Vision, Modeling, and Visualization (2020)
J. Krüger, M. Niessner, and J. Stückler (Eds.)

Real-time High-resolution Visualisation

F. Frieß[1], C. Müller[1] and T. Ertl[1]

[1]University of Stuttgart

Abstract

While visualisation often strives for abstraction, the interactive exploration of large scientific data sets like densely sampled 3D fields or massive particle data sets still benefits from rendering their graphical representation in large detail on high-resolution displays such as Powerwalls or tiled display walls driven by multiple GPUs or even GPU clusters. Such visualisation systems are typically rather unique in their setup of hardware and software which makes transferring a visualisation application from one high-resolution system to another one a complicated task. As more and more such visualisation systems get installed, collaboration becomes desirable in the sense of sharing such a visualisation running on one site in real time with another high-resolution display on a remote site while at the same time communicating via video and audio. Since typical video conference solutions or web-based collaboration tools often cannot deal with resolutions exceeding 4K, with stereo displays or with multi-GPU setups, we designed and implemented a new system based on state-of-the-art hardware and software technologies to transmit high-resolution visualisations including video and audio streams via the internet to remote large displays and back. Our system architecture is built on efficient capturing, encoding and transmission of pixel streams and thus supports a multitude of configurations combining audio and video streams in a generic approach.

CCS Concepts

• ***Human-centered computing*** → *Visualization systems and tools;* • ***Computing methodologies*** → *Graphics systems and interfaces;*

1. Introduction

Recently, we have all been using different streaming technologies for working remotely and entertainment. Video conferencing systems such as Skype, WebEx, Zoom, etc. support many users simultaneously while making compromises with respect to video and audio quality under heavy load. Streaming platforms like Netflix on the contrary can deliver 4K video resolution and high-quality audio, but only in one direction. Cloud gaming services deliver high-quality content and allow for low-latency interaction, but for specific applications and desktop resolution only. Scientific visualisation is one application domain in which users would clearly benefit from being able to collaborate remotely, combining video and audio conference setups with the possibility of sharing high-resolution interactive visualisations. However, for large – often tiled – displays and image resolutions beyond 4K no obvious generic, let alone commercial, solution exists. This drove us – motivated by the needs of a large collaborative research project in visual computing involving two universities about a two-hours drive apart – to develop a new system for that purpose based on latest the graphics and video technologies. Typically, there would have been lots of travel in the project for seminar talks, project demonstrations, etc., but since each site already had a large Powerwall-style visualisation system installed, it was obvious to use these for as many presentations as possible. Figure 1 illustrates the original usage scenario and the different hardware setups involved. We later added a low-latency solution to share the frames generated by any application on a large high-resolution display with another one, regardless of the different hardware setups and without changing the code of the application. This paper describes the design considerations, technological choices, details of the architecture and the implementation and results of performance testing.

These design considerations are mainly based on an observation from our target environment of two existing tiled displays: while most desktop computers – and in turn the visualisation software running on them – are alike, large high-resolution display setups are oftentimes unique in their own way. The reason for this stems from the fact that such systems are typically research prototypes in the first place, designed to answer questions like "How can I build a display exceeding one gigapixel?" [PPK13] or "How does stereo perception improve if I significantly decrease the size of a pixel?" [MRE13]. Even if the system design is based on application needs, the overall system design might greatly vary. For instance, a system designed to mainly run browser-based information visualisation applications is typically using as many GPUs with as many outputs as possible in a single machine such that the application can run just like on a desktop computer. In contrast, if the system is designed to visualise large and complex scientific data sets like particle data or flow fields, the system designers might want to add

DOI: 10.2312/vmv.20201195

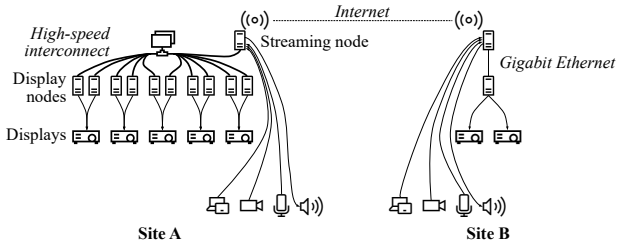

Figure 1: *Conceptual overview of the two Powerwalls for which our system was originally developed. The high-resolution display on the left-hand side uses a cluster of computers to drive its projectors. Like in a HPC cluster, the nodes are connected using a high-speed, low-latency InfiniBand network. In contrast, the display on the right-hand side uses a single computer with multiple outputs to drive the displays. In both cases, a dedicated streaming node is tasked with collecting and distributing all audio and video signals, although this role could be delegated to another node as well.*

as much GPU power and memory as possible, hence opting for a GPU cluster. Therefore, particularly if the goal is the interactive scientific visualisation of large data sets, unique software might be written for a unique display and compute system, which makes it difficult to fulfil the often-made promise for wall-sized displays: their suitability for collaborative work [JH14]. While all of the aforementioned diversity is irrelevant if a few people are working in front of a single display wall, it becomes a problem if users want to collaborate between two sites with different display walls: software might have to be adapted in code to run on the other site, or one of the systems might not have the computational power or graphics memory to run a 3D visualisation of a large simulation data set. Even if the computational resources are not the problem, it might be prohibitively time-consuming to transfer the whole simulation trajectory between the sites. However, regardless of the hardware and software infrastructure used to build a large high-resolution display, all of them share one property: they have a large (distributed) frame buffer to display coloured pixels. Therefore, exploiting advances in networking technology and the wide availability of fast, low-latency hardware video encoders, we wanted to revisit pixel streaming as a solution for sharing interactive visualisations in such environments while systematically using hardware video encoding and decoding – not least because it has even found its way into gaming, which notoriously craves for speed and low latency [SLNC13, CCT*11].

