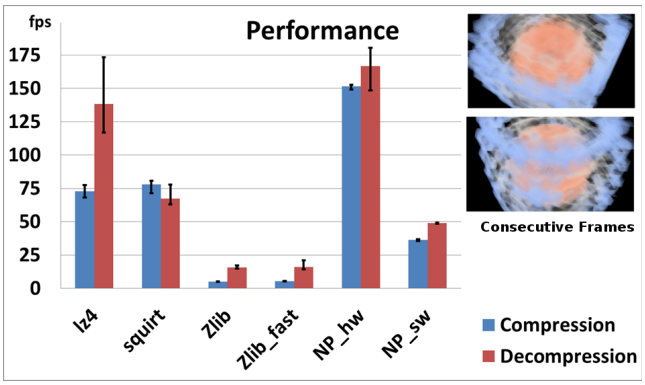

Figure 3: Frame rate of compression and decompression in our 'low coherency' experiments. Our libary's hardware-based backend achieves slightly better performance than competors, while delivering far smaller payloads.

Table 2: Compression ratio in 'low coherency' experiments. Our library has 42x better space saving than the closest compressor, Zlib.

	LZ4	Squirt	Zlib	Zlib_fast	NP_hw	NP_sw
r_c	1.29:1	1.43:1	3.43:1	2.78:1	144:1	152:1

4 Conclusion

Our library is an improvement upon existing visualization systems' remote image delivery mechanism. It saves greater than 42x 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 H.264's lossles configuration [5]. With ParaView's decomposition of 'interactive' versus 'still' renders, one could envision using the lossy system during interaction with 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 images over PCIe in uncompressed form.

References

[1] Ffmpeg, 2016. [Online; accessed 22-August-2016].
[2] A. Henderson, J. Ahrens, C. Law, et al. *The ParaView Guide*. Kitware Clifton Park, NY, 2004.
[3] I. Iszaidy, R. Ahmad, N. Kahar, M. Rahman, and S. N. Yaakob. The investigation of bitrate effects on the video quality of PLECORD system.
[4] NVIDIA. *NVIDIA Video Codec SDK Application Note*, 2016.
[5] 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.
[6] 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.
[7] 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.