Eurographics Symposium on Parallel Graphics and Visualization (2018)
H. Childs, F. Cucchietti (Editors)

Hardware-Accelerated Multi-Tile Streaming for Realtime Remote Visualization

T. Biedert[1], P. Messmer[2], T. Fogal[2] and C. Garth[1]

[1]Technische Universität Kaiserslautern, Germany
[2]NVIDIA Corporation

Abstract

The physical separation of compute resource and end user is one of the core challenges in HPC visualization. While GPUs are routinely used in remote rendering, a heretofore unexplored aspect is these GPUs' special purpose video en-/decoding hardware that can be used to solve the large-scale remoting challenge. The high performance and substantial bandwidth savings offered by such hardware enable a novel approach to the problems inherent in remote rendering, with impact on the workflows and visualization scenarios available. Using more tiles than previously thought reasonable, we demonstrate a distributed, low-latency multi-tile streaming system that can sustain stable 80 Hz when streaming up to 256 synchronized 3840x2160 tiles and can achieve 365 Hz at 3840x2160 for sort-first compositing over the internet.

Categories and Subject Descriptors (according to ACM CCS): I.3.2 [Computer Graphics]: Graphics Systems—Distributed/network graphics

1. Introduction

The growing use of distributed computing in computational sciences has put increased pressure on visualization and analysis techniques. A core challenge of HPC visualization is the physical separation of visualization resources and end-users. Furthermore, with increasing dataset sizes, in-situ scenarios, and complex visualization algorithms, transfer to a separate visualization system becomes impractical. Modest demand for interactivity, low screen resolutions and user bases on relatively high-speed connections made frame based compression sufficient to provide a workable remote visualization experience. With novel interactive workflows, commodity high-resolution monitors, complex rendering algorithms, latency sensitive display technologies and globally distributed user bases, new approaches to solve the remoting challenge are required.

The wide availability of GPUs in current and future generation HPC systems allows not only to leverage the GPUs' rendering capabilities, but also their special purpose video en-/decoding hardware: to both weakly scale and render significantly more pixels without requiring large amounts of streaming bandwidth, and to strongly scale the rendering tasks and reduce the overall latency.

Being able to drive high-resolution displays directly from a remote supercomputer opens up novel use cases. In particular, it enables cheaper infrastructure at the client's side, as all the heavy lifting is done on the server side. It also allows the visualization and rendering system to scale with the scale of the simulation, rather than having to scale a separate system for visualization of large-scale data sets. As will be shown in Section 2, driving remote tiled displays is nothing new. However, contemporary resolutions of at least 4K or more per display at interactive frame rates far exceed the capabilities of previous approaches.

Strong scaling the rendering and delivery task enables novel interactive uses of HPC systems. Splitting the rendering load enables expensive rendering solutions at interactive frame rates. This can improve perception when visualizing a high-resolution simulation's results. In addition, the renewed interest in virtual reality with head mounted displays begs the question for streaming directly from the HPC system.

In this comprehensive case study, we demonstrate the impact of video compression and multi-tile streaming for low-latency distributed remote rendering at interactive frame rates. Using comprehensive benchmarks we demonstrate the practicability of this approach and the impact on possible workflows and visualization scenarios. With these investigations, we address several basic questions:

- Is it feasible to stream content directly from a supercomputer to remote large-scale tiled displays at sufficiently high frame rates (e.g. interactively)? Which latency and bandwidth requirements does this entail, and how do they correlate to image contents?
- How does hardware-accelerated progressive video compression compare to conventional CPU-based compressors applied to individual frames?
- What frame rates can be delivered to a remote end-user by strong scaling the rendering and delivery task, i.e., using video hardware for direct-send sort-first compositing?

DOI: 10.2312/pgv.20181093

- Could this even be used for latency-sensitive environments such as VR?

It is not our intention to validate remote visualization as a general approach, but rather to highlight possible process improvements and streamline the end-user experience through the use of hardware-accelerated video compression.

The outline of the paper is as follows. In Section 2 we provide some background on related activities in this field. Sections 3 and 4 show the general setup of our multi-tile streaming approach using hardware-accelerated video compression. Section 5 demonstrates the achievable frame rates that our system provides in various configurations. Possible opportunities for enhancements are discussed in Section 6.

2. Related Work

As observed above, remote visualization techniques have by necessity been in practical use since the advent of computational sciences. Fundamentally, this is a consequence of the fact that end-user analysis resources cannot be expected to scale with data-production resources, severely limiting visualization capability for state-of-the-art problems.

A substantial body of previous work focuses on using dedicated computational resources (such as a visualization server or visualization clusters) where images are generated and transmitted to commodity hardware such as a PC or mobile device. Engel et al. [ESE00] observe the necessity to access remotely available high-performance visualization clusters, and provide a web-based interface to the remote servers. Lamberti and Sanna [LS07] and Noguera and Jiminez [NJ16] examine the challenges and opportunities of using low-powered, mobile viewing devices, which naturally integrate into this setting. Several general purpose frameworks have been described in the literature, e.g. SAGE2 [MAN*14] and Equalizer [EMP09], but standard visualization tools such as e.g. VisIt [CBW*12] and ParaView [AGL05, HOPU08] also support corresponding modes of operation. As visualization is typically used in an interactive setting, latency is important. Stegmaier et al. [SDWE03] describe improvements to naive image streaming aimed at improving poor interactive performance due to high latency. Most notably, they find image compression to be beneficial in reducing latency. Focusing especially on compression of visualization images, specific solutions can be developed for particular algorithms. For example, Cui et al. addressed latency by transmitting annotated depth images from which different viewpoints can be reconstructed without data retransmission [CMP14]. Similarly, Lalgudi et al. [LMB*09] exploit view coherency to derive effective compression for volume rendering. Similar techniques have also been used in other applications, e.g. remote gaming [FE10]. While both these works exemplify non-standard compression schemes for specific visualization techniques, it is difficult to extend them to a general setting.

The present paper is inspired by the work of Jiang et al. [JFWM16], who describe a lightweight general purpose image compression library that utilizes the video compression hardware on NVIDIA GPUs [Nvi]. They use temporal and spatial image coherence during compression to obtain 25x improved compression ratios at reduced latency, and demonstrated the benefits of their approach through integration with ParaView for single-tile streaming. The compression scheme is based on the H.264 standard [WSBL03], and is thus in principle a lossy approach. However, this is acceptable in practice as evidenced by the nowadays ubiquitous use of such codecs in the entertainment industry. While to the best of our knowledge the specialized video hardware available in modern GPUs is still mostly unused in high performance visualization, an interesting alternative application was recently presented by Leaf et al. [LMM17], who have demonstrated in-situ compression of floating-point volume data using hardware encoders.

In this paper, we take the hardware-accelerated video streaming approach to the extreme by applying it to diverse large-scale multi-tile scientific visualization scenarios. Although demonstrated on vendor hardware, the technique is in principle vendor-independent and could be extended to utilize corresponding hardware in Intel [Int] and AMD GPUs [Adv], which provide similar codec capabilities. Furthermore, given modern hardware support, it appears feasible to adopt the HEVC standard [SOHW12] that improves image quality while retaining strong compression. In this work, we consider both H.264 and HEVC codecs, but focus on the still more ubiquitous H.264 variant, given its wide availability of hardware implementations.