The overall performance of visual computing systems remains a bottleneck for many advanced visualisation techniques dealing with complex and large data and is therefore subject of ongoing research. Besides performance aspects of individual CPU/GPU nodes and how to optimise techniques for dedicated architectures, the methodology of measuring and reporting runtime performance has recently been investigated [BMFE20]. Multi-node systems and large displays add another level of complexity when investigating visual computing performance. The research presented in this paper was initially driven by the question whether streaming of Powerwall content with real-time interaction is at all possible based on contemporary hardware setups. The answer is: yes, a well-thought-out system design can satisfy the bandwidth and latency requirements, but only if the full potential of hardware support is exploited by low-level APIs. We claim that our collaborative high-resolution visualisation solution based on this system is currently the only one which can meet these demands. Our contribution includes the description of the system architecture and its major technical components, design considerations which we consider to be helpful for other high-performance visual computing systems involving streaming, and a quantitative evaluation of the system in real-world scenarios. The presented system also serves as a basis for ongoing research on adaptive techniques for further performance improvements like "adaptive video encoding" [FLB*18] or foveated multi-tile rendering.

2. Related work

Research for ways of supporting collaboration across multiple wall-sized displays lead to a multitude of systems, frameworks and libraries. The two, probably most widely known, collaboration environments for display clusters are the Scalable Adaptive Graphics Environment (SAGE) [RRS*04] and its successor, the Scalable Amplified Group Environment (SAGE2) [MAN*14, RMA*16]. While SAGE is based on pixel streams, it requires code changes to the applications in order to generate these streams. The newer SAGE2 is a browser-centric reimplementation around a central Node.js-based web server, which is responsible for distributing the shared content to different clients. This way, it naturally supports any client ranging from desktop computers over laptops to smartphones as long as a recent browser is available. In order to support high-resolution displays driven by graphics clusters, SAGE2 provides the means to synchronise the page redraw on connected browsers. Although this approach works for any device it does not offer the required performance to share the content of a large high-resolution display in real time. Other systems implementing some type of window management solution for tiled displays include the one of Klapperstueck et al. [KCG*18], which focusses on interactive collaboration with mobile devices and displays of various sizes. CubIT [Rit13] is a web-based like SAGE2 and builds its shared workspace across multi-touch screens, while DisplayCluster by Johnson et al. [JAW*12], and its successor Tide [Blu] specifically target large tiled displays driven by clusters. Their system also supports streaming content from remote sources and distributed playback of canned media files. LACOME [Liu07, MHB*12] aggregates multiple computer desktops on a shared large screen display by means of Virtual Network Compute (VNC) servers. Beck et al. [BKKF13] presented a multi-user immersive telepresence virtual reality system that allows users to meet in a shared virtual 3D world. It provides users with perspectively correct stereoscopic images and 3D information about the local workspace.

On a lower level, a variety of parallel rendering frameworks and middleware have been suggested to drive tiled displays. Equalizer [ESP18], a parallel OpenGL-based middleware, is one of them and provides a large set of features to support a wide variety of research and industry applications. For instance, OmegaLib [FNM*14] is based on Equalizer and provides tools to develop immersive 2D and 3D applications for systems ranging from large display

Figure 2: *The conference scenario with a talk given at site A. The tiled display shows the speaker's slides captured from the laptop on the left and the 4K video stream from site B on the right. Site B receives the slides and a 4K video stream from site A, while communication between the sites is established by bidirectional audio streams.*

Figure 3: *An example of the screen sharing scenario as displayed on the tiled display on site A: site B is sharing the content of their large high-resolution display (seen on the left). As the aspect ratio is wider on site A, the 4K video stream of the presenter (seen on the right) can be shown beside the shared visualisation. Bidirectional audio streams allow users to talk with each other while investigating the visualisation shared.*

walls to CAVEs. The Cross Platform Cluster Graphics Library (CGLX) [DK11, PDW*11] is another OpenGL-based framework for distributed high-performance visualisations. It allows users to adapt existing or develop new OpenGL-based applications for clusters such as tiled displays. Additionally, CGLX supports co-located collaboration through multiple multi-touch devices to which the updated scene information is streamed.

Streaming solutions gained some popularity in recent years, like the one of Biedert et al. [BMFG18], which uses video compression to achieve high frame rates. This solution is similar to the system we propose for screen sharing, but it requires the same hardware configuration for the source and destination display, whereas our system is able to work with different configurations. Marrinan et al. [MRI*19] showed a system that allows streaming high-resolution content, without using any compression, in real-time. Their solution allows for parallel processing of rendering and transmitting a frame via TCP to one or more clients. Although the system is able to reach high frame rates without dropping frames, it needs a connection with a very high bandwidth to do so. Furthermore, neither of the solution includes audio streams.

3. System architecture

The architecture and implementation of our system is defined by three main goals we wanted to achieve: first and foremost, the software should be based on moving pixels around while leveraging any hardware support available, because we considered this the only way to achieve application-agnostic interactivity even on tiled large high-resolution displays. Second, we wanted the system to support several real-world scenarios, namely using the system for on-site and remote presentations in the way a standard projector would work (Figure 2), sharing the full-resolution tiled display screen between remote sites in an application-agnostic way (Figure 3) and combining both scenarios with video and audio streams for a video conference setup. Third, the system should run in different kinds of tiled display environments, namely the ones using separate cluster nodes to drive the tiles as well as the ones using as many GPUs as possible in a single machine to drive the whole display with one computer. In the scenario shown in Figure 1, the tiled display at site A has a resolution of $10,800 \times 4096$ pixels, which are composed from ten projectors (five for each stereo channel) driven by a cluster of 20 nodes. The display at site B comprises 5224×2160 pixels from two projectors driven by a single node. At both sites, we added a dedicated computer to handle all the video and audio devices, the streaming node, which also has the required internet visibility to serve streams to the other site and to RTSP clients. As detailed in section 4, these tasks could also be handled by other machines, but the dedicated node greatly facilitates the physical setup and the configuration of the network.

In the following, we describe – by following audio and video samples from obtaining the original input to audio and video rendering – how we designed our system to achieve the aforementioned goals.

Input streams: The goal of streaming live videos at high resolutions and exposing the tiled displays as standard projection devices to end users bringing their own laptops mandates the use of hardware frame grabbers. Such devices like BlackMagic Design's DeckLink family or the Unigraf UFG family – the two device families we use in our setup – have significantly dropped in price over the recent years and are a simple solution for achieving this goal. The use of Windows Media Foundation, which is typically supported by all capturing cards on Windows platforms, enabled us to write software that is agnostic to hardware vendors and custom SDKs when it comes to obtaining video signals. Media Foundation is the sole source of external video signals from cameras and frame grabbers for capturing the screen of a laptop. However, there are additional sources outside its scope, which are audio samples from the low-latency ASIO (Audio Stream Input/Output) API, video samples from the Desktop Duplication API [Mic18] for application-agnostic transmission of the tiled desktop, and network streams originating from the remote site.

Encoding: Video and audio streams transmitted over networks need to be encoded to match the available bandwidth of the network. Media Foundation provides a variety of out-of-the-box en-

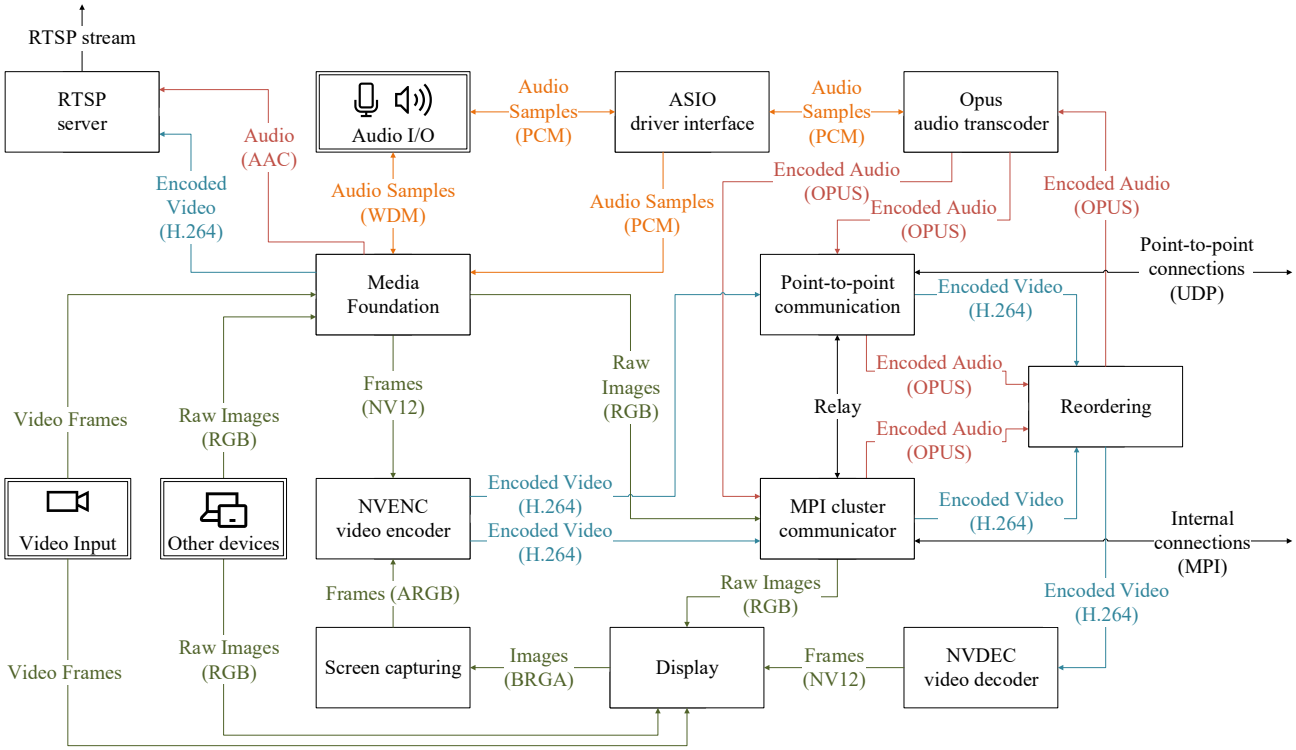

Figure 4: *Overview of the components of the system and data flow between them. The diagram shows all possible components that can be configured differently on each node. The orange connections represent raw audio samples whereas encoded audio samples are represented by red connections. Raw video frames are shown in green and encoded video frames are azure. Black-coloured connections represent any of the above, i. e. data-agnostic communication links.*

coders, but many of them are software-based or can use hardware encoders only if certain preconditions are met. The point-to-point communication between the two tiled display sites uses Opus audio coding [Int11] and the NVENC hardware encoder [NVIb] for the video, available on all reasonably new NVIDIA graphics cards. We eventually opted for implementing both, the video and the audio path, outside Media Foundation as wrapping this functionality in Media Foundation transform would have required additional coding effort while adding unnecessary latency at runtime.

Network transfer: The network technologies used in our system fall into two classes: site-local communication between computers used to drive a single tiled display and internet communication to transmit audio and video data between the sites. The former is implemented using the Message Passing Interface (MPI), which is the de-facto standard for communication in HPC clusters. Besides providing a portable API, MPI adds the additional advantage of having specialised implementations for high-speed network technologies like InfiniBand (IB) that talk almost directly to the hardware. For instance, InfiniBand implementations of MPI use IB verbs bypassing the whole TCP/IP stack (in contrast to IP over IB), thus greatly reducing network latency and increasing bandwidth. On systems where no such implementation is available, MPI can be used on top of TCP/IP and standard Ethernet connections. Ethernet and IP are also the means for connecting the two remote sites. As using an out-of-the-box RTSP implementation proved to add too much latency for video conferences, we opted for implementing our own, UDP-based protocol, which resembles the packet format of the Real-Time Transport Protocol (RTP), but strips off the session management part of RTSP.