A necessity of using distributed resources to compute visualization images in a remote scenario is compositing: to send a single image to the end-user, partial results computed on different nodes must first be composited onto a single node. This implies that the image data incurs latency twice – first when sent to the compositing node, and a second time when sent to the client. Since compositing speed and latency can dramatically limit end-to-end performance of remote visualization systems, a significant body of prior work has therefore investigated how to in particular conduct the compositing phase with minimum latency. Corresponding strategies broadly fall into several classes. In direct send compositing [EP07, SML*03], render nodes directly send pixel data to the compositing node. The binary swap [MPHK94] and radix-k [YWM08, PGR*09, KPH*10] strategies improve on this by intelligently spreading intermediate compositing operations across many nodes and exchanging pixel data using optimized communication schemes. The performance and scalability of corresponding implementations are demonstrated e.g. in the IceT framework [MKPH11].

In constrast, in this paper, we demonstrate that it is feasible and advantageous to perform direct-send compositing on the client and display tiles at very high frame rates. This enables a system design in which render nodes directly send to the end-user client. We set out to demonstrate that with vastly improved (de-)compression, even a large number of render nodes do not overwhelm the recipient of their images.

3. Multi-Tile Streaming

Hardware-accelerated multi-tile streaming is a promising approach to make the distributed rendering capabilities of the GPUs within a remote HPC system directly accessible to visualization systems, such as tiled displays, CAVEs [FNT*13], workstations, virtual reality (VR) headsets, thin clients or mobile devices.

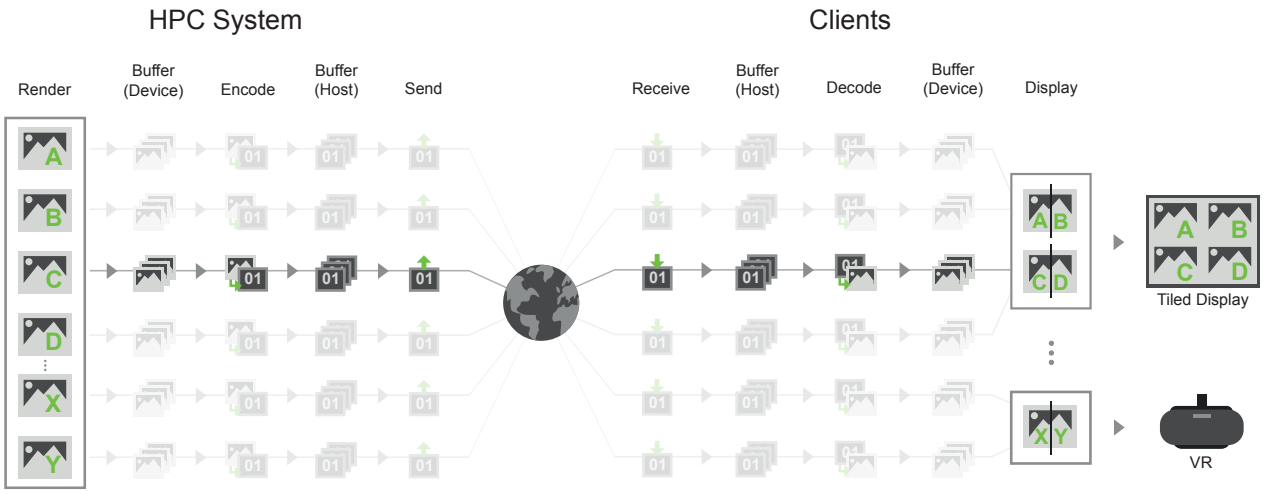


Figure 1: *Conceptual overview of the multi-tile streaming approach using asynchronous pipelines. Several tiles are streamed directly from the compute nodes' GPUs in the HPC system (left) to muliple client GPUs for display (right). Each pipeline consists of individual threads for hardware-accelerated en-/decoding and communication. The threads within a pipeline interact through buffers that are strategically placed on device or host memory, thereyby minimizing costly PCIe bus transfers. Frames rendered by the source GPU are immediately encoded and sent to the destination GPU for direct decoding to the display engine.*

A multitude of contemporary GPUs is equipped with dedicated hardware units that enable low latencies and interactive frame rates for high-resolution remote streaming. For instance, NVIDIA GPUs contain one or more hardware-based decoders and encoders (separate from the CUDA cores) that provide hardware-accelerated video decoding and encoding for several popular codecs [Nvi]. With decoding/encoding offloaded, the rendering engine and the CPU are free for other operations, such as visualization, computation, and data management. In contrast to the stand-alone compression of individual frames based on compression algorithms such as LZ4, Squirt (run-length encoding) or Zlib that have been used in previous remote visualization applications [AGL05], progressive video-based encoders such as H.264 or HEVC provide better compression ratios by exploiting not only the spatial coherence within but also the temporal coherence between frames in scientific visualization applications. Using the hardware-accelerated codec units of NVIDIA GPUs, we have implemented a prototypical client/server library providing concurrent streaming pipelines between source and target applications.

Conceptually, a pipeline connects two GPUs and streams (parts of) a source image from the rendering framebuffer to a remote destination for display. Figure 1 illustrates multiple concurrent pipelines connecting rendering nodes from a remote HPC cluster and local visualization systems driving tiled displays or virtual reality devices. A streaming pipeline essentially consists of three stages for encoding, transmission and decoding. Technically, each pipeline is implemented as a series of parallel tasks connected through thread-safe queues. This ensures the high degree of asynchronicity within each pipeline that is crucial to achieve high throughput at low latencies. All involved hardware units, i.e., rendering/display (GPU), encoding/decoding (GPU) and network transmission (CPU), work concurrently. Depending on the stage, buffers are strategically placed on either device or host memory to minimize host bus transfers. Notably, host memory transfers can be completely avoided by using DMA transfers (e.g. GPUDirect RDMA on recent NVIDIA GPUs). This technique requires certain hardware configurations and is out of the scope of this work.

The following section describes all stages in the life of a streamed frame (or tile) through the pipeline from source to destination. After rendering the main render thread accesses the framebuffer and pushes the requested tile into the device-memory encode buffer. The encode thread pulls from its input buffer and forwards the raw image data into the hardware encoder. The resulting compressed bitstream is enqueued into a host-memory send queue. Compressed data in the send buffer is continuously transmitted onto the network by a separate thread. On the client side, a pipeline consists of a receive thread and a decode thread. As the receive thread receives compressed frames from its network socket, it pushes them into the host-memory decode queue. The subsequent decode thread pulls compressed data from its input queue and performs hardware-accelerated decoding. The resulting raw tiles are placed into a device-memory display queue. The main display thread of the client application pulls from the display queue to assemble the complete output image. A single server or client process can contain one or more concurrent pipelines. For instance, a server application could split the framebuffer into two tiles and use two concurrent pipelines to better utilize the available hardware encoding units. A similar approach can be employed at client-side, either to decode and display full frames to distinct monitors, or composite partial tiles of a single display.