Rendering: Since the introduction of the Windows Display Driver Model (WDDM), most of the graphics in Windows is based on the DirectX Graphics Infrastructure (DXGI). Therefore, resources like the Windows desktop and GPU-based Media Foundation samples, are just Direct3D textures at their core. The coherent and seamless way of using them with Direct3D as rendering API is one of the reasons we opted for Direct3D 11 as rendering API. Furthermore, Direct3D has a long history of being very explicit when it comes to exposing the display topology to the programmer. The fact that we can explicitly address each GPU and their outputs is a valuable feature when implementing software for tiled displays that typically include multiple GPU and display scenarios.

Audio rendering can be handled by the streaming audio renderer of Windows Media Foundation or using ASIO, which has been designed for minimal latency. Furthermore, depending on the un-

derlying hardware, ASIO allows for addressing a large number of audio channels individually, which can be used for positional audio in an immersive display installation.

Configuration: The variety of different scenarios is currently managed using a declarative approach based on XML. All computers involved in a scenario – display nodes connected to the output as well as dedicated streaming nodes – are configured in the same file and identified by pertinent strings like the host name, an IP address or one of the MAC addresses of the computer. Likewise, input and output devices are identified by their hardware IDs and network streams by their URL. All of these resources can be connected in a graph-like manner, which allows the system to build the corresponding in-memory representation at startup. Furthermore, the configuration file also allows for configuring most components, e. g. the quality settings of a video encoder.

4. Implementation details

Our system is implemented in C++ and aims for efficiency. For instance, we try to use zero-copy implementations on the CPU and to avoid memory allocations by using pools of frequently used objects. Naturally, we avoid moving data between CPU and GPU and pass around texture handles instead or use GPU-to-GPU copies if necessary. In the following, we describe how all of this is integrated into the overall system depicted in Figure 4 and the interplay between the components of the system.

Media Foundation: Besides as source of external video data, we use Windows Media Foundation for format and colour conversions and resampling. At the heart of Windows Media Foundation is a data flow graph (topology), which connects media sources, sinks and optional transforms. As our software starts, the user-defined application scenario is translated in such a topology. For some features of our software, like the support of ASIO, we had to extend Media Foundation by custom types of nodes. Many of them are so-called sample grabbers, which free the programmer of the burden of implementing the complicated control flow of generic sinks. From a programmer's point of view, a sample grabber simply implements a push-based data flow where samples arrive as they are ready. The most important grabbers we implemented extract video samples for encoding them using NVENC and the audio and video sinks of the RTSP server, which receive encoded H.264 and AAC samples.

ASIO: Steinberg developed ASIO specifically with low-latency in mind, which constitutes a more than welcome side effect of the fact that we had to support ASIO as the audio technology at one of the test sites is based on ASIO. Technically, ASIO devices are addressed by copying raw audio samples to and from its input and output buffers at regular time intervals. A custom live source is used to feed audio samples from ASIO devices into the Media Foundation topology.

Screen sharing: Starting with Windows 8, the Desktop Duplication API provides fast access to the full desktop image as a texture. For our screen sharing implementation, we feed this texture into the NVENC encoder. As NVENC expects a different input than the native desktop format and has, depending on the underlying hardware, limitations for the size of the frames that can be encoded, the texture needs to be converted. Using compute shaders for any conversion avoids transferring the image between GPU and main memory. If the desktop needs to be tiled, we try to assign a dedicated NVENC session to each of them in order to minimise latency.

Encoding: We opted for a lock-free asynchronous approach using ring buffers to decouple video and audio sources from encoding (and decoding) happening asynchronously in separate threads or in hardware. For frame sources like sample grabbers or network threads, encoding is a fire-and-forget operation. We use the Opus codec for audio streams, which has been derived from the open-source format CELT and Skype's SILK. It is an open standard specifically designed for real-time communication over the internet. Its use is straightforward as the programmer just copies the interleaved PCM16 samples into an input buffer and receives the encoded byte stream. Video streams are encoded in H.264 via the NVIDIA Video Codec SDK (NVENC), which provides access to the hardware encoders used e. g. by NVIDIA's own GeForce Now game streaming service. For our application, an important advantage besides speed is the possibility to encode frames without the need of piping data that is already on the GPU through main memory. Once the frame is encoded, it is downloaded, sliced to fit the MTU size of the underlying network and is sent to the consumers via point-to-point connections.

MPI cluster communicator: Communication between nodes at one site is implemented using MPI. While the advantages of this approach have already been outlined, this choice also comes with the drawback of many MPI implementations, although offering non-blocking calls, require all API calls being made from a single thread. However, our producers are all asynchronous, mandating all data being funnelled through another message queue before being transmitted to local nodes. Furthermore, as the communication patterns in the local cluster depend on the user-provided configuration of streams, the implementation of that patterns must ensure that user input cannot result in deadlocks due to mutually pending unicast operations. We therefore minimise the number of communication calls by coalescing all data packets queued during rendering a frame into one chunk for each stream type and send it at once via MPI, which has the added benefit of increasing the network throughput. Additionally, all nodes send these chunks once per frame using a non-blocking operation to all other nodes that subscribed to the respective streams. Afterwards, non-blocking receive calls are made, after which all send and receive operations are awaited to safely recycle the memory used for the transfer.

Point-to-point communication Communication between sites is done using UDP to achieve the best possible saturation of the network and to avoid latency introduced by TCP. Technically, all send and receive operations are processed asynchronously using an I/O completion port, which is a queue for completion events of asynchronous I/O operations managed by the operating system. A thread pool sized according to the native parallelism of the underlying hardware is used to process the completion events. This way, all internet communication is implemented at one place, allowing to control the life time of the memory used for data transfer. This memory is pooled in chucks of the largest packet the system has seen, which avoids costly heap allocations in every frame. As the UDP datagrams containing frames might arrive out of order on the

receiver side, which is problematic for most decoders, we use a ring buffer to reorder packets and drop packets that are late altogether.

Decoding: The video decoder (NVDEC) is initialised on the fly, since it needs the settings from the encoder, which clients receive when connecting to a remote stream. It works, in a similar fashion to the encoder, on its own internal ring buffer on the GPU. The encoded frames are queued to NVDEC, which triggers an asynchronous chain of callbacks. In the last callback, the decoded frame is copied into a separate ring buffer, consisting of Direct3D interop textures, used to hold the frames until they are displayed.

The Opus decoder works similarly to the Opus encoder and provides interleaved PCM16 samples for chunks of the encoded byte stream. They need to be deinterleaved and changed to the correct PCM format before being queued for playback by the ASIO device, which happens in an asynchronous continuation function.

Display: Rendering using Direct3D is more or less straightforward drawing the video samples as textured quads. All format conversions, scaling and positioning transformations are performed on the fly on the GPU, which includes clipping the video to match the segment of the tiled display. If a video stream is displayed across multiple nodes, each node has to show the same frame at the same time in order to avoid tearing artefacts. When supported by the hardware, we use synchronisation based on NVIDIA's Quadro Sync technology [NVIa]. As a fallback, we offer synchronising all nodes before the buffer swap using a standard MPI barrier.

RTSP: For scenarios in which a person is giving a talk at one site which is transmitted to the other one, we found it desirable to enable external viewers to connect to the stream. However, these clients have no large high-resolution display attached and we wanted users to be able to connect with standard technologies like RTSP. We therefore integrated an implementation based on live555 [LN]. Each node within the cluster can theoretically be configured as the RTSP server, but this role is typically held by the dedicated streaming node (cf. Figure 1). Encoded audio and video data for the RTSP stream as well as synchronisation of audio and video samples are provided by Windows Media Foundation, for which we provided a stream sink which serves as adapter to live555.

5. Results

We performed a quantitative evaluation of our system in the setup shown in Figure 1, where the two sites are approximately 180 km apart from each other. Additionally, we tested 8K streaming in a local area network setting on site A. Every 2000 measurements, we computed the minimum, maximum, average and median duration and the median absolute deviation for each operation from these measurements. We used the algorithm by Cristian [Cri89] to synchronise the clocks of the nodes to an external time source.

Hardware details: The display nodes at site A, of which we only use ten for non-stereo content, are each equipped with an NVIDIA Quadro M6000, two Xeon E5-2640v3 CPUs and 256 GB of RAM. Each of them is responsible for a fraction of 1200×4096 pixels of the overall screen. The display node at site B, which is responsible for the whole 5224×2160 pixels, uses the same GPU, but has two Xeon E5-2600v3 CPUs and 512 GB of RAM. The streaming node at each side is connected to external audio devices and equipped with an HDMI and and SDI frame grabber card. Both use 64 GB of RAM while the node at site A comprises an Intel Core i7-6850K and a GeForce GTX 1060, the node at site B is equipped with a Core i7-5930K and a GeForce GTX 960. At site A, the streaming node is part of the low-latency InfiniBand network and is connected to the internet with a 10 Gb network adapter. Site B uses Gigabit Ethernet for all connections including the internet uplink. For the 8K screen sharing scenario, we used an Intel Core i9-9900K, 64 GB of RAM and a Titan RTX to drive a display with a resolution of 7680×4320 px.This machine was located in the same building as the Powerwall of site A, which provides sufficient resolution to display the full 8K desktop.

Table 1: *Overview of the measured latency, in milliseconds, for each encoder setting used in the screen sharing scenario. The Complete (Source) column shows the complete duration from capturing the frame to sending it. Likewise, the Complete (Display) column shows the complete duration from receiving the encoded frame to displaying it. The colour coding indicates the severity of the impact of each step on the overall latency of the scenario (blue representing low, white medium and red high impact).*

Encoder setting		Convert	Encode	Complete (source)	Decode	Display queue	Complete (display)
low	avg	1.17	6.62	7.76	0.07	0.47	5.82
	min	0.59	5.95	7.04	0.05	0.29	5.55
	max	3.58	12.45	13.83	0.16	0.90	6.35
	med	1.16	6.08	7.25	0.06	0.52	5.85
	mad	0.02	0.01	0.06	0.01	0.10	0.10
medium	avg	1.20	7.02	8.21	0.15	0.50	6.32
	min	0.45	5.94	7.02	0.04	0.28	5.58
	max	3.34	12.46	13.86	0.43	1.03	11.01
	med	1.18	6.07	7.25	0.06	0.54	5.93
	mad	0.02	0.09	0.15	0.01	0.11	0.27
high	avg	1.26	11.93	14.82	0.55	0.51	6.90
	min	0.59	5.96	7.07	0.04	0.29	5.63
	max	3.61	22.33	26.43	1.99	1.70	13.01
	med	1.17	6.69	7.71	0.10	0.53	6.00
	mad	0.01	0.68	0.56	0.05	0.11	0.28

Test scenarios: In the first test scenario, we shared the full tiled display at site A in the local network while the trajectory of a molecular dynamics simulation was shown twice for 100 s. This scenario provides a controlled network environment and it does not involve any video scaling as it uses the left and the right stereo channel of the tiled display, thus allowing for visual inspection of latency. In order to evaluate our solution for the limited resolution supported by some hardware encoders, we transmitted the 8K frame buffer of a single machine to the tiled display while showing the same simulation in a second test. Finally, we tested the video conference scenario by transmitting full HD presentation slides and a 4K video from site A to site B.