For synchronization purposes the frame number of each tile is passed through the pipeline, in addition to latency statistics. While synchronization between server processes is optional depending on the context (e.g., typically already provided by the underlying

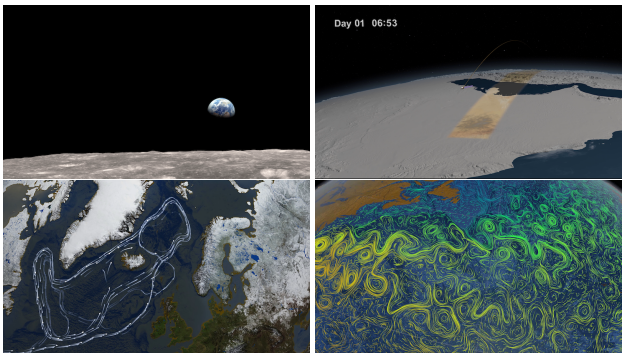


Figure 2: *NASA's* Synthesis 4K *video footage used for streaming benchmarks, representing typical content of scientific visualizations at various image complexities:* Space *(low),* Orbit *(medium),* Ice *(high),* Streamlines *(extreme).*

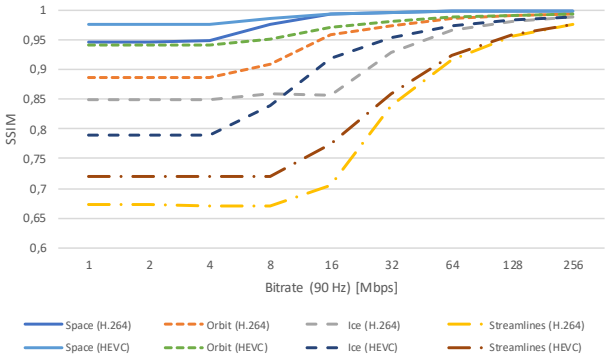


Figure 3: *Structural similarity (SSIM) index of the four test scenes at different encoder bitrates for H.264 and HEVC.*

simulation in case of in-situ scenarios), synchronization at client-side is obligatory to ensure consistent display of all contributing tiles. In our benchmarks we perform synchronization across processes at both server-side (before frame grab) and client-side (before display). Note that by synchronization across processes we only refer to processes contributing to a single output device, e.g., all pipelines to a tiled display wall in case of weak scaling or all pipelines to a single display in case of strong scaling. There is no need for precise per-frame synchronization between seperate isolated output devices.

4. Implementation

We have implemented the multi-tile streaming approach based on concurrent asynchronous pipelines as a server-client library that can be combined with arbitrary rendering applications. Both server and client internally use MPI for synchronization across processors and support using multiple GPUs for encoding and decoding, respectively. Newly created pipeline instances are assigned to the node's available GPUs in a round robin way.

The streaming server creates a standard TCP socket and listens for incoming connections. The use of the TCP protocol instead of UDP relieves from additional frame/packet loss handling, while still enabling high performance as demonstrated in Section 5. For each connection, a server pipeline instance is created as described in Section 3. The rendering application can retrieve the bounding rectangles of the required tiles from the server instance to optimize the rendering process by restricting rasterization or ray casting to the relevant regions. OpenGL-based renderers require the use of the CUDA/GL interoperability APIs to copy (parts of) the frame buffer to device memory, either via PBO or framebuffer texture access. OptiX-based renderers output directly to CUDA device memory. Pre-rendered frames as used in our synthetic benchmarks are placed in device memory.

Analogously, the streaming client can connect to one or multiple servers simultaneously and request specific (sub) tiles. For each connection, the application creates a separate client pipeline instance. Similar to an OpenGL-based server, this step utilizes CUDA/GL interop to copy each tile into the mapped color texture of a framebuffer object or a target PBO. Using the frame number provided with each tile, initial client-side synchronization at the beginning of the streaming process is ensured by dropping outdated tiles based on the maximum frame number until all clients involved are synchronized. Since frame loss is prevented by TCP, display of subsequent frames is easily synchronized using barriers. In case of multiple distributed client processes, e.g., multiple nodes driving a tiled display wall, an MPI-based all-reduction is used to collectively determine the maximum frame number all clients must synchronize to.

5. Results

We have conducted comprehensive benchmarks to demonstrate the practicality of the presented multi-tile streaming approach and its impact on possible workflows and visualization scenarios.

As a stand-in for high-resolution scientific visualization renderings, we have selected the *Synthesis 4K* footage from NASA. Figure 2 illustrates scenes of varying complexity we have streamed from this source. Pre-rendered frames are extracted from the video, copied into device memory, and pushed into the multi-tile streaming server, as described in Sections 3 and 4. We have restricted our benchmarks to two commonly used resolutions: 3840x2160 and 2160x1200. The former is usually referred to as 4K and the latter is a typical resolution for current-generation VR devices, such as Oculus Rift and HTC Vive. We have used YUV 4:2:0 color format with a color depth of 8 bits per channel. The required CUDA-based conversion kernels between RGB and YUV have been profiled to be of negligible impact.

All benchmarks have been conducted on the Piz Daint supercomputer at the Swiss National Supercomputing Centre (CSCS). In our experiments we have used up to 512 GPU nodes for simultaneous multi-tile encoding and decoding. Additionally, at client-side we have benchmarked a multitude of scenarios on three locations for multi-tile decoding:

- Site A (ping 5 ms, 1x NVIDIA Quadro GP100)
- Site B (ping 25 ms, 2x NVIDIA Quadro GP100)
- Site C (ping 200-1000 ms, 4x NVIDIA Tesla P100)

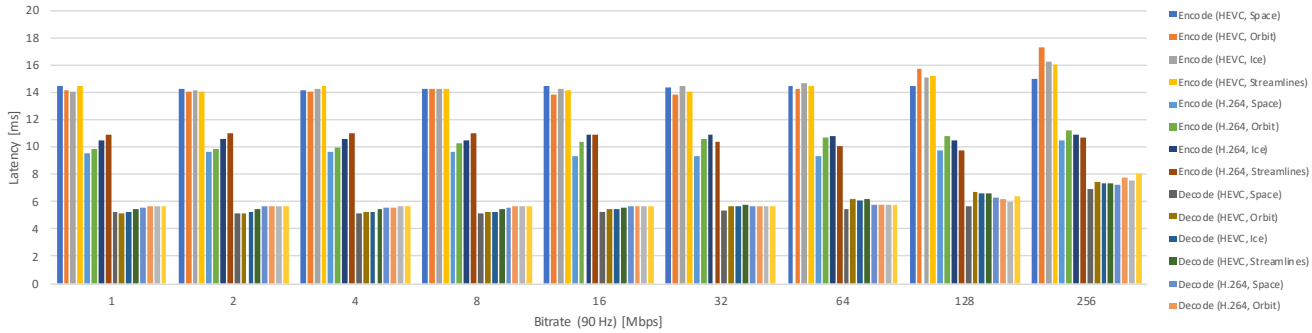


Figure 4: *Comparison of en-/decode latencies for the four test scenes at different bitrates for H.264 and HEVC at 4K resolution.*

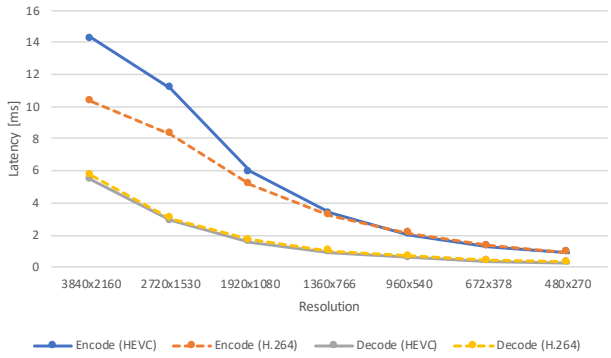


Figure 5: *Comparison of en-/decode latencies (averaged across test scenes) at different resolutions for H.264 and HEVC. Default bitrate for 4K at 90 Hz is 32/16 Mbps (H.264/HEVC), downscaled proportionally with pixel count.*

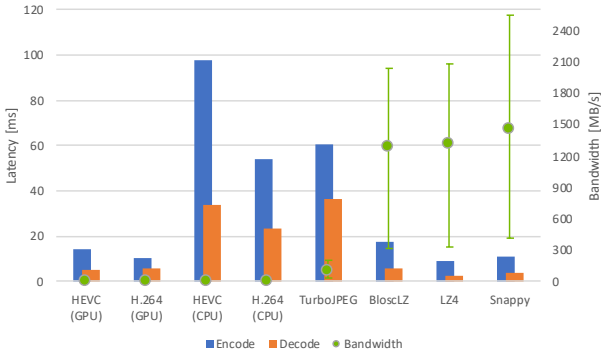


Figure 6: *Comparison of average en-/decode latencies and required bandwidths at 90 Hz for 4K using different GPU and CPU codecs. Bandwidth shows min/max ranges across test cases. H.264/HEVC configured to 32/16 Mbps at 90 Hz, JPEG quality at 80, others at maximum compression level.*

Both the NVIDIA Tesla P100 and the Quadro GP100 are from the Pascal architecture family, featuring two independent hardware units for both encoding and decoding. In all benchmarks the hardware encoders were configured to use the *low latency - high quality* preset for both H.264 and HEVC. The target frame rate was set to 90 Hz, which always has to be considered in conjunction with a specific bitrate.

5.1. Codec Performance

The encoding quality of H.264 and HEVC can be configured to aim for a constant target bitrate. Intuitively, bandwidth requirements increase with frame complexity to sufficiently represent important details. We have used the structural similarity index (SSIM) [WBSS04] to measure and compare the encoding quality of H.264 and HEVC. Figure 3 shows the SSIM for the four test scenes at different bitrate settings. Higher bitrates improve reconstruction detail, where more complex scenes require higher bitrate settings to look acceptable. In general, HEVC outperforms H.264 by providing higher quality at the same bitrate. However, interestingly the high complexity *Ice* scene is slightly better reconstructed using H.264 for lower bitrates. Based on our visual experiments, we have chosen default bitrates of 32 Mbps for H.264 and 16 Mbps for HEVC in all following streaming benchmarks. If not specified otherwise, benchmarks are based on the *Ice* scene as a representative for high complexity.

The impact of image complexity on en-/decoding latencies is close to negligible for both H.264 and HEVC as illustrated in Figure 4. Also, the bitrate level only starts to affect latencies slightly at very high settings with increased memory bus pressure. In general, H.264 requires approx. 10 ms for encoding a full 4K frame on our test hardware, whereas HEVC needs 14 ms at the same bitrate. Decoding latency is similar for both codecs at 5-6 ms. When considering smaller tiles, overall latencies decrease as expected and the gap between encoding latencies narrows, as shown in Figure 5.

We set out to demonstrate that hardware-accelerated progressive video encoding is superior to conventional single frame compression approaches for realtime streaming scenarios. Figure 6 shows the latency and required bandwidths at 4K resolution and a fixed frame rate of 90 Hz for several popular compression algorithms in comparison to the GPU. Both latency and bandwidth depict averages measured across the four test scenes. While the impact of

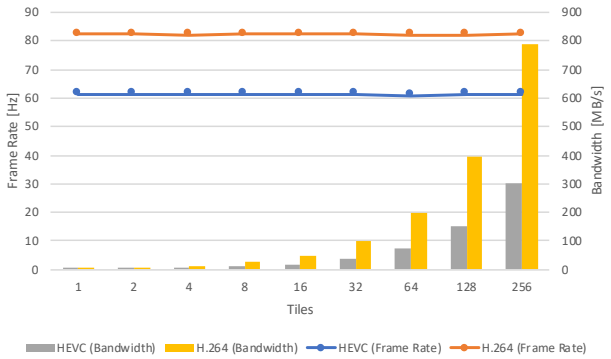


Figure 7: *Weak scaling (N:N) within Piz Daint: Client-side frame rates and bandwidths for up to 256 concurrent full 4K streams of the* Ice *video at 32/16 Mbps (H.264/HEVC).*

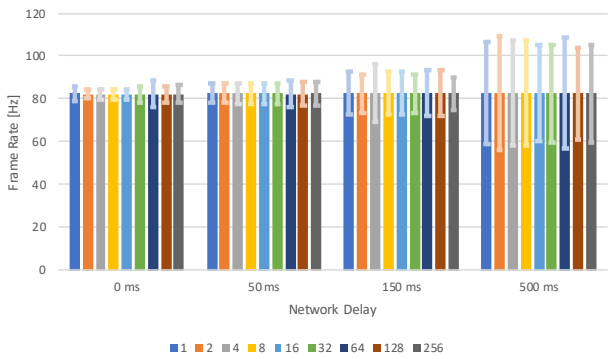


Figure 8: *Weak scaling (N:N) within Piz Daint: Mean frame rates and min/max ranges for different simulated network delays (+10% jitter), streaming up to 256 concurrent full 4K tiles of* Ice *sequence using H.264 at 32 Mbps.*

image complexity is negligible for latency, high complexity content requires increased amounts of bandwidth. This is indicated by the error bars depicting the bandwidth range for the low and extreme complexity scenes. All CPU-based compressor benchmarks have been conducted on an Intel Core i7-6850K CPU at 3.6 GHz with hyper threading. TurboJPEG has been configured to a quality of 80, and the general purpose compressors have been set to use multi-threading and the maximum compression level possible. When considering latencies, GPU-accelerated video codecs are faster than their CPU-variant (using FFmpeg) by an order of magnitude. Interestingly, modern CPU-based compressors such as LZ4 or Snappy are comparable in latency to GPU video encoding, even with the additional PCIe bus transfer of the rendered frame between device and host memory included. The compressors Zlib and Zstd are not shown due to impractically large encoding latencies. In contrast, when considering required streaming bandwidth, progressive video encoding starts to shine due to its consideration of temporal coherence between frame. While for instance HEVC at 16 Mbps needs approximately 2 MB/s to stream the high complexity *Ice* scene at good quality, LZ4 requires an excessive bandwidth of more than 2 GB/s. The latency of TurboJPEG is comparable to the CPU-based video codecs, however its required bandwidth is up to two orders of magnitude larger.

5.2. Full Tiles Streaming

Being able to drive tiled displays directly from a remote supercomputer enables cheaper infrastructure at the client side, as all the heavy lifting is done on the server side, and allows the rendering system to scale with the simulation, rather than having to scale a separate system for visualization.

This begs the question of how many simultaneous full resolution streams can be handled by GPUs and network to fulfill given requirements such as interactivity. For instance, how many GPUs are required to drive a particular tiled display or CAVE configuration at a certain target frame rate? To investigate such questions, we have conducted synthentic benchmarks for streaming within the Piz Daint supercomputer: up to 256 synchronized server nodes stream 4K frames from the complex *Ice* scene to up to 256 synchronized client nodes, which virtually display into an offscreen framebuffer via EGL. Figure 7 shows the mean frame rates and accumulated bandwidths for both H.264 and HEVC. Even for 256 concurrent streaming pipelines, the system manages to sustain stable frame rates of 80 Hz and 60 Hz, respectively. The required bandwidths increase linearly with node count. While this setup may appear overly optimistic at first, we expect this to be a plausible environment for possible improvements towards cheaper infrastructures: provided a sufficiently large low-latency link, a large-scale tiled display wall could be directly driven by super computer located either at the same site or possibly even a remote facility.

By delaying the sending of frames for a predefined amount of simulated network latency plus a random jitter within a given range, we have measured the impact of network latency on the multi-tile setup. A sufficiently large buffer was used to correctly simulate in-flight network packets without blocking the encoding pipeline. As each frame is assigned a random amount of delay within the specified window, frame burst patterns start to emerge, thereby affecting frame rate stability. Figure 8 shows the mean, minimum and maximum frame rates for different network delays with 10% jitter.

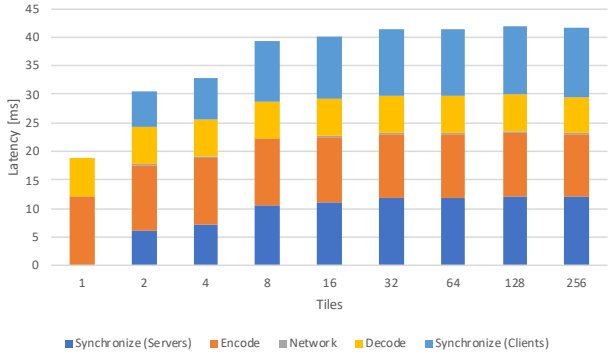


Figure 9: *Weak scaling (N:N) within Piz Daint: Pipeline latencies for full 4K streams of* Ice *using H.264 (32 Mbps).*

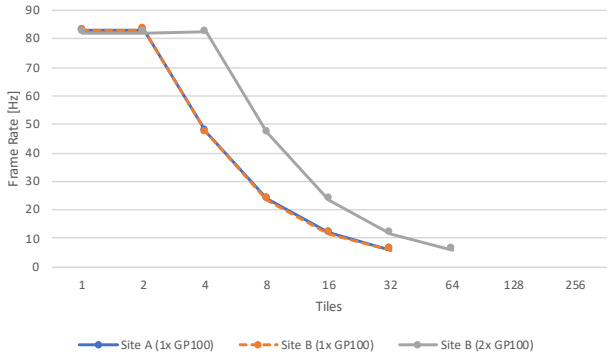


Figure 10: *Weak scaling (N:1): Client-side achieved frame rate for streaming full 4K frames of* Ice *sequence to Site A (blue) and Site B (orange, gray) using H.264 at 32 Mbps.*

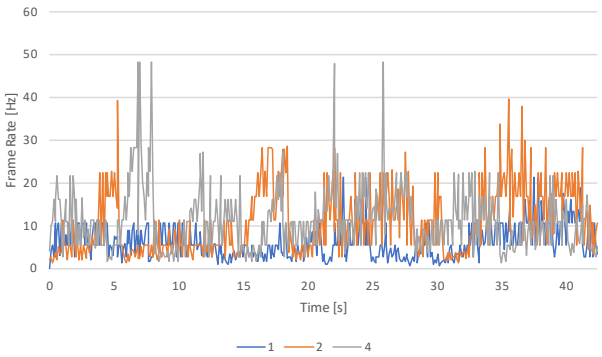


Figure 11: *Weak scaling (N:1): Streaming up to 4 full 4K frames of* Ice *sequence across Atlantic Ocean to Site C using HEVC at 16 Mbps. Frame rate sampled at 10 Hz.*

While frame rate stability decreases with simulated network latency due to increased amounts of jitter, the overall frame rate remains in interactive magnitudes.

However, not only throughput, i.e., frame rate, but also the complete pipeline latency for a single frame from source to destination has to be considered for interactivity. Figure 9 illustrates the mean latencies of each pipeline stage across nodes without simulated network delay. Besides the expected encoding and decoding latencies, synchronization across nodes at both server and client side contributes a considerable amount of additional latency, which increases with node count. With overall latencies of up to approx. 40 ms plus potential network delay this can still be considered highly interactive. While server-side synchronization can be eliminated depending on the scenario, client-side synchronization is obligatory for multi-node clients, e.g. for coherent display and buffer swap on a tiled display wall.

Since modern GPUs can drive multiple screens at 4K or higher resolutions, an interesting question is how many full streams can be handled by a single device at which frame rates, esp. when aiming for cheaper infrastructure. Figure 10 shows the achievable frame rates at client side for 4K resolution when streaming multiple concurrent full size tiles to Site A and Site B. A single NVIDIA Quadro GP100 card can decode up to two full 4K streams at approximately 80 Hz, using its two independent hardware decoding units. Given that this card features four display connectors, it is interesting to see that it can still handle four full 4K streams at 50 Hz. This performance capability enables high-resolution tiled displays with high tile counts driven by only a few GPUs for multi-tile decoding at interactive frame rates. As indicated by the shifted curve when using two GP100s for decoding, using hardware-accelerated multi-tile streaming is a scalable approach with the number of devices and is mainly limited by the available network bandwidth. As expected, the difference of network latency between Site A and Site B seems to be of negligible impact on frame rate.

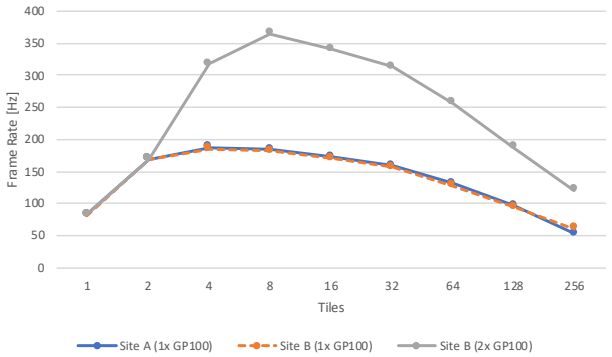
To demonstrate the severe effect of a highly erratic link on frame rate, we have benchmarked multiple 4K streams from Piz Daint to Site C, California, USA. Figure 11 shows the effective client-side frame rate sampled at 10 Hz over 45 seconds. The presented graphs exemplify the unpredictability in such a setup, where due to frequent extreme latency spikes the overall frame rate drops significantly.

5.3. Strong Scaling / Sort-First Compositing

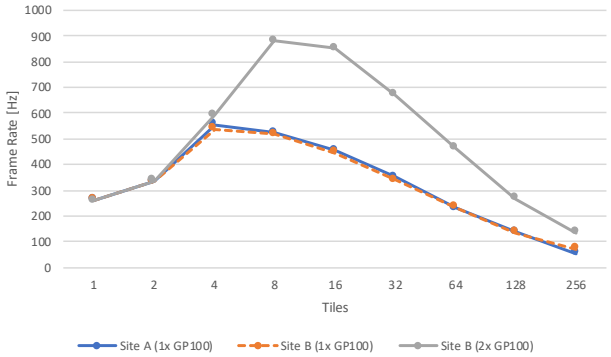
Strong scaling the remote rendering and delivery task opens up novel interactive uses of HPC systems. A plethora of server nodes enables expensive rendering solutions at interactive frame rates and low latency. This can be in support of improved perception in challenging visualization tasks, for instance by computing sophisticated global illumination models for complex geometries. In addition, the renewed interest in virtual reality (VR) gives rise to the question if streaming directly from the HPC system is a viable approach for time-critical VR scenarios. We have conducted several experiments to quantify the strong-scaled multi-tile streaming performance using up to 256 Piz Daint nodes streaming to a single client which performs sort-first compositing.

Figure 12 shows the achievable frame rates using strong scaling for 4K and VR resolution. Considering a single Quadro GP100 for decoding, frame rates up to 190 Hz are possible for 4K when utilizing 4 servers for encoding (Figure 12a). Full 4K encoding on a single node is limited by the encoder at approx. 80 Hz. With increased tile counts the maximum achievable frame rate slowly decreases due to increasing overhead from decode session multiplexing. The curves for Site A and Site B are almost identical for the 1x GP100 case, confirming that 20 ms of network latency has little effect on achievable throughput. Adding a second GP100 for decoding strongly amplifies the decoding capabilities at client side, enabling frame rates up to 365 Hz for 4K when utilizing 8 servers for encoding. Evidently, also the second card is subject to overhead at high tile counts. Yet, strong scaled multi-tile 4K streaming from 256 servers is still possible at 120 Hz using two GP100s for decoding.

The graphs look similar for the reduced VR resolution (Figure 12b), generally showing much higher frame rates, which is expected due to reduced en-/decoding latency. A full 2160x1200 tile can be streamed from a single server node at 260 Hz, whereas



(a) *4K (3840x2160)*



(b) *VR (2160x1200)*

Figure 12: *Strong scaling (N:1): Client-side frame rate for streaming* Ice *sequence at 4K (left panel) and VR resolution (right panel) from up to 256 servers to Site A (blue), Site B (orange) and Site B with two GPUs per tile (gray) using H.264 (32 Mbps).*

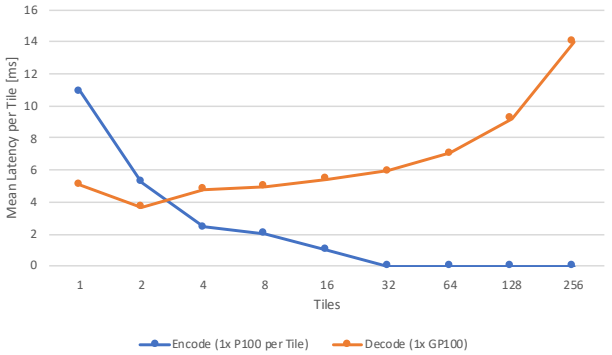


Figure 13: *Mean latency for encoding a 4K frame (blue) on the server side (strong-scaled, N:1) and decoding all tiles on the client side (Site B, orange) using H.264 (32 Mbps).*

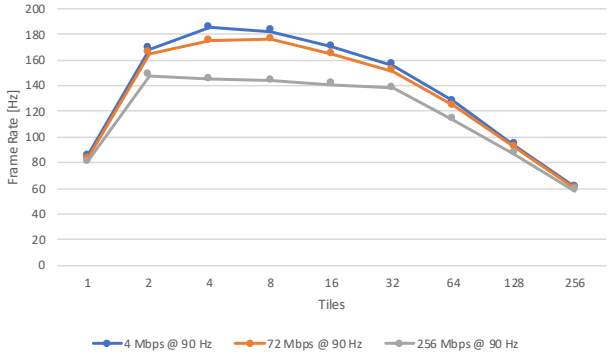


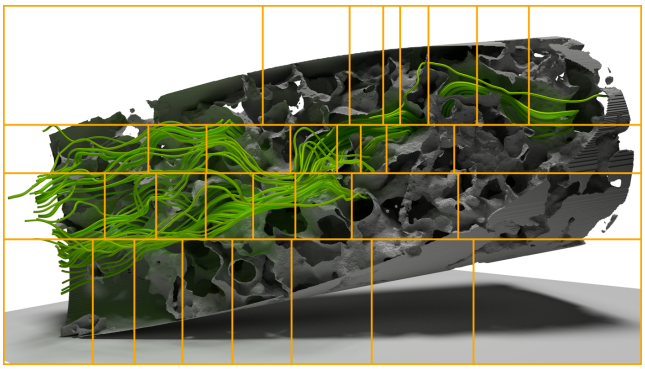
Figure 14: *Client-side frame rate for strong-scaled streaming of* Ice *4K frames from up to 256 servers at different H.264 bitrates. Highest setting is limited by link bandwidth (gray).*

streaming multiple tiles peaks at 520 Hz in the 1x GP100 case. Strongly scaling to 8 server nodes can achieve frame rates up to 900 Hz in the 2x GP100 case. Despite the decoding bottleneck at high tile counts, utilizing two GPUs for decoding of 256 tiles we can still achieve 135 Hz, comfortably above the suggested 90 Hz threshold for common VR applications. Again, the curves for Site A and Site B are almost identical.

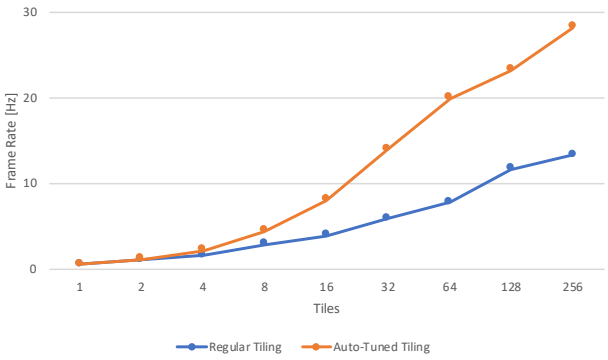
Figure 13 illustrates the strong scaled en-/decoding latencies for 4K frames in the 1x GP100 case and helps to further understand curve progression in Figure 12. Clearly, full 4K frame encoding (approx. 11 ms) is more than twice as costly as decoding (approx. 5 ms). In this context, note that the NVIDIA Tesla P100 and Quadro GP100 cards are very similar with respect to their en-/decoding hardware units. In the strong scaling scenario, encoding latencies drop as expected with increased tile counts due to the reduced pixel count per tile. Note that each tile is encoded by a different GPU in this case. At the same time, the mean decode latency per tile increases due to the high number of simultaneous decode sessions on the single decoding GPU. Based on Figure 13, the frame rate peak at 4 tiles in Figure 12 can thus be interpreted as the setup with minimal decode overhead where the frame rate is not limited by the encoder anymore.

The influence of encoding bitrate on frame rate is depicted in Figure 14. The results demonstrate the negative effect of high bitrate settings on the maximum achievable throughput, which drops from 185 Hz at low quality (4 Mbps @ 90 Hz) to 176 Hz at high quality (72 Mbps @ 90 Hz), and to 148 Hz at extreme quality (256 Mbps @ 90 Hz) for H.264. However, note that the extreme quality setting maximizes bandwidth utilization at approx. 46 MB/s, which coincides with the physical bandwidth limit measured in a separate network speed benchmark between Site B and Piz Daint. This finding is backed by the clamped nature of the high bitrate graph in comparison to the lower bitrate graphs, suggesting a theoretically greater achievable frame rate on high bandwidth connections. In practice, the particular bandwidth envelope at hand must be considered for stream design to prevent frame queueing effects from reducing interactivity.

In addition to the synthetic pre-rendered benchmarks, we have



(a) *Groundwater (Auto-Tuned Tiling)*



(b) *Strong Scaling*

Figure 15: *Left:* Groundwater *scene used in the OptiX-based pathtracer for the 4K strong scaling rendering experiments with auto-tuned distribution of tile sizes. Data courtesy of TACC/FIU. Right: Client-side frame rate achieved for strong scaling of path traced image at 4K resolution on up to 256 nodes using fixed tile distribution (blue) and auto-tuned tile sizes (orange).*

implemented a basic distributed path tracer, which builds on the path tracer sample from the OptiX SDK and uses screen space tiling in combination with replicated geometry. As a test scene we have used the *Groundwater* data set, which consists of approximately 8 million triangles. Previous benchmarks have used a regular tiling, but this tiling exhibits a strong load imbalance for the path tracing situation. We thus use an irregular tiling, an example of which is shown in Figure 15a, for benchmarks with the path tracer.

We have implemented simple two-dimensional recursive auto-tuning for this irregular tiling. The tiling is iteratively modified in a guided way until all tiles are subject to approx. the same rendering times. Starting with the regular tiling for a given number of tiles, e.g. 8x4 for 32 tiles, the algorithm works in two phases. First, only the vertical tiling is optimized by shifting the row heights until the differences between the rendering times of all rows are minimal. Once the optimal vertical tiling has been determined, the same procedure is applied to the horizontal tiling in each row recursively. In both dimensions, each iteration the shifting process determines the tile with maximum rendering time, decreases its size and increases the size of the fastest tile appropriately. After each iteration timings are re-evaluated and the tiling is shifted until convergence. The tiling of each iteration is checked against a recorded history for cycle detection. Convergence is then approximated by selecting the tiling configuration from the different tilings in the detected cycle with the minimum total rendering latency. An example auto-tuned tiling for 32 tiles is depicted in Figure 15a, demonstrating smaller tiles due to increased computational requirements for path tracing in the vicinity of the emissive streamlines going through the stone geometry.

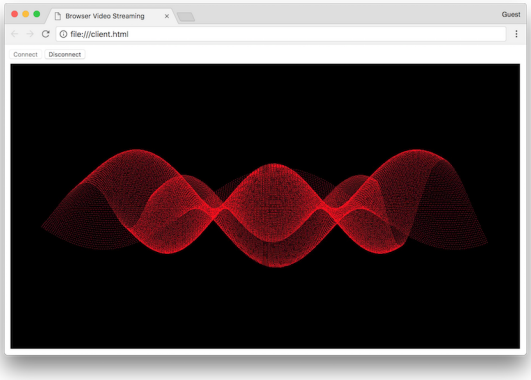


Figure 16: *Streaming unmodified CUDA* simpleGL *example from headless node to web browser using EGL-based shim library as GLUT replacement, WebSocket-based bidirectional communication (including interaction), on-the-fly MP4 wrapping of raw H.264 stream, and a JavaScript client.*

Figure 15b shows the achievable client-side frame rates when strong scaling the path tracer at 4K resolution up to 256 nodes streaming to Site A. Clearly, the auto-tuned tiling provides highly improved load balancing and thus reduced overall latencies in comparison to regular tiling. Starting with 0.5 Hz when streaming from a single node, the auto-tuned tiling strong-scales well: up to 14 Hz on 32 nodes, a scaling efficiency of almost 90%. Adding additional servers further increases the overall achievable frame rate up to 28 Hz on 256 nodes, yet with worse scaling efficiency. This can be explained by the high sensitivity of the approach towards load imbalances at high counts of tiny tiles, for which the path tracer in this scenario is a good example.

While effective load-balancing of distributed path tracers is out of the scope of this work, the presented path tracer benchmark shows how an otherwise slow-performing rendering application can easily be strong-scaled to interactive frame rates by using a remote HPC system to benefit from hardware-accelerated multi-tile streaming at low latencies.

5.4. Interoperability

Although demonstrated on vendor hardware, the presented approach is compatible with any hardware or software following the H.264/HEVC specification, thus inherently providing interoperability. For instance, for verification we have successfully mixed GPU and FFmpeg-based compressors, or have directly written a sequence of raw H.264 to a file for playback with VLC media player.

Motivated by direct (potentially hardware-accelerated) support for video decoding in modern web browsers, we have implemented a prototypical streaming implementation based on WebSockets and JavaScript through Media Source Extensions. Figure 16 shows the *simpleGL* CUDA example running on Piz Daint and streamed into the browser. Aiming for minimal changes at application-side, we have created an EGL-based shim library as dynamic replacement for GLUT, which is normally used by that demo for display and interaction. Our shim library internally creates an EGL offscreen buffer for rendering, starts a WebSocket server for bidirectional communication with the browser client (including mouse interaction events), and replaces the buffer swap by encoding and streaming. The latter involves an additional on-the-fly wrapping of the raw H.264 stream into the MP4 container format, as required by the browser. Since the container specification only allows constant frame rates (as common for standard video content), the streaming application must take care not to let the browser buffer frames, thereby delaying interaction events and reducing responsiveness. In our experiments we have specified the container frame rate to 30 Hz and adjusted the encoder to drop frames as nececessary to maintain a constant browser-side buffer underrun.

6. Conclusion

We have demonstrated how the specialized video encoding/decoding hardware on current and future generation GPUs can be harnessed for fast multi-tile streaming at low latency. This promising approach addresses modern challenges in remote HPC visualization, such as interactive workflows with complex rendering algorithms, supporting globally distributed user bases and driving latency-sensitive display technologies at high resolutions. Using hardware-accelerated streaming, traditional dedicated visualization clusters or workstations can be reduced to mere thin clients that leave the heavy lifting to the remote supercomputer.

Based on these encouraging results, we anticipate that many enhancements are possible and look forward to seeing multi-tile streaming as a technical foundation for future HPC visualization workflows. Similar to classic direct send, our approach will be limited in scalability at high tile counts. A hybrid tree-based/direct send compositing approach such as radix-k could alleviate this bottleneck. Hardware-accelerated sort-last compositing could be investigated based on depth buffer compression using video codecs. With high frame rates and low latencies indicating suitability for time-critical VR scenarios, the combined strong-scaled hardware power could be used for predictive rendering and streaming to remedy network latency.

Acknowledgements

Synthesis 4K footage by M. Starobin from NASA Scientific Visualization Studio. Groundwater simulation by S. Garcia and M. Sukop from Florida International University; and K. Cunningham from the United States Geological Survey.

References

[Adv] Advanced Media Framework. http://gpuopen.com/gaming-product/advanced-media-framework. Accessed: 2018-01-20. 2

[AGL05] AHRENS J., GEVECI B., LAW C.: ParaView: An End-User Tool for Large Data Visualization. In *The Visualization Handbook*, Hansen C. D., Johnson C. R., (Eds.). 2005. 2, 3

[CBW*12] CHILDS H., BRUGGER E., WHITLOCK B., MEREDITH J., AHERN S., PUGMIRE D., BIAGAS K., MILLER M., WEBER G. H., KRISHNAN H., FOGAL T., SANDERSON A., GARTH C., BETHEL E. W., CAMP D., RÜBEL O., DURANT M., FAVRE J., NAVRATIL P.: VisIt: An End-User Tool for Visualizing and Analyzing Very Large Data. In *High Performance Visualization—Enabling Extreme-Scale Scientific Insight*, Bethel E. W., Childs H., Hansen C., (Eds.), Chapman & Hall, CRC Computational Science. CRC Press/Francis–Taylor Group, Boca Raton, FL, USA, Nov. 2012, pp. 357–372. 2

[CMP14] CUI J., MA Z., POPESCU V.: Animated Depth Images for Interactive Remote Visualization of Time-Varying Data Sets. *IEEE Transactions on Visualization and Computer Graphics 20*, 11 (2014), 1474–1489. 2

[EMP09] EILEMANN S., MAKHINYA M., PAJAROLA R.: Equalizer: A Scalable Parallel Rendering Framework. *IEEE Transactions on Visualization and Computer Graphics 15*, 3 (May 2009), 436–452. 2

[EP07] EILEMANN S., PAJAROLA R.: Direct Send Compositing for Parallel Sort-last Rendering. In *Proceedings of the 7th Eurographics Conference on Parallel Graphics and Visualization* (2007), EGPGV '07, pp. 29–36. 2

[ESE00] ENGEL K., SOMMER O., ERTL T.: *A Framework for Interactive Hardware Accelerated Remote 3D-Visualization*. Springer Vienna, Vienna, 2000, pp. 167–177. 2

[FE10] FECHTELER P., EISERT P.: Accelerated video encoding using render context information. In *Proceedings of the International Conference on Image Processing, ICIP 2010, September 26-29, Hong Kong, China* (2010), IEEE, pp. 2033–2036. 2

[FNT*13] FEBRETTI A., NISHIMOTO A., THIGPEN T., TALANDIS J., LONG L., PIRTLE J. D., PETERKA T., VERLO A., BROWN M., PLEPYS D., SANDIN D., RENAMBOT L., JOHNSON A., LEIGH J.: CAVE2: a hybrid reality environment for immersive simulation and information analysis, 2013. 3

[HOPU08] HERELD M., OLSON E., PAPKA M. E., URAM T. D.: Streaming visualization for collaborative environments. *Journal of Physics: Conference Series 125*, 1 (2008). 2

[Int] Intel Media SDK. https://software.intel.com/en-us/media-sdk. Accessed: 2018-01-20. 2

[JFWM16] JIANG J., FOGAL T., WOOLLEY C., MESSMER P.: A Lightweight H.264-based Hardware Accelerated Image Compression Library. In *2016 IEEE 6th Symposium on Large Data Analysis and Visualization (LDAV)* (2016), pp. 99–100. 2

[KPH*10] KENDALL W., PETERKA T., HUANG J., SHEN H.-W., ROSS R.: Accelerating and Benchmarking Radix-k Image Compositing at Large Scale. In *Proceedings of the 10th Eurographics Conference on Parallel Graphics and Visualization* (2010), EG PGV'10, pp. 101–110. 2

[LMB*09] LALGUDI H. G., MARCELLIN M. W., BILGIN A., OH H., NADAR M. S.: View Compensated Compression of Volume Rendered Images for Remote Visualization. *IEEE Transactions on Image Processing 18*, 7 (2009), 1501–1511. 2

[LMM17] LEAF N., MILLER B., MA K. L.: In situ video encoding of floating-point volume data using special-purpose hardware for a posteriori rendering and analysis. In *2017 IEEE 7th Symposium on Large Data Analysis and Visualization (LDAV)* (2017), pp. 64–73. 2

[LS07] LAMBERTI F., SANNA A.: A Streaming-Based Solution for Remote Visualization of 3D Graphics on Mobile Devices. *IEEE Transactions on Visualization and Computer Graphics 13*, 2 (2007), 247–260. 2

[MAN*14] MARRINAN T., AURISANO J., NISHIMOTO A., BHARADWAJ K., MATEEVITSI V., RENAMBOT L., LONG L., JOHNSON A., LEIGH J.: SAGE2: A new approach for data intensive collaboration using Scalable Resolution Shared Displays. In *10th IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing* (Oct 2014), pp. 177–186. 2

[MKPH11] MORELAND K., KENDALL W., PETERKA T., HUANG J.: An image compositing solution at scale. In *2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC)* (2011), pp. 1–10. 2

[MPHK94] MA K.-L., PAINTER J. S., HANSEN C. D., KROGH M. F.: Parallel volume rendering using binary-swap compositing. *IEEE Computer Graphics and Applications 14*, 4 (1994), 59–68. 2

[NJ16] NOGUERA J. M., JIMÃL'NEZ J. R.: Mobile Volume Rendering: Past, Present and Future. *IEEE Transactions on Visualization and Computer Graphics 22*, 2 (Feb 2016), 1164–1178. 2

[Nvi] NVIDIA Video Codec SDK. https://developer.nvidia.com/nvidia-video-codec-sdk. Accessed: 2018-01-20. 2, 3

[PGR*09] PETERKA T., GOODELL D., ROSS R., SHEN H.-W., THAKUR R.: A Configurable Algorithm for Parallel Image-compositing Applications. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis* (2009), SC '09, pp. 4:1–4:10. 2

[SDWE03] STEGMAIER S., DIEPSTRATEN J., WEILER M., ERTL T.: Widening the remote visualization bottleneck. In *3rd International Symposium on Image and Signal Processing and Analysis, 2003. ISPA 2003. Proceedings of the* (Sept 2003), vol. 1, pp. 174–179 Vol.1. 2

[SML*03] STOMPEL A., MA K.-L., LUM E. B., AHRENS J., PATCHETT J.: SLIC: Scheduled Linear Image Compositing for Parallel Volume Rendering. In *Proc. of the 2003 IEEE Symposium on Parallel and Large-Data Visualization and Graphics* (2003), PVG '03, pp. 6–. 2

[SOHW12] SULLIVAN G. J., OHM J. R., HAN W. J., WIEGAND T.: Overview of the High Efficiency Video Coding (HEVC) Standard. *IEEE Transactions on Circuits and Systems for Video Technology 22*, 12 (2012), 1649–1668. 2

[WBSS04] WANG Z., BOVIK A. C., SHEIKH H. R., SIMONCELLI E. P.: Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing 13*, 4 (2004), 600–612. 5

[WSBL03] WIEGAND T., SULLIVAN G. J., BJONTEGAARD G., LUTHRA A.: Overview of the H.264/AVC video coding standard. *IEEE Transactions on Circuits and Systems for Video Technology 13*, 7 (2003), 560–576. 2

[YWM08] YU H., WANG C., MA K.-L.: Massively Parallel Volume Rendering Using 2-3 Swap Image Compositing. In *Proc. of the 2008 ACM/IEEE Conference on Supercomputing* (2008), SC '08, pp. 48:1–48:11. 2