Test results: Table 1 shows the time, in milliseconds, required

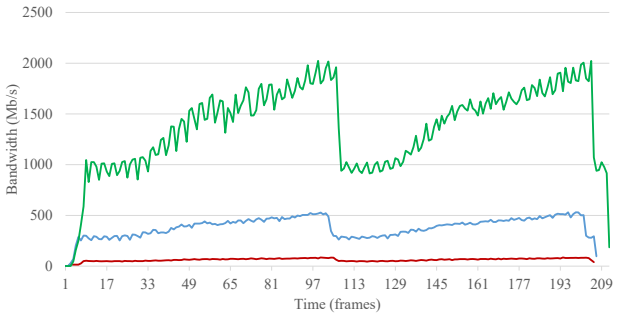

Figure 5: *Bandwidth aggregated over all nodes in the screen sharing scenario for each of the three quality settings: low (red), medium (blue), high (green). The maximum bandwidth required for the settings are: 85.3 Mb/s, 530.6 Mb/s and 2023.1 Mb/s.*

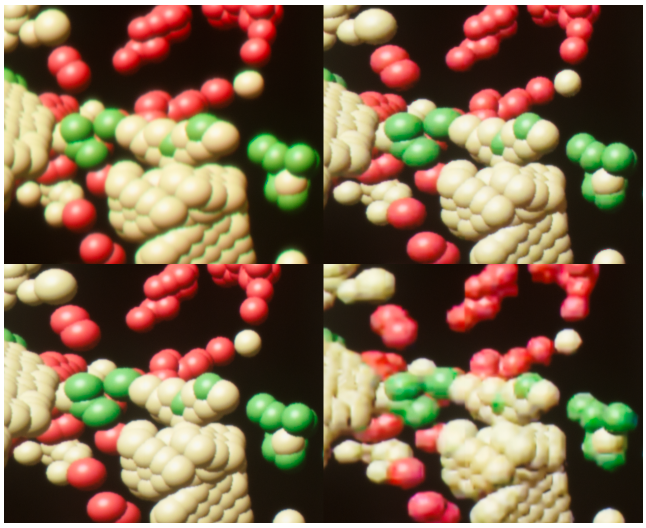

Figure 6: *There is no visible visible difference between the original (top left) the medium encoder setting (top right) and the high encoder setting (bottom left). Only the low encoder setting (bottom right) produces visible artefacts.*

for each step of the controlled, local screen sharing scenario. Each column states the values of the worst case measured across all nodes, with the variance between the nodes is at most 2 ms. The latency introduced by the MPI communication, the reordering of network messages and the assembly of the encoded frames is well below 1 ms and therefore omitted from the table. However, all operations and communication are included in the *Complete (source)* column and the *Complete (display)* column. As can be seen, the encoding introduces the highest latency with up to 22 ms, for key frames. The minimum and median values show the latency of the intra frames and they are between 6 ms and 7 ms. The highest contribution, with more than 5 ms, on the decoder site comes from mapping the decoded frame after NVDEC has finished.

The 8K screen sharing scenario is specifically designed to test the in-memory slicing and scaling of the overall desktop required as Quadro M6000 cannot decode an 8K stream at once. Additionally, the Titan RTX has a maximum of two NVENC sessions that can run in parallel. As we need to split the 8K screen into four tiles, each session needs to encode two tiles, therefore producing solely key frames. This increases the encoding latency to 25 ms and the bandwidth by a factor of 2.4.

We also measured the required bandwidth for the three encoder settings low, medium and high. Bandwidth was measured as the sum of all outgoing data per second and is shown in Figure 5. The big drop in the middle of the graph is the time when simulation trajectory starts again. With 530 Mb/s for the medium setting, the required bandwidth is well below the limit of 1 Gb/s for a reasonable image quality. Using a reasonable compromise for the encoder settings, the required bandwidth for the transmission of the 4K camera in our scenario is about 50 Mb/s. The transmission of the presentation slides makes use of equal frame detection and thus transmits only new slides to the clients. The maximum bandwidth measured for the slides was around 70 Mb/s while showing an embedded video. Displaying the full HD slides at site A consumed around 7 Gb/s on the IB network. We chose to accept this high bandwidth, because the network can handle it and the median display latency thus lies around 5 ms. Audio samples could be encoded in at most 1 ms (0.2 ms on average) and decoded in less than 1 ms (0.09 ms on average). The end-to-end transmission of audio samples took at most 2 ms in our arguably very good network situation, way below the ITU-recommended maximum of 150 ms for voice communication.

6. Discussion

Our system has been specifically designed to move lots of pixels efficiently, which we found to be a viable solution when exploiting the potential of state-of-the-art hardware. We also deliberately chose to base sharing of external screens on signal capturing cards, since such devices have become affordable, and because this approach leads to a unified software design. Additionally, this solution offers the best user experience resembling a standard projector.

Since the maximum required bandwidth for reasonable encoder settings is well below the limit of 1 Gb/s, our system can be used in a realistic setting. The screen sharing of $10,800 \times 4096$ pixels at 60 frames per second is possible even at a lower available bandwidth. The low-quality encoder setting yields visible artefacts, which are noticeable while standing close to the large display (see Figure 6). The medium encoder setting produced an image very close to the original, such that the difference is hardly visible, while still using a realistically available bandwidth. With the high-quality setting no difference is noticeable, but the required bandwidth exceeds what is typically available over the internet. There are multiple options to address connections that have a very small available bandwidth. Firstly the encoder can be configured to use a maximum bitrate instead of the constant quality settings (low, medium and high) we used for our tests. This reduces the required bandwidth to the specified bitrate but it reduces the quality of the encoded video stream.

Secondly the already implemented downscaling can be used to reduce the size of the content. Again this reduces the bandwidth while reducing the quality. Thirdly the equal frame detection can also be used for all video streams. This might only have a small impact on the required bandwidth but it does not reduce the quality. And lastly other bandwidth reduction methods like [FLB*18] can be used. A Combination of these options is also possible.

Latency for all screen sharing scenarios was hardly noticeable in side-to-side comparison, with at most 30 ms for the whole pipeline. The major contribution to the latency, with up to 22 ms for the high settings, on the source side comes from the NVENC, which we attribute to the encoding of key frames. For intra frames, the latency was less than half and for the medium and low settings the key frames where also encoded in less than 13 ms.

While developing our system, we naturally explored some solutions that did not turn out to be viable. Most notably, the original plan of using RTSP streaming for the video conference part turned out to never achieve latencies below even 500 ms, which is too much for two parties to talk naturally. Our custom UDP-based network layer solves this problem, obviously at the cost of interoperability. Furthermore, we found that software-based video encoders provided by Windows Media Foundation are barely capable of handling full HD streams on nodes handling more than one role or multiple streams. We therefore conclude, that video conferencing streams and external sources with 4K resolution in real time only work thanks to a software design that systematically makes use of modern graphics hardware and zero-copy operations whenever possible. One drawback of the hardware solution is, however, the variation of hardware support. Although we already restrict ourselves to NVIDIA's cards, there is still a great variety of encoder and decoder chips, of the number of parallel sessions allowed and of supported codecs, which required us to provide workarounds for cases when the cards involved are not on the same feature level.

As in any major software system, we also ended up with solutions that work, but that one would do differently now. Most importantly, the use of Windows Media Foundation did not solve as many problems as we had initially hoped. The reasons for this are manifold, ranging from the fact that the built-in encoder transform nodes in Media Foundation where not fast enough, over the need to support existing ASIO hardware to the fact that our frame grabbers only use formats that are supported out-of-the-box if running with a resolution less than 4K. All of this led to an implementation that combines several implementation islands which are more or less tied into the initial Media Foundation core of the software. For instance, the ASIO adapter is closely integrated as a Media Foundation source, whereas the video encoder is an independent low-level NVENC implementation, which is a daunting endeavour by itself. Likewise, the Opus-based audio path is not at all integrated with Media Foundation, which negates the benefit of Media Foundation synchronising the timestamps of all media sources involved. In hindsight, it would have been reasonable to either "go full Media Foundation" and implement everything as reusable media source, sink or transform, or to to extract all information using sample grabbers at the earliest possible convenience and base all processing on a own implementation. This way, we could also have prevented the presence of different sources of parallelism ranging from Media Foundation queues, the thread pool of the IOCP over the live555 worker to threads dedicated to the Opus transcoder.

Our setup uses a dedicated node for capture, audio playback and for transmitting and receiving streams. As incoming video streams are relayed to the display nodes via MPI, we initially expected the need to delay audio playback to be synchronous with the video stream. However, this turned out to be unnecessary for the current setup. Likewise, many low-level optimisations we made are probably unnecessary as well. As an example, we initially used lock-free means of moving data between video-related threads via a triple buffer to avoid the cost of system-level synchronisation primitives. This turned out to be impractical as it does not enforce that all nodes in the cluster actually draw the same frame, even if the buffer swap is synchronised. We therefore had to replace that implementation with a more conventional one using mutexes, which in the end had no noticeable impact on the performance of the system.

The Powerwalls we developed the system for are running on Microsoft Windows, making it a natural choice to leverage its coherent media processing capabilities. However, the building blocks for our software, like NVENC and capturing SDKs, are also available on Linux, making the concept transferable to this system. For compatible hardware, the NVIDIA Capture SDK provides equivalent capabilities to the Desktop Duplication API. Developing a system with similar capabilities is therefore a mere implementation effort.

7. Conclusion and future work

We presented a system for different remote collaboration scenarios on large high-resolution displays based on pixel streaming using modern graphics hardware, which grew out of the need to actually support these scenarios in a research project in visual computing. Our system supports collaboration across sites by sharing the content of a large display to others at its native resolution in real time. This is application-agnostic, i. e. works for any visualisation that can be displayed on the respective tiled display without changing the code of the application. In addition to this scenario, the system supports real-time video conferences and multi-site talks that complement the video stream with a stream of the output of a laptop. Key to our solution is the multi-threaded and hardware-centric design of the software, which makes our solution viable even as one of the sites has only Gigabit Ethernet connectivity.

In the future, we plan to make the software more dynamic such that sources and streams can be added or removed while the programme is running. Such an improvement would ideally be complemented by a comprehensive user interface for configuring the system. Another direction for improvements is the integration of technologies like WebRTC, which would enable users that are currently mere spectators of the RTSP stream to engage in discussions. Additionally we plan to investigate foveated encoding to reduce the required bandwidth while keeping the quality high.

Acknowledgments

This work was partially funded by Deutsche Forschungsgemeinschaft (DFG) as part of SFB-TRR 161 (Project ID 251654672). The authors want to thank Timon Kilian for supporting the experiments performed for this work at site B.

References

[BKKF13] BECK S., KUNERT A., KULIK A., FROEHLICH B.: Immersive group-to-group telepresence. *IEEE Trans. Visual Comput. Graphics 19*, 4 (2013), 616–625. 2

[Blu] BLUE BRAIN PROJECT: Tide: Tiled Interactive Display Environment. Online, last accessed 02/09/2020. https://github.com/BlueBrain/Tide. 2

[BMFE20] BRUDER V., MÜLLER C., FREY S., ERTL T.: On evaluating runtime performance of interactive visualizations. *IEEE Transactions on Visualization and Computer Graphics* (Early Access 2020). URL: https://ieeexplore.ieee.org/document/8637795, doi:10.1109/TVCG.2019.2898435. 2

[BMFG18] BIEDERT T., MESSMER P., FOGAL T., GARTH C.: Hardware-accelerated multi-tile streaming for realtime remote visualization. In *Proc. EGPGV* (2018), pp. 33–43. 3

[CCT*11] CHEN K.-T., CHANG Y.-C., TSENG P.-H., HUANG C.-Y., LEI C.-L.: Measuring the latency of cloud gaming systems. In *Proc. ACM Int. Conf. on Multimedia* (2011), pp. 1269–1272. 2

[Cri89] CRISTIAN F.: Probabilistic clock synchronization. *Distrib. Comput. 3*, 3 (Sept. 1989), 146–158. 6

[DK11] DOERR K. U., KUESTER F.: CGLX: A scalable, high-performance visualization framework for networked display environments. *IEEE Trans. Visual Comput. Graphics 17*, 3 (2011), 320–332. 3

[ESP18] EILEMANN S., STEINER D., PAJAROLA R.: Equalizer 2.0 – convergence of a parallel rendering framework. *IEEE Trans. Visual Comput. Graphics* (2018), 1292–1307. 2

[FLB*18] FRIESS F., LANDWEHR M., BRUDER V., FREY S., ERTL T.: Adaptive encoder settings for interactive remote visualisation on high-resolution displays. In *Proc. LDAV* (2018). 2, 8

[FNM*14] FEBRETTI A., NISHIMOTO A., MATEEVITSI V., RENAMBOT L., JOHNSON A., LEIGH J.: Omegalib: A multi-view application framework for hybrid reality display environments. In *Proc. IEEE VR* (2014), pp. 9–14. 2

[Int11] INTERNET ENGINEERING TASK FORCE: Opus Interactive Audio Codec. Online, last accessed 02/09/2020, 2011. https://opus-codec.org/. 4

[JAW*12] JOHNSON G. P., ABRAM G. D., WESTING B., NAVRÁTIL P., GAITHER K.: DisplayCluster: An interactive visualization environment for tiled displays. *Proc. IEEE Int. Conf. on Cluster Computing* (2012), 239–247. 2

[JH14] JAKOBSEN M. R., HORNBÆK K.: Up close and personal: Collaborative work on a high-resolution multitouch wall display. *ACM Trans. Comput.-Hum. Interact. 21*, 2 (2014), 11:1–11:34. 2

[KCG*18] KLAPPERSTUECK M., CZAUDERNA T., GONCU C., GLOWACKI J., DWYER T., SCHREIBER F., MARRIOTT K.: Contextuwall: Multi-site collaboration using display walls. *J. Visual Languages and Computing 46* (2018), 35–42. 2

[Liu07] LIU Z.: *Lacome: a cross-platform multi-user collaboration system for a shared large display*. PhD thesis, 2007. 2

[LN] LIVE NETWORKS I.: live555. Online, last accessed 02/09/2020. www.live555.com. 6

[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 *Proc. IEEE Int. Conf. on Collaborative Computing: Networking, Applications and Worksharing* (2014), pp. 177–186. 2

[MHB*12] MACKENZIE R., HAWKEY K., BOOTH K. S., LIU Z., PERSWAIN P., DHILLON S. S.: Lacome: A multi-user collaboration system for shared large displays. In *Proc. ACM Conf. on Computer Supported Cooperative Work Companion* (2012), pp. 267–268. 2

[Mic18] MICROSOFT: Desktop Duplication API. Online, last accessed 02/09/2020, 2018. https://docs.microsoft.com/en-us/windows/desktop/direct3ddxgi/desktop-dup-api. 3

[MRE13] MÜLLER C., REINA G., ERTL T.: The VVand: A two-tier system design for high-resolution stereo rendering. In *CHI POWERWALL 2013 Workshop* (2013). 1

[MRI*19] MARRINAN T., RIZZI S., INSLEY J. A., LONG L., RENAMBOT L., PAPKA M. E.: Pxstream: Remote visualization for distributed rendering frameworks. In *Proc. LDAV* (2019), pp. 37–41. 3

[NVIa] NVIDIA: NVIDIA Quadro Sync. Online, last accessed 02/09/2020. https://www.nvidia.com/en-us/design-visualization/solutions/quadro-sync. 6

[NVIb] NVIDIA: NVIDIA Video Codec SDK. Online, last accessed 02/09/2020. https://developer.nvidia.com/nvidia-video-codec-sdk. 4

[PDW*11] PONTO K., DOERR K., WYPYCH T., KOOKER J., KUESTER F.: CGLXTouch: A multi-user multi-touch approach for ultra-high-resolution collaborative workspaces. *Future Gener. Comput. Syst. 27*, 6 (2011), 649–656. 3

[PPK13] PAPADOPOULOS C., PETKOV K., KAUFMAN A.: Building the Reality Deck. In *CHI POWERWALL 2013 Workshop* (2013). 1

[Rit13] RITTENBRUCH M.: CubIT: Large-scale Multi-user Presentation and Collaboration. *Proc. Int. Conf. on Interactive Tabletops and Surfaces* (2013), 441–444. 2

[RMA*16] RENAMBOT L., MARRINAN T., AURISANO J., NISHIMOTO A., MATEEVITSI V., BHARADWAJ K., LONG L., JOHNSON A., BROWN M., LEIGH J.: SAGE2: A collaboration portal for scalable resolution displays. *Future Gener. Comput. Syst. 54* (2016), 296–305. 2

[RRS*04] RENAMBOT L., RAO A., SINGH R., JEONG B., KRISHNAPRASAD N., VISHWANATH V., CHANDRASEKHAR V., SCHWARZ N., SPALE A., ZHANG C., GOLDMAN G., LEIGH J., JOHNSON A.: SAGE: the Scalable Adaptive Graphics Environment. *Proc. WACE* (2004), 2004–2009. 2

[SLNC13] SHEA R., LIU J., NGAI E. C.-H., CUI Y.: Cloud gaming: architecture and performance. *IEEE Network 27*, 4 (2013), 16–21. 